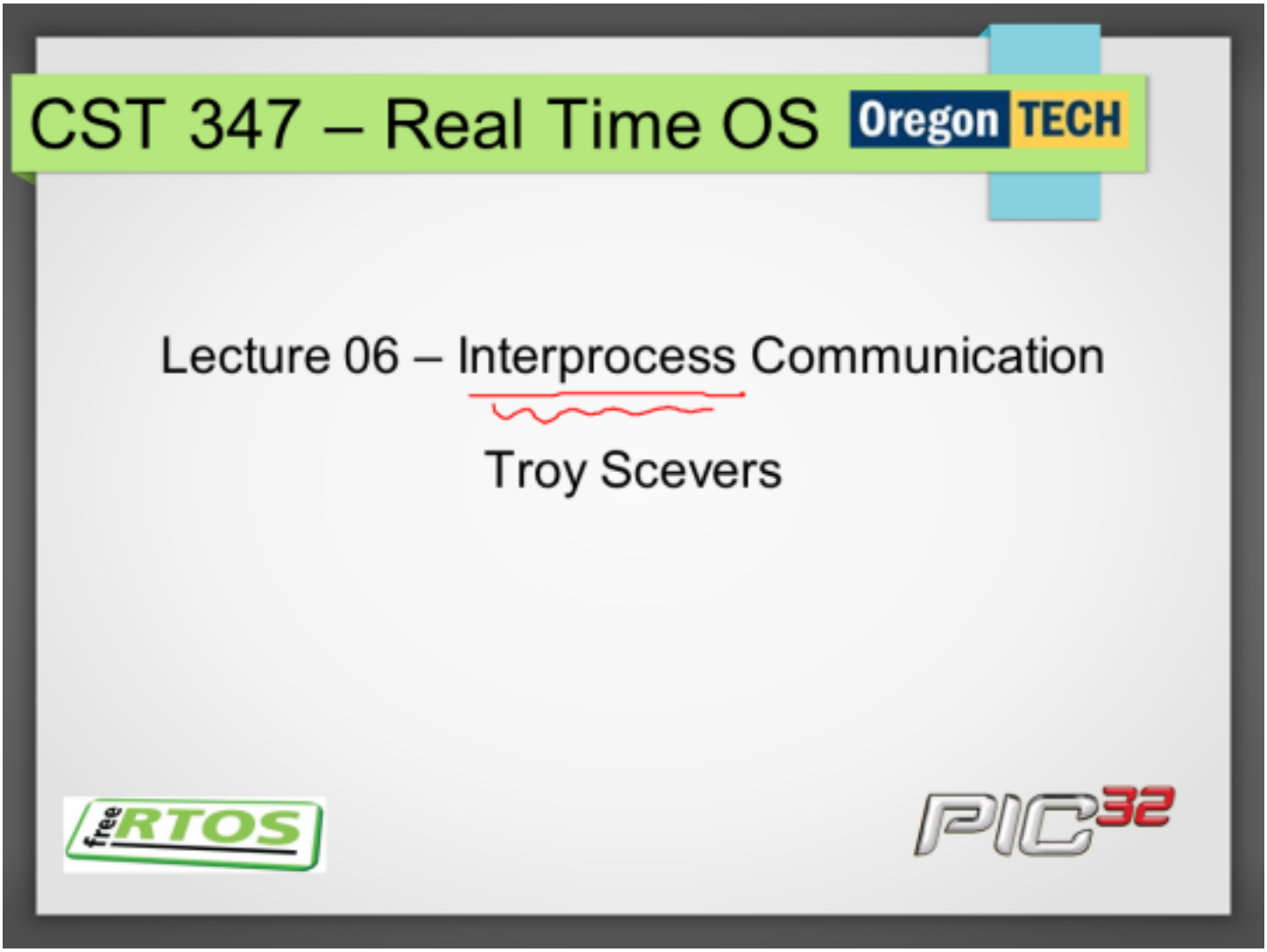
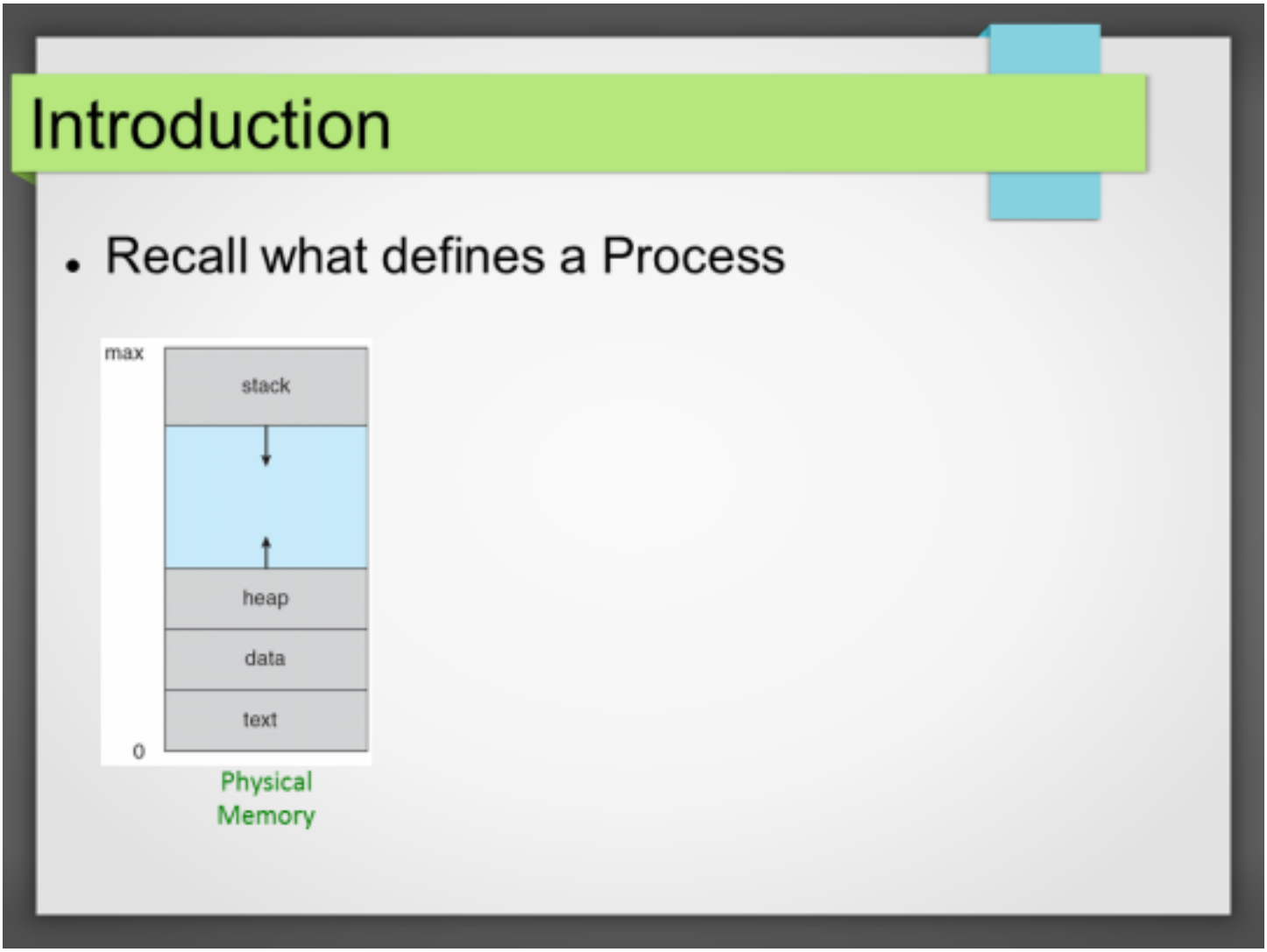
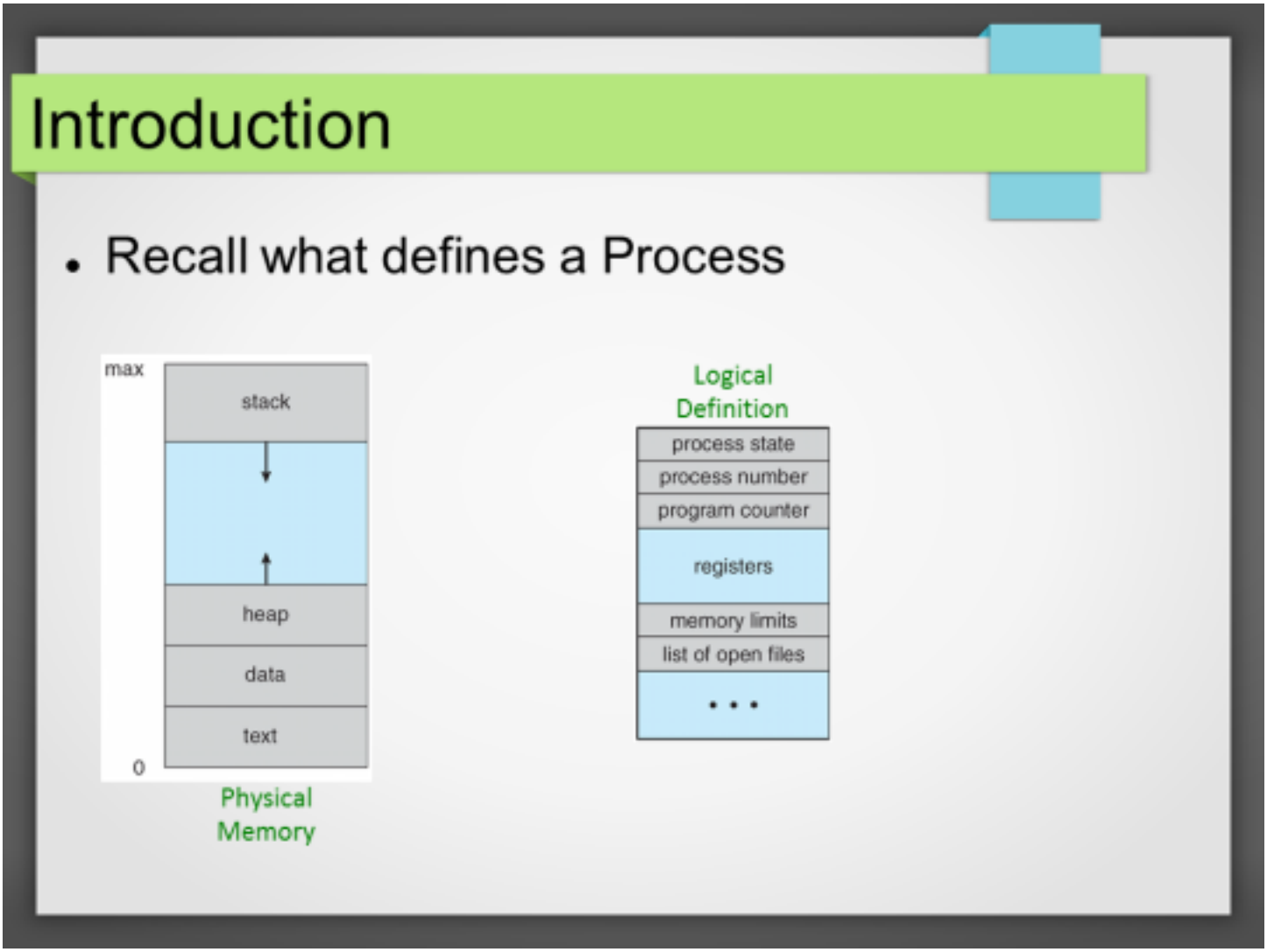
CST 347 ?? Real Time OS



