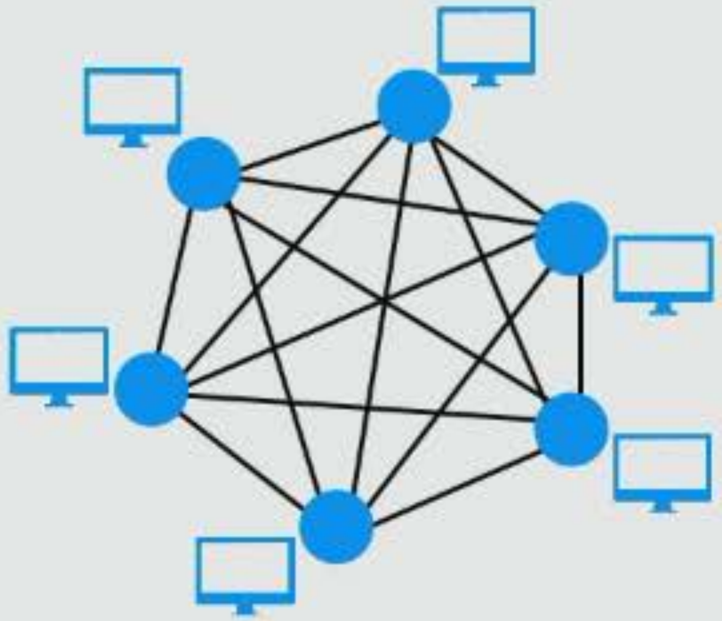
RDMA: Provably More Powerful Communication

Naama Ben-David (CMU)

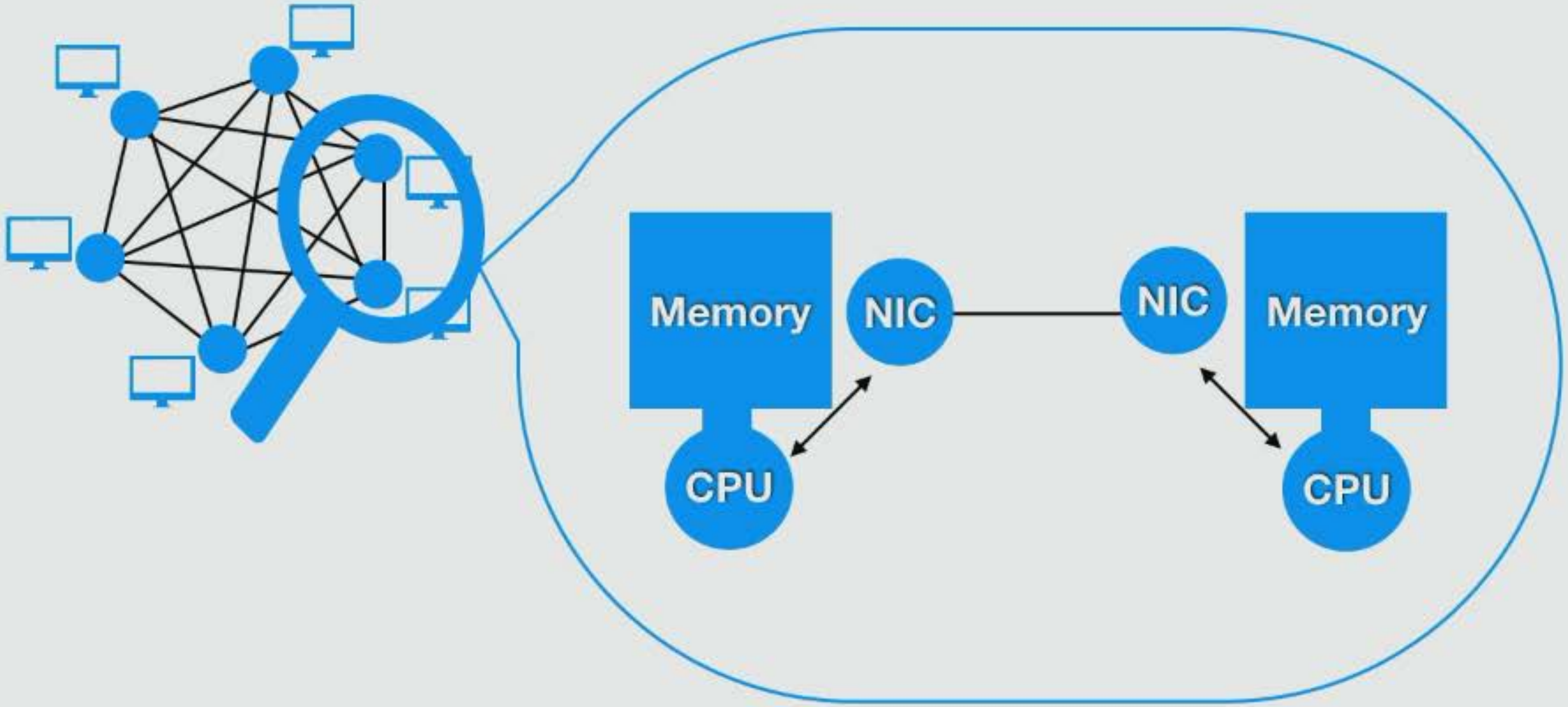
PODC’18, PODC’19

Marcos Aguilera, Irina Calciu, Rachid Guerraoui, Virendra Marathe,
Erez Petrank, Sam Toueg, Igor Zablotchi

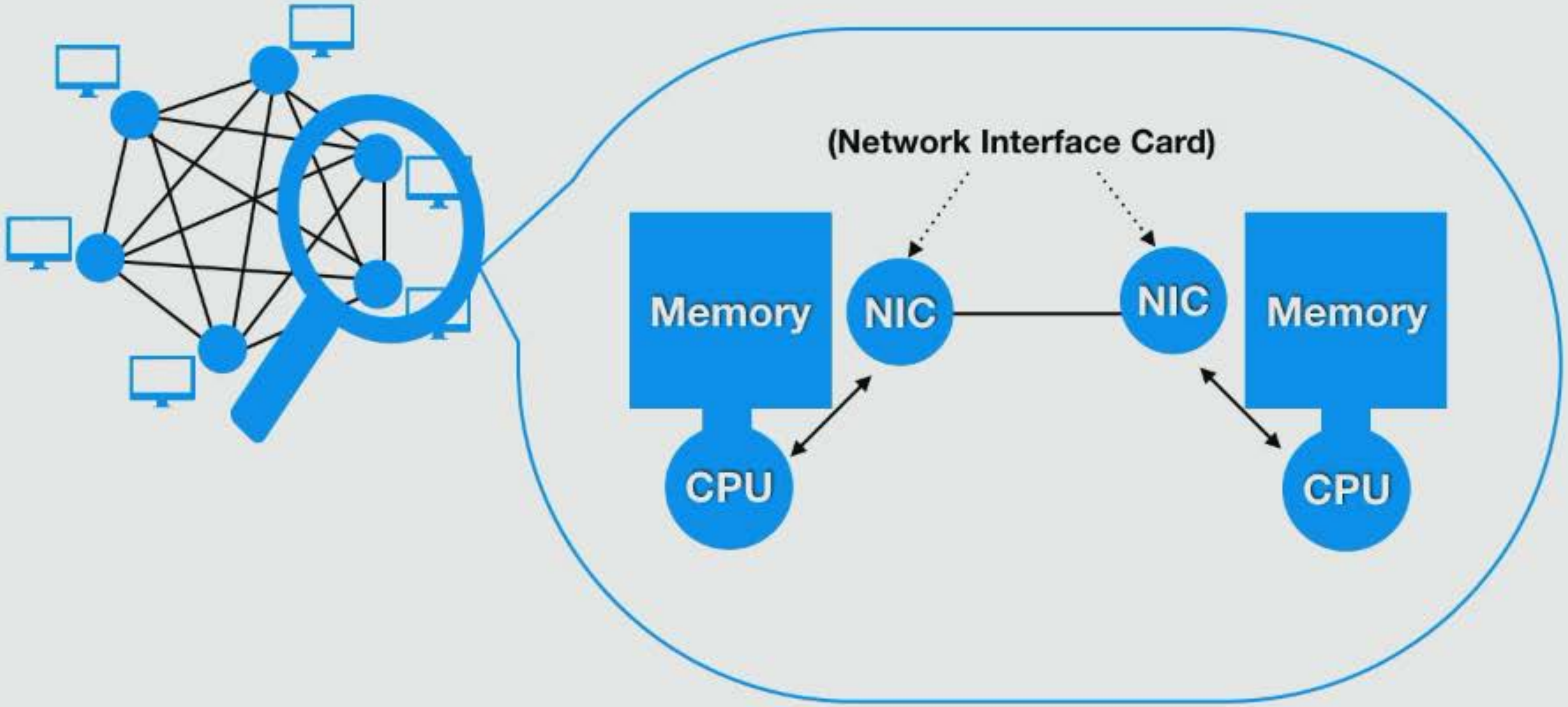
Data Center Technology: RDMA



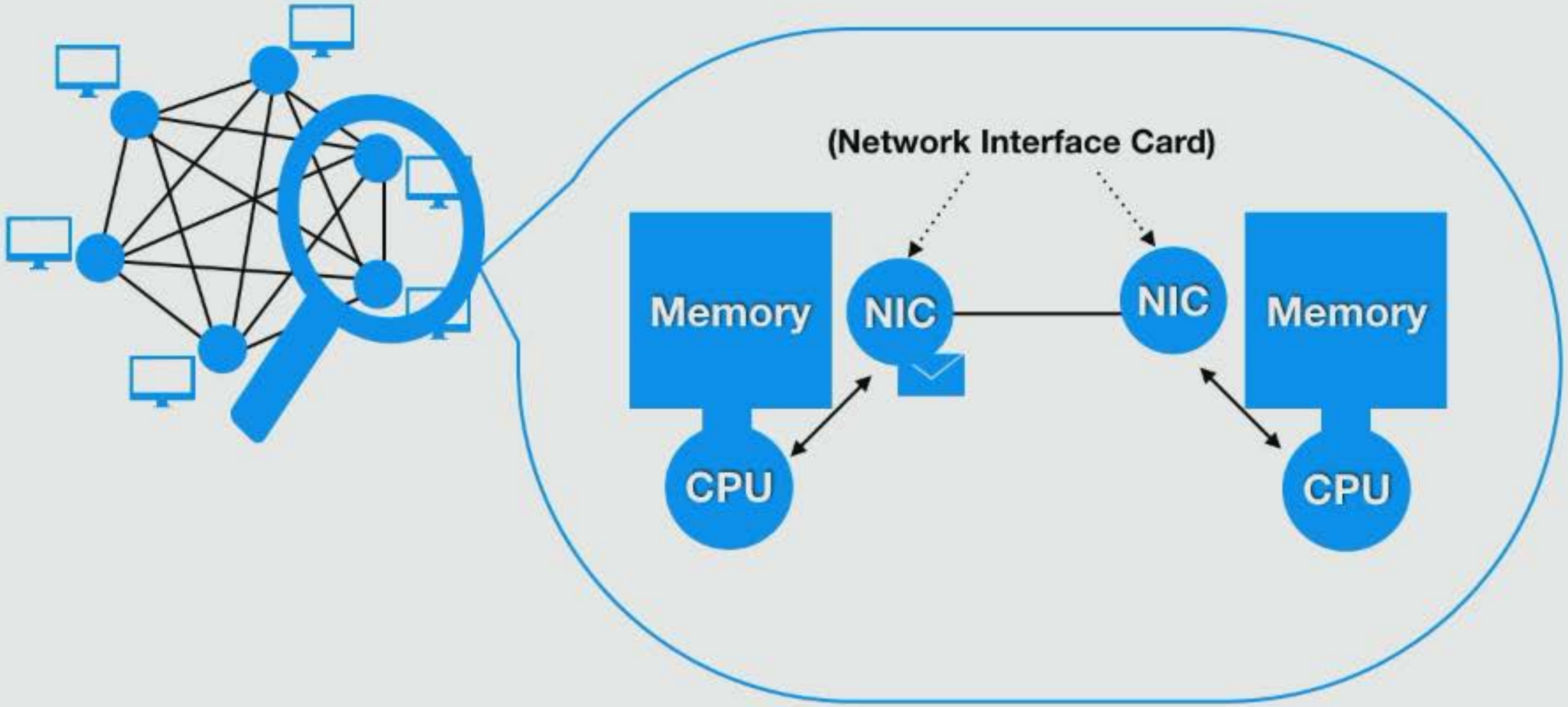
Data Center Technology: RDMA



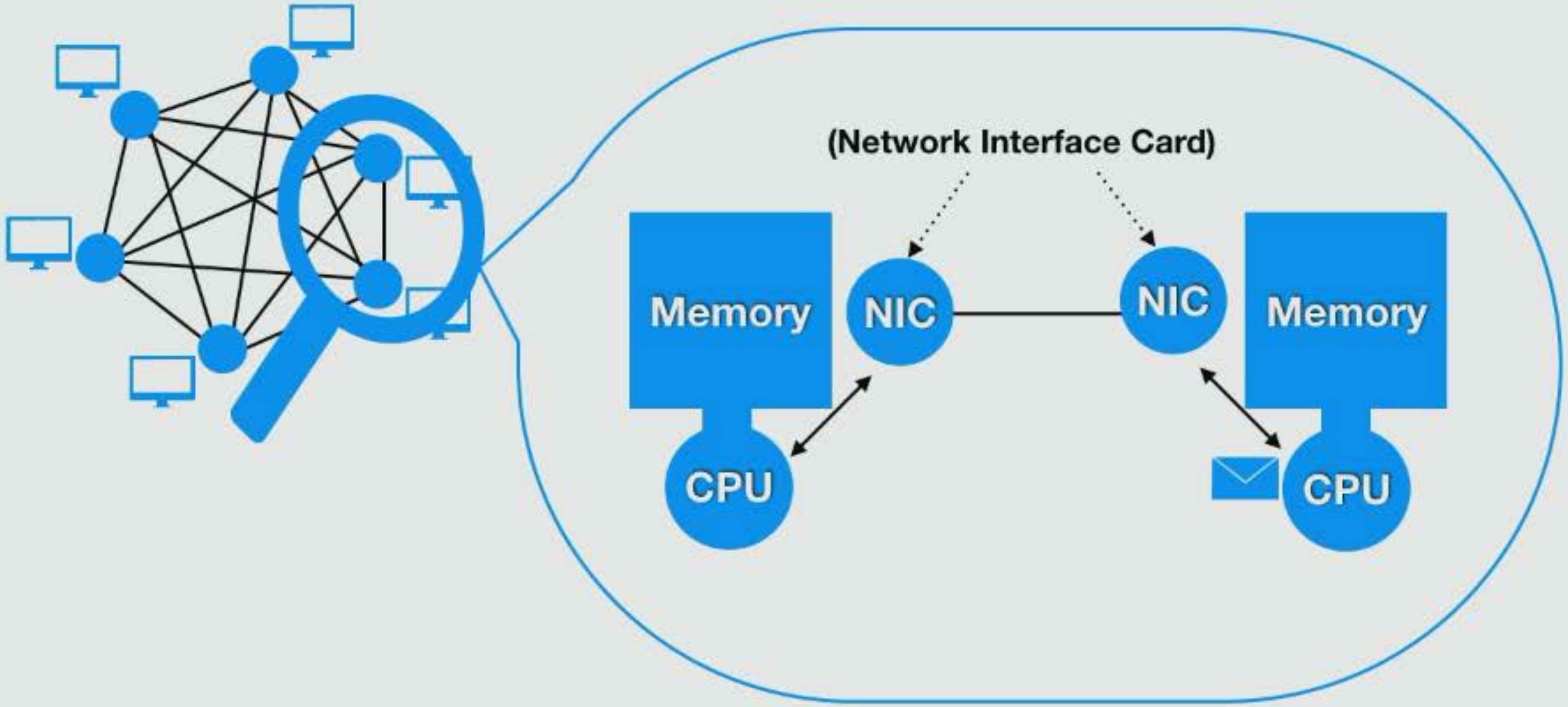
Data Center Technology: RDMA



Data Center Technology: RDMA

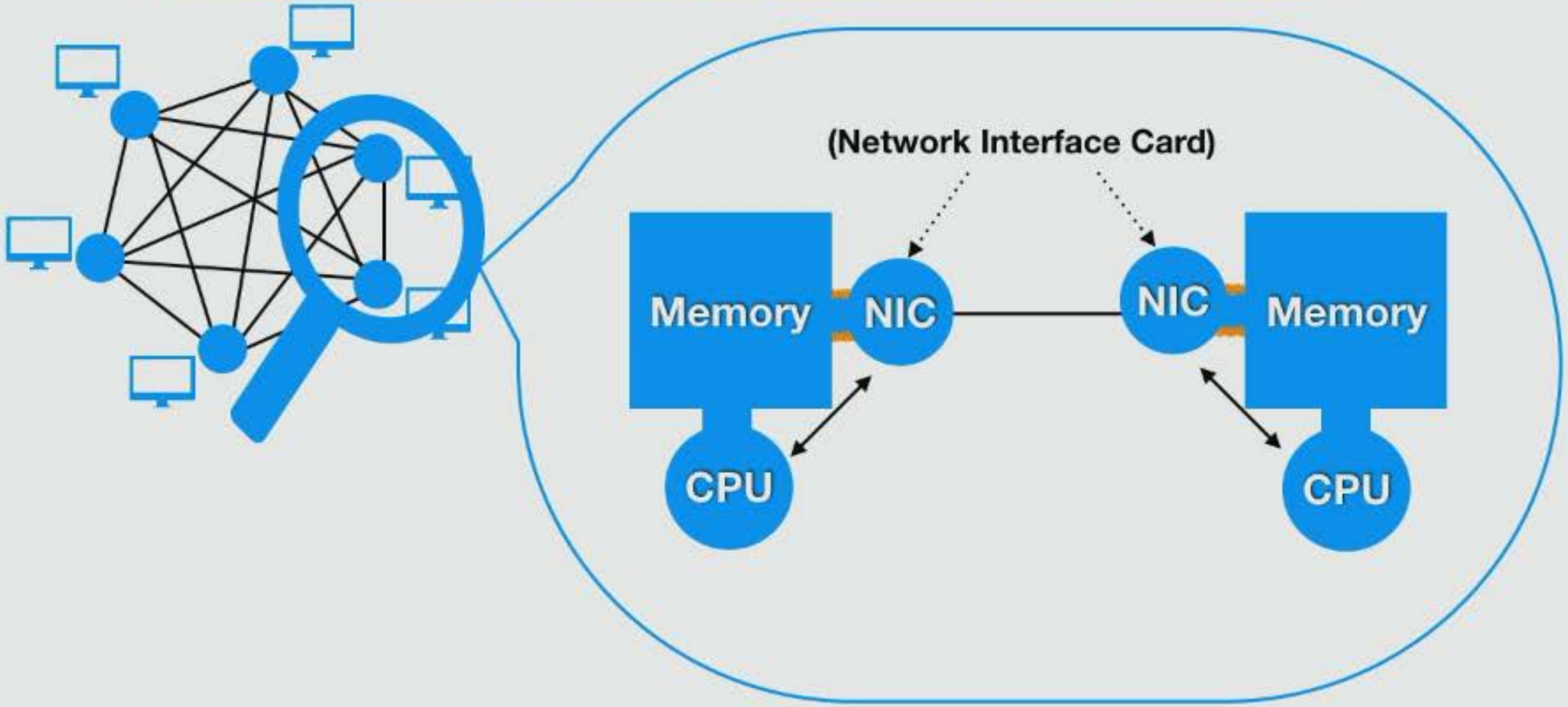


Data Center Technology: RDMA



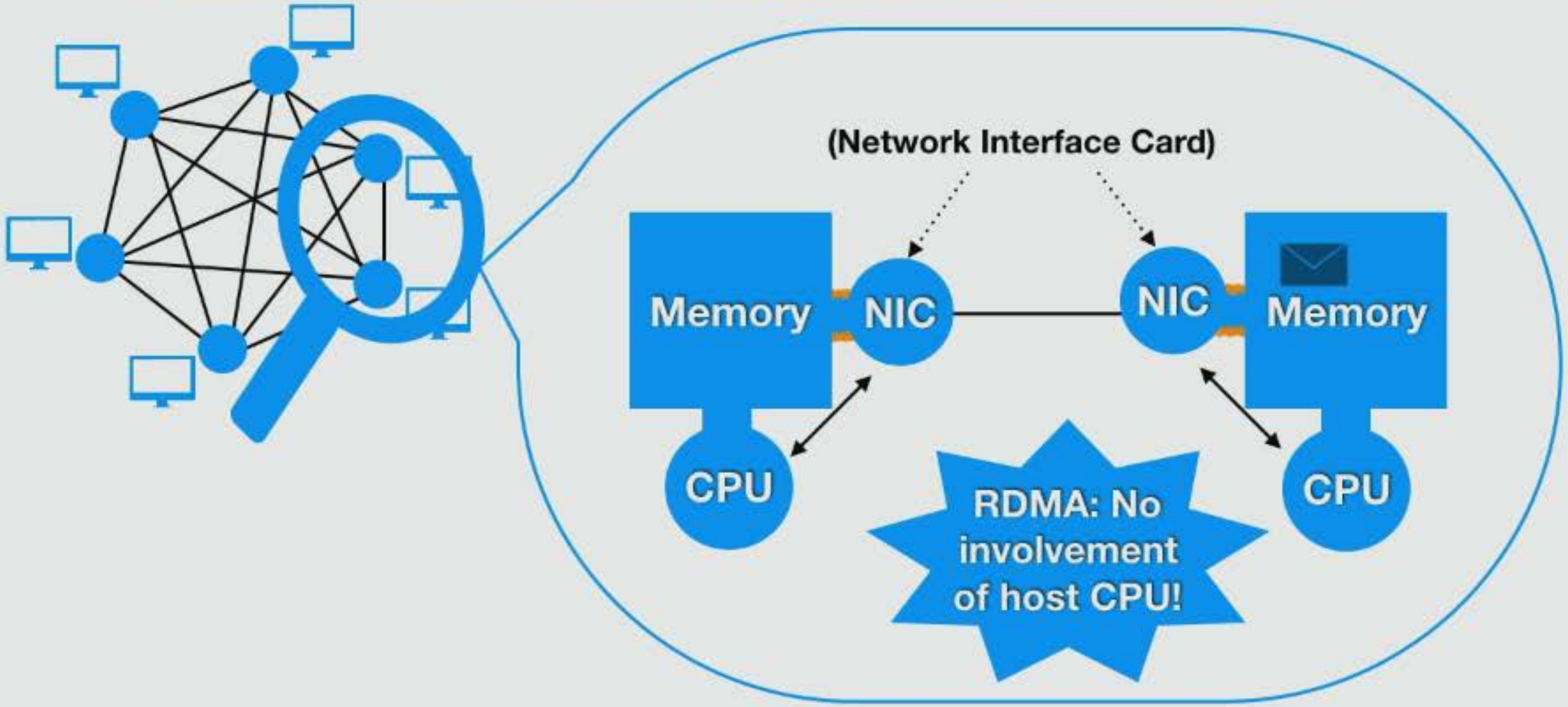
Data Center Technology: RDMA

Remote Direct Memory Access (RDMA)



Data Center Technology: RDMA

Remote Direct Memory Access (RDMA)

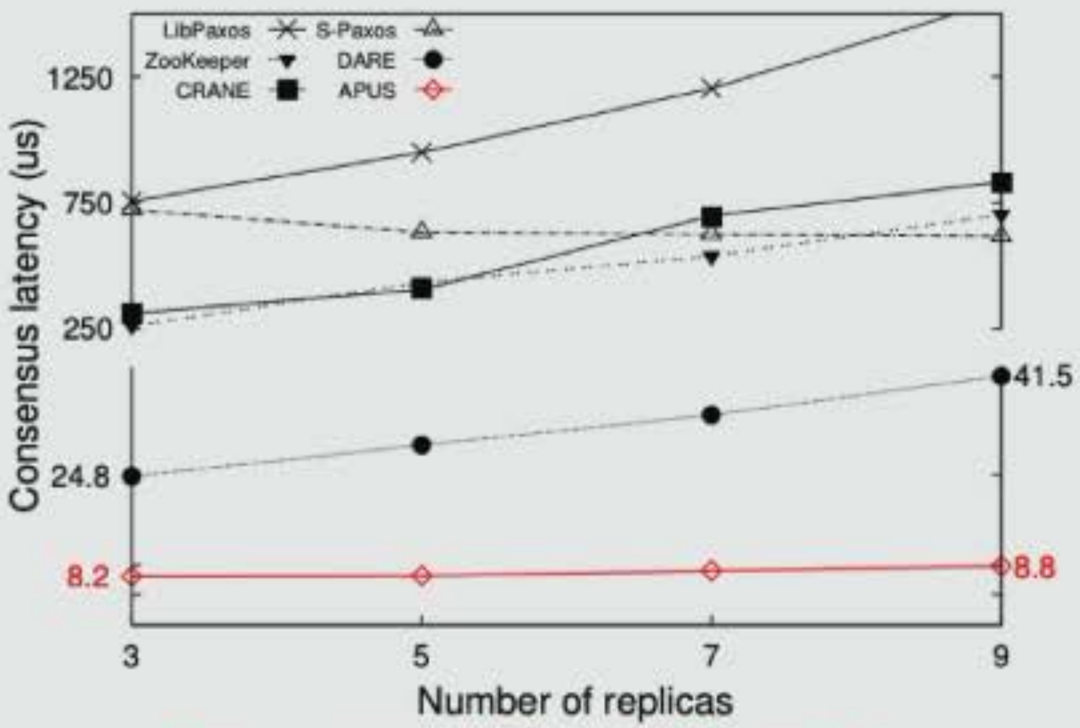


RDMA in Practice

- Huge speedups with RDMA

RDMA in Practice

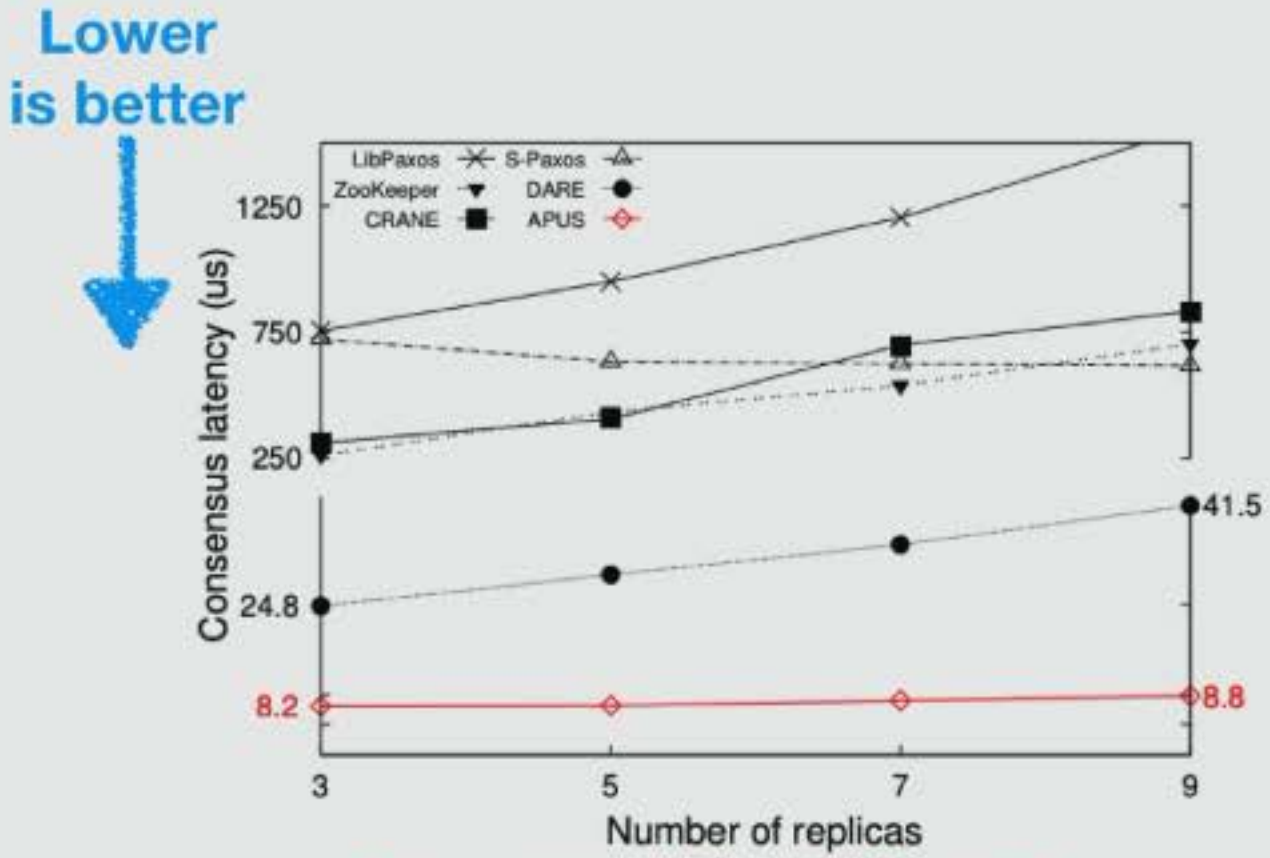
- Huge speedups with RDMA



[WangJaingChenYiCui'17]

RDMA in Practice

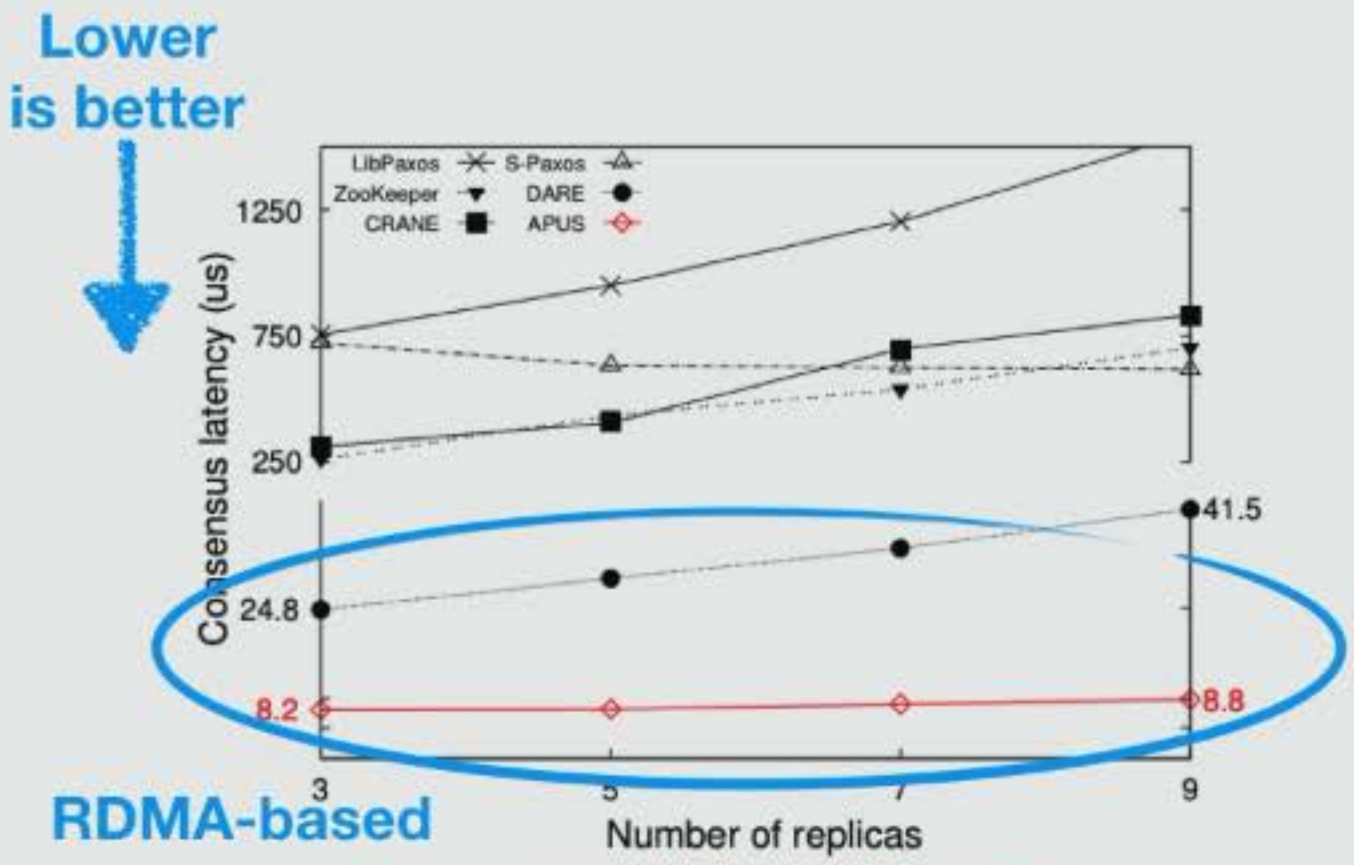
- Huge speedups with RDMA



[WangJaingChenYiCui'17]

RDMA in Practice

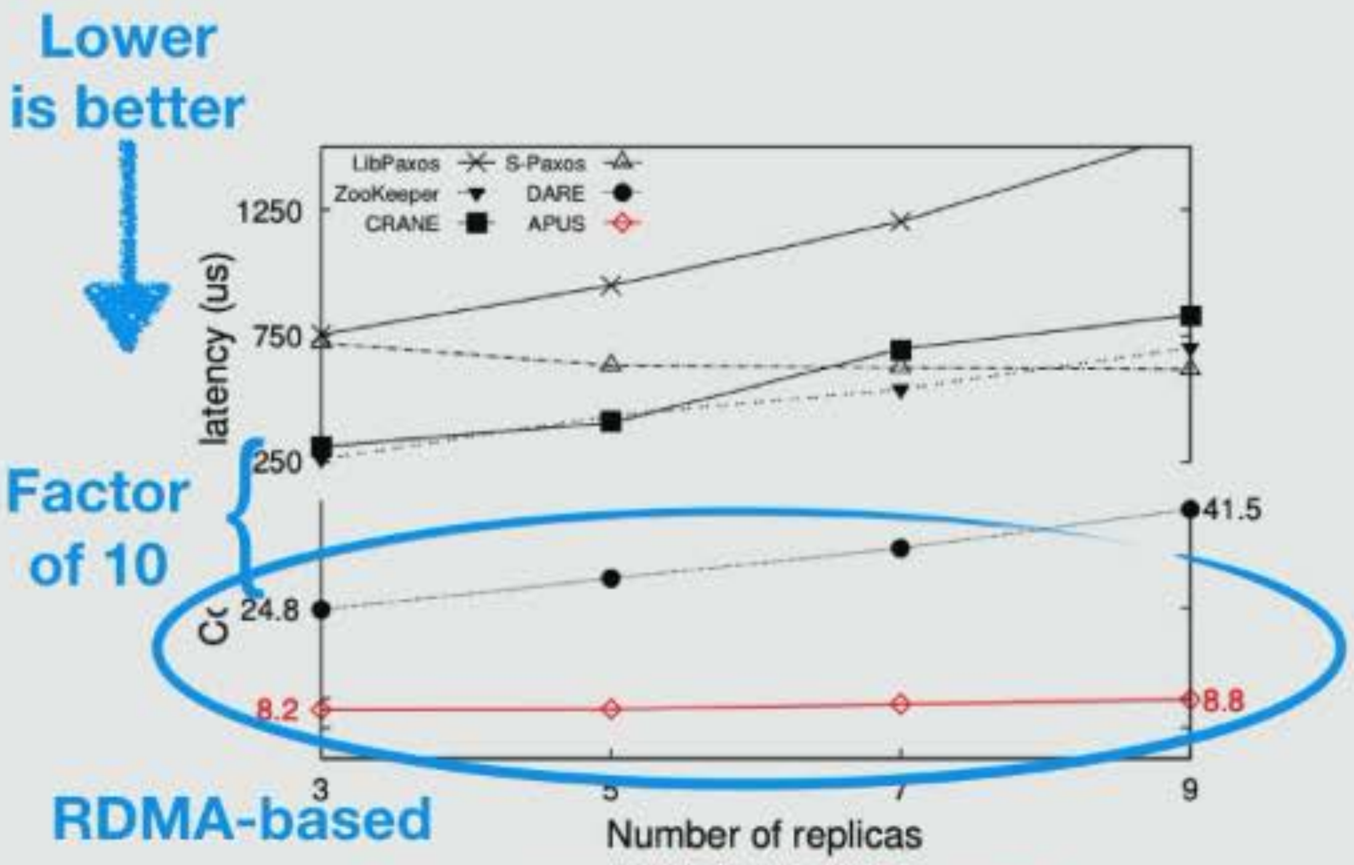
- Huge speedups with RDMA



[WangJaingChenYiCui'17]

RDMA in Practice

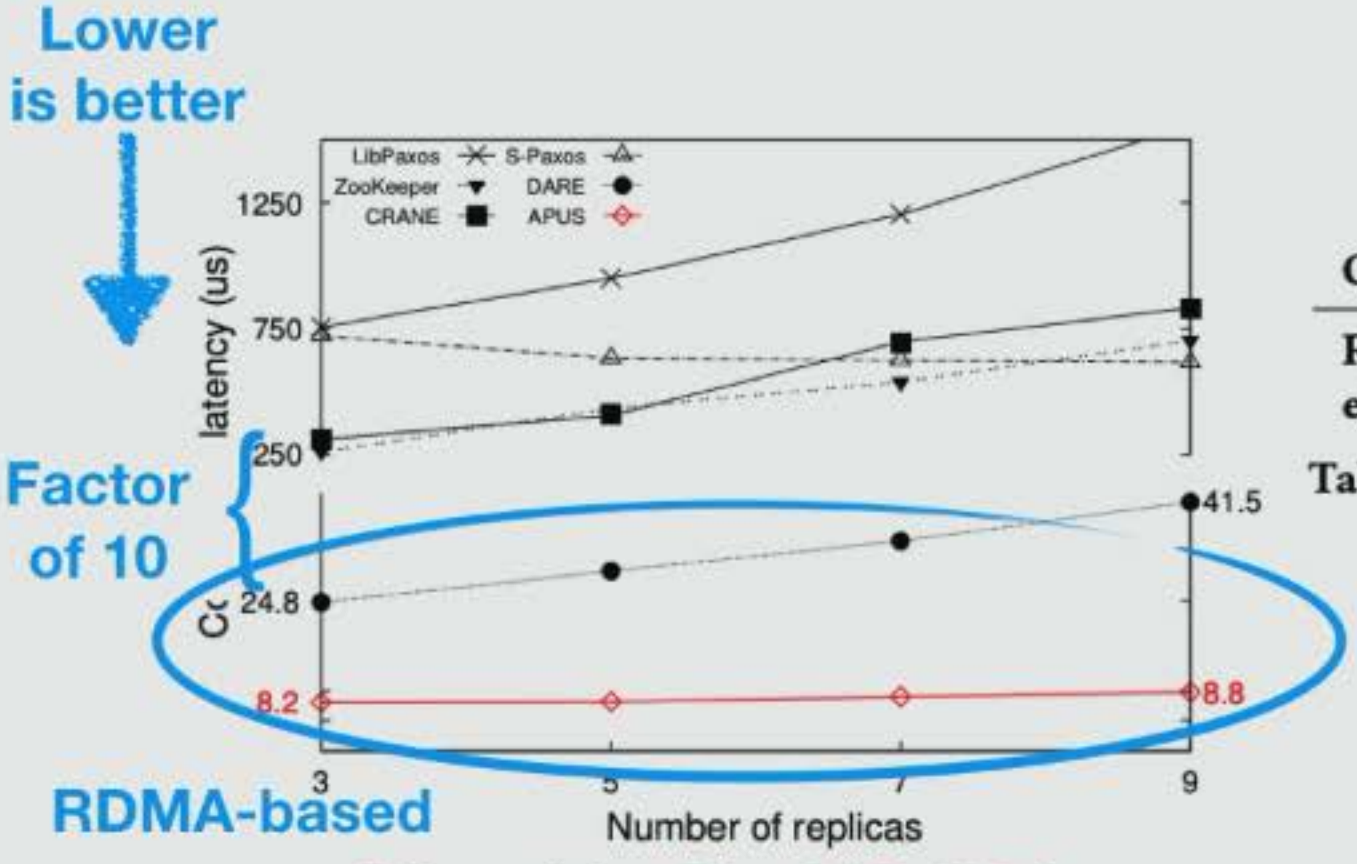
- Huge speedups with RDMA
- But is this improvement fundamental to RDMA?



[WangJaingChenYiCui'17]

RDMA in Practice

- Huge speedups with RDMA
- But is this improvement fundamental to RDMA?



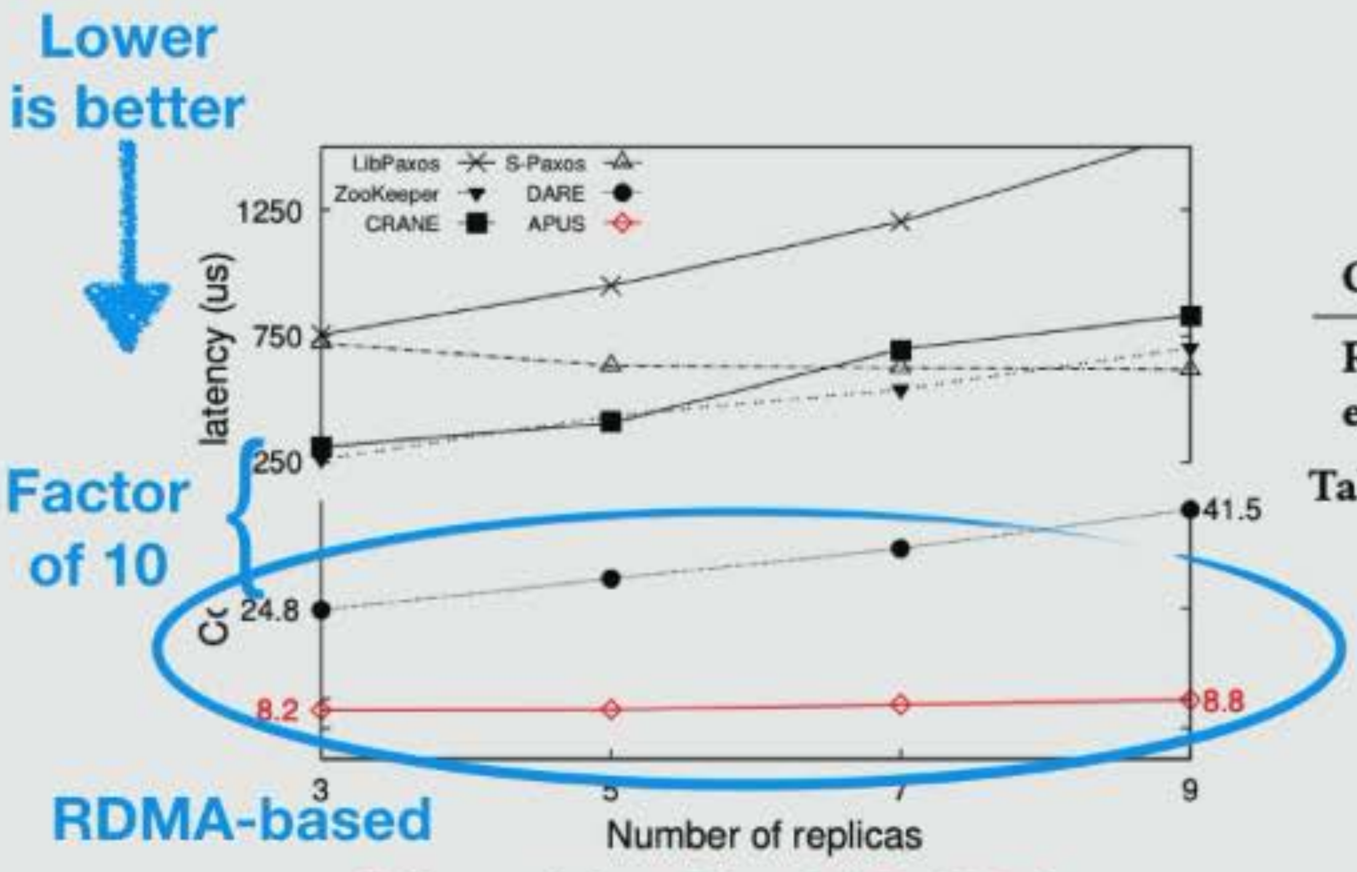
[WangJaingChenYiCui'17]

Cluster	CX3 (InfiniBand)	CX4 (Eth)	CX5 (Eth)
RDMA read	1.7 µs	2.9 µs	2.0 µs
eRPC	2.1 µs	3.7 µs	2.3 µs

Table 2: Comparison of median latency with eRPC and RDMA

RDMA in Practice

- Huge speedups with RDMA
- But is this improvement fundamental to RDMA?



[WangJaingChenYiCui'17]

Cluster	CX3 (InfiniBand)	CX4 (Eth)	CX5 (Eth)
RDMA read	1.7 µs	2.9 µs	2.0 µs
eRPC	2.1 µs	3.7 µs	2.3 µs

Tabl[illegible] Comparison of median latency with eRPC and RDMA

Old technology, optimized software

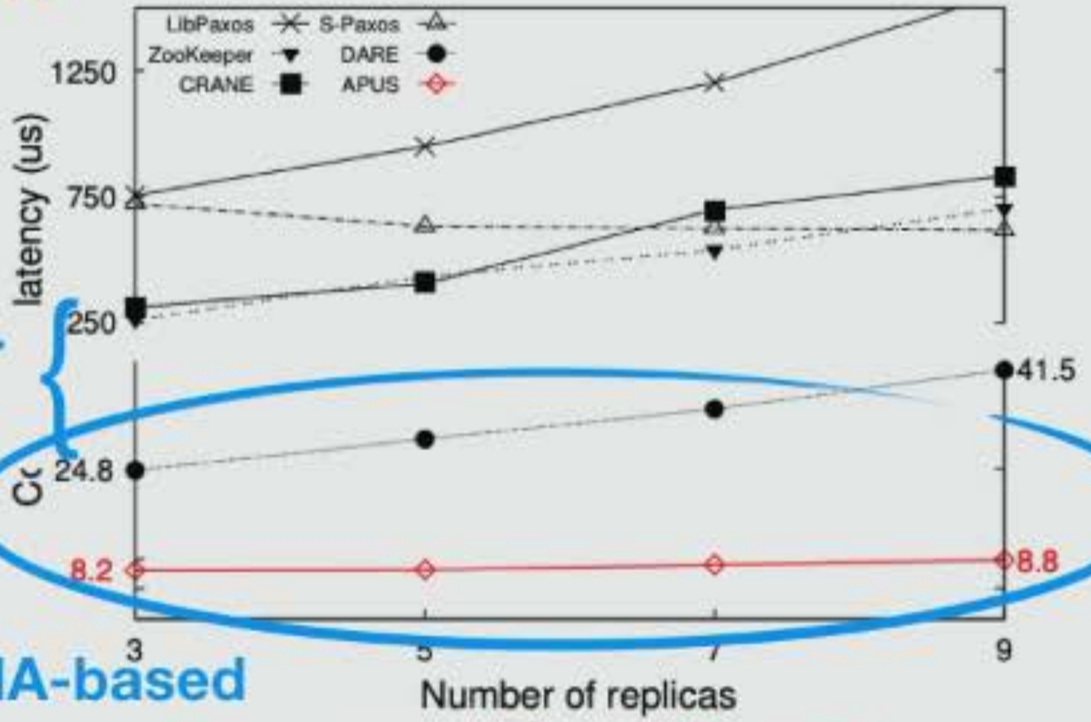
[KaliaKaminskiAndersen'19]

RDMA in Practice

- Huge speedups with RDMA
- But is this improvement fundamental to RDMA?

Lower is better

Factor of 10



RDMA-based

[WangJaingChenYiCui'17]

Performance comparable with RDMA

Cluster	CX3 (InfiniBand)	CX4 (Eth)	CX5 (Eth)
RDMA read	1.7 µs	2.9 µs	2.0 µs
eRPC	2.1 µs	3.7 µs	2.3 µs

Tabl Comparison of median latency with eRPC and RDMA

Old technology, optimized software

[KaliaKaminskiAndersen'19]

Is RDMA fundamentally better than other communication mechanisms?

Performance

Is RDMA fundamentally better than other communication mechanisms?

Fault Tolerance

Performance

Is RDMA fundamentally better than other communication mechanisms?

Fault Tolerance

Performance

Scalability

Is RDMA fundamentally better than other communication mechanisms?

Consensus as a Lens

Consensus: **Agreement**

Consensus as a Lens

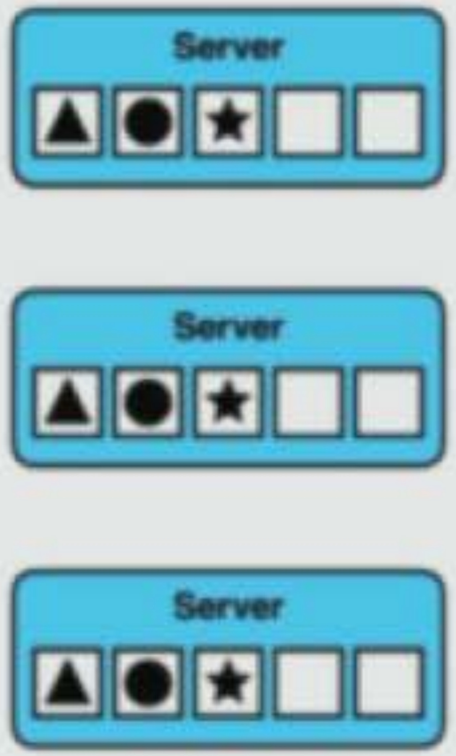
Consensus: **Agreement**

Replication, shared data structures, blockchains

Consensus as a Lens

Consensus: **Agreement**

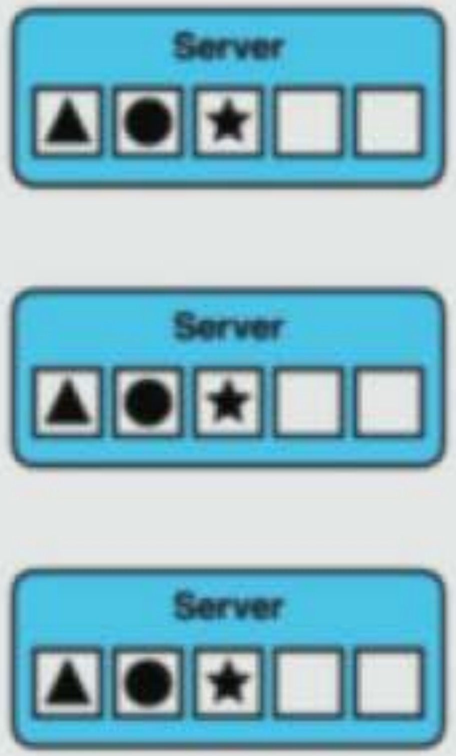
Replication, shared data structures, blockchains



Consensus as a Lens

Consensus: **Agreement**

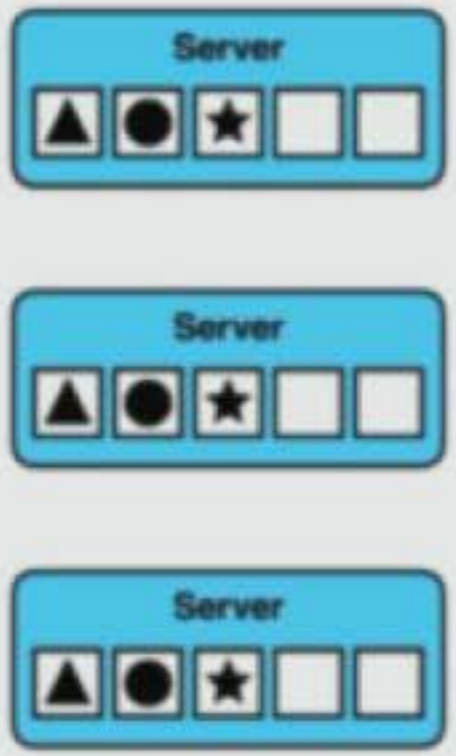
Replication, shared data structures, blockchains

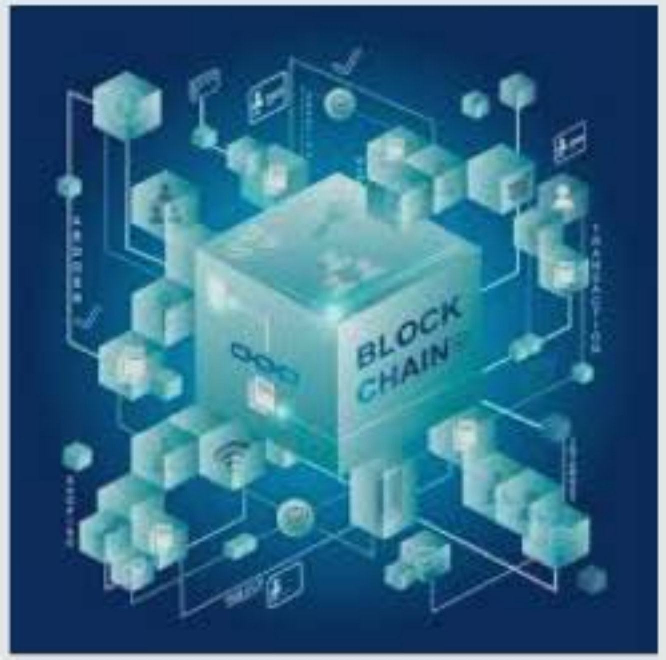


Consensus as a Lens

Consensus: **Agreement**

Replication, shared data structures, blockchains





Consensus: Definition

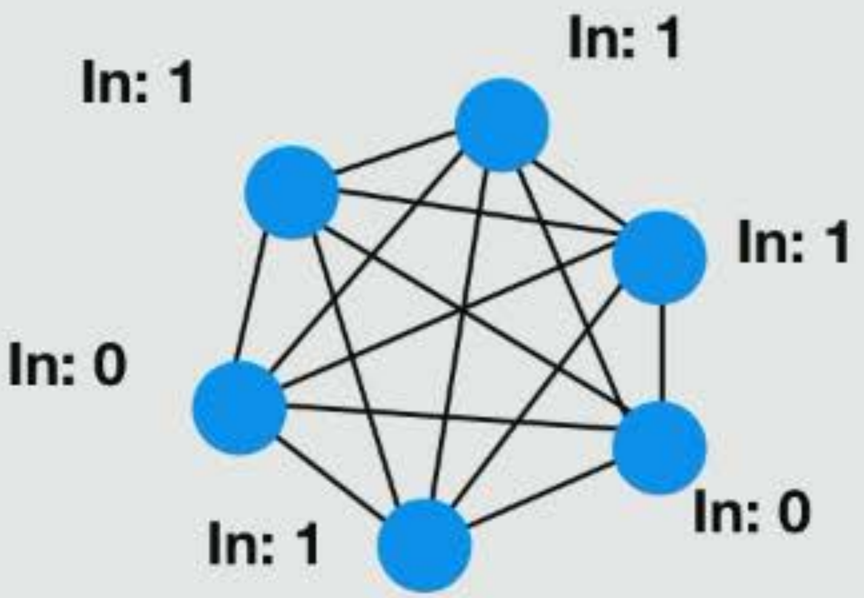
- **Input**: every process gets input
- **Output**: Every process outputs something

Consensus: Definition

- **Input**: every process gets input
- **Output**: Every process outputs something

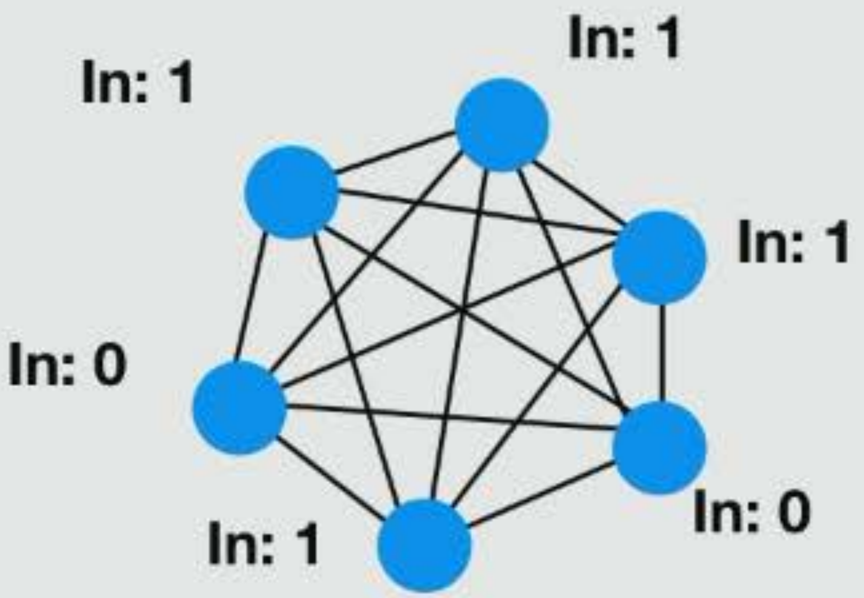
Consensus: Definition

- **Input**: every process gets input
- **Output**: Every process outputs something



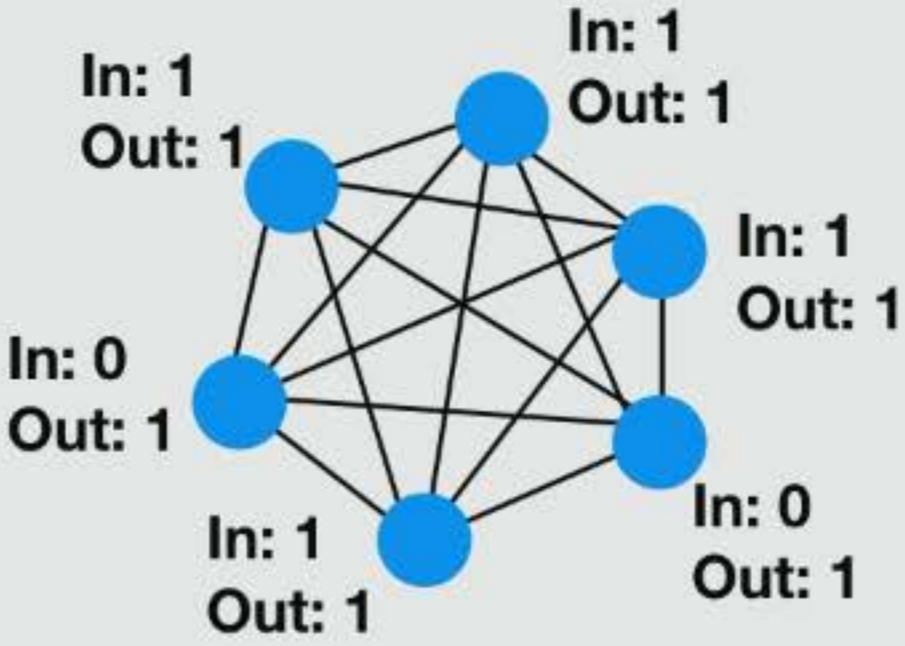
Consensus: Definition

- **Input**: every process gets input
- **Output**: Every process outputs something
  - Agreement: Every process outputs the same value



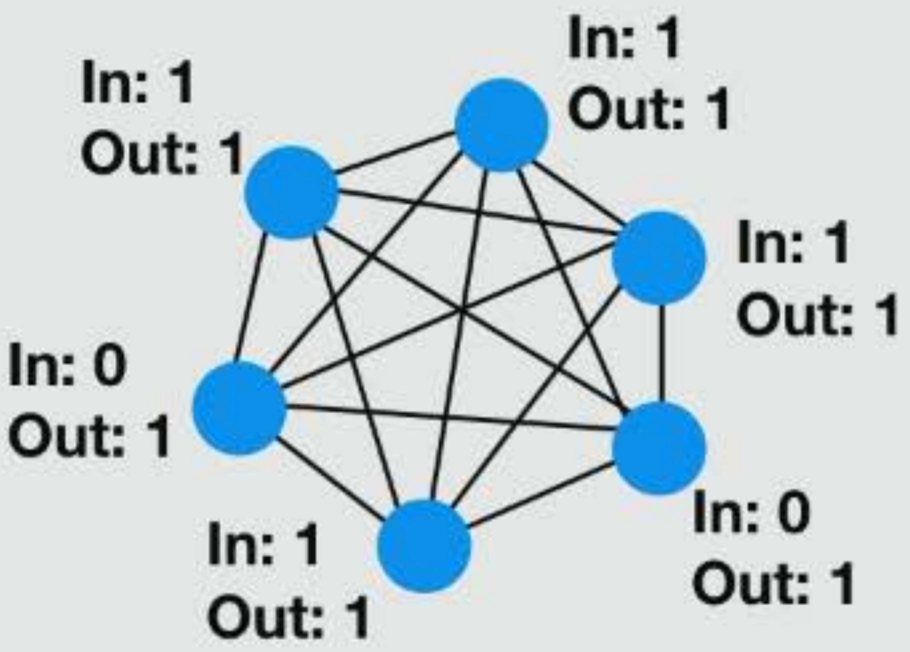
Consensus: Definition

- **Input**: every process gets input
- **Output**: Every process outputs something
  - Agreement: Every process outputs the same value



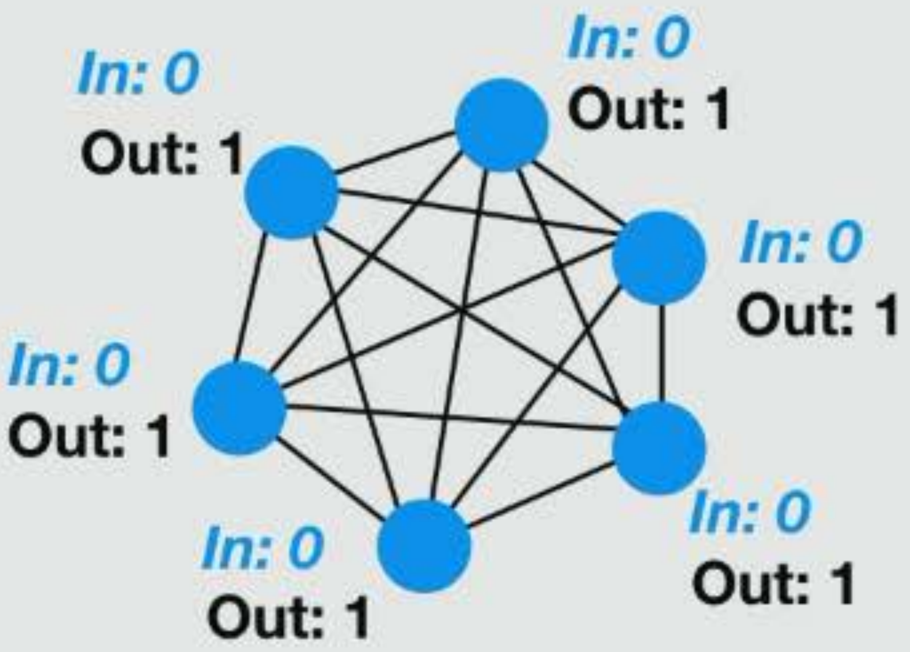
Consensus: Definition

- **Input**: every process gets input
- **Output**: Every process outputs something
  - Agreement: Every process outputs the same value
  - Validity: output value must be input of some process



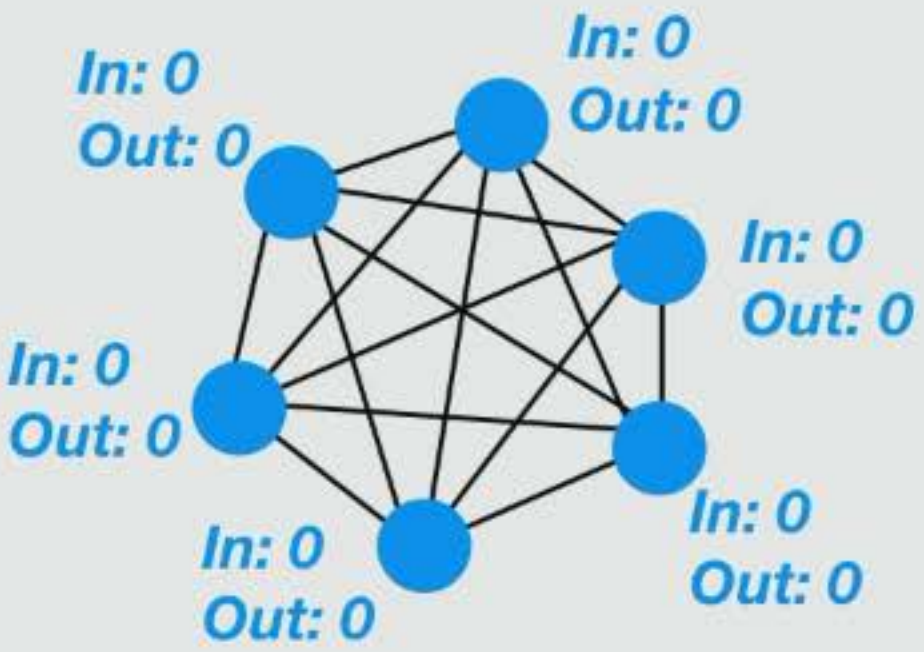
Consensus: Definition

- **Input**: every process gets input
- **Output**: Every process outputs something
  - Agreement: Every process outputs the same value
  - Validity: output value must be input of some process



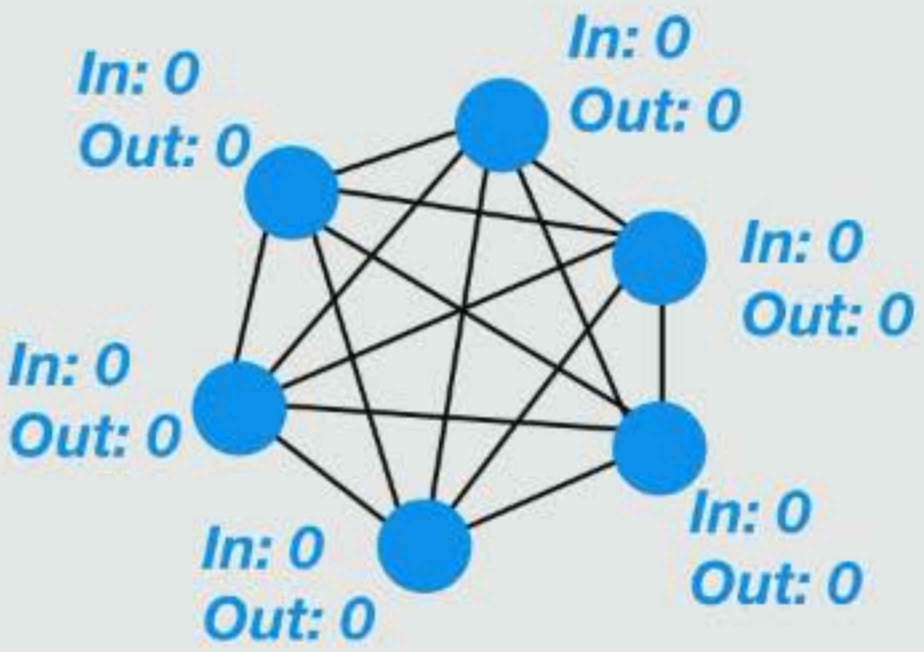
Consensus: Definition

- **Input**: every process gets input
- **Output**: Every process outputs something
  - Agreement: Every process outputs the same value
  - Validity: output value must be input of some process



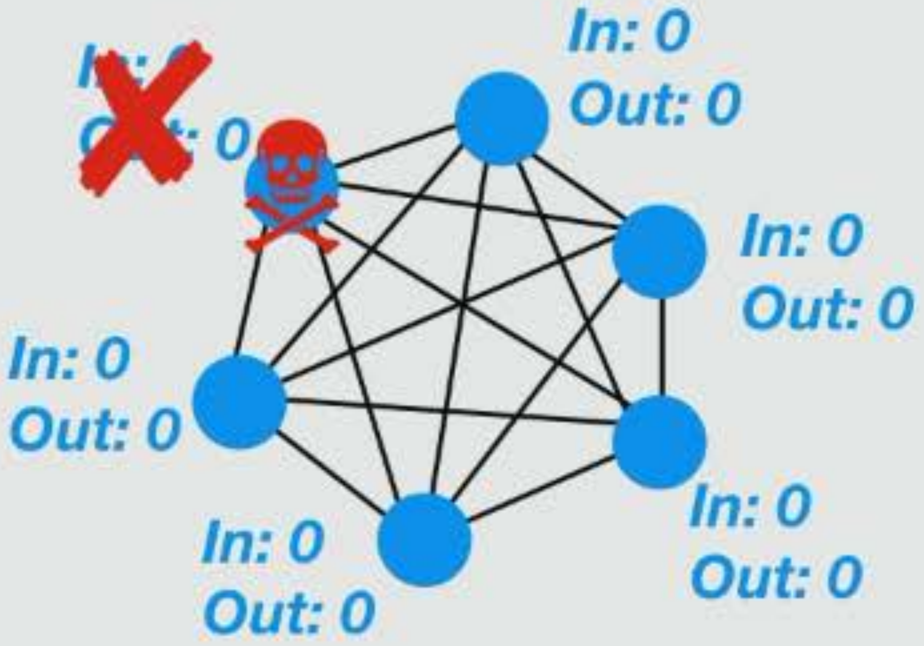
Consensus: Definition

- **Input**: every process gets input
- **Output**: Every process outputs something
  - Agreement: Every process outputs the same value
  - Validity: output value must be input of some process
- **Challenges**: **Asynchrony**, processes **crash** or are **Byzantine**



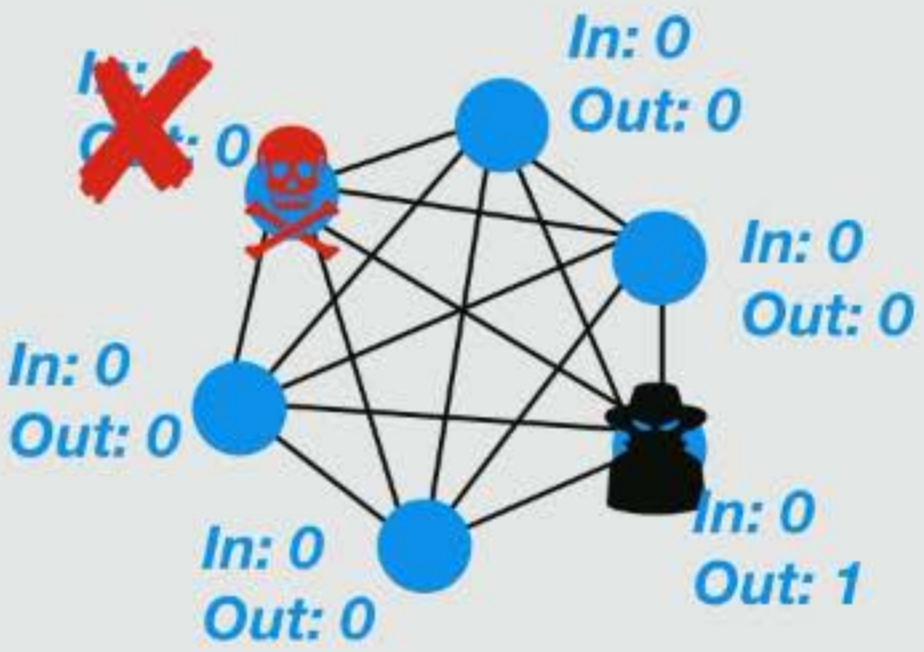
Consensus: Definition

- **Input**: every process gets input
- **Output**: Every process outputs something
  - Agreement: Every process outputs the same value
  - Validity: output value must be input of some process
- **Challenges**: **Asynchrony**, processes **crash** or are **Byzantine**



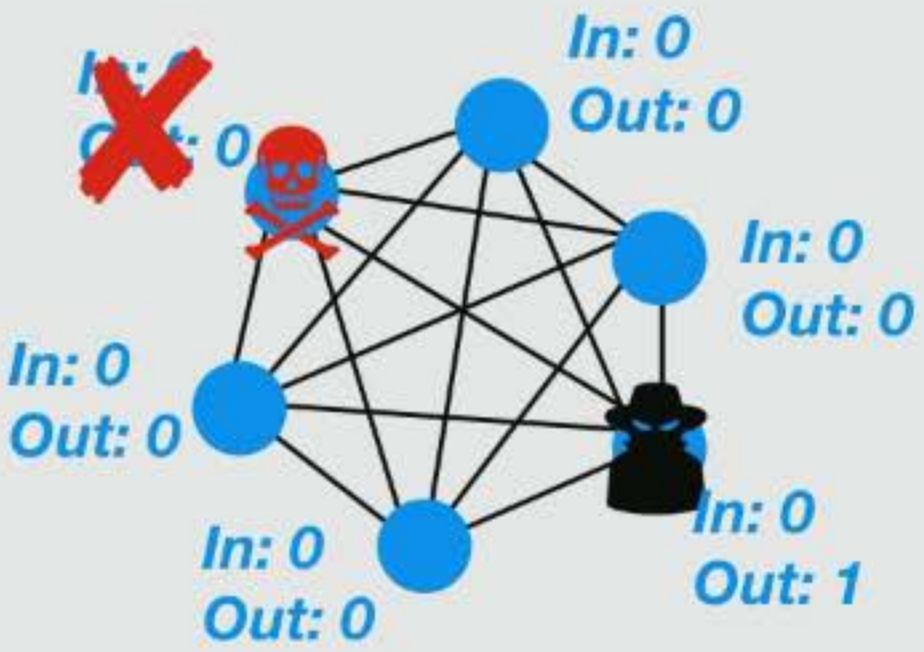
Consensus: Definition

- **Input**: every process gets input
- **Output**: Every process outputs something
  - Agreement: Every process outputs the same value
  - Validity: output value must be input of some process
- **Challenges**: **Asynchrony**, processes **crash** or are **Byzantine**



Consensus: Definition

- **Input**: every process gets input
- **Output**: Every *correct* process outputs something
  - Agreement: Every process outputs the same value
  - Validity: output value must be input of some process
- **Challenges**: **Asynchrony**, processes **crash** or are **Byzantine**



Fault Tolerance

Performance

Scalability

Is RDMA fundamentally better than other communication mechanisms?

