Cassandra

Structured Storage System over a P2P Network

Avinash Lakshman, Prashant Malik

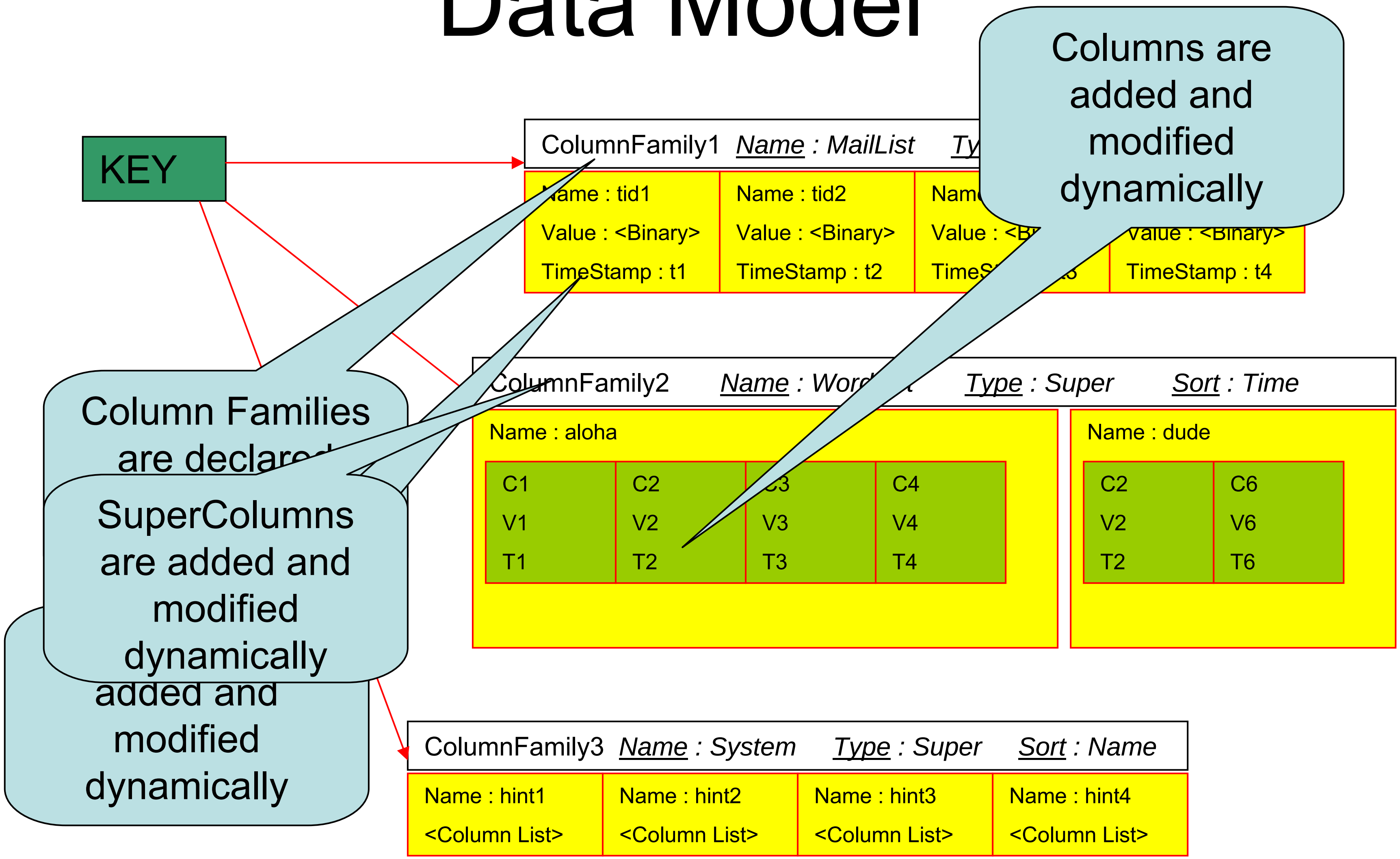
Why Cassandra?

- Lots of data
  - Copies of messages, reverse indices of messages, per user data.
- Many incoming requests resulting in a lot of random reads and random writes.
- No existing production ready solutions in the market meet these requirements.

Design Goals

- High availability
- Eventual consistency
  - trade-off strong consistency in favor of high availability
- Incremental scalability
- Optimistic Replication
- "Knobs" to tune tradeoffs between consistency, durability and latency
- Low total cost of ownership
- Minimal administration

