

# HTTP - Infrastructure

TEchnologies Internet (TEI)

---

Olivier Liechti

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

# Schedule & Evaluation

4w

## Foundations

Java IO

Network programming with the Socket API

1 grade : programming  
exercise in class  
(25.03.2013)

6w

## HTTP

Protocol + Client/Server Programming  
Infrastructure + Performance/Load Testing

3 grades : http proxy in  
Java lab, infra labs 1+2,  
theory test (includes  
foundations topics)

3w

## LDAP

Data Model + Protocol + querying  
Tools (server, browser) + import procedure

1 grade : practical test  
in class (27.05.2013)

3w

## Web Services

Big Web Services vs RESTful Web Services  
Using and creating REST APIs

1 grade : REST lab

# *Qualités systémiques*

# Notion de qualité systémique

- > **Comment définir le cahier des charges d'un système informatique?**
  - Le système est construit pour offrir un **"service"** (web, messagerie, contrôle d'accès, gestion de stocks, etc.).
  - La première partie du cahier des charges définit les **exigences fonctionnelles**. Elle spécifie ce que le service offre aux utilisateurs (le **"quoi"**).
  - La deuxième partie du cahier des charges définit les **exigences non-fonctionnelles**. Elle spécifie différentes **qualités** que doit avoir le système (le **"comment"**).
- > **Pour la 2ème partie, on parle donc de "qualité systémique":**
  - Il existe de nombreuses qualités différentes.
  - Les choix d'architecture vont permettre d'augmenter le niveau de ces qualités, souvent avec un certain coût.
  - En fonction de la situation, il faut déterminer quelles qualités sont les plus importantes et quantifier les besoins.

# Un exemple

**Système**  
Véhicule motorisé

**Exigences fonctionnelles**  
Déplacer des personnes

**Exigences non-fonctionnelles**  
Performance  
Capacité  
Fiabilité  
Coût  
Esthétique  
Facilité d'utilisation





*Certaines  
qualités font  
l'objet de forces  
antagonistes.*

*Définir  
l'architecture du  
système, c'est  
trouver l'équilibre  
entre ces forces!*





# Quelques qualités systémiques importantes

## > Temps de réponse (response time)

- Mesure la vitesse avec laquelle une réponse est donnée aux utilisateurs
- Important du point de vue de l'utilisateur.

## > Débit (throughput)

- Mesure le nombre de requêtes traitées dans un intervalle de temps.
- Important du point de vue du fournisseur de service.

## > Scalabilité (scalability)

- Mesure la facilité avec laquelle le système peut être adapté pour traiter une charge supplémentaire.
- Idéalement: scalabilité linéaire. "J'ai deux fois plus d'utilisateurs, je dois simplement doubler le nombre de serveurs."

## > Disponibilité (availability)

- Mesure le pourcentage de temps pendant lequel le service peut être utilisé.
- Disponibilité de 99% = indisponibilité moyenne de 3.65 jours par année, soit 1 heure et 41 minutes par semaine.

# Quelques qualités systémiques importantes

## > **Fiabilité (reliability)**

- Mesure la capacité du système à "remplir sa fonction" sur une période.
- Un système peut avoir une fiabilité plus élevée que celle d'un de ces sous-système (notamment grâce à la redondance).
- Mesure exprimée en "temps moyen entre 2 pannes" (Mean Time Between Failures ou MTBF).

## > **Evolutivité (evolutivity)**

- Mesure la facilité avec laquelle le système pourra être adapté pour satisfaire à de nouvelles exigences (fonctionnelles et non-fonctionnelles!).

## > **Sécurité (security)**

- Mesure la capacité du système à empêcher une utilisation abusive.

## > **Facilité d'administration (manageability)**

- Mesure la facilité avec laquelle l'état du système pourra être surveillé, les pannes détectées, des opérations de maintenance réalisées, etc.



# Décrire l'architecture d'un système informatique

## > Tiers

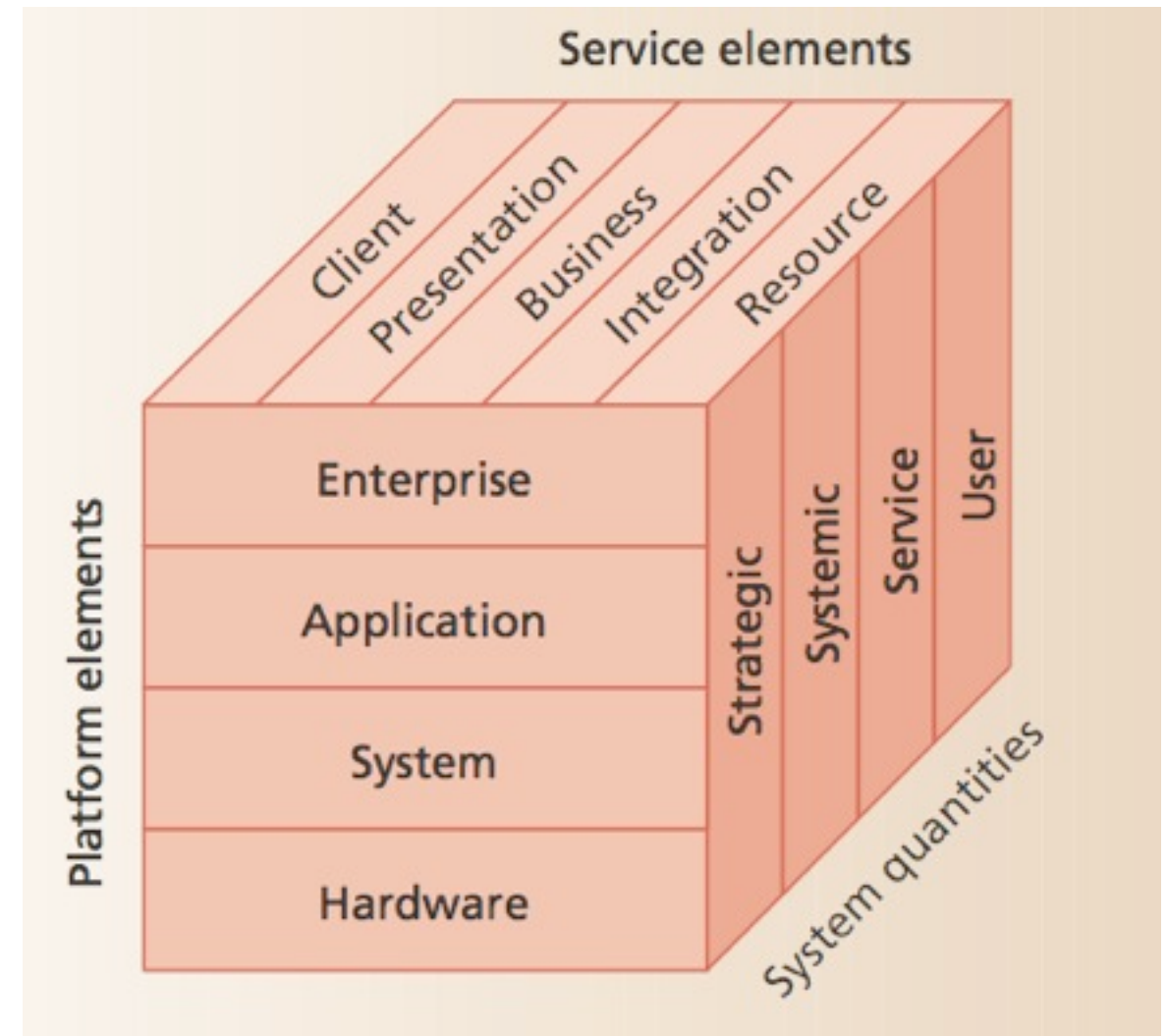
- Répartition des composants par rapport au traitement séquentiel d'une requête utilisateur.
- Client - Métier - Données

## > Couches

- Répartition des composants depuis la couche matérielle jusqu'à la couche information.

## > Qualités systémiques.

- Chaque qualité du système doit être étudiée de manière globale.
- A travers les tiers.
- A travers les couches.