Communication Mechanisms

Communication Mechanisms

“Message Passing”

Communication Mechanisms

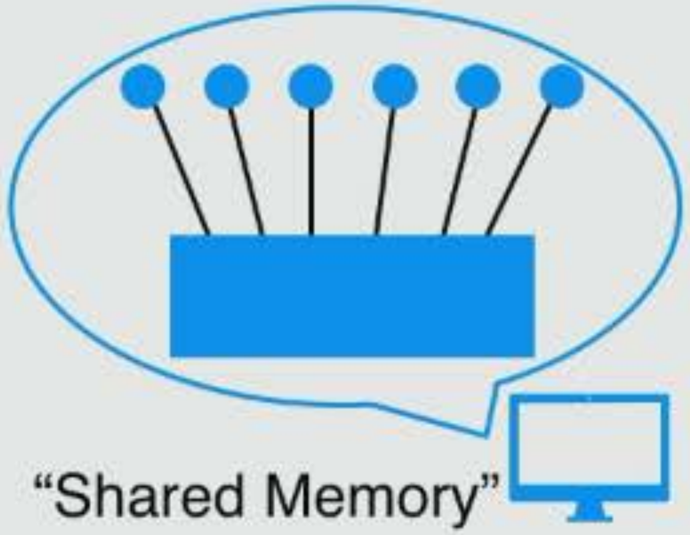
"Message Passing"

- Data centers, Internet

Communication Mechanisms

"Message Passing"

- Data centers, Internet

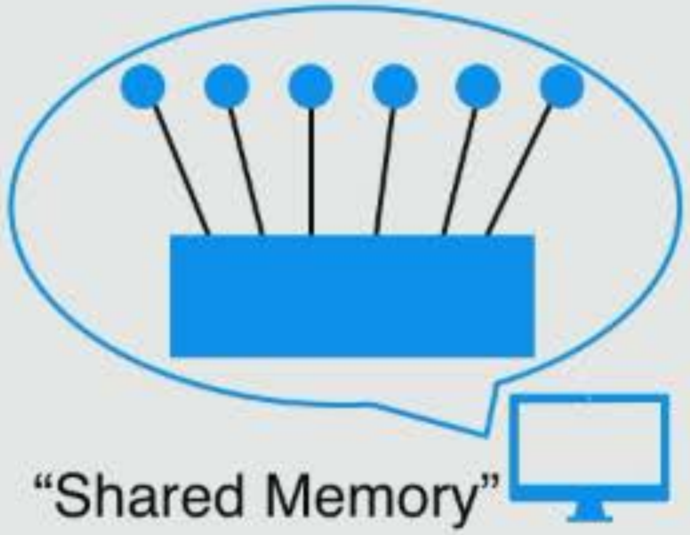


"Shared Memory"

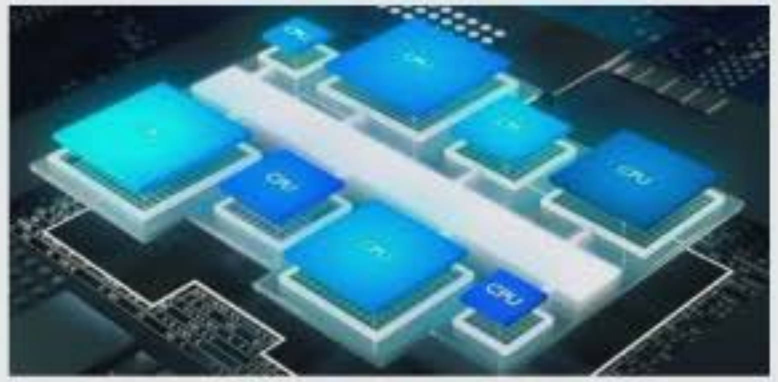
Communication Mechanisms

"Message Passing"

- Data centers, Internet

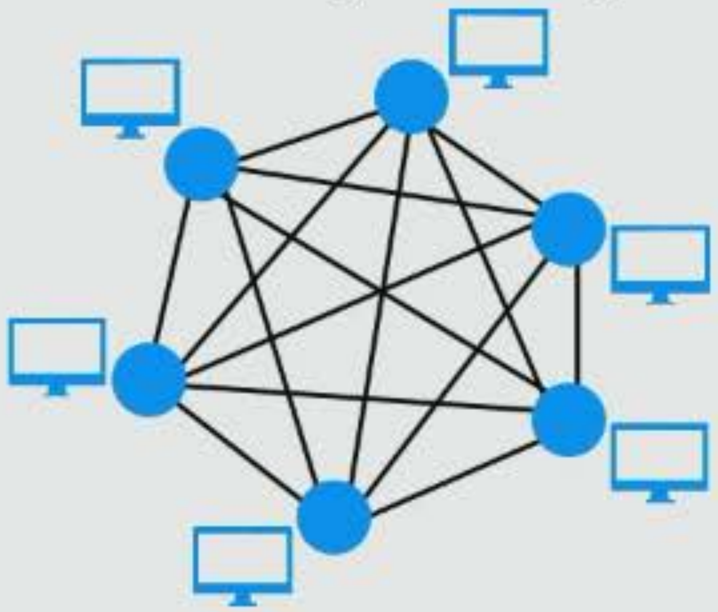


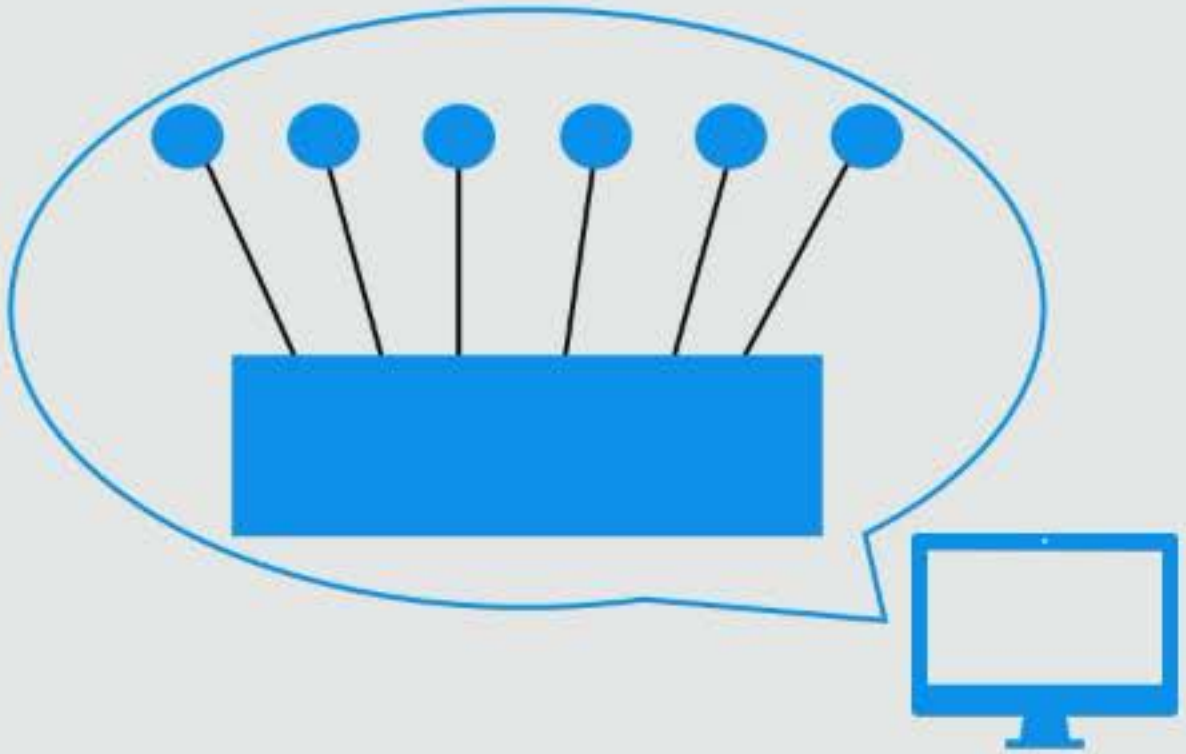
"Shared Memory"



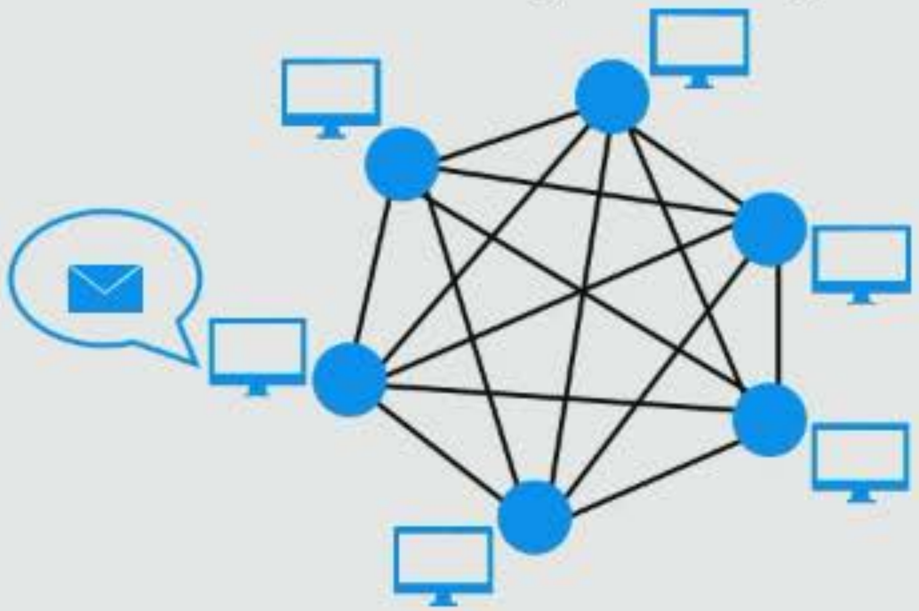
- Multicore machines

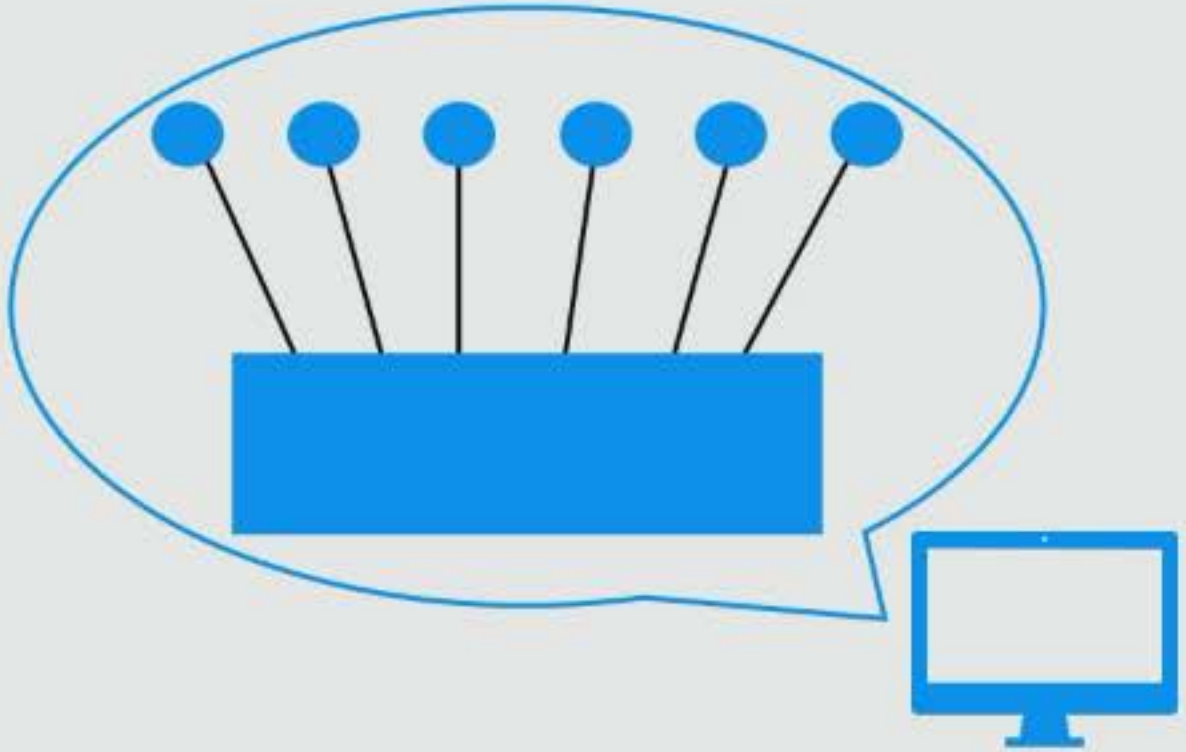
Message Passing vs Shared Memory





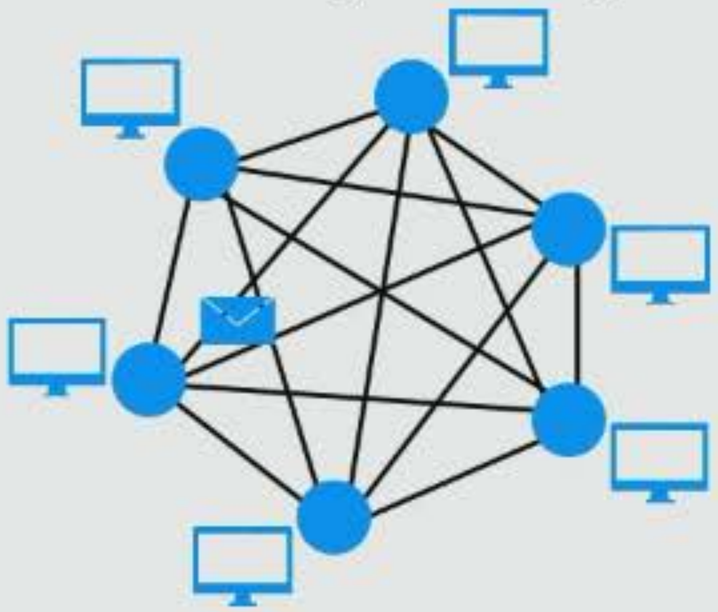
Message Passing vs Shared Memory



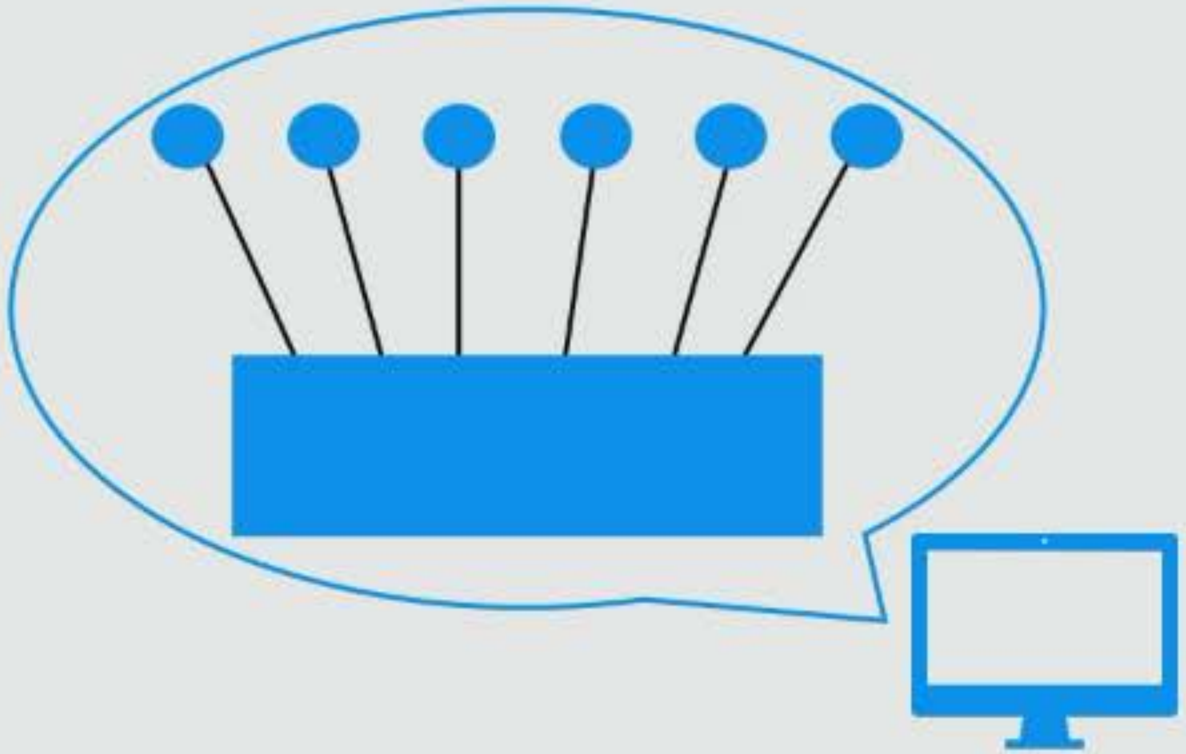


Message Passing vs Shared Memory

Message Passing



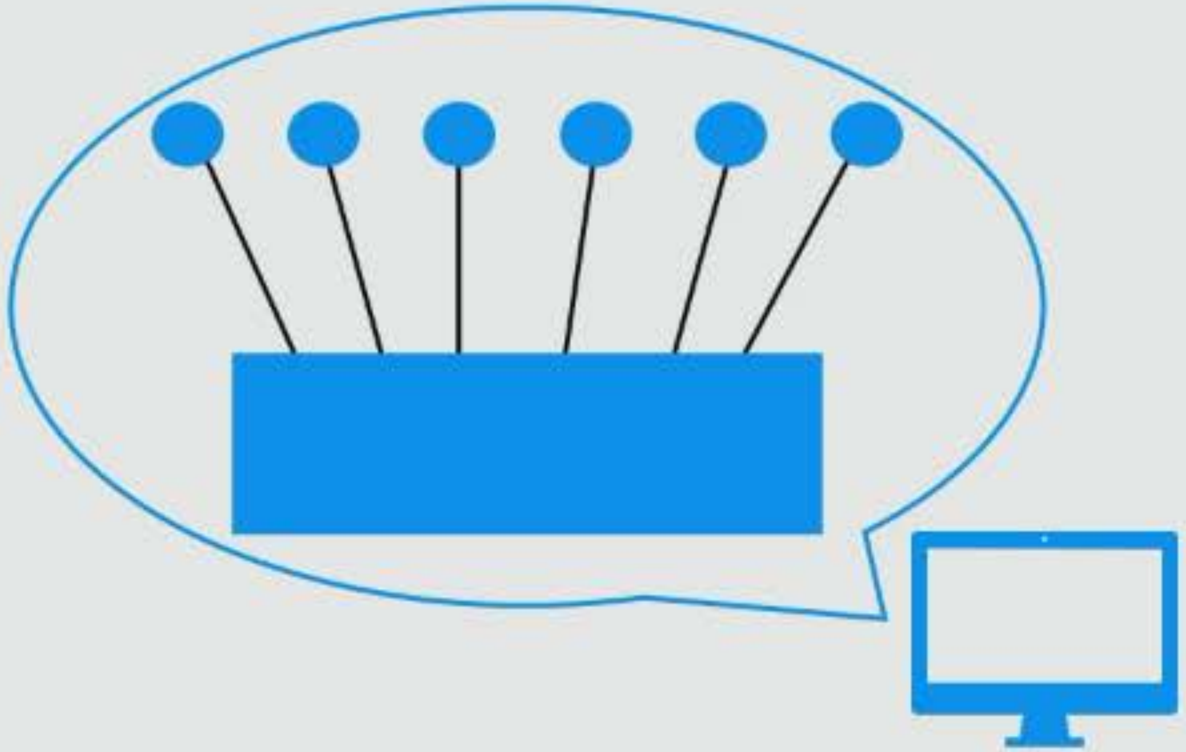
Shared Memory



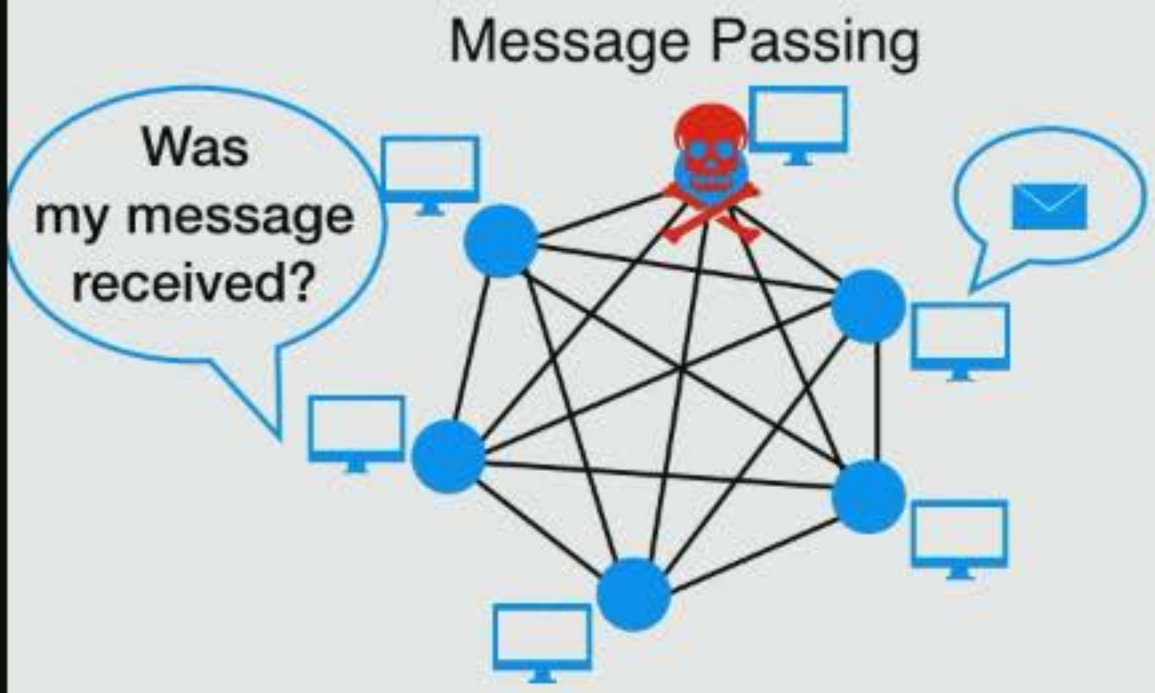
Message Passing vs Shared Memory

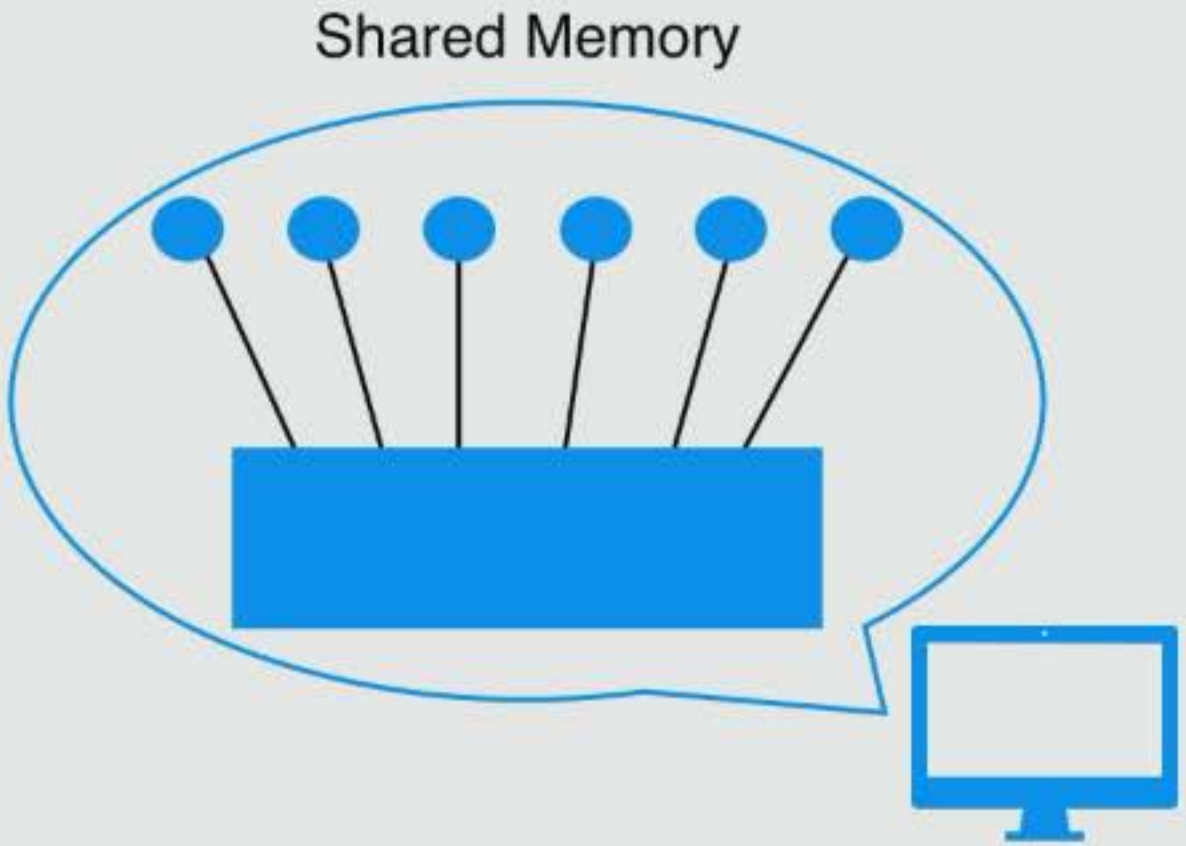
Message Passing

Shared Memory

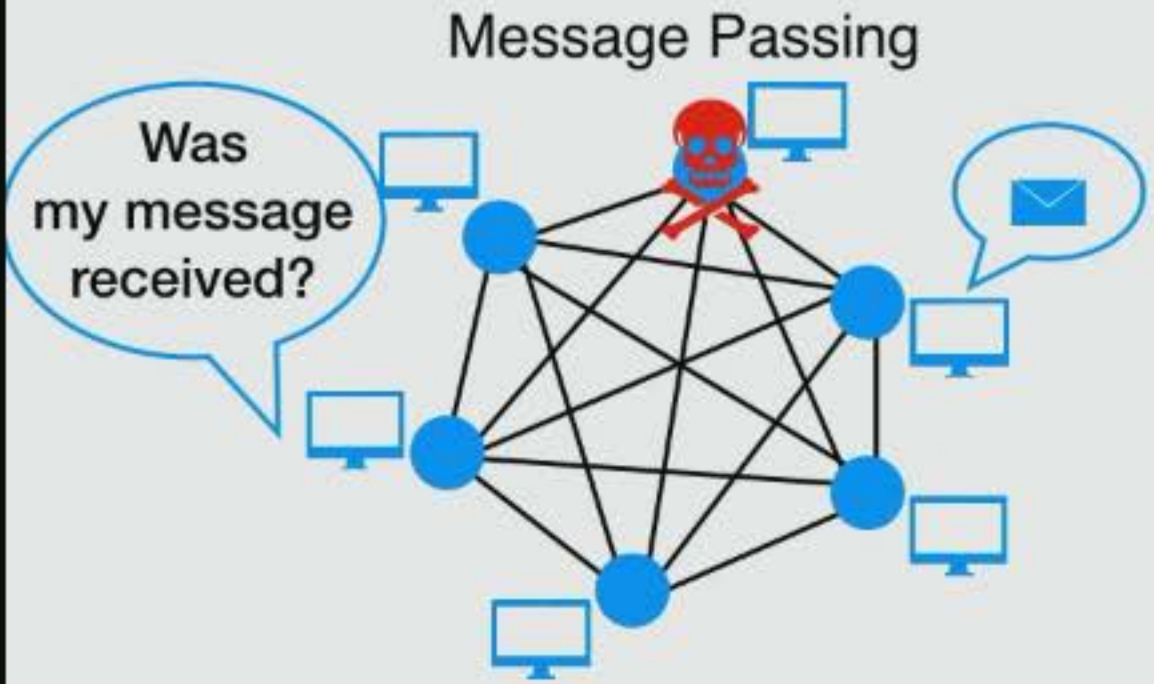


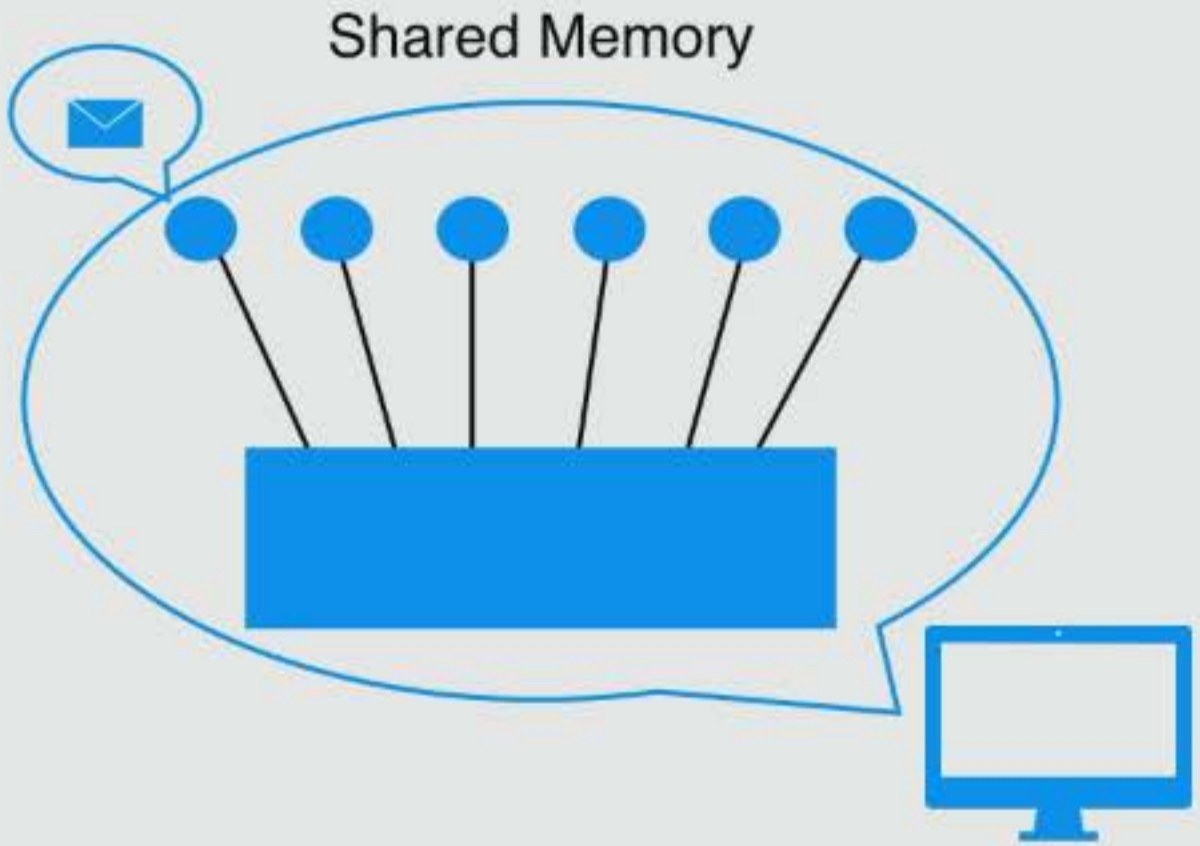
Message Passing vs Shared Memory



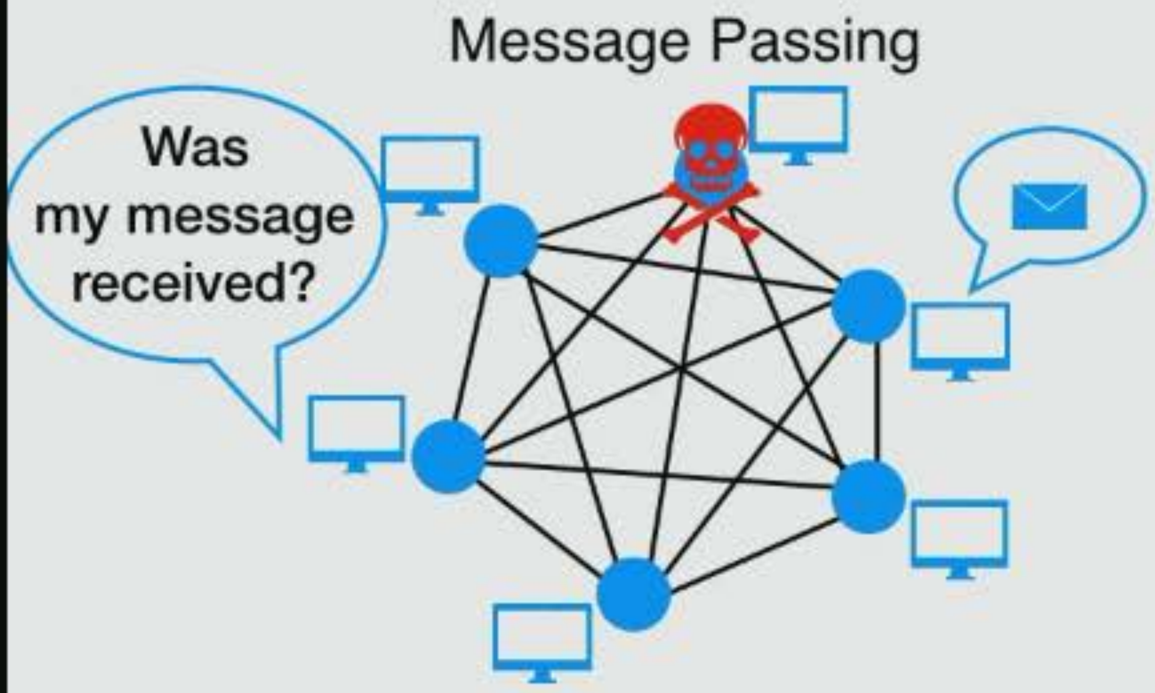


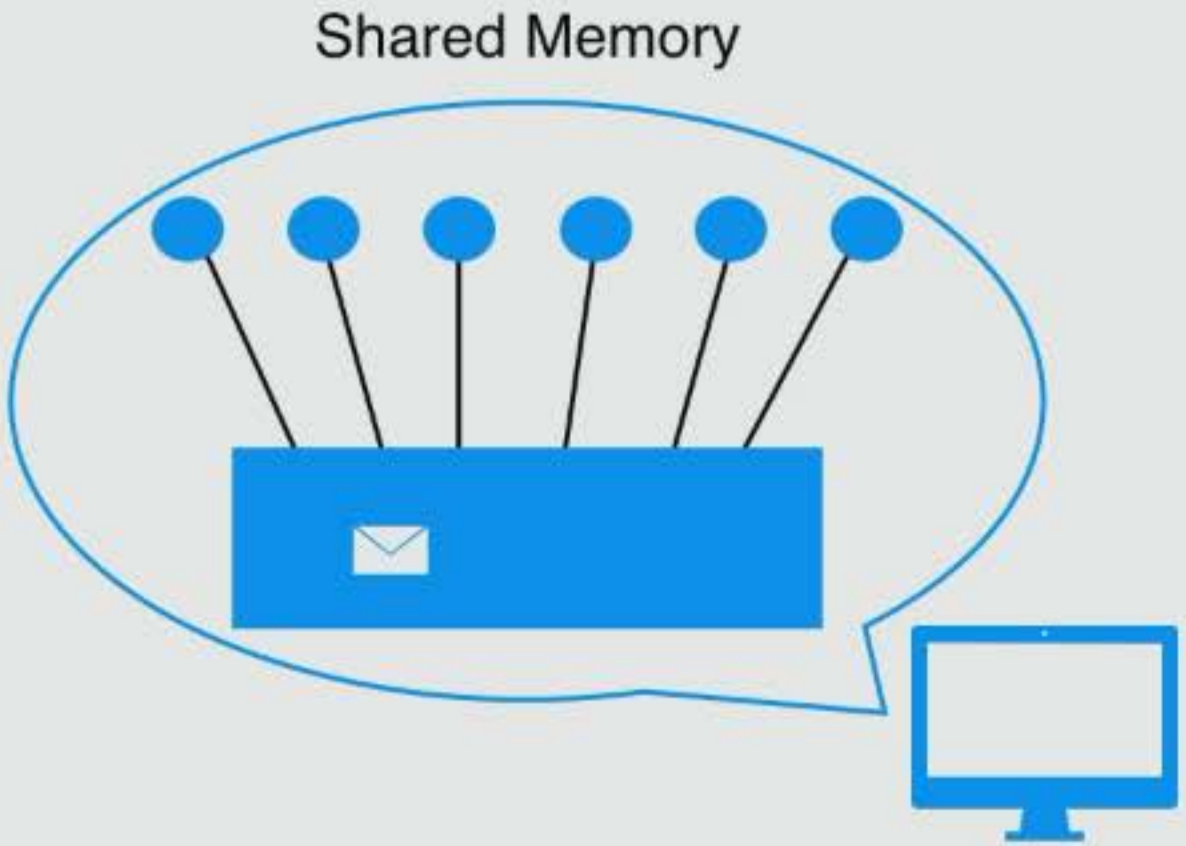
Message Passing vs Shared Memory





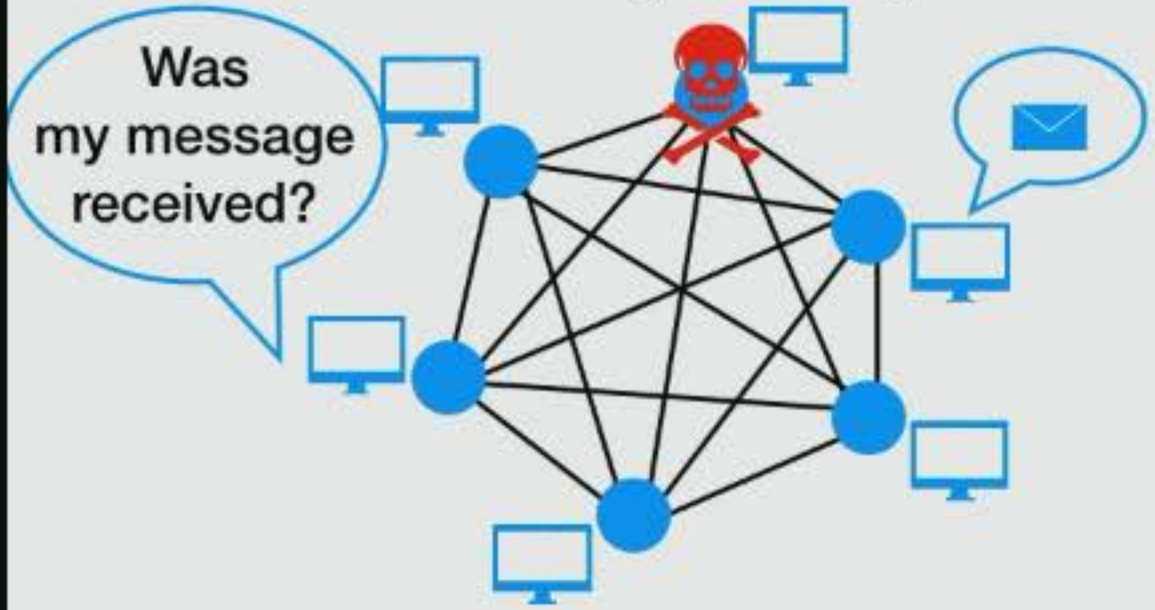
Message Passing vs Shared Memory



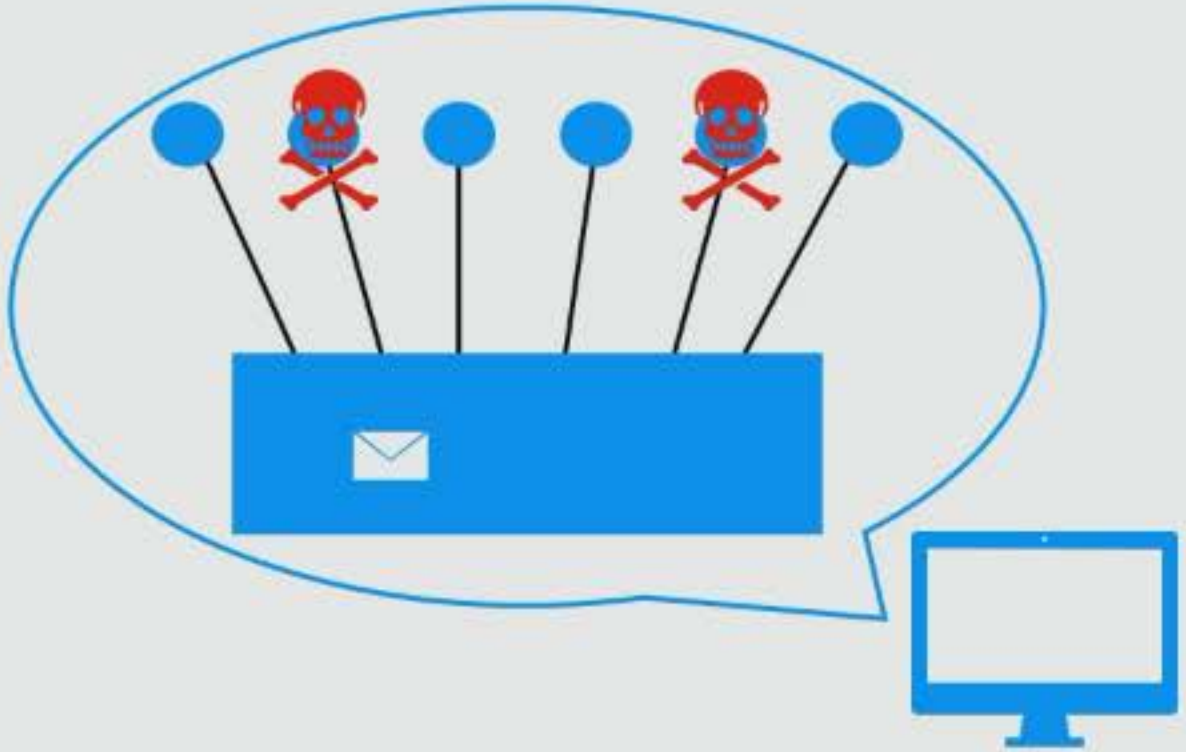


Message Passing vs Shared Memory

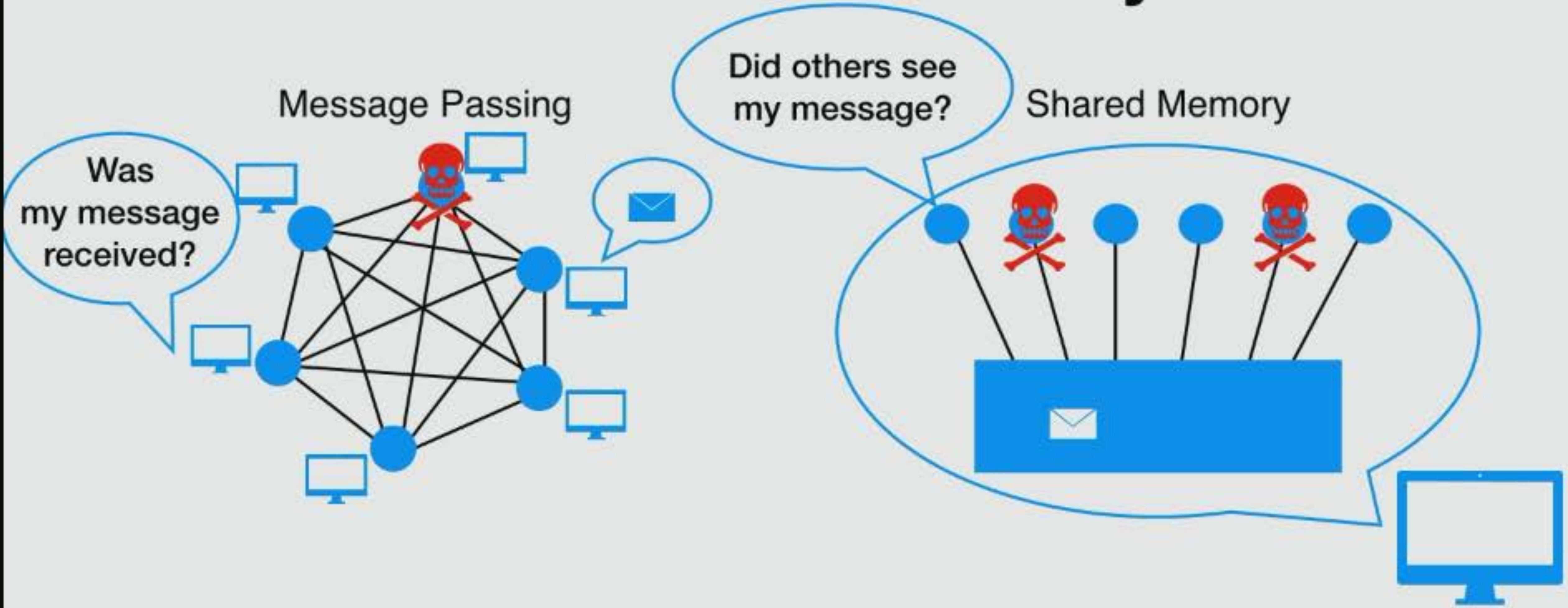
Message Passing



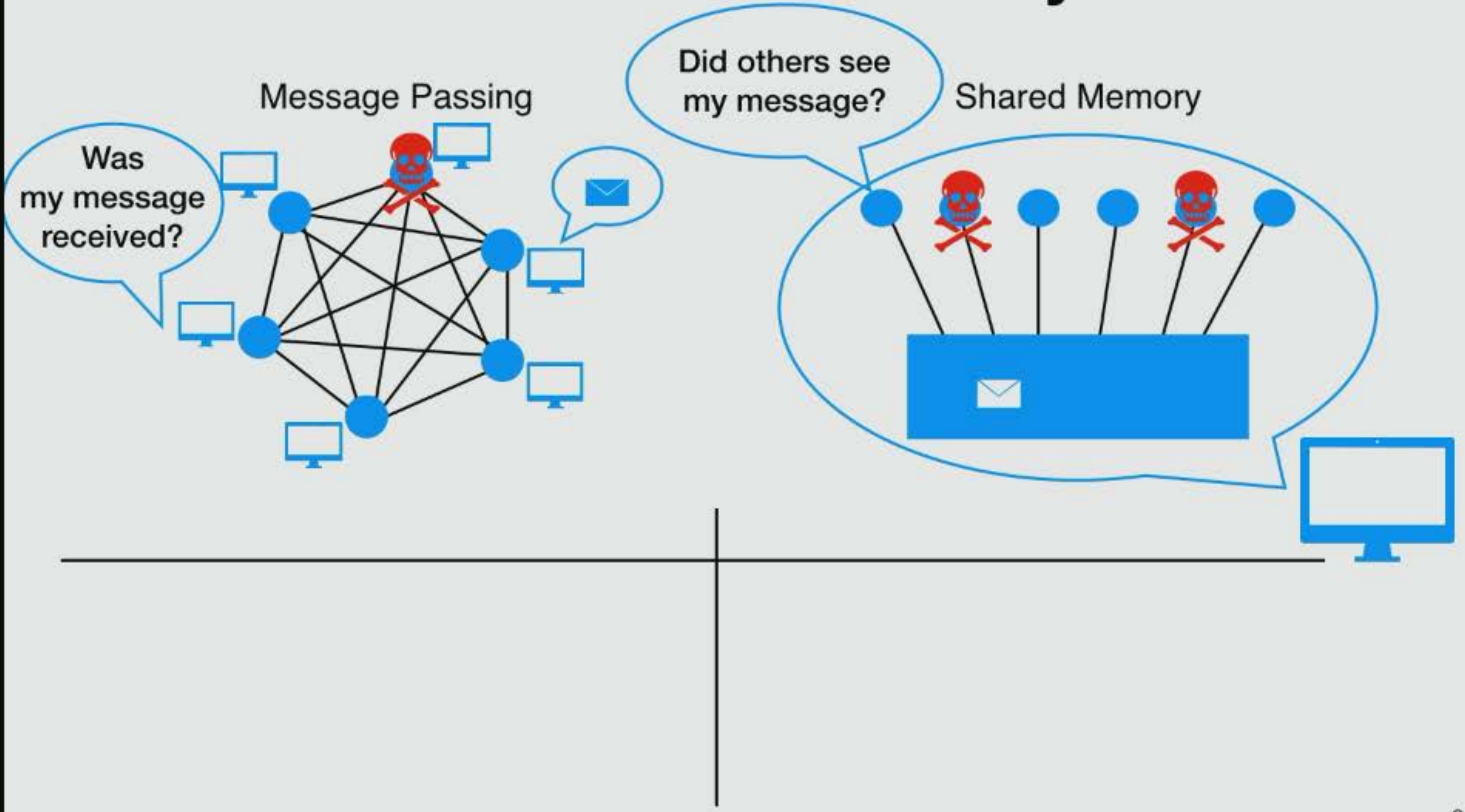
Shared Memory



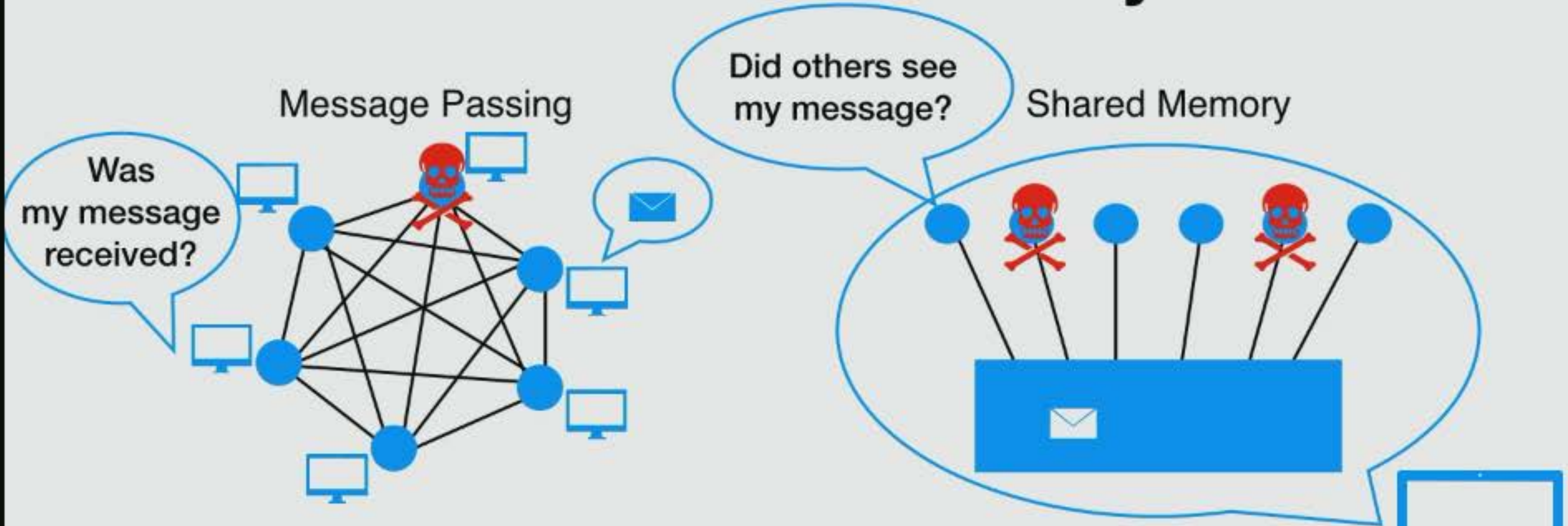
Message Passing vs Shared Memory



Message Passing vs Shared Memory

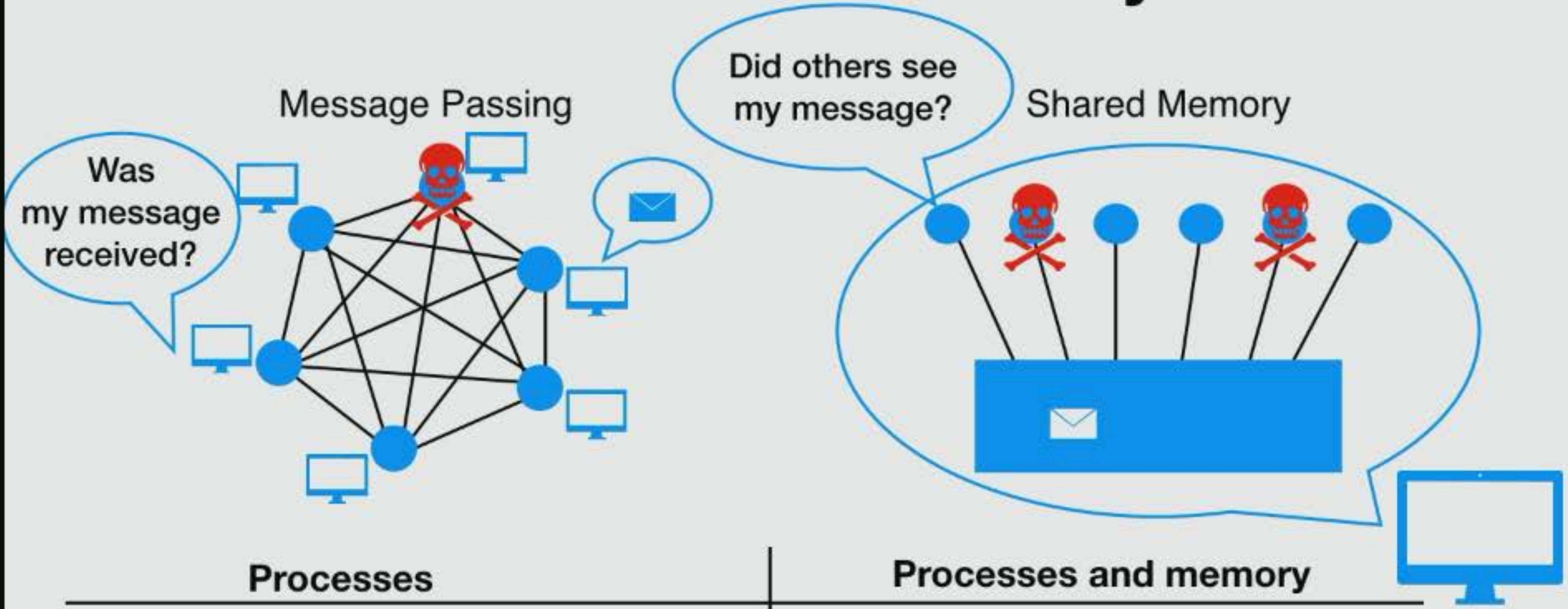


Message Passing vs Shared Memory



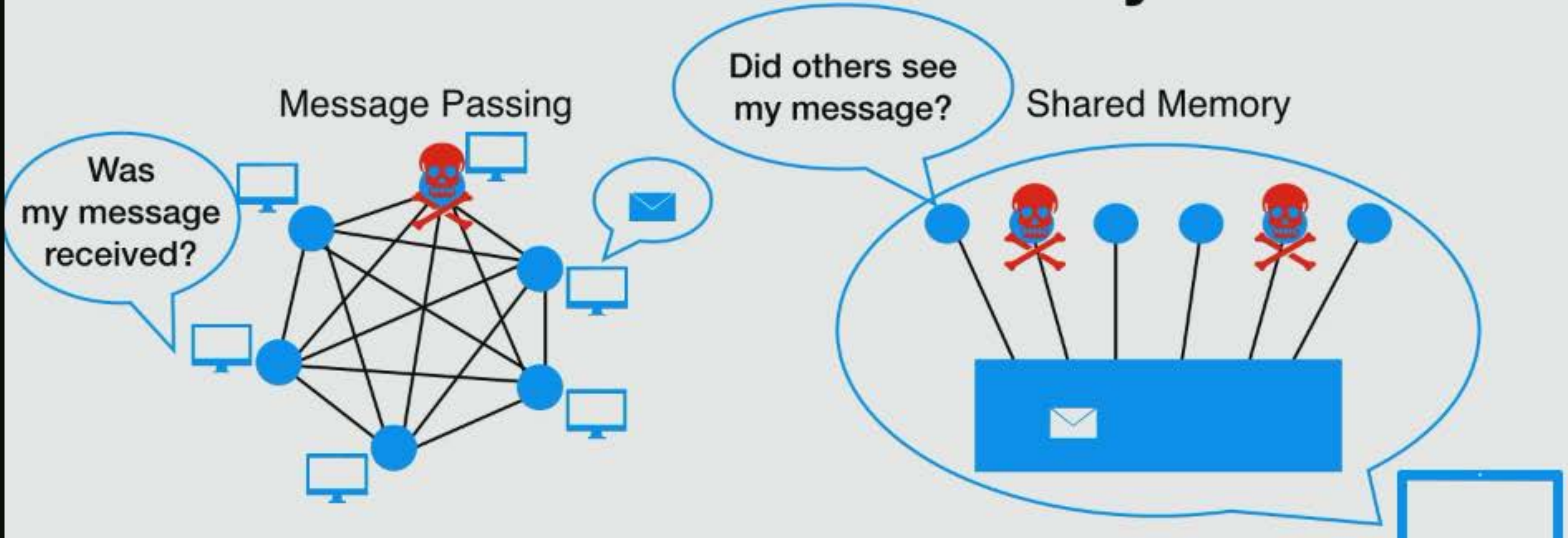
Processes

Message Passing vs Shared Memory



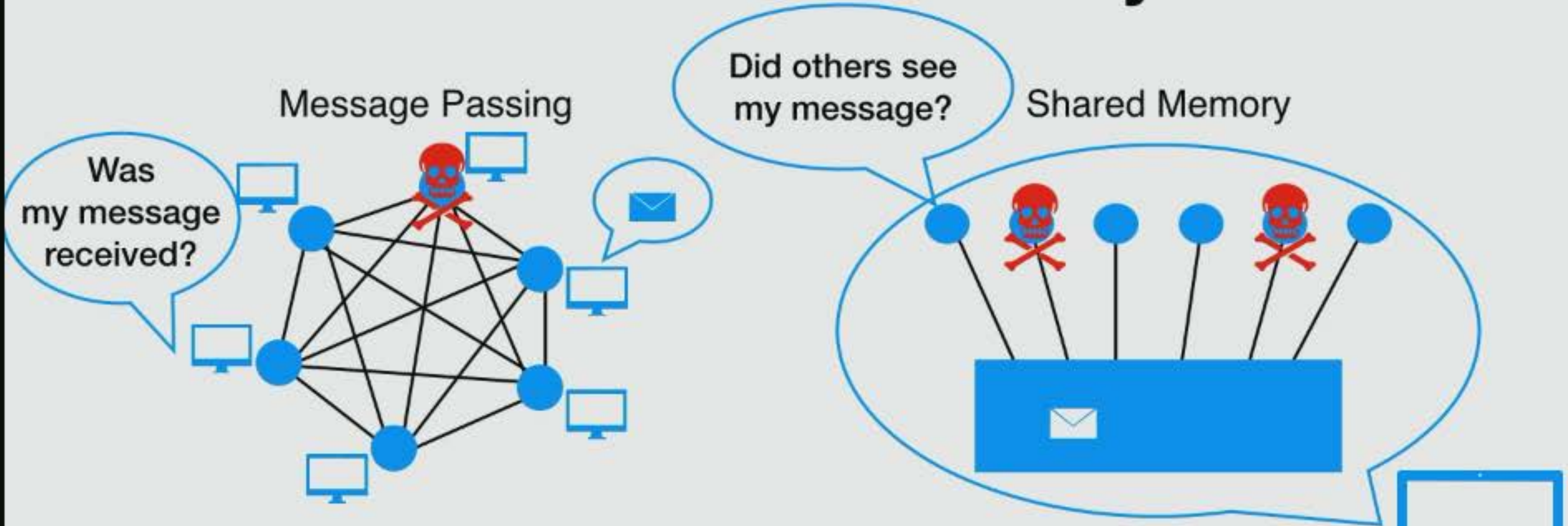
Processes	**Processes and memory**

Message Passing vs Shared Memory



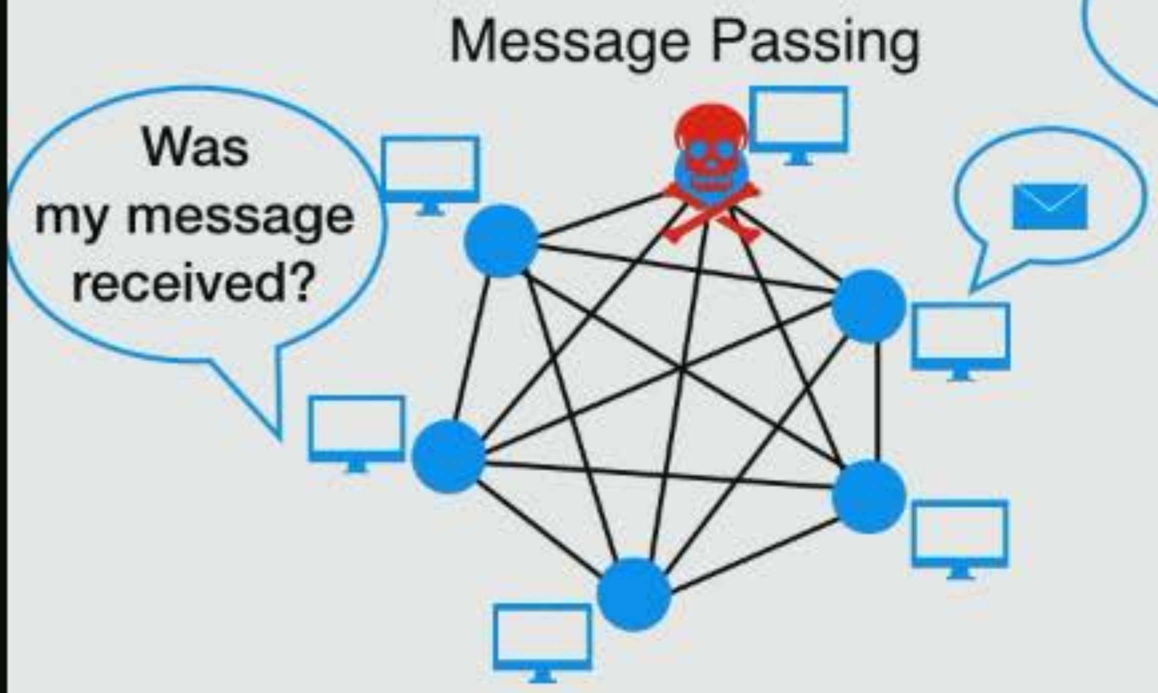
Processes	Processes and memory
Rely on other processes to know if communication was successful	

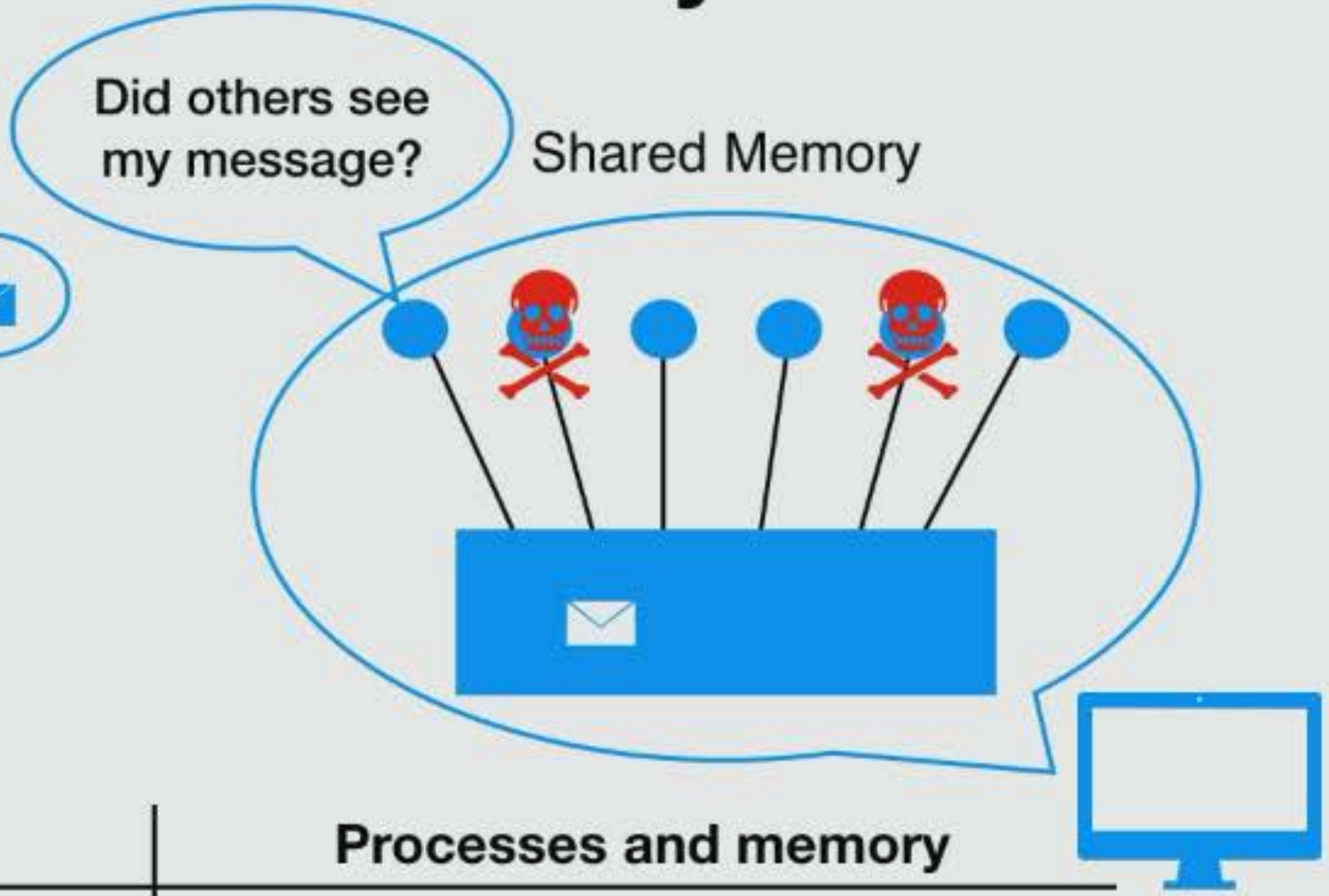
Message Passing vs Shared Memory



Processes	Processes and memory
Rely on other processes to know if communication was successful	Rely on memory to know if communication was successful

