BIG DATA AND CONSISTENCY

Amy Babay

Outline

- Big Data
  - What is it?
  - How is it used? What problems need to be solved?
- Replication
  - What are the options?
  - Can we use this to solve Big Data's problems?
- Putting them together
  - What works?
  - What are existing tools doing?

Big Data: Numbers

- Facebook: 100 petabytes of photos and videos
  - http://newsroom.fb.com/Infrastructure
- Large Hadron Collider: produces 15 petabytes of data annually
  - http://home.web.cern.ch/about/computing
- Cassandra at Netflix (as of July 2012):
  - 472 total machine; 65 TB of data (total across 30 clusters)
  - 72 machines; 28 TB of data (largest cluster)
  - http://www.slideshare.net/greggulrich/cassandra-operations-at-netflix
- Ebay's Cassandra "taste graph" (as of March 2013):
  - 32 nodes; 5 TB (replicated twice = 10 TB), expected to quadruple in 1 year
  - http://www.slideshare.net/planetcassandra/e-bay-nyc
- Twitter metrics in Cassandra (Cuckoo): 492 GB/day
  - http://www.scribd.com/doc/59830692/Cassandra-at-Twitter

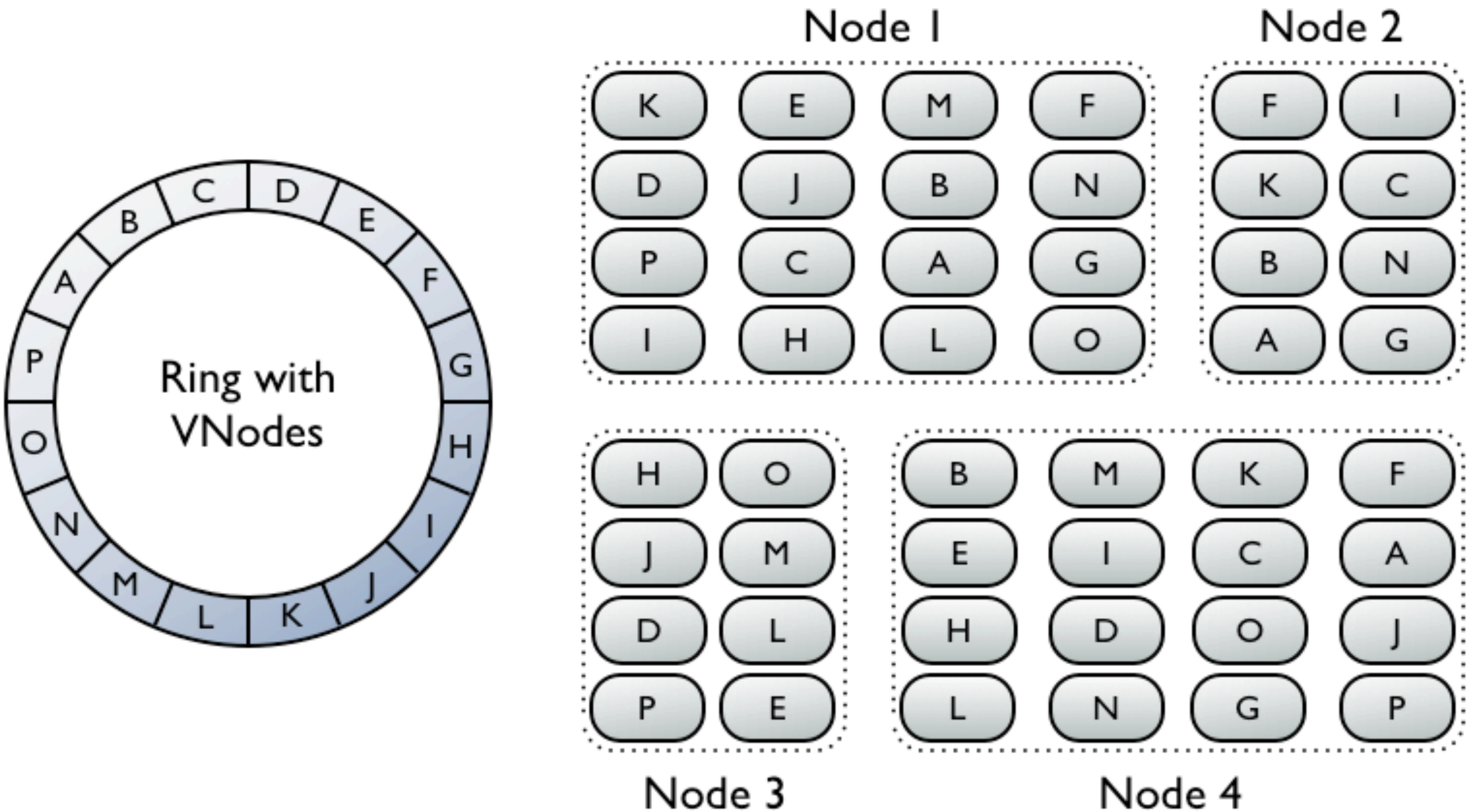
Using Big Data – Different Use Cases

- Write once
- Simple key-value updates
- Compound key-value updates
- Database transactions

Accessing Big Data: Write Once/Read Many

- Data never changes once it is written; new data can be added
- Requirements:
  - Partition data
  - Locate/retrieve data
- Cassandra Solutions:
  - Consistent hashing
  - Gossip to propagate data locations

Partitioning Data: Consistent Hashing



http://www.datastax.com/dev/blog/virtual-nodes-in-cassandra-1-2

Locating Data: Gossip

- New nodes start with the addresses of a small set of “seed” nodes, which they contact to get information about the cluster
- Once per second, exchange state with up to 3 other nodes
- Information about which ranges belong to which nodes is propagated by eventual path

Accessing Big Data: Simple Key-value Updates

- Each update only affects one key-value pair
- Requirements:
  - Get the update to all replicas
  - Potentially enforce guarantees on the visibility of the update
- Cassandra Solutions:
  - Hinted handoff, read repair, anti-entropy sessions
  - "Tunable consistency"

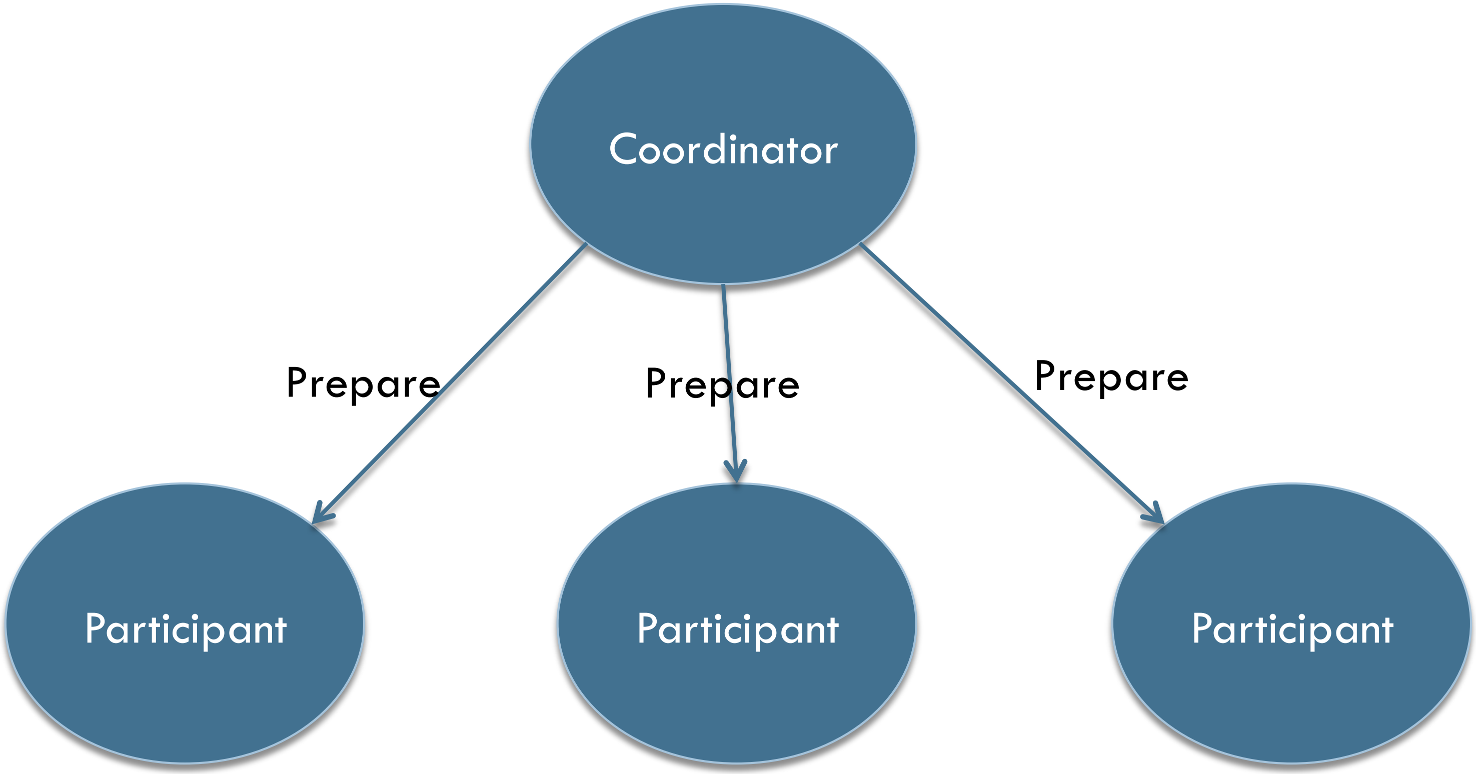
Accessing Big Data: Compound Updates and Transactions

- Updates can affect multiple key-value pairs
- Updates may be conditional
- Requirements:
  - Coordinate across replicas for different key-value pairs
- Cassandra Solutions:
  - Adding support for atomic batches – not quite there yet
- What else can we do?

