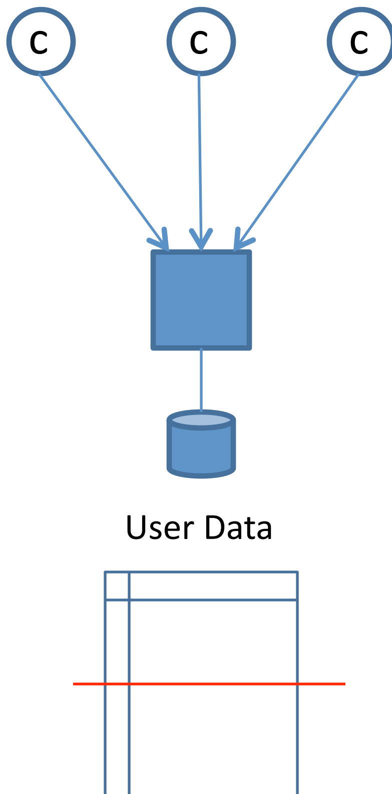
PARTITIONING IN LARGE DATA SYSTEMS

(IF IT IS NOT DYNAMIC, WHAT'S THE POINT?)

NYU Advanced Databases Class, Invited Lecture, Oct/2009

alberto.lerner@gmail.com

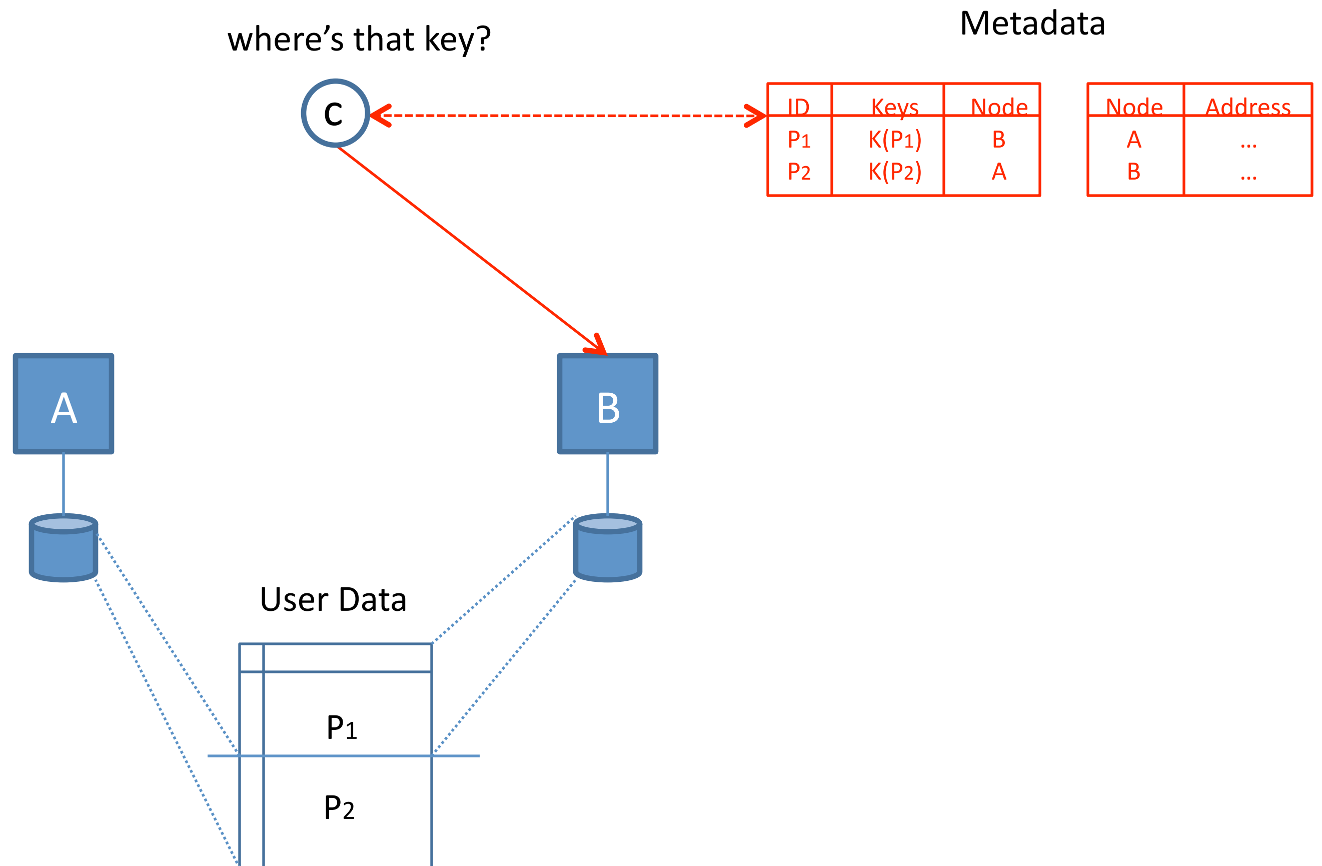
Motivation



Data partitioning as a way to scale

If only one could “stretch” a single server indefinitely as one grew…

Motivation



Which server holds a given key?

The system can maintain metadata on participant nodes and on partitioning. A client (library) can lookup metadata.

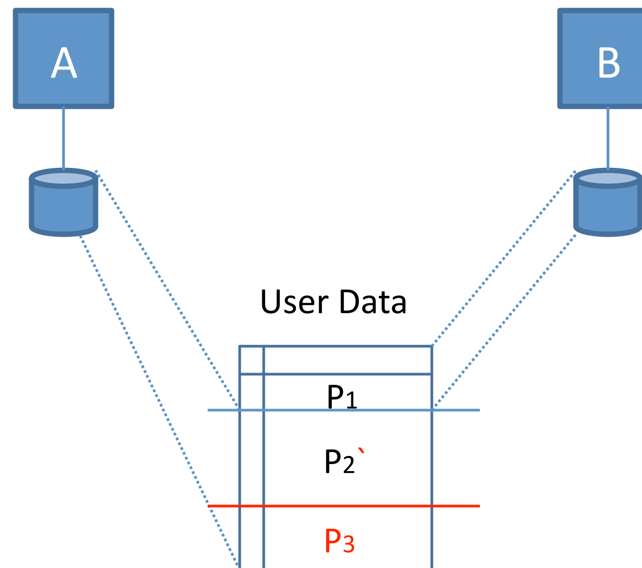
Motivation

Metadata

ID	Keys	Node
P_1	$K(P_1)$	B
P_2	$K(P_2)$	A
P_3	$K(P_3)$	A

Node	Address
A	...
B	...

"split" partition P_2



A partition's size changed? Repartition as it goes

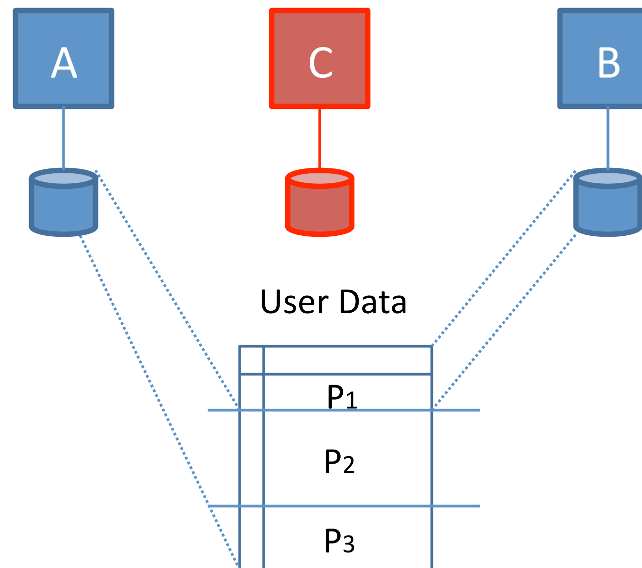
From the metadata perspective, the system can adjust a partitioning boundaries.

Motivation

Metadata

ID	Keys	Node
P1	K(P1)	B
P2	K(P2)	A
P3	K(P3)	A

Node	Address
A	...
B	...
C	



If need be, system can accept new nodes

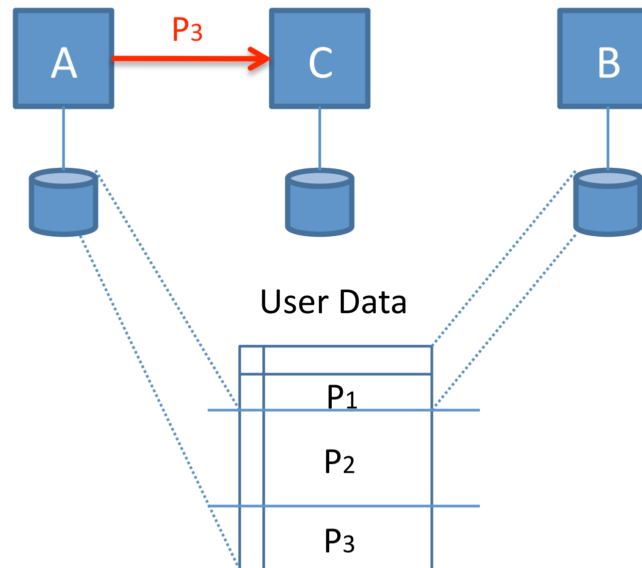
Again (and quite simplistically) adding a node would mean adjusting the system metadata.

Motivation

Metadata

ID	Keys	Node
P1	K(P1)	B
P2	K(P2)	A
P3	K(P3)	A -> C

Node	Address
A	...
B	...
C	...



