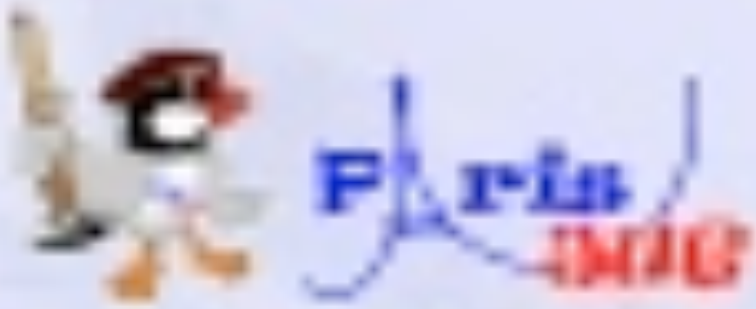
Paris JUG

12/05/2009

www.parisjug.org

www.parisjug.org



12/05/2009

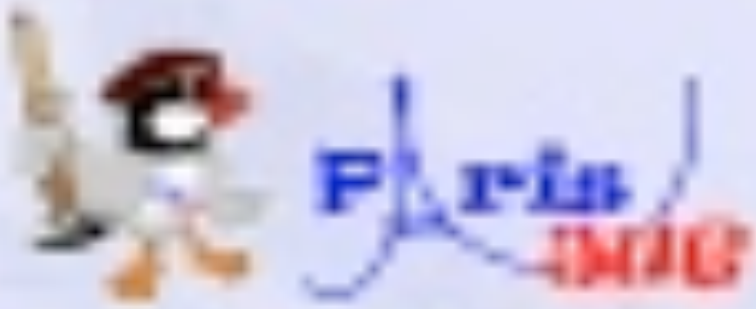
Data grid

Du cache distribué aux grilles de données

Cyrille Le Clerc
Xebia

Xebia

www.parisjug.org



12/05/2009

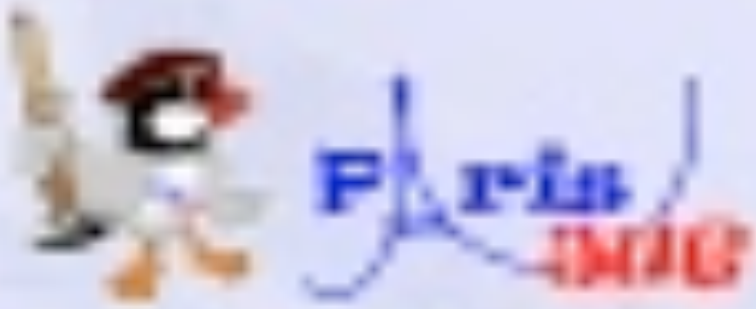
Data grid

Du cache distribué aux grilles de données

Erwan Alliaume
Xebia

Xebia

www.parisjug.org



12/05/2009

Data grid

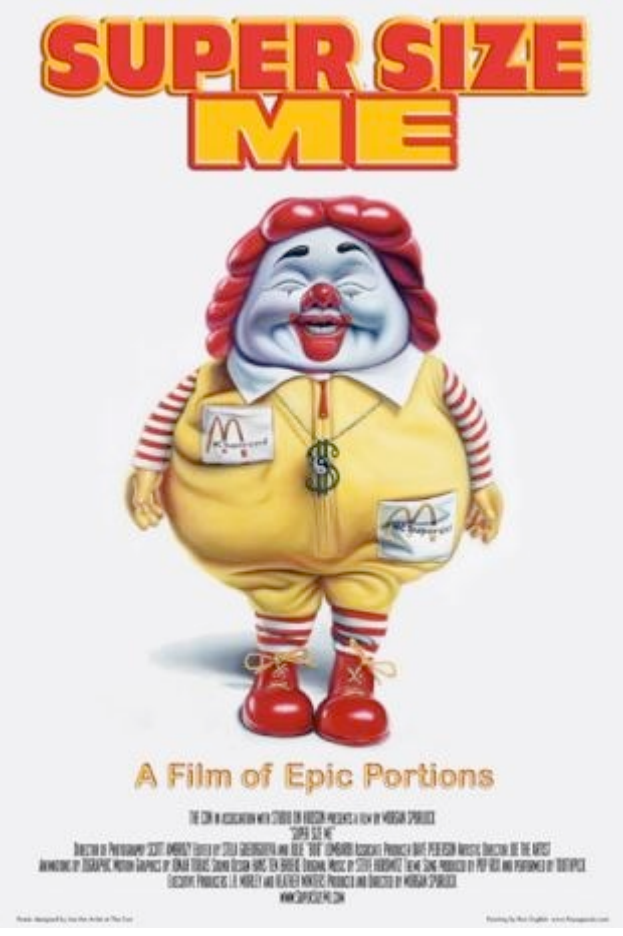
Du cache distribué aux grilles de données

Jean-Michel Bea
Fast Connect

www.parisjug.org

« Du cache distribué

aux grilles de données »

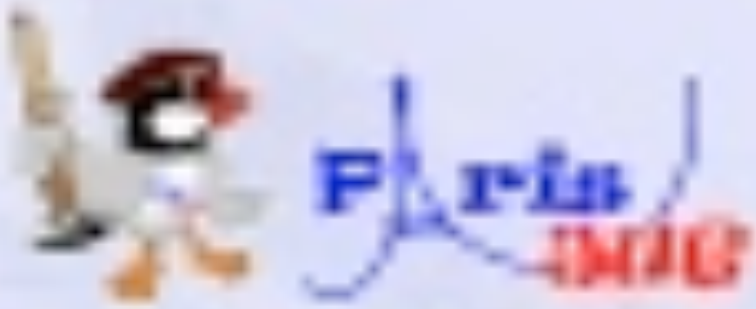


Intervenants

- Cyrille Le Clerc (Xebia)
- Erwan Alliaume (Xebia)
- Jean-Michel Bea (Fast Connect)

Sommaire

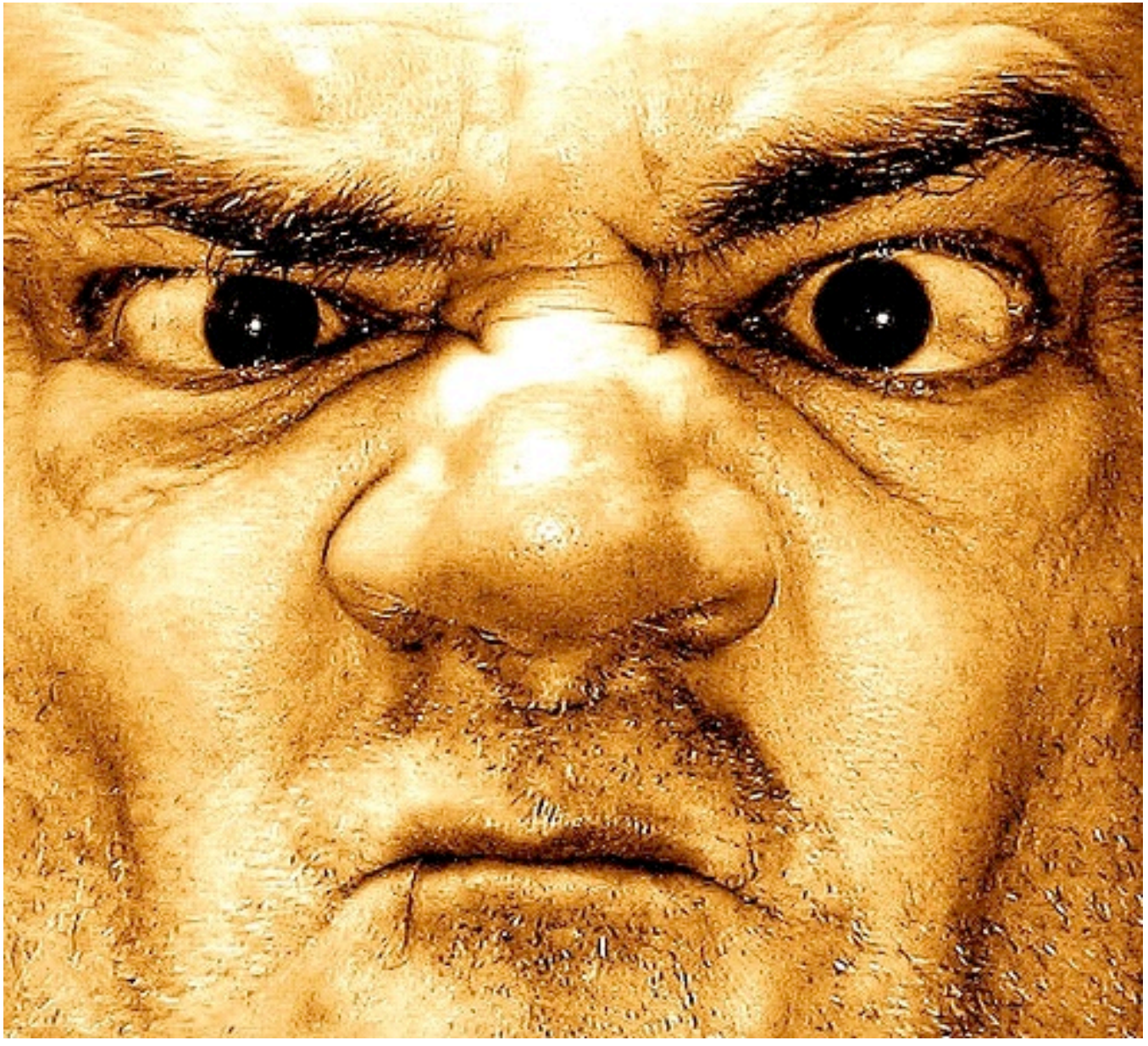
- Caches distribués
- Network Attached Memory
- Grilles de données
- Data Grid, Cloud et les autres …



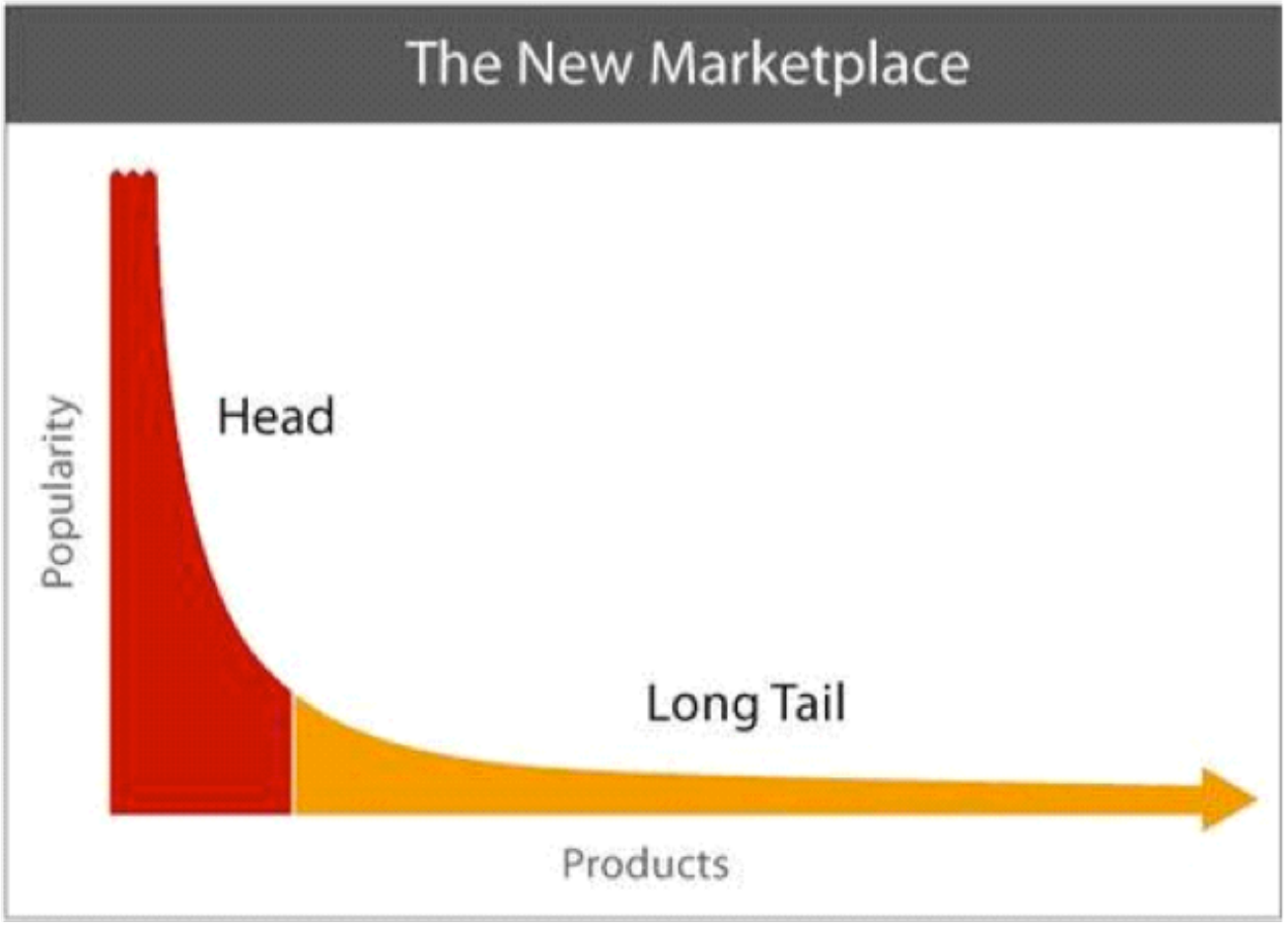
Cache distribué

www.parisjug.org

Votre DBA !



Votre manager



- **Small number of items may make up the bulk of sales**

Chris Anderson – Wired Magazine

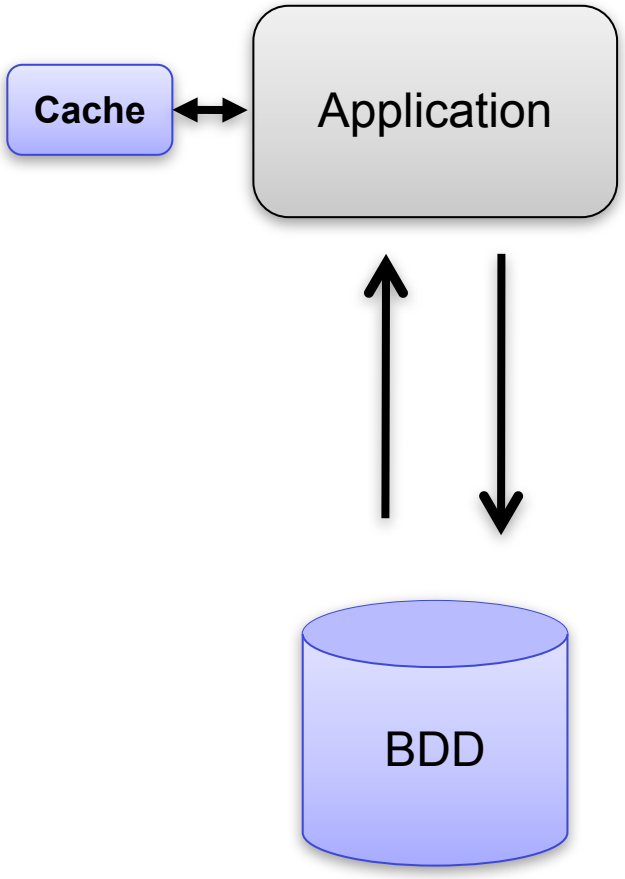
Votre application

Une solution ?

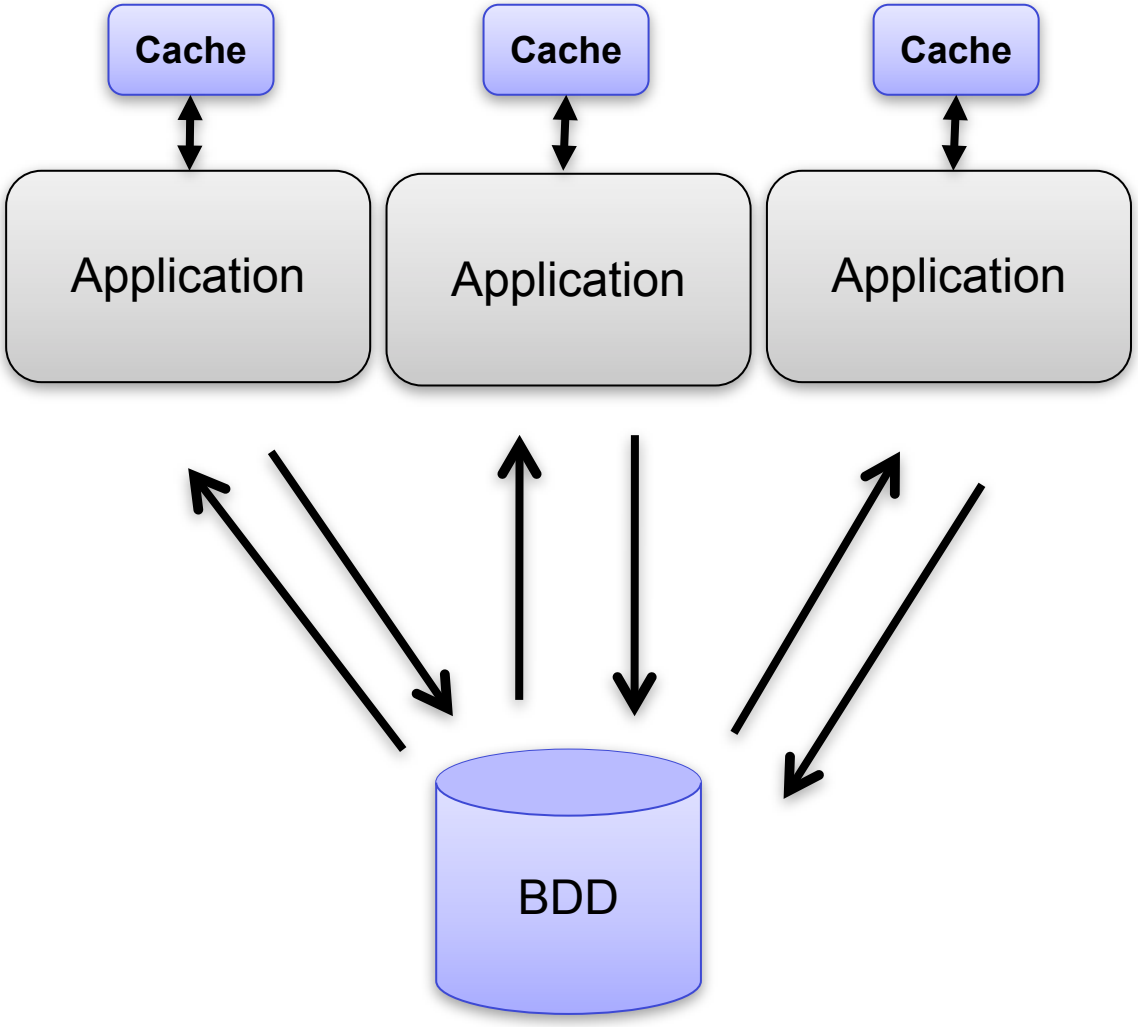
J'ai une idée !

Distribuons et cachons notre application…

Architecture existante



Architecture cible



Exemples de produits

Ehcache

- Solution open source la plus populaire
- Configuration XML (ehcache.xml)
- Très léger
- Singleton / multi instances
- JSR-107 JCache implementation
- …
- Memory / Disk storage
- Ajout possible de listeners (Cache Manager / Cache Event)
- JMX
- …
- **Distributable**

Here we are !

