Interprocess Communication

- Primitives
- Message Passing: issues
- Communication Schemes
- *Reading: Colouris, Chapter 4*

Interprocess Communication (IPC)

lack of shared memory ⇨ communicate by sending messages

Primitives for interprocess communication

- message passing
  - the RISC among the IPC primitives
- remote procedure call (RPC)
  - process interaction at language level
  - type checking
- transactions
  - support for operations and their synchronization on shared objects

Message Passing

- The primitives:

```
send expression_list to destination_identifier;

receive variable_list from source_identifier;
```

- Variations:

guarded receive:

```
receive variable_list from source_id when B;
```

selective receive:

```
select
  receive var_list from source_id1;
 |receive var_list from source_id2;
 |receive var_list from source_id3;
end
```

Semantics of Message-Passing Primitives

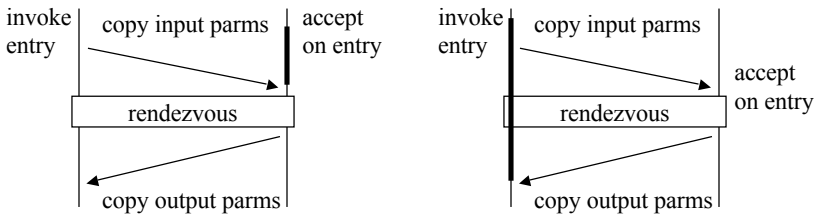
- blocking *vs.* non-blocking
- buffered *vs.* unbuffered
- reliable *vs.* unreliable
- fixed-size *vs.* variable-size messages
- direct *vs.* indirect communication

Blocking *vs.* Non-Blocking Primitives

	blocking	non-blocking
send	Returns control to user only after message has been sent, or until acknowledgment has been received.	Returns control as soon as message queued or copied.
receive	Returns only after message has been received.	Signals willingness to receive message. Buffer is ready.
problems	•*Reduces concurrency.*	•*Need buffering:* •*still blocking* •*deadlocks!* •*Tricky to program.*

Buffered *vs.* Unbuffered Primitives

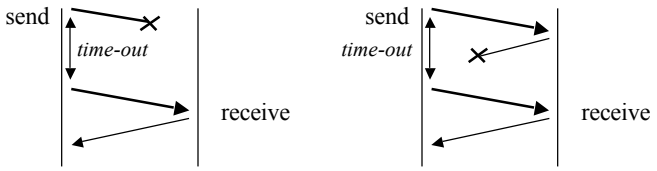
- Asynchronous `send` is never delayed
  - may get arbitrarily ahead of `receive`.
- However: messages need to be buffered.
- If no buffering available, operations become blocking, and processes are synchronized on operations: **rendezvous**.



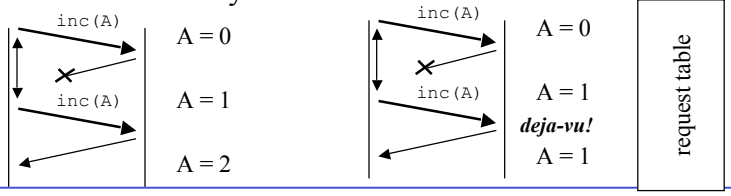
Reliable *vs.* Unreliable Primitives

- Transmission problems:

  corruption loss duplication reordering
- Recovery mechanism: Where?
- Reliable transmission: acknowledgments



- At-least-one *vs.* exactly-one semantics



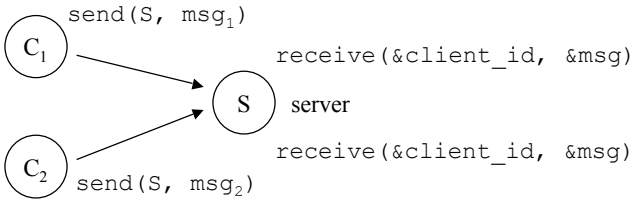
Direct *vs.* Indirect Communication

- Direct communication:

```
send(P, message)
receive(Q, message)
```

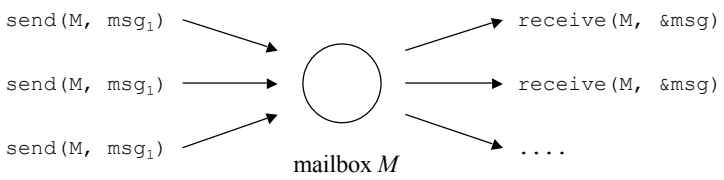
- Variation thereof:

```
send(P, message)
receive(var, message)
```



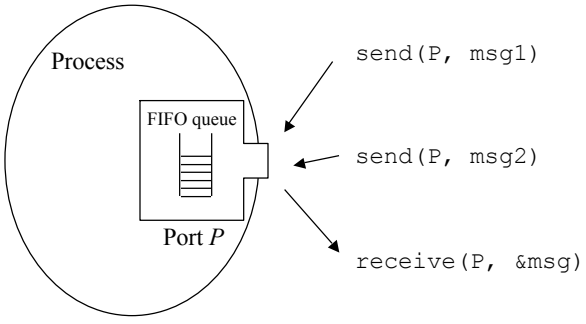
Indirect Communication

- Treat communication paths as first-class objects.
- Example: Mailboxes



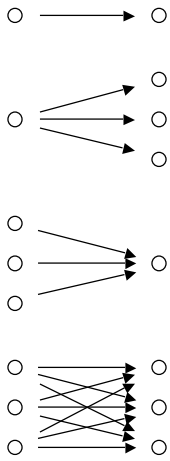
Indirect Communication (2)

- Example: Accent (CMU)



- multiple senders
- only one receiver
- access to port is passed between processes in form of capabilities

Communication Schemes



one-to-one unicast

one-to-many multicast

many-to-one

many-to-many