Lecture 35

- Administration

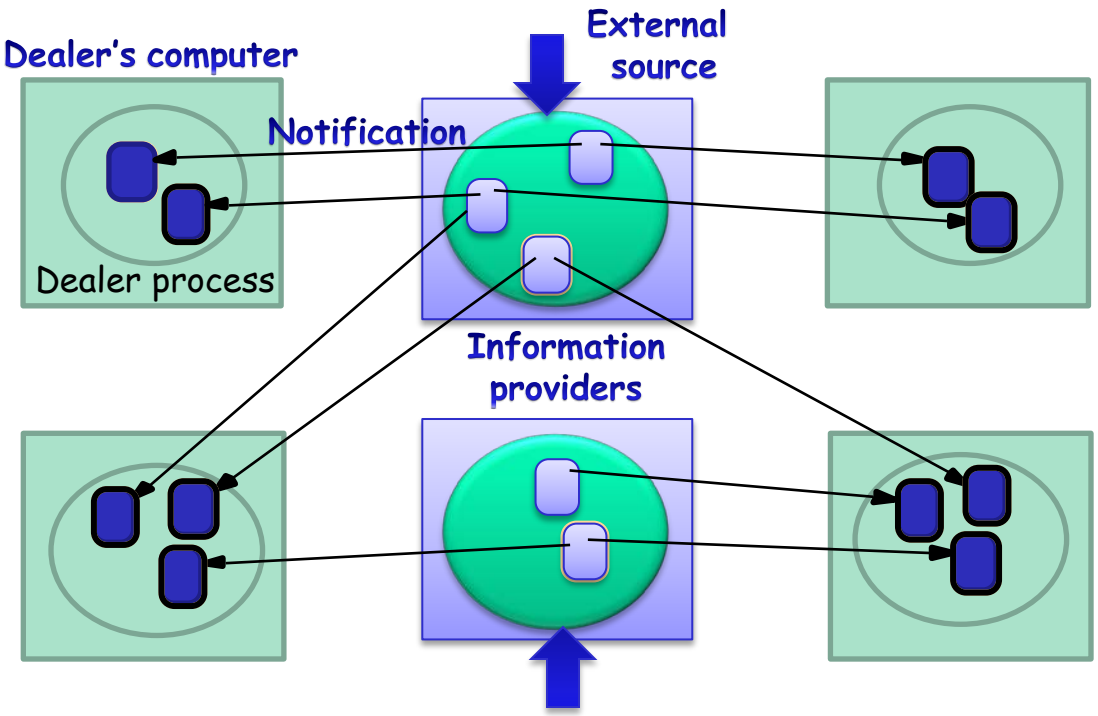
Publish-Subscribe

- An event-based communication mechanism
  - Publishers publish events to an event service
  - Subscribers express interest in particular events

- Large number of producers distribute information to large number of consumers

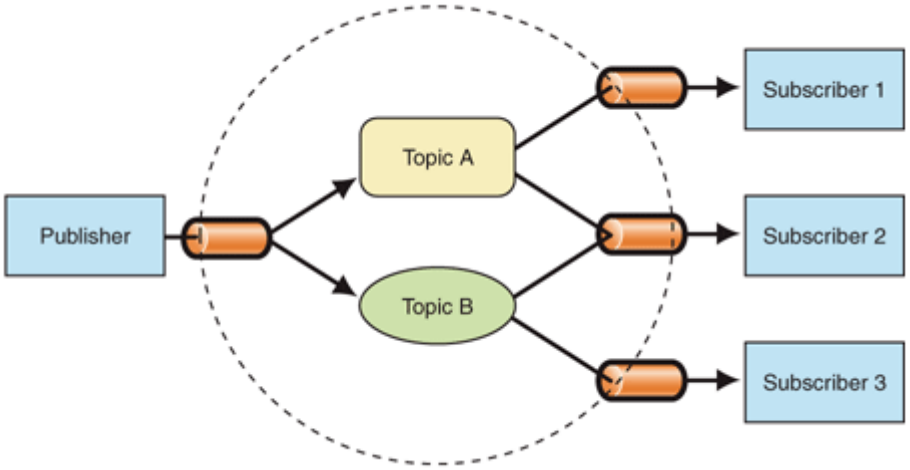
2. Publish-Subscribe (cont'd)