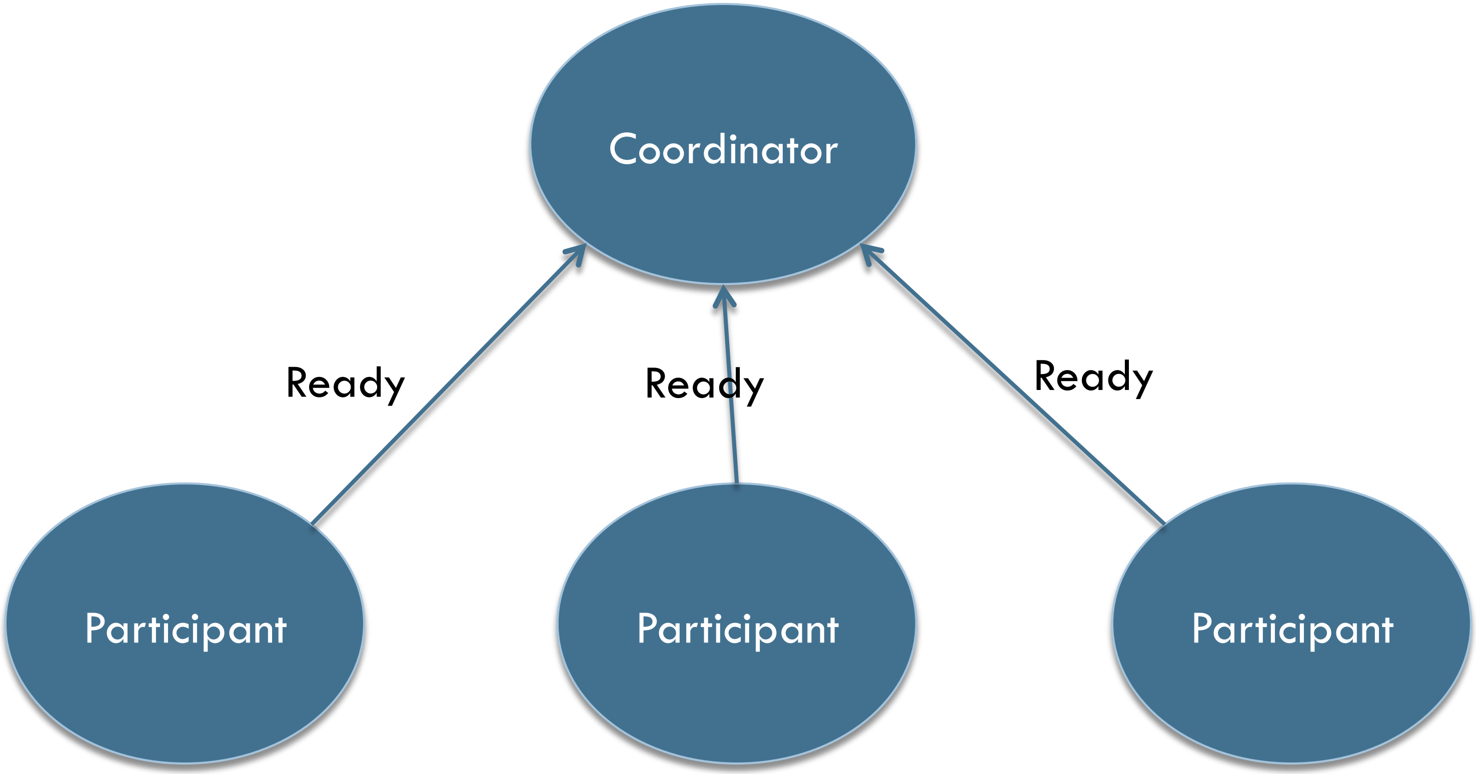
Replication Protocols

- Ensure that all replicas apply updates in the same order
- A replication engine can impose a total order on all updates in the system