Jboss Cache

- Cache Thread-Safe ‘in memory’
- Participation aux transaction JTA
- JSR-107 Jcache implementation

  RI pour le JDK SE 8 ou 9 :)

  …
- Memory / Disk / Cluster storage
- Listeners may be plugged in (Cache Manager / Cache Event)
- JMX
- …
- **Distribuable sur l’un ou plusieurs nœud d’un cluster**

Utilisation d'un cache

Manipulation directe

- cache.put(Element element)
- cache.get (Object key)

Pull through / Self populating

- cache.get (Object key)

Utilisation d'un cache

Manipulation directe

- cache.put(Element element)
- cache.get (Object key)

Pull through / Self populating

- cache.get (Object key)

Hibernate cache L2

- *Hibernate / sans cache : 192 secondes*
- *Hibernate / Ehcache L2 : 60 secondes*
- *Accès direct à Ehcache : 20 secondes*
- *Accès direct à Ehcache / session optim: 1 secondes*

=> L2 Ca ne scale pas ! get / put sur 1 seul objet => données à répliquer partout

Storage

Mémoire

- Toujours disponible / Thread safe
- Souvent basée sur une LinkedHashMap du JDK
- Peut contenir des ‘Object’
- Non persistent

Disque

- Optionnel, à configurer
- Doit contenir des ‘Serializable’

Attention à la sérialisation Java : un Byte[] est 20 fois plus rapide qu’un String

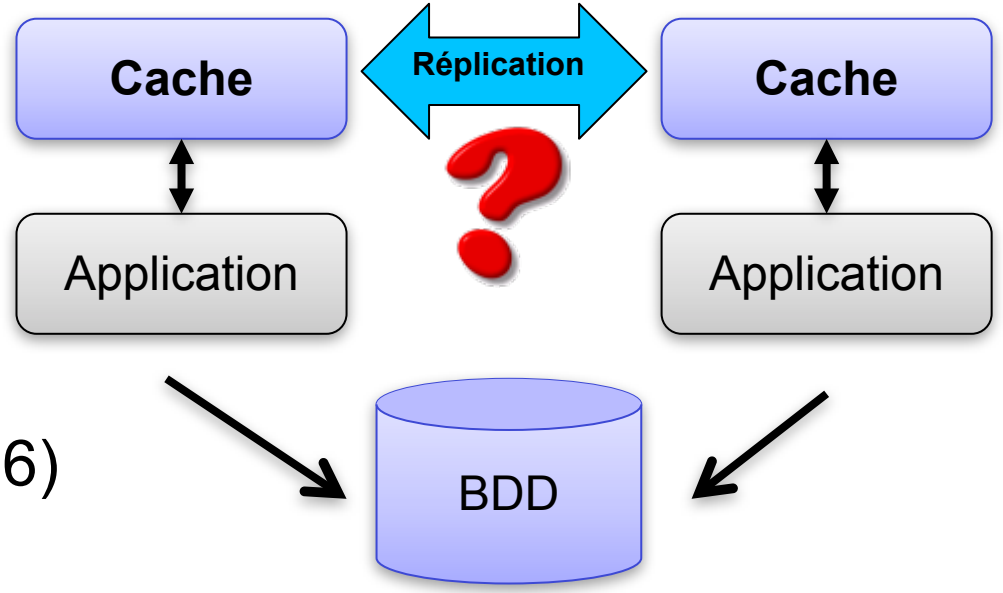
Jdbc

- Attention à ne pas perdre de vu le but du cache !

Cache distribué

Plusieurs mécanismes, même problématiques

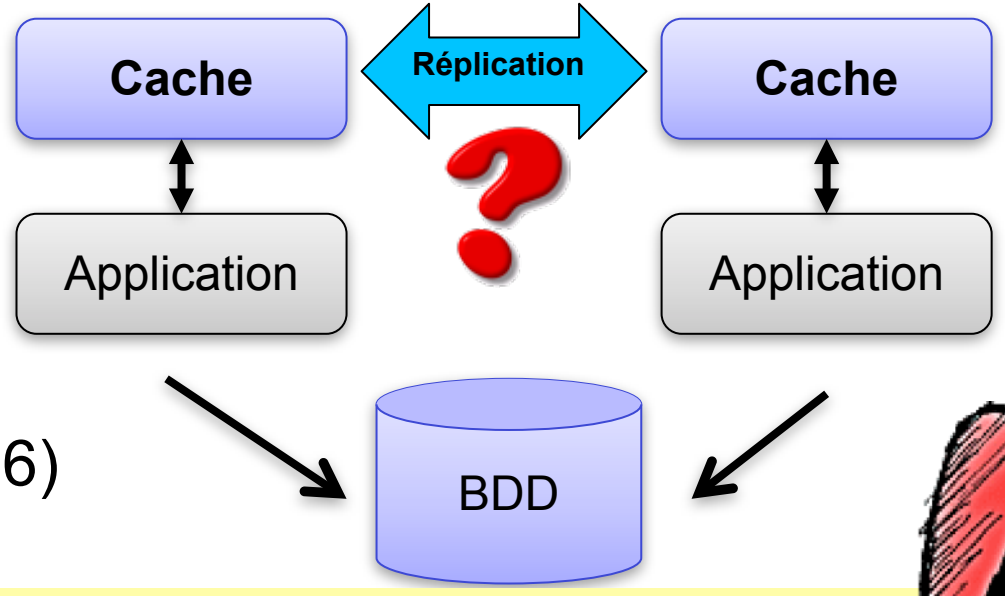
- RMI
- JGoups
- Cache Server
- Terracotta
- JMS replication (ehcache 1.6)



Cache distribué

Plusieurs mécanismes, même problématiques

- RMI
- JGoups
- Cache Server
- Terracotta
- JMS replication (ehcache 1.6)



Plusieurs sources de données, danger !

- Possibilité de lire des données incohérentes
  - Mises à jour concurrentes
  - Notifications, évictions, contraintes de fiabilité
- Même problème qu'une mémoire partagée
- Est-ce un problème ? Pas forcément !

Eviction

Suppression des éléments du cache

- Permet de sizer votre cache
- Time-based / Cache size / Heap size Based – LRU / LFU / FIFO / Custom
- Quelque fois régionalisable (dans Jboss Cache) (deprecated)
  - /myshop : éviction au bout de 10 minutes
  - /myshop/product/pricelist : aucune éviction
  - /myshop/shoppingCarts : éviction au bout de 30 minutes

Transaction et réplication

Avec transaction

- Réplication au commit, en 1 seul message
- Au rollback, aucune réplication

Sans transaction

- Réplication immédiate !

  1000 mises à jours => 1000 messages

Transaction et réplication

Avec transaction

- Réplication au commit, en 1 seul message
- Au rollback, aucune réplication

Sans transaction

- Réplication immédiate !

  1000 mises à jours => 1000 messages

Transactions => cache plus performant ?

- Nécessite la création de message pour faire des locks
- Fait transférer de gros volumes de données d'un coup

Réplication VS invalidation

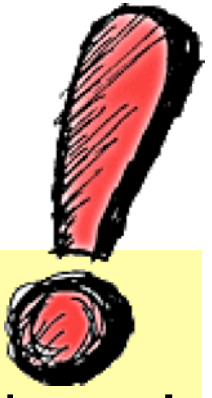
La meilleure stratégie ? Ca dépend !

- Si vous êtes le seul à connaitre l'élément
  - Utilisez l'élément lui-même en payload (réplication)
- Si l'élément est disponible en BDD, 2 options
  - La réplication : envoie de l'élément sur les autres caches
  - L'invalidation des autres caches

Réplication VS invalidation

La meilleure stratégie ? Ca dépend !

- Si vous êtes le seul à connaitre l'élément
  - Utilisez l'élément lui-même en payload (réplication)
- Si l'élément est disponible en BDD, 2 options
  - La réplication : envoie de l'élément sur les autres caches
  - L'invalidation des autres caches



Problèmes liés à l'invalidation (ou faible TTL)

- But des caches distribués ? Réduire le nombre d'acces à la base !
- L'invalidation force le rechargement des objets à partir de la BDD
- L'invalidation n'est pas adapté pour les objets souvent mis à jours

Notifications ...

Mises à jour Asynchrones

- Rend la main rapidement

Mises à jour Synchrones

- Freine l'exécution locale

Notifications …

Mises à jour Asynchrones

- Rend la main rapidement

Mises à jour Synchrones

- Freine l'exécution locale

Les mises à jour asynchrones …

- peuvent engendrer des incohérences de données

Les mises à jour synchrones …

- permet d'attendre la validation des notifications

Loaders

Charger des éléments dans le cache

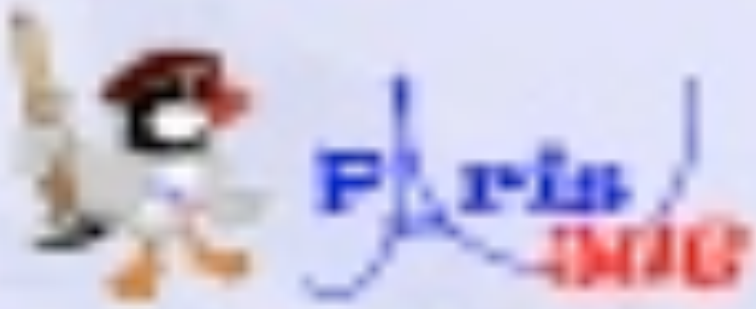
- A partir d'une source de données (store)
  - File store / Jdbc Store / Oracle Berkeley DB / Jdbm

Exporter des éléments du cache

- Lors des put et évictions …

Différents type de loaders

- Loader hiérarchiques
  - utilise TCP pour acceder à des caches distants
- Clustered loader
  - Recherche de données sur un cluster / Lazy state transfer
- Loader chainé

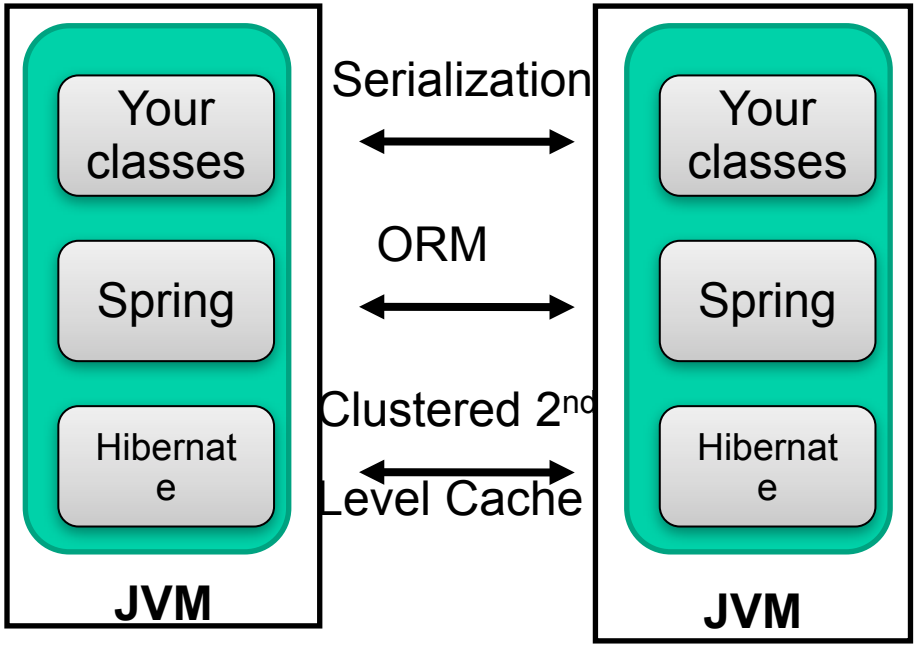


Network attached memory

www.parisjug.org

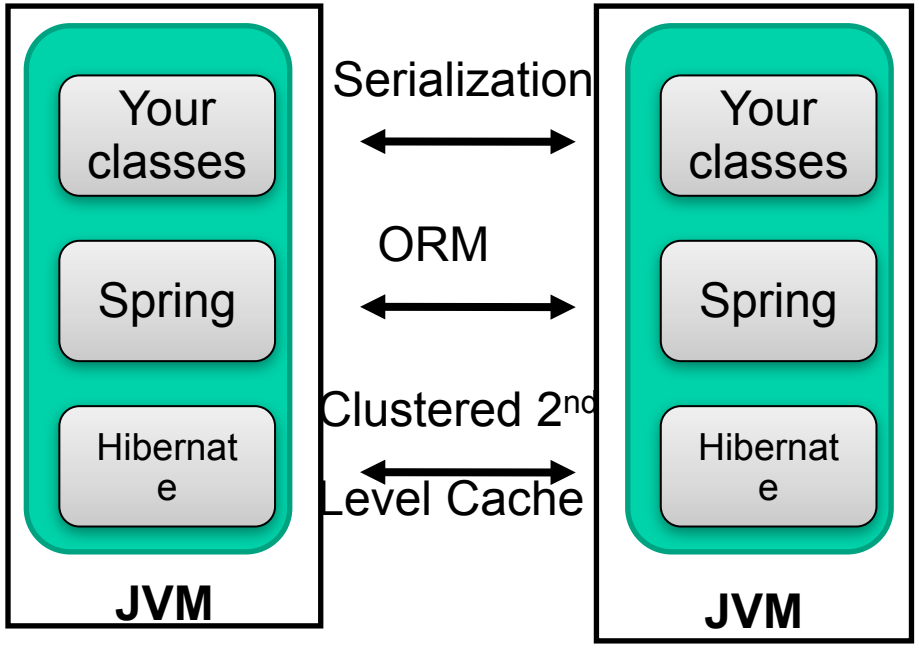
Application VS JVM clustering

Caches distribués …

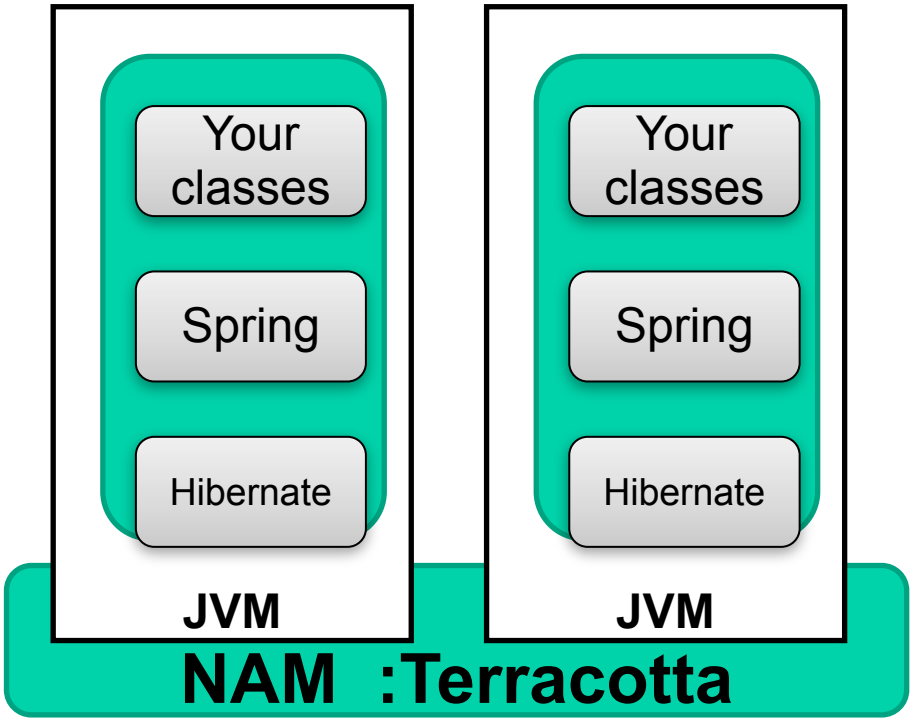


Application VS JVM clustering

Caches distribués …



Network attached memory



Cas d'utilisation :

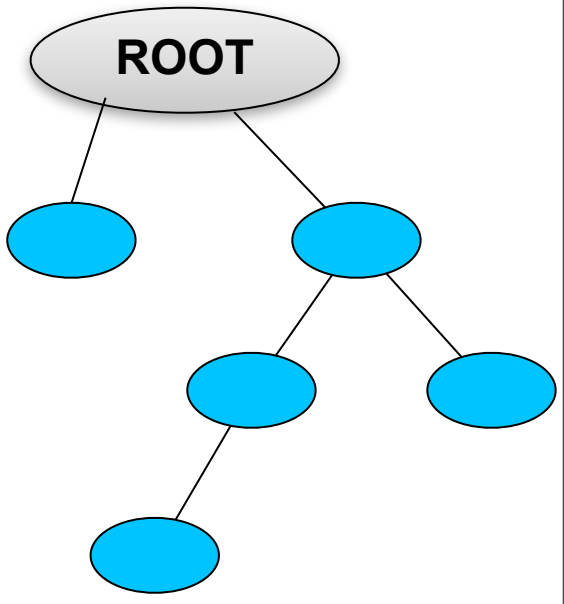
- Réplication de session
- Cache distribué
- Database offload
- Workload partitioning

Terracotta overview

Joue la carte de la transparence

- Aucun GET ni PUT
- ‘Clustering-as-a-service’

=> Utilisation massive de l’instrumentalisation

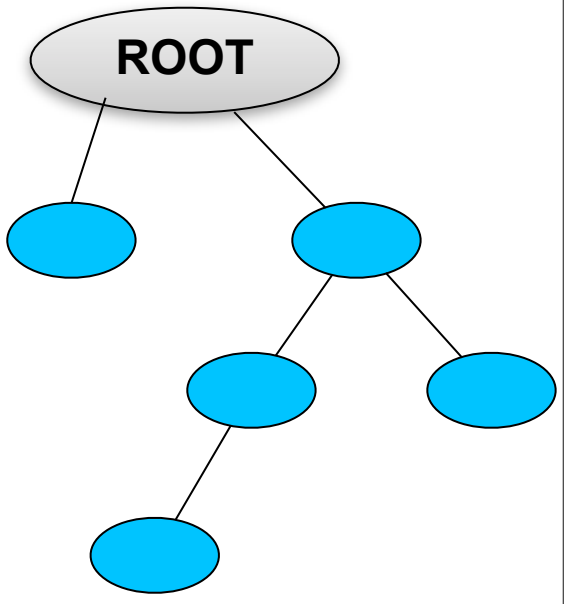


Définition d’objets ‘Root’ partagés

- Durée de vie de la JVM
- Jamais nettoyés par la JVM
- Objets ‘Super static’

Terracotta overview

Joue la carte de la transparence



- Aucun GET ni PUT
- ‘Clustering-as-a-service’

=> Utilisation massive de l’instrumentalisation

Définition d’objets ‘Root’ partagés

- Durée de vie de la JVM
- Jamais nettoyés par la JVM
- Objets ‘Super static’

Attention à ne pas partager n’importe quoi

- Trop de partage tue le partage (**surtout le réseau !!**)
- La transparence, c’est vite la porte ouverte à faire n’importe quoi

Network attached memory

Virtual Heap

- Contient tous les objets partagés du cluster
- Managée par le(s) TC server(s) Terracotta
- Les clients utilisent leur objets à leur souhait
  - Tout ou partie du des graphes d'objets
- Simulation de Swap
  - TC peut insérer des valeurs fictives pour optimiser la mémoire

Virtual GC

- Supprime les objets non référencés
  - Objets atteignable par aucune Root
  - Objets ne résidant chez aucun client

Les mécanismes de locks

Named locks

- Aux points d'entrée et sortie seulement
- Granularité approximative

Autolocks

- Plus efficace que les 'Named Locks'
- Utilisable sur des objets du cluster

Niveaux de Lock

- Ecriture
- Ecriture synchrone
- Lecture
- Concurrent

Les mécanismes de locks

Named locks

- Aux points d'entrée et sortie seulement
- Granularité approximative

Autolocks

- Plus efficace que les 'Named Locks'
- Utilisable sur des objets du cluster

Niveaux de Lock

- Ecriture
- Ecriture synchrone
- Lecture
- Concurrent

⇒ Il manquerait une intégration avec un Transaction Manager Spring

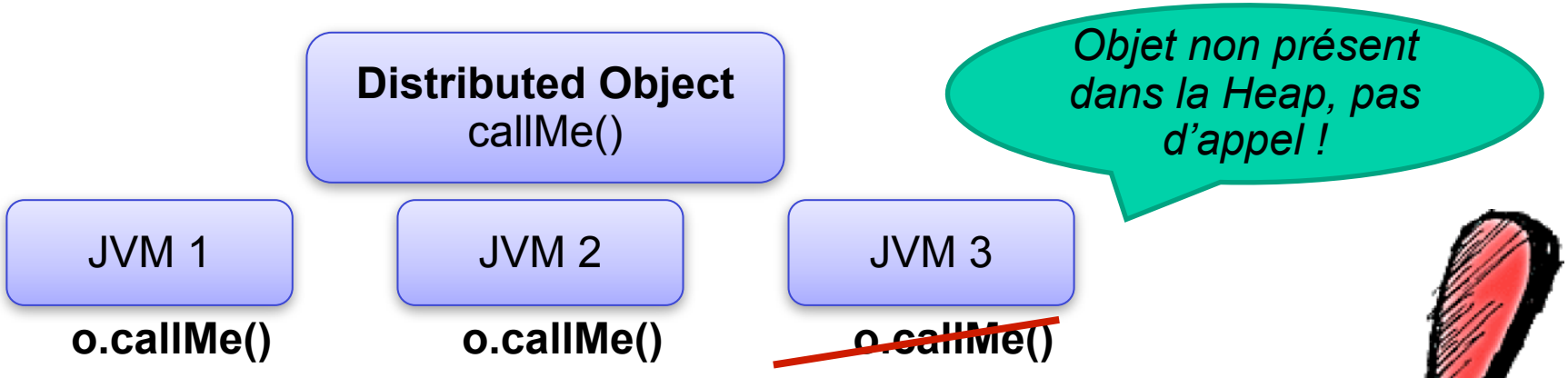
Invocation de méthodes distribuées (DMI)

