

Byzantine

Fault

Tolerance - BFT

Last class:

- FAIL-STOP
- FAIL-RECOVER
- REALITY



Bugs or attacks don't always lead to crashes!

- If only they did

Goal: Avoid assumptions about the behavior of faults

nodes: - BYZANTINE FAULT TOLERANCE

Where: High-assurance (The Real Byzantine Generals - Dworkin et al '04)

- Submarines
- Space


Cryptocurrency \$\$\$

Supply chains, critical cloud services (soon)

Really, a cost - benefit tradeoff

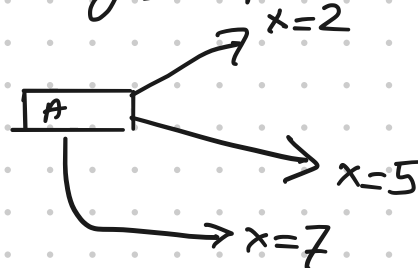
- BFT often slower, needs more computation, etc.
- So really about whether it is worth it to you!

Question: What can a faulty node do in the Byzantine model?

- Crash 
- Not send some messages (omission)
- Pretend to be a different node (impersonation)



- Send contradictory messages (equivocation)



- Corrupt internal state
- Implement a different protocol
- ...

Q: Which of these do fail-stop protocols already handle?

Q: Which can we detect?

Problems that are FREQUENTLY A FOCUS

- IMPERSONATION
 - Why

MAC, HMAC - Standard approach: Authenticate? Digital signatures?

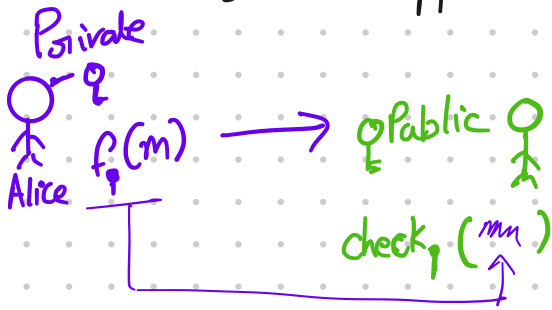
f, g I_p I_{priv} $f(m, I_p) = c \rightarrow g(c, I_{priv}) = m$

Emerging problems with the approach

- EQUIVOCATION

- Why?

- Standard approach: Authenticated channels /

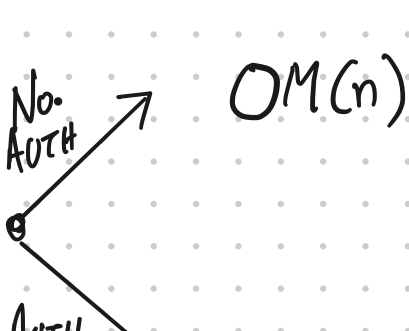


Transferable authentication
Signed messages +
Protocol to exchange messages

- Concerns: cost

- LIVENESS / TERMINATION

FROM LAMPORT



Q: From earlier, assumed model?

- Asynchronous

Auth CHANNEL \rightarrow SM(n)

- Partially synchronous
- Synchronous

Why?

OM(n): Without auth. need $N \geq 3f + 1$

Req.

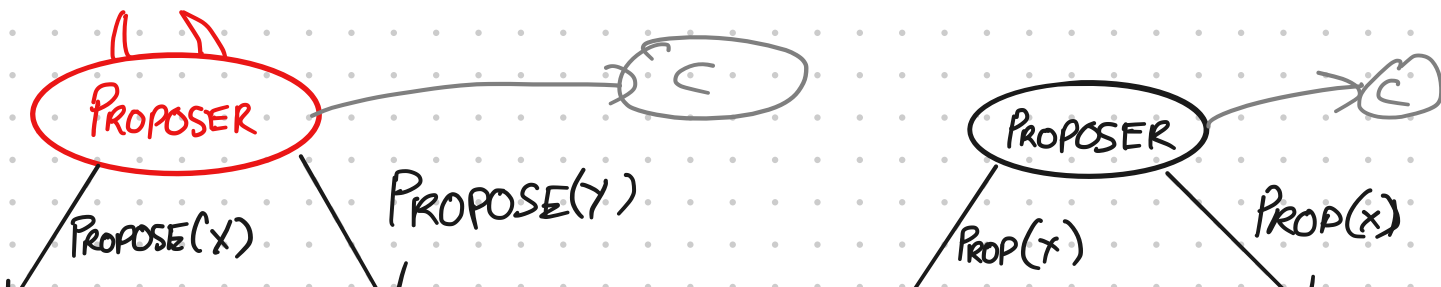
- ^{Weak} Agreement: All non-faulty nodes agree (IC1)
- Validity: Derived assuming LEADER (DESIGNATED PROPOSER)

