Do we have a quorum?

Quorum Systems

Given a set U of servers, |U| = n:

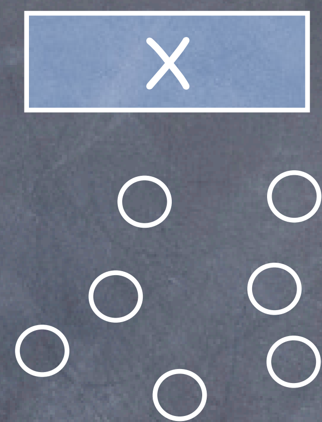
A quorum system is a set $\mathcal{Q} \subseteq 2^U$

such that

$$\forall Q_1, Q_2 \in \mathcal{Q} : Q_1 \cap Q_2 \neq \emptyset$$

Each Q in $\mathcal{Q}$ is a quorum

How quorum systems work: A read/write shared register

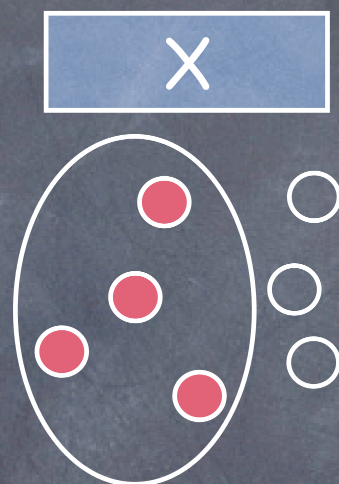


store at each server
a (v,ts) pair

Write(x,d)

- Ask servers in some Q for ts
- Set ts_c > max({ts} $\cup$ any previous ts_c)
- Update some Q′ with (d,ts_c)

How quorum systems work: A read/write shared register



store at each server
a (v,ts) pair

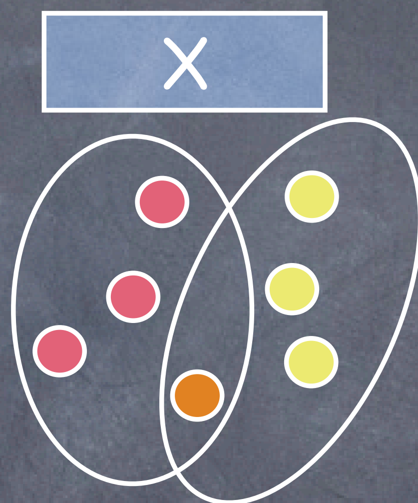
Write(x,d)

- Ask servers in some Q for ts
- Set $ts_c > \max(\{ts\} \cup$ any previous $ts_c)$
- Update some Q′ with (d, ts_c)

Read(x)

- Ask servers in some Q for (v,ts)
- Select most recent (v,ts)

How quorum systems work: A read/write shared register



store at each server
a (v,ts) pair

Write(x,d)

- Ask servers in some Q for ts
- Set ts_c > max({ts} $\cup$ any previous ts_c)
- Update some Q′ with (d,ts_c)

Read(x)

- Ask servers in some Q for (v,ts)
- Select most recent (v,ts)

What semantics?

- Safe:

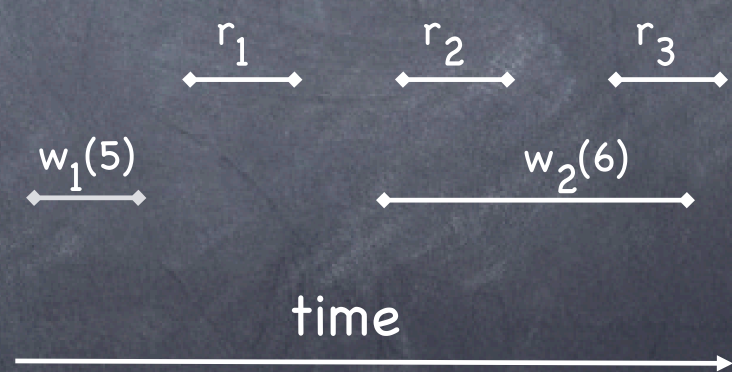
A read not concurrent with any write returns the most recently written value

- Regular:

Safe + a read that overlaps with a write obtains either the old or the new value

- Atomic:

Reads and writes are totally ordered so that values returned by reads are the same as if the operations had been performed with no overlapping



What semantics?