Joseph Williams, "Correctly Assessing the "ilities" Requires More than Marketing Hype", IT Professional, Volume 2, Issue 6 (November 2000)

# *Un cas particulier: le serveur apache (httpd)*

# Le serveur apache

## > Historique

- "Descendant" d'un des premiers serveurs http: httpd du NCSA
- Utilisé depuis 1996
- Géré par une des premières et plus prospères communautés Open Source: le Groupe Apache (qui créer la "Apache Software Foundation" en 1999)
- Versions majeures: 1.3, 2.0, 2.2

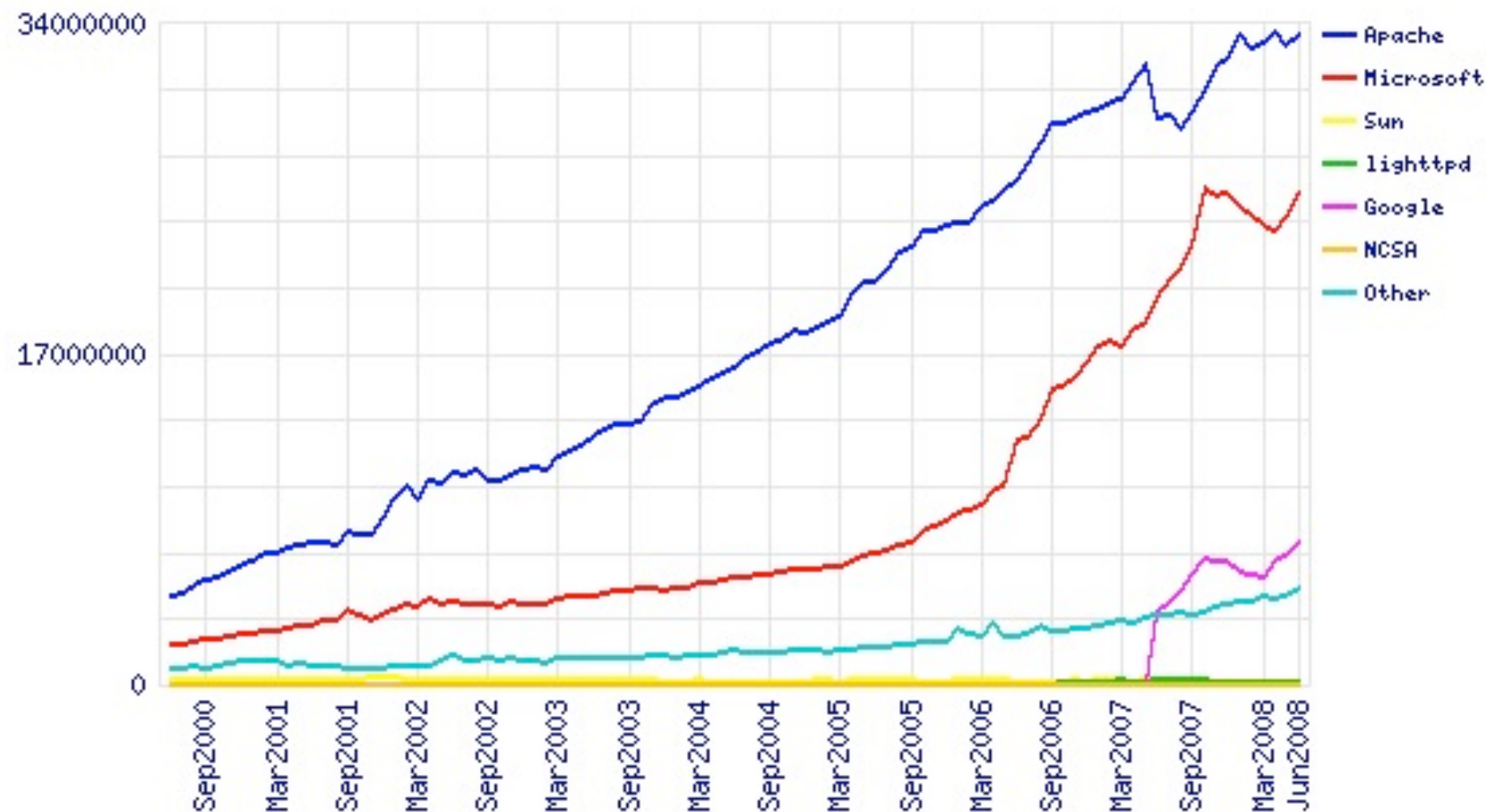
## > Popularité

- Apache est très populaire sur Internet: à peu près 50% des sites!

## > Avantages

- Stabilité, performance, flexibilité, extensibilité (architecture modulaire)
- Grande communauté d'utilisateurs, beaucoup de références
- Open Source
- Gratuit

# Le serveur apache



[http://news.netcraft.com/archives/2008/06/22/june\\_2008\\_web\\_server\\_survey.html](http://news.netcraft.com/archives/2008/06/22/june_2008_web_server_survey.html)

Developer	May 2008	Percent	June 2008	Percent	Change
Apache	32,839,213	48.04%	33,419,978	46.97%	-1.07
Microsoft	24,040,260	35.17%	25,292,798	35.54%	0.38
Google	6,592,598	9.64%	7,256,264	10.20%	0.55
Sun	173,679	0.25%	176,726	0.25%	-0.01
lighttpd	92,041	0.13%	100,536	0.14%	0.01



# Installer le serveur apache

## > Solution 1: "laisser faire les autres"

- Beaucoup de systèmes d'exploitation \*NIX intègrent le serveur apache.
- **Attention:** pas forcément la version 2.2, pas forcément tous les modules, pas forcément les bonnes options de compilation...

## > Solution 2: "installer une version standard"

- <http://httpd.apache.org/download.cgi>
- A nouveau, pas de contrôle sur tous les paramètres de compilation.

## > Solution 3: "compiler les sources"

- Contrôle total
- Un peu plus de travail...



La commande httpd accepte des "flags" qui vous permettent de connaître les détails de votre installation!



# Apache HTTP Server Version 2.2 Documentation

Available Languages: [de](#) | [en](#) | [es](#) | [fr](#) | [ja](#) | [ko](#) | [pt-br](#) | [tr](#)

Google Search

## Release Notes

[New features with Apache 2.1/2.2](#)  
[New features with Apache 2.0](#)  
[Upgrading to 2.2 from 2.0](#)  
[Apache License](#)

## Reference Manual

[Compiling and Installing](#)  
[Starting](#)  
[Stopping or Restarting](#)  
[Run-time Configuration Directives](#)  
[Directive Quick-Reference](#)  
[Modules](#)  
[Multi-Processing Modules \(MPMs\)](#)  
[Filters](#)  
[Handlers](#)  
[Server and Supporting Programs](#)  
[Glossary](#)

## Users' Guide

[Binding](#)  
[Configuration Files](#)  
[Configuration Sections](#)  
[Content Caching](#)  
[Content Negotiation](#)  
[Dynamic Shared Objects \(DSO\)](#)  
[Environment Variables](#)  
[Log Files](#)  
[Mapping URLs to the Filesystem](#)  
[Performance Tuning](#)  
[Security Tips](#)  
[Server-Wide Configuration](#)  
[SSL/TLS Encryption](#)  
[Suexec Execution for CGI](#)  
[URL Rewriting Guide](#)  
[Virtual Hosts](#)

## How-To / Tutorials

[Authentication, Authorization, and Access Control](#)  
[CGI: Dynamic Content](#)  
[.htaccess files](#)  
[Server Side Includes \(SSI\)](#)  
[Per-user Web Directories \(public\\_html\)](#)

## Platform Specific Notes

[Microsoft Windows](#)  
[Novell NetWare](#)  
[EBCDIC Port](#)

## Other Topics

[Frequently Asked Questions](#)  
[Sitemap](#)  
[Documentation for Developers](#)  
[Other Notes](#)

Available Languages: [de](#) | [en](#) | [es](#) | [fr](#) | [ja](#) | [ko](#) | [pt-br](#) | [tr](#)

# *Performance et scalabilité du service HTTP*

# Optimiser les performances du service HTTP

## > Deux approches complémentaires

- Scaling Up (scalabilité verticale)
- Scaling Out (scalabilité horizontale)

## > Scale Up

- On vise à améliorer les **performances d'un composant**, en procédant à des optimisations et en ajoutant des ressources (CPU, mémoire, bande passante, etc.).

## > Scale Out

- On vise à améliorer les **performances du système global**, en répartissant la charge entre plusieurs composants équivalents.

