

Figure 6.1 System layers

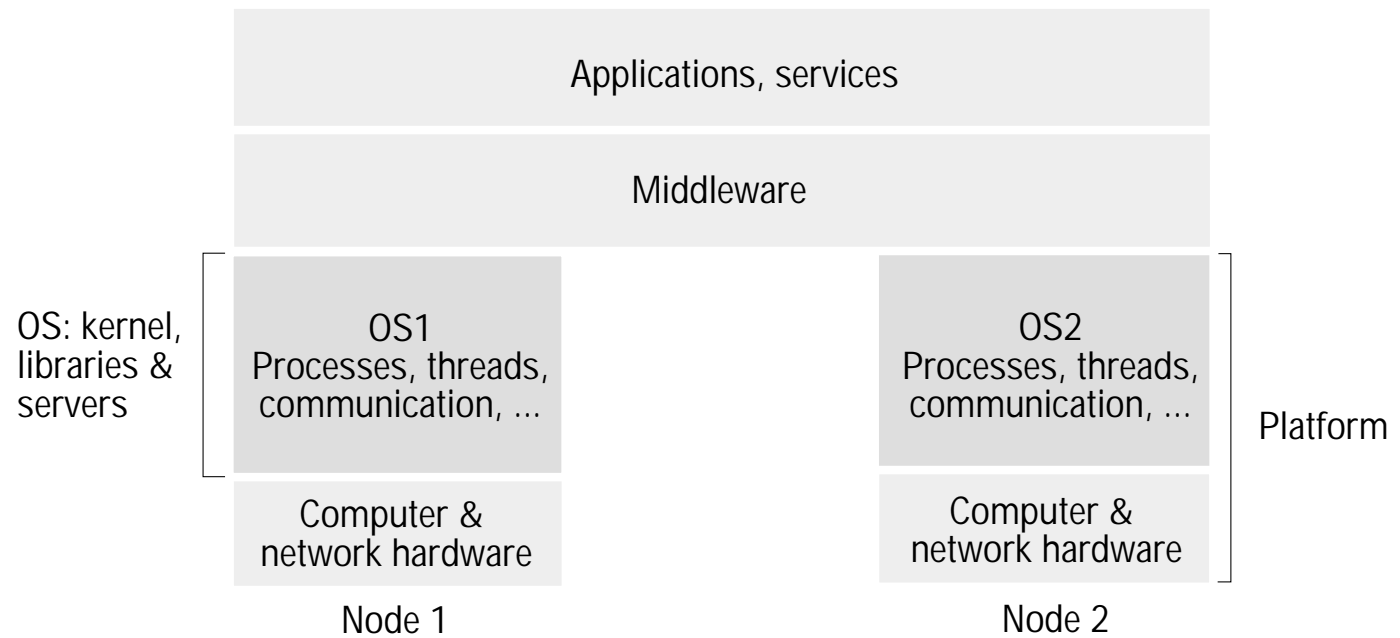


Figure 6.2 Core OS functionality

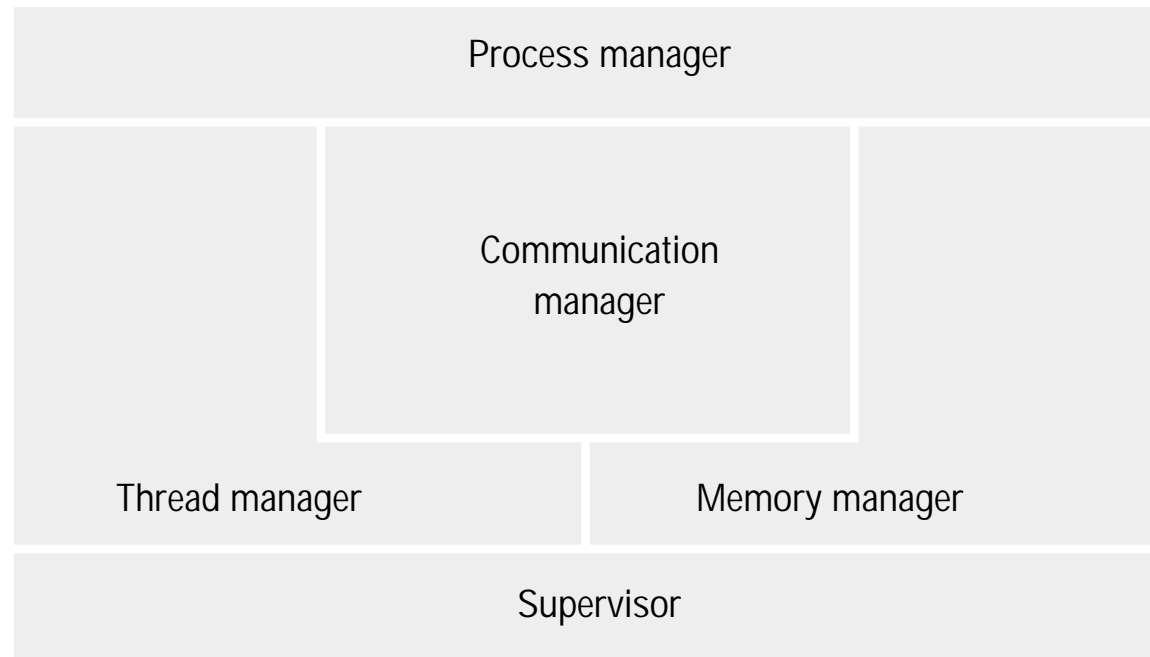


Figure 6.3 Address space