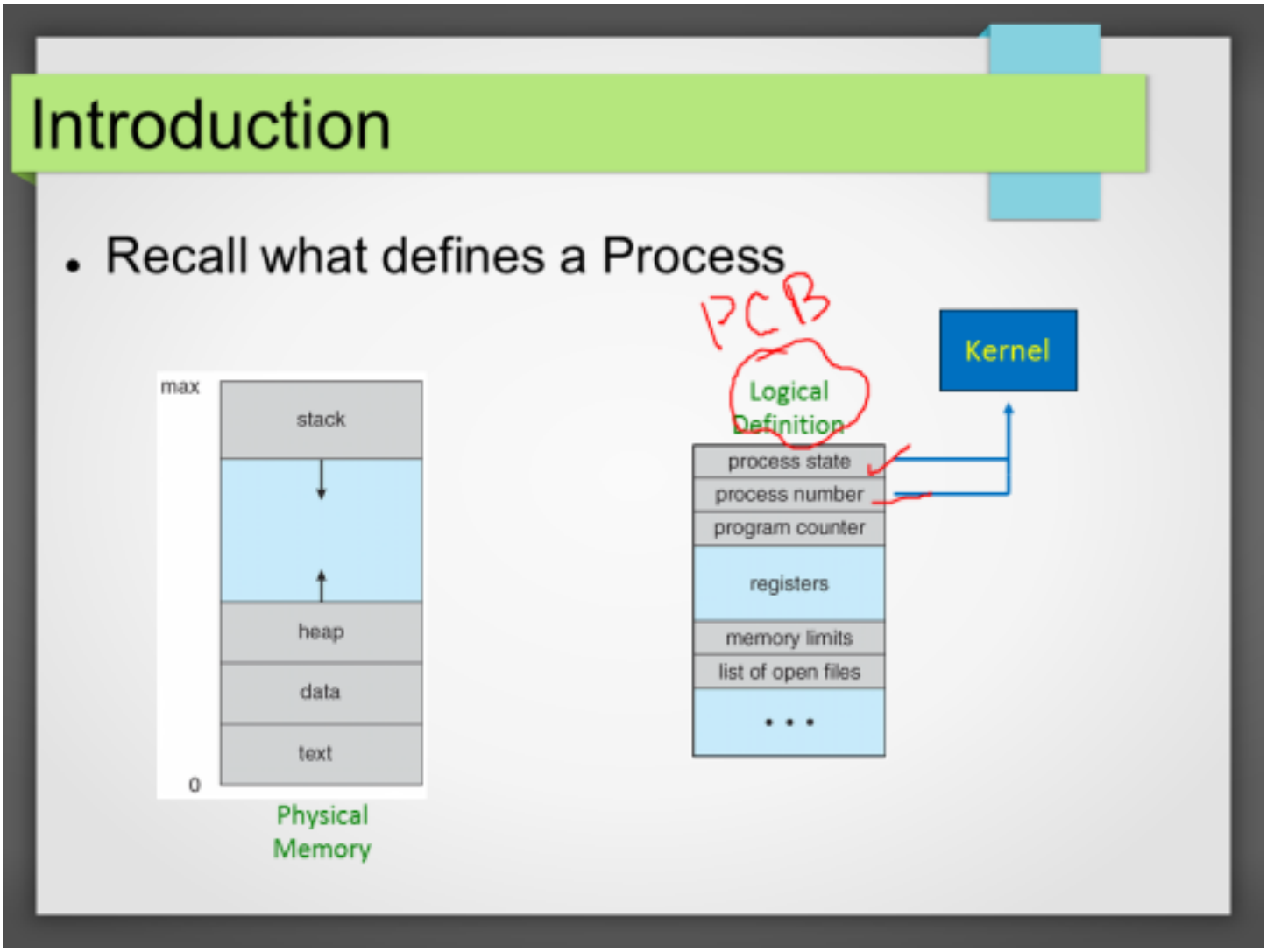
Introduction



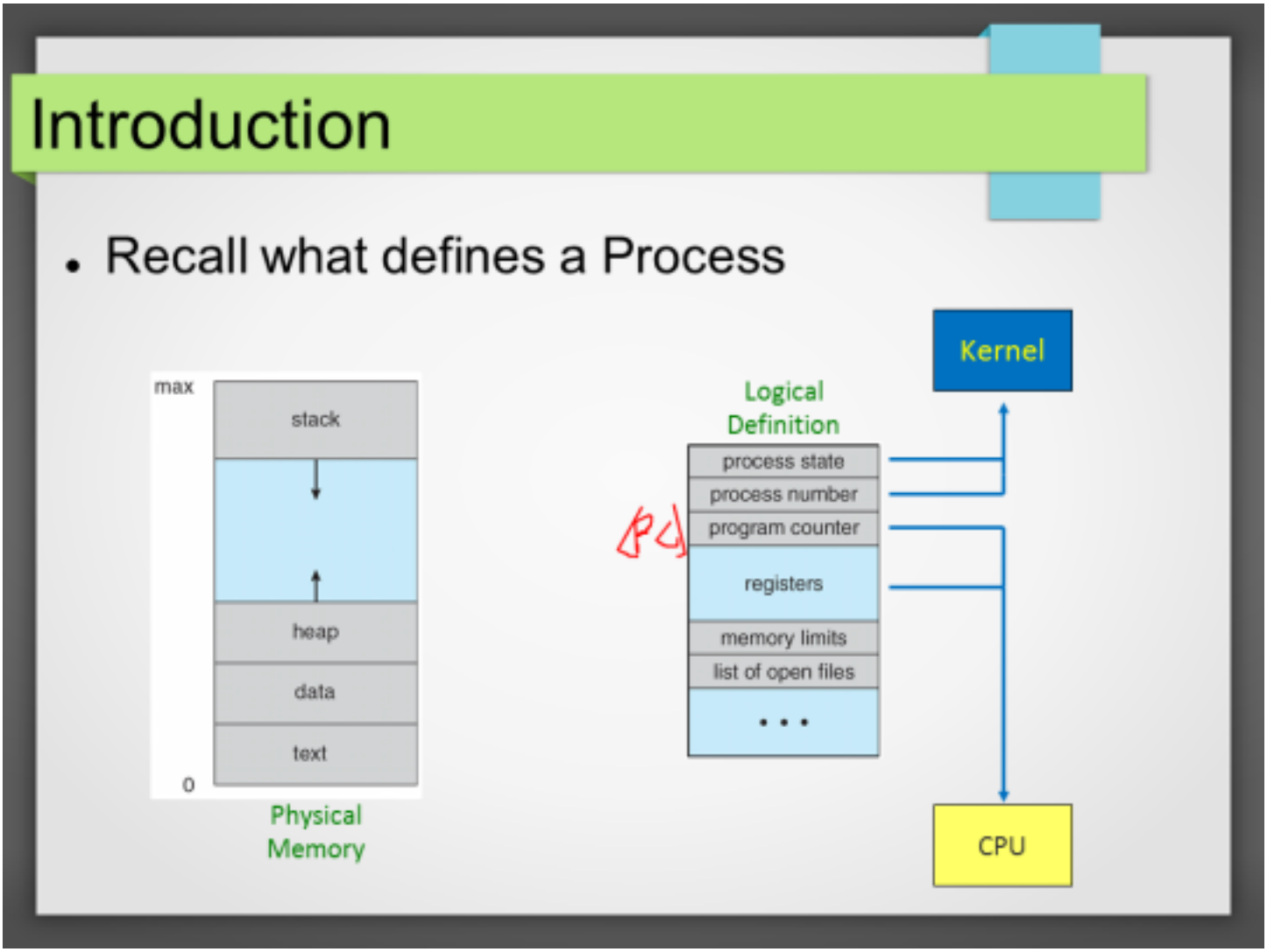
Introduction



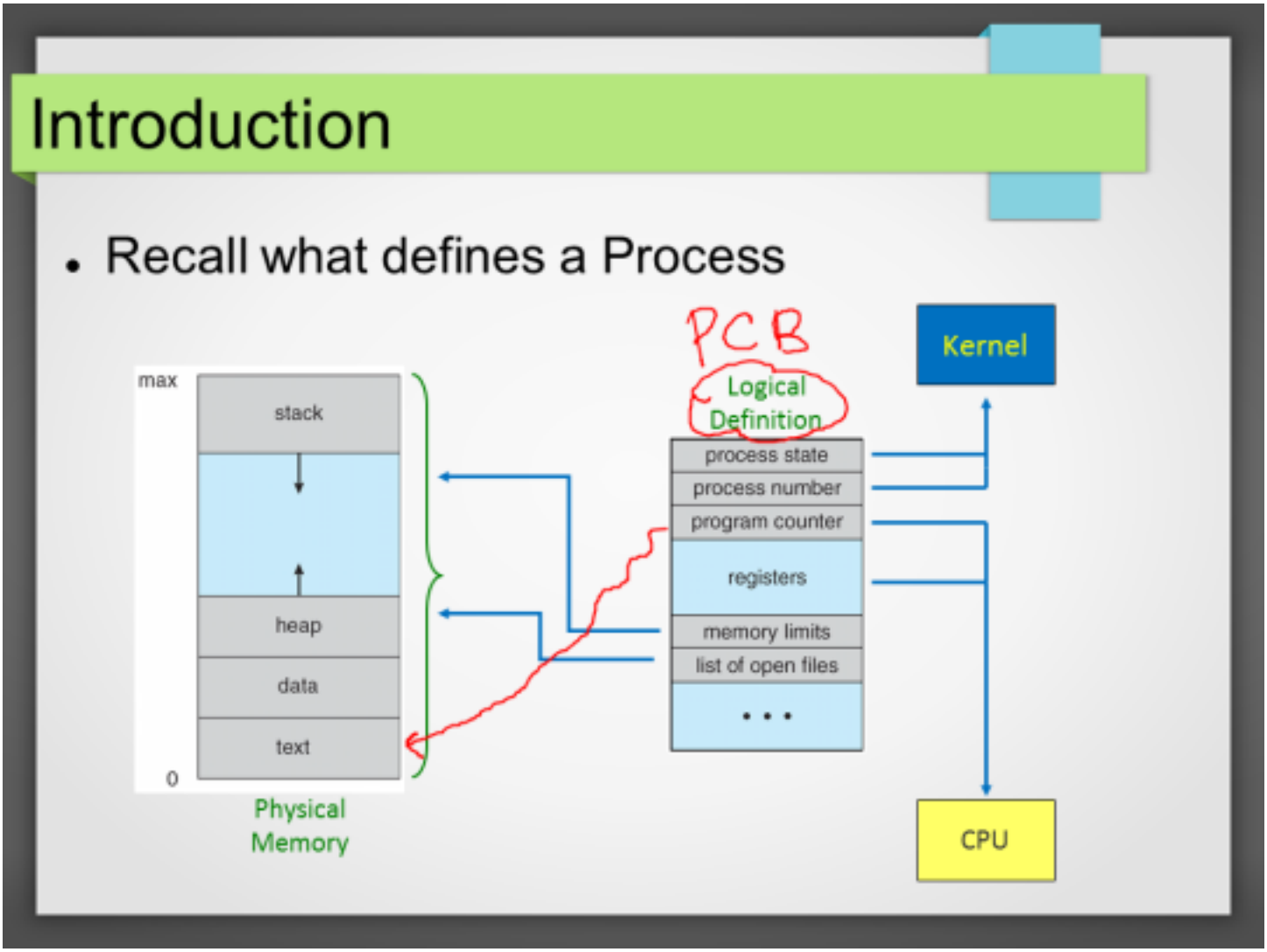
Introduction



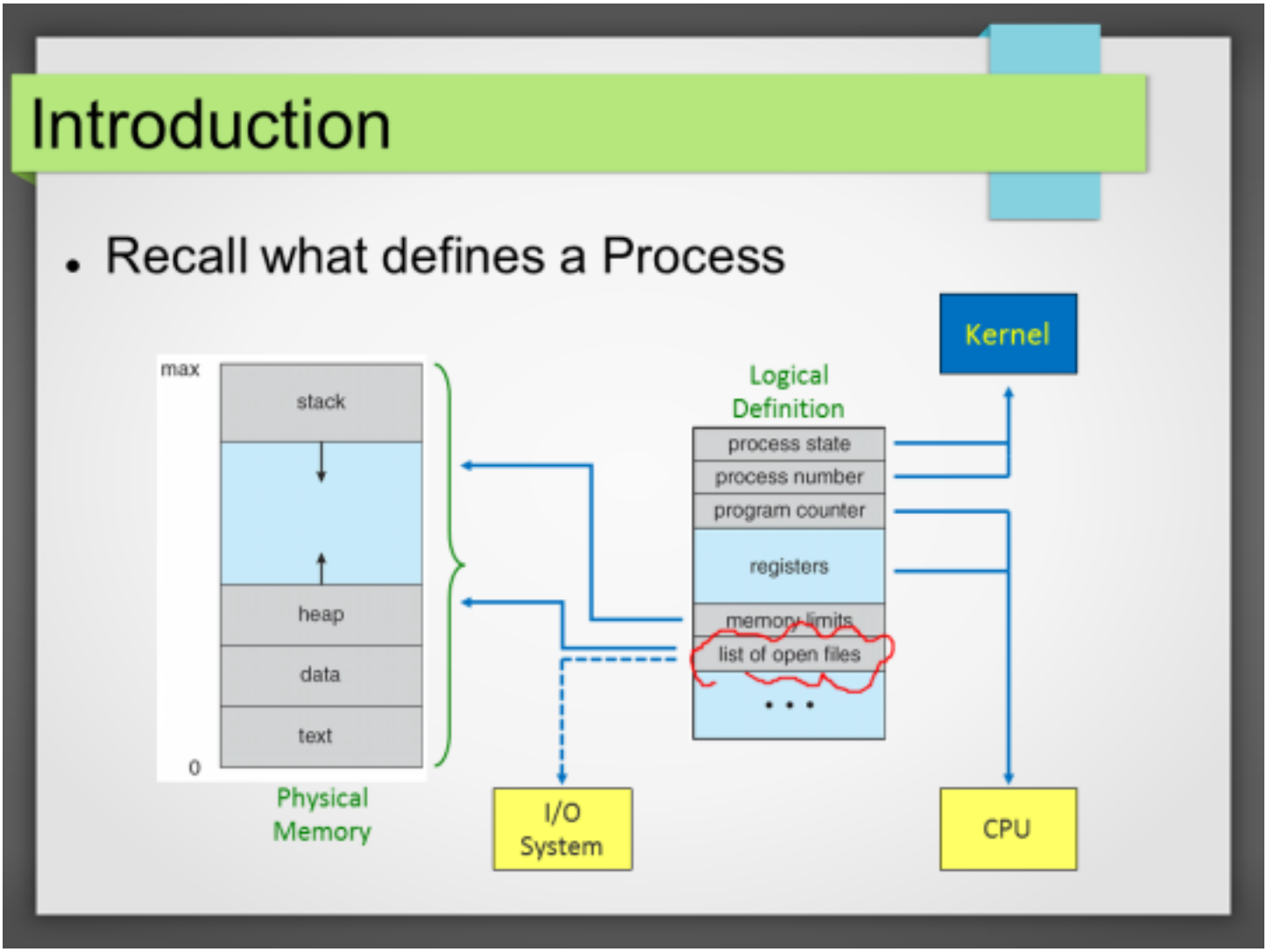
Introduction



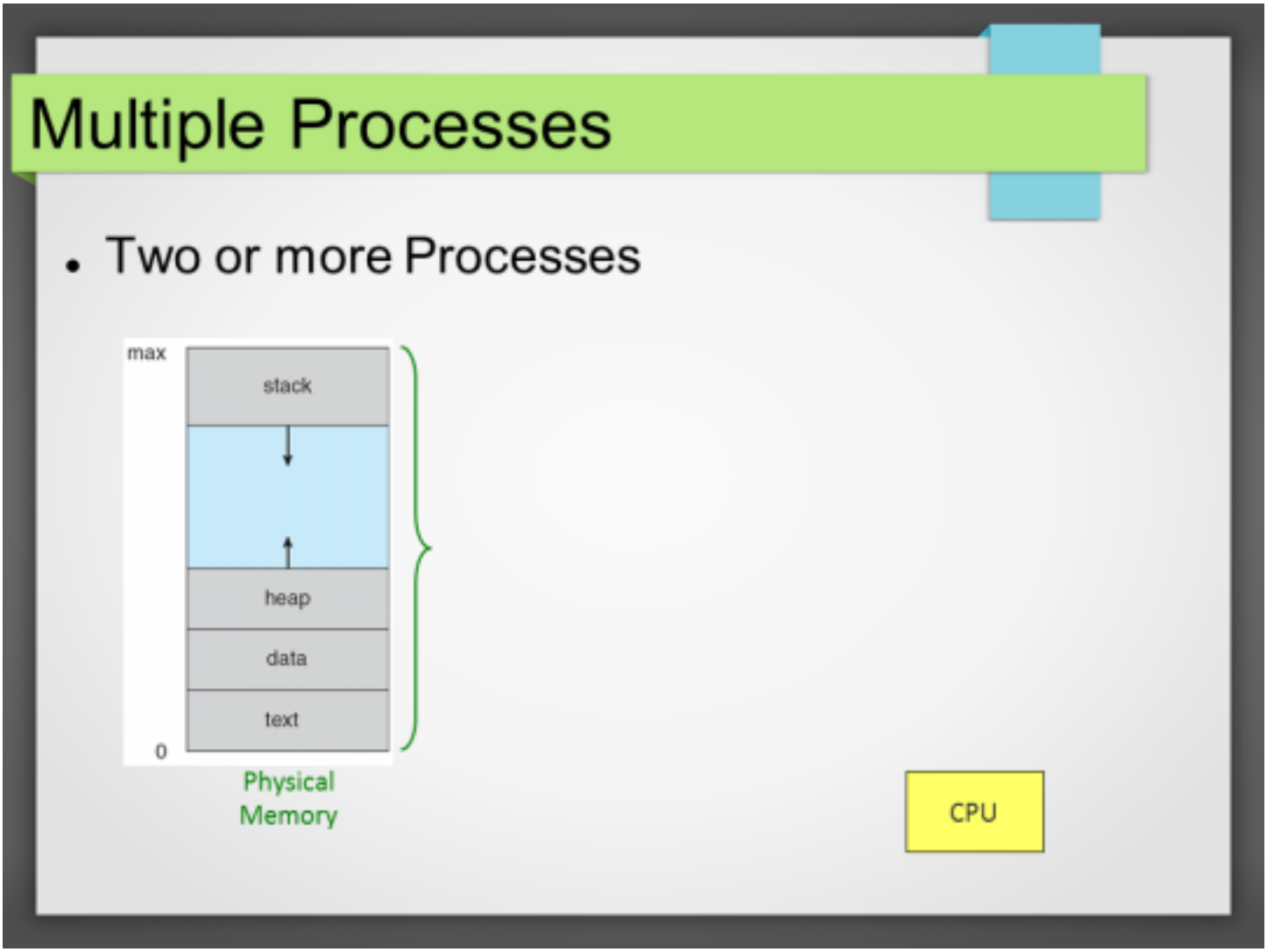
Introduction



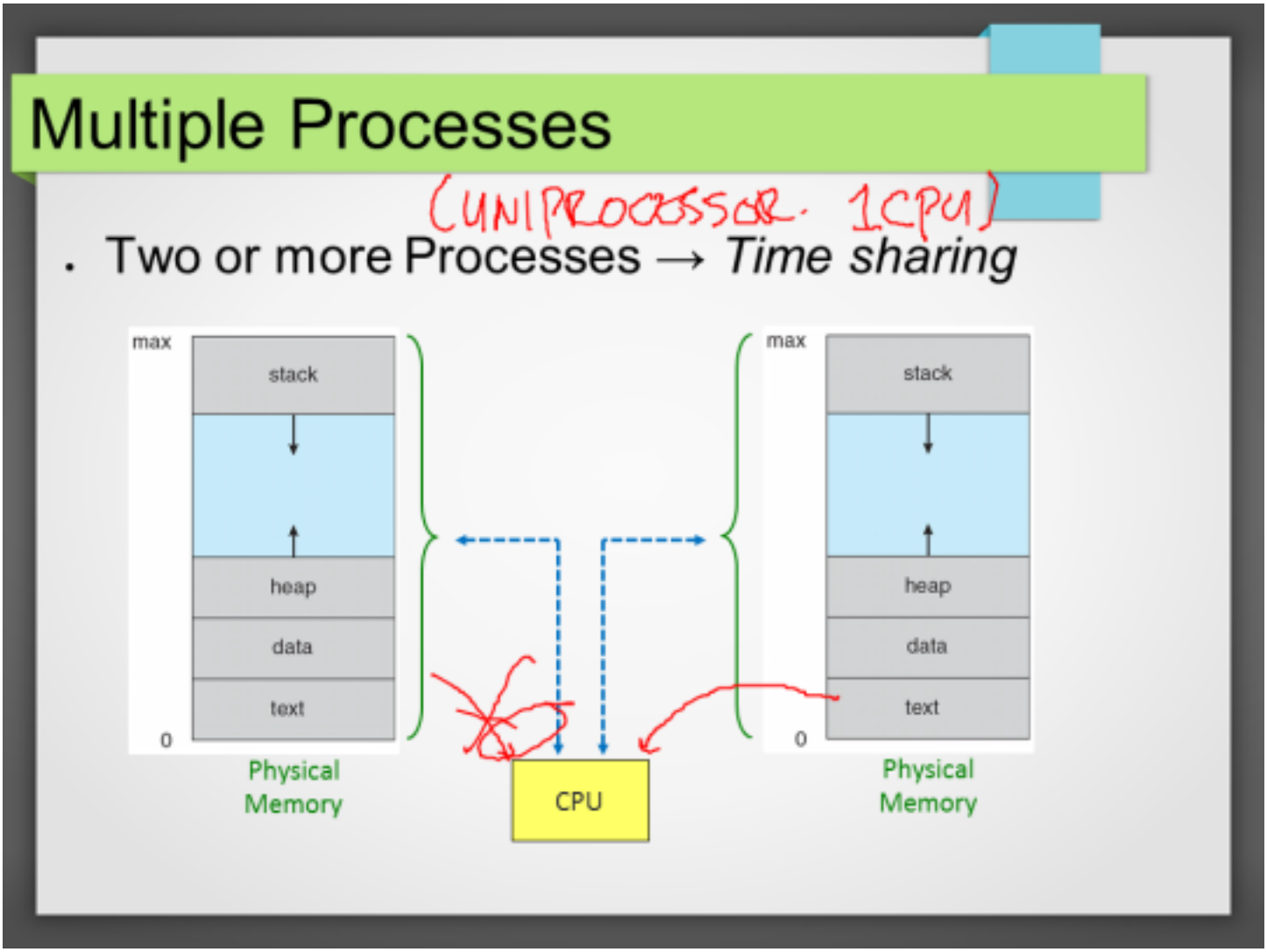
Introduction



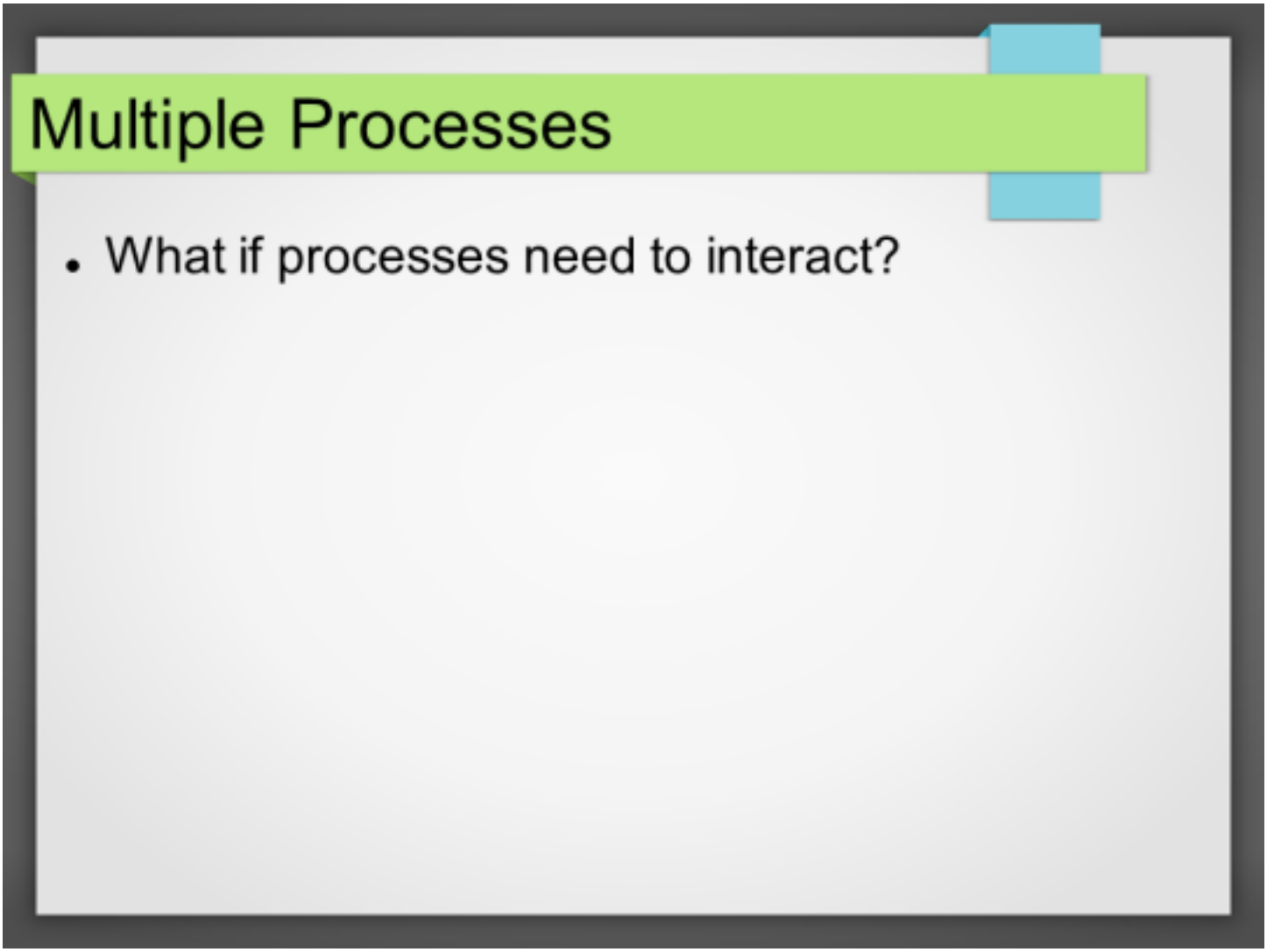
Multiple Processes



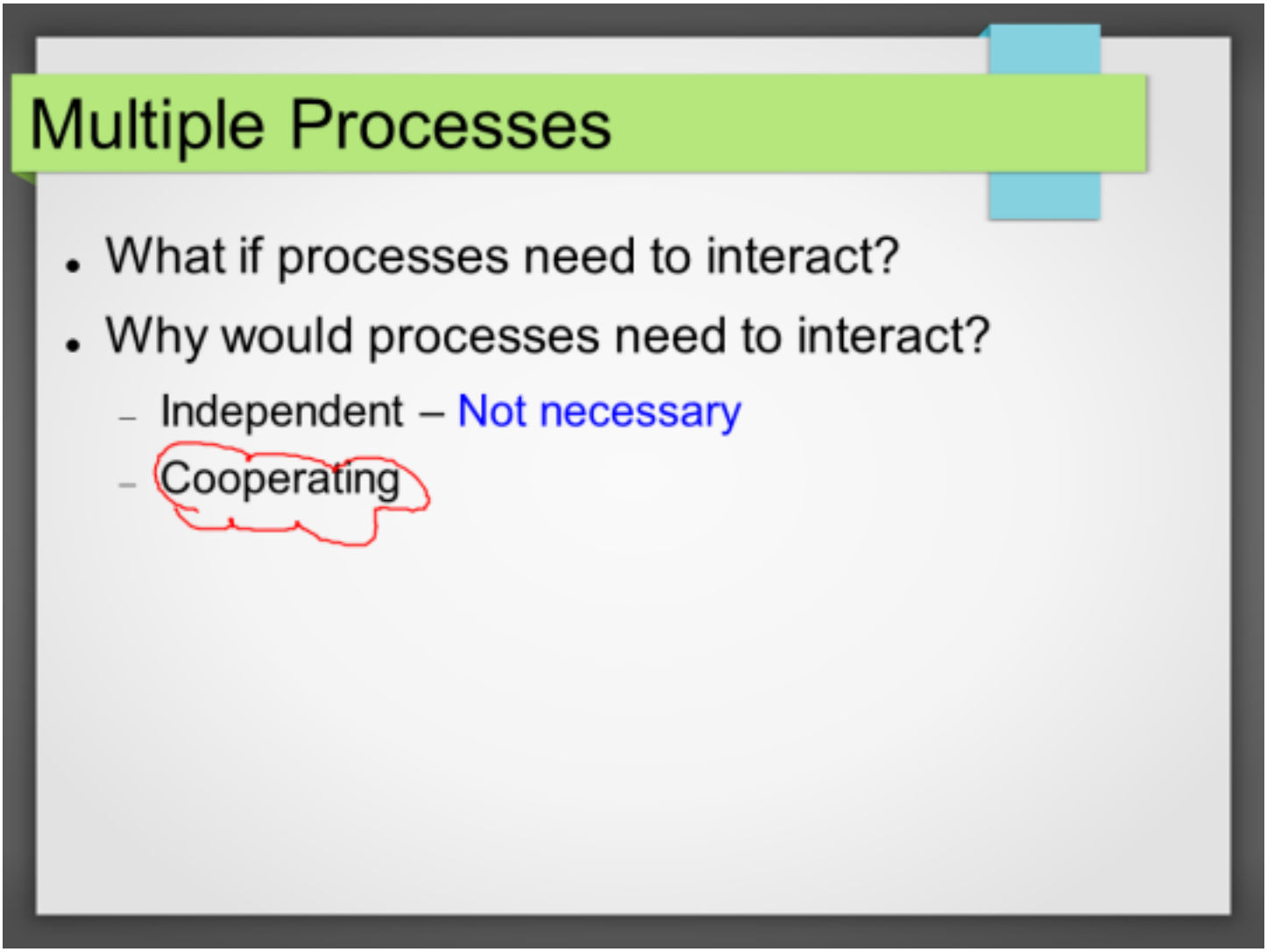
Multiple Processes



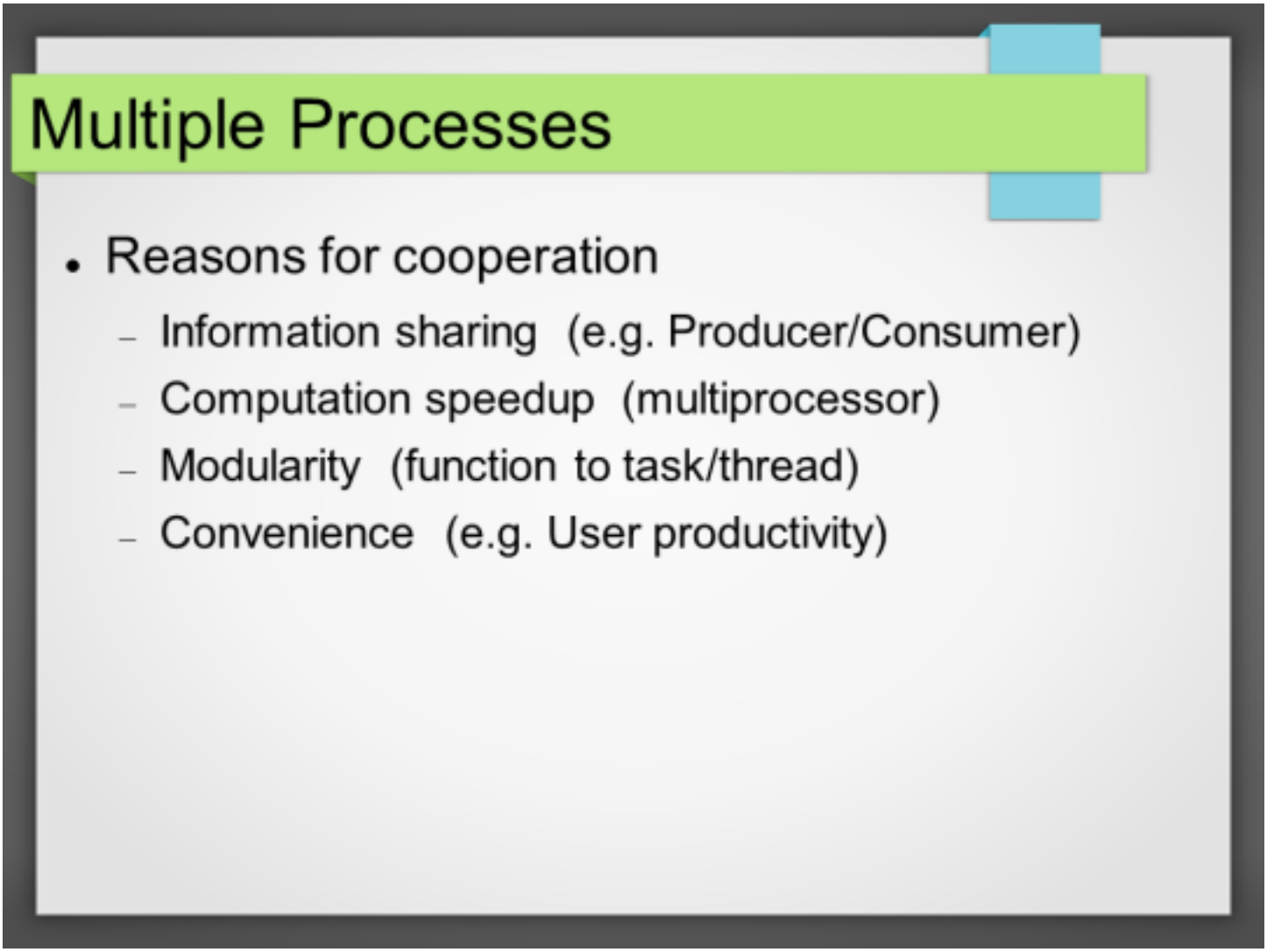
Multiple Processes



