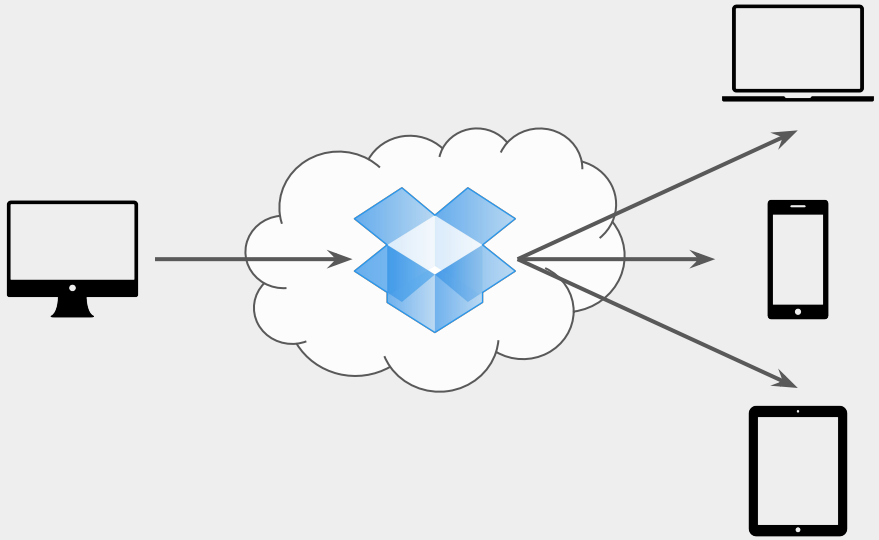
Responsive Consistency in User-Centric Dynamic Network Environments

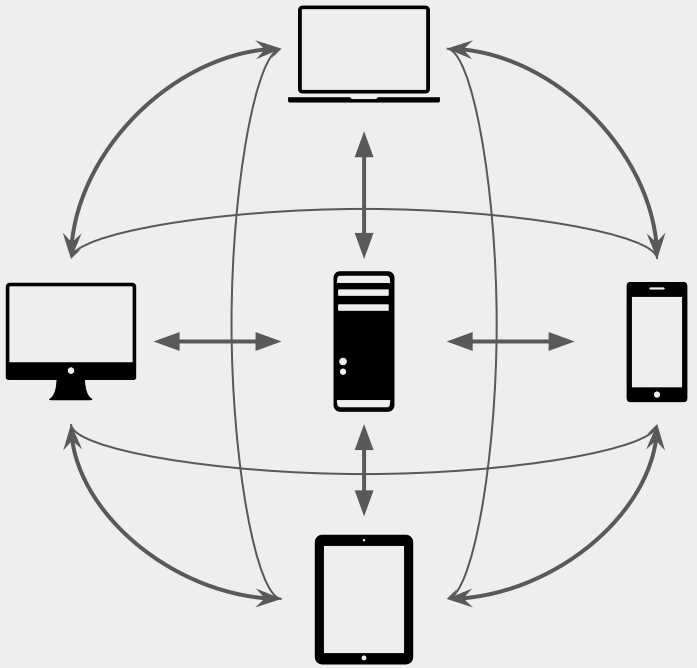
Benjamin Bengfort

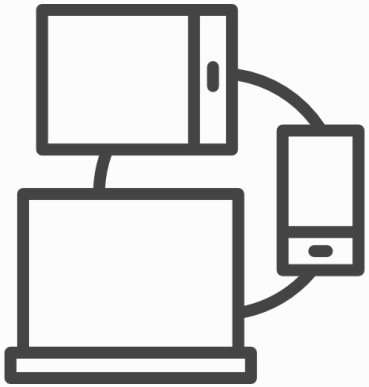
Proposed Research for the Degree of Doctor of Philosophy
December 6, 2016

Can we build a better Dropbox?

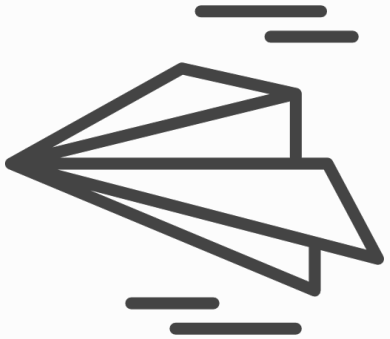
There is a semantic gap between a file system and cloud storage (Zhang et al. 2013)



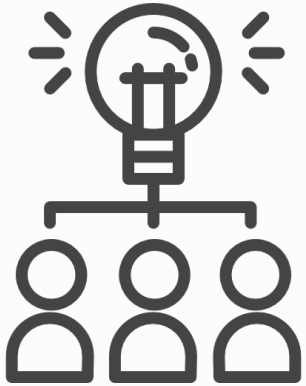




More devices per user.
(Anderson 2015)



Increased connectivity
(Project Loon 2016)



More users connected.
(ITU 2016)

Internet of Things
(Miorandi et al. 2012)

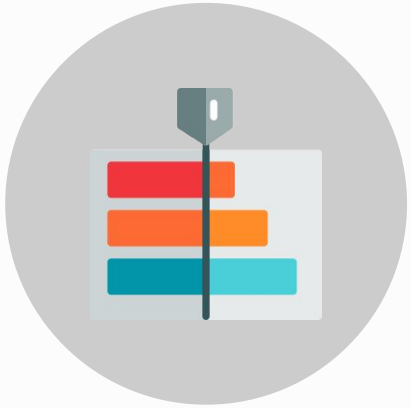
Motivation: Increased Number of Users and Devices

Motivation: Collaboration and Sharing

Clouds do not take advantage of local area networks and always impose wide area bandwidth costs (rate limiting/congestion)

Clouds are untrusted stores with no guarantee of security or privacy
(Feldman et al. 2010)

Vendor lock-in means storage is only as durable as the business that owns it.
(Bessani et al. 2013)



Users expect writes to files to create a linear, sequential ordering of versions, creating a version history.

Even if the network becomes temporarily unavailable, users should still be able to access the file system.

Conflicts will occur in a multi-user system; file system conflicts must be detected and resolved, they cannot be dropped.

Search and Rescue Networks

Interplanetary Networks

Sensor Networks

Temporary Infrastructure

Major Events

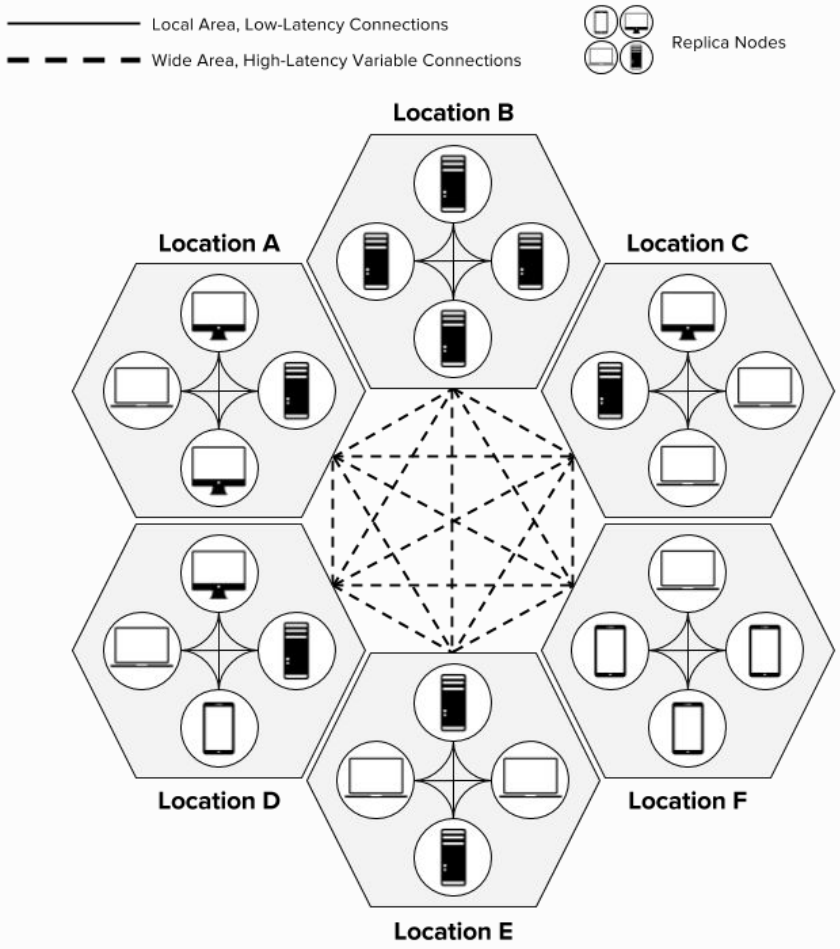
Vehicular Networks

Motivation: Distributed Systems are not only Data Center Applications

User-Centric Dynamic Networks

Characterized as:

- Geographically distributed
- Highly variable latency and bandwidth
- Heterogenous devices
- Partition and failure prone
- Mobile: devices move
- Dynamic: topology and membership changes

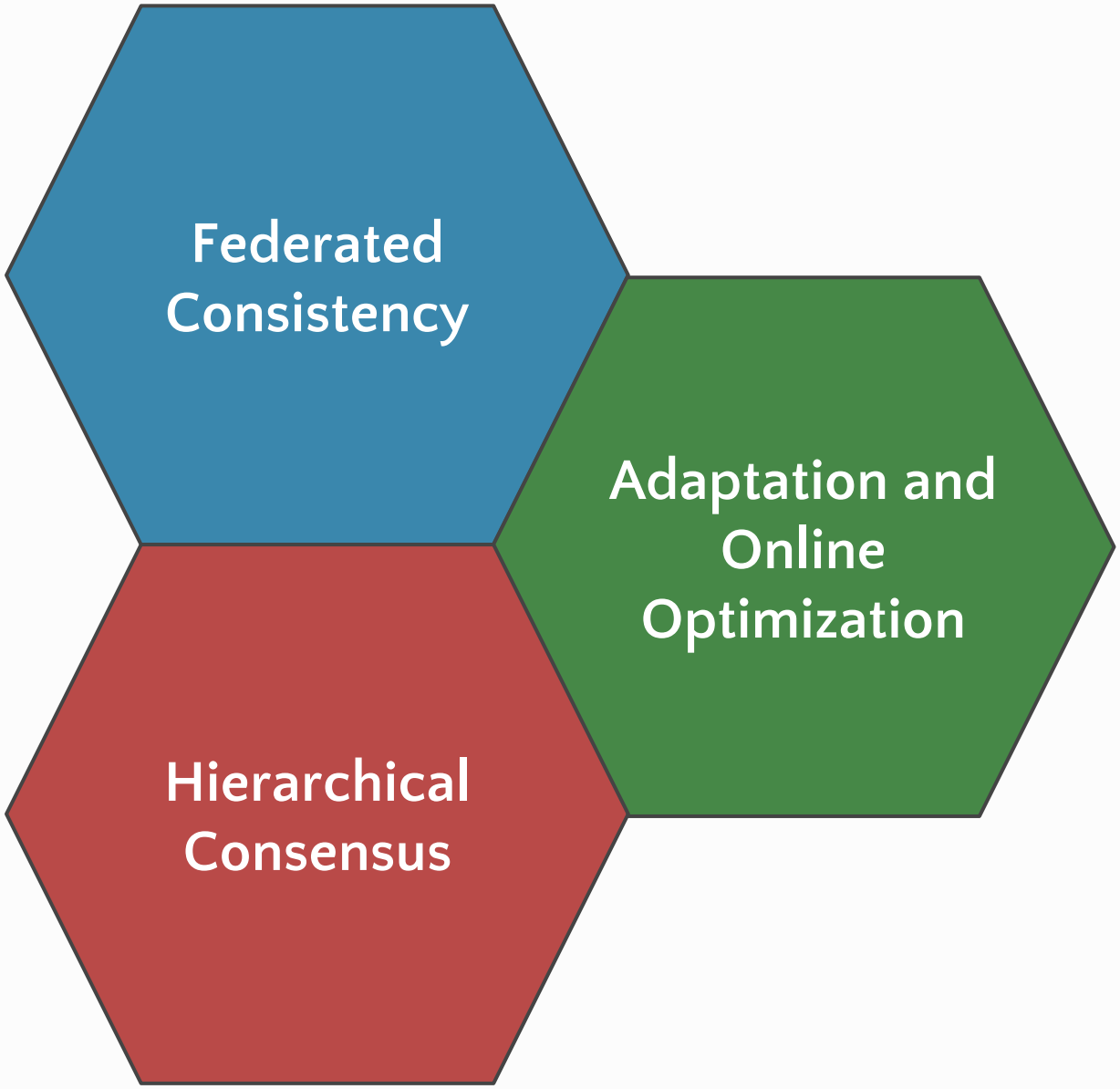


We contend that consistency depends on the environment.

A New Consistency Context is Required

In data centers with very low latencies, eventual consistency is "consistent enough".

(Bailis et al. 2012; Bailis et al. 2014; Bermbach and Tai 2011)



A Responsive Consistency Model

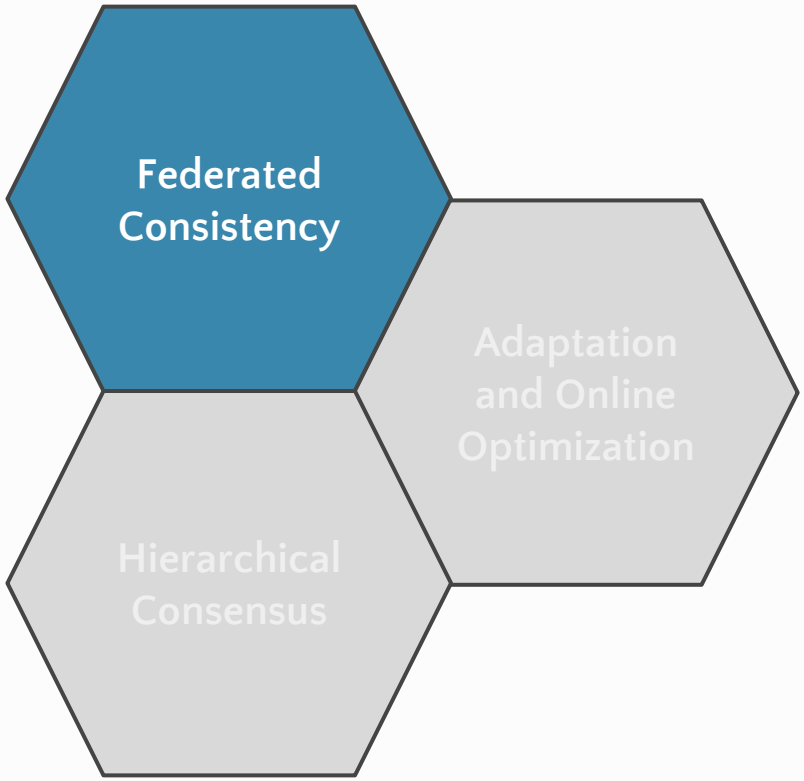
Federated Consistency

Hybridization of consistency on a per-replica basis:

- Weakly consistent but highly available replicas
- Strongly consistent replicas

And potentially anything in between.

The system becomes flexible to select consistency as environment demands.

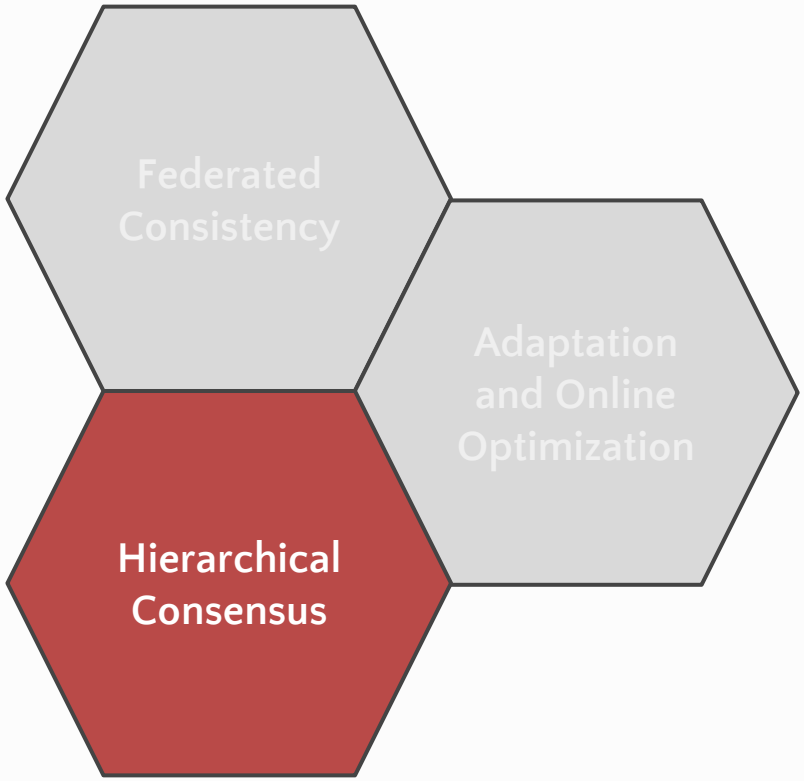


Hierarchical Consensus

Improve scalability, availability, and fault tolerance of traditional consensus:

- Tiered structure of coordination
- Localized decision making
- Disjoint quorums

Flexible consensus: quorums can be allocated based on accesses not contiguous blocks of data.

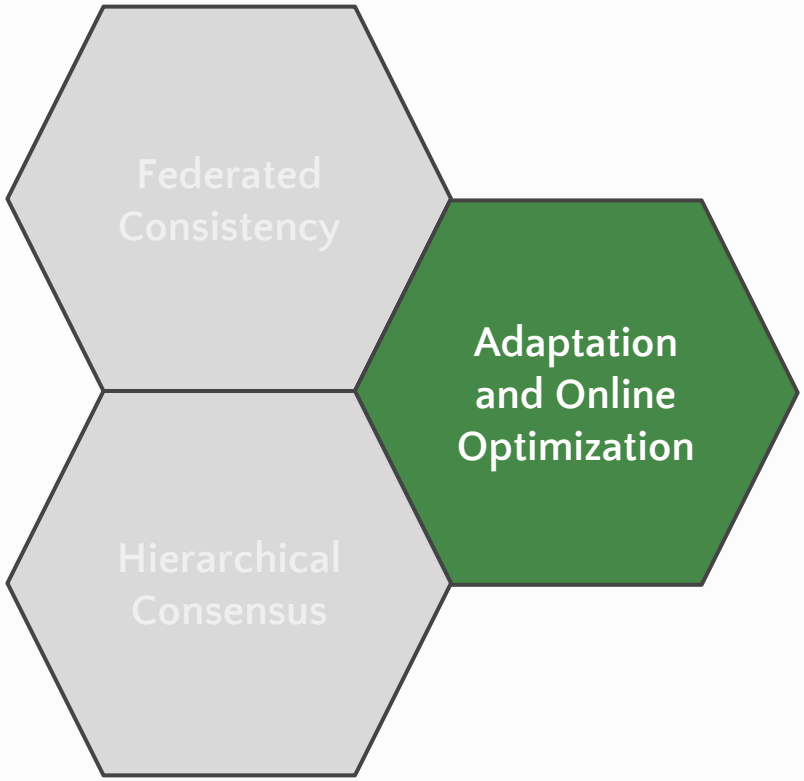


Adaptive Consistency

Both federated consistency and hierarchical consensus make the system *flexible*.

The system can be directly tuned to best performance for the environment.

By including online monitoring of the environment and user accesses, the system can automatically adapt using learning mechanisms.



The combination of federated consistency with hierarchical consensus will provide higher availability and stronger correctness guarantees in user-centric dynamic networks.

Proposal: A Replicated File System (FlowFS)

File system will utilize our responsive consistency model as a substrate.

Operates over a wide area for multiple users.

Implements sequential ordering of Close-to-Open consistency (Howard et al. 1988).

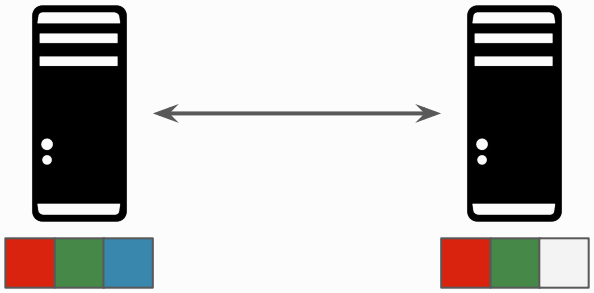
Evaluate through simulation in order to ask challenging topological questions.

Evaluate implementation for real-life workloads.

Compare to homogenous consistency models.

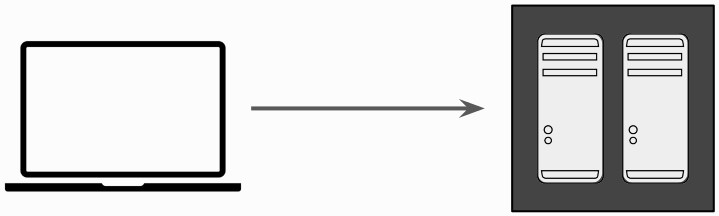
Consistency Model

Data vs. Client Oriented Consistency



Data Oriented

Concerned with the difference in state between replicas.



Client Oriented

Guarantees to clients who view system as a black box.

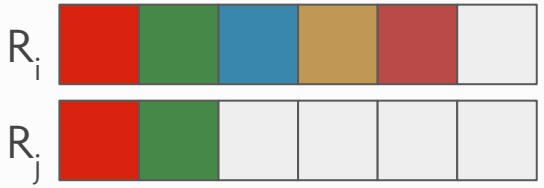
(Bermbach and Kuhlenkamp 2013)

Generalizing Consistency

- Replicas are state machines that append commands to a log.
- Correctness: all replicas reach the same, consistent state.
- Goal: get close to "real-time" ordering of all commands.
- Consistency is not discrete, but rather a *continuum* defined by ordering and staleness dimensions.

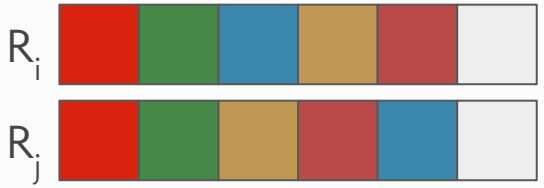
Staleness

Describes how far behind a replica is from the global state.



