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

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

$$S \quad \geq \quad ? \quad \geq \quad \Omega(\Diamond S)$$

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* x Ω is the weakest for consensus.

Shared Memory vs. Message Passing

<u>weakest failure detector:</u>

If we can find the weakest failure detector to emulate registers, say <u>*F*</u>, then the failure detector class <u>*F* x Ω</u> is the weakest for consensus.

<u>Implementing consensus</u>

Consensus can be implemented using registers and Ω, in every environment (tolerate up to n-1 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?

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, Σ:

Outputs a list of *trusted* processes

Properties satisfied with Σ:

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 Σ:

Atomic actions in the model:

1. Receive from other processes

2. Query the failure detector

3. State change and send to other processes

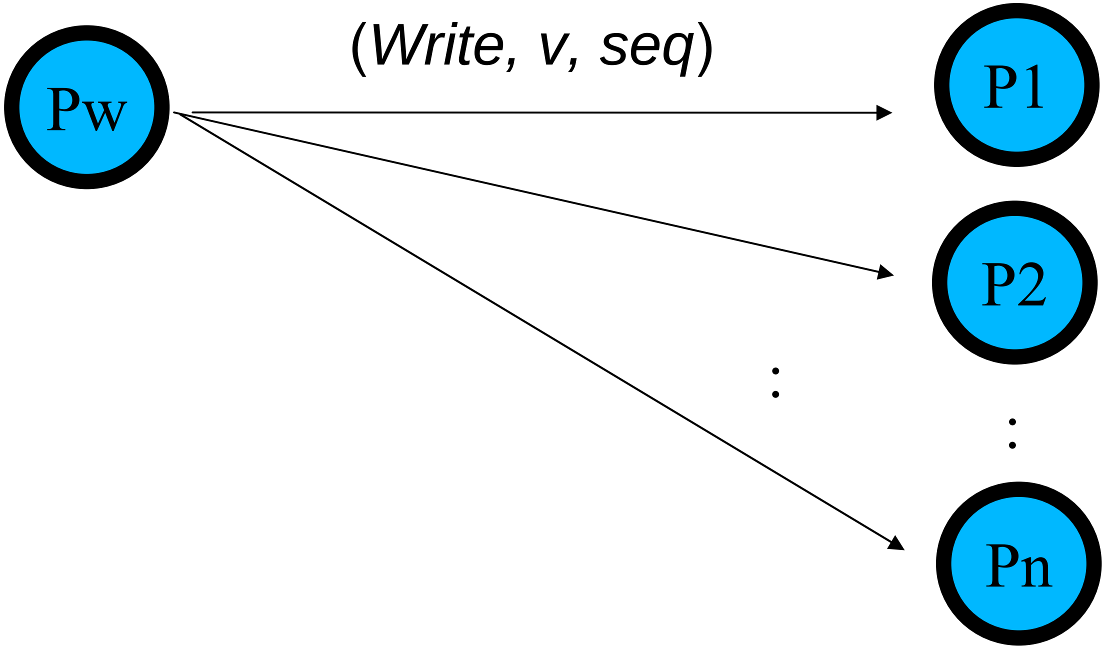
Local vars for each process

current : current value of emulated register

last_write : keep track of the time stamp for the current value (initially set to -1)

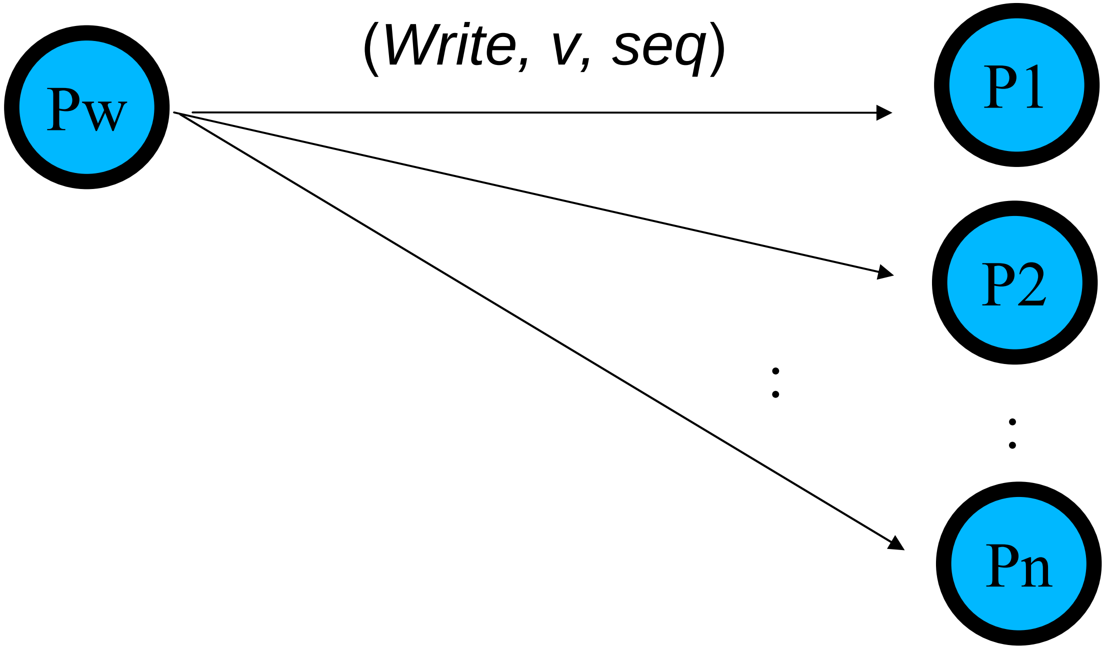
Emulate SWSR register using ∑:

Initially *seq* = 0



Emulate SWSR register using ∑:

Initially *seq* = 0

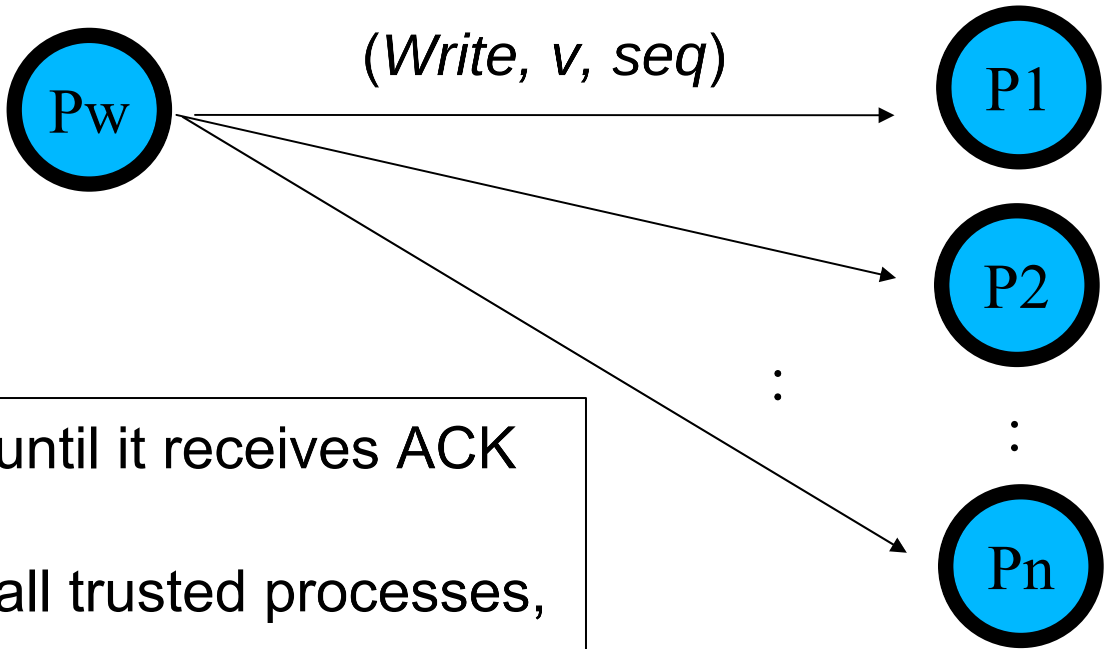


If *seq* >= *last_write* **then**
current:=v
last_write:=seq
send (*ACK_WRITE,seq*)

Shared Memory vs. Message Passing

Emulate SWSR register using ∑:

Initially *seq* = 0



If *seq* > *last_write* **then**
current:=*v*
last_write:=*seq*
send (*ACK_WRITE*,*seq*)

