

# Cloud et Systèmes Web



## OBJECTIFS DU MODULE

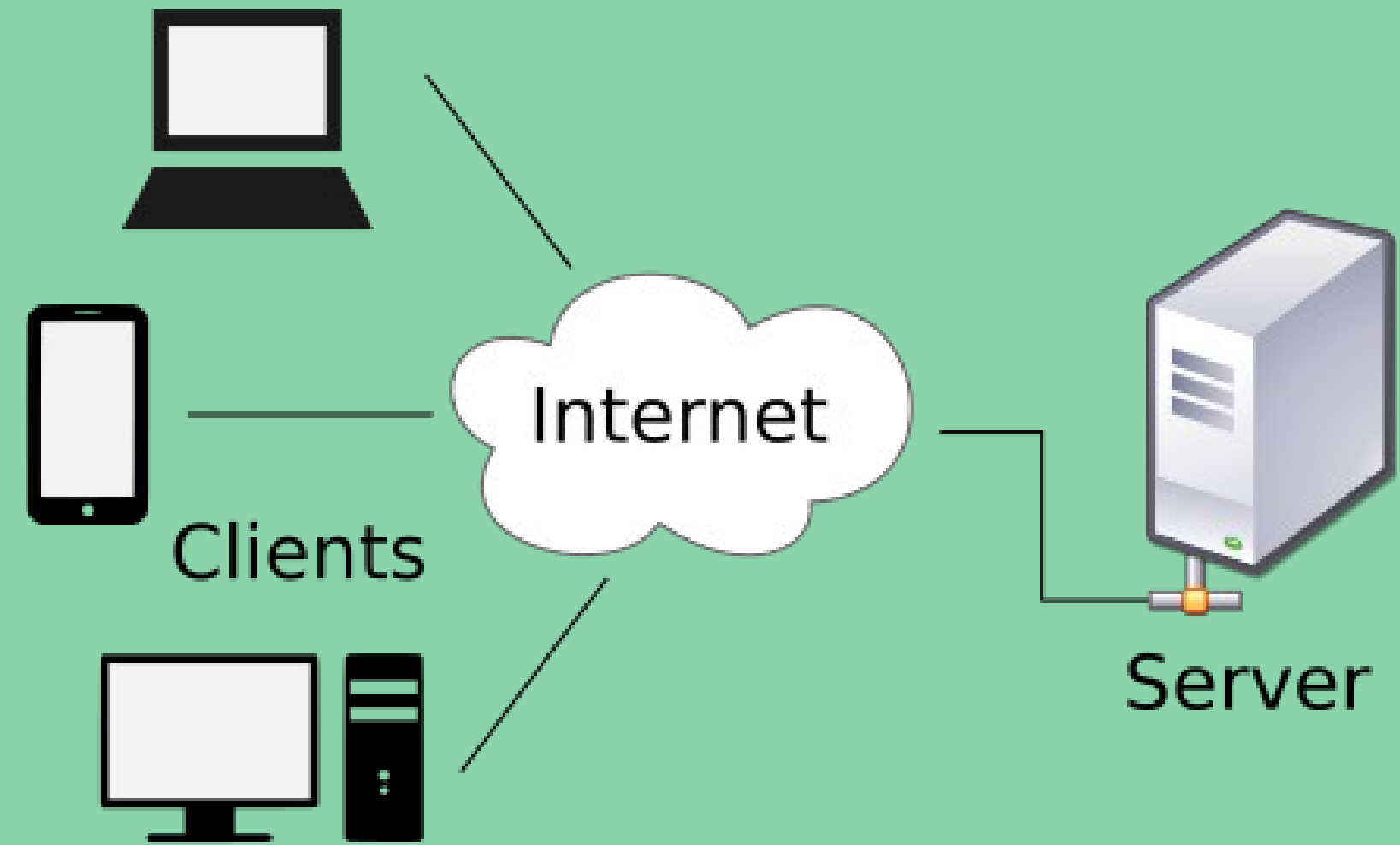
- Comprendre une offre cloud : infrastructure, plateforme...
- Sensibiliser à la dépendance des applications vis-à-vis d'une offre cloud
- Connaître les impacts des techniques du cloud dans la conception des sites actuels
- Connaître les services du web en mode SaaS.

# Standard et protocoles Web


Qu'est-ce qu'Internet ?

Qu'est-ce que le World Wide Web ?

Quelles normes et protocoles permettent  
leur utilisation ?



# ARCHITECTURE CLIENT/SERVEUR

 Mode de communication à travers un réseau entre plusieurs programmes : l'un, qualifié de client, envoie des requêtes ; l'autre ou les autres, qualifiés de serveurs, attendent les requêtes des clients et y répondent

# CLIENT / SERVEUR VS PEER TO PEER

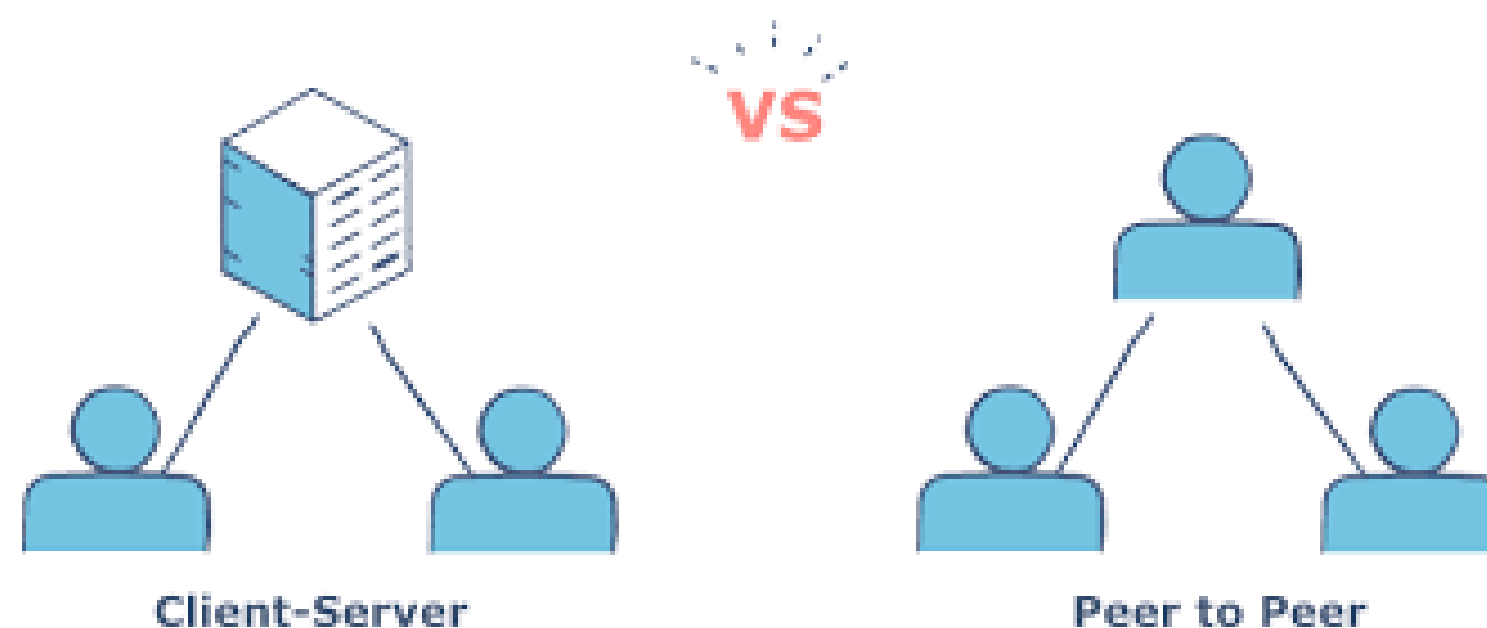
Les applications (partie visible pour l'utilisateur) sont la raison d'être des réseaux informatiques.

Leur nombre et évolution depuis les débuts d'Internet est considérable :

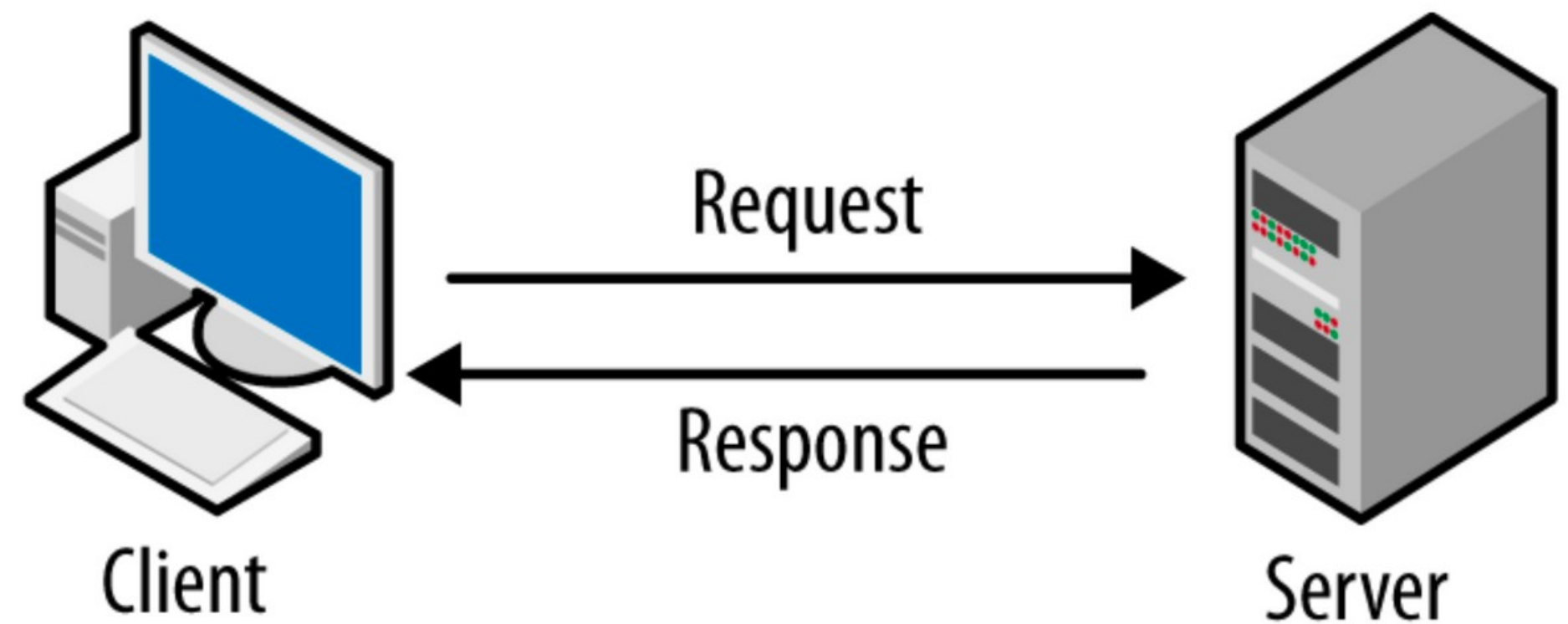
- sous forme textuelle au début : messagerie électronique, terminaux distants, transferts de fichiers, ...
- principalement multimédia aujourd'hui : diffusion vidéo à la demande, streaming, VoIP, ...

Ces applications communiquent en utilisant l'un des deux modèles d'architecture du Web :

- Point-à-point ("peer to peer") : chaque application est connectée à l'ensemble des autres applications
- Client-serveur : les applications utilisateur (client ou "frontend") interagissent uniquement avec un ou des serveurs partagés par les utilisateurs ("backend")



# ARCHITECTURE CLIENT / SERVEUR



Les processus client / serveur ne sont pas identiques : ils communiquent pour réaliser le traitement des données

- Le serveur est à l'écoute d'une requête cliente éventuelle
- Le client initie l'échange
- Le service (traitement des données) est effectué par le serveur
- Le client réceptionne le résultat et l'affiche à l'utilisateur

# LE SERVEUR

Tourne en permanence, attendant des requêtes.

Peut répondre à plusieurs clients en même temps.

Nécessite :

- Une machine robuste et rapide, qui fonctionne 24h/24  
alimentation redondante, technologie RAID, stockage et sauvegarde des données
- La présence d'administrateurs systèmes et réseau pour gérer les serveurs.



# EXEMPLES DE SERVEURS



Base de données



Serveur d'impression



# LE CLIENT

Son instance est dédiée à la machine de l'utilisateur...  
...mais peut être partagée par plusieurs comptes ou profils !

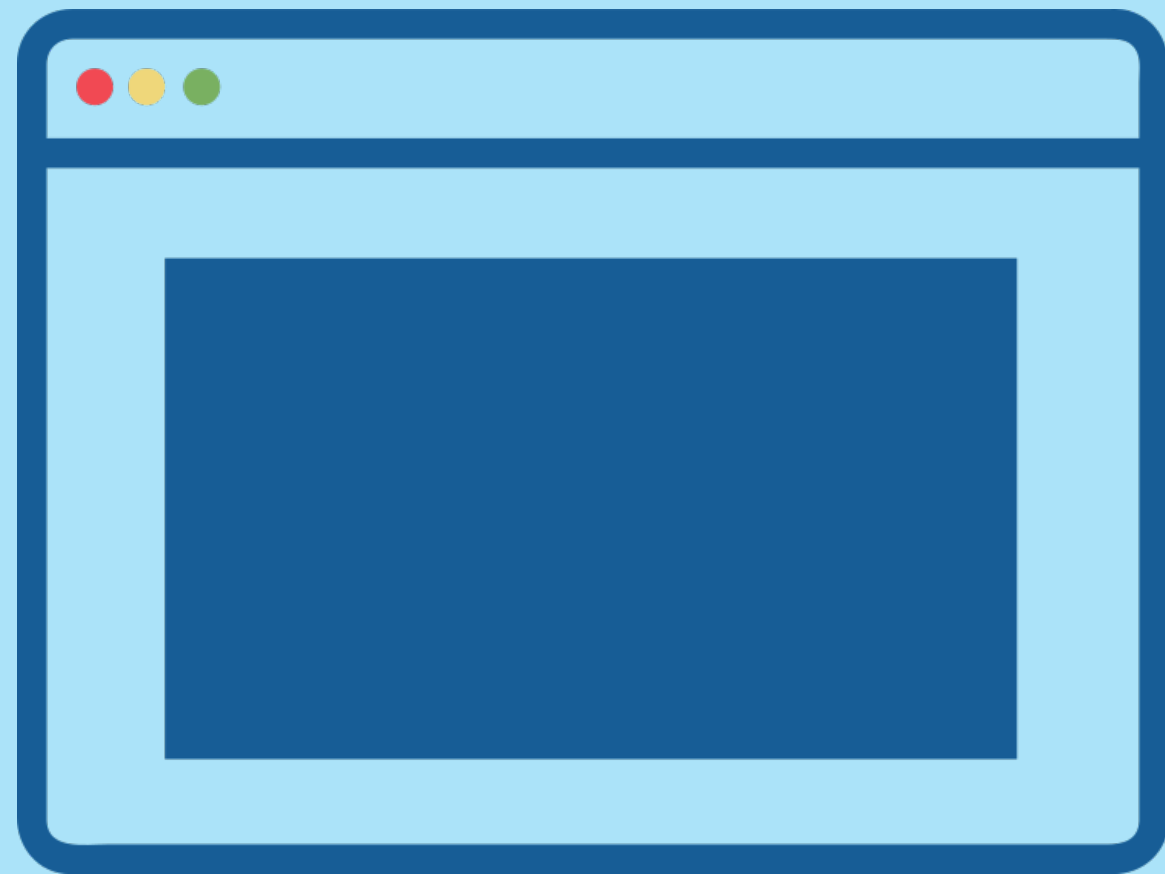
Il est démarré par l'utilisateur seulement pour le temps d'utilisation de l'application.

Dépend des ressources de la machine de l'utilisateur :

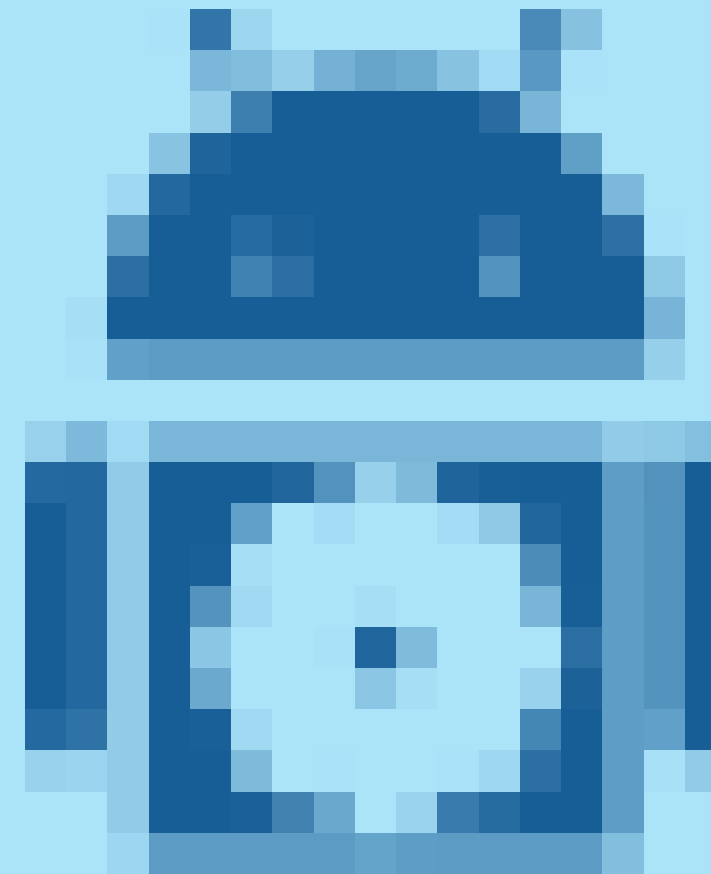
- Limitation mémoire, bande passante, CPU, ...
- Dépendences peu maîtrisées : Android vs iPhone, version des librairies, ...



# EXEMPLES DE CLIENTS

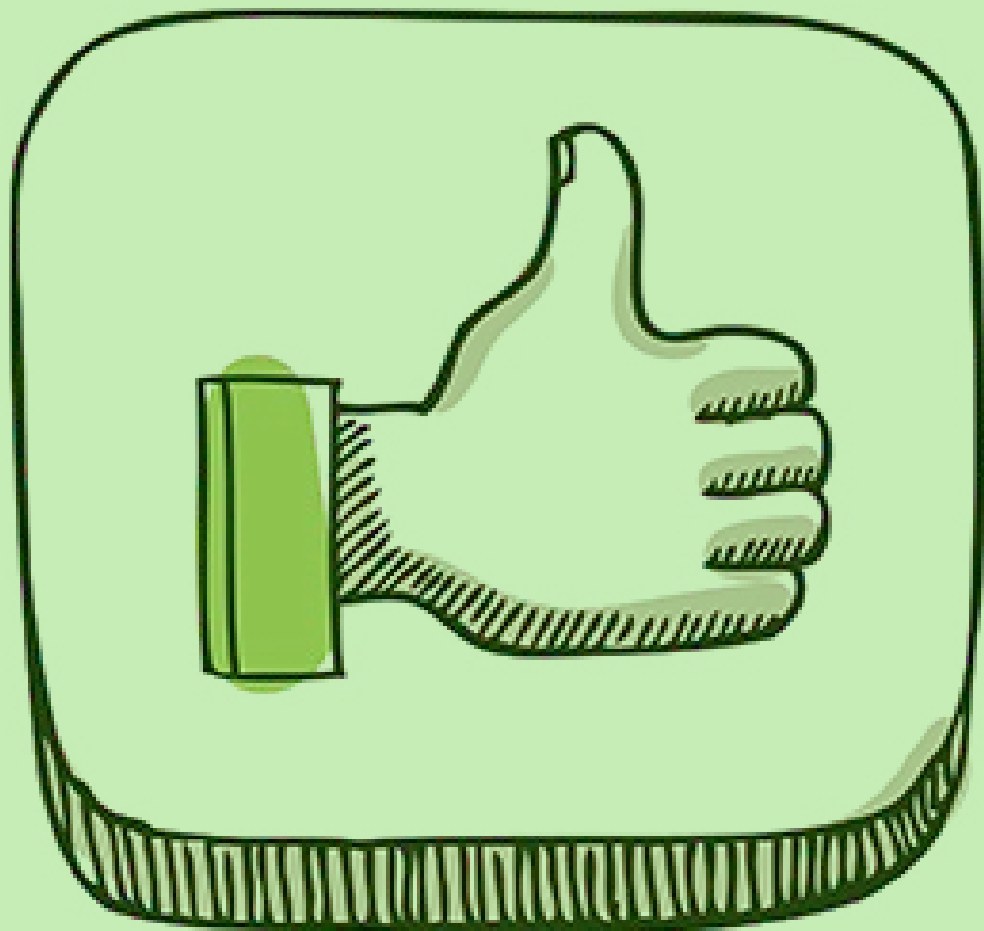


Navigateur web



Client Android

# AVANTAGES DE L'ARCHITECTURE CLIENT/SERVEUR



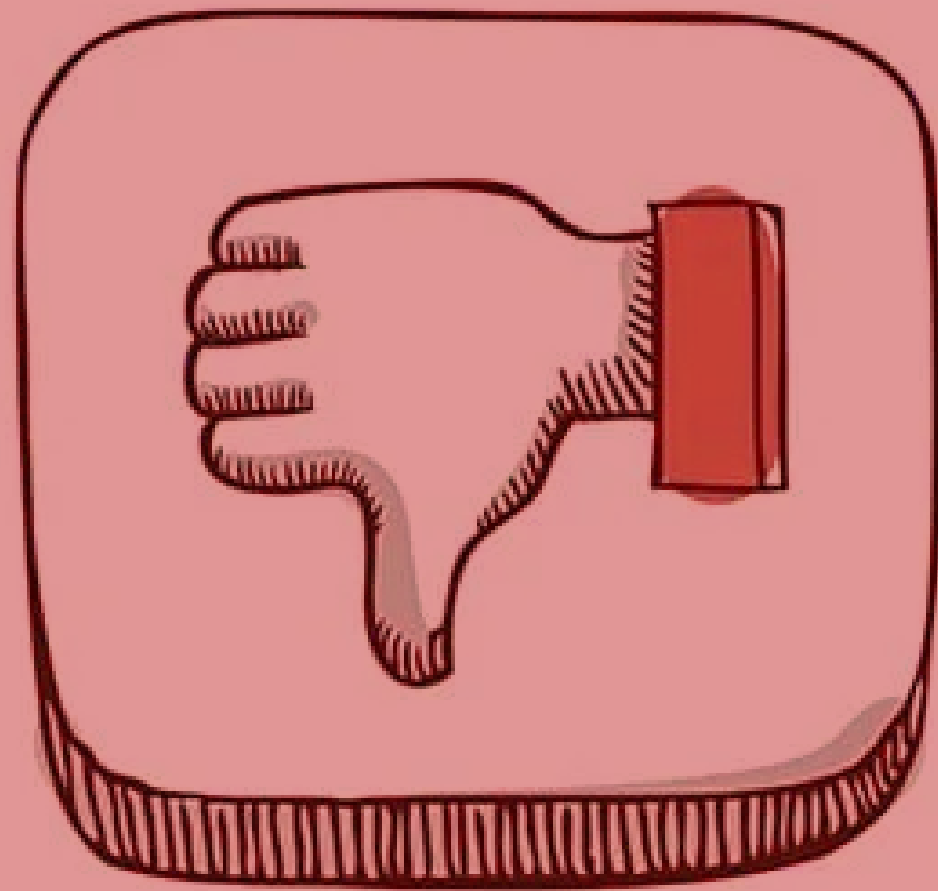
Ressources centralisées : le serveur gère les ressources communes à tous les utilisateurs (base de données, ...)  
la cohérence des données est aisée