- Déclaration de DM sur un objet du cluster
- Appel de l'a méthode sur l'un des nœud du cluster
- Chaque JVM possédant cet objet exécute cette méthode

Distributed Method Invocation

Invocation de méthodes distribuées (DMI)

- Déclaration de DM sur un objet du cluster
- Appel de l'a méthode sur l'un des nœud du cluster
- Chaque JVM possédant cet objet exécute cette méthode

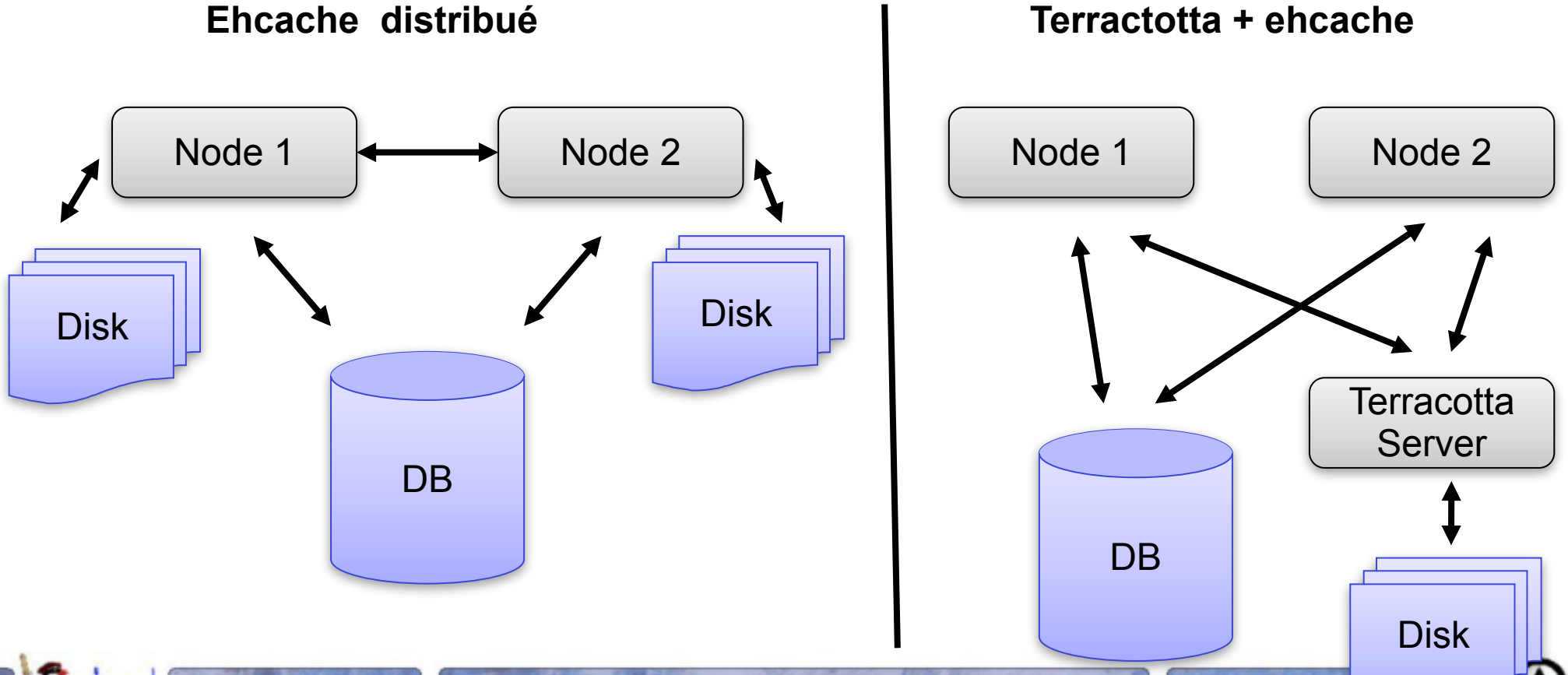


Attention à l'utilisation !

- Aucune garantie que la méthode soit exécutée partout !
- Mauvaise idée si l'on veut coordonner les actions aux données

Terracotta est utilisable pour du caching

- Avec EHCache (en mode non distribué)
- Le cache est persistable sur le disque



Terracotta optimizations

Privilégie la lecture dans les caches mémoire

- Les champs des objets sont séparés sur le disque, mais restent aussi proches que possible
- Chaque champ est récupérable individuellement

Lit seulement ce qu'on a besoin, rien de plus

- Lecture à partir du serveur ⇔ temps lecture sur disque
- Message customisé, la JVM patch sa propre mémoire

Ecriture en batch, des mises à jour fines

- N'utilise pas la sérialisation Java
- Ecriture champ à champs => on ne transmet pas le graph complet

Ecriture sur disque en 'append' uniquement

- Rien n'est trié, indexé, déplacé sur le disque

Terracotta optimizations

Privilégie la lecture dans les caches mémoire

- Les champs des objets sont séparés sur le disque, mais restent aussi proches que possible
- Chaque champ est récupérable individuellement

Lit seulement ce qu'on a besoin, rien de plus

- Lecture à partir du serveur ⇔ temps lecture sur disque
- Message customisé, la JVM patch sa propre mémoire

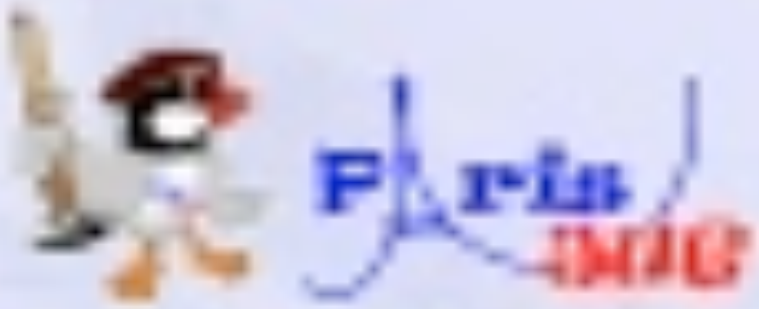
=> Peu générer beaucoup de requêtes sur le réseau

Ecriture en batch, des mises à jour fines

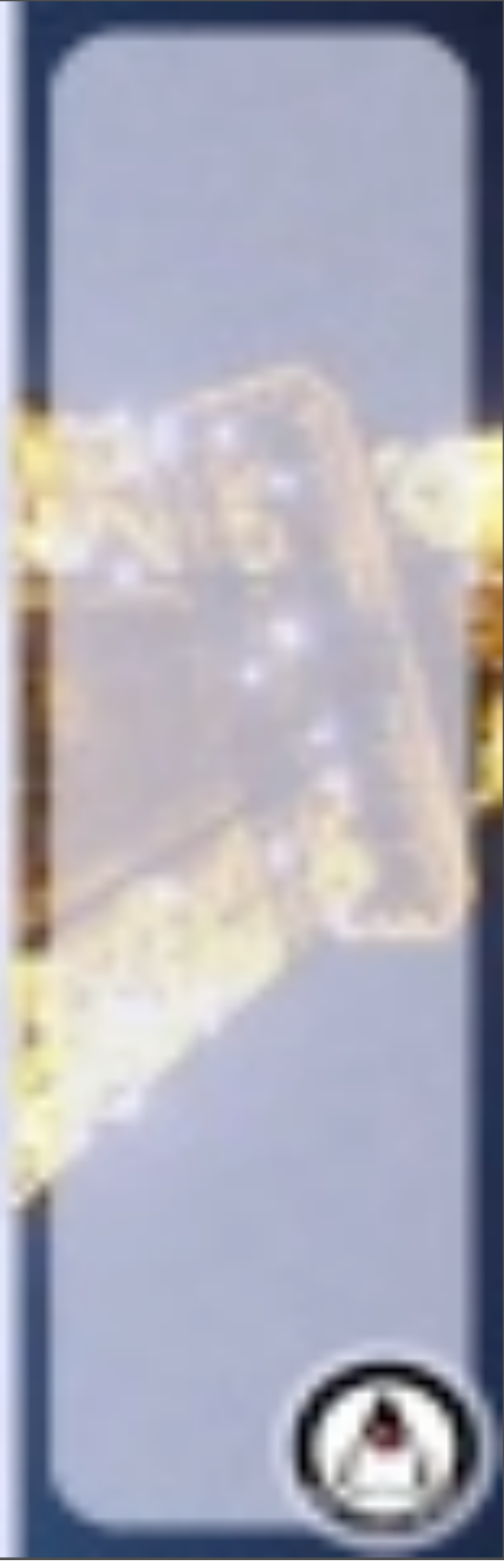
- N'utilise pas la sérialisation Java
- Ecriture champ à champs => on ne transmet pas le graph complet

Ecriture sur disque en 'append' uniquement

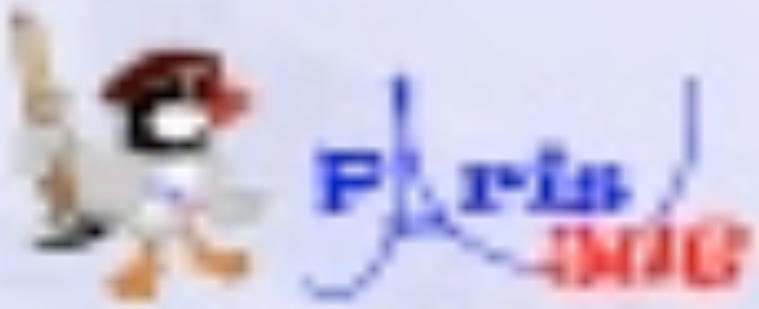
- Rien n'est trié, indexé, déplacé sur le disque



Grilles de données



www.parisjug.org

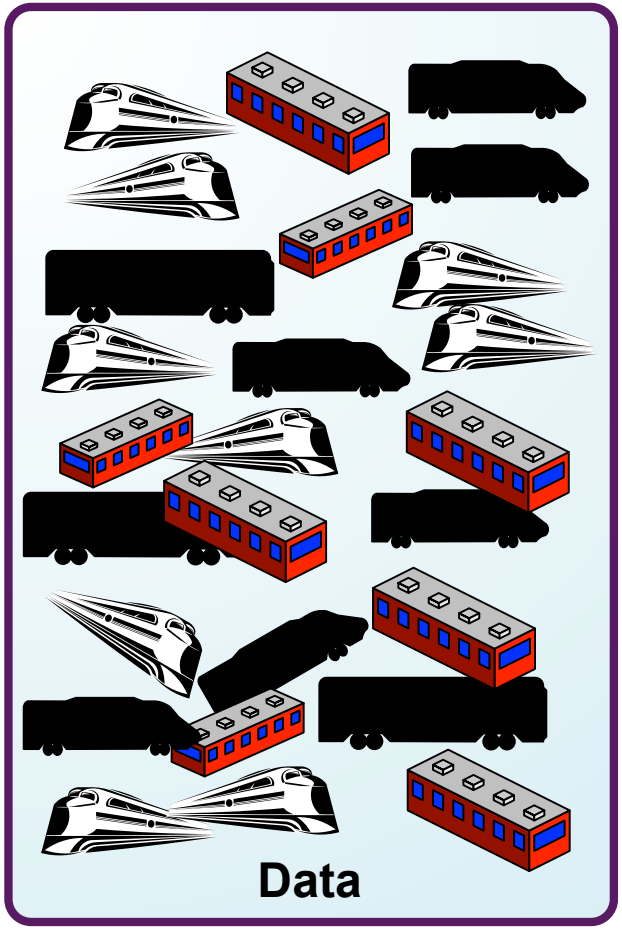


Partitionner pour tenir la charge

www.parisjug.org

Partitionner pour tenir la charge

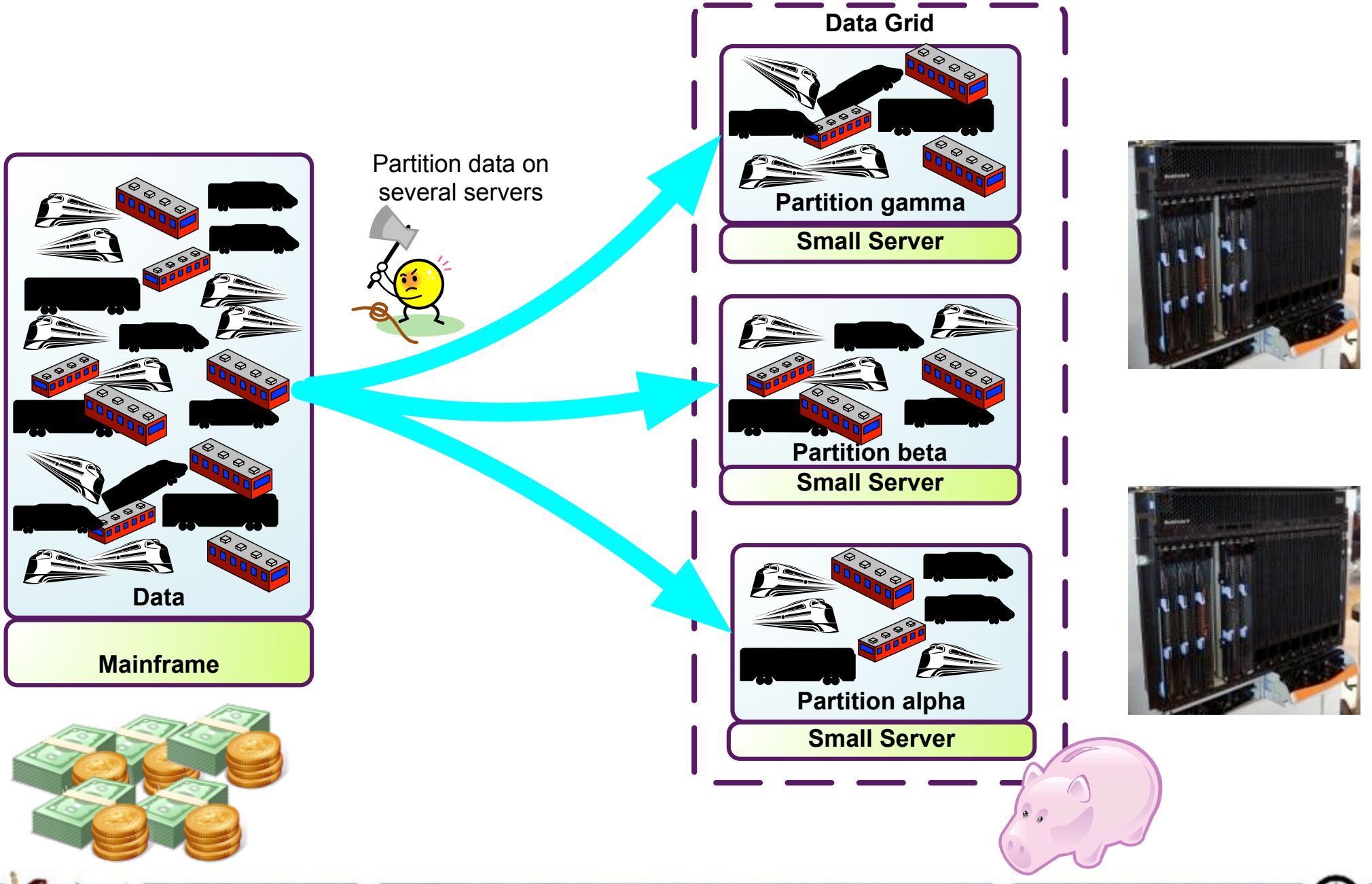
Tout sur le même serveur :
déjà vu, souvent possible mais très couteux