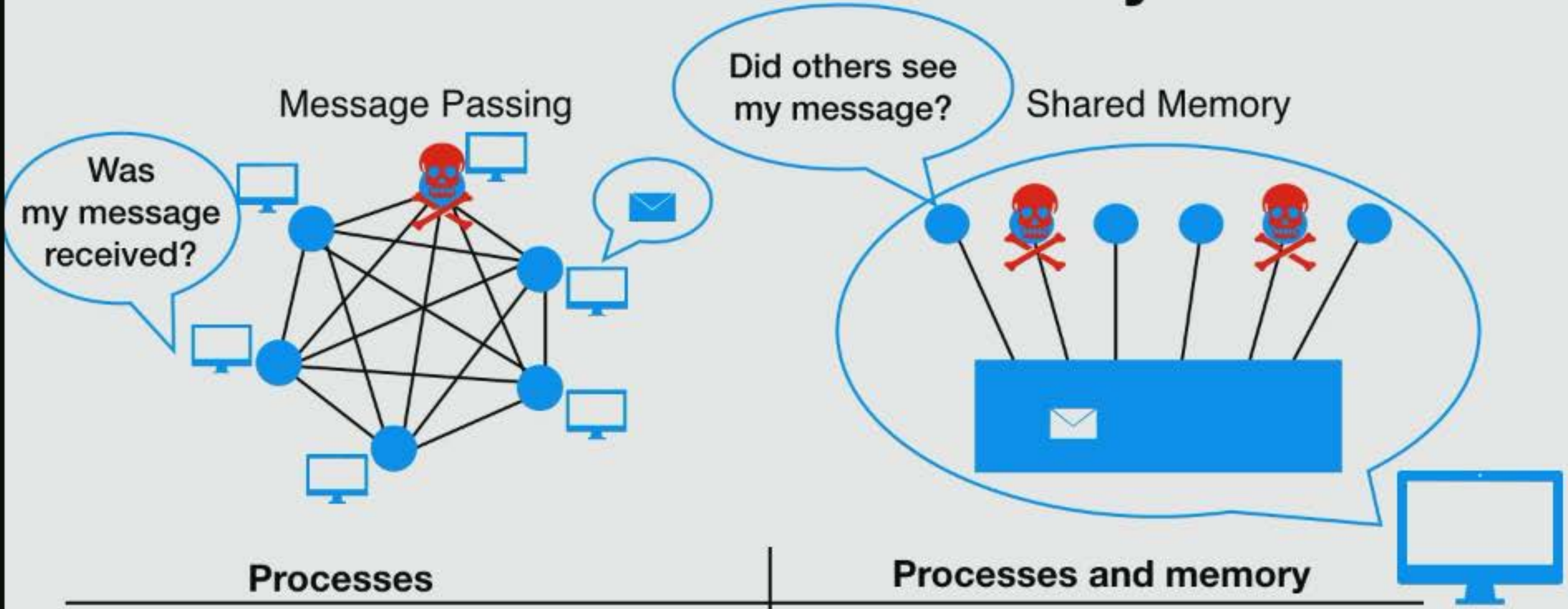
Message Passing vs Shared Memory





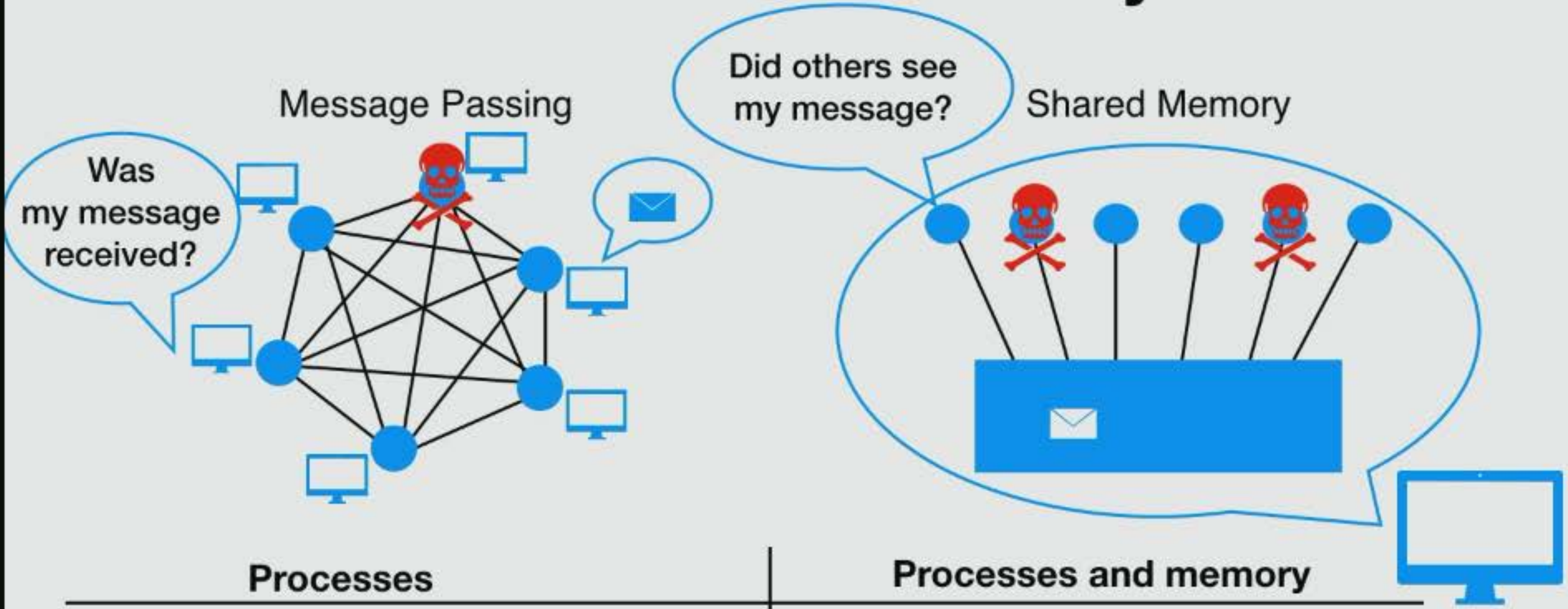
Processes	Processes and memory
Rely on other processes to know if communication was successful	Rely on memory to know if communication was successful

Message Passing vs Shared Memory



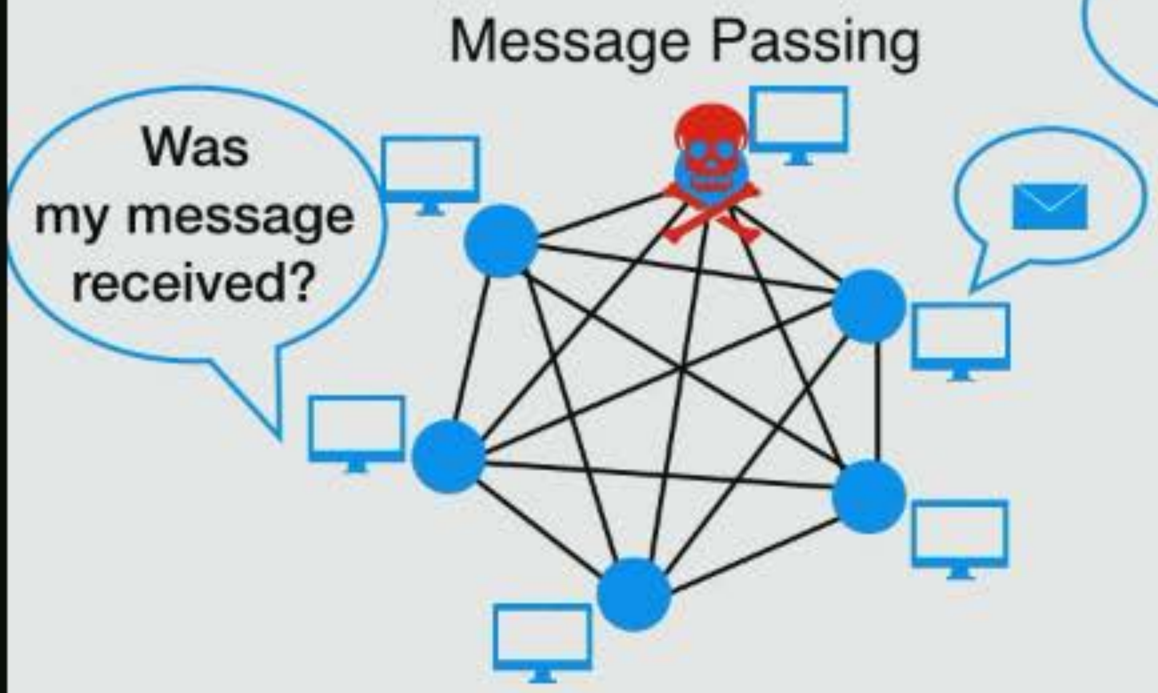
Processes	Processes and memory
Rely on other processes to know if communication was successful	Rely on memory to know if communication was successful
Need "ack" from each process	

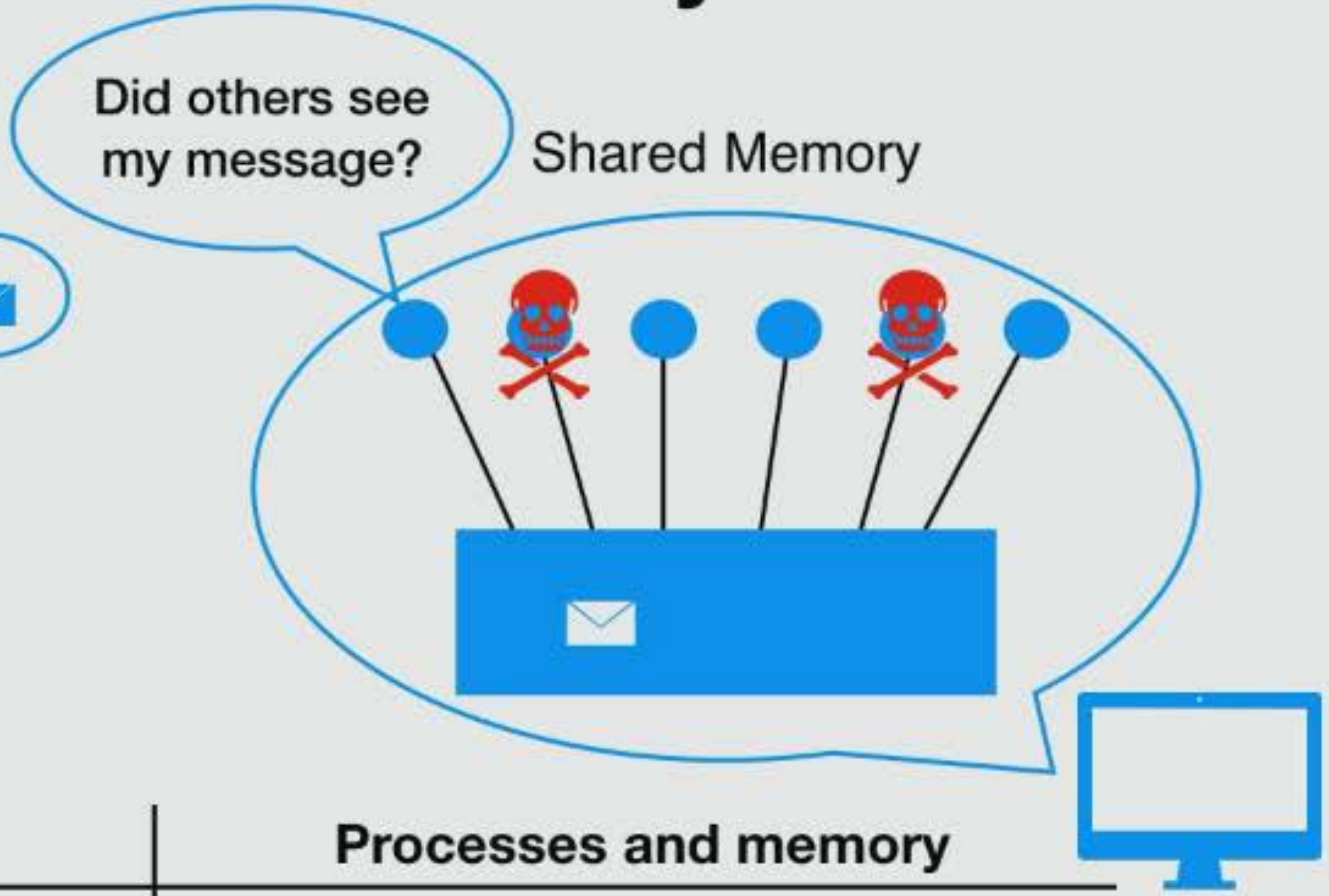
Message Passing vs Shared Memory



Processes	Processes and memory
Rely on other processes to know if communication was successful ☹	☺ **Rely on memory** to know if communication was successful
Need **“ack” from each process**	**Single “ack”** from memory

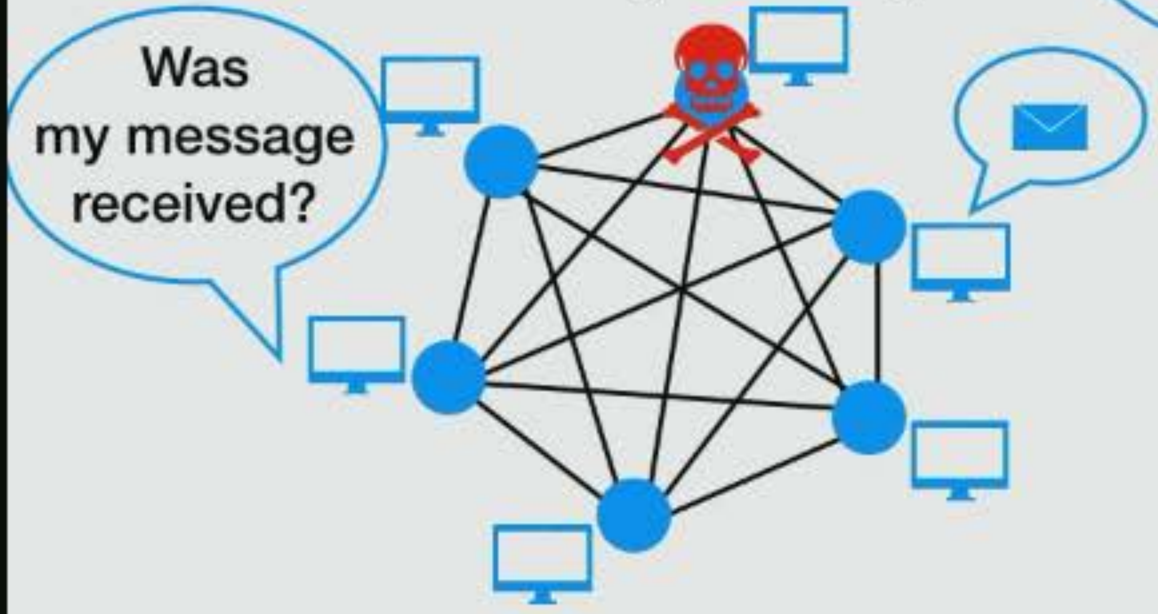
Message Passing vs Shared Memory

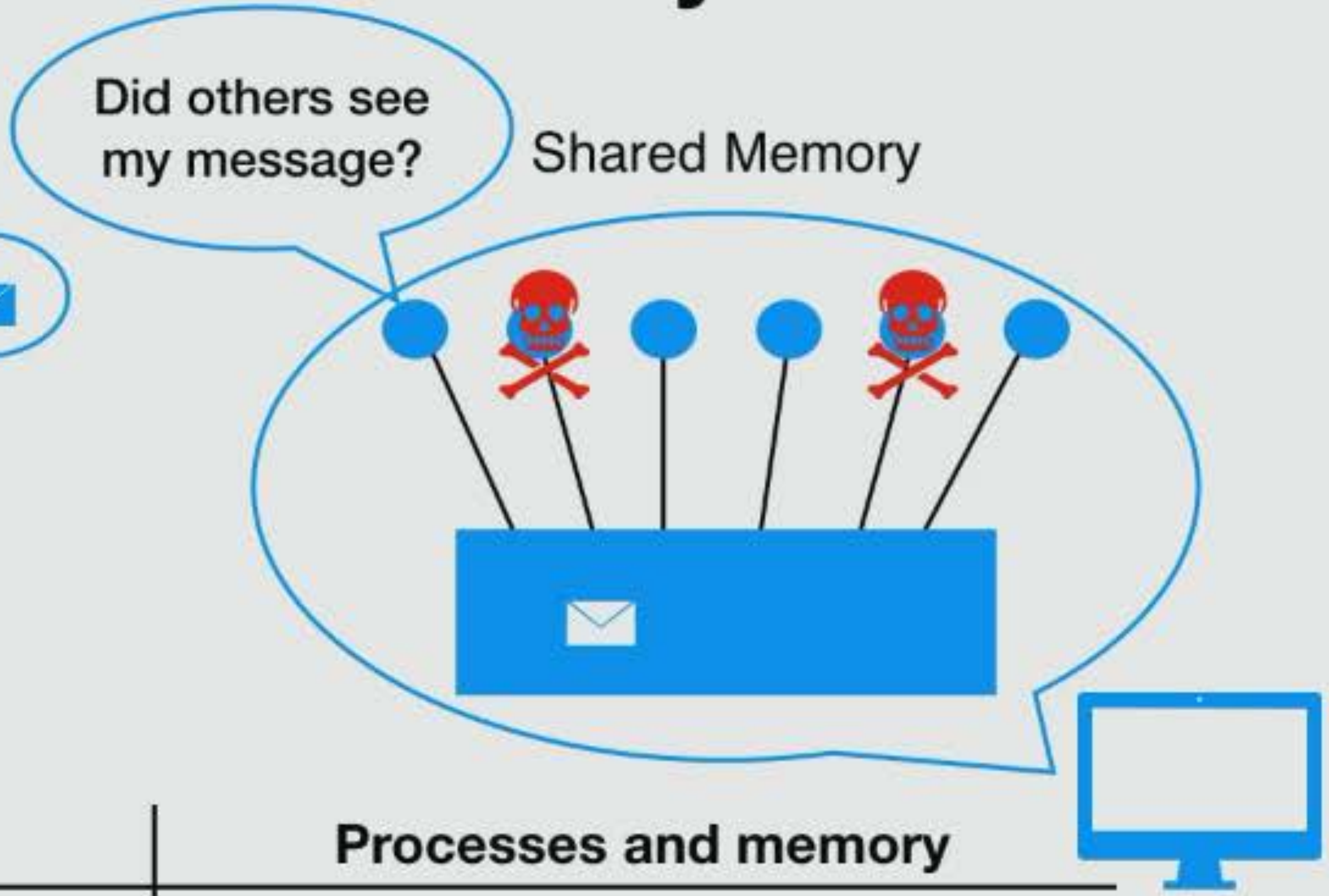




Processes			Processes and memory
Rely on other processes to know if communication was successful	☹	☺	Rely on memory to know if communication was successful
Need "ack" from each process	☹	☺	Single "ack" from memory

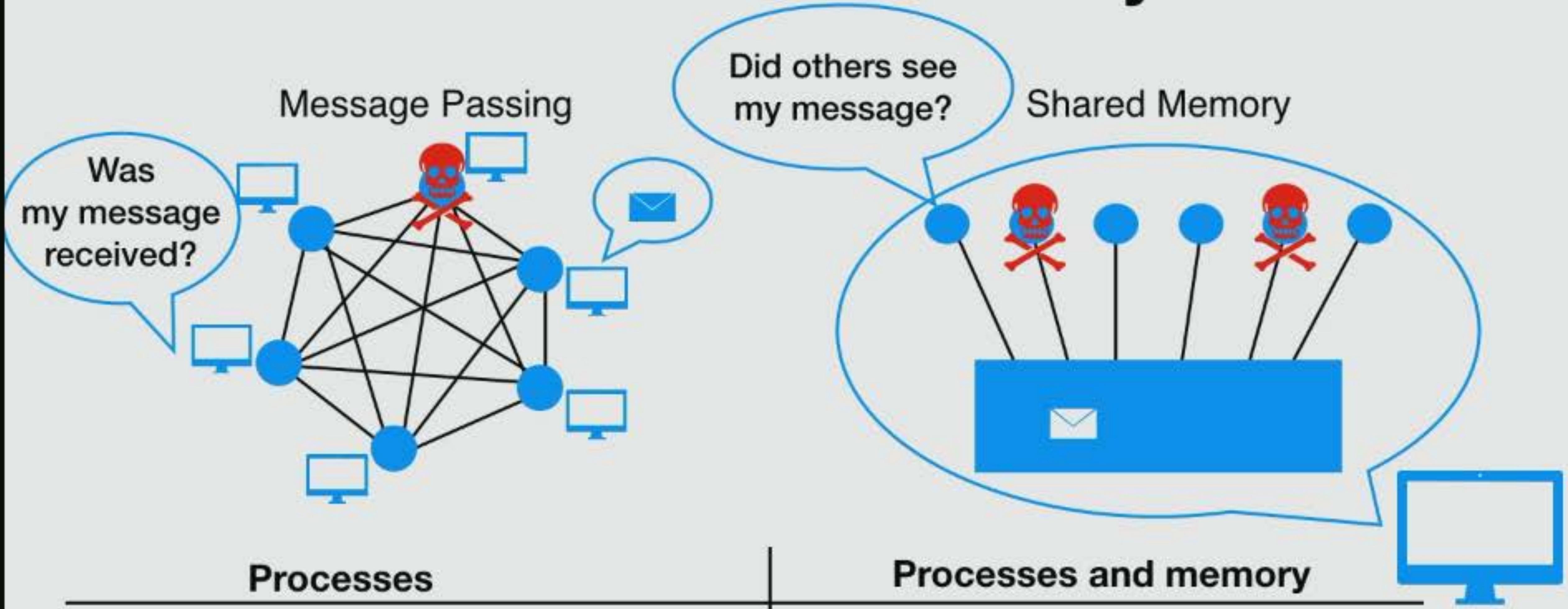
Message Passing vs Shared Memory





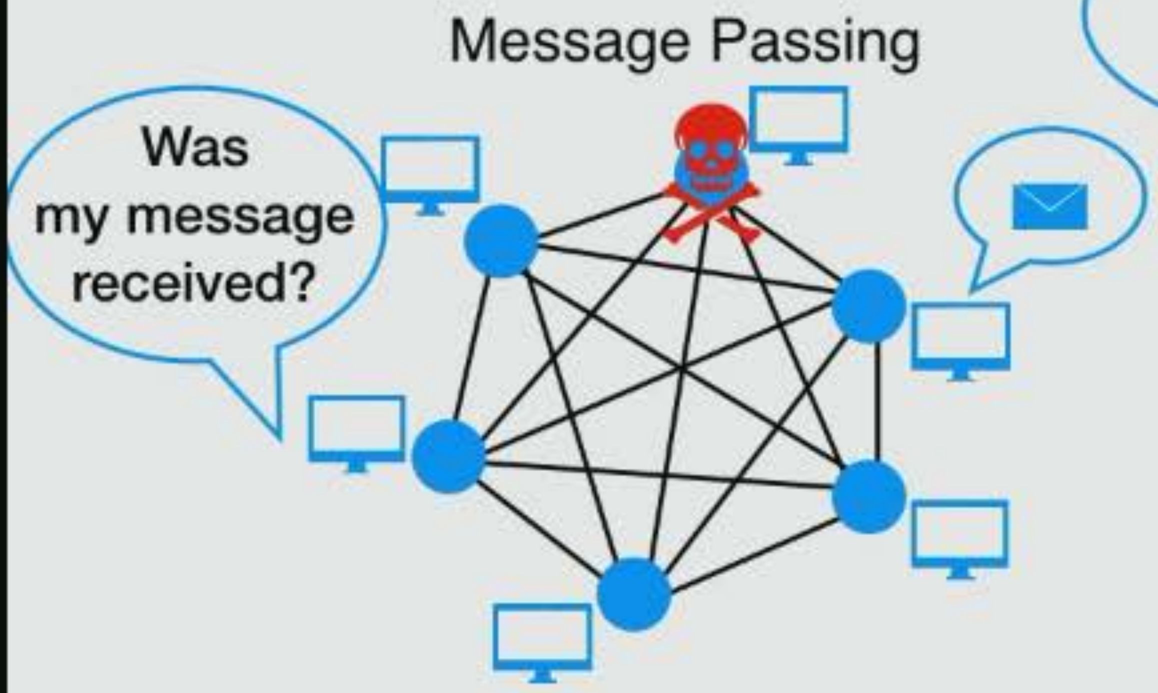
Processes	Processes and memory
Rely on other processes to know if communication was successful	**Rely on memory to know if communication was successful**
Need “ack” from each process	**Single “ack” from memory**
“Ack” can have arbitrary information	

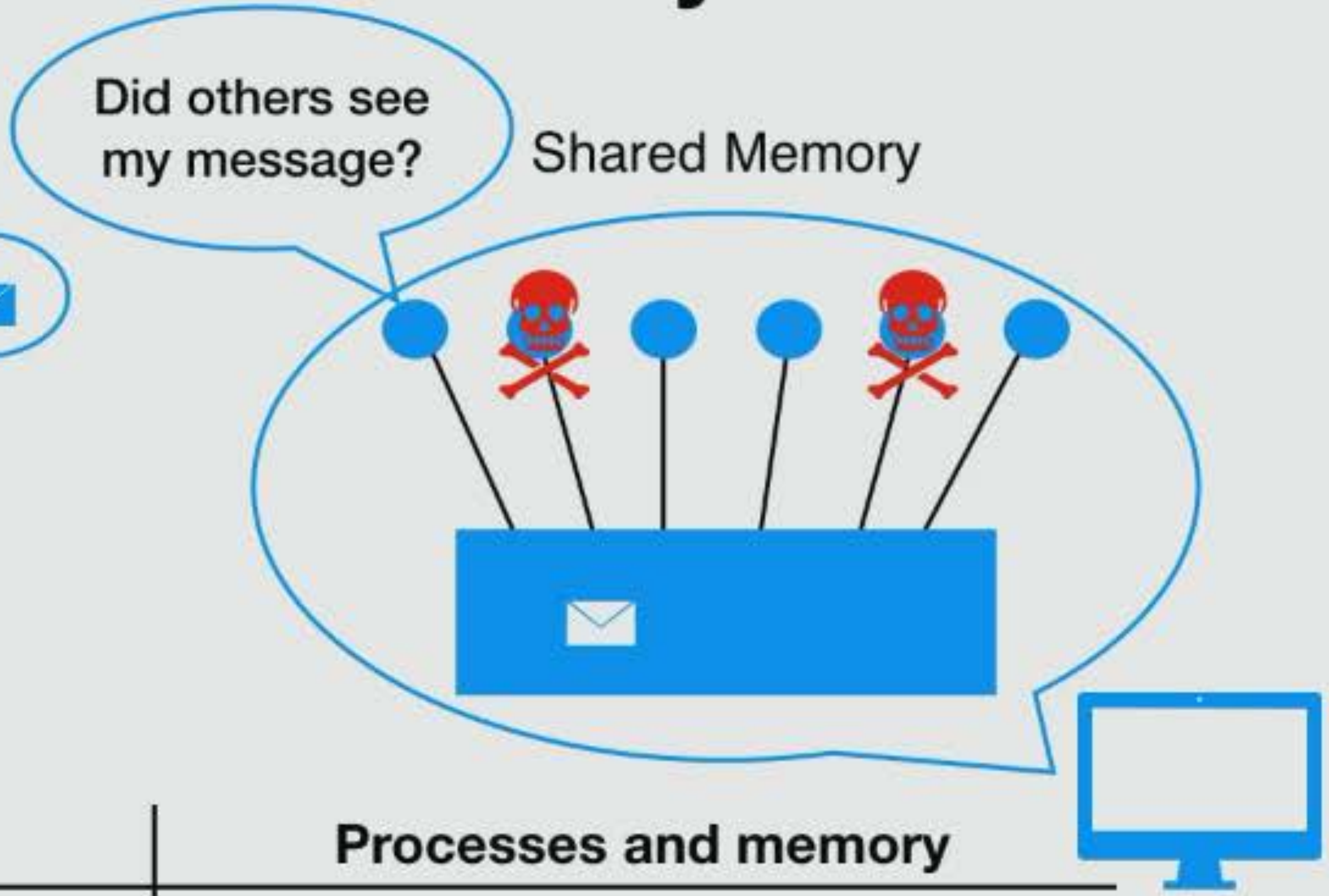
Message Passing vs Shared Memory



Processes	Processes and memory
Rely on other processes to know if communication was successful	Rely on memory to know if communication was successful
Need “ack” from each process	Single “ack” from memory
“Ack” can have arbitrary information	“Ack” has little new information

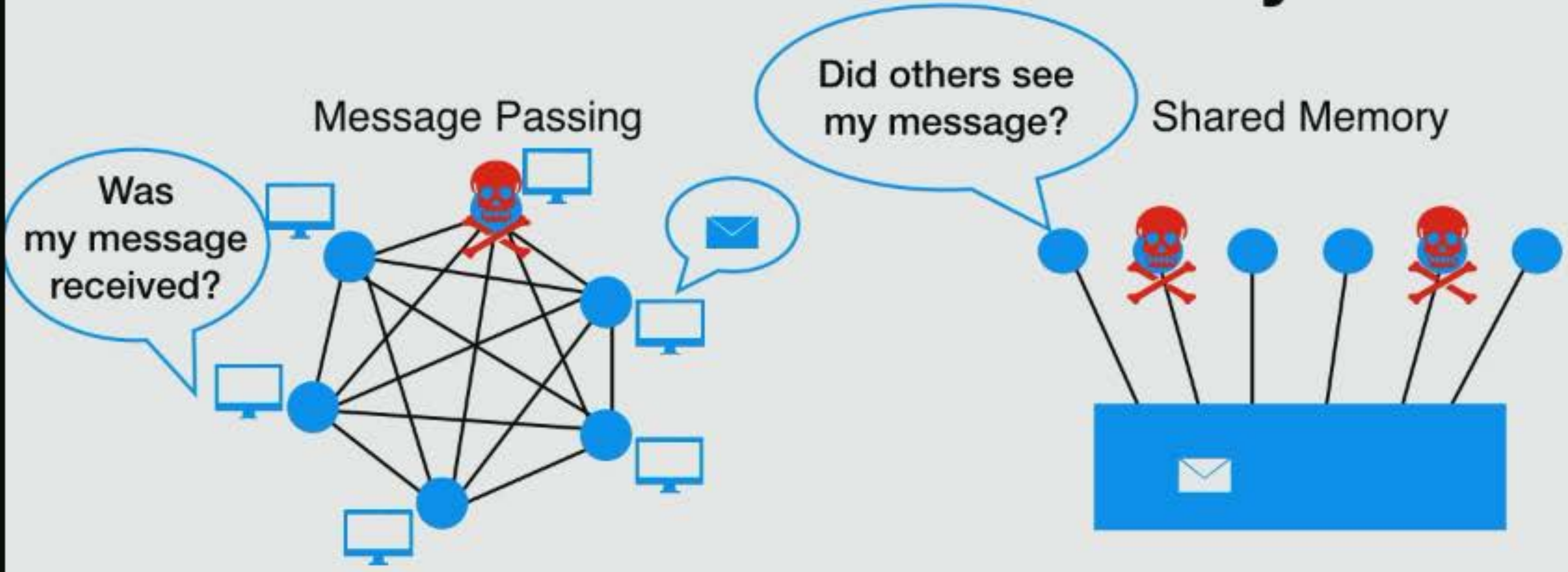
Message Passing vs Shared Memory





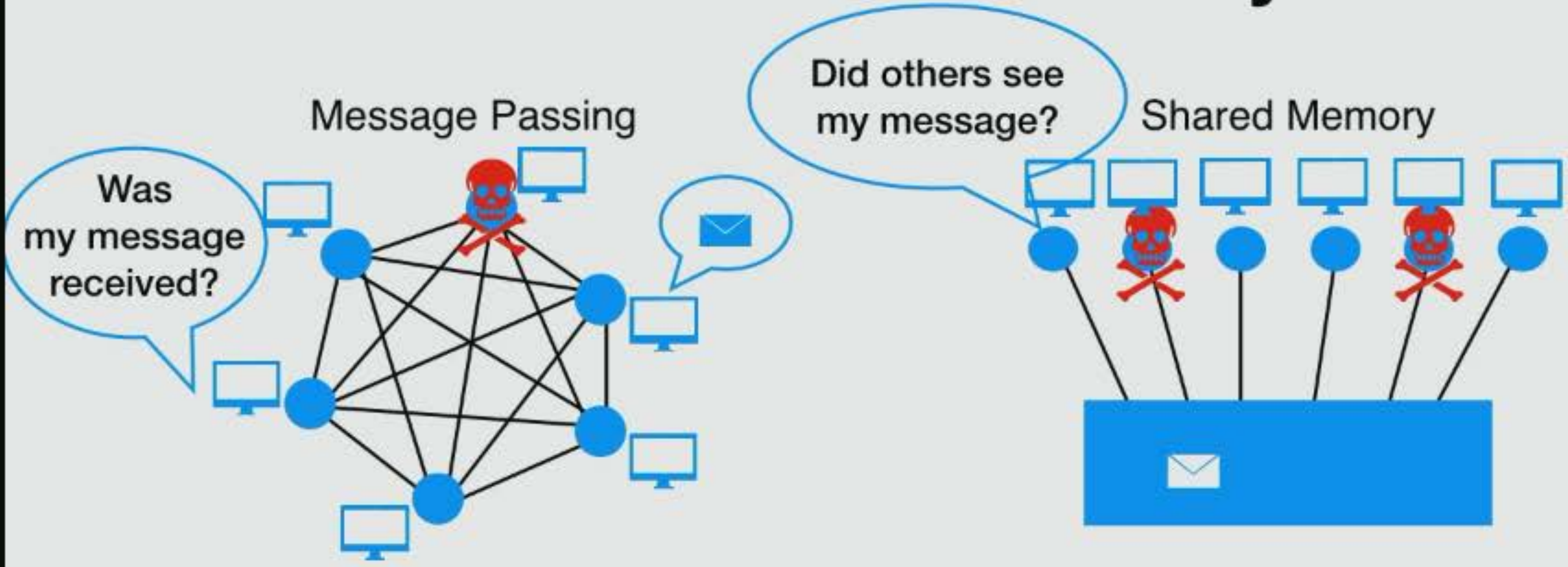
Processes			Processes and memory
Rely on other processes to know if communication was successful	☹	☺	Rely on memory to know if communication was successful
Need “ack” from each process	☹	☺	Single “ack” from memory
“Ack” can have arbitrary information	☺	☹	“Ack” has little new information

Message Passing vs Shared Memory



Processes	Processes and memory
Rely on other processes to know if communication was successful	Rely on memory to know if communication was successful
Need "ack" from each process	Single "ack" from memory
"Ack" can have arbitrary information	"Ack" has little new information

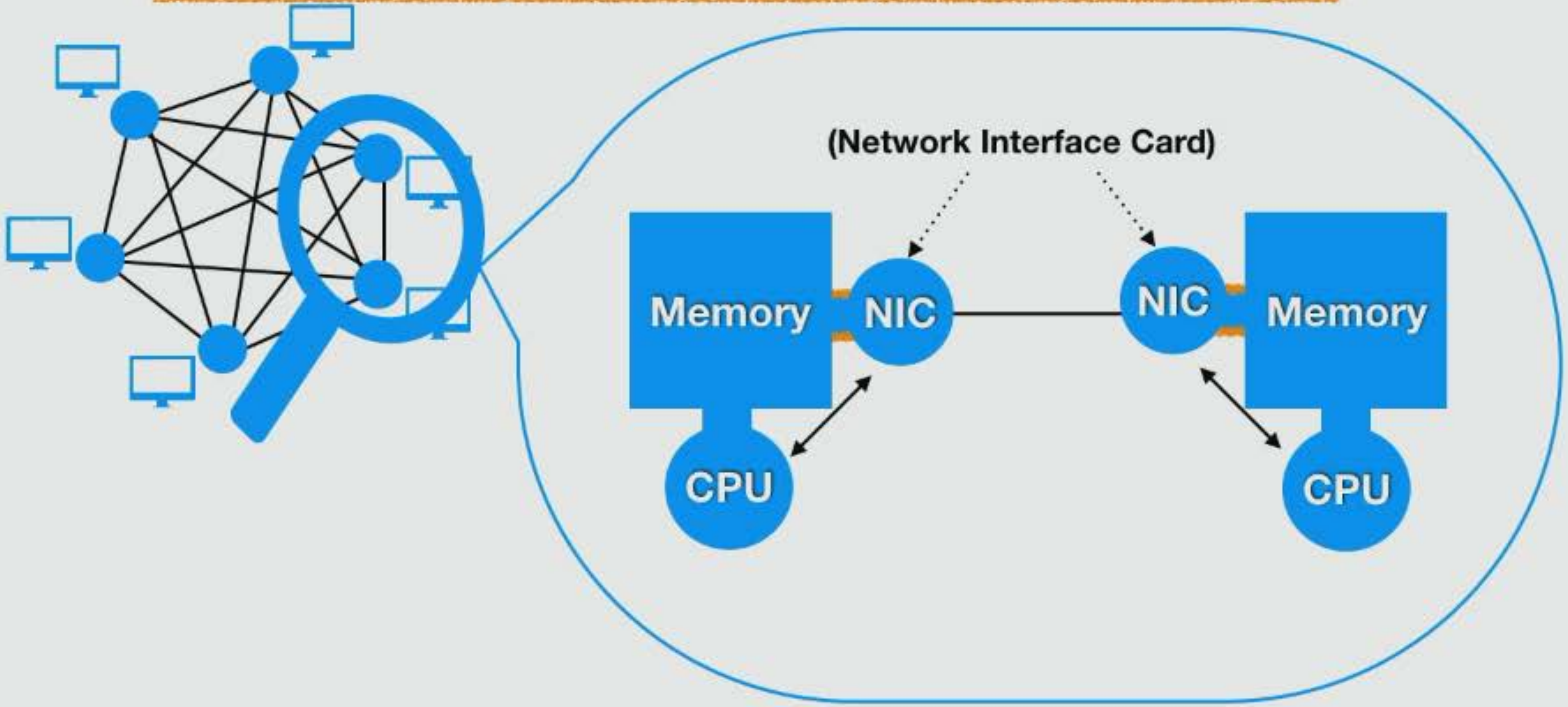
Message Passing vs Shared Memory



Processes			Processes and memory
Rely on other processes to know if communication was successful	☹	☺	Rely on memory to know if communication was successful
Need "ack" from each process	☹	☺	Single "ack" from memory
"Ack" can have arbitrary information	☺	☹	"Ack" has little new information

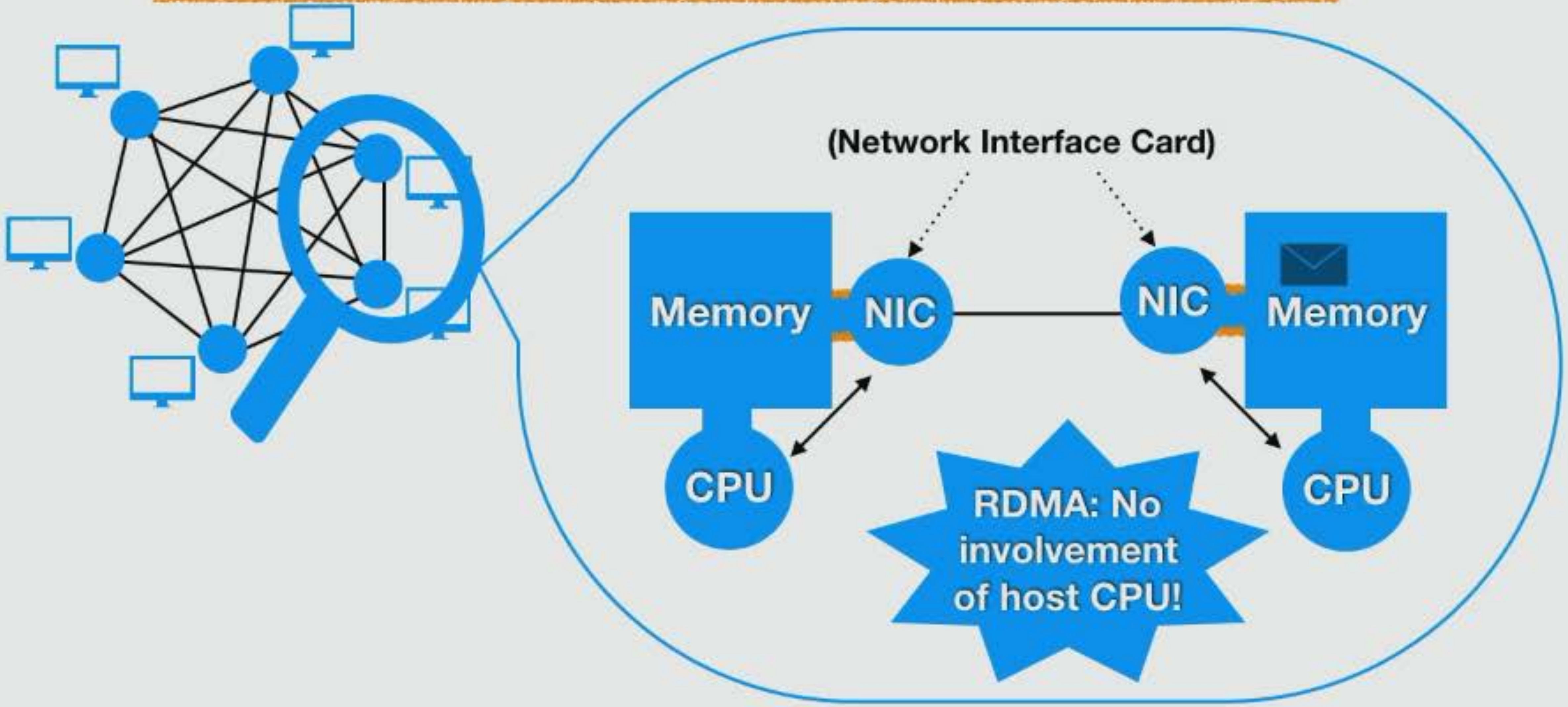
RDMA: Messages and Memory

Remote Direct Memory Access (RDMA)



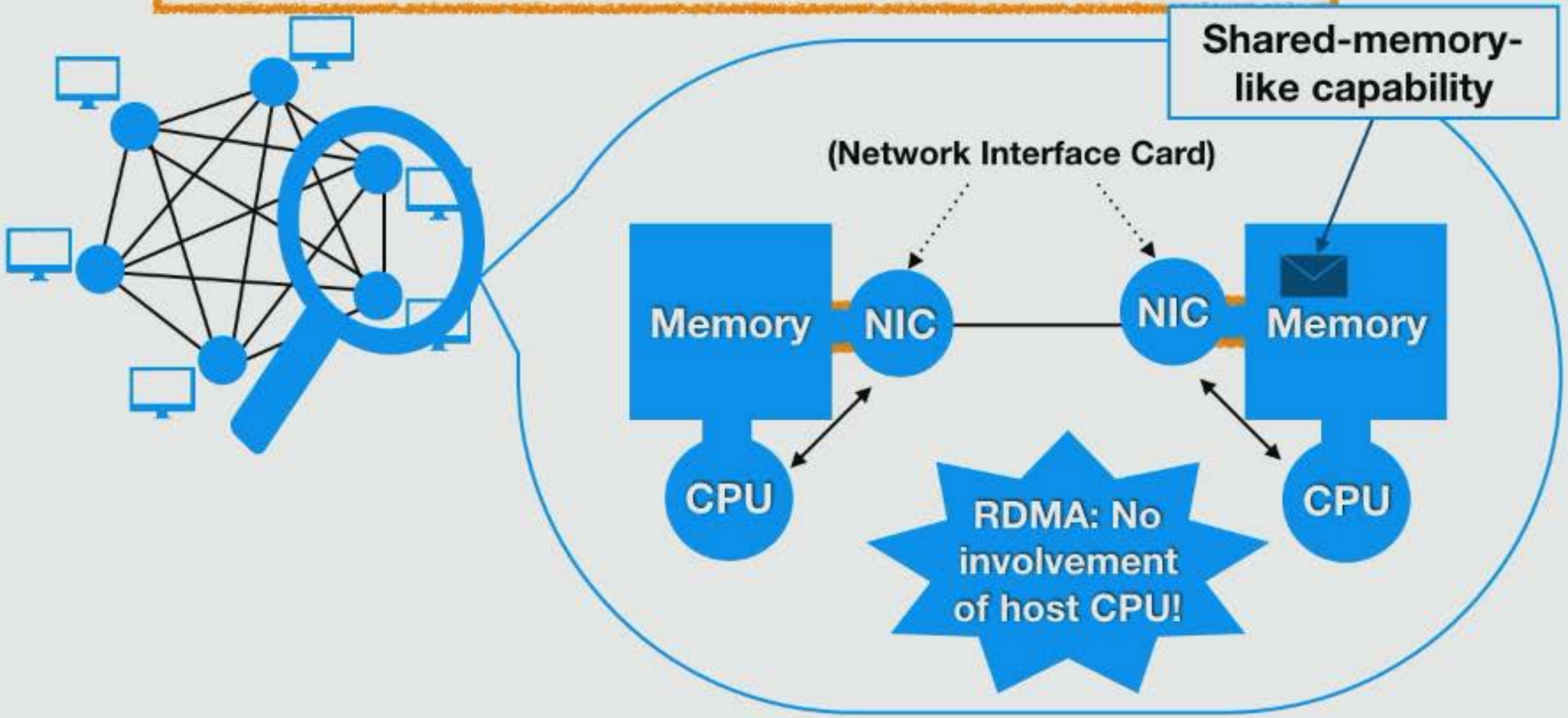
RDMA: Messages and Memory

Remote Direct Memory Access (RDMA)



RDMA:
Messages and Memory

Remote Direct Memory Access (RDMA)



Fault Tolerance

Performance

Scalability

Is RDMA fundamentally better than other communication mechanisms?

Message Passing

Shared Memory

Main Take-Away

RDMA improves tradeoff between fault tolerance, performance and scalability

Main Take-Away

Byzantine **or** ***crash*** **failures of processes**

RDMA improves tradeoff between fault tolerance, performance and scalability

Main Take-Away

Byzantine or crash failures of processes

RDMA improves tradeoff between fault tolerance, performance and scalability

Common-case running time

Main Take-Away

Byzantine or *crash* failures of processes

RDMA improves tradeoff between fault tolerance, performance and scalability

Common-case running time

Common case:

- Synchronous
- No Failures

Running time (agreement):

- Time until *first process* decides

Main Take-Away

Byzantine or *crash* failures of processes

RDMA improves tradeoff between fault tolerance, performance and scalability

Common-case running time

Best case performance
Worst case resilience

Common case:
- Synchronous
- No Failures

Running time (agreement):
- Time until *first process* decides

Outline

- RDMA details and previous results
- Improving fault tolerance and performance
- Improving scalability

Previous Results

Previous Results

n = num processes f = num failures	Shared Memory	Message Passing

Previous Results

n = num processes f = num failures		Shared Memory	Message Passing
Fault Tolerance	Crash	n>f	n>2f
	Byzantine	N/A	n>3f

Previous Results

n = num processes f = num failures		Shared Memory	Message Passing
Fault Tolerance	Crash	n>f	n>2f
	Byzantine	N/A	n>3f

Previous Results

n = num processes f = num failures		Shared Memory	Message Passing
Fault Tolerance	Crash	n>f	n>2f
	Byzantine	N/A	n>3f

Previous Results

n = num processes f = num failures		Shared Memory	Message Passing
Fault Tolerance	Crash	n>f	n>2f
	Byzantine	N/A	n>3f

Previous Results

n = num processes f = num failures		Shared Memory	Message Passing
Fault Tolerance	Crash	n>f 	n>2f 
	Byzantine	N/A	n>3f 
Complexity* (Best Case Round Trips)		2	1

Previous Results

n = num processes f = num failures		Shared Memory	Message Passing
Fault Tolerance	Crash	n>f	n>2f
	Byzantine	N/A	n>3f
Complexity* (Best Case Round Trips)		2	1
Scalability (processes in network)		10-100	10,000 - 100,000

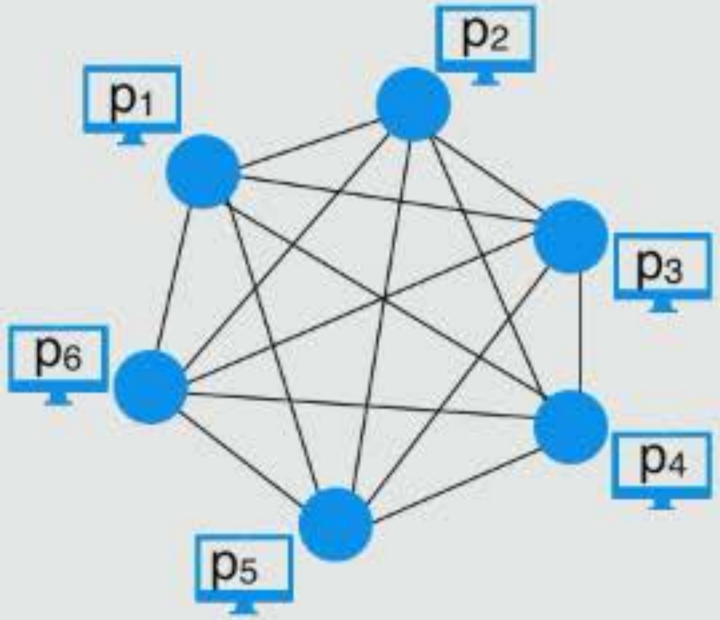
Previous Results

n = num processes f = num failures		Shared Memory	Message Passing	RDMA Full [ABGMZ'19]
Fault Tolerance	Crash	n>f	n>2f	n>f
	Byzantine	N/A	n>3f	n>2f
Complexity* (Best Case Round Trips)		2	1	1
Scalability (processes in network)		10-100	10,000 - 100,000	10-100

Previous Results

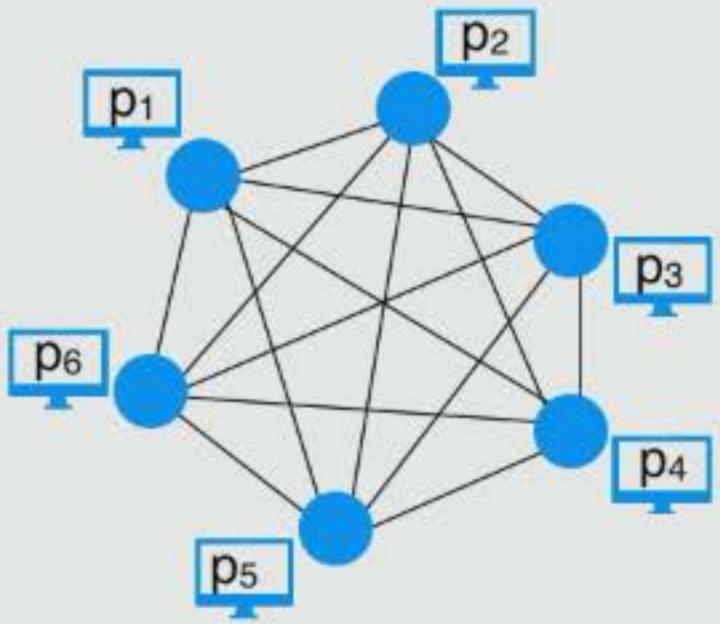
n = num processes f = num failures		Shared Memory	Message Passing	RDMA Full [ABGMZ'19]	RDMA Scale [ABCGPT'18]
Fault Tolerance	Crash	n>f	n>2f	n>f	n>f+x (x∈[0,f])
	Byzantine	N/A	n>3f	n>2f	-
Complexity* (Best Case Round Trips)		2	1	1	-
Scalability (processes in network)		10-100	10,000 - 100,000	10-100	10-100,000

Data Center Technology: RDMA



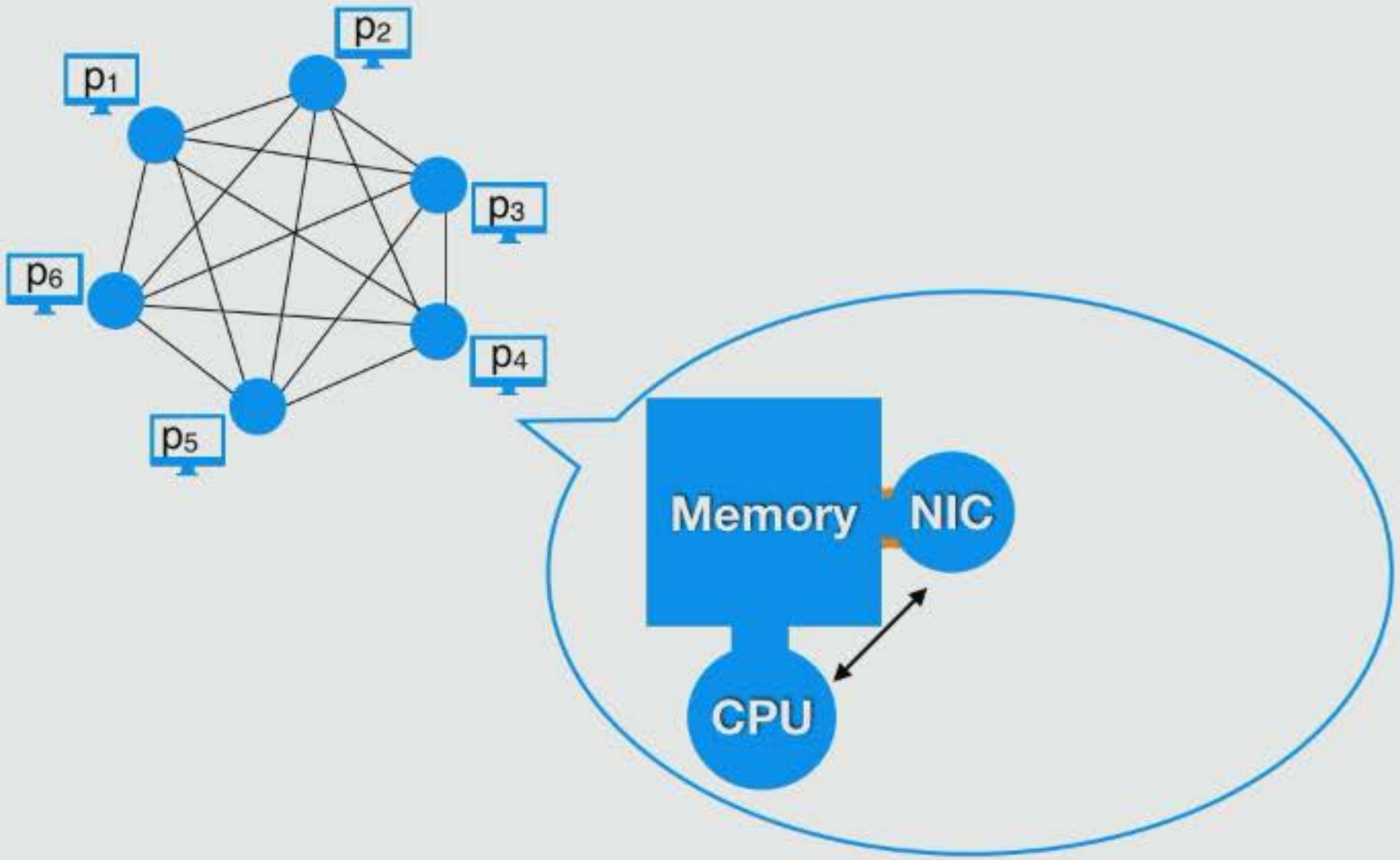
Data Center Technology: RDMA

- Can choose RDMA **connections**



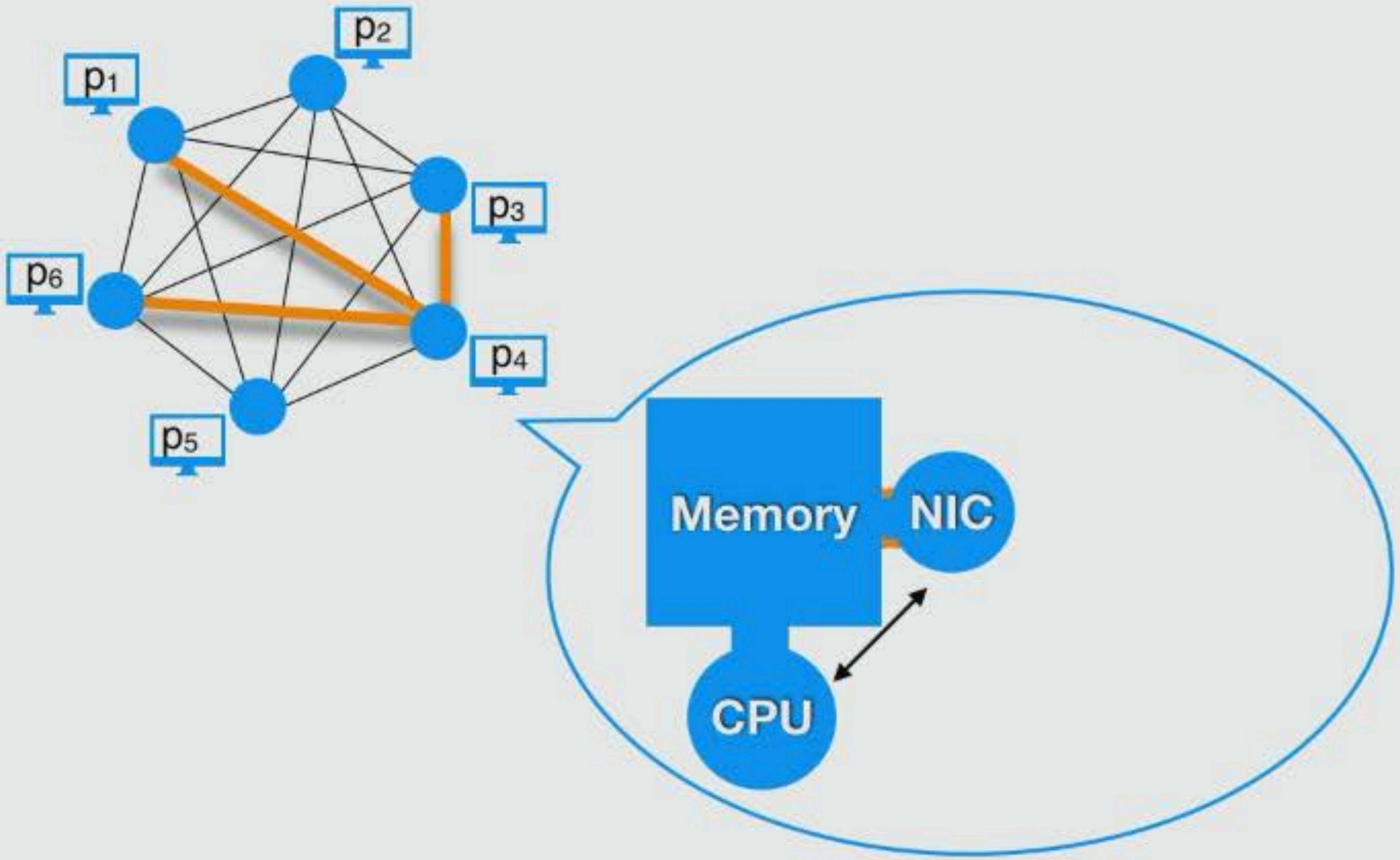
Data Center Technology: RDMA

- Can choose RDMA **connections**



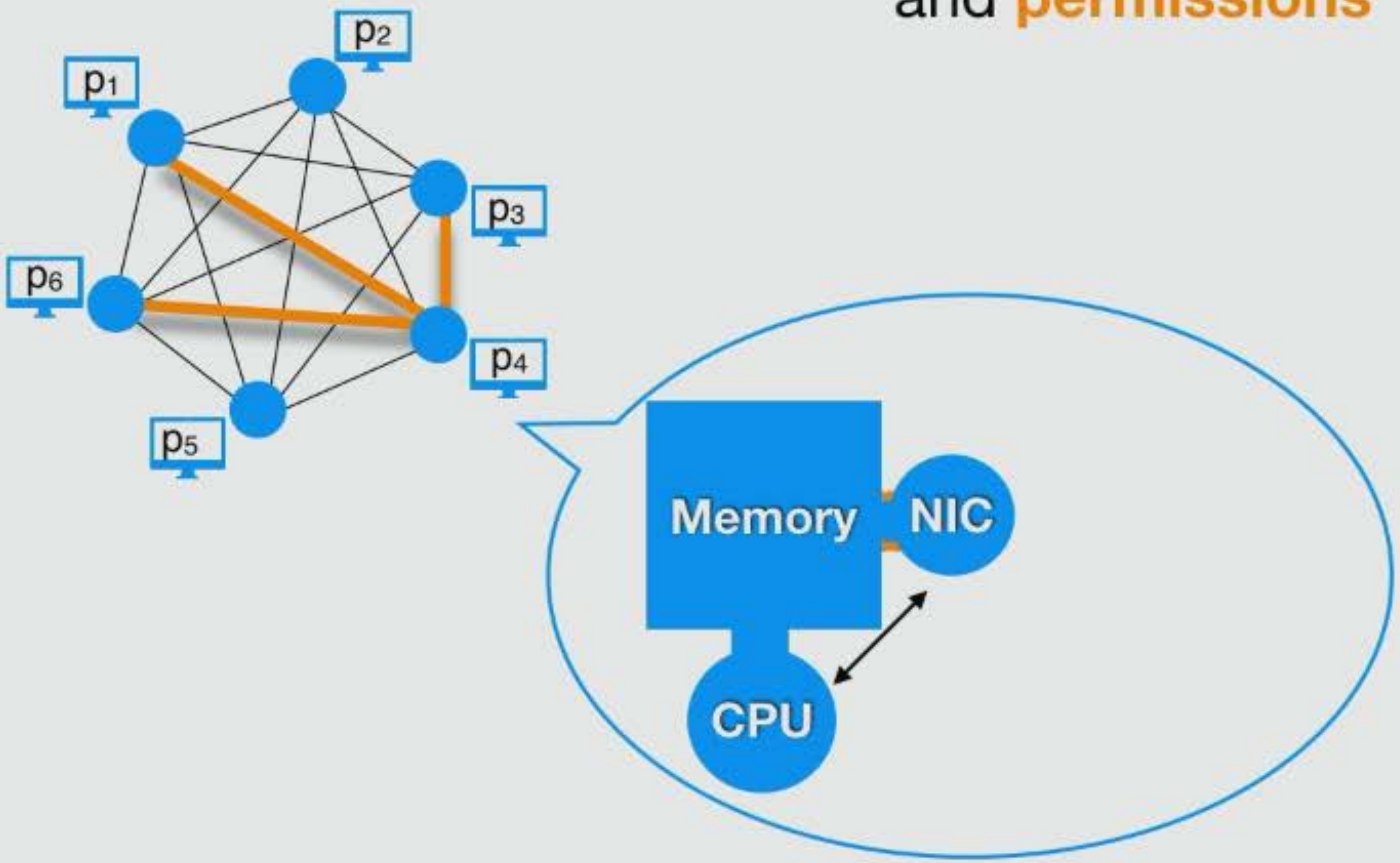
Data Center Technology: RDMA

- Can choose RDMA **connections**



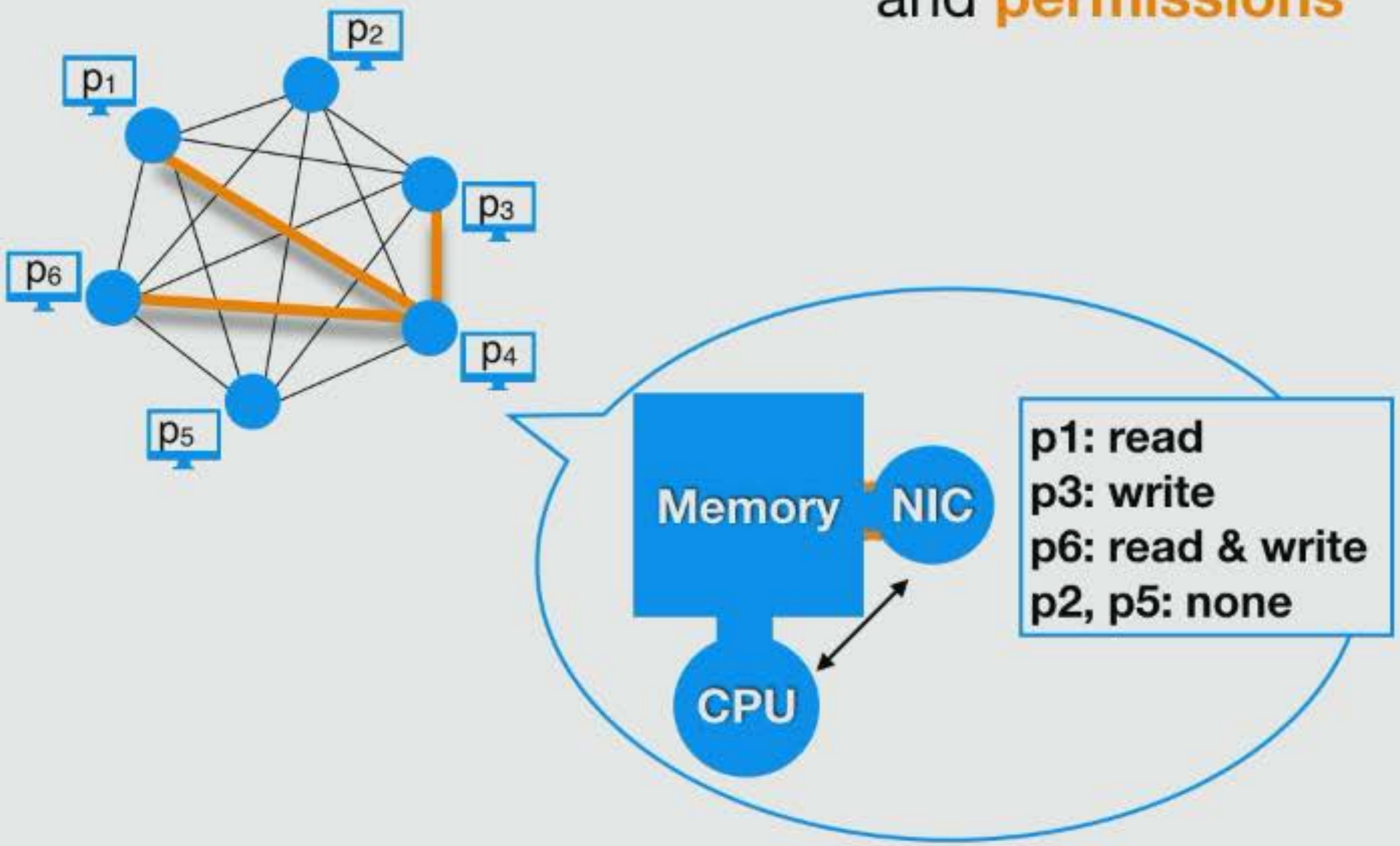
Data Center Technology: RDMA

- Can choose RDMA **connections** and **permissions**

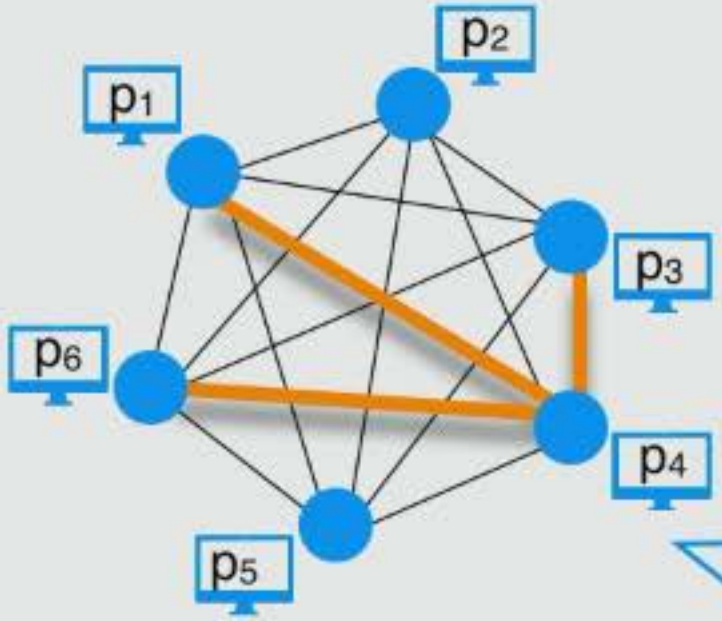


Data Center Technology: RDMA

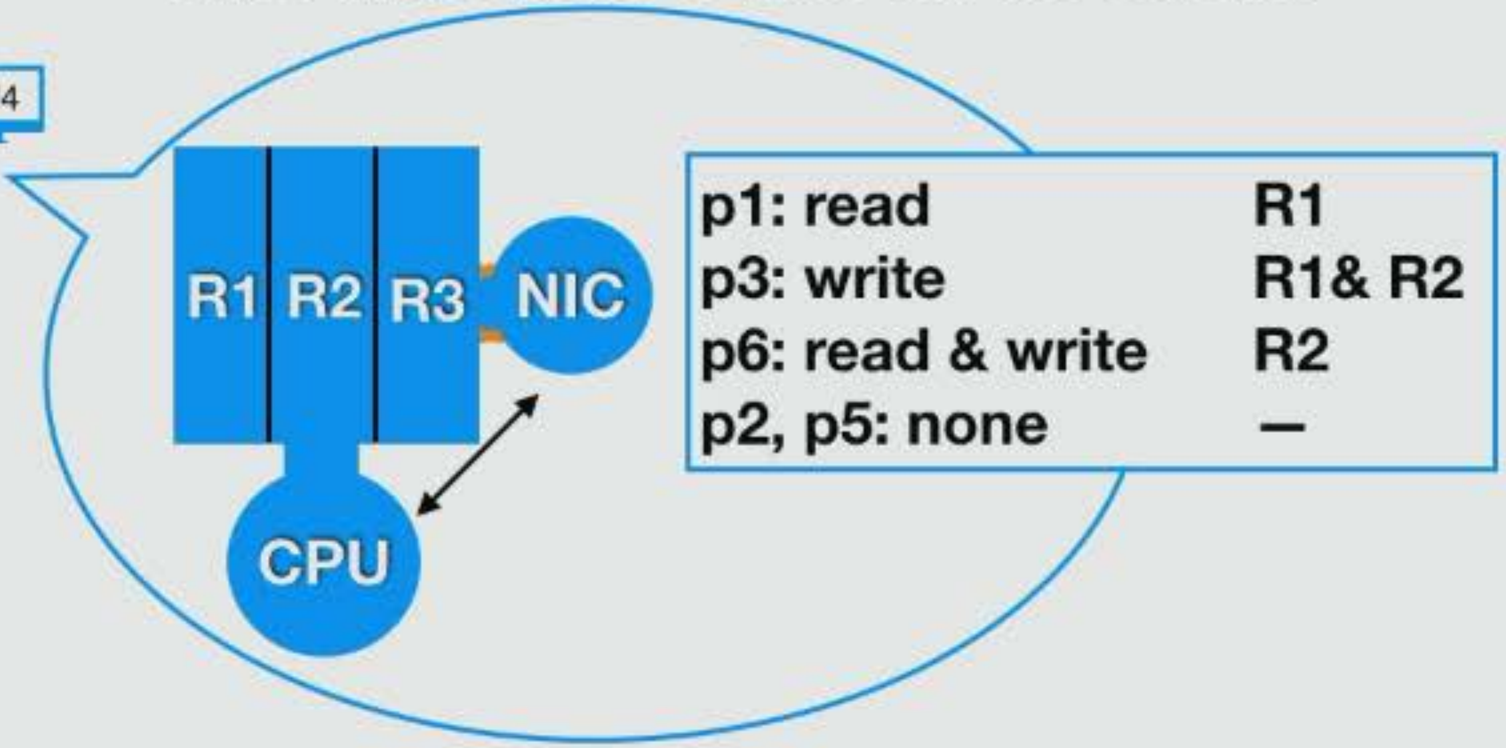
- Can choose RDMA **connections** and **permissions**



