EECS 591
Distributed Systems

Manos Kapritsos
Fall 2021

PROVING AGREEMENT

```
To execute propose($v_i$):
  round $k, 1 \le k \le f+1$
    Send $\{v \in V: p_i$ has not already sent $v\}$ to all
    for all $j, 0 \le j \le n+1, j \neq i$, do
      receive $S_j$ from $p_j$
      $V := V \cup S_j$

decide( ) occurs as follows:
    if $k = f+1$
      decide min($V$)
```

Lemma 2

In every execution, at the end of round $f+1$, $V_i = V_j$ for every correct process p_i and p_j

Agreement follows from Lemma 2, since *min* is a deterministic function

Proof

- Show that if a correct p has x in its V at the end of round $f+1$ then every correct process has x in its V at the end of round $f+1$
- Let r be the earliest round x is added to the V set of a correct process. Let that process be p^*
- If $r \le f$,then p^*sends x in round $r+1 \le f+1$ Every correct process receives x and adds it to its V in round $r+1$
- **What if $r = f+1$?**
  - By Lemma 1, there exists a sequence of distinct processes $p_0, \ldots, p_{f+1} = p^*$
  - Consider processes $p_0, \ldots, p_f$
  - $f+1$ processes; only f can be faulty
  - One of $p_0, \ldots, p_f$ is correct and adds x to its V before p^* does it in round r

Contradiction!

State Machine Replication

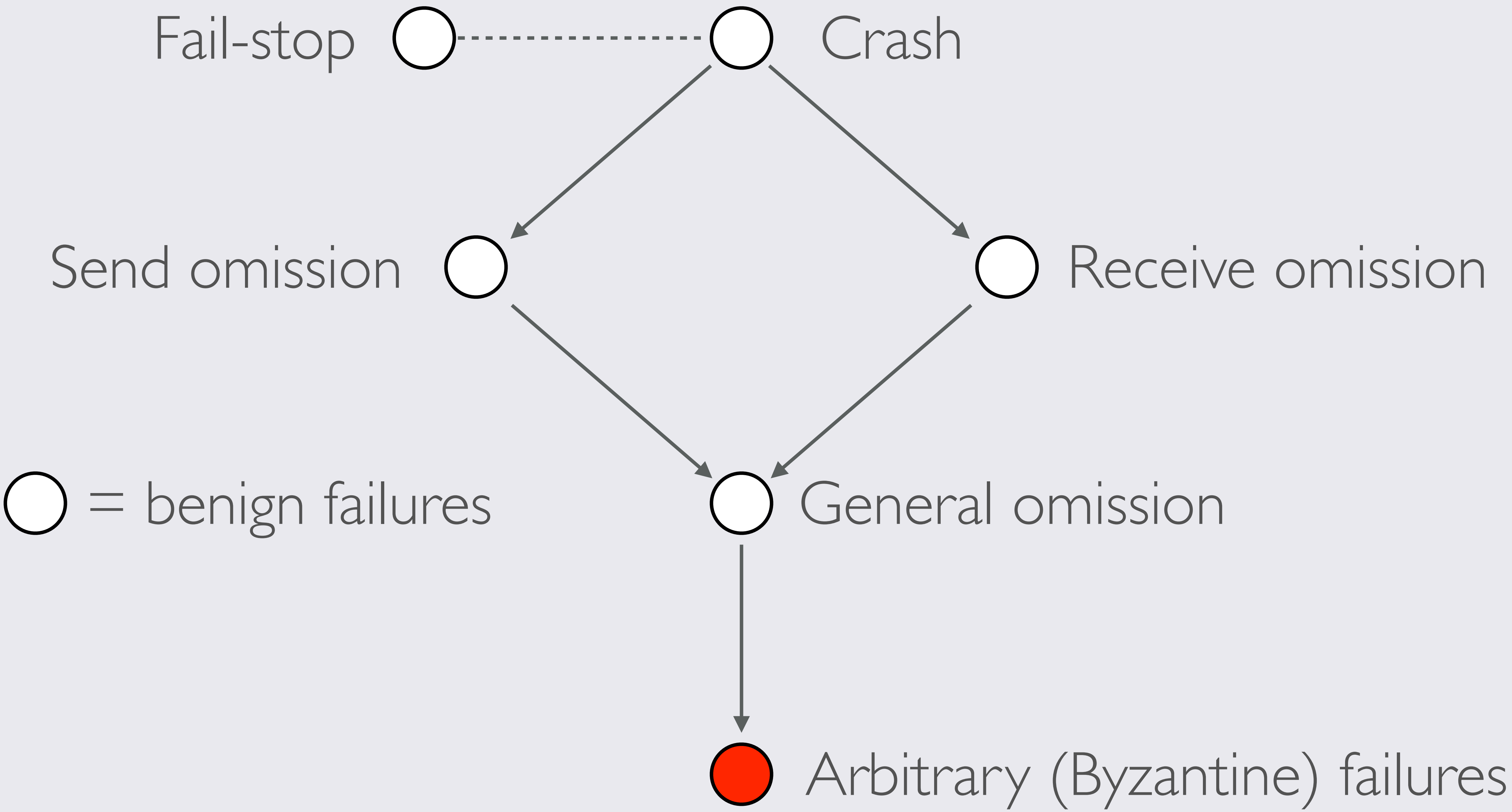
MODELING FAULTS

- Mean Time To Failure/Mean Time To Recover
  - used mostly for disks
  - of questionable value in expressing reliability

- Threshold: f out of n
  - makes condition for correct operation explicit
  - measures fault-tolerance of the architecture, not of individual components