Rebalances as it goes as well

The system may reassign load from one node to another.

Motivation

Metadata

ID	Keys	Node
P1	K(P1)	B
P2	K(P2)	A
P3	K(P3)	C

Node	Address
A	…
B	…
C	…

- **Locate** a given key
- **Split** a partition
- **Add** a node
- **Migrate** a partition
- Merge partitions
- Subtract a node

Recap: dynamic partitioning primitives

What's hard about implementing these primitives in practice?

Agenda

Metadata

ID	Keys	Node
P1	K(P1)	B
P2	K(P2)	A
P3	K(P3)	C

Node	Address
A	…
B	…
C	…

1. **Partition** system data across nodes on the system
    - Changes should be made in the responsible node
    - No node has global information

2. **Replicate** (not necessarily fully) system data on all system nodes
    - Whenever a change is made, propagate across the system
    - All nodes have a view of global information, even if sometimes not a current one

For one, metadata *must not* be a single point of failure!

Let's look at alternative strategies to implement dynamic partitioning from the ***metadata perspective***.

Agenda

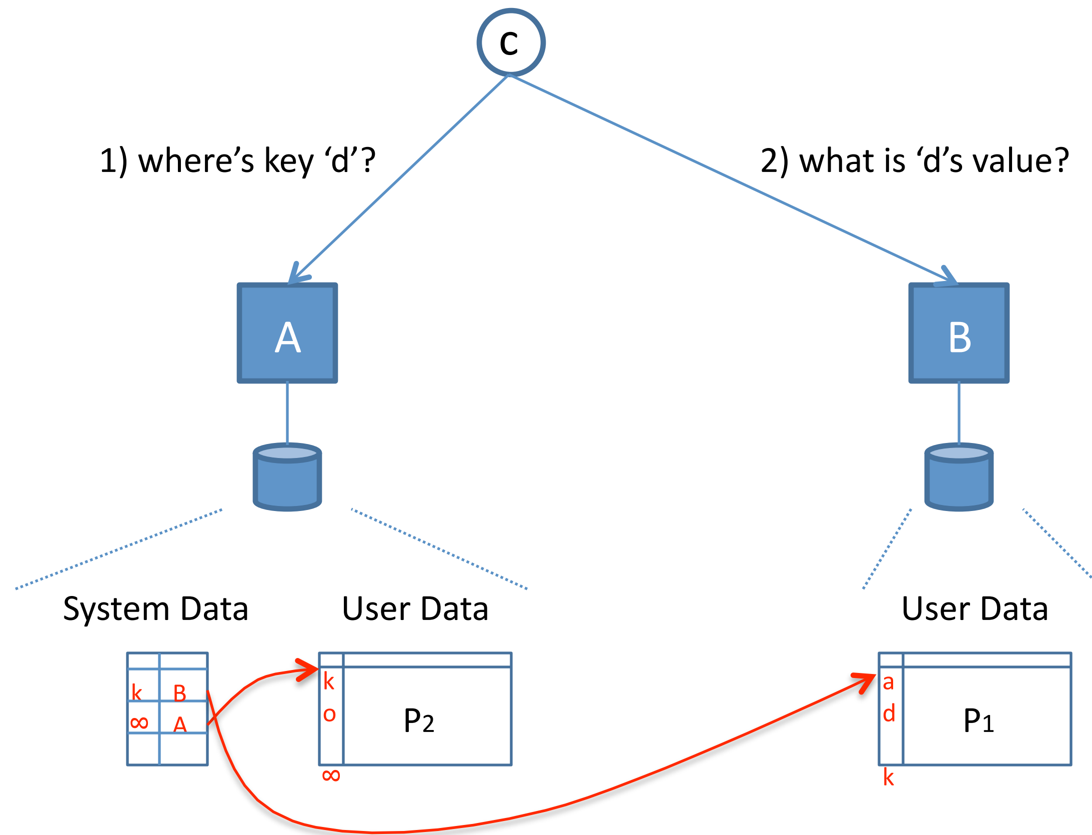
• We'll assume a simple **key-value** data model. (In practice, models can be more sophisticated.) And we'll try to make the fewest possible assumptions about th**e underlying storage** layout. (Of course, partitioning the data has its challenges too.)

• For our purposes here, let's assume that there is a completely orthogonal **data replication scheme** for fault tolerance**. (**Orthogonal?! Yeah, right.)

• We won't go into **crash recovery** nor into **fault detection**

What the talk is *not* about

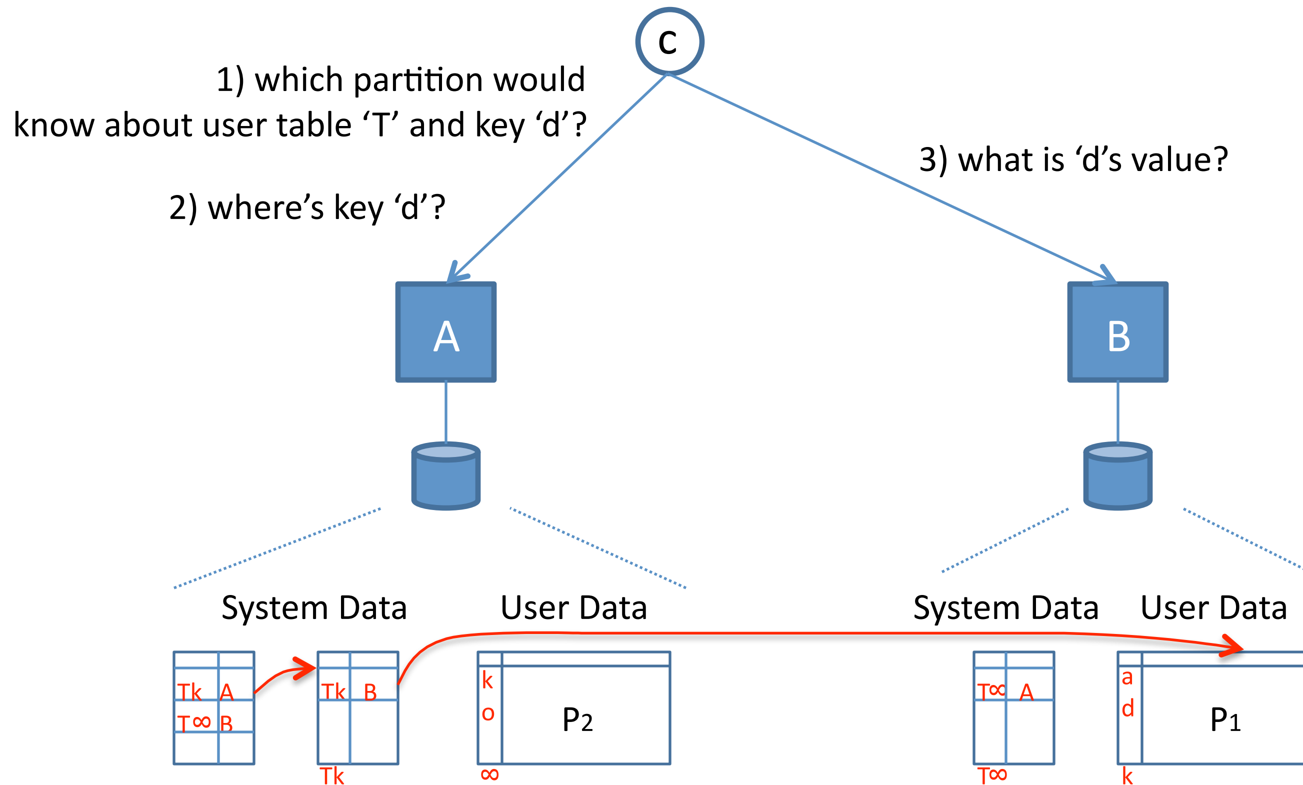
But, please, let's fill a whiteboard on any of these offline!



Idea: the partition table is … a table!

But only the system can update it. Table read is public, though.

Partition

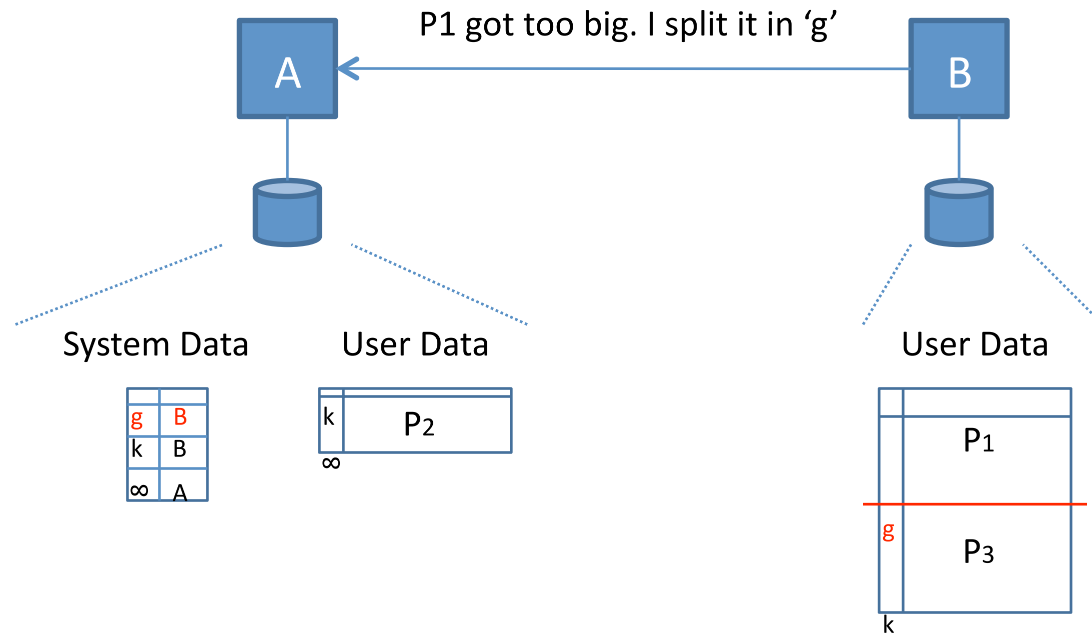


The partition table can be… partitioned

Lookup’s complexity is logarithmic.

