Technologies pour les applications en réseau : contribution au profil NetDevOps

RSX102

Document provisoire.
Copie et diffusion non autorisées sans accord écrit.
Documents liés aux cours, exercices et TP : https://rsx102.seancetenante.com

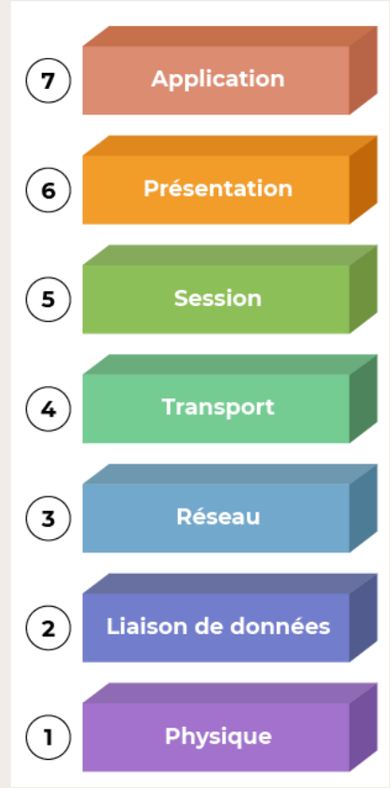
2026 François Lacomme <francois.lacomme@2isa.net>

Présentation de l'UE

Contenu et organisation

- **Contenu**
  - Cette unité d'enseignement, RSX102, **Technologies pour les applications en réseau**, (Technologies pour les applications client-serveur jusqu'en 2020), concerne donc :
  - Les couches hautes du modèle OSI (Open System Interconnection)
  - Les applications C/S (client/serveur) et distribuée dans l'architecture internet
  - Voir le plan des cours et TP dans la partie *documents* des pages rsx102.seancetenante.com

- **Rappel**
  - le modèle OSI est illustré en annexe.
  - Les couches supérieures d'OSI de l'ISO ont été décrites dans le cours UTC505. Elles sont rapidement rappelées ci-dessous.



Préambule

Rappel

- **La couche session (Session layer)**
  - Établir des sessions pour des utilisateurs travaillant sur différents postes informatiques.
  - Gestion de dialogues (gestion d'un jeton de parole : seul le processus qui possède le jeton peut réaliser une opération critique).
  - Synchronisation par gestion de points de reprise
  - La notion de session est souvent associée à l'utilisation de base de données ou de transfert de fichiers.

- **La couche présentation (Presentation layer)**
  - Syntaxe et sémantique de l'information transmise.
  - Encodage et décodage de données suivant une norme reconnue (Unicode, MIME, HTML, ASN.1/BER, *Abstract Syntax Notation-One/Basic Encoding Rule*, etc.)

Préambule

Rappel

- **La couche application (Application layer)**
  - C'est le point d'accès aux services réseaux.
    - Normes et protocoles proches de l'utilisateur de services communicants.
    - Bibliothèques et API (packages, librairies) d'accès à des services tels que :
      - Transferts et systèmes de fichiers
      - Courrier électronique
      - Messagerie instantanée
      - VoIP, ToIP ; visioconférence
      - Exécution de travaux ou de sessions à distance
      - Consultation et gestion d'annuaires

  - Dans l'architecture liée à internet, on considère plutôt :
    - La couche **application** de l'architecture TCP/IP
    - Pour le modèle lié à internet, on regroupe sous le terme de couche Application les couches session, présentation et application d'OSI (les couches supérieures d'OSI).

1 - Introduction

1.1 - Définitions

- **Client-serveur**
  - Mode de communication à travers un réseau entre plusieurs programmes ou logiciels :
    - l'un, qualifié de **client**, envoie des requêtes ;
    - l'autre, qualifié de **serveur**, attend les requêtes des clients et y répond.
  - Principales architectures
    - **Architecture à deux niveaux**, *two-tier architecture*. Cf. figure ci-contre.
    - **Architecture à 3 niveaux**, *three-tier architecture*.
      Ex. : un client web demande une ressource à un serveur web associé à un serveur d'application qui est le client d'un serveur de données.
    - ***Architecture pair à pair**, peer-to-peer…*

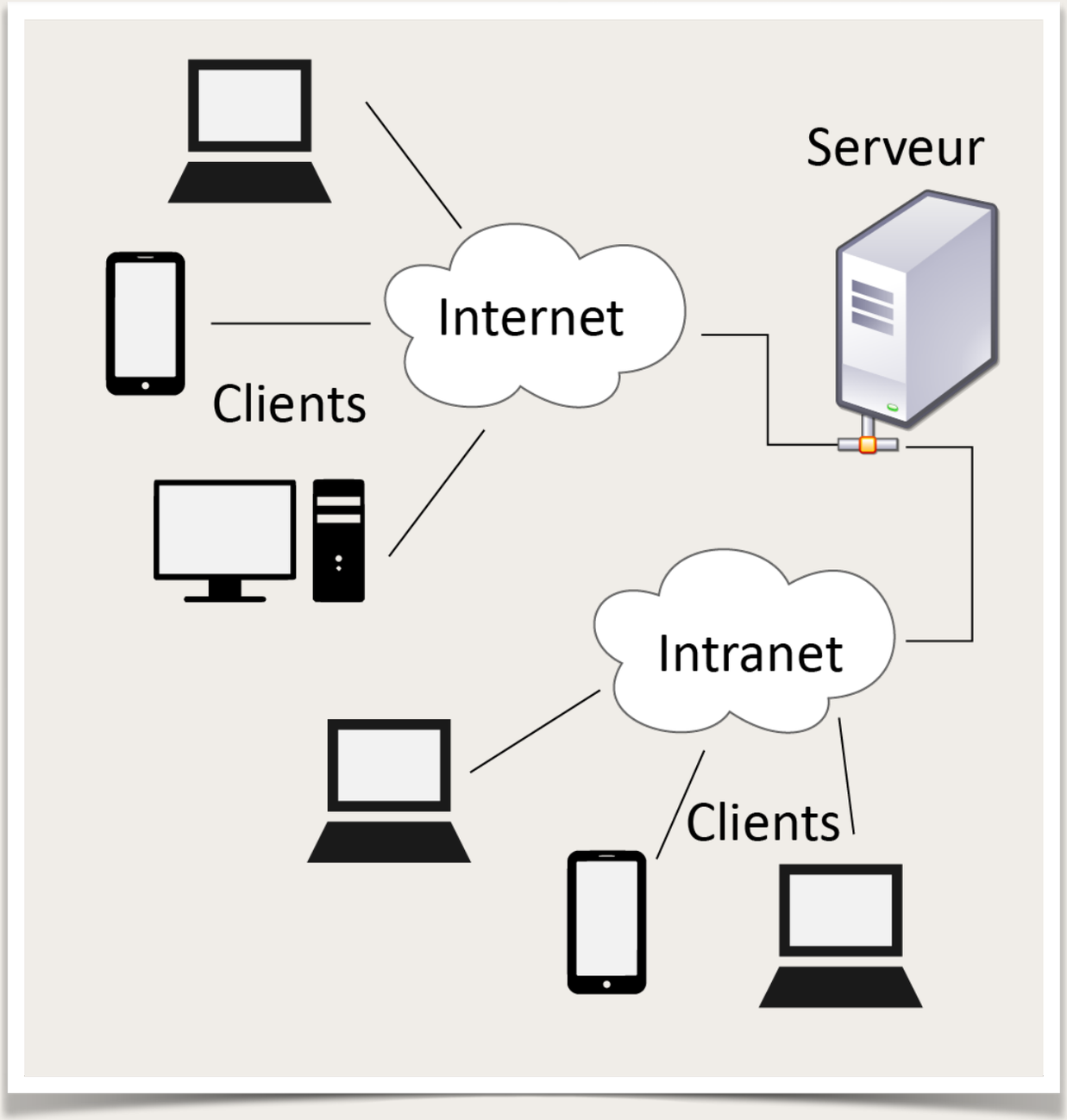


Fig 1.1 - Architecture Client-Serveur ; Internet et Intranet

1 - Introduction

1.1 - Définitions

- **P2P**
  - Une architecture **pair à pair** (***peer-to-peer*** ou P2P en anglais) est un environnement client–serveur où chaque programme connecté est susceptible de jouer **tour à tour** ou **à la fois** le rôle de client et celui de serveur.
  - Les composants d'un système P2P sont appelés **nœuds**, **pairs** ou **utilisateurs**.
  - Certains systèmes sont :
    - partiellement centralisés ; un serveur intermédiaire gère une partie des échanges
    - totalement décentralisés ; sans infrastructure particulière

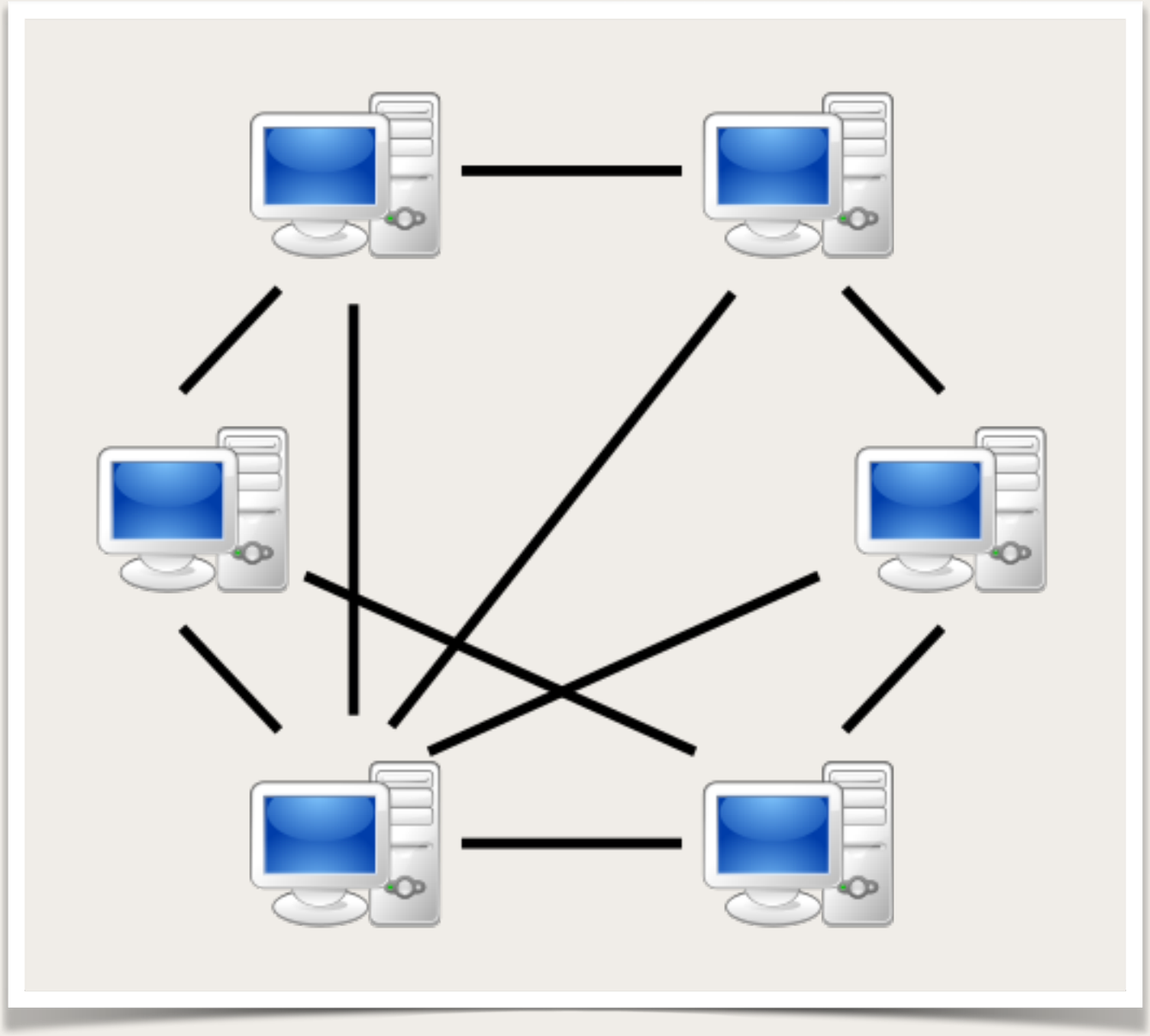


Fig 1.2 - Architecture P2P ou pair à pair

Introduction

- DNS, *Domain Name System* (Système de Nom de Domaine), mis en place en 1988, recouvre deux notions :
  - Une **base de données répartie**, grâce à un grand nombre de serveurs de noms qui communiquent entre eux.
  - Un **protocole**, de niveau application :
    - Il utilise UDP pour le transport
    - Le port 53 est utilisé par convention.
  - *Domain Name System* sert notamment à traduire un nom de domaine en adresse IP, ou en d'informations d'autres types.
  - Voir STD 13, RFC 1034 (*Domain names - concepts and facilities)*, RFC 1035 (*Domain names - implementation and specification)*, etc.

Service de noms

- Un service de noms permet aux utilisateurs d'accéder à une ressource en la désignant par son nom, plutôt que par son adresse.
- DNS est un service de nommage standard sur internet (RFC 1033 à 1035).
- Un des objectifs est donc de retrouver **l'adresse IP** d'une machine à partir de **son nom de domaine**.

Nom de domaine

- Il résulte d'un système hiérarchique :
  - Domaine de haut-niveau (TLD, ***Top Level Domain***) ou domaine de premier niveau
    - **Générique** : gTLD ; *generic TLD*
      Ex. : .com, .net, .org, .mil, .gov, .biz, .info, .name, .pro, .aero, .coop, .xxx, .museum…

      Depuis 2014, des milliers de nouveaux gTLD (nommés nTLD pour **News TLD**) peuvent être demandés, avec priorité aux propriétaires de marques déposées.
      Ex. : .paris, .bzh, .sncf, .guru…
    - **National** : ccTLD ; *country code TLD*
      Ex. : .fr, .uk, .nl, .it, .jp, .eu, .us…
- Sous-domaine ou ***label*** : chaque sous-domaine est enregistré auprès du domaine supérieur.
  Il faut donc :
  - la validation du domaine supérieur
  - ajouter un enregistrement dans les serveurs de noms du domaine supérieur.
- Par exemple, dans le domaine `media.nperf.com`, `media` est un sous-domaine de `nperf.com`.

- **Nom de domaine**
  - Les TLD sont gérés par l'ICANN (Internet Corporation for Assigned Names and Numbers) ; l'ICANN délègue la gestion de chaque TLD à un organisme appelé registre de haut-niveau (*registry*).
  - L' AFNIC (l'Association Française pour le Nommage Internet en Coopération) est le registry pour la France. Elle gère les ccTDL .fr, .re, .yt, .wf, .tf et .pm
  - VeriSign est le *registry* qui gère les .com, .net, .org…
  - Exemple 1

simon.info.arcep.fr

fr =	ccTLD pour la France ; géré par l'AFNIC
arcep =	Domaine de 2d niveau, enregistré et validé par l'AFNIC (propriétaire : l'Autorité de Régulation des Communications Electroniques et des Postes)
info =	Service informatique enregistré pour l'ARCEP.
simon =	Nom d'une machine (celle de Simon) au sein du service informatique de l'ARCEP.

- **Nom de domaine** (suite…)
  - Exemple 2

ftp.aecnam.asso.fr

fr = ccTLD pour la France ; géré par l'AFNIC

aecnam.asso = Domaine composé d'une association enregistré et validé par l'AFNIC

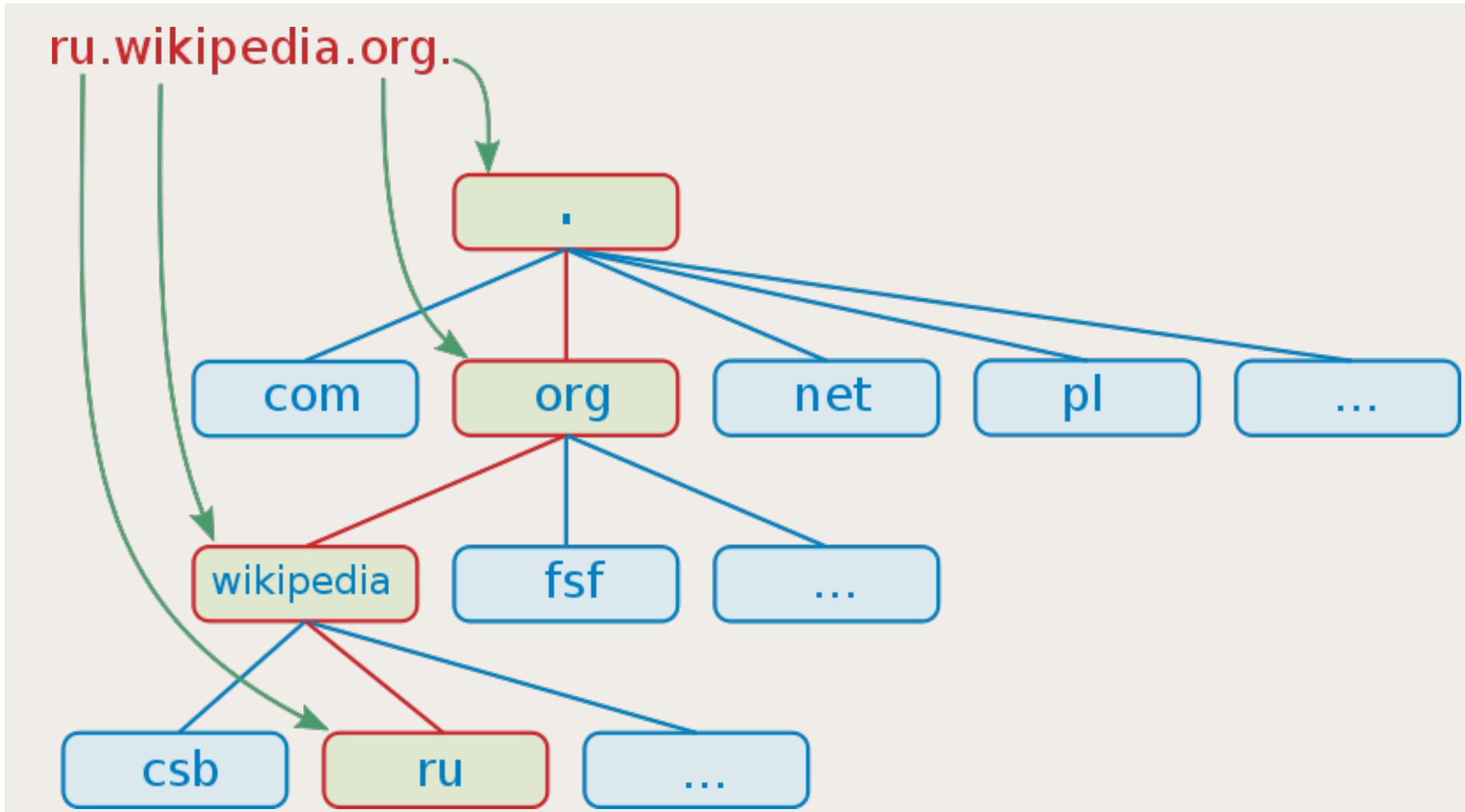
ftp = Nom d'une machine (ou d'un service) de l'association.

- Dans l'exemple 2 ci-dessus, **aecnam** est un élément de nom de domaine ou ***label***.
- Un élément d'un nom de domaine a au maximum 63 caractères.

2 - Applications client-serveur dans internet

2.1 - DNS - Domain Name System

- **Nom de domaine** (suite…)
  - Exemple 3 - La hiérarchie du domaine « ru.wikipedia.org. »



Racine

TLD

SLD
Second level domain

Fig 2.1 - Hiérarchie de domaine

FQDN : Fully Qualified Domain Name

- Un nom de domaine pleinement qualifié (FQDN) est un nom de domaine écrit de façon absolue.
  Il comporte donc :
  - tous les sous-domaines (ou labels),
  - le domaine de premier niveau (TLD),
  - et il est ponctué par un point final qui représente la racine.
- Exemple : `ru.wikipedia.org.`

Avec un point final !

Enregistrer un nom de domaine

- On utilise un **bureau d'enregistrement** de noms de domaine, ***registrar*** en anglais, qui gère la réservation de nom de domaine, conformément aux règles imposées par les registres de haut-niveau (***registry***).
- Voir :
  - www.gandi.net, www.ovh.com/fr/domaines/, etc.
  - ou l'annuaire des bureaux d'enregistrement de l'AFNIC : www.afnic.fr/fr/votre-nom-de-domaine/comment-choisir-et-creer-mon-nom-de-domaine/

- **Enregistrement de ressources**
  - Un enregistrement de ressources, ***resource record***, se compose de **cinq éléments** :

    `Nom de domaine ; Durée de vie ; Classe ; Type ; Valeur`

  - `Nom de domaine` : un **FQDN**, qui se termine donc par un point (.), symbole de la racine des domaines. Ex. : `simon.info.arcep.fr.`
  - `Durée de vie` en secondes
  - `Classe` : IN pour Internet
  - `Type` : type d'enregistrement
    - A = Address Record : la valeur est l'adresse IP v4 du nom de domaine
    - AAAA = Address Record : la valeur est l'adresse IP v6 du nom de domaine
    - MX = Relai de messagerie
    - SOA = Start of Authority : serveur principal d'une zone
    - NS = Serveur de noms
    - CNAME = nom canonique : Alias de nom de domaine
    - PTR = Pointeur
    - HINFO = description de l'hôte
    - TXT = Texte de commentaire
  - `Valeur` : en fonction du type

Enregistrement de ressources

- Exemple :

Nom de domaine	Durée de vie	Classe	Type	Valeur
fr.wikipedia.org.	575	IN	CNAME	text.wikimedia.org.
text.wikimedia.org.	1522	IN	CNAME	text.esams.wikimedia.org.
text.esams.wikimedia.org.	2193	IN	A	91.198.174.232

2 - Applications client-serveur dans internet

2.1 - DNS - Domain Name System — dig

- **Exemple avec l'utilisation de dig :**
  - *dig* est une commande du package dnsutils sous Linux pour **interroger** des serveurs DNS. Les réponses sont des **enregistrements de ressources**.

```
iMac-F:~ francois$ dig fr.wikipedia.org

; <<>> DiG 9.6.0-APPLE-P2 <<>> fr.wikipedia.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28507
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;fr.wikipedia.org.   IN   A

;; ANSWER SECTION:
fr.wikipedia.org.       575     IN  CNAME   text.wikimedia.org.
text.wikimedia.org.     1522    IN  CNAME   text.esams.wikimedia.org.
text.esams.wikimedia.org. 2193 IN   A    91.198.174.232

;; Query time: 42 msec
;; SERVER: 212.27.40.241#53(212.27.40.241)
;; WHEN: Sat Aug 27 20:15:20 2011
;; MSG SIZE  rcvd: 104
```

Résolution de noms

- L'espace des noms DNS est divisé en zones distinctes. Cet espace forme un arbre et chaque zone contient :
  - une partie de l'arbre
  - des serveurs de noms :
    - un serveur primaire
    - des serveurs secondaires
- Un programme demandeur transmet une requête pour un nom de domaine à un solveur (*resolver*) local.
  - Si la réponse est dans le cache local, celle-ci est retournée au programme demandeur ;
  - Si la destination est locale, le solveur local donne la réponse, en retournant un enregistrement officiel ;
  - si la destination est distante, le solveur local peut répondre si l'information est disponible dans son cache ;
  - sinon, le solveur interroge le serveur primaire, qui résout la requête directement ou par requêtes **itératives** ou **récursives** jusqu'à l'obtention de l'enregistrement officiel.

- **Résolveurs DNS**
  - BIND 9 est un logiciel client-serveur (C-S) répandu pour fournir un service de noms. Il utilise un démon (*deamon*) : `named`, pour sa partie serveur.
  - Voir aussi Knot Resolver et Unbound.
- **Commandes et outils**
  - **dig** est une commande du package dnsutils sous Linux pour interroger des serveurs DNS.
    - `dig fr.wiktionary.org`
    - `dig mx gmail.com`
  - **host** affiche plus simplement les redirections DNS.
    - `host fr.wiktionary.org`
  - **nslookup** sera utilisable sous Windows au lieu de dig.
  - **whois** permet d'effectuer des recherches sur les bases de données de bureau d'enregistrement.
  - RDAP, *Registration Data Access Protocol* est une alternative à Whois ; voir **rdap.org**

- **DoH ; DNS over HTTPS**
  - DNS via HTTPS, *DNS over HTTPS* (**DoH**) est un protocole pour la résolution DNS à distance via le protocole HTTPS.
  - Les requêtes DNS habituelles sont réalisées en clair (non chiffrés) vers le port 53 du serveur DNS par défaut, ce qui pose des problèmes de sécurité et de confidentialité.
  - **DoH** permet de sécuriser les requêtes en faisant passer le trafic DNS sur le protocole sécurisé HTTPS, vers un serveur tel que Cloudflare (https://mozilla.cloudflare-dns.com/dns-query) ou NextDNS (https://trr.dns.nextdns.io/).
  - Mozilla Firefox est le premier navigateur web à proposer ce protocole DoH
    - Voir : https://support.mozilla.org/fr/kb/dns-via-https-firefox

- **À voir :**
  - RFC 9499: DNS Terminology - Stéphane Bortzmeyer
  - Guide pratique du titulaire d'un nom de domaine en .fr - AFNIC

2 - Applications client-serveur dans internet

2.1 - DNS - Domain Name System — Exercices

Exercices

Vos réponses avec le document Exercices-RSX102-DNS.docx

Interrogation de serveurs DNS :

- a/ Quelle est l'adresse IP de *amio-millau.fr* ?
- b/ Quel est l'enregistrement de type MX lié à *amio-millau.fr* ?

Enregistrement de nom de domaine :

- c/ Comment procéder pour enregistrer un nom de domaine :
  - en .fr ;
  - en .com ?
- d/ Combien coûte l'enregistrement de nom de domaine :
  - en .fr ;
  - en .com ;
  - en .security ?
- e/ Quel est le rôle de l'AFNIC ?
- f/ Qui est le propriétaire du domaine *aliceandbob.io* ?
  - Et comment avez-vous fait pour le savoir ?

DoH :

- g et h/ Activez DNS-over-HTTPS (DoH) dans Firefox et dans Windows 11.
  - Quels pages web vous a aidé.

Introduction ; les standards de base du web

- La toile, WWW, *World Wide Web*.
- Créé au CERN (Conseil Européen pour la Recherche Nucléaire) par Tim Berners-Lee en 1989, il repose alors sur les standards URL, HTTP, HTML, puis CGI.
- Depuis, d'autres standards importants doivent être considérés.
- www.home.cern/science/computing/birth-web
- En 1993, le CERN a mis le logiciel World Wide Web dans le domaine public. Plus tard, le CERN a publié une version sous licence libre, pour optimiser sa diffusion. Ces actions ont permis au web de prospérer.

Fig 2.1 - Tim Berners-Lee en 2010

Fig 2.2 - Premier logo du web, par Robert Cailliau

- **Introduction ; les standards de base du web**

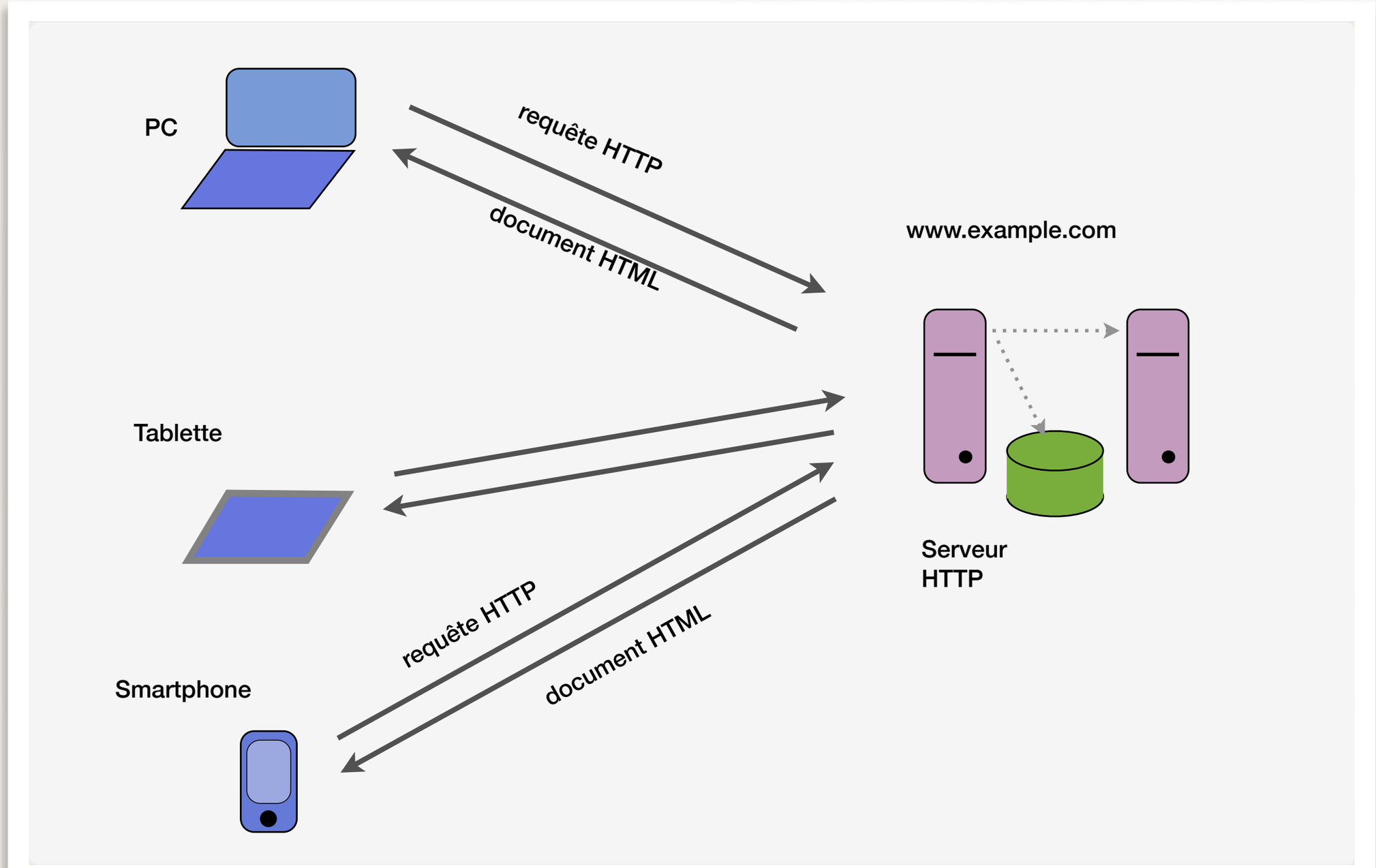


Fig 2.3 - Fonctionnement de base du web

- **URL ; URI**

  - URL , *Uniform Resource Locator*.

  - Format de nommage de ressources ; adresse réticulaire ; (usuellement) adresse web.

- On décompose en général une URL en quatre parties :

  - Le nom du protocole (http ; https ; ftp ; file ; mailto ; news...), suivi par « : »
  - Le nom du serveur : le nom de domaine du serveur, précédé par « // »
  - (en option) Le numéro de port TCP ; pour HTTP, le n° de port par défaut est 80.
  - Le chemin d'accès à la ressource ; si la ressource nécessite des paramètres, elle est suivi par un point d'interrogation (?) et des paramètres.

