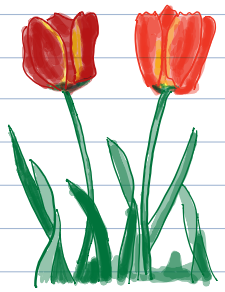
# Secure Distributed Systems

## CompSci 661 / 461





## This video



- Recovery from failure using replicas
- Coordination among replicas
- Correct operation despite malicious failure
  (Byzantine Failures)
- Lamport's Solution

# Types of failure

Did you read?

Pages 324-328

based on

| Type of failure | Description |
|---|---|
| Crash failure | A server halts, but is working correctly until it halts |
| Omission failure | A server fails to respond to incoming requests |
| *Receive omission* | A server fails to receive incoming messages |
| *Send omission* | A server fails to send messages |
| Timing failure | A server's response lies outside the specified time interval |
| Response failure | A server's response is incorrect |
| *Value failure* | The value of the response is wrong |
| *State transition failure* | The server deviates from the correct flow of control |
| Arbitrary failure | A server may produce arbitrary responses at arbitrary times |

**Figure 8-1.** Different types of failures.

**Byzantine Failures** are the worst kind of failure

- assumes the worst possible result from the one that messes with your alg. the most.
- even coordinating with other servers maliciously.
- assume they are not obviously malicious.

**Techniques for masking failures:** page 330

- Add information redundancy
  - parity bits
  - error correcting codes
- add physical redundancy
  - multiple servers
- time redundancy
  - repeat the request if no answer

# Masking failure with replication



First, we need a method of coordinating the servers.

Here are some methods.

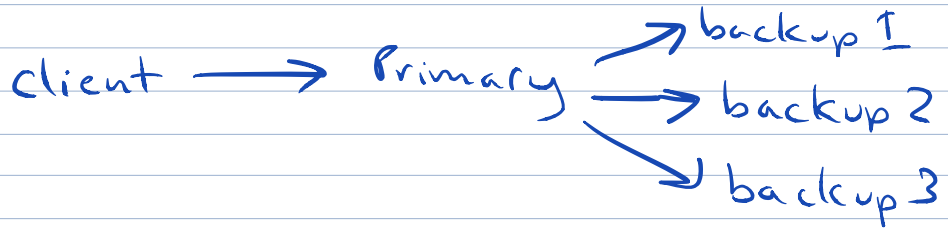
## ① Primary-based Protocols

page 308, § 7.5.2

- easier and simpler (good)
- single point of failure (bad)

remote-write protocol

- Any write to a data item x (e.g., such as a to a database table) is forwarded to the primary server by the client.
- Primary performs the write
- then it forwards the update to the backup servers
- backups send Acknowledgements upon success.
- When all backups have acked, primary acks.
- Everything blocked until primary acks to the client.




## ② Active Replication (§7.5.3)

- complex (bad)
- no single point of failure (good)

- a write is sent to all N replicas
- events are ordered (e.g. using Lamport Clocks)
- to read, ask any single replica
- does not scale well to large values of N.

## ③ Quorum-based protocols.

Like ②, replicated writes

- Say we have N servers
- We enforce a rule that to write an update to item X, a client must first contact at least half plus 1 of the N servers: $N/2 + 1$

(a strict majority)

- Once agreement is had, data item x is considered updated and a new revision number is issued.
- To Read: at least $N/2+1$ of the servers must be contacted to determine the current version.

(ask each for version number, and accept the max)

More generally, we require that if

$N$ is the number of replicas.

$N_R$ is the number we read from.

$N_W$ is the number we write to.

then:

① $N_R + N_W > N$

avoids errors from reading after write

② $N_W > N/2$

avoids write-after-write errors.

## Example 1:

$N_R = 3$

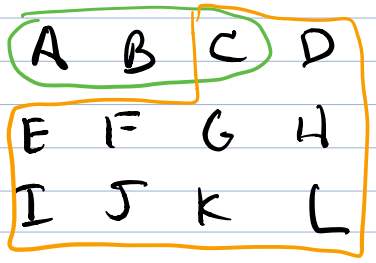
$N_W = 10$

$N = 12$ servers

$3 + 10 > 12$

and

$10 > \frac{12}{2} = 6$




There is no way to pick 3 servers to read from such that one doesn't have the latest version

Neither rule is violated.