Scale Up:  
"Dopez vos performances!"



Scale Out:  
"L'union fait la force!"





# Optimiser les performances du service HTTP

## > Références

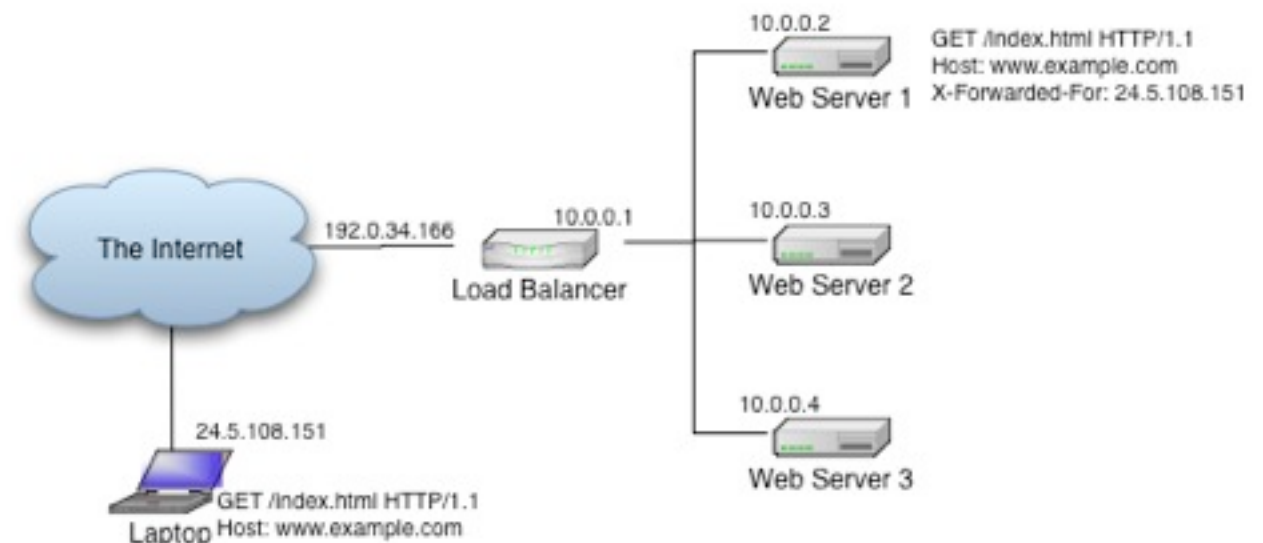
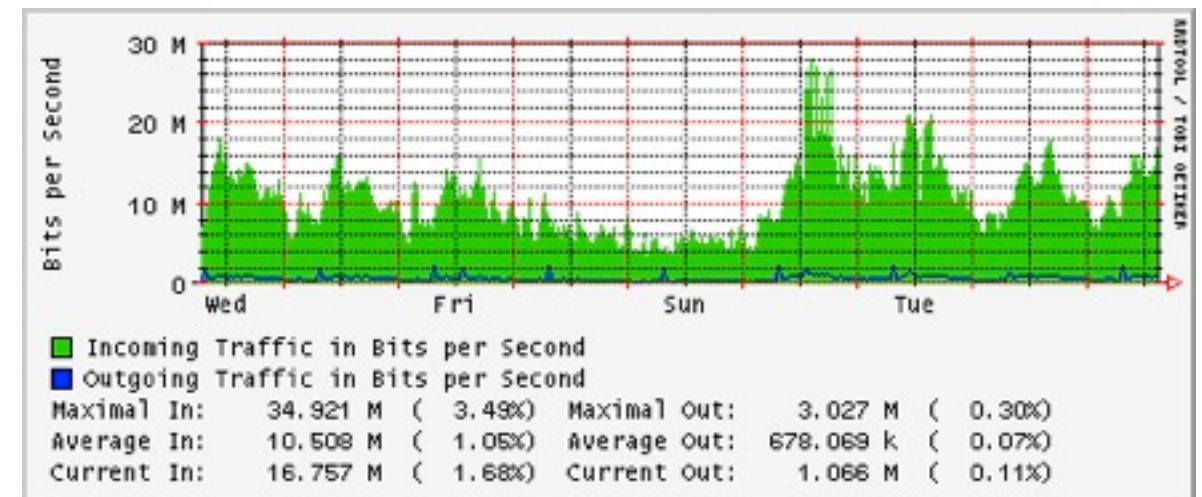
- Sander Temme, expert performance
- Articles et présentations: <http://people.apache.org/~sctemme>

## > Scale Up

- Avant d'optimiser, mesurer!
- Choisir le bon modèle de (multi-process vs. multi-thread), en fonction de la plateforme (système d'exploitation)

## > Scale Out

- Utiliser Apache en mode reverse proxy et exploiter les modules de caching.



# Caching

## > Qu'est-ce qu'un "cache"

- Une zone mémoire où on peut stocker des données, que l'on pourrait (ré-)utiliser dans le futur.
- "Pourquoi recalculer quelque chose que j'ai déjà calculé il y a 5 minutes?"
- "Si je dois aller acheter une bouteille d'eau, autant acheter directement une harasse pour avoir une réserve."

## > Quels sont les problèmes liés à l'utilisation d'un cache?

- "Fraîcheur" des données: est-ce que ce que je lis dans le cache est encore une copie fidèle de l'original?
- Quelle politique définit la gestion du cache (ajouter, supprimer des données, etc.)
- Synchronisation: que se passe-t-il quand j'utilise plusieurs caches (par exemple dans un cluster)?



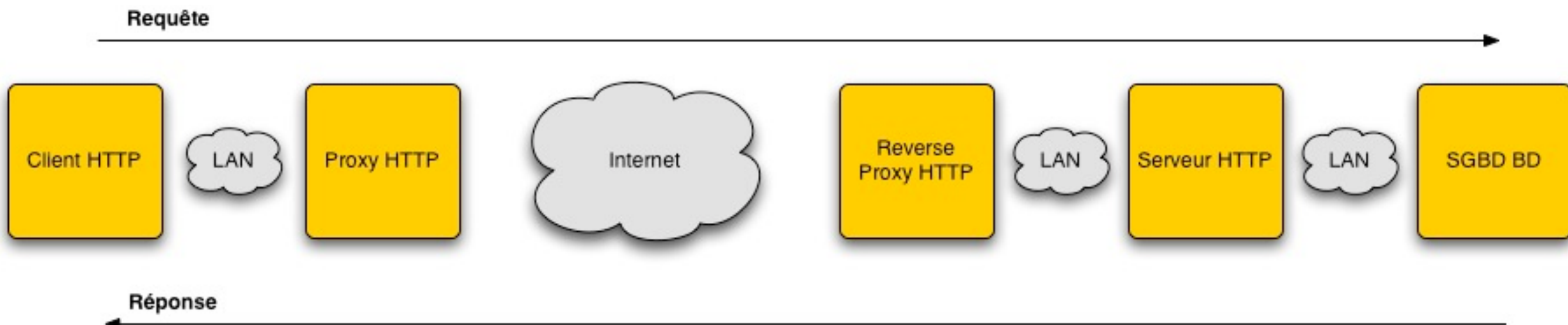
# Caching et HTTP

## > Exemple:

- On considère une application web, qui permet d'accéder à des ressources stockées dans une base de données.