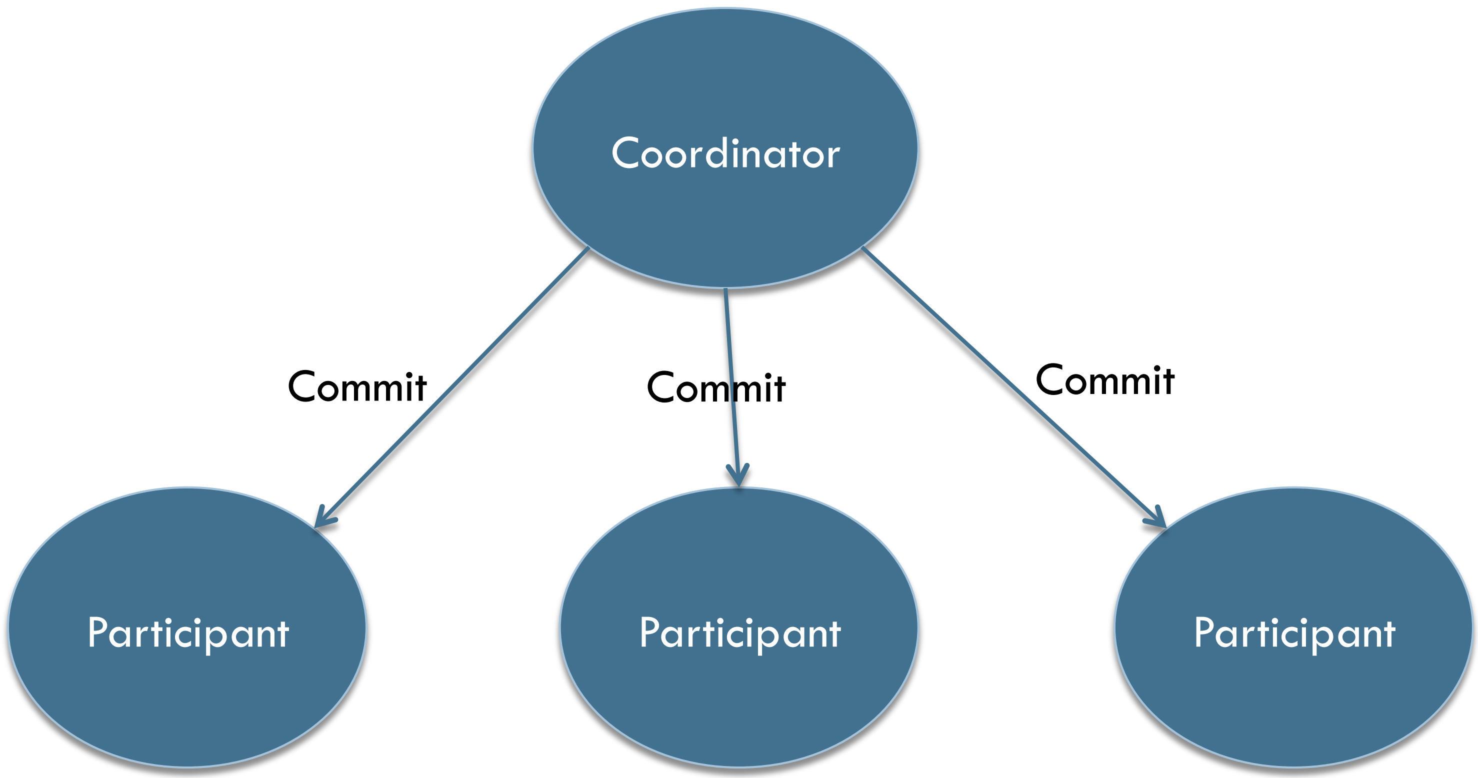
Two-phase Commit: Example 1



Two-phase Commit: Example 1

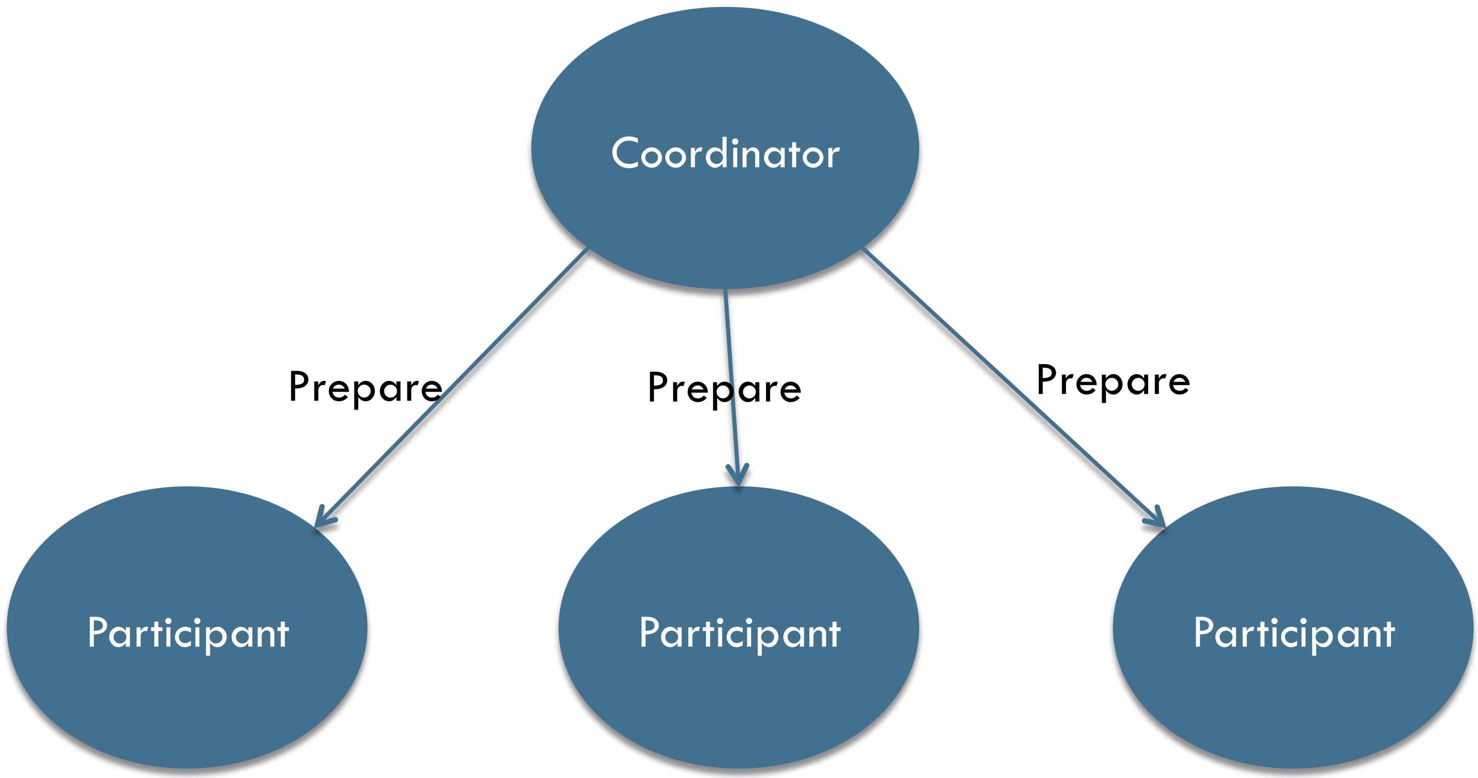


Two-phase Commit: Example 1

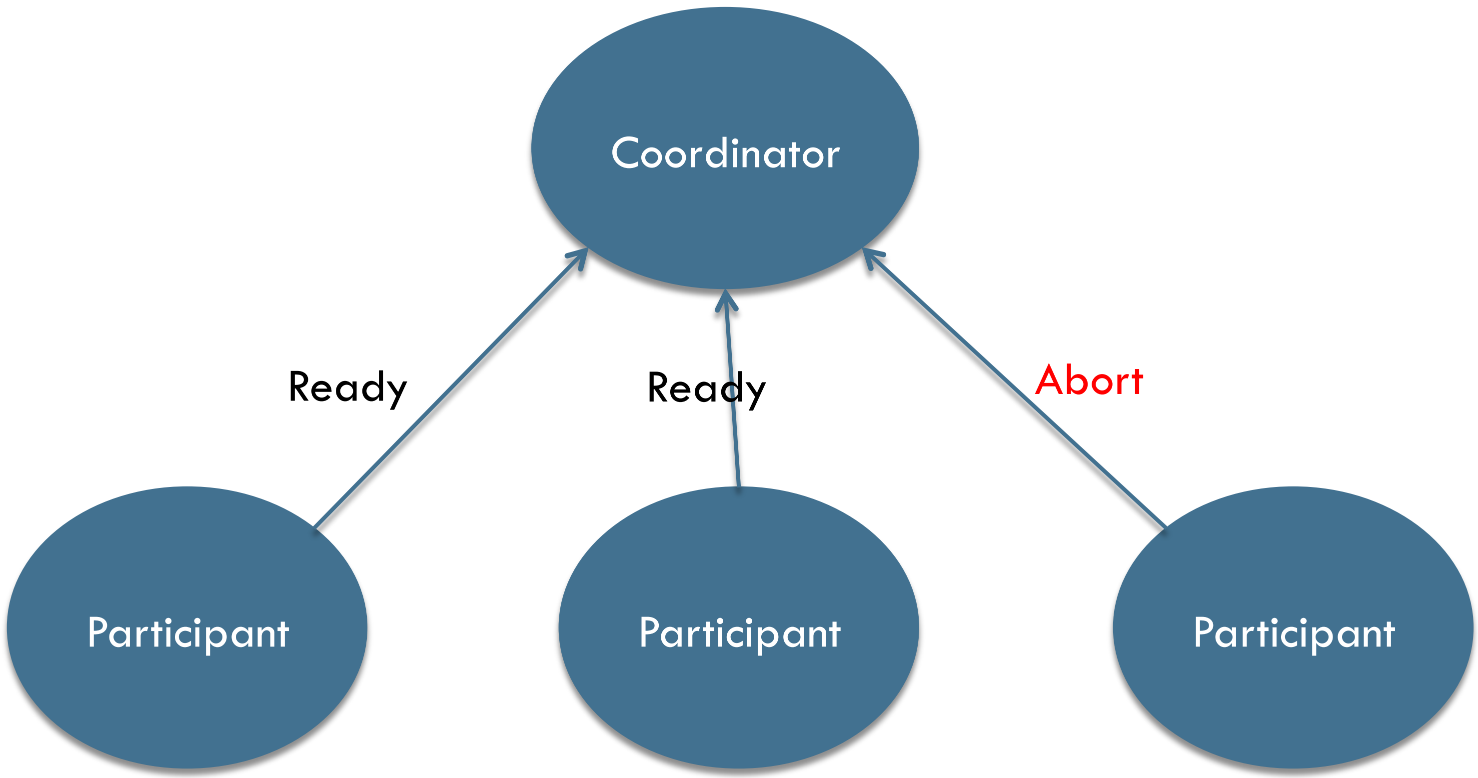


Transaction Committed

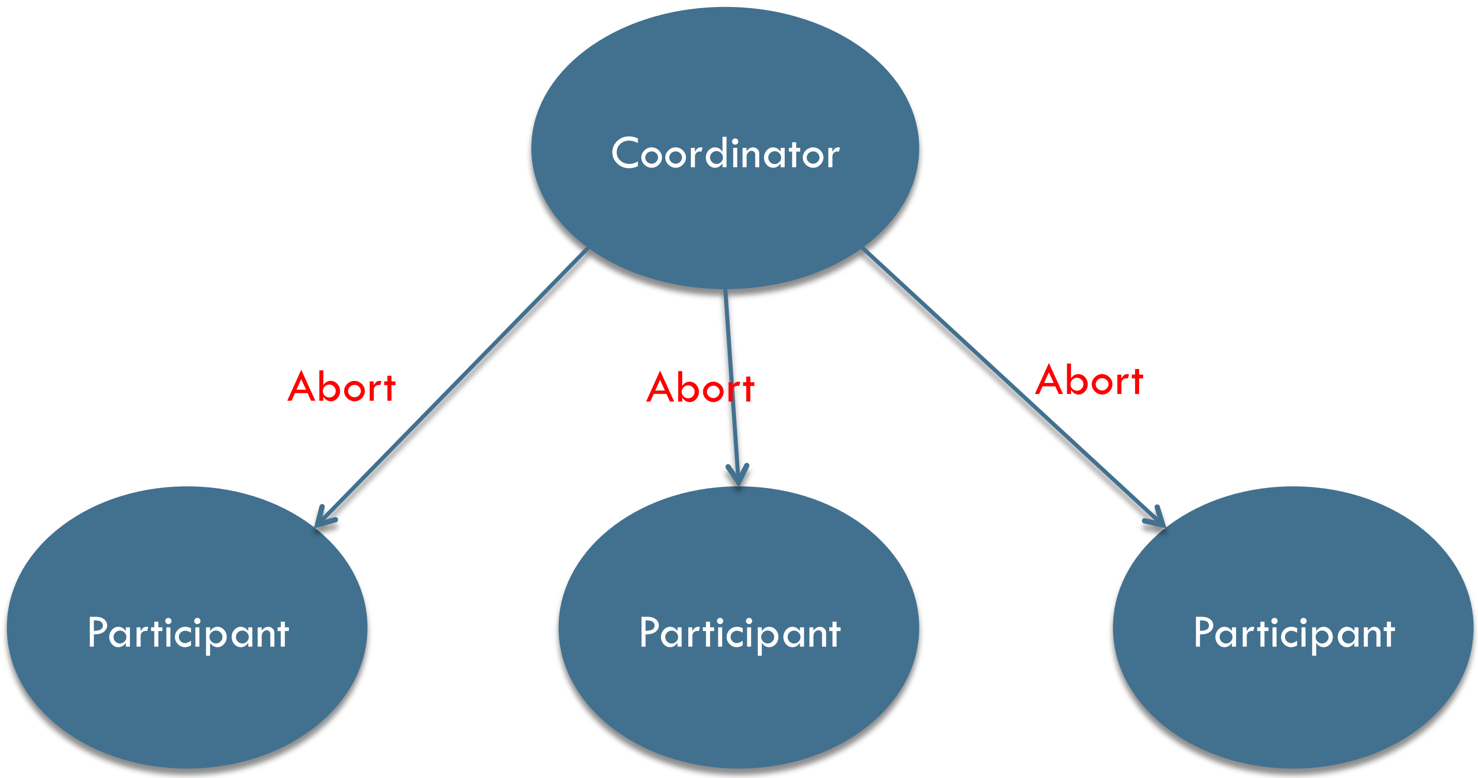
Two-phase Commit: Example 2



Two-phase Commit: Example 2



Two-phase Commit: Example 2



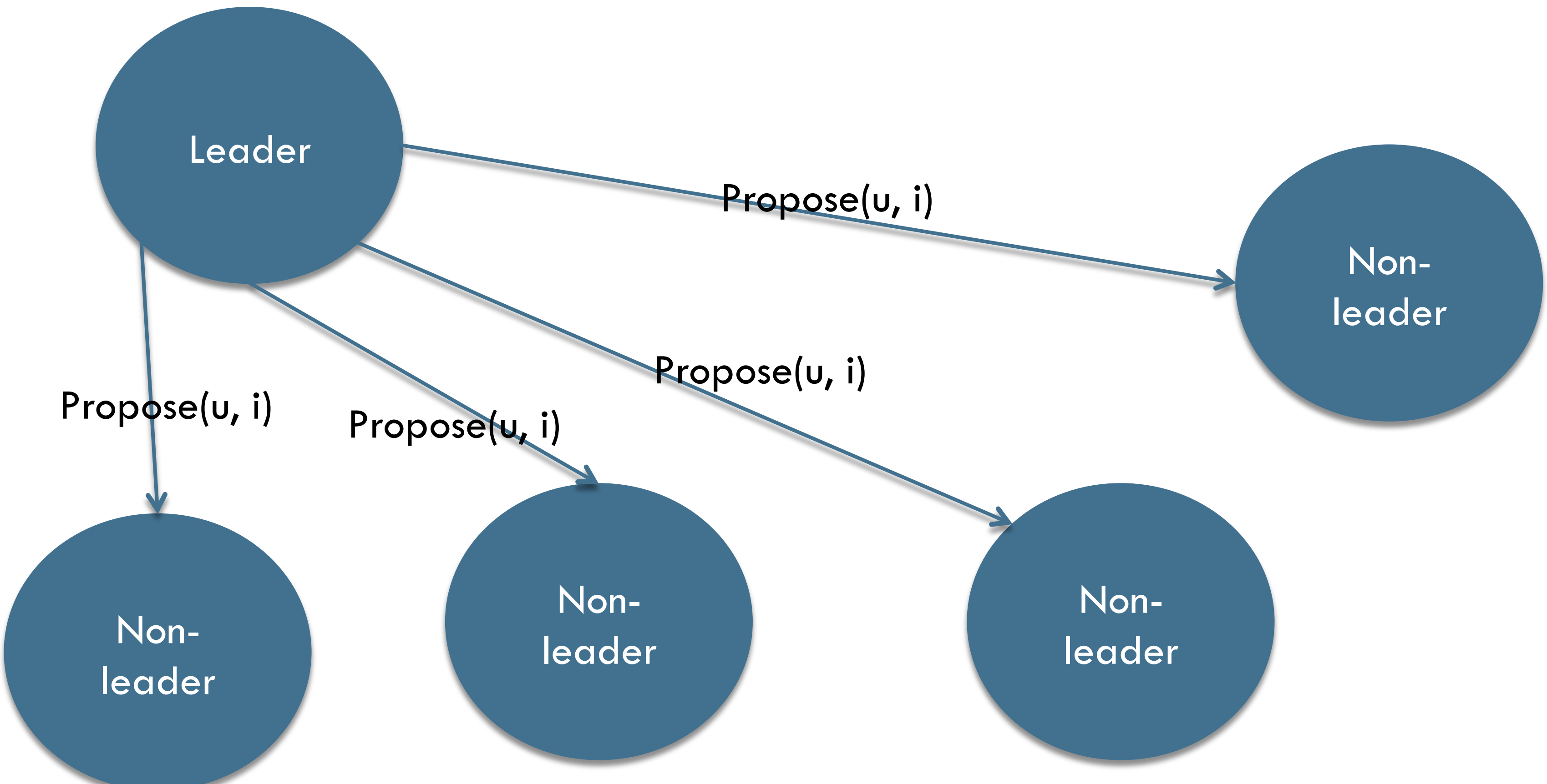