- Enumerate failure scenarios

A HIERARCHY OF FAILURE MODELS



Fail-stop

Crash

Send omission

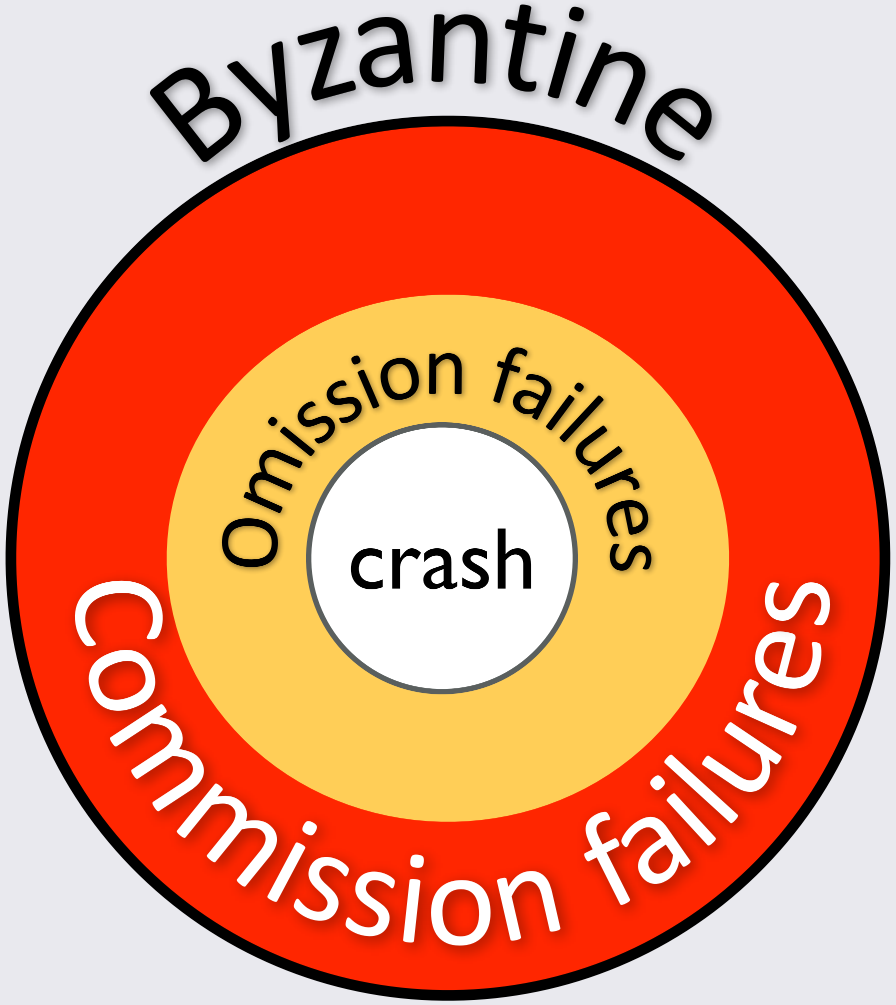
Receive omission

○ = benign failures

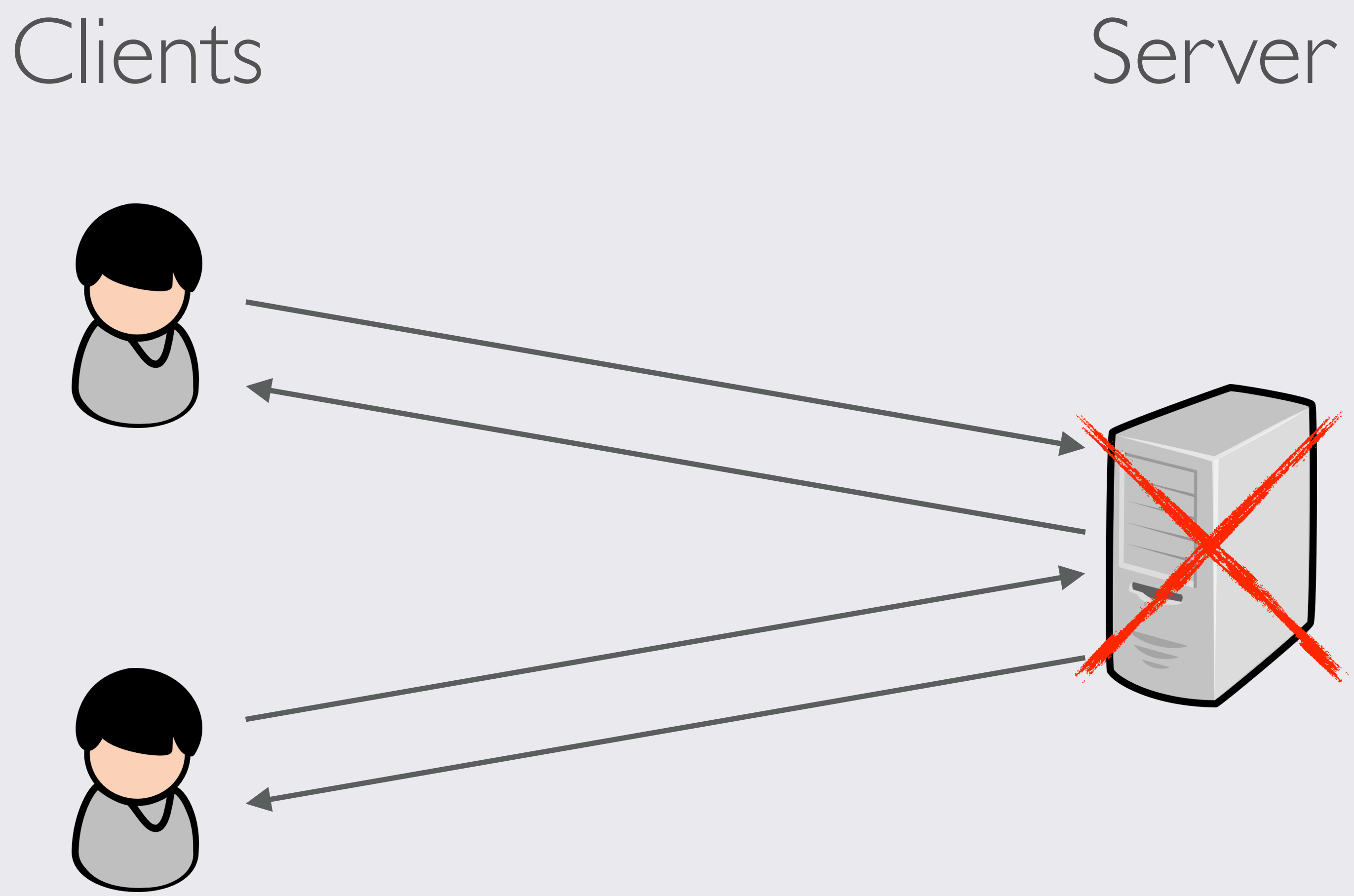
General omission

Arbitrary (Byzantine) failures

A HIERARCHY OF FAILURE MODELS



FAULT TOLERANCE: THE PROBLEM



Solution: replicate the server

Replication in time

- When a server fails, restart it or replace it
- Failures are **detected**, not masked
- Lower maintenance, lower availability
- Tolerates only benign failures

REPLICATION IN SPACE

- Run multiple copies of a server (replicas)
- Vote on replica output
- Failures are **masked**
- High availability and can tolerate arbitrary failures
  - but at high cost