Administration centralisée : seul le serveur est critique

Réseau évolutif : ajout / suppression de clients sans impact sur les autres clients

Sécurité : l'accès aux données est masqué par le serveur, les clients en manipulent le minimum

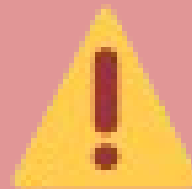
# INCONVÉNIENTS DE L'ARCHITECTURE CLIENT/SERVEUR



Maillon faible : tout le réseau est architecturé autour du serveur celui-ci doit avoir une forte tolérance aux pannes

Coût élevé (ressources et maintenance) dû à la technicité du serveur

Communication directe impossible entre les clients

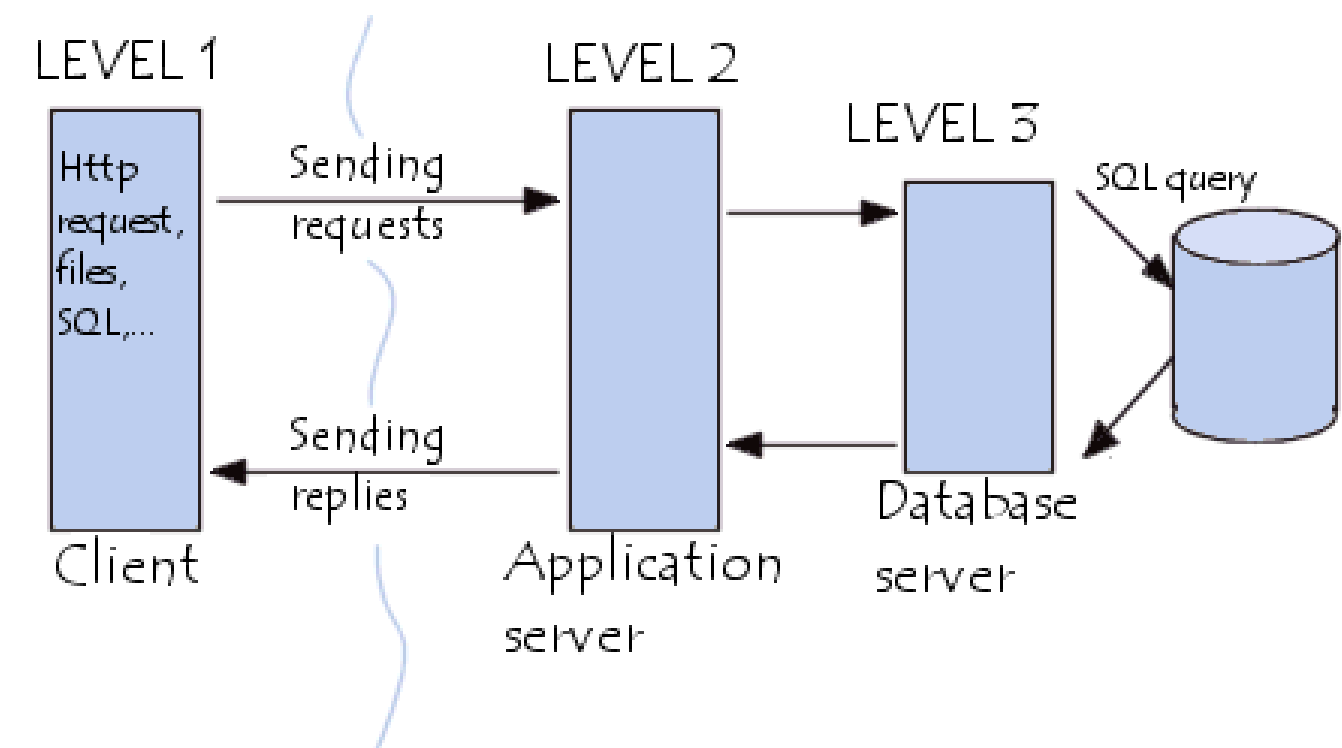


Le serveur est donc le seul élément critique mais sa maintenance est compliquée et coûteuse.

# DIFFÉRENTES ARCHITECTURES CLIENT/SERVEUR



Il est possible de déléguer plus ou moins de traitements au serveur. On parlera alors de client lourd si le client effectue une majorité des opérations vs client léger si le serveur réalise la majorité des traitements.



Il est possible de réaliser une architecture à N niveaux en déléguant des traitements à des sous-serveurs masqués au client. Cela permet de segmenter l'utilisation de ressources ou services et leur administration

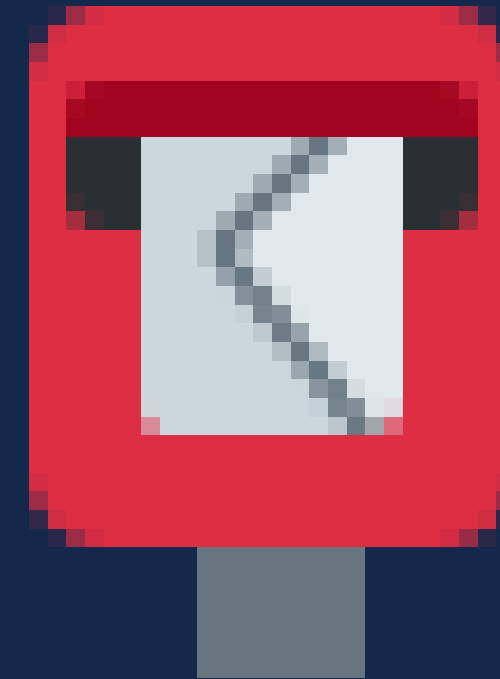
# Les protocoles

# Les protocoles

# DES CONVENTIONS AUX PROTOCOLES



Exemple lors de l'envoi d'un courrier postal :  
Quelles sont les conventions respectées par les  
différent acteurs ?



Il en va de même pour l'envoi d'un courrier électronique ou lorsqu'un navigateur affiche une page d'un site web : la communication a alors lieu entre deux ordinateurs, et cette communication est possible parce qu'un ensemble de conventions sont respectées.



Pour des conventions de communication  
entre machines, on utilise le terme  
"protocole"



Dans une architecture client/serveur, le  
serveur joue le rôle de "La Poste" entre les  
clients

# Internet

Internet est un réseau mondial regroupant un ensemble de sous-réseaux non-homogènes d'architectures et de technologies différentes.

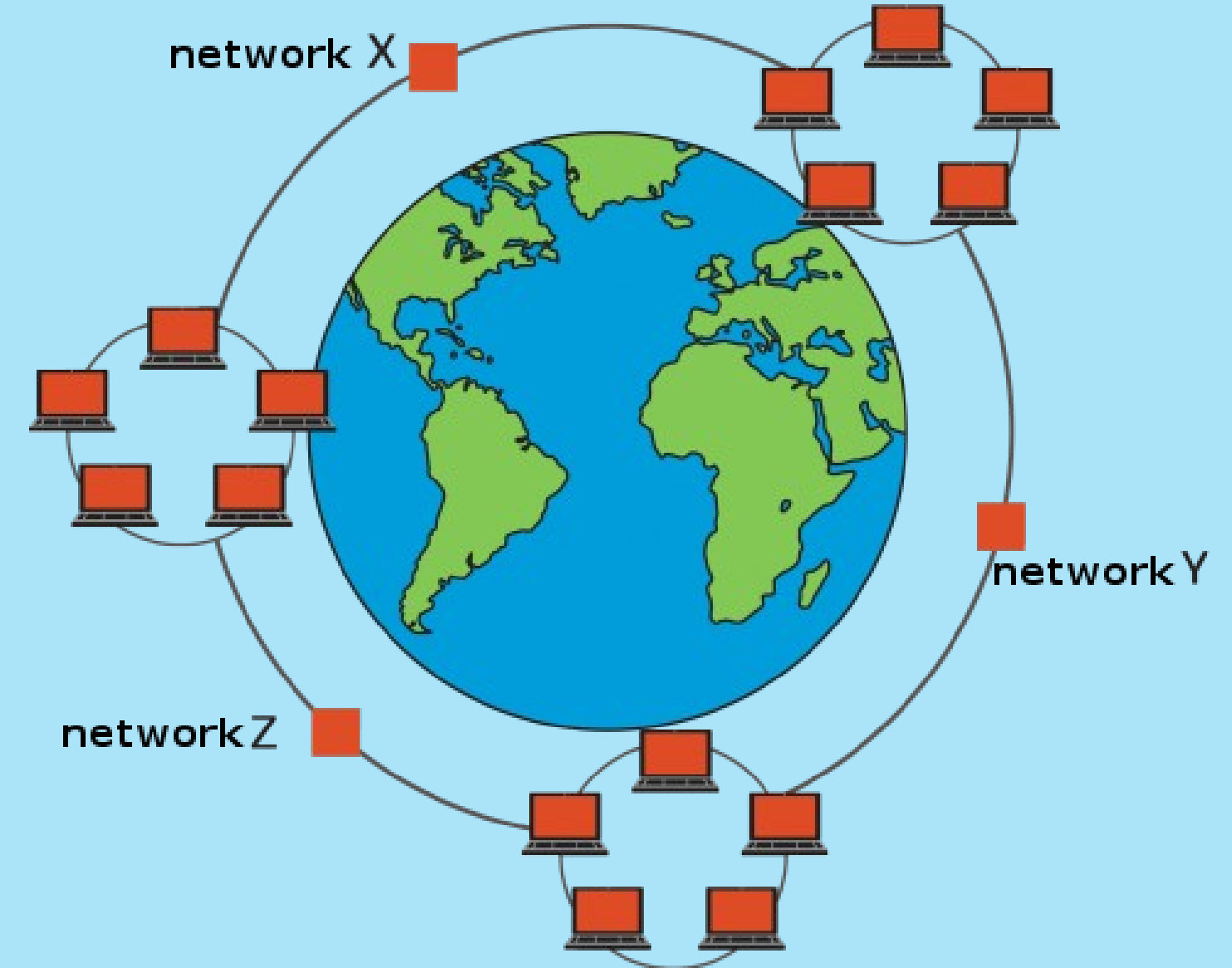


Internet est donc bien plus qu'un réseau... c'est un réseau de réseaux !

Son nom vient d'ailleurs d' "inter-network" qui décrit bien cette idée.



Comment faire alors pour permettre la communication entre des entités de réseaux très différents et n'utilisant pas les mêmes technologies ?





# LE MODELE OSI : SÉPARER LES COUCHES RÉSEAU



Le modèle OSI est une norme précisant comment les machines doivent communiquer entre elles.

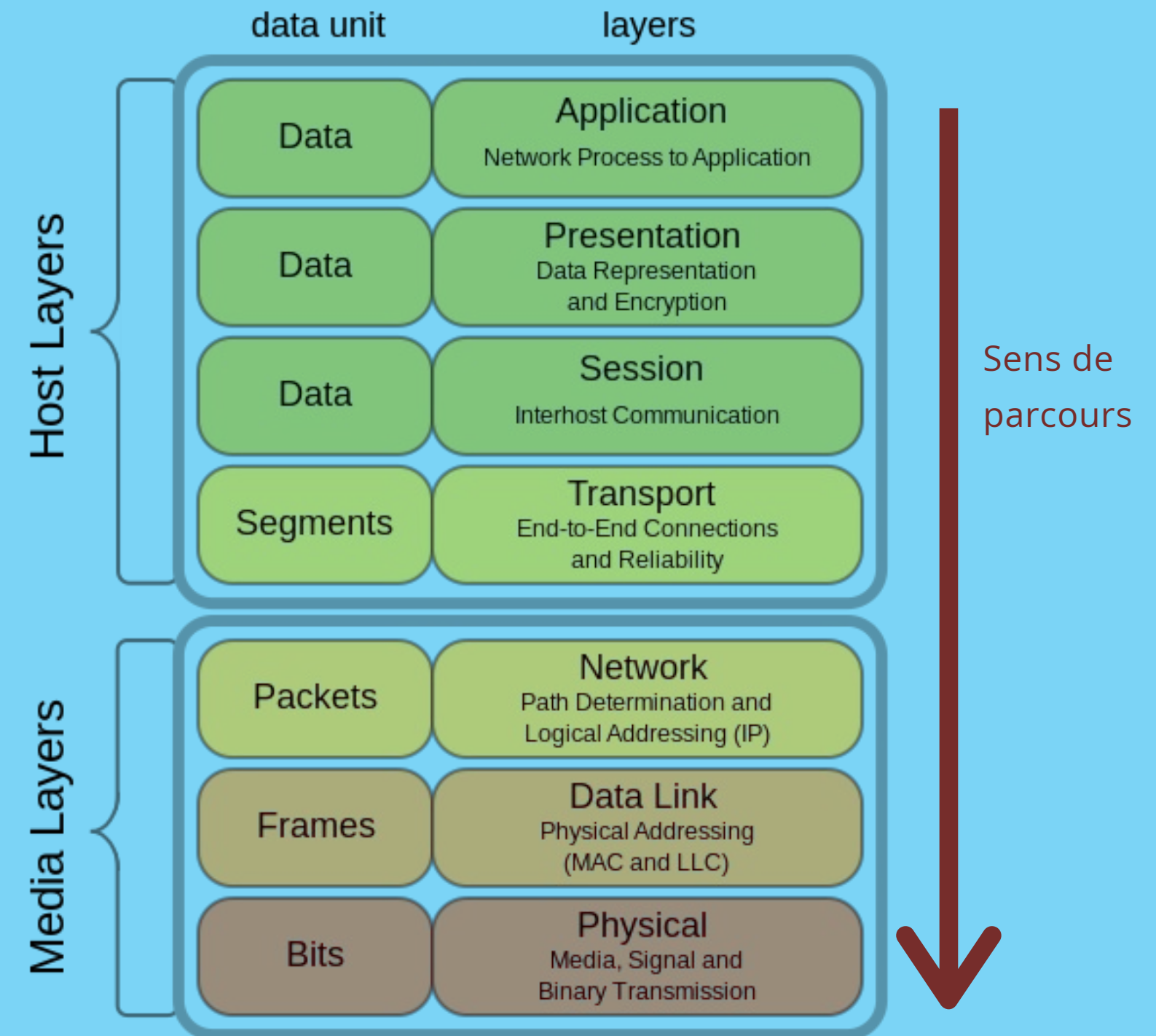
C'est un modèle de 7 couches indépendantes, ayant chacune un rôle particulier.

Chaque couche ne peut communiquer qu'avec une couche adjacente.



Le modèle OSI est un modèle théorique utile pour segmenter les couches réseau.

En pratique, on utilise principalement le modèle TCP/IP



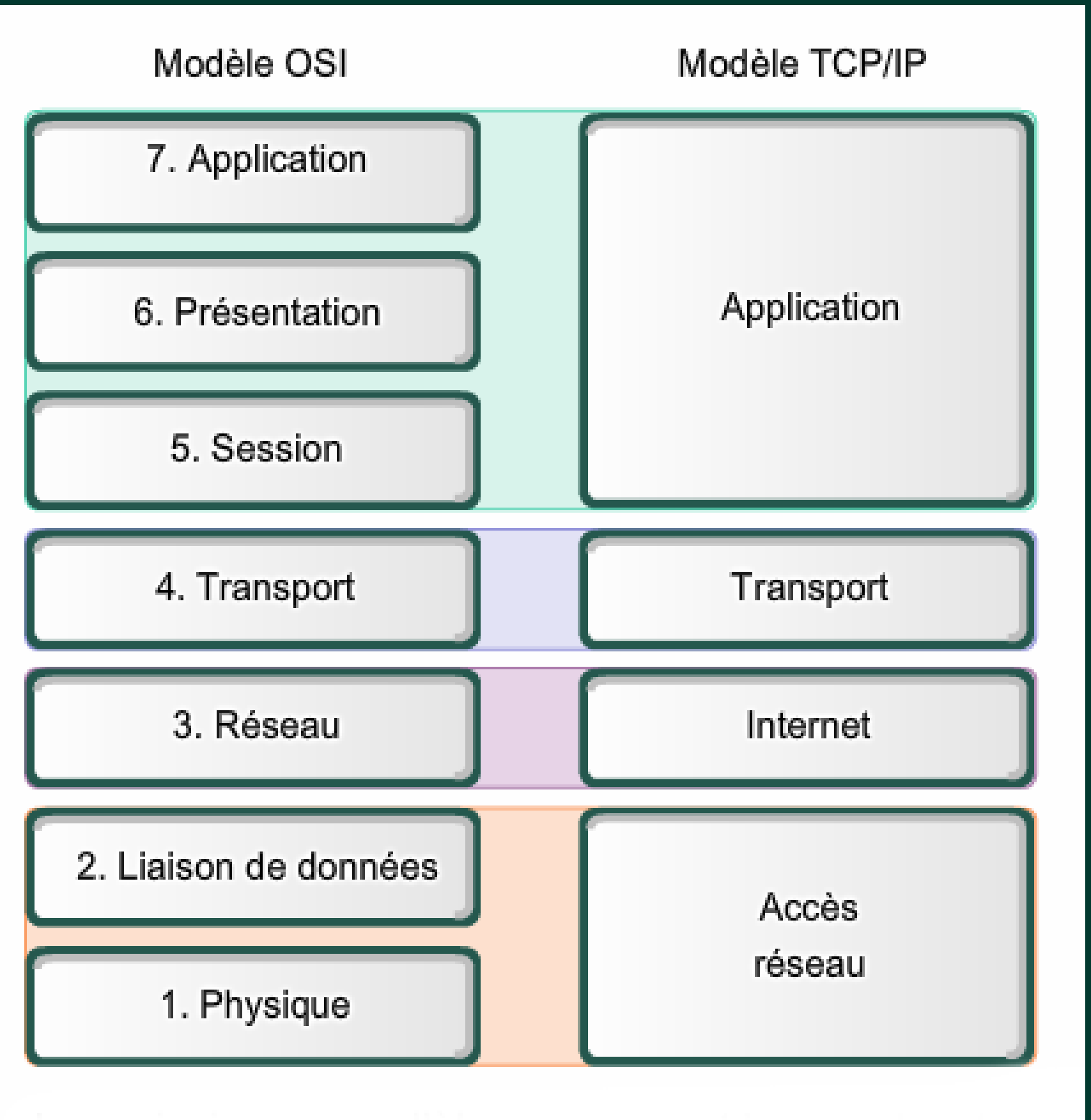
# LE MODELE TCP/IP



Le modèle TCP/IP est le modèle utilisé très majoritairement dans la pratique.

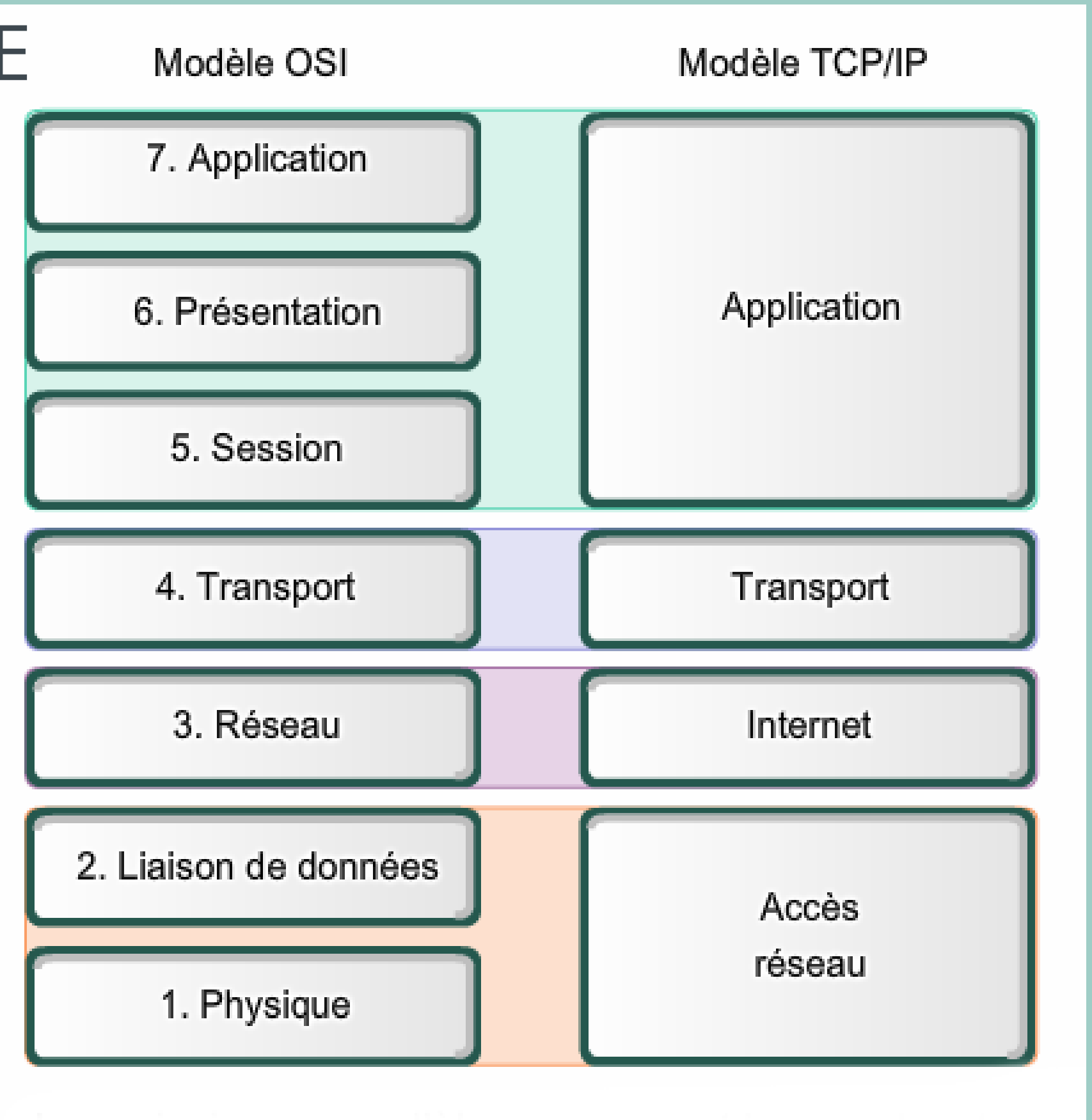
C'est un modèle de 4 couches fusionnant certaines des couches OSI.

