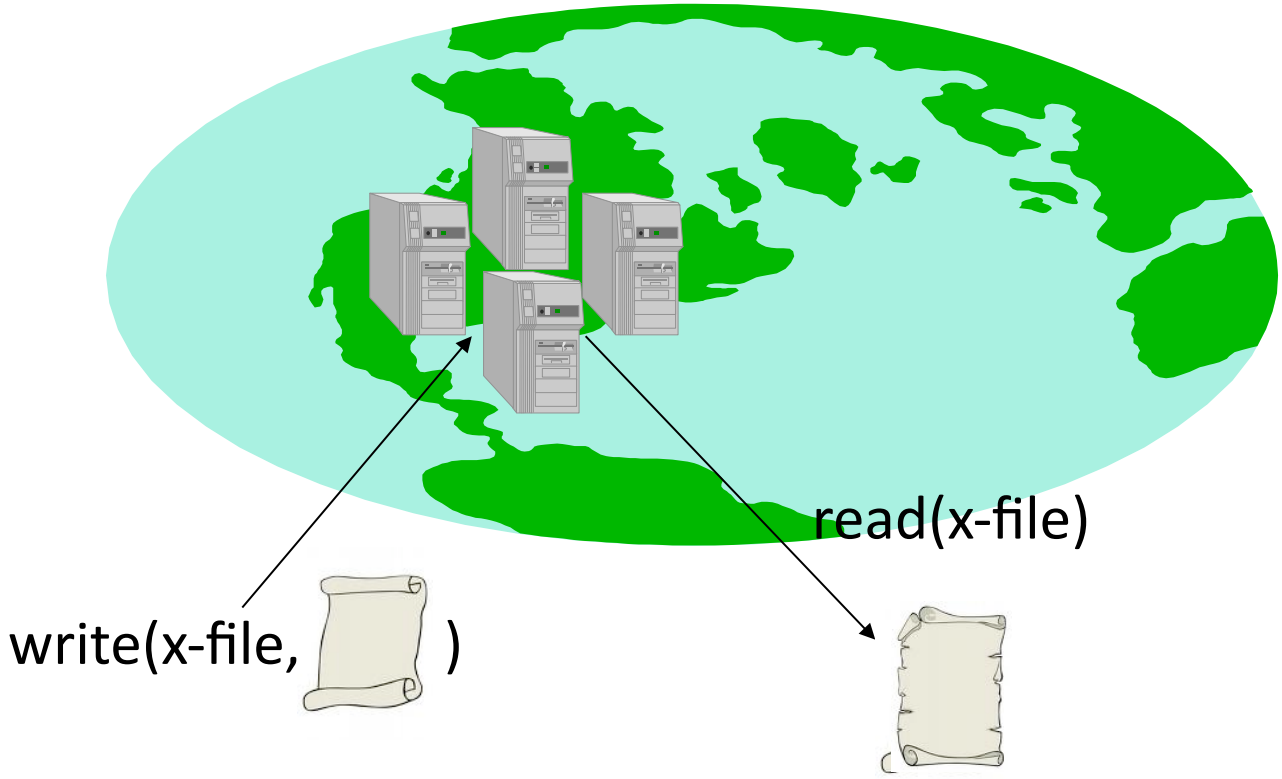
Tiered Fault Tolerance for Long-Term Integrity

Byung-Gon Chun (Intel Research Berkeley)

Joint work with
Petros Maniatis (Intel Research Berkeley),
Scott Shenker (UC Berkeley, ICSI),
and John Kubiatowicz (UC Berkeley)