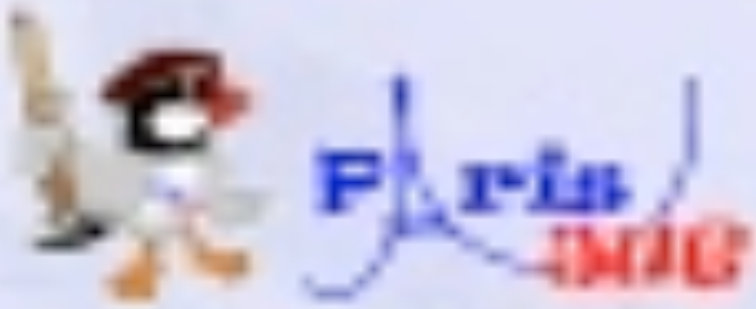


Mainframe

Jusqu'à 1.5 To de RAM
et 64 processeurs

Partitionner pour tenir la charge

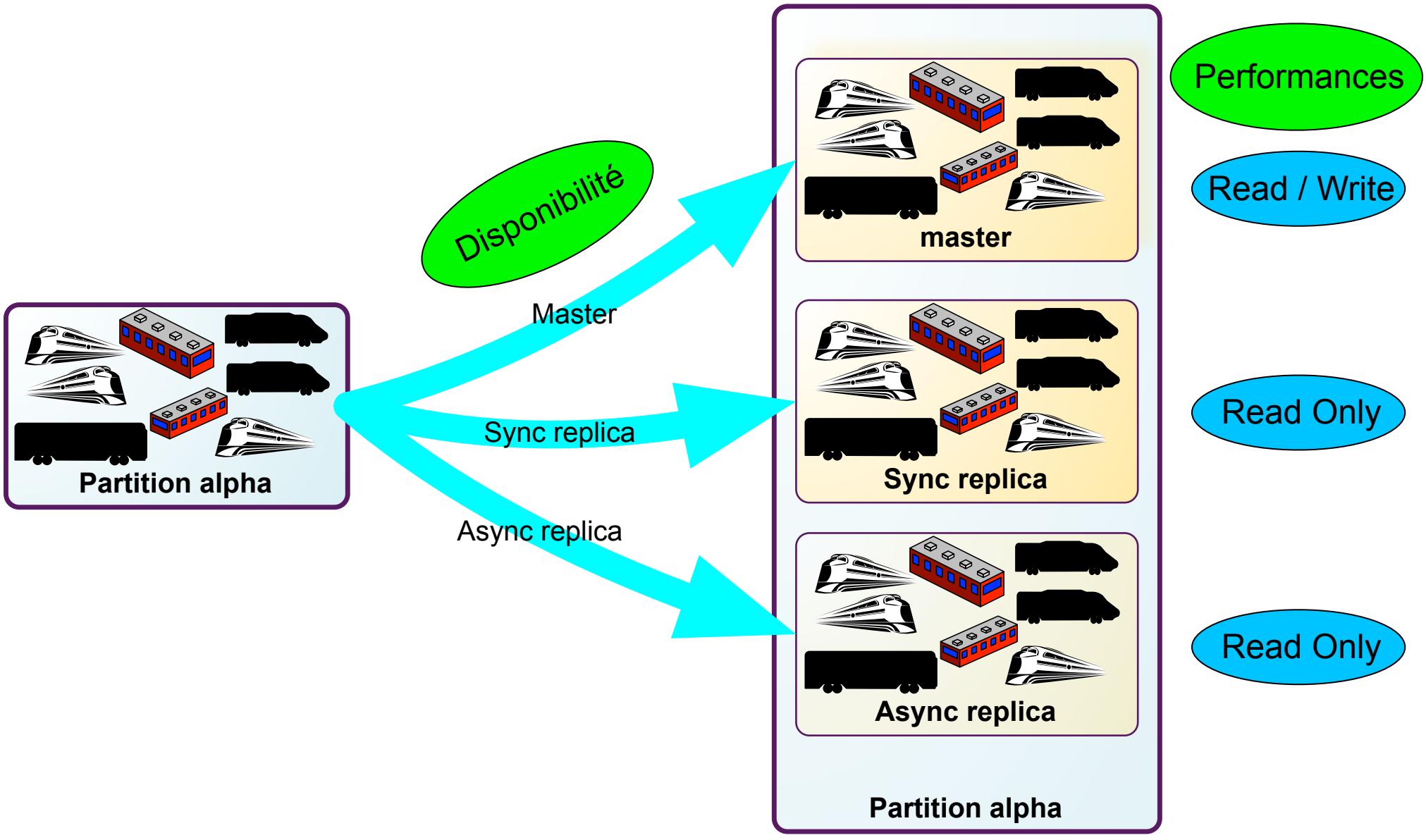




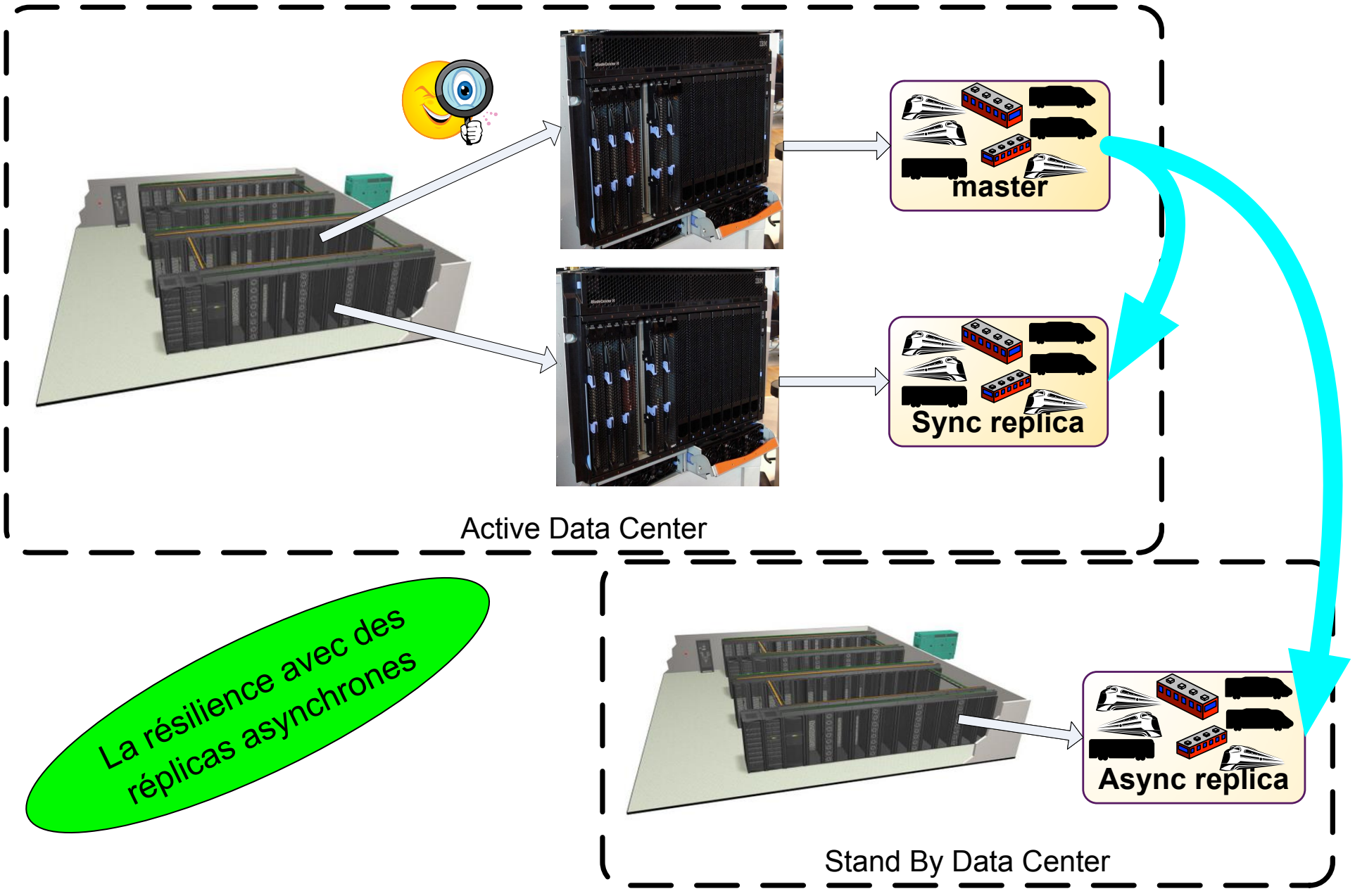
Répliquer pour la disponibilité

www.parisjug.org

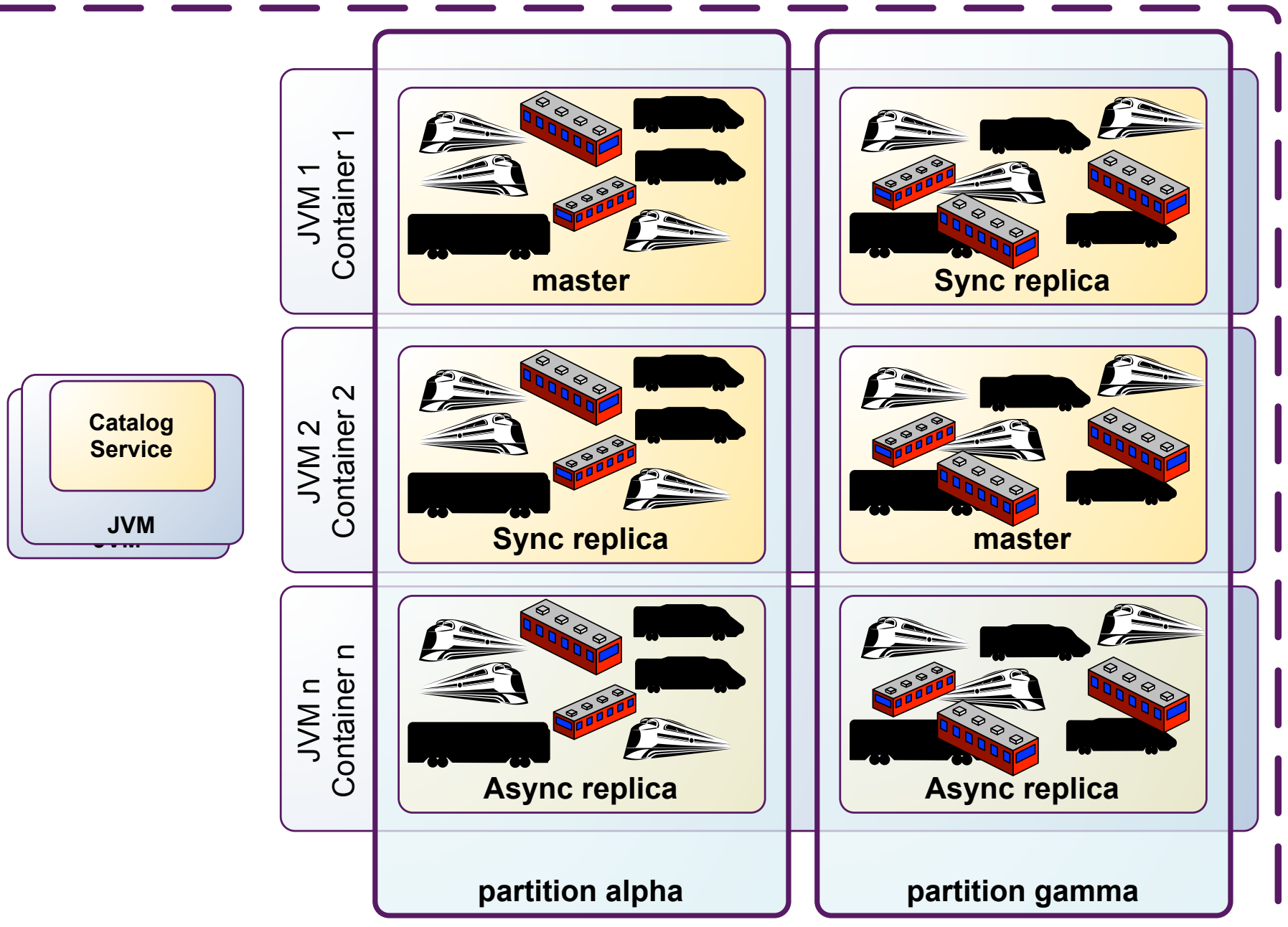
Répliquer pour la disponibilité

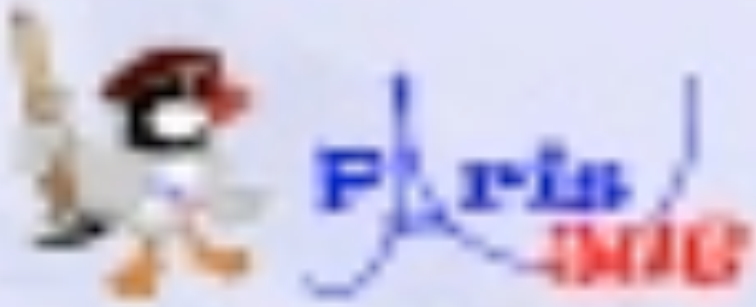


Répliquer pour la disponibilité



Partitionner et Répliquer



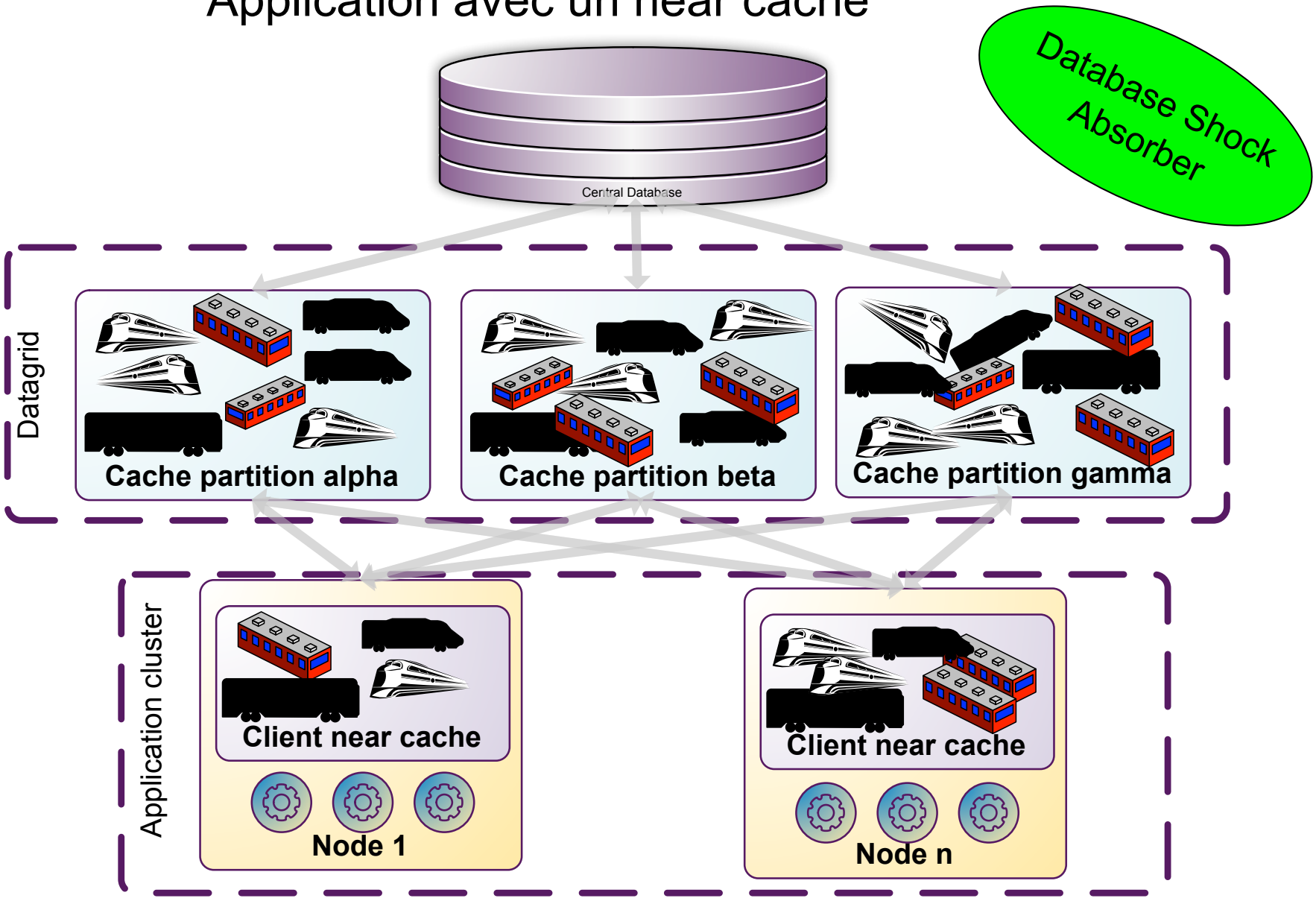


Du Near Cache à la Data Grid

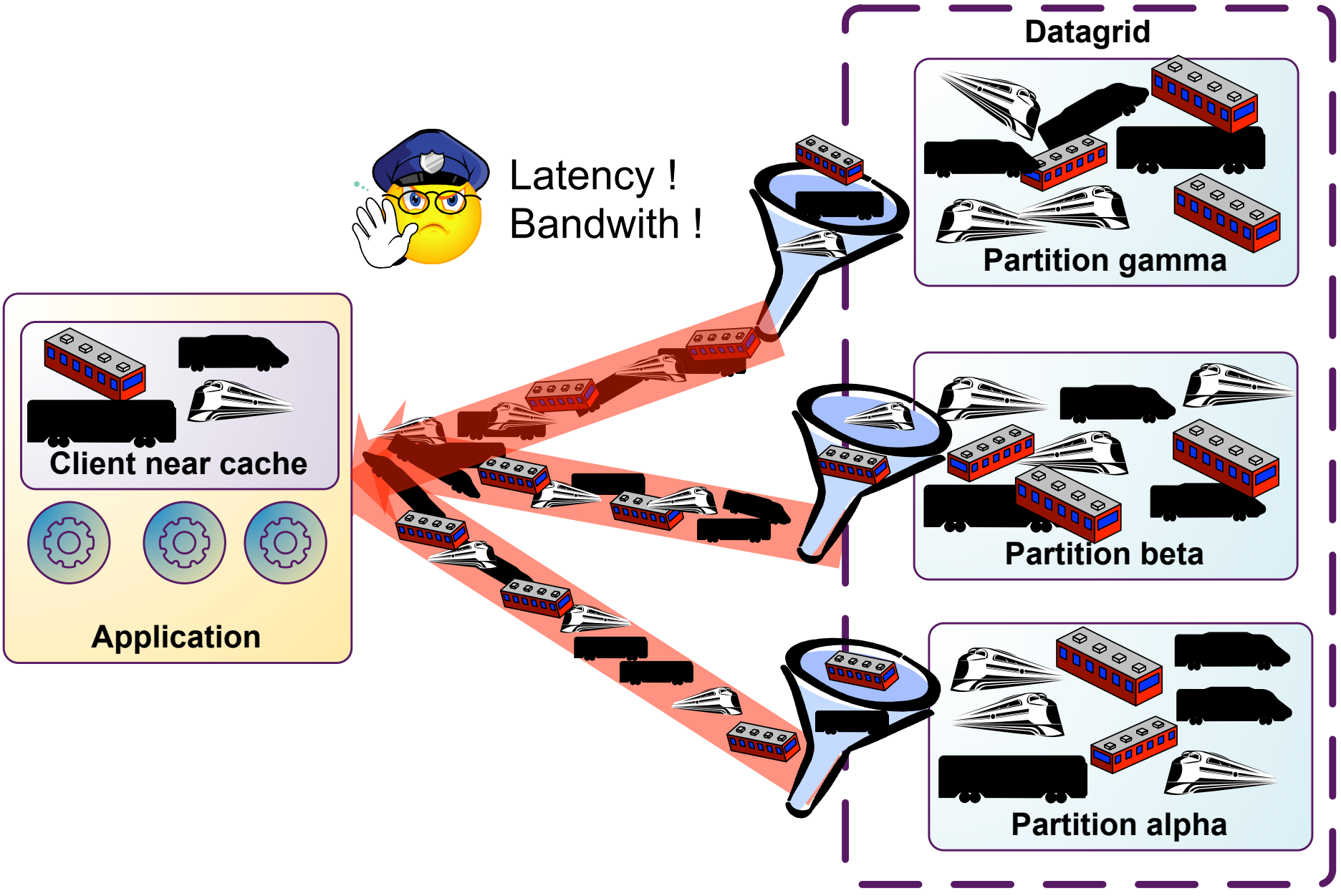
www.parisjug.org

Le Near Cache

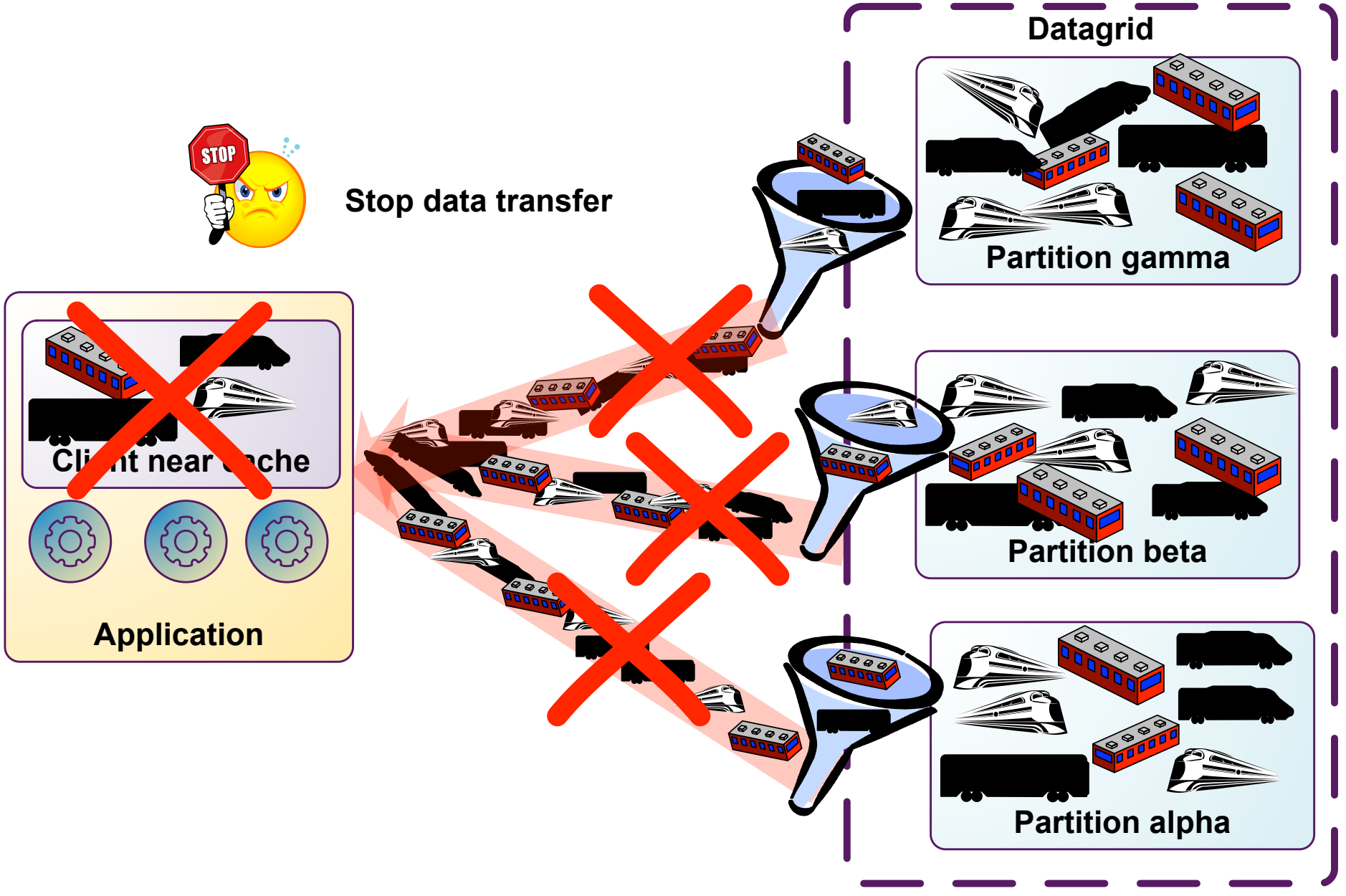
Application avec un near cache



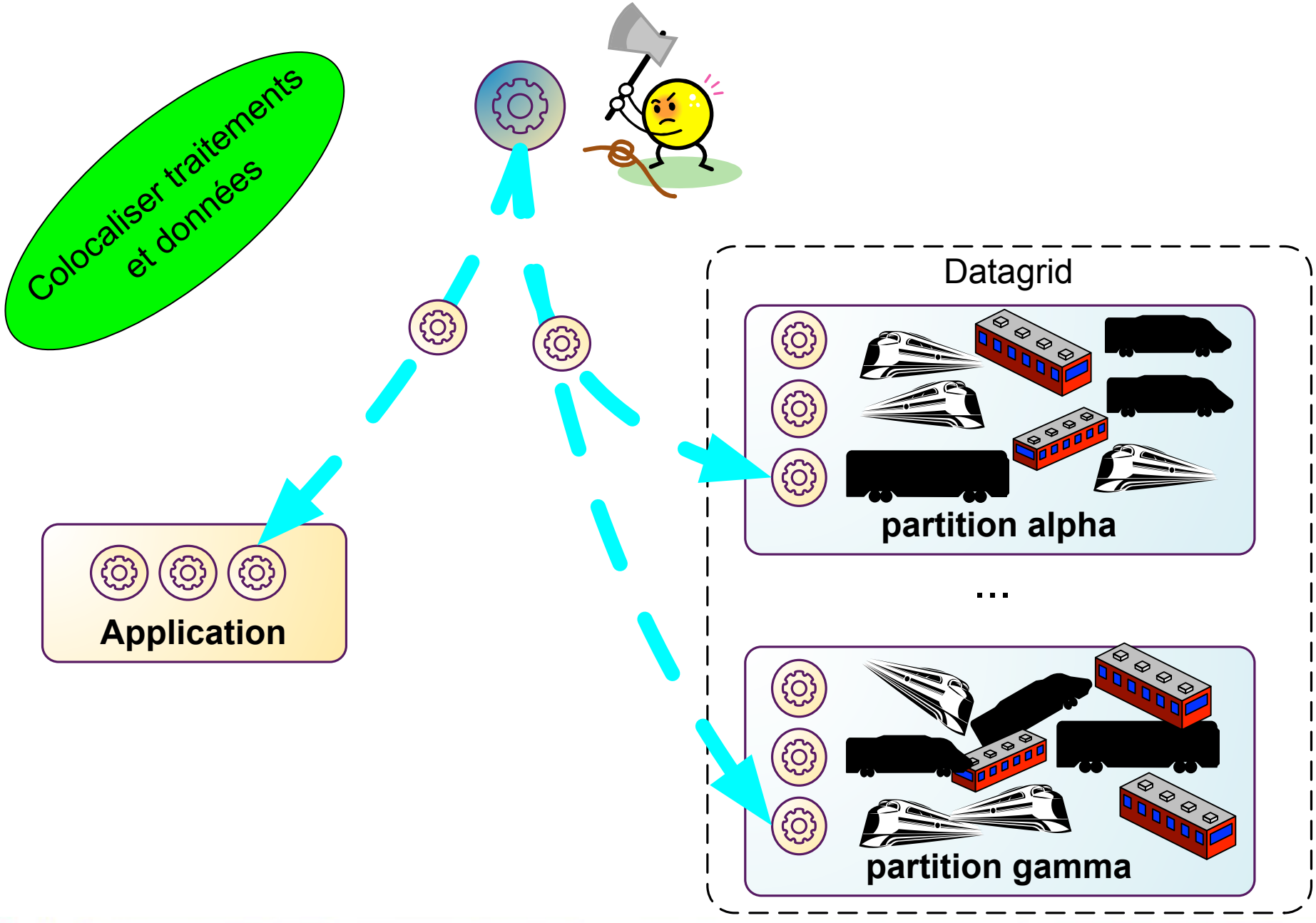
Les limites du Near Cache

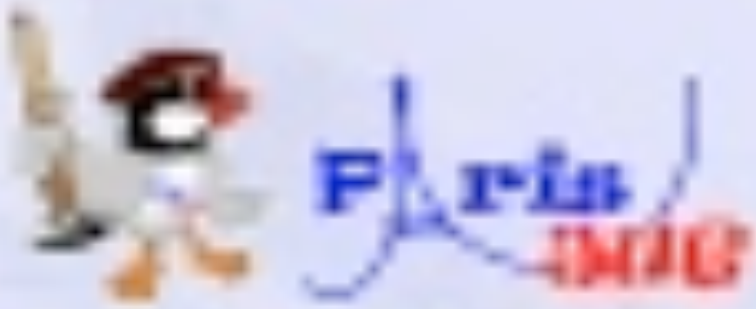


Du Near Cache à la Grille de Données



