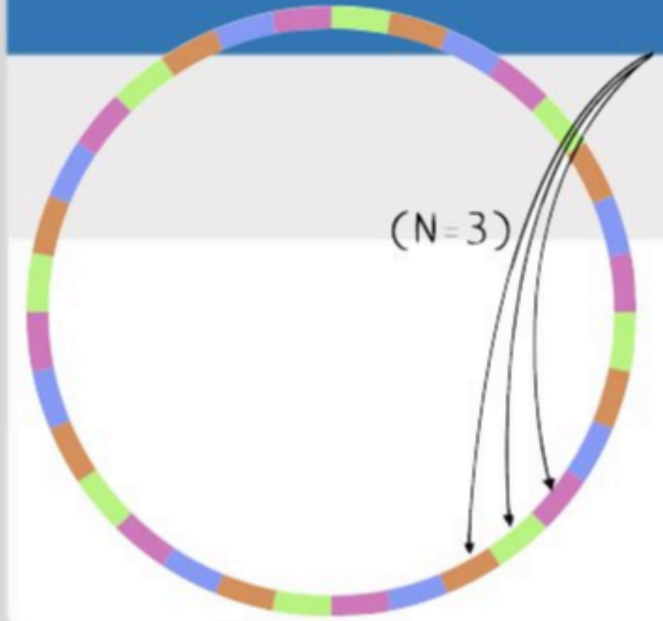
NoSQL Databases

Shamelessly stolen from Lorenzo Alberton
by Vincent Leroy

Lorenzo Alberton
@lorenzoalberton

NoSQL Databases: Why, what and when



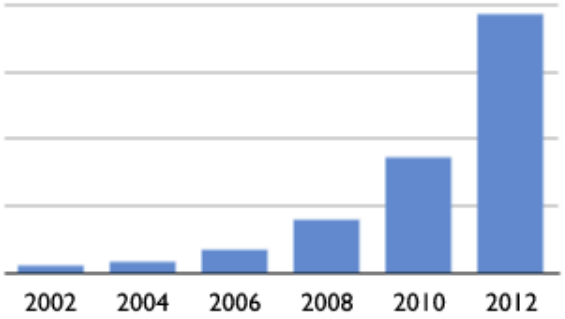
NoSQL Databases Demystified

PHP UK Conference, 25th February 2011

NoSQL: Why

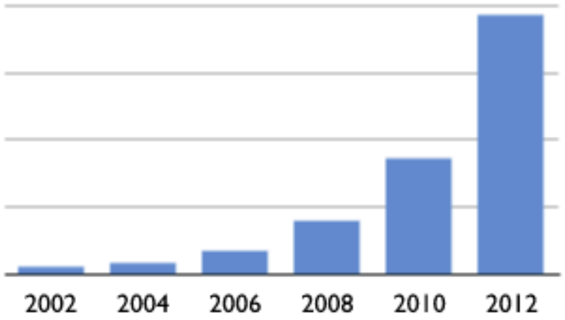
Scalability, Concurrency, New trends

New Trends



Big data

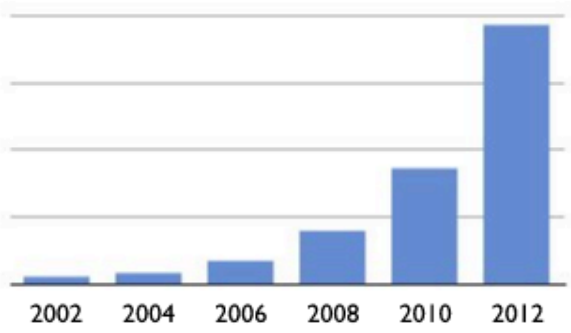
New Trends



Big data

Concurrency

New Trends

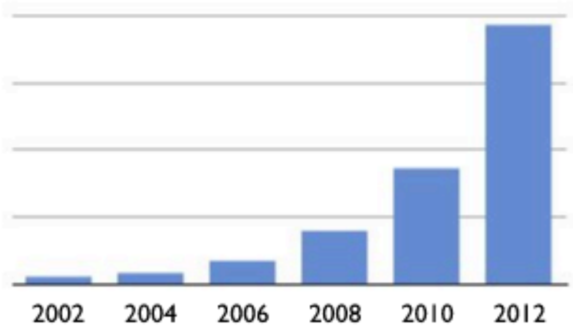


Big data

Connectivity

Concurrency

New Trends



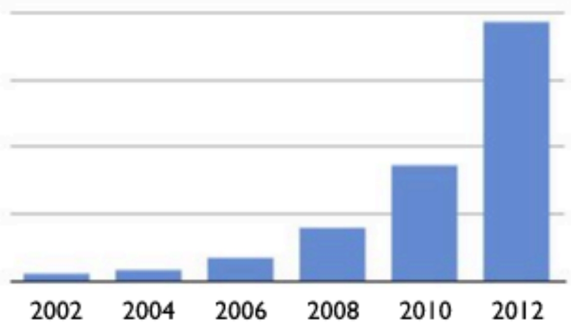
Big data

Connectivity

Concurrency

Diversity

New Trends



Big data

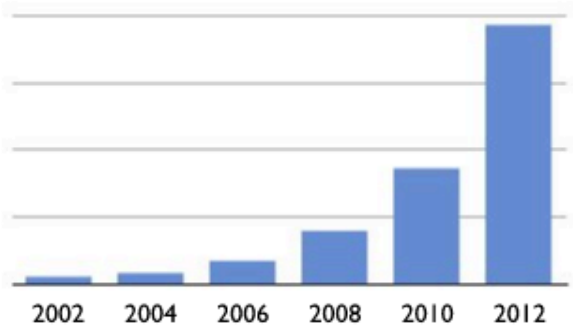
Connectivity

P2P Knowledge

Concurrency

Diversity

New Trends



Big data

Connectivity

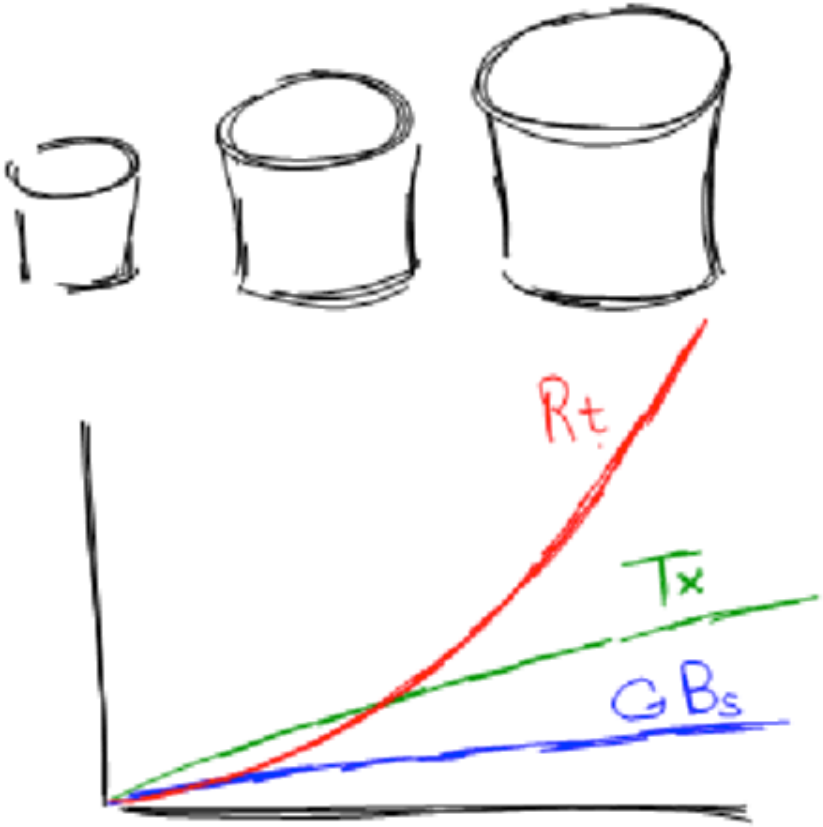
P2P Knowledge

Concurrency

Diversity

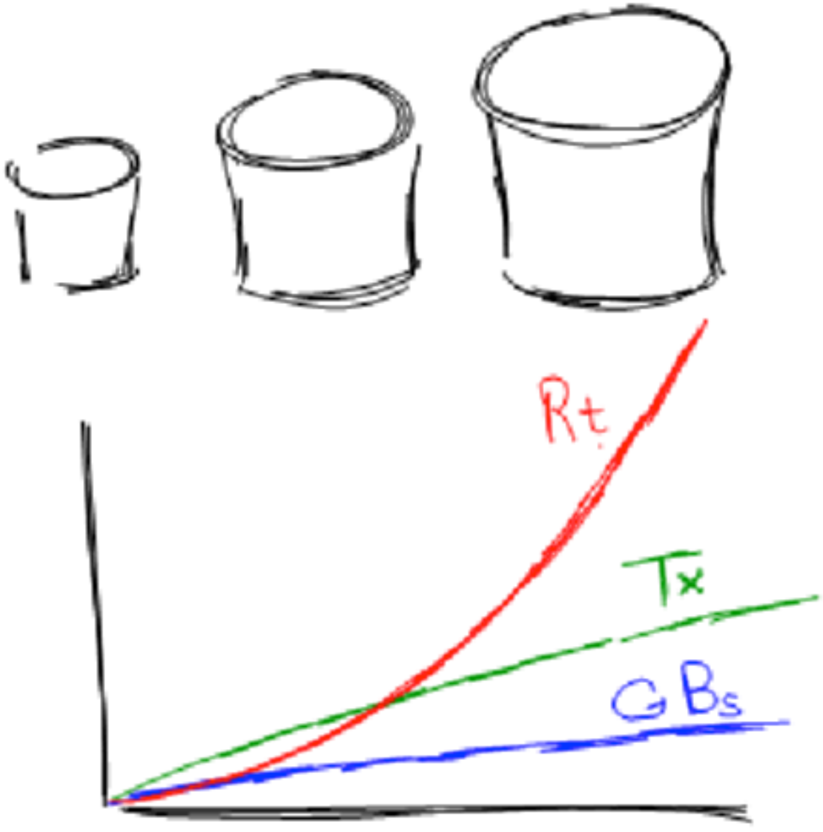
Cloud-Grid

What’s the problem with RDBMS’s?



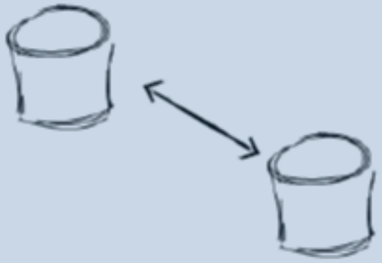
http://www.codefutures.com/database-sharding

What's the problem with RDBMS's?



http://www.codefutures.com/database-sharding

Caching



Master/Slave

Master/Master

Cluster

Table Partitioning

Federated Tables

Sharding

Distributed DBs

What's the problem with RDBMS's?

Quick Comparison

Overview from 10,000 feet

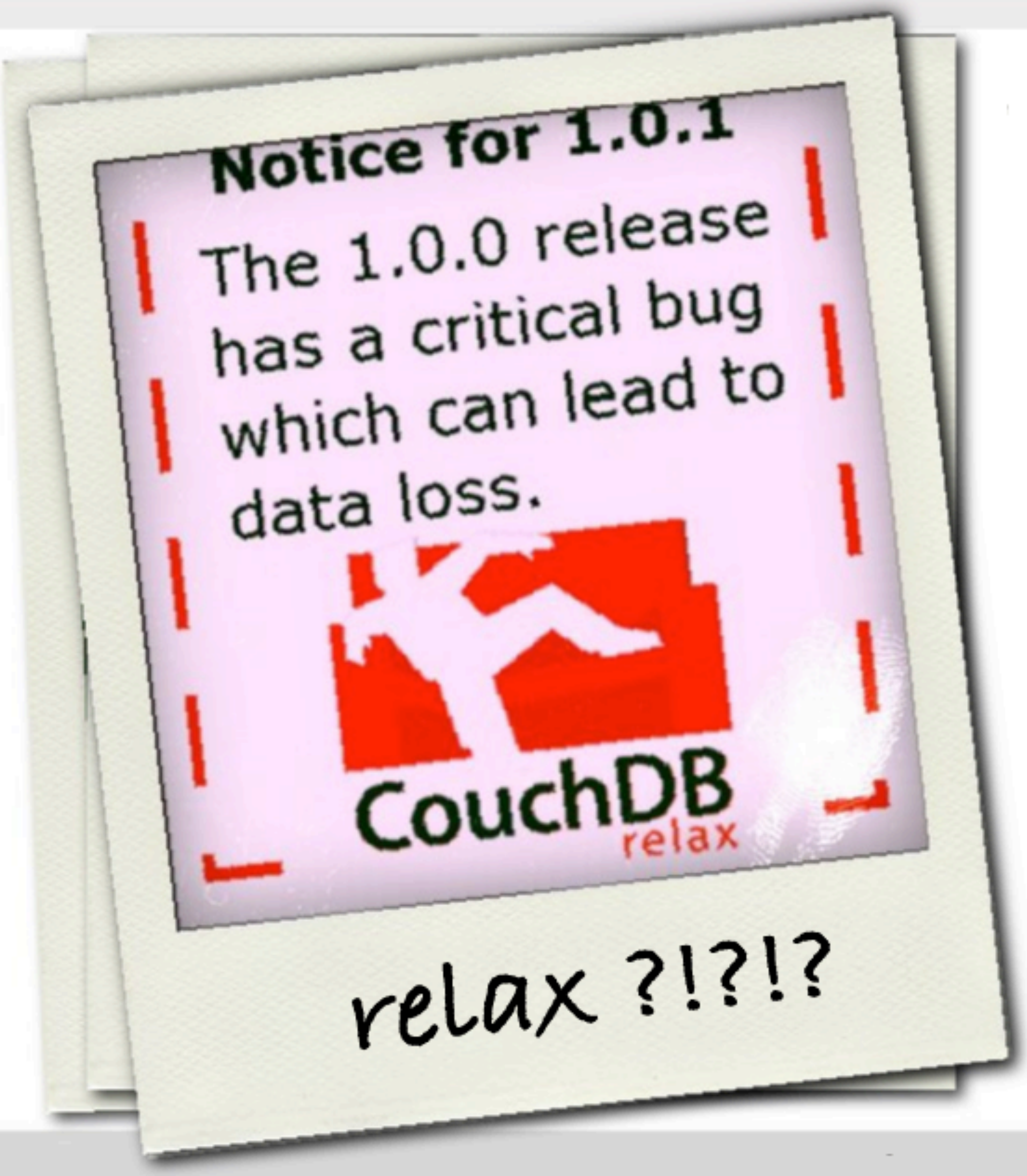
(random impressions from the interwebs)

MongoDB is web-scale

...but /dev/null
is even better!

Cassandra is teh schnitz

CouchDB: Relax!



Notice for 1.0.1

The 1.0.0 release has a critical bug which can lead to data loss.

CouchDB
relax

relax ?!?!?

No, seriously...*

(*) Not another “Mine is bigger” comparison, please

A little theory

Fundamental Principles
of (Distributed) Databases

http://www.timbarcz.com/blog/PassionInProgrammers.aspx

ACID

ATOMICITY: All or nothing

CONSISTENCY: Any transaction will take the db from one consistent state to another, with no broken constraints (referential integrity)

ISOLATION: Other operations cannot access data that has been modified during a transaction that has not yet completed

DURABILITY: Ability to recover the committed transaction updates against any kind of system failure (transaction log)

Isolation Levels, Locking & MVCC

Isolation |ˌīsəˈlā sh ən| noun

Property that defines how/when the changes made by one operation become visible to other concurrent operations

Isolation Levels, Locking & MVCC

Isolation |ˌīsəˈlā sh ən| noun

Property that defines how/when the changes made by one operation become visible to other concurrent operations

- **SERIALIZABLE**
  All transactions occur in a completely isolated fashion, as if they were executed serially

Isolation Levels, Locking & MVCC

Isolation |ˌīsəˈlā sh ən| noun

Property that defines how/when the changes made by one operation become visible to other concurrent operations

- **SERIALIZABLE**
  All transactions occur in a completely isolated fashion, as if they were executed serially

- **REPEATABLE READ**
  Multiple SELECT statements issued in the same transaction will always yield the same result

Isolation Levels, Locking & MVCC

Isolation |ˌīsəˈlā sh ən| noun

Property that defines how/when the changes made by one operation become visible to other concurrent operations

- **SERIALIZABLE**
  All transactions occur in a completely isolated fashion, as if they were executed serially
- **READ COMMITTED**
  A lock is acquired only on the rows currently read/updated
- **REPEATABLE READ**
  Multiple SELECT statements issued in the same transaction will always yield the same result

Isolation Levels, Locking & MVCC

Isolation |ˌīsəˈlā sh ən| noun

Property that defines how/when the changes made by one operation become visible to other concurrent operations

- **SERIALIZABLE**
  All transactions occur in a completely isolated fashion, as if they were executed serially

- **READ COMMITTED**
  A lock is acquired only on the rows currently read/updated

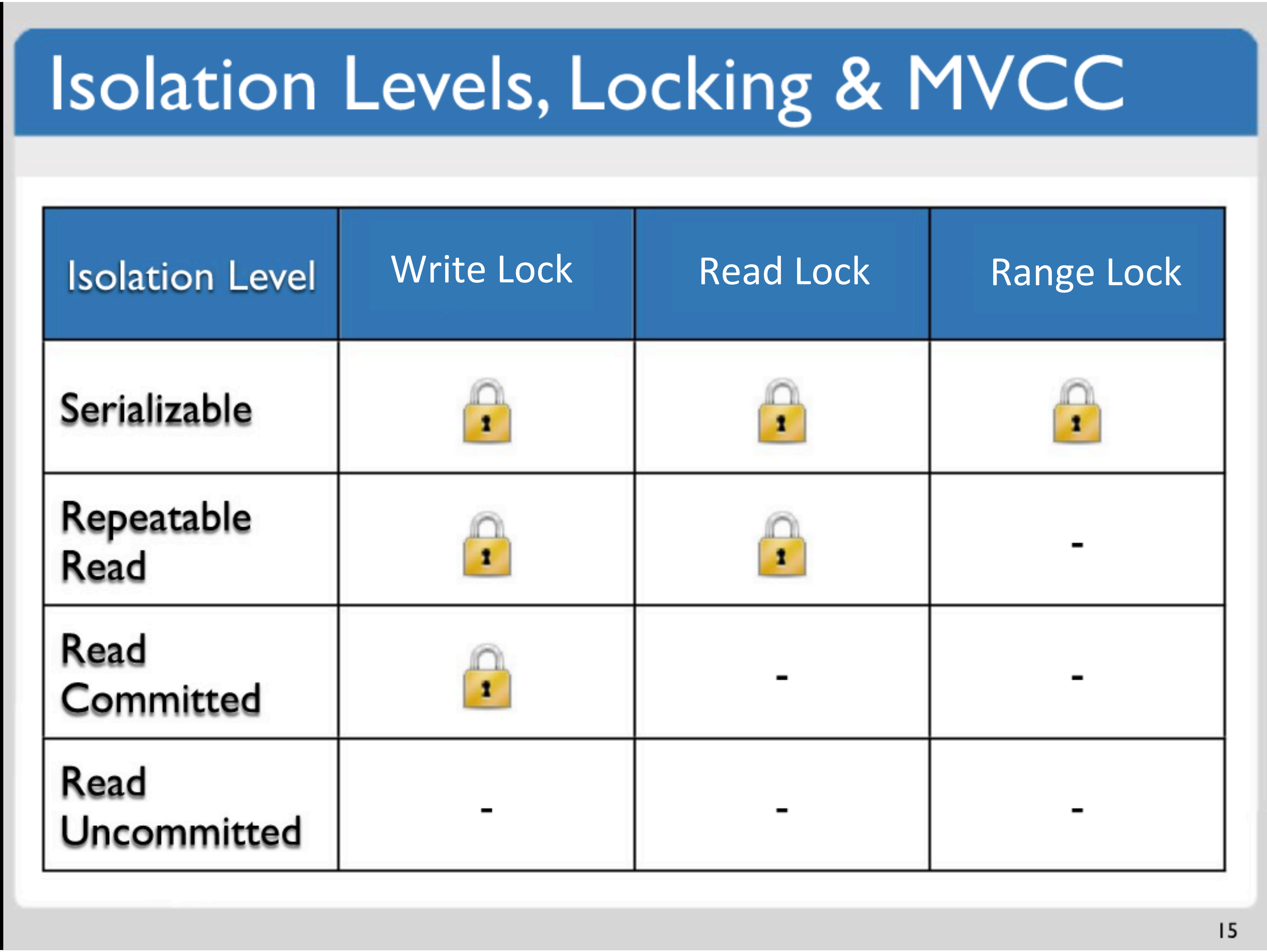
- **REPEATABLE READ**
  Multiple SELECT statements issued in the same transaction will always yield the same result

- **READ UNCOMMITTED**
  A transaction can access uncommitted changes made by other transactions

Isolation Levels, Locking & MVCC

Isolation Level	Dirty Reads	Non-repeatable reads	Phantoms
Serializable	-	-	-
Repeatable Read	-	-	⚠
Read Committed	-	⚠	⚠
Read Uncommitted	⚠	⚠	⚠

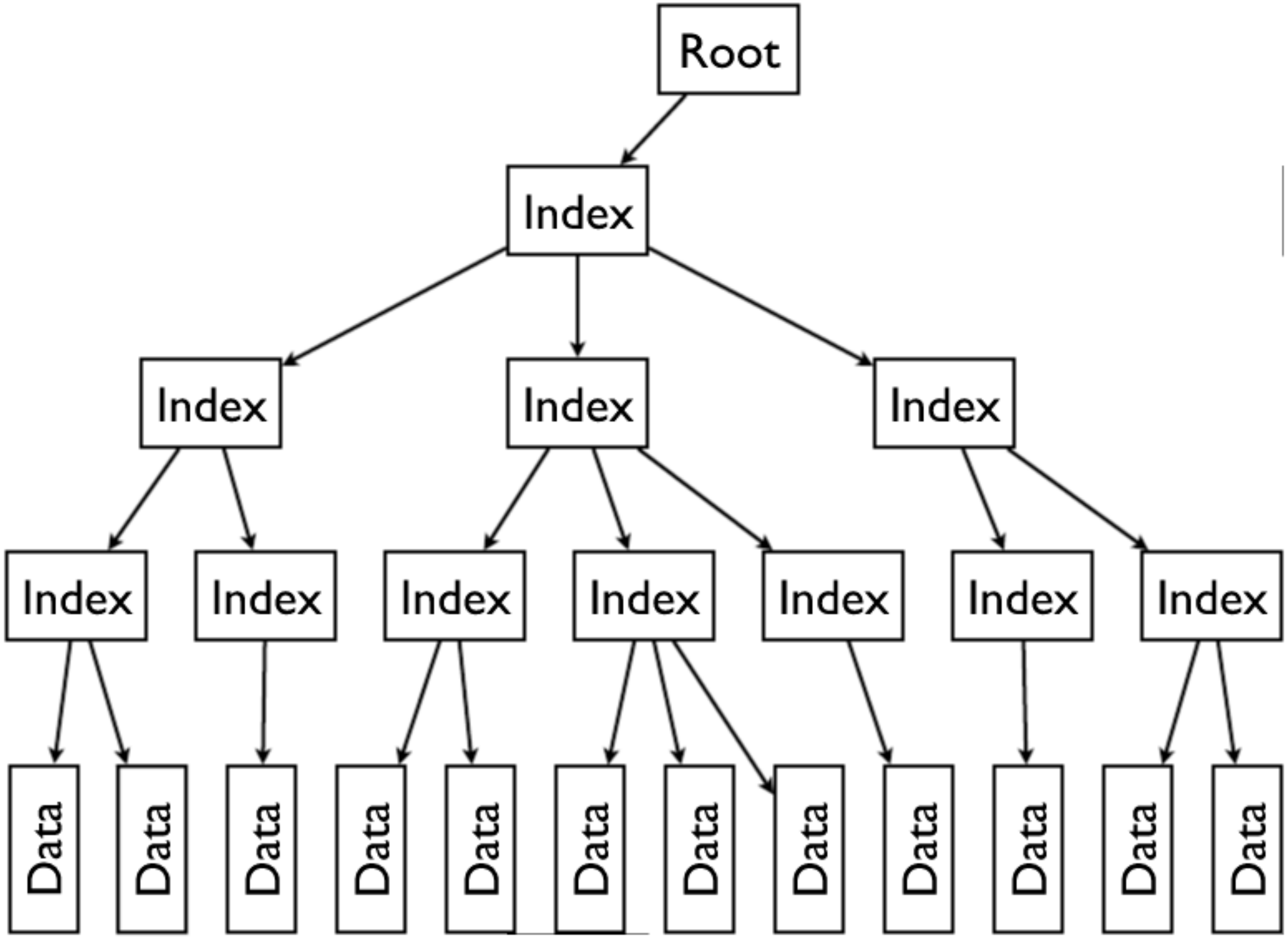
http://www.adayinthelifeof.nl/2010/12/20/innodb-isolation-levels/



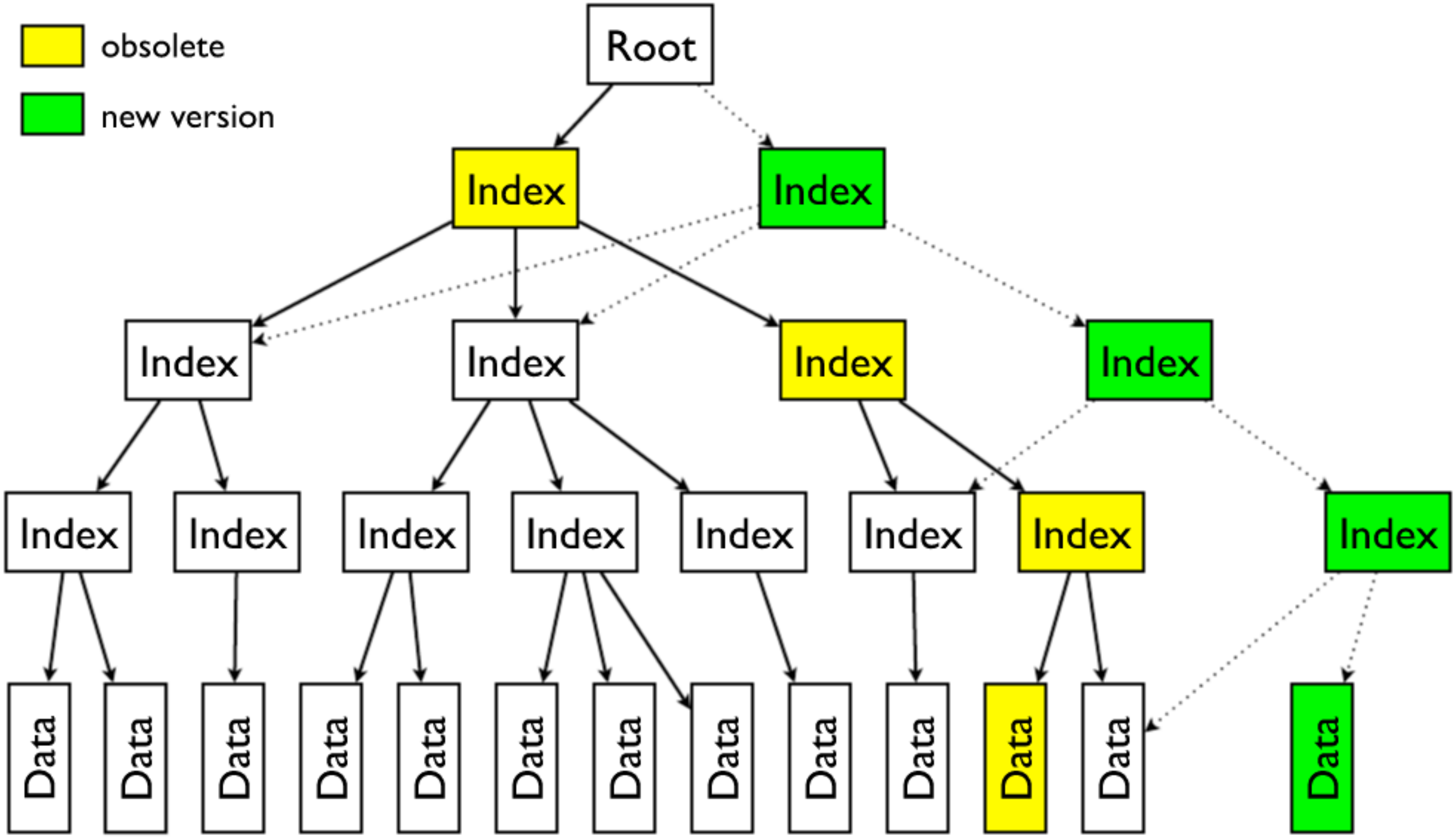
Isolation Levels, Locking & MVCC

Isolation Level	Write Lock	Read Lock	Range Lock
Serializable	🔒	🔒	🔒
Repeatable Read	🔒	🔒	-
Read Committed	🔒	-	-
Read Uncommitted	-	-	-

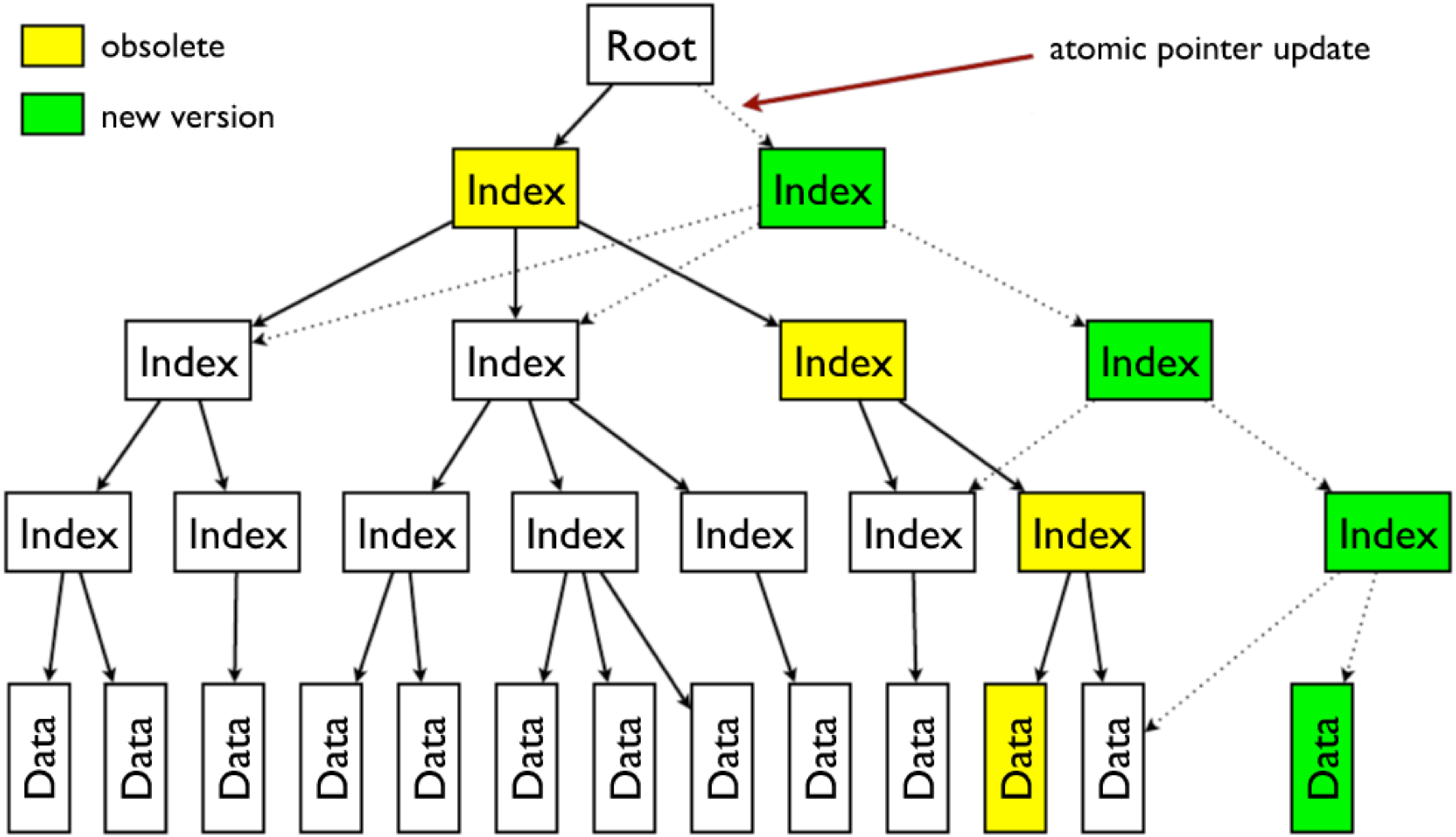
Multi-Version Concurrency Control



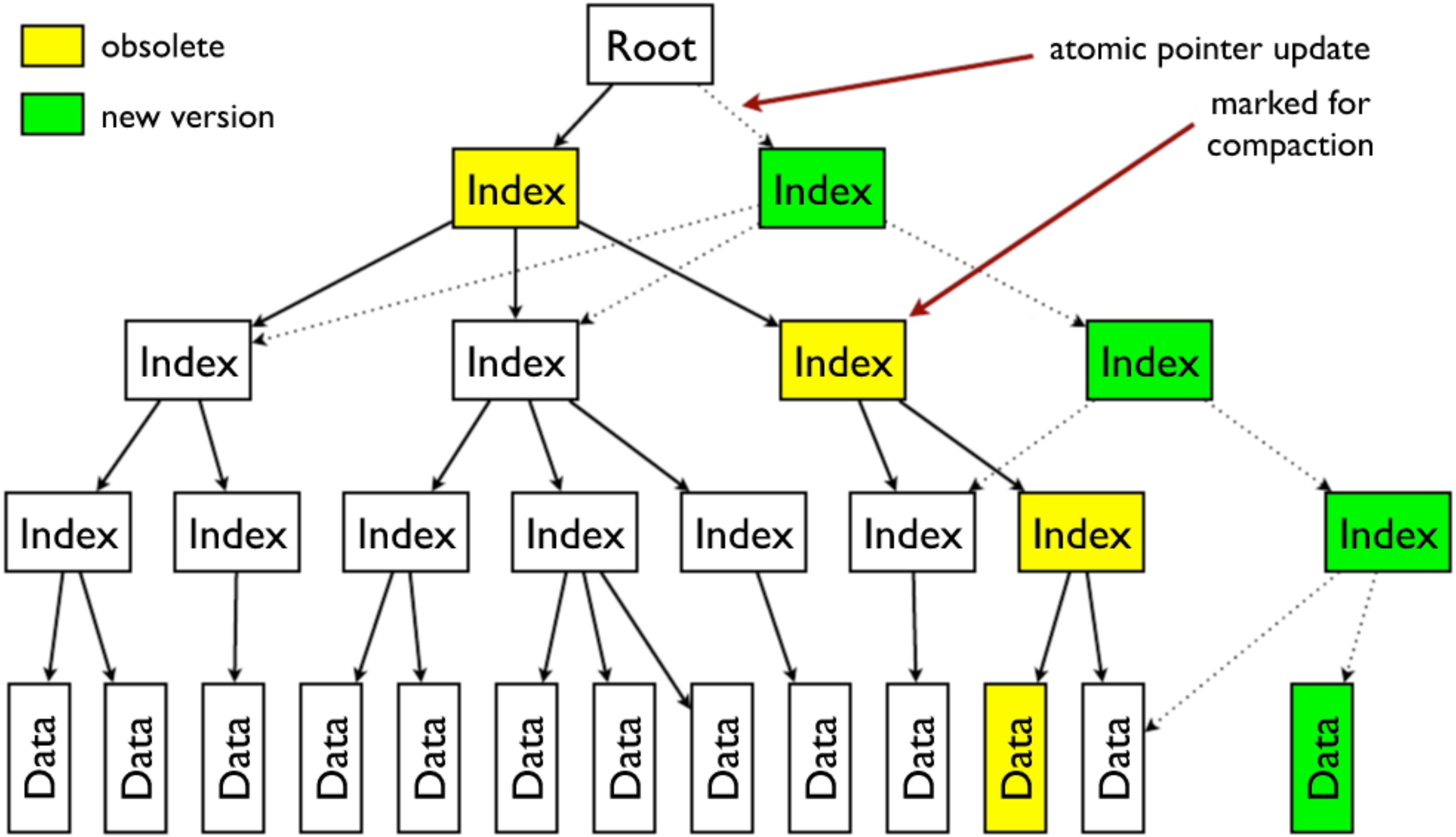
Multi-Version Concurrency Control



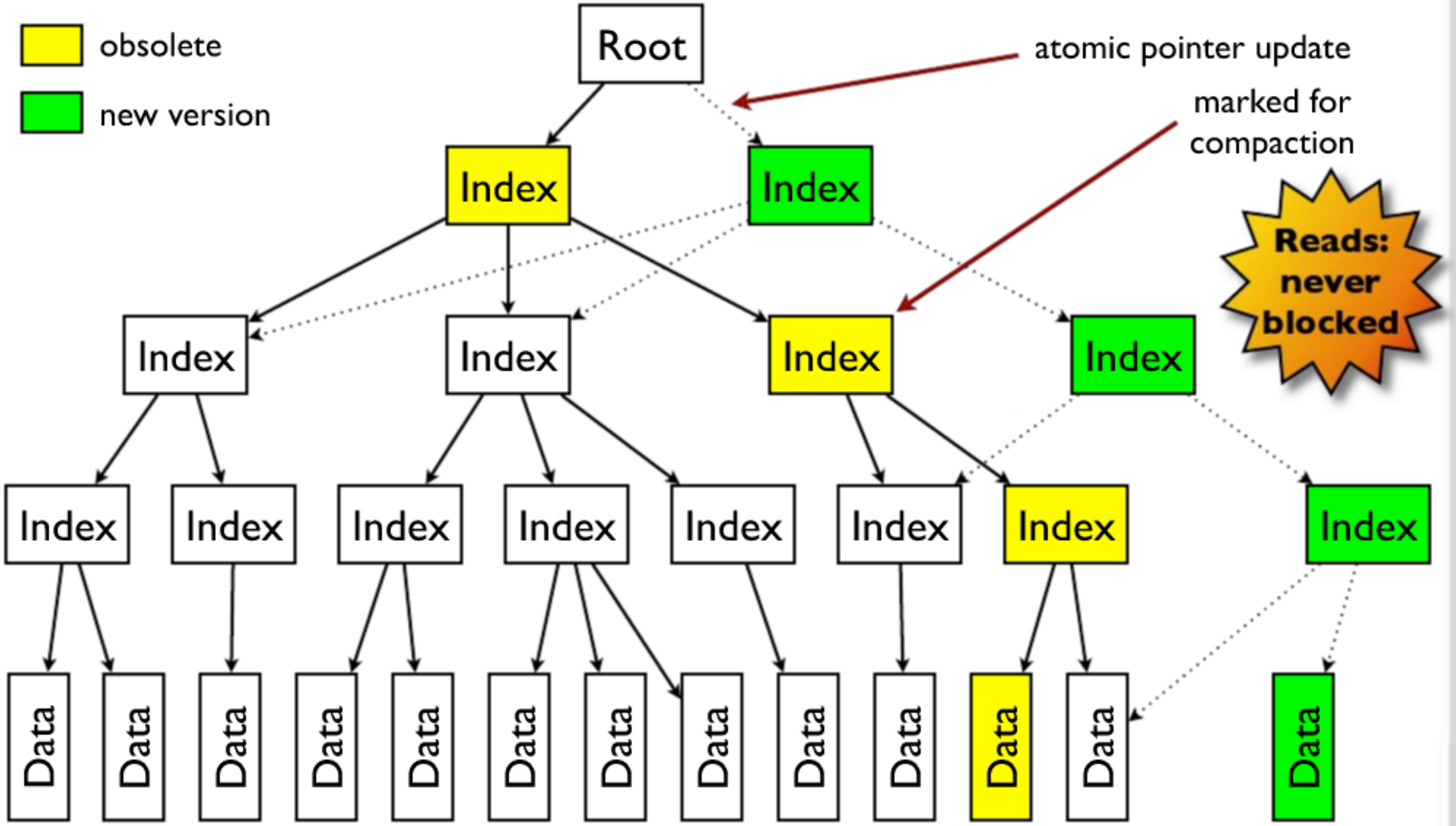
Multi-Version Concurrency Control



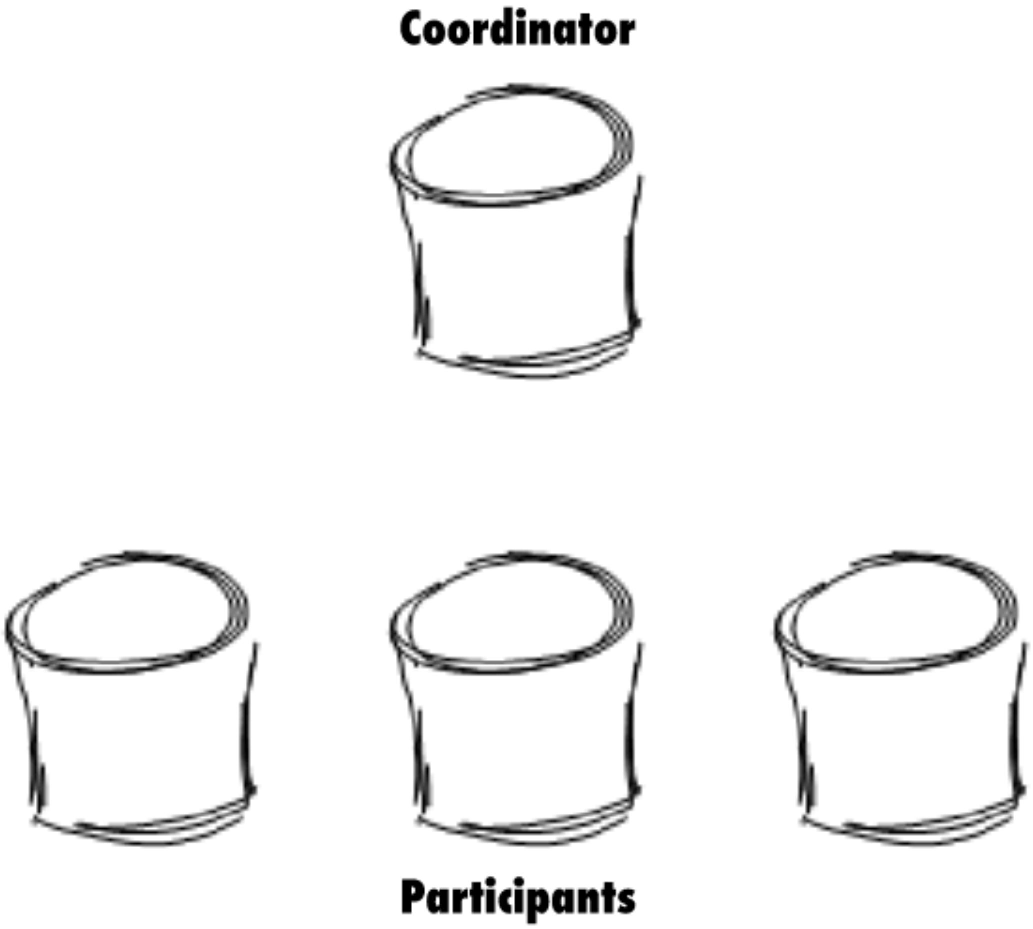
Multi-Version Concurrency Control



Multi-Version Concurrency Control

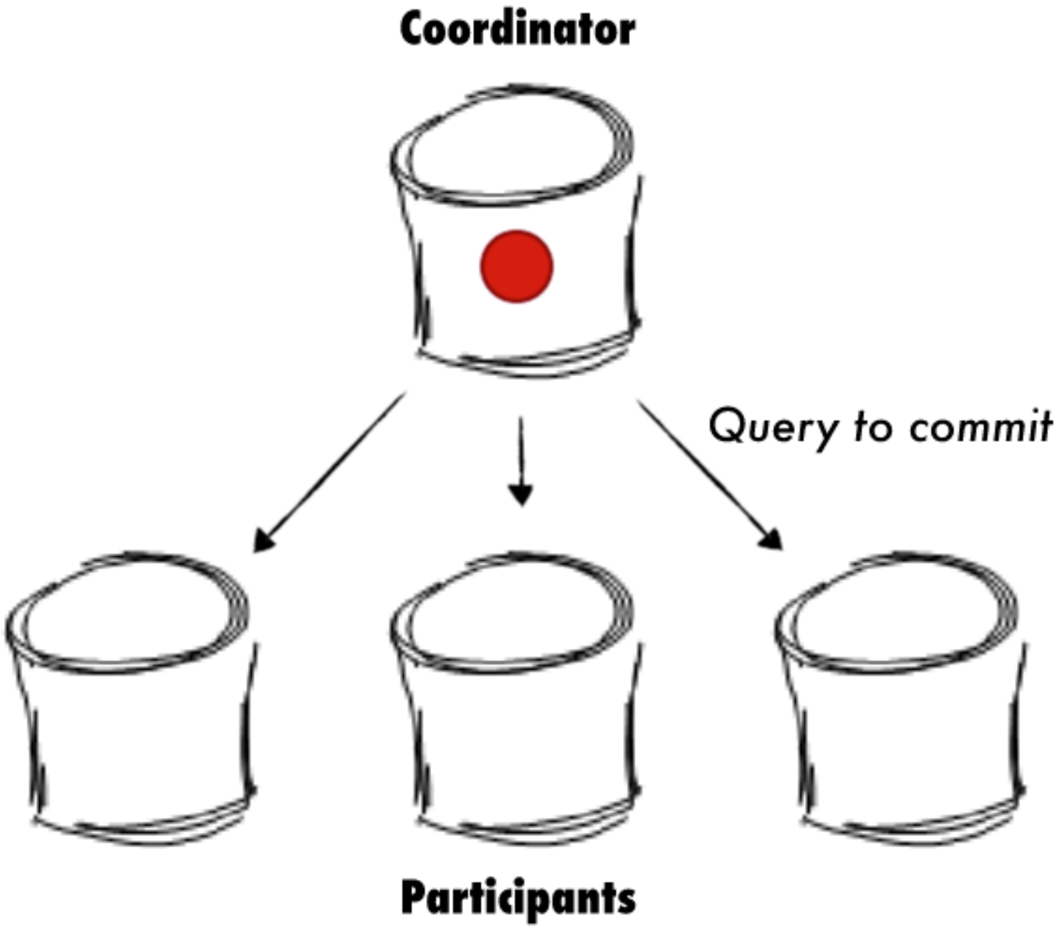


Distributed Transactions - 2PC



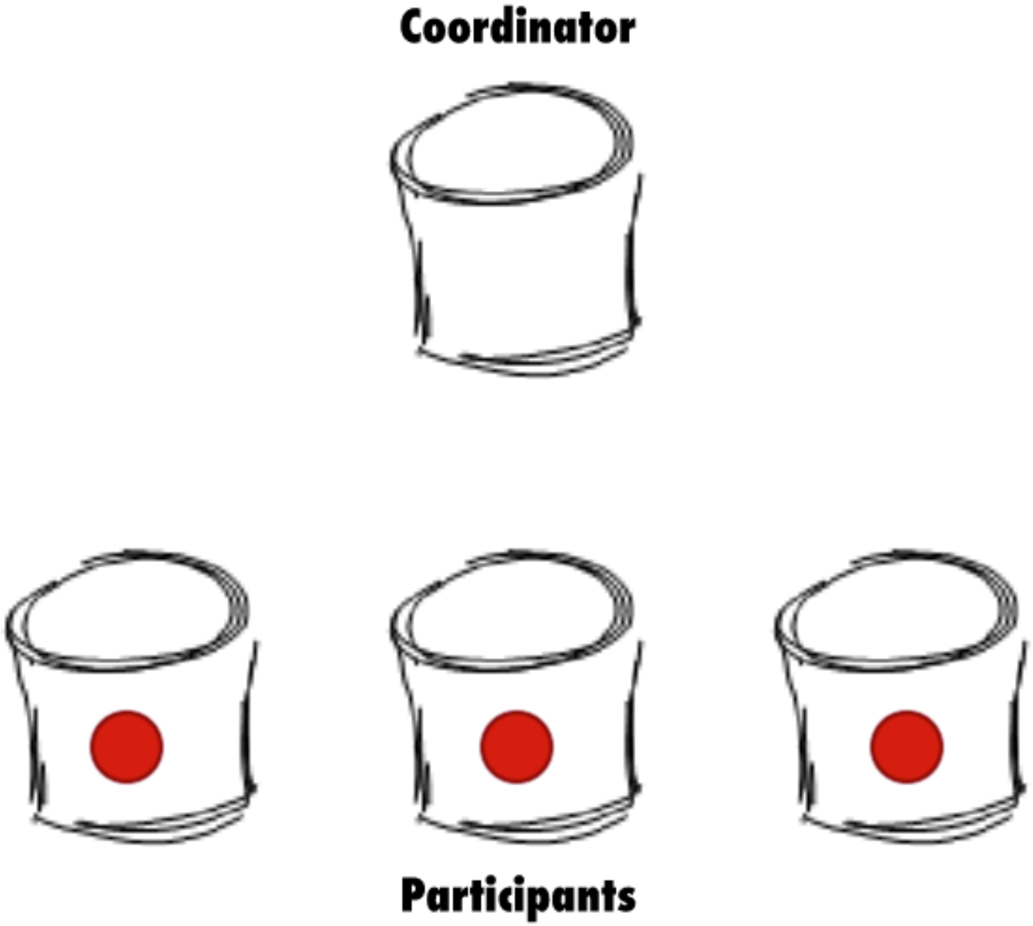
1) COMMIT REQUEST PHASE (voting phase)

