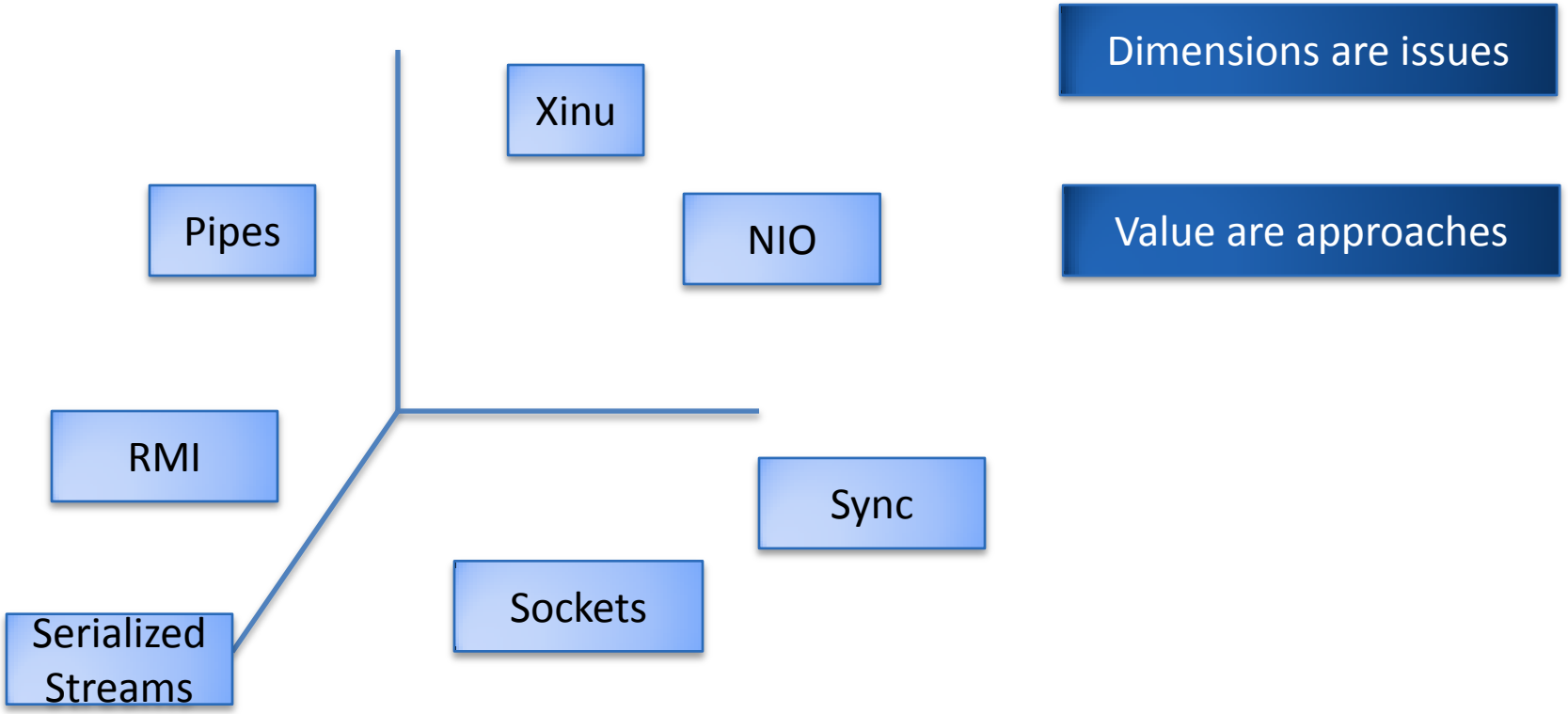
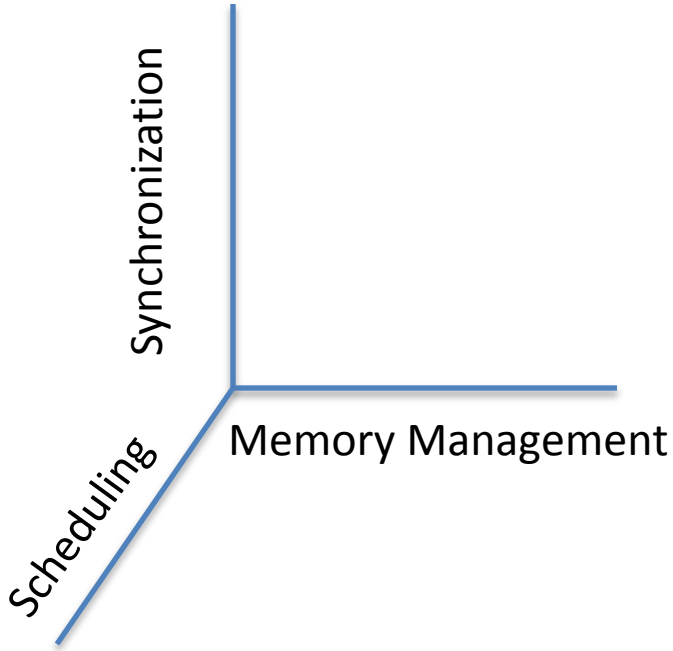
Inter Process and Thread Communication: Design Space

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)

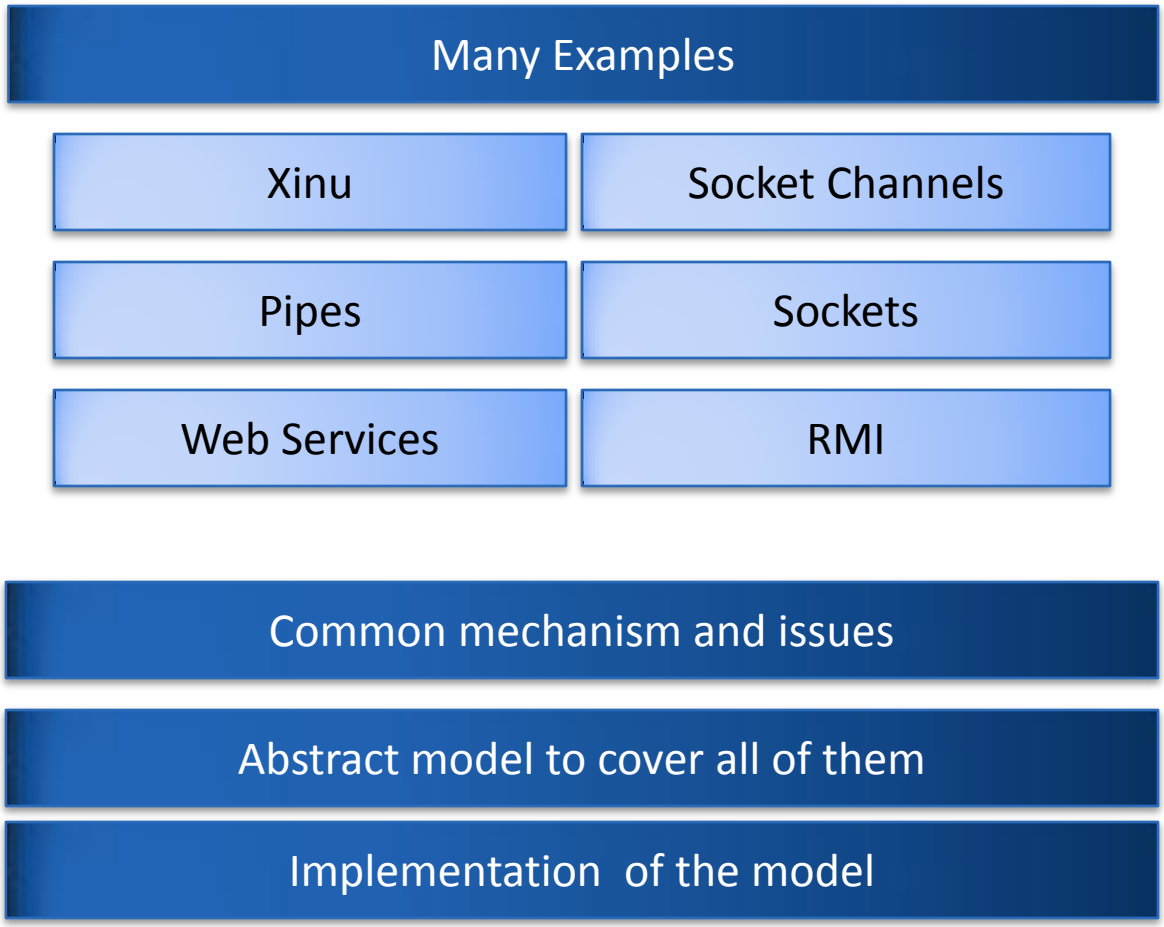
Points vs. Design space



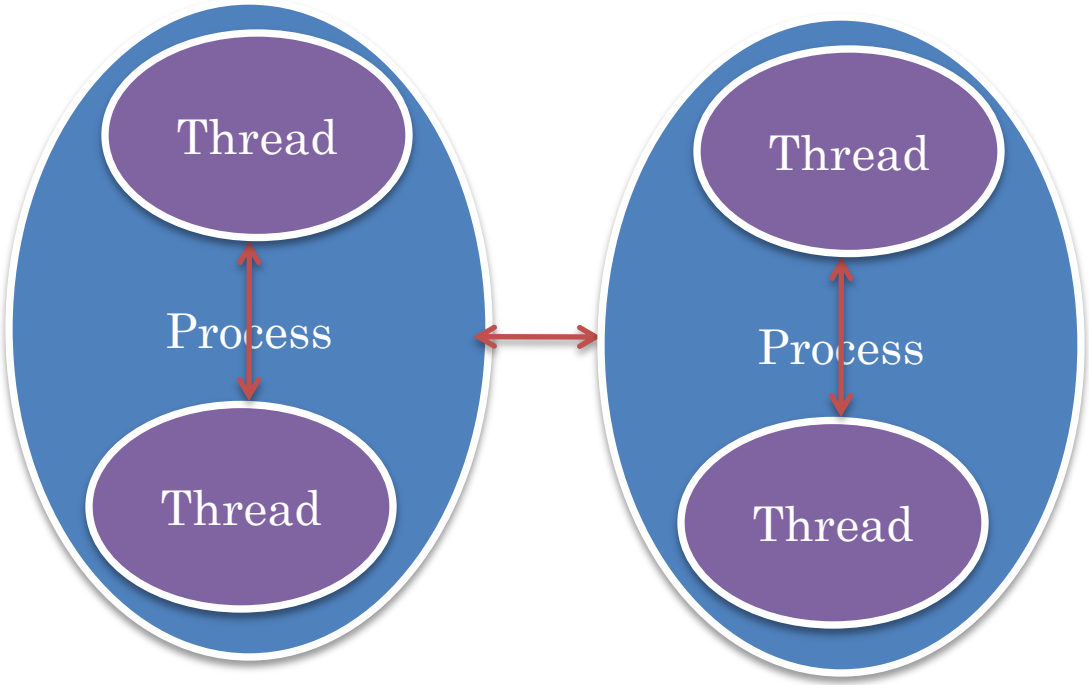
OS Example



Why IPC Design Space?



Scope



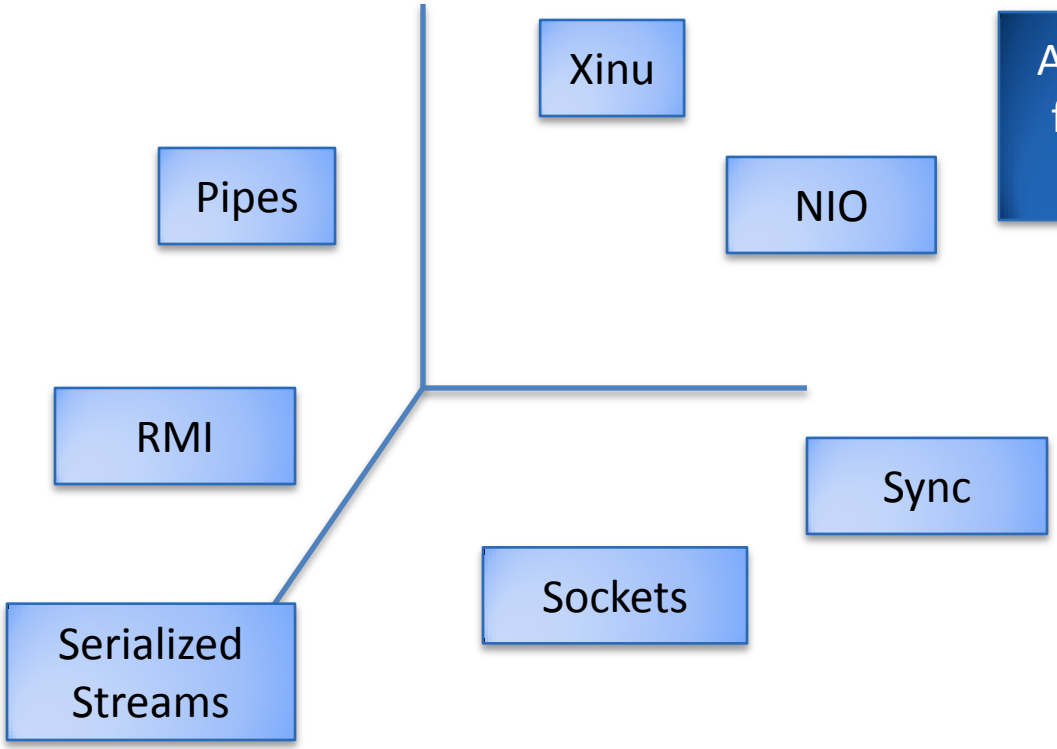
Distributed program

→ Inter-process coupling

→ inter-process information communication

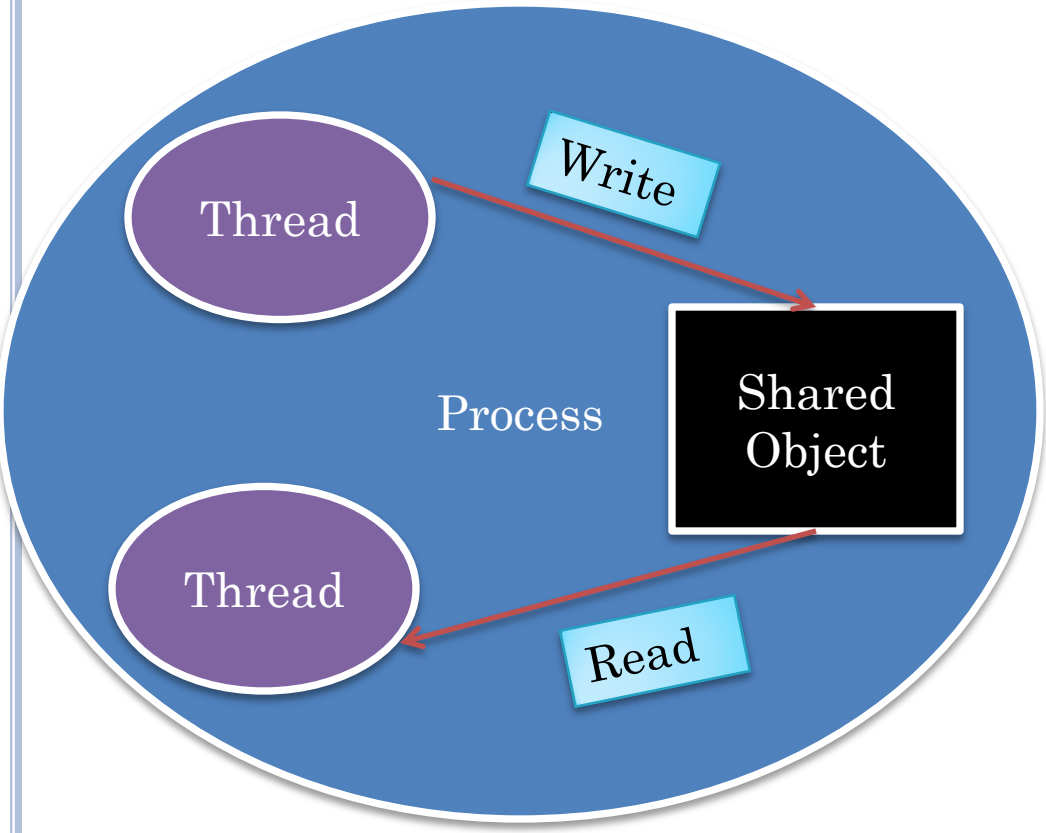
→ Communication among lightweight or heavyweight processes

Unifying Basis?



Are there forms of IPC that are fundamentally different from these?

Intra Process Shared Memory



Write sends info, read receives it

e.g.: Shared object could be result of matrix communication

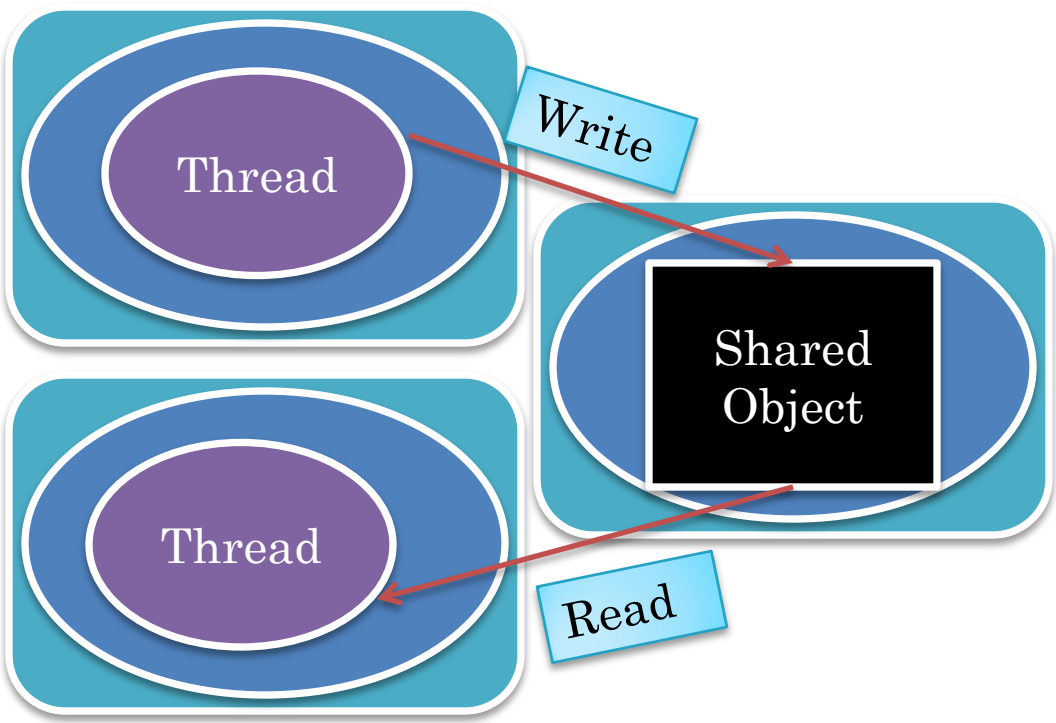
How does information consumer get notified about new information?

Thread synchronization mechanisms (semaphores, conditions, …)

Receiving thread must wait

Threads in different processes?

Inter Process Shared Memory



Shared file/database/memory

If not on same computer, then need process around it

Need some other mechanism to communicate with the process

Shared Memory Pros and Cons

Familiar model

Not sufficient when threads on different computers

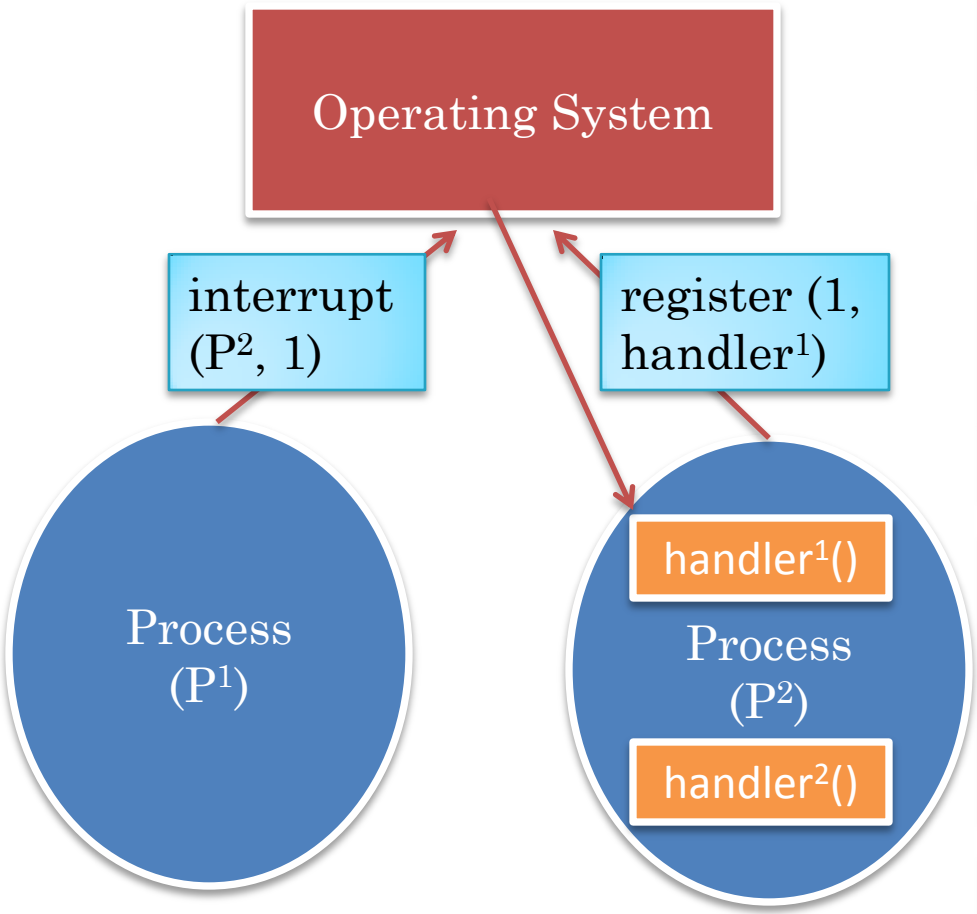
Global variables considered bad

Threads other than the writer and reader can access them

Receiving thread must wait for or poll for information (unless some notification mechanism added)

Inter-process notification method?

Software Interrupts (Signals)



Information receiving process registers software interrupt number and handlers with OS

register (interupt#, handler)

Information sender interrupts (signals) receiving process by naming process and handler#

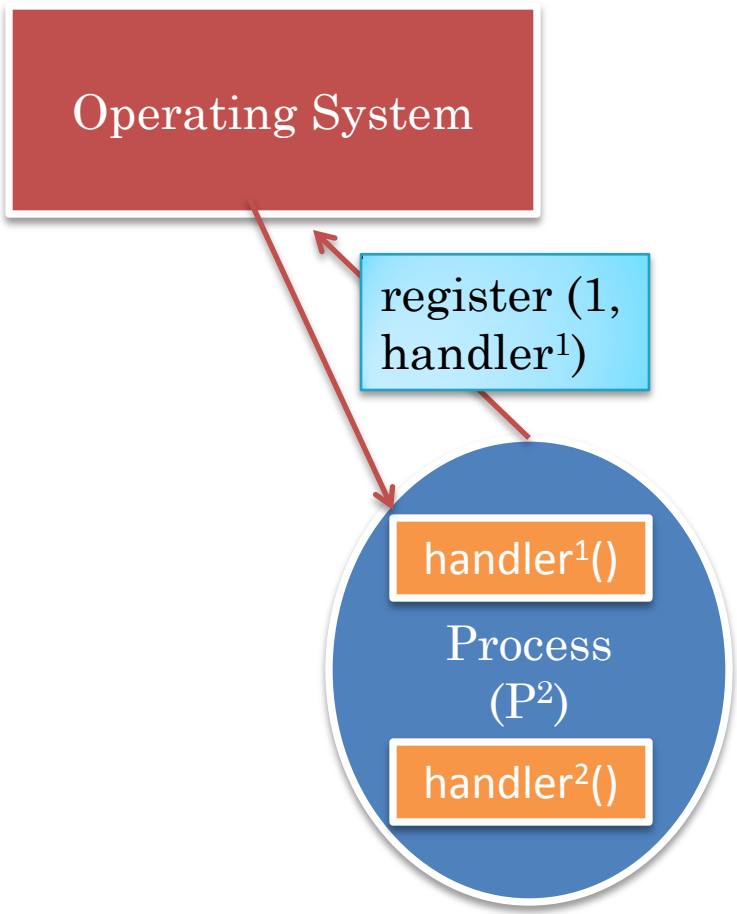
interrupt (process id, interrupt#)

OS calls software interrupt handler in process, interrupting its current activity, just as hardware calls hardware-interrupt handler in CPU.

New thread created?

Uses stack of current thread

OS-Process Communication



User wishes to kill process (CTRL-C)

An alarm set by process goes on

Some limit such as file size or virtual time expired

Software Interrupts Pro and Cons

React to OS Event

Familiar interrupt model for IPC

Communicates only event id, not parameters of the event

Assumes processes share an operating system thus not suitable when processes on different computers

Process ids (usually) make no sense on other computers

Receiver cannot delay processing information (Dual of shared memory problem)

No way of queuing signals

Message Passing

Alternative to Shared Memory and Software Interrupts

Many Examples

- Sockets
- Socket Channels
- Pipes
- HTTP
- Web Services
- RMI

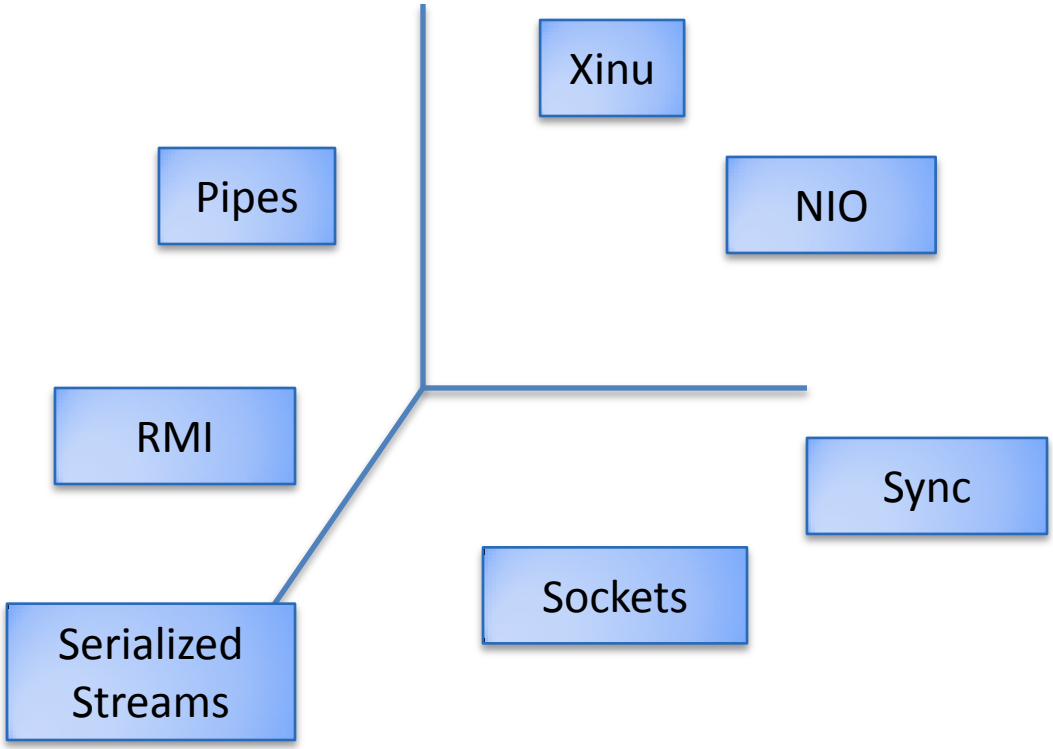
Traditionally: RPC not considered message passing

Common mechanism and issues

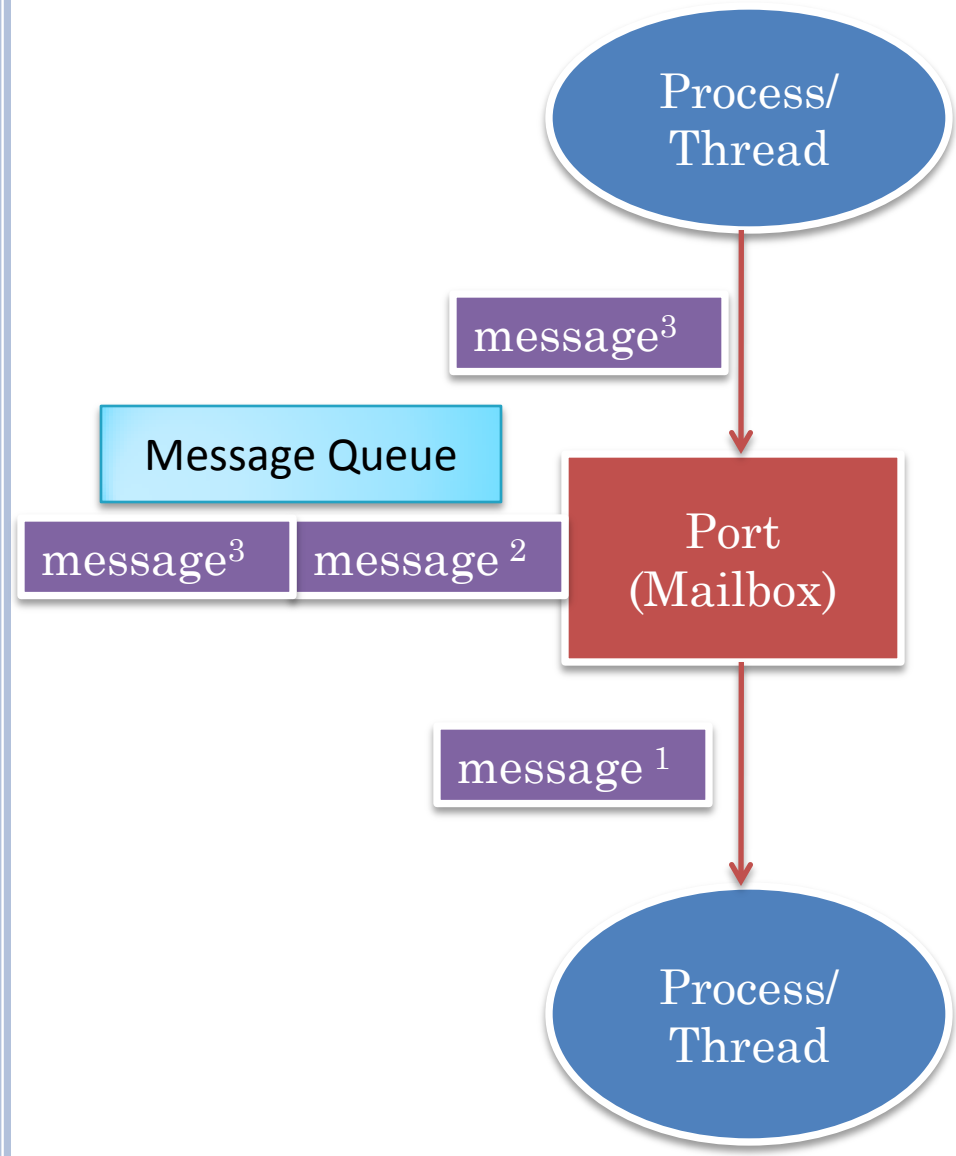
Abstract model to cover all of them

GIPC (Generalized IPC) based on abstract model

Unifying Basis?



