# Introduction au system design

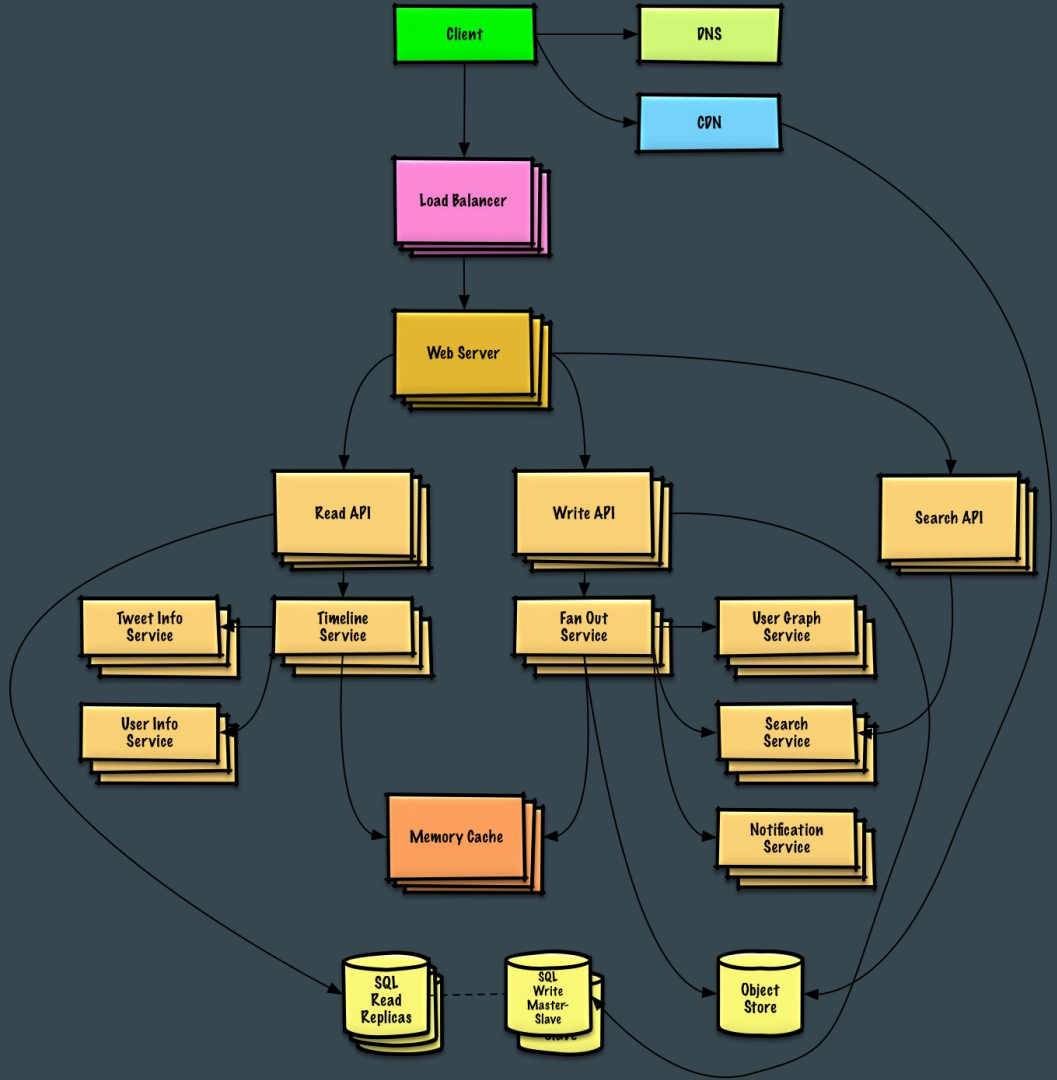
...

# System Design

Le system design permet d'architecturer une application.

On utilise tous les composants d'ingénierie connus

Anticiper les problématiques à haut niveau


Client
DNS
CDN
Load Balancer
Web Server
Read API
Write API
Search API
Tweet Info Service
Timeline Service
Fan Out Service
User Graph Service
User Info Service
Search Service
Memory Cache
Notification Service
SQL Read Replicas
SQL Write Master-Slave
Object Store

# System Design : Synchrone vs Asynchrone

Qu’est-ce qu’un appel synchrone ?

Qu’est-ce qu’un appel asynchrone ?

# System Design : Synchrone vs Asynchrone

Synchrone : Le client attend la réponse du serveur.

Asynchrone : Le client n'attend pas de réponse du serveur. Il peut être notifié plus tard par un système de callback.

Avantages et inconvénients des deux ?

# System Design : Synchrone vs Asynchrone

Synchrone :

- La sûreté / facilité / le client attend la réponse
- Lenteur

Asynchrone :

- Rapidité / Le client n’attend pas de réponse
- Plus complexe à mettre en place et à debug

# System Design : Performance vs scalability

Performance versus scalabilité

Qu’est-ce que signifie un problème de performance ?