- Safe:

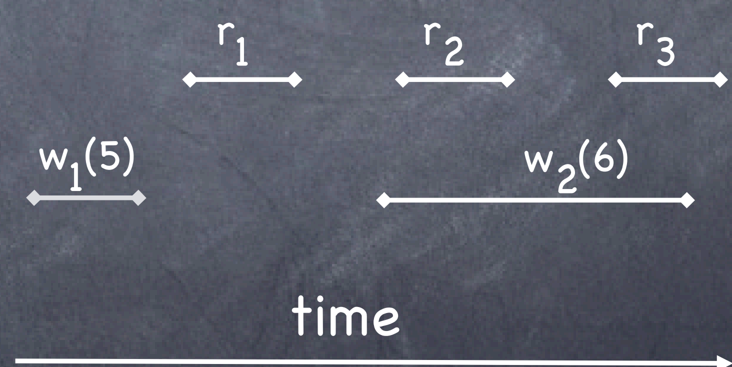
  A read not concurrent with any write returns the most recently written value

- Regular:

  Safe + a read that overlaps with a write obtains either the old or the new value

- Atomic:

  Reads and writes are totally ordered so that values returned by reads are the same as if the operations had been performed with no overlapping



What semantics?

- Safe:

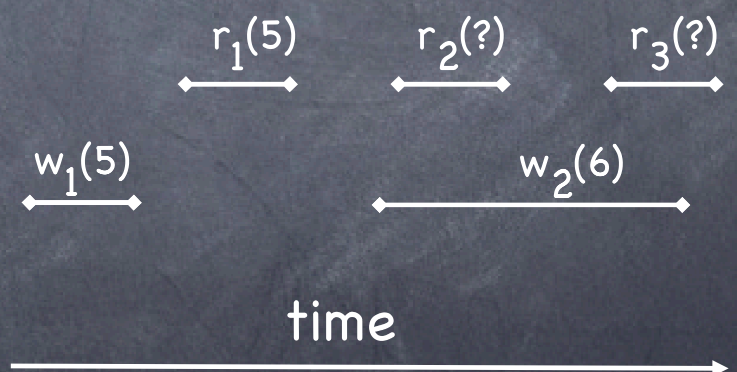
  A read not concurrent with any write returns the most recently written value

- Regular:

  Safe + a read that overlaps with a write obtains either the old or the new value

- Atomic:

  Reads and writes are totally ordered so that values returned by reads are the same as if the operations had been performed with no overlapping



What semantics?

- Safe:

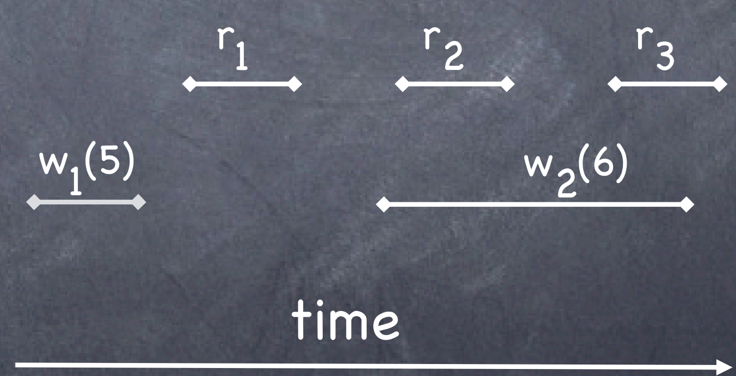
  A read not concurrent with any write returns the most recently written value

- Regular:

  Safe + a read that overlaps with a write obtains either the old or the new value

- Atomic:

  Reads and writes are totally ordered so that values returned by reads are the same as if the operations had been performed with no overlapping



What semantics?