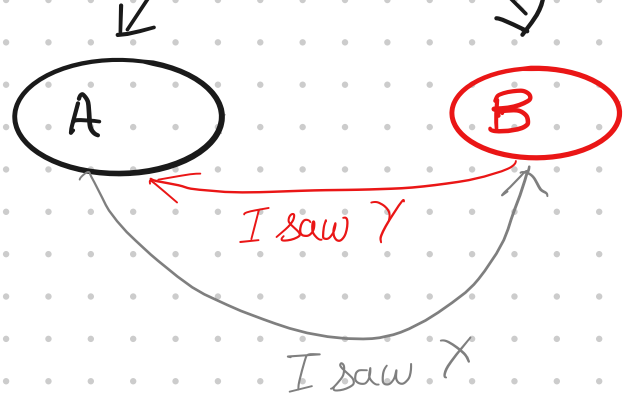
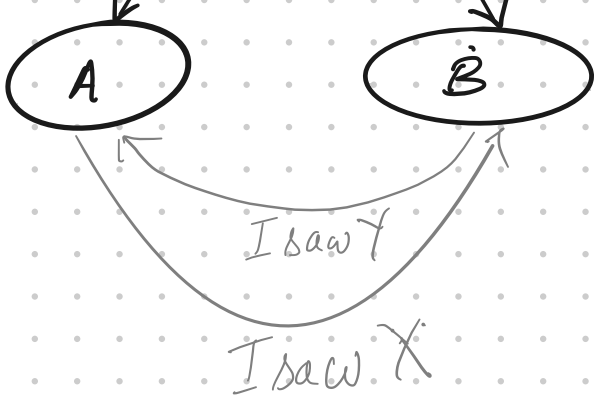
If proposer correct \rightarrow

non-faulty nodes decide proposed value

- Termination: Eventually decision is reached.

Why $N \geq 3f + 1$

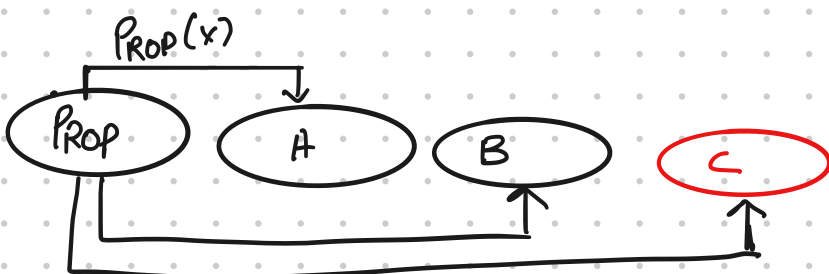




What should A do?

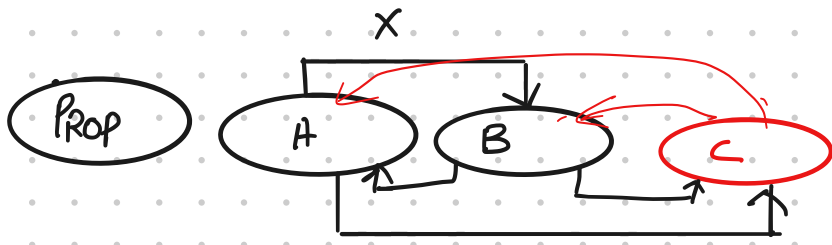
a. Why would authenticated channels help?