Data Center Technology: RDMA



- Can choose RDMA **connections** and **permissions**
- Can give different permissions for different **memory regions**
- NIC must store this on its cache



RDMA Scalability

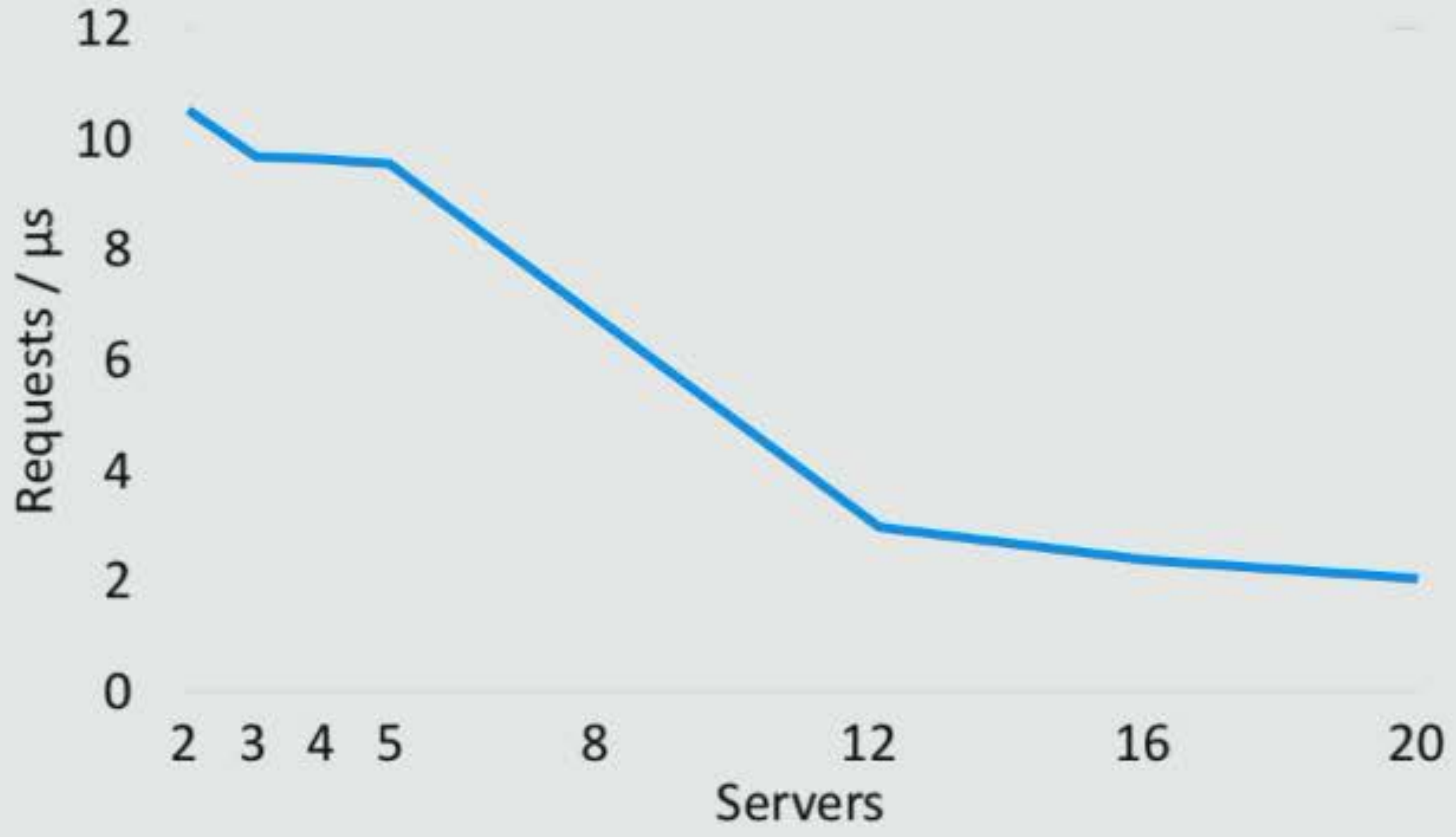


Figure 5: Impact of connection multiplexing

[DragojevicNarayananHodsonCastro'14]

RDMA Scalability

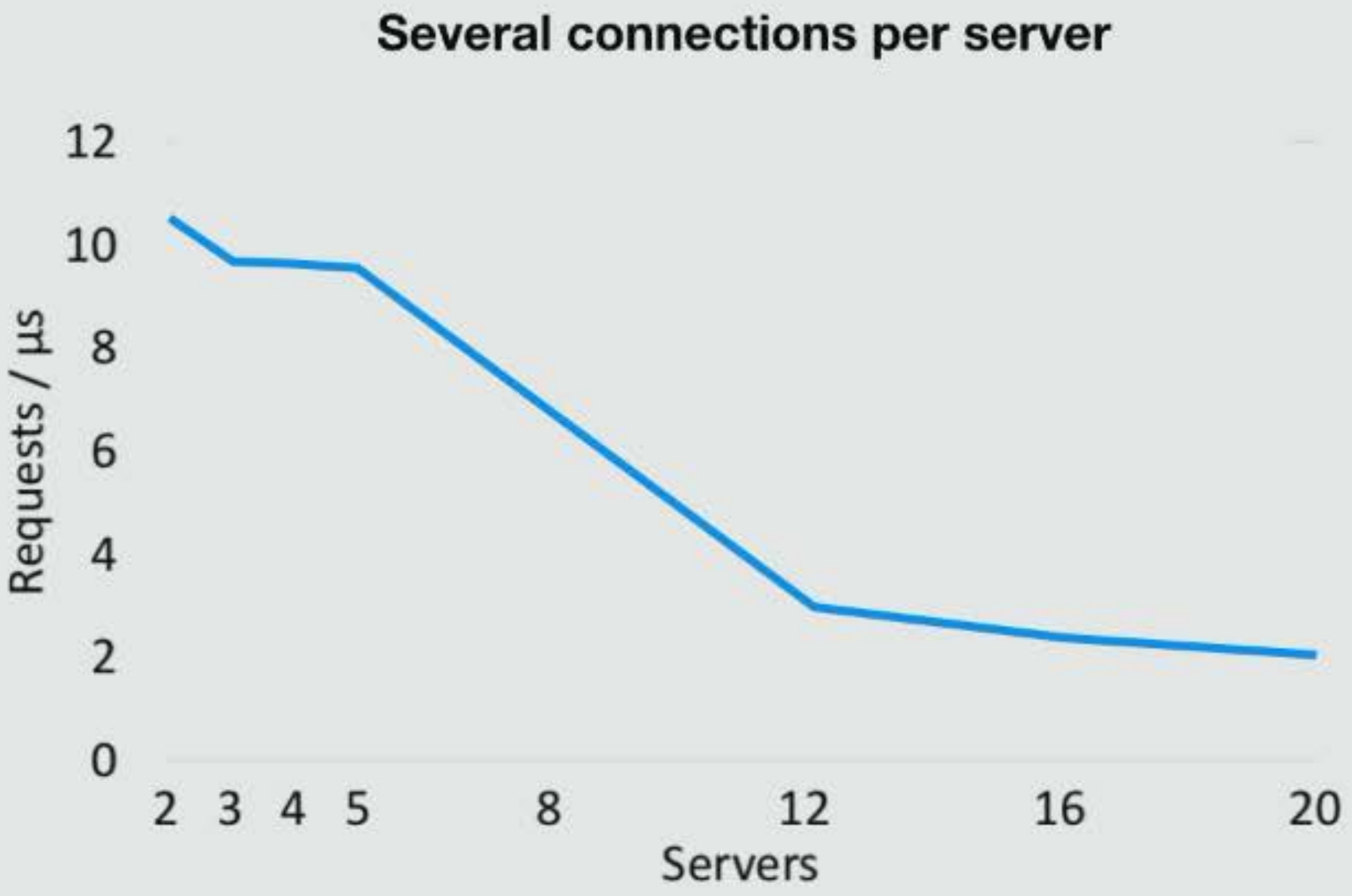


Figure 5: Impact of connection multiplexing

[DragojevicNarayananHodsonCastro'14]

RDMA Scalability

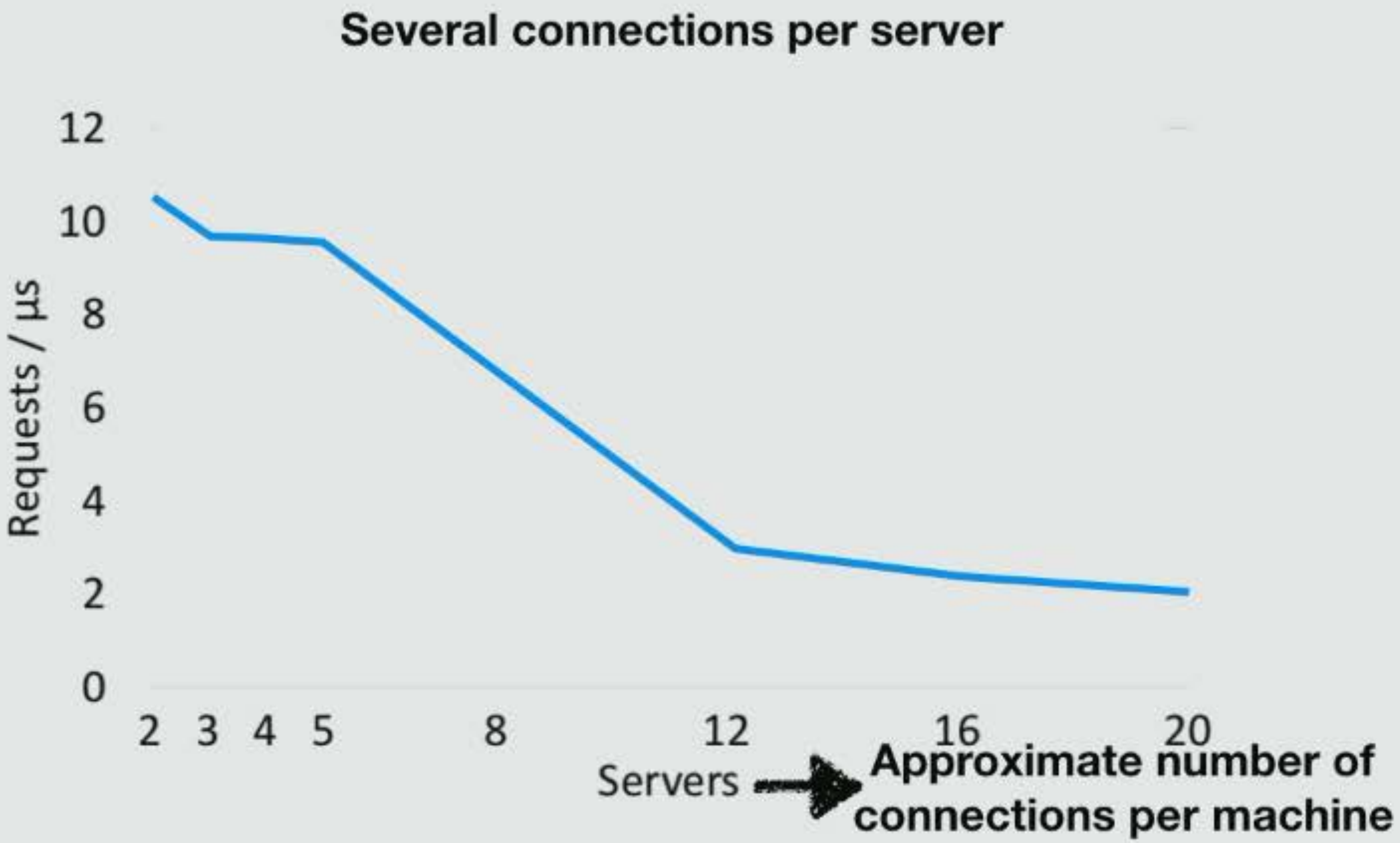


Figure 5: Impact of connection multiplexing

[DragojevicNarayananHodsonCastro'14]

RDMA Scalability

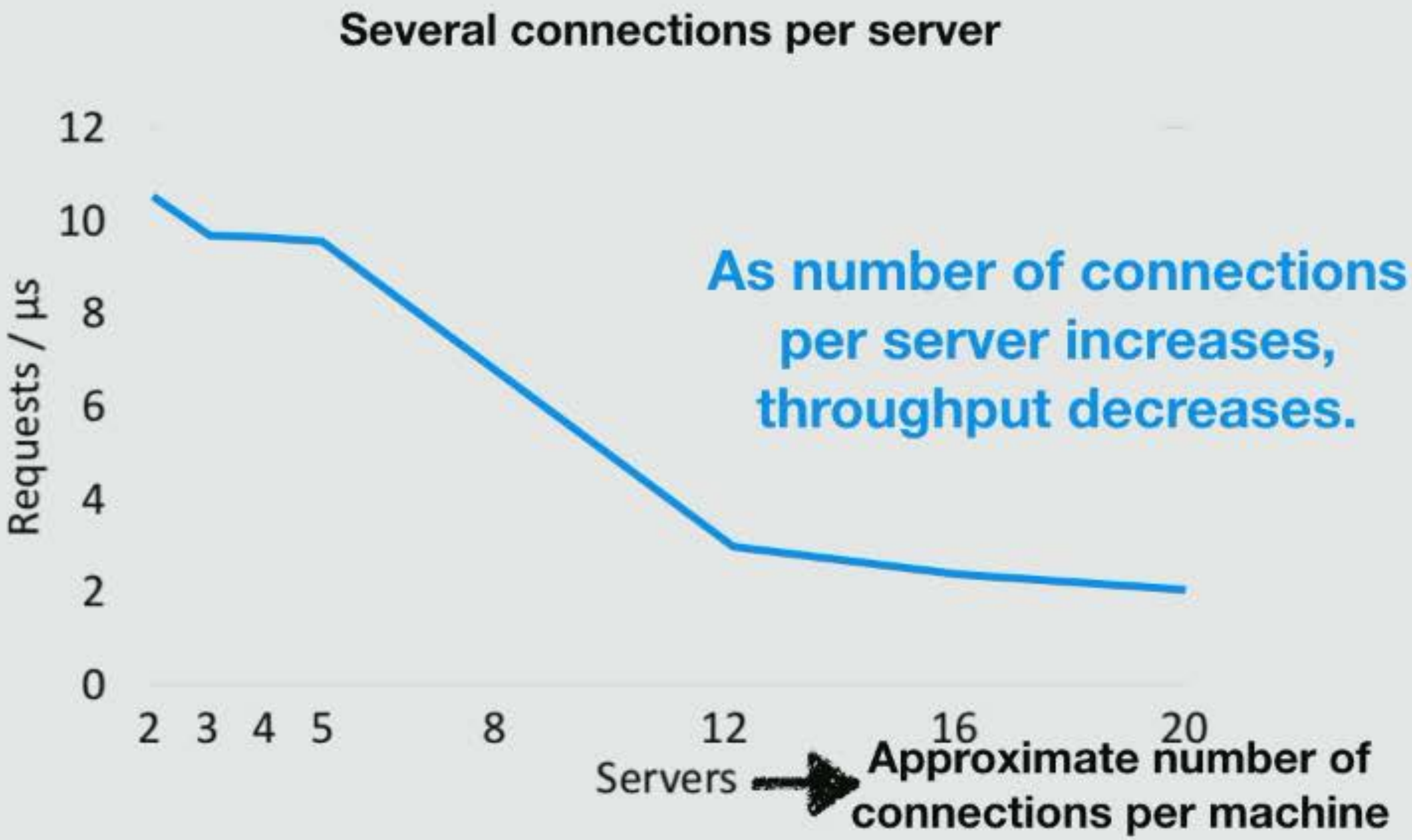


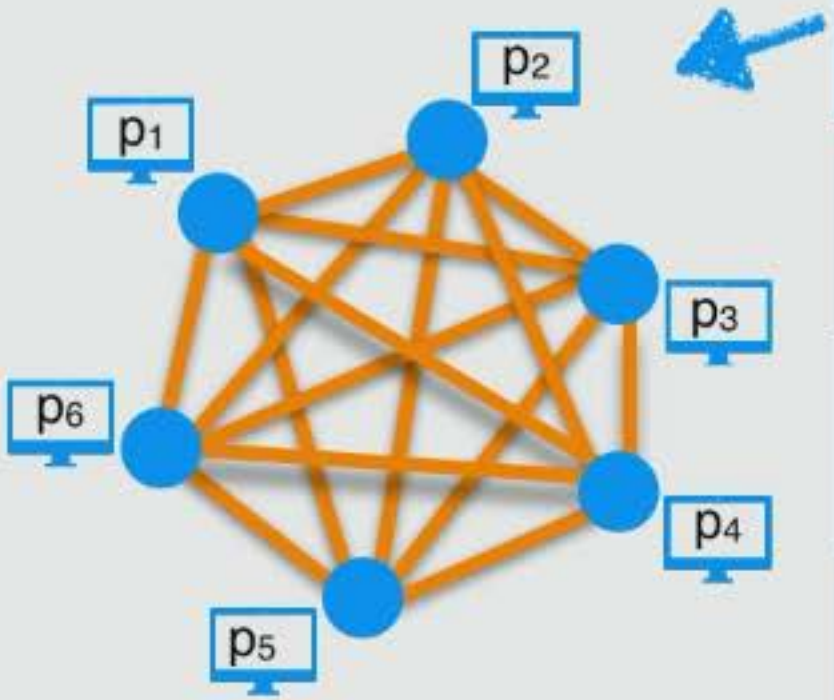
Figure 5: Impact of connection multiplexing

[DragojevicNarayananHodsonCastro'14]

Modeling Scalability

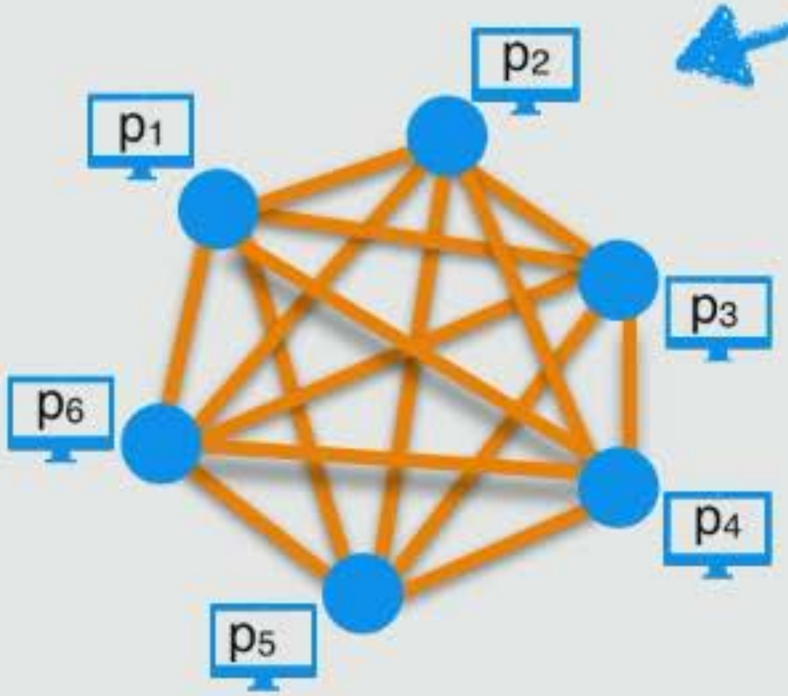
RDMA Full [ABGMZ'19]	RDMA Scale [ABCGPT'18]
n>f	n>f+x (xϵ[0,f])
n>2f	-
1	-
10-100	10-100,000

Modeling Scalability

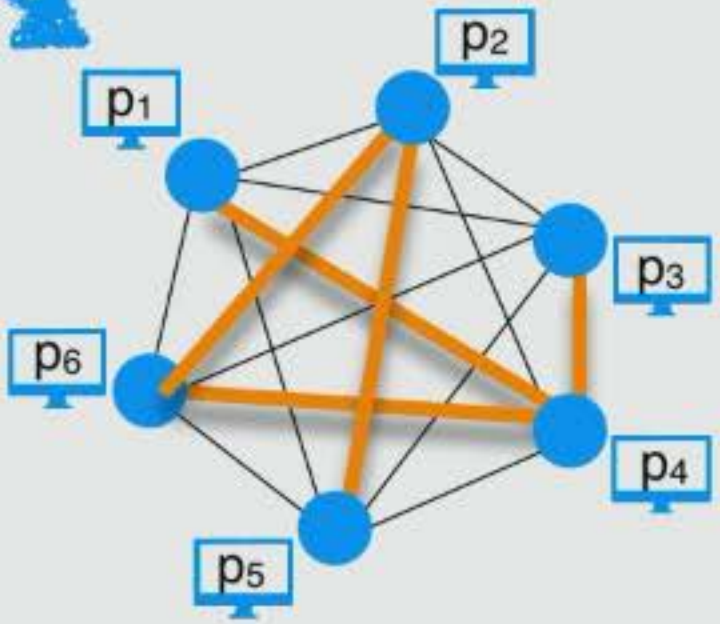


RDMA Full [ABGMZ'19]	RDMA Scale [ABCGPT'18]
n>f	n>f+x (xϵ[0,f])
n>2f	-
1	-
10-100	10-100,000

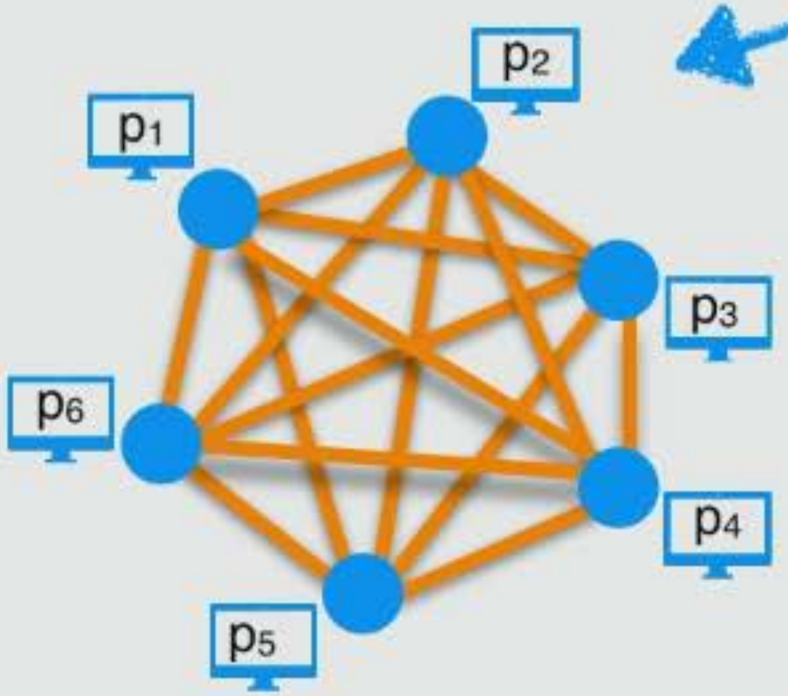
Modeling Scalability



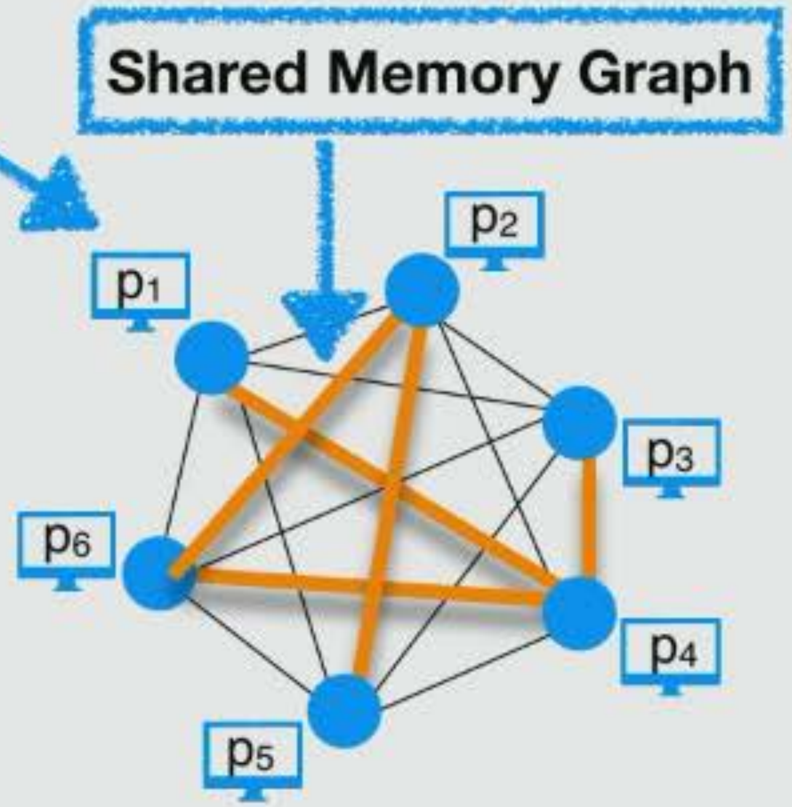
RDMA Full [ABGMZ'19]	RDMA Scale [ABCGPT'18]
n>f	n>f+x (xϵ[0,f])
n>2f	-
1	-
10-100	10-100,000



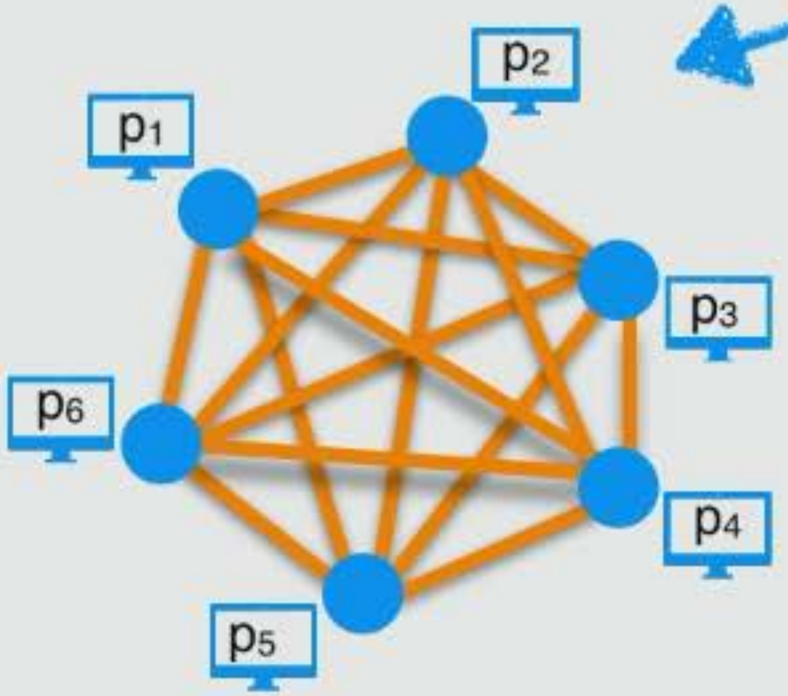
Modeling Scalability



RDMA Full [ABGMZ'19]	RDMA Scale [ABCGPT'18]
n>f	n>f+x (xϵ[0,f])
n>2f	-
1	-
10-100	10-100,000

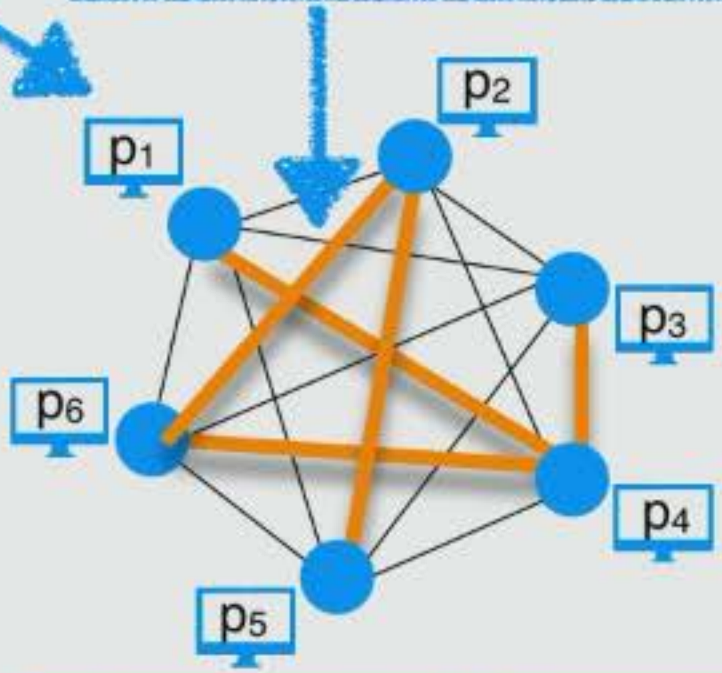


Modeling Scalability



RDMA Full [ABGMZ'19]	RDMA Scale [ABCGPT'18]
n>f	n>f+x (x∈[0,f])
n>2f	-
1	-
10-100	10-100,000

Shared Memory Graph



Fault tolerance will depend on topology of SM graph

Outline

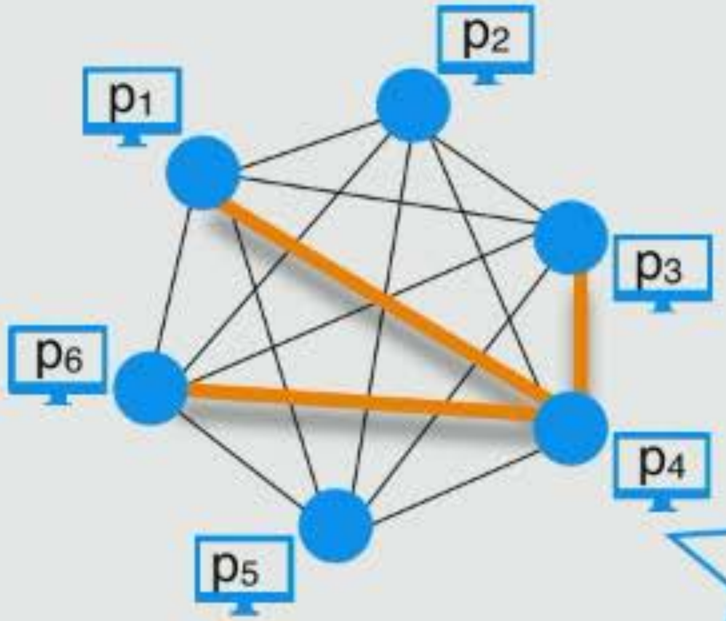
✔ **RDMA details**

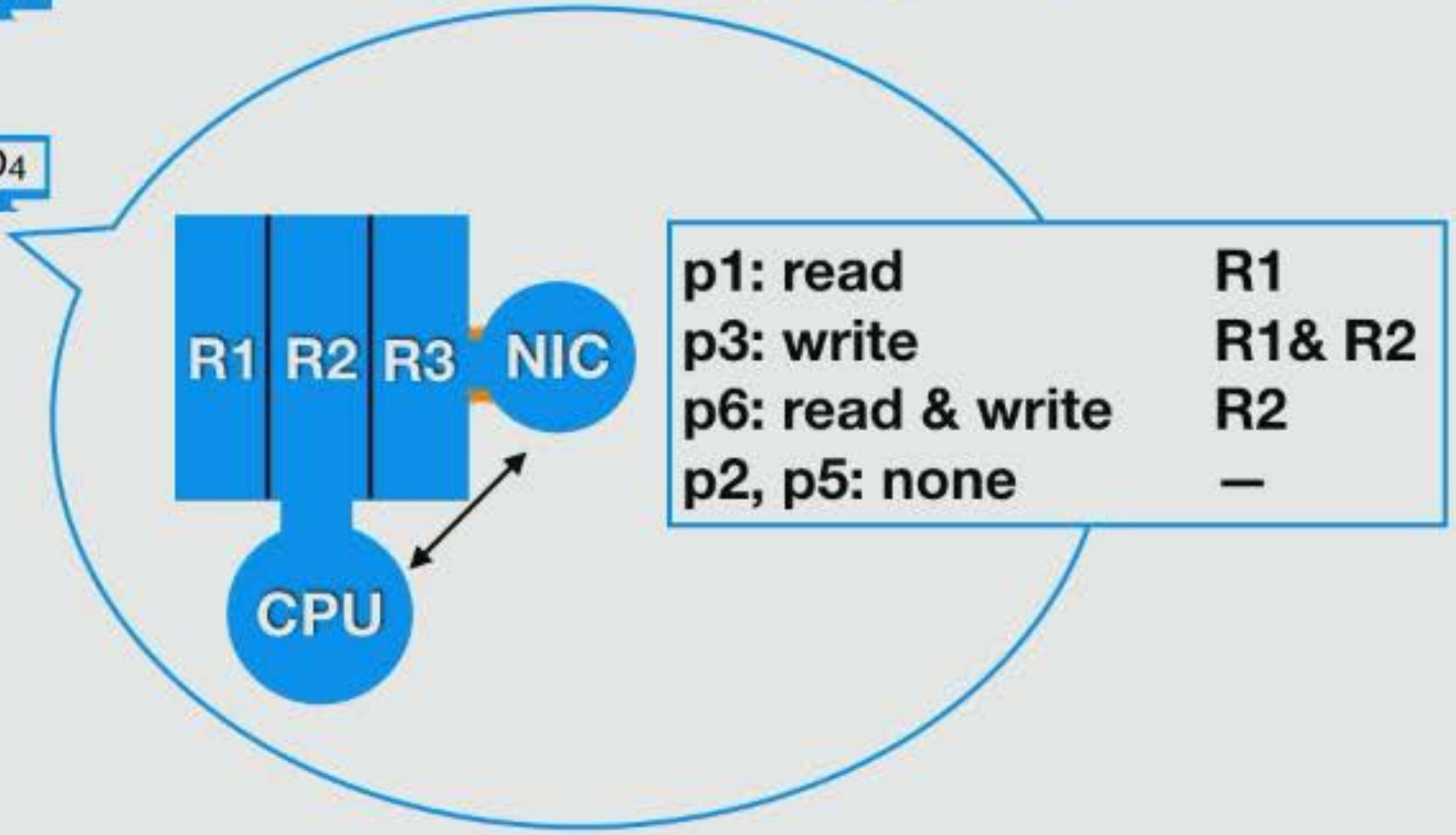
- Setting 1: **RDMA's full power** (complete graph)
  - **Crash-only** algorithm: ***n>f*** tolerant, 1 round-trip
  - **Byzantine** algorithm: ***n>2f*** tolerant, 1 round-trip

- Setting 2: **Scalability: Using RDMA sparingly** (incomplete graph)
  - Crash-only Algorithm: tolerance vs topology

Data Center Technology: RDMA

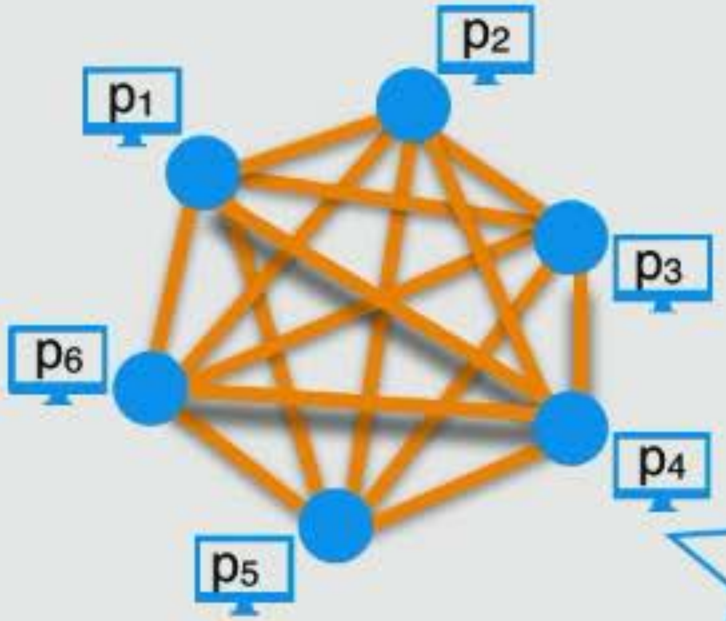
- Can choose RDMA **connections** and **permissions**
- Can give different permissions for different **memory regions**

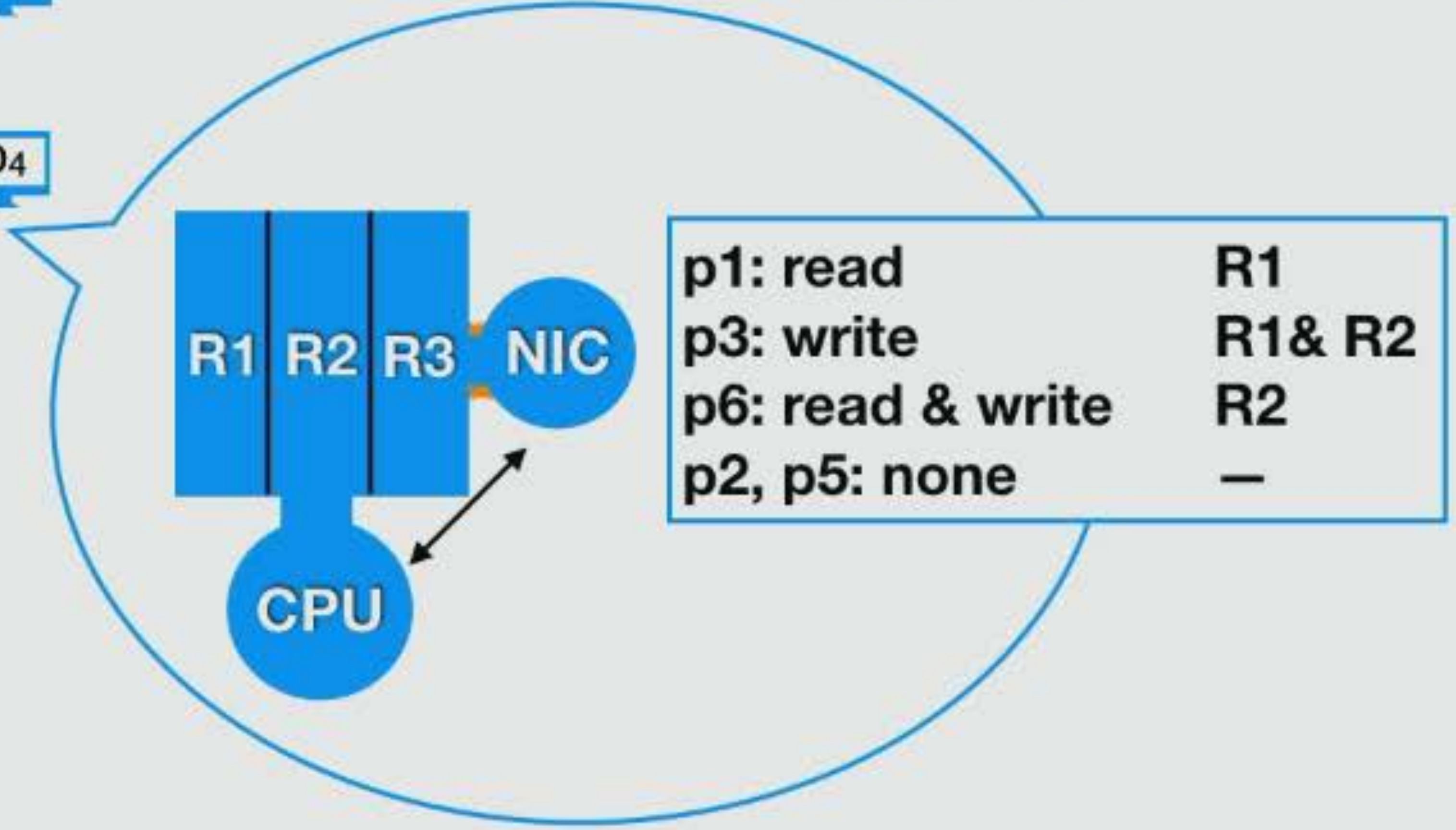




Data Center Technology: RDMA

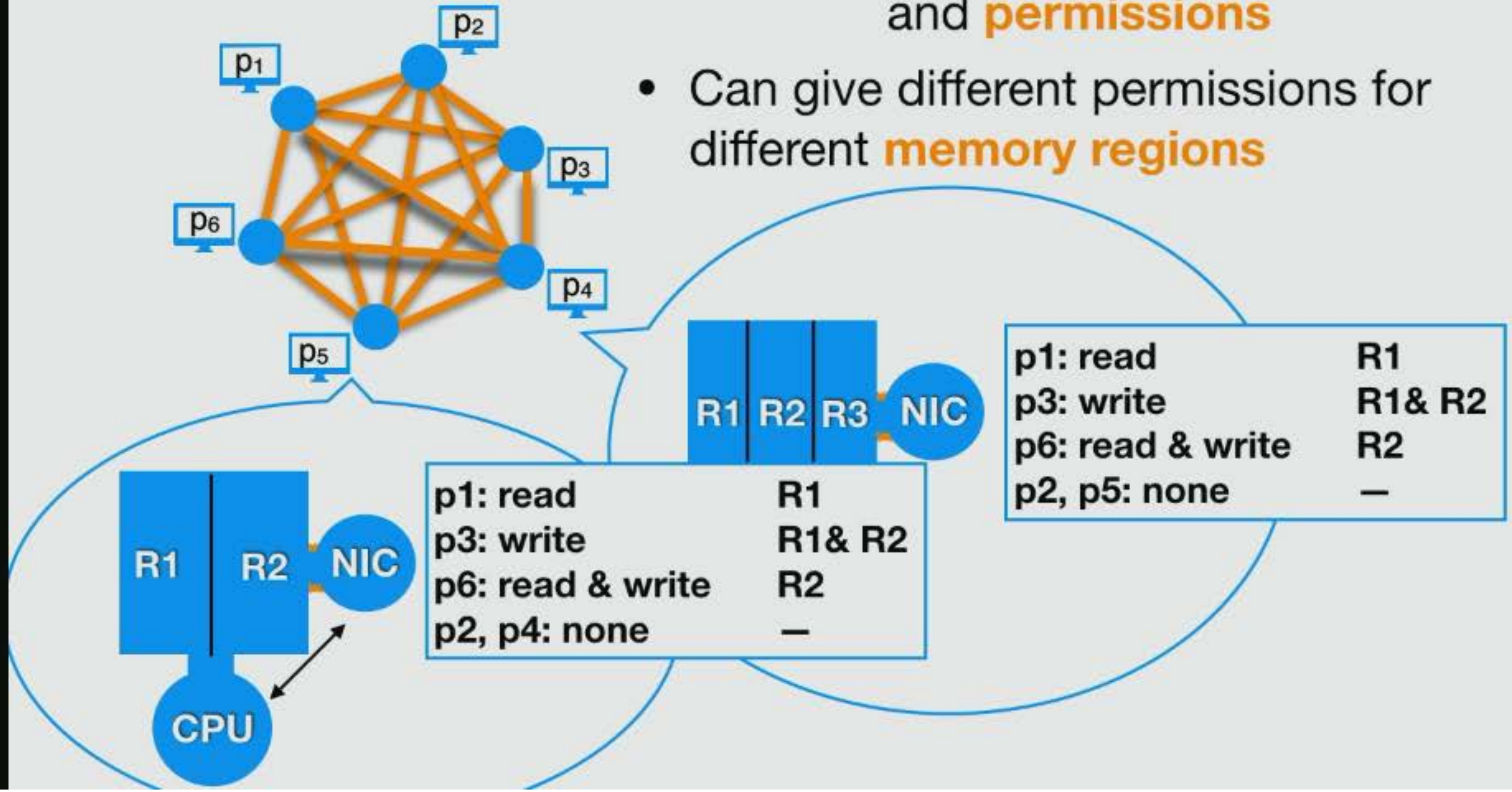
- Can choose RDMA **connections** and **permissions**
- Can give different permissions for different **memory regions**



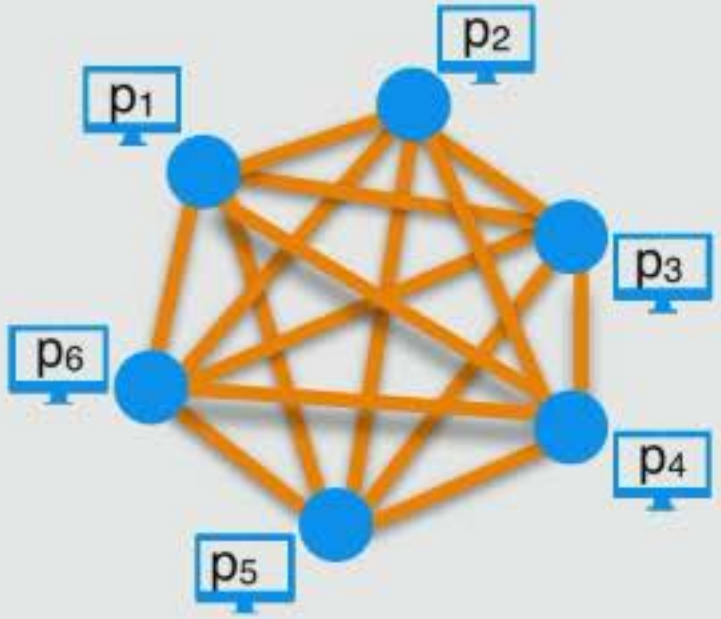


Data Center Technology: RDMA

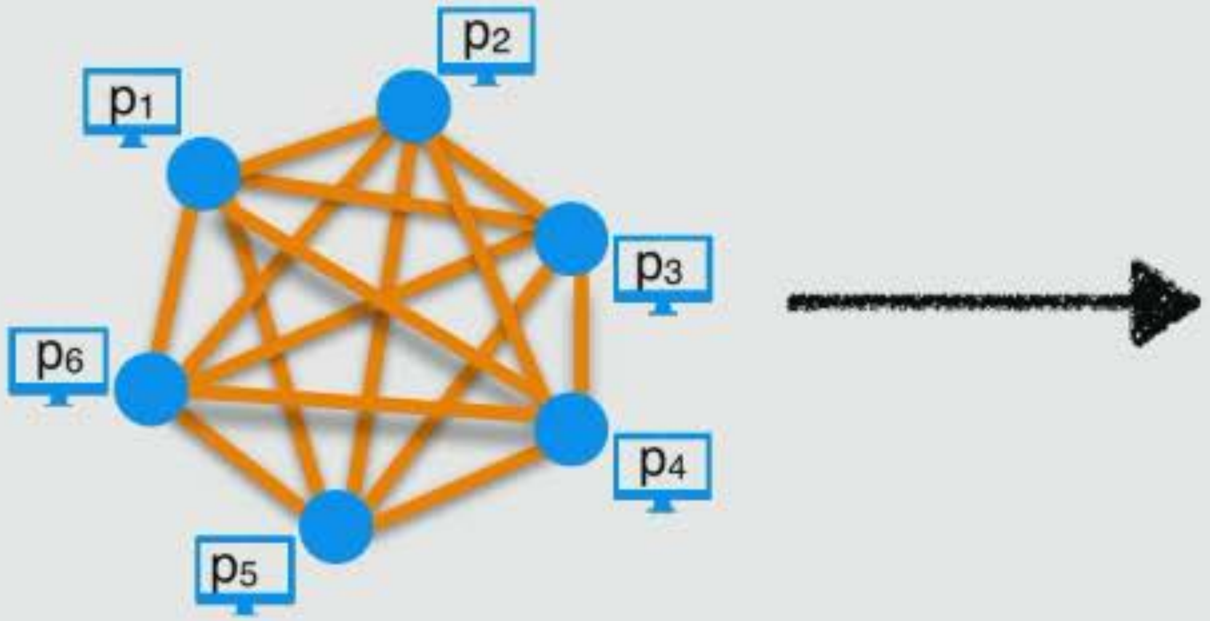
- Can choose RDMA **connections** and **permissions**
- Can give different permissions for different **memory regions**



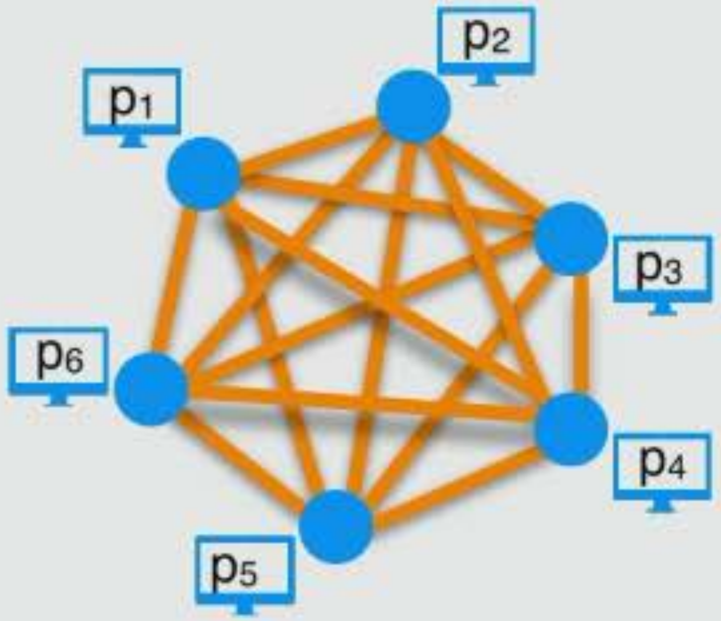
Representing an RDMA Network

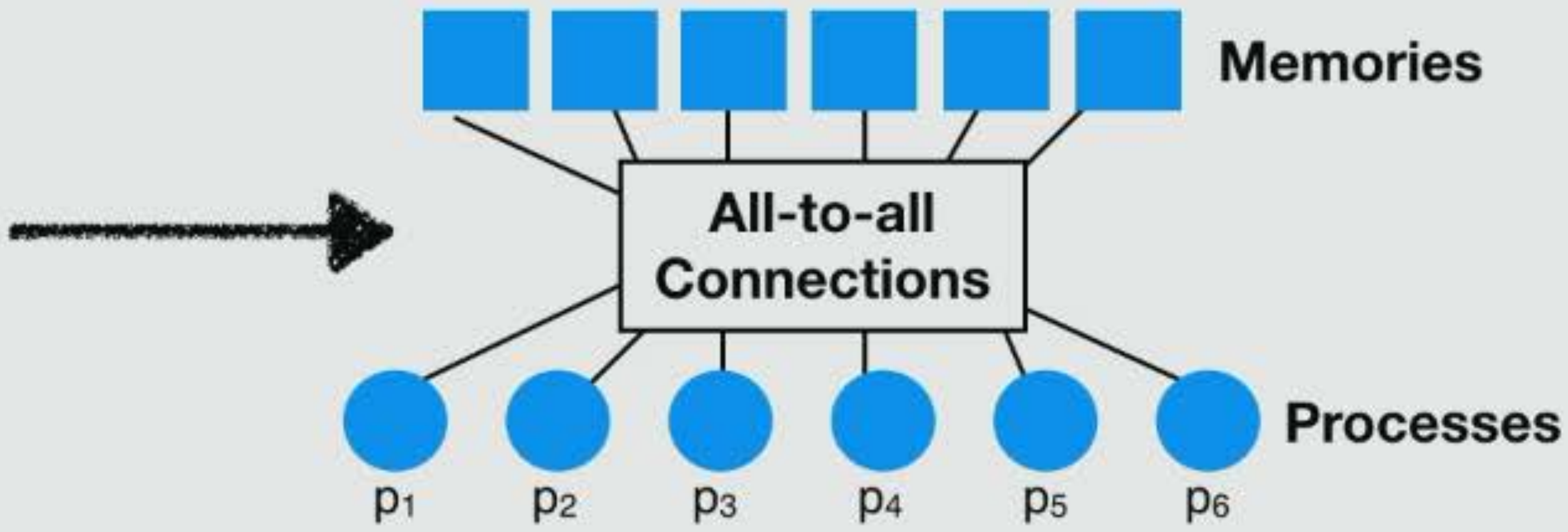


Representing an RDMA Network

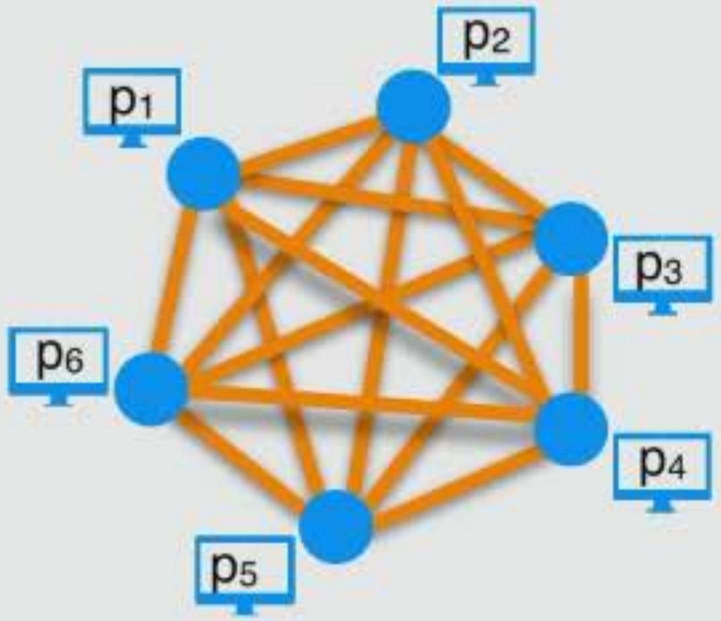


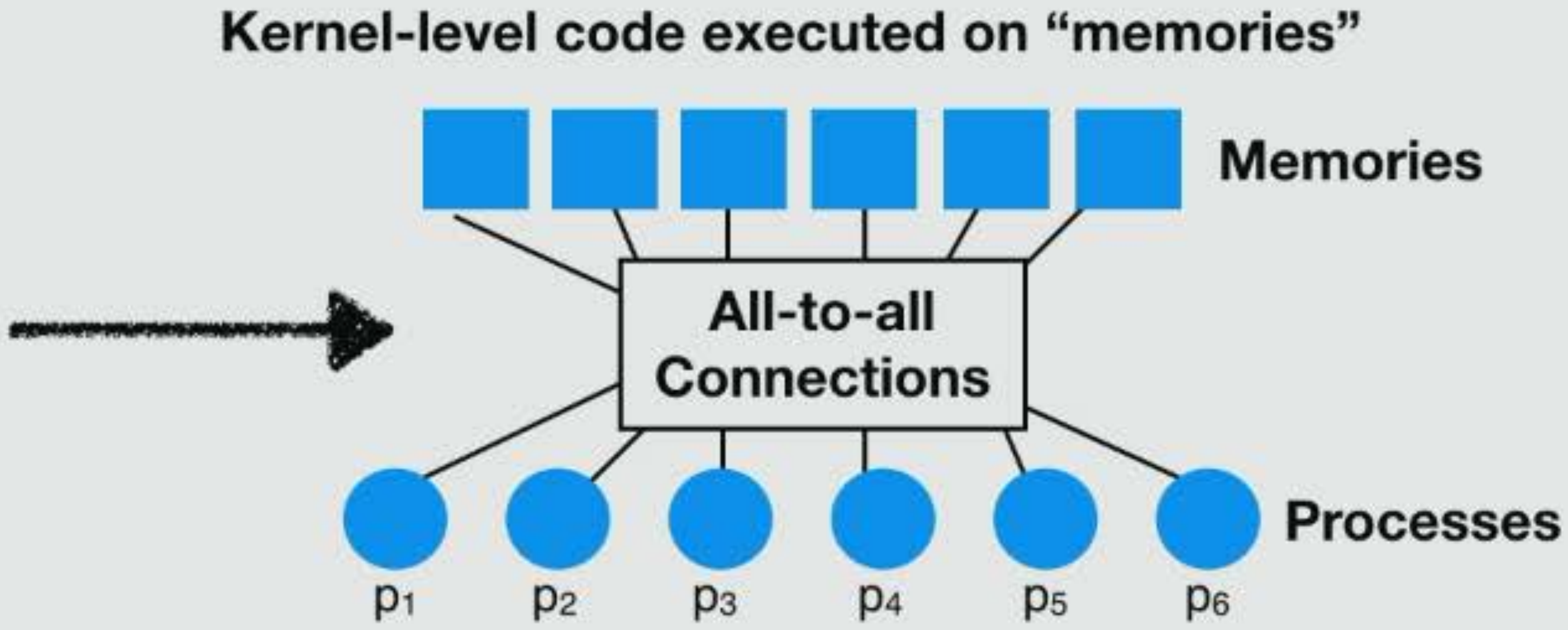
Representing an RDMA Network



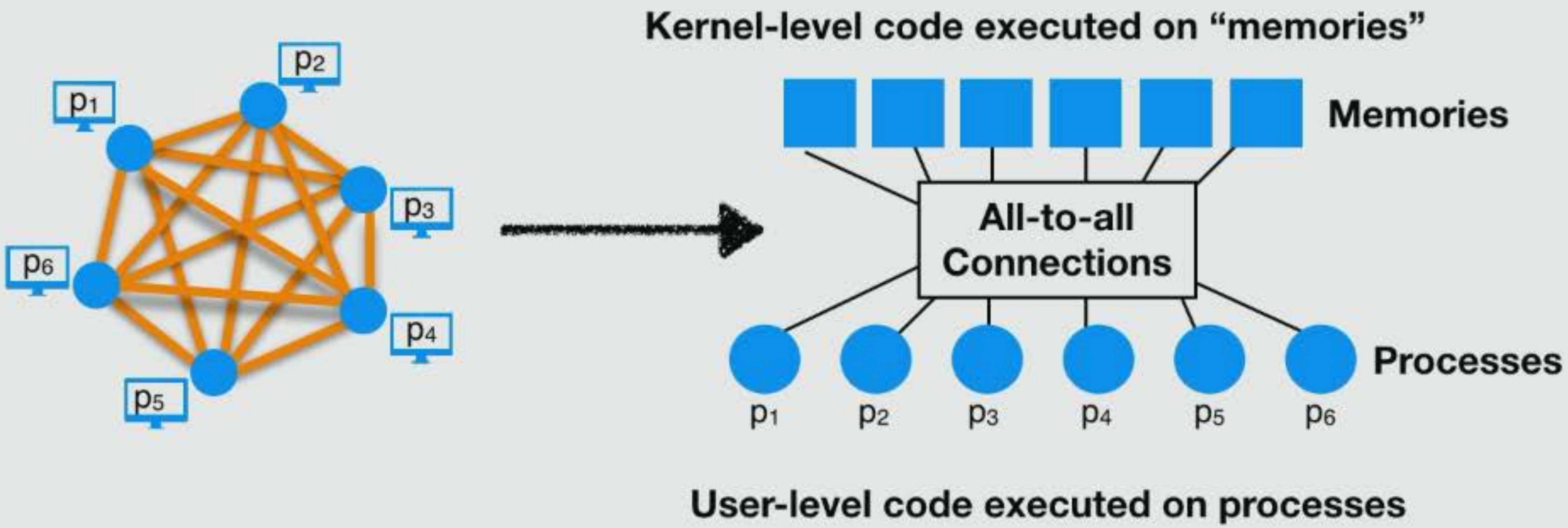


Representing an RDMA Network



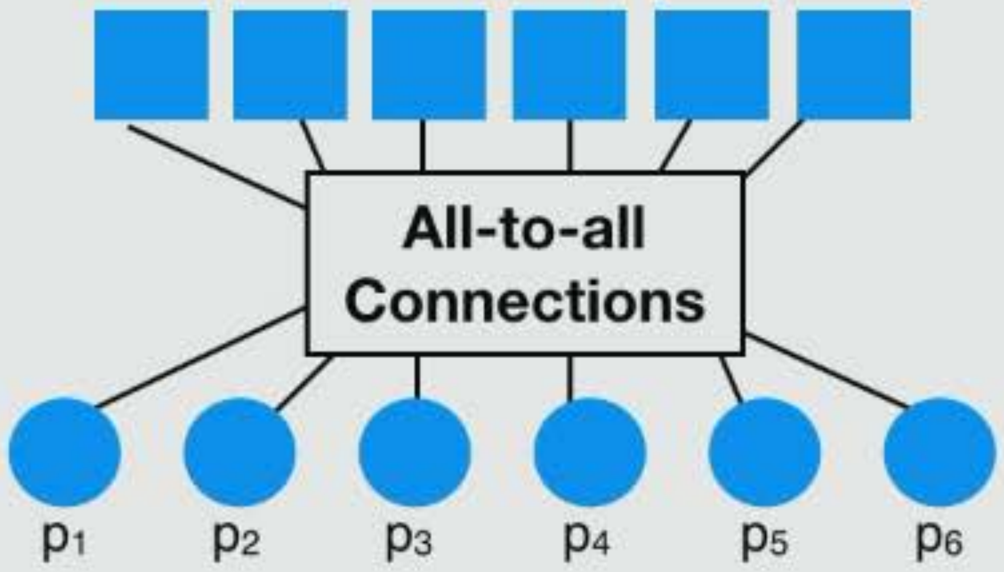


Representing an RDMA Network



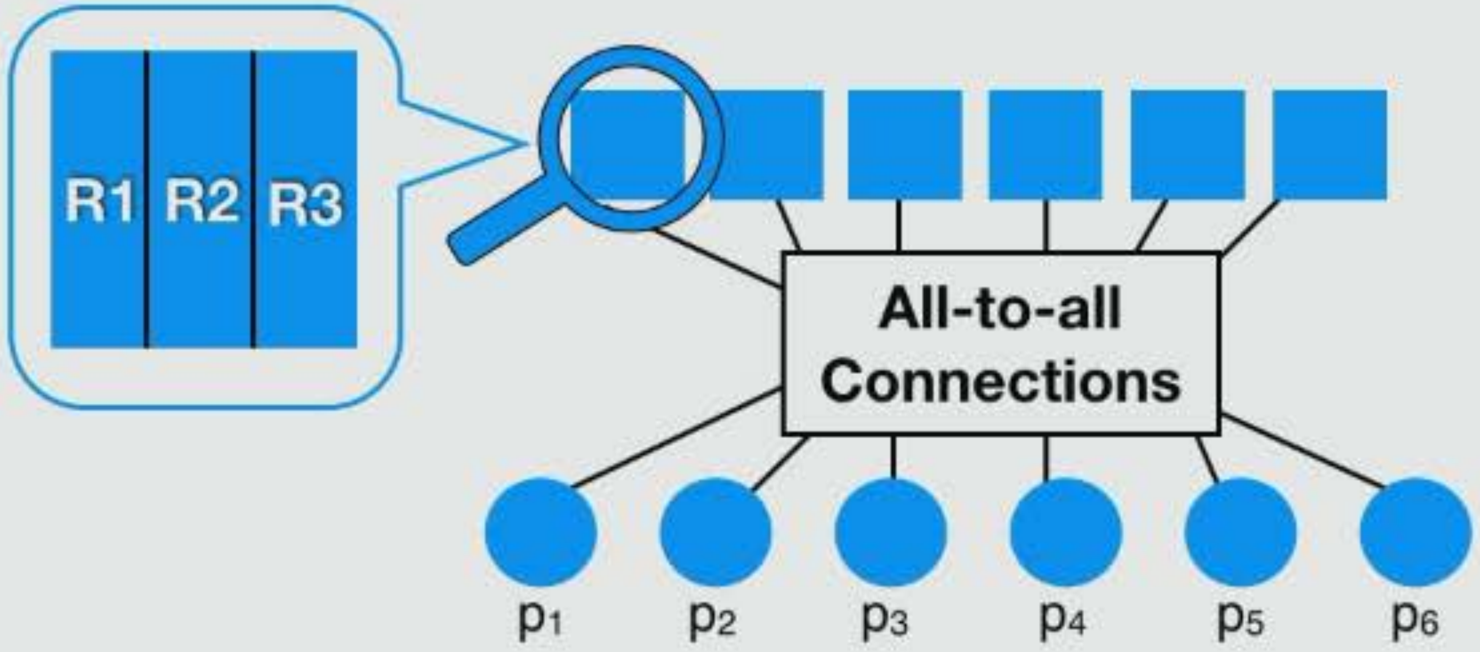
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**



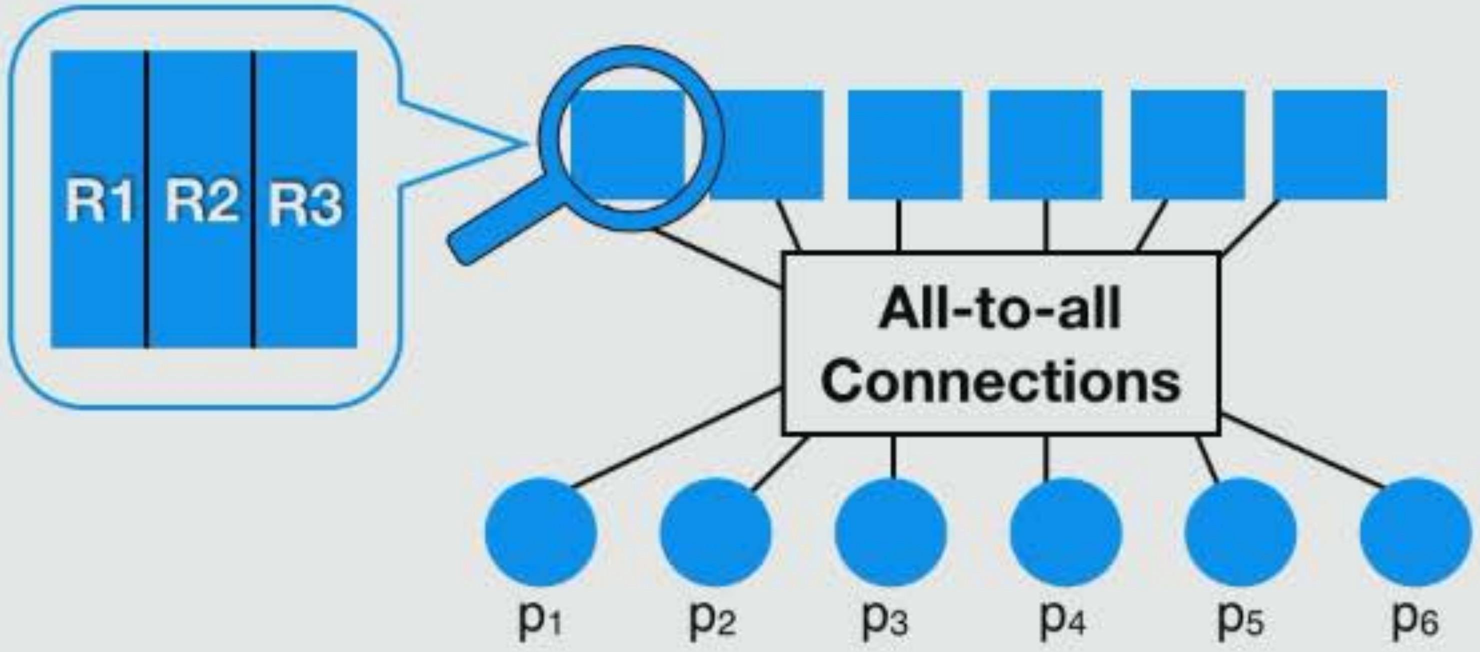
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**