Long-term applications

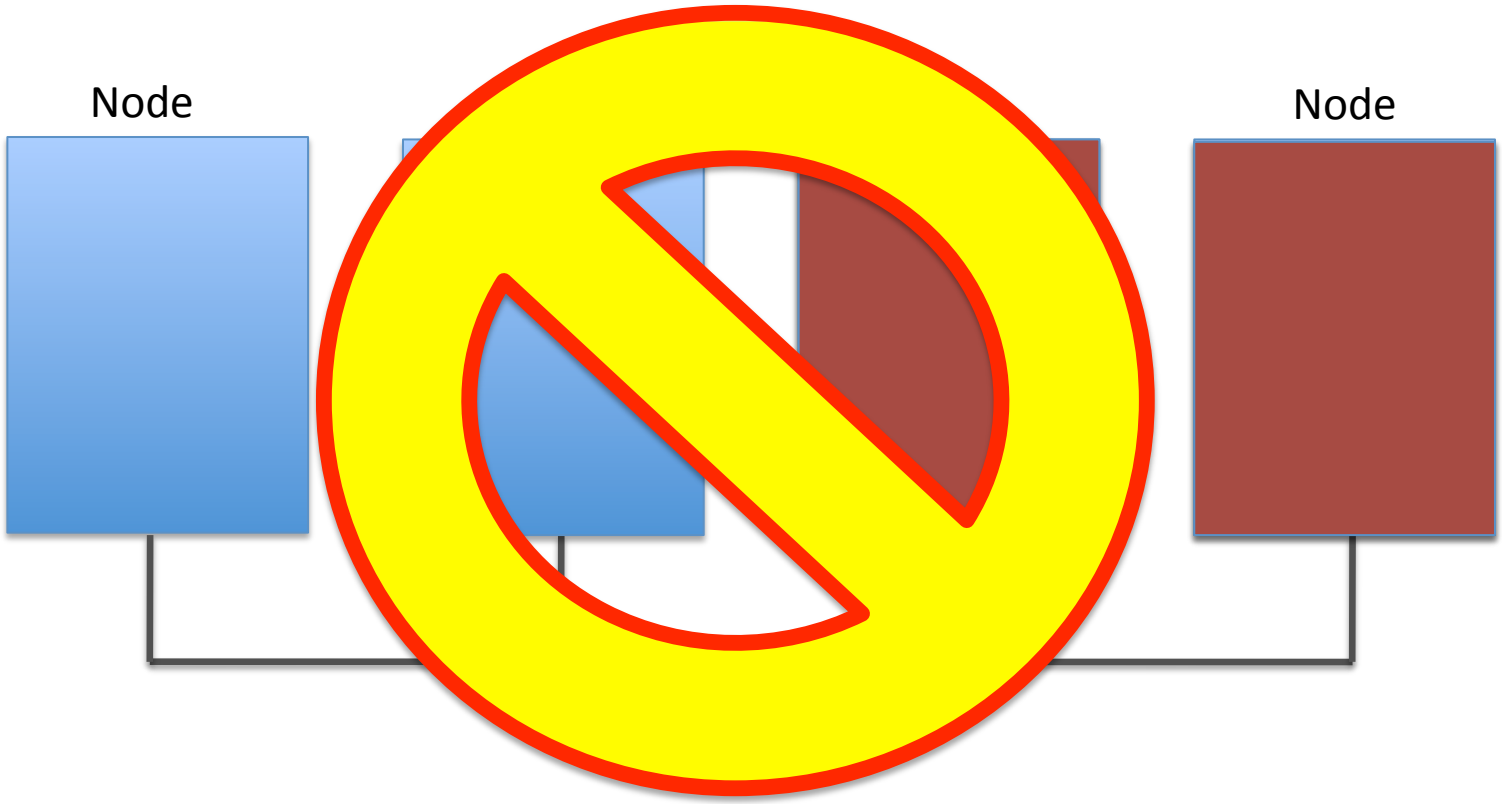


Are current fault-tolerant replicated service designs suitable for long-term applications?

Near-term solutions do not fit

- BFT replicated systems: correct if the number of faulty replicas is always less than some fixed threshold (1/3 of the replicas)

Near-term solutions do not fit



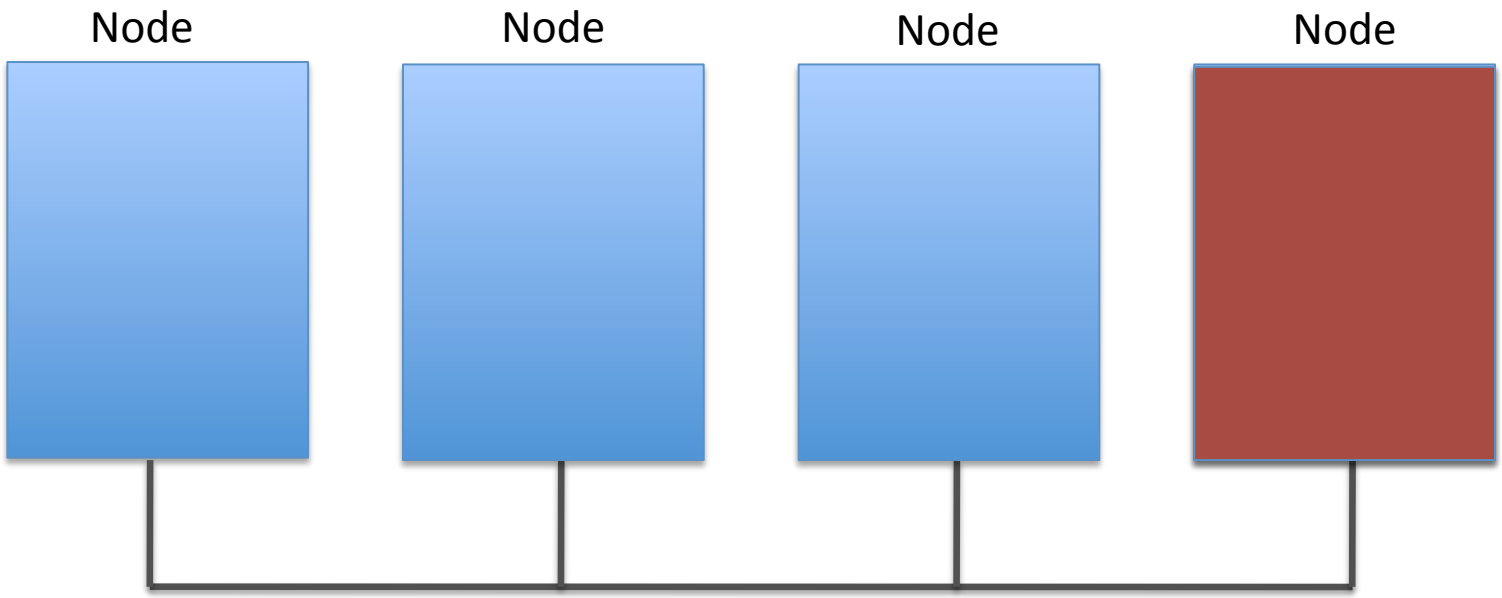
A new approach to designing long-term applications