Problem also suggests path forward: use voting to figure out proposal



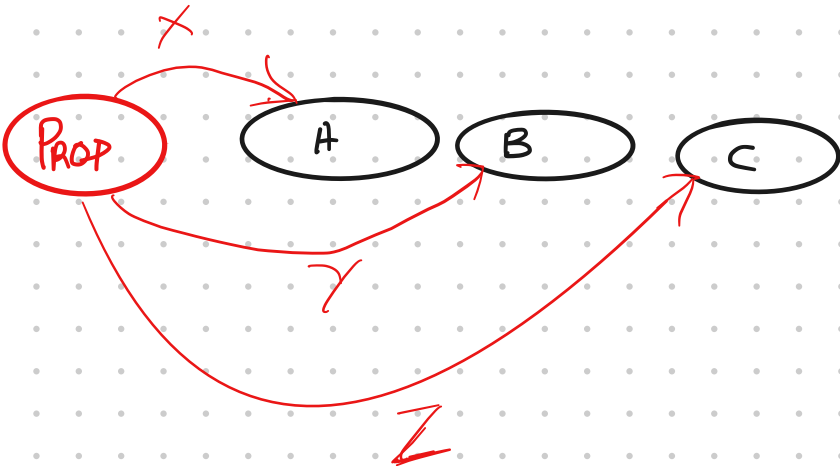
A

From	Value
Prop	X
B	
C	



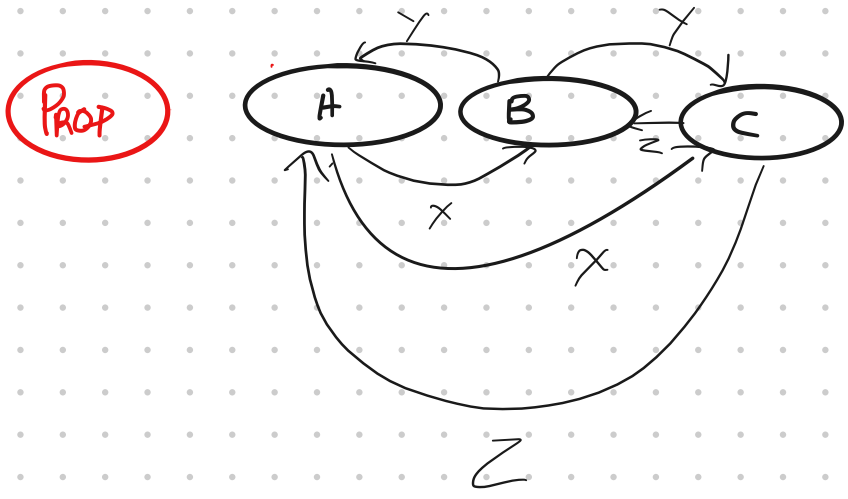
From	Value
Prop	X
B	X
C	?

} [X]



A

From	Value
Prop	X
B	
C	



From	Value
Prop	X
B	Y
C	Z

} Pick deterministically.

Trickier as we increase number of faulty servers but same principle

Synchrony is hard - what about partial synchrony

PBFT : Note, other protocols are available, this is just the popular one.

- Assumes authenticated channels
- Messages are broadcast
- Client counts responses to determine if op committed

↳ Why?

Similar to other protocols we saw : Quorum based. + W/Leader

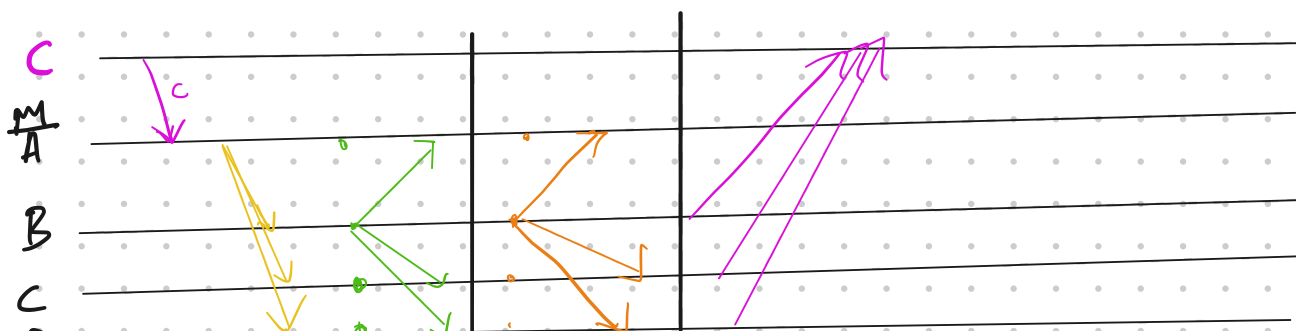
But quorum of

$2f+1$: A majority of nodes that make a decision are connect $f < f+1$



Need $\geq 3f+1$ nodes
↳ Why?

Leader : Designated proposer \leftrightarrow Decides what to propose



wait for $\geq 2f+1$

- PREPREPARE ($\langle c \rangle_\sigma$, $\langle \text{view (term)}, n, \text{hash}(\langle c \rangle_\sigma) \rangle_A$)
↳ signed by C. Why?
signed by A.

- PREPARE ($\langle \text{view}, n, \text{hash}(\langle c \rangle_\sigma), B \rangle_B$)
↳ Why?

wait for $2f+1$ prepare for a previously pre-prepared command.

→ $f+1$ correct nodes agree

$\langle \text{view}, n \rangle \rightarrow c$

- COMMIT ($\langle \text{view}, N, \text{hash}(\langle c \rangle_\sigma), B \rangle_B$)

wait for $2f+1$ commit command for prev. prepared command.

Client waits for $f+1$ responses

- Why $f+1$ not $2f+1$?

Liveness

- Leader is **faulty**?

Client triggers
view change

View change/leader election

Two requirements

① Faulty nodes cannot monopolize leadership
→ View ID determines leader

	A	B	C	D
Leader	\emptyset	1	2	3
In View	4	5	6	7

...

② Faulty nodes cannot trigger view change

- Huh?

- How?

View active until $2f+1$ nodes agree.

BFT TODAY

- Regaining interest

- Question: Can we reduce overheads?

- How? Add information — failure detector

- Trusted hardware

↳ TPM

→ Enclaves

→ ...

- Timing assumptions

- ???

