Stockage et analyse temps réel d'événements avec Riak chez Booking.com

Damien Krotkine

Damien Krotkine

- Ingénieur chez Booking.com

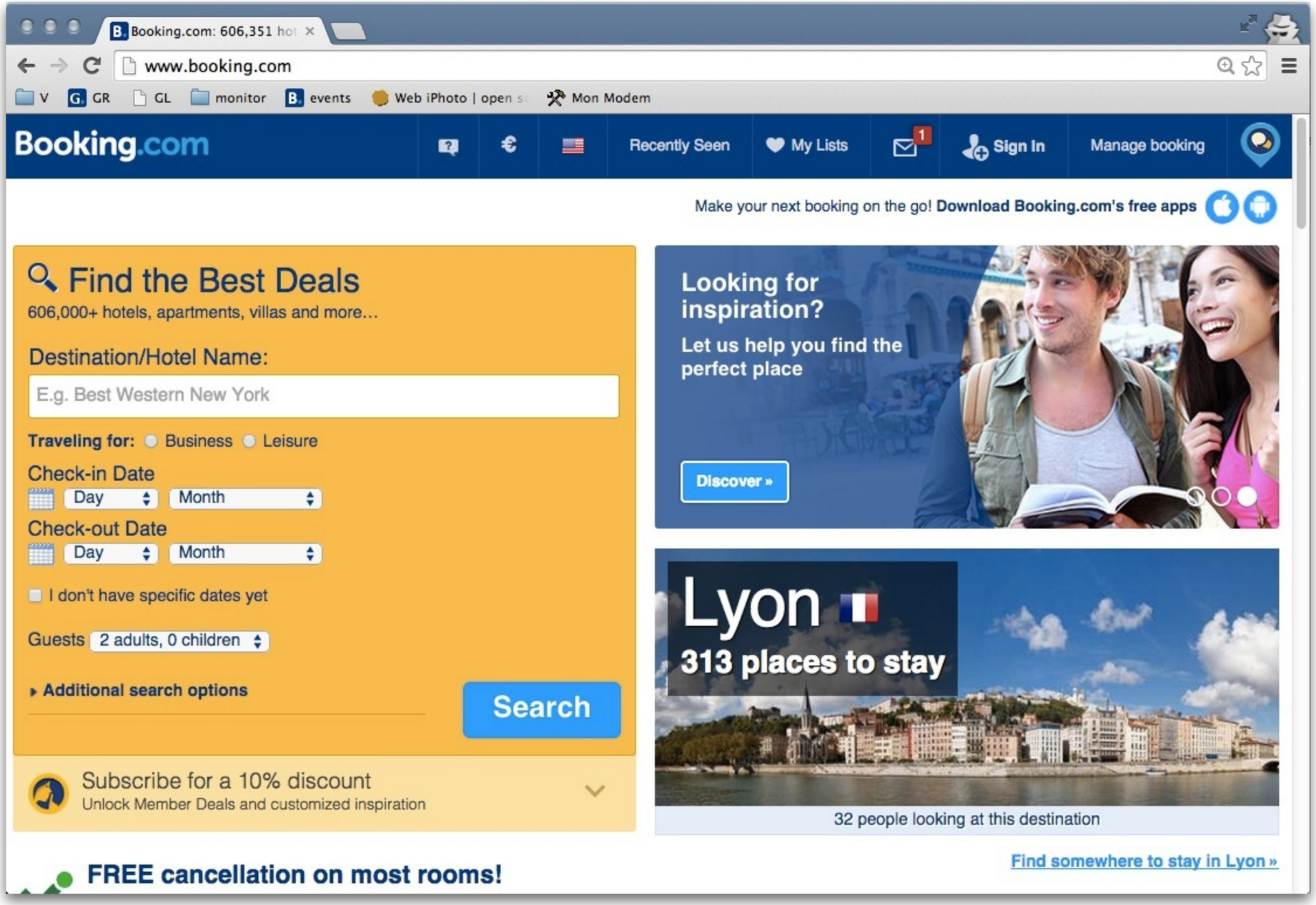
- github.com/dams

- @damsieboy

- dkrotkine

- 800,000 chambres reservées par jour

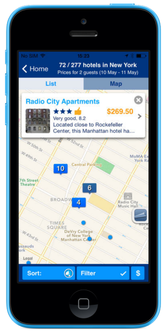
INTRODUCTION

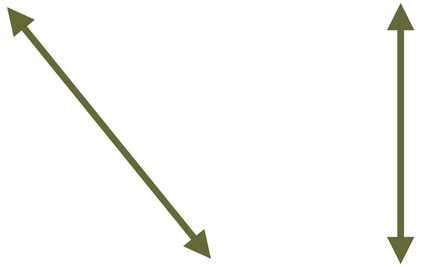

Booking.com: 606,351 hot
www.booking.com
V
GR
GL
monitor
events
Web iPhoto | open s
Mon Modem
Booking.com
Recently Seen
My Lists
1
Sign In
Manage booking
Make your next booking on the go! Download Booking.com's free apps
Find the Best Deals
606,000+ hotels, apartments, villas and more…
Destination/Hotel Name:
E.g. Best Western New York
Traveling for: Business Leisure
Check-in Date
Day
Month
Check-out Date
Day
Month
I don't have specific dates yet
Guests 2 adults, 0 children
Additional search options
Search
Looking for inspiration?
Let us help you find the perfect place
Discover »
Lyon
313 places to stay
32 people looking at this destination
Subscribe for a 10% discount
Unlock Member Deals and customized inspiration
FREE cancellation on most rooms!
Find somewhere to stay in Lyon »


Booking.com: 606,351 hot
www.booking.com
V
GR
GL
monitor
events
Web iPhoto | open s
Mon Modem
Booking.com
Recently Seen
My Lists
Sign In
Manage booking
Make your next booking on the go! Download Booking.com's free apps
Find the Best Deals
606,000+ hotels, apartments, villas and more...
Destination/Hotel Name:
E.g. Best Western New York
Traveling for: Business Leisure
Check-in Date
Day
Month
Check-out Date
Day
Month
I don't have specific dates yet
Guests 2 adults, 0 children
Additional search options
Search
Subscribe for a 10% discount
Unlock Member Deals and customized inspiration
FREE cancellation on most rooms!
Looking for inspiration?
Let us help you find the perfect place
Discover »
Lyon
313 places to stay
32 people looking at this destination
Find somewhere to stay in Lyon »

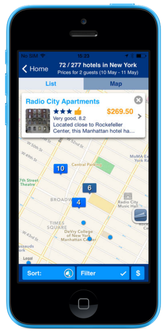

Booking.com
Find the Best Deals
Destination/Hotel Name:
E.g. Best Western New York
Check-in Date
Checkout Date
Search
Looking for inspiration?
Let us help you find the perfect place
Lyon
313 places to stay
FREE cancellation on most rooms!

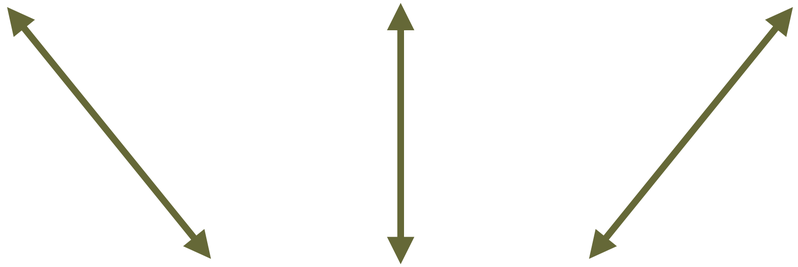

Booking.com
Find the Best Deals
Looking for inspiration?
Let us help you find the perfect place
Lyon
313 places to stay
Search
FREE cancellation on most rooms!

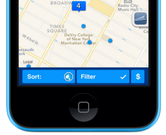

Home
72 / 277 hotels in New York
List
Map
Radio City Apartments
$269.50
Sort:
Filter




Booking.com
Find the Best Deals
Looking for inspiration?
Let us help you find the perfect place
Lyon
313 places to stay
Search
FREE cancellation on most rooms!


Home
72 / 277 hotels in New York
List
Map
Radio City Apartments
$269.50
Sort:
Filter



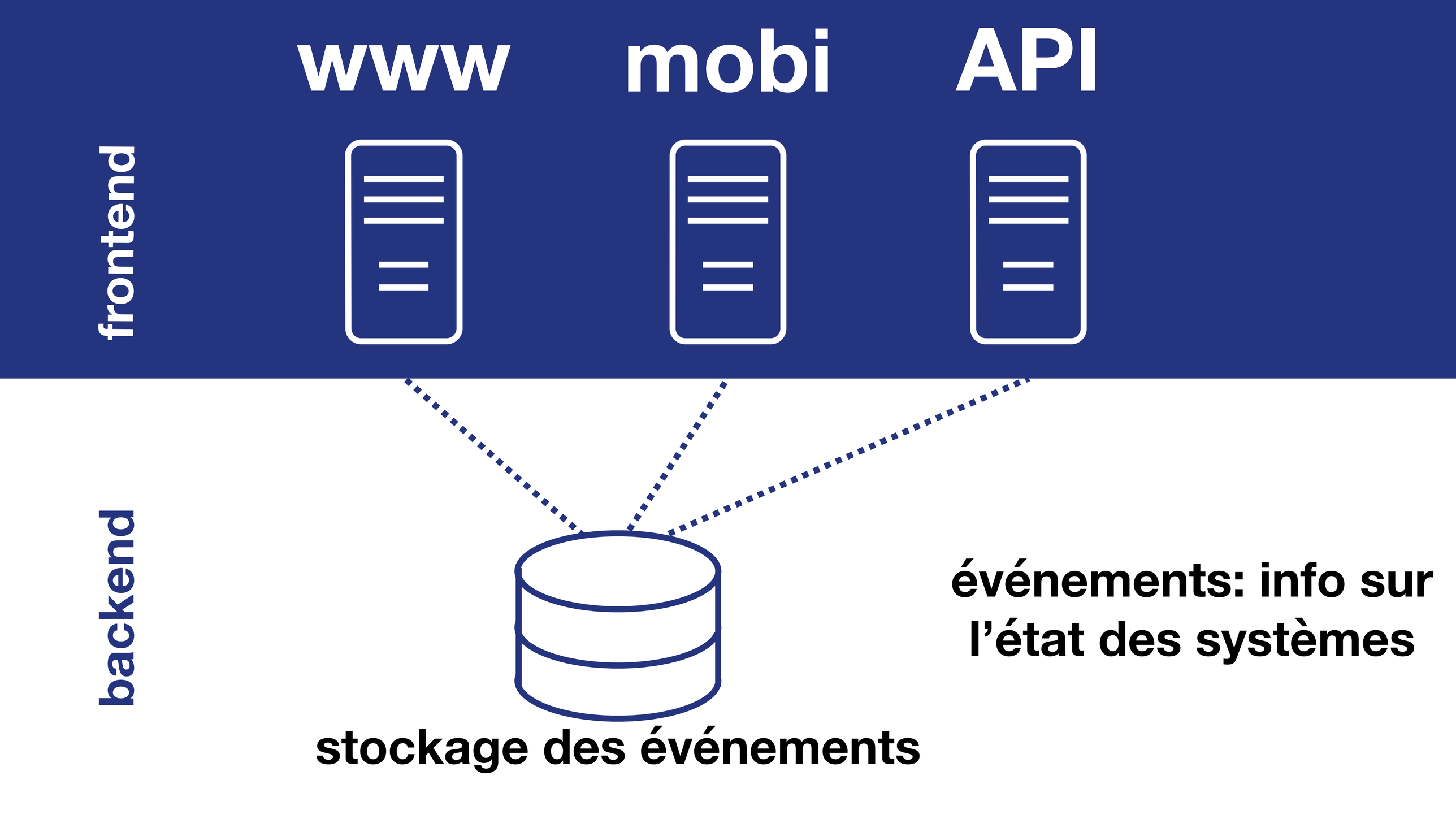


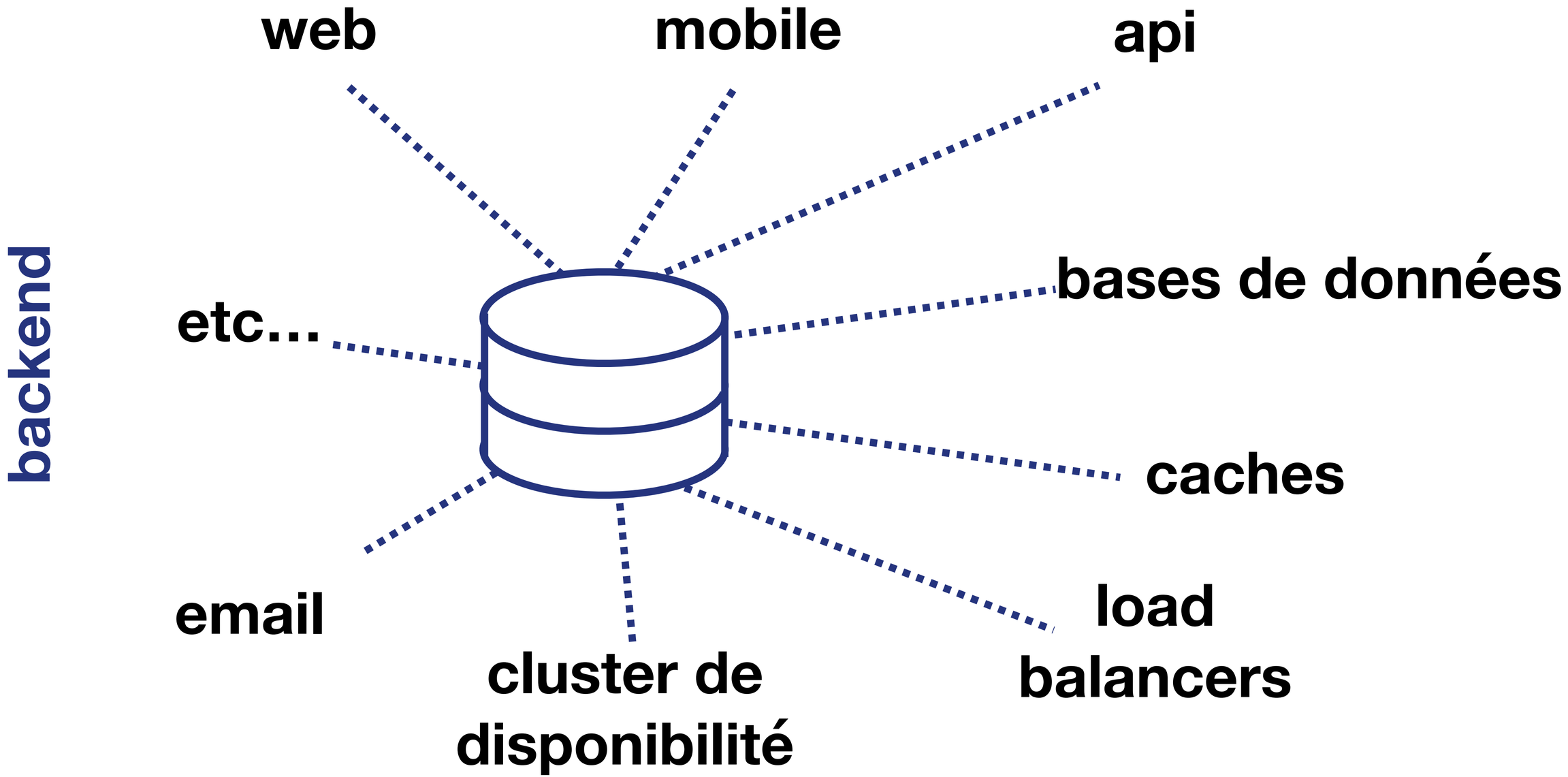
frontend

www

mobi

API


www
mobi
API
frontend
backend
stockage des événements
événements: info sur
l’état des systèmes


backend
web
mobile
api
bases de données
etc…
caches
email
cluster de
disponibilité
load
balancers

QU’EST-CE QU’UN EVENEMENT ?

STRUCTURE D’UN EVENEMENT

- **Information** sur un **système**
- Données
- HashMap profond
- Timestamp
- Data Center + Type + Sous-type
- Le reste: données spécifiques
- **Sans Schema**

```
{ timestamp => 12345,
  type => 'WEB',
  subtype => ‘app',
  dc => 1,
  action => { is_normal_user => 1,
              pageview_id => '188a362744c301c2',
              # ...
            },
  tuning => { the_request => 'GET /display/...'
              bytes_body => 35,
              wallclock => 111,
              nr_warnings => 0,
              # ...
            },
  # ...
}
```

```
{ type => 'FAV',
  subtype => 'fav',
  timestamp => 1401262979,
  dc => 1,
  tuning => {
      flatav => {
          cluster => '205',
          sum_latencies => 21,
          role => 'fav',
          num_queries => 7
      }
  }
}
```

PROPRIETES DU FLUX D’EVENEMENTS

- Lecture seule
- Sans schema
- Flux continu, ordonné, temporel
- 15 K événements par sec
- **1.25 Milliards** événements par jour
- pics à 70 MB/s, min 25MB/s
- 100 GB par heure

UTILISATION

DEFINIR LES BESOINS

- Avant d’aborder le stockage
- Penser à l’utilisation

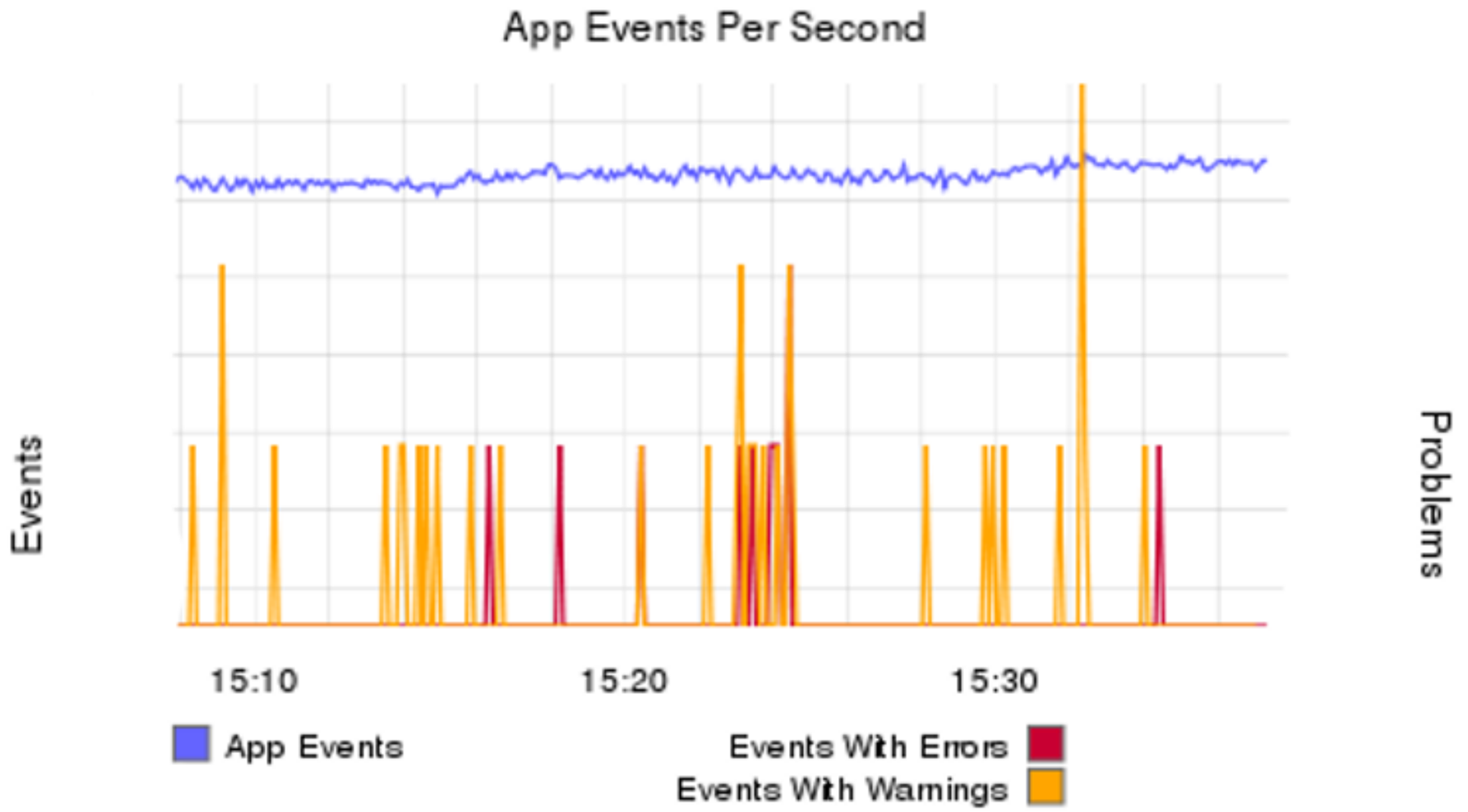
UTILISATION

1. GRAPHS
2. PRISE DE DECISION
3. ANALYSE COURT TERME
4. A/B TESTING

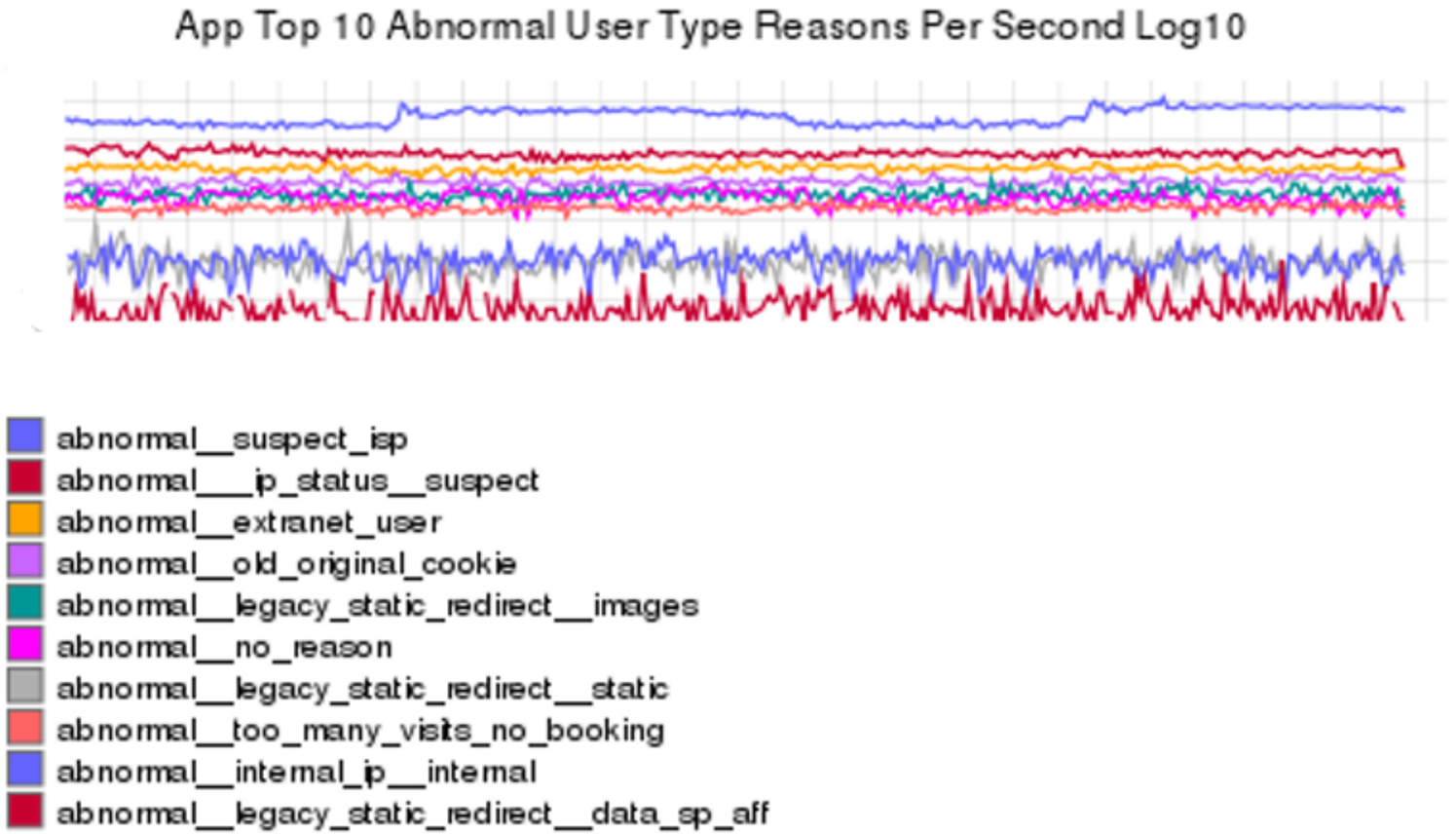
GRAPHS

- Graph en temps réel (lag = quelques secondes)
- Graph pour le plus de systèmes possibles
- Etat de santé général de la plateforme