Distributed Transactions - 2PC



1) COMMIT REQUEST PHASE
(voting phase)

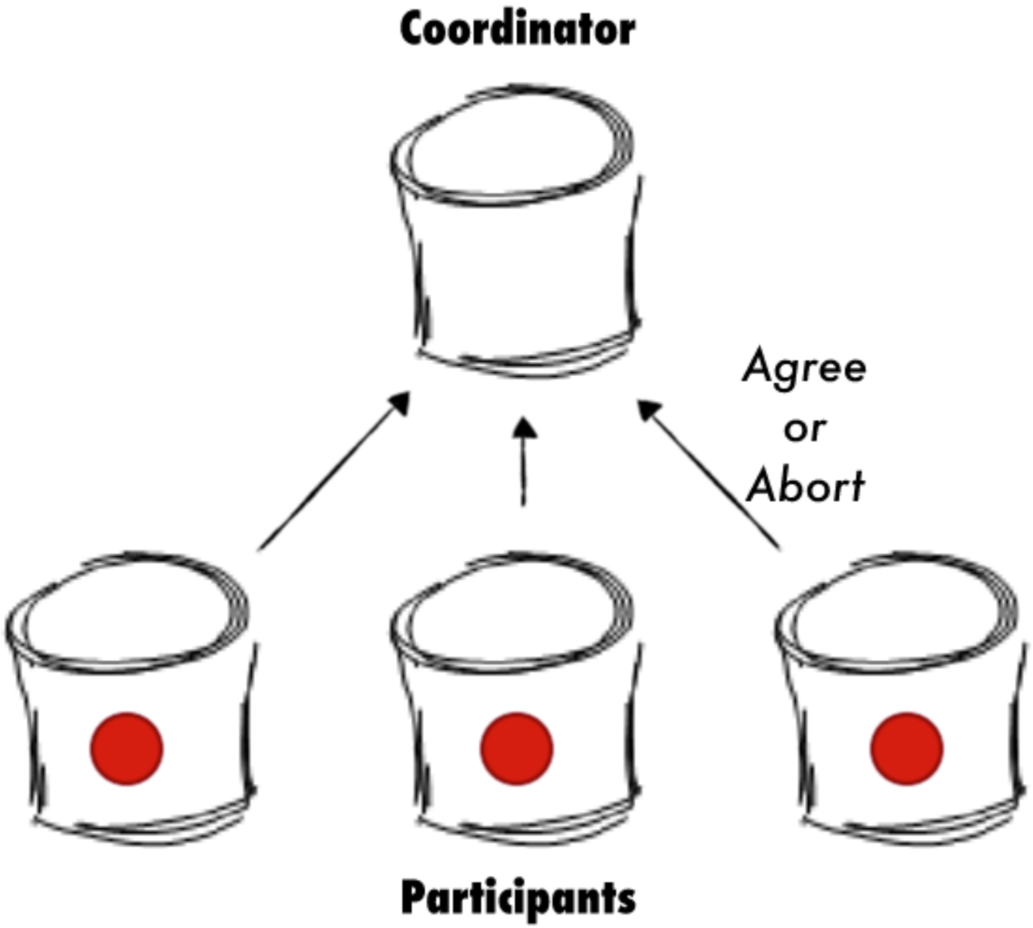
Distributed Transactions - 2PC



1) Exec Transaction up to the COMMIT request
2) Write entry to undo and redo logs

1) COMMIT REQUEST PHASE
(voting phase)

Distributed Transactions - 2PC



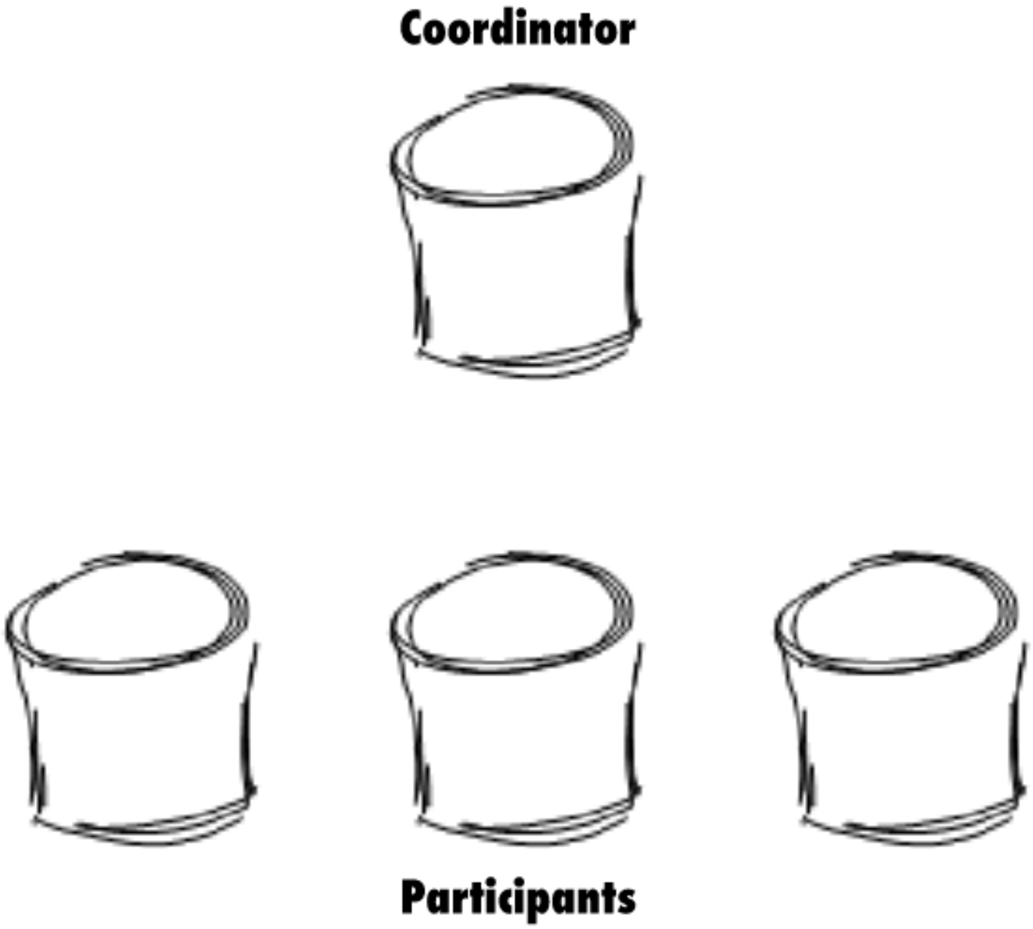
Coordinator

Agree
or
Abort

Participants

1) COMMIT REQUEST PHASE
(voting phase)

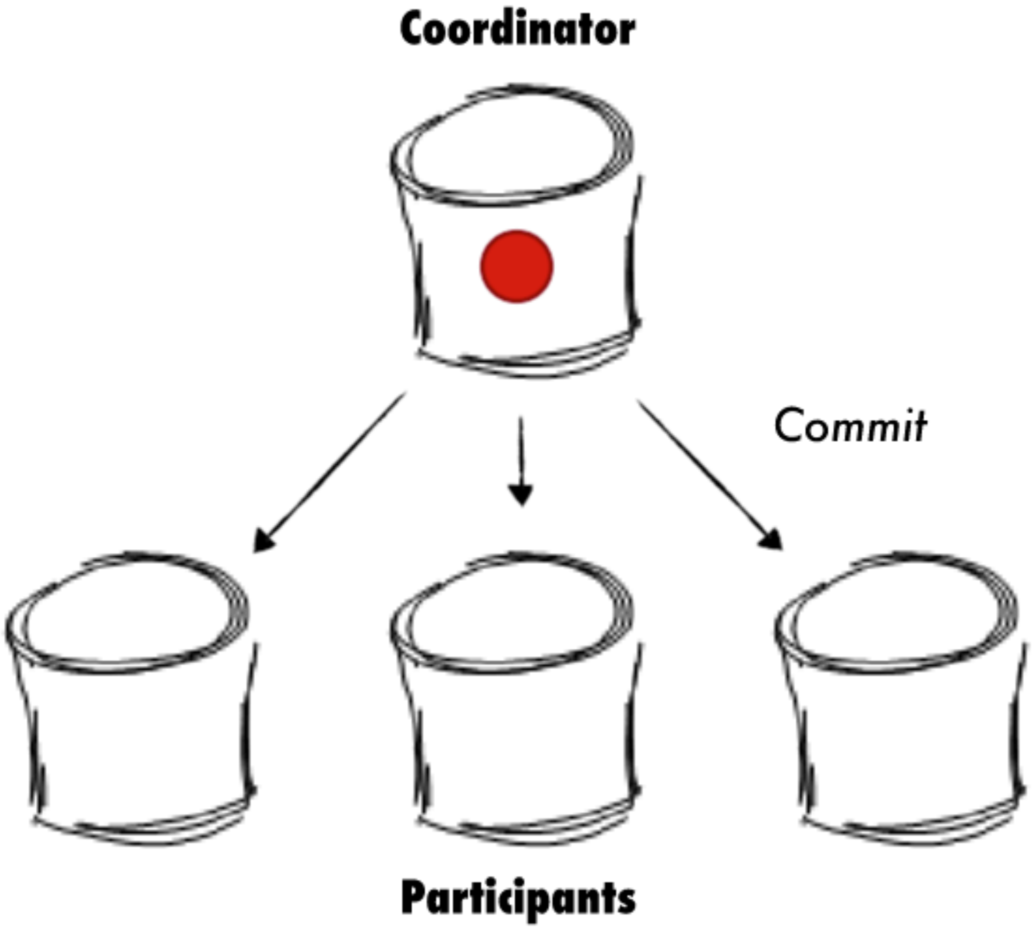
Distributed Transactions - 2PC



2) COMMIT PHASE (completion phase)

a) SUCCESS (agreement from all)

Distributed Transactions - 2PC



Coordinator

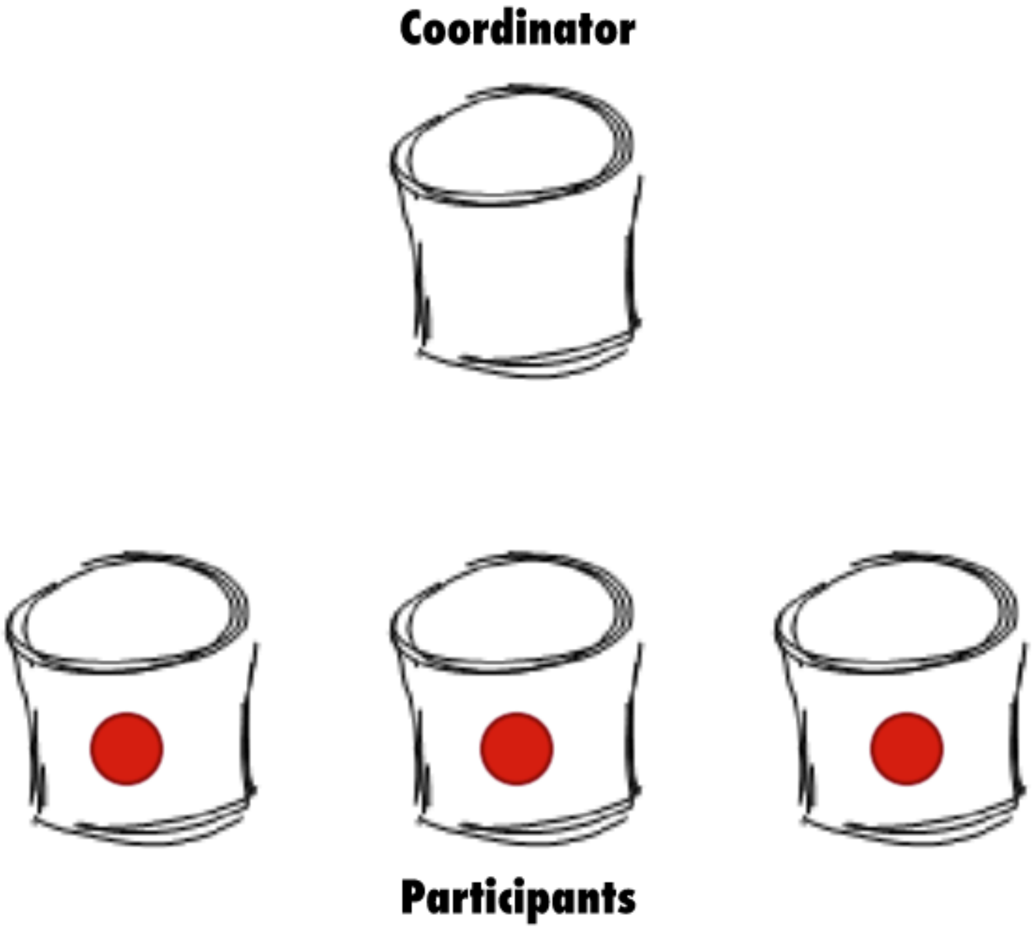
Commit

Participants

2) COMMIT PHASE (completion phase)

a) SUCCESS (agreement from all)

Distributed Transactions - 2PC

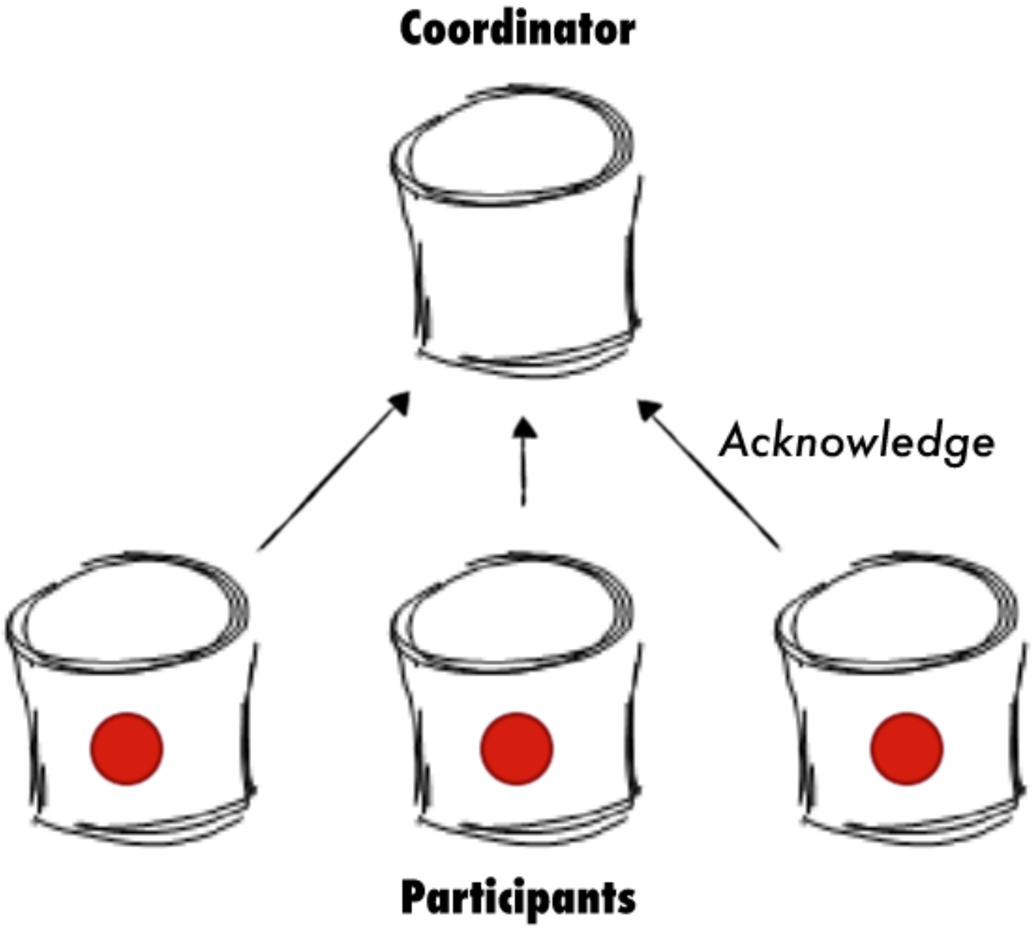


1) Complete operation
2) Release locks

2) COMMIT PHASE (completion phase)

a) SUCCESS (agreement from all)

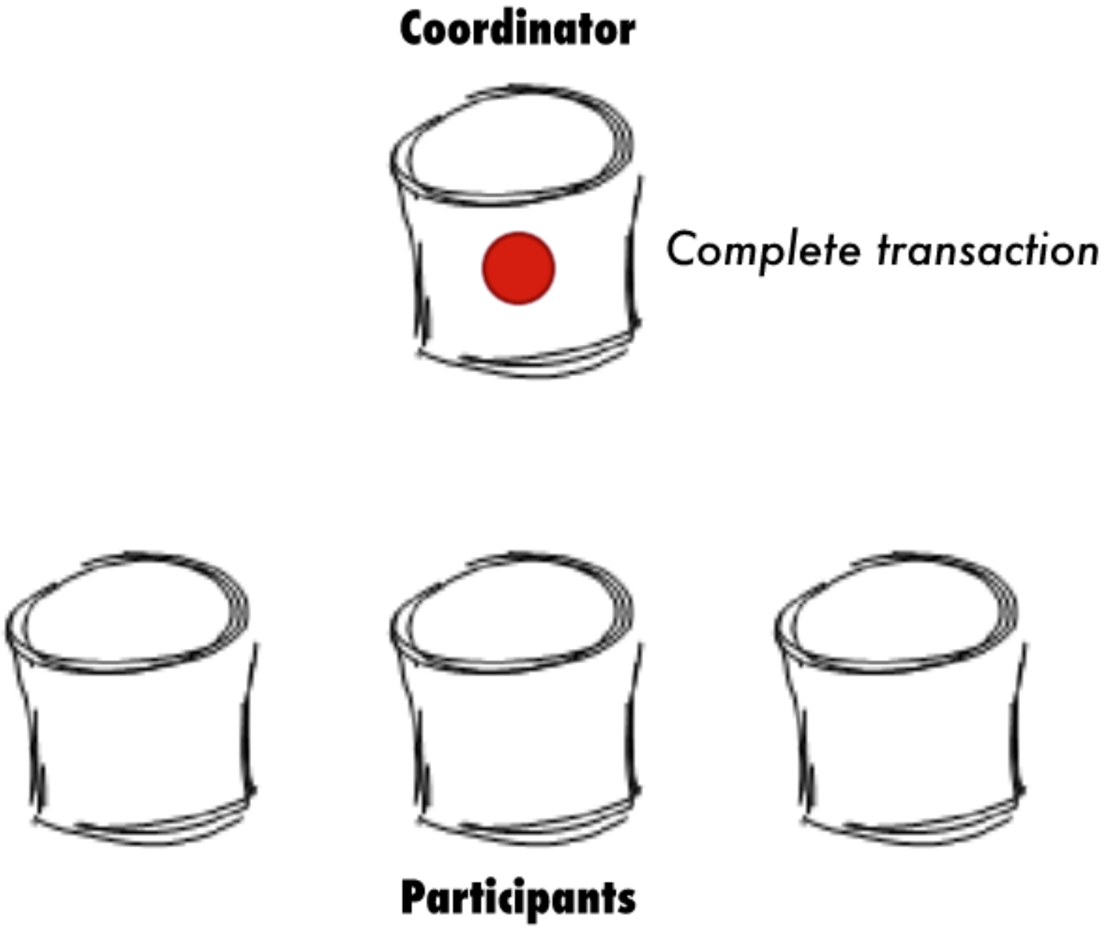
Distributed Transactions - 2PC



2) COMMIT PHASE (completion phase)

a) SUCCESS (agreement from all)

Distributed Transactions - 2PC



Coordinator

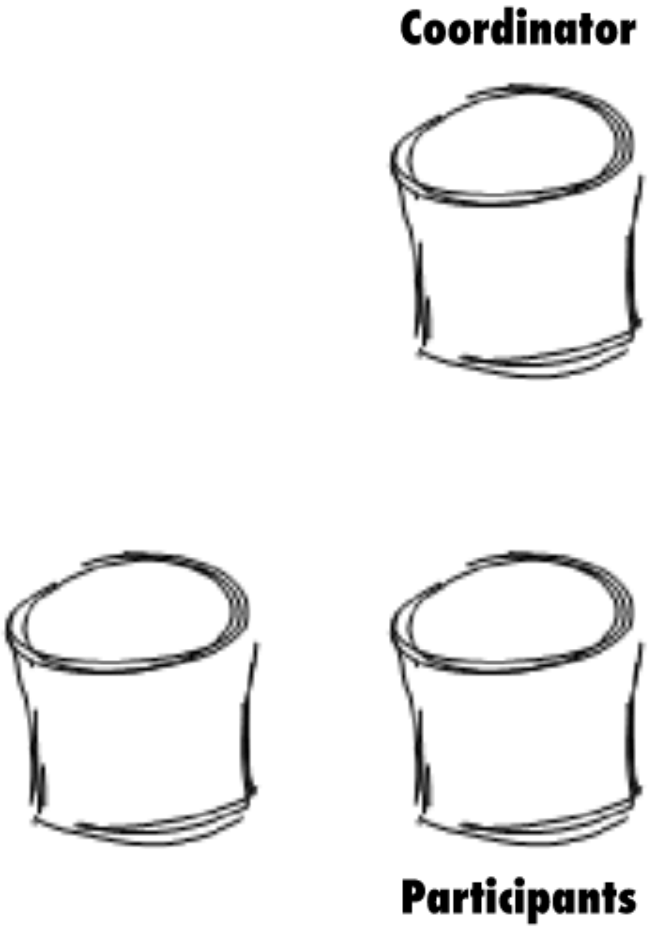
Complete transaction

Participants

2) COMMIT PHASE (completion phase)

a) SUCCESS (agreement from all)

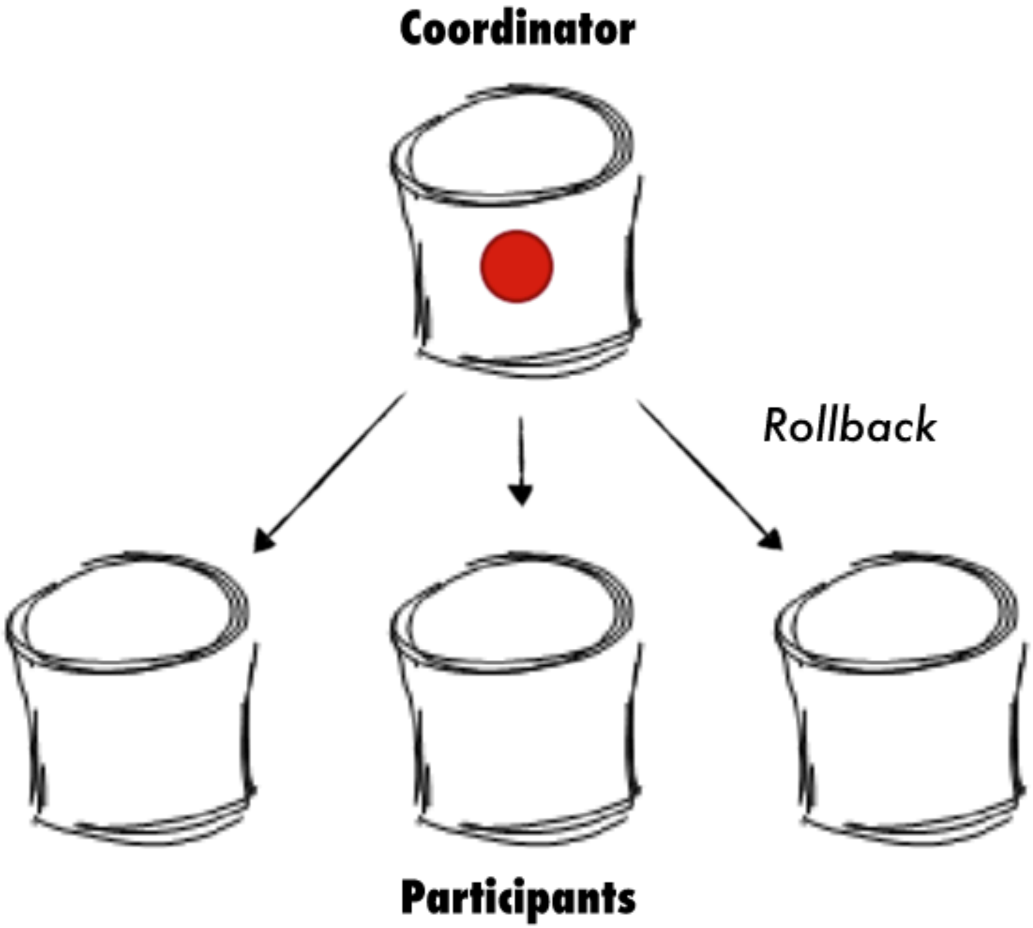
Distributed Transactions - 2PC



2) COMMIT PHASE (completion phase)

b) FAILURE (abort from any)

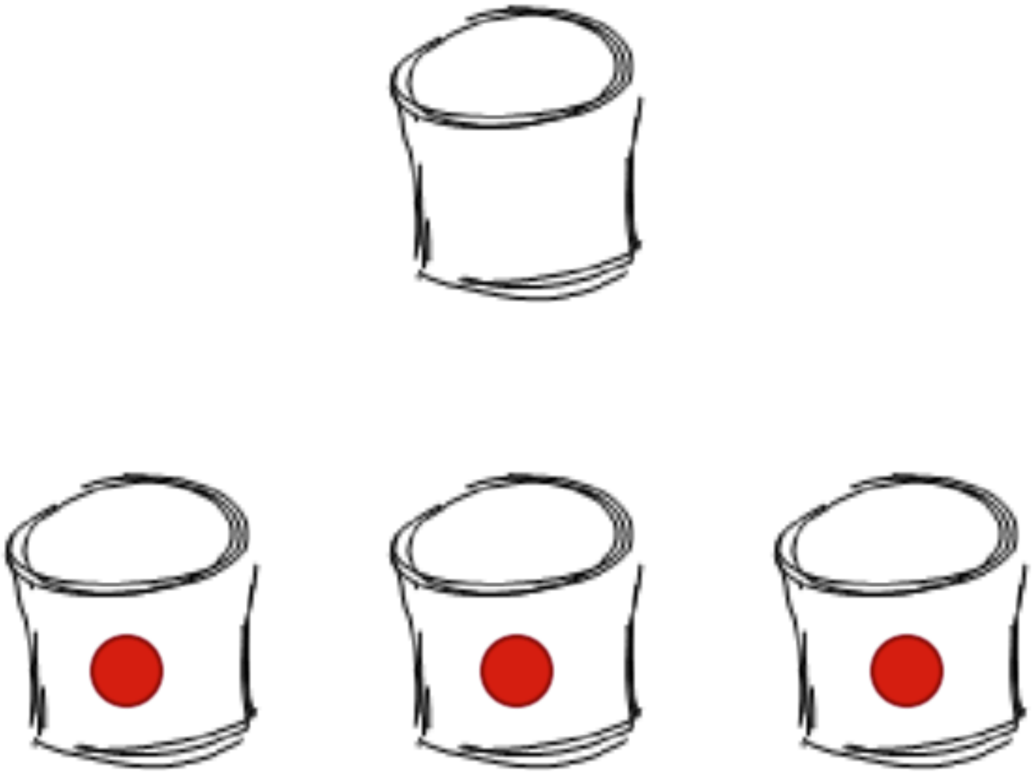
Distributed Transactions - 2PC



2) COMMIT PHASE (completion phase)

b) FAILURE (abort from any)

Distributed Transactions - 2PC



Coordinator

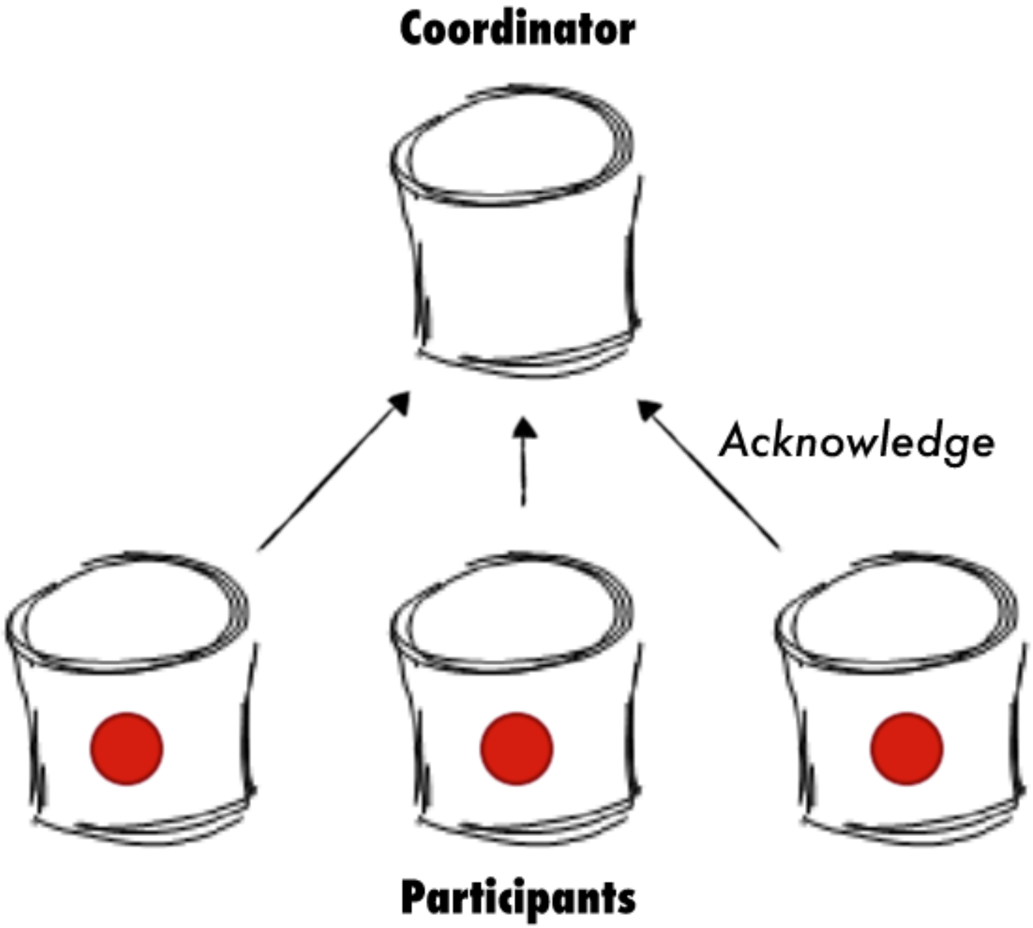
Participants

1) Undo operation
2) Release locks

2) COMMIT PHASE (completion phase)

b) FAILURE (abort from any)

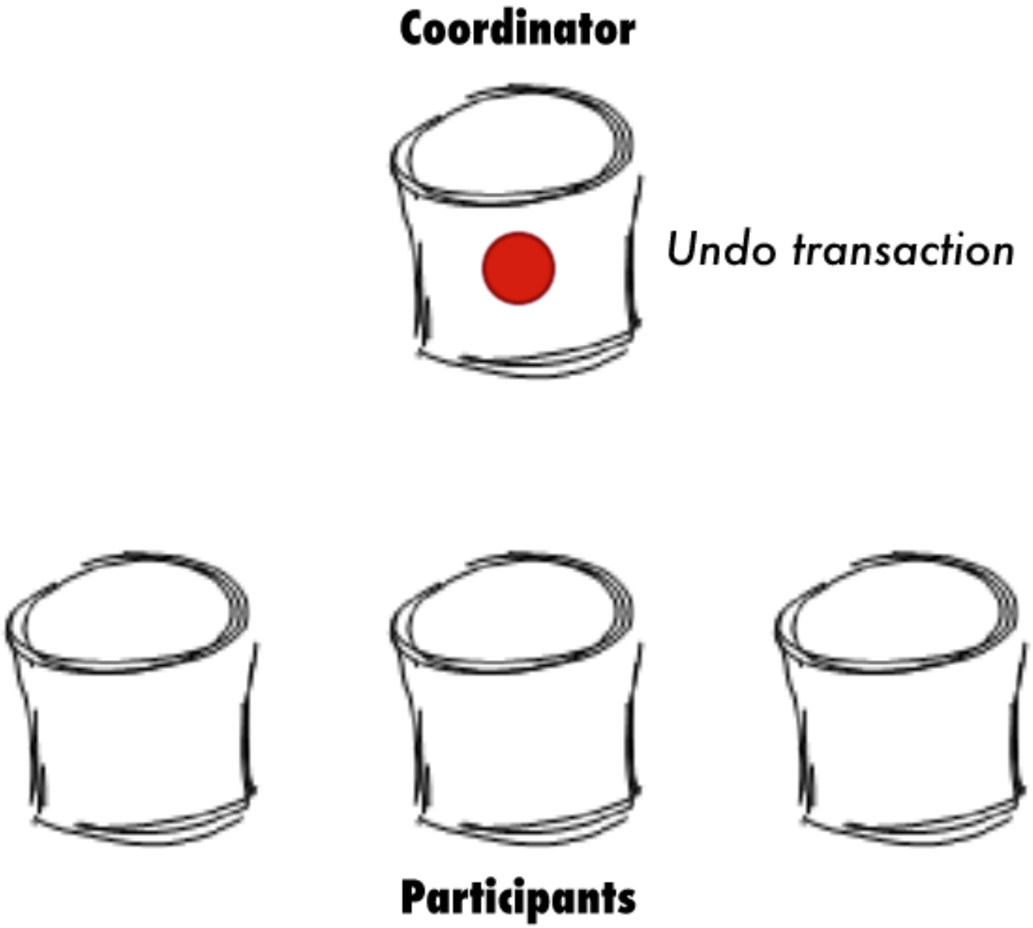
Distributed Transactions - 2PC



2) COMMIT PHASE (completion phase)

b) FAILURE (abort from any)

Distributed Transactions - 2PC



2) COMMIT PHASE (completion phase)

b) FAILURE (abort from any)

Problems with 2PC

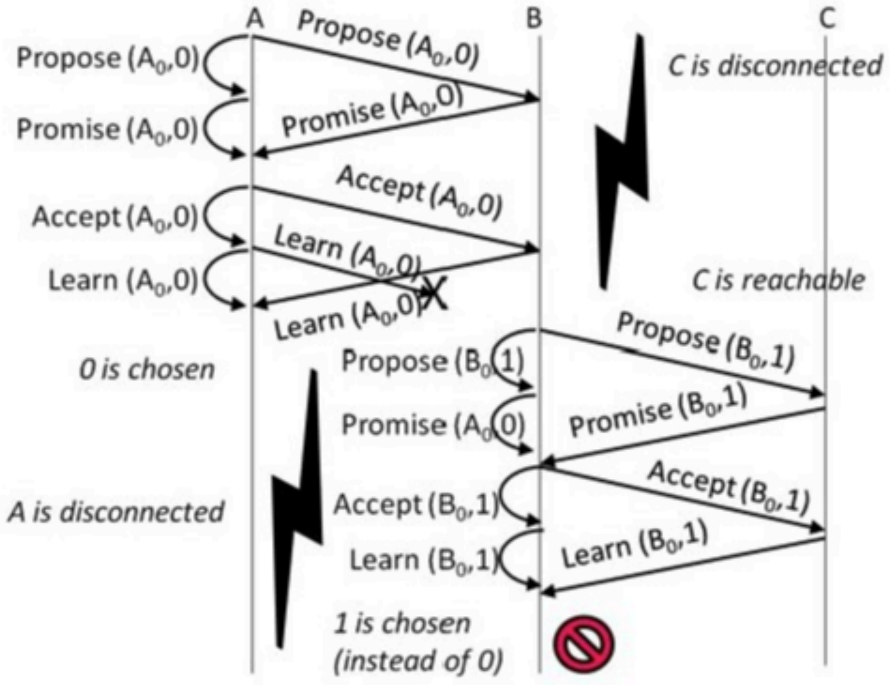
Blocking Protocol

Risk of indefinite cohort blocks if coordinator fails

Conservative behaviour biased to the abort case

Paxos Algorithm (Consensus)

- Family of Fault-tolerant, distributed implementations
- Spectrum of trade-offs:
  - Number of processors
  - Number of message delays
  - Activity level of participants
  - Number of messages sent
  - Types of failures

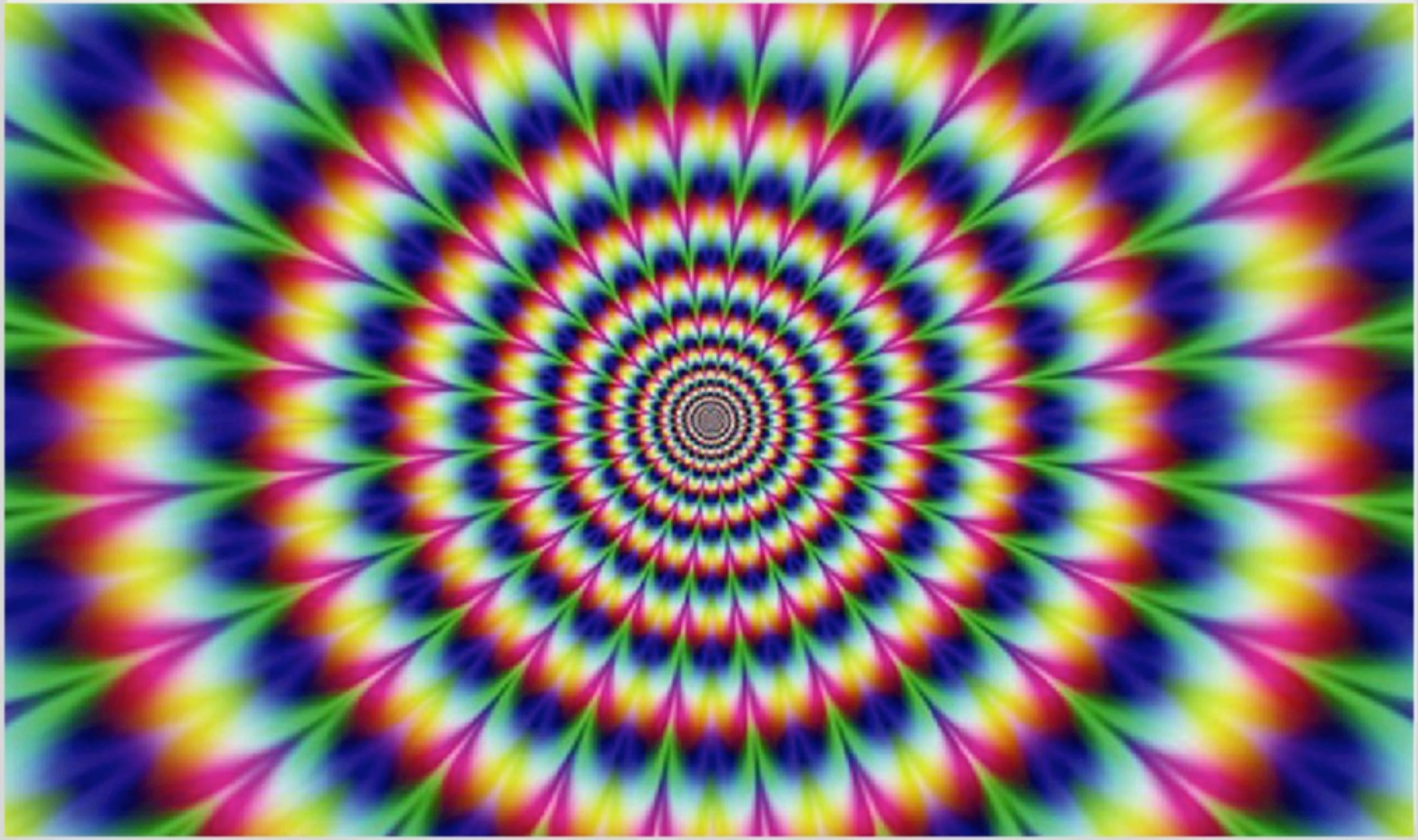


http://www.usenix.org/event/nsdi09/tech/full_papers/yabandeh/yabandeh_html/

http://en.wikipedia.org/wiki/Paxos_algorithm

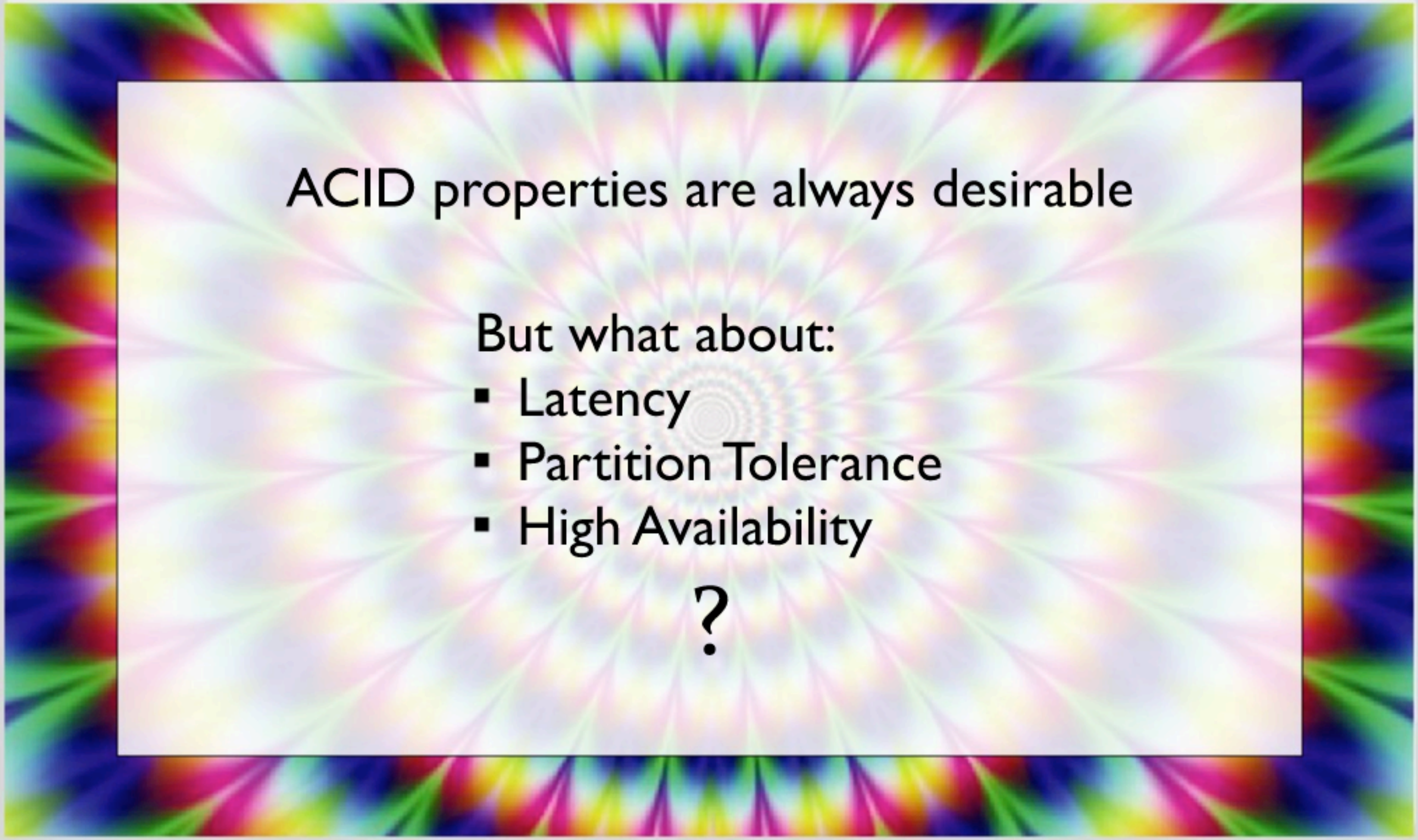
[PSE image alert]

ACID & Distributed Systems



http://images.tribe.net/tribe/upload/photo/deb/074/deb074db-81fc-4b8a-bfbd-b18b922885cb

ACID & Distributed Systems



ACID properties are always desirable

But what about:

- Latency
- Partition Tolerance
- High Availability

?

http://images.tribe.net/tribe/upload/photo/deb/074/deb074db-81fc-4b8a-bfbd-b18b922885cb

CAP Theorem (Brewer’s conjecture)

2000 Prof. Eric Brewer, PoDC Conference Keynote

2002 Seth Gilbert and Nancy Lynch, ACM SIGACT News 33(2)

> “*Of three properties of shared-data systems - data **C**onsistency, system **A**vailability and tolerance to network **P**artitions - only two can be achieved at any given moment in time.*”

http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf

CAP Theorem (Brewer's conjecture)

2000 Prof. Eric Brewer, PoDC Conference Keynote

2002 Seth Gilbert and Nancy Lynch, ACM SIGACT News 33(2)

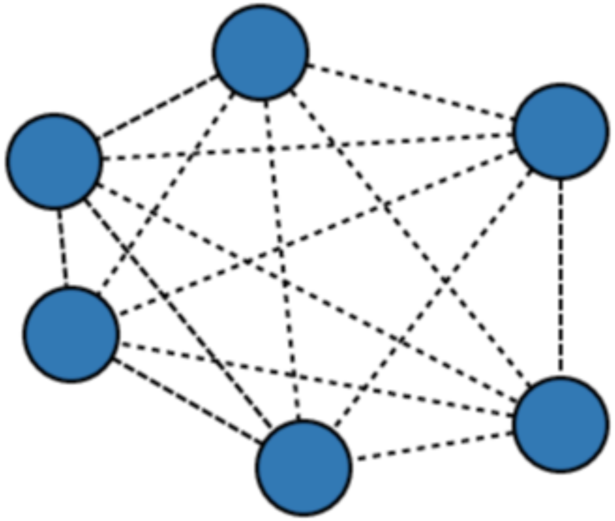
> “*Of three properties of shared-data systems - data **C**onsistency, system **A**vailability and tolerance to network **P**artitions - only two can be achieved at any given moment in time.*”

http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf

Partition Tolerance - Availability

"The network will be allowed to lose arbitrarily many messages sent from one node to another" [...]