- A reliability of a system's components over long spans of time can vary dramatically
- Consider this differentiation for long-term applications
  => Tiered fault-tolerant system framework
- Apply the framework to construct *Bonafide*, a long-term key-value store

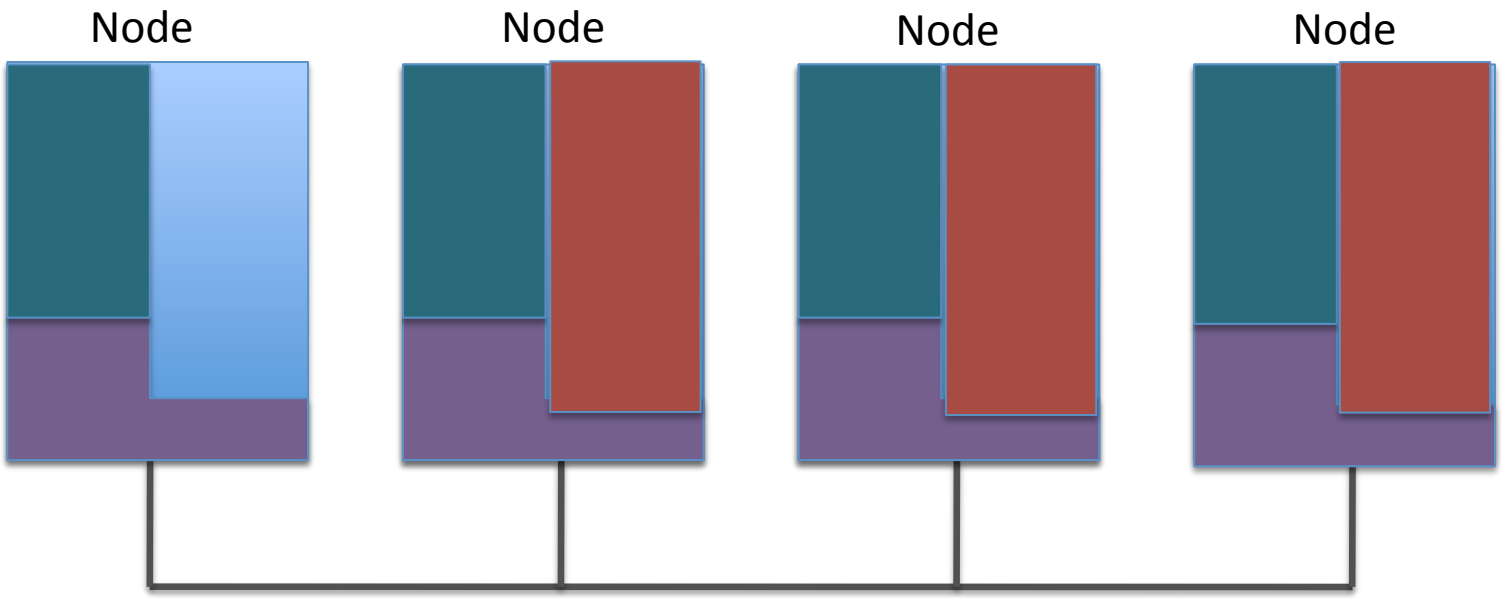
Roadmap

- Tiered fault tolerance framework
- Bonafide: a long-term key-value store
  - Tiers: Trusted, Semi-trusted, Untrusted
- Evaluation

Monolithic fault-tolerant system model



Tiered fault-tolerant system model



Sources of differentiation

- Different assurance practices
  - Formally verified components vs. type-unsafe software
- Care in the deployment of a system
  - Tight physical access controls, responsive system administration vs. unreliable organization
- Rolling procurement of hardware and software
  - A trusted logical component vs. a less trusted component
- Limited exposure
  - Mostly offline vs. online

Reallocation of dependability budget

- Use differentiation to refactor systems into multiple components in different fault tiers
- Different operational practices for each component class

Roadmap

- Tiered fault tolerance framework
- Bonafide: a long-term key-value store
  - Tiers: Trusted, Semi-trusted, Untrusted
- Evaluation