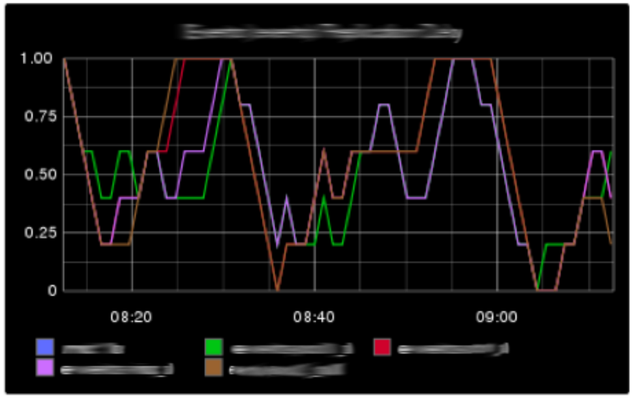
GRAPHS

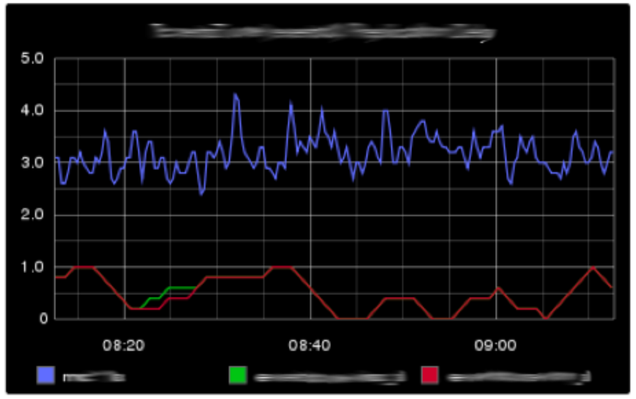

App Events Per Second
Events
Problems
15:10
15:20
15:30
App Events
Events With Errors
Events With Warnings

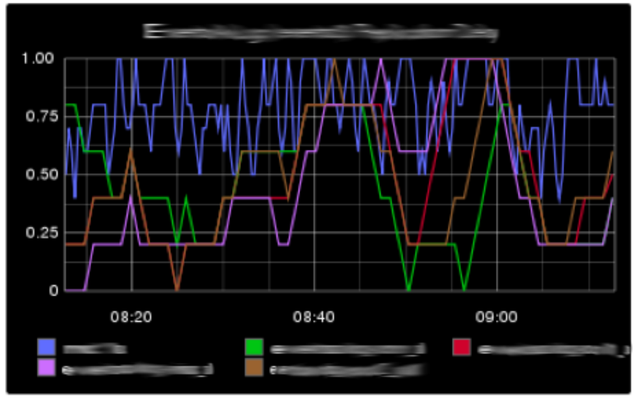
GRAPHS

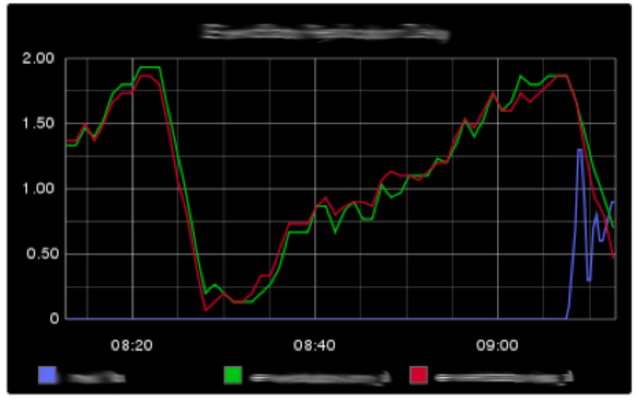

App Top 10 Abnormal User Type Reasons Per Second Log10
abnormal__suspect_isp
abnormal___ip_status__suspect
abnormal__extranet_user
abnormal__old_original_cookie
abnormal__legacy_static_redirect__images
abnormal__no_reason
abnormal__legacy_static_redirect__static
abnormal__too_many_visits_no_booking
abnormal__internal_ip__internal
abnormal__legacy_static_redirect__data_sp_aff

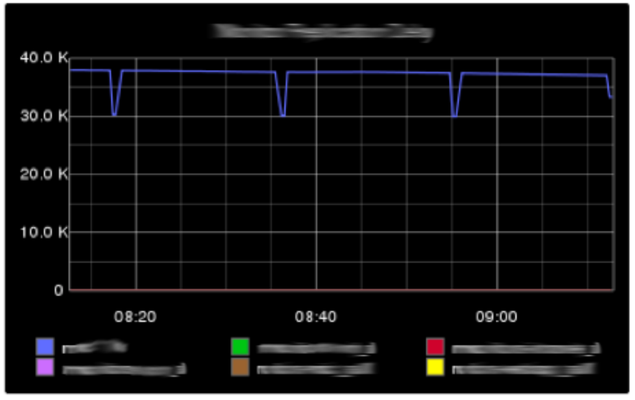
DASHBOARDS

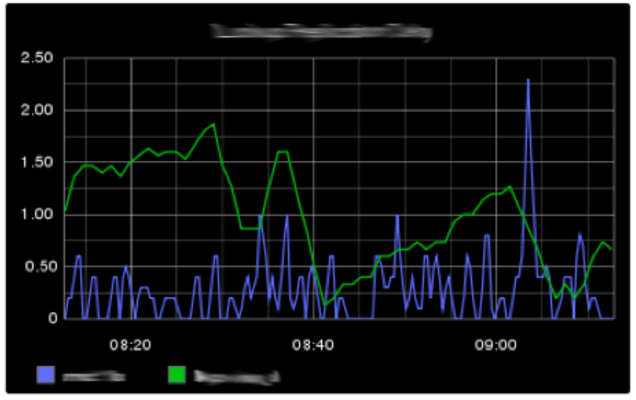

1.00
0.75
0.50
0.25
0
08:20
08:40
09:00

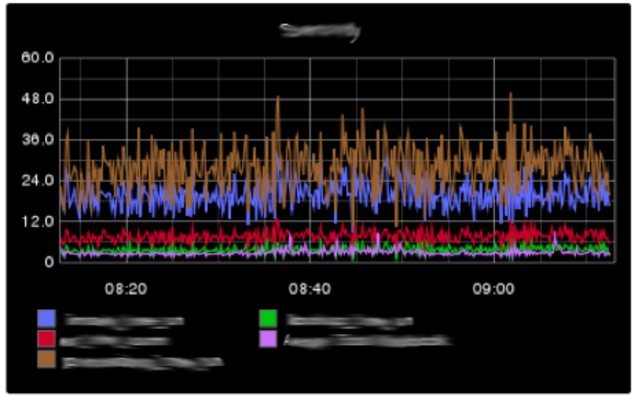

5.0
4.0
3.0
2.0
1.0
0
08:20
08:40
09:00

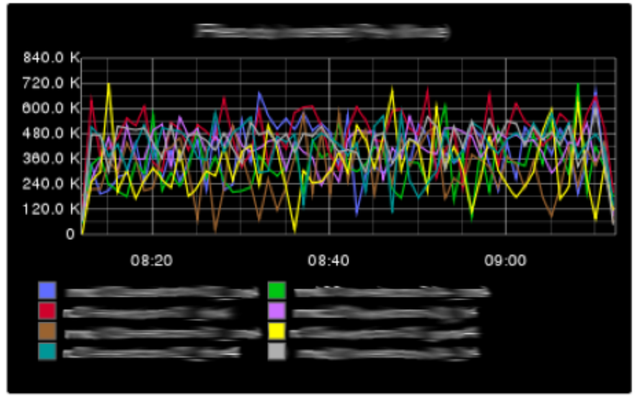

1.00
0.75
0.50
0.25
0
08:20
08:40
09:00

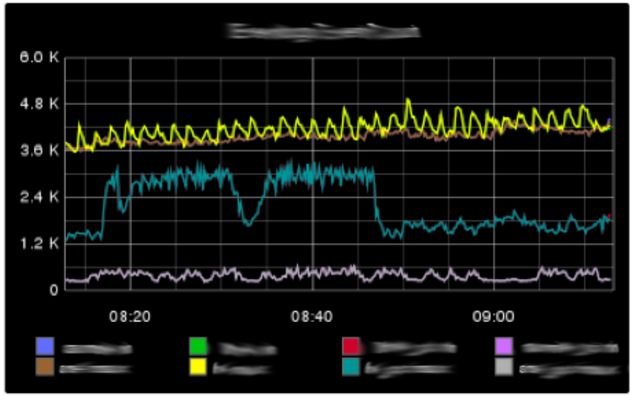

2.00
1.50
1.00
0.50
0
08:20
08:40
09:00

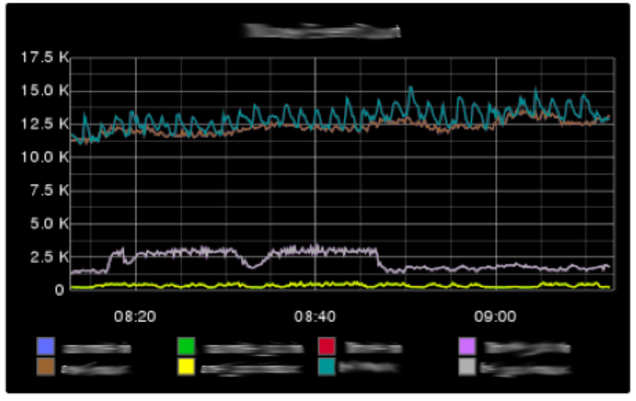

40.0 K
30.0 K
20.0 K
10.0 K
0
08:20
08:40
09:00

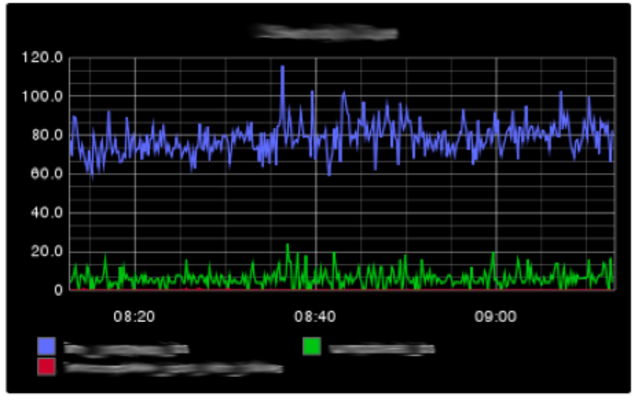

2.50
2.00
1.50
1.00
0.50
0
08:20
08:40
09:00

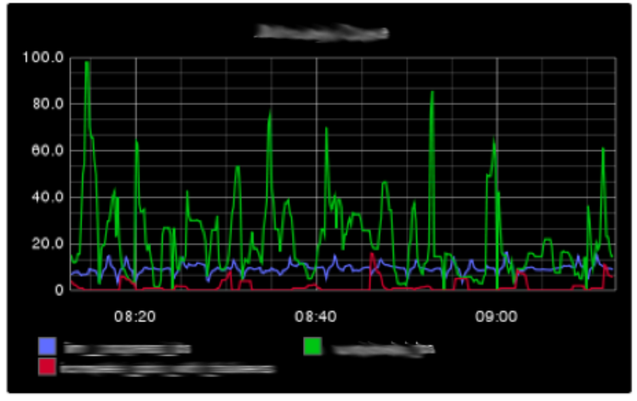

60.0
48.0
36.0
24.0
12.0
0
08:20
08:40
09:00


840.0 K
720.0 K
600.0 K
480.0 K
360.0 K
240.0 K
120.0 K
0
08:20
08:40
09:00

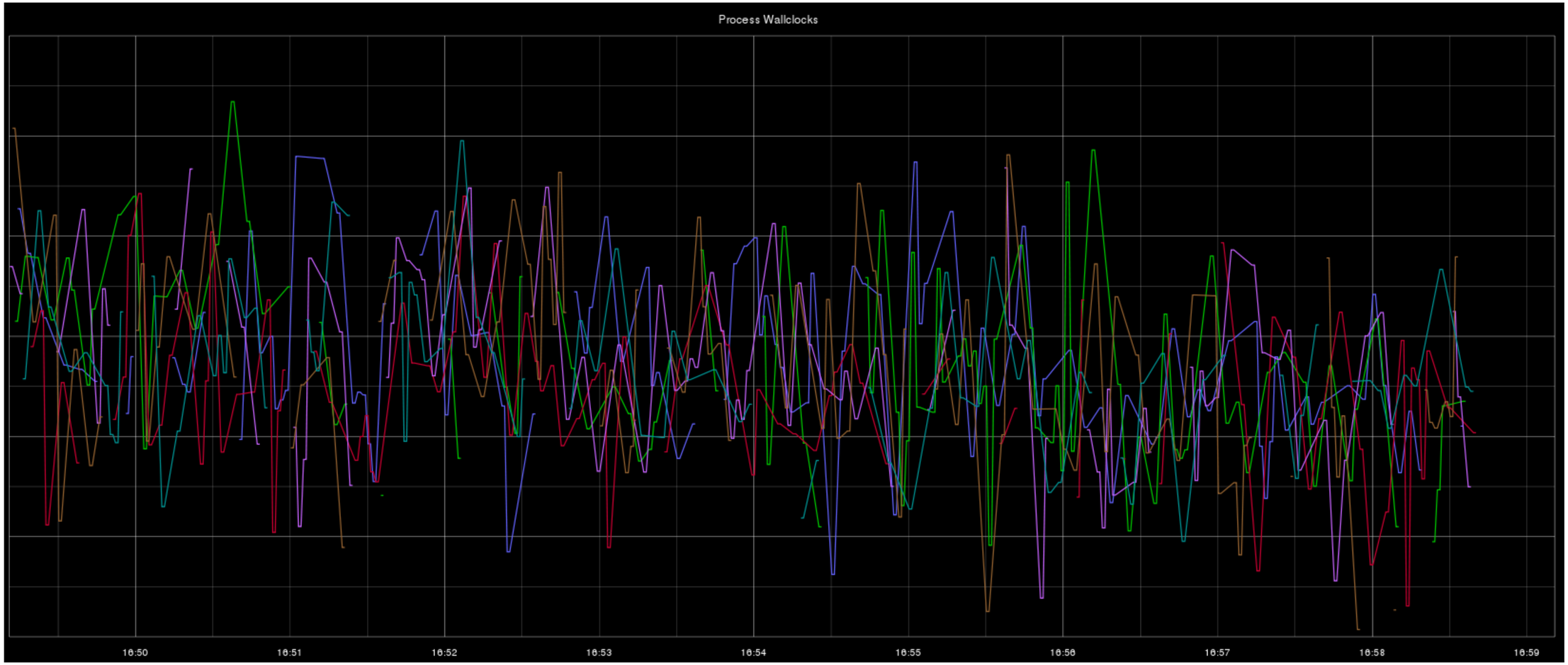

6.0 K
4.8 K
3.6 K
2.4 K
1.2 K
0
08:20
08:40
09:00


17.5 K
15.0 K
12.5 K
10.0 K
7.5 K
5.0 K
2.5 K
0
08:20
08:40
09:00


120.0
100.0
80.0
60.0
40.0
20.0
0
08:20
08:40
09:00


100.0
80.0
60.0
40.0
20.0
0
08:20
08:40
09:00

META GRAPHS


Process Wallclocks
16:50
16:51
16:52
16:53
16:54
16:55
16:56
16:57
16:58
16:59

USAGE

1. GRAPHS
2. PRISE DE DECISION
3. ANALYSE COURT TERME
4. A/B TESTING

DECISION MAKING

- Decisions Stratégiques (philosophie “use facts”)
- Court terme / long terme
- Décisions techniques / non techniques

USAGE

1. GRAPHS
2. PRISE DE DECISION
3. ANALYSE COURT TERME
4. A/B TESTING

SHORT TERM ANALYSIS

- 10 dernières secondes -> il y a 8 jours
- Déploiement de code et rollback
- Détecteur d’Anomalies

USAGE

1. GRAPHS
2. PRISE DE DECISION
3. ANALYSE COURT TERME
4. A/B TESTING

A/B TESTING

- Philosophie “use facts”
- Implique A/B testing
- Concept d’”experimentations”: Experiments
- Evénements procurent la matière première