Chaque couche possède ses propres protocoles et peut être remplacée indépendamment des autres couches.



# LE MODELE TCP/IP : COUCHE APPLICATION

Les protocoles de la couche application gèrent l'accès applicatif aux services réseau : gestion des noms de domaines (DNS), configuration dynamique des postes (DHCP), Hypertext Transfert Protocol (HTTP), transfert de fichiers (FTP), transfert de mails (SMTP), ...



# LE MODELE TCP/IP : COUCHE TRANSPORT

Les protocoles de la couche transport établissent les communications :

- de manière connectée en utilisant des sessions
- ou en diffusant les messages sans connexion

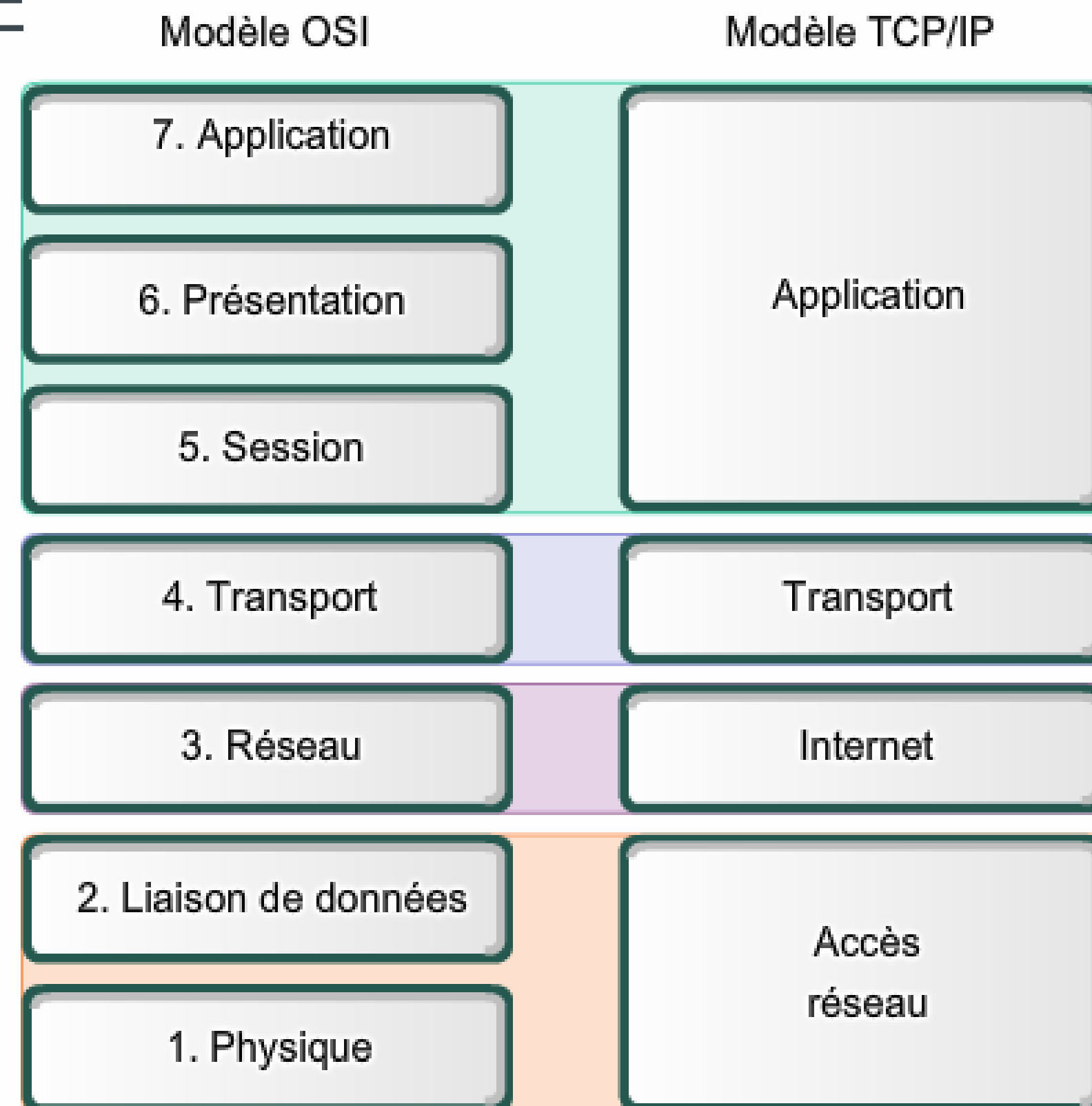
Les communications par session sont réalisées par le Transmission Control Protocol (TCP).

La diffusion directe de message sans connexion est réalisée par le User Datagram Protocol (UDP).



TCP est bien plus fiable qu'UDP grâce à l'usage de sessions mais plus coûteux.

Pour les applications à destination des utilisateurs finaux, on réservera UDP aux applications réalisant des échanges massifs de données non critiques (streaming vidéo, ...)

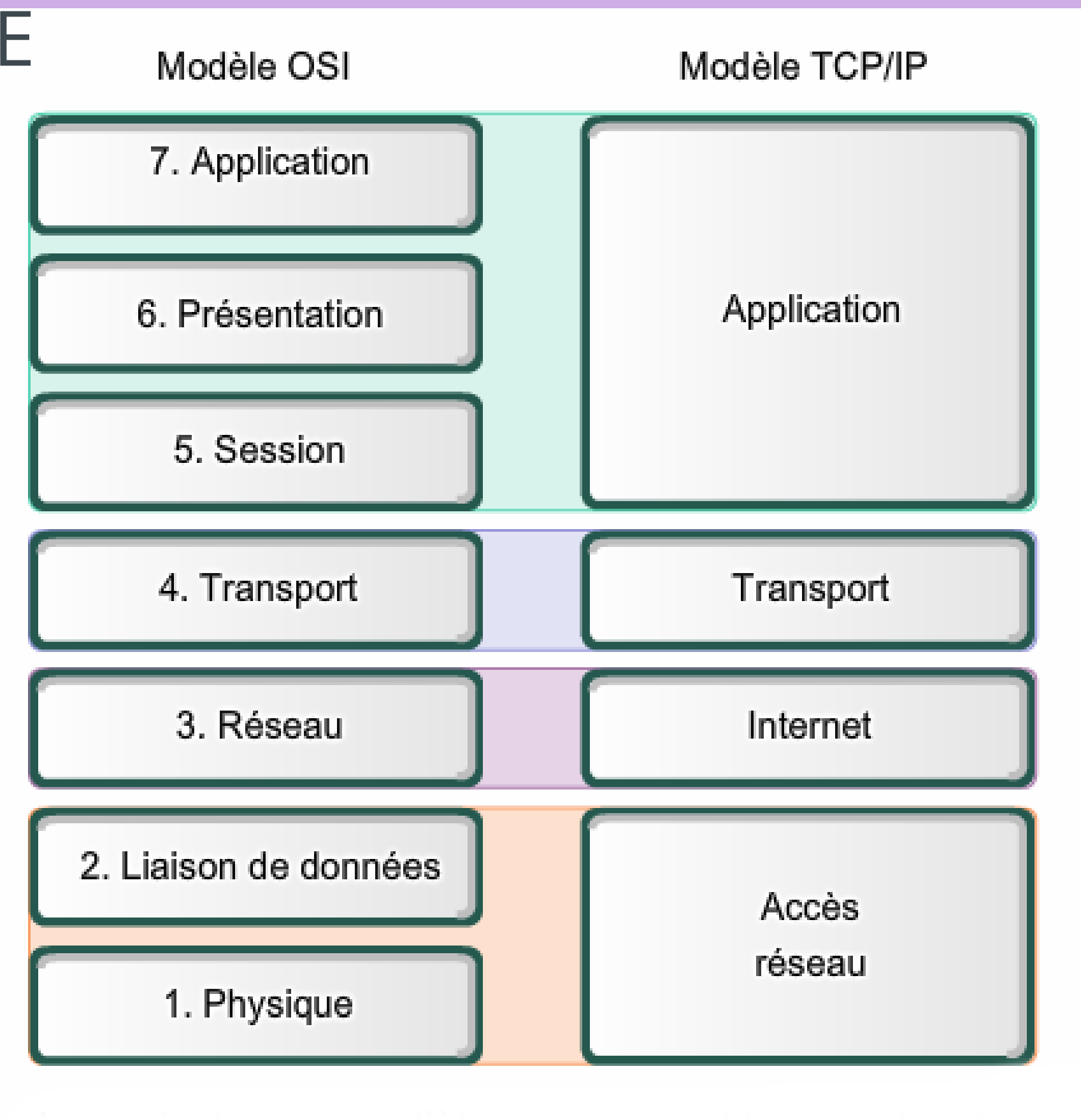


# LE MODELE TCP/IP : COUCHE INTERNET

Les protocoles de la couche internet gère l'acheminement (routing) et l'encapsulation des données au travers de paquets Internet Protocol (IP).



Cette couche gère aussi les protocoles de résolution d'adresse (ARP) et de communication entre équipement réseau (ICMP, utilisé par exemple lors d'un "ping" ).



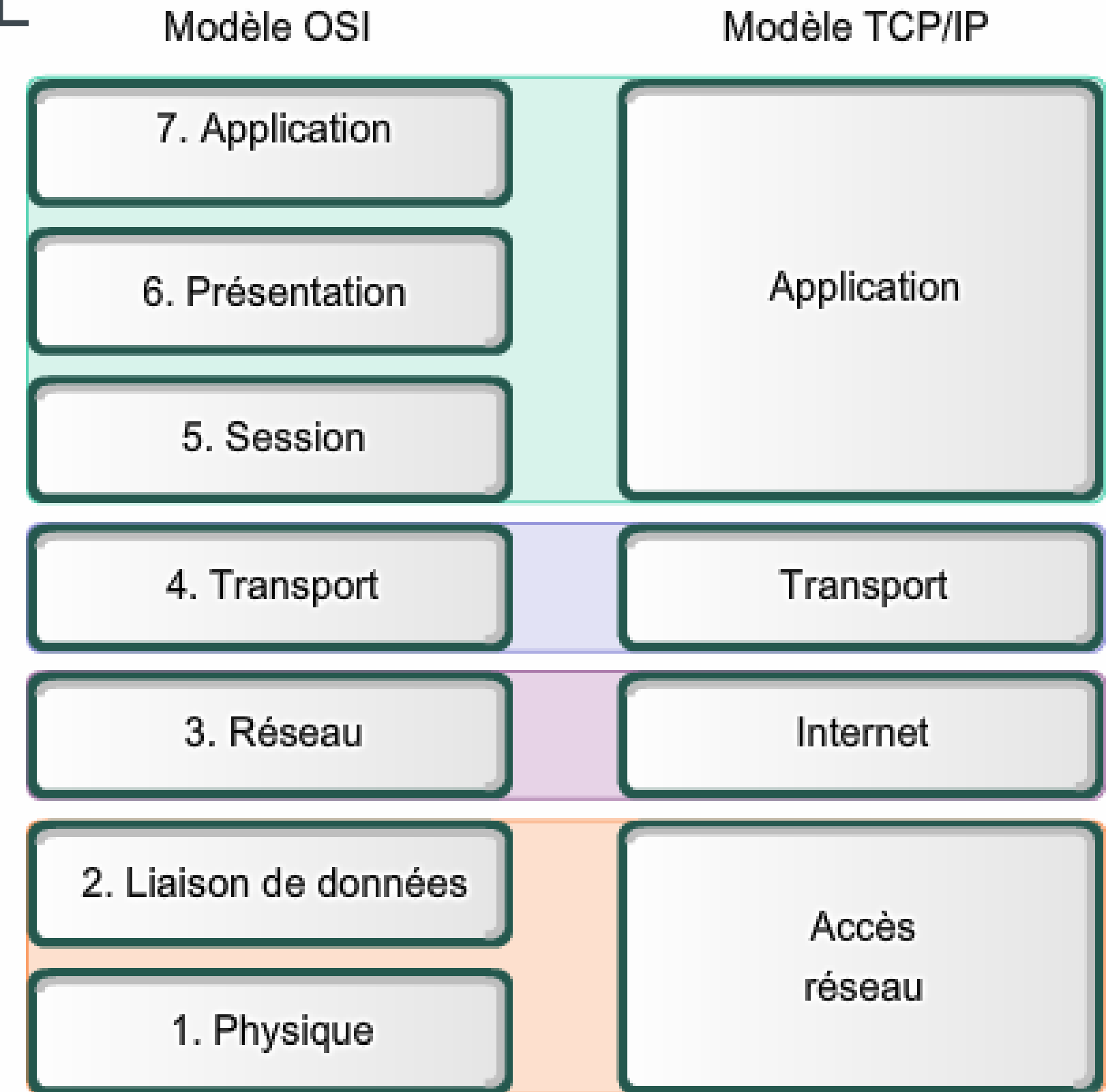
# LE MODELE TCP/IP : COUCHE RÉSEAU

Les protocoles de la couche réseau font transiter les frames de données sur le réseau.



Ces protocoles peuvent être variés : ils incluent les différents réseaux locaux (LAN) : Ethernet, Token Ring, ..., les technologies de télécommunications sur les réseaux globaux (WAN) : POTS, ISDN, ATM, ...

Leur configuration est gérée par les administrateurs réseaux et est généralement masquée aux développeurs.



# IP

Pour envoyer un courrier postal, l'expéditeur inscrit l'adresse du destinataire dans un format prédéfini (rue, code postal, ville). De la même manière, il est nécessaire de définir un format pour identifier une entité sur le réseau.

TCP / IP utilise 2 schémas de nommage pour identifier les machines (hosts) et les réseaux dans un internetwork :



Les adresses IP : ce sont des adresses logiques sur 32 bits (4 octets) de la forme w.x.y.z

Celles-ci sont partitionnées en 2 segments grâce à un "masque de sous-réseau" (subnet mask) : un ID réseau (NetID) suivi par un ID de machine (HostID).

Les adresses IP sont la clé primaire pour identifier une machine ou un réseau.

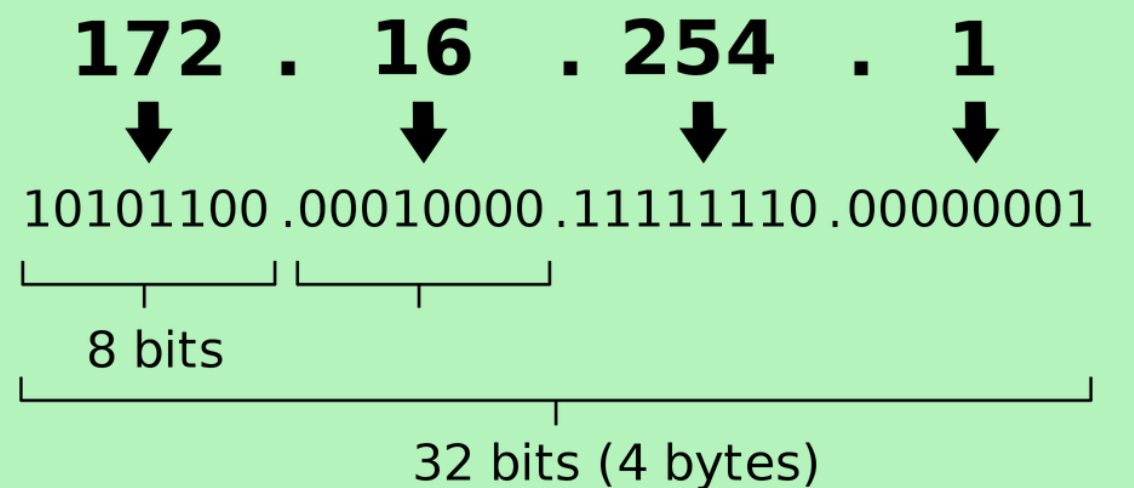


Exemple :

L'adresse IP 205.116.8.44 est partitionnée par le masque 255.255.255.0 en :

- un NetID : 25.116.8.0
- un HostID : 44

IPv4 address in dotted-decimal notation



# IP



Une autre notation possible pour le masque de sous-réseau est la notation décimale.

Cette notation indique le nombre de bits réservés par le nom du réseau dans l'adresse IP.

Par exemple, le masque 255.255.255.0 réserve les 3 premiers octets (=3x8 bits) au réseau (et le dernier octet à l'identification de l'hôte dans le réseau). Ce masque peut donc aussi s'écrire /24



Chaque réseau possède 2 adresses particulières :

- l'adresse avec un HostID = 0 est l'adresse identifiant le réseau
- l'adresse avec un HostID dont tous les bits sont à 1 est l'adresse de diffusion (broadcast). Cette adresse sert à diffuser un message à tous les hôtes sur le réseau.

Par exemple, pour le réseau 25.116.8.0/24 :

- l'adresse de réseau est : 25.116.8.0
- l'adresse de broadcast est : 25.116.8.255



# IPv6



Les adresses IPv6 : Les adresses de la forme w.x.y.z sont en réalité des adresses de la version IPv4.

C'est la version ultra majoritaire dans les déploiements actuels.

L'épuisement des adresses IPv4 sur 32 bits a conduit à la création d'une nouvelle norme : IPv6. Cependant, sa complexité de mise en oeuvre en fait un cas d'utilisation encore rare aujourd'hui.




Les adresses IPv6 ne s'écrivent pas en notation décimale 192.168.0.1 mais en notation hexadécimale (8 groupes de 2 octets soit 128 bits).


Par exemple : 2001:0db8:0000:85a3:0000:0000:ac1f:8001

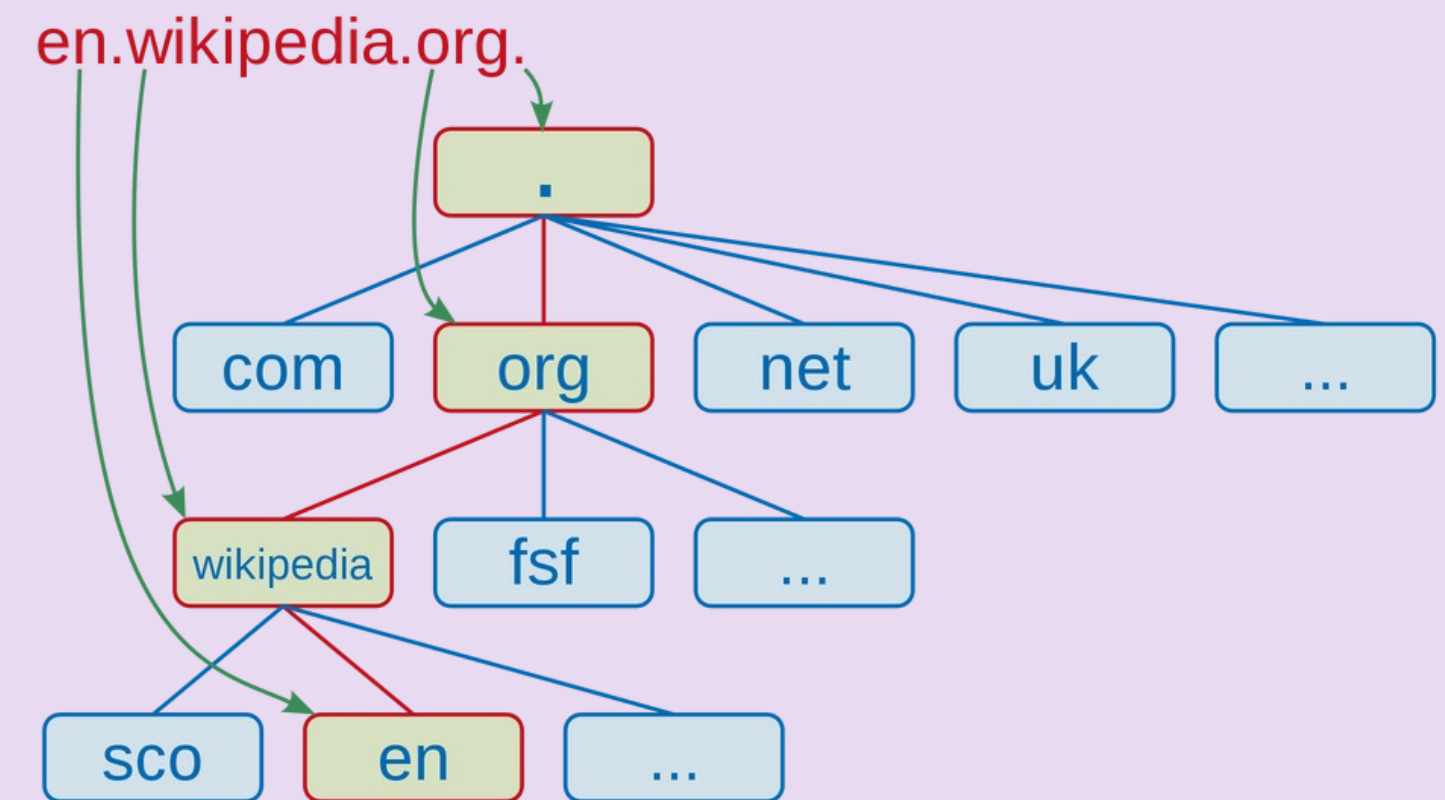


Il est possible de mélanger des déploiements de chaque version, une interface réseau peut même posséder une adresse IPv4 et une adresse IPv6 en même temps.