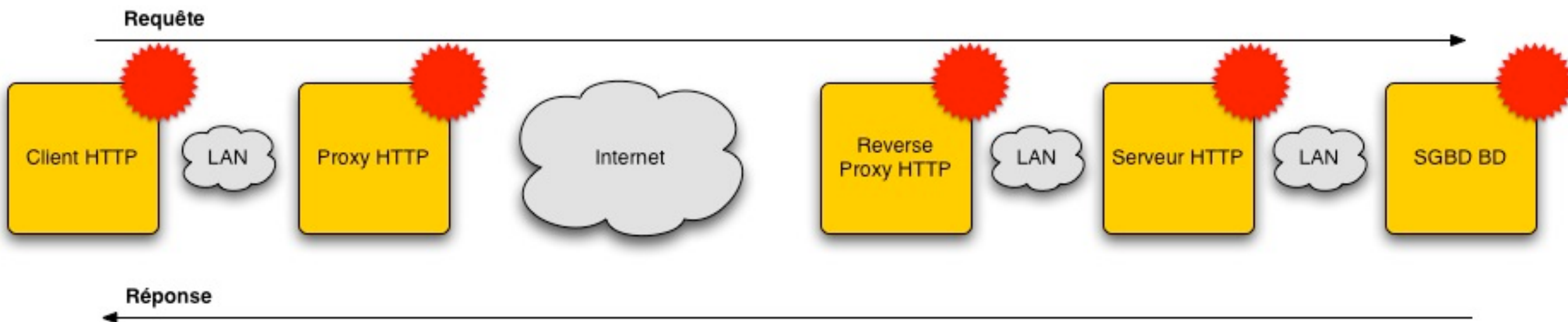
## > Sans cache:

- Chaque requête fait "un long voyage" (du client à la base de données)
  - Plusieurs réseaux sont traversés.
  - Plusieurs composants doivent fournir un travail.
- > **Question: où pourrait-on gérer un cache, pour pouvoir réutiliser toute ou partie des données liées à la requête?**



# Réponse: à tous les niveaux!

- > **Dans le navigateur:**
  - On ne va pas télécharger le logo de CNN tous les jours!
- > **Dans un proxy du côté client:**
  - On ne va pas télécharger l'article du jour pour chaque utilisateur!
- > **Dans un proxy du côté serveur:**
  - On ne va pas recalculer la page d'accueil pour chaque visiteur!
- > **Dans le serveur http (niveau de l'application web):**
  - On ne va pas tout recalculer, on ne va pas chaque fois aller jusqu'à la DB.
- > **Dans la base de données (on ne va pas aller jusqu'au disque)**





# Caching: que dit la spécification HTTP?

- > **Une section entière est consacrée à ce sujet:**
  - <http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13>
  - 26 pages!
- > **Modèle d'expiration (13.2)**
- > **Modèle de validation (13.3)**
- > **Pertinence de la mise en cache des réponses (13.4)**
- > **Construction de réponses grâce aux caches (13.5)**

## Expiration et validation

*Caching would be useless if it did not significantly improve performance.*

*The goal of caching in HTTP/1.1 is to eliminate the need to send requests in many cases, and to eliminate the need to send full responses in many other cases.*

*The former reduces the number of network round-trips required for many operations; we use an "**expiration**" mechanism for this purpose (see section 13.2). The latter reduces network bandwidth requirements; we use a "**validation**" mechanism for this purpose (see section 13.3).*

# Expiration et validation

## > Expiration

- Quand on reçoit une réponse, on veut déterminer combien de temps elle va rester "valide".
- Pendant ce temps, il sera possible de récupérer la représentation de la ressource dans le cache.

## > Validation

- Après expiration, on n'est pas sûr que le contenu du cache soit encore une représentation "fraîche" de la ressource.
- On peut alors procéder en 2 temps:
  - ➔ On envoie une première requête au serveur pour demander si la ressource a changé.
  - ➔ Si oui, c'est seulement dans un deuxième temps qu'on récupère la nouvelle représentation de la ressource.
- **Ressources (à étudier pour les travaux écrits et l'examen!!)**
  - <http://www.peej.co.uk/articles/http-caching.html>
  - <http://www.symkat.com/understanding-http-caching>

# Réaliser un "caching proxy" avec Apache

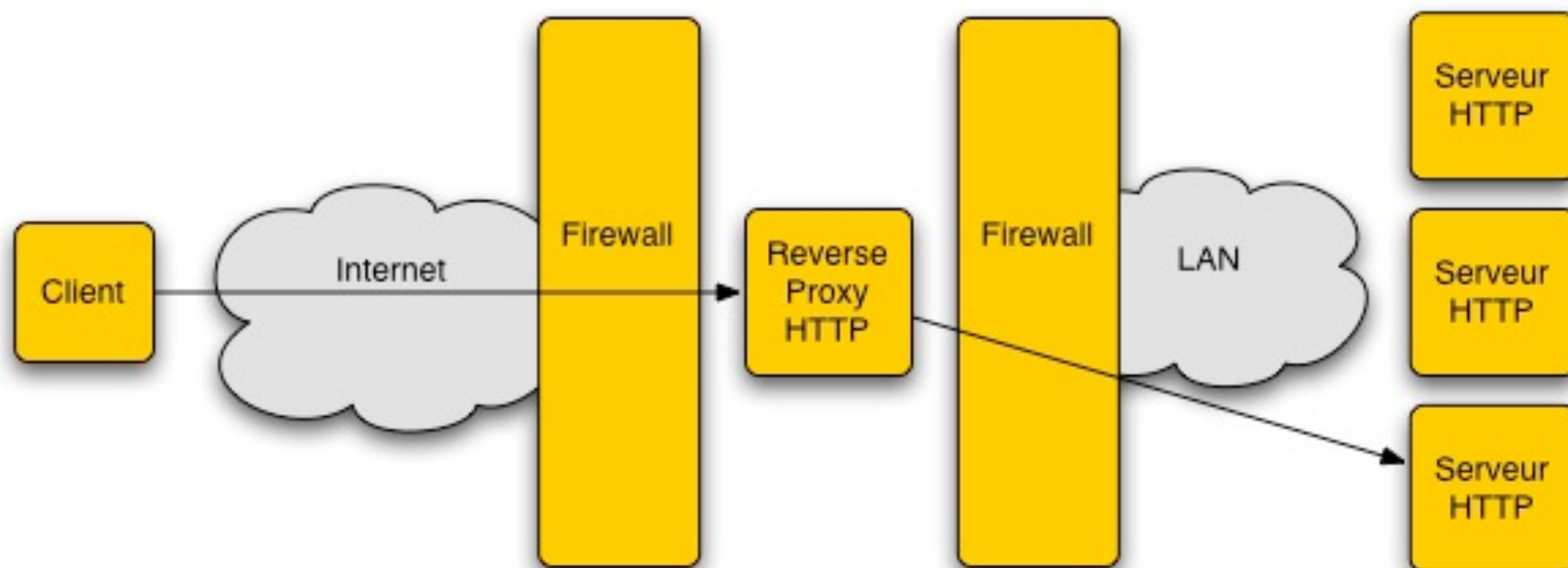
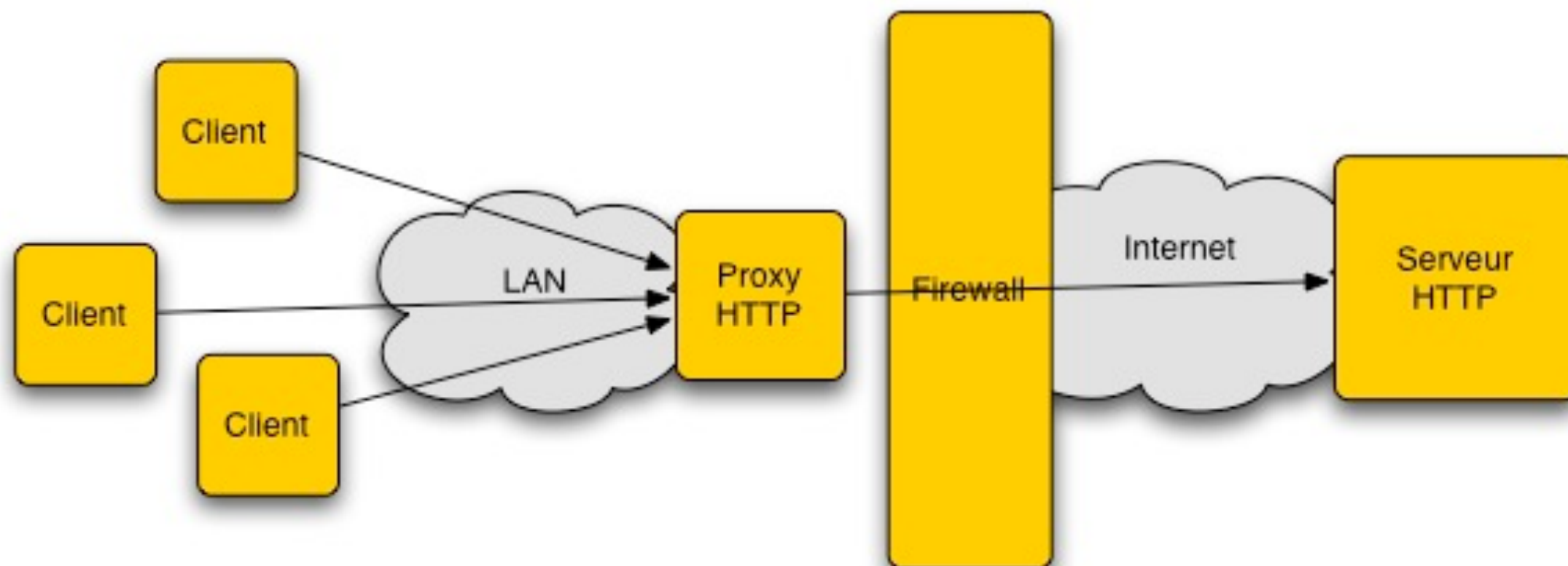
## > Forward Proxy

- "Près du client" (le proxy qu'on utilise pour sortir)
- Utilisé pour des raisons de sécurité et de performance.

