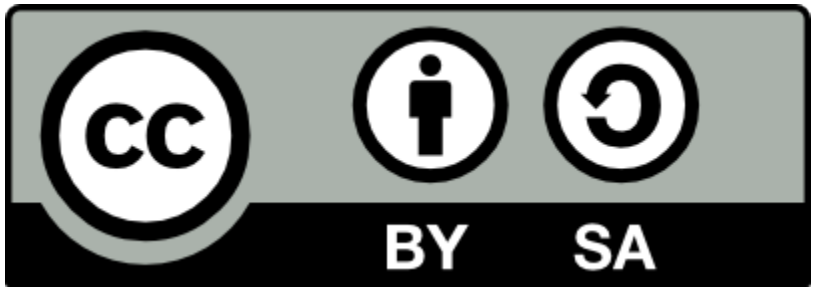
Apache Cassandra

By Markus Klems
(2014)

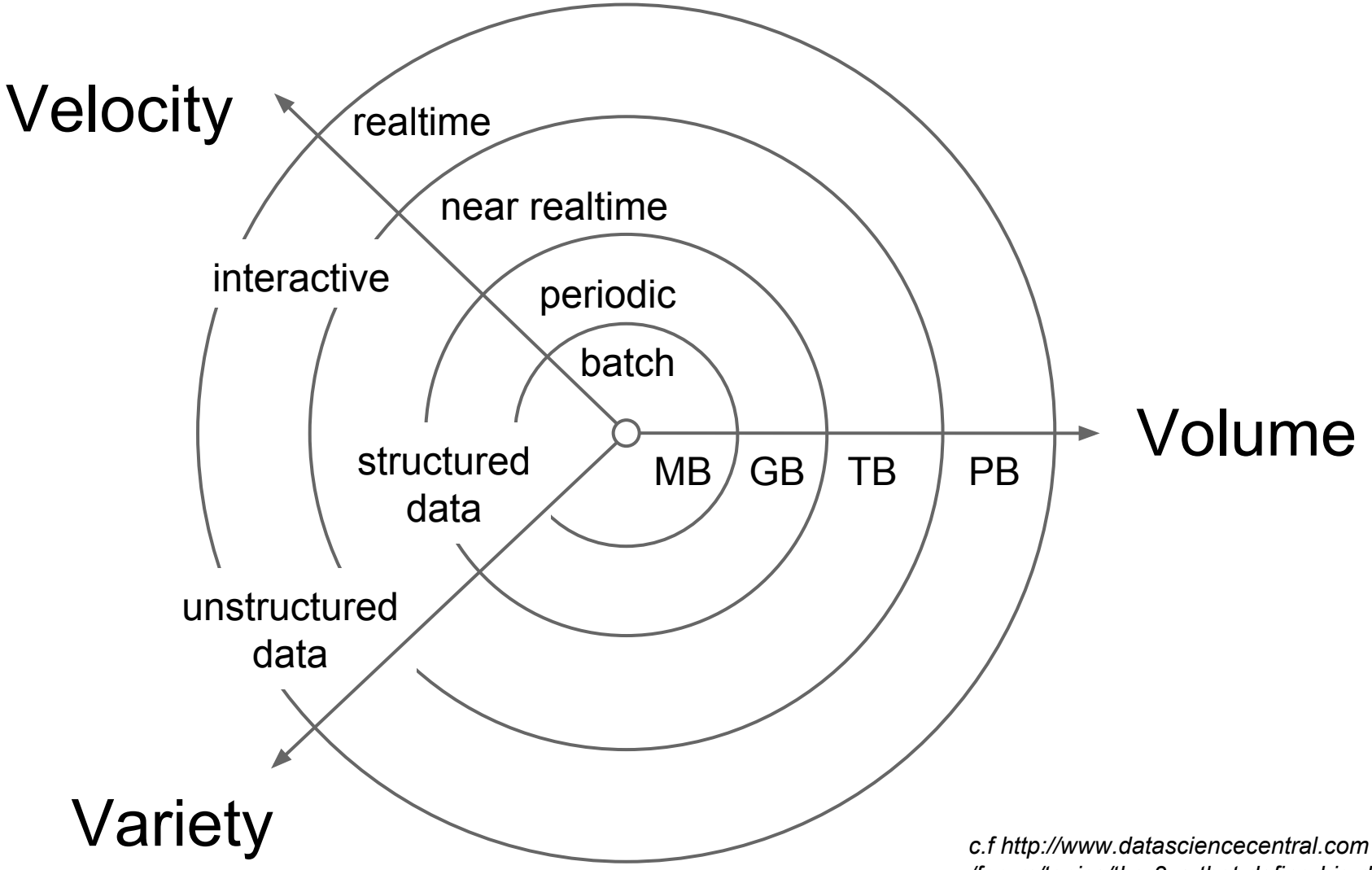


Big Data & NoSQL

Why do we need NoSQL Databases?

- Modern Internet application requirements
  - low-latency CRUD operations
  - elastic scalability
  - high availability
  - reliable and durable storage
  - geographic distribution
  - flexible schema
- Less prioritized
  - Transactions, ACID guarantees
    - ... but some form of data consistency is still desirable
  - SQL support
    - … but some SQL features are still desirable

What is Big Data?



c.f http://www.datasciencecentral.com /forum/topics/the-3vs-that-define-big-data

Scalability & High Availability

- Workload and data volume grows -> Partition and distribute data across multiple servers (horizontal scaling)
  - How to determine partition boundaries?
  - Can partitions be changed dynamically?
  - How to route requests to the right server?
- High Availability -> replicate data
  - What kind of failures can we deal with?
  - Sync or async replication?
  - Local or geo replication?
  - Consistency model?

NoSQL Databases categorized by System Architecture

Architecture	Techniques	Systems
Dynamo-style Ring (P2P)	All nodes are equal. Each node stores a data partition + replicated data. Eventually consistent.	Cassandra, Riak, Voldemort, Amazon DynamoDB
Master-Slave	Data partitioned across slaves. Each slave stores a data partition + replicated data. Strong consistency guarantees.	HBase, MongoDB, Redis, Yahoo! Sherpa, Neo4j
Full replication	Bi-directional, incremental replication between all nodes. Each node stores all data. Eventually consistent.	CouchDB

NoSQL Databases categorized by Data Model and Storage

Wide-Column	Each row stores a flexible number of columns. Data is partitioned by row key.	Cassandra, HBase, Amazon DynamoDB
Document	Storage and retrieval of structured data in the form of JSON, YAML, or RDF documents.	CouchDB, MongoDB
Key-value	Row-oriented data storage of simple (key,value) pairs in a flat namespace.	Riak, Redis, Voldemort, Yahoo! Sherpa
Graph	Storage and retrieval of data that is stored as nodes and links of graphs in a graph-space.	Neo4j

Cassandra Background: Amazon Dynamo + Google BigTable

Dynamo: Amazon's highly available key-value store

- **Amazon Dynamo Paper**

  Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles* (SOSP '07). ACM, New York, NY, USA, 205-220.
- Download: http://www.read.seas.harvard.edu/~kohler/class/cs239-w08/decandia07dynamo.pdf

Amazon Dynamo: Techniques (1)

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.

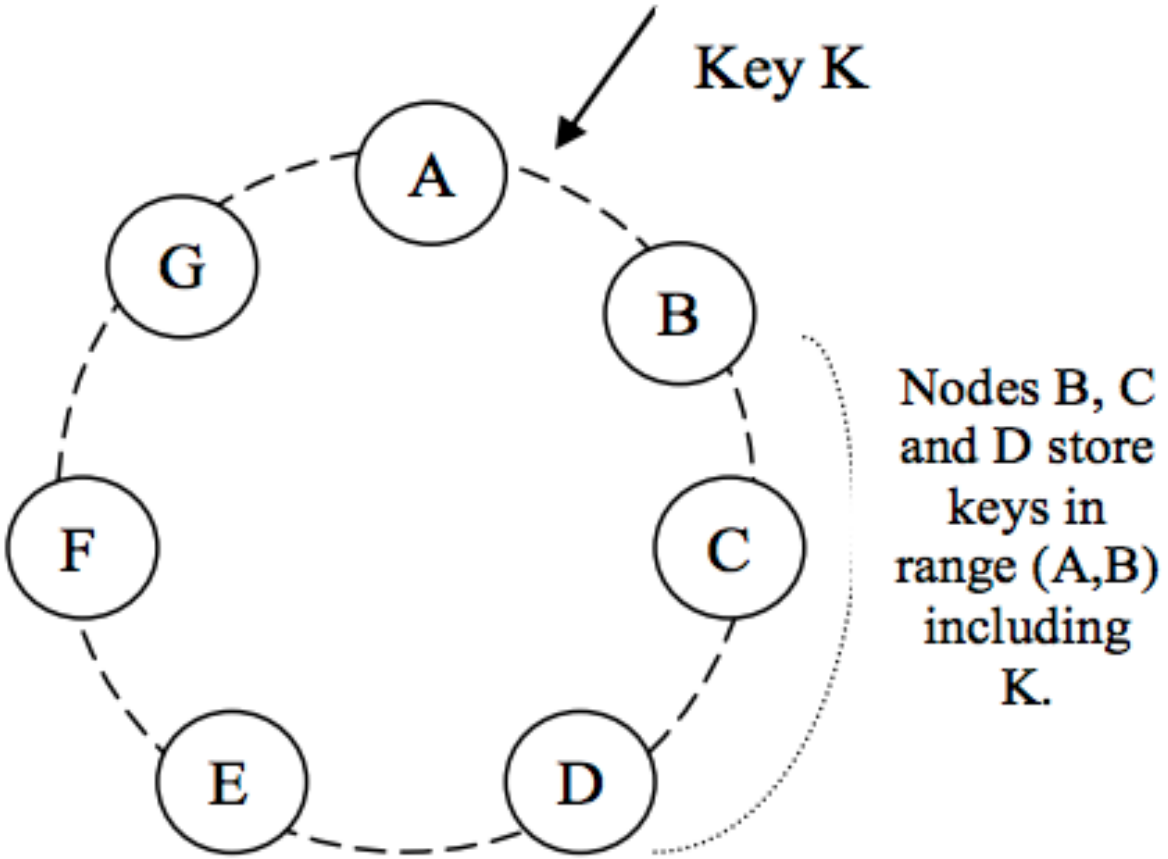
Amazon Dynamo: Techniques (2)

Problem	Technique	Advantage
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

Dynamo: Incremental scalability

- Simple key/value API
- Consistent hashing
  - Cryptographic MD5 hash of key generates a 128-bit identifier
  - The largest value wraps around the smallest one. The keyspace is a "ring".

Dynamo: Incremental scalability



Source: Amazon Dynamo paper

Dynamo: Data versioning

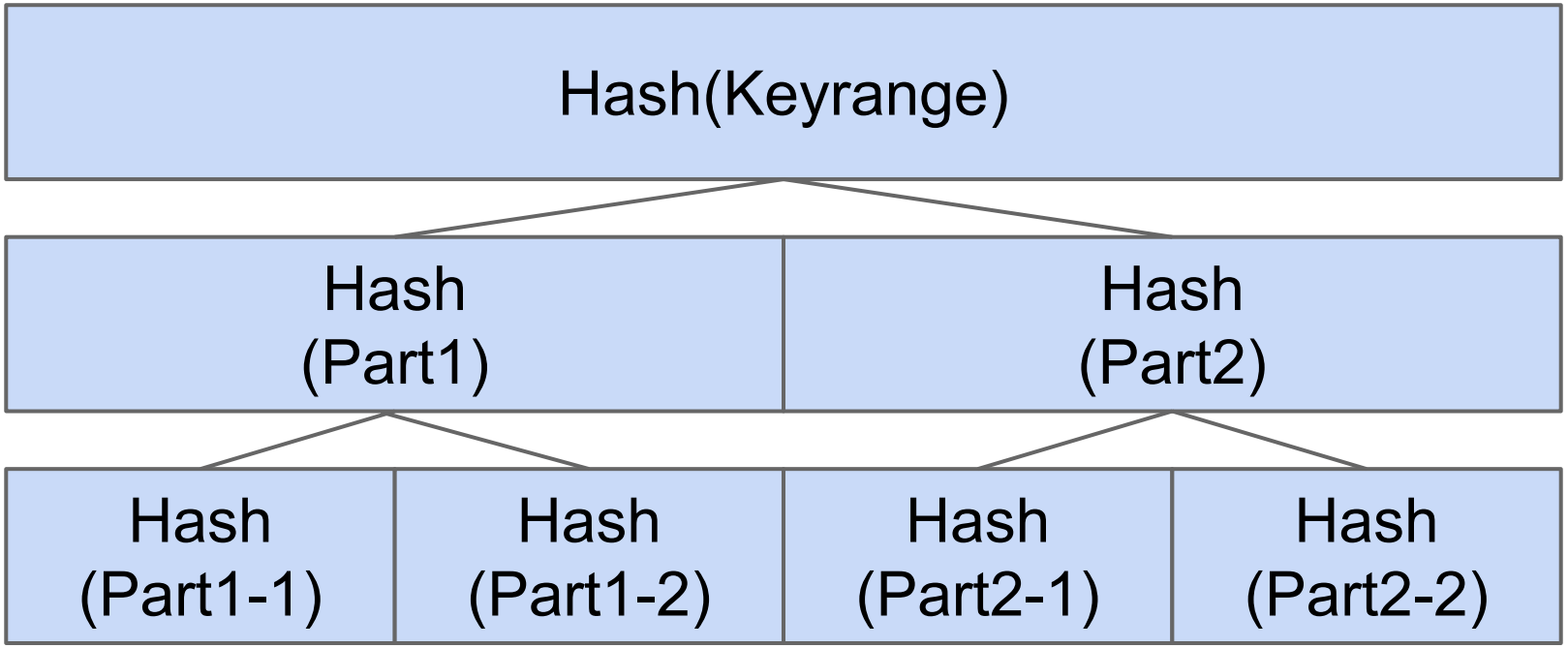
- Optimistic replication: updates may travel asynchronously. This can lead to inconsistencies.
  - How to identify inconsistencies?
  - Data versions, vector clocks!

Dynamo: Sloppy Quroum

- (N,R,W) quorum configuration
  - N replica
  - R votes for a successful READ operation
  - W votes for a successful WRITE operation
- Quorum intersection invariants
  - N < R+W
  - (W > N/2)
- Sloppy quorum
  - If a node goes down, save the data temporarily as "Hinted Handoffs" on another node.
  - Thus avoiding unavailability.
  - Hinted Handoffs must be resolved after some time.

