Message Queues

Message Queues

- A message queue is a buffer-like object through which tasks and ISRs send and receive messages to communicate and synchronize with data
- It temporarily holds message from a sender until the intended receiver is ready to read them.

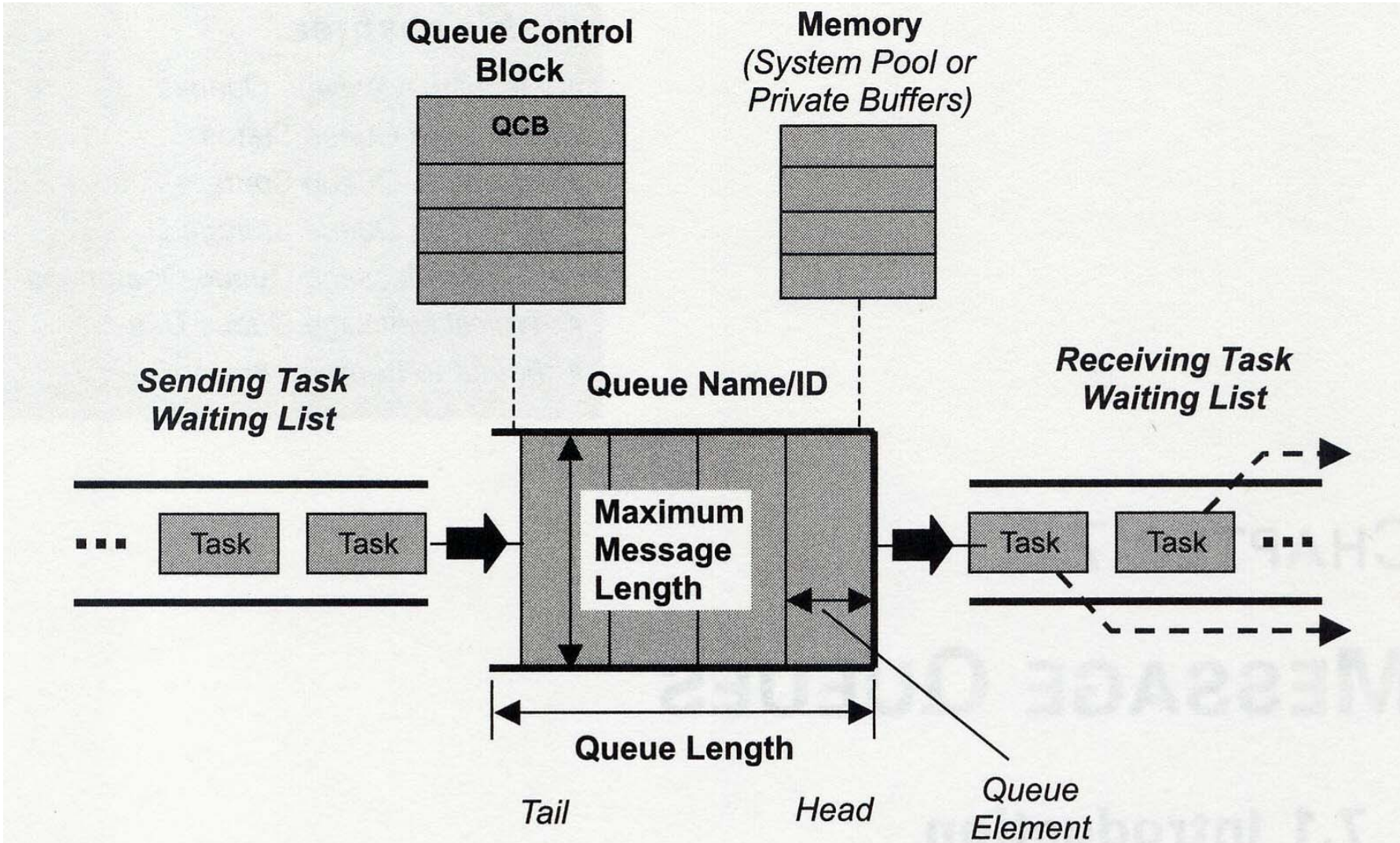


Figure 7.1 A message queue, its associated parameters, and supporting data structures.