Qu’est-ce que signifie un problème de scalabilité ?

# System Design : Performance vs scalability

Un service est scalable si sa performance croît avec les ressources ajoutées.

Problème de performance : Le système est lent pour un utilisateur

Problème de scalabilité : Le système est rapide pour un utilisateur, mais lent lors de pics de charge

# System Design : Latency vs throughput

Latence vs débit

Qu’est-ce que la latence ?

Qu’est-ce que le débit ?

# System Design : Latency vs throughput

La latence est le temps pour réaliser une action ou produire un résultat

Le débit est le nombre d'actions réalisées en un certain temps

Généralement, on vise un débit maximal pour une latence acceptable.

# System Design : CAP theorem

Le théorème de CAP (Consistency Availability Partitioning) énonce qu'il faut choisir 2 solutions parmi :

- Consistency : Toutes les lectures reçoivent la dernière écriture ou une erreur
- Availability : Toutes requêtes reçoit une réponse, même si elle n'est pas la dernière version
- Partition Tolerance : Le système continue de fonctionner même s'il y a des pannes réseaux

# System Design : CAP theorem

Quelle(s) combinaison(s) choisir ? CA / AP / CP

# System Design : CAP theorem

Puisque le réseau n’est pas fiable, on doit toujours choisir AP ou CP.

CP : Attendre une réponse peut amener à un timeout. Le système n’est plus disponible (surchargé ou mort)

AP : Retourne la dernière version visible, même si elle n’est pas à jour. Les écritures peuvent être lentes en cas de reprise de panne.

# System Design : Consistency

Il existe 3 types de consistency :

- Weak

- Eventual

- Strong

# System Design : Weak Consistency

Après une écriture, la lecture est visible ou non. C'est une approche best-effort.

Où peut-on la retrouver ?

# System Design : Eventual Consistency

Après une écriture, la lecture sera rapidement visible (quelques ms).

Données répliquées de manière ?

Où peut-on la retrouver ?

# System Design : Strong Consistency

Après une écriture, la lecture est visible immédiatement.

Données répliquées de manière ?

Où peut-on la retrouver ?

# Points d’entrée

…

# System Design : DNS

Qu’est-ce qu’un DNS ?

# System Design : DNS

Un serveur hiérarchique qui transcrit un nom de domaine en adresse IP.

Il existe des services hébergés : Route 53 / CloudFlare ...

Renvoie l'IP selon différentes stratégies

# System Design : DNS

Weighted Round-robin

- A/B testing
- Eviter les serveurs en maintenance

Latency

Geographical

Inconvénients du DNS ?

# System Design : CDN

Qu’est-ce qu’un CDN ?

# System Design : CDN

Un CDN est un réseau de serveur proxy distribué géographiquement pour servir du contenu rapidement (généralement statique)

Push CDN : Vous envoyez les infos à cache au CDN.

Pull CDN : Lors de la 1ère requête, on met en cache le résultat

Inconvénients du CDN ?

# System Design : CDN

Inconvénients :

- Le résultat en cache peut ne pas être à jour si le TTL est trop élevé.

- Force à modifier les URL du contenu statique pour pointer sur le CDN

# System Design : Load balancer

Qu’est-ce qu’un load balancer ?

# System Design : Load balancer

Un load balancer distribue le trafic client vers différents serveurs.

Avantages :

- Empêcher les requêtes d'aller sur des serveurs 'unhealthy'
- Empêcher un serveur d'être surchargé
- Eliminer les Single Point of Failure
- Améliorer le scaling horizontal
- SSL termination

Quels sont les inconvénients d'un load balancer ?