Data Model

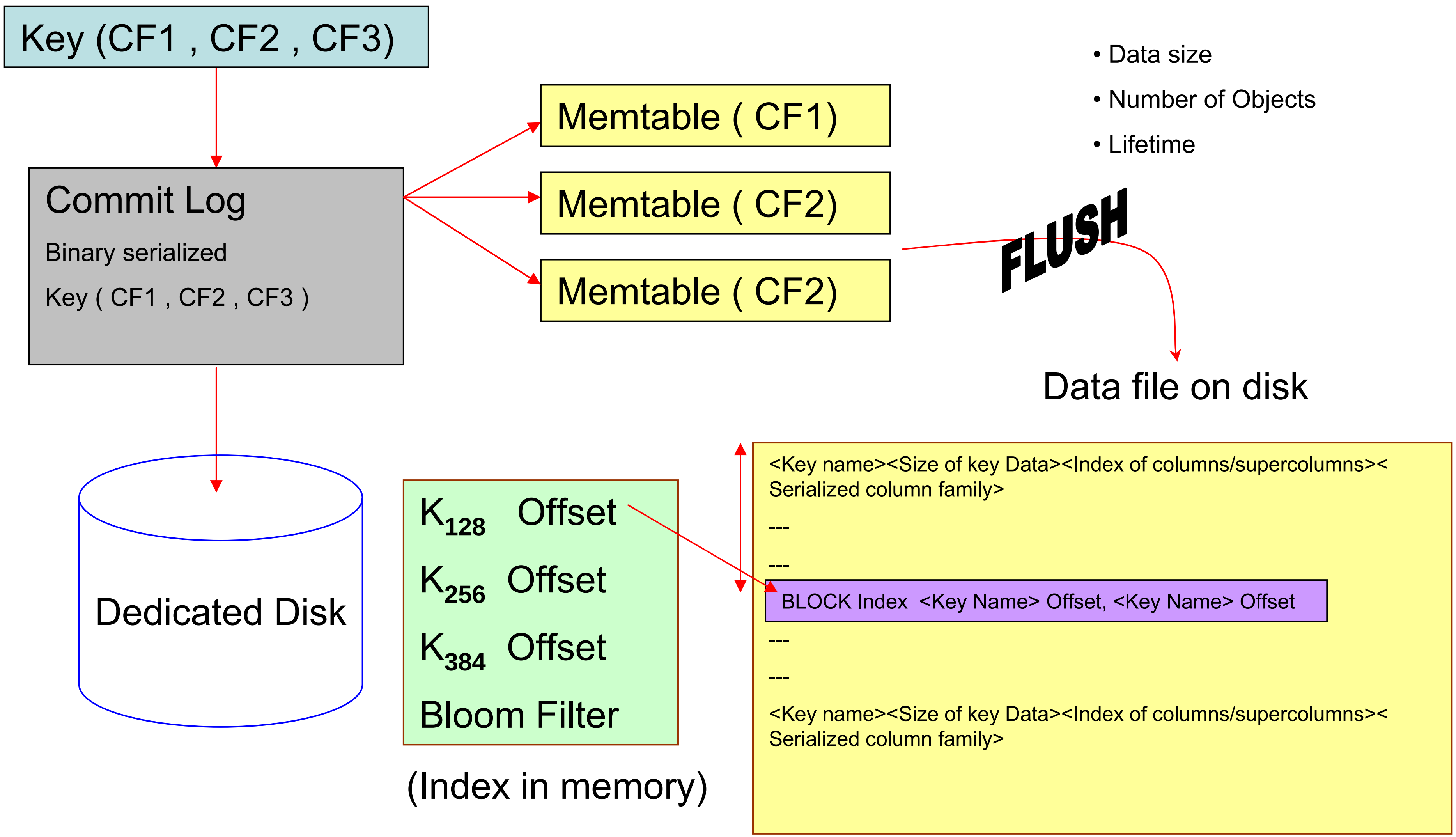


KEY
ColumnFamily1 Name : MailList Ty
Name : tid1
Value : <Binary>
TimeStamp : t1
Name : tid2
Value : <Binary>
TimeStamp : t2
Value : <Binary>
TimeStamp : t4
Columns are added and modified dynamically
ColumnFamily2 Name : Word Type : Super Sort : Time
Name : aloha
C1 V1 T1
C2 V2 T2
V3 T3
C4 V4 T4
Name : dude
C2 V2 T2
C6 V6 T6
Column Families
SuperColumns are added and modified dynamically
added and modified dynamically
ColumnFamily3 Name : System Type : Super Sort : Name
Name : hint1
<Column List>
Name : hint2
<Column List>
Name : hint3
<Column List>
Name : hint4
<Column List>

Write Operations

- A client issues a write request to a random node in the Cassandra cluster.
- The “Partitioner” determines the nodes responsible for the data.
- Locally, write operations are logged and then applied to an in-memory version.
- Commit log is stored on a dedicated disk local to the machine.