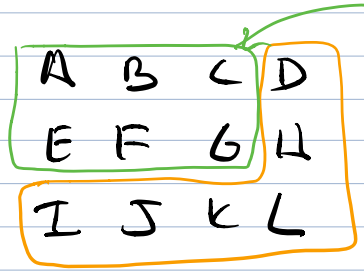
## Example 2:

$N_R = 7$

$N_W = 6$

$7 + 6 > 12$

$6 \not> 12/2$ violates rule (2)




write set 1
write set 2
} each give the same version number

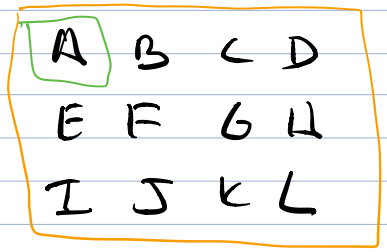
This is a write-after-write error

## Example 3:

$N_R = 1$

$N_W = 12$

Read once - Write All



## (4) Blockchains (forcing into this paradigm...)

- Leader Election for deciding Primary-based Replication
  - (1) cryptographic puzzle decides winner
  - (2) honest nodes mark winner by adding to the chain

arguably $N = \frac{N}{2} + 1$ ← # of honest nodes required

$N_R = N/2 + 1$ → must be an honest node

$N_W = N/2 + 1$

Not really a read-once system.

Servers is really hash rate.

---

Given the coordination method,

How many failures can we survive?

We say something is **K-fault tolerant**

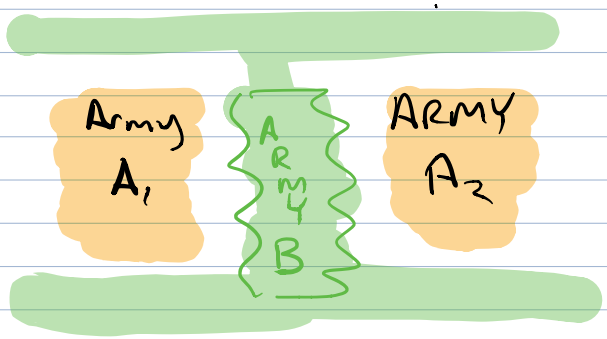
if it can survive faults in k of its components and still operate correctly.

To survive K silent failures
- we need 1 out of K+1 servers to be non-faulty.

To survive K Byzantine Failures
- we need 2K+1 out of 3K+1 to be non-faulty.

Note that $\frac{2K+1}{3K+1} > \frac{2}{3}$

# The Two Army Problem
(has no solution)




Army A is separated by ArmyB

IF Both $A_1$ and $A_2$ attack at the same time: Victory!

Otherwise they fail.

- Messages between $A_1$ and $A_2$ require sneaking thru Army B. (i.e., packet loss in a network)

$A_1$ and $A_2$ want to attack, but require assurance.

$A_1 \rightarrow A_2$: Attack saturday at Noon! Ack if you agree!
$A_2 \rightarrow A_1$: I agree; Ack if you get this!
$A_1 \rightarrow A_2$: I agree; Ack if you get this!
$A_2 \rightarrow A_1$: I agree; Ack if you get this!
$A_1 \rightarrow A_2$: I agree; Ack if you get this!

No solution!

**Instead: We relax conditions!**

$A_1$ will attack if there is a high probability that $A_2$ will too.

Let p be the probability of the message getting thru.

$A_1$ sends the message $n = 1/p$ times. (or $n = 100/p$)

$A_1 \rightarrow A_2$: Attack at Noon! x n times

We have "eventual consistency"
That's Bitcoin... wait n blocks.

# Byzantine Generals Problem

Scenario: Several divisions of the Byzantine Army stand to capture an enemy city IF they can coordinate.

- They coordinate only by messages.
- Each division is led by a general.
- One or more generals are faulty and malicious.
- Can the non-faulty generals find the faulty ones so that they can coordinate?

This is all an analogy. Each general is a server. We ask all servers for the latest version of a file. And some are hacked into. We want to discount/ignore/survive the malicious responses of the hacked servers.

Our goal is to have:

- All loyal (non-faulty) generals to come to the same decision on whether to attack (consensus)
  (traitors can do what they want)

- We don't want the disloyal (faulty) generals to trick or force the non-faulty generals in the wrong plan.