- **URL ; URI**

```
https://www.w3.org/People/Berners-Lee/FAQ.html
```

Protocole — Nom de domaine — Accès à la ressource

```
http://altavista.digital.com:8080/find.pl?q=url&lang=fr
```

Protocole — Nom de domaine — Port — Accès à la ressource

- Voir : https://rsx102.seancetenante.com/anatomie-url.php

URL ; URI

- URI, *Uniform Resource Identifier*, est un identifiant de ressource physique ou abstraite.
- URI inclut les URL et les URN (Uniform Resource Name). Ce dernier permet d'identifier une ressource dans un espace de noms.

```
urn:ietf:rfc:2396
```

Identifiant de type URN pour le RFC 2396

```
urn:isbn:0-395-36341-1
```

Identifiant de type ISBN (International Standard Book Numbers) :
Référence d'un livre publié.

```
http://fr.wikipedia.org/wiki/Uniform_Resource_Locator

http://codex.wordpress.org/Using_Permalinks
```

Identifiants de type URL

- **URL ; URI**
  - Voir :
    - URL, URI, URN, quelles différences ? - Alsacréations
  -

- **HTTP**
  - HTTP, *HyperText Transfer Protocol*
  - C'est le protocole de transfert de documents web (les documents HTML, les images JPEG, PNG, GIF..., les documents CSS, JS, etc.) demandés usuellement par un navigateur (client web) à un serveur HTTP.
  - Lorsque par exemple l'utilisateur clique sur un lien dans un document affiché par son navigateur, les opérations suivantes adviennent :

```
<a href="http://www.w3.org/Graphics/PNG/Overview.html">Portable Network Graphics</a>
```

  - Le navigateur détermine l'URL de la ressource demandée ;
  - Il demande au résolveur DNS l'adresse IP de www.w3.org ;

- **HTTP**
  - DNS répond avec un enregistrement de ressource :

```
www.w3.org.  300    IN     A      128.30.52.37
```

  - Le navigateur se connecte en TCP avec le port 80 de 128.30.52.37 ;
  - Il envoie une requête HTTP ; celle-ci comporte une commande GET indiquant la ressource demandée :

```
GET /Graphics/PNG/Overview.html HTTP/1.1
...
```

- **HTTP**
  - Le serveur retourne le document demandé ;
  - Le navigateur interprète le document, en affiche le texte et refait autant de requête HTTP que d'images ou autres ressources incluses dans le document.
  - Nota :
    - Avec HTTP version 0.9, la connexion TCP était libérée après chaque réponse à une requête.
    - Avec HTTP version 1.1, les autres documents liés à une ressource sont demandés et transférés pendant une seule connexion TCP.
    - Suivant le type MIME du document :
    - le navigateur l'affiche directement
    - ou un module d'extension (plug-in) le traite
    - ou le navigateur appelle une application auxiliaire (helper)
- HTTPS est une variante de HTTP sécurisée par l'usage des protocoles SSL ou TLS

- **HTTP**

  - Principales commandes HTTP :
    - GET Demande en lecture d'une ressource
    - HEAD Demande d'information sur la ressource
    - POST Soumission ou création de données ;
      L'URI indiquée est celle de la ressource qui traite les données envoyées
    - PUT Mise à jour de ressource
    - DELETE Suppression de ressource

  - Voir en annexe : HTTP : Requête et réponse.

- **HTTP/2**
  - HTTP/2 est finalisé par IETF depuis mai 2015 avec le RFC 7540.
  - Les deux axes majeurs de HTTP/2 sont la rapidité et la sécurité de la navigation.
  - Les avancées de HTTP/2 :
    - La compression des headers des requêtes et des réponses. Cette optimisation permet de réduire la bande passante lorsque les headers (tels que des cookies) sont similaires.
    - Le multiplexage des requêtes au serveur, pour économiser les multiples connexions entre le client et le serveur.
      - De nombreux flux logiques sont envoyés sur la même connexion TCP physique.
    - Le push des ressources du serveur au navigateur. Désormais, le serveur pourra envoyer l'ensemble des ressources référencées dans une même page (CSS, JS…), avant même que le navigateur n'ait analysé celle-ci.
    - HTTP/2 a réduit le problème de blocage de tête de ligne HTTP (*head-of-line blocking*), dans lequel les clients devaient attendre la fin de la première requête en ligne avant que la suivante ne puisse être envoyée. Cela grâce au multiplexage des requêtes et réponses.
  - Voir : https://http2-explained.haxx.se/fr

- **HTTP/3**
  - HTTP/3, alias HTTP-over-QUIC, est normalisé en juin 2022 (RCF 9114).
  - C'est le premier et principal protocole à être transporté sur QUIC.
  - Le client envoie sa requête HTTP sur un flux QUIC bidirectionnel initié par le client.
  - Les requêtes HTTP effectuées via HTTP/3 utilisent un ensemble spécifique de flux.
    - Les flux QUIC sont légèrement différents des flux HTTP/2.
  - Dans QUIC, les flux sont fournis par le transport lui-même (dans HTTP/2, les flux étaient créés dans la couche HTTP).
  - Les flux étant indépendants les uns des autres, le protocole de compression d'en-tête utilisé pour HTTP/2 ne pourrait pas être utilisé sans provoquer une situation de blocage de tête de bloc.

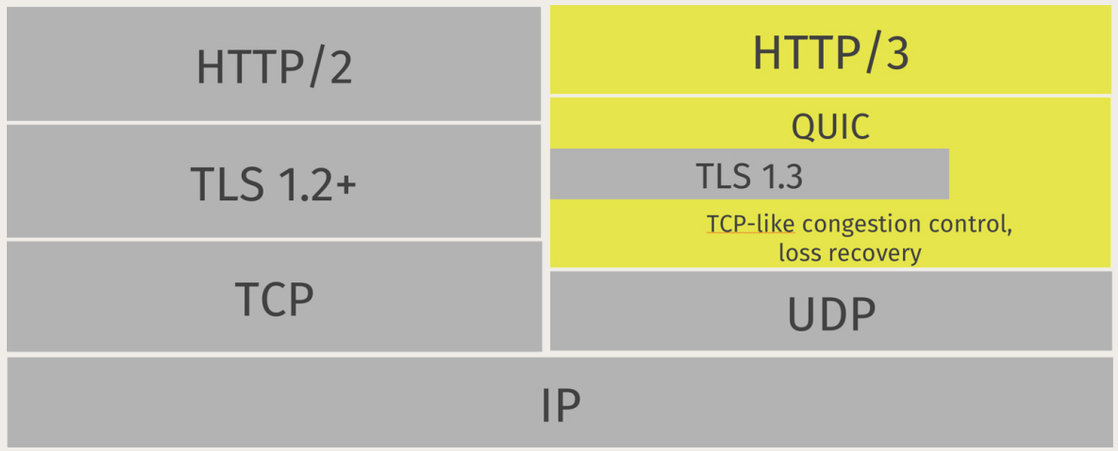


Fig 2.4 - HTTP/2 vs HTTP/3

- **HTTP/3**
  - HTTP/3 utilise des URLs en https: uniquement.
  - Le client envoie une requête HTTP sur un flux QUIC bidirectionnel initié par le client.
    - Le serveur renvoie sa réponse HTTP sur ce flux bidirectionnel : une trame HEADERS, une série de trames DATA et éventuellement une dernière trame HEADERS.
  - HTTP/3 envoi donc un ensemble de trames à l'autre extrémité.
  - Les types de trames les plus importants sont :
    - HEADERS, qui envoie des en-têtes HTTP compressés ;
    - DATA, envoie le contenu des données binaires ;
    - GOAWAY, veuillez arrêter cette connexion.
  - Un *push server* est la réponse à une requête que le client n'a jamais envoyée !
    - Les push serveur ne sont autorisés que si le côté client les a acceptés.
  - Voir :
    - https://http3-explained.haxx.se/fr
    - Ch. 3, § 3.3 - QUIC
    - https://www.bortzmeyer.org/9114.html

- **HTTP**

  - Quelques logiciels serveur populaires :
    - Nginx (30 % des sites web)
    - Apache HTTP Server (25 % des sites web)
    - Cloudflare server
    - LiteSpeed Web Server
    - Caddy
    - Apache Tomcat (Un serveur l'application)
  - Voir aussi :
    - MS IIS (Internet Information Services) ; Node.js ; Freenginx

  - Voir sur Geekflare : 6 serveurs Web Open Source pour les petits et grands sites
  - Guide Apache HTTP Server - Stéphane ROBERT

HTTP

- Quelques logiciels client :
  - NCSA Mosaic (1993 à 1997)
  - Netscape Navigator (1995 à 2008)
  - Internet Explorer (1995) très déprécié et remplacé par :
  - Microsoft Edge (2015 - project **Spartan**) un nouveau navigateur pour Windows 10 et 11 sur PC, tablette et smarphone
  - Mozilla Firefox (2004)
  - Opera (1994)
  - Safari (2003)
  - Chrome (2008)
  - Brave (2016)
- Autres utilitaires :
  - **Wget** : outil de ligne de commande qui permet de télécharger des fichiers dans votre répertoire actif
  - **cURL** : cf. Page suivante.

- **cURL,** ***client URL request library*** **ou** ***see URL***
  ou ***curl URL request library***

  - Utilitaire en ligne de commande qui repose sur une bibliothèque *libcurl*
  - Permet de lire, créer ou modifier une ressource à l'aide d'une URL.
  - Usage : voir `curl -h`
  - Exemple :
    - `curl -I https://amio-millau.fr` # requête HTTPS méthode HEAD
    - ```
      curl 'https://rsx102.seancetenante.com/documents/fichier10M.txt'
      -H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11;
      rv:78.0) Gecko/20100101 Firefox/78.0' -H 'Accept: text/plain'
      ```

  - Voir :
    - https://ec.haxx.se
    - https://www.it-connect.fr/curl-loutil-testeur-des-protocoles-divers/
      ```

HTML et CSS

- HTML, *HyperText Markup Language*
- C'est le langage de description de page web, constitué de balises et interprété par le navigateur.
  - C'est une simplification de SGML (*Standard Generalized Markup Language*), norme ISO de description de documents.
  - HTML décrit la présentation du document, qui reste statique. L'interactivité se limite aux liens hypertextes (les hyperliens) qui permettent d'afficher une autre page.
  - HTML a évolué en différentes versions jusqu'à une version 4.0 qui perdure depuis 1999. Certains préfèrent une variante basée sur XML, avec une version XHTML 1.0.
  - HTML 5.0, normalisé en octobre 2014, apporte de grandes et bonnes nouveautés.
- HTML est lié à la structure d'une page web.
- CSS, *Cascading Style Sheets* (Feuille de styles en cascade)
  - Ce langage offre des outils avancés permettant la mise en page et la présentation du contenu de pages web.
  - CSS3 reste en cours de spécification.

- **JavaScript**

  - JavaScript est un langage de script, reposant sur de la programmation événementielle.
    - Le code et les fonctions sont incluses dans le code d'une page HTML et exécutées, sur le client web, soit à différents niveaux du cycle de vie de la page, soit en réponse à certaines actions de l'utilisateur.
  - Par exemple, JavaScript :
    - aide à contrôler les données saisies dans des formulaires HTML
    - ou bien permet d'interagir avec le document HTML via l'interface DOM (*Document Object Model*).
  - Le '*Document Object Model*' décrit la structure et le contenu des pages web.
  - **WebAssembly** est conçu pour compléter et fonctionner avec JavaScript
    - WebAssembly est un langage de bas niveau, dont le bytecode est produit en compilant un langage de plus haut niveau (Rust, C , C++, C#, Go, Java, PHP, Python, Ruby, etc.).

- **Ajax**

  - Ajax, *Asynchronous JavaScript and XML*, est une combinaison de technologies comme :
    - **JavaScript** et **DOM** (Document Object Model), qui sont utilisés pour modifier l'information présentée dans le navigateur par programmation.
    - l'objet **XMLHttpRequest**, alias XHR, est utilisé pour dialoguer de manière asynchrone avec le serveur Web.
    - CSS (Cascading Style Sheets ; feuilles de style en cascade)
    - XML, utilisé pour l'échange de données. En alternative, les applications Ajax peuvent utiliser les fichiers texte ou JSON (*JavaScript Object Notation*).
  - Ajax permet de développer des sites web dynamiques (et des applications). Il s'inscrit dans la mode du *Web 2.0*.

- **Ajax** (suite…)
  - Avec cette nouvelle architecture, trois concepts sont utilisés :
    - **Des événements légers coté serveur** : des composants d'une application web peuvent effectuer des petites requêtes sur le serveur, pour obtenir des informations qui ne vont modifier qu'une partie du DOM ; le rafraichissement complet de la page n'est pas nécessaire.
    - **Asynchronisme** : les requêtes envoyées ne bloquent pas le navigateur, se sorte que l'utilisateur peut utiliser d'autres parties de l'application. L'interface graphique peut l'informer de la requête en cours.
    - **Généralisation** : tout événement de l'utilisateur (clic souris, survol du pointeur, action sur des touches du clavier) peut déclencher une requête asynchrone.
- **L'API Fetch**
  - Elle fournit une interface JavaScript permettant d'effectuer des requêtes HTTP et de traiter les réponses.
  - La méthode globale *fetch()* permet un accès pratique aux ressources récupérées de façon asynchrone sur le réseau.
  - À la différence de XMLHttpRequest qui fonctionne à l'aide de fonctions de rappel (callbacks), l'API Fetch utilise les promesses et fournit une meilleure alternative.
    - L'API Fetch intègre également des concepts HTTP avancés tels que le CORS (*Cross-origin resource sharing*) et d'autres extensions de HTTP.

- **CGI ; Scripts coté serveur**

  - CGI , *Common Gateway Interface* (littéralement : Interface de passerelle commune...)

  - Avec les formulaires, qui existent depuis HTML 2.0, il est devenu nécessaire de construire coté serveur des pages web dynamiques, générées en fonction de données saisies par l'utilisateur.

- **CGI ; Scripts coté serveur**

  - Voici un exemple :
    - ① L'utilisateur, après avoir rempli les champs d'un formulaire d'une page web, clique sur un bouton 'Submit' ;
    - ② Le navigateur envoie une requête HTTP :
      - avec l'URL généralement indiquée comme valeur de l'attribut 'action' de la balise <form> ;
      - et en associant les données saisies soit en fin d'URL (méthode GET) soit dans le corps de la requête HTTP (méthode POST) ;
    - ③ Le serveur HTTP :
      - reçoit la requête ;
      - lance un programme ou un script de type CGI, *Common Gateway Interface*...
      - qui récupère les données soit via des variables d'environnement (méthode GET), soit dans un flux d'entrée (méthode POST) ;
    - ④ Ce programme procède aux traitements en fonction des données transmises et retourne une page web relayée par le serveur HTTP ;
    - ⑤ Le navigateur affiche la page web résultante.

- **CGI ; Scripts coté serveur**

  - Nota :
  - En ③ , on préfère actuellement les variantes suivantes :
    - FastCGI : cela permet de lancer le programme qu'une fois. Il reste ensuite en cache pour une utilisation ultérieure
    - Les interprèteurs sont maintenant intégrés au sein du serveur HTTP sous forme de modules.
  - **Scripts coté serveurs**
    Les CGI sont des programmes compilés ou (très souvent) interprétés. Les langages serveur suivants sont populaires :
    - Perl
    - Python
    - JSP (*Java Server Pages*)
    - ASP (*Active Server Pages*)
    - PHP (*PHP HyperText Preprocessor*). Ex. PHP-FPM.
    - Servlets (avec Java)

- **CGI ; Scripts coté serveur**

  - Exemple avec PHP :
    - Sur un serveur web, on place un document : *formulaire.php*
      - Il nous faut alors un accès FTP (*File Tranfer Protocol*) pour pouvoir déposer des fichiers sur le serveur.
    - Le code de ce document est visible avec ce document texte : *formulaire.txt*
    - Ouvrir le formulaire : https://rsx102.seancetenante.com/minimum/formulaire.php
    - Examiner le code source du formulaire avec votre navigateur, avant puis après la soumission du formulaire.
    - ==> Le code PHP est exécuté sur le serveur, générant ainsi le HTML, qui sera ensuite envoyé au client.

  - Voir aussi : https://www.php.net/manual/fr/

- **Introduction**
  - XML est un langage de description adapté à la représentation structurée de données.
  - Langage de balisage extensible :
    - C'est un langage **extensible**, utilisant des balises (***markup***).
  - Un document XML est un fichier texte fortement structuré, destiné à représenter tout type de données.
  - Un inconvénient de HTML est qu'il ne permet pas de séparer la forme du contenu d'un document.
    XML peut servir à décrire un **contenu** ; la forme, le cas échéant, dépend de feuilles de styles associées.
  - XML, dérivé de SGML (*Standard Generalized Markup Language*), est un standard d'échange de données normalisé par W3C (*World Wide Web Contorsium*) : la version 1.0 en 1998 et la version 1.1 en 2004.
  - XML est utilisé pour :
    - stocker ou échanger des données ;
    - définir des langages, comme XHTML, SVG *(Scalable Vector Graphics)*, etc.

- **Exemples de balises :**

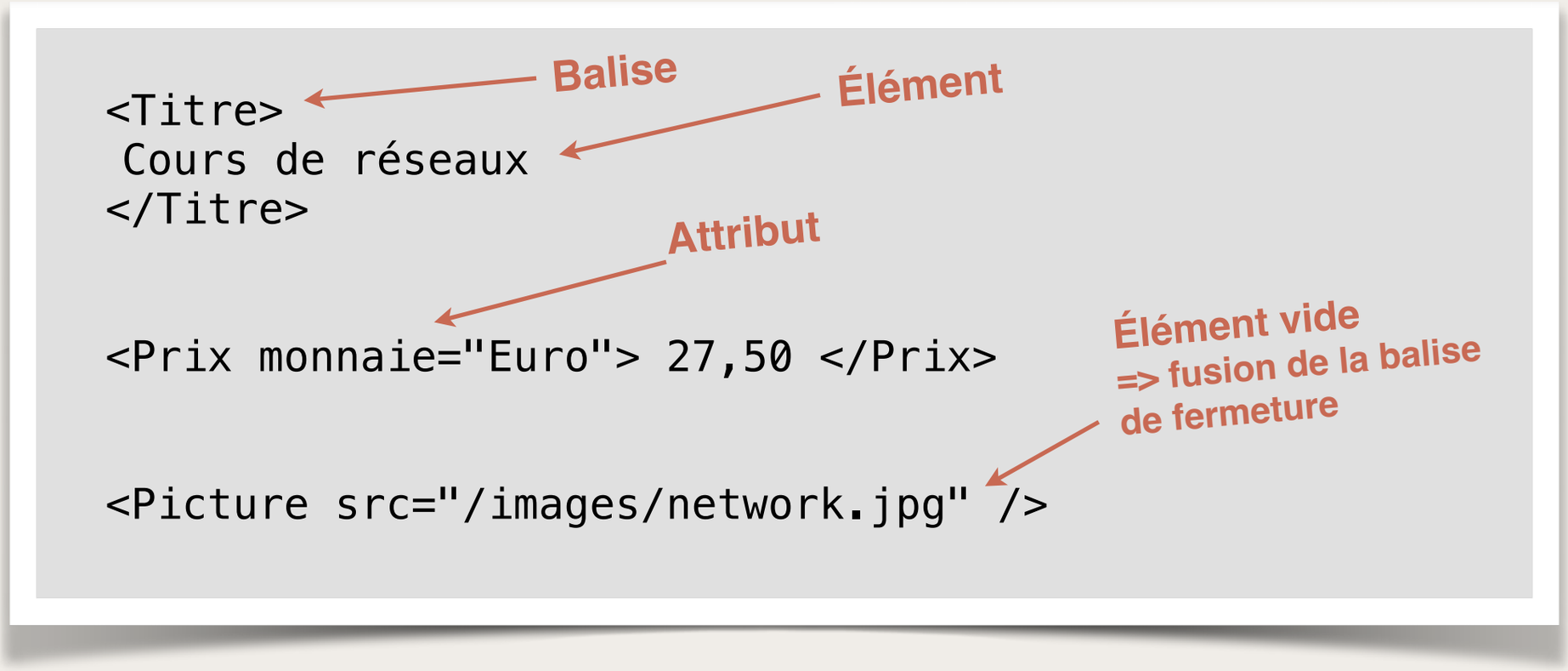


Fig 2.5 - XML : Balise, élement et attribut

- **Utilisation de XML**
  - Une application liée à XML utilise jusqu'à trois types de documents :
    1. Le document XML
    2. Le document DTD (*Document Type Definition*)
    3. Une feuille de style XSL (*eXtensible Stylesheet Language*)

- ① **Le document XML**
  - Il contient les données du document, structurées à l'aide des marqueurs ou balises.
  - Il doit être bien formé :
    - Il a toujours une et une seule racine, alias *nœud du document* ;
    - Une balise doit toujours être refermée. Sans entrecroisements.
      Ex. `<auteur>Victor <b>Hugo</b></auteur>`
    - Une balise ne peut pas commencer par `-.` et ne peut pas contenir d'espace ni `! »#$%&'()*+,/;<=>?@[\]^`{|}~`
  - Voici un exemple de document XML :

2.3 - XML ; Extensible Markup Language

```xml
<?xml version="1.0" encoding="iso-8859-1" standalone="no" ?>
<!DOCTYPE biblio SYSTEM "sample.dtd">
<!-- Ci-dessus le prologue : déclaration et DTD utilisé -->
<!-- Élément racine -->
<biblio>
  <!-- Premier enfant -->
  <livre>
    <!-- Élément enfant titre -->
    <titre>Les Misérables</titre>
    <auteur>Victor Hugo</auteur>
    <nb_tomes>3</nb_tomes>
  </livre>
  <livre>
    <titre>Les Confessions</titre>
    <auteur>Jean-Jacques Rousseau</auteur>
    <auteur>Jacques Perrin</auteur>
  </livre>
  <livre lang="en">
    <titre>David Copperfield</titre>
    <auteur>Charles Dickens</auteur>
    <nb_tomes>3</nb_tomes>
  </livre>
</biblio>
```

Fig 2.6 - Exemple de document XML

2 - Applications client-serveur dans internet

2.3 - XML ; Extensible Markup Language

- Il débute avec un prologue, composé d'une déclaration XML et, en option, d'une déclaration de type de document

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no" ?>
<!DOCTYPE biblio SYSTEM « sample.dtd">
```

- La déclaration de type de document précise l'élément racine du document XML (**biblio** dans l'exemple) et indique un lien (sample.dtd) vers un document **DTD,** *Document Type Definition*

2 - Applications client-serveur dans internet

2.3 - XML ; Extensible Markup Language

- La ligne suivante est la balise d'ouverture de la racine du document XML ; la racine (biblio) définit une structure (une liste de livres dans cet exemple) :

```
<biblio>
  <livre> premier ouvrage </livre>
  <livre> deuxième ouvrage </livre>
  <!-- etc. -->
</biblio>
```

- La racine peut admettre des attributs et contient un ensemble d'éléments fils (livre) ou petit fils (titre, auteur, etc.)

- ② **Le document DTD, *Document Type Definition***
  - Ce 2e type de document est celui indiqué le cas échéant avec la 2e ligne du prologue du document XML principal.
  - Il spécifie les règles pour les éléments et les attributs présents dans le document XML.
    - La DTD va permettre de vérifier la conformité du document XML.
    - Le document XML sera dit valide s'il respecte la DTD.
    - Le DTD est optionnel. L'attribut standalone prendra dans la déclaration XML la valeur "yes" s'il n'y a pas à rechercher de DTD externe, "no" sinon.

- XML Schema est une alternative plus sophistiquée à l'usage de DTD.
  - C'est un langage de description de format de document XML permettant de définir la structure et le type de contenu d'un document XML.

- ③ **Une feuille de style XSL, *eXtensible Stylesheet Language***
  - Ce 3e type de document, optionnel, est indiqué dans le document XML principal par une ligne du type

```
<?xml-stylesheet type="text/xml" href="biblio.xsl"?>
```

  - Le document XSL :
    - dicte le formatage des éléments lors de leur affichage ;
    - il est composé au moins de 2 parties distinctes et complémentaires
    - XSLT, un langage de transformation de documents (ex. génération d'une page HTML à partir d'un fichier XML et de la feuille de style)
    - XSL-FO, un ensemble d'instructions de formatage XML dédié à la présentation (*Formatting Objects*) de documents en vue d'une impression. Il comporte un modèle de format et des propriétés de style basées sur CSS2.

- **Exploitation de document XML**
  - Pour l'exploitation d'un document XML, deux API sont couramment utilisées par les applications :
  - DOM (*Document Object Model*) est une API pour des objets de type document définis par une structure XML ; cela permet de naviguer dans le document (décrit par un arbre) et d'accéder à ses parties.
    - La totalité du document XML est chargé en mémoire, sous forme d'objet.
  - SAX : *Simple API for XML* est une interface plus simple, pour un même usage.
    - Le document est analysé avec une approche en flux.

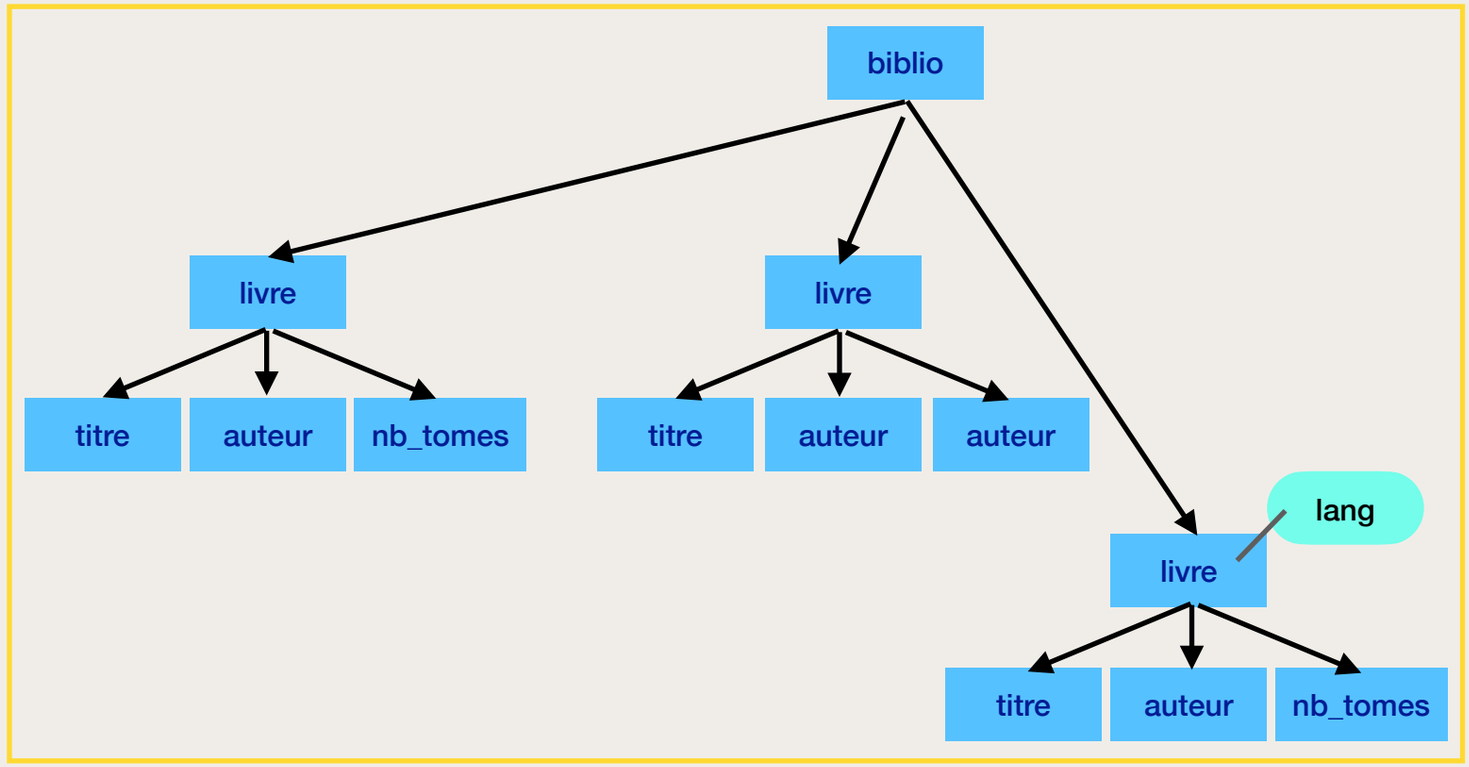


Fig 2.7 - Arbre du DOM d'un document XML

- **Principales applications de XML**
  - Un document XML sert dès qu'on a besoin d'échanger ou stocker des données structurées de façon standardisée et lisible par des machines.
    - Échange de données entre applications ;
    - Formats de fichiers applicatifs. Ex. les documents MS Office (docx, xlsx, pptx).
    - Fichiers de configuration d'applications, de serveurs, de frameworks
    - Formats de domaines spécialisés. Par ex. SVG (*Scalable Vector Graphics*), MathML (formules mathématiques), GML (données géographiques)…
    - Format d'échange pour les services web (SOAP et parfois REST).

- **À voir :**
  - https://developer.mozilla.org/fr/docs/Web/XML/XML_introduction
  - https://www.w3schools.com/xml/

2 - Applications client-serveur dans internet

2.4 - JSON ; JavaScript Object Notation

- **Comme XML, JSON est un langage de description adapté à la représentation structurée de données.**

```
{
  "menu": "File",
  "commands": [
     {
        "value": "New",
        "action":"CreateDoc"
     },
     {
        "value": "Open",
        "action":"OpenDoc"
     },
     {
        "value": "Close",
        "action":"CloseDoc"
     }
  ]
}
```

menubar.json