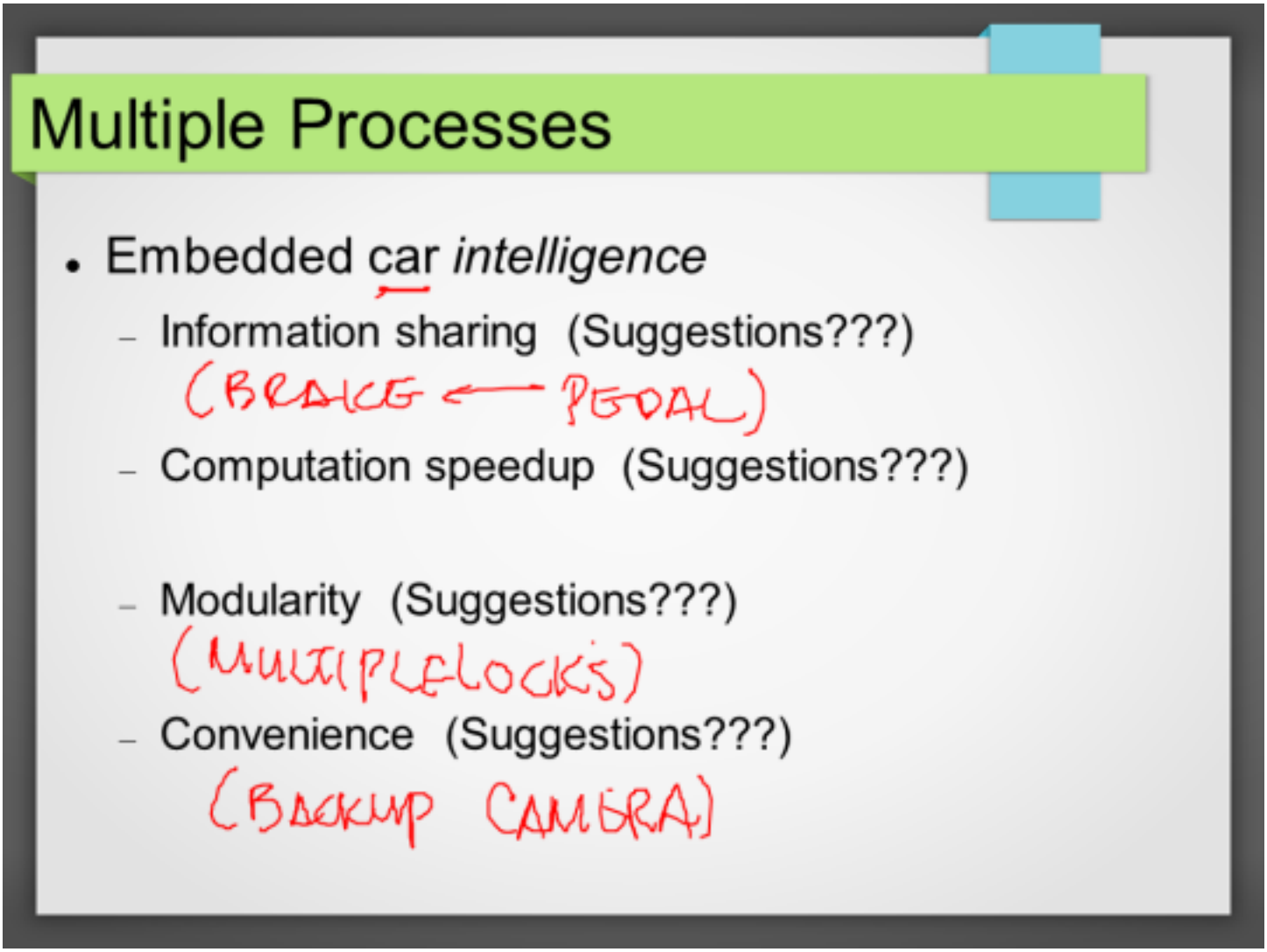
Multiple Processes



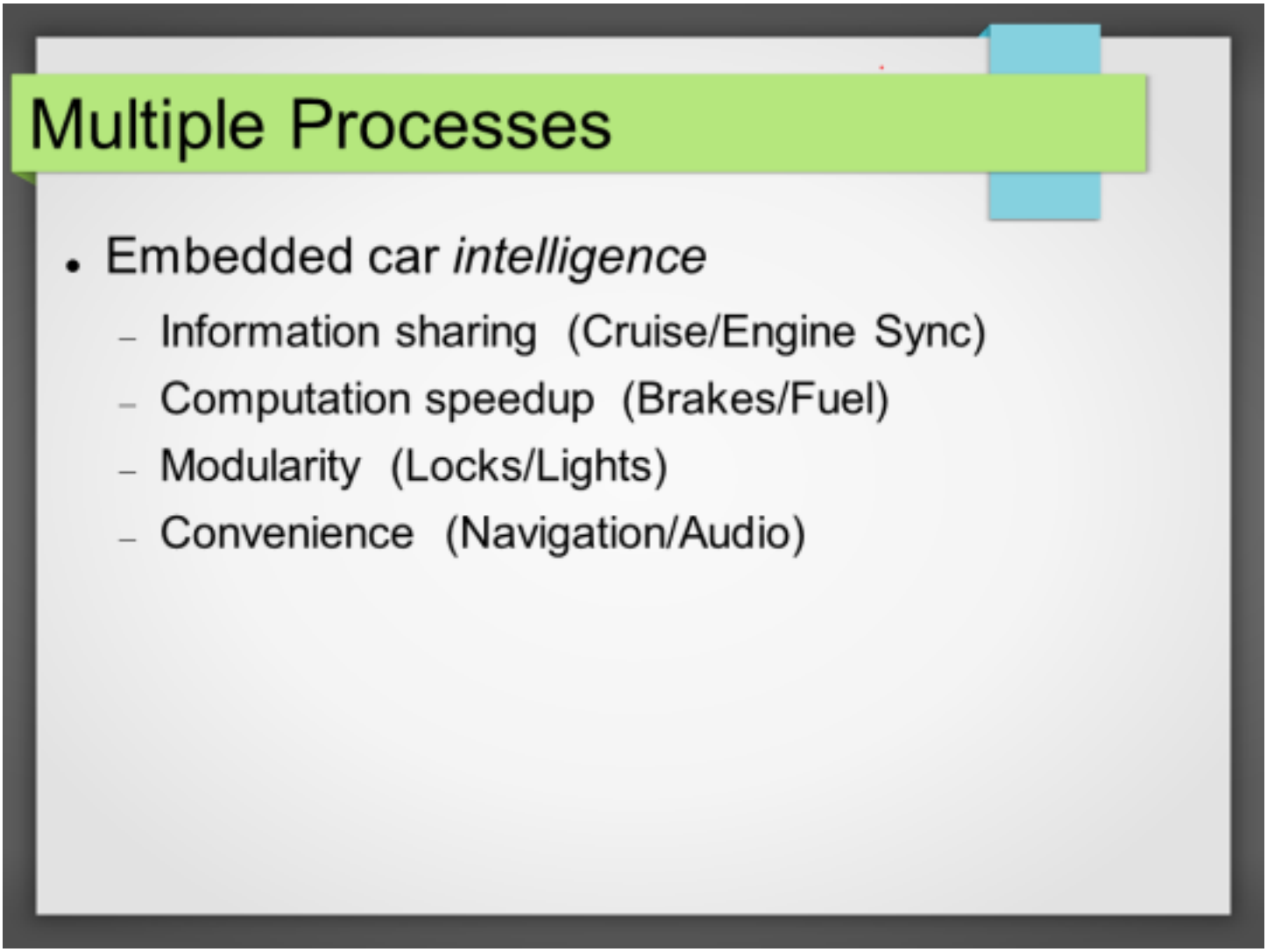
Multiple Processes



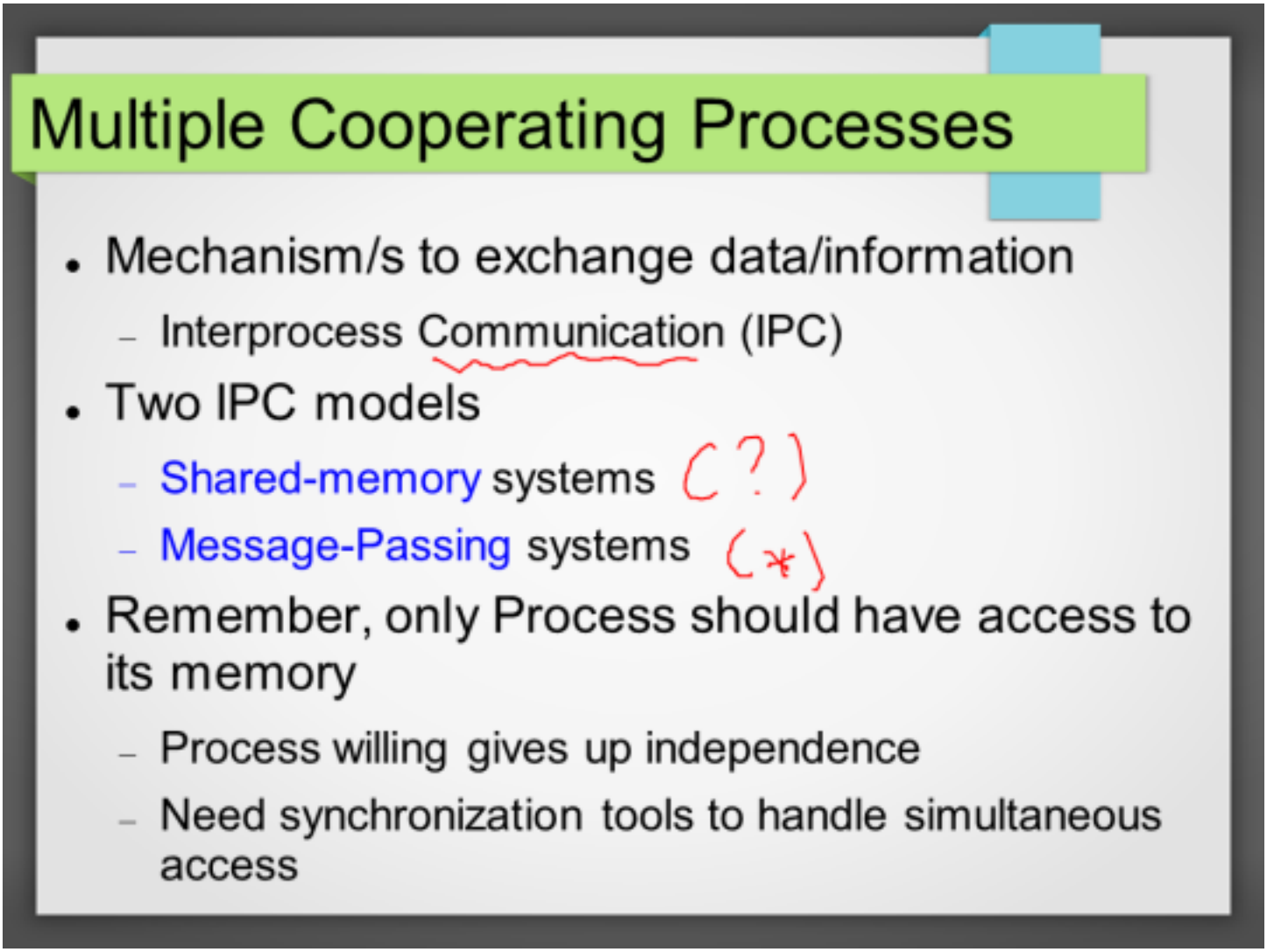
Multiple Processes



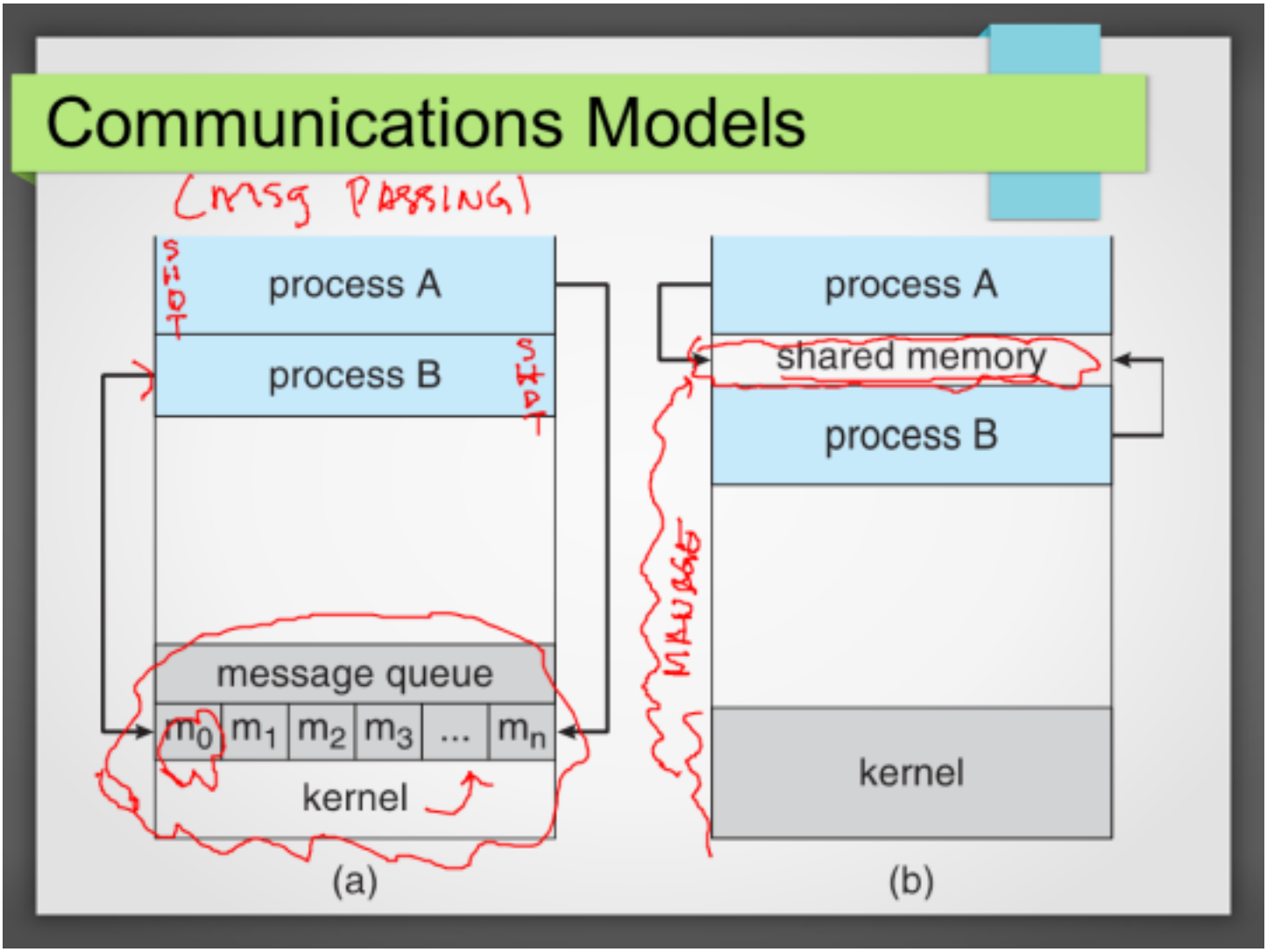
Multiple Processes



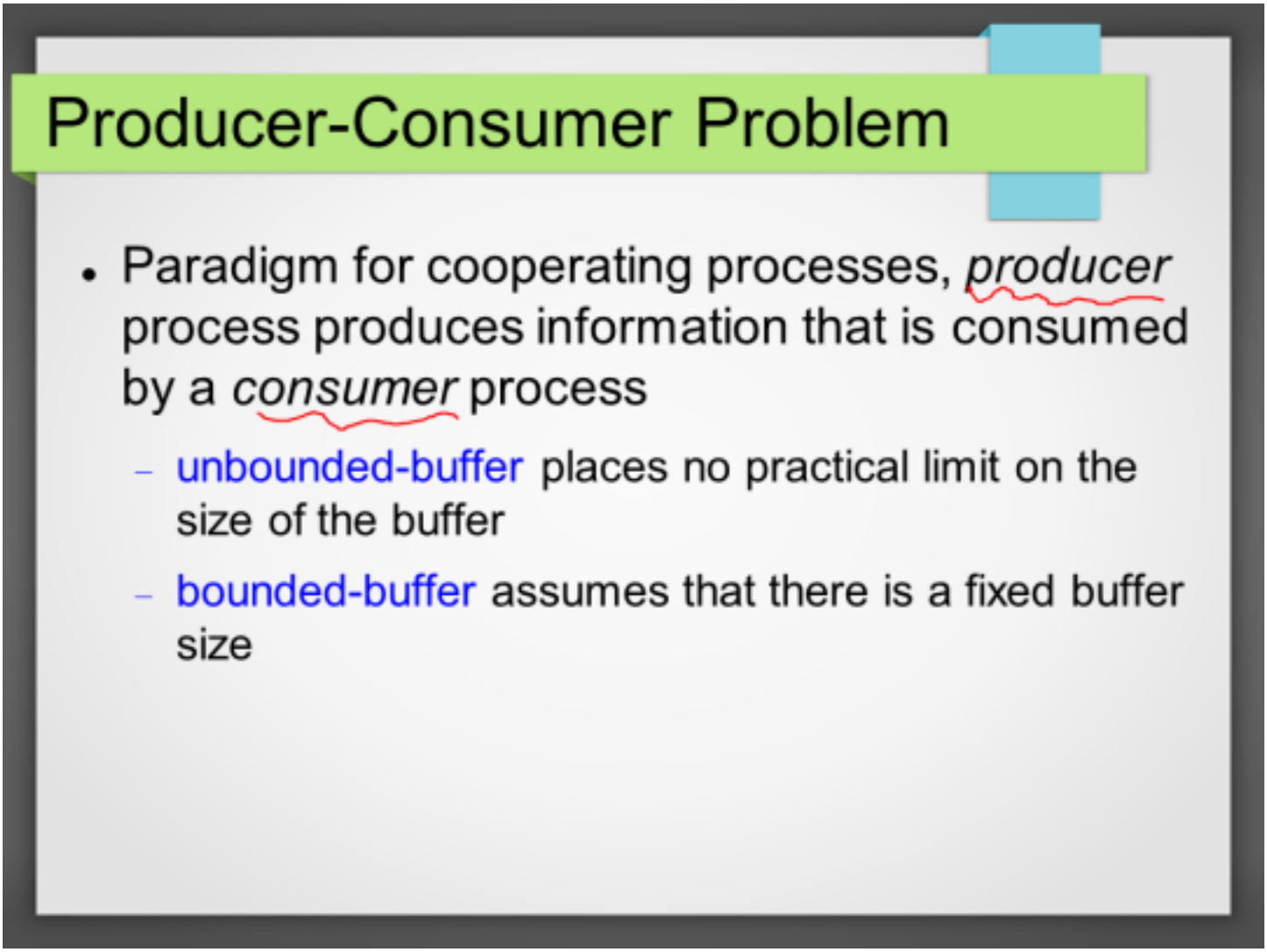
Multiple Cooperating Processes



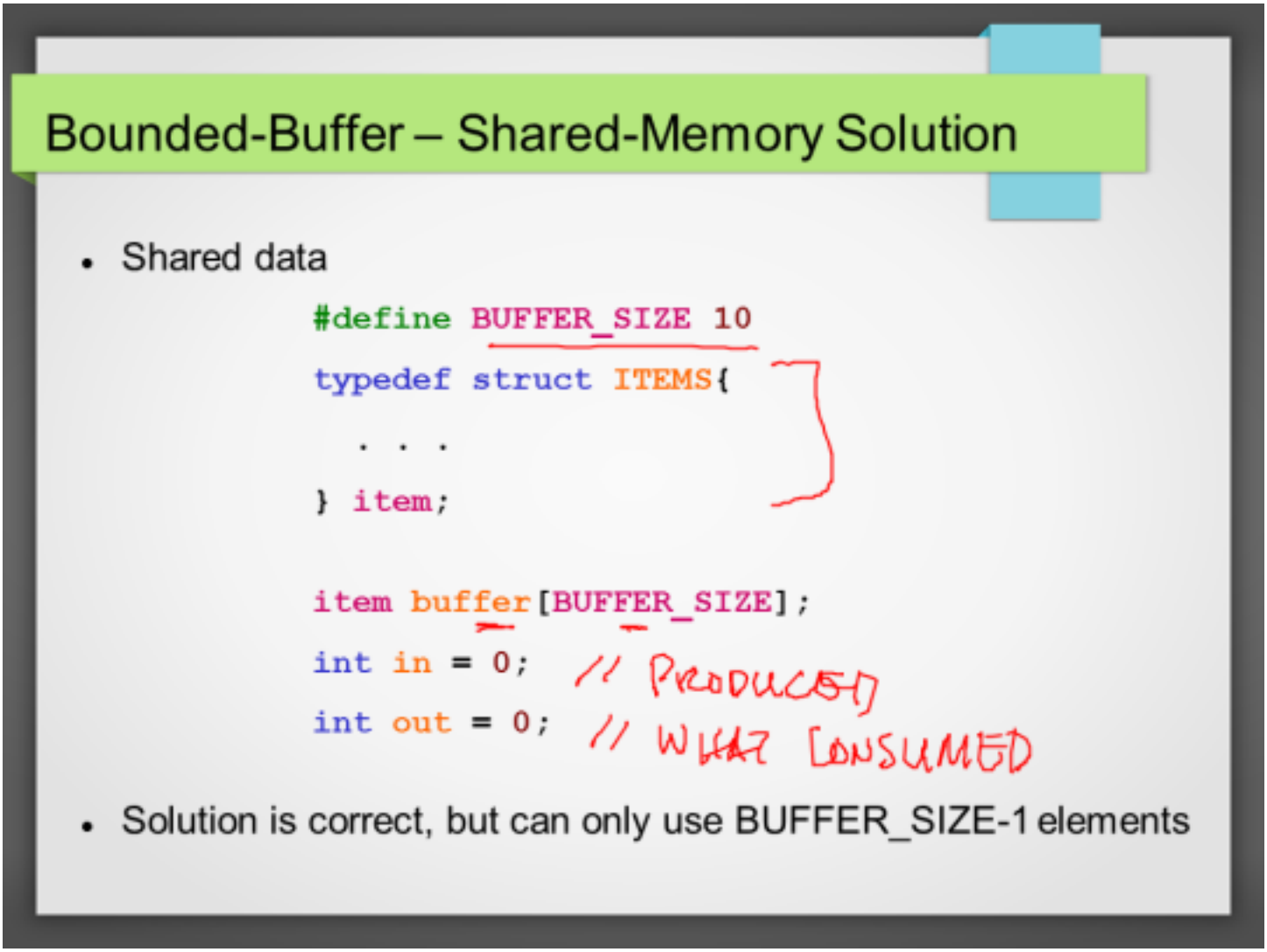
Communications Models



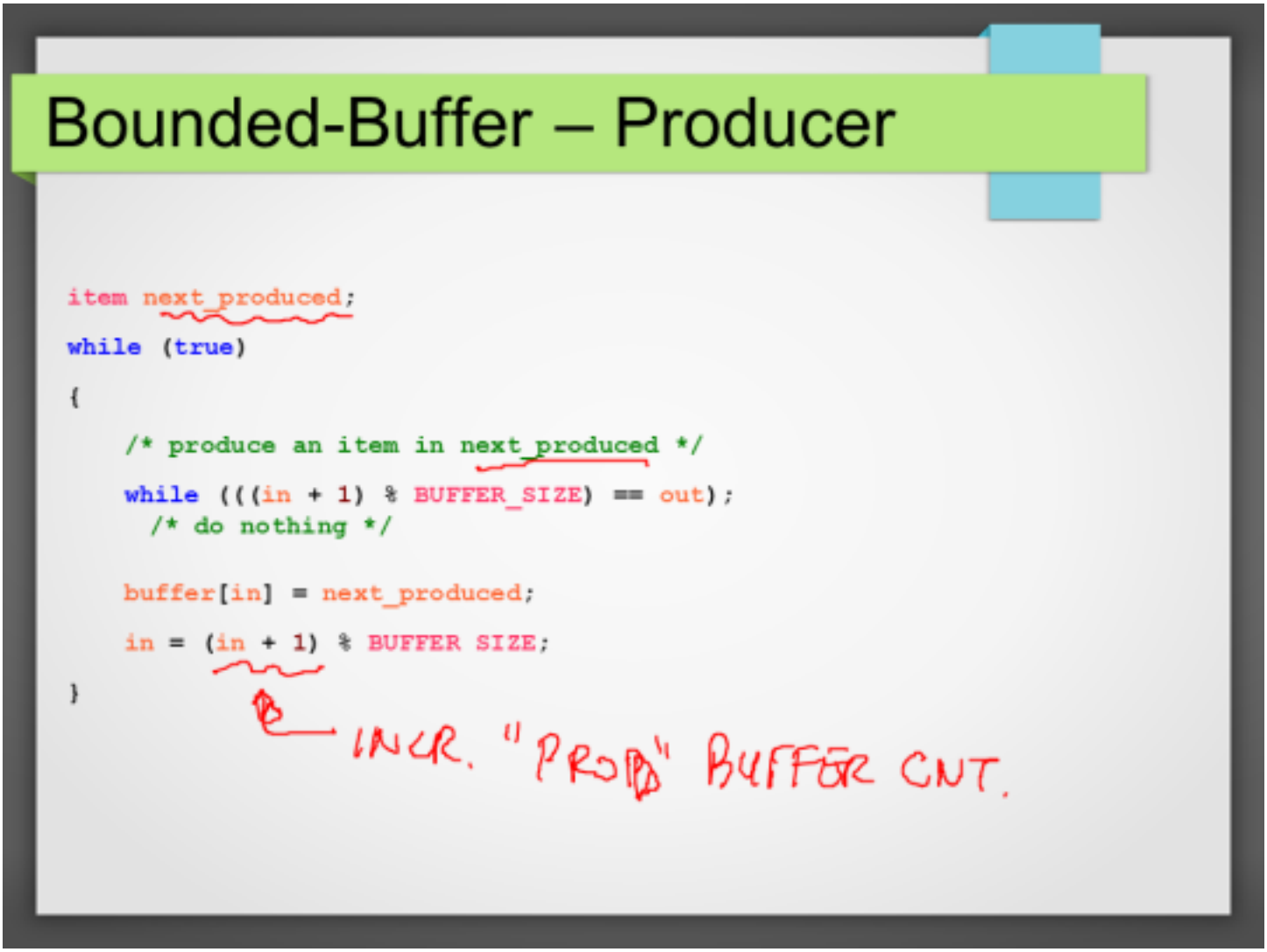
Producer-Consumer Problem



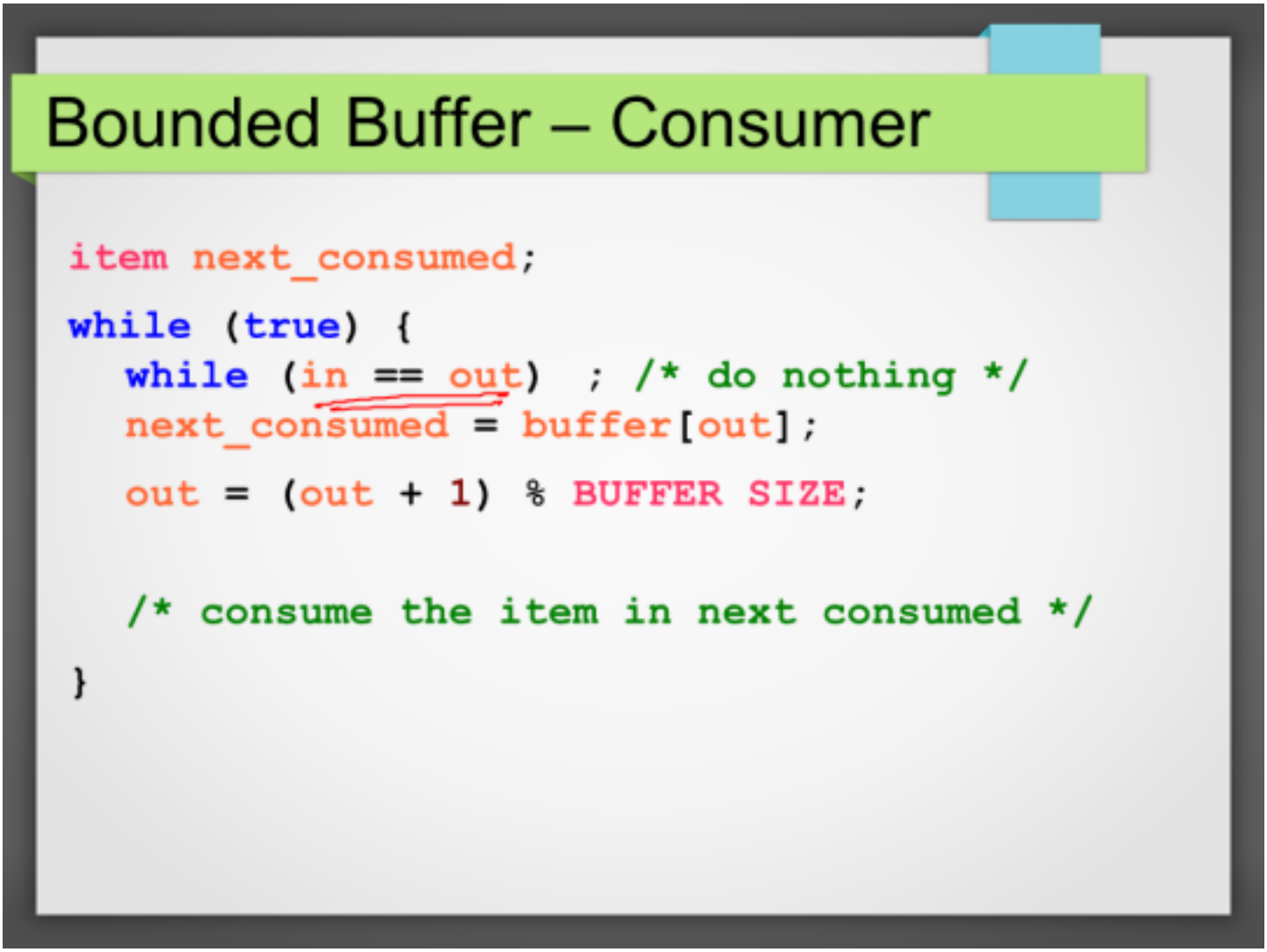
Bounded-Buffer ?? Shared-Memory Solution