Wait until it receives ACK

from all trusted processes,

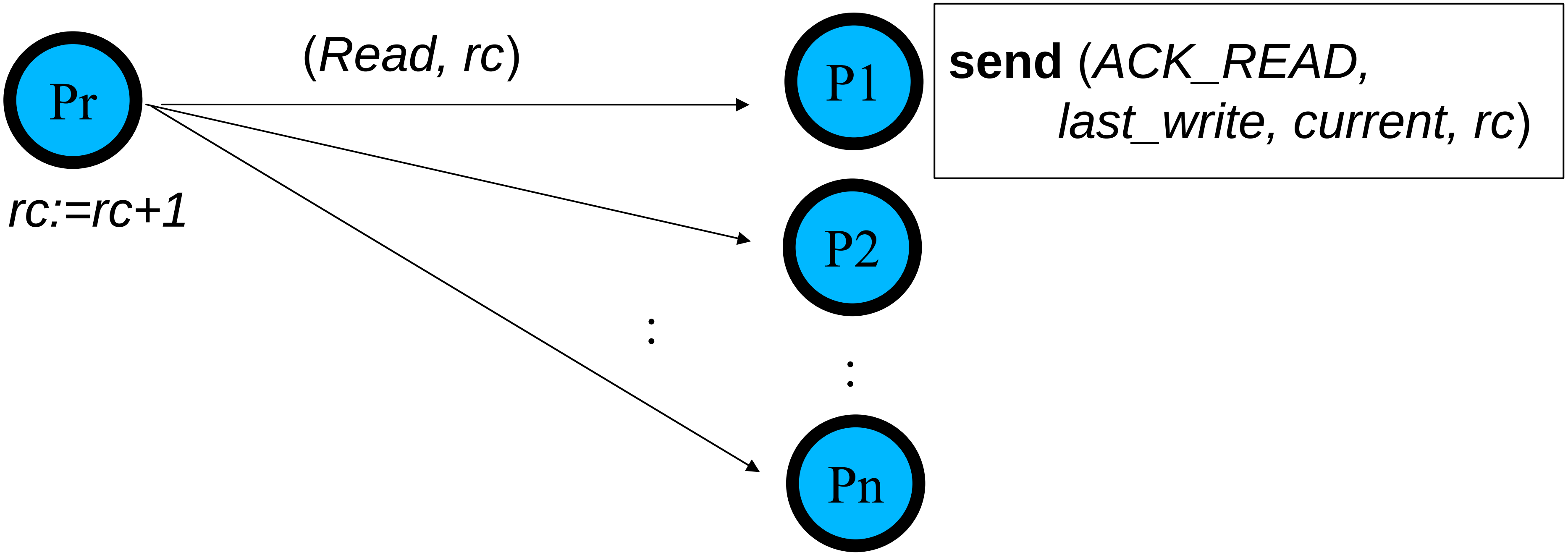
and *seq*++

Trusted processes would change before write terminates

Shared Memory vs. Message Passing

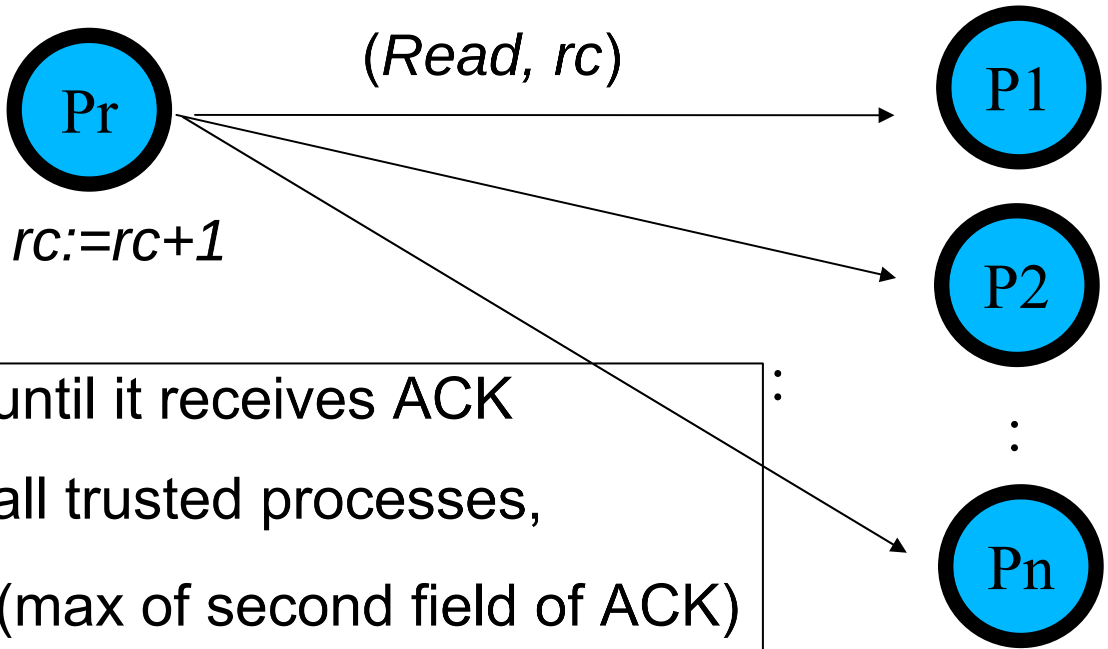
Emulate SWSR register using ∑:

Initially *rc* = 0 (read counter)