Du Near Cache à la Grille de Données





La grille de donnée

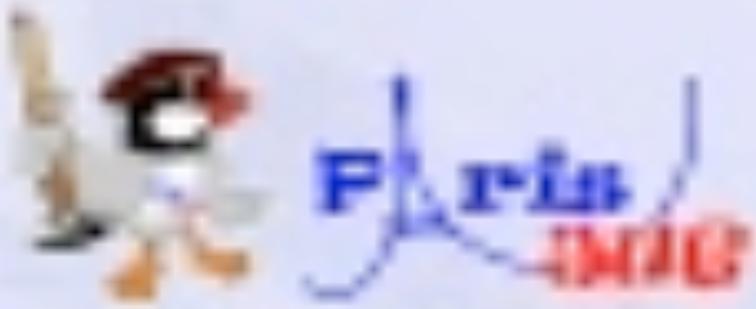
www.parisjug.org

La grille de données

Reduced data transfer
Collocated data & business logic

La grille de données

- Les données sont partionnées
- Traitements et données sont colocalisés
- Le volume de données échangées est limité aux requêtes et aux fragments de résultat

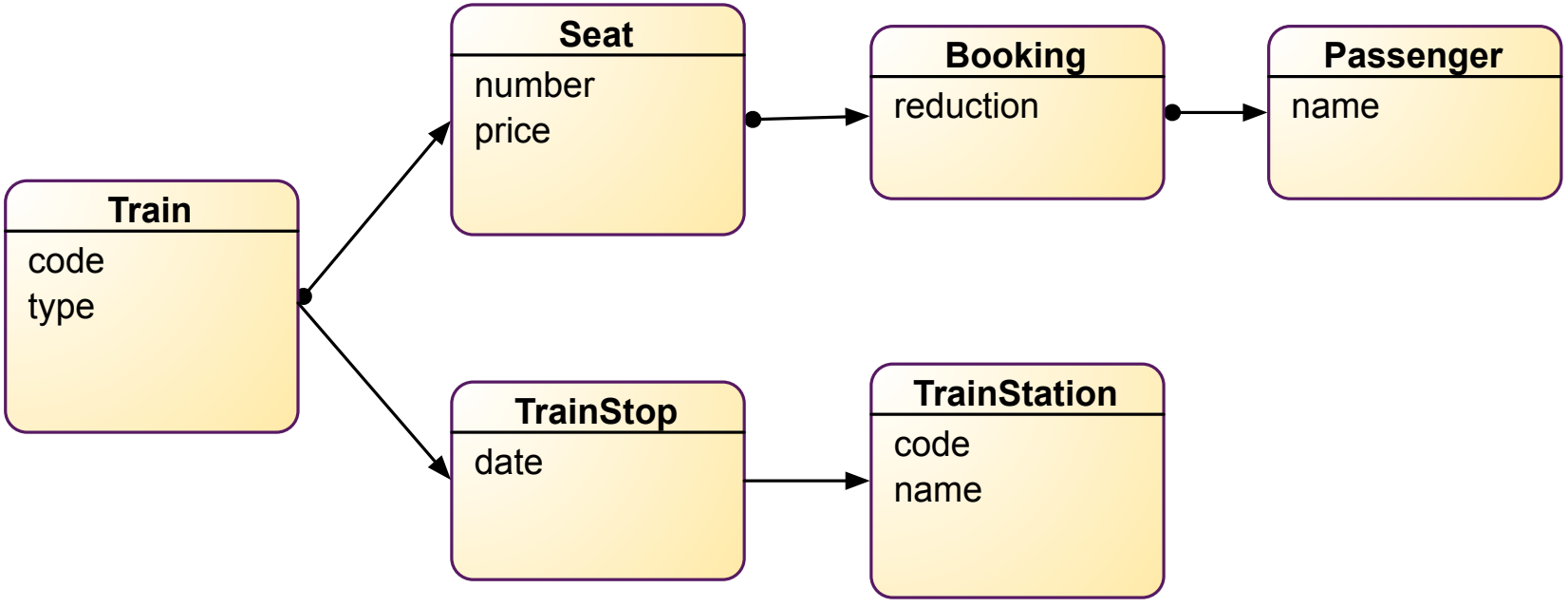


Structurer ses données pour la grille

www.parisjug.org

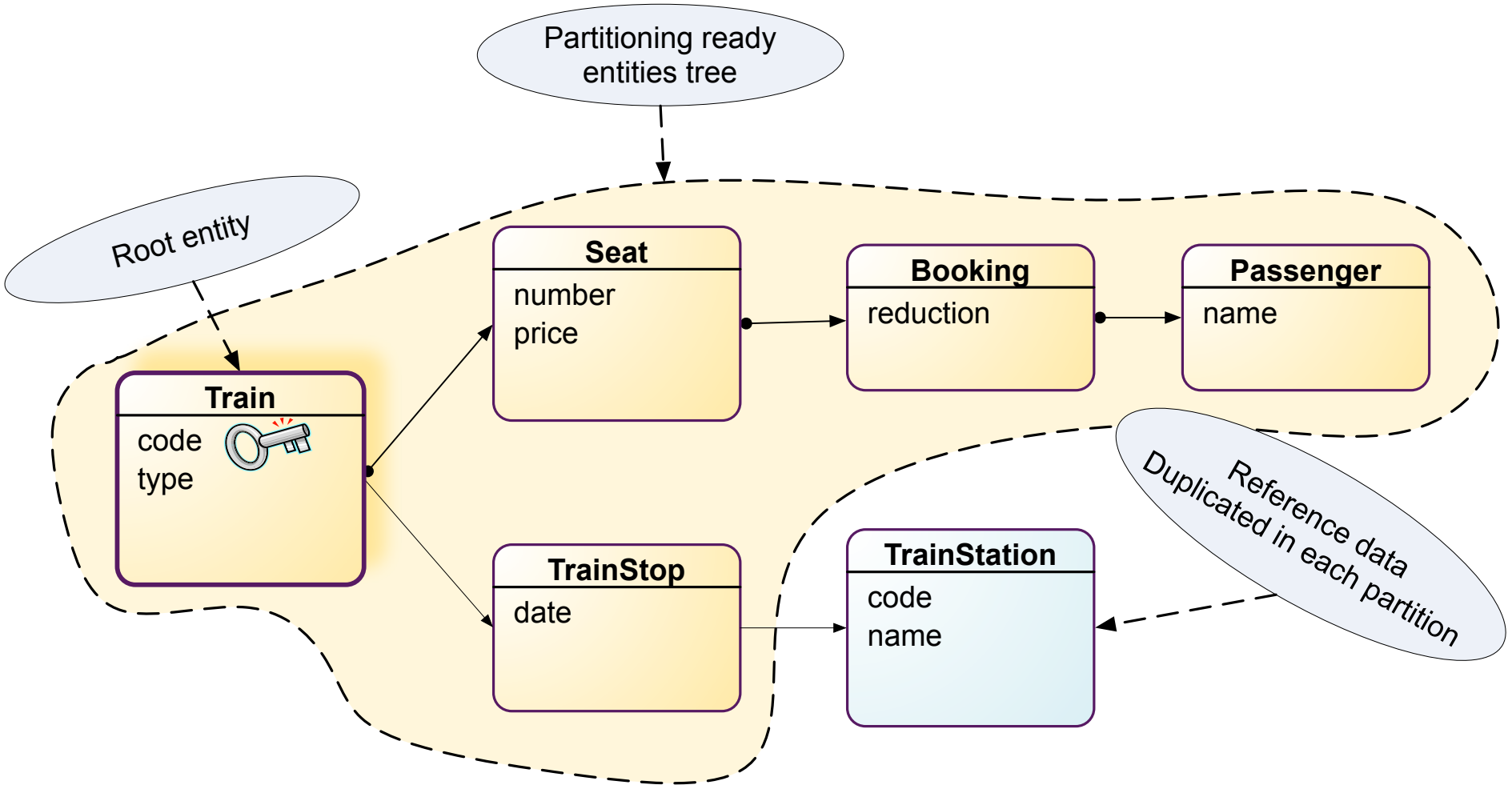
Un système de réservation de billets de train !

Partitionner les données



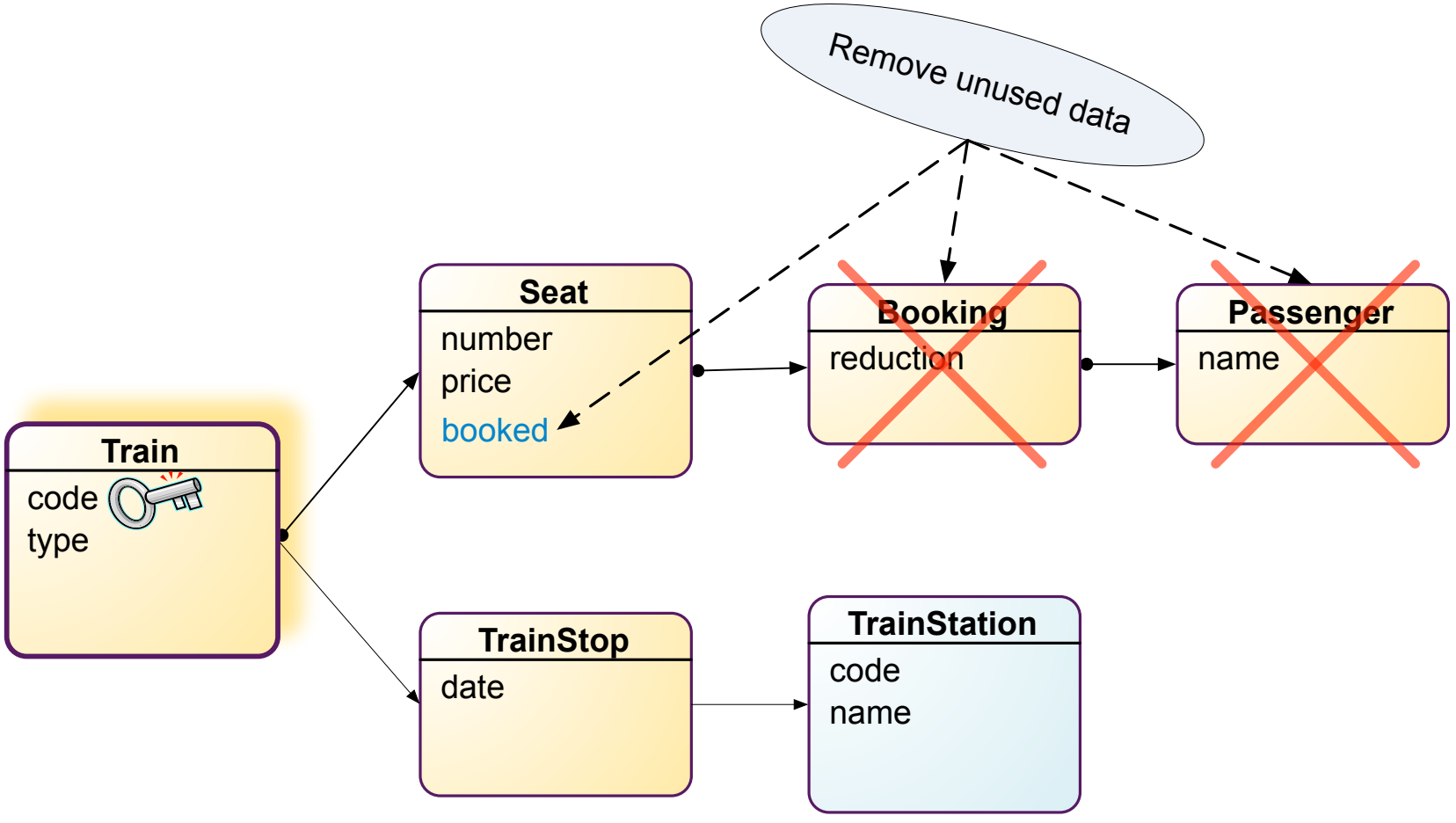
Modélisation orientée objet *classique*

Partitionner les données



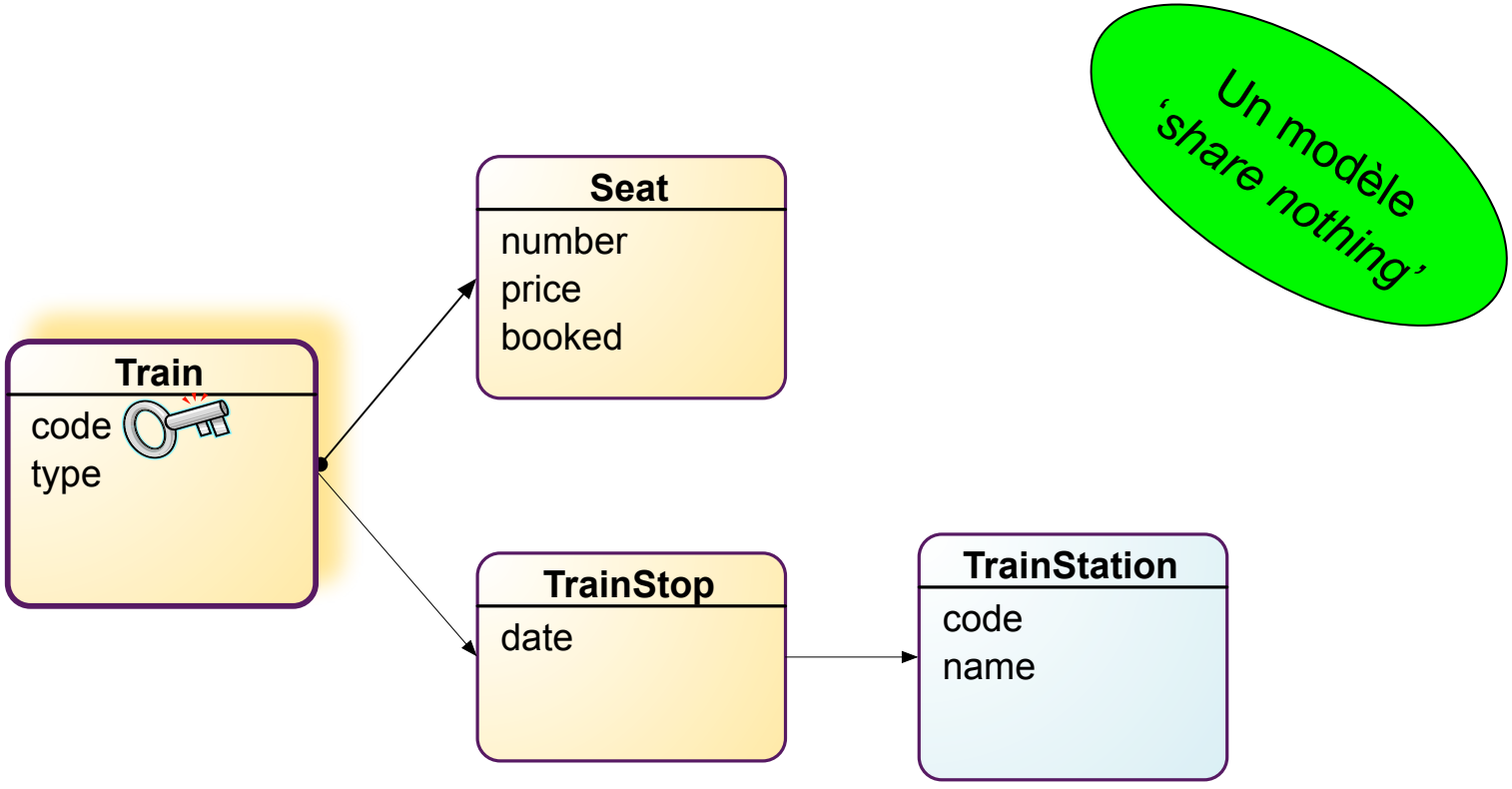
Root entity, clef de partitionnement, sub-entities & duplication de données de référence