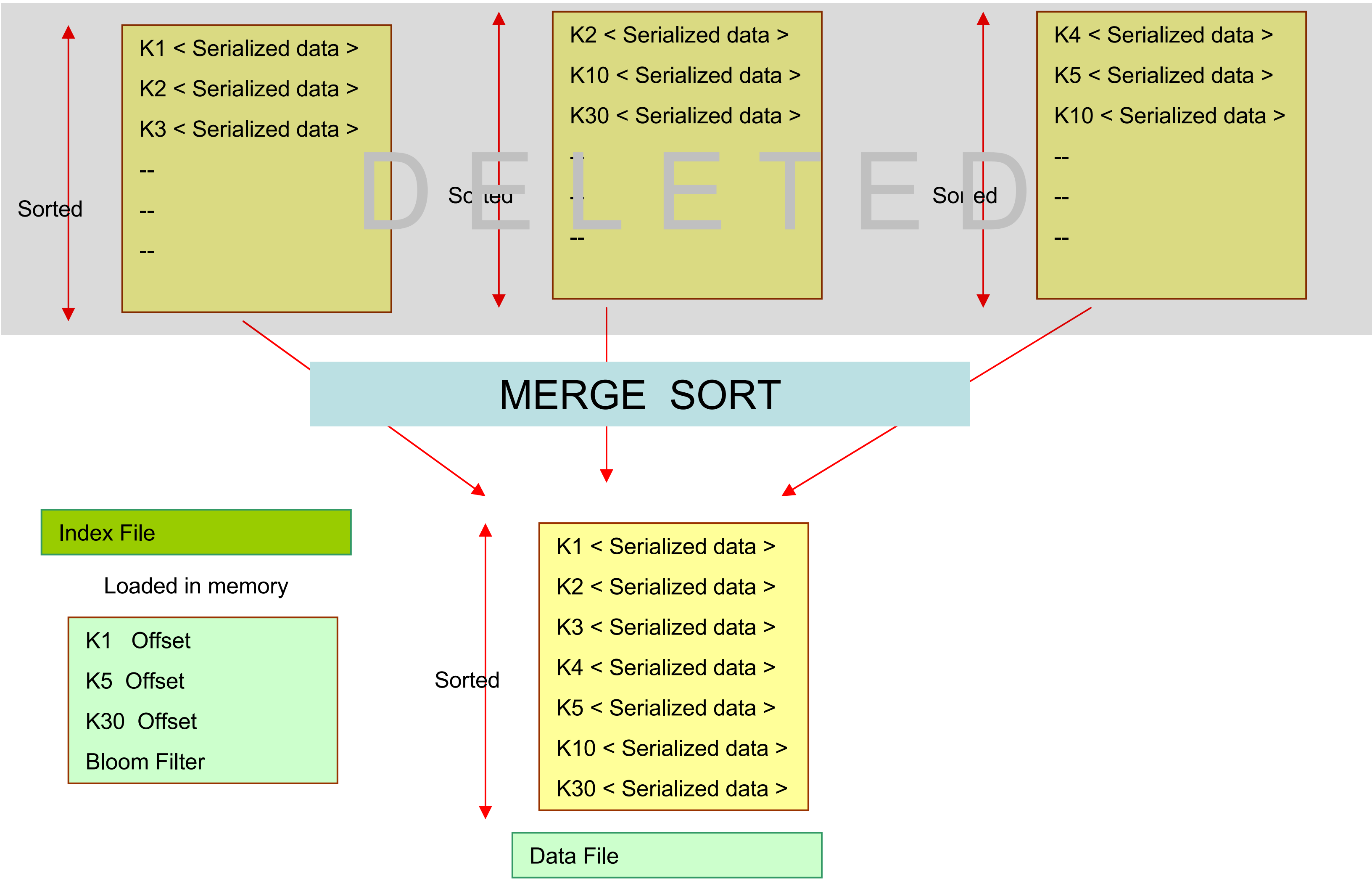
Write cont'd


Key (CF1 , CF2 , CF3)
Commit Log
Binary serialized
Key (CF1 , CF2 , CF3)
Memtable (CF1)
Memtable (CF2)
Memtable (CF2)
• Data size
• Number of Objects
• Lifetime
FLUSH
Data file on disk
Dedicated Disk
K128 Offset
K256 Offset
K384 Offset
Bloom Filter
(Index in memory)
<Key name><Size of key Data><Index of columns/supercolumns>< Serialized column family>

BLOCK Index <Key Name> Offset, <Key Name> Offset

<Key name><Size of key Data><Index of columns/supercolumns>< Serialized column family>