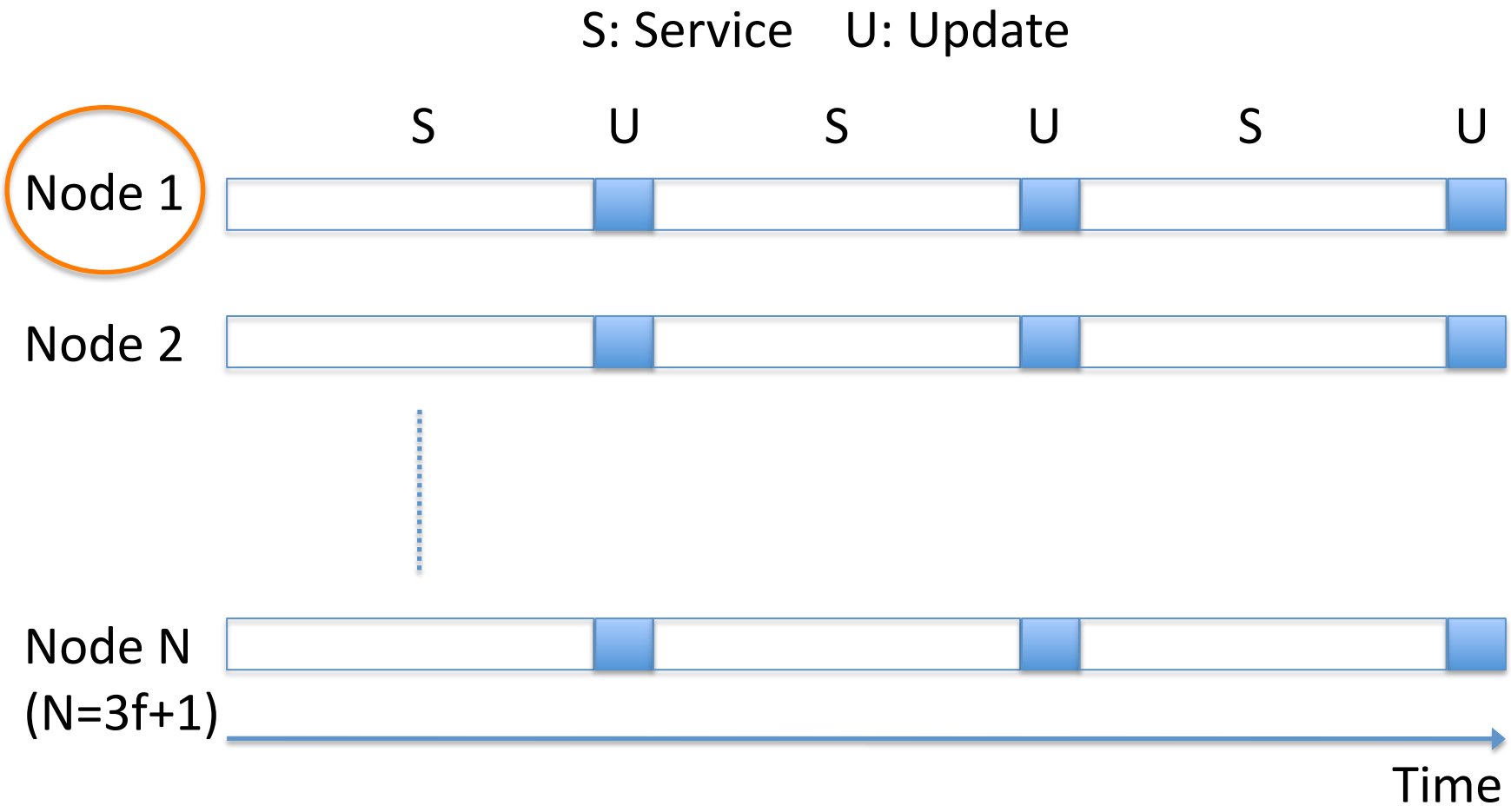
Bonafide

- A key-value store designed to provide long-term integrity using the tiered fault framework
  - Non-self-certifying data
  - A naming service for self-certifying archival storage
- Simple interface:
  - Add(*key, value*)
  - Get(*key*) -> *value*

Design Rationale

- Refactor the fuctionality of the service into
  - A more reliable fault tier for state changes
  - A less reliable fault tier for read-only state queries
- Isolation between these two tiers
  - Trusted component for protecting state during execution of the unreliable tier
  - Use an algorithm to protect large service state with the component
- Mask faults of the component in the more reliable tier
  - Use a BFT replicated state machine
  - Mostly offline, execute in a synchronized fashion

Operation of Bonafide



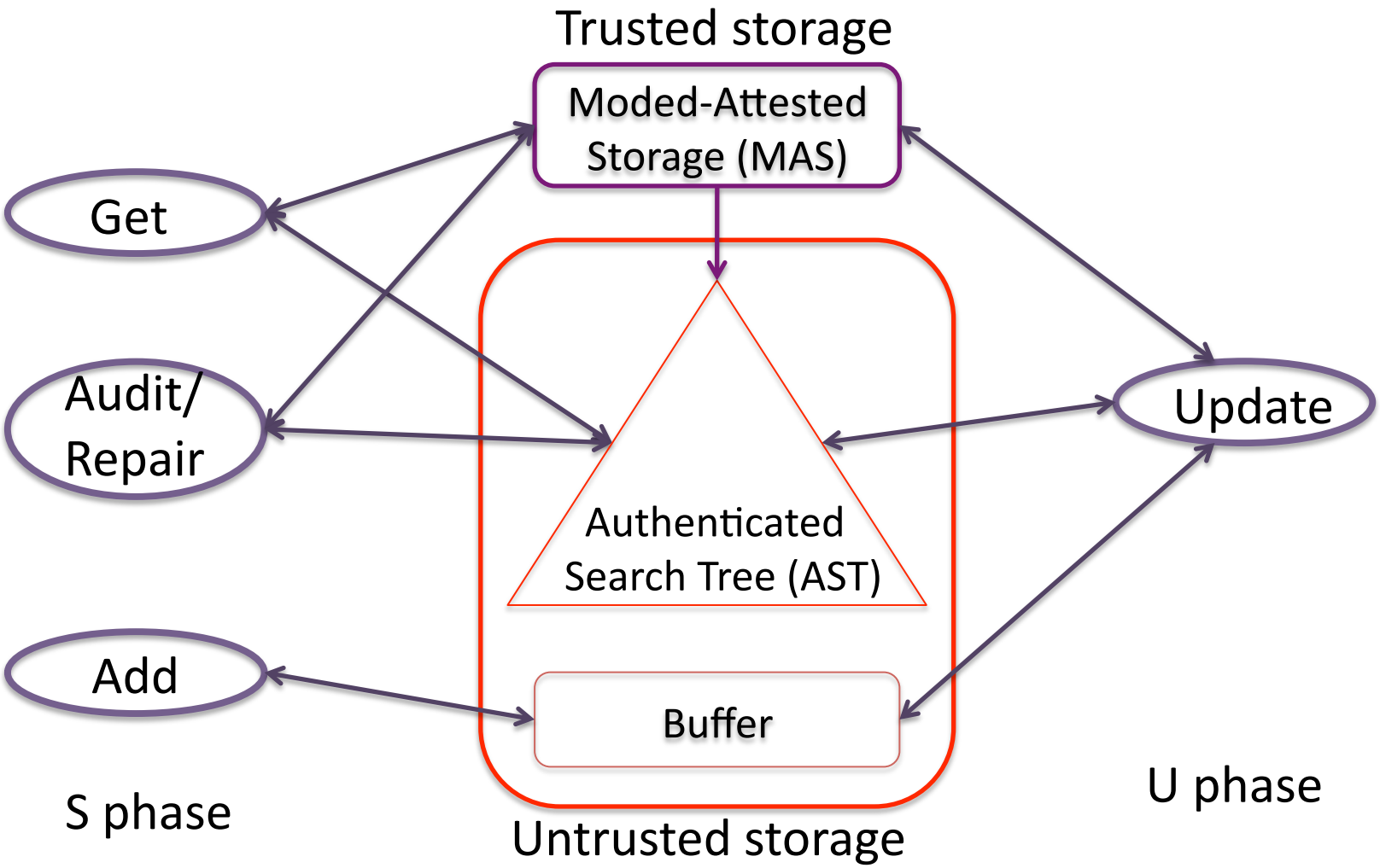
Components in Bonafide and their associated fault tiers