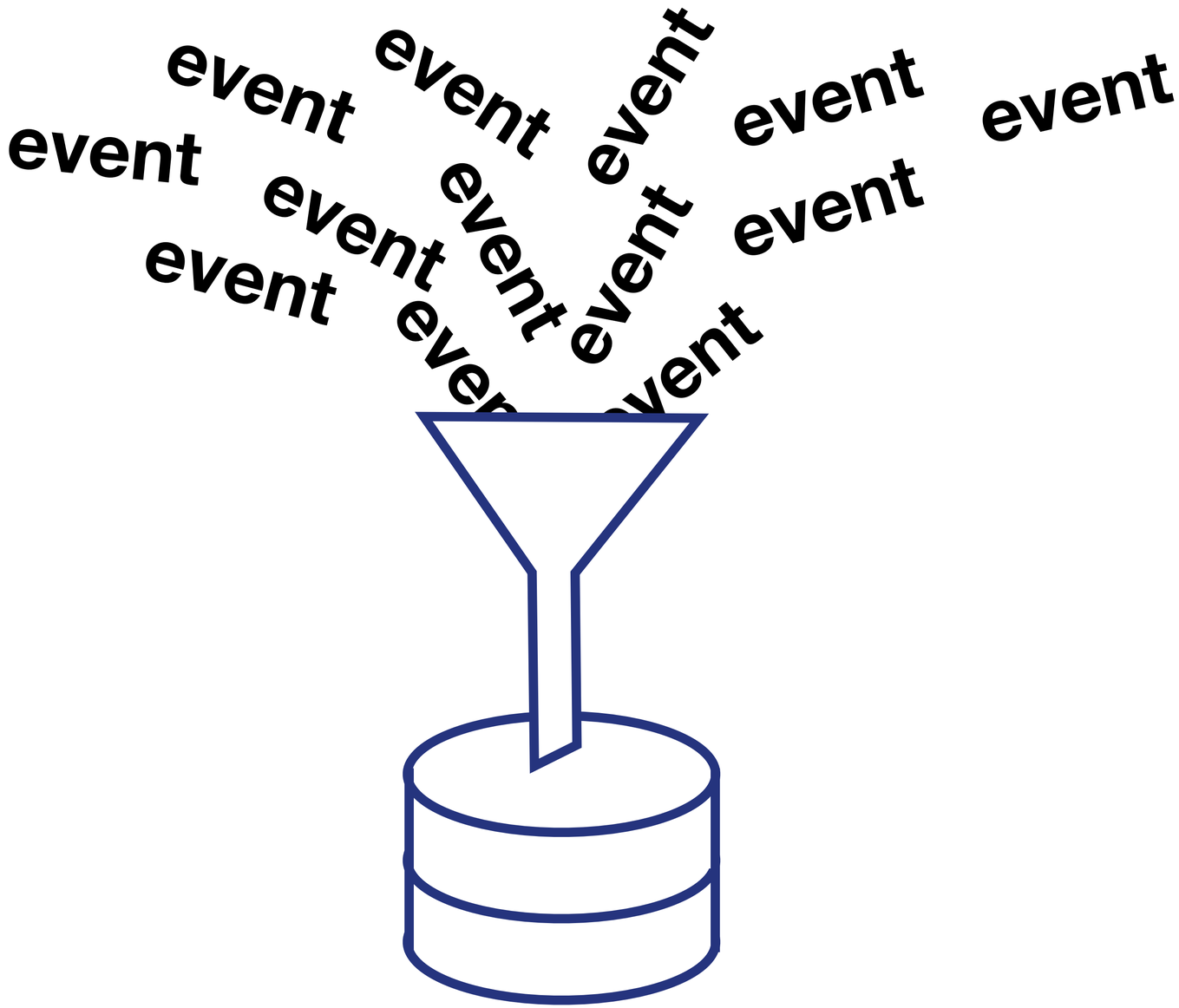
AGGREGATION D’EVENEMENTS

AGGREGATION D’EVENEMENTS

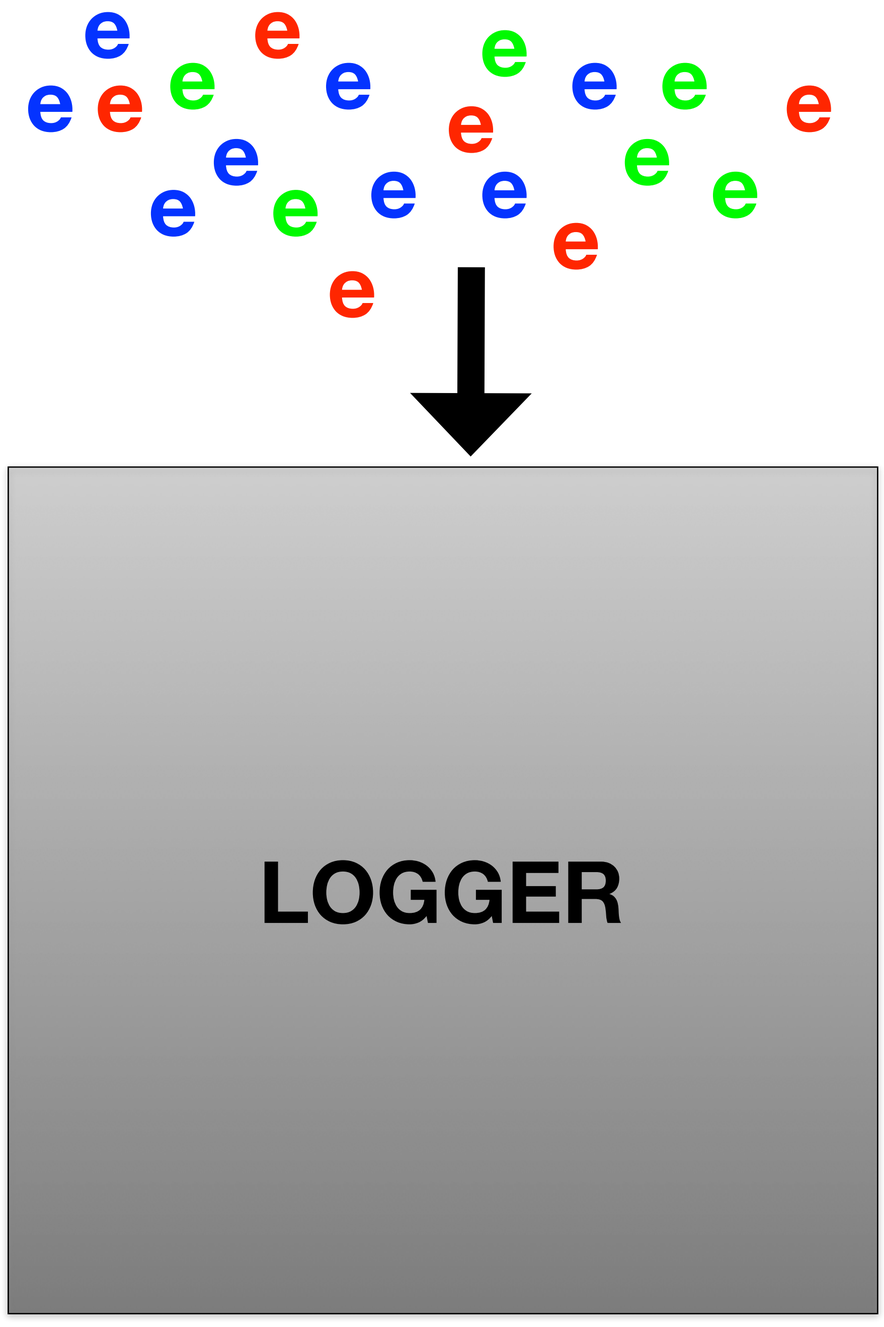
- Regrouper les événements
- Granularité nécessaire: seconde

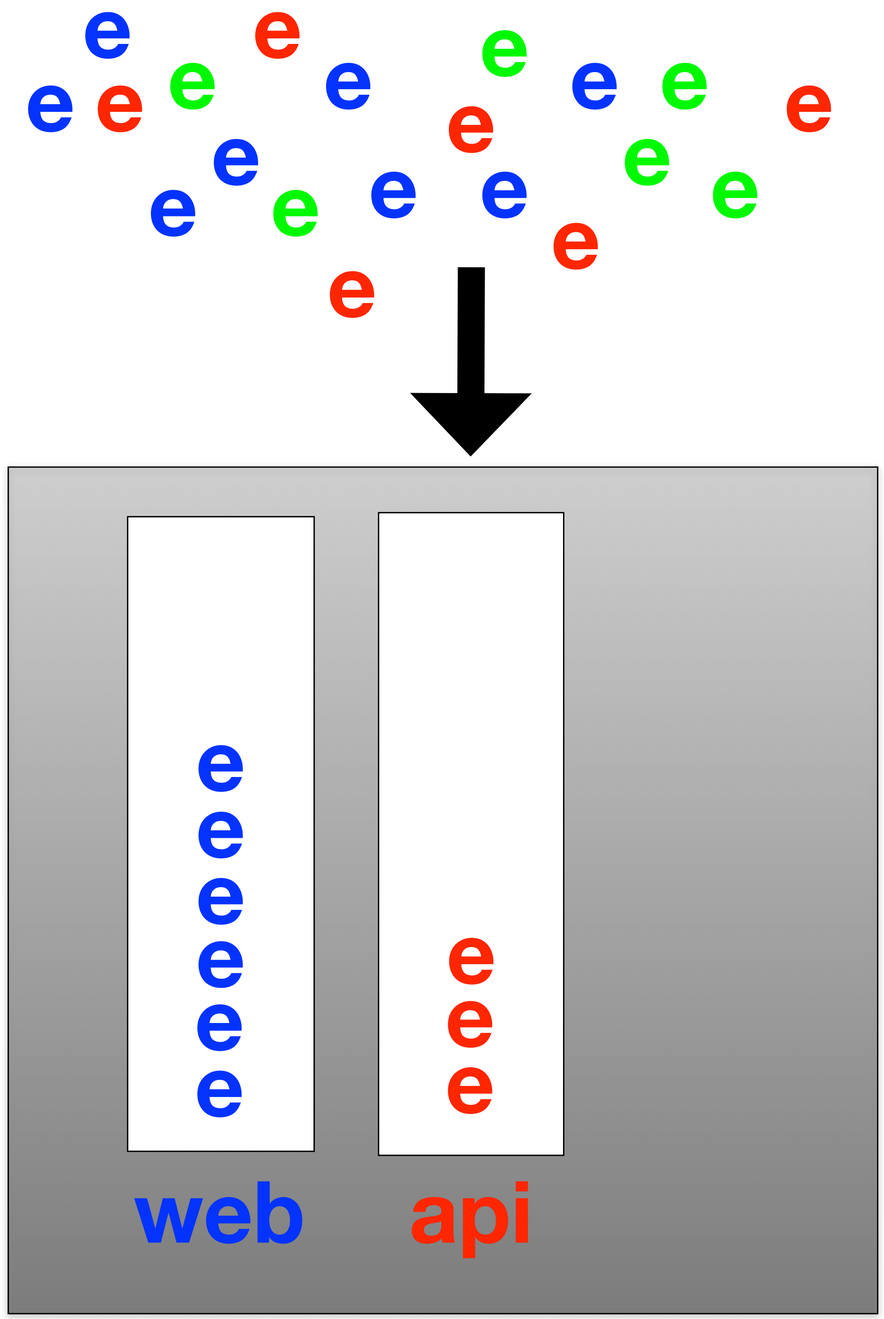
SERIALISATION

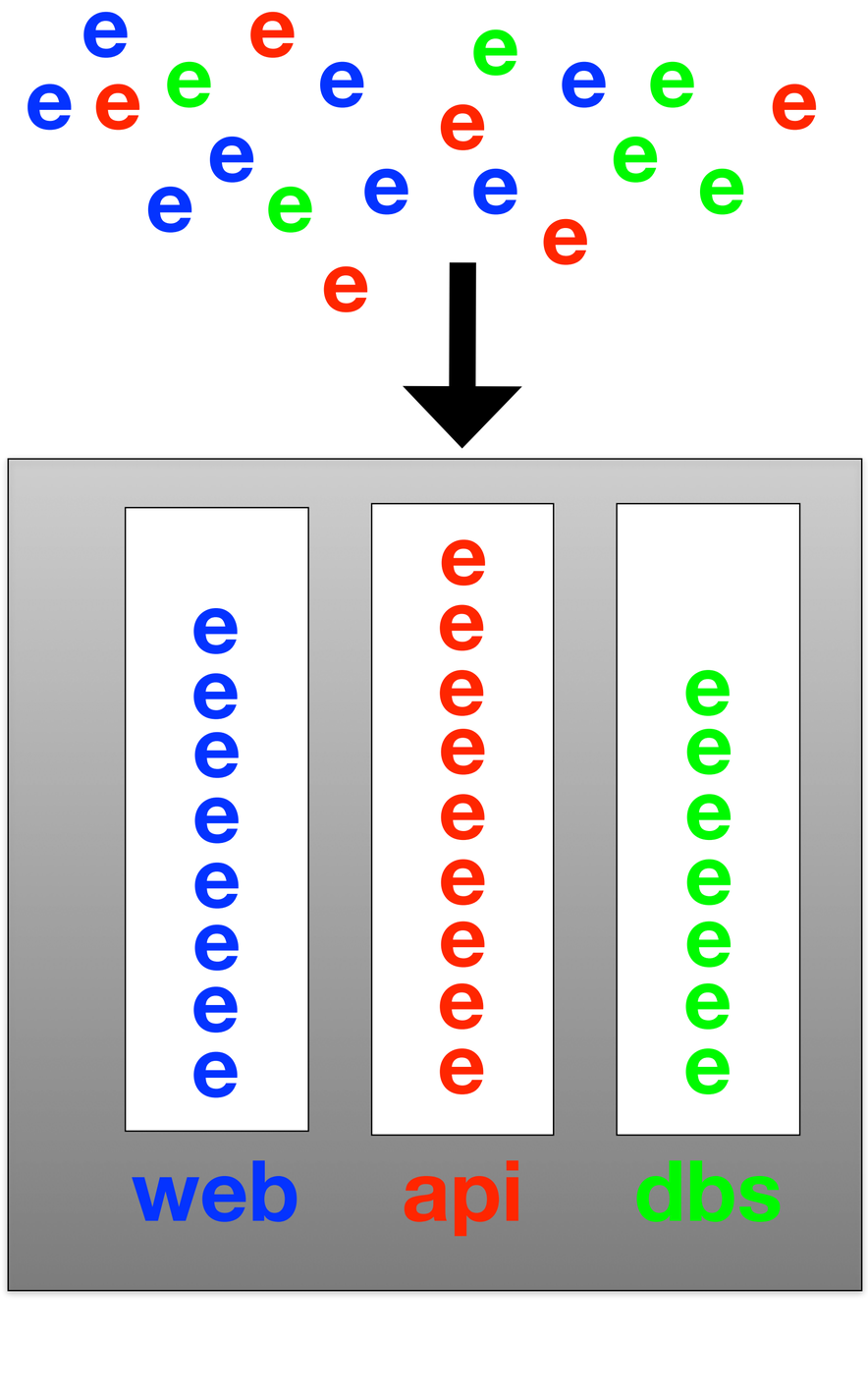
- JSON ne nous allait pas (lent, gros, manque de fonct.)
- Création de Sereal en 2012
- « *Sereal, a new, **binary** data serialization format that provides high-**performance**, **schema-less** serialization* »
- https://github.com/Sereal/Sereal