Transaction Aborted

Two-phase Commit: Properties

- Can be used for general transactions; not only the special case of replication
- Vulnerable to coordinator failure

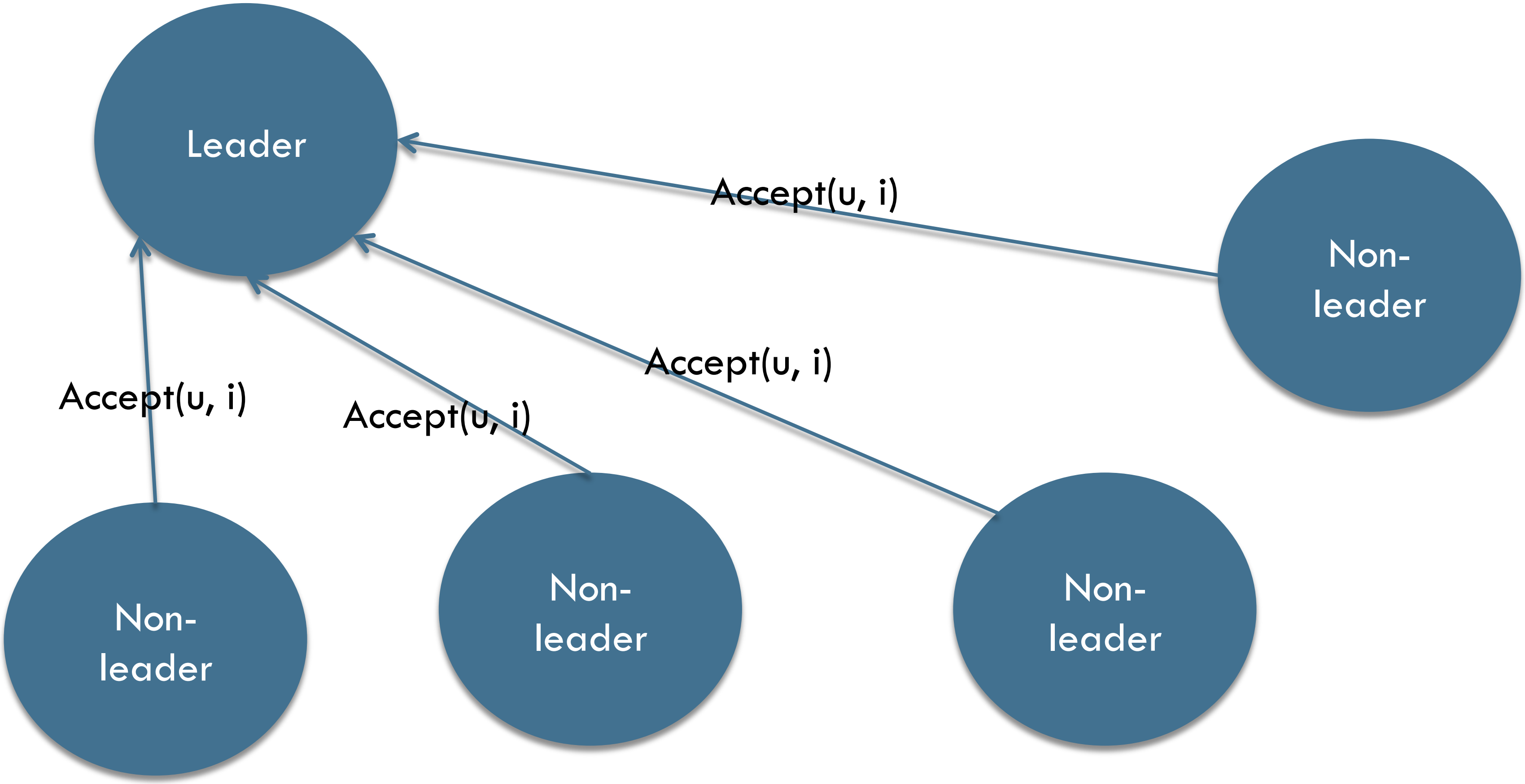
Paxos: Normal Case

When the leader receives update u from some replica:



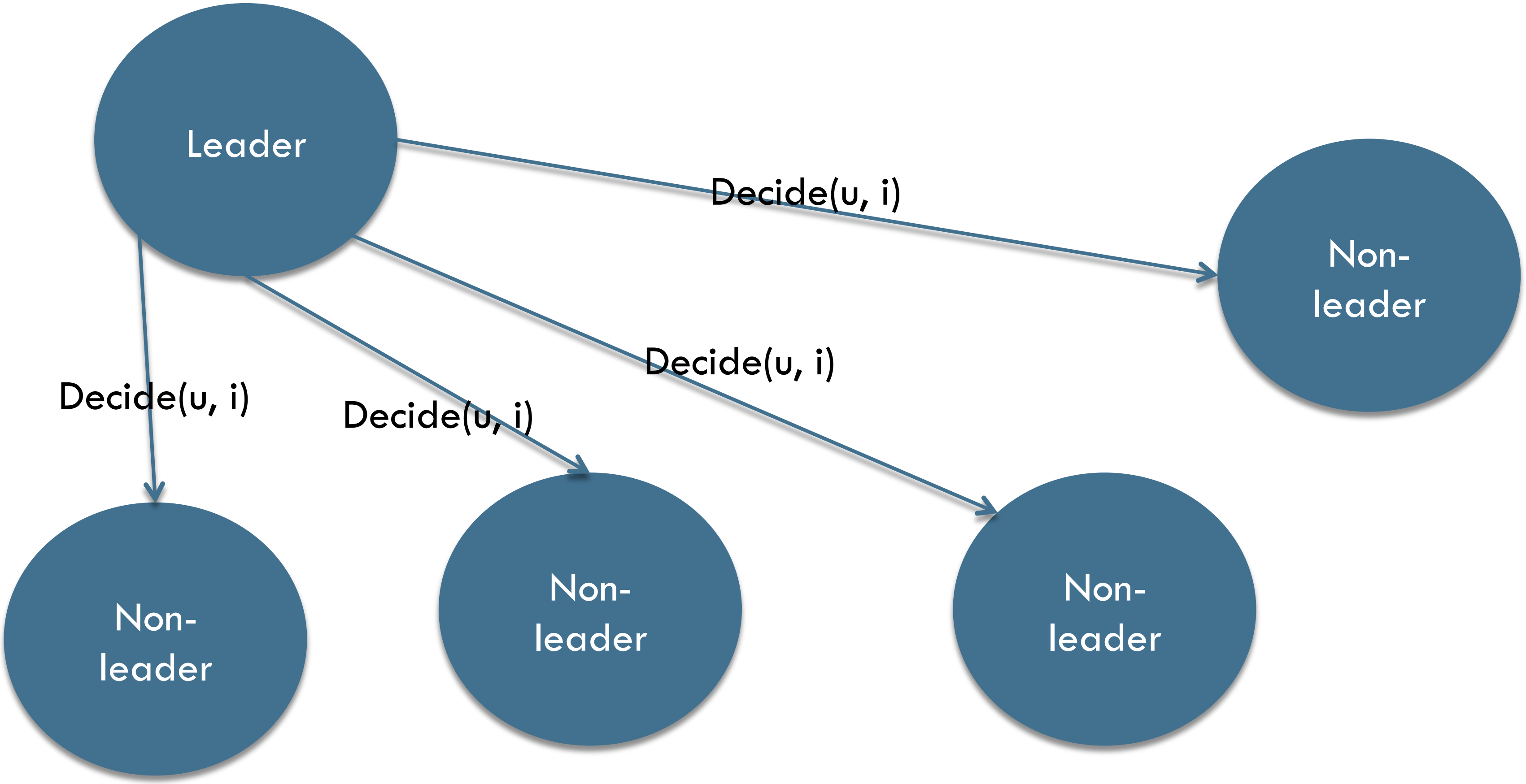
Paxos: Normal Case

If no replica has assigned an update u' != u to sequence i:



Paxos: Normal Case

Once the leader receives "accept" from a majority:

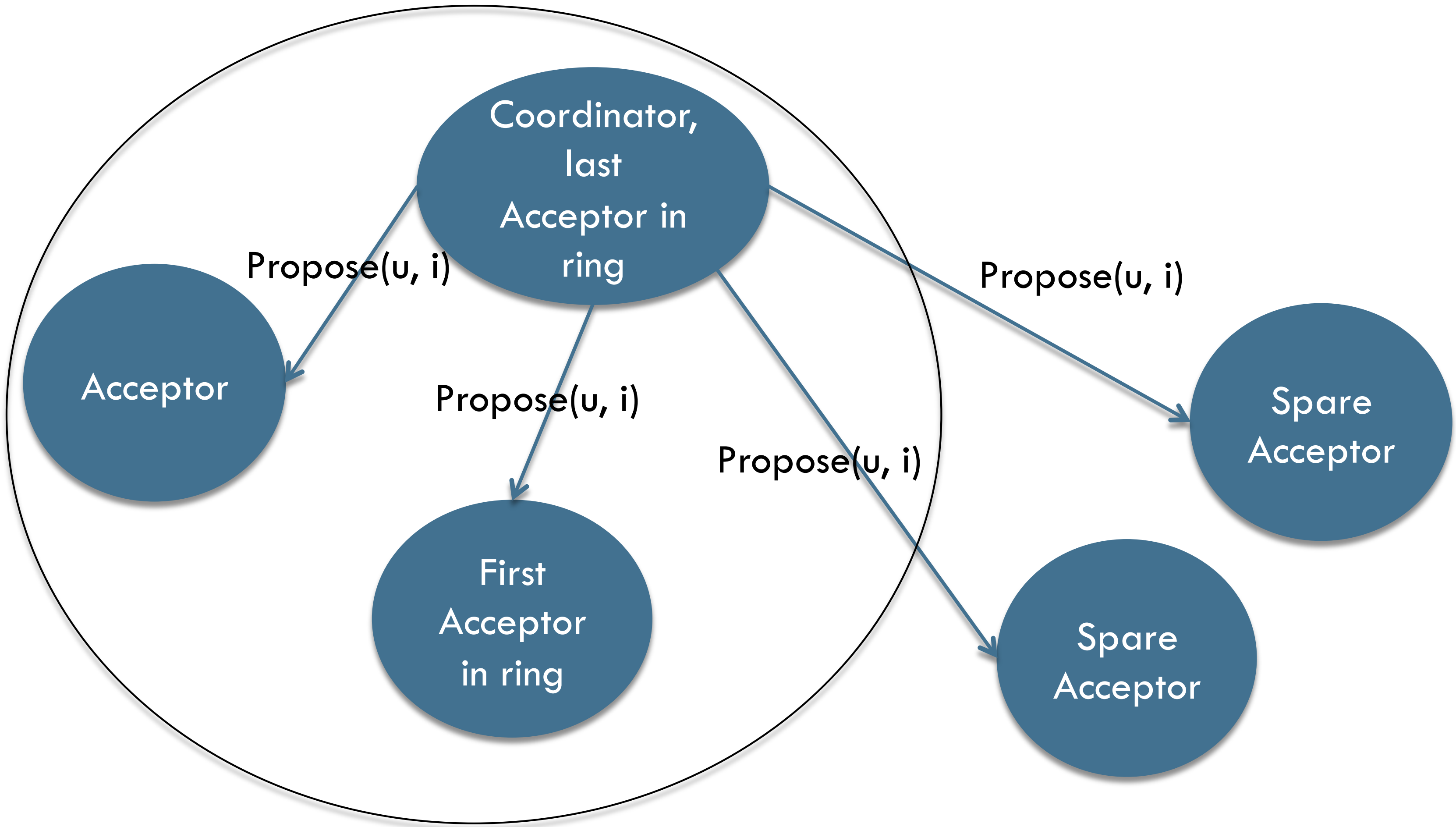


Paxos: Properties

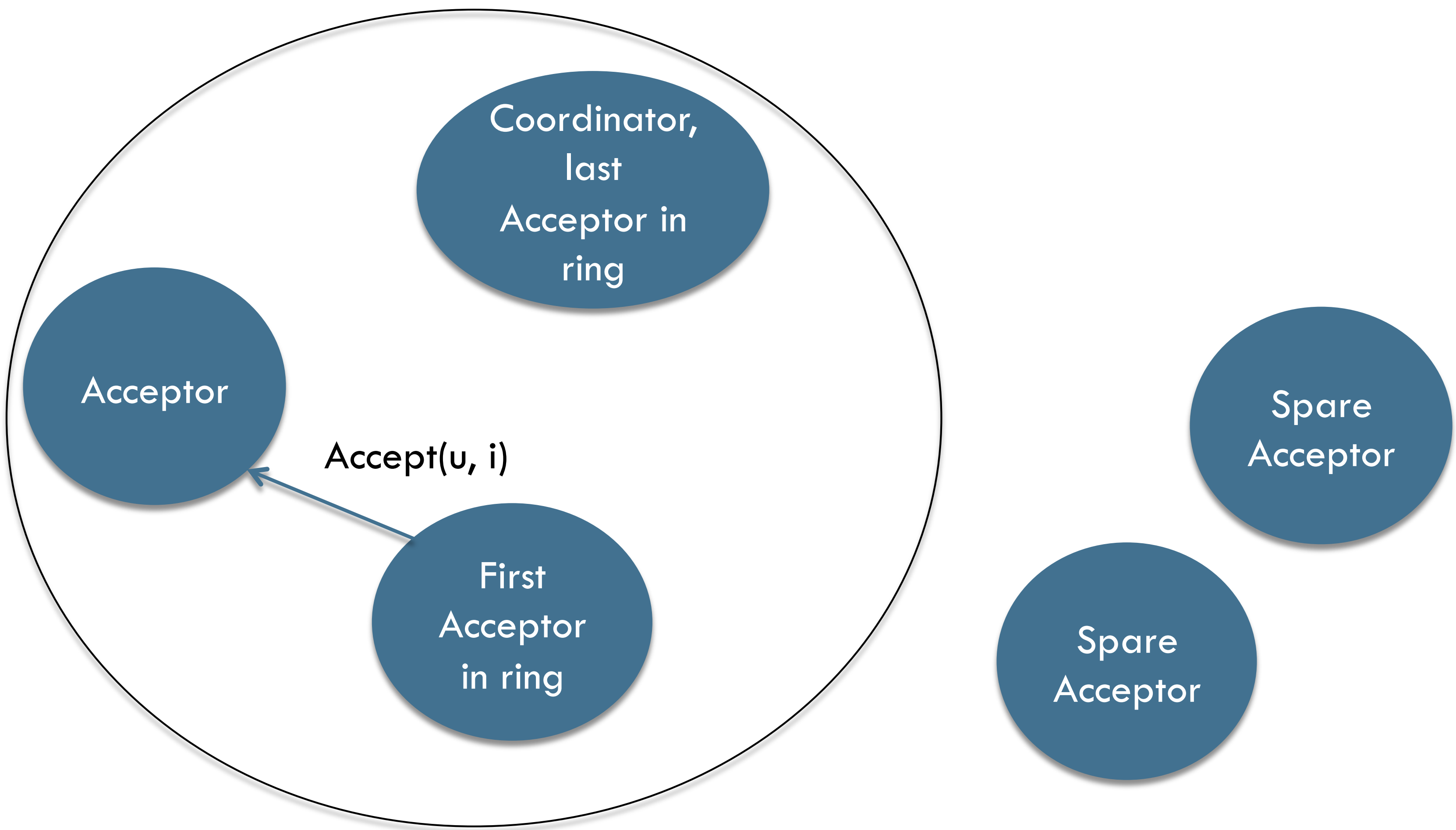
- Extremely resilient: leader + any quorum can make progress
- Provides strong consistency (only)
- Processing many “accept” messages may limit performance