## > Reverse Proxy

- "Près du serveur"
- Utilisé pour des raisons de sécurité, de performance, de disponibilité, de scalabilité





# Configuration Apache: mod\_proxy

```
ProxyRequests On
ProxyVia On

<Proxy *>
Order deny,allow
Deny from all
Allow from internal.example.com
</Proxy>
```

## Forward Proxy

```
ProxyRequests Off

<Proxy *>
Order deny,allow
Allow from all
</Proxy>

ProxyPass /foo http://foo.example.com/bar
ProxyPassReverse /foo http://foo.example.com/bar
```

### Directives

[AllowCONNECT](#)  
[BalancerMember](#)  
[NoProxy](#)  
[<Proxy>](#)  
[ProxyBadHeader](#)  
[ProxyBlock](#)  
[ProxyDomain](#)  
[ProxyErrorOverride](#)  
[ProxyFtpDirCharset](#)  
[ProxyIOBufferSize](#)  
[<ProxyMatch>](#)  
[ProxyMaxForwards](#)  
[ProxyPass](#)  
[ProxyPassInterpolateEnv](#)  
[ProxyPassMatch](#)  
[ProxyPassReverse](#)  
[ProxyPassReverseCookieDomain](#)  
[ProxyPassReverseCookiePath](#)  
[ProxyPreserveHost](#)  
[ProxyReceiveBufferSize](#)  
[ProxyRemote](#)  
[ProxyRemoteMatch](#)  
[ProxyRequests](#)  
[ProxySet](#)  
[ProxyStatus](#)  
[ProxyTimeout](#)  
[ProxyVia](#)

# Configuration Apache: mod\_cache

## Sample httpd.conf

```
#
# Sample Cache Configuration
#
LoadModule cache_module modules/mod_cache.so

<IfModule mod_cache.c>
    #LoadModule disk_cache_module modules/mod_disk_cache.so
    # If you want to use mod_disk_cache instead of mod_mem_cache,
    # uncomment the line above and comment out the LoadModule line
    # below.
    <IfModule mod_disk_cache.c>
        CacheRoot c:/cachroot
        CacheEnable disk /
        CacheDirLevels 5
        CacheDirLength 3
    </IfModule>

    LoadModule mem_cache_module modules/mod_mem_cache.so
    <IfModule mod_mem_cache.c>
        CacheEnable mem /
        MCacheSize 4096
        MCacheMaxObjectCount 100
        MCacheMinObjectSize 1
        MCacheMaxObjectSize 2048
    </IfModule>

    # When acting as a proxy, don't cache the list of security updates
    CacheDisable http://security.update.server/update-list/
</IfModule>
```

## Directives

[CacheDefaultExpire](#)  
[CacheDisable](#)  
[CacheEnable](#)  
[CacheIgnoreCacheControl](#)  
[CacheIgnoreHeaders](#)  
[CacheIgnoreNoLastMod](#)  
[CacheIgnoreQueryString](#)  
[CacheLastModifiedFactor](#)  
[CacheMaxExpire](#)  
[CacheStoreNoStore](#)  
[CacheStorePrivate](#)

## Topics

- [Related Modules and Directives](#)
- [Sample Configuration](#)

## See also

- [Caching Guide](#)

# *Disponibilité du service HTTP*



# Disponibilité du service HTTP

## > Objectif

- Garantir que le service HTTP de mon entreprise va être utilisable "la plus grande partie du temps".

## > Causes pour lesquelles mon service pourrait être inutilisable

- **Causes prévisibles:** mises à jour et déploiement d'une nouvelle version (du système d'exploitation, du serveur HTTP, de l'application web, etc.)
- **Causes imprévisibles:** pannes matérielles, pannes de réseau, bugs dans le serveur HTTP, surcharge, attaque (e.g. denial of service), catastrophe naturelle, etc.

## > Mesurer la disponibilité

- $A = \text{MTBF} / (\text{MTBF} + \text{MTTR})$
- MTBF: Mean Time Between Failure (temps moyen entre 2 pannes)
- MTTR: Mean Time To Repair (temps moyen pour résoudre un problème)

## > Question:

- Sur la base de cette définition et en imaginant que le MTBF d'un serveur HTTP (physique) soit de 100 jours, comment peut-on augmenter la disponibilité du service HTTP? Quelles sont les différentes possibilités et leurs avantages respectifs?



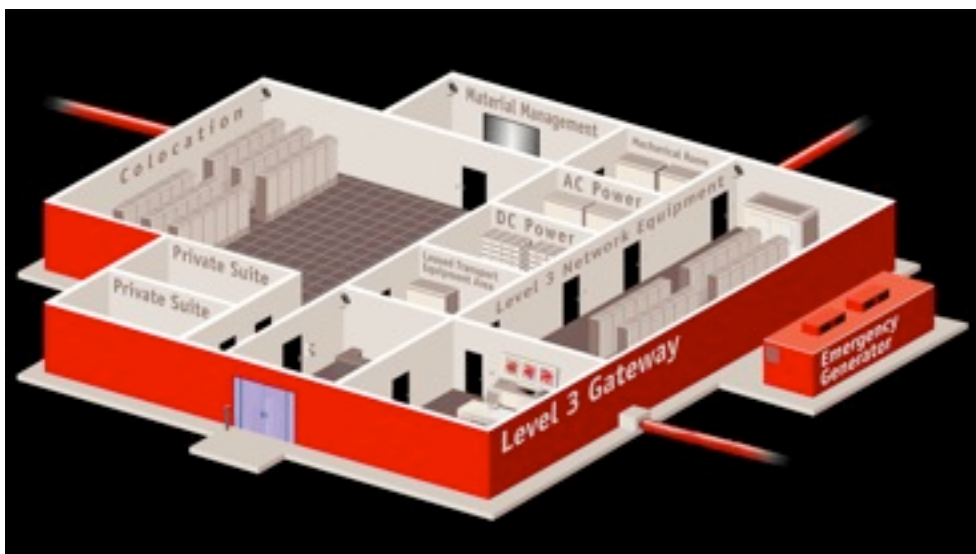
# Disponibilité du service HTTP

- > **Taux de disponibilité: 98%**
  - indisponibilité de 7.3 jours par an, soit 3 heures 22 minutes par semaine
- > **Taux de disponibilité: 99%**
  - indisponibilité de 3.65 jours par an, soit 1 heure 41 minutes par semaine
- > **Taux de disponibilité: 99.9%**
  - indisponibilité de 8 heures 45 min par an, soit 10 min 5 sec par semaine
- > **Taux de disponibilité: 99.99%**
  - indisponibilité de 52.5 minutes par an, soit 1 minute par semaine
- > **Taux de disponibilité: 99.999%**
  - indisponibilité de 5.25 minutes par an, soit 6 secondes par semaine
- > **Taux de disponibilité: 99.9999%**
  - indisponibilité de 31.5 secondes par an, soit 0.6 secondes par semaine
  - indisponibilité de 6 minutes en 11.4 années!

# Comment augmenter la disponibilité?

- > **La disponibilité du système dépend de la qualité et de la fiabilité des composants du système:**
  - Les spécifications des composants matériels indiquent souvent une valeur de MTBF.
  - Il sera difficile d'assurer la disponibilité d'un service si le logiciel est "buggé".
- > **La disponibilité dépend également de la charge supportée par le système:**
  - Un système qui se comporte bien à charge normale peut exhiber des problèmes quand un seuil est atteint (contentions).
- > **Ajouter de la redondance à l'intérieur du système, à différents niveaux, permet d'augmenter la disponibilité:**
  - Si un composant tombe en panne, on peut "basculer" sur un composant équivalent de rechange.