THE ENEMY: NON-DETERMINISM

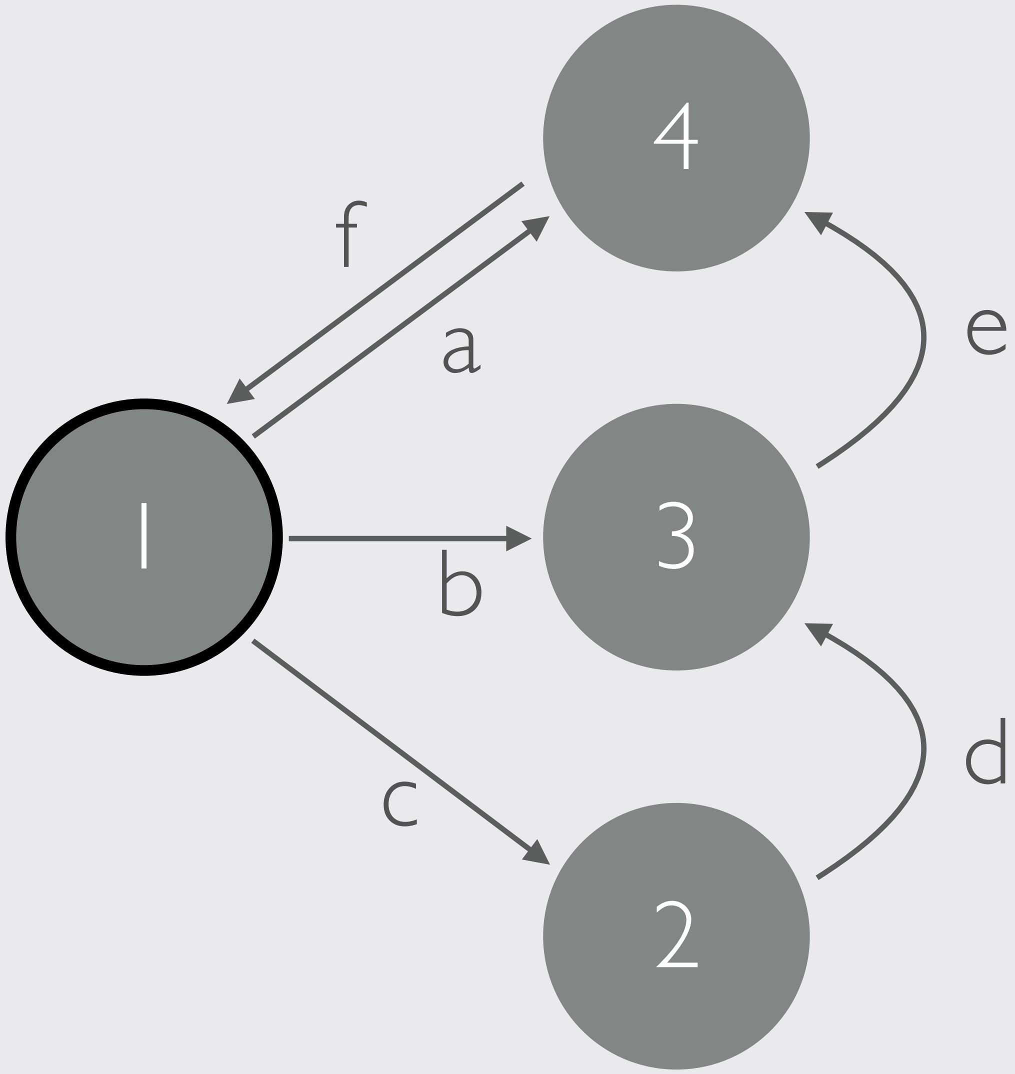
An event is non-deterministic if its output is not uniquely determined by its input

The problem with non-determinism:

- Replication in time: must reproduce the original outcome of all non-deterministic events
- Replication in space: each replica must handle non-deterministic events identically