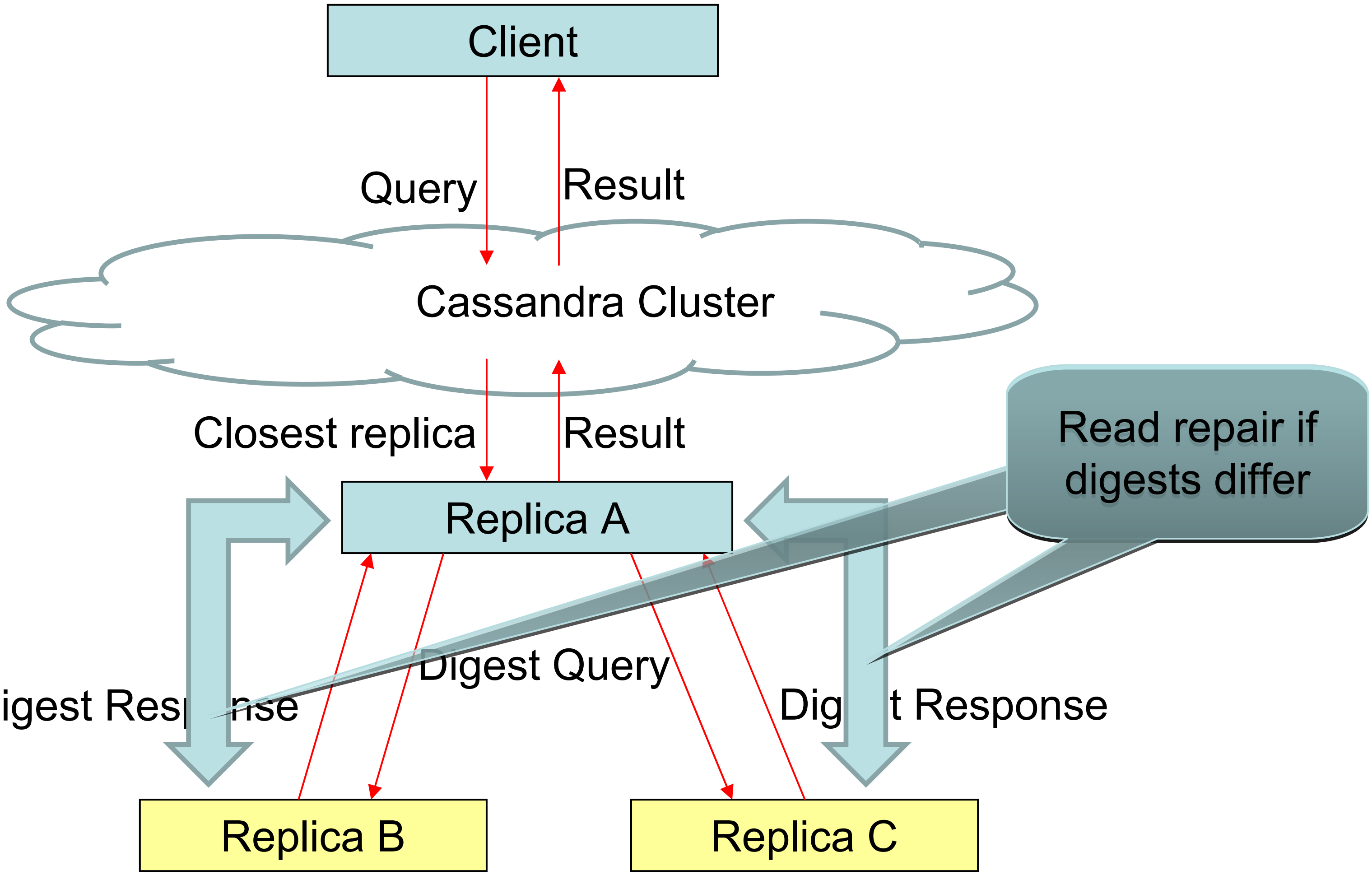
Compactions



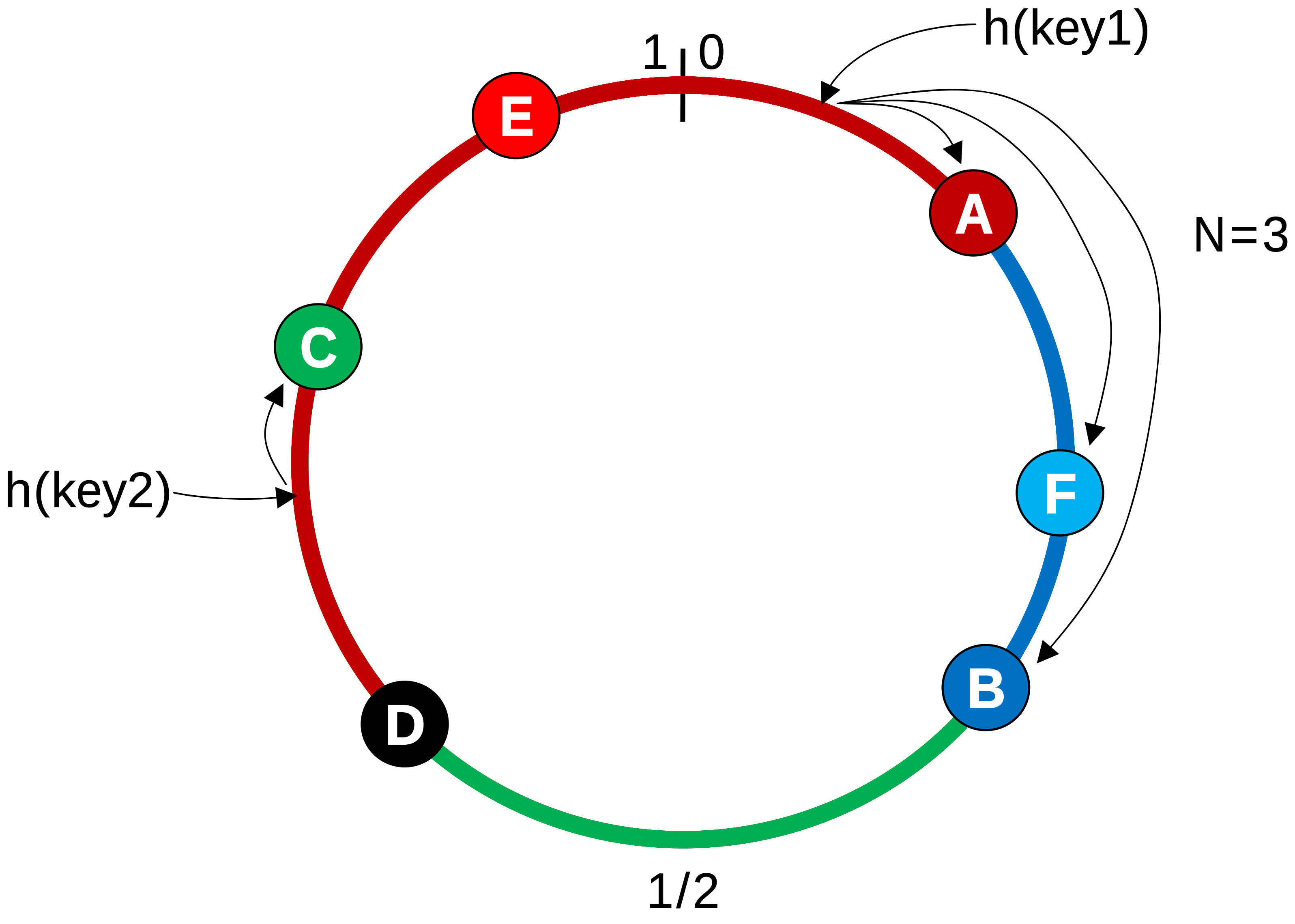
Write Properties

- No locks in the critical path
- Sequential disk access
- Behaves like a write back Cache
- Append support without read ahead
- Atomicity guarantee for a key
- "Always Writable"
  - accept writes during failure scenarios

