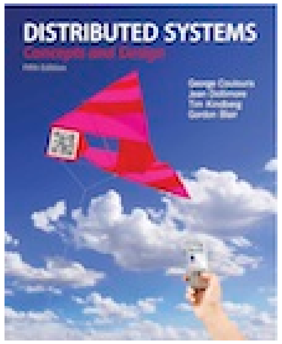
# Slides for Chapter 5: Operating System support

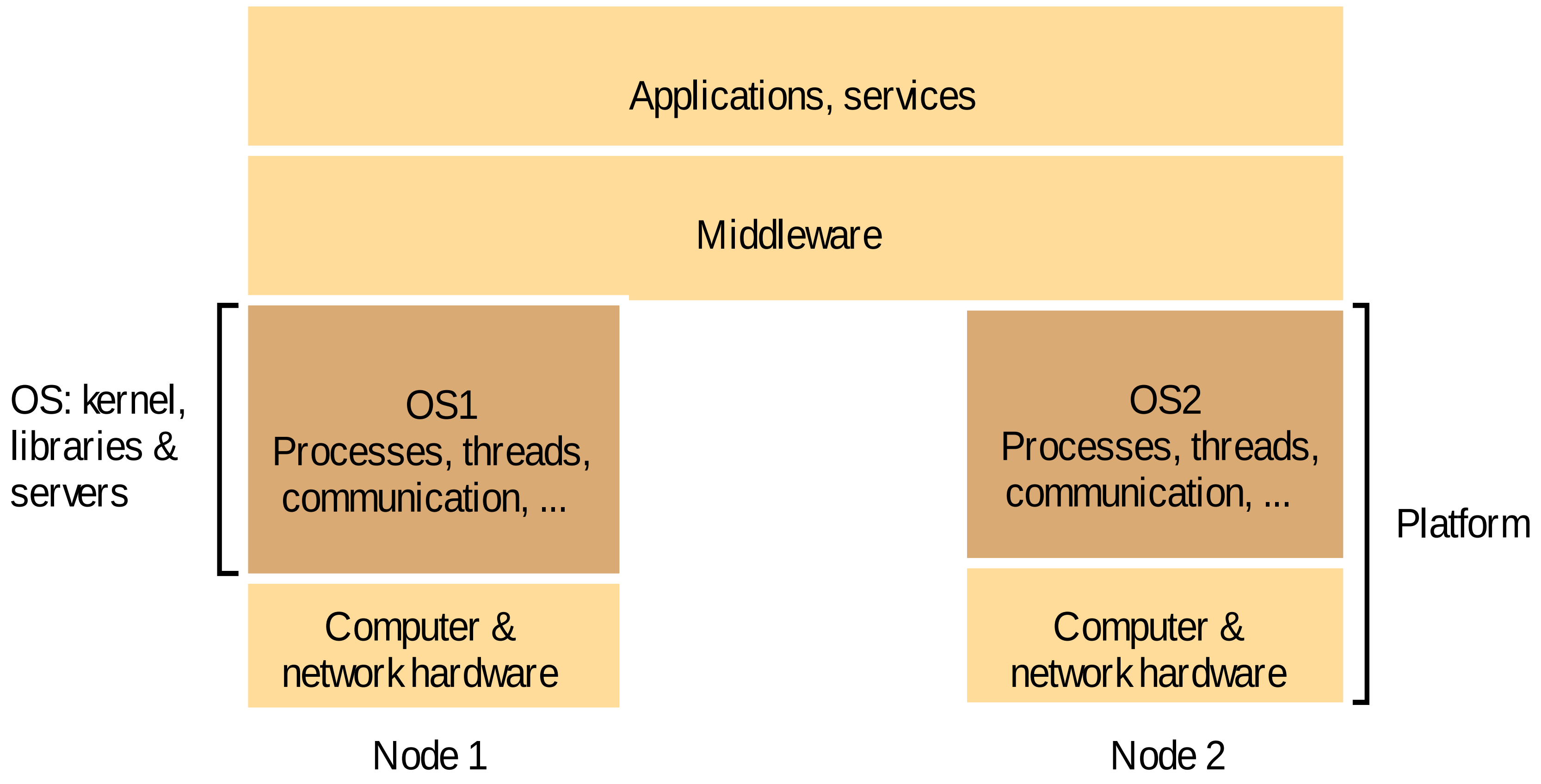


*From* Coulouris, Dollimore, Kindberg and Blair

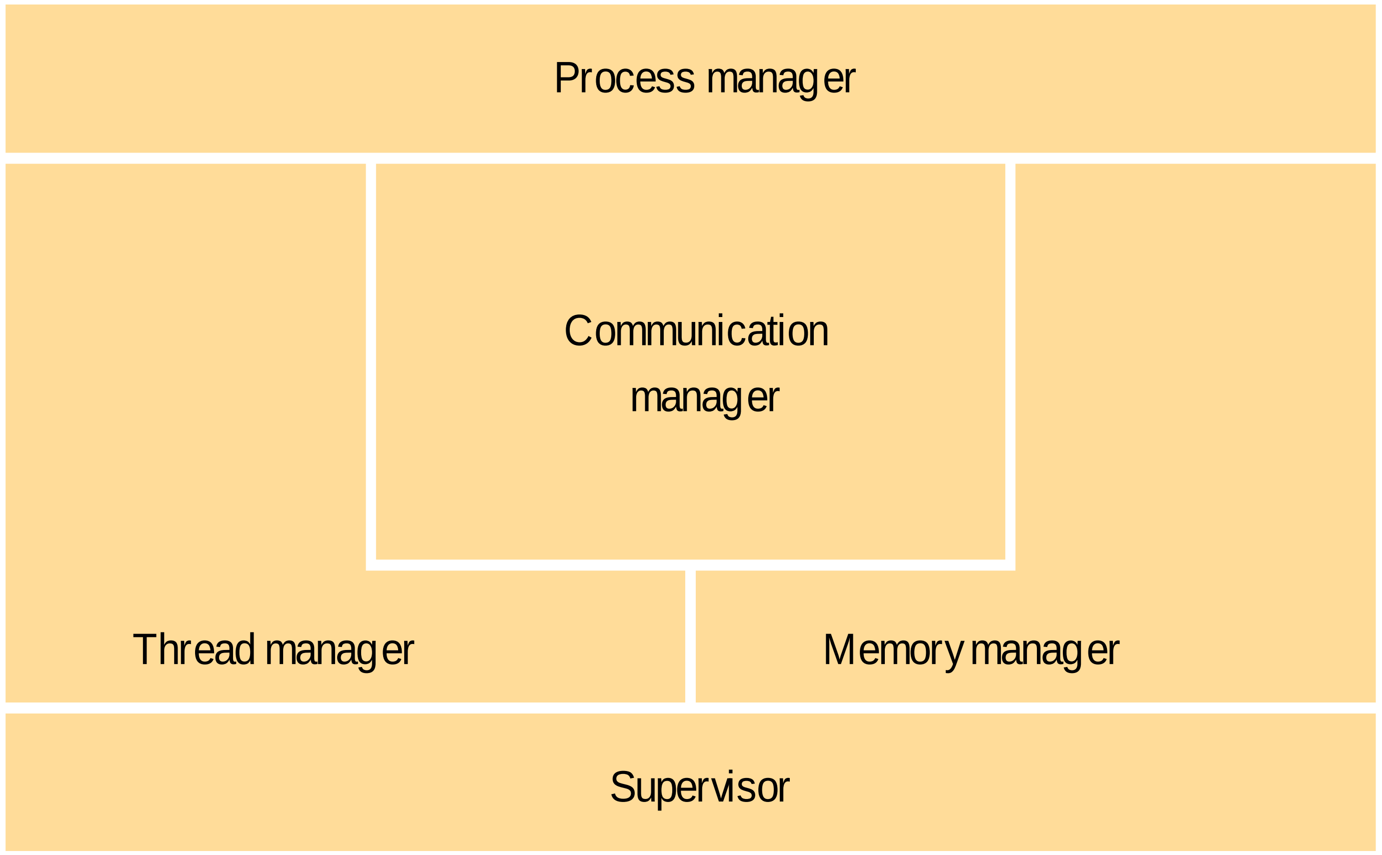
Distributed Systems:
Concepts and Design