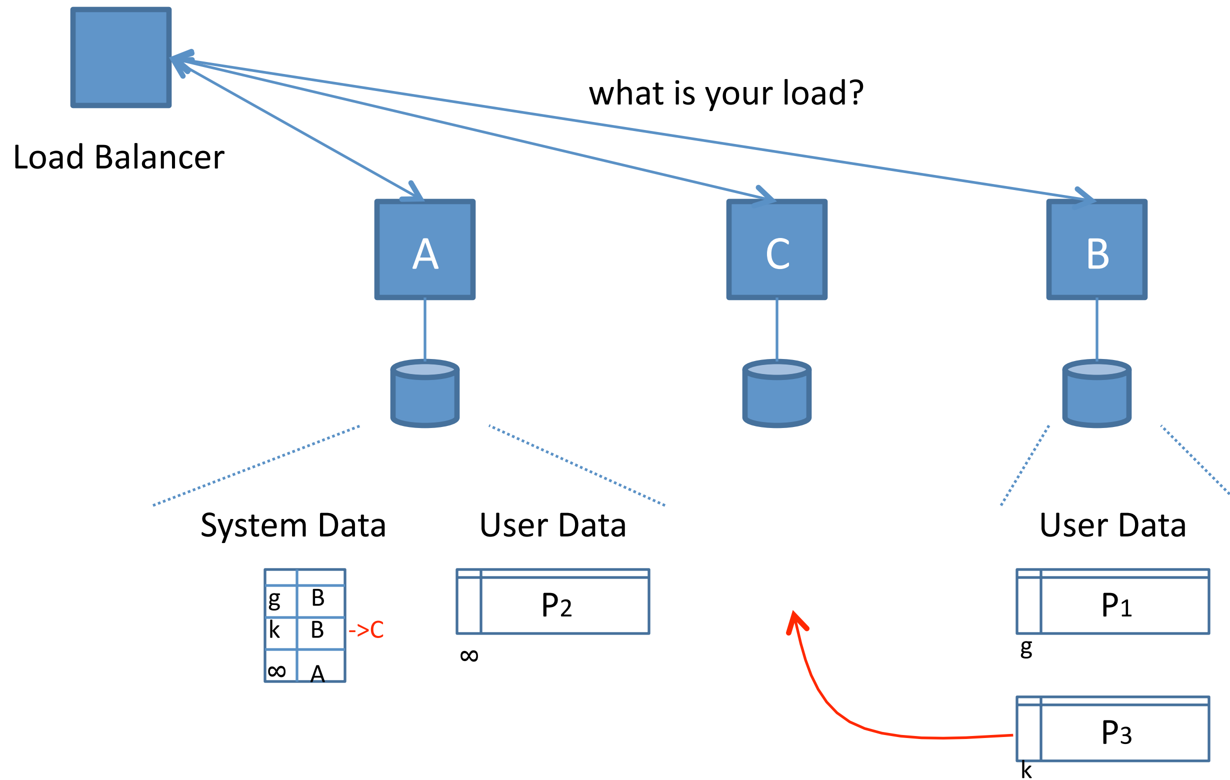
Partition

C



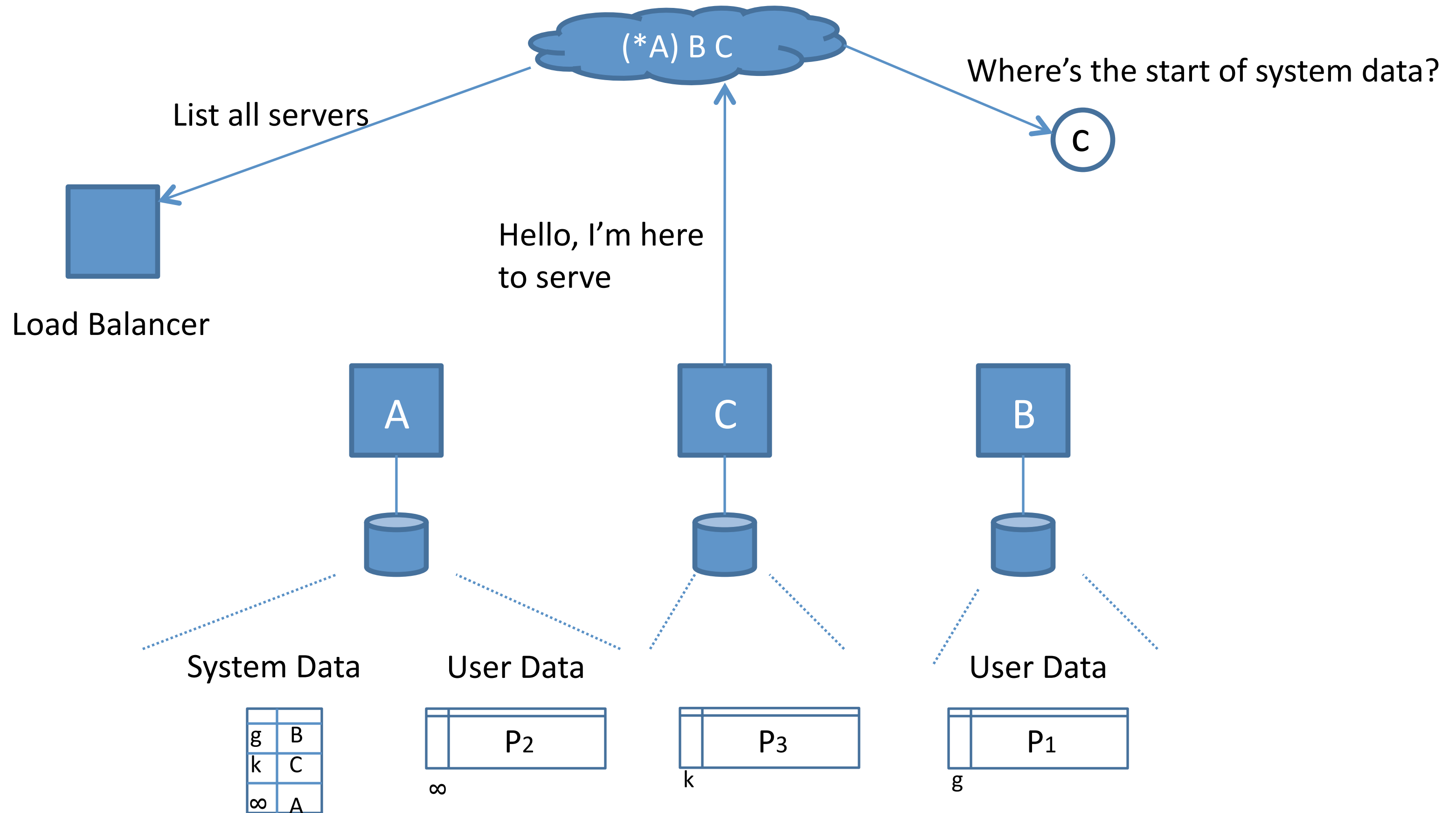
Is split a local operation?

From the metadata perspective, probably not.



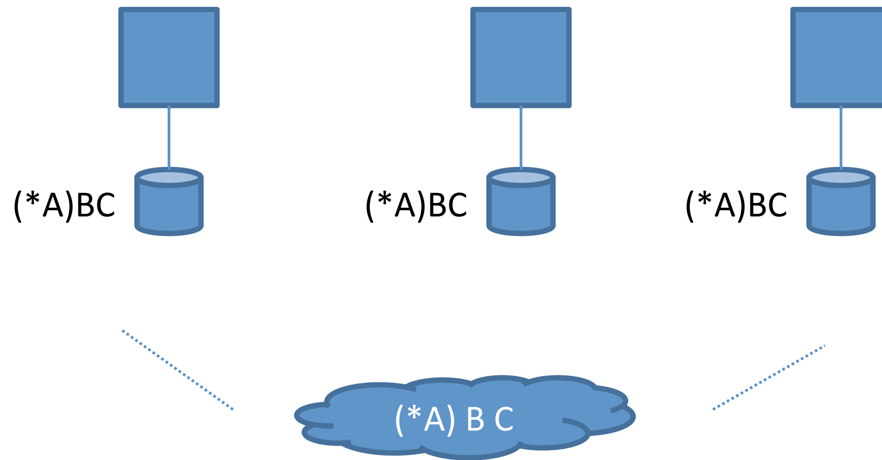
Load Balancing

Pull load information from all the nodes. Migrate partitions if necessary. Do it again periodically.



A missing piece?

Where does a client start looking for system data? How does the load balancer know about the nodes in the system? How does a node join the system?

