

More severe failures

What if the data repositories can be arbitrarily faulty?

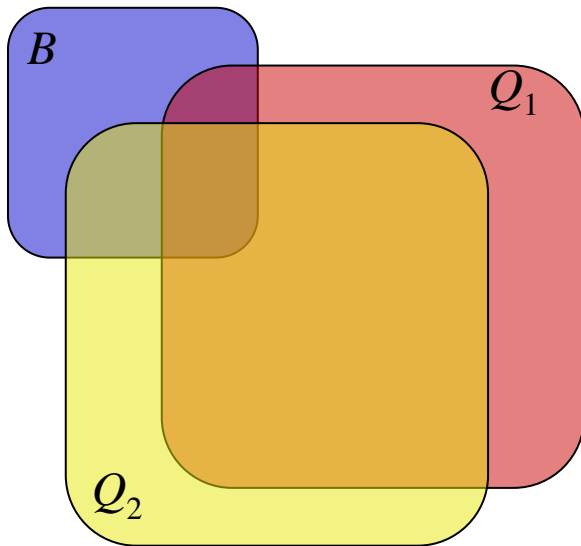
Byzantine Quorum Systems

- β is the set of fail prone sets (*fail prone system*).
A maximal set of faulty processes in some run
- $S \subseteq 2^U$ is a set of quorums (a coterie).
- Clients behave correctly (for now).

More severe failures

M-Consistency: a client can identify results being from at least one nonfaulty server.

$$\forall Q_1, Q_2 \in S, \forall B_1, B_2 \in \beta : \neg(((Q_1 \cap Q_2) - B_1) \subseteq B_2)$$



M-Availability

$$\forall B \in \beta : \exists Q \in S : B \cap Q = \{\}$$

Masking Quorum Systems

Write

- Client c queries the servers to obtain a set of timestamps $A = \{\langle t_u \rangle\}$ for $u \in Q$ and $Q \in S$.
- c chooses a timestamp t greater than any timestamp value in A and greater than any timestamp c has previously chosen (from some space T_c).
- c sends the update to the servers until it has received an acknowledgment for this update from every server in some quorum $Q' \in S$.

... a server s updates its copies of t and s only if the client's timestamp is greater than t . In any case, s always acknowledges the update.

Masking Quorum Systems

Read

- Client c queries the servers to obtain a set of timestamp/value pairs $A = \{\langle t_u, v_u \rangle\}$ for $u \in Q$ and $Q \in S$.

- c computes

$$A' = \{\langle t, v \rangle : \exists B^+ \subseteq Q : \bigwedge \forall B \in \beta : \neg(B^+ \subseteq B) \\ \bigwedge \forall u \in B^+ : t_u = t \wedge v_u = v\}$$

- c chooses a pair $\langle t, v \rangle$ in A' with the highest t and returns v . If A' is empty, then the client returns \perp .

This implements a *multi-reader, multi-writer safe* register.

Masking Quorum System

$U = \{s_1, s_2, s_3, s_4\}$, $S = \{\text{all subsets of } U \text{ of size } 3\}$,
 $\beta = \{\{s_1\}, \{s_2\}, \{s_3\}, \{s_4\}\}$.

$\langle 1, v_1 \rangle$

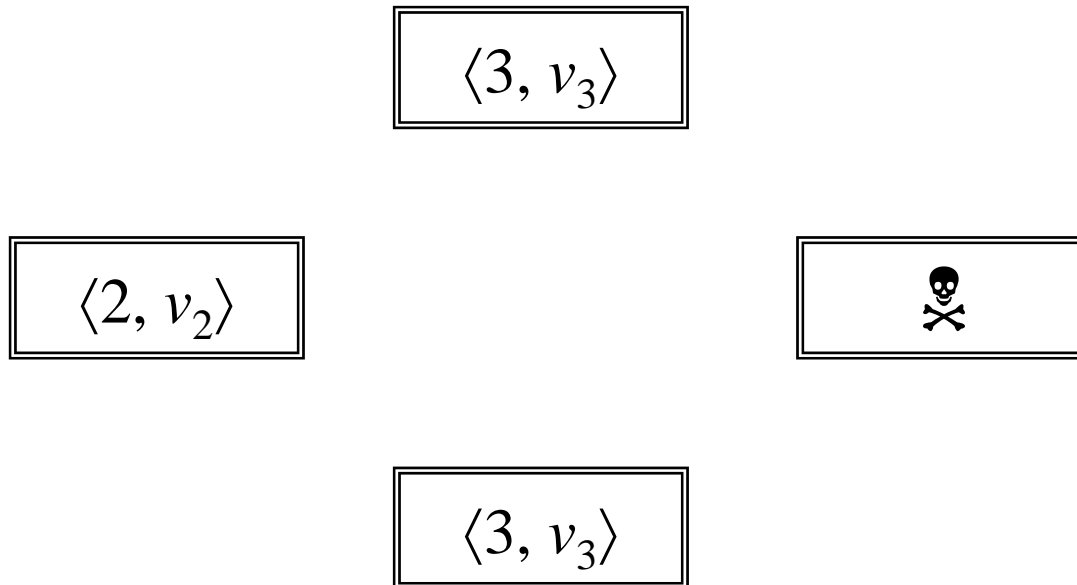
$\langle 2, v_2 \rangle$



$\langle 3, v_3 \rangle$

Masking Quorum System

$U = \{s_1, s_2, s_3, s_4\}$, $S = \{\text{all subsets of } U \text{ of size } 3\}$,
 $\beta = \{\{s_1\}, \{s_2\}, \{s_3\}, \{s_4\}\}$.