Emulate SWSR register using ∑:

Initially *rc* = 0 (read counter)



send (*ACK_READ,*
last_write, current, rc)

Wait until it receives ACK
from all trusted processes,
mlw=(max of second field of ACK)
if *mlw*>last_write **then**
current := (third field of ACK)
last_write := *mlw*
return *current*

Correctness:

Assertion 1.

if *Pw* has not finished its *k*-th writing, then

for all processes, *last_write* <= *k*

Termination for write

from *completeness* of ∑

Eventually, ∑ 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,

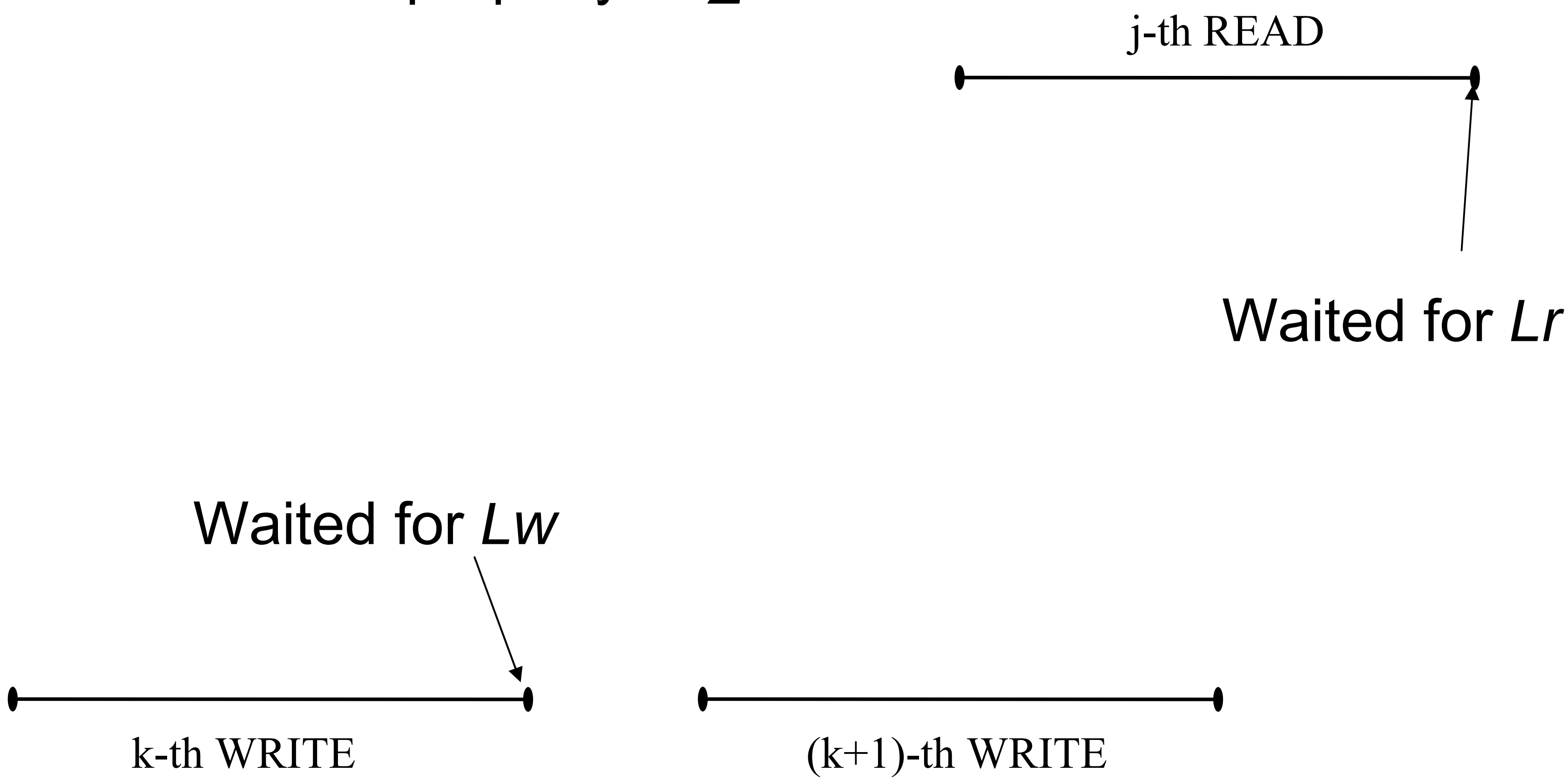
or a value written concurrently with this read.

(If there is no overlapping read/write, read should

return the last value written.)