Example: Financial trading



Publisher/subscribers

- Searching the web for this method, one sees that it is mainly described in the context of general, distributed systems, for instance in servicing customers.
- However, it is also very interesting for distributed data acquisition / processing configuration

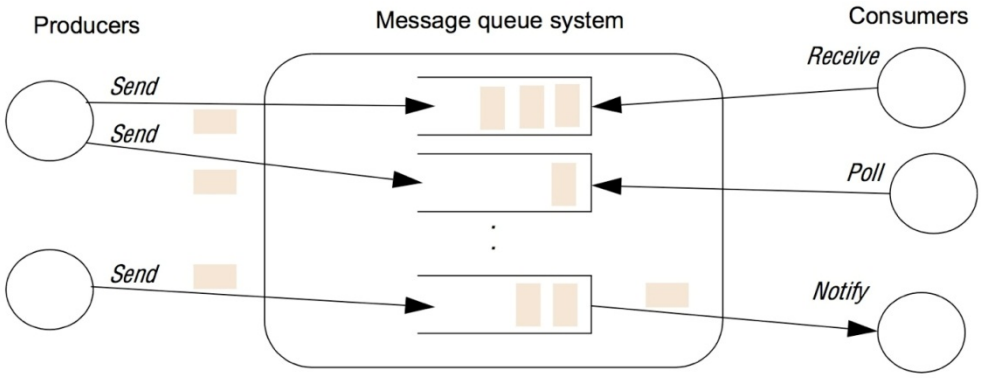


"Message queues"

- The generic name Message Queues is a software-engineering component used for interprocess communication or inter-thread communication within the same process. It uses a queue for messaging – the passing of control or of content
- Message queues provide an asynchronous communications protocol, meaning that the sender and receiver of the message do not need to interact with the message queue at the same time. Messages placed onto the queue are stored until the receiver retrieves them.