# Composants qui peuvent être redondants



*Centre de calcul*



*Switch*

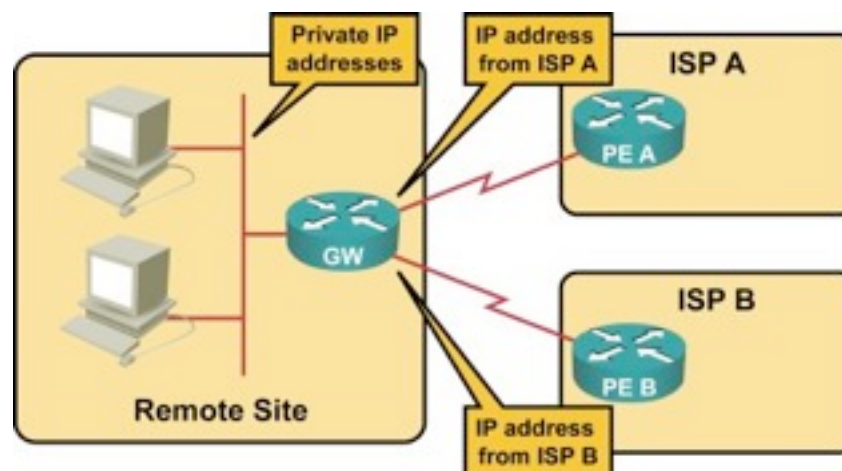


*Disques*



*Serveurs*

heig-vd



*Accès Internet*



*Alimentations électriques*

olivier.liechti@heig-vd.ch

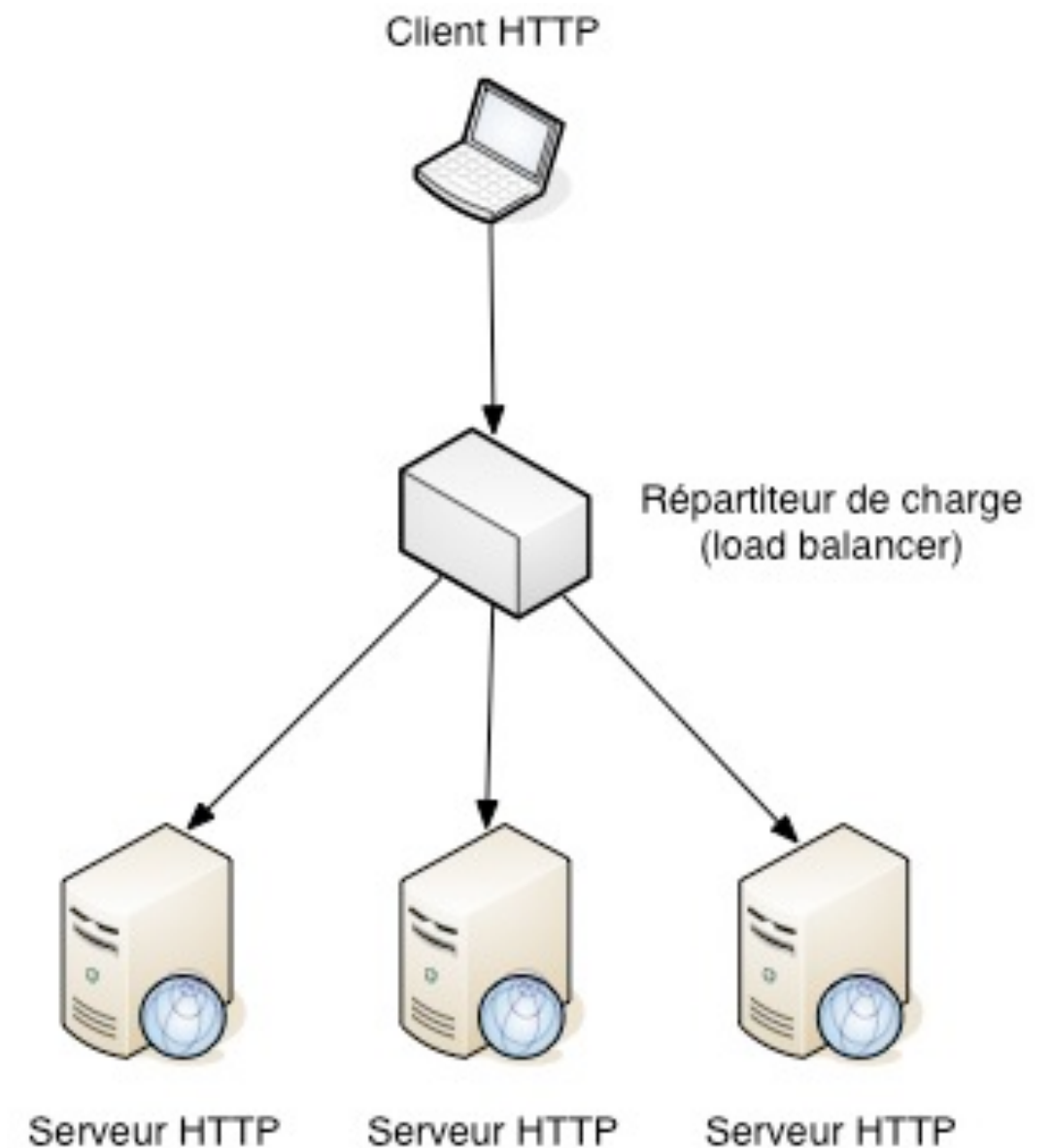
# Service redondant avec le serveur Apache (httpd)

## > Architecture

- au moins 2 serveurs physiques (ou deux machines virtuelles).
- au moins 2 serveurs httpd configurés de manière identique.
- 1 répartiteur de charge.

## > Fonctionnement

- Les requêtes HTTP sont envoyées vers le répartiteur de charge (Virtual IP address).
- Le répartiteur de charge distribue les requêtes entre les différents serveurs.
- Une politique définit les modalités de cette distribution!





# Comment réaliser le répartiteur de charge?

## > Solution matérielle

- Des entreprises fournissent des équipements spécialisés pour la répartition de charge.
- Ils peuvent être configurés de manière flexible.
- Ils peuvent être eux-mêmes déployés de manière à être hautement disponibles.
- Exemple: **BIG-IP de F5**. Voir: <http://www.f5.com/solutions/resources/deployment-guides/>



## > Solution logicielle

- La fonction de répartition de charge peut aussi être assurée par un logiciel installé sur un serveur "standard".
- **Exemple: Apache mod\_proxy\_balancer + mod\_proxy**
  - ➔ [http://httpd.apache.org/docs/2.2/mod/mod\\_proxy\\_balancer.html](http://httpd.apache.org/docs/2.2/mod/mod_proxy_balancer.html)
  - ➔ [http://httpd.apache.org/docs/2.2/mod/mod\\_proxy.html#proxypass](http://httpd.apache.org/docs/2.2/mod/mod_proxy.html#proxypass)

# Affinité de session (sticky sessions) (1)

## > Situation:

- On a besoin de haute disponibilité pour une application.
- On installe un **cluster** de serveurs apache.
- Un load balancer distribue les requêtes entre les instances du cluster.
- L'application gère une **session** (e.g. shopping cart).

## > Problème:

- Que se passe-t-il si les requêtes d'un utilisateur arrivent vers des instances différentes du cluster?

## > Solution:

- Le load balancer doit "**reconnaître**" des requêtes qui appartiennent à la même session.
- Toutes ces requêtes sont systématiquement envoyées à la même instance.



## Affinité de session (sticky sessions) (2)

> **Question:**

- Comment le load balancer peut-il reconnaître les requêtes d'une même session?

> **Solution 1:**

- En utilisant un cookie spécial

> **Solution 2:**

- En réécrivant l'URL et en ajoutant un ID de session

> **Solution 3:**

- Certains produits ont des méthodes plus sophistiquées.



# *Sécurité du service HTTP*



# Authentification et autorisation

## > Objectif

- On veut **protéger l'accès à certaines ressources**.
- On doit donc pouvoir **identifier et authentifier** l'utilisateur qui envoie une requête HTTP vers un ressource.
- On doit également pouvoir spécifier des **règles d'accès** pour les ressources gérées par le serveur HTTP.