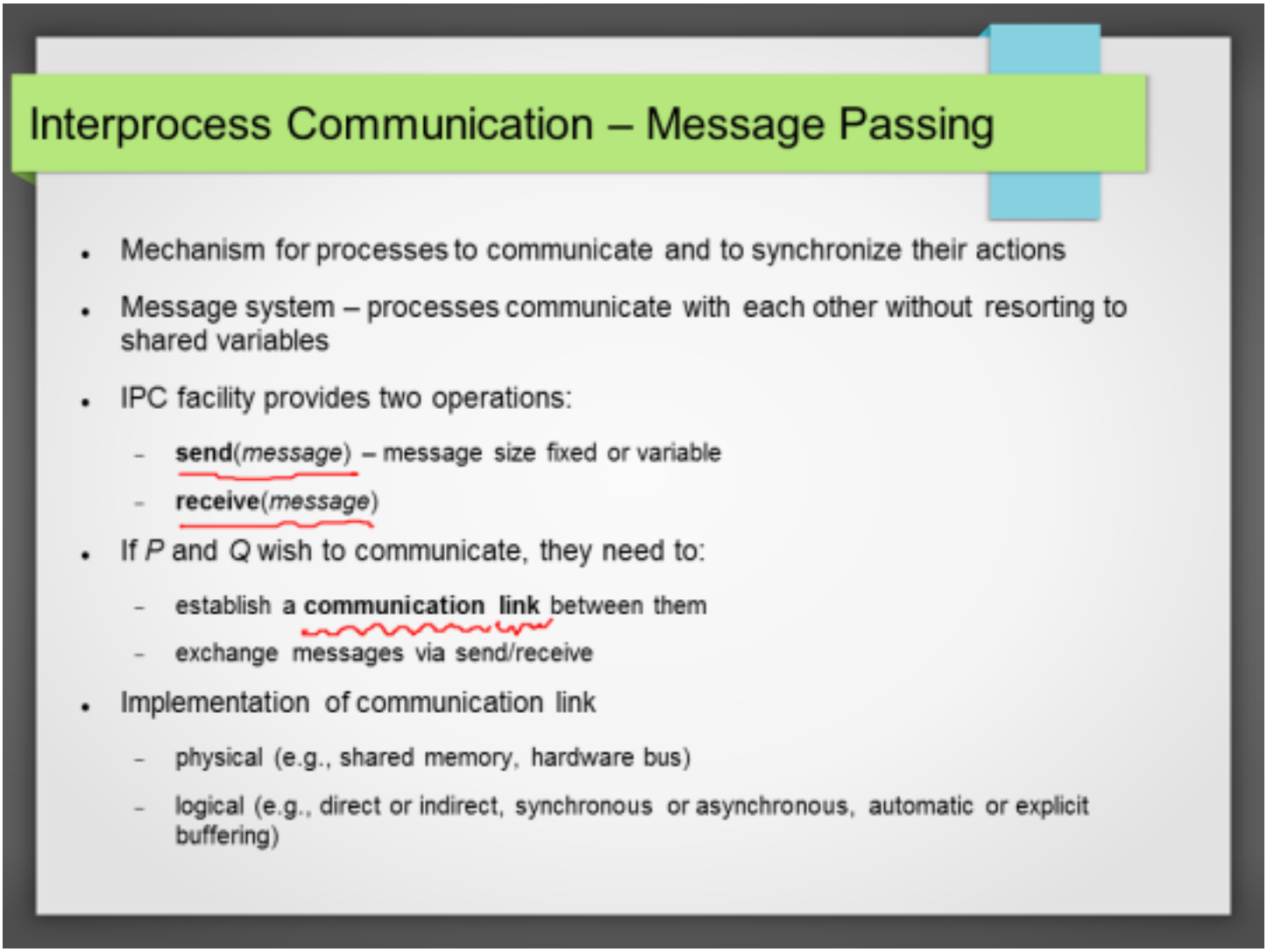
Bounded-Buffer ?? Producer



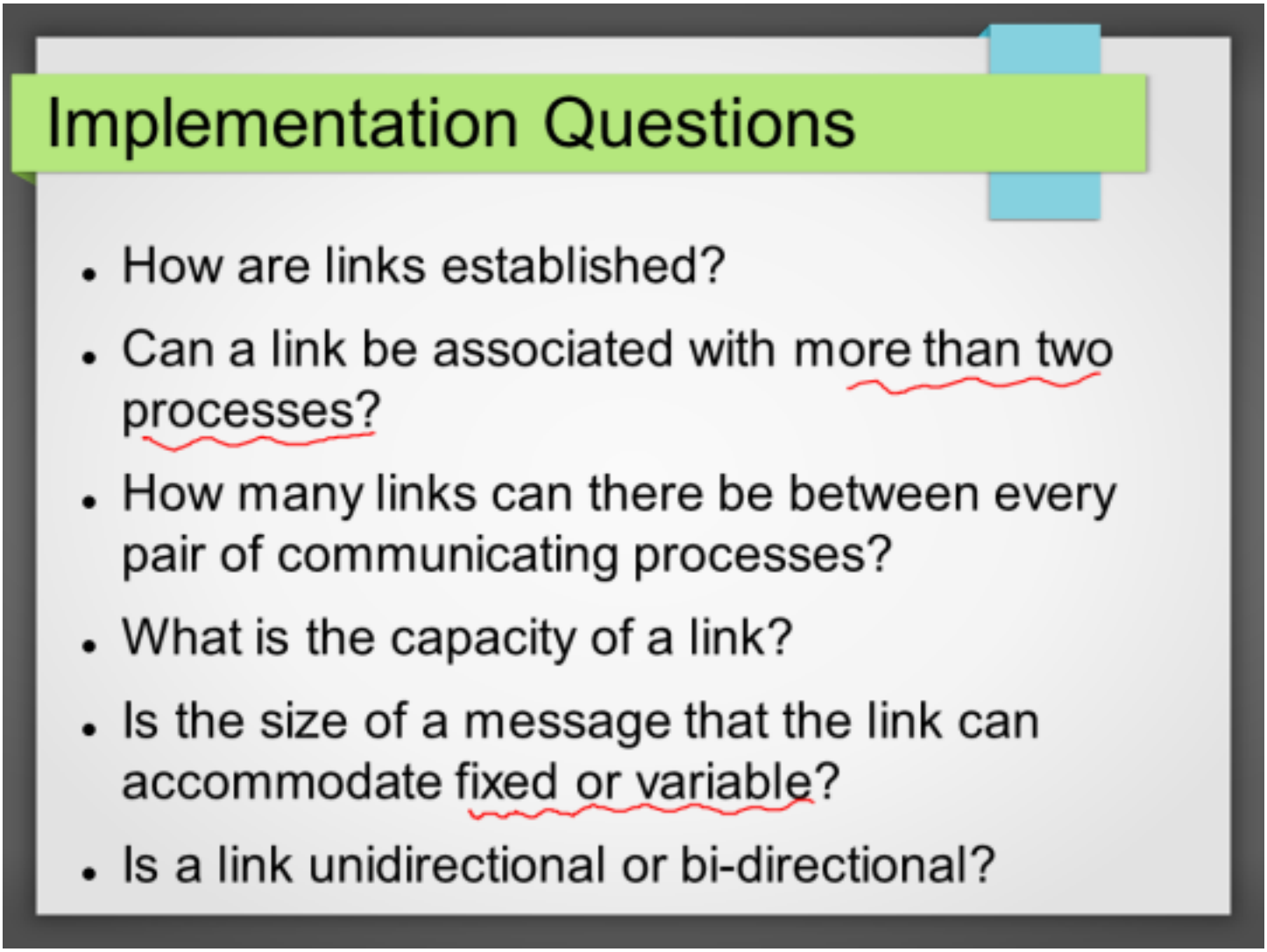
Bounded Buffer ?? Consumer



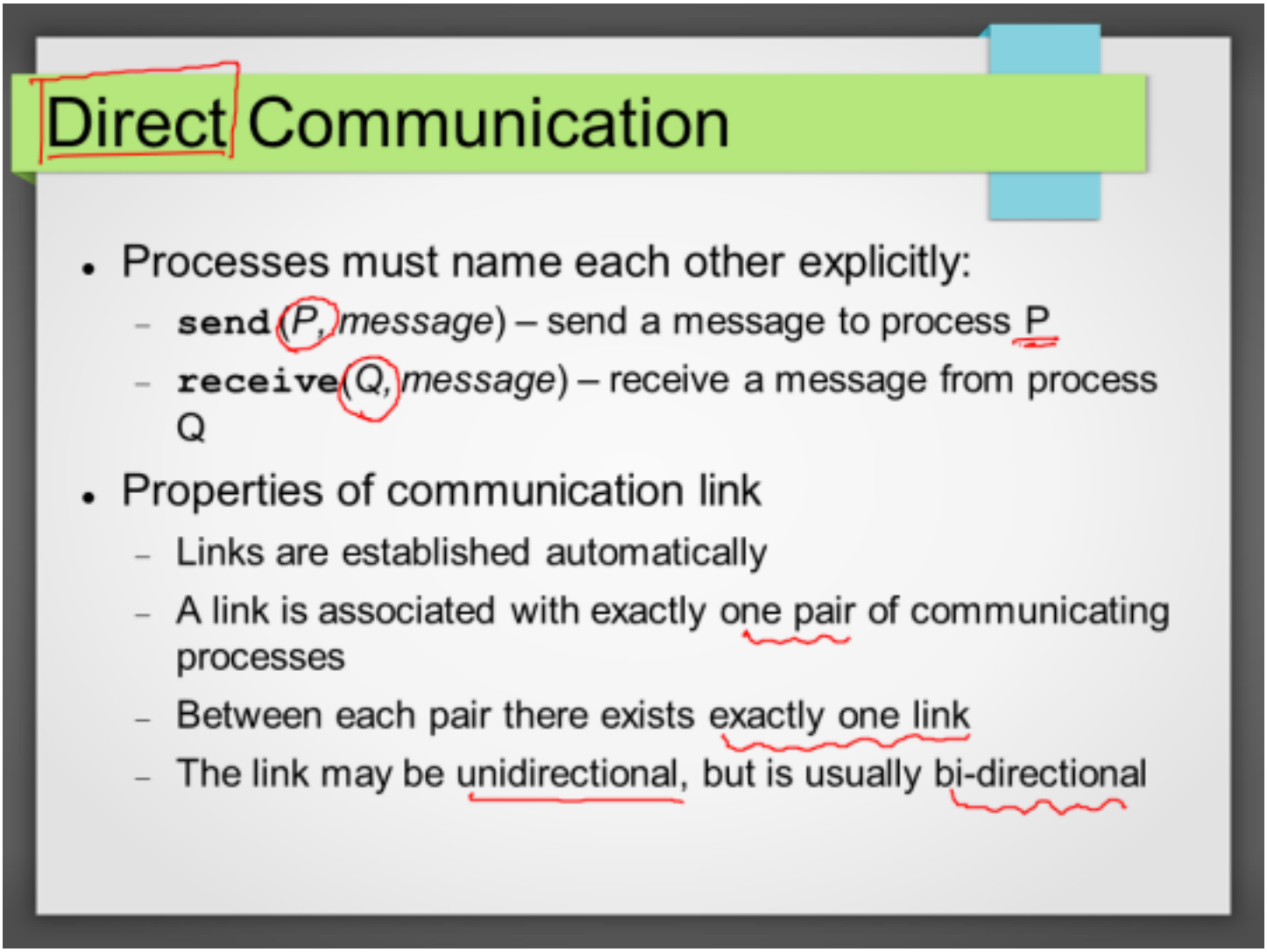
Interprocess Communication ?? Message Passing



