PREPARING FOR THE “SYSTEMS” PART OF THE COURSE

I will announce a list of papers

You each pick one to present (email me 3 preferences)

- ~35 minutes presentation
- Send me the slides two days before your presentation
Consistency

Is the server’s response correct?

(are all the server’s responses consistent with each other?)
Consistency is a property of the execution; a constraint on the values of the reads and writes returned by the server.
Monotonic read consistency

If a client reads the value of a data item \( x \), any successive read operation on \( x \) by that client will always return that same value or a more recent value.

Are these runs monotonic read consistent?

\[
W_1(x,3) \quad R_1(x)=4 \quad W_2(x,4) \quad R_2(x)=4
\]

\[
R_1(x)=1 \quad R_1(y)=1 \quad W_2(y,4) \quad R_1(x)=4
\]

\[
\ldots \quad R_1(x)=1 \quad R_1(y)=1 \quad W_2(y,4) \quad R_1(x)=4
\]
Read-your-writes consistency

A read by a client on data item \( x \) should reflect all previous writes of that client on \( x \).

Are these runs read-your-writes consistent?

\[
\begin{align*}
  W_1(x,3) & \quad R_1(x)=4 & \quad W_2(x,4) & \quad R_2(x)=4 \\
  R_1(x)=1 & \quad R_1(y)=1 & \quad W_2(x,4) & \quad R_1(x)=4 \\
  W_1(x,1) & \quad W_1(x,2) & \quad W_2(x,4) & \quad R_1(x)=2
\end{align*}
\]
Causal consistency

All processes see causally related events in the same order.
Causal consistency

All processes see causally related events in the same order.
Causal consistency

All processes see causally related events in the same order.

A student removes advisor from friends list and then posts Spring Break photos

The advisor should not be able to see the pictures
**Sequential Consistency**

The result of any execution is the same as if the operations of all the processes were executed in *some sequential order*, and the operations of each individual process appear in this sequence in the *order specified by its program*.

Are these runs sequentially consistent?

1) P₁: W(x,3)  
   P₂: W(x,5) R(x)=3

2) P₁: W(x,3)  
   P₂: [W(x,5), R(x)=3]
**Linearizability**

Same as sequential consistency, but the sequential order must preserve the *real-time* constraints of non-overlapping operations.

1)  
   \[ c_1 \xrightarrow[W(x,3)]{} W(x,5) \xrightarrow[R(x)=3]{} c_2 \]

2)  
   \[ c_1 \xrightarrow[W(x,3)]{} W(x,5) \xrightarrow[R(x)=3]{} c_2 \]

3)  
   \[ c_1 \xrightarrow[W(x,3)]{} W(x,5) \xrightarrow[R(x)=3]{} c_2 \]

4)  
   \[ c_1 \xrightarrow[W(x,3)]{} W(x,5) \xrightarrow[R(x)=3]{} c_2 \]
VISUALIZATION OF LINEARIZABILITY

Operations must appear to happen instantaneously, at some "linearization" point between their start and end.

1)

2)
Operations must appear to happen instantaneously, at some “linearization” point between their start and end.
IN OTHER WORDS...

**Linearizability:** Once an operation is complete, its effects are immediately visible to all new operations

(operations are consistent, even in the presence of out-of-band communication)
Weak and strong properties

Property A is stronger than property B if A admits fewer executions than B (we also say that B is weaker than A)

Linearizability is stronger than sequential consistency is stronger than causal consistency is stronger than FIFO consistency
Consensus and Reliable Broadcast
If a process sends a message \( m \), then every process eventually delivers \( m \).

How can we adapt the spec for an environment where processes may fail?
RELIABLE BROADCAST

Validity  If the sender is correct and broadcasts a message $m$, then all correct processes eventually deliver $m$

Agreement If a correct process delivers a message $m$, then all correct processes eventually deliver $m$

Integrity Every correct process delivers at most one message, and if it delivers $m \neq SF$, then some process must have broadcast $m$
**Terminating Reliable Broadcast**

<table>
<thead>
<tr>
<th>Validity</th>
<th>If the sender is correct and broadcasts a message $m$, then all correct processes eventually deliver $m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agreement</td>
<td>If a correct process delivers a message $m$, then all correct processes eventually deliver $m$</td>
</tr>
<tr>
<td>Integrity</td>
<td>Every correct process delivers at most one message, and if it delivers $m \neq SF$, then some process must have broadcast $m$</td>
</tr>
<tr>
<td>Termination</td>
<td>Every correct process eventually delivers some message</td>
</tr>
</tbody>
</table>
CONSENSUS

Validity
If all processes that propose a value propose \( v \), then all correct processes eventually decide \( v \)

Agreement
If a correct process decides \( v \), then all correct processes eventually decide \( v \)

Integrity
Every correct process decides at most one value, and if it decides \( v \), then some process must have proposed \( v \)

Termination
Every correct process eventually decides some value
Properties of \texttt{send(m)} and \texttt{receive(m)}

Benign failures:

**Validity**
If $p$ sends $m$ to $q$, and $p, q$ and the link between them are correct, then eventually receives $m$.

**Uniform**
For every message $m, q$ receives $m$ at most once from $p$, and only if $p$ sent $m$ to $q$.

* A property is called uniform if it applies to both correct and faulty processes.
MODEL

- Synchronous message passing
  - Execution is a sequence of rounds
  - In each round every process takes a step
    - sends messages to neighbors
    - receives messages send in that round
    - changes its state
- Network is fully connected
- No communication failures
A simple consensus algorithm

Process $p_i$: 
Initially $V = \{v_i\}$

To execute $\text{propose}(v_i)$:
1. Send $\{v_i\}$ to all

$\text{decide}(\ )$ occurs as follows:
2. for all $j, 0 \leq j \leq n + 1, j \neq i$, do
3. receive $S_j$ from $p_j$
4. $V := V \cup S_j$
5. decide $\min(V)$
What should $p_3$ decide at the end of the round?
What should $p_3$ decide at the end of the round?
A process that receives a proposal in round 1, relays it to others during round 2.

Suppose \( p_3 \) hasn’t heard from \( p_2 \) at the end of round 2. Can \( p_3 \) decide?
What is going on

A correct process $p$ has not received all proposals by the end of round $i$. Can $p$ decide?

Another process may have received the missing proposal at the end of round $i$ and be ready to relay it in round $i + 1$. 
Dangerous chains

Dangerous chain

The last process in the chain is correct, all others faulty
Living dangerously

How many rounds can a dangerous chain span?

- $f$ faulty processes
- At most $f + 1$ nodes in the chain
- Spans at most $f$ rounds

It is safe to decide by the end of round $f + 1$!
The algorithm

Process $p_i$:

Initially $V = \{v_i\}$

To execute $\text{propose}(v_i)$:

round $k, 1 \leq k \leq f + 1$

1. Send $\{v \in V: p_i \text{ has not already sent } v\}$ to all
2. for all $j, 0 \leq j \leq n + 1, j \neq i$, do
3. receive $S_j$ from $p_j$
4. $V := V \cup S_j$

$\text{decide}(\ )$ occurs as follows:

5. if $k = f + 1$
6. decide $\min(V)$