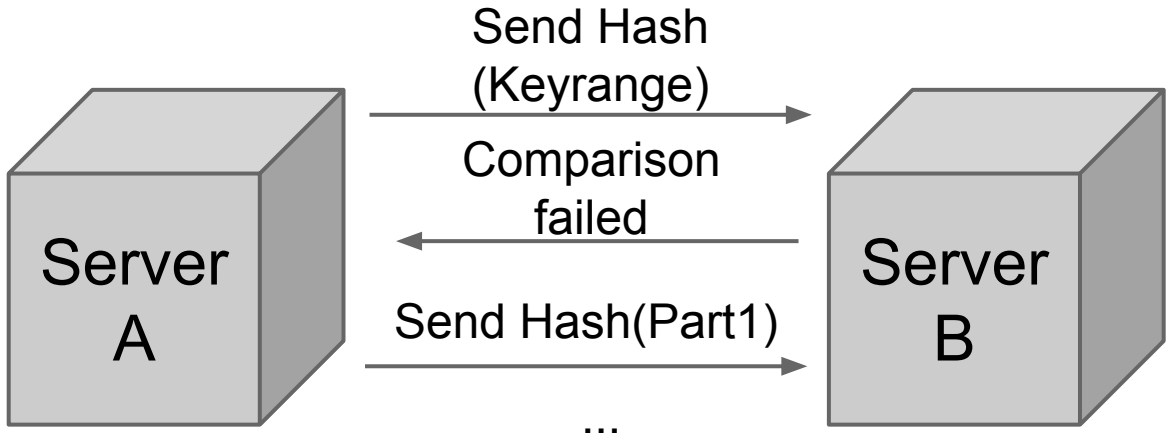
Dynamo: Merkle Trees

- Each server node calculates one Merkle Tree for one owned keyrange.
- Merkle Tree = A tree of hash values of parts of the keyrange.



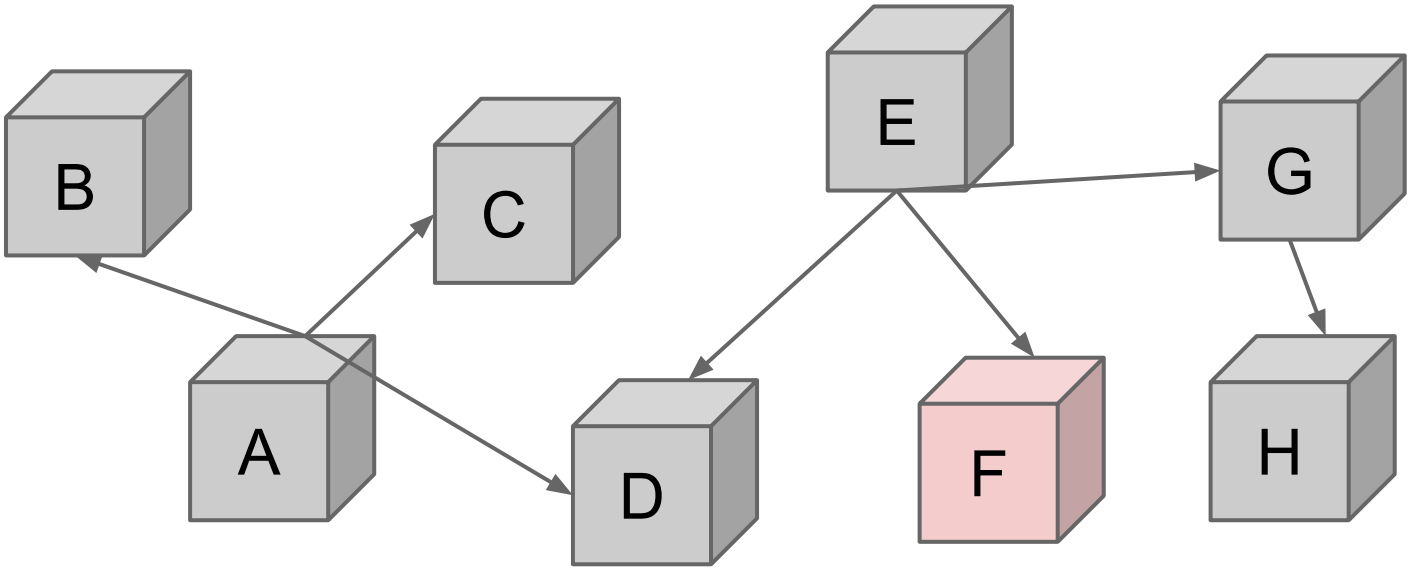
Dynamo: Anti Entropy Protocol

- Anti-entropy protocol
  - Exchange Merkle Trees with replica server nodes.
  - Lazy loading of Merkle Trees
  - Compare the tree and track down inconsistencies.
  - Trigger conflict resolution (last write wins)



Dynamo: Membership & failure detection

- Gossip-based protocol
  - All nodes (eventually) share the same view of the system.
  - Information is exchanged via gossip between peers



Dynamo: Summary

- **CAP theorem**: in a widely distributed system, strong consistency and high availability (+ low latency) cannot be achieved at the same time.
- **Optimistic replication**: improve availability + latency at the cost of inconsistencies
- Requires **conflict resolution**:
  - when to resolve conflicts?
  - who resolves conflicts?

Bigtable: A Distributed Storage System for Structured Data

- **Google Bigtable Paper**

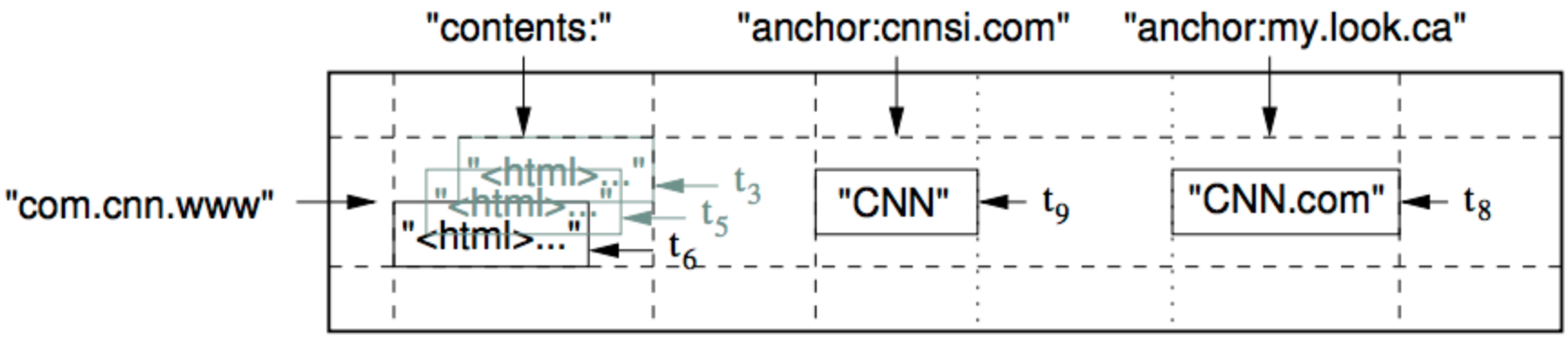
  Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2008. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* 26, 2, Article 4 (June 2008), 26 pages.
- http://research.google.com/archive/bigtable.html

Google Bigtable

"A Bigtable is a sparse, distributed, persistent multi-dimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes."

Source: Bigtable paper

Google Bigtable: Data Model



Source: Bigtable paper

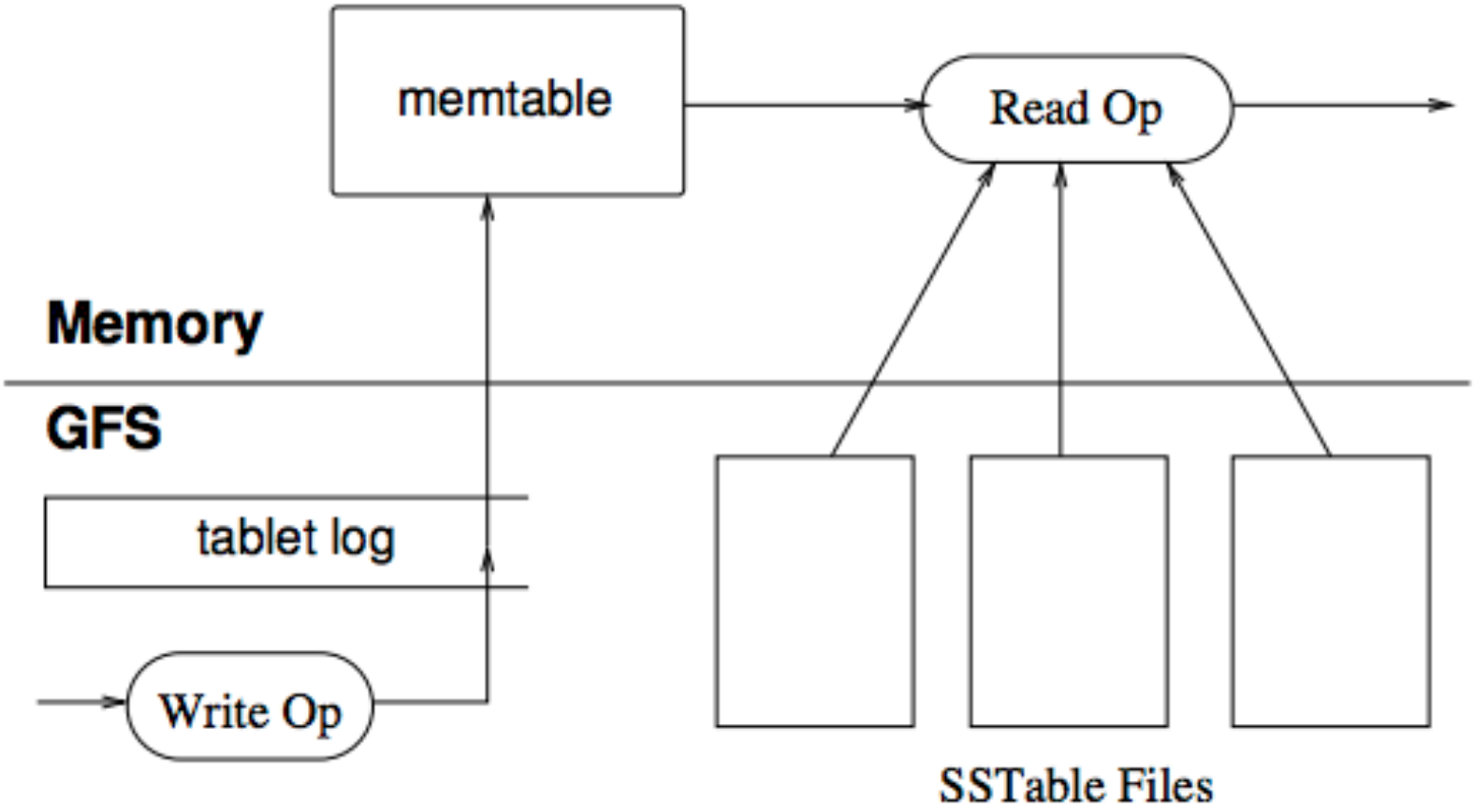
Google Bigtable: Structure and query model

- A table contains a number of rows and is broken down into tablets which each contain a subset of the rows
- Tablets are the unit of distribution and load balancing
- Rows are sorted lexicographically by row key
  - Subsequent row keys are within a tablet
  - Allows efficient range queries for small numbers of rows
- Operations: Read, write and delete items + batch writes
- Supports single-row transactions

Google Bigtable: Local persistence

- Tablet servers store updates in commit logs written in so-called SSTables
- Fresh updates are kept in memory (memtable), old updates are stored in GFS
- Minor compactions flush memtable into new SSTable
- Major compaction merge SSTables into just one new SSTable

Google Bigtable: Local persistence



Source: Bigtable paper

Cassandra Architecture and Main Features

Cassandra: a decentralized structured storage system

- **Cassandra Paper**

  Avinash Lakshman and Prashant Malik. 2010. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.* 44, 2 (April 2010), 35-40.
- URL: http://www.cs.cornell.edu/Projects/ladis2009/papers/Lakshman-ladis2009.PDF

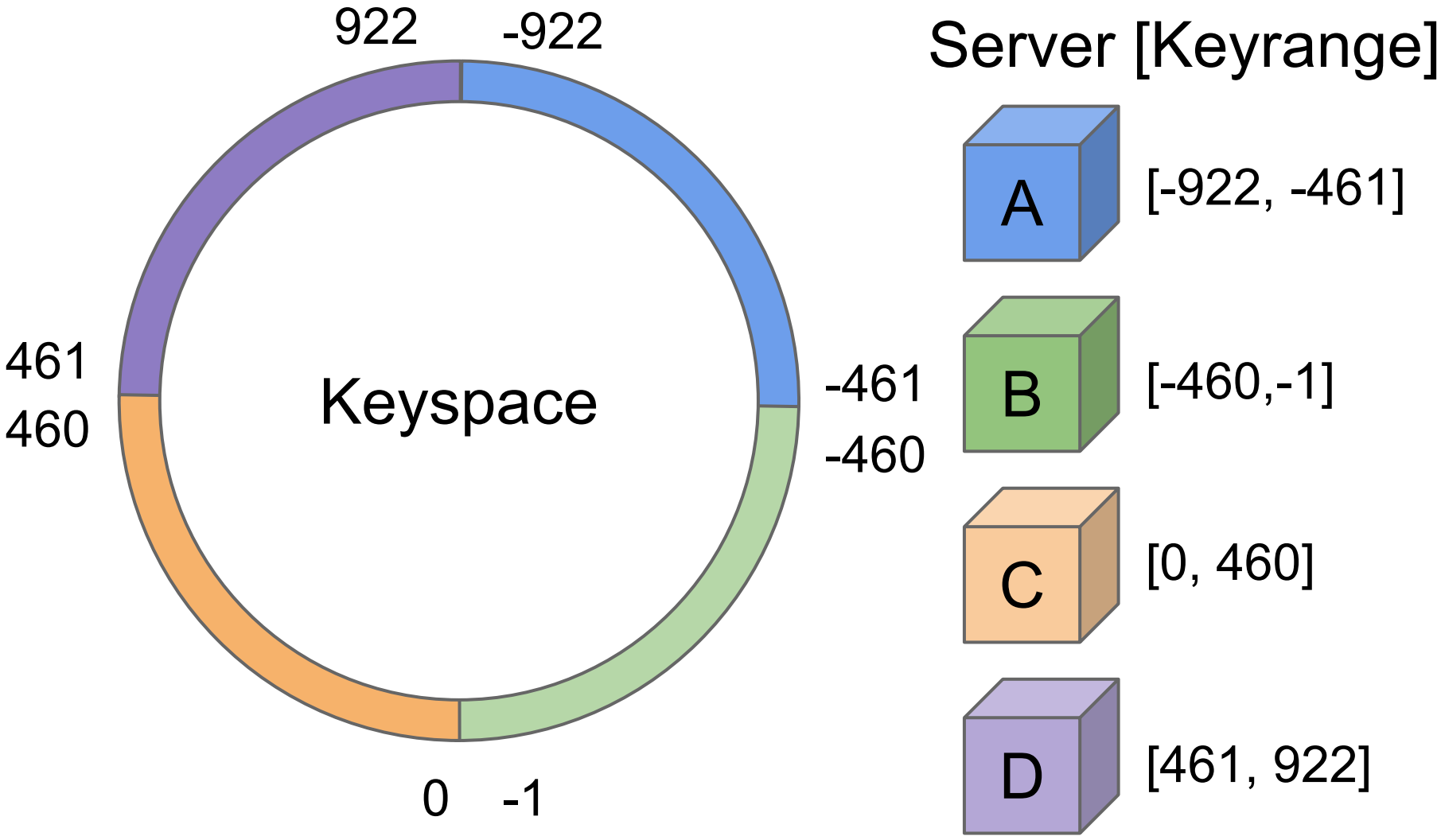
Architecture Topics

- Data Partitioning & Distribution
  - Partitioners
  - Virtual Nodes
- Data Replication
- Network Toplogy (Snitches)
- Server-to-Server Communication (Gossip)
  - Membership
  - Failure detection
- Scaling a Cluster
- Client-Server Communication
- Local Persistence