Message Passing: Common Basis



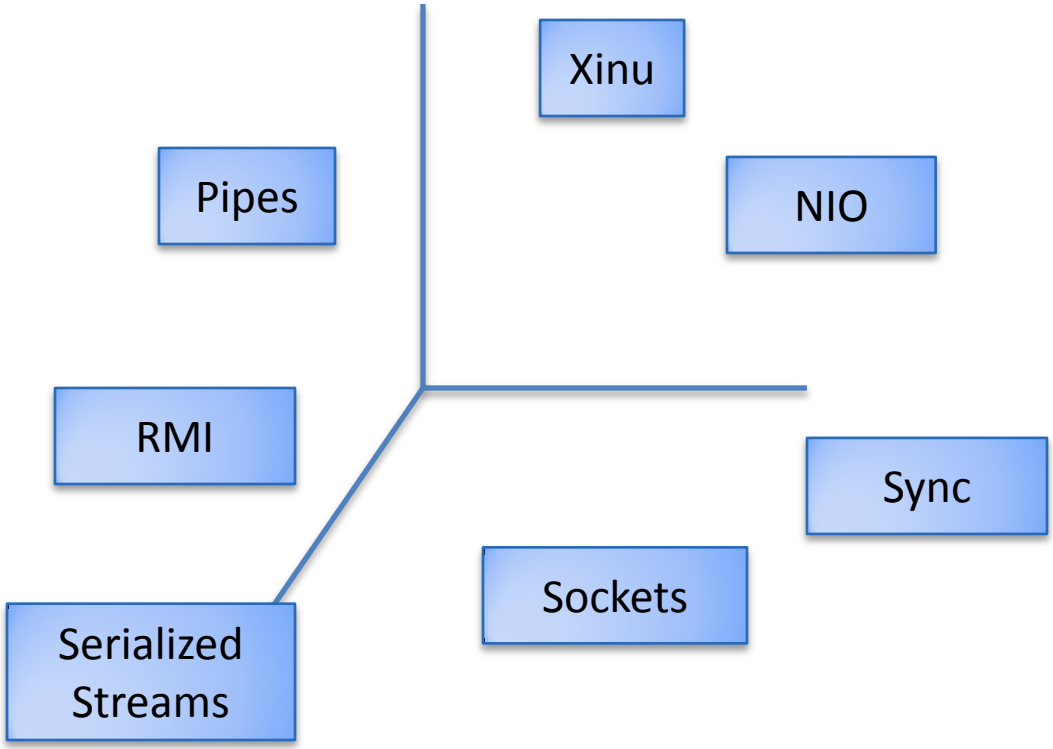
Processes /threads send each other messages

Sender deposits message in port queue (of varying size)

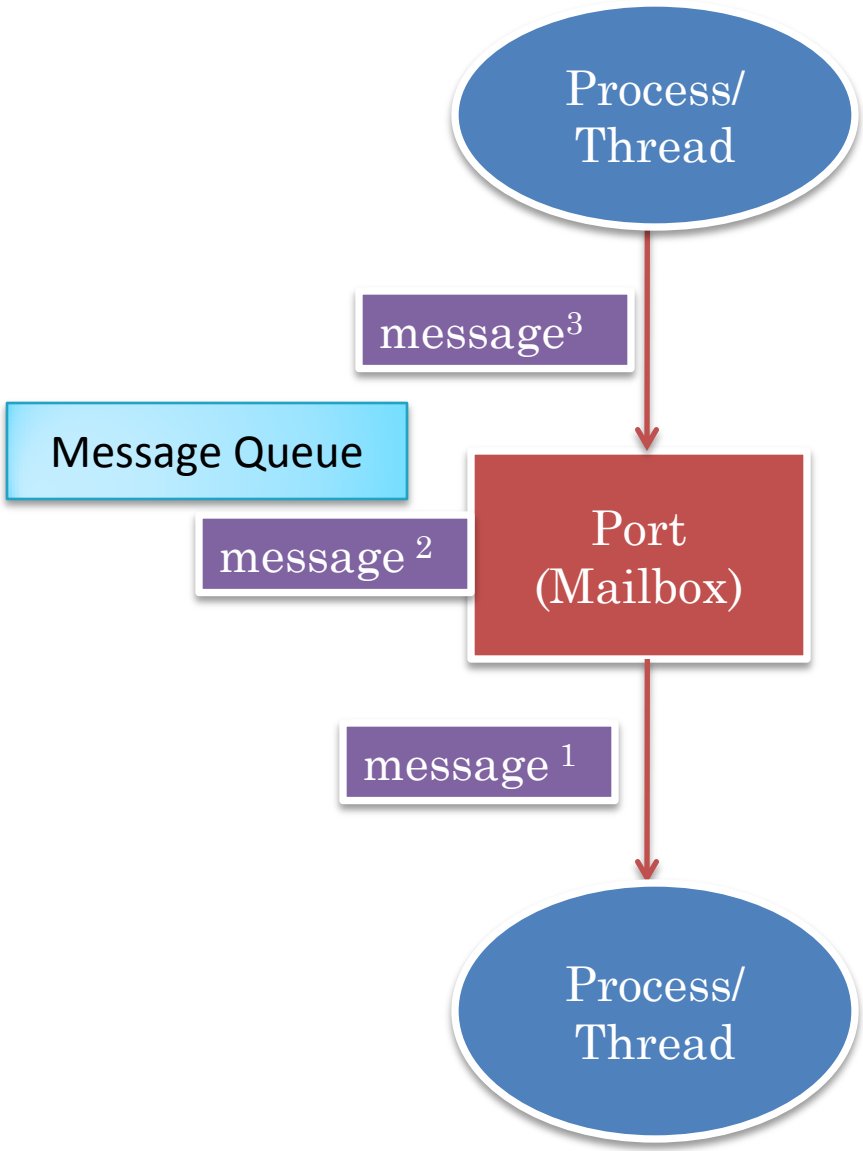
Receiver gets message from queue

Issues?

Distinguishing Issues?

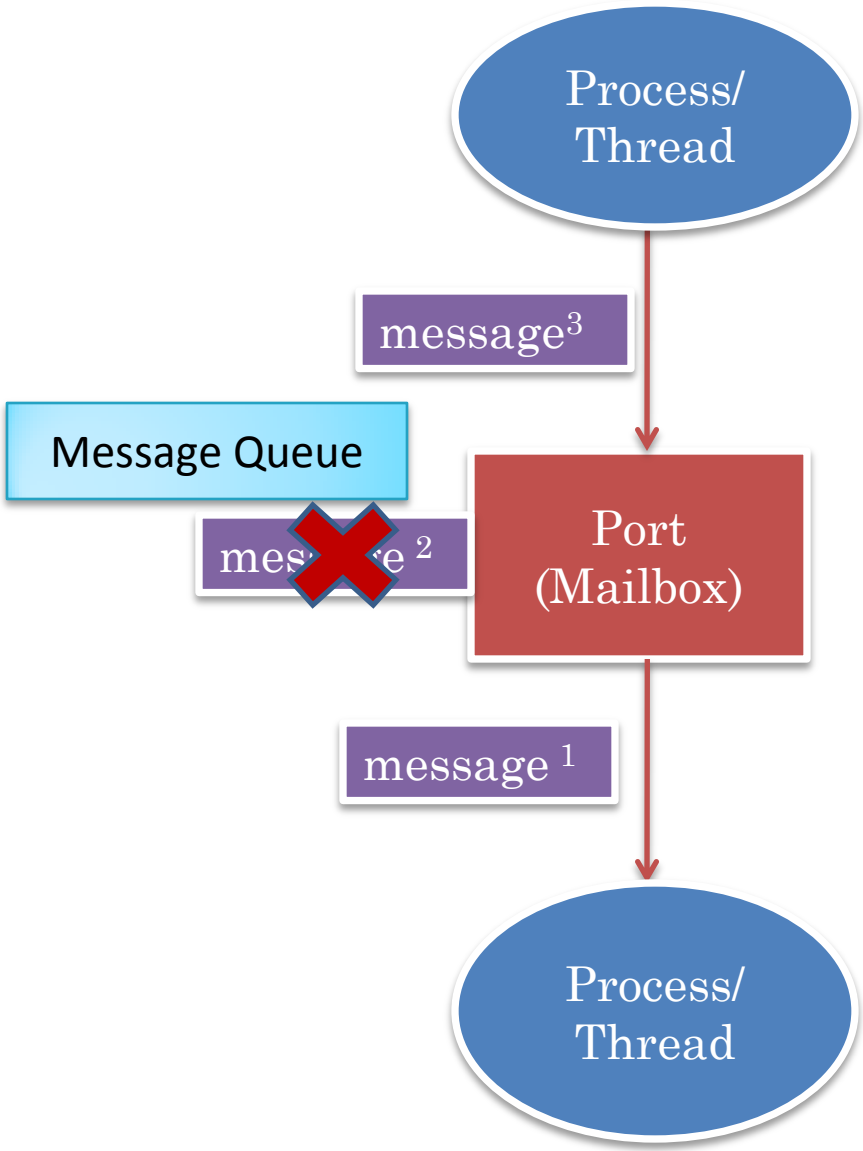


Issues in Message Passing



- Reliable message delivery?
- In-order message delivery?
- Port access (and message routing)?
- Operations?
- Synchronous vs asynchronous ?
- Blocking vs non blocking?
- Buffering of messages at queue?
- Location of communicating threads?

Reliable vs. Un-Reliable



Reliable: every message sent to a port is receivable

Noise, disconnection can cause unreliability

When reliability needed, programmer does not need to implement it

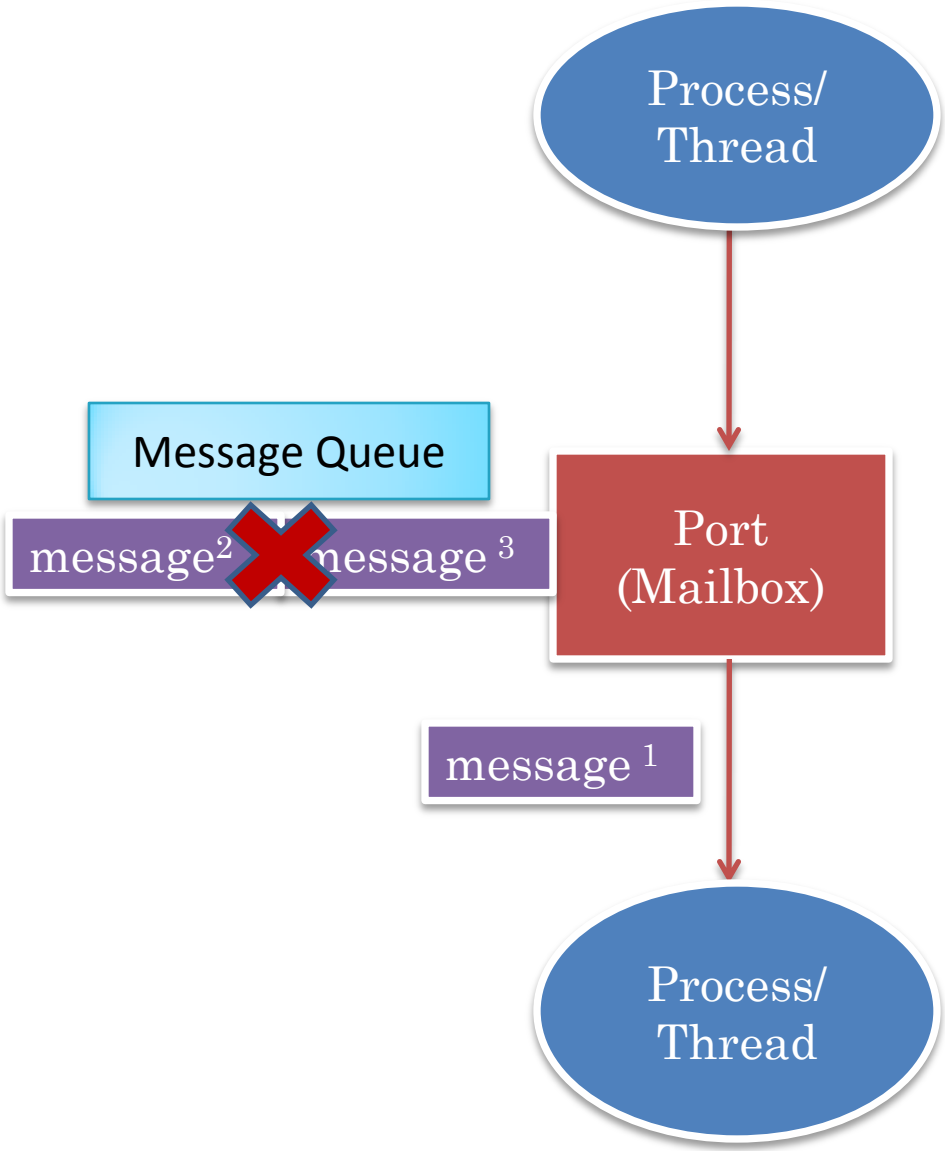
Reliable not always needed

Spam, load averages, telepointers

Reliable is less space efficient (redundancy) and time efficient (reliability algorithm)

Networking handles this issue

In-Order vs. Not In-Order Delivery



Order: message received in order in which they are sent

Different routes can take different times

When order needed, programmer does not need to implement it

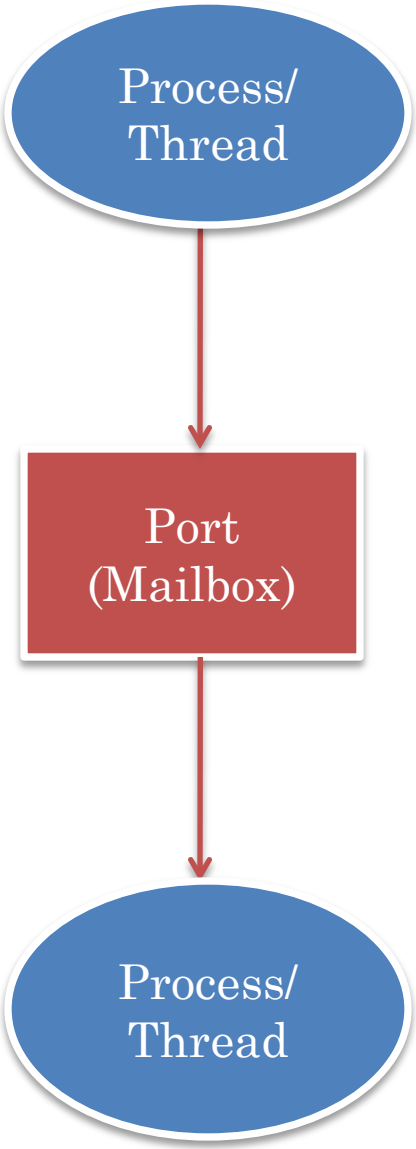
Order not always needed

Commuting operations

In-order is less space efficient (seq #) and time efficient (in-order algorithm)

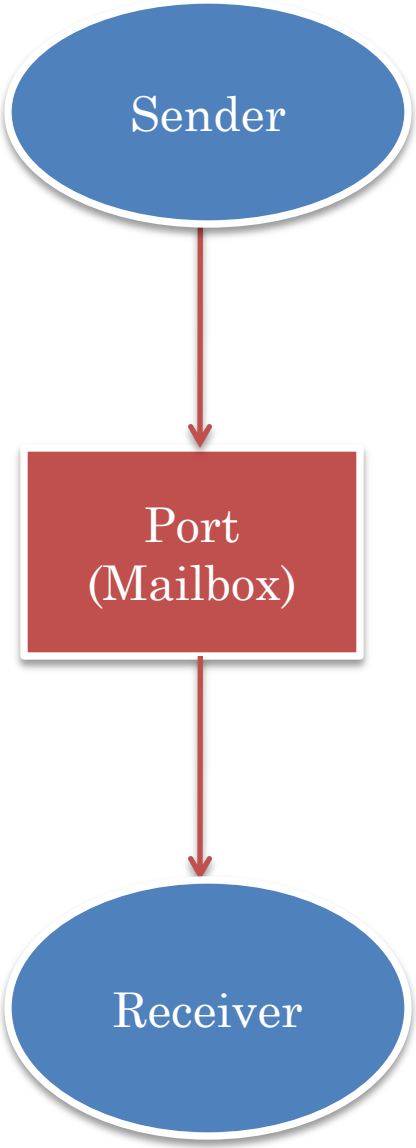
Networking handles this issue, but relevant to distributed computing as we see later

Access/Routing



Set of senders and receivers?

Simplex Bound Port



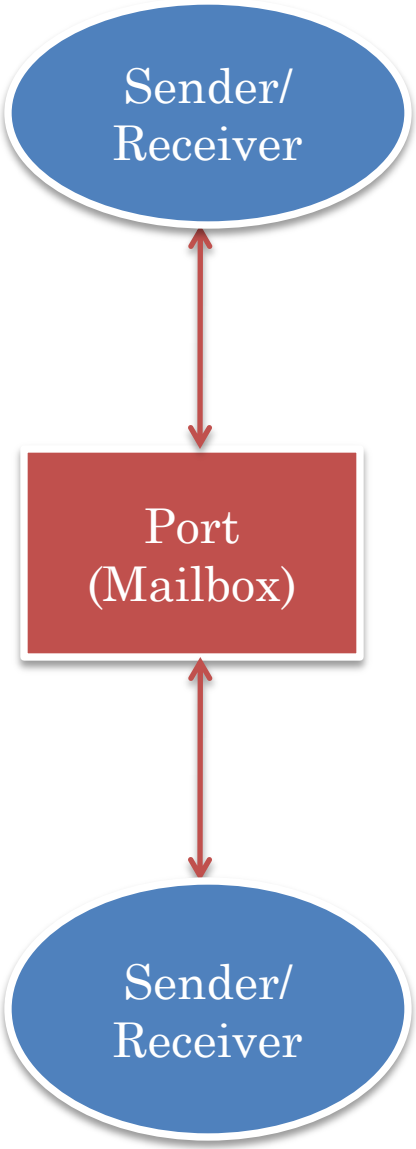
Communication between a pair of processes/threads

One is the sender and another the receiver

Example?

A port can represent an opened file in a file server - File server sends file bytes to the file client

Duplex Bound Port



Communication between a pair of processes/threads

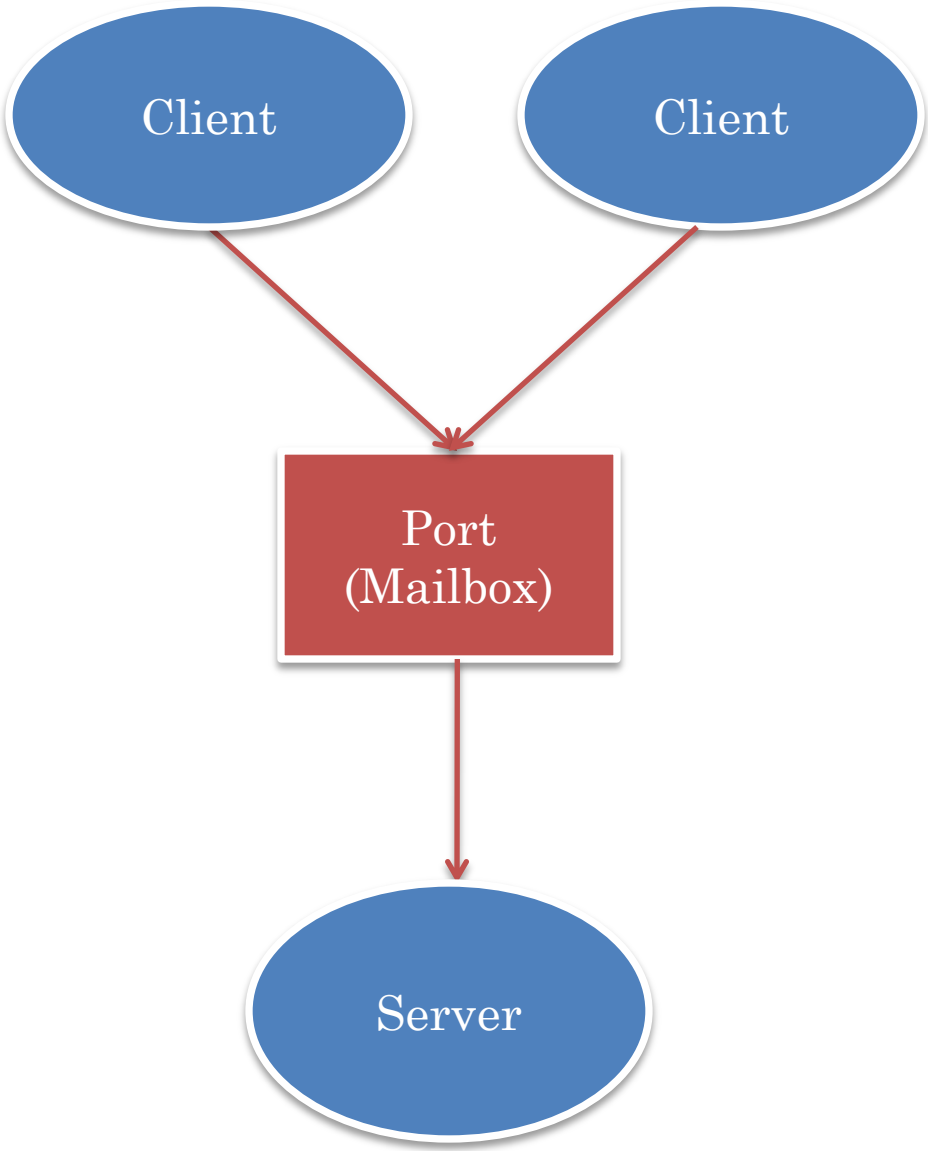
Either can send or receive

Example?

HTTP connection, two–player game

> 1 sender, receiver?

Simplex Input Port



A single receiver, called server

Arbitrary number of senders, called clients

Set of simplex ports?

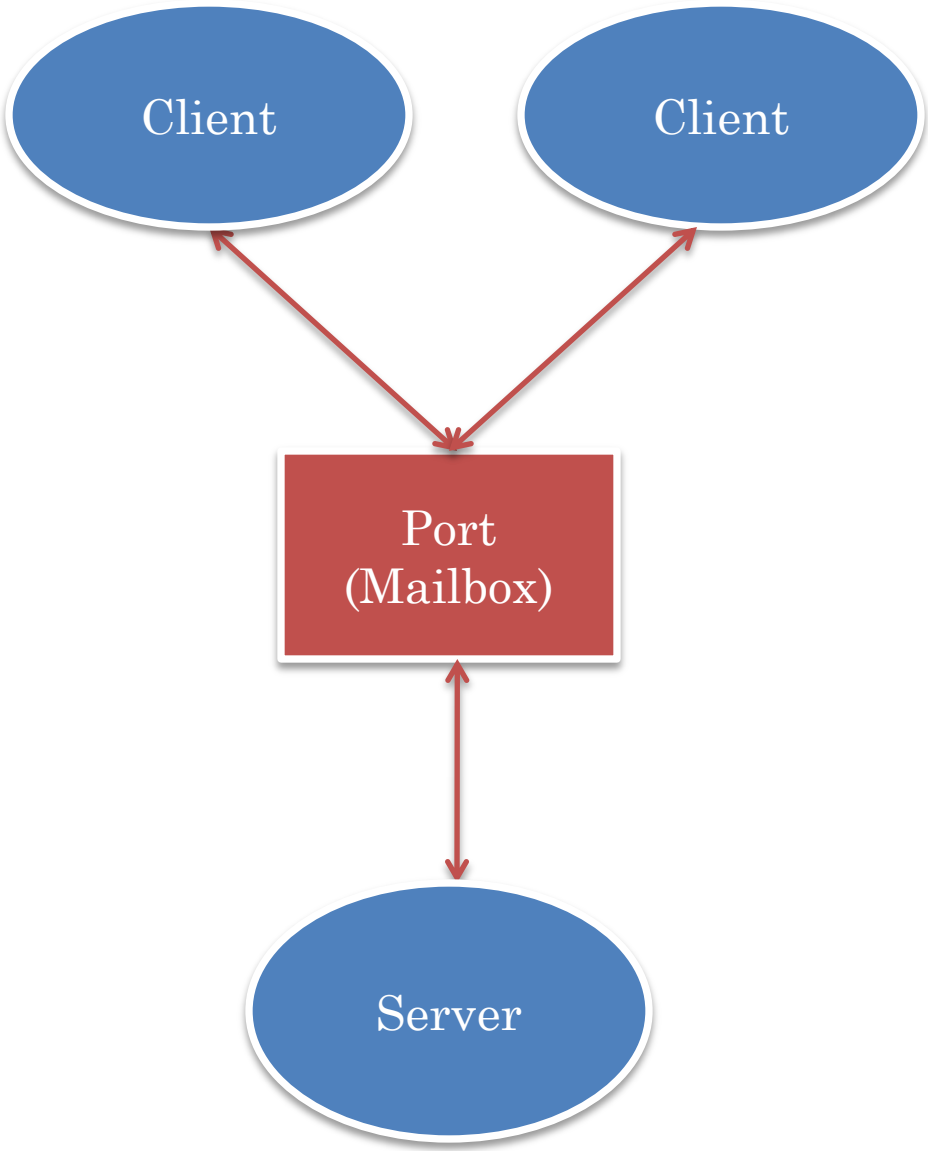
Messages from all clients queued together (not a collection of simplex bound ports)

Example?

Print Server

Few examples where no information comes back from server

(Replying) Duplex Input Port



Like simplex equivalent except server can also send messages to clients, each with separate queue

In replying duplex input port, server can only reply back to messages, cannot initiate sends

Security, simpler API

Example of replying?

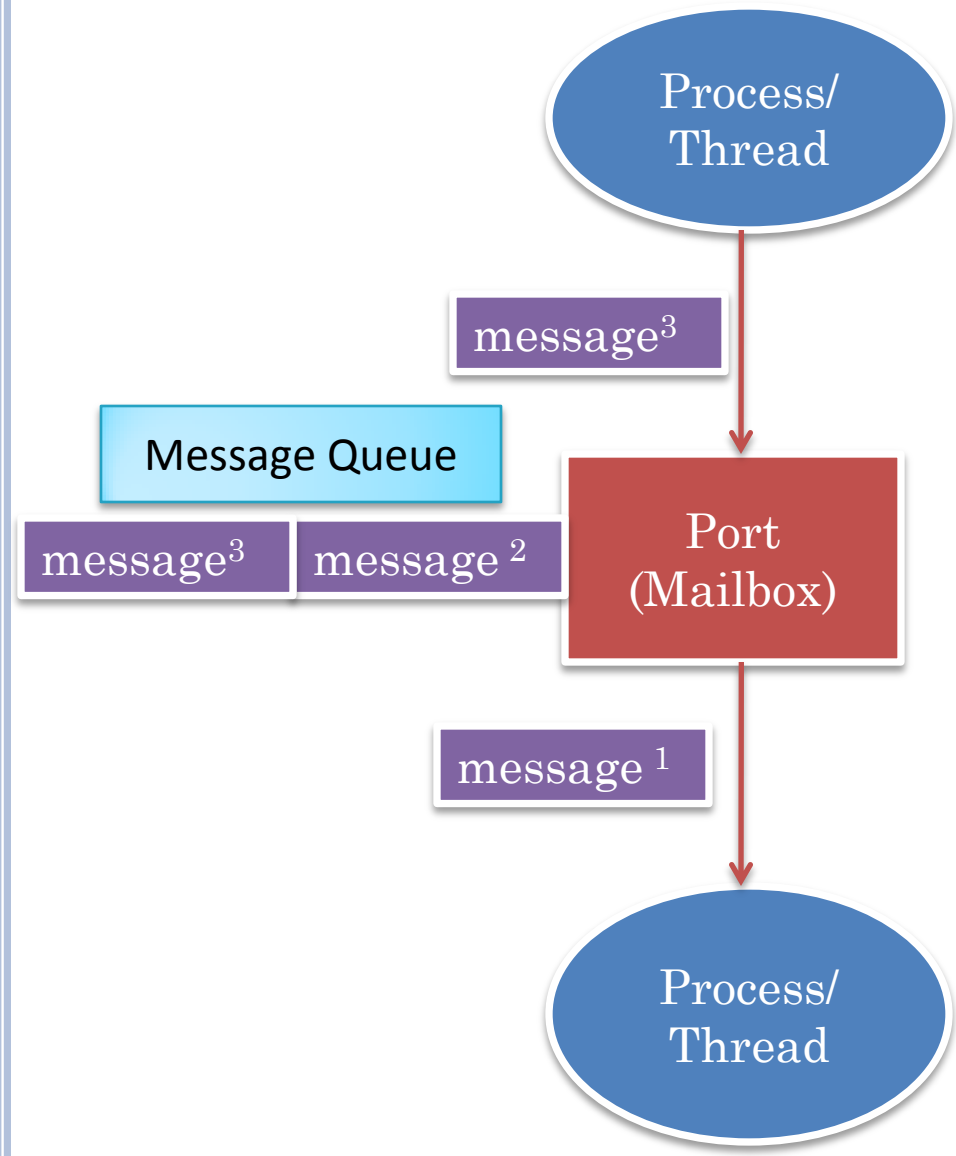
Http Server, File Server (open, close)

Example of general?

Session manager, relayer

Dual of input port?

Message Passing: Common Basis (Review)



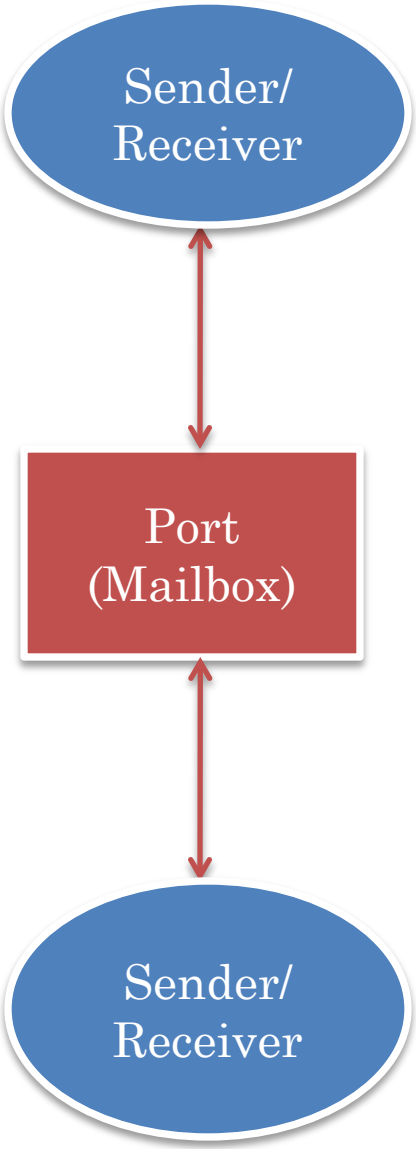
Processes /threads send each other messages

Sender deposits message in port queue (of varying size)

Receiver gets message from queue

Issues?

Duplex Bound Port (Review)



Communication between a pair of processes/threads

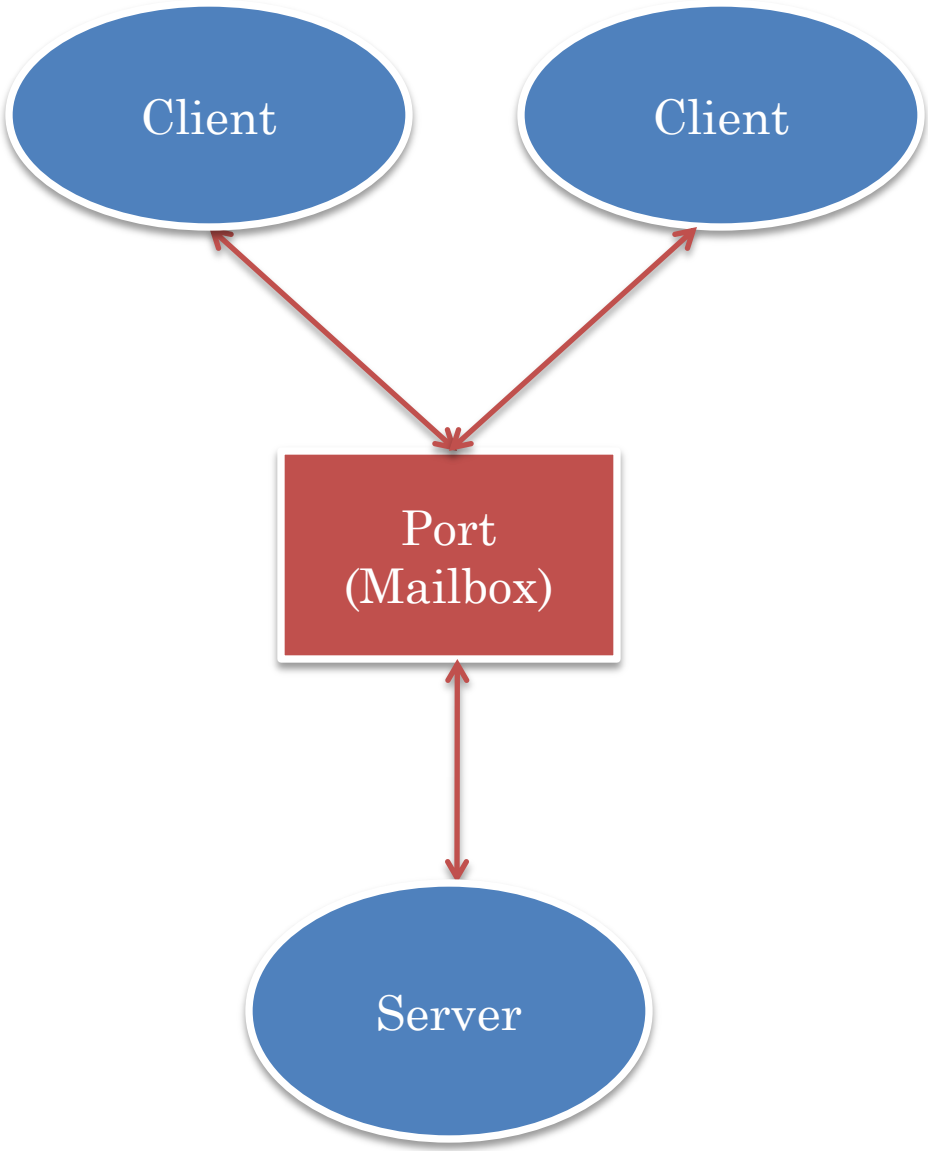
Either can send or receive

Example?

