The Atomic Distributed Object Model for Distributed System Verification

PhD Dissertation Defense

Wolf Honoré

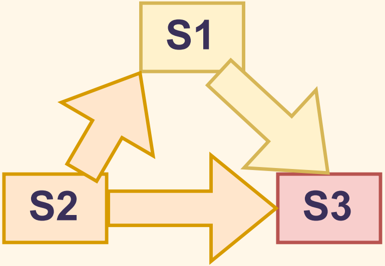
Yale University

August 19, 2022

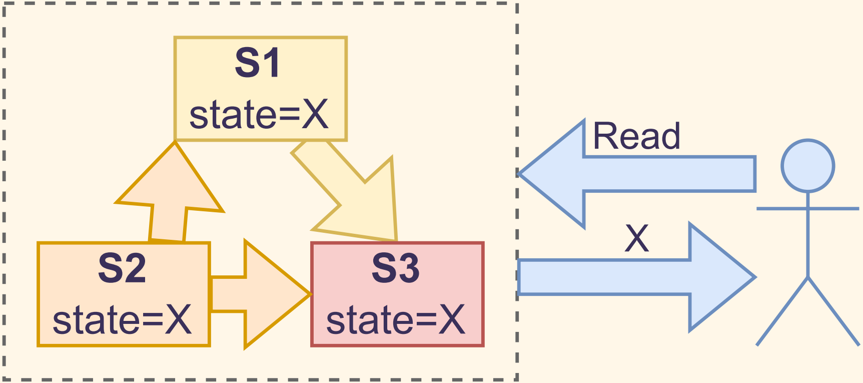
Roadmap

- **Motivation**
    - What is a distributed system?
    - What is formal verification?
    - Why are they important?
- ADO Overview
- Case Study: Advert
- Case Study: Adore
- Case Study: AdoB
- Conclusions

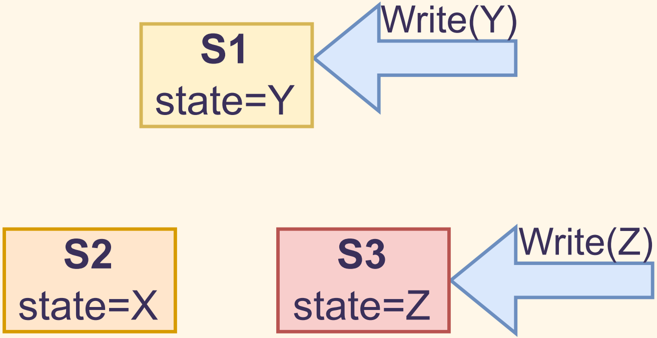
What is a Distributed System?



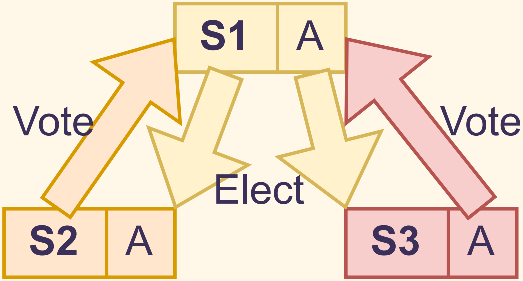
What is a Distributed System?



Replication: Challenges

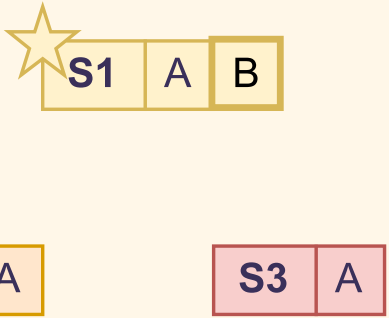


Consensus: Reaching Agreement



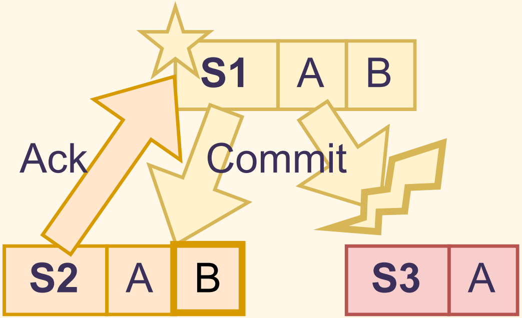
election:
S1 collects
votes

Consensus: Reaching Agreement



local update:
S1 applies B

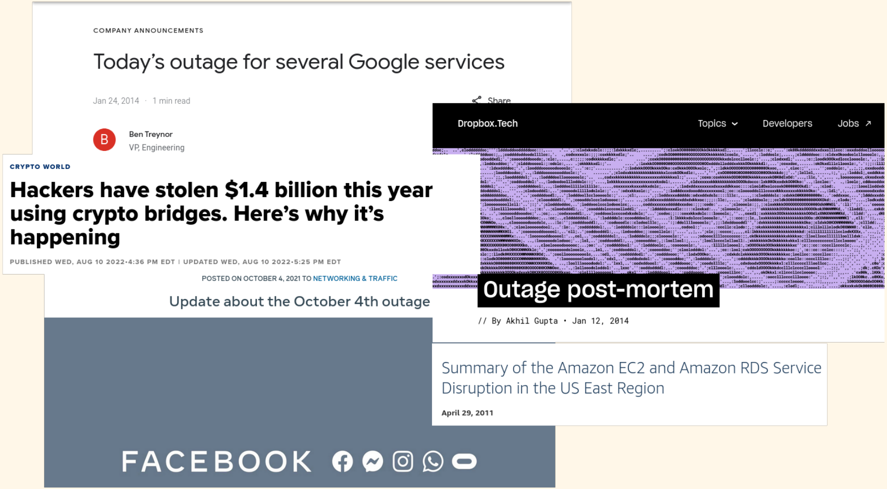
Consensus: Reaching Agreement



commit:
S1 replicates B

2 out of 3 is sufficient

What Can Go Wrong?



COMPANY ANNOUNCEMENTS

Today's outage for several Google services

Jan 24, 2014 · 1 min read

B Ben Treynor
VP, Engineering

CRYPTO WORLD

Hackers have stolen $1.4 billion this year using crypto bridges. Here's why it's happening

PUBLISHED WED, AUG 10 2022·4:36 PM EDT | UPDATED WED, AUG 10 2022·5:25 PM EDT

POSTED ON OCTOBER 4, 2021 TO NETWORKING & TRAFFIC

Update about the October 4th outage

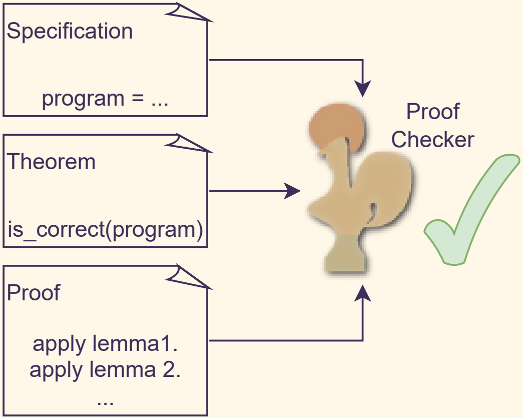
Outage post-mortem

// By Akhil Gupta • Jan 12, 2014

Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region

April 29, 2011

Formal Verification: Proving Correctness



Specification

program = ...

Theorem

is_correct(program)

Proof

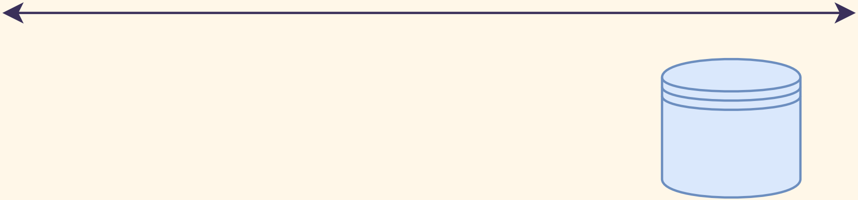
apply lemma1.
apply lemma 2.
...

Proof Checker

Abstraction Layers

Abstraction Layers

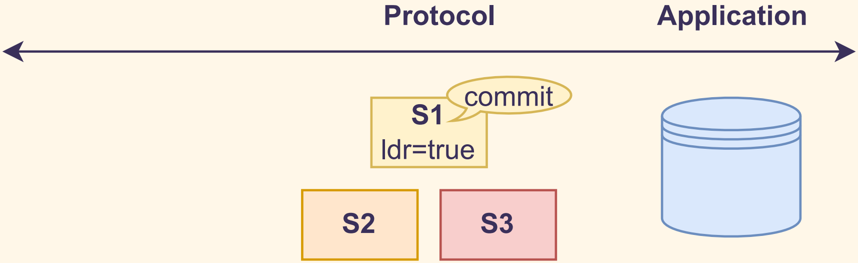
Application



Theorem

write(x) followed by
read() returns x

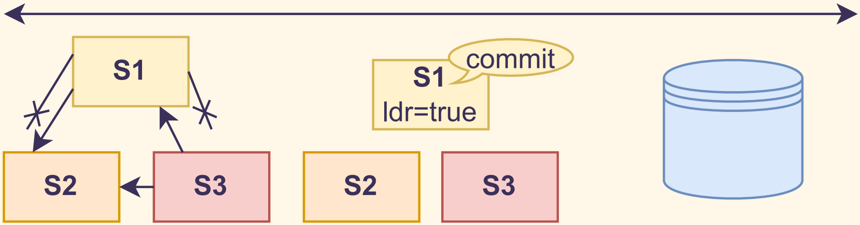
Abstraction Layers



Theorem
Only leaders can commit

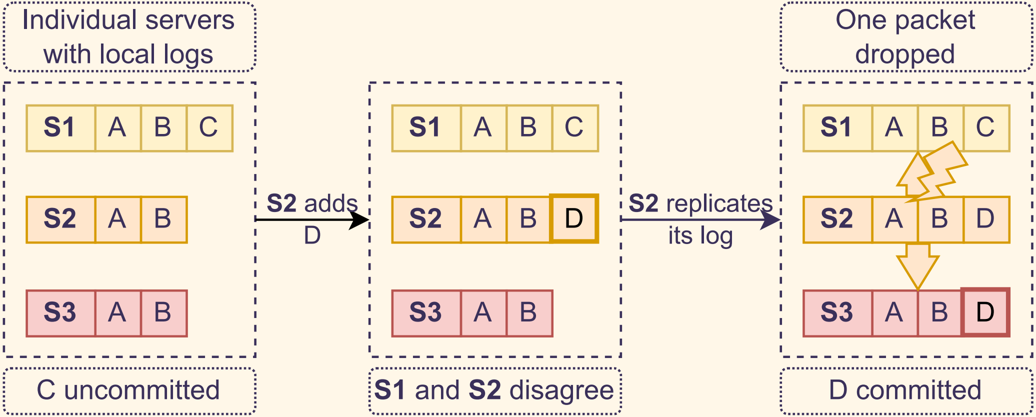
Abstraction Layers

Network Protocol Application



Theorem
Messages are
delivered in order

Network-Based Models



Individual servers with local logs

S1 A B C

S2 A B

S3 A B

C uncommitted

S2 adds D

S1 A B C

S2 A B D

S3 A B

S1 and S2 disagree

S2 replicates its log

One packet dropped

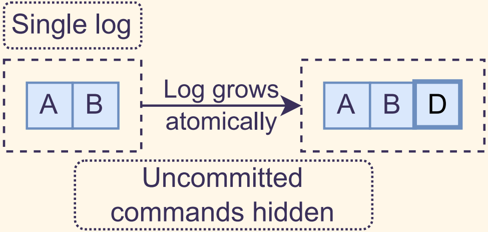
S1 A B C

S2 A B D

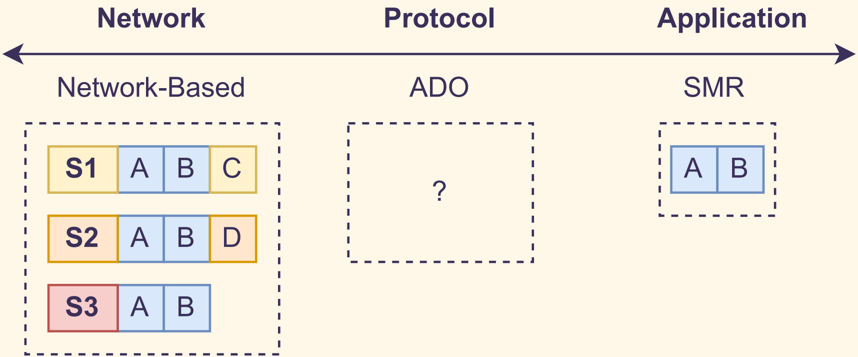
S3 A B D

D committed

State Machine Replication (SMR)



Abstraction Spectrum



Prior Consensus Verification Work

IronFleet (SOSP '15) Semi-automates refining network-level specifications with SMT.