Partitionner les données



Suppression des données inutiles à la logique métier

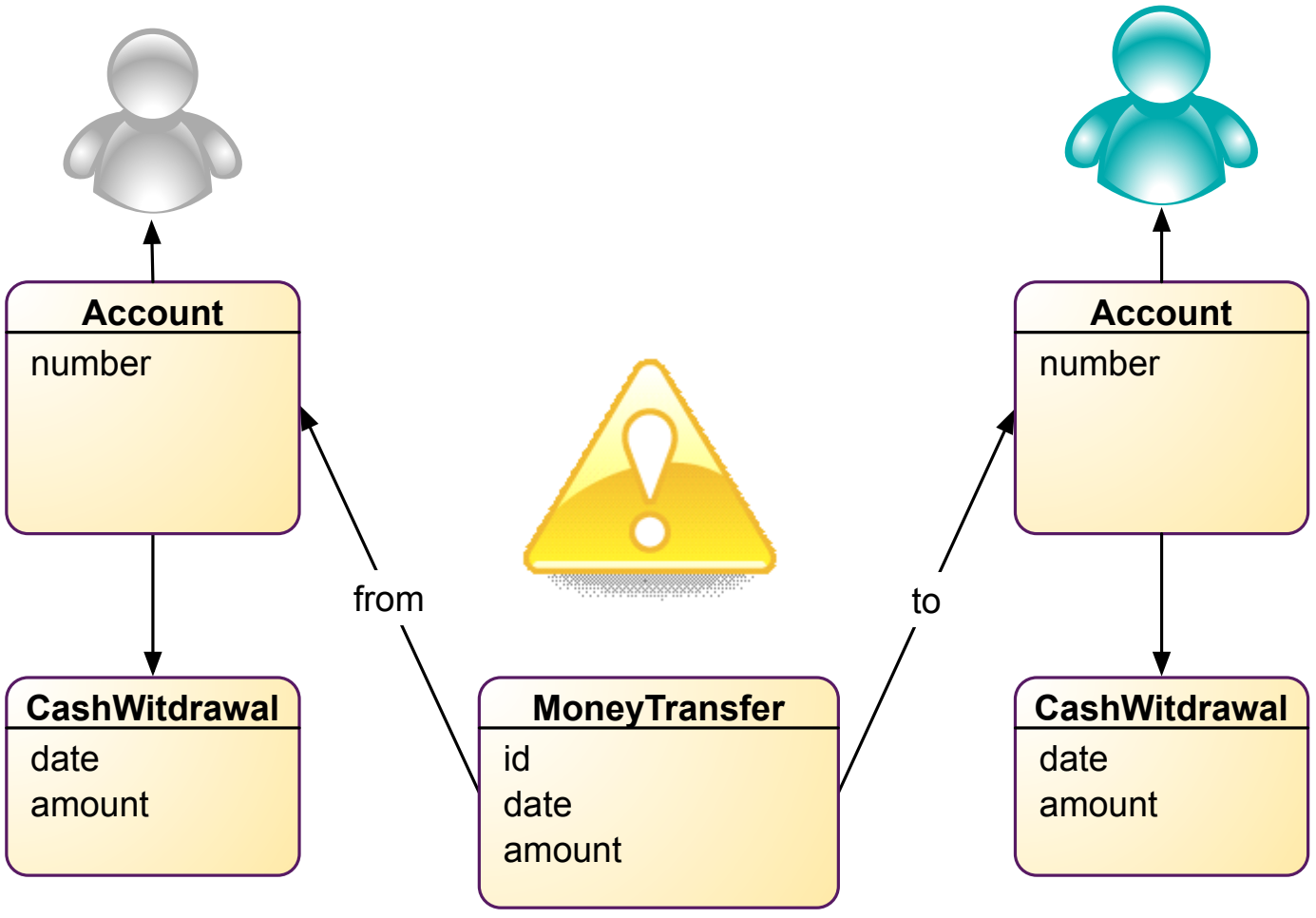
Partitionner les données



Modélisation partitionnée '*grid ready*'

Des données façonnées au besoin métier

Partitionner les données

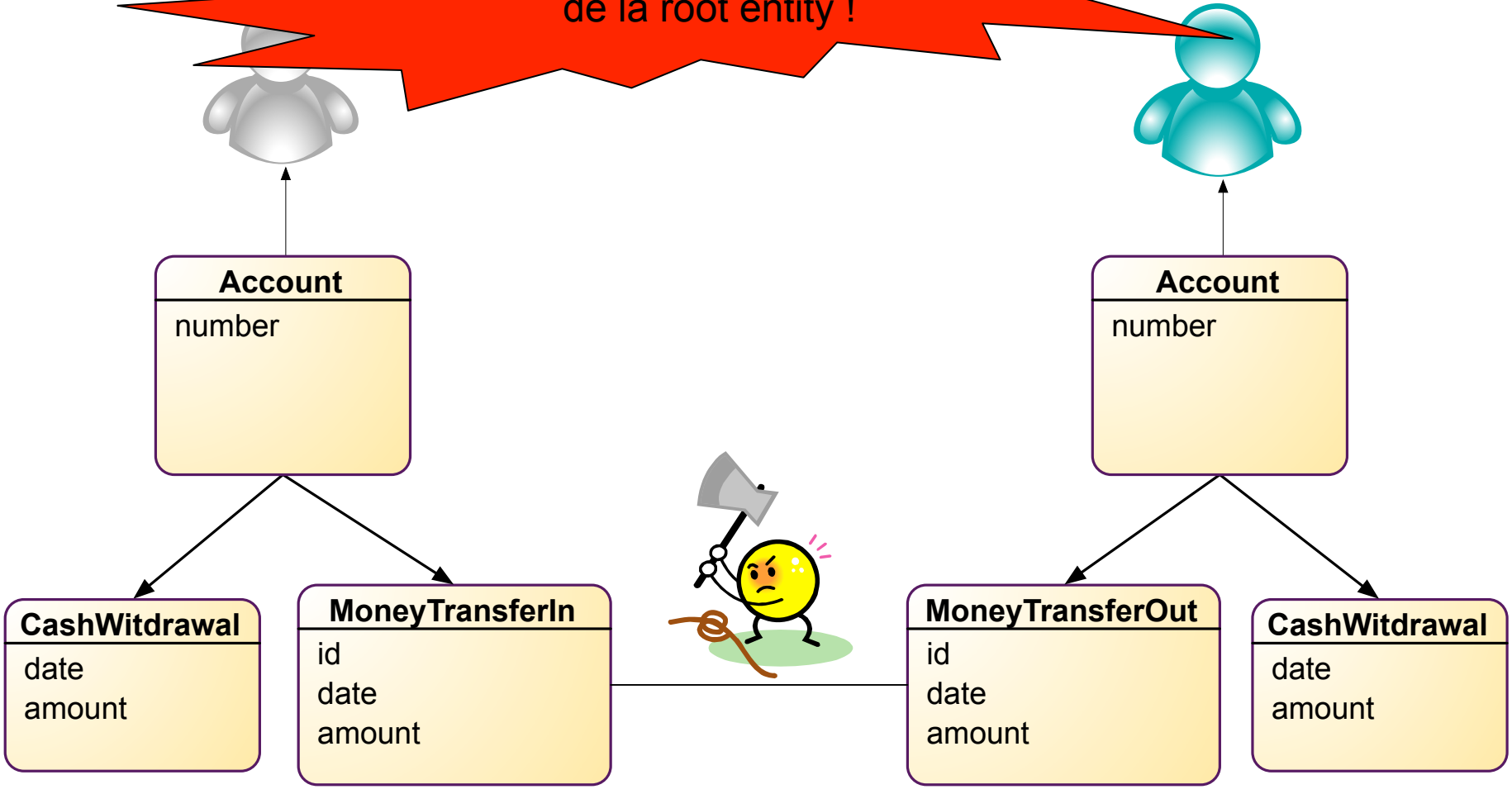


Casser les relations pour partitionner

Situation de comptes bancaires

Partitionner les données

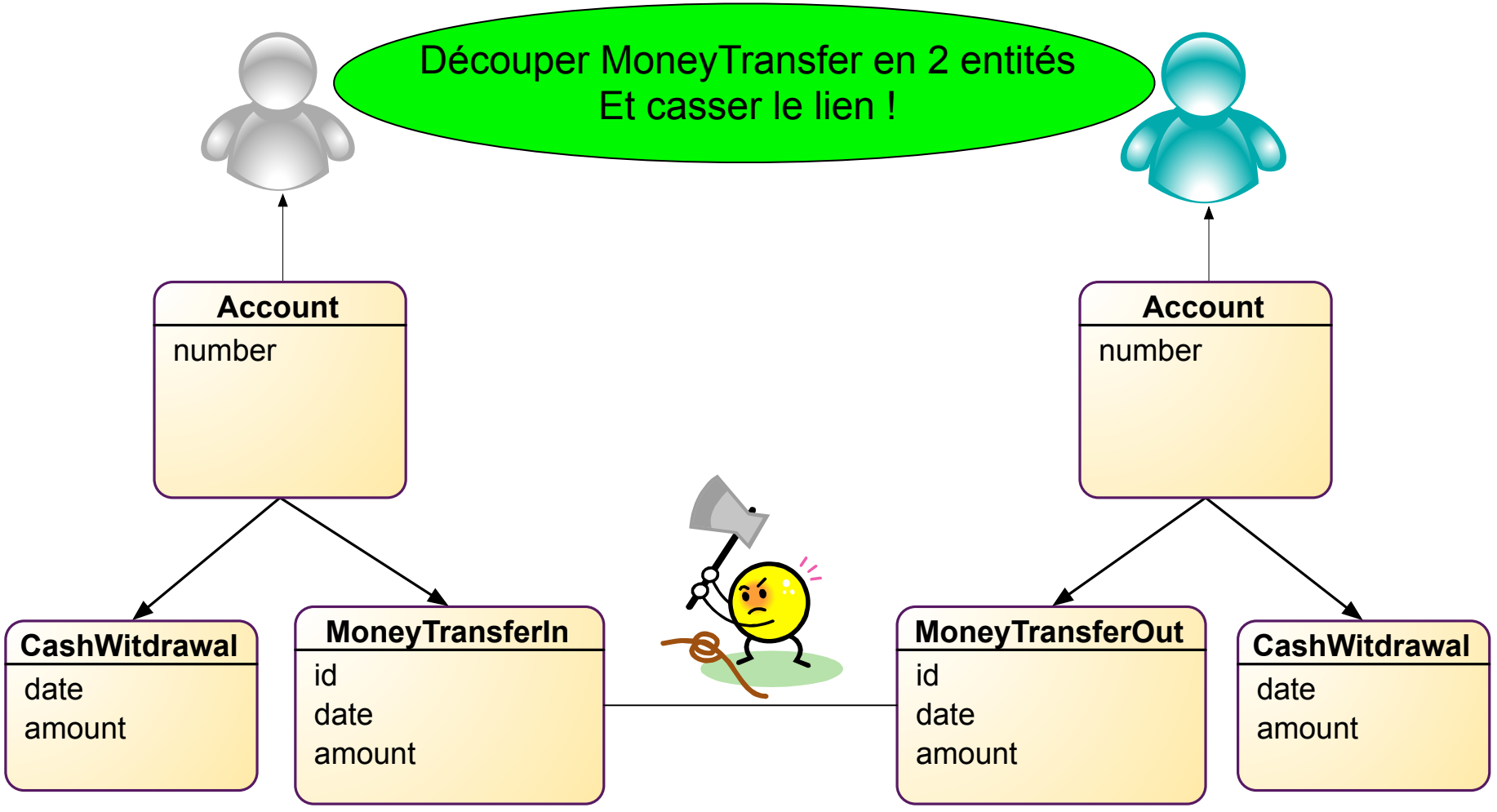
MoneyTransfer est partagé par 2 instances de la root entity !



Casser les relations pour partitionner

Situation de comptes bancaires

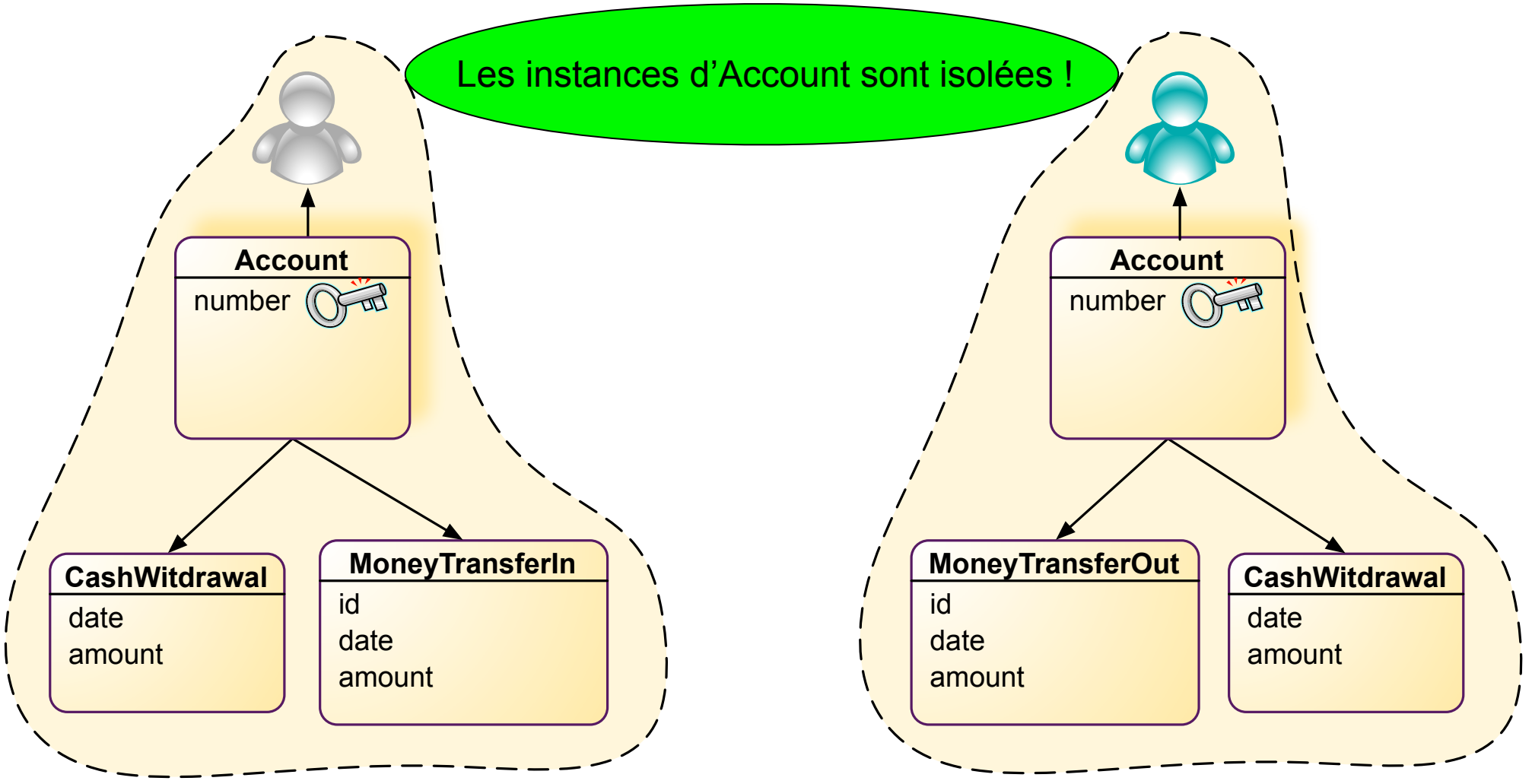
Partitionner les données



Casser les relations pour partitionner

Situation de comptes bancaires

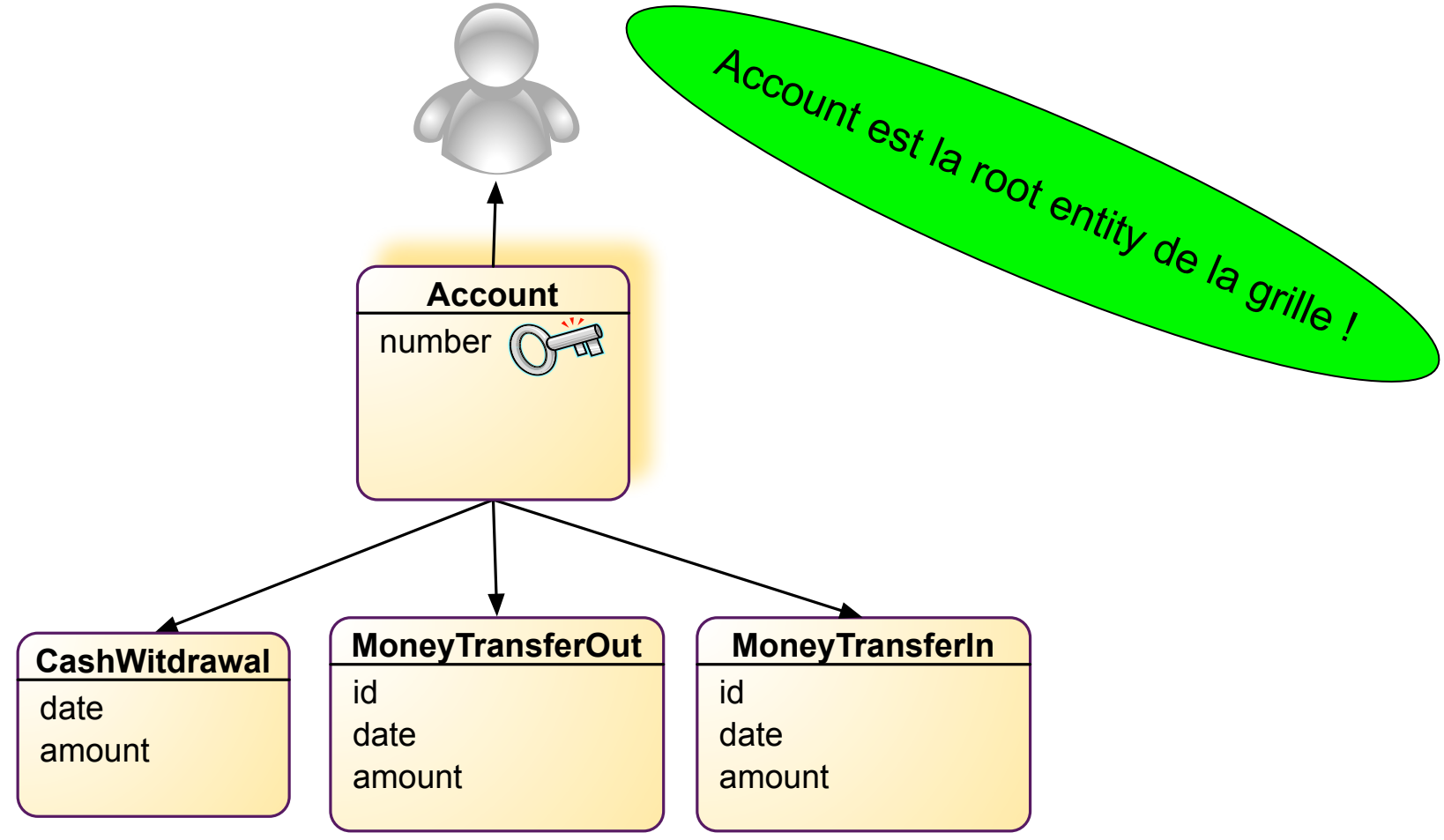
Partitionner les données



Casser les relations pour partitionner

Situation de comptes bancaires

Partitionner les données

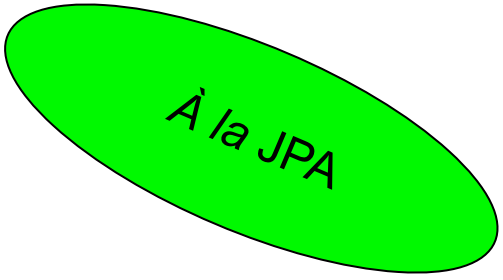


Modélisation partitionnée '*grid ready*'

Des données façonnées au besoin métier

Le modèle de programmation

```java
@Entity(schemaRoot = true)
public class Train implements Serializable {

    /**
     * Train's business identifier
     */
    @Basic
    @Index
    protected String code;

    @Id
    protected long id;

    @OneToMany(cascade = CascadeType.ALL)
    protected List<Seat> seats = new ArrayList<Seat>();

    @OneToMany(cascade = CascadeType.ALL)
    protected List<TrainStop> trainStops = new ArrayList<TrainStop>();

    @Basic
    protected Type type;

    @Version
    protected int version;
```



Le modèle de programmation

API de haut niveau *à la* JPA (persist, merge, remove, find)