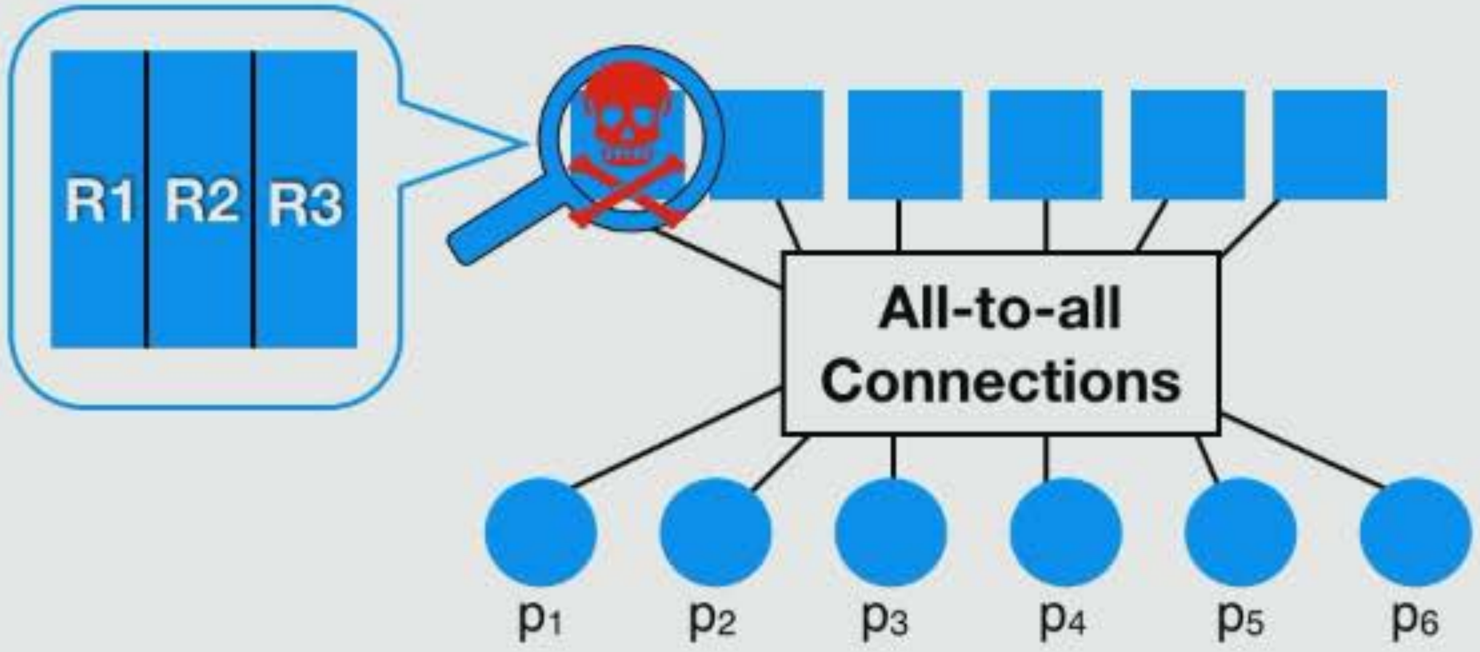
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**
- **Memories** fail by **crashing**, **processes** fail by **crashing** or being **Byzantine**



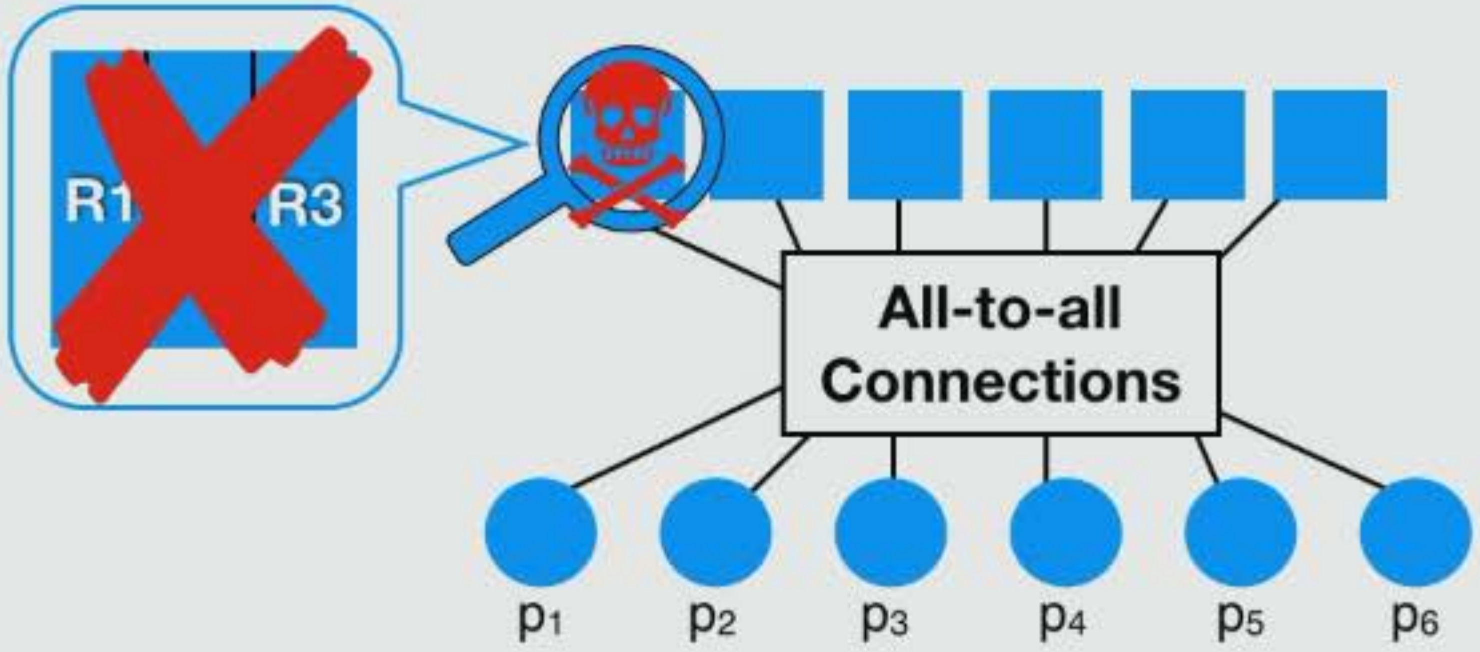
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**
- **Memories** fail by **crashing**, **processes** fail by **crashing** or being **Byzantine**



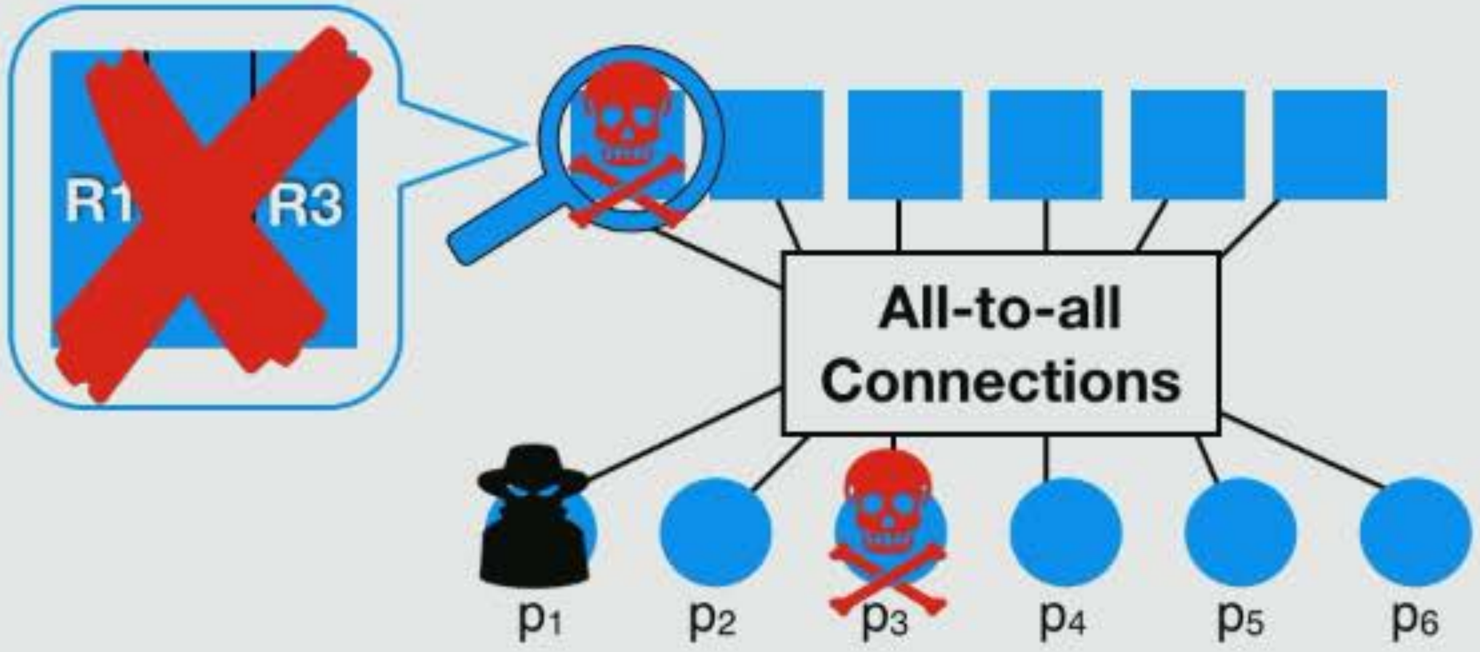
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**
- **Memories** fail by **crashing**, **processes** fail by **crashing** or being **Byzantine**



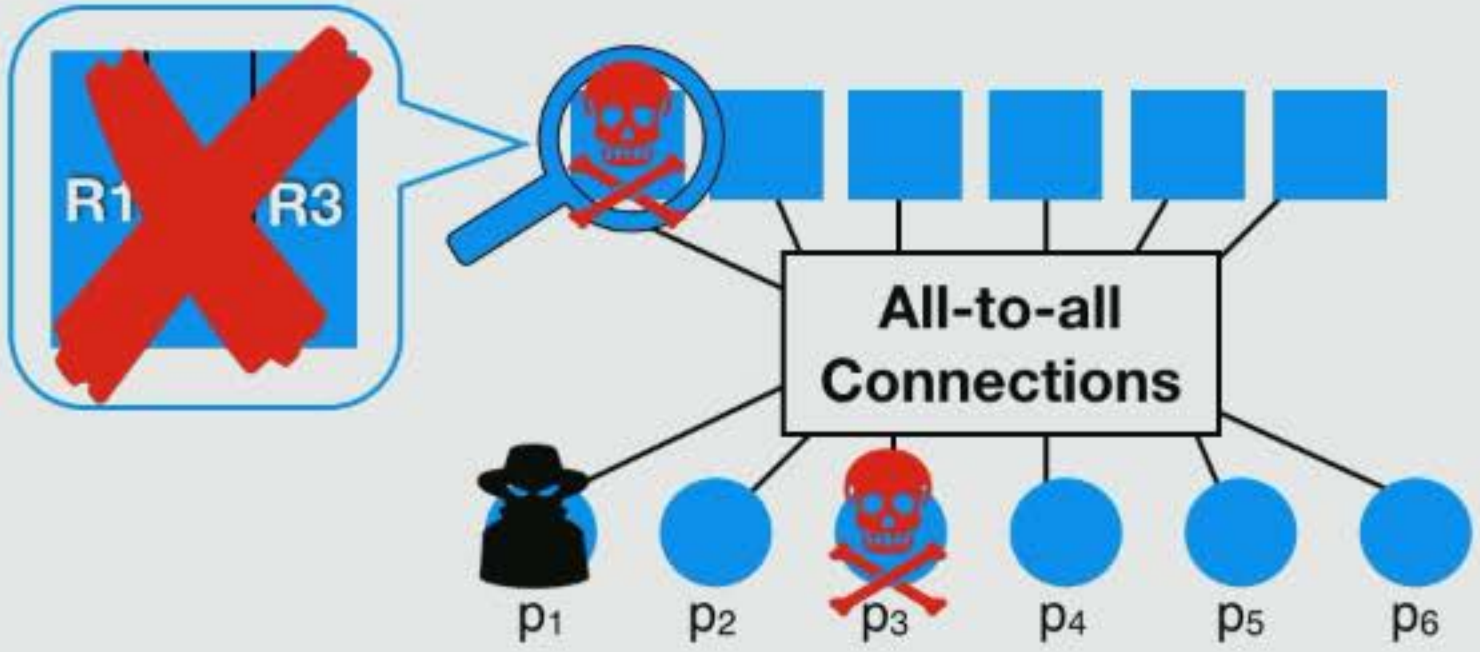
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**
- **Memories** fail by **crashing**, **processes** fail by **crashing** or being **Byzantine**



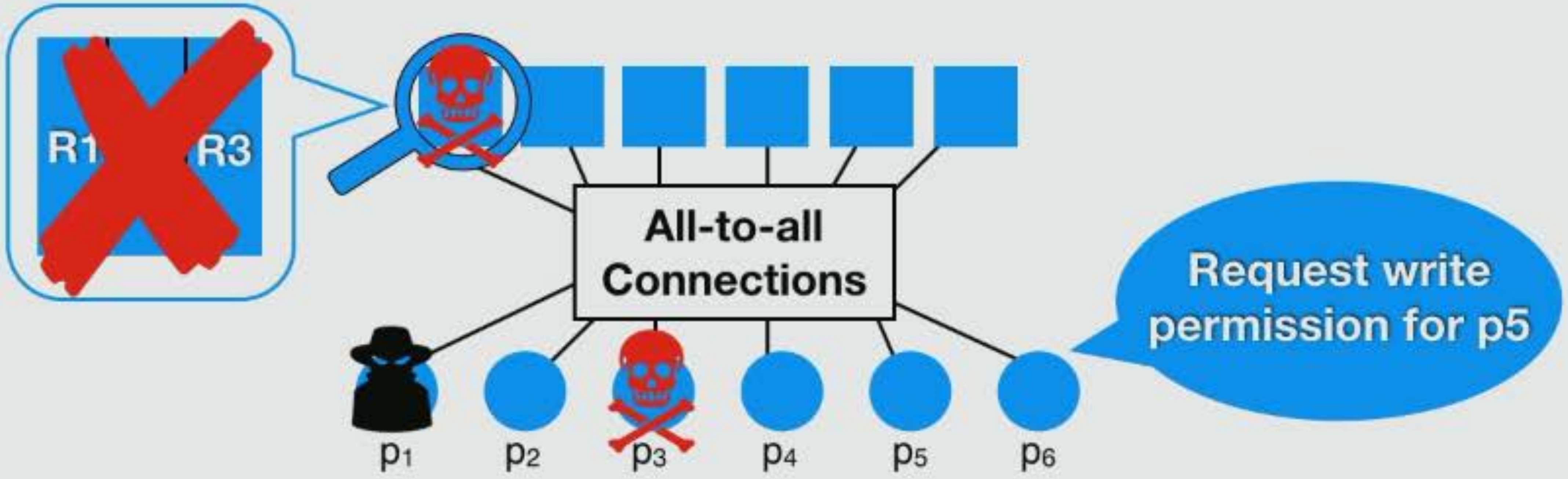
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**
- **Memories** fail by **crashing**, **processes** fail by **crashing** or being **Byzantine**
- Processes access memories through **read**, **write**, and **changePermission**



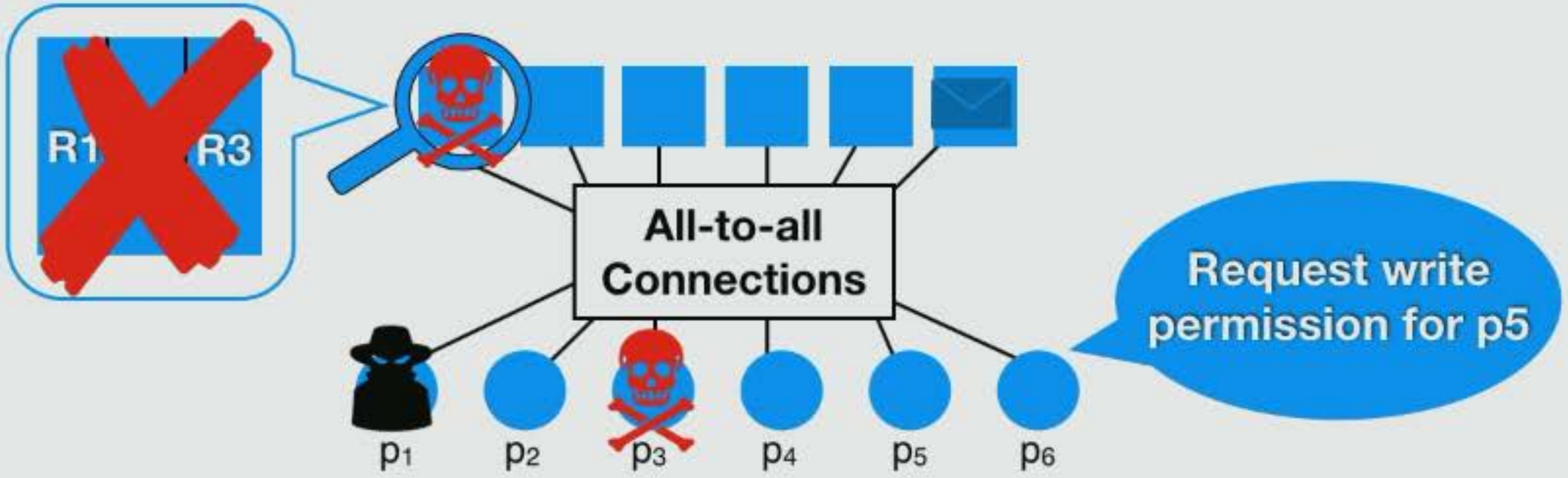
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**
- **Memories** fail by **crashing**, **processes** fail by **crashing** or being **Byzantine**
- Processes access memories through **read**, **write**, and **changePermission**



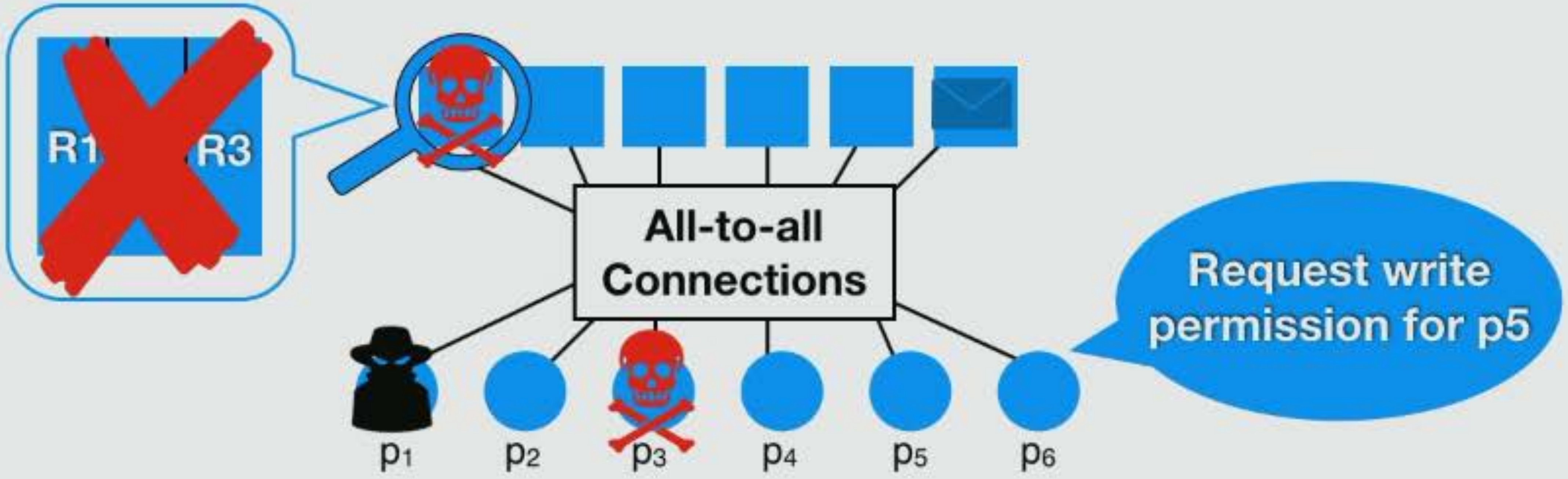
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**
- **Memories** fail by **crashing**, **processes** fail by **crashing** or being **Byzantine**
- Processes access memories through **read**, **write**, and **changePermission**



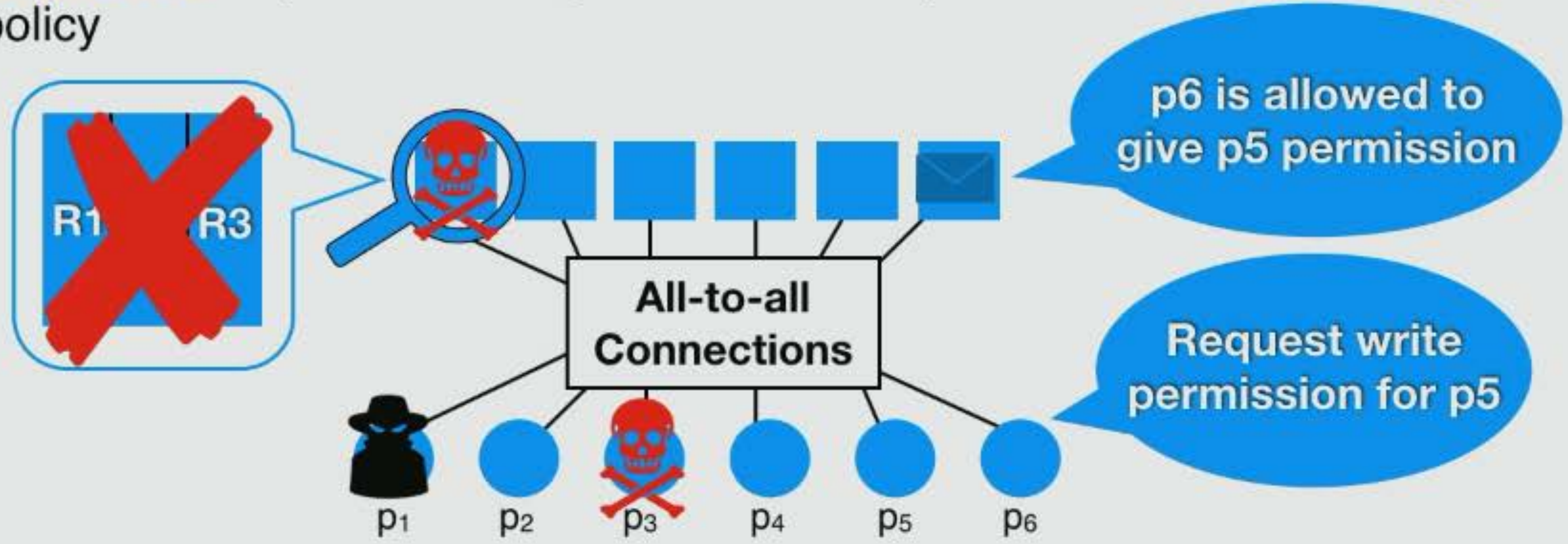
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**
- **Memories** fail by **crashing**, **processes** fail by **crashing** or being **Byzantine**
- Processes access memories through **read**, **write**, and **changePermission**
- **Memories** respond to changePermission requests with **acceptChange** policy



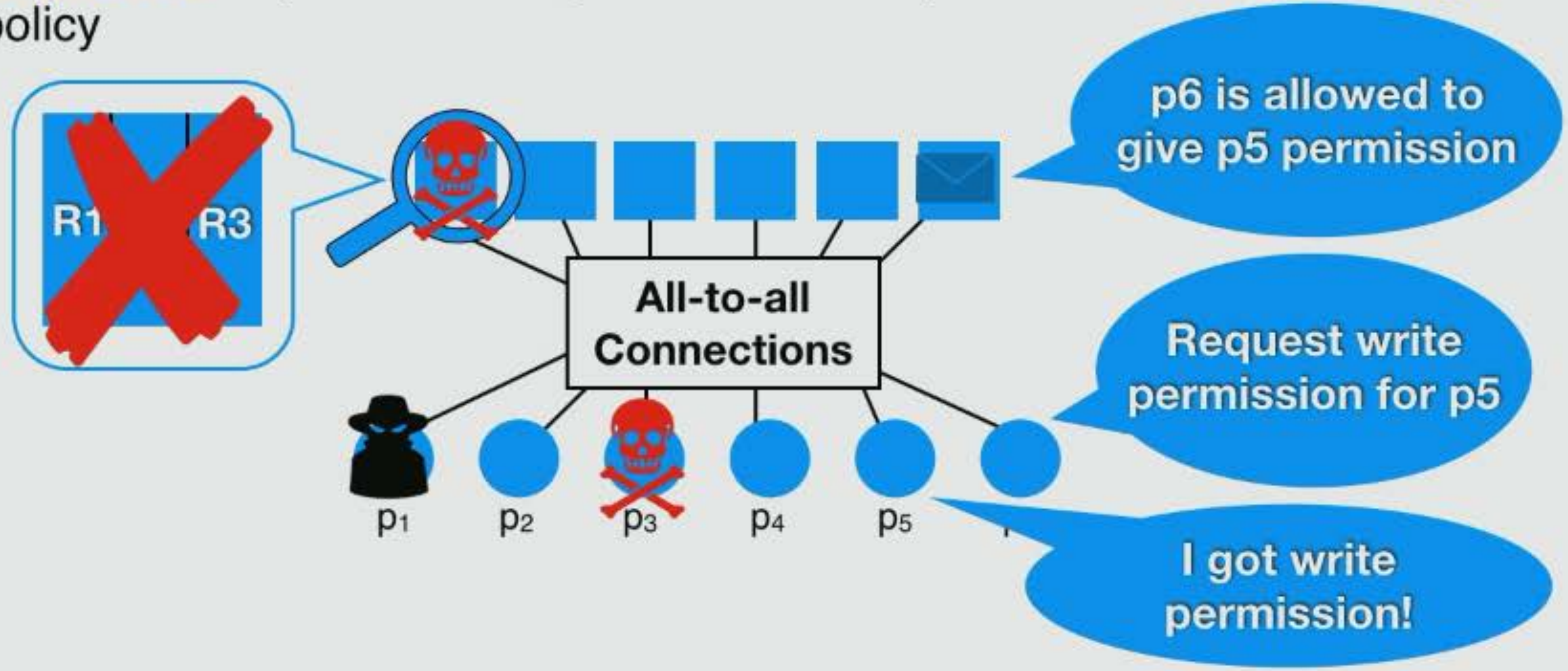
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**
- **Memories** fail by **crashing**, **processes** fail by **crashing** or being **Byzantine**
- Processes access memories through **read**, **write**, and **changePermission**
- **Memories** respond to changePermission requests with **acceptChange** policy



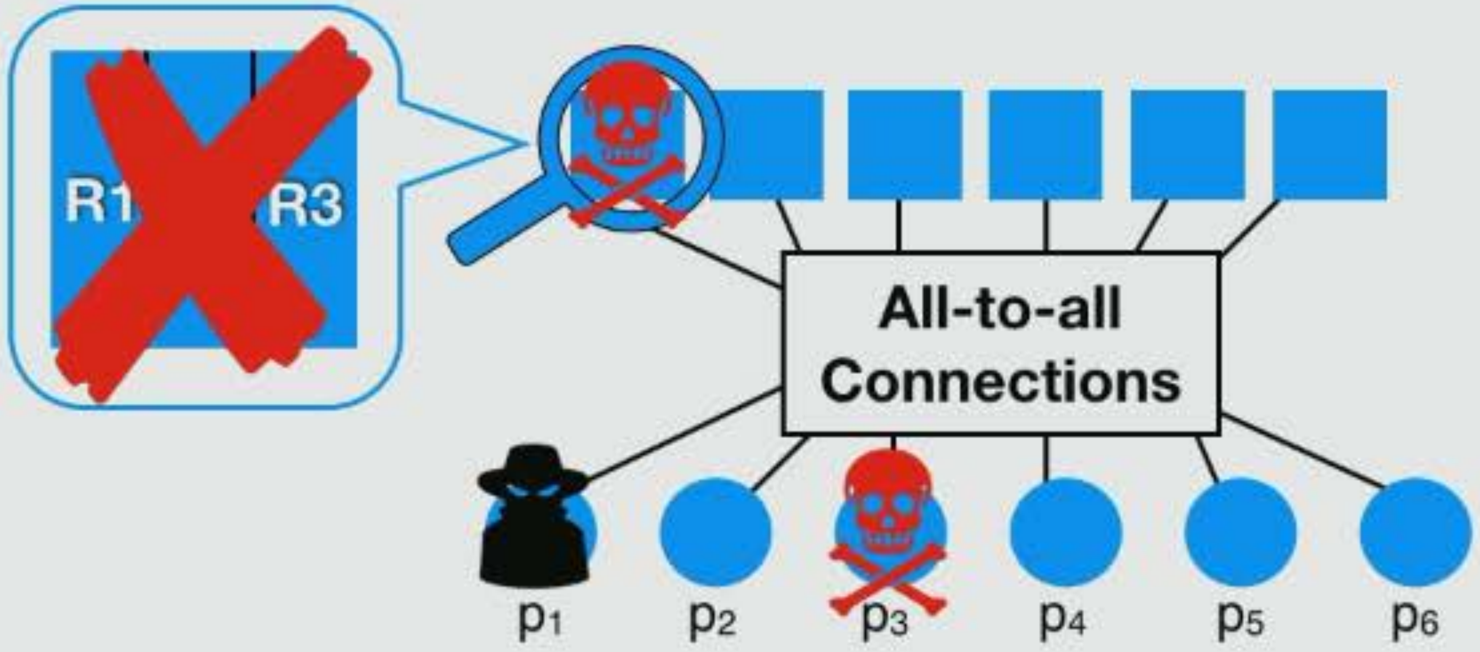
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**
- **Memories** fail by **crashing**, **processes** fail by **crashing** or being **Byzantine**
- Processes access memories through **read**, **write**, and **changePermission**
- **Memories** respond to changePermission requests with **acceptChange** policy



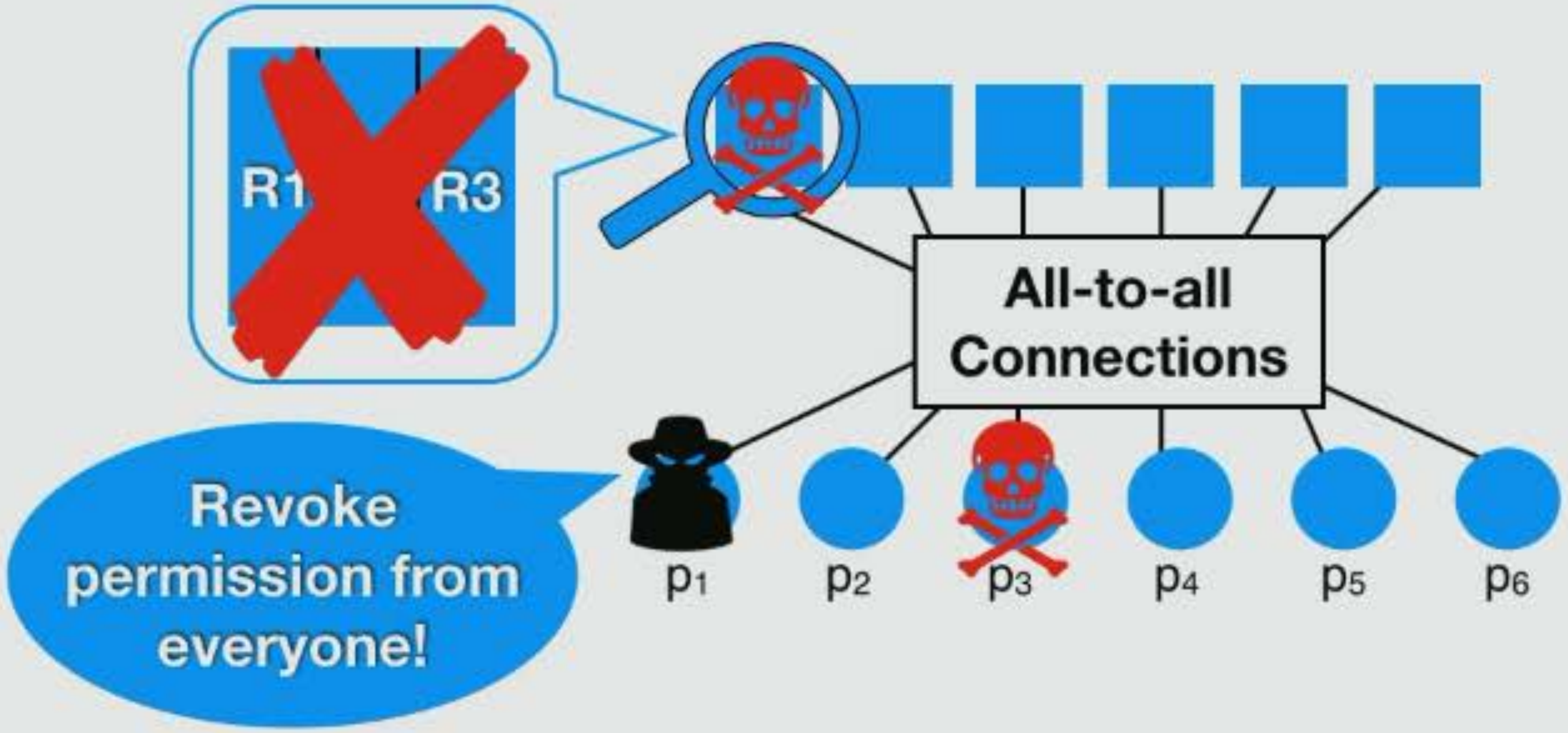
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**
- **Memories** fail by **crashing**, **processes** fail by **crashing** or being **Byzantine**
- Processes access memories through **read**, **write**, and **changePermission**
- **Memories** respond to changePermission requests with **acceptChange** policy



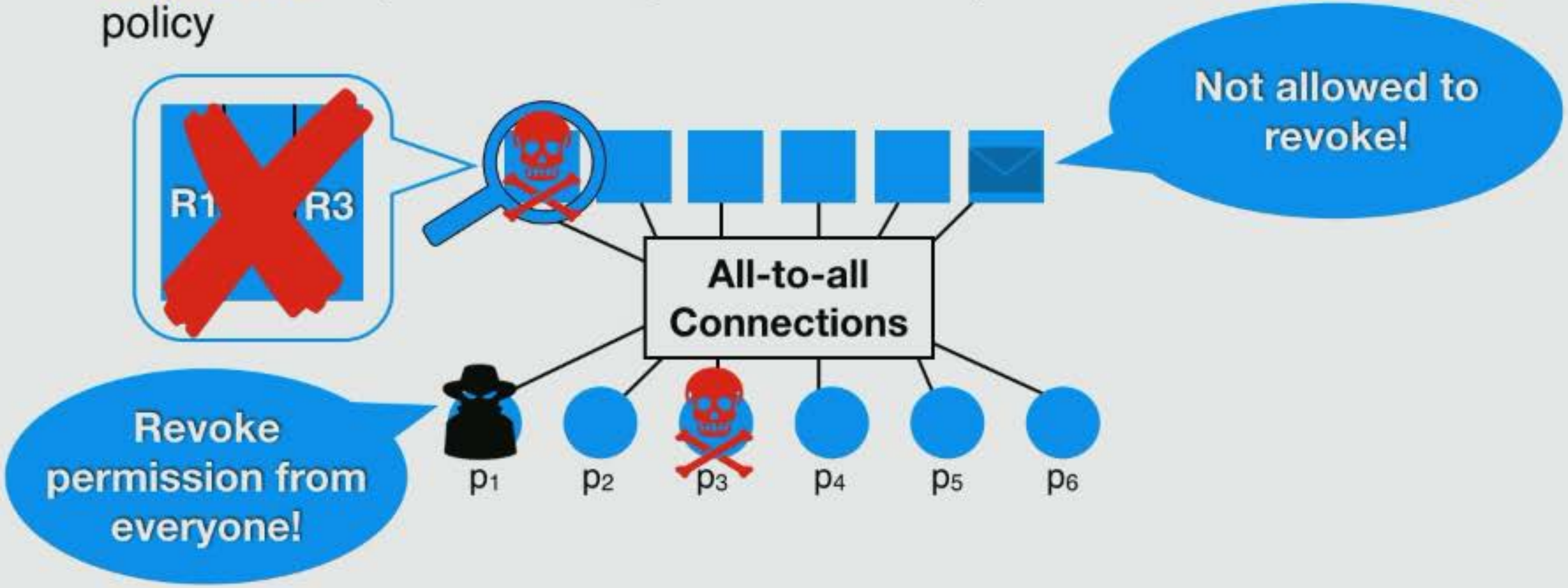
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**
- **Memories** fail by **crashing**, **processes** fail by **crashing** or being **Byzantine**
- Processes access memories through **read**, **write**, and **changePermission**
- **Memories** respond to changePermission requests with **acceptChange** policy



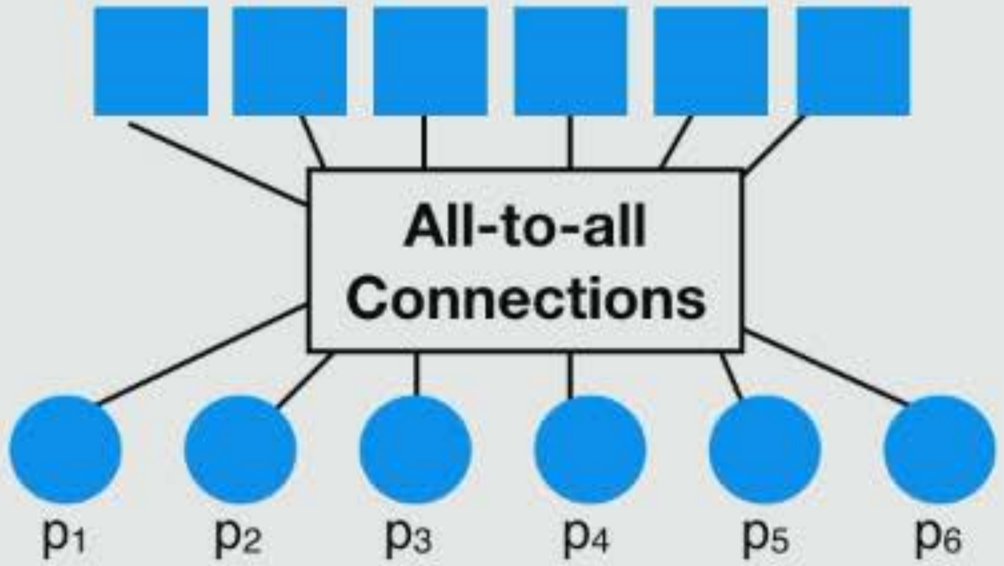
RDMA Model

- **Asynchronous** network of ***n* processes** and ***m* memories**
- **Memories** fail by **crashing**, **processes** fail by **crashing** or being **Byzantine**
- Processes access memories through **read**, **write**, and **changePermission**
- **Memories** respond to changePermission requests with **acceptChange** policy



Handling Memory Failures

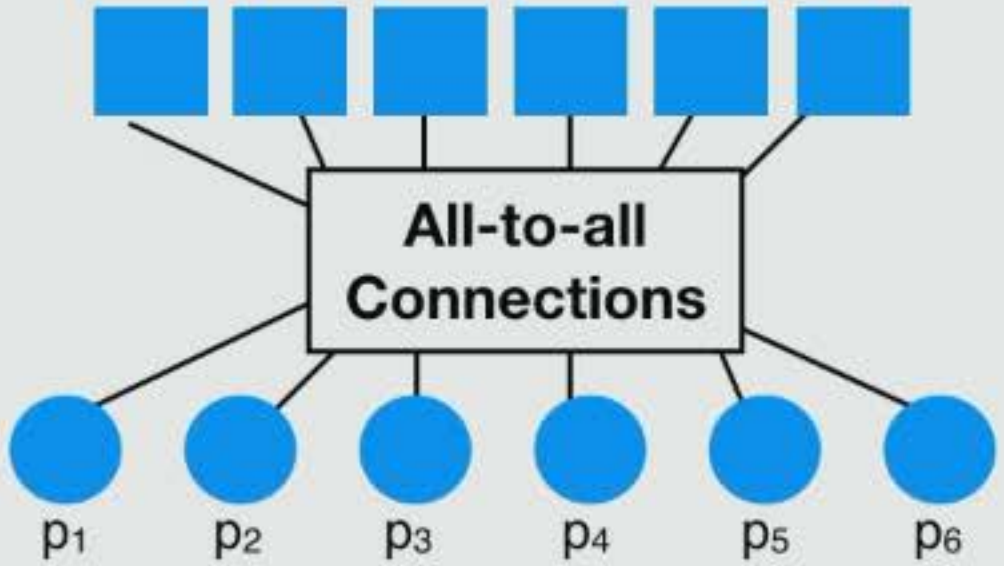
Replication: Treat all memories the same



Handling Memory Failures

Replication: Treat all memories the same

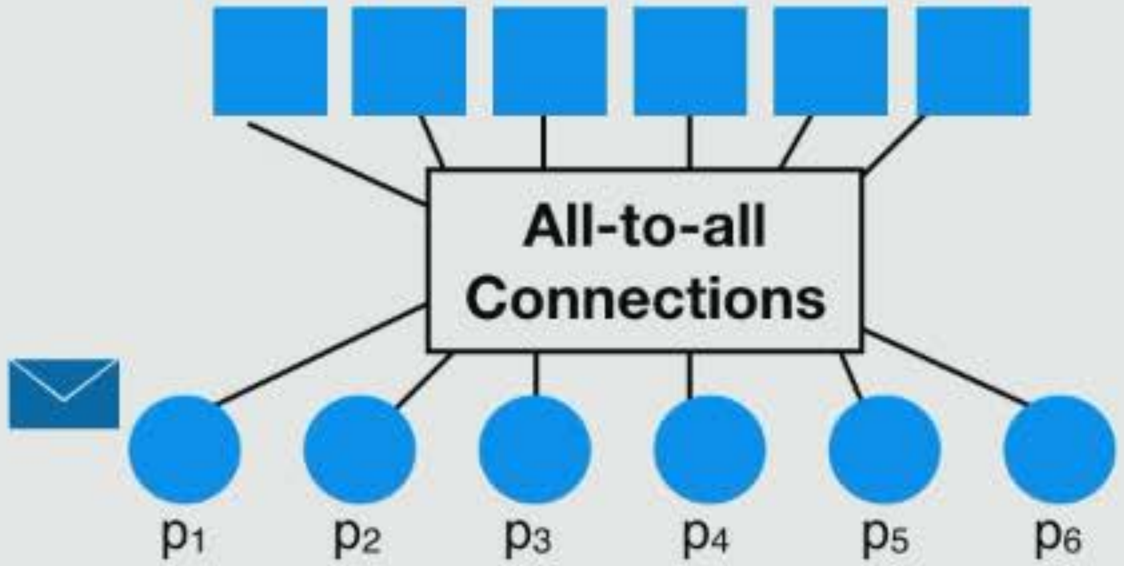
Send all write/read requests to all memories, wait to hear acknowledgement from majority



Handling Memory Failures

Replication: Treat all memories the same

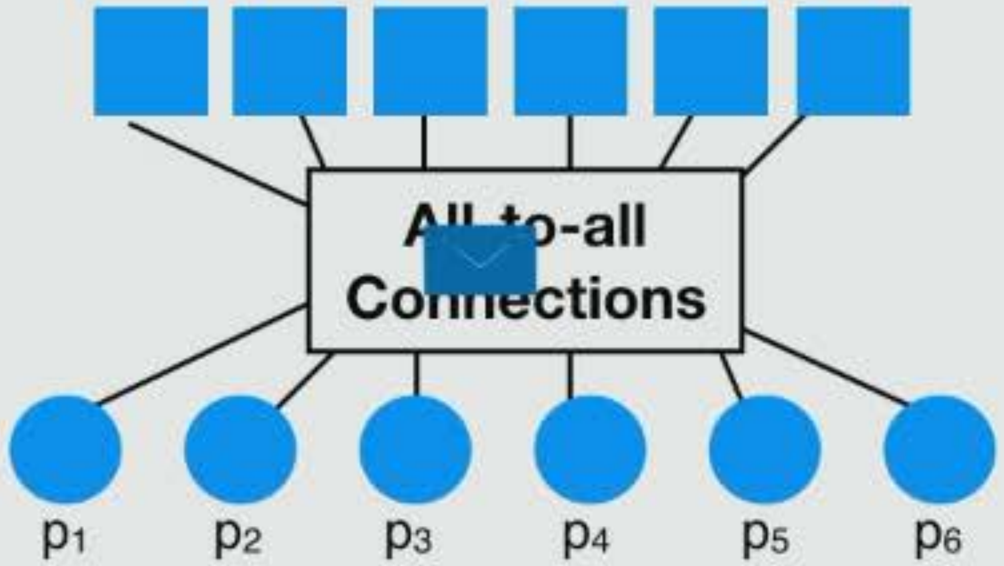
Send all write/read requests to all memories, wait to hear acknowledgement from majority



Handling Memory Failures

Replication: Treat all memories the same

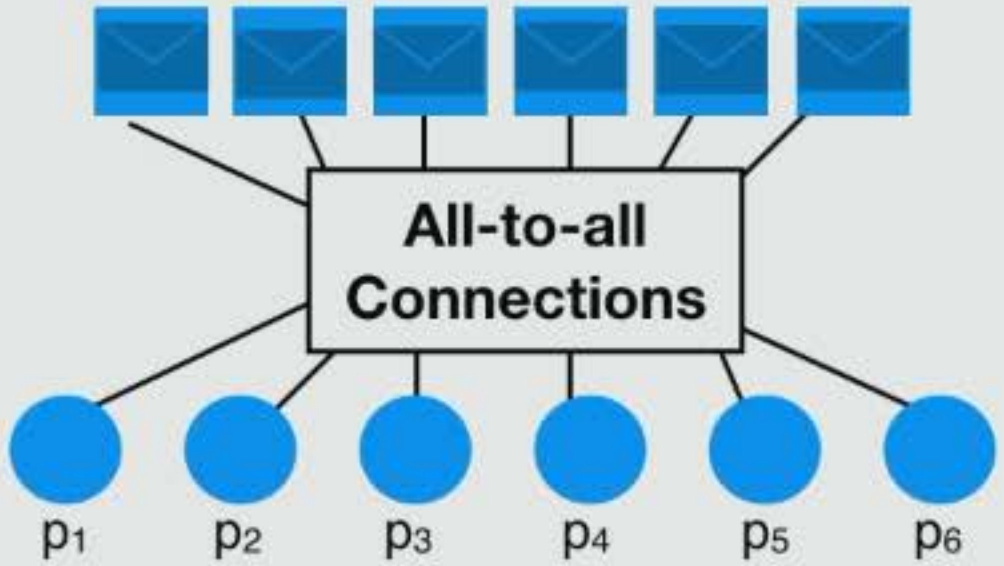
Send all write/read requests to all memories, wait to hear acknowledgement from majority



Handling Memory Failures

Replication: Treat all memories the same

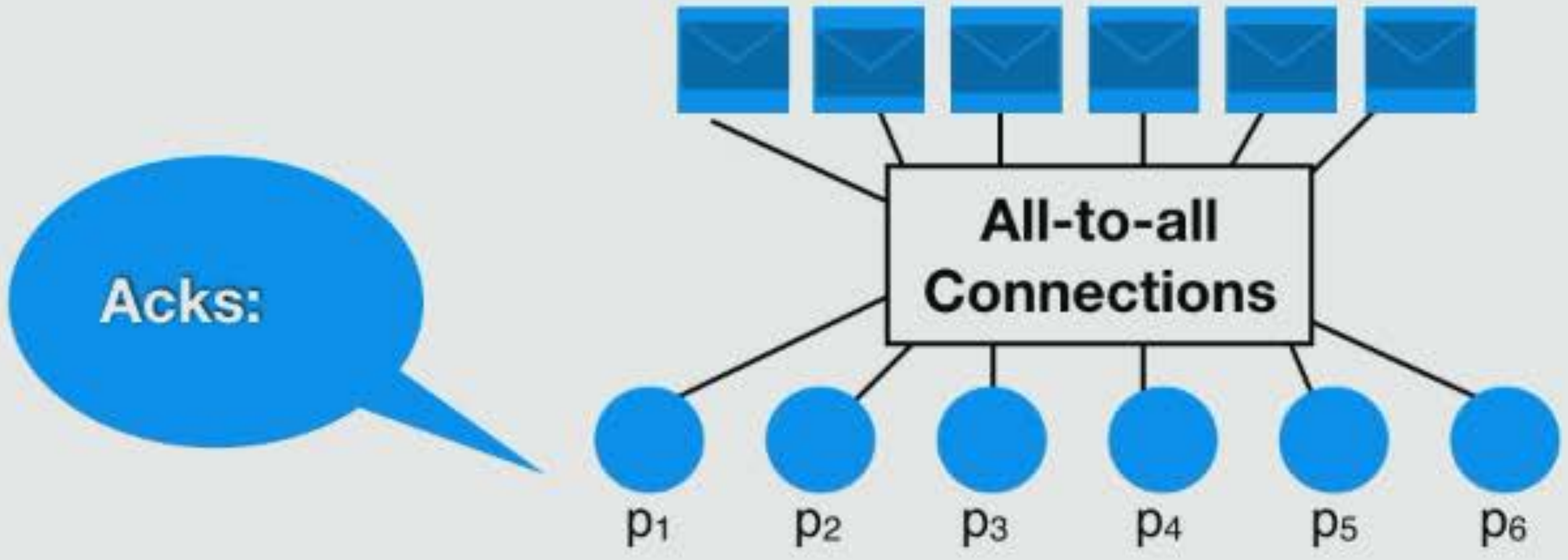
Send all write/read requests to all memories, wait to hear acknowledgement from majority



Handling Memory Failures

Replication: Treat all memories the same

Send all write/read requests to all memories, wait to hear acknowledgement from majority

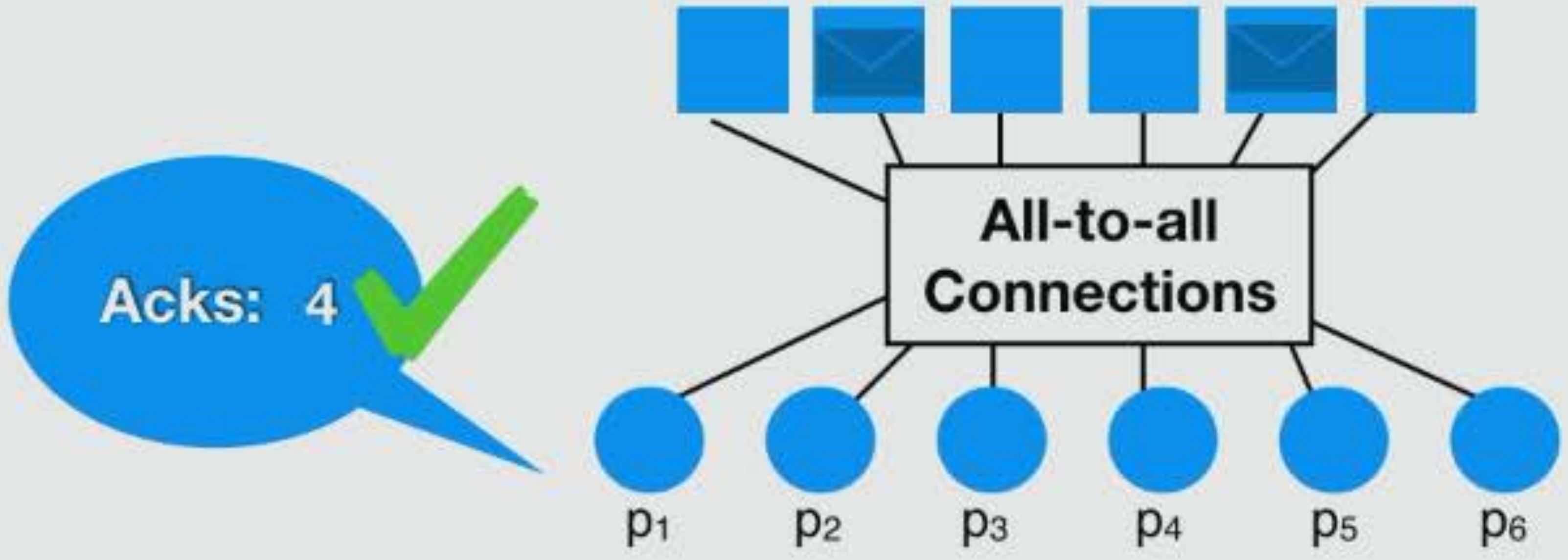


Handling Memory Failures

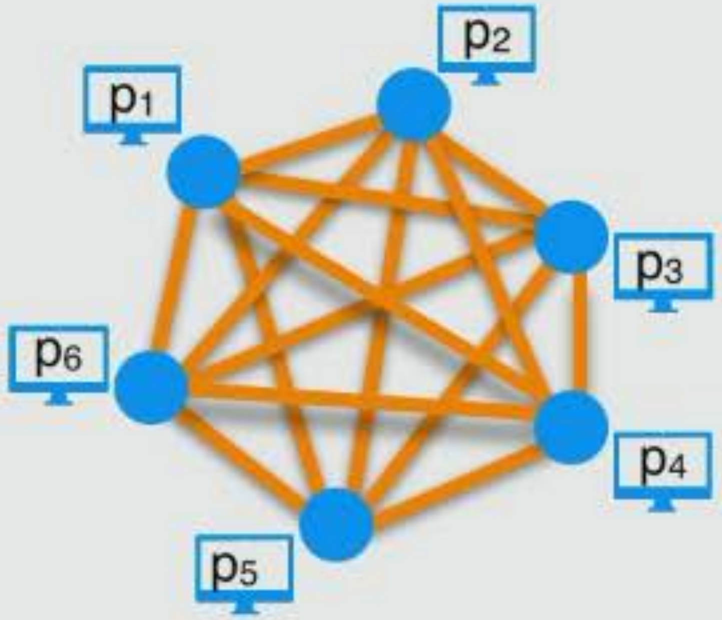
Replication: Treat all memories the same

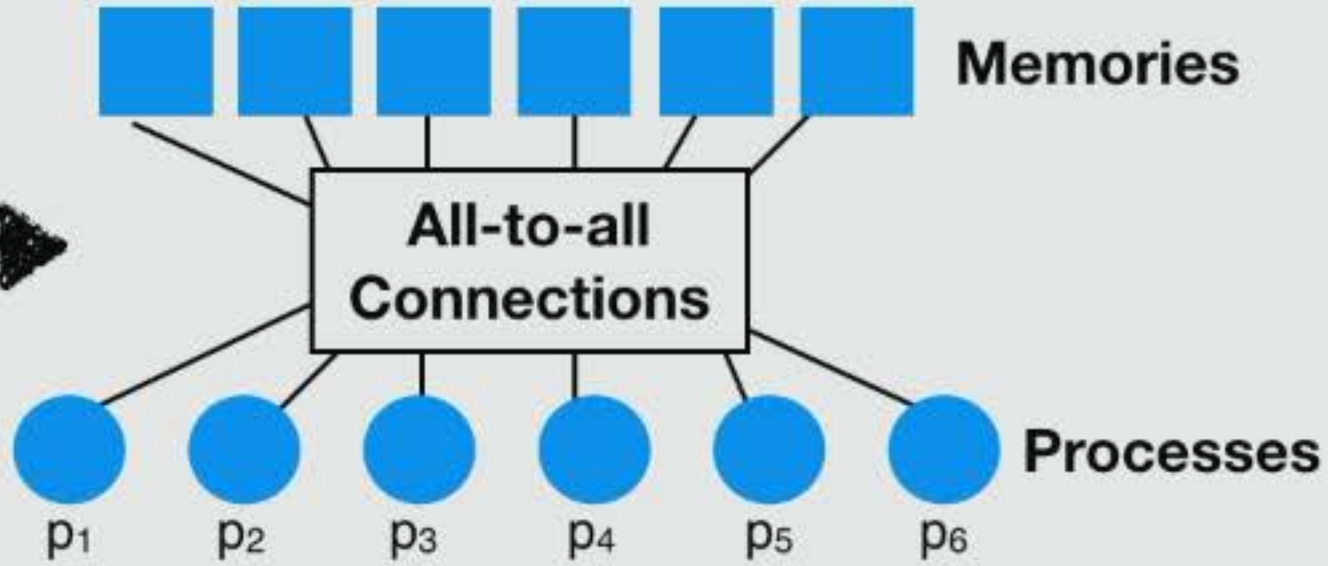
Send all write/read requests to all memories, wait to hear acknowledgement from majority

Instead of many faulty memories, we can now think of *one non-faulty memory!*

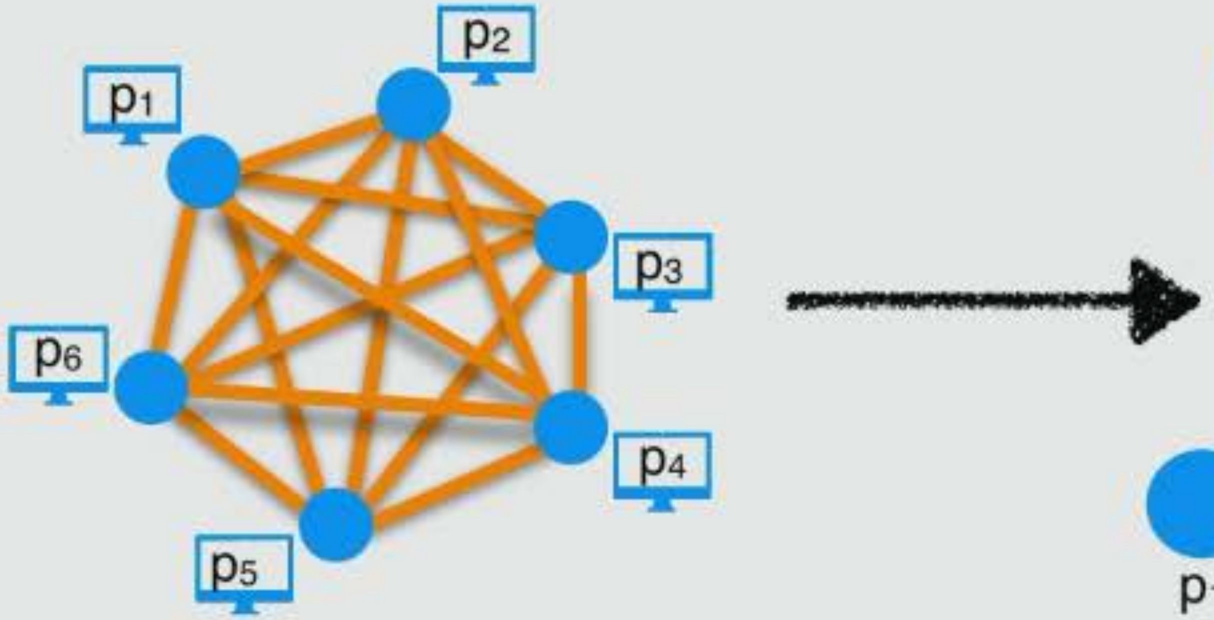


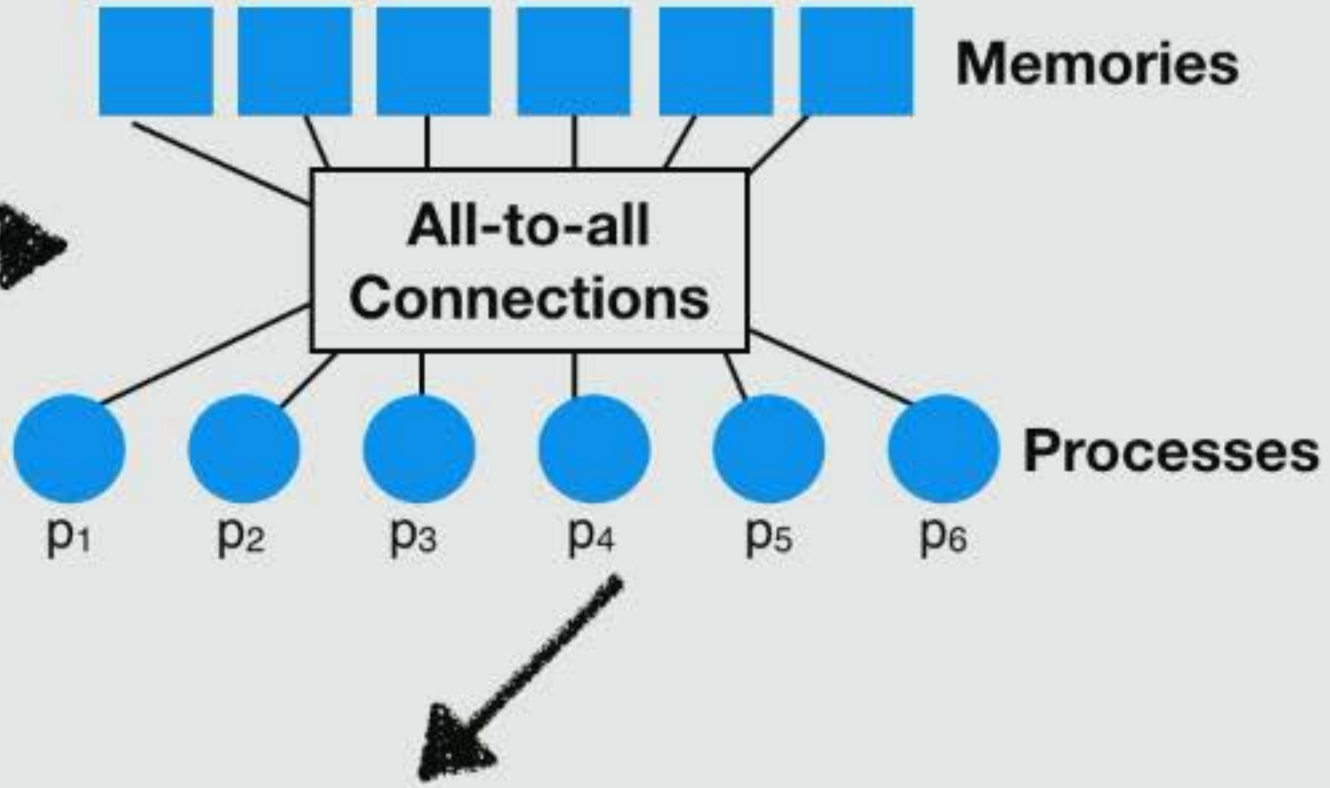
Representing an RDMA Network



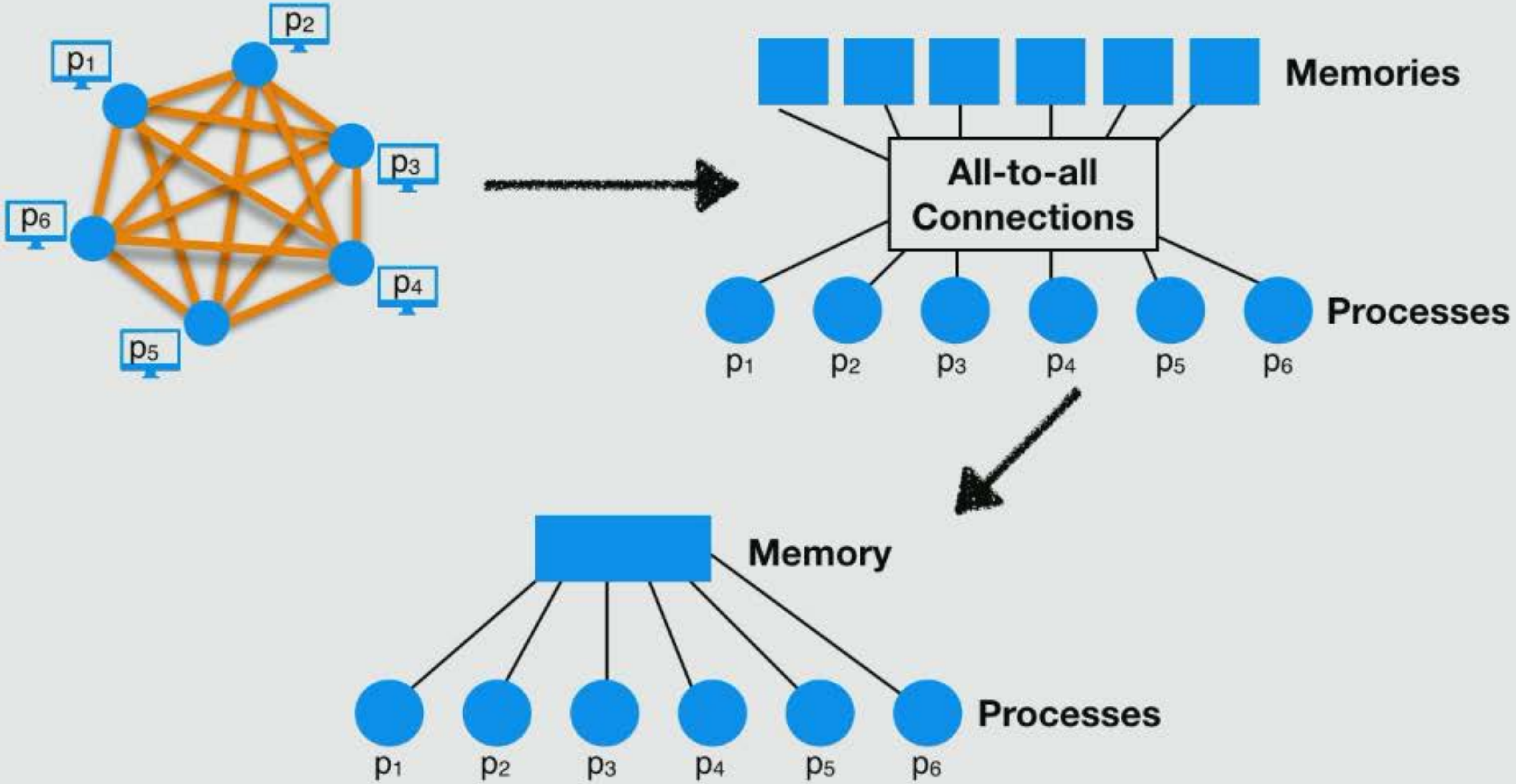


Representing an RDMA Network

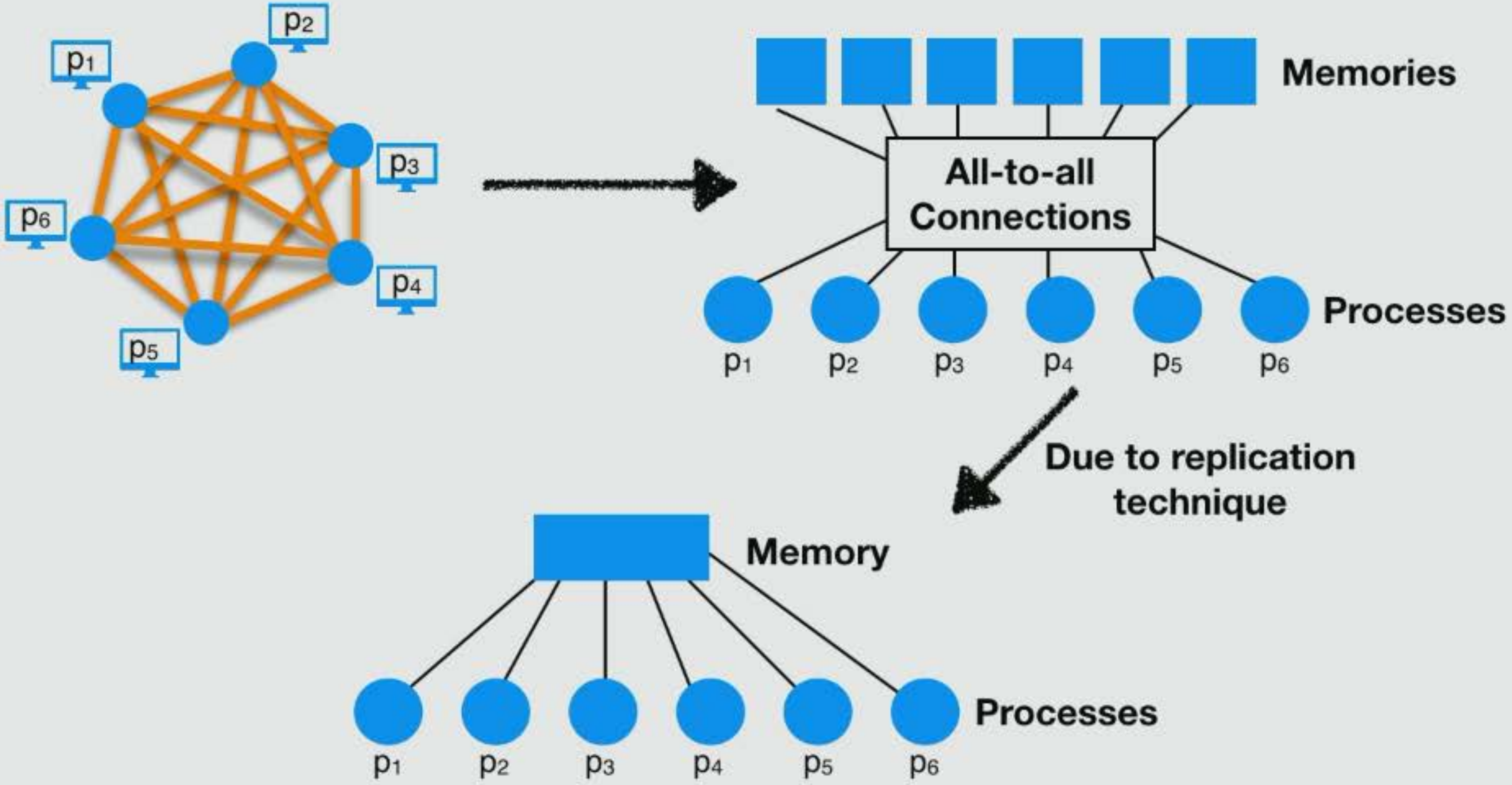




Representing an RDMA Network



Representing an RDMA Network



Outline

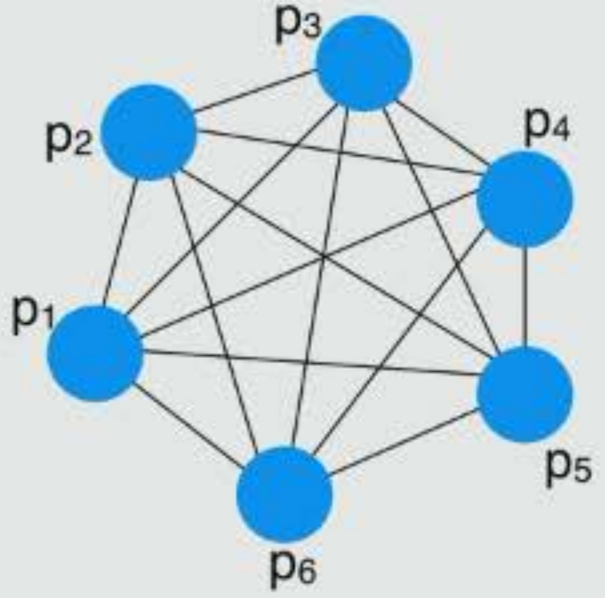
- **RDMA details**
- Setting 1: **RDMA's full power** (complete graph)
  - **Crash-only** algorithm: ***n>f*** tolerant, 1 round-trip
  - **Byzantine** algorithm: ***n>2f*** tolerant, 1 round-trip

- Setting 2: **Scalability: Using RDMA sparingly** (incomplete graph)
  - Crash-only Algorithm: tolerance vs topology

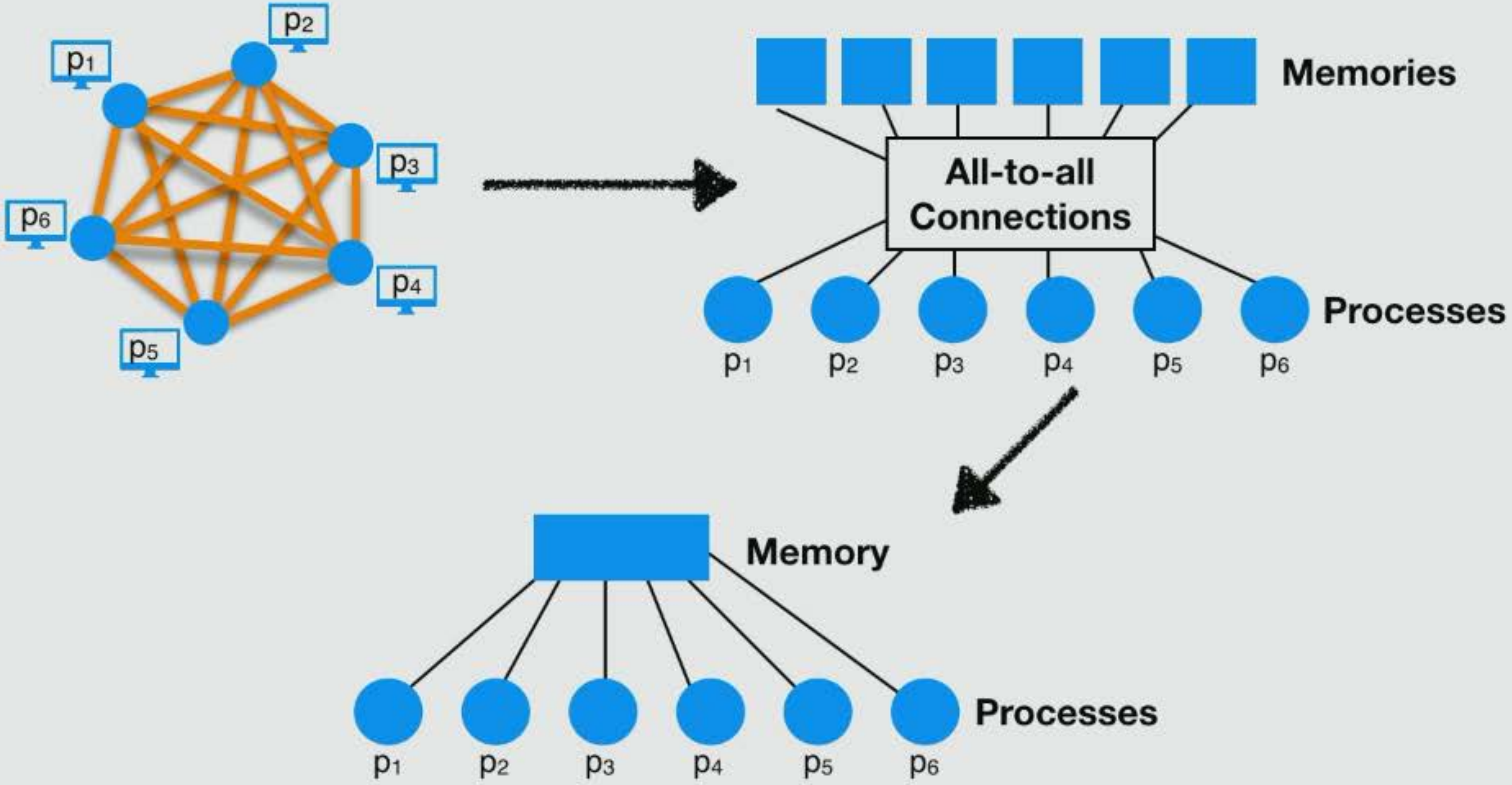
[Lamport'98]

Paxos

Message passing consensus



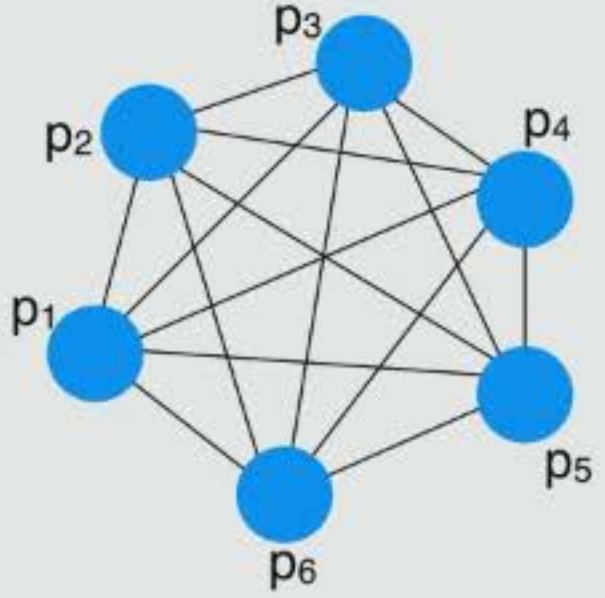
Representing an RDMA Network



[Lamport'98]

Paxos

Message passing consensus

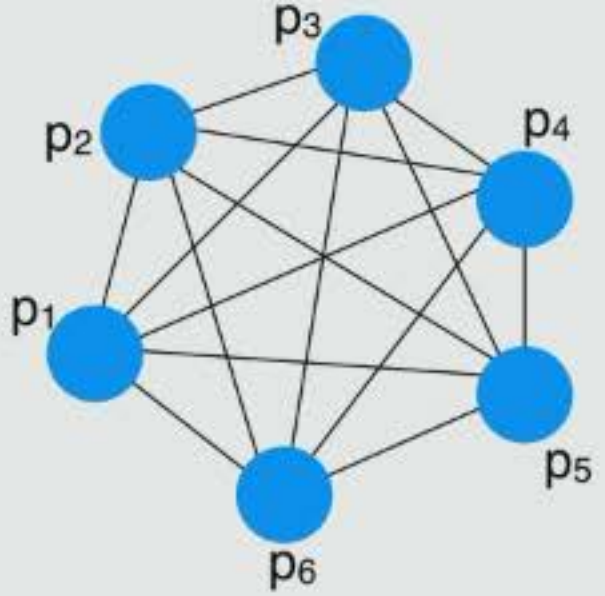


[Lamport'98]

Paxos

Message passing consensus

- Two rounds:

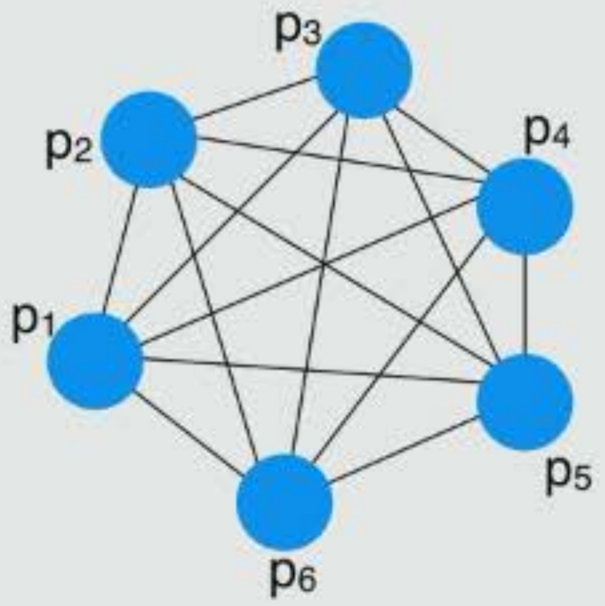


[Lamport'98]

Paxos

Message passing consensus

- Two rounds:
    1. **prepare:** I want to propose a value!