Architecture Topics

- **Data Partitioning & Distribution**
  - **Partitioners**
  - **Virtual Nodes**
- Data Replication
- Network Toplogy (Snitches)
- Server-to-Server Communication (Gossip)
  - Membership
  - Failure detection
- Scaling a Cluster
- Client-Server Communication
- Local Persistence

Data Partitioning & Distribution



Partitioners

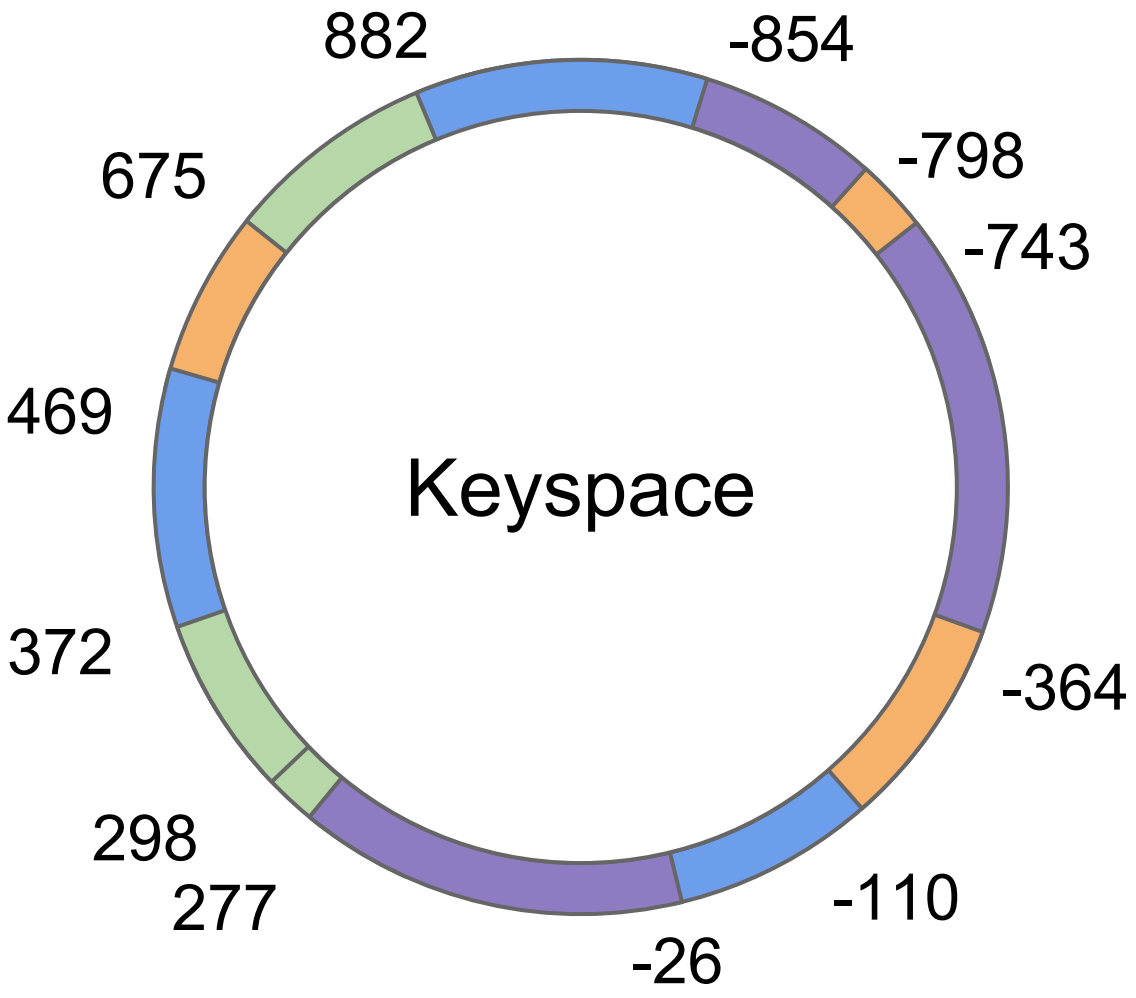
- ByteOrderedPartitioner (not recommended)
  - Plus: You can scan across lexically ordered keys
  - Minus: bad load balancing, hotspots, etc.
- RandomPartitioner (default before 1.2)
  - The RandomPartition distributes data evenly across the nodes using an MD5 hash value of the row key. The possible range of hash values is from 0 to 2^{127} -1.
- Murmur3Partitioner (default since 1.2)
  - The Murmur3Partitioner uses the MurmurHash function. This hashing function creates a 64-bit hash value of the row key. The possible range of hash values is from -2^{63} to $+2^{63}$.

Source: http://www.datastax.com/docs/1.2/cluster_architecture/partitioners

Data Partitioning & Distribution

- Virtual Nodes (Vnodes)
- Since Cassandra 1.2: Virtual Nodes for
  - better load balancing
  - easier scaling with differently sized servers

Virtual Nodes
Example with num_tokens: 3



Server [Keyrange]

A
[882, -854]
[-110, --26]
[372, 469]

B
[675, 882]
[227, 298]
[298, 372]

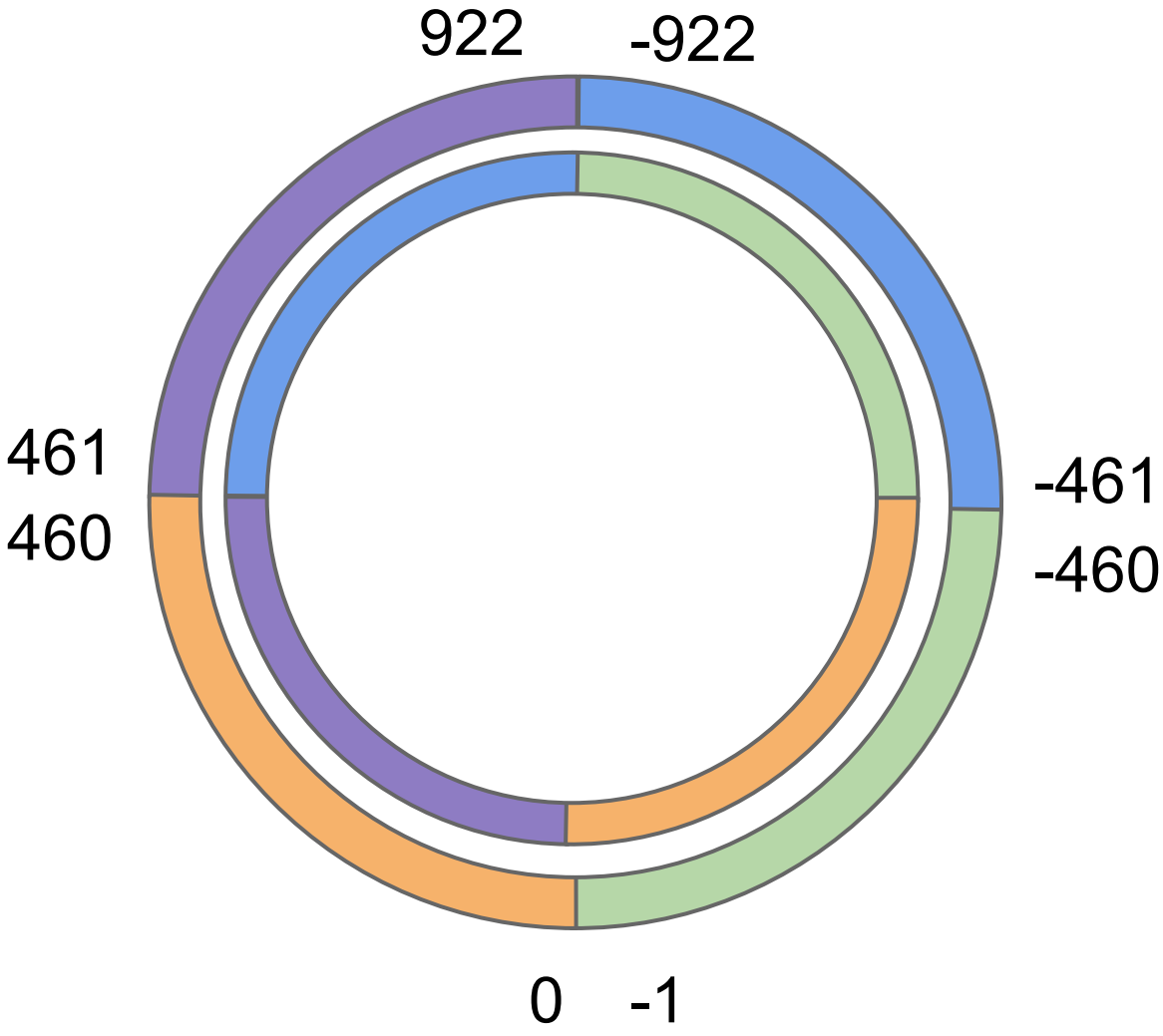
C
[-798, -743]
[-364, -110]
[469, -675]

D
[-854, -798]
[-743, -364]
[-26, 277]

Architecture Topics

- Data Partitioning & Distribution
  - Partitioners
  - Virtual Nodes
- **Data Replication**
- Network Toplogy (Snitches)
- Server-to-Server Communication (Gossip)
  - Membership
  - Failure detection
- Scaling a Cluster
- Client-Server Communication
- Local Persistence

Data Replication



Server [Keyrange]

A [461, -461]

B [-922,-1]

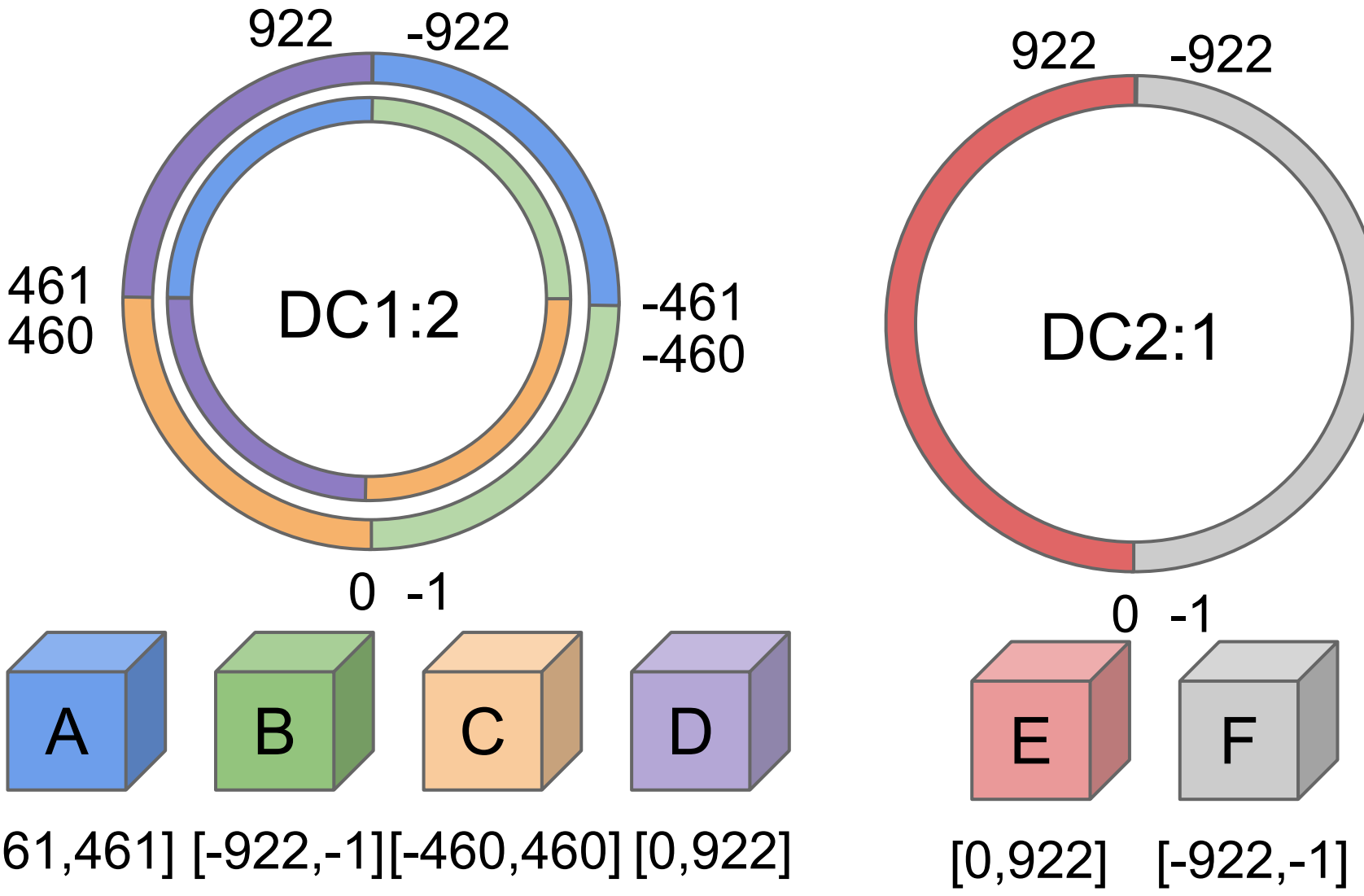
C [-460, 460]

D [0, 922]

Data Replication

- Replication for high availability and data durability
  - **Replication factor** N: Each row is replicated at N nodes.
  - Each row key k is assigned to a coordinator node.
  - The coordinator is responsible for replicating the rows within its key range.