Read


Client
Query
Result
Cassandra Cluster
Closest replica
Result
Read repair if digests differ
Replica A
Digest Query
Replica B
Replica C

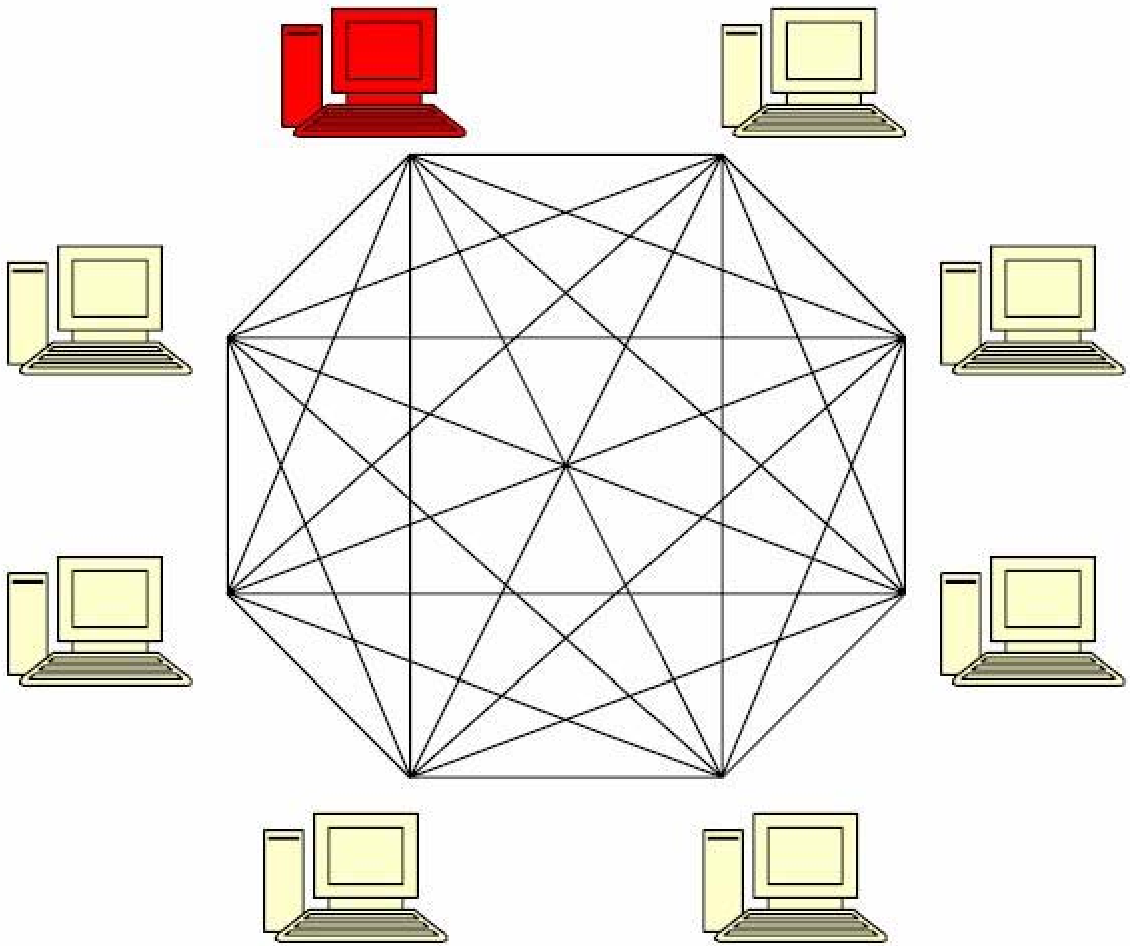
Partitioning And Replication



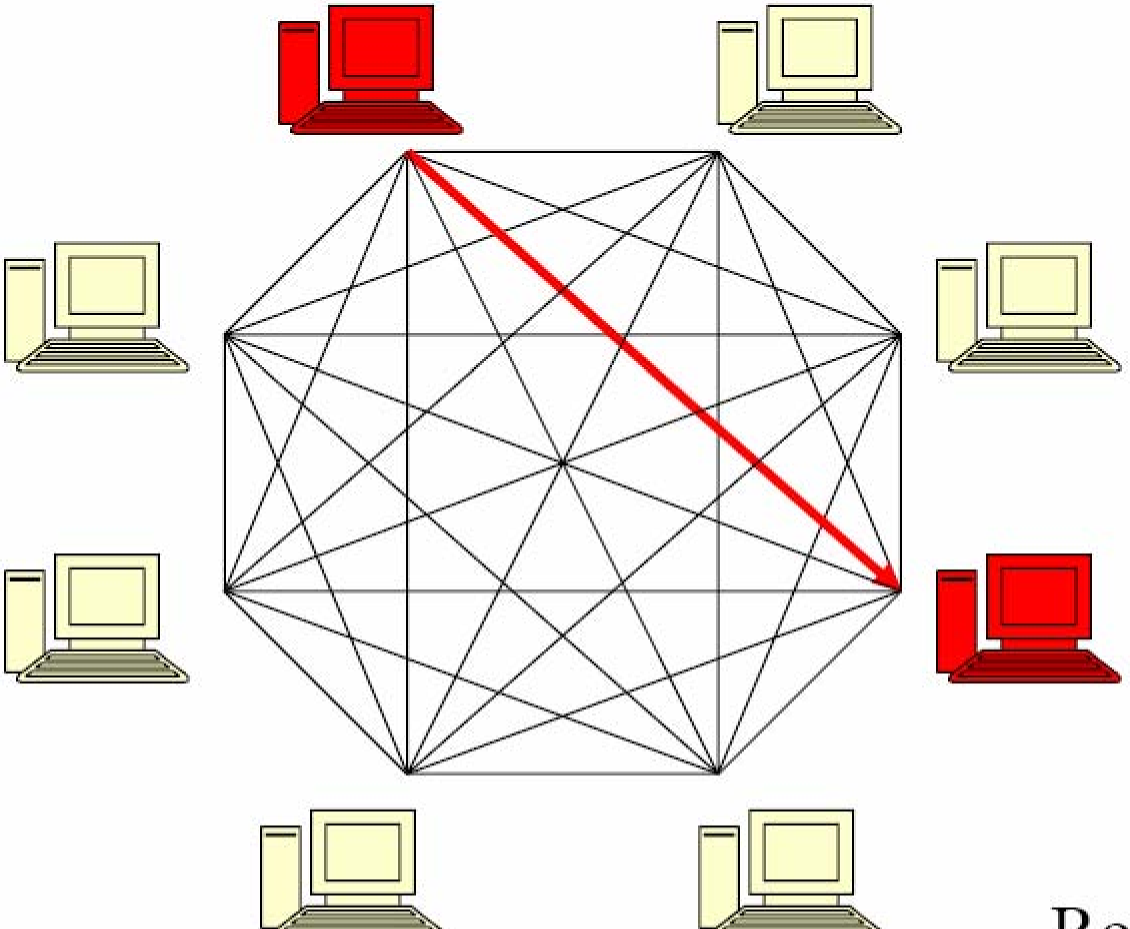
Cluster Membership and Failure Detection

- Gossip protocol is used for cluster membership.
- Super lightweight with mathematically provable properties.
- State disseminated in O(logN) rounds where N is the number of nodes in the cluster.
- Every T seconds each member increments its heartbeat counter and selects one other member to send its list to.
- A member merges the list with its own list .