This is key:

1. All generals are known.
2. All messages are received within a bounded time window (else considered lost)

} not true for blockchain!

Something more formal:

We require an Algorithm so that
Loyal generals all report the same correct value
where v(i) is value reported by general i.
i = retreat or attack. (or an integer...)

attack or retreat ⇒ 0 or 1
it's the simplest consensus decision ever.

To make is clear who the parties involved are, we'll say that one general is a "commander", and the others are lieutenants.

Sometimes the commander is faulty, sometimes non-faulty. Sometimes it's the lieutenants, etc.

We are not going to sign messages with cryptography. That definitely would change things! Put that aside.

Let's focus on one lieutenant. She's trying to determine the consensus of the non-faulty nodes.

Trivial case: 2 generals. No solution!

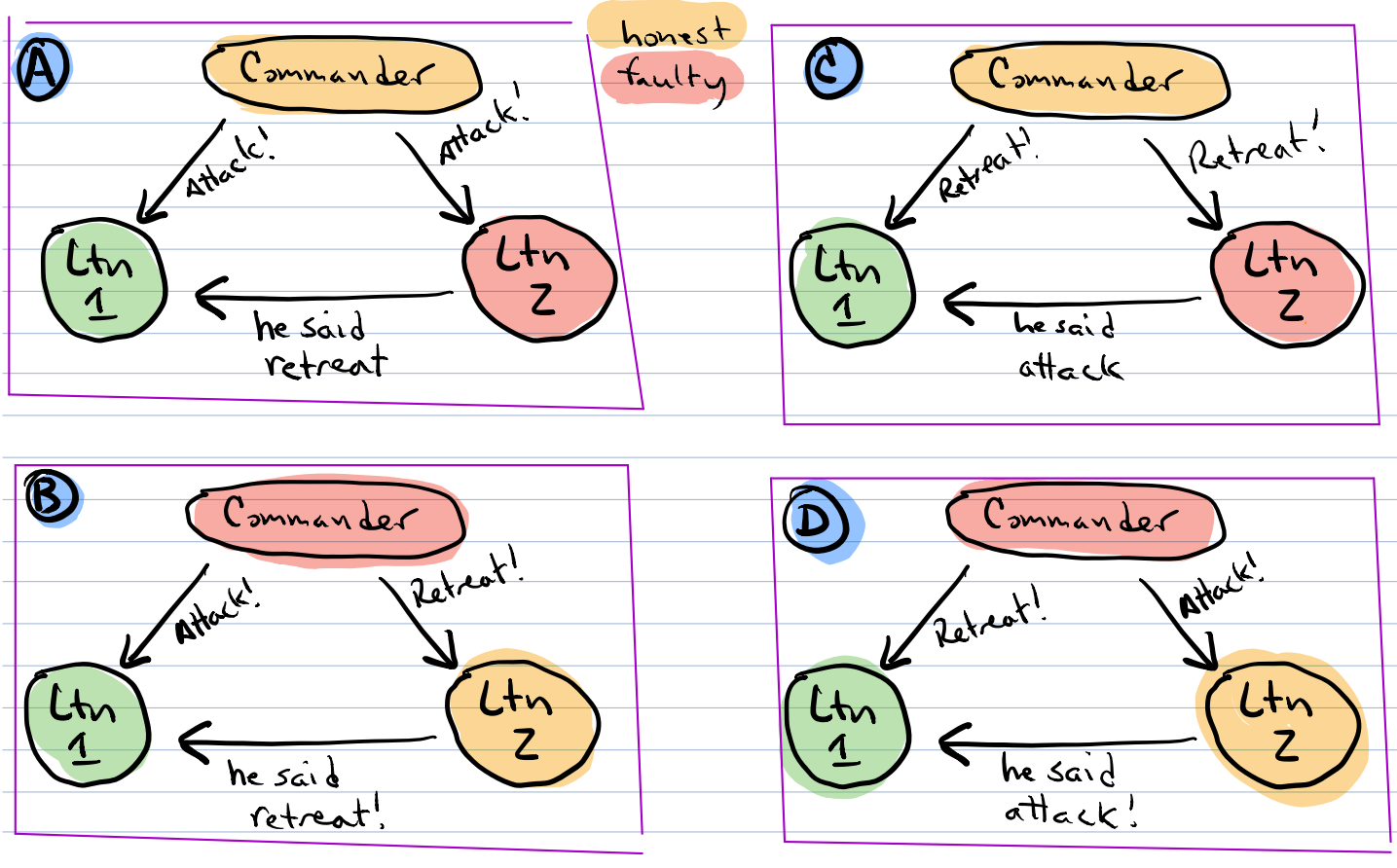
Consider our Ltn and a commander. Assume our Ltn is non-faulty.
She can't tell if the commander is faulty

IF true command is attack, then a non-faulty commander would send attack.
But if the true command is retreat, a fault commander would send attack as well.