# System Design : Load balancer

Si le load balancer n’arrive pas à gérer la charge de redirection

Il devient un SPOF

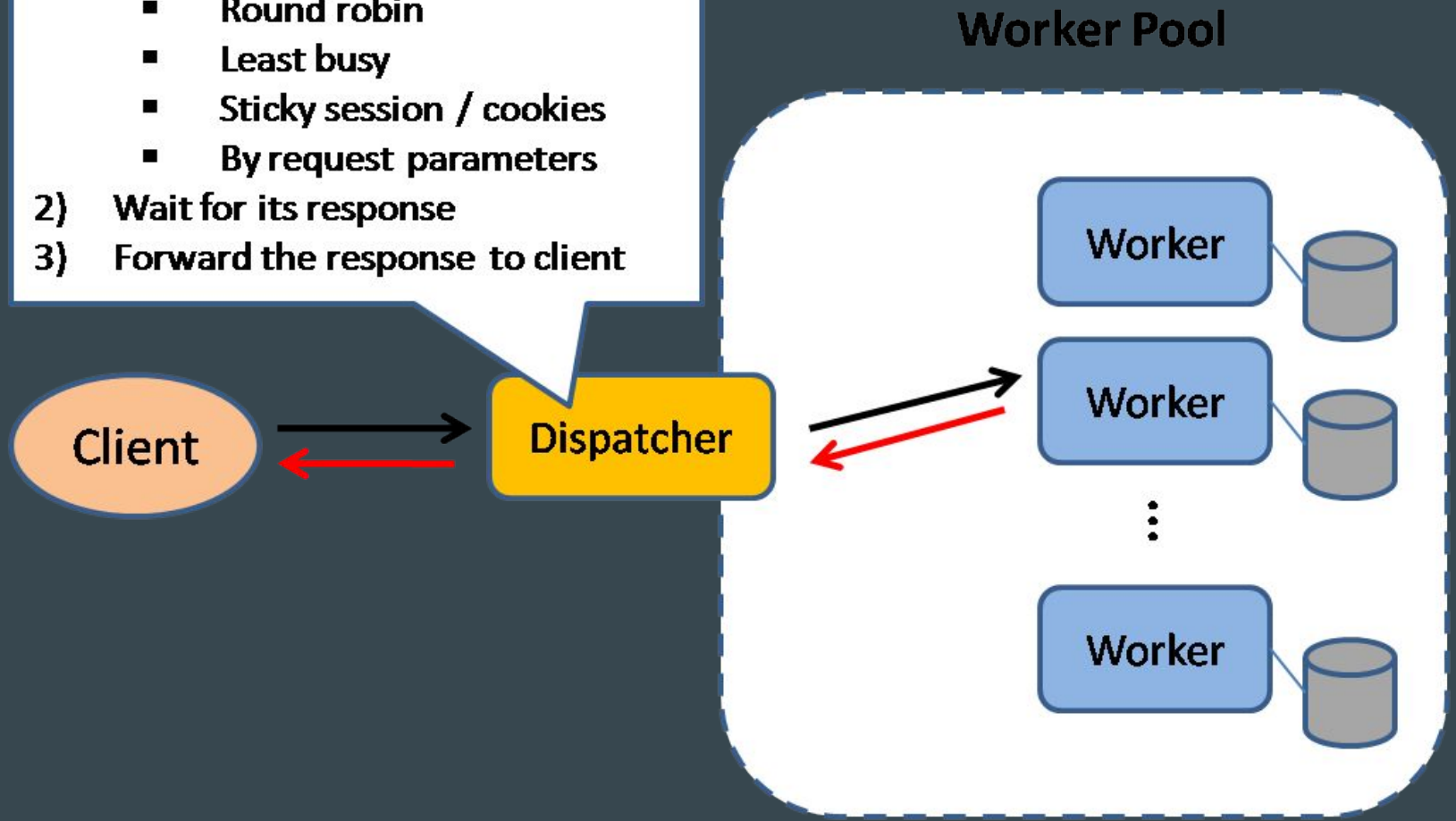
# System Design : Load balancer

Quelles stratégies appliquer pour rediriger le trafic ?

# System Design : Load balancer

- Random
- Moins chargé
- Session / Cookies
- Round robin / Weighted round-robin
- Layer 4 - TCP / UDP
- Layer 7 - Application (header, message, cookie, ...)

1) Pick a worker to forward request
   - Random
   - Round robin
   - Least busy
   - Sticky session / cookies
   - By request parameters
2) Wait for its response
3) Forward the response to client



# System Design : Reverse proxy

Un reverse proxy peut être le point d'entrée d'une application web

Les avantages sont :

- Sécurité
- Compression
- Cache
- Static file
- SSL termination

Quels sont les inconvénients ?

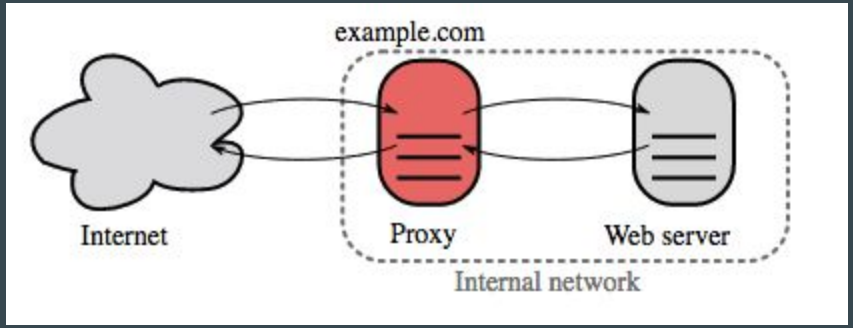
# System Design : Reverse proxy

reverse proxy vs load balancer ?