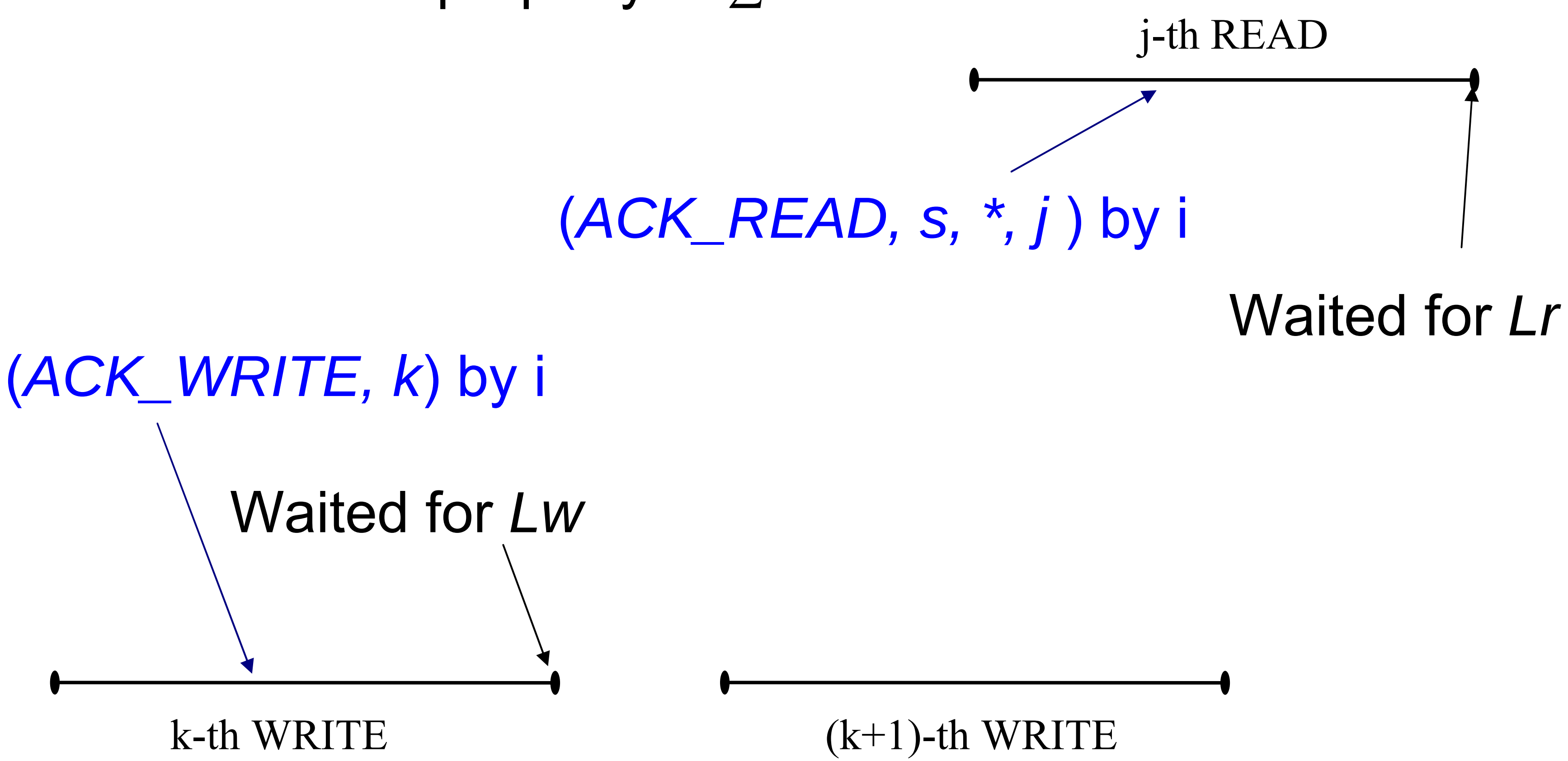
Validity

Pi is in $Lw \cap Lr$ because of the intersection property of Σ