Multi-DC Data Replication



Architecture Topics

- Data Partitioning & Distribution
  - Partitioners
  - Virtual Nodes
- Data Replication
- **Network Toplogy (Snitches)**
- Server-to-Server Communication (Gossip)
  - Membership
  - Failure detection
- Scaling a Cluster
- Client-Server Communication
- Local Persistence

Network Topology (Snitches)

- Selected Snitches
  - SimpleSnitch (default)
  - RackInferringSnitch
  - PropertyFileSnitch
  - GossipingPropertyFileSnitch

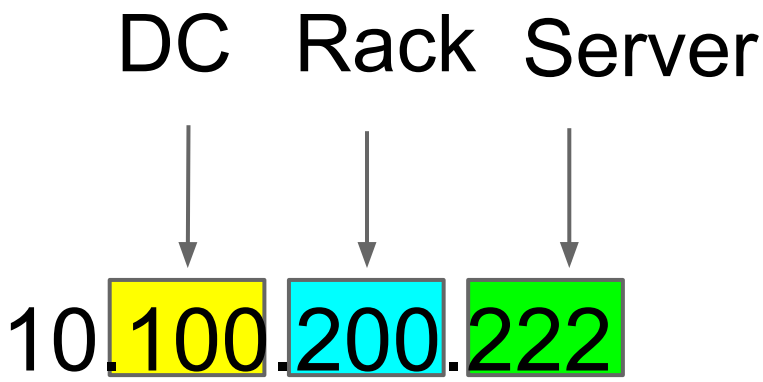
- Set the `endpoint_snitch` property in *cassandra.yaml*

SimpleSnitch

- SimpleSnitch does not recognize data center or rack information
- Only useful for small single-DC deployments

RackInferringSnitch

- Assumes the network topology from the node's IP address



PropertyFileSnitch

- Uses *conf/cassandra-topology.properties* file to infer data center and rack information
- Useful if cluster layout is not matched by IP addresses or if you have complex grouping requirements
- Example properties file:

```
# Data Center One
175.56.12.105=DC1:RAC1
120.53.24.101=DC1:RAC2

# Data Center Two
110.56.12.120=DC2:RAC1
50.17.10.203=DC2:RAC2
```

GossipingPropertyFileSnitch

- Each node sets its own data center and rack info via conf/cassandra-rackdc.rpoperties file.
- The info is propagated to other nodes via gossip. Fits nicely the P2P style of Cassandra.
- Example properties file:

```
dc=DC1
rack=RAC1
```

Dynamic Snitching

- Dynamic snitching avoids routing requests to badly performing nodes.
- Properties in the *cassandra.yaml*

```
dynamic_snitch_update_interval_in_ms: 100
dynamic_snitch_reset_interval_in_ms: 600000
dynamic_snitch_badness_threshold: 0.1
```

Architecture Topics

- Data Partitioning & Distribution
  - Partitioners
  - Virtual Nodes
- Data Replication
- Network Toplogy (Snitches)
- **Server-to-Server Communication (Gossip)**
  - **Membership**
  - **Failure detection**
- Scaling a Cluster
- Client-Server Communication
- Local Persistence

Server-to-Server Communication: Gossip

- Cassandra uses a gossip protocol to exchange information between servers in a cluster in a peer-to-peer fashion
  - The gossip process runs every second on each Cassandra server
  - Each server sends its state in a message to other servers in the cluster
  - Each gossip message has a version. Old gossip state information on a server is overwritten.

Server-to-Server Communication: Seeds

- Seeds
  - The list of `seeds` addresses which in the *cassandra.yaml* file **is only used during initial bootstrapping of a new server in the cluster.**
  - The bootstrapping server establishes gossip communication with the servers in the `seeds` list.
- You should use the same *seeds* list on all servers to prevent split-brain partitions in gossip communication.
- In a Multi-DC setup, the seeds list must include at least one server from each DC.

Server-to-Server Communication

- Delete gossip state on a server
  - You can delete the gossip state of server by adding the following in your *cassandra-env.sh* file:

    `-Dcassandra.load_ring_state=false`
- This is necessary in certain situations when you restart one or more servers, such as
  - You restart a server after its IP address has been changed
  - You restart all servers with a new `cluster_name`

Server-to-Server Communication: Failure Detection

- Cassandra implements a Gossip-based accrual failure detector that **adapts the time interval based on historic latency data**.
  - Every Cassandra node maintains a sliding window of inter-arrival times of Gossip messages.
  - The Cassandra failure detector assumes an exponential distribution of inter-arrival times.
  - The failure detector can be configured with the *cassandra.yaml* parameter `phi_convict_threshold`
  - Tip: You can make the failure detector less sensitive to latency variability, for example during times of network congestion or in Multi-DC setups, by increasing the `phi_convict_threshold` value.

Heartbeat Failure Detection

- Heartbeat failure detector

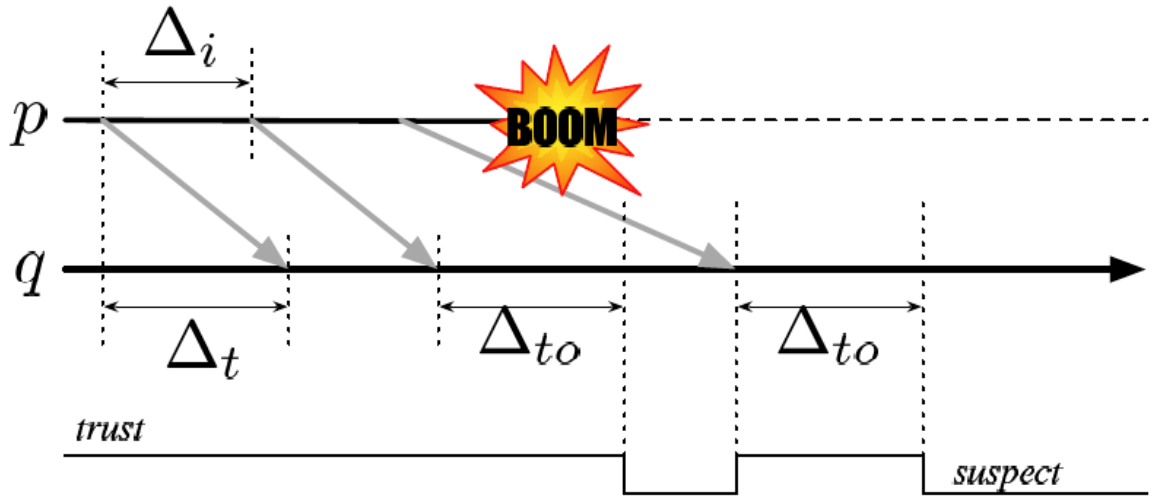
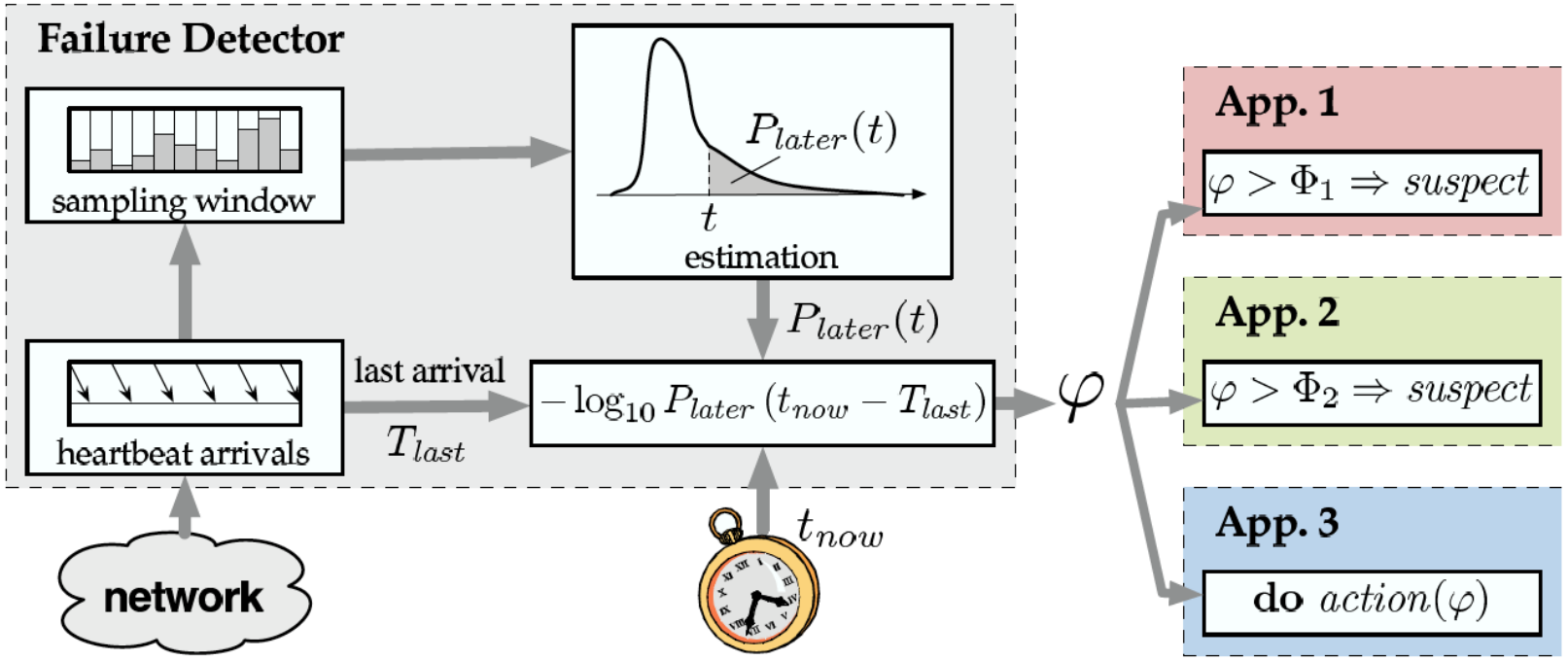


Fig. 1. Heartbeat failure detection and its main parameters.

Naohiro Hayashibara, Xavier Defago, Rami Yared, and Takuya Katayama. 2004. The Phi Accrual Failure Detector. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems* (SRDS '04). IEEE Computer Society, Washington, DC, USA, 66-78.

Accrual Failure Detection

- Accrual failure detector



Naohiro Hayashibara, Xavier Defago, Rami Yared, and Takuya Katayama. 2004. The Phi Accrual Failure Detector. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems* (SRDS '04). IEEE Computer Society, Washington, DC, USA, 66-78.

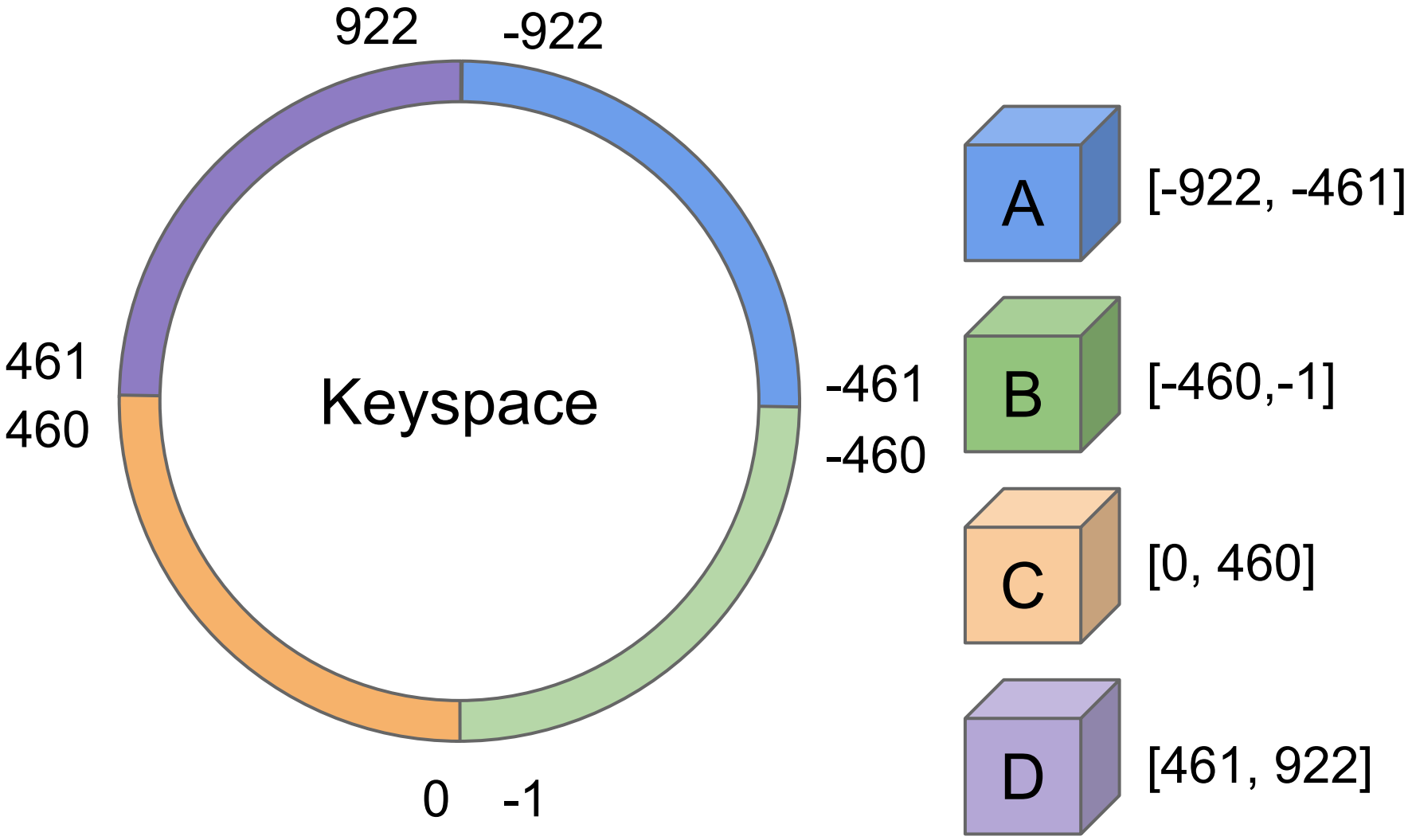
Architecture Topics

- Data Partitioning & Distribution
  - Partitioners
  - Virtual Nodes
- Data Replication
- Network Toplogy (Snitches)
- Server-to-Server Communication (Gossip)
  - Membership
  - Failure detection
- **Scaling a Cluster**
- Client-Server Communication
- Local Persistence

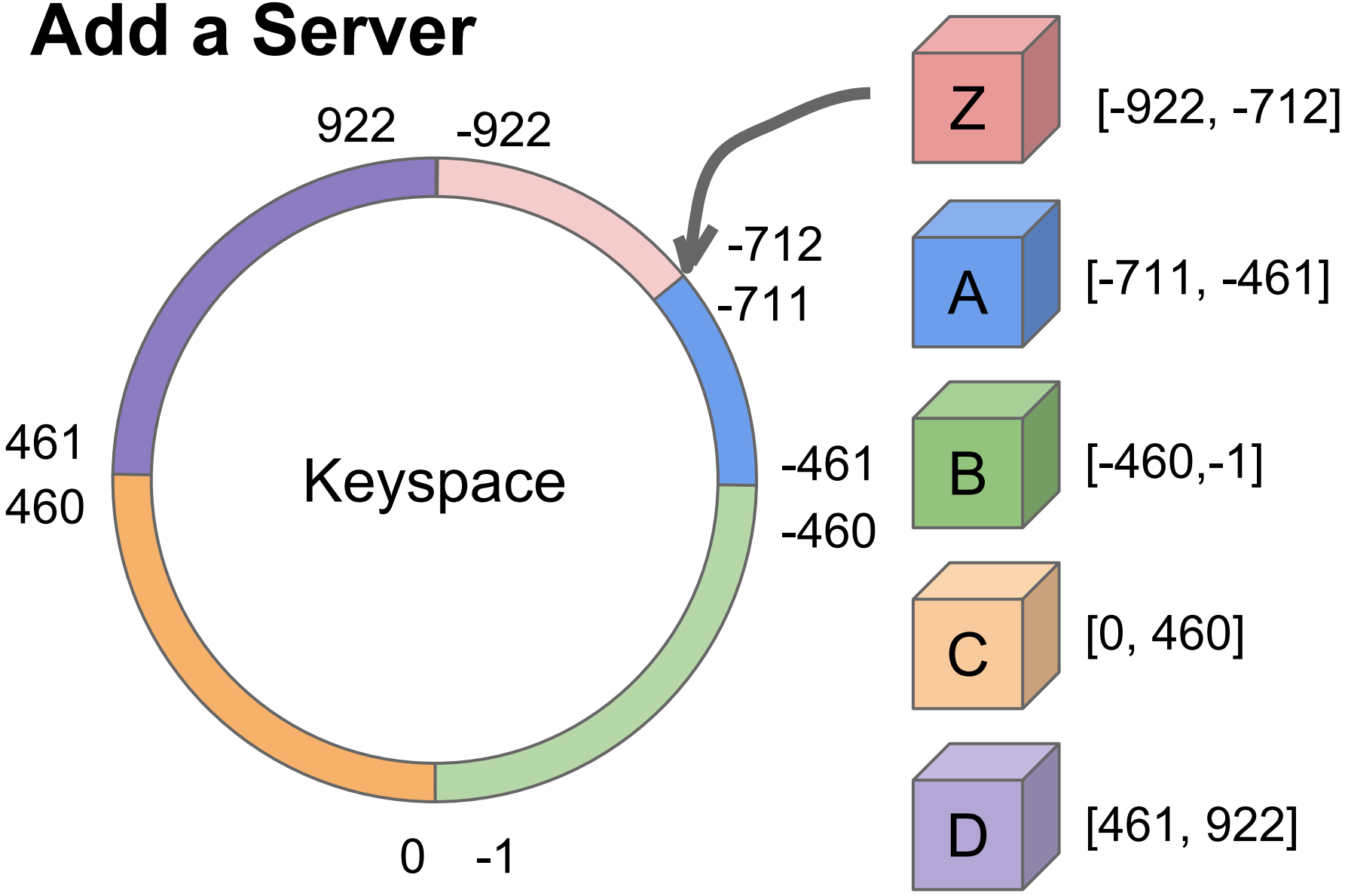
Scaling a Cluster

- Bootstrapping managed via command line admin tool
- A node starts for the first time
  - Chose position on the ring (via token)
  - Join cluster: connect with seed node(s) and start data streaming from neighboring node.
  - Bootstrapping/joining phase is completed when all data for the newly split keyrange has been streamed to the new node.
- Basically the same process for a cluster with **Vnodes**, however, a server with Vnodes has multiple positions in the ring and streams from multiple neighboring servers.