HTTP connection, two–player game

> 1 sender, receiver?

(Replying) Duplex Input Port (Review)



Like simplex equivalent except server can also send messages to clients, each with separate queue

In replying duplex input port, server can only reply back to messages, cannot initiate sends

Security, simpler API

Example of replying?

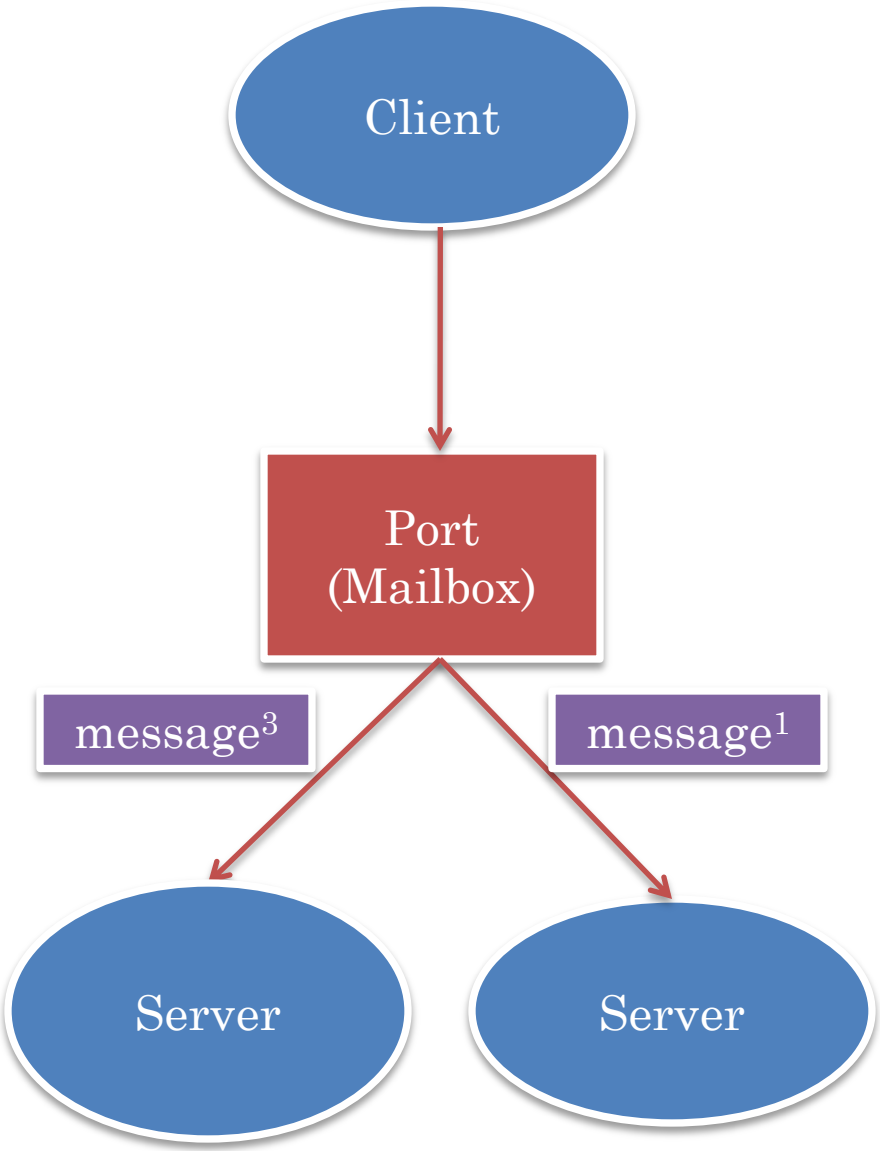
Http Server, File Server (open, close)

Example of general?

Session manager, relayer

Dual of input port?

Simplex Output Port



A single sender, called client

Arbitrary number of receivers, called servers, sharing a single queue of received messages

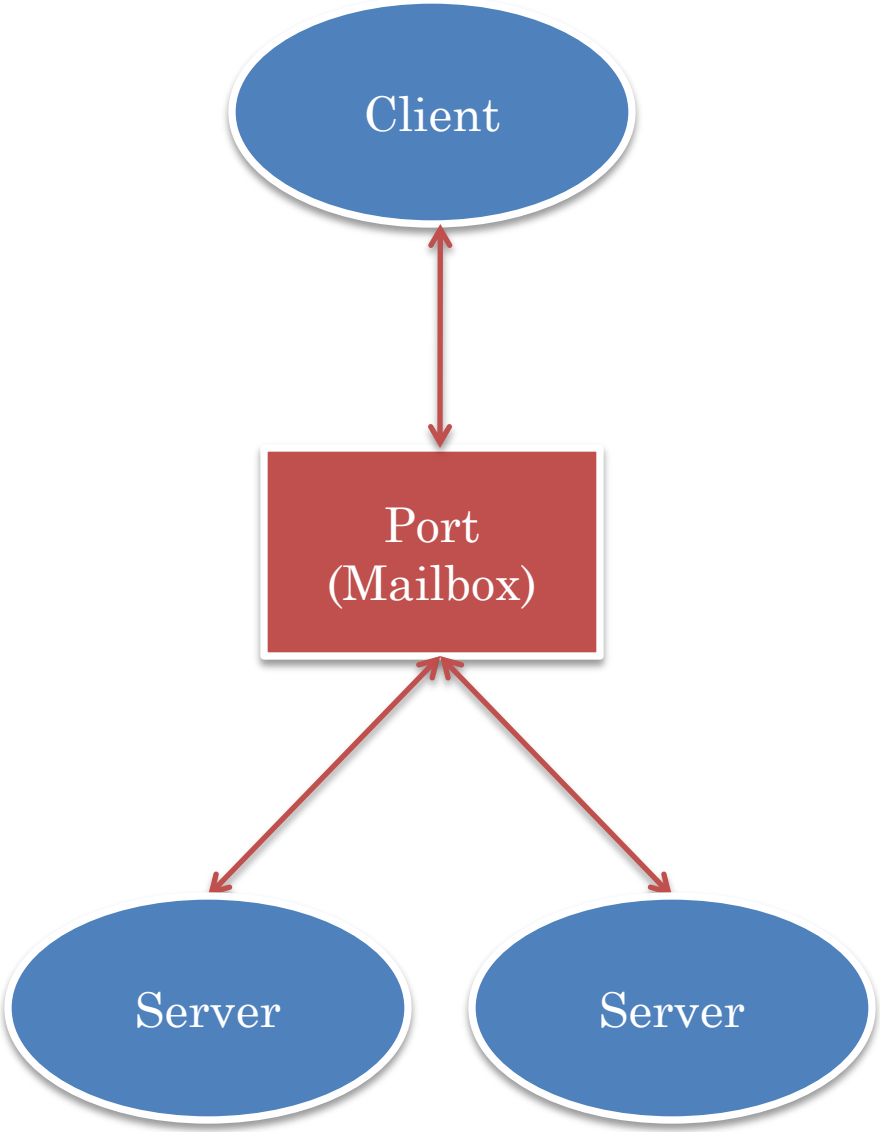
Broadcast?

A message goes to a single server – it is not broadcast

Example?

Coupon distribution

Duplex Output Port



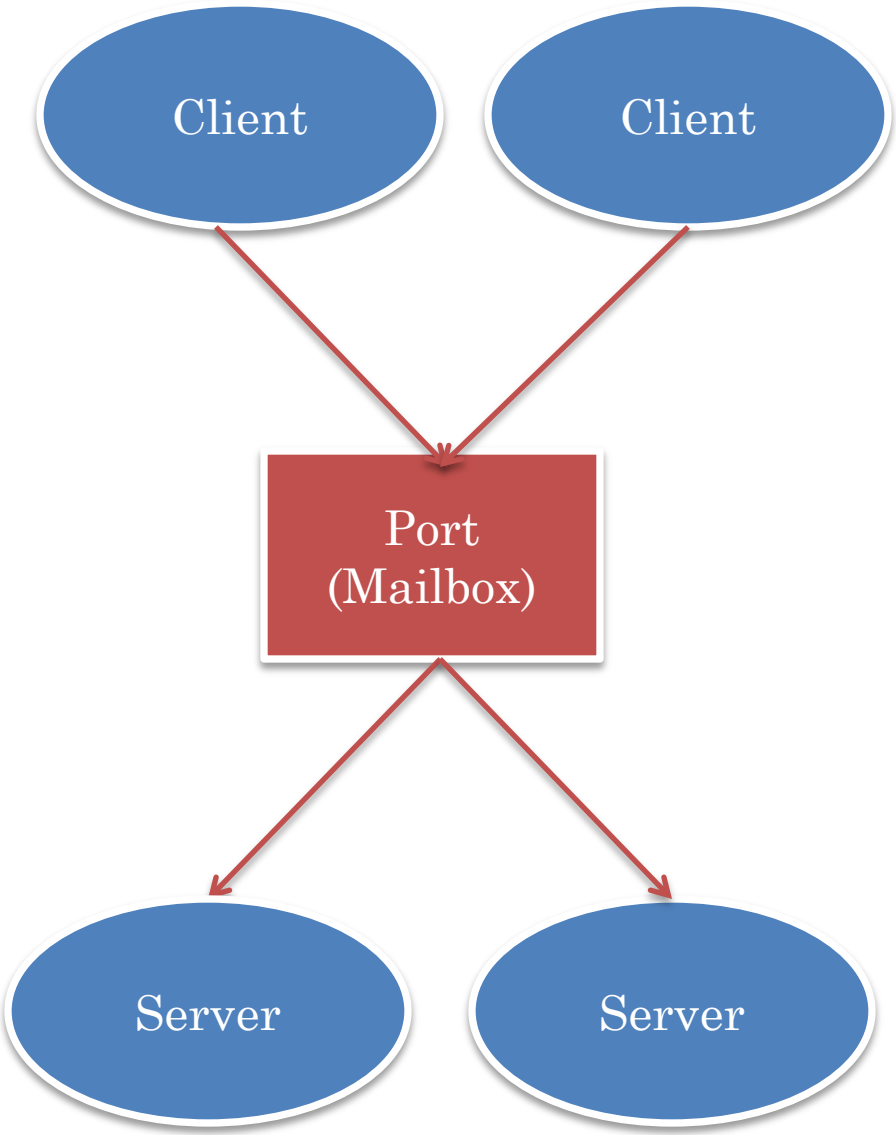
Like simplex output port, except servers can send messages to client

Example?

SETI computation

Few examples of one client with many servers

Simplex Free Port



Arbitrary number of senders, called clients, to a single message queue

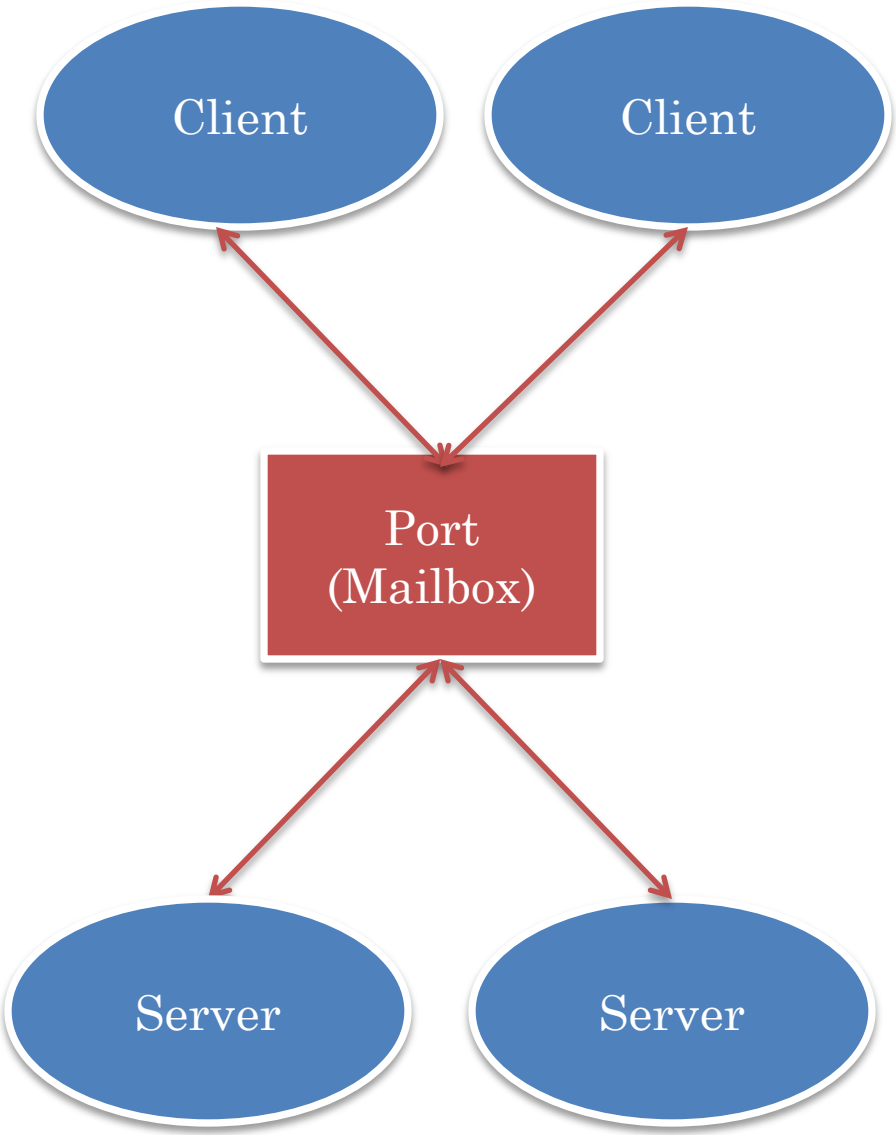
Arbitrary number of receivers, called servers, sharing a single queue of received messages

Examples?

Load distribution among servers

Shared pool of print servers

Duplex Free Port



Like simplex free port, except servers can send messages to client

Unlike servers, clients do not share a common message queue

Example?

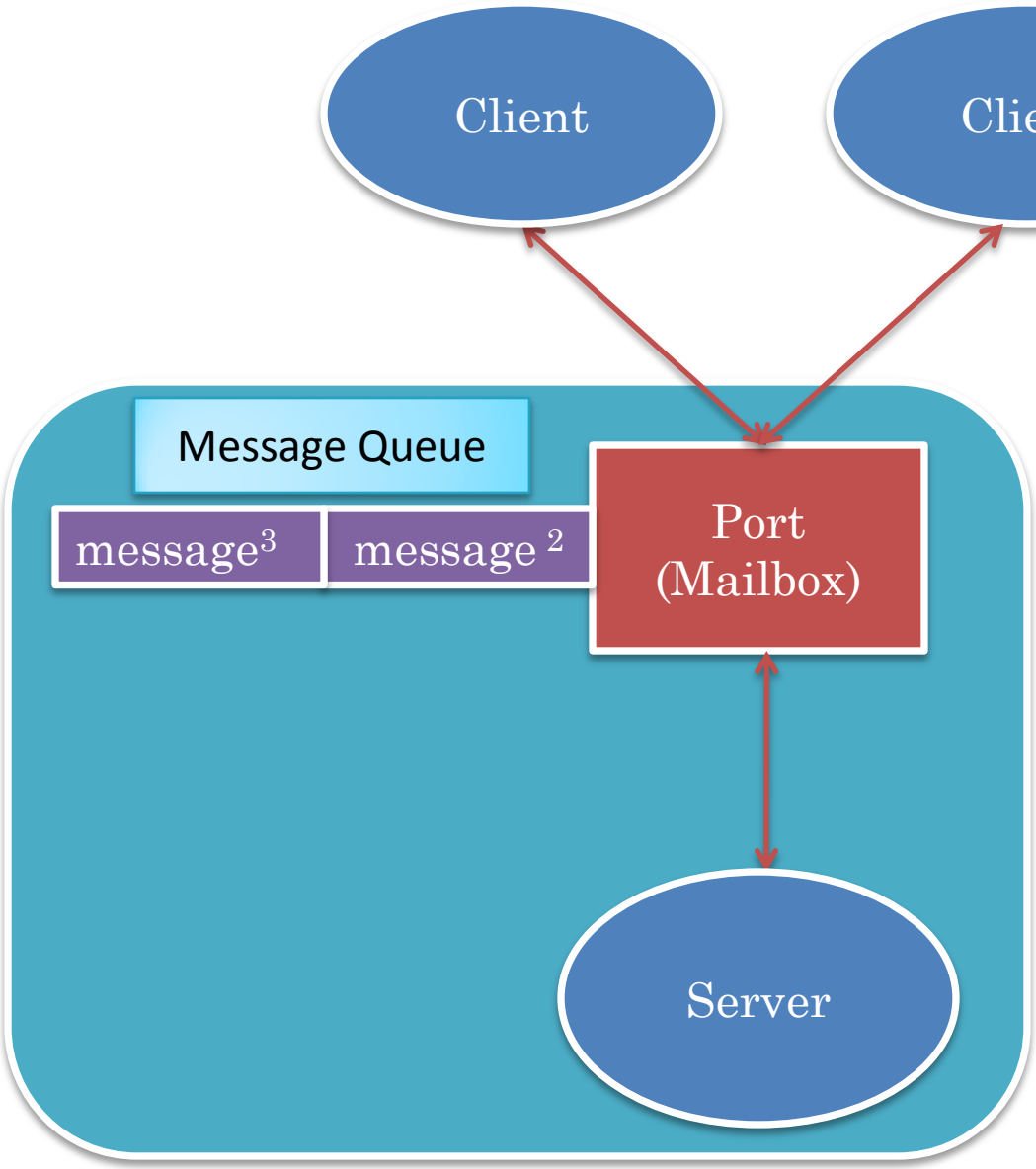
Web search server

General scalabale solution

Less efficient when clients on different computers

Which machine has the queue?

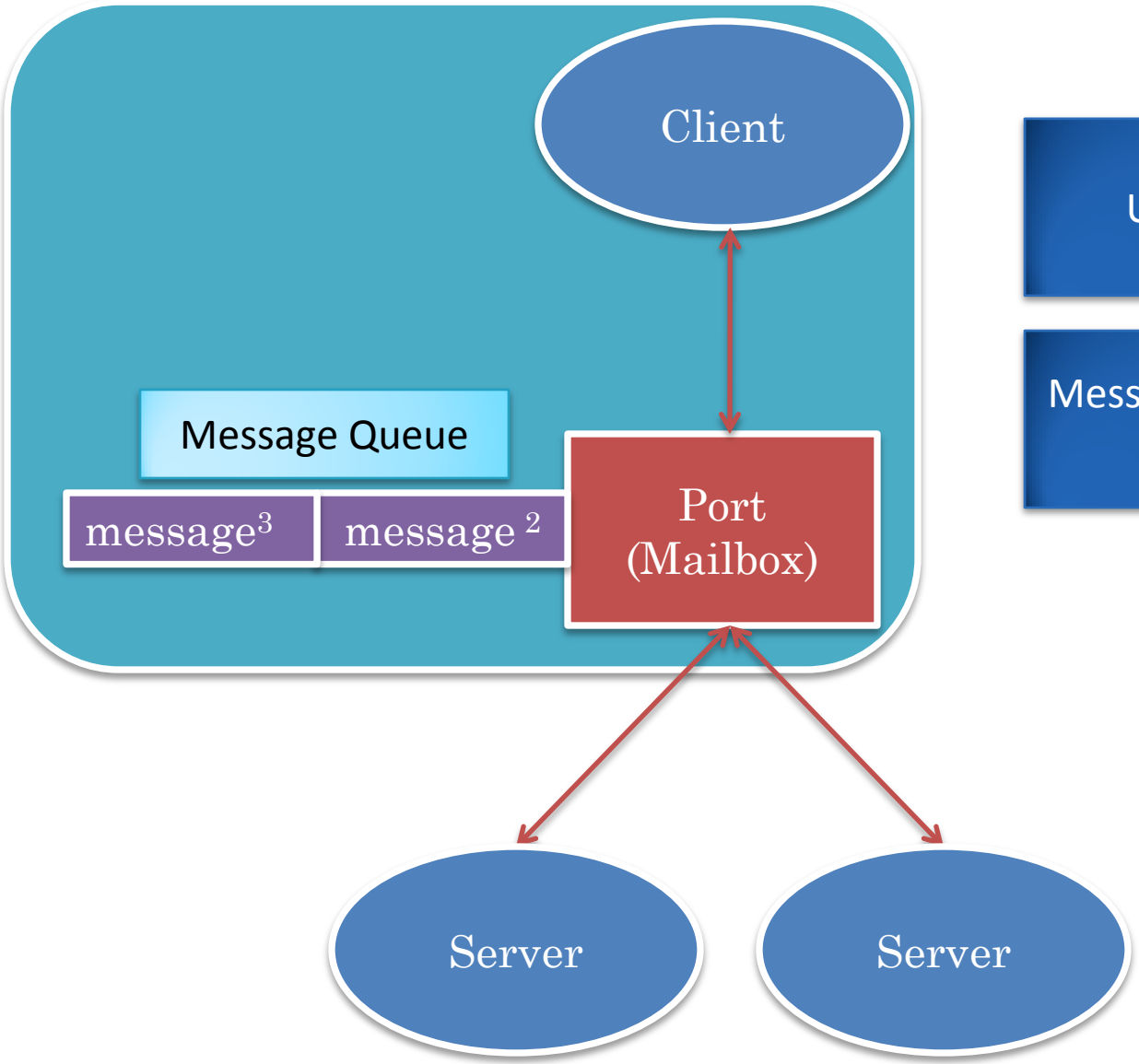
Queue Location in Input Port and Message Latency



Unique server has queue

Message sent from client to server - one hop

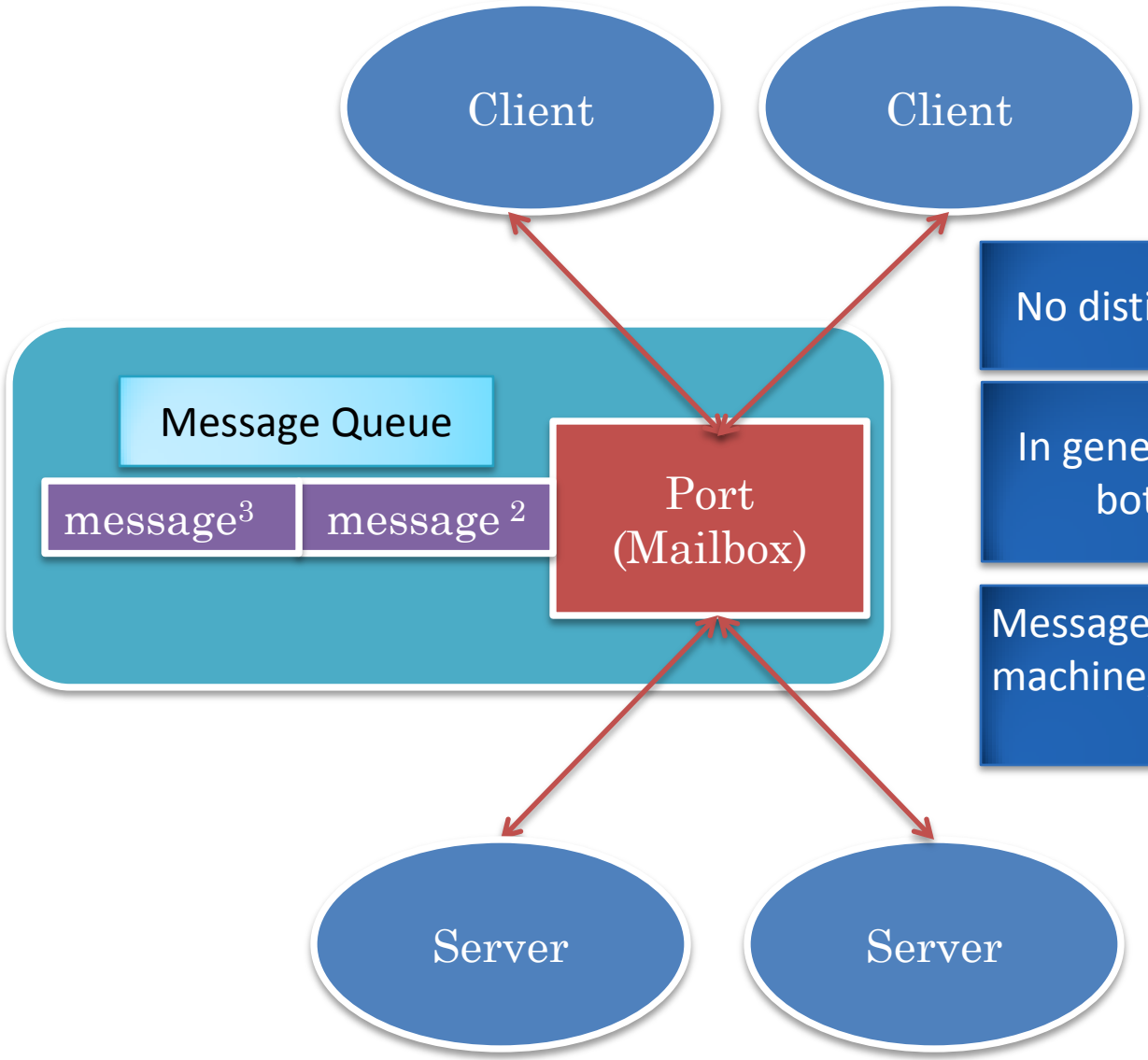
Duplex Output Port



Unique client, has queue

Message fetched by servers from client - one hop

Duplex Free Port

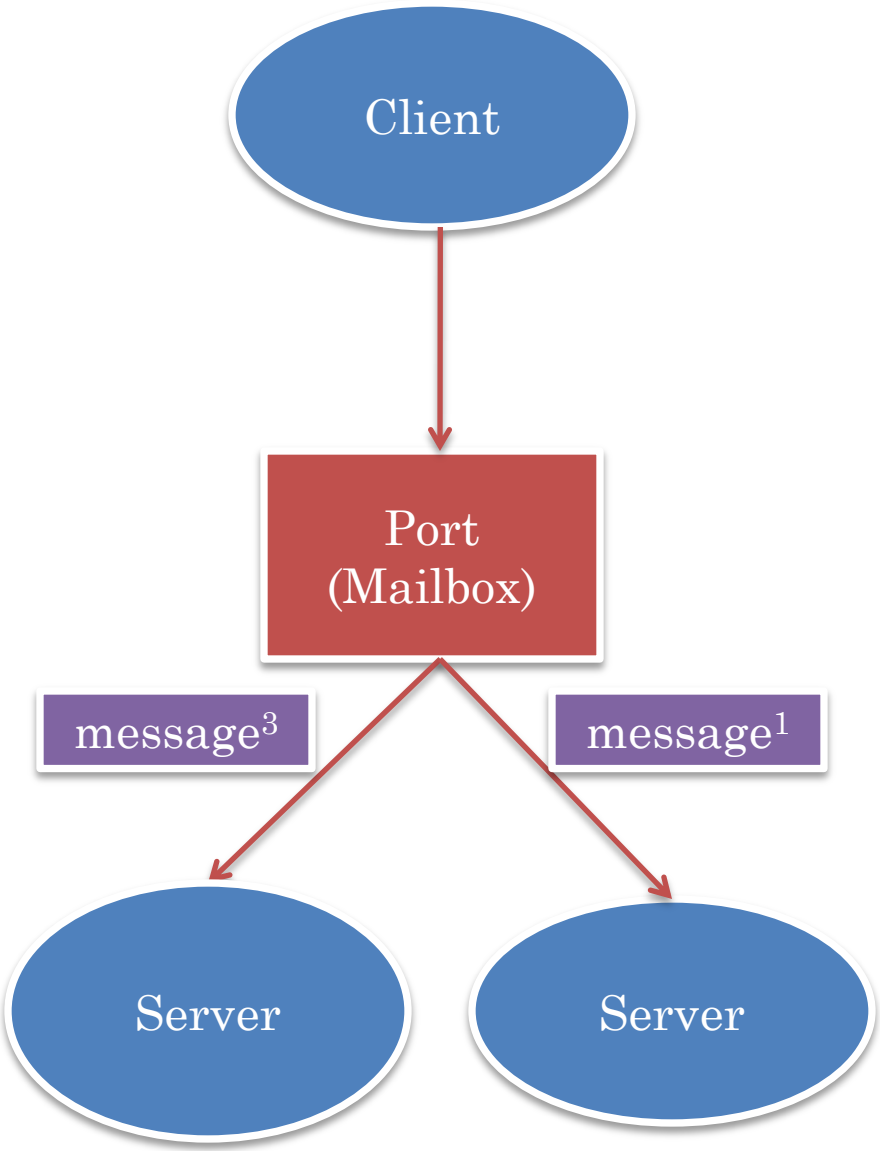


No distinguished client or server

In general queue is remote from both a client and server

Message goes from client to queue machine, and from queue machine to server

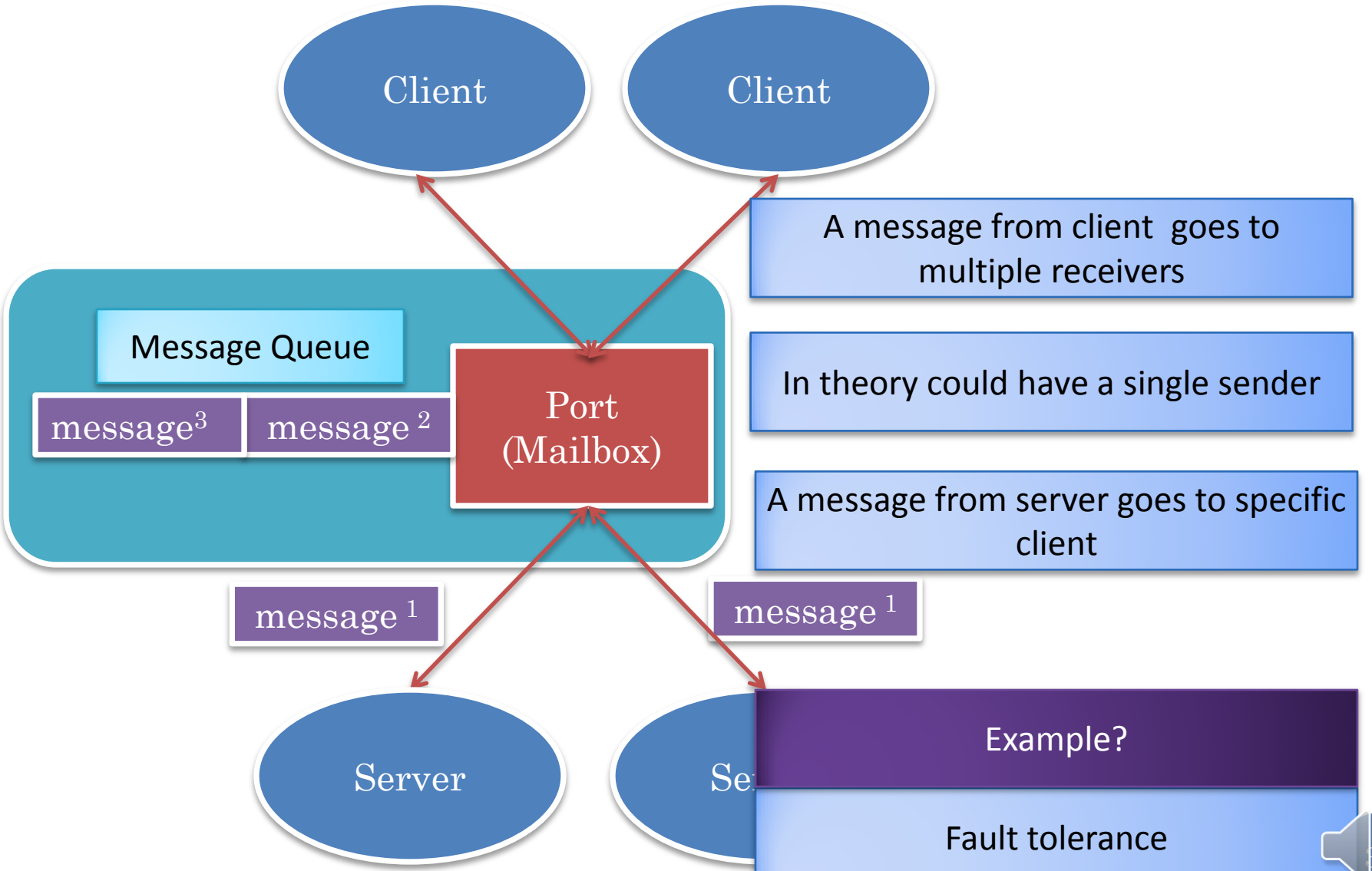
Output/Free Port



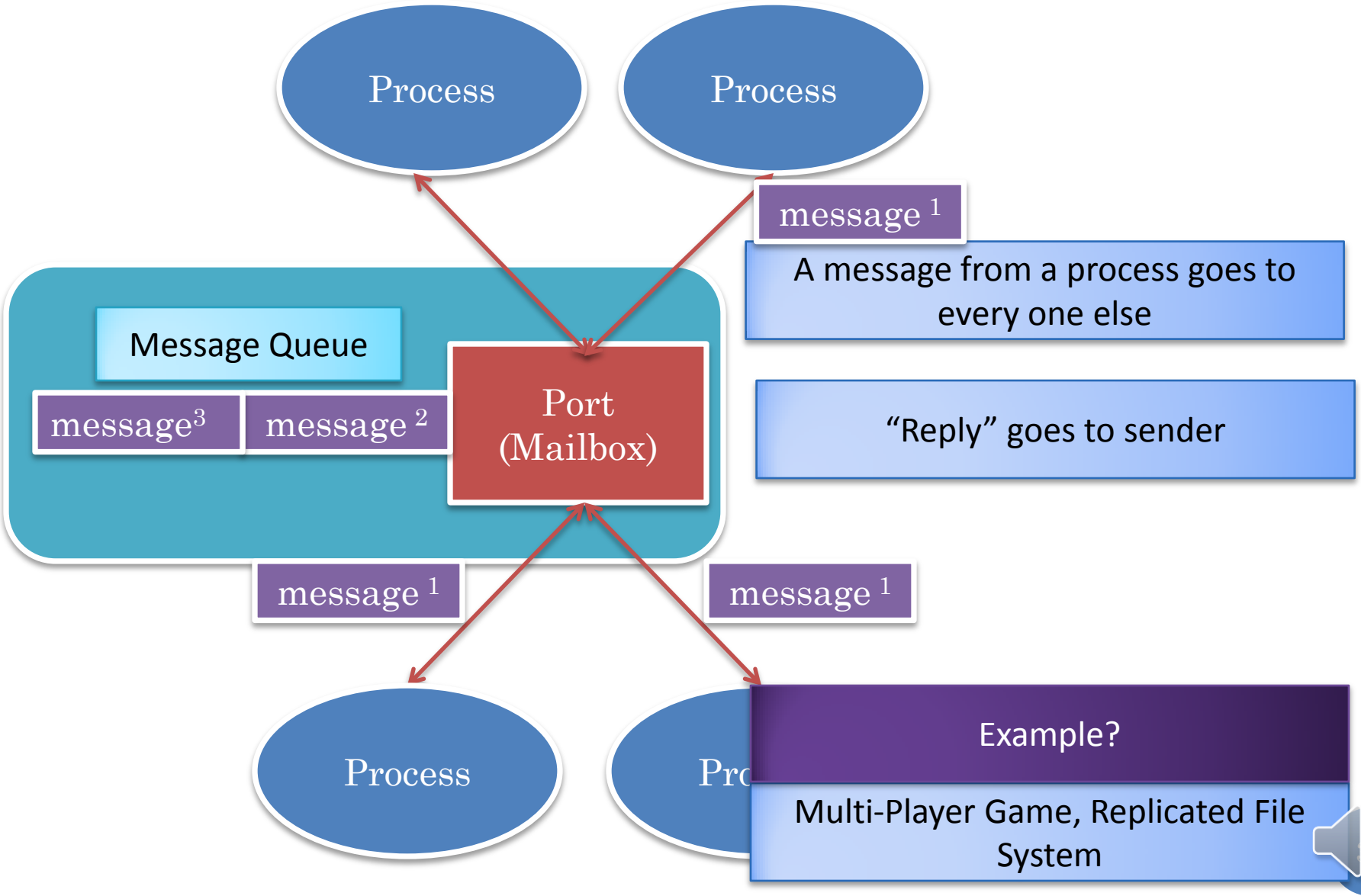
Broadcast?