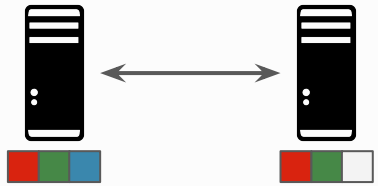
Ordering

Defines by how much logs can vary the sequence of commands.



(Bermbach and Kuhlenkamp 2013; Yu and Vahdat 2002)

Data Centric Consistency Levels



Strictness of Ordering

Eventual Consistency (EC)

Updates applied in any order if system converges on a single state given no accesses. (DeCandia et al. 2007)

Causal Consistency (CC)

Update can only be applied if dependencies have already been applied. (Lloyd et al. 2011)

Sequential Consistency (SC)

All replicas must apply the same exact sequence of updates, no log deviation. (Lamport 1978)

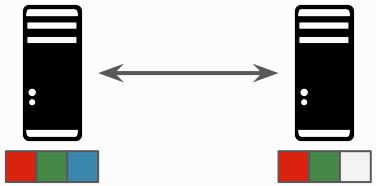
Linearizability (LIN)

All replicas must apply the same sequence of updates *and* no reads can be stale. (Herlihy and Wing 1990)

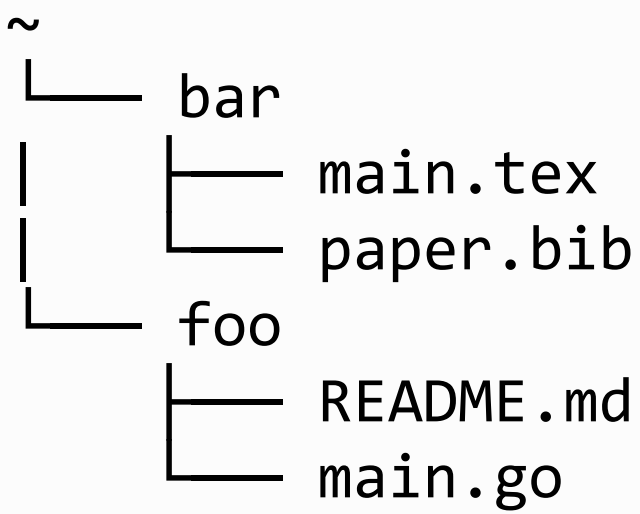
FlowFS Consistency Range

Too Strict

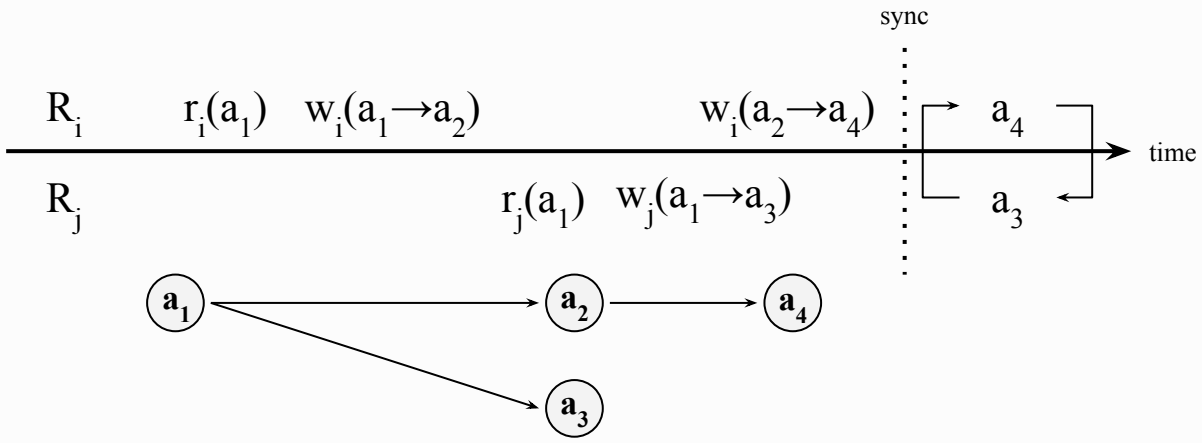
File System Consistency: Accesses



- Close-to-open Consistency:
  - read on open
  - write on close
- File access is *atomic*
- Open guaranteed to see "most recent" close.
- Consistency is the sequential ordering of opens to closes.



File System Consistency: Forks



- The "state" of a replica is defined by its *view* of the file system: the set of most recent versions for all known objects.
- A consistent view requires a linear version history.
- A fork is two concurrent writes to the same object, which is inconsistent because there are now two possible orderings to the log.

No aggregation of accesses, no transactions leads to forks.

Stale Reads

The replication protocol has not yet made the latest version available locally.

True Conflict

Concurrent access to the same object by multiple writers.

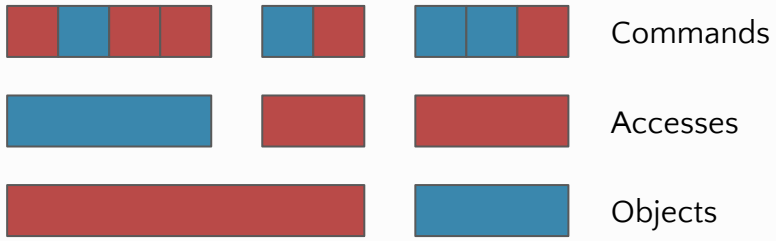
There are two primary causes of forks.

Federated Consistency

Hybridization of Consistency

Consistency rationing or hybrid consistency models blend multiple consistency levels to achieve specific results:

- Per-command consistency level (Li et al. 2012)
- Cassandra per-access consistency level (Lakshman and Malik 2010)
- Per-object/tablet consistency levels (Kraska et al. 2013)



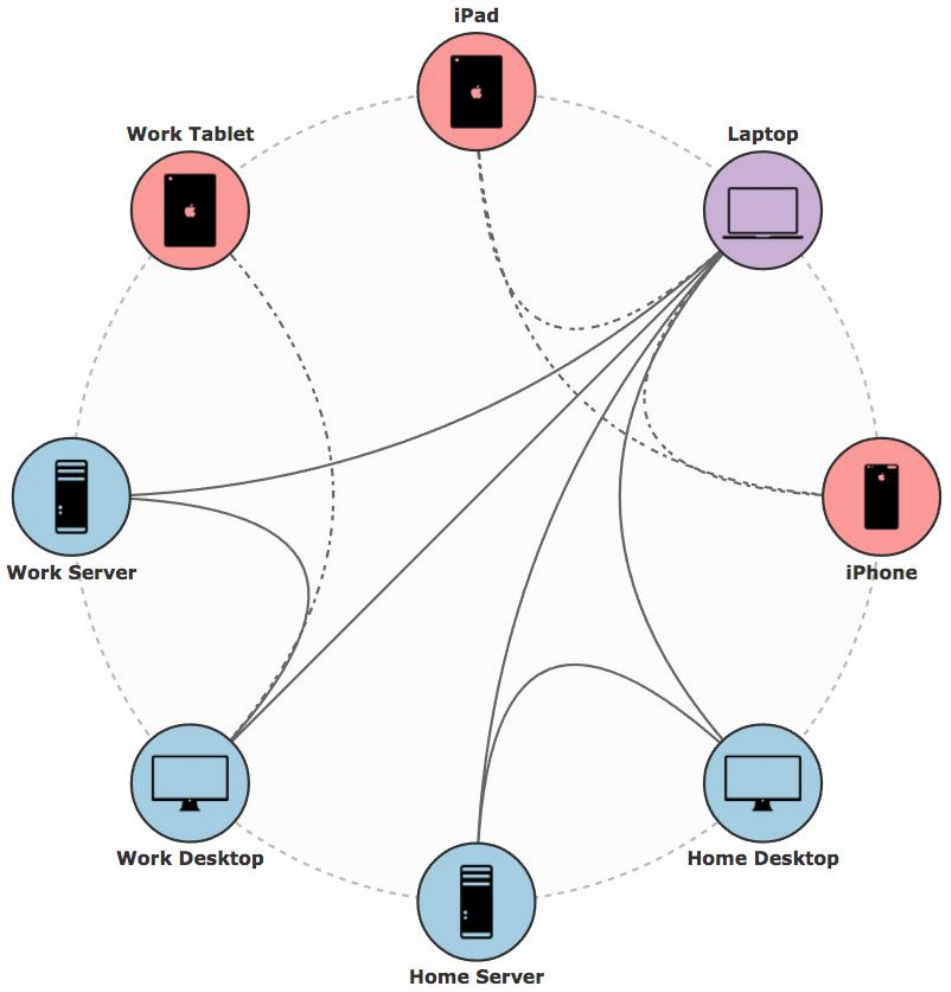
Hybridization: means of defining flexibility.

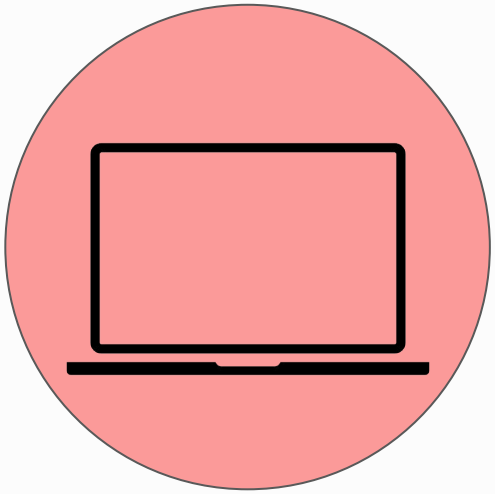
Can be optimized, e.g. to minimize cost of cloud service. (Kraska et al. 2009)

Federated Consistency

A hybridization at the *replica* layer rather than at lower levels.

Replicas set consistency level with policies based on resource and environment constraints.



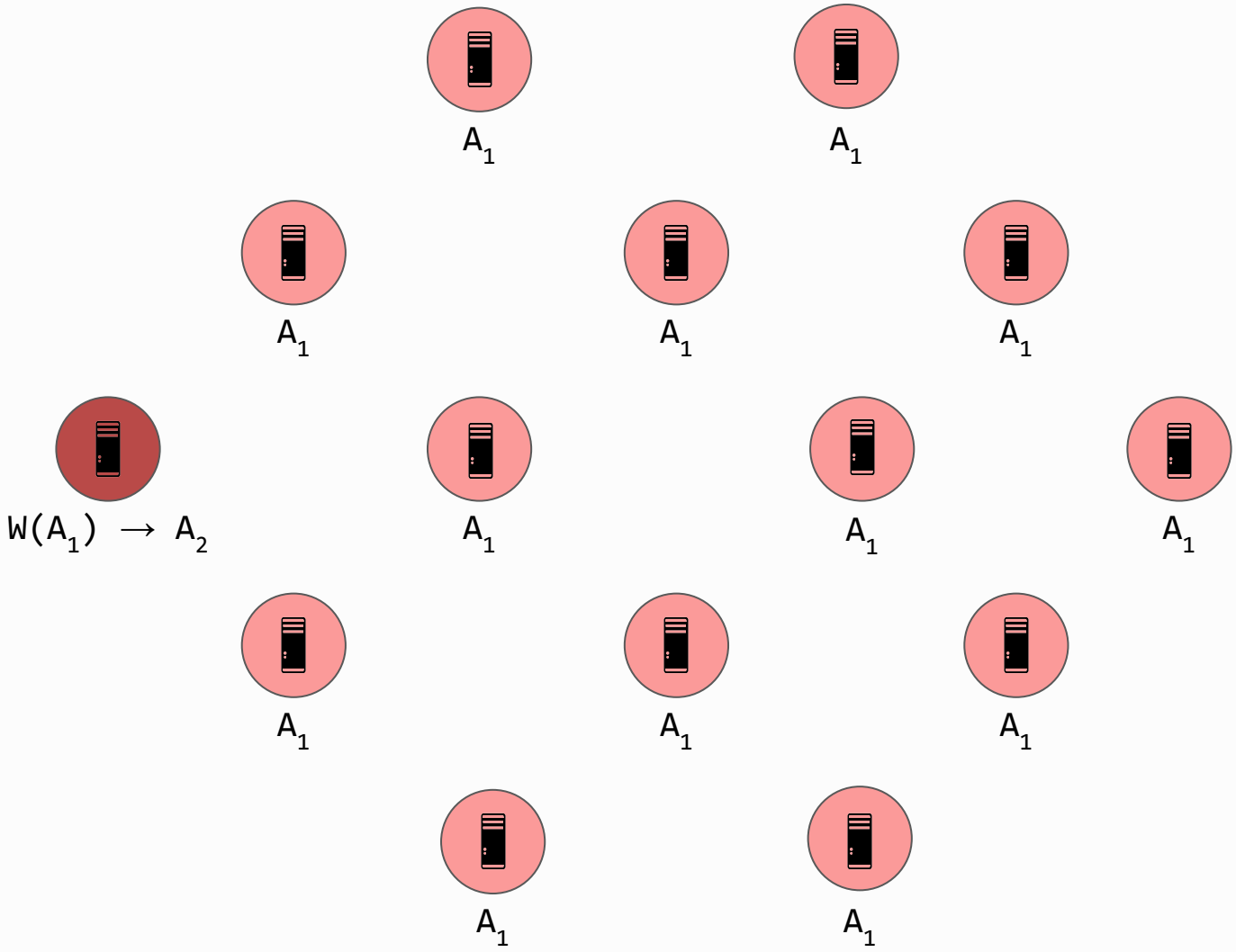


Anti-Entropy

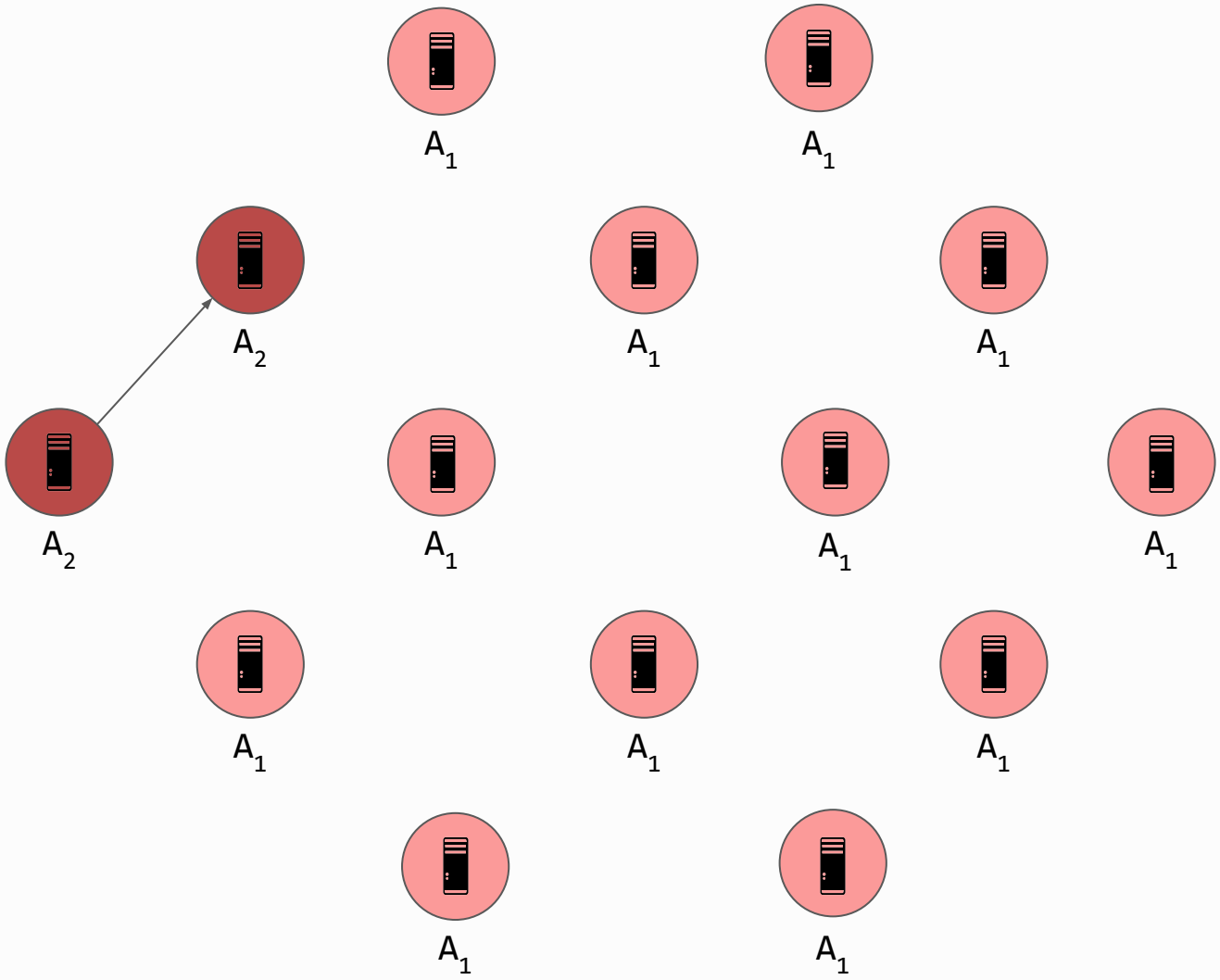
Eventual consistency implemented with gossip protocol and LWW.

Raft Consensus

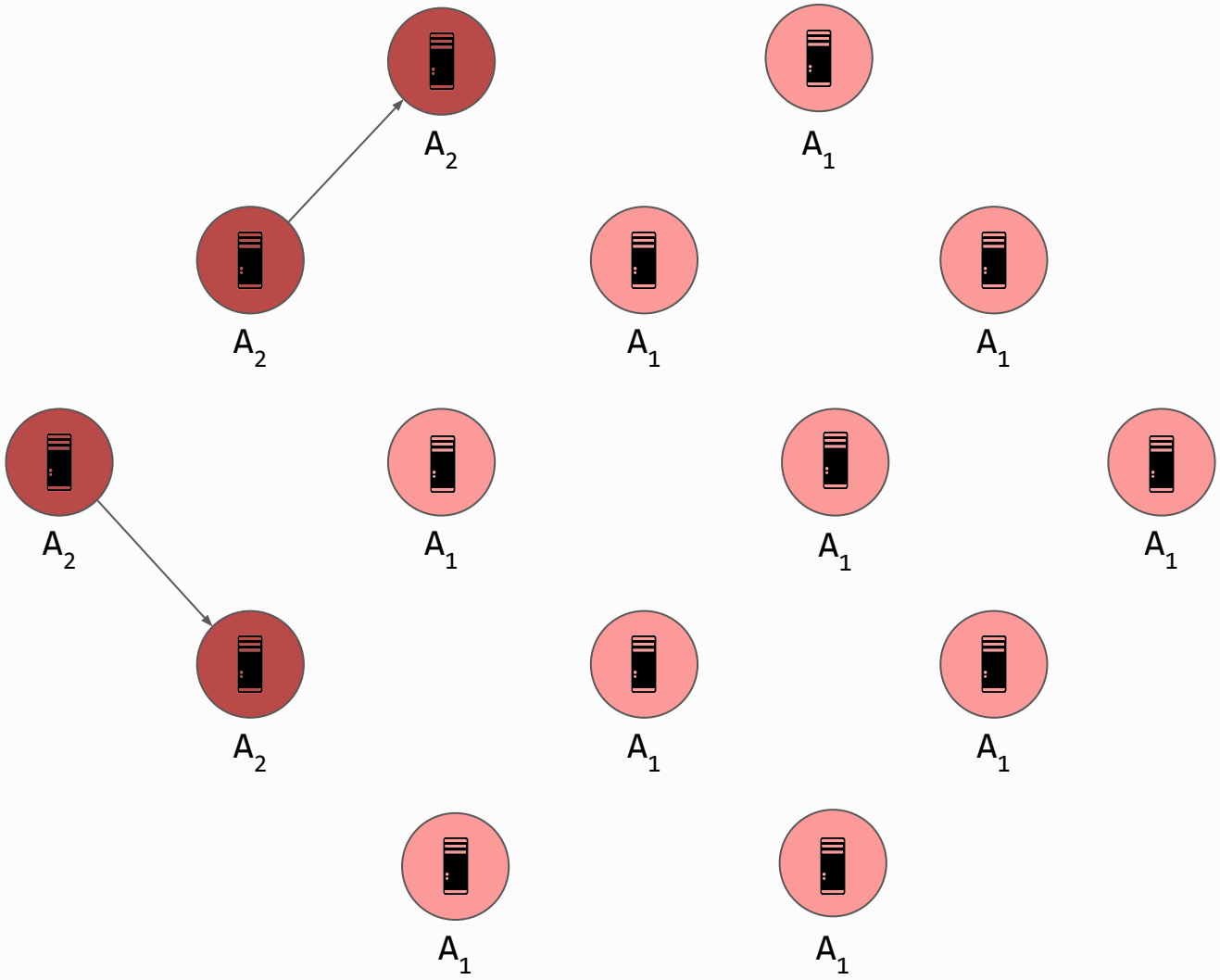
Sequential consistency implemented with consensus based coordination.