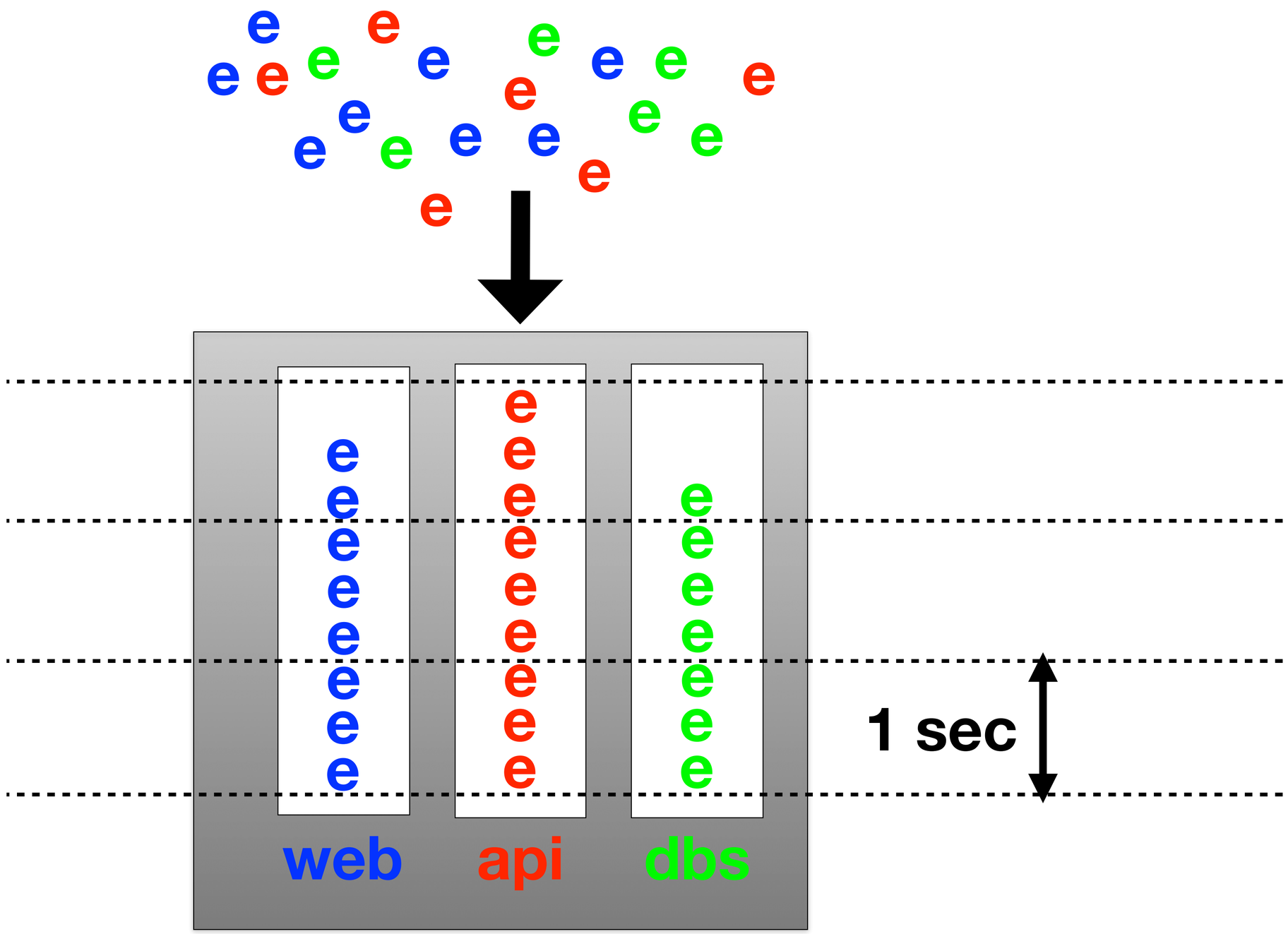


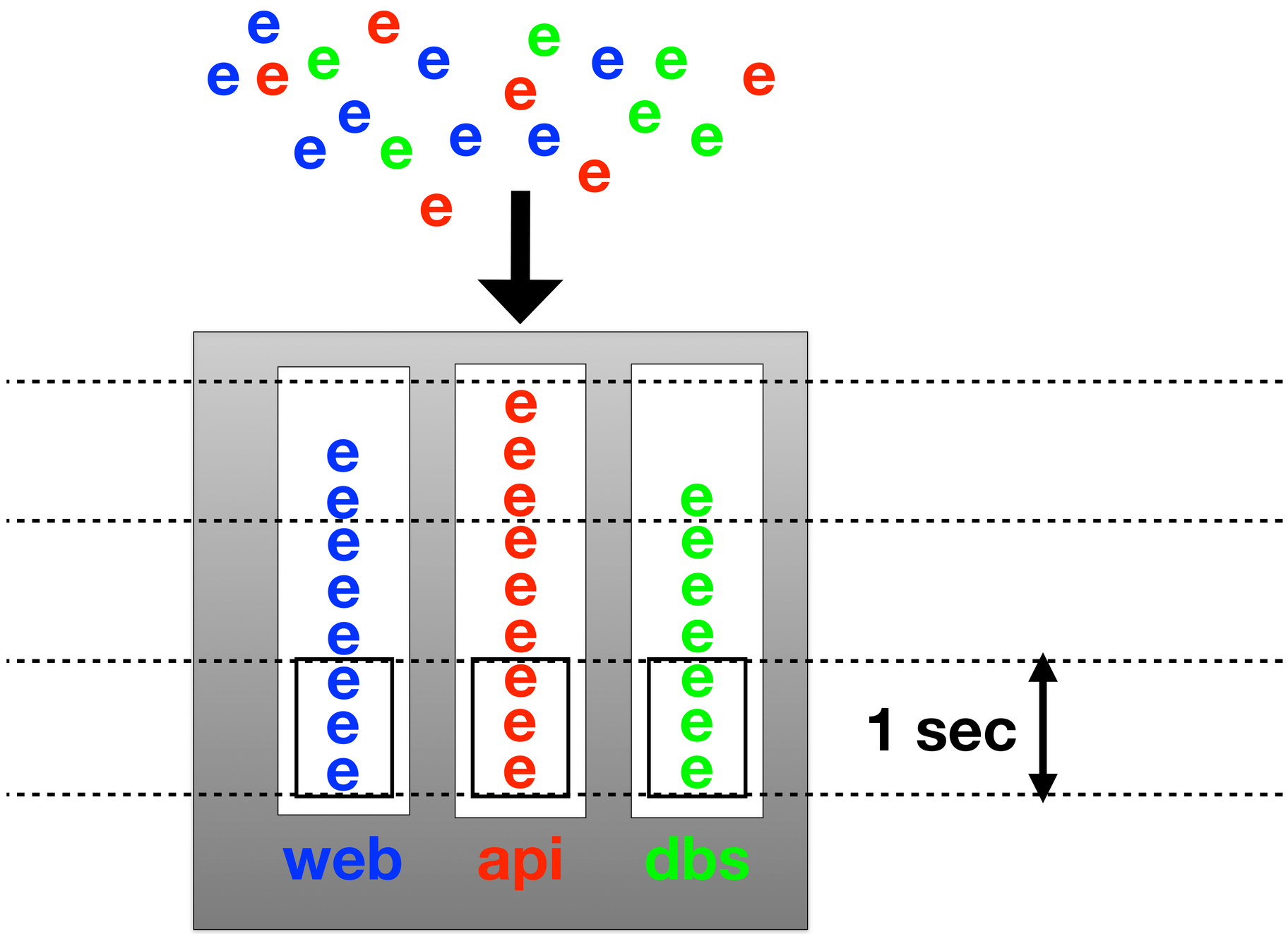
stockage des événements

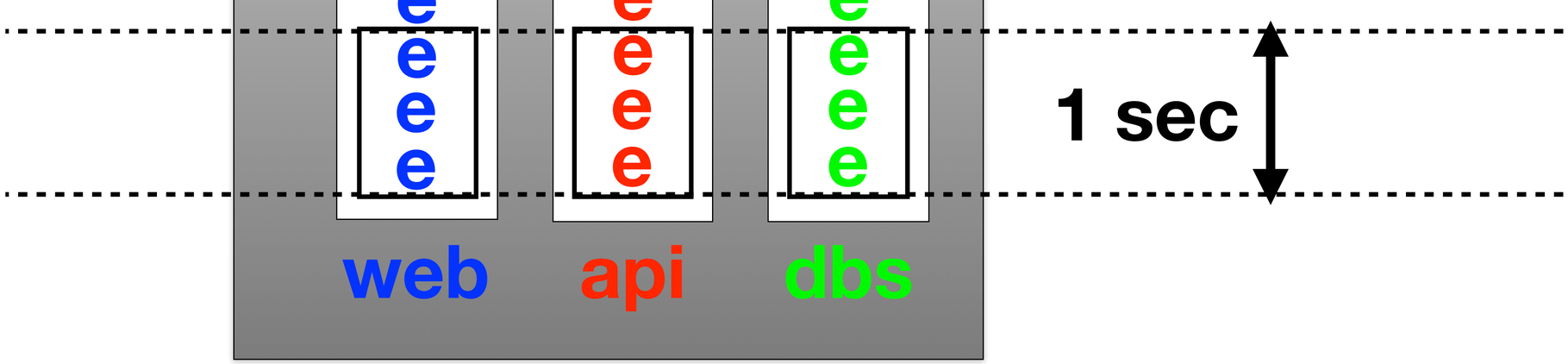

LOGGER

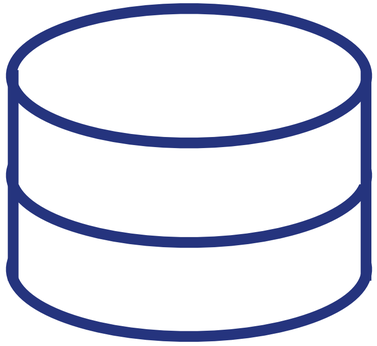

web
api


web
api
dbs

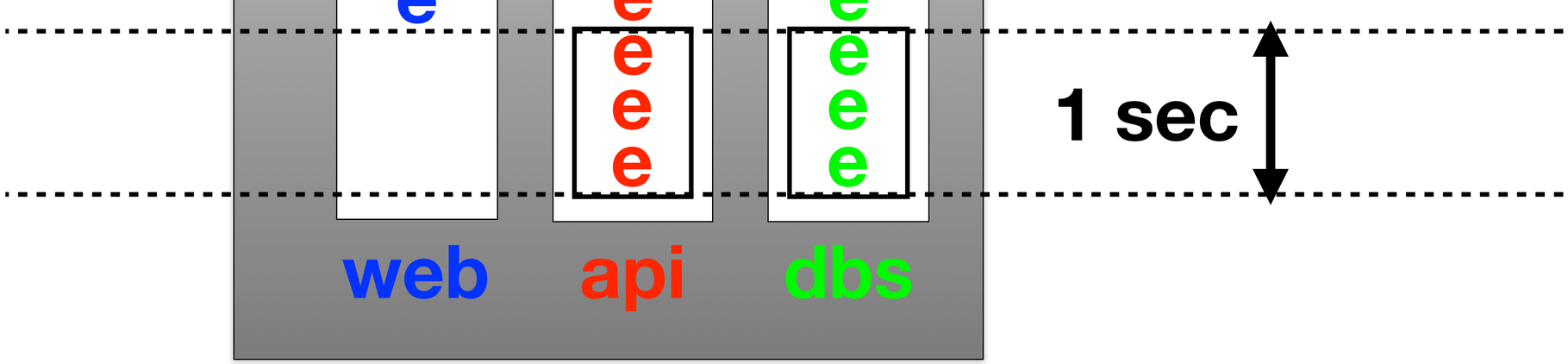

web
api
dbs
1 sec

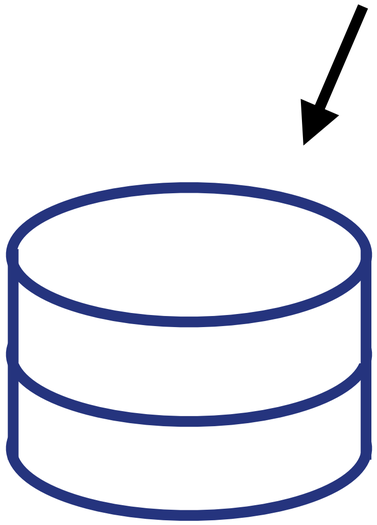

web
api
dbs
1 sec



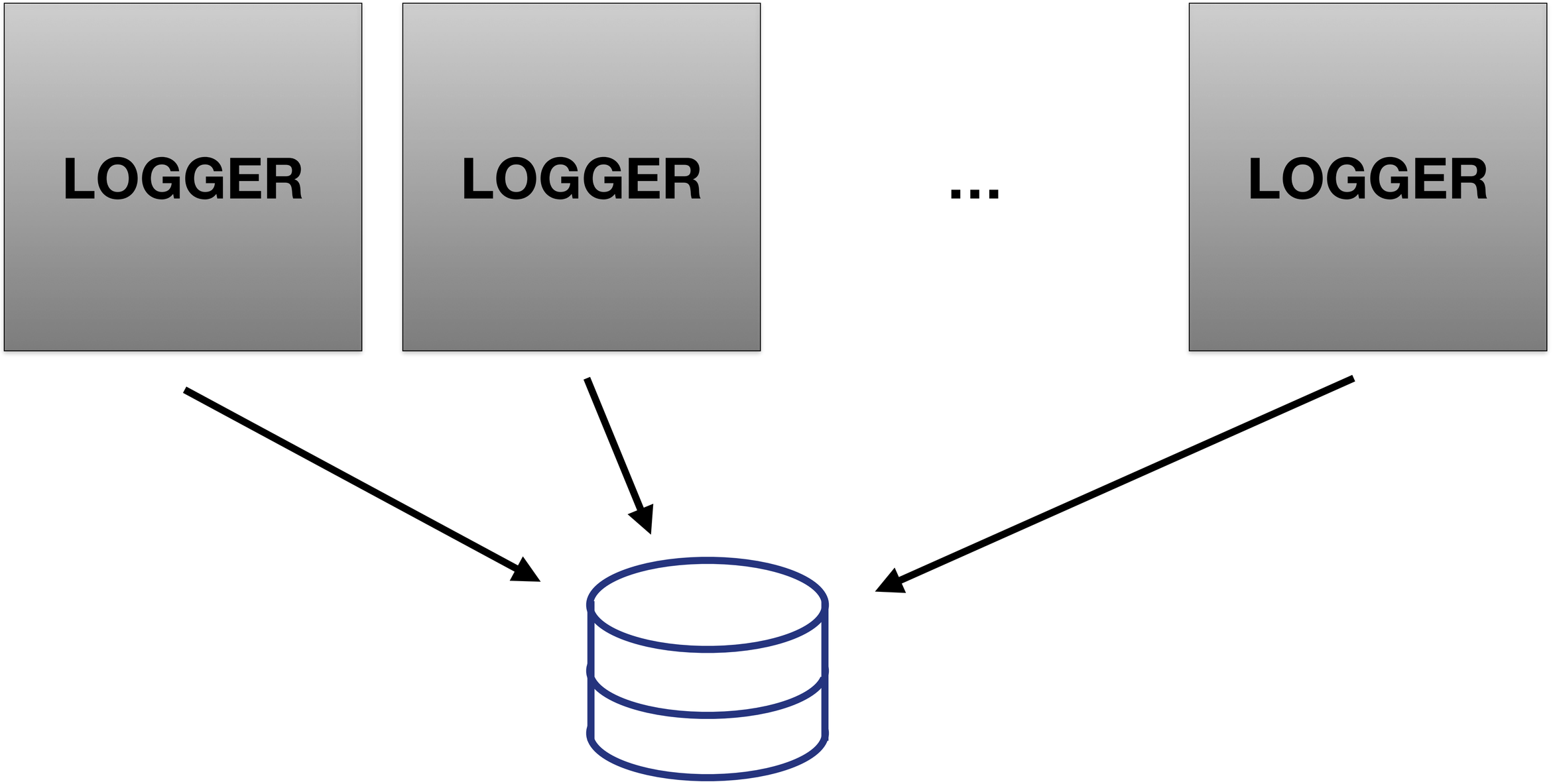


stockage des événements





stockage des événements



stockage des événements

STOCKAGE

BESOINS

- Sécurité de stockage
- Performance écriture massive
- Performance lecture massive
- Administration Facile
- Très scalable

NOUS AVONS CHOISI RIAK

- Sécurité: cluster, distribué, replication, Ring très robuste
- Performances bonnes et prévisibles
- Très facile à administrer

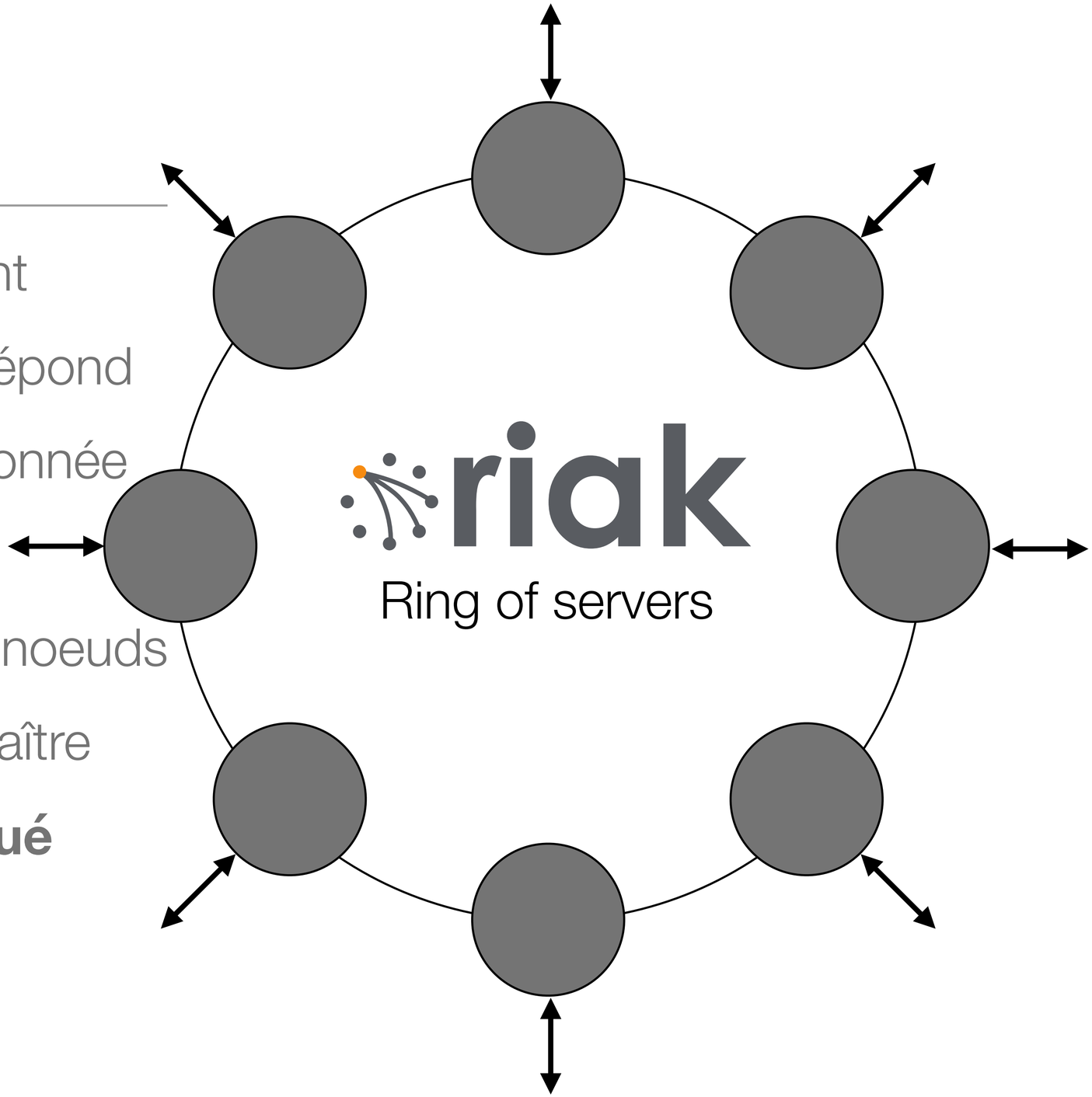
- Fonctions avancées (MapReduce, triggers, 2i, CRDTs …)
- Riak Search
- Réplication Multi-Datacenter


riak

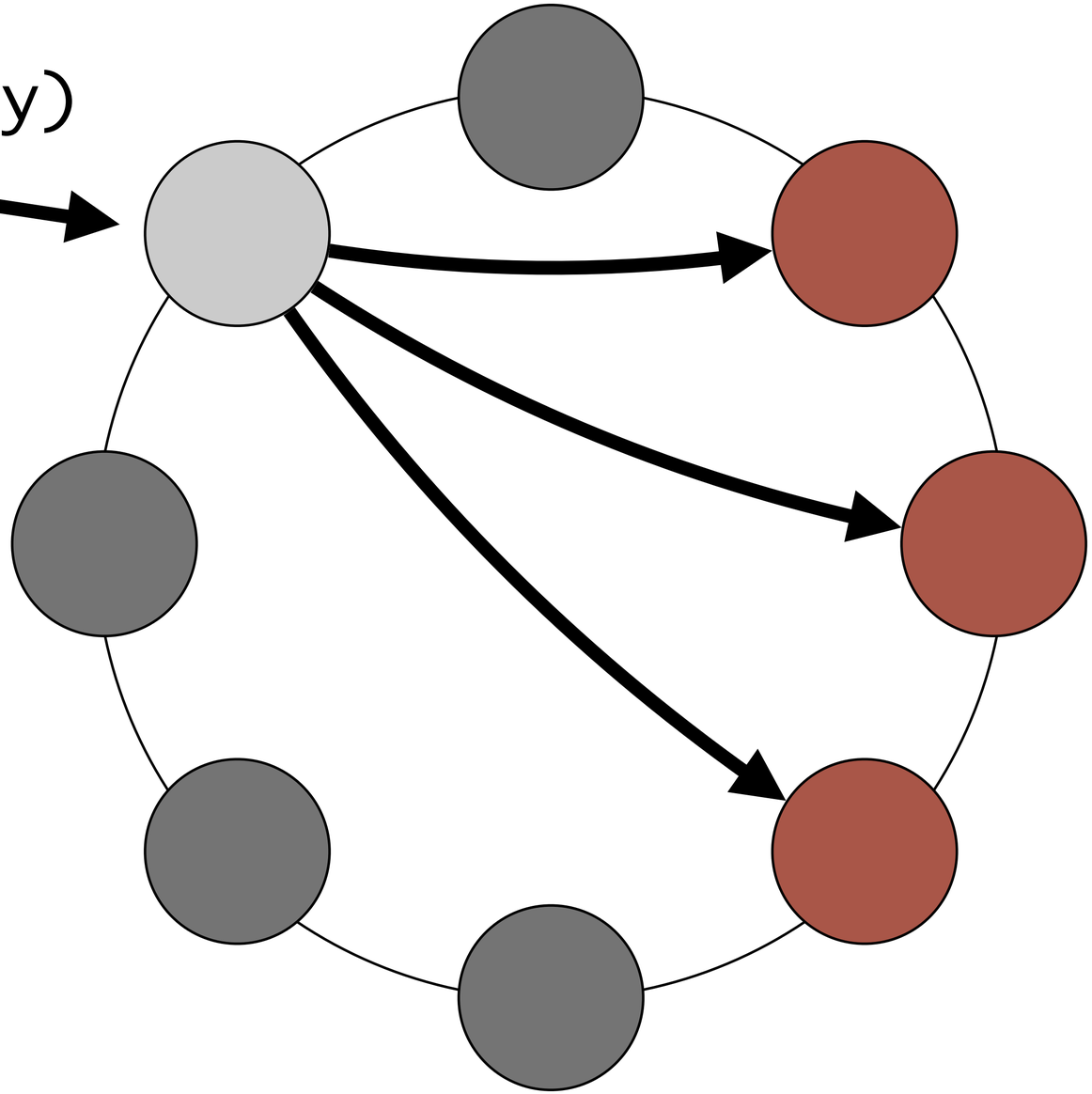
CLUSTER

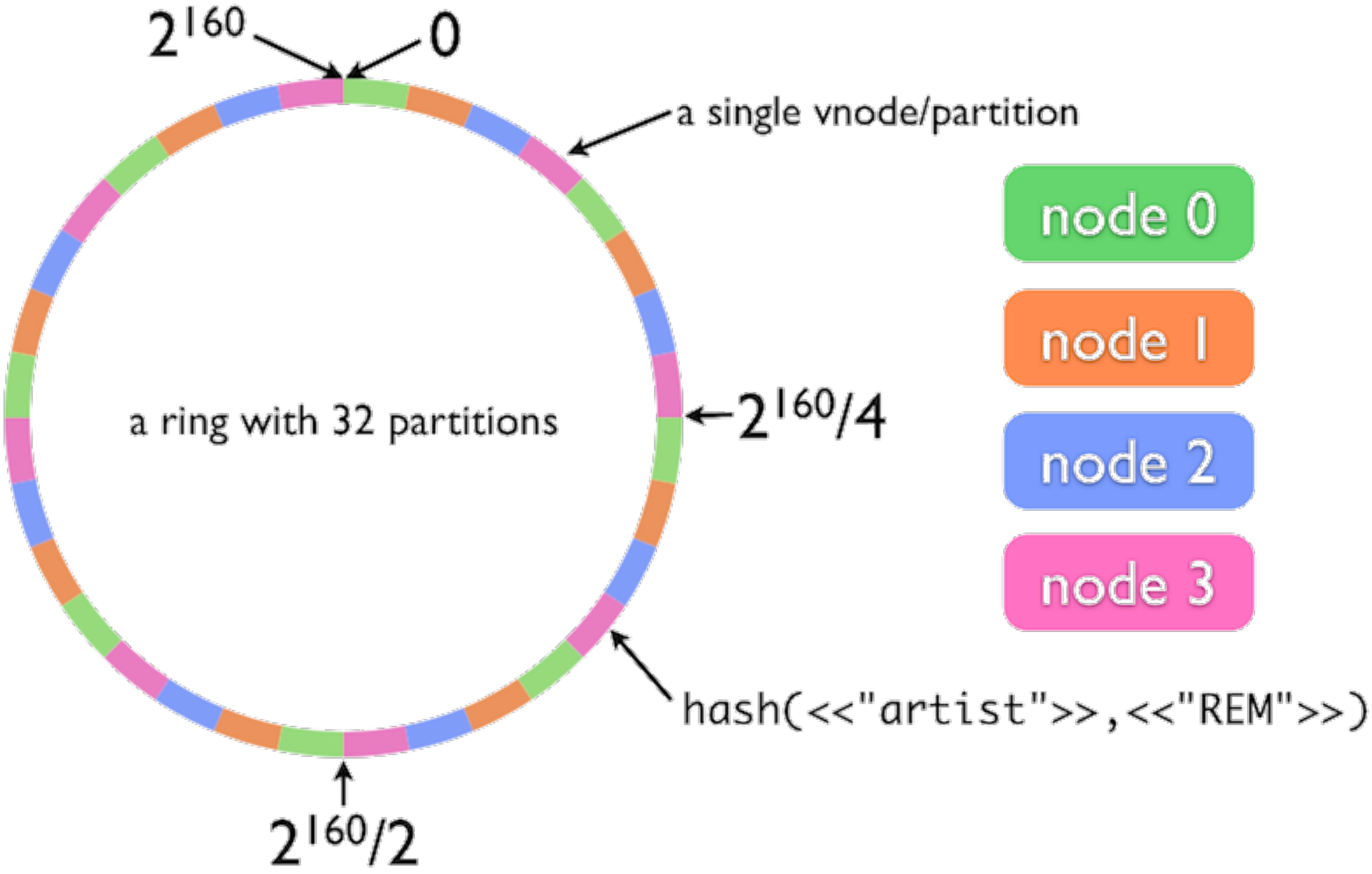
- Hardware courant
- Chaque noeud répond
- Réplication de donnée

- Gossip entre les noeuds
- Pas de noeud maître
- Système **distribué**



hash(key)




2^{160}
0
a single vnode/partition
node 0
node 1
node 2
node 3
a ring with 32 partitions
$2^{160}/4$
hash(<<"artist">>,<<"REM">>)
$2^{160}/2$

STOCKAGE CLEF VALEUR

- Espaces de noms: **bucket**
- Valeurs: **opaque** ou CRDTs

RIAK: FONCTIONALITES AVANCEES

- MapReduce
- Index secondaires (2i)
- Riak Search
- Réplication Multi-DataCenter

MULTI-BACKEND DE STOCKAGE

- Bitcask
- Eleveldb
- Memory

BACKEND: BITCASK

- Backend basé sur un système de logs
- AOF (Append-only files)
- Expiration de données fine
- Performance prévisibles (1 seek max)
- Parfait pour des données séquentielles

CONFIGURATION DU CLUSTER

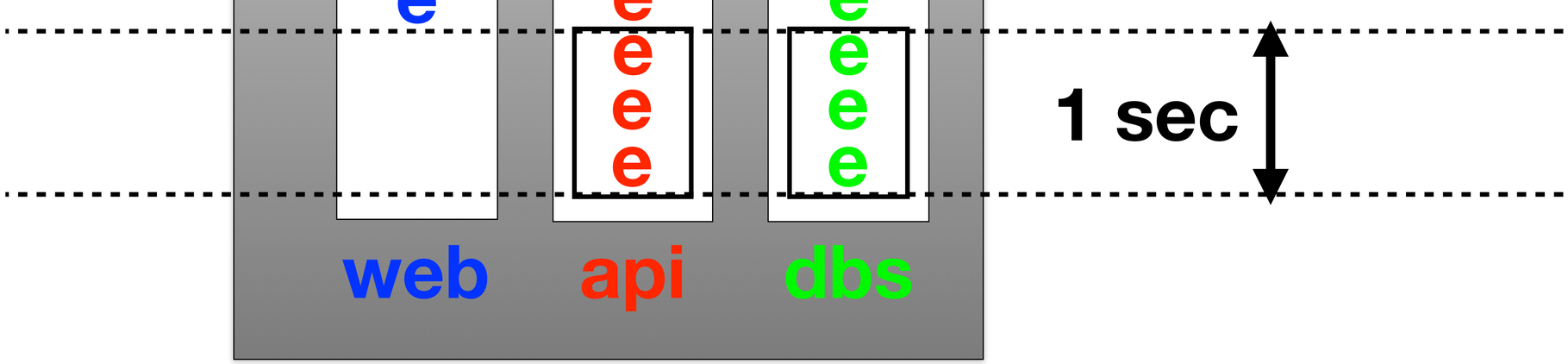
ESPACE DISQUE NECESSAIRE

- 8 jours
- 100 GB par heure
- Réplication = 3
- 100 * 24 * 8 * 3
- Besoin de **60 T**

HARDWARE UTILISE

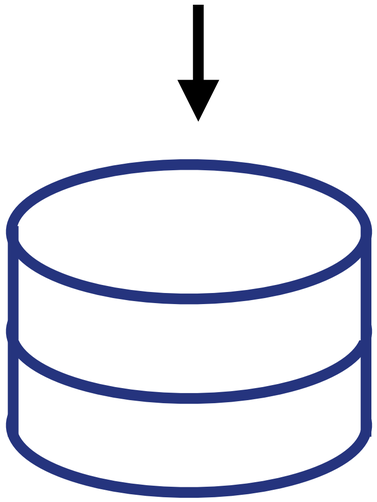
- 12 puis 16 nodes (bientôt 24)
- 12 CPU cores (Xeon 2.5Ghz) par noeud
- 192 GB RAM
- réseau 1 Gbit/s
- 8 TB (raid 6)
- Espace total: 128 TB

DATA DESIGN



regroupé par EPOCH / DC / CELL / TYPE / SUBTYPE

500 KB max par morceaux



stockage des événements

DONNEES

- Bucket : “`data`“
- Clef: “`12345:1:cell0:WEB:app:chunk0`“
- Valeur: Liste sérialisée compressée des events HashMap
- 200 clefs par secondes

META-DONNEES

- Bucket : “metadata“
- Clef: <epoch>-<dc> “1428415043-2“
- Valeur: liste de clef données:

```
[ “1428415043:1:cell0:WEB:app:chunk0“,
  “1428415043:1:cell0:WEB:app:chunk1“
…
  “1428415043:4:cell0:EMK::chunk3“ ]
```

- PSV (pipe separated value)

ECRIRE LES DONNEES

ECRIRE LES DONNEES

- Dans chaque DC, cellule, Loggers poussent vers Riak
- 2 protocoles: REST ou ProtoBuf
- Chaque seconde:
  - Ecrire les valeurs “data” vers Riak, en asynchrone
  - Attendre retour succès
  - Ecrire les meta-données “metadata”

JAVA

```
Bucket DataBucket = riakClient.fetchBucket("data").execute();
DataBucket.store("12345:1:cell0:WEB:app:chunk0", Data1).execute();
DataBucket.store("12345:1:cell0:WEB:app:chunk1", Data2).execute();
DataBucket.store("12345:1:cell0:WEB:app:chunk2", Data3).execute();

Bucket MetaDataBucket = riakClient.fetchBucket("metadata").execute();
MetaDataBucket.store("12345-1", metaData).execute();
riakClient.shutdown();
```