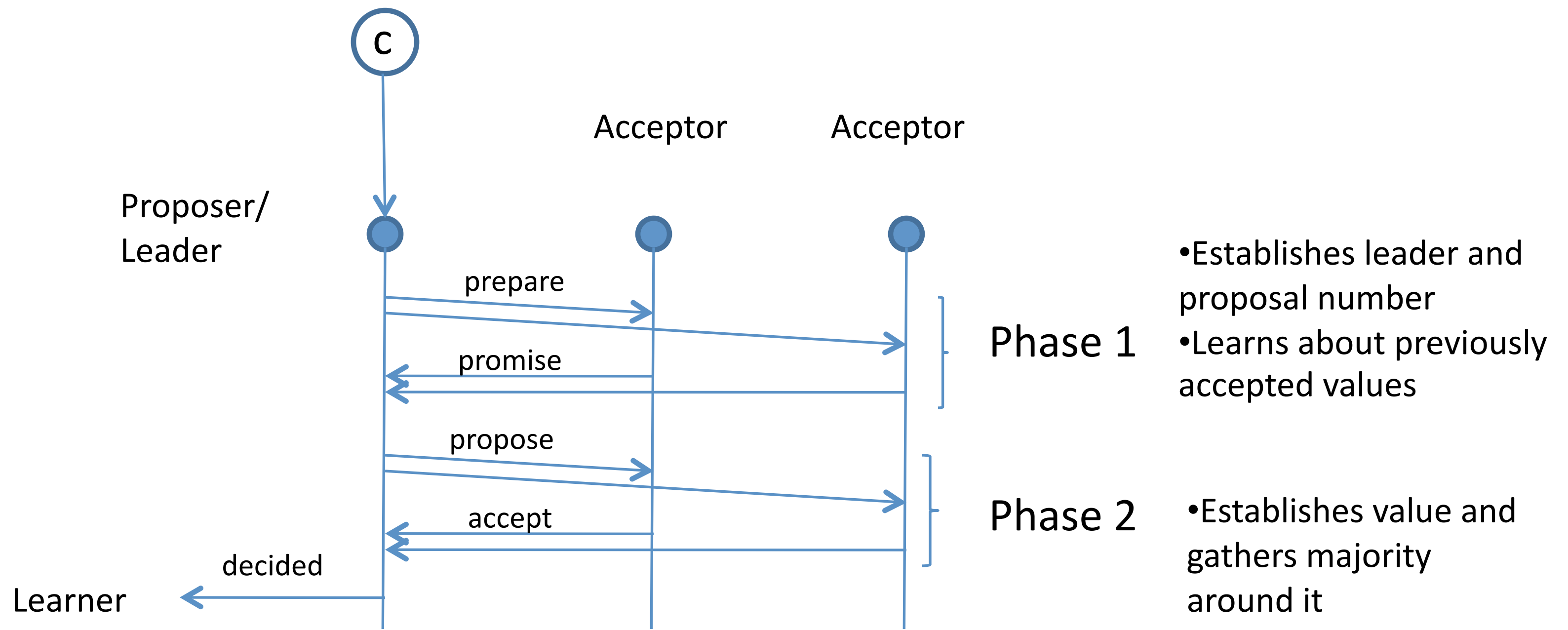


Partition

Was that a central point of failure?

No. The information is kept in several places that agree on values using a distributed consensus algorithm.

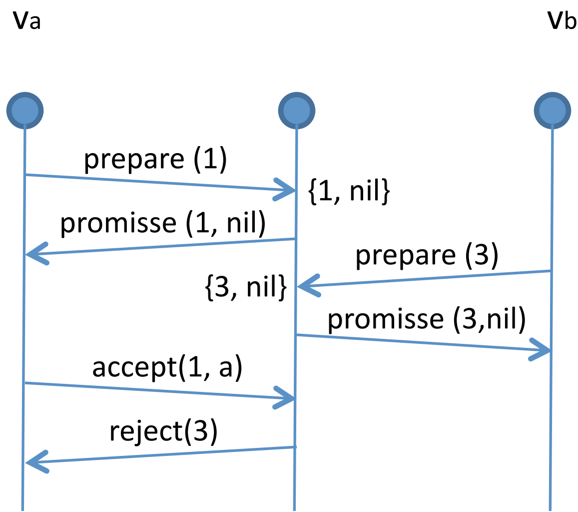
Partition



Paxos

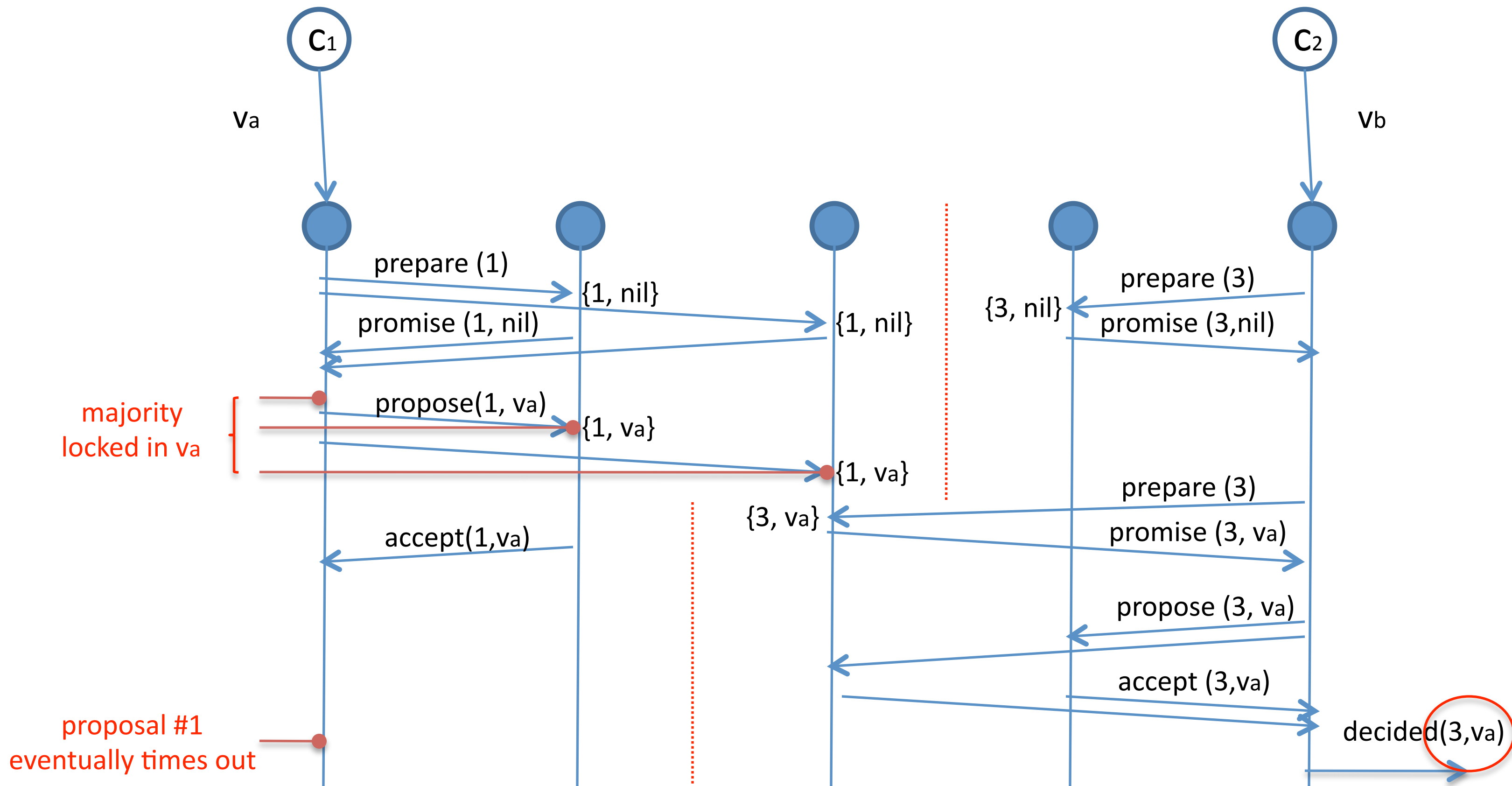
Solves the distributed consensus problems in scenarios involving crashes, omissions, and restarts. Depicted here is a very well behaved instance. Under faults, each phase may involve several rounds of messages.

Partition



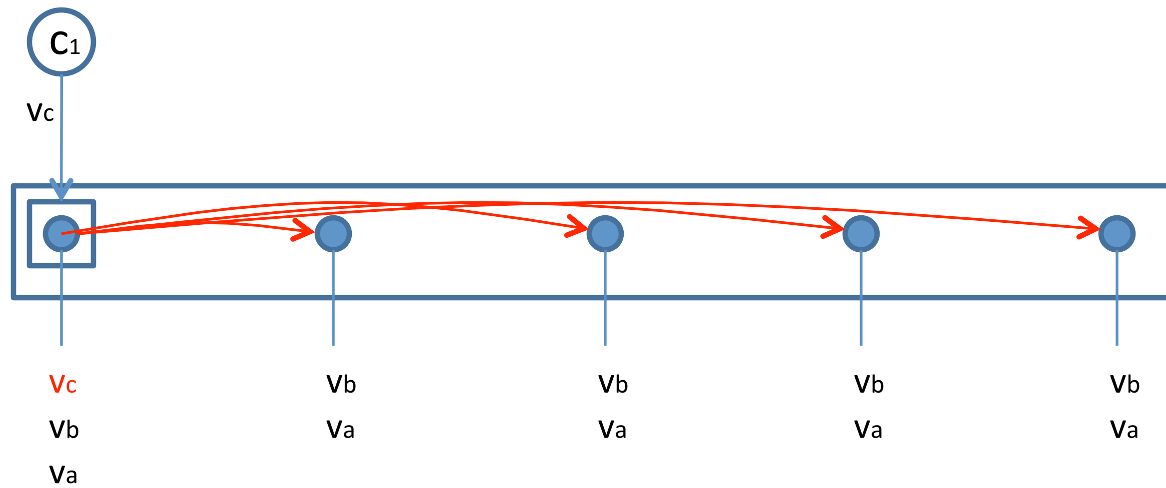
Liveness

Any node can be a proposer. In fact, this guarantees liveness, in case a previous leader crashes or gets disconnected.