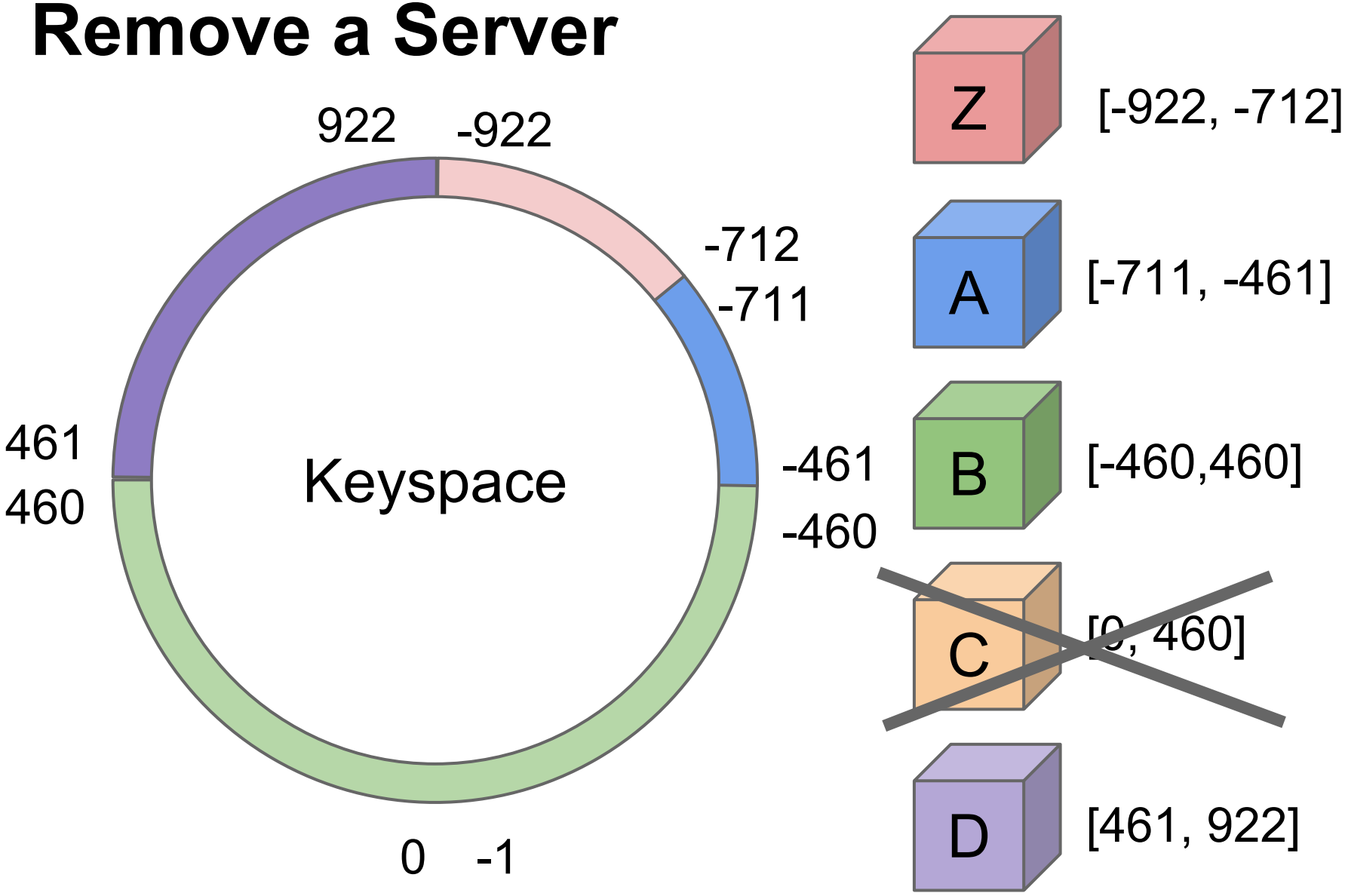
Add Server


922
-922
461
460
Keyspace
-461
-460
0
-1
A
[-922, -461]
B
[-460,-1]
C
[0, 460]
D
[461, 922]

Add a Server


922
-922
-712
-711
461
460
Keyspace
-461
-460
0
-1
Z
[-922, -712]
A
[-711, -461]
B
[-460,-1]
C
[0, 460]
D
[461, 922]

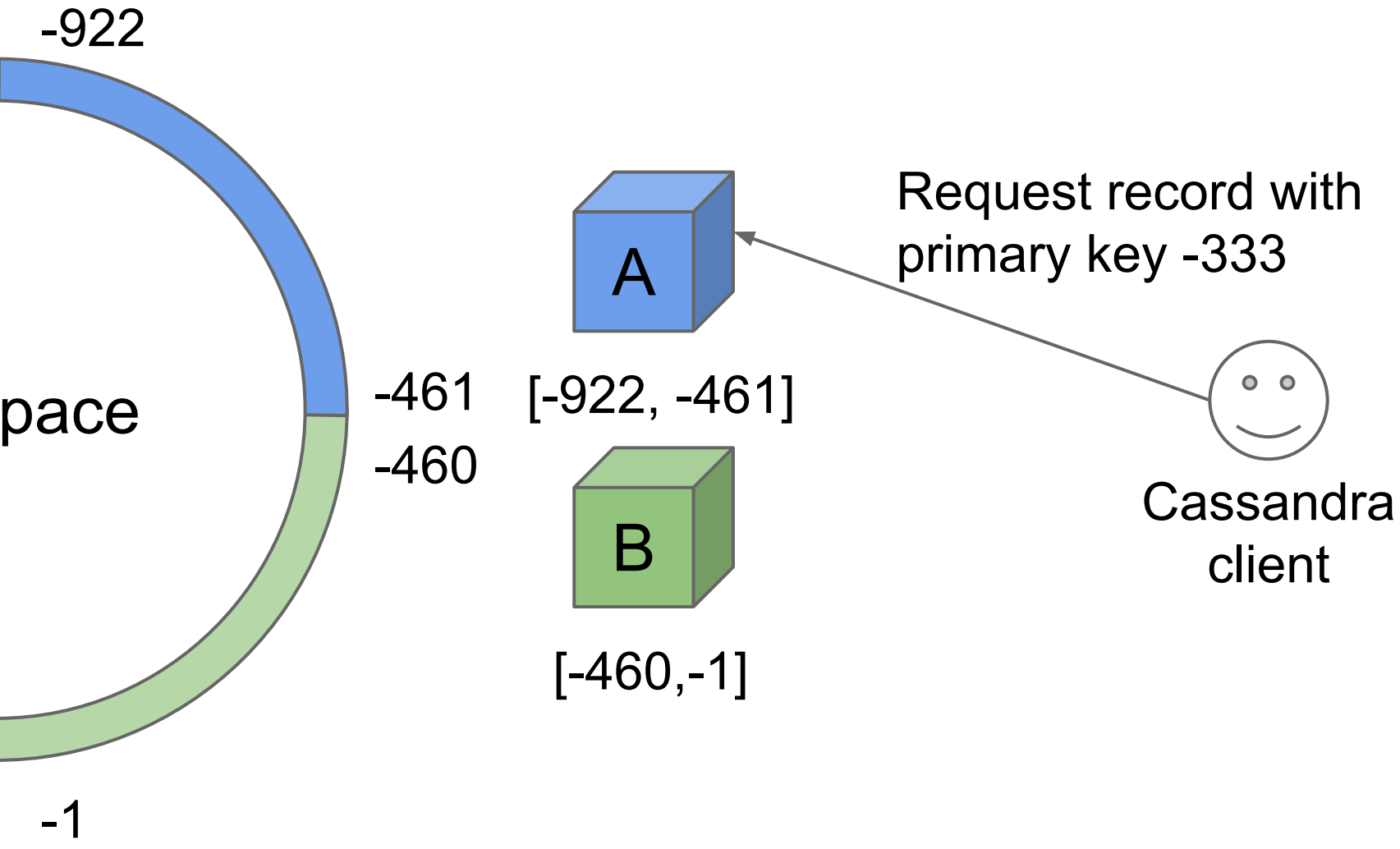
Remove a Server



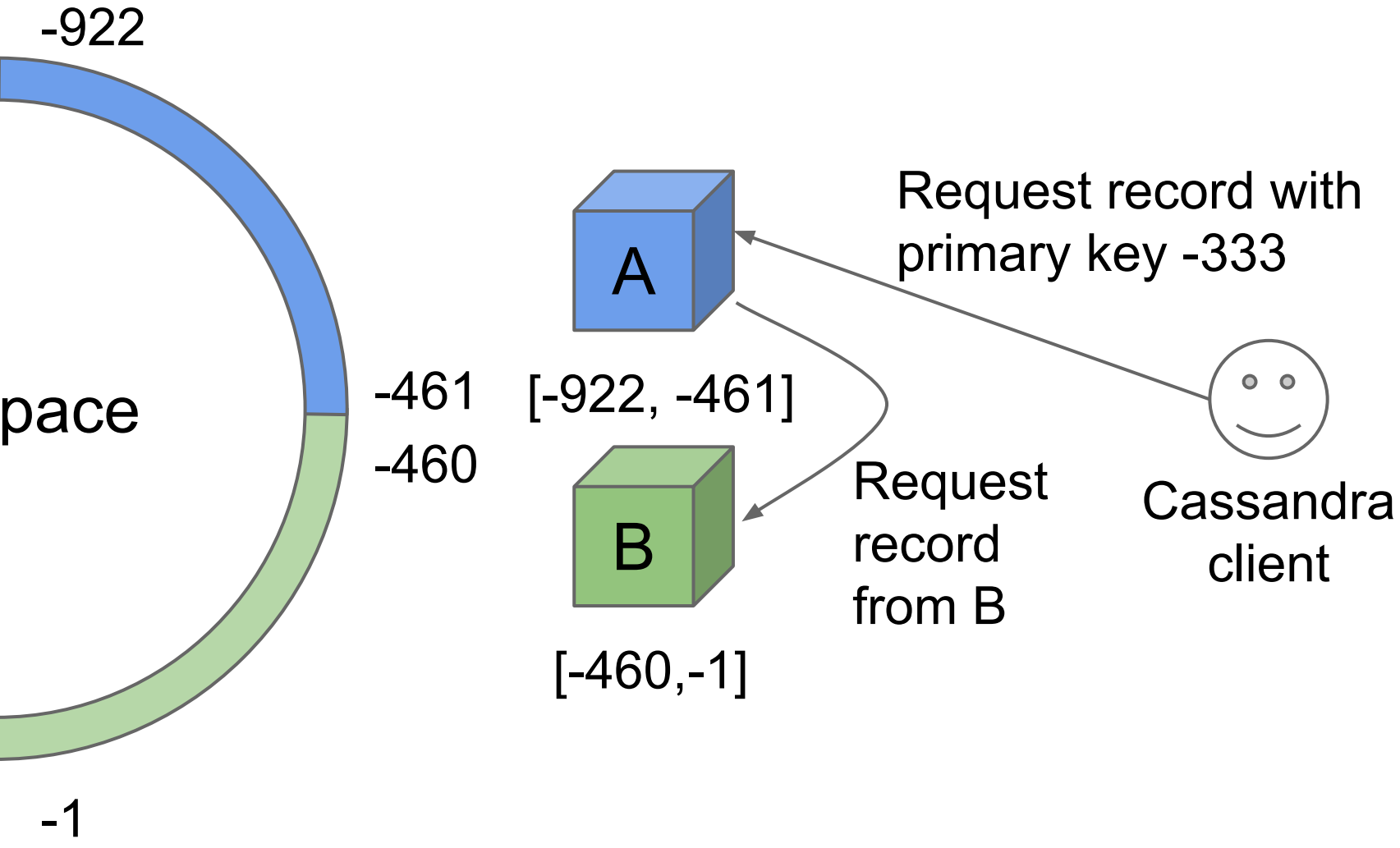
Architecture Topics

- Data Partitioning & Distribution
  - Partitioners
  - Virtual Nodes
- Data Replication
- Network Toplogy (Snitches)
- Server-to-Server Communication (Gossip)
  - Membership
  - Failure detection
- Scaling a Cluster
- **Client-Server Communication**
- Local Persistence