Verdi (PLDI '15) Transforms simplified network specifications into more fault-tolerant equivalents.

Paxos Made EPR (OOPSLA '17) Reduces the safety of Paxos to a decidable first-order logic.

Velisarios (ESOP '18) Proves PBFT's safety using happens-before relations on network events.

Aneris (ESOP '20) Supports modular network-based specifications with thread-level concurrency.

Contributions

- ADO Model: A novel, protocol-level model for consensus.

Contributions

- ADO Model: A novel, protocol-level model for consensus.
- Compositional distributed application reasoning.

Contributions

- ADO Model: A novel, protocol-level model for consensus.
- Compositional distributed application reasoning.
- Safety and liveness proofs.
  - First to support hot reconfiguration.
  - First to generically support benign and byzantine failures.

Contributions

- ADO Model: A novel, protocol-level model for consensus.
- Compositional distributed application reasoning.
- Safety and liveness proofs.
  - First to support hot reconfiguration.
  - First to generically support benign and byzantine failures.
- Refinement with multiple protocols.
  - Paxos (single, multi, vertical, CAS)
  - Chain Replication
  - Raft
  - Jolteon

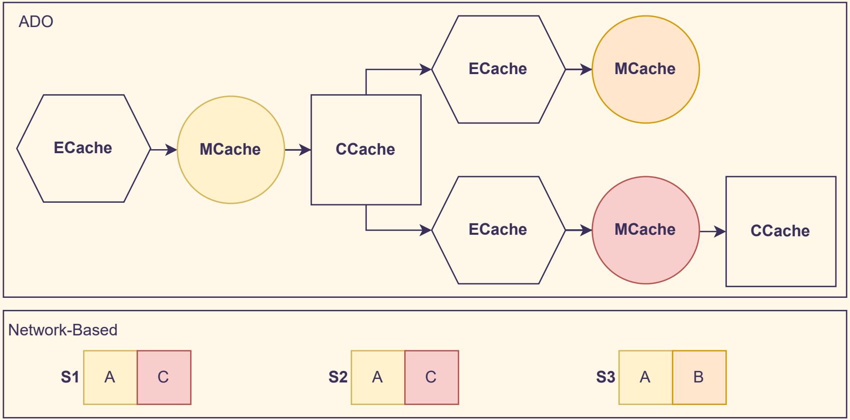
Acknowledgments

- Jieung Kim: Paxos safety and refinement.
- Ji-Yong Shin: Paxos refinement, OCaml extraction, performance experiments.
- Longfei Qiu: Jolteon refinement.
- Yoonseung Kim: Jolteon refinement.

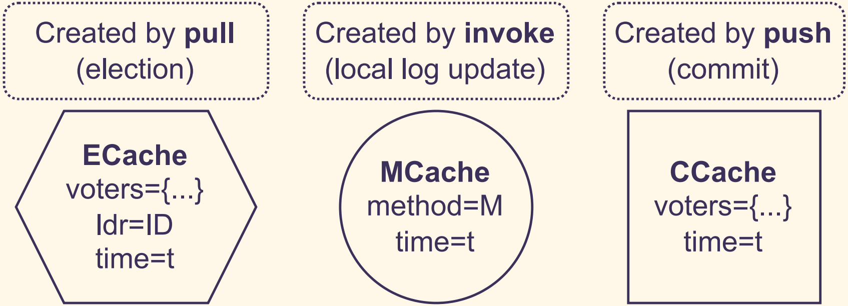
Roadmap

- Motivation
- **ADO Overview**
  - Atomic Distributed Objects
  - Global state representation (*cache tree*).
  - Atomic interface (*pull*, *invoke*, *push*).
- Case Study: Advert
- Case Study: Adore
- Case Study: AdoB
- Conclusions

ADO State — Cache Tree



ADO State — Cache Tree



Created by **pull** (election)

ECache
voters={...}
ldr=ID
time=t

Created by **invoke** (local log update)

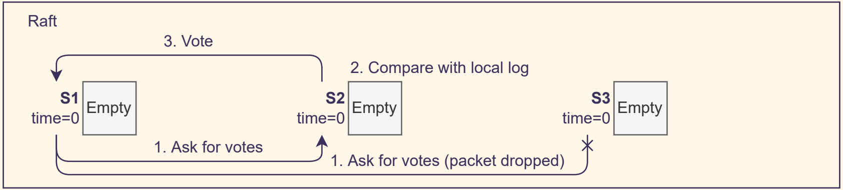
MCache
method=M
time=t

Created by **push** (commit)

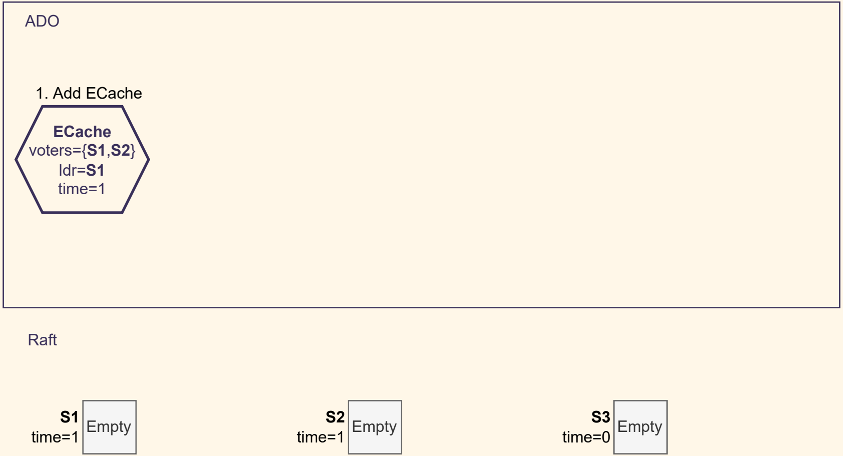
CCache
voters={...}
time=t

ADO API — Pull

ADO



ADO API — Pull



ADO

1. Add ECache

ECache
voters={**S1**,**S2**}
ldr=**S1**
time=1

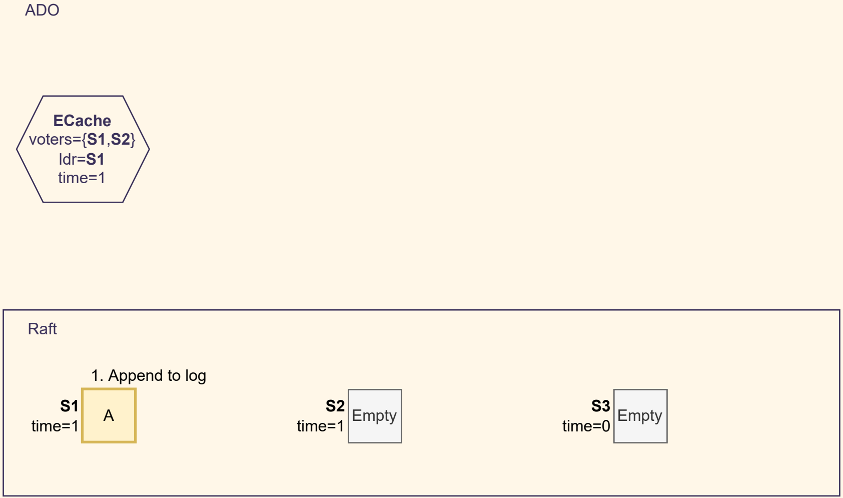
Raft

S1
time=1
Empty

S2
time=1
Empty

S3
time=0
Empty

ADO API — Invoke



ADO

ECache
voters={**S1**,**S2**}
ldr=**S1**
time=1

Raft

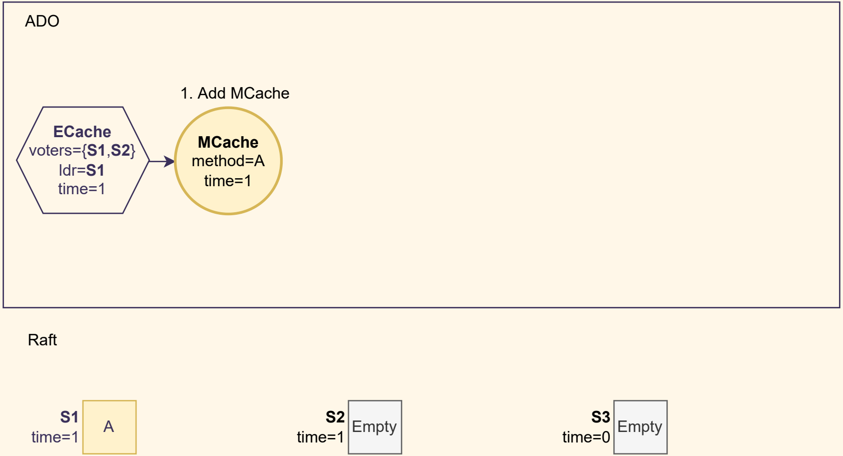
1. Append to log

S1
time=1
A

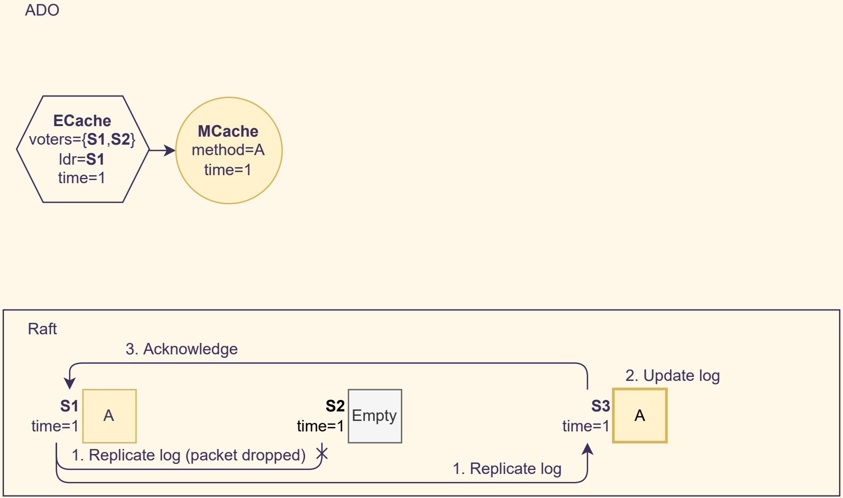
S2
time=1
Empty

S3
time=0
Empty

ADO API — Invoke



ADO API — Push



ADO

ECache
voters={**S1**,**S2**}
ldr=**S1**
time=1

MCache
method=A
time=1

Raft

3. Acknowledge

2. Update log

S1
time=1
A

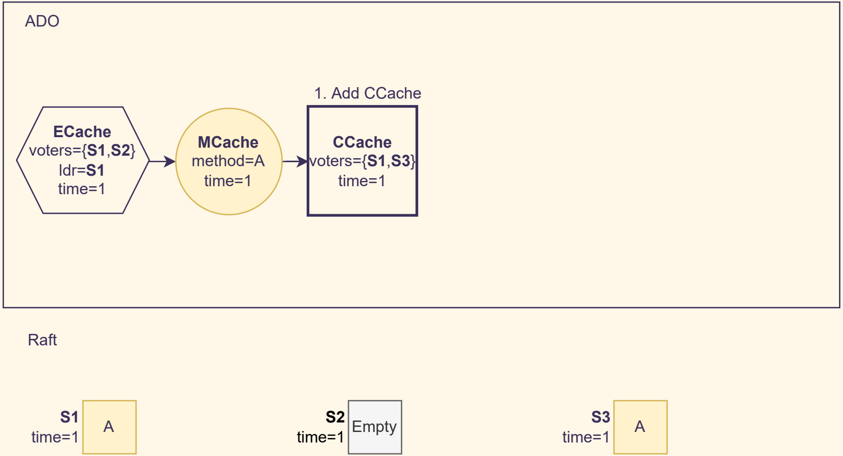
S2
time=1
Empty

S3
time=1
A

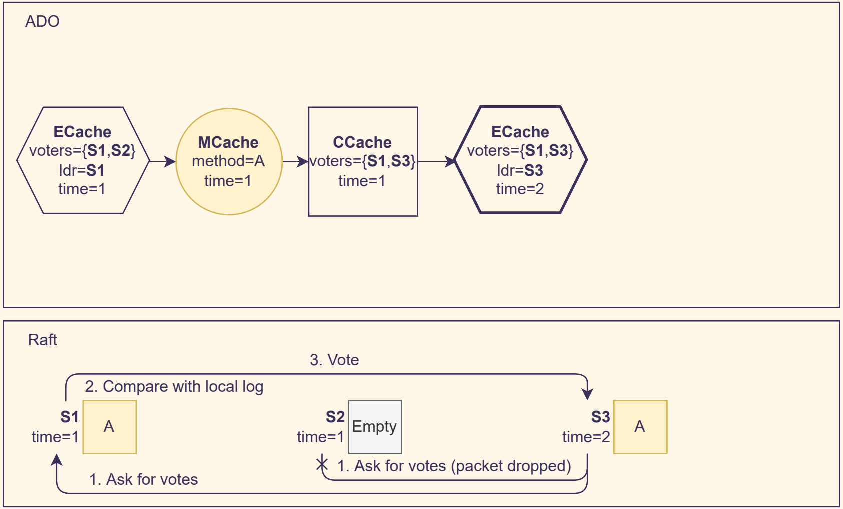
1. Replicate log (packet dropped)