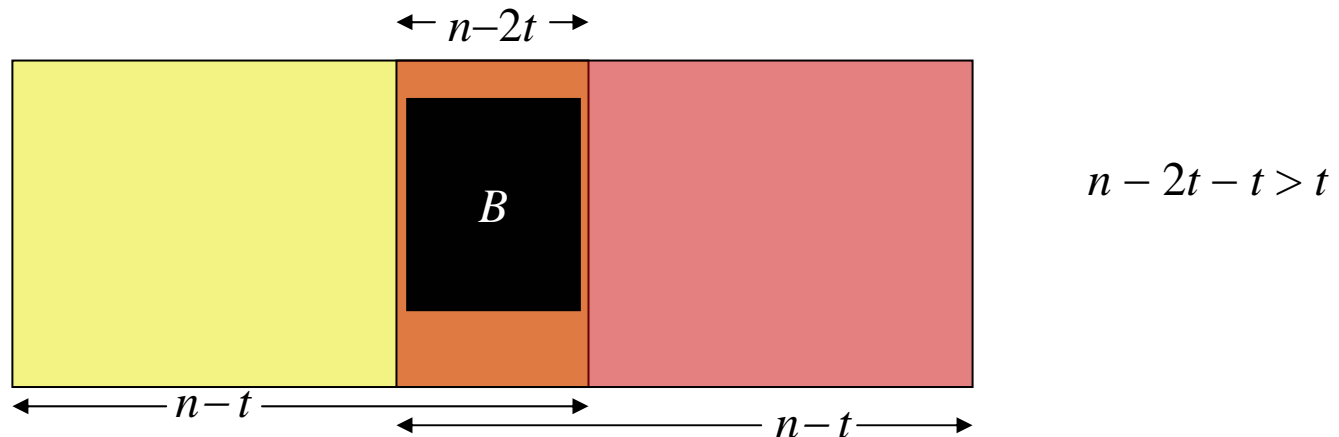


Masking Quorum Systems

This approach requires:

$$\forall Q_1, Q_2 \in S, \forall B_1, B_2 \in \beta : \neg(((Q_1 \cap Q_2) - B_1) \subseteq B_2)$$

If β is all subsets of U that have t elements, then this becomes the constraint $n > 4t$.



Dissemination Quorum Systems

If data is *self-verifying*, then all we need is:

D-Consistency: $\forall Q_1, Q_2 \in S, \forall B \in \beta : \neg(Q_1 \cap Q_2 \subseteq B)$

D-Availability: $\forall B \in \beta : \exists Q \in S : B \cap Q = \{\}$

... which reduces the replication requirement to
 $n > 3t$.

Oblivious Quorum Systems

Revealing β to the clients is often not a great idea...

- There may be no compact representation of b .
- Doing so reveals vulnerabilities which could be exploited by an attacker to guide an active attack against the system.

... instead, use a *voting approach* that does not require clients to know β .

Oblivious Quorum Systems

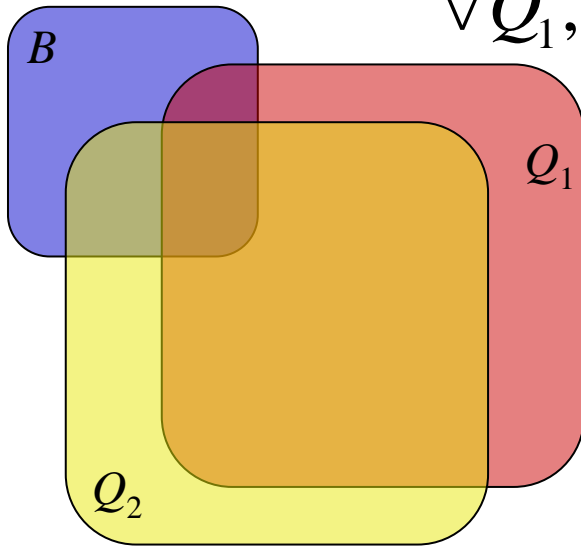
O-Consistency1: avoid suppressing writes.

$$\forall Q_1, Q_2 \in S, \forall B \in \beta : |(Q_1 \cap Q_2) - B| \geq |(Q_2 \cap B) \cup (Q_2 - Q_1)|$$

O-Consistency2: majority value correct.

$$\forall Q_1, Q_2 \in S, \forall B \in \beta : |(Q_1 \cap Q_2) - B| > |Q_2 \cap B|$$

need not be >.