Ring Paxos: Normal Case

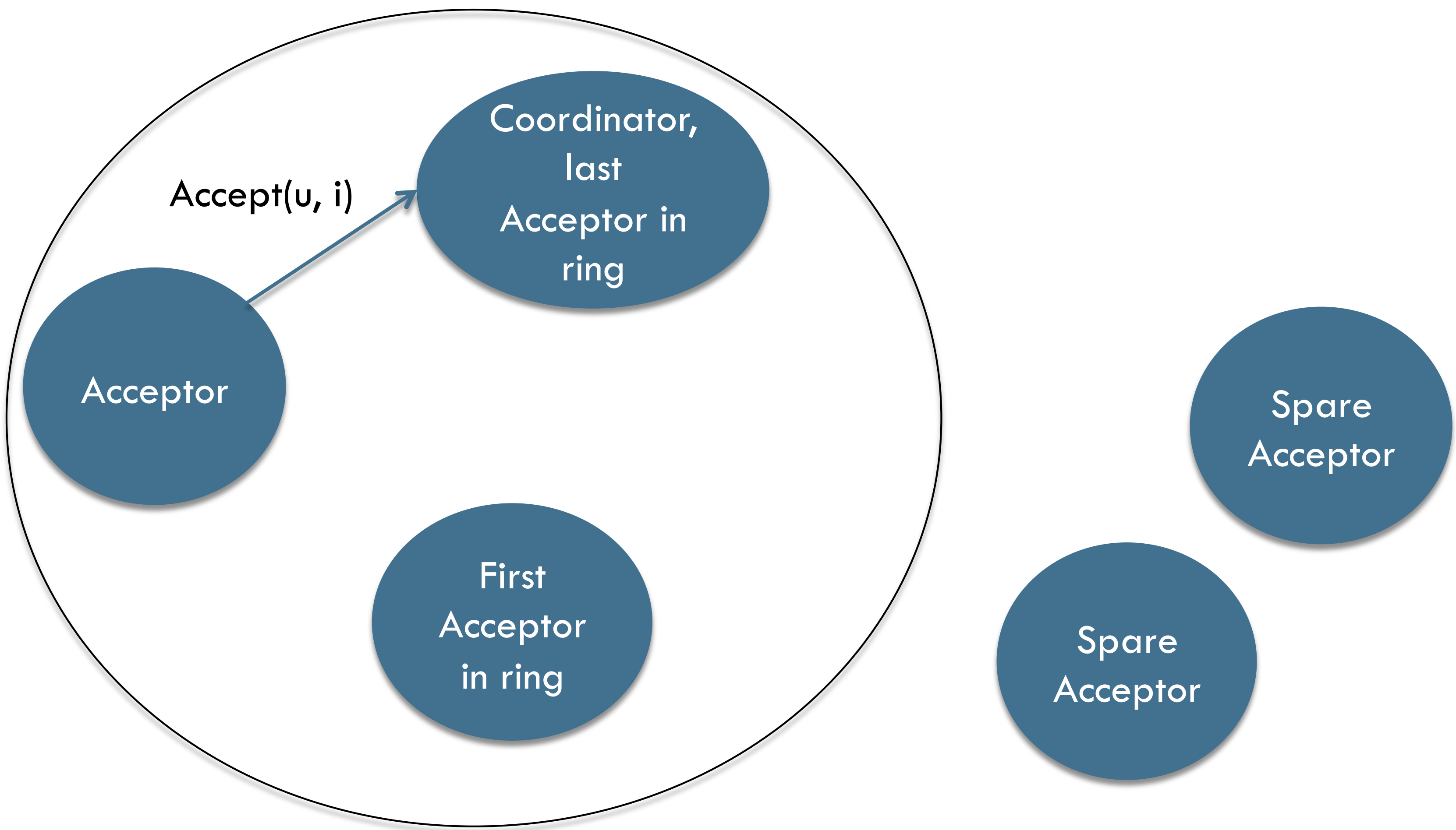
When the leader receives update u:



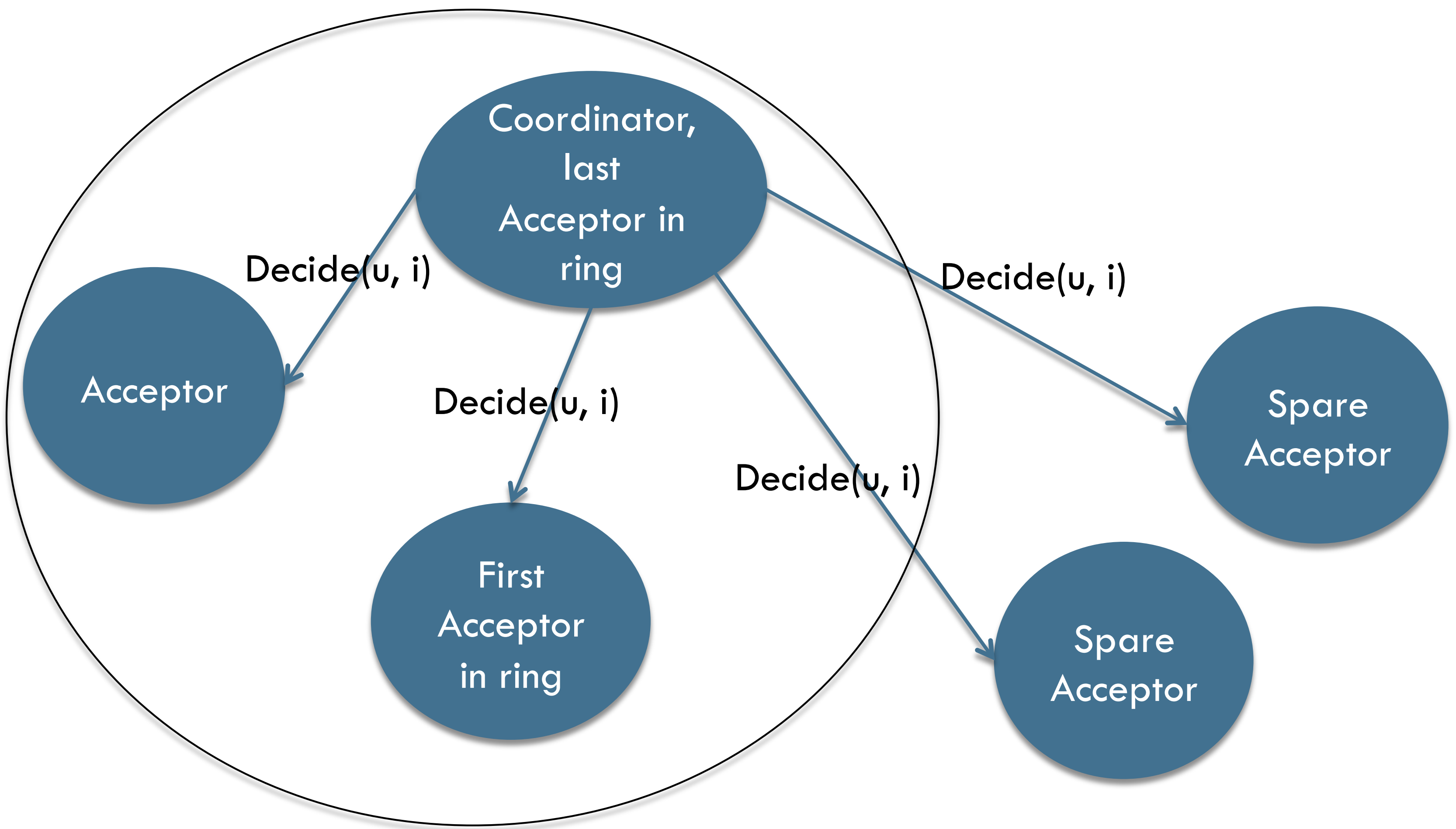
Ring Paxos: Normal Case



Ring Paxos: Normal Case



Ring Paxos: Normal Case



Ring Paxos: Properties

- Improves the performance of Paxos (eliminates "accept" bottleneck)
- Reduces the resiliency of Paxos (what if a member of the ring fails?)
- Same semantics as Paxos—strong consistency

Congruity: Normal Case

- Replicas send updates via a group communication service using safe delivery
- While in a primary component, replicas can apply updates as soon as they are delivered (by group communication service)
- While not in a primary component, updates are still exchanged but not applied (if strong consistency is needed)