# Nom de domaine

 Les noms de domaine (FQDN : Fully Qualified Domain Name) : ce sont des noms alphanumériques de la forme `nom_de_machine.nom_de_domaine` où `nom_de_domaine` est un nom DNS (système de nommage hiérarchique mondialisé).

 Exemple :  
Le nom de domaine `server12.microsoft.com` représente la machine `server12` qui appartient à un réseau nommé `microsoft` qui est un sous-réseau du réseau `com` (lui-même sous-réseau du réseau racine : `."` (dot))



# DNS

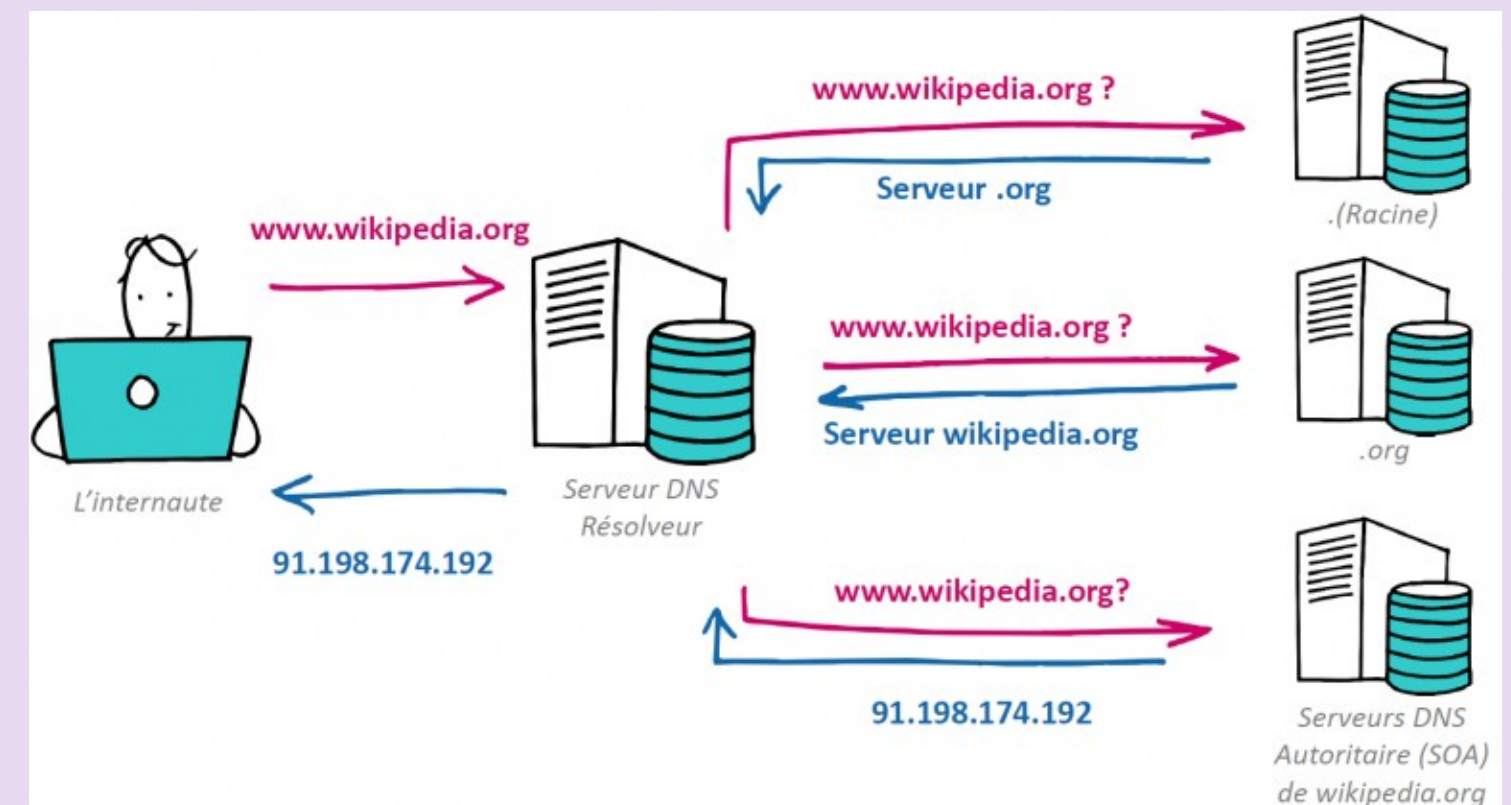


Le protocole Domain Name System (DNS) est le service informatique distribué utilisé pour traduire les noms de domaine Internet en adresse IP ou autres enregistrements

Les noms de domaine sont plus simples à retenir pour un humain et plus parlant (logs, ...) mais les machines communiquent essentiellement par adresses IP.



DNS est un service hiérarchique : les serveurs DNS de 1er niveau (TLD : Top Level Domain) sont connus de tous, et délèguent la résolution de sous-domaines à des serveurs intermédiaires (exemple : .org), et ainsi de suite jusqu'au noeud DNS terminal connaissant l'IP de la machine.



# http://



Le protocole Hypertext Transfer Protocol (HTTP) est un protocole de communication client-serveur développé pour le Web.



HTTP est un protocole de la couche applicative, il n'impose pas de couche de transport mais dans la pratique on utilise TCP. Les clients HTTP les plus connus sont les navigateurs Web permettant à un utilisateur d'accéder à un serveur contenant les données



(HTTP1.0), HTTP 1.1 et HTTP/2 sont les versions utilisées du protocole, HTTP/3 n'étant pas encore bien supporté par les navigateurs.



Par défaut, un serveur HTTP utilise le port 80 (443 pour HTTPS)



HTTP permet l'authentification de l'utilisateur par username/password.

# http://

Dans le protocole HTTP, une méthode est une commande spécifiant un type de requête, c'est-à-dire qu'elle demande au serveur d'effectuer une action. En général l'action concerne une ressource identifiée par l'URL qui suit le nom de la méthode, par exemple :  
GET / HTTP/1.1 Host:www.google.com

GET : C'est la méthode la plus courante pour demander une ressource. Une requête GET est sans effet sur la ressource, il doit être possible de répéter la requête sans effet.

HEAD : Cette méthode ne demande que des informations sur la ressource, sans demander la ressource elle-même.

POST : Cette méthode est utilisée pour transmettre des données en vue d'un traitement à une ressource (le plus souvent depuis un formulaire HTML). Le résultat peut être la création de nouvelles ressources ou la modification de ressources existantes.

OPTIONS : Cette méthode permet d'obtenir les options de communication d'une ressource ou du serveur en général.

CONNECT : Cette méthode permet d'utiliser un proxy comme un tunnel de communication.

TRACE : Cette méthode demande au serveur de retourner ce qu'il a reçu, dans le but de tester et effectuer un diagnostic sur la connexion.

PUT : Cette méthode permet de remplacer ou d'ajouter une ressource sur le serveur.

PATCH : Cette méthode permet, contrairement à PUT, de faire une modification partielle d'une ressource.

DELETE : Cette méthode permet de supprimer une ressource du serveur.



# http://

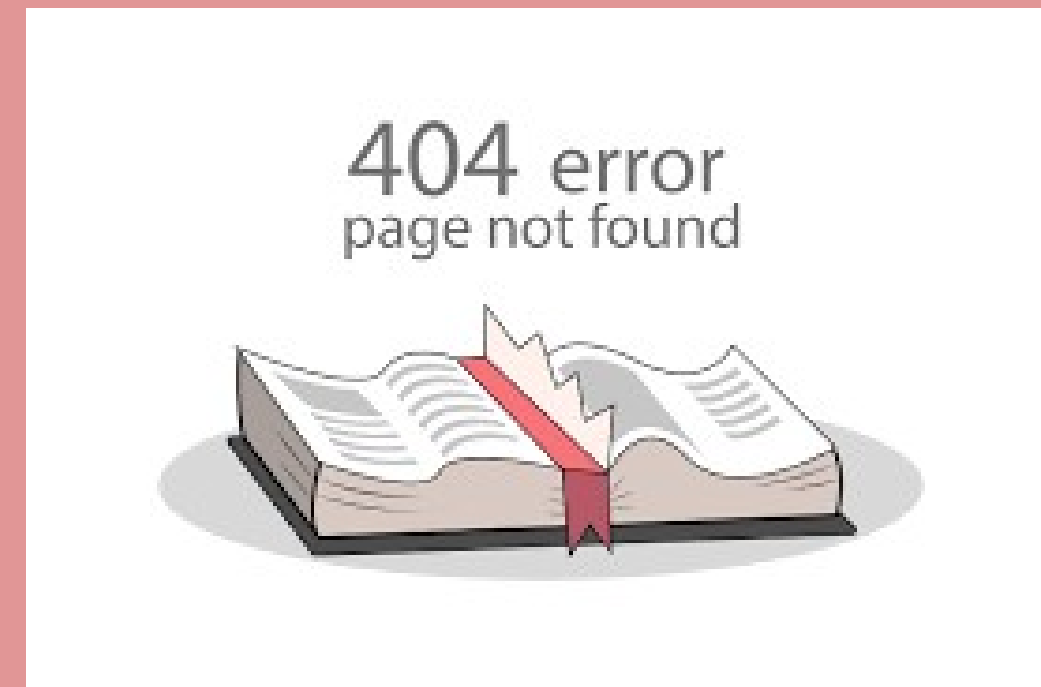
Les réponses aux requêtes HTTP renvoient toutes :

- Un code de status et une description. Ce code permet d'identifier et d'automatiser le traitement des erreurs.
- Un ou plusieurs en-têtes (headers) optionnels
- Un corps de message (body) optionnel de plusieurs lignes pouvant contenir des données binaires

Un codes de status est un nombre à 3 chiffres défini par la norme.

Les codes de status sont regroupés en 5 catégories :

- 1xx – Informational
- 2xx – Successful
- 3xx -Multiple Choice
- 4xx– Client Error
- 5xx -Server Error



# https://



HTTPS (avec S pour secured) est la variante sécurisée de HTTP par l'usage des protocoles Transport Layer Security (TLS).



Ce protocole est en réalité l'utilisation du protocole sécurisé TLS (ou SSL) encapsulant le protocole HTTP dans les données qui transitent.

La sécurité des informations est rendue possible par l'utilisation de certificats SSL.

Le serveur émet un certificat (une clé publique) signée avec sa clé privée : c'est la détention de cette clé privée qui garantit l'identité du serveur.

Les échanges de données sont alors chiffrés en utilisant les clés du serveur et du client, ce qui garantit la confidentialité et l'intégrité des échanges.



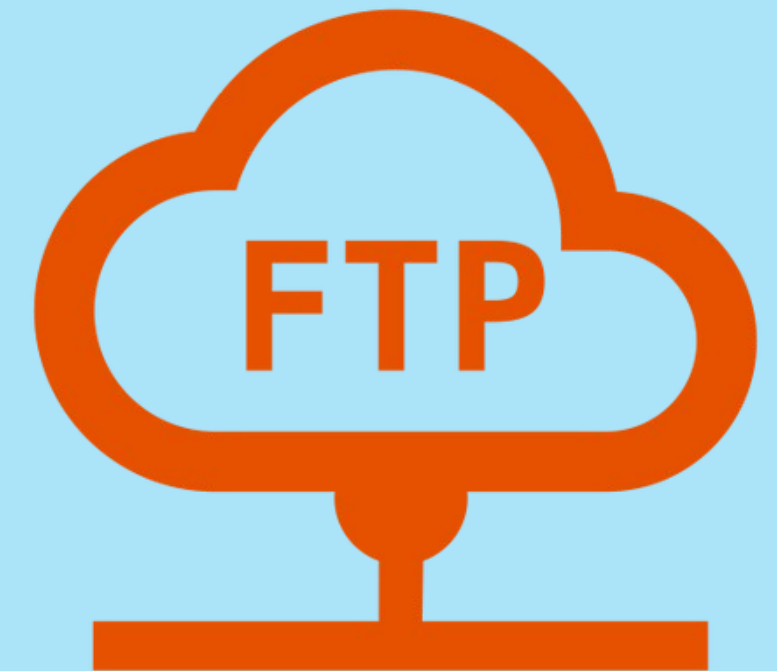
Le certificat du serveur est en principe signé de nouveau avec la clé publique d'une 2e autorité, et ce procédé est appliqué récursivement jusqu'à un signataire de confiance.

Ainsi, le client peut connaître une liste limitée de signataires de confiance, qui en signant le certificat d'un tiers vont lui déléguer leur confiance, et ainsi de suite jusqu'au serveur à atteindre.

# ftp://



Le protocole File Transfer Protocol (FTP) est un protocole de communication client-serveur destiné au partage de fichiers sur un réseau TCP/IP



Comme HTTP on utilise le protocole TCP pour la couche de transport.

Le protocole utilise deux types de connexions TCP :

- Une connexion de contrôle initialisée par le client pour transmettre les commandes concernant les fichiers (suppression de fichiers, renommage, liste des fichiers...).
- Une connexion de données initialisée par le client ou le serveur pour transférer les données requises (contenu des fichiers, liste de fichiers).



SFTP (avec S pour secured) est la variante sécurisée de FTP par l'usage des protocoles TLS ou SSL.



Par défaut, un serveur FTP utilise le port 21 pour les commandes. Les données circulent sur le port 20 (mode actif) ou un port aléatoire du serveur (mode passif).



# ws://



Le protocole WebSocket (ws) permet d'établir sur les ports web standards une connexion permanente et bidirectionnelle entre le client et le serveur distant.

Il devient alors possible de procéder à l'échange de données instantanément, hors du mode requête/réponse classique de http, et même à du « push » de données du serveur vers le client, sans que ce dernier n'ait émis la moindre requête.

Comme HTTP et FTP, on utilise le protocole TCP pour la couche de transport.



Ce protocole est très utile pour des application nécessitant des interactions entre les 2 parties (visio-conférence, ...) ou pour échanger de grande quantités de données non critiques (streaming, ...)



Contrairement aux précédent protocoles qui sont souvent implémentés dans des clients dédiés, les WebSockets sont souvent ajoutées à une application Web (en JavaScript) pour optimiser une partie des échanges.





# IDENTIFIANTS DE RESSOURCES

# URI



Une Uniform Resource Identifier (URI) est une courte chaîne de caractères identifiant une ressource physique ou abstraite sur un réseau (par exemple une ressource Web)

Les URI sont la technologie de base du World Wide Web car tous les hyperliens du Web sont exprimés sous forme d'URI.

Un URI doit permettre d'identifier une ressource de manière permanente, même si la ressource est déplacée ou supprimée.



Les URI ne se limitent pas au Web : par exemple, l'ISBN identifiant les livres de manière unique est une URI.



Exemple : `urn:ietf:rfc:2396` est un URI identifiant la RFC 2396.



Les URI les plus utilisées dans le Web sont les URL et les URN.

# URL



Une Uniform Resource Locator (URL) est une URI qui fournit également les moyens d'agir sur une ressource ou d'obtenir une représentation de la ressource.

Pour ce faire, l'URL décrit le mode d'accès primaire ou « emplacement » réseau de la ressource qu'elle identifie. Les URL ont été créées pour indiquer aux navigateurs Web comment accéder à toutes les ressources d'Internet.

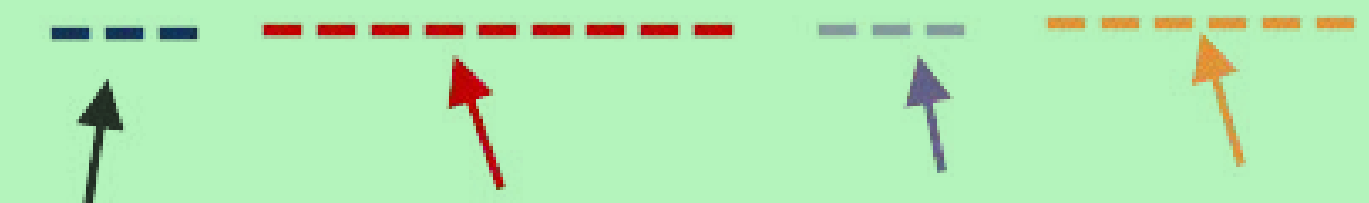
Par exemple, l'URL `http://www.wikipedia.org/` est une URI qui identifie une ressource (page d'accueil Wikipédia) et implique qu'une représentation de cette ressource (une page HTML en caractères encodés) peut être obtenue via le protocole HTTP depuis un réseau hôte appelé `www.wikipedia.org`



Outre les adresses Web, une URL peut aussi décrire :

- un forum Usenet : `news:fr.comp.infosystemes.www.auteurs`
- une adresse mail : `mailto:john.doe@mail.com`
- une ressource FTP :  
`ftp://ftp.mozilla.org/pub/mozilla.org/firefox/releases/`
- un numéro de téléphone: `tel:+33 1 234 567 890`

`http://www.mysite.com:8080/index.htm`



**protocol   domain name   port   resource**

Format d'une URL pour les adresses Web

# URN



Une Uniform Resource Name (URN) est une URI qui identifie une ressource (un document, une image, un enregistrement sonore, etc.) globalement, durant toute son existence, indépendamment de sa localisation ou de son accessibilité par Internet.

Une URN peut être employé pour parler d'une ressource sans que cela préjuge de son emplacement ou de la manière de la référencer.



Une ressource est identifiée par son nom dans un espace de noms sous la forme : urn:NID:NSS où

- urn est une constante indiquant que cette URI est une URN
- NID (Namespace Identifier) est un identificateur d'espace de noms
- NSS (Namespace specific String) est le nom de la ressource dans l'espace de noms.



Les ressources ne sont pas localisées : l'URN urn:isbn:0-395-36341-1 est un URI qui, étant un numéro de l'International Standard Book Number (ISBN), permet de faire référence à un livre, mais ne suggère ni où, ni comment en obtenir une copie réelle.



Les URN sont très peu utilisées aujourd'hui, on leur préfère l'usage d'un moteur de recherche pour trouver la nouvelle URL après le déplacement d'une ressource.

L'arrivée du Web 3.0 (Web sémantique) pourrait créer un besoin d'URNs pour identifier la même ressource entre différentes machines.

# Le Web



Le Web est donc un ensemble d'hyperliens exprimés sous forme d'URIs (généralement des URLs) identifiant des ressources HTML obtenues via le protocole HTTP.

Exemple de codeHTML :

```
<HTML>
<HEAD> <TITLE> Bienvenue sur cette page web </TITLE>
</HEAD>
<BODY>
<TABLE BORDER>
<CAPTION >Sites Web</CAPTION>
<TR>
<TH>Sites</TH>
<TH>Liens</TH>
</TR>
<TR>
<TD>Facebook</TD>
<TD><A HREF="http://www.facebook.com ">Vers le Facebook</A></TD>
</TR>
<TR>
<TD>Youtube</TD>
<TD><A HREF="http://www.youtube.com">Vers le Youtube</A></TD>
</TR>
</TABLE>
<font color="yellow"> Merci pour votre visite </font>
</BODY>
</HTML>
```

# PARTIE II

## Introduction à la notion de cloud

# Introduction à la notion de cloud

*Qu'est-ce que le Cloud ?*

*Pourquoi le Cloud ?*

*Quels sont les différents types de Cloud ?*



# Le cloud



Un Cloud est un ensemble de services informatiques (serveurs, stockage, mise en réseau, logiciels) disponibles via Internet à partir d'un fournisseur.

Utiliser le Cloud, c'est donc externaliser des services et/ou des ressources en les déléguant à un fournisseur externe.

La localisation et l'implémentation de ces services est souvent masquée aux utilisateurs.



Le terme Cloud vient de la représentation courante d'Internet dans les infographies sous forme de nuage.

Cela décrit bien le but de ce procédé : abstraire et masquer au client la gestion et l'administration des services proposés pour que celui-ci puisse se concentrer uniquement sur l'utilisation des services qui lui sont dédiés..



Il existe plusieurs architectures de Cloud et plusieurs types de services de Cloud pouvant être regroupés en plusieurs catégories que nous allons voir.

# Histoire du Cloud

Dans les années 50, les ressources matérielles étaient coûteuses et les postes de travail très limités en ressources.

On avait l'habitude d'utiliser des serveurs partagés pour réaliser tous les traitements et stockages de données afin de se limiter à l'usage d'un terminal léger sur son poste de travail (terminal SSH, connexion VNC ou RDP, ...).

L'idée du Cloud était née : déléguer des services à une machine distante.

La réduction considérable des coûts de machines personnelles capable d'exécuter l'ensemble des services d'un utilisateur a fait abandonner cette pratique au profit de l'utilisation de services locaux.

Dans les années 2000, certaines grandes entreprises IT (Amazon, Google) ont de grands pics de charge à certaines périodes de l'année (Noël) et tentent de rentabiliser leurs coûts d'infrastructure lorsque ces ressources ne sont pas utilisées.

L'arrivée de réseaux permettant le transfert de grandes quantités de données à haut débit va pousser ces entreprises à déléguer une partie de leurs ressources quand elles ne les utilisent pas contre rémunération : c'est le début du cloud moderne.

# Histoire du Cloud

Aujourd'hui, le Cloud computing est en pleine expansion :

- principalement dans une optique de rationalisation des coûts IT (en entreprise).
- pour pallier au manque de ressources de certains terminaux (smartphone, objets connectés)
- pour simplifier l'architecture des applications et réduire les coûts de développement en mutualisant des services (login, ...) et des données (accès unique aux données stockées en ligne)

D'après le cabinet Forrester (Octobre 2020) :

« La migration massive vers le cloud, qui s'opérait déjà à un rythme soutenu avant la pandémie, atteindra un pic en 2021 »

« L'an prochain, 60% des (grandes) entreprises dans le monde utiliseront des conteneurs sur les plateformes de cloud public »

<https://go.forrester.com/blogs/predictions-2021-edge-computing-hits-an-inflection-point/>

# Les types de Cloud : privé



Le cloud privé (ou cloud dédié) est la forme la plus courante de Cloud.

Dans cette architecture, l'infrastructure est entièrement dédiée à un panel restreint d'utilisateurs (souvent liés à un unique client).



