Distributed Systems :- Replicated State Machines

Where we are

- Last week
  - Consistency models
    - ↳ Linearizability
    - → Sequential Consistency
    - → Strict Serializability
  - Ended by talking about ways to build fault tolerant consistent ADTs

- Today : RSMs : Building fault tolerant systems

- Next week/Lab : Raft/ doing it in practice

REORDERING THINGS A BIT COMPARED TO LAST SEMESTER, GOING TO TALK ABOUT A FEW IDEAS TODAY THAT WE WILL REVISIT LATER

What we want : fault tolerance

Informally : Users ( or ) should not observe

system due to

any changes in a system ...

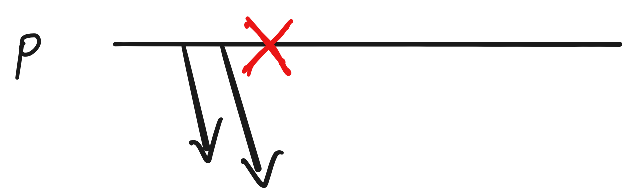
failures

Generally requires assumptions about

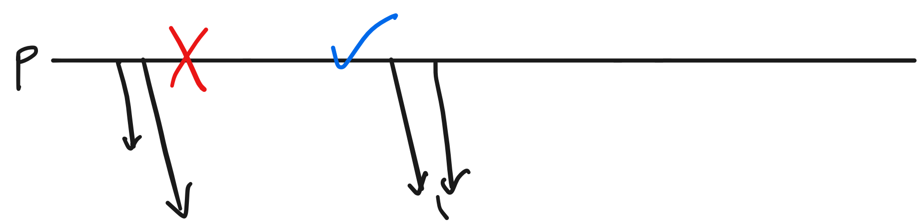
FAILURE MODEL
- # of processes that have failed
- Types of failures

Types of Failures

- Fail stop



- Fail Recover



- Byzantine

FOR MOST OF THE CLASS, GOING TO ASSUME FAIL-STOP

(but we will look at Byzantine later)

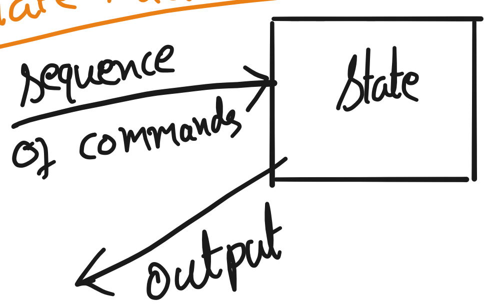
What we want: fault tolerance for an ADT/service

→ Assuming failures fit within model, want to ensure ADT behaves as if no failures

Our goal: A general framework for building fault tolerant ADTs:

Replicated State Machines

State Machine



State & output determined by sequence of commands

(Observe, definition assumes TOTAL ORDER OF COMMANDS)

- Queue

- R.Counter
    get():
        return count

    maybe_inc():
        if rand()%2 ==0:

count ++

- R. Counter-2

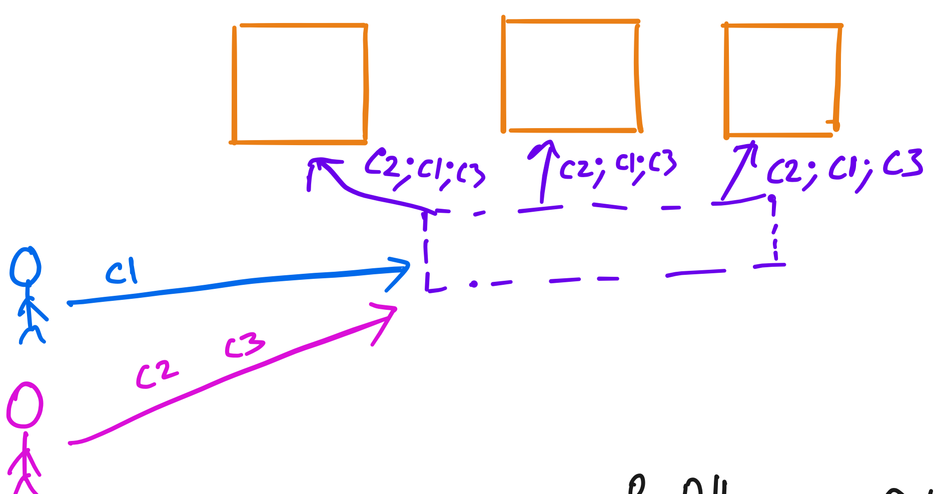
```
get():
    return count

maybe_inc(r):
    if r%2==0:
        count ++
```