Gossip

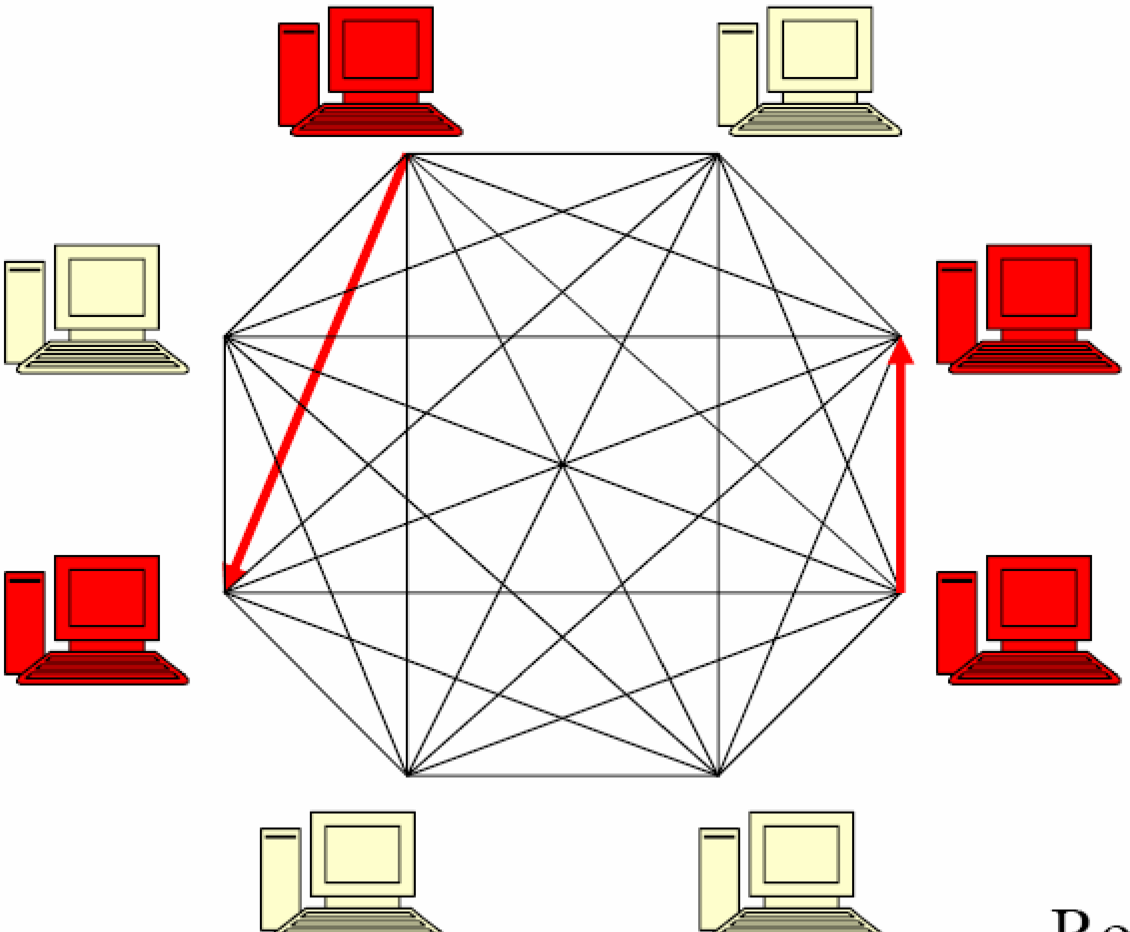


Gossip



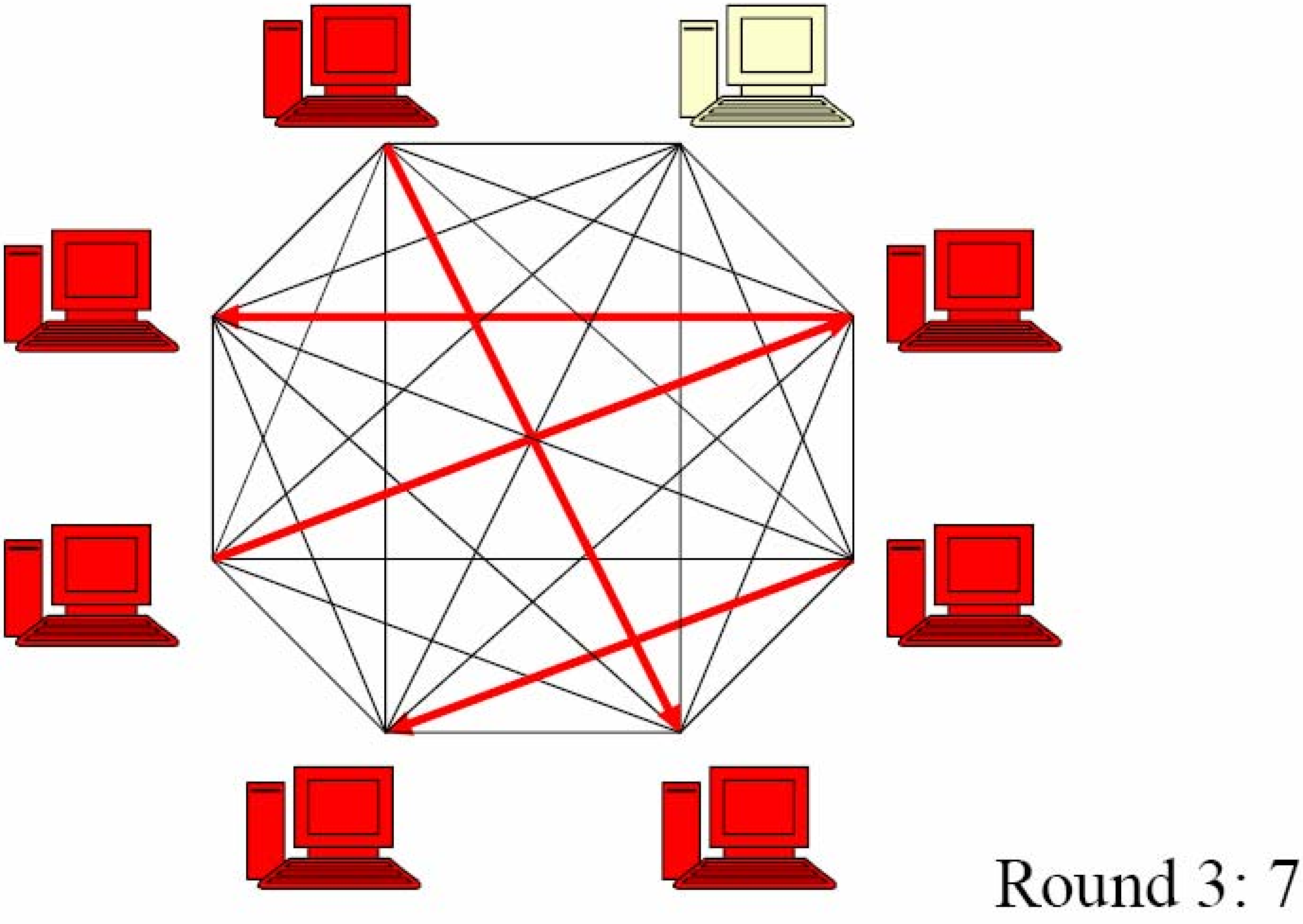
Round 1: 2

Gossip



Round 2: 4

Gossip



Accrual Failure Detector

- Valuable for system management, replication, load balancing etc.
- Defined as a failure detector that outputs a value, PHI, associated with each process.
- Also known as Adaptive Failure detectors - designed to adapt to changing network conditions.
- The value output, PHI, represents a suspicion level.
- Applications set an appropriate threshold, trigger suspicions and perform appropriate actions.
- In Cassandra the average time taken to detect a failure is 10-15 seconds with the PHI threshold set at 5.

