Replicated State Machines

(Or a preview of what is to come)

Where we are

- Distributed Systems: Processes + Async Network
  (can fail) (can drop/delay messages)

- Traces: way to model a single execution

- Correctness: constraints a trace should meet

  $\Rightarrow$ Linearizability

Brief diversion: Failure Models
Characterizing Failures

- What Can Fail: \( n; \; \frac{n}{2} - 1 \)
- What Happens When Something Fails

What Happens When a Process Fails

- Fail - [Stop]
  - No response
  - No longer running
  - No send or receive
  - No processing

- Fail - Recover
Fail-stop

Mach-up

Process Recovery is allowed

Must store state before failure

Communicate current state

Byzantine Failures

Others
Back to our regularly scheduled programming: RSMs

Two options:

1. I want game server

No games when server fails

Multiplayer
Tetris + Pong
Server
* Build new fault tolerant game server
* Run many copies of the same game server
  → Replicas

Why replication is useful / good + who uses this
Desirable Properties

- Non-faulty processes agree on committed log values.
- Committed log values were previously proposed by some client.
- Commands from a single client appear in order.
- What about commands from different clients?

Committed Entries

Decided

\[
\begin{array}{c|c|c|c|c}
\hline
1 & 2 & 3 & 4 & 5 \\
\hline
A & X & & & \\
B & & & & \\
\hline
\end{array}
\]
Log Agreement Between Processes

- Agreement: Processes agree on committed log entries.
- Validity: All committed entries were proposed by a process or client.

Next Week

FLP

How to Address This Problem

Partial Synchrony
Saw a similar approach in the Lamport paper.

Ordering requests from a single client.
Ordering Requests From Multiple Clients
View Change

\[ \exists A, B, C, D, E \]

\[ \exists A, D, E \]

OUN Commit(\text{NN})

CAN I JOIN?