A Message Creation

- QCB
- Queue name (developer)
- Unique ID
- Memory buffers
- Queue length (developer)
- Max message length (developer)
- Task-waiting-list

Message Queue States

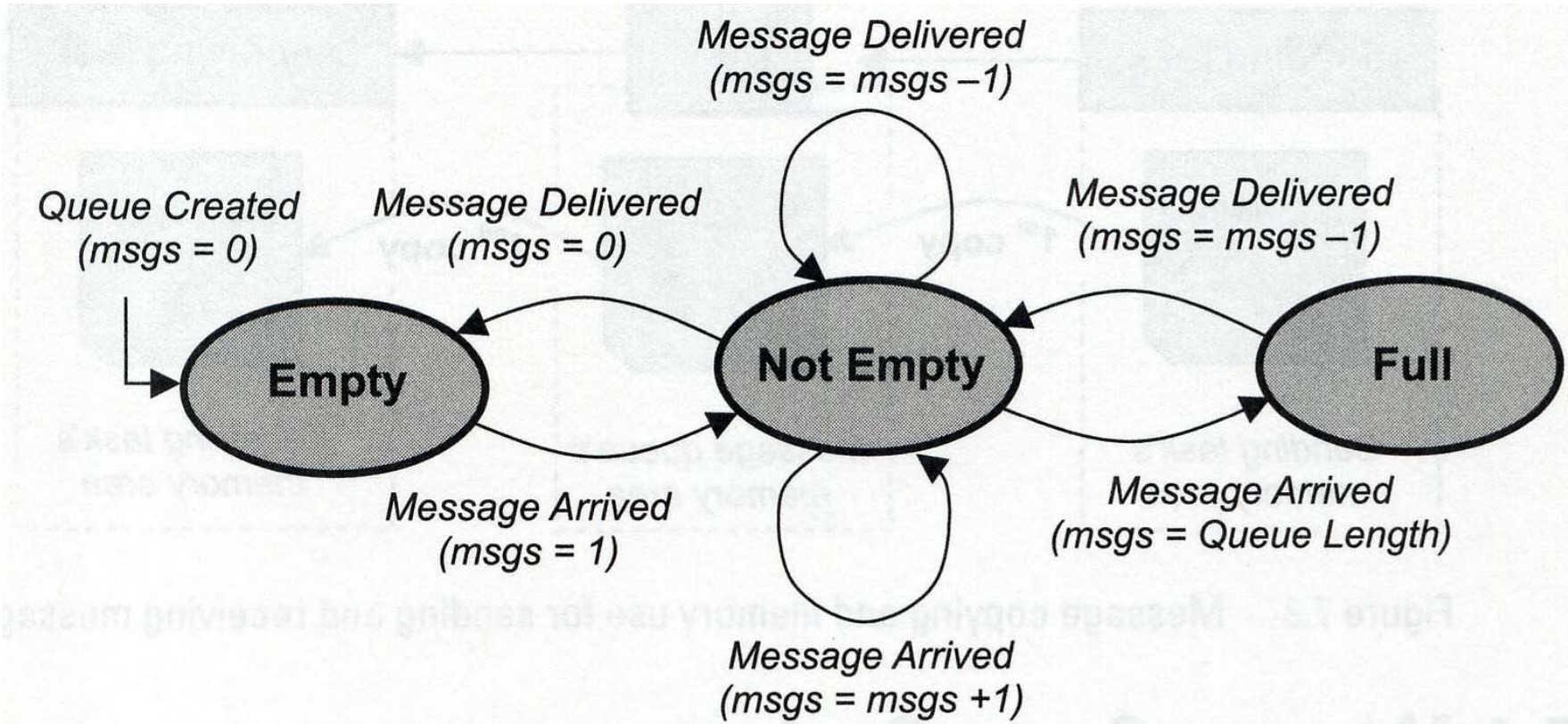
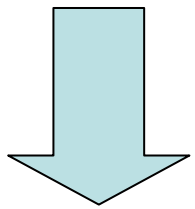


Figure 7.2 The state diagram for a message queue.