1. Replicate log

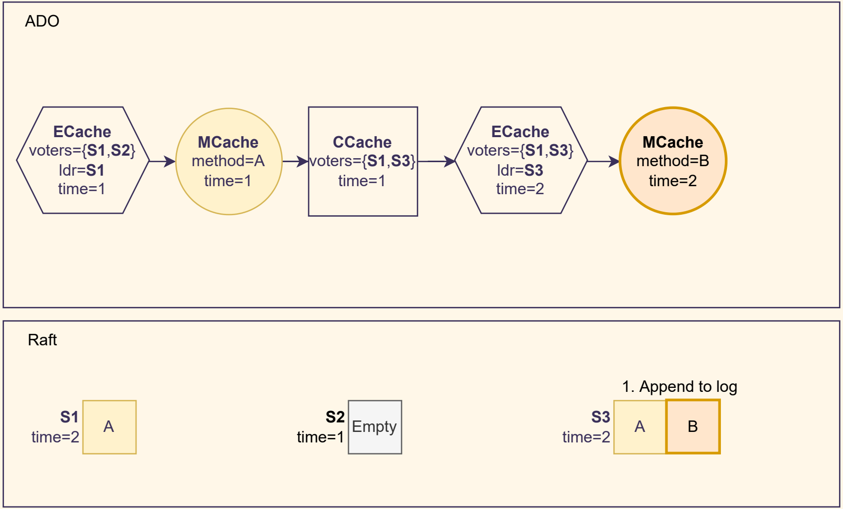
ADO API — Push


ADO
1. Add CCache
ECache
voters={**S1**,**S2**}
ldr=**S1**
time=1
MCache
method=A
time=1
CCache
voters={**S1**,**S3**}
time=1
Raft
S1
time=1
A
S2
time=1
Empty
S3
time=1
A

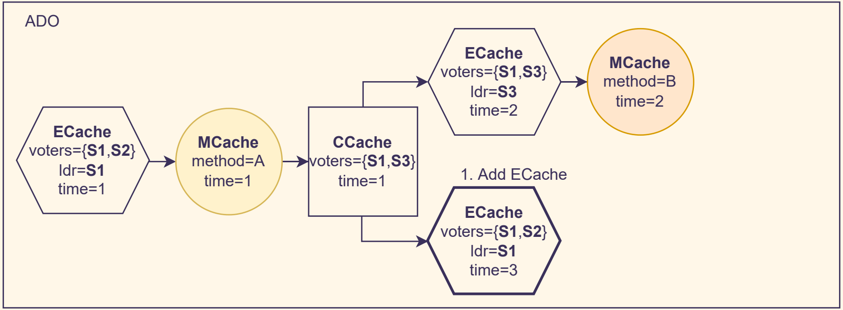
ADO API — Steady State

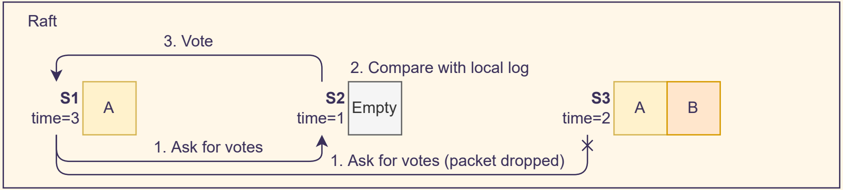


ADO API — Steady State

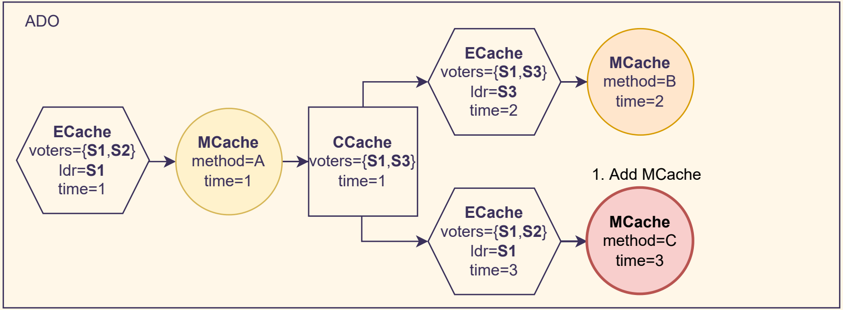


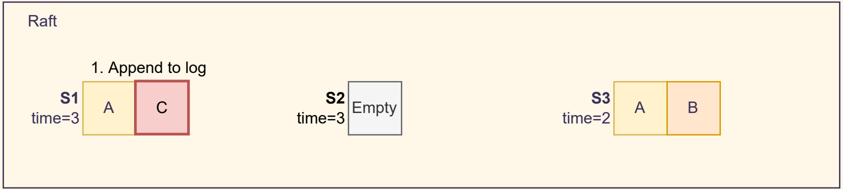
ADO API — Branching



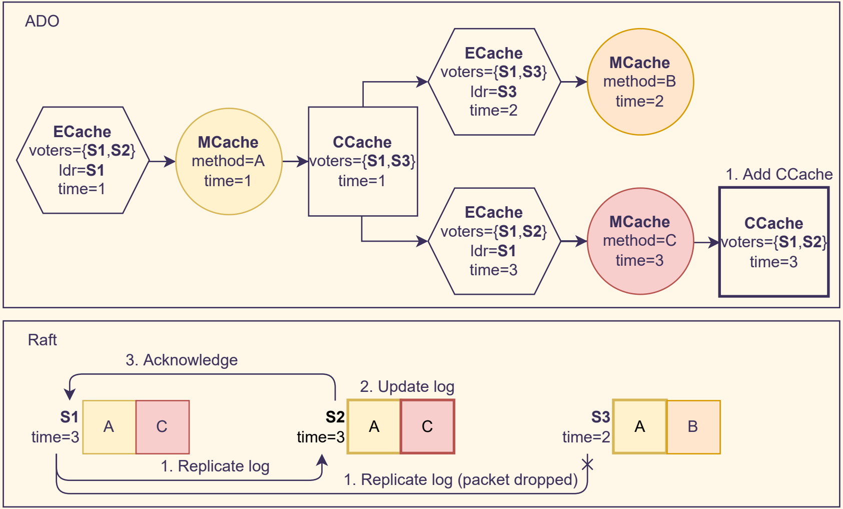


ADO API — Branching

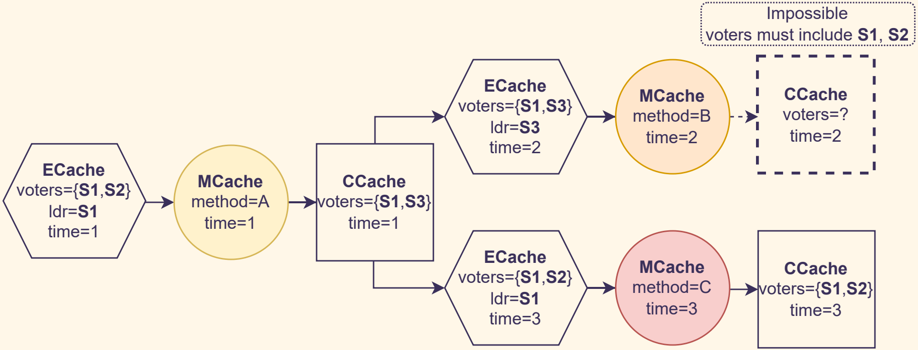




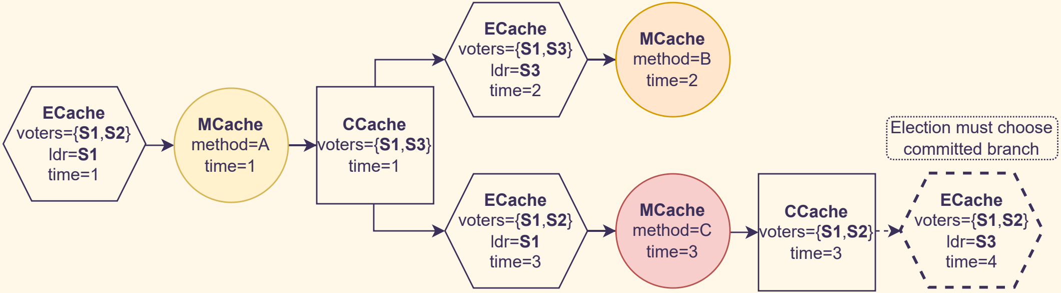
ADO API — Branching


ADO
ECache voters={S1,S2} ldr=S1 time=1
MCache method=A time=1
CCache voters={S1,S3} time=1
ECache voters={S1,S3} ldr=S3 time=2
MCache method=B time=2
ECache voters={S1,S2} ldr=S1 time=3
MCache method=C time=3
1. Add CCache
CCache voters={S1,S2} time=3
Raft
3. Acknowledge
2. Update log
S1 time=3
A
C
S2 time=3
A
C
S3 time=2
A
B
1. Replicate log
1. Replicate log (packet dropped)

Safety


Impossible
voters must include **S1**, **S2**
ECache
voters={**S1**,**S2**}
ldr=**S1**
time=1
MCache
method=A
time=1
CCache
voters={**S1**,**S3**}
time=1
ECache
voters={**S1**,**S3**}
ldr=**S3**
time=2
MCache
method=B
time=2
CCache
voters=?
time=2
ECache
voters={**S1**,**S2**}
ldr=**S1**
time=3
MCache
method=C
time=3
CCache
voters={**S1**,**S2**}
time=3

Safety



Election must choose committed branch

Roadmap

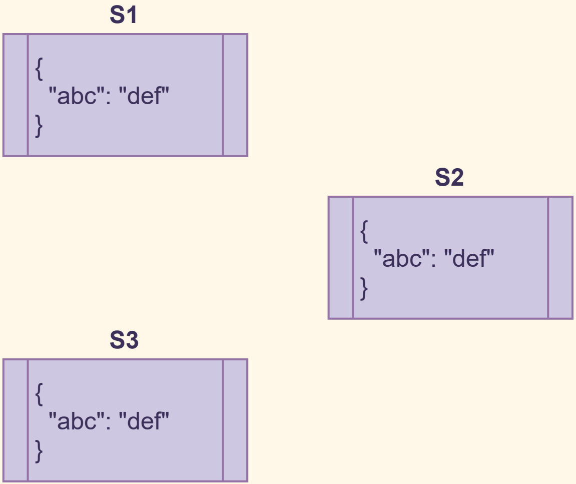
- Motivation
- ADO Overview
- **Case Study: Advert**
  - Atomic Distributed Object Verification Toolchain
  - Expose partial failures for distributed application optimization.
  - Support ADO composition.
- Case Study: Adore
- Case Study: AdoB
- Conclusions

Distributed Applications with Partial Failures

> *Partial failure is a central reality of distributed computing. [. . .] Being robust in the face of partial failure requires some expression at the interface level.*
> *(Jim Waldo. A Note on Distributed Computing. 1994)*

- Unavoidable feature unique to distributed systems.
- Interact with all aspects of distributed protocols (e.g., leader election and reconfiguration).
- Can be used for performance optimizations.
  - TAPIR (SOSP '15): Transactions with out-of-order commits.
  - Speculator (SOSP '05): Speculative distributed file system.