- Safe:

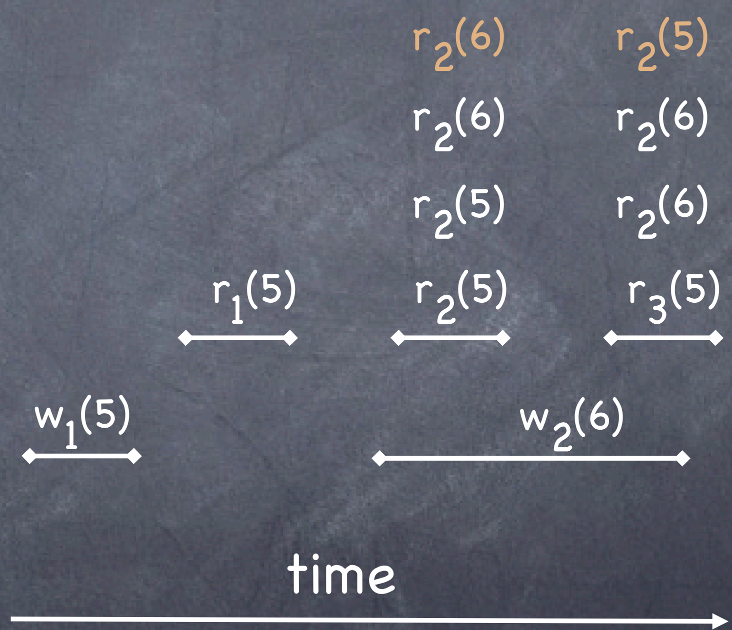
  A read not concurrent with any write returns the most recently written value

- Regular:

  Safe + a read that overlaps with a write obtains either the old or the new value

- Atomic:

  Reads and writes are totally ordered so that values returned by reads are the same as if the operations had been performed with no overlapping



What semantics?

- Safe:

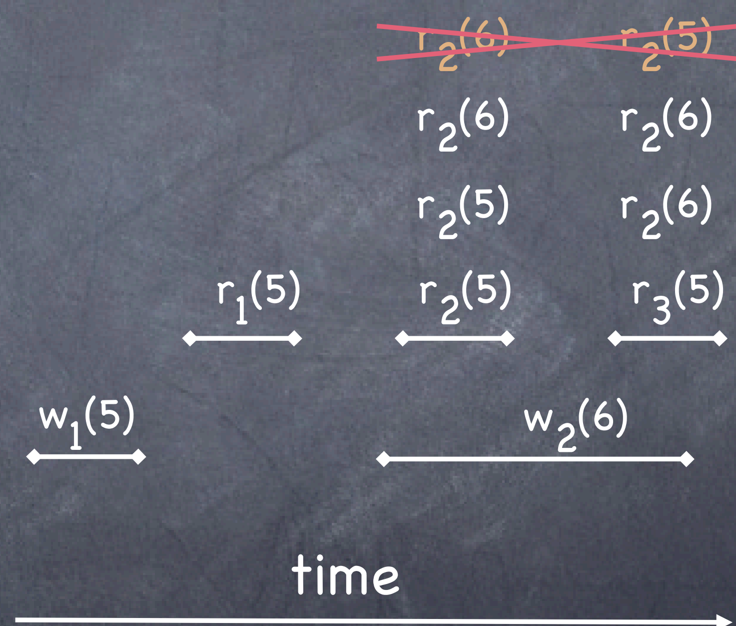
A read not concurrent with any write returns the most recently written value

- Regular:

Safe + a read that overlaps with a write obtains either the old or the new value

- Atomic:

Reads and writes are totally ordered so that values returned by reads are the same as if the operations had been performed with no overlapping



System Model

- Universe U of servers, |U| = n
- Byzantine faulty servers
  - modeled as a non-empty fail-prone system $\mathcal{B} \subseteq 2^U$
  - no $B \in \mathcal{B}$ is contained in another
  - some $B \in \mathcal{B}$ contains all faulty servers
- Clients are correct (can be weakened)
- Point-to-point authenticated and reliable channels

A correct process q receives a message from another correct process p if and only if p sent it

Masking Quorum System

[Malkhi and Reiter, 1998]

