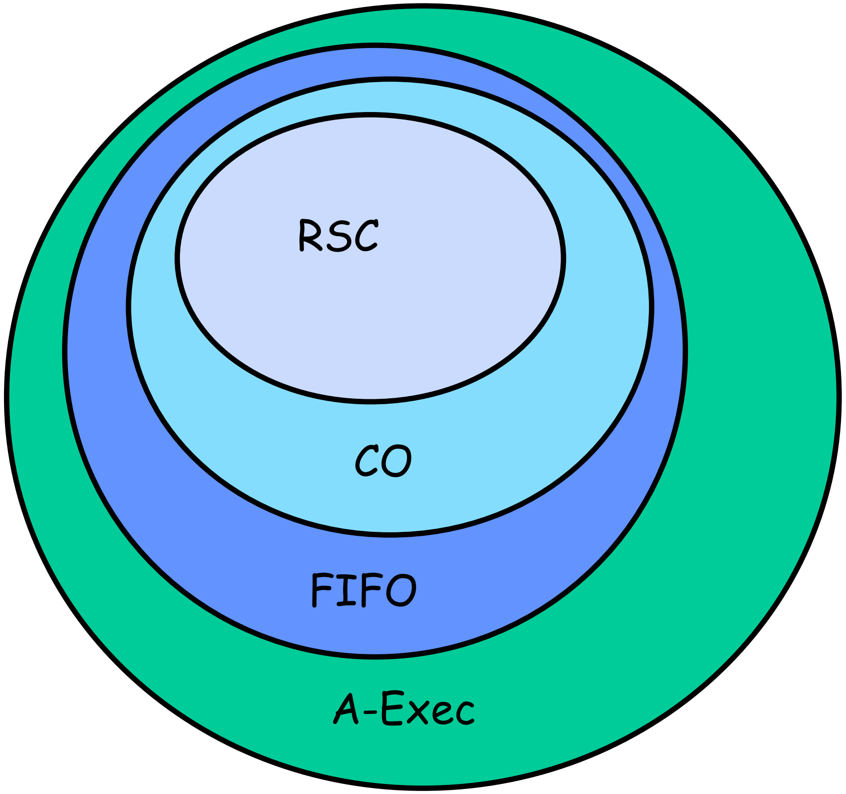
Lecture 27

- Administration
- Chapter 6 (section 1,2,3)

Distributed Computing Principles, Algorithms, and Systems
Ajay D. Kshemkalyani and Mukesh Singhal

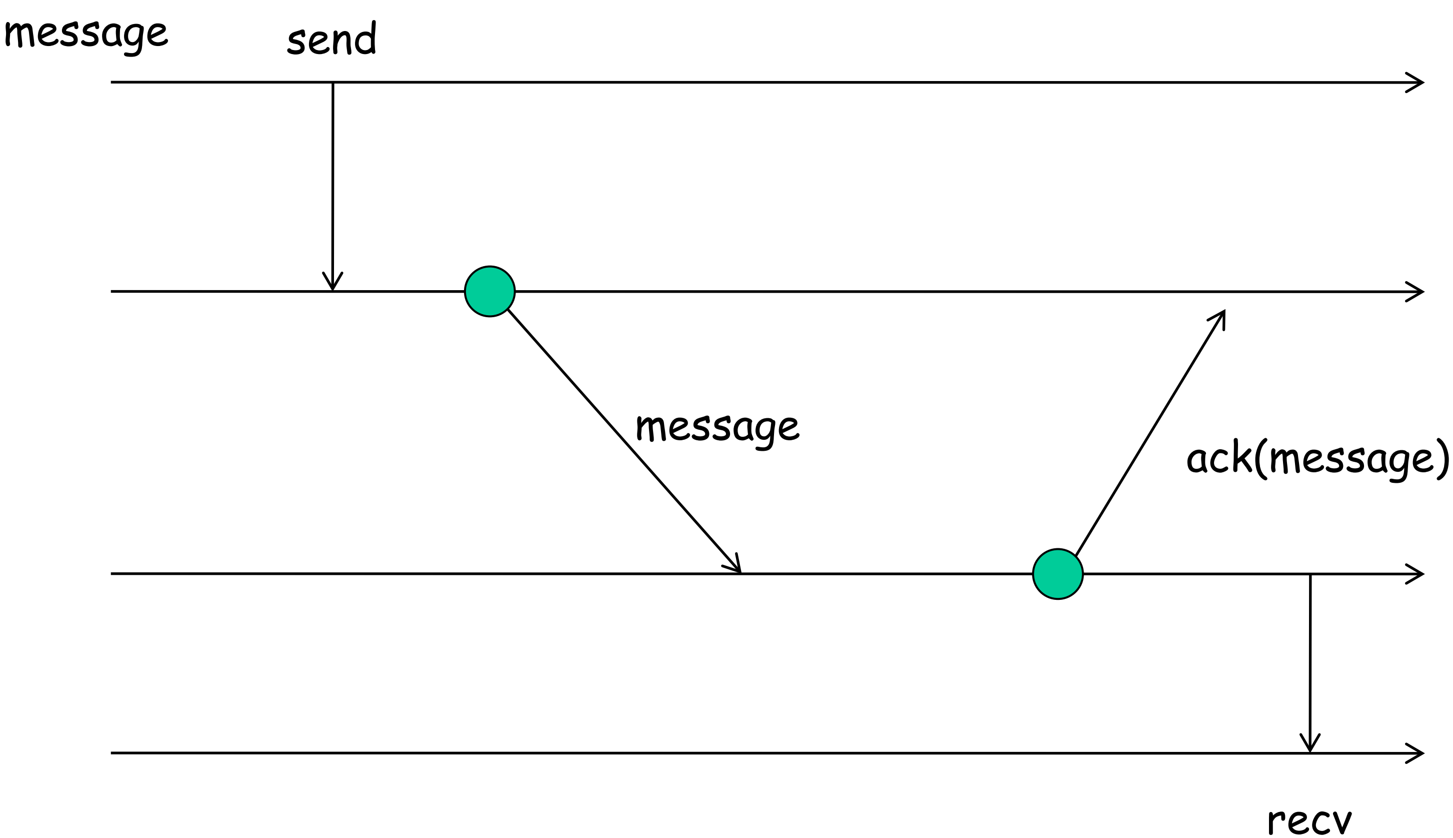
Ordered Communication Hierarchy



Bagrodia's Algorithm

- Assumptions:
  - RECV commands are always enabled (within scope of a construct)
  - SEND once enabled remain enabled before the send is executed
  - Process IDs can be used to break symmetry
  - Only one send command per process enabled.
  - It is possible to receive small protocol messages

Binary Rendezvous



Basic Idea

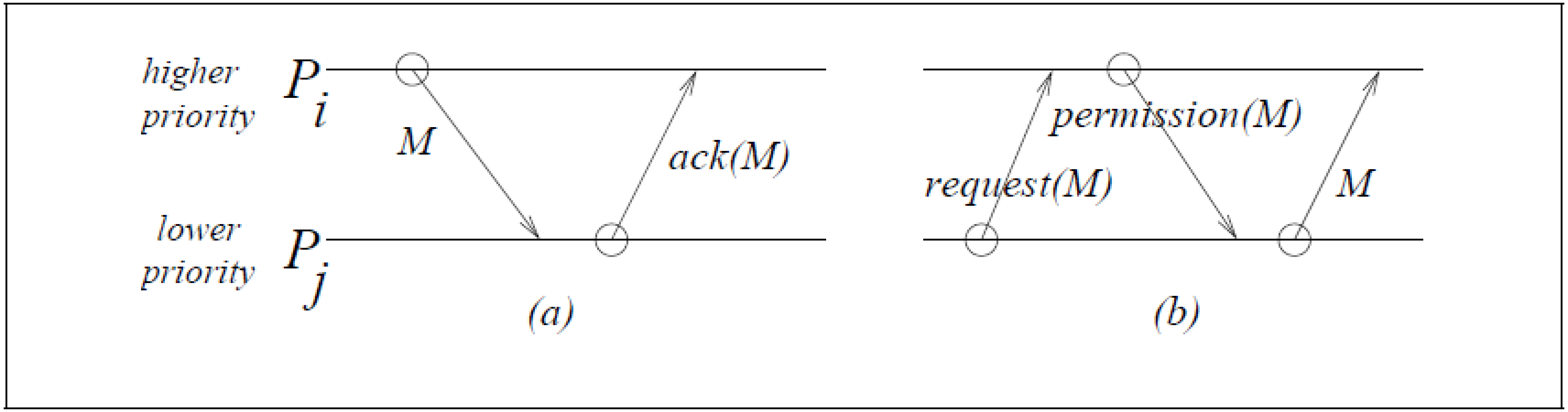


Figure 6.9: Messages used to implement synchronous order. P_i has higher priority than P_j. (a) P_i issues SEND(M). (b) P_j issues SEND(M).

(message types)
M, *ack(M)*, *request(M)*, *permission(M)*

1. **P_i wants to execute SEND(M) to a lower priority process P_j:**
   P_i executes *send(M)* and blocks until it receives *ack(M)* from P_j. The send event SEND(M) now completes.

   Any M' message (from a higher priority processes) and *request(M')* request for synchronization (from a lower priority processes) received during the blocking period are queued.

2. **P_i wants to execute SEND(M) to a higher priority process P_j:**

   (a) P_i seeks permission from P_j by executing *send(request(M))*.
   // to avoid deadlock in which cyclically blocked processes queue messages.

   (b) While P_i is waiting for permission, it remains unblocked.

      i. If a message M' arrives from a higher priority process P_k, P_i accepts M' by scheduling a RECEIVE(M') event and then executes *send(ack(M'))* to P_k.

      ii. If a *request(M')* arrives from a lower priority process P_k, P_i executes *send(permission(M'))* to P_k and blocks waiting for the message M'. When M' arrives, the RECEIVE(M') event is executed.

   (c) When the *permission(M)* arrives, P_i knows partner P_j is synchronized and P_i executes *send(M)*. The SEND(M) now completes.

3. ***Request(M)* arrival at P_i from a lower priority process P_j:**
   At the time a *request(M)* is processed by P_i, process P_i executes *send(permission(M))* to P_j and blocks waiting for the message M. When M arrives, the RECEIVE(M) event is executed and the process unblocks.

4. **Message M arrival at P_i from a higher priority process P_j:**
   At the time a message M is processed by P_i, process P_i executes RECEIVE(M) (which is assumed to be always enabled) and then *send(ack(M))* to P_j.

5. **Processing when P_i is unblocked:**
   When P_i is unblocked, it dequeues the next (if any) message from the queue and processes it as a message arrival (as per Rules 3 or 4).

Example

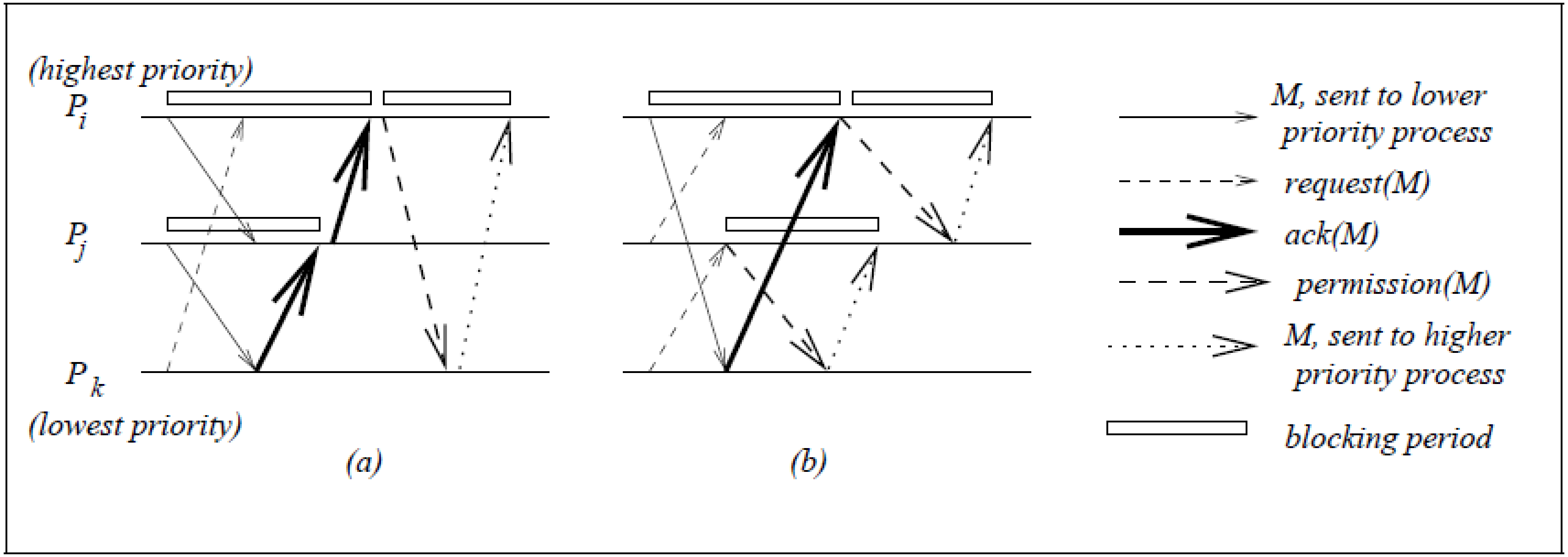


Figure 6.11: Examples showing how to schedule messages sent with synchronous primitives.