Distributed Applications with Partial Failures



S1

```
{
  "abc": "def"
}
```

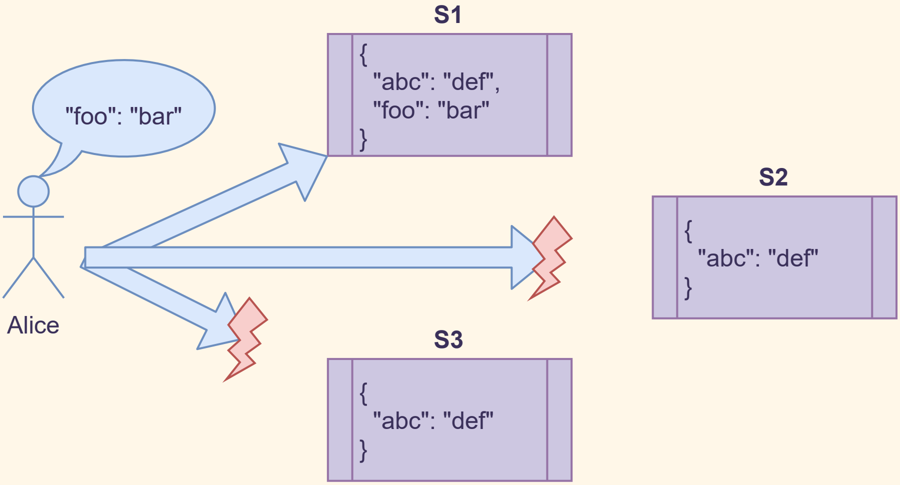
S2

```
{
  "abc": "def"
}
```

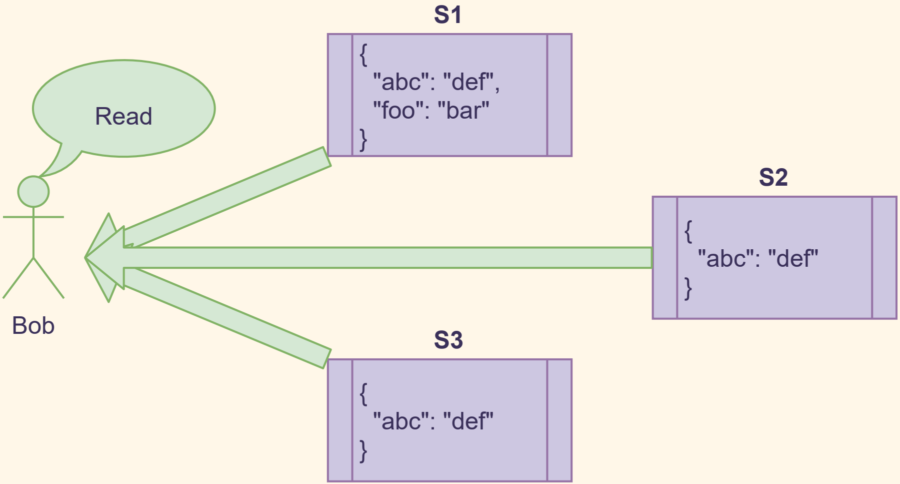
S3

```
{
  "abc": "def"
}
```

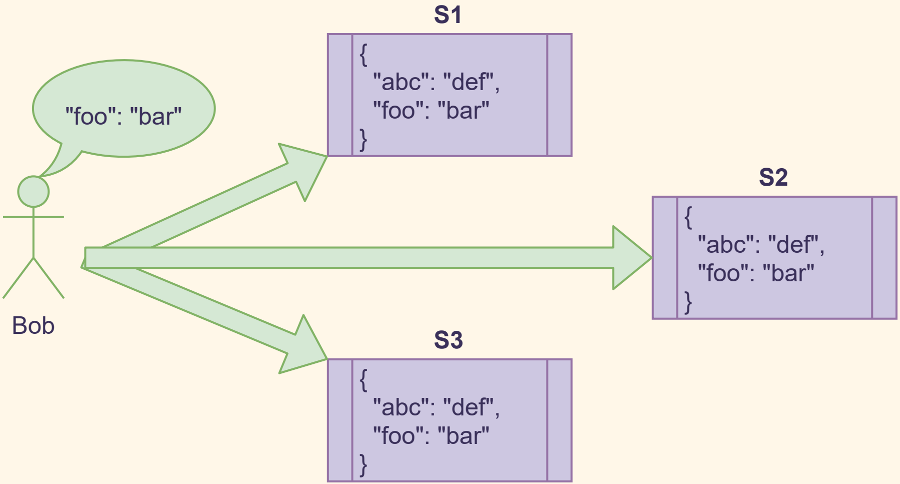
Distributed Applications with Partial Failures



Distributed Applications with Partial Failures



Distributed Applications with Partial Failures



S1:

```
{
  "abc": "def",
  "foo": "bar"
}
```

S2:

```
{
  "abc": "def",
  "foo": "bar"
}
```

S3:

```
{
  "abc": "def",
  "foo": "bar"
}
```

Distributed Applications

```
ADO KV {
  shared kv : [string ⋆ int] := [];
  method set(k, v) { this.kv[hash(k)] := (v, len(v)); }
  method get(k) { return this.kv[hash(k)][0]; }
  method getmeta(k) { return this.kv[hash(k)][1]; }
}
```

Distributed Applications

```
ADO DVec[T] {
  shared data : [T] := [];
  method insert(idx, x) { this.data[idx] := x; }
  method get(idx) { return this.data[idx]; }
}
ADO DLock {
  shared owner : option N := None;
  method tryAcquire() { ... }
  method release() { ... }
}
DApp KVLock(lk: DLock, data: DVec[string], meta: DVec[int]) {
  proc set(k, v) {
    ... /* acquire, set data, set meta, release */
  }
  ... /* get, getmeta */
}
```

Distributed Applications

```
DApp KVLock(lk: DLock, data: DVec[string], meta: DVec[int]) {
  proc set(k, v) {
    lk.pull();




  }
}
```

Distributed Applications

```
DApp KVLock(lk: DLock, data: DVec[string], meta: DVec[int]) {
  proc set(k, v) {
    while (lk.pull() == FAIL) {}




  }
}
```

Distributed Applications

```
DApp KVLock(lk: DLock, data: DVec[string], meta: DVec[int]) {
  proc set(k, v) {
    while (lk.pull() == FAIL) {}
    ok := lk.invoke(tryAcquire());



  }
}
```

Distributed Applications

```
DApp KVLock(lk: DLock, data: DVec[string], meta: DVec[int]) {
  proc set(k, v) {
    while (lk.pull() == FAIL) {}
    ok := lk.invoke(tryAcquire());
    while (lk.push() == FAIL) {}
    if (!ok) { return; }
    /* ... */
  }
}
```

Handling Failures

```
DApp KVLockAbort(lk: DLock, data: DVec[string], meta: DVec[int]) {
  proc set(k, v) {
    if (lk.pull() == FAIL) { return; }
    ok := lk.invoke(tryAcquire());
    if (lk.push() == FAIL) { return; }
    if (!ok) { return; }
    /* ... */
  }
}
```

Handling Failures

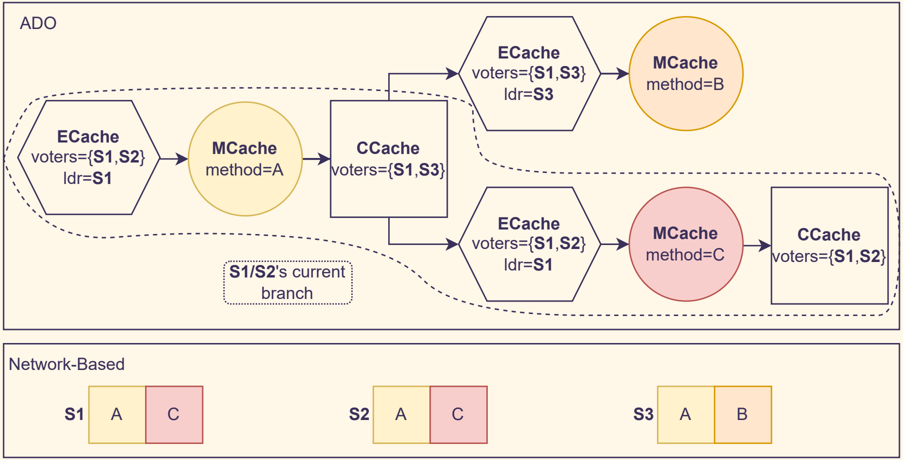
```
DApp KVLockRetry(lk: DLock, data: DVec[string], meta: DVec[int]) {
  proc set(k, v) {
    for retry in 0..N {
      if (lk.pull() == FAIL) { continue; }
      ok := lk.invoke(tryAcquire());
      if (lk.push() == FAIL) { continue; }
      if (!ok) { continue; }
    }
    if (retry == N) { return; }
    /* ... */
  }
}
```

Handling Failures

```
obj.m()! :=
  while (obj.pull() == FAIL) {}
  obj.invoke(m());
  while (obj.push() == FAIL) {}

DApp KVLock(lk: DLock, data: DVec[string], meta: DVec[int]) {
  proc set(k, v) {
    ok := lk.tryAcquire()!;
    if (!ok) { return; }
    data.insert(hash(k), v)!;
    meta.insert(hash(k), len(v))!;
    lk.release()!;
  }
}
```

End-to-End Verification



ADO

ECache voters={**S1**,**S2**} ldr=**S1**

MCache method=A

CCache voters={**S1**,**S3**}

ECache voters={**S1**,**S3**} ldr=**S3**

MCache method=B

ECache voters={**S1**,**S2**} ldr=**S1**

MCache method=C

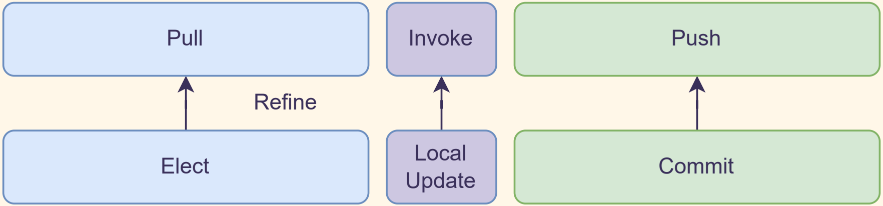
CCache voters={**S1**,**S2**}

S1/S2's current branch

Network-Based

S1	A	C
S2	A	C
S3	A	B

End-to-End Verification



Pull

Invoke

Push

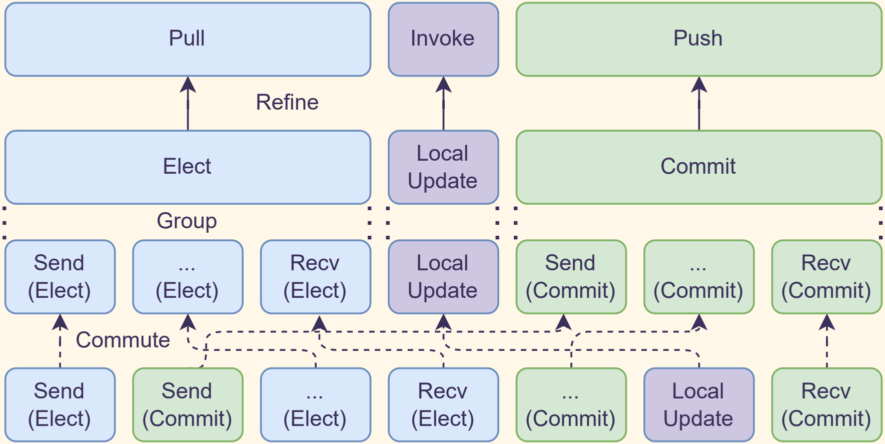
Refine

Elect

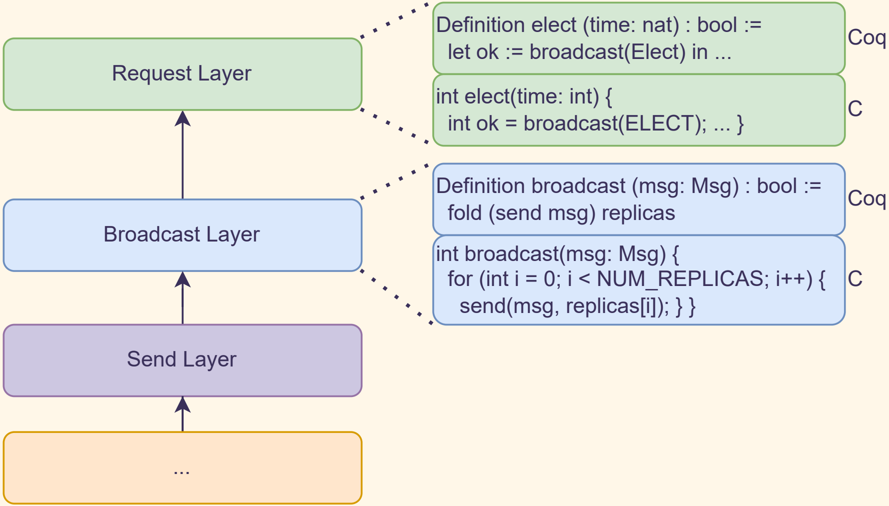
Local Update

Commit

End-to-End Verification



End-to-End Verification



Coq

```
Definition elect (time: nat) : bool :=
  let ok := broadcast(Elect) in ...
```