A quorum system $\mathcal{Q}$ is a masking quorum system for a fail-prone system $\mathcal{B}$ if the following properties hold:

M-Consistency

$\forall Q_1, Q_2 \in \mathcal{Q}\ \forall B_1, B_2 \in \mathcal{B} : (Q_1 \cap Q_2) \setminus B_1 \not\subseteq B_2$

M-Availability

$\forall B \in \mathcal{B}\ \exists Q \in \mathcal{Q} : B \cap Q = \emptyset$

Dissemination Quorum System

A masking quorum system for
self-verifying data
client can detect modification by faulty server

D-Consistency

$$\forall Q_1, Q_2 \in \mathcal{Q}\ \forall B \in \mathcal{B} : (Q_1 \cap Q_2) \not\subseteq B$$

D-Availability

$$\forall B \in \mathcal{B}\ \exists Q \in \mathcal{Q} : B \cap Q = \emptyset$$

f-threshold Masking Quorum Systems

$$\mathcal{B} = \{B \subseteq U : |B| = f\}$$

M-Consistency

$$\forall Q_1, Q_2 \in \mathcal{Q} : |Q_1 \cap Q_2| \geq 2f + 1$$

M-Availability

$$|Q| \leq n - f$$

$$\mathcal{Q}$$

$$\mathcal{Q} = \left\{ Q \subseteq U : |Q| = \left\lceil \frac{n + 2f + 1}{2} \right\rceil \right\}$$

$$n$$

$$n \geq 4f + 1$$

D-Consistency

$$\forall Q_1, Q_2 \in \mathcal{Q} : |Q_1 \cap Q_2| \geq f + 1$$

D-Availability

$$|Q| \leq n - f$$

$$\mathcal{Q}$$

$$\mathcal{Q} = \left\{ Q \subseteq U : |Q| = \left\lceil \frac{n + f + 1}{2} \right\rceil \right\}$$

$$n$$

$$n \geq 3f + 1$$

A safe read/write protocol

Client c executes:

Write(d)

→ Ask all servers for their current timestamp t

← Wait for answer from |Q| different servers

Set ts_c > max({t} $\cup$ any previous ts_c)

→ Send (d,ts_c) to all servers

← Wait for |Q| acknowledgments

Read()

→ Ask all servers for latest value/timestamp pair

← Wait for answer from |Q| different servers

Select most recent (v,ts) ~~for which at least f+1 answers agree (if any)~~

verifiable

Reconfigurable quorums

Design a Byzantine data service that

- monitors environment
  - uses statistical techniques to estimate number of faulty servers
- adjusts its tolerance capabilities accordingly:
  - fault-tolerance threshold changes within [f_{min}... f_{max}] range
    - very efficient when no or few failures
    - can cope with new faults as they occur
  - does not require read/write operations to block
- provides strong semantics guarantees

Managing the threshold

- Keep threshold value in a variable T
- Refine assumption on failures:

  For any operation *o*, number of failures never exceeds f, the minimum of:

  a) value of T before *o*
  b) any value written to T concurrently with *o*.

- Which threshold value should we use to read T ?
  - Update T by writing to an announce set

    A set of servers whose intersection with every quorum (as defined by f in [f_{min}...f_{max}]) contains sufficiently many correct servers to allow client to determine T's value unambiguously.

The announce set

- Intersects all quorums in at least $2f_{max} + 1$ servers
-
- Conservative announce set size: $n - f_{max}$
-
- Hence: $\dfrac{n + 2f_{min} + 1}{2} + (n - f_{max}) - n \geq 2f_{max} + 1$

$$\boxed{n \geq 6f_{max} - 2f_{min} + 1}$$

Updating T

- Client c (with current threshold f) executes:

Write(d)

→ Ask all servers for their current timestamp t

← Wait for answer from |Q| different servers

Set $ts_c > \max(\{t\} \cup \text{any previous } ts_c)$

→ Send (d, ts_c) to all servers

← Wait for |Q| acknowledgements

Read()