Edition 5, © Addison-Wesley 2012

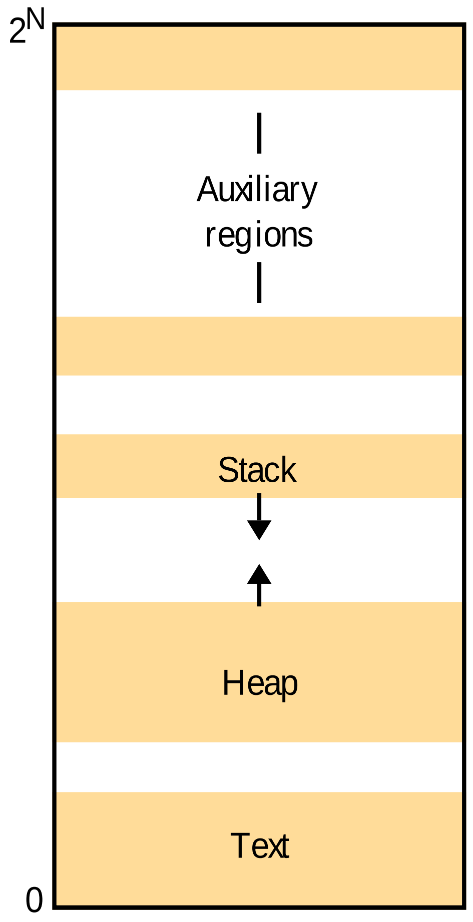
# Figure 5.1
# System layers


Applications, services
Middleware
OS: kernel, libraries & servers
OS1 Processes, threads, communication, ...
OS2 Processes, threads, communication, ...
Platform
Computer & network hardware
Computer & network hardware
Node 1
Node 2