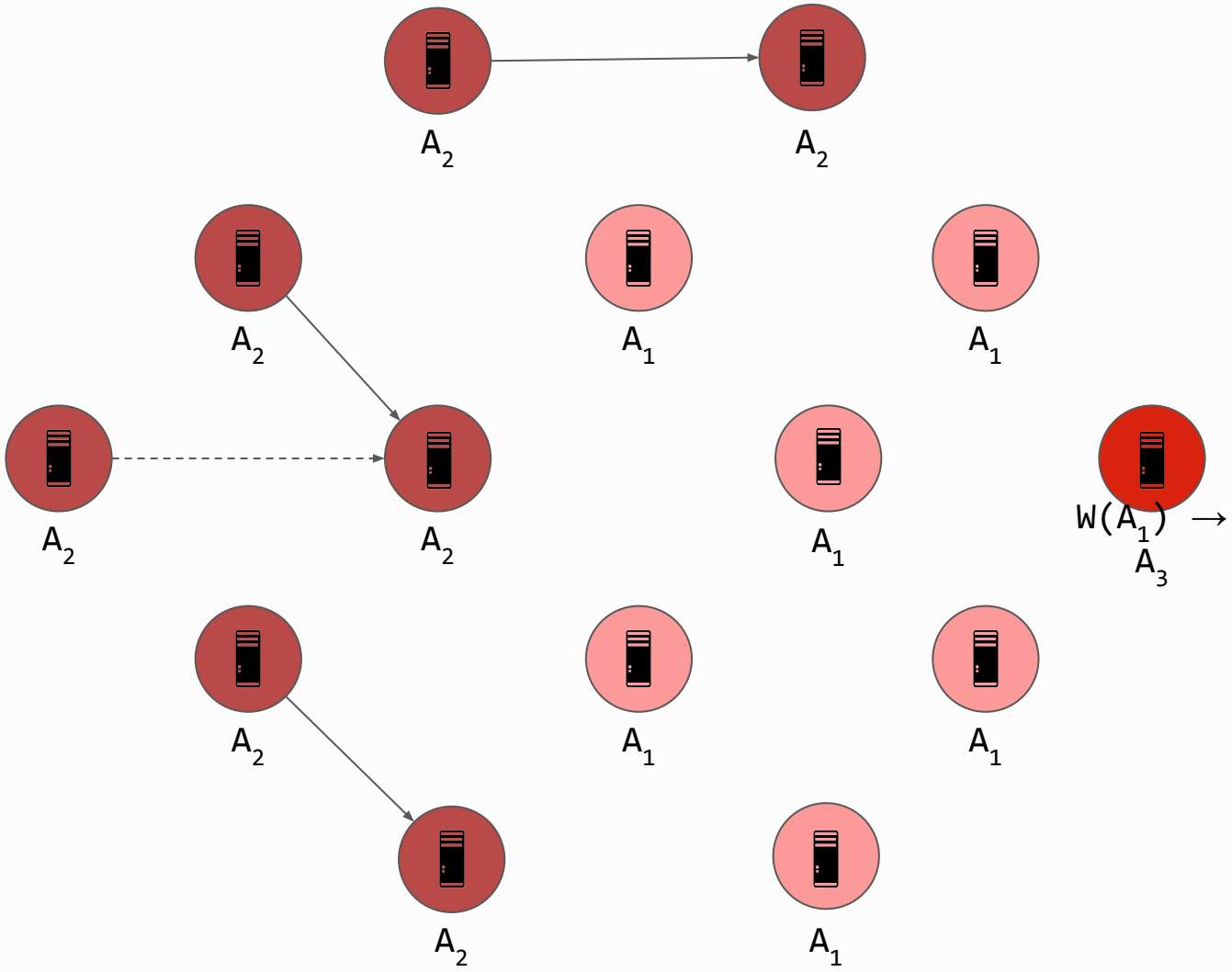
Anti-Entropy Replication: Last Writer Wins (LWW)



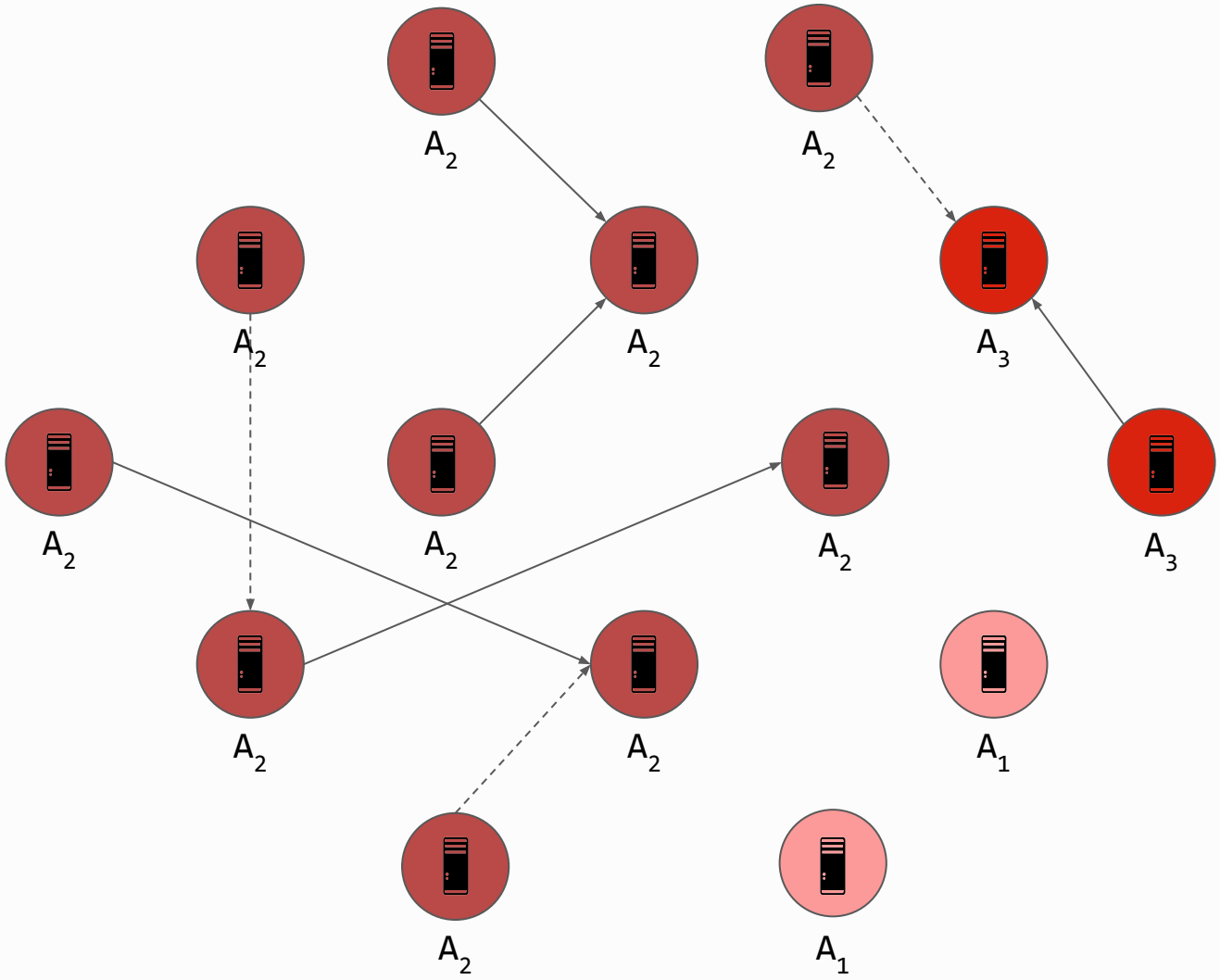
Anti-Entropy Replication: Last Writer Wins (LWW)



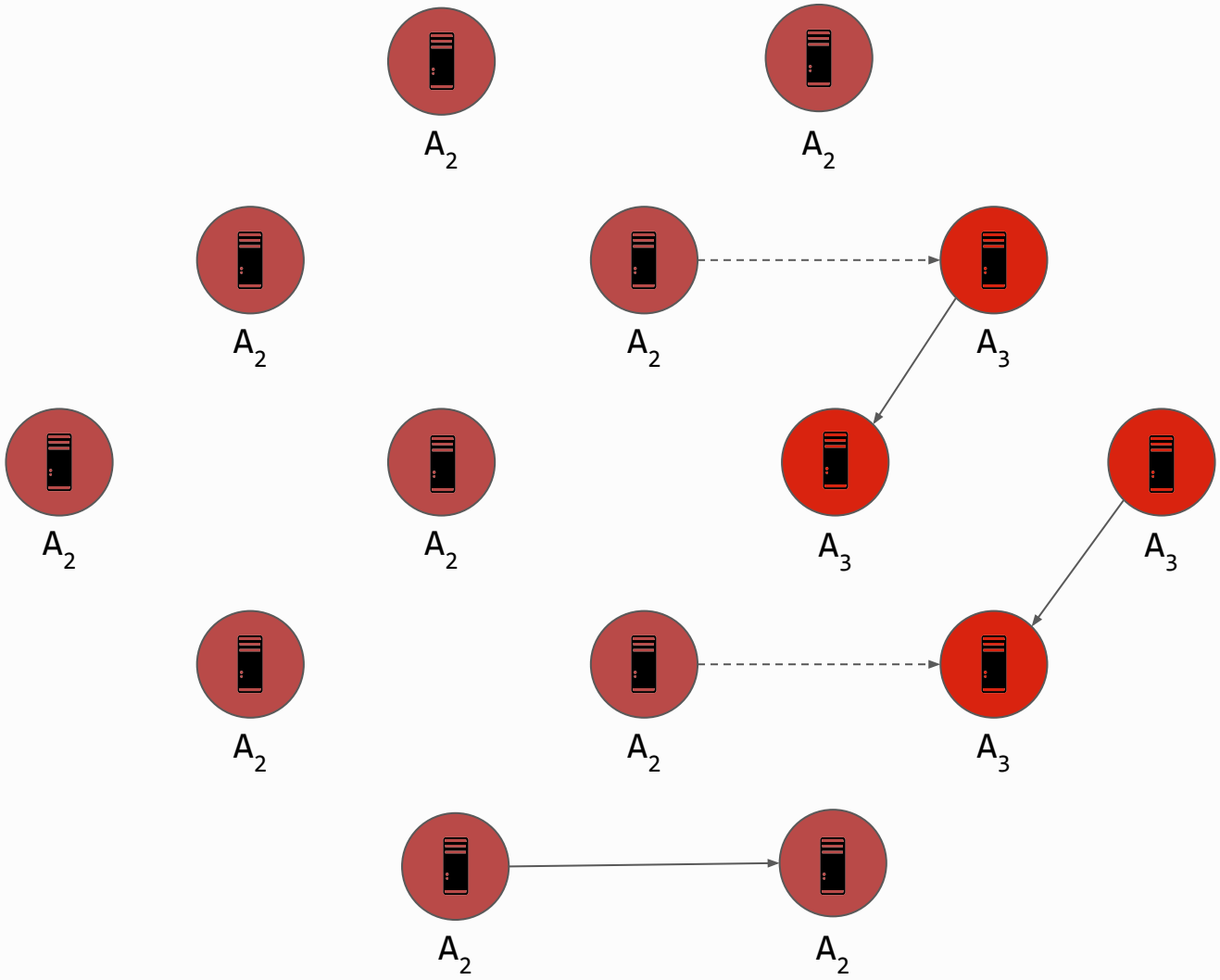
Anti-Entropy Replication: Last Writer Wins (LWW)



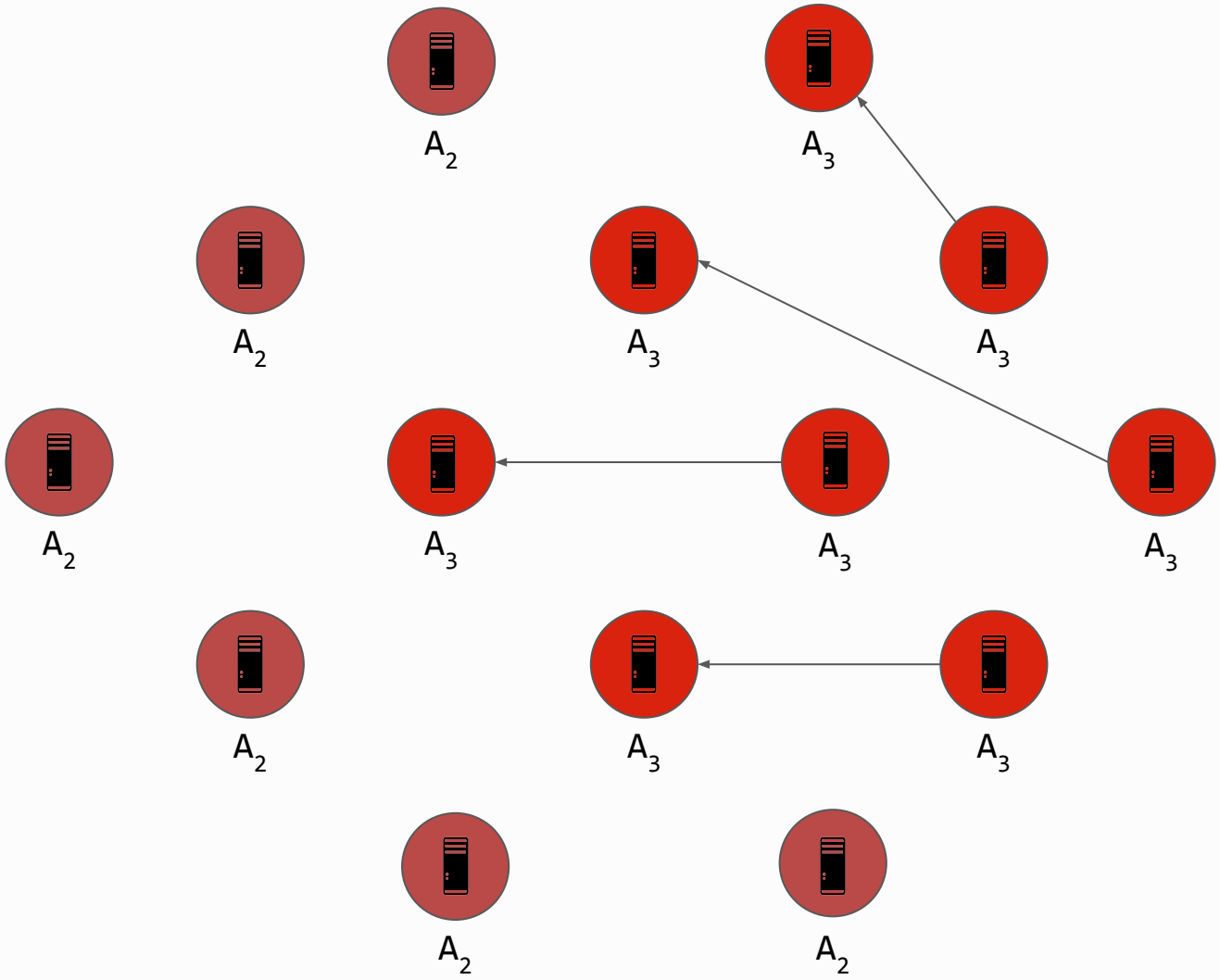
Anti-Entropy Replication: Last Writer Wins (LWW)



Anti-Entropy Replication: Last Writer Wins (LWW)

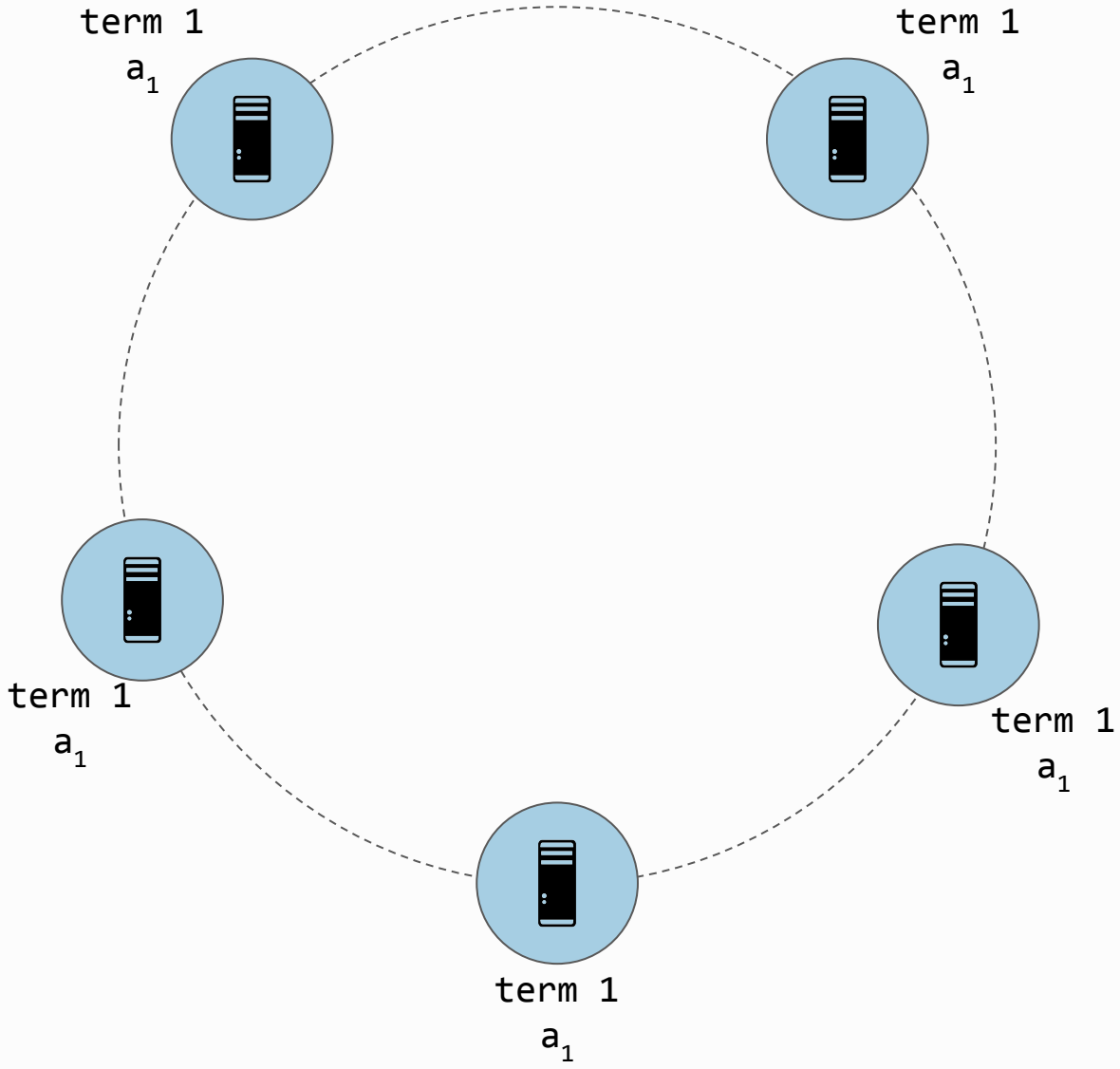


Anti-Entropy Replication: Last Writer Wins (LWW)



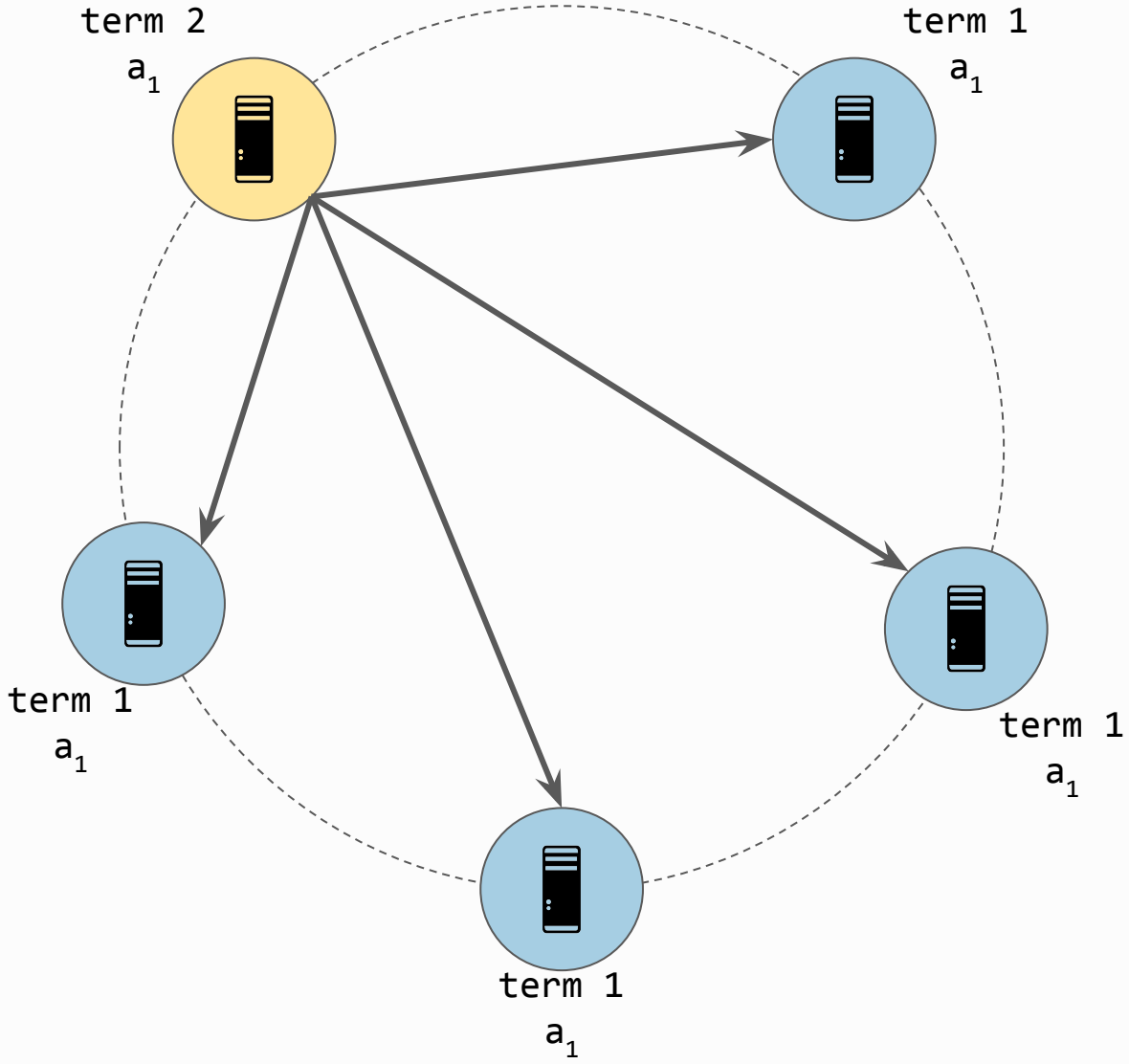
Anti-Entropy Replication: Last Writer Wins (LWW)