→ Ask all servers for latest value/timestamp pair

← Wait for answer from |Q| different servers

Select most recent (v,ts) for which at least f + 1 answers agree (if any)

Updating T

- Client c (with current threshold f) executes:

Write(d)

→ Ask all servers for their current timestamp t

← Wait for answer from ~~|Q| different servers~~ an announce set

Set $ts_c > \max(\{t\} \cup \text{any previous } ts_c)$

→ Send (d, ts_c) to all servers

← Wait for ~~|Q|~~ acknowledgments from an announce set

Read()

→ Ask all servers for latest value/timestamp pair

← Wait for answer from |Q| different servers

Select most recent (v,ts) for which at least f + 1 answers agree (if any)

Updating T

- Client c (with current threshold f) executes:

Write(d)

→ Ask all servers for their current timestamp t

← Wait for answer from ~~|Q| different servers~~ an announce set

Set $ts_c > \max(\{t\} \cup$ any previous $ts_c)$

→ Send (d, ts_c) to all servers

← Wait for ~~|Q|~~ acknowledgments from an announce set

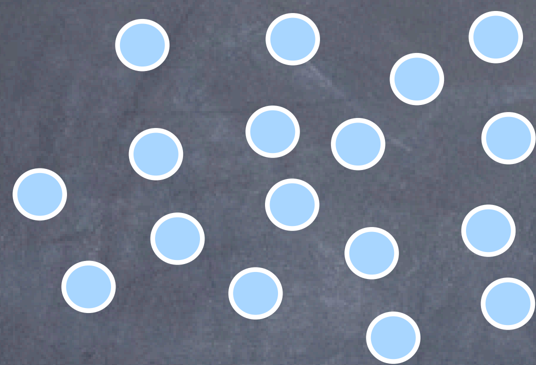
Read()

→ Ask all servers for latest value/timestamp pair

← Wait for answer from $|Q_{min}|$ different servers

Select most recent (v,ts) for which at least $f_{max} + 1$ answers agree (if any)

A problem



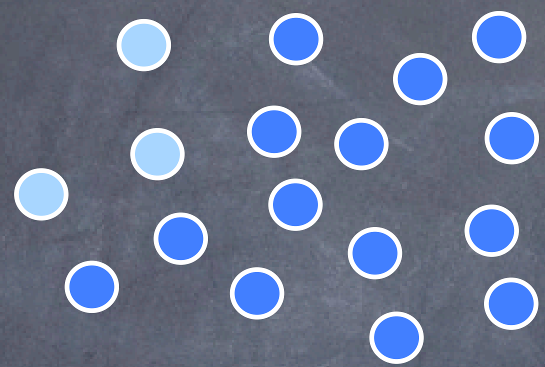
$f_{min} = 1 \quad f_{max} = 3$

$n = 17 \quad Q_{min} = 10$

announce set = 14

Initially, T = 1

A problem



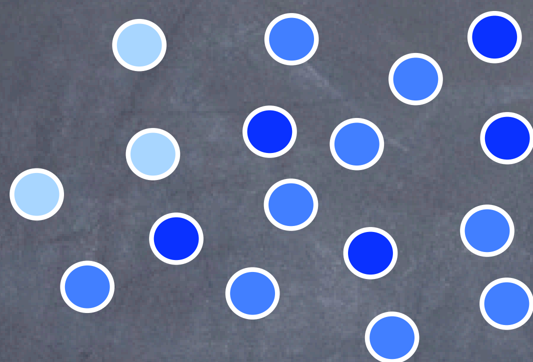
$f_{min} = 1 \quad f_{max} = 3$

$n = 17 \quad Q_{min} = 10$

announce set = 14

Threshold write: T = 2

A problem



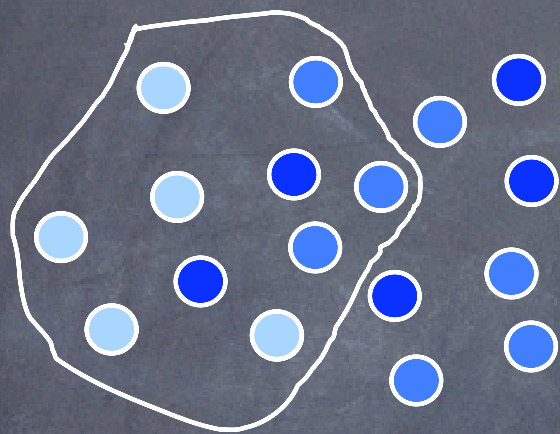
$f_{min} = 1 \quad f_{max} = 3$

$n = 17 \quad Q_{min} = 10$

announce set = 14

While a client is performing a threshold write to set T = 3...