- When a task attempts to send a message to a full message queue



- Sending functions returns an error code

or

- Block -> sending task-waiting list

Message Queue Content

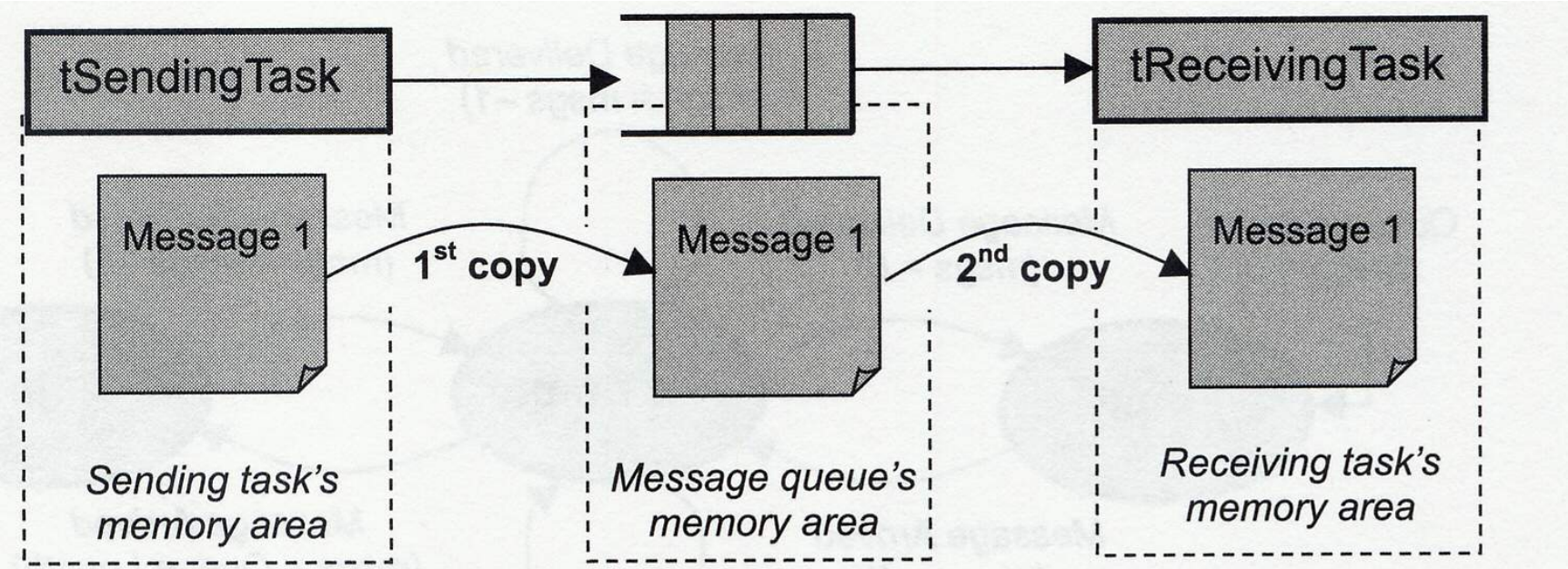


Figure 7.3 Message copying and memory use for sending and receiving messages.