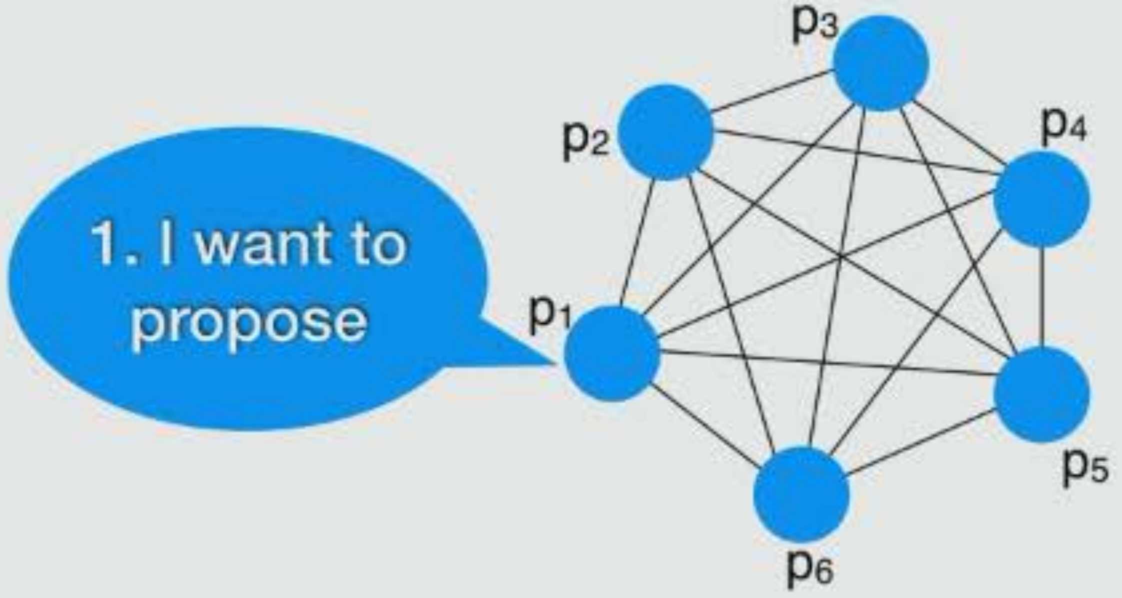


[Lamport'98]

Paxos

Message passing consensus

- Two rounds:
    1. **prepare:** I want to propose a value!

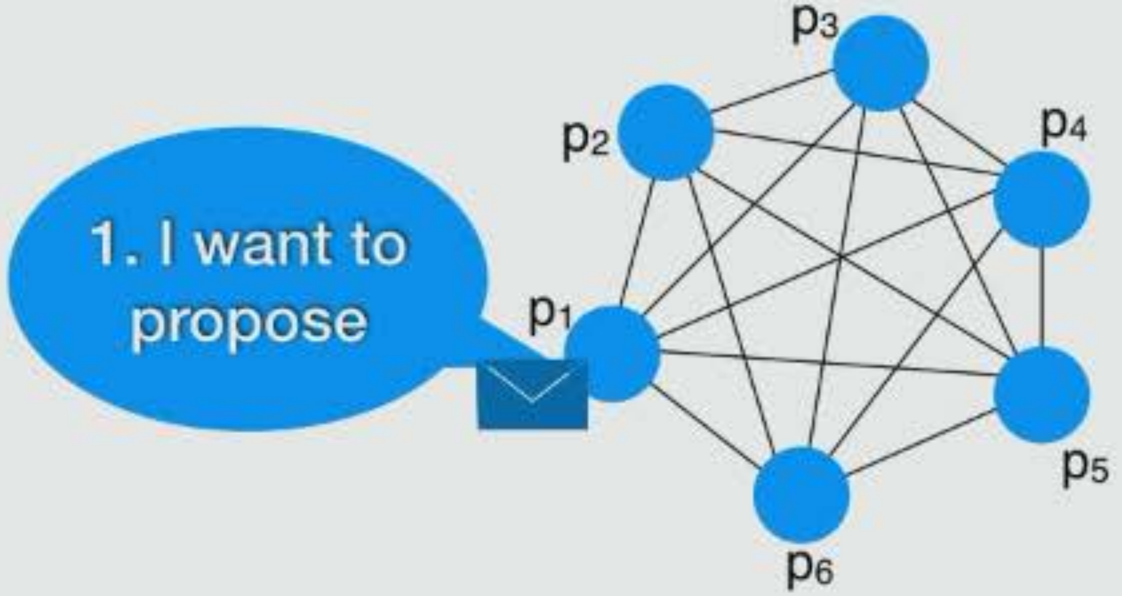


[Lamport'98]

Paxos

Message passing consensus

- Two rounds:
    1. **prepare:** I want to propose a value!

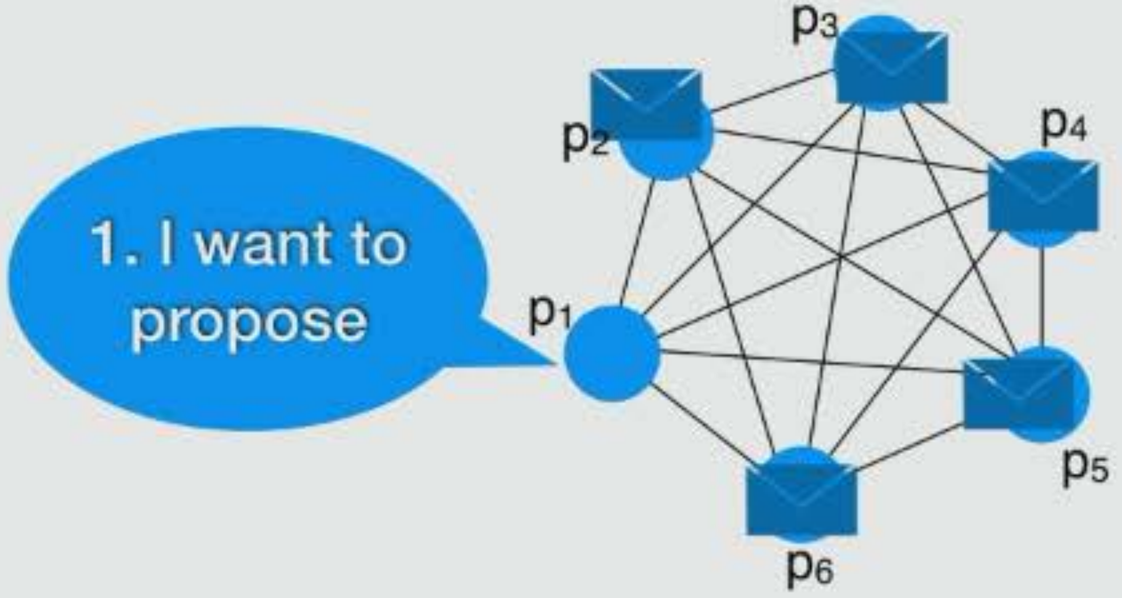


[Lamport'98]

Paxos

Message passing consensus

- Two rounds:
    1. **prepare:** I want to propose a value!

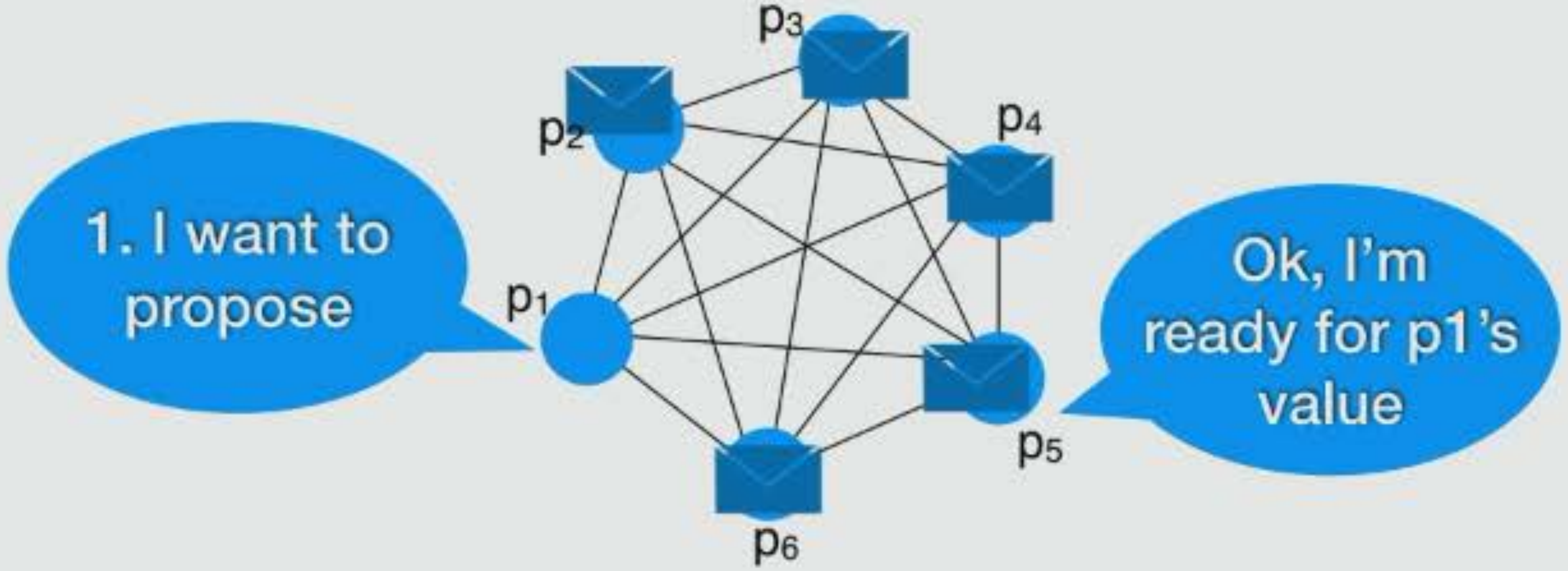


[Lamport'98]

Paxos

Message passing consensus

- Two rounds:
    1. **prepare:** I want to propose a value!

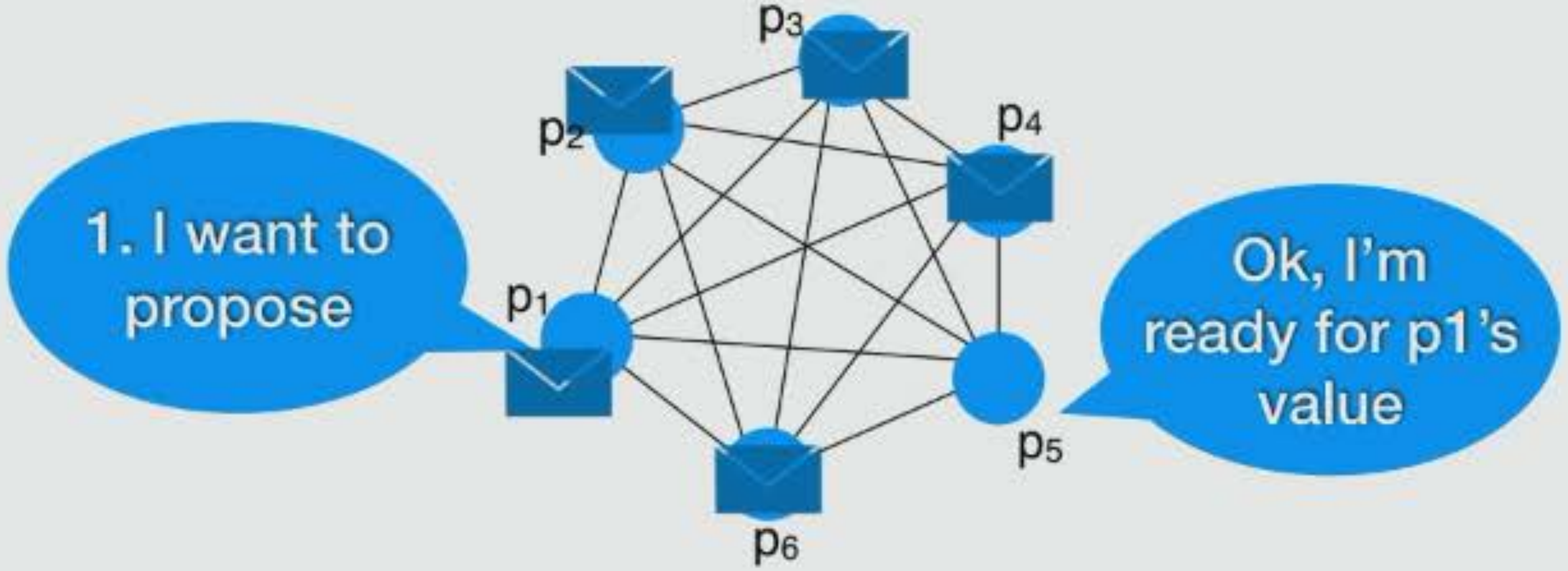


[Lamport'98]

Paxos

Message passing consensus

- Two rounds:
  1. **prepare:** I want to propose a value!

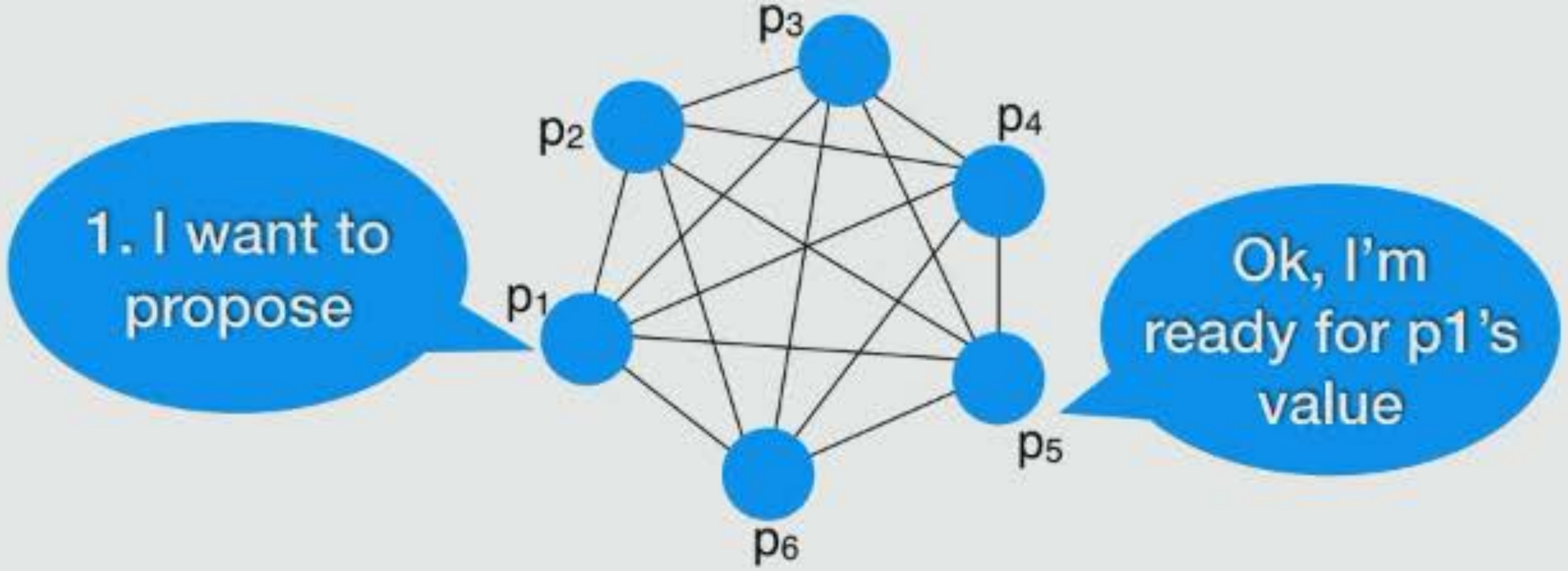


[Lamport'98]

Paxos

Message passing consensus

- Two rounds:
    1. **prepare:** I want to propose a value!

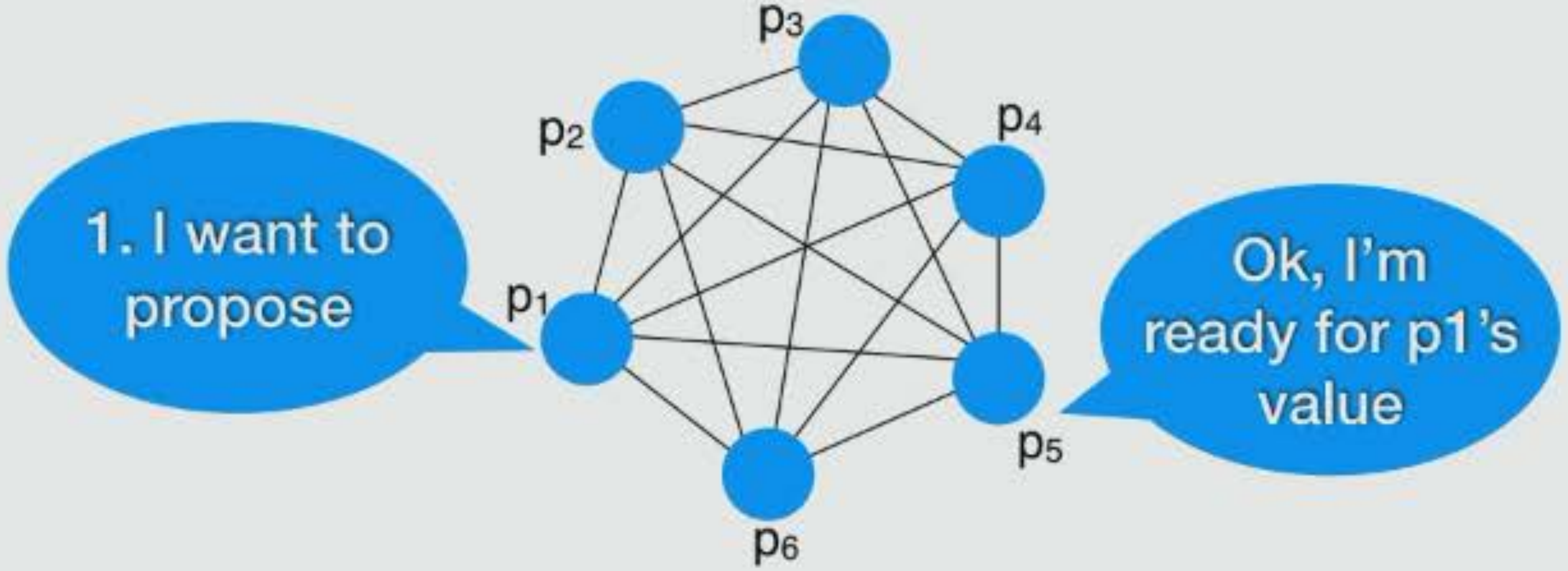


[Lamport'98]

Paxos

Message passing consensus

- Two rounds:
    1. **prepare:** I want to propose a value!
    2. **accept:** Here is my value!

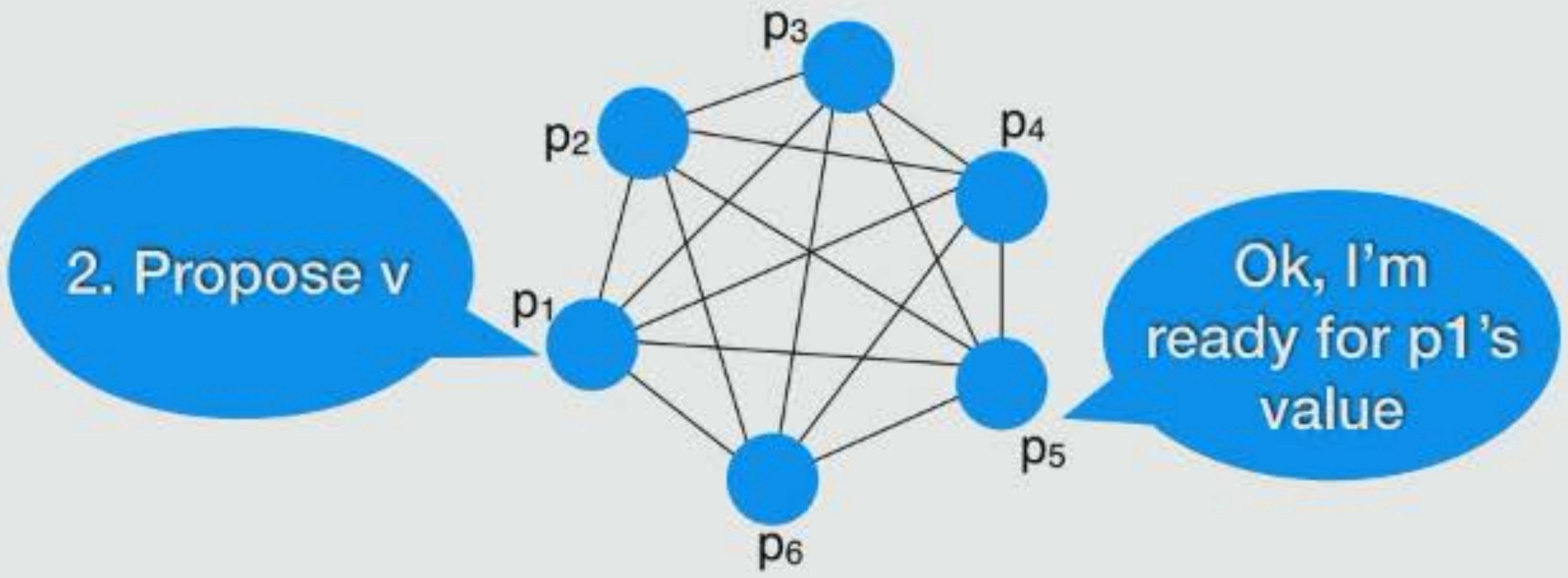


[Lamport'98]

Paxos

Message passing consensus

- Two rounds:
  1. **prepare:** I want to propose a value!
  2. **accept:** Here is my value!

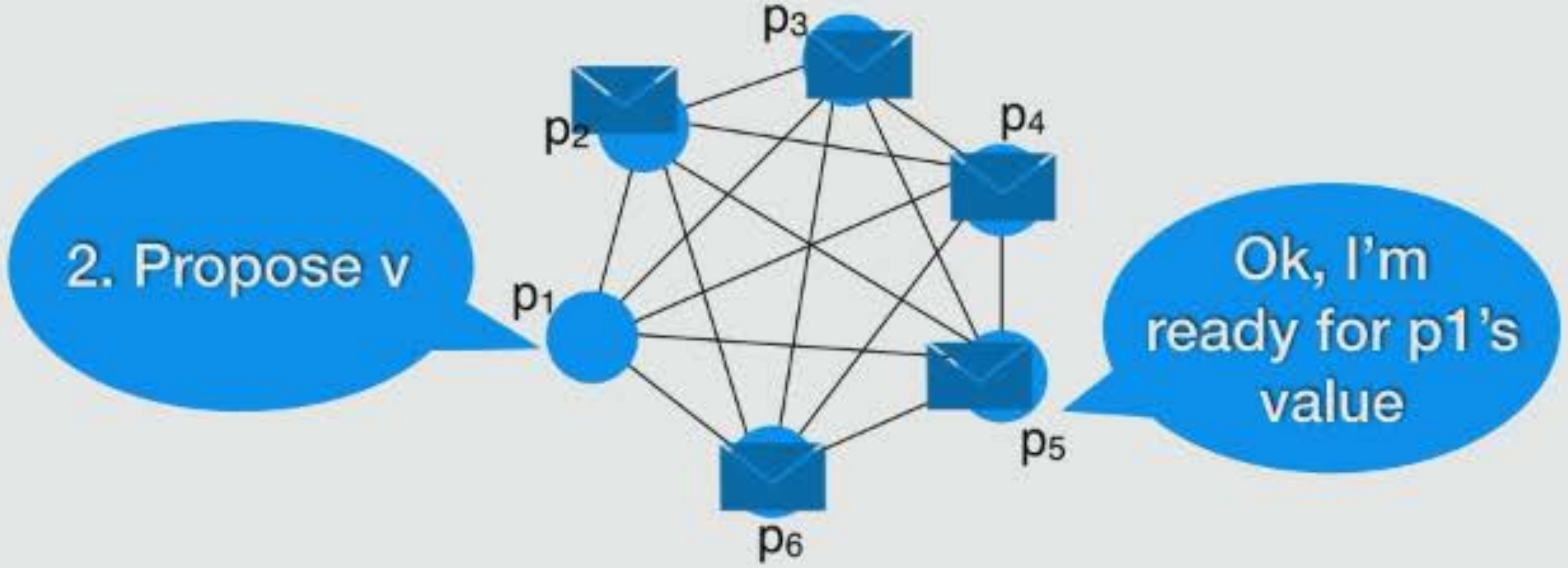


[Lamport'98]

Paxos

Message passing consensus

- Two rounds:
    1. **prepare:** I want to propose a value!
    2. **accept:** Here is my value!

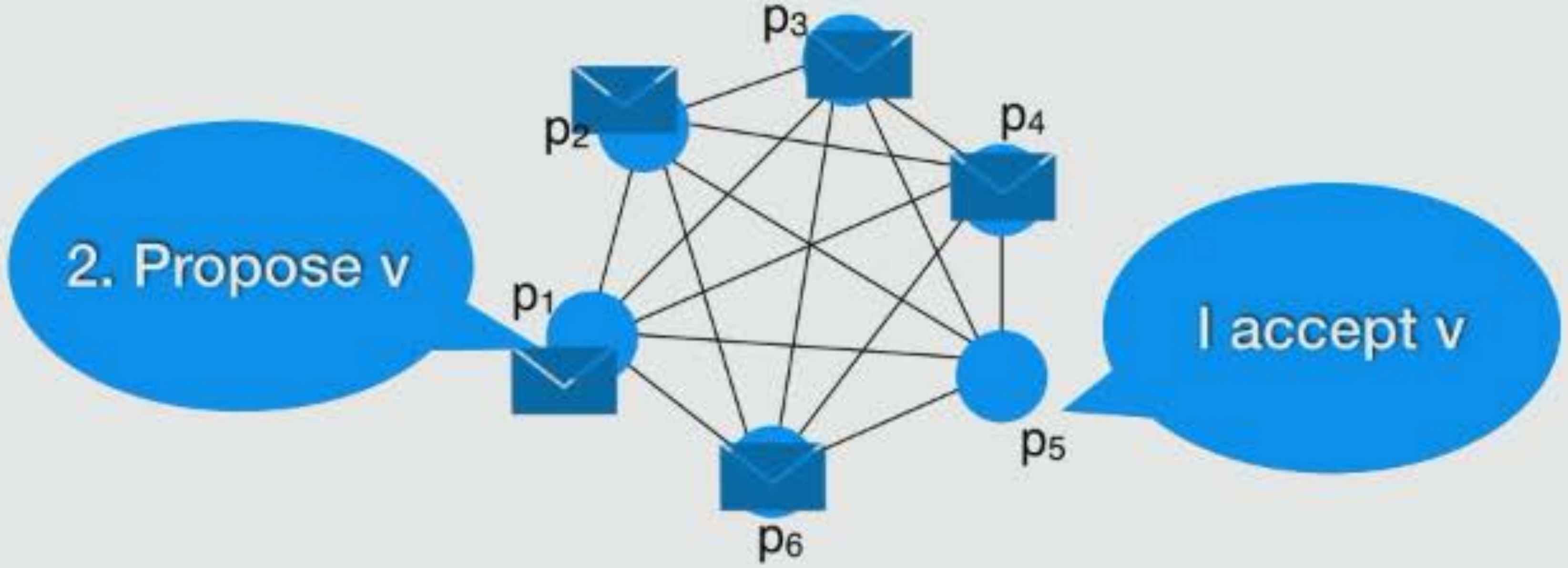


[Lamport'98]

Paxos

Message passing consensus

- Two rounds:
    1. **prepare:** I want to propose a value!
    2. **accept:** Here is my value!

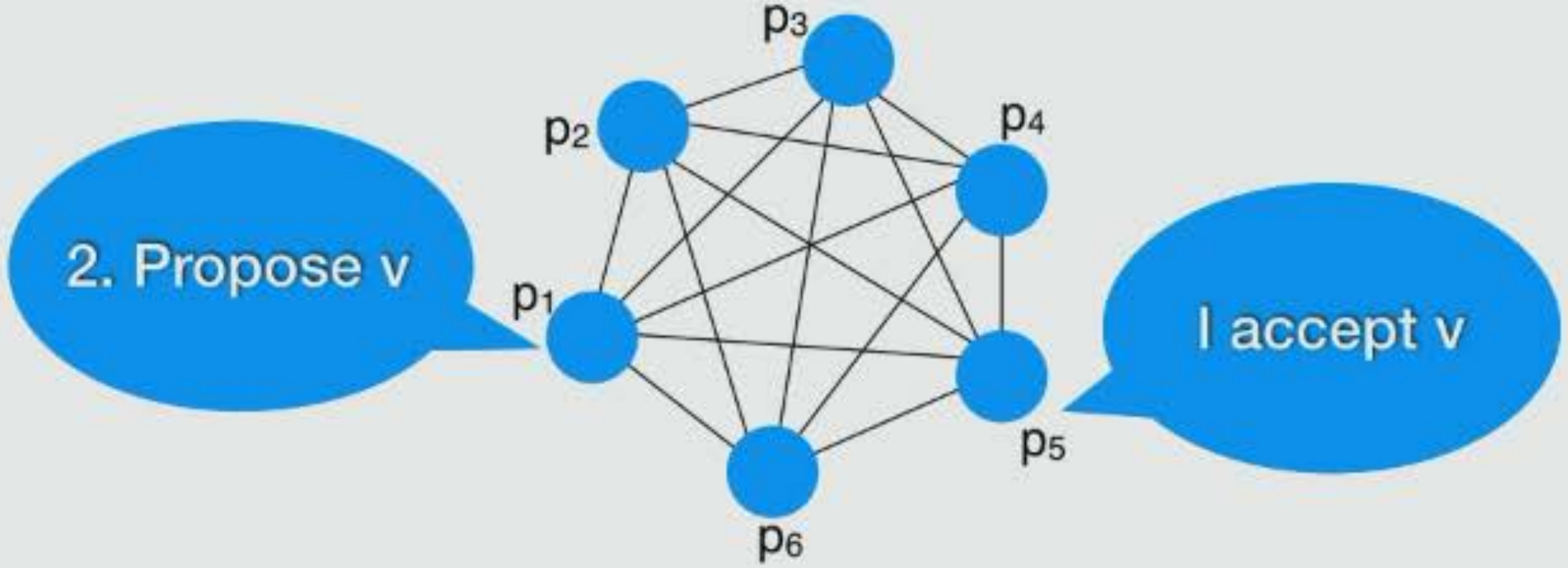


[Lamport'98]

Paxos

Message passing consensus

- Two rounds:
  1. **prepare:** I want to propose a value!
  2. **accept:** Here is my value!
- Complications: If multiple processes try, **accept only last prepared**

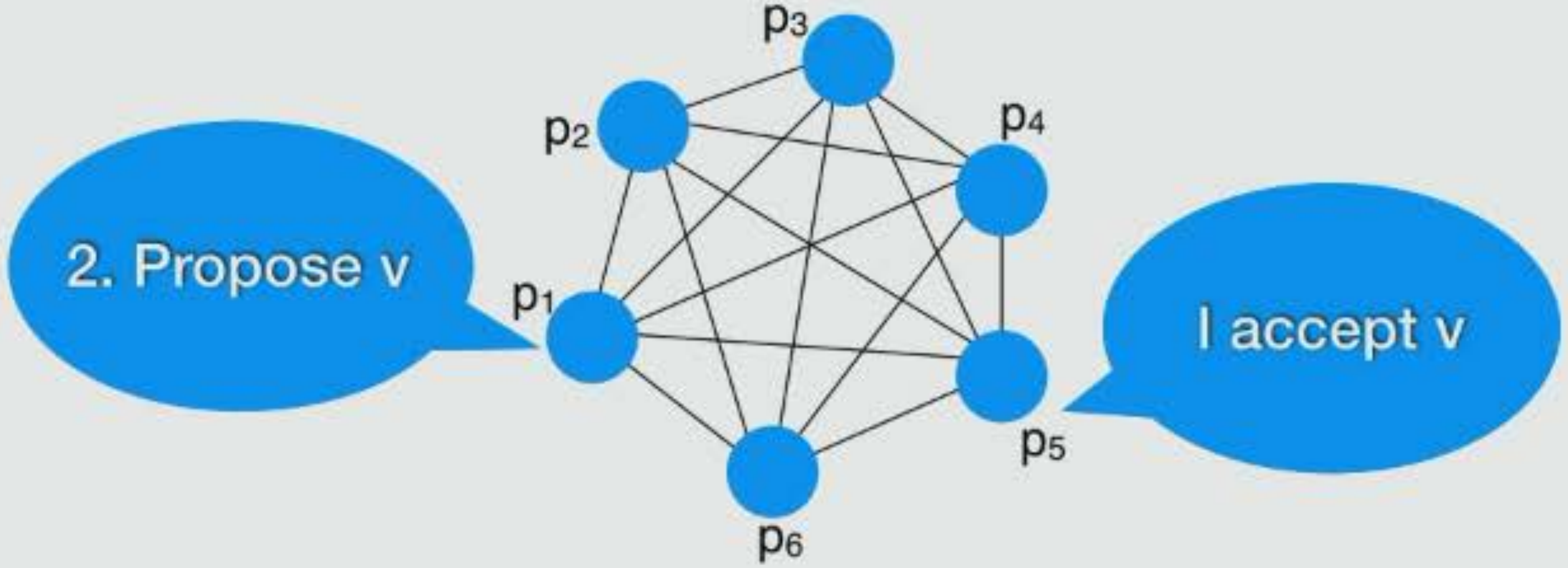


[Lamport'98]

Paxos

Message passing consensus

- Two rounds:
  1. **prepare:** I want to propose a value!
  2. **accept:** Here is my value!
- Complications: If multiple processes try, **accept only last prepared**



Idea:

- **Choose leader a priori, let it skip prepare phase**
- **If leader is slow, others start executing prepare**

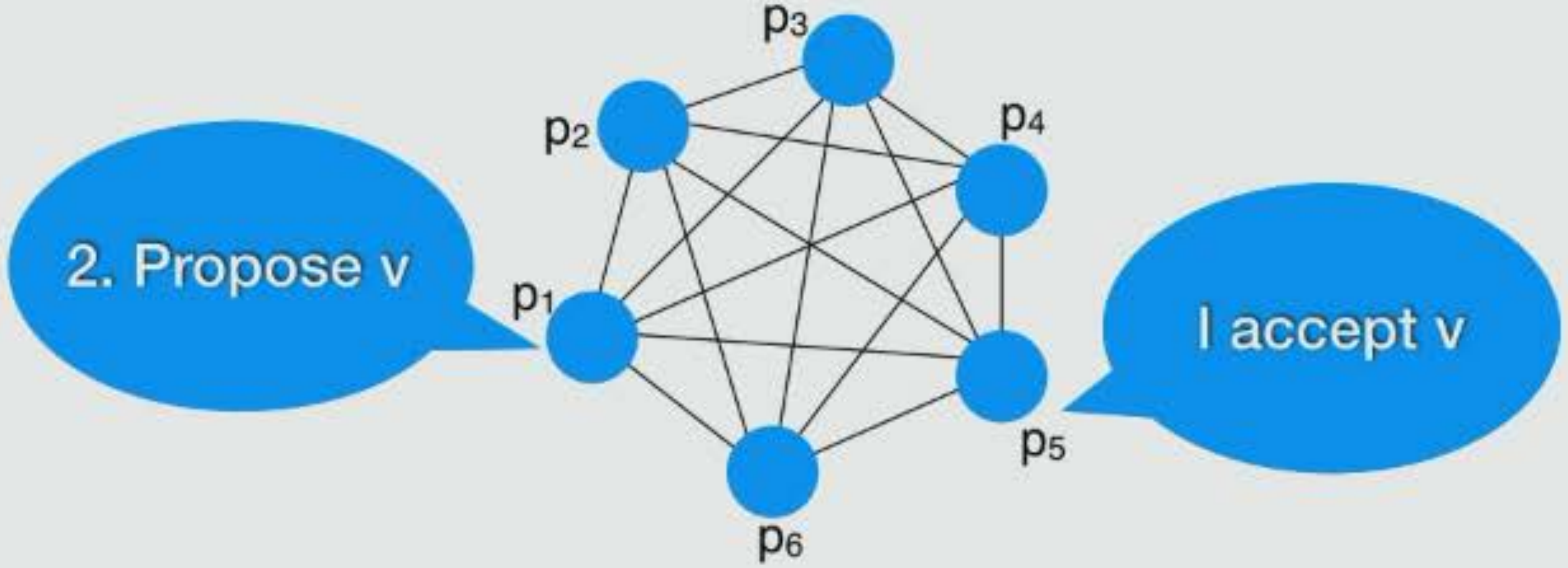
[Lamport'98]

Paxos

Message passing consensus

- Two rounds:
    1. **prepare:** I want to propose a value!
    2. **accept:** Here is my value!
- Complications: If multiple processes try, **accept only last prepared**

In the best case, only need one round!



Idea:

- **Choose leader a priori, let it skip prepare phase**
- **If leader is slow, others start executing prepare**

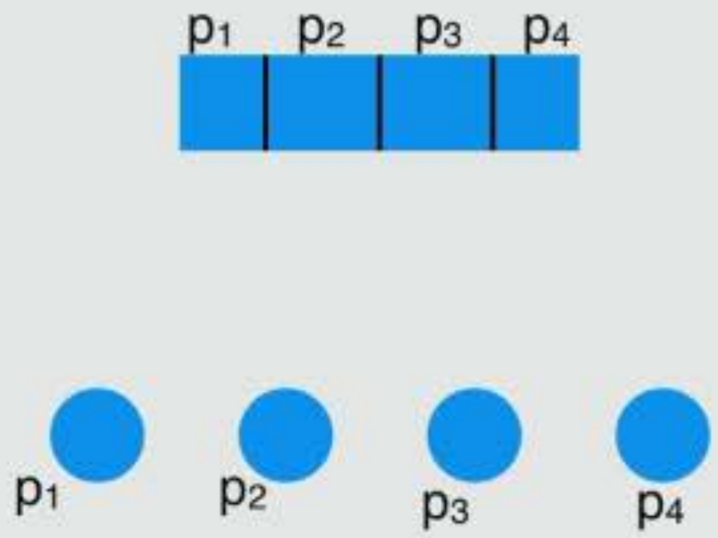
Disk Paxos

[GafniLamport'02]

Idea: run Paxos on shared memory

To send: **write your message in your slot in disk**

To receive: read others' slots in disk



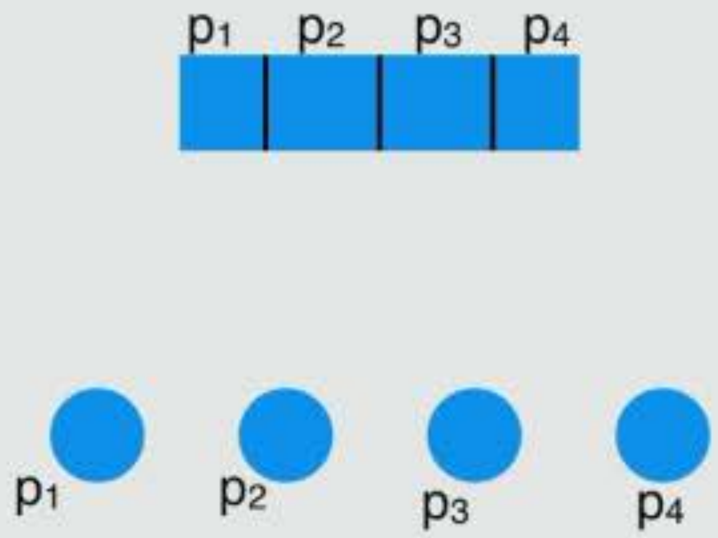
[GafniLamport'02]

Disk Paxos

Idea: run Paxos on shared memory

Fault tolerance
n>f instead of n>2f

To send: write your message in your slot in disk
To receive: read others' slots in disk



[GafniLamport'02]

Disk Paxos

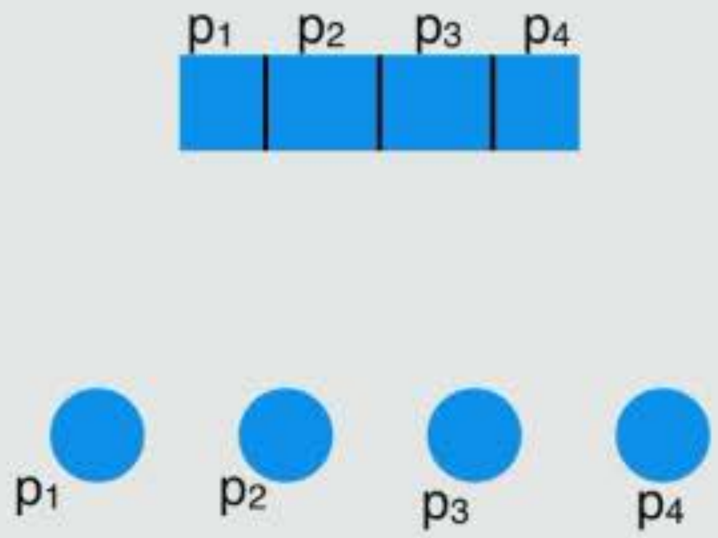
Idea: run Paxos on shared memory

Fault tolerance
n>f instead of n>2f

To send: write your message in your slot in disk
To receive: read others' slots in disk

Paxos:

- **Leader proposes its value by sending it to everyone.**
- **Everyone tells leader whether they heard from anyone else**



[GafniLamport'02]

Disk Paxos

Idea: run Paxos on shared memory

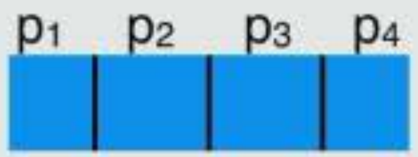
Fault tolerance
n>f instead of n>2f

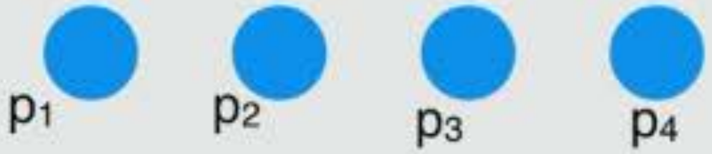
To send: write your message in your slot in disk
To receive: read others' slots in disk

Paxos:

- **Leader proposes its value by sending it to everyone.**
- **Everyone tells leader whether they heard from anyone else**

What happens in common-case execution?





[GafniLamport'02]

Disk Paxos

Idea: run Paxos on shared memory

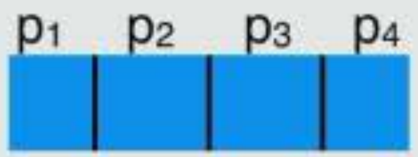
Fault tolerance
n>f instead of n>2f

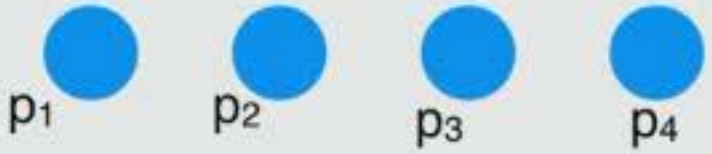
To send: write your message in your slot in disk
To receive: read others' slots in disk

Paxos:
- **Leader proposes its value by sending it to everyone.**
- **Everyone tells leader whether they heard from anyone else**

What happens in common-case execution?

- p1 proposes by writing





[GafniLamport'02]

Disk Paxos

Idea: run Paxos on shared memory

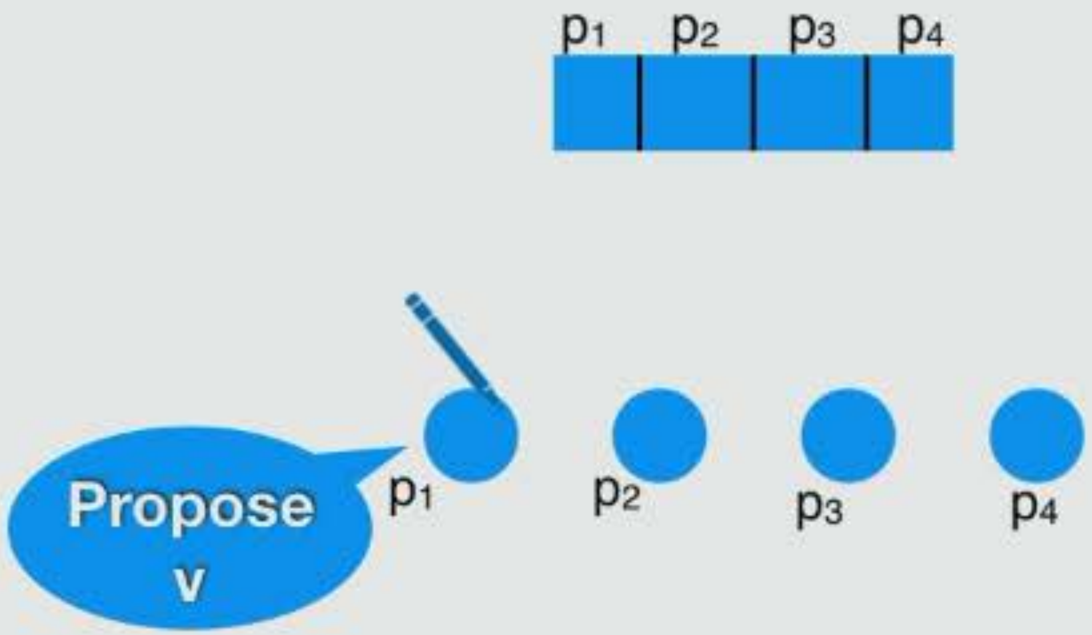
Fault tolerance
n>f instead of n>2f

To send: write your message in your slot in disk
To receive: read others' slots in disk

Paxos:
- **Leader proposes its value by sending it to everyone.**
- **Everyone tells leader whether they heard from anyone else**

What happens in common-case execution?

- p1 proposes by writing



[GafniLamport'02]

Disk Paxos

Idea: run Paxos on shared memory

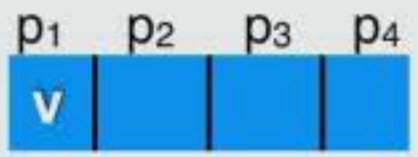
Fault tolerance
n>f instead of n>2f

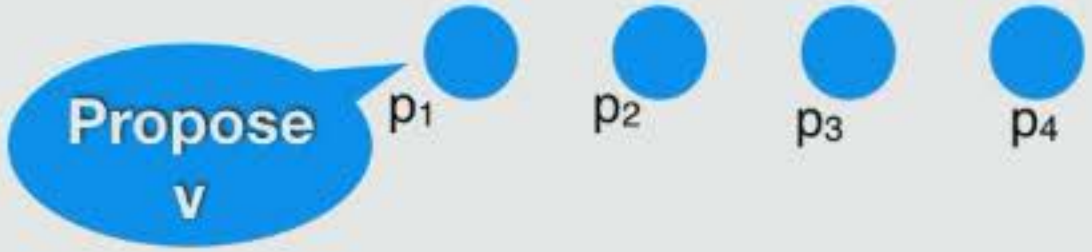
To send: write your message in your slot in disk
To receive: read others' slots in disk

Paxos:
- **Leader proposes its value by sending it to everyone.**
- **Everyone tells leader whether they heard from anyone else**

What happens in common-case execution?

- p1 proposes by writing





[GafniLamport'02]

Disk Paxos

Idea: run Paxos on shared memory

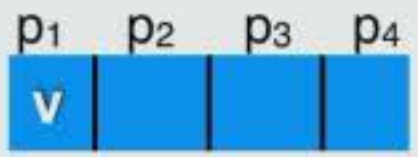
Fault tolerance
n>f instead of n>2f

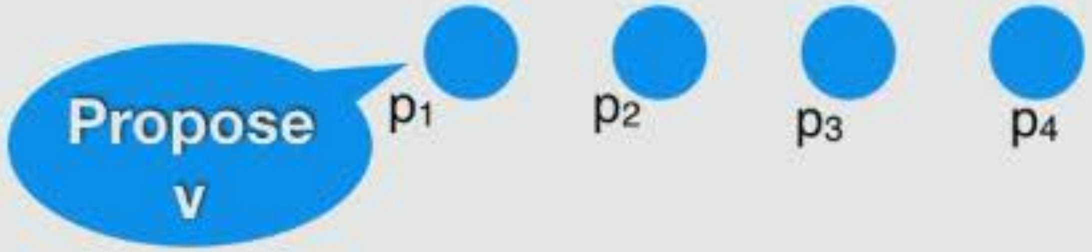
To send: write your message in your slot in disk
To receive: read others' slots in disk

Paxos:
- **Leader proposes its value by sending it to everyone.**
- **Everyone tells leader whether they heard from anyone else**

What happens in common-case execution?

- p1 proposes by writing
- p1 must read all slots to ensure it's the only proposer





Disk Paxos

[GafniLamport'02]

Idea: run Paxos on shared memory

Fault tolerance
n>f instead of n>2f

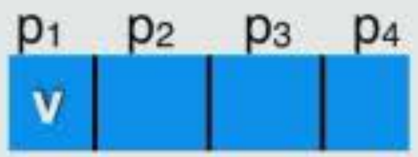
To send: write your message in your slot in disk
To receive: read others' slots in disk

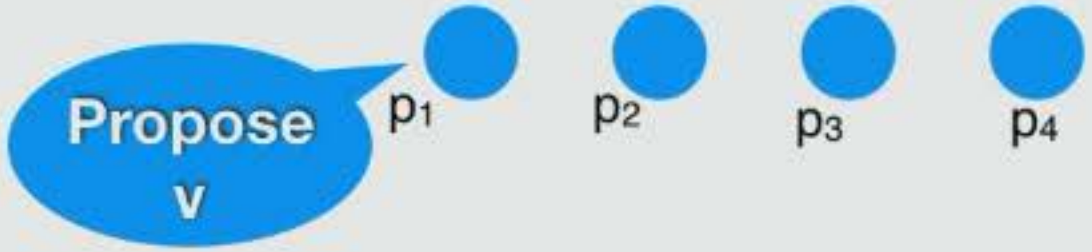
Paxos:
- **Leader proposes its value by sending it to everyone.**
- **Everyone tells leader whether they heard from anyone else**

What happens in common-case execution?

- p1 proposes by writing
- p1 must read all slots to ensure it's the only proposer

P1 doesn't know it's a good execution!





Disk Paxos

[GafniLamport'02]

Idea: run Paxos on shared memory

Fault tolerance
n>f instead of n>2f

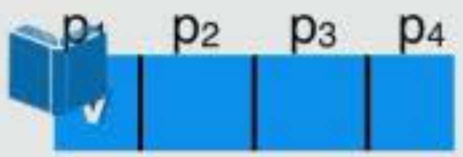
To send: write your message in your slot in disk
To receive: read others' slots in disk

Paxos:
- **Leader proposes its value by sending it to everyone.**
- **Everyone tells leader whether they heard from anyone else**

What happens in common-case execution?

- p1 proposes by writing
- p1 must read all slots to ensure it's the only proposer

P1 doesn't know it's a good execution!



[GafniLamport'02]

Disk Paxos

Idea: run Paxos on shared memory

Fault tolerance
n>f instead of n>2f

To send: write your message in your slot in disk
To receive: read others' slots in disk

Paxos:

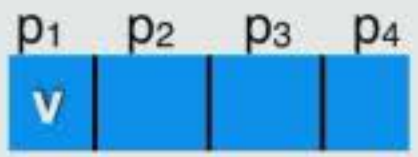
- **Leader proposes its value by sending it to everyone.**
- **Everyone tells leader whether they heard from anyone else**

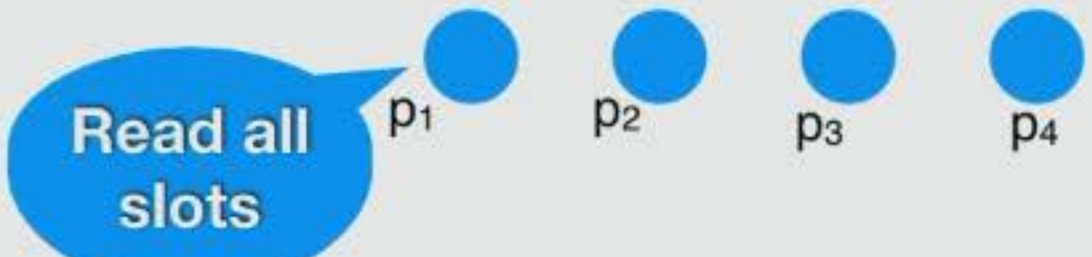
What happens in common-case execution?

- p1 proposes by writing
- p1 must read all slots to ensure it's the only proposer

P1 doesn't know it's a good execution!

Time (in round trips):





[GafniLamport'02]

Disk Paxos

Idea: run Paxos on shared memory

Fault tolerance
n>f instead of n>2f

To send: write your message in your slot in disk
To receive: read others' slots in disk

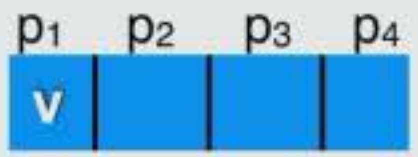
Paxos:
- **Leader proposes its value by sending it to everyone.**
- **Everyone tells leader whether they heard from anyone else**

What happens in common-case execution?

Time (in round trips):

- p1 proposes by writing — **1**
- p1 must read all slots to ensure it's the only proposer — **1**

total: 2



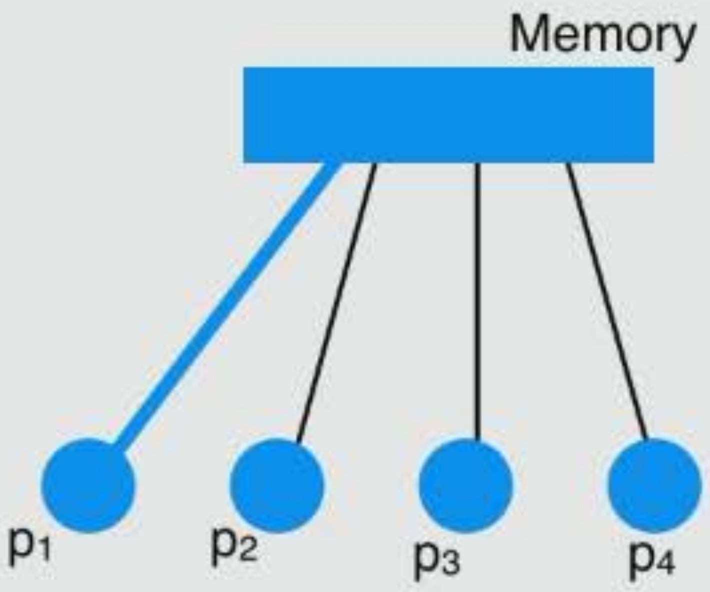
P1 doesn't know it's a good execution!

[ABGMZ'19]

Disk Paxos with Permissions

In Disk, proposer must **read every value from disk** to know whether someone is competing with it.

Idea: leverage RDMA dynamic permissions to get rid of this step.

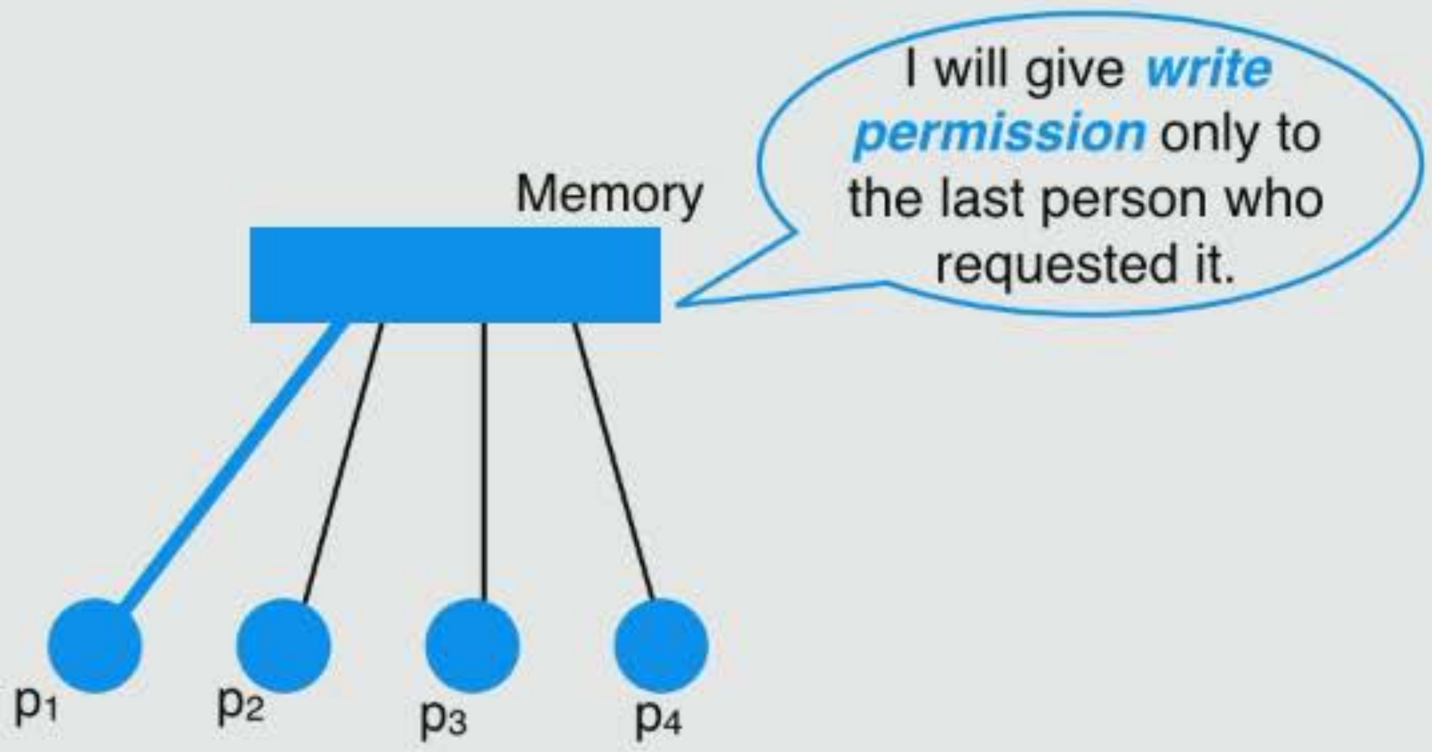


Disk Paxos with Permissions

[ABGMZ'19]

In Disk, proposer must **read every value from disk** to know whether someone is competing with it.

Idea: leverage RDMA dynamic permissions to get rid of this step.

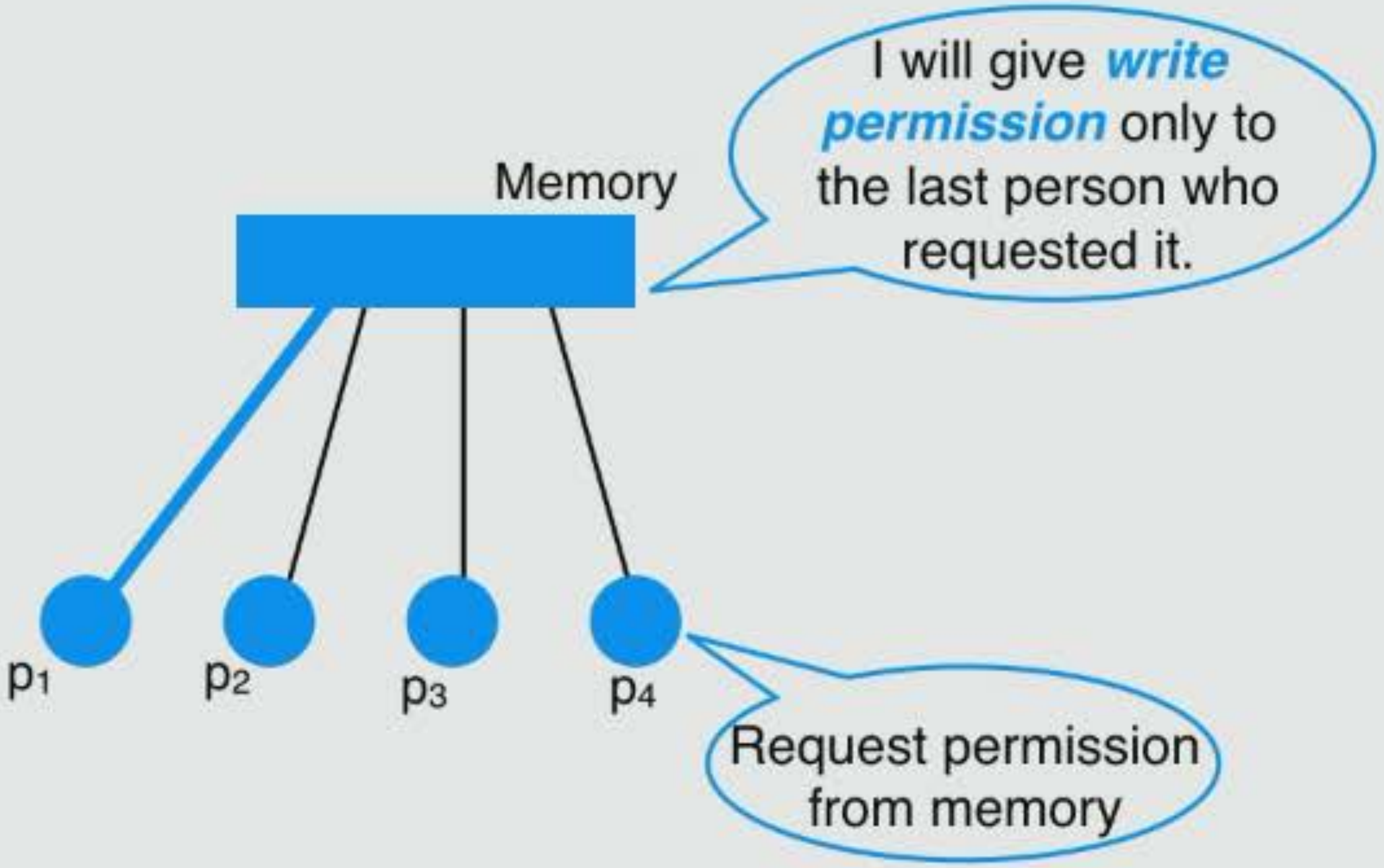


[ABGMZ'19]

Disk Paxos with Permissions

In Disk, proposer must **read every value from disk** to know whether someone is competing with it.