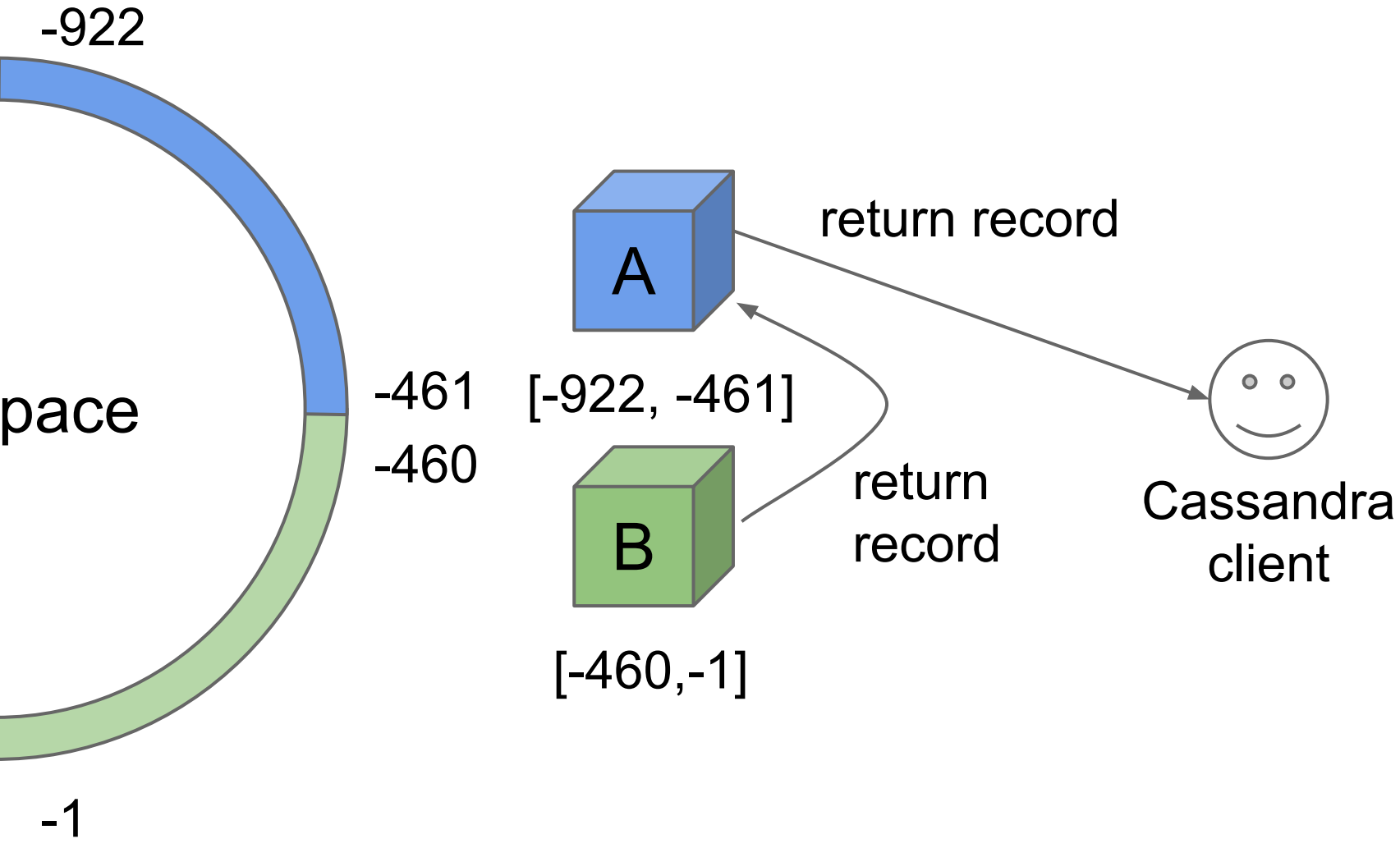
Client-Server Communication



Client-Server Communication



Client-Server Communication



Architecture Topics

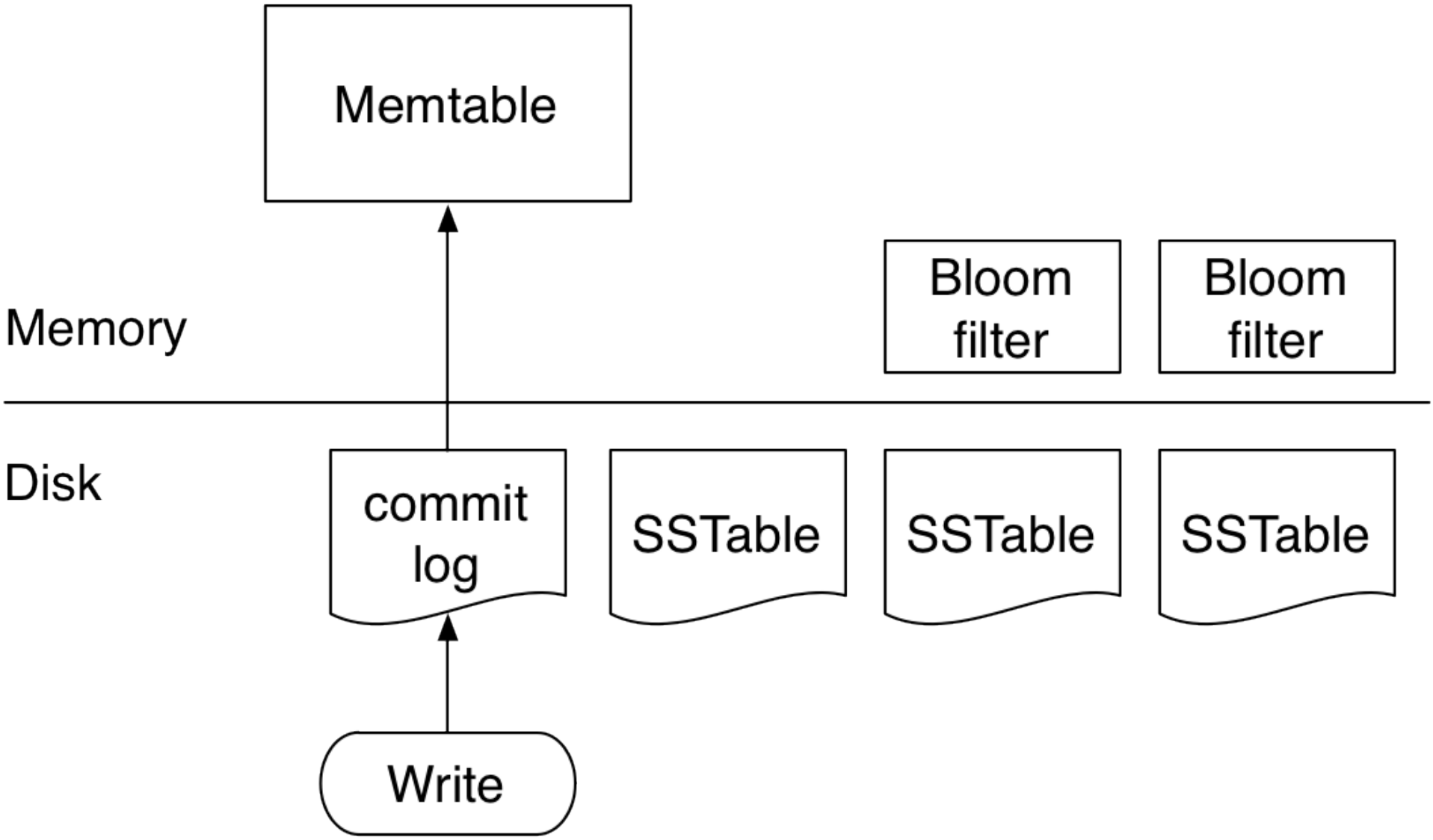
- Data Partitioning & Distribution
  - Partitioners
  - Virtual Nodes
- Data Replication
- Network Toplogy (Snitches)
- Server-to-Server Communication (Gossip)
  - Membership
  - Failure detection
- Scaling a Cluster
- Client-Server Communication
- **Local Persistence**

Local Persistence - Write

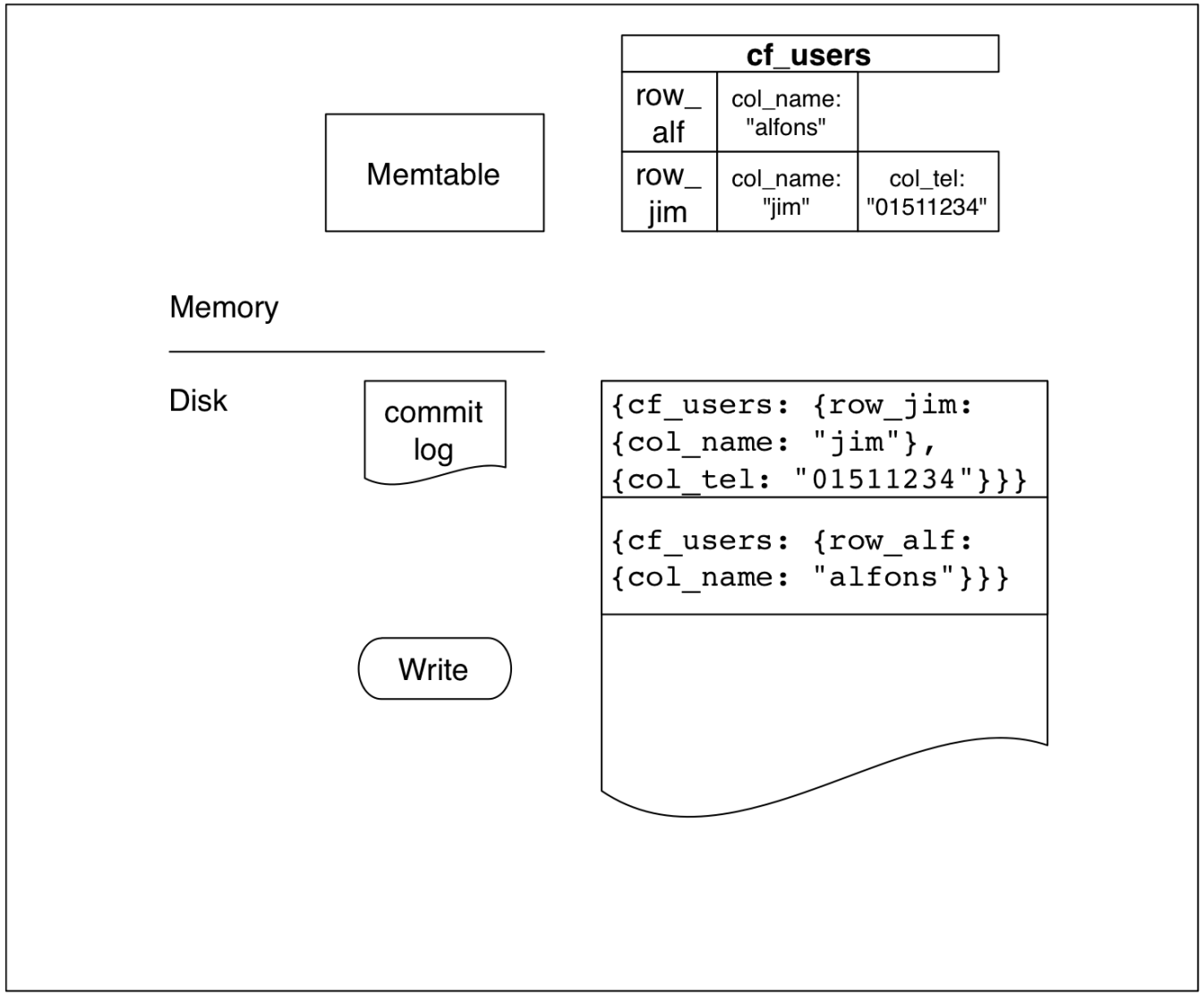
- Write
  1. Append to commit log for durability (recoverability)
  2. Update of in-memory, per-column-family Memtable

- If Memtable crosses a threshold
  1. Sequential write to disk (SSTable).
  2. Merge SSTables from time to time (compactions)

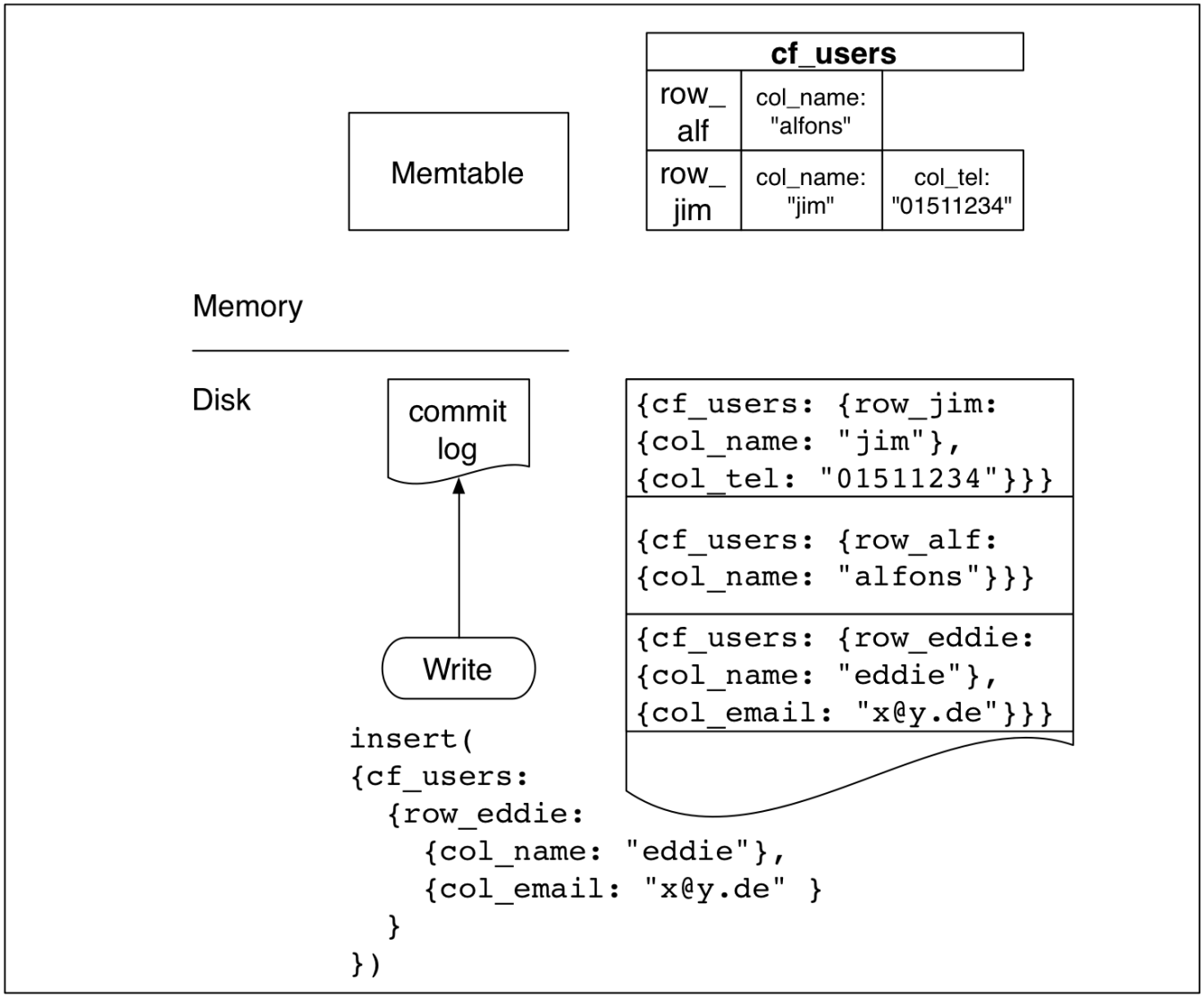
Local Persistence - Write (2)