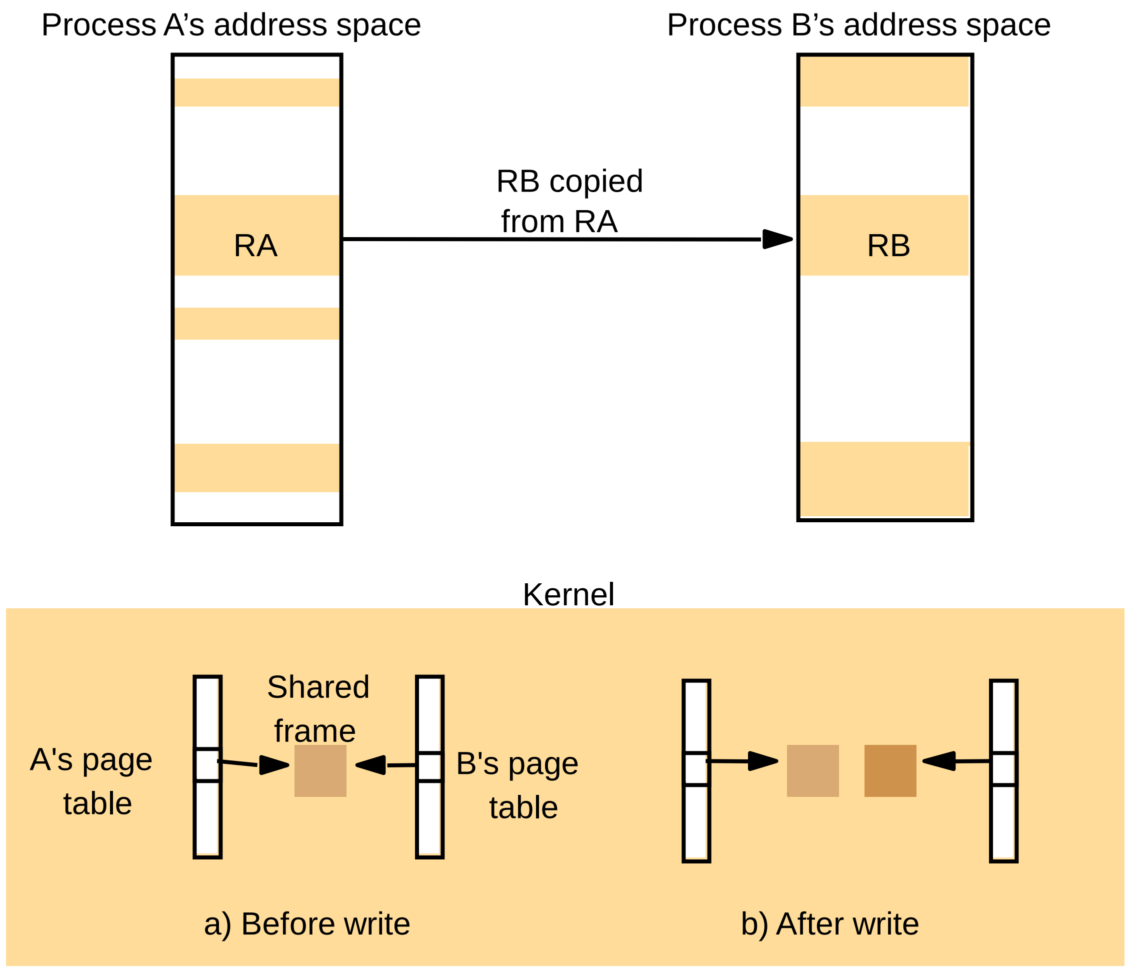
## Figure 5.2
## Core OS functionality



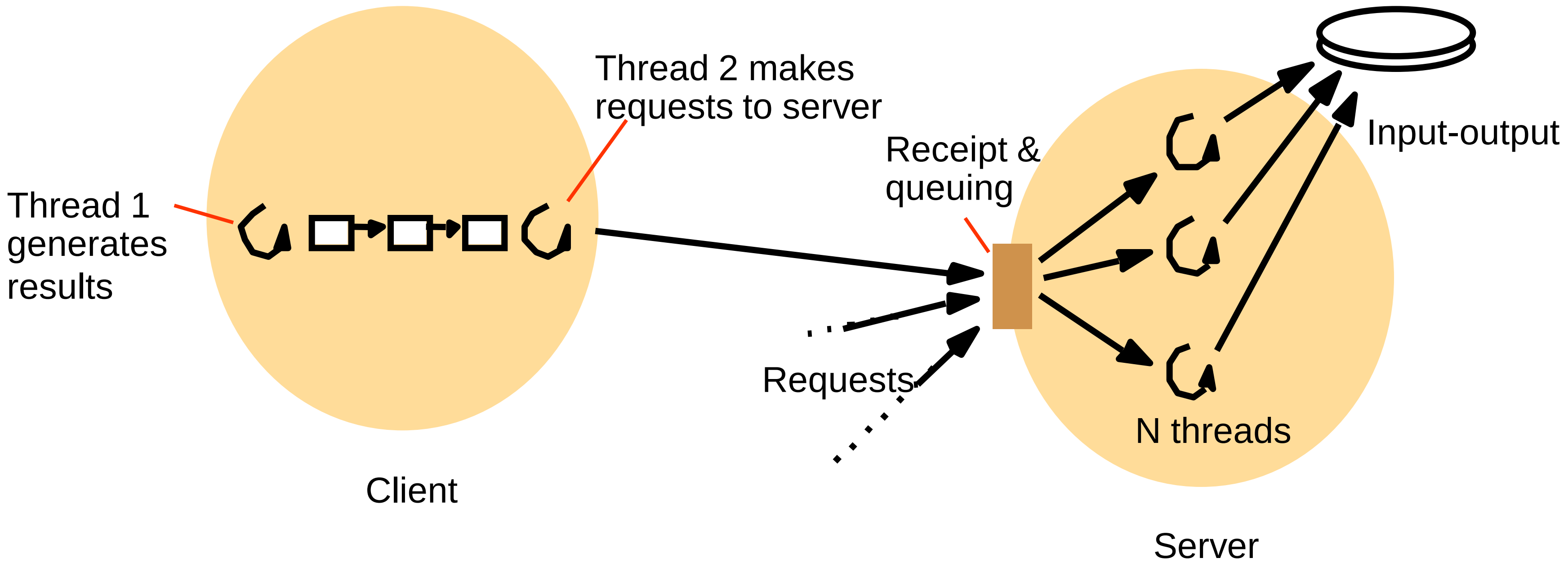
# Figure 5.3
# Address space



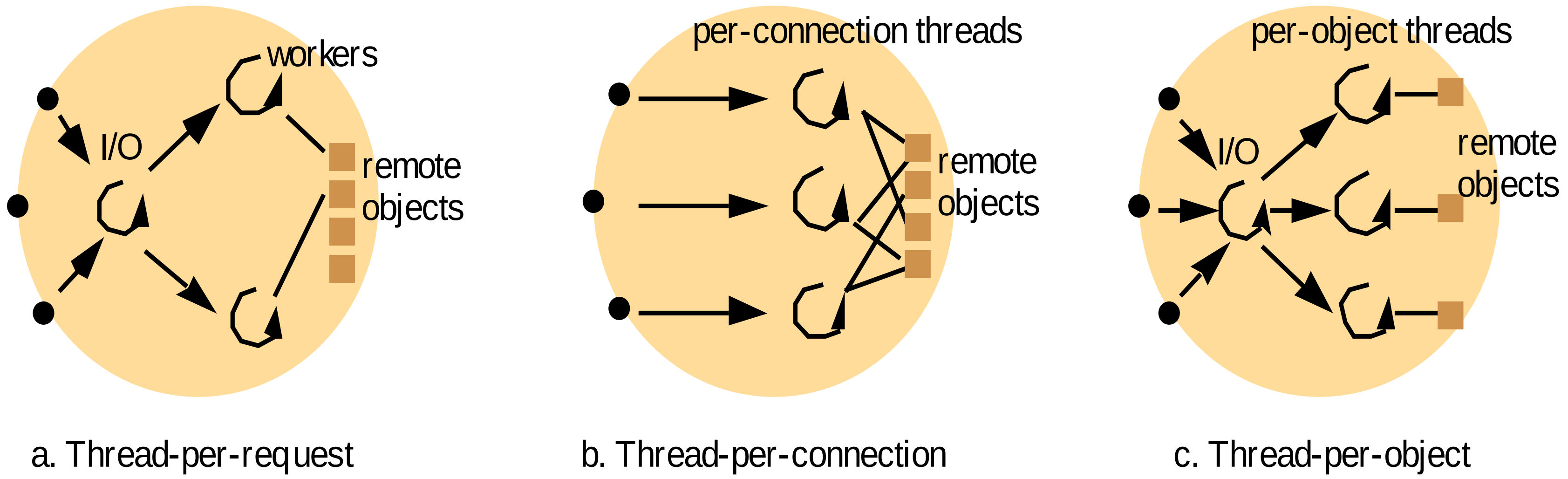
# Figure 5.4
# Copy-on-write



## Figure 5.5
## Client and server with threads



## Figure 5.6
## Alternative server threading architectures (see also Figure 5.5)



## Figure 5.7
## State associated with execution environments and threads

| *Execution environment* | *Thread* |
|---|---|
| Address space tables | Saved processor registers |
| Communication interfaces, open files | Priority and execution state (such as *BLOCKED)* |
| Semaphores, other synchronization objects | Software interrupt handling information |
| List of thread identifiers | Execution environment identifier |
| Pages of address space resident in memory; hardware cache entries | |

## Figure 5.8
## Java thread constructor and management methods

*Thread(ThreadGroup group, Runnable target, String name)*
Creates a new thread in the *SUSPENDED* state, which will belong to *group* and be identified as *name*; the thread will execute the *run()* method of *target*.

*setPriority(int newPriority), getPriority()*
Set and return the thread's priority.

*run()*
A thread executes the *run()* method of its target object, if it has one, and otherwise its own *run()* method (*Thread* implements *Runnable*).

*start()*
Change the state of the thread from *SUSPENDED* to *RUNNABLE*.

*sleep(int millisecs)*
Cause the thread to enter the *SUSPENDED* state for the specified time.

*yield()*
Causes the thread to enter the *READY* state and invoke the scheduler.

*destroy()*
Destroy the thread.

## Figure 5.9
## Java thread synchronization calls

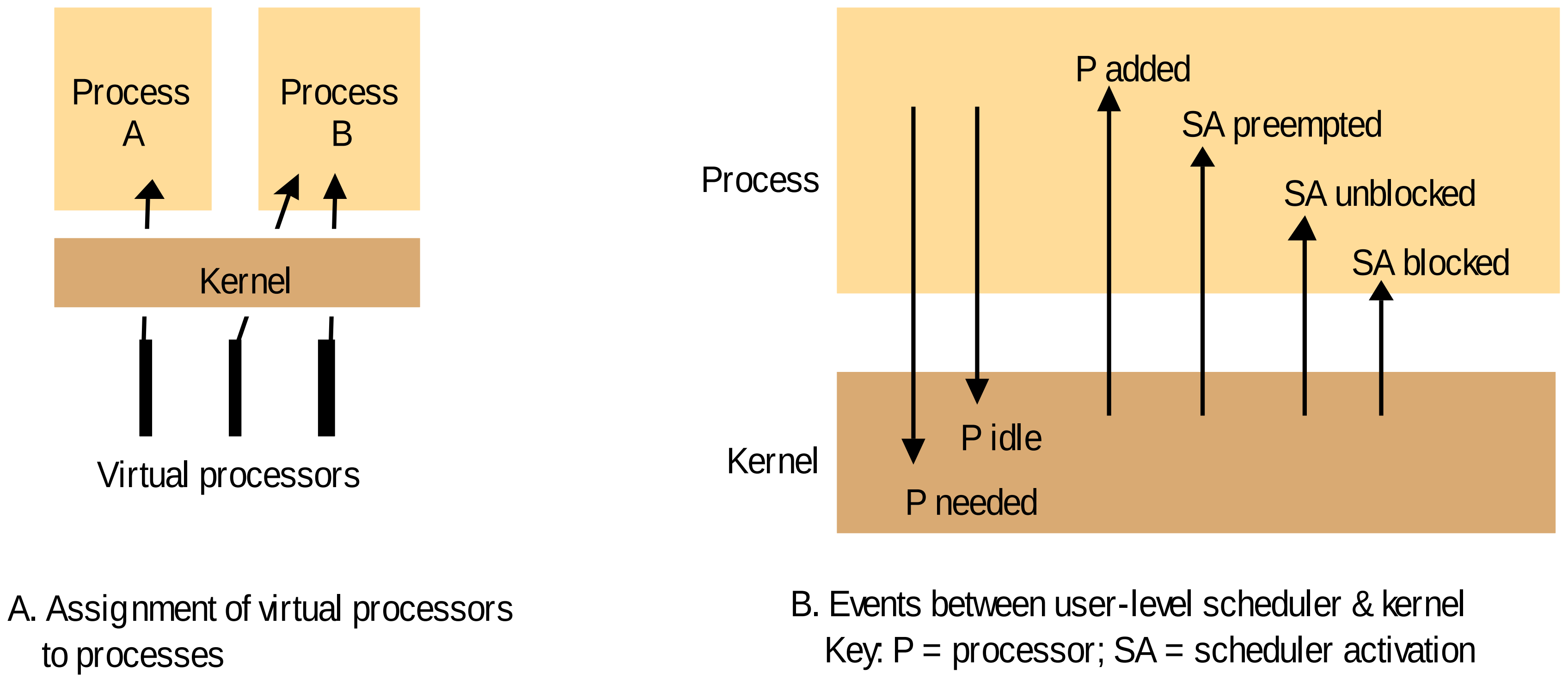
*thread.join(int millisecs)*
: Blocks the calling thread for up to the specified time until *thread* has terminated.

*thread.interrupt()*
: Interrupts *thread*: causes it to return from a blocking method call such as *sleep()*.

*object.wait(long millisecs, int nanosecs)*
: Blocks the calling thread until a call made to *notify()* or *notifyAll()* on *object* wakes the thread, or the thread is interrupted, or the specified time has elapsed.

*object.notify(), object.notifyAll()*
: Wakes, respectively, one or all of any threads that have called *wait()* on *object*.