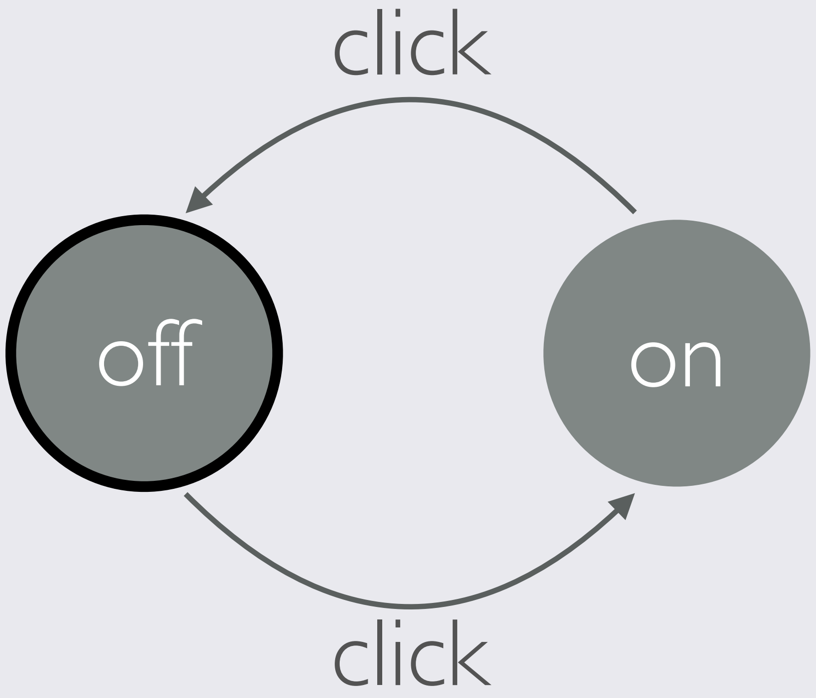
THE SOLUTION: STATE MACHINES

Design the server as a deterministic state machine

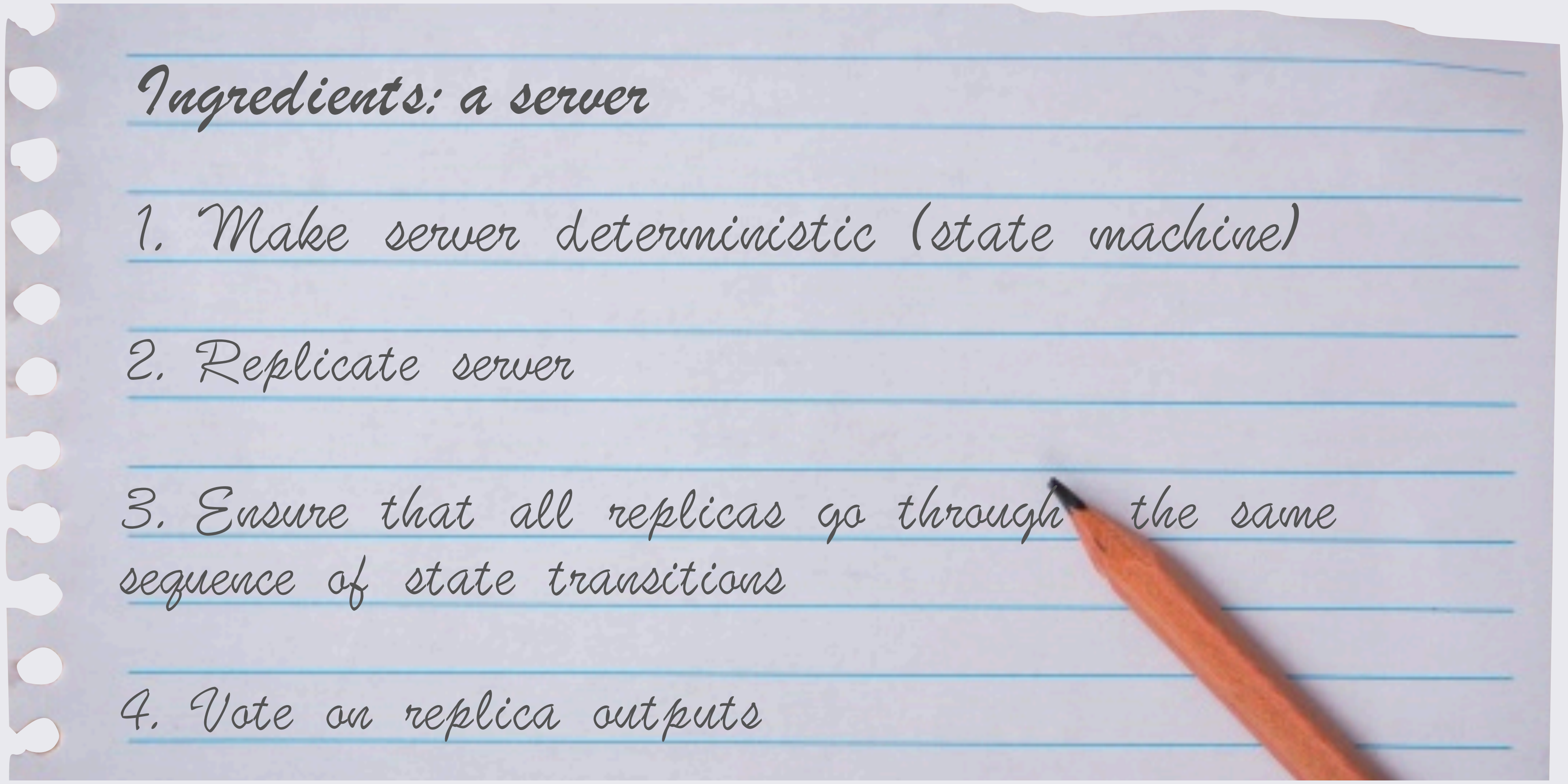


THE SOLUTION: STATE MACHINES

State machine example: a switch



State Machine Replication



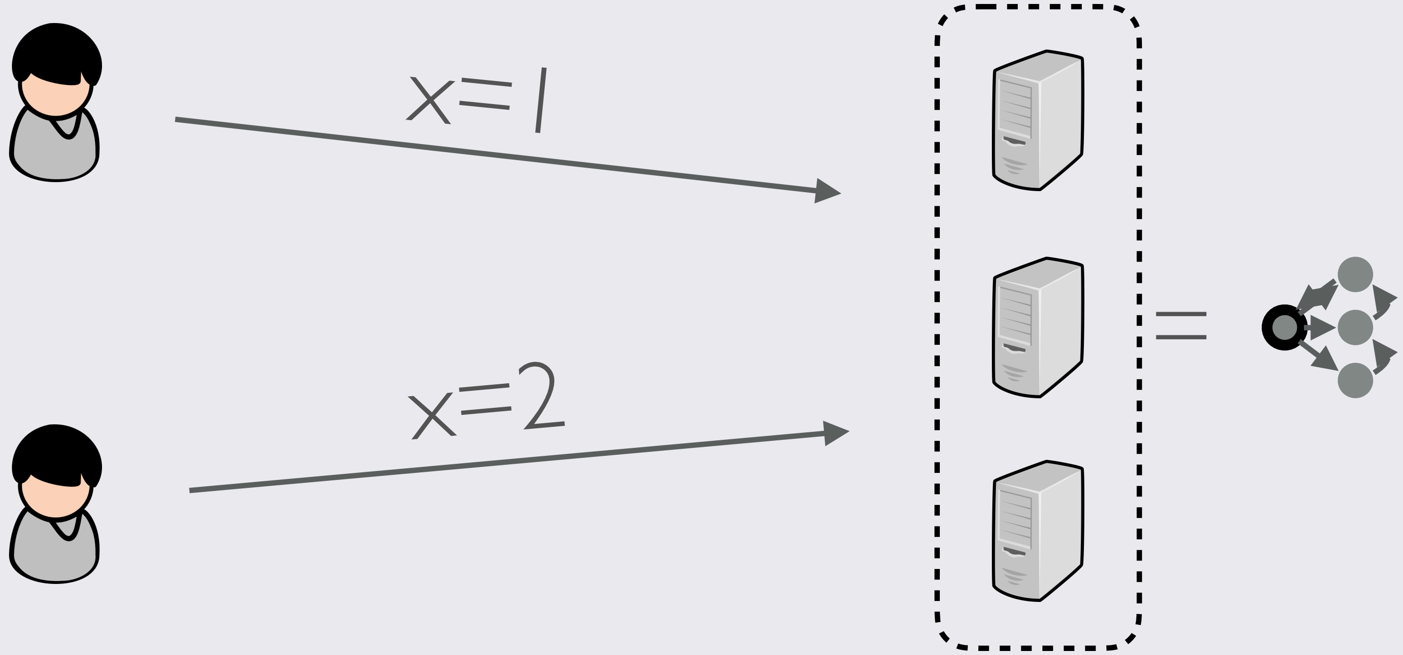
Ingredients: a server

1. Make server deterministic (state machine)

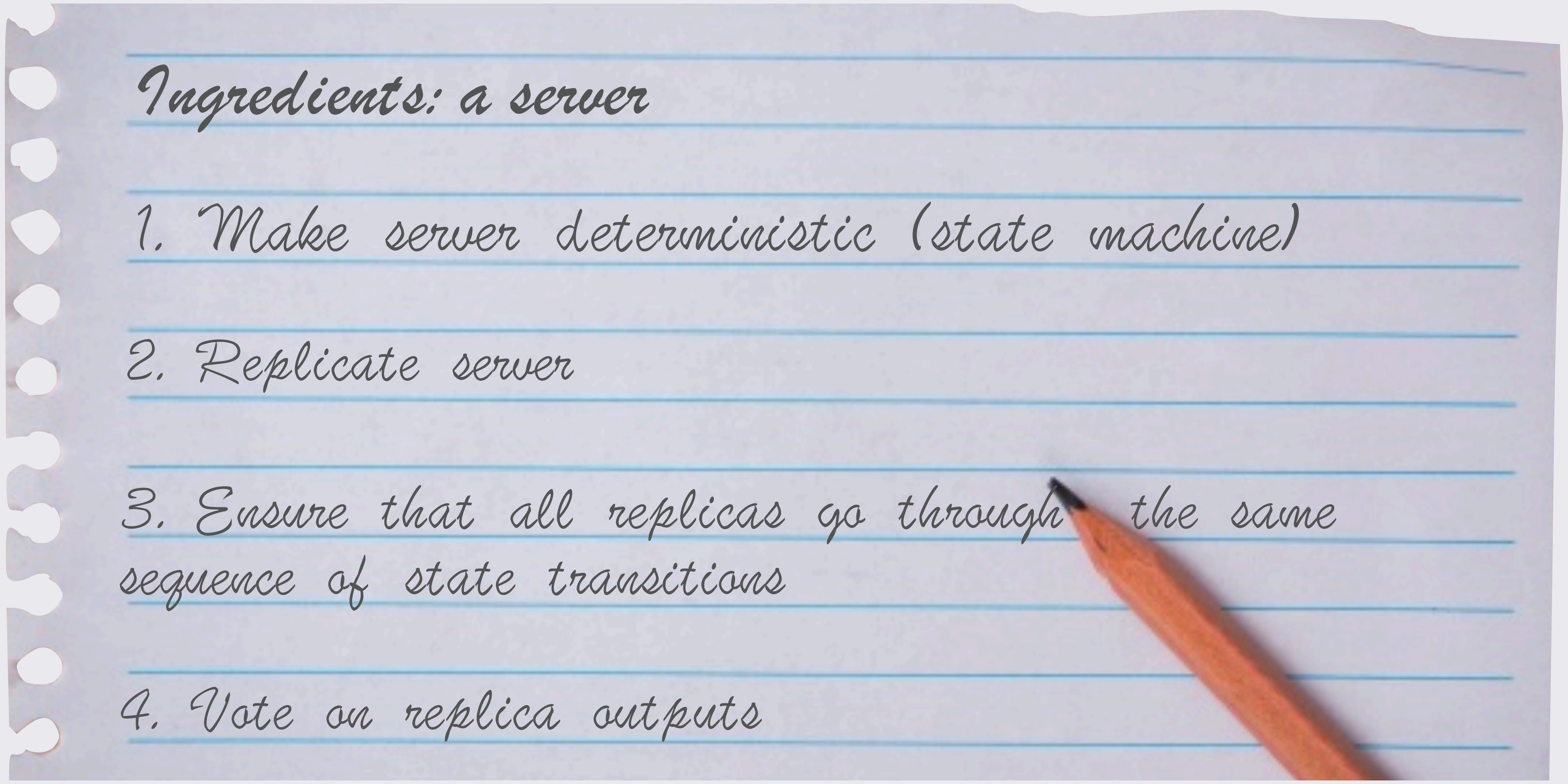