Fault bound	Component	When	How used
0	Watchdog	Periodic	Invoked
	MAS (Moded	S phase	Read
	Attested Storage)	U phase	Written/Read
1/3 Byzantine	Update	U phase	Replicate store Serve ADDs
Unbounded	Service	S phase	Serve GETs Buffer ADDs Audit/Repair

Guarantees

- Guarantees **integrity** of returned data under our tiered fault assumption
- Ensures **liveness** of S phases with fewer than 2/3 faulty replicas during S phases
- Ensures **durability** if the system creates copies of data faster than they are lost

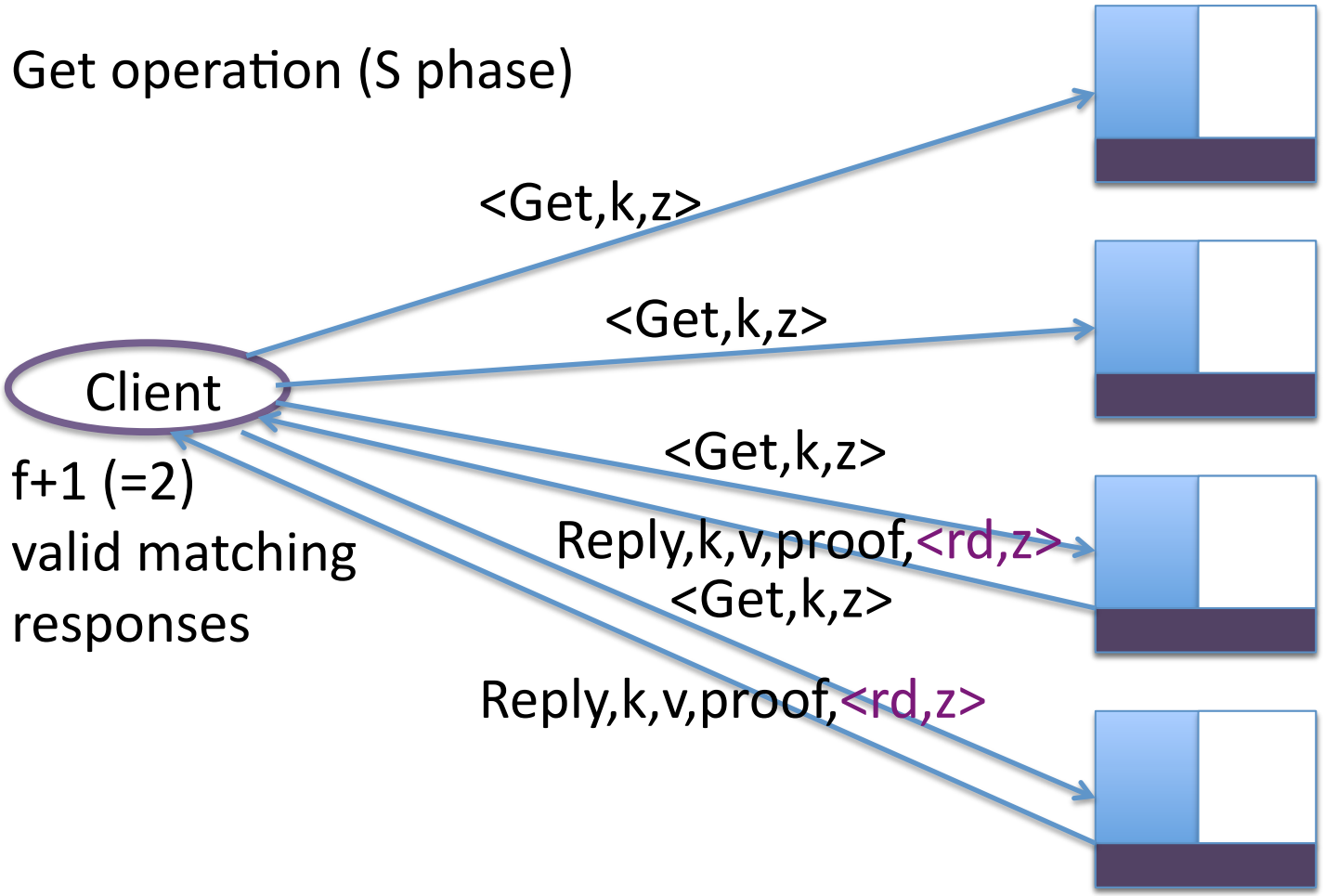
Bonafide replica state and process



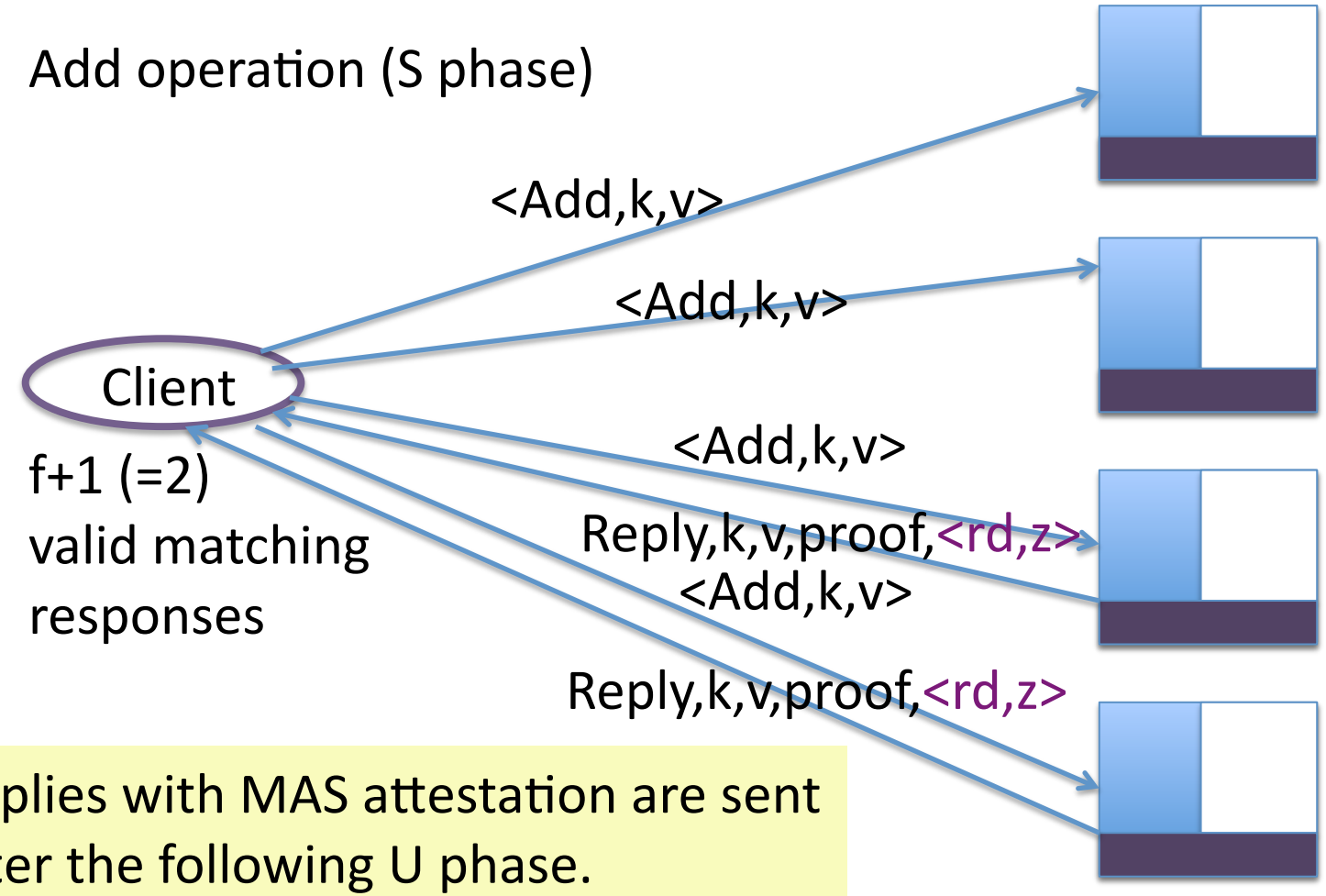
Top tier: trusted

- Cryptography and trusted hardware
- Watchdog: time source, periodic reboot, sets a mode bit of MAS
- MAS: a mode bit, a set of storage slots, signing key
  - Store(*q, v*): store value *v* at slot *q* only in U phases
  - Lookup(*q, z*) -> value *v* of slot *q* and fresh attestation (nonce *z*)