```
<?xml version="1.0" ?>
<menubar>
 <menu name="File">
  <command value="New" action="CreateDoc" />
  <command value="Open" action="OpenDoc" />
  <command value="Close" action="CloseDoc" />
 </menu>
</menubar>
```

menubar.xml

Fig 2.8 - Comparaison d'un document Json et XML

2.4 - JSON ; JavaScript Object Notation

- **JSON, *JavaScript Object Notation***
  - est un format de données d'abord dédié aux échanges entre le navigateur et le serveur
  - est très facile à utiliser en **JavaScript** ; il fait partie du langage
  - est un format texte complètement indépendant de tout langage
  - il se base sur deux structures :
    - Un tableau associatif de JavaScript (www.xul.fr/ecmascript/associatif.php)
    - Un tableau, donc une liste de valeurs ordonnées
  - il date de 2002 et est devenu populaire lorsqu'Ajax à commencé à être largement utilisé
  - il est même devenu un type de donnée dans PostgreSQL
- **Le format JSON reconnaît les même types de données que JavaScript**
  - **Number, String, Boolean** et **null** sont des primitives que l'on peut assigner à une clé qui doit être une chaîne
  - **Array** est une liste de clé-valeurs placées entre crochets
  - **Object** est une liste de clé-valeurs placées entre accolades
- **Pour en savoir plus : http://json.org**

- **Exemple d'échange entre client et serveur web**

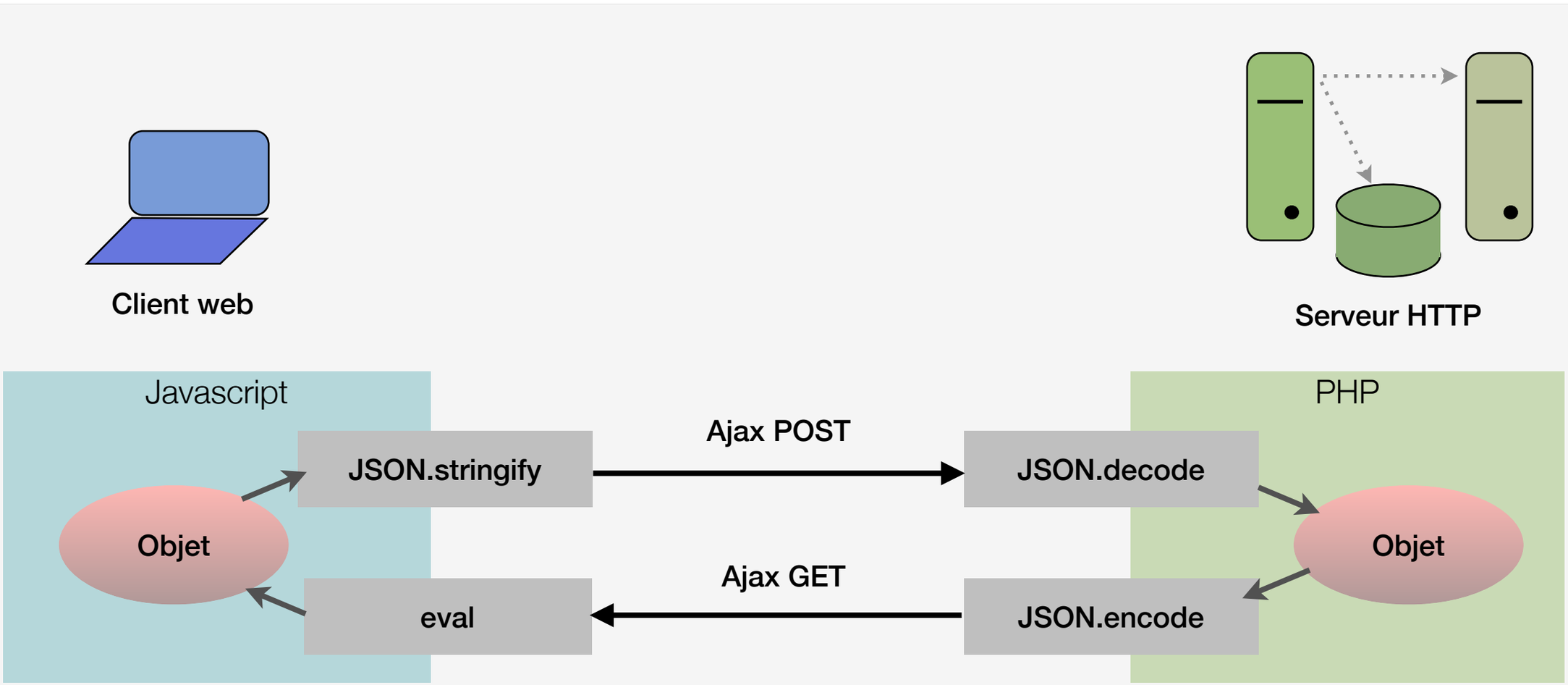


Fig 2.9 - Échange via Json entre client et serveur web

2 - Applications client-serveur dans internet

2.5 - Architectures web 3 tiers

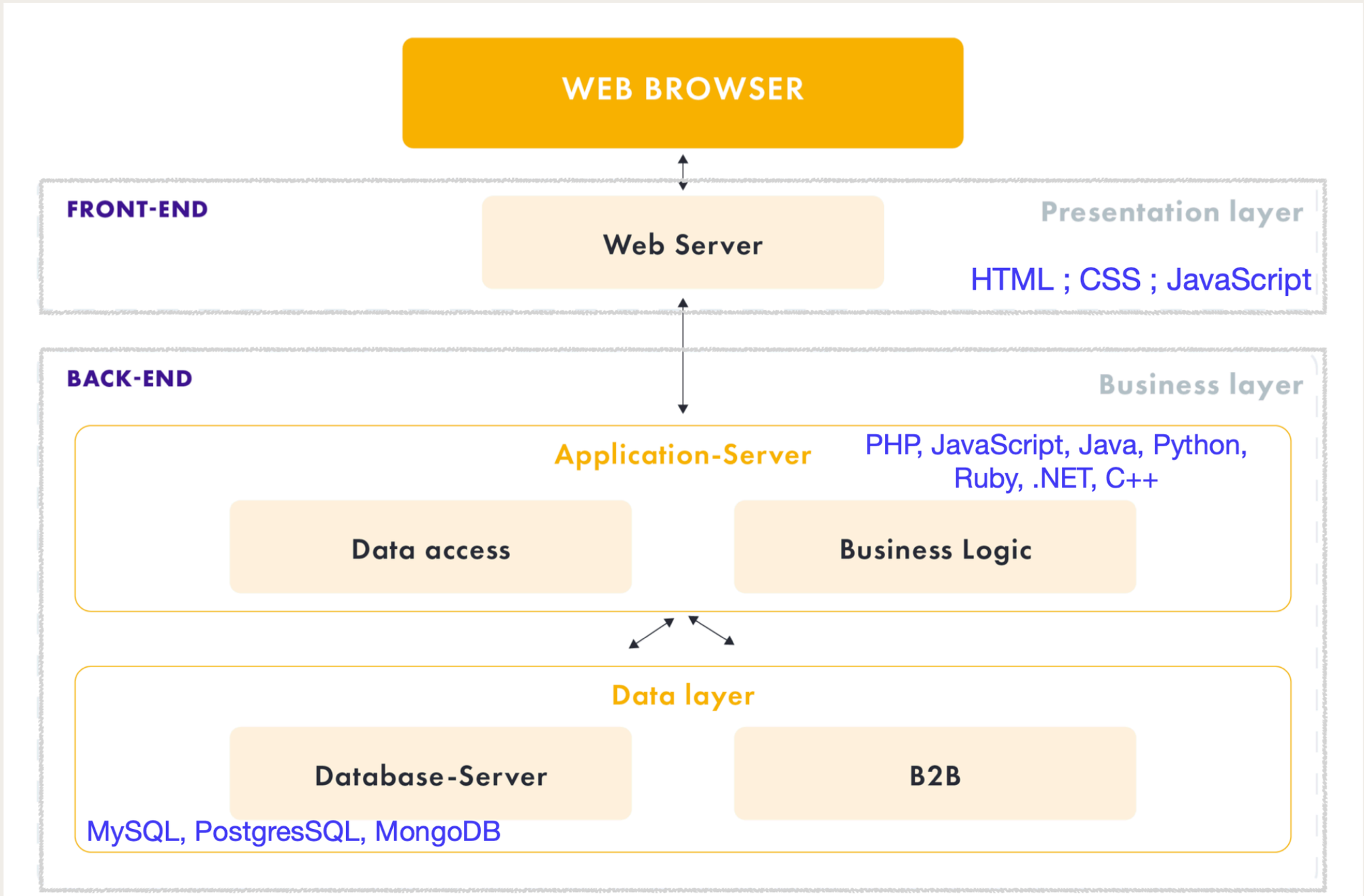
- **Front-End, Back-End, Data Base**
  - 

Fig 2.10 - Architectures web 3 tiers

- **Introduction**
  - Les services web constituent un système d'échange de données entre applications et systèmes hétérogènes dans des environnements distribués.
  - Généralement, les services web sont adaptés :
    - lorsque les applications distantes sont conçues indépendamment,
    - ou avec des applications hétérogènes (modèles, plates-formes, langages).
  - Les services web utilisent un ensemble de protocoles standardisés et ouverts comme :
    - **XML** pour les données échangées entre un serveur et une application cliente
    - **XML-RPC,** *XML-Remote Procedure Call* ou **SOAP**, *Simple Object Access Protocol,* pour le codage en XML des données constituant les appels de procédure distance
    - **HTTP**, FTP, SMTP ou XMPP, *eXtensible Messaging & Presence Protocol,* pour le transport des données
    - **WSDL**, *Web Services Description Language*, une description XML de la façon de communiquer en utilisant le service web
    - **UDDI**, *Universal Description Discovery & Integration*, un annuaire mondial pour faire connaitre le service web aux applications qui recherche le service web dont elles ont besoin.
  - Deux architectures sont très utilisées : **SOAP** et **REST**.

- **SOAP (*Simple Object Access Protocol*)**
  - Protocole de **RPC** orienté objet.
    - Il définit la structure des messages échangés par les applications via internet ;
    - Il permet la transmission de messages en XML entre objets distants ;
    - Si HTTP est souvent utilisé pour le transfert, d'autres protocoles peuvent être utilisés, comme SMTP, FTP, POP3, etc.
  - Avantages de SOAP :
    - Il repose souvent sur deux standards, XML et HTTP.
      - HTTP (avec la méthode POST) permet alors un transport entre sites distants sans reconfigurer les pare-feux et proxys.
    - Indépendant de la plate-forme et du langage.
  - Inconvénients de SOAP :
    - Verbeux, à cause de XML ;
    - Le couplage reste fort entre le serveur et ses clients.
    - Plus utilisés dans les nouveaux développements, mais ils persistent dans les systèmes existants.
  - **WSDL**, *Web Services Description Language,* décrit une interface publique d'accès à un service web, notamment dans le cadre d'architectures de type SOA, *Service Oriented Architecture*.
    - Description écrite en XML qui indique « comment communiquer pour utiliser le service »

- **REST, *REpresentational State Transfer***
  - Il utilise des fonctions HTTP classiques (GET, POST, PUT ou DELETE) pour respectivement **lire**, **déposer**, **modifier** ou **effacer** des ressources.
  - URI, *Uniform Resource Identifier*, permet l'identification de ressources exposées par un fournisseur de services.

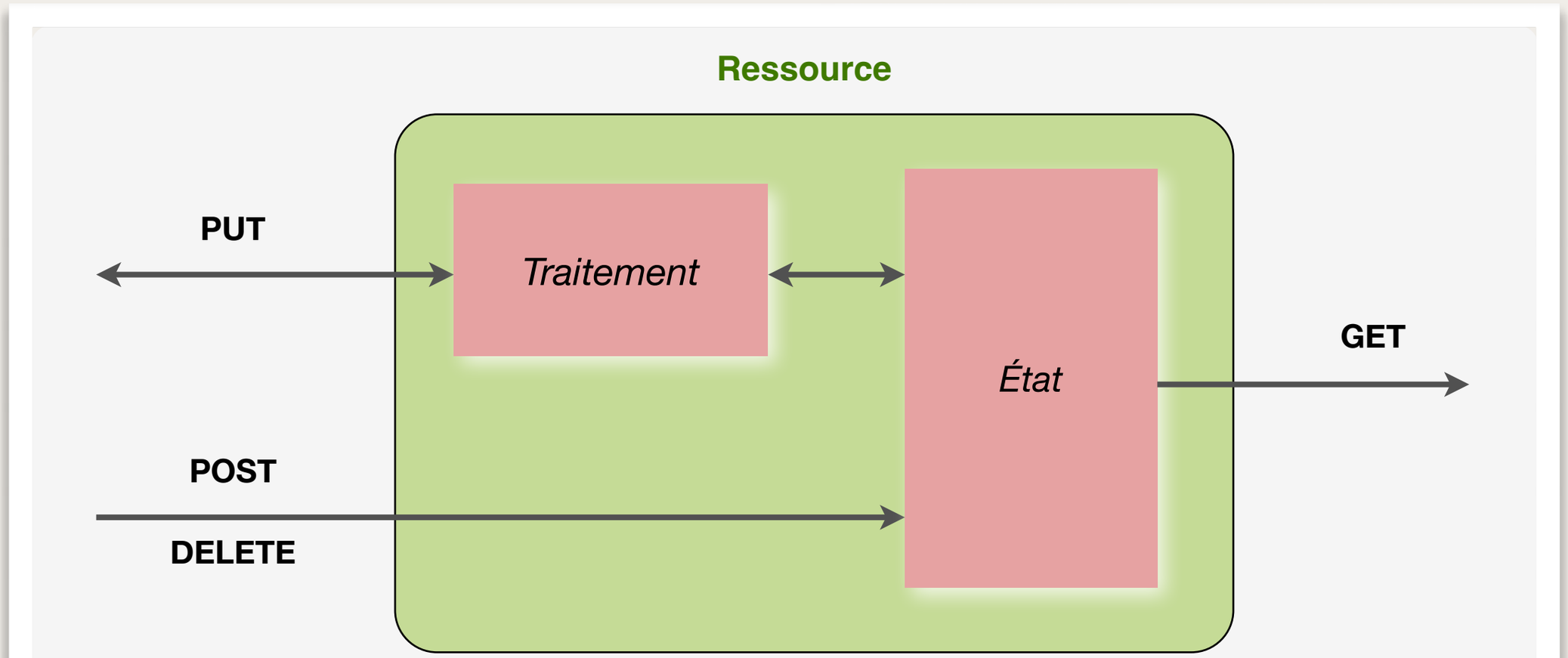


Fig 2.11 - Actions sur une ressource

- **REST**
  - Repose sur une communication asynchrone et un couplage lâche.
  - Se limite au champ du transfert de données d'une application à l'autre.
  - Il est bien adapté à des projets ouverts (à travers internet ou un extranet par exemple).
- **Les verbes utilisés permettent de réaliser les opérations CRUD : *Create, Read, Update & Delete***
  - **POST** pour ajouter une ressource (*Create*) ;
  - **GET** pour lire une ressource (*Read*) ; Méthode idempotentes ;
  - **PUT** pour modifier une ressource (*Update/Replace*) ; Méthode idempotentes ;
  - **DELETE** pour effacer une ressource (*Delete*) ; Méthode idempotentes ;
- **Voir**
  - Qu'est-ce qu'une API REST ? - OVHcloud

2 - Applications client-serveur dans internet

2.7 - SSL-TLS

- **SSL, *Secure Sockets Layer* & TLS, *Transport Layer Security***
  - SSL ~ Couche de sockets sécurisée
  - TLS ~ Sécurité de la couche de transport
  - Historique :
    - 1994 - SSL 1.0 développé par Netscape mais jamais mis en œuvre
    - 1995 : norme SSL 2.0
    - 1996 : SSL 3.0, la dernière version de SSL (RFC 6101 en 2008)
    - 1999 : norme TLS 1.0, développé par l'IETF pour succéder au SSL
    - 2006 : TLS 1.1
    - 2008 : TLS 1.2
    - 2011 : abandon de la compatibilité avec SSLv2 pour toutes les versions de TLS (RFC 6176)
    - 2014 : SSL 3.0 est banni
    - 2014 à 2018 : brouillons pour TLS 1.3
    - 2018 : norme TLS 1.3 :
      - Abandon du support des algorithmes trop faibles comme MD4, RC4, DSA ou SHA-224 ;
      - Négociation en moins d'étapes (plus rapide par rapport à TLS 1.2) ;
      - Réduction de la vulnérabilité aux attaques par replis.

2 - Applications client-serveur dans internet

2.7 - SSL-TLS

- **TLS, *Transport Layer Security***
  - TLS (ou SSL) fonctionne suivant un mode client-serveur. Il permet de satisfaire les objectifs de sécurité suivants :
    - l'authentification du serveur ;
    - la confidentialité des données échangées (ou session chiffrée) ;
    - l'intégrité des données échangées ;
    - de manière optionnelle, l'authentification du client (mais dans la réalité celle-ci est souvent assurée par la couche applicative).
  - Les protocoles de la couche application n'ont pas à être profondément modifiés.
  - TLS se place au niveau ***Session*** du modèle OSI : entre les couches *Transport* et *Application*
  - Ainsi HTTPS est une simple variante de HTTP qui implémente TLS.

2 - Applications client-serveur dans internet

2.7 - SSL-TLS

- **TLS**, *Transport Layer Security*

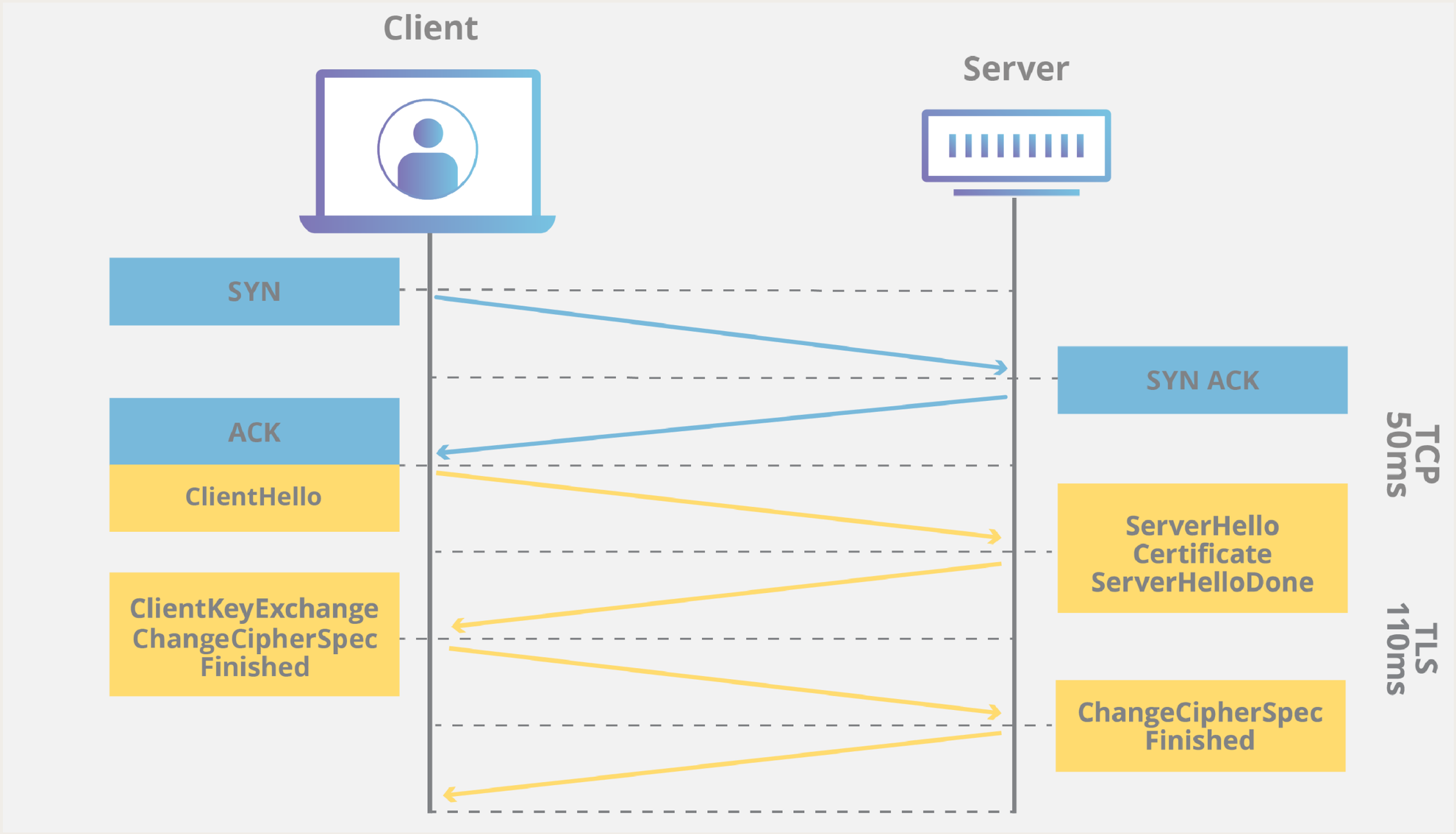


Fig 2.12 - TCP et TLS hanshakes

2 - Applications client-serveur dans internet

2.7 - SSL-TLS

- **TLS, *Transport Layer Security*, avec HTTPS** (suite…)
- Lorsqu'un utilisateur se connecte à un site web qui utilise TLS v1.2, les étapes suivantes se succèdent :
  - 1/ `[ClientHello]` Le navigateur du client envoie au serveur une demande de mise en place de connexion sécurisée par TLS.
  - 2/ `[ServerHello ; Certificate]` Le serveur envoie au client son certificat, qui contient sa clé publique, ses informations (nom et adresse postale de la société, e-mail de contact...) ainsi qu'une signature numérique.
  - 3/ `[ClientKeyExchange ; ChangeCipherSpec]` Le navigateur vérifie la signature numérique du certificat du serveur en utilisant les clés publiques contenues dans les certificats des autorités de certifications (AC) intégrées par défaut dans le navigateur.
    - 3.1/ Si l'une d'entre elles fonctionne, le navigateur web en déduit le nom de l'autorité de certification qui a signé le certificat envoyé par le serveur. Il vérifie que celui-ci n'est pas expiré puis envoie une demande OCSP, *Online Certificate Status Protocol*, à cette autorité pour vérifier que le certificat du serveur n'a pas été révoqué.
    - 3.2/ Si aucune d'entre elles ne fonctionne, le navigateur web vérifie la signature numérique du certificat du serveur avec la clé publique contenue dans celui-ci
      - 3.2.1/ En cas de réussite, cela signifie que le serveur web a lui-même signé son certificat. Un message d'avertissement s'affiche alors sur le navigateur web, prévenant l'utilisateur que l'identité du serveur n'a pas été vérifiée par une autorité de certification et qu'il peut donc s'agir potentiellement d'un site frauduleux.
      - 3.2.2/ En cas d'échec, le certificat est invalide, la connexion ne peut pas aboutir.

- **TLS, *Transport Layer Security*, avec HTTPS**
- Après vérification aboutie de la signature numérique (3/)
  - 4/ Le navigateur génère une clé de chiffrement symétrique, appelée clé de session, qu'il chiffre à l'aide de la clé publique contenue dans le certificat du serveur puis transmet cette clé de session au serveur.
  - 5/ Le serveur déchiffre la clé de session envoyée par le client grâce à sa clé privée.
  - 6/ Le client et le serveur commencent à s'échanger des données en chiffrant celles-ci avec la clé de session qu'ils ont en commun. On considère à partir de ce moment que **la connexion TLS est établie** entre le client et le serveur.
  - 7/ Une fois la connexion terminée (déconnexion volontaire de l'utilisateur ou si durée d'inactivité trop élevée), le serveur révoque la clé de session.

- Voir :
  - https://www.cloudflare.com/fr-fr/learning/ssl/transport-layer-security-tls/
  - https://doc.ubuntu-fr.org/tutoriel/comment_creer_un_certificat_ssl

2 - Applications client-serveur dans internet

2.8 - FTP ; File Transfer Protocol

- **FTP utilise TCP, *Transmission Control Protocol*, comme protocole de transport.**
- **Deux connexions sont utilisés**
  - Le client ouvre une connexion de contrôle ou de service sur le port 21 du serveur FTP. Cette connexion sert à l'échange de message de connexion, de contrôle et de signalisation.
  - Le client ouvre, pour chaque transfert,
    - En mode actif, le serveur FTP initialise une connexion de son port de données (port 20) vers le port spécifié par le client
    - En mode passif, une connexion est initiée par le client sur un port déterminé et communiqué préalablement par le serveur
- **La session FTP**
  - Elle commence par une procédure d'identification.
  - Exemple :