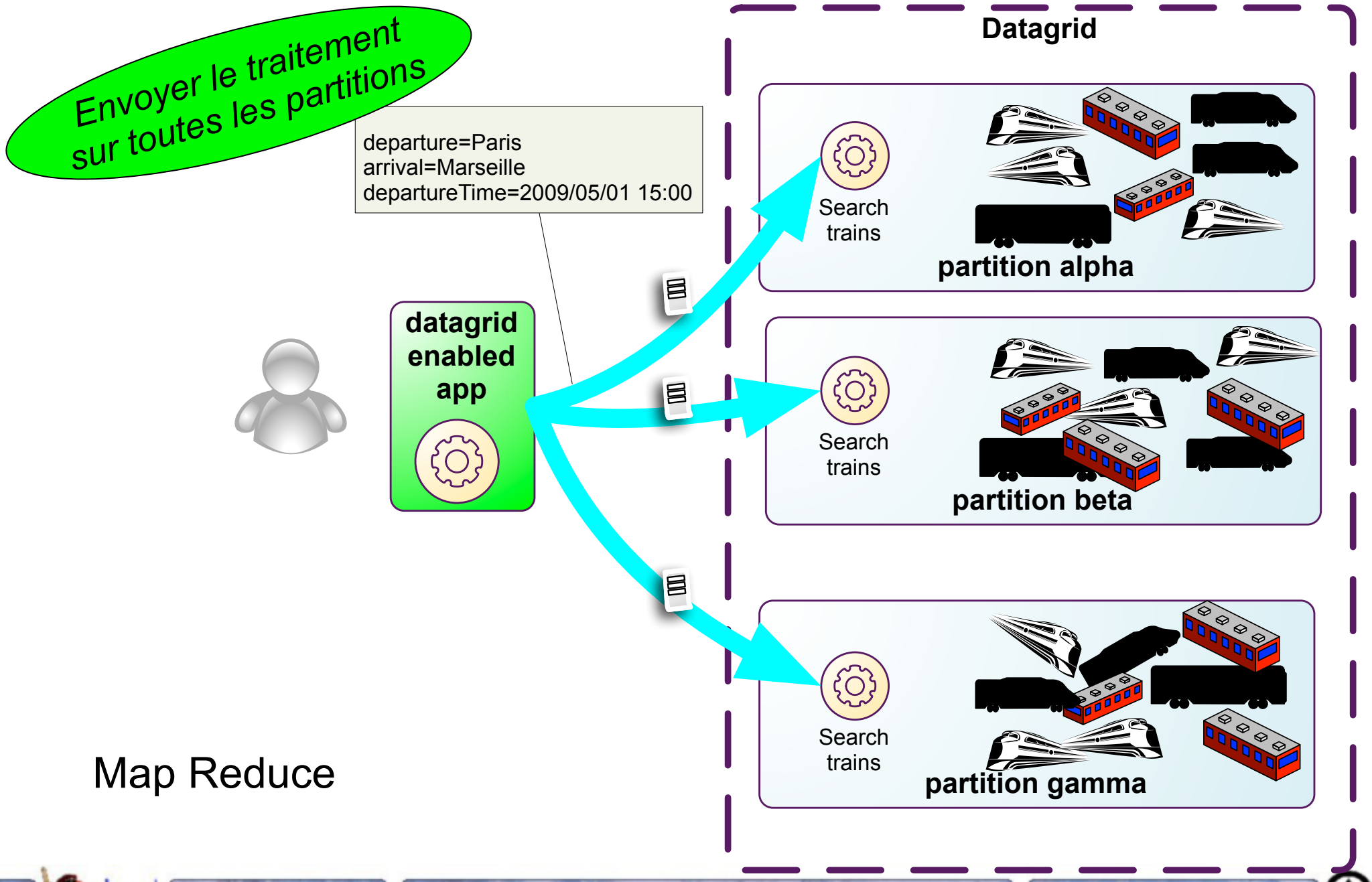
Relations entre les entités (@OneToMany, @ManyToOne, @OneToOne)

Versioning (@version)

Indexation des attributs (@Index)

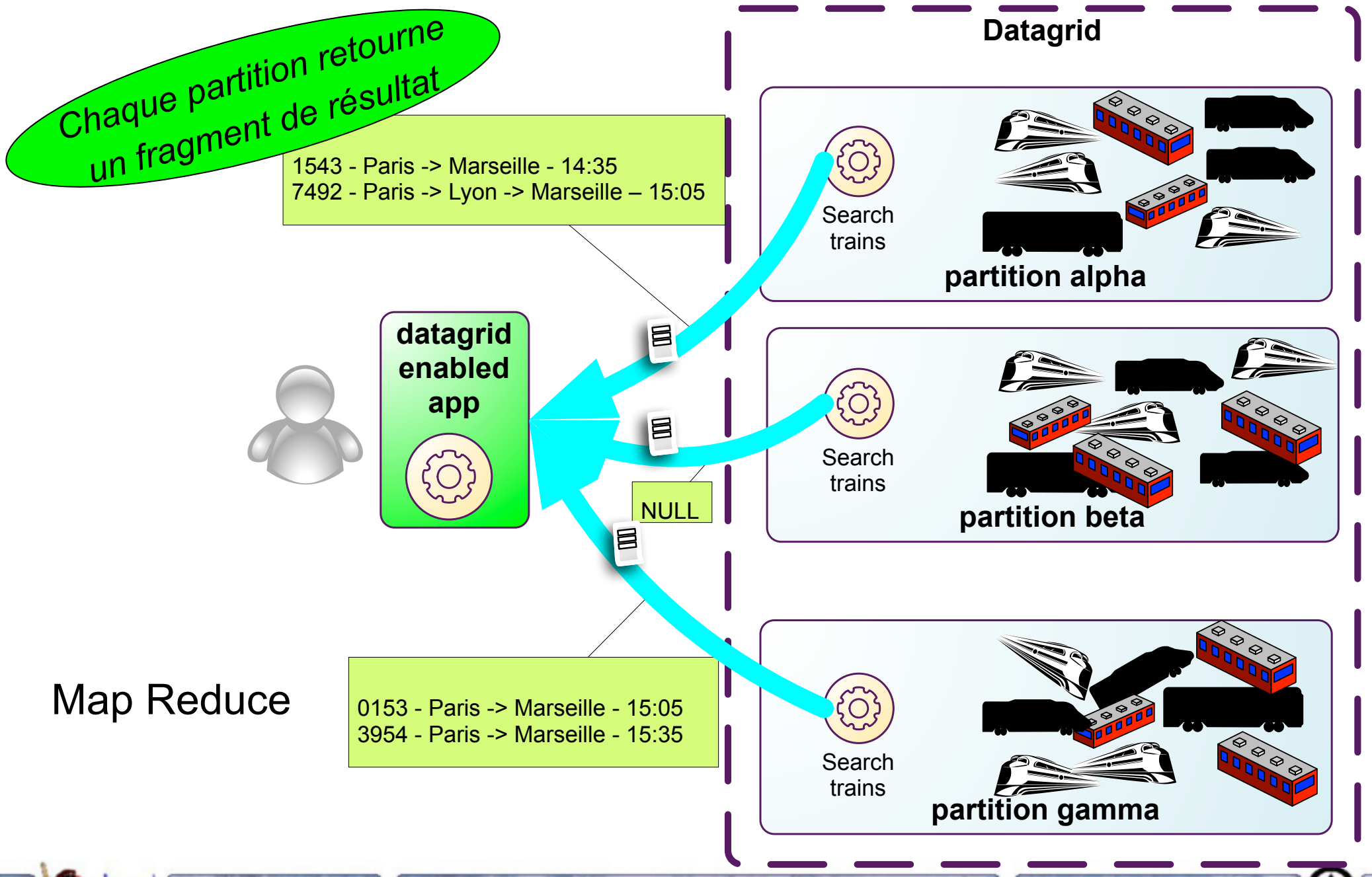
Query language ("select e from Employee e where e.lastName=:lastName")

Les Design patterns



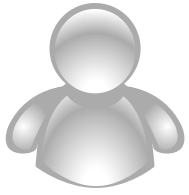
Map Reduce

Les Design patterns



Les Design patterns

Le client agrège les fragments de résultats



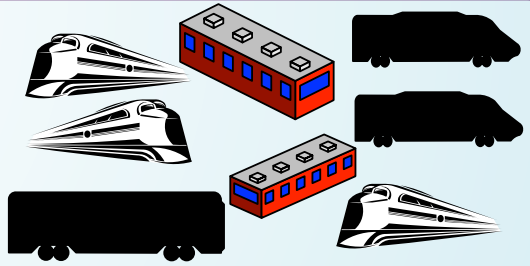
datagrid enabled app

1543 - Paris -> Marseille - 14:35
7492 - Paris -> Lyon -> Marseille – 15:05
0153 - Paris -> Marseille - 15:05
3954 - Paris -> Marseille - 15:35

Map Reduce

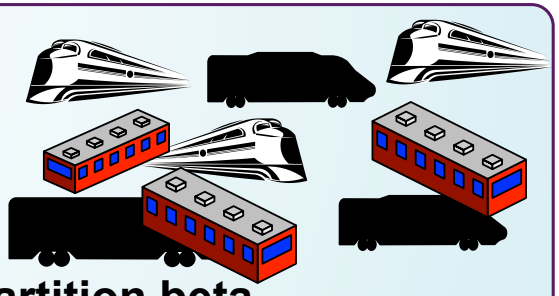
Datagrid

Search trains



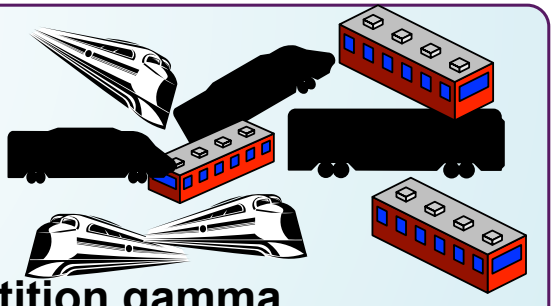
partition alpha

Search trains



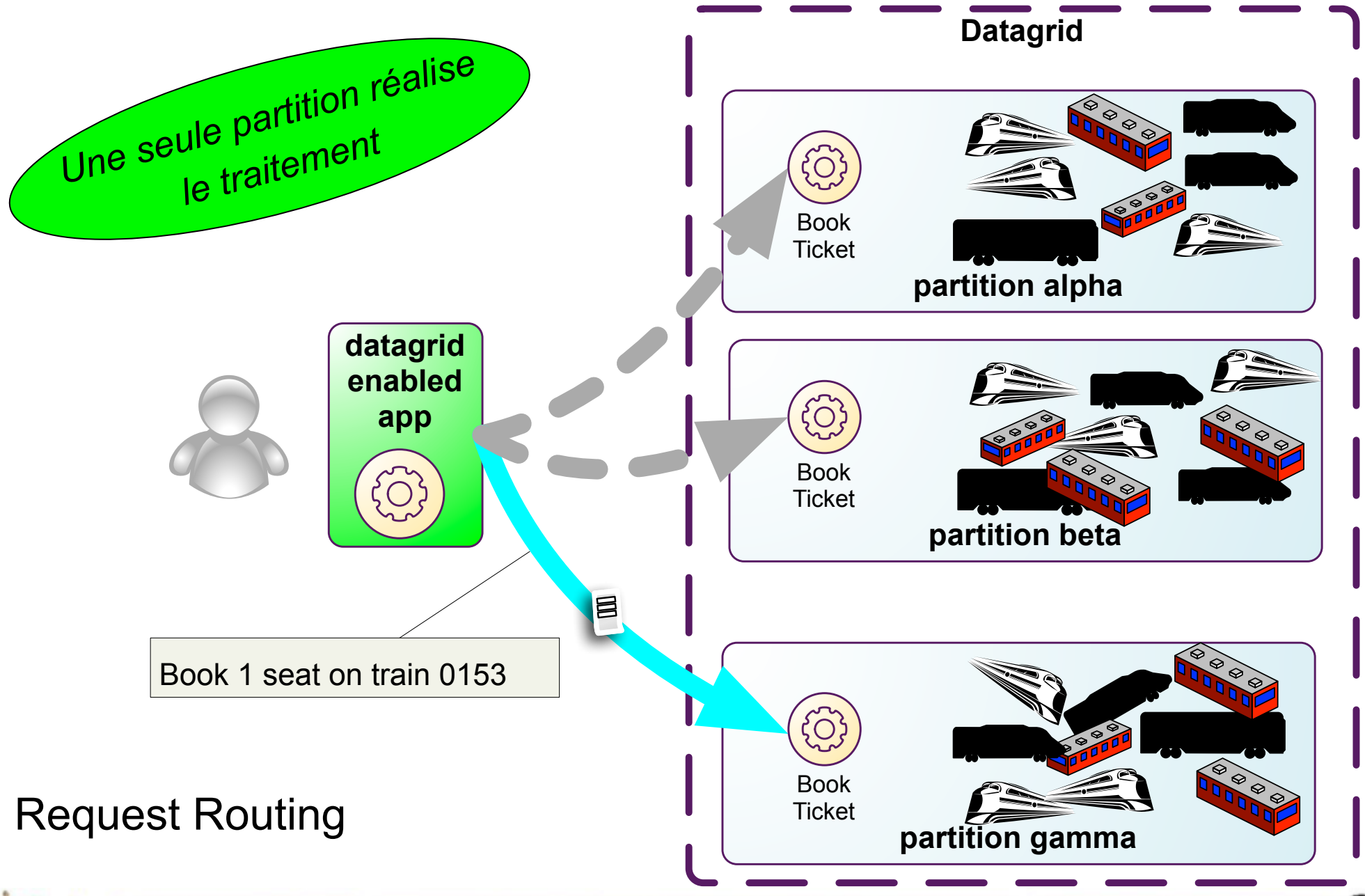
partition beta

Search trains



partition gamma

Les Design patterns



Request Routing

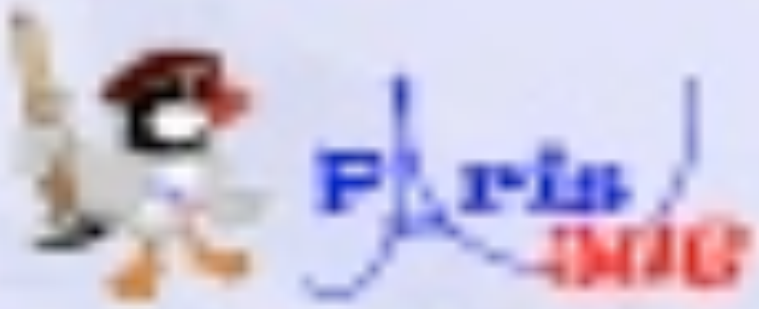
Transactions et ACIDité

Mais et l' ACID dans tout ça ?

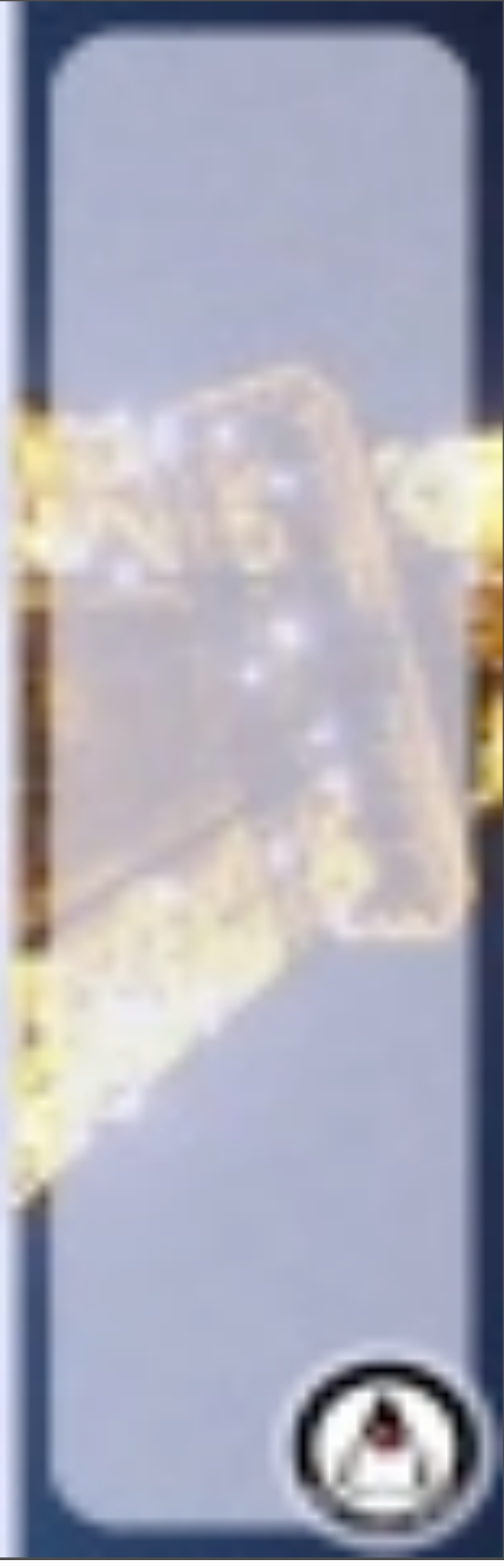
Théoriquement possible (Transaction distribuée, 2 phases commit)

Supporté par certains outils (GigaSpaces)

Est-ce vraiment une bonne chose ?