2. Replicate server

3. Ensure that all replicas go through the same sequence of state transitions

4. Vote on replica outputs



State Machine Replication



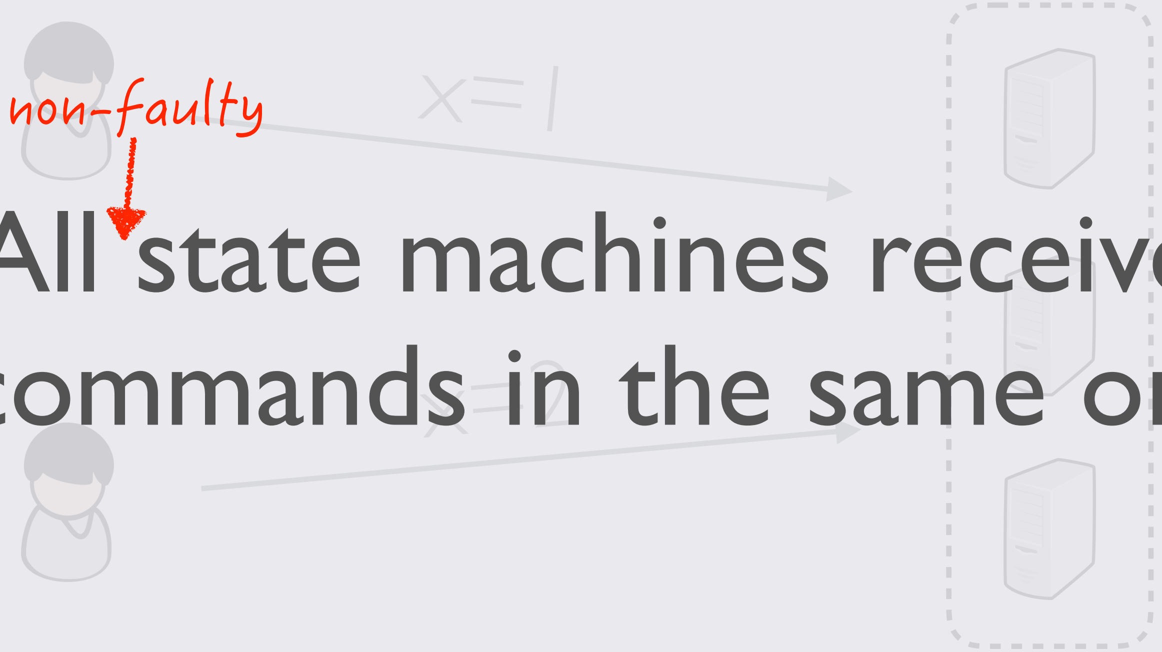
Ingredients: a server

1. Make server deterministic (state machine)

2. Replicate server

3. Ensure that all replicas go through the same sequence of state transitions

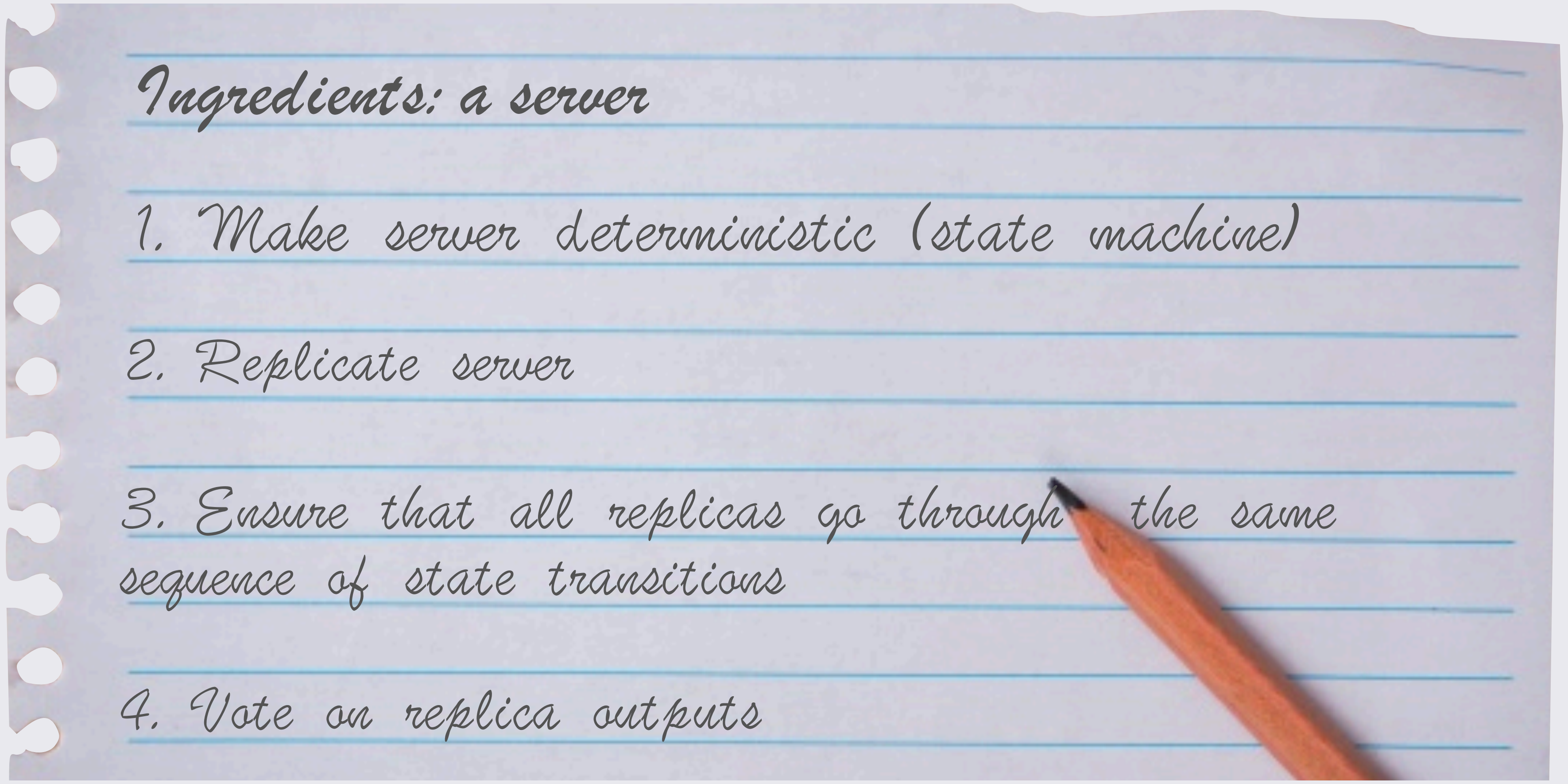