Même si l'infrastructure est dédiée à un client, les ressources physiques ne sont pas forcément localisées sur son réseau. Un cloud privé peut être hébergé et/ou rattaché au réseau interne d'un client, ou disponible sur un réseau publique avec authentification.

## Les avantages :

- sécurité : l'infrastructure étant dédiée, pas de partage de services ou de données avec d'autres clients. Pour cette raison, la majorité des entreprises utilisent un cloud privé
- il est adapté sur mesure aux besoins d'infrastructure du client
- les coûts augmentent peu avec l'accroissement des données

## Les limites :

- il est (très) coûteux à mettre en place, surtout pour de petits volumes de données
- il est difficile à faire évoluer et peu réactif à la montée en charge

# Les types de Cloud : public



Le cloud public est une architecture où l'infrastructure du ou des services est unique et est partagée entre tous les utilisateurs.



Par exemple, Google Drive est un service de cloud public : les utilisateurs utilisent les mêmes serveurs pour accéder au service et y stocker des données.

## Les avantages :

- il peut être (très) abordable : pour cette raison, il est principalement utilisé par des particuliers ou de petites entreprises
- il est très réactif à la montée en charge et très évolutif

## Les limites :

- sécurité : c'est l'implémentation des services qui garantit la segmentation des données : en cas de faille, diffusion des données aux autres utilisateurs
- les besoins et les usages sont standardisés : pas ou peu de customisation
- coûts élevés si forte utilisation de données

# Les types de Cloud : hybride



Le cloud hybride est une architecture de cloud privé extensible par un cloud public. Son but est de tirer le meilleur parti des deux mondes :

- on privilégie en principe l'utilisation de l'infrastructure dédiée (cloud privé)
- en cas de forte charge, les services et/ou données sont transférés sur la partie publique du cloud pour répondre au besoin de croissance de l'infrastructure



Le cloud hybride est puissant mais son implémentation est complexe car elle nécessite une capacité à interagir et échanger des données entre la partie privée et la partie publique du cloud, sans compromis de sécurité ou de performances



Le cloud hybride peut être une solution intéressante d'infrastructure hybride si les données et/ou services sont bien segmentés. On pourra par exemple :

- utiliser la partie dédiée pour les applications en production et la gestion des données des clients
- utiliser la partie publique pour les tests et toute autre utilisation de données non critiques



# Les services de Cloud

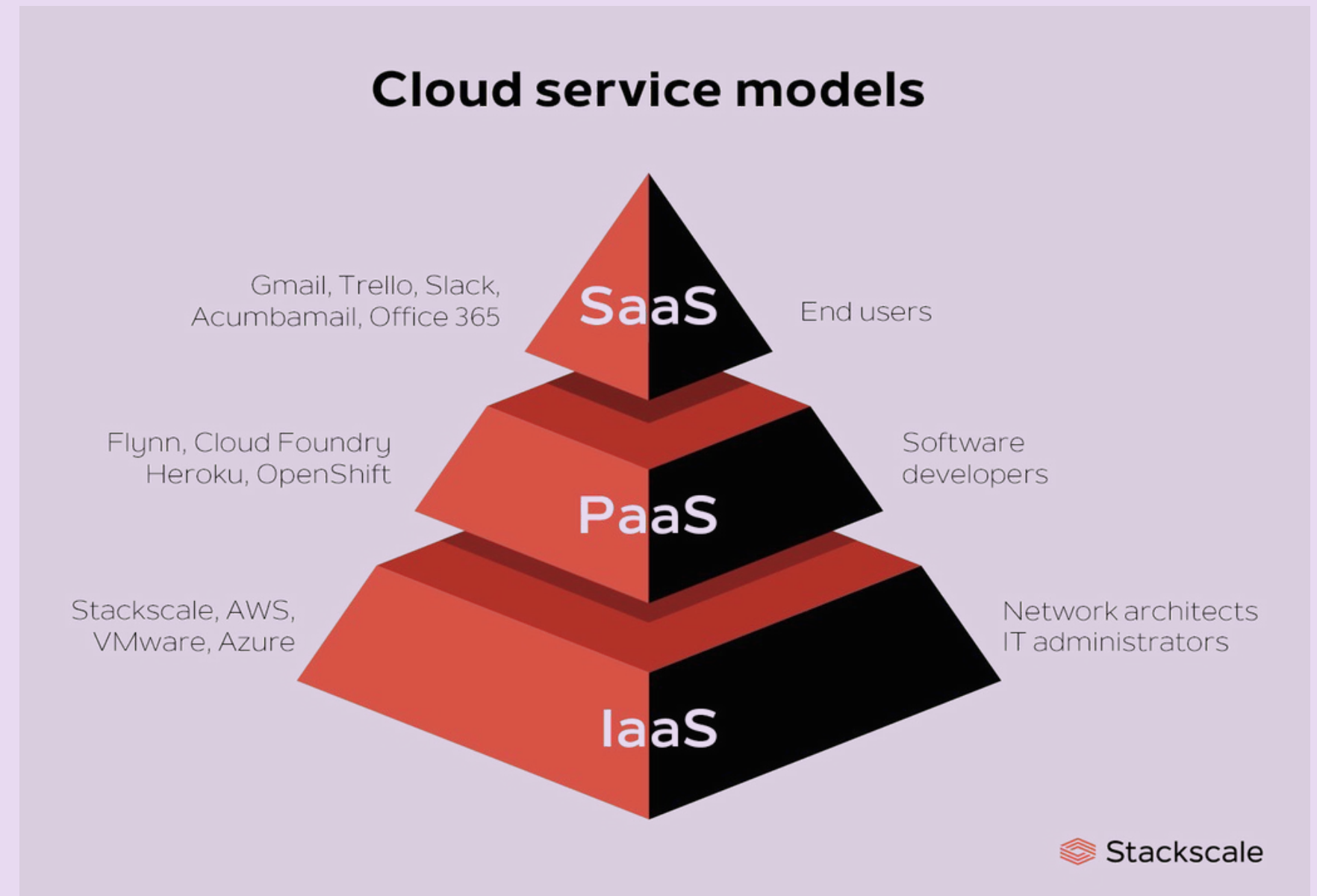
Les environnements de Cloud sont découpés en plusieurs modèles d'architecture en fonction de l'infrastructure qu'ils virtualisent et des services qu'ils proposent au client.

On regroupe la plupart des environnements en 3 catégories :

- Software-as-a-Service
- Platform-as-a-Service
- Infrastructure-as-a-Service

Il existe de nombreux autres types de Cloud plus anecdotiques :

- Data as a service
- Desktop as a service (DaaS)
- Network as a service (NaaS)
- Storage as a service (STaaS)
- Communication as a service (CaaS)
- Workplace as a service (WaaS)
- Mobile backend as a service (MBaaS), ...



Pour un déploiement "traditionnel" (hors cloud), on parle de déploiement "on-premise".

# SaaS

## Le levier de transformation numérique



Le modèle Software-as-a-Service (SaaS) est le modèle "clé en main" : le client a directement et uniquement accès à l'interface utilisateur du service applicatif.

L'intégralité de l'infrastructure, du déploiement / administration des serveurs à la maintenance applicative, sont sous la responsabilité du fournisseur de Cloud.

Le client n'a donc aucun contrôle sur l'administration de l'application, y compris la version utilisée.



Exemples de cloud SaaS : Gmail, les différents hébergements de données dans le cloud (Google Drive, Microsoft One Drive), Facebook, ...



Le SaaS est un bon moyen de réduire les coûts de maintenance pour un développeur logiciel : en fournissant une unique version déployée de son produit, il gère facilement l'administration et la mise à jour de celui-ci. Il s'abstrait ainsi de la livraison du produit au client et garde le contrôle total sur les données.

Par exemple, Facebook aurait du mal à synchroniser les données utilisateurs si elles étaient réparties sur tous les ordinateurs personnels et smartphones.



# PaaS

## Pour l'innovation



À la différence du SaaS, le « Platform as a Service » consiste à déployer sur l'infrastructure Cloud ses propres applications, dans la mesure où le fournisseur supporte le langage de programmation.

L'entreprise ne contrôle pas l'infrastructure Cloud sous-jacente, mais elle peut configurer l'environnement d'hébergement applicatif et ainsi conserver une parfaite maîtrise de ses applications déployées.



Exemples de cloud PaaS : Microsoft Azure,  
Google App Engine, ...

# IaaS

## La puissance à moindre coût



Avec "l'Infrastructure as a Service", les entreprises peuvent disposer, à la demande, de ressources informatiques essentielles : capacité de traitement, capacité de stockage de données, composants réseau, intergiciels (middleware). Elles peuvent notamment démarrer, arrêter ou configurer une machine virtuelle selon leurs désirs. L'entreprise contrôle les systèmes d'exploitation, les bases de données et les applications déployées.

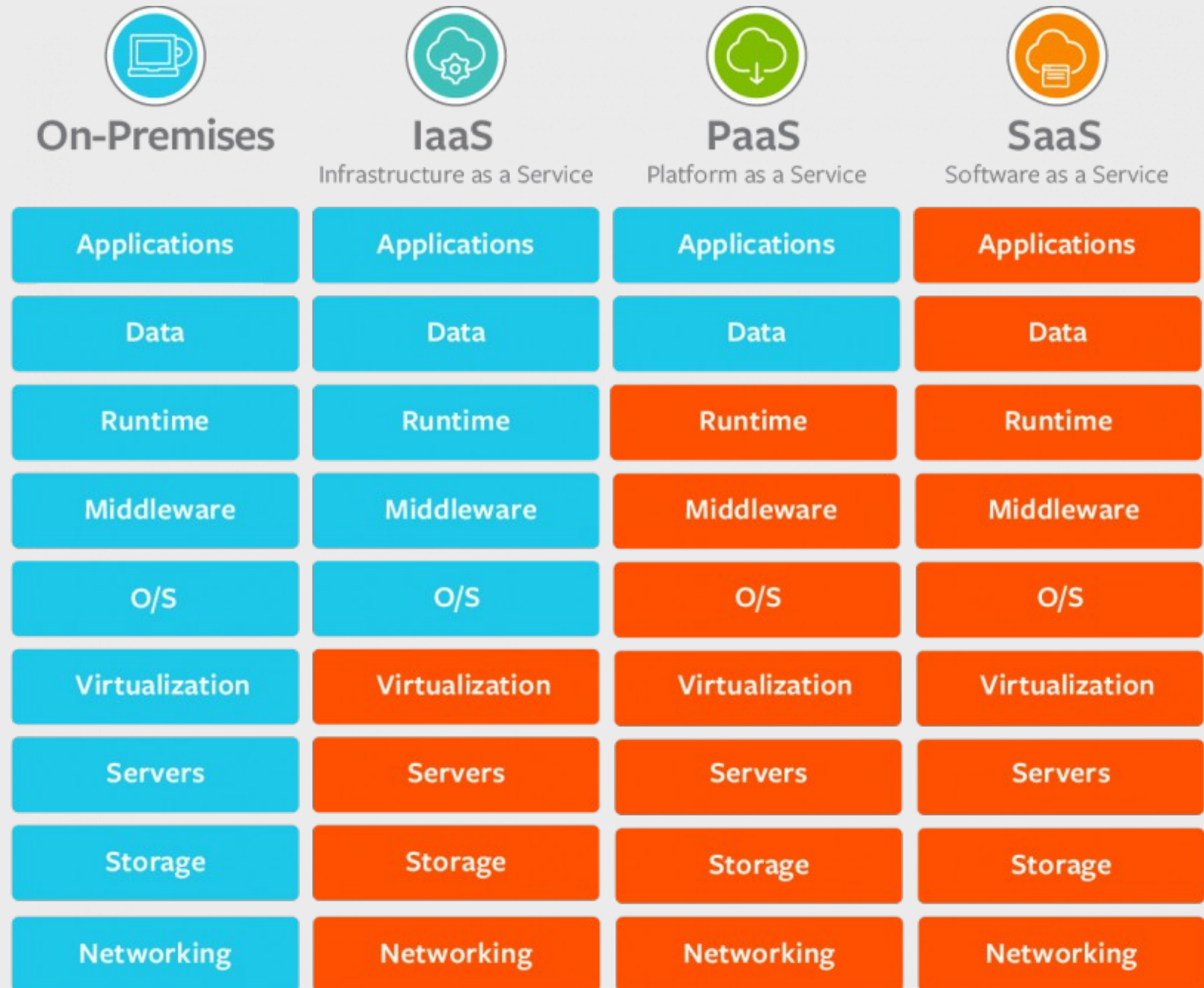


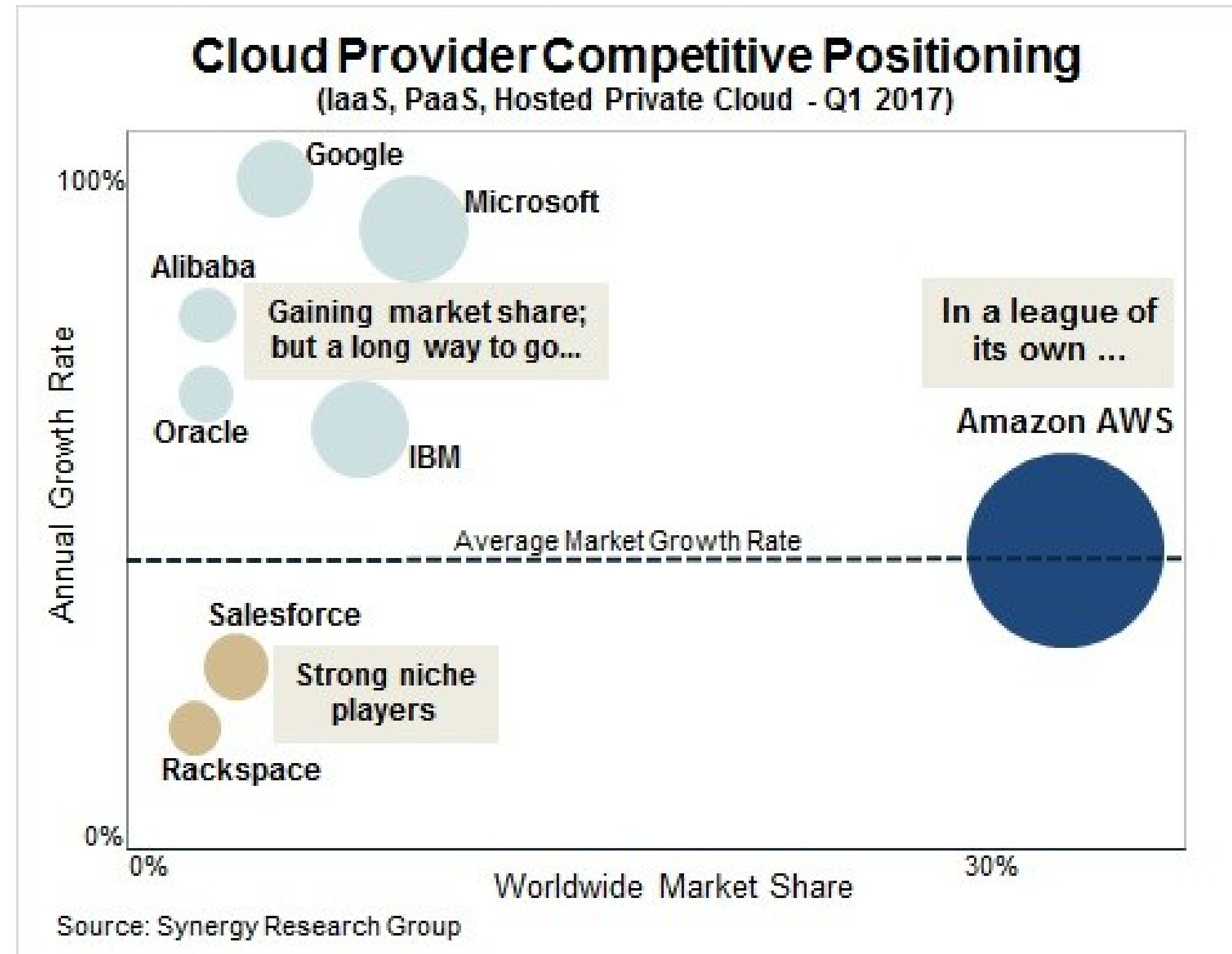
La majorité des éditeurs SaaS s'appuient sur les infrastructures IaaS des grands fournisseurs américains pour déployer leur propre Cloud



Amazon AWS reste (de loin) le leader en 2020 des fournisseurs IaaS avec plus de la moitié des parts de marché

# Résumé modèles de Cloud





Les principaux fournisseurs de Cloud (2017)

# Amazon Web Services

Dans les années 2000, la startup Amazon connaît une croissance exponentielle et son infrastructure a des difficultés à suivre cette évolution.

De plus, la multiplication de nouveaux projets rend difficile leur déploiement : les délais sont souvent dépassés alors même que la base de données n'est pas encore déployée !

Amazon a alors l'idée de développer un outil interne (AWS) pour rationaliser son infrastructure et permettre le partage et la réutilisation de briques d'infrastructure entre les différents projets (base de données, composants de stockage, ...)

En 2003, un nouveau projet visant à rentabiliser AWS en vendant un "système d'exploitation pour Internet" est lancé. C'est une plateforme sur laquelle les clients peuvent déployer leur application, proche d'un PaaS actuel.

Quelques années plus tard, Amazon élargira son offre Cloud en publiant intégralement sa plateforme d'IaaS. Aujourd'hui, Amazon est de loin le leader du cloud, notamment en IaaS (grâce à son arrivée précoce sur le marché notamment) et son offre est très diversifiée.

# Amazon EC2

Amazon Web Services (AWS) est une offre Cloud complète disposant de (très) nombreux services. On peut cependant retenir quelques services principaux qui sont les fondations de cette offre :

Amazon EC2 (Elastic Compute Cloud) est le premier service Cloud lancé par Amazon (et le premier "véritable" service Cloud au monde). EC2 fournit un service de Cloud IaaS en permettant la création à la demande de machines virtuelles, depuis des templates préconfigurés.

L'utilisateur dispose alors d'une machine virtuelle d'un système d'exploitation complet (souvent Linux, voir Windows Server) et accessible en ligne de commandes.

EC2 est le principal service d'AWS et celui qu'il convient de maîtriser le mieux avant de pouvoir utiliser d'autres services Cloud d'Amazon.

Le coût des machines virtuelles dépend très principalement de la puissance qui leur est allouée (mémoire, CPU, ...)



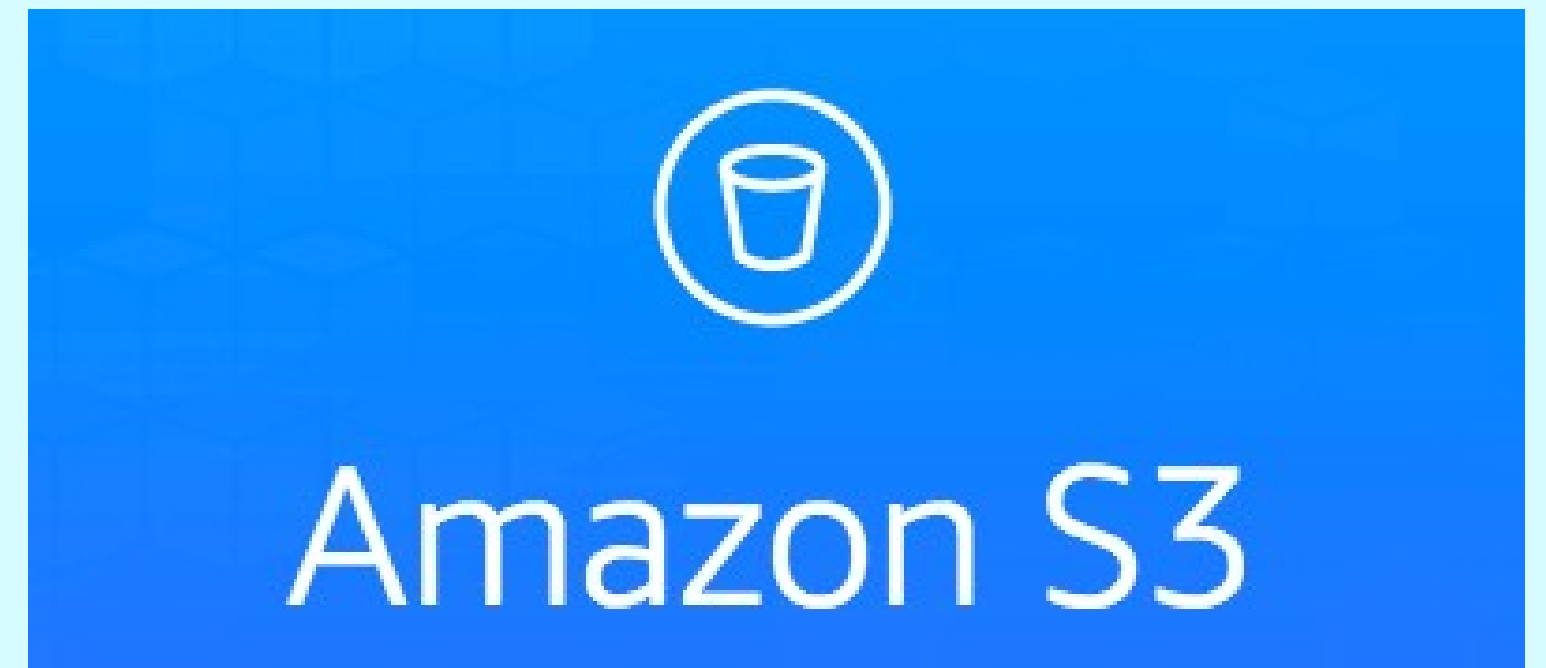


# Amazon S3

Amazon S3 (Simple Storage Service) est le deuxième service à la base d'AWS. C'est un service de stockage et de distribution de fichiers, de sa forme la plus simple à la plus évoluée.

S3 gère de simples stockages de données, les disques des machines virtuelles EC2 (incluant les backups, ...), des fichiers haute disponibilité...

Le coût du stockage dépend principalement de la taille des disques et des exigences qualité (SSD, nombre de répliqués, ...)



# Amazon RDS

Amazon RDS (Relational Database Service) est un service très utilisé dans le développement d'applications Web, qui nécessitent souvent une base de données. RDS fournit ce besoin en gérant toute la partie opérationnelle (déploiement et configuration de la base de données)

RDS ne fournit aucune possibilité opérationnelle sur les bases de données (mise à jour, ...) : c'est à la fois sa puissance (les besoins opérationnels sont abstraits au développeur) et sa faiblesse (aucun contrôle sur la version de la base de données utilisée, ...)