Validity

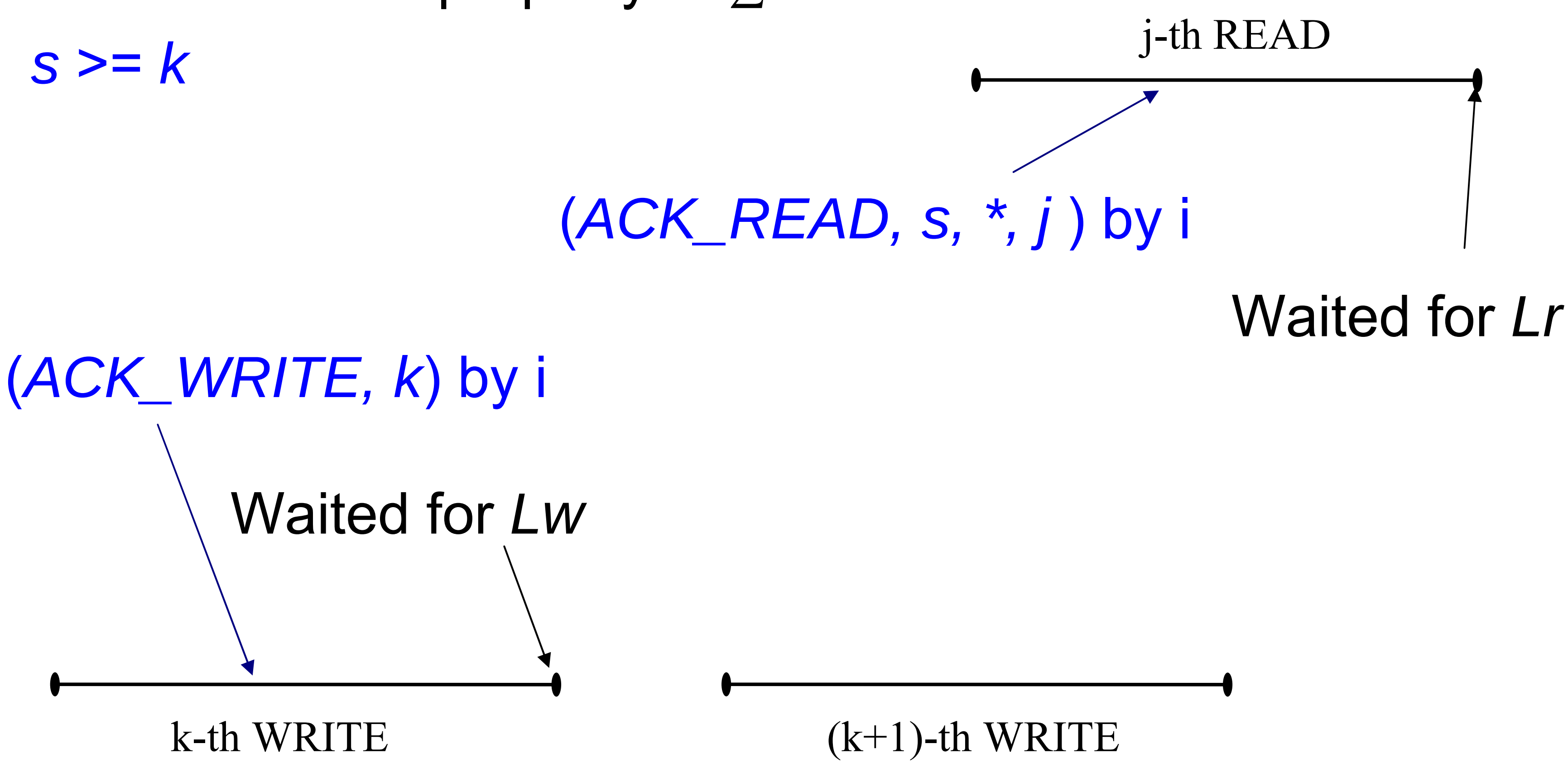
Pi is in $Lw \cap Lr$ because of the intersection property of Σ