A message goes to a single server – it is not broadcast

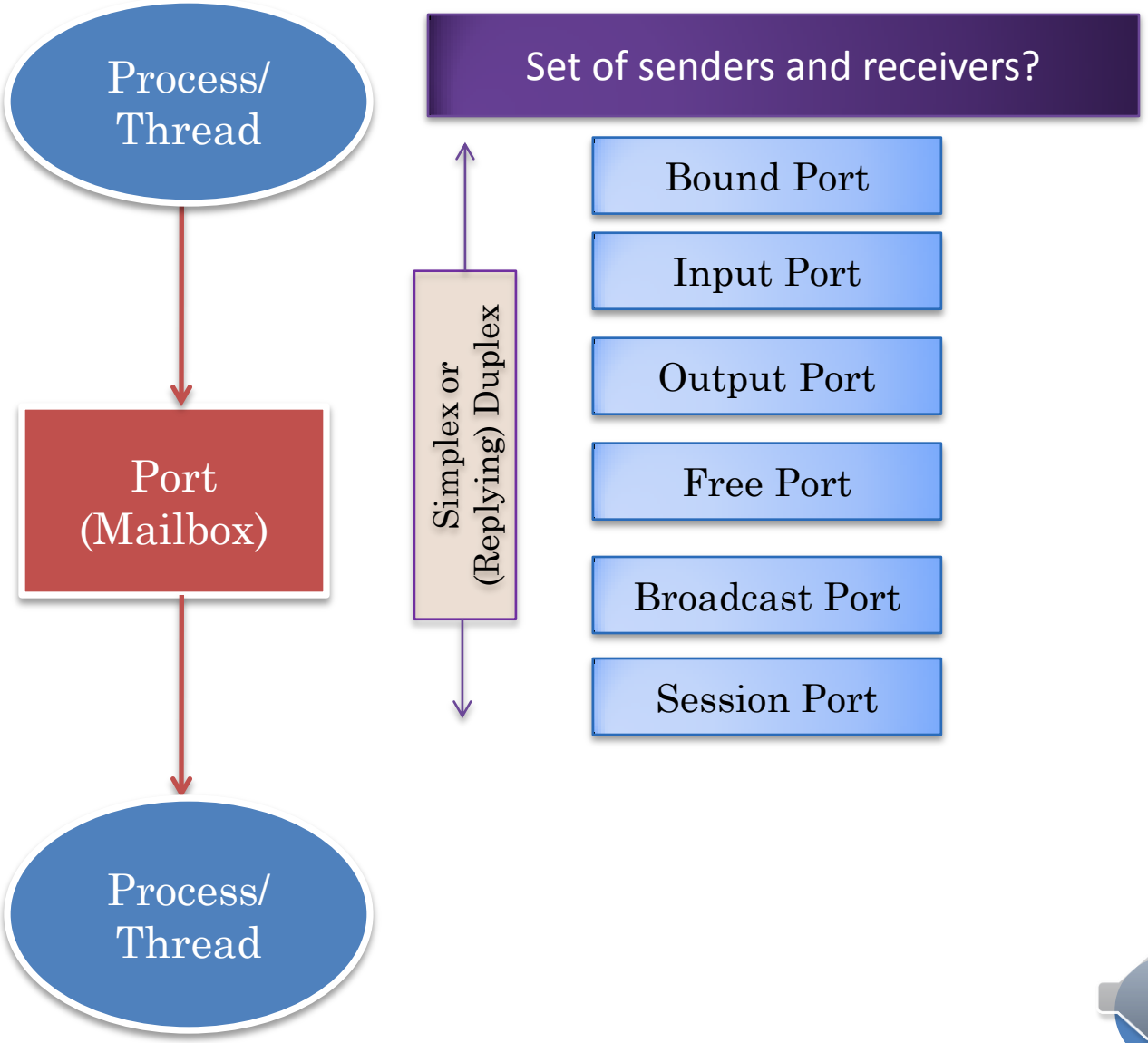
Multicast/Broadcast Port (Routing)



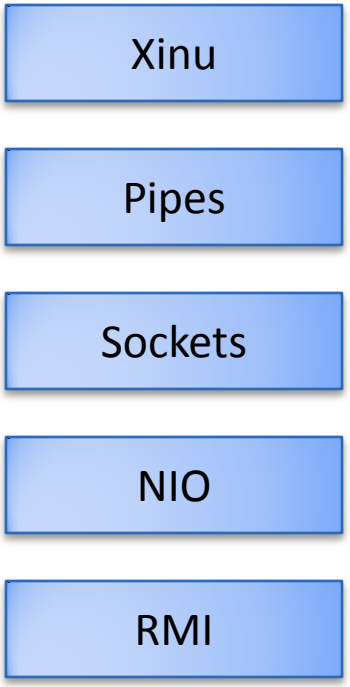
Session Port



Access/Routing



Distinction based on Access?



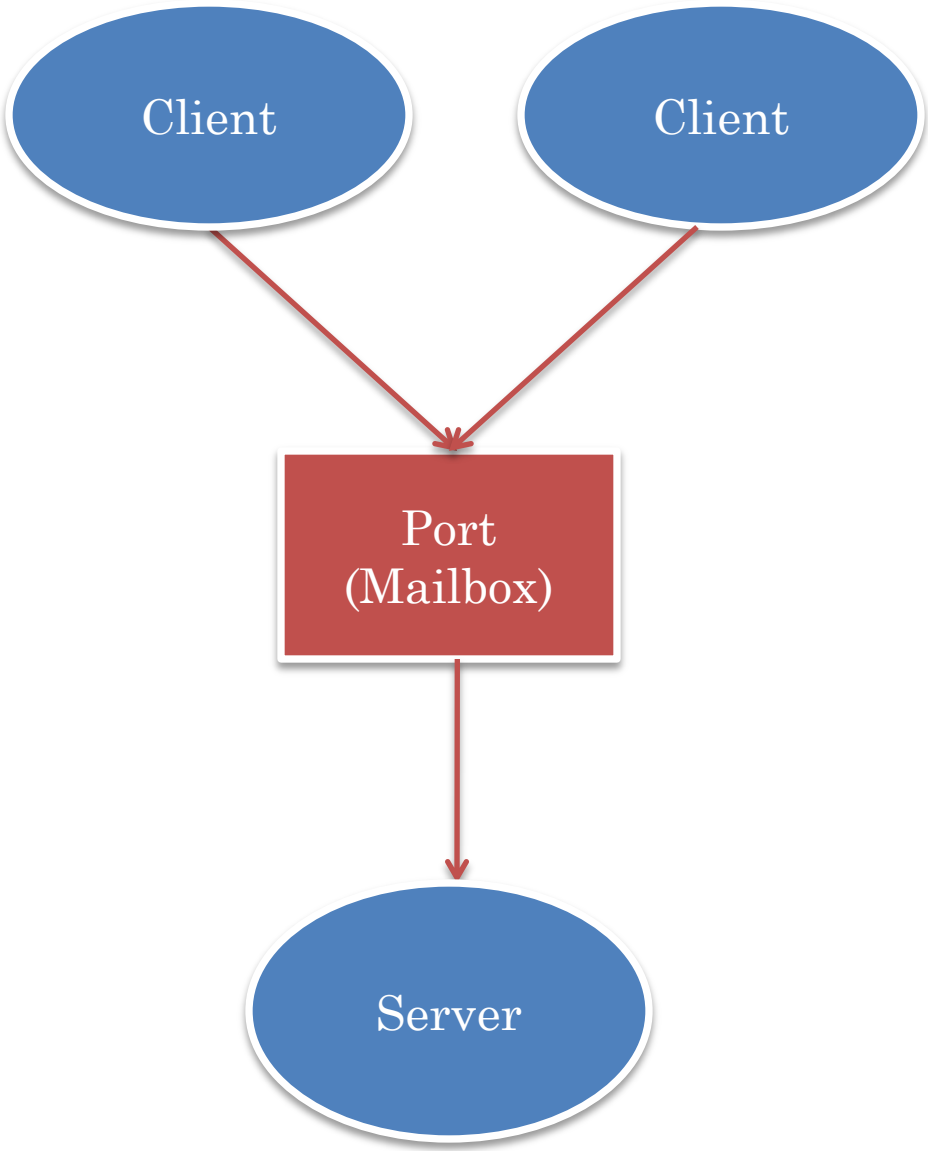
XINU

send (<thread_id>, <int expression>)

int receive ()

int recvclr ()

Xinu: Simplex Input Port



Each thread (process) has a single built in input port

A la each Java monitor having a single built in condition

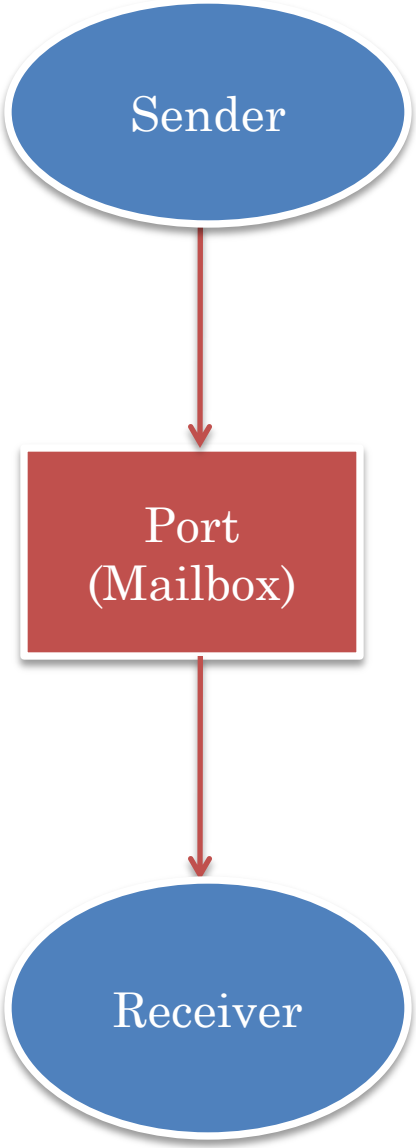
Less flexible

No need for port creation operations

Thread id is the port id

Not duplex, but sender can be sent back a message on its input port

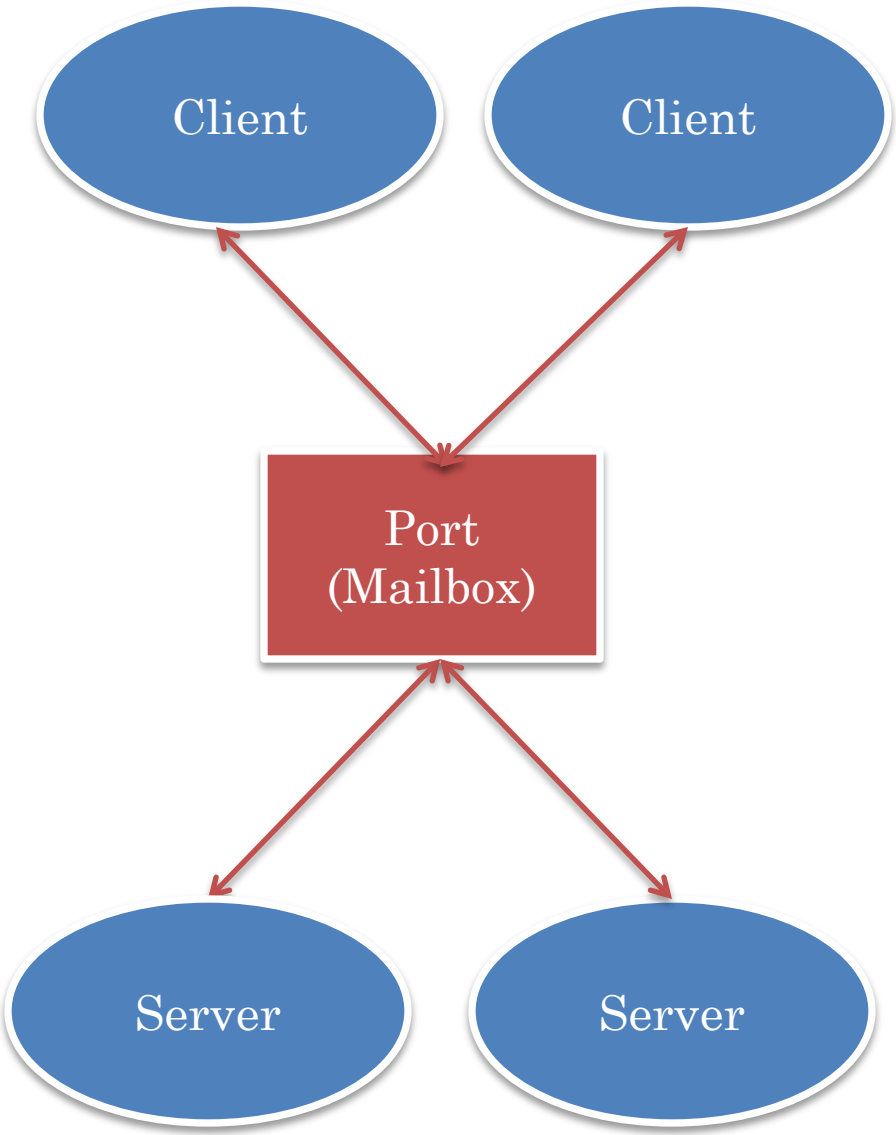
Pipes: Simplex Bound Port?



man 2 pipe | more

Command interpreter connects output of a process to input of another process

Duplex Free Port

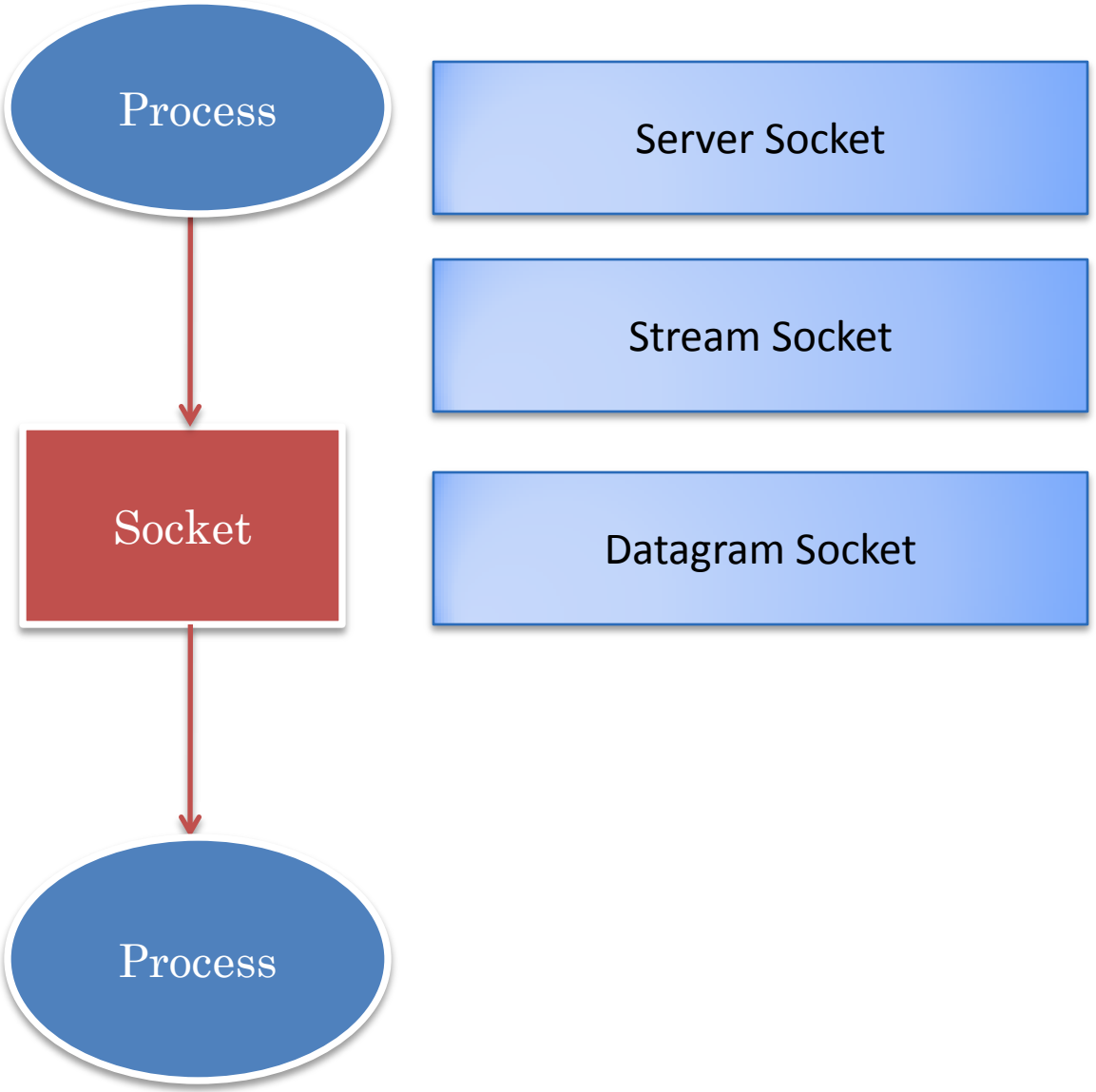


Some process calls pipe() to create the port

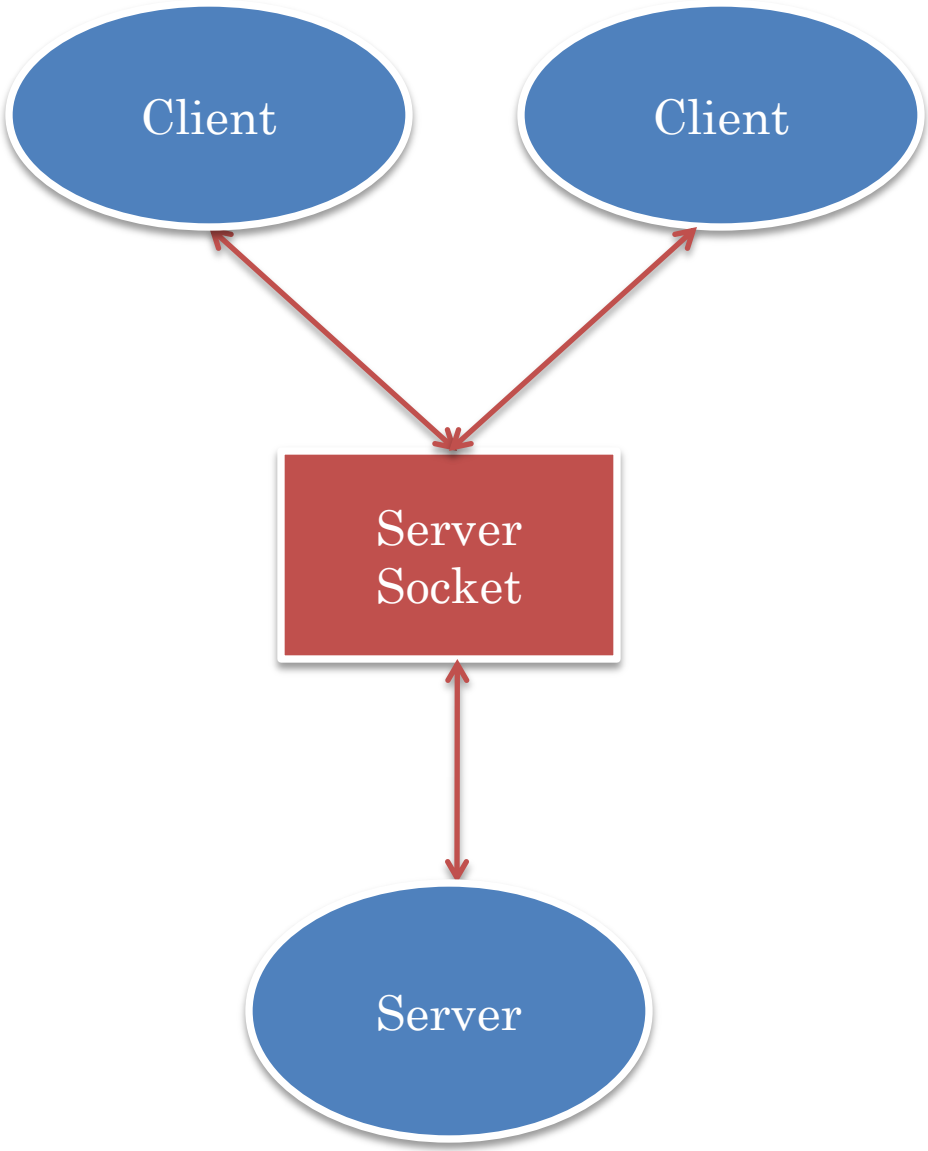
Children of pipe creator can send and receive messages

Usually one child writes and another reads

Socket/NIO Access: Many Kinds of Sockets



Server Socket: Duplex Input Port

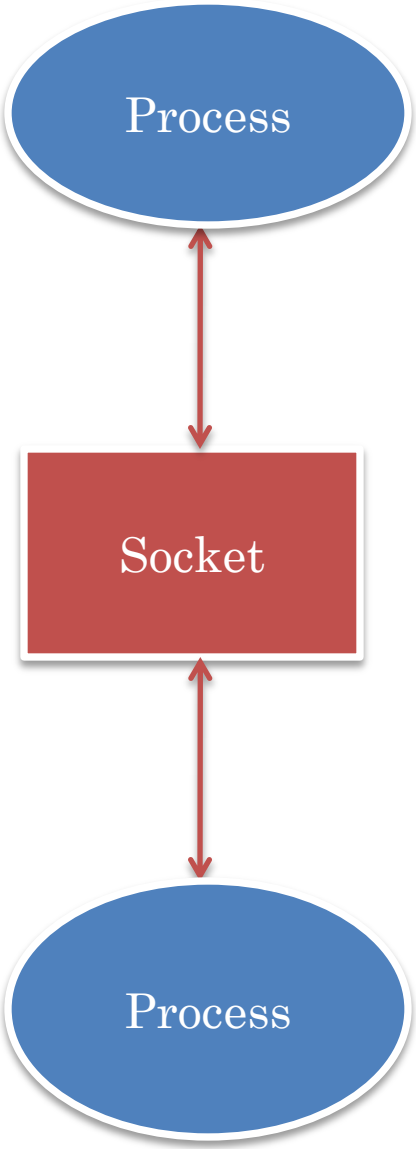


Server socket is used to create regular sockets

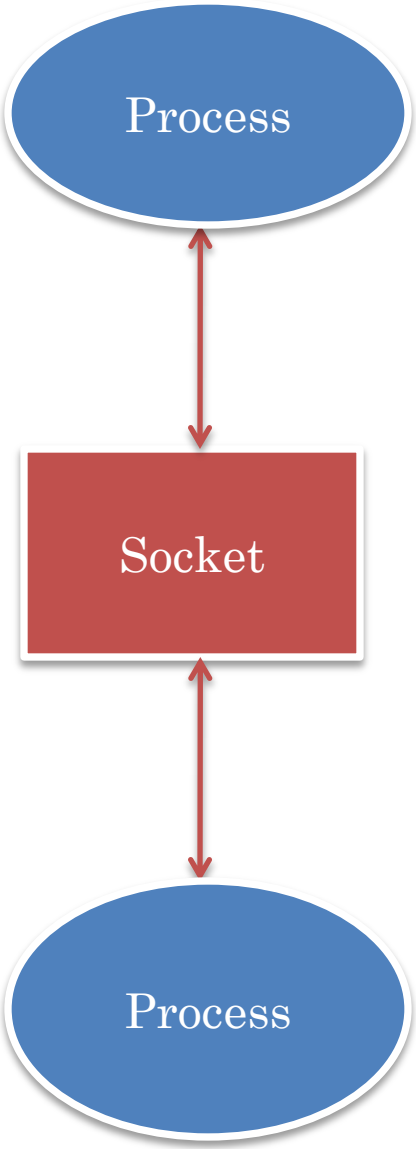
Represent a non-stream input port handling multiple clients

Each client connects to it to create a dedicated stream duplex port

Stream Socket: Duplex Bound Port?



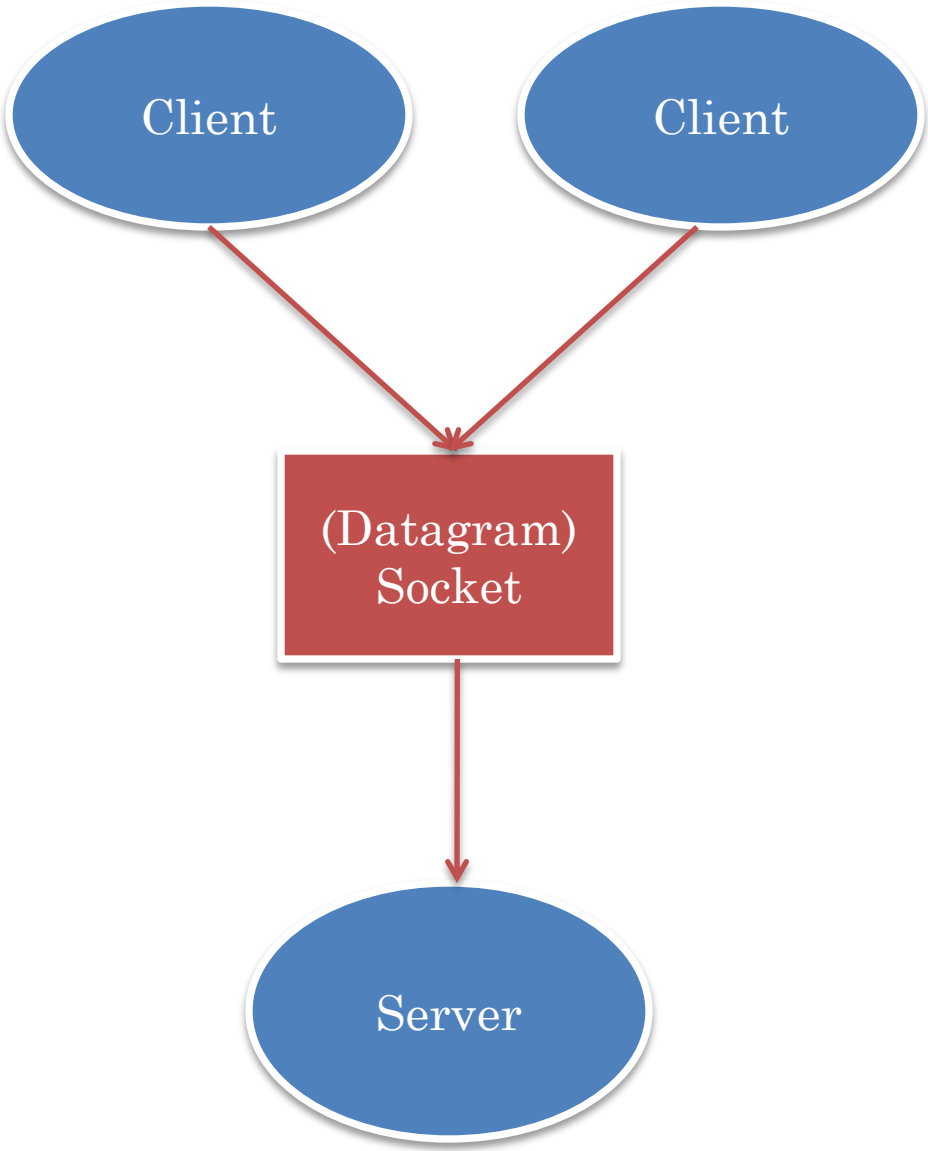
Regular Stream Socket: Duplex/Free Port



Regular socket represents a duplex port

Children processes share descriptors, so actually free port

Datagram Server Socket: Simplex Input Port

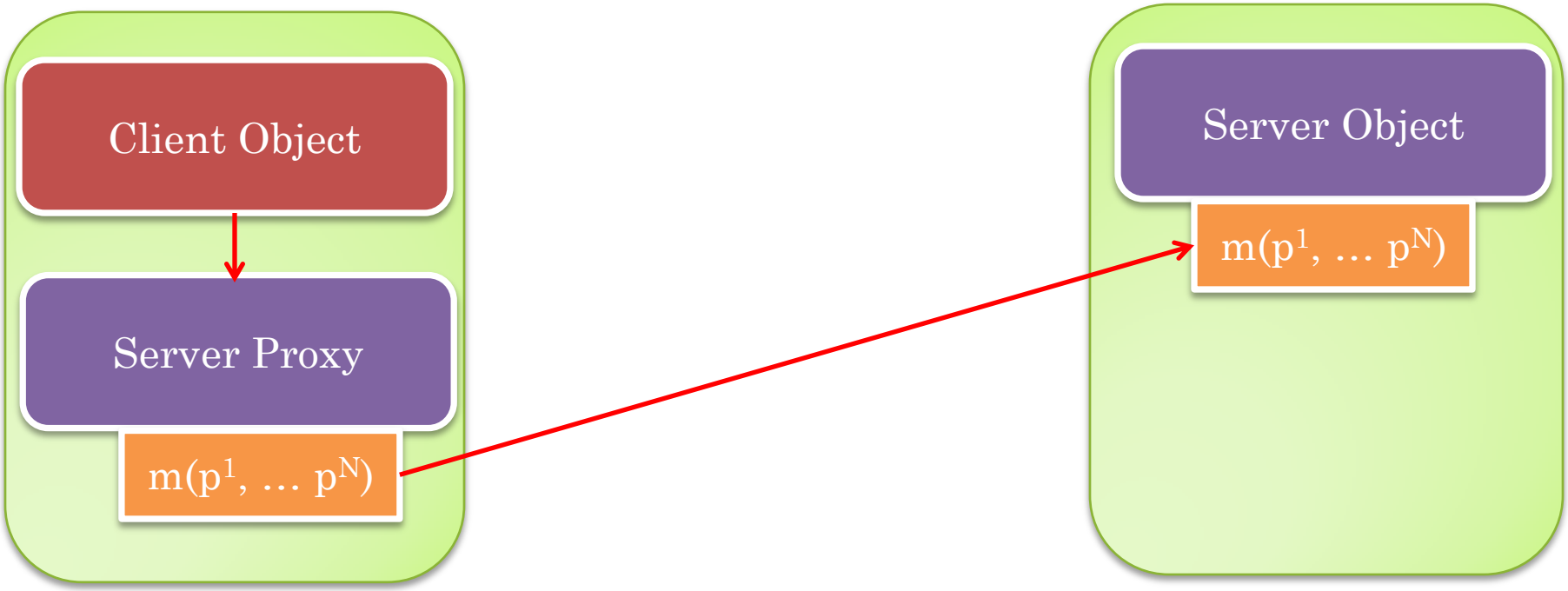


No need to create ServerSocket

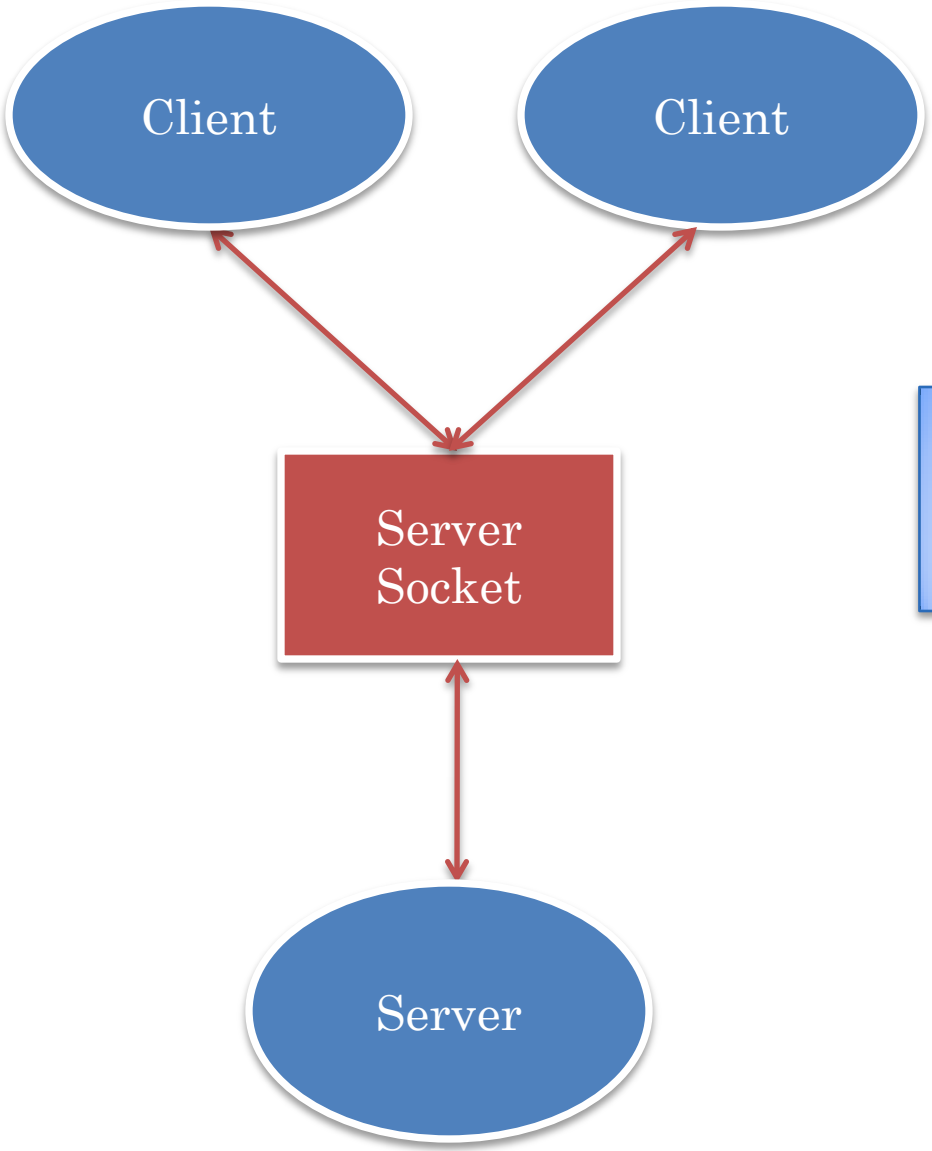
Each client also creates a simplex input port to receive replies

Address of client simplex port sent with each message

RMI Access?