Bottom tier: get

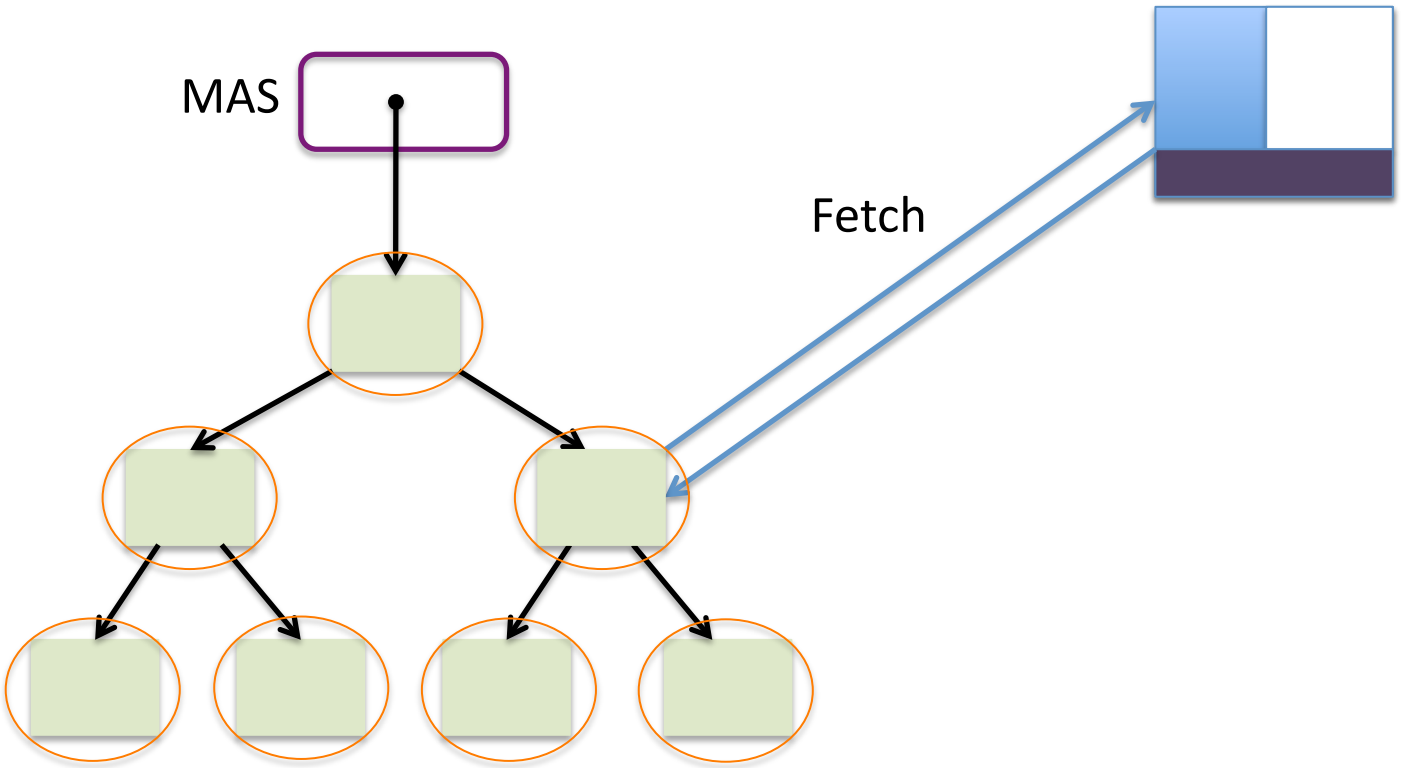


Bottom tier: add

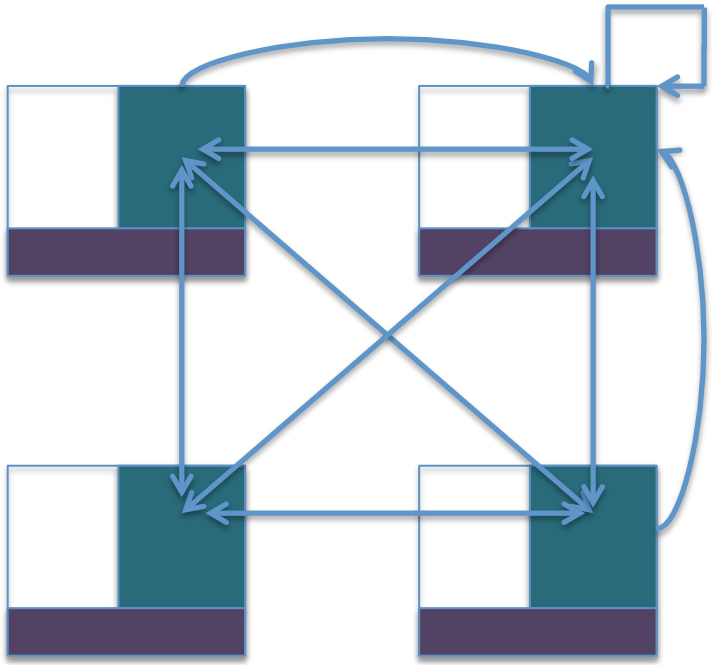


Replies with MAS attestation are sent after the following U phase.