Initial State



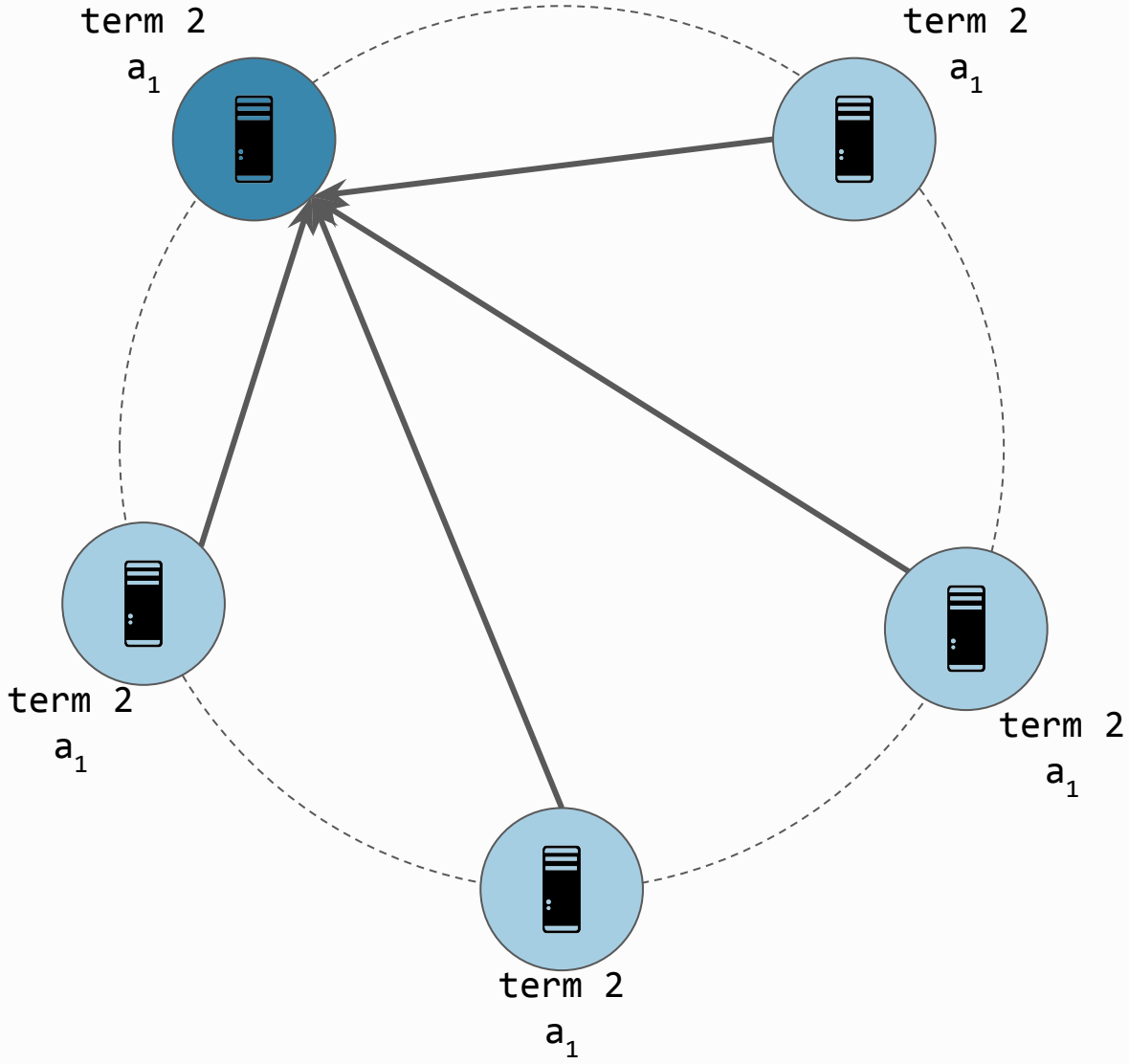
Raft Consensus in a Nutshell: Accept First Writer

Request Vote



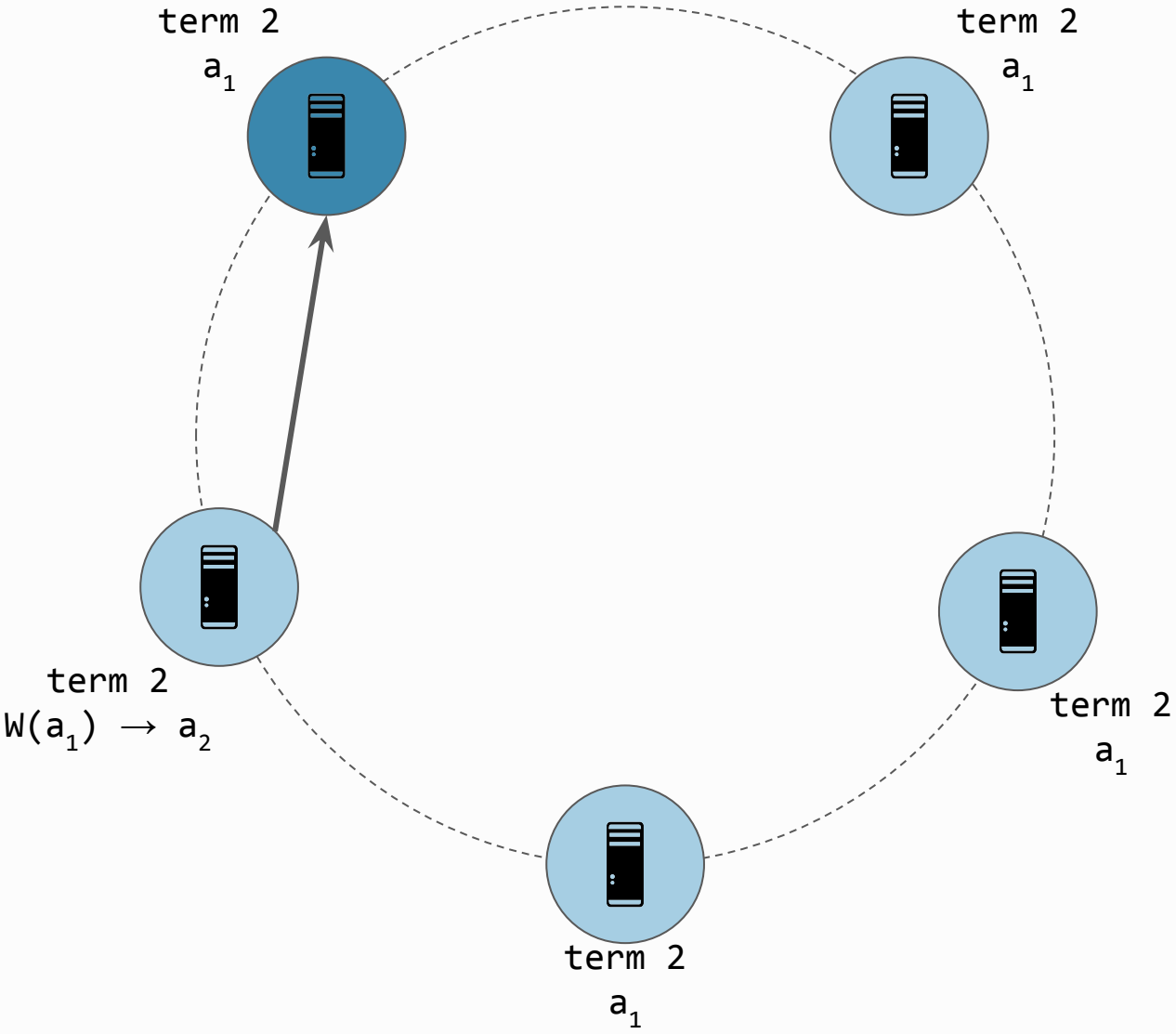
Raft Consensus in a Nutshell: Accept First Writer

Leader Elected



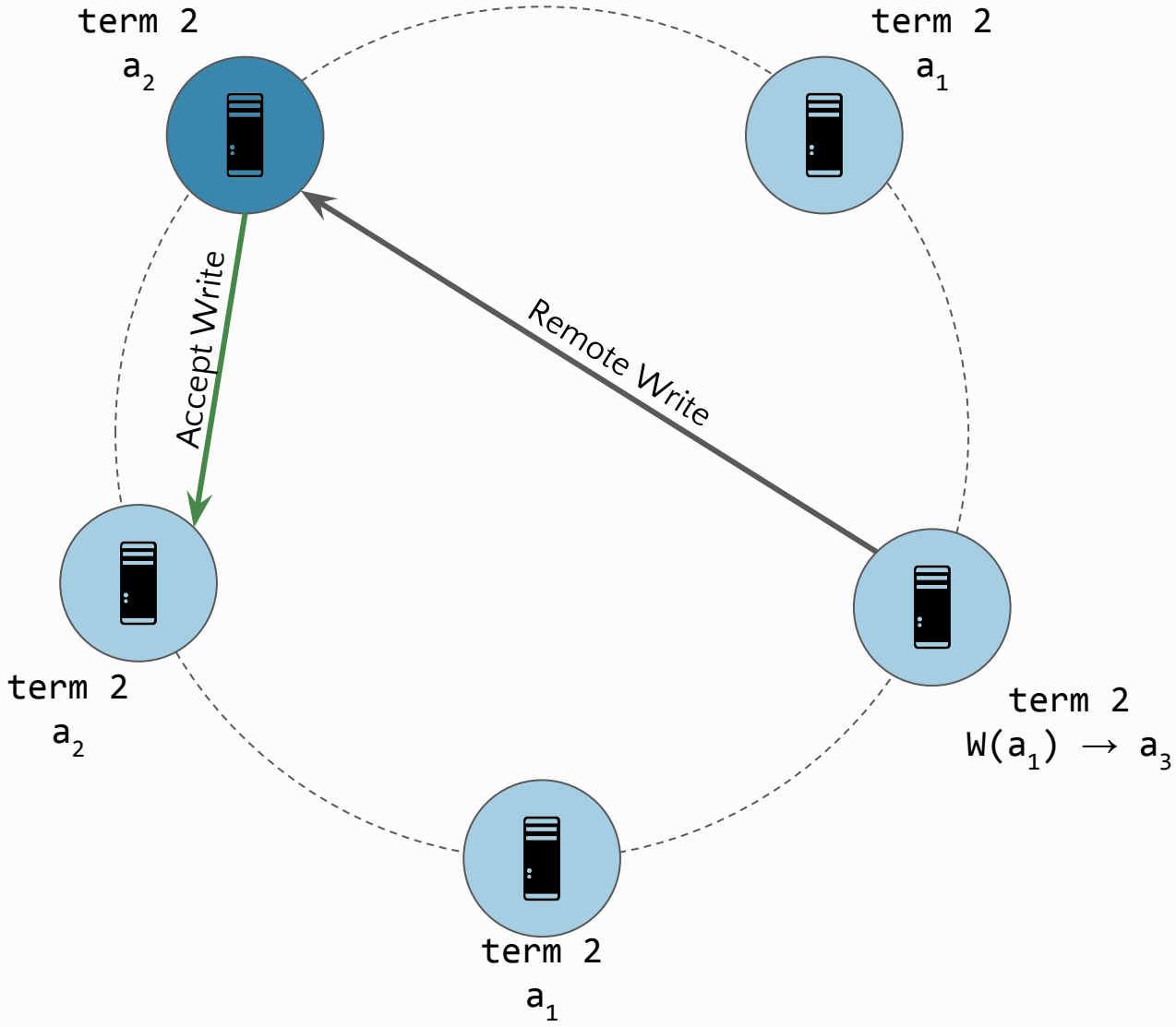
Raft Consensus in a Nutshell: Accept First Writer

Remote Write



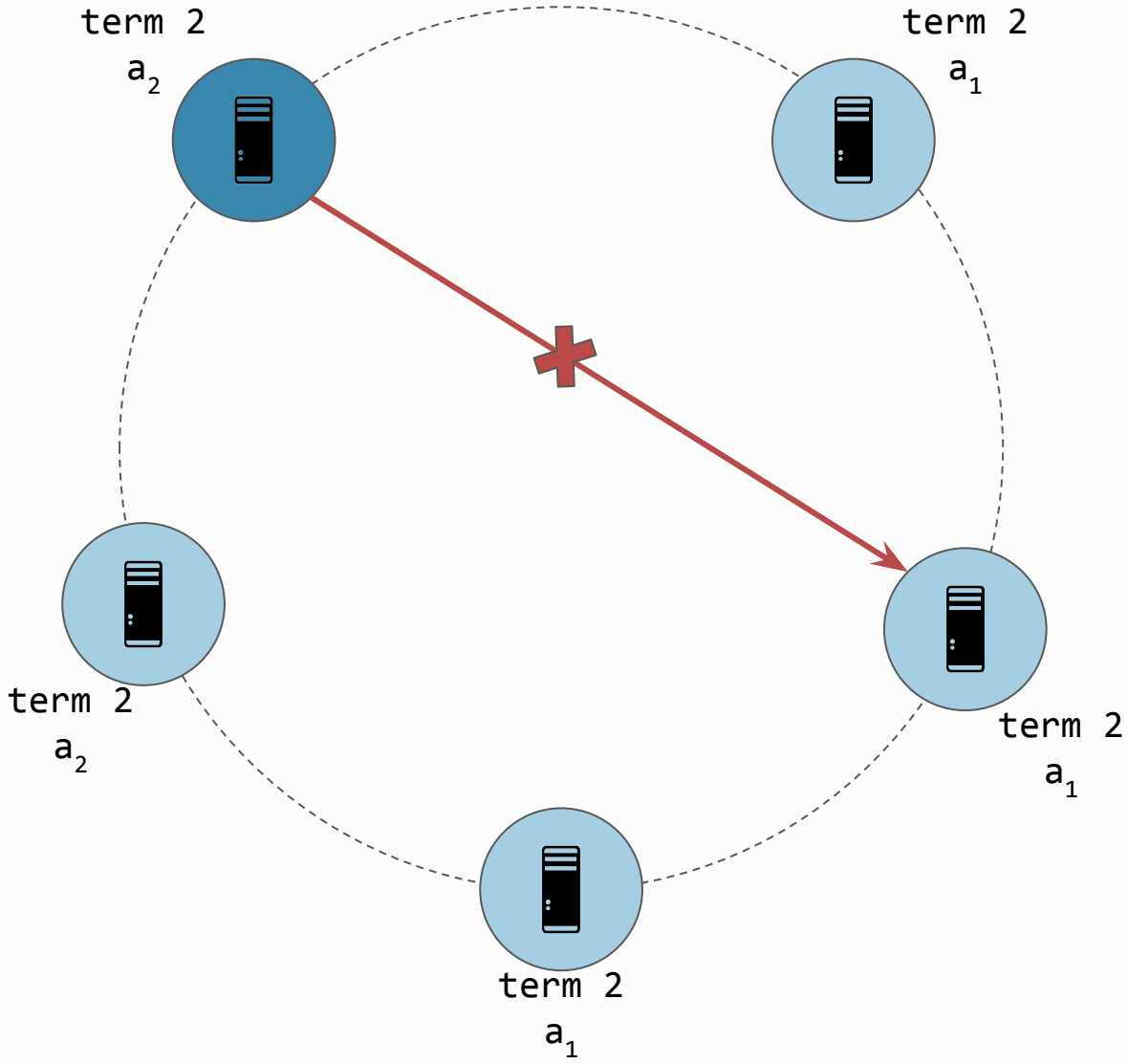
Raft Consensus in a Nutshell: Accept First Writer

Write Response



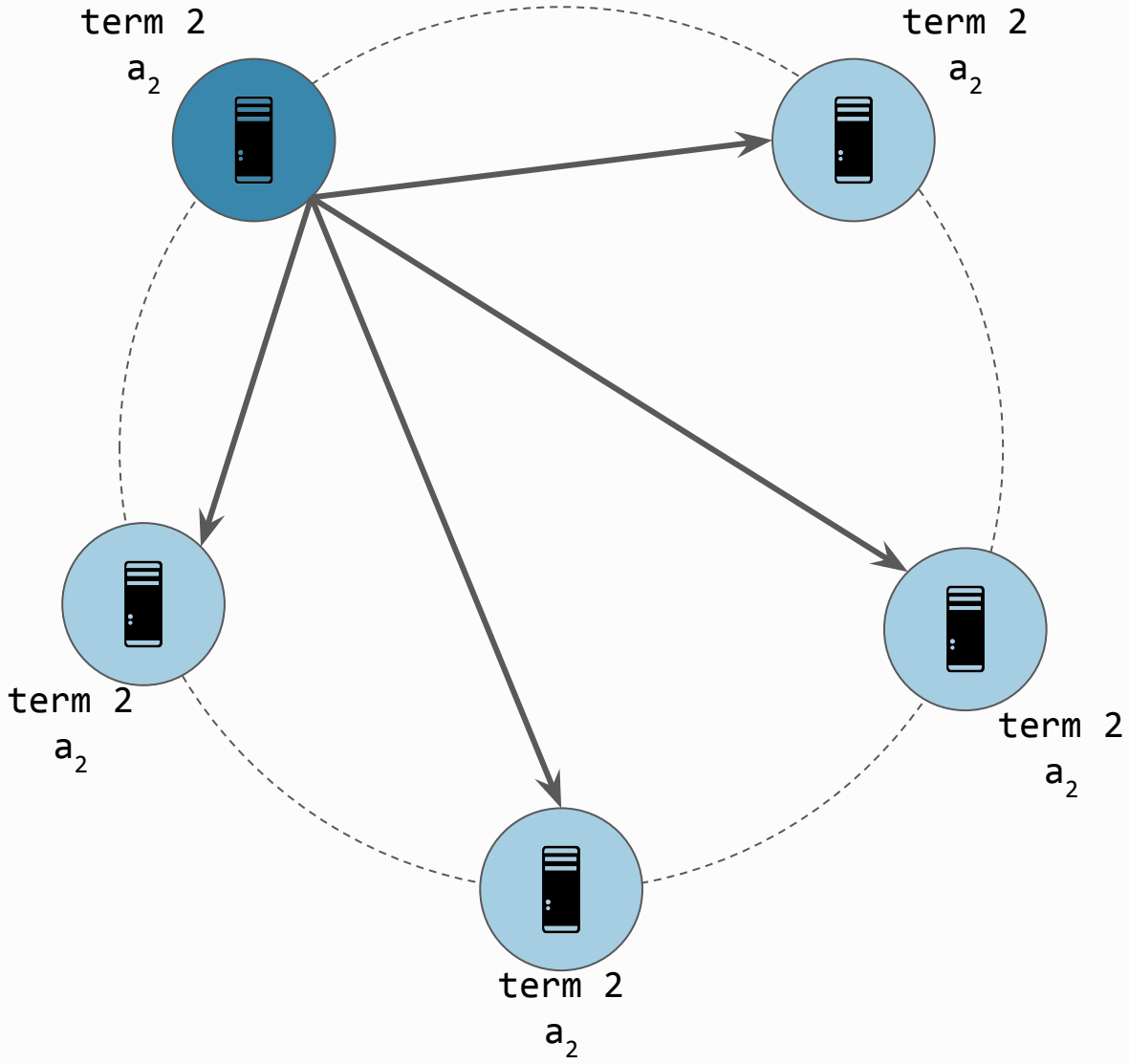
Raft Consensus in a Nutshell: Accept First Writer

Leader Rejects Forks



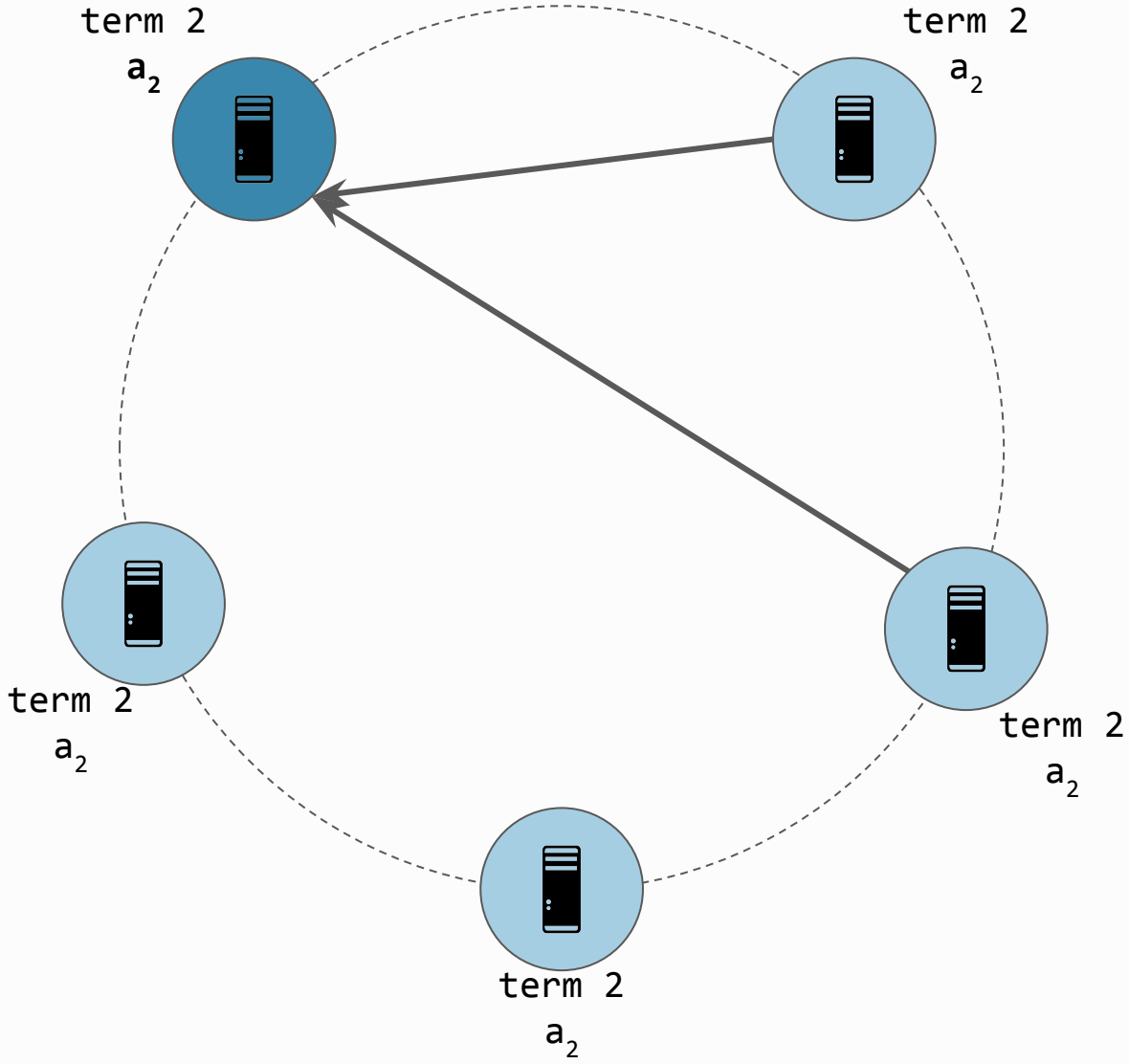
Raft Consensus in a Nutshell: Accept First Writer