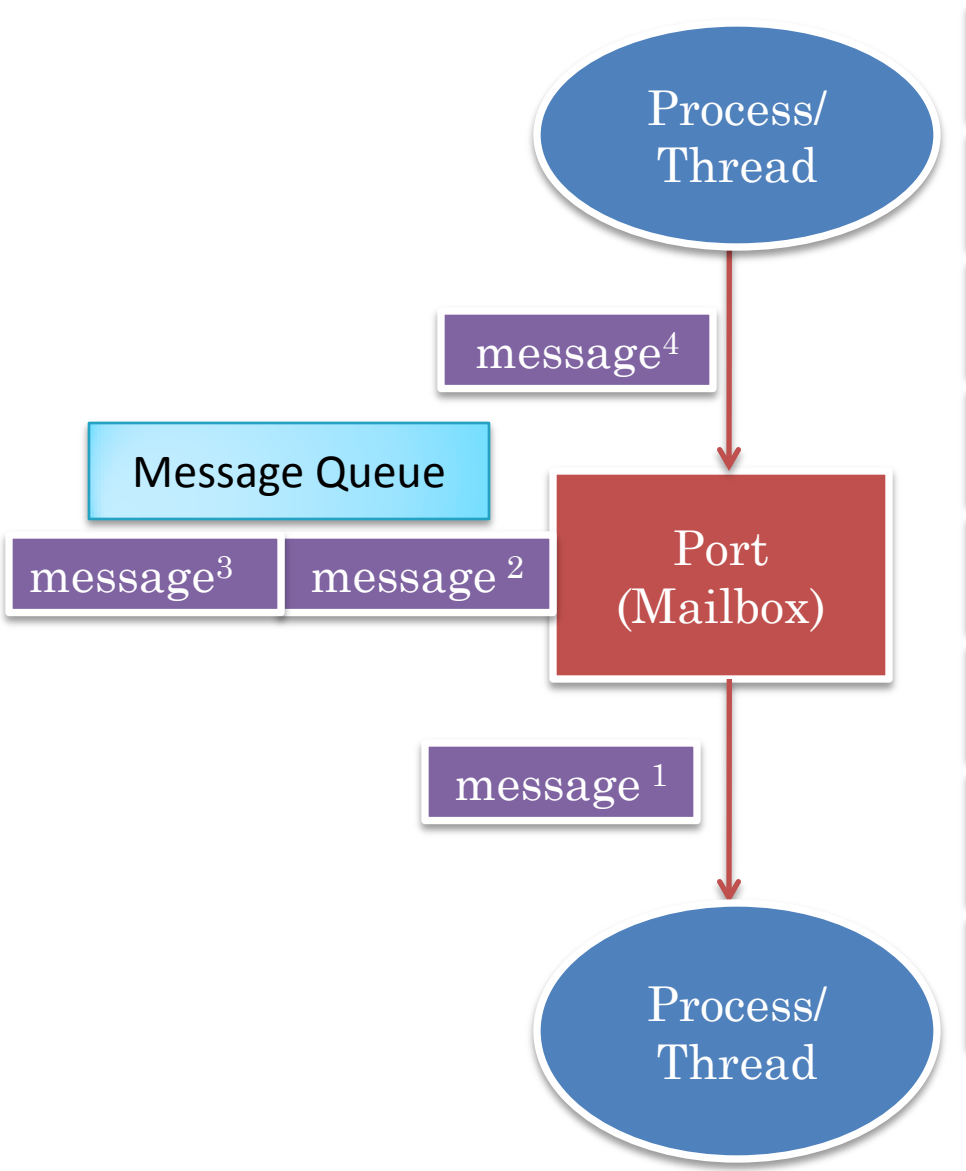


RMI: Replying Duplex Input Port



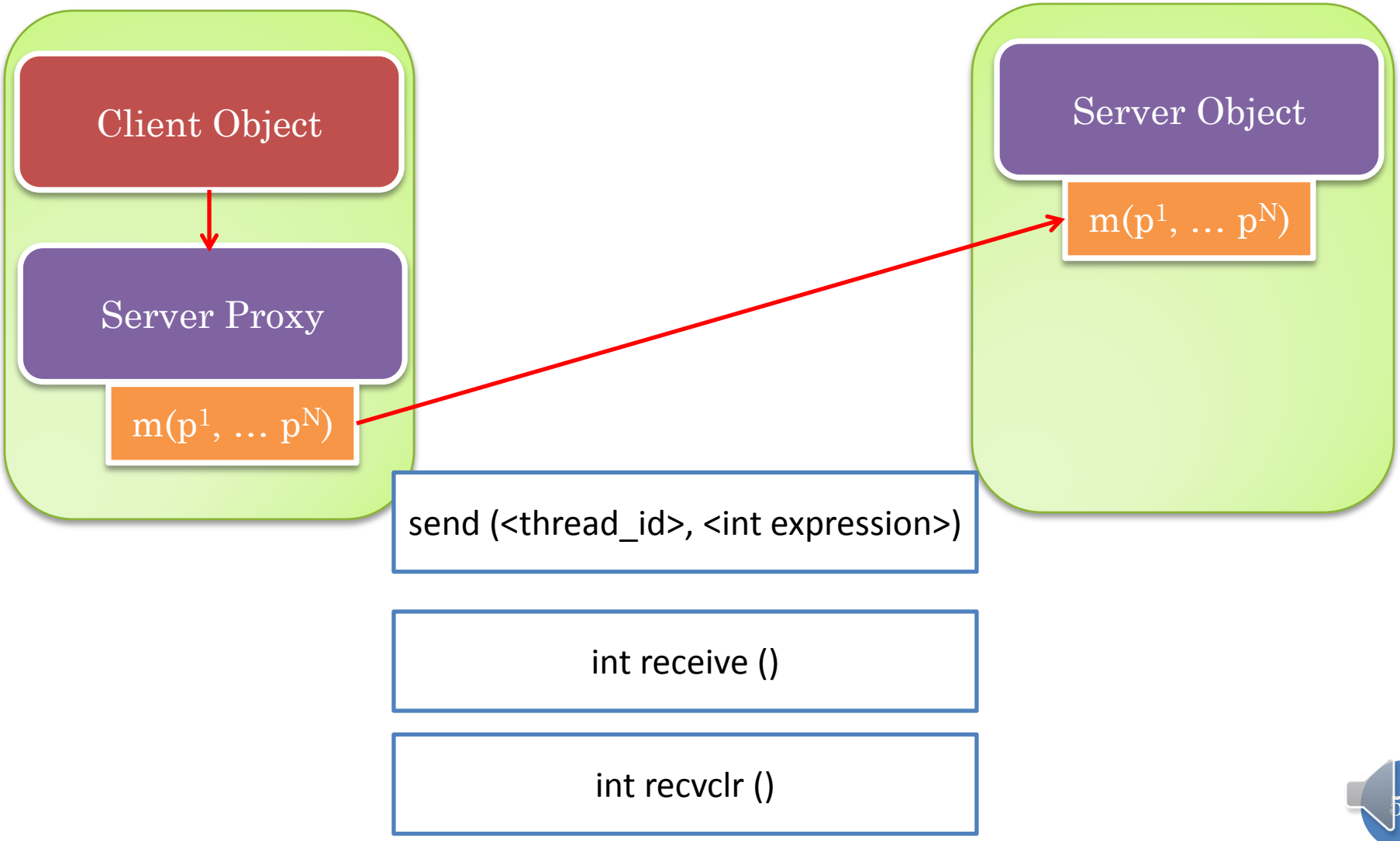
Can imagine simplex, bound, output, free port semantics for RMI

Issues in Message Passing

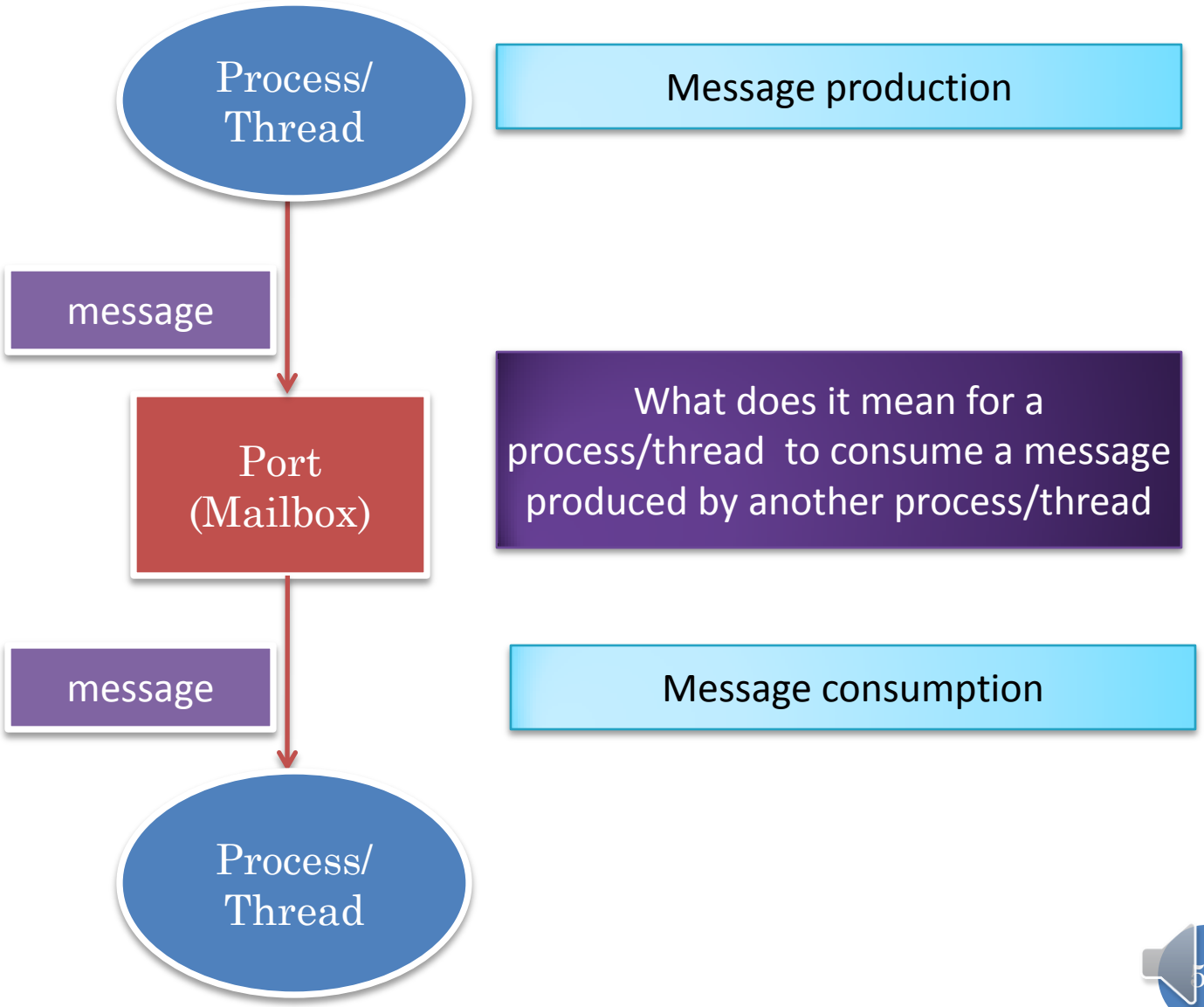


- Reliable message delivery?
- In-order message delivery?
- Port access?
- Semantics?
- Synchronous vs asynchronous ?
- Blocking vs non blocking?
- Buffering of messages at queue?
- Location of communicating threads?

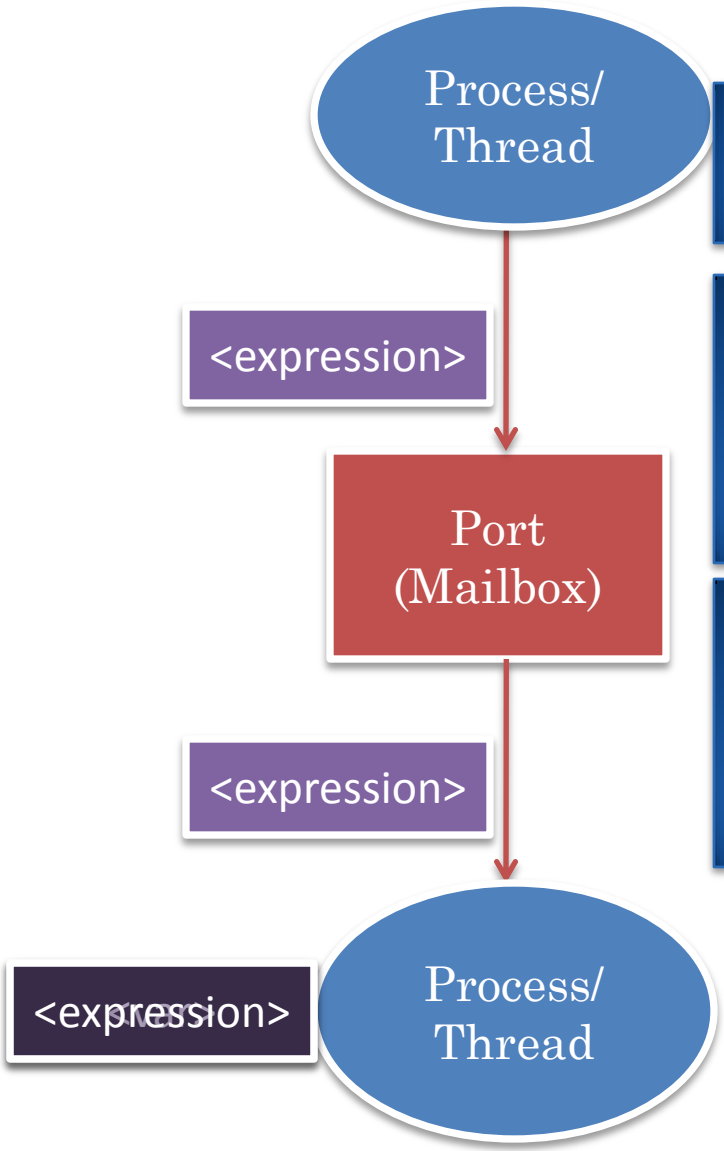
RMI vs. Xinu?


Client Object
Server Proxy
m(p1, … pN)
Server Object
m(p1, … pN)
send (<thread_id>, <int expression>)
int receive ()
int recvclr ()

Send Semantics?



Remote Assignment (Data Communication)



Message is data to be remotely assigned to a variable

The result of send is to assign a sent expression to be assigned to some variable (typically called buffer) in the remote process/thread

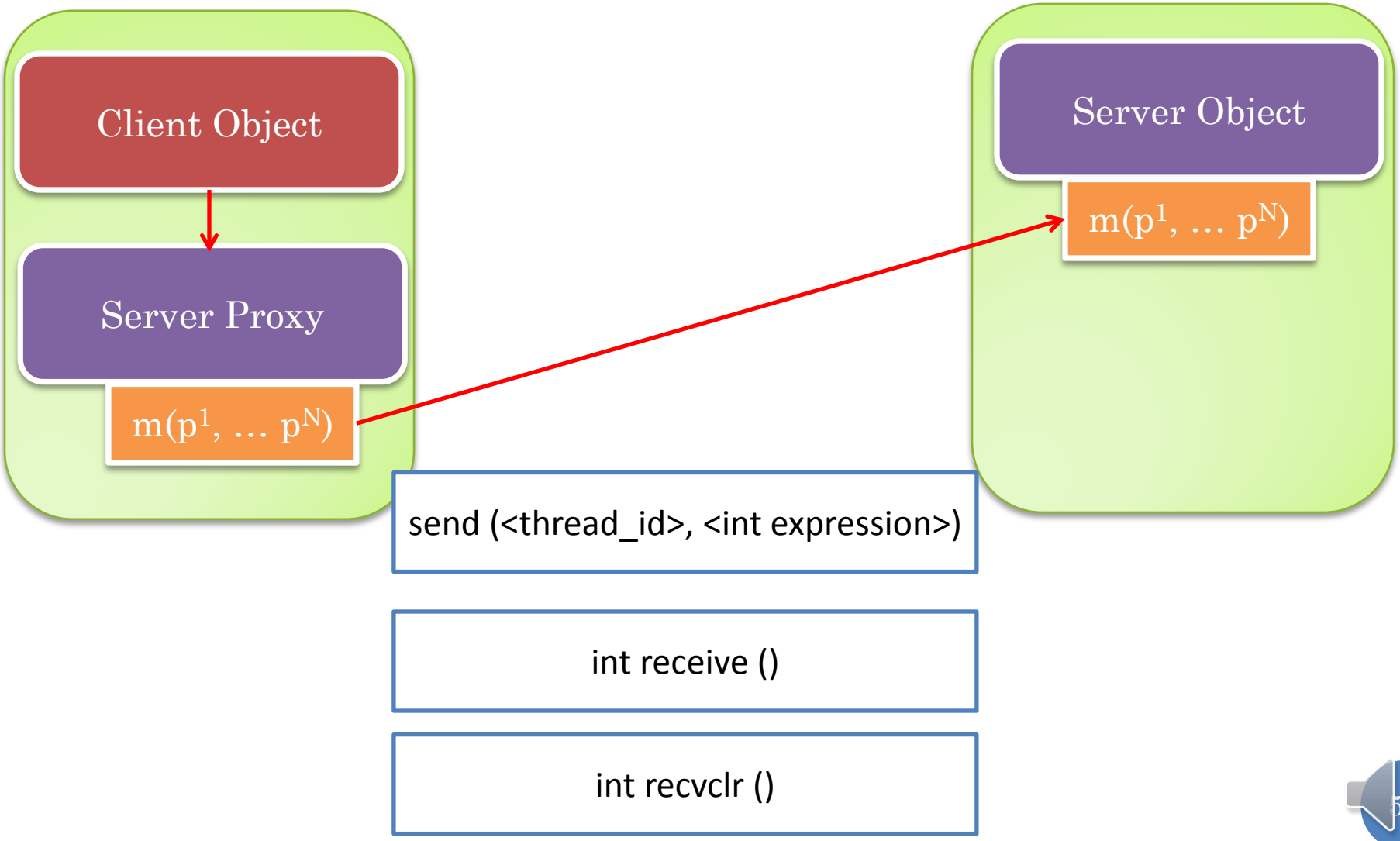
Type of expression depends on mechanisms and whether language support is provided to define primitives

P!<Expression>

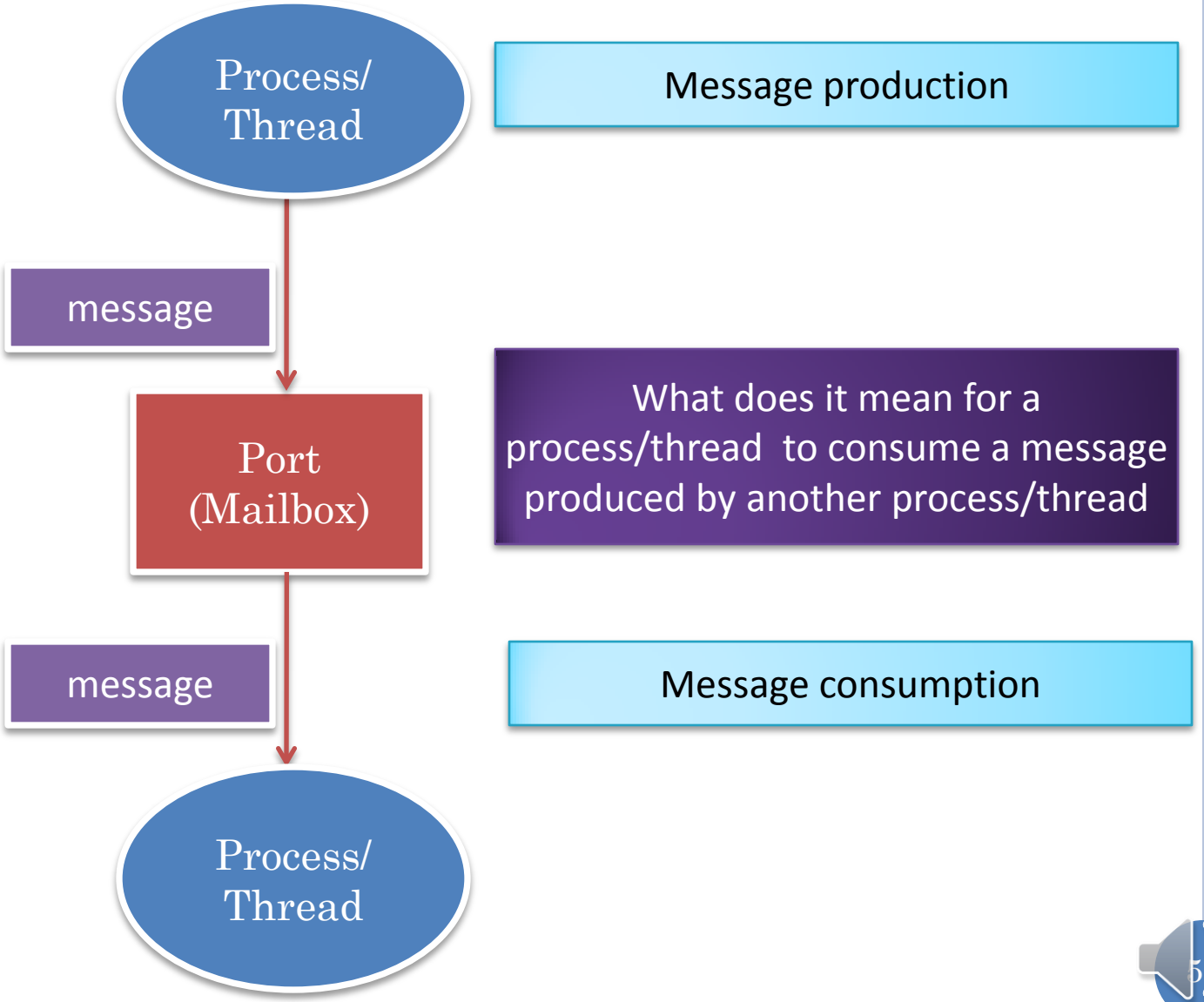
Q?<Var>

CSP (Hoare)

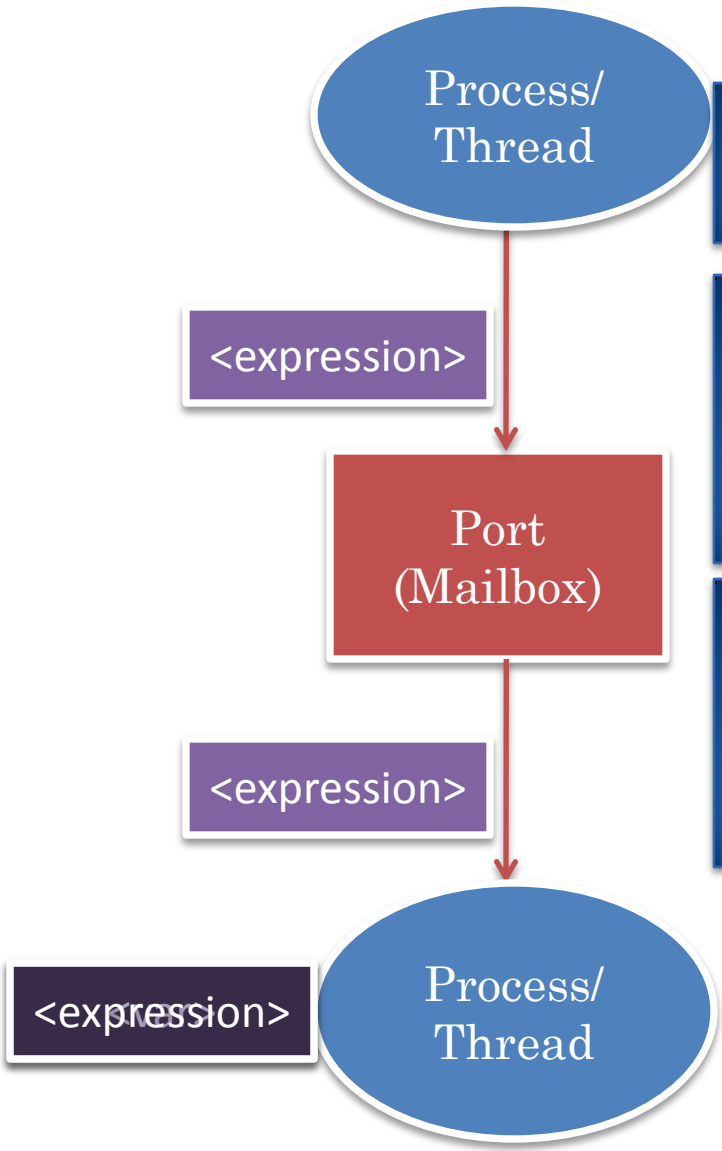
RMI vs. Xinu? (Review)



Send Semantics? (Review)



Remote Assignment (Review)



Message is data to be remotely assigned to a variable

The result of send is to assign a sent expression to be assigned to some variable (typically called buffer) in the remote process/thread

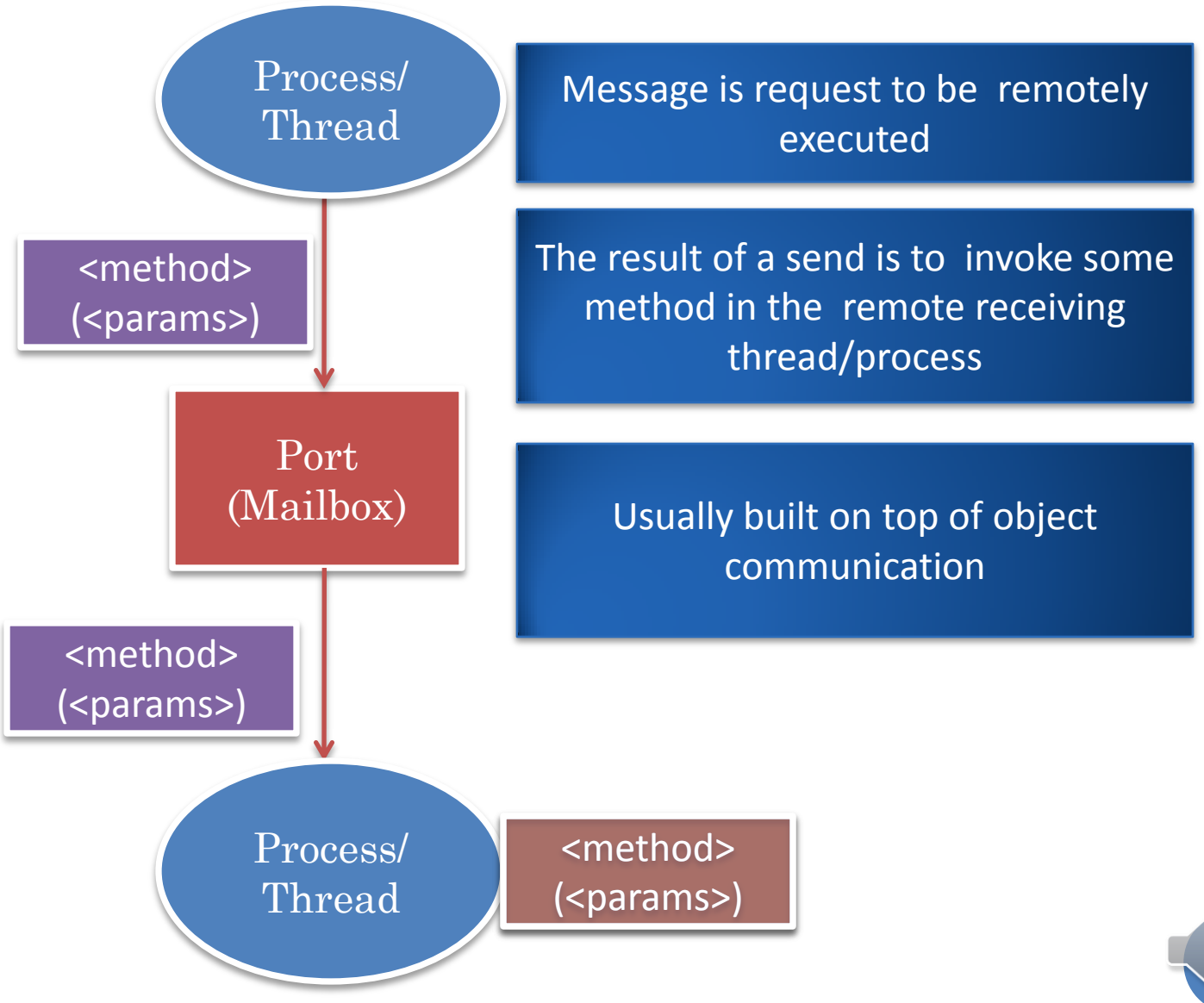
Type of expression depends on mechanisms and whether language support is provided to define primitives