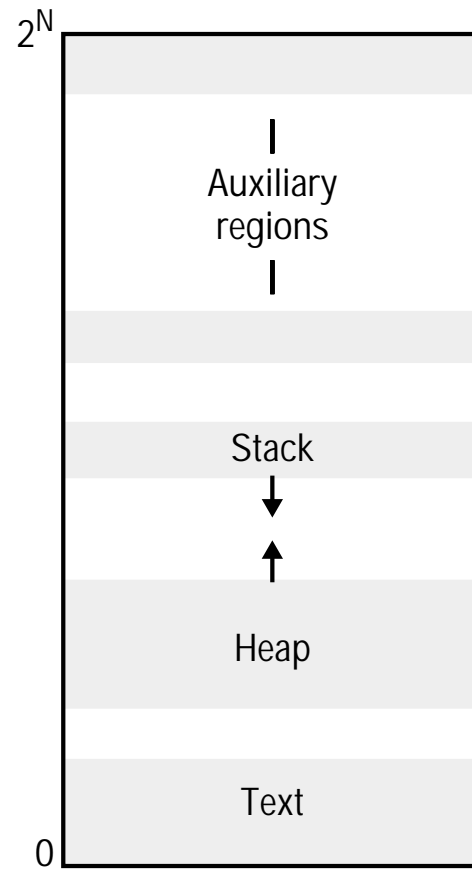


Figure 6.4 Copy-on-write

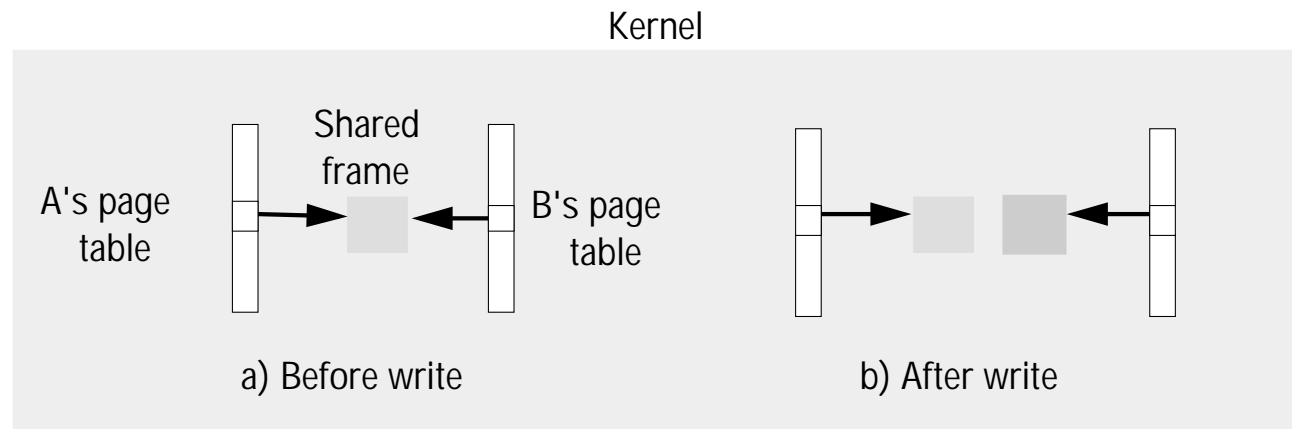
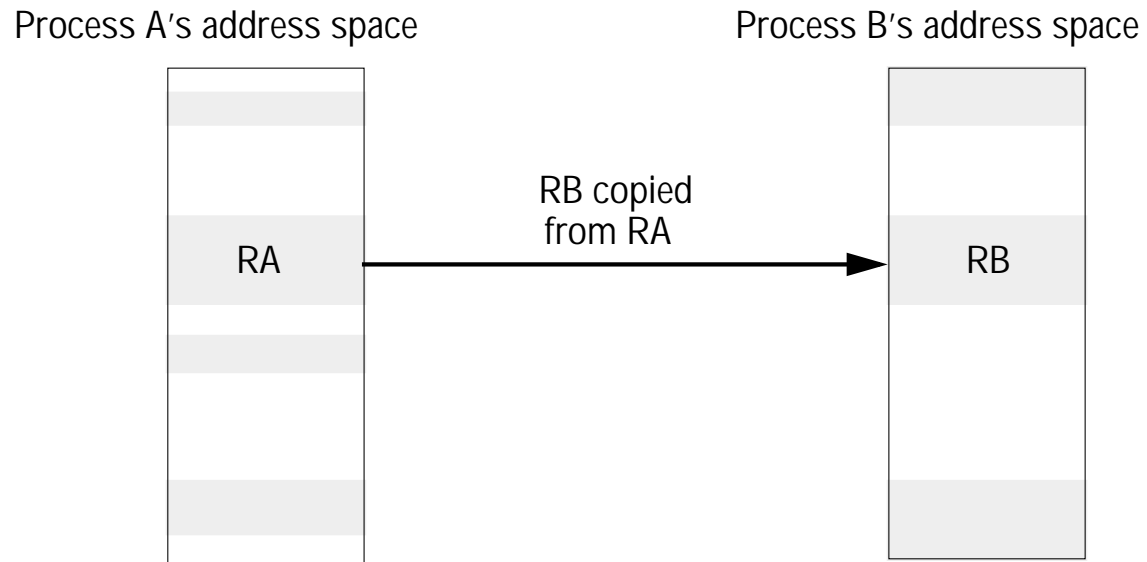