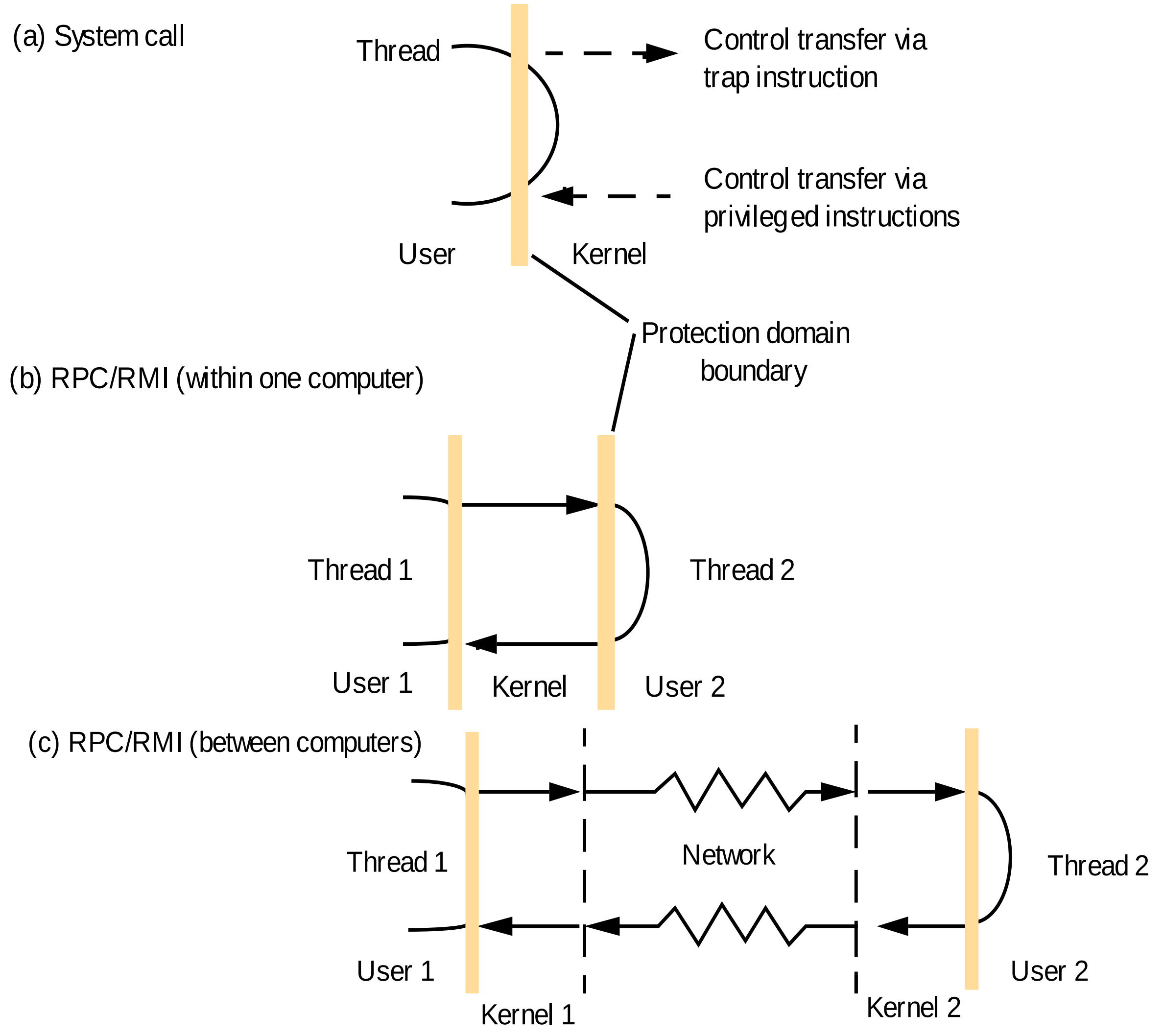
# Figure 5.10
# Scheduler activations



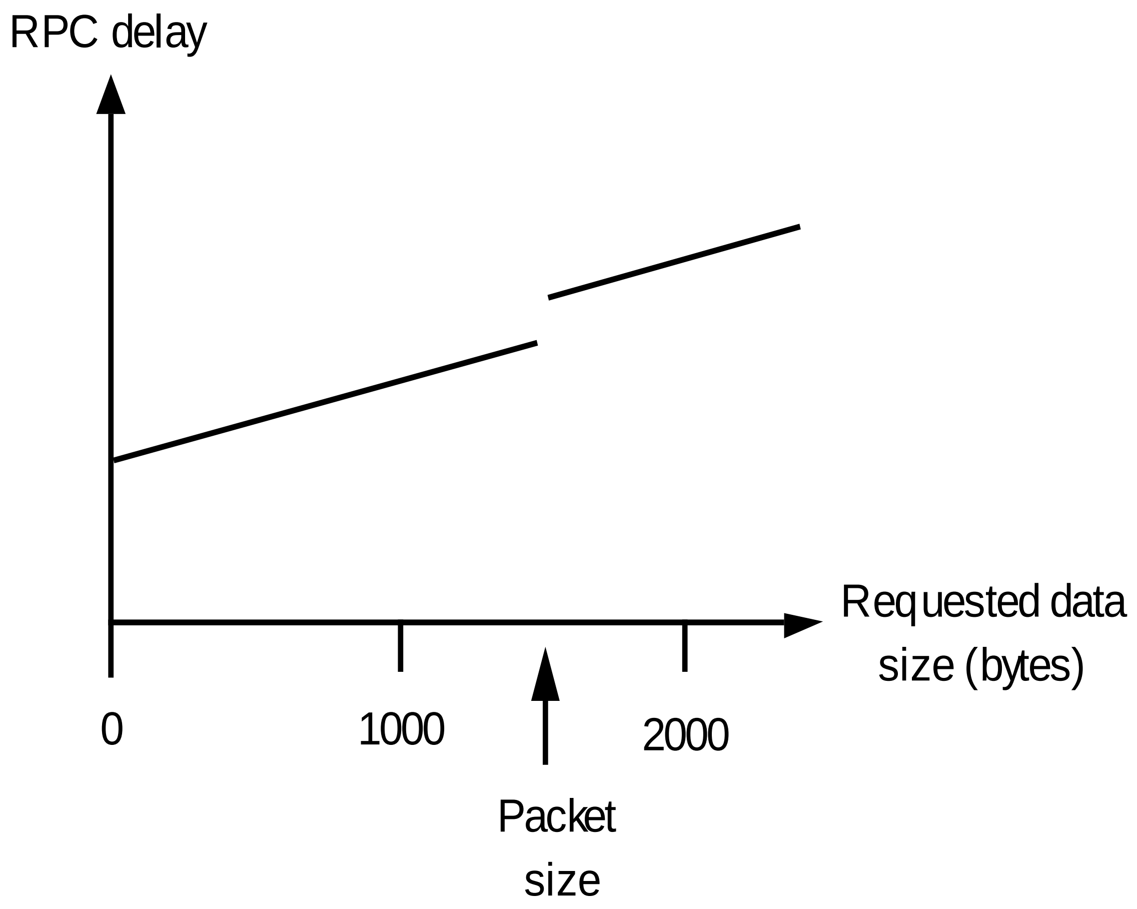
A. Assignment of virtual processors to processes

B. Events between user-level scheduler & kernel
Key: P = processor; SA = scheduler activation

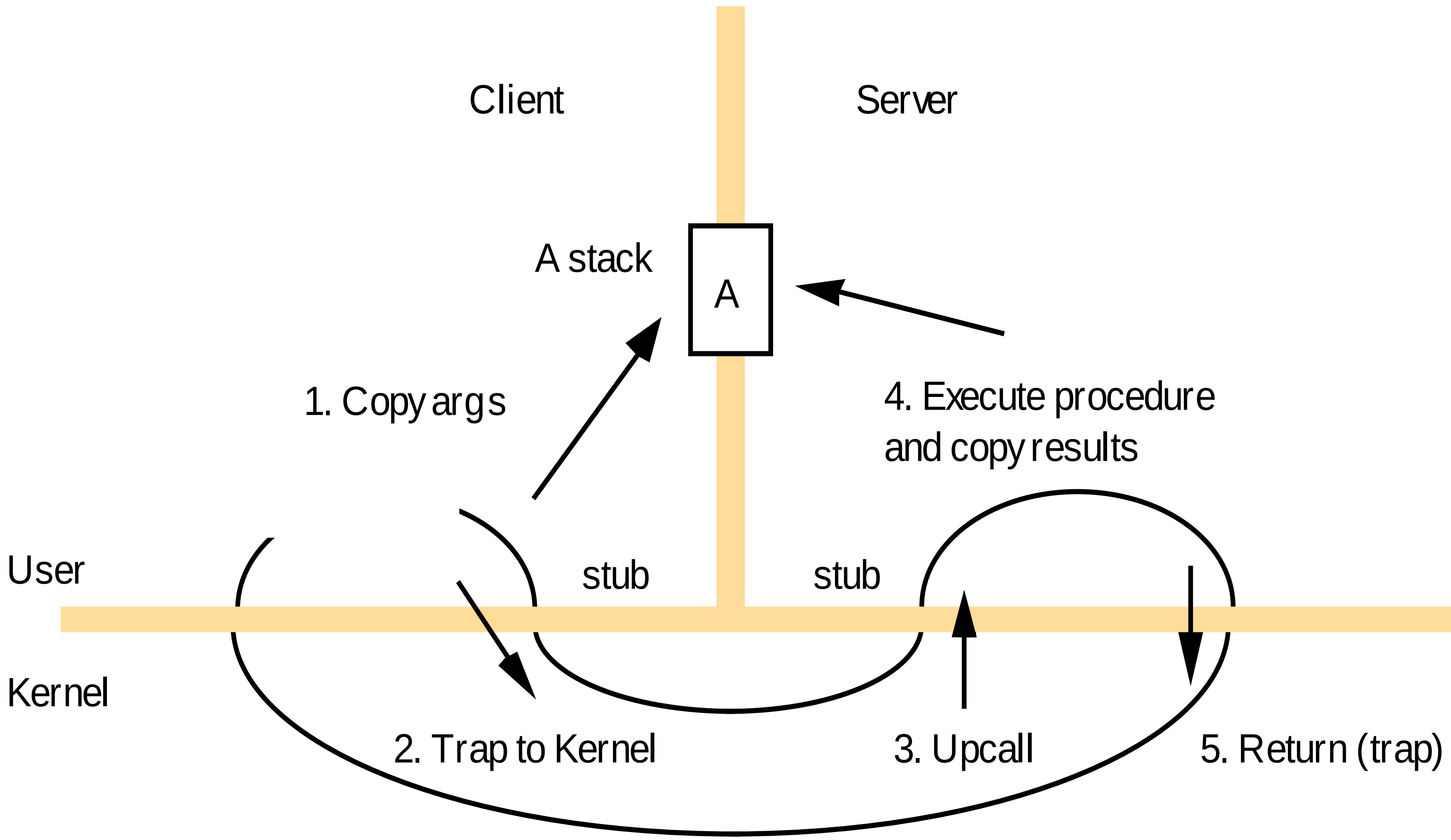
## Figure 5.11
## Invocations between address spaces