Deterministic WRT Commands

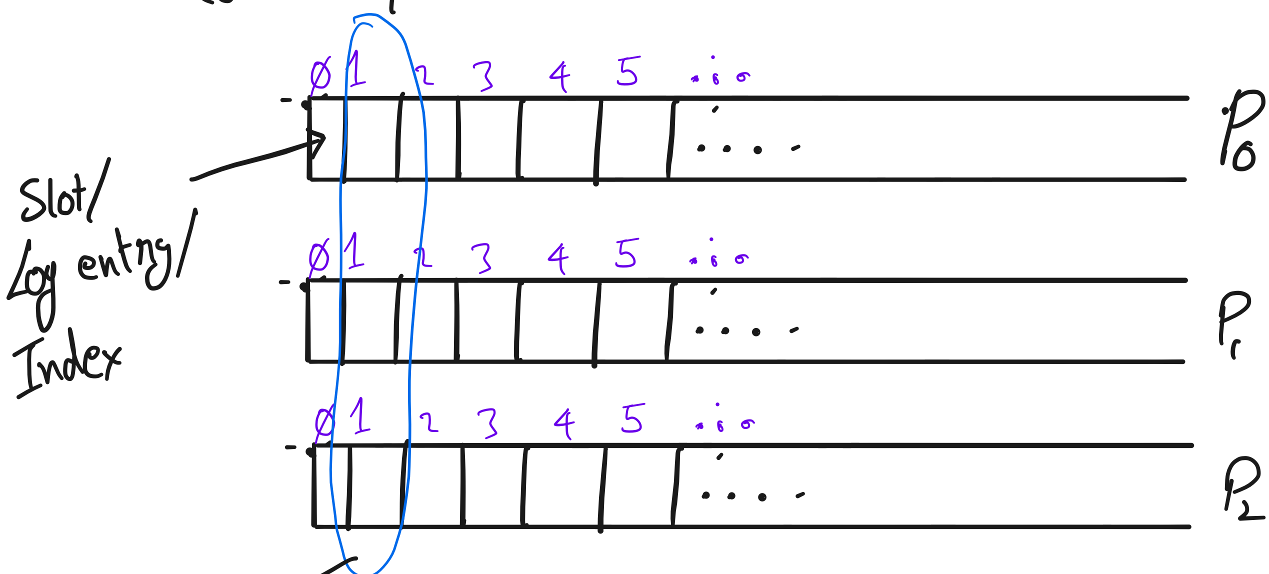
↳ COMMON ASSUMPTION FOR FAULT TOLERANCE

Replicated State Machines



Agreement & Ordering: All non-faulty replicas receive the same commands in the same order. Eventually

Core requirement: Agree on an ordered log



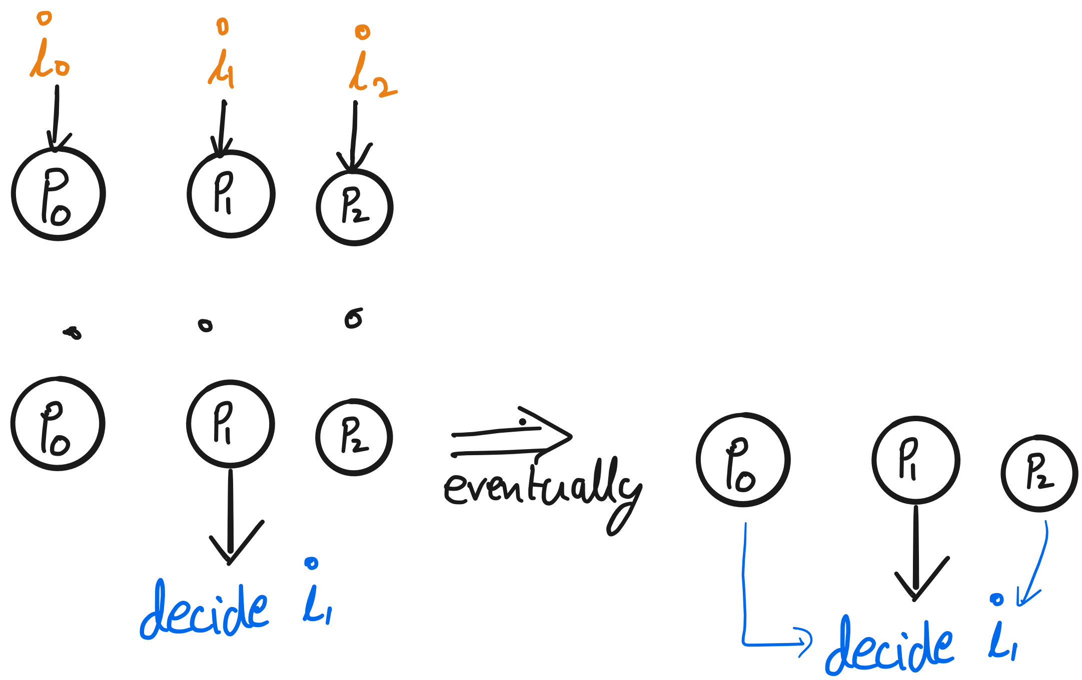
↳ Look at one slot[i]

Requirement:

(IC1) Agreement: If a correct process DECIDES slot i contains comand c, eventually all correct processes DECIDE i contains c.