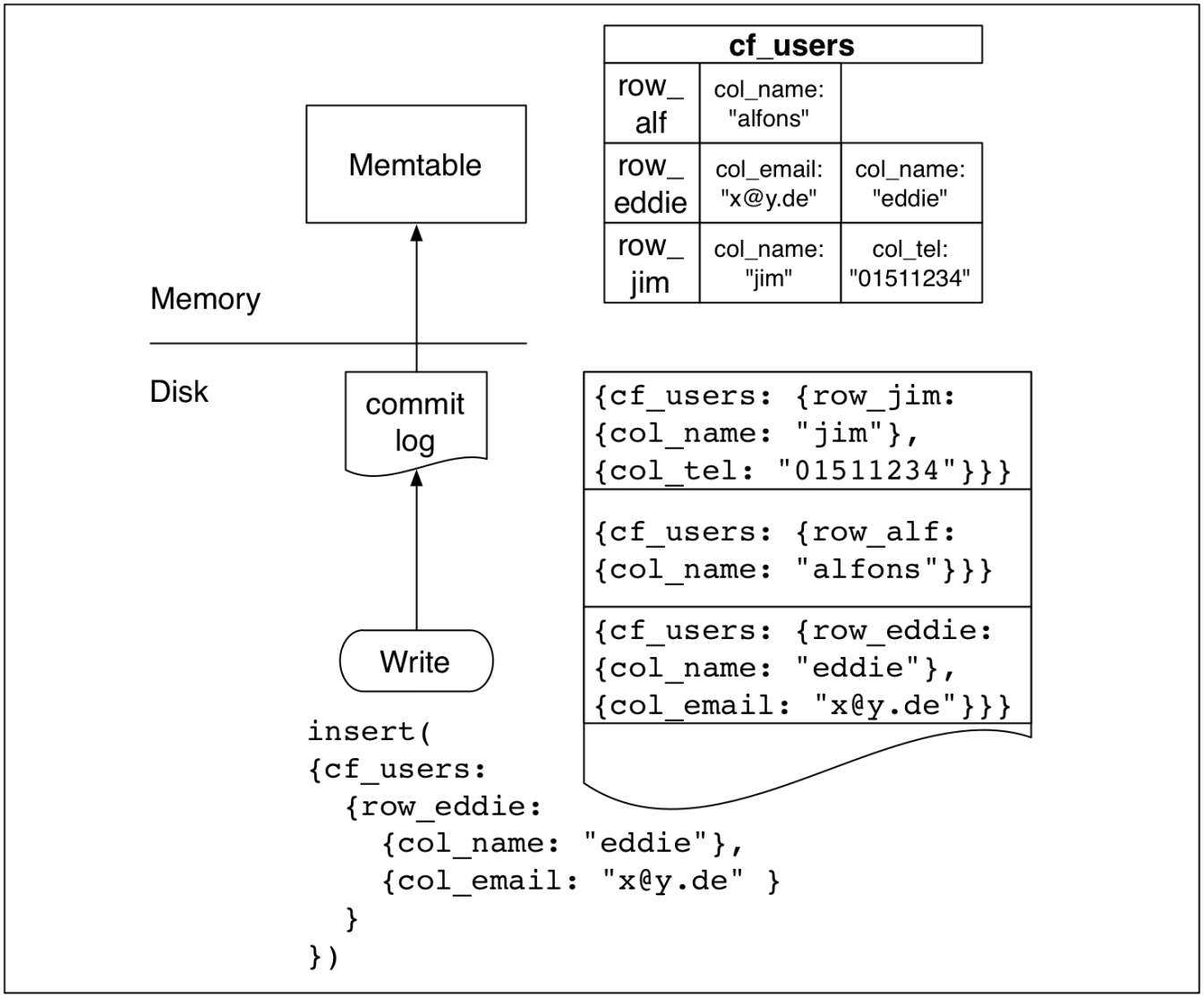
Local Persistence - Write Example (1)



Local Persistence - Write Example (2)



Local Persistence - Write Example (3)



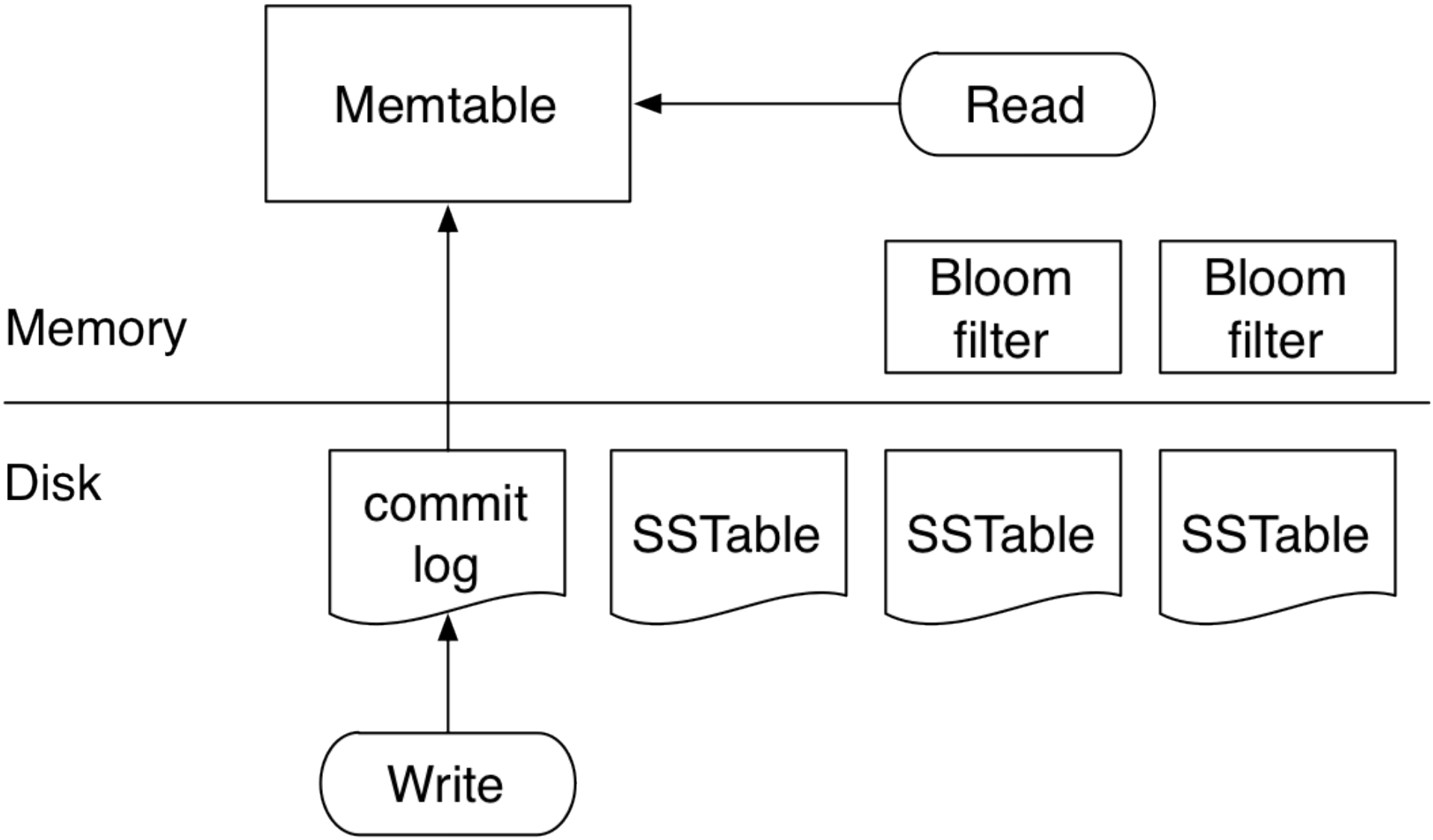
Local Persistence - CommitLog *cassandra.yaml*

```
# commitlog_sync may be either "periodic" or "batch."
# When in batch mode, Cassandra won't ack writes until the
# commit log has been fsynced to disk.  It will wait up to
# commitlog_sync_batch_window_in_ms milliseconds for other
# writes, before performing the sync.
# commitlog_sync: batch
# commitlog_sync_batch_window_in_ms: 50
# the other option is "periodic" where writes may be acked
# immediately and the CommitLog is simply synced every
# commitlog_sync_period_in_ms milliseconds.
commitlog_sync: periodic
commitlog_sync_period_in_ms: 10000
```

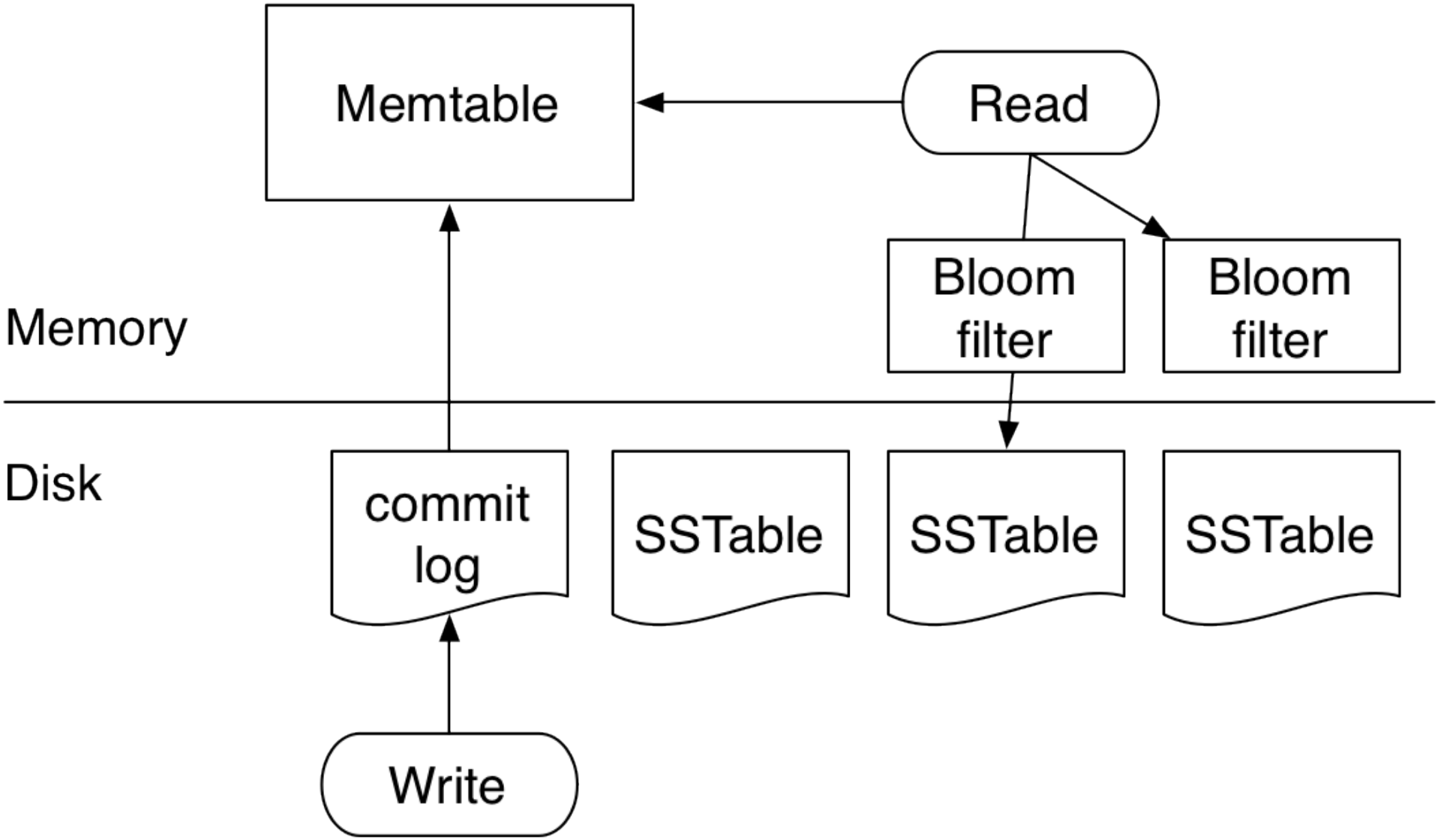
Local Persistence - Read

- Read
  - Query in-memory Memtable
  - Check in-memory bloom filter
    - Used to prevent unnecessary disk access.
    - A bloom filter summarizes the keys in a file.
    - False Positives are possible
  - Check column index to jump to the columns on disk as fast as possible.
    - Index for every 256K chunk.

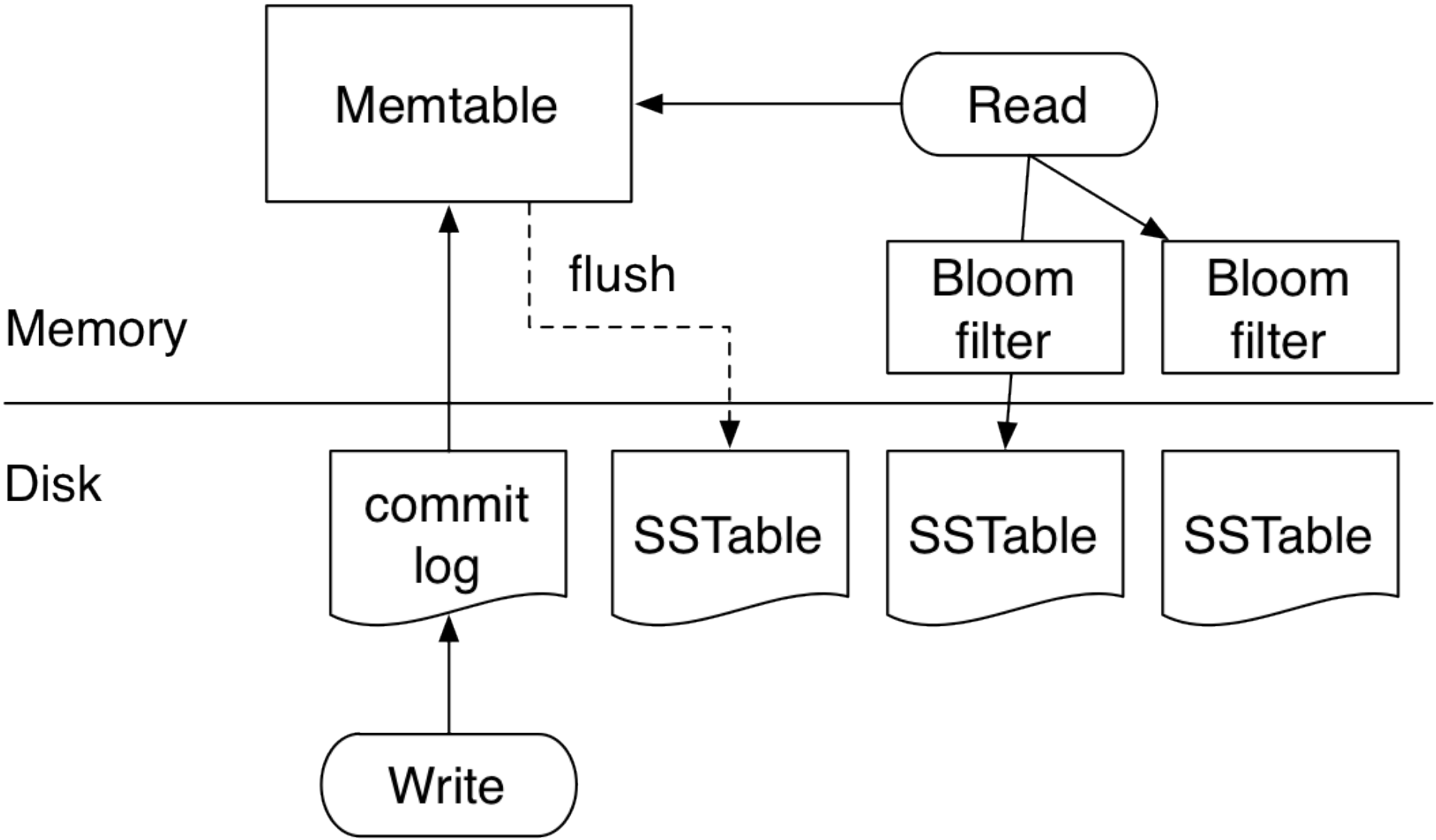
Local Persistence - Read (2)