(a) System call
Thread
Control transfer via trap instruction
Control transfer via privileged instructions
User
Kernel
Protection domain boundary
(b) RPC/RMI (within one computer)
Thread 1
Thread 2
User 1
Kernel
User 2
(c) RPC/RMI (between computers)
Thread 1
Network
Thread 2
User 1
User 2
Kernel 1
Kernel 2

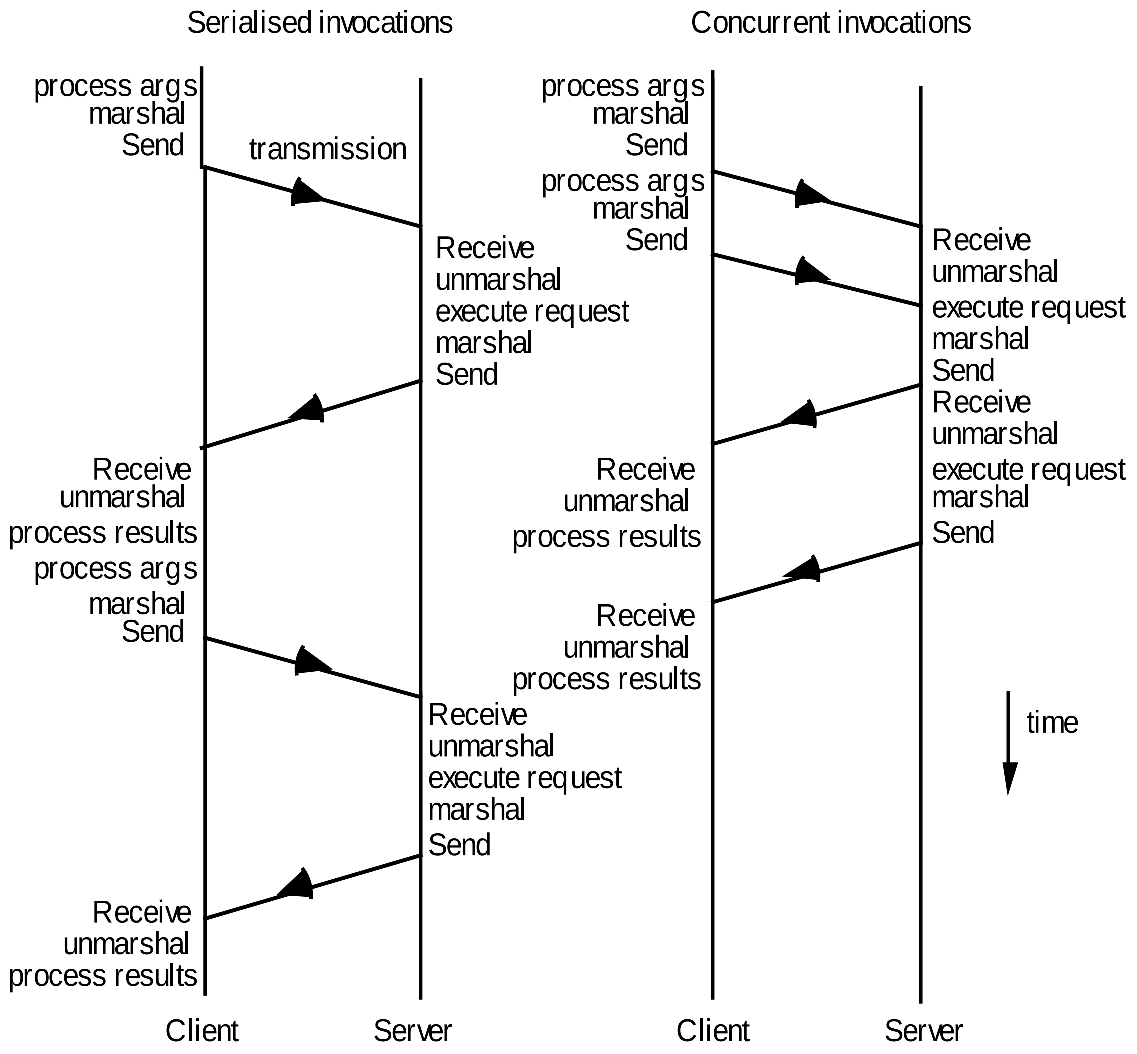
## Figure 5.12
## RPC delay against parameter size



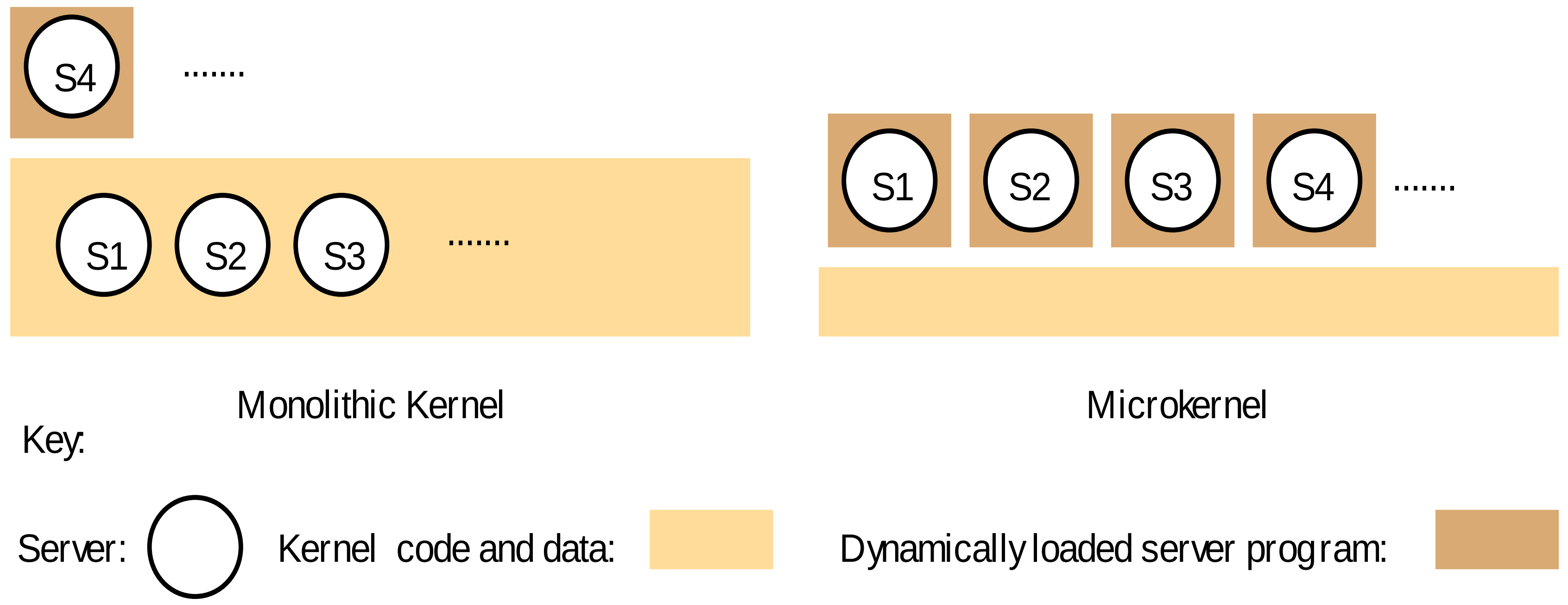
# Figure 5.13
A lightweight remote procedure call