Perl

```
my $client = Riak::Client->new(…);

$client->put(data => '12345:1:cell0:WEB:app:chunk0', $data1);
$client->put(data => '12345:1:cell0:WEB:app:chunk1', $data2);
$client->put(data => '12345:1:cell0:WEB:app:chunk2', $data3);

$client->put(metadata => '12345-1', $metadata, 'text/plain' );
```

LIRE LES DONNEES

LIRE UNE SECONDE

- Pour **une seconde** (une epoch donnée)
- Lire les meta-données pour l’<epoch>-DC
- Parser la valeur
- **Filtrer les types / sous-types**
- Récupérer les données de “data”

Perl

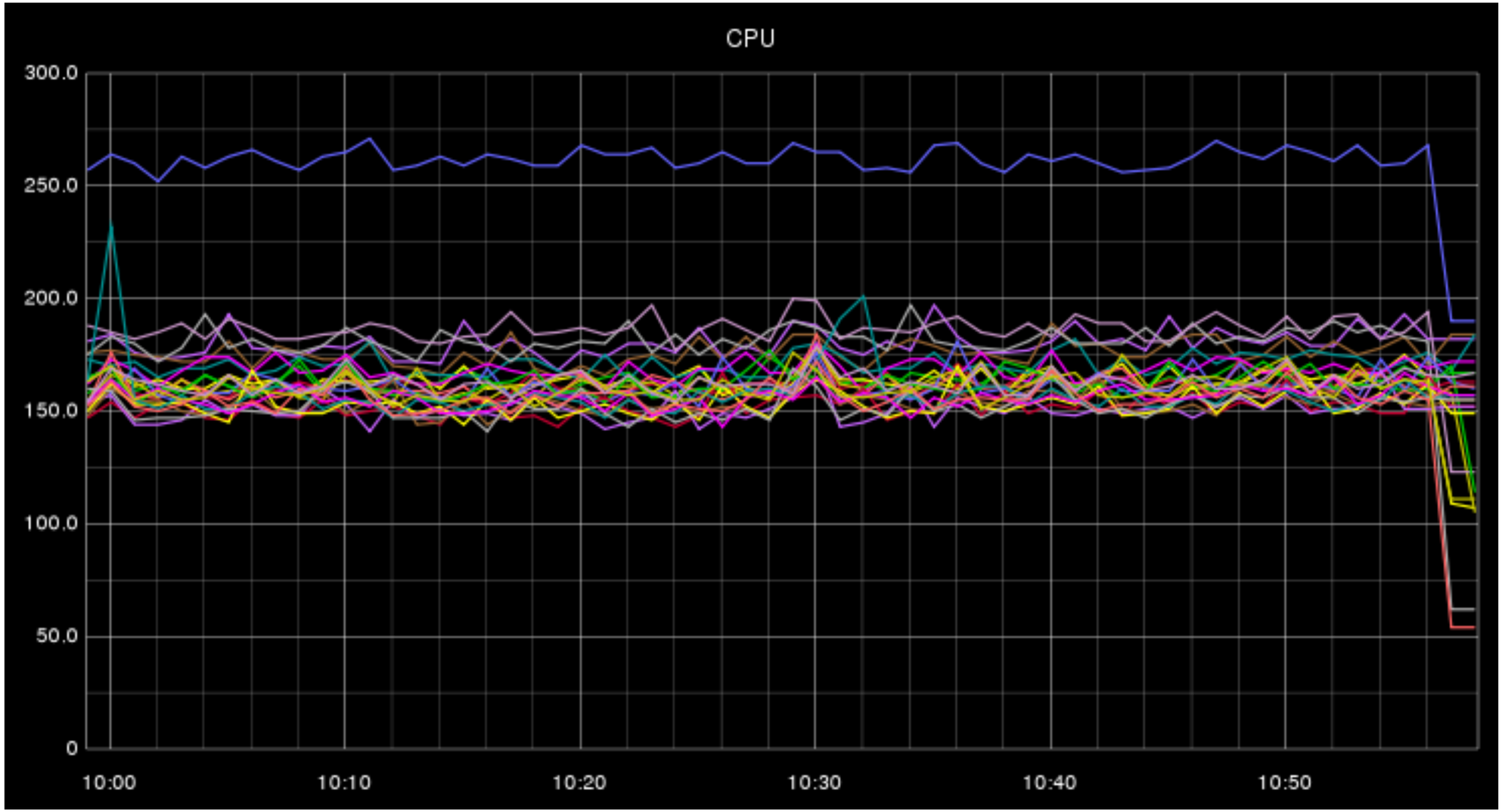
```
my $client = Riak::Client->new(…);
my @array = split '\|', $client->get(metadata => '1428415043-1');
@filtered_array = grep { /WEB/ } @array;
$client->get(data => $_) foreach @filtered_array;
```

LIRE UNE PERIODE

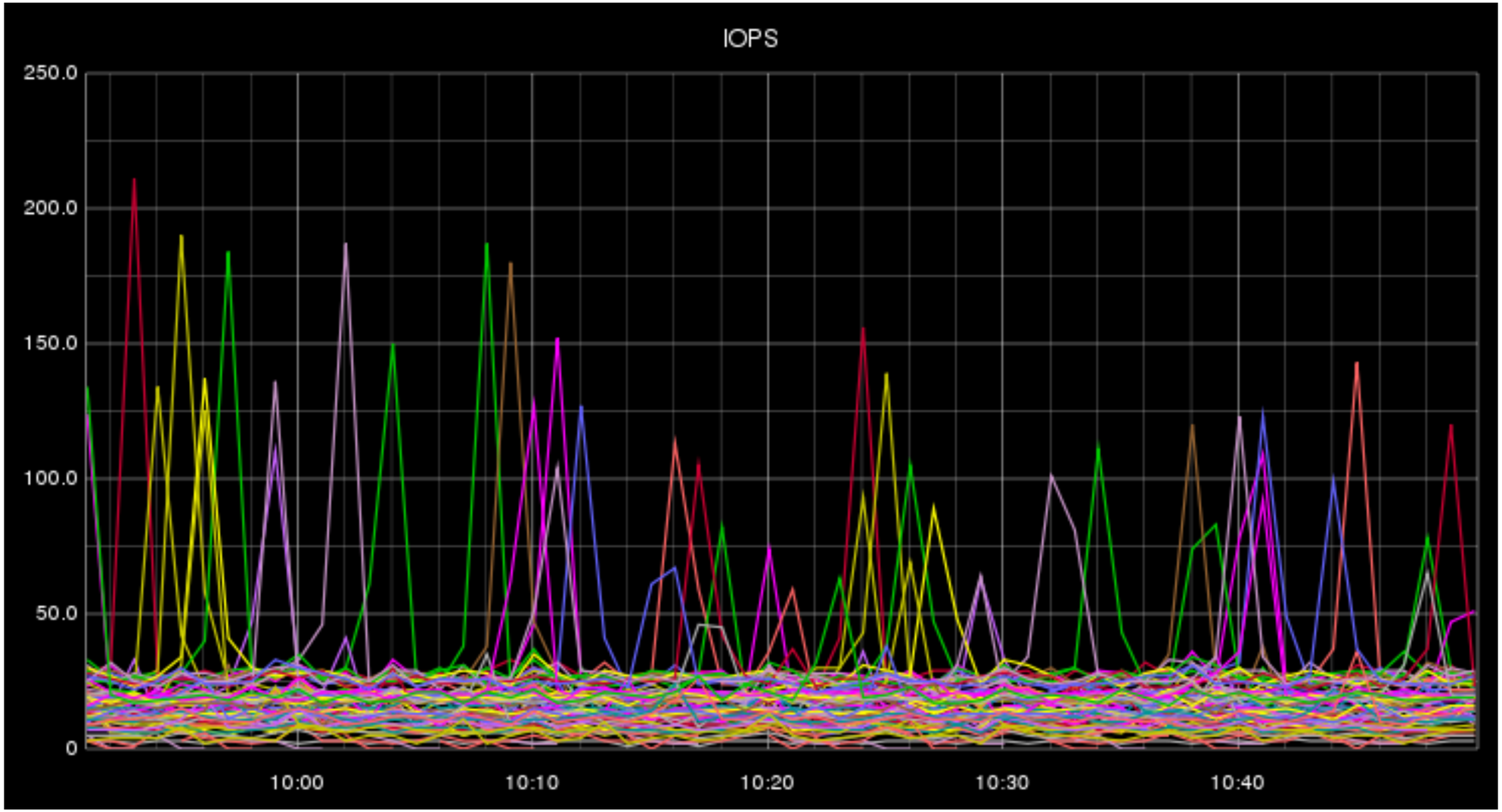
- Pour **un intervalle** epoch1 -> epoch2
- Générer la liste des epochs
- Récupérer en parallèle

RIAK CLUSTER HEALTH

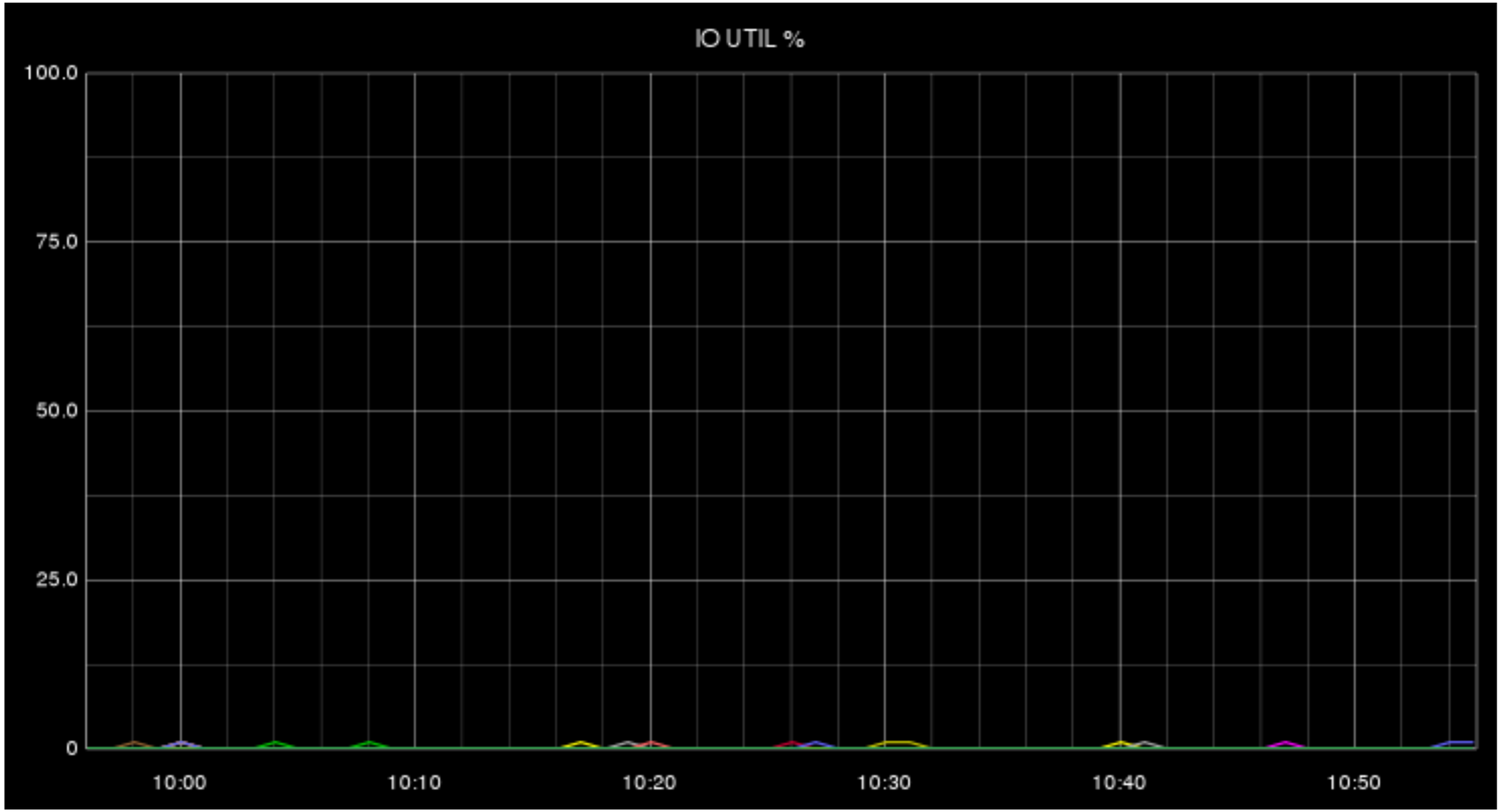
CPU


CPU
300.0
250.0
200.0
150.0
100.0
50.0
0
10:00
10:10
10:20
10:30
10:40
10:50

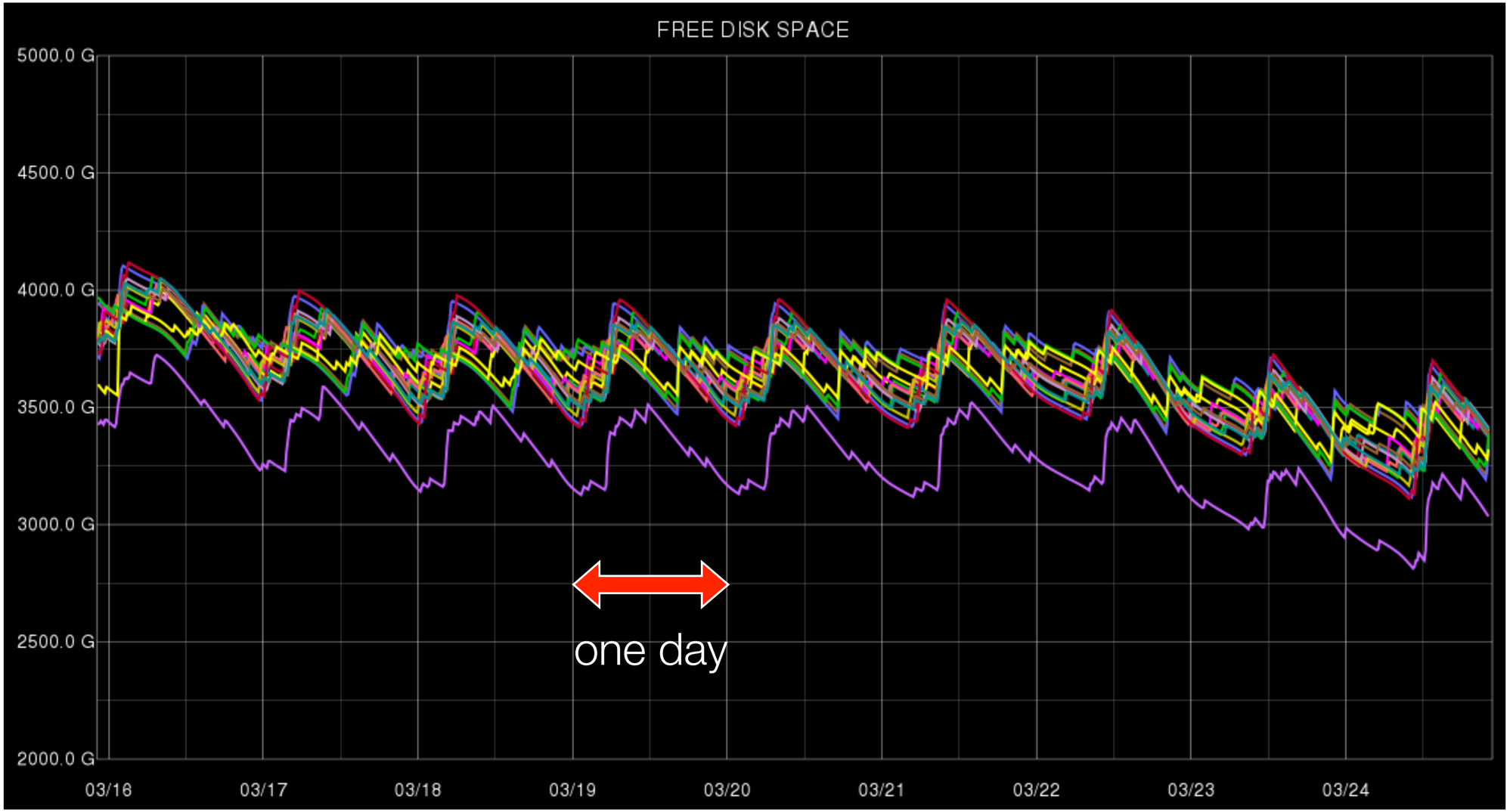
DISK IOPS



DISK IO %



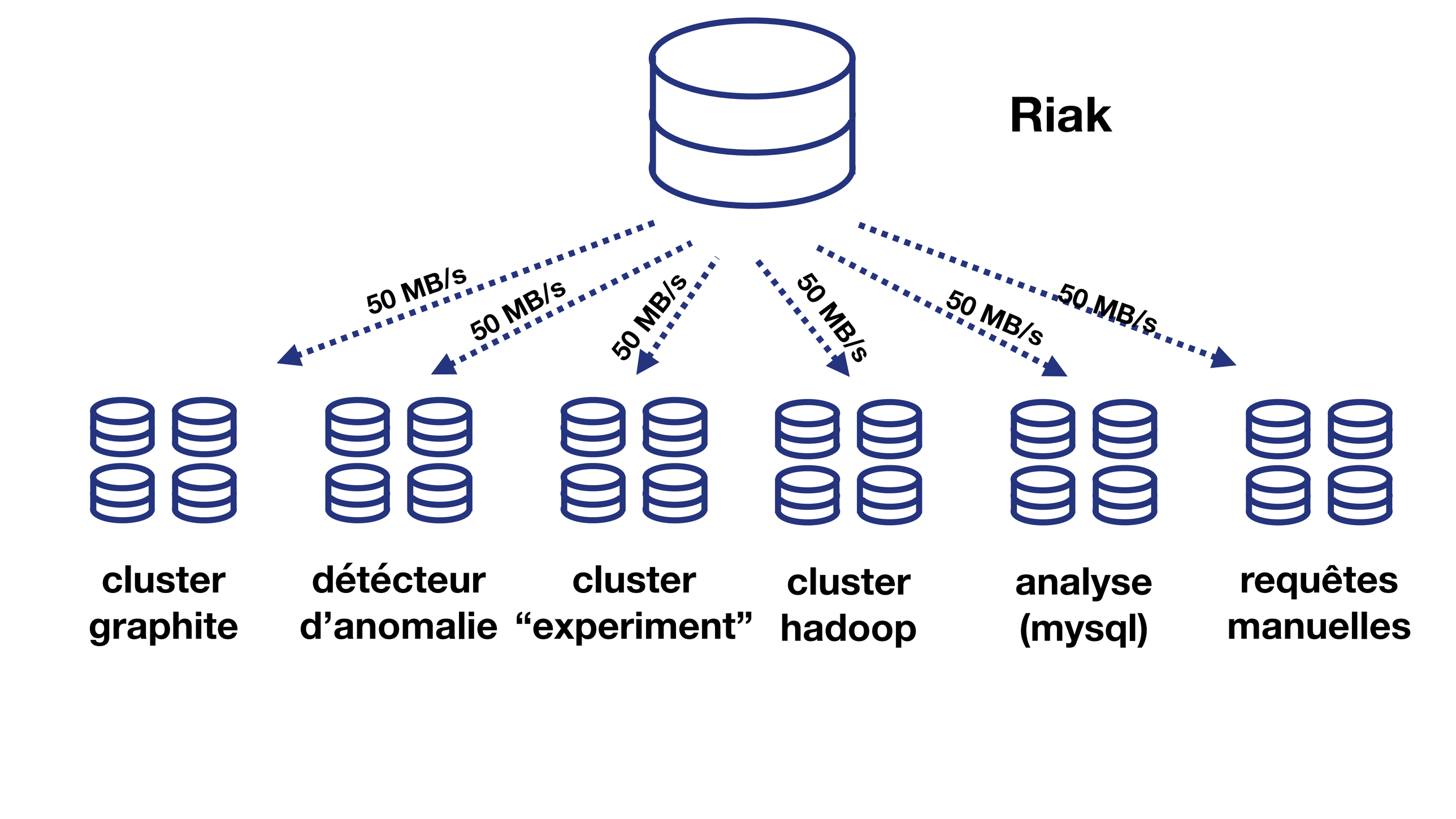
EXPIRATON & ESPACE DISQUE



TRAITEMENT TEMPS REEL HORS RIAK

EXEMPLES

- Streaming => Cluster Graphite (chaque seconde)
- Streaming => Détecteur d'Anomalie (agrégation 2 min.)
- Streaming => Cluster "Experiment" (agrégation 1 jour)
- Chaque minute => Cluster Hadoop
- Requête manuelle => tests, debug, investigation
- Requête Batch => analyse ad hoc
- => quantité **énorme** de requête en **lecture**


Riak
50 MB/s
50 MB/s
50 MB/s
50 MB/s
50 MB/s
50 MB/s
cluster graphite
détécteur d’anomalie
cluster “experiment”
cluster hadoop
analyse (mysql)
requêtes manuelles

VRAI TEMPS REEL

- 1 seconde de données
- Stockée en moins d’une seconde
- Disponible en moins d’une seconde
- Problème long terme : saturation réseau

TRAITEMENT TEMPS REEL DANS RIAK

L’IDEE

- Au lieu de
  - Lire la donnée
  - Traiter la donnée (ex: moyenne)
  - Produire un petit résultat
- Faire
  - Envoyer le code près des données
  - Traiter les données sur Riak
  - Récupérer le résultat