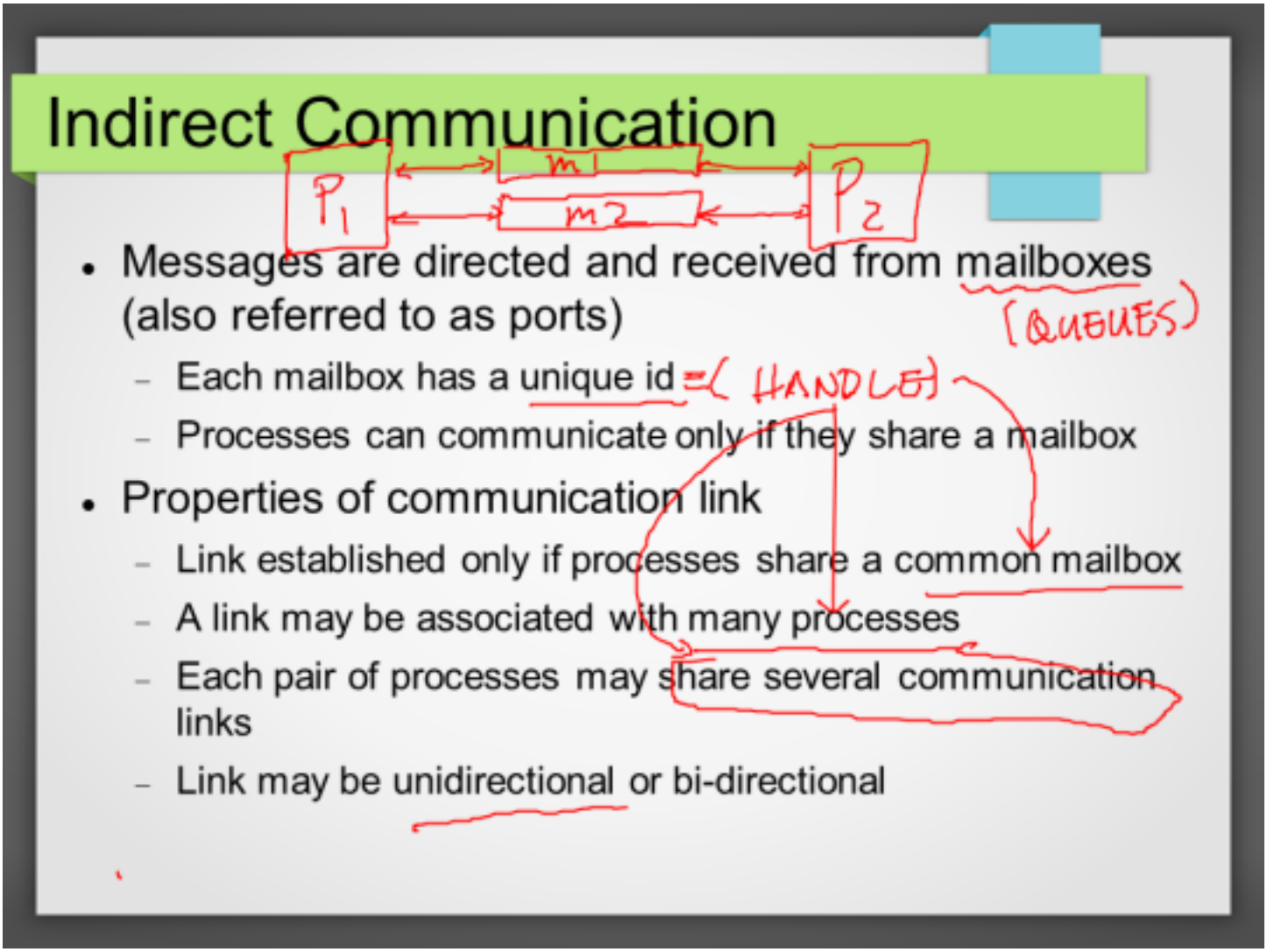
Implementation Questions



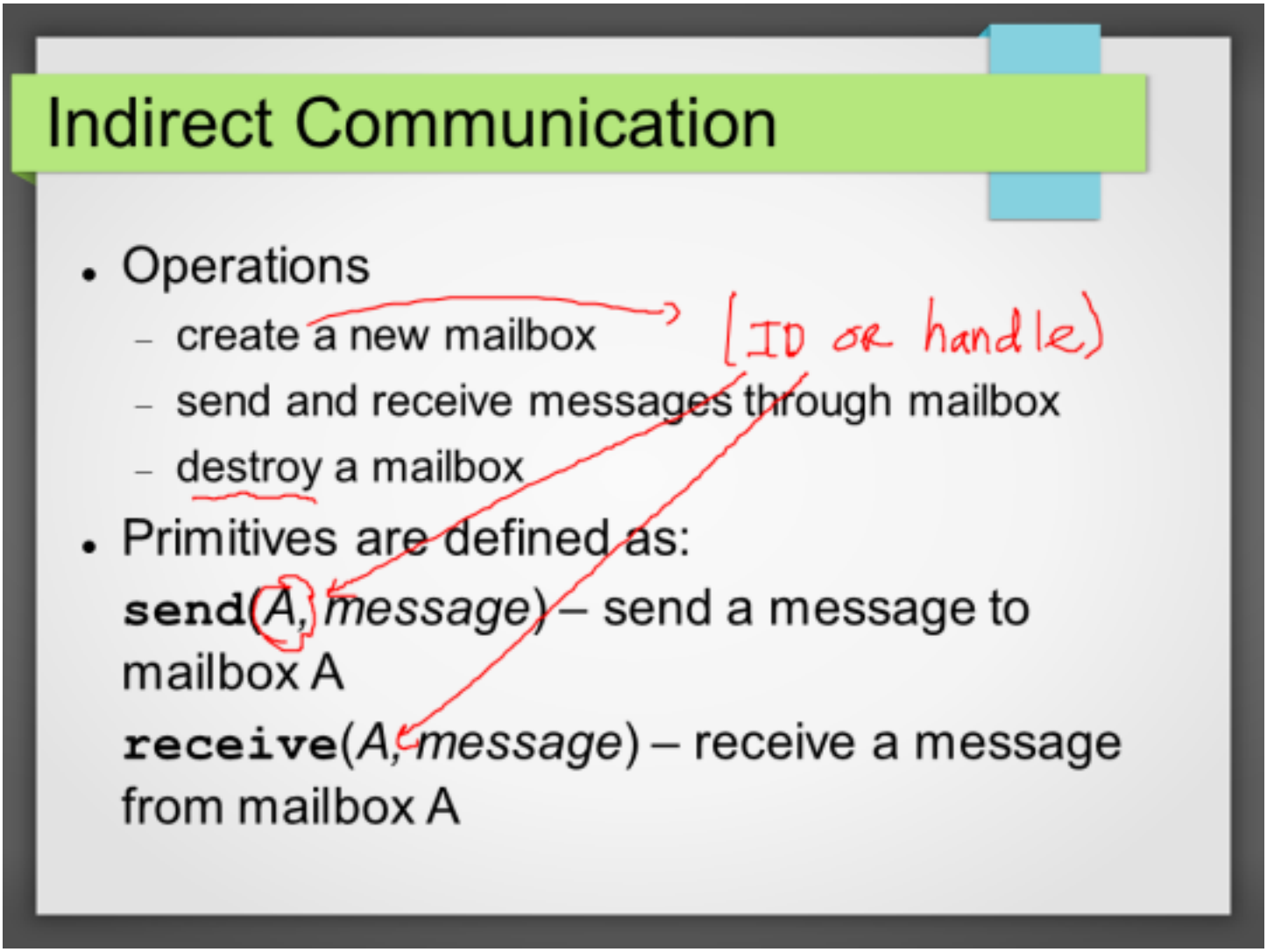
Direct Communication



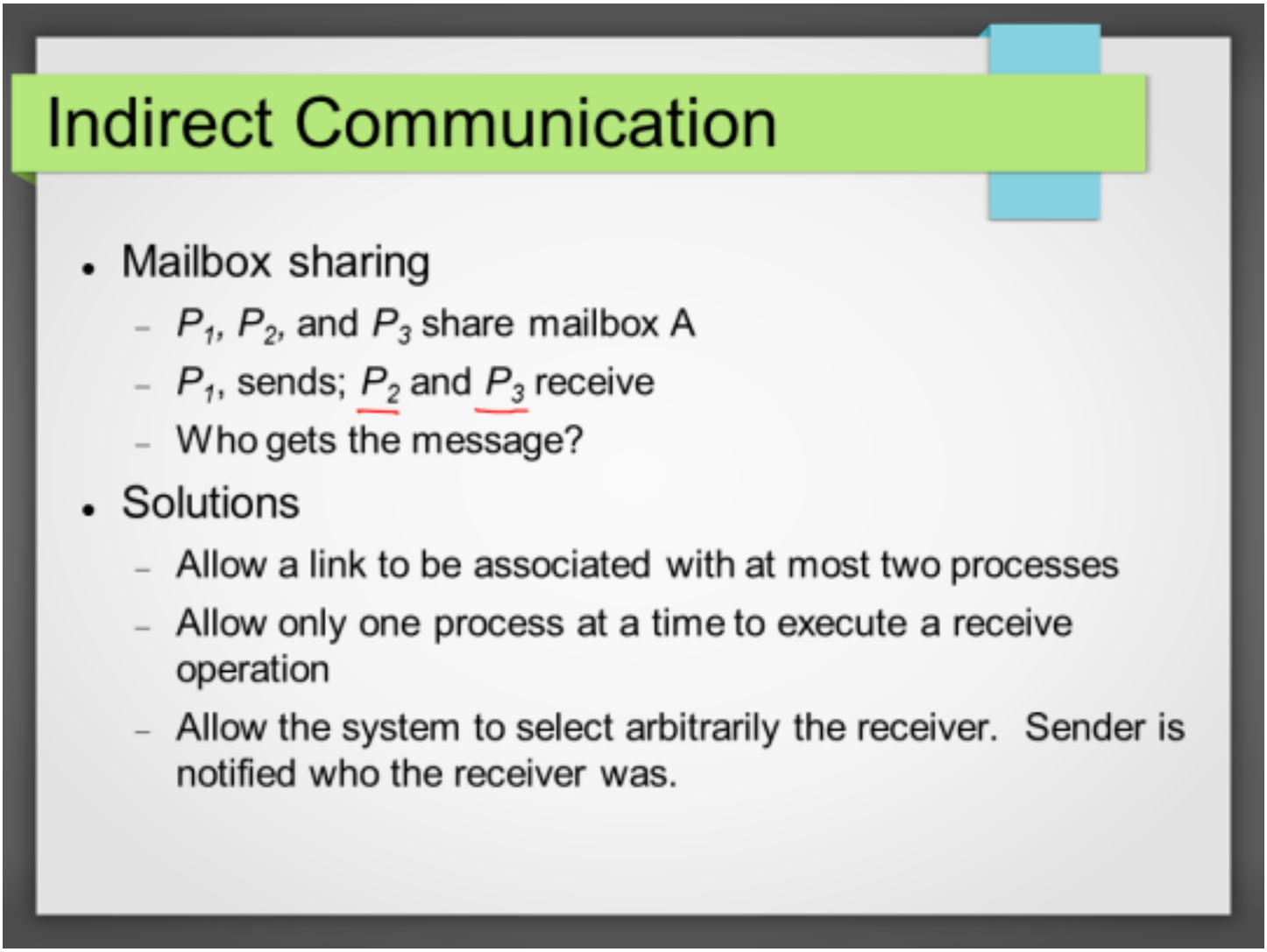
Indirect Communication



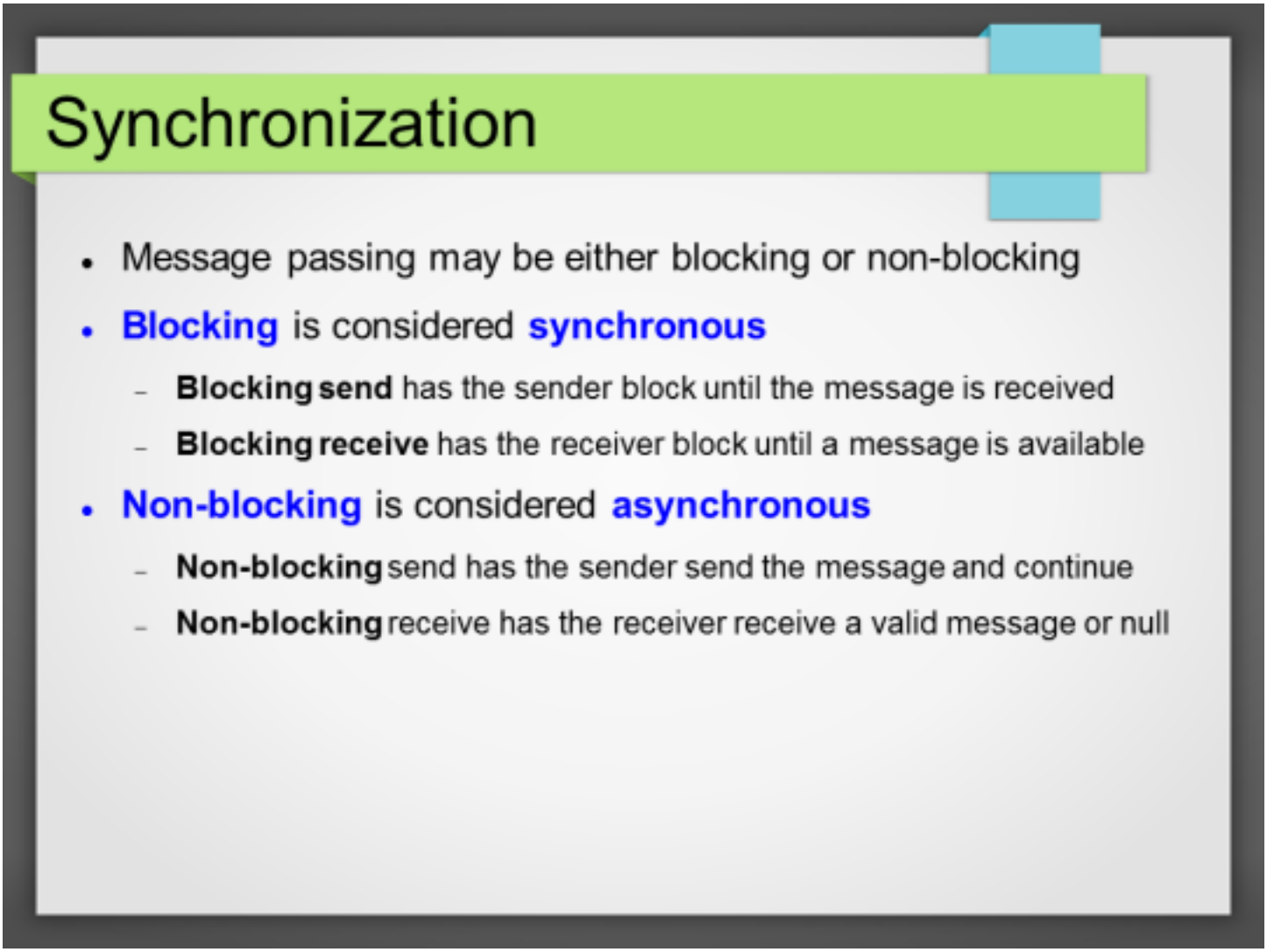
Indirect Communication



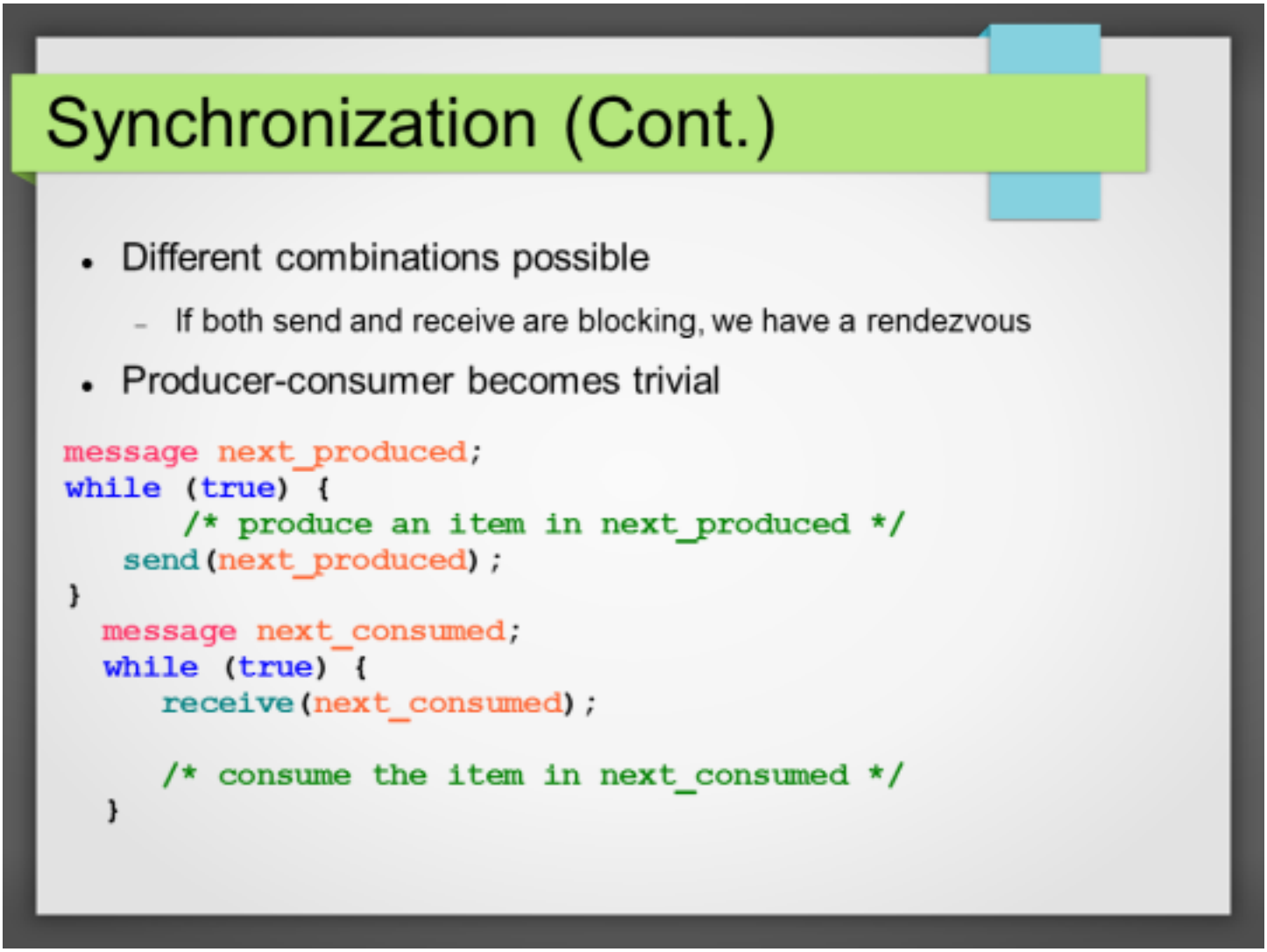
Indirect Communication



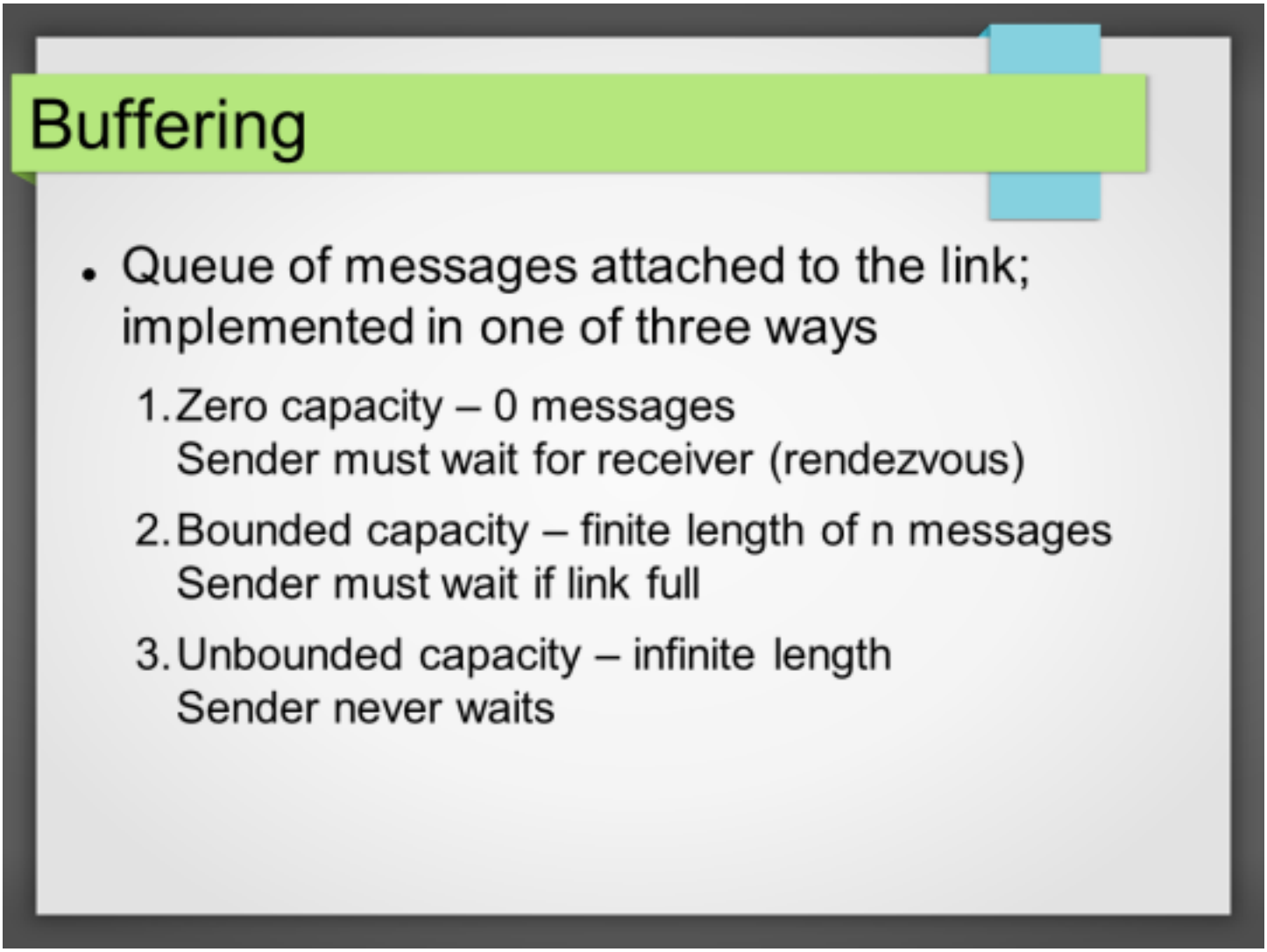
Synchronization



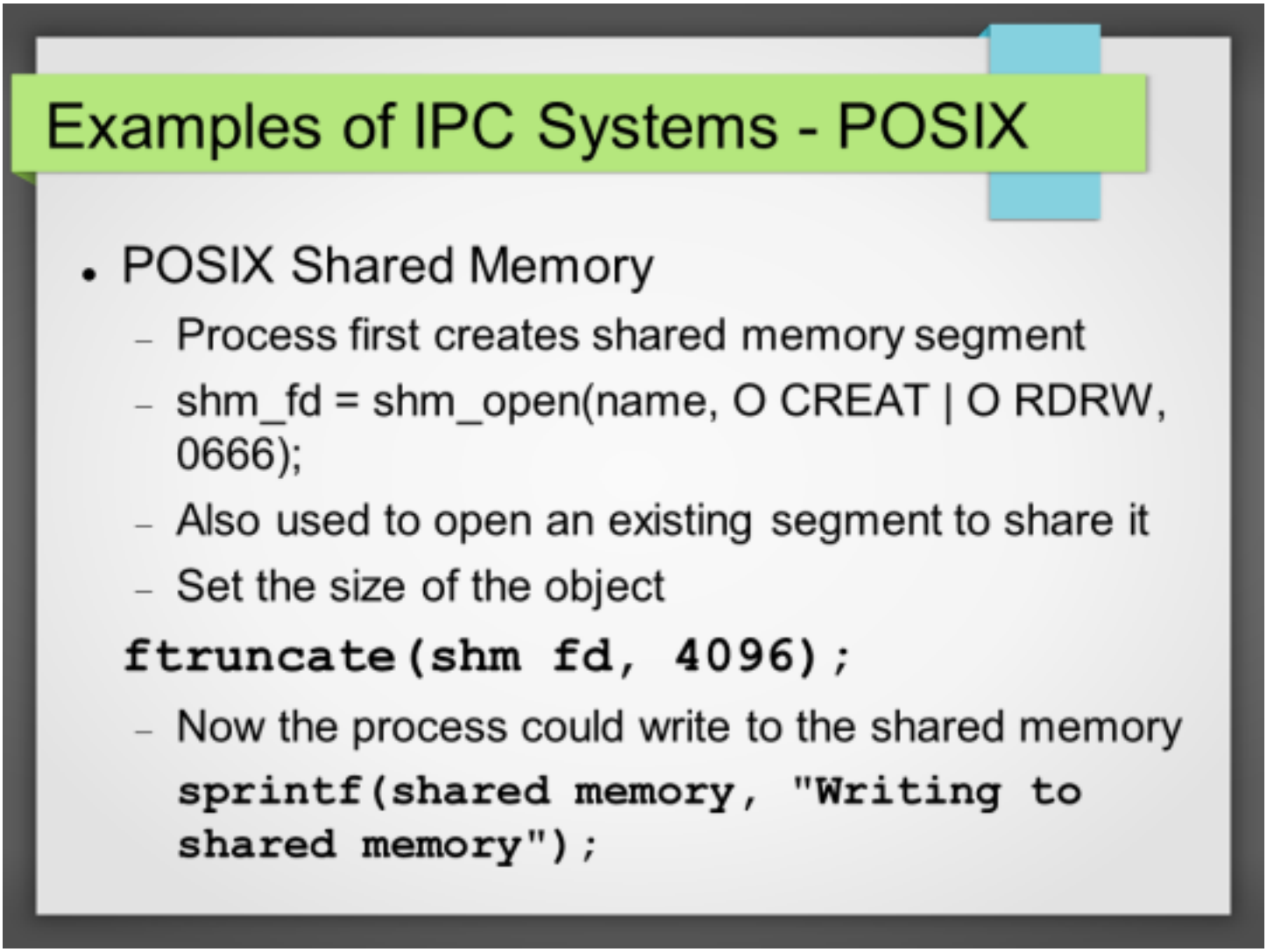
Synchronization (Cont.)



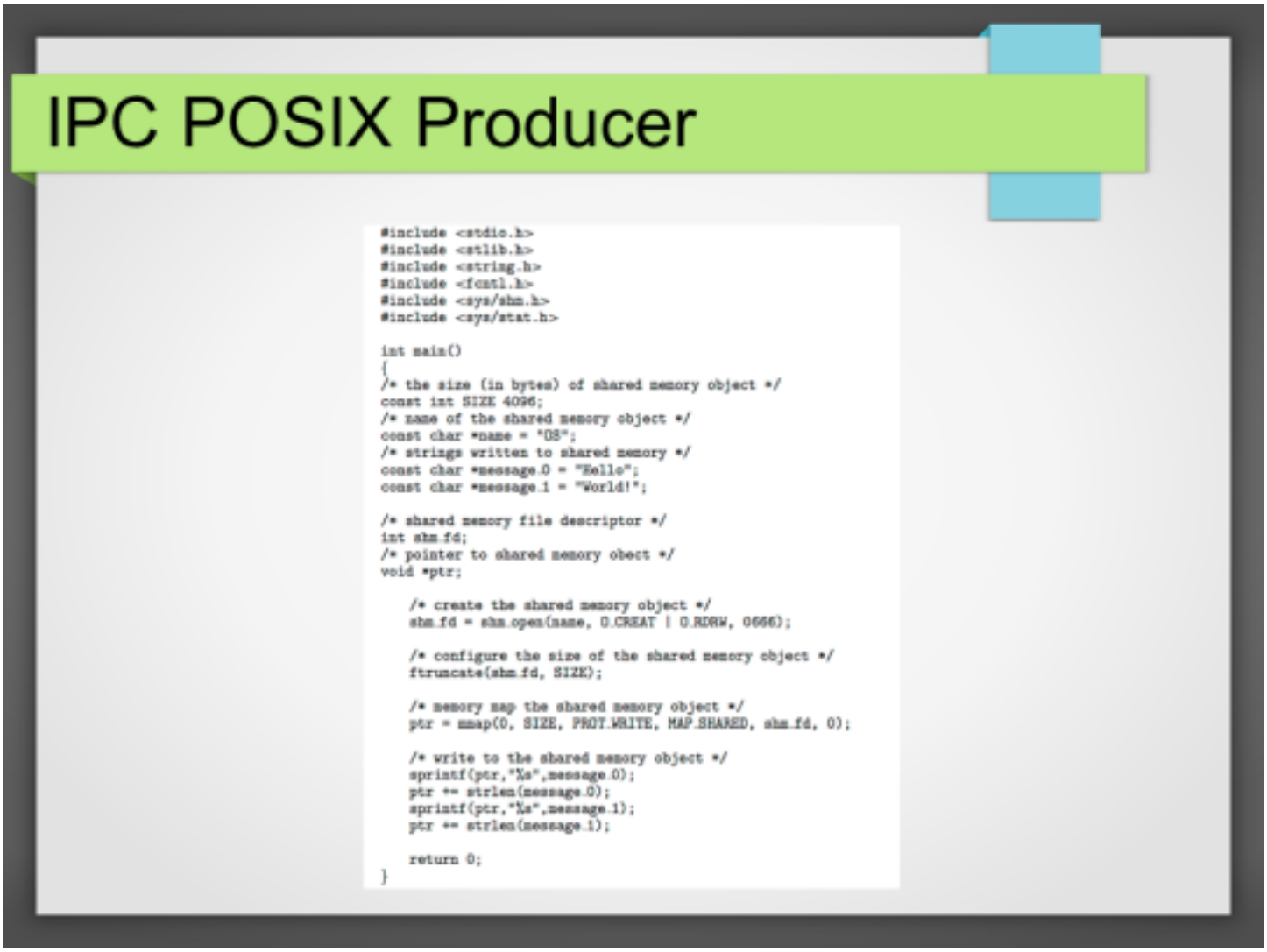
Buffering


Buffering
Queue of messages attached to the link; implemented in one of three ways
1. Zero capacity – 0 messages
Sender must wait for receiver (rendezvous)
2. Bounded capacity – finite length of n messages
Sender must wait if link full
3. Unbounded capacity – infinite length
Sender never waits