# System Design : Reverse proxy

On utilise un load balancer lorsque l'on a plusieurs serveurs à servir

Beaucoup de solutions (nginx, HAProxy) font les deux


example.com
Internet
Proxy
Web server
Internal network

# Bases de données SQL

...

# System Design : Bases de données SQL

Une base de données SQL permet le stockage et la récupération de données organisées.

La donnée est stockée dans des tables avec un schéma fermé.

Les requêtes sont effectuées sous forme de transactions

Utilisé dans toutes les industries depuis des dizaines d'années

# System Design : Bases de données SQL

A quoi correspond l’acronyme ACID ?

# System Design : Bases de données SQL

Atomicité : Une transaction est atomique (tout ou rien)

Cohérence : Une transaction laisse la base de données dans un état cohérent

Isolation : Exécuter des transactions en parallèle a le même résultat qu'en série

Durabilité : Lorsqu'une transaction est commit, elle le reste.

# System Design : Bases de données SQL

Il existe de multiples techniques pour scale une DB :

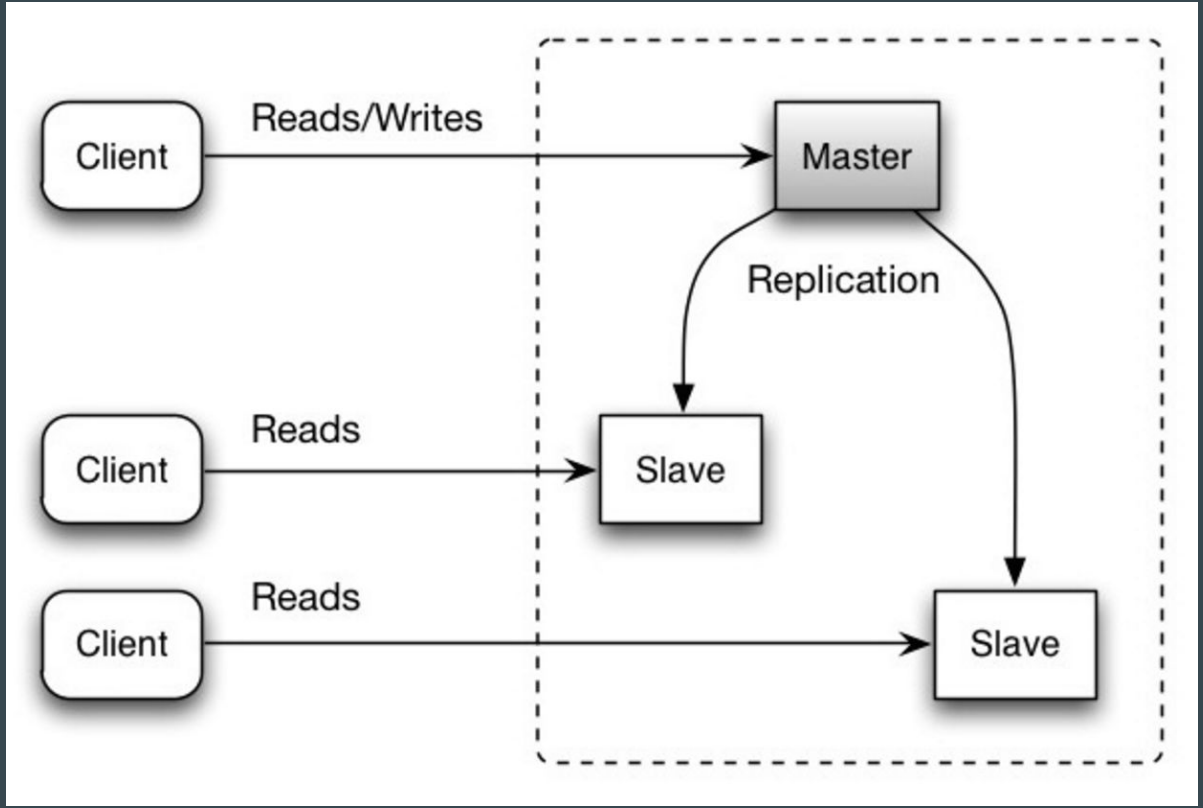
- master-slave replication
- master-master replication
- federation
- sharding
- denormalization

# System Design : Master-slave replication

Le master exécute les read/write.

Les writes sont répliquées sur les slaves.

Les slaves exécutent les reads.