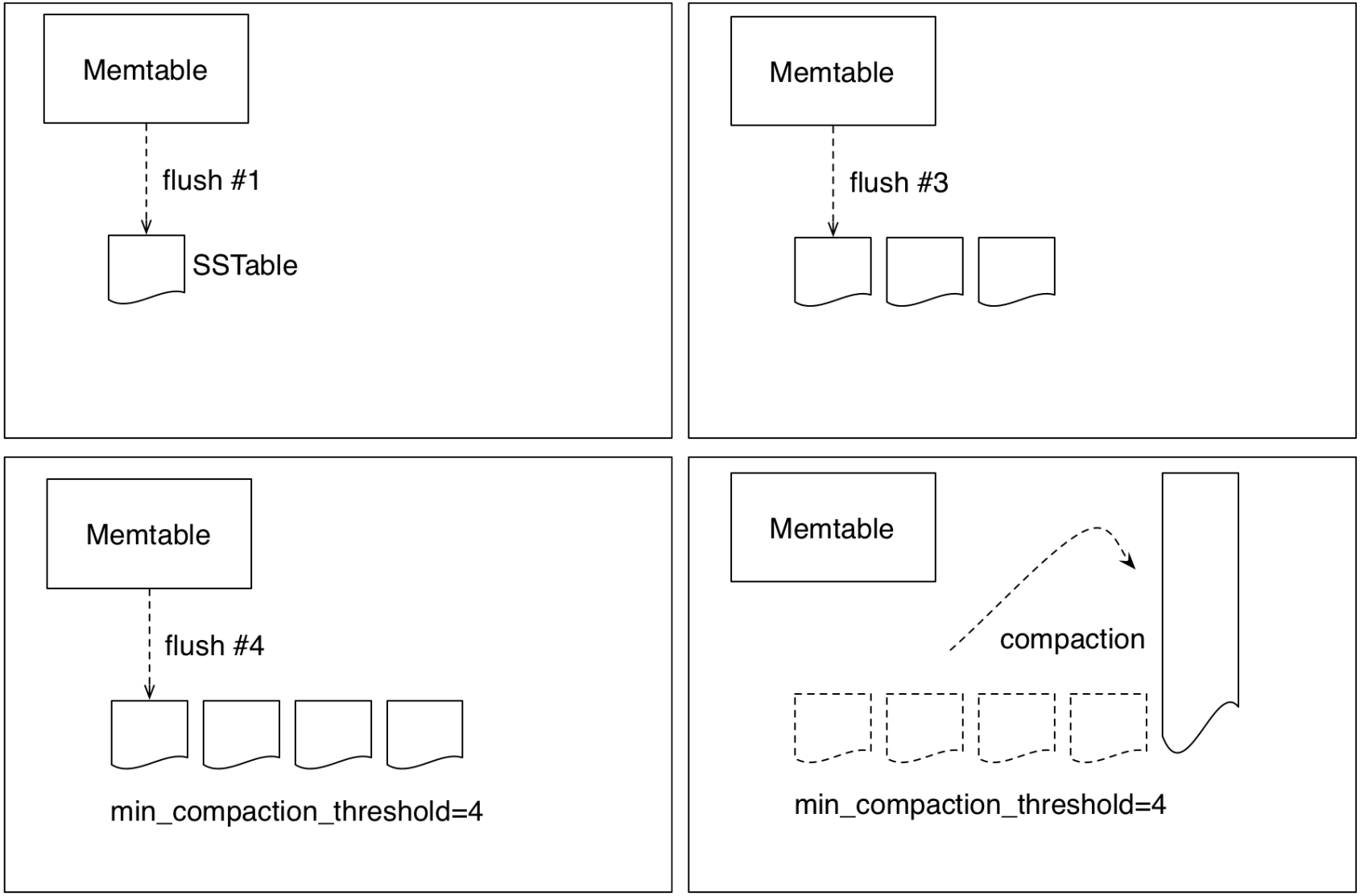
Local Persistence - Read (3)



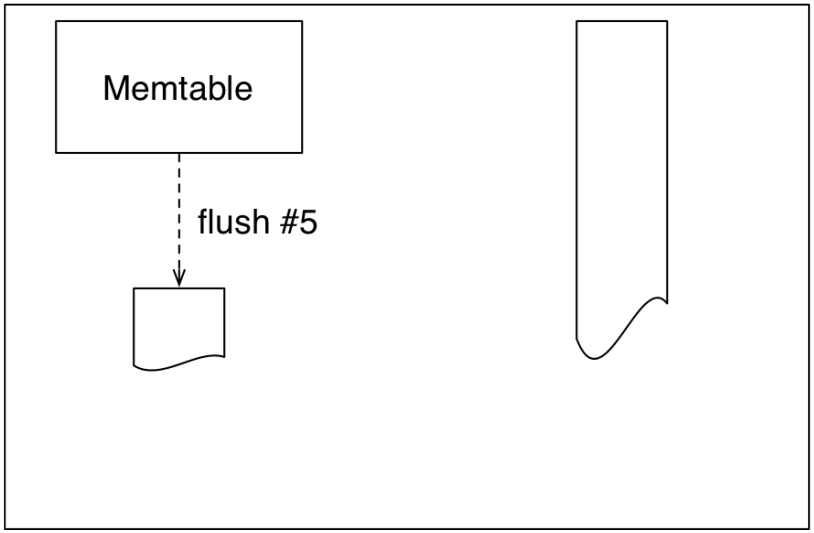
Local Persistence - Read (4)

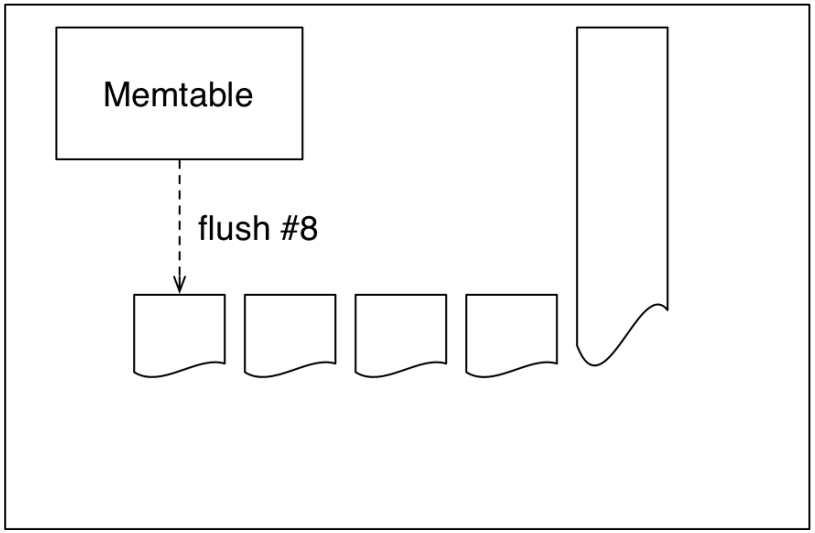


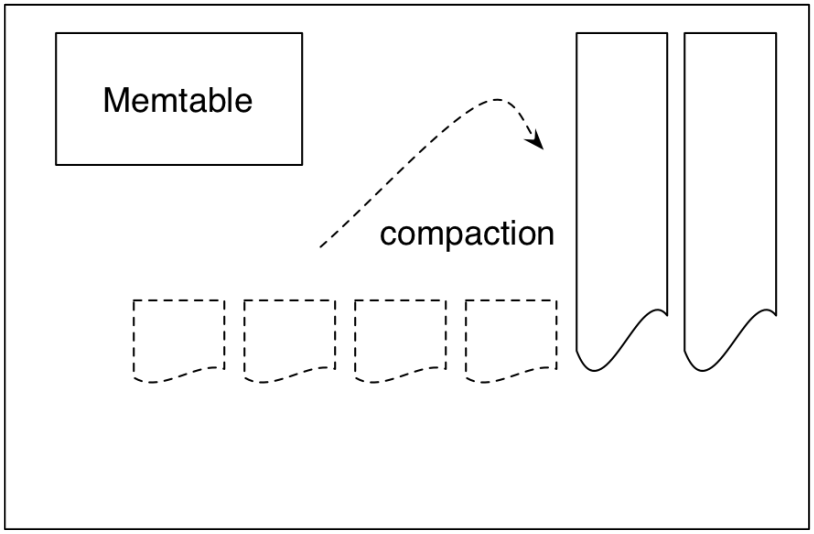
Compactions in Cassandra



Compactions in Cassandra (2)

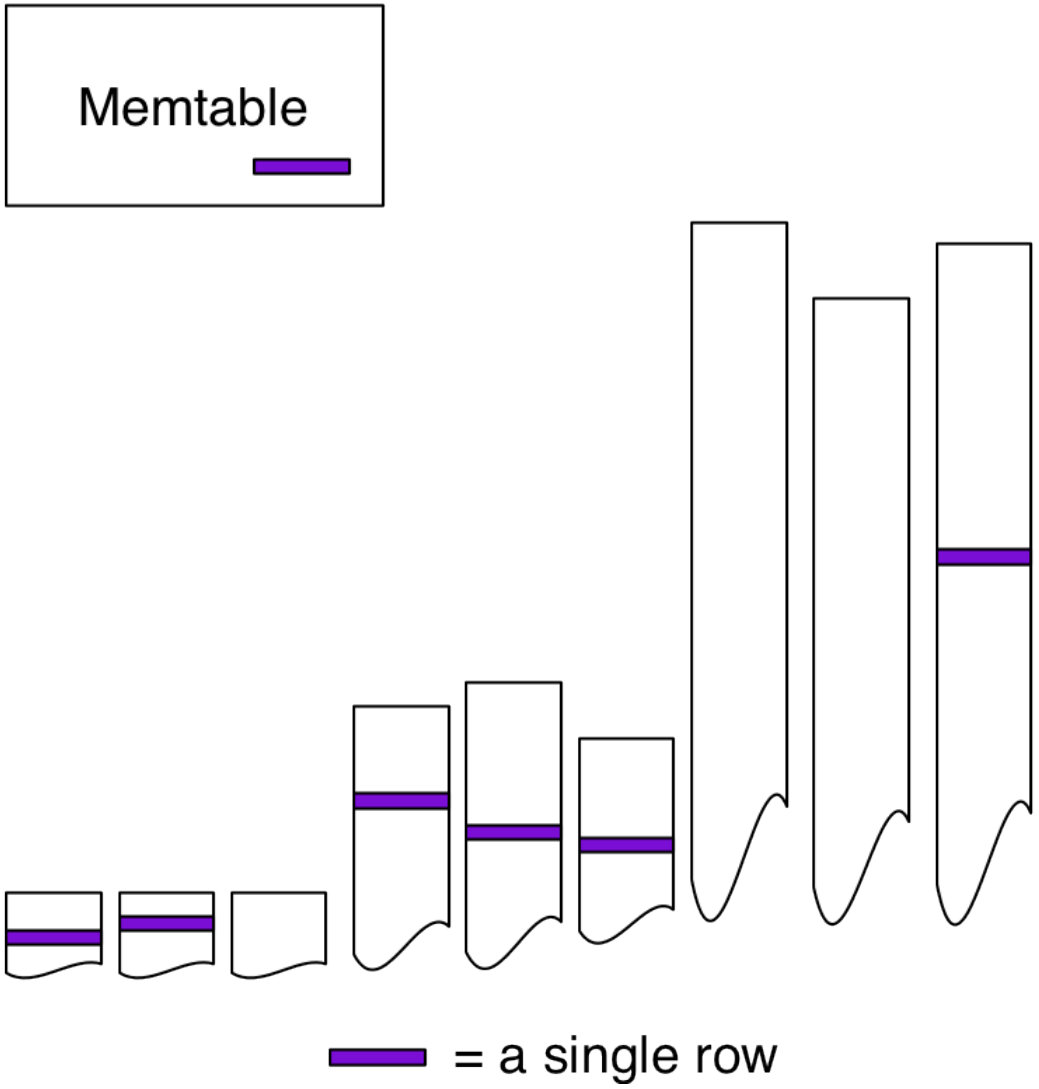




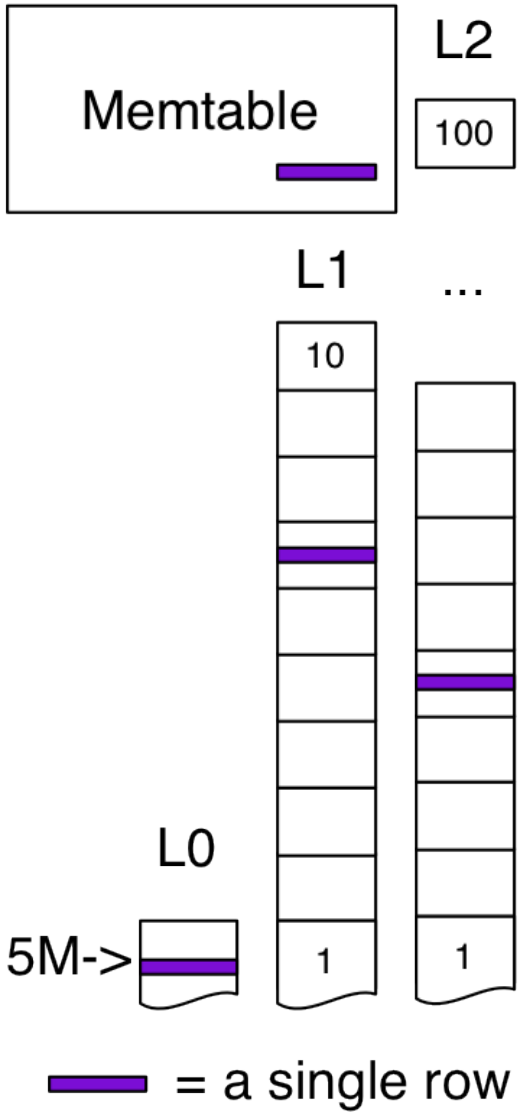


...

Size-Tiered Compaction Strategy



Leveled Compaction Strategy



Cassandra: Summary

- Scalability: Peer-to-Peer architecture
- High availability:
  - replication
  - quorum-based replica control
  - failure detection and recovery
- High performance (particularly for WRITE operations): Bigtable-style storage engine
  - All writes to disk are sequential. Files are written once (immutable) and not updated hereafter