Validity

Pi is in *Lw*∩*Lr* because of the intersection property of ∑

$s >= k$

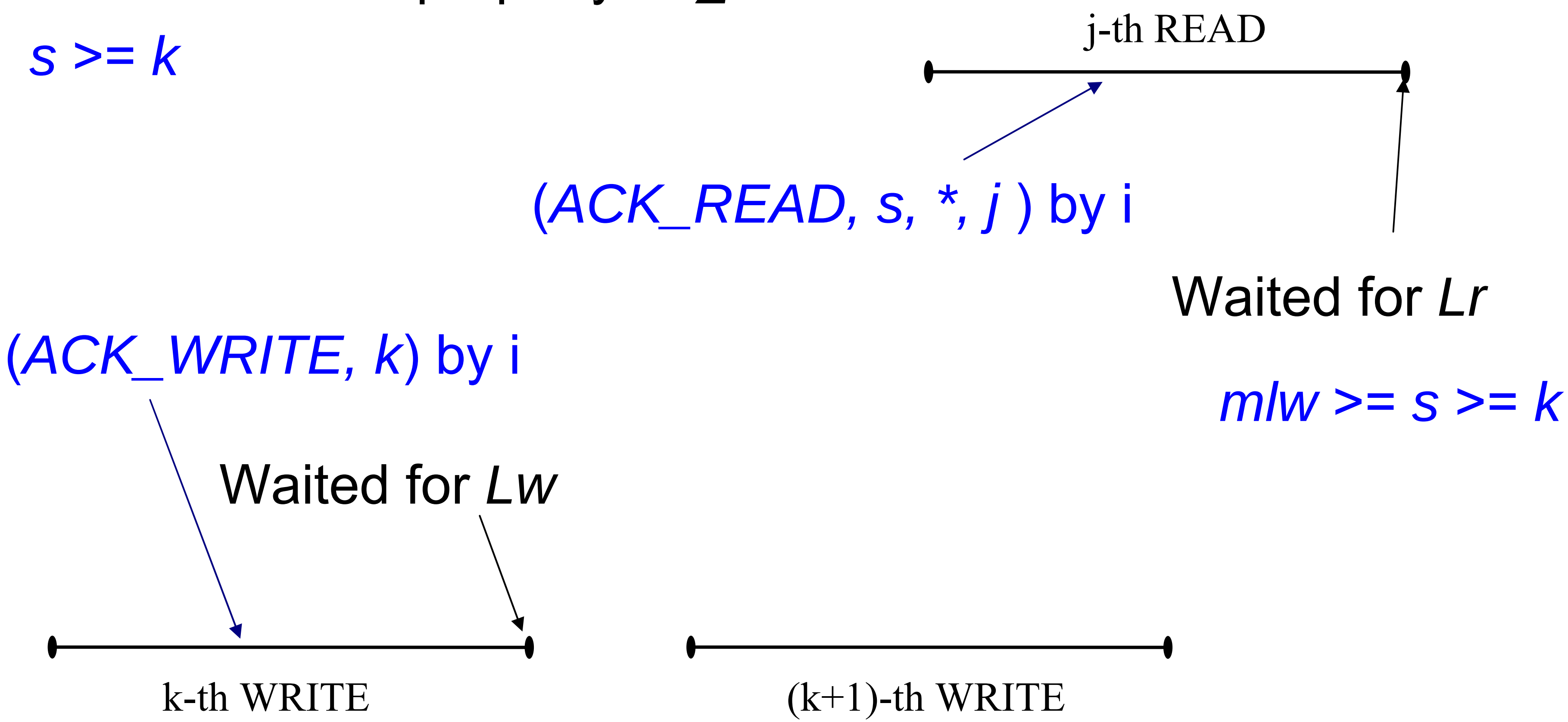


Shared Memory vs. Message Passing

<u>Validity</u>

Pi is in $Lw \cap Lr$ because of the intersection property of Σ

$s >= k$

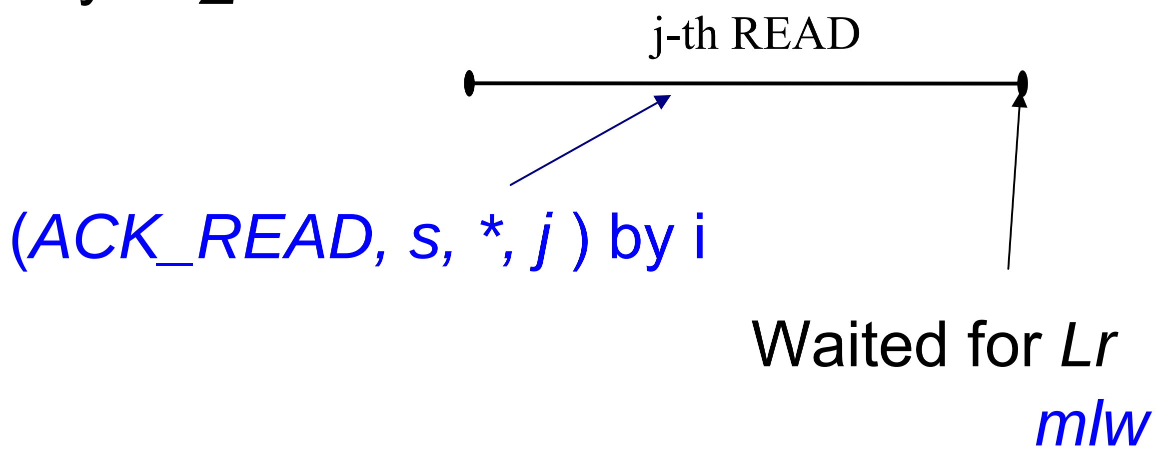


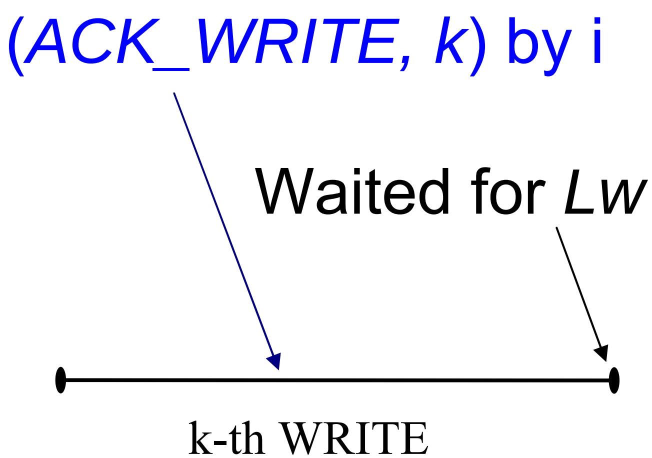
Validity

Pi is in $Lw \cap Lr$ because of the intersection property of Σ

s >= k

mlw >= k



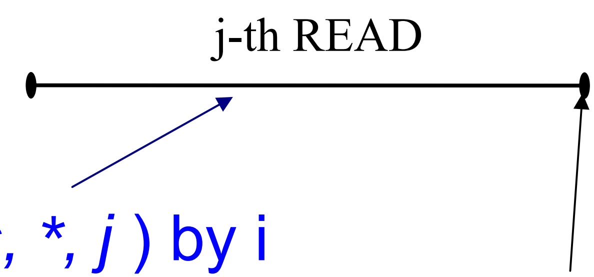


<u>Validity</u>

Pi is in *Lw*∩*Lr* because of the intersection property of ∑

s >= *k*

mlw >= *k*

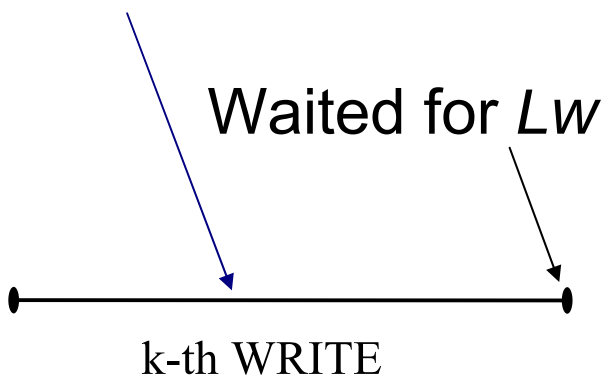


(*ACK_READ, s, *, j*) by i

Waited for *Lr*

mlw

(*ACK_WRITE, k*) by i



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

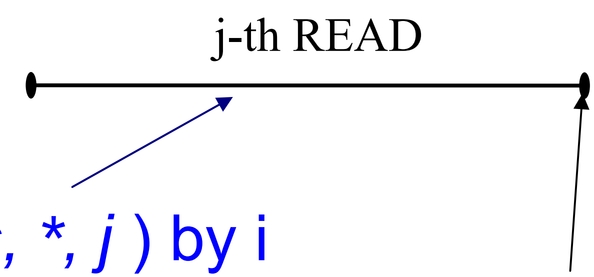
=> for any (ACK_READ, x, *, j), x <= k

Validity

Pi is in $Lw \cap Lr$ because of the intersection property of Σ