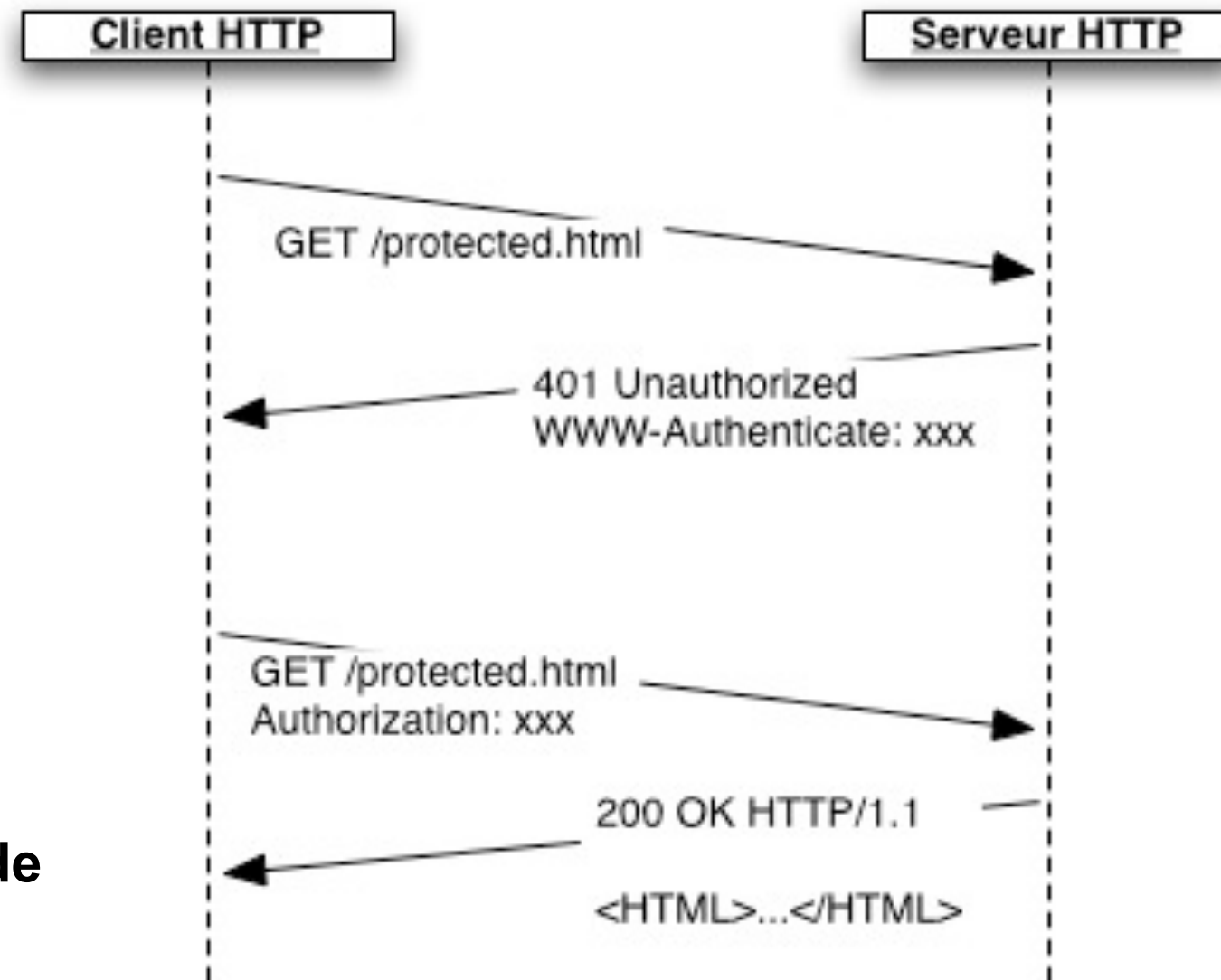
## > Questions:

- Que fournit la spécification (protocole)?
- Que fournit l'implémentation (serveur particulier)?



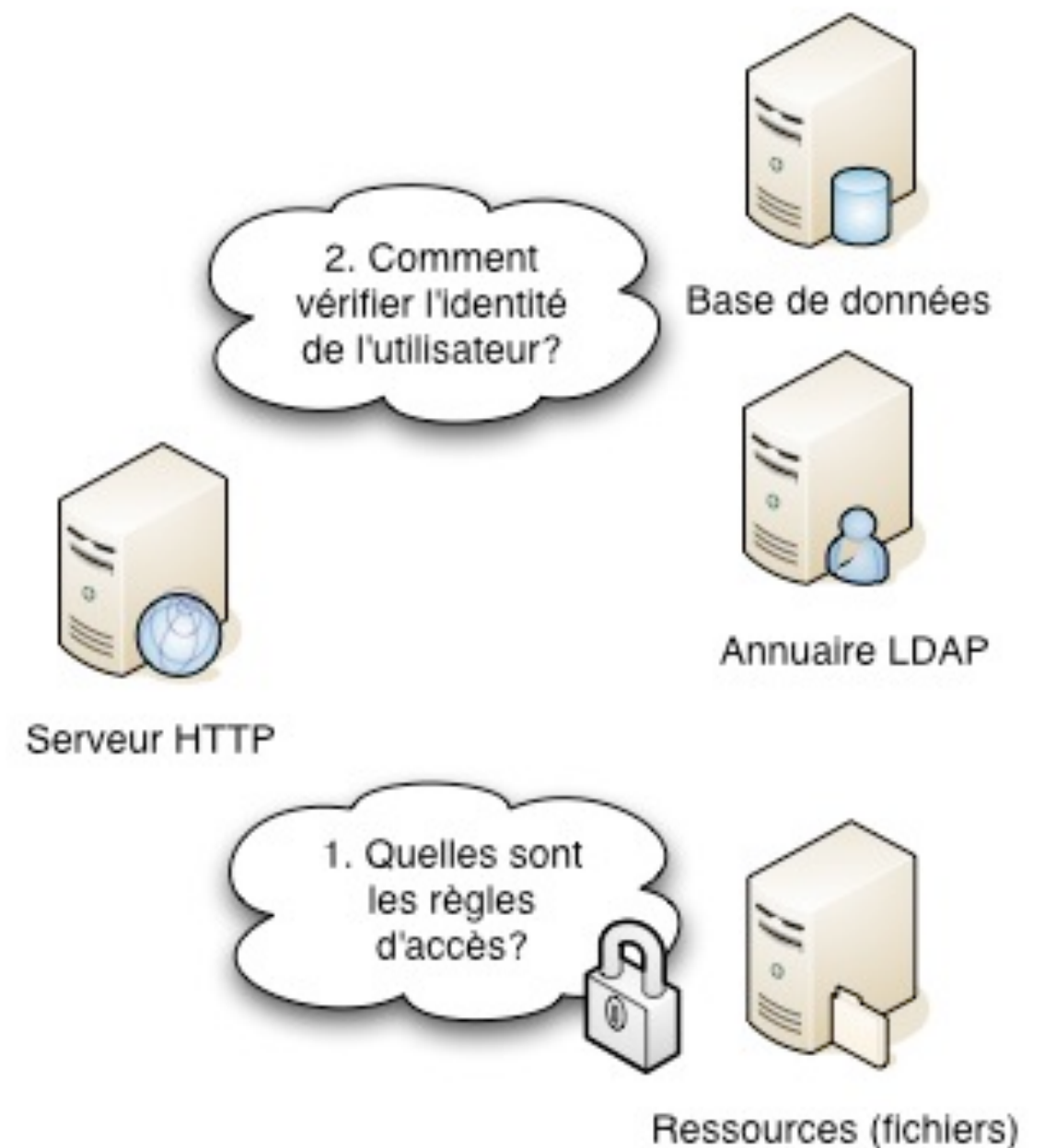
# Contrôle d'accès: que fournit le protocole?