P!<Exprerssion>

Q?<Var>

CSP (Hoare)

Remote Procedure Call (RPC)



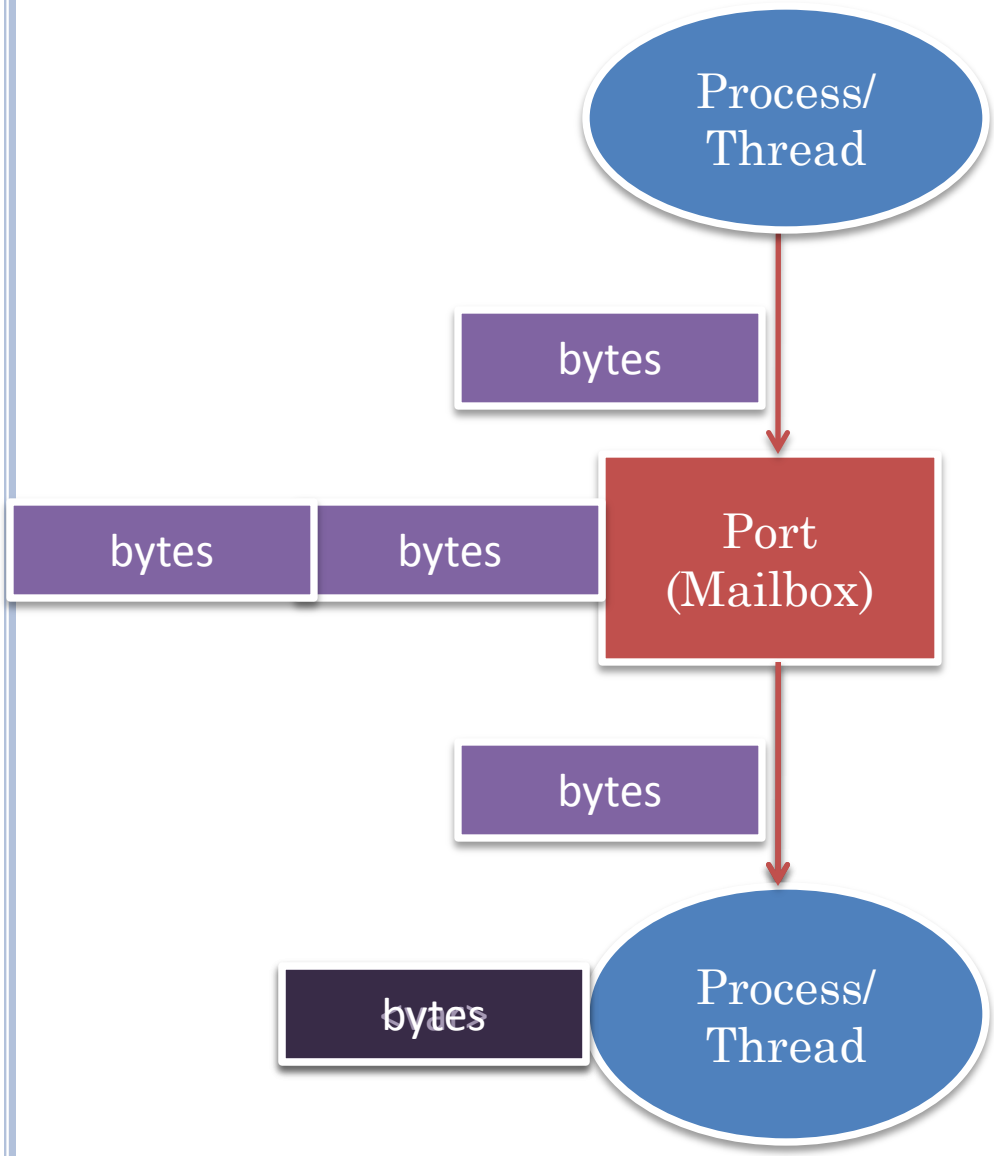
Variations of Remote Assignment

Socket/NIO Stream

Socket/NIO Datagram

Socket Object Stream

Byte Data Communication

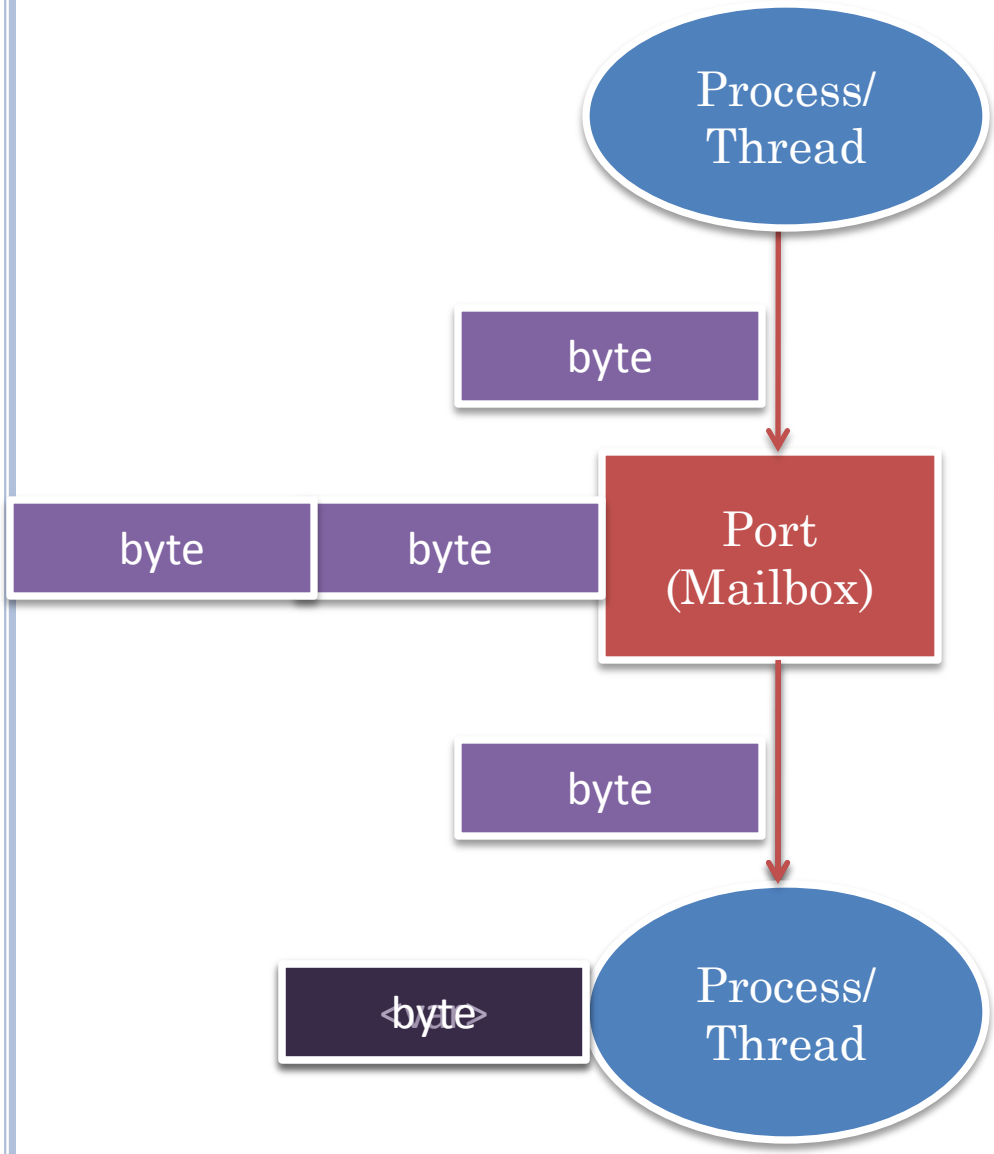


Provided by OS interface

Assume expression is a byte sequence

It is assigned to a variable that can hold such a sequence

Byte Communication Communication



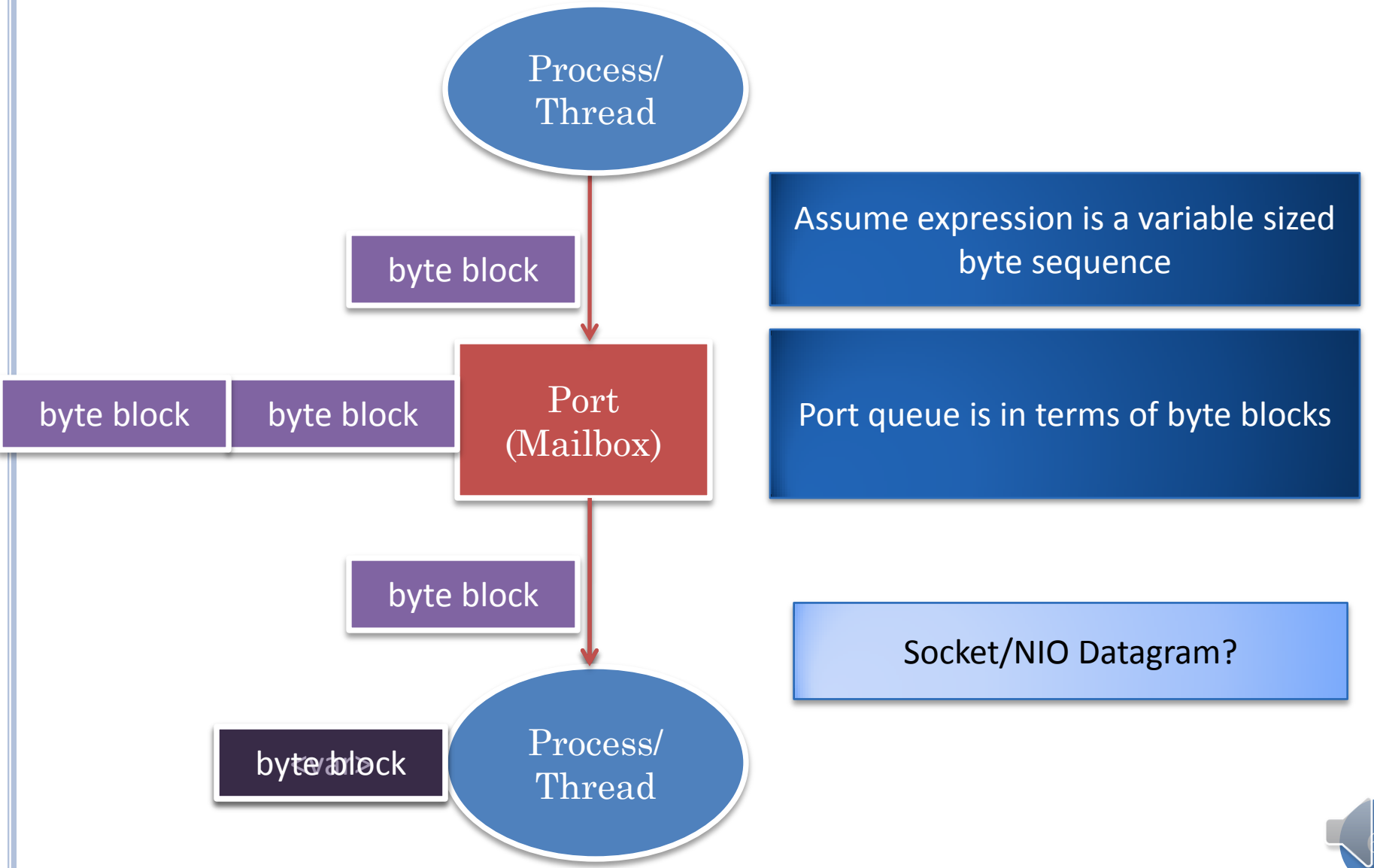
Assume expression is a single byte

Byte blocks can be communicated through libraries

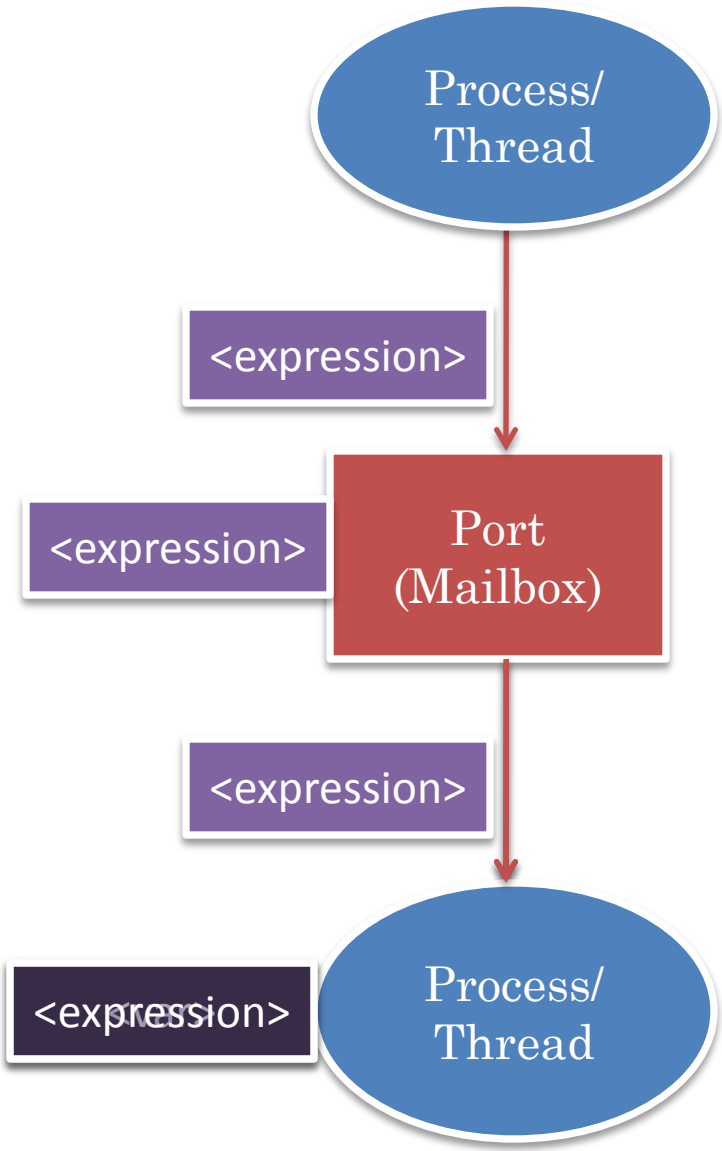
These blocks are not known to port queue

Socket/NIO Stream

Block Communication



Object Data Communication



Provided by OO Language

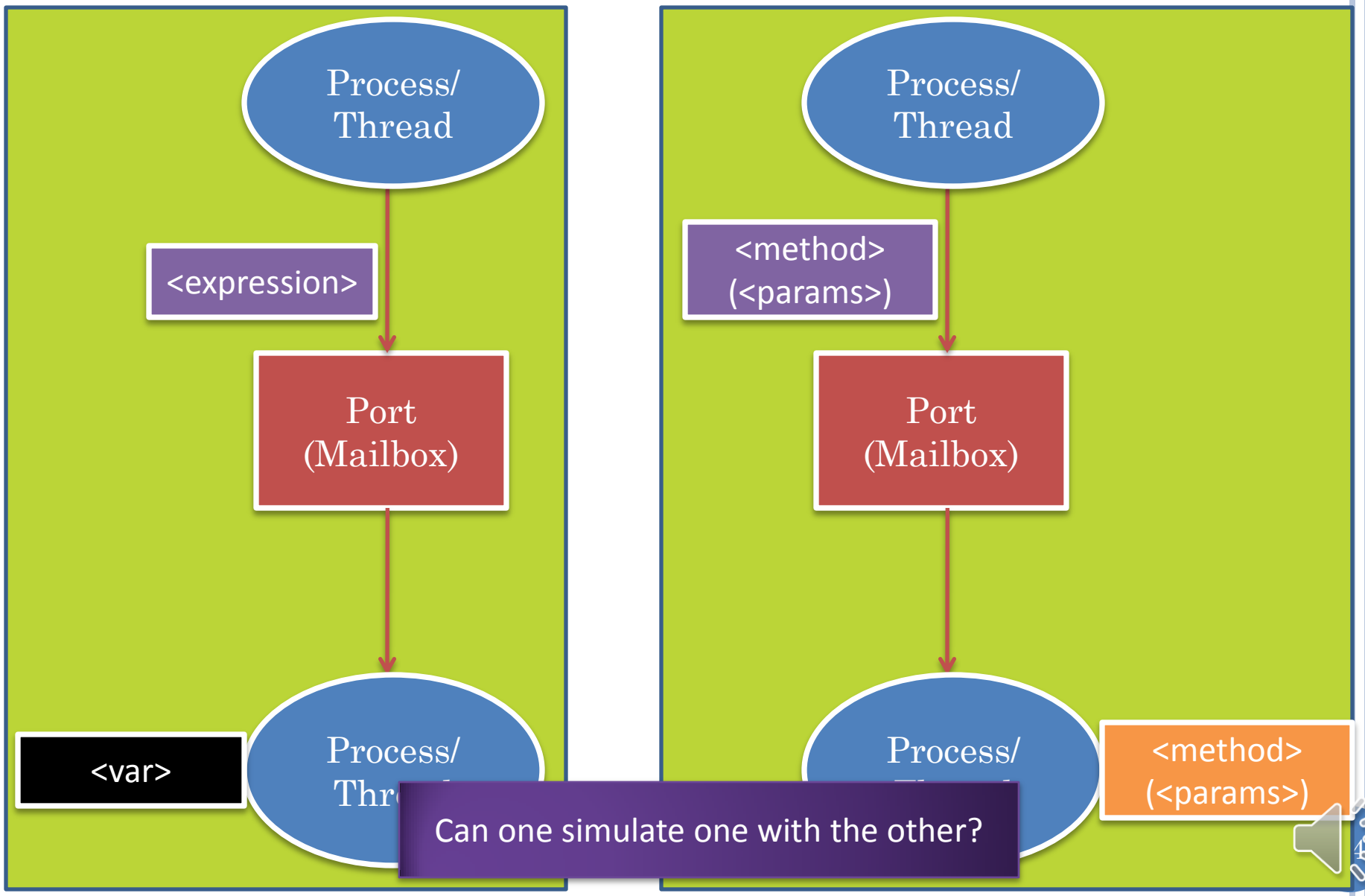
Assume expression is a location-independent object

It is assigned to a variable that can hold such an object

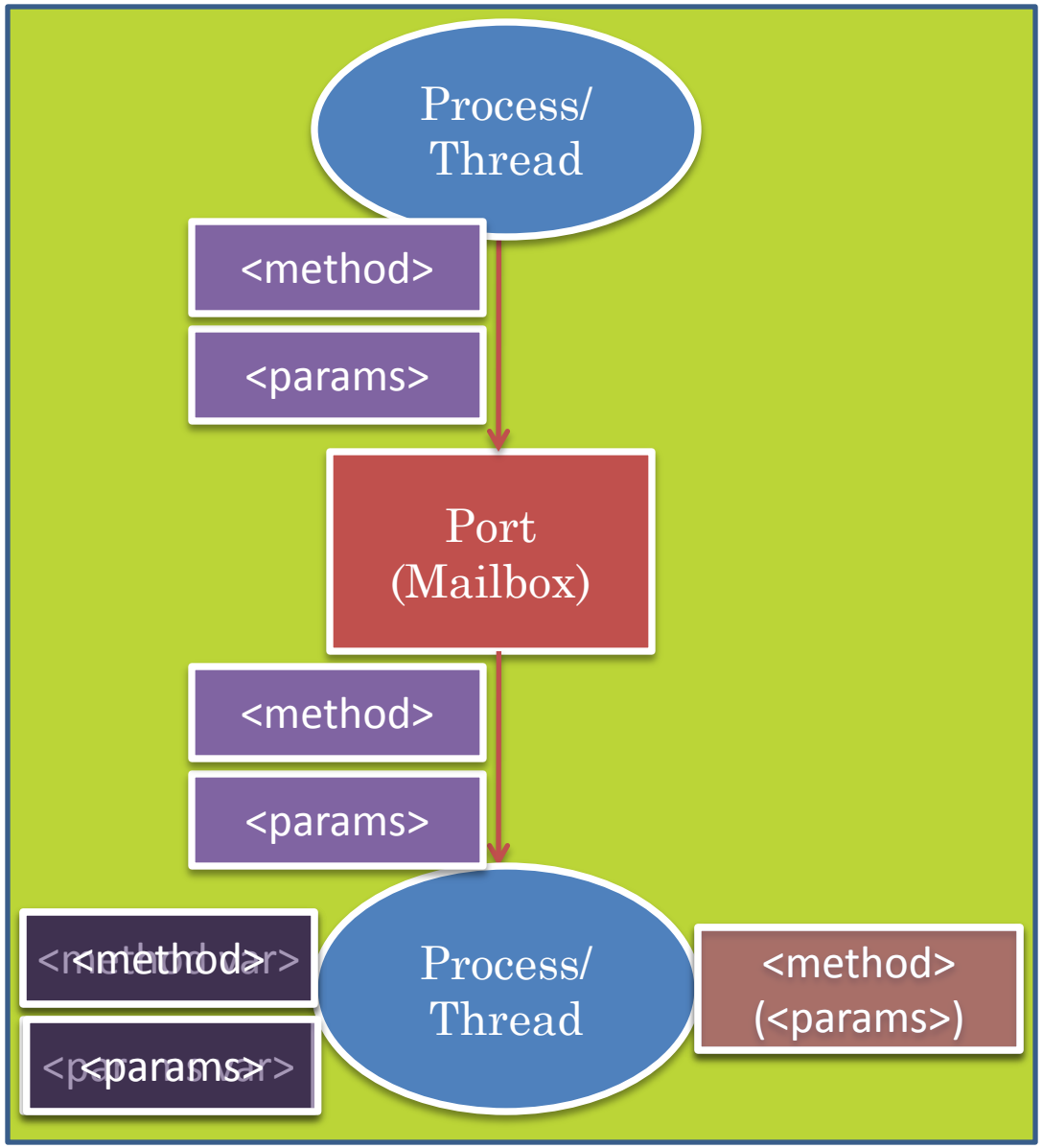
Typically implemented on top of stream or block communication

Typically port queue does not understand objects

Remote Assignment vs. Procedure Call



Simulating Remote Procedure Call



Goal is to call a method with params

Send expressions encoding method and parameters

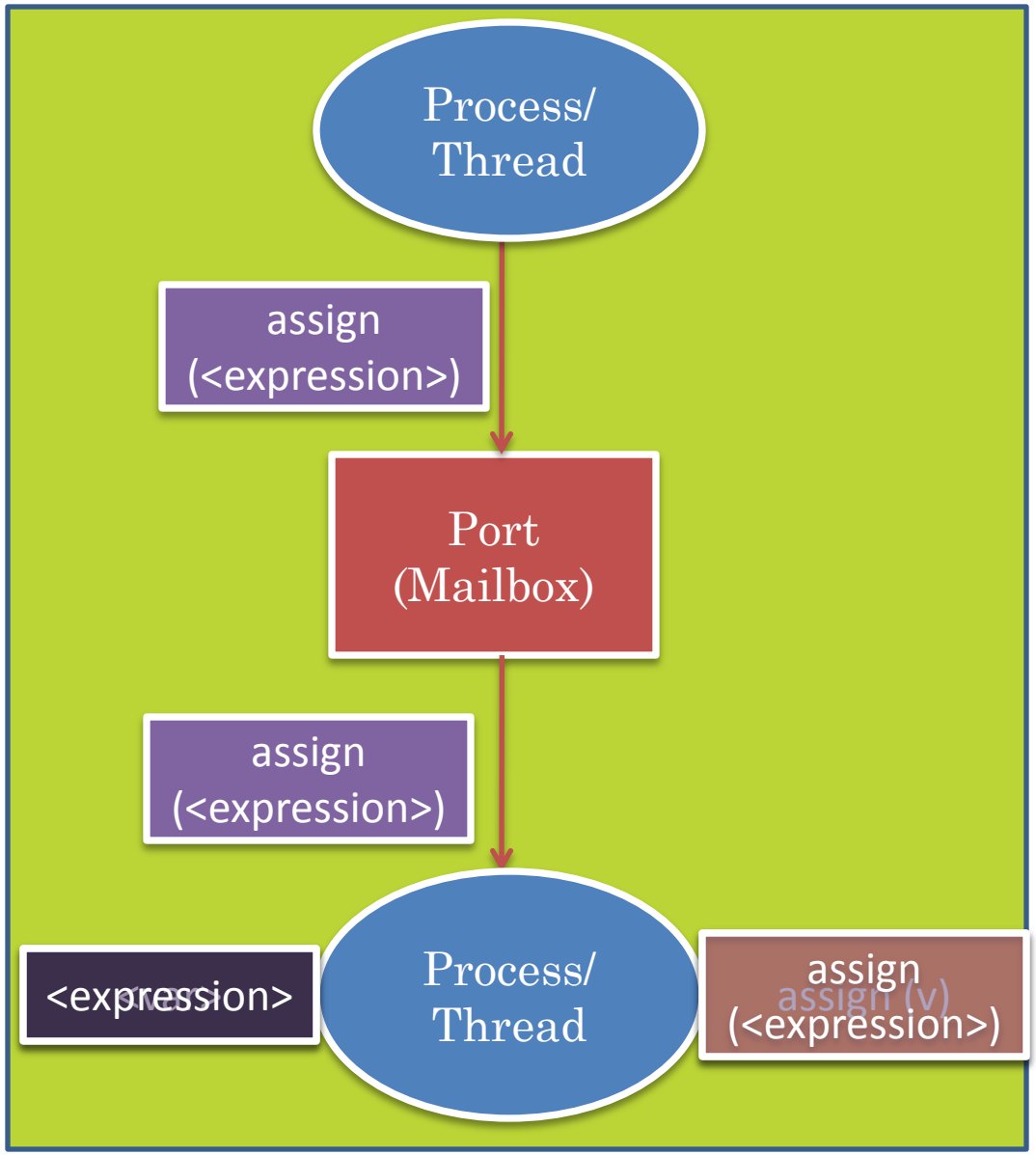
These are assigned to corresponding remote variables

Side effect of assignment is to call method with parameters

Efficiency?

Several simulating calls for one simulated call – less efficient if OS involved

Simulating Remote Assignment



Goal is to assign some expression to a buffer variable

Define special assign method with single formal parameter

Call this method with the expression to be assigned to the buffer variable

Side effect of method call is to assign actual parameter to buffer variable

Efficiency?

More expensive RPC to do RA.

Remote Procedure Call vs. Remote Assignment

Suitable when requests must be serviced

E.g. join a session

Can simulate remote assignment

Simulation awkward and not efficient

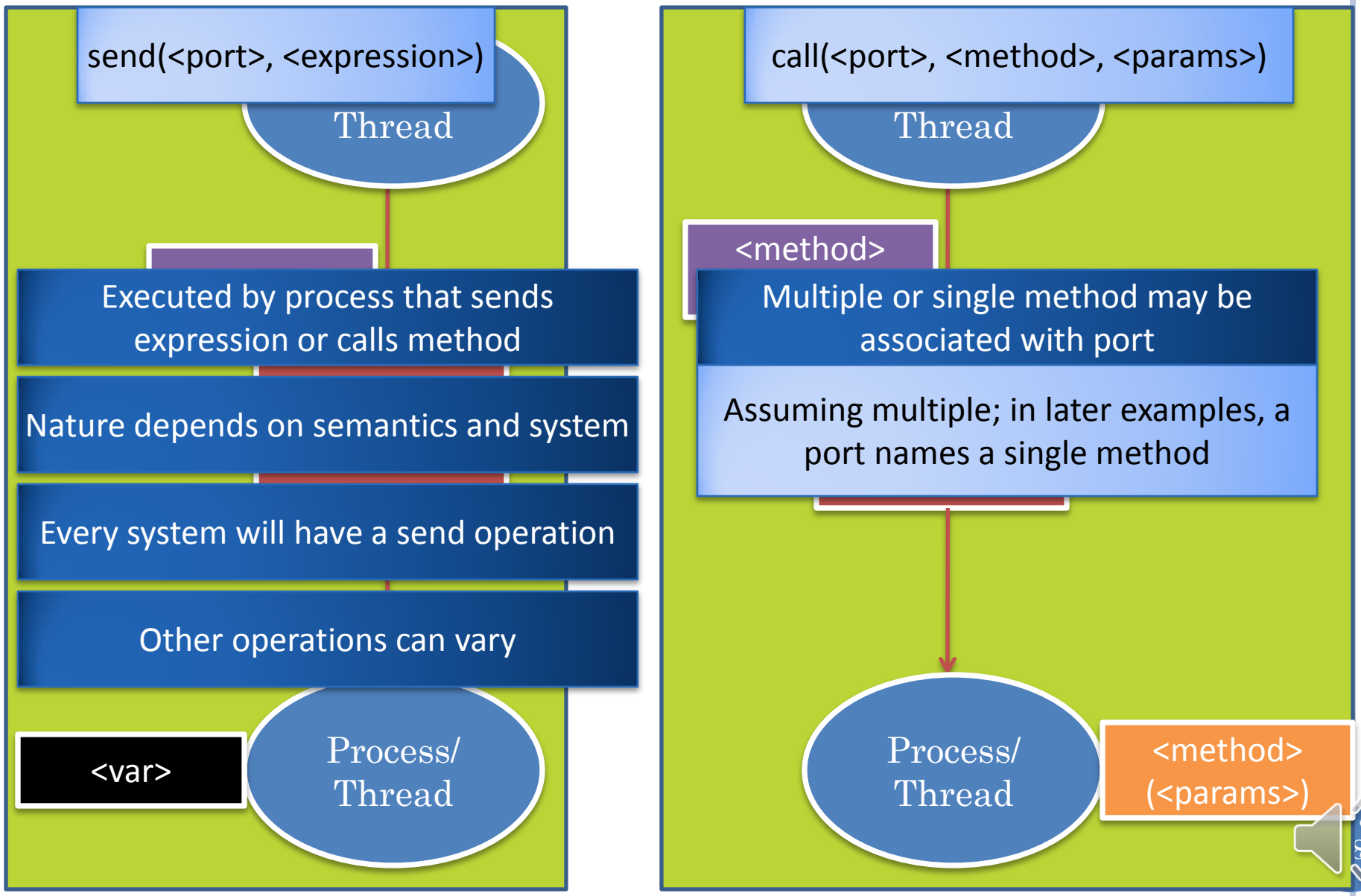
Remote assignment suitable when data must be sent

E.g. Next command in simulation

Remote assignment can implement RPC

However, simulation awkward and not as efficient

Operations?


send(<port>, <expression>)
Thread
Executed by process that sends expression or calls method
Nature depends on semantics and system
Every system will have a send operation
Other operations can vary
<var>
Process/ Thread
call(<port>, <method>, <params>)
Thread
<method>
Multiple or single method may be associated with port
Assuming multiple; in later examples, a port names a single method
Process/ Thread
<method> (<params>)

Language/Compiler Support: Type Checking and Special Syntax

No language support

send(<port>, <expression>)

call(<port>, <method>, <params>)

Allows typed <expression> of arbitrary type to be assigned to <var> of arbitrary type

Allows safe method calls, where name is port is associated with (one or more) signatures

send <port>(<typed expression>)

send load(5.2)

send echoer("hello")