4. Vote on replica outputs



non-faulty

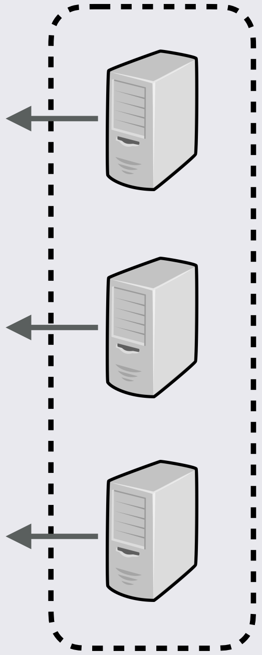
All state machines receive all commands in the same order

State Machine Replication

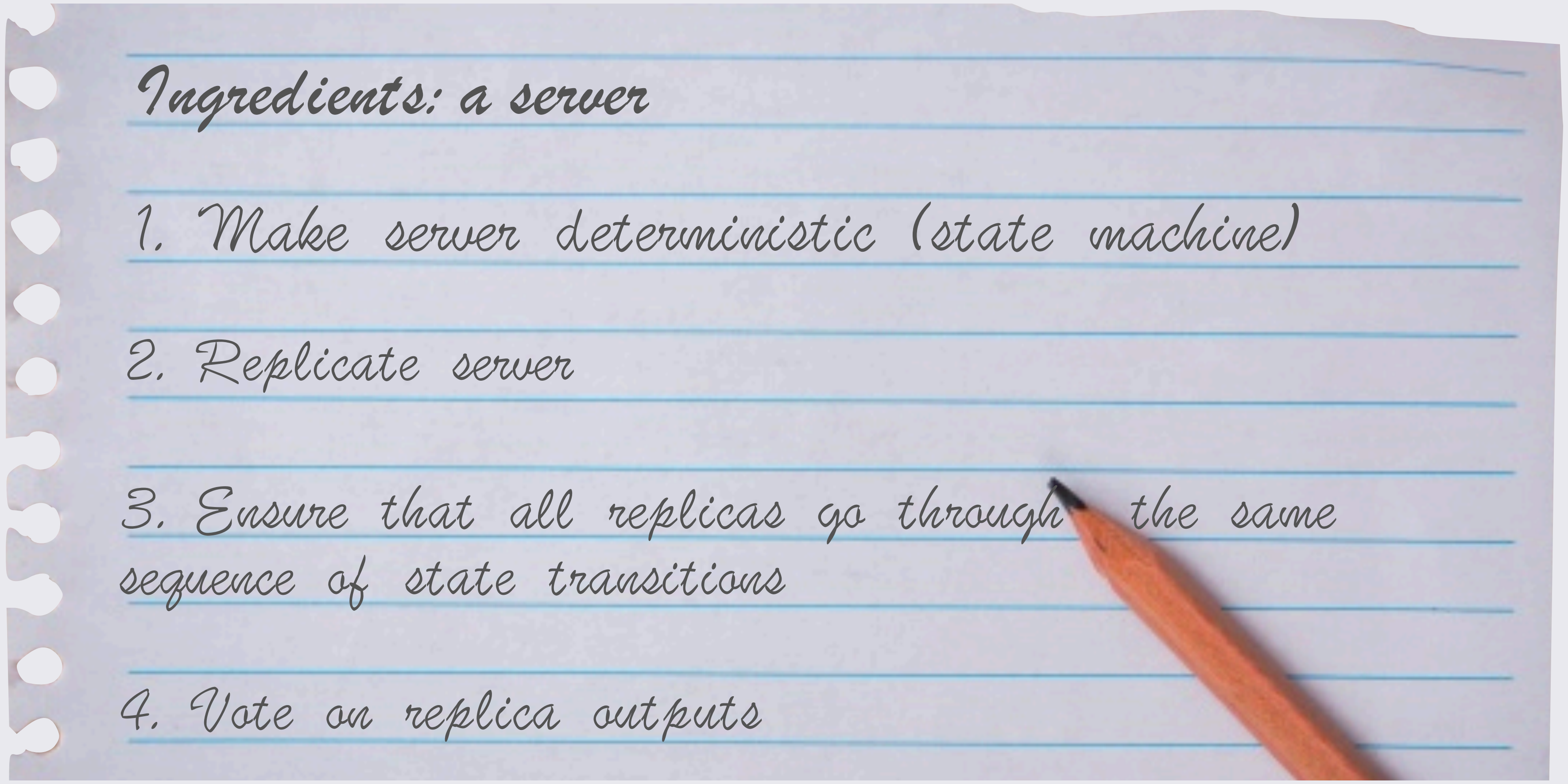


Ingredients: a server

1. Make server deterministic (state machine)
2. Replicate server
3. Ensure that all replicas go through the same sequence of state transitions
4. Vote on replica outputs