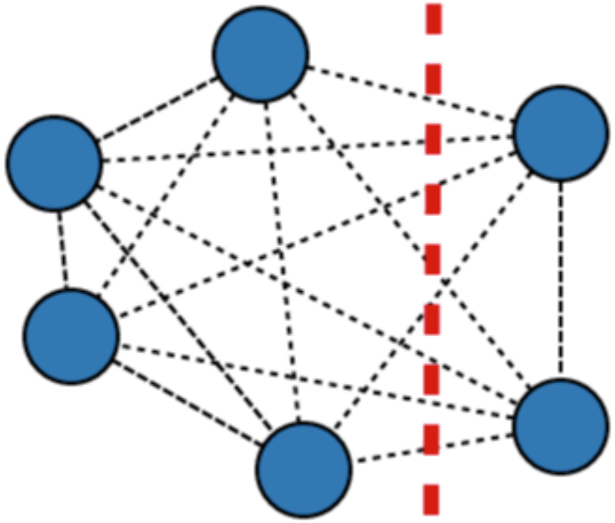
"For a distributed system to be continuously available, every request received by a non-failing node in the system must result in a response" - *Gilbert and Lynch, SIGACT 2002*



http://codahale.com/you-cant-sacrifice-partition-tolerance http://pl.atyp.us/wordpress/?p=2521

Partition Tolerance - Availability

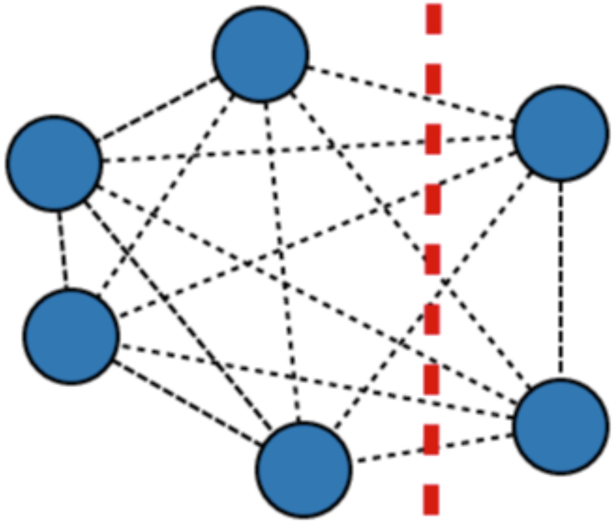
"The network will be allowed to lose arbitrarily many messages sent from one node to another" [...]
"For a distributed system to be continuously available, every request received by a non-failing node in the system must result in a response" - *Gilbert and Lynch, SIGACT 2002*



http://codahale.com/you-cant-sacrifice-partition-tolerance http://pl.atyp.us/wordpress/?p=2521

Partition Tolerance - Availability

> “The network will be allowed to lose arbitrarily many messages sent from one node to another” [...]
>
> “For a distributed system to be continuously available, every request received by a non-failing node in the system must result in a response” - *Gilbert and Lynch, SIGACT 2002*



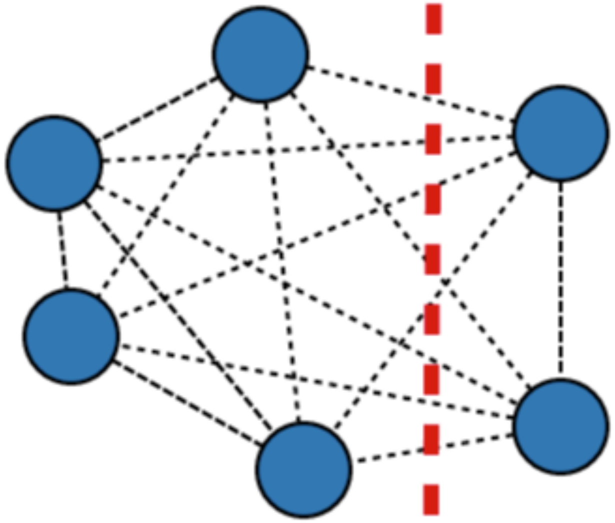
- **CP:** requests can complete at nodes that have quorum
- **AP:** requests can complete at any live node, possibly violating strong consistency

http://codahale.com/you-cant-sacrifice-partition-tolerance http://pl.atyp.us/wordpress/?p=2521

Partition Tolerance - Availability

"The network will be allowed to lose arbitrarily many messages sent from one node to another" [...]

"For a distributed system to be continuously available, every request received by a non-failing node in the system must result in a response" - *Gilbert and Lynch, SIGACT 2002*



HIGH LATENCY

≈

NETWORK PARTITION

http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.html

http://codahale.com/you-cant-sacrifice-partition-tolerance http://pl.atyp.us/wordpress/?p=2521

Consistency: Client-side view

A service that is *consistent* operates fully or not at all.

- Strong consistency (as in ACID)
- Weak consistency (no guarantee) - Inconsistency window

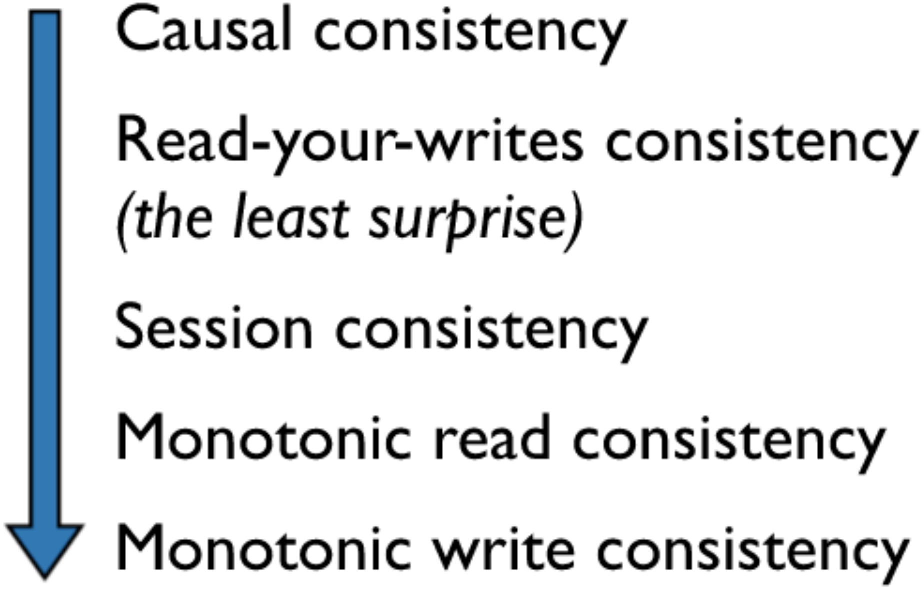
(*) Temporary inconsistencies (e.g. in data constraints or replica versions) are accepted, but they're resolved at the earliest opportunity

http://www.allthingsdistributed.com/2008/12/eventually_consistent.html

Consistency: Client-side view

A service that is *consistent* operates fully or not at all.

- Strong consistency (as in ACID)
- Weak consistency (no guarantee) - Inconsistency window
  - Eventual* consistency (e.g. DNS)



Causal consistency

Read-your-writes consistency
(the least surprise)

Session consistency

Monotonic read consistency

Monotonic write consistency

(*) Temporary inconsistencies (e.g. in data constraints or replica versions) are accepted, but they're resolved at the earliest opportunity

http://www.allthingsdistributed.com/2008/12/eventually_consistent.html

Consistency: Server-side (Quorum)

N = number of nodes with a replica of the data
W = number of replicas that must acknowledge the update(*)
R = minimum number of replicas that must participate in a successful read operation

(*) but the data will be written to N nodes no matter what

$W + R > N$ ⟶ Strong consistency (usually N=3, W=R=2)

$W = N$, $R = 1$ ⟶ Optimised for reads
$W = 1$, $R = N$ ⟶ Optimised for writes

(durability not guaranteed in presence of failures)

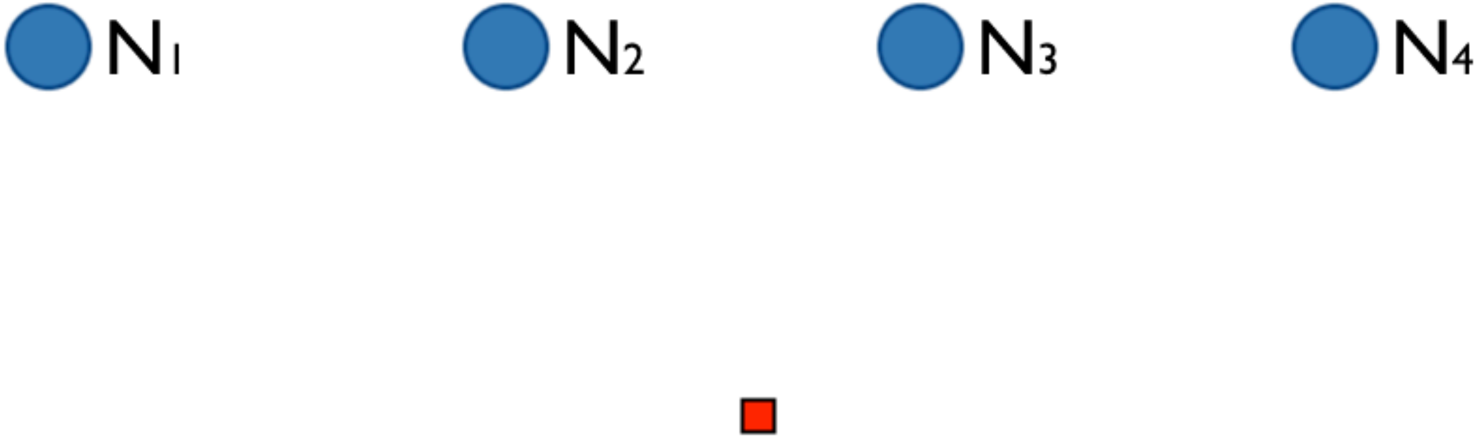
$W + R <= N$ ⟶ Weak consistency

Amazon Dynamo Paper

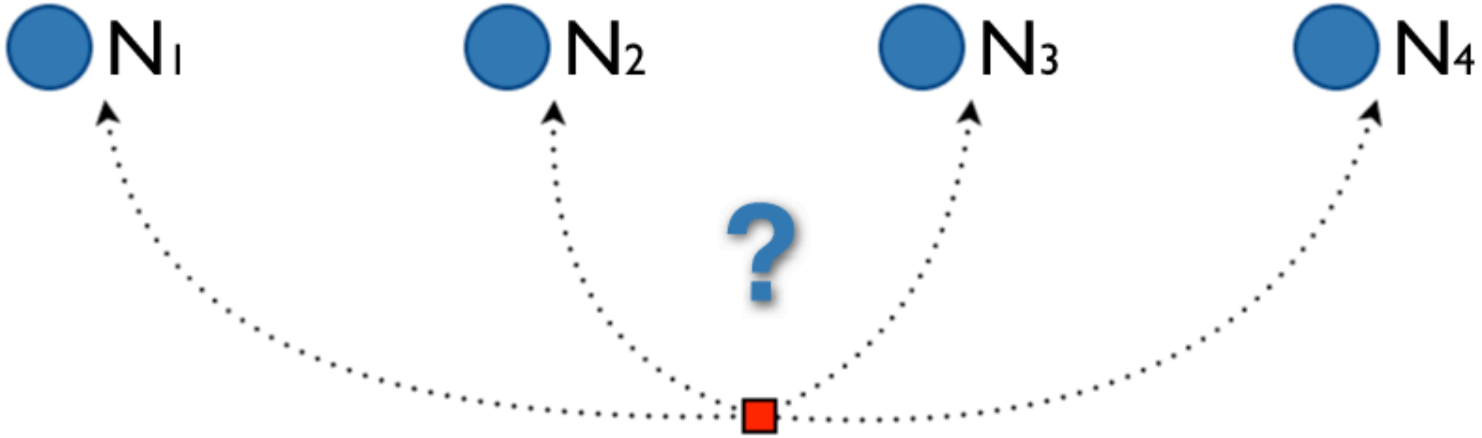
- Consistent Hashing
- Vector Clocks
- Gossip Protocol
- Hinted Handoffs
- Read Repair

http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf

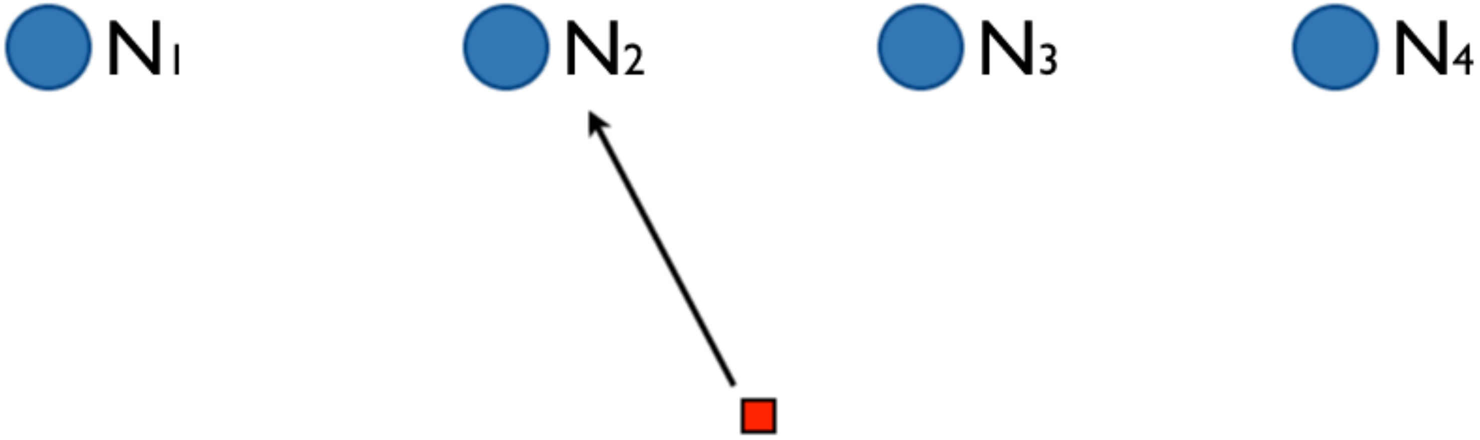
Modulo-based Hashing



Modulo-based Hashing

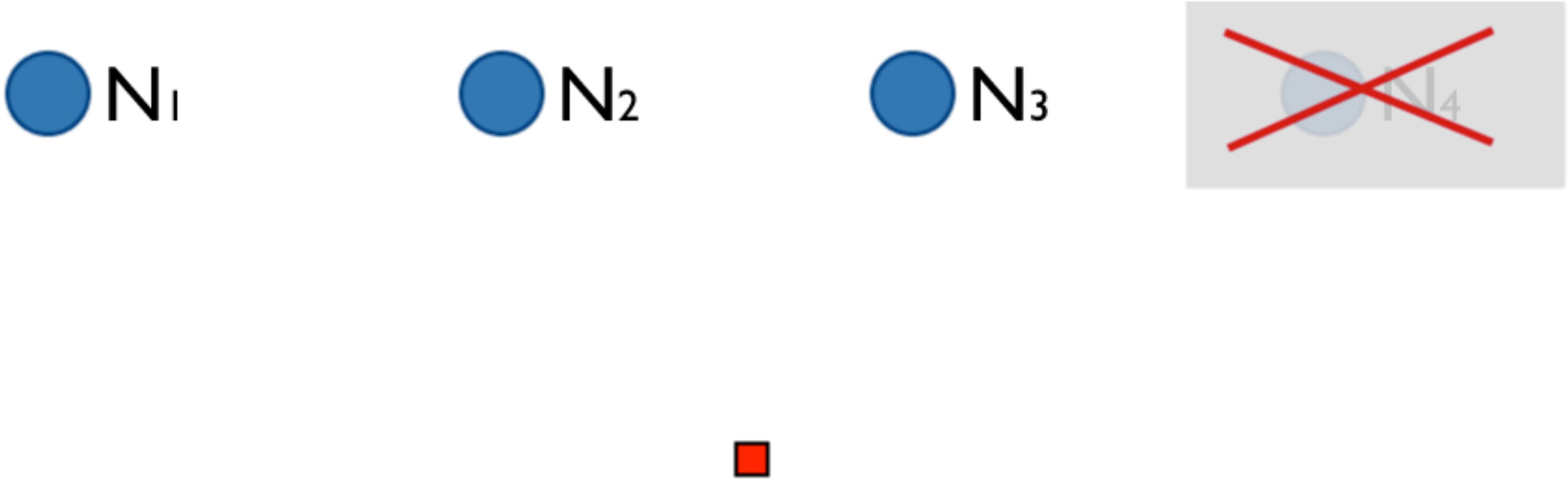


Modulo-based Hashing



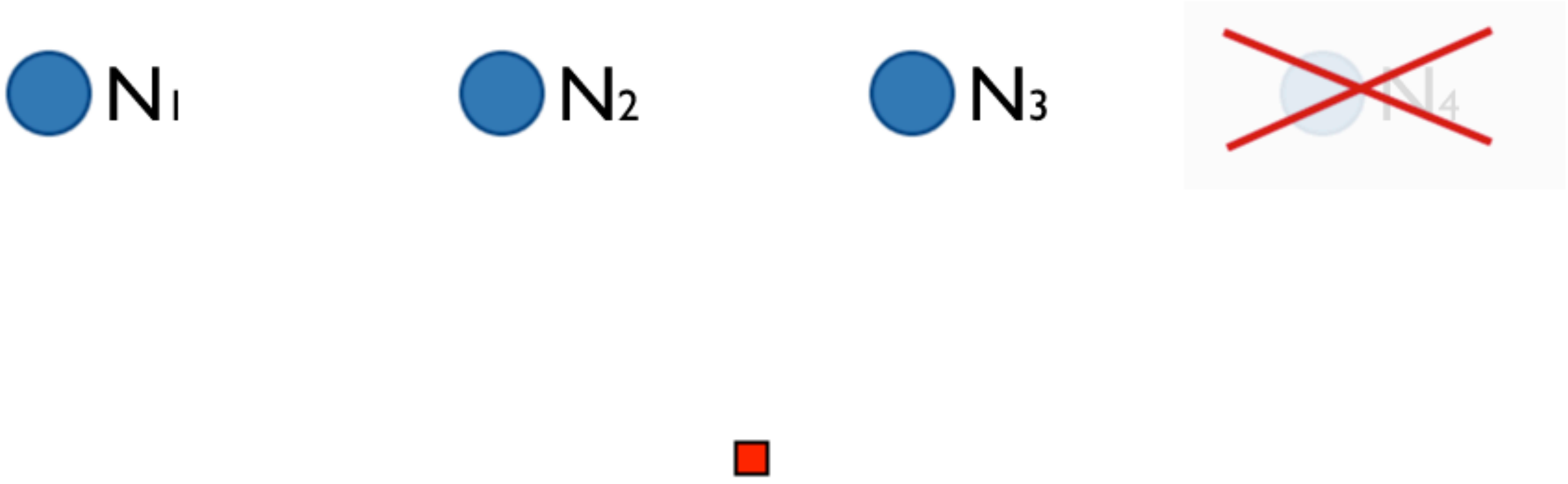
```
partition = key % n_servers
```

Modulo-based Hashing



```
partition = key % (n_servers - 1)
```

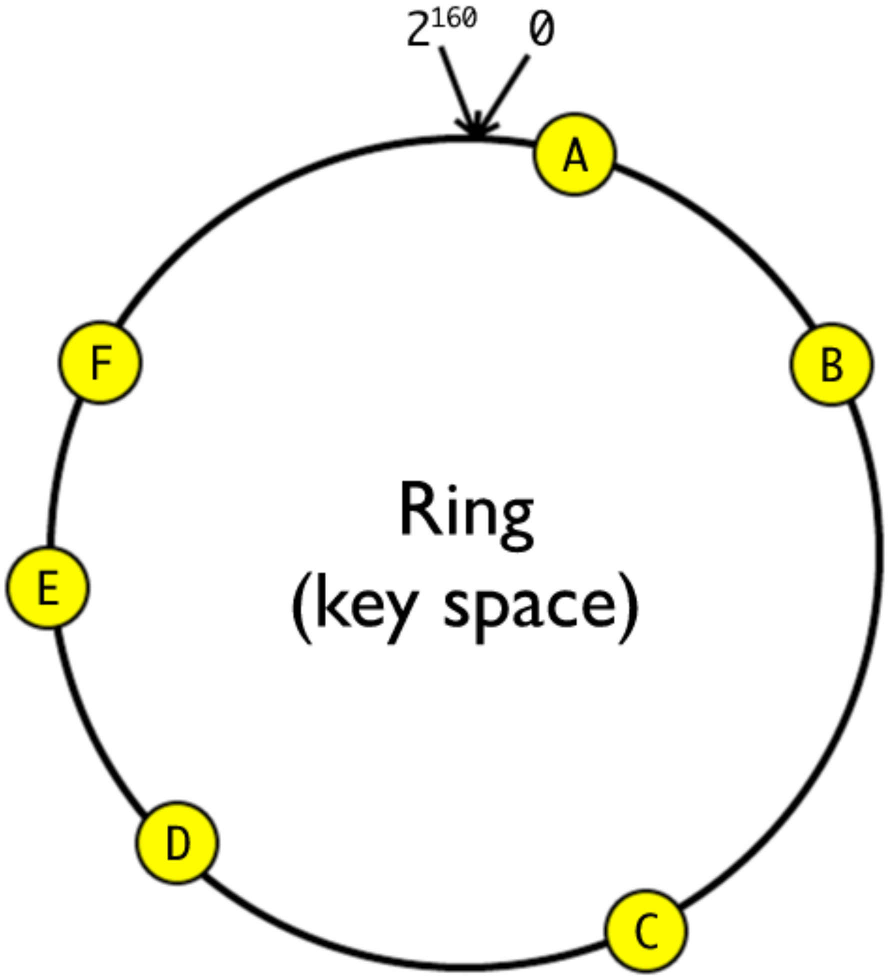
Modulo-based Hashing



```
partition = key % (n_servers - 1)
```

⚠ Recalculate the hashes for all the entries if *n_servers* changes (i.e. full data redistribution when adding/removing a node)

Consistent Hashing



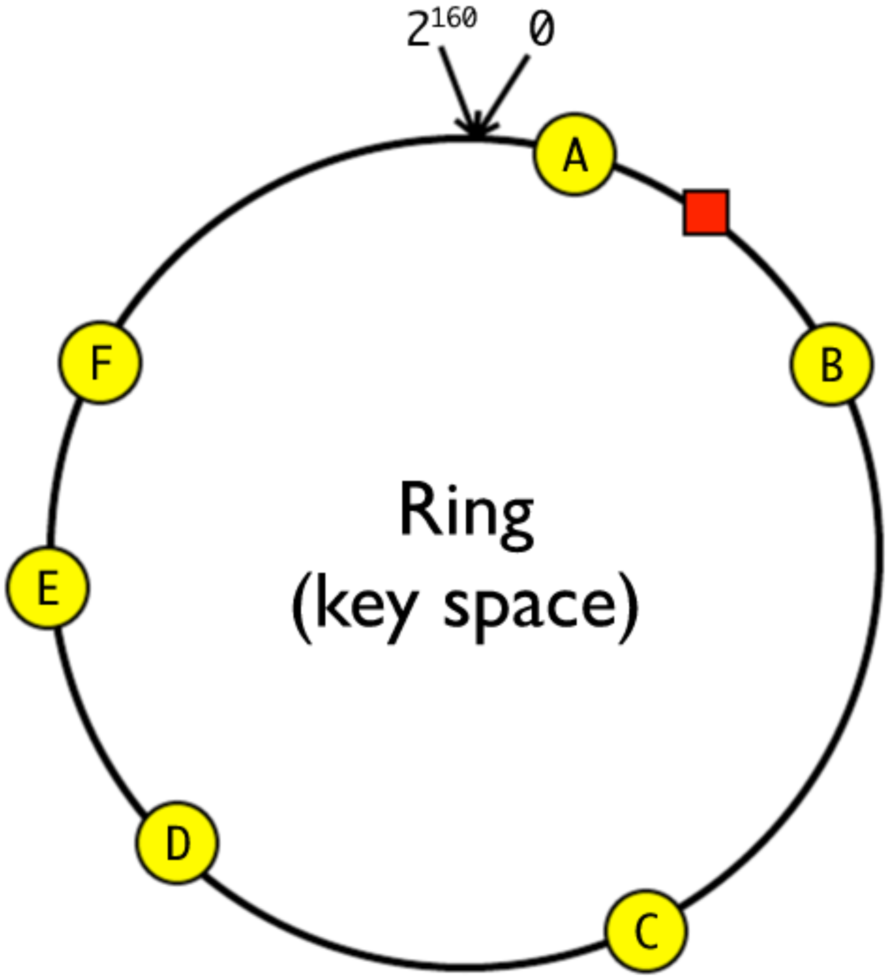
Same hash function for data and nodes

`idx = hash(key)`

Coordinator: next available clockwise node

http://en.wikipedia.org/wiki/Consistent_hashing

Consistent Hashing



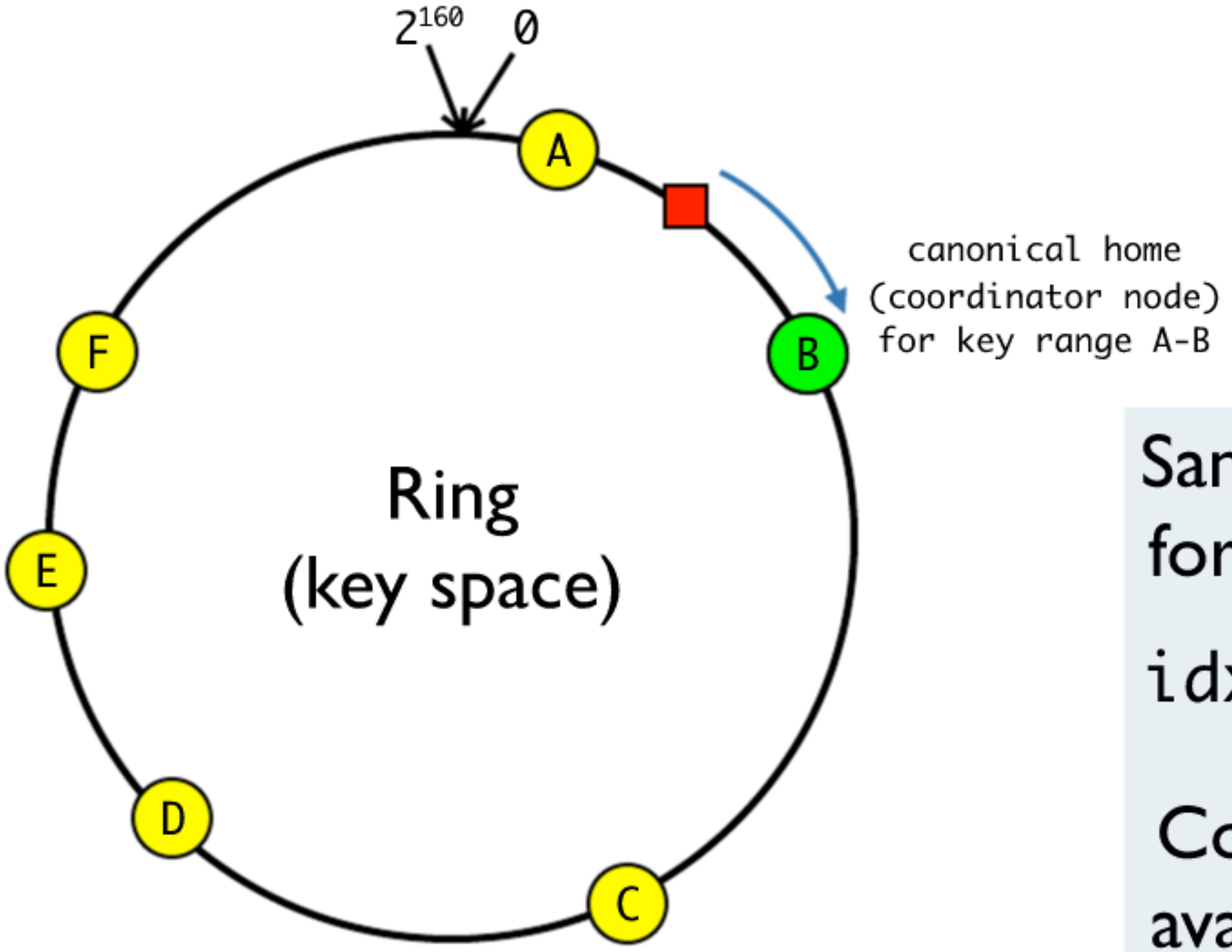
Same hash function for data and nodes

```
idx = hash(key)
```

Coordinator: next available clockwise node

http://en.wikipedia.org/wiki/Consistent_hashing

Consistent Hashing



canonical home
(coordinator node)
for key range A-B

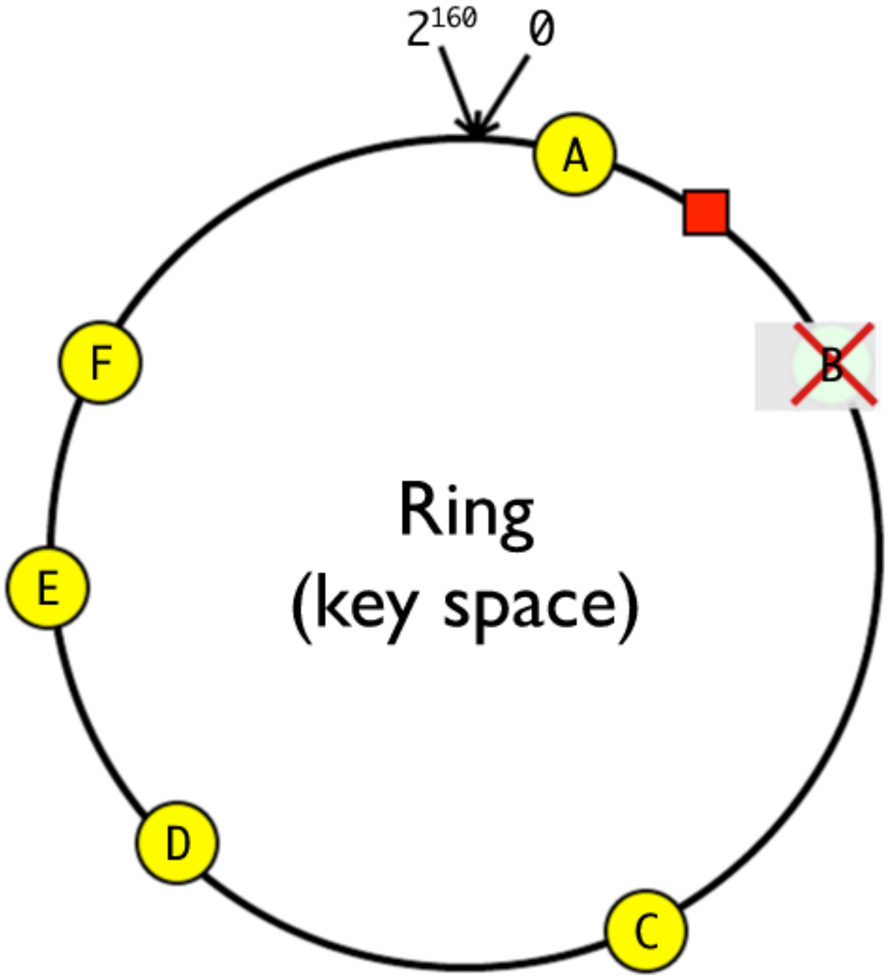
Same hash function for data and nodes

`idx = hash(key)`

Coordinator: next available clockwise node

http://en.wikipedia.org/wiki/Consistent_hashing

Consistent Hashing



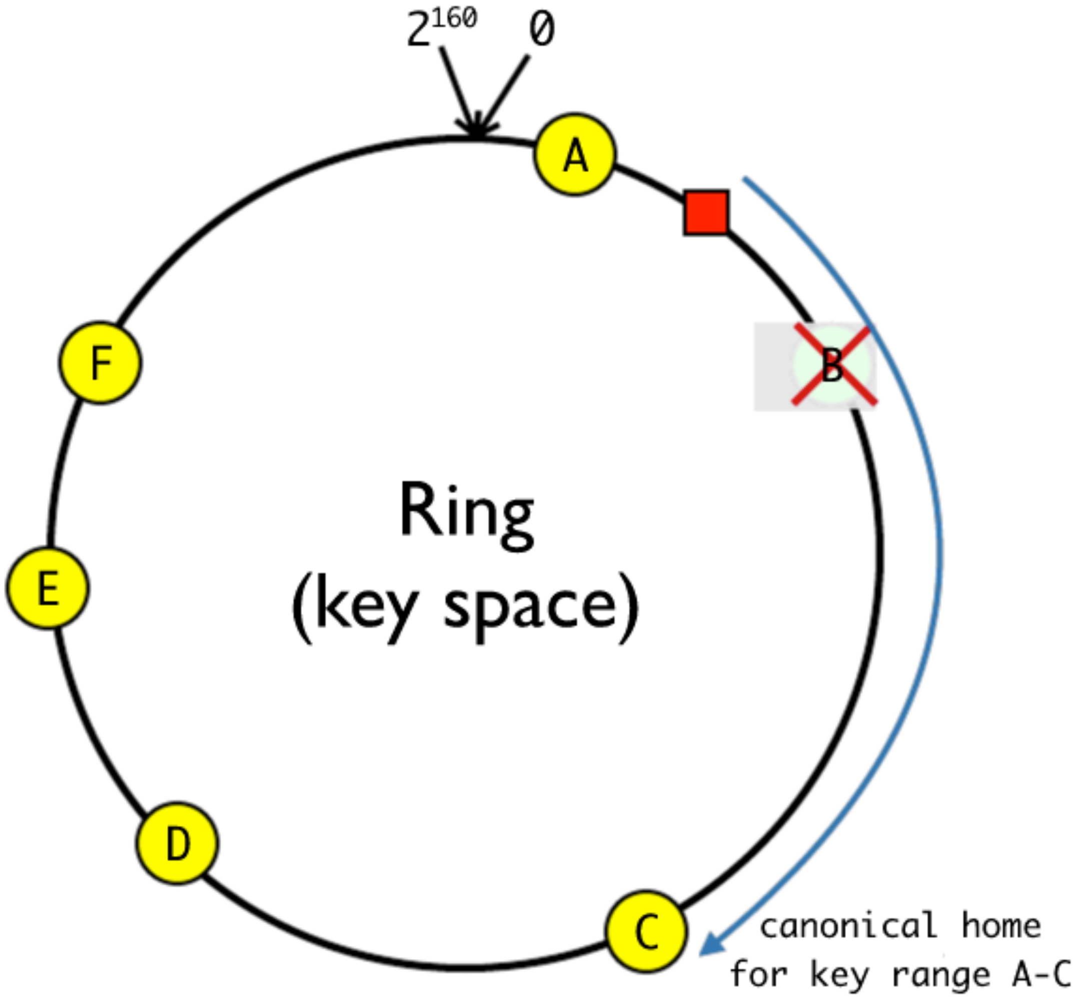
Same hash function for data and nodes

`idx = hash(key)`

Coordinator: next available clockwise node

http://en.wikipedia.org/wiki/Consistent_hashing

Consistent Hashing



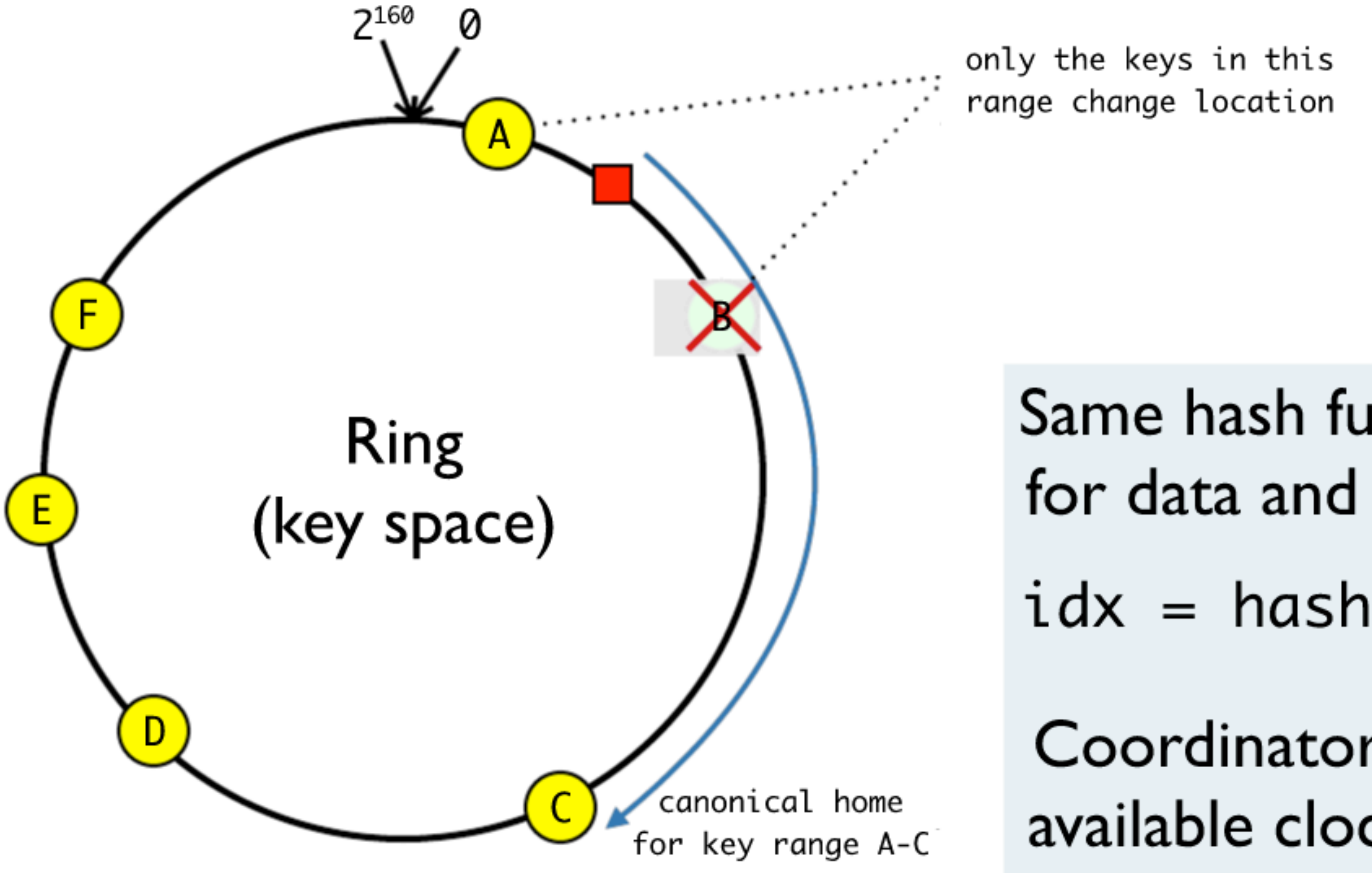
Same hash function for data and nodes

`idx = hash(key)`

Coordinator: next available clockwise node

http://en.wikipedia.org/wiki/Consistent_hashing

Consistent Hashing


2^{160}
0
only the keys in this range change location
A
B
C
D
E
F
Ring (key space)
canonical home for key range A-C

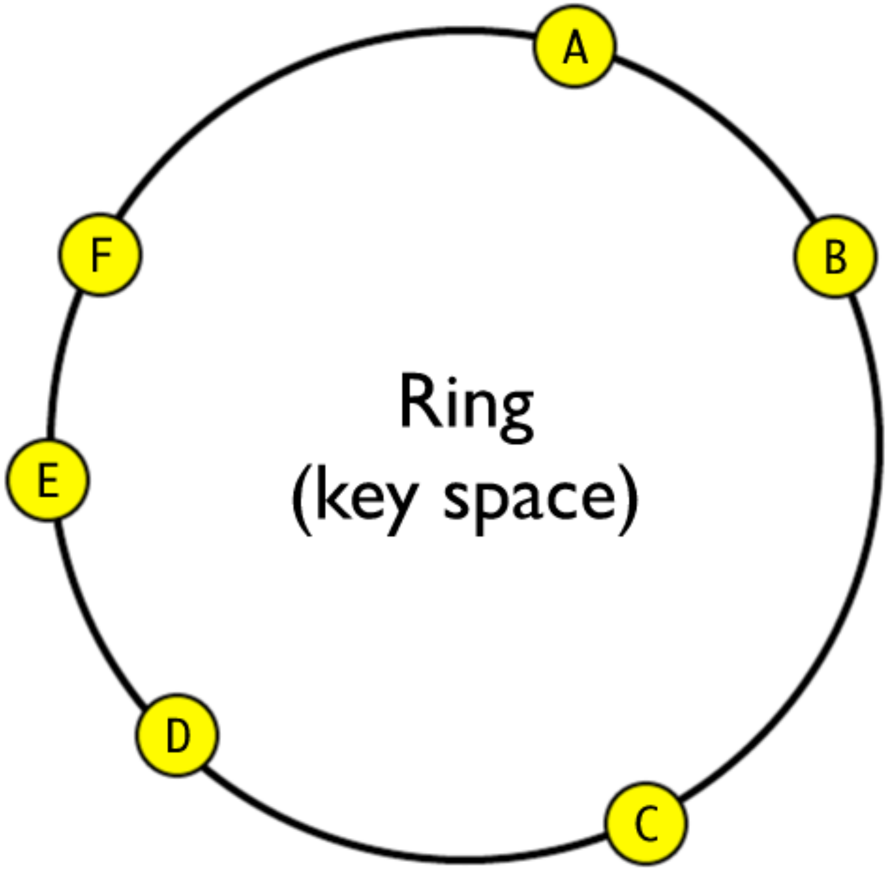
Same hash function for data and nodes

idx = hash(key)

Coordinator: next available clockwise node

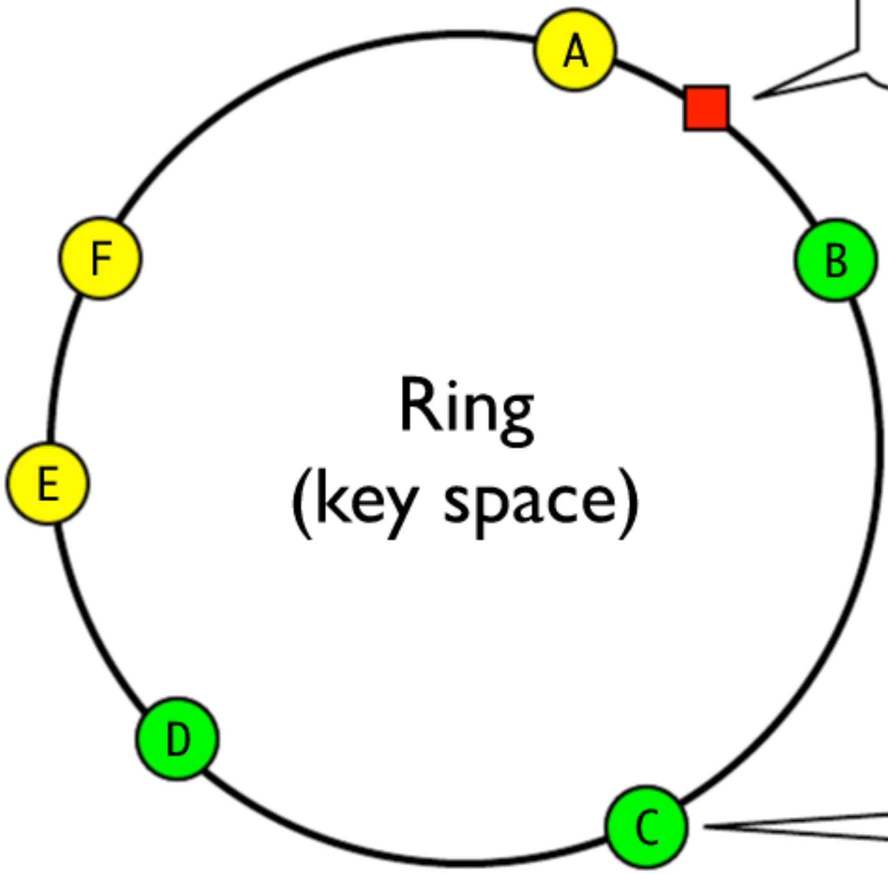
http://en.wikipedia.org/wiki/Consistent_hashing

Consistent Hashing - Replication



http://horicky.blogspot.com/2009/11/nosql-patterns.html

Consistent Hashing - Replication



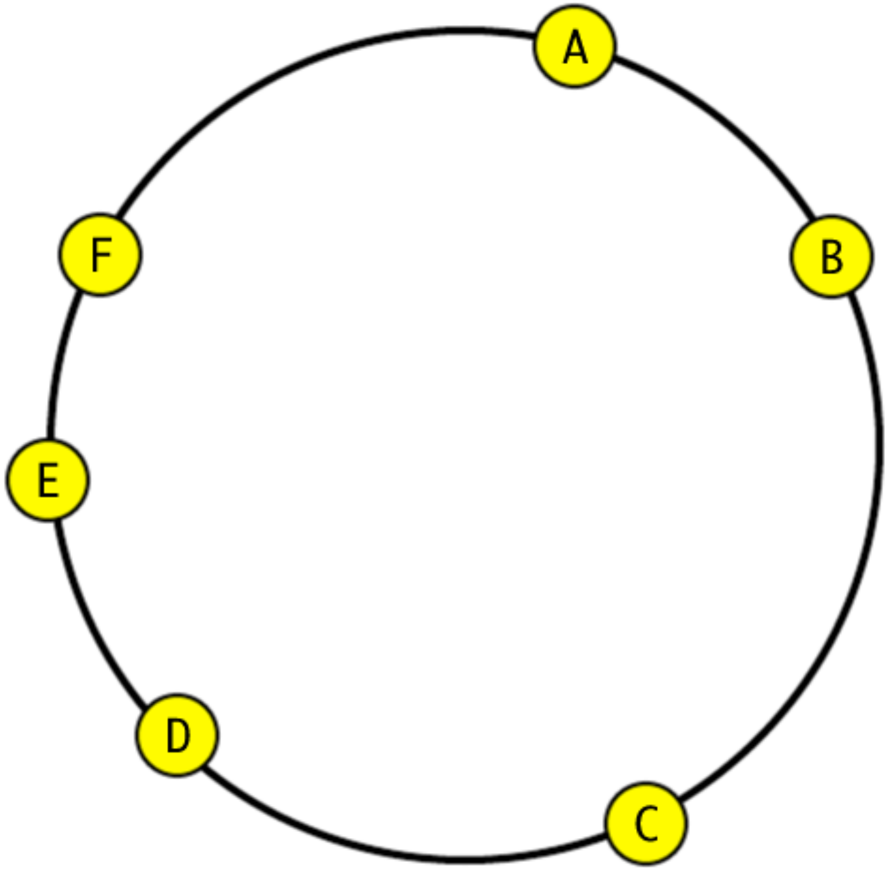
Key$_{AB}$ hosted in B, C, D

Data replicated in the N-1 clockwise successor nodes

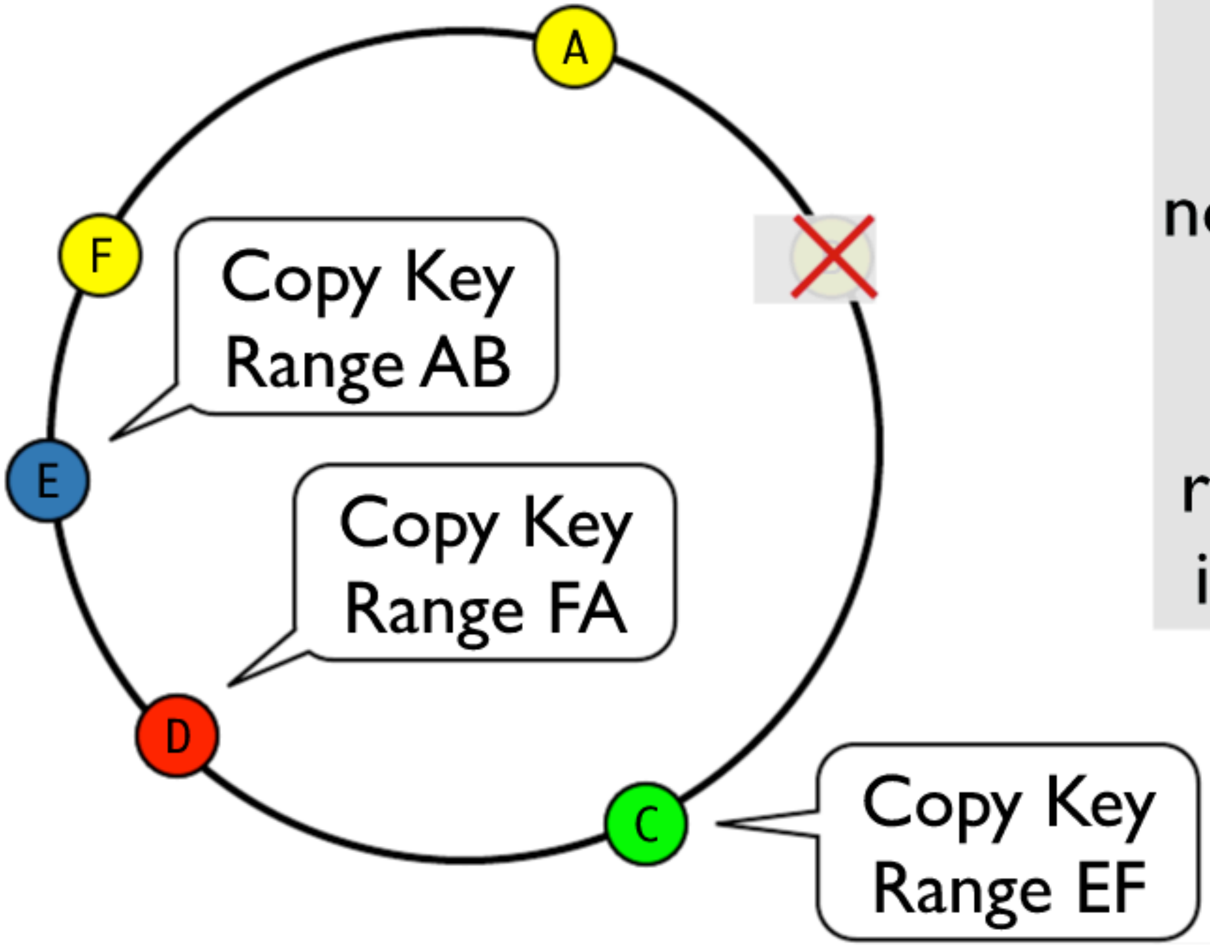
Node hosting Key$_{FA}$, Key$_{AB}$, Key$_{BC}$

http://horicky.blogspot.com/2009/11/nosql-patterns.html

Consistent Hashing - Node Changes

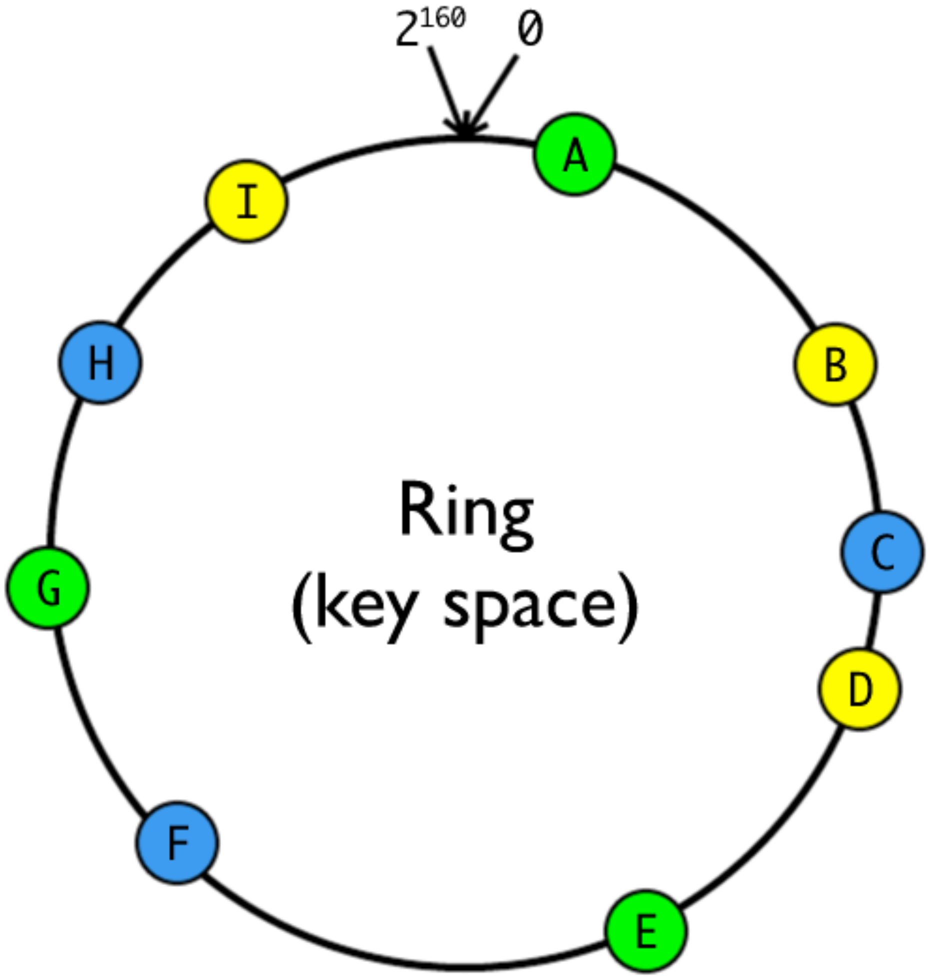


Consistent Hashing - Node Changes



Key membership and replicas are updated when a node joins or leaves the network. The number of replicas for all data is kept consistent.