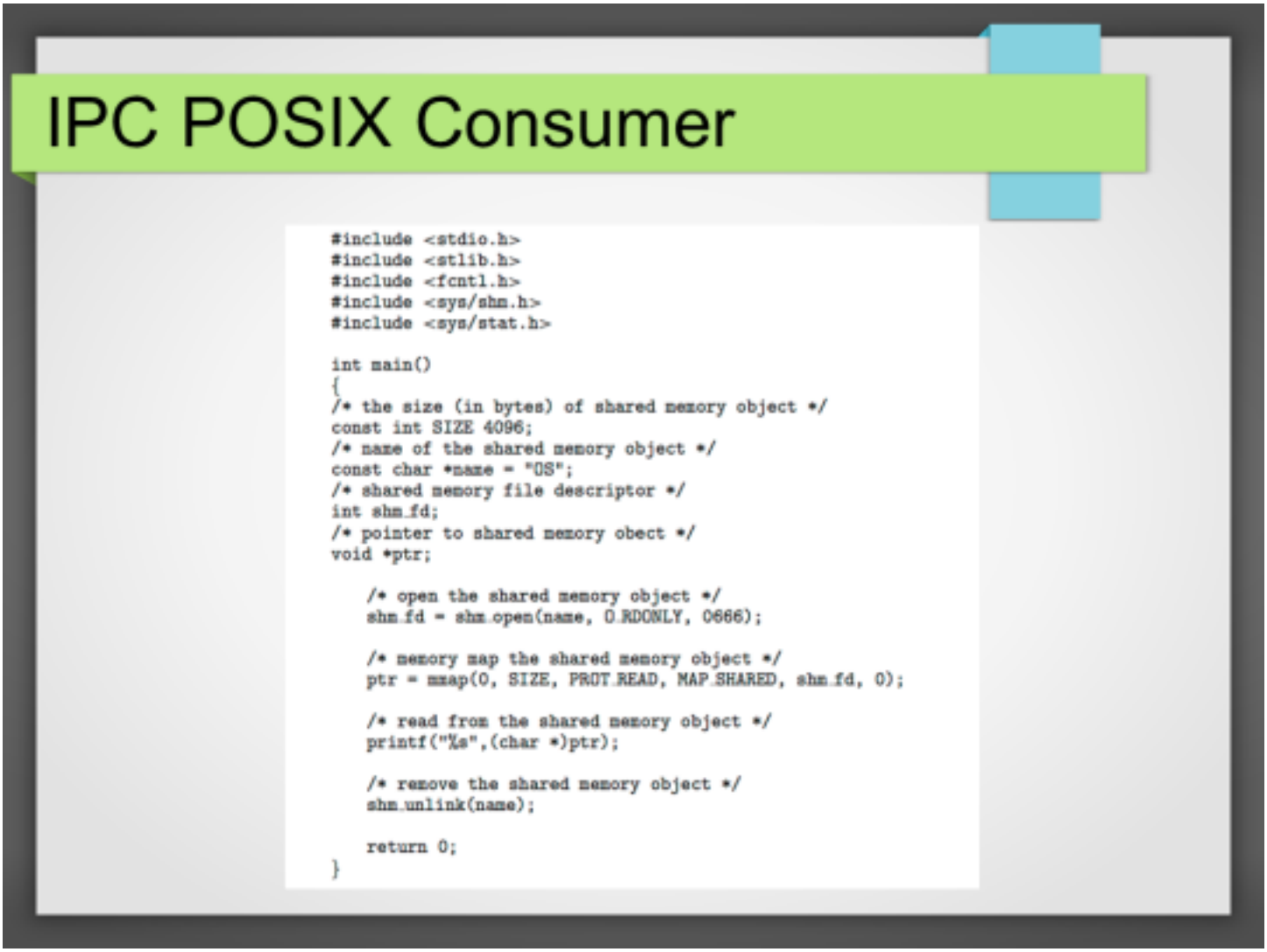
Examples of IPC Systems - POSIX



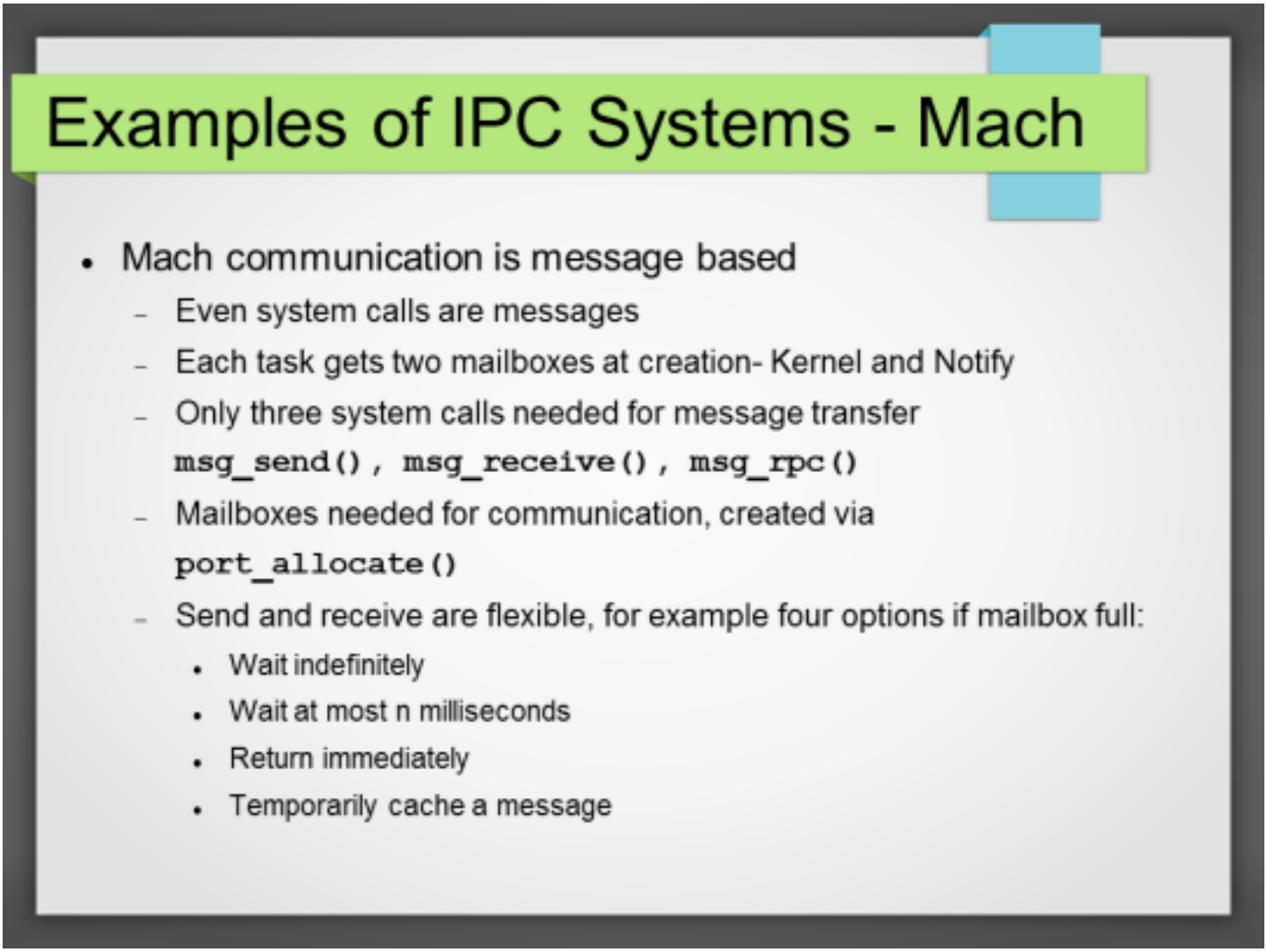
IPC POSIX Producer



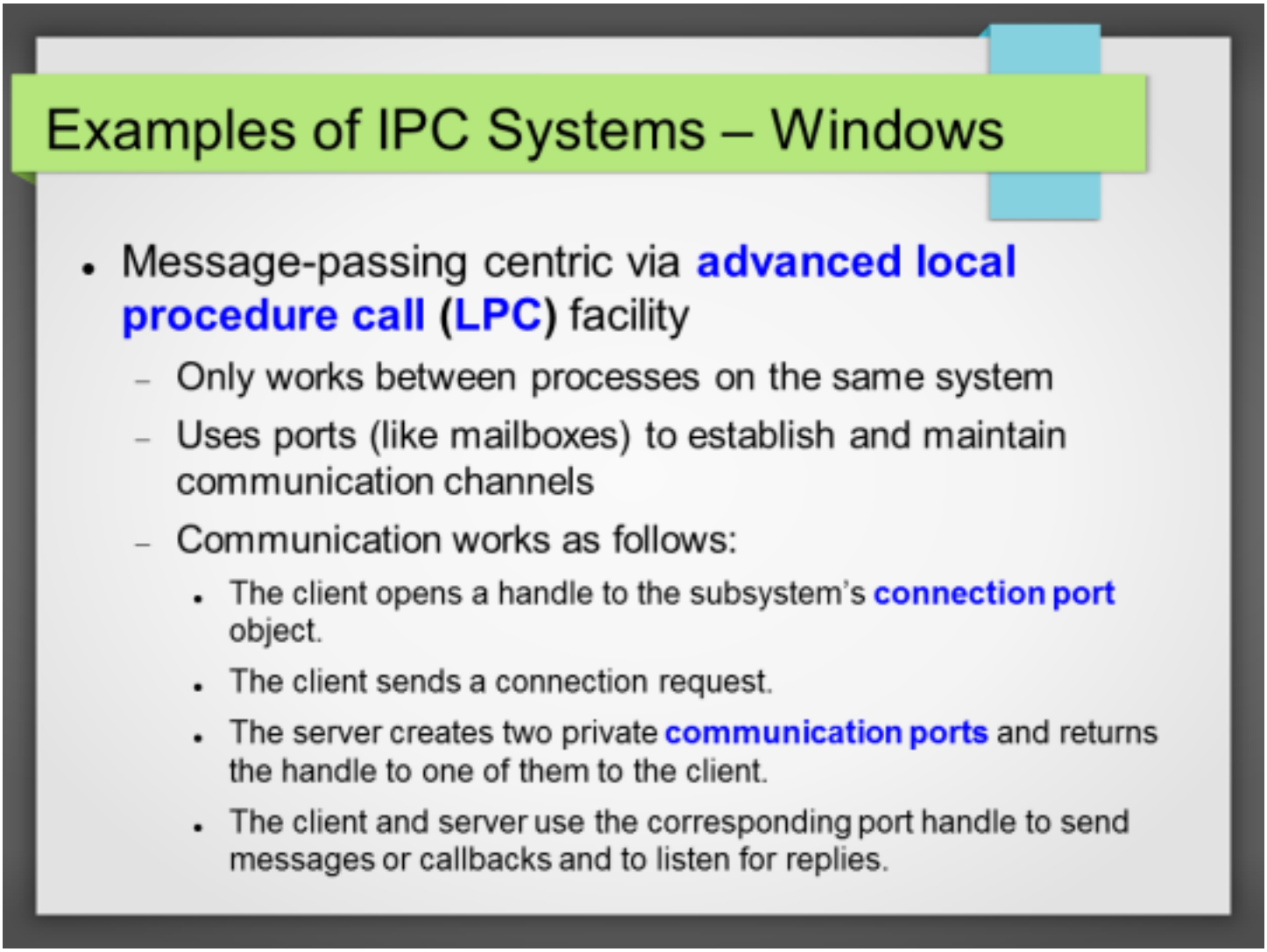
IPC POSIX Consumer



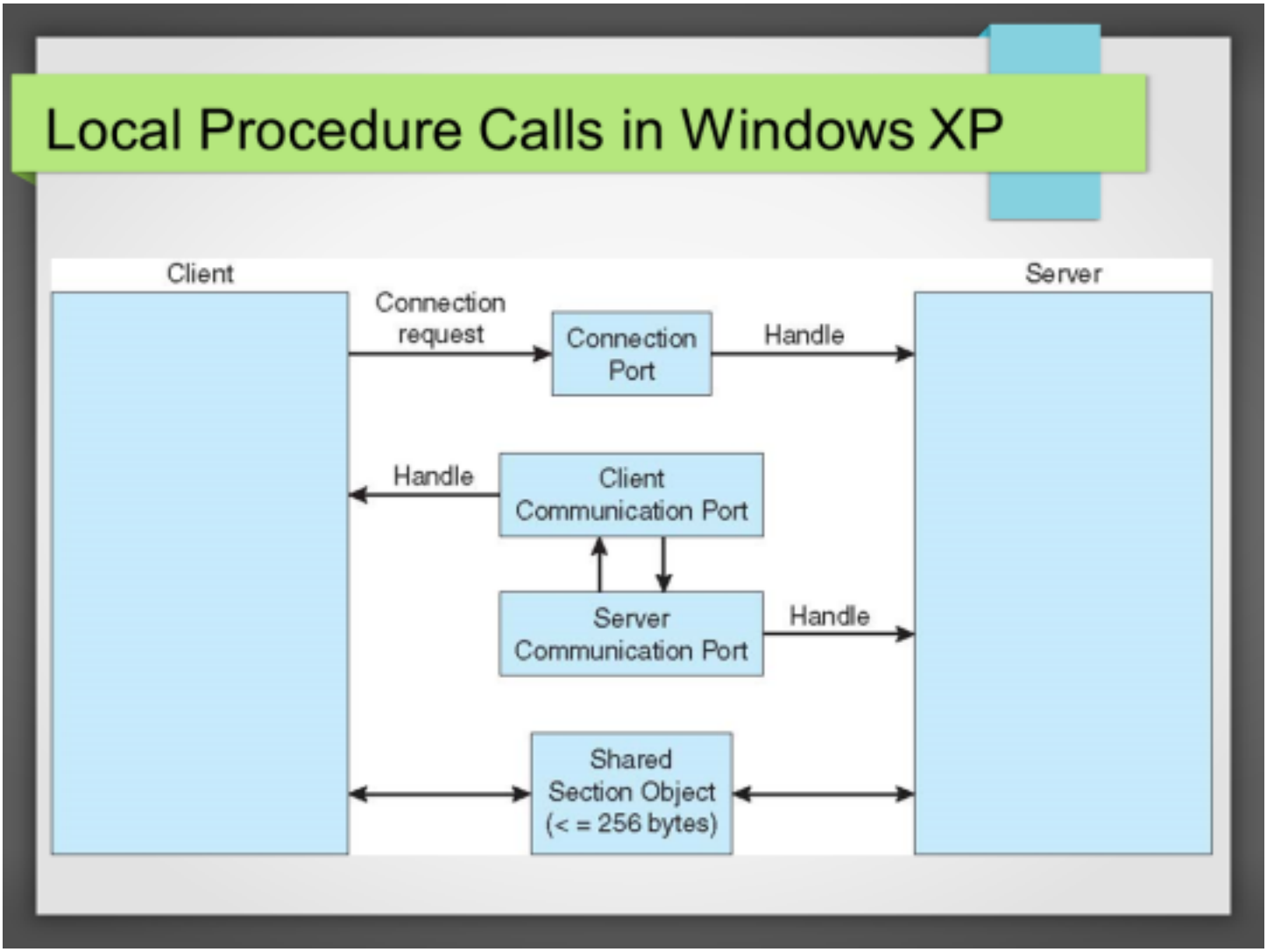
Examples of IPC Systems - Mach



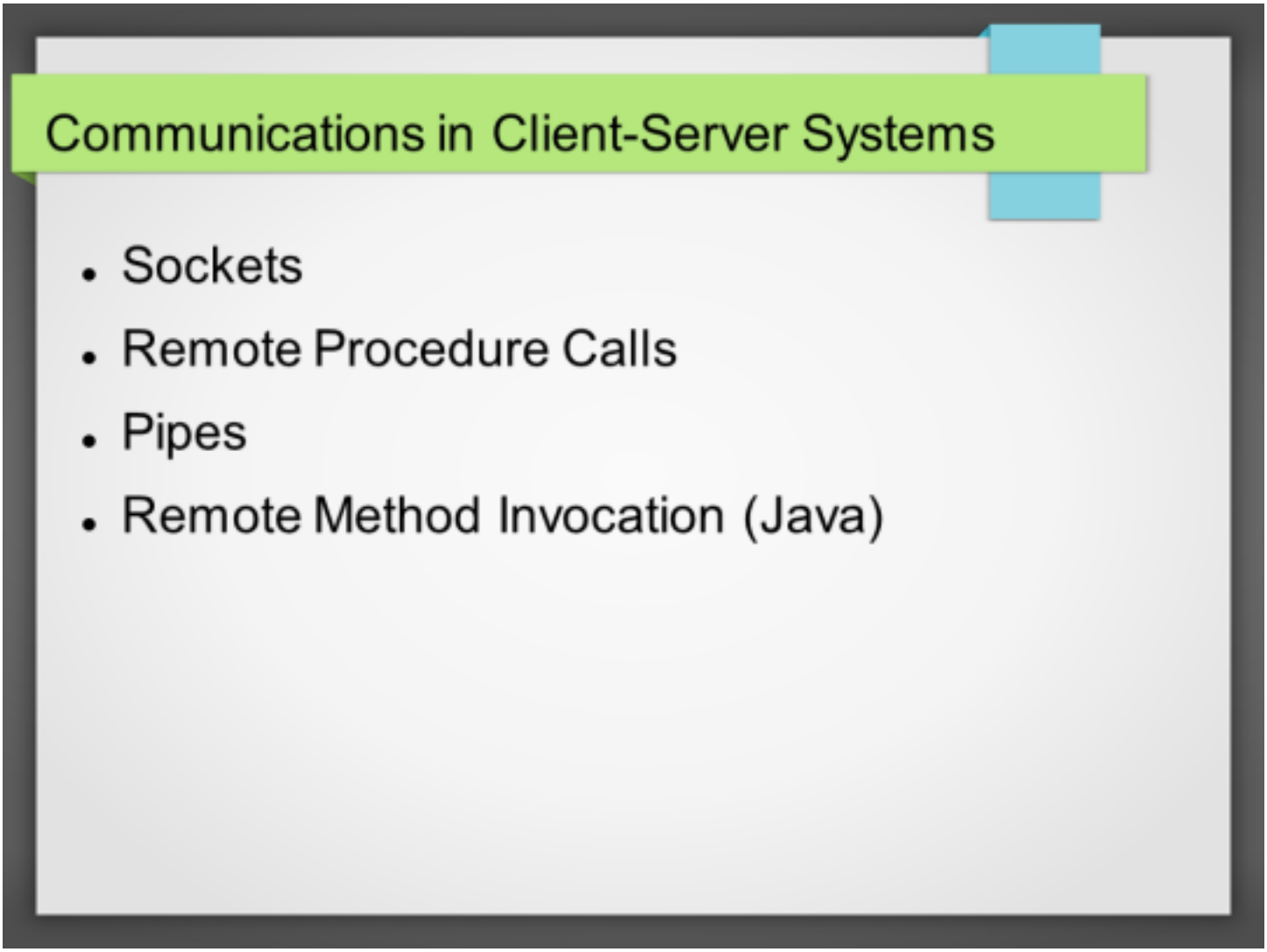
Examples of IPC Systems ?? Windows



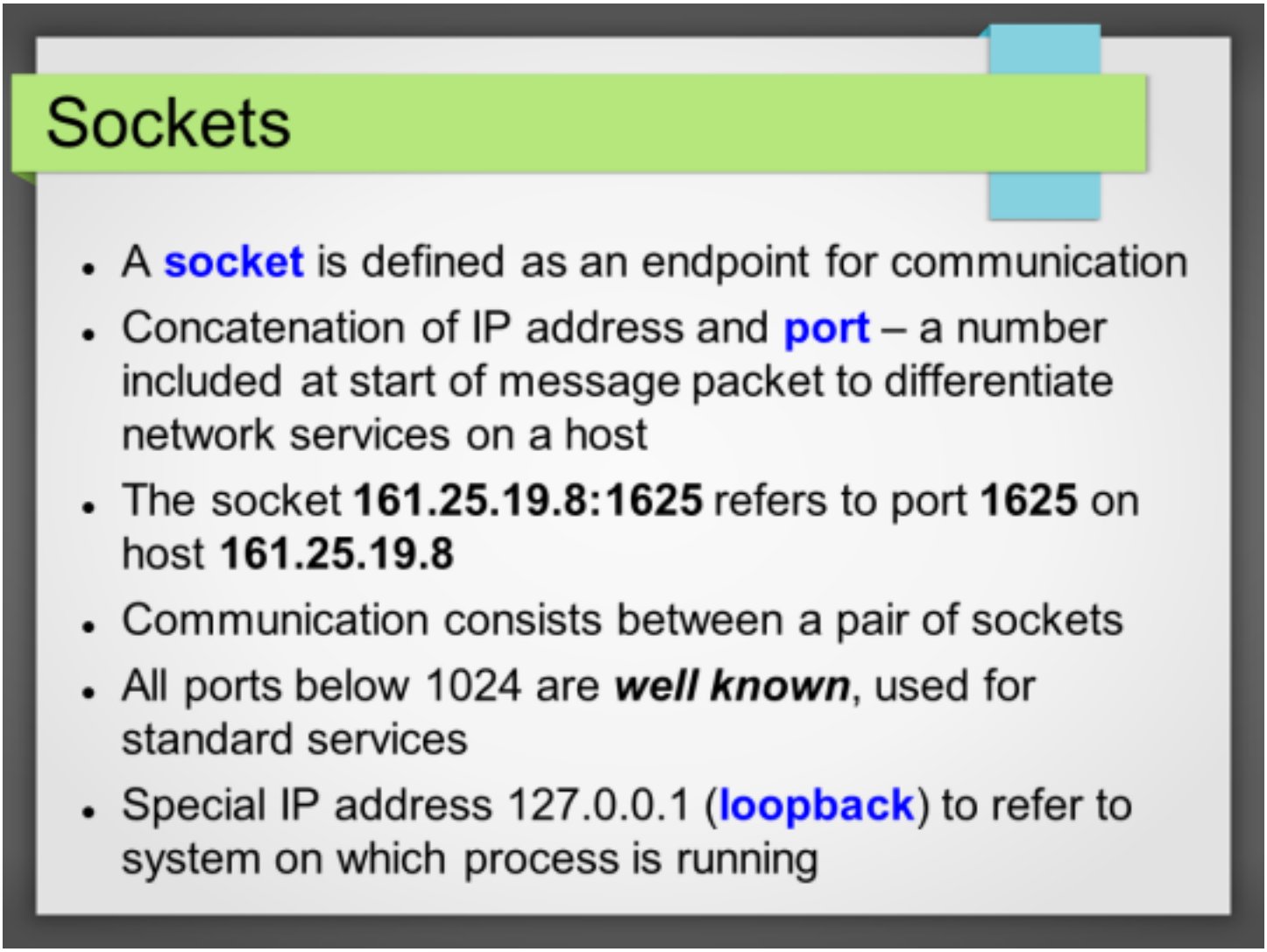
Local Procedure Calls in Windows XP



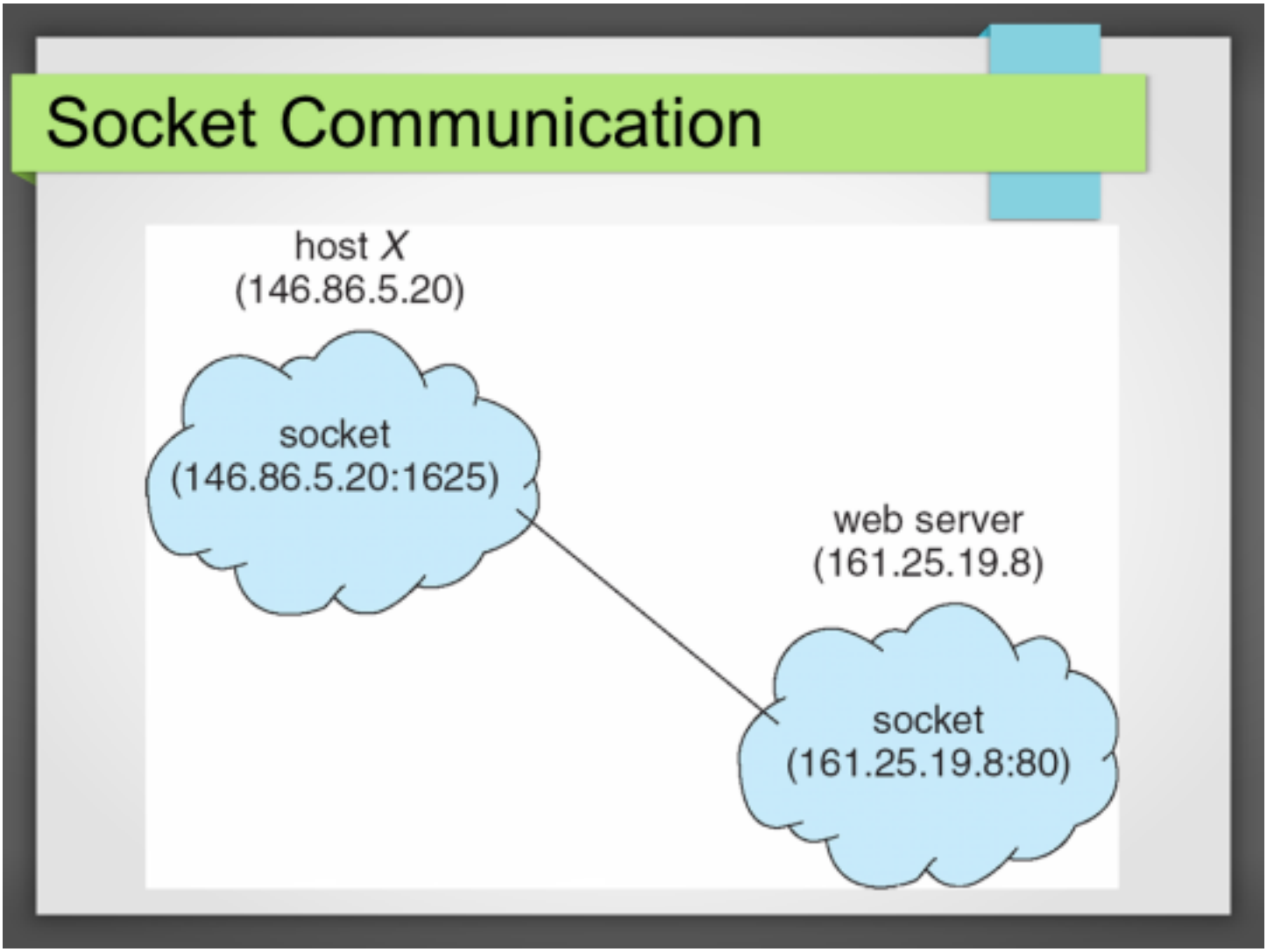
Communications in Client-Server Systems



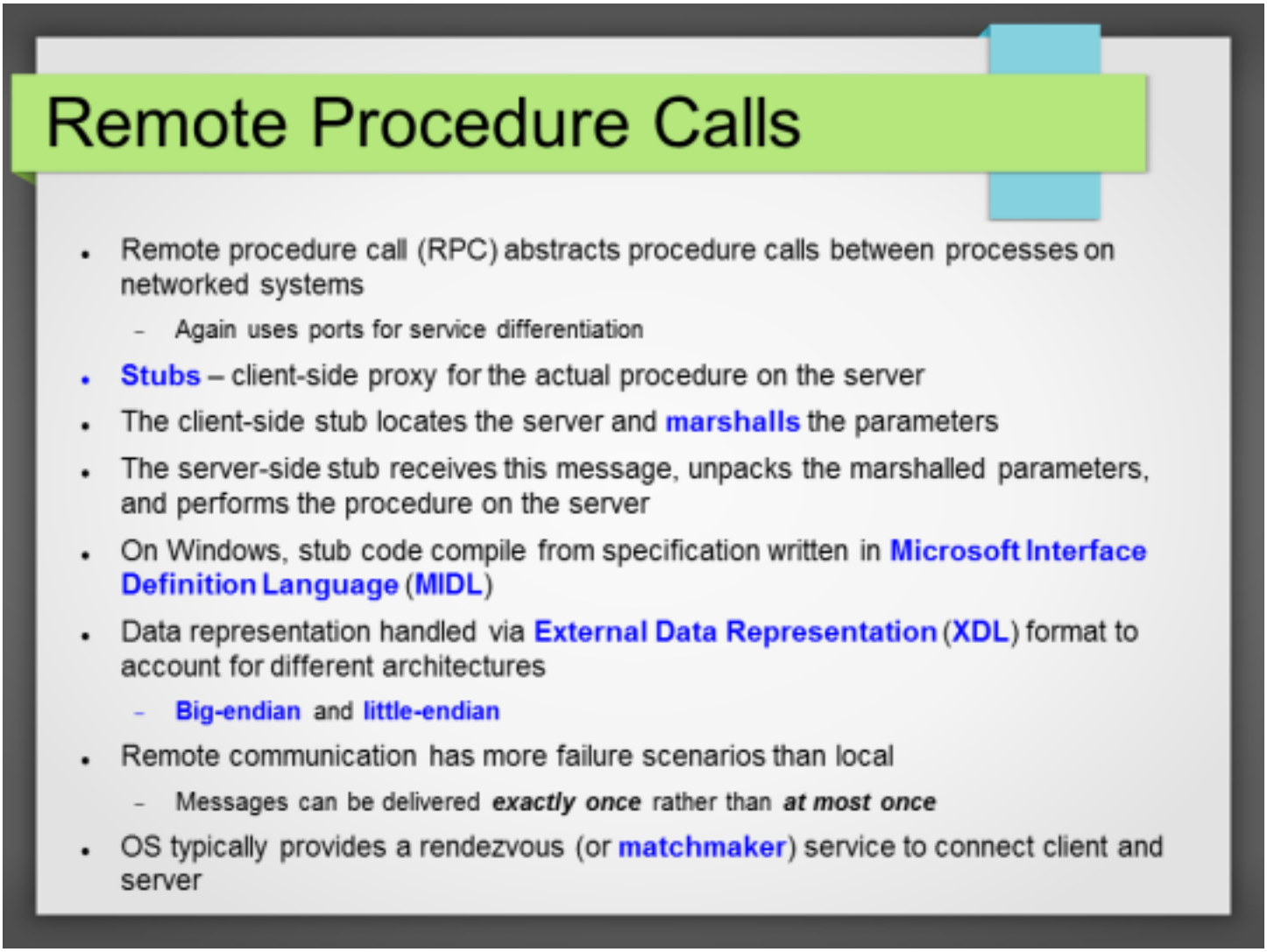
Sockets


Sockets
A socket is defined as an endpoint for communication
Concatenation of IP address and port – a number included at start of message packet to differentiate network services on a host
The socket 161.25.19.8:1625 refers to port 1625 on host 161.25.19.8
Communication consists between a pair of sockets
All ports below 1024 are well known, used for standard services
Special IP address 127.0.0.1 (loopback) to refer to system on which process is running

