WEEK 10

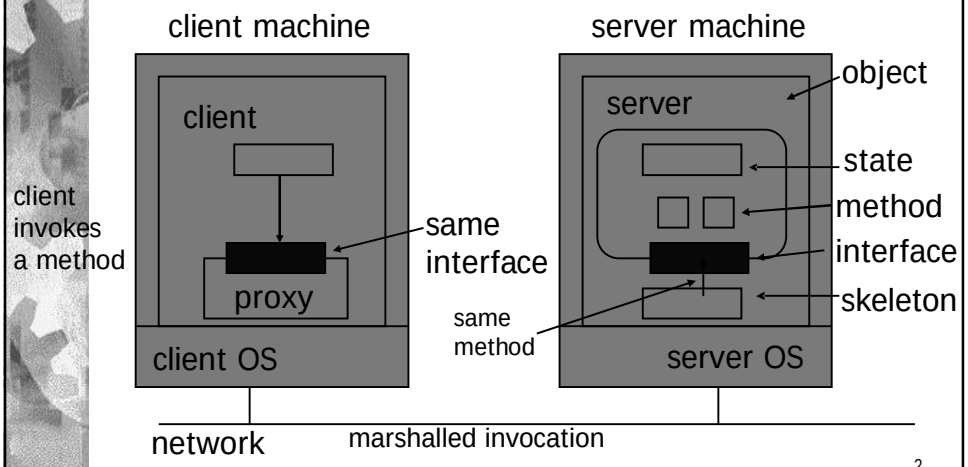
Distribution issues, global state, clusters, CORBA, etc