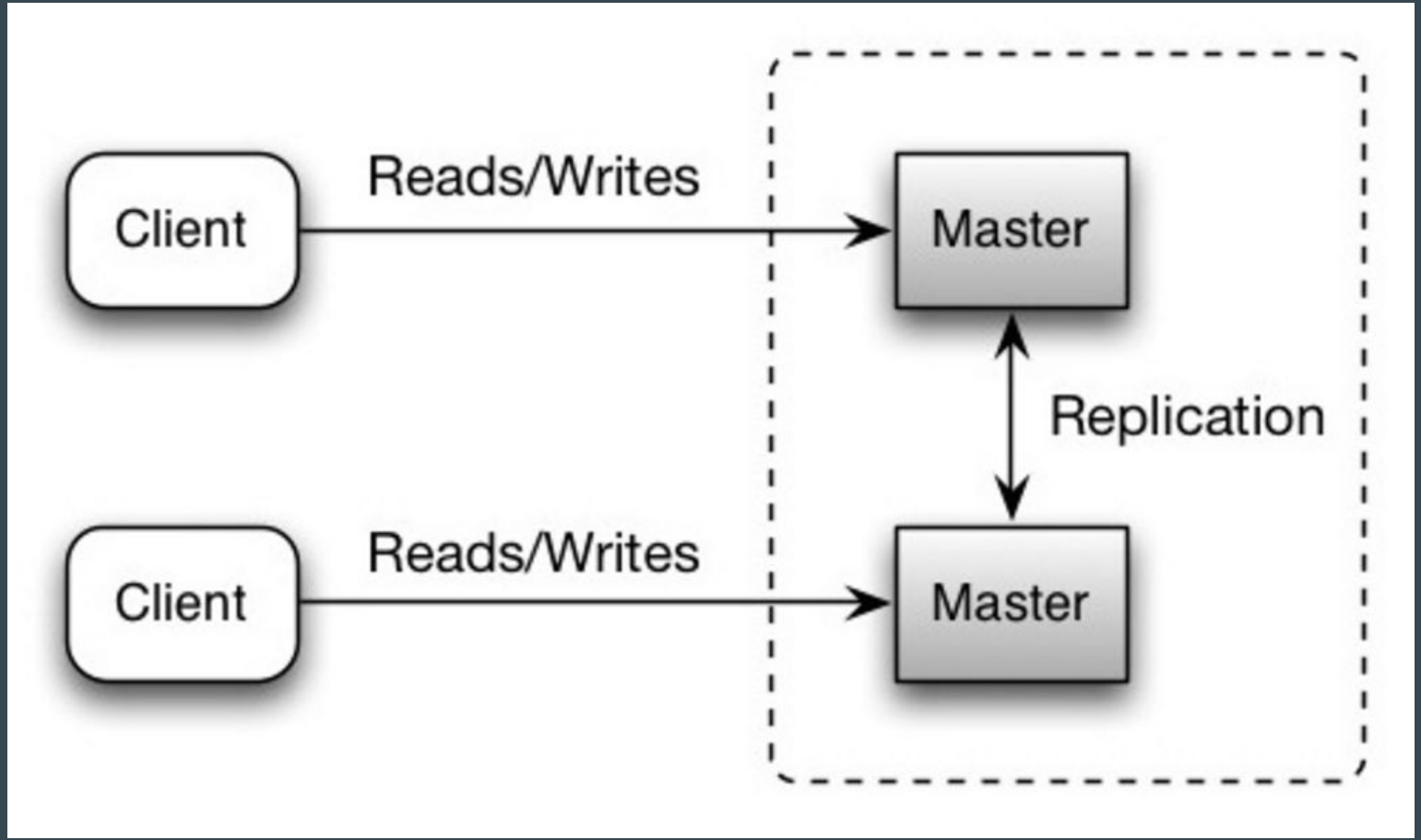

Client
Reads/Writes
Master
Replication
Client
Reads
Slave
Client
Reads
Slave

# System Design : Master-master replication

Les master exécutent les read/write.

Les writes sont répliquées sur les masters.

On appelle aussi cela la haute disponibilité (HA)


Client
Reads/Writes
Master
Replication
Client
Reads/Writes
Master

# System Design : Replication

Quels sont les inconvénients de la réplication ?

# System Design : Replication

- Perte de donnée si le master tombe avant de répliquer la data
- S'il y a beaucoup d'écritures, les slaves vont devoir recommencer les lectures
- Plus il y a de read slave, plus il faut répliquer, plus la latence est importante
- La réplication nécessite plus de hardware

# System Design : Replication

Quels sont les inconvénients de la réplication master-slave ?

# System Design : Replication

Quels sont les inconvénients de la réplication master-slave ?

- Que faire si le master tombe ?

# System Design : Replication

Quels sont les inconvénients de la réplication master-master ?

# System Design : Replication

Quels sont les inconvénients de la réplication master-master ?