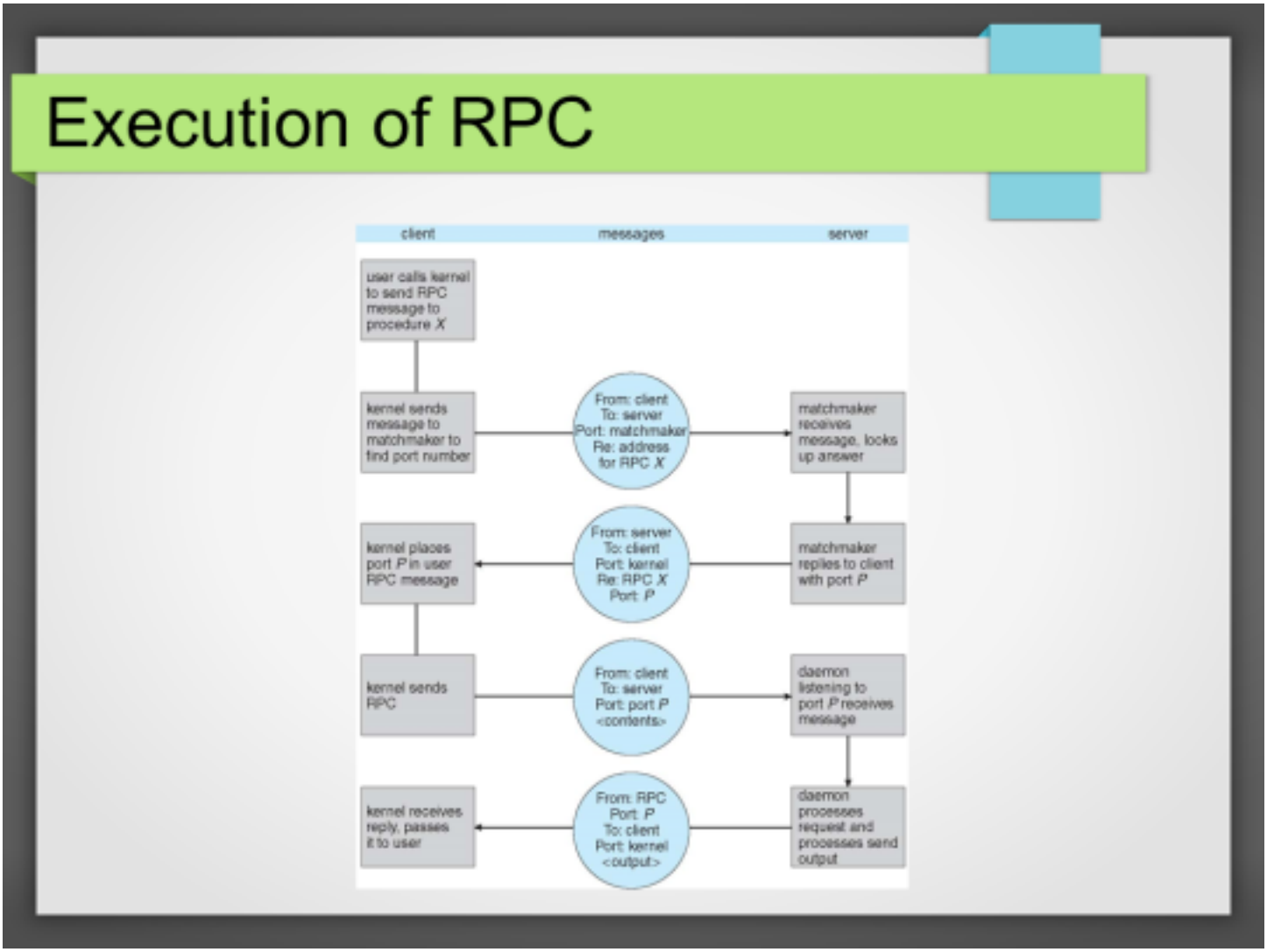
Socket Communication



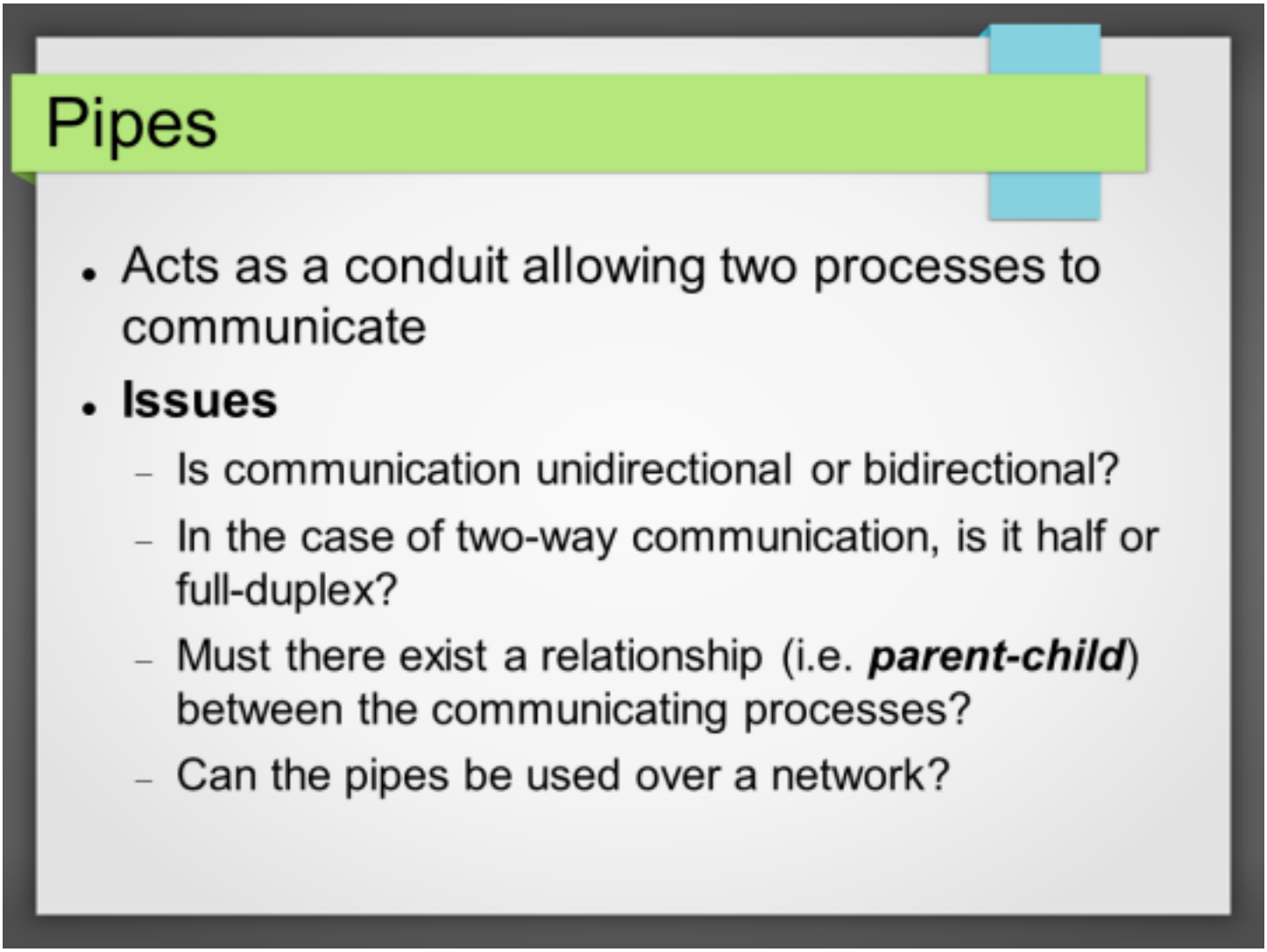
Remote Procedure Calls



Execution of RPC



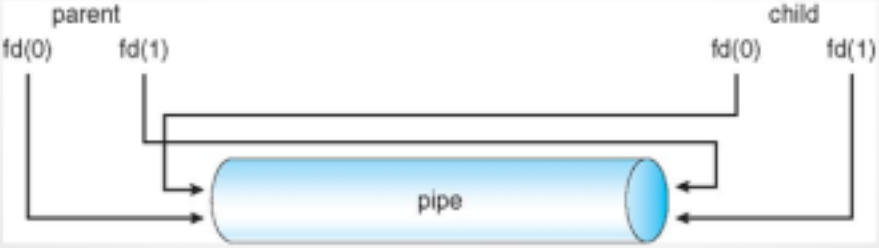
Pipes


Pipes
Acts as a conduit allowing two processes to communicate
Issues
Is communication unidirectional or bidirectional?
In the case of two-way communication, is it half or full-duplex?
Must there exist a relationship (i.e. parent-child) between the communicating processes?
Can the pipes be used over a network?

Ordinary Pipes

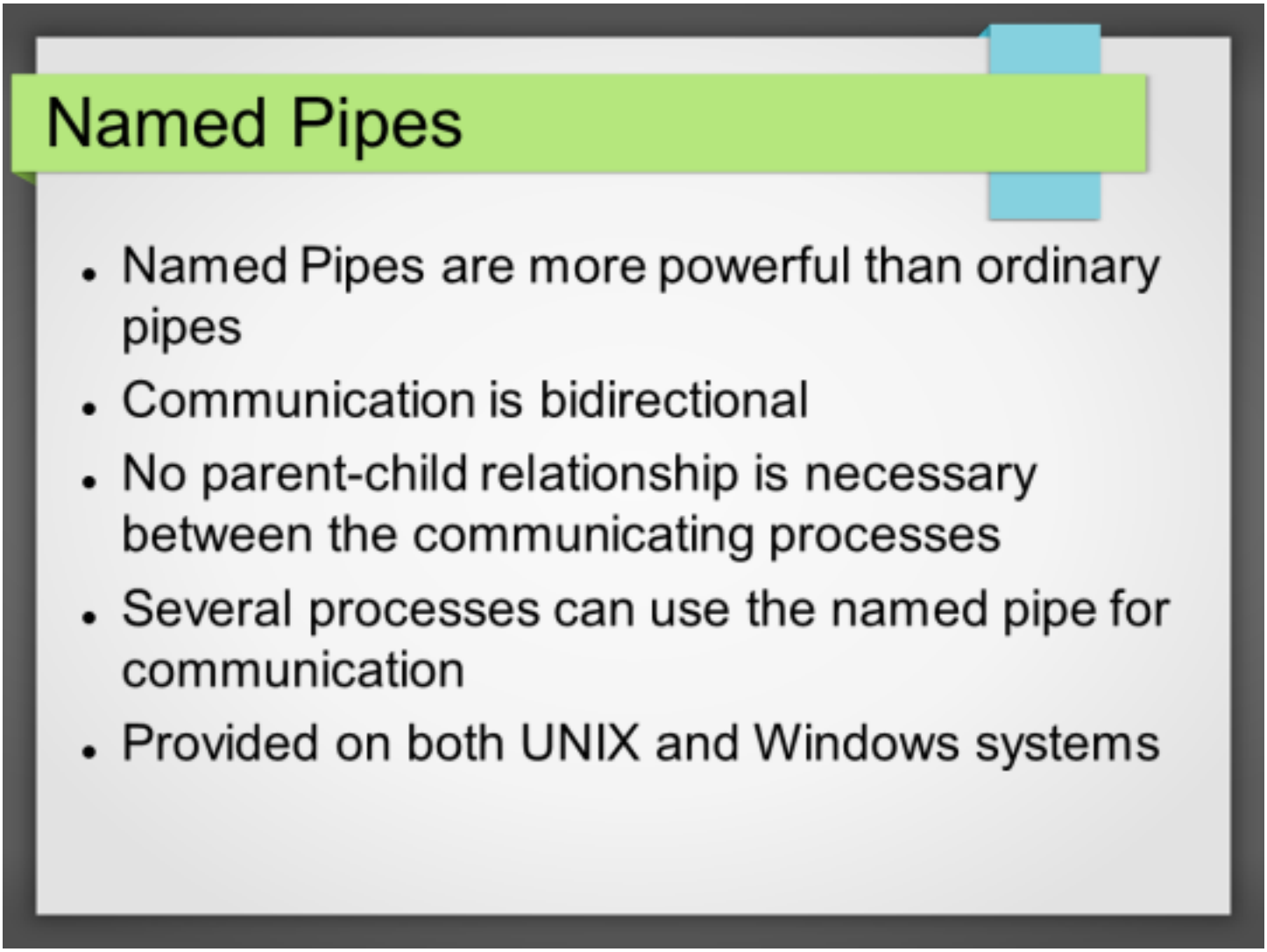
Ordinary Pipes

- Ordinary Pipes allow communication in standard producer-consumer style
- Producer writes to one end (the **write-end** of the pipe)
- Consumer reads from the other end (the **read-end** of the pipe)
- Ordinary pipes are therefore unidirectional
- Require parent-child relationship between communicating processes



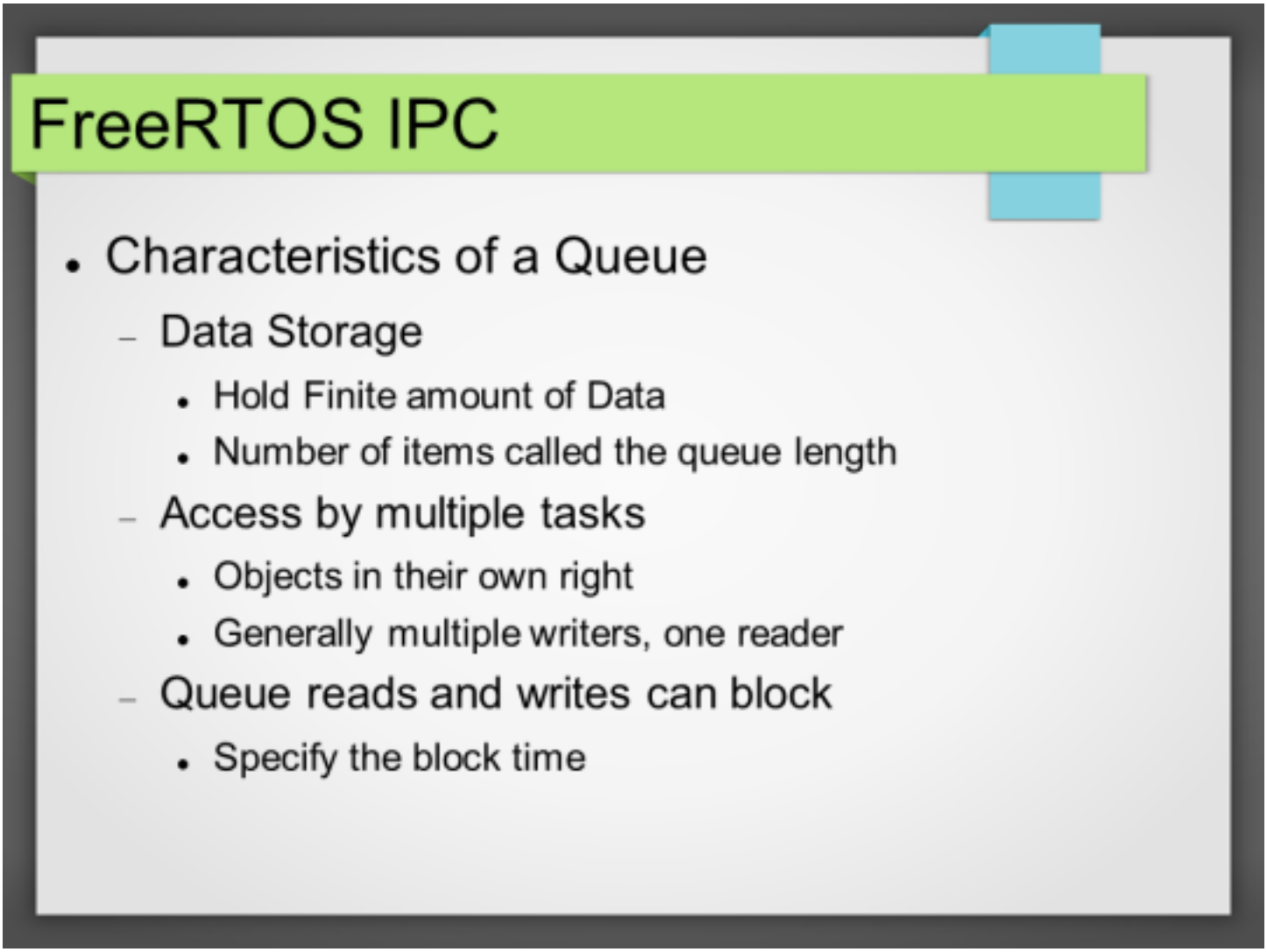
- Windows calls these **anonymous pipes**
- See Unix and Windows code samples in textbook

Named Pipes

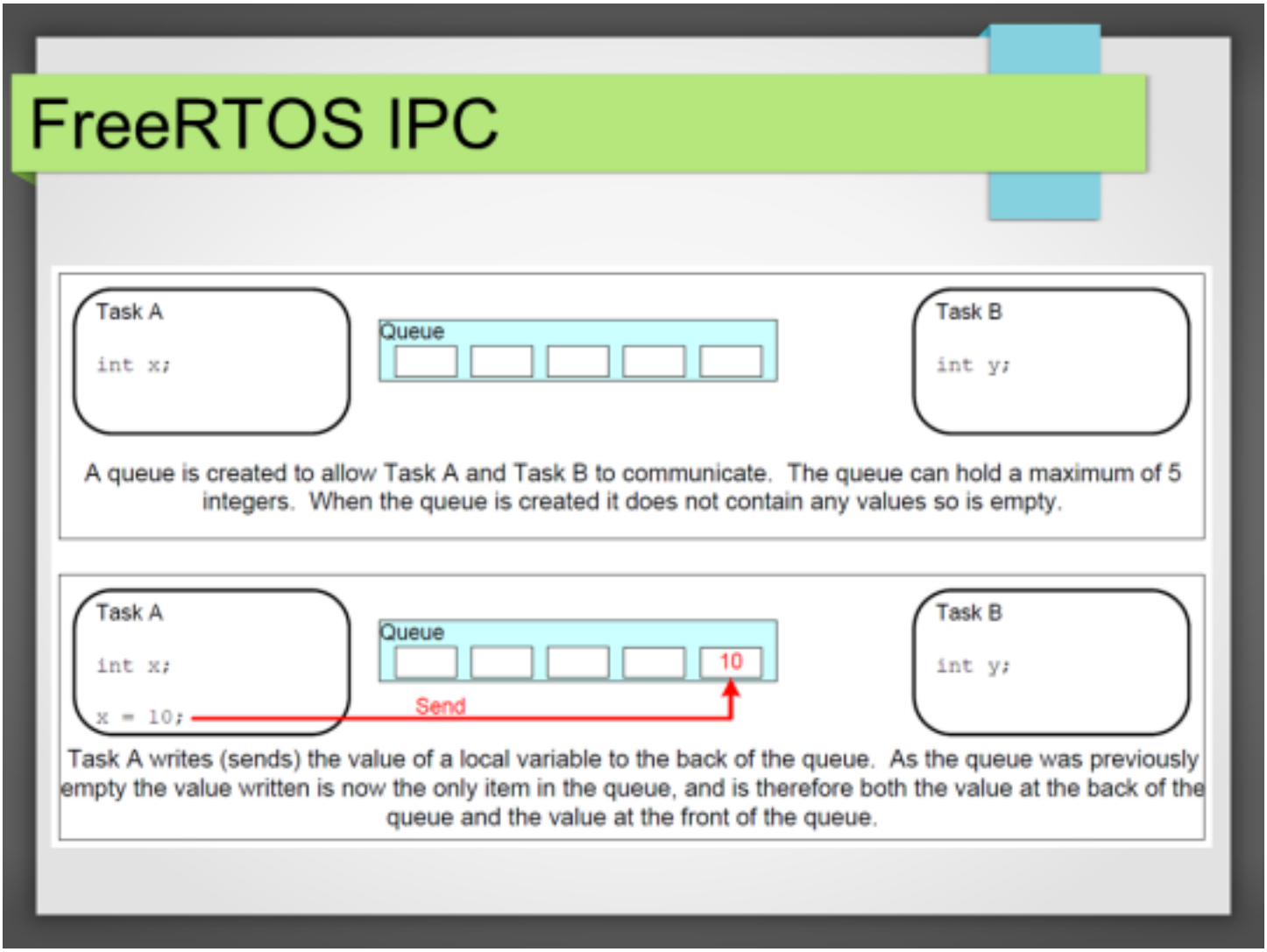

Named Pipes
Named Pipes are more powerful than ordinary pipes
Communication is bidirectional
No parent-child relationship is necessary between the communicating processes
Several processes can use the named pipe for communication
Provided on both UNIX and Windows systems