O-Availability

$$\forall B \in \beta : \exists Q \in S : B \cap Q = \{\}$$

Oblivious Quorum Systems

Read

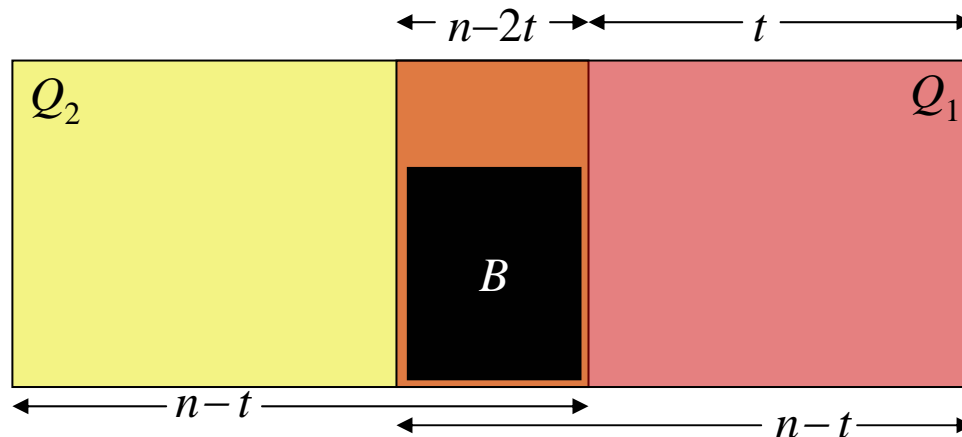
- Client c queries the servers to obtain a set of timestamp/value pairs $A = \{\langle t_u, v_u \rangle\}$ for $u \in Q$ and $Q \in S$.
- Client c chooses the pair $\langle t, v \rangle$ that appears most often in A , and if there are multiple such pairs, the one with the highest timestamp. Client c then returns the associated v .

Oblivious Quorum Systems

$$\forall Q_1, Q_2 \in S, \forall B \in \beta : |(Q_1 \cap Q_2) - B| \geq |(Q_2 \cap B) \cup (Q_2 - Q_1)|$$

$$\begin{aligned} n - 3t &= |Q_1 \cap Q_2| - |B| = |(Q_1 \cap Q_2) - B| \\ &\geq |(Q_1 - Q_2) \cup (Q_1 \cap B)| = |(Q_1 - Q_2) \cup B| \\ &\geq t + |B| = 2t \end{aligned}$$

or $n > 5t$



Faulty Clients

Faulty clients might send different updates to different servers and may fail to contact a full quorum.

Write

1. Client c chooses a timestamp $t \in T_c$ greater than any value it has chosen before.
2. Client c chooses quorum Q and sends $\langle \mathbf{update}, Q, v, t \rangle$ to each server in Q .
3. If after a timeout, c has not received an ack from each server in Q , c repeats Steps 2 and 3.

Faulty Clients

Update

1. If server s receives $\langle \mathbf{update}, Q, v, t \rangle$ from a client c where $t \in T_c$ and s has not previously received $\langle \mathbf{update}, Q, v, t \rangle$ from c where either $t' = t$ and $v' = v$ or $t' > t$, then s sends $\langle \mathbf{echo}, Q, v, t \rangle$ to all in Q .
2. If s receives identical $\langle \mathbf{echo}, Q, v, t \rangle$ from all in Q , then it sends $\langle \mathbf{ready}, Q, v, t \rangle$ to all in Q .

Faulty Clients

Update

3. If s receives identical $\langle \mathbf{ready}, Q, v, t \rangle$ from B^+ : $\forall B \in \beta$:
 $\neg (B^+ \subseteq B)$, then sends $\langle \mathbf{ready}, Q, v, t \rangle$ to all in Q if has not done so already.
4. If s receives identical $\langle \mathbf{ready}, Q, v, t \rangle$ from Q^- : $\exists B \in \beta$:
 $Q^- = Q - B$, then
 1. If t is greater than the timestamp s currently has then s updates its stored values of v and t .
 2. Regardless of whether s updates its v and t , it sends an ack to c where $t \in T_c$.

Faulty Clients

1. A correct server delivers $\langle v, t \rangle$ only if some correct server receives $\langle \mathbf{update}, Q, v, t \rangle$.
2. If a correct server delivers $\langle v, t \rangle$ and a correct server delivers $\langle v', t \rangle$, then $v = v'$.

... So, let W be the set of writes preceding the read.
From 2, the writes are well-defined.

By 1, no correct server has delivered a write not in W .
There is a serialization order imposed by the writes,
and so the register is safe.