- Nécessite un load balancer (ou du code de réplication)

- De plus en plus de conflits à mesure que l'on augmente les masters

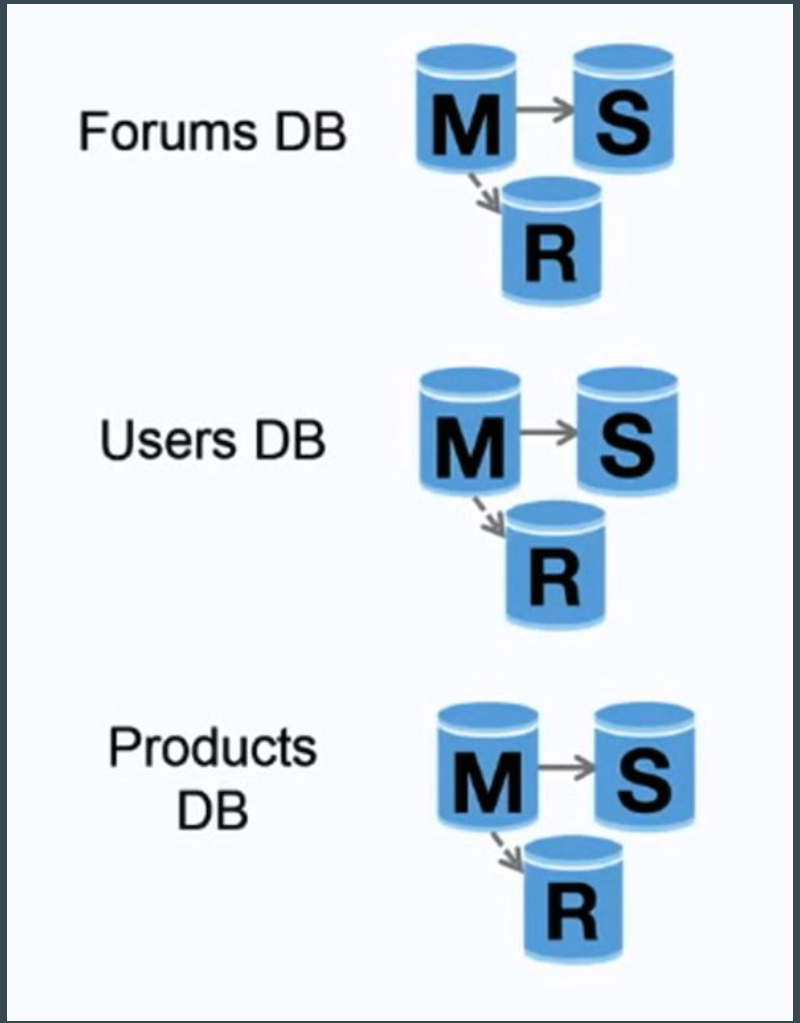
- Perte de consistance ou augmentation de la latence

# System Design : Federation

La fédération découpe les bases de données par fonctionnalités.

Les grosses DB ont moins de trafic et gagnent en performance

Les petites DB peuvent tenir en mémoire et gagnent donc en performance


Forums DB
M
S
R
Users DB
M
S
R
Products DB
M
S
R

# System Design : Federation

Quels sont les inconvénients de la fédération ?

# System Design : Federation

- Plus de hardware / ressources

- Modification du code pour supporter le multi DB

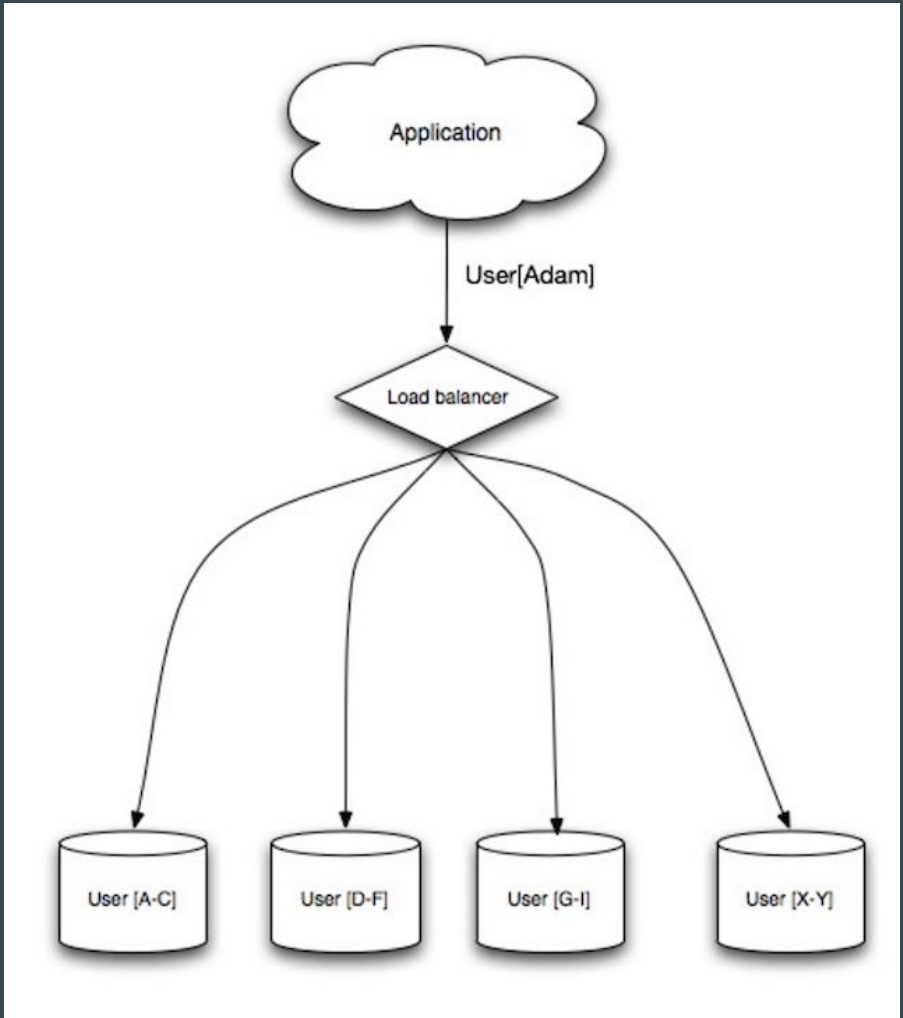
- Les jointures deviennent très compliquées