Append Entries Broadcast



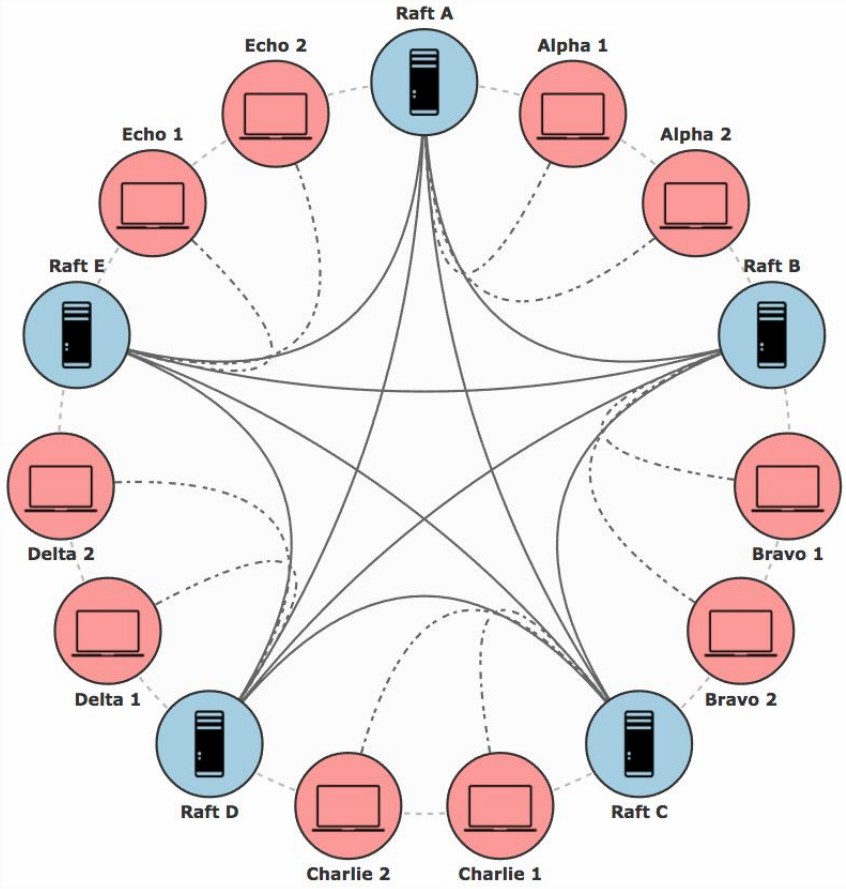
Raft Consensus in a Nutshell: Accept First Writer

Commit on Majority Response



Raft Consensus in a Nutshell: Accept First Writer

Eventually consistent replicas for availability; a strong central quorum for stronger consistency guarantees.



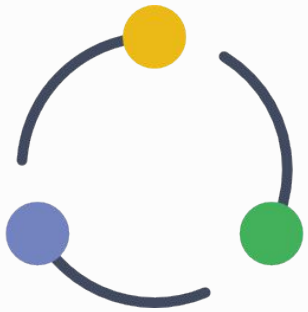
Federating Heterogeneous Systems

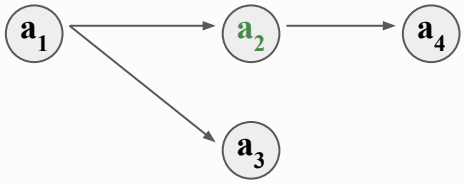
Communication Integration

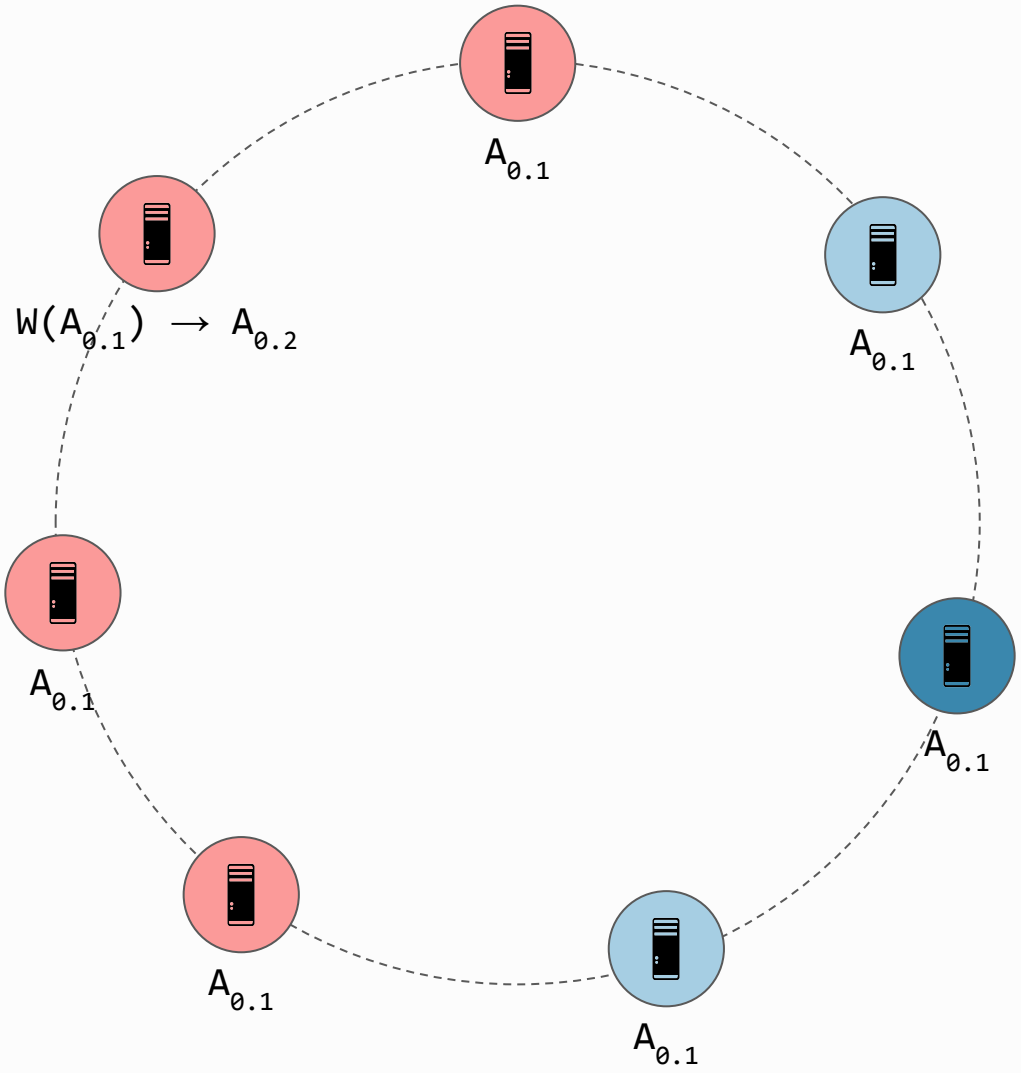
- All replicas respond to all RPC messages according to their local consistency level.
- Anti-Entropy synchronizes Raft followers with Eventual.
- Pairwise selection probabilities for Raft vs. Wide vs. Local.

Consistency Integration

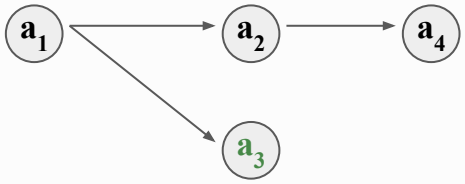
- Last writer wins in eventual; first writer accepted in Raft.
- Raft determines which fork is globally accepted.
- Use of a *forte* number pushes Raft decisions back to eventual cloud.

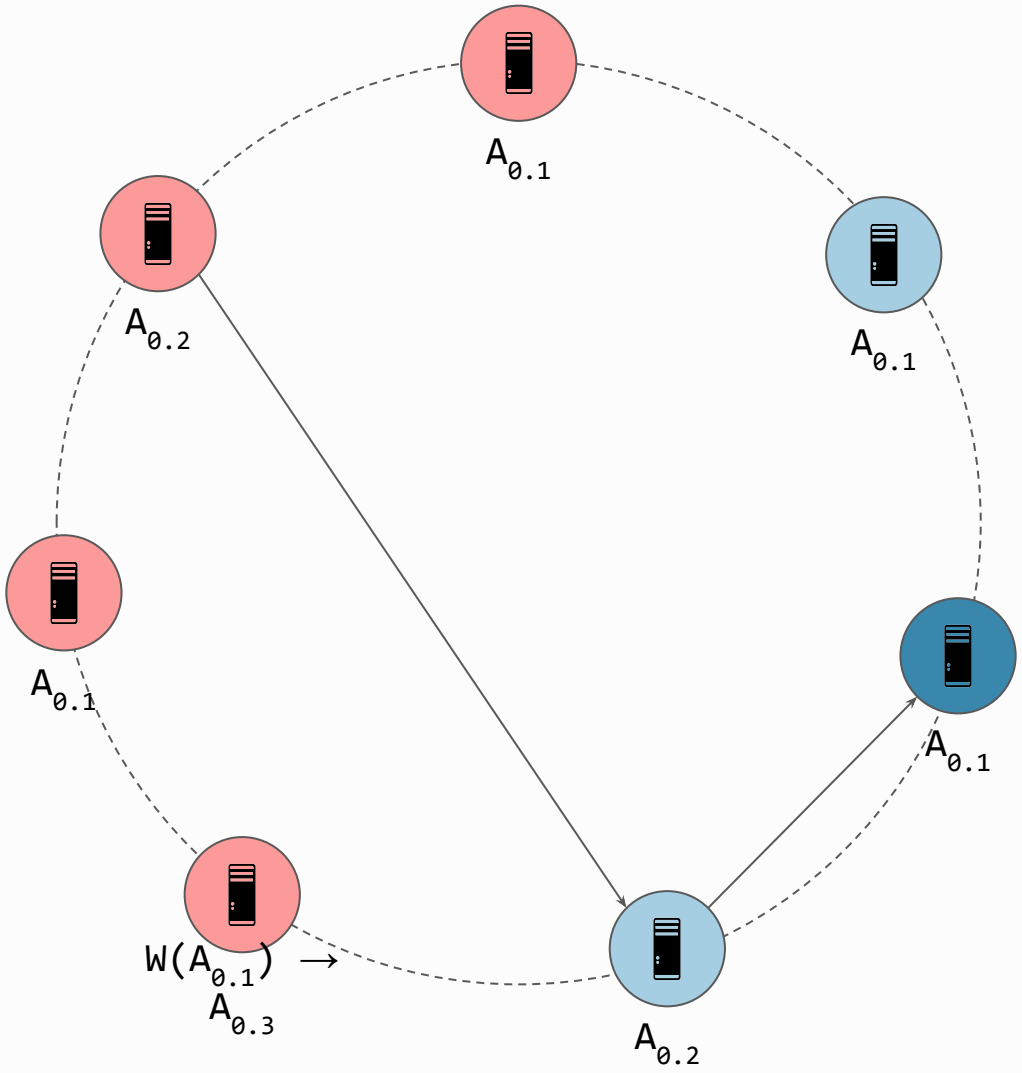




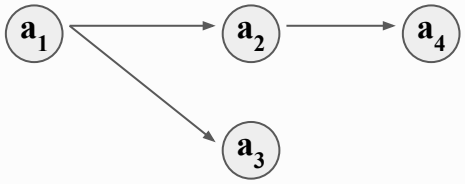


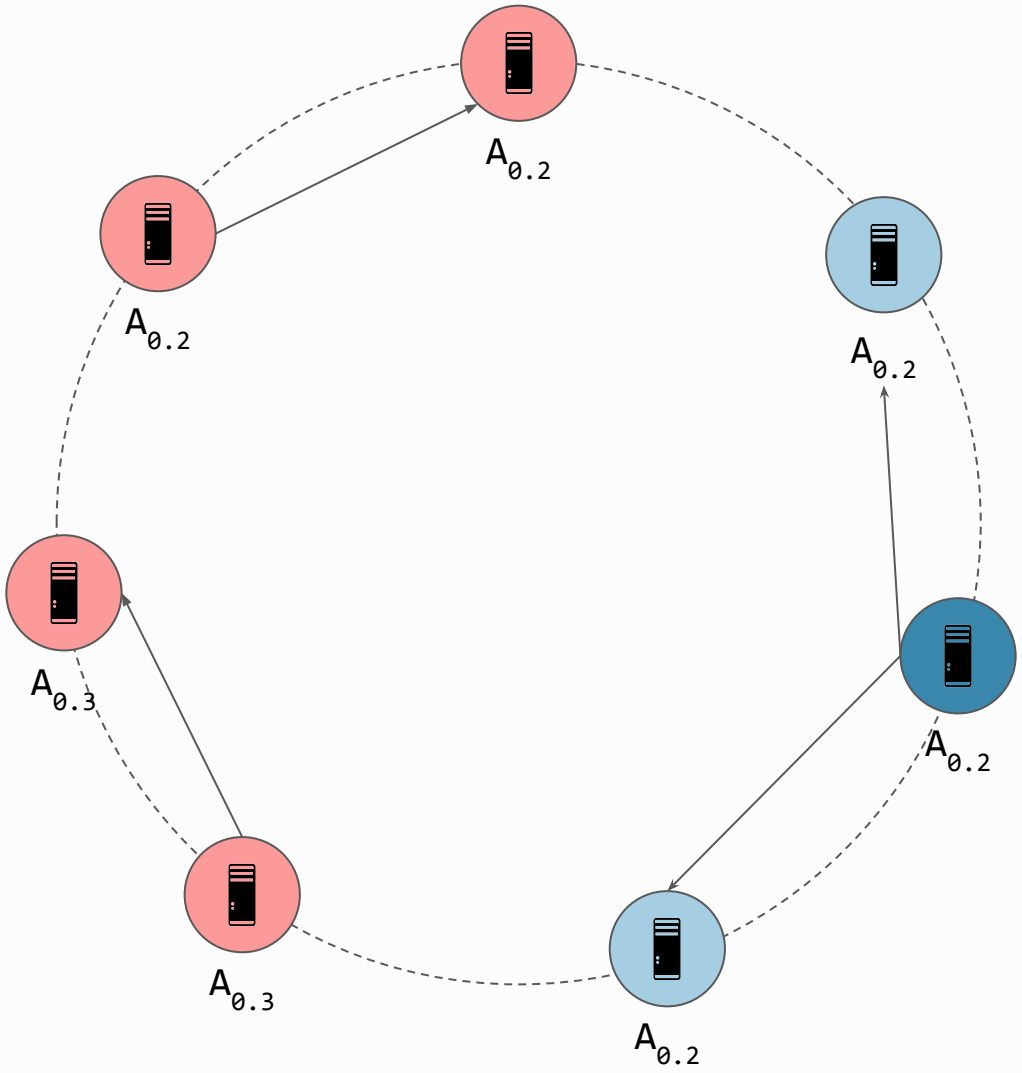
The Forte Number: Initial Write



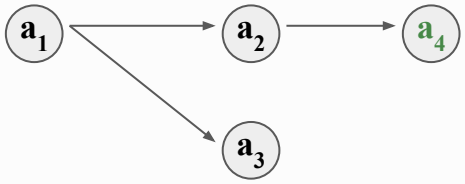


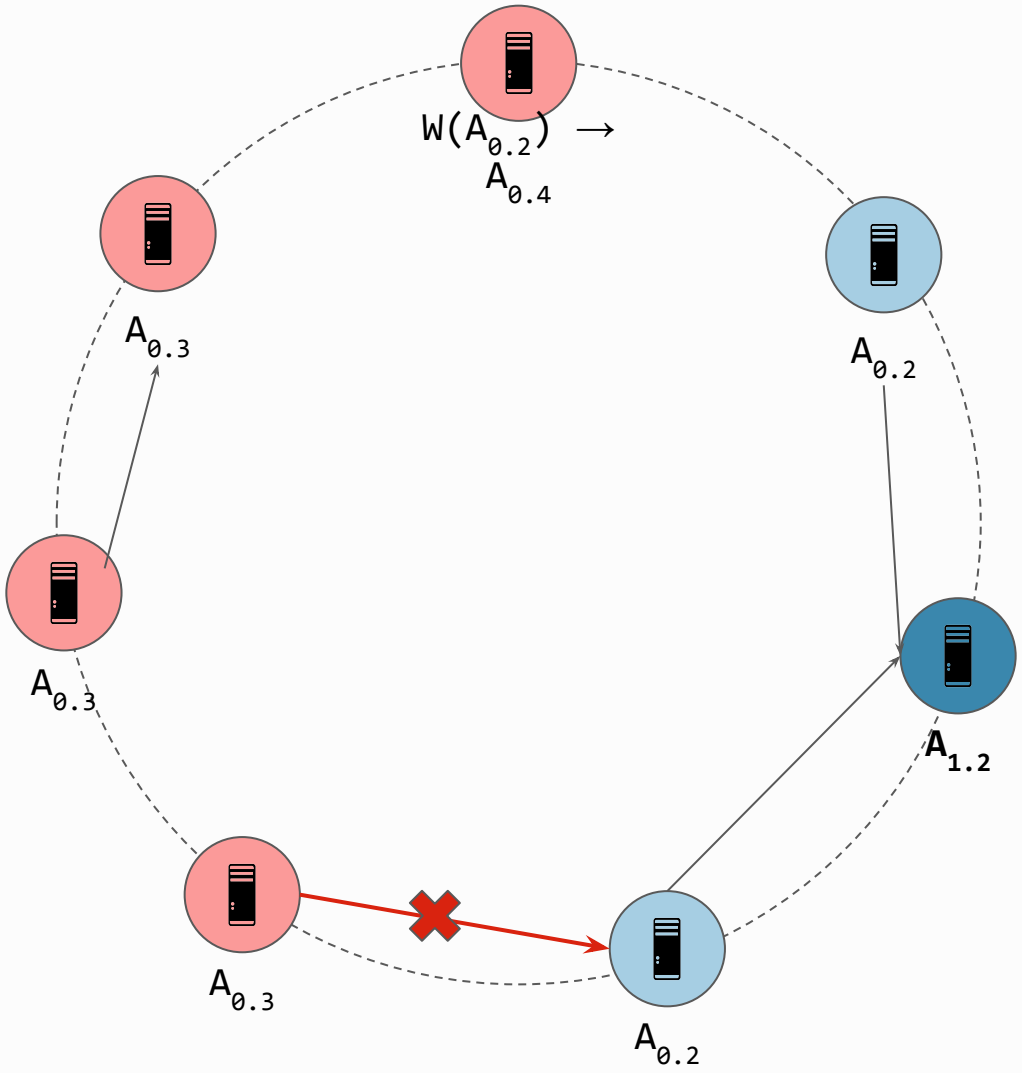
The Forte Number: Anti-Entropy with Raft node → Remote Write



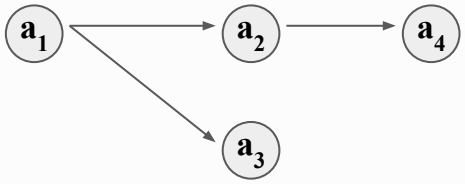


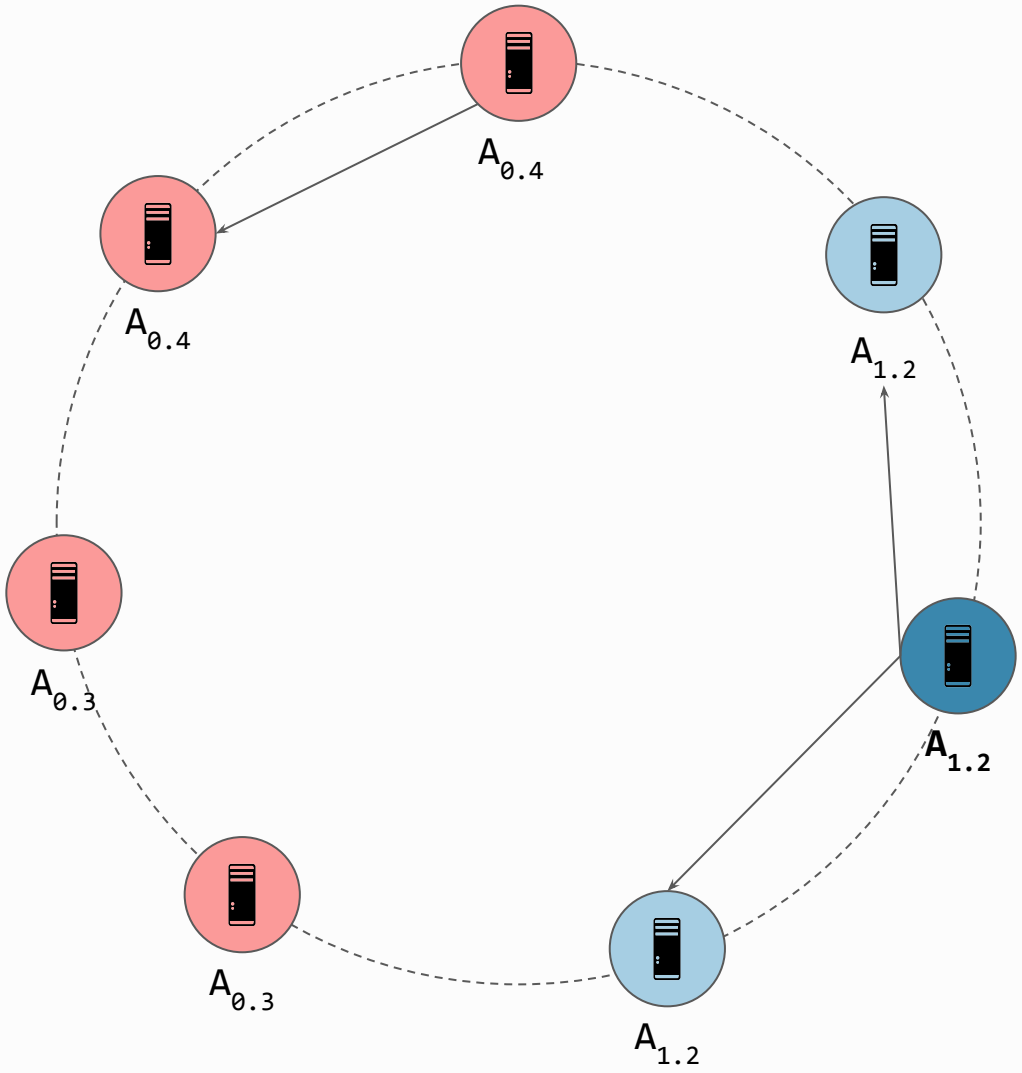
The Forte Number: Replication via Anti-Entropy and Append Entries