Validity: If a correct process decides slot i contains c, then some client must have issued c.

General Problem: AGREEMENT/CONSENSUS



Fischer, Lynch, Paterson 1985 (FLP)/ Impossibility of Consensus

No deterministic protocol that works in the asynchronous setting can solve consensus & tolerate 1 (or more) failures.

Solve: Show that any safe algorithm might not terminate.

CIRCUMVENTING THIS

- Relax asynchrony assumptions:
  Partial Synchrony (Dwork & Lynch '88)

OR

Synchronous

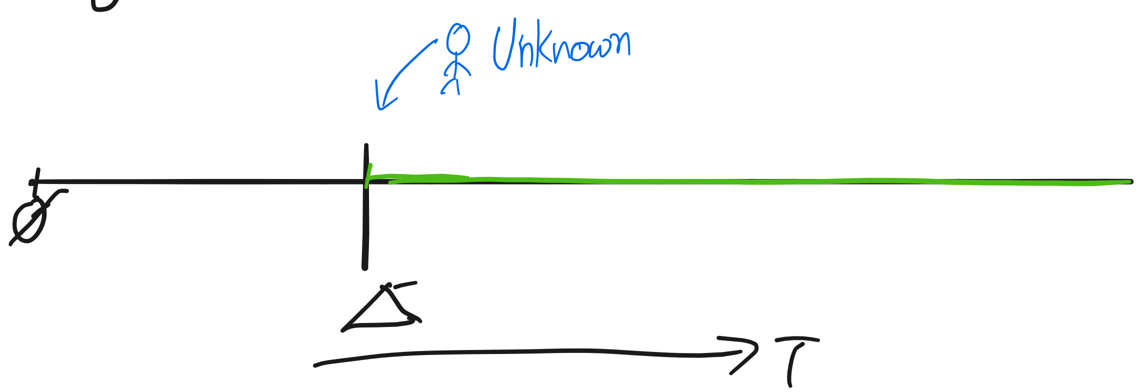
- Randomized algorithms

  Note: Using randomness for the algorithm, not in the state machine

- Failure detectors/Oracles

  (Add components to assume it away)

Partial Synchrony



Implications: Safety: Agreement & Validity

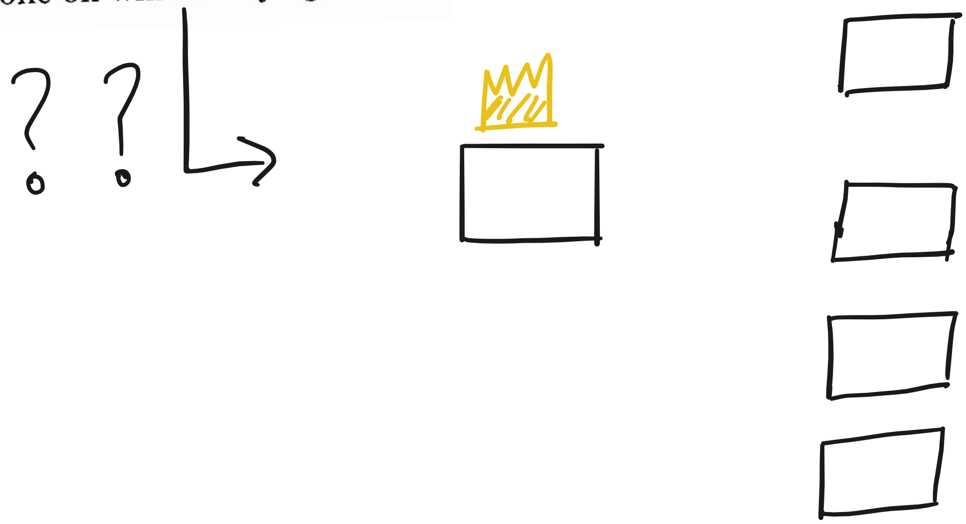
Liveness: Termination

Agreement from the paper

IC1: All nonfaulty processors agree on the ← Agreement from same value.