Congruity: Properties

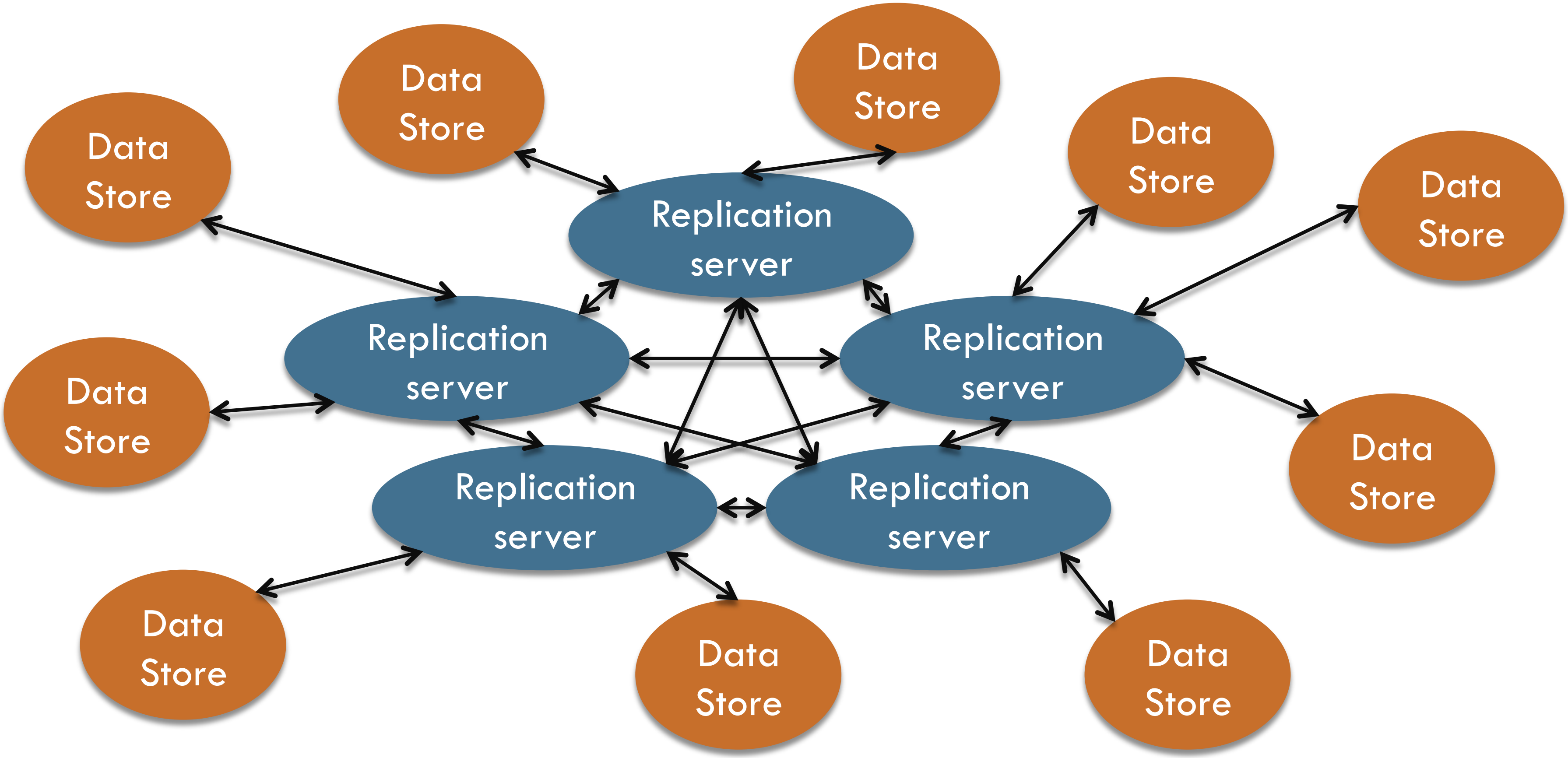
- Flexible semantics: weak consistency queries, dirty queries, commutative updates/timestamp semantics
  - Allows replicas not in a primary component to respond to queries
- Exchange updates while not in primary component + exchange state on membership change → Propagation by eventual path
- Avoids acknowledging every update
- Requires membership (reduces resiliency)

Revisiting Big Data

- Write once
- Simple key-value updates
- Compound key-value updates
- Database transactions

Replication Engines

- Provide total order on updates in the system
- Need to be able to handle throughput of the system

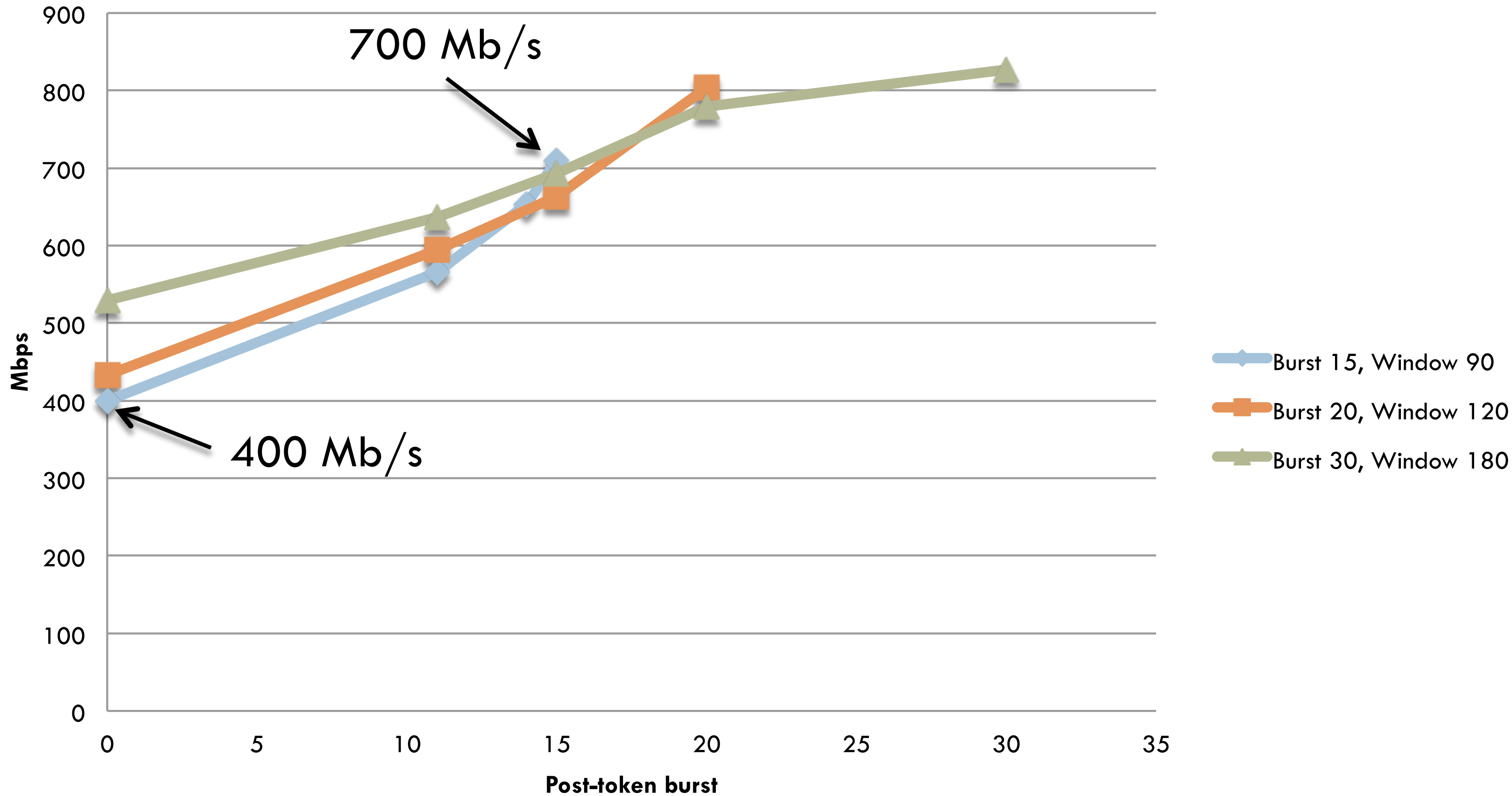


Improving Throughput for Group-Communication-based Replication

- Standard ring protocol:
  - Token circulates logical ring
  - Upon receiving the token, a participant sends all the messages it has/is allowed for that round, then passes the token to the next participant
- Accelerated ring protocol:
  - Token circulates logical ring
  - Upon receiving the token, a participant sends some fraction of the messages it has/is allowed for that round, passes the token, and then sends the remaining messages it has/is allowed

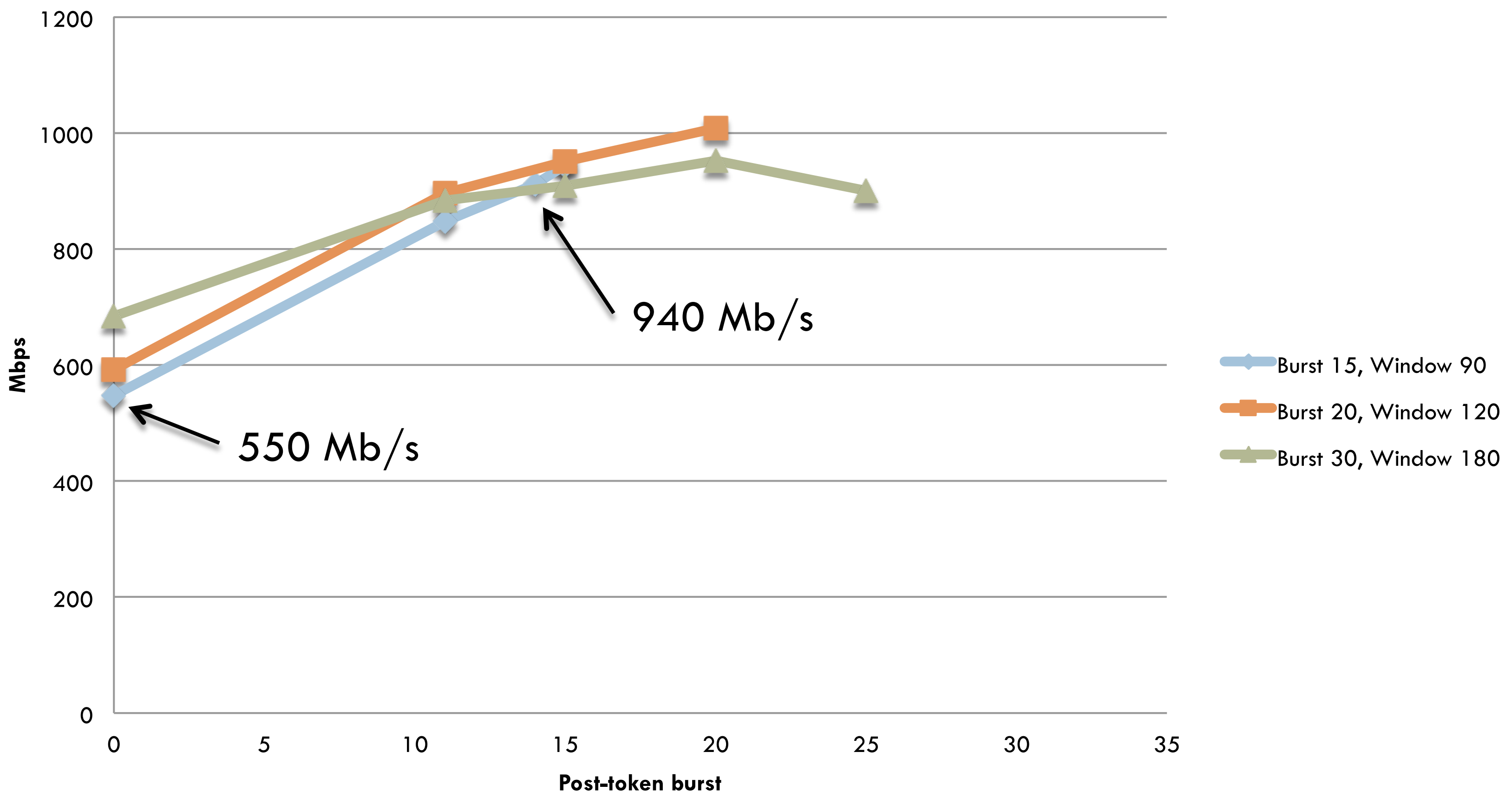
Throughput Comparison

Clouds 1-6: 1 Gb/s Cisco switch:



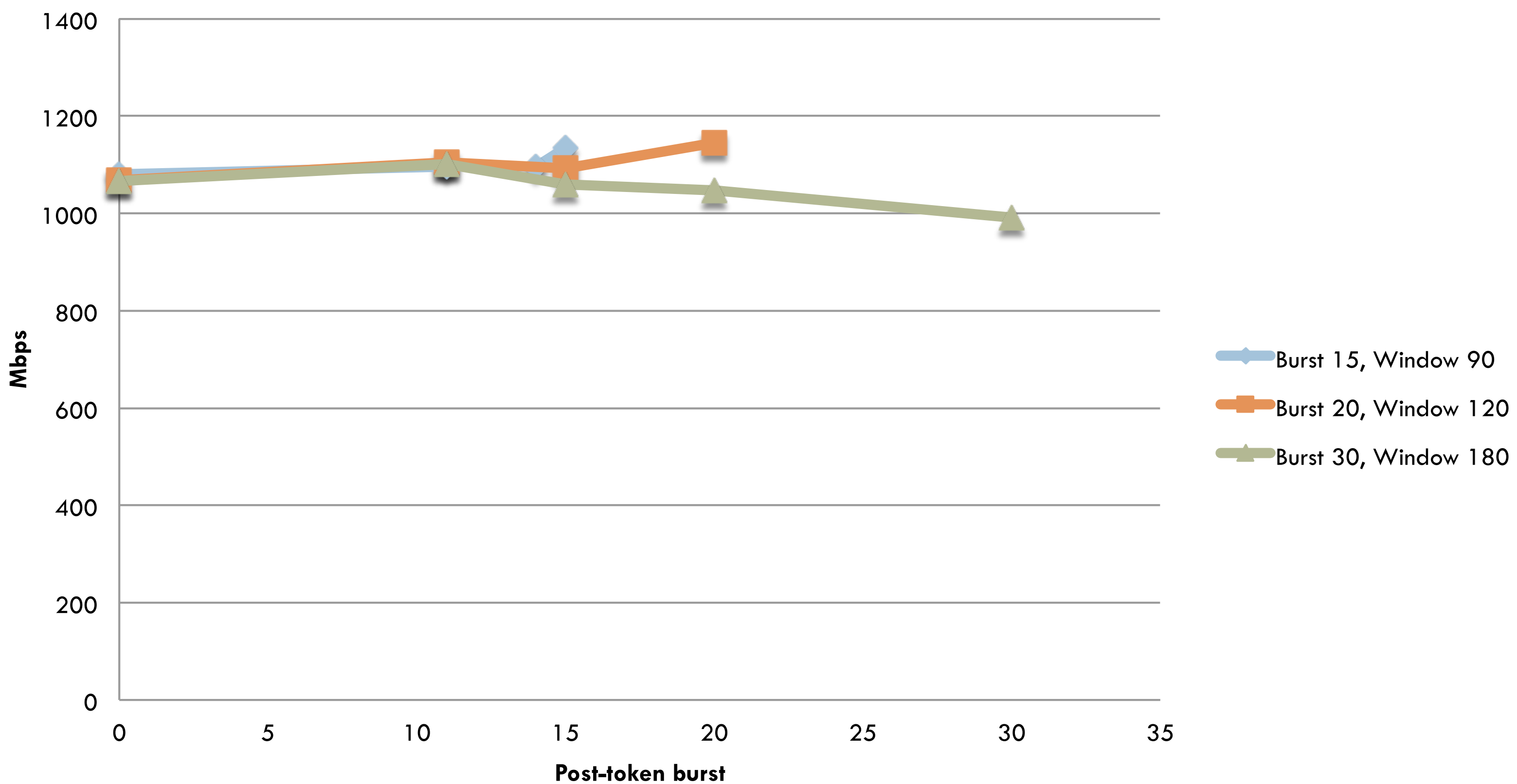
Throughput Comparison

Rains 1-5: 1 Gb/s Cisco switch:



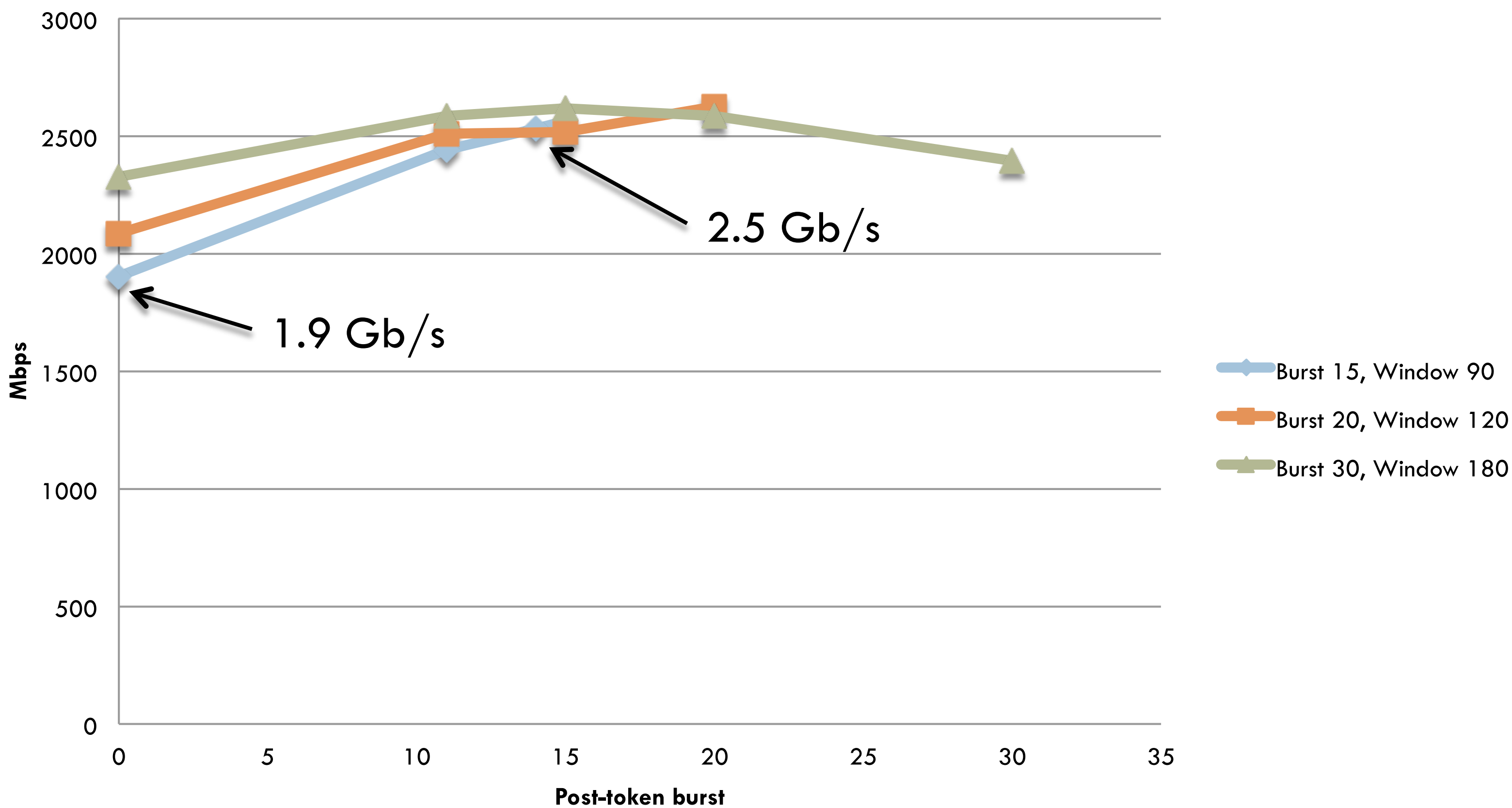
Throughput Comparison

Rains 1-5: 10 Gb/s Arista switch:



Throughput Comparison

Rains 9-16: 10 Gb/s Arista switch:



Solving the Big Data Problems

- Compound key-value updates: replication engines enforce total ordering of updates across servers
- Distributed transactions: is total order sufficient?
  - Consider update$_i$: “Read A. If A =1, write B=2”
  - All servers will assign the same order to the update, but if A is on server1 and B is on server2, what should server2 do when it orders update$_i$?

Solving the Big Data Problems

- Need something more general than replication to handle distributed transactions
  - Two-phase commit
  - Three-phase commit? Enhanced three-phase commit?
- Number of replicas per item is small (3-4)
  - More efficient to coordinate per item, rather than using a replication service for the entire system
  - Regular Paxos – performance optimizations aren't relevant for this number of replicas

Toward Solving the Big Data Problems

- Spanner: Google's proprietary approach http://research.google.com/archive/spanner.html
- Paxos between replicas; 2PC for transactions involving multiple Paxos groups

Big Data: Conclusions

- Requirements for large-scale data stores depend on use patterns
- Eventually consistent key-value store approaches work well for read only data and single-row operations
- A more general and complex approach is necessary to provide transactional guarantees across rows
- An open source implementation / realization may be useful