Objectives:

What is the weakest failure detector to emulate a shared memory?

Model:
- Message passing system with reliable channels
- Crash failures
Why important?:

Can find the weakest failure detector
to implement consensus with a majority of faulty process
Why important?:

Can find the **weakest failure detector**

to implement consensus with a majority of
faulty process (up to n-1)

\[ S \geq \Omega(\diamond S) \]

Can tolerate n-1 failures  Can tolerate n/2 failures
Shared Memory vs. Message Passing

Why important?:

Can find the weakest failure detector to implement consensus with a majority of faulty process (up to n-1)

Can tolerate n-1 failures

\[ S \geq ? \geq \Omega(\Diamond S) \]

Can tolerate n-1 failures

Can tolerate n/2 failures
weakest failure detector:
If we can find the weakest failure detector to emulate registers, say $F$, then the failure detector class $F \times \Omega$ is the weakest for consensus.
weakest failure detector:
If we can find the weakest failure detector to emulate registers, say $F$, then the failure detector class $F \times \Omega$ is the weakest for consensus.

Implementing consensus
Consensus can be implemented using registers and $\Omega$, in every environment (tolerate up to $n-1$ failures)
weakest failure detector:
If we can find the weakest failure detector to emulate registers, say $E$, then the failure detector class $E \times \Omega$ is the weakest for consensus.

Weakest?
For any failure detector class $D$ that implements consensus, 1). $D \geq \Omega$ (proved in the class)
Using consensus as a building block, we can implement register by $D$. Thus 2). $D \geq F$. 
Quorum failure detector, $\Sigma$:

Outputs a list of trusted processes

Properties satisfied with $\Sigma$:

1. Intersection:
   Any two outputs at any time, for any process includes at least one same process.

2. Completeness:
   Eventually no faulty process is ever trusted by any correct process.
Emulate SWSR register using $\textit{\Sigma}$:

Atomic actions in the model:

1. Receive from other processes
2. Query the failure detector
3. State change and send to other processes

Local vars for each process

- $\textit{current}$: current value of emulated register
- $\textit{last_write}$: keep track of the time stamp for the current value (initially set to -1)
Emulate SWSR register using $\Sigma$:

Initially $seq = 0$
Emulate SWSR register using $\Sigma$:

Initially $\text{seq} = 0$

- Initially $\text{seq} = 0$

- If $\text{seq} \geq \text{last_write}$ then
  - $\text{current} := \text{v}$
  - $\text{last_write} := \text{seq}$
  - Send $(\text{ACK\_WRITE, seq})$
Emulate SWSR register using $\Sigma$:

Initially $\text{seq} = 0$

Wait until it receives ACK from all trusted processes, and $\text{seq}++$

Trusted processes would change before write terminates
Emulate SWSR register using $\sum$:

Initially $rc = 0$ (read counter)

$rc := rc + 1$

send $(ACK\_READ, last\_write, current, rc)$
Emulate SWSR register using $\Sigma$:

Initially $rc = 0$ (read counter)

Wait until it receives ACK from all trusted processes, $mlw=$(max of second field of ACK)

if $mlw$>last_write then
  \[ \text{current := (third field of ACK)} \]
  \[ \text{last_write := } mlw \]

return $current$
Correctness:

**Assertion 1.**

if $Pw$ has not finished its $k$-th writing, then for all processes, $last\_write \leq k$

**Termination for write**

from *completeness* of $\sum$

Eventually, $\sum$ outputs only correct processes.

From assertion 1, all correct processes acknowledge.

Termination for read is similar.
Correctness:

Assertion 2.
If any process sends \((ACK\_READ, s, v, *)\),
then \(v\) is the value of the \(s\)-th write operation.

Validity
Have to show: every read operation returns either
the value written by the last write that precedes it,
or a value written concurrently with this read.
(If there is no overlapping read/write, read should
return the last value written.)
Validity

$P_i$ is in $L_w \cap L_r$ because of the intersection property of $\sum$

Waited for $L_w$

Waited for $L_r$

j-th READ

k-th WRITE

(k+1)-th WRITE
Validity

\( Pi \) is in \( Lw \cap Lr \) because of the intersection property of \( \Sigma \)

\((ACK\_READ, s, *, j) \) by \( i \)

Waited for \( Lr \)

\((ACK\_WRITE, k) \) by \( i \)

Waited for \( Lw \)

j-th READ

k-th WRITE

(k+1)-th WRITE
Validity

\( Pi \) is in \( Lw \cap Lr \) because of the intersection property of \( \Sigma \)

\[ s \geq k \]

(ACK\_READ, \( s, *, j \)) by i

(ACK\_WRITE, \( k \)) by i

Waited for \( Lw \)

Waited for \( Lr \)

j-th READ

\( (k+1) \)-th WRITE
Validity

\( P_i \) is in \( L_w \cap L_r \) because of the intersection property of \( \Sigma \).

\[ s \geq k \]

\[ (ACK\_READ, s, *, j) \text{ by } i \]

\[ (ACK\_WRITE, k) \text{ by } i \]

Waited for \( L_w \)

\[ \text{Waited for } L_r \]

\[ m_{lw} \geq s \geq k \]
Shared Memory vs. Message Passing

Validity

\( Pi \) is in \( Lw \cap Lr \) because of the intersection property of \( \Sigma \)

\[ s \geq k \]

\[ mlw \geq k \]

(\( ACK\_READ, s, *, j \)) by \( i \)

(\( ACK\_WRITE, k \)) by \( i \)

Waited for \( Lw \)

Waited for \( Lr \)

\( mlw \)
Validity

\( P_i \) is in \( Lw \cap Lr \) because of the intersection property of \( \Sigma \)

\[ s \geq k \]

\[ mlw \geq k \]

(ACK_READ, s, *, j) by i

Waited for \( Lr \)

If the \( k+1 \)th write has not started, then all processes have a last_write <= k

=> for any (ACK_READ, x, *, j), x <= k

(ACK_WRITE, k) by i

Waited for \( Lw \)

k-th WRITE

j-th READ

mlw
Validity

\( Pi \) is in \( Lw \cap Lr \) because of the intersection property of \( \Sigma \)

\[
\begin{align*}
  s & \geq k \\
  mlw & \geq k
\end{align*}
\]

\((ACK\_WRITE, k)\) by \( i \)

Waited for \( Lw \)

\((ACK\_READ, s, *, j)\) by \( i \)

Waited for \( Lr \)

\( mlw \)

If the \( k+1^{th} \) write has not started, then all processes have a \( \text{last\_write} \leq k \)

\[
\Rightarrow \text{for any } (ACK\_READ, x, *, j), \quad x \leq k
\]

\[
\Rightarrow mlw = k, \text{ which implies } j\text{-th read returns the value written by the } k\text{-th write}
\]
Correctness:

Ordering:

Have to show: if a read operation $r$ precedes a read operation $r'$, then $r'$ cannot return a value written before the value returned by $r$.

Proof sketch:

last_write for a reader makes sure consistency.
Have to show that $\sum$ is the weakest.
=> have to emulate $\sum$ using a failure detector that implements register.
Have to show that $\sum$ is the weakest.

=>$\sum$ must be emulated using a failure detector that implements register.

Proof: not for this time...

Summary: we showed $\sum$ is the weakest failure detector to implement register.

Can tolerate $n-1$ failures

$S \geq \Omega \times \sum \geq \Omega(\Diamond S)$

Can tolerate $n-1$ failures  Can tolerate $n/2$ failures