# PARTIE III

## INFRASTRUCTURE E- BUSINESS : CLOUD ET WEB

# INFRASTRUCTURE E-BUSINESS : CLOUD ET WEB

*Quel impact du Web dans l'architecture des applications ?  
Comment utiliser des services du Cloud dans une application moderne ?*

# Architectures des clients

On classe généralement les clients selon l'importance des services qu'ils proposent par rapport au serveur.

Un client peut en effet choisir de déléguer plus ou moins de données ou de services à un ou plusieurs serveurs : par exemple, on peut stocker les préférences de l'utilisateur directement sur sa machine (par exemple dans des cookies du navigateur) ou en utilisant la base de données du serveur.

Dans le cas d'un client proposant la majorité des services de l'application, on parle de client lourd.

Dans le cas d'un client délégant la majorité des services de l'application au serveur, on parle de client léger.



Après l'effervescence des applications lourdes (années 80 - 2000), la tendance est aujourd'hui de nouveau aux clients (très) légers. Cela s'explique par :

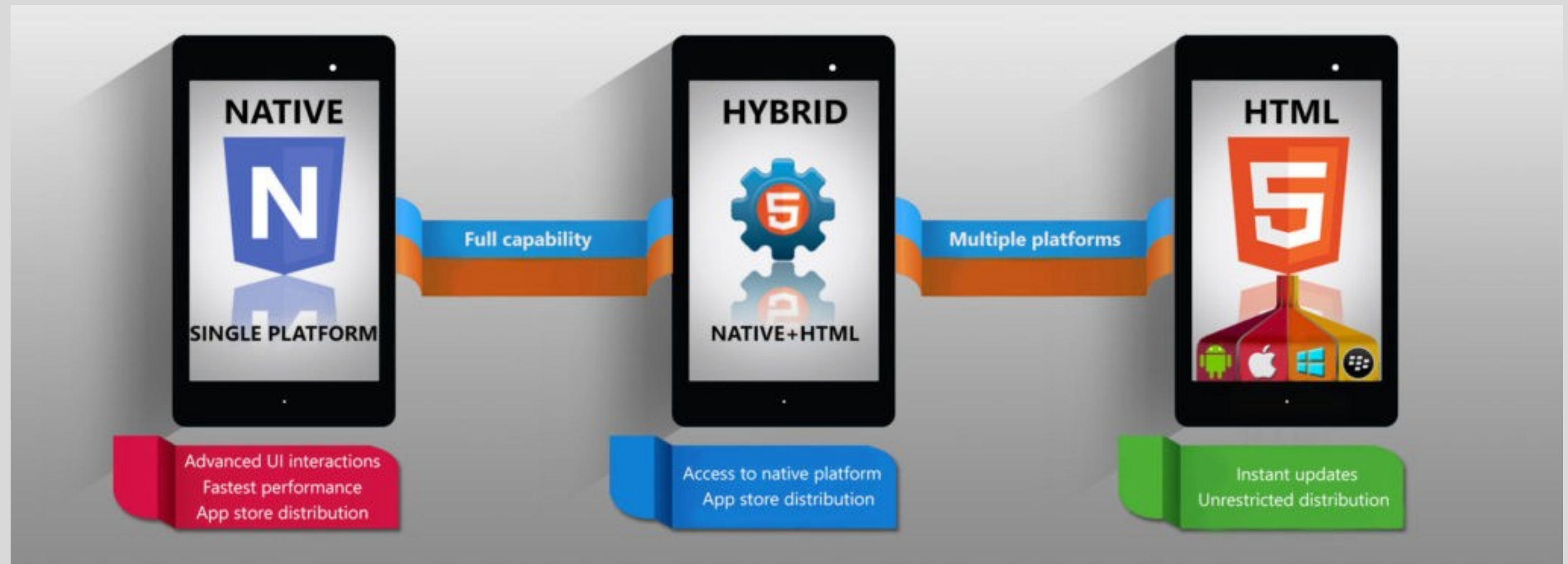
- la multiplication des clients et des points d'accès (ordinateur, smartphone, box, ...)
- la capacité des infrastructures à supporter de forts échanges de données



# Architecture client des terminaux mobiles

Le choix d'un client léger ou d'un client lourd a notamment un fort impact sur l'architecture web des applications visant des terminaux mobiles (smartphone, tablette, ...)

On distingue 3 familles d'applications mobiles liés à ce choix :



# Architecture client des terminaux mobiles

## La WebApp



La WebApp est développée à partir de technologies web en imposant une forte « responsivité » du site. Elle peut alors être utilisée directement via le navigateur du mobile.

Pour des raisons liées à la visibilité de l'application, un composant navigateur peut être intégré à une application mobile (très légère) pour rendre l'application déployable dans les stores.

Une autre solution est d'utiliser la technologie Progressive Web Apps (PWA) pour "installer" l'application web sur la machine du client (il s'agit en réalité d'une fonctionnalité du navigateur simulant une application desktop).

Ces applications sont également directement disponibles sur tous les clients "classiques" (ordinateur personnel, ...) mais elles ne fonctionnent pas (ou mal pour la PWA) sans le serveur Web.



Dans une démarche itérative, une WebApp est un point de départ minimaliste et qui peut être réutilisé. Cette solution est donc économique et peut être déployée rapidement. Cependant, l'utilisation des capteurs locaux est limitée à la géolocalisation.

# Architecture client des terminaux mobiles

## L'application hybride



L'application hybride est développée avec des technologies utilisant les capacités JavaScript des mobiles. On retrouve des solutions comme Sencha, PhoneGap, Ionic, Corona. Ces solutions permettent de générer des applications déployables sur les stores et de commencer à utiliser les ressources des terminaux mobiles.



Les applications hybrides permettent d'aller un pas plus loin pour prendre en compte certains capteurs (appareil photo, calendrier, micro ...). Développer une application mobile hybride a alors l'avantage d'être rapide avec une maintenance simple qui nécessitera un seul développement.

# Architecture client des terminaux mobiles

## L'application native



Le développement d'une application native consiste à utiliser un environnement de développement propre à chaque terminal (Swift/Objective-C pour iOS, Java/Kotlin pour Android).

Cela implique un double développement mais avec au final une expérience utilisateur très poussée.



En alternative à ce double développement, certaines solutions (Xamarin, ...) proposent un environnement de développement qui sera converti en code natif pour chaque système d'exploitation.



Une application native permettra d'aller plus loin dans les capacités de l'application mobile, l'expérience utilisateur sera maximisée et la connexion Internet ne sera pas requise pour son utilisation.

Toutefois, son développement représente une maintenance plus complexe, les mises à jour doivent se faire sur chaque application et sont contraintes par l'évolution des stores.



# Web 2.0 : Les Web Services

A web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards.

—W3C, Web Services Glossary



Un Web Service est donc un service web conçu pour avoir une interface existant dans un format traitable par des machines.



Il est généralement décrit dans une grammaire de type Web Service Description Language (WSDL) et utilise le protocole HTTP pour la communication.

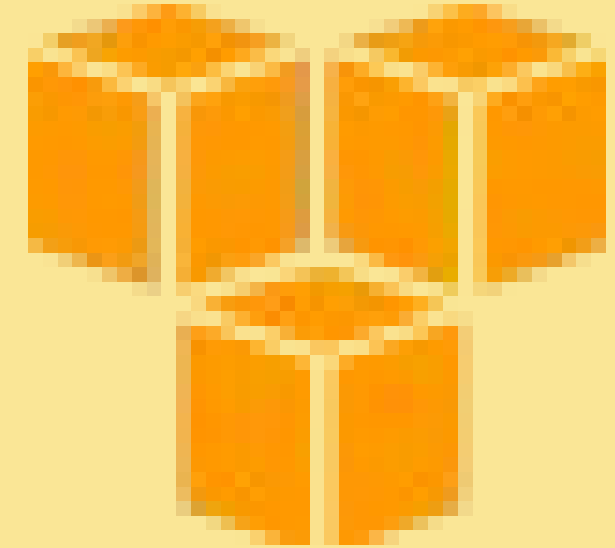
Même si la norme préconise d'échanger les données en utilisant XML, en pratique le format JSON est de plus en plus utilisé. De même, des supports de communication comme AJAX, REST / RESTful et XML-RPC sont souvent utilisés à la place de SOAP.



# Web 2.0 : Les Web Services



En pratique, un Web service fournit en général une interface Web orientée objet vers un serveur de base de données. Ce service est utilisé par exemple par un autre serveur Web ou une application mobile dans son application "end-user".



Il est courant pour un fournisseur de données proposant un service sur un site web (service météo, ...) de fournir également ces données en XML ou JSON à travers un Web service.



Quelle différence avec une API ?

La seule différence est qu'un service Web facilite l'interaction entre deux machines sur un réseau alors qu'une API sert d'interface entre deux applications différentes afin qu'elles puissent communiquer entre elles

# Applications modernes



Une "application moderne" est une application qui supporte des clients multiples : site web, application mobile, et/ou autre application cliente qui utilise les données et services de l'application à travers son API.

Ces applications fournissent une API HTTP(s) commune à tous leurs clients (et non une API optimisée pour chaque client) qui expose tous les services et fonctionnalités accessibles au client par une interface utilisateur ou une CLI. Les données sont disponibles dans un format générique et consommable (JSON, ...) et l'API représente les objets et services de manière claire et organisée (RESTful, GraphQL, ...)

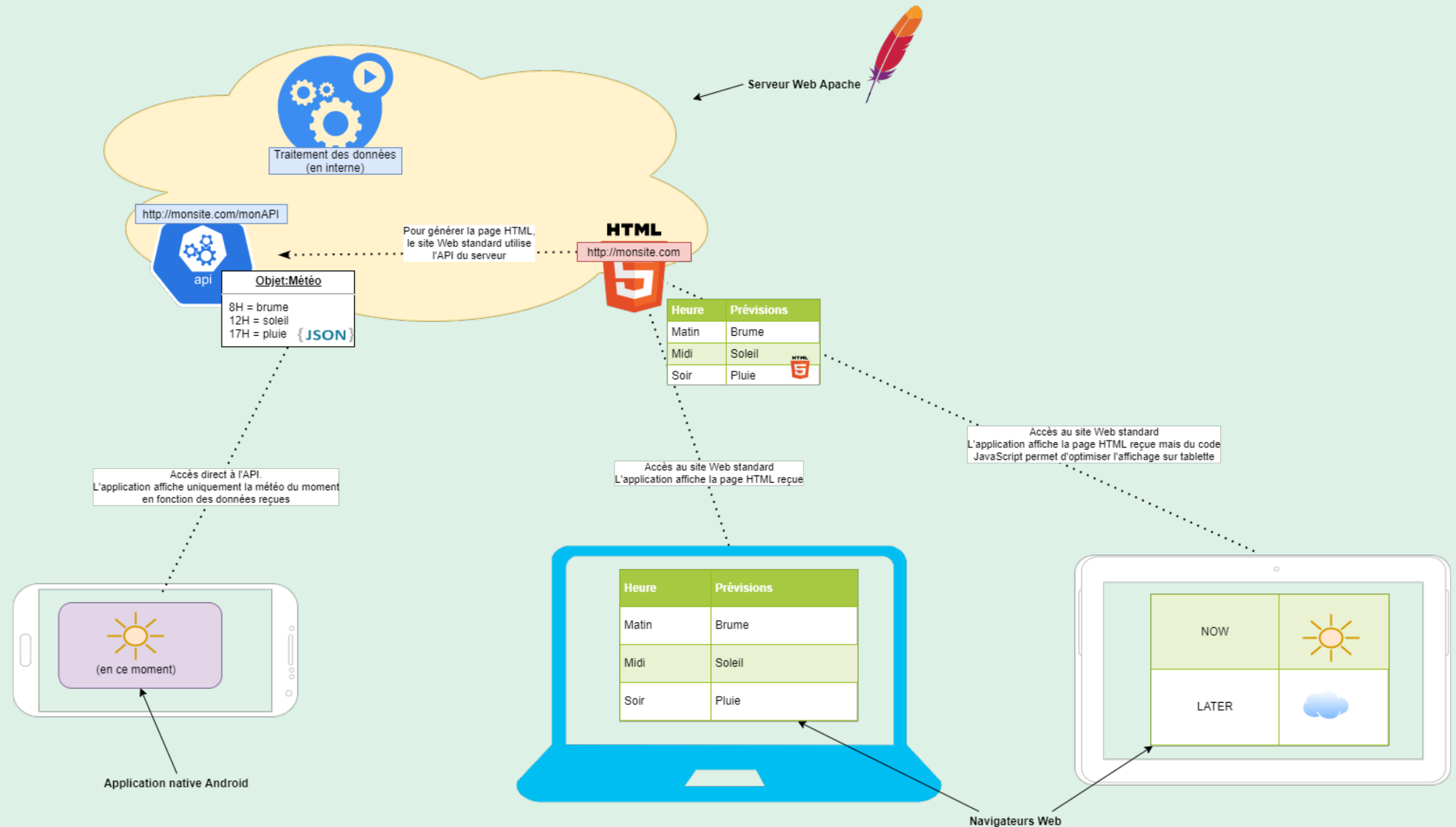


Les application modernes ne font pas d'hypothèse sur le nombre d'utilisateurs qui utilisera les données et services de l'application, elles sont donc conçues pour être hautement réactives aux montées en charge.



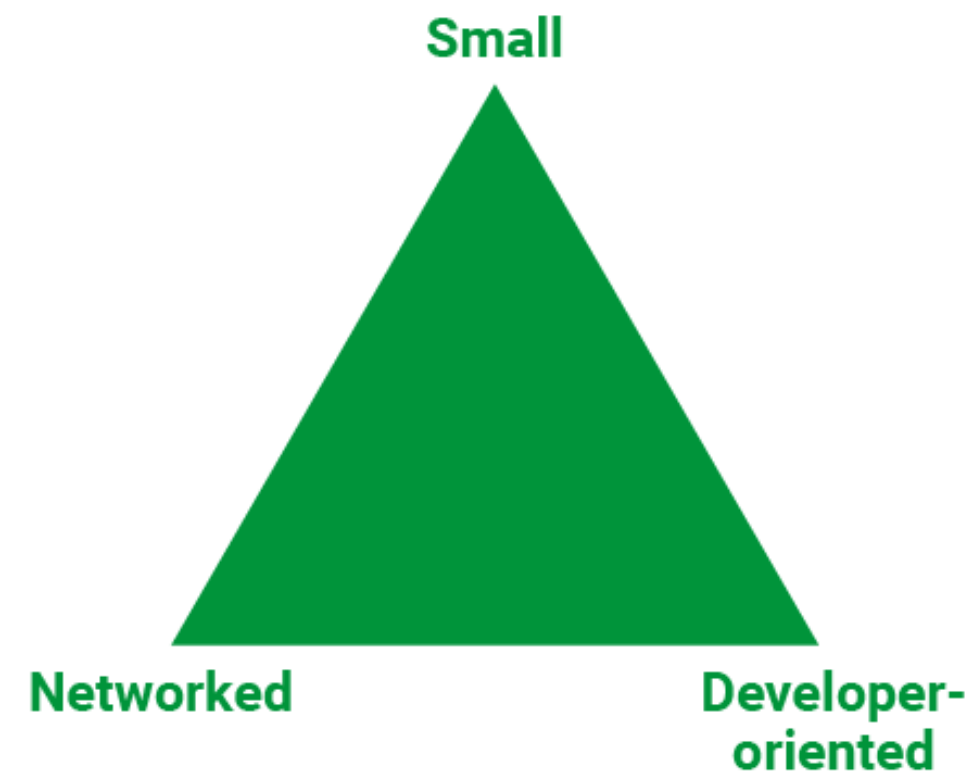
L'utilisation d'une API (unique) permet de limiter l'intégration avec les interfaces utilisateur et de pouvoir faire évoluer ceux-ci rapidement.

# Applications modernes



# Applications modernes

## Key Principles of Modern Application Development



Pour plus d'informations : <https://www.nginx.com/blog/principles-of-modern-application-development>

# PARTIE IV : APPLICATION WEB ET CLOUD

# APPLICATION WEB ET CLOUD

*Quels sont les composants d'une application Cloud ?  
Comment développer une application dans le Cloud ?*

# APPLICATION WEB ET CLOUD

4.1

## LES COMPOSANTS

# Fonctions du navigateur

Un navigateur web doit être capable, au minimum, d'afficher le texte d'une page web.

Les navigateurs couramment utilisés sont cependant capables de beaucoup plus : utiliser une typographie élaborée, décoder et afficher des images depuis de nombreux formats, jouer de la musique et des animations, interagir avec les actions de l'utilisateur (clavier, souris, microphone, caméra, ...).

Pour afficher une page web, le navigateur génère une représentation du code HTML : le Domain Object Model.

Ce modèle est affiché à l'écran grâce à l'utilisation de feuilles de style CSS qui définissent comment représenter les différents éléments de l'interface DOM de manière unifiée.

Un script peut être adjoint au document pour réaliser un ensemble de tâches simples (vérification des champs d'un formulaire, ...) ou complexes (générer des éléments du DOM, ...). Ces scripts sont très majoritairement codés en JavaScript, les autres langages du navigateur étant en cours de disparition : Flash, applet Java, ...



# Fonctions du navigateur



La plupart des frameworks frontend récents utilisent des documents HTML minimalistes permettant surtout de charger la librairie du framework et les feuilles de style CSS. L'intégralité du document est ensuite générée dynamiquement par accès au DOM : cela permet beaucoup plus de modularité.



Les navigateurs récents génèrent en fait des modèles bien plus complexes qu'un simple DOM : Shadow DOM (souvent disponibles), Web Components (en cours de normalisation), ... Ces nouvelles fonctionnalités sont utilisées (ou simulées par des polyfill) par les frameworks modernes (Angular, React, ...), permettant une réutilisation poussée du code et une forte optimisation des données à télécharger.

Pour plus d'informations : <https://speakerdeck.com/notwaldorf/how-to-train-your-dragon-web-standard>



Au-delà de la pure navigation Web, un navigateur offre souvent également tout un panel de fonctionnalités liées à l'Internet : client FTP, décodeur de PDF, gestion des impressions, enregistrement des mots de passe, transmission sécurisée en HTTPS, ...

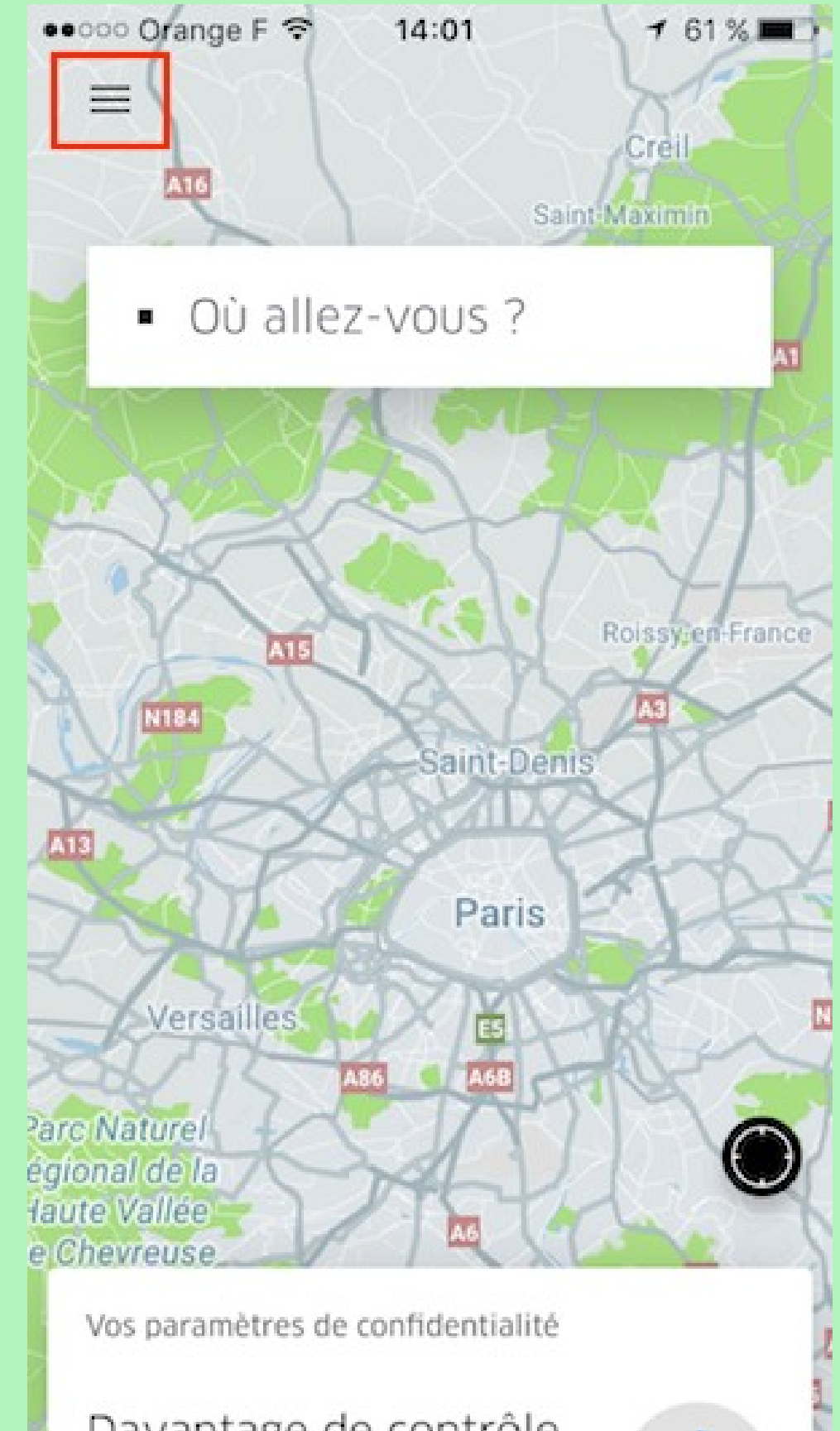
# Navigation mobile

La navigation mobile possède quelques caractéristiques propres :

Choisir le bon menu :

L'espace sur un écran de smartphone ou tablette est réduit, on choisira un menu discret mais visible.