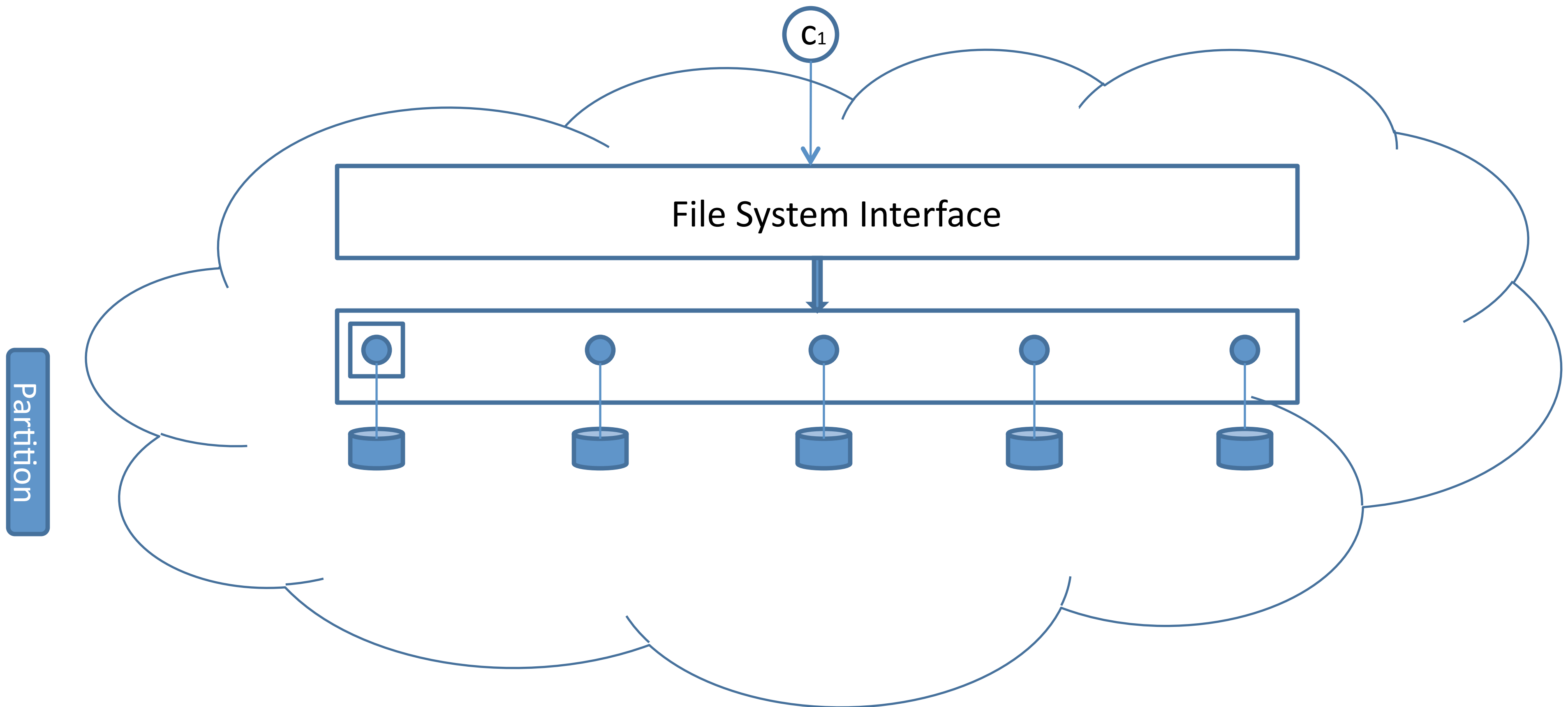
Correctness

Once a majority of nodes locks in a value, that value is propagated through later proposals.



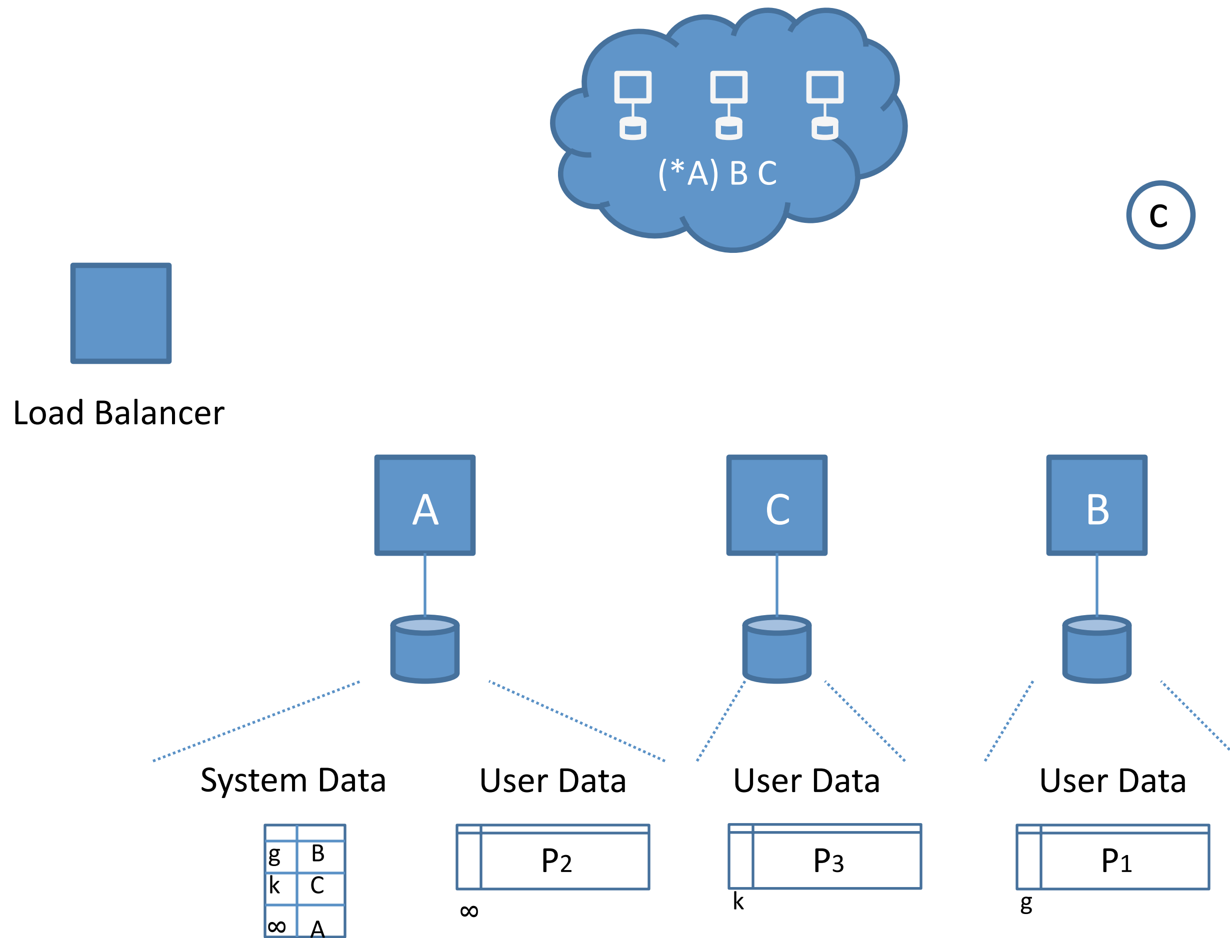
Replicated State Machines

Use a Paxos instance to decide on the next operation (or on the next instance leader). Having a leader across instances speeds things up.



Distributed Lock Manager

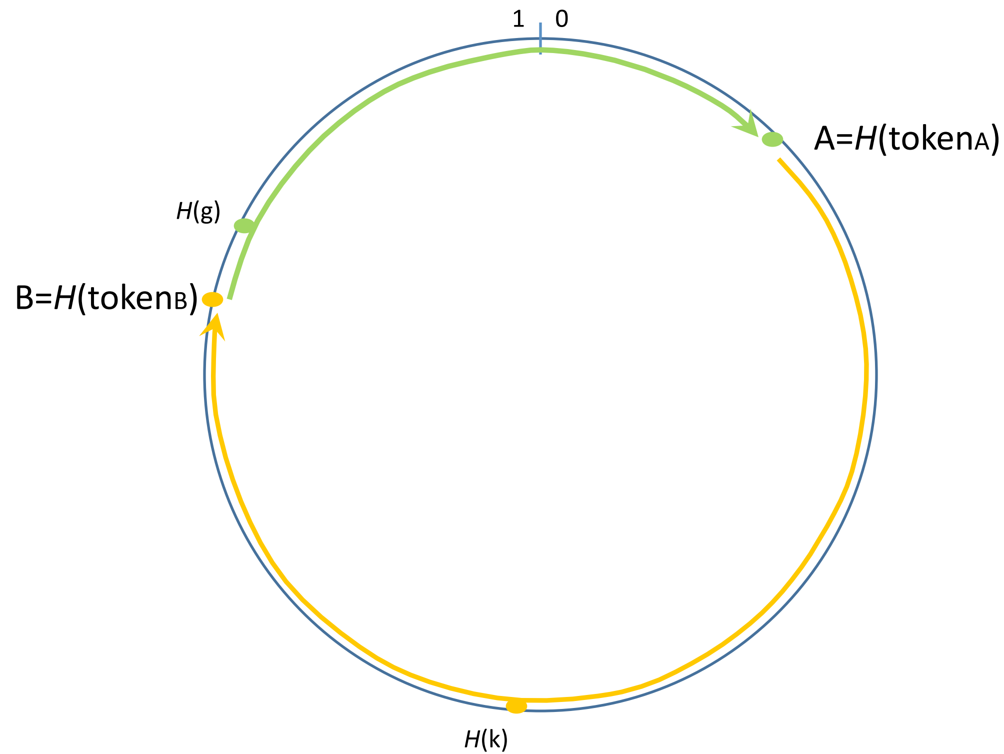
Wraps a Paxos layer with a file system interface, complete with locks, sequencers, and watchers. Now, can you solve the group membership problem?