# System Design : Sharding

Le sharding distribue la donnée dans différentes DB.

On split les lignes grâce à une clé

Moins de lecture, moins d'écriture, index plus petit


Application
User[Adam]
Load balancer
User [A-C]
User [D-F]
User [G-I]
User [X-Y]

# System Design : Sharding

Quels sont les inconvénients du sharding ?

# System Design : Sharding

- Plus de hardware / ressources

- Modification du code pour supporter le multi DB

- Les jointures deviennent très compliquées

- La distribution de la donnée peut nécessiter une redistribution

# System Design : Denormalization

La denormalization vise à augmenter les performances de lecture en perdant en écriture.

Les données sont stockées dans une seule table pour éviter les jointures

Il existe des solutions comme les vues matérialisées pour palier à la redondance

# System Design : Denormalization

Quels sont les inconvénients de la denormalization ?

# System Design : Denormalization

- La donnée est dupliquée

- Si la charge en écriture est élevée, les performances s'effondrent

# Cache

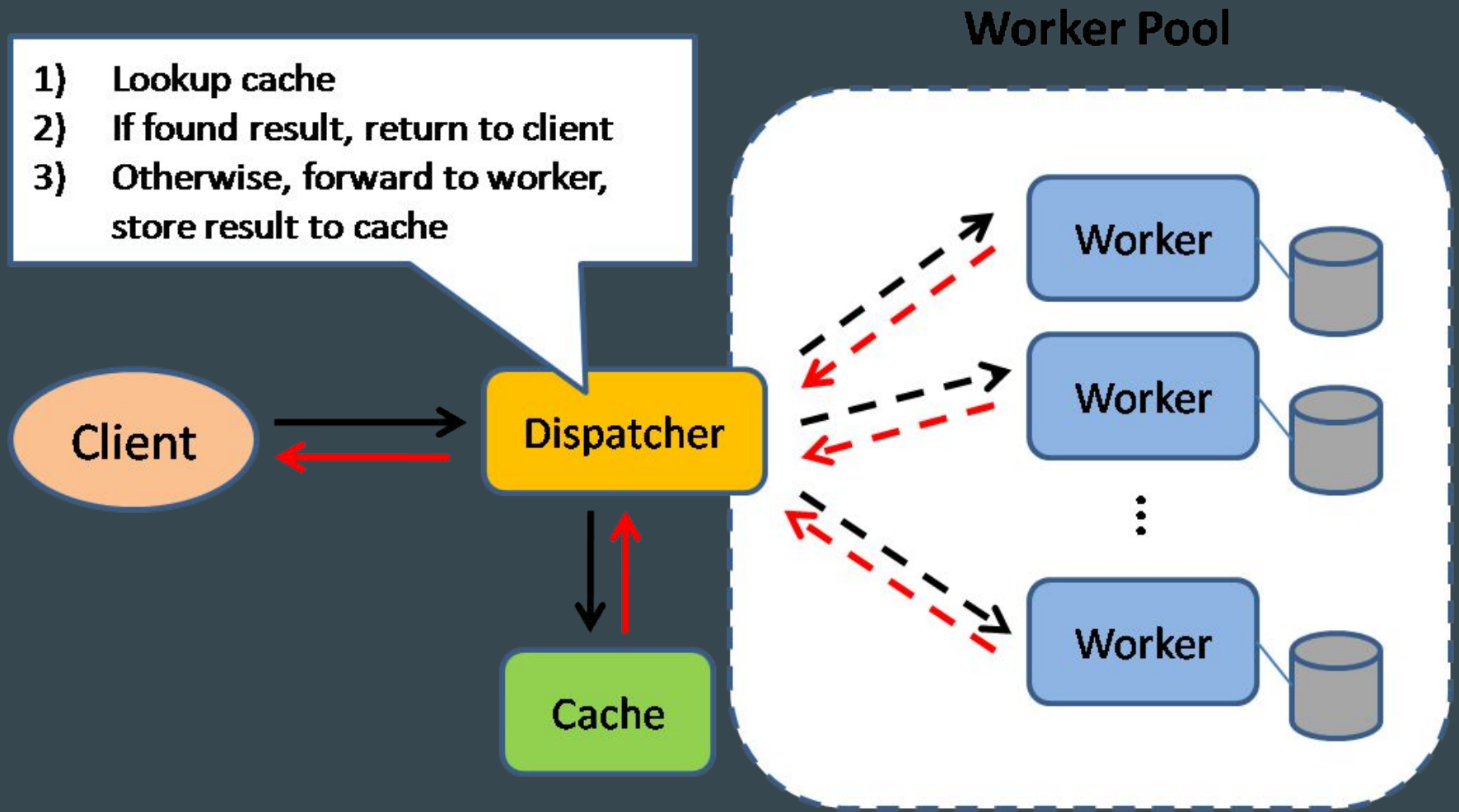
...