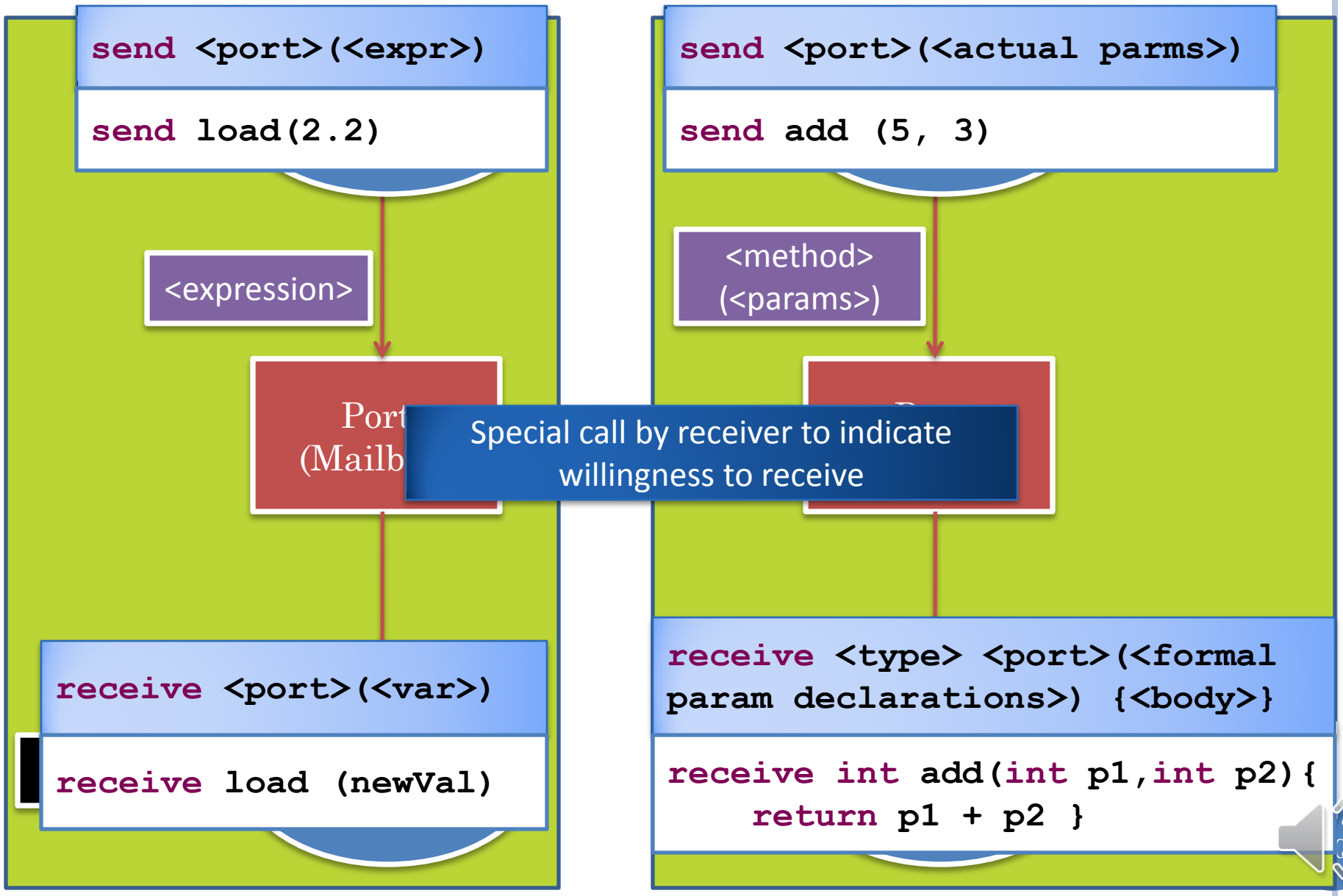
send <port/method>
(<actual params>);

send add(5,3);

send open("rpc.doc");

In example, single signature, add(int, int), associated with port, and procedure name is port name

Explicit Receive (Language Support)



Implicit Receive (Language Support)

```
send <port>(<expr>)
send load(2.2)
```

<expression>

```
register <port>(<var>)
register load (newVal)
```

```
send <port>(<actual parms>)
send add (5, 3)
```

<method>

```
register <type> <port>(<formal
param declarations>) {<body>}
register int add(int p1,int
p2){
    return p1 + p2 }
```

System provides way to automatically assign some known matching variable or call some known matching procedure when message arrives

These could have been registered by a special register call

Reply?

```
send <port>(<expr>)
```

```
send <port>(<actual parms>)
```

```
reply (<expr>)
```

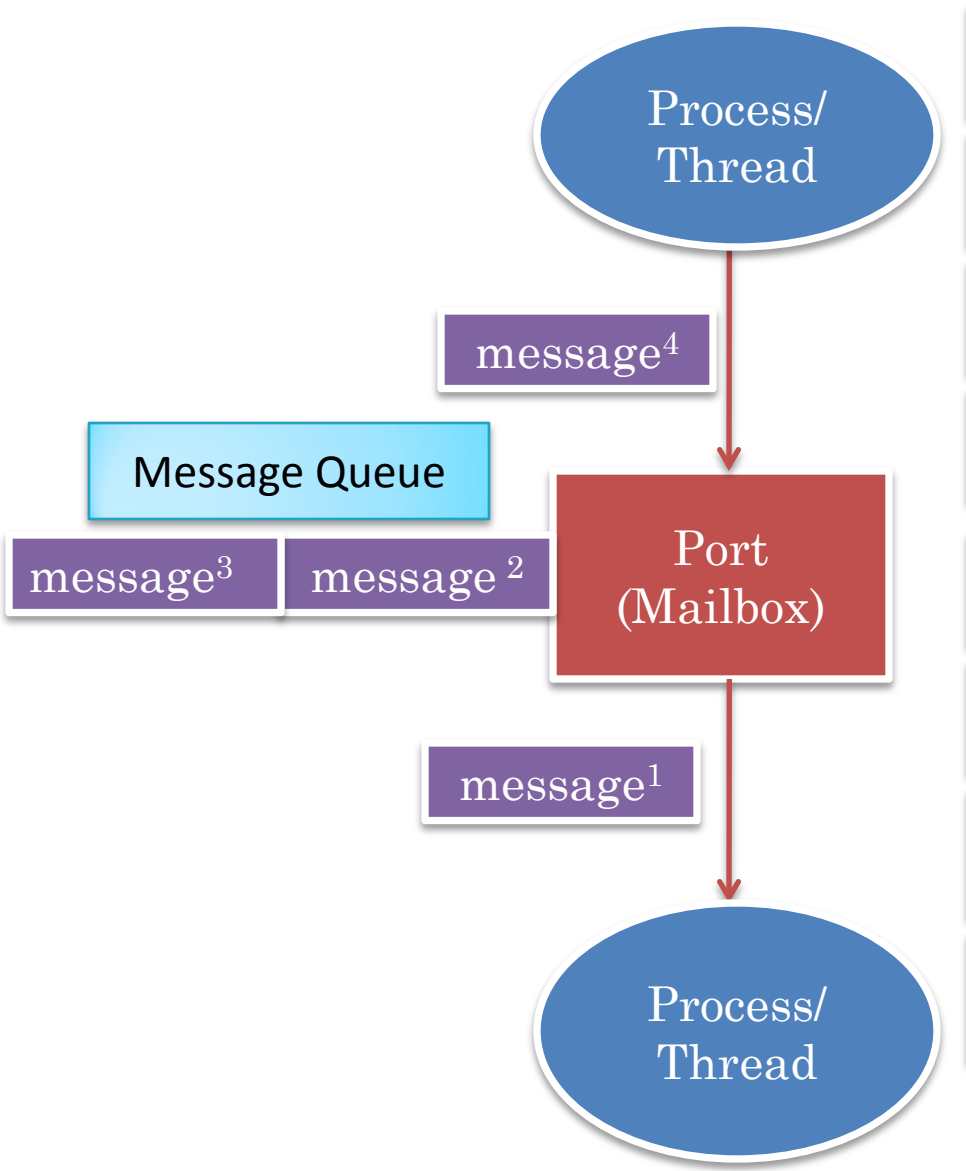
```
reply (<actual parms>)
```

Syntax and semantics?

Last sender is implicit port

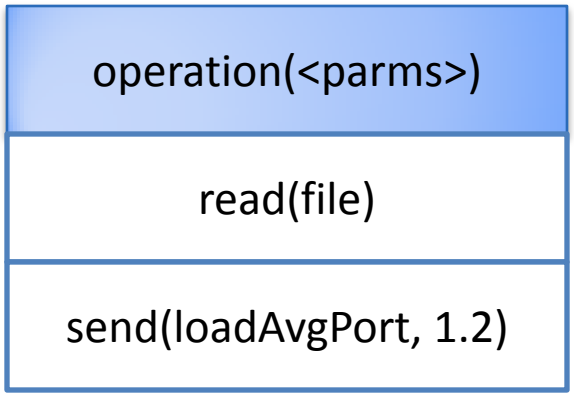
Could reply multiple times according to this definition

Issues in Message Passing



- Reliable message delivery?
- In-order message delivery?
- Port access?
- Semantics?
- Synchronous vs asynchronous ?
- Blocking vs non blocking?
- Buffering of messages at queue?
- Location of communicating threads?

Synchronous vs. Asynchronous



Synchronous: Operation invoker waits until the operation finishes

Asynchronous: Operation invoker does not wait until completion

Some other operation (e.g. software interrupt) needed to wait for result or completion status

Synchronous vs. Asynchronous vs. Blocking Operations

operation(<parms>)
read(file)
send(loadAvgPort, 1.2)

Blocking: Operation invoker waits, unblocking possibly before, until, or after operation completion
Synchronous is always blocking
Blocking is not always synchronous

Logical blocking times?

Synchronous vs. Asynchronous vs. Blocking Operations (Review)

operation(<parms>)
read(file)
send(loadAvgPort, 1.2)

Blocking: Operation invoker waits, unblocking possibly before, until, or after operation completion
Synchronous is always blocking
Blocking is not always synchronous

Logical blocking times?

Blocking (Logical) Times? A La Binding Time

Binding time: When is some property bound to an entity?

E.g. when variable bound to to an address, value

Program writing, compilation, link, load, runtime

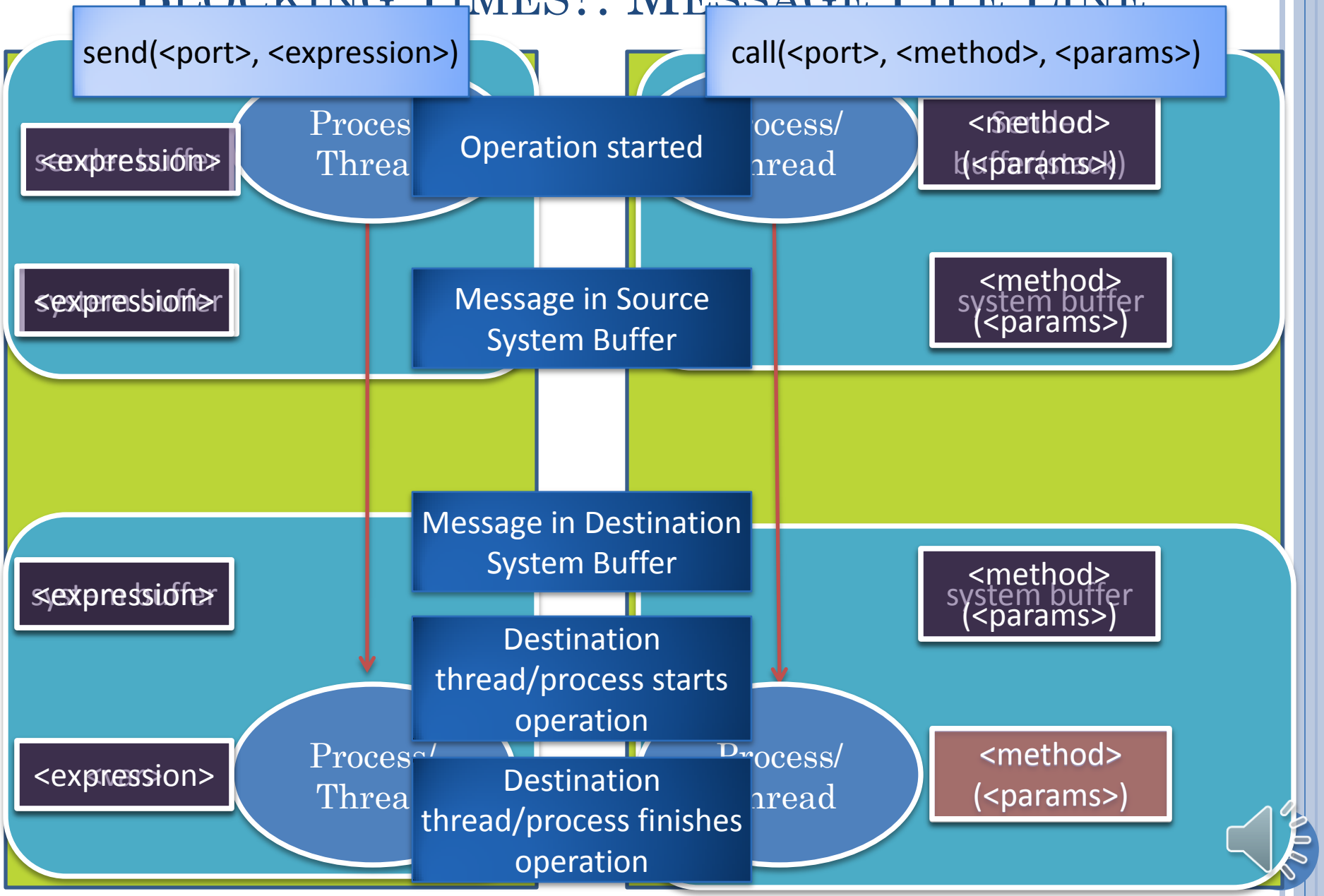
Tied to phases in the lifetime of a program

Logical times for blocking?

Phases in the lifetime of a message?

Communication pipe line?

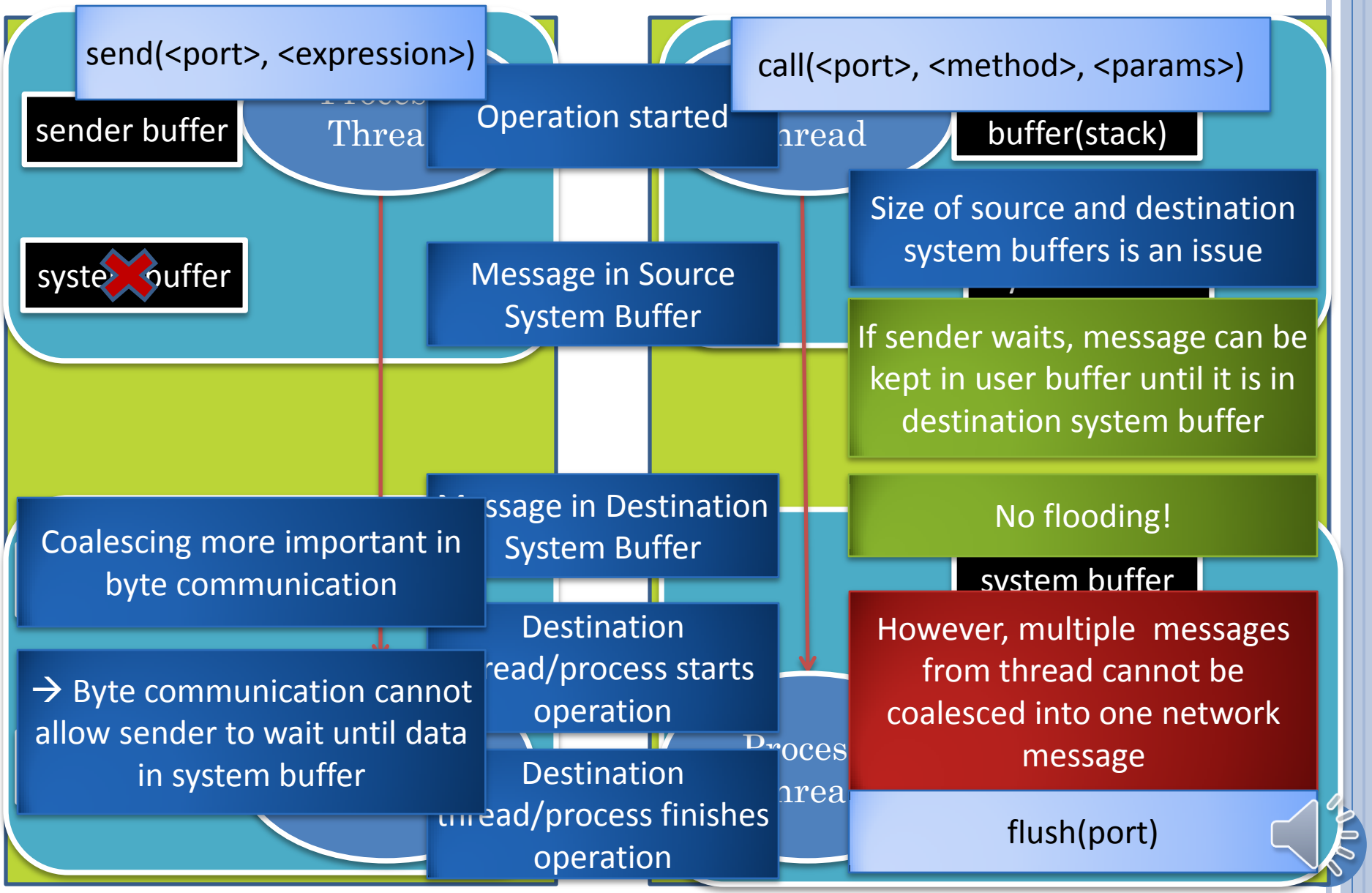
Blocking Times?: Message Pipe Line



Blocking Times Pros

Operation started	No waiting, most concurrency (without using extra thread)	Sending load average
Message in Source System Buffer	Waits until buffer available, prevents flooding	Sending tele-pointers
Message in Destination System Buffer	Sender knows message did not get lost in network	Sending load average on unreliable port
Destination thread/process starts operation	Sender knows receiving process did not fail or ignore message	Remote animation started
Destination thread/process finishes operation	Sender knows operation finished	Reservation made by airline, file received by Dropbox

Late Blocking and Buffering



Using Sender Buffer

Async NIO has direct buffer to prevent copying

Programmer can use select to determine when buffer is available

Synchronous vs. Asynchronous Receive (Semantics)

```
receive <port>(<var>)
```

```
receive <type> <port>(<formal param declarations>) {<body>}
```

Synchronous: Operation invoker waits until the operation finishes

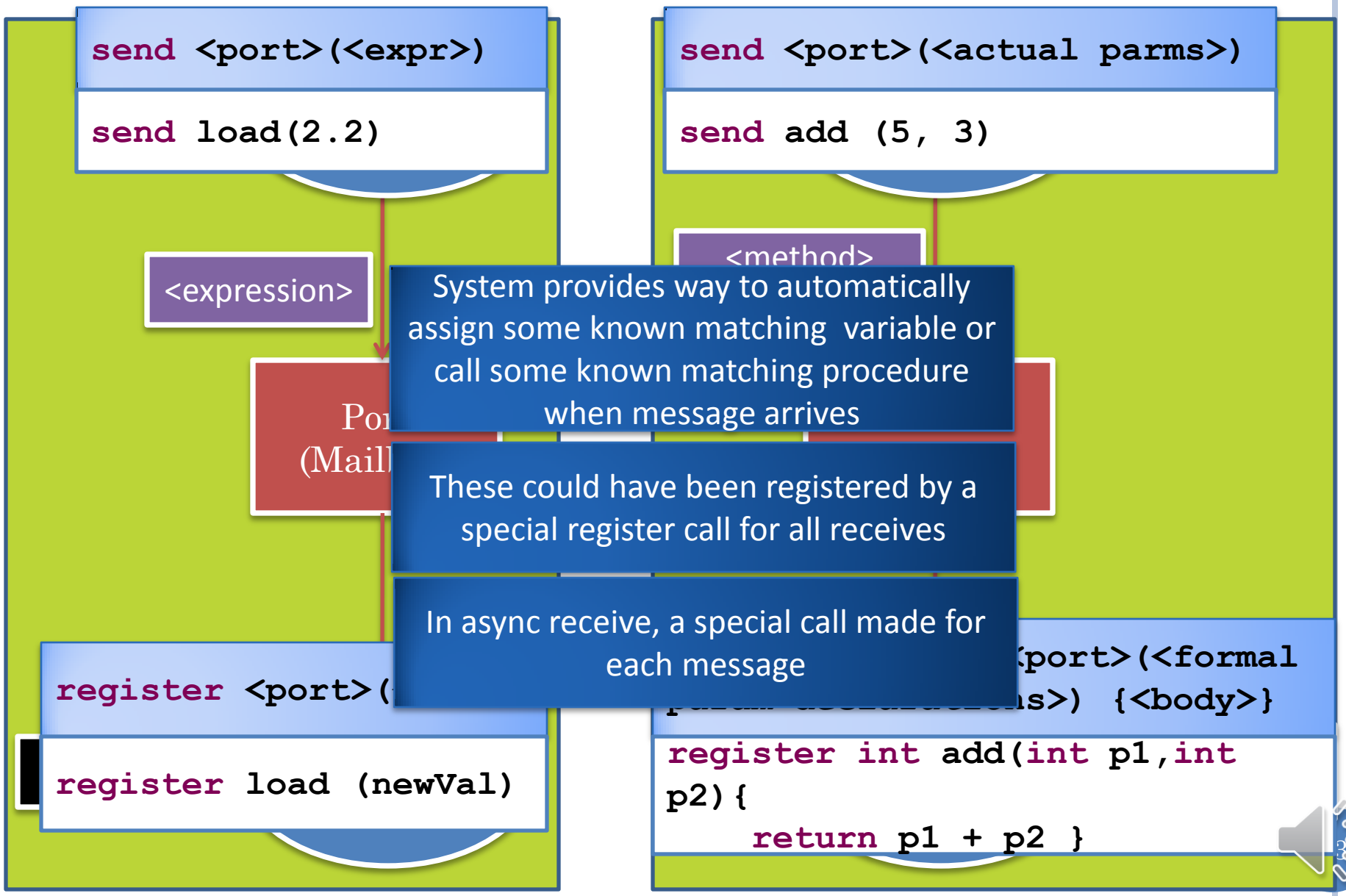
Synchronous receive: Receive blocks until remote assignment or procedure call finishes

Asynchronous: Operation invoker initiates operation and does not wait until completion

Asynchronous receive: Receive provides buffers to receive <expression> or <method> call

Some other operation (e.g. software interrupt) needed to wait for result or completion status

Implicit vs. Async Receive (Semantics)?



Blocking Times in Sockets

```
socket.connect(
 new InetSocketAddress(host,port));
```

Block until server accepts connection to server socket

```
Socket socket =
    serverSocket.accept();
```

Block until next client tries to contact the server socket

```
outputStream = socket.getOutputStream();
outputStream.write(buf, offset, length);
```

Block until in system buffer

```
inputStream = socket.getInputStream();
int retVal =
    inputStream.read(buf, offset, length);
```

Block until <= length >=1 bytes received

Asynchronous RPC?

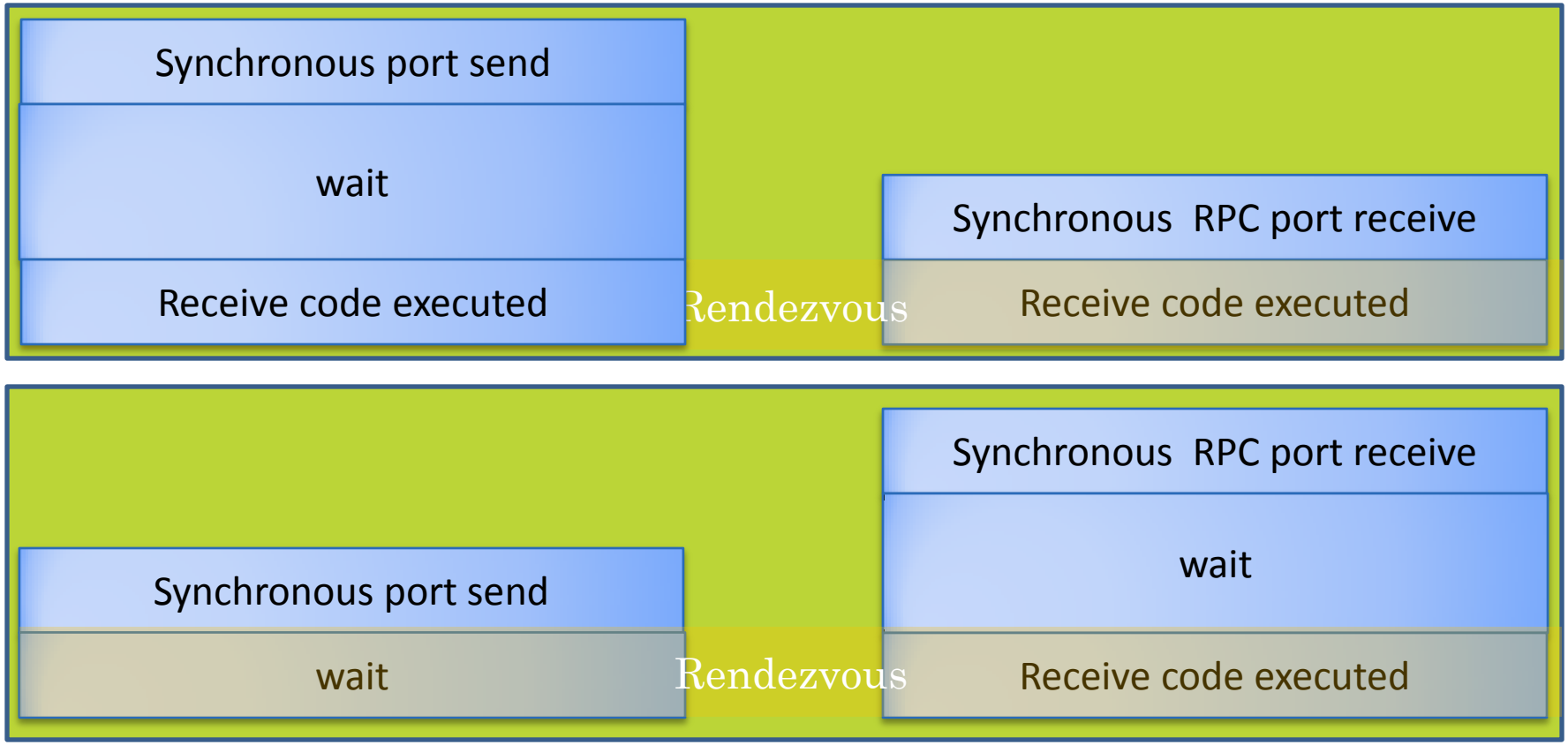
Asynchronous RPC semantics?

Starting remote thread

Example?

A remote animation

Rendezvous in Sync RPC Send, Receive



Receiver may make the sender wait

Sender may make the receiver wait

(Ada) Rendezvous: when RPC is executing – both sender and receiver are waiting

Synchronous vs. Asynchronous Receive (Pros/cons)

Sync: no need to have separate operation to determine when operation finishes

Sometimes receiver does not need notification, shared memory model

Alpha/beta search optimization parameters, when receiver next looks at them, the value may have been set

Receiver can block on a port on which a message does not arrive

Waiting on Multiple Receipts

```
receive <port1>(<var>)
```

```
receive <type1>
<port1>(<formal params>
{<body>}
```

```
receive <port1>(<var>)
```

```
receive <type1>
<port2>(<formal params>)
{<body>}
```

A single thread/process can expect messages on multiple ports

Synchronous receive → fork a thread for each receive (inefficient)

Asynchronous receive → a single thread can wait on multiple receives

New construct for single thread and sync receive?

Abstract Sync Select with Sync Receive

```
select
    receive … <port1> …
    receive … <portn> …
end
```

Select operation waits until a matching send arrives for one of the receives

It completes when the receive completes

If more than one matching send?

Pick one non deterministically

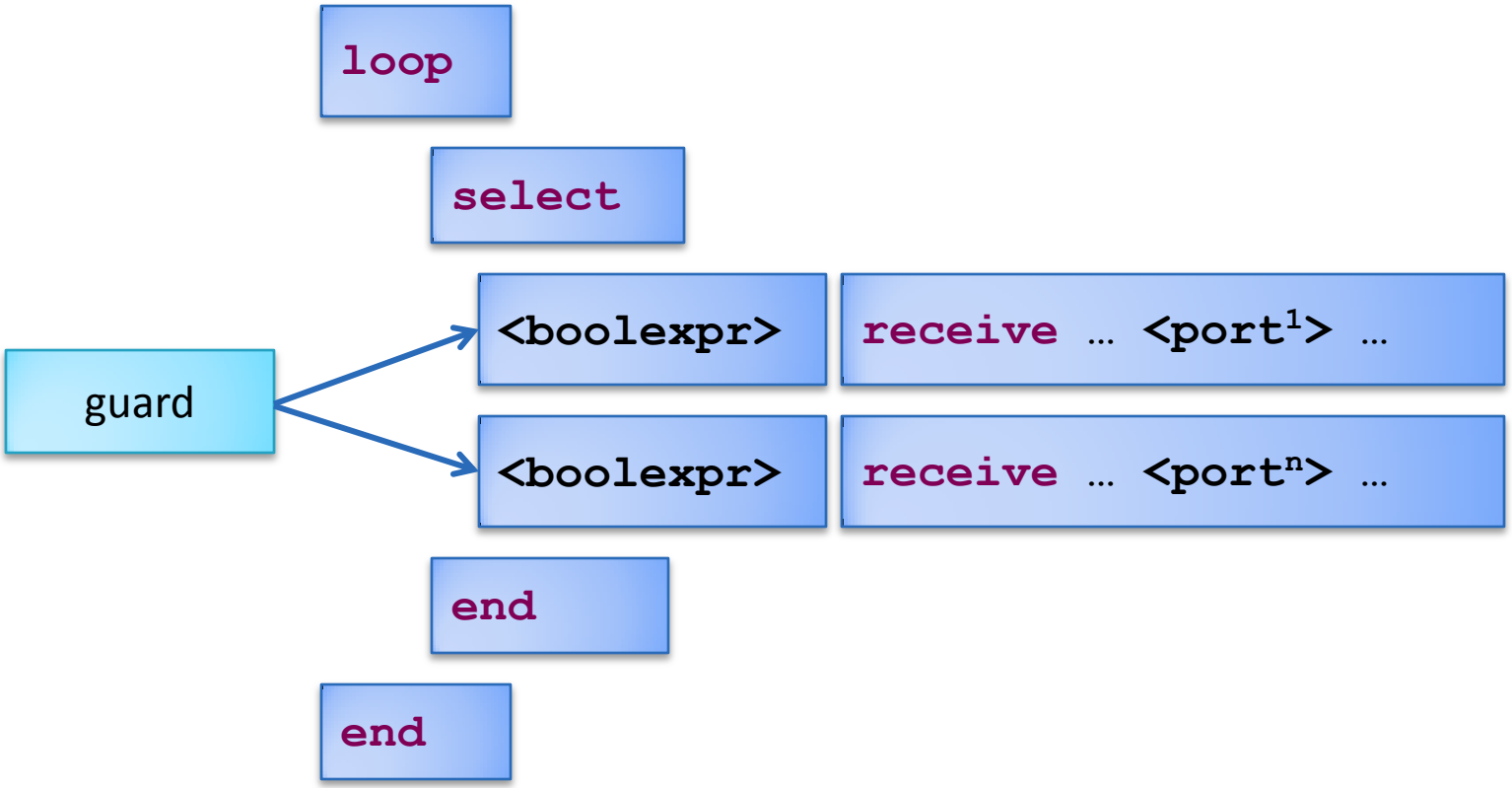
Usually Receive All Requests

```
loop
    select
        receive … <port1> …
        receive … <portn> …
    end
end
```

Select is typically in a loop

Each receive is executed atomically

Guarded Receive



A receive is matched if a matching send arrives and the guard evaluates to true

Use?