CE QUI PREND DU TEMPS

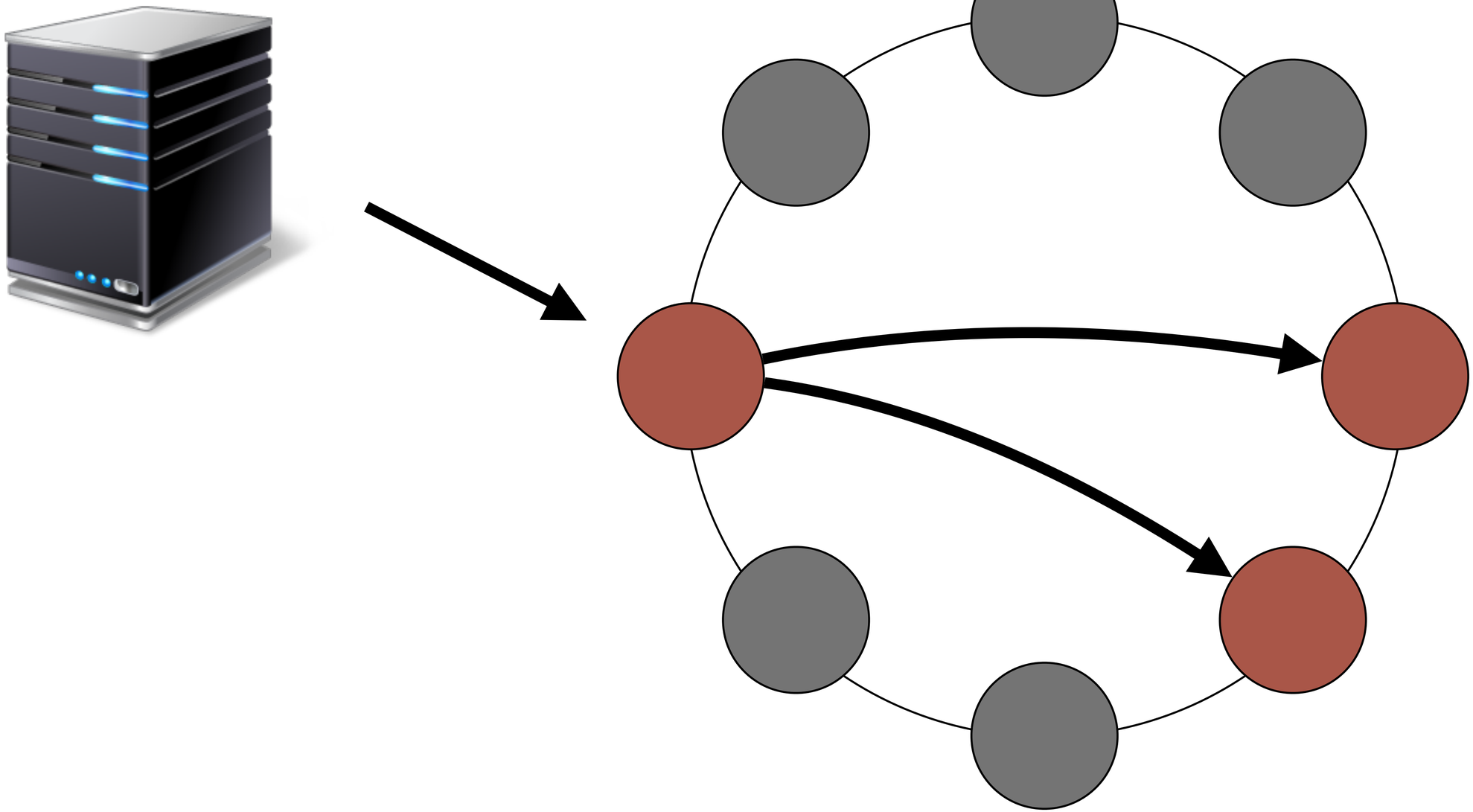
- Prend du temps et de la bande passante
  - Récupérer les données: **BP Réseau**
  - Décompresser/désérialiser: **Temps CPU**
- Ne prend pas de temps
  - Traiter les données

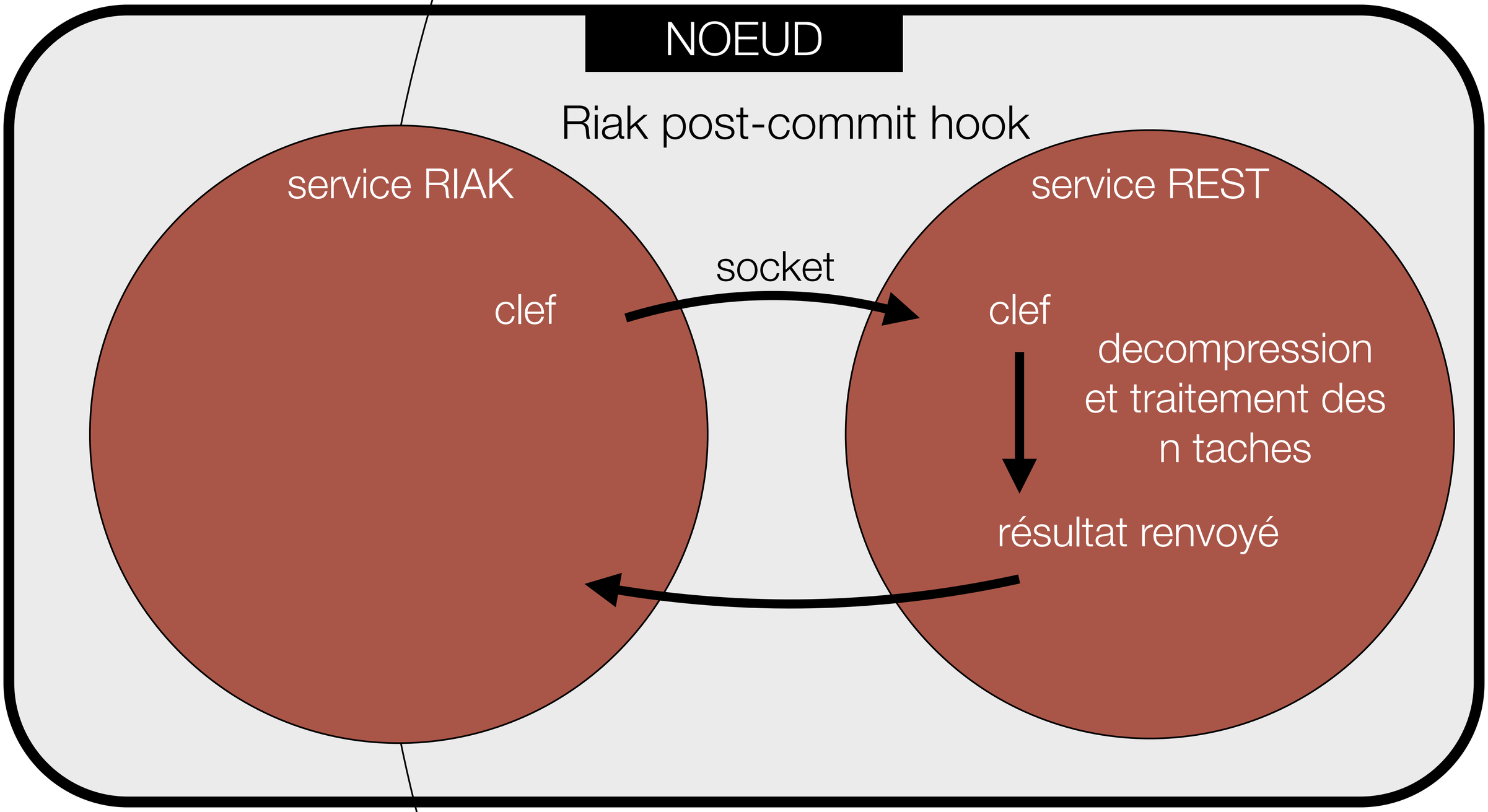
MAPREDUCE

- Entrée: epoch-dc
- Map1: clef meta-donnée -> clefs données
- Map2: calcul sur la donnée
- Reduce: regrouper
- Temps réel: OK
- OK pour la **BP réseau**, pas pour le **Temps CPU**

HOOKS

- A chaque écriture des méta-données
- Post-Commit hook déclenché
- Traite les données sur les noeuds




NOEUD
Riak post-commit hook
service RIAK
clef
socket
service REST
clef
decompression
et traitement des
n taches
résultat renvoyé

HOOK CODE

```
metadata_stored_hook(RiakObject) ->
    Key = riak_object:key(RiakObject),
    Bucket = riak_object:bucket(RiakObject),
    [ Epoch, DC ] = binary:split(Key, <<"-">>),
    MetaData = riak_object:get_value(RiakObject),
    DataKeys = binary:split(MetaData, <<"|">>, [ global ]),
    send_to_REST(Epoch, Hostname, DataKeys),
    ok.
```

```
send_to_REST(Epoch, Hostname, DataKeys) ->
    Method = post,
    URL = "http://" ++ binary_to_list(Hostname)
       ++ ":5000?epoch=" ++ binary_to_list(Epoch),
    HTTPOptions = [ { timeout, 4000 } ],
    Options = [ { body_format, string },
                { sync, false },
                { receiver, fun(ReplyInfo) -> ok end }
              ],
    Body = iolist_to_binary(mochijson2:encode( DataKeys )),
    httpc:request(Method, {URL, [], "application/json", Body},
                  HTTPOptions, Options),
    ok.
```

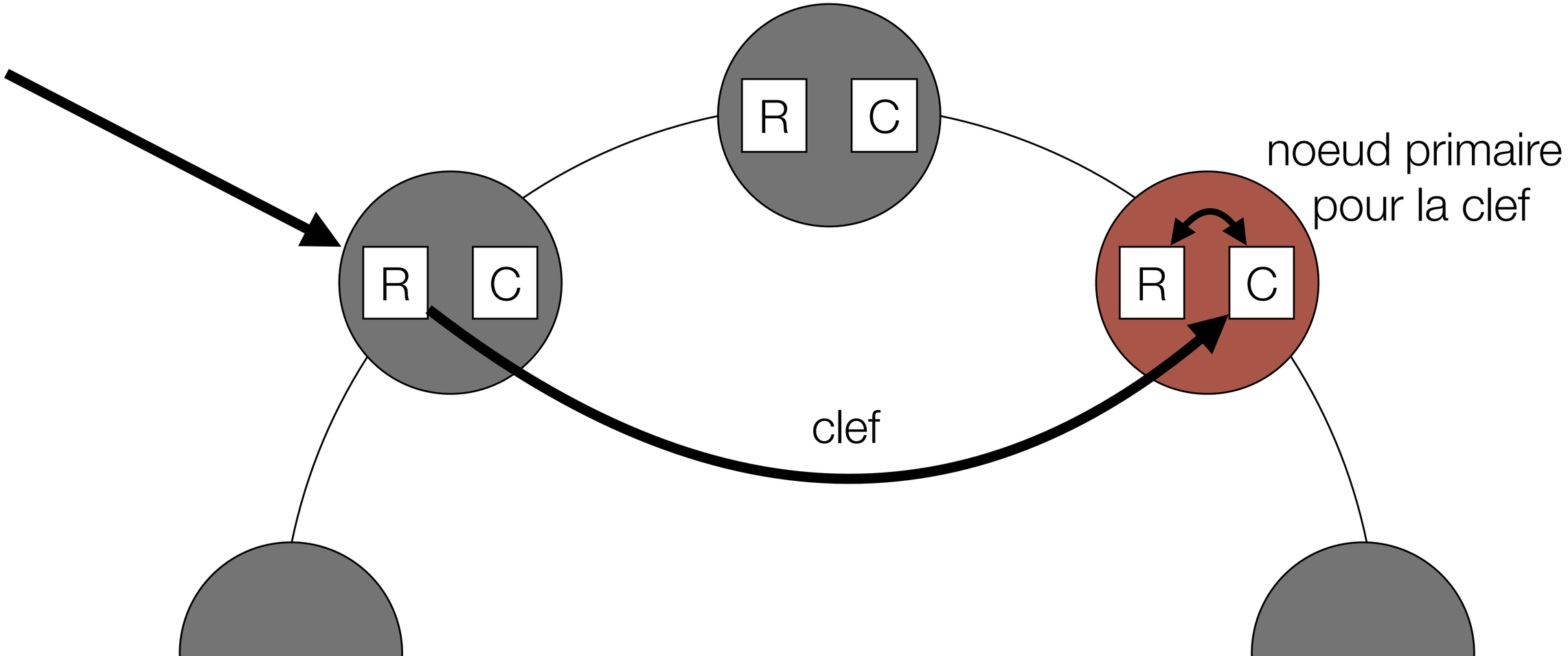
SERVICE REST

- En Perl, norme PSGI (WSGI), Starman, preforks
- Permet d’écrire les traitements en Perl
- Supporte chargement de code à la demande

SERVICE REST: SCALABLE

- **Scalable**
- 1 seconde = 200 clefs
- 16 noeuds, 10 CPUs (2 pour Riak)
- 1 clef doit être traitée en 16*10/200 sec = 0.8 sec
- Inclus réseau + CPU + overhead
- => Amplement le temps

OPTIMISATION



AVANTAGES

- Utilisation CPU et tps d’execution peut être limité
- Donnée est **locale** au traitement
- Donnée décompressée **une seule fois**
- Tous les traitements sont fait sur une donnée
- Les deux systèmes sont découplés (Erlang surveille)
- Peut être écrit en n’importe quel langage

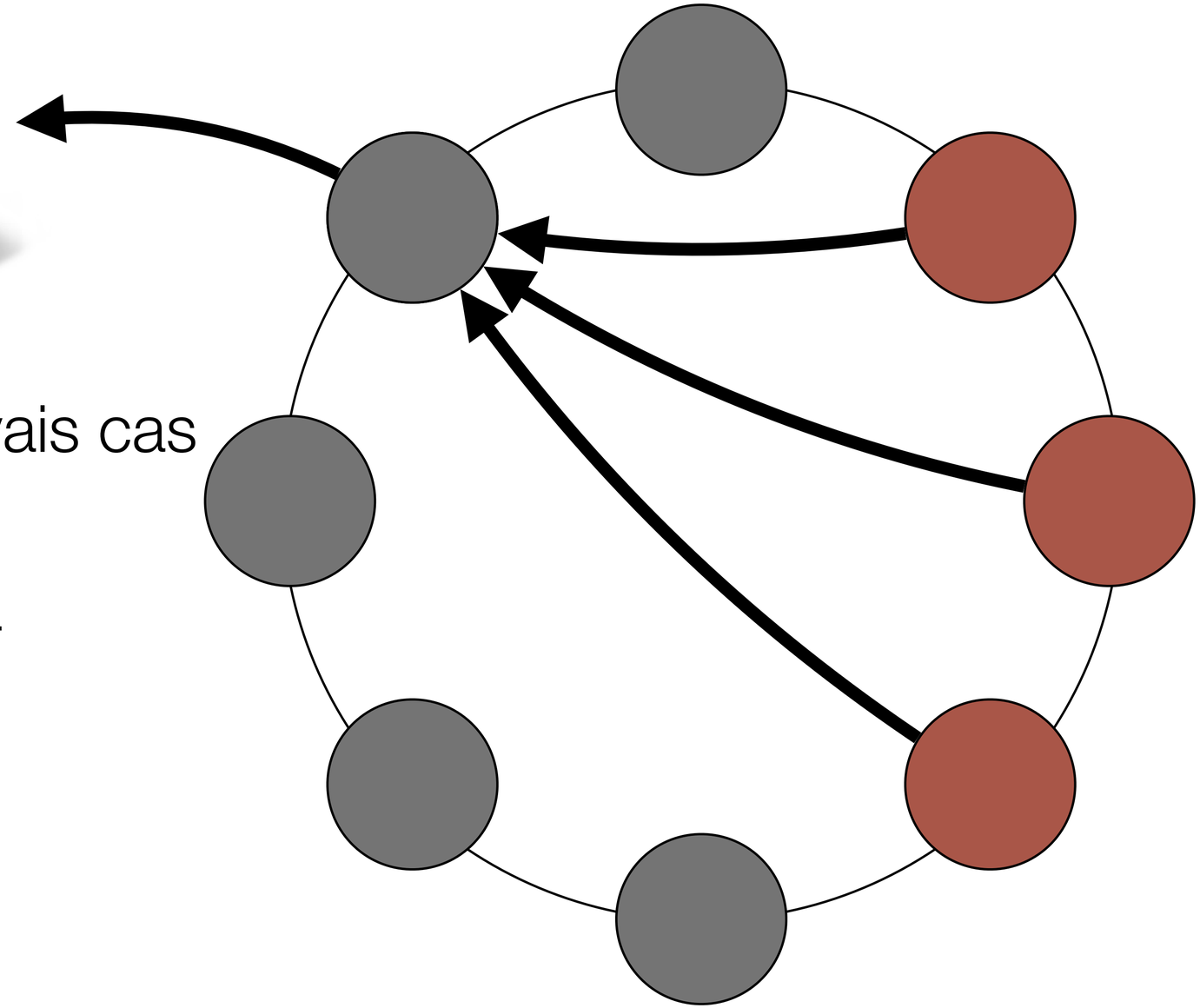
DESAVANTAGES

- Seulement pour données temps réel
- Traitement inter-epoch moins facile

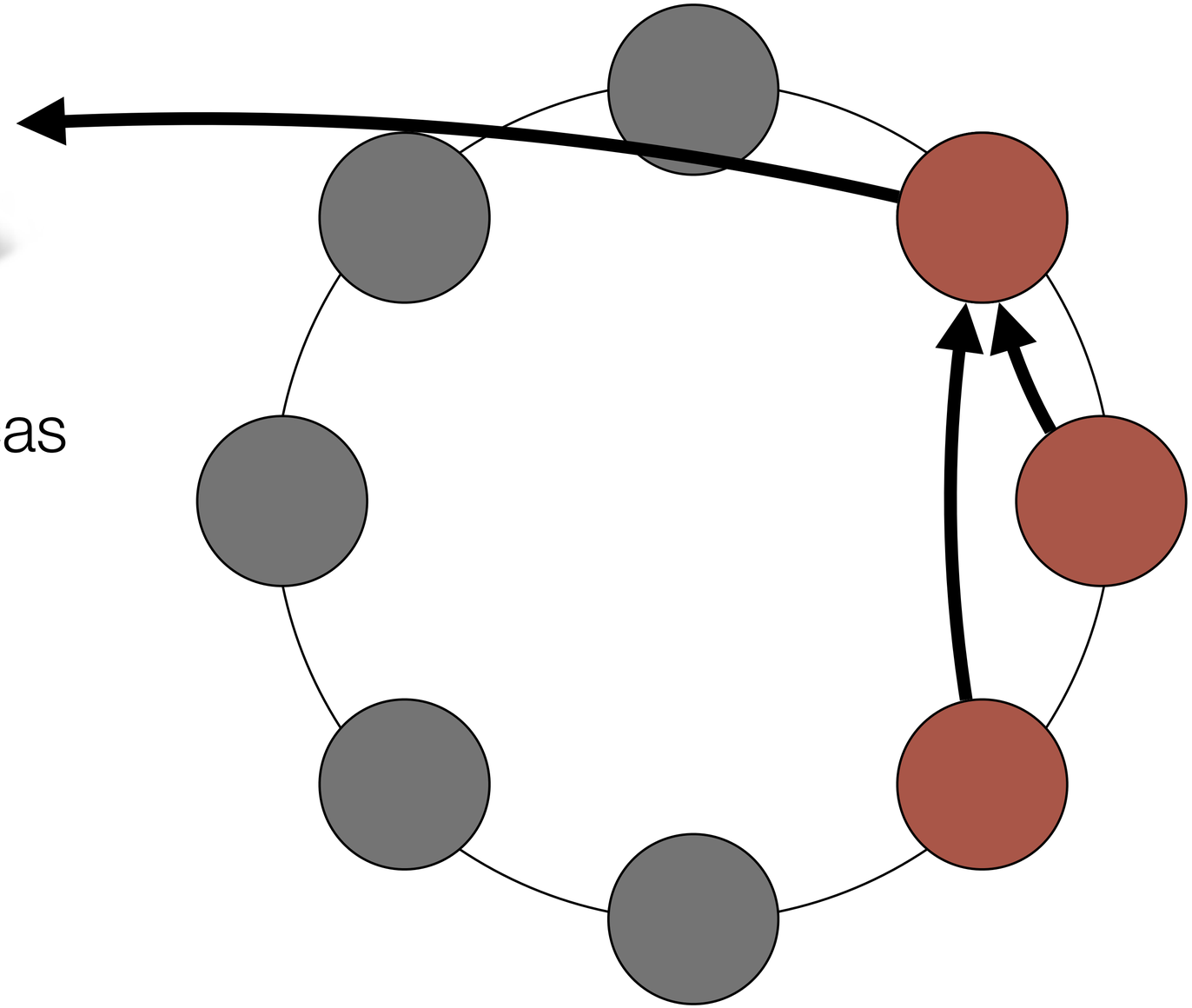
FUTUR

- Utiliser ce système pour réduire l’utilisation réseau
- Explorer la piste de Riak Search

BANDE PASSANTE RESEAU: PROBLEME



- **GET** - mauvais cas
- intérieur =
  3 x extérieur



- **GET** - bon cas
- intérieur =
  2 x extérieur

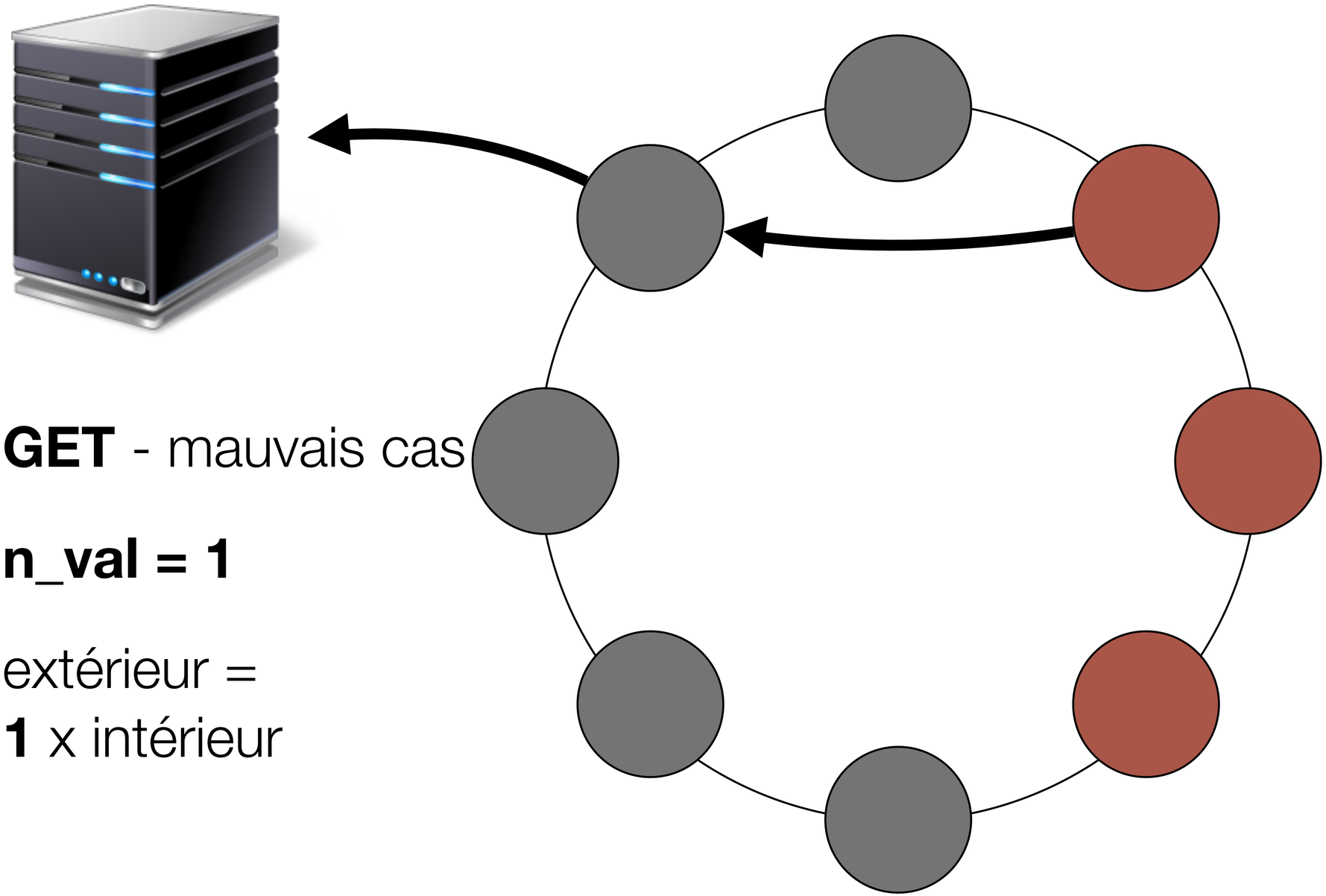
BANDE PASSANTE

- Ratio tend vers 3
- Cas général : OK
- Dans notre cas: problème
- Grosses valeurs, PUT constants, beaucoup de GET
- Limité à 1 Gbits/s