# System Design : Cache

Une couche de stockage à grande vitesse

Stocke de la donnée volatile

Le cache est chargé lors de lecture, et vider lors d’écriture


Worker Pool
1) Lookup cache
2) If found result, return to client
3) Otherwise, forward to worker, store result to cache
Client
Dispatcher
Cache
Worker
Worker
Worker

# System Design : Cache

Où avons-nous des caches ?

# System Design : Cache

Côté client (OS, browser, ...)

CDN

Web server (reverse proxy, @Cache de RESTEasy)

Database

Application (map, Memcache, Redis ...)

# System Design : Cache

Il existe plusieurs stratégies pour le cache :

- Cache-aside

- Write-through

- Write-behind

- et d’autres (refresh-ahead ...)

# System Design : Cache-Aside

Je regarde si l'entrée existe dans le cache, si oui je la renvoie, sinon :

- Je charge la donnée depuis la DB
- Je la met dans le cache
- Je renvoie le résultat

Quels sont les inconvénients ?

# System Design : Cache-Aside

Tous les miss forcent à faire 3 requêtes (GET cache, GET DB, PUT cache)

Si la donnée est mise à jour dans la DB, il faut :

- Vider le cache
- Write-through
- Mettre un TTL si les lignes

# System Design : Write-through

On ne passe que par le cache pour les lectures / écritures.

Le cache fait les appels à la DB s'il ne trouve pas la données ou doit flush

Les appels sont synchrones

Quels sont les inconvénients ?

# System Design : Write-through

- La majorité de la data ne sera jamais lue

- Lenteur (tout va dans le cache)

# System Design : Write-behind

Le même principe que le Write-through, mais l'écriture est asynchrone

Améliore les performances d'écriture par rapport au write-through

Quels sont les inconvénients ?

# System Design : Write-behind

- Plus compliqué à mettre en place que write-through / cache-aside

- Perte de donnée si le cache tombe avant la propagation de l'écriture

# System Design : Cache

Quels sont les inconvénients du cache ?

# System Design : Cache

- Maintenir la cohérence entre le cache et la DB

- L'invalidation du cache est un problème difficile

- Nécessite des changements de code, ajouté Redis ou Memcached)

# Interview

...

# Interview

Un client demande : Je veux faire le concurrent de WhatsApp. Comment allez-vous faire ?

# Interview

1°) Clarifier le sujet et se mettre d'accord sur la portée du projet

- Cas d'utilisation : qui va l'utiliser ? comment ?

- Contraintes : Identifier le trafic, le scaling ...

# Interview

2°) Réaliser un design architecture abstrait / haut-niveau

- On présente les composants importants :
    - La couche application qui reçoit les requêtes
    - Lister les différents services requis
    - La couche de BD
    - Sur un système scalable, on va préciser le load balancer, le type de DB ...

# Interview

3°) Design des composants (si demandé)

Description de chaque composant et API

Design OO pour les features

Design de la base de données

# Interview

4°) Comprendre les bottlenecks

Peut-être que le système requiert un LB et plein de serveurs derrière ?

Peut-être que la data est très importante et il faut la distribuer ?

Est-ce que la BD va être lente et il faut un cache ?

Être capable d'expliquer les avantages et inconvénients

# Interview

5°) Scale son design plus bas-niveau

Préciser le type de scaling (vertical / horizontal)

Caching

Load balancing

Database replication

Map-Reduce

Microservices

# System Design : Conclusion

Pas de conclusion, c'était assez costaud comme ça !