Properties of the Failure Detector

- If a process p is faulty, the suspicion level

  $\Phi(t) \rightarrow \infty$ as $t \rightarrow \infty$.
- If a process p is faulty, there is a time after which $\Phi(t)$ is monotonic increasing.
- A process p is correct $\Leftrightarrow$ $\Phi(t)$ has an ub over an infinite execution.
- If process p is correct, then for any time T,

  $\Phi(t) = 0$ for $t >= T$.

Implementation

- PHI estimation is done in three phases
  - Inter arrival times for each member are stored in a sampling window.
  - Estimate the distribution of the above inter arrival times.
  - Gossip follows an exponential distribution.
  - The value of PHI is now computed as follows:
    - $\Phi(t) = -\log_{10}(P(t_{\mathbf{now}} - t_{\mathbf{last}}))$

      where P(t) is the CDF of an exponential distribution. P(t) denotes the probability that a heartbeat will arrive more than t units after the previous one. $P(t) = (1 - e^{-t\lambda})$

The overall mechanism is described in the figure below.

Information Flow in the Implementation

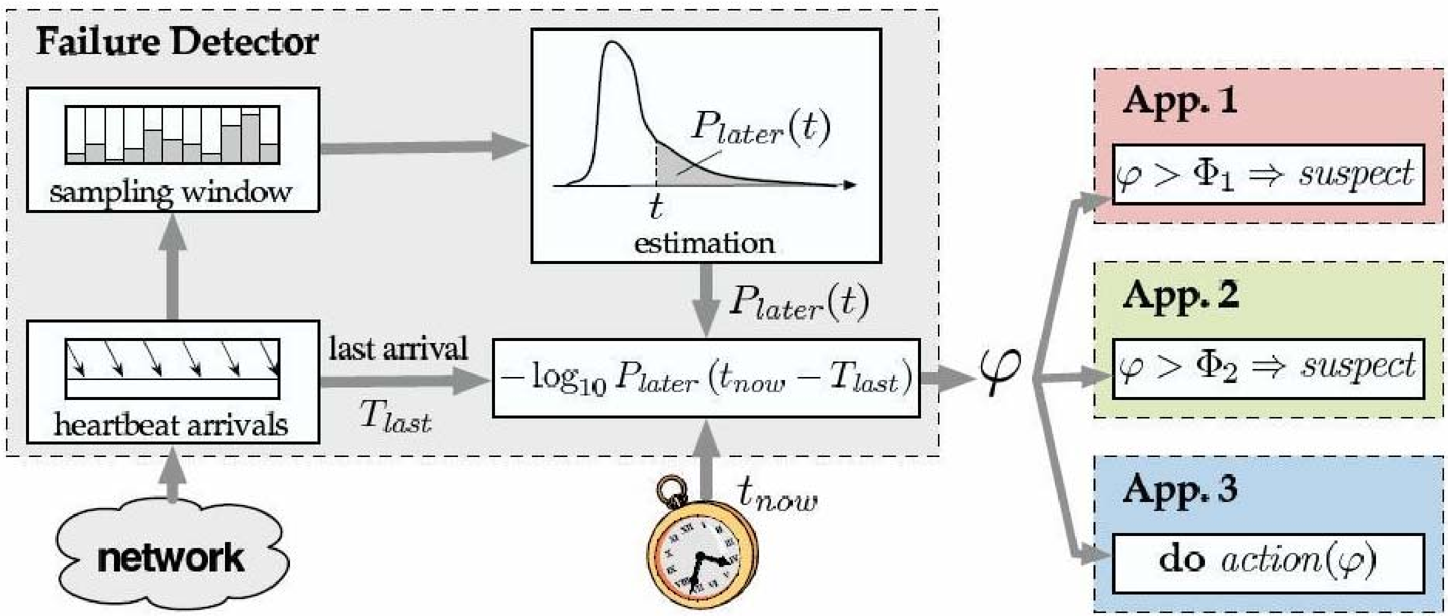


Fig. 4. Information flow in the proposed implementation of the φ failure detector, as seen at process q. Heartbeat arrivals arrive from the network and their arrival time are stored in the sampling window. Past samples are used to estimate some arrival distribution. The time of last arrival T_{last}, the current time t_{now} and the estimated distribution are used to compute the current value of φ. Applications trigger suspicions based on some threshold (Φ_1 for App. 1 and Φ_2 for App. 2), or execute some actions as a function of φ (App. 3).

Performance Benchmark

- Loading of data - limited by network bandwidth.
- Read performance for Inbox Search in production:

	Search Interactions	Term Search
Min	7.69 ms	7.78 ms
Median	15.69 ms	18.27 ms
Average	26.13 ms	44.41 ms

MySQL Comparison

- MySQL > 50 GB Data
  Writes Average : ~300 ms
  Reads Average : ~350 ms
- Cassandra > 50 GB Data
  Writes Average : 0.12 ms
  Reads Average : 15 ms

Lessons Learnt

- Add fancy features only when absolutely required.
- Many types of failures are possible.
- Big systems need proper systems-level monitoring.
- Value simple designs

Future work

- Atomicity guarantees across multiple keys
- Analysis support via Map/Reduce
- Distributed transactions
- Compression support
- Granular security via ACL’s

Questions?