Consistent Hashing - Load Distribution



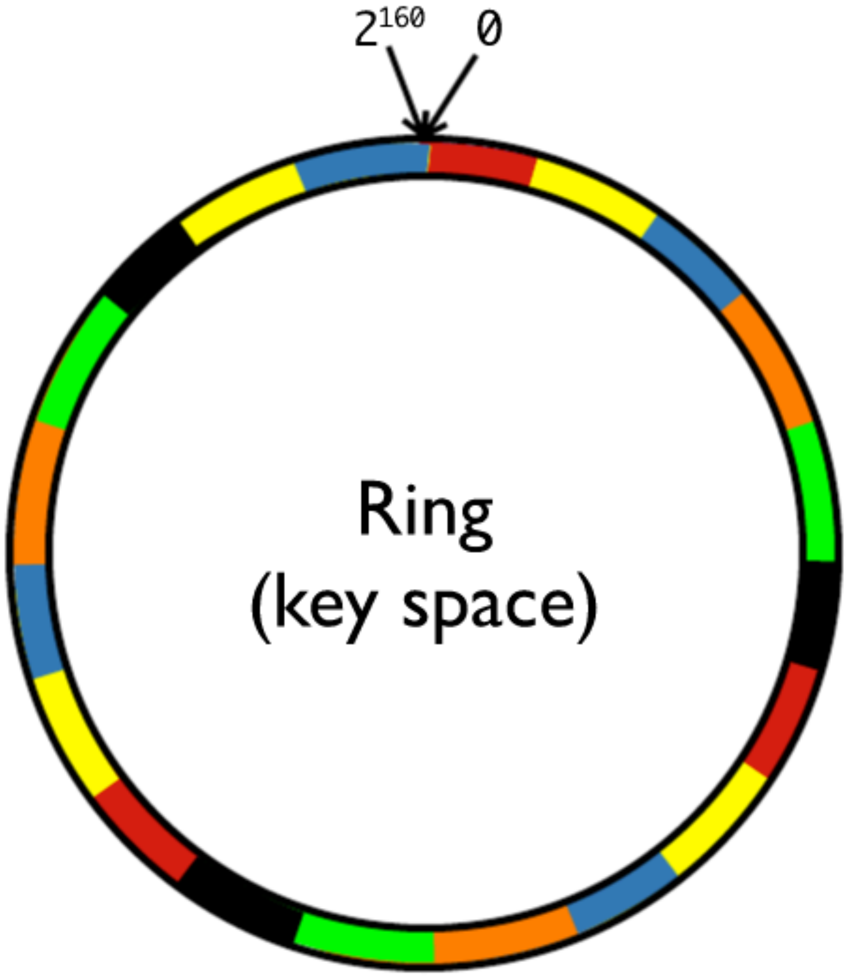
Different Strategies

Virtual Nodes

Random tokens per each physical node, partition by token value

🟢 Node 1: tokens A, E, G
🔵 Node 2: tokens C, F, H
🟡 Node 3: tokens B, D, I

Consistent Hashing - Load Distribution



Different Strategies

Virtual Nodes

Q equal-sized partitions, ***S*** nodes, ***Q/S*** tokens per node (with **Q** >> **S**)

🟥 Node 1

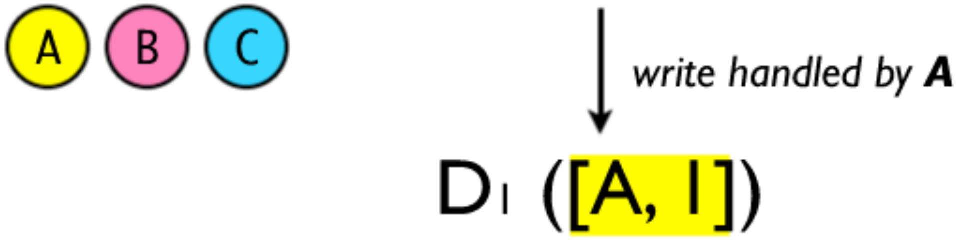
🟩 Node 2

🟦 Node 3

🟧 Node 4

...

Vector Clocks & Conflict Detection



*write handled by **A***

D_1 ([A, 1])

Causality-based partial order over events that happen in the system.

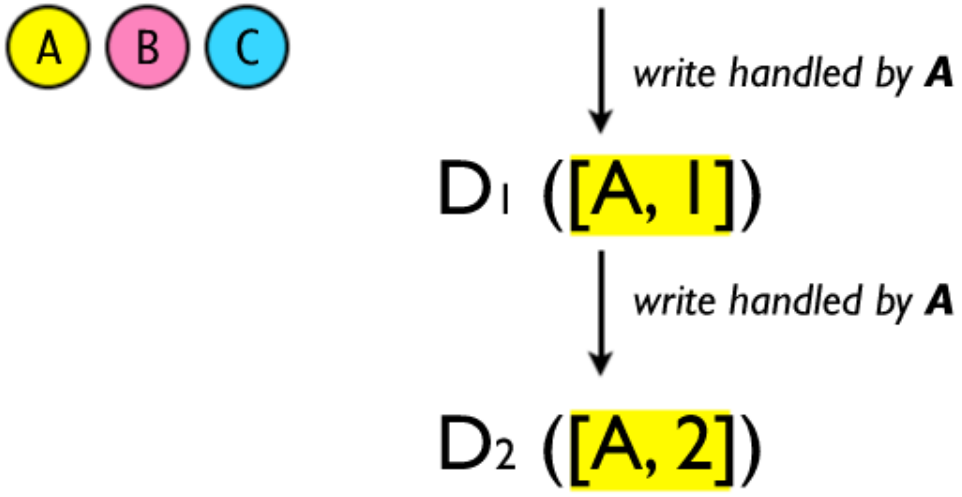
Document version history: a counter for each node that updated the document.

If all update counters in V_1 are smaller or equal to all update counters in V_2, then V_1 precedes V_2.

http://en.wikipedia.org/wiki/Vector_clock

http://pl.atyp.us/wordpress/?p=2601

Vector Clocks & Conflict Detection



write handled by ***A***

D_1 ([A, 1])

write handled by ***A***

D_2 ([A, 2])

Causality-based partial order over events that happen in the system.

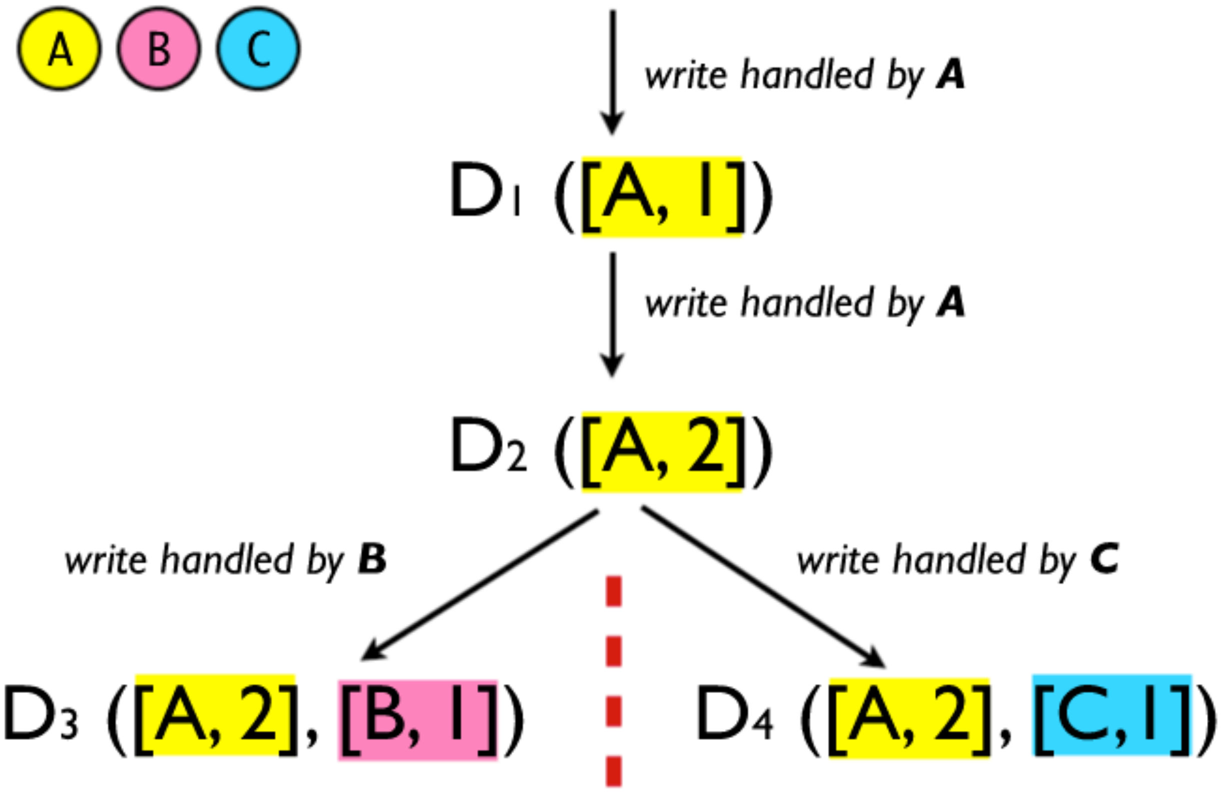
Document version history: a counter for each node that updated the document.

If all update counters in V_1 are smaller or equal to all update counters in V_2, then V_1 precedes V_2.

http://en.wikipedia.org/wiki/Vector_clock

http://pl.atyp.us/wordpress/?p=2601

Vector Clocks & Conflict Detection



A B C

write handled by ***A***

D_1 ([A, 1])

write handled by ***A***

D_2 ([A, 2])

write handled by ***B***

write handled by ***C***

D_3 ([A, 2], [B, 1])

D_4 ([A, 2], [C, 1])

Causality-based partial order over events that happen in the system.

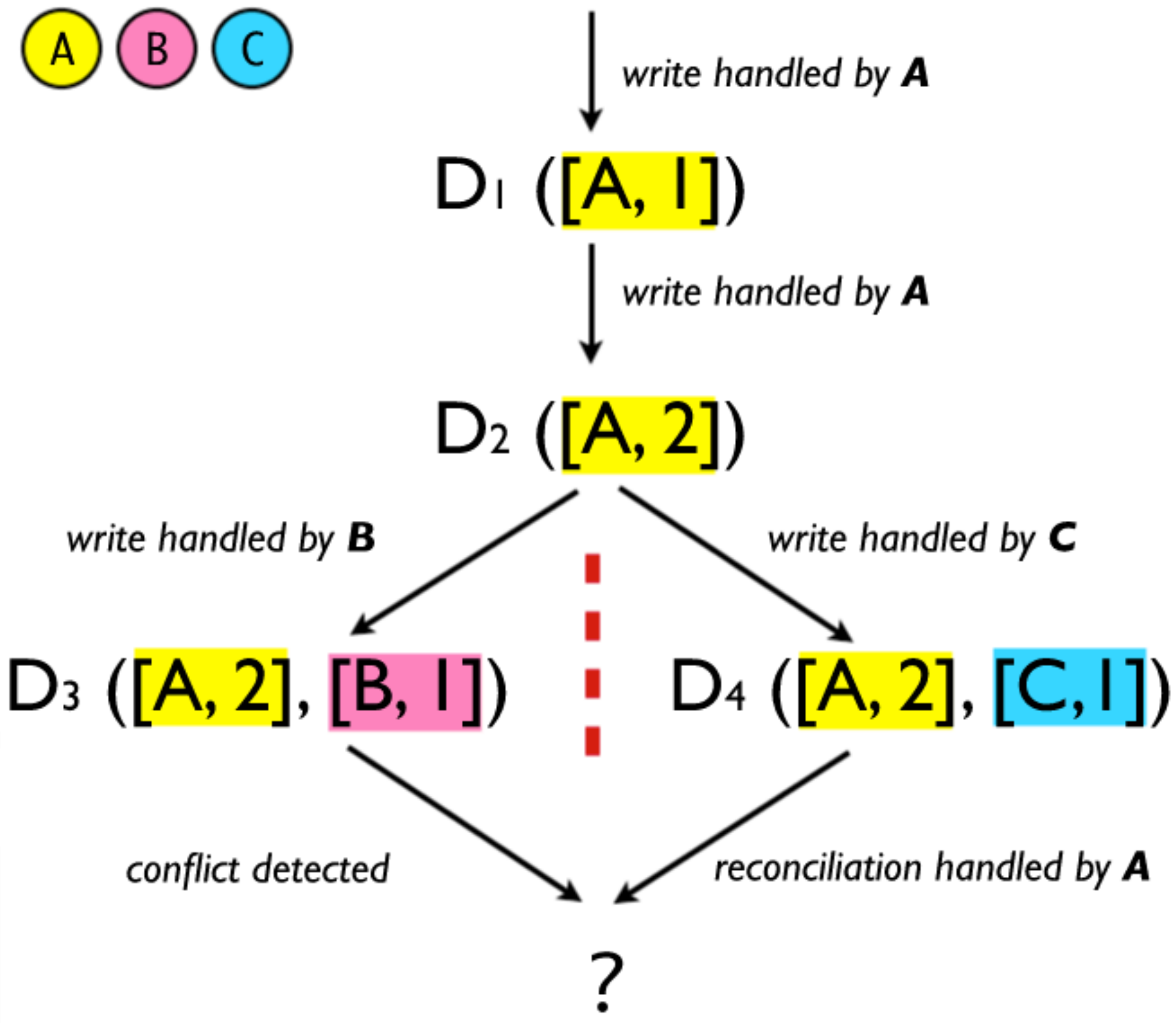
Document version history: a counter for each node that updated the document.

If all update counters in V_1 are smaller or equal to all update counters in V_2, then V_1 precedes V_2.

http://en.wikipedia.org/wiki/Vector_clock

http://pl.atyp.us/wordpress/?p=2601

Vector Clocks & Conflict Detection



Causality-based partial order over events that happen in the system.

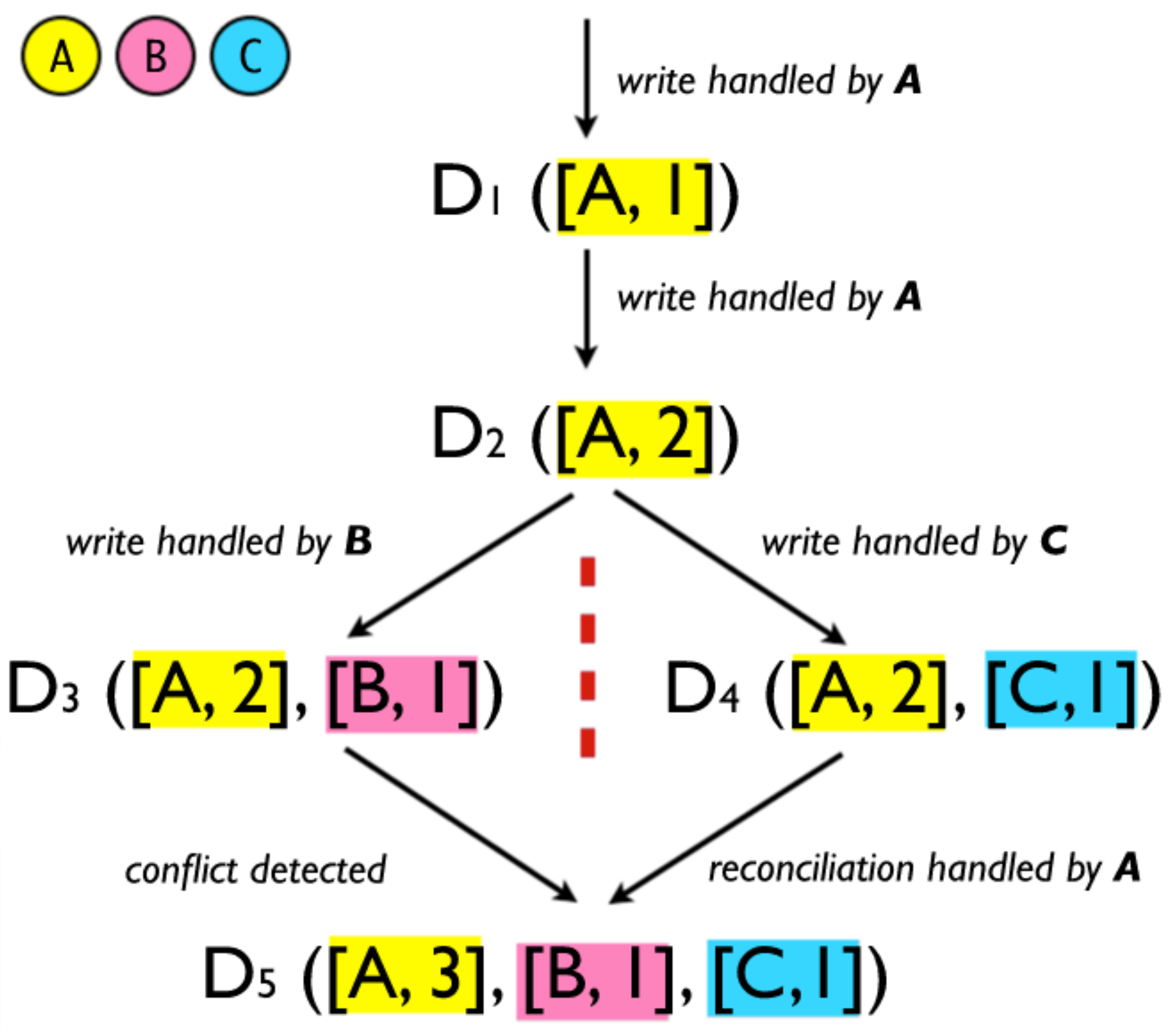
Document version history: a counter for each node that updated the document.

If all update counters in V_1 are smaller or equal to all update counters in V_2, then V_1 precedes V_2.

http://en.wikipedia.org/wiki/Vector_clock

http://pl.atyp.us/wordpress/?p=2601

Vector Clocks & Conflict Detection



A B C

*write handled by **A***

D_1 ([A, 1])

*write handled by **A***

D_2 ([A, 2])

*write handled by **B***

*write handled by **C***

D_3 ([A, 2], [B, 1])

D_4 ([A, 2], [C,1])

conflict detected

*reconciliation handled by **A***

D_5 ([A, 3], [B, 1], [C,1])

Causality-based partial order over events that happen in the system.

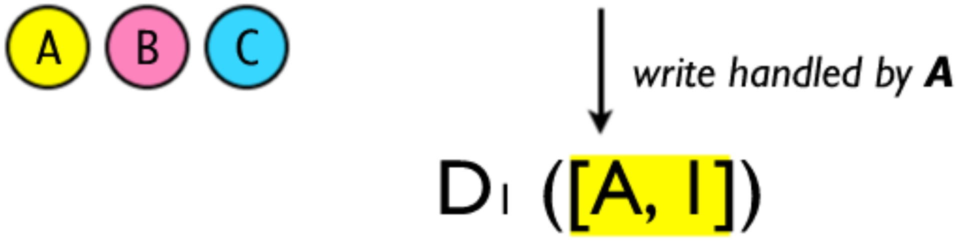
Document version history: a counter for each node that updated the document.

If all update counters in V_1 are smaller or equal to all update counters in V_2, then V_1 precedes V_2.

http://en.wikipedia.org/wiki/Vector_clock

http://pl.atyp.us/wordpress/?p=2601

Vector Clocks & Conflict Detection



D_1 ([A, 1])

Vector Clocks can *detect* a conflict. The conflict *resolution* is left to the application or the user.

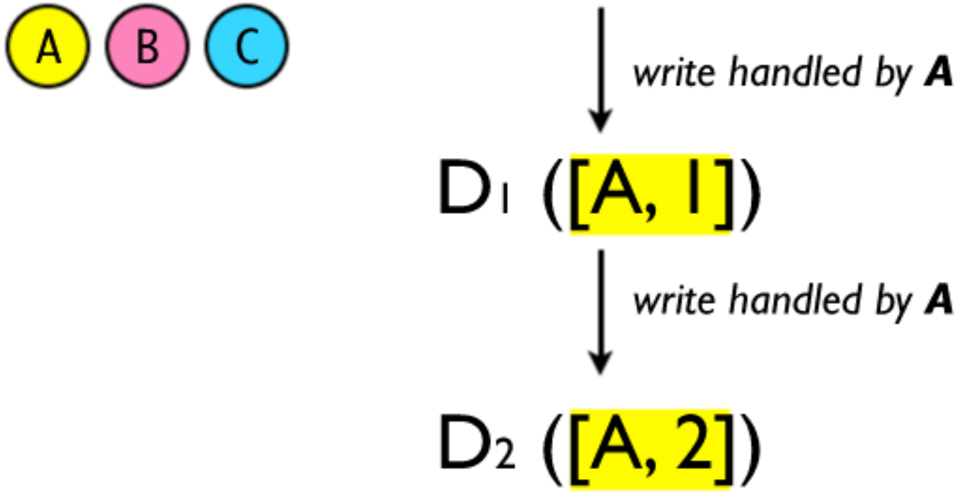
The application *might* resolve conflicts by checking relative timestamps, or with other strategies (like merging the changes).

Vector clocks can grow quite large (!)

http://en.wikipedia.org/wiki/Vector_clock

http://pl.atyp.us/wordpress/?p=2601

Vector Clocks & Conflict Detection



Vector Clocks can *detect* a conflict. The conflict *resolution* is left to the application or the user.

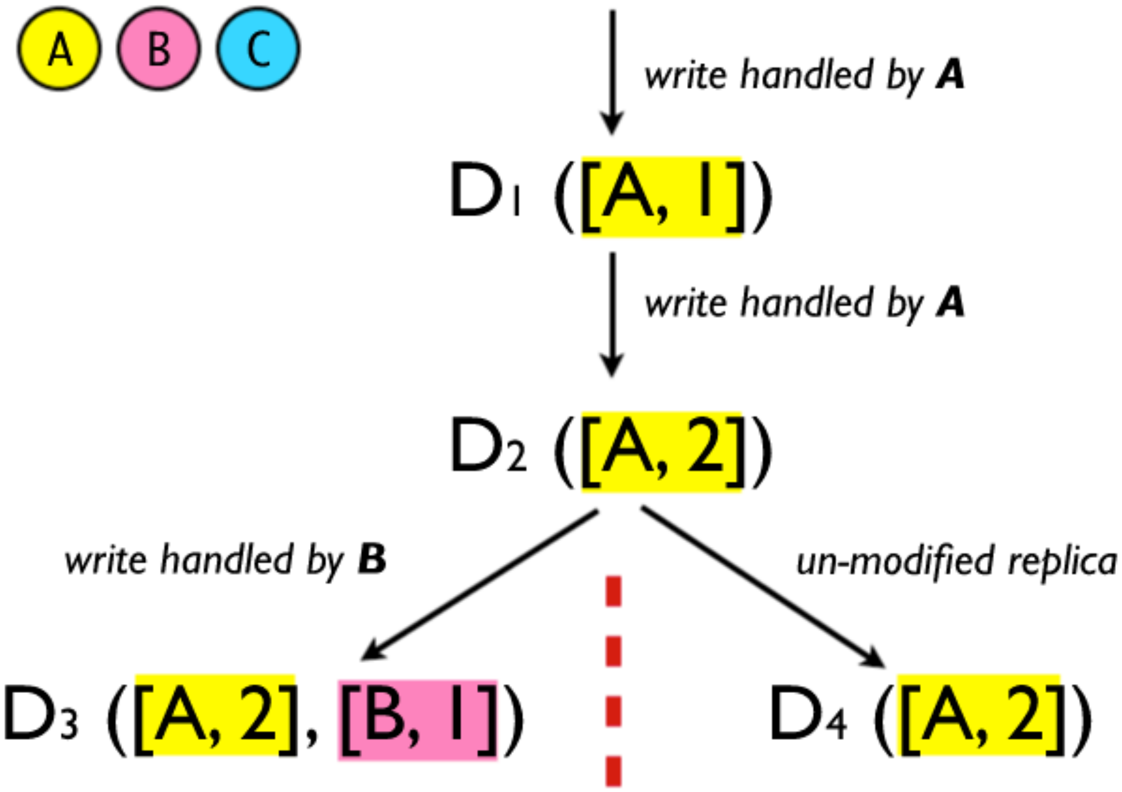
The application *might* resolve conflicts by checking relative timestamps, or with other strategies (like merging the changes).

Vector clocks can grow quite large (!)

http://en.wikipedia.org/wiki/Vector_clock

http://pl.atyp.us/wordpress/?p=2601

Vector Clocks & Conflict Detection



A B C

write handled by ***A***

D1 ([A, 1])

write handled by ***A***

D2 ([A, 2])

write handled by ***B***

un-modified replica

D3 ([A, 2], [B, 1])

D4 ([A, 2])

Vector Clocks can *detect* a conflict. The conflict *resolution* is left to the application or the user.

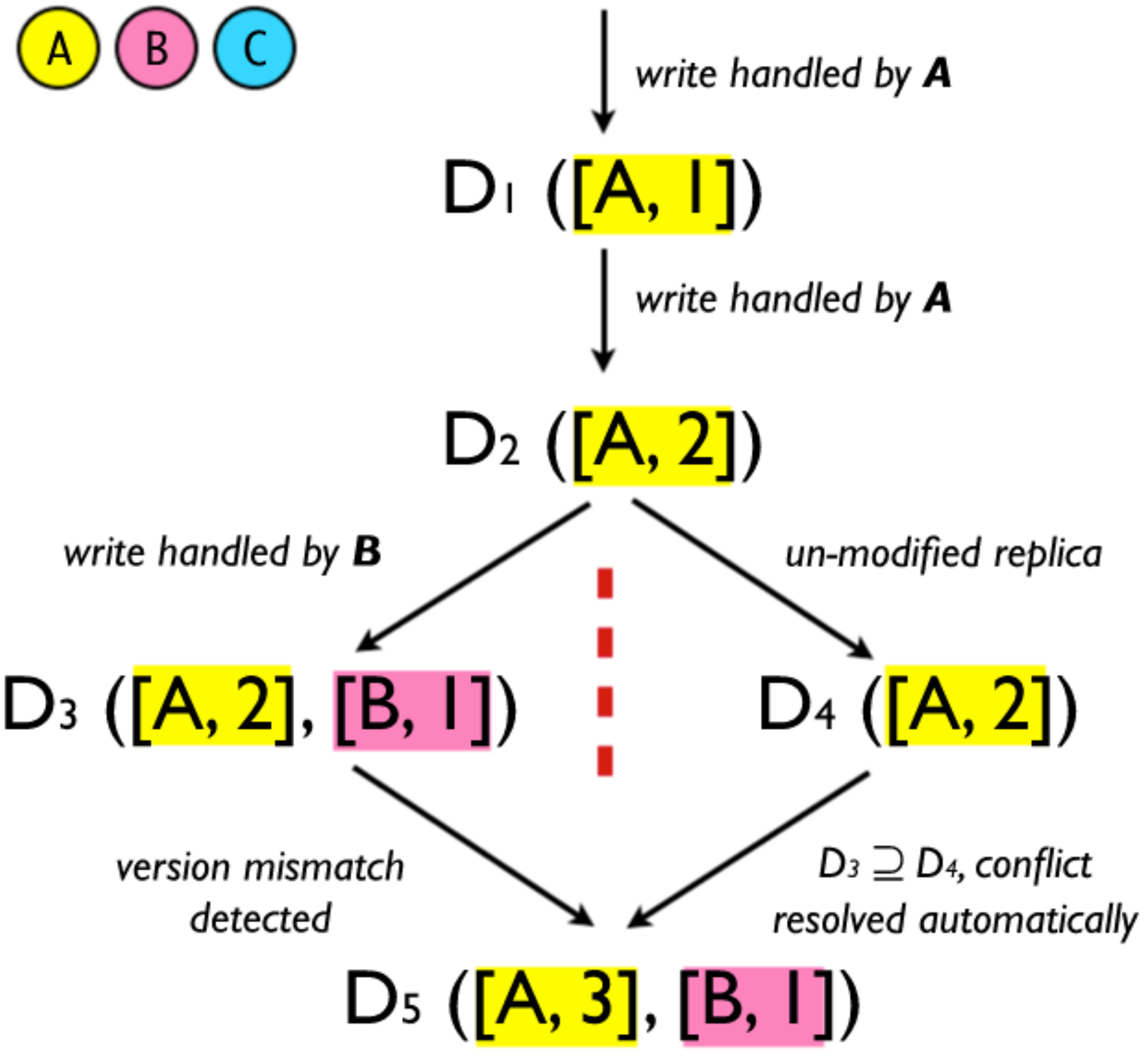
The application *might* resolve conflicts by checking relative timestamps, or with other strategies (like merging the changes).

Vector clocks can grow quite large (!)

http://en.wikipedia.org/wiki/Vector_clock

http://pl.atyp.us/wordpress/?p=2601

Vector Clocks & Conflict Detection



A B C

*write handled by **A***

D_1 ([A, 1])

*write handled by **A***

D_2 ([A, 2])

*write handled by **B***

un-modified replica

D_3 ([A, 2], [B, 1])

D_4 ([A, 2])

version mismatch detected

$D_3 \supseteq D_4$, conflict resolved automatically

D_5 ([A, 3], [B, 1])

Vector Clocks can *detect* a conflict. The conflict *resolution* is left to the application or the user.

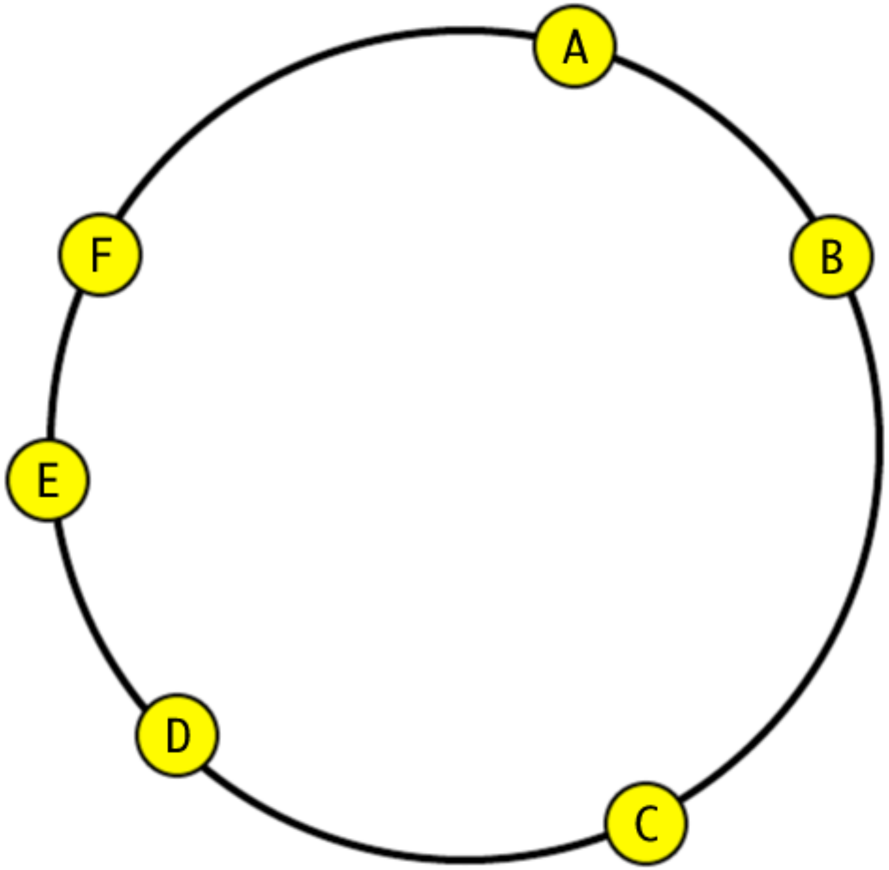
The application *might* resolve conflicts by checking relative timestamps, or with other strategies (like merging the changes).

Vector clocks can grow quite large (!)

http://en.wikipedia.org/wiki/Vector_clock

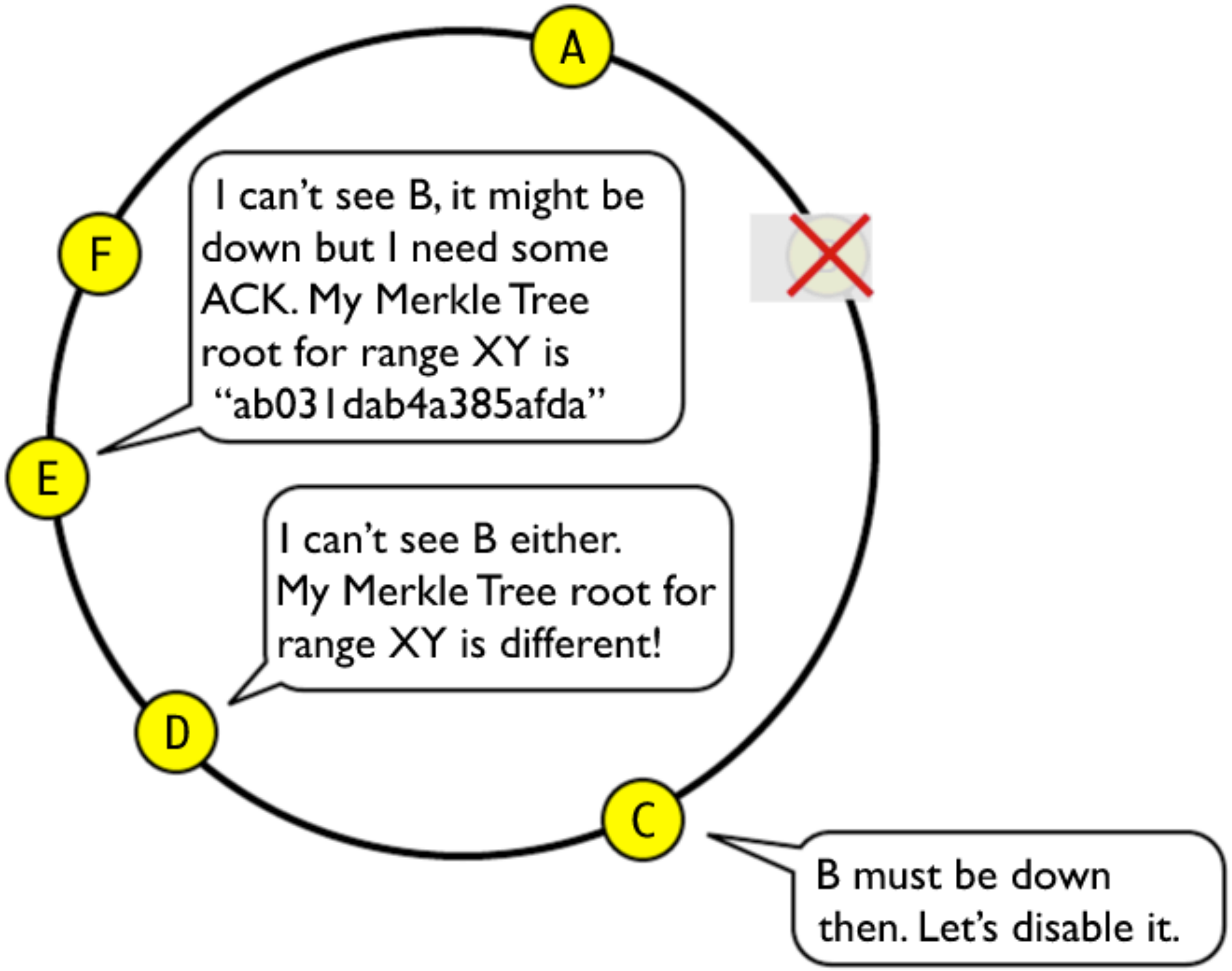
http://pl.atyp.us/wordpress/?p=2601

Gossip Protocol + Hinted Handoff



periodic, pairwise, inter-process interactions of bounded size among randomly-chosen peers

Gossip Protocol + Hinted Handoff



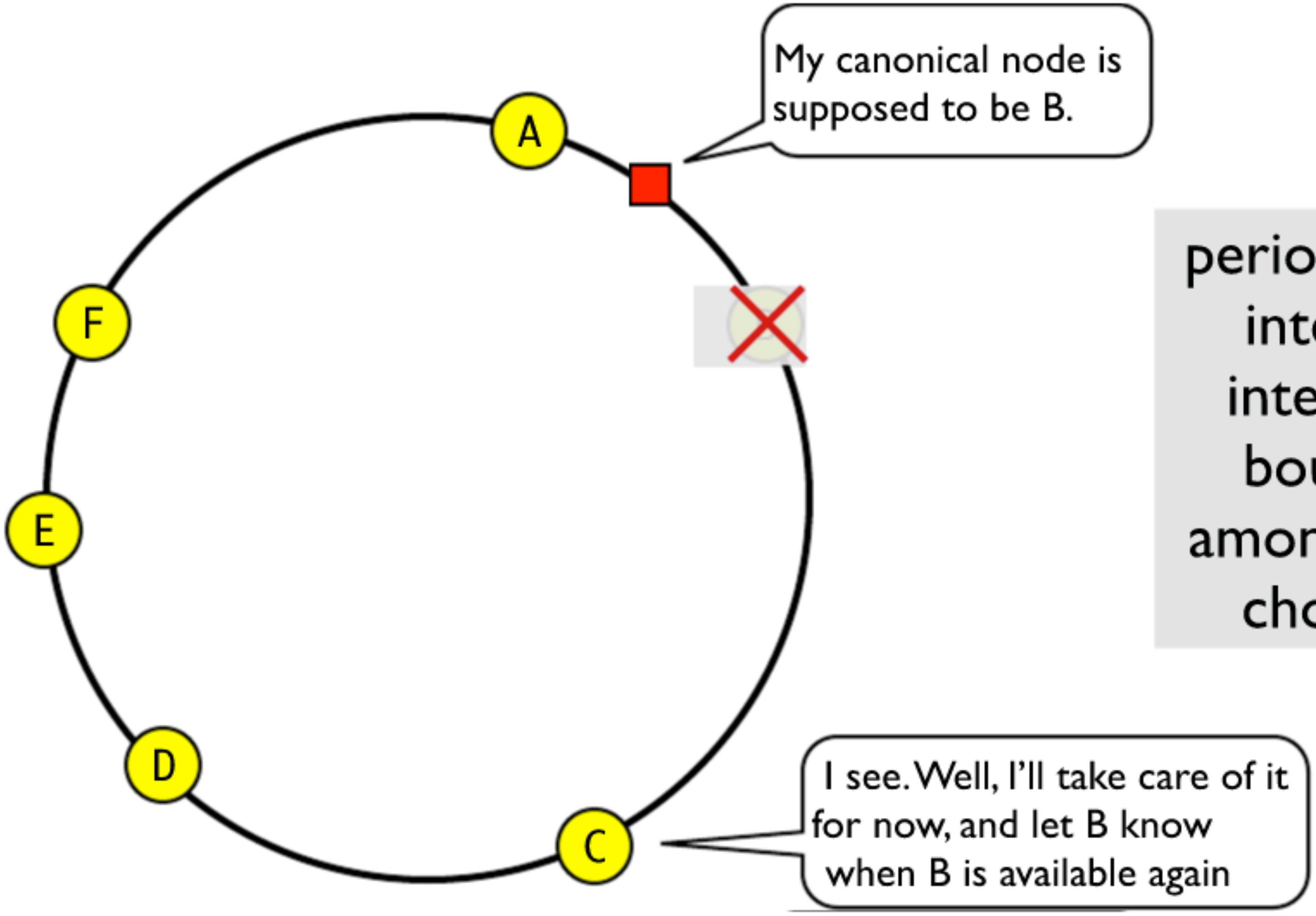
I can't see B, it might be down but I need some ACK. My Merkle Tree root for range XY is "ab031dab4a385afda"

I can't see B either. My Merkle Tree root for range XY is different!