C

```
int elect(time: int) {
  int ok = broadcast(ELECT); ... }
```

Coq

```
Definition broadcast (msg: Msg) : bool :=
  fold (send msg) replicas
```

C

```
int broadcast(msg: Msg) {
  for (int i = 0; i < NUM_REPLICAS; i++) {
    send(msg, replicas[i]); } }
```

Proof Effort

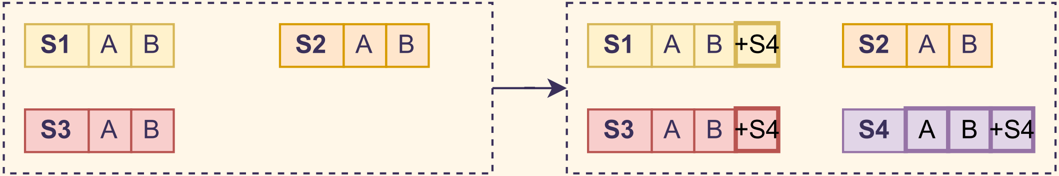
	Proof LOC (Coq)
KVLock DApp	∼600
KVLockFree DApp	∼300
2PC DApp	∼600
Generic Paxos Refinement	∼5k
Chain Replication Refinement	∼2k
Shared Libraries	∼11k
Multi Paxos C Refinement	∼44k
Single Paxos	∼80
Multi Paxos	∼90
Vertical Paxos	∼100
CASPaxos	∼80

Roadmap

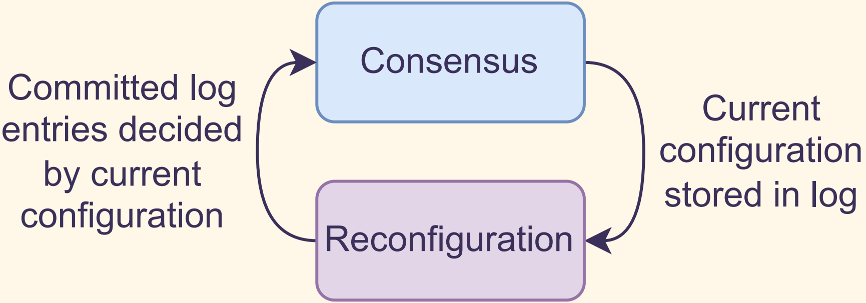
- Motivation
- ADO Overview
- Case Study: Advert
- **Case Study: Adore**
  - Atomic Distributed Objects with Certified Reconfiguration
  - Prove safety at the ADO level.
  - Support hot reconfiguration.
- Case Study: AdoB
- Conclusions

Reconfiguration

Hot Reconfiguration



Reconfiguration



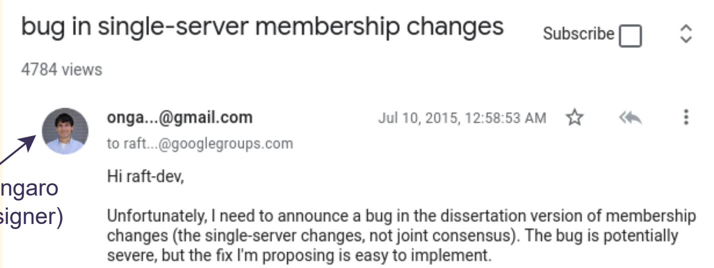
Committed log entries decided by current configuration

Consensus

Reconfiguration

Current configuration stored in log

Reconfiguration



bug in single-server membership changes

Subscribe □

4784 views

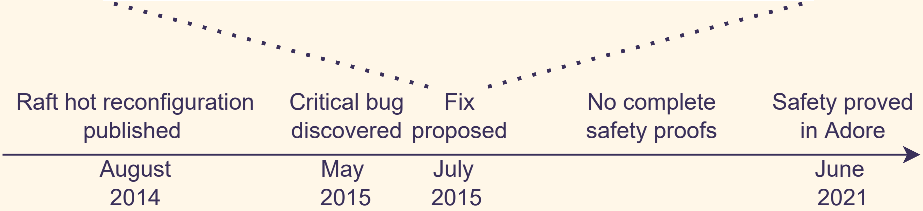
onga...@gmail.com Jul 10, 2015, 12:58:53 AM

to raft...@googlegroups.com

Hi raft-dev,

Unfortunately, I need to announce a bug in the dissertation version of membership changes (the single-server changes, not joint consensus). The bug is potentially severe, but the fix I'm proposing is easy to implement.

Diego Ongaro
(Raft designer)



Raft hot reconfiguration published — August 2014

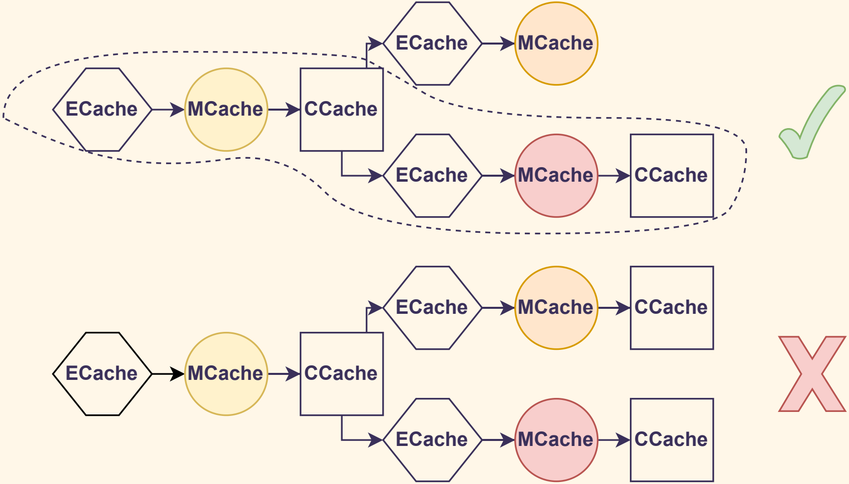
Critical bug discovered — May 2015

Fix proposed — July 2015

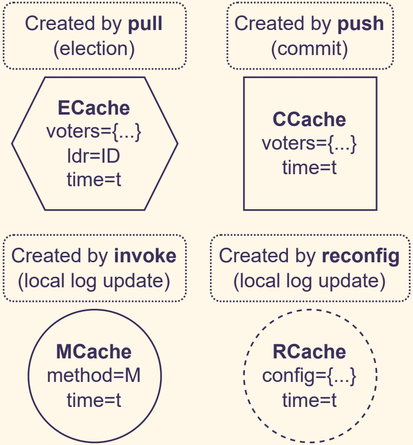
No complete safety proofs

Safety proved in Adore — June 2021

Safety in Adore



Reconfiguration in Adore



Created by **pull** (election)

ECache
voters={...}
ldr=ID
time=t

Created by **push** (commit)

CCache
voters={...}
time=t

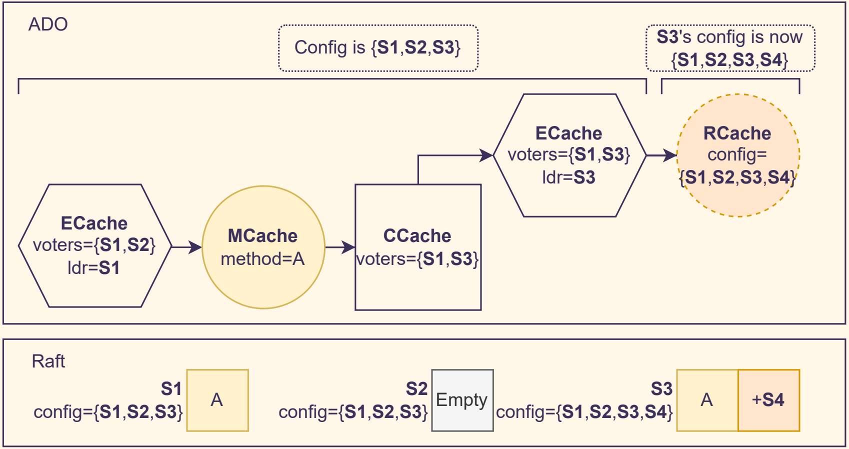
Created by **invoke** (local log update)

MCache
method=M
time=t

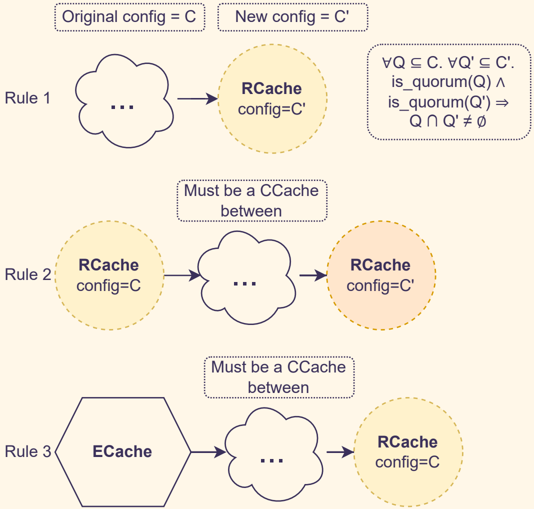
Created by **reconfig** (local log update)

RCache
config={...}
time=t

Reconfiguration in Adore



Reconfiguration Rules



Original config = C

New config = C'

Rule 1

...

RCache
config=C'

$$\forall Q \subseteq C.\ \forall Q' \subseteq C'.\ \text{is\_quorum}(Q) \wedge \text{is\_quorum}(Q') \Rightarrow Q \cap Q' \neq \emptyset$$

Must be a CCache between

Rule 2

RCache
config=C

...

RCache
config=C'

Must be a CCache between

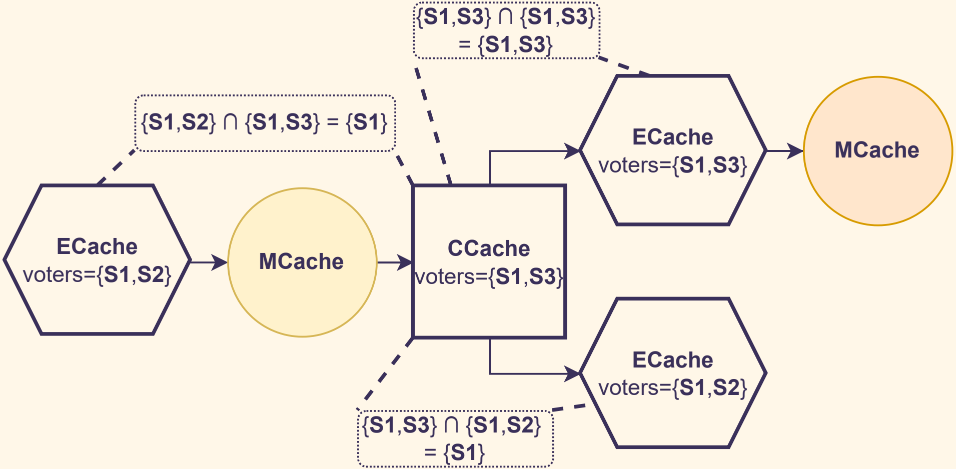
Rule 3

ECache

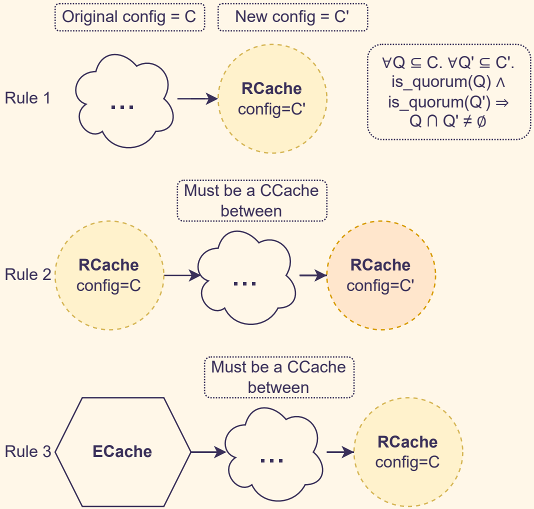
...