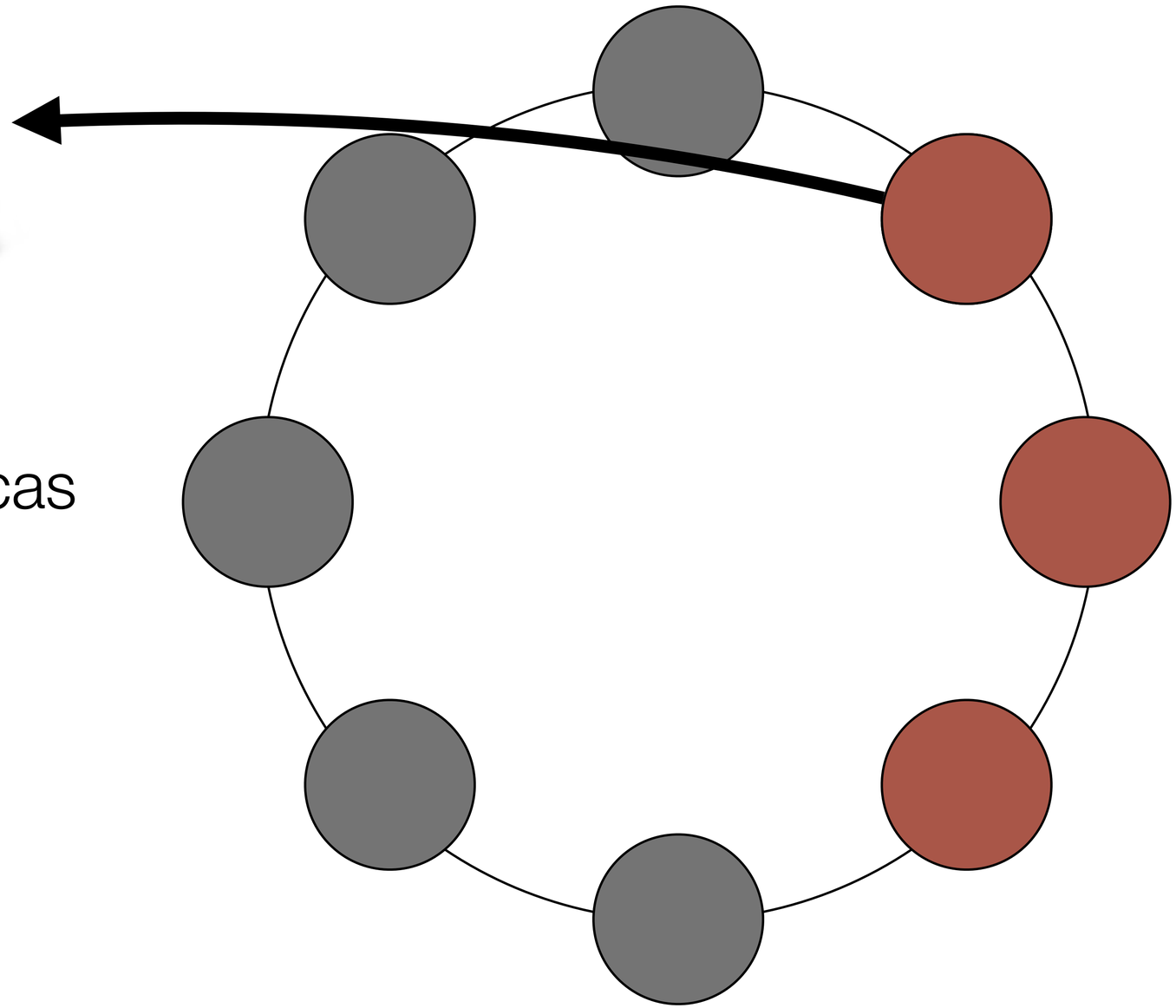
BANDE PASSANTE RESEAU: SOLUTIONS

BANDE PASSANTE RESEAU: SOLUTIONS

1. GET à faible consommation réseau
2. Ne pas choisir un noeud au hasard



- **GET** - mauvais cas
- **n_val = 1**
- extérieur =
  1 x intérieur



- **GET** - bon cas
- **n_val = 1**
- exterieur =
  0 x intérieur

RESULTAT

- Utilisation réseau réduite par 2 !

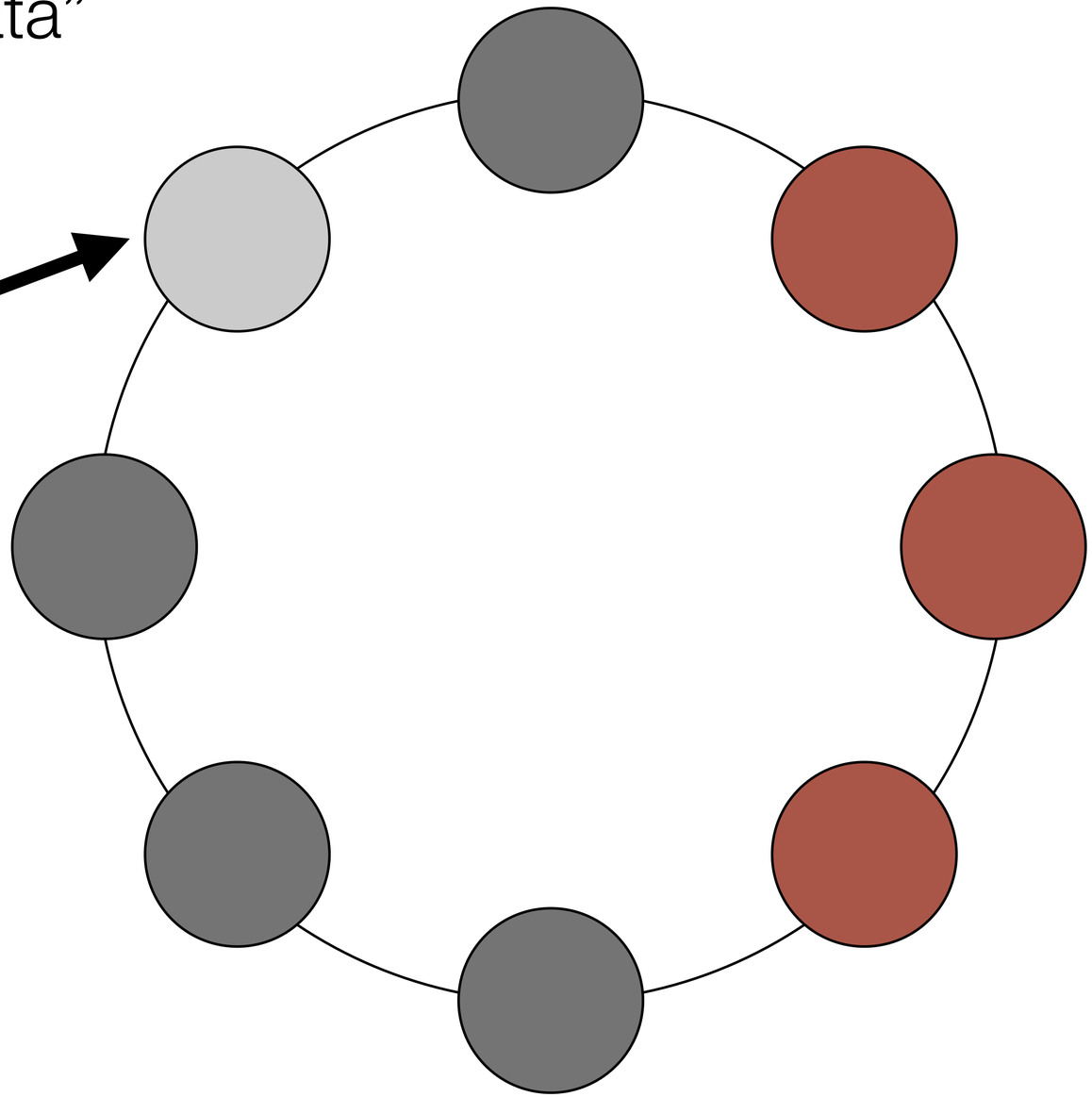
ATTENTION

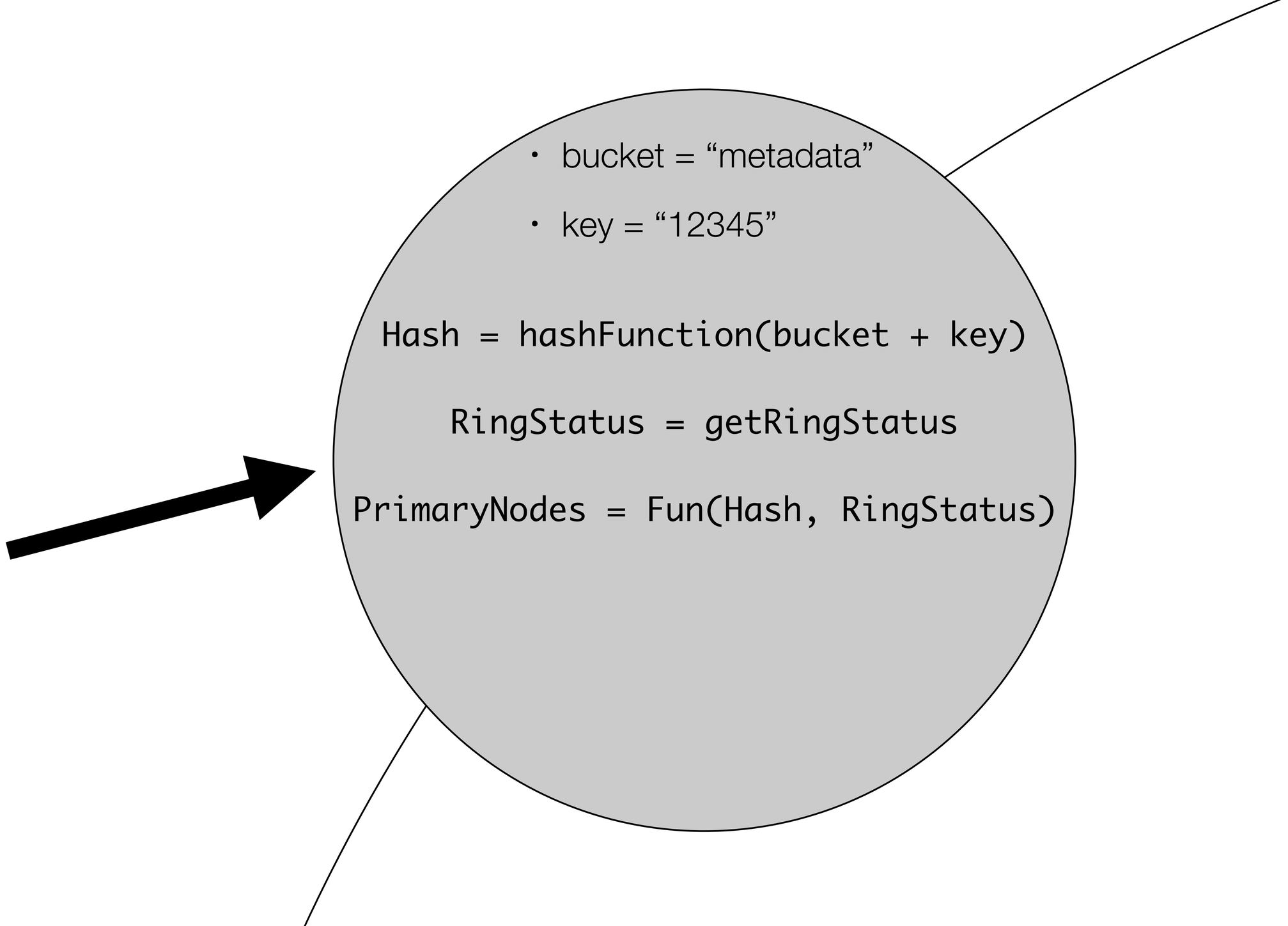
- Possible car les données sont en lecture seule
- Données ont un checksum interne
- Aucun conflit n’est possible
- Corruption de donnée auto-détectée
- Prévoir un fallback avec n_val=3

BANDE PASSANTE RESEAU: SOLUTIONS

1. GET à faible consommation réseau
2. Ne pas choisir un noeud au hasard

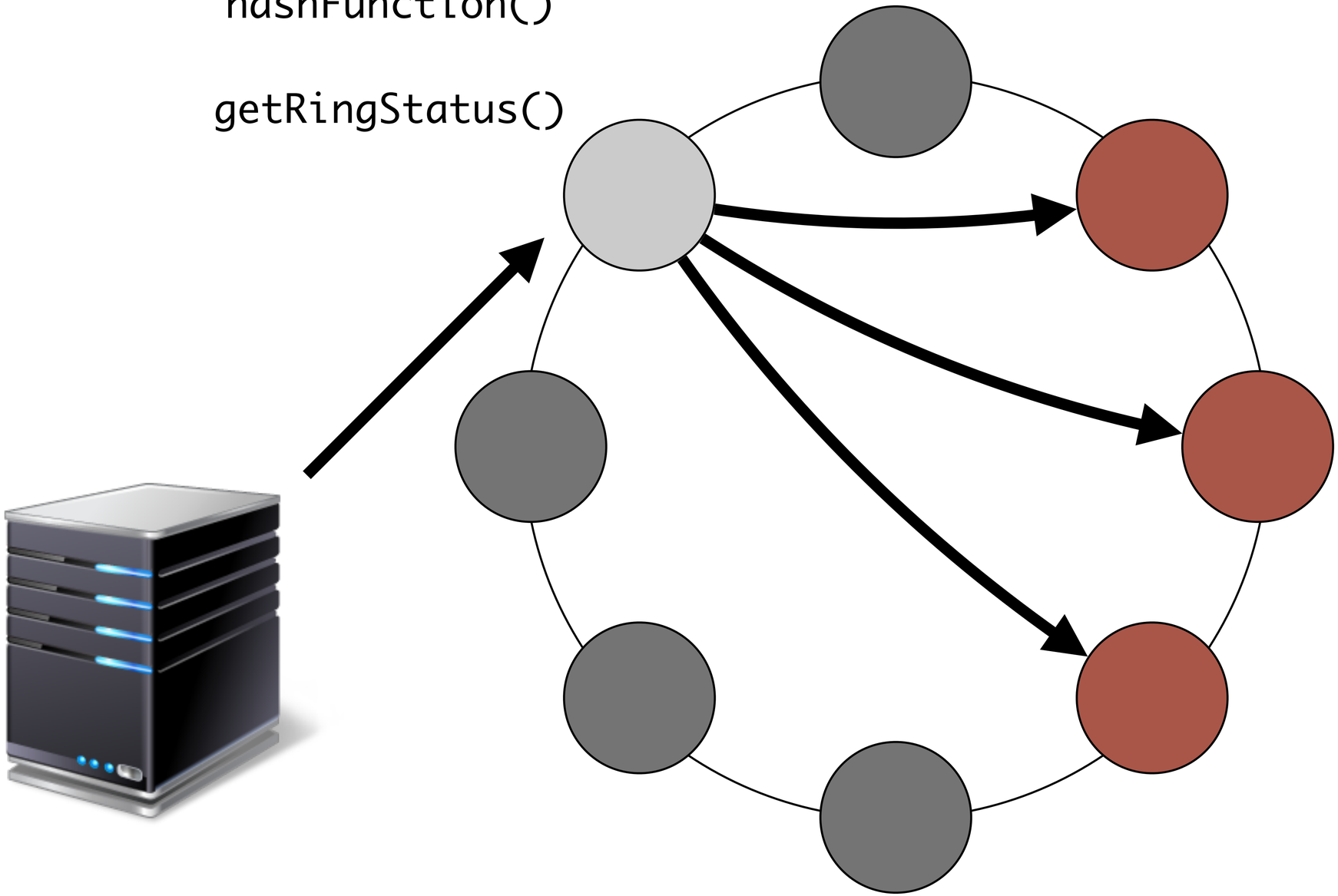
- bucket = “metadata”
- key = “12345”




• bucket = “metadata”
• key = “12345”
Hash = hashFunction(bucket + key)
RingStatus = getRingStatus
PrimaryNodes = Fun(Hash, RingStatus)

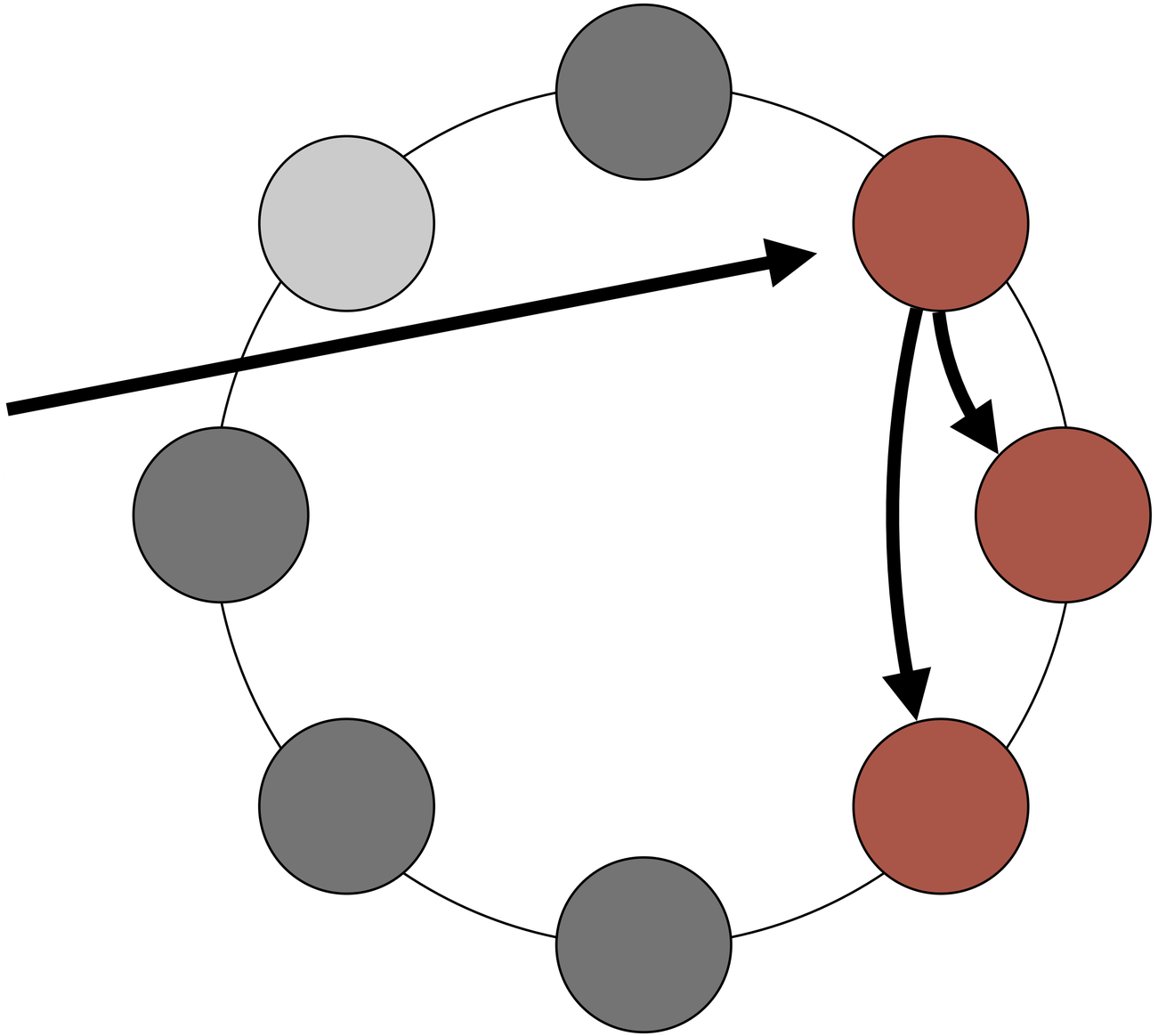
hashFunction()

getRingStatus()



hashFunction()

getRingStatus()



LE PRINCIPE

- Faire le hashing **côté client**
- Fonction de hash par défaut:
  - `“chash_keyfun”:{"mod":"riak_core_util", "fun":"chash_std_keyfun"},`
- Allons voir le code sur github

LA FONCTION DE HASH

```
%% @spec chash_std_keyfun(BKey :: riak_object:bkey()) -> chash:index()
%% @doc Default object/ring hashing fun, direct passthrough of bkey.
chash_std_keyfun({Bucket, Key}) -> chash:key_of({Bucket, Key}).
```

```
%% @doc Given any term used to name an object, produce that object's key
%%      into the ring.  Two names with the same SHA-1 hash value are
%%      considered the same name.
-spec key_of(ObjectName :: term()) -> index().
key_of(ObjectName) ->
    sha(term_to_binary(ObjectName)).
```

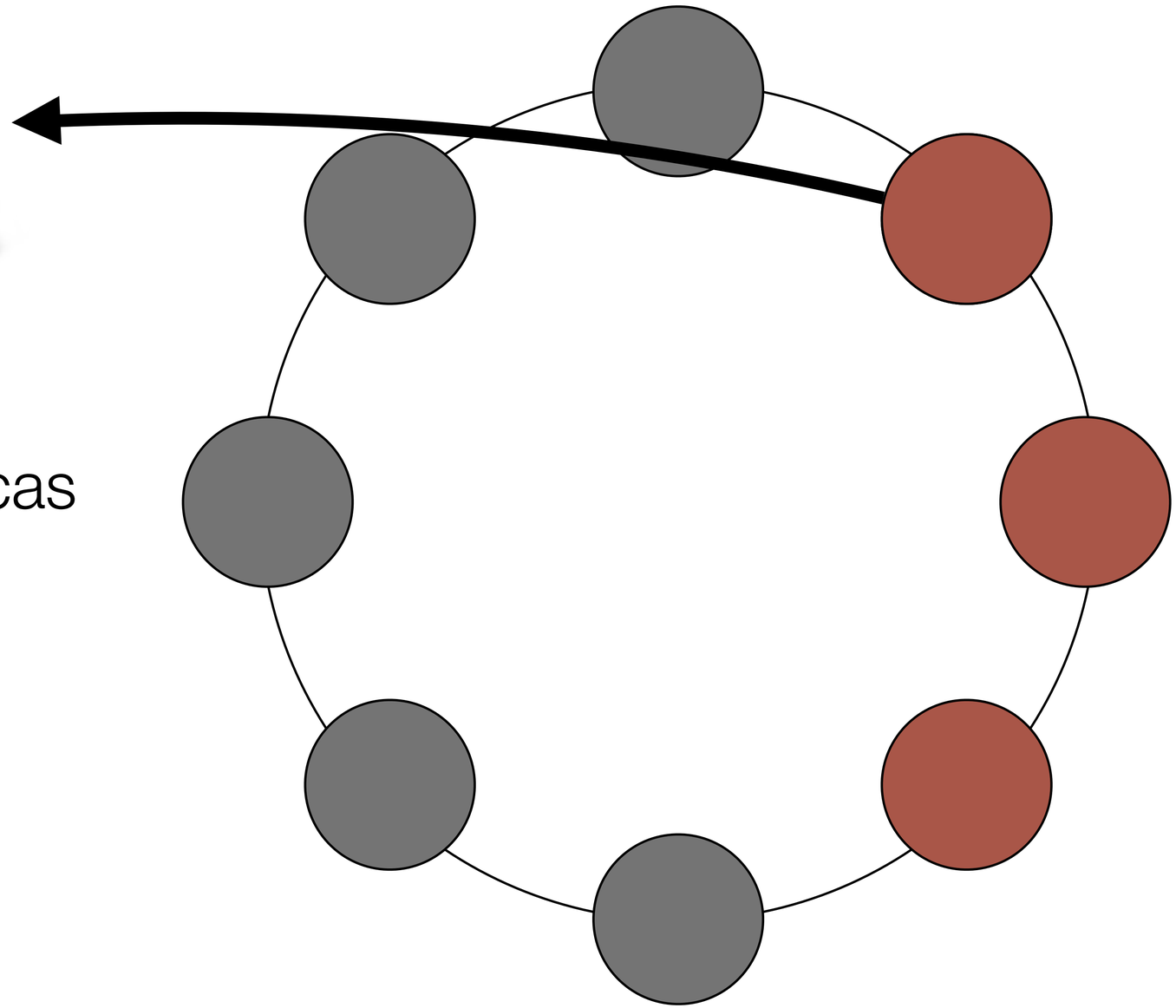
- riak_core/src/chash.erl

LA FONCTION DE HASH

- Facile à réimplémenter côté client:
- sha1 représenté en BigInt
- 400899077082383952423314387779798054553098649 6
- entre 0 et 2^160

ETAT DU RING

```
$ curl -s  -XGET http://$host:8098/riak_preflists/myring
{
  "0"                                                  :"riak@host-16"
  "570899077082383952423314387779798054553098649 6"    :"riak@host-17"
  "1141798154164767904846628775559596109106197299 2":"riak@host-18"
  "1712697231247151857269943163339394163659295948 8":"riak@host-19"
  …etc…
}
```



- **GET** - bon cas
- **n_val = 1**
- exterieur =
  0 x intérieur

RESULTAT

- Utilisation réseau encore diminuée
- Particulièrement pour les GETs

ATTENTION

- Possible seulement si
  - Noeuds sont monitorés (ici Zookeeper)
  - En cas d’échec, retour à noeud au hasard
  - Données lues de manière uniforme

CONCLUSION

CONCLUSION

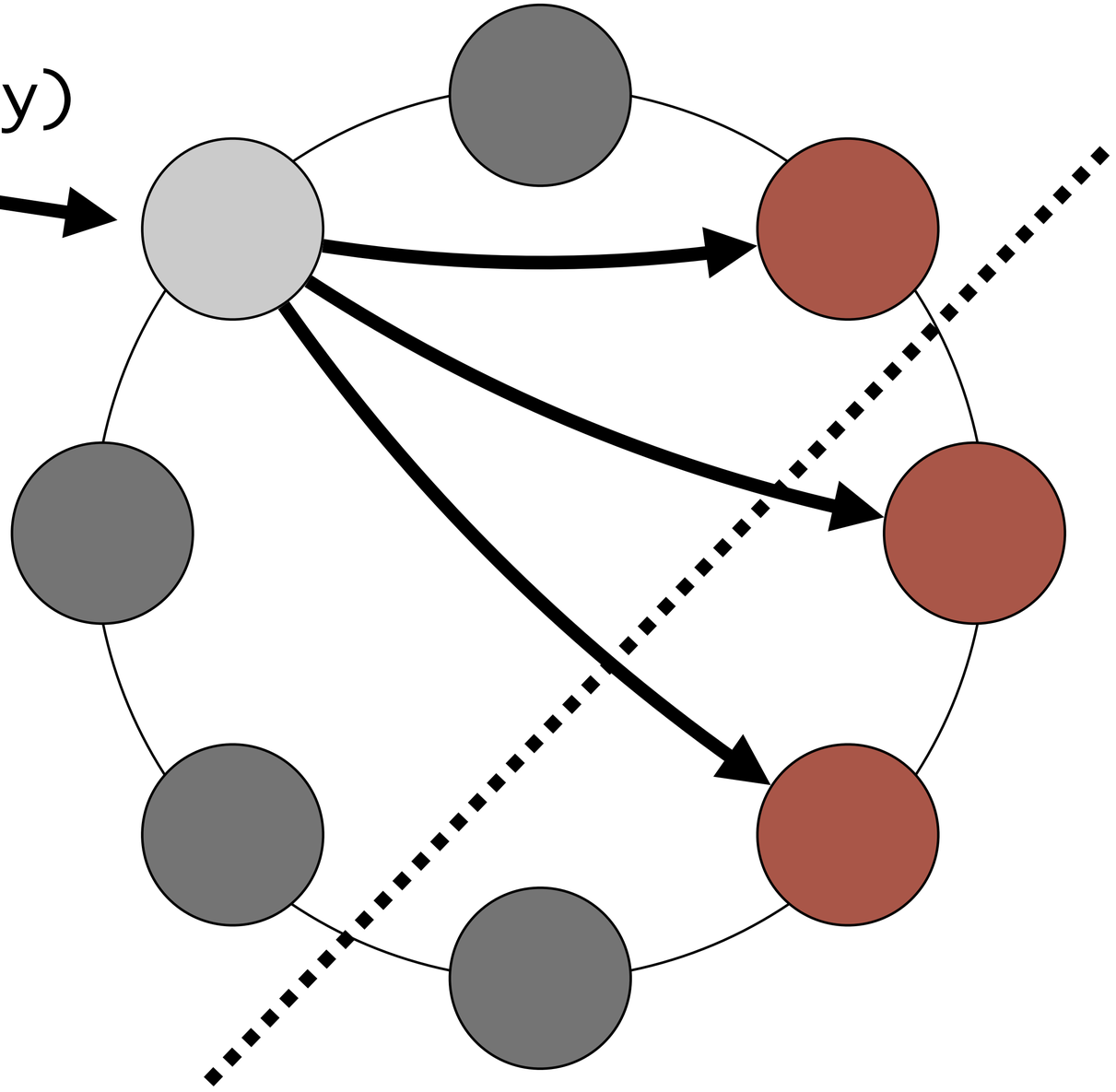
- Version Open Source de Riak
- Pas de training, autodidacte, équipe très petite
- Riak est une bonne solution
  - Robuste, rapide, scalable, facile
  - Très flexible and hackable
- Nous permet de continuer à scaler chez Booking.com

Q&A

 @damsieboy

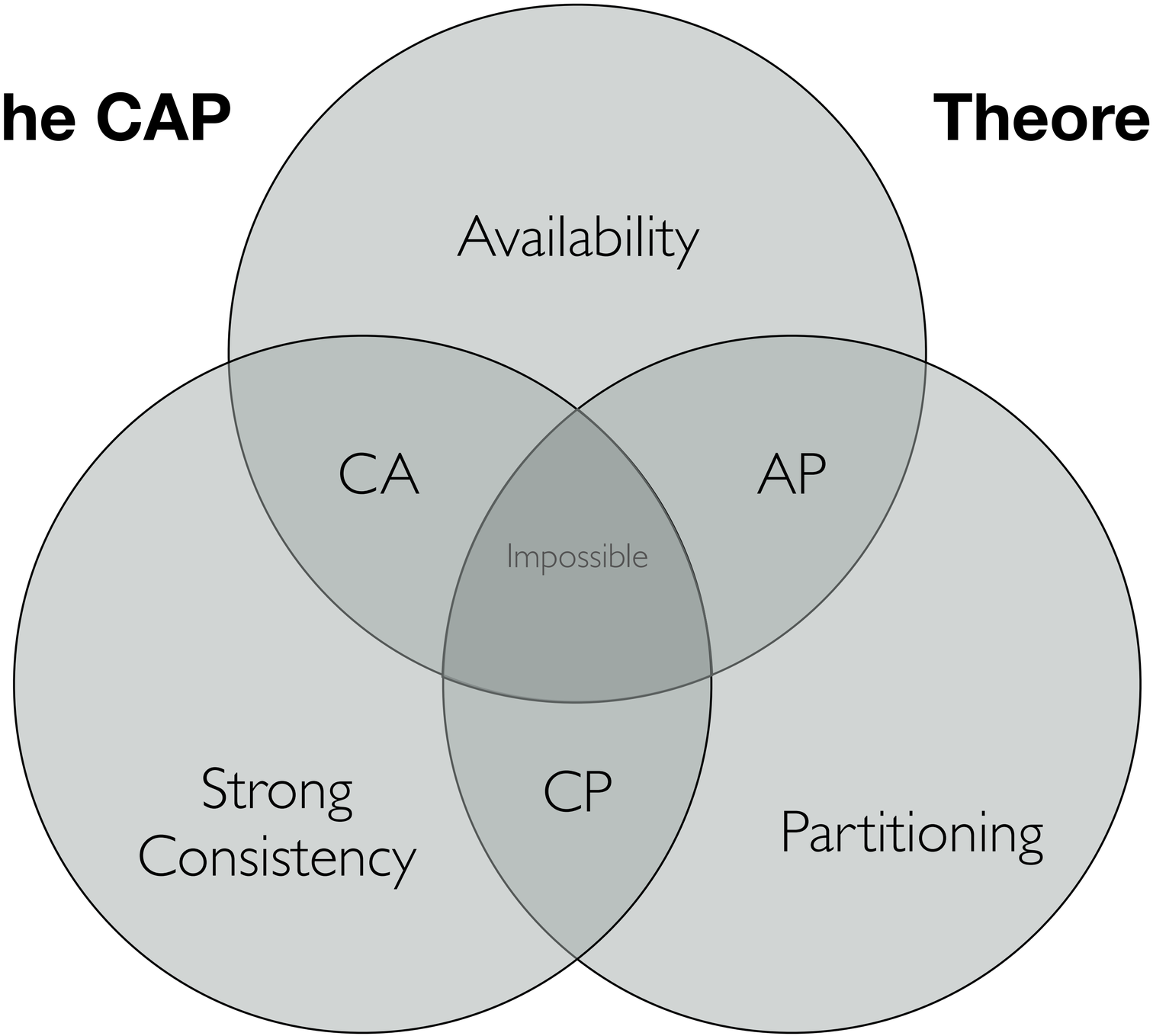
EXTRA: RIAK ET LE “CAP THEOREM”

hash(key)

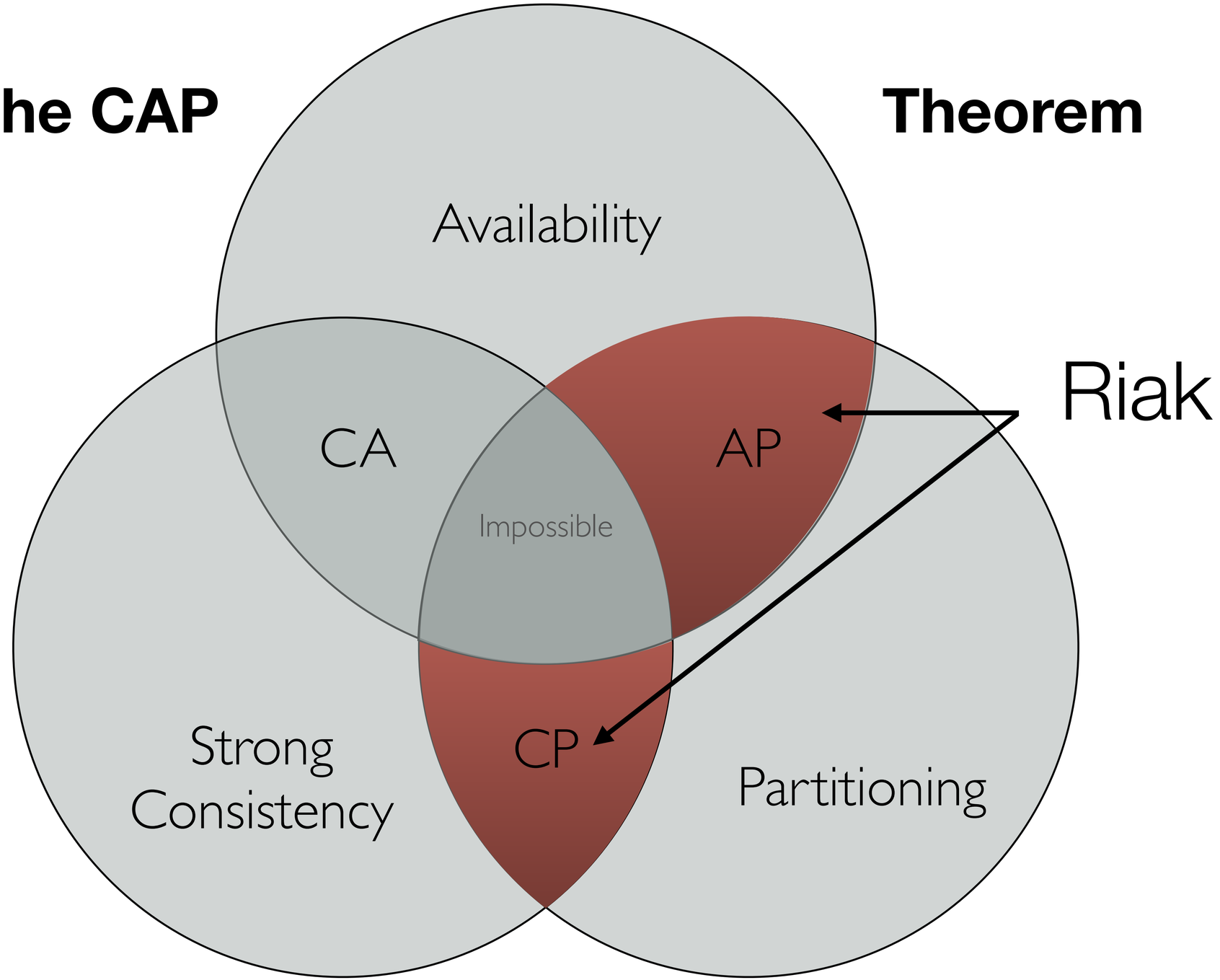


partition

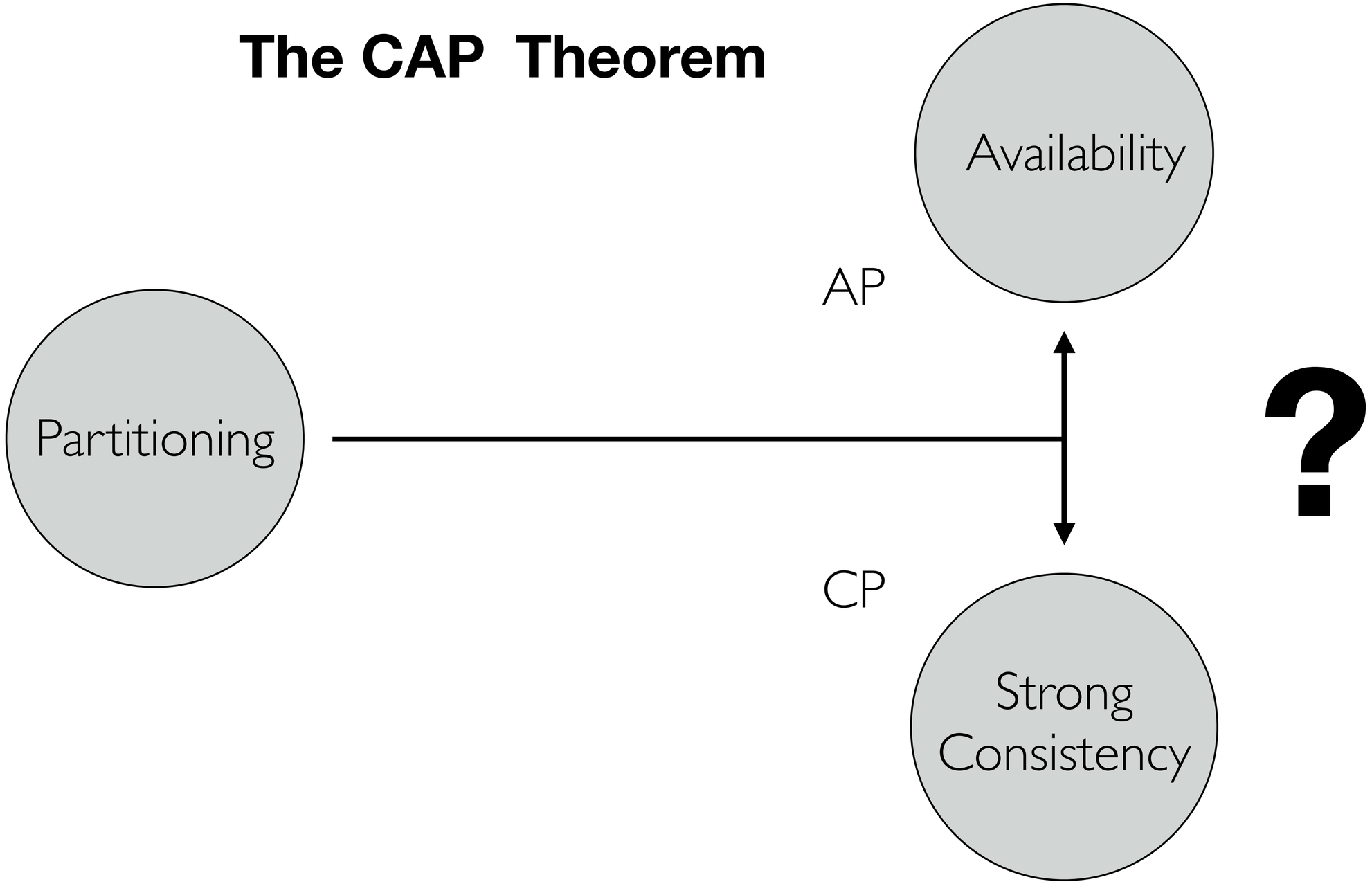
The CAP Theorem

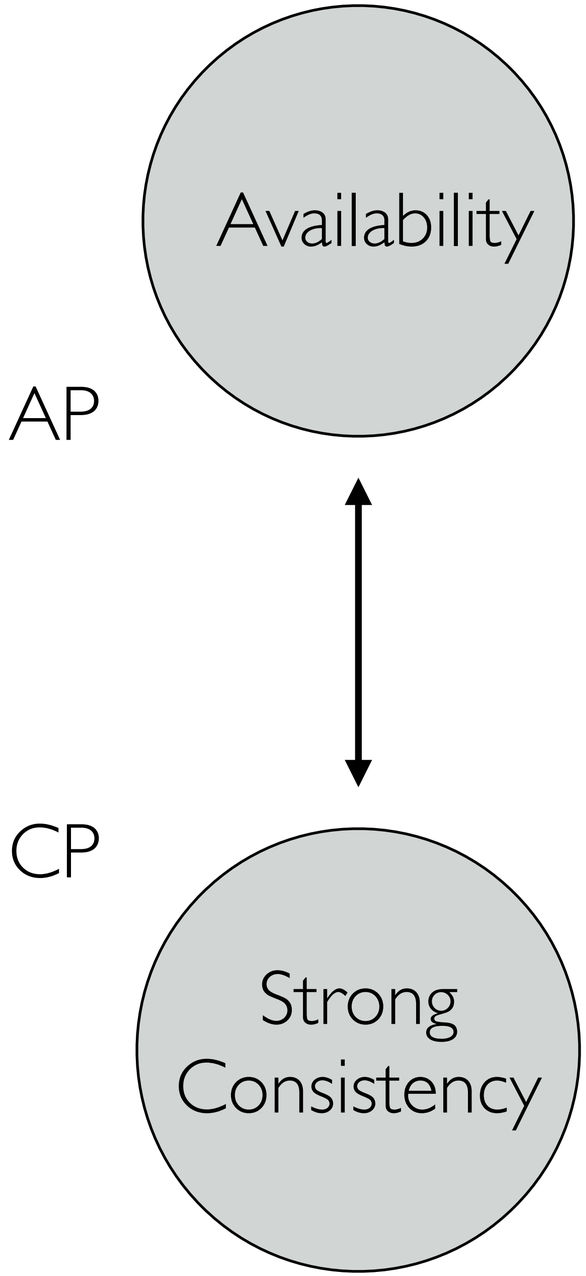


The CAP Theorem



The CAP Theorem





Eventual consistency
+ full availability

Riak does both

Strong consistency
+ partial availability