B must be down then. Let's disable it.

periodic, pairwise, inter-process interactions of bounded size among randomly-chosen peers

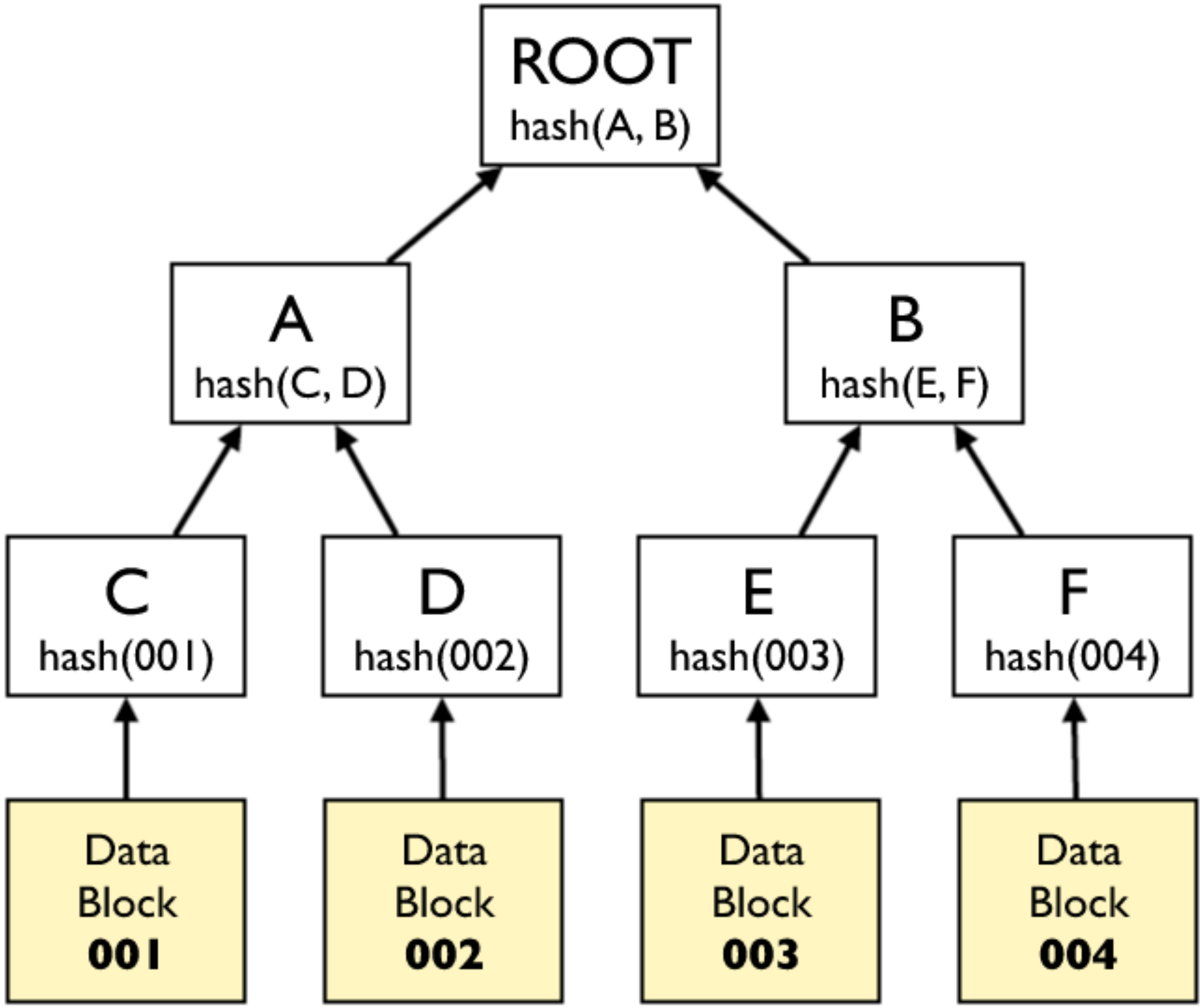
Gossip Protocol + Hinted Handoff



My canonical node is supposed to be B.
A
F
E
D
C
I see. Well, I'll take care of it for now, and let B know when B is available again

periodic, pairwise, inter-process interactions of bounded size among randomly-chosen peers

Merkle Trees (Hash Trees)



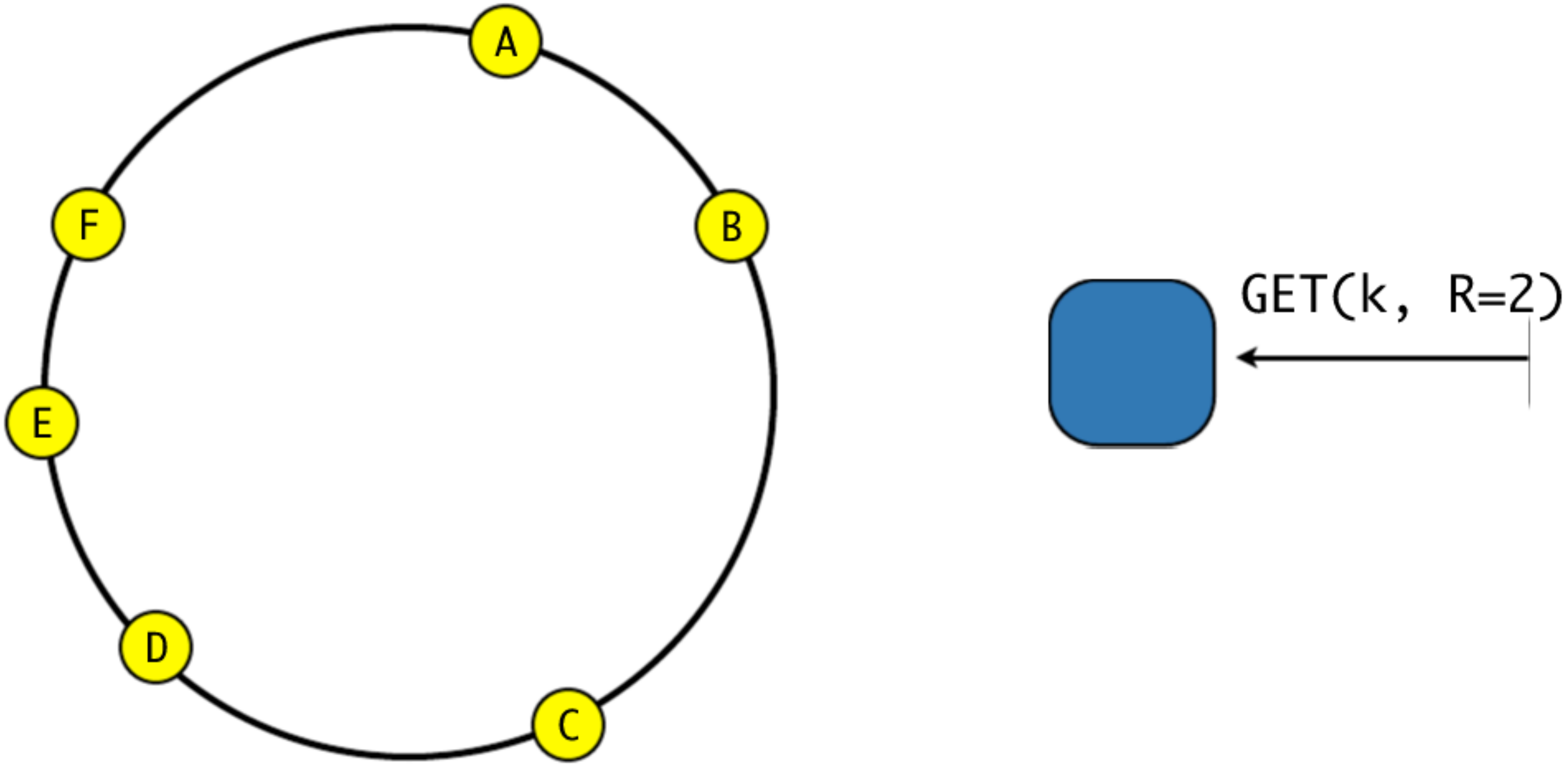
Leaves: hashes of data blocks.

Nodes: hashes of their children.

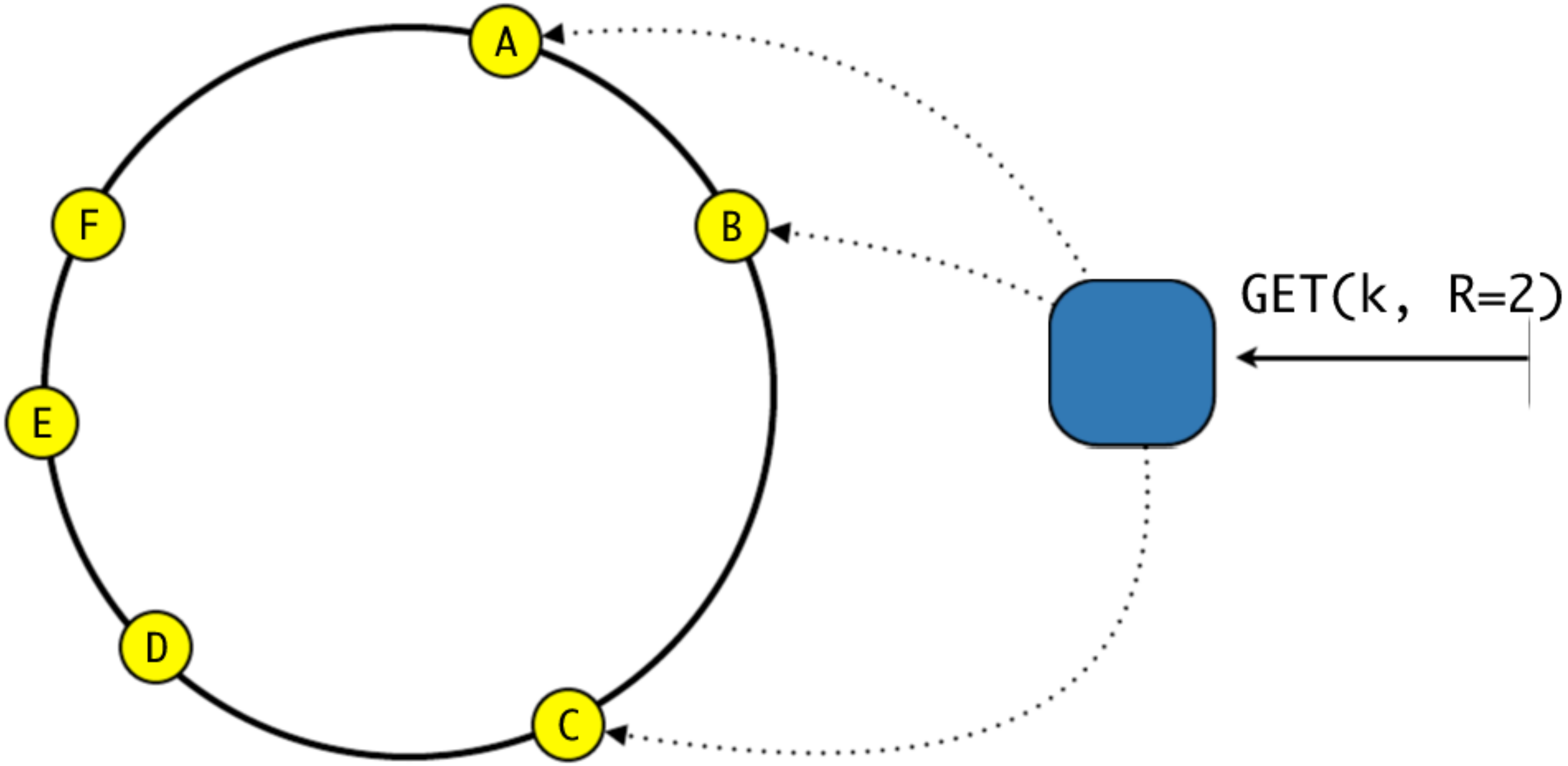
Used to detect inconsistencies between replicas (anti-entropy) and to minimise the amount of transferred data

http://en.wikipedia.org/wiki/Hash_tree

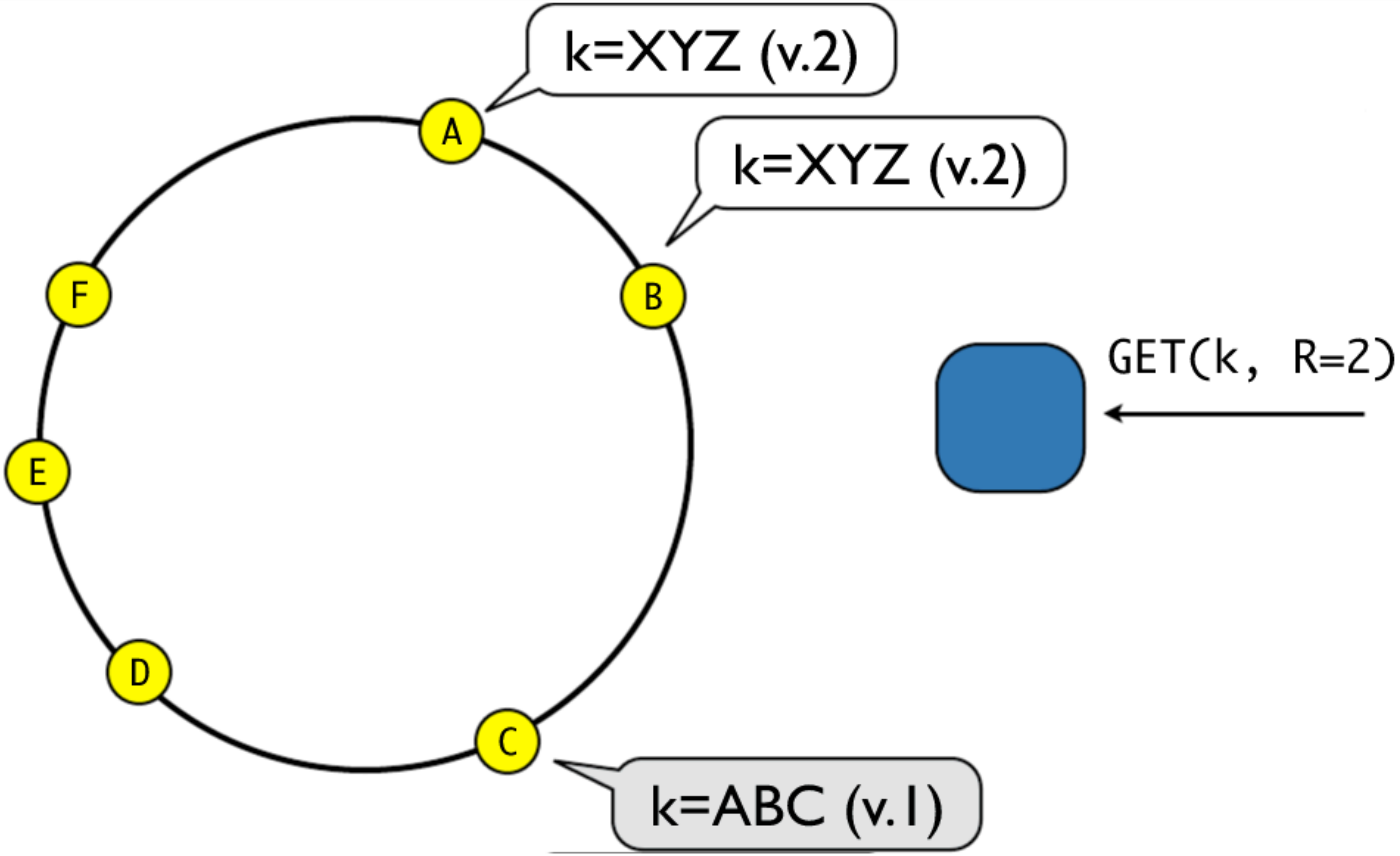
Read Repair



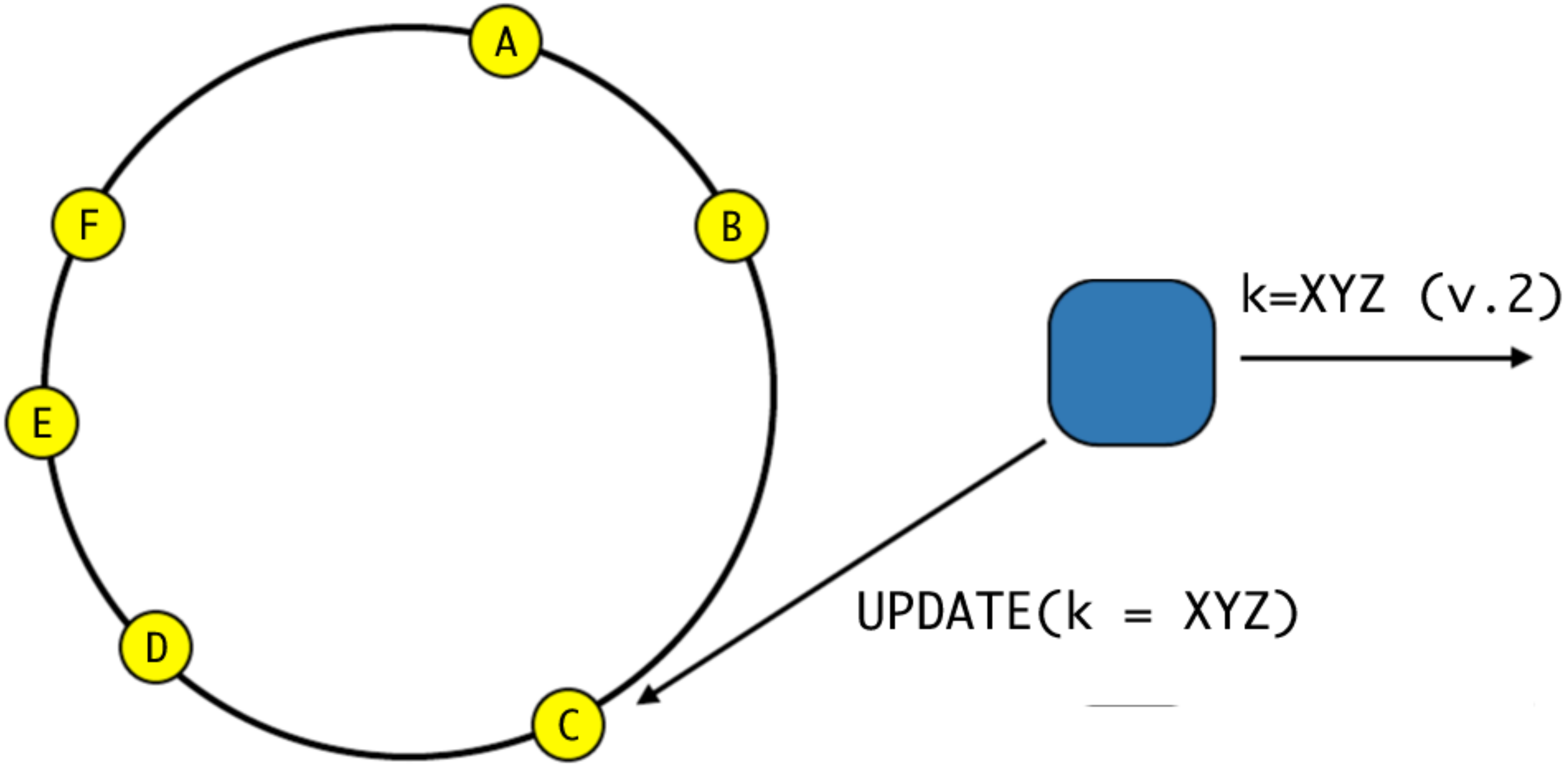
Read Repair



Read Repair

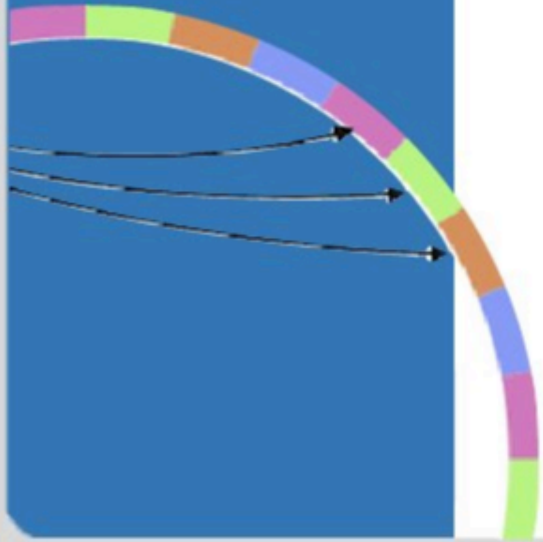


Read Repair

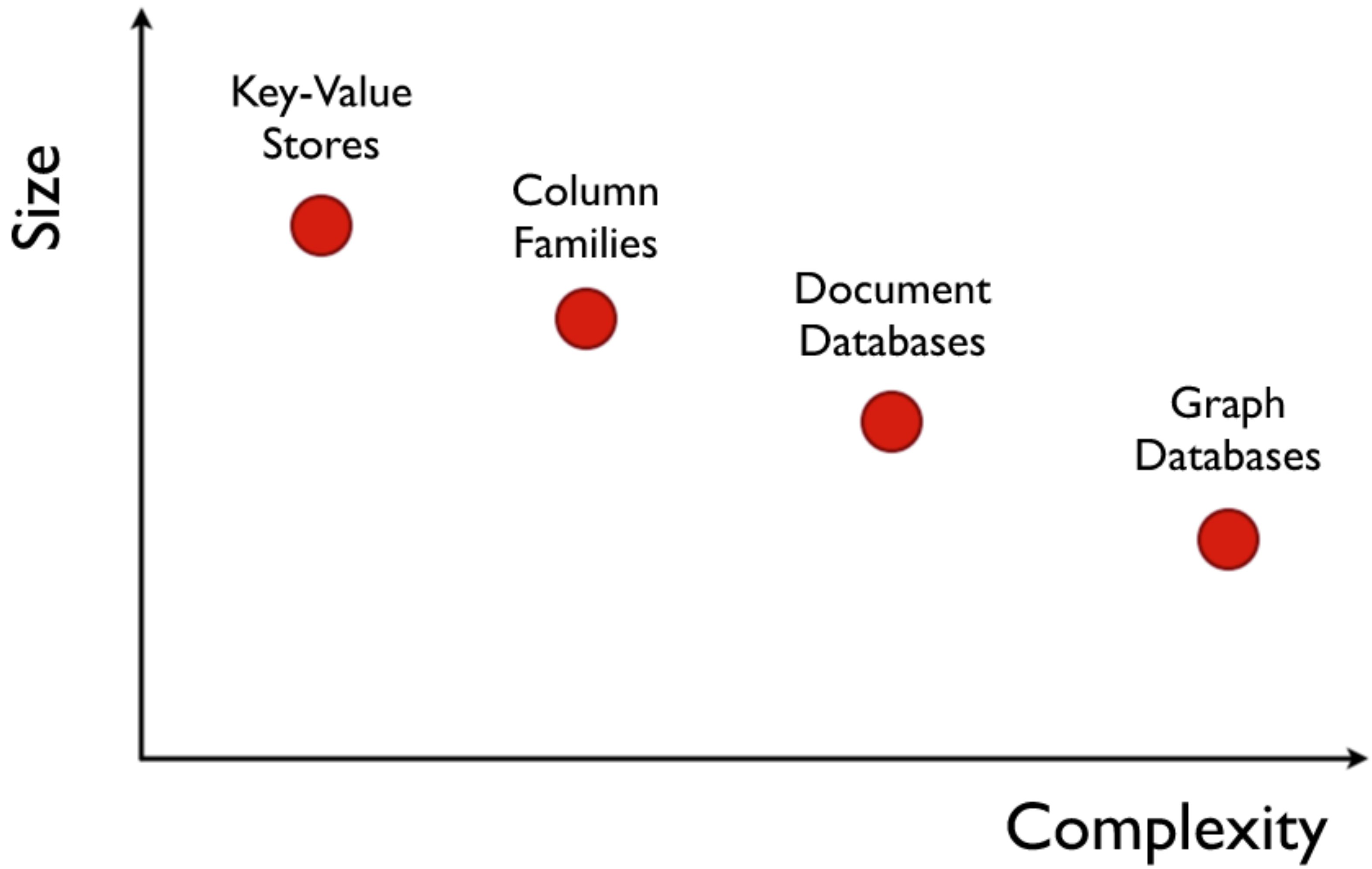


NoSQL Break-down

Key-value stores, Column Families, Document-oriented dbs, Graph databases



Focus Of Different Data Models

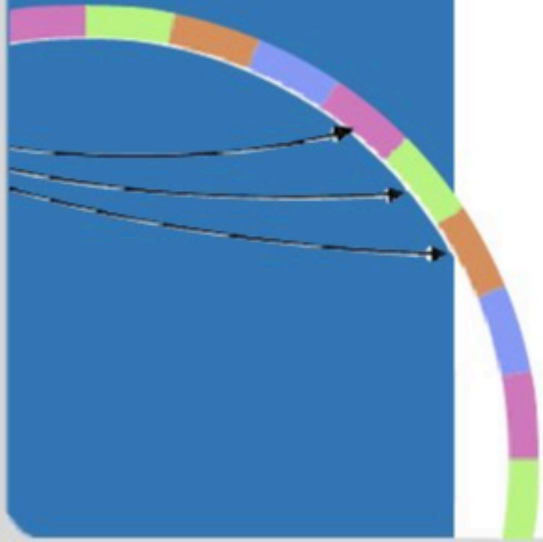


http://www.slideshare.net/emileifrem/nosql-east-a-nosql-overview-and-the-benefits-of-graph-databases

1) Key-value stores

Amazon Dynamo Paper

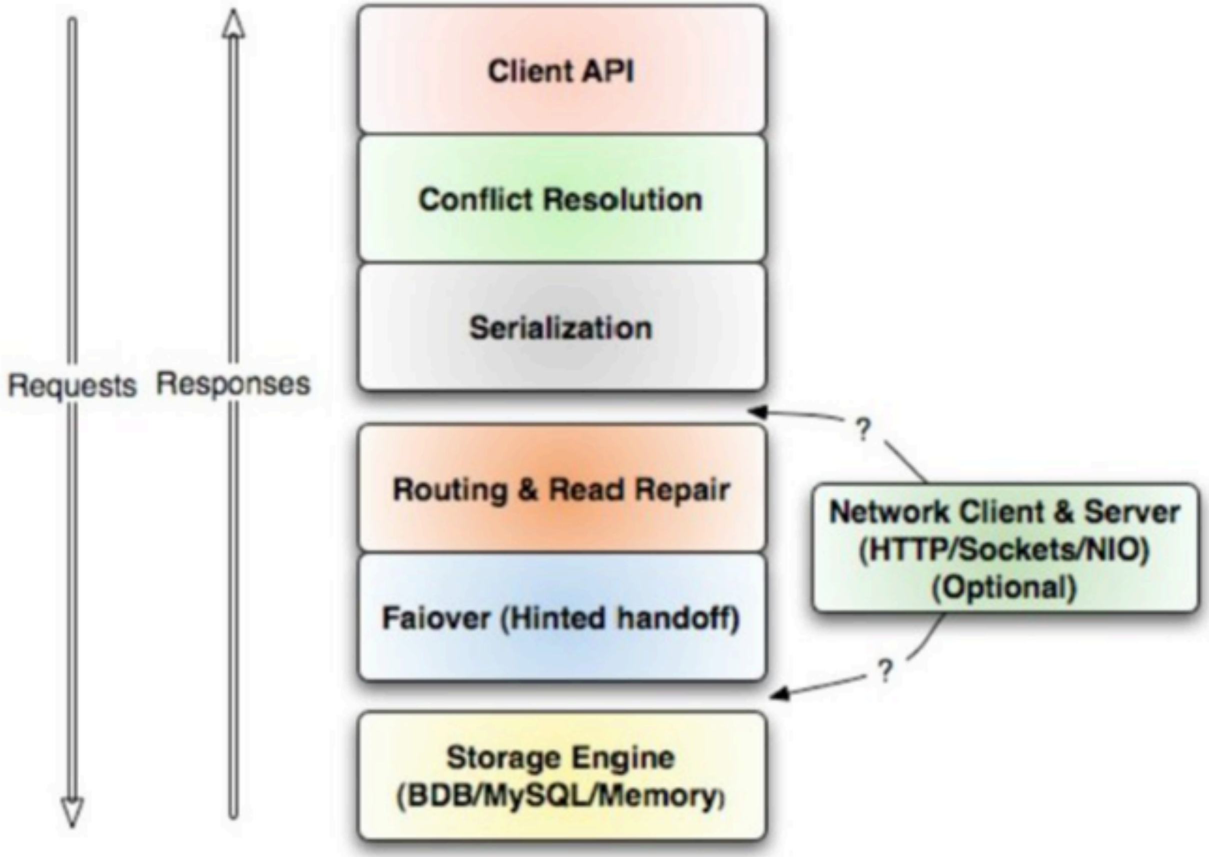
Data model: collection of key-value pairs



Voldemort

AP

Dynamo DHT implementation
Consistent hashing,Vector clocks

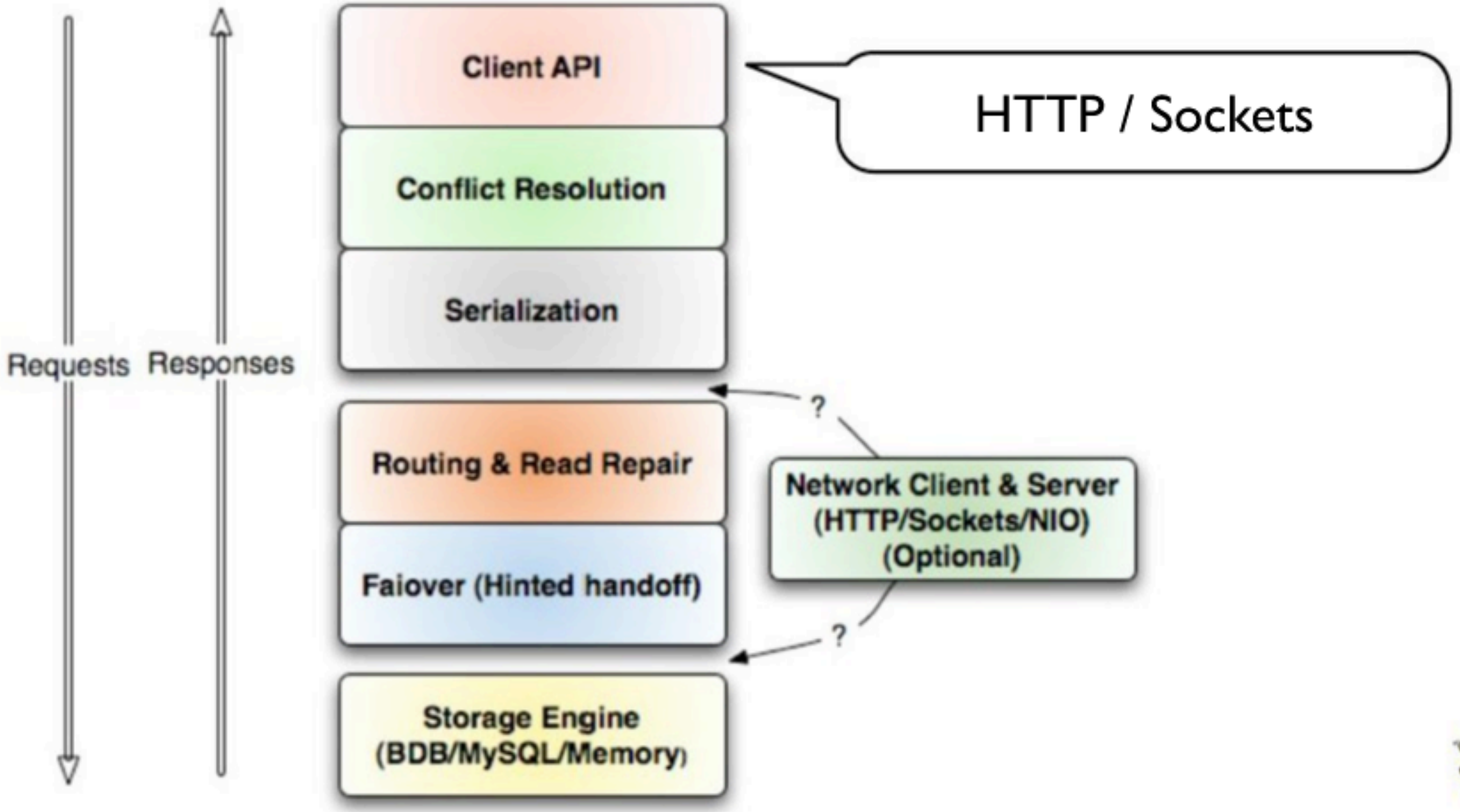


LICENSE
Apache 2
LANGUAGE
Java
API/PROTOCOL
HTTP Java Thrift Avro Protobuf
PERSISTENCE
Pluggable BDB/MySQL
CONCURRENCY
MVCC

Voldemort

AP

Dynamo DHT implementation
Consistent hashing, Vector clocks



Requests

Responses

Client API

HTTP / Sockets

Conflict Resolution

Serialization

?

Routing & Read Repair

Network Client & Server
(HTTP/Sockets/NIO)
(Optional)

Faiover (Hinted handoff)

?

Storage Engine
(BDB/MySQL/Memory)

LICENSE

Apache 2

LANGUAGE

Java

API/PROTOCOL

HTTP Java
Thrift
Avro
Protobuf

PERSISTENCE

Pluggable
BDB/MySQL

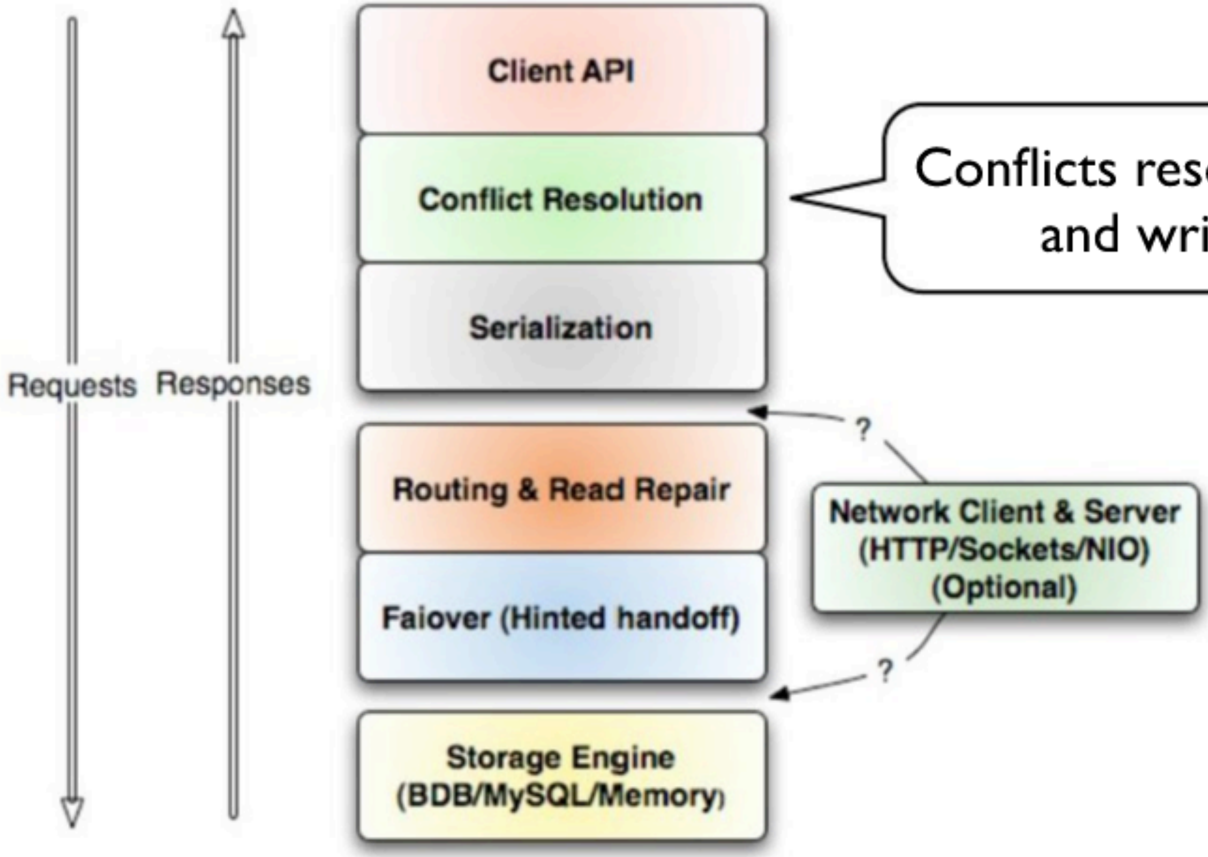
CONCURRENCY

MVCC

Voldemort

AP

Dynamo DHT implementation
Consistent hashing, Vector clocks



Requests

Responses

Client API

Conflict Resolution

Serialization

Routing & Read Repair

Faiover (Hinted handoff)

Storage Engine
(BDB/MySQL/Memory)

Network Client & Server
(HTTP/Sockets/NIO)
(Optional)

?

?

Conflicts resolved at read and write time

LICENSE

Apache 2

LANGUAGE

Java

API/PROTOCOL

HTTP Java
Thrift
Avro
Protobuf

PERSISTENCE

Pluggable
BDB/MySQL

CONCURRENCY

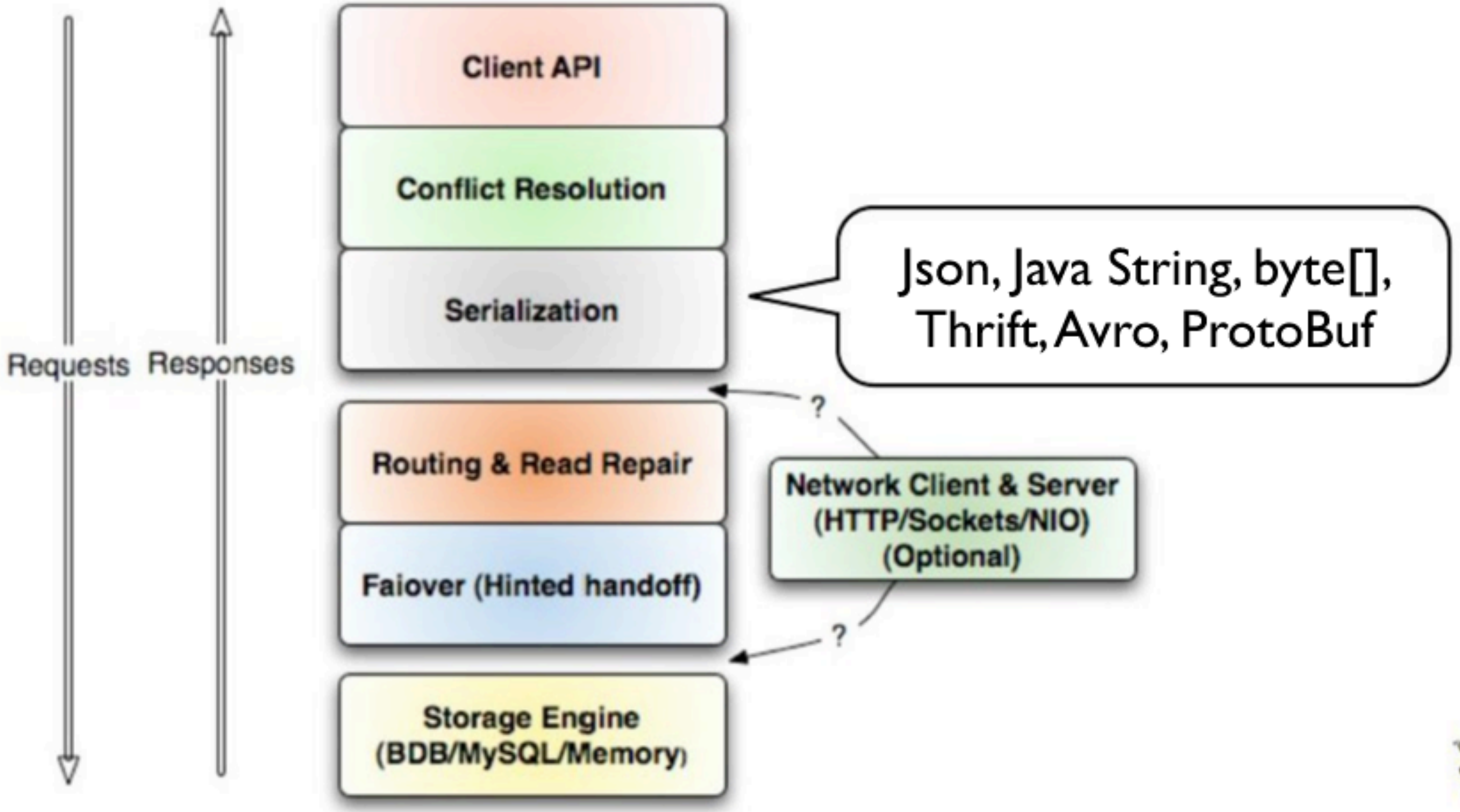
MVCC

Voldemort

AP

Dynamo DHT implementation
Consistent hashing, Vector clocks

Linked in



Json, Java String, byte[],
Thrift, Avro, ProtoBuf

LICENSE

Apache 2

LANGUAGE

Java

API/PROTOCOL

HTTP Java
Thrift
Avro
Protobuf

PERSISTENCE

Pluggable
BDB/MySQL

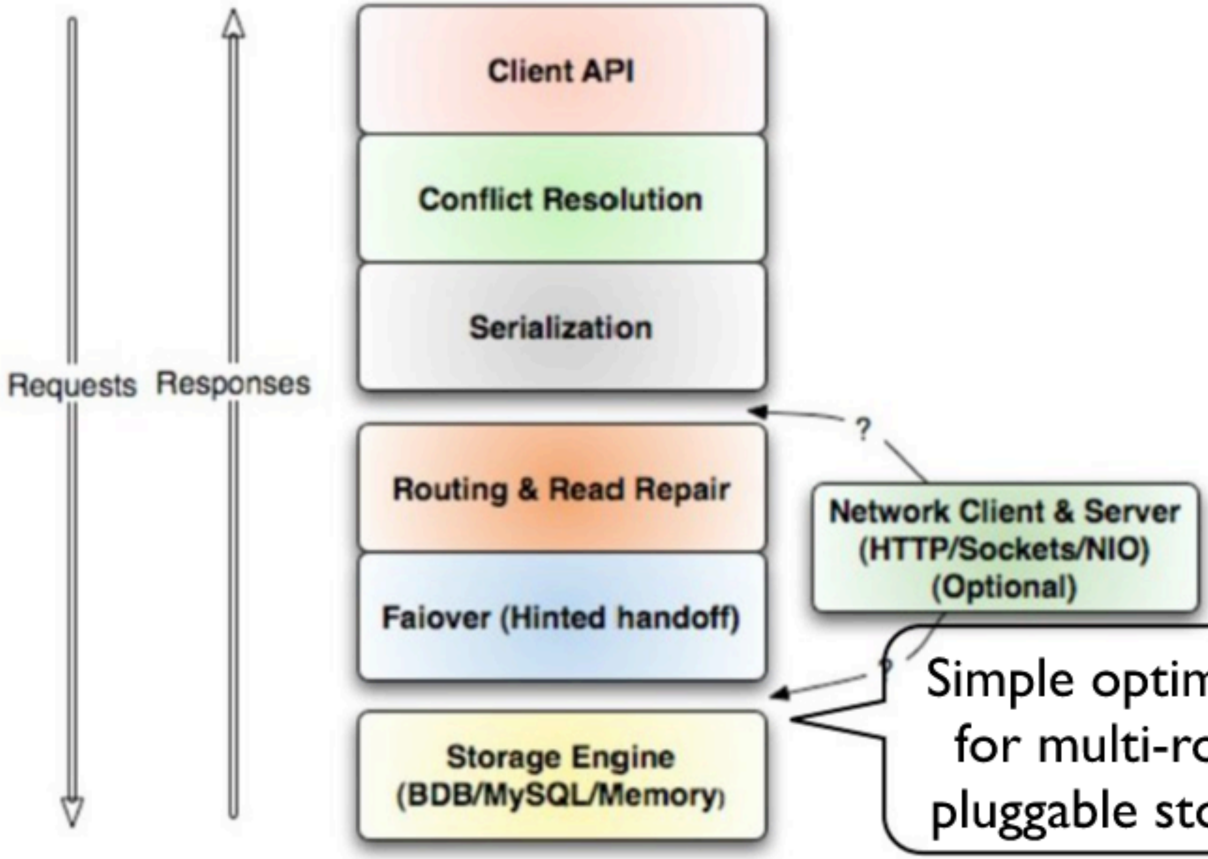
CONCURRENCY

MVCC

Voldemort

AP

Dynamo DHT implementation
Consistent hashing,Vector clocks



Simple optimistic locking for multi-row updates, pluggable storage engine

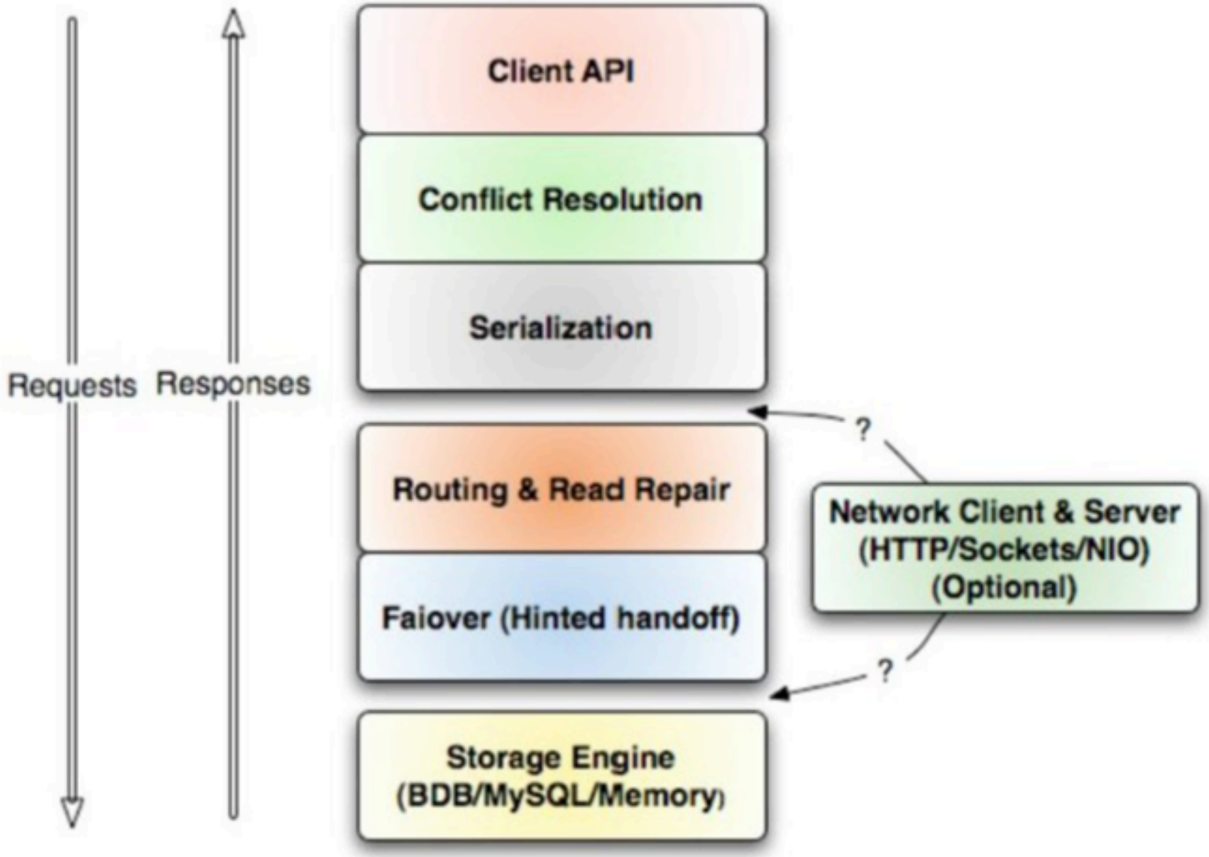
LICENSE
Apache 2
LANGUAGE
Java
API/PROTOCOL
HTTP Java Thrift Avro Protobuf
PERSISTENCE
Pluggable BDB/MySQL
CONCURRENCY
MVCC

Voldemort

AP

Dynamo DHT implementation
Consistent hashing,Vector clocks

Linked in



LICENSE
Apache 2
LANGUAGE
Java
API/PROTOCOL
HTTP Java Thrift Avro Protobuf
PERSISTENCE
Pluggable BDB/MySQL
CONCURRENCY
MVCC

hadoop

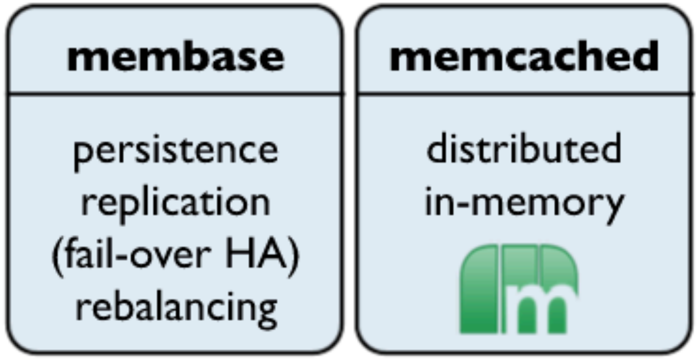
Membase

CP

DHT (K-V), no SPoF

membase.org

“VBuckets”



membase

persistence
replication
(fail-over HA)
rebalancing

memcached

distributed
in-memory

Unit of consistency and replication
Owner of a subset of the cluster key space

LICENSE

Apache 2

LANGUAGE

C/C++
Erlang

API/PROTOCOL

REST/JSON
memcached

http://dustin.github.com/2010/06/29/memcached-vbuckets.html

Membase

CP

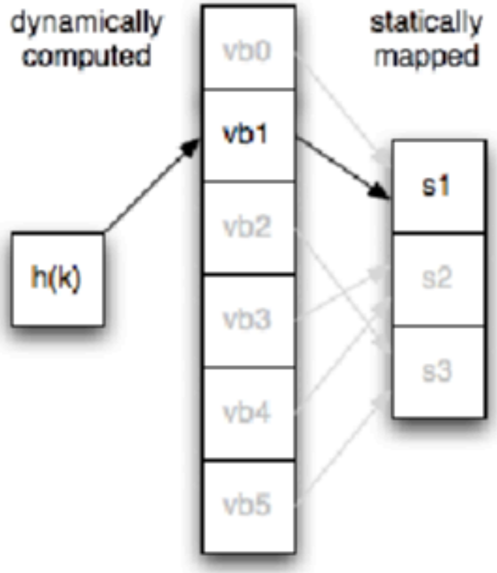
DHT (K-V), no SPoF

“VBuckets”

membase	memcached
persistence replication (fail-over HA) rebalancing	distributed in-memory

Unit of consistency and replication
Owner of a subset of the cluster key space

membase.org



h(k) → vb1 → s1

hash function + table lookup

LICENSE

Apache 2

LANGUAGE

C/C++
Erlang

API/PROTOCOL

REST/JSON
memcached

http://dustin.github.com/2010/06/29/memcached-vbuckets.html

Membase

CP

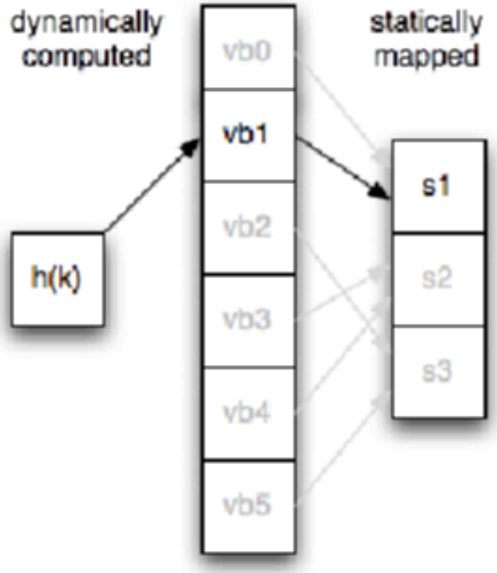
DHT (K-V), no SPoF

"VBuckets"

membase	memcached
persistence replication (fail-over HA) rebalancing	distributed in-memory

Unit of consistency and replication
Owner of a subset of the cluster key space

membase.org



h(k) → vb1 → s1

hash function + table lookup

LICENSE

Apache 2

LANGUAGE

C/C++
Erlang

API/PROTOCOL

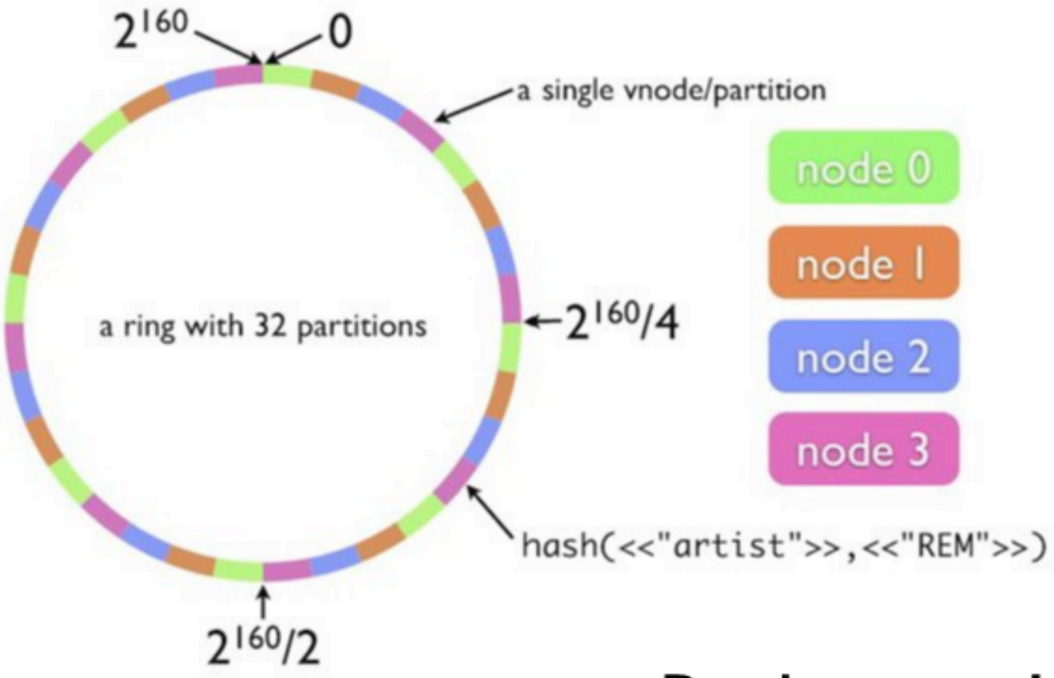
REST/JSON
memcached

All metadata kept in memory (high throughput / low latency).
Manual/Programmatic failover via the Management REST API.

http://dustin.github.com/2010/06/29/memcached-vbuckets.html

Riak

AP



basho riak

LICENSE

Apache 2

LANGUAGE

C, Erlang

API/PROTOCOL

REST HTTP
*
ProtoBuf

Buckets → K-V

“Links” (~relations)

Targeted JS Map/Reduce

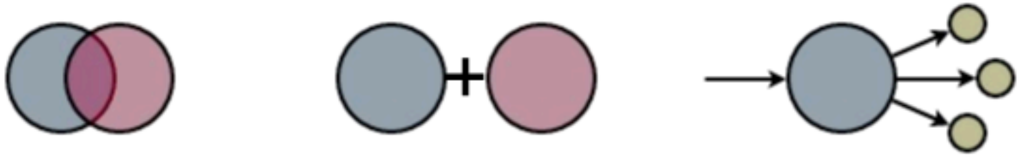
Tune-able consistency (one-quorum-all)

Redis

CP

~~K-V store~~ "Data Structures Server"

Map, Set, Sorted Set, Linked List
Set/Queue operations, Counters, Pub-Sub, Volatile keys



10-100K op/s (whole dataset in RAM + VM)

Persistence via snapshotting (tunable fsync freq.)