RCache
config=C

Reconfiguration Rules



Reconfiguration Rules



Original config = C

New config = C'

Rule 1

∀Q ⊆ C. ∀Q' ⊆ C'.
is_quorum(Q) ∧
is_quorum(Q') ⇒
Q ∩ Q' ≠ ∅

Must be a CCache between

Rule 2

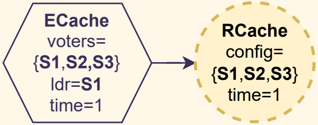
Must be a CCache between

Rule 3

Reconfiguration Rules

Original config = {**S1**,**S2**,**S3**,**S4**}

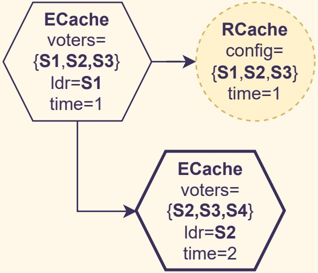
No Rule 3 leads to a safety bug



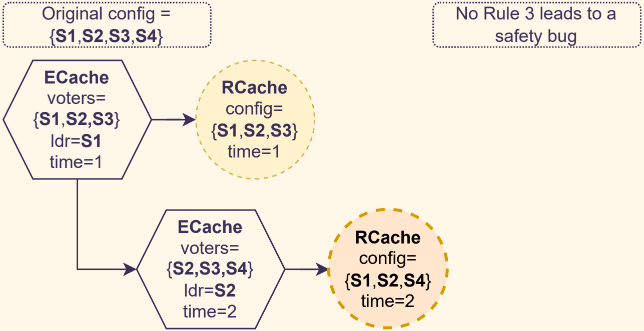
Reconfiguration Rules

Original config = {**S1**,**S2**,**S3**,**S4**}

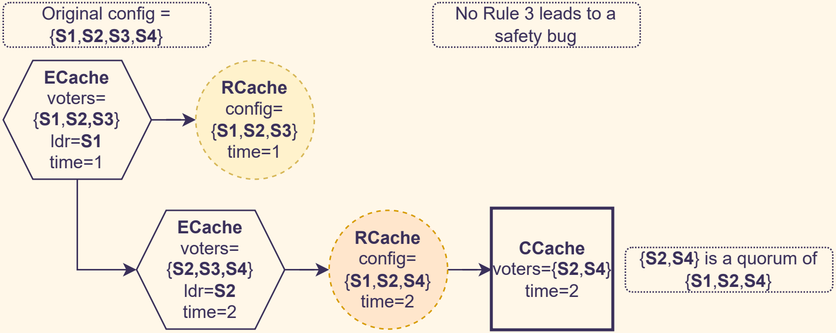
No Rule 3 leads to a safety bug



Reconfiguration Rules



Reconfiguration Rules



Original config = {**S1**,**S2**,**S3**,**S4**}

No Rule 3 leads to a safety bug

ECache voters= {**S1**,**S2**,**S3**} ldr=**S1** time=1

RCache config= {**S1**,**S2**,**S3**} time=1

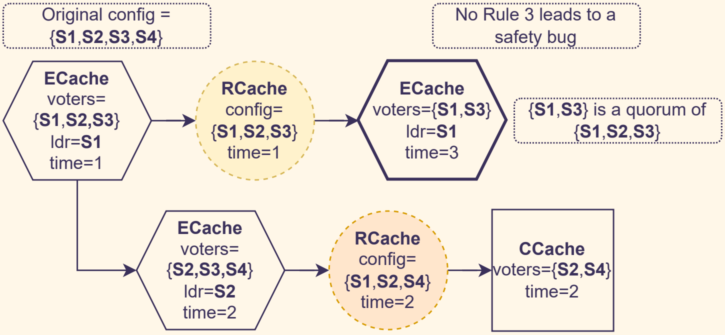
ECache voters= {**S2**,**S3**,**S4**} ldr=**S2** time=2

RCache config= {**S1**,**S2**,**S4**} time=2

CCache voters={**S2**,**S4**} time=2

{**S2**,**S4**} is a quorum of {**S1**,**S2**,**S4**}

Reconfiguration Rules

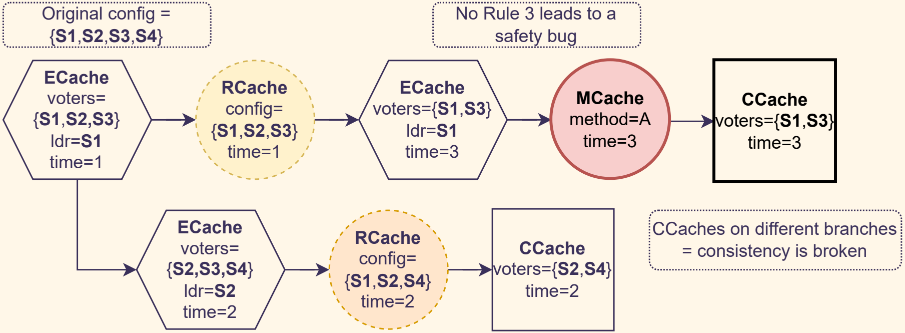


Original config = {**S1**,**S2**,**S3**,**S4**}

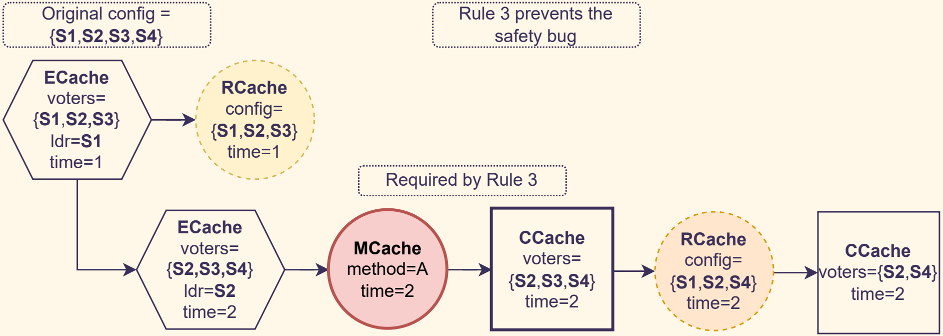
No Rule 3 leads to a safety bug

{**S1**,**S3**} is a quorum of {**S1**,**S2**,**S3**}

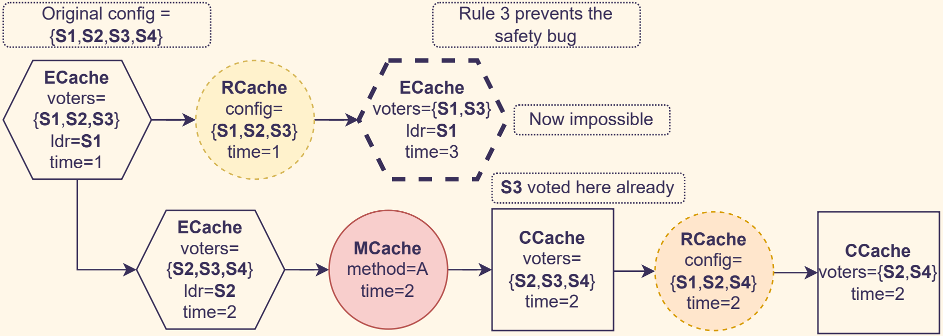
Reconfiguration Rules


Original config = {S1,S2,S3,S4}
No Rule 3 leads to a safety bug
ECache voters= {S1,S2,S3} ldr=S1 time=1
RCache config= {S1,S2,S3} time=1
ECache voters={S1,S3} ldr=S1 time=3
MCache method=A time=3
CCache voters={S1,S3} time=3
ECache voters= {S2,S3,S4} ldr=S2 time=2
RCache config= {S1,S2,S4} time=2
CCache voters={S2,S4} time=2
CCaches on different branches = consistency is broken

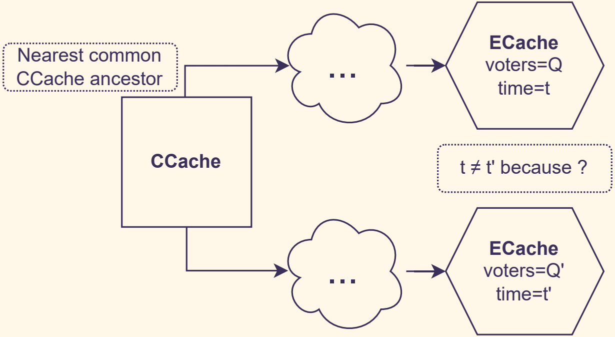
Reconfiguration Rules



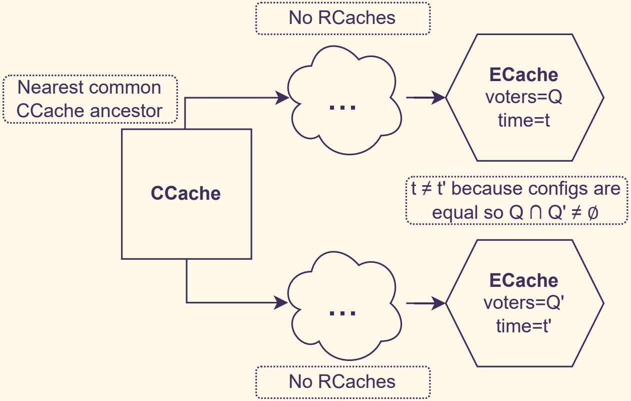
Reconfiguration Rules



Proving Safety



Proving Safety


No RCaches

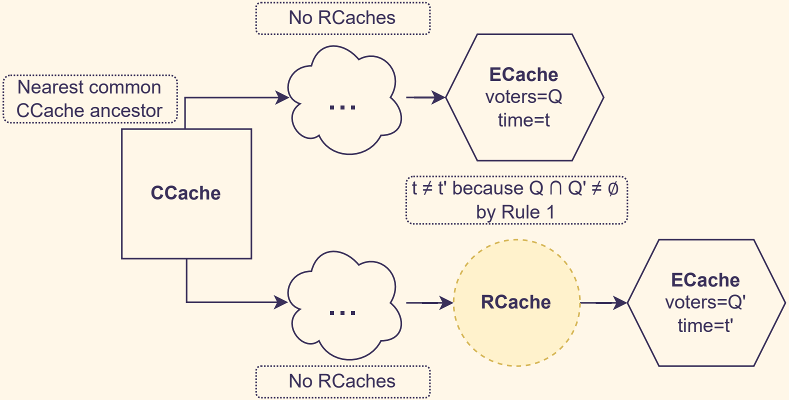
Nearest common CCache ancestor

CCache

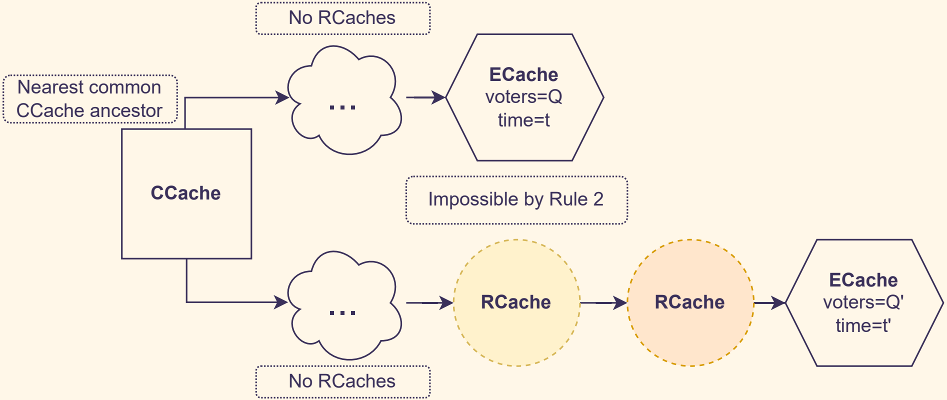
t ≠ t' because configs are equal so Q ∩ Q' ≠ ∅

No RCaches

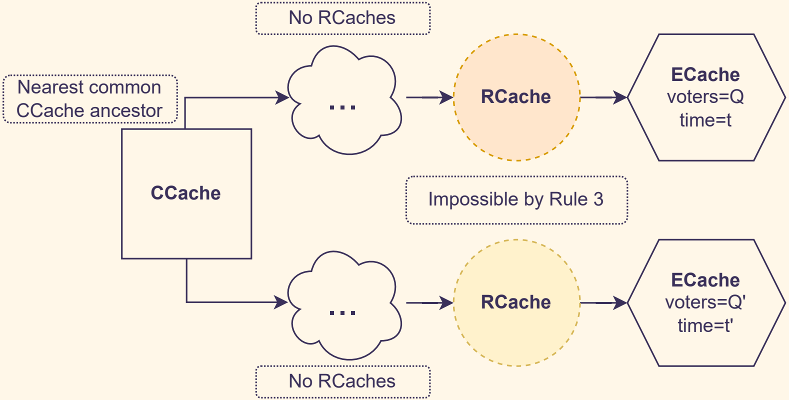
Proving Safety



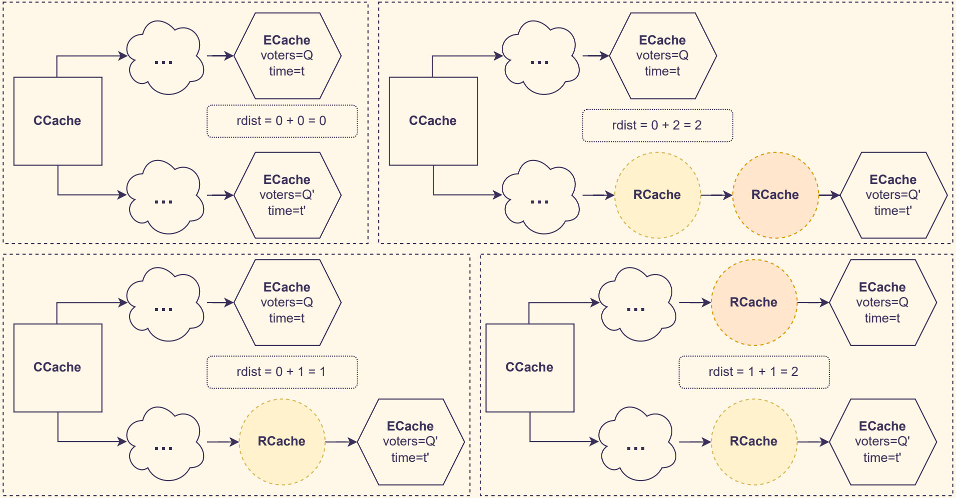
Proving Safety



Proving Safety



Proving Safety



CCache

...