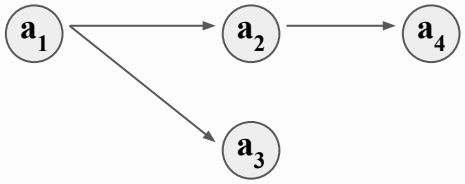


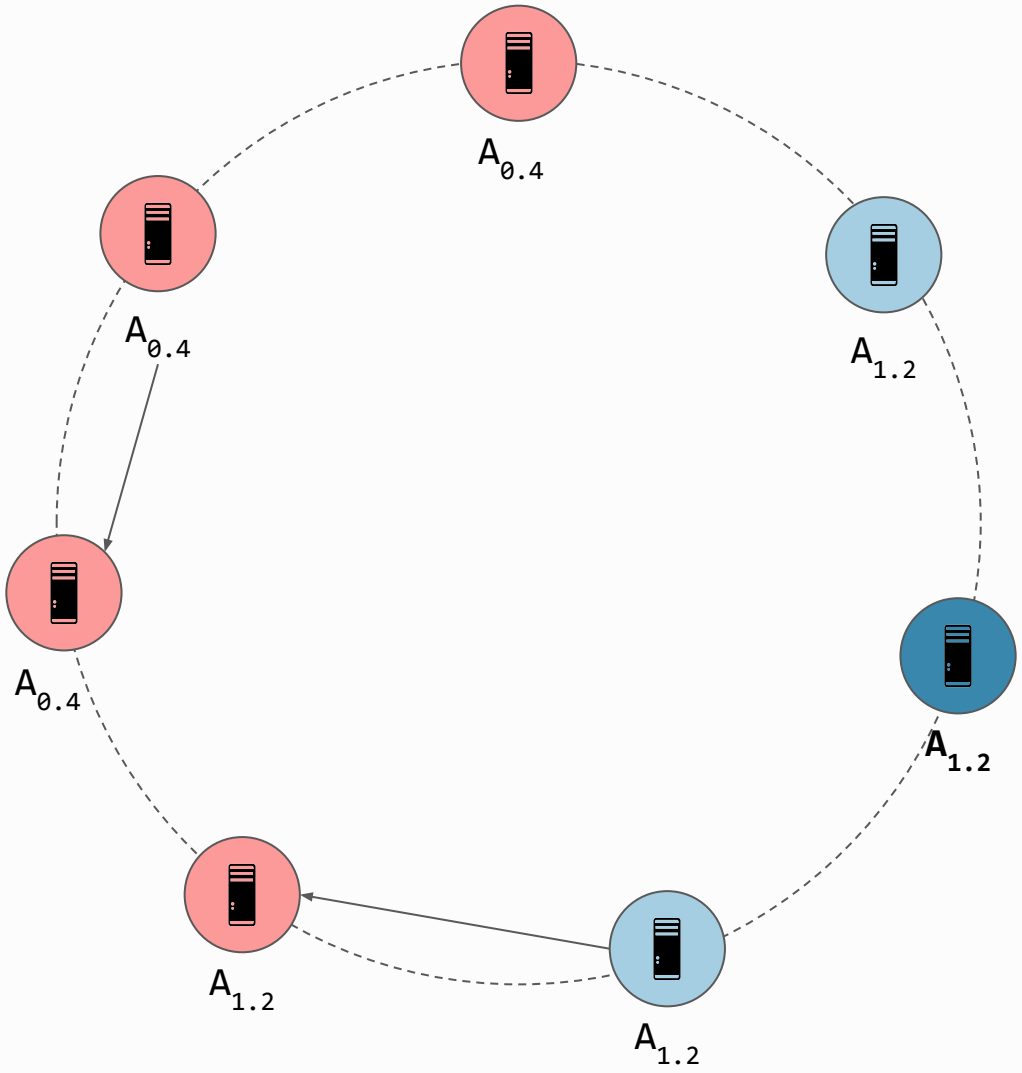
The Forte Number Incremented on Commit



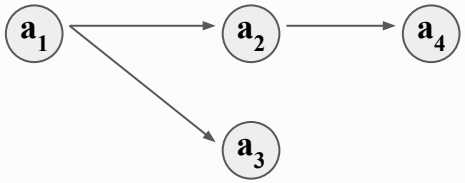


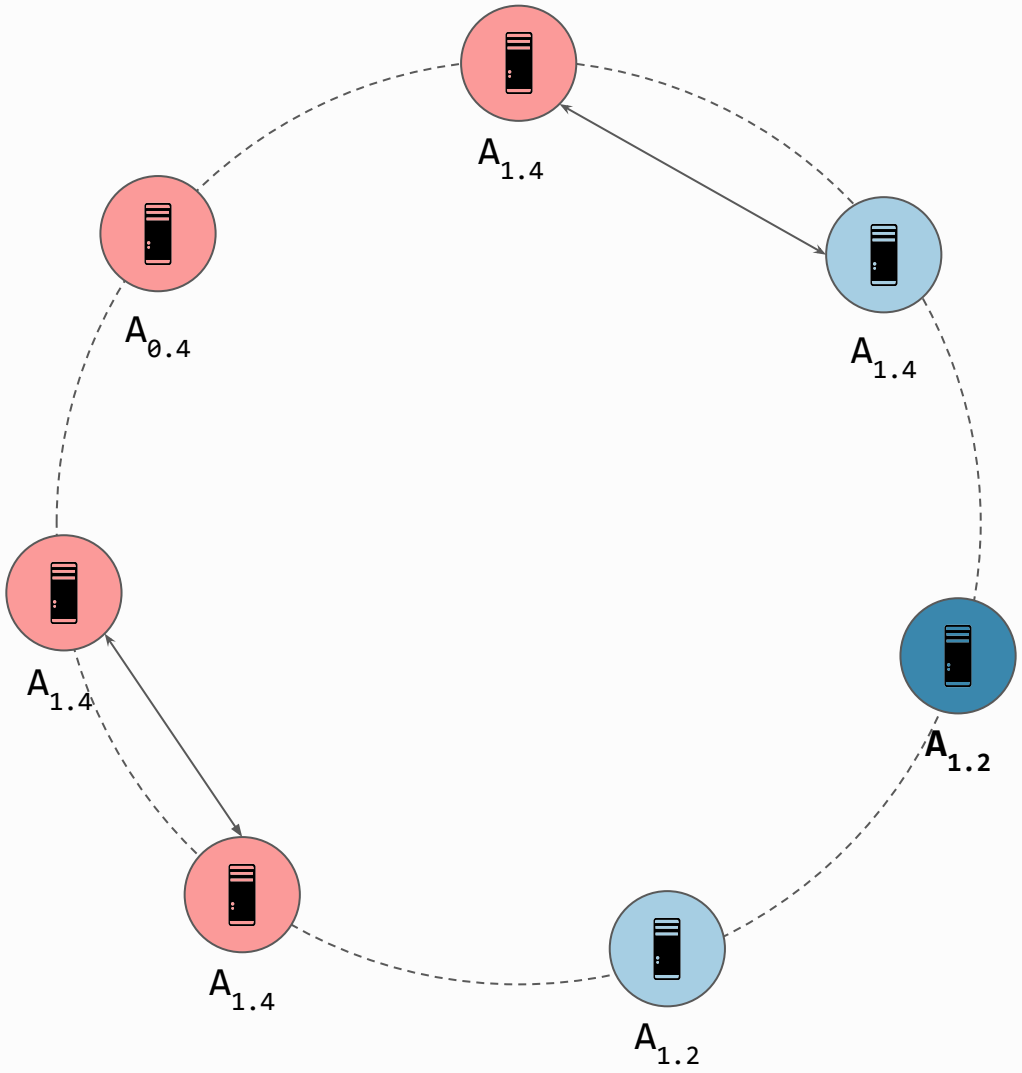
The Forte Number: Replication via Anti-Entropy and Append Entries



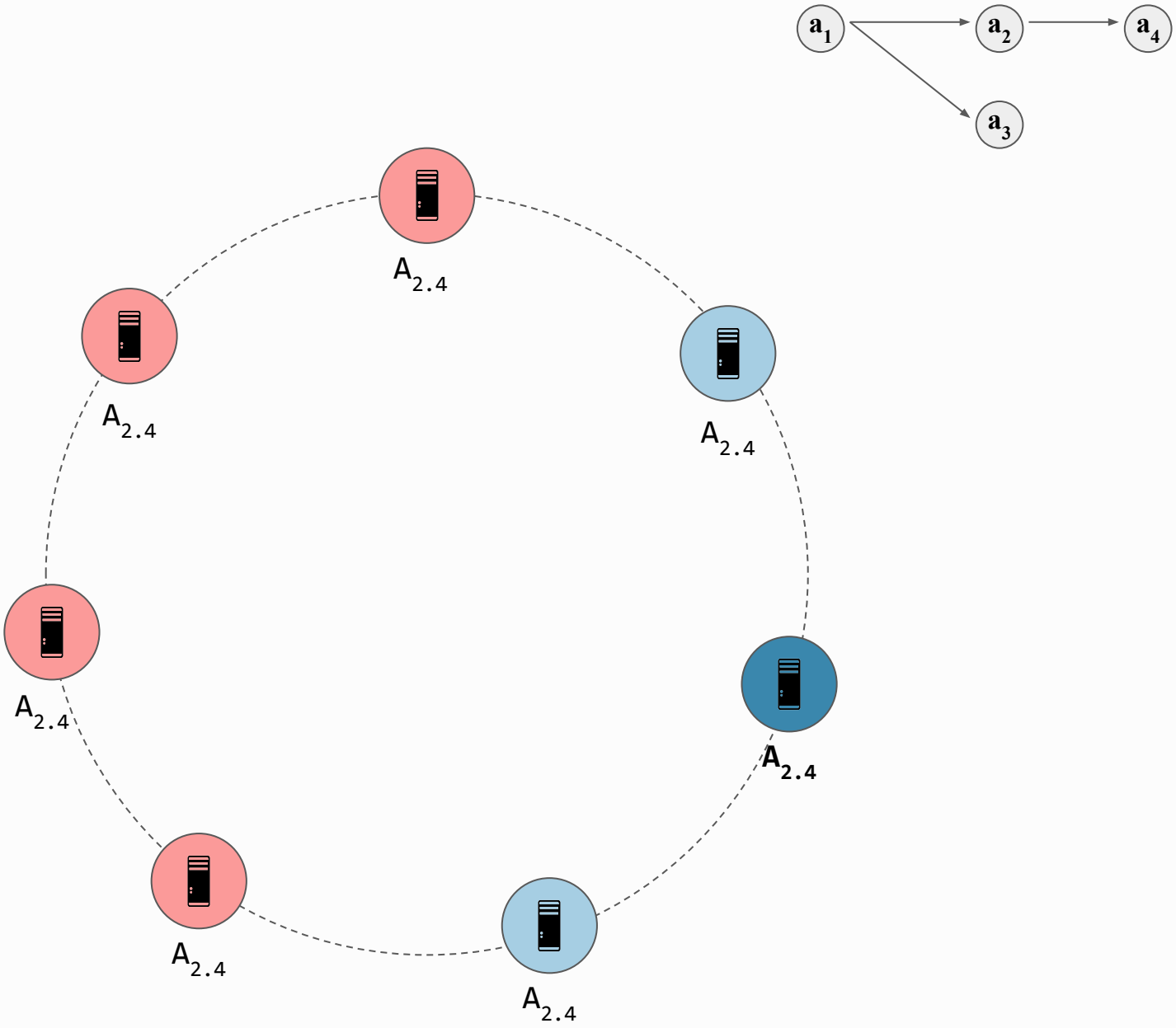


The Forte Number: Overrides the Fork during Anti-Entropy

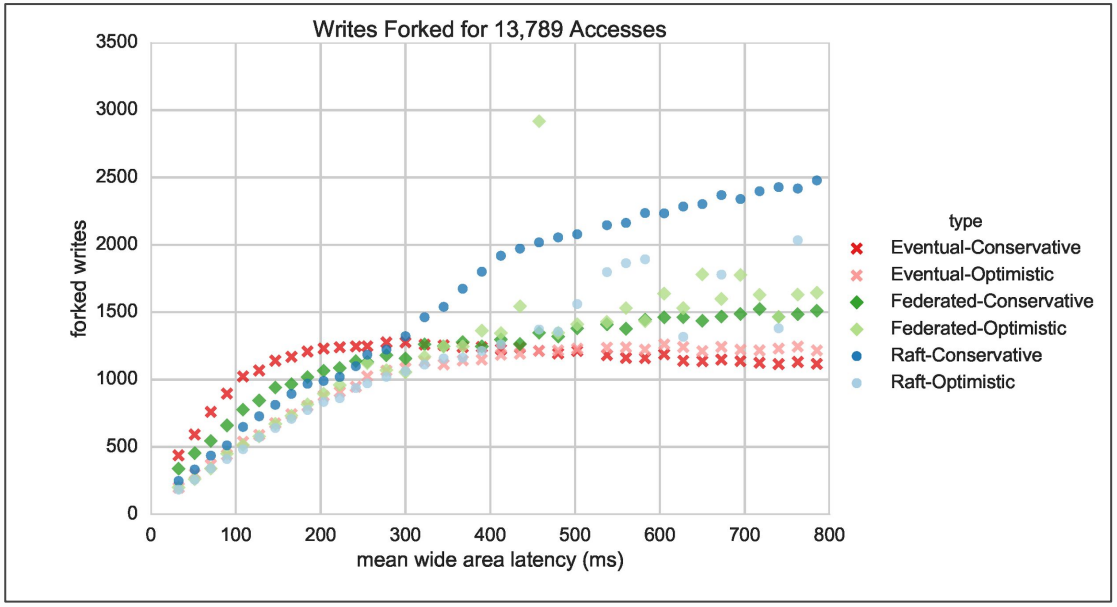


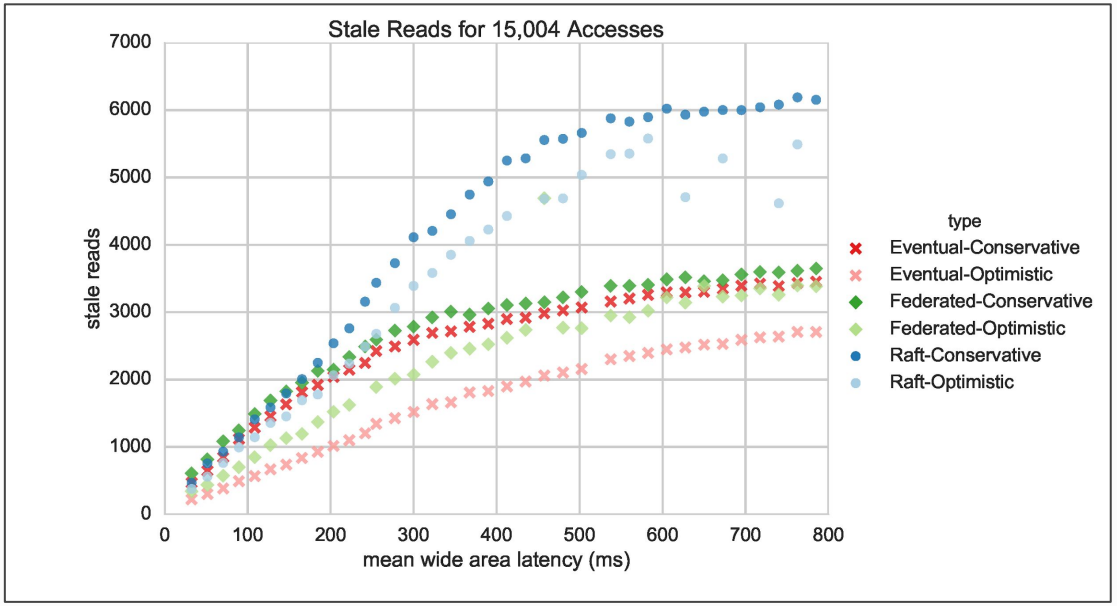


The Forte Number: Descendants Updated with Forte of Ancestor



The Forte Number: Skipping Ahead – Final State





Simulation Results: Increasing Variable Latency

Federated Consistency Overview

Benefits

- Allows replicas to flexibly select consistency level.
- Allows system to flexibly define global consistency.
- Better consistency (less forks and stale reads) than eventual.
- Better availability (lower write latency) than Raft.
- Fault tolerant in the case of outages.

Limitations

- Centralized consensus group does not scale.
- Requirement for centralized quorum in order to increase consistency.
- Better overall, room for improvement.
- Eventual can “go around” the central quorum causing duplicate decisions.

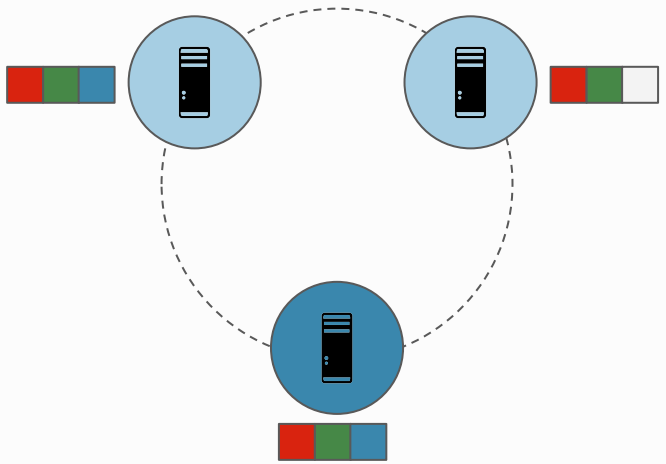
Hierarchical Consensus

Consensus Algorithms

Most Consensus Algorithms are variants of Paxos (Lamport 2005)

- **Multi-Paxos** and **Raft**: dedicated leader (Lamport 2001; Ongaro and Ousterhout 2014)
- **Fast-Paxos**: multiple leaders per round (Lamport 2006)
- **S-Paxos**: distribute leadership (Biely et al. 2012)
- **Flexible Paxos**: multiple quorum intersections (Howard et al. 2016)
- **Egalitarian Paxos**: fast and slow path voting. (Moraru, Andersen, and Kaminsky 2013)

Variants improve performance while maintaining correctness.



`Prepare ⊛ Accept ⊛ Commit`

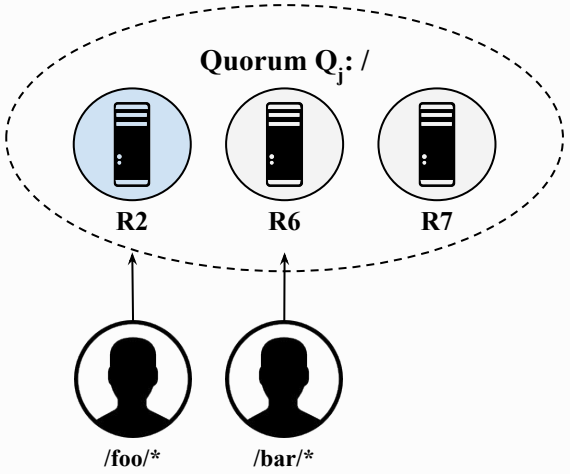
Consensus for File System Sequential Consistency

Consensus can order:

- Lock grants
- Lease grants
- **Accesses**

File systems require specific policies:

- Writes must be forwarded to the leader (sync or async).
- Leader drops writes that are inconsistent (e.g. forked) and client must retry (first writer wins).
- Writes rolled back on leader failure.

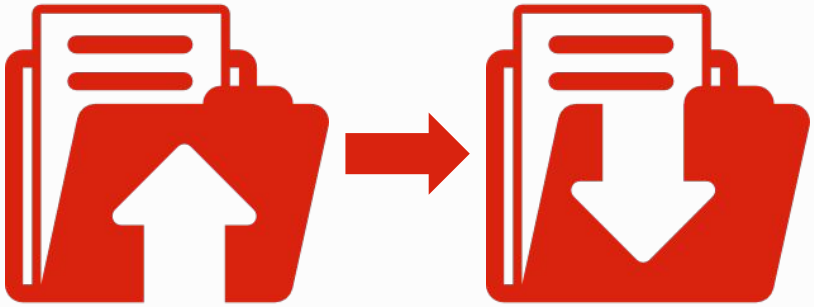


Close to Open Sequential Consistency Policies

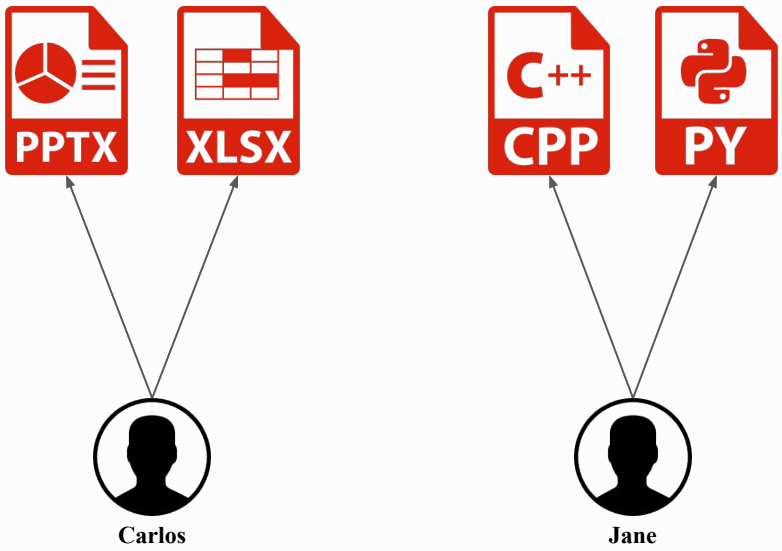
Options for reading a file on open:

- Remote Read: treat as null write and request from leader.
- Read Committed: the last known committed value
- Read Latest: the latest value broadcast though it might not be committed.

