SHARED MEMORY

(End Of Semester Announcements)

Final Exam

- 12/16, 5:10-7pm This Room
- Same rules as midterm: open book, notes, no collaboration
- Cumulative: covers everything, this semester

Poster Session
- THIS ROOM; NEXT WEEK DURING CLASS
- IN AEO OF PRINTING POSTERS (SINCE NO LONGER FREE/GUID?)
  JUST SHOW ON YOUR LAPTOP

- EXPECTATION:
  + WALK AROUND, TALK TO YOUR CLASSMATES, UNDERSTAND WHAT THEY DID
  + TALK TO ME, TELL ME WHAT YOU DID + WHAT YOU FOUND
- UPLOAD POSTER TO LINK ON CAMPUSWIRE BEFORE 5PM NEXT WEEK

PROJECT REPORT

- DUE MONDAY, DEC 13
- UP TO 7 PAGES (+ UNLIMITED CITATIONS)
  → PREFER BREVITY, NO NEED TO HIT PAGE LIMIT

- PLEASE PROOFREAD YOUR REPORT TO MAKE SURE I CAN UNDERSTAND IT.
  → TOTALLY FINE IF YOU DESCRIBE A CONCEPT RATHER THAN USE JARGON
- USE SECTIONS THAT ARE SIMILAR TO RAFT PAPER OR ONE OF THE OTHER PAPERS
SHARED MEMORY

(PHEW)

WHY?

- INTERESTING PROGRAMMING MODEL

- RELEVANT PROGRAMMING MODEL

- A CHANCE TO TALK ABOUT PARTIAL NETWORK CONNECTIVITY

RELEVANCE

- A RECENT WAY TO BUILD DISTRIBUTED APPLICATIONS
*Problem with slow distributed applications*

**Where?**
- Serverless
- Kubernetes, many resource management frameworks
- Task Queues
  - Dropbox

**Why?**

**Why Not?**
- Security
- Performance
  - messy complexity
  - latency

**Why now?**

Networks
PARTIAL NETWORK CONNECTIVITY

MODEL THUS FAR

ASYNC

FAIR

Consequence

$P_0$ sends copy of $m$ to $P_i$ 
$\Rightarrow P_i$ receives?

$P_0$ sends copy of $m$ to $P_2$ 
$\Rightarrow P_2$ receives?

REALITY IS MORE COMPLICATED
- Networks are sometimes asymmetric
  - Why? Configuration or implementation bugs

- Our protocols do not necessarily account for asymmetry

*Diagram*

```
0 --- AEC(1, 1, ...) --- M1
|         |                  |
V         |                  |
1 ---     --- 2
|         |                  |
V         |                  |
3 ---     --- 4
|         |                  |
V         |                  |
```

FAIL

**Registers**

- Many types

**Who can access**

**How consistent**

<table>
<thead>
<tr>
<th>SR SW</th>
<th>Atomic</th>
</tr>
</thead>
<tbody>
<tr>
<td>M R S W</td>
<td>Causal</td>
</tr>
<tr>
<td>M R M W</td>
<td>Eventual</td>
</tr>
</tbody>
</table>

...
Atomic vs Linearizable

1. Atomic

\[ x = 1 \]
\[ w_x(5) \rightarrow \text{OK}(c) \]
\[ r_x \rightarrow \text{OK}(c1) \]

2. Linearizable

\[ x = 1 \]
\[ r_x(0) \rightarrow \text{OK}(c5) \]
\[ r_x(0) \rightarrow \text{OK}(c) \]
\[ r_x(5) \rightarrow \text{OK}(c1) \]

3. MRSW \rightarrow MRMW

\[ a \leq b \]
\[ a \leq c \]
\[ a \leq d \]

**Example:**

- **Read:**
  - Read \( x_0 \ldots x_n \)
  - Find max version
  - Return value of register with max version, min index

- **Write:**
  - Find max version
  - \( x_n \leftarrow (\text{max version, value}) \)

\[ 3 \]
3 Shared Memory / Registers → Message Passing

4 Message Passing → Shared Memory (Attiya, Bar-Noy, Dolev)

MR SW \[\text{Writer knows previous versions}\]

Focus on unbounded case: Version ∈ \(\mathbb{Z}^+\)

WRITE

```
WRITE (VALUE, VERSION)
```

Wait for majority / quorum to Ack.

READ
Pick max Version. In this case $V_w$. Why?

Wait for Ack from Majority.

Why is this step needed?

$W_x(5)$  $O_k(\cdot)$

$X = 1$  $R_x(\cdot)$  $O_k(\cdot)$  $R_x(\cdot)$  $O_k(\cdot)$
So is everything equivalent?

Not quite

Message passing $\rightarrow$ Shared memory

Seems to reduce fault tolerance.
(Operations require Acks from majority).

Two possibilities

1. Our emulation protocol is bad & there is no gap (they are equivalent)

2. There is a gap (but does it require a majority)?
How To Answer the Weakest Failure Detector for Implementing S.M.

Why Does This Answer The Question?

Quorum Failure Detector $\Sigma$

$$\Sigma(\pi, t) = 2^\pi \quad \text{s.t.}$$

$$\forall p_1, p_2, t_1, t_2 \quad \Sigma(p_1, t_1) \cap \Sigma(p_2, t_2) \neq \emptyset$$

All Events Complete
Z → MRSW Register

Z For a Quantum Intersection

TRUSTED No faulty Nodes

AND Evidential Reason
Given: Protocol that implements MRSW registers for processes \( \Pi \)

Want: \( \Sigma (p, t) \): Quorum FD at process \( p \), time \( t \).

Create a MRSW register \( \Sigma_p \) for each process \( p \).

\[
\text{Init: } \Sigma (p, 0) = \Pi, \quad \Sigma_p = (0, \Pi)
\]

\( F_p = \left[ \Pi \right] \) [Trust All Processes Initially]

Loop: [\( k \) loop variable]

- **Step 1:** Process \( p \) writes \( \pi_p \leftarrow (k, F_p) \) & gathers set of processes \( \{w_k\} \) involved in write.

  Fault tolerance + atomicity

  \[\Rightarrow \exists w_k \geq 1 \& w_k \text{ contains connected process.}\]

- **Step 2:** Process \( p \) reads \( \pi_j \forall j \in \Pi \) [Including \( \pi_p \)]

For convenience \( \pi_2 = [p_1, p_2, p_3, \ldots] \)

\[\text{where } p_{1:k} \subseteq \Pi\]

- **Step 3:** Process \( p \) sends a message to all processes and waits until \( \exists j \forall j \in F, \text{ that} \)

\[
\text{(message sent)}
\]
and waits until $V_{ij}$'s "right hand has responded. [Note: Never need to wait for more than $|\mathcal{M}|$ processes. Why?]

Step 4: $\Sigma(p, k+1) = w_k \cup \bigcup_{i \in \mathcal{M}} E_{p_i \text{ appended } (\Sigma(p, k+1))}$

Step 2 - 4: Intersection?

How can this ensure that eventually $\Sigma(p, k)$ only contains correct processes?

Where does all of this leave us?

- Can emulate registers with message passing
- On message passing with registers

- But comes at the cost of fault tolerance

- DOES THE FT LOSS MATTER?
Back to Shared Memory in Practice

Logically Distinct.

- What does F7 Observation Mean for This?

```
 STORAGE
  - etc.
  - S3
  - Dynamo
  ...

1

2
```

```
1

2
```