ECache voters=Q time=t

rdist = 0 + 0 = 0

...

ECache voters=Q' time=t'

CCache

...

ECache voters=Q time=t

rdist = 0 + 2 = 2

...

RCache

RCache

ECache voters=Q' time=t'

CCache

...

ECache voters=Q time=t

rdist = 0 + 1 = 1

...

RCache

ECache voters=Q' time=t'

CCache

...

RCache

ECache voters=Q time=t

rdist = 1 + 1 = 2

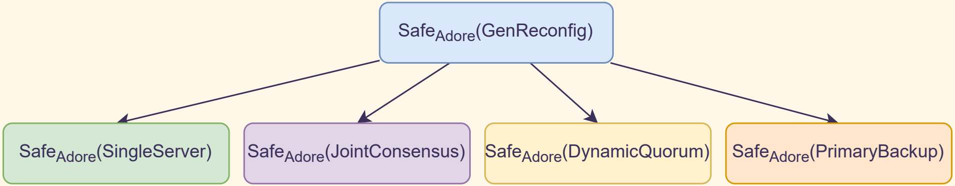
...

RCache

ECache voters=Q' time=t'

Generalized Quorums

- Safety proved once for generic reconfiguration scheme.
- A quorum is any set that guarantees overlap.
- Can be instantiated many times with minimal proof effort.


$\text{Safe}_{\text{Adore}}$(GenReconfig)
$\text{Safe}_{\text{Adore}}$(SingleServer)
$\text{Safe}_{\text{Adore}}$(JointConsensus)
$\text{Safe}_{\text{Adore}}$(DynamicQuorum)
$\text{Safe}_{\text{Adore}}$(PrimaryBackup)

Generalized Quorums

Single-Server

$$
\begin{aligned}
\mathit{Config} &\triangleq \mathit{Set}(\mathbb{N}_{\mathit{nid}}) \\
\mathit{canReconfig}(C, C') &\triangleq C = C' \lor \\
&\qquad \exists s.\, C = C' \cup \{s\} \lor C' = C \cup \{s\} \\
\mathit{isQuorum}(S, C) &\triangleq |C| < 2 * |S \cap C|
\end{aligned}
$$

Generalized Quorums

Joint Consensus

$$\begin{aligned}
\mathit{Config} &\triangleq \mathit{Set}(\mathbb{N}_{nid}) * \mathit{Option}(\mathit{Set}(\mathbb{N}_{nid})) \\
\mathit{canReconfig}(C, C') &\triangleq \exists\, \mathit{old}.\ \big(C = (\mathit{old}, \bot) \wedge C' = (\mathit{old}, \_)\big) \vee \\
&\quad\;\; \exists\, \mathit{new}.\ \big(C = (\_, \mathit{new}) \wedge C' = (\mathit{new}, \bot)\big) \\
\mathit{isQuorum}(S, (\mathit{old}, \mathit{new})) &\triangleq |\mathit{old}| < 2 * |S \cap \mathit{old}| \wedge \\
&\quad\;\; (\mathit{new} = \bot \vee |\mathit{new}| < 2 * |S \cap \mathit{new}|)
\end{aligned}$$

Generalized Quorums

Dynamic Quorum Size

$$
\begin{aligned}
\mathit{Config} &\triangleq \mathbb{N} * \mathit{Set}(\mathbb{N}_{nid}) \\
\mathit{canReconfig}((q, C), (q', C')) &\triangleq (C \subseteq C' \wedge |C'| < q + q') \vee \\
& (C' \subseteq C \wedge |C| < q + q') \\
\mathit{isQuorum}(S, (q, C)) &\triangleq q \leq |S \cap C|
\end{aligned}
$$

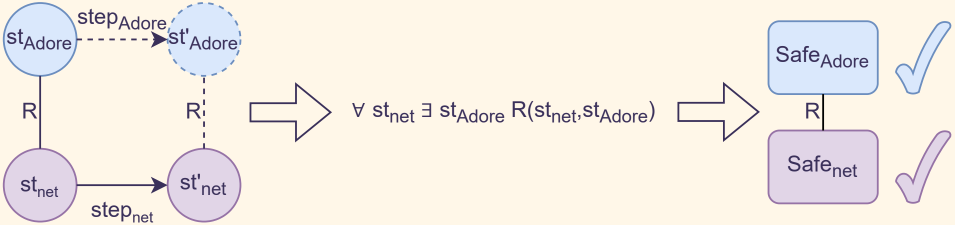
Generalized Quorums

Primary Backup

$$
\begin{aligned}
\mathit{Config} &\triangleq \mathbb{N}_{nid} * \mathit{Set}(\mathbb{N}_{nid}) \\
\mathit{canReconfig}((P, \_), (P', \_)) &\triangleq P = P' \\
\mathit{isQuorum}(S, (P, \_)) &\triangleq P \in S
\end{aligned}
$$

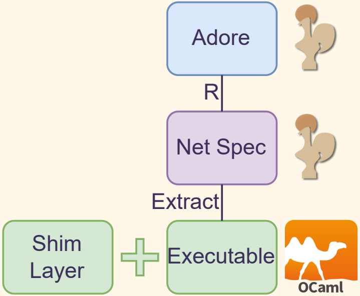
Refinement

- Refinement between Raft network-based specification and Adore.
- Also generic with respect to reconfiguration scheme.



Extraction

- Automated extraction from Coq specification to executable OCaml.
- Extracted code contains core logic, unverified shim layer handles network communication.
- Safety guaranteed through Adore and refinement.



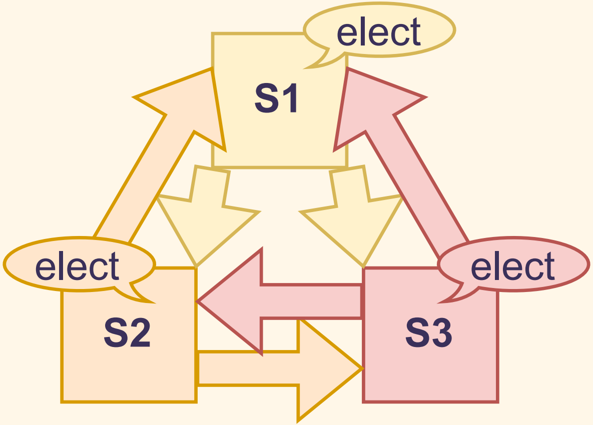
Proof Effort

	Proof LOC (Coq)	**Proof Time**
Cache Tree Library/Properties	∼6k	2 person-weeks
Safety Proof	∼4k	3 person-weeks
Refinement Proof	∼13k	9 person-weeks
Reconfiguration Schemes (6)	∼300	<1 person-week

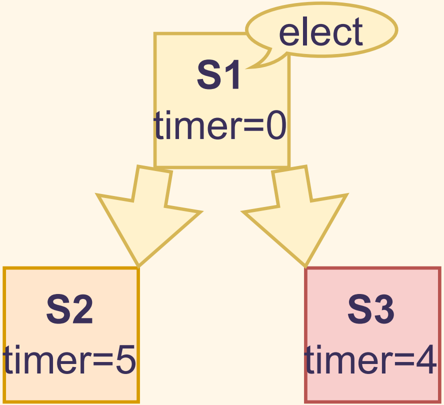
Roadmap

- Motivation
- ADO Overview
- Case Study: Advert
- Case Study: Adore
- **Case Study: AdoB**
  - Atomic Distributed Objects for Benign/Byzantine Consensus
  - Prove liveness at the ADO level.
  - Support benign and byzantine failures in a generic abstraction.
- Conclusions

Liveness

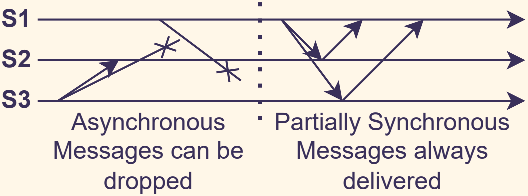


Liveness



Liveness Assumptions

- Partial synchrony



Liveness Assumptions

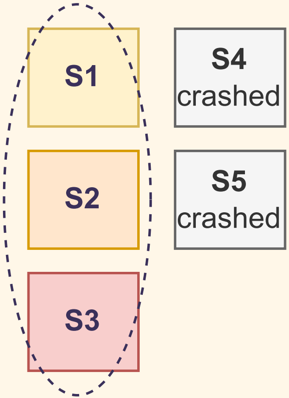
- Partial synchrony
- Productive strategy

```
if not isLeader() and timer() == 0:
  startElection()
else if isLeader() and hasUncommitted():
  startCommit()
else if timer() == 0:
  sendTimeout()
```

Liveness Assumptions

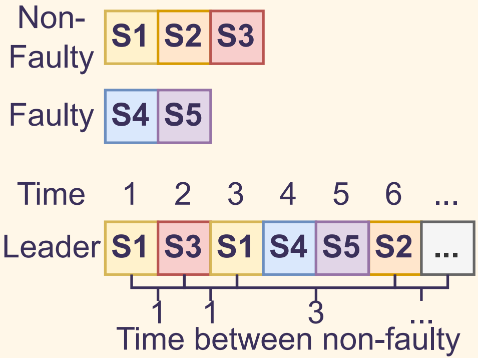
- Partial synchrony
- Productive strategy
- Non-faulty quorum

Quorum = majority = 3/5



Liveness Assumptions

- Partial synchrony
- Productive strategy
- Non-faulty quorum
- Fair election rotation



Non-Faulty: S1 S2 S3

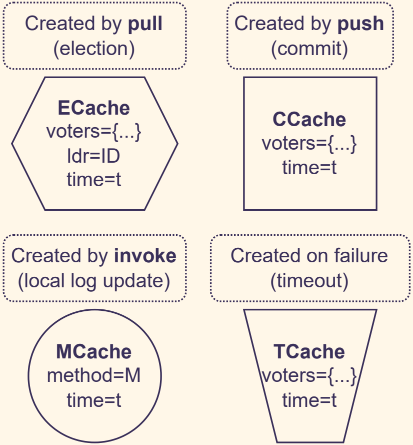
Faulty: S4 S5

Time	1	2	3	4	5	6	...
Leader	S1	S3	S1	S4	S5	S2	...

1 1 3 ...