Ensure preconditions, block synchronous sender

Bounded Buffer Thread

Bounded Buffer Thread/Process

```
loop
  select  {
     when  size < MAX_SIZE:
        receive void put(ElementType element)  {
           buffer[nextIn] = element;
           nextIn = (nextIn + 1) % MAX_SIZE;
           size++;
        }
     when  size > 0:
        receive ElementType get()  {
          ElementType retVal =
            (ElementType) buffer[nextOut];
          nextOut = (nextOut + 1) % MAX_SIZE;
          size--;
          return retVal;
        }
  }
```

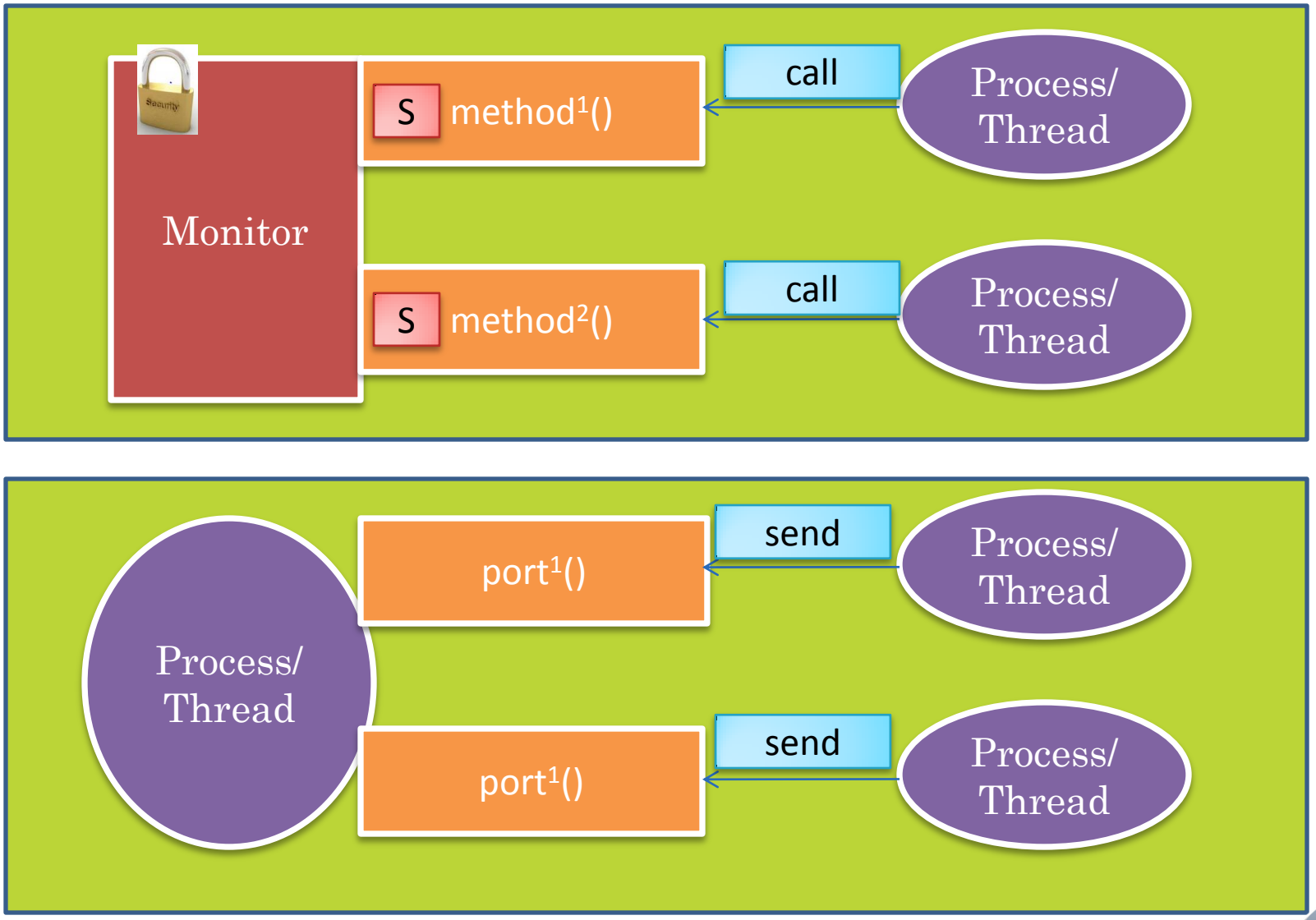
Producer

```
send put("hello");
```

Consumer

```
retVal = send get();
```

Thread/Process vs. Monitor



Shared Memory/Message Passing Duality

```
loop
  select {
     when size < MAX_SIZE:
        receive void put(ElementType element) {
           buffer[nextIn] = element;
           nextIn = (nextIn + 1) % MAX_SIZE;
           size++;
        }
     when size > 0:
        receive ElementType get() {
          ElementType retVal =
            (ElementType) buffer[nextOut];
          nextOut = (nextOut + 1) % MAX_SIZE;
          size--;
          return retVal;
        }
  }
}
```

```
public synchronized void put(ElementType element) {
  while (size >= MAX_SIZE) {
    nonFull.condWait();
  }
  buffer[nextIn] = element;
  nextIn = (nextIn + 1) % MAX_SIZE;
  size++;
  nonEmpty.condBroadcastSignal();
}
public synchronized ElementType get() {
  while (size == 0) {
    nonEmpty.condWait();
   }
  ElementType retVal =
    (ElementType) buffer[nextOut];
  nextOut = (nextOut + 1) % MAX_SIZE;
  size--;
  nonFull.condSignal()
  return retVal;
}
```

```
send put("hello");
```

```
put("hello");
```

- Synchronous RPC port receive
- Loop and select 1 receive at a time
- Guard
- Return from RPC receive
- Synchronous RPC port send

- Procedure
- Entry procedure
- Condition wait
- Signal and return
- Entry procedure call

Flexibility Comparison

```
loop
  select  {
     when size < MAX_SIZE:
        receive void put(ElementType element) {
           buffer[nextIn] = element;
           nextIn = (nextIn + 1) % MAX_SIZE;
           size++;
        }
     when size > 0:
        receive ElementType get() {
          ElementType retVal =
            (ElementType) buffer[nextOut];
          nextOut = (nextOut + 1) % MAX_SIZE;
          size--;
          return retVal;
        }
   }
}
```

```
public synchronized void put(ElementType element) {
  while (size >= MAX_SIZE) {
    nonFull.condWait();
  }
  buffer[nextIn] = element;
  nextIn = (nextIn + 1) % MAX_SIZE;
  size++;
  nonEmpty.condBroadcastSignal();
}
public synchronized ElementType get() {
  while (size == 0) {
    nonEmpty.condWait();
   }
  ElementType retVal =
    (ElementType) buffer[nextOut];
  nextOut = (nextOut + 1) % MAX_SIZE;
  size--;
  nonFull.condSignal()
  return retVal;
}
```

Guard evaluated once at start

One signal, which is a return

Multiple waits in entry procedure

Multiple signals

Do we need the extra monitor flexibility?

Probability not

Explicit Sync vs. Implicit Receive (Pros, Cons)

Can be used to make sender wait when sync send

Necessary to simulate monitors

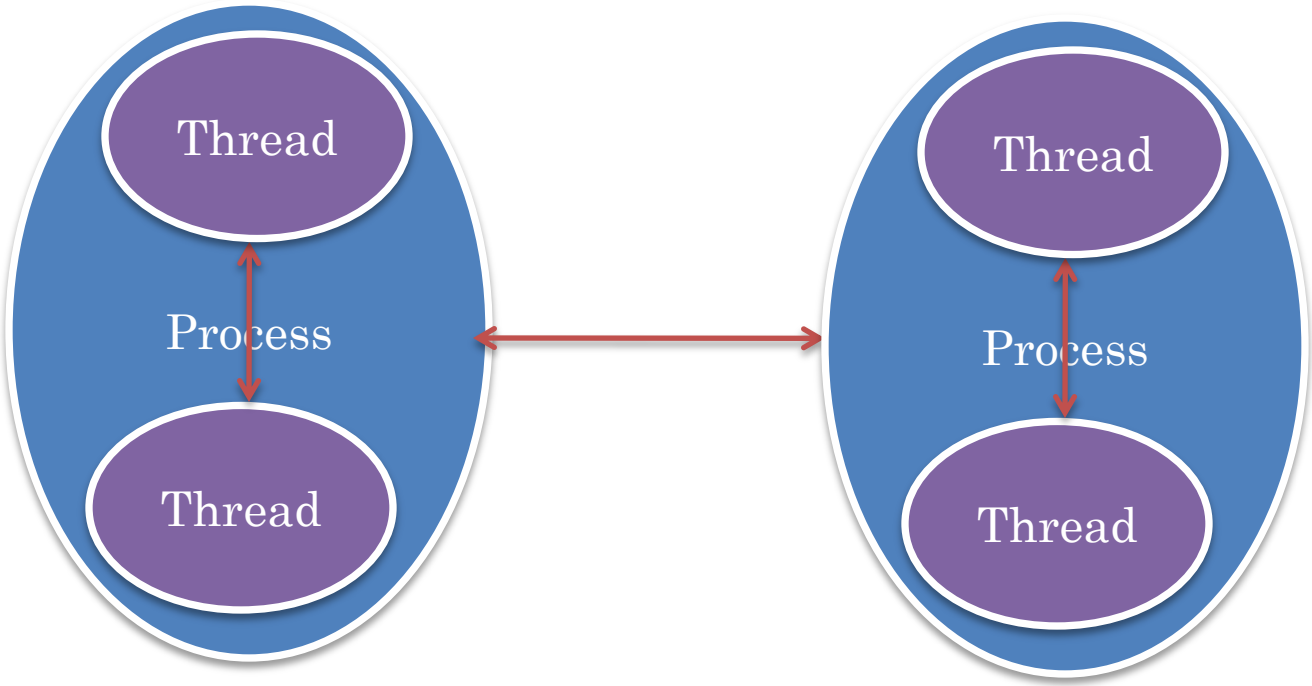
Sender has to wait

loop, select, receive needed

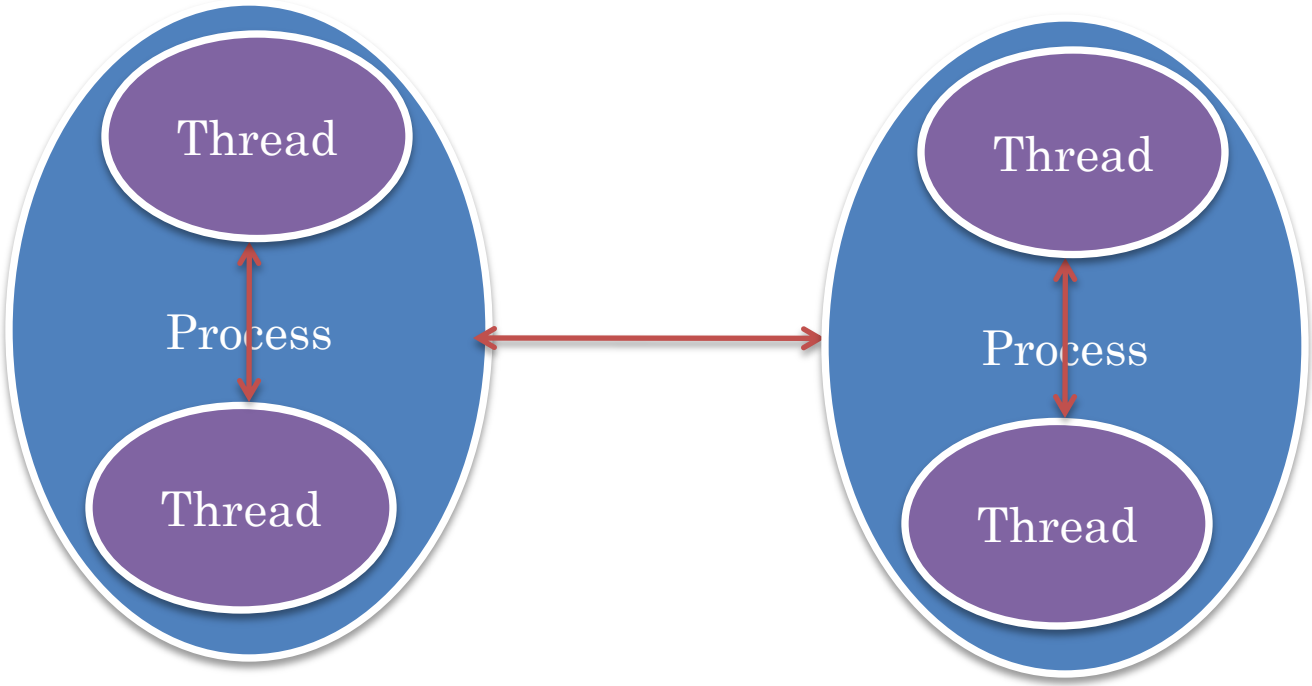
If other mechanisms available to make sender wait, then implicit receive better

Usually implicit receive for RPC, and explicit receive for remote assignment (data communication)

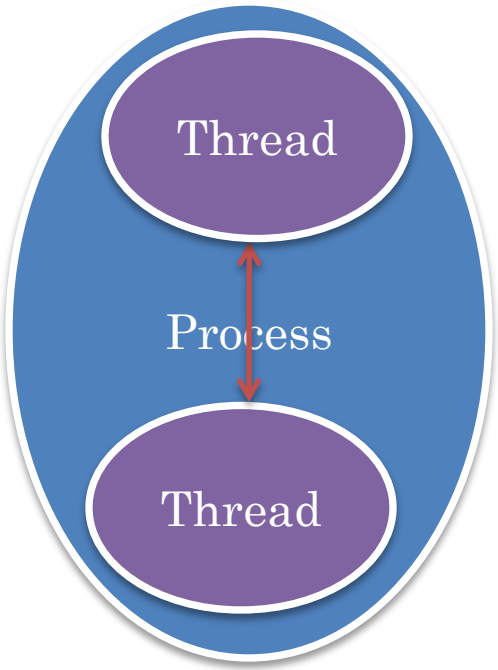
Location of Communicating Threads?



Location of Communicating Threads?



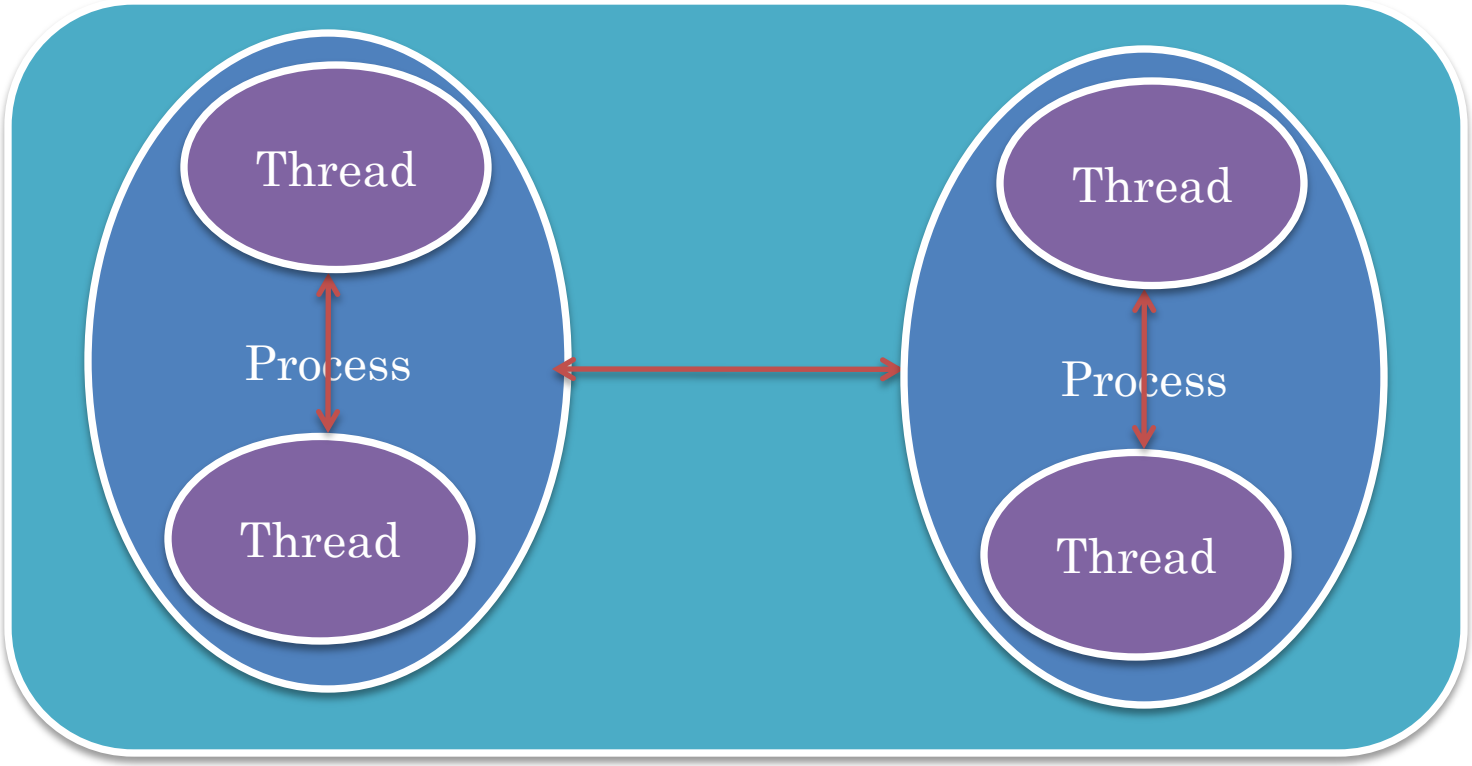
Intra-Process (Address Space) Communication



Rationale?

Intra-Process: Avoids global memory

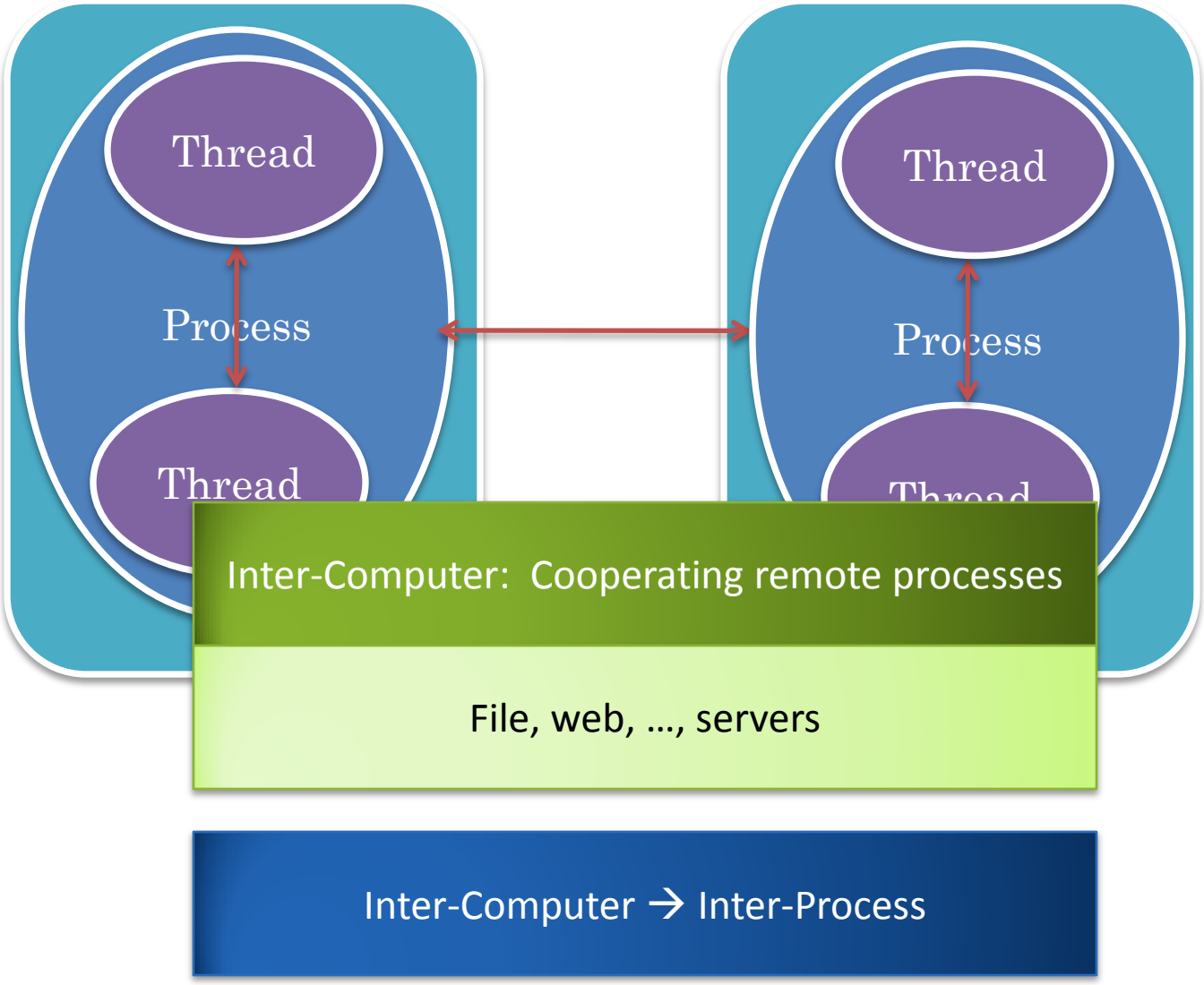
Intra-Computer Message Passing



Inter-Process: Cooperating processes

ls | more

Inter-Computer Message Passing



Location: Applications

Intra-Process: Avoids global memory

Inter-Process: Cooperating processes

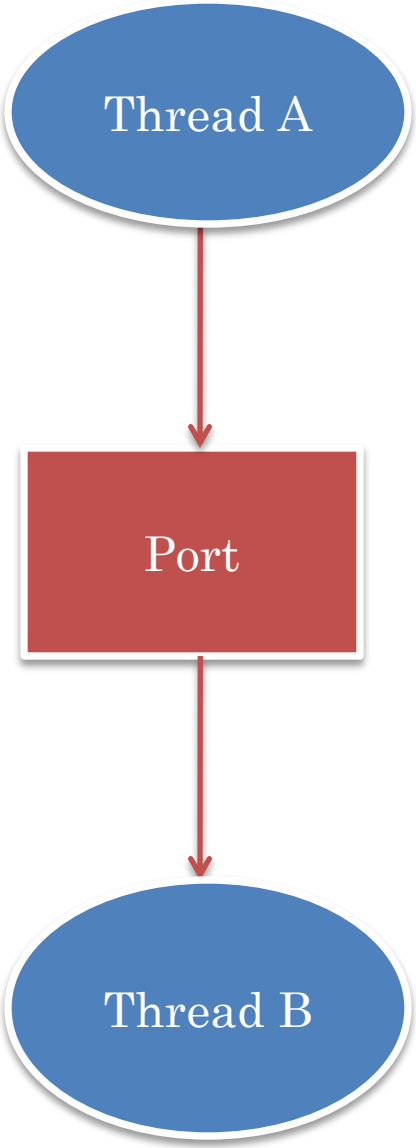
ls | more

Inter-Computer: Cooperating remote processes

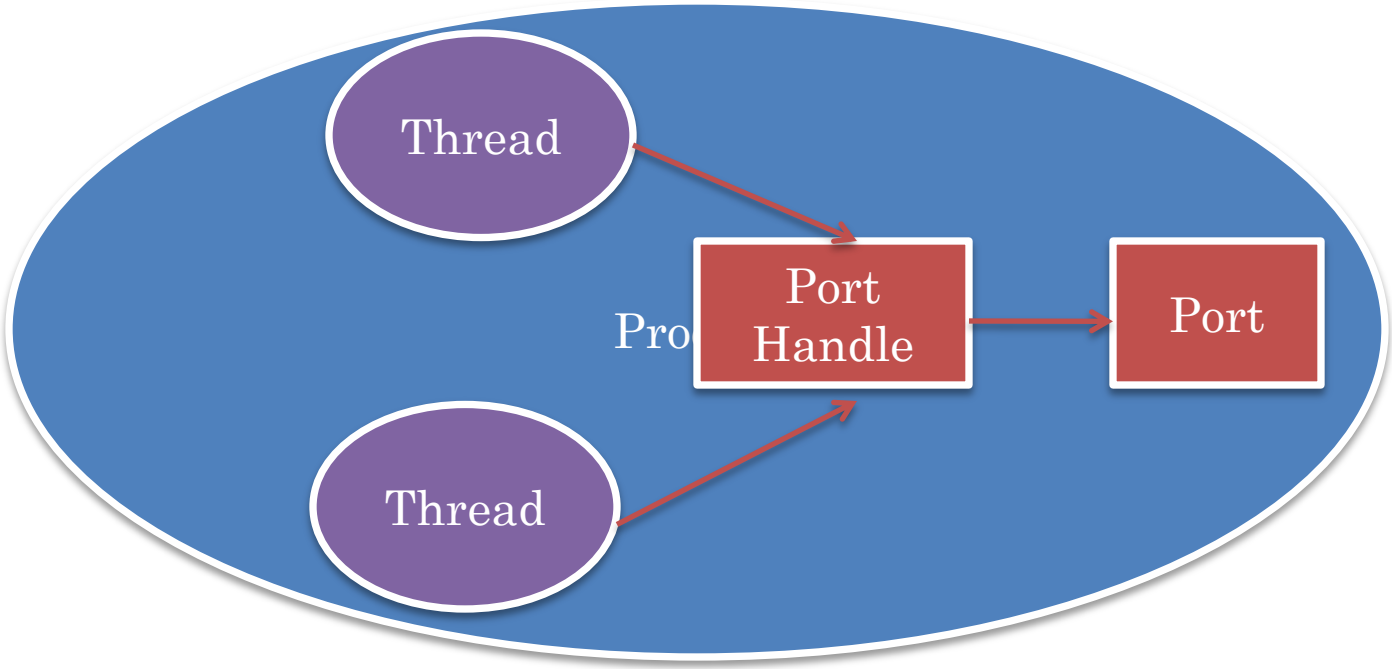
File, web, …, servers

More flexible and distant the message passing, the more complex the API and implementation

How to Name Port?



Intra-Process (Address Space) Communication

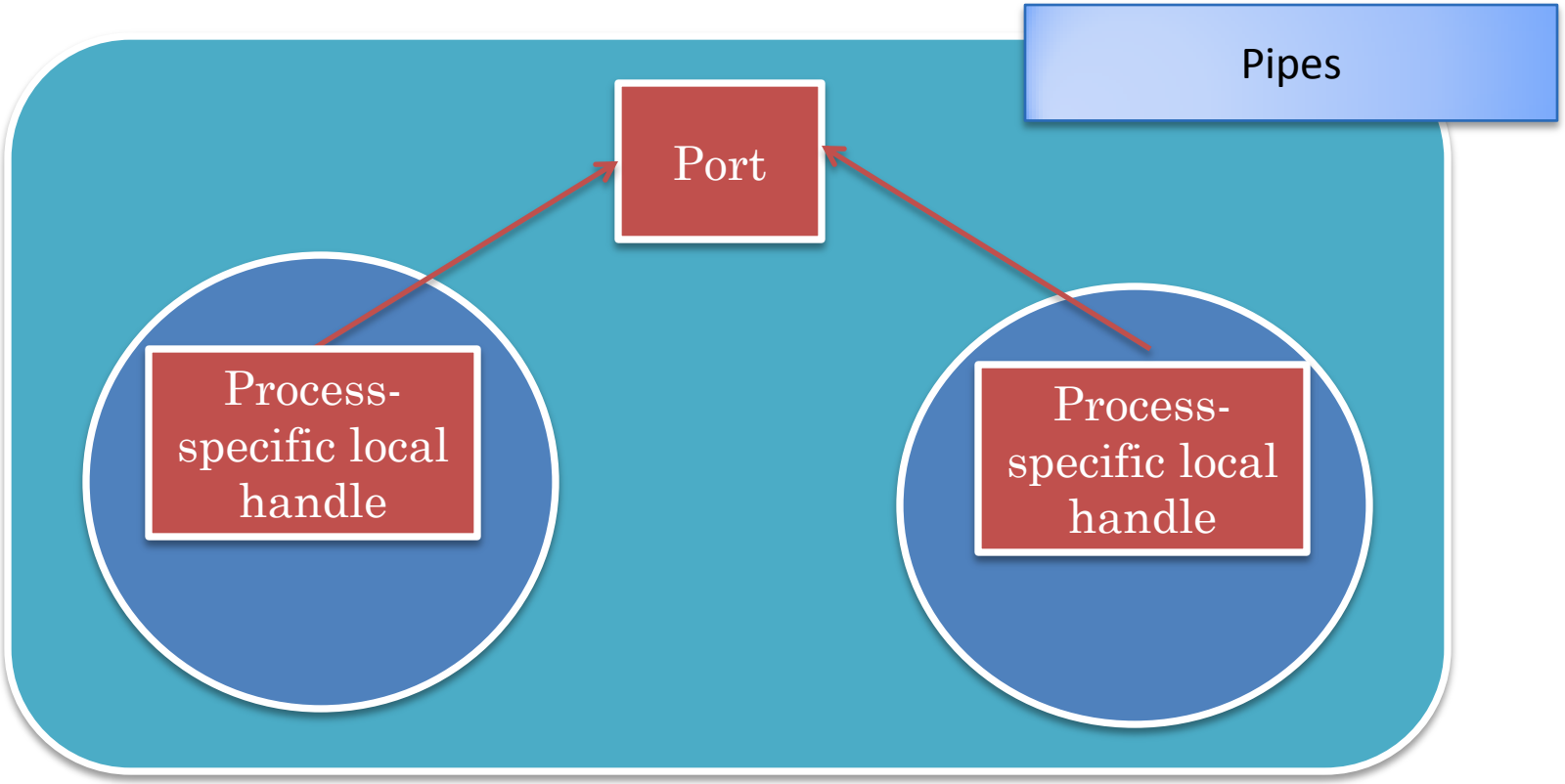


In same process, shared memory

A single port variable can be used by all threads

Xinu, CSP

Intra-Computer Message Passing

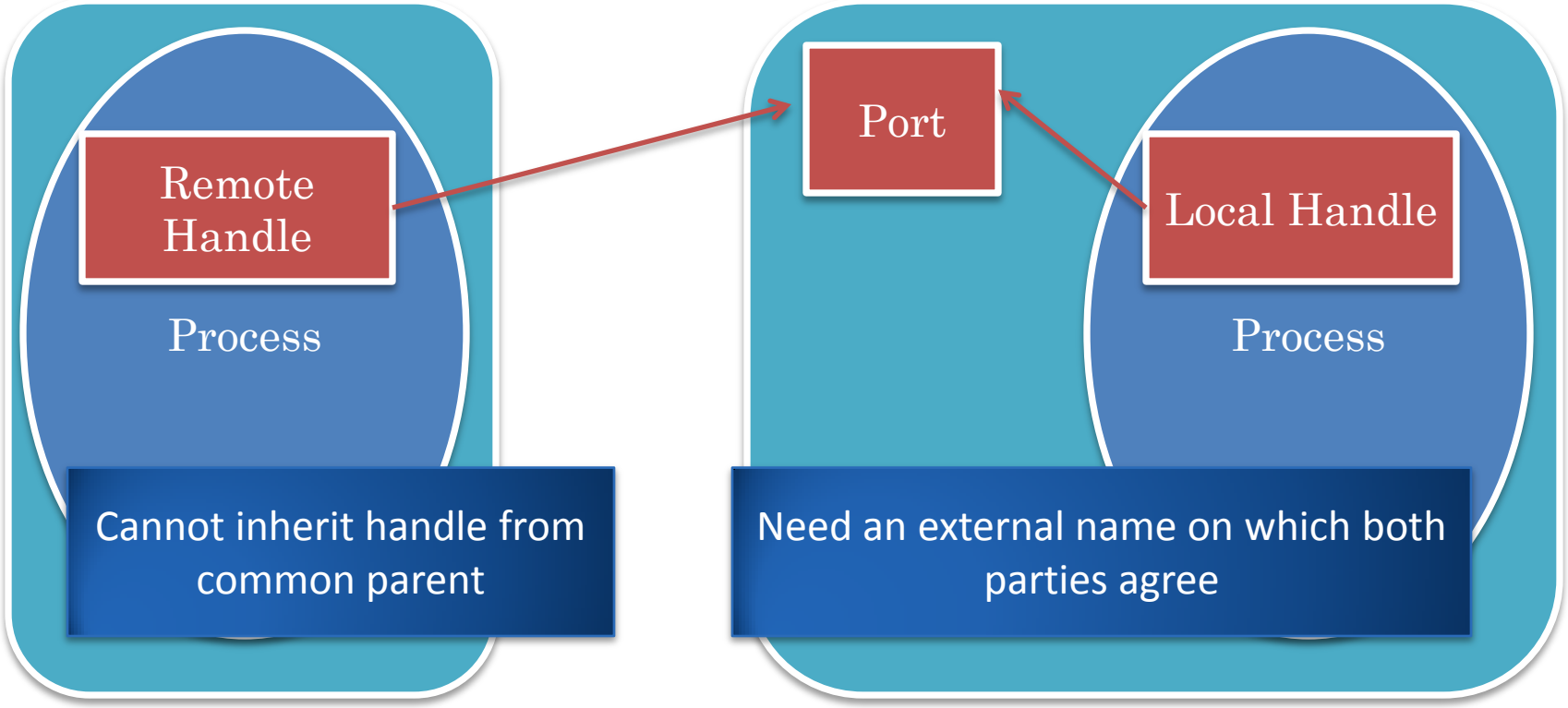


Similar to naming shared file

With different processes, on same OS, different resource (file) descriptors name common port

These can be inherited from common parent process (like standard I/O) or they can have external name like a file name

General Message Passing: External Name



Alternatives

Can use external name for each message, specifying machine address each time

Datagram Socket

Not compatible if underlying communication mechanism requires handshake and thus connection

Access control needed on each message

External name may be bound to handle, which may result in handshake between two machines and authentication and access control at connection time

Stream Socket

An external name server keeps name to handle binding

Java RMI

Connection vs. Non Connection

Non connection based IPC can use any of these mechanisms

Connection based ipc uses handles, not involving external name server, representing specific connection that defines data associated with connection such as offset in stream and rights

IPC Design Space

Shared Memory

Software Interrupts

Message Passing (Multiple Dimensions)

- Reliable?
- In-order?
- Access?
- RPC or RA
- Byte, Block or Object

- Buffer sizes?
- Send blocking times
- Language support?

- Explicit receive?
- Receive Blocking times?
- Select?
- Location of communicating threads?
- Naming?

- Reply?
- Reply blocking times?