Idea: leverage RDMA dynamic permissions to get rid of this step.

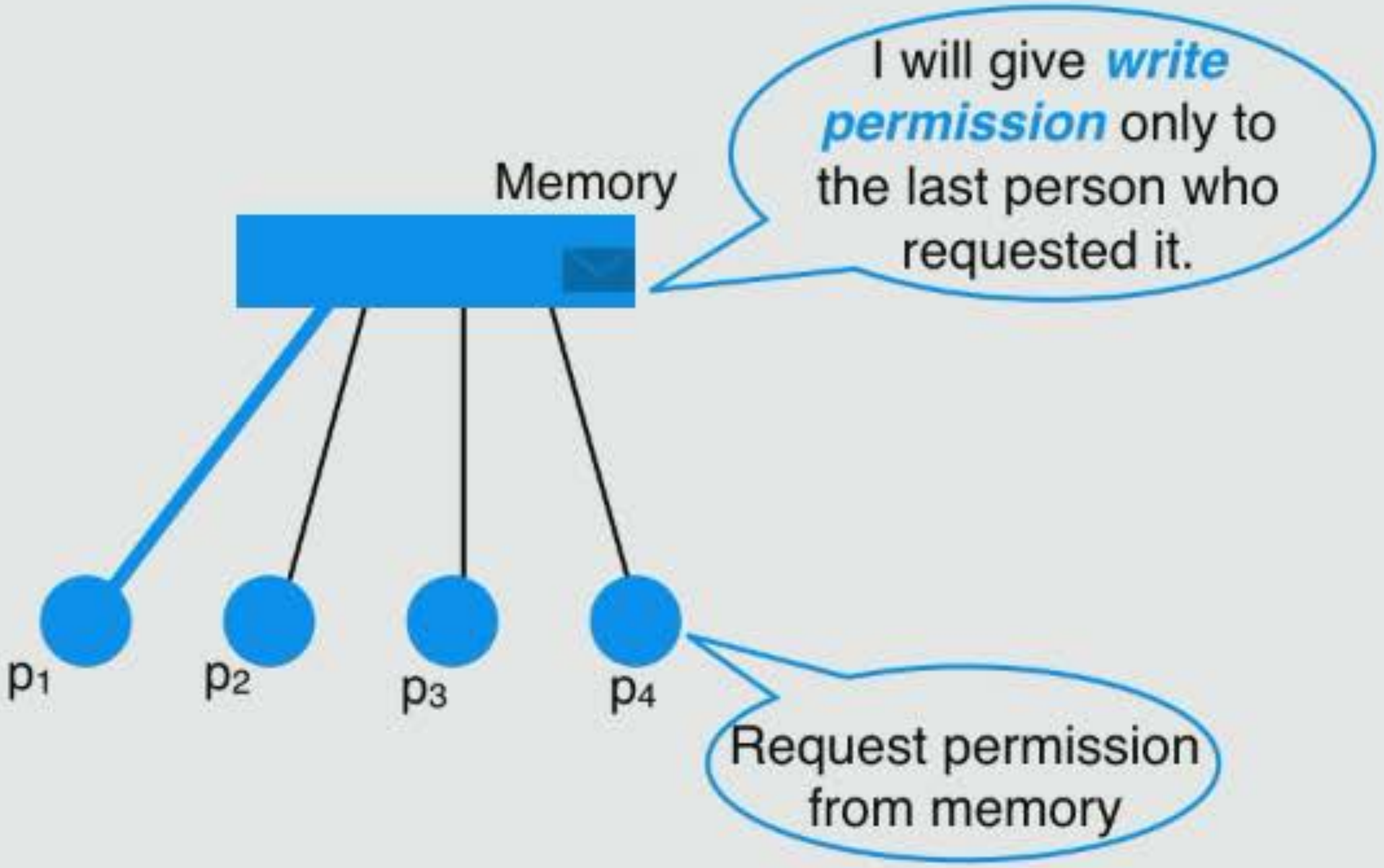


Disk Paxos with Permissions

[ABGMZ'19]

In Disk, proposer must **read every value from disk** to know whether someone is competing with it.

Idea: leverage RDMA dynamic permissions to get rid of this step.

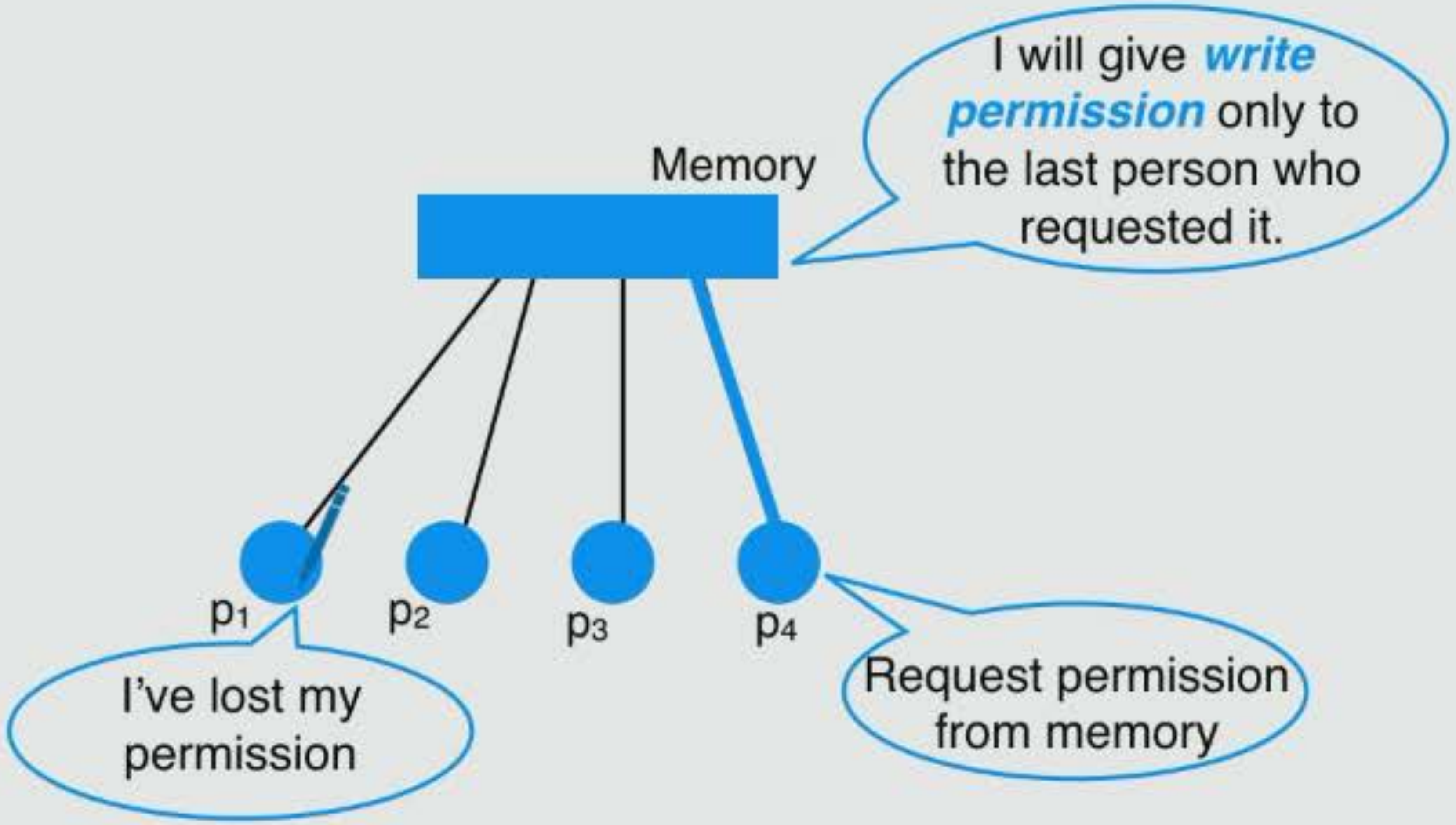


Disk Paxos with Permissions

[ABGMZ'19]

In Disk, proposer must **read every value from disk** to know whether someone is competing with it.

Idea: leverage RDMA dynamic permissions to get rid of this step.



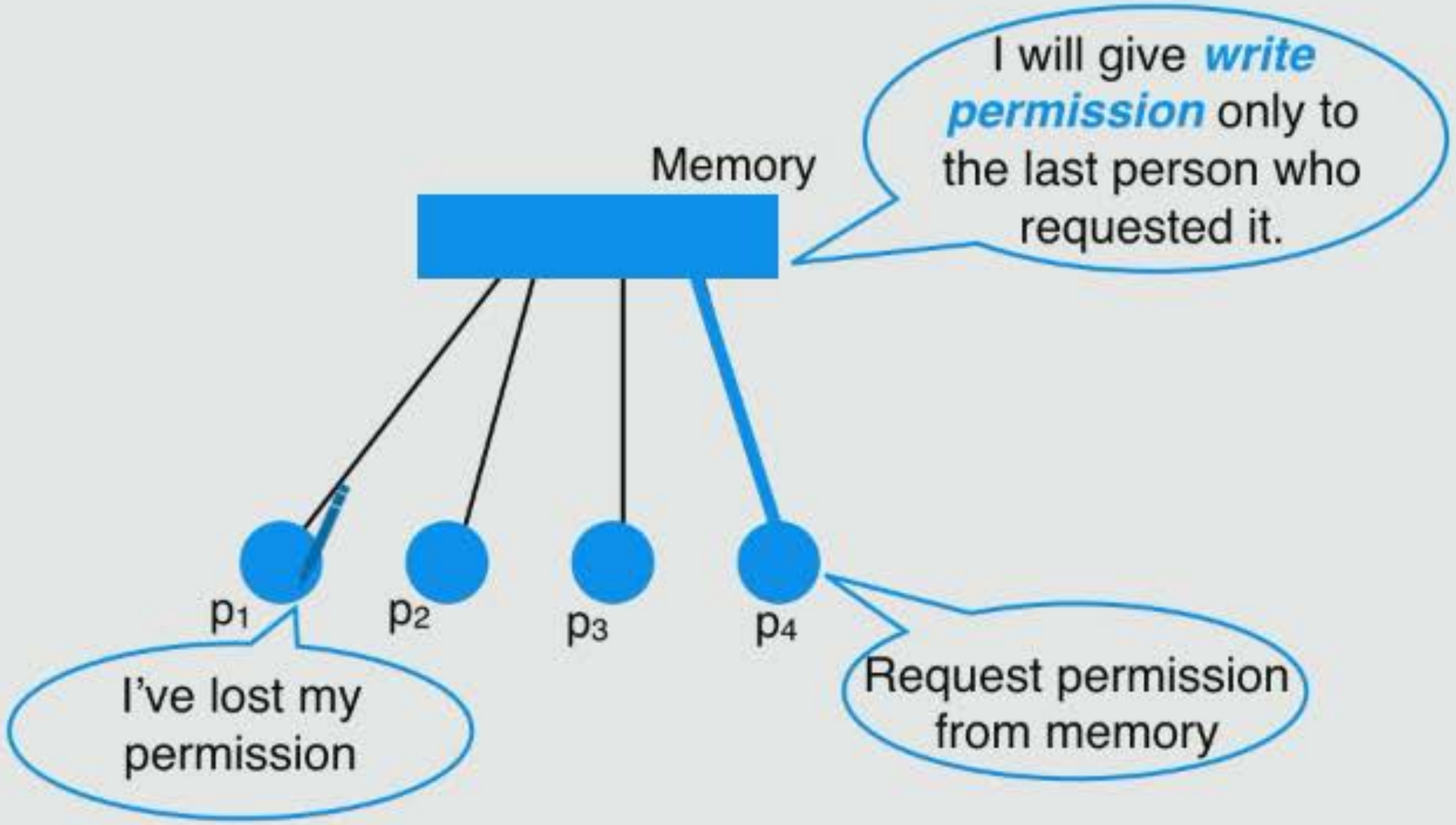
[ABGMZ'19]

Disk Paxos with Permissions

In Disk, proposer must **read every value from disk** to know whether someone is competing with it.

Idea: leverage RDMA dynamic permissions to get rid of this step.

Lost permission iff contention



[ABGMZ'19]

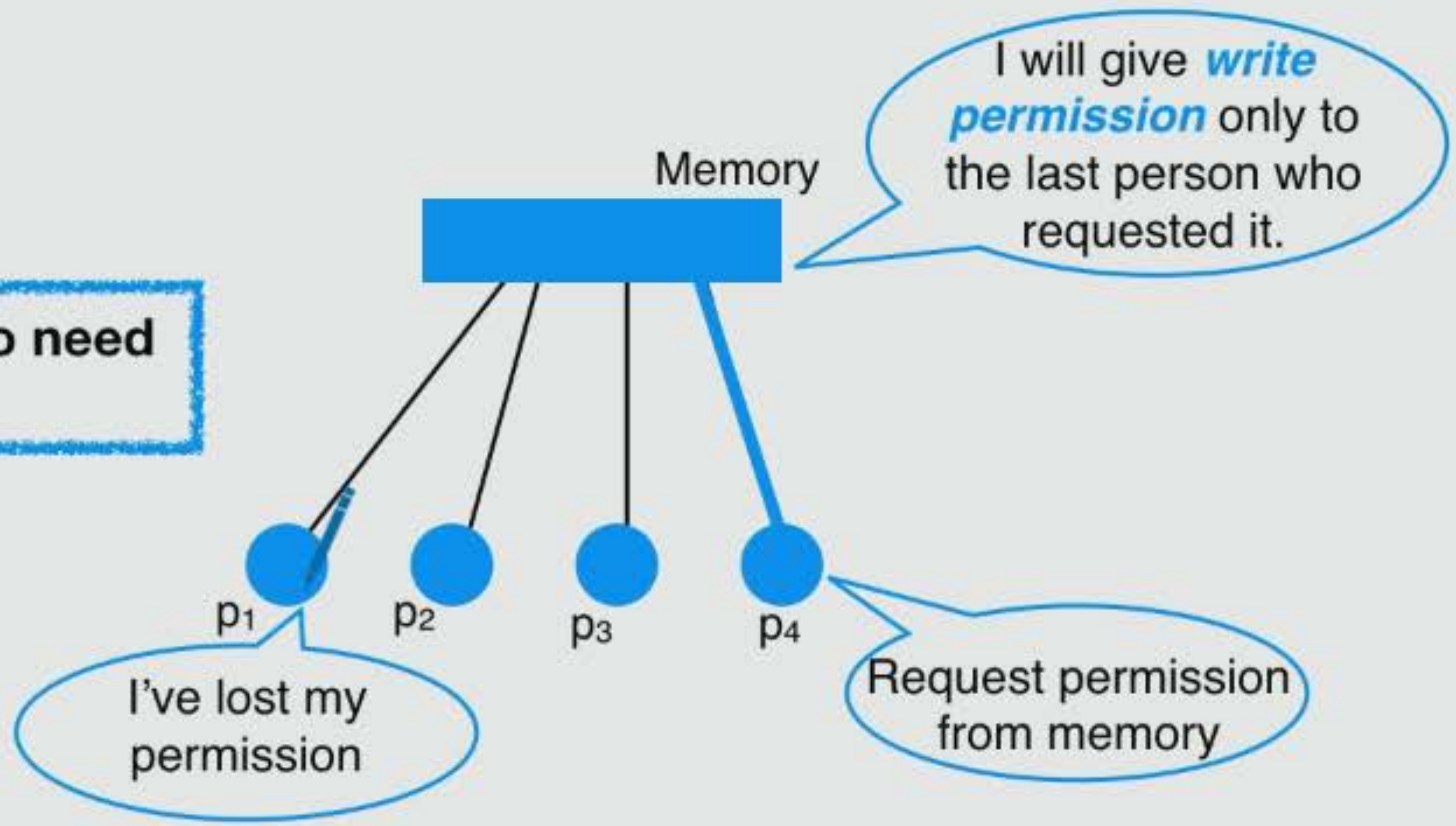
Disk Paxos with Permissions

In Disk, proposer must **read every value from disk** to know whether someone is competing with it.

Idea: leverage RDMA dynamic permissions to get rid of this step.

Lost permission iff contention

If wrote successfully, no need to read!



[ABGMZ'19]

Disk Paxos with Permissions

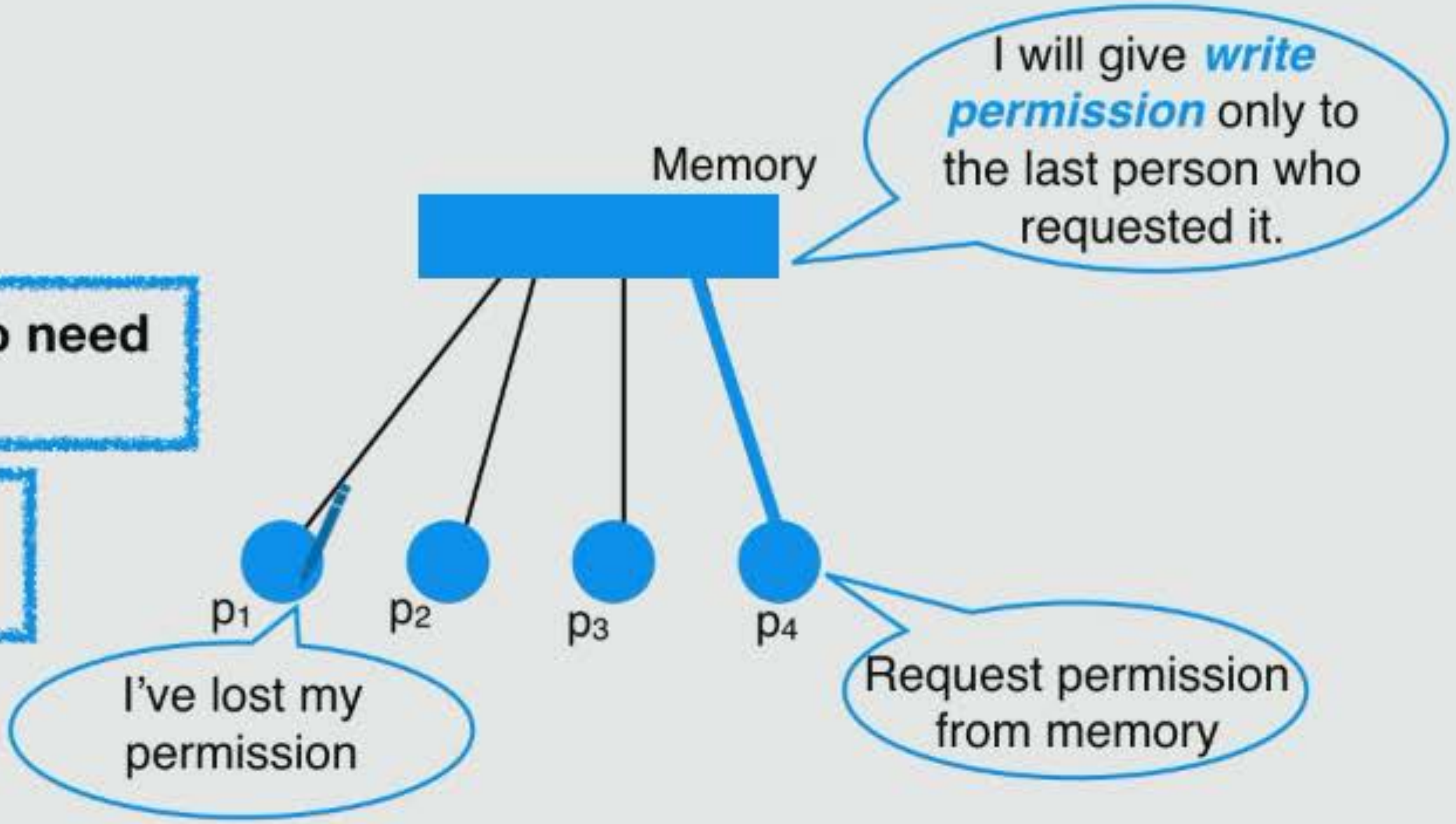
In Disk, proposer must **read every value from disk** to know whether someone is competing with it.

Idea: leverage RDMA dynamic permissions to get rid of this step.

Lost permission iff contention

If wrote successfully, no need to read!

2 round trips —> 1 round trip



Outline

✓ **RDMA details**

- Column 1: **RDMA's full power** (complete graph)
  - ✓ **Crash-only** algorithm: ***n>f*** tolerant, 1 round-trip
  - **Byzantine** algorithm: ***n>2f*** tolerant, 1 round-trip

- Column 2: **Scalability: Using RDMA sparingly** (incomplete graph)
  - Crash-only Algorithm: tolerance vs topology

[A**B**GMZ'19]

Byzantine Algorithm Breakdown

Two pieces:

[ABGMZ'19]

Byzantine Algorithm Breakdown

Two pieces:

- **CheapQuorum:** Fast (1 round trip) algorithm that **aborts** at first sign of trouble

[ABGMZ’19]

Byzantine Algorithm Breakdown

Two pieces:

- **CheapQuorum:** Fast (1 round trip) algorithm that **aborts** at first sign of trouble

CheapQuorum

[ABGMZ'19]

Byzantine Algorithm Breakdown

Two pieces:

- **CheapQuorum:** Fast (1 round trip) algorithm that **aborts** at first sign of trouble
    - Use permissions to get speed

CheapQuorum

[ABGMZ'19]

Byzantine Algorithm Breakdown

Two pieces:

- **CheapQuorum:** Fast (1 round trip) algorithm that **aborts** at first sign of trouble
  - Use permissions to get speed
- **Robust Backup:** Slow algorithm that is tolerant to $n > 2f$ Byzantine failures

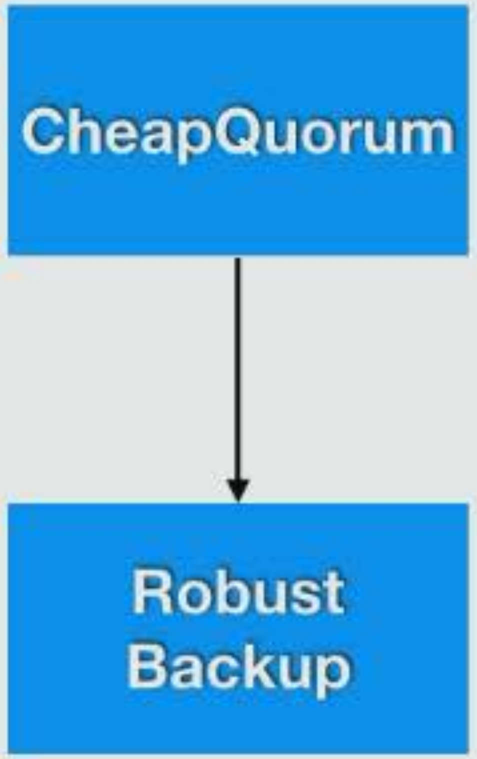
CheapQuorum

[ABGMZ'19]

Byzantine Algorithm Breakdown

Two pieces:

- **CheapQuorum:** Fast (1 round trip) algorithm that **aborts** at first sign of trouble
    - Use permissions to get speed
- **Robust Backup:** Slow algorithm that is tolerant to n > 2f Byzantine failures
    - Shared memory algorithm



[ABGMZ'19]

Robust Algorithm

- High level idea: run Paxos, but replace messaging primitives (send/receive) with special ***non-equivocating broadcast/deliver***

[A**B**GMZ'19]

Robust Algorithm

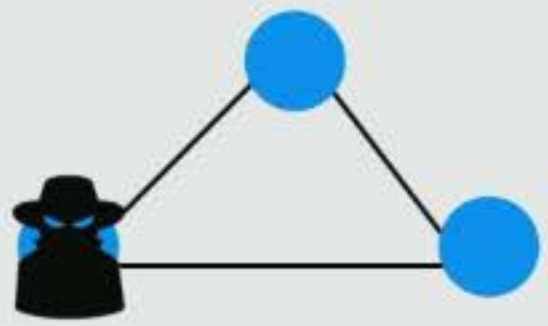
- High level idea: run Paxos, but replace messaging primitives (send/receive) with special ***non-equivocating broadcast/deliver***

  Equivocation:

[ABGMZ'19] Robust Algorithm

- High level idea: run Paxos, but replace messaging primitives (send/receive) with special ***non-equivocating broadcast/ deliver***

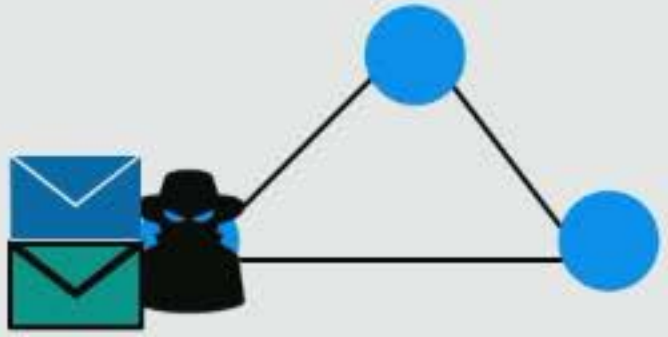
  Equivocation:



[ABGMZ'19] Robust Algorithm

- High level idea: run Paxos, but replace messaging primitives (send/receive) with special ***non-equivocating broadcast/ deliver***

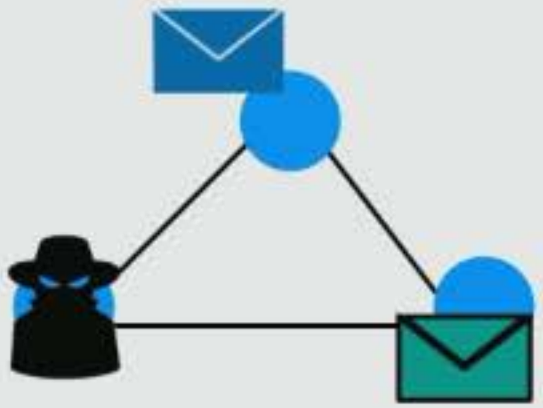
  Equivocation:



[ABGMZ'19] Robust Algorithm

- High level idea: run Paxos, but replace messaging primitives (send/receive) with special ***non-equivocating broadcast/deliver***

  Equivocation:



[ABGMZ'19]

Robust Algorithm

- High level idea: run Paxos, but replace messaging primitives (send/receive) with special ***non-equivocating broadcast/ deliver***

  Equivocation:

Broadcast primitive prevents this behavior

[ABGMZ'19] Robust Algorithm

- High level idea: run Paxos, but replace messaging primitives (send/receive) with special ***non-equivocating broadcast/ deliver***

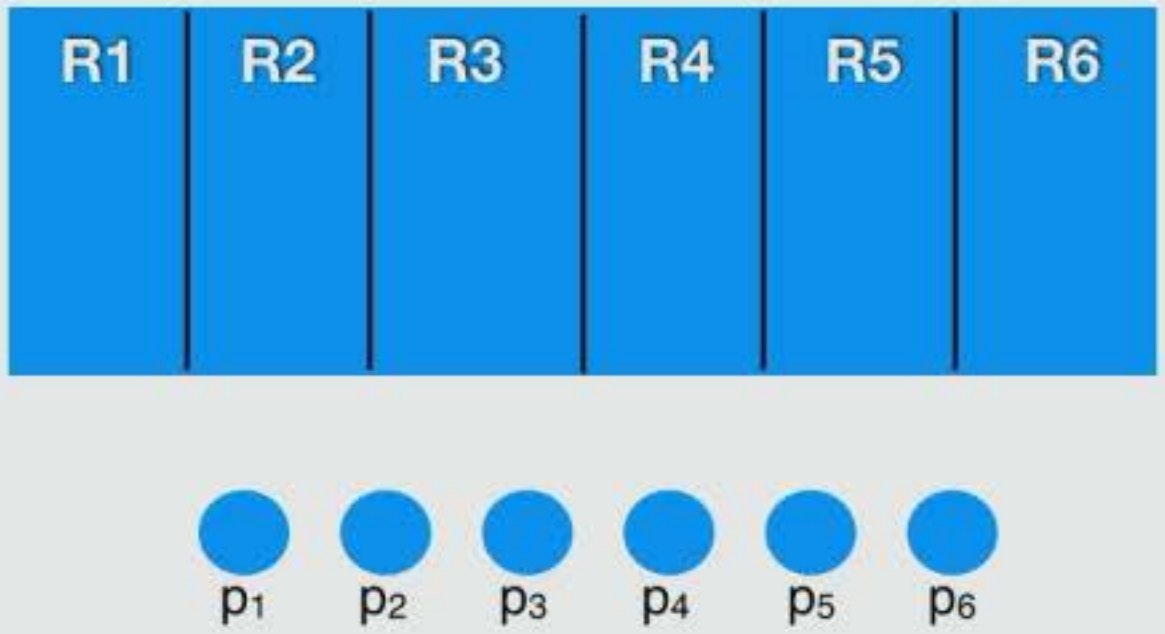
  Equivocation:

- If we can prevent equivocation, then we can solve Byzantine agreement with $n > 2f$ [ClementJunqueiraKateRodrigues'12]

[ABGMZ'19]

Preventing Equivocation

Single Writer Multi Reader region per process

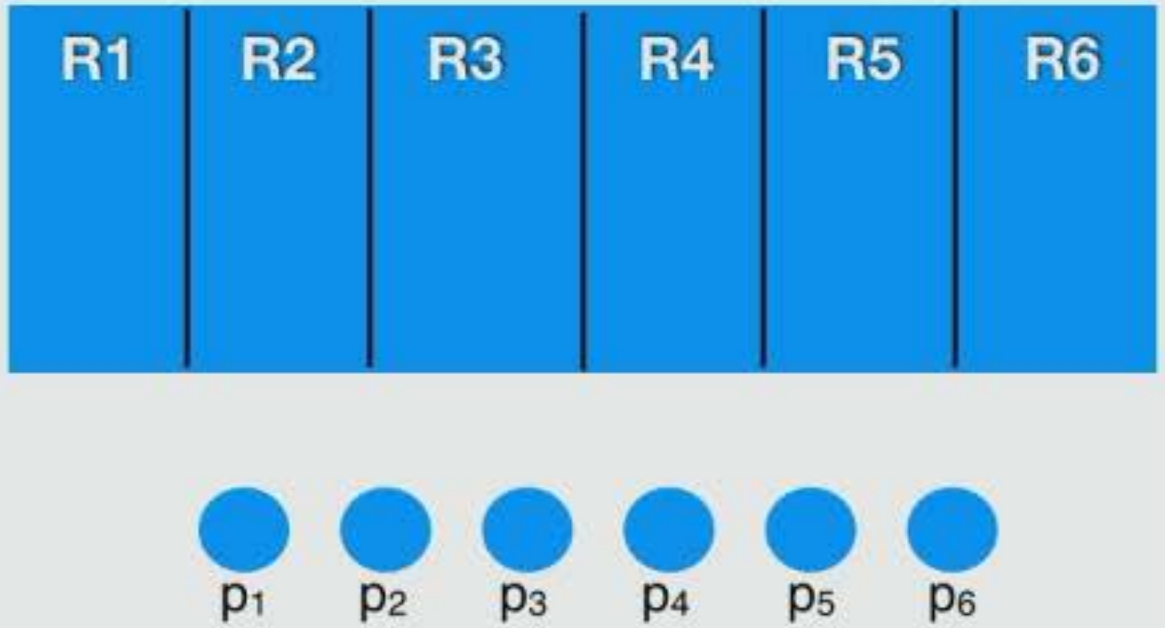


Each process gets its own SWMR region

[ABGMZ'19]

Preventing Equivocation

Single Writer Multi Reader region per process

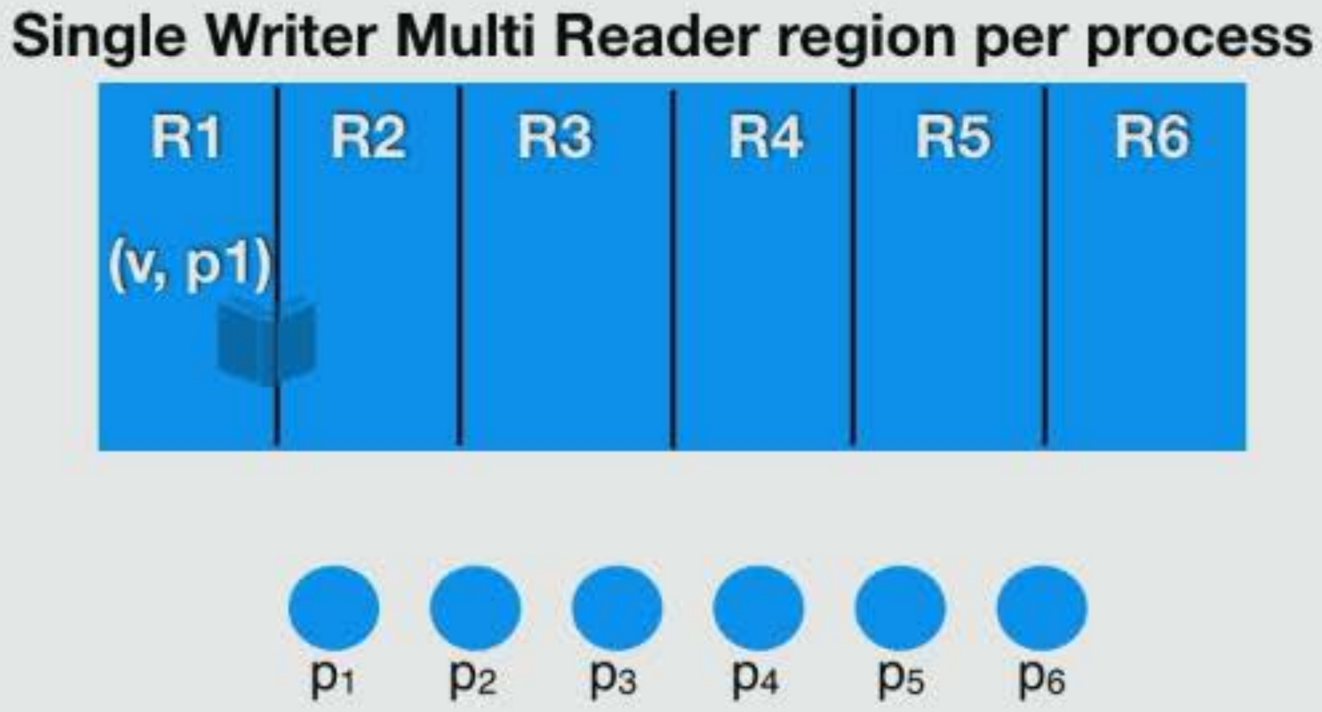


Each process gets its own SWMR region

Protocol: Sign and copy over everything that you see

Preventing Equivocation

[ABGMZ'19]



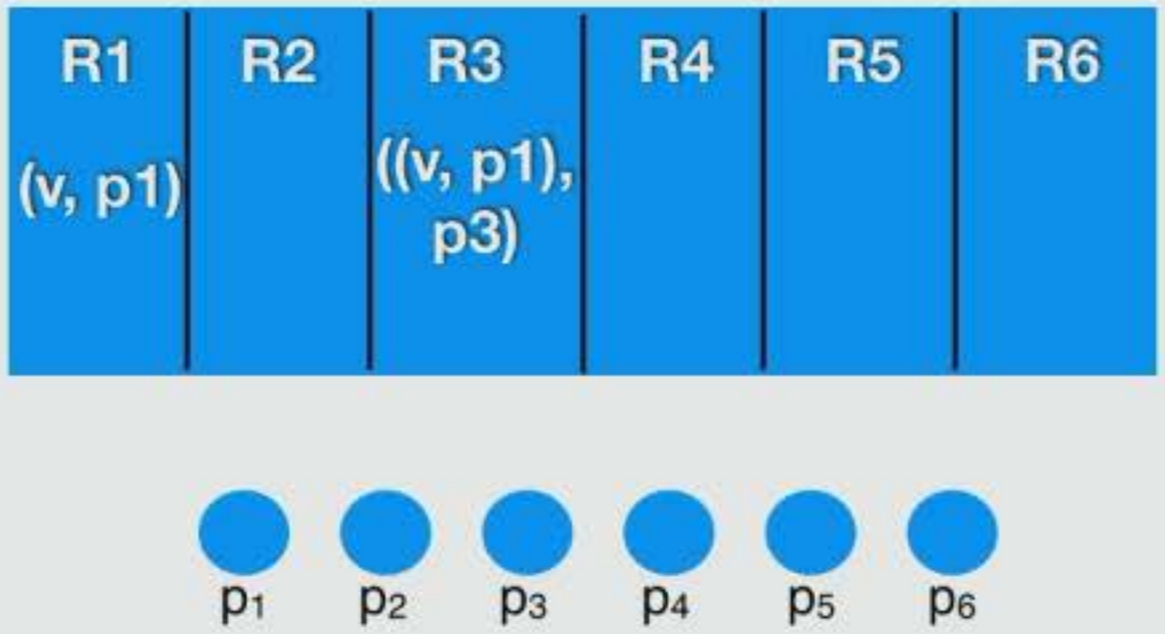
Each process gets its own SWMR region

Protocol: Sign and copy over everything that you see

[ABGMZ'19]

Preventing Equivocation

Single Writer Multi Reader region per process



Each process gets its own SWMR region

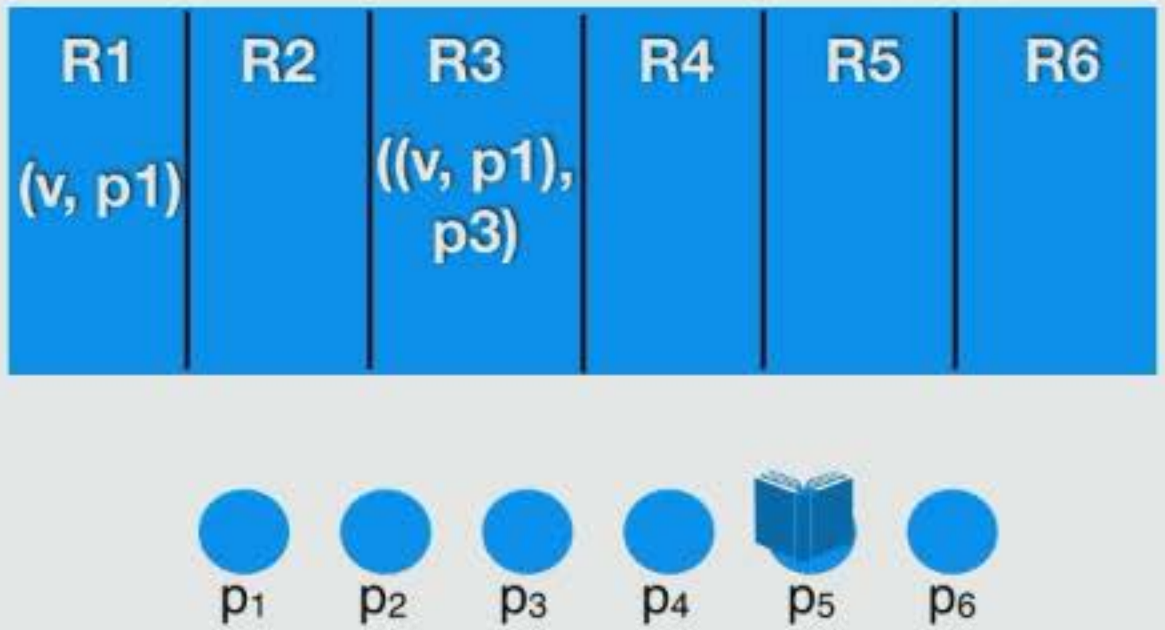
Protocol: Sign and copy over everything that you see

Can now verify that others read the same value

[ABGMZ'19]

Preventing Equivocation

Single Writer Multi Reader region per process



Each process gets its own SWMR region

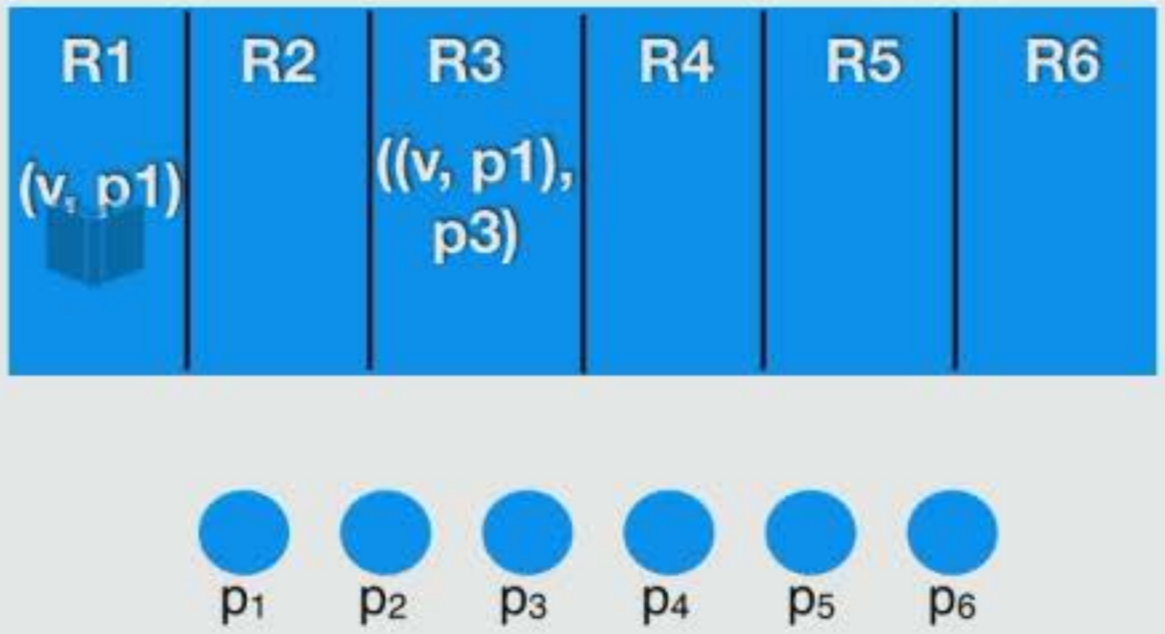
Protocol: Sign and copy over everything that you see

Can now verify that others read the same value

[ABGMZ'19]

Preventing Equivocation

Single Writer Multi Reader region per process



Each process gets its own SWMR region

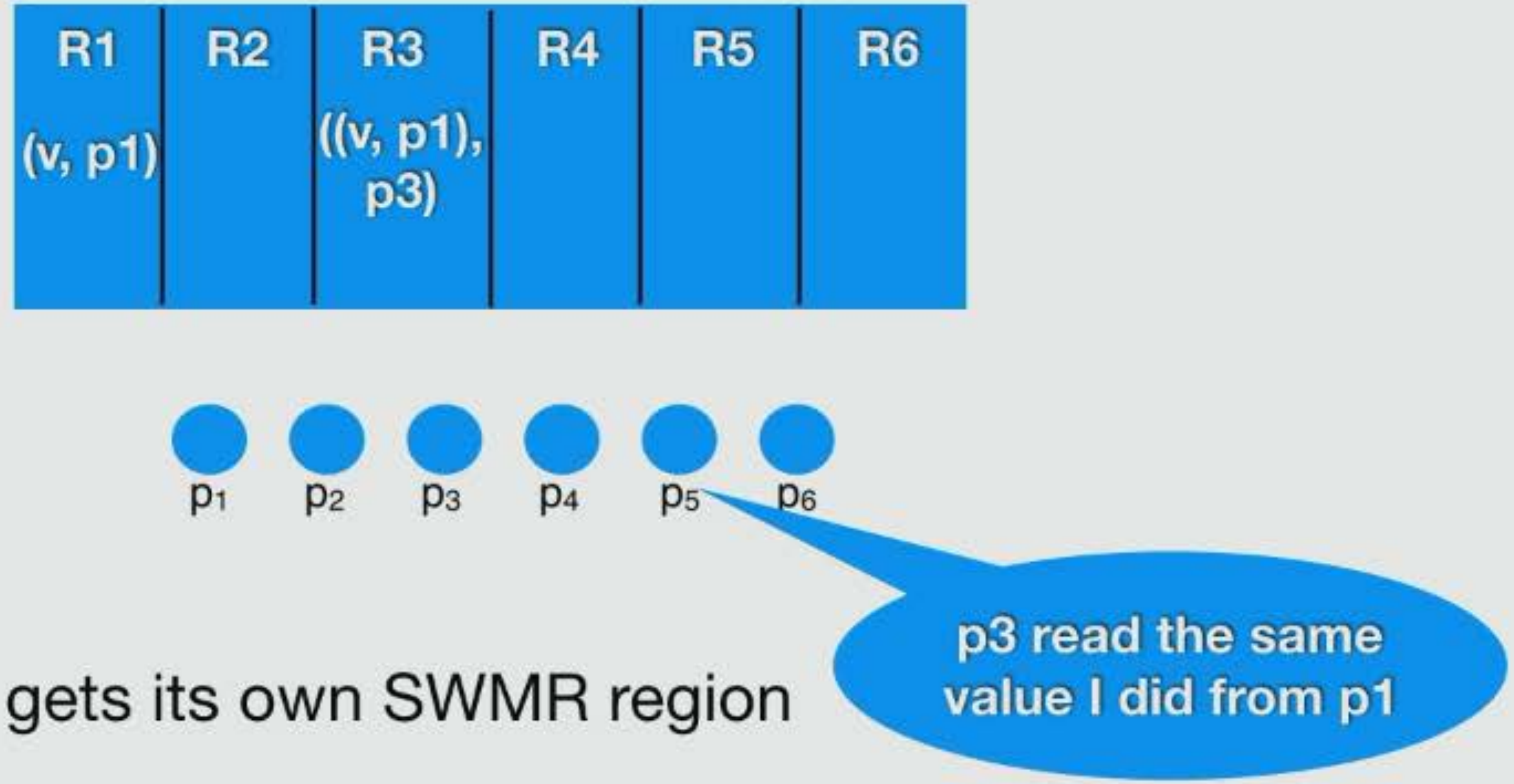
Protocol: Sign and copy over everything that you see

Can now verify that others read the same value

[ABGMZ'19]

Preventing Equivocation

Single Writer Multi Reader region per process



Each process gets its own SWMR region

Protocol: Sign and copy over everything that you see

Can now verify that others read the same value

RDMA vs Previous Results

n = num processes f = num failures		Shared Memory	Message Passing	RDMA Full [ABGMZ'19]	RDMA Scale [ABCGPT'18]
Fault Tolerance	Crash	n>f	n>2f	n>f	n>f+x (xϵ[0,f])
	Byzantine	N/A	n>3f	n>2f	-
Complexity* (Best Case Round Trips)		2	1	1	-
Scalability (processes in network)		10-100	10,000 - 100,000	10-100	10-100,000

RDMA vs Previous Results

n = num processes f = num failures		Shared Memory	Message Passing	RDMA Full* [ABGMZ'19]	RDMA Scale [ABCGPT'18]
Fault Tolerance	Crash	n>f	n>2f	n>f	n>f+x (xϵ[0,f])
	Byzantine	N/A	n>3f	n>2f	-
Complexity* (Best Case Round Trips)		2	1	1	-
Scalability (processes in network)		10-100	10,000 - 100,000	10-100	10-100,000

*With up to half of the memories crashing

Fault Tolerance

Performance

Scalability

Is RDMA fundamentally better than other communication mechanisms?

Fault Tolerance

Performance

Scalability

Is RDMA fundamentally better than other communication mechanisms?

Yes!

RDMA gives us the power of shared memory without compromising performance

Fault Tolerance

Performance

Scalability

Is RDMA fundamentally better than other communication mechanisms?

Yes!

RDMA gives us the power of shared memory without compromising performance

Plus better Byzantine algorithms

Fault Tolerance

✓ Performance

Scalability

Is RDMA fundamentally better than other communication mechanisms?

Yes!

RDMA gives us the power of shared memory without compromising performance

Plus better Byzantine algorithms

✔ Fault Tolerance

✔ Performance

Scalability

Is RDMA fundamentally better than other communication mechanisms?

Yes!

RDMA gives us the power of shared memory without compromising performance

Plus better Byzantine algorithms

✔ Performance

✔ Fault Tolerance

✘ Scalability

Is RDMA fundamentally better than other communication mechanisms?

Yes!

RDMA gives us the power of shared memory without compromising performance

Plus better Byzantine algorithms

Can we scale better and still retain some of RDMA's advantages?

Scalability

What prevented our algorithms from scaling?

Scalability

What prevented our algorithms from scaling?

Many open connections

Scalability

What prevented our algorithms from scaling?

Many open connections

↓

NIC experiences frequent cache misses

Scalability

What prevented our algorithms from scaling?

Many open connections

↓

NIC experiences frequent cache misses

↓

Scalability

What prevented our algorithms from scaling?

Many open connections

↓

NIC experiences frequent cache misses

↓

Slower communication

Scalability

What prevented our algorithms from scaling?

Many open connections

↓

NIC experiences frequent cache misses

↓

Slower communication

Solution: don't open all connections.

Scalability

What prevented our algorithms from scaling?

Many open connections

↓

NIC experiences frequent cache misses

↓

Slower communication

Solution: don't open all connections.

Scalability

What prevented our algorithms from scaling?

Many open connections *per machine*

NIC experiences frequent cache misses

Slower communication

Solution: don't open all connections.

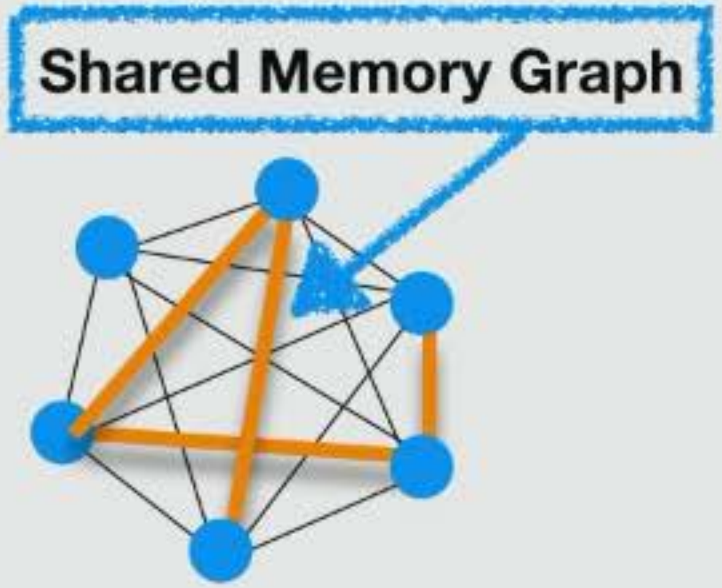
Scalability

What prevented our algorithms from scaling?

Many open connections ***per machine***

NIC experiences frequent cache misses

Slower communication



Solution: don’t open all connections.

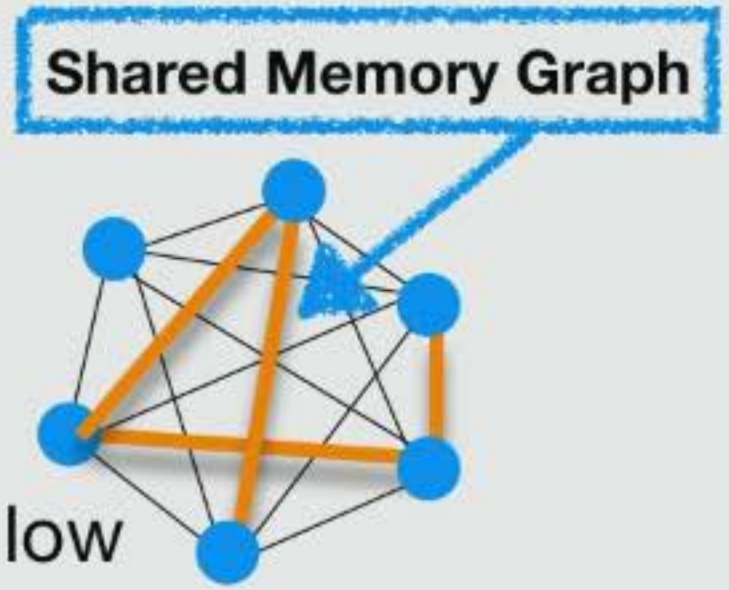
Scalability

What prevented our algorithms from scaling?

Many open connections ***per machine***

NIC experiences frequent cache misses

Slower communication

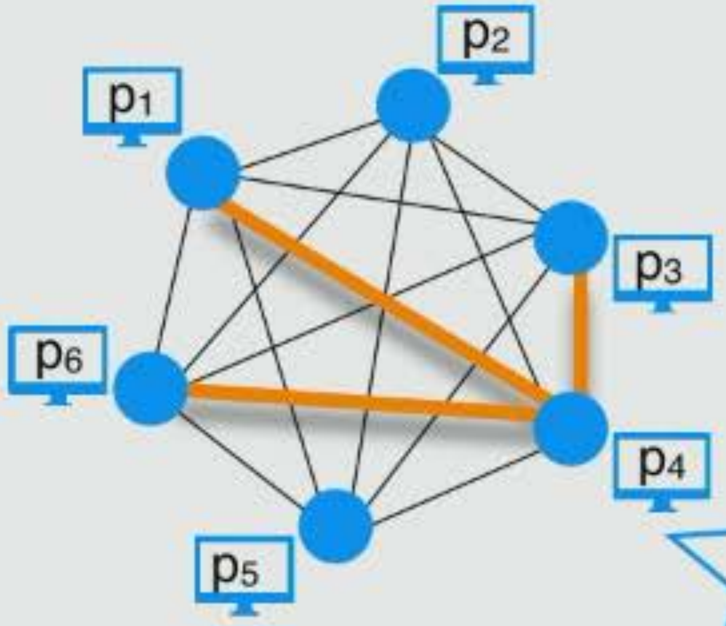


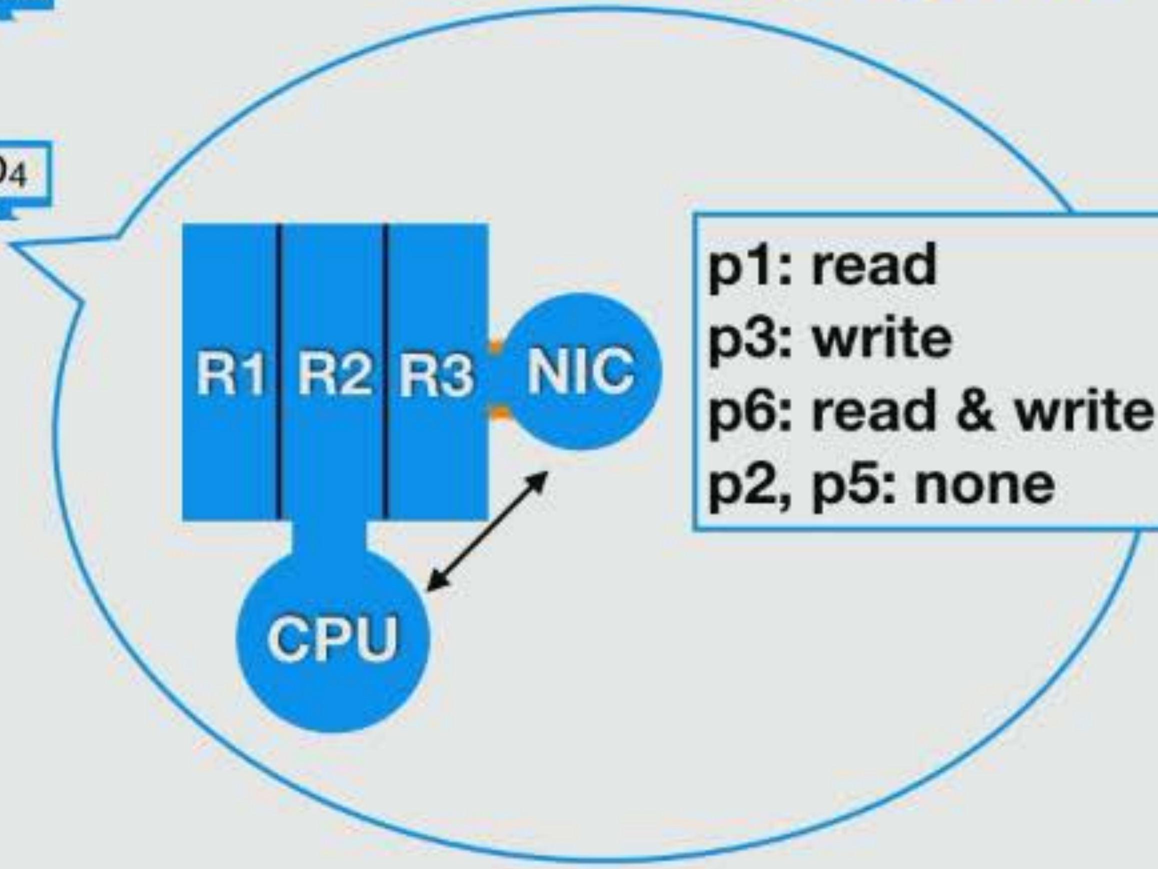
Solution: don't open all connections.

Goal: Keep **degree** of shared memory graph low

Simplifying the Model

- Can choose RDMA **connections** and **permissions**
- Can give different permissions for different **memory regions**

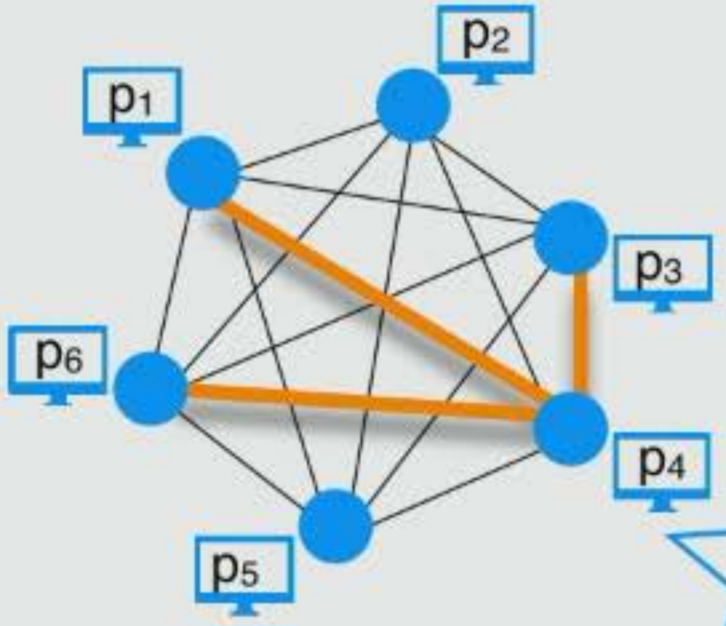


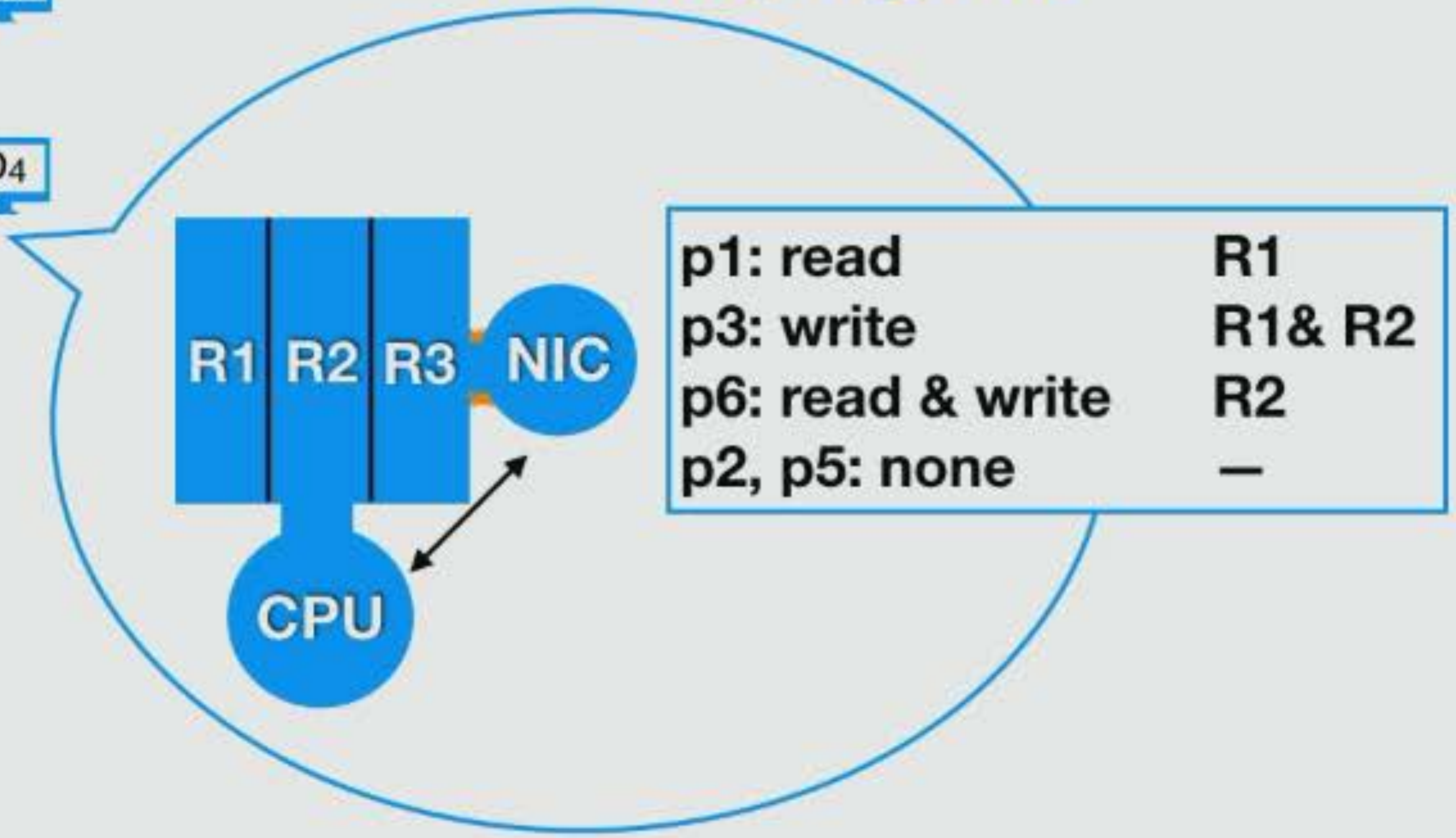


p1: read	**R1**
p3: write	**R1& R2**
p6: read & write	**R2**
p2, p5: none	**—**

Simplifying the Model

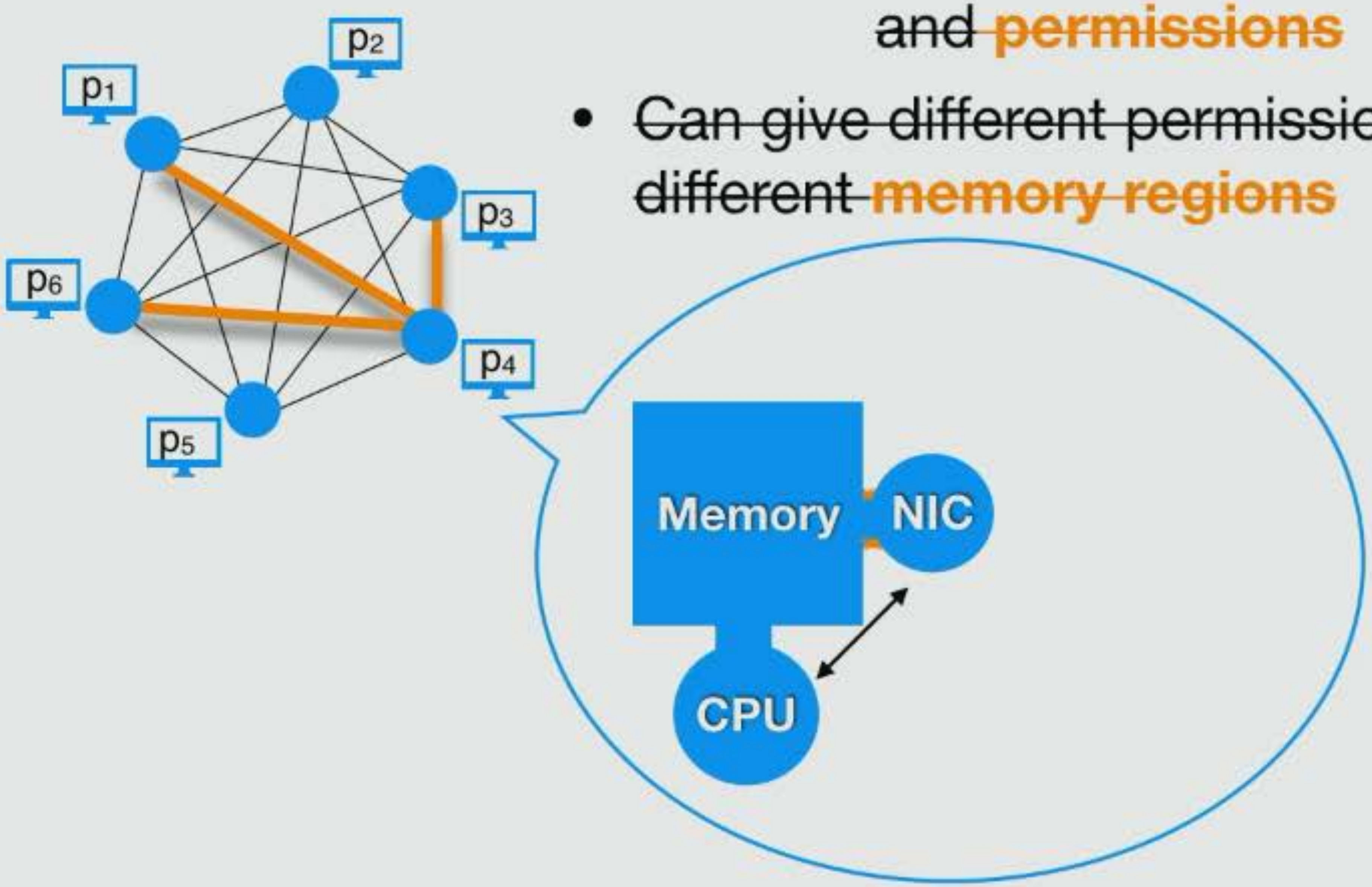
- Can choose RDMA **connections** ~~and **permissions**~~
- ~~Can give different permissions for different **memory regions**~~





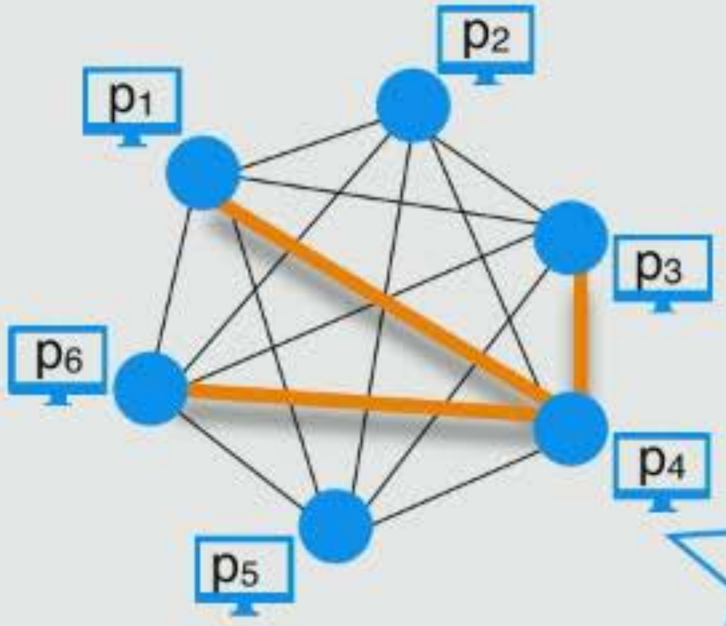
Simplifying the Model

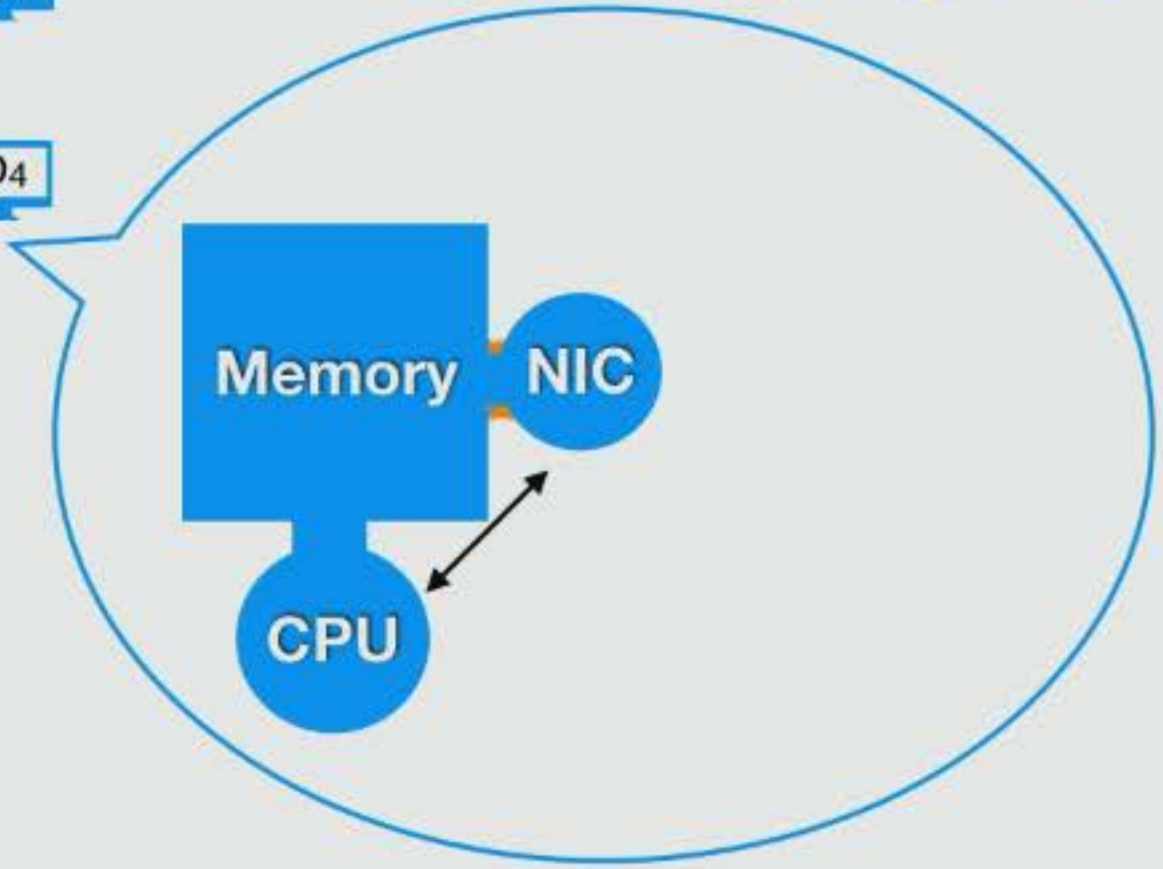
- Can choose RDMA **connections** ~~and **permissions**~~
- ~~Can give different permissions for different **memory regions**~~