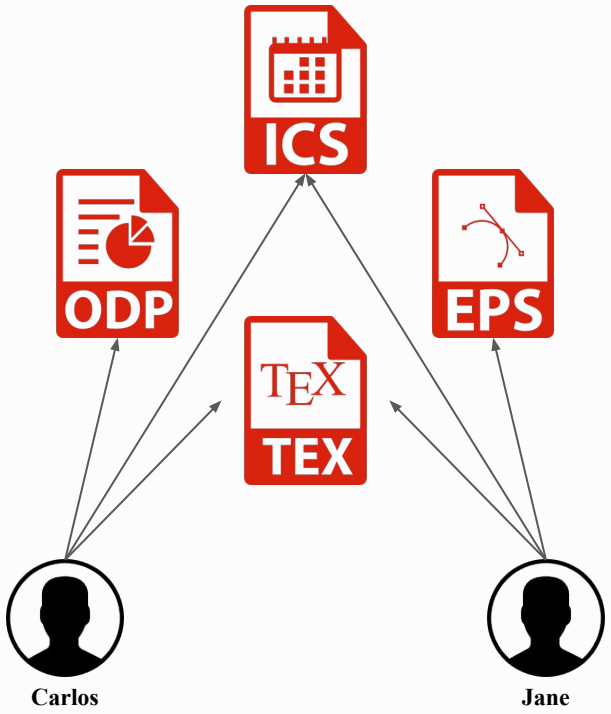
Local writes are always cached and read until a later version (read your writes consistency).



Only write-sync-commit & read-remote guarantee no staleness, but neither policy prevents forks.



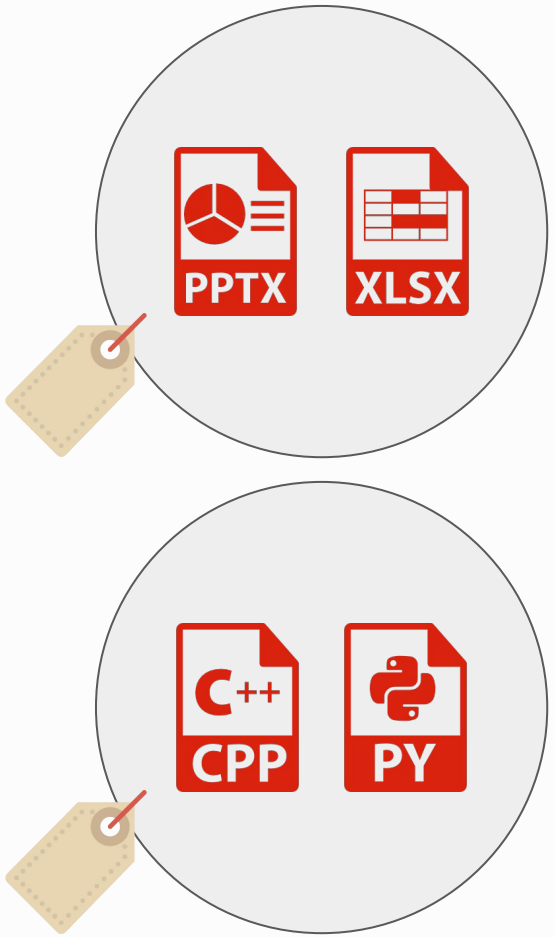
Day One:
Carlos is working on teaching
Jane is coding a project



Day Two:
Combined work on a paper that will be submitted.

Dependent Object Groupings

- **Tag**: a time-annotated subset of the namespace.
- Tags define explicit dependencies between similar objects.
- All accesses within a single tag should be ordered with respect to each other.
- **Tagspace**: set of non-overlapping tags that define the namespace for a given period.

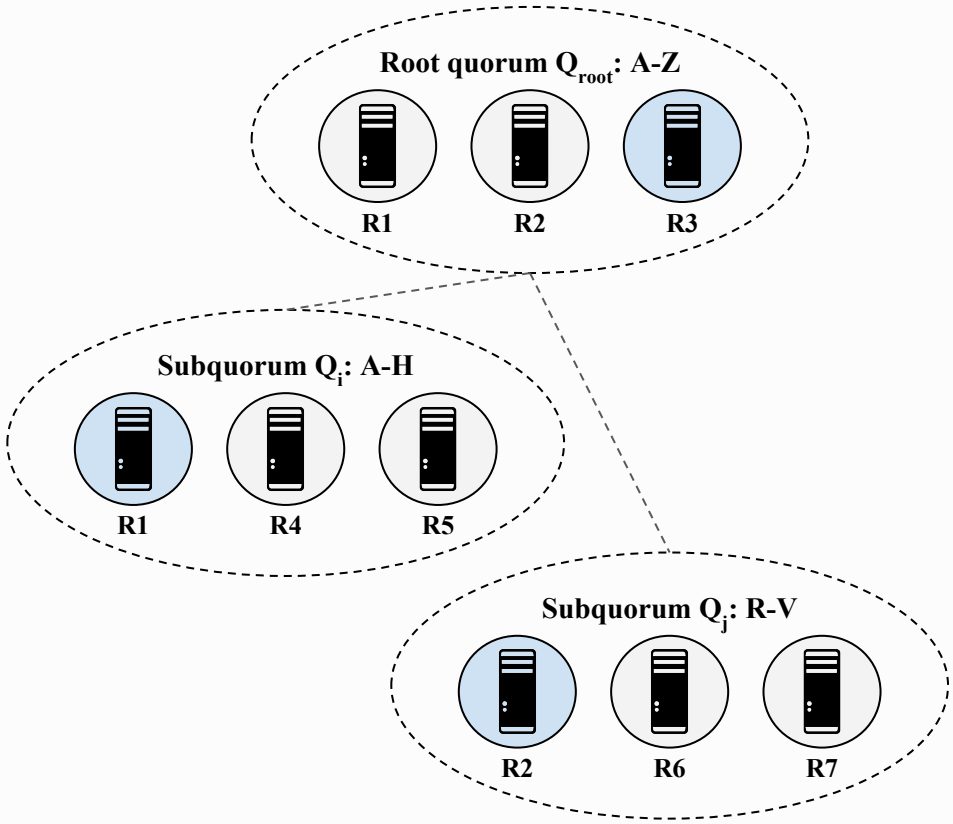


Tag Consensus Operations

The root quorum coordinates and allocates subquorums which each manage accesses to their own tag.

- **Split:** modify the tagspace to create new tags.
- **Join**: merge tags together or back into the global tagspace.

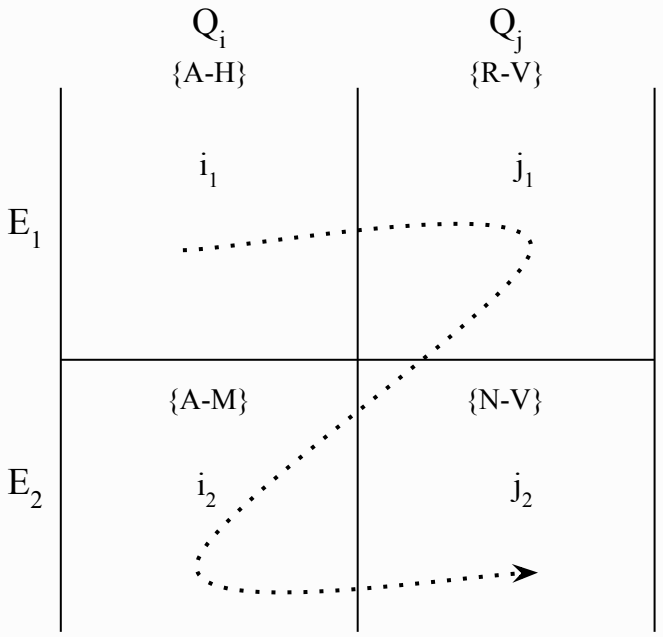
Once a subquorum has been created, accesses can be forwarded to the subquorum leader.



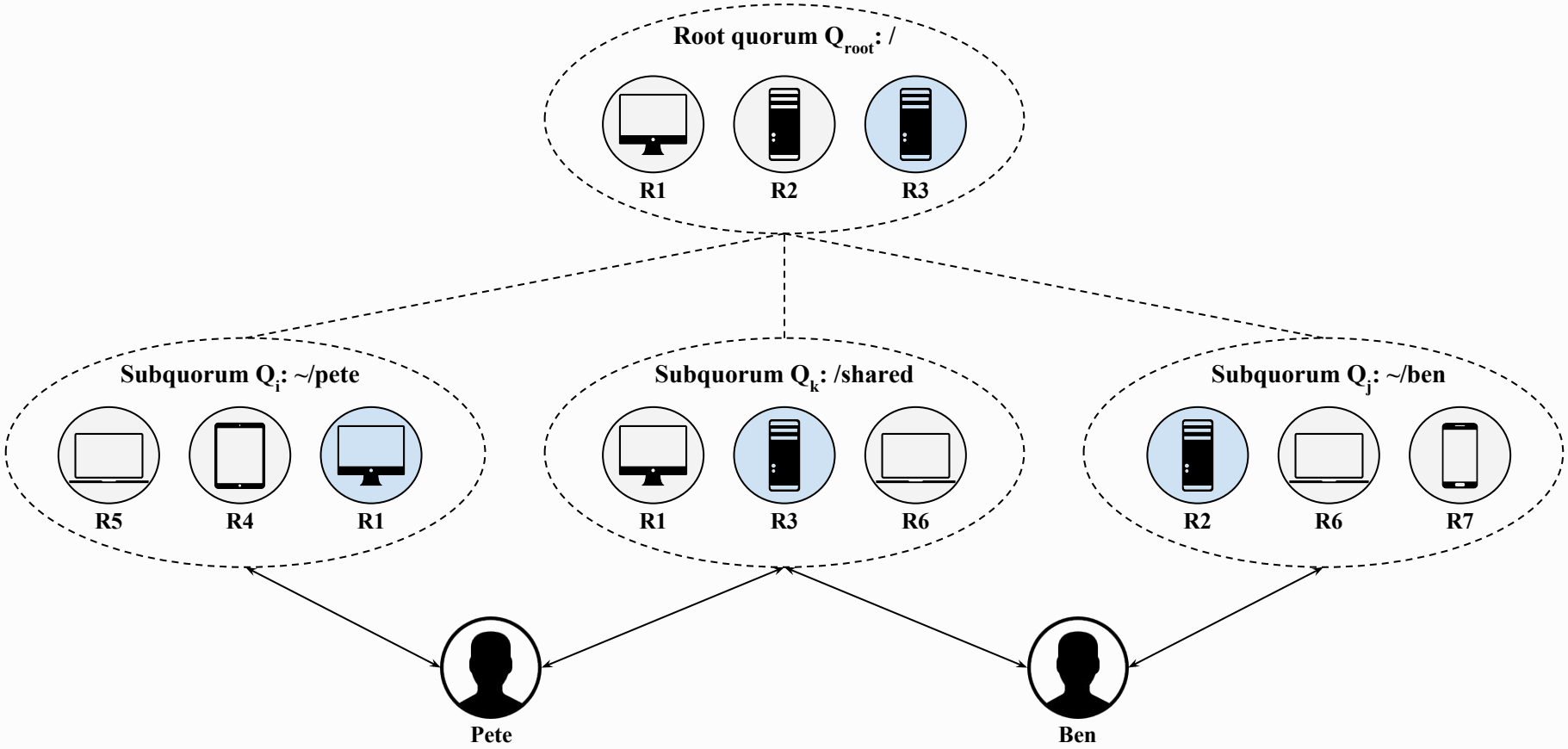
One Subquorum per Tag!

Epochs and Sequential Ordering

- **Epoch**: monotonically increasing counter that uniquely identifies a tag space.
- On split/join consensus decision, the epoch is incremented.
- Any access in epoch $E_i \rightarrow E_{i+1}$
- Alternatively, accesses in E_{i+1} depend on those in E_i
- Accesses across tags are concurrent globally, but ordered locally.



One acceptable ordering of accesses.

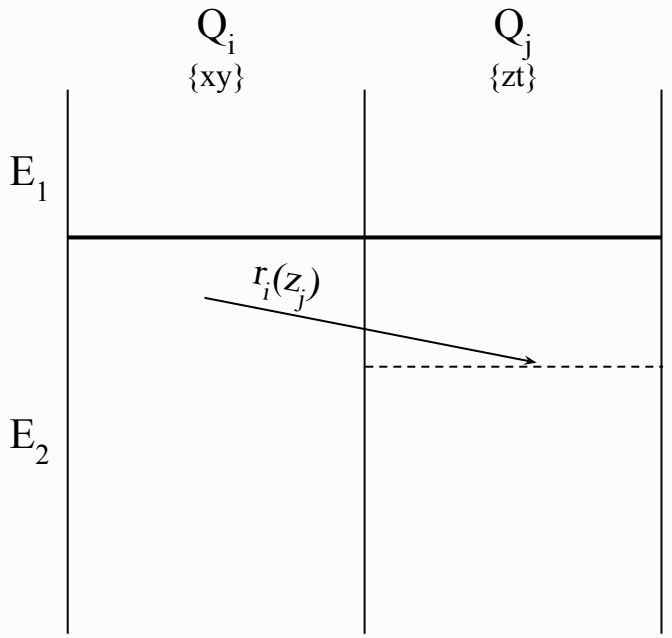


Allocating the Namespace

Remote Accesses and Subepochs

Epoch changes can introduce unnecessary complexity for a small number of dependent accesses.

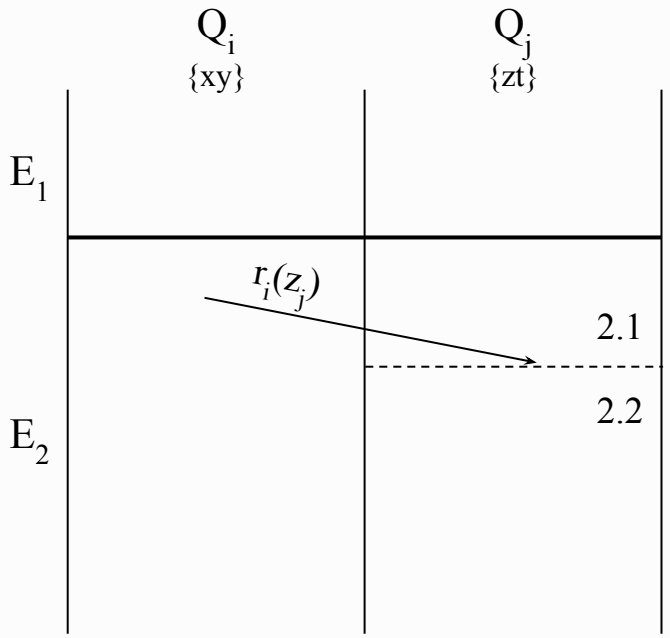
- **Remote accesses** to different tags create inter-epoch dependencies.
- **Subepochs**: delineate the ordering of accesses → remote access and those that happen after.



Remote Accesses and Subepochs

Epoch changes can introduce unnecessary complexity for a small number of dependent accesses.

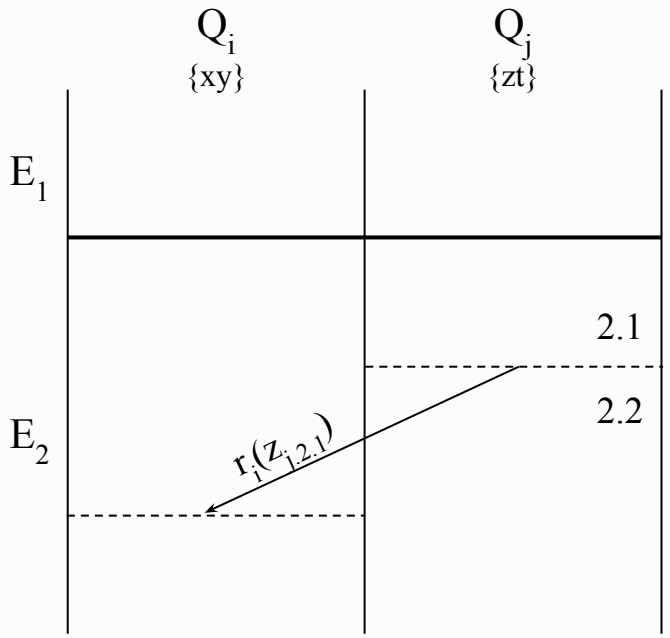
- **Remote accesses** to different tags create inter-epoch dependencies.
- **Subepochs**: delineate the ordering of accesses → remote access and those that happen after.



Remote Accesses and Subepochs

Epoch changes can introduce unnecessary complexity for a small number of dependent accesses.

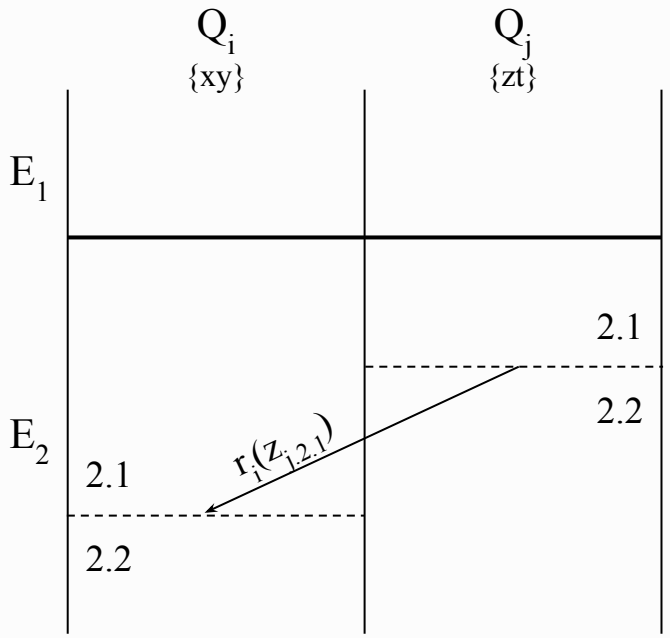
- **Remote accesses** to different tags create inter-epoch dependencies.
- **Subepochs**: delineate the ordering of accesses → remote access and those that happen after.



Remote Accesses and Subepochs

Epoch changes can introduce unnecessary complexity for a small number of dependent accesses.

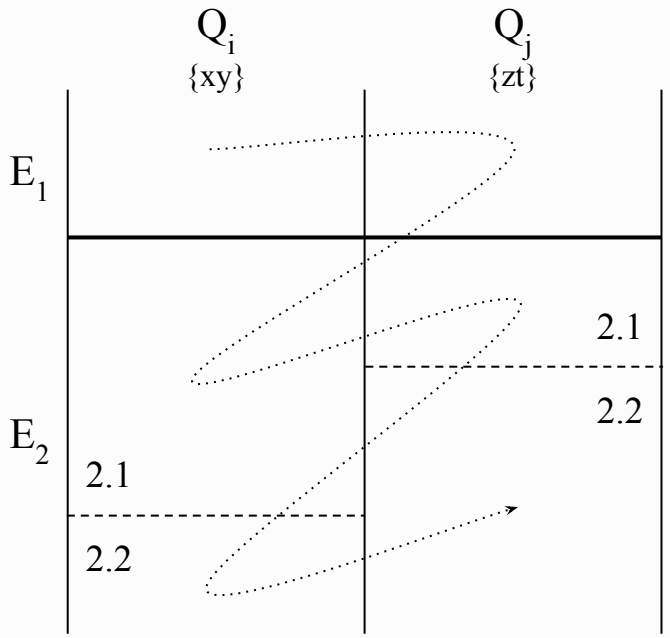
- **Remote accesses** to different tags create inter-epoch dependencies.
- **Subepochs**: delineate the ordering of accesses → remote access and those that happen after.

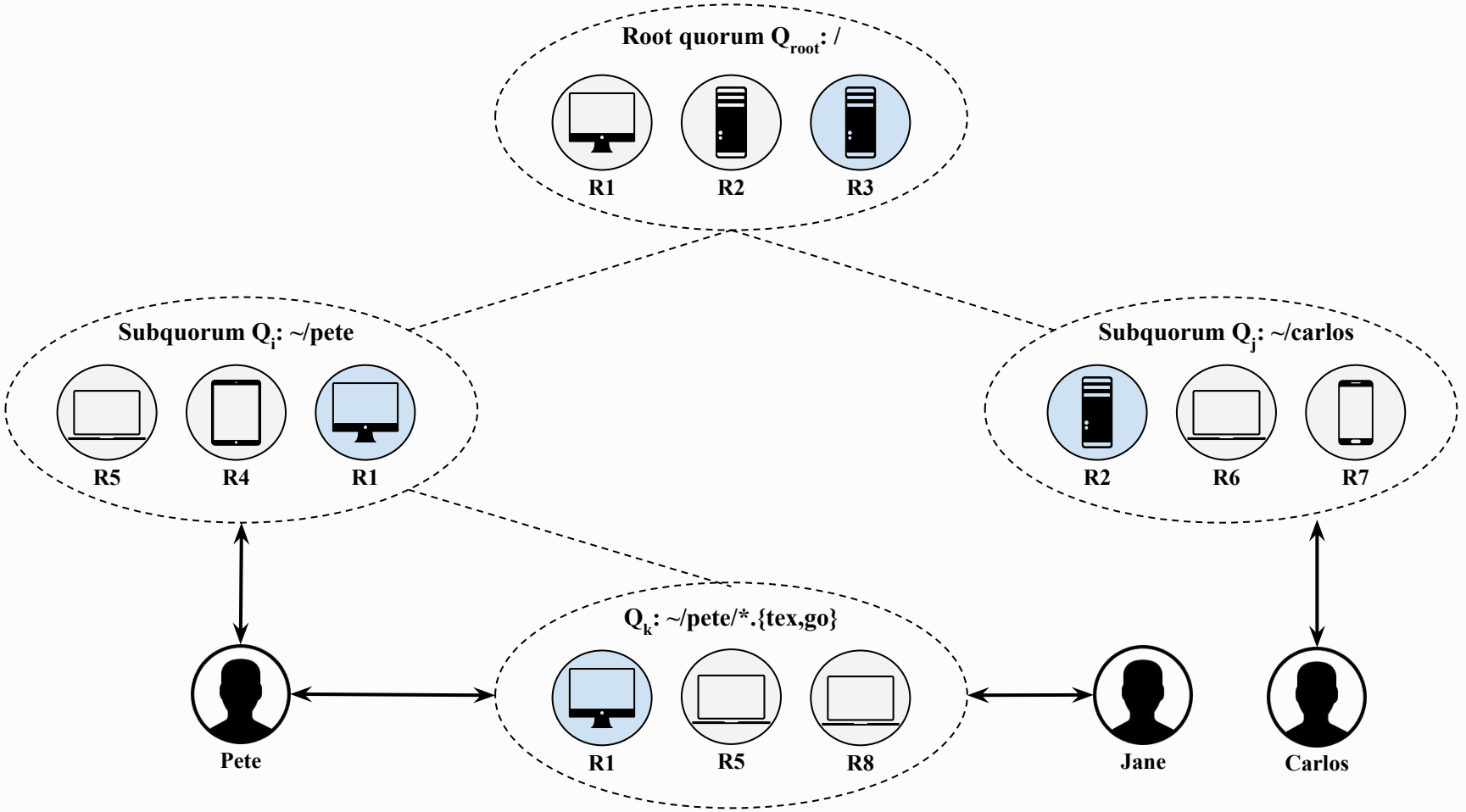


Remote Accesses and Subepochs

Epoch changes can introduce unnecessary complexity for a small number of dependent accesses.