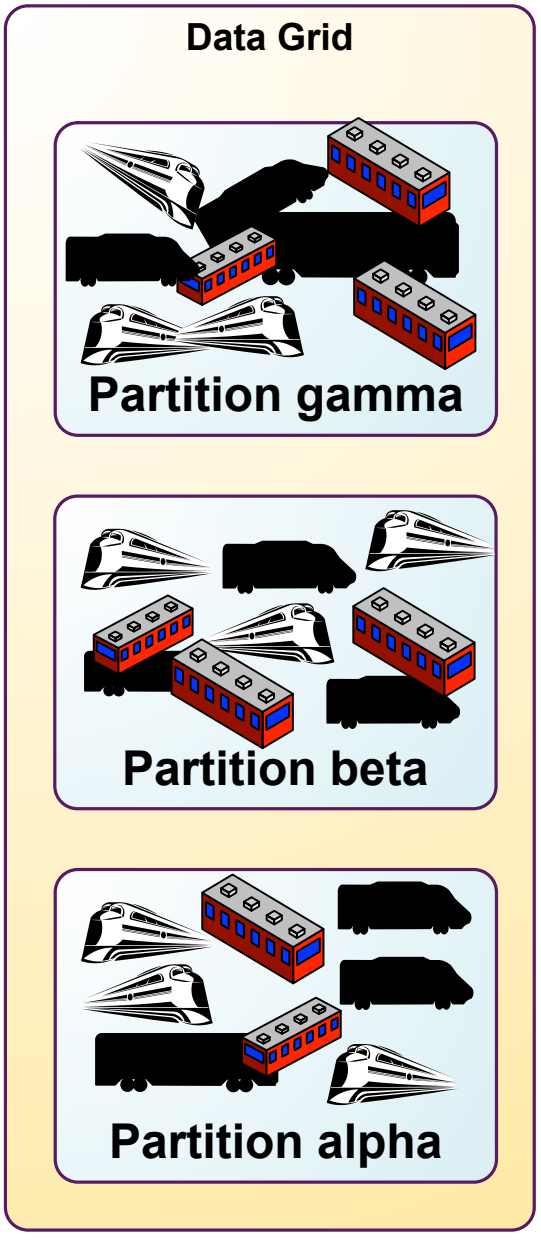


Interactions avec la base de données



www.parisjug.org

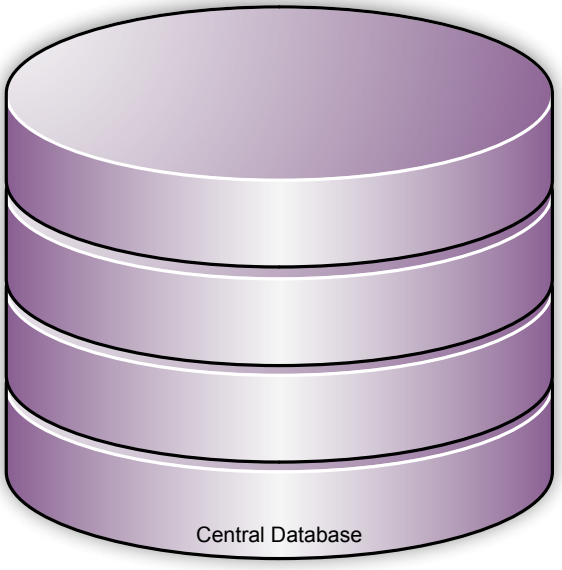
Interactions avec la BDD



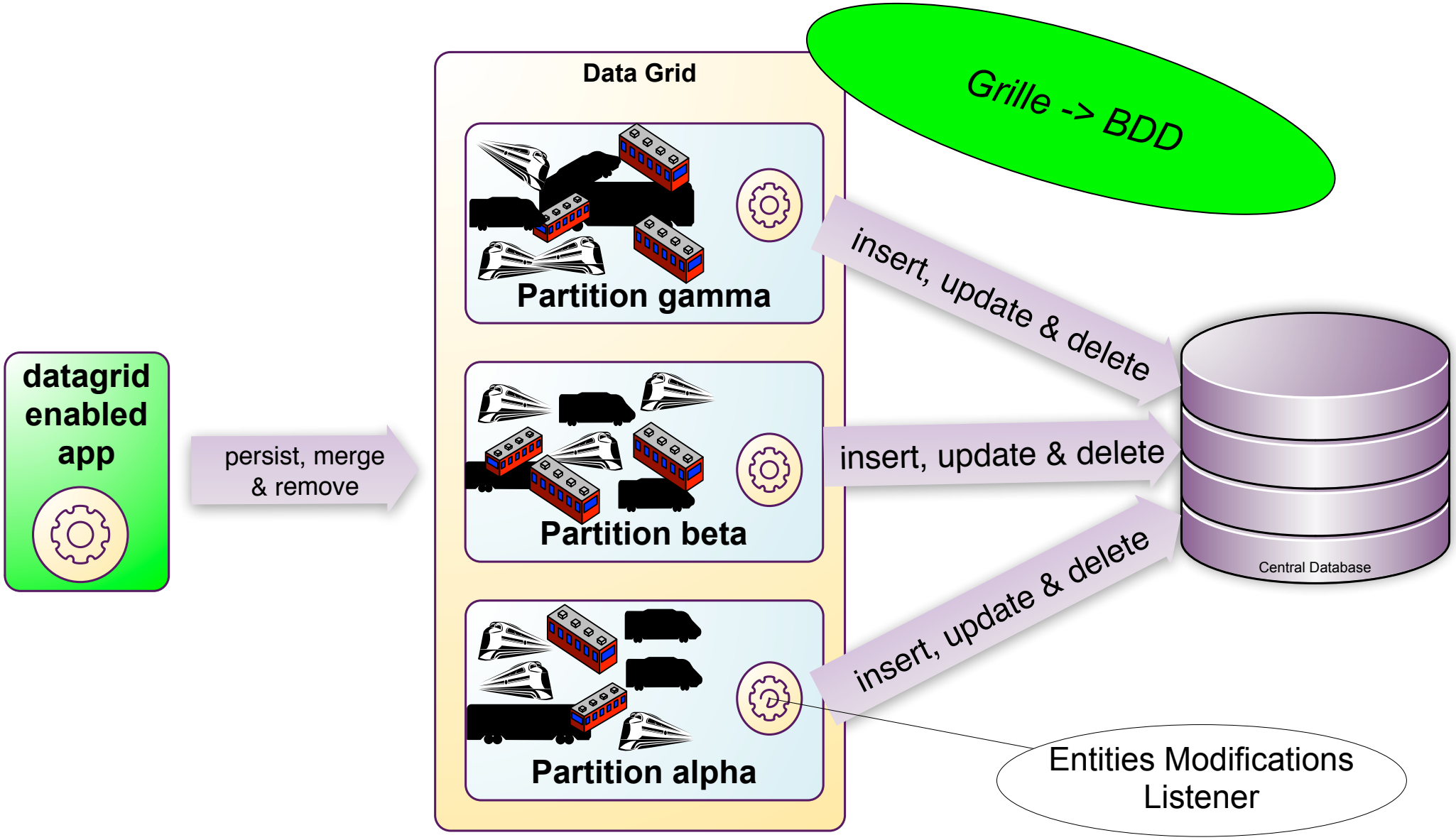
Comment synchroniser ?

insert, update & delete

Select (new, modified, deleted)



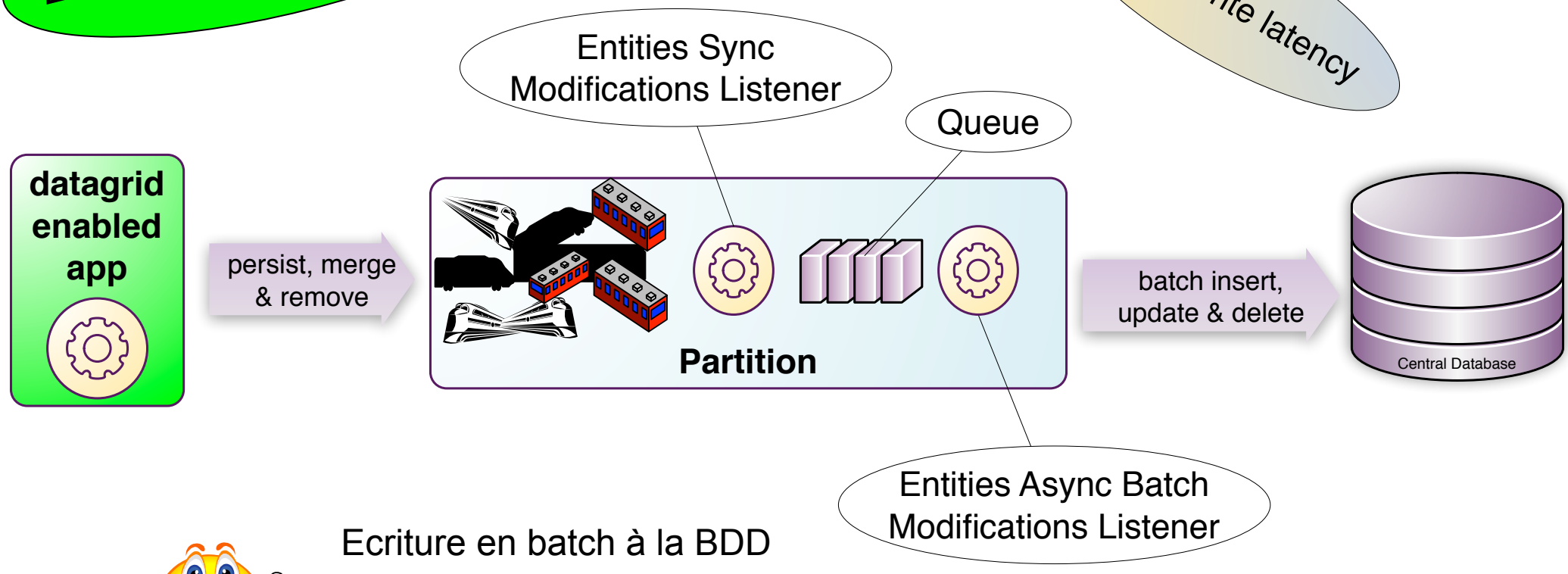
Interactions avec la BDD



Interactions avec la BDD

Ecriture en batch sur la BDD

Cache write latency



Ecriture en batch à la BDD

Supporte l'indisponibilité de la BDD

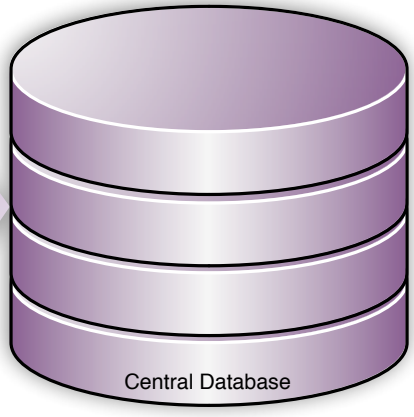
Hautement disponible par la réplication de la *queue*

Interactions avec la BDD

Comment détecter les changements de la BDD ?

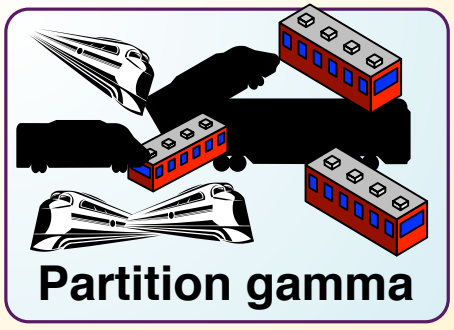
Non datagrid enabled app

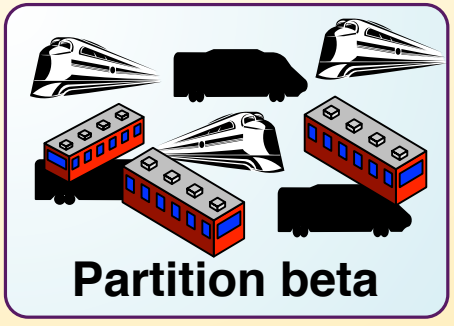
insert, update & delete

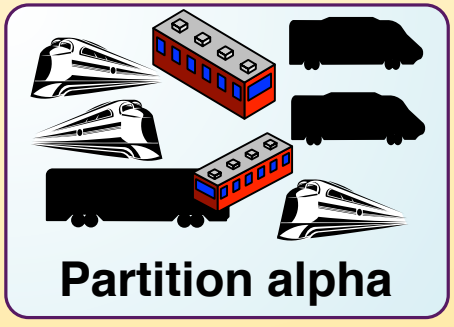


create, merge & remove

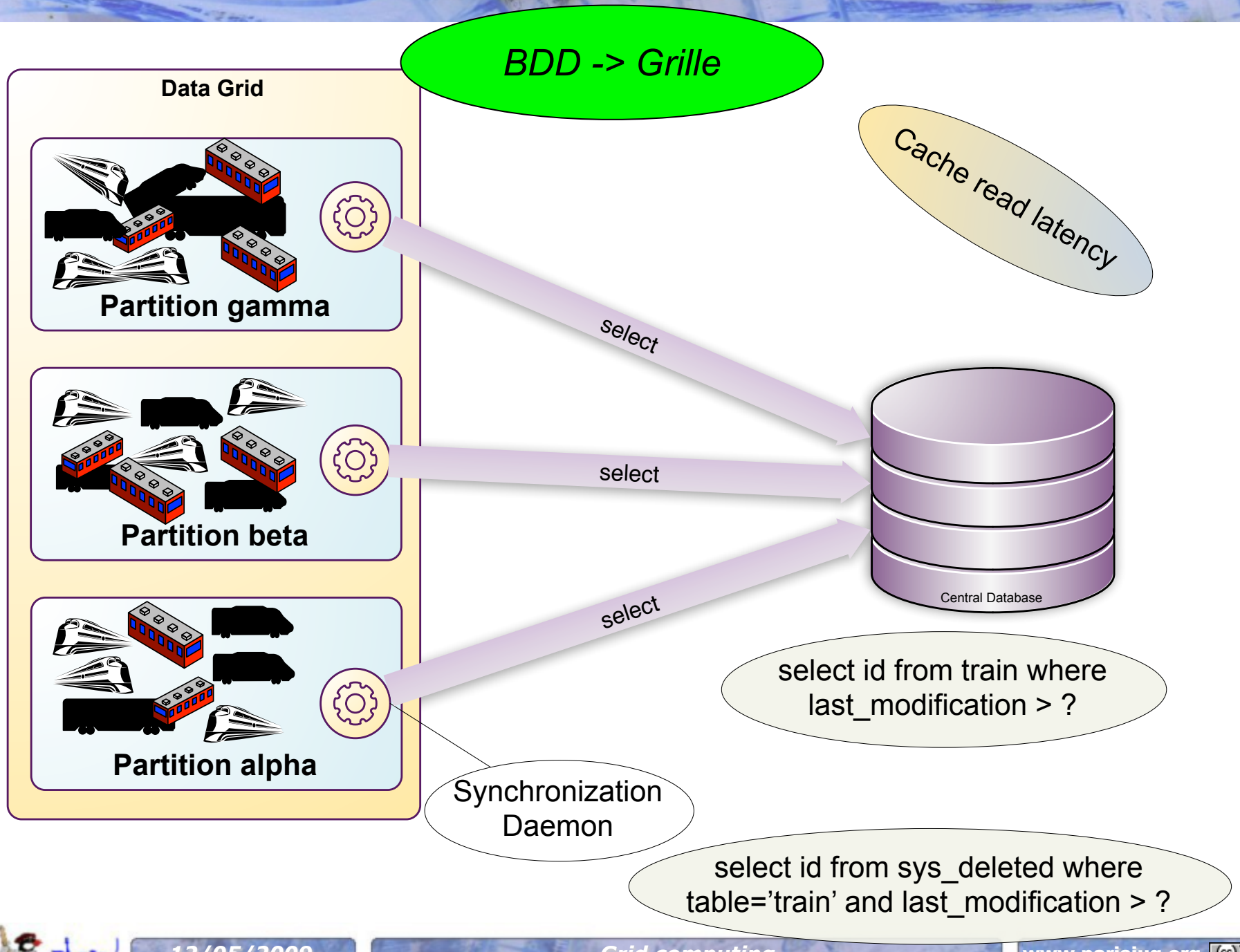
Data Grid







Interactions avec la BDD



Interactions avec la BDD

System Change Number

- **Sur chaque table, une colonne système porte un numéro croissant de la dernière transaction**
- **Utilisé pour les checkpoints et la réplication**
- **Support natif dans :**
  - Oracle: introduit en10G, le SCN est un nombre similaire au numéro de révision Subversion
  - DB2: introduit en 9.5, le SCN est un timestamp
- **Equivalent manuel : une colonne timestamp**
- **La liste des lignes supprimées est stockée dans une table système**

Interactions avec la BDD

Les API de loaders

- **Lire en base les données manquantes**
  - `List<entity> get(List<id>)`
- **Persister en base les modifications**
  - `void persist(List<type, entity>)`
  - type : insert, update, delete
- **Bulk load depuis la base au démarrage de la grille**
  - `preload(grid)`

/!\ les opérations doivent être multi entités pour tenir la charge !

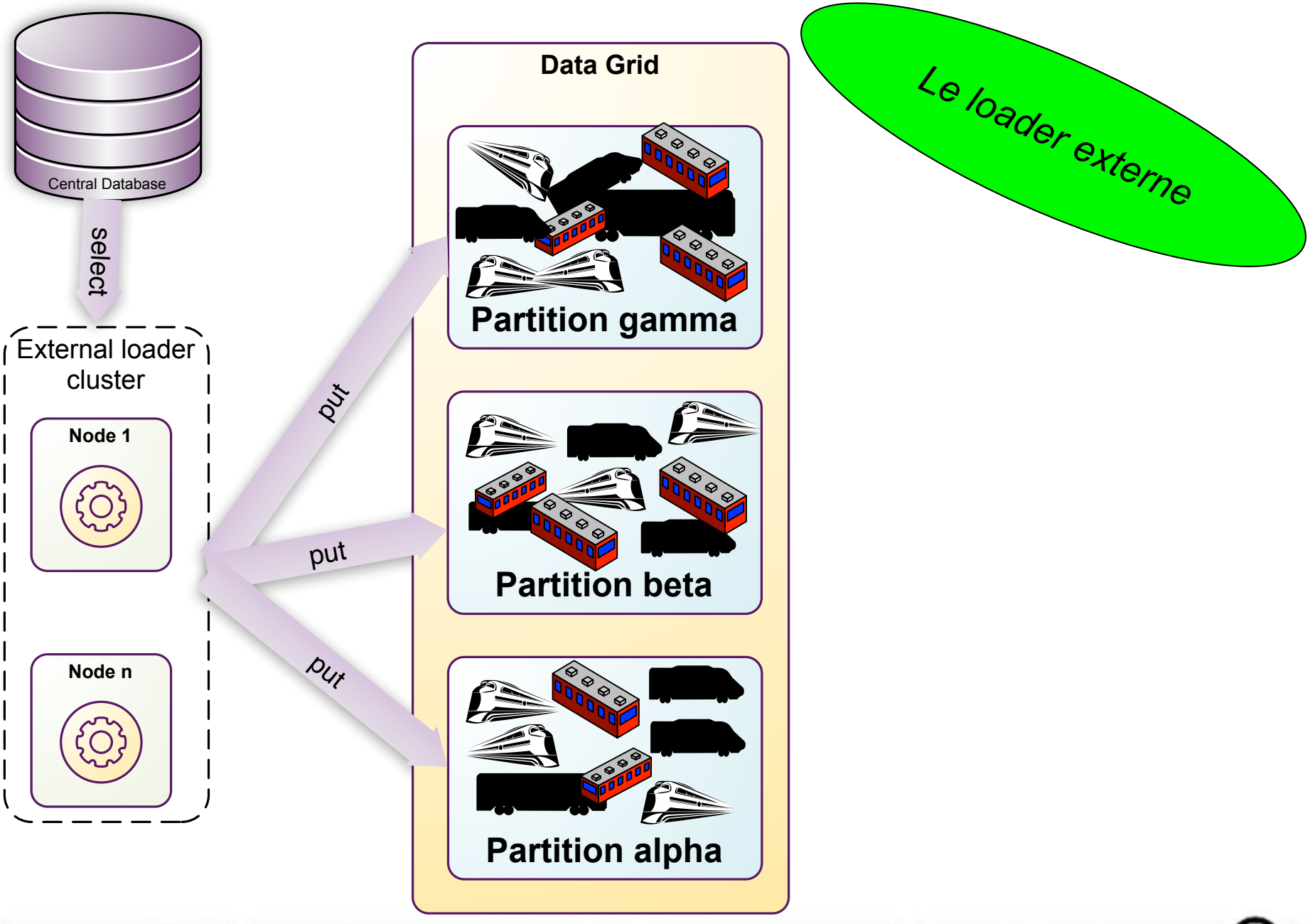
Interactions avec la BDD

Loaders internes :

- **Pros**
  - L'accès à la base est regroupé dans la grille => cohesiveness
  - La charge d'accès à la base est répartie sur les partitions
  - Write behind permet de *batcher* les accès à la base
- **Cons**
  - Comment un loader peut-il ne charger que les données de sa partition ?

« Je suis le loader de la partition beta, comment puis-jet interroger la base sur les seules données gérées par la partition beta ? Je dois connaitres l'algorithme de partionnement »

Interactions avec la BDD



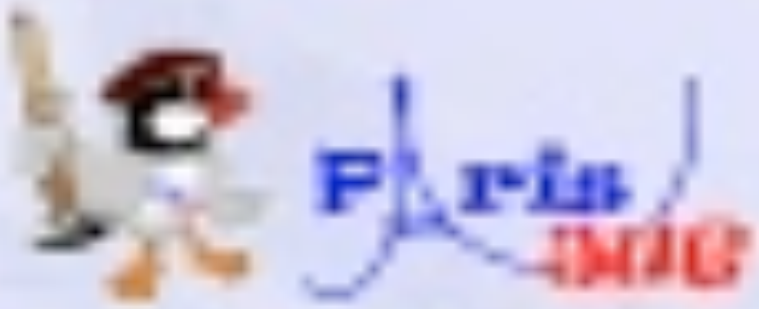
Interactions avec la BDD

Loaders externes :

- **Pros**
  - Simple, le partitionnement est réalisé par la grille
- **Cons**
  - Le loader externe devient le goulet d'étranglement des performances
  - Le code d'accès à la base est disséminé entre la grille et le loader externe => loss of cohesiveness

Et les batchs ?

Une grille de données peut écourter les nuits batch en parallélisant les traitements.



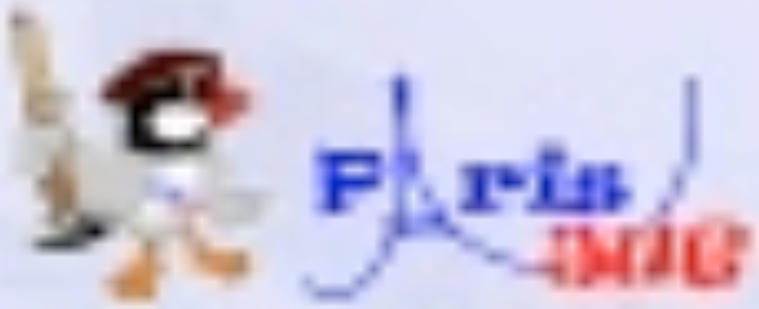
Cloud, Grid, XTP et les autres

www.parisjug.org

Les acteurs

Les acteurs

- **Distributed HashTable (DHT), Key Value Store**
  - SimpleDB, DataStore / BigTable / Hypertable, Memcached, Project Voldermort, CouchDB
- **Compute Grid**
  - GridGain
- **In Memory Data Grid**
  - GigaSpaces, Oracle Coherence, Websphere eXtreme Scale.
- **Distributed File system**
  - Hadoop



Questions / Réponses

www.parisjug.org

Sponsors

Merci de votre attention!

www.parisjug.org

Licence

- http://creativecommons.org/licenses/by-nc-sa/2.0/fr/