Distributed if client supports consistent hashing

LICENSE
BSD
LANGUAGE
ANSI C
API
*
PROTOCOL
Telnet-like
PERSISTENCE
in memory bg snapshots
REPLICATION
master-slave

http://redis.io/presentation/Redis_Cluster.pdf

2) Column Families

Google BigTable paper
Data model: big table, column families

Google BigTable Paper

Sparse, distributed, persistent multi-dimensional sorted map indexed by (*row_key*, *column_key*, *timestamp*)

http://labs.google.com/papers/bigtable-osdi06.pdf

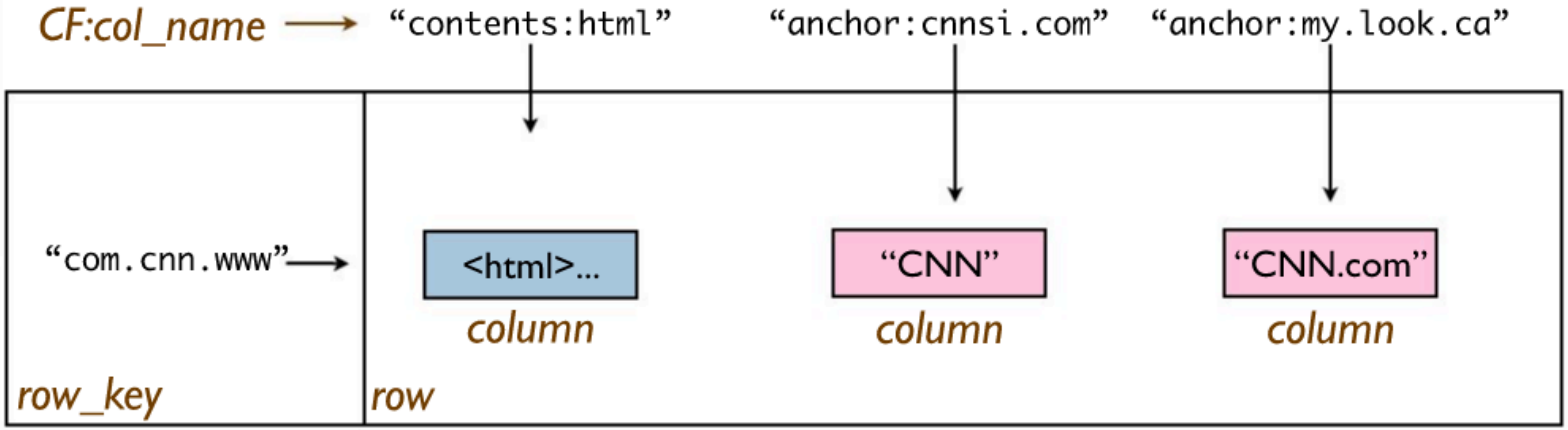
Google BigTable Paper

Sparse, distributed, persistent multi-dimensional sorted map indexed by (*row_key*, *column_key*, *timestamp*)

http://labs.google.com/papers/bigtable-osdi06.pdf

Google BigTable Paper

Sparse, distributed, persistent multi-dimensional sorted map indexed by (*row_key*, *column_key*, *timestamp*)



CF:col_name → "contents:html" "anchor:cnnsi.com" "anchor:my.look.ca"

"com.cnn.www" →

<html>...

column

"CNN"

column

"CNN.com"

column

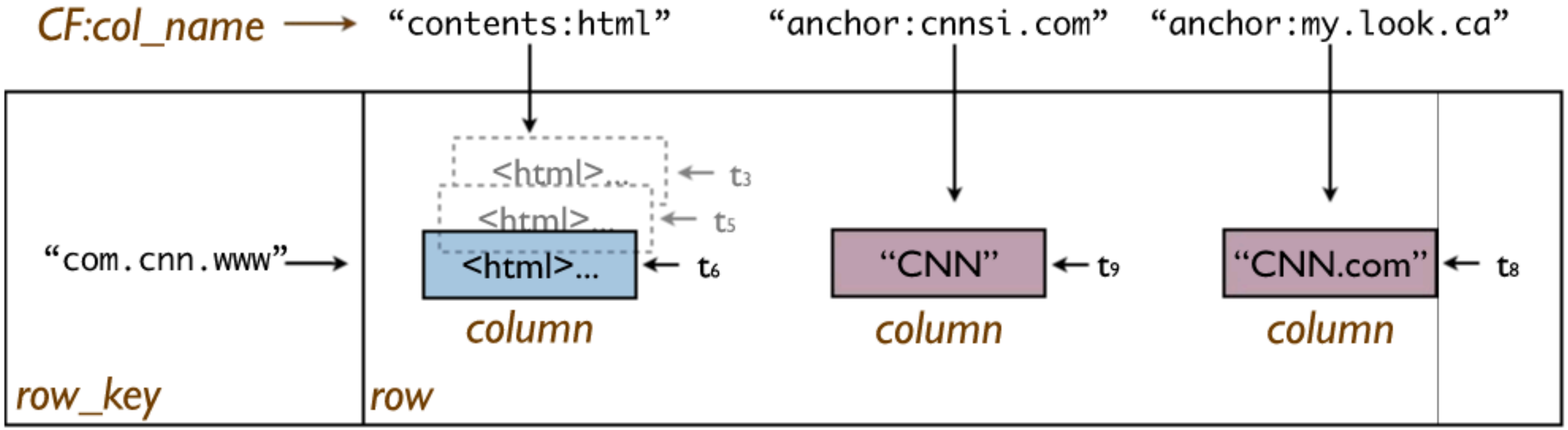
row_key

row

http://labs.google.com/papers/bigtable-osdi06.pdf

Google BigTable Paper

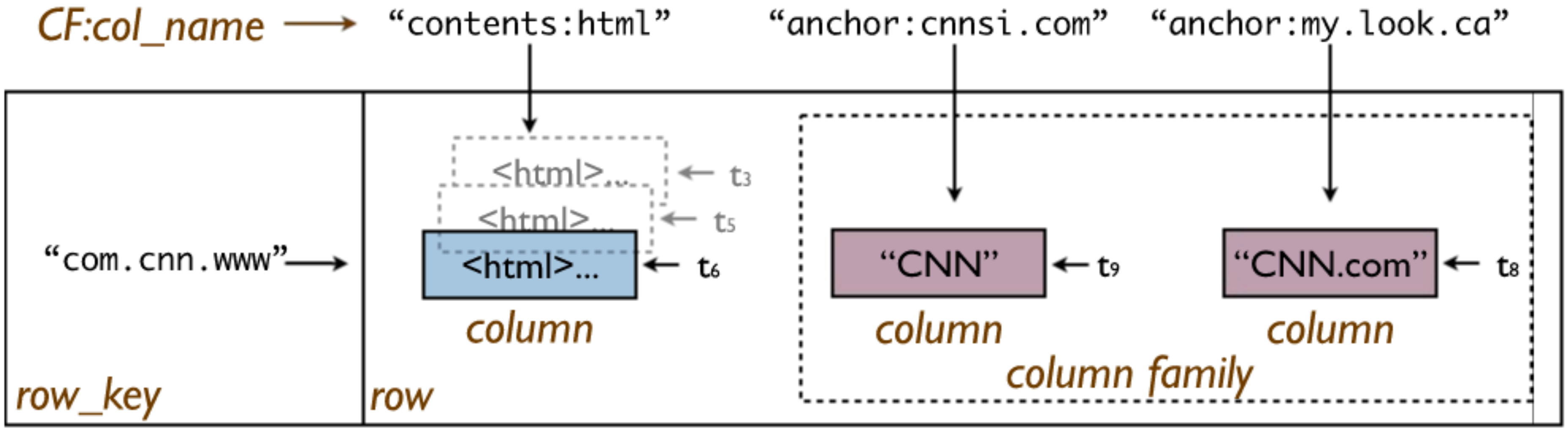
Sparse, distributed, persistent multi-dimensional sorted map indexed by (*row_key*, *column_key*, *timestamp*)



http://labs.google.com/papers/bigtable-osdi06.pdf

Google BigTable Paper

Sparse, distributed, persistent multi-dimensional sorted map indexed by (*row_key*, *column_key*, *timestamp*)

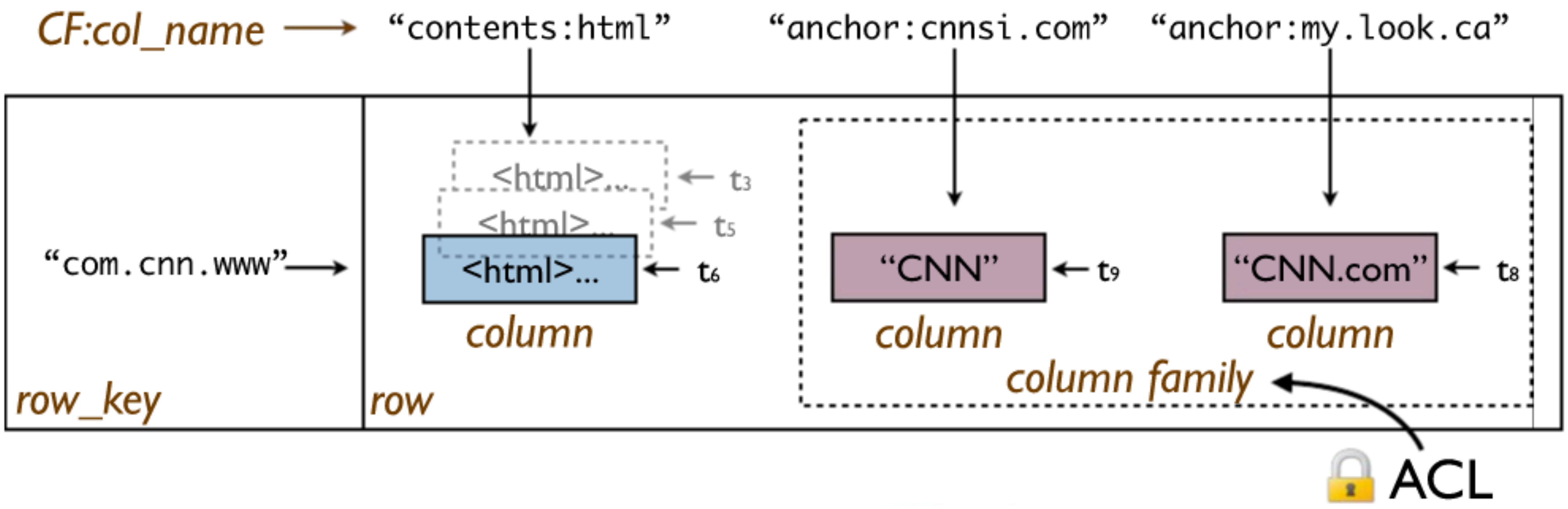


http://labs.google.com/papers/bigtable-osdi06.pdf

Google BigTable Paper

Sparse, distributed, persistent multi-dimensional sorted map indexed by (*row_key*, *column_key*, *timestamp*)

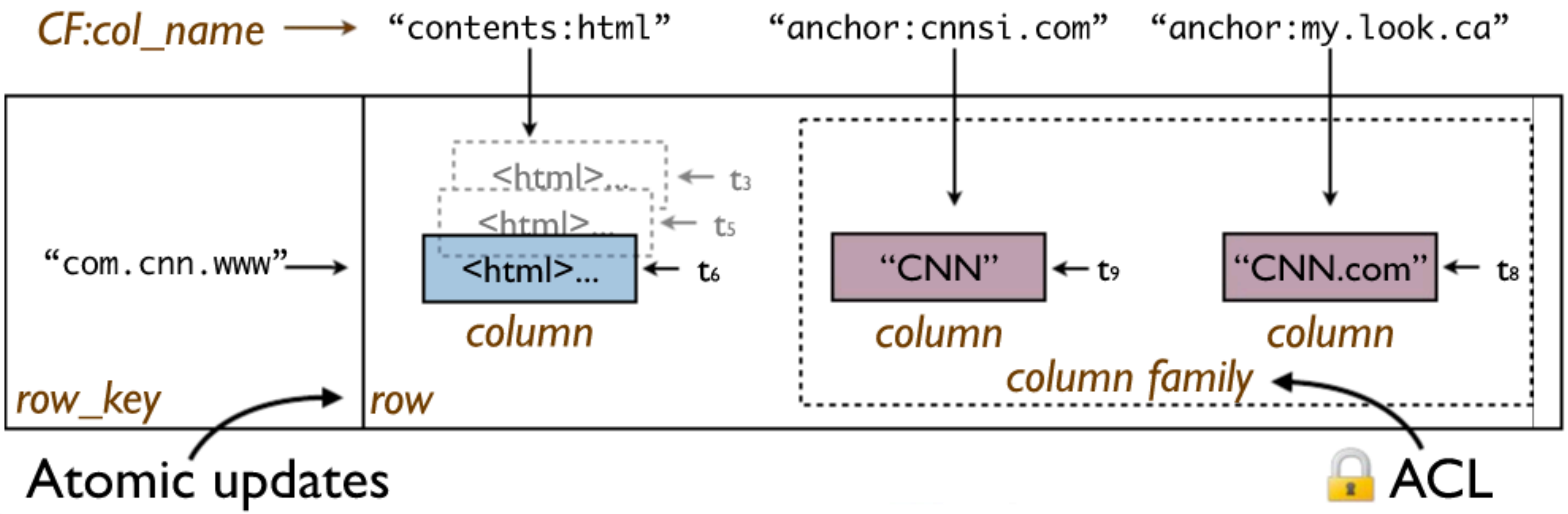
Google™



http://labs.google.com/papers/bigtable-osdi06.pdf

Google BigTable Paper

Sparse, distributed, persistent multi-dimensional sorted map indexed by (*row_key*, *column_key*, *timestamp*)



CF:col_name →

“contents:html”

“anchor:cnnsi.com”

“anchor:my.look.ca”

“com.cnn.www” →

<html>... ← t_3

<html>... ← t_5

<html>... ← t_6

“CNN” ← t_9

“CNN.com” ← t_8

column

column

column

column family

row_key

row

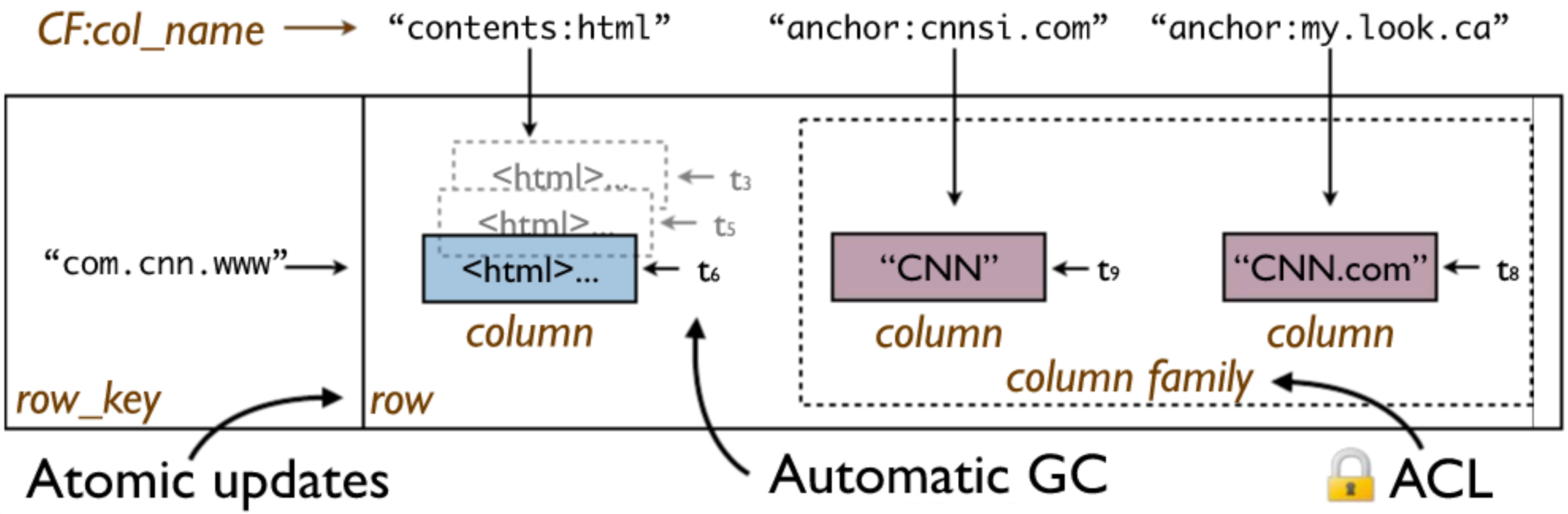
Atomic updates

ACL

http://labs.google.com/papers/bigtable-osdi06.pdf

Google BigTable Paper

Sparse, distributed, persistent multi-dimensional sorted map indexed by (*row_key*, *column_key*, *timestamp*)



http://labs.google.com/papers/bigtable-osdi06.pdf

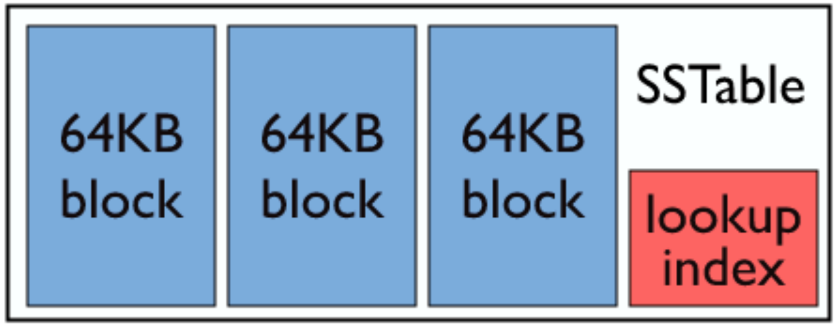
Google BigTable: Data Structure

SSTable

Smallest building block

Persistent immutable Map[k,v]

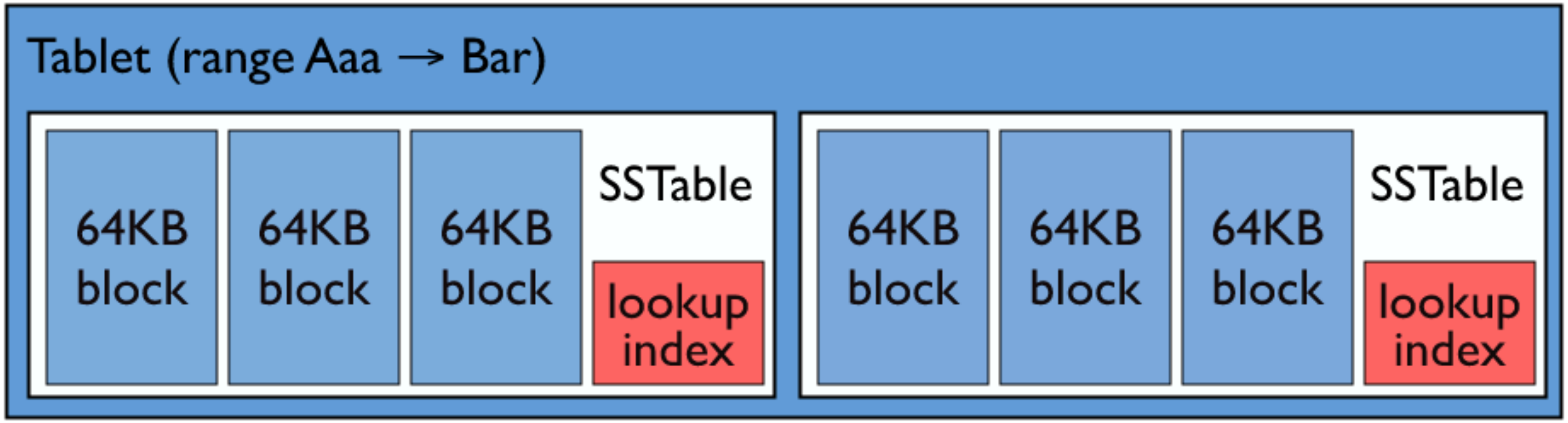
Operations: lookup by key / key range scan



Google BigTable: Data Structure

Tablet

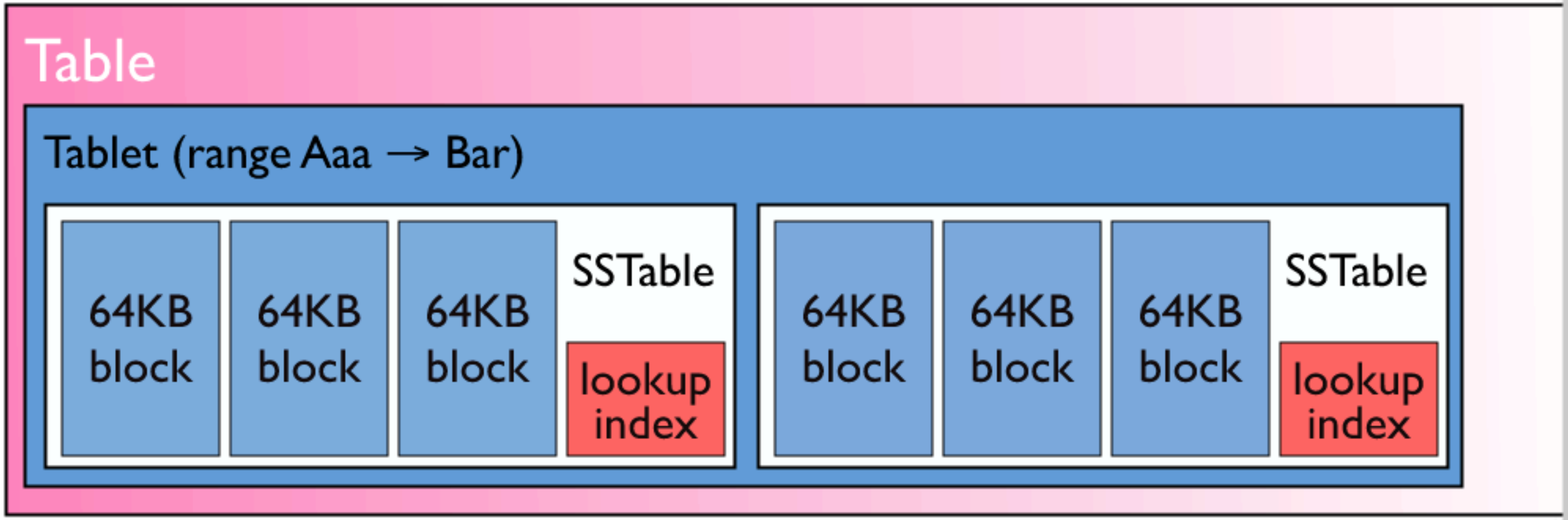
Dynamically partitioned range of rows
Built from multiple SSTables
Unit of distribution and load balancing



Google BigTable: Data Structure

Table

Multiple Tablets (table segments) make up a table



Table

Tablet (range Aaa → Bar)

64KB block

64KB block

64KB block

SSTable

lookup index

64KB block

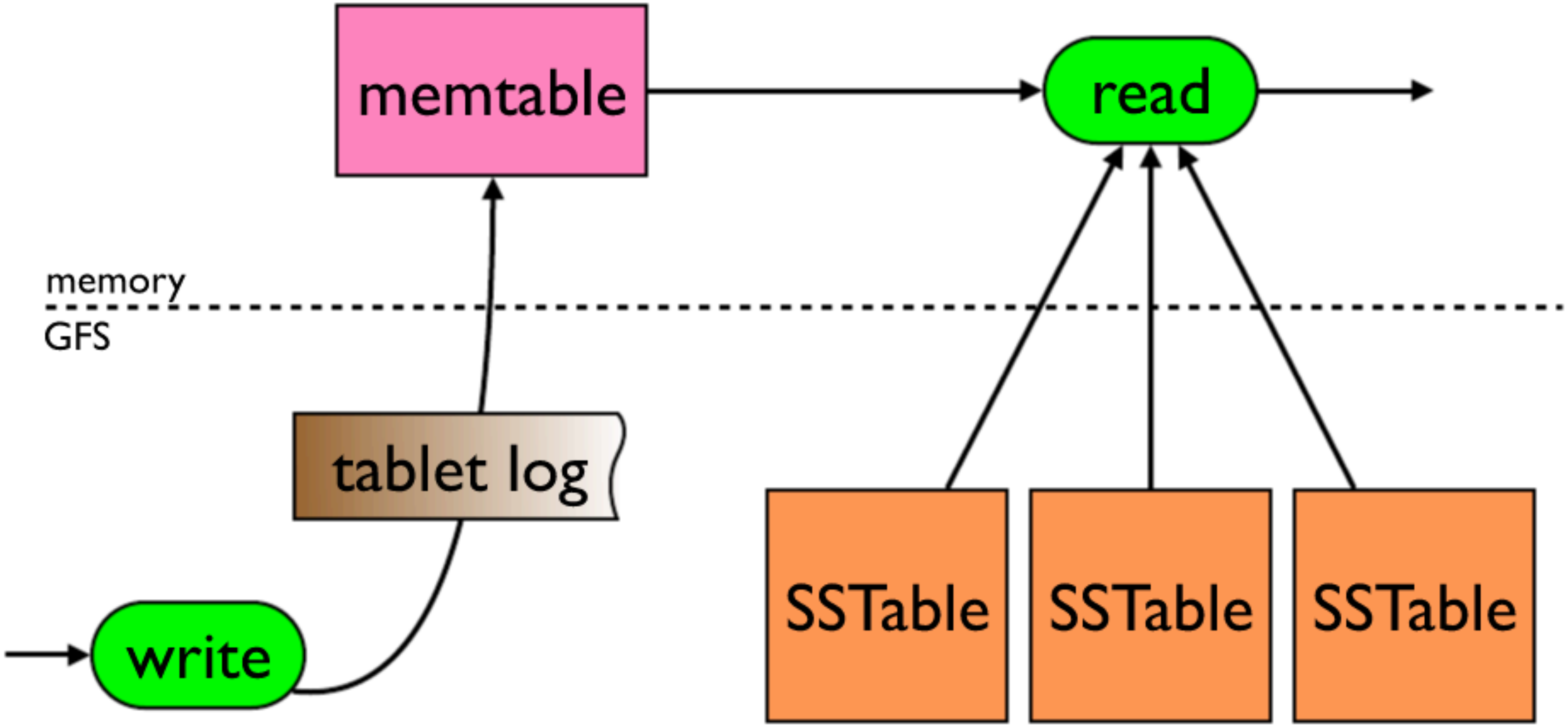
64KB block

64KB block

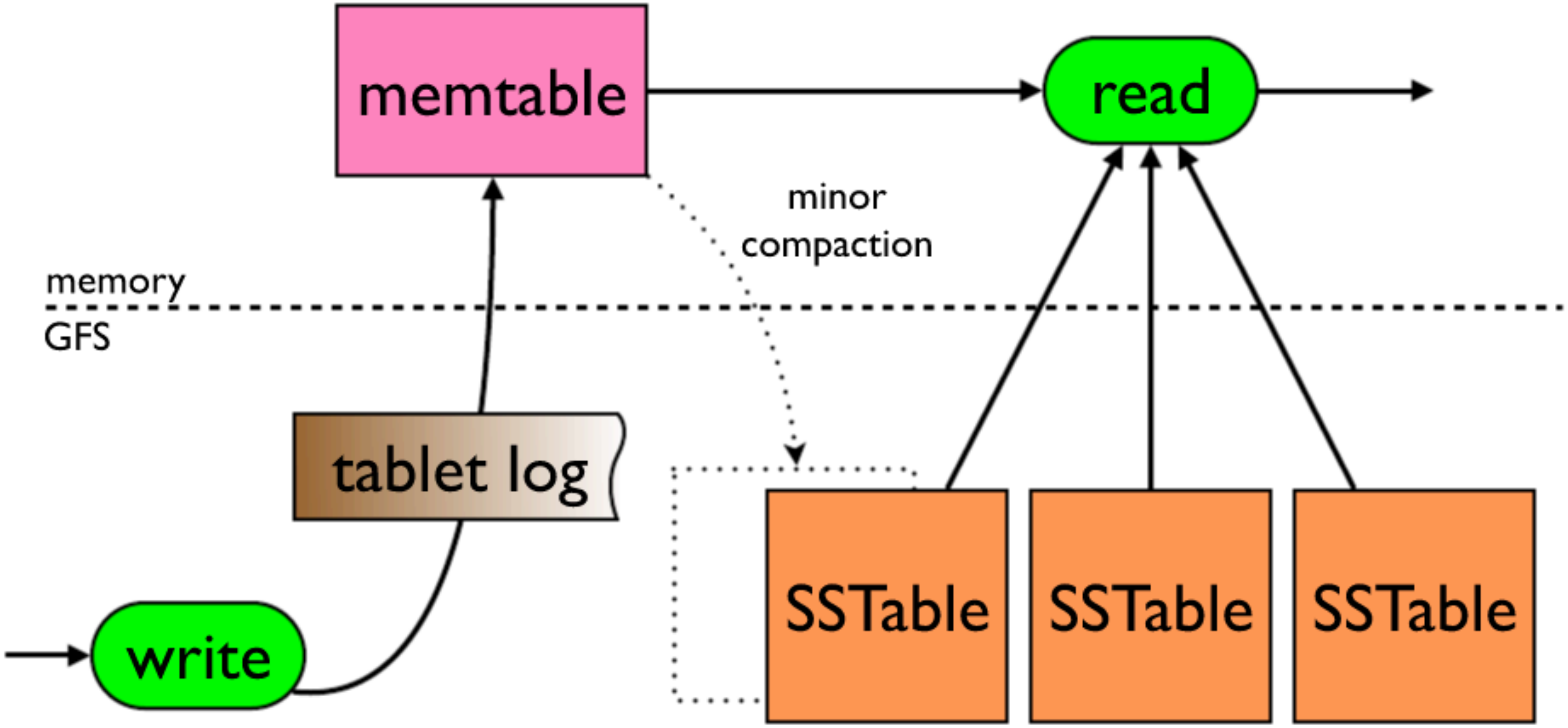
SSTable

lookup index

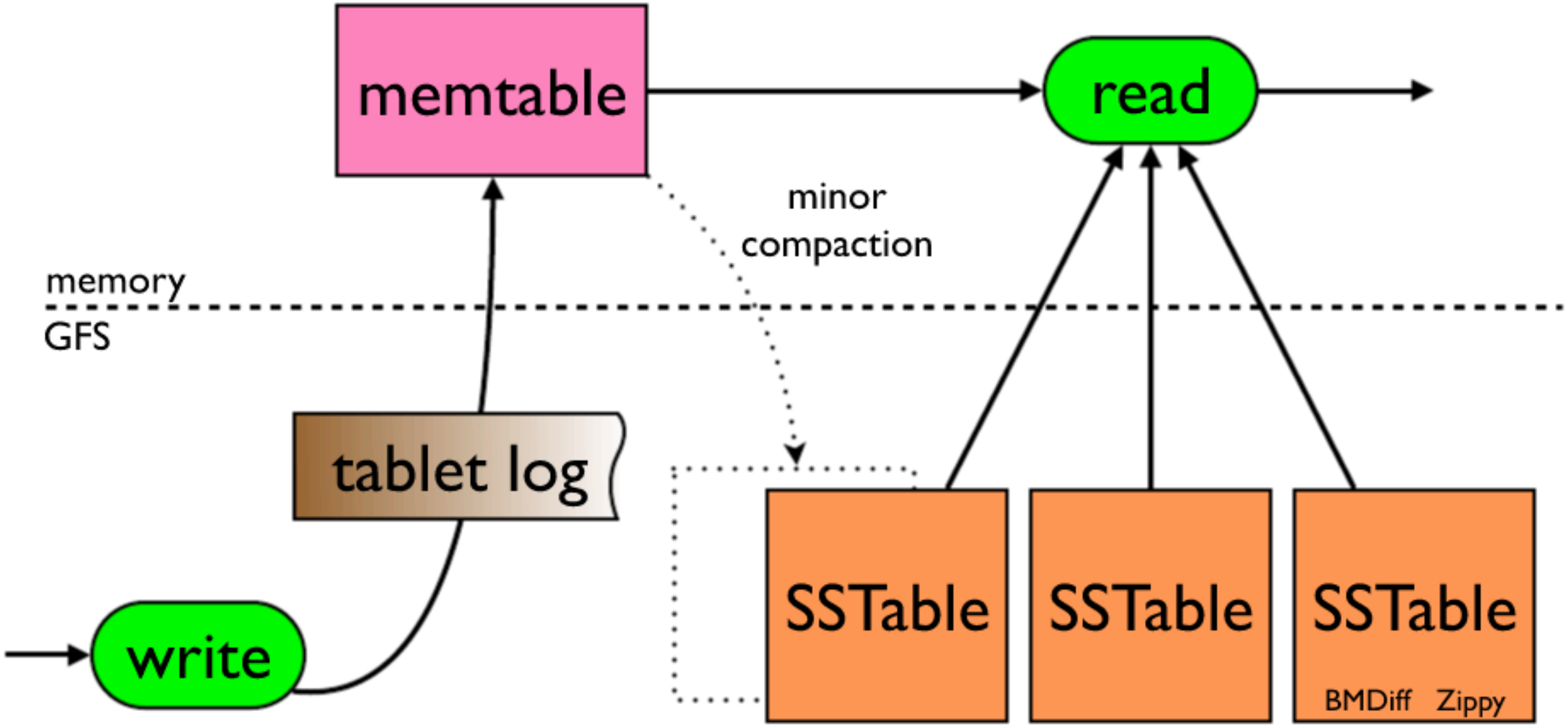
Google BigTable: I/O



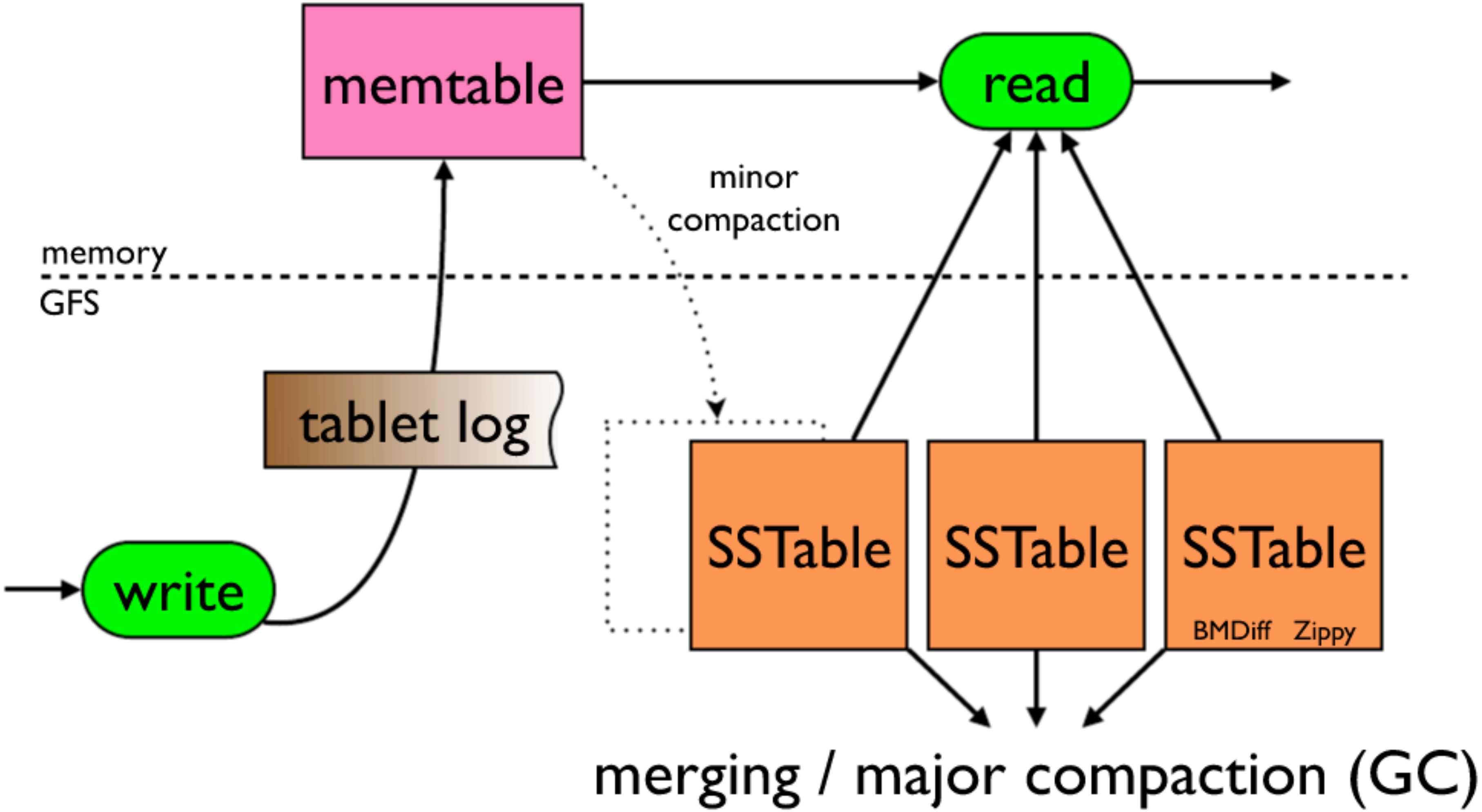
Google BigTable: I/O



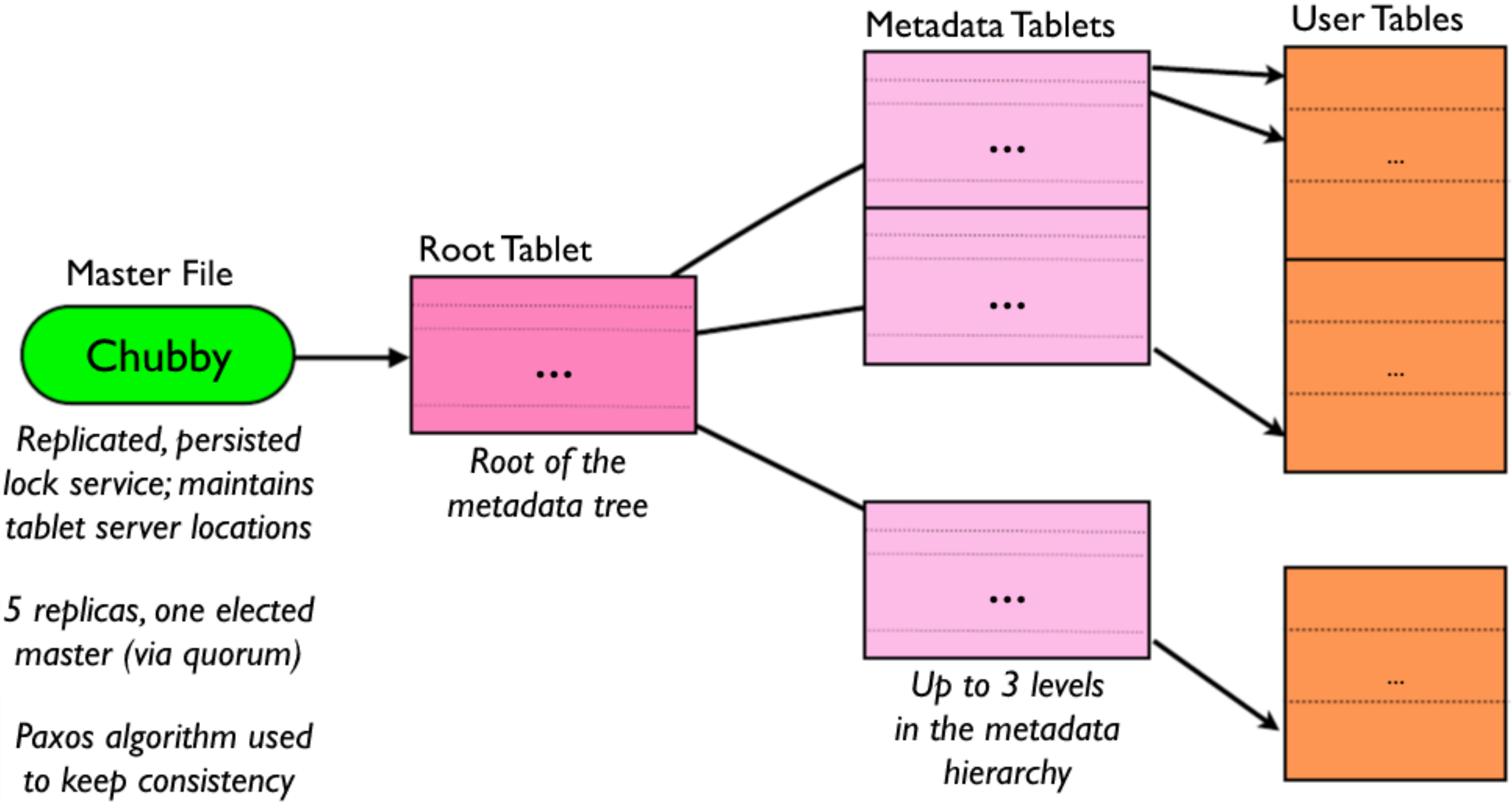
Google BigTable: I/O



Google BigTable: I/O



Google BigTable: Location Dereferencing



Master File

Chubby

Replicated, persisted lock service; maintains tablet server locations

5 replicas, one elected master (via quorum)

Paxos algorithm used to keep consistency

Root Tablet

...

Root of the metadata tree

Metadata Tablets

...

...

...