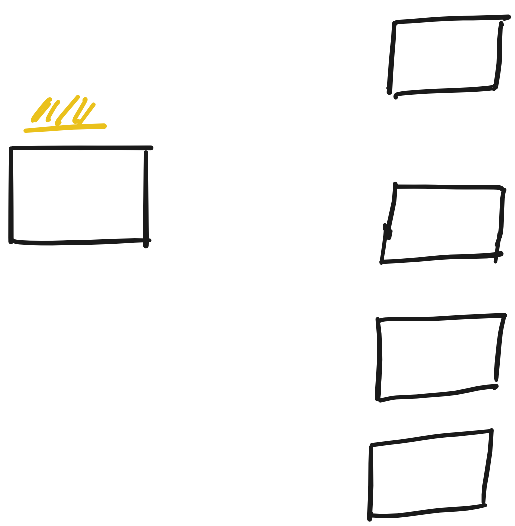
IC2: If the transmitter is nonfaulty, then all nonfaulty processors use its value as the one on which they agree.

Consensus



ORDERING

–



- Causal Ordering

- Other approaches

↳ Decouple replication & ordering

→

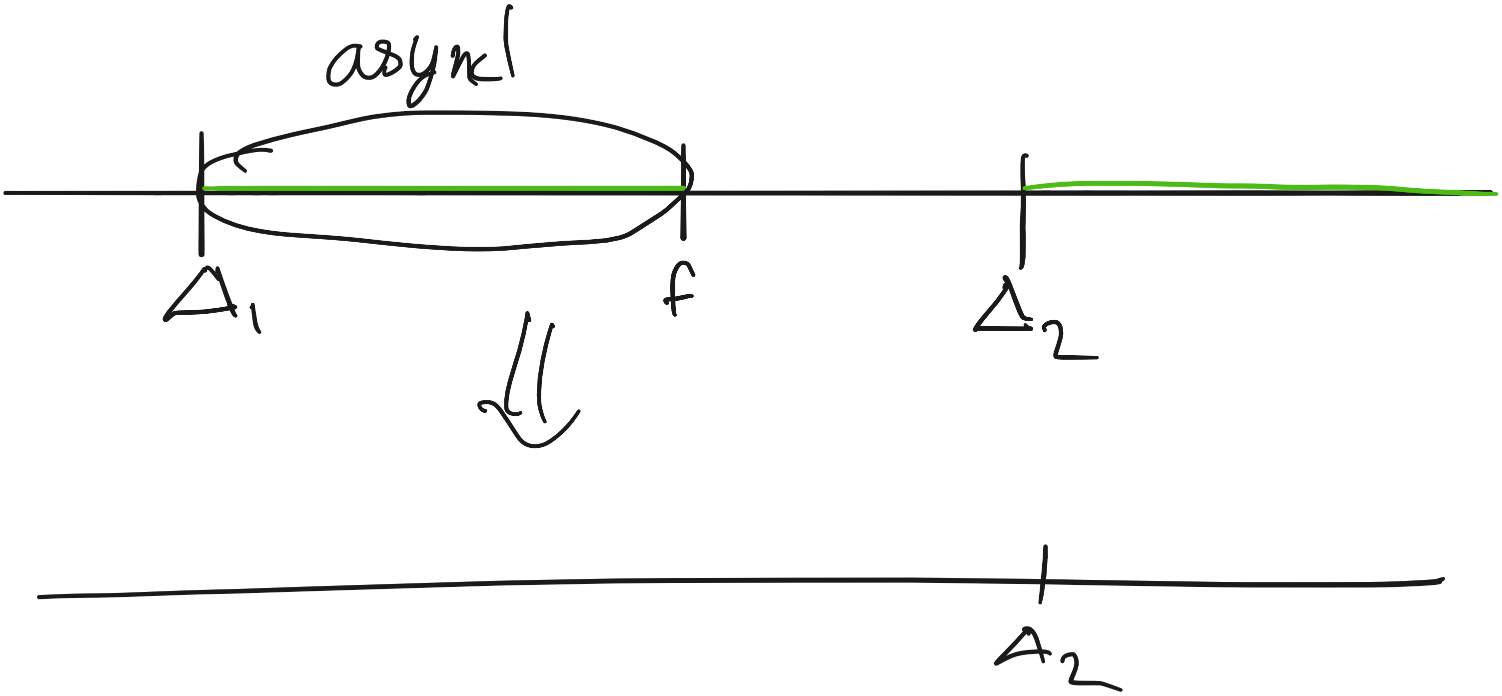
Our (as in this classes) toolkit for Building RSM

ⓐ Assume partial synchrony

ⓑ Use the same protocol for ordering

└ Raft (next class, lab)

But, important to remember other options are available.


async
Δ1
f
Δ2
Δ2