- For long message, use pointer

Typical Message Queue Operations

- Create
- Delete

- Send
- Receive

Sending Messages: Filling queue

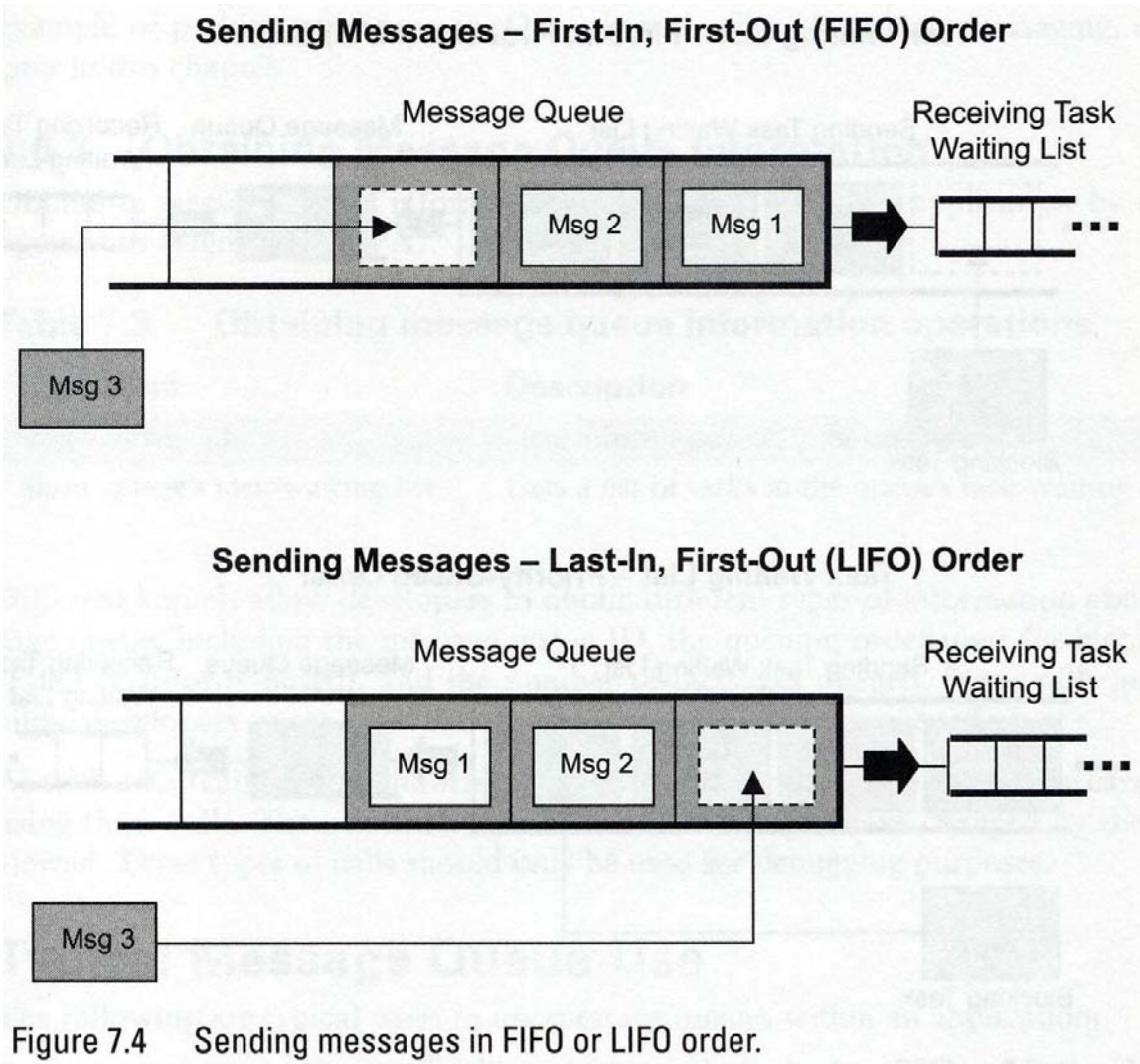


Figure 7.4 Sending messages in FIFO or LIFO order.

Sending Messages: When full queue

- Not blocked (ISRs and tasks)
- Block with a timeout (tasks only)
- Block forever (tasks only)