Up to 3 levels in the metadata hierarchy

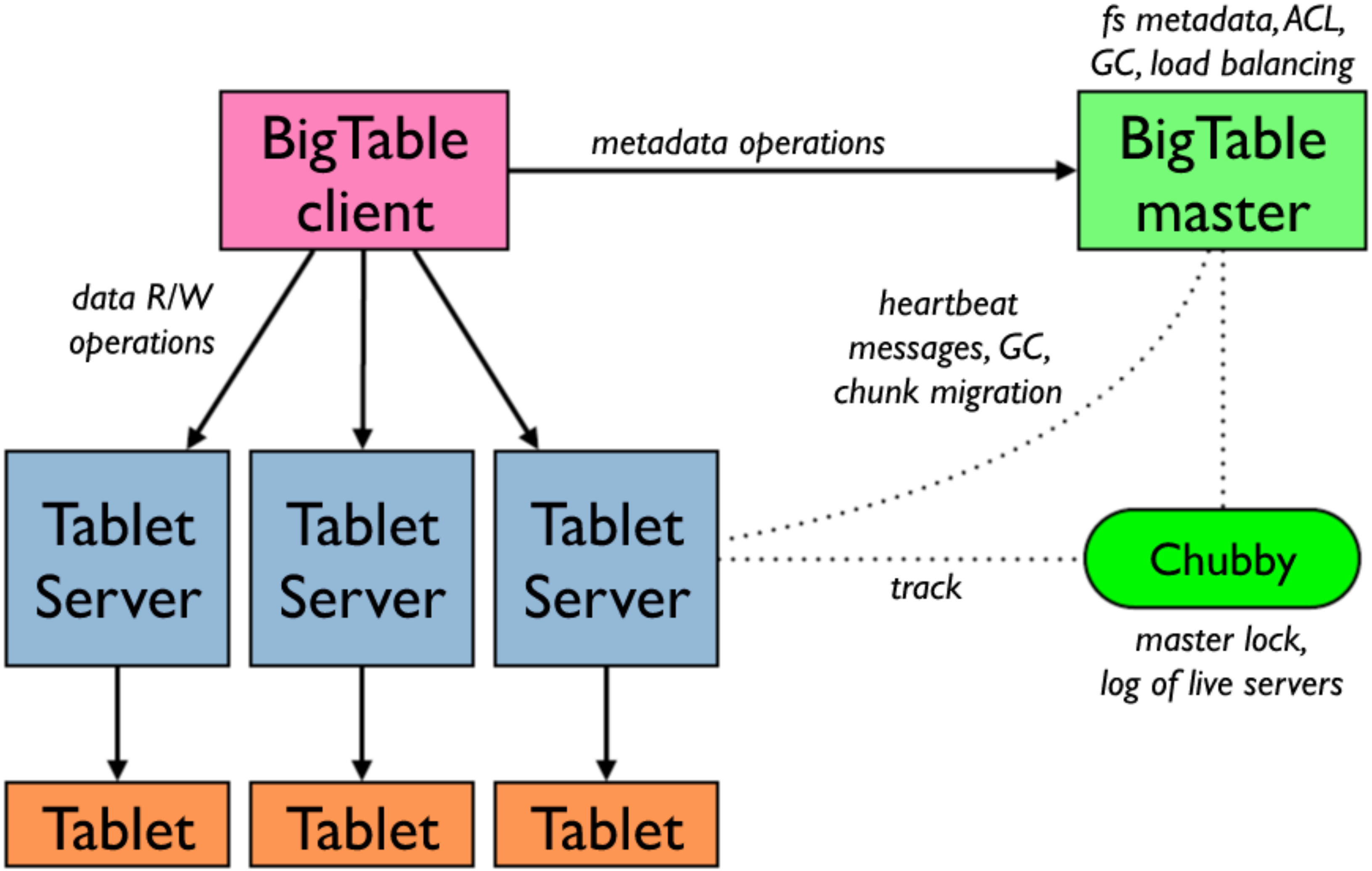
User Tables

...

...

...

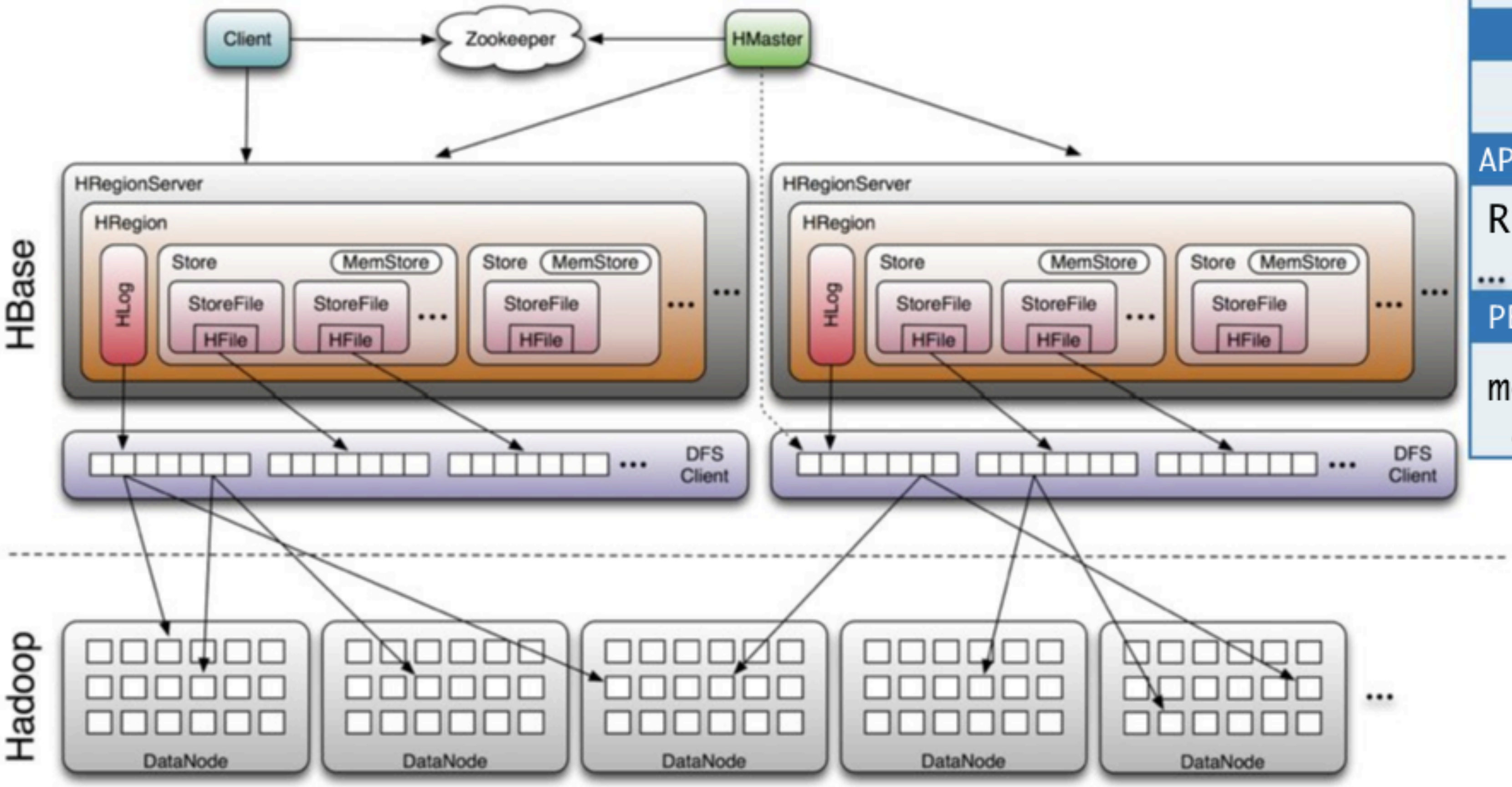
Google BigTable: Architecture



HBase

CP

OSS implementation of BigTable

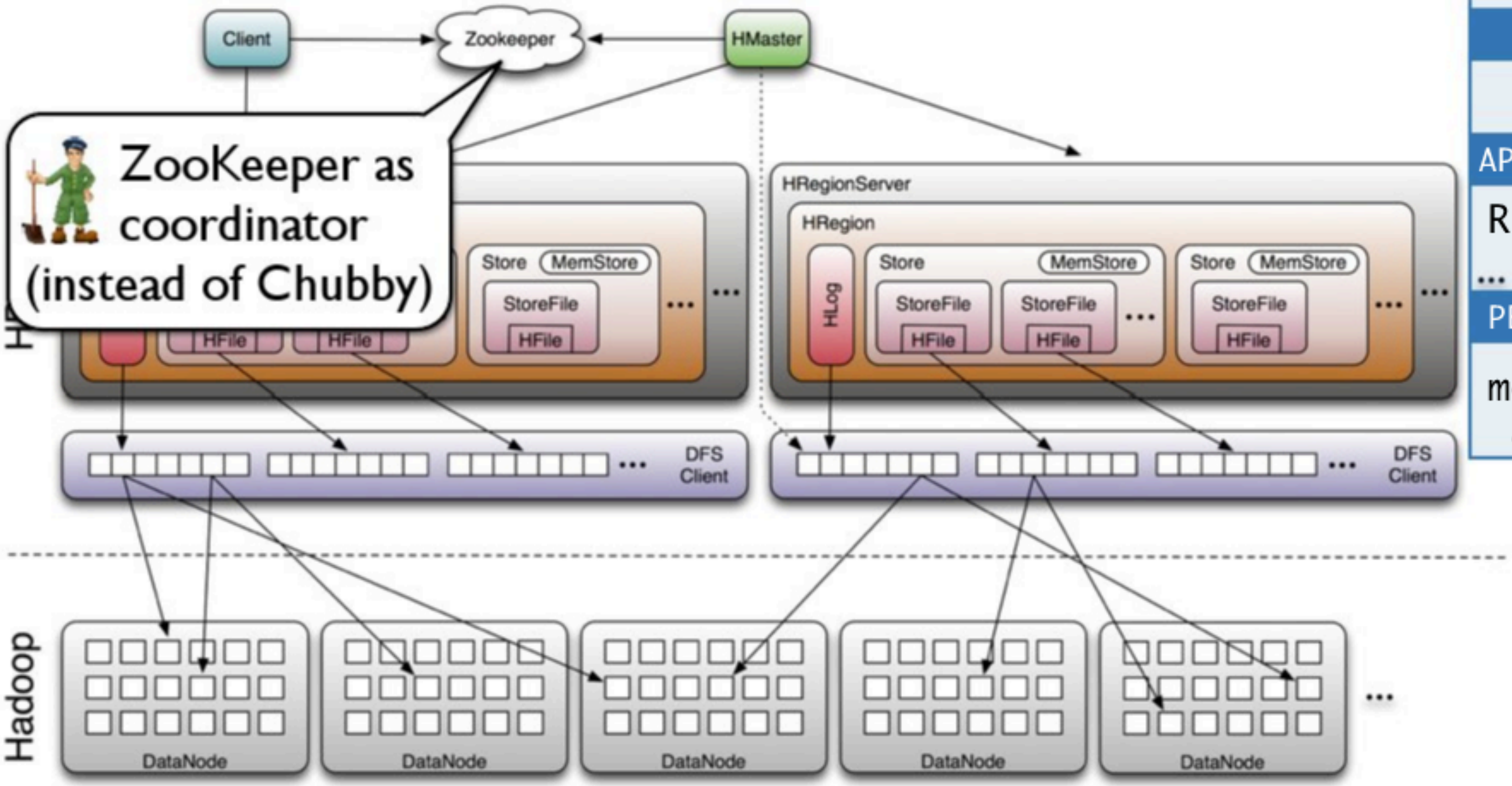


LICENSE
Apache 2
LANGUAGE
Java
API/PROTOCOL
REST HTTP ... Thrift
PERSISTENCE
memtable/ SSTable

HBase

CP

OSS implementation of BigTable



ZooKeeper as coordinator (instead of Chubby)

LICENSE

Apache 2

LANGUAGE

Java

API/PROTOCOL

REST HTTP
... Thrift

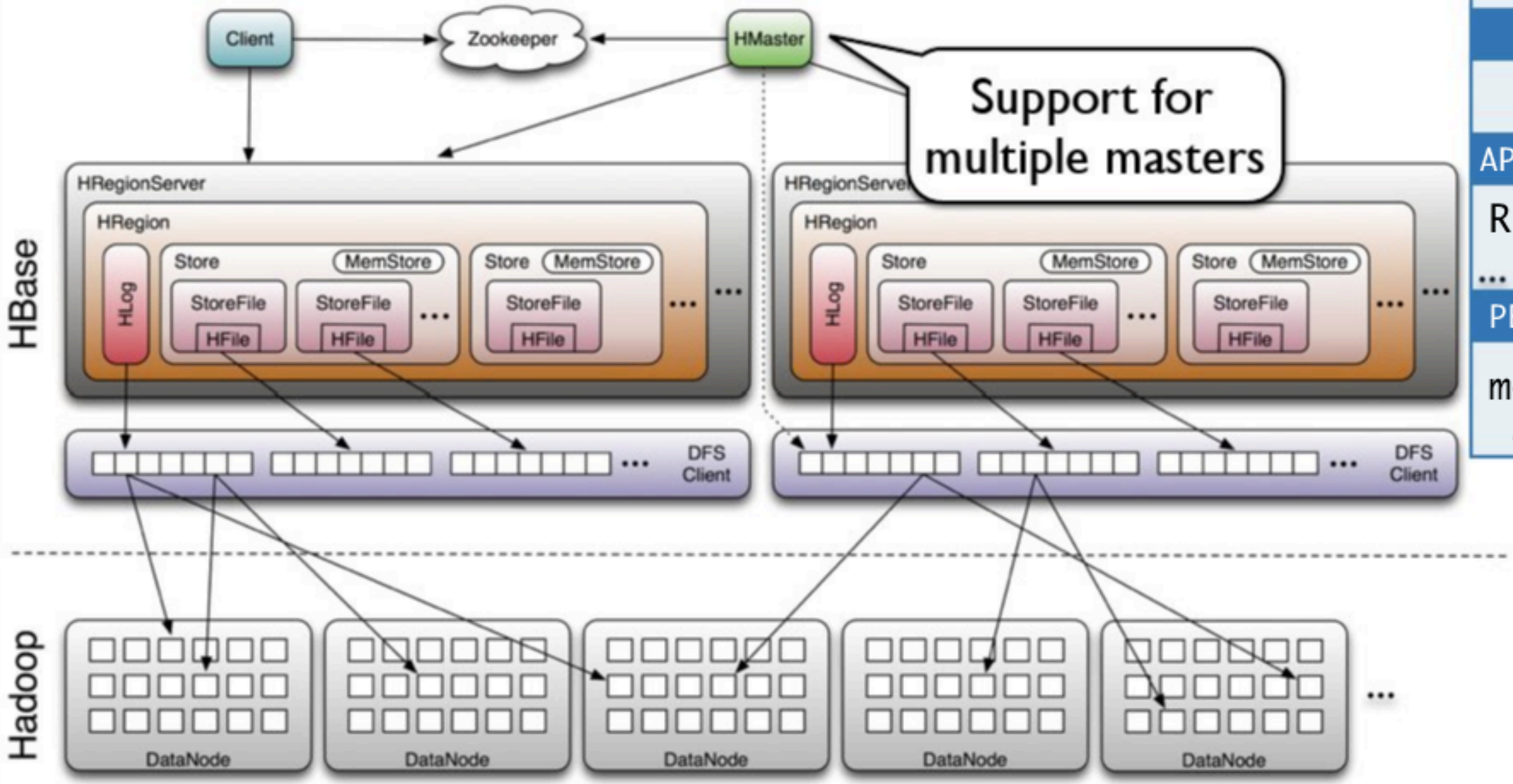
PERSISTENCE

memtable/
SSTable

HBase

CP

OSS implementation of BigTable



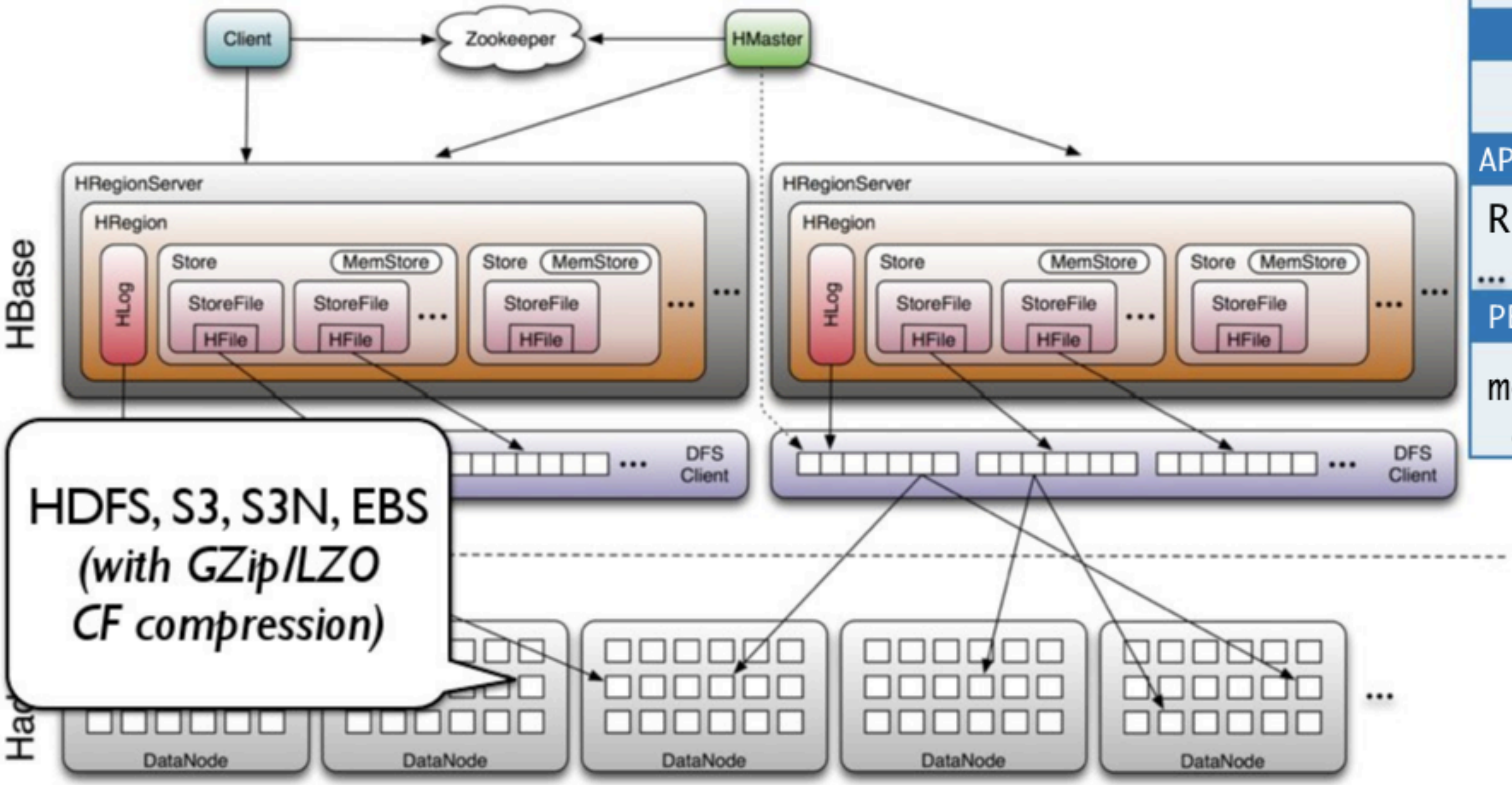
Support for multiple masters

LICENSE	Apache 2
LANGUAGE	Java
API/PROTOCOL	REST HTTP ... Thrift
PERSISTENCE	memtable/ SSTable

HBase

CP

OSS implementation of BigTable



HDFS, S3, S3N, EBS *(with GZip/LZO CF compression)*

LICENSE

Apache 2

LANGUAGE

Java

API/PROTOCOL

REST HTTP
... Thrift

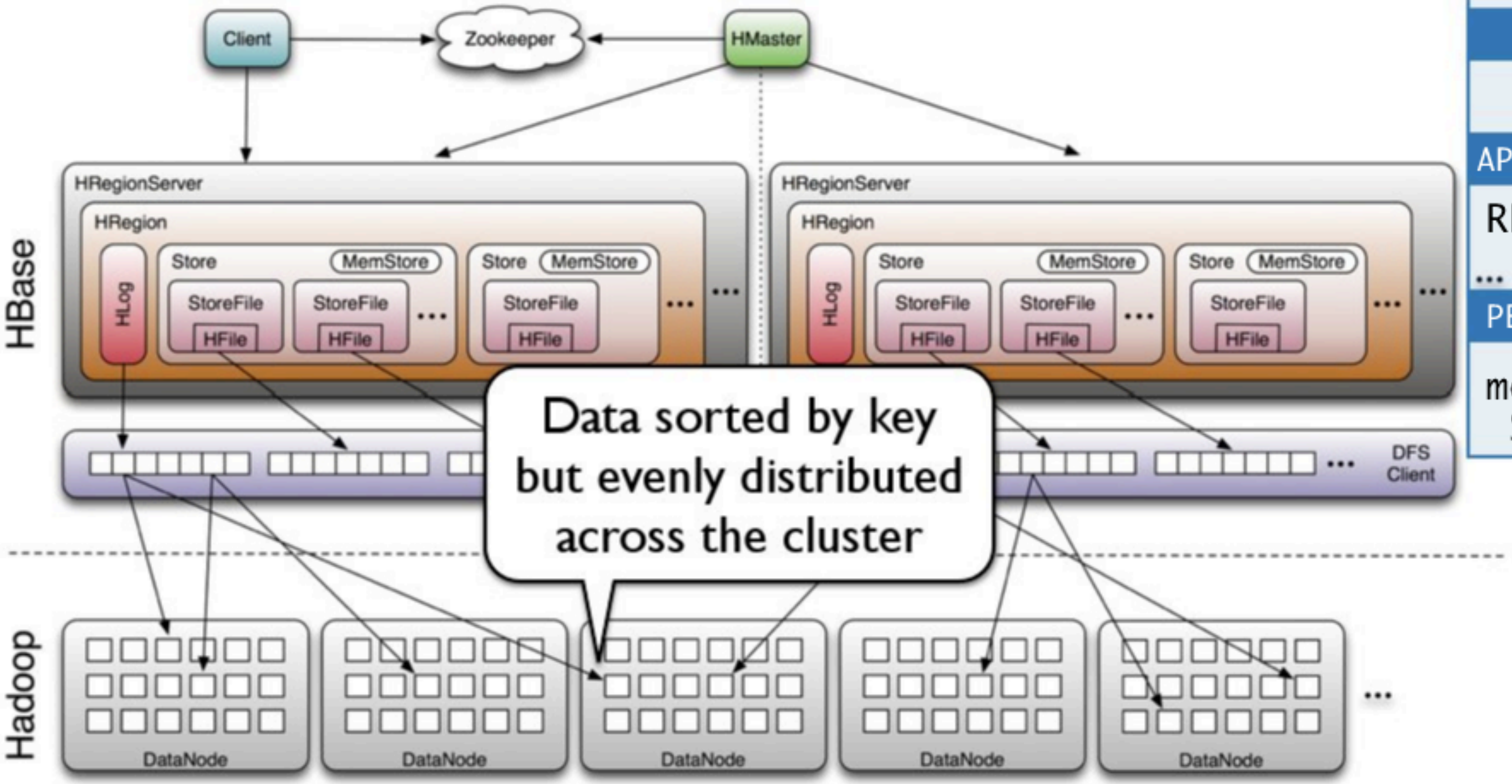
PERSISTENCE

memtable/
SSTable

HBase

CP

OSS implementation of BigTable



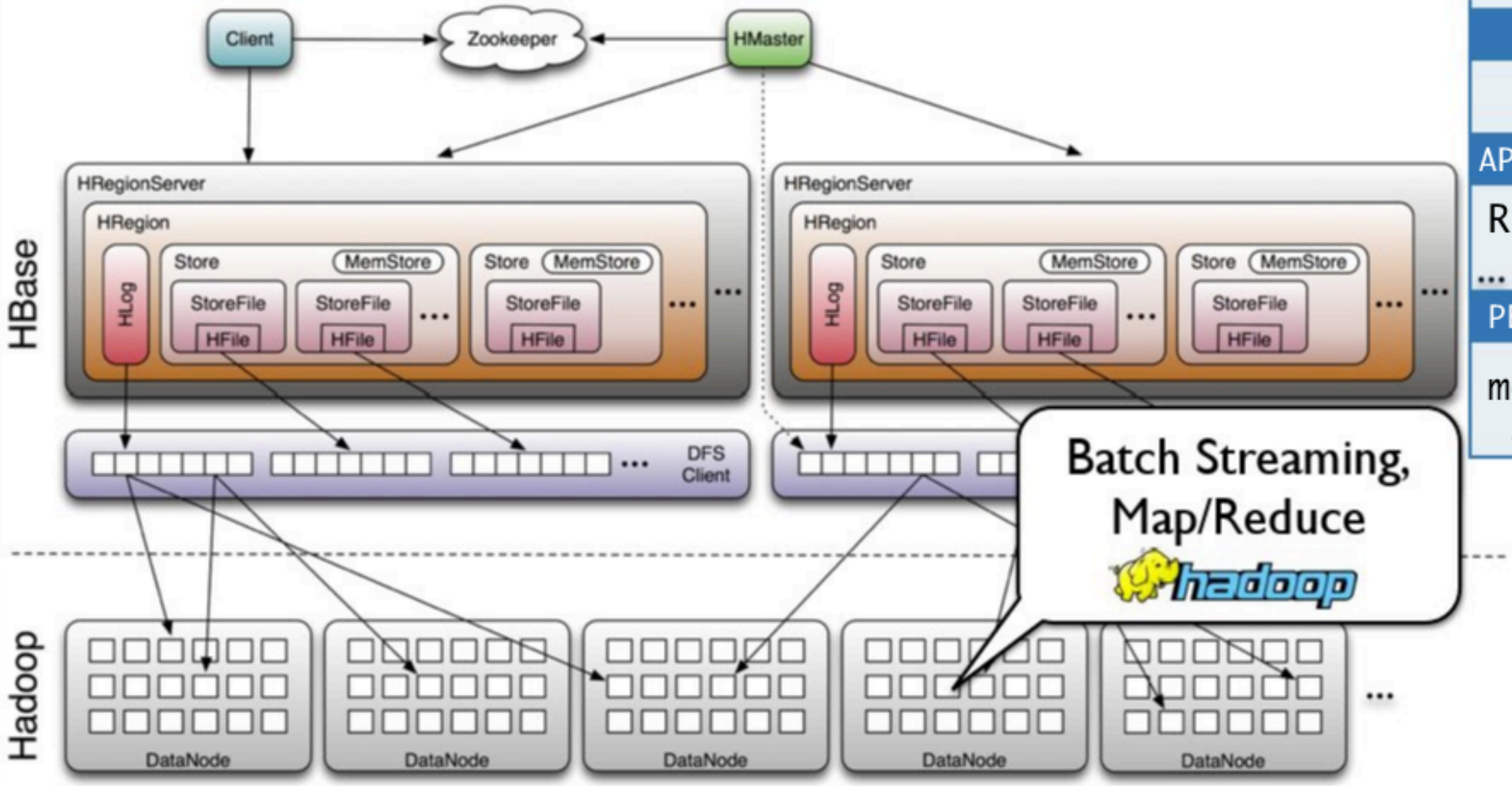
Data sorted by key but evenly distributed across the cluster

LICENSE
Apache 2
LANGUAGE
Java
API/PROTOCOL
REST HTTP ... Thrift
PERSISTENCE
memtable/ SSTable

HBase

CP

OSS implementation of BigTable



Batch Streaming, Map/Reduce

hadoop

LICENSE

Apache 2

LANGUAGE

Java

API/PROTOCOL

REST HTTP
... Thrift

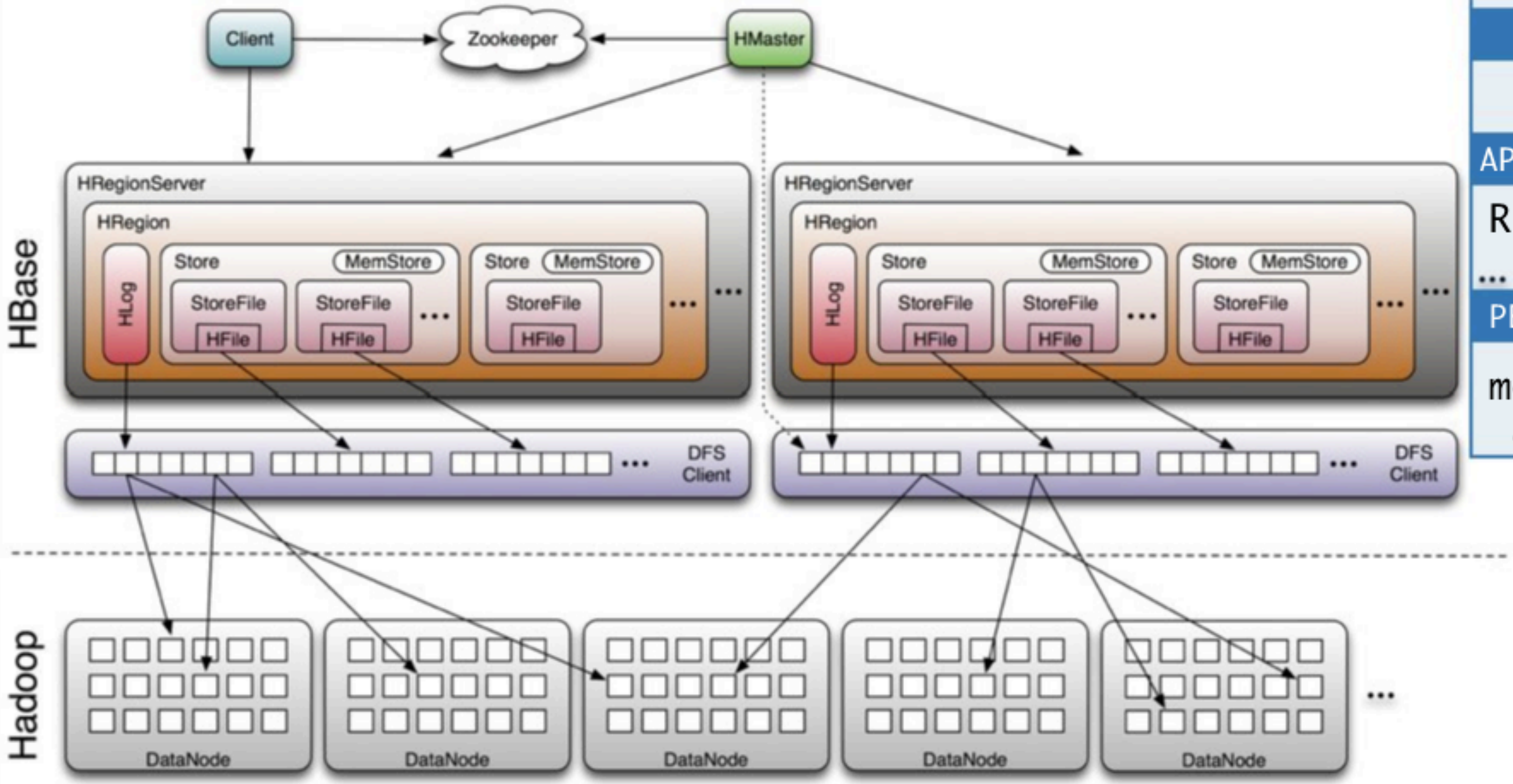
PERSISTENCE

memtable/
SSTable

HBase

CP

OSS implementation of BigTable



LICENSE

Apache 2

LANGUAGE

Java

API/PROTOCOL

REST HTTP
... Thrift

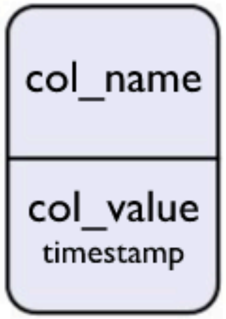
PERSISTENCE

memtable/
SSTable

Cassandra

AP

Data model of BigTable, infrastructure of Dynamo



Column

LICENSE

Apache 2

LANGUAGE

Java

PROTOCOL

Thrift
Avro

PERSISTENCE

memtable/
SSTable

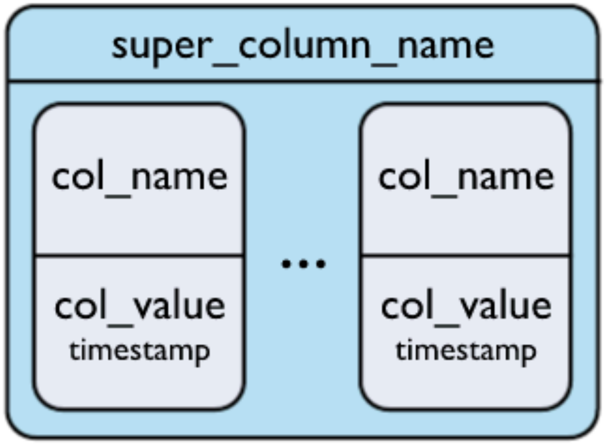
CONSISTENCY

Tunable
R/W/N

Cassandra

AP

Data model of BigTable, infrastructure of Dynamo


super_column_name

col_name

col_value
timestamp

...

col_name

col_value
timestamp

LICENSE

Apache 2

LANGUAGE

Java

PROTOCOL

Thrift
Avro

PERSISTENCE

memtable/
SSTable

CONSISTENCY

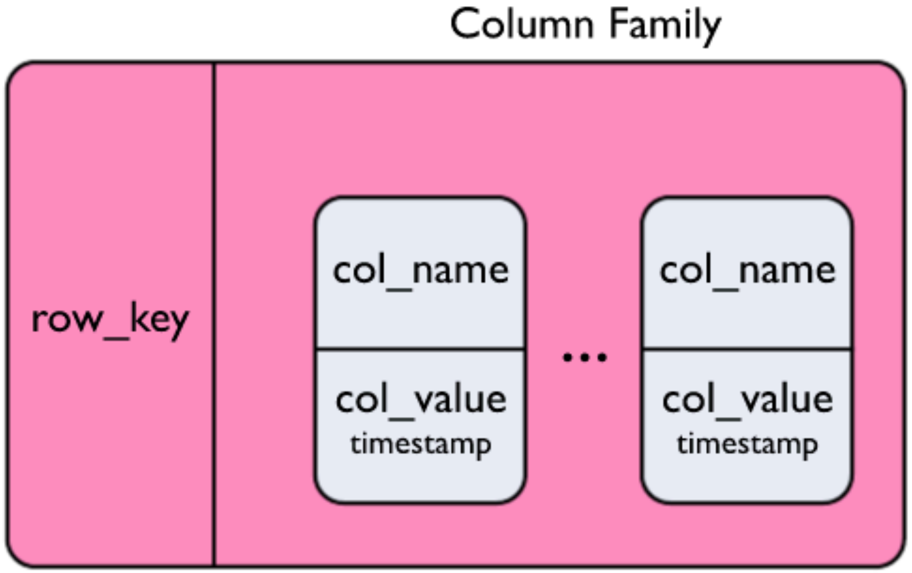
Tunable
R/W/N

Cassandra

facebook

AP

Data model of BigTable, infrastructure of Dynamo



LICENSE

Apache 2

LANGUAGE

Java

PROTOCOL

Thrift
Avro

PERSISTENCE

memtable/
SSTable

CONSISTENCY

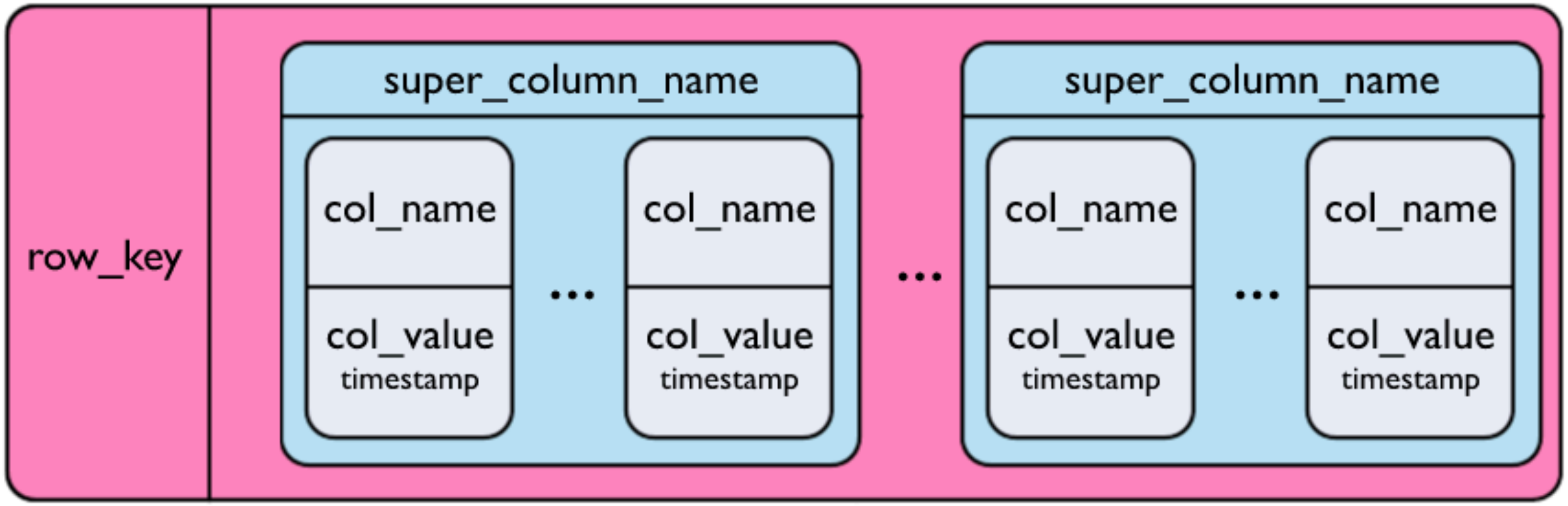
Tunable
R/W/N

Cassandra

AP

Data model of BigTable, infrastructure of Dynamo

Super Column Family



row_key

super_column_name

col_name

col_value
timestamp

...

col_name

col_value
timestamp

...

super_column_name

col_name

col_value
timestamp

...

col_name

col_value
timestamp

```
keyspace.get("column_family", key, ["super_column",] "column")
```

LICENSE

Apache 2

LANGUAGE

Java

PROTOCOL

Thrift
Avro

PERSISTENCE

memtable/
SSTable

CONSISTENCY

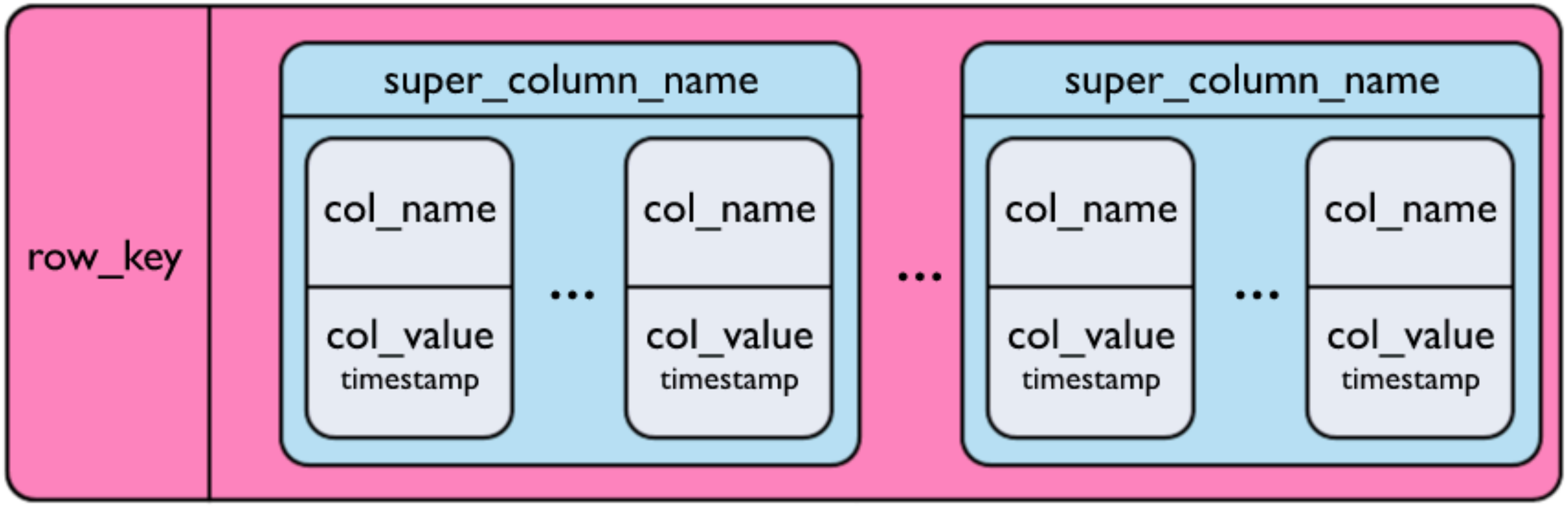
Tunable
R/W/N

Cassandra

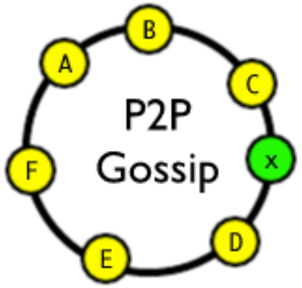
AP

Data model of BigTable, infrastructure of Dynamo

Super Column Family



```
keyspace.get("column_family", key, ["super_column",] "column")
```



LICENSE

Apache 2

LANGUAGE

Java

PROTOCOL

Thrift
Avro

PERSISTENCE

memtable/
SSTable

CONSISTENCY

Tunable
R/W/N

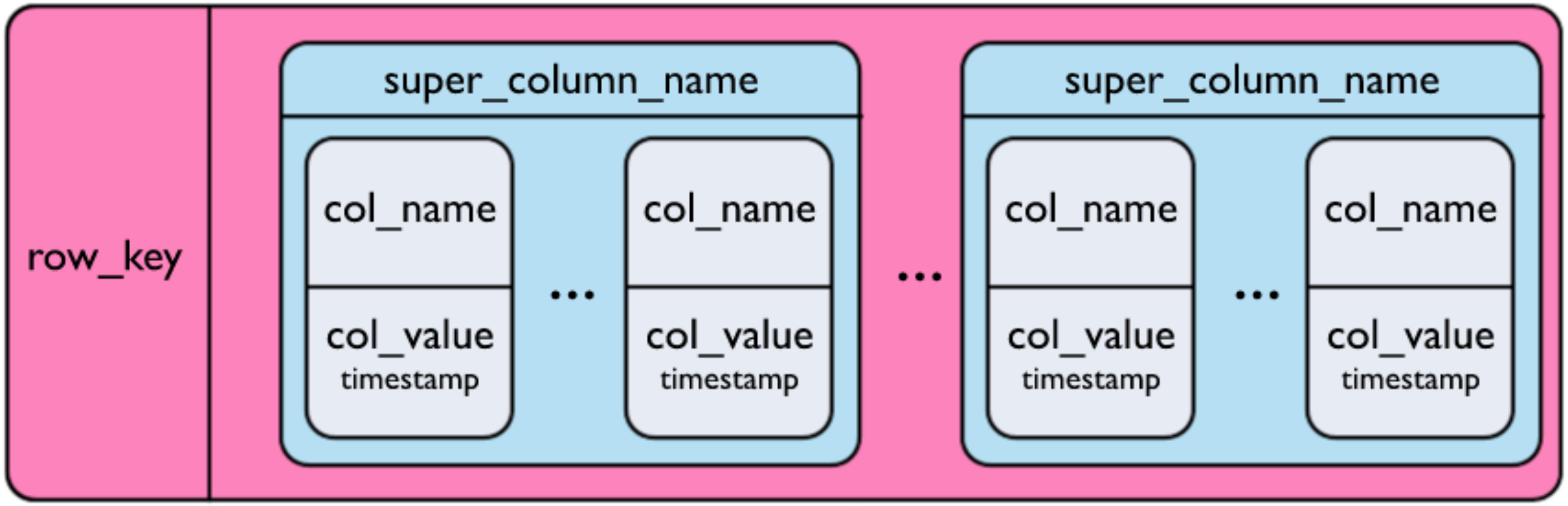
Cassandra

facebook

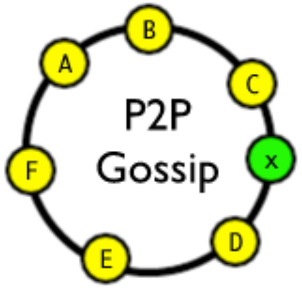
AP

Data model of BigTable, infrastructure of Dynamo

Super Column Family



```
keyspace.get("column_family", key, ["super_column",] "column")
```



ALL
ONE
QUORUM

LICENSE

Apache 2

LANGUAGE

Java

PROTOCOL

Thrift
Avro

PERSISTENCE

memtable/
SSTable

CONSISTENCY

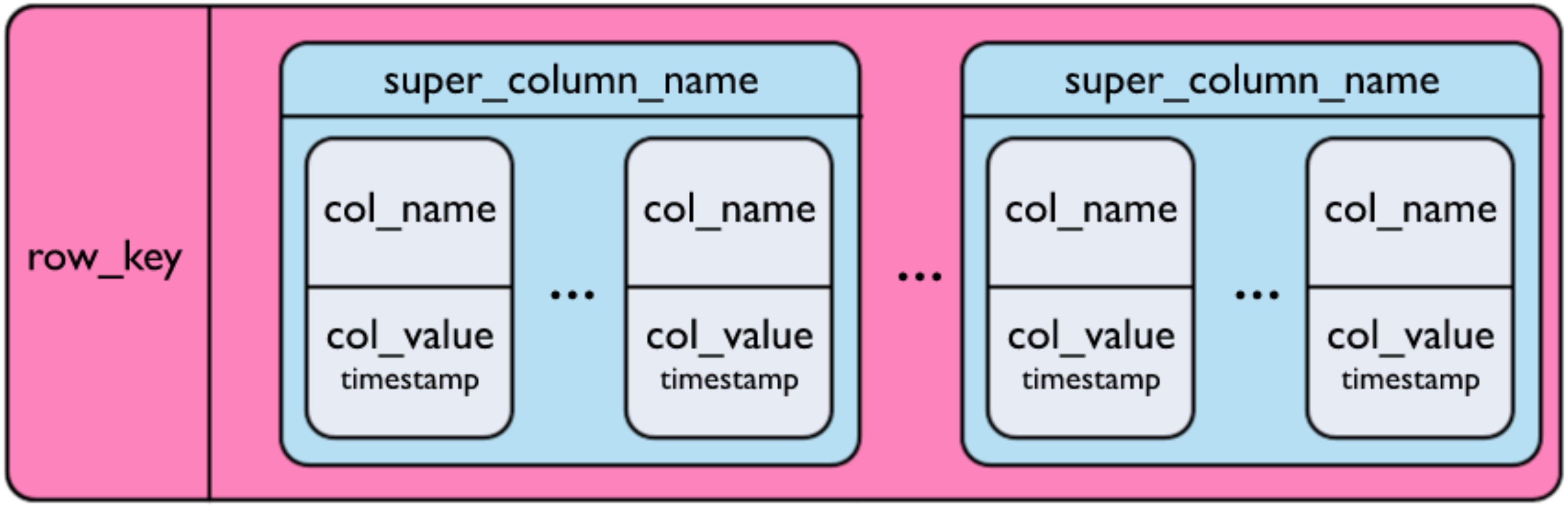
Tunable
R/W/N

Cassandra

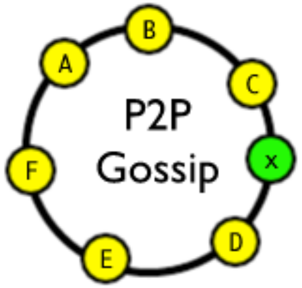
AP

Data model of BigTable, infrastructure of Dynamo

Super Column Family



```
keyspace.get("column_family", key, ["super_column",] "column")
```



ALL
ONE
QUORUM

RandomPartitioner (MD5)
OrderPreservingPartitioner
⇩
Range Scans, Fulltext Index (Solandra)

LICENSE
Apache 2

LANGUAGE
Java

PROTOCOL
Thrift
Avro

PERSISTENCE
memtable/
SSTable

CONSISTENCY
Tunable
R/W/N

3) Document DBs

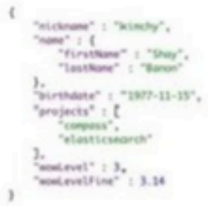
Lotus Notes

Data model: collection of K-V collections

CouchDB

CouchDB relax

AP



```
{
    "nickname" : "kimchy",
    "name" : {
        "firstName" : "Shay",
        "lastName" : "Banon"
    },
    "birthdate" : "1977-11-15",
    "projects" : [
        "compass",
        "elasticsearch"
    ],
    "wowLevel" : 3,
    "wowLevelFine" : 3.14
}
```