Le menu "hamburger" (comme dans l'application Uber) est le plus populaire.

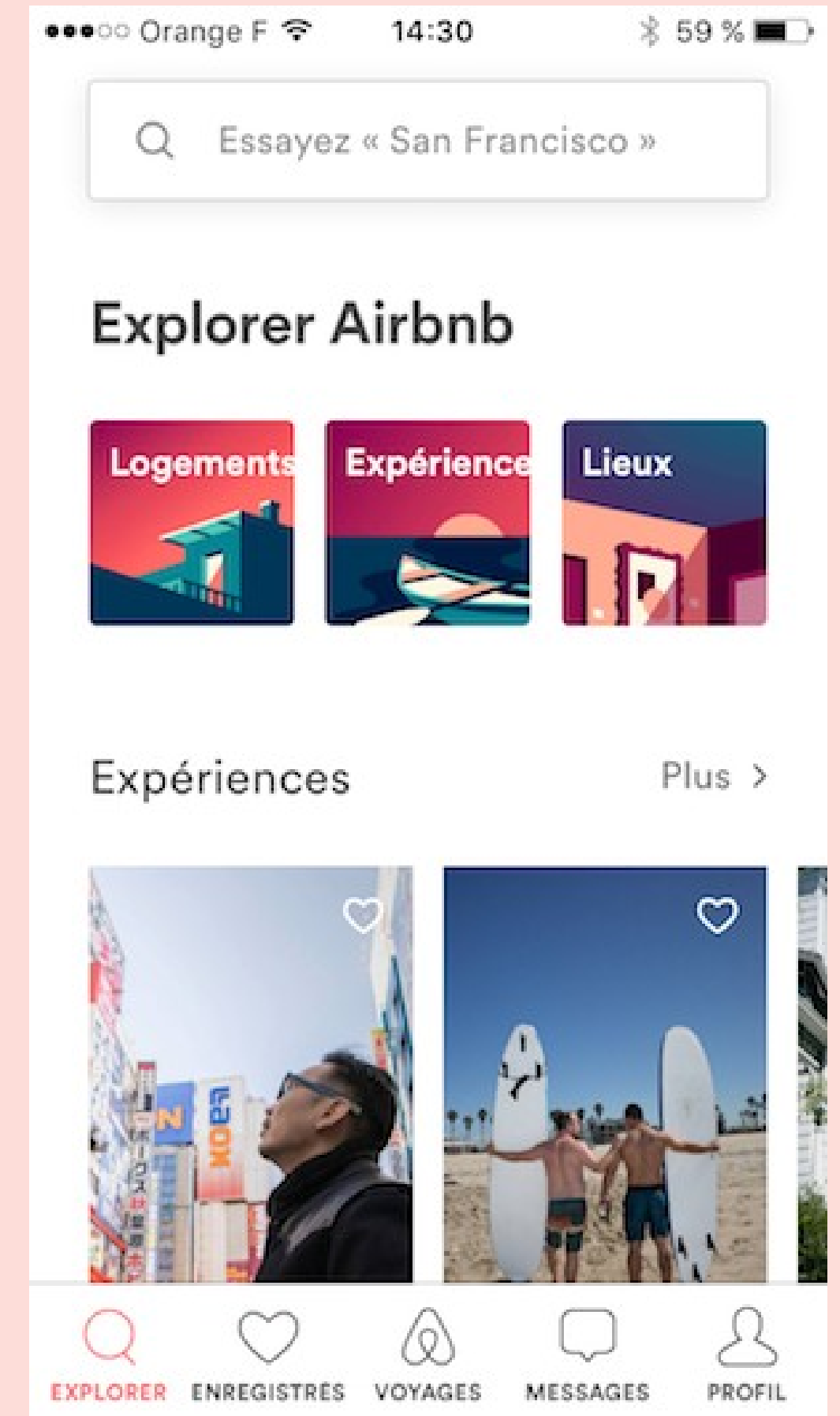


# Navigation mobile

La navigation mobile possède quelques caractéristiques propres :

Bien mettre en évidence les onglets :

- Une simple icône ne suffit souvent pas pour décrire une fonctionnalité
- Mettre en évidence l'onglet courant (couleur différente, ...)
- Hiérarchiser les onglets (gauche à droite, ou inversement suivant la langue)



# Navigation mobile


La navigation mobile possède quelques caractéristiques propres :

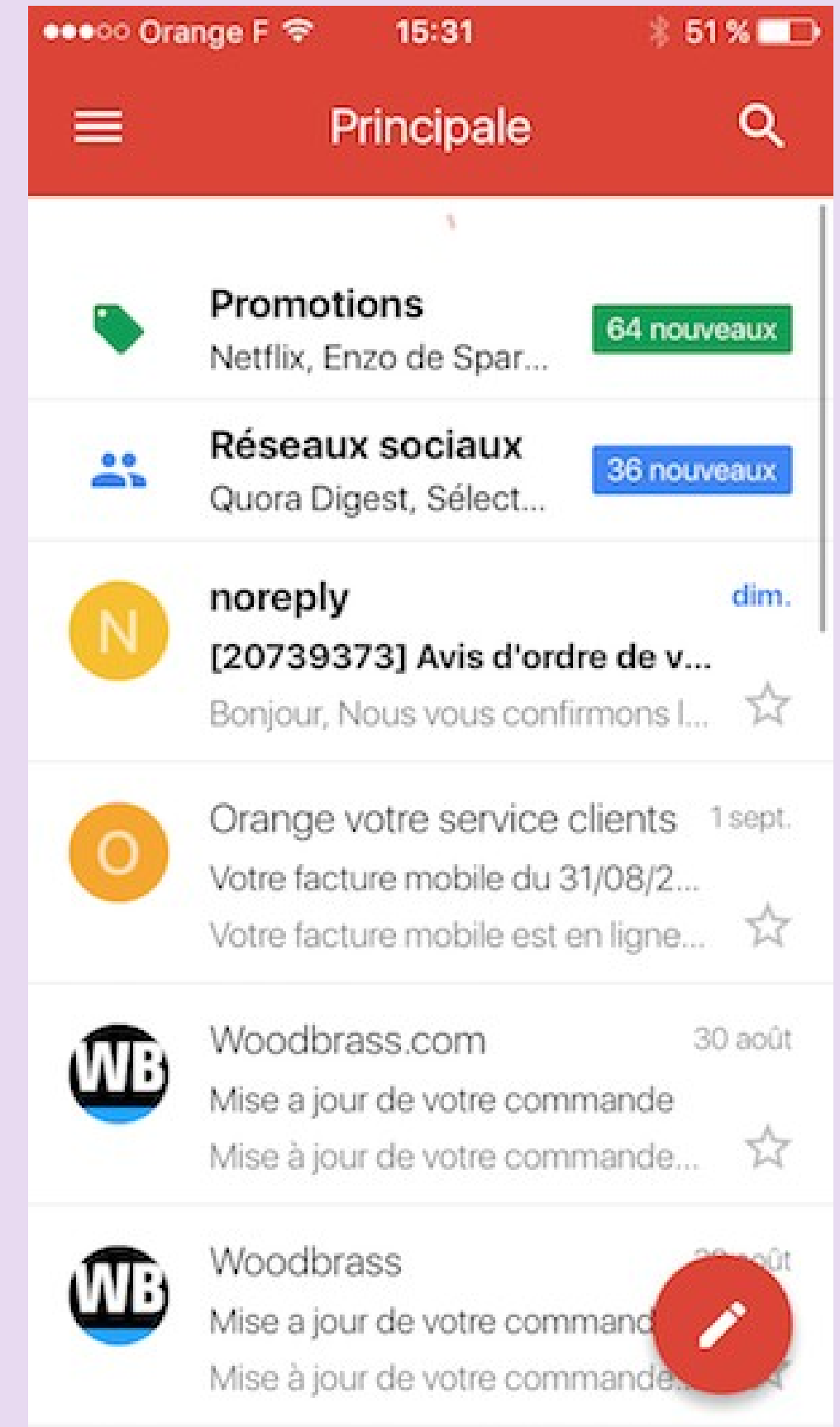
Bien utiliser les boutons d'action flottants :

Ce sont des boutons (souvent ronds) qui semblent flotter au-dessus de l'interface d'une application mobile.

C'est un moyen efficace de guider l'utilisateur vers l'action principale, en économisant de l'espace.

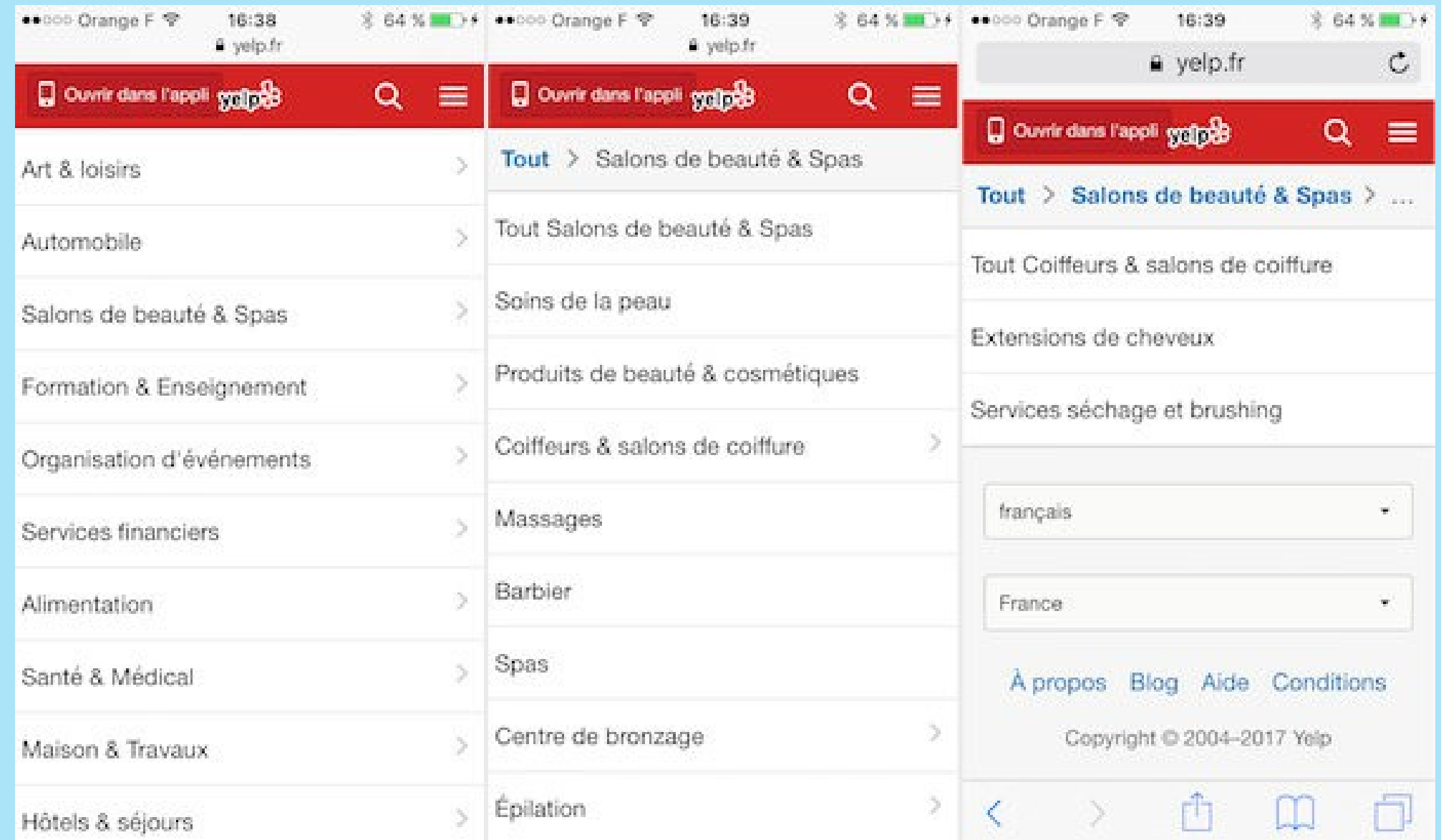
Il peut cependant masquer du contenu, distraire l'utilisateur ou être difficile à comprendre.

 Pour faciliter l'expérience utilisateur, on associe toujours un bouton flottant à une action positive : créer, rechercher, ...



# Navigation mobile

Penser à la navigation en plein écran :  
Ce design se distingue par sa simplicité et son efficacité (sauf si l'utilisateur effectue des allers et retours).



Il peut être judicieux de l'utiliser en complément d'un menu hamburger pour y placer les informations secondaires

# Navigation mobile

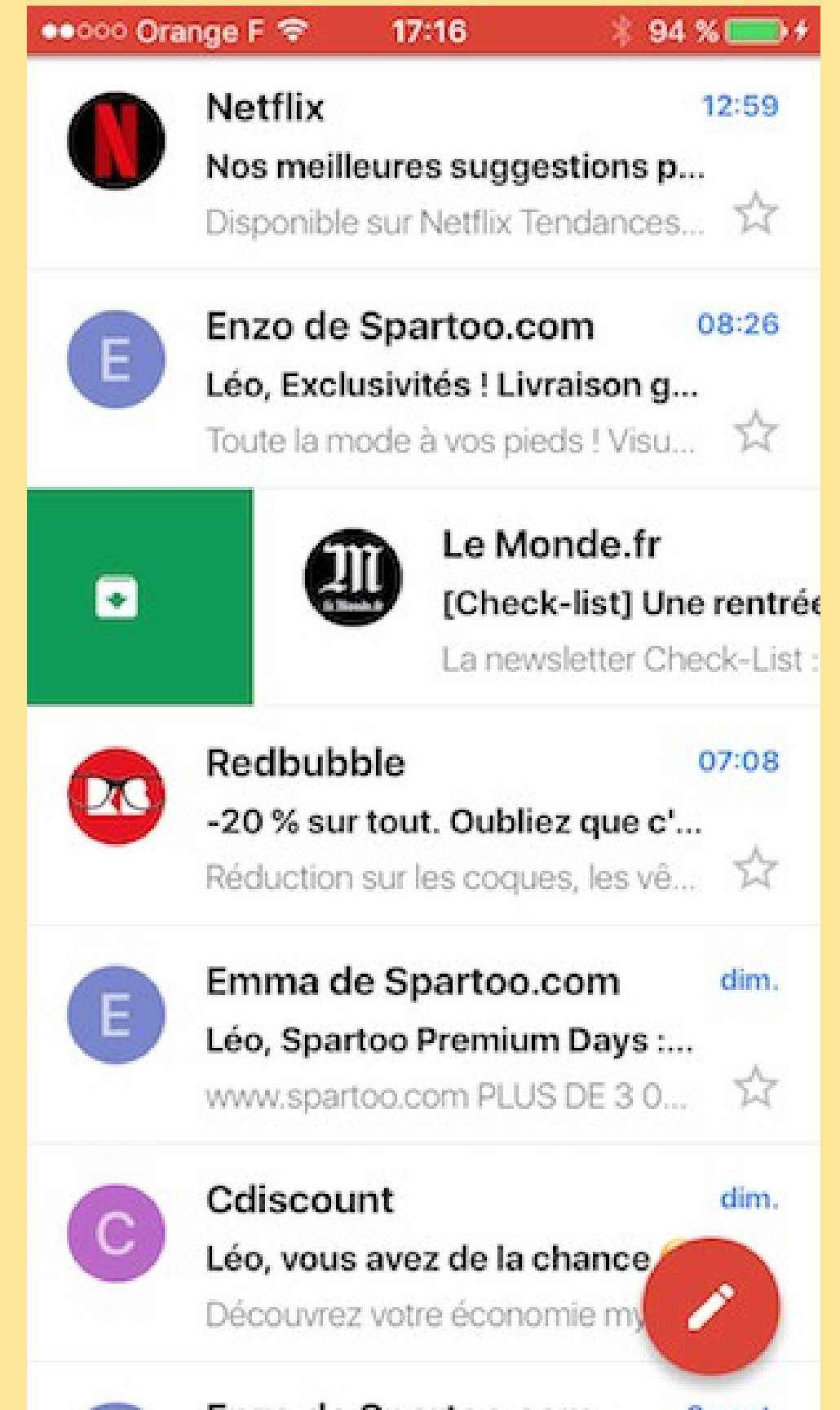
La navigation mobile possède quelques caractéristiques propres :

Exploiter les possibilités du tactile :

Si appuyer sur des boutons et faire défiler des écrans sont des interactions classiques, d'autres gestes basiques comme le "swipe" ou le zoom à 2 doigts sont souvent ignorés.

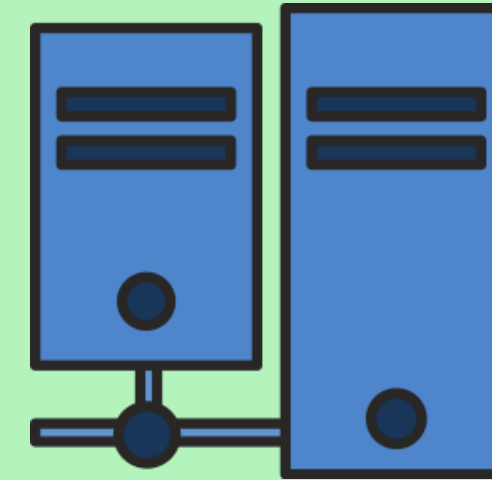
💡 Les interactions tactiles peuvent ne pas être intuitives pour l'ensemble des utilisateurs : un exemple par une animation peut grandement faciliter son apprentissage.

💡 Les interactions tactiles sont rarement considérées comme une interface utilisateur fiable : il est courant de prévoir une alternative à l'action sans gestuelle.





# Serveur web



La fonction principale d'un serveur Web est simple : diffuser des contenus Web sur Internet (ou plus rarement sur un Intranet).

Le terme serveur web peut désigner suivant le contexte :

- le logiciel qui permet la diffusion du contenu
- la machine qui héberge le programme et le contenu (on utilisera dans ce cours le terme machine hôte pour les différencier)

Lorsque l'utilisateur visite un site Web, il renseigne l'adresse Internet correspondante dans son navigateur.

Autrement dit, le navigateur (client) envoie une requête au serveur Web et ce dernier lui envoie une réponse sous forme de page HTML.

Un tel document HTML peut être enregistré sur l'hébergeur de manière statique (contenu constant) ou dynamique (i.e. le serveur Web doit exécuter des codes de programme, généralement PHP ou Java, avant la réponse).

Le navigateur (client) interprète la réponse, ce qui occasionne de nouvelles demandes au serveur qui ont pour but d'intégrer par exemple les images ou les données CSS associées.

# Serveur web



Les autres fonctions d'un serveur Web :

Au-delà de la diffusion simple de contenu, de nombreux serveurs web proposent des fonctions supplémentaires :

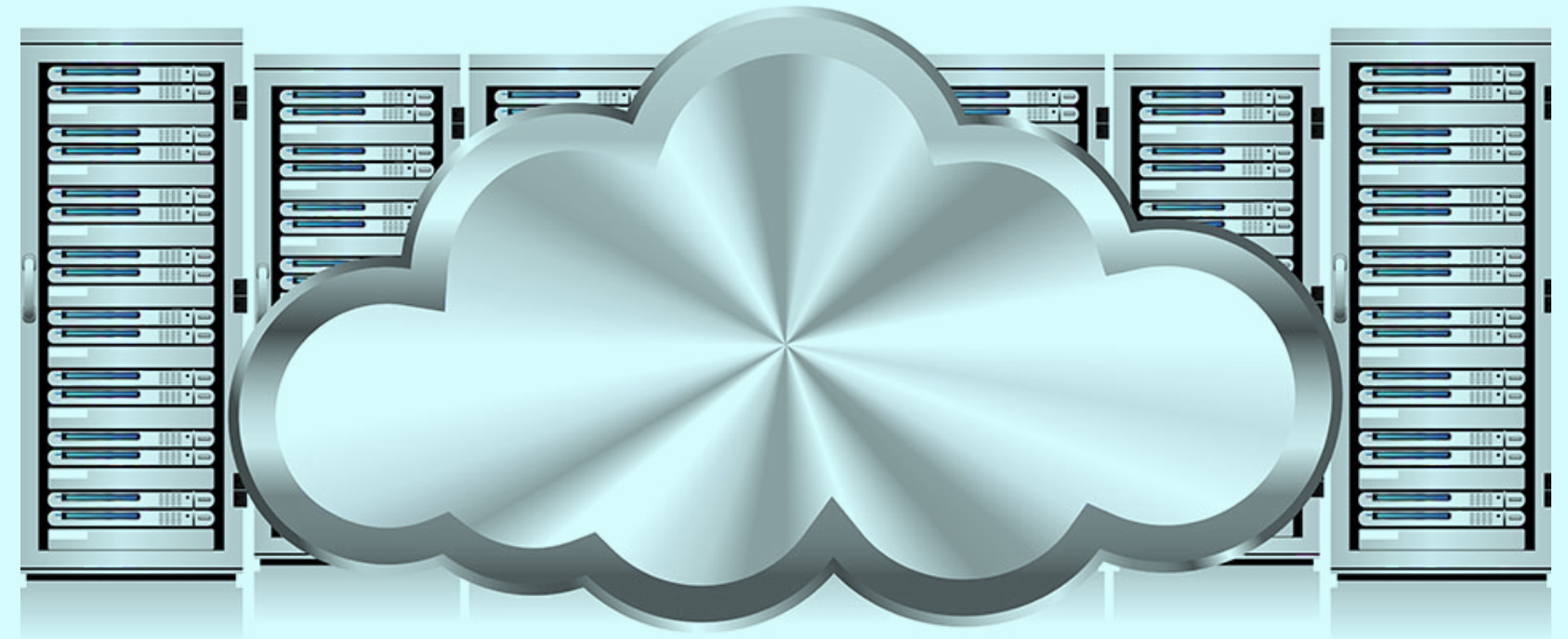
- Sécurité : Cryptage des communications entre serveur et client Web (via HTTPS)
- Identification utilisateur : Authentification HTTP autorisant l'accès à des domaines particuliers du Web
- Redirections : Redirection et réécriture d'URL (URL Rewriting) grâce au Rewrite-Engine
- Caching (mémoire cache) : Enregistrement intermédiaire de documents dynamiques pour répondre plus vite aux demandes et prévenir une surcharge du serveur Web
- Attribution de Cookies : Envoi et traitement de cookies HTTP



La transmission est effectuée via le protocole de communication HTTP(s), à travers les protocoles réseau IP et TCP (et UDP dans de rares exceptions).