State Machine Replication



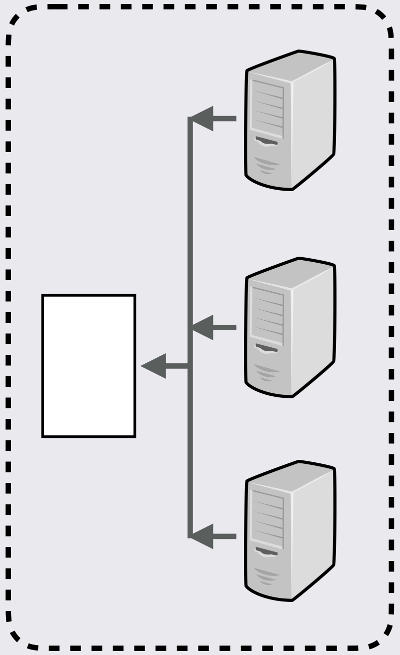
Ingredients: a server

1. Make server deterministic (state machine)

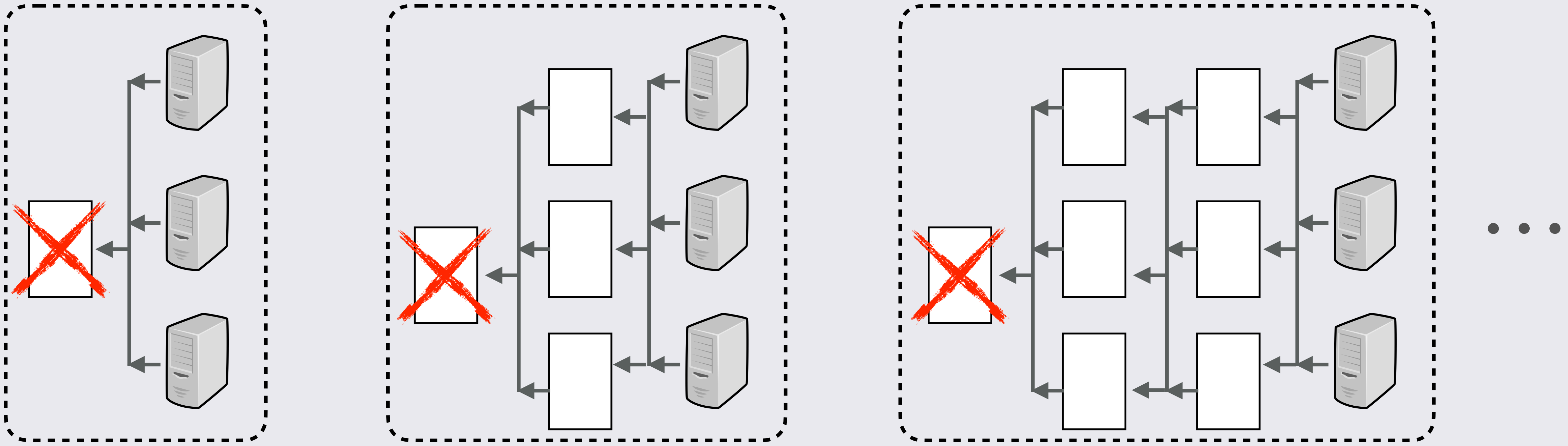
2. Replicate server

3. Ensure that all replicas go through the same sequence of state transitions

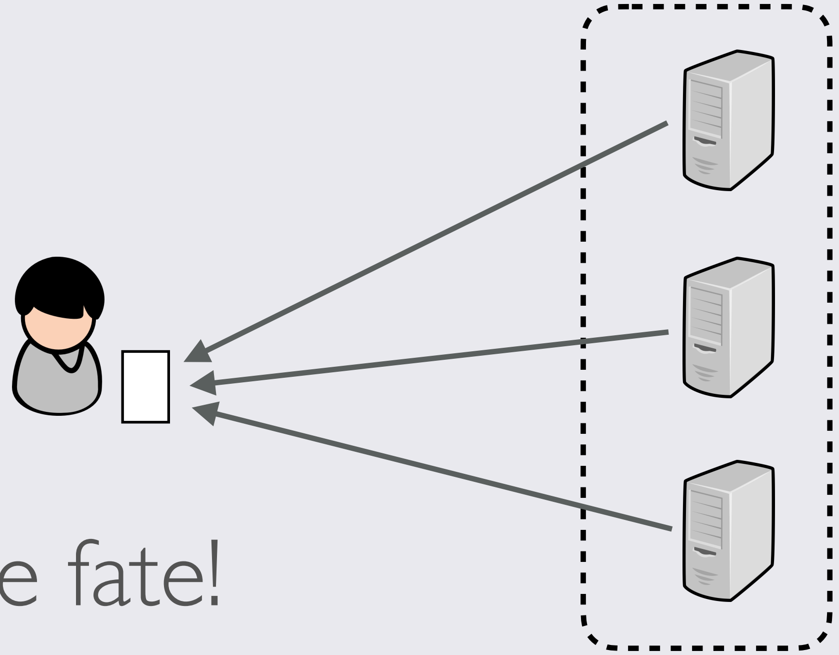
4. Vote on replica outputs



4. Vote on replica outputs



When in trouble, cheat!



Voter and client share fate!

ADMINISTRIVIA

- Send me your paper preferences by **tonight**
- Send me your group declaration preferences by **Oct 1**
- Homework #2 will be released on Wednesday
  - due Monday, Oct 11, before class
- Implementation project will be out next Monday
  - due Monday October 25, by end of day
  - Maximum team size: 2
- Research project topics due next Friday, **Oct 8**

Primary-Backup

The model

Failure model: **crash**

Network model: **synchrony**

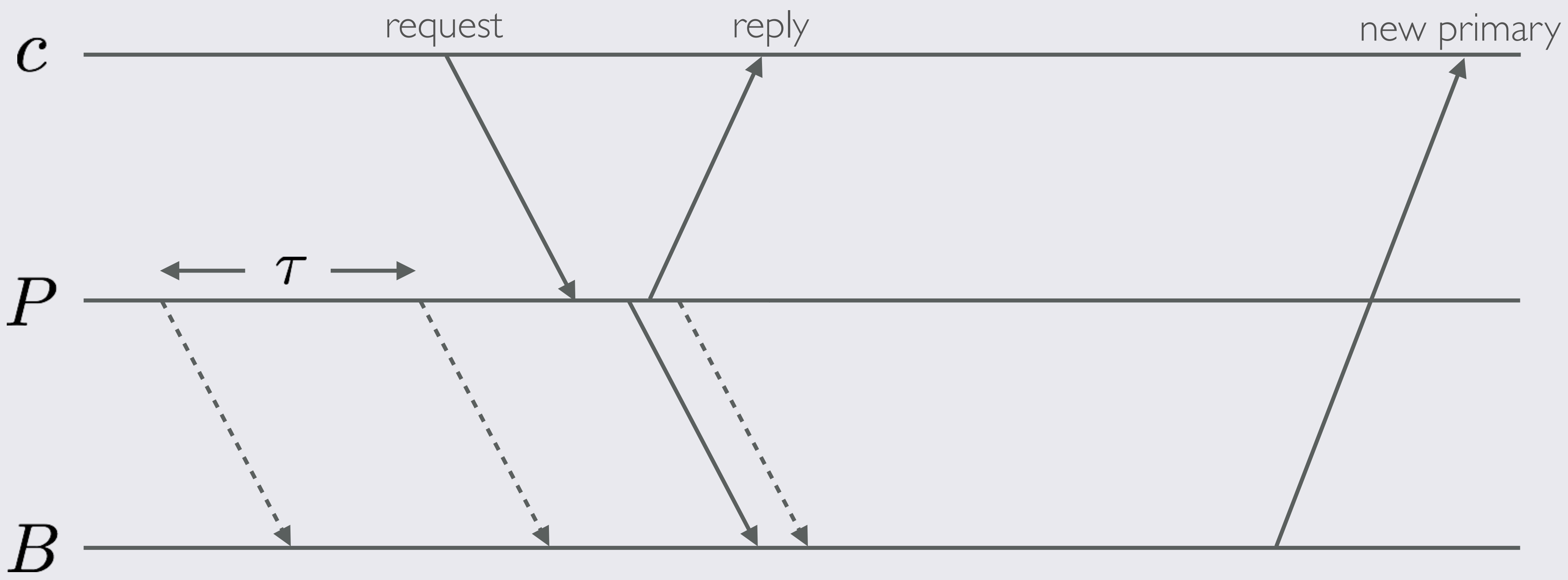
- Reliable, FIFO channels
- All messages are delivered within δ time