$s >= k$

$mlw >= k$



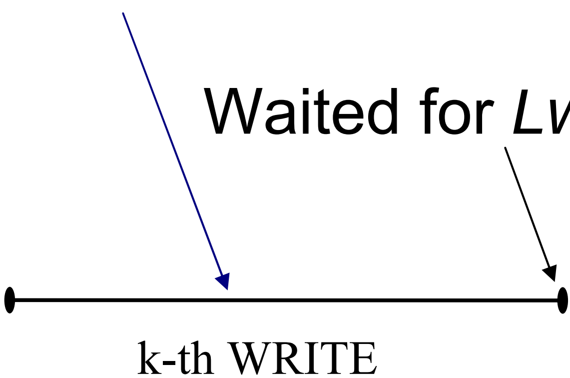
(*ACK_READ, s, *, j*) by i

Waited for *Lr*

mlw

(*ACK_WRITE, k*) by i

Waited for *Lw*



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

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

=> mlw=k, which implies
j-th read returns the value
written by the k-th write

<u>Correctness:</u>

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.

Shared Memory vs. Message Passing

Have to show that Σ is the weakest.

=> have to emulate Σ using a failure detector

that implements register.

Have to show that ∑ is the weakest.

=> have to emulate ∑ using a failure detector

that implements register.

Proof: not for this time...

Summary: we showed ∑ is weakest failure detector to implement register.

Can tolerate n-1 failures

$S \geq \Omega x \Sigma \geq \Omega(\Diamond S)$

Can tolerate n-1 failures

Can tolerate n/2 failures