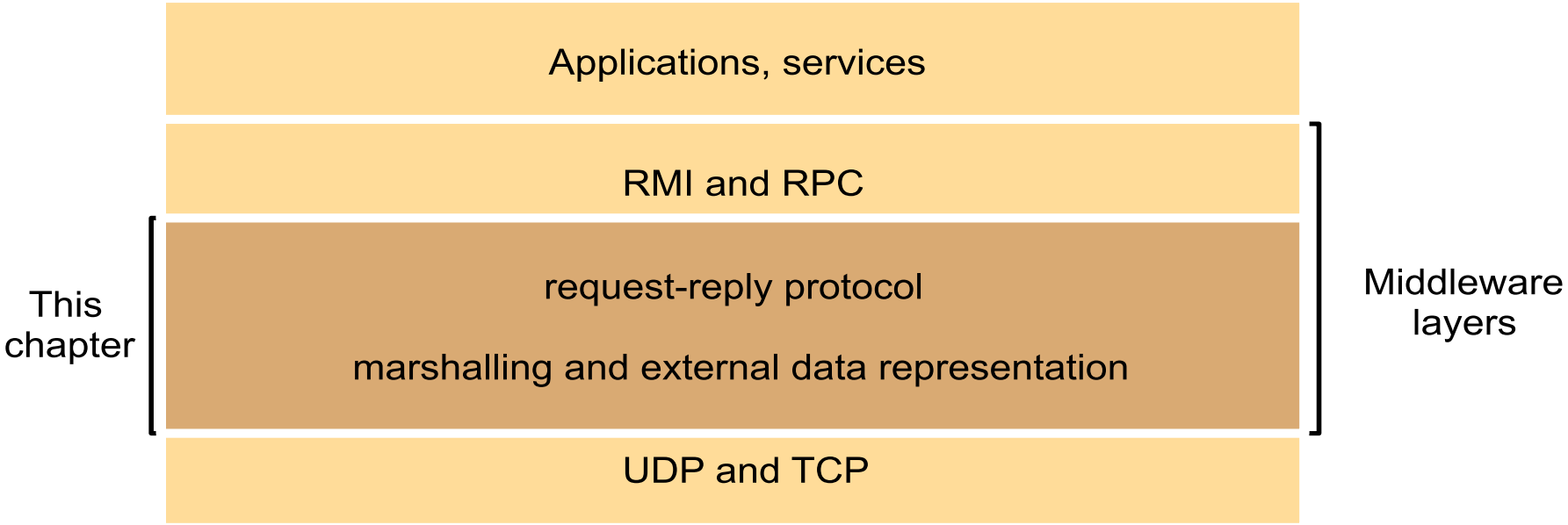
Interprocess communication

- How do two processes communicate with each other?
  - Not just machines, but programs.
- This is the fundamental building block of distributed systems.
- Other techniques, like remote method invocation, are implemented on top of this layer. They are just an abstraction above raw interprocess communication.

Interprocess communication (IPC)

- IPC (chapter 4) and RMI (chapter 5) are all about middleware. They are the abstraction layer above lower level protocols that actually move data between processes.



Interprocess communication

- IPC is all about *message passing*.
- Message passing involves two sides:
  - Sender: The source of the outgoing message.
  - Receiver: The destination of the outgoing message.

- Typically processes on the endpoints use *send* and *receive* operations in their API to perform this action of sending or receiving.
  - C sockets
  - Java sockets

Characteristics of IPC

- Synchronous vs. asynchronous messages
- Message destinations
  - Addressing, ports.
- Reliability
- Ordering

Synchronous vs. asynchronous comms.

- Synchronous messaging: Both sender and receiver synchronize for every message.
  - This is achieved by blocking on the paired send/recv until the communication is complete.
- Asynchronous messaging: One or both sides does not synchronize with the other.
  - Send or receive does not block, and immediately returns before the communication completes.
- Sometimes the difference is called blocking vs. non-blocking communications.

Asynchronous communications

- Clearly some issues arise that aren't present in the synchronous case.
- Consider first how a synchronous receive works.
- The receiver makes a call to `recv()` and provides a buffer into which it wants the message to be placed.
  - E.g.: `recv(src, &buf, BUFSIZE, …);`
- When the `recv()` completes and the receiver unblocks, the buffer will be guaranteed to have the message in it (or some error code will have occurred to indicate a problem).
- What about asynchronous?

Asynchronous communications

- In the asynchronous receive, the receive is posted and the buffer provided, but the call returns immediately.
  - No guarantee on what the buffer actually contains. It will get filled in by the communications subsystem when the message arrives at some point in the future.
- So, with asynchronous messaging, one needs to take care to check if communications have completed before using the contents of the buffers.
- Similar issue arises on sending side. Need to make sure buffer has been transmitted and safe to overwrite if using asynchronous sends.

Asynchronous communications

- The book talks about today's systems not providing these forms of receive. This isn't universally true.
- The point of asynchronous messaging is to overlap computation with communication.
  - Instead of blocking, do useful work while the messages transmit.
- This is tedious to write. BUT, some systems do provide it.
- Most common in high performance computing, where every cycle is precious.
  - Example: MPI immediate send/receive calls