Tolerates f crash failures

THE IDEA

- Clients communicate with a single replica (**primary**)
- Primary:
  - sequences and processes clients' requests
  - updates other replicas (**backups**)
- Backups use **timeouts** to detect failure of primary
- On primary failure, a backup becomes the new primary

A SIMPLE PRIMARY-BACKUP PROTOCOL $(f = 1)$

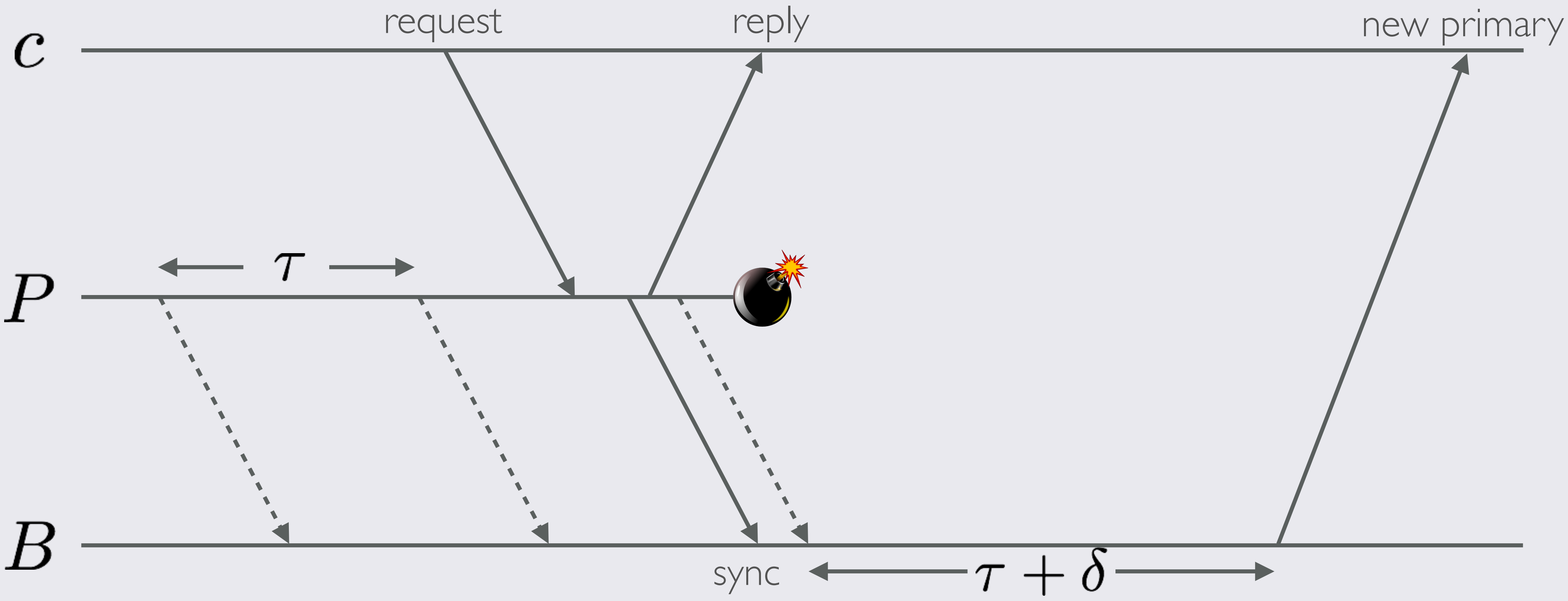


Active replication: sync = client request(s)

Passive replication: sync = state update

A SIMPLE PRIMARY-BACKUP PROTOCOL

$(f = 1)$



Active replication: sync = client request(s)

Passive replication: sync = state update

WEAKENING THE MODEL

Failure model: **crash**

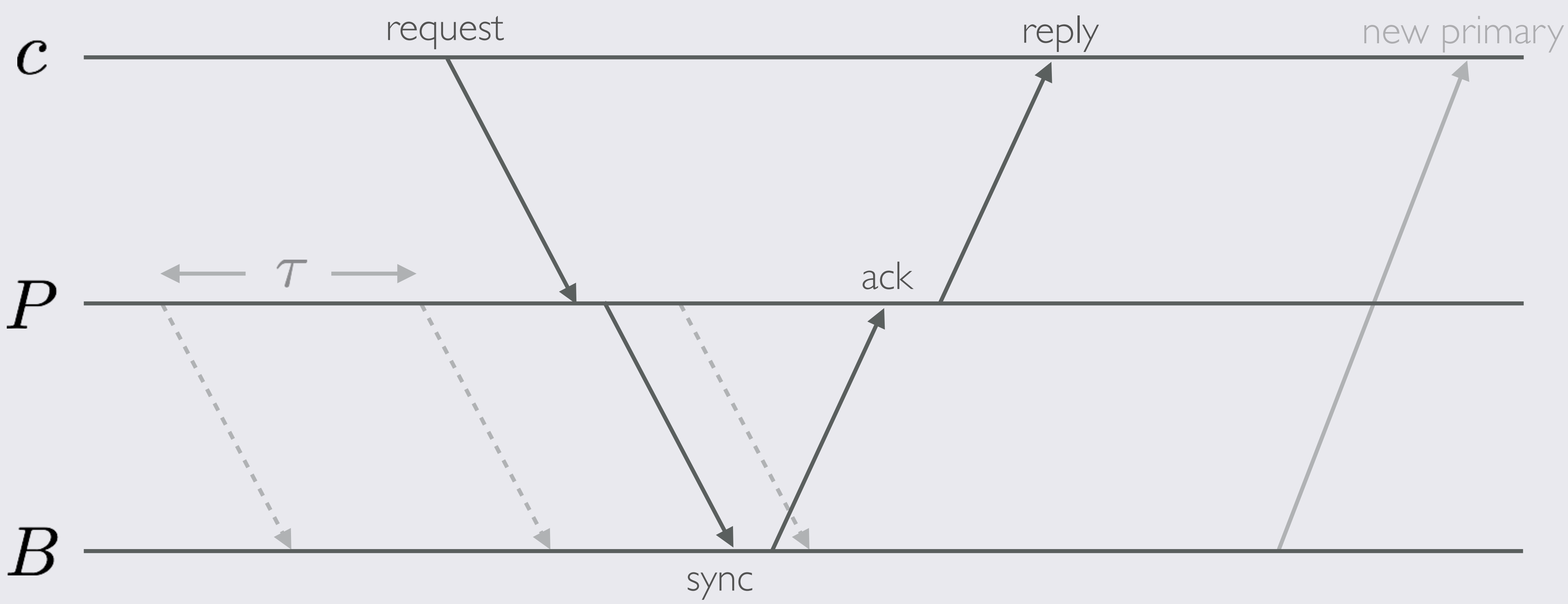
Network model: **synchrony**

- Unreliable, FIFO channels
- Channels may drop messages
- All messages are delivered within δ time
  - (looks paradoxical)

Tolerates f crash failures

A SLIGHTLY DIFFERENT PRIMARY-BACKUP PROTOCOL

$(f = 1)$



c
request
reply
new primary
P
τ
ack
B
sync