Stallings, Chapters 14 & 15 + Appendix B

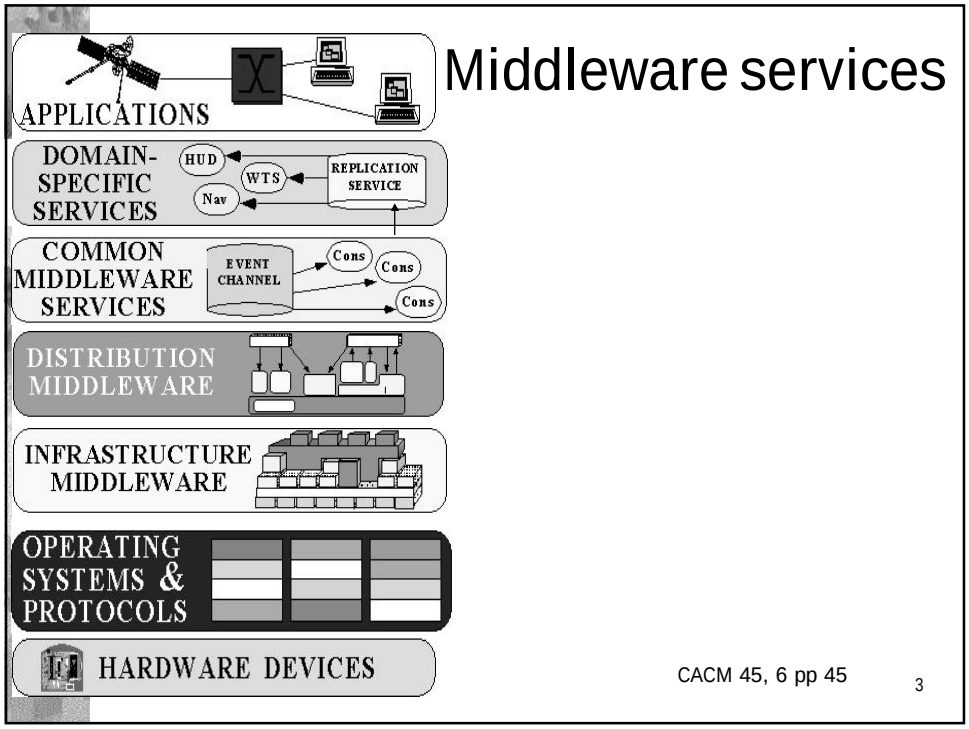
Prev. edition; Chapters 13&14

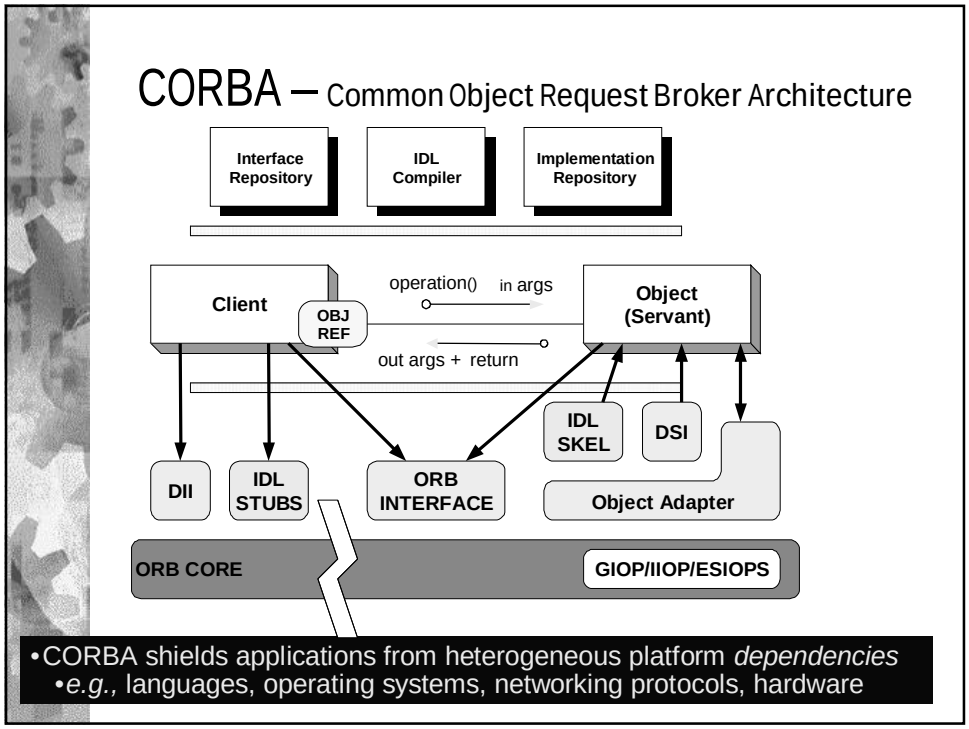
1

Distributed objects



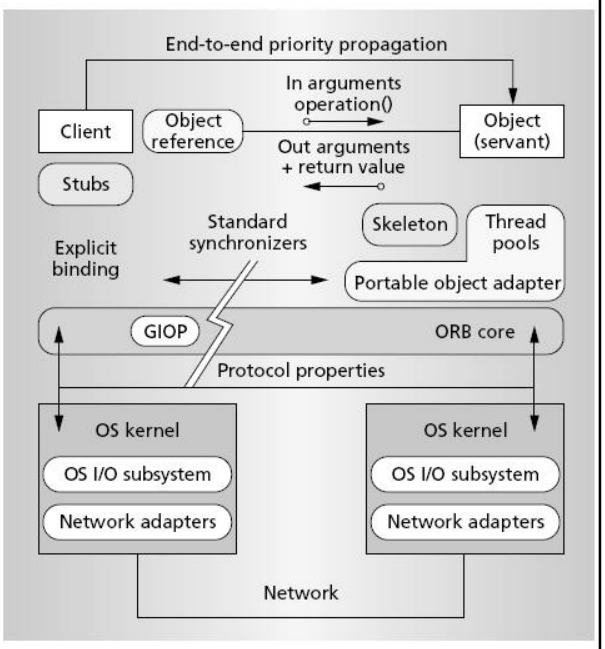
2


Middleware services
APPLICATIONS
DOMAIN-SPECIFIC SERVICES
HUD
WTS
Nav
REPLICATION SERVICE
COMMON MIDDLEWARE SERVICES
EVENT CHANNEL
Cons
Cons
Cons
DISTRIBUTION MIDDLEWARE
INFRASTRUCTURE MIDDLEWARE
OPERATING SYSTEMS & PROTOCOLS
HARDWARE DEVICES
CACM 45, 6 pp 45
3