FreeRTOS IPC

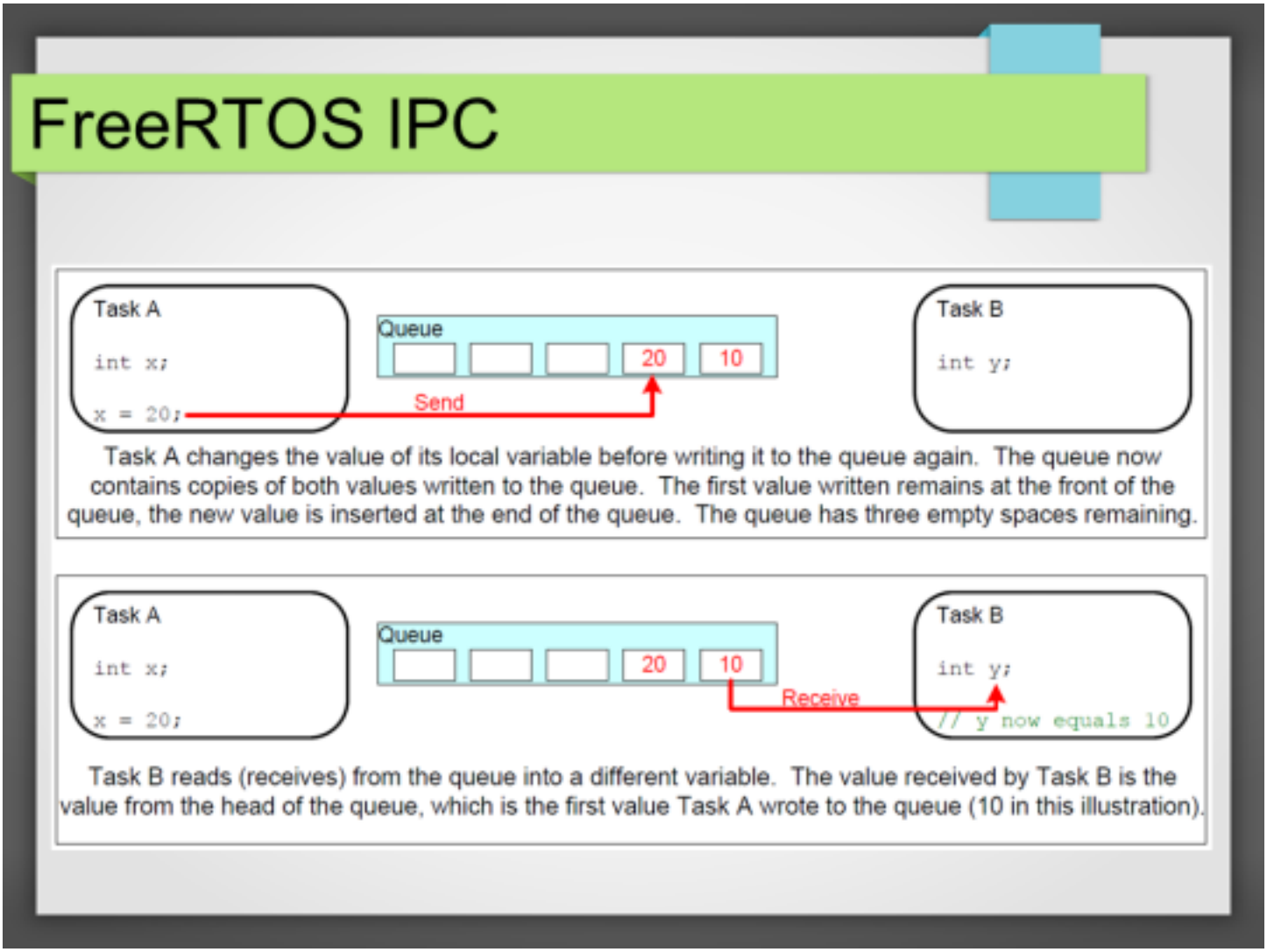
FreeRTOS IPC



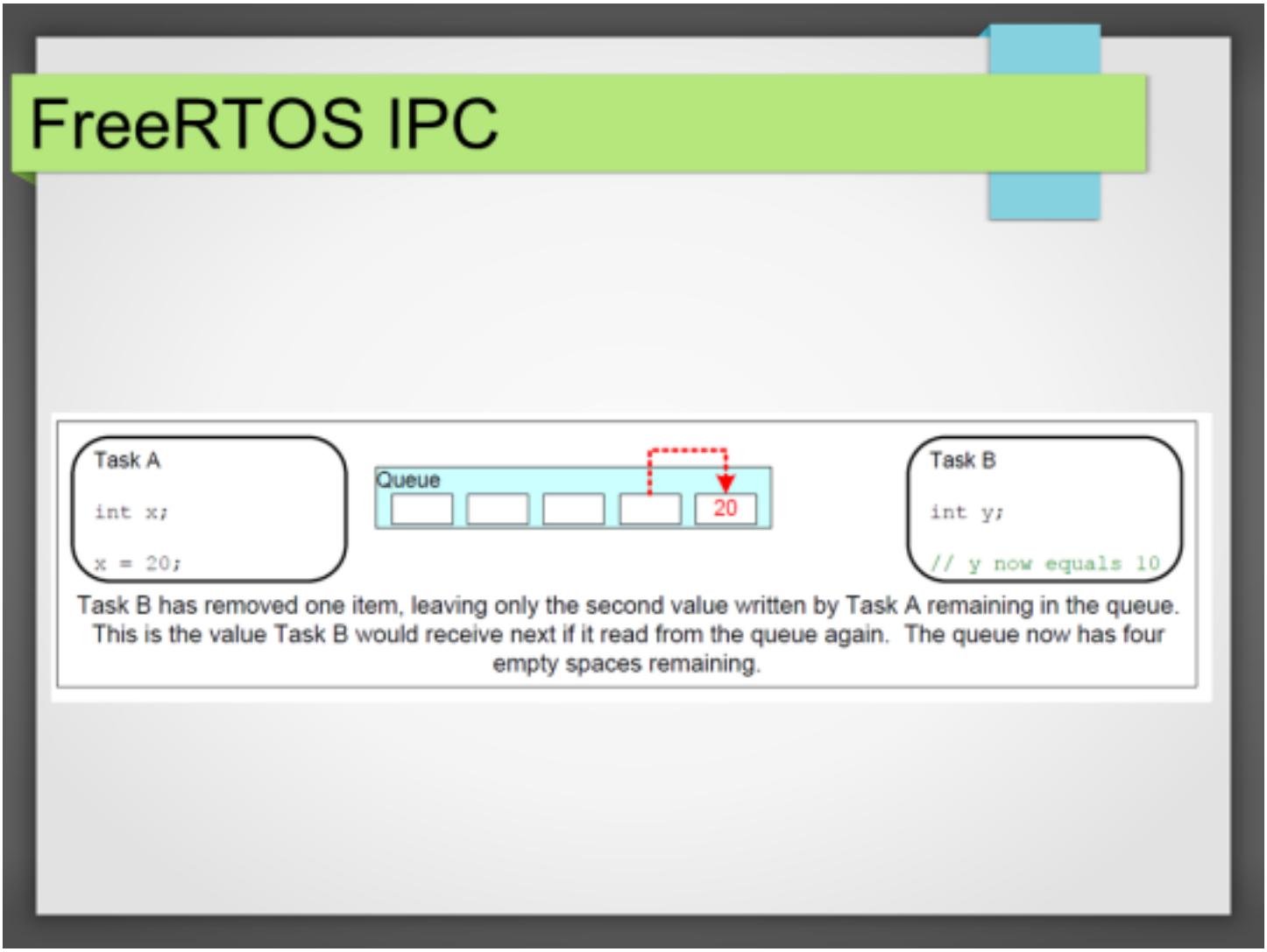
FreeRTOS IPC



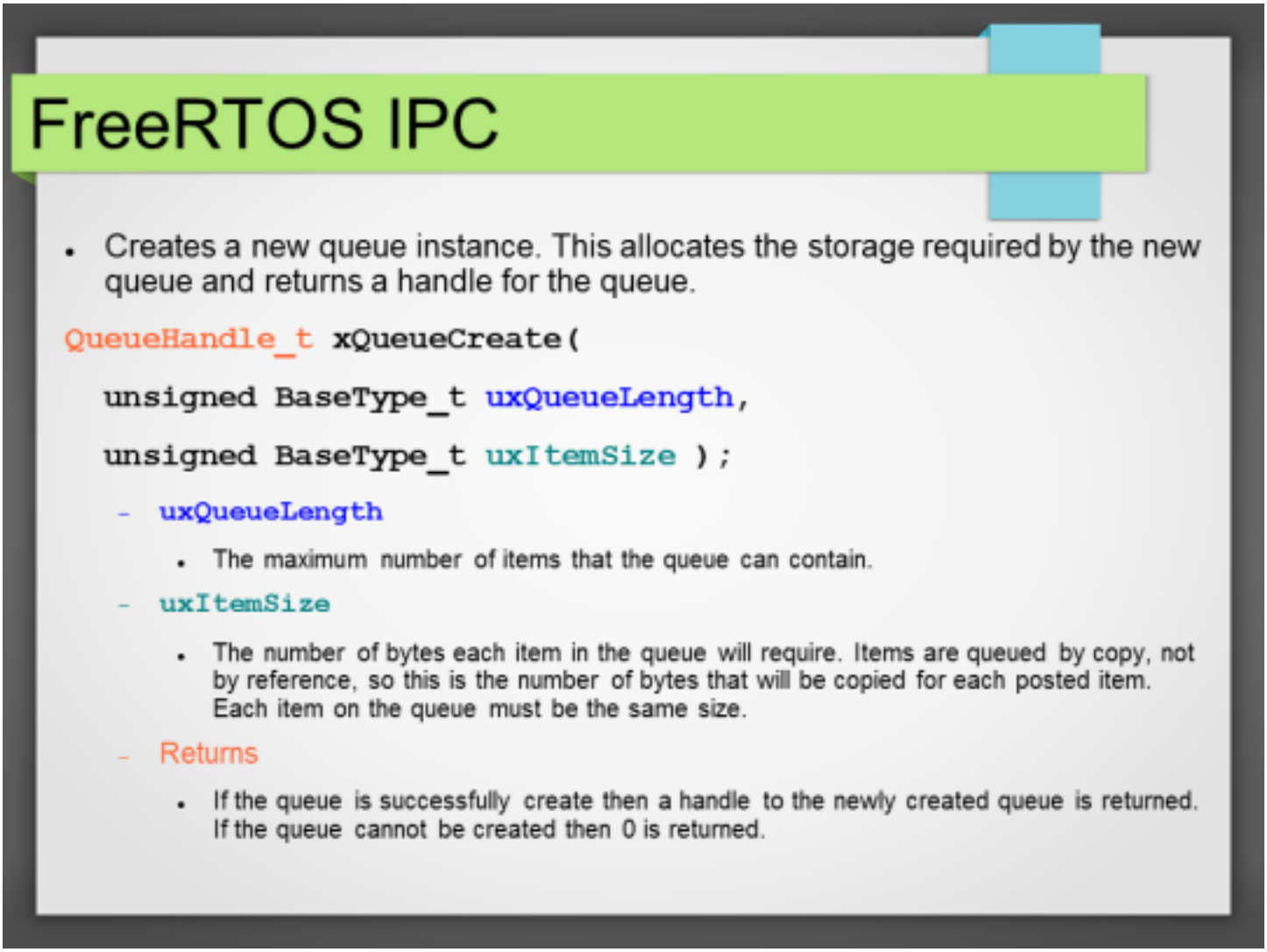
FreeRTOS IPC



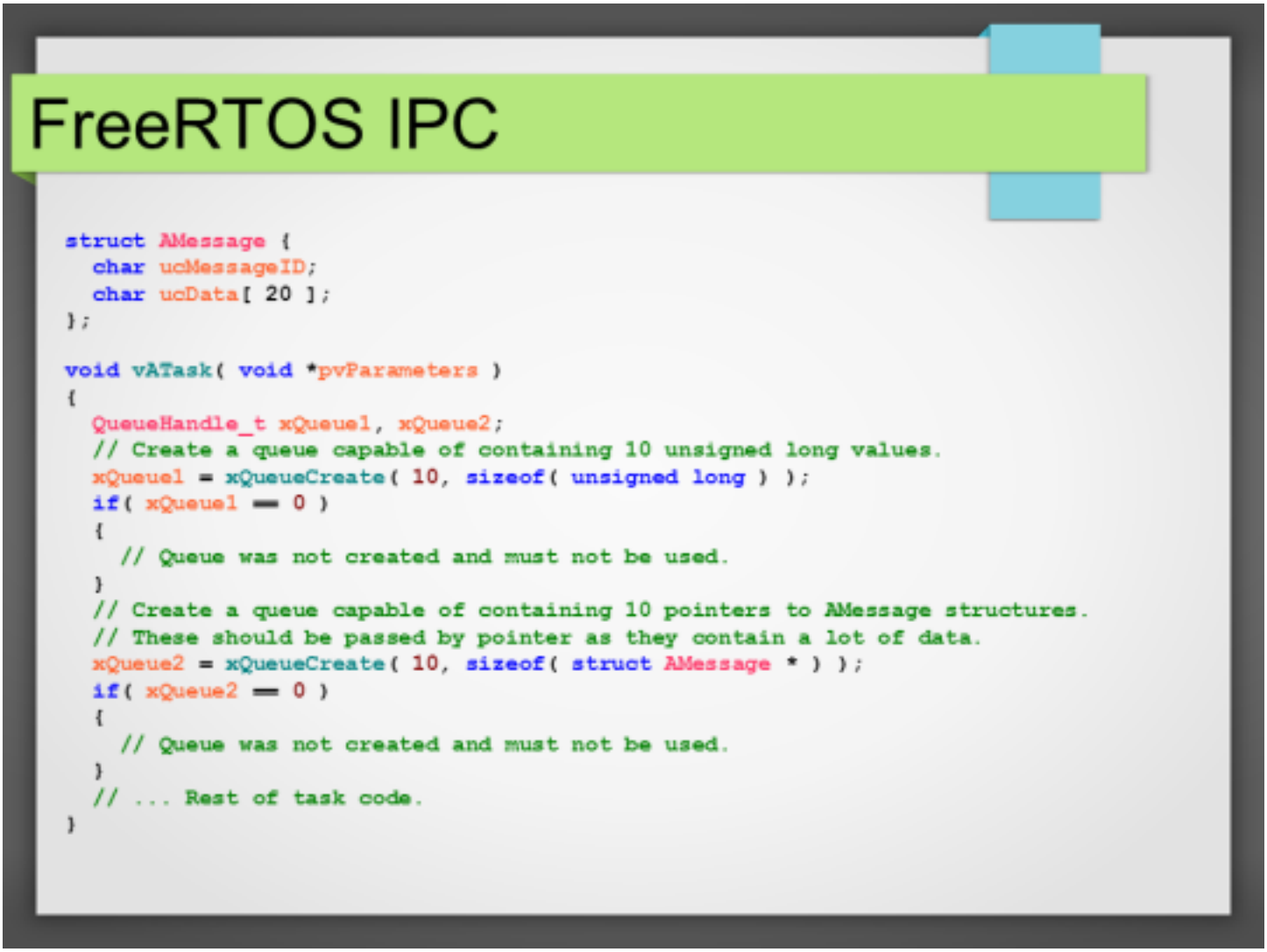
FreeRTOS IPC



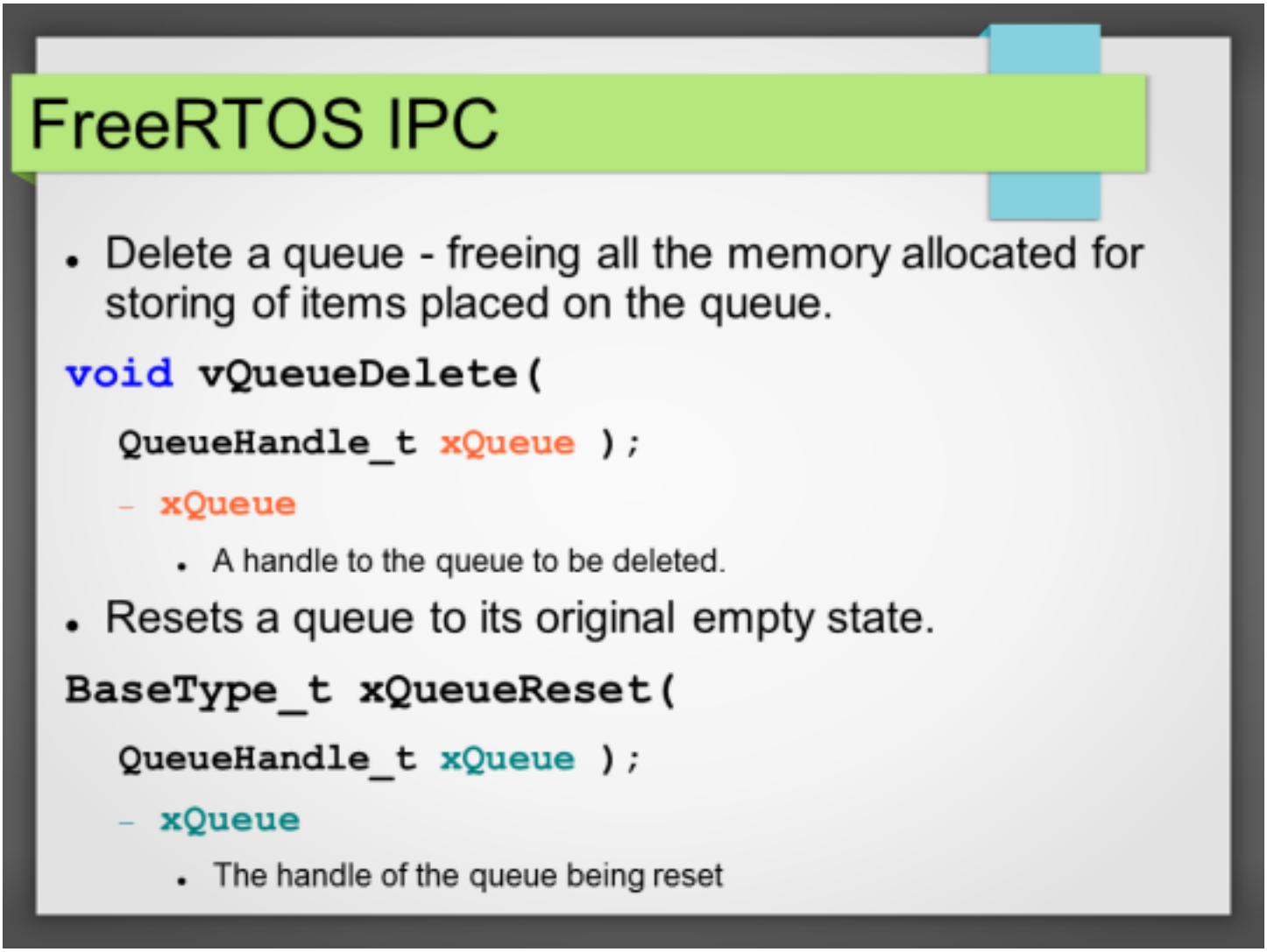
FreeRTOS IPC



FreeRTOS IPC



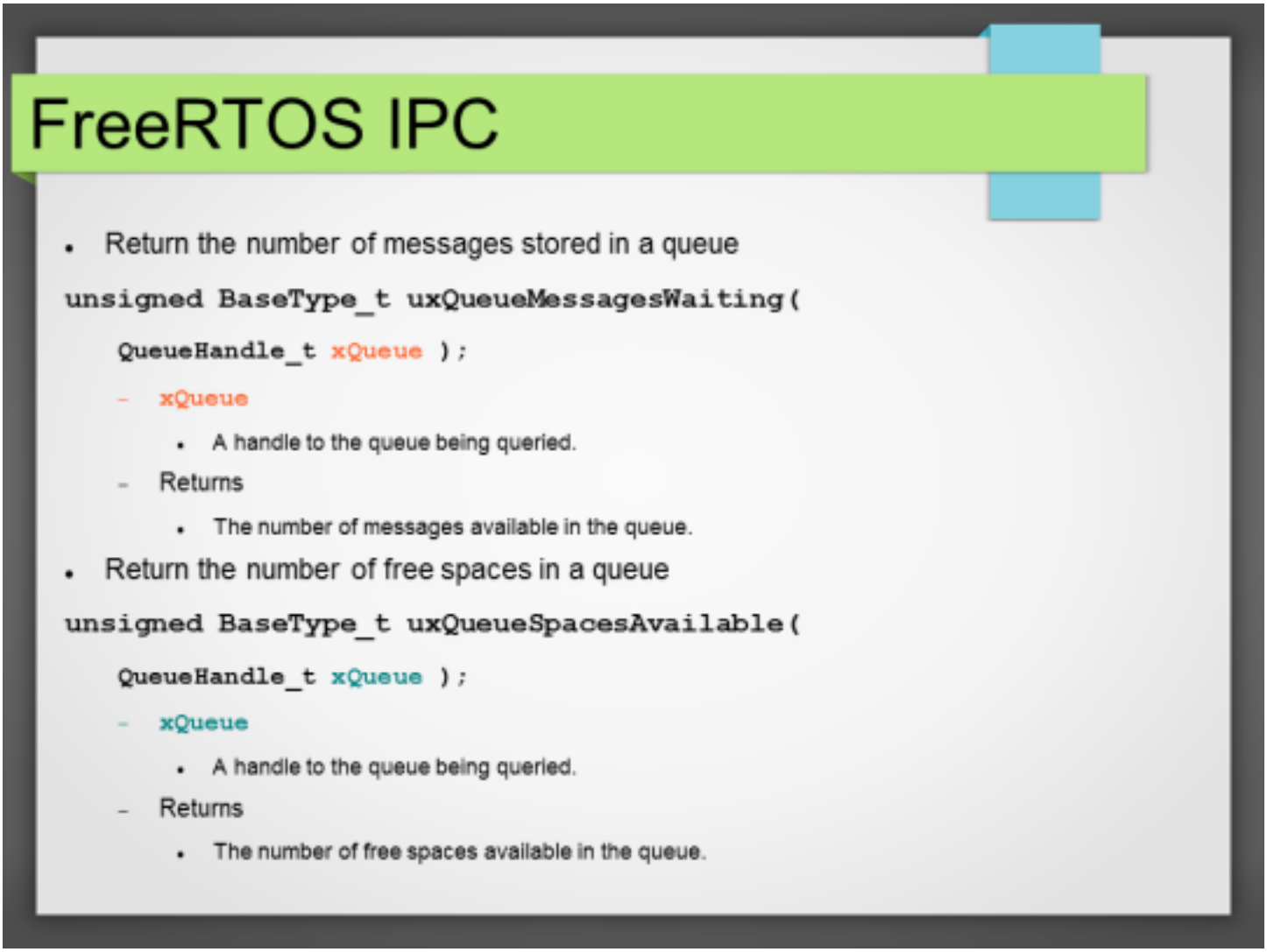
FreeRTOS IPC



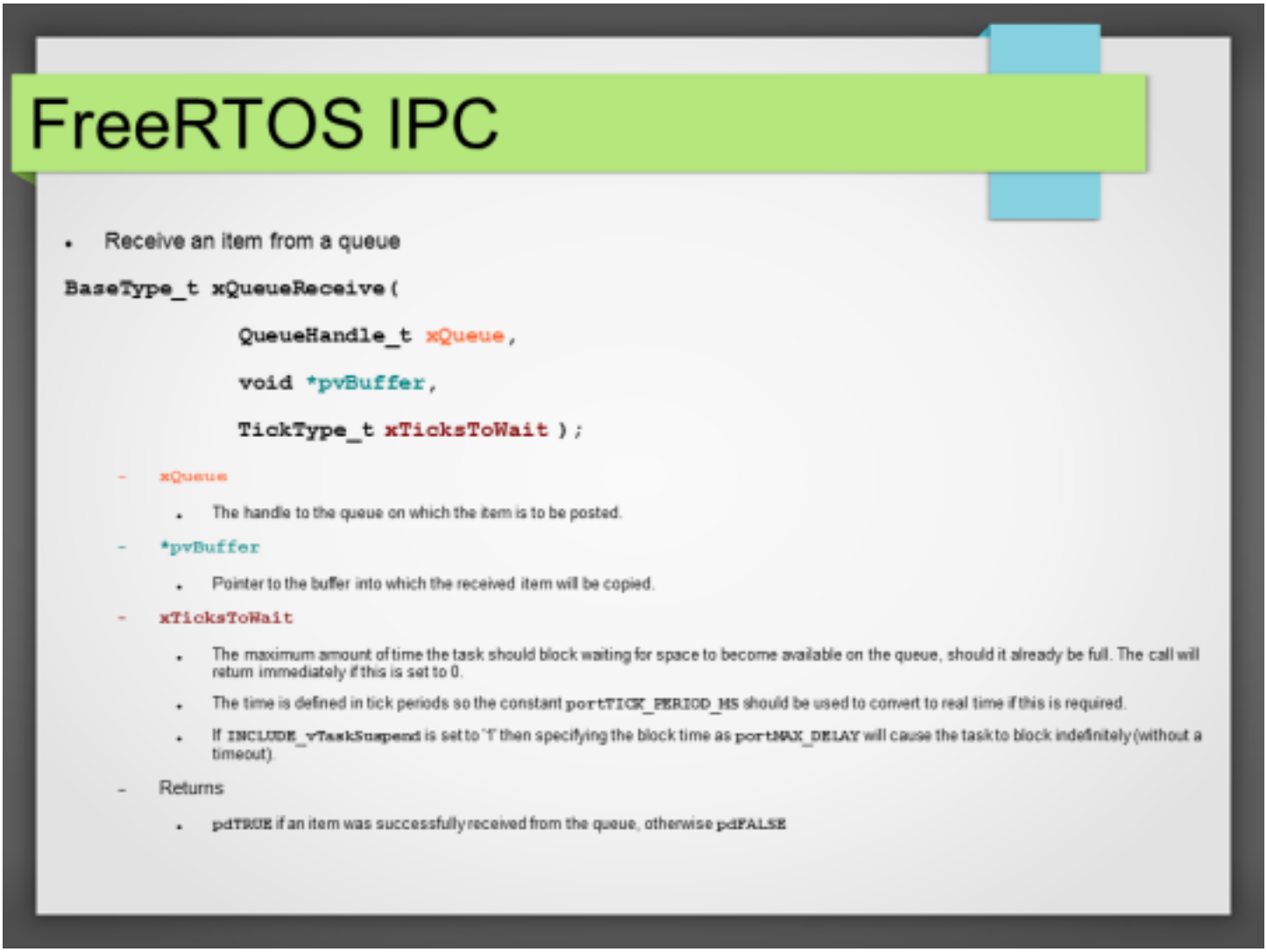
FreeRTOS IPC

FreeRTOS IPC

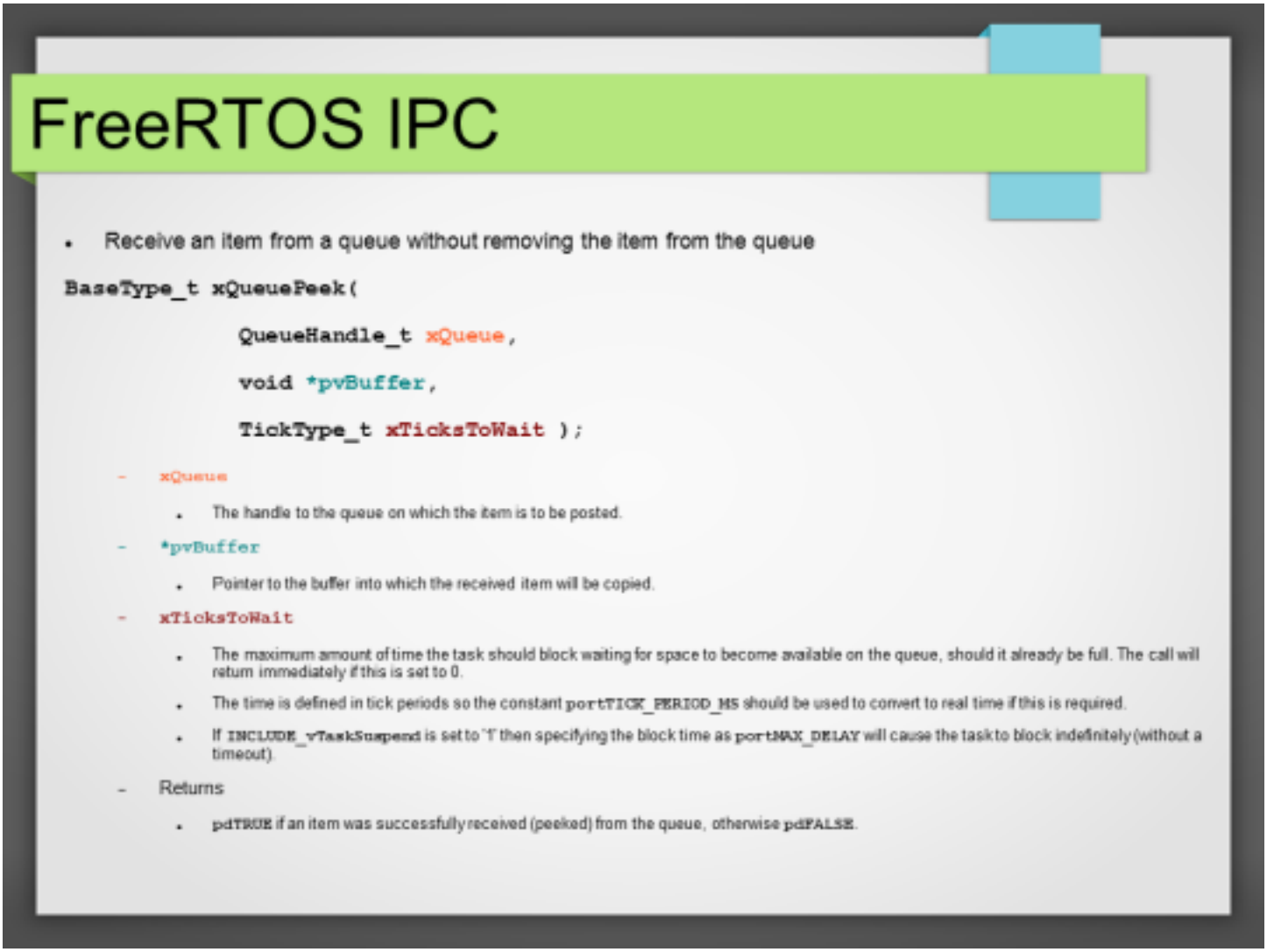
FreeRTOS IPC



FreeRTOS IPC



FreeRTOS IPC



FreeRTOS IPC

FreeRTOS IPC

- A version of **`xQueueSendToBack()`** that will write to the queue even if the queue is full, overwriting data that is already held in the queue.
- **`xQueueOverwrite()`** is intended for use with queues that have a length of one, meaning the queue is either empty or full.

```
BaseType_t xQueueOverwrite(
                QueueHandle_t xQueue,
                const void * pvItemToQueue );
```

- `xQueue`
  - The handle of the queue to which the data is to be sent.
- `pvItemToQueue`
  - A pointer to the item that is to be placed in the queue. The size of the items the queue will hold was defined when the queue was created, and that many bytes will be copied from pvItemToQueue into the queue storage area.
- Returns
  - **`pdPASS`** is the only value that can be returned because **`xQueueOverwrite()`** will write to the queue even when the queue was already full.