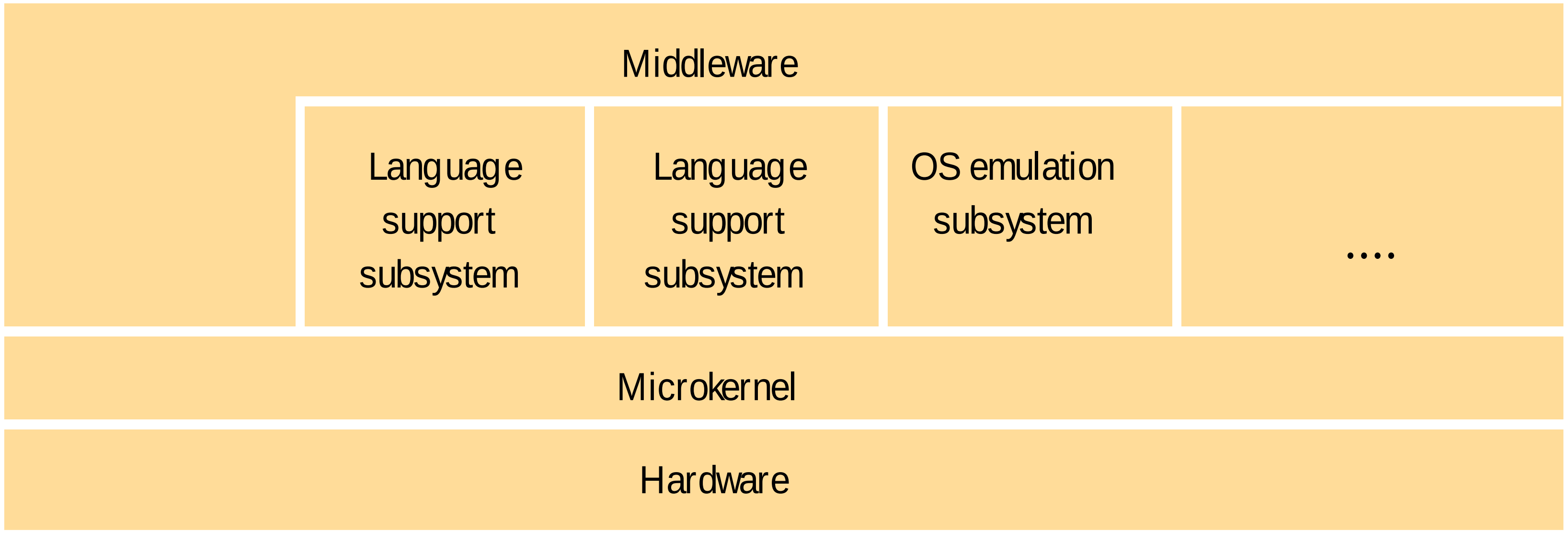
# Figure 5.14
# Times for serialized and concurrent invocations