```
ftp ftp.seancetenante.com
user francois
pass *******
```

  - La session se termine par la commande ***quit*** ou ***bye*** du client, ou par un timeout du serveur.
  - Pendant une session, différentes commandes sont utilisable. Par ex. :
    - *help, pwd, lpwd, cd, lcd, ls, get, put, delete...*

2 - Applications client-serveur dans internet

2.8 - FTP ; File Transfer Protocol

- **Certains serveurs acceptent un accès anonyme :**
  - À installer de préférence au sein d'un intranet (pour des raisons de sécurité).
  - Accès en lecture seule au contenu d'un répertoire souvent nommé 'public' (et de ses sous-répertoires) ;
  - *anomymous* comme login, et votre adresse de courrier électronique comme mot de passe.
- **Les alternatives à FTP sont :**
  - SFTP, *SSH File Transfer Protocol* ; SFTP utilise le port 22, comme SSH
  - FTPS, *FTP over SSL*
  - WebDAV, *Web Distributed Authoring and Versioning*, une extension de HTTP.

2 - Applications client-serveur dans internet

2.8 - FTP ; File Transfer Protocol

- **Les logiciels client de transfert de fichier populaires sont**
  - FireFTP
  - FileZilla
  - Secure FTP
  - Cyberduck
  - CrossFTP
  - WinSCP (pour Windows)
- **Coté serveur, on trouve**
  - ProFTPd
  - Pure-FTPd
  - VsFTPd
  - Ou pour Windows : FileZilla Server
- **Voir aussi :**
  - Configuration d'un serveur ProFTPd - Wikilivres

2.9 - Serveurs proxy et reverse proxy

- **Qu'est-ce qu'un serveur proxy ?**
  - Un **serveur proxy**, ou serveur mandataire, joue le rôle d'**intermédiaire** entre un client et un serveur distant.
  - Par exemple, entre les ordinateurs connectés au réseau local d'une entreprise et Internet (et ses milliards de sites).
-

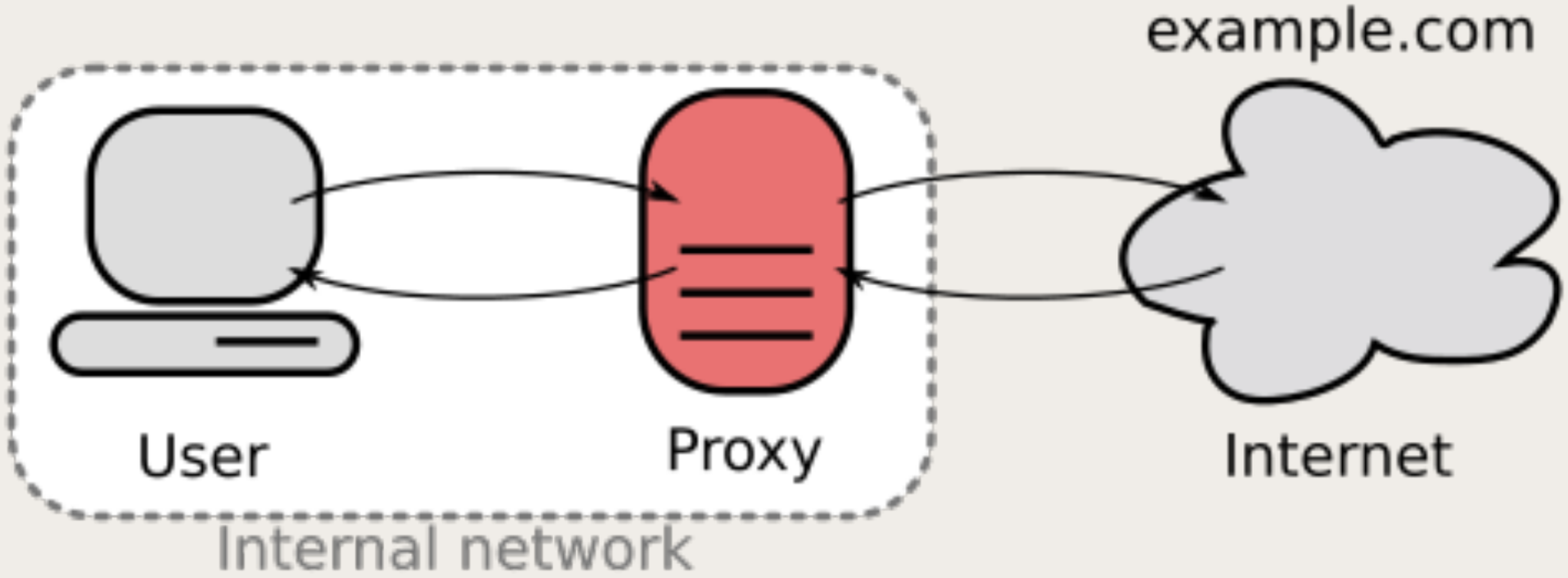


Fig 2.13 - Serveur proxy

2 - Applications client-serveur dans internet

2.9 - Serveurs proxy et reverse proxy

- **Les actions d'un serveur proxy :**
  - **Filtrage** : bloquer certains sites ou certaines catégories de sites.
  - **Cache** : mettre en cache les requêtes (exemple : page web) afin de les retourner plus rapidement aux postes de travail.
  - **Compression :** réduire le poids des pages au moment de retourner le résultat aux clients.
  - **Journalisation** : toutes les requêtes des clients seront stockées dans des journaux (logs).
  - **Anonymat** : puisque votre poste de travail se cache derrière le proxy, le serveur Web verra seulement le proxy.
  - **Droits d'accès** : puisque le proxy est un intermédiaire, il peut servir à positionner des droits d'accès pour filtrer les accès, au-delà du filtrage Web.
- **Quel logiciel pour un serveur proxy ?**
  - Squid : serveur proxy performant qui supporte le caching pour HTTP, HTTPS, FTP et améliore les temps de réponse en réduisant la consommation de bande passante.
  - Proxifier : pour Windows, macOS et Android.

- **Qu'est-ce qu'un reverse proxy ?**
  - Un serveur reverse proxy, ou proxy inverse, fonctionne comme une interface de communication sur le réseau.
  - Il reçoit les requêtes et les transmet à un serveur cible par procuration.
  - Un reverse proxy est toujours placé entre les clients (par exemple, les navigateurs Web) et les serveurs dorsaux ou backend (par exemple, les serveurs Web, les serveurs de base de données ou les applications).

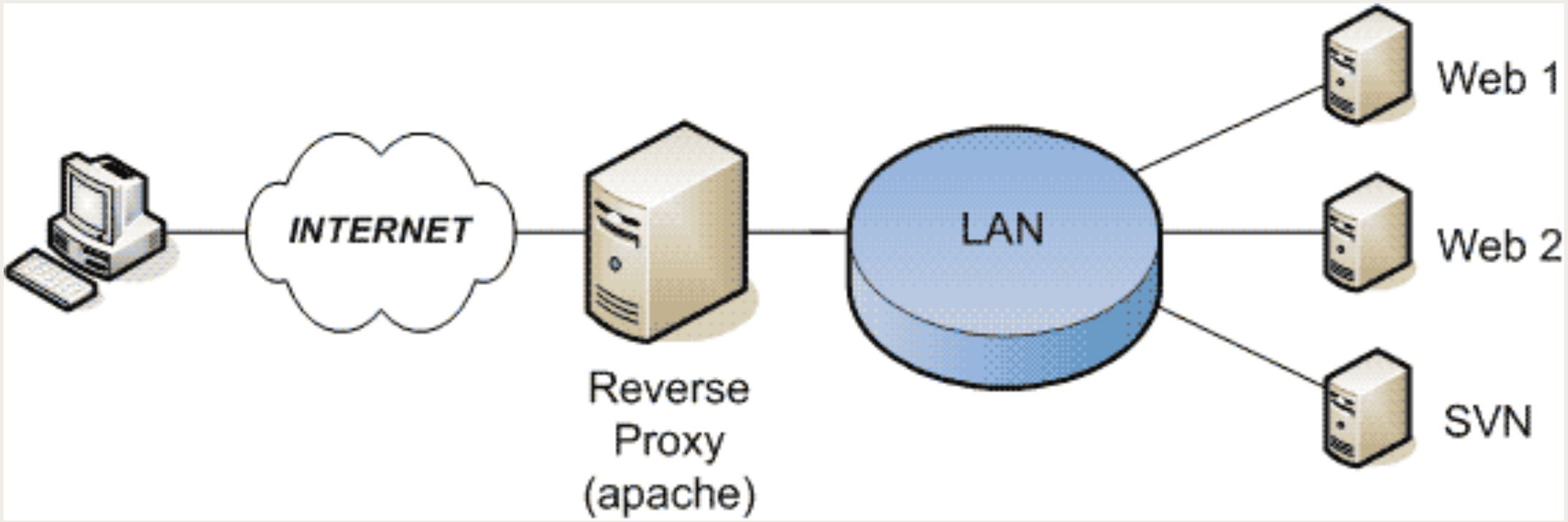


Fig 2.14 - Serveur reverse proxy

2 - Applications client-serveur dans internet

2.9 - Serveurs proxy et reverse proxy

- **Fonctionnement :**
  1. **Réception de la requête du client** : le reverse proxy accepte les requêtes HTTP, HTTPS, ou d'autres requêtes comme FTP ou WebSocket.
  2. **Analyse de la requête** : le proxy vérifie si la requête est valide, si elle présente des risques de sécurité et s'il existe une version mise en cache.
  3. **Renvoi vers le serveur approprié** : si la demande ne peut pas être satisfaite à partir du cache, le reverse proxy envoie la demande à l'un des serveurs internes.
  4. **Réponse au client** : le reverse proxy reçoit la réponse du serveur dorsal, la traite si nécessaire (ex. : chiffrement) et la renvoie au client qui a fait la demande.
- **Quel logiciel pour un reverse proxy ?**
  - Apache et son module mod_proxy
  - Nginx : simple, polyvalent, très répandu
  - Traefik : reverse proxy et load balancer open source pensé pour les environnements modernes : conteneurs, microservices, Kubernetes, cloud, etc.
  - HAProxy
  - Squid : en tant que reverse proxy avec un pare-feu PfSense.
  - Envoy : proxy L7 généraliste pour le cloud-native et les service meshes.

2 - Applications client-serveur dans internet

2.9 - Serveurs proxy et reverse proxy

- **Voir :**
  - Qu'est-ce qu'un serveur proxy ? - IONOS
  - Qu'est-ce qu'un reverse proxy ? - IONOS
  - Les serveurs proxy et reverse proxy pour les débutants - IT-Connect

2 - Applications client-serveur dans internet

2.10 - NFS ; Network File System

- **Présentation**
  - NFS est un **système de fichiers en réseau** développé par Sun Microsystems (racheté par Oracle en 2009). NFS est supporté par un grand nombre de systèmes d'exploitation.
  - Avec NFS, utilisateurs et programmes accèdent aux dossiers et fichiers distants comme s'ils étaient locaux.
  - C'est un système client-serveur
    - Le serveur rend des répertoires accessibles : il exporte ou partage des répertoires
    - Le client monte (***mount***) un répertoire lorsqu'il attache un répertoire distant à un système de fichier local.
  - NFS est souvent utilisé dans les environnements de virtualisation pour fournir un stockage partagé aux machines virtuelles.
  - L'IETF, *Internet Engineering Task Force*, est maintenant chargé du développement des protocoles NFS.
  - NFSv4.1 (RFC 5661) a été publié en 2010.

2.10 - NFS ; Network File System

- **Organisation**
  - NFS fonctionne grâce des différents démons :

Démon	Serveur	Client	Détail
nfsd	√		réponses aux requêtes client
mountd	√		demande de montage
nfslogd	√		journal
rquotad	√	√	quota
lockd	√	√	verrouillage
statd	√	√	surveillance de l'état du réseau

2 - Applications client-serveur dans internet

2.10 - NFS ; Network File System

- **Configuration du serveur**
  - Le but du partage de système de fichier est de :
    - partager des données entre utilisateur d'un même groupe
    - éviter la duplication de répertoires
    - offrir des programmes et des données centralisés
    - fournir de l'espace disque à des clients sans disque
  - Commande **share** (système Solaris) :
    - elle lit les répertoire à exporter dans /etc/dfs/dfstab
  - Commande **exportfs** (système Linux)
    - elle lit en /etc/exports : les répertoires à exporter, les machines qui peuvent y accéder et les droits d'accès associés.

- **Configuration de clients**
  - Commande showmount : obtenir des informations sur un serveur ;
  - Commande mount : montage de répertoire :
    - `mount -t nfs nom_du_serveur:dossier_distant dossier_local`
  - Commande umount : démontage de répertoire.

- **NFSv4**
  - L'IETF, Internet Engineering Task Force, est maintenant chargé du développement des protocoles NFS.
    - L'ensemble du protocole est repensé, et le code totalement réécrit.
    - Il s'agit d'un système de fichiers objet.
  - La version 4 de NFS est révisée en 2003 (RFC 3530) ;
  - NFSv4.1 (RFC 5661) a été publié en 2010 ;
  - NFSv4.2 (RFC 7862) a été publié en 2016.

- **NFSv4**
  - NFSv4, par rapport à ses prédécesseurs, embarque de nombreuses avancées telles que :
    - Une technologie de cache agressive (délégation) ;
    - Le regroupement des requêtes réseau (*Compound request*) ;
    - La sécurisation négociée et le chiffrement des données : Kerberos 5, Certificats (SPKM, *Simple Public Key Mechanism*), Clefs publiques/privées (LIPKEY, *Low Infrastructure Public Key*) ;
    - La capacité pour les clients de maintenir des sessions ou de les récupérer malgré un crash serveur ou une panne du réseau ;
    - La possibilité (à terme) de rediriger la charge de serveurs saturés vers un autre serveur, de manière transparente pour les clients ;
    - Le support d'attributs fichier nommés par l'utilisateur (ex. un attribut ‘photos’).

- **Autres systèmes de fichiers réseaux**
  - **SMB**, *Server Message Block2*, permet le partage de ressources (fichiers et imprimantes) sur des réseaux locaux avec des PC sous Windows.
  - **Samba** est une implémentation libre des protocoles SMB/CIFS pour **Linux et Unix**.
    - Samba v3 fournit sur un réseau local des fichiers et services d'impression pour divers clients Windows et peut s'intégrer à un domaine Windows Server ou faire partie d'un domaine Active Directory.

- **Systèmes de fichiers distribués**
  - **AFS**, *Andrew File System*, est un système d'archivage distribué.
  - **Lustre** est un système de fichiers distribué libre, souvent utilisé pour **de très grandes grappes de serveurs**.

- **Voir aussi**
  - Apache Subversion et Git, logiciels de gestion de versions.

3 - Protocoles de transport

3.1 - Préambule

- **La couche transport de l'architecture TCP/IP**
- **Ses deux principaux protocoles sont :**
  - TCP, *Transmission Control Protocol,* propose un service fiable orienté connexion
  - UDP, *User Datagram Protocol,* est un protocole plus simple, non fiable et sans connexion
- **On trouve également :**
  - QUIC, *Quick UDP Internet Connections,* transport pour **HTTP/3**
  - MPTCP, *MultiPath TCP ;* permettre à TCP d'utiliser plusieurs chemins d'accès
  - RTP, *Real-Time Transport Protocol* permet le transport de données soumises à des contraintes de temps réel, tels que des flux média audio ou vidéo.
    - Il utilise en fait UDP.
    - Il permet le transport de média pour les services de la voix sur IP, de vidéo conférence et de streaming.
    - RTP, protocole de transport, est toujours associé à un protocole de niveau Application comme SIP, *Session Initiation Protocol* (VoIP ou vidéo conférence) ou RTSP, *Real Time Streaming Protocol*.

3 - Protocoles de transport

3.2 - TCP et UDP

- **Voir le cours UTC505**
  - Le paragraphe 4 — *La couche Transport dans Internet* — du chapitre 5 du cours UTC505 détaille les protocoles TCP et UDP.
  - Extrait du cours UTC505

3 - Protocoles de transport

3.3 - QUIC

- **QUIC, *Quick UDP Internet Connections***
  - Nouveau protocole de transport standardisé en juin 2021 (RFC 9000), conçu au départ par Google.
  - Adopté par l'IETF, *Internet Engineering Task Force,* pour **remplacer TCP** dans le protocole HTTP/3, alias *HTTP-over-QUIC.*
  - QUIC utilise UDP et un chiffrement équivalent à TLS (*Transport Layer Security*)
- Avec QUIC, le chiffrement est obligatoire.
  - TLS ne fait que la poignée de main initiale et l'échange des clés. QUIC chiffre ensuite tout seul comme un grand, en utilisant les clés fournies par TLS.

Fig 3.1 - Logo de QUIC

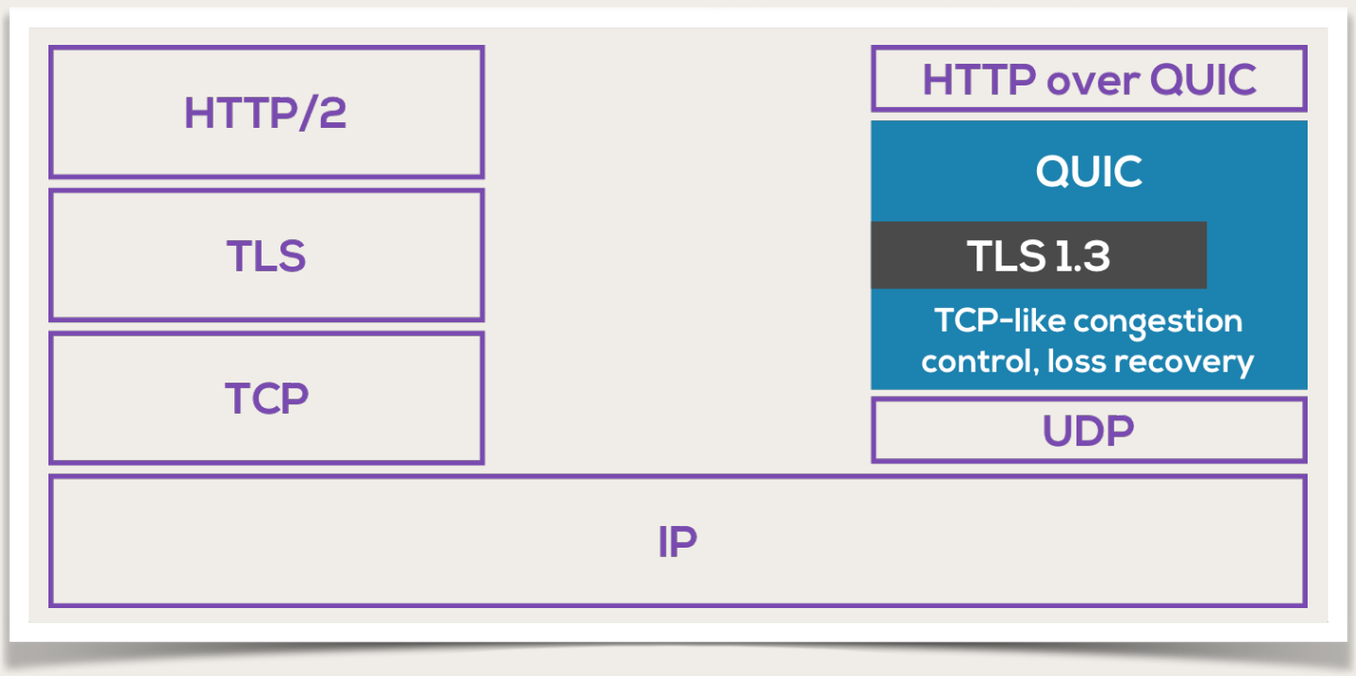


Fig 3.2 - Architecture de QUIC

3.3 - QUIC

- **QUIC, *Quick UDP Internet Connections***
  - De nombreuses améliorations :
    - L'objectif principal de QUIC est d'améliorer la performance perçue des applications Web orientées connexion qui utilisent actuellement TCP.
    - QUIC prend en charge un ensemble de **connexions multiplexées** entre deux points de terminaison à travers UDP
    - Connexion rapide et une **latence réduite** :
      - Lors de sa première connexion, le client envoie une requête initiale avec toutes les informations nécessaires. À ce paquet, le serveur renvoie un autre paquet (Certificats, etc) ainsi qu'un identifiant qui sera associé à ce client. Cela implique 1 RTT, *round trip time*.
      - À sa prochaine connexion, le client n'aura pas besoin de refaire cette requête tant que le serveur n'aura pas changé de cycle de vie. On arrive à 0 RTT.
  - Une estimation de la bande passante dans chaque direction permet d'éviter la congestion.

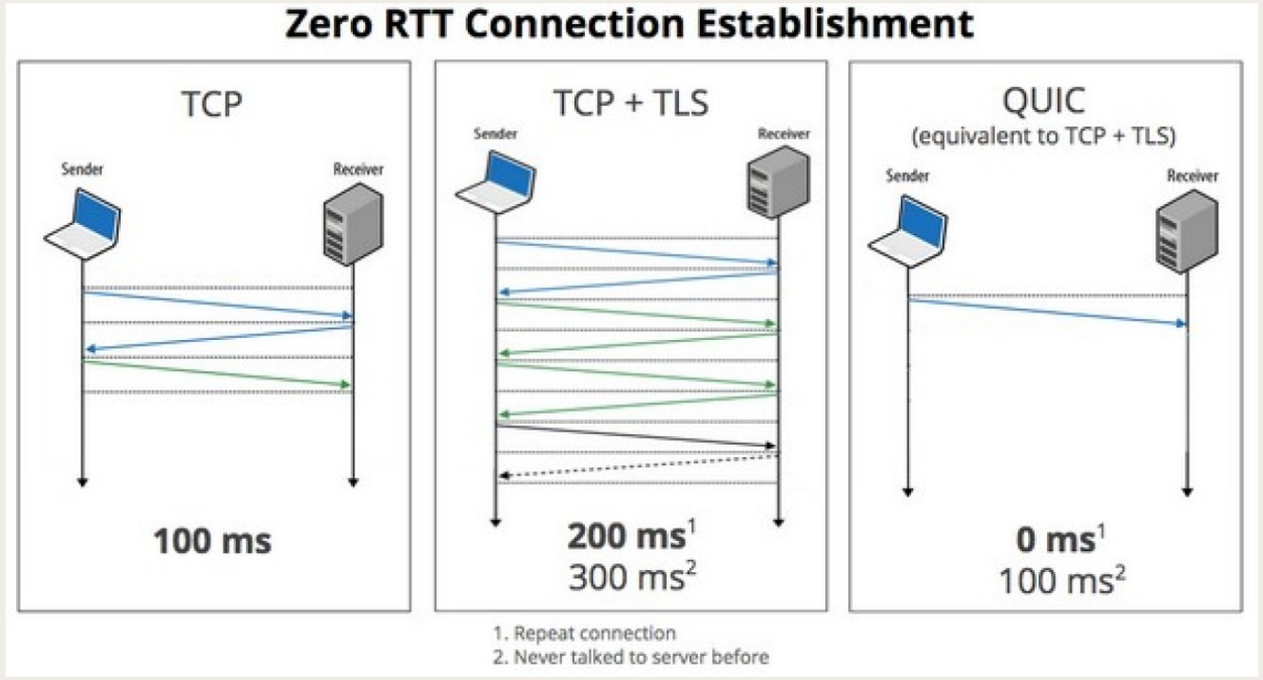


Fig 3.3 - Établissement de connexion à 0 RTT

3 - Protocoles de transport

3.3 - QUIC

- **QUIC, *Quick UDP Internet Connections***

  - Travaux autour de QUIC :
    - HTTP/3, *Hypertext Transfer Protocol Version 3*
      - https://datatracker.ietf.org/doc/rfc9114/
    - QUIC v2
      - https://datatracker.ietf.org/doc/rfc9369/
    - MASQUE, *Multiplexed Application Substrate over QUIC Encryption*
      - https://datatracker.ietf.org/wg/masque/about/
  - À voir :
    - HTTP/3 explained - Français - Présentation du livre en ligne
    - https://www.bortzmeyer.org/9114.html
    - Blog Stéphane Bortzmeyer: Le protocole QUIC désormais normalisé
    - QUIC, a multiplexed stream transport over UDP - The Chromium Projects
    - https://air.imag.fr/index.php/Quick_UDP_Internet_Connection_(QUIC) - Polytech Grenoble

3.4 - MPTCP

- **MPTCP, *MultiPath TCP***
  - Protocole permettant d'agréger plusieurs flux TCP.
  - Norme expérimentale et proposition de standard de l'IETF, *Internet Engineering Task Force*.
  - Le **multiplexage inverse** permet d'atteindre un débit global quasi égal à la somme des débits des flux TCP.
  - Multipath TCP est par exemple utile dans le contexte des réseaux sans fil, pour profiter simultanément d'une connexion Wi-Fi et d'un réseau de téléphonie mobile.

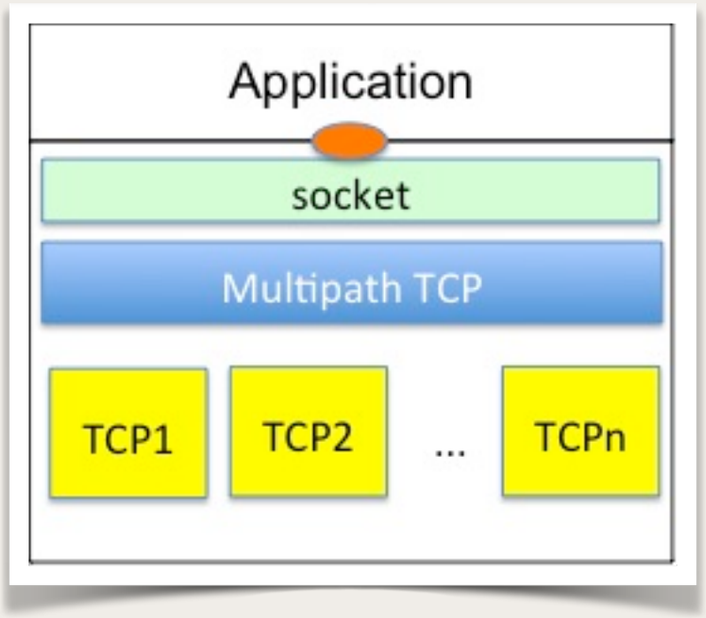


Fig 3.4 - Architecture de MPTCP

- MPTCP peut être implémenté :
  - Dans le noyau Linux : http://www.multipath-tcp.org
  - Pour FreeBSD : www.freebsdfoundation.org/project/multipath-tcp-for-freebsd/
  - À partir de Apple iOS 7 ou Mac OS X 10.10
  - OverTheBox d'OVH Télécom
  - Freebox Delta

3 - Protocoles de transport

3.5 - Sockets

- **Introduction**
  - L'API Sockets a été proposée en 1986, avec Unix BSD, puis adapté aux Linux et à Windows.
  - Cette API propose un mécanisme de communication de bas niveau, en utilisant directement un protocole de transport comme TCP ou UDP.
  - Pour réaliser un service réparti, on peut donc choisir :