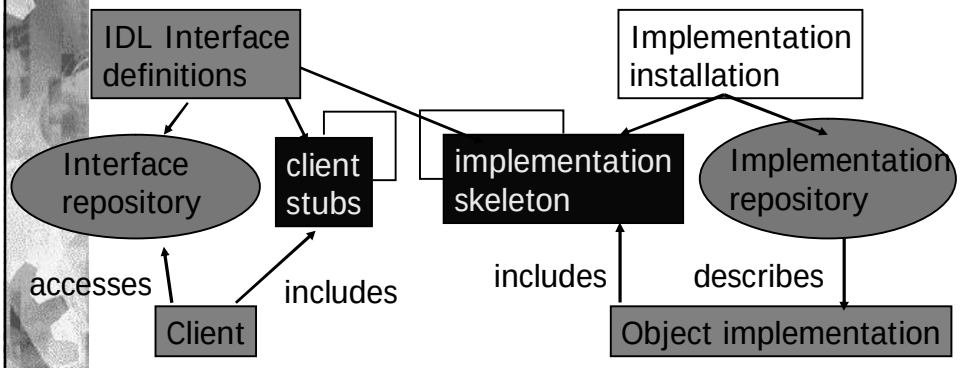

CORBA – Common Object Request Broker Architecture
Interface Repository
IDL Compiler
Implementation Repository
Client
OBJ REF
operation()
in args
out args + return
Object (Servant)
IDL SKEL
DSI
DII
IDL STUBS
ORB INTERFACE
Object Adapter
ORB CORE
GIOP/IIOP/ESIOPS
•CORBA shields applications from heterogeneous platform *dependencies*
•*e.g.*, languages, operating systems, networking protocols, hardware

RT-CORBA

- This figure shows the elements above and below ORB.
- RT features are not part of this course (they are explicit binding and priority propagation)



Making an application to CORBA



6

Interface definition: IDL

- IDL

```
module HelloApp{
        interface Hello {
                string sayHello();
        };
};
```

7

Client: 1/2

```
Import HelloApp.*;
//contains generated stubs
Import org.omg.Cosnaming.*;
Import org.omg.CORBA*;
//must be in all CORBA applications

Public class HelloClient {
  public static void main(String args[]) {
  try {
        // create and initialize the ORB
        ORB orb = ORB.init(args, null);
        // get the root naming context
        org.omg.CORBA.object objRef =
          = orb.resolve_initial_references("NameService");
        NamingContext ncRef =
          NamingContextHelper.narrow(objRef);
```

8

Client: 2/2

```
// Resolve the object reference in naming
   NameComponent nc = new NameComponent (”Hello”, ””);
   NameComponent path[] = {nc};
   // get the stub
   Hello HelloRef =
            HelloHelper.narrow(ncRef.resolve(path));
   // Call the Hello server object and print results
    String hello = helloRef.sayHello();
    system.out.println(hello);
} catch (Exception e) {
     System.out.println(”Error : ” + e);
     e.printStachTrace(System.out);
}
}
```

9

Server object 1/3

```
Import HelloApp.*;
// contains generated stubs
Import org.omg.Cosnaming.*;
Import omg.org.cORBA.*;

Public class HelloServer {
    public static void main(String args[]);
    //registration, entering server loop
    try{
        ORB orb = ORB.init(args, null);
```

10

Server object 2/3

```
// create the servant and register with orb
   HelloServant helloRef = new HelloServant();
   orb.connect(helloRef);
//get root naming context
   omg.org.CORBA.object objRef =
      orb.resolve_initial_references(”NameService”):
   NamingContext ncRef =
      NamingContextHelper.narrow(objRef);
//bind the object reference to name
   NameComponent nc = new NameComponent(”Hello”, ””);
   NameComponent path[] = {nc};
   ncRef.rebind(path, helloRef);
```

11

Server object 3/3

```
//wait for invocations from clients
java.lang.object sync = new java.lang.Object ();
syncronized (sync)  {
    sync(wait);
}
catch (Exection e) {
    system.err.println(”Error: ” + s);
    s.printStackTrace (System.out);}}

//the actual service
Class HelloServant extends _HelloImplBase {
   public string sayHello () {
   return “\nHello world!\n”;}}
```

12

More information

- CORBA Tutorial
  - Schmidt, D., CORBA Tutorial. http://www.eng.uci.edu/~schmidt/PDF/corba4.pdf
- CORBA-sivustoja
  - http://www.corba.org/
  - http://www.cs.wustl.edu/~schmidt/corba.html
  - http://www.puder.org/corba/