## Figure 5.15
## Monolithic kernel and microkernel

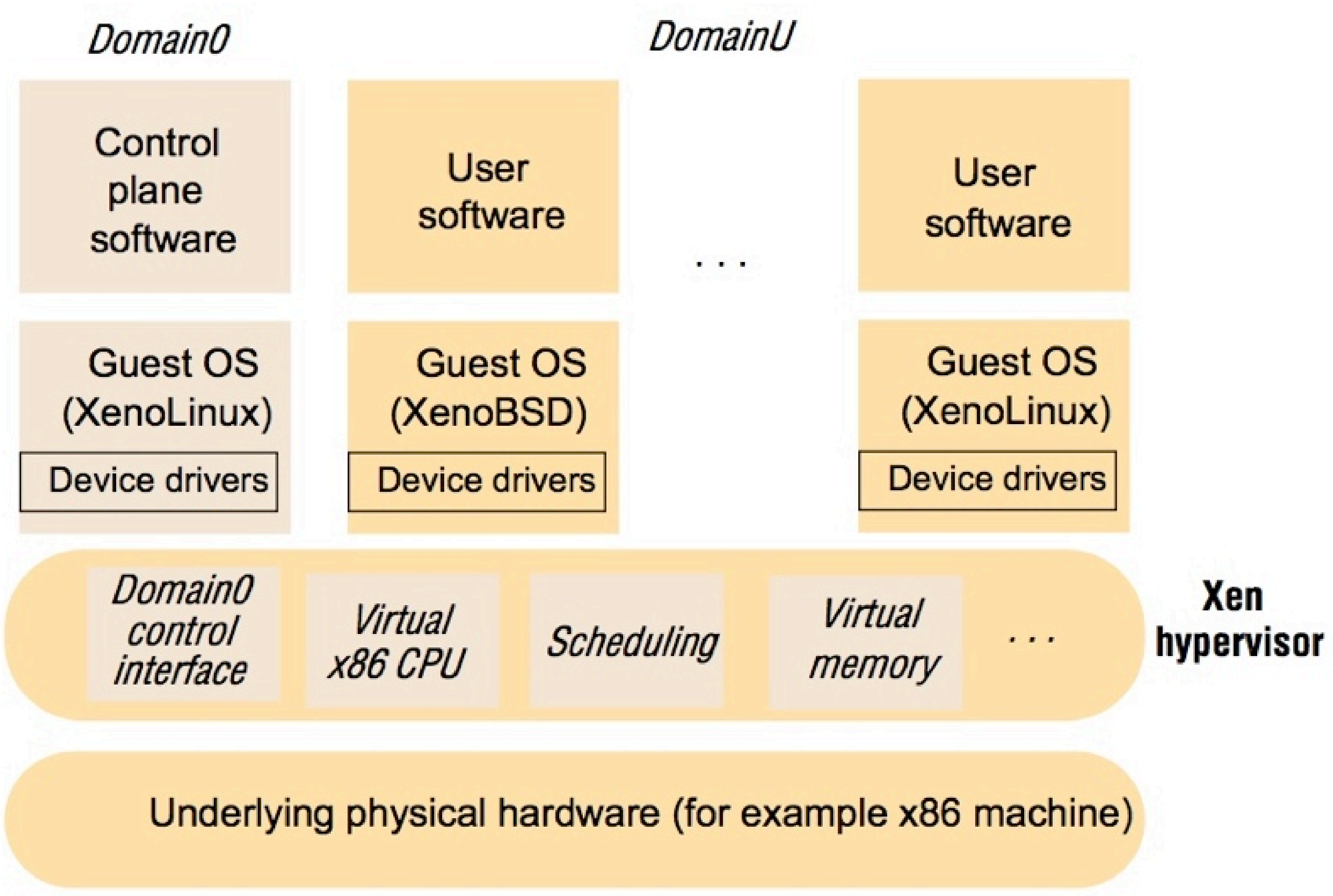


## Figure 5.16
## The role of the microkernel

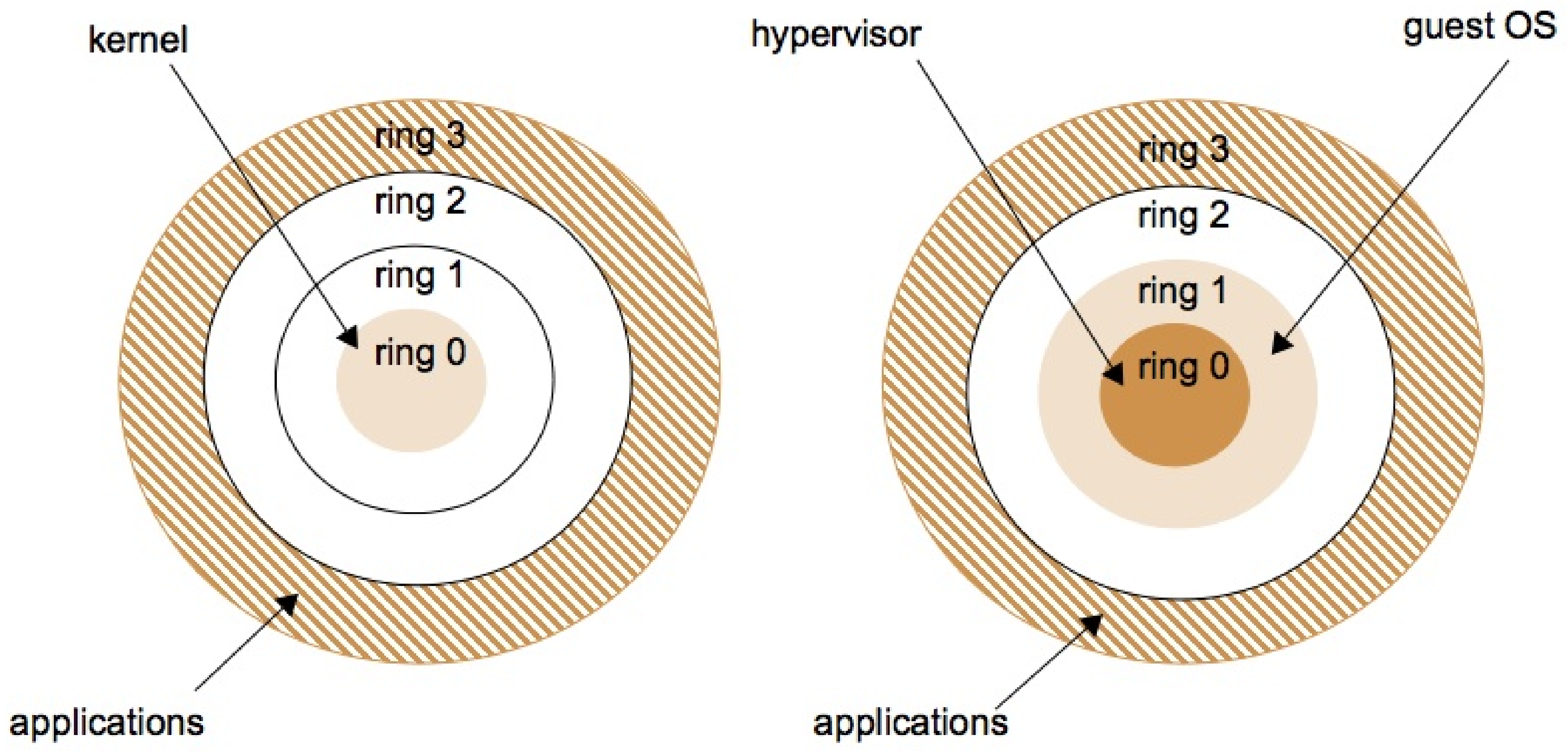


The microkernel supports middleware via subsystems

## Figure 5.17
## The architecture of Xen



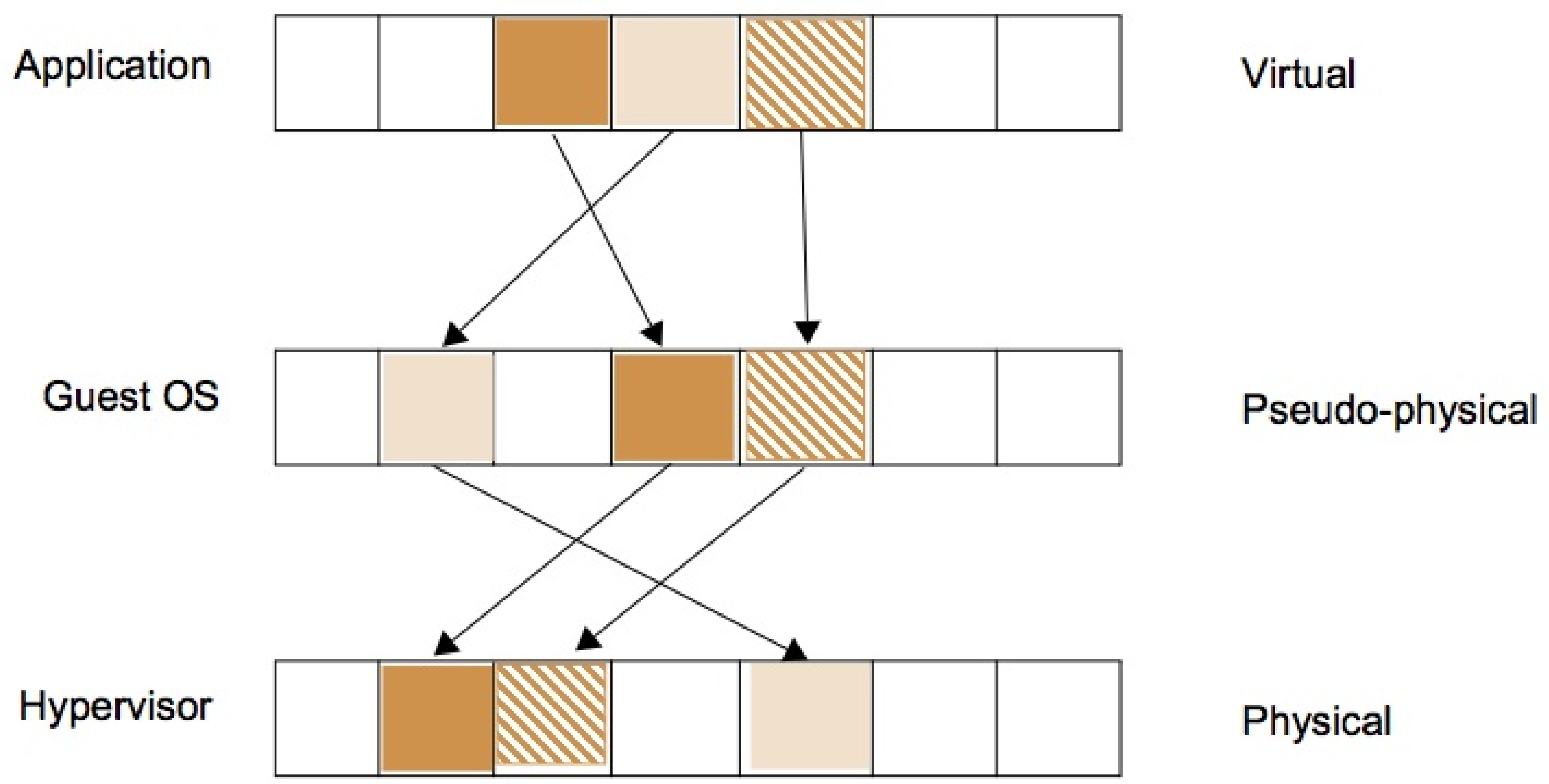
## Figure 5.18
## Use of rings of privilege