Time between non-faulty

Time in AdoB



Created by **pull** (election)

ECache
voters={...}
ldr=ID
time=t

Created by **push** (commit)

CCache
voters={...}
time=t

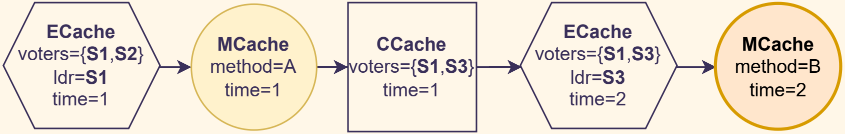
Created by **invoke** (local log update)

MCache
method=M
time=t

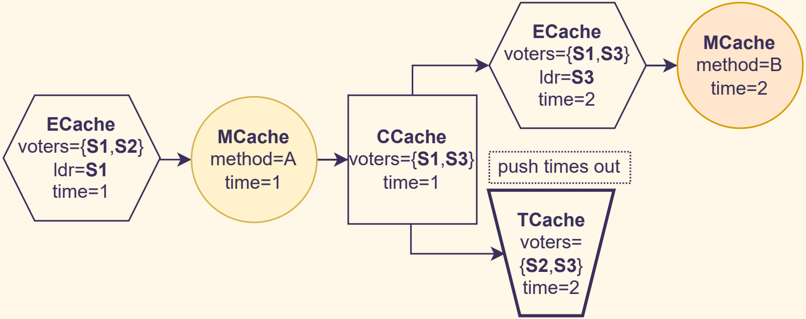
Created on failure (timeout)

TCache
voters={...}
time=t

Liveness in AdoB



Liveness in AdoB



ECache
voters={**S1**,**S2**}
ldr=**S1**
time=1

MCache
method=A
time=1

CCache
voters={**S1**,**S3**}
time=1

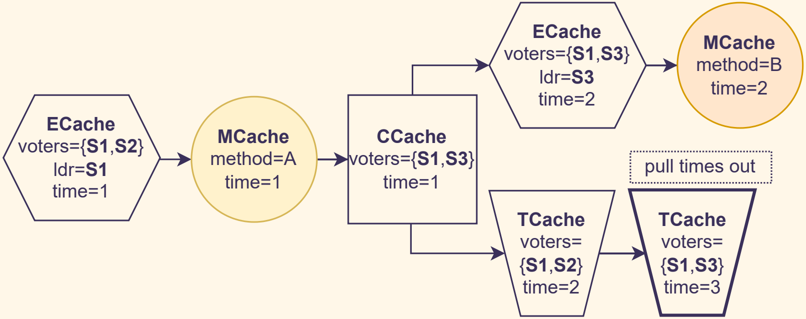
ECache
voters={**S1**,**S3**}
ldr=**S3**
time=2

MCache
method=B
time=2

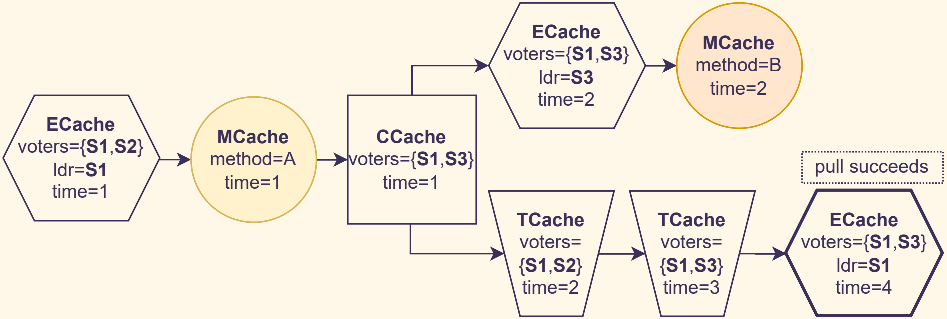
push times out

TCache
voters=
{**S2**,**S3**}
time=2

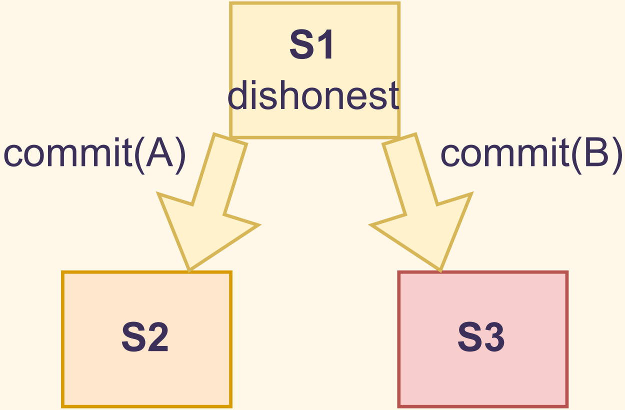
Liveness in AdoB



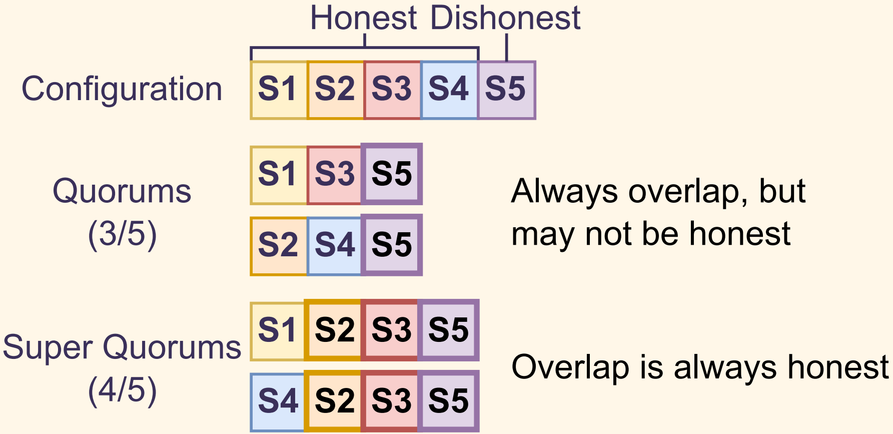
Liveness in AdoB



Byzantine Failures



Byzantine Failures



Honest Dishonest

Configuration

S1 S2 S3 S4 S5

Quorums (3/5)

S1 S3 S5

S2 S4 S5

Always overlap, but may not be honest

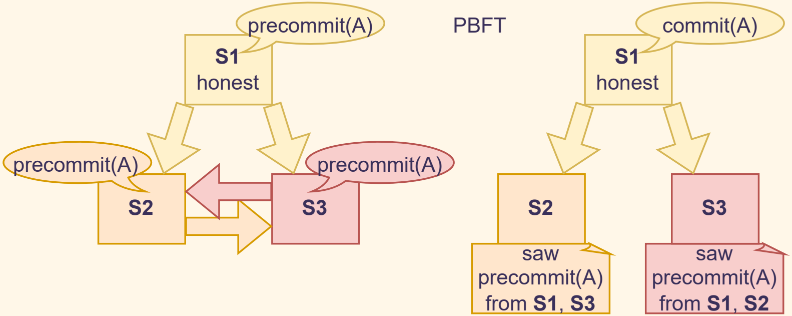
Super Quorums (4/5)

S1 S2 S3 S5

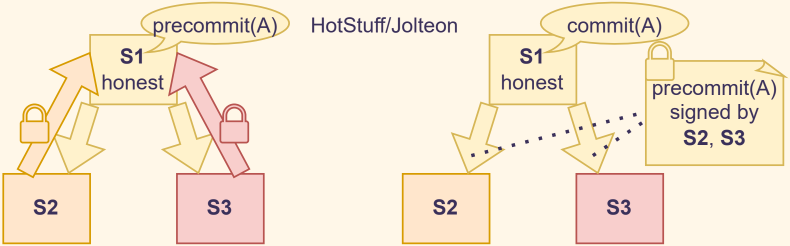
S4 S2 S3 S5

Overlap is always honest

Byzantine Failures



Byzantine Failures



precommit(A)

S1
honest

S2

S3

HotStuff/Jolteon

commit(A)

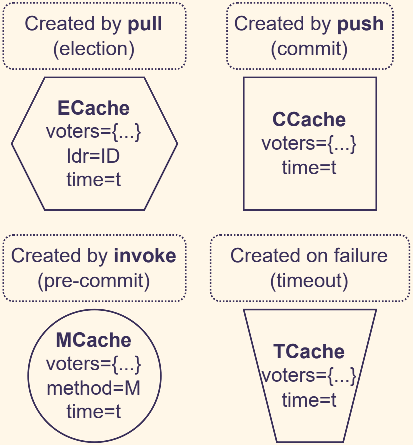
S1
honest

precommit(A)
signed by
S2, **S3**

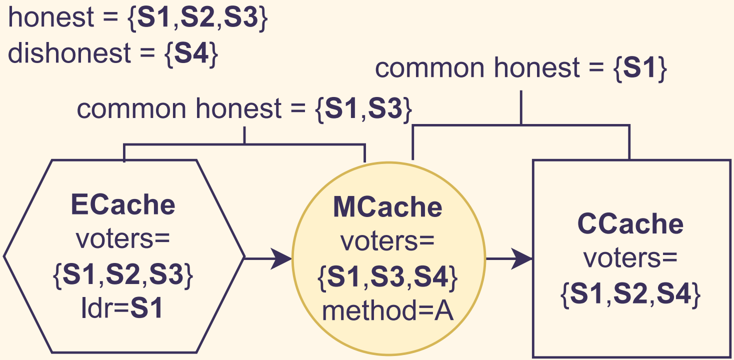
S2

S3

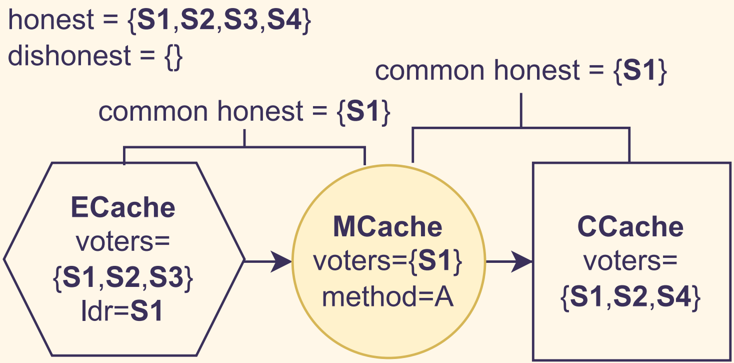
Byzantine Failures in AdoB



Generalizing Benign and Byzantine Failures



Generalizing Benign and Byzantine Failures



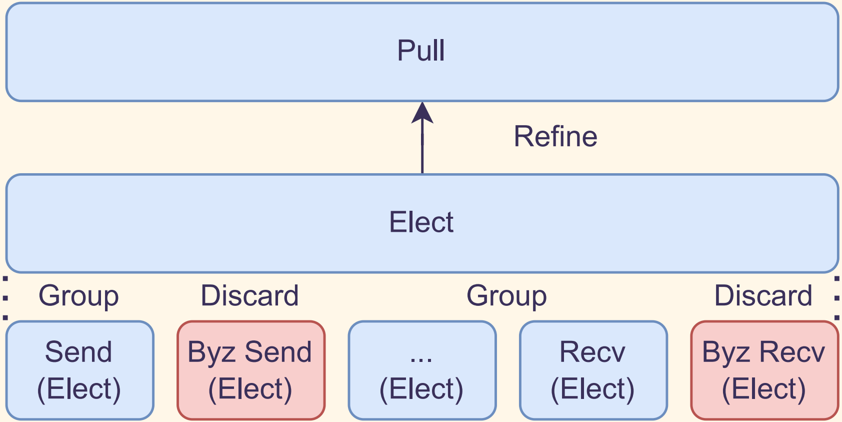
Generalizing Benign and Byzantine Failures

Failure Model	**Required Number of Votes**		
	pull	invoke	push
Benign	Quorum	Only leader	Quorum
Byzantine	Super Quorum	Super Quorum	Super Quorum
Generalized	Super Quorum	MQuorum	Super Quorum

Definition

Two quorums have a common voter (e.g., > 1/2 of configuration).
Super quorums have a common honest voter (e.g., > 2/3 of configuration).
An MQuorum and super quorum with the same leader have a common honest voter.

Refinement


Pull
Refine
Elect
Group
Discard
Group
Discard
Send (Elect)
Byz Send (Elect)
... (Elect)
Recv (Elect)
Byz Recv (Elect)

Proof Effort

	Proof LOC (Coq)	**Proof Time**
Safety Proof	∼3k	2 person-weeks
Liveness Proof	∼3k	2 person-weeks
Refinement Proof	∼4k	6 person-weeks

Roadmap

- Motivation
- ADO Overview
- Case Study: Advert
- Case Study: Adore
- Case Study: AdoB
- **Conclusions**
  - Summary of results.
  - Future work.

Summary

It facilitates formal verification by hiding network-level details behind a global tree-based state representation and atomic interface.

- ADO model: novel protocol-level abstraction for consensus.
- Atomic tree-based representation of replicated state.
- Exposes partial failures to distributed applications (Advert).
- Enables safety and liveness reasoning (Adore, AdoB).
- Correctly models a wide range of consensus protocols both benign (Advert, Adore) and byzantine (AdoB).
- Supports practical extensions like reconfiguration (Adore).

Future Work

- Automate refinement.
  - Verdi verified system transformers (PLDI '15).
  - CSPEC (OSDI '18), pretend synchrony (POPL '19), inductive sequentialization (PLDI '20).
- Generate code from ADO specification.
  - DeepSEA (OOPSLA '19).
- Expand beyond consensus.
  - Conflict-free replicated data types.
  - Causal consistency.