And back to the big picture

Sync point: we've seen key location, partition split, partition migration, and node addition under the first partitioning scheme. What could be a motivation for another scheme?

Node	Address
A	…
B	…

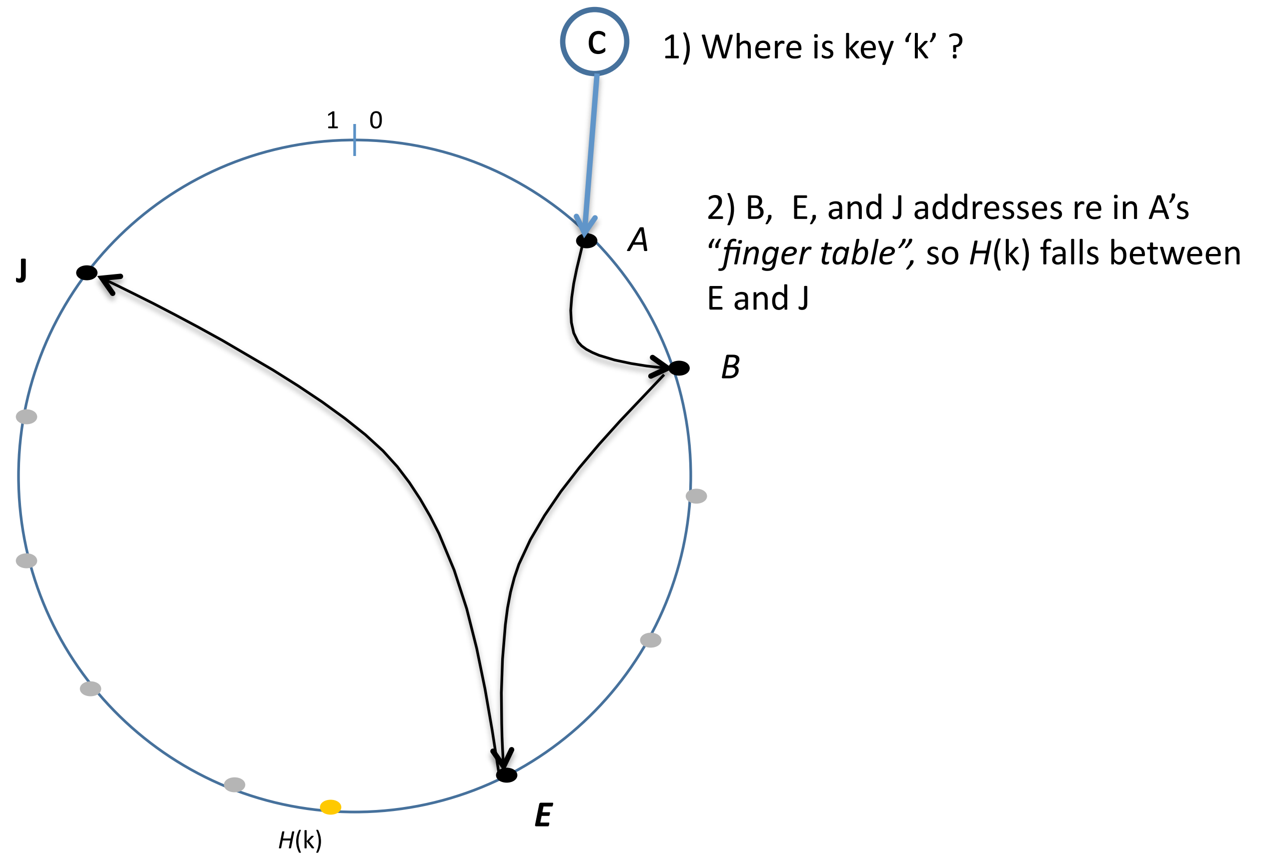


User Data

Replicate

Consistent Hashing

Nodes’ tokens and user data’s keys are hashed and placed on a ring representing the hash space. A key belongs to the first clock-wise node. That is, ***partitioning is implicit***.