    - **Les sockets** (communication entre processus, en bas niveau)
    - **RPC**, *Remote Procedure Call* (qui utilise Socket)
    - **Appel de méthode à distance** (ex.: Java RMI)
    - **Middleware intégré** (Corba, EJB, .Net, Web services...)
- Un socket représente une prise par laquelle une application peut envoyer et recevoir des données. On parle d'interfaces de connexion.
- Le principe de fonctionnement des Sockets se base sur trois phases :
  1. Le serveur crée un « socket serveur », associé à un port ; il se met en attente ;
  2. Un client demande une connexion à la socket serveur, qui crée alors un « socket service client » où se connecte effectivement le client.
  3. Client et serveur communiquent par ces sockets, et le socket serveur peut accepter de nouvelles connexions.

3 - Protocoles de transport

3.5 - Sockets

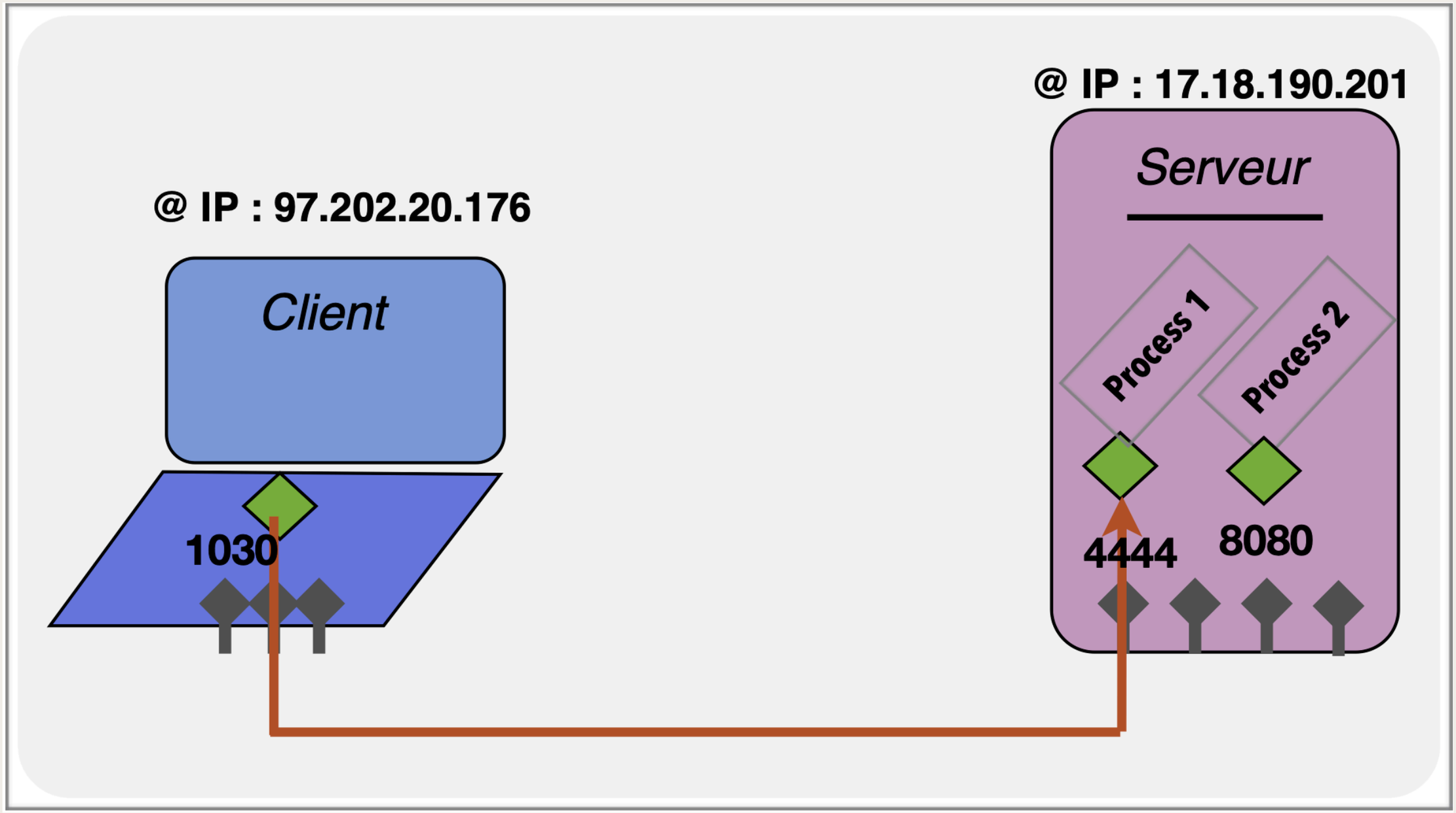
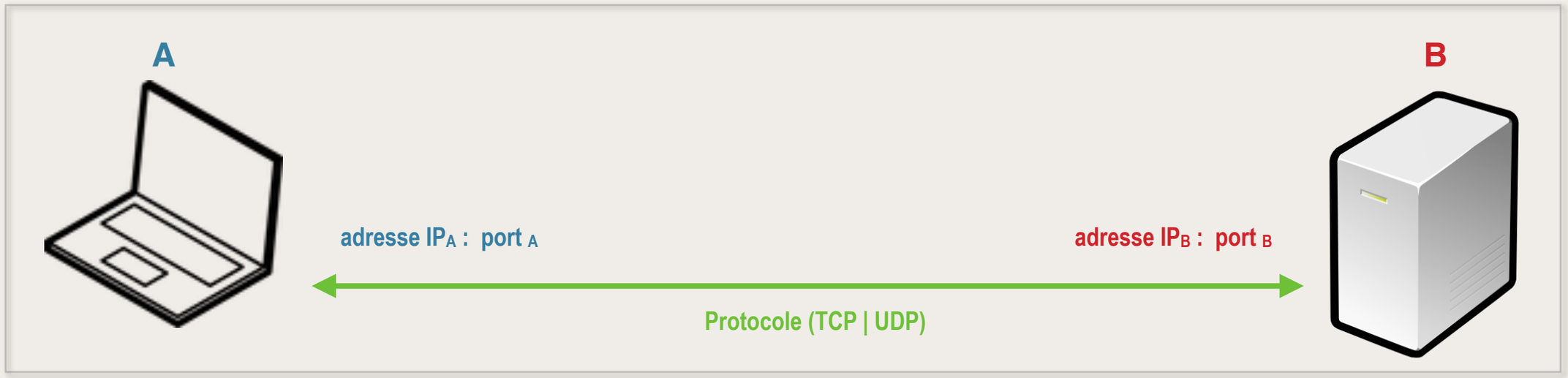


Fig 3.5 - Connexion via sockets

3.5 - Sockets

- **Rappel sur la notion de Sockets**
  - **Les connexions TCP :**
    - Elles sont bidirectionnelles, en mode point à point entre deux adresses de sockets
  - **Un socket est définie par l'association de :**
    - Protocole (UDP ou TCP)
    - Adresse IP de la source
    - Port de la source
    - Adresse IP de destination
    - Port de destination



3.5 - Sockets

- **Les principaux modes**
  - **Un socket stream**, en mode connecté, utilise TCP :
    - Ouverture d'une connexion
    - Utilisation de cette connexion pour une suite d'échanges bidirectionnels (le serveur préserve son état entre deux requêtes)
    - Libération de la connexion
    - On bénéficie alors des garanties de TCP : ordre, contrôle de flux et fiabilité.
    - Ce mode est adapté aux échanges d'une certaine durée (avec plusieurs messages échangés).
  - **Un socket datagramme**, en mode non connecté, utilise UDP :
    - Indépendance des requêtes ;
    - Pas de liaison permanente => indication d'une adresse lors de chaque requête ;
    - Pas de garanties de remise (mode datagramme)
    - Échanges brefs
    - On bénéficie alors de la simplicité et de l'efficacité d'UDP.
  - On ne détaillera pas ici **les sockets raw**.

3.5 - Socket

- **Programme de sockets en Java**
  - Nous trouverons notre bonheur dans le paquetage **java.net**. Pour utiliser, par exemple, des sockets stream :
  - **Coté serveur :**
    - On utilise un objet `ServerSocket`, avec le numéro de port à écouter ;
    - La méthode `accept()` écoute le port et se mets en attente ;
    - Lorsqu'un client se connecte, `accept()` accepte la connexion et retourne un objet `Socket` utilisé alors par le serveur, dans un nouveau thread, pour gérer la communication.
    - Le serveur appelle de nouveau `accept()` pour attendre une nouvelle connexion.
  - **Coté client :**
    - `Socket` implémente un socket dédiée à une communication basée sur un flux ;
    - Une fois celle-ci créée, les méthodes `getInputStream()` et `getOutputStream()` retournent les flux d'entrées/sorties, de façon très similaire aux entrées/sorties d'un fichier sur disque.
  - java.net contient également les classes :
    - `URL`
    - `URLConnection`
    - `DatagramSocket`
    - `DatagramPacket`
  - Voir : Package java.net (Standard Edition 7) et package java.net (Version 25)

4 - Architectures client-serveur

4.1 - Eléments d'architecture

- **Vu d'un utilisateur**
  - Le système est réduit à un espace client.
  - À travers une interface utilisateur (UI, *User Interface*), il utilise des applications qui manipulent des données.

- **Les applications se répartissent en fait coté client et coté serveur.**
  - Les données sont stockées dans des bases de données :
    - non relationnelles, dans les anciens mainframes
    - relationnelles au début des années 80, grâce aux **SGBDR** (IBM DB2, Oracle Database, Microsoft SQL Server, PostgreSQL, MySQL...)
    - orientées objet, avec les **SGBDO** (évolution de SGBDR ou Objectivity, ObjectStore…)
    - NoSQL (Not only SQL), comme MongoDB, Cassandra, etc.
  - L'espace serveur est donc composé des applicatifs permettant l'accès aux données et des données stockées elles-mêmes.
  - Cet espace peut inclure un seul ou plusieurs serveurs physiques ou virtuels.
  - Les serveurs sont **fournisseurs de services** aux clients qui les consomment.

4 - Architectures client-serveur

4.1 - Eléments d'architecture

- **Middleware**
  - Le **middleware** est typiquement le ciment faisant tenir l'édifice : c'est la partie logicielle qui relie l'interface utilisateur (UI), applications et données, entre client et serveur ou entre applications réparties.
  - Le **middleware** :
    - C'est la barre de l'abréviation **C/S** ;
    - se traduit par **intergiciel** ou par **logiciel médiateur** ;
    - est un ensemble de logiciels réutilisables faisant la **médiation entre les applications et le réseau**.
  - Un middleware est du logiciel qui permet à différentes applications d'échanger et d'interopérer.
  - Les **composants logiciels** du middleware assurent **la communication** entre les applications quels que soient les ordinateurs impliqués et quelles que soient les caractéristiques matérielles et logicielles des réseaux informatiques, des protocoles réseau, des systèmes d'exploitation impliqués.

4 - Architectures client-serveur

4.2 - Critères de comparaison

- **Type de données véhiculées**
  - Une information de présentation (que le client affiche simplement)
  - Données extraites via SQL (le client doit déterminer comment afficher ces données)
  - Données résultantes d'appels de procédures à distance
  - Données sous forme d'un flux d'octets, issues de la sérialisation d'objets
- **Type de dialogue client-serveur**
  - En mode connecté (pendant une session)
  - En mode non connecté (ex. du protocole HTTP)
- **Structure et localisation des applicatifs**
  - Une centralisation des applicatifs facilite la maintenance, le déploiement et la gestion.
    - Localisation principalement sur le client => **client lourd** ;
    - Localisation principalement sur le serveur => **client léger** ;
    - répartition des applications entre clients et serveurs.

4 - Architectures client-serveur

4.3 - Type d'architecture

Types d'architectures client-serveur

						Architecture	Client	Serveur		Notes
1960						Mainframe	Léger Simple terminal	Lourd Applications et données		
1970										
1980										
						Client-serveur de données	Applications et interfaces graphiques	Données		SGBDR
1990						Client-serveur distribué	Applications	Serveur d'application	Serveur de données	Architecture à 3 niveaux
						Client-serveur Web	Léger Navigateur web	Serveur web (+serveur d'application)	Serveur de données ou autre serveur	Architecture à 2 ou 3 niveaux
2000										
						Web 2	Applic. coté client (applets, js)	Serveur web (+serveur d'application)	Serveur de données ou autre serveur	Architecture à 2 ou n niveaux
2010										
2020						IoT ; M2M	*Objet*	*Cloud*		
2030 ?										

Fig 4.1 - Types d'architectures client-serveur

4 - Architectures client-serveur

4.3 - Type d'architecture

- **Client-serveur à client passif**
  - Client : simple terminal
  - Serveur : ordinateur central ou **mainframe**.

- **Client-serveur de données**
  - Client lourd (logique applicative, interface graphique)
  - Serveur de données
  - Communication via SQL, *Structured Query Language*

- **Client-serveur distribué**
  - **Architecture à trois niveaux**, avec un serveur tiers, à savoir **le serveur d'application**
    - Les traitements sont appelés avec un protocole **RPC**, *Remote Procedure Call* (Appel de procédure à distance).

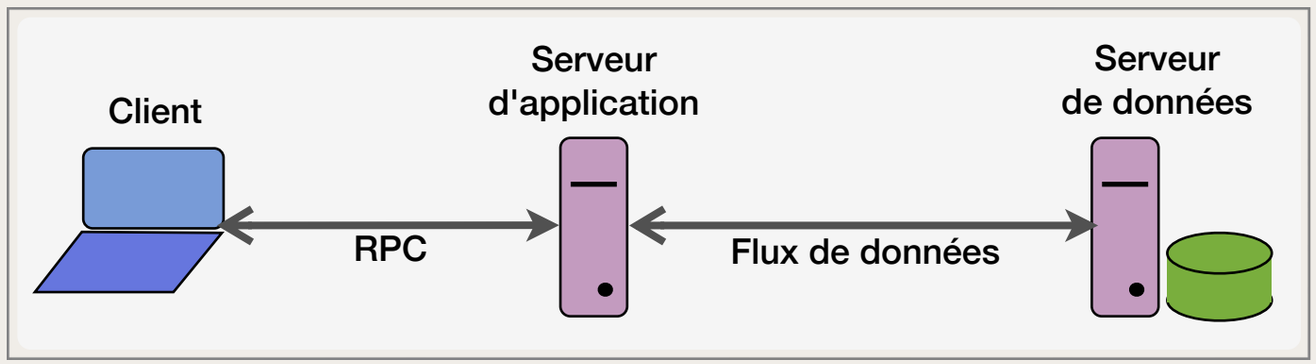


Fig 4.2 - Architectures à 3 niveaux

4 - Architectures client-serveur

4.3 - Type d'architecture

- **Client-serveur à objets distribués**
  - **Corba**, *Common Object Request Broker Architecture*.
    - Le middleware de Corba est un **ORB,** *Object Request Broker*, soit un courtier de requêtes objet ;
  - RMI, *Remote method invocation*, API Java incluse dans Java SE (standart edition) et souvent utilisée avec l'architecture de composants logiciels EJB, *Enterprise JavaBeans*.
- **Architecture client-serveur web**
  - **Client léger** : le navigateur web n'est qu'un simple afficheur de pages HTML. Le terminal est cette fois graphique.
  - Serveur web + serveur d'application + serveur de données.
  - Dialogue client-serveur via HTTP, en mode non connecté

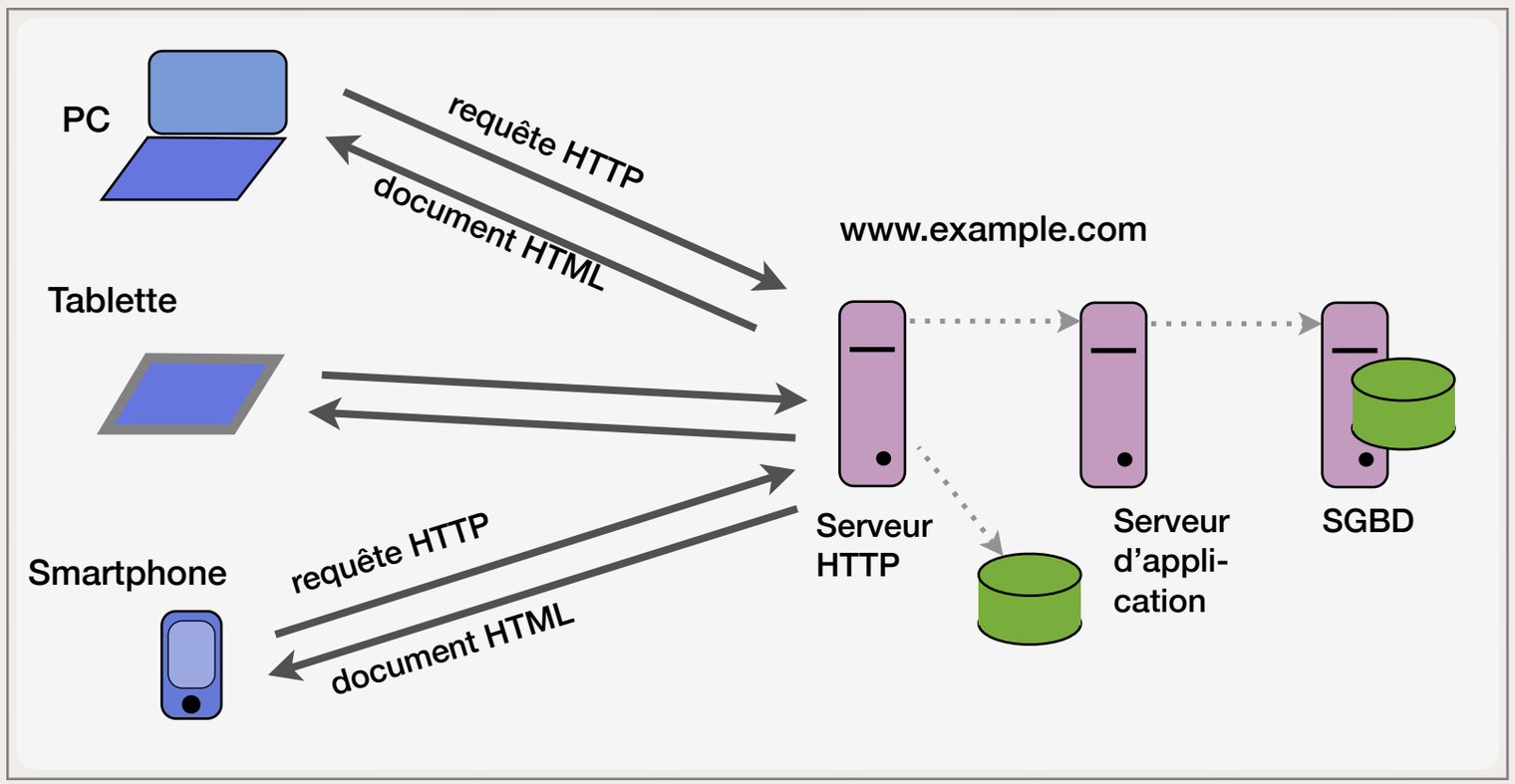


Fig 4.3 - Architectures client-serveur web

4 - Architectures client-serveur

4.3 - Type d'architecture

- **Architecture à code mobile de type client-serveur de données ou distribué**
  - Client plus lourd : des applications y sont téléchargées.
    - L'application cliente est téléchargée ;
    - Le client se connecte à un serveur d'application (dialogue en mode connecté) ;
    - Par ex. via IIOP/Corba. (Internet Inter-ORB Protocol).
  - Serveur web + serveur d'application + serveur de données.

4 - Architectures client-serveur

4.3 - Type d'architecture

- **CDN, *Content delivery network***
  - Réseau de serveurs sur Internet qui coopèrent pour rendre accessible à des utilisateurs du contenu ou des données.
  - Le réseau est constitué :
    - D'un serveur d'origine, d'où les contenus sont répliqués vers …
    - … des serveurs périphériques (*proxy servers* ou *edge servers*), déployés dans différentes zones géographiques ;
    - un mécanisme de routage pour qu'une requête utilisateur puisse être servi par le serveur périphérique le plus proche.

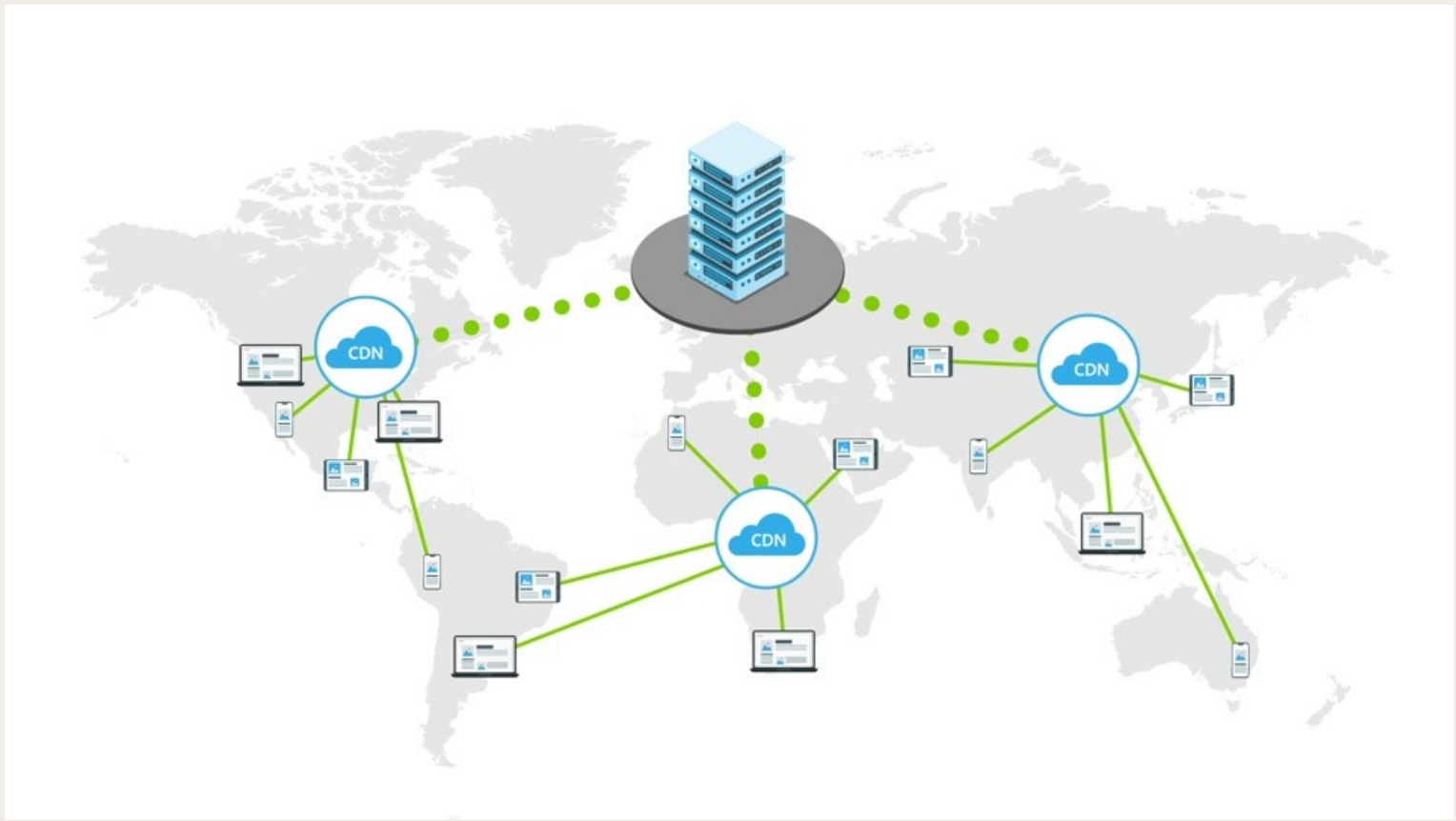


Fig 4.4 - Content delivery network

4 - Architectures client-serveur

4.3 - Type d'architecture

- **Architecture peer-to-peer (P2P)**
  - Une architecture **pair à pair** (***peer-to-peer*** ou P2P en anglais) est un environnement client–serveur où chaque programme connecté est susceptible de jouer **tour à tour** ou **à la fois** le rôle de client et celui de serveur.
  - Les composants d'un système P2P sont appelés **nœuds**, **pairs** ou **utilisateurs**.
  - Certains systèmes sont :
    - partiellement centralisés ; un serveur intermédiaire gère une partie des échanges
    - totalement décentralisés ; sans infrastructure particulière

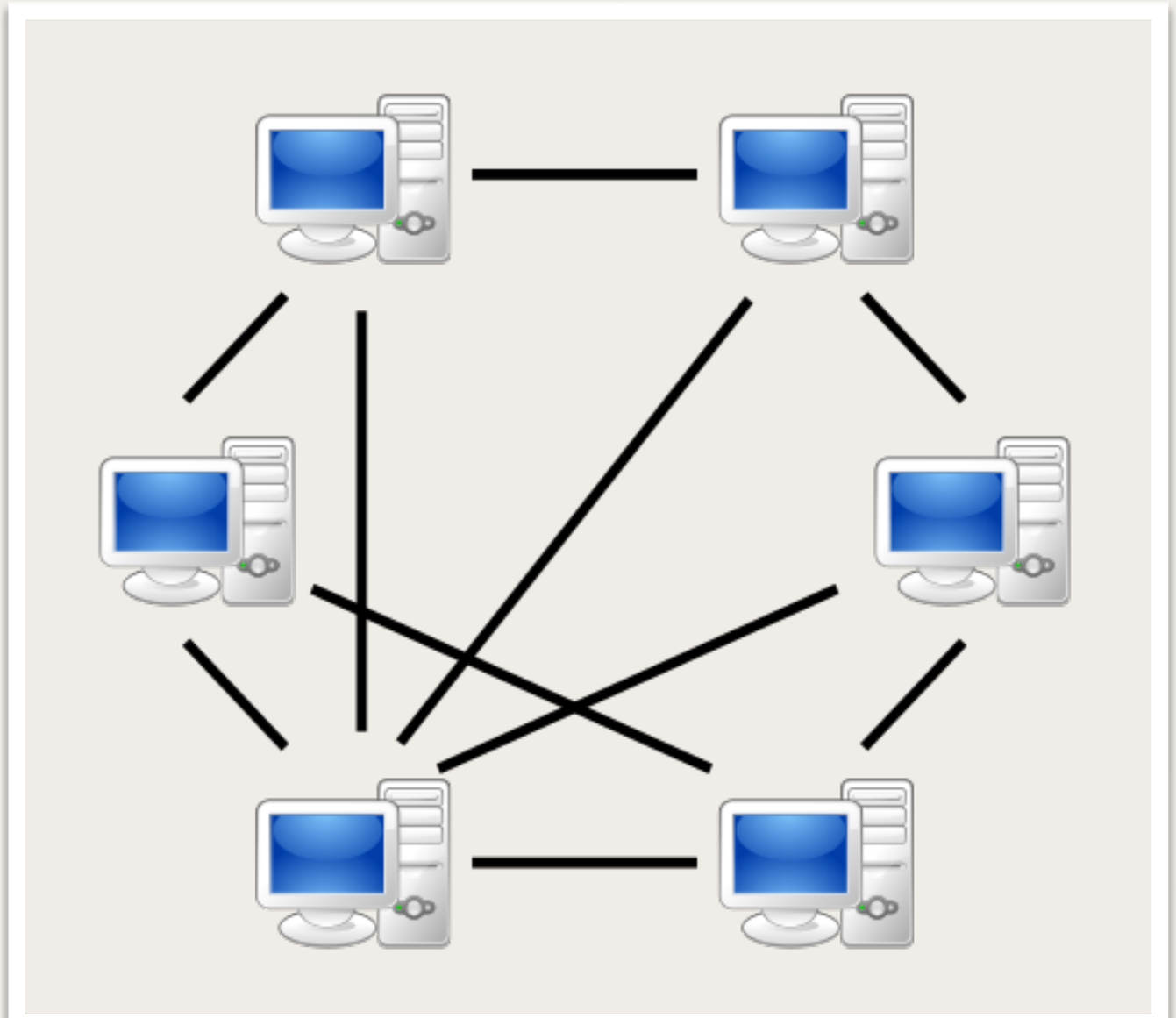


Fig 4.5 - Architectures peer-to-peer

- Un tel réseau est adapté au partage d'objets (fichiers, flux multimédia, calcul réparti, etc.)

- **Registre distribué, *distributed ledger* ou *shared ledger***
  - **DLT**, *Distributed Ledger Technology*
    - Infrastructure numérique dans laquelle les données sont stockées de manière décentralisée.
  - **Registre** simultanément enregistré et synchronisé sur un réseau informatique, qui évolue par l'addition de nouvelles informations préalablement validées par l'entièreté du réseau et destinées à ne jamais être modifiées ou supprimées.
    - La mise à jour d'un registre distribué se répercute sur l'ensemble du réseau. En conséquence, chaque membre possède en permanence la dernière version du registre.
  - Le fonctionnement d'un système DLT nécessite un réseau **pair-à-pair** et un **algorithme de consensus**.
  - ***Blockchain,*** système de la chaîne de blocs, qui peut être public ou privé, est une des formes de registre distribué.

Chaîne de blocs, *blockchain*

- **Base de données** distribuée et sécurisée, dans laquelle sont stockées chronologiquement, sous forme de blocs liés les uns aux autres, les transactions successives effectuées entre ses utilisateurs depuis sa création.
  - Chaque bloc est similaire à un dossier scellé contenant des données ;
  - Tous les blocs sont reliés les uns aux autres ;
  - Les blocs sont ajoutés les uns à la suite des autres sans qu'il soit possible de changer leur ordre d'ajout.
  - Une chaîne de blocs est une version unique de la vérité rendue possible par un registre immuable, horodaté et sécurisé, dont des copies sont détenues par plusieurs parties.
  - La sécurisation d'un blockchain se fait grâce à certains utilisateurs, appelés **mineurs,** dont le rôle est défini dans la technologie.
- Registre sécurisé et inviolable avec des transactions horodatées, réparties entre un certain nombre d'entités, sans autorité centrale.

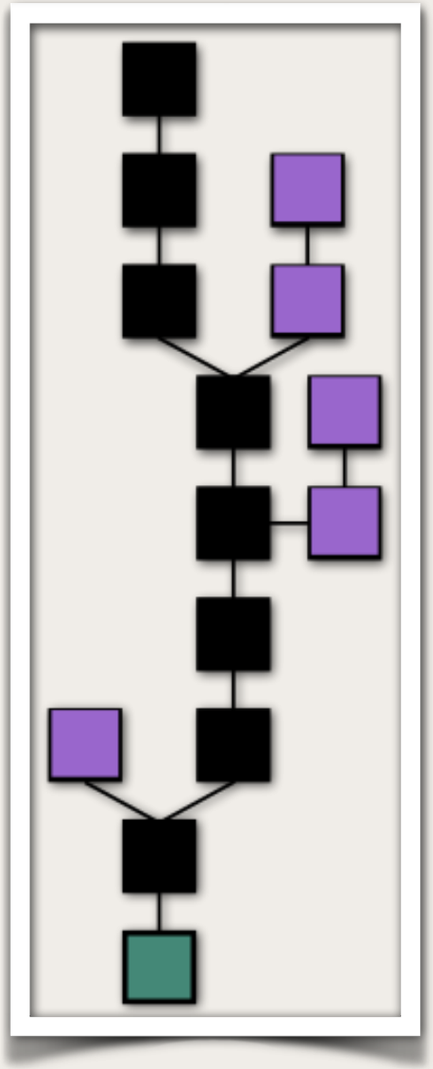


Fig 4.6 - Représentation d'une chaîne de blocs

4.4 - Registre distribué & blockchain

- **Chaîne de blocs,** ***blockchain*** *(suite…)*

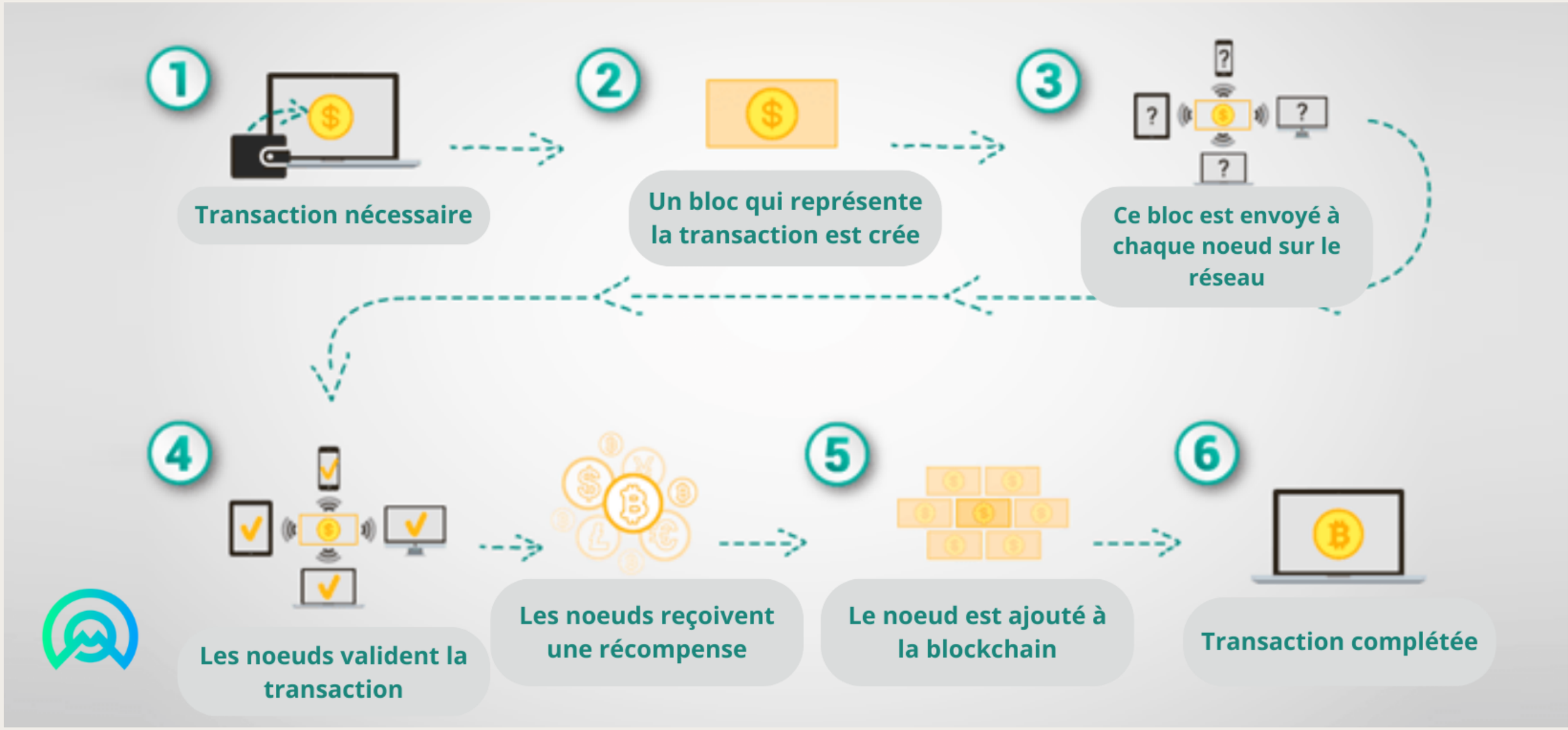


Fig 4.7 - Fonctionnement d'une blockchain (https://moneyradar.org/lexique/blockchain/)

- **Chaîne de blocs, *blockchain*** (suite…)
  - Applications :
    - Crypto-monnaies : Bitcoin, Ether, Monero…
    - Contrats intelligents (*Smart contracts*), permettant d'échanger toutes sortes de biens ou de services ;
    - Traçabilité de marchandises
    - Santé : partage sécurisé de dossiers médicaux et meilleure coordination entre acteurs.

  - Voir :
    - https://www.economie.gouv.fr/entreprises/blockchain-definition-avantage-utilisation-application
    - https://medium.com/@godefroy.galas/analyse-et-comparaison-des-m%C3%A9canismes-de-consensus-dans-la-blockchain-f91aee511ea3

4 - Architectures client-serveur

4.5 - Rôles d'un serveur et serveur virtuel

- **Services et fonctions**
  - Servir de multiples clients
    - Le serveur attend les requêtes de clients (soit pendant une session dédiée à un client donné, soit avec une session réutilisable (cas de gestion de pool dynamique de sessions).
    - Quand une requête arrive
      - le serveur doit la traiter rapidement, dans une tâche indépendante (thread) ;
      - Il traite les différentes requêtes de façon simultanée, tout en protégeant l'intégrité des données et ressources partagées ;
      - il doit répondre rapidement au client, même pendant les heures de pointe.