A problem



$f_{min} = 1 \quad f_{max} = 3$

$n = 17 \quad Q_{min} = 10$

announce set = 14

...another client tries to read T

Countermanding

(v,ts) is countermanded if at least $f_{max}+1$ servers return a timestamp greater than ts

Write(f)

→ Ask all servers for their current timestamp t

← Wait for answer from an announce set

Set $ts_c > \max(\{t\} \cup$ any previous $ts_c)$

→ Send (d, ts_c) to all servers

← Wait for acknowledgements from an announce set

Read()

→ Ask all servers for latest value/timestamp pair

← Wait for answer from $|Q_{min}|$ different servers

Select most recent $\underbrace{(v,ts)}_{\text{not countermanded}}$ for which at least $f_{max}+1$ answers agree (if any)

Minimizing quorum size

Who cares? Machines are cheap...

But achieving independent failures is expensive!

- Independently failing hardware
- Independently failing software
  - Independent implementations of server
  - Independent implementation of underlying OS
  - Independent versions to maintain

A simple observation

- Client c (with current threshold f) executes:

Write(d)

→ Ask all servers for their current timestamp t

← Wait for answer from |Q| different servers

Set $ts_c > \max(\{t\} \cup$ any previous $ts_c)$

→ Send (d, ts_c) to all servers

← ~~Wait for |Q| acknowledgements~~

Read()

→ Ask all servers for latest value/timestamp pair

← Wait for answer from |Q| different servers

Select most recent (v,ts) for which at least f + 1 answers agree (if any)

(Asynchronous)
Authenticated
Reliable channels

A correct process q receives a message from another correct process p if and only if p sent it

A-Masking Quorum Systems

A quorum system Q is an a-masking quorum system for a fail-prone system B if the following properties hold for Q_r and Q_w:

AM-Consistency

$$\forall Q_r \in \mathcal{Q}_r \; \forall Q_w \in \mathcal{Q}_w \; \forall B_1, B_2 \in \mathcal{B}$$
$$(Q_r \cap Q_w) \setminus B_1 \not\subseteq B_2 :$$

AM-Availability

$$\forall B \in \mathcal{B} \; \exists Q_r \in \mathcal{Q}_r : B \cap Q_r = \emptyset$$

Tradeoffs

best known n	confirmable	non-confirmable
self-verifying	3f+1	2f+1
generic	4f+1	3f+1