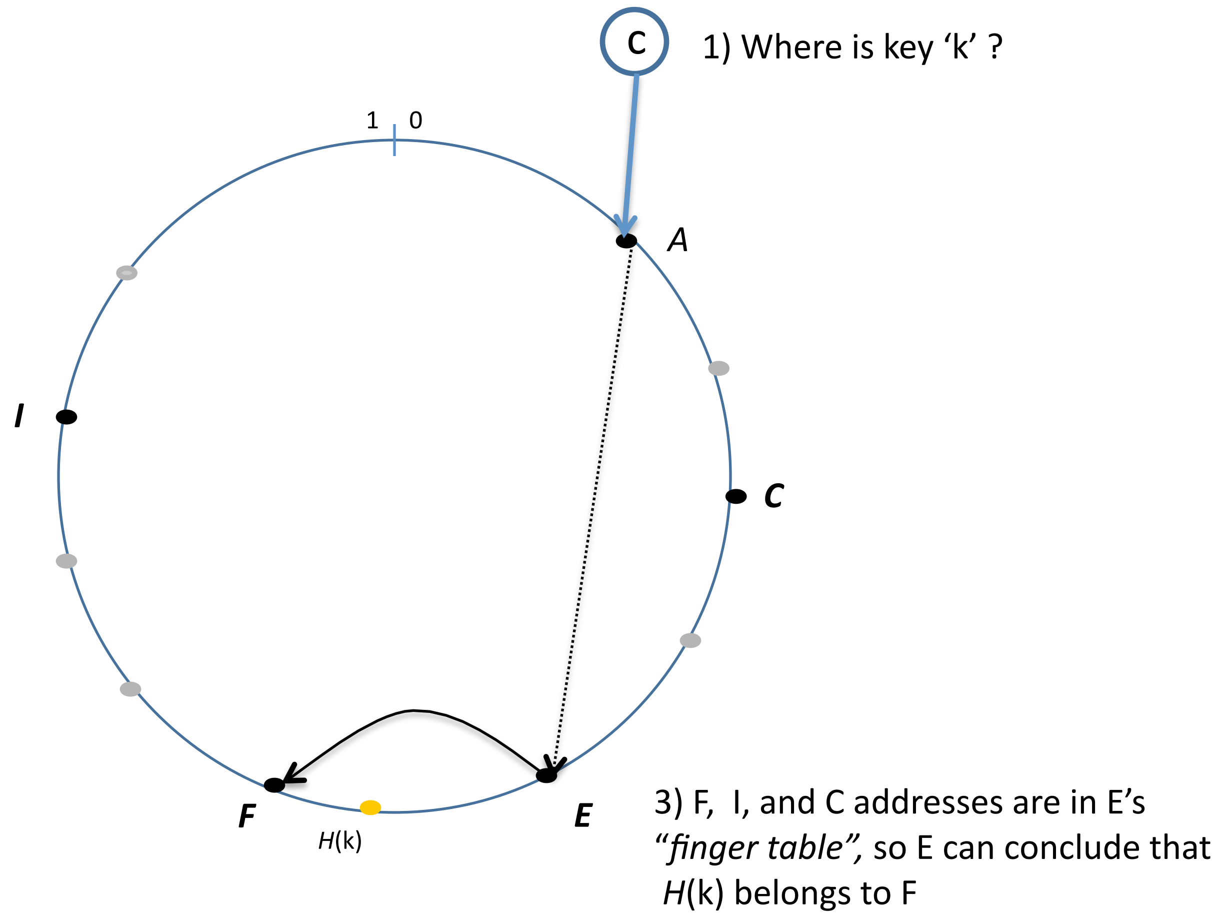


Replicate

Finger tables

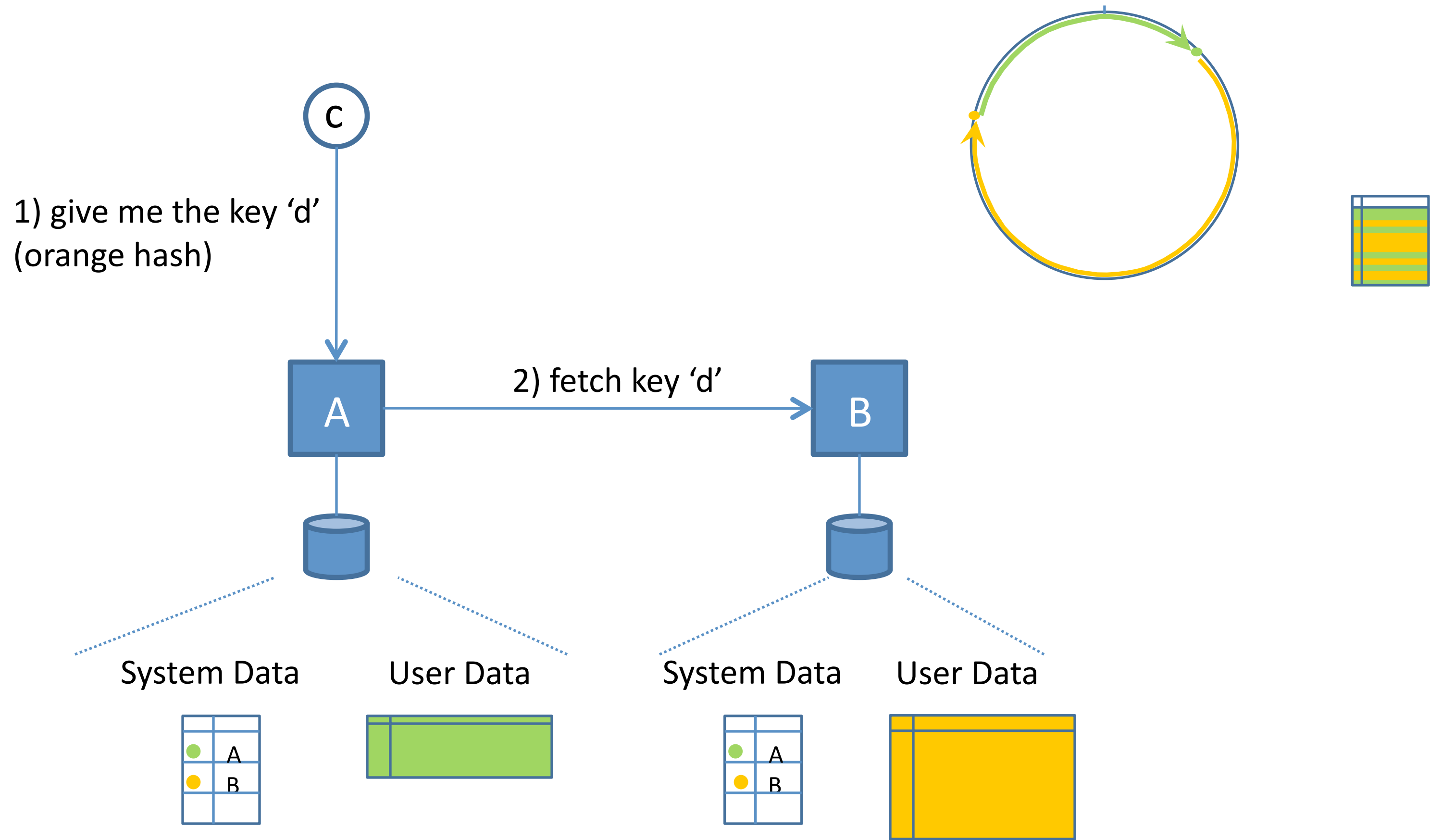
Each node knows the addresses (and hashes) of increasingly distant nodes in the ring, in particular, it’s *successor.*

Replicate



Routing in DHT's

Efficient schemes are known that route requests in O(log n). Replicating a "complete" finger table would bring that to O(1) but...

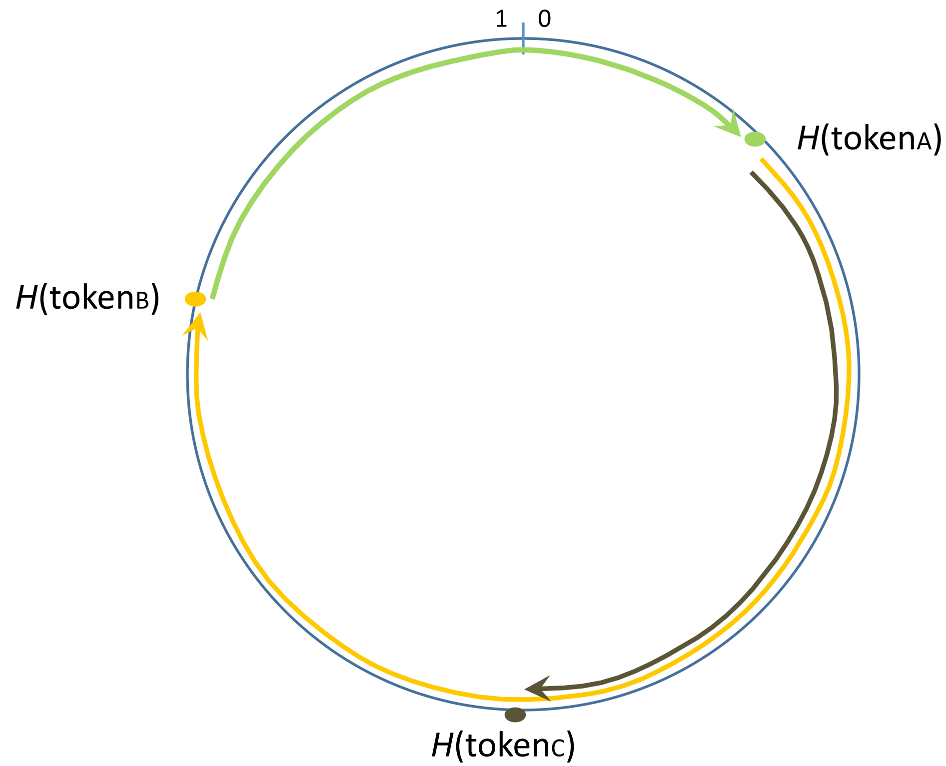


Replicate

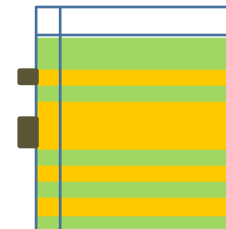
How does a client go about finding a key?

Logic of key location can be pushed to clients. Or the client can ask a node to redirect requests.

Node	Address
A	…
B	…
C	



User Data



Adding a new node

Impact only on “following” node, which has to transfer some of its keys to the new arrival.