  - Le serveur peut offrir différents niveaux de priorités à ses clients.
  - Il doit pouvoir lancer des tâches de fond, sans liens directs avec le but principal (impressions, téléchargements, sauvegardes...)

- **VPS, *Virtual Private Server*, Serveur dédié virtuel**
  - Un serveur physique est partitionné, en utilisant des techniques de virtualisation, en plusieurs **serveurs virtuels** indépendants qui ont chacun les caractéristiques d'un serveur dédié.
  - VPS, *Virtual Private Server* est souvent confondu avec VDS, *Virtual Dedicated Server*.
  - Un serveur virtuel est une instance logique isolée au sein d'un serveur physique d'hébergement.
  - C'est une forme de machine virtuelle (VM, *Virtual Machine*). Il existe plusieurs technologies de virtualisation pour créer ces serveurs virtuels :
    1. Virtualisation basée sur les conteneurs :
       - Utilise l'isolation au niveau du noyau du système d'exploitation hôte.
       - Exemples : OpenVZ, LXC (Linux Containers) ; Virtuozzo est une solution commerciale basée sur la technologie OpenVZ
       - Avantages : Efficace en termes de ressources, démarrage rapide
       - Limitations : Généralement restreint aux systèmes d'exploitation Linux, partage le noyau de l'hôte

- **VPS, *Virtual Private Server*, Serveur dédié virtuel** (suite…)

2. Virtualisation complète :
   - Émule entièrement le matériel, permettant l'exécution de systèmes d'exploitation invités non modifiés
   - Exemples : VMware ESXi, KVM (Kernel-based Virtual Machine)
   - Avantages : Supporte une large gamme de systèmes d'exploitation invités
   - Inconvénients : Peut avoir un impact plus important sur les performances

3. Paravirtualisation :
   - Les systèmes d'exploitation invités sont modifiés pour fonctionner plus efficacement avec l'hyperviseur
   - Exemple : Xen (dans ses versions antérieures)
   - Avantages : Performances améliorées par rapport à la virtualisation complète
   - Inconvénients : Nécessite des systèmes d'exploitation invités modifiés

- **VPS, *Virtual Private Server*, Serveur dédié virtuel (suite…)**
  - Avantages :
    - Gestion de l'OS et des logiciels (serveurs FTP, mail, web, etc.). Flexibilité d'un serveur dédié ;
    - Coût inférieur à un serveur dédié.
  - Inconvénients :
    - Performances moindres qu'un serveur dédié ;
    - Coût supérieur à un hébergement mutualisé.
  - Voir :
    - https://www.ovhcloud.com/fr/learn/what-is-virtual-server/
    - https://www.ovhcloud.com/fr/vps/
    - https://www.ionos.fr/serveurs/vps

4 - Architectures client-serveur

4.6 - Cloud computing : l'informatique dans les nuages

- **Introduction**

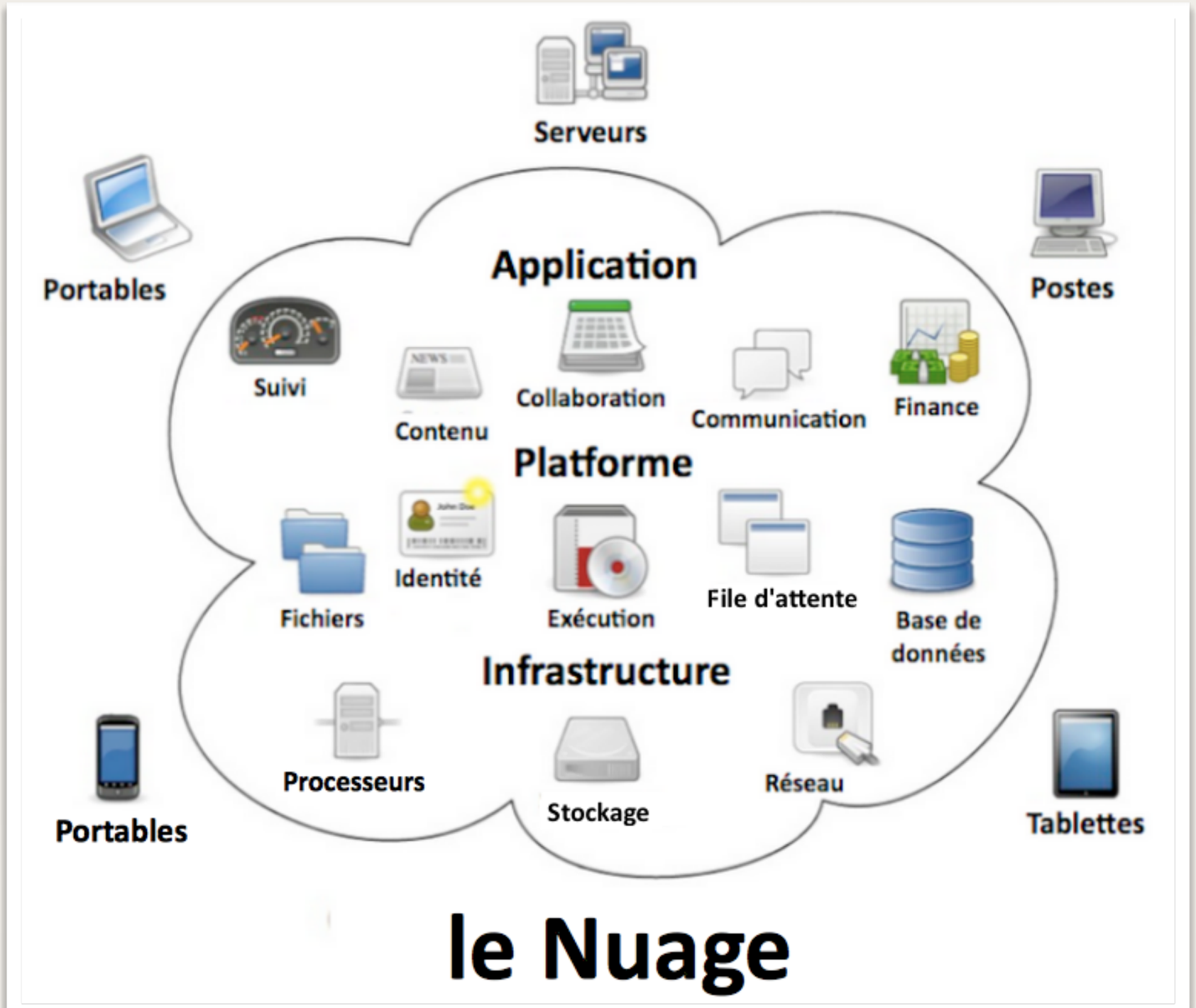


Fig 4.8 - Le nuage - the Cloud

By Sam Johnston [CC BY-SA 4.0], via Wikimedia Common

4 - Architectures client-serveur

4.6 - Cloud computing : l'informatique dans les nuages

- **Avènement du Cloud Computing (CC)**
  - Années 1980
    - La virtualisation (hyperviseur)
    - l'infogérance
    - l'externalisation
  - Dernière décennie
    - Généralisation d'internet, des FAI, des hébergeurs
    - développement des réseaux à haut débit
    - la location d'application
    - le paiement à l'usage
    - la quête de mobilité...

- **Cloud Computing ≈ Virtualization + pay as you go + self service**

  - **Virtualisation** (*Virtualization*)
    - Partage des ressources
    - Abstraction sur la localisation
    - Élasticité

  - **Pay as you go**
    - Paiement à l'usage ; paiement des ressources effectivement utilisées (processeur, stockage, réseau)

  - **Self service**
    - Allocation de ressources d'exécution via une interface Web, avec effet en quelques minutes.

- **Les formes de cloud computing**
  - les **clouds privés internes**, gérés en interne par une entreprise pour ses besoins.
  - les **clouds privés externes**, dédiés aux besoins propres d'une seule entreprise, mais dont la gestion est externalisée chez un prestataire,
  - les **clouds publics**, gérés par des fournisseurs spécialisés qui louent leur services à de nombreuses entreprises.
  - un **cloud hybride** consiste à associer un ou plusieurs clouds publics à un cloud privé. Un système d'orchestration connecte et ordonnance les ressources de cloud privé avec des ressources de cloud public.

4 - Architectures client-serveur

4.6 - Cloud computing : l'informatique dans les nuages

- **Les modèles de service :**
  - **Software as a Service** (SaaS) (ou logiciel à la demande) : utilisation de services proposés en abonnement ou payés à la demande.
  - **Platform as a Service** (PaaS) : la plateforme est louée par l'entreprise.
  - **Infrastructure as a Service** (IaaS) : l'infrastructure est virtualisée ; le fournisseur Cloud maintient la virtualisation, le matériel serveur, le stockage et les réseaux.
  - **Serverless** : construction et exécution d'applications sans gérer les serveurs sous-jacents.
    - Le fournisseur Cloud gère le provisionnement et la surveillance des serveurs ;
    - Le développeur se concentre sur le code de l'application et le déploiement de ses API.
    - Le terme "s**erverless**" est trompeur car les serveurs sont toujours impliqués. Mais du point de vue du développeur, il n'y a pas de serveurs visibles à gérer.
  - ⚠️ **Data as a Service** (DaaS) : les données sont fournies à un endroit précis.
  - ⚠️ **Desktop as a Service** (DaaS) : bureau virtuel hébergé.
  - **Database as a Service** (DBaaS) : système de base de donnée via le Cloud

4 - Architectures client-serveur

4.6 - Cloud computing : l'informatique dans les nuages

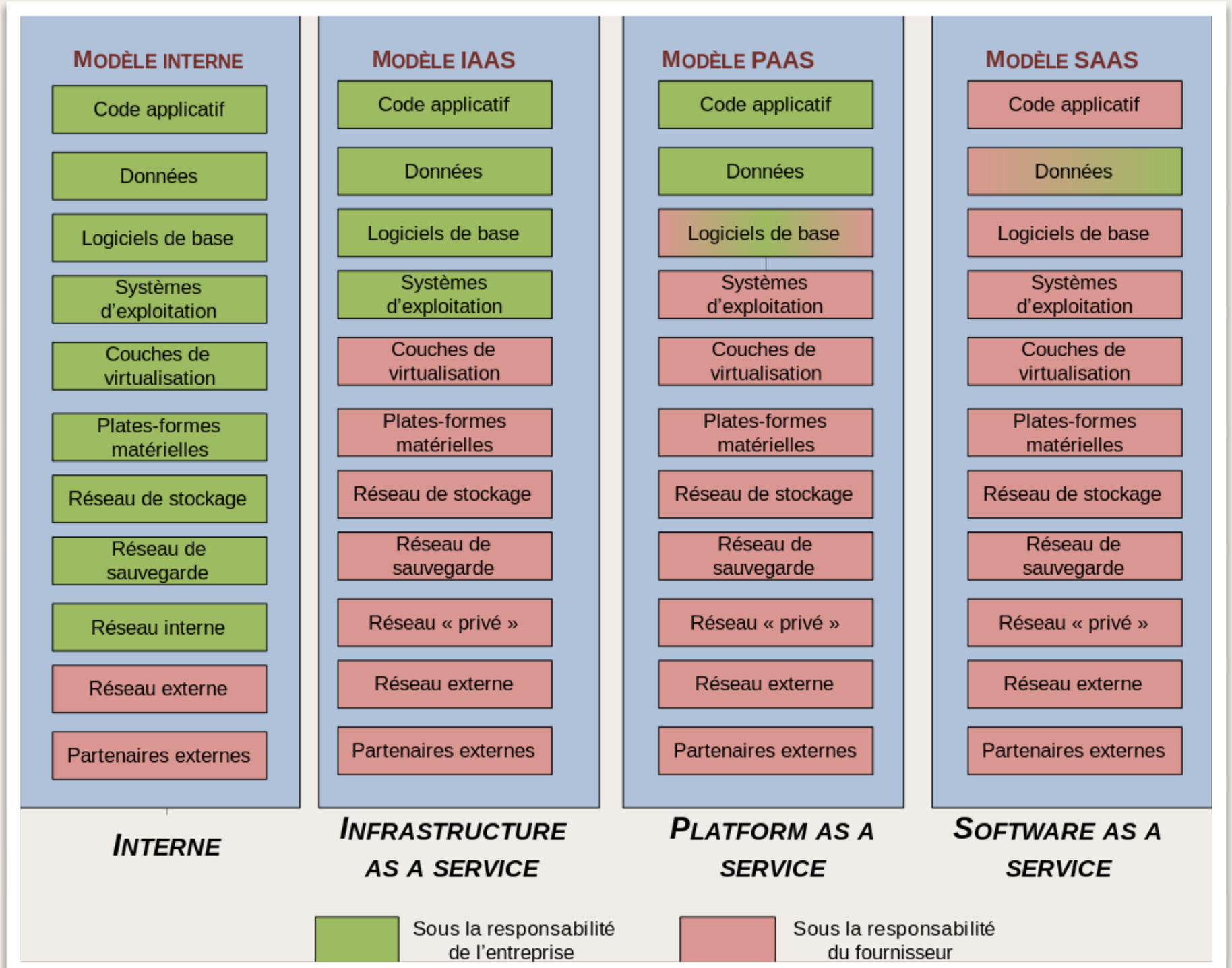


Fig 4.9 -Les modèles de cloud
https://fr.wikipedia.org/wiki/Cloud_computing#/media/File:Cloud_Computing_-_les_différents_modèles_de_service.svg

4 - Architectures client-serveur

4.6 - Cloud computing : l'informatique dans les nuages

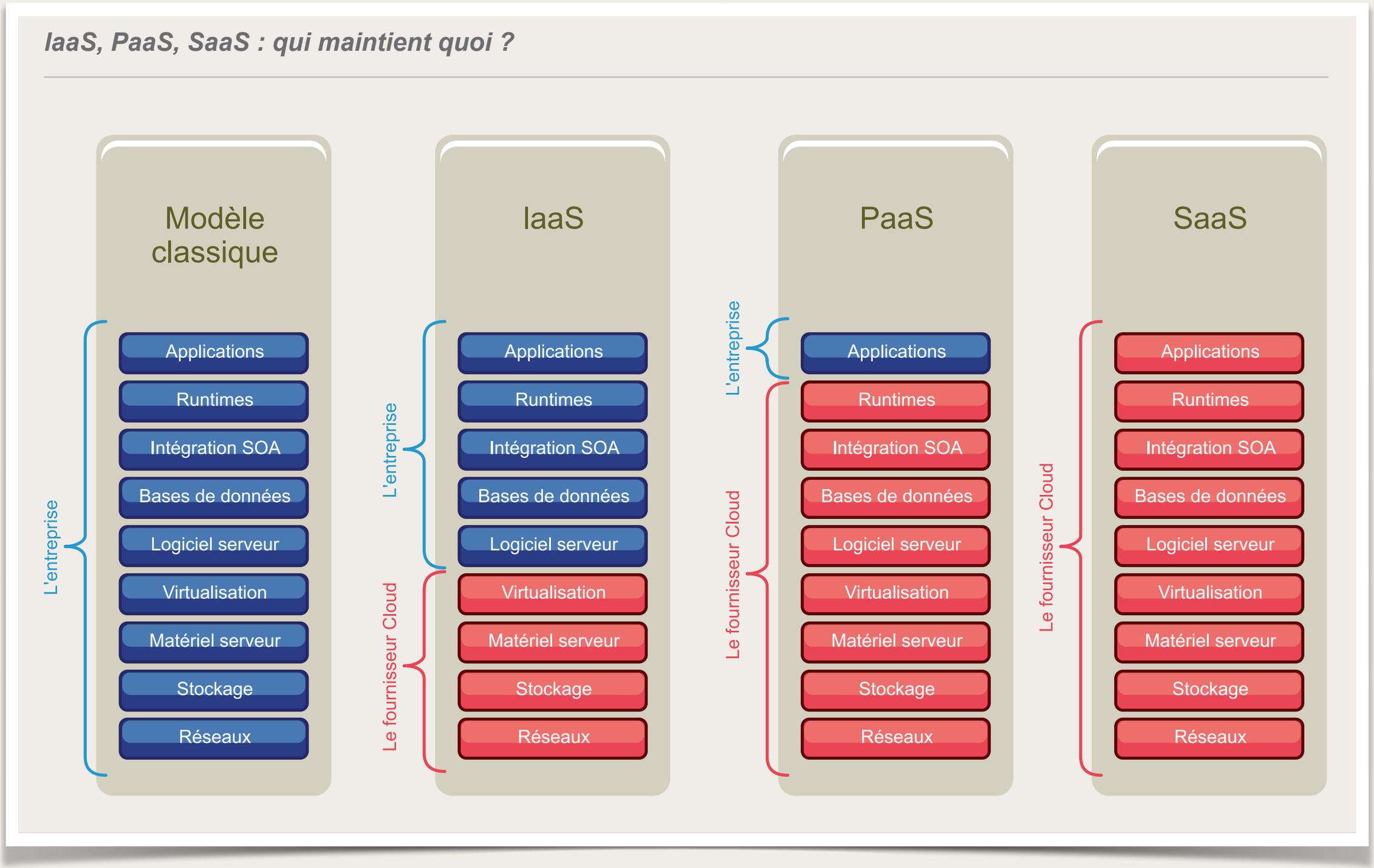


Fig 4.10 -Les modèles de cloud
D'après : Le Livre blanc du cloud computing - Syntec informatique

4 - Architectures client-serveur

4.6 - Cloud computing : l'informatique dans les nuages

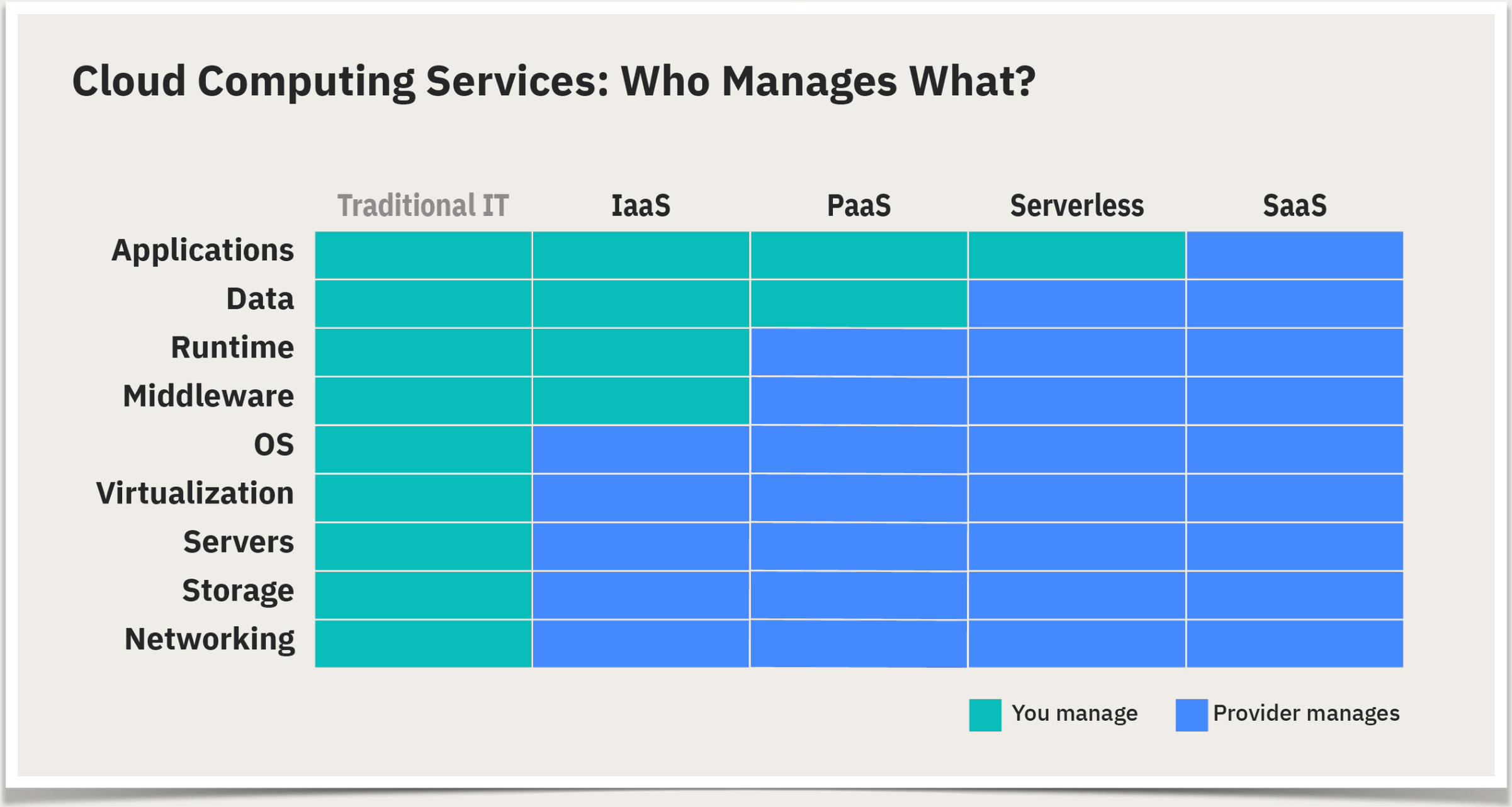


Fig 4.11 -Les modèles de cloud
D'après : What is Cloud Computing? | IBM

4 - Architectures client-serveur

4.6 - Cloud computing : l'informatique dans les nuages

- **Les premières applications Web 2.0 que l'on trouve sur les « nuages » sont :**
  - Webmail et messagerie ;
  - Les outils collaboratifs et de web-conferencing ;
  - Bureautique en ligne : Google Docs / Google Workspace (édition collaborative de documents directement sur les serveurs de Google) ;
  - Réseaux sociaux : Facebook, MySpace, etc.
  - Les environnements de développement et de test ;
  - Le CRM, *Customer Relationship Management* ≈ gestion de la relation client
    - Salesforce est un CRM 100% Web.
  - L'informatique décisionnelle (*Business Intelligence*).

4 - Architectures client-serveur

4.6 - Cloud computing : l'informatique dans les nuages

- **Quelques acteurs et applications**
  - Des systèmes de cloud computing proposent des API pour communiquer
    - Amazon EC2
    - Microsoft Azure
    - Google Cloud et les produits Compute Engine, App Engine, etc.
  - Applications logicielles en ligne
    - Google Workspace (ex G Suite) inclus : Gmail, Agenda, Docs, Drive, Forms, Sites, etc.
    - Microsoft 365
    - HCL Notes and Domino (ex IBM Notes and Domino, ex Lotus)
- IAAS Open source
  - OpenStack, qui a rejoint la Fondation Linux en 2025.
  - Apache CloudStack permet de créer, de gérer et de déployer des infrastructures cloud publiques et privées.

- **Autres acteurs en France…**
  - Orange Business Services (OBS) ; SFR Business Team ; Dassault Systèmes…
  - NUMSPOT, alliance de Docaposte, Dassault Systèmes, Bouygues Telecom et la Banque des Territoires
- **en Europe…**
  - Gaia-X, était un projet d'infrastructure de Cloud européen. Les premières expérimentations était prévues pour 2021.
  - Un cloud numérique souverain européen reste en développement.
    Voir : EU Cloud and AI Development Act.
- **et dans le monde**
  - Citrix
  - Salesforce

4 - Architectures client-serveur

4.7 - La protection du réseau

- **Les principaux aspects de la sécurité des réseaux**
  - Voir le cours SEC105, *Architectures et bonnes pratiques de la sécurité des réseaux, des systèmes, des données et des applications* et RSX112, *Sécurité et réseaux*
- **La protection du réseau**
  - Filtrage par le routeur d'accès :
    - Un routeur d'accès peut assurer un filtrage simple par analyse des adresses IP sources et destination
    - Il agit au niveau 3 (couche réseau), à l'aide de listes d'adresses acceptées ou refusées.
    - **ACL**, *Access Control List*
  - La traduction d'adresse
    - NAT, *Network Address Translation ;*
    - NAPT, *Network Address and Port Translation*
    - C'est un moyen de masquer le plan d'adressage de l'entreprise
    - (et de pallier à la pénurie d'adresses IPv4...)
    - NAPT permet notamment de faire correspondre une seule adresse externe publique visible sur internet à toutes les adresses d'un réseau privé.

4 - Architectures client-serveur

4.7 - La protection du réseau

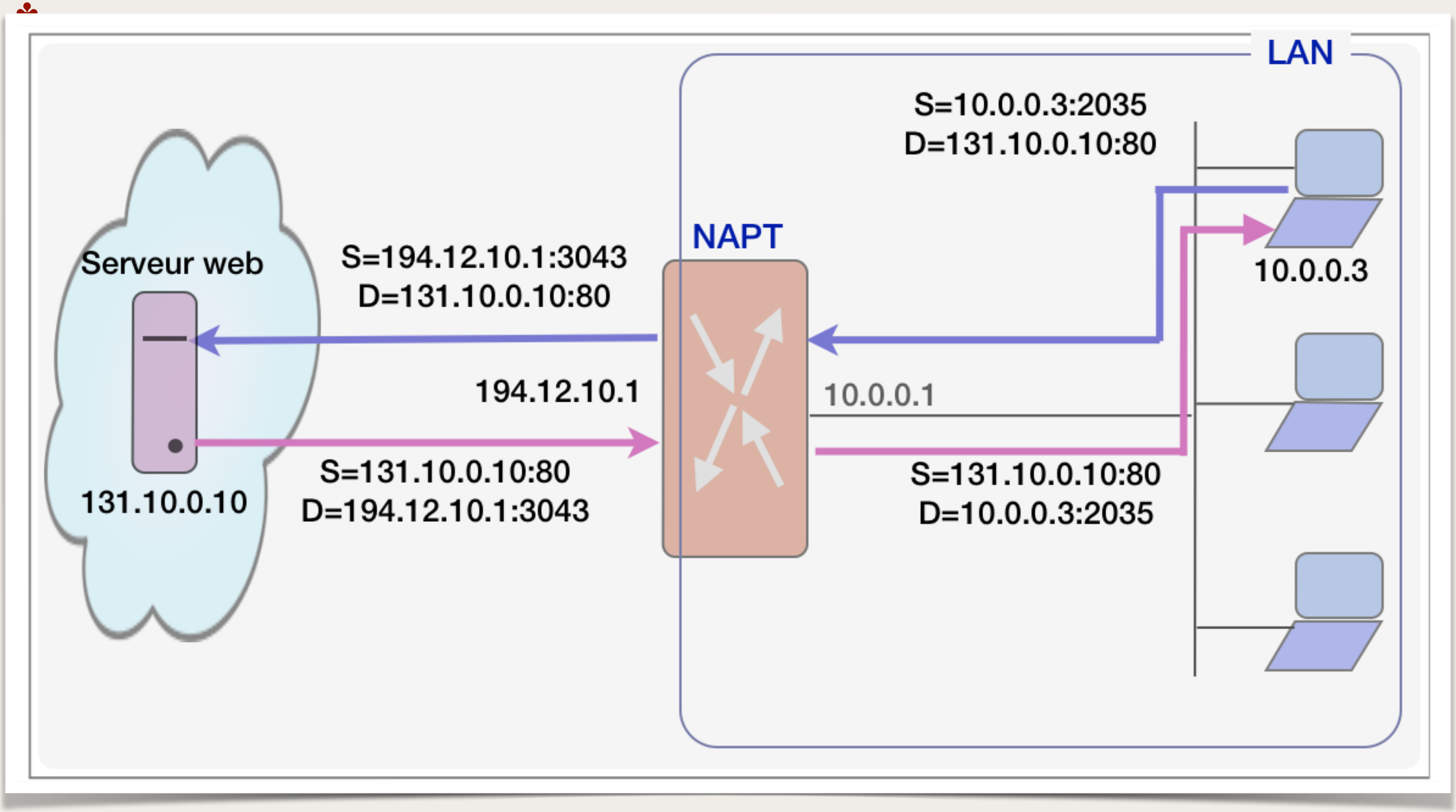


Fig 4.12 -Routeur avec NAPT

4 - Architectures client-serveur

4.7 - La protection du réseau

- Pare-feu ; *Firewall*
  - Un pare-feu (ou coupe-feu) offre un filtrage évolué pour faire respecter la politique de sécurité du réseau.

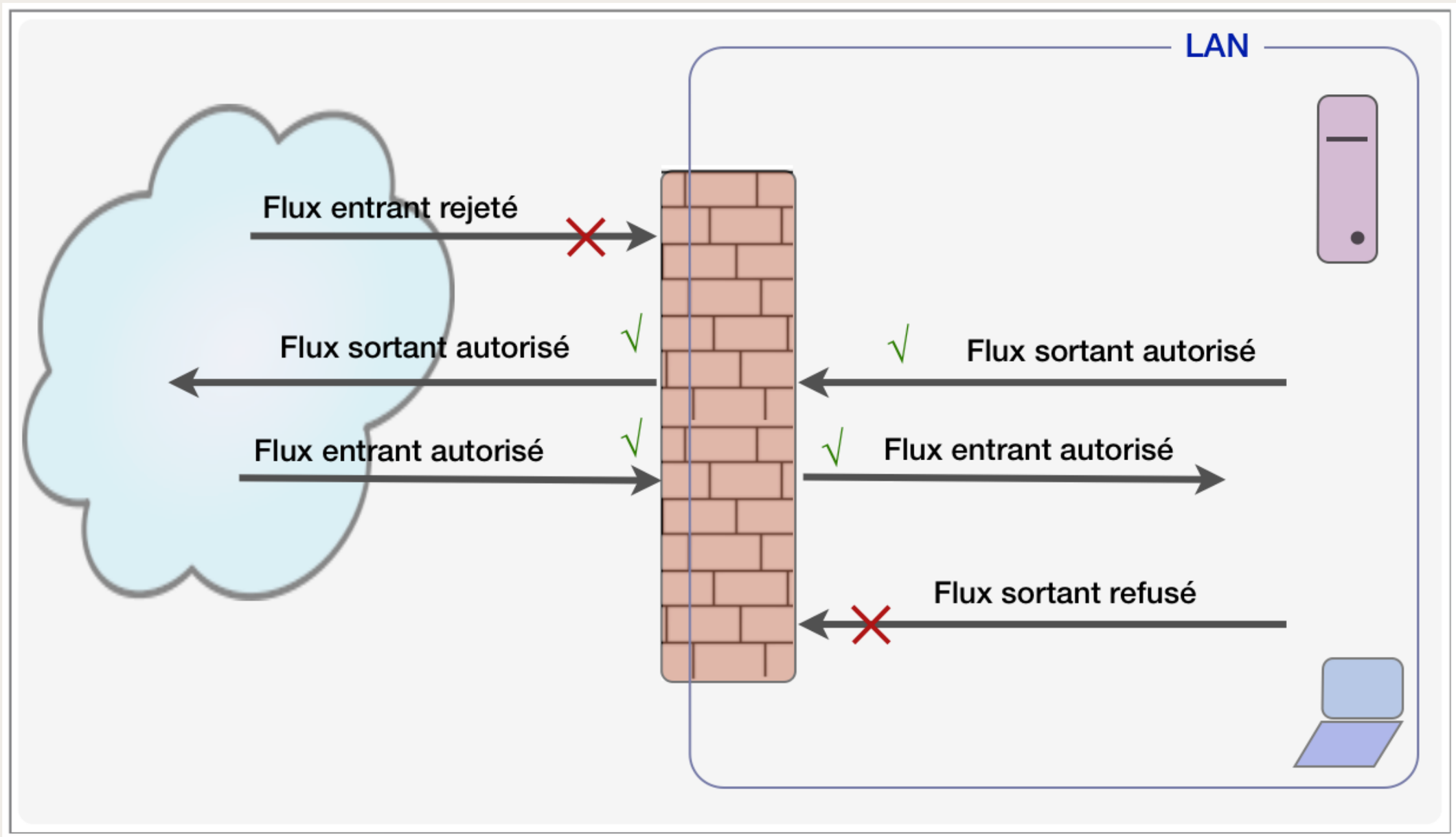


Fig 4.13 - Pare-feu

4 - Architectures client-serveur

4.7 - La protection du réseau

- Pare-feu ; ***Firewall,*** (suite…)
- Chaque paquet reçu est examiné ; il est rejeté ou accepté en fonction de
  - adresse destination
  - port destination
  - adresse source
  - port source
  - protocole transporté (ICMP, UDP, TCP...)
  - la valeur de certains drapeaux d'en-tête
  - etc.
- Un pare-feu peut être configuré en tenant compte d'une DMZ : **DeMilitarized Zone**, soit zone de sécurité.

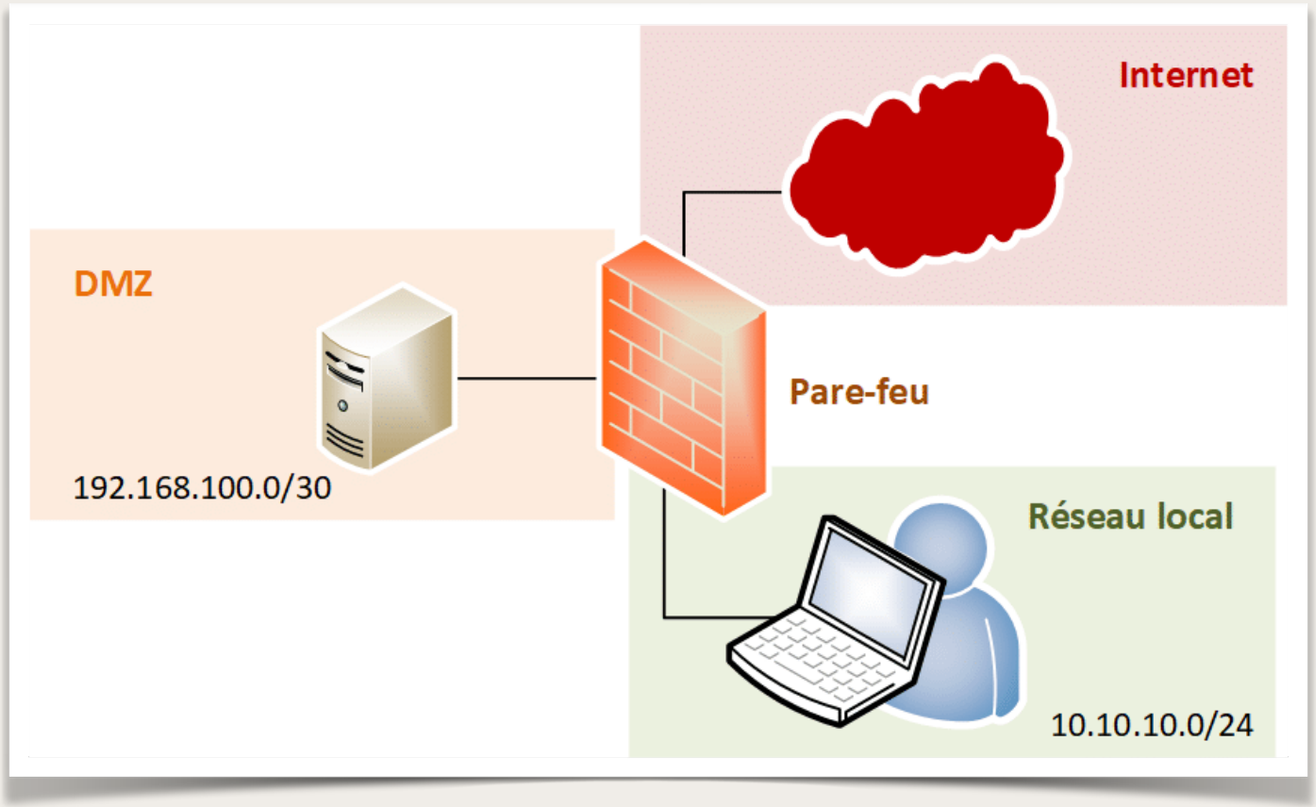


Fig 4.14 -DMZ avec un seul pare-feu
D'après : Informatique : c'est quoi une DMZ ? | IT-Connect

4 - Architectures client-serveur

4.7 - La protection du réseau

- Catégories de pare-feu
  - **Pare-feu sans état** : il intégrait d'anciens routeurs, afin d'examiner les paquets indépendamment les uns des autres, suivant des règles nommées, suivant les constructeurs :
    - **ACL** : Access Control List (Cisco)
    - **Policy** (Juniper)
    - Règles, filtres, etc.
  - **Pare-feu à états** (*stateful firewall*). Suivant le type de transport :
    - TCP : il vérifie que les paquets sont liées à une même connexion TCP et sont conformes aux ACL.
    - UDP : si les ACL autorise un datagramme UDP du fait du quadruplet (ip_src, port_src, ip_dest, port_dest) la réponse (avec le quadruplet inverse) sera également acceptée.
- **Pare-feux applicatifs** ou **passerelle au niveau de l'application** (*Application Layer Gateway* ou *Proxy-Server*) :
  des ensembles de paquets sont décapsulés pour examiner la conformité au niveau applicatif.
  - Un tel pare-feu est composé de deux routeurs, filtrants des paquets (au niveau 3) et d'une passerelle d'application qui permet ce filtre plus conséquent
  - Le filtrage est donc effectué au niveau de chaque service offert
  - Des conversions de protocoles sont alors possibles
  - Certains virus et chevaux de Troie restent malgré tout indécelables.

4 - Architectures client-serveur

4.7 - La protection du réseau

- **VPN, Virtual Private Network ; Réseau privé virtuel**
  - Les types de VPN :
    - *Site-to-site VPN* ; VPN de site-à-site ; une liaison sécurisée entre réseaux physiques.
    - *Remote access VPN* ; un accès sécurisé à distance, pour l'indépendant, l'employé en télétravail. C'est un moyen d'accès au système d'information pour des utilisateurs nomades.
  - Un VPN de site-à-site permet une extension des réseaux locaux tout en préservant la sécurité logique. :
    - Il permet une interconnexion de réseaux locaux via une technique de « tunnel » (*tunneling*).
  - Internet est souvent utilisé comme support de transmission en utilisant un protocole de « tunnellisation », c'est-à-dire encapsulant les données à transmettre de façon chiffrée.
  - Le VPN permet donc d'obtenir une liaison sécurisée à moindre coût, si ce n'est la mise en œuvre des équipements terminaux. En contrepartie, il ne permet pas d'assurer une qualité de service comparable à une ligne louée dans la mesure où le réseau physique est public, donc non garanti.

- **VPN, Virtual Private Network ; Réseau privé virtuel** (suite…)
  - Le VPN implique :
    - L'authentification (et donc l'identification) du client et du serveur
    - La confidentialité des données par chiffrement
  - On distingue les protocoles de VPN suivants :
    - IPSec, *Internet Protocol Security Standard*, protocole de niveau 3, offre les services de contrôle d'accès, d'authentification, d'intégrité et de confidentialité de données. Il utilise un mécanisme anti-rejeu et admet un bon nombre d'algorithmes de chiffrement et de hachage.
    - PPTP, *Point-to-Point tunneling Protocol*, protocole de niveau 2 conçu par Microsoft.
    - L2F, *Layer Two Forwarding* de Cisco. Obsolète
    - L2TP, *Layer Two Tunneling Protocol,* est l'aboutissement des travaux de l'IETF (RFC 3931) pour faire converger les fonctionnalités de PPTP et L2F. C'est un protocole de niveau 2 s'appuyant sur PPP.

4 - Architectures client-serveur

4.8 - Appel de procédure distante : RPC, Remote Procedure Call

- **Introduction**
  - On utilise un mécanisme similaire à un appel classique d'une fonction avec, en entrée, des paramètres d'appel, et la récupération d'un résultat.
  - Dans le cas du RPC :
    - C'est une technique client-serveur, en mode requêtes/réponses
    - Un programme appelle une procédure située sur un hôte distant :