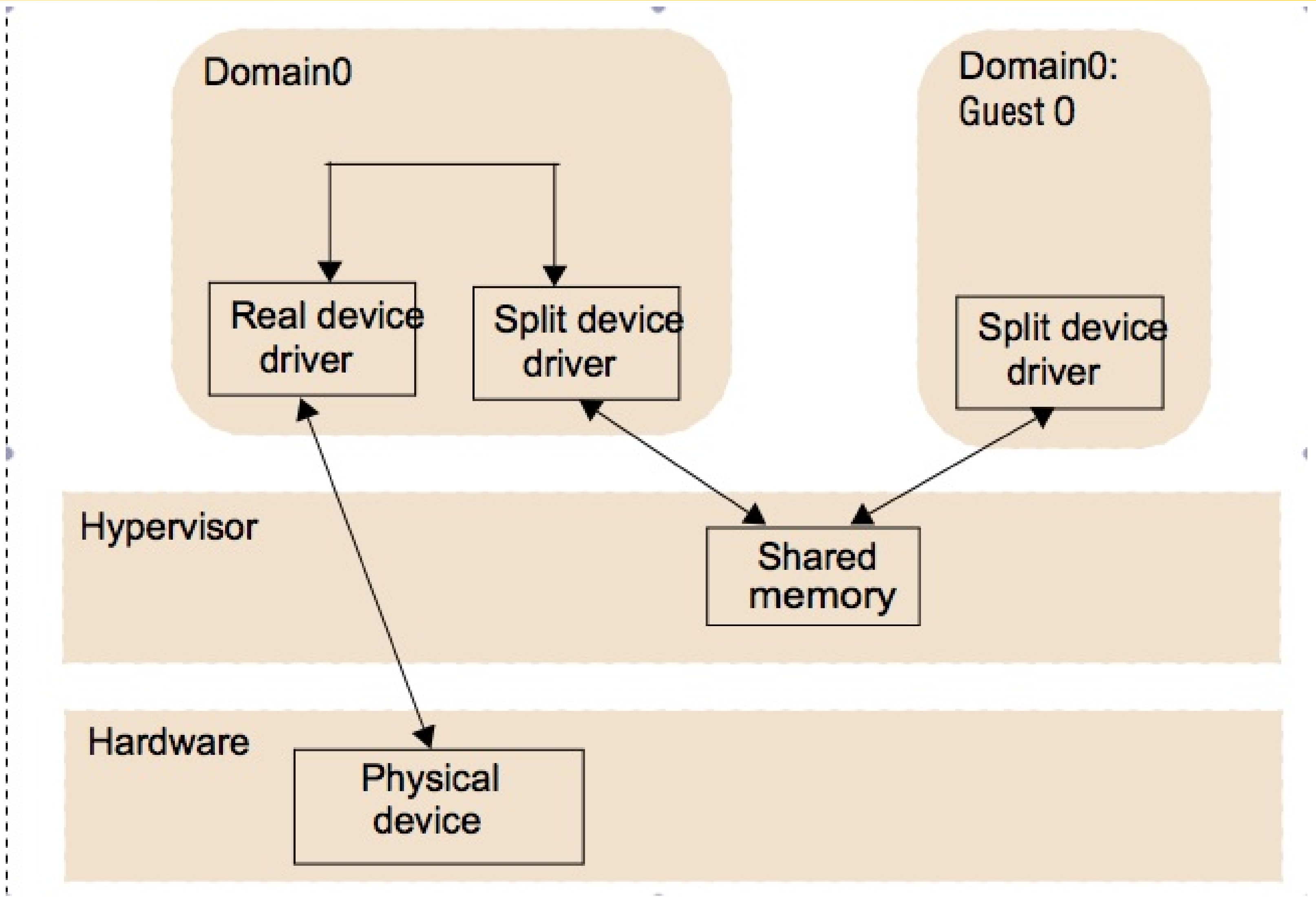
a) kernel-based operating systems

b) paravirtualization in Xen

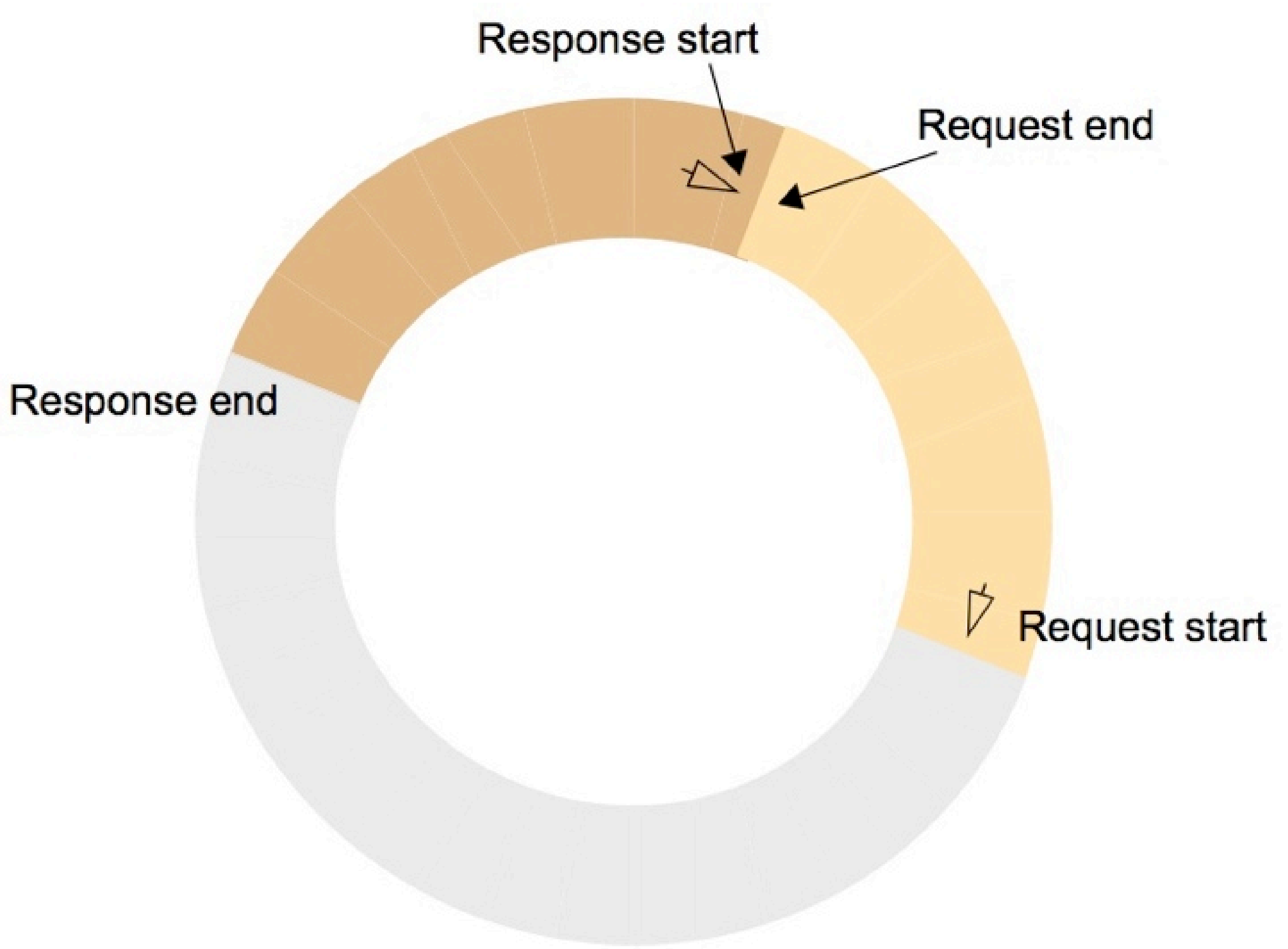
## Figure 5.19
## Virtualization of memory management



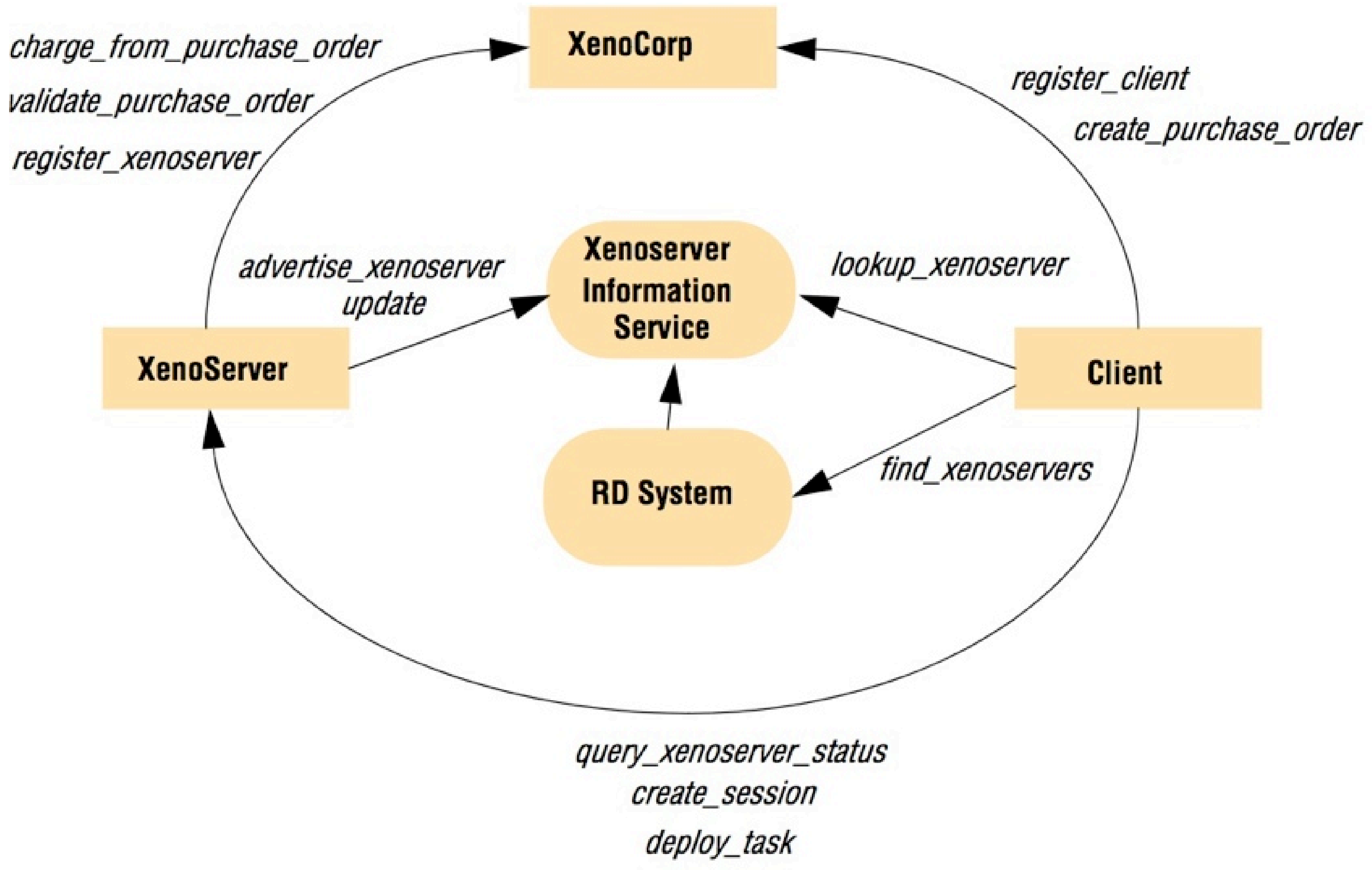
## Figure 5.20
## Split device drivers



## Figure 5.21
## I/O rings



# Figure 5.22
# The XenoServer Open Platform Architecture


XenoCorp
charge_from_purchase_order
validate_purchase_order
register_xenoserver
register_client
create_purchase_order
Xenoserver Information Service
advertise_xenoserver
update
lookup_xenoserver
XenoServer
Client
RD System
find_xenoservers
query_xenoserver_status
create_session
deploy_task