Replicate

Replicate

1) hello, I'm 'token C' and here's my address

2) transfer "brown" keys

A

B

C

System Data

User Data

System Data

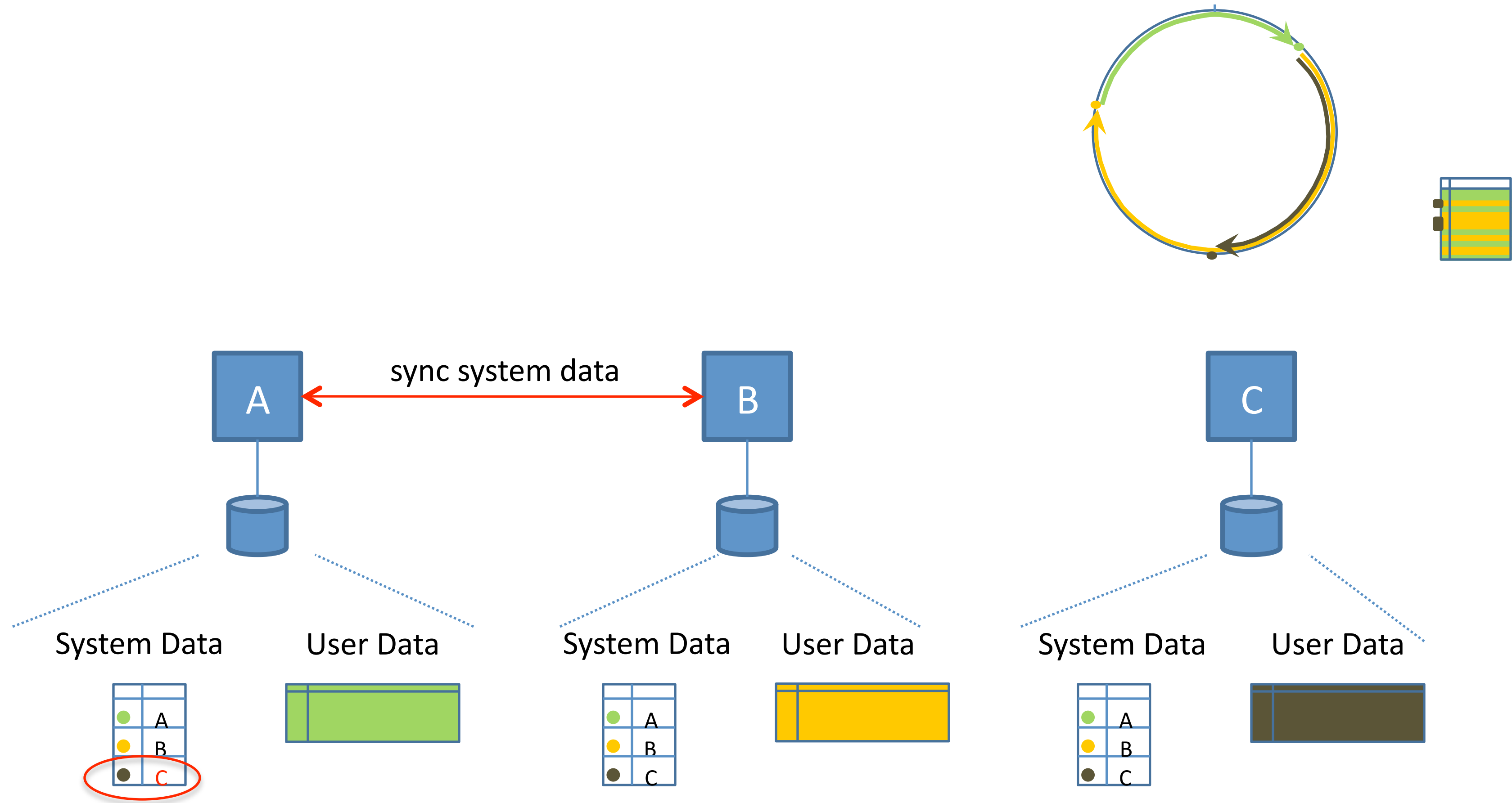
User Data

System Data

User Data

Adding a node

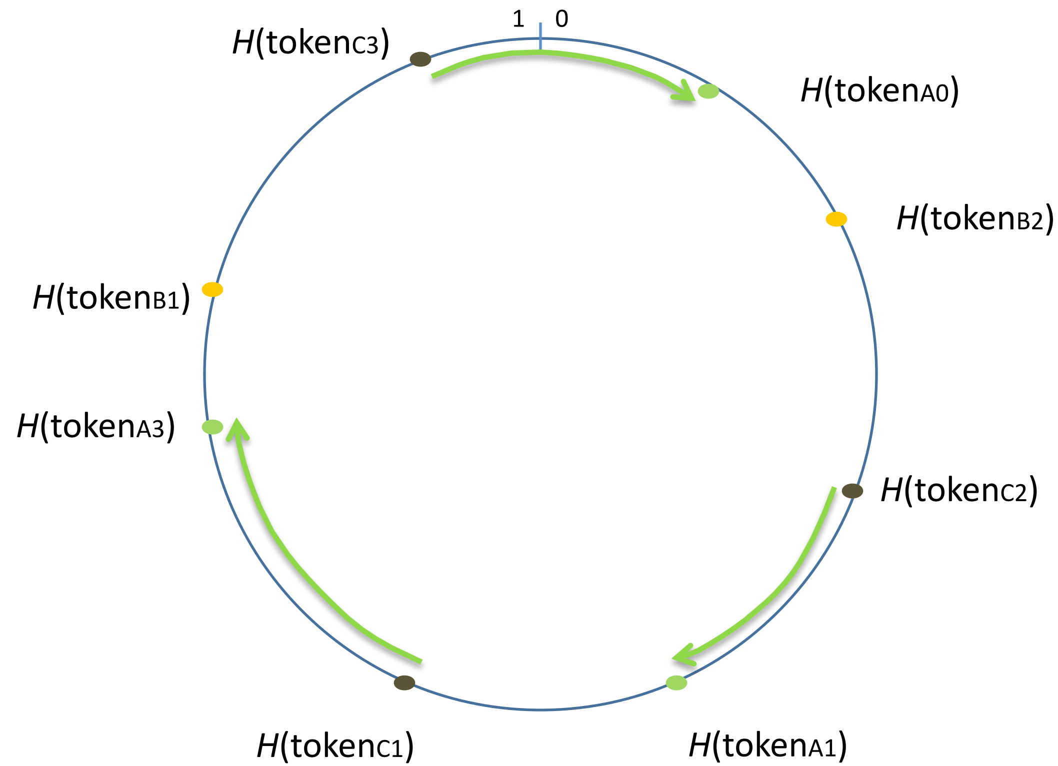
Node announces itself to successor. The latter offers to transfers the relevant keys.



Group membership through gossip

Each node contacts a random node periodically and they reconcile their system data. Eventually all nodes learn about incoming and outgoing nodes.

Node	Address
A	...
B	...
C	...

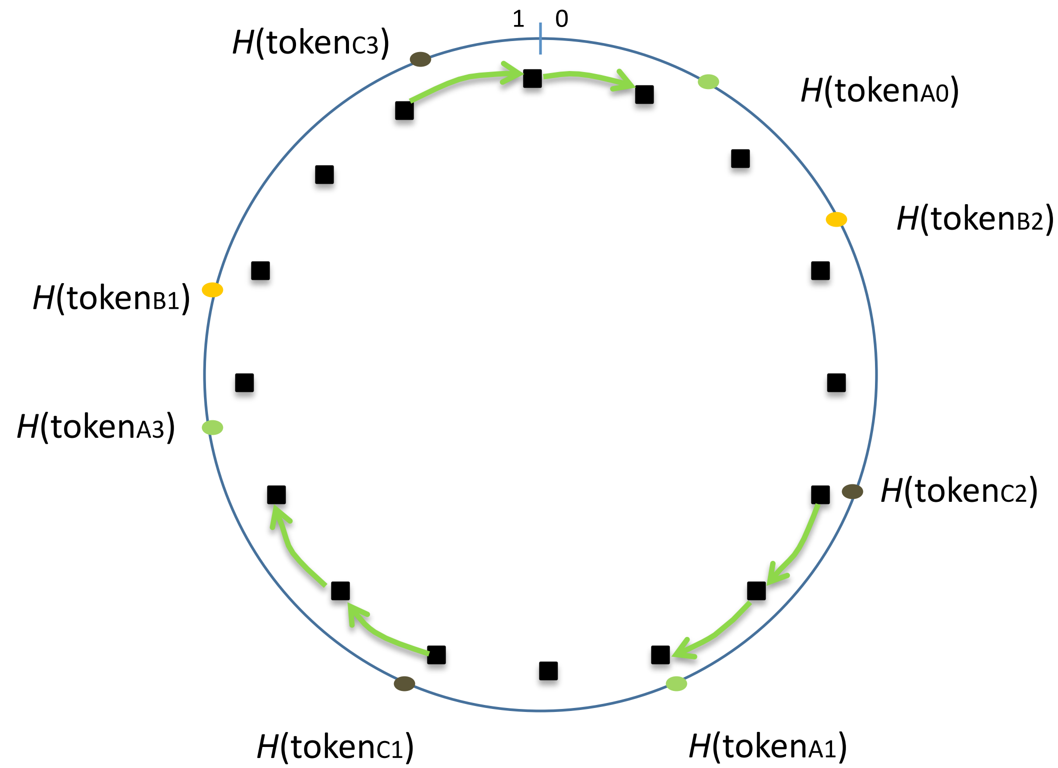


Replicate

Even partitioning through virtual nodes

One may assign T tokens per node to assure even key distribution. Now, what if we wanted to keep the same partition number and just throw one more server?

Node	Address
A	...
B	...
C	...



Replicate

Equal-sized partitions

Use Q partitions regardless of number of servers. Now, can Q vary?

- Does a uniform **key distribution** guarantee uniform **load distribution**?

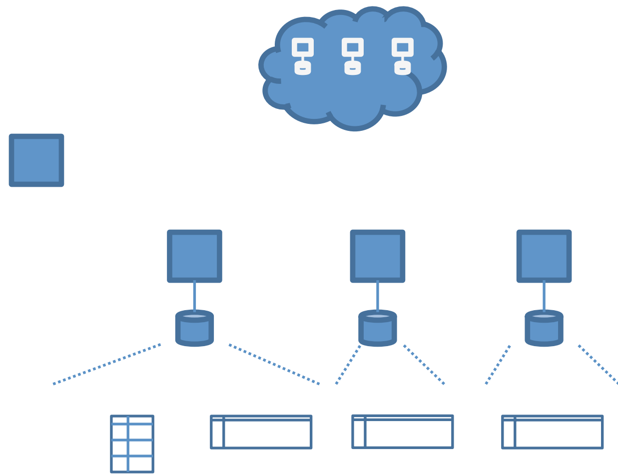
- How about the ability to issue **sequential scans**? Is it lost for the sake of load balancing?

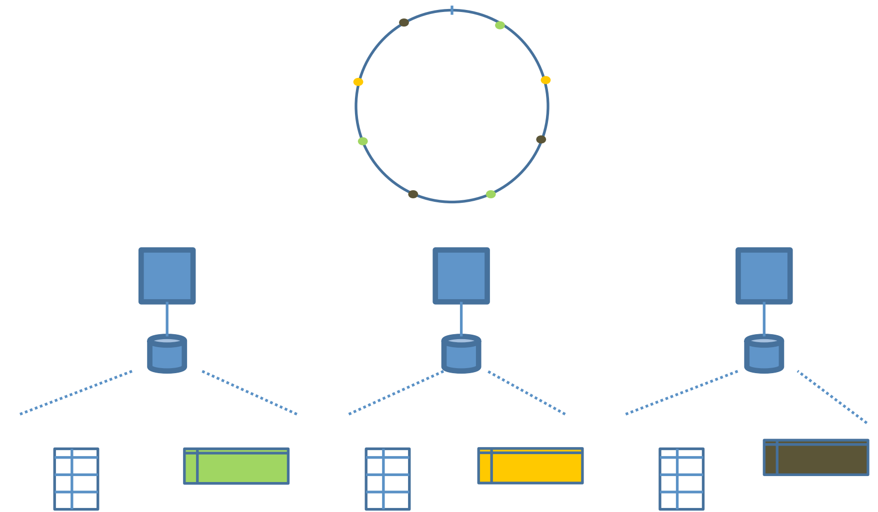
- How likely replication system data would keep scaling?

Replicate

Load balancing, questions

Adopters have reported positive answers to all the above.





- Partitions metadata information
- Uses a central authority
- Partitioning is explicit (range)
- Maps partitions to nodes

- Replicates metadata information
- Does away without a central authority
- Partitioning is implicit (hashing)
- Partitions “fall” into nodes

Are they really that different?

Ultimately, yes. And both are successfully deployed.

Wrap-up

References

- Bigtable: A Distributed Storage System for Structured Data, Chang et al., OSDI'06
- The Chubby Lock Service for Loosely Coupled Distributed Systems, Mike Burrows, OSDI'06
- Paxos Made Simple, Leslie Lamport

- Dynamo: Amazon's Highly Available Key Value Store, DeCandia et al., SOSP'07
- Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the WWW, STOC'97
- Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, Stoica et al, SIGCOMM'01

Wrap-up