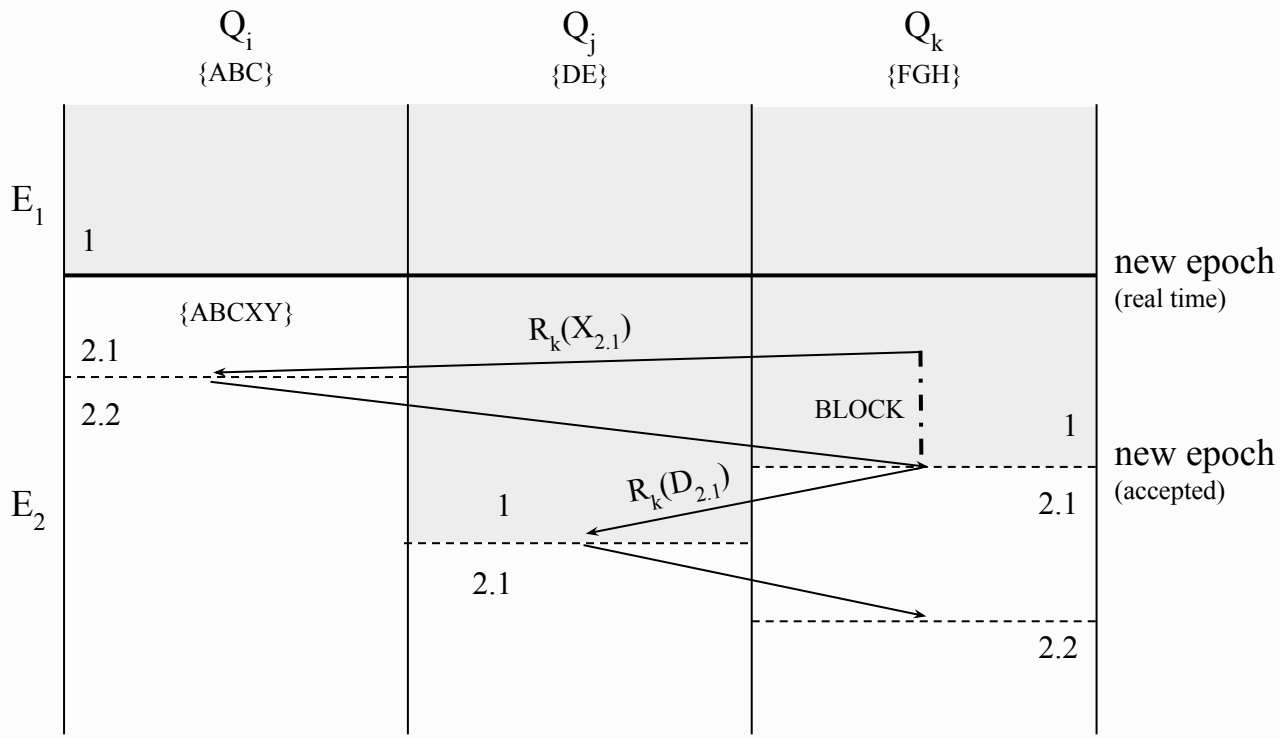
- **Remote accesses** to different tags create inter-epoch dependencies.
- **Subepochs**: delineate the ordering of accesses → remote access and those that happen after.





Subepochs may also lead to deeper hierarchies

Fuzzy Epochs for Fault Tolerance



The epoch changes when affected subquorums confirm. All other subquorums can lag or delay updating their epoch: global coordination is not required.

Hierarchical Consensus Overview

Benefits

- Scales to any arbitrarily sized system.
- Provides sequential ordering of all close-to-open accesses.
- Localizes decision making.
- All consensus decisions only depend on a small fraction of the overall topology.
- Failures do not prevent progress.

Future Work

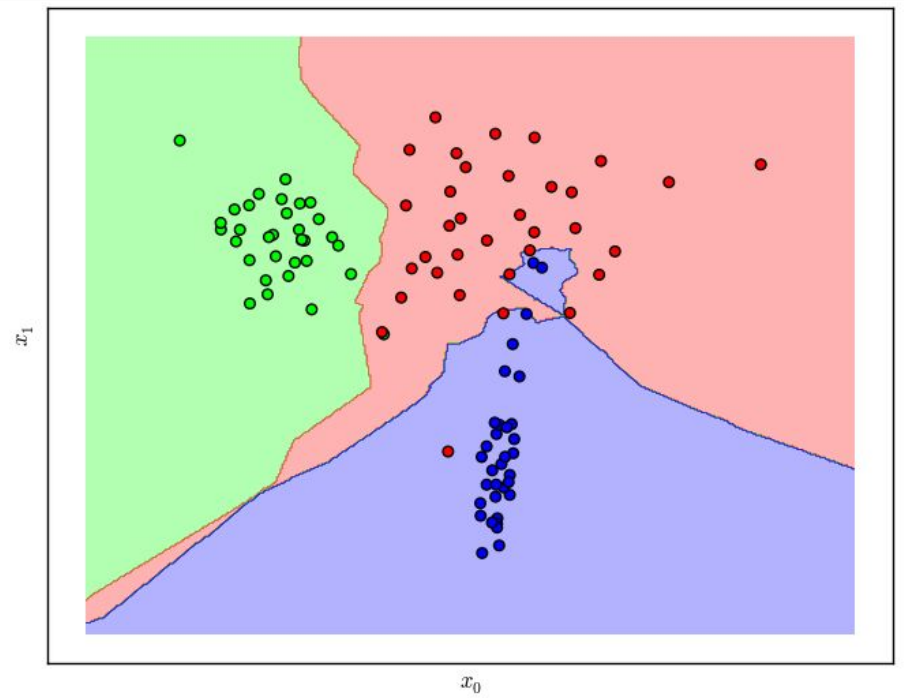
- Mechanism to allocate the tagspace on demand.
- Membership management and quorum formation protocol.
- Proof of correctness
- Study of the benefits of quiescence

Adaptive Consistency

Learning Policies

Per-user trained models of environmental conditions:

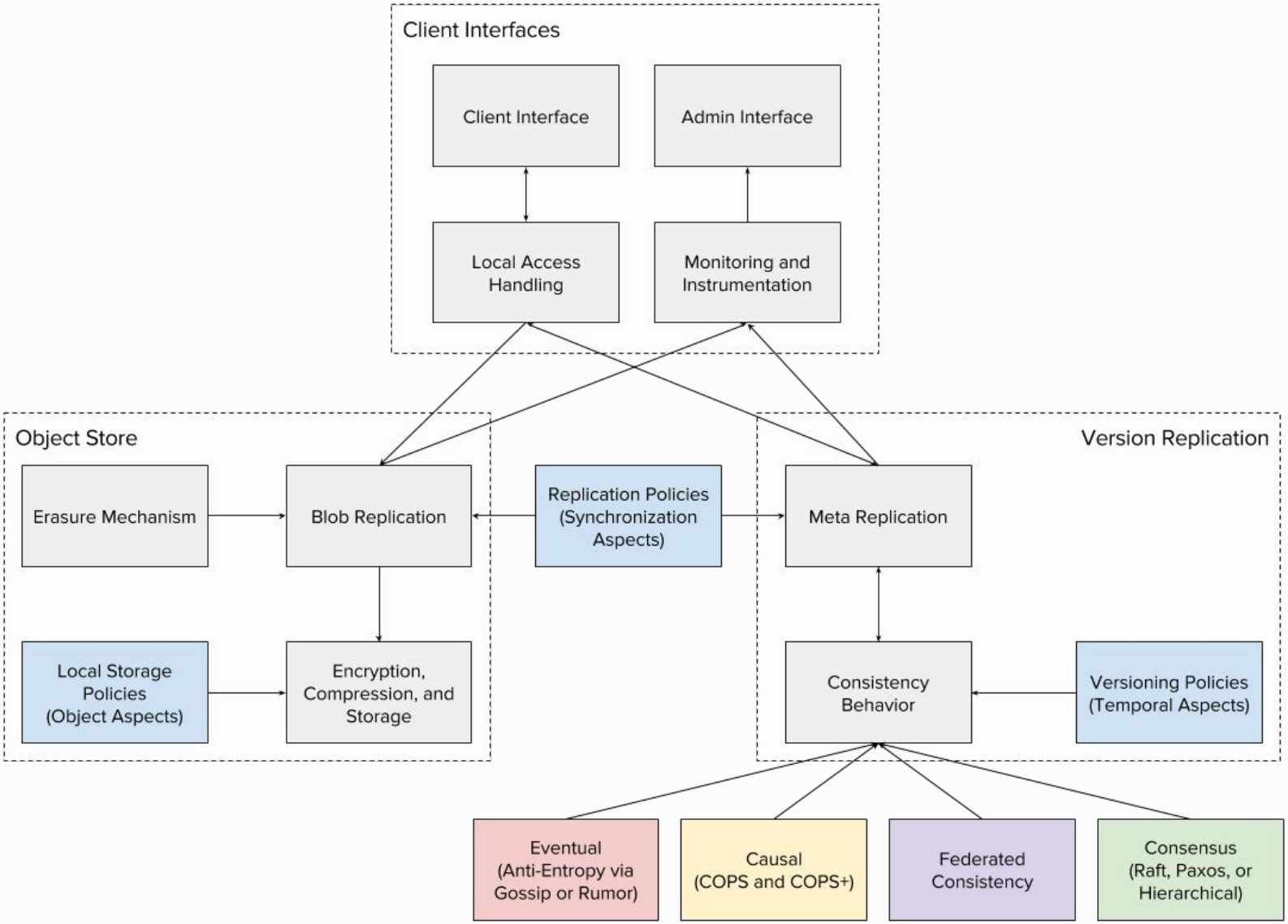
- stable/unstable/moderate classifiers
- File similarity/clustering for object policies
- Anomaly detection algorithms for significant environmental changes
- Regression models to compute T parameter
- Pattern learning to predict when to perform a remote access vs. when to perform an epoch change



Real Time Adaptation

- Online monitoring of network conditions, location, access patterns, communications.
- Bandit algorithms to modify timing parameters, blob replication.
- Optimize selection of neighbors during anti-entropy.

System Model



Component Architecture

File System Consistency: Versions

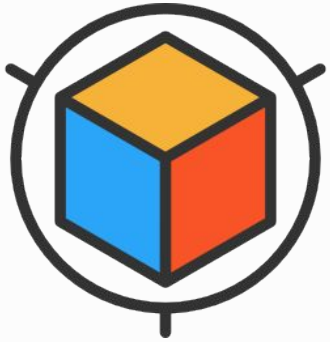
Each object in the file system is represented as a *version* – piece of metadata containing:

- Parent version: the version read when the file was opened.
- Conflict-free version number
- Explicit listing of dependencies
- List of blobs that contain the data of the file.

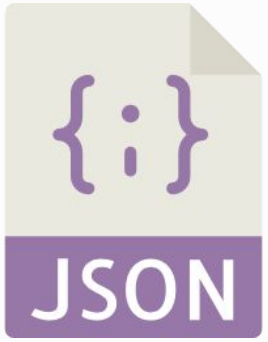
```
{
    parent: (1, 2, 0),
    version: (2, 3, 0),
    dependencies: [
        ...
    ],
    blobs: [
        1124c7b, 921e45a
    ]
}
```


Decoupling Object Store from Version Metadata

- Files chunked into fixed length blobs of opaque data.
- Blobs are immutable and identified by their hash.
- Replicated via anti-entropy.
- Blobs can be requested on demand or clustered for locality and durability.

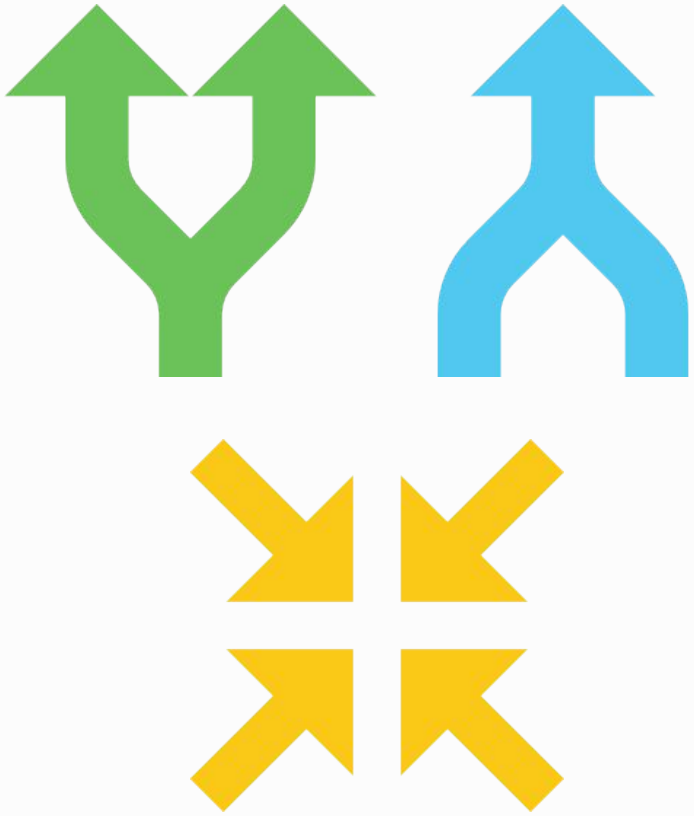


- Versions define the local view of the file system.
- Consistency only depends on the replication of versions.
- Versions are compact leading to smaller messages and better throughput.



Conflict Detection and Resolution

- Conflict detected when two versions have the same parent.
- Chunking of text files allows git-style automatic merges.
- Binary files updated as “all-or-nothing” and are write-once, read-many.
- Goal: bring conflicts to the attention of user ASAP.

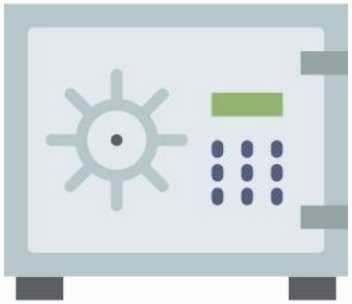


Security in FlowFS

Our system assumes non-byzantine failure, but will implement the following minimum security practices:

- RPC messages signed with expiring tokens (HAWK)
- Transport over SSL/TLS
- Per-User Blob Encryption

We believe security should not be an afterthought in systems.



The combination of federated consistency with hierarchical consensus will provide higher availability and stronger correctness guarantees in user-centric dynamic networks.

Project	Priority	Time Estimate
Simulation of Federated Consistency	△	1 months
Simulation of Hierarchical Consensus	△	2 months
Implementation of the FlowFS	△	3-4 months
Evaluation of FlowFS on real workloads	△	2-3 months
Proof of correctness and consistency	△	1 month
Heuristics to optimize and allocate FlowFS	▽	1 month
Online optimization and consistency adaptation	▽	2-3 months
TOTAL		12-16 months

Paper	Title	Location	RFP	Date
Federated Consistency	IEEE ICDCS 2017	Atlanta, GA	Dec 5, 2016	Jun 5-8 2017
User-Centric Dynamic Clouds	ACM HotStorage 2017	Santa Clara, CA	N/A	Jul 10-11, 2017
Hierarchical Consensus	ACM PODC 2017	Washington DC	Feb 10, 2017	July 2017
Hierarchical FlowFS	ACM SOSP 2017	Shanghai, China	Apr 21, 2017	Oct 29-31, 2017
Responsive Consistency	USENIX NSDI 2018	N/A	N/A	N/A

Timeline

Thank You!

Bibliography

[1] M. Anderson, “Smartphone, computer or tablet? 36% of Americans own all three,” Pew Research Center, 25-Nov-2015. .

[2] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica, “Probabilistically bounded staleness for practical partial quorums,” Proceedings of the VLDB Endowment, vol. 5, no. 8, pp. 776–787, 2012.

[3] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica, “Quantifying eventual consistency with PBS,” The VLDB Journal, vol. 23, no. 2, pp. 279–302, 2014.

[4] D. Bermbach and J. Kuhlenkamp, “Consistency in distributed storage systems,” in Networked Systems, Springer, 2013, pp. 175–189.

[5] D. Bermbach and S. Tai, “Eventual consistency: How soon is eventual? An evaluation of Amazon S3’s consistency behavior,” in Proceedings of the 6th Workshop on Middleware for Service Oriented Computing, 2011, p. 1.

Bibliography

[6] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, “DepSky: dependable and secure storage in a cloud-of-clouds,” ACM Transactions on Storage (TOS), vol. 9, no. 4, p. 12, 2013.

[7] M. Biely, Z. Milosevic, N. Santos, and A. Schiper, “S-paxos: Offloading the leader for high throughput state machine replication,” in Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on, 2012, pp. 111–120.

[8] G. DeCandia et al., “Dynamo: amazon’s highly available key-value store,” in ACM SIGOPS Operating Systems Review, 2007, vol. 41, pp. 205–220.

[9] A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten, “SPORC: Group Collaboration using Untrusted Cloud Resources.,” in OSDI, 2010, vol. 10, pp. 337–350.

[10] M. P. Herlihy and J. M. Wing, “Linearizability: A correctness condition for concurrent objects,” ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 12, no. 3, pp. 463–492, 1990.

Bibliography

[11] H. Howard, D. Malkhi, and A. Spiegelman, “Flexible Paxos: Quorum intersection revisited,” ArXiv e-prints, Aug. 2016.

[12] J. H. Howard et al., “Scale and performance in a distributed file system,” ACM Transactions on Computer Systems (TOCS), vol. 6, no. 1, pp. 51–81, 1988.

[13] ICT Data and Statistics Division, “ICT Facts and Figures 2016,” International Telecommunication Union, Geneva, Switzerland, Jun. 2016.

[14] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann, “Consistency rationing in the cloud: pay only when it matters,” Proceedings of the VLDB Endowment, vol. 2, no. 1, pp. 253–264, 2009.

[15] T. Kraska, G. Pang, M. J. Franklin, S. Madden, and A. Fekete, “MDCC: Multi-data center consistency,” in Proceedings of the 8th ACM European Conference on Computer Systems, 2013, pp. 113–126.

Bibliography

[16] A. Lakshman and P. Malik, “Cassandra: a decentralized structured storage system,” ACM SIGOPS Operating Systems Review, vol. 44, no. 2, pp. 35–40, 2010.

[17] L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” Communications of the ACM, vol. 21, no. 7, pp. 558–565, 1978.

[18] L. Lamport, “Paxos made simple,” ACM Sigact News, vol. 32, no. 4, pp. 18–25, 2001.

[19] L. Lamport, “Generalized consensus and Paxos,” Technical Report MSR-TR-2005-33, Microsoft Research, 2005.

[20] L. Lamport, “Fast paxos,” Distributed Computing, vol. 19, no. 2, pp. 79–103, 2006.

[21] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues, “Making geo-replicated systems fast as possible, consistent when necessary,” in Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12), 2012, pp. 265–278.

Bibliography

[22] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, “Don’t settle for eventual: scalable causal consistency for wide-area storage with COPS,” in Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, 2011, pp. 401–416.

[23] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, “Internet of things: Vision, applications and research challenges,” Ad Hoc Networks, vol. 10, no. 7, pp. 1497–1516, 2012.

[24] I. Moraru, D. G. Andersen, and M. Kaminsky, “There is more consensus in egalitarian parliaments,” in Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, 2013, pp. 358–372.

[25] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in 2014 USENIX Annual Technical Conference (USENIX ATC 14), 2014, pp. 305–319.

[26] D. B. Terry, A. J. Demers, K. Petersen, M. J. Spreitzer, M. M. Theimer, and B. B. Welch, “Session guarantees for weakly consistent replicated data,” in Parallel and Distributed Information Systems, 1994., Proceedings of the Third International Conference on, 1994, pp. 140–149.

Bibliography

[27] W. Vogels, “Eventually consistent,” Communications of the ACM, vol. 52, no. 1, pp. 40–44, 2009.

[28] X Company, “Project Loon,” 2016. [Online]. Available: https://x.company/loon/. [Accessed: 29-Nov-2016].

[29] H. Yu and A. Vahdat, “Design and evaluation of a conit-based continuous consistency model for replicated services,” ACM Transactions on Computer Systems (TOCS), vol. 20, no. 3, pp. 239–282, 2002.

[30] Y. Zhang, C. Dragga, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, “*-Box: towards reliability and consistency in dropbox-like file synchronization services,” in Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems, 2013.