Simplifying the Model

- Can choose RDMA **connections** ~~and **permissions**~~
- ~~Can give different permissions for different **memory regions**~~

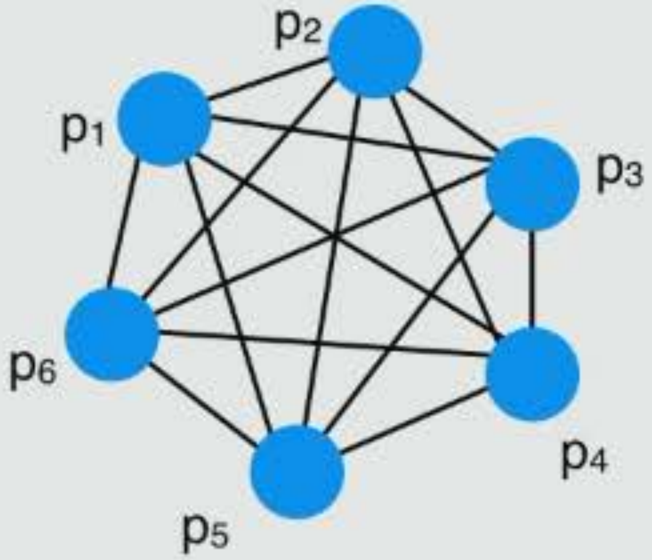




No Byzantine failures
No memory failures

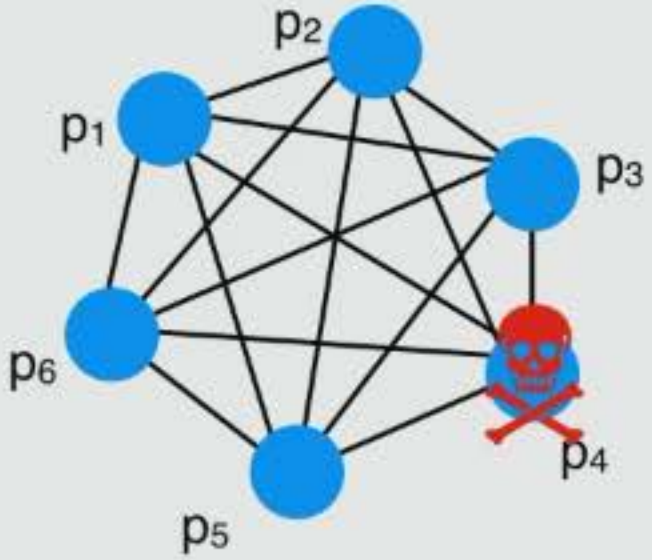
The M&M model

- ***Asynchronous*** network of *n* processes with up to *f* ***crash failures***



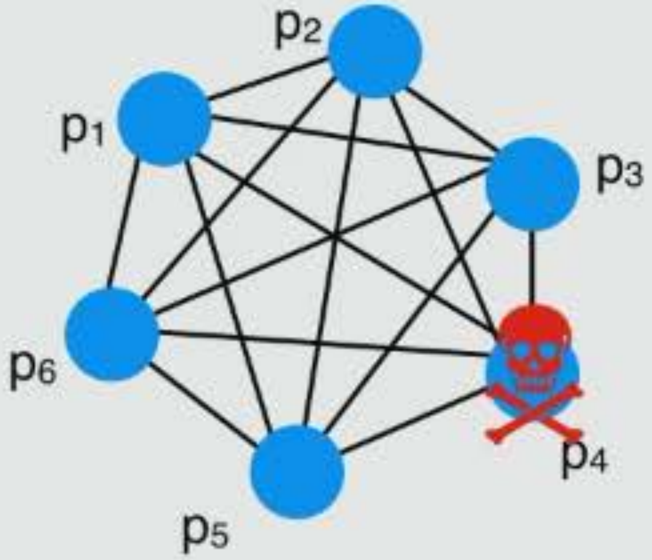
The M&M model

- ***Asynchronous*** network of *n* processes with up to *f* ***crash failures***



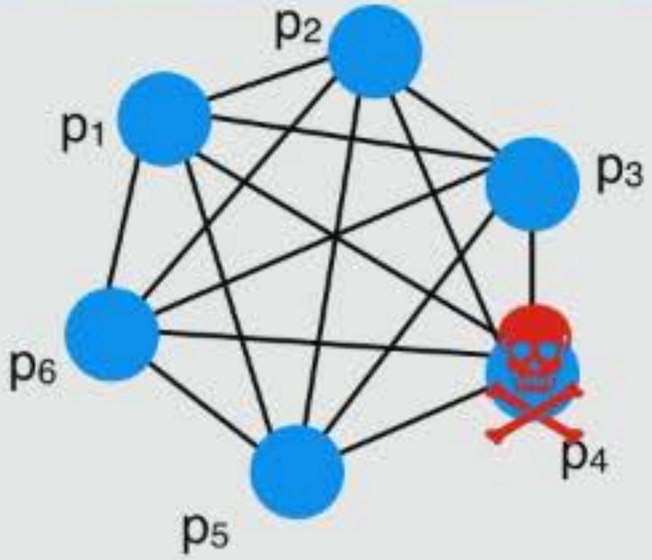
The M&M model

- ***Asynchronous*** network of *n* processes with up to *f* ***crash failures***
- Fully-connected message passing network: ***nodes=procs, edges=links***



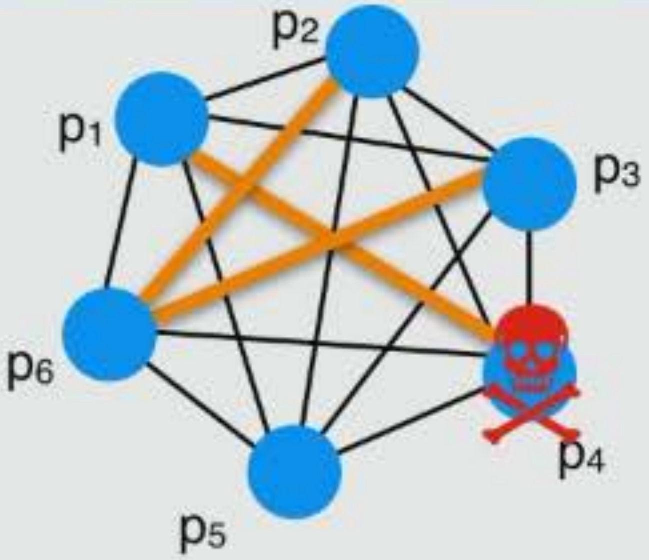
The M&M model

- ***Asynchronous*** network of *n* processes with up to *f* ***crash failures***
- Fully-connected message passing network: ***nodes=procs, edges=links***
- ***Each node owns a piece of memory***



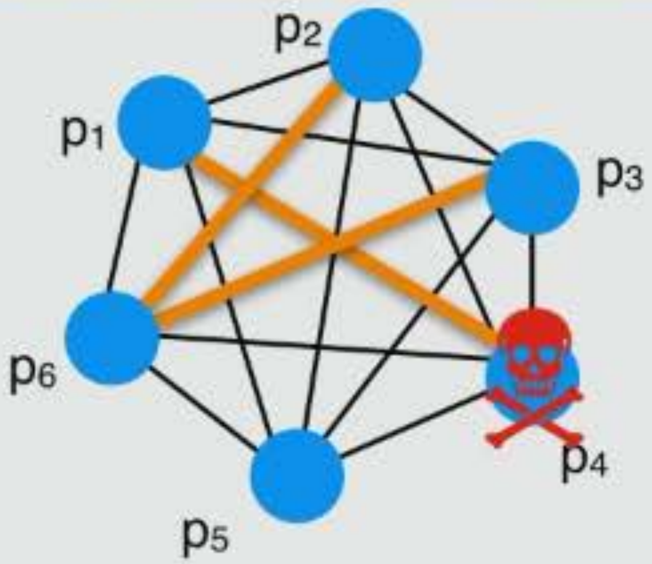
The M&M model

- ***Asynchronous*** network of *n* processes with up to *f* ***crash failures***
- Fully-connected message passing network: ***nodes=procs, edges=links***
- ***Each node owns a piece of memory***
- **Shared memory graph,** ***$G_{SM} = (V, E)$***



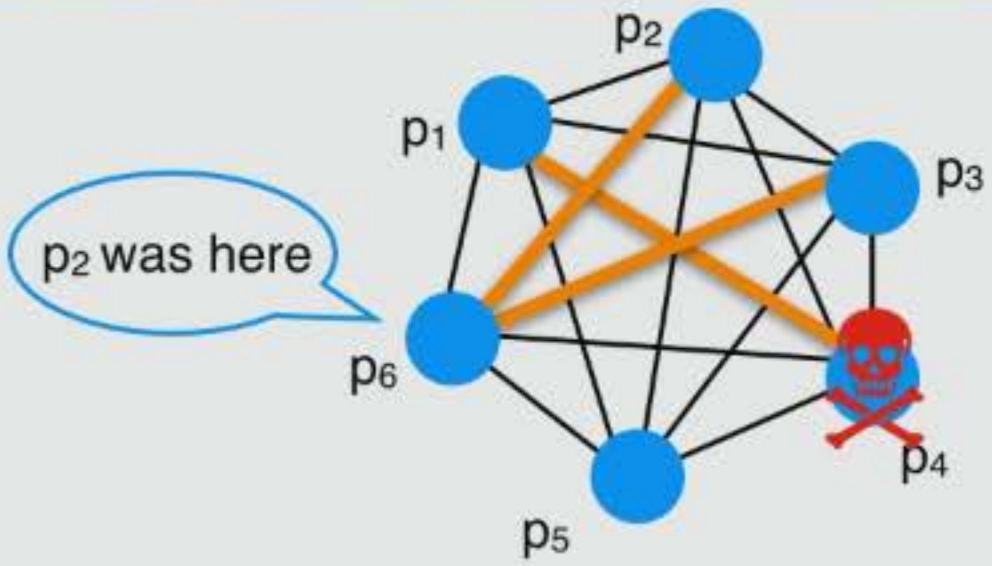
The M&M model

- ***Asynchronous*** network of *n* processes with up to *f* ***crash failures***
- Fully-connected message passing network: ***nodes=procs, edges=links***
- ***Each node owns a piece of memory***
- **Shared memory graph, $G_{SM} = (V, E)$**
- **Nodes *u* and *v* can access each other's memory iff $(u,v) \in E$**



The M&M model

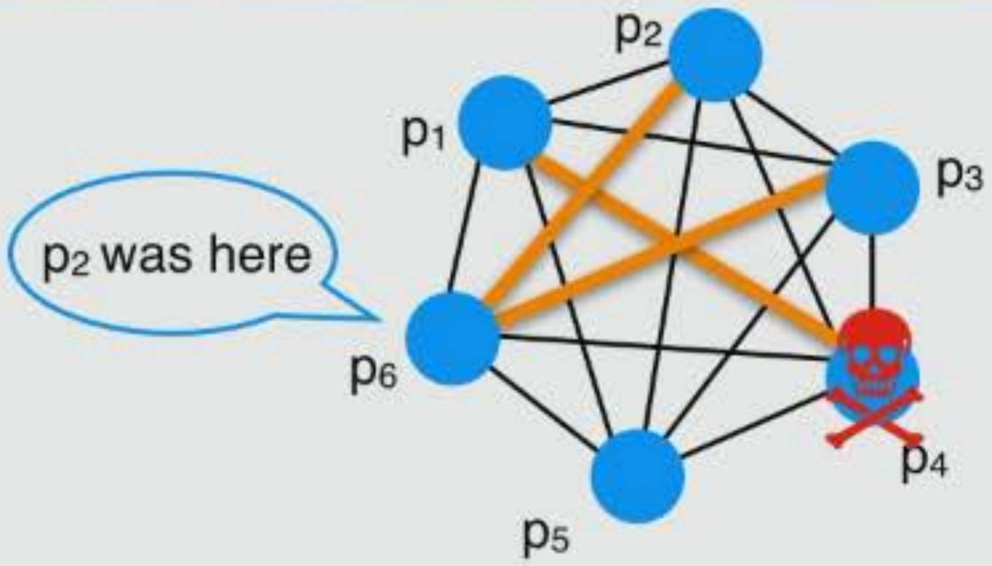
- ***Asynchronous*** network of *n* processes with up to *f* ***crash failures***
- Fully-connected message passing network: ***nodes=procs, edges=links***
- ***Each node owns a piece of memory***
- **Shared memory graph, *G_{SM} = (V, E)***
- **Nodes *u* and *v* can access each other's memory iff *(u,v)* ϵ *E***



The M&M model

- ***Asynchronous*** network of *n* processes with up to *f* ***crash failures***
- Fully-connected message passing network: ***nodes=procs, edges=links***

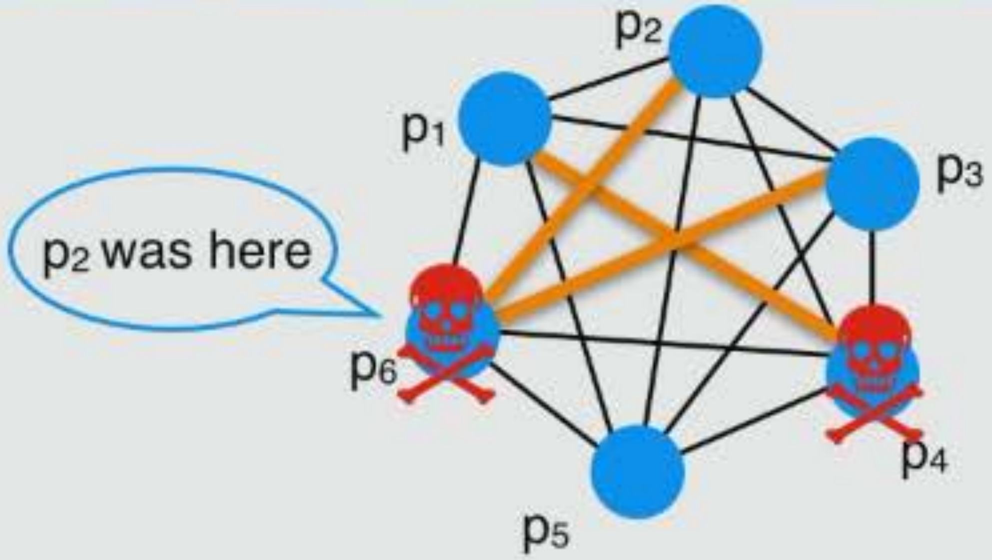
- ***Each node owns a piece of memory***
- **Shared memory graph, *G_{SM} = (V, E)***
- **Nodes *u* and *v* can access each other's memory iff *(u,v) ϵ E***
- **Processes may crash, but their *memory remains accessible***



The M&M model

- ***Asynchronous*** network of *n* processes with up to *f* ***crash failures***
- Fully-connected message passing network: ***nodes=procs, edges=links***

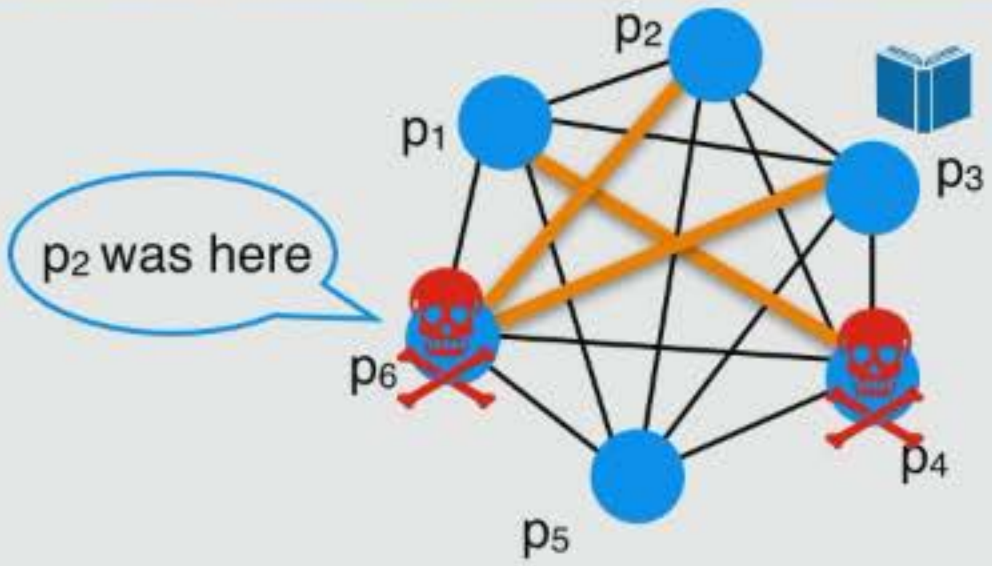
- ***Each node owns a piece of memory***
- **Shared memory graph, $G_{SM} = (V, E)$**
- **Nodes *u* and *v* can access each other's memory iff *(u,v)* ϵ *E***
- **Processes may crash, but their *memory remains accessible***



The M&M model

- ***Asynchronous*** network of *n* processes with up to *f* ***crash failures***
- Fully-connected message passing network: ***nodes=procs, edges=links***

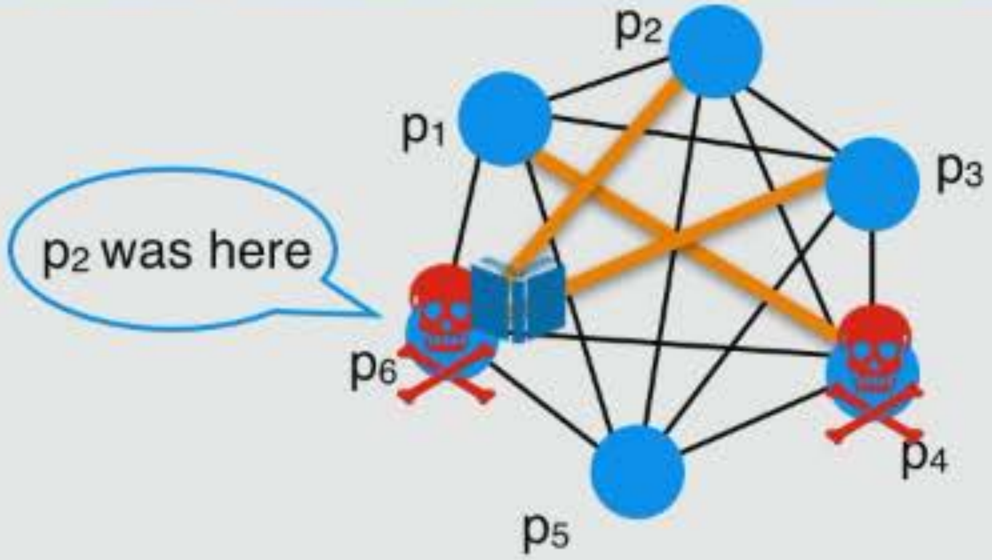
- ***Each node owns a piece of memory***
- **Shared memory graph, *G_{SM} = (V, E)***
- **Nodes *u* and *v* can access each other's memory iff *(u,v) ϵ E***
- **Processes may crash, but their *memory remains accessible***



The M&M model

- ***Asynchronous*** network of *n* processes with up to *f* ***crash failures***
- Fully-connected message passing network: ***nodes=procs, edges=links***

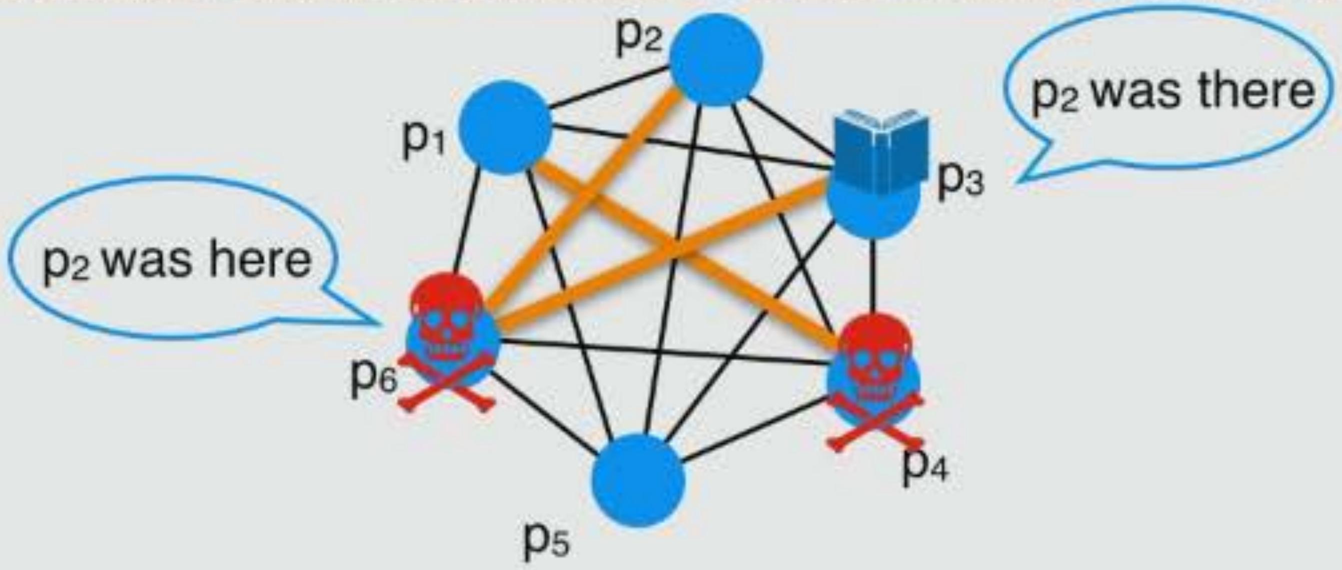
- ***Each node owns a piece of memory***
- **Shared memory graph,** $G_{SM} = (V, E)$
- **Nodes *u* and *v* can access each other's memory iff** $(u,v) \in E$
- **Processes may crash, but their *memory remains accessible***



The M&M model

- ***Asynchronous*** network of *n* processes with up to *f* ***crash failures***
- Fully-connected message passing network: ***nodes=procs, edges=links***

- ***Each node owns a piece of memory***
- **Shared memory graph, *G_{SM} = (V, E)***
- **Nodes *u* and *v* can access each other's memory iff *(u,v) ϵ E***
- **Processes may crash, but their *memory remains accessible***



RDMA vs Previous Results

n = num processes f = num failures		Shared Memory	Message Passing	RDMA Full* [ABGMZ'19]	RDMA Scale [ABCGPT'18]
Fault Tolerance	Crash	n>f	n>2f	n>f	n>f+x (xϵ[0,f])
	Byzantine	N/A	n>3f	n>2f	-
Complexity* (Best Case Round Trips)		2	1	1	-
Scalability (processes in network)		10-100	10,000 - 100,000	10-100	10-100,000

[ABGMZ'19]

M&M Consensus

"Pretend" more processes are alive by sending their messages too

[ABGMZ'19]

M&M Consensus

"Pretend" more processes are alive by sending their messages too

Simulate a ***message passing algorithm*** by replacing messages with ***list of messages*** representing your ***shared memory neighbors***

[ABGMZ'19] M&M Consensus

"Pretend" more processes are alive by sending their messages too

n>2f: Tolerates n/2 failures

Simulate a ***message passing algorithm*** by replacing messages with ***list of messages*** representing your ***shared memory neighbors***

[ABGMZ'19]

M&M Consensus

"Pretend" more processes are alive by sending their messages too

n>2f: Tolerates n/2 failures

n>f: SM consensus needs 1 process

Simulate a ***message passing algorithm*** by replacing messages with ***list of messages*** representing your ***shared memory neighbors***

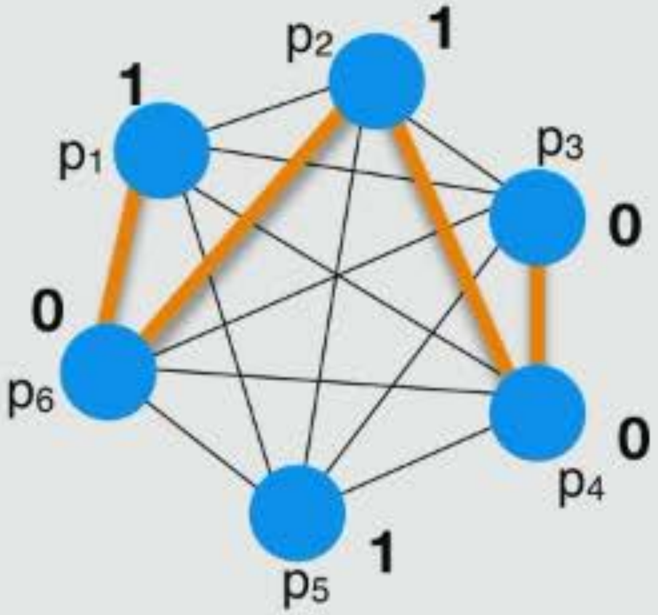
[ABGMZ'19] M&M Consensus

"Pretend" more processes are alive by sending their messages too

n>2f: Tolerates n/2 failures

n>f: SM consensus needs 1 process

Simulate a ***message passing algorithm*** by replacing messages with ***list of messages*** representing your ***shared memory neighbors***



[A**B**GMZ'19]

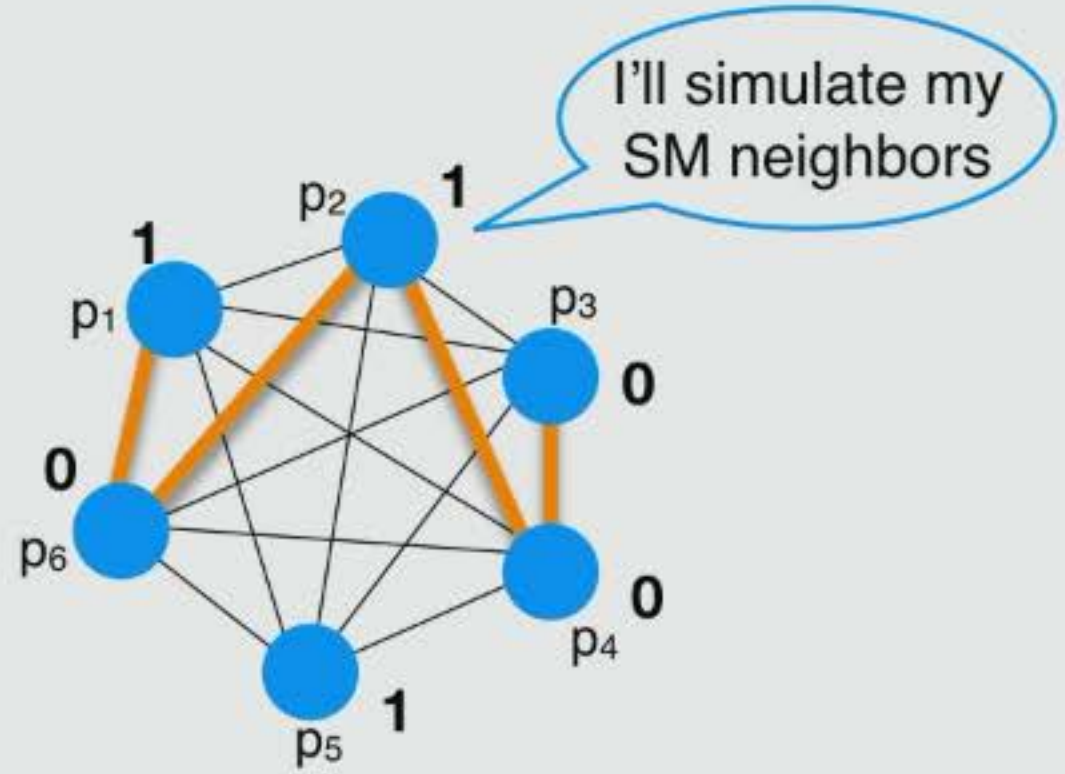
M&M Consensus

"Pretend" more processes are alive by sending their messages too

n>2f: Tolerates n/2 failures

n>f: SM consensus needs 1 process

Simulate a ***message passing algorithm*** by replacing messages with ***list of messages*** representing your ***shared memory neighbors***



[ABGMZ'19]

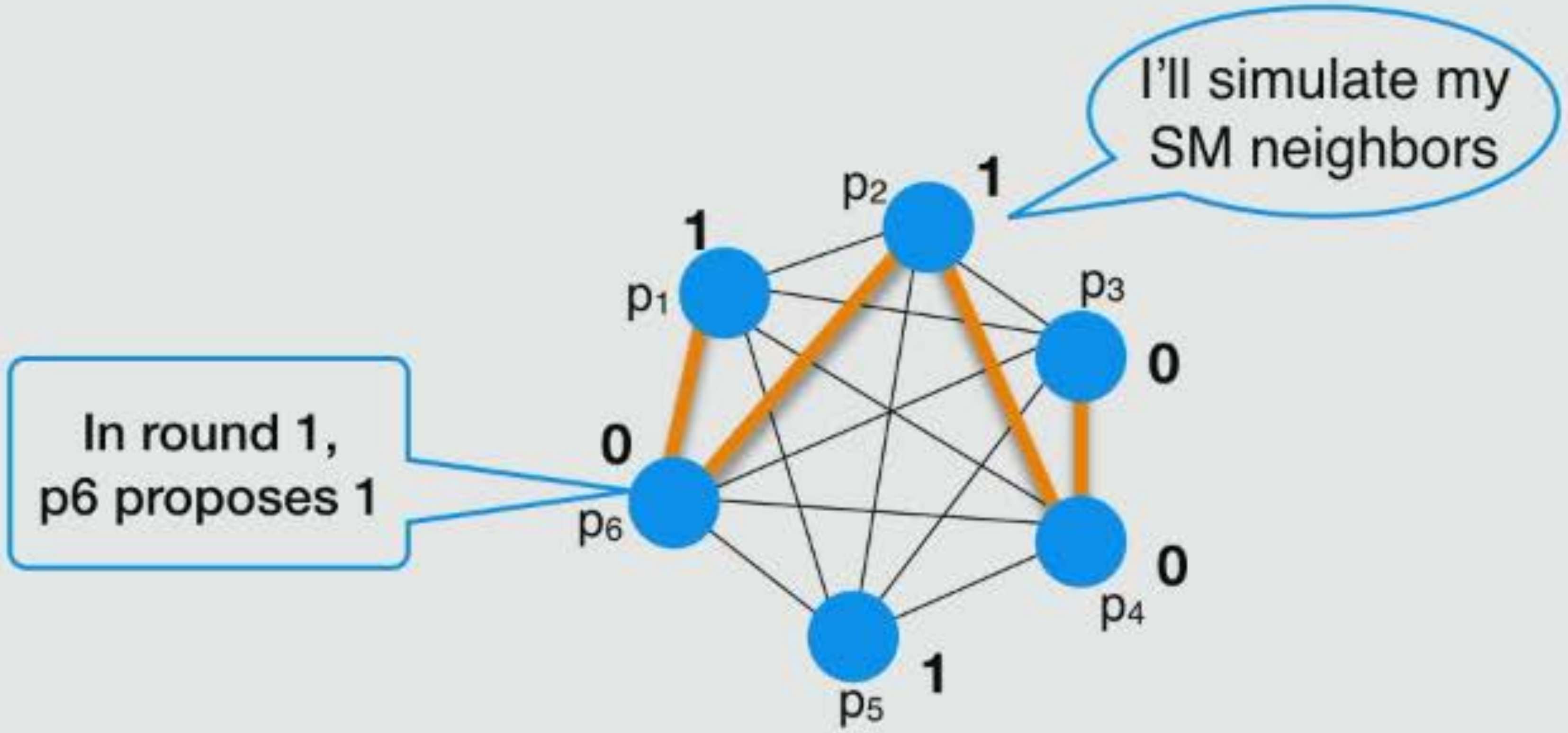
M&M Consensus

"Pretend" more processes are alive by sending their messages too

n>2f: Tolerates n/2 failures

n>f: SM consensus needs 1 process

Simulate a ***message passing algorithm*** by replacing messages with ***list of messages*** representing your ***shared memory neighbors***



[ABGMZ'19]

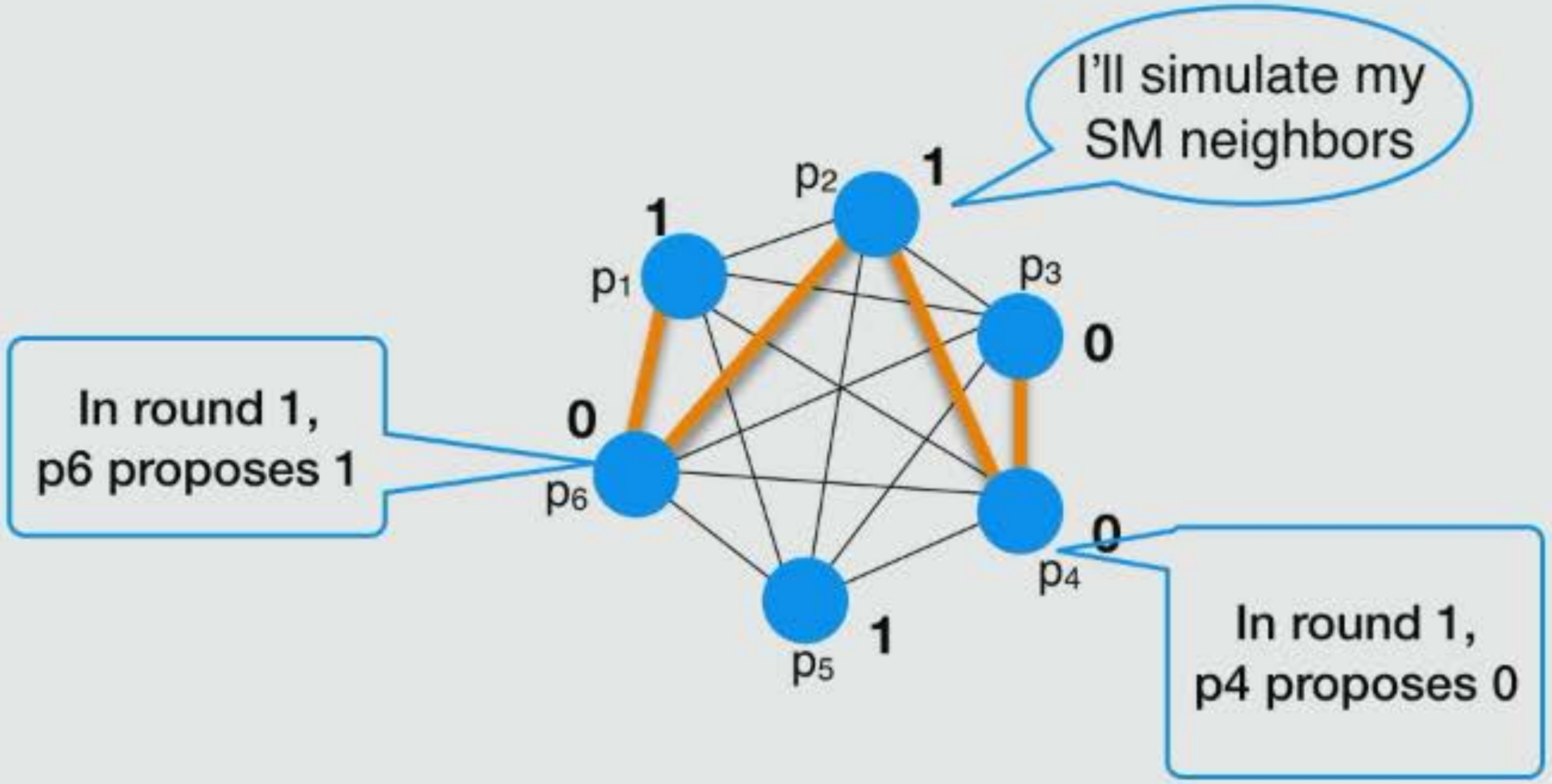
M&M Consensus

"Pretend" more processes are alive by sending their messages too

n>2f: Tolerates n/2 failures

n>f: SM consensus needs 1 process

Simulate a ***message passing algorithm*** by replacing messages with ***list of messages*** representing your ***shared memory neighbors***



M&M Consensus

[ABGMZ'19]

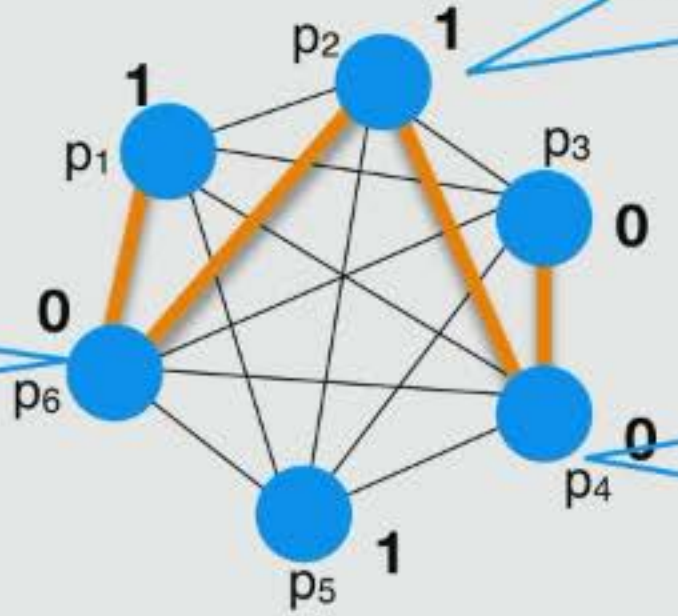
"Pretend" more processes are alive by sending their messages too

n>2f: Tolerates n/2 failures

n>f: SM consensus needs 1 process

Simulate a ***message passing algorithm*** by replacing messages with ***list of messages*** representing your ***shared memory neighbors***

Message:
[(p2, 1), (p6, 1), (p4, 0)]



In round 1,
p6 proposes 1

In round 1,
p4 proposes 0

[ABGMZ'19]

M&M Consensus

"Pretend" more processes are alive by simulating neighbors

[ABGMZ'19]

M&M Consensus

"Pretend" more processes are alive by simulating neighbors

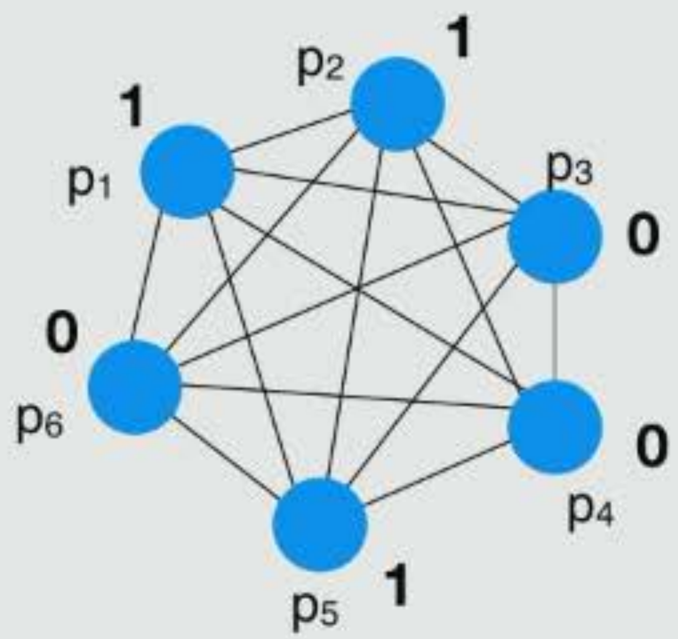
Original Algorithm

[ABGMZ'19]

M&M Consensus

"Pretend" more processes are alive by simulating neighbors

Original Algorithm

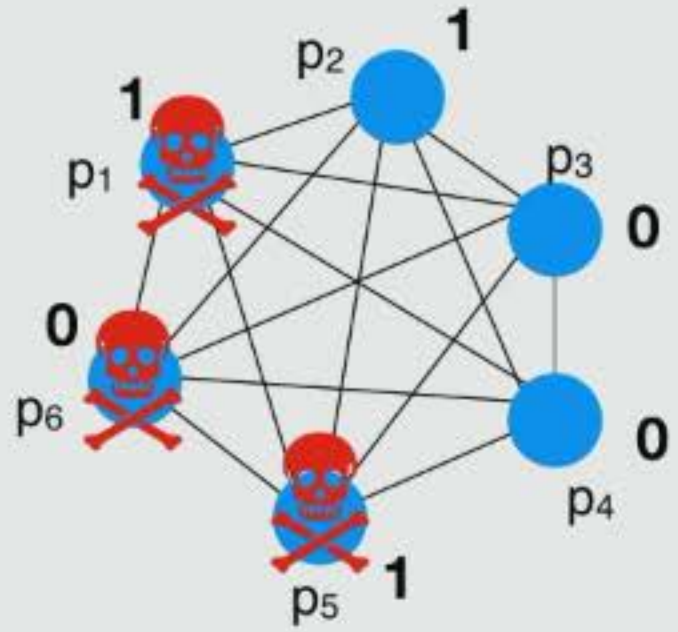


[ABGMZ'19]

M&M Consensus

"Pretend" more processes are alive by simulating neighbors

Original Algorithm

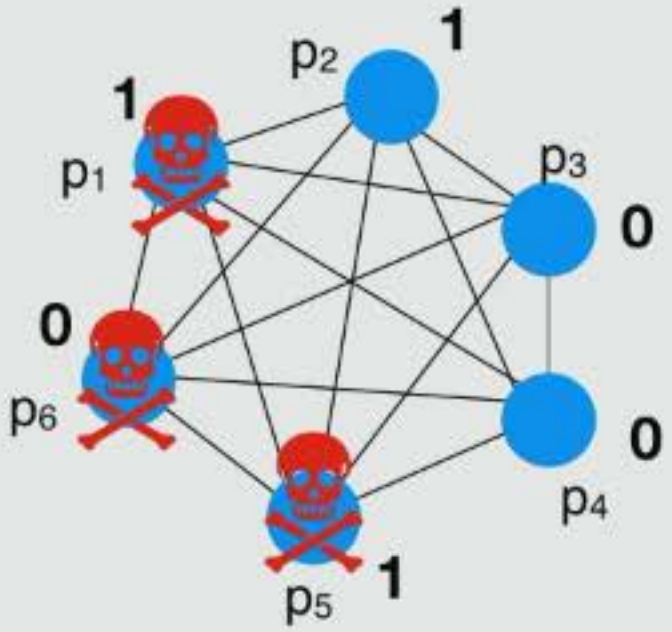


[ABGMZ'19]

M&M Consensus

"Pretend" more processes are alive by simulating neighbors

Original Algorithm



Message passing can only tolerate n>2f

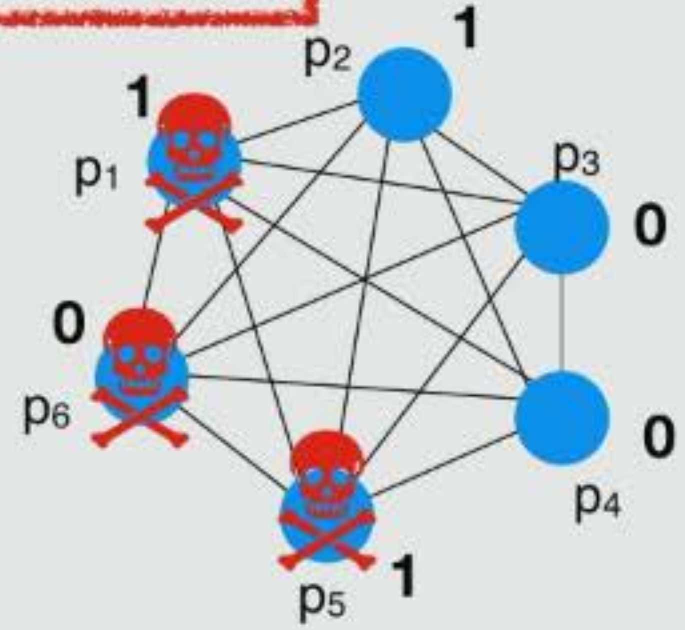
[ABGMZ'19]

M&M Consensus

"Pretend" more processes are alive by simulating neighbors

Original Algorithm

Algorithm is doomed



Message passing can only tolerate n>2f

[ABGMZ'19]

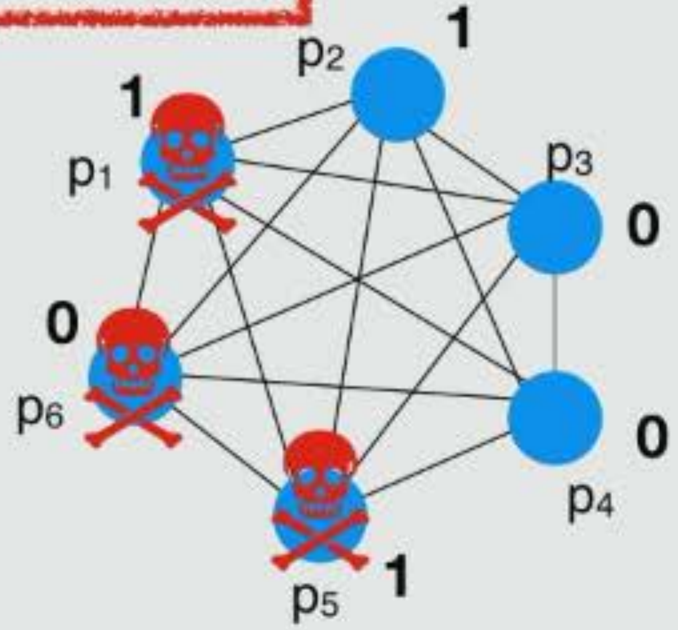
M&M Consensus

"Pretend" more processes are alive by simulating neighbors

Original Algorithm

M&M Algorithm

Algorithm is doomed



Message passing can only tolerate n>2f

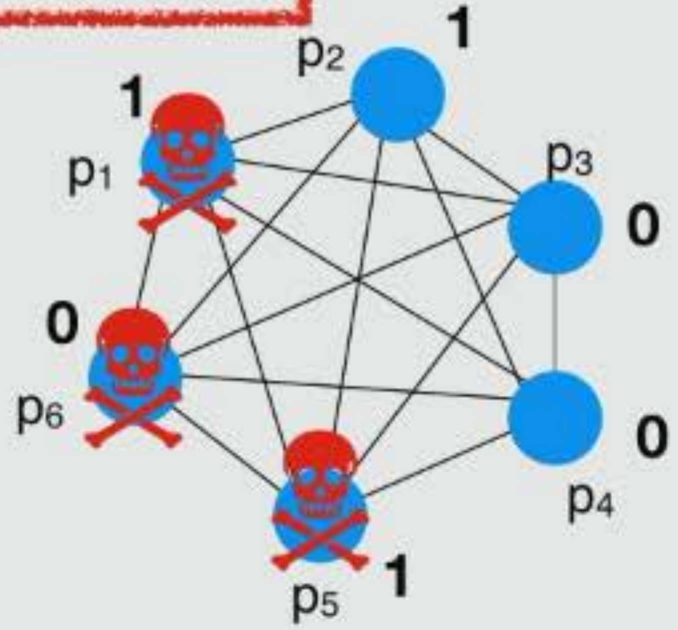
[ABGMZ'19]

M&M Consensus

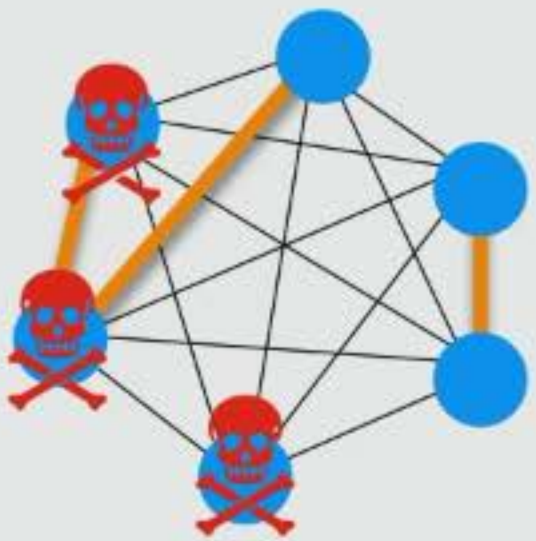
"Pretend" more processes are alive by simulating neighbors

Original Algorithm

Algorithm is doomed



M&M Algorithm

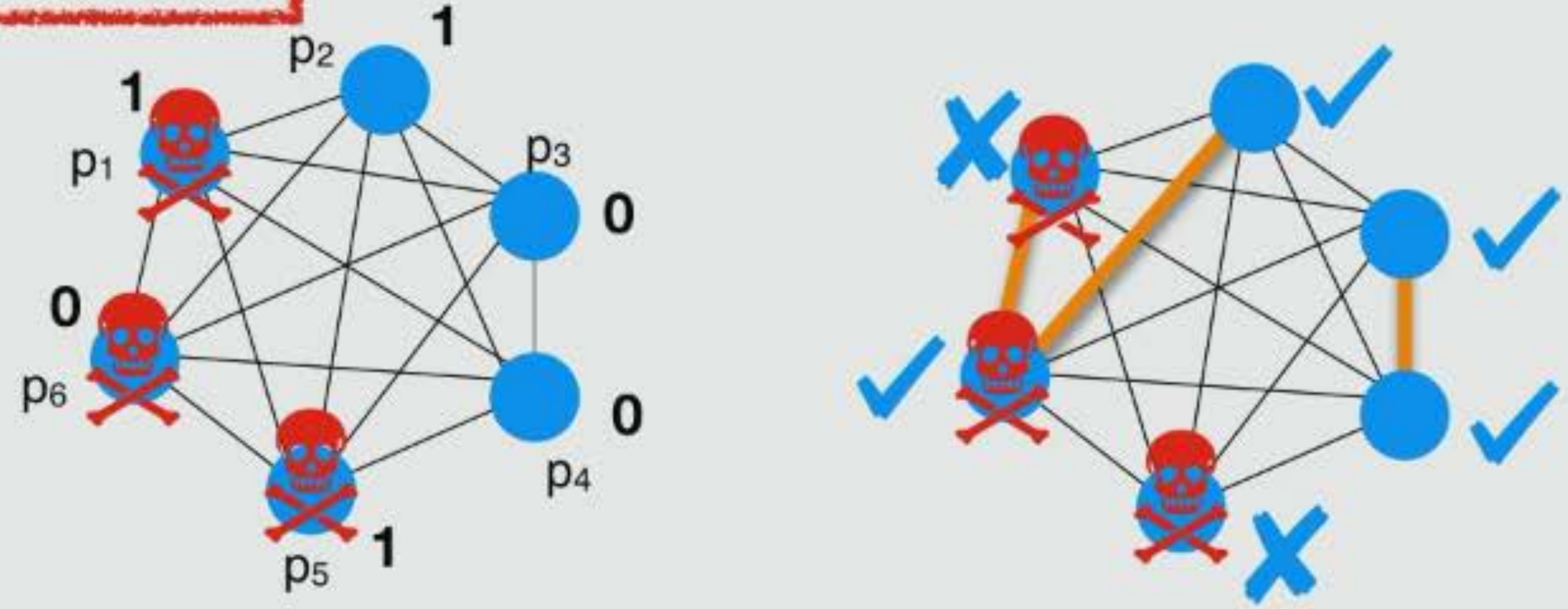


Message passing can only tolerate n>2f

[ABGMZ'19]

M&M Consensus

"Pretend" more processes are alive by simulating neighbors



Message passing can only tolerate n>2f

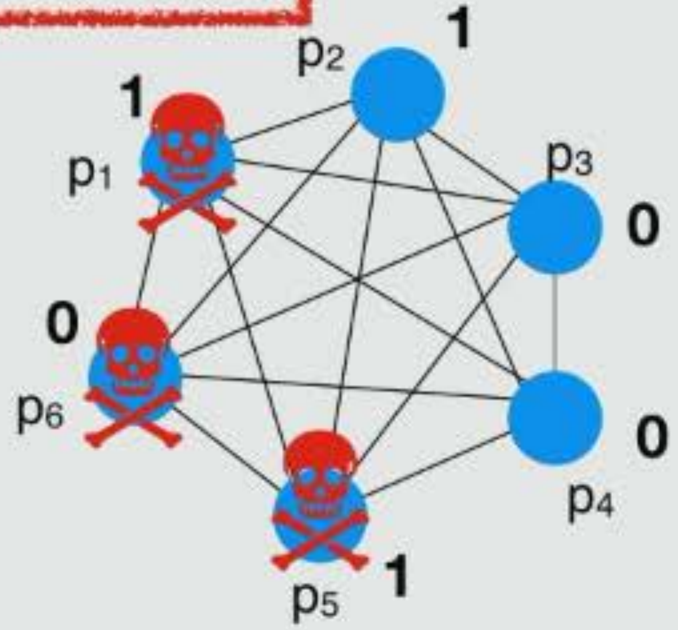
[ABGMZ'19]

M&M Consensus

"Pretend" more processes are alive by simulating neighbors

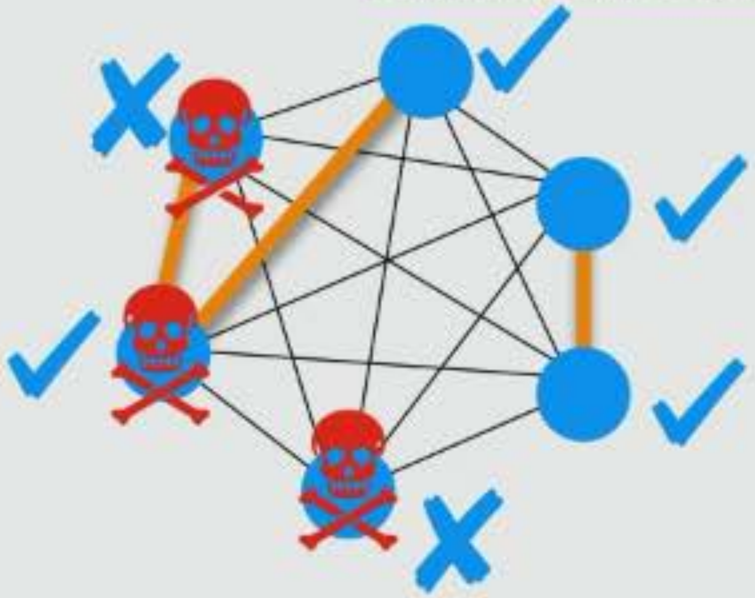
Original Algorithm

Algorithm is doomed



M&M Algorithm

More than half -> Success!



Message passing can only tolerate n>2f

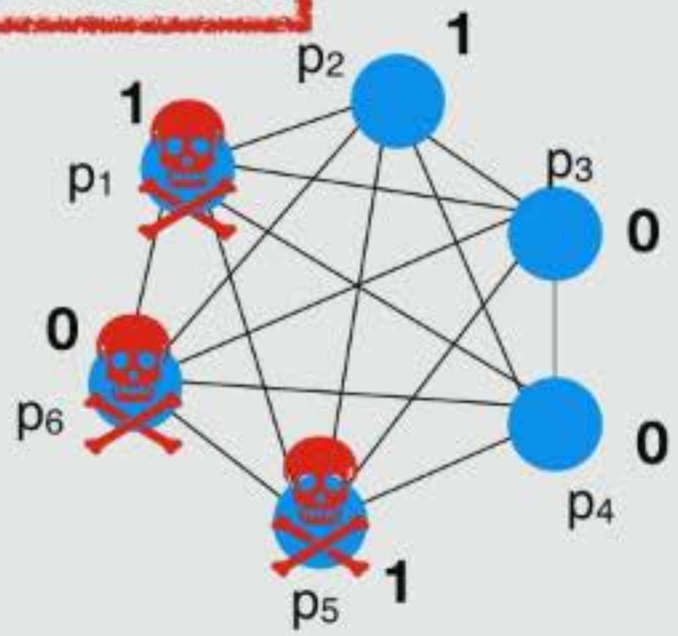
[ABGMZ'19]

M&M Consensus

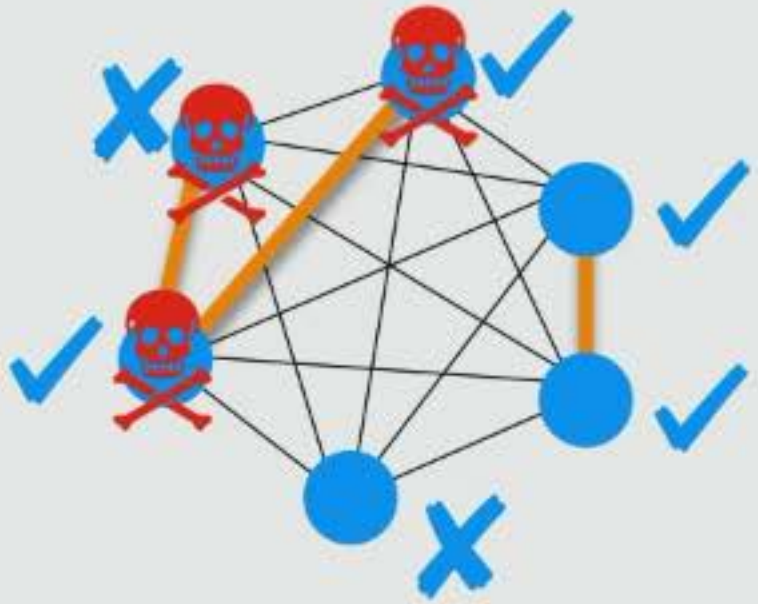
"Pretend" more processes are alive by simulating neighbors

Original Algorithm

Algorithm is doomed



M&M Algorithm



Message passing can only tolerate n>2f

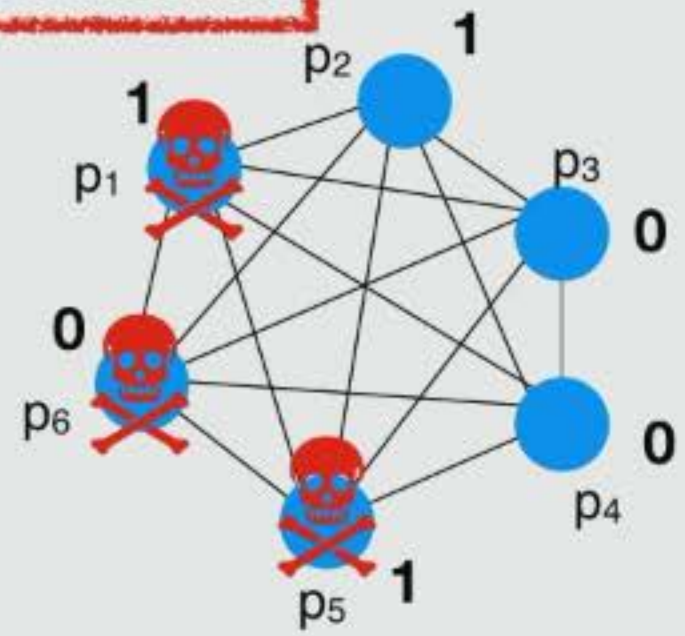
[ABGMZ'19]

M&M Consensus

"Pretend" more processes are alive by simulating neighbors

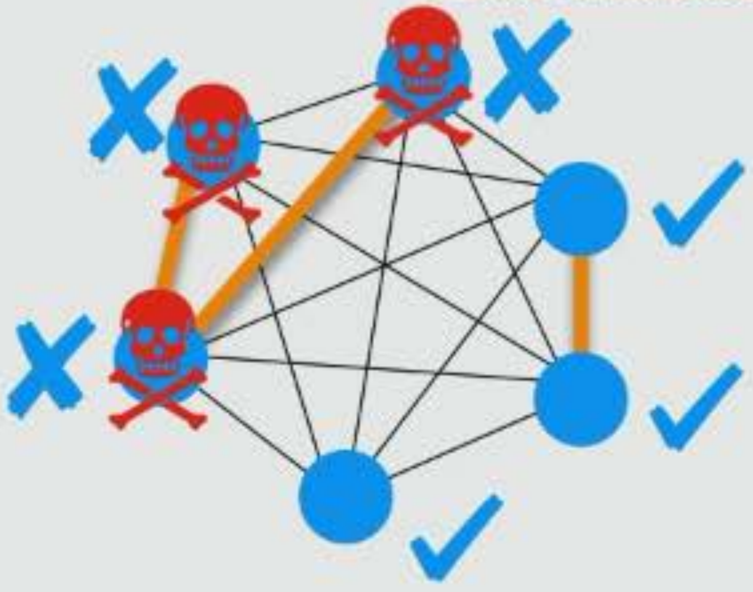
Original Algorithm

Algorithm is doomed



M&M Algorithm

Exactly half -> Failure!



Message passing can only tolerate n>2f

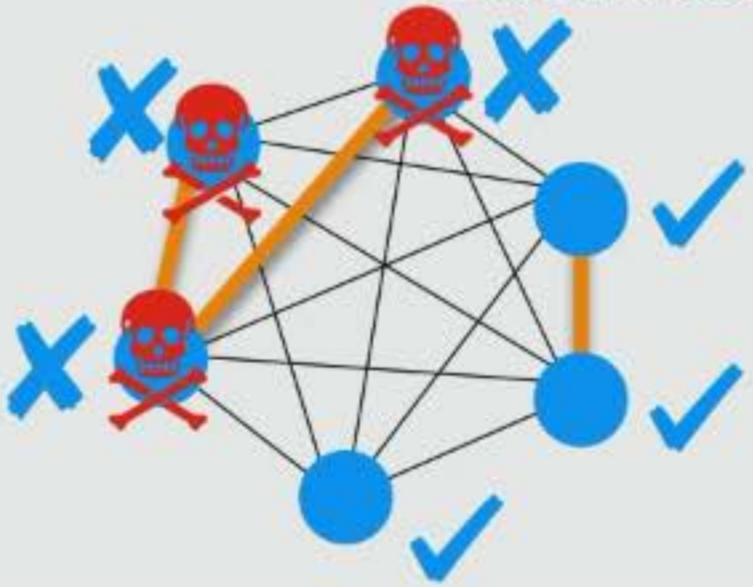
[ABGMZ'19]

M&M Consensus

"Pretend" more processes are alive by simulating neighbors

M&M Algorithm

Exactly half -> Failure!



We care about the number of neighbors of correct processes

Message passing can only tolerate n>2f

Expander Graphs

Fault tolerance depends on shared memory graph:

Number of neighbors of correct processes

Expander Graphs

Fault tolerance depends on shared memory graph:

Number of neighbors of correct processes

Expander graphs to the rescue!

Expander Graphs

Fault tolerance depends on shared memory graph:

Number of neighbors of correct processes

Expander graphs to the rescue!

"G has high expansion"

↔

"Every subset of the vertices has many neighbors"

Expander Graphs

Fault tolerance depends on shared memory graph:

Number of neighbors of correct processes

Expander graphs to the rescue!

"G has high expansion"

↔

"Every subset of the vertices has many neighbors"

$$h(G)=\min_{S \text{ s.t. } |S|\leq|V|/2} |\delta S|/|S|$$

Expander Graphs

Fault tolerance depends on shared memory graph:

Number of neighbors of correct processes

Expander graphs to the rescue!

"G has high expansion"

$\leftrightarrow$

"Every subset of the vertices has many neighbors"

Neighbors of set S

$$h(G)=\min_{S \text{ s.t. } |S|\leq|V|/2} |\delta S|/|S|$$

Subset S of vertices

Expander Graphs

Fault tolerance depends on shared memory graph:

Number of neighbors of correct processes

Expander graphs to the rescue!

"G has high expansion"

↔

"Every subset of the vertices has many neighbors"

Neighbors of set S

$$h(G) = \min_{S \text{ s.t. } |S| \leq |V|/2} |\delta S| / |S|$$

S with worst ratio defines graph's expansion

Subset S of vertices

[ABGMZ'19] Putting it Together

- Think of ***set of live processes*** as S
- Adversary will pick S to be the set with the least expansion

Graph with high expansion can tolerate more failures

[A**B**GMZ'19]

Putting it Together

- Think of ***set of live processes*** as S
- Adversary will pick S to be the set with the least expansion

Graph with high expansion can tolerate more failures

Theorem [Aguilera**B**CalciuGuerraouiPetrankToueg'18]:
If shared memory graph has vertex expansion ratio *h*,

then we can tolerate $f < \left(1 - \frac{1}{2 \cdot h}\right) \cdot n$ failures

[ABGMZ'19]

Putting it Together

- Think of ***set of live processes*** as S
- Adversary will pick S to be the set with the least expansion

Graph with high expansion can tolerate more failures

Theorem [AguileraBCalciuGuerraouiPetrankToueg'18]:
If shared memory graph has vertex expansion ratio *h*,

then we can tolerate $f < \left(1 - \frac{1}{2 \cdot h}\right) \cdot n$ failures

Also show impossibility result:
relation to expansion is ***inherent***.

Outline

✔ **RDMA details**

- Setting 1: **RDMA's full power** (complete graph)
  - ✔ **Crash-only** algorithm: ***n>f*** tolerant, 1 round-trip
  - ✔ **Byzantine** algorithm: ***n>2f*** tolerant, 1 round-trip

- Setting 2: **Scalability: Using RDMA sparingly** (incomplete graph)
  - ✔ Crash-only Algorithm: tolerance vs topology

[A**B**GMZ'19]

Putting it Together

- Think of ***set of live processes*** as S
- Adversary will pick S to be the set with the least expansion

Graph with high expansion can tolerate more failures

[ABGMZ'19]

Putting it Together

- Think of ***set of live processes*** as S
- Adversary will pick S to be the set with the least expansion

Graph with high expansion can tolerate more failures

Theorem [AguileraBCalciuGuerraouiPetrankToueg'18]:
If shared memory graph has vertex expansion ratio *h*,

then we can tolerate $f < \left(1 - \frac{1}{2 \cdot h}\right) \cdot n$ failures

Also show impossibility result:
relation to expansion is ***inherent***.

Outline

RDMA details

- Setting 1: **RDMA's full power** (complete graph)
  - **Crash-only** algorithm: ***n>f*** tolerant, 1 round-trip
  - **Byzantine** algorithm: ***n>2f*** tolerant, 1 round-trip

- Setting 2: **Scalability: Using RDMA sparingly** (incomplete graph)
  - Crash-only Algorithm: tolerance vs topology

RDMA vs Previous Results

n = num processes f = num failures		Shared Memory	Message Passing	RDMA Full* [ABGMZ'19]	RDMA Scale [ABCGPT'18]
Fault Tolerance	Crash	n>f	n>2f	n>f	n>f+x (xϵ[0,f])
	Byzantine	N/A	n>3f	n>2f	-
Complexity* (Best Case Round Trips)		2	1	1	-
Scalability (processes in network)		10-100	10,000 - 100,000	10-100	10-100,000

*With up to half of the memories crashing

Can we scale better and still retain some of RDMA's advantages?

Can we scale better and still retain some of RDMA's advantages?

Yes!

We can tolerate more crash failures than message passing

Can we scale better and still retain some of RDMA's advantages?

Yes!
We can tolerate more crash failures than message passing

Can we scale better and still retain some of RDMA's advantages?

Yes!

We can tolerate more crash failures than message passing

Crash Tolerance

Byzantine Tolerance

Can we scale better and still retain some of RDMA's advantages?

Yes!
We can tolerate more crash failures than message passing

Crash Tolerance

Performance

Byzantine Tolerance

Can we scale better and still retain some of RDMA's advantages?

Yes!

We can tolerate more crash failures than message passing

✔ Crash Tolerance

Performance

Byzantine Tolerance

Can we scale better and still retain some of RDMA's advantages?

Yes!

We can tolerate more crash failures than message passing

✔ Crash Tolerance

? Performance

? Byzantine Tolerance

Can we scale better and still retain some of RDMA's advantages?

Yes!

We can tolerate more crash failures than message passing

✓ Fault Tolerance

✓ Performance

✗ Scalability

Is RDMA fundamentally better than other communication mechanisms?

Yes!

RDMA gives us the power of shared memory without compromising performance

Fault Tolerance

Performance

Scalability

Is RDMA fundamentally better than other communication mechanisms?

Yes!

RDMA gives us the power of shared memory without compromising performance

Theory —> Practice

Fault Tolerance

Performance

Scalability

Is RDMA fundamentally better than other communication mechanisms?

Fault Tolerance

Performance

Scalability

Is RDMA fundamentally better than other communication mechanisms?

Consensus

Fault Tolerance

Performance

Scalability

Is RDMA fundamentally better than other communication mechanisms?

Consensus

Can RDMA solve other problems better as well?

Summary

- **Consensus** as a lens to study **RDMA**
  - RDMA improves tradeoff between **fault tolerance** and **performance**
  - RDMA could **scale** to large networks

Future Directions

- Strengthening scalability model
- Implementing these solutions
- Problems beyond consensus

Summary

- **Consensus** as a lens to study **RDMA**
  - RDMA improves tradeoff between **fault tolerance** and **performance**
  - RDMA could **scale** to large networks

Future Directions

- Strengthening scalability model
- Implementing these solutions
- Problems beyond consensus

Thank you!