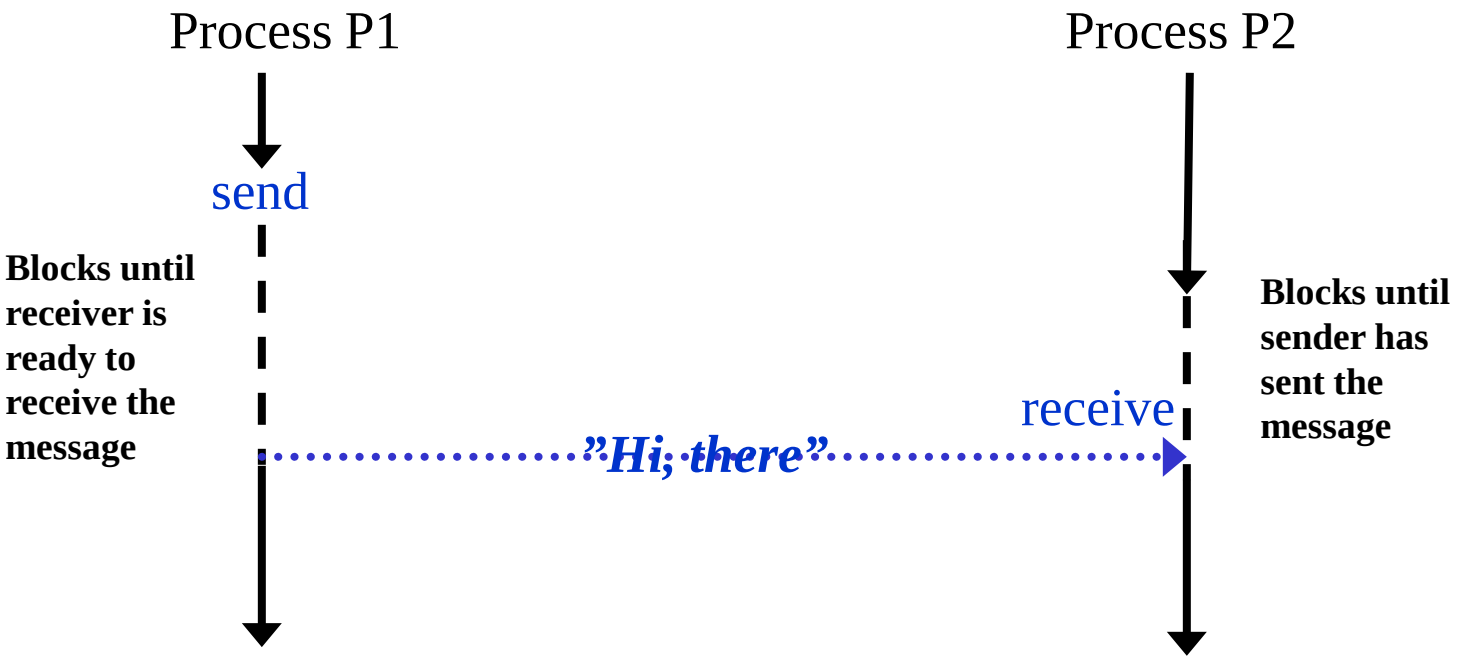
Message Queues

- A refinement of Publish-Subscribe where
  - Producers deposit the messages in a queue
  - Messages are delivered to consumers through different methods
  - Queue takes care of ensuring message delivery
- Advantages
  - Enables space decoupling
  - Enables time decoupling