Bottom tier: audit and repair



Middle tier: update process



Reboot

2f + 1 (=3)
PBFT agreements

AST update/
Checkpoint

Time

Evaluating the performance of Bonafide implementation

- A prototype built with sfslite, PBFT, Berkeley DB libraries
  - Server Add/Get, Audit/Repair, Update processes
  - Client proxy process
- Experiment setup
  - Four replica nodes (outdated P4 PCs) running Fedora in a LAN
  - 1 million key-value pairs initially populated
  - Add/Get time, Audit/repair time, U phase duration

Performance evaluation

Get/Add time

Operation	Time (ms) Mean (std)
Get	3.1 (0.24)
Add	1.0 (0.21)

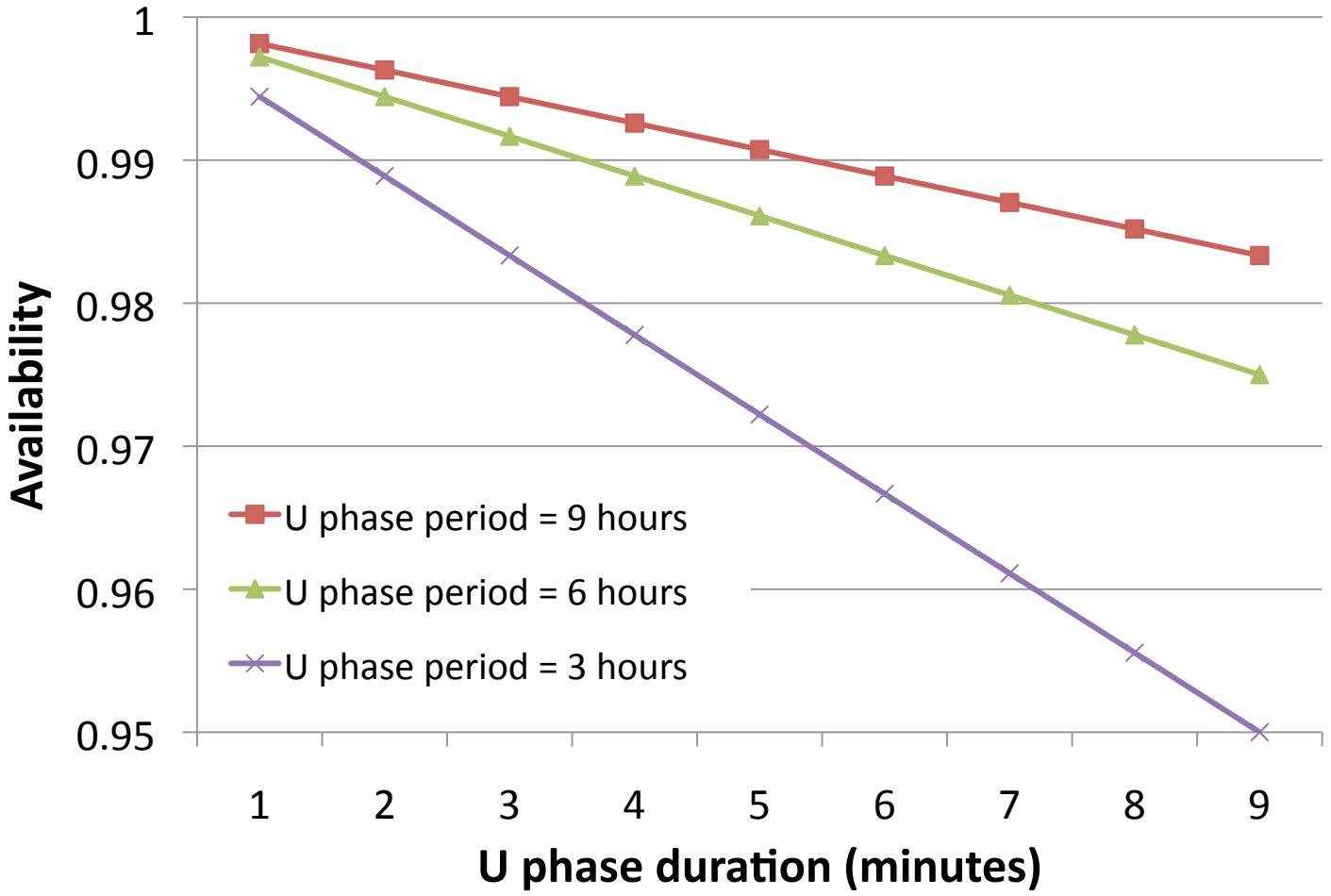
Audit/Repair time

Data loss (%)	Audit/Repair Time (s) Mean (std)
0	554.5 (54.6)
1	612.9 (30.3)
10	1147.6 (33.3)
100	3521.5 (201.6)

U phase duration

Action	Time (s) Mean (std)
Reboot	86.6 (2.1)
Proposal creation	8.0 (4.0)
Agreement	5.2 (1.0)
AST update/Checkpoint	271.1 (24.8)
Total	**370.9 (24.0)**

Availability



Related work

- BFT systems
  - PBFT, PBFT-PR, COCA
  - BFT-2F, A2M-PBFT, A2M
  - BFT erasure-coded storage
- Differentiating trust levels
  - Hybrid system model – wormholes model
  - Hybrid fault model
  - Different fault thresholds to different sites or clusters
- Long-term stores
  - Self-certifying bitstore – Antiquity, Oceanstore, Pergamum, Glacier, etc.
  - LOCKSS, POTSHARDS, CATS

Conclusion

- Present a tiered fault-tolerant system framework
  - A2M (SOSP07), Bonafide (FAST09), TrInc (NSDI09)

- Build Bonafide, a safer key-value store (of non-self-certifying data) for long-term integrity with the framework

Thank you!