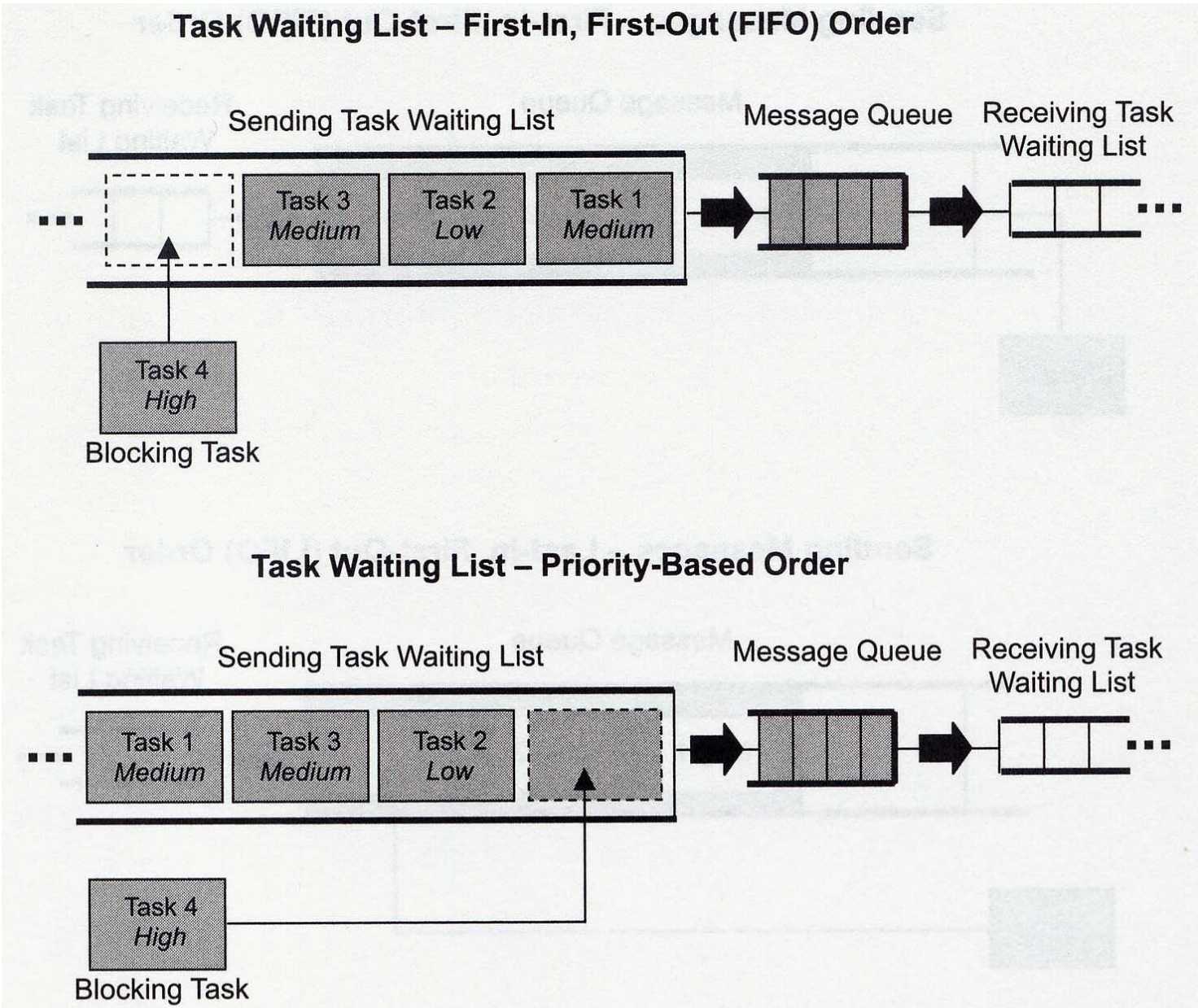


Figure 7.5 FIFO and priority-based task-waiting lists.

Receiving Messages: When queue empty

- Start to fill receiving task waiting list

- Not blocking
- Blocking with a timeout
- Blocking forever

Typical Message Queue Use

- Non-interlocked, one-way data communication (loosely coupled)
- Not synchronized
- Does not require ACK

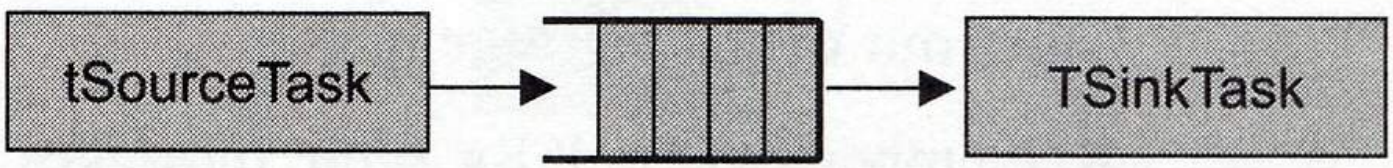


Figure 7.6 Non-interlocked, one-way data communication.

If tSinkTask higher priority

- tSinkTask runs first and blocks on empty queue
- As soon as tSourceTask sends the message to the queue, tSinkTask receives the message and unblocks

If tSinkTask lower priority

- tSourceTask fills the message queue, queue filled up
- tSourceTask blocks on full queue
- tSinkTask wake up and start receiving message

ISR Non-Interlocked and One-way Communication

- tSinkTask runs and blocks on empty queue
- Hardware triggers ISR and ISR put message in the queue
- After ISR complete running, tSinkTask has a chance to run and receive the message
- When ISR send a message to a full queue, the message is lost