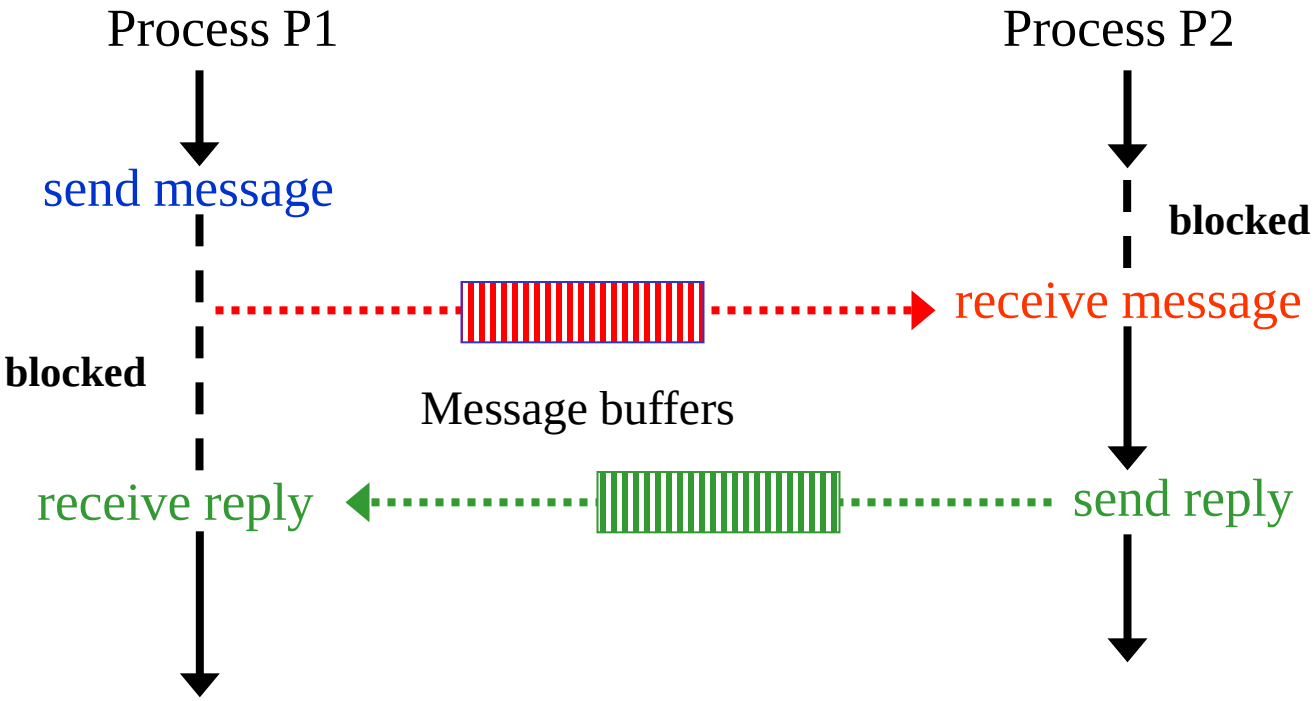
Synchronous message handling

- No buffering, space for only one message, blocking
  - A send-receive pair is both data transfer and synchronization
  - But how does the receiver process know that there is a message ?



Buffered communication

- Buffer space required
  - Buffer write/read operations
  - Blocking occurs when buffer is full for write or empty for read
    - However, POSIX Asynchronous I/O is non-blocking!



Asynchronous communication

- For example, an application may need to notify another that an event has occurred, but does not need to wait for a response.
- Another example occurs in publish/subscribe distributed systems, where an application "publishes" information for any number of clients to read
- In both these examples it would not make sense for the sender of the information to have to wait if, for example, one of the recipients had crashed.
- However, the communication may include a reply message, in this case the initial sender may want to read the answer when it suits her.

Some problems with asynchronous communication

- Potentially infinite buffers are needed to store unread messages
- The programs get more complex, in particular when signals are used to notify the receiver
- It is probably more difficult to prove the correctness of a system
- What to do with messages which seems to be never read?
  - With the IEEE 1596 Scalable Coherent Interface (SCI) one can specify "time-of-death" for a message. The basic topology of SCI is a ring. The «scrubber» node will remove a message that has reached its «time-of-death» such that it will not circle on the ring forever.

Process naming

- Two distinct sub-issues
  - direction versus indirection
  - symmetry
- With direct naming, the sender explicitly names the receiver:

  send <message> to <process-name>

- With indirect naming, the sender names an intermediate entity (e.g. a channel, mailbox, link or pipe):

  send <message> to <message queue>/channel/mailbox

(ref. B&W)

Process naming (contd)

- A naming scheme is symmetric if both sender and receiver name each other (directly or indirectly)
  - send <message> to <process-name>
  - wait <message> from <process-name>

  - send <message> to <mailbox>
  - wait <message> from <mailbox>
- It is asymmetric if the receiver names no specific source but accepts messages from any process
  - wait <message>
- Asymmetric naming fits the client-server paradigm
- With indirect the intermediary could have:
  - a many-to-one structure
  - a many-to-many structure
  - a one-to-one structure
  - a one-to-many **(as in publisher/subscribers)**

Review

- Theory (5)
- Concurrent (5)
- Time (10)
- Coordination-Agreement (15)
- Group Communication (15)
- Case Studies (10)

Review

- Theory
  - Safety, Liveness issues
  - LTS and FSP
  - Modeling concurrency
- Concurrent
  - Concurrent objects and data structures
  - QC, SC, Linearizability

Review

- Time
  - Message systems (MPI)
  - Time/space diagrams
  - Async, FIFO, CO, Sync
  - Clocks, synchronizing clocks
- Coordination-Agreement
  - Leader Election (Chang/Roberts)
  - Chandy/Lamport Snapshot
  - Distributed Debugging
  - Synchronous communication, rendezvous

Review

- Group Communication
  - Reliable multicast
  - Ordered multicast (ISIS)
  - consensus
- Case Studies
  - Pastry (P2P)
  - Map-Reduce, Google
  - MOM