```
/* API classique */
  /* Coté client : invoque génère l'appel distant et récupère le résultat*/
invoque (id_client, id_serveur, nom_procedure, paramètres);

  /* Coté serveur : reçoit, traite un appel et répond */
traite (id_client, id_serveur, nom_procedure, paramètres);

  /* Service intégré objet */
  /* Coté client : on invoque une procédure localisée à distance*/
ref_objet_serveur.nom_procedure (paramètres);

  /* Coté serveur : on déploie l'objet qui implante la procédure*/
method nom_procedure (paramètres);
```

4 - Architectures client-serveur

4.8 - Appel de procédure distante : RPC, Remote Procedure Call

- **Introduction** (suite…)
  - Objectif :
  - Simplifier la programmation d'applications distribuées/réparties
    - Ne pas se préoccuper de la localisation de la procédure
    - Ne pas se préoccuper du traitement de défaillances
  - RPC traditionnels :
    - Sun ONC/RPC : *Open Network Computing / Remote Procedure Call*
    - OSF - DCE : *Open Software Foundation - Distributed Computing Environnment*
    - SGBD : procédures stockées
  - RPC intégrés :
    - (SUN) Oracle Java RMI, *Remote Method Invocation*
    - (SUN) Oracle J2EE EJB : *Java 2 (Platform) Enterprise Edition - Enterprise Java Beans*
    - OMG - CCM : *Object Management Group - Corba Component Model*
    - WS-SOAP : *Web Services - Simple Object Access Protocol*
    - Microsoft-DCOM = *Distributed Component Object Model*
  - Le protocole de transport généralement utilisé est UDP, donc en mode non connecté.
  - Secure RPC protège les appels RPC avec un mécanisme d'authentification.
  - Standardisation : RFC 1057, 5531 « *Remote Procedure Call Protocol Specification version 2* »

4 - Architectures client-serveur

4.8 - Appel de procédure distante : RPC, Remote Procedure Call

- **Mise en œuvre du RPC**
  - Le programme client contacte un stub client (la souche client, ou talon client) auquel il est lié, via un appel local.
  - Le stub client emballe et aligne les paramètres et une référence dans un message (*marshalling* ou sérialisation)

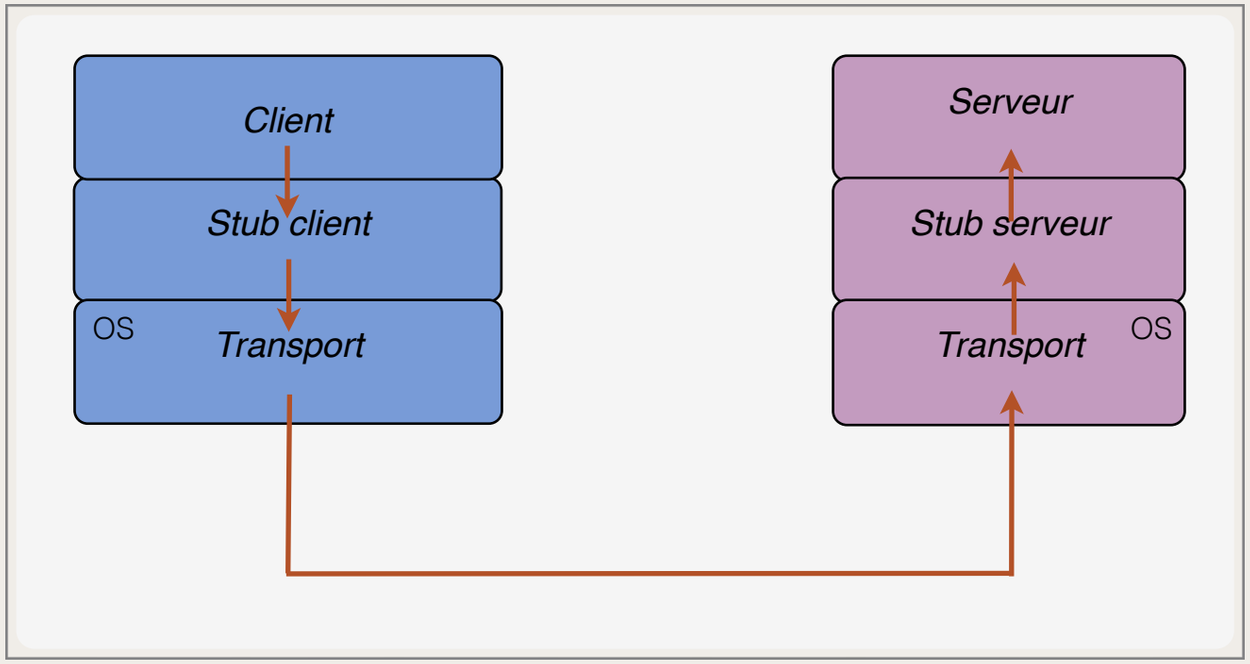


Fig 4.15 - Couches logicielles du RPC

  - La souche client détermine l'adresse du serveur et émet un appel système pour envoyer le message
  - L'OS envoie le message vers le serveur, via une entité de transport
  - Le paquet entrant est fourni au stub (ou skeleton) serveur
  - Le stub serveur crée ou sélectionne une nouvelle procédure, déballe les paramètres et appelle la procédure avec les paramètres décodés

4 - Architectures client-serveur

4.8 - Appel de procédure distante : RPC, Remote Procedure Call

- **Mise en œuvre du RPC** (suite…)
  - La réponse suit le même trajet, dans l'autre sens.
  - *Nota* :
    - *stub* ≈ souche ou talon;
    - *skeleton* ≈ squelette ;
    - *proxy* ≈ délégué

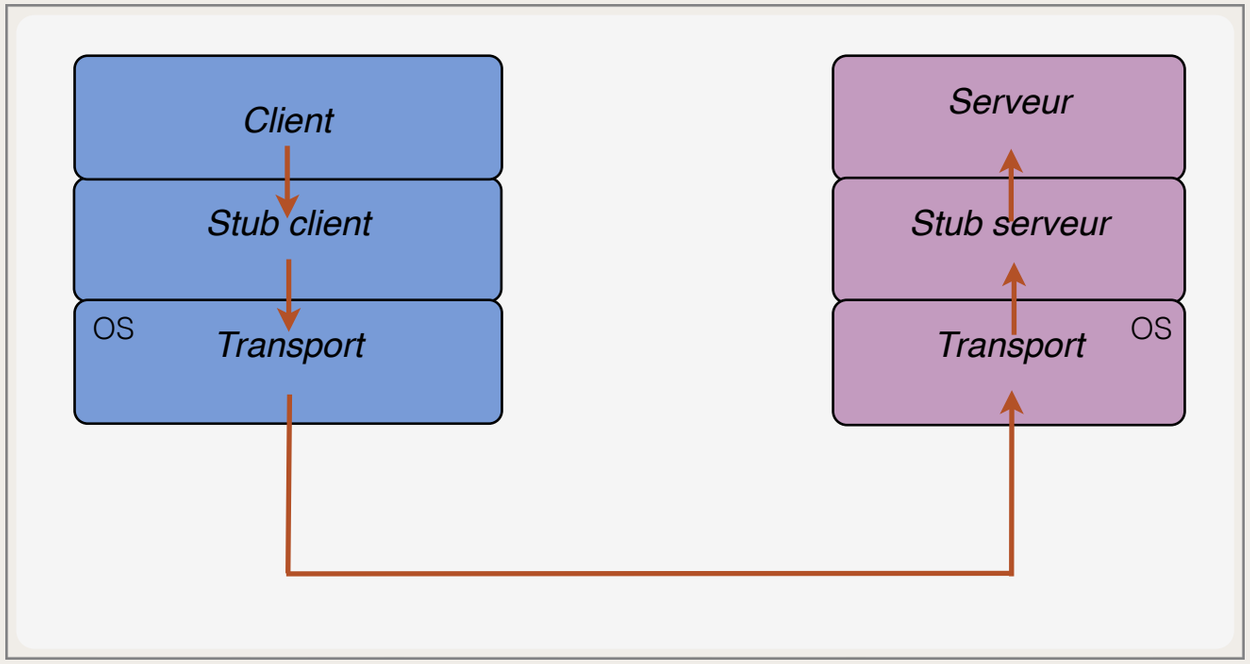


Fig 4.12 - Couches logicielles du RPC

- **Le stub client**
  - Représente le serveur sur le site client
  - Reçoit l'appel local
  - Emballe les paramètres
  - Crée un identificateur unique pour l'appel
  - Exécute l'appel distant
  - Met le processus client en attente
  - Reçoit et déballe les résultats
  - Exécute le retour vers l'appelant (comme un retour local de procédure)

- **Le stub serveur**
  - Représente le client sur le site serveur
  - Reçoit l'appel sous forme de message et déballe les paramètres
  - Crée ou sélectionne un processus (ou *thread*) et lui fait exécuter la procédure
  - Emballe les résultats et les transmet à l'appelant

- **Les problèmes à résoudre :**
  - Adressage et liaison : comment connaître l'adresse du serveur ?
  - Passage des paramètres : comment emballer et déballer les paramètres d'appel et les résultats ?
  - Mode synchrone ou asynchrone coté client
  - Exécution séquentielle ou parallèle des appels coté serveur
  - Concurrence d'accès : gestion d'appels simultanés ; persistance des données partagées
  - Gestion des pannes
  - Génération des souches client et serveur

4 - Architectures client-serveur

4.8 - Appel de procédure distante : RPC, Remote Procedure Call

- **Liaison : comment localiser une procédure distante ?**
  - Au moyen d'un service de gestion de noms (**serveur d'annuaire**)
  - **Liaison statique** : pas d'appel à un serveur de noms mais un appel fixé à la compilation
  - **Liaison au premier appel** : consultation du serveur de noms au premier appel seulement
  - **Liaison à chaque appel** : consultation du serveur de noms à chaque appel
- **Passage des paramètres**
  - Client et serveur sont dans des espaces d'adressage différents
  - Pas de passage de pointeur : on peut remplacer un passage par référence par une opération de **copie/restauration**.
  - La sérialisation (*marshalling)* des paramètres impose
    - soit l'utilisation d'un standard de représentation de données (**ASN.1,** *Abstract Syntax Notation-1*),
    - soit une représentation externe commune (ex. : XDR, *External Data Representation*, de Sun),
    - soit un système de conversion coté serveur ou client.
  - Définition de nouveaux langages de syntaxe abstraite adaptés aux appels de procédure distante : **IDL**, *Interface Definition Language*

4 - Architectures client-serveur

4.8 - Appel de procédure distante : RPC, Remote Procedure Call

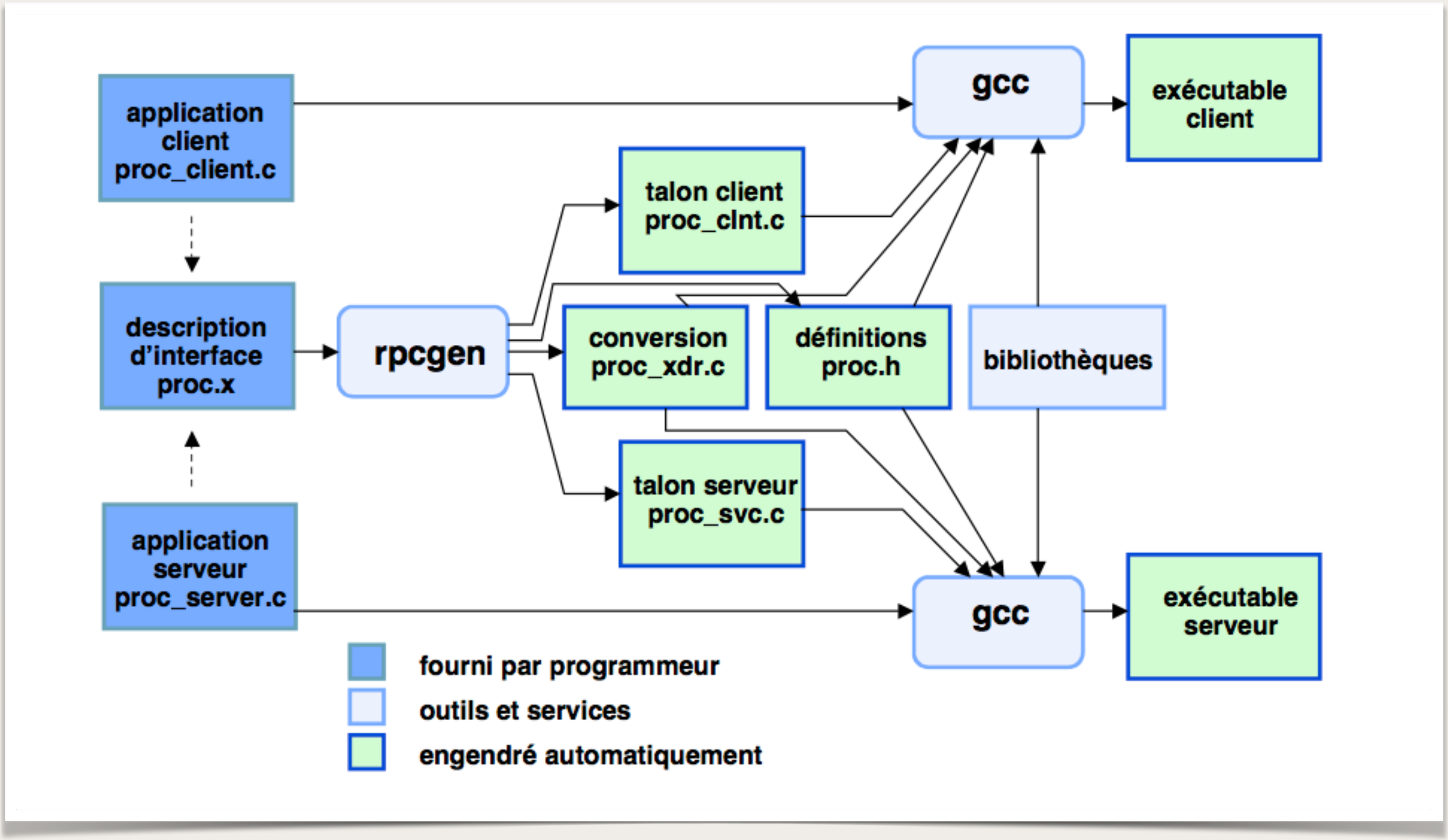
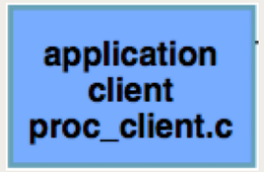


Fig 4.16 - RPC - Génération des exécutables client et serveur

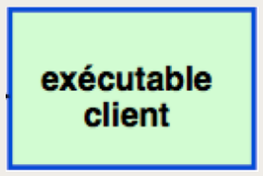
- **Conception d'applications**
  - Trois composantes sont développées :
    - Le programme principal de l'application cliente



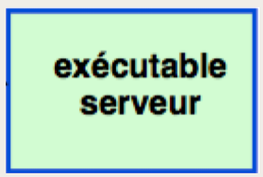
    - Les procédures de l'application serveur

    - La description des échanges entre client et serveur (avec le langage IDL, *Interface Definition Language*)

- **Pour créer les exécutables :**
  - Le compilateur IDL permet de générer **les stubs client et serveur**
  - Construction de l'**exécutable client** : compiler programme client et le stub client ; lier les objets obtenus.



  - Construction de l'**exécutable serveur** : compiler le programme serveur et le stub serveur ; lier les objets obtenus.



4 - Architectures client-serveur

4.8 - Appel de procédure distante : RPC, Remote Procedure Call

- **Voir aussi**
  - https://www.ionos.fr/digitalguide/serveur/know-how/quest-ce-que-le-remote-procedure-call/
  - gRPC : framework RPC open source initialement développé par Google
    - HTTP/2 est utilisé pour le transport
    - Protocol Buffers est utilisé comme langage de description d'inferface, au lieu d'IDL.
    - https://www.grpc.io

4.9 - MOM : Message-Oriented Middleware

- **Message-Oriented Middleware**
  - Système de logiciels qui permet l'échange de messages de façon asynchrone entre les applications présentes sur un réseau informatique. Un MOM permet une forme de couplage faible entre applications.
  - C'est un **middleware par files d'attente**. En français : **Middleware orientés Message**.
  - La communication de deux applications via un *Message Oriented Middleware* est complètement asynchrone, c'est à dire que l'émetteur et le destinataire n'ont pas besoin d'être connectés simultanément lorsqu'ils communiquent.

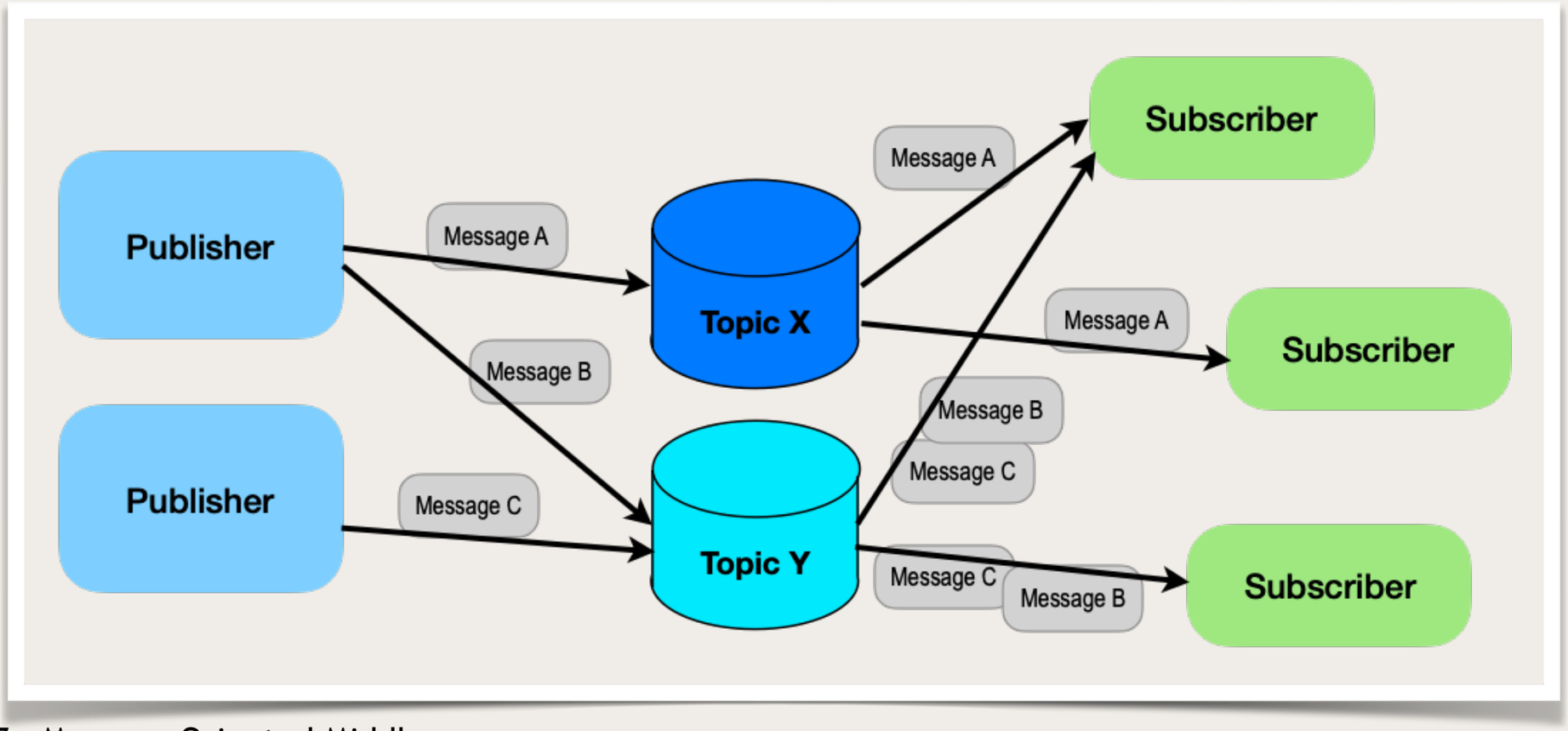


Fig 4.17 - Message-Oriented Middleware

- **Mode de fonctionnement**
  - Le mode classique est en point à point. Les MOM utilisent des files d'attentes ou ***queues*** par lesquelles transitent les messages. Lorsqu'une application produit et envoie un message :
    - il se connecte au *broker* de messages (courtier de messages) ou *provider*
    - il y envoie le message en précisant l'identifiant de la file d'attente.
    - Quand le destinataire du message se connecte à son tour au *broker*, il peut lire la file d'attente et il peut alors rechercher et lire le message.
    - Une fois qu'un message est lu, il est retiré de la file d'attente.
  - Une file d'attente donnée peut être utilisée pour plusieurs couples d'applicatifs (pas besoin de dédier une file par liaison applicative) puisque les MOM comportent différents critères de sélection de messages lors de la lecture.

- **Message**
  - Le message représente les informations échangées par deux applications via le MOM. Il est constitué de trois parties distinctes :
    - Les données elles mêmes, ou charge utile (*payload*)
    - L'entête du message, qui comporte ses caractéristiques techniques (son identifiant, date de dépôt, date d'expiration...)
    - Ses propriétés, les caractéristiques fonctionnelles du message, qui sont différentes pour chaque application émettrice. Les propriétés forment un ensemble de (clés, valeurs).

- **Fonctionnalités offertes par les MOM**
  - Les Middleware Orientés Message, offrent des services de base :
    - **Acheminement** (envoi, réception)
    - **Stockage** (soit en mode **persistant** si les messages sont stockés sur disque, soit en mode **non persistant** s'ils résident en mémoire vive)
    - **Recherche** des messages

- **Fonctionnalités offertes par les MOM,** (suite…)
  - Ils offrent en général des services évolués comme :
    - Gestion de priorités
    - Compression des données utiles du message
    - Gestion des expirations ou des dates de disponibilités de message
    - Routage des messages d'un nœud à l'autre (un peu à la manière des serveurs de mails)
    - Triggering : lancement d'applications lorsque des messages sont disponibles pour elle
    - Gestion d'alertes (présence de messages dans une file donnée)
    - Gestion de quotas.
  - La plupart des MOM actuels implémentent l'interface JMS, *Java Message Service*, standard pour la communication en mode message en Java.
  - *Advanced Message Queuing Protocol* (AMQP) est est un protocole ouvert et un standard pour les échanges entre serveurs de messages.
  - Apache Qpid propose des outils et API de messagerie qui parlent AMQP, pour beaucoup de langages et de plates-formes.

- **Quelques MOM commerciaux :**
  - IBM WebSphere MQ d'IBM
  - Message Queuing (MSMQ) de Microsoft

- **Quelques MOM open-source :**
  - RabbitMQ de Pivotal, avec le protocole AMQP
  - ActiveMQ du projet Geronimo d'Apache, très utilisé, surtout dans des environnements Java/JMS
  - Zeromq ou ØMQ. Une bibliothèque de messagerie asynchrone haute performance qui peut fonctionner sans message broker.

4 - Architectures client-serveur

4.9 - MOM : Message-Oriented Middleware

- **Middleware de publication/abonnement (*Publish / Subscribe*)**
  - C'est un concept proche du MOM
  - On parle de technologie de « *push* ». (Le serveur 'pousse' les publications).
- **Fonctionnement :**
  - Un producteur réalise une publication, enregistrée par un courtier d'évènement (*Event broker*).
  - Un consommateur s'abonne à une catégorie d'évènement (un *topic*).

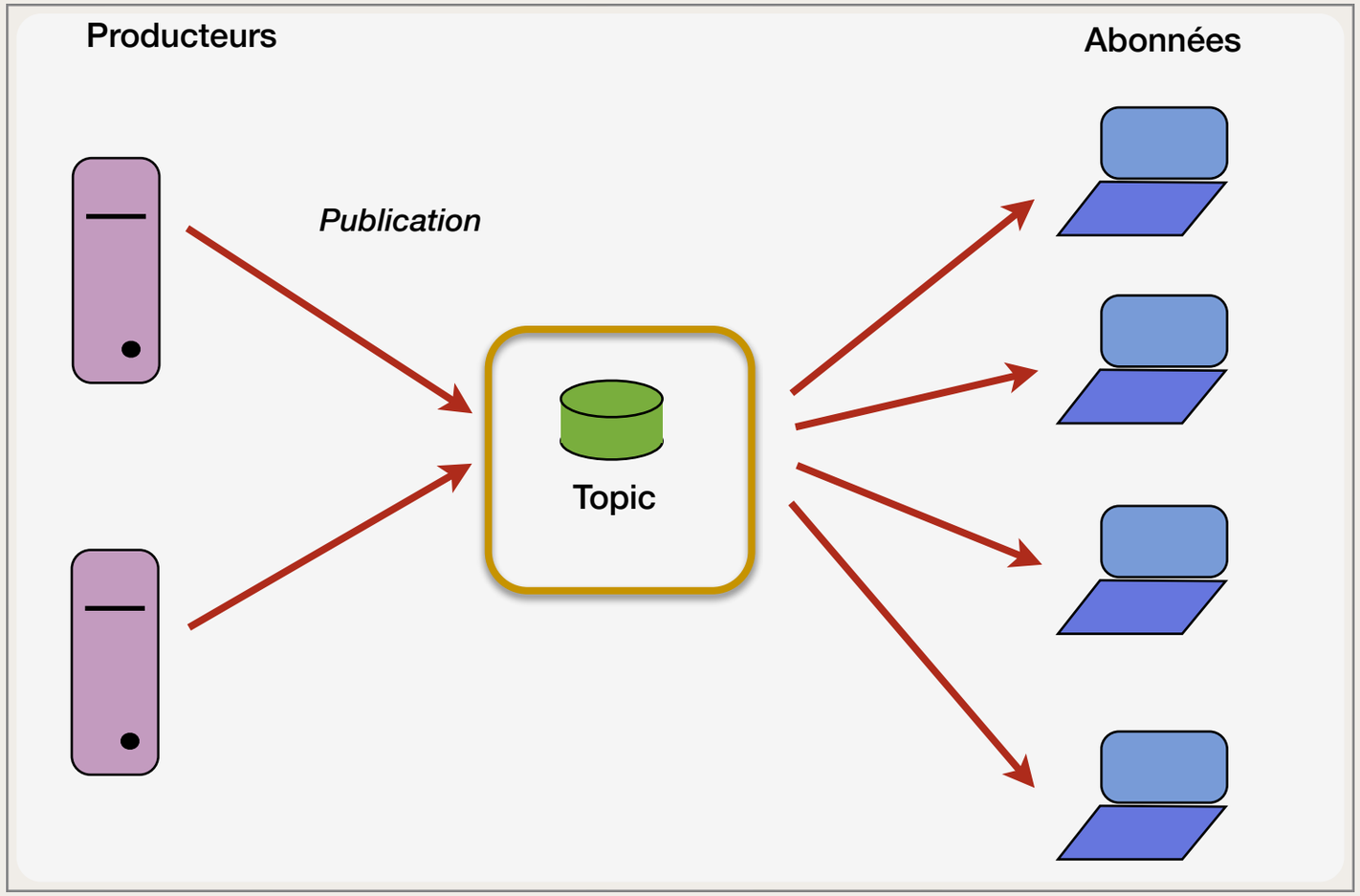


Fig 4.18 - Middleware de publication/abonnement

- Le middleware gère la distribution des évènements entre producteurs et consommateurs.
- Les messages envoyés à un *topic* restent dans la file d'attente jusqu'à ce que toutes les applications abonnées aient lu le message.

4 - Architectures client-serveur

4.9 - MOM : Message-Oriented Middleware

- **Variante :**
  - Le mode « pull » est parfois également proposé
- **Exemples :**
  - Cloud Pub/Sub de Google

4 - Architectures client-serveur

4.9 - MOM : Message-Oriented Middleware

- **MQTT, *Message Queuing Telemetry Transport***
  - Protocole de messagerie *publish-subscribe* basé sur le protocole TCP/IP
  - MQTT v5.0 et MQTT v3.1.1 sont des standards OASIS
  - un serveur utilise TCP port MQTT 1883 ou Secure MQTT 8883 (MQTT over TLS)
  - Un courtier de message (*message broker*) est un serveur recevant des messages de clients producteurs et les publient aux clients abonnés.
- MQTT est, entre autres, adapté à l'Internet des Objets et au M2M (*Machine to Machine*).
- Les principaux agents open-sources sont :
  - ActiveMQ
  - ejabberd
  - JoramMQ, OW2 JORAM
  - Mosquitto

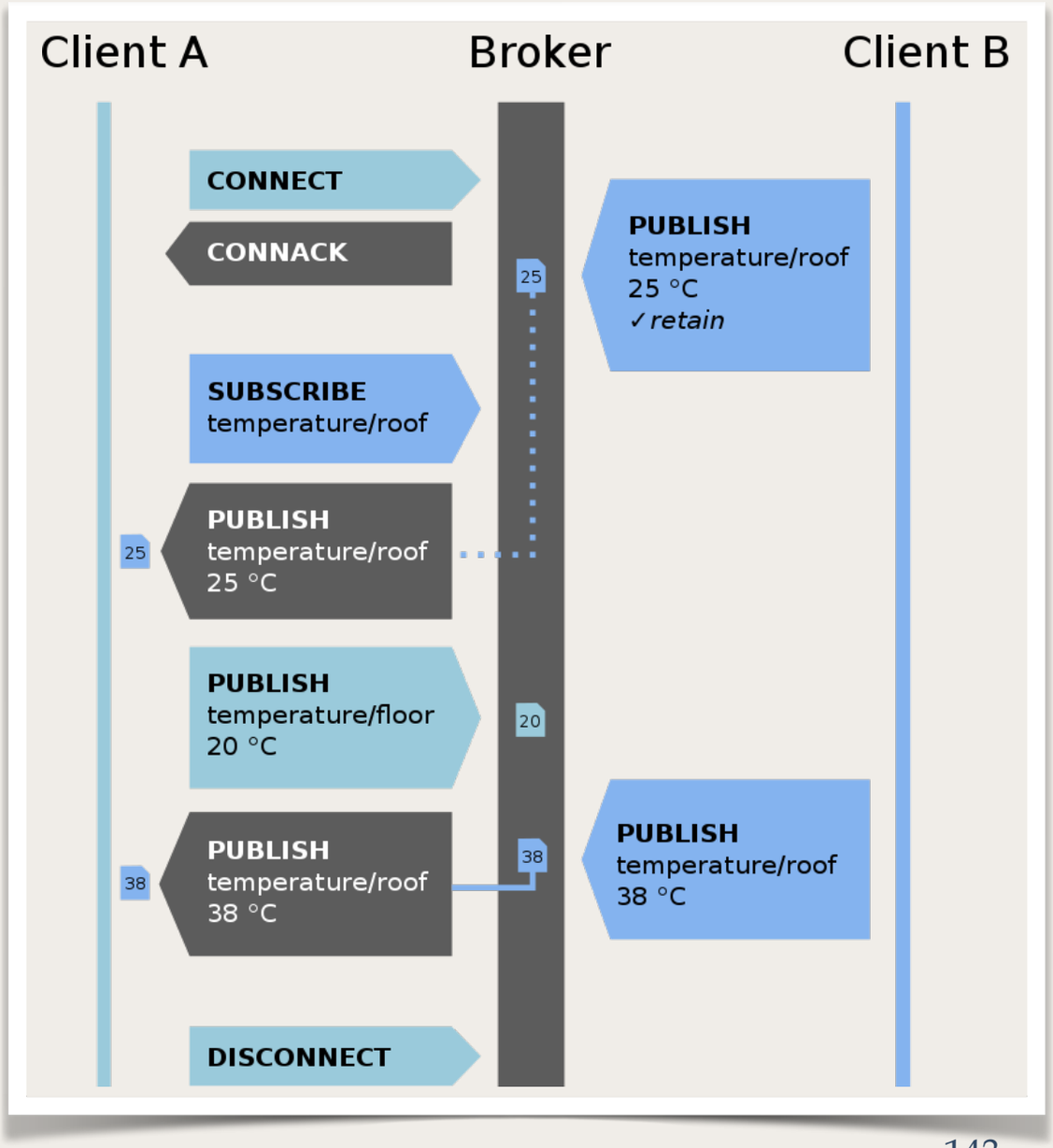


Fig 4.19 - *Message Queuing Telemetry Transport*

5 - Systèmes distribués

5.1 - Introduction

- **Définitions**
  - Un **système distribué** (aussi appelé **système réparti**) est un ensemble d'unités de traitement **autonomes** interconnectées entre-elles.
    - Exemples : Internet ; RTC (Réseau téléphonique commuté) ; GPS ; Réseau de capteurs…
    - **Autonomes** => OS et/ou programmes locaux ; mémoires locales ; asynchrones ; pas de temps global
    - **Interconnexion** => Échange d'informations par envoi de messages
  - Un ensemble d'ordinateurs indépendants qui apparaît à un utilisateur comme un système unique et cohérent
    - Les machines sont autonomes
    - Les utilisateurs ont l'impression d'utiliser un seul système.
- Un **algorithme distribué** (ou réparti) est une suite d'instructions répartie sur plusieurs sites. Chaque site calcule et communique avec d'autres sites.
  - Ex. : Algorithmes de routage ; algorithmes d'équilibrage de charge.

5 - Systèmes distribués

5.2 - Partage de données dans le cloud

- **Problématique**
  - Les systèmes distribués utilisent la puissance de calcul et l'espace de stockage de plusieurs unités de traitement interconnectées par des réseaux.
  - Leur taille peut aller du plus petit système composé de deux entités aux millions de nœuds que forment le réseau internet.
  - La **localisation** des unités de traitement et des données doit être **transparente** pour l'utilisateur.
  - Le système doit gérer la localisation et le transfert des données et ce, de manière transparente.
  - Une **grille informatique** mutualise un ensemble de ressources informatiques géographiquement distribuées dans différents sites.
    - La grille consiste en une infrastructure composée de ressources informatiques hétérogènes interconnectées par un réseau.
    - L'idée est de proposer un super-ordinateur en réunissant la puissance de calcul et l'espace de stockage de sites géographiquement dispersés.
  - **Cohérence des données** : capacité pour un système à refléter sur la copie d'une donnée les modifications intervenues sur d'autres copies de cette donnée

5 - Systèmes distribués

5.2 - Partage de données dans le cloud

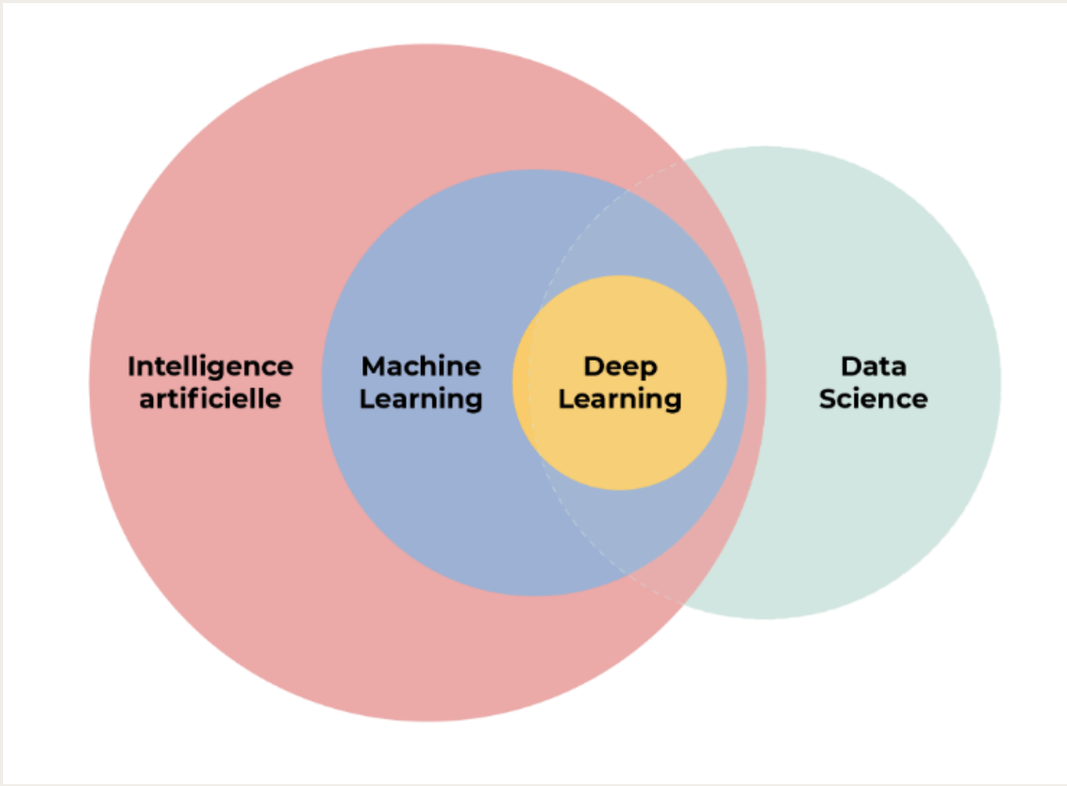
- **Définitions**
  - HPC, *high-performance computing* ; Calcul haute performance.
  - MPP, *Massively Parallel Processing* ; Traitement massivement parallèle :
    - Exécution coordonnée d'un programme par plusieurs processeurs.
  - ETL, *Extract, Transform, Load* ; Extraction, Transformation, Chargement.
    - Voir Apache Hive
  - *Grid ;* Grille informatique :
    - Infrastructure virtuelle constituée d'un ensemble de ressources informatiques partagées, distribuées, hétérogènes, délocalisées et autonomes.
  - *Computer cluster* ; grappe de serveurs, ferme de calcul pour réaliser le calcul distribué, *Distributed computing*.

- **Définitions, suite…**
  - *Data store* ; Dépôt de données :
    - Terme générique qui désigne l'ensemble des bases de données, des systèmes de fichiers ou de répertoires.
  - *Data warehouse* ; entrepôt de données ;
    - Type particulier de base de données.
  - *Data lake* ; lac de données :
    - Stockage de données massives où les données sont gardées dans leur formats natifs, dans des base NoSQL par exemple.
  - *Data Mining* ; exploration de données :
    - Consiste à rechercher des relations qui n'ont pas encore été identifiées.
  - *Data science* ; Science des données :
    - Domaine interdisciplinaire qui utilise des méthodes, des processus, des algorithmes et des systèmes scientifiques pour extraire des connaissances et des idées à partir de données structurées et non structurées.

5 - Systèmes distribués

5.2 - Partage de données dans le cloud

- **Définitions, suite…**
  - *Dataviz* ; visualisation de données.
  - *Open data* ; données ouvertes :
    - Données numériques dont l'accès et l'usage sont laissés libres aux usagers.
  - BI, *Business Intelligence* ; informatique décisionnelle :
    - Processus d'analyse des données qui vise à doper les performances métier en aidant à prendre des décisions plus avisées.
  - *Machine Learning*, apprentissage automatique :
    - Sous-ensemble de l'IA.
    - Méthode d'analyse de données permettant aux ordinateurs d'apprendre et de s'adapter sans programmation explicite, en identifiant des modèles dans de grandes quantités de données.
  - *Deep Learning*, apprentissage profond :
    - Il s'applique souvent sur des quantités de données beaucoup plus importantes que le *Machine Learning*.
    - Il apprend de cette masse d'exemples et obtient dans certains cas de bien meilleurs résultats que les disciplines traditionnelles d'intelligence artificielle.



5 - Systèmes distribués

5.2 - Partage de données dans le cloud

- **Big data**
  - Le *Big Data* fait référence à **l'explosion du volume des données** dans l'entreprise et des nouveaux moyens technologiques proposés par les éditeurs pour y répondre.