Interlocked, One-way Data Communication

- For reliable communication or task synchronization

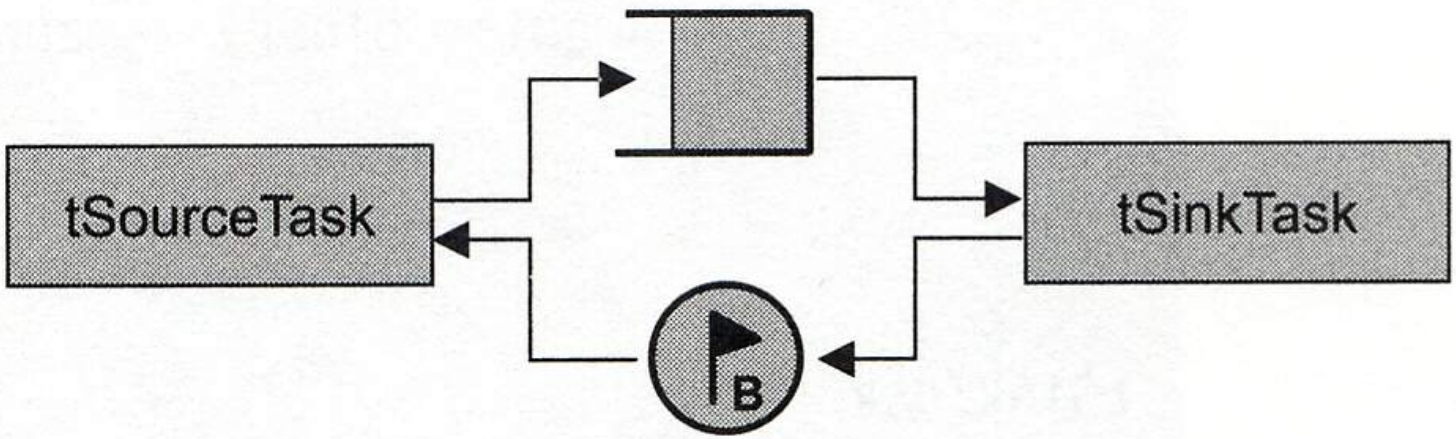


Figure 7.7 Interlocked, one-way data communication.

Interlocked, One-way Data Communication

- Initially, binary semaphore=0, message queue with a length 1 (mailbox)
- tSourceTask send a message and blocks on empty semaphore
- tSinkTask receives the message and give a semaphore
- tSourceTask unblocks and send another message and blocks again on empty semaphore

Interlocked, One-way Data Communication

```
tSourceTask()
{
   send message
   acquire binary semaphore
}
tSinkTask()
{
   receive message
   give binary semaphore
}
```

Interlocked, Two-way Data Communication

- Full-duplex, tightly coupled
- Useful for client/server-based system

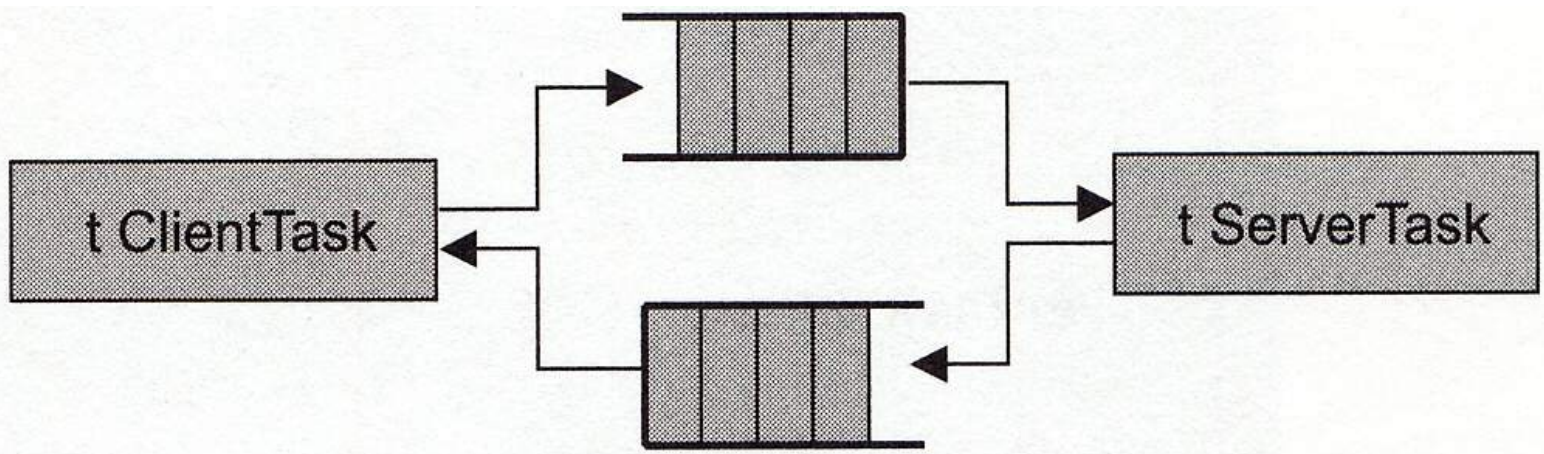


Figure 7.8 Interlocked, two-way data communication.

Interlocked, Two-way Data Communication

- tClientTask send a request to tServerTask
- tServerTask fulfills that request by sending a message back to tClientTask

Interlocked, Two-way Data Communication

```
tClientTask()
{
   send a message
   wait for message from server queue
}
tServerTask()
{
   receive a message from the request queue
   send a message to the client queue
}
```

Interlocked, Two-way Data Communication

- Two separate message queue
- tServerTask set to a higher priority