- > **Un code de status: "401 Unauthorized"**
  - pour indiquer que l'accès non-authentifié est interdit.
- > **Une entête: "WWW-Authenticate"**
  - pour envoyer un challenge à l'agent.
- > **Une entête: "Authorization"**
  - pour envoyer la réponse (credentials) au serveur.
- > **Le RFC 2617 spécifie les méthodes d'authentification:**
  - Basic Scheme (mots de passe en clair)
  - Digest Scheme (fonction de hachage)
- > **Il est possible de définir d'autres méthode d'authentification:**
  - Exemple: Amazon Web Services (<http://docs.amazonwebservices.com/AmazonS3/latest/dev/RESTAuthentication.html>)



# Contrôle d'accès: que fournit l'implémentation?

- > **Une méthode pour spécifier quelles ressources doivent être protégées**
  - Via des fichiers de configuration
  - Via une interface graphique
  - etc.
- > **Une méthode pour indiquer comment et où les utilisateurs sont**
  - Dans un fichier
  - Dans une base de données
  - Dans un annuaire LDAP
  - etc.





# Contrôle d'accès: l'exemple du serveur apache

## > How-To

- <http://httpd.apache.org/docs/2.0/howto/auth.html>

## > Définition des droits d'accès

- Les règles peuvent être définies dans le fichier de configuration principal, dans une section `<Directory>`.
- Elles peuvent également être définies séparément, dans un fichier `.htaccess` placé directement dans un répertoire.
- Une règle spécifie quels utilisateurs et/ou quels groupes d'utilisateurs sont autorisés à accéder aux ressources en question (mot clé `Require`).

## > Définition des utilisateurs et des groupes

- Apache permet l'utilisation de fichiers textes, avec la commande `htpasswd`.
- Apache permet aussi d'utiliser un référentiel utilisateurs existant. Par exemple un annuaire LDAP avec le module `mod_auth_ldap`.
- [http://httpd.apache.org/docs/2.0/mod/mod\\_auth\\_ldap.html](http://httpd.apache.org/docs/2.0/mod/mod_auth_ldap.html)



# Contrôle d'accès: l'exemple du serveur apache

```
AuthType Digest
AuthName "By Invitation Only"
AuthUserFile /usr/local/apache/passwd/passwords
AuthGroupFile /usr/local/apache/passwd/groups
Require group GroupName
```

```
# Autoriser l'accès à un utilisateur particulier
Require user userId

# Autoriser l'accès à tous les utilisateurs authentifiés
Require valid-user

# Autoriser l'accès aux membres d'un groupe
Require group GroupName
```

```
# Coordonnées du service LDAP
AuthLDAPURL ldap://ldap.airius.com/o=Airius?uid

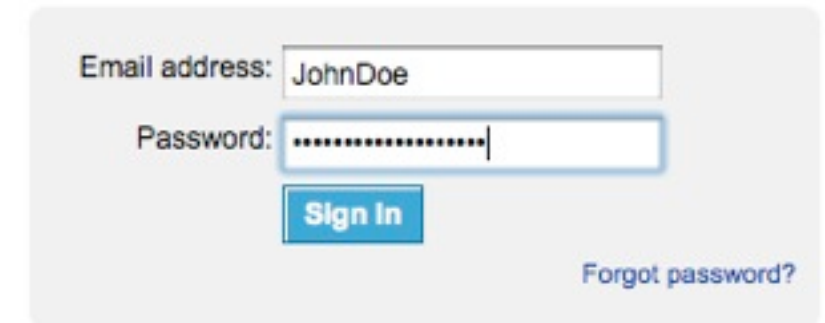
# Règle d'accès
Require group cn=Administrators, o=Airius
```

# Confidentialité des échanges

- > **Dans la version 1.0, les échanges HTTP n'étaient pas sécurisés**
  - Mots de passe envoyés en clair ("basic digest")
  - Contenu envoyé en clair
- > **Pour répondre à la demande de nouvelles applications (commerce):**
  - Netscape a développé une solution et publié des spécifications ("The SSL Protocol" en 1995, "The SSL 3.0 Protocol" en 1996).
  - Le protocole SSL est la base du protocole TLS, décrit dans le RFC 5246.
- > **SSL et TLS permettent:**
  - A un client et à un serveur de s'authentifier mutuellement, au moyen de certificats (cryptographie asymétrique)
  - De négocier les modalités d'une session sécurisée (version, algorithmes)
  - D'échanger une clé de session (cryptographie symétrique)
  - D'échanger des messages HTTP au travers d'un canal sécurisé.

# Gestion de la sécurité au niveau applicatif

- > Beaucoup d'applications web n'utilisent pas l'authentification HTTP.
- > L'authentification est gérée par l'application elle-même:
  - L'utilisateur accède à un formulaire
  - Il est envoyé au serveur (HTTPS)
  - L'application procède à l'authentification (flexibilité dans le choix de la méthode!)
- > On a une notion de "session authentifiée":
  - Avec la possibilité de faire expirer la session avec flexibilité
  - Avec la possibilité d'utiliser des rôles

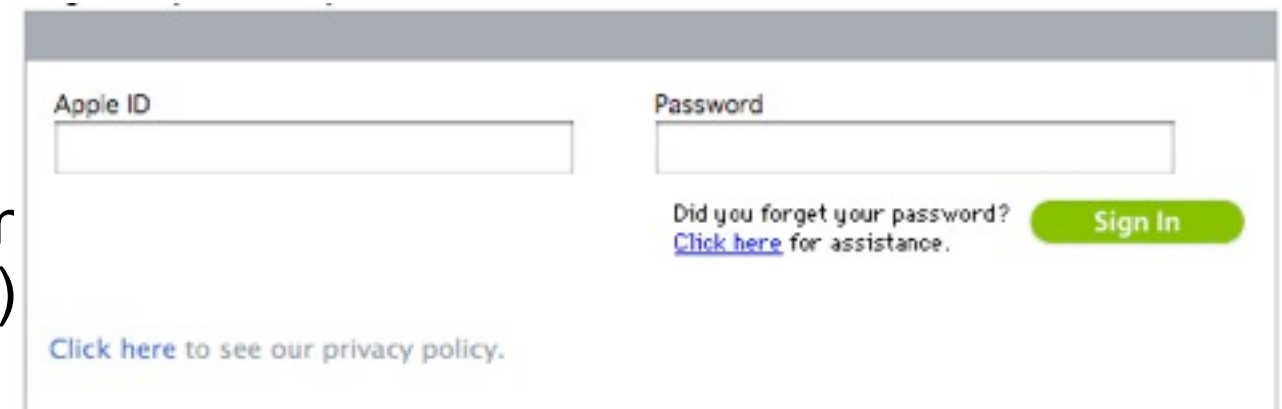


Email address:

Password:

[Sign In](#)

[Forgot password?](#)



Apple ID

Password

Did you forget your password? [Click here](#) for assistance.

[Sign In](#)

[Click here to see our privacy policy.](#)



Enter the agreement number

To verify your authorization, please enter the following information:

Agreement number:

[Continue](#)

If you have any problems with login, please contact the [Online help](#).

# *Conclusions*



# Conclusions

- > **Rédiger un cahier des charges, c'est:**
  - Spécifier des exigences fonctionnelles (qu'est-ce que fait le système?)
  - Spécifier des exigences non-fonctionnelles (comment le fait-il?)
- > **Les exigences non-fonctionnelles sont des "qualités systémiques"**
  - En anglais, on parle des "-ilities"
  - Il en existe beaucoup et elles sont parfois contradictoires
  - Concevoir l'architecture, c'est trouver le bon équilibre
- > **Satisfaire aux exigences non-fonctionnelles demande**
  - Un travail à travers les tiers (depuis le client jusqu'aux données)
  - Un travail à travers les couches (depuis le matériel jusqu'à l'application)
  - Une collaboration entre "Le Développement" et "L'Infrastructure"

# Conclusions

- > **Dans le cas particulier d'un service HTTP, il convient:**
  - d'exploiter les mécanismes prévus dans le protocole.
  - de définir l'architecture de déploiement globale (topologie, composants réseaux, composants logiciels).
  - d'optimiser la configuration des composants individuels, et notamment celle du serveur HTTP (p.ex. apache).
- > **Les notions de Proxy et de Reverse Proxy permettent de traiter plusieurs qualités systémiques:**
  - Performance (utilisation de caches, répartition de charge)
  - Scalabilité (utilisation de caches, répartition de charge)
  - Disponibilité (répartition de charge)
  - Sécurité (contrôle d'accès, audit, accès au travers d'une DMZ)