    - YouTube **reçoit** 500 heures de vidéo toutes les minutes ;
    - Plus d'1 milliard d'heures de vidéos sont visionnées sur YouTube par les internautes à travers le monde tous les jours ;
    - 4 pétaoctets (4×10^{15} octets soit 4000 téraoctets) de données **transitent** chaque jour sur Facebook ;
    - 500 millions de tweets étaient **envoyés** par jour en 2017 ;
    - 8,6 milliards de téléphones mobiles en activité dans le monde ;
    - 1000 articles commandés par minute chez *amazon.fr* pendant le Black Friday ;
    - 188 millions d'emails sont échangés toutes les minutes ;
    - 90% des données créées dans le monde l'ont été au cours des 2 dernières années ;

5 - Systèmes distribués

5.2 - Partage de données dans le cloud

- **Big data, suite…**
  - *Big data,* données massives :
    - Volume de données numériques créées dans le monde :
      - 2 zettaoctets en 2010
      - 47 zettaoctets en 2020
      - Environ 175 Zo en 2025
      - *612 Zo en 2030 ?*
      - *2,1 Yo en 2035 ?*
    - 1 zettaoctet = 1 Zo = 10^{21} octets = 1 000 000 Po.
    - 1 yottaoctet = 1Yo = 1000 zettaoctets

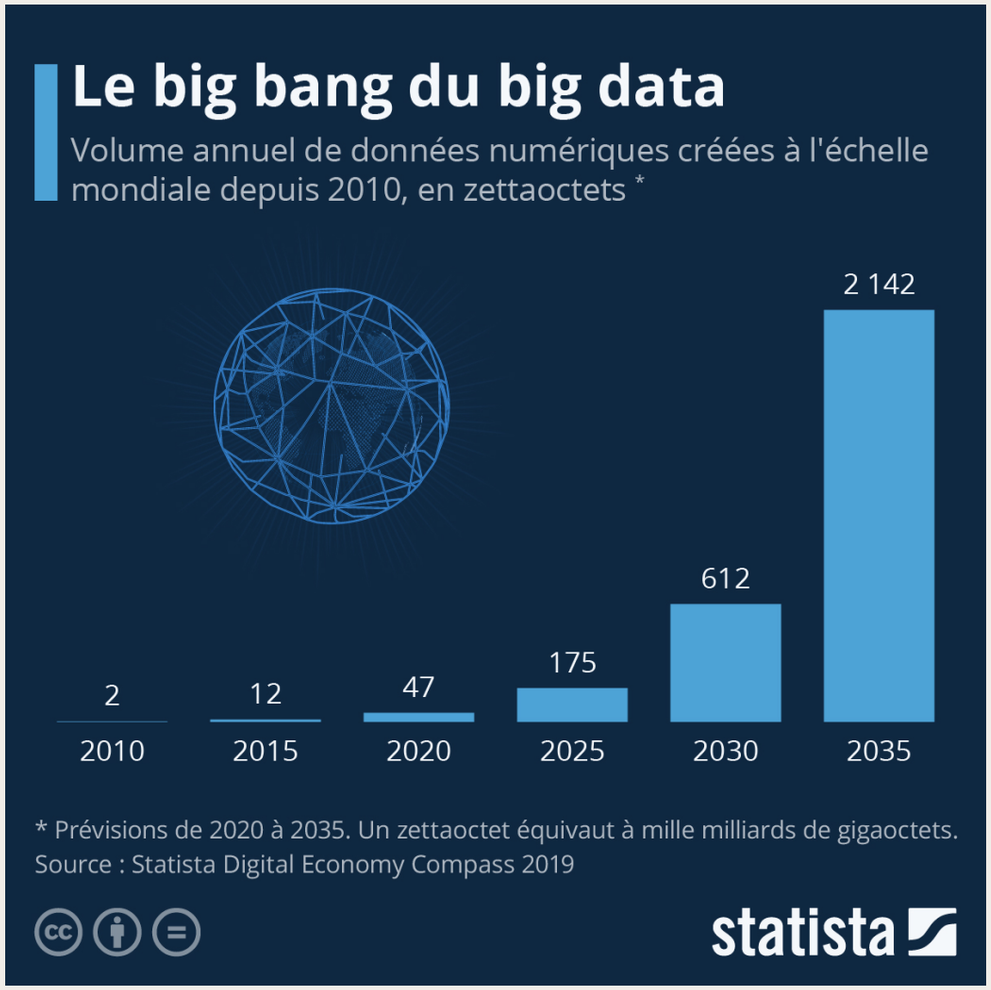


Fig 5.1 - Le Big Bang du Big Data

- Les 4V :
  - **Volume** de données + **Vitesse** de traitement + **Variété** de formats de données + **Véracité** (erreurs, incomplétude, confiance, fraîcheur, etc.)

- **La cohérence de données (*data consistency*)**
  - Un système en **cohérence forte** assure que toute lecture d'une copie d'une donnée reflétera toute modification antérieure à la lecture intervenue sur n'importe quelle copie de la donnée.
  - Un système en **cohérence faible** assure que si une copie est modifiée, toutes les copies des données refléteront ces modifications au bout d'un certain temps, mais la modification n'est pas forcément immédiate.
  - **Cohérence stricte** : c'est la forme la plus intuitive de la cohérence. Elle correspond à garantir au programmeur que chaque lecture d'une donnée correspond bien à la version la plus fraîchement modifiée dans tous les caches du système.
  - **Cohérence séquentielle** : Elle consiste à préserver l'ordre global des accès mémoire en lecture. Cet ordre correspond à l'ordre du programme. Le résultat final d'une exécution est similaire au résultat d'une exécution séquentielle du programme.
  - **Cohérence faible** : le programmeur est impliqué dans la gestion de la cohérence en utilisant des opérateurs de synchronisation séquentiellement cohérents.
  - **Cohérence par nœud** : L'ordre des écritures effectuées par un même nœud est préservé tandis que les écritures des différents nœuds peuvent être vues autrement.
  - **Cohérence relâchée** : on considère deux types d'opérateurs de synchronisation : *acquire* (pour acquérir l'accès à une variable partagée), *release* (pour libérer l'accès à une variable partagée). Chaque opérateur est considéré cohérent par nœud.

- **Théorème CAP, alias théorème de Brewer**
  - CAP = ***Consistency, Availability & Partition Tolerance***, soit Cohérence, Disponibilité & Tolérance au partitionnement.
  - **Eric Brewer** de l'université de Californie à Berkeley a énoncé une conjecture, devenu le **théorème de Brewer** ou Théorème CAP, qui prouve que :
  - Il est impossible sur un système informatique de calcul distribué de garantir en même temps (c'est-à-dire de manière synchrone) les trois contraintes suivantes :
    - *Consistency* (Cohérence) : tous les nœuds du système voient exactement les mêmes données au même moment ;
    - *Availability* (Disponibilité) : garantie que toutes les requêtes reçoivent une réponse ;
    - *Partition Tolerance* (Tolérance au partitionnement) : aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement.
  - D'après ce théorème, un système de calcul/stockage distribué ne peut garantir à un instant ***t*** que deux de ces contraintes mais pas les trois.

5 - Systèmes distribués

5.2 - Partage de données dans le cloud

- **Théorème CAP, alias théorème de Brewer, suite…**
  - D'où les trois catégories de systèmes distribués :
    - **Système AP** (disponibilité et tolérance au partitionnement)
      - Exemple : DNS
      - Si une entrée du DNS est modifiée, cela peut prendre plusieurs jours avant que cette modification ne soit transmise à toute la hiérarchie du système et ne puisse être vue par tous les clients
    - **Système CA** (cohérence et disponibilité) ;
      - Exemple : SGBD
      - Où la tolérance au partitionnement joue un rôle mineur
    - **Système CP** (cohérence et tolérance au partitionnement)
      - Exemple : les applications financières et bancaires

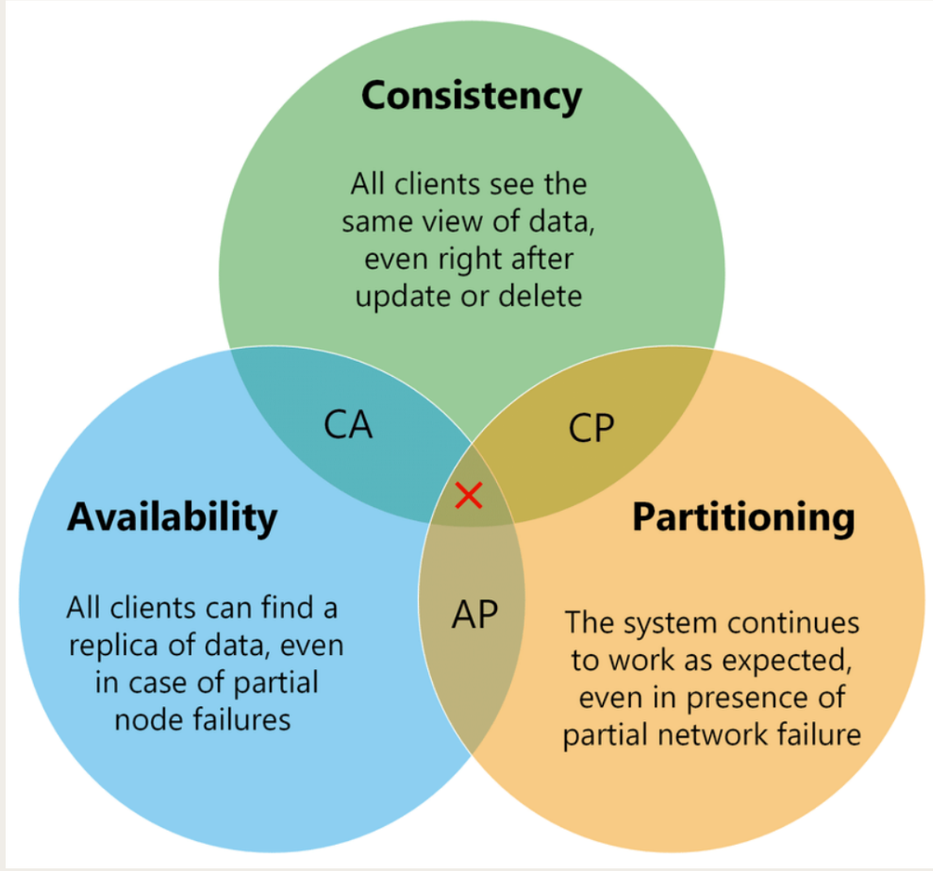


Fig 5.2 -Diagramme du théorème CAP

- Voir : Théorème CAP : cohérence, disponibilité et tolérance au partitionnement - Digital Guide IONOS
- Qu'est-ce que le théorème CAP ? - IBM

5 - Systèmes distribués

5.2 - Partage de données dans le cloud

- **Big data**
  - Les besoins :
    - Systèmes qui peuvent gérer de gros volumes de données
    - *Scalable* (extensible)
    - Robuste
    - Haute disponibilité
    - Économique
  - Nouveaux modèles de bases de données
    - Les bases de données relationnelles ne sont pas adaptés
    - Des nouveaux modèles de repésentation sont utilisés (des patrons d'architectures), comme BDAF, *Big Data Architecture Framework,* exploités par MapReduce créé par Google.
- Stockage
  - *Data lake*, Lac de données : les données sont gardées dans leur formats originaux dans des base NoSQL par exemple.
  - Le Cloud computing propose des services comme Google BigQuery, AWS Big Data, etc.
  - DFS, *distributed file system,* Systèmes de fichiers distribués : les données sont stockées là où elles peuvent être traitées.

- **Apache Hadoop, un framework open source**
  - Pour créer des applications de traitement et de **gestion intensive de données**
  - Doug Cutting est un cofondateur du projet Hadoop (chez Google, puis Yahoo)
  - Framework écrit en Java et inspiré de Google FileSystem et de Google MapReduce
  - Objectifs :
    - Stocker les données dans un espace de stockage massif ;
    - Rechercher et restituer des données ;
    - Avec une énorme puissance de traitement et la possibilité de prendre en charge une quantité de tâches virtuellement illimitée.

- Voir :
  - Hadoop : qu'est-ce que c'est et comment apprendre à l'utiliser ? - DataScientest

- **Apache Hadoop, un framework open source**
  - Trois constituants essentiels :
    - **HDFS**, *Hadoop Filesystem*, un système de fichiers virtuel qui regroupe les ressources de stockage de plusieurs machines au sein d'un cluster ;
    - **Hadoop MapReduce** : un framework logiciel en Java permettant de développer des programmes exécutables de manière distribués grâce à l'utilisation de l'algorithme **MapReduce** développé par Google.
    - Apache **HBase,** une base de données NoSQL, scalable et distribuée conçue pour les analyses Big Data.
    - L'écosystème Hadoop comprend des logiciels et des utilitaires associés, notamment Apache **Hive**, Apache **HBase**, **Spark**, **Kafka**, etc.
- **Acteurs**
  - Fondation Apache
  - **Cloudera**, start-up de la Silicon Valley ; développement de logiciels de Big Data basées sur le framework Hadoop
  - **Hortonworks**, société de logiciels informatique Californienne ; développement et soutien de Hadoop. Elle a fusionné avec Cloudera en 2018.
  - Les GAFAM

5.3 - Hadoop

- **Apache Hadoop, l'écosystème**

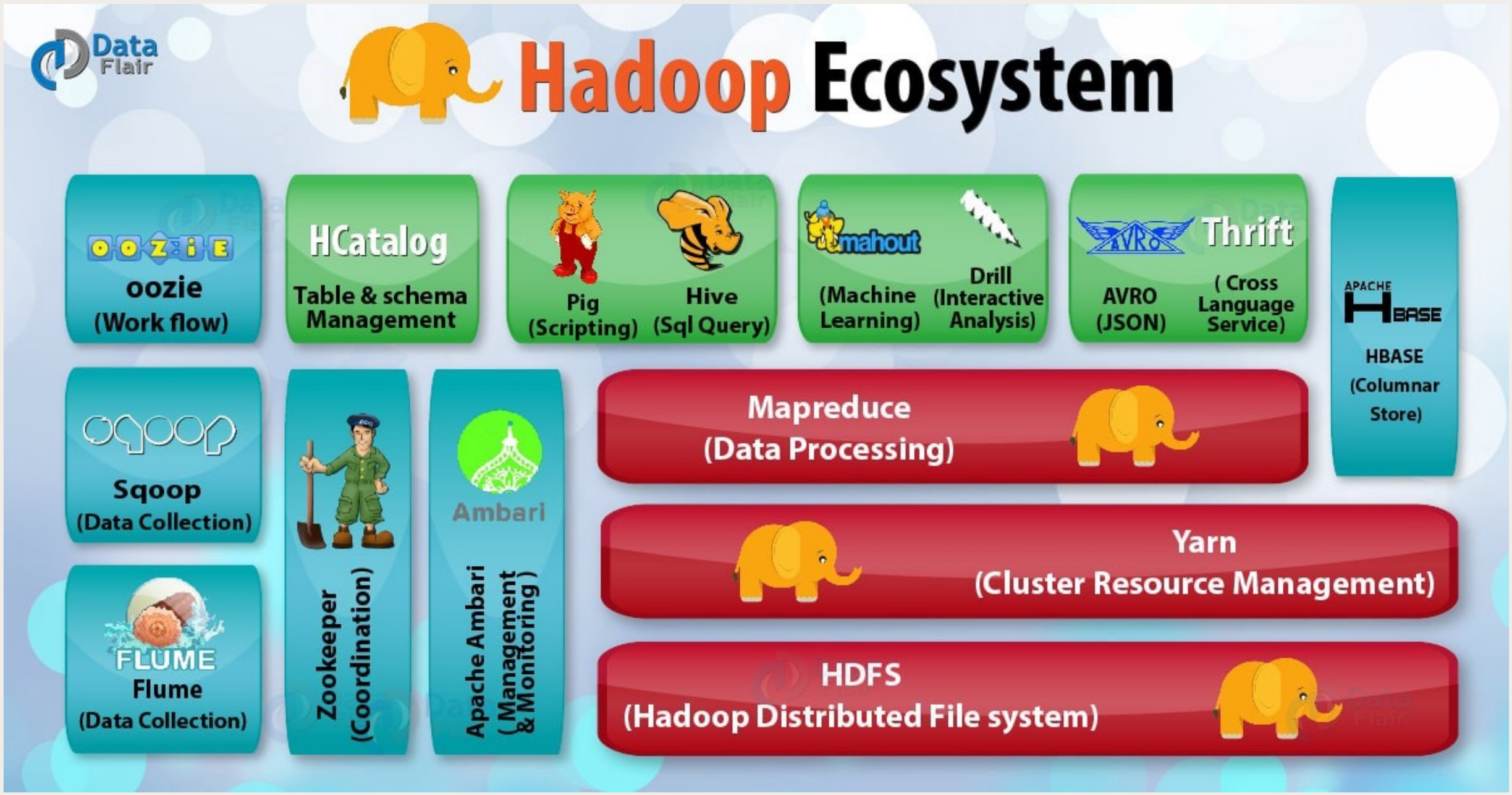


Fig 5.3 - L'écosystème Hadoop

- **MapReduce et Spark**
  - **MapReduce** est un modèle de programmation (un patron d'architecture) qui fournit un cadre pour automatiser le calcul parallèle sur des données massives.
    - Deux étapes :
      - **map** : consiste à appliquer une même fonction à tous les éléments de la liste ;
      - **reduce** : applique une fonction récursivement à une liste et retourne un seul résultat.
    - **map** et **reduce** sont des opérateurs génériques et leur combinaison permet donc de modéliser énormément de problèmes
    - Dans un calcul MapReduce :
      - on commence par découper le problème en de nombreux sous-problèmes indépendants (étape *Map*)
      - que l'on confie à des ordinateurs distincts.
      - Ces machines résolvent les sous-problèmes et envoient leurs résultats à d'autres machines qui ont pour tâche de combiner les résultats (étape *Reduce*).
      - L'objectif est de pouvoir travailler sur d'énormes volumes de données en parallélisant les calculs sur des grappes d'ordinateurs.
      - MapReduce ne permet de traiter que des problèmes qui peuvent se décomposer en de multiples tâches parallèles.
    - Voir : https://www.youtube.com/watch?v=mhnxcYbdE6M&t=476s

MapReduce et Spark

- **Apache Spark** est également une solution pour écrire simplement des applications distribuées
  - **Spark** propose des bibliothèques de traitement classique.
  - Sa performance est remarquable ; il peut travailler sur des données sur disque ou des données chargées en RAM.
  - Il est plus jeune que MapReduce mais il dispose d'une communauté énorme.
  - c'est donc une solution qui s'avère être le successeur de MapReduce.
  - Voir : https://www.youtube.com/watch?v=ymtq8yjmD9I

- **Pour en savoir plus**
  - Un déluge de données - Interstices
  - Définition : Qu'est-ce que le Big Data ? - Le Big Data
  - Hadoop – Tout savoir sur la principale plateforme Big Data - Le Big Data
  - Apache Hadoop
  - Hadoop : qu'est-ce que c'est et comment apprendre à l'utiliser ? - DataScientest
  - Apache Spark (ne pas confondre avec 🎸 Sparks)
  - Qu'est-ce qu'Apache Hadoop dans Azure HDInsight ?

6 - Architectures d'applications sur le Cloud

6.1 - Architectures orientées micro-services

- **Les micro-services**
  - Les architectures orientées micro-services consistent à découper une application en petits services autonomes, faiblement couplés, qui communiquent via des API légères.
    - Approche moderne de conception des systèmes logiciels, dans laquelle une application est décomposée en un ensemble de services indépendants, chacun dédié à une fonctionnalité spécifique.
  - Chaque service peut être développé, déployé et mis à jour indépendamment, ce qui améliore l'agilité et la capacité à faire évoluer le système
  - Contrairement aux architectures monolithiques traditionnelles, où toutes les fonctionnalités sont regroupées dans un seul bloc applicatif, les micro-services favorisent la modularité, la flexibilité et l'évolutivité.
  - Chaque micro-service est conçu pour être autonome :
    - il possède sa propre logique métier, ses données, et communique avec les autres services via des interfaces bien définies, généralement à travers des API (REST, gRPC, etc.) ou des bus de messages (Kafka, RabbitMQ).
    - Cette indépendance permet de développer, tester, déployer et faire évoluer chaque service séparément, sans impacter l'ensemble du système.

6 - Architectures d'applications sur le Cloud

6.1 - Architectures orientées micro-services

- **Les micro-services**

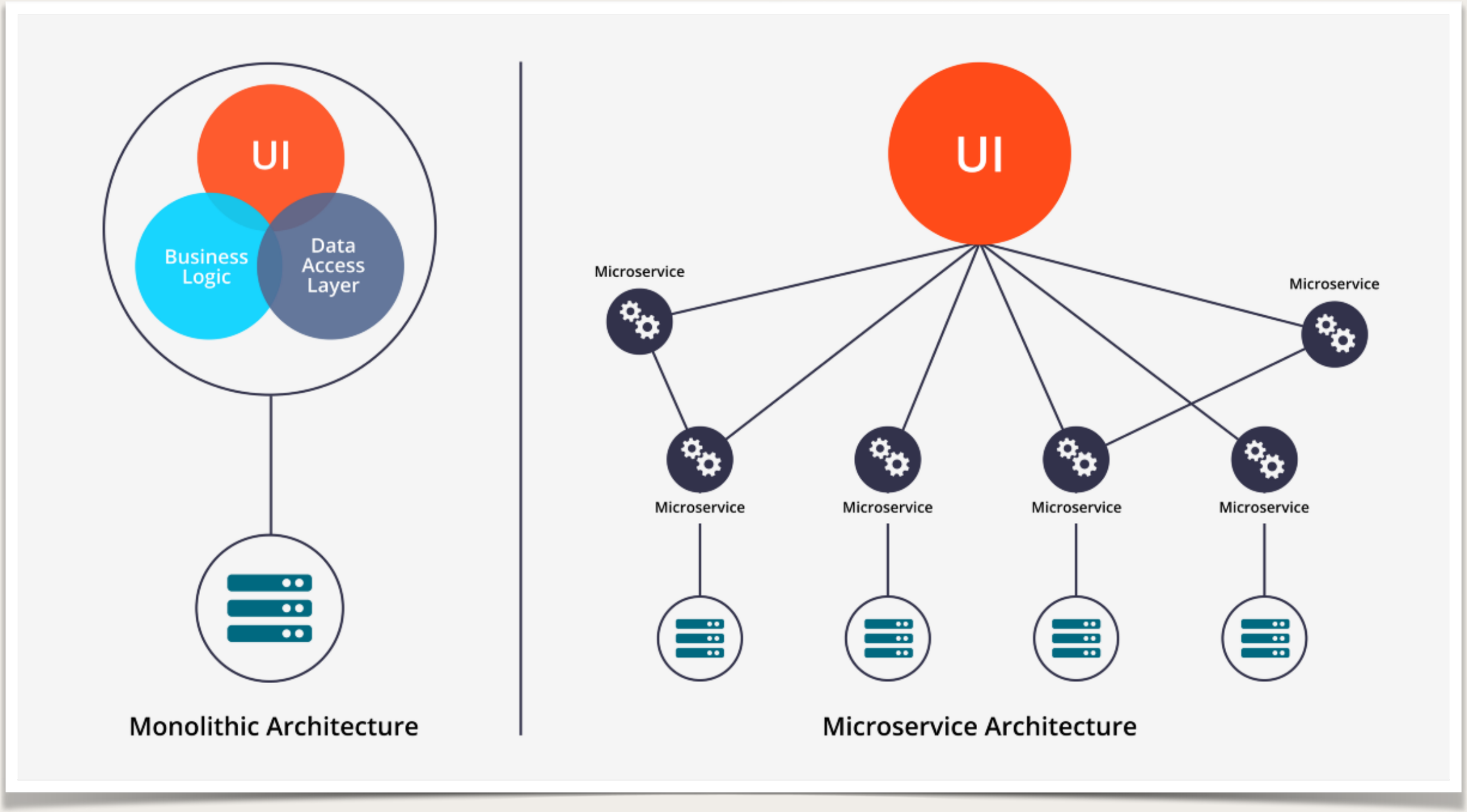


Fig 6.1 - Comparaison entre l'architecture monolithique et les micro-services. devopedia.org

6 - Architectures d'applications sur le Cloud

6.1 - Architectures orientées micro-services

- **Les micro-services**
  - Pour être considéré comme un micro-service, un service doit être :
    - Élastique
    - Résilient
    - Modulaire
    - Minimal
    - Complet
  - Ce modèle s'inscrit particulièrement bien dans des environnements distribués et cloud-native, où les technologies de conteneurisation (comme **Docker**) et d'orchestration (comme **Kubernetes**) facilitent la gestion et la scalabilité des services.
  - Les entreprises adoptent les micro-services pour :
    - améliorer leur agilité,
    - réduire le temps de mise sur le marché,
    - mieux gérer la complexité croissante de leurs applications.

6 - Architectures d'applications sur le Cloud

6.1 - Architectures orientées micro-services

- **Les micro-services**
  - Cependant, cette approche introduit également de nouveaux défis, notamment en matière de communication inter-services, de gestion des données distribuées, de supervision, et de sécurité.
    - Elle nécessite donc une bonne maîtrise des outils et des pratiques associées, comme l'observabilité, les architectures event-driven ou encore les mécanismes de tolérance aux pannes.
- **Les architectures orientées micro-services**
  - Elles nécessitent une forte maturité en matière de DevOps et d'automatisation :
    - infrastructure as code (Terraform),
    - pipelines CI/CD (*Continuous Integration / Continuous Deployment*),
    - gestion des secrets,
    - une surveillance proactive des performances et des dépendances réseau.
  - Bien qu'elles améliorent la scalabilité et la maintenabilité à grande échelle, elles complexifient significativement le debugging, la gestion des transactions distribuées et la gouvernance globale du système.

6 - Architectures d'applications sur le Cloud

6.2 - NFS et stockage large échelle

- **Voir :**
  - Chapitre 2 - Applications client-serveur dans internet, partie 2.10 - NFS ; Network File System
  - Chapitre 5 - Systèmes distribués, partie 5.2 - Partage de données dans le cloud

6 - Architectures d'applications sur le Cloud

6.3 - Exemple du Cloud Microsoft Azure

- **Microsoft Azure**
  - Microsoft Azure est une plateforme de cloud computing public.
    - Lancée en 2010 (sous le nom de Windows Azure) ;
    - Elle fut renommée **Microsoft Azure** en 2014 ;
    - La plateforme Azure repose sur l'infrastructure de Microsoft contenant plus de 4 millions de serveurs et plus de 100 centres de données dans 36 régions à travers le monde.
    - Certains utilisateurs trouvent la plateforme plus intuitive s'ils ont déjà une expérience des outils et des pratiques de développement Microsoft.
    - Azure propose des modèles de services basés sur IaaS et PaaS.
  - Azure propose une gamme de services cloud, y compris le calcul, l'analytique, le stockage, le réseau et l'IA.
    - Azure Virtual Machines,
    - Azure Blob Storage,
    - Azure SQL Database,
    - Azure Cognitive Services,
    - Azure Kubernetes Service,
    - Azure Active Directory, devenu **Microsoft Entra ID** (IAM, gestion des identités et des accès),
    - etc.

6 - Architectures d'applications sur le Cloud

6.3 - Exemple du Cloud Microsoft Azure

- **Microsoft Azure**
  - Azure fournit plus de 200 cloud services, principalement sous trois modèles :
    - IaaS (Infrastructure as a Service) : ressources informatiques à la demande (machines virtuelles, réseau, stockage)
    - PaaS (Platform as a Service) : plateforme pour développer/déployer des applications sans gérer l'infrastructure
    - SaaS (Software as a Service) : applications cloud complètes (ex. Microsoft 365)
      Les clients choisissent parmi ces services pour développer et mettre à l'échelle de nouvelles applications, ou exécuter leurs applications existantes, dans le cloud.
  - Un ensemble d'API permet d'utiliser et d'accéder à Azure et aux services associés.
    - Au travers d'un portail web qui permet de gérer l'ensemble des services Azure.
    - Cependant, pour une utilisation plus opérationnelle, il est recommandé d'utiliser « Azure PowerShell » qui permet d'effectuer des actions plus granulaires, scriptables et pouvant être regroupées dans une boîte à outils.
  - Azure aide les entreprises à gérer et à déployer des applications à l'échelle mondiale avec facilité et flexibilité.

6 - Architectures d'applications sur le Cloud

6.3 - Exemple du Cloud Microsoft Azure

- **Microsoft Azure**
  - Services principaux d'Azure

Catégorie	Services clés
Compute	Machines virtuelles (Windows Server, Linux), Azure Containers, Azure Kubernetes Service (AKS), Web Apps
Stockage	Azure Storage (blobs, tables, queues, fichiers), Stockage Premium (SSD), lecteurs VHD
Bases de données	Azure SQL Database, MySQL, Cosmos DB (NoSQL), bases de données en tant que service
Network	Réseau virtuel (VNet), VPN, Azure Traffic Manager, Load Balancer
Identité	Entra ID (anciennement Azure Active Directory) : authentification unique (SSO), authentification multi-facteur, gestion utilisateurs
IA & Machine Learning	Azure AI, Azure Machine Learning, Azure OpenAI Service
Analytics	Azure HDInsight (Hadoop), analyse de données, Big Data
Sécurité	Azure Security Center (surveillance des vulnérabilités), chiffrement des données au repos

- **Microsoft Azure**
  - Les principaux **autres acteurs** sont :
    - AWS, Amazon Web Services,
    - GCP, Google Cloud Platform,
    - OCI, Oracle Cloud Infrastructure,
    - IBM Cloud,
    - Alibaba Cloud.

- Voir :
  - azure.microsoft.com
  - Débuter avec Azure - Youtube
  - Microsoft Azure : tout savoir sur la plateforme de Cloud public - lebigdata.fr
  - Introduction aux services de cloud Microsoft Azure - Stanislas Quastana