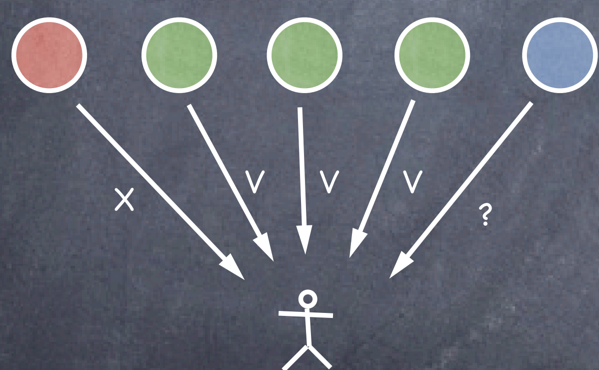
Tradeoffs

best known n	confirmable	non-confirmable
self-verifying and generic	3f+1	2f+1

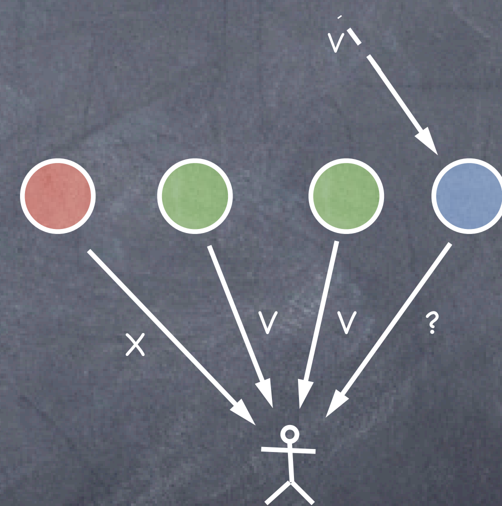
Lower bound: never two rows again!