# Serveur web



Serveur web autogéré ou hébergement web ?

Un hébergement web est un service en ligne de stockage et de mise à disposition de contenus web.

Le client de ce service reçoit un espace de stockage dédié pour mettre en ligne ses contenus.

La machine hôte est gérée par les administrateurs système de l'hébergeur et le serveur web applicatif est souvent mutualisé. Cela permet de réduire drastiquement les coûts d'exploitation et facilite la mise en production, mais offre moins de flexibilité.

 Il existe donc différentes manières de mettre à disposition un contenu web :

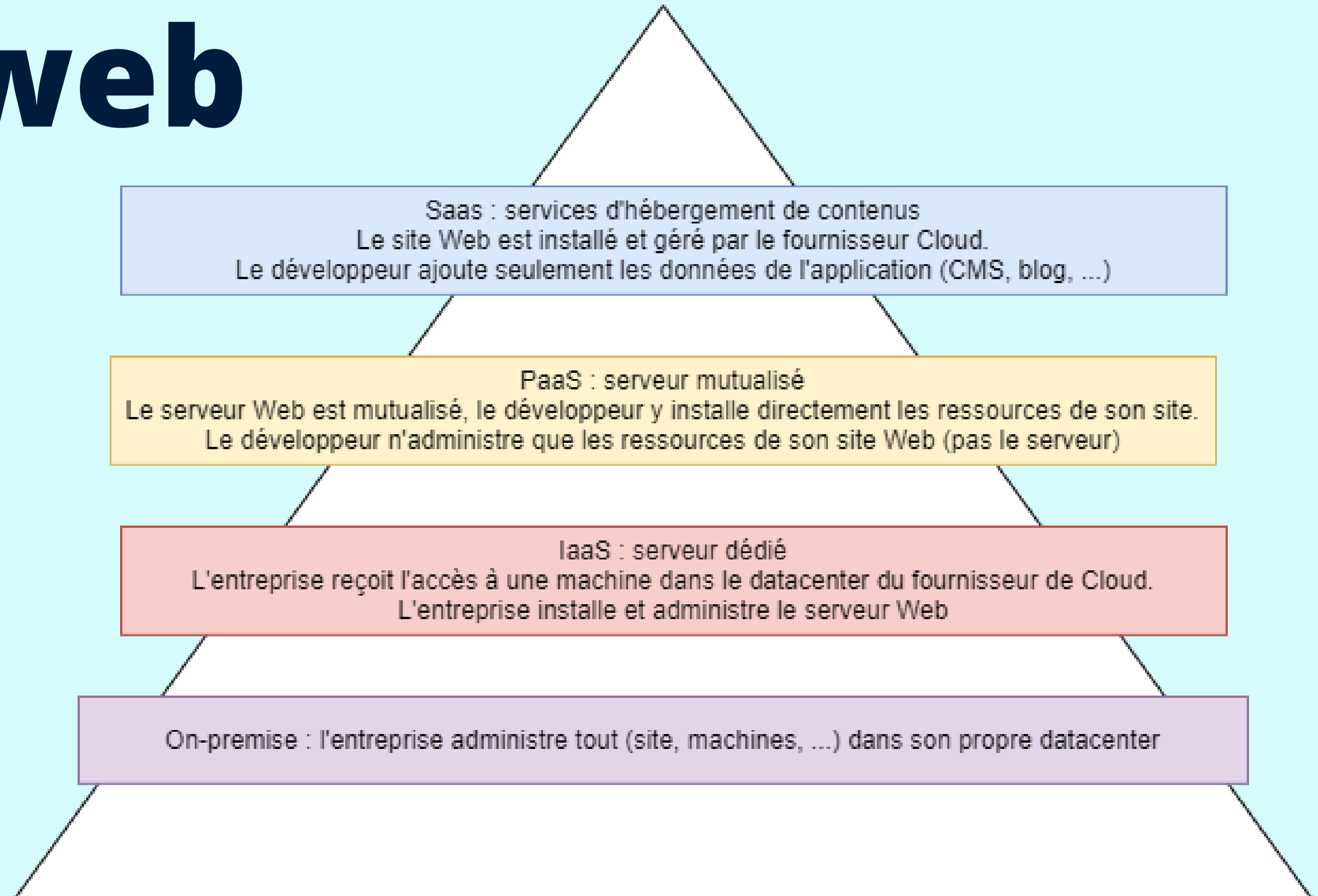
- Déployer et administrer entièrement le serveur web applicatif et la machine hôte soi-même (serveur physique)

- Administrer son propre serveur web applicatif sur une machine hôte gérée par un hébergeur (serveur dédié)

- Partager un serveur web mutualisé en y installant uniquement le programme et le contenu web à déployer (serveur virtuel, serveur cloud)

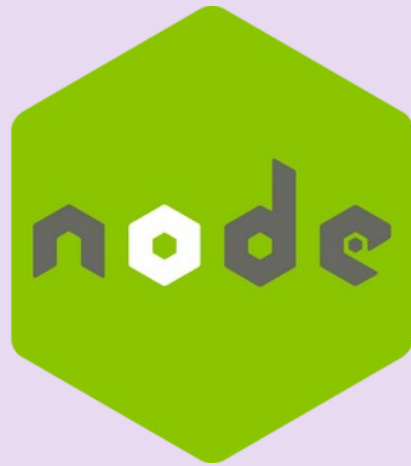
- Utiliser directement un service Cloud au-dessus d'un web serveur (CMS, blog, ...) : le site web est déjà déployé, il suffit d'y ajouter son contenu

# Serveur web



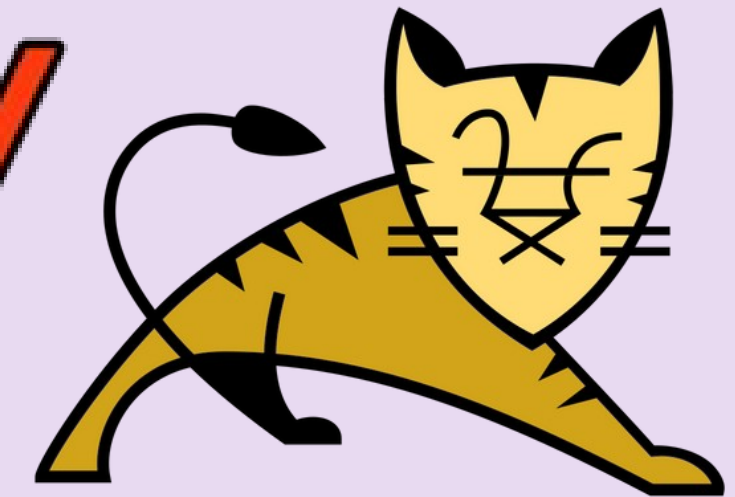
*Hébergement Web : sur site ou solutions Cloud*

# Serveur web



# NGINX

# jetty://



Serveur web applicatif : quel produit ?

Il existe de nombreux logiciels offrant les fonctionnalités des serveurs web, les principaux étant :

- Apache web server, notamment pour les sites dynamiques (JavaEE, PHP, ...). Ce serveur web est très puissant et entièrement configurable mais difficile à administrer et gourmand en ressources
- NginX, notamment pour les sites statiques, les sites aux contenus légers et en "reverse proxy" (passerelle vers des web serveurs secondaires). NginX est redoutablement efficace mais limité en fonctionnalités
- Node.js, principalement pour des applications NodeJS ou comme serveur de développement sur un ordinateur personnel
- Il en existe de nombreux autres, la plupart visant un objectif spécifique (Jetty pour les performances, ...)

# APPLICATION WEB ET CLOUD

4.2

DÉVELOPPER EN  
UTILISANT LE CLOUD

# Services web en mode SaaS

De nombreux services du Web sont déployés dans le Cloud et peuvent être intégrés directement lors du développement d'une application.

On peut citer par exemple :

- Les services de suivi opérationnel de projet (ticketing, logs, Service Desk, ...) : JIRA, Zendesk, Dynatrace, ...
- Les services de login : Google, Facebook, ...
- Les Customer relationship management (CRM) : Salesforce, Okta, ...
- Les services de communication : Slack, ...
- Les suites bureautiques : Google G Suite, Microsoft Office, ...
- La signature électronique de documents : DocuSign, ...
- Les services de base de données hébergées : Oracle Cloud DB, ...
- Les services d'indexation et de recherche sémantique : Elasticsearch, ...



Il existe de très nombreux autres services disponibles avec lesquels s'interfacer. Avant de redévelopper une fonctionnalité ou de déployer une dépendance, il est donc avantageux d'étudier l'offre disponible.

The image shows a Google sign-in interface. At the top is the Google logo. Below it, the text "Sign in with your Google Account" is displayed. In the center is a large, light gray circular placeholder for a profile picture. Below the placeholder are two input fields: "Email" and "Password". Below these fields is a blue "Sign In" button. At the bottom left, there is a checkbox labeled "Stay signed in". At the bottom right, there is a link labeled "Need help?".

# Services web en mode SaaS



L'intégration avec ces services permet d'éviter de coder, déployer et maintenir un module de code avec peu d'intérêt (service de login, base de données, ...) pour se concentrer sur les fonctionnalités business de l'application.

Cependant, ces services sont souvent payants tout au long du cycle de vie de l'application et les données sont déléguées au fournisseur du service.

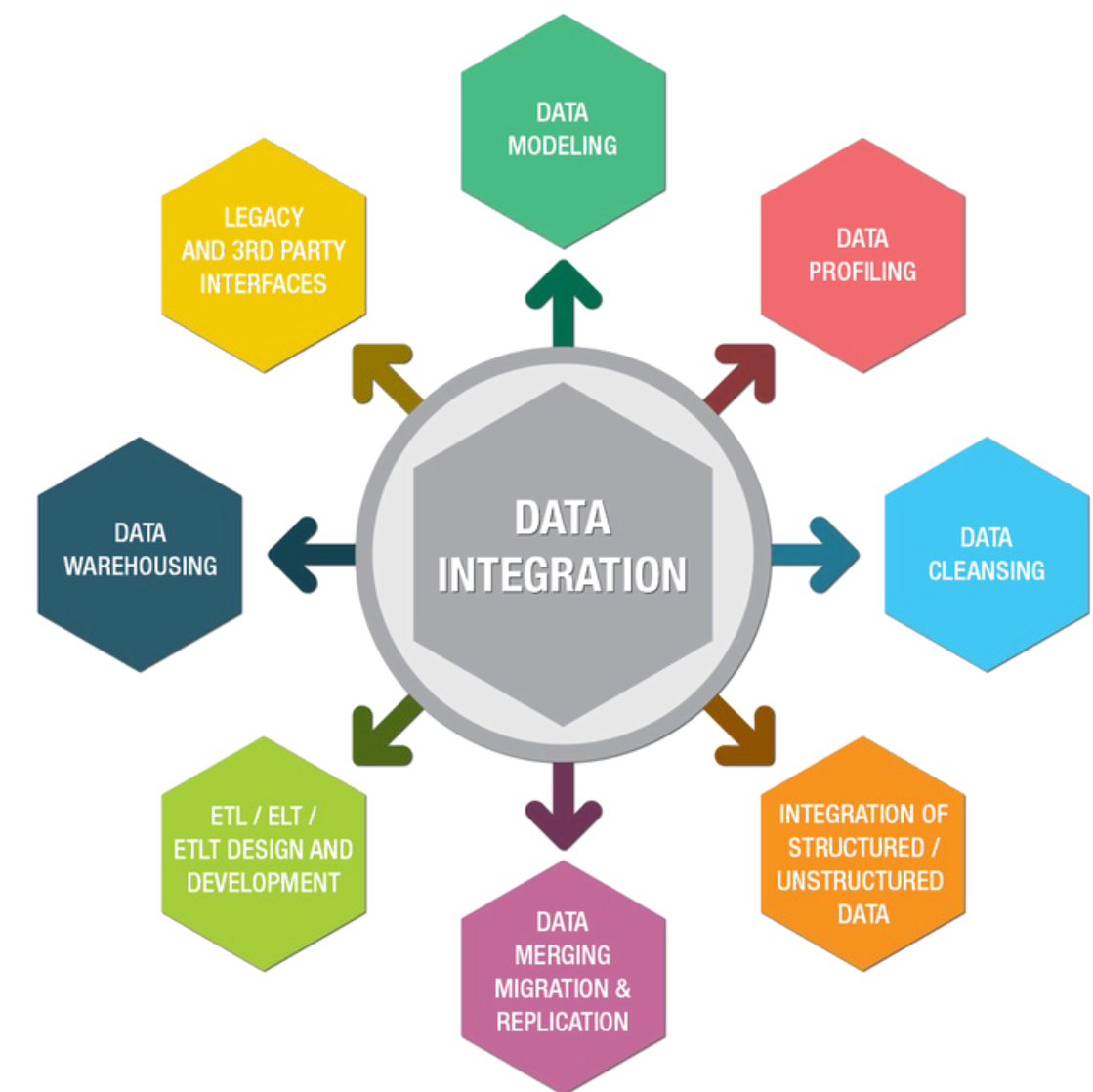




# Intégration des services Cloud dans une application Web


 Il existe 2 manières d'intégrer des services du Cloud dans une application :

Par intégration de données : synchronisation des données entre les répertoires de l'application et ceux du service hébergé. Les données sont traitées, transportées et/ou transformées pendant leur intégration. La connection au service est strictement orientée données.



# Intégration des services Cloud dans une application Web

Par intégration applicative : les différents services hébergés sont interconnectés et arrangés pour fournir une continuité applicative avec le reste de l'application. Il s'agit non seulement d'intégrer des données mais aussi d'exécuter des requêtes ou des commandes déclenchant des événements business ou des processus métier.

 L'intégration de ces services se fait en utilisant les APIs qu'ils fournissent : voir la section "Applications Modernes" chapitre III pour plus d'informations.





# Développement SaaS

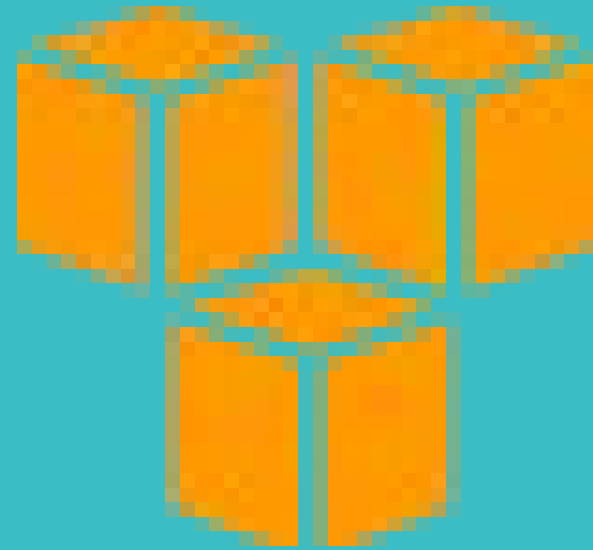


Le développement d'une application Web en mode Cloud se fait en suivant 12 principes essentiels appelés "12 factor apps" :

- Base de code : Une base de code suivie avec un système de contrôle de version, plusieurs déploiements
- Dépendances : Déclarez explicitement et isolez les dépendances
- Configuration : Stockez la configuration dans l'environnement
- Services externes : Traitez les services externes comme des ressources attachées
- Assemblez, publiez, exécutez : Séparez strictement les étapes d'assemblage et d'exécution
- Processus : Exécutez l'application comme un ou plusieurs processus sans état
- Associations de ports : Exportez les services via des associations de ports
- Concurrency : Grossissez à l'aide du modèle de processus
- Jetable : Maximisez la robustesse avec des démarrages rapides et des arrêts gracieux
- Parité dev/prod : Gardez le développement, la validation et la production aussi proches que possible
- Logs : Traitez les logs comme des flux d'évènements
- Processus d'administration : Lancez les processus d'administration et de maintenance comme des one-off-processes

Pour plus d'informations : <https://12factor.net/fr/>

# Les offres du marché (2023)



Amazon Web Services (AWS) :  
leader du marché, notamment en  
IaaS et en diversité de services



Microsoft Azure :  
#2 du marché, notamment en  
offres hybrides, apprécié des  
entreprises



Google Cloud Platform

Google Cloud Platform :  
#3 en forte croissance

## Les généralistes

# Les offres du marché (2023)



Salesforce :  
Spécialiste du SaaS



Cisco Cloud :  
Cloud orienté réseaux



Alibaba Cloud :  
leader en Chine



Oracle Cloud :  
Cloud autour de la base de  
données Oracle

## Les niches



IBM + RedHat :  
Cloud fortement hybride

# Les offres du marché

Figure 1. Magic Quadrant for Cloud Infrastructure as a Service, Worldwide



Source: Gartner (July 2019)

# Pour aller plus loin : les microservices

L'augmentation considérable des débits de données sur Internet et l'arrivée du Cloud ont vu le développement d'une nouvelle infrastructure logicielle : les microservices.

Les microservices sont des "morceaux" d'application faiblement couplés les uns aux autres et permettant chacun de réaliser une fonction dédiée de l'application. Ces découpages sont donc orientés services (fonctionnalités) et non dépendances techniques (base de données, ...)



Un exemple de microservice est un service de login centralisé : ce service est complètement indépendant (données dédiées, ...), fournit une fonctionnalité (le login), mais n'a pas d'utilité concrète s'il n'est pas assemblé dans une application finale.



Les architectures de microservices sont indépendantes du Cloud, mais en pratique elles se reposent souvent sur les fonctionnalités des plateformes PaaS qui répondent à leurs besoins opérationnels importants : équilibrage de charge, routage entre services, ...

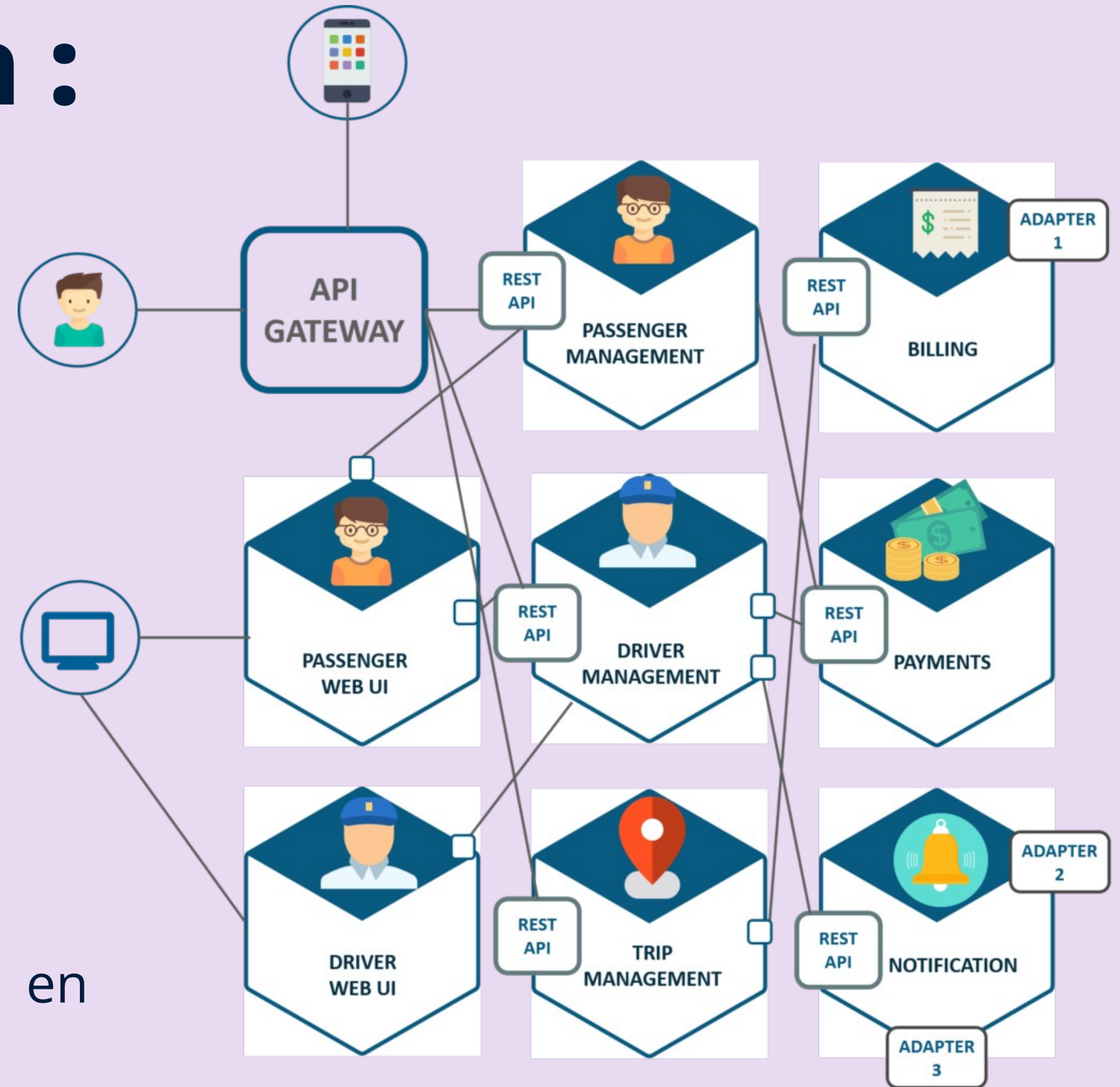
En revanche, déployer dans le Cloud ne veut pas dire créer des microservices !



Dans la pratique, les applications dites microservices sont souvent des applications hybrides : elles segmentent les grandes fonctionnalités en micro applications mais s'autorisent souvent à partager des ressources communes (base de données, ...)



# Pour aller plus loin : les microservices



Un exemple d'application architecturée en microservices

# Pour aller plus loin : lectures recommandées

- Architectures logicielle  
[https://www.eni-training.com/client\\_net/mediabook.aspx?idR=168690](https://www.eni-training.com/client_net/mediabook.aspx?idR=168690)
- Objectif Cloud  
[https://www.eni-training.com/client\\_net/mediabook.aspx?idR=92065](https://www.eni-training.com/client_net/mediabook.aspx?idR=92065)
- Cloud computing  
[https://www.eni-training.com/client\\_net/mediabook.aspx?idR=46636](https://www.eni-training.com/client_net/mediabook.aspx?idR=46636)