Figure 6.5 Client and server with threads

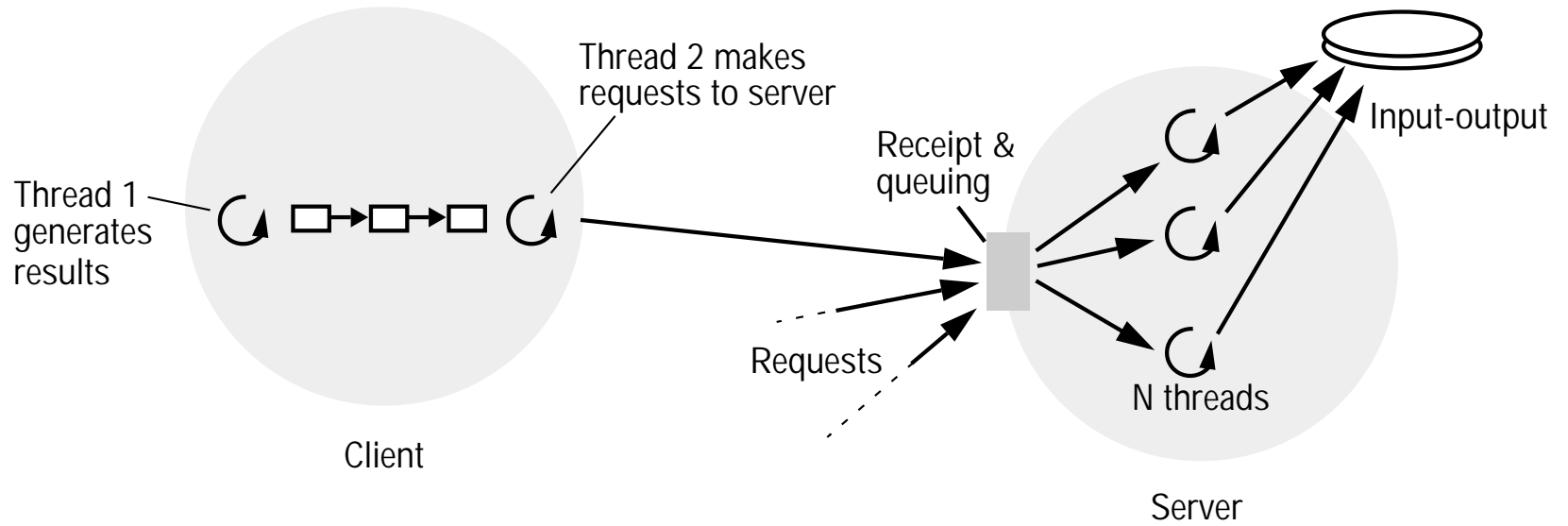


Figure 6.6 Alternative server threading architectures (see also Figure 6.5)