There is also no solution for 3 generals.

How can our lieutenant determine if commander is faulty or if the other lieutenant is faulty, below:

She cant distinguish scenarios (A) and (B); nor (C) and (D).



Here's Lamport's Proof that no solution exists for fewer than $3m+1$ generals when $m$ of them are traitors/fault.

Proof by contradiction

Assume a solution exists for $3m$ generals (or fewer) when $m$ of them are faulty.

Let's use this assume solution to solve the case above of $m=1$ : 3 general, 1 is faulty.

We have $3m$ generals total, place them in 3 groups


Whatever assumed solution works contradicts the four cases above.

Therefore we have a contradiction, and no such solution exists. QED

Important follow on note:

- IF messages can be signed cryptographically
- And there is no sybil attack
- And keys can be distributed correctly
- And messages received in error (eg., due to tampering) can be corrected within a bounded time.
  - and keys can be distributed correctly
  - and there are synchronized clocks to accurately to apply the timeout/deadlines

Then a single traitor can be detected.

# Lamport's algorithm for consensus in the Byzantine Generals Problem. (1982)

- Assumes synchronous, ordered, bounded broadcast.
- n processes (servers, peer, etc)
- each process $i$ provides a value $V_i$ to the others.
- goal: to let each non-faulty process construct a vector $V$ of length $n$, such that if process $i$ is non-faulty then $V[i] = V_i$

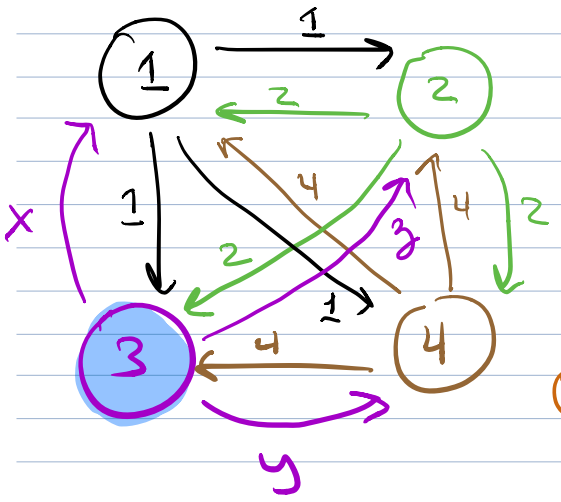
Assume at most $K$ faulty processes.

Here's an example of the alg for $n = 4$ and $K = 1$

It's four steps.

**Step 1** Every process sends values $v_i$ to <u>all</u> other processes. (For simplicity, assume $v_i = i$)

- Non-faulty processes will send the same value to all.
- Faulty ones can send a different value to each process.




V

1 got (1, 2, x, 4)
2 got (1, 2, y, 4)
3 got (1, 2, 3, 4)
4 got (1, 2, z, 4)

**STEP 2** Form the vectors.

③ is faulty – it sends x, y, and z to 1, 2, and 4

**STEP 3** Everyone passes their vectors to everyone else!

③ lies and doesn't even send out (1, 2, 3, 4)

| ① Gets | ② gets | ④ gets |
|---|---|---|
| | 1: (1, 2, x, 4) | 1: (1, 2, x, 4) |
| 2: (1, 2, y, 4) | | 2: (1, 2, y, 4) |
| 3: (a, b, c, d) | 3: (e, f, g, h) | 3: (i, j, k, l) |
| 4: (1, 2, z, 4) | 4: (1, 2, z, 4) | |

**STEP 4**

majority vote: 1 2 <u>U</u> 4 | 1 2 <u>U</u> 4 | 1 2 <u>U</u> 4

unknown = <u>U</u>

We don't care what <u>U</u> is, because we don't care what ③ ends up doing anyway.