The intuition

Trade replication in space for replication in time



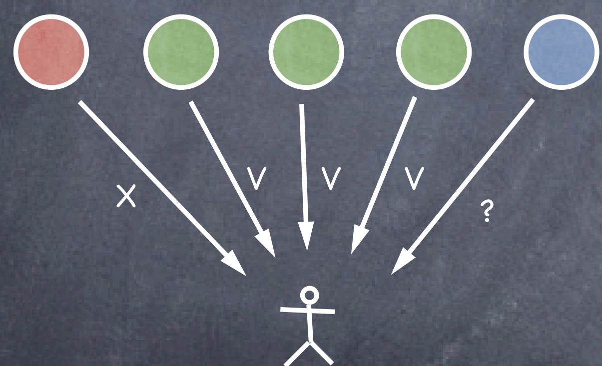
Traditional: 4f+1 servers



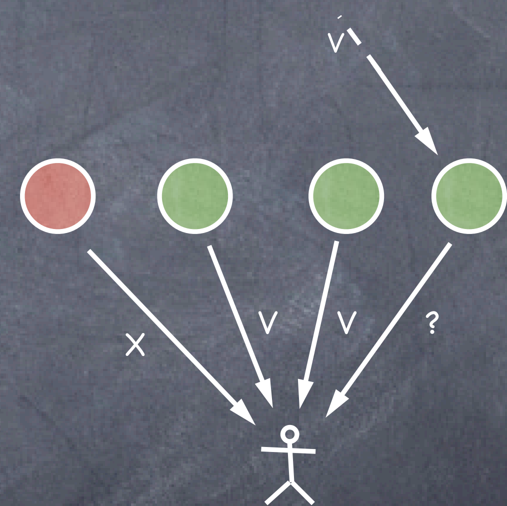
Now: 3f+1 servers

The intuition

Trade replication in space for replication in time



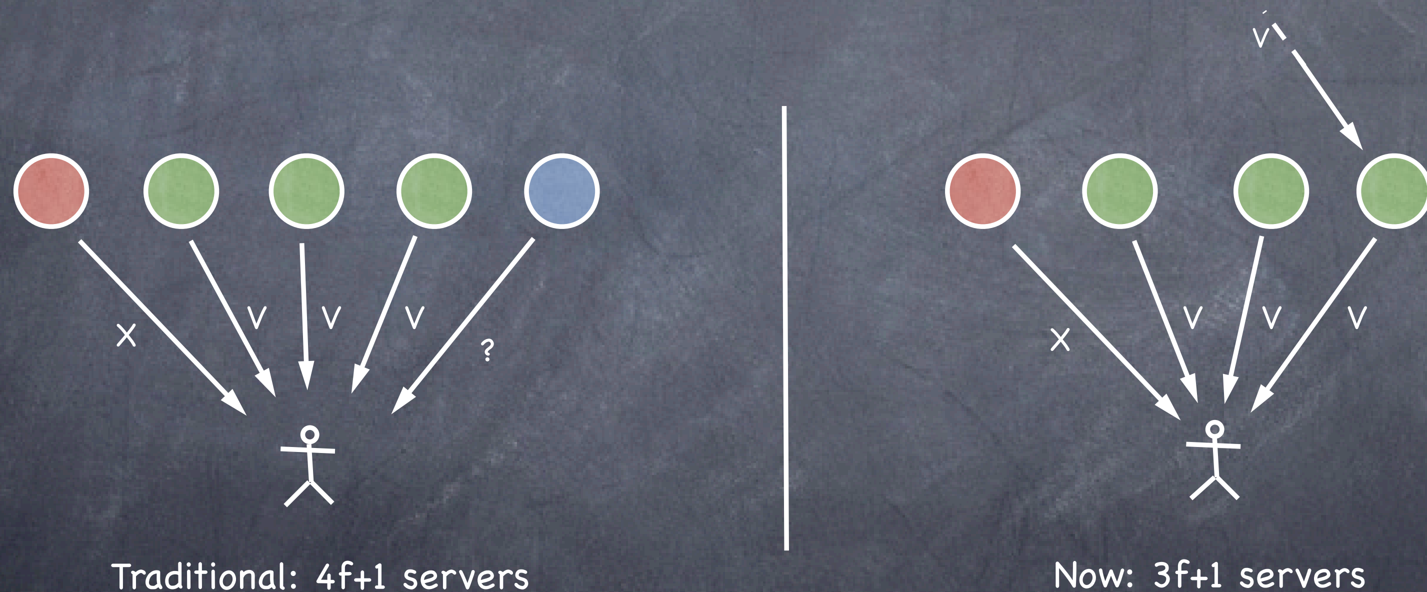
Traditional: 4f+1 servers



Now: 3f+1 servers

The intuition

Trade replication in space for replication in time



Both cases: wait until 4th server receives write

The protocol

$\lvert Q_w \rvert$	$\lvert Q_r \rvert$
$\left\lceil \frac{n+f+1}{2} \right\rceil$	$\left\lceil \frac{n+3f+1}{2} \right\rceil$

Client c executes:

Write(d)

→ Ask all servers for their current timestamp t

← Wait for answer from $\lvert Q_w \rvert$ different servers

Set $ts_c > \max(\{t\} \cup \text{any previous } ts_c)$

→ Send (d,ts) to all servers

← Wait for $\lvert Q_w \rvert$ acknowledgments

Read()

→ send READ-START to server set Q_r

repeat

← receive a reply (D, ts) from s in Q_r

set answer[s,ts] := D

until some A in answer[ ][ ] is vouched for by $\lvert Q_w \rvert$ servers

→ send READ-STOP to Q_r

return A

The Slim-Fast version

1. Whenever c gets first message from a server, it computes

   T = {largest f+1 timestamps from distinct servers}

2. (D,ts) from answer[s][] is discarded unless either

   a) $ts \in T$ or

   b) ts is the latest timestamp received from s

The Goodies

Theorem

The protocol guarantees atomic semantics

Proof: Safety

Lemma 1: If it is live, it is atomic

a) After write of ts_1, no read returns earlier ts

- Suppose write for ts_1 has completed
- $\left\lceil \frac{n+f+1}{2} \right\rceil$ servers acked the write
- At least $\left\lceil \frac{n-f+1}{2} \right\rceil$ are correct
- Remaining $\left\lceil \frac{n+f-1}{2} \right\rceil$ servers < $|Q_w|$

b) After c reads ts_1, no later read returns earlier ts

- c reads $ts_1 \longrightarrow \left\lceil \frac{n+f+1}{2} \right\rceil$ servers say ts_1
- At least $\left\lceil \frac{n-f+1}{2} \right\rceil$ are correct
- Remaining $\left\lceil \frac{n+f-1}{2} \right\rceil$ servers < $|Q_w|$
- Any read that starts after ts_1 returns $ts \geq ts_1$

Proof: Liveness

Lemma 2: Every operation eventually terminates

WRITE: trivial, because only waits for $|Q_w| < n - f$

READ:

- Consider T after c gets first message from last server.
- Let t_{max} be the largest timestamp from a correct server in T.
- A client never removes t_{max} from its answers[s][], for a correct s
- Eventually, all correct servers see a write with ts = t_{max} and echo client
- Since $|Q_r| = \left\lceil \frac{n+3f+1}{2} \right\rceil$, $|Q_w| \leq |Q_r| - f$ and the read terminates