JSON docs

LICENSE

Apache 2

LANGUAGE

Erlang

API/PROTOCOL

REST/JSON

PERSISTENCE

Append-only B+Tree

CONCURRENCY

MVCC

CONSISTENCY

crash-only design

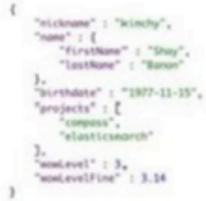
REPLICATION

multi-master

http://horicky.blogspot.com/2008/10/couchdb-implementation.html

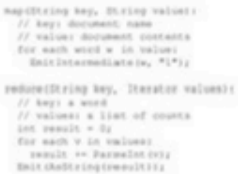
CouchDB

AP



```
{
    "nickname" : "kimchy",
    "name" : {
        "firstName" : "Shay",
        "lastName" : "Banon"
    },
    "birthdate" : "1977-11-15",
    "projects" : [
        "compass",
        "elasticsearch"
    ],
    "wowLevel" : 3,
    "wowLevelFine" : 3.14
}
```

JSON docs



```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

map-reduce "views"
(materialised resultset)

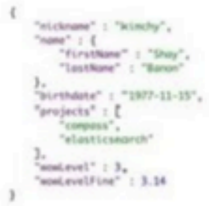
LICENSE	Apache 2
LANGUAGE	Erlang
API/PROTOCOL	REST/JSON
PERSISTENCE	Append-only B+Tree
CONCURRENCY	MVCC
CONSISTENCY	crash-only design
REPLICATION	multi-master

http://horicky.blogspot.com/2008/10/couchdb-implementation.html

CouchDB

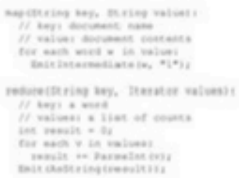
CouchDB relax

AP



```
{
    "nickname" : "kimchy",
    "name" : {
        "firstName" : "Shay",
        "lastName" : "Banon"
    },
    "birthdate" : "1977-11-15",
    "projects" : [
        "compass",
        "elasticsearch"
    ],
    "wowLevel" : 3,
    "wowLevelFine" : 3.14
}
```

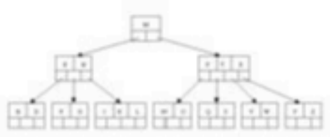
JSON docs



```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

map-reduce "views"
(materialised resultset)



Storage + View Indexes (B+Tree)
[by_id_index, by_seqnum_index]

LICENSE

Apache 2

LANGUAGE

Erlang

API/PROTOCOL

REST/JSON

PERSISTENCE

Append-only B+Tree

CONCURRENCY

MVCC

CONSISTENCY

crash-only design

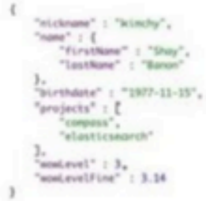
REPLICATION

multi-master

http://horicky.blogspot.com/2008/10/couchdb-implementation.html

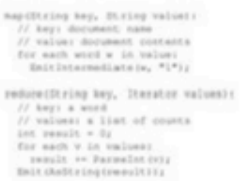
CouchDB

AP



```
{
    "nickname" : "kimchy",
    "name" : {
        "firstName" : "Shay",
        "lastName" : "Banon"
    },
    "birthdate" : "1977-11-15",
    "projects" : [
        "compass",
        "elasticsearch"
    ],
    "wowLevel" : 3,
    "wowLevelFine" : 3.14
}
```

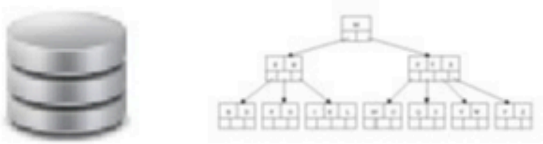
JSON docs



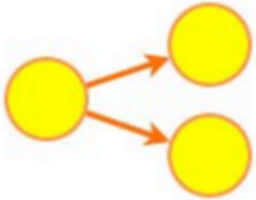
```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

map-reduce "views"
(materialised resultset)



Storage + View Indexes (B+Tree)
[by_id_index, by_seqnum_index]



Replication used as a way to scale *transactions* volume

LICENSE	Apache 2
LANGUAGE	Erlang
API/PROTOCOL	REST/JSON
PERSISTENCE	Append-only B+Tree
CONCURRENCY	MVCC
CONSISTENCY	crash-only design
REPLICATION	multi-master

http://horicky.blogspot.com/2008/10/couchdb-implementation.html

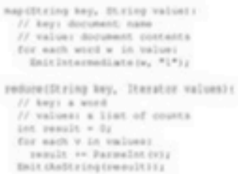
CouchDB

CouchDB relax

AP

```
{
    "nickname" : "kimchy",
    "name" : {
        "firstName" : "Shay",
        "lastName" : "Banon"
    },
    "birthdate" : "1977-11-15",
    "projects" : [
        "compass",
        "elasticsearch"
    ],
    "wowLevel" : 3,
    "wowLevelFine" : 3.14
}
```

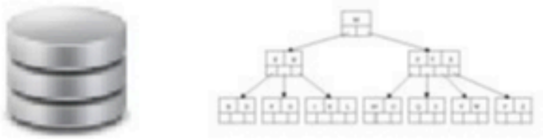
JSON docs



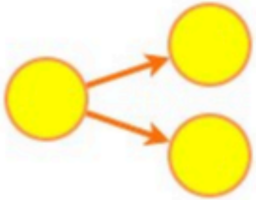
```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

map-reduce "views"
(materialised resultset)



Storage + View Indexes (B+Tree)
[by_id_index, by_seqnum_index]



Replication used
as a way to scale
transactions volume

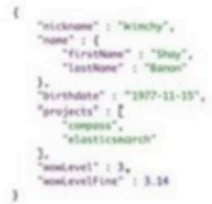
Conflict Resolution
at *application* level

LICENSE	Apache 2
LANGUAGE	Erlang
API/PROTOCOL	REST/JSON
PERSISTENCE	Append-only B+Tree
CONCURRENCY	MVCC
CONSISTENCY	crash-only design
REPLICATION	multi-master

http://horicky.blogspot.com/2008/10/couchdb-implementation.html

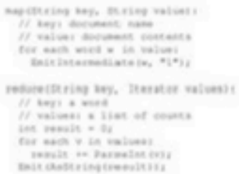
CouchDB

AP



```
{
    "nickname" : "kimchy",
    "name" : {
        "firstName" : "Shay",
        "lastName" : "Banon"
    },
    "birthdate" : "1977-11-15",
    "projects" : [
        "compass",
        "elasticsearch"
    ],
    "wowLevel" : 3,
    "wowLevelFine" : 3.14
}
```

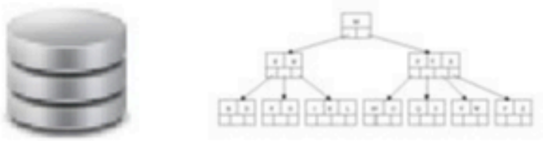
JSON docs



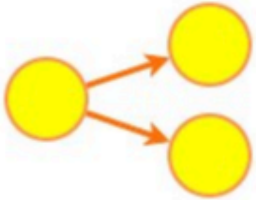
```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

map-reduce "views"
(materialised resultset)

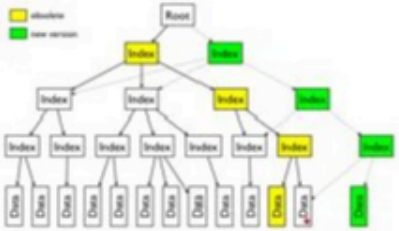


Storage + View Indexes (B+Tree)
[by_id_index, by_seqnum_index]



Replication used
as a way to scale
transactions volume

Conflict Resolution
at *application* level



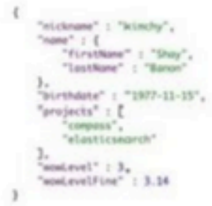
MVCC (copy-on-modify)
Volatile Versioning

LICENSE	Apache 2
LANGUAGE	Erlang
API/PROTOCOL	REST/JSON
PERSISTENCE	Append-only B+Tree
CONCURRENCY	MVCC
CONSISTENCY	crash-only design
REPLICATION	multi-master

http://horicky.blogspot.com/2008/10/couchdb-implementation.html

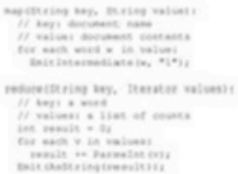
CouchDB

AP



```
{
    "nickname" : "kimchy",
    "name" : {
        "firstName" : "Shay",
        "lastName" : "Banon"
    },
    "birthdate" : "1977-11-15",
    "projects" : [
        "compass",
        "elasticsearch"
    ],
    "wowLevel" : 3,
    "wowLevelFine" : 3.14
}
```

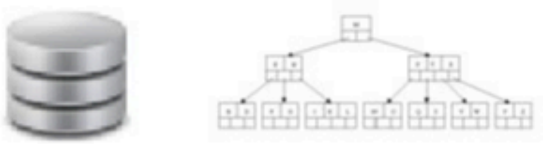
JSON docs



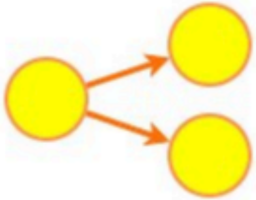
```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

map-reduce “views”
(materialised resultset)

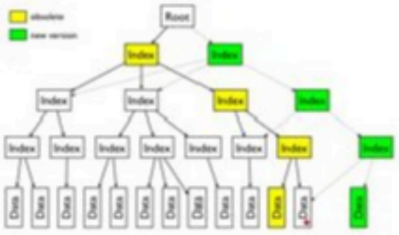


Storage + View Indexes (B+Tree)
[by_id_index, by_seqnum_index]



Replication used
as a way to scale
transactions volume

Conflict Resolution
at *application* level



MVCC (copy-on-modify)
Volatile Versioning

Online Compaction
(very primitive VACUUM)

LICENSE

Apache 2

LANGUAGE

Erlang

API/PROTOCOL

REST/JSON

PERSISTENCE

Append-only
B+Tree

CONCURRENCY

MVCC

CONSISTENCY

crash-only
design

REPLICATION

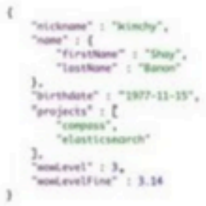
multi-master

http://horicky.blogspot.com/2008/10/couchdb-implementation.html

CouchDB

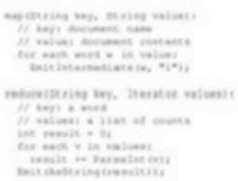
CouchDB relax

AP



```
{
    "nickname" : "kimchy",
    "name" : {
        "firstName" : "Shay",
        "lastName" : "Banon"
    },
    "birthdate" : "1977-11-15",
    "projects" : [
        "compass",
        "elasticsearch"
    ],
    "wowLevel" : 3,
    "wowLevelFine" : 3.14
}
```

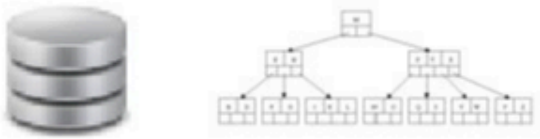
JSON docs



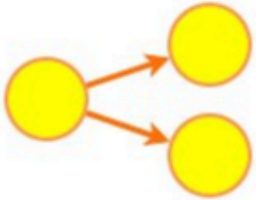
```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

map-reduce "views"
(materialised resultset)

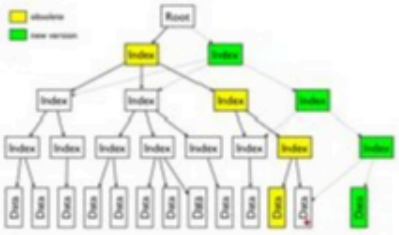


Storage + View Indexes (B+Tree)
[by_id_index, by_seqnum_index]



Replication used
as a way to scale
transactions volume

Conflict Resolution
at *application* level



MVCC (copy-on-modify)
Volatile Versioning

Online Compaction
(very primitive VACUUM)

Update validation /
Auth triggers

LICENSE
Apache 2

LANGUAGE
Erlang

API/PROTOCOL
REST/JSON

PERSISTENCE
Append-only
B+Tree

CONCURRENCY
MVCC

CONSISTENCY
crash-only
design

REPLICATION
multi-master

http://horicky.blogspot.com/2008/10/couchdb-implementation.html

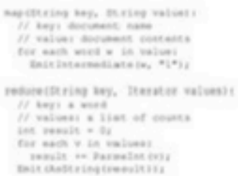
CouchDB

CouchDB relax

AP

```
{
    "nickname" : "kimchy",
    "name" : {
        "firstName" : "Shay",
        "lastName" : "Banon"
    },
    "birthdate" : "1977-11-15",
    "projects" : [
        "compass",
        "elasticsearch"
    ],
    "wowLevel" : 3,
    "wowLevelFine" : 3.14
}
```

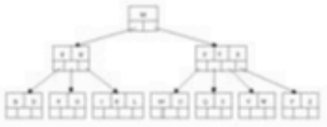
JSON docs



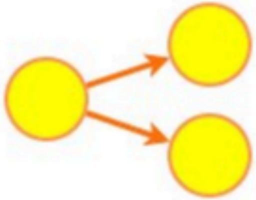
```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

map-reduce "views"
(materialised resultset)

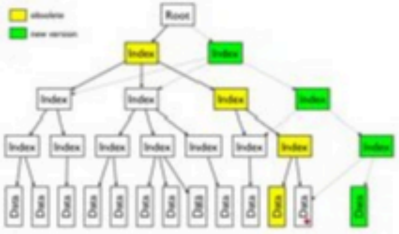


Storage + View Indexes (B+Tree)
[by_id_index, by_seqnum_index]



Replication used
as a way to scale
transactions volume

Conflict Resolution
at *application* level



MVCC (copy-on-modify)
Volatile Versioning

Online Compaction
(very primitive VACUUM)

Update validation /
Auth triggers

Delayed commits
(write performance)

LICENSE

Apache 2

LANGUAGE

Erlang

API/PROTOCOL

REST/JSON

PERSISTENCE

Append-only
B+Tree

CONCURRENCY

MVCC

CONSISTENCY

crash-only
design

REPLICATION

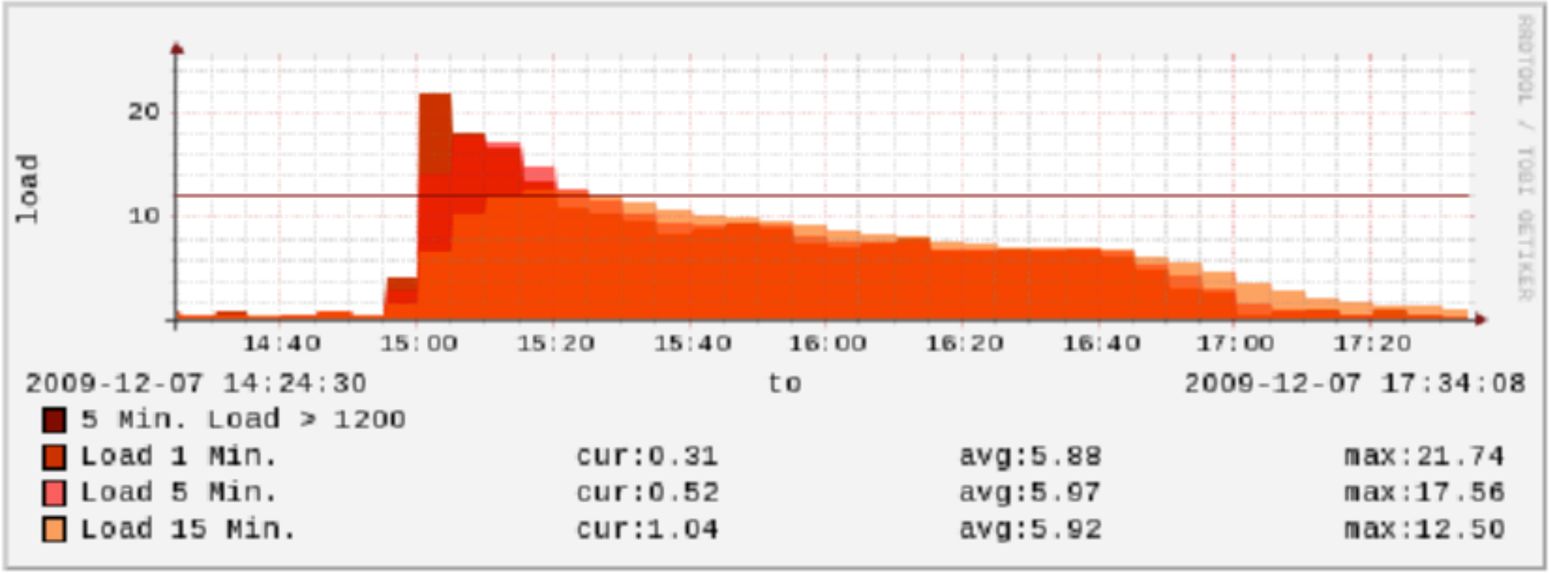
multi-master

http://horicky.blogspot.com/2008/10/couchdb-implementation.html

CouchDB

AP

MVCC consequences: compaction load, disk space



2009-12-07 14:24:30 to 2009-12-07 17:34:08

■ 5 Min. Load > 1200			
■ Load 1 Min.	cur:0.31	avg:5.88	max:21.74
■ Load 5 Min.	cur:0.52	avg:5.97	max:17.56
■ Load 15 Min.	cur:1.04	avg:5.92	max:12.50

http://enda.squarespace.com/tech/2009/12/8/couchdb-compaction-big-impacts.html

http://chesnok.com/talks/mvcc_couchcamp.pdf (PgSQL VACUUM)

LICENSE

Apache 2

LANGUAGE

Erlang

API/PROTOCOL

REST/JSON

PERSISTENCE

Append-only B+Tree

CONCURRENCY

MVCC

CONSISTENCY

crash-only design

REPLICATION

multi-master

http://horicky.blogspot.com/2008/10/couchdb-implementation.html

MongoDB

mongoDB

CP

bson_encode()
bson_decode()

BSON serialisation
(storage & transfer)

LICENSE	AGPLv3
LANGUAGE	C++
API/PROTOCOL	REST/BSON *
PERSISTENCE	B+Tree, Snapshots
CONCURRENCY	In-place updates
REPLICATION	master-slave replica sets

MongoDB

mongoDB

CP

bson_encode()
bson_decode()

BSON serialisation
(storage & transfer)

Auto-Sharding,
Master-Slave,
Auto-Failover

LICENSE	AGPLv3
LANGUAGE	C++
API/PROTOCOL	REST/BSON *
PERSISTENCE	B+Tree, Snapshots
CONCURRENCY	In-place updates
REPLICATION	master-slave replica sets

MongoDB

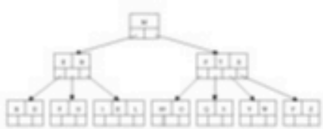
mongoDB

CP

```
bson_encode()
bson_decode()
```

BSON serialisation
(storage & transfer)

Auto-Sharding,
Master-Slave,
Auto-Failover



B-Tree Indexes
(on different cols too)

LICENSE	AGPLv3
LANGUAGE	C++
API/PROTOCOL	REST/BSON *
PERSISTENCE	B+Tree, Snapshots
CONCURRENCY	In-place updates
REPLICATION	master-slave replica sets

MongoDB

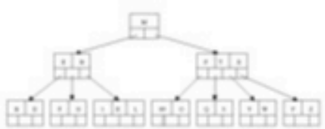
mongoDB

CP

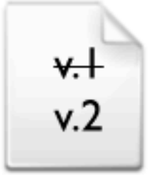
bson_encode()
bson_decode()

BSON serialisation
(storage & transfer)

Auto-Sharding,
Master-Slave,
Auto-Failover



B-Tree Indexes
(on different cols too)



Update in place
(no versioning, no append-only log)

LICENSE

AGPLv3

LANGUAGE

C++

API/PROTOCOL

REST/BSON
*

PERSISTENCE

B+Tree,
Snapshots

CONCURRENCY

In-place
updates

REPLICATION

master-slave
replica sets

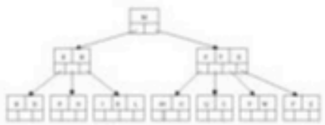
MongoDB

CP

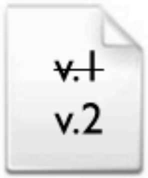
bson_encode()
bson_decode()

BSON serialisation
(storage & transfer)

Auto-Sharding,
Master-Slave,
Auto-Failover



B-Tree Indexes
(on different cols too)



Update in place
(no versioning, no append-only log)

Geo-Spatial Indexes

LICENSE
AGPLv3
LANGUAGE
C++
API/PROTOCOL
REST/BSON *
PERSISTENCE
B+Tree, Snapshots
CONCURRENCY
In-place updates
REPLICATION
master-slave replica sets

MongoDB

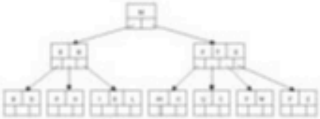
mongoDB

CP

bson_encode()
bson_decode()

BSON serialisation
(storage & transfer)

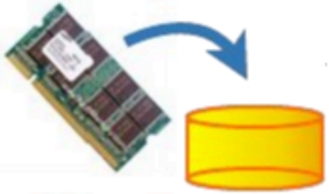
Auto-Sharding,
Master-Slave,
Auto-Failover



B-Tree Indexes
(on different cols too)

Update in place
(no versioning, no append-only log)

Geo-Spatial Indexes



Persistence via
Replication +
Snapshotting

LICENSE

AGPLv3

LANGUAGE

C++

API/PROTOCOL

REST/BSON
*

PERSISTENCE

B+Tree,
Snapshots

CONCURRENCY

In-place
updates

REPLICATION

master-slave
replica sets

MongoDB

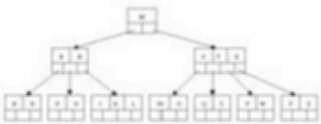
mongoDB

CP

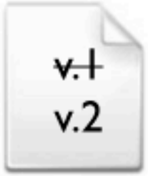
`bson_encode()`
`bson_decode()`

BSON serialisation
(storage & transfer)

Auto-Sharding,
Master-Slave,
Auto-Failover

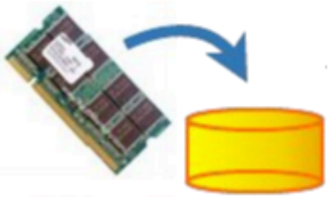


B-Tree Indexes
(on different cols too)



Update in place
(no versioning, no append-only log)

Geo-Spatial Indexes



Persistence via
Replication +
Snapshotting

`GROUP BY`

Map/Reduce
(well, aggregation)

LICENSE
AGPLv3
LANGUAGE
C++
API/PROTOCOL
REST/BSON *
PERSISTENCE
B+Tree, Snapshots
CONCURRENCY
In-place updates
REPLICATION
master-slave replica sets

MongoDB

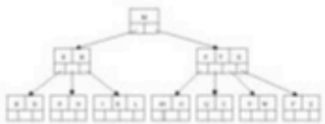
mongoDB

CP

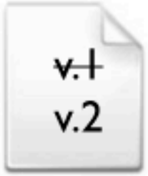
bson_encode()
bson_decode()

BSON serialisation
(storage & transfer)

Auto-Sharding,
Master-Slave,
Auto-Failover

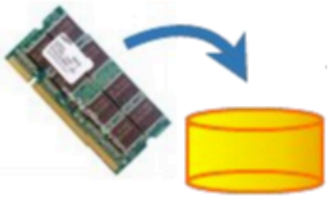


B-Tree Indexes
(on different cols too)



Update in place
(no versioning, no append-only log)

Geo-Spatial Indexes



Persistence via
Replication +
Snapshotting

GROUP BY

Map/Reduce
(well, aggregation)

No ACK on Updates
(or ensure N replicas)

LICENSE	AGPLv3
LANGUAGE	C++
API/PROTOCOL	REST/BSON *
PERSISTENCE	B+Tree, Snapshots
CONCURRENCY	In-place updates
REPLICATION	master-slave replica sets

MongoDB

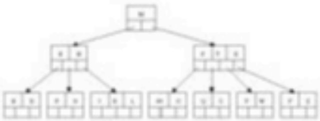
mongoDB

CP

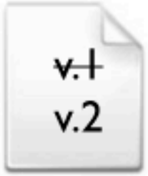
`bson_encode()`
`bson_decode()`

BSON serialisation
(storage & transfer)

Auto-Sharding,
Master-Slave,
Auto-Failover

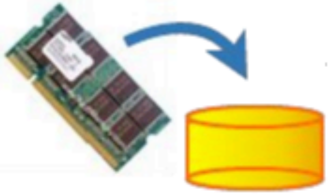


B-Tree Indexes
(on different cols too)



Update in place
(no versioning, no append-only log)

Geo-Spatial Indexes



Persistence via
Replication +
Snapshotting

`GROUP BY`

Map/Reduce
(well, aggregation)

No ACK on Updates
(or ensure N replicas)

New!
Improved!
`--dur` flag

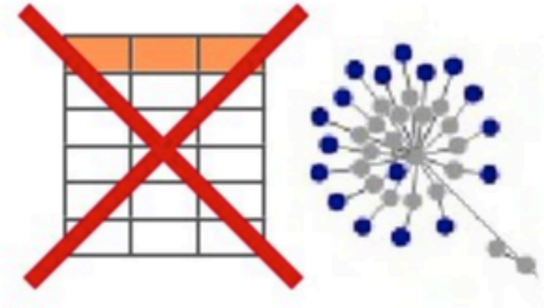
LICENSE	AGPLv3
LANGUAGE	C++
API/PROTOCOL	REST/BSON *
PERSISTENCE	B+Tree, Snapshots
CONCURRENCY	In-place updates
REPLICATION	master-slave replica sets

4) Graph databases

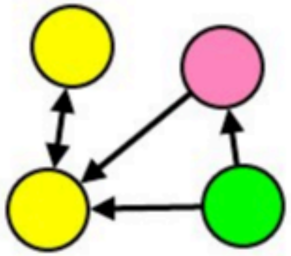
Graph Theory

Neo4j

Neo4j
the graph database



Graph Data Structure



Nodes,
Relationships,
Properties on both

Vertical Scalability
*(1000s times faster,
but not distributed)*

LICENSE

AGPLv3 / Commercial

LANGUAGE

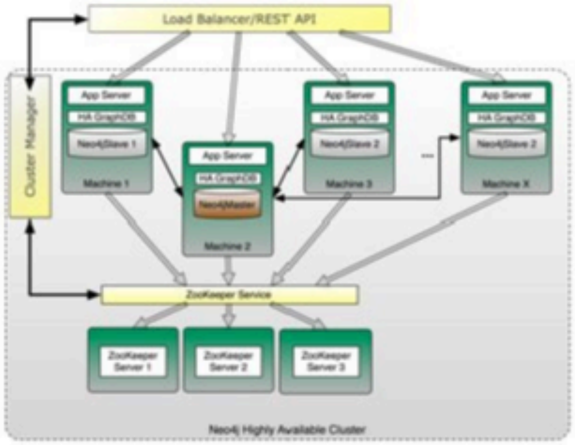
Java

API/PROTOCOL

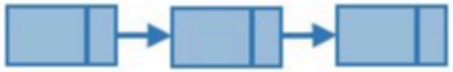
REST
Java
SPARQL

PERSISTENCE

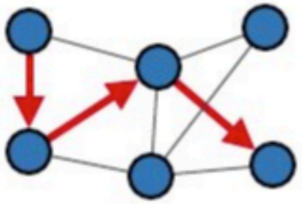
On-disk linked-list



HA cluster with ZooKeeper
(nodes = exact replicas)



Physical structure:
LinkedList stored on disk



High-performance
node traversal

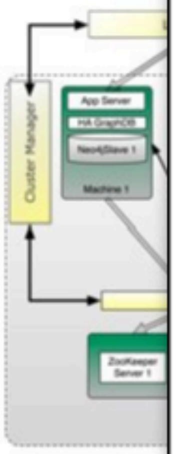
SPARQL

http://docs.neo4j.org/chunked/stable/ha-how.html

Neo4j

Neo4j
the graph database

Gra



HA clus

(node

```
NeoService neo = ... // factory

Transaction tx = neo.beginTx();

Node n1 = neo.createNode();
n1.setProperty(“name”, “John”);
n1.setProperty(“age”, 35);

Node n2 = neo.createNode();
n2.setProperty(“name”, “Mary”);
n2.setProperty(“age”, 29);
n2.setProperty(“job”, “engineer”);

n1.createRelationshipTo(n2, RelTypes.KNOWS);

tx.commit();
```

node traversal

LICENSE
AGPLv3 / Commercial
LANGUAGE
Java
API/PROTOCOL
REST Java SPARQL
PERSISTENCE
On-disk linked-list

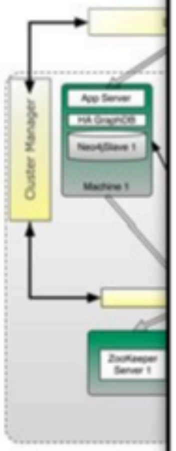
SPARQL

http://docs.neo4j.org/chunked/stable/ha-how.html

Neo4j

Neo4j
the graph database

Gra



HA clus

(node

```
Traverser friendTraverser = n1.traverse(
    Traverser.order.BREADTH_FIRST,
    StopEvaluator.END_OF_GRAPH,
    ReturnableEvaluator.ALL_BUT_START_NODE,
    RelTypes.KNOWS,
    Direction.OUTGOING
);
// Traverse the node space
System.out.println("John's friends: ");
for (Node friend : friendsTraverser) {
    System.out.printf("At depth %d => %s%n",
      friendTraverser.currentPosition().
        getDepth(),
      friendTraverser.getProperty("name")
    );
}
```

node traversal

LICENSE
AGPLv3 / Commercial
LANGUAGE
Java
API/PROTOCOL
REST Java SPARQL
PERSISTENCE
On-disk linked-list

SPARQL

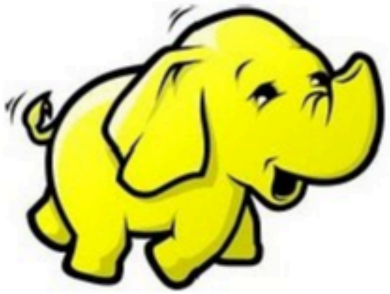
http://docs.neo4j.org/chunked/stable/ha-how.html

Final Considerations

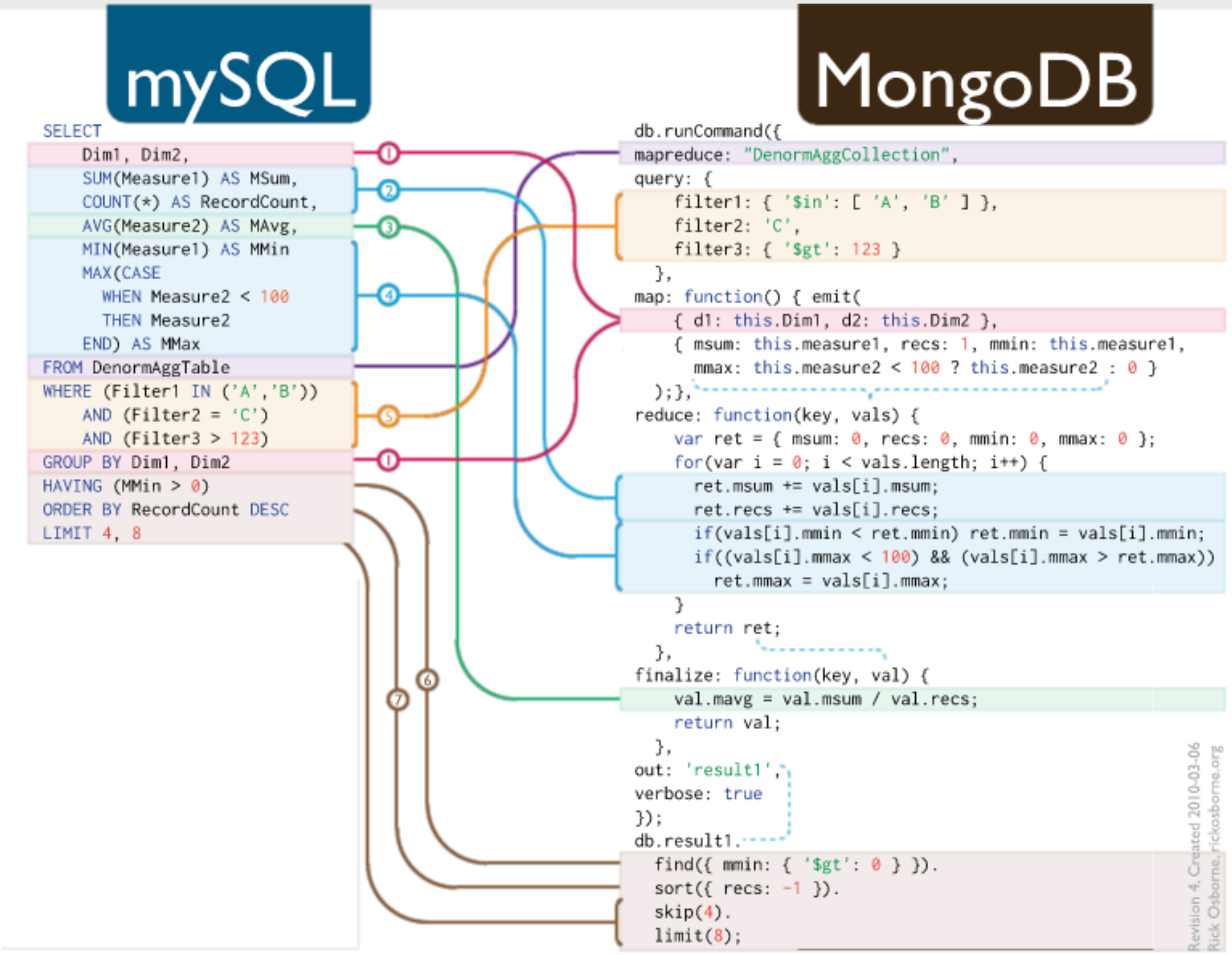
Query modes
Achievements and Problems

Query Modes: a new "SQL"?

Map/Reduce



SQL vs. Map/Reduce



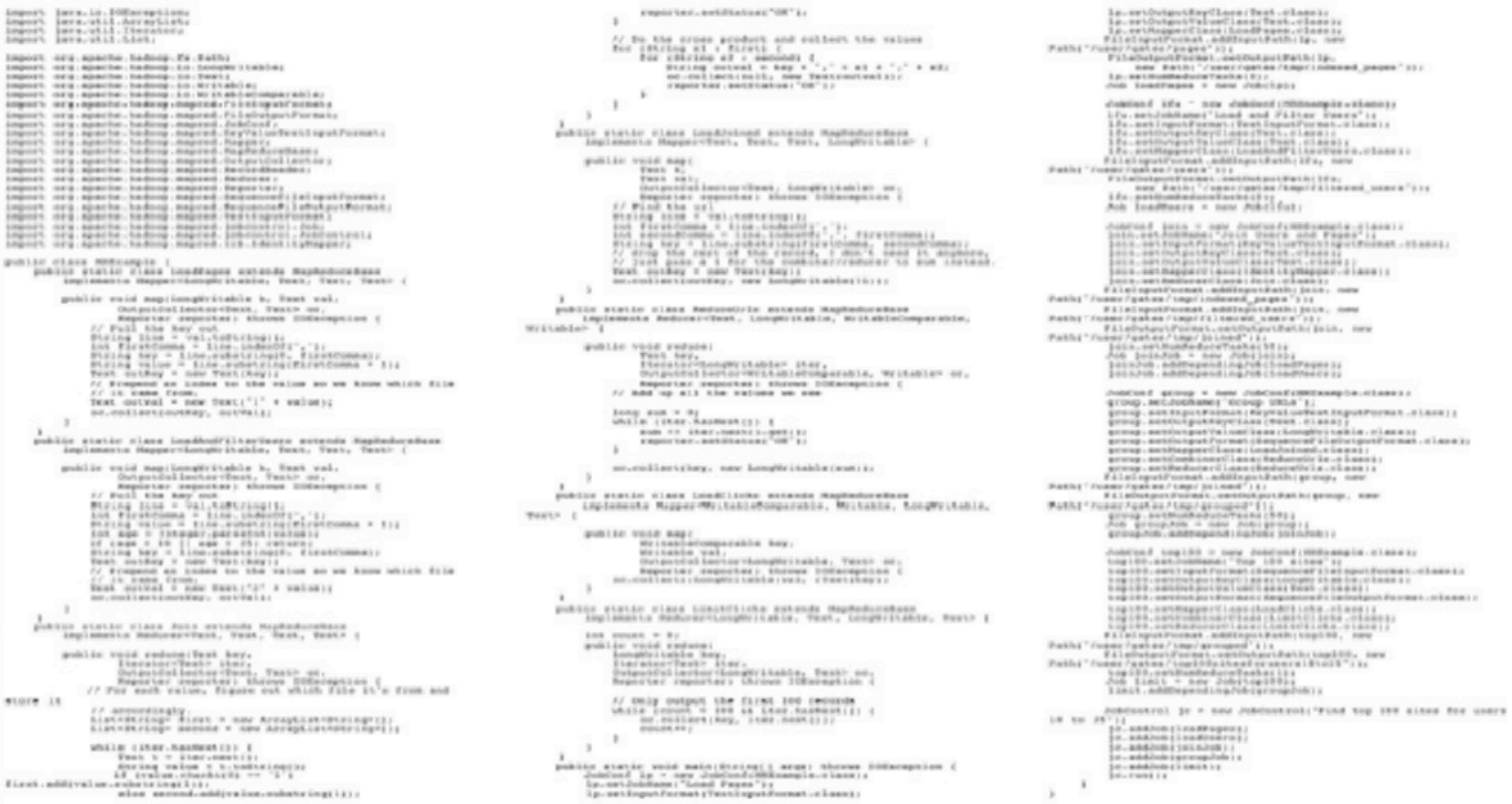
```
SELECT
    Dim1, Dim2,
    SUM(Measure1) AS MSum,
    COUNT(*) AS RecordCount,
    AVG(Measure2) AS MAvg,
    MIN(Measure1) AS MMin
    MAX(CASE
      WHEN Measure2 < 100
      THEN Measure2
    END) AS MMax
FROM DenormAggTable
WHERE (Filter1 IN ('A','B'))
    AND (Filter2 = 'C')
    AND (Filter3 > 123)
GROUP BY Dim1, Dim2
HAVING (MMin > 0)
ORDER BY RecordCount DESC
LIMIT 4, 8
```

```
db.runCommand({
mapreduce: "DenormAggCollection",
query: {
    filter1: { '$in': [ 'A', 'B' ] },
    filter2: 'C',
    filter3: { '$gt': 123 }
  },
map: function() { emit(
    { d1: this.Dim1, d2: this.Dim2 },
    { msum: this.measure1, recs: 1, mmin: this.measure1,
      mmax: this.measure2 < 100 ? this.measure2 : 0 }
  );},
reduce: function(key, vals) {
    var ret = { msum: 0, recs: 0, mmin: 0, mmax: 0 };
    for(var i = 0; i < vals.length; i++) {
      ret.msum += vals[i].msum;
      ret.recs += vals[i].recs;
      if(vals[i].mmin < ret.mmin) ret.mmin = vals[i].mmin;
      if((vals[i].mmax < 100) && (vals[i].mmax > ret.mmax))
        ret.mmax = vals[i].mmax;
    }
    return ret;
  },
finalize: function(key, val) {
    val.mavg = val.msum / val.recs;
    return val;
  },
out: 'result1',
verbose: true
});
db.result1.
  find({ mmin: { '$gt': 0 } }).
  sort({ recs: -1 }).
  skip(4).
  limit(8);
```

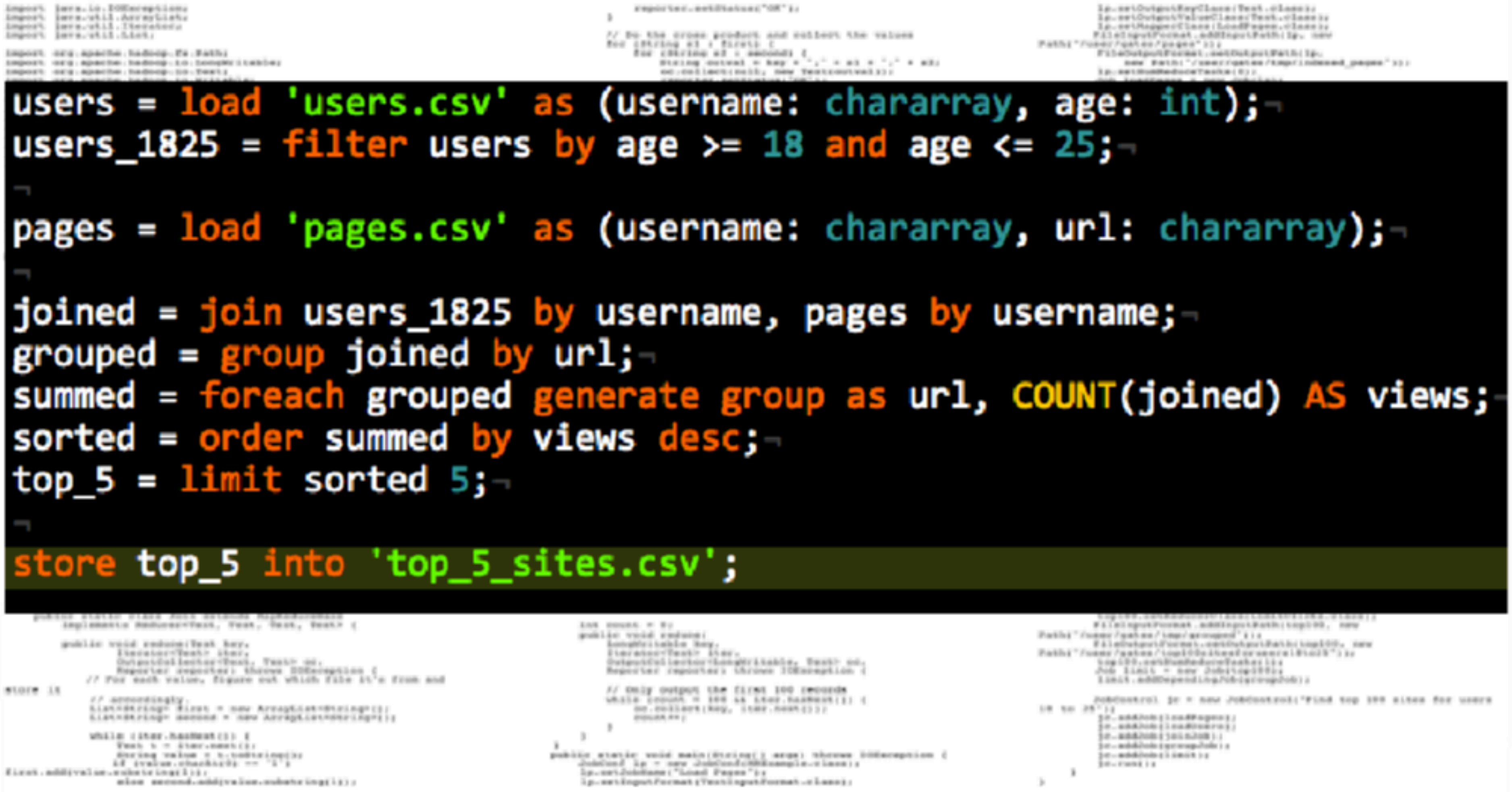
Revision 4, Created 2010-03-06
Rick Osborne, rickosborne.org

http://rickosborne.org/download/SQL-to-MongoDB.pdf

Pig vs. Map/Reduce



Pig vs. Map/Reduce



```
users = load 'users.csv' as (username: chararray, age: int);
users_1825 = filter users by age >= 18 and age <= 25;

pages = load 'pages.csv' as (username: chararray, url: chararray);

joined = join users_1825 by username, pages by username;
grouped = group joined by url;
summed = foreach grouped generate group as url, COUNT(joined) AS views;
sorted = order summed by views desc;
top_5 = limit sorted 5;

store top_5 into 'top_5_sites.csv';
```

Data model, Relations and Consistency

A step backwards?

Data model, Relations and Consistency

A step backwards?

Scalability, availability and resilience
come at a cost

Big Data

collect

store

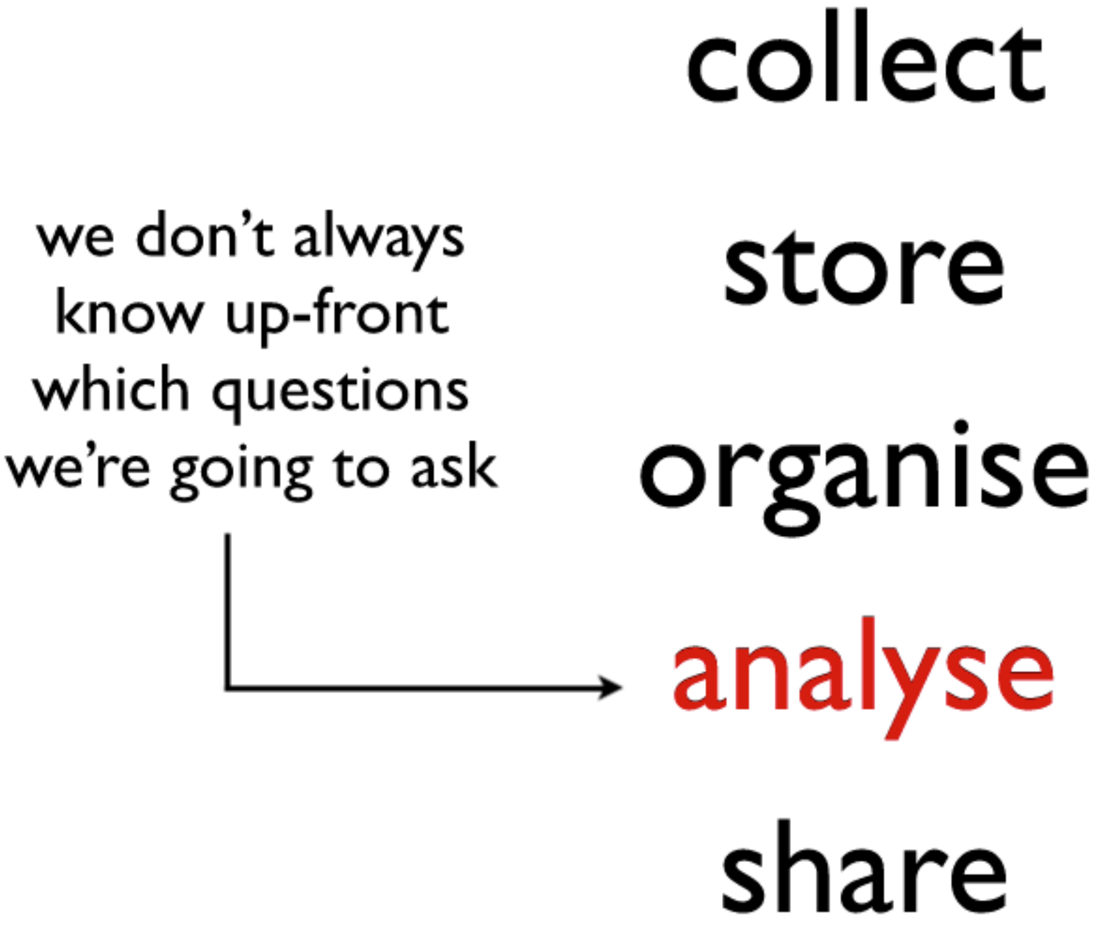
organise

analyse

share

Werner Vogels, CTO, Amazon
- STRATA Conf 2011

Big Data



collect

store

organise

analyse

share

we don't always know up-front which questions we're going to ask

Werner Vogels, CTO, Amazon
- STRATA Conf 2011