13

Clusters

14

Cluster Computer Architecture

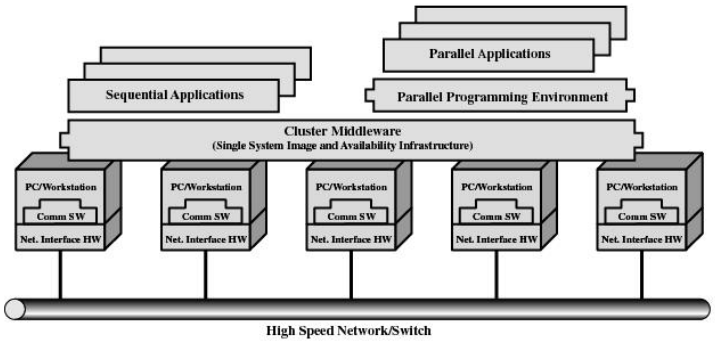


Figure 14.14 Cluster Computer Architecture [BUYY99a]

15

Classifications

Clustering Method	Description	Benefits	Limitations
Passive Standby	A secondary server takes over in case of primary server failure.	Easy to implement.	High cost because the secondary server is unavailable for other processing tasks.
Active Secondary:	The secondary server is also used for processing tasks.	Reduced cost because secondary servers can be used for processing.	Increased complexity.
- Separate Servers	Separate servers have their own disks. Data is continuously copied from primary to secondary server.	High availability.	High network and server overhead due to copying operations.
- Servers Connected to Disks	Servers are cabled to the same disks, but each server owns its disks. If one server fails, its disks are taken over by the other server.	Reduced network and server overhead due to elimination of copying operations.	Usually requires disk mirroring or RAID technology to compensate for risk of disk failure.
- Servers Share Disks	Multiple servers simultaneously share access to disks.	Low network and server overhead. Reduced risk of downtime caused by disk failure.	Requires lock manager software. Usually used with disk mirroring or RAID technology.

Windows Cluster Service

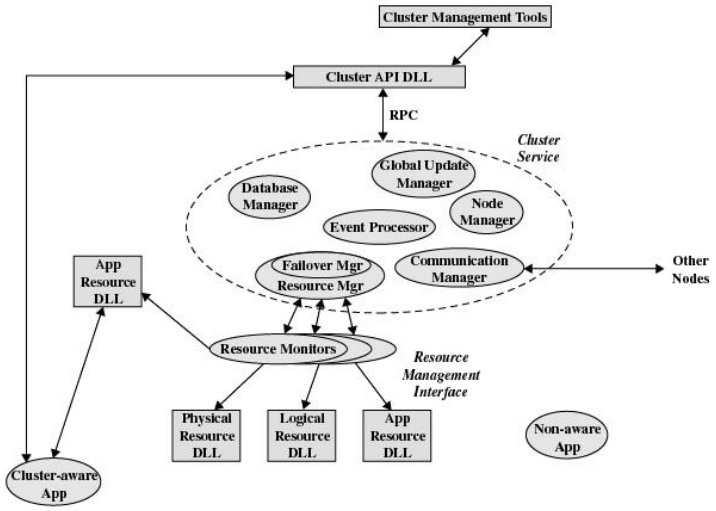


Figure 14.15 Windows Cluster Server Block Diagram [SHOR97]

17

Sun Cluster

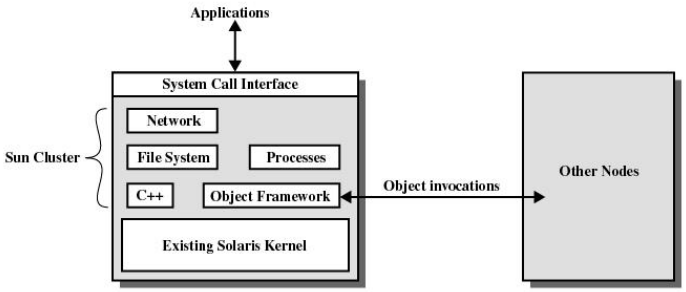


Figure 14.16 Su

- Support for objects and their communication
- Process management
- Distributed file system

18

Beowulf

www.beowulf.org

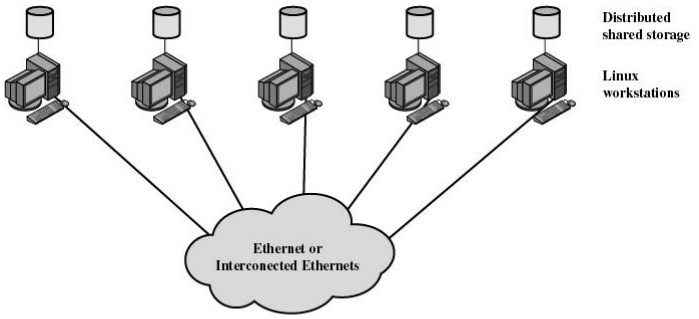


Figure 14.18 Generic Beowulf Configuration

19

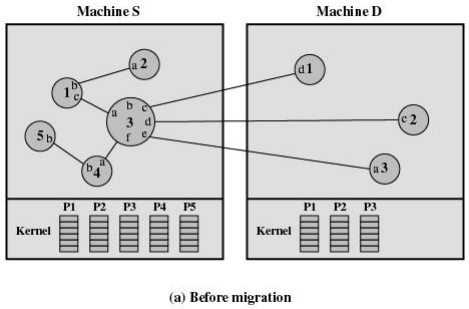
LUENTO 20

Distributed Process Management

Stallings, chapter 15

20

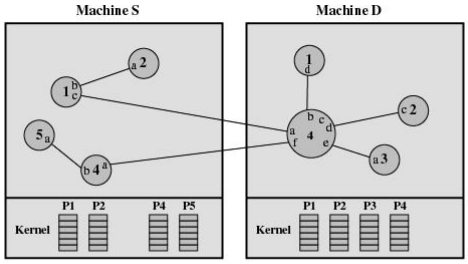
Process Migration



(a) Before migration

Prosessi 3 (solmu S)
Prosessiksi 4 (solmu D)

- Load balancing, availability, move closer to a resource



(b) After migration

21

Transfer negotiation: Charlotte

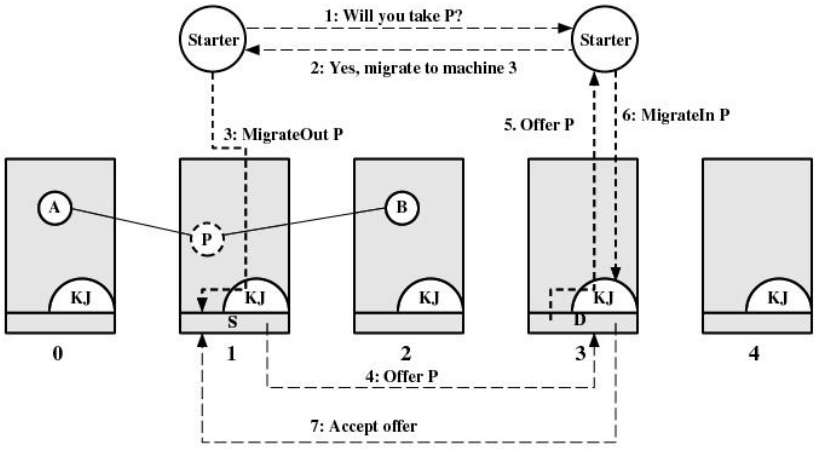
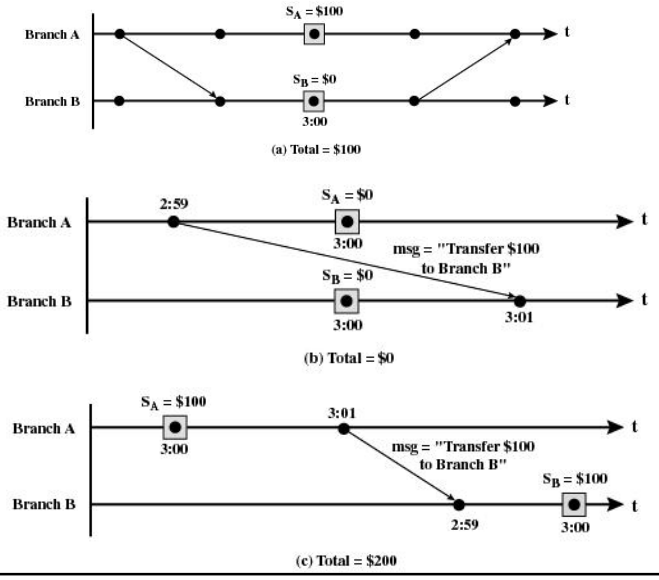


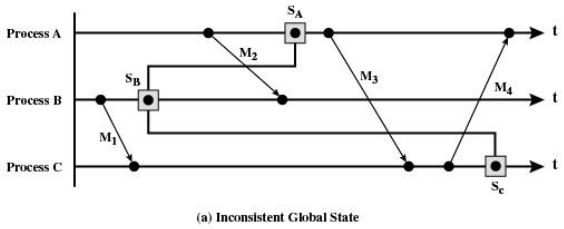
Figure 15.2 Negotiation of Process Migration

Global state



23

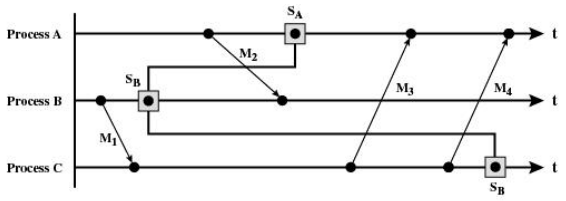
Inconsistent Global State



- A correct global state (combination of snapshots and messages currently on way, does not allow messages going from future to the past.

24

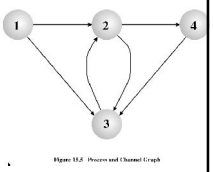
Consistent Global State



(b) Consistent Global State

25

Distributed Snapshot Algorithm



Process 1 Outgoing channels 2 sent 1, 2, 3, 4, 5, 6 3 sent 1, 2, 3, 4, 5, 6 Incoming channels	**Process 3** Outgoing channels 2 sent 1, 2, 3, 4, 5, 6, 7, 8 Incoming channels 1 received 1, 2, 3 stored 4, 5, 6 2 received 1, 2, 3 stored 4 4 received 1, 2, 3
Process 2 Outgoing channels 3 sent 1, 2, 3, 4 4 sent 1, 2, 3, 4 Incoming channels 1 received 1, 2, 3, 4 stored 5, 6 3 received 1, 2, 3, 4, 5, 6, 7, 8	**Process 4** Outgoing channels 3 sent 1, 2, 3 Incoming channels 2 received 1, 2 stored 3, 4

26

Lamport: logical time stamps

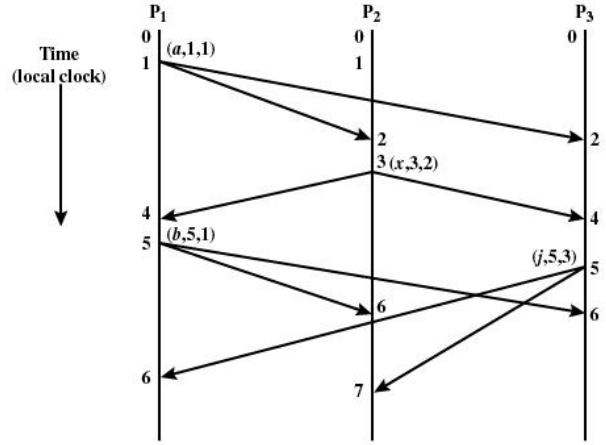


Figure 15.8 Example of Operation of Timestamping Algorithm

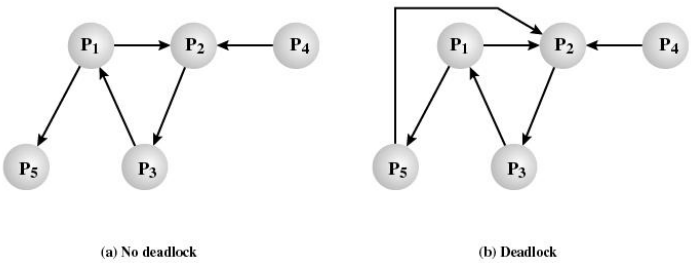
27

Distributed mutual exclusion and deadlocks

28

Lukkiuma sanomanvälityksessä

- Mutual Waiting
  - All nodes are locked simply because they are all waiting for a message from someone else and there is no message on the way.



(a) No deadlock (b) Deadlock

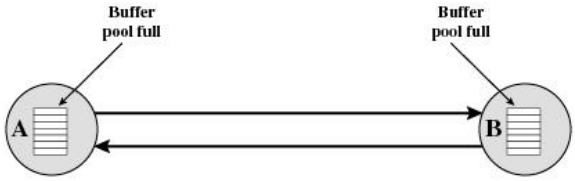
P1 odottaa joko P2 tai P5

Figure 15.16 Deadlock in Message Communication

29

Lock in message passing

- Buffers full
  - A's buffer is full of messages going to B and B's buffer is full of messages going to A and neither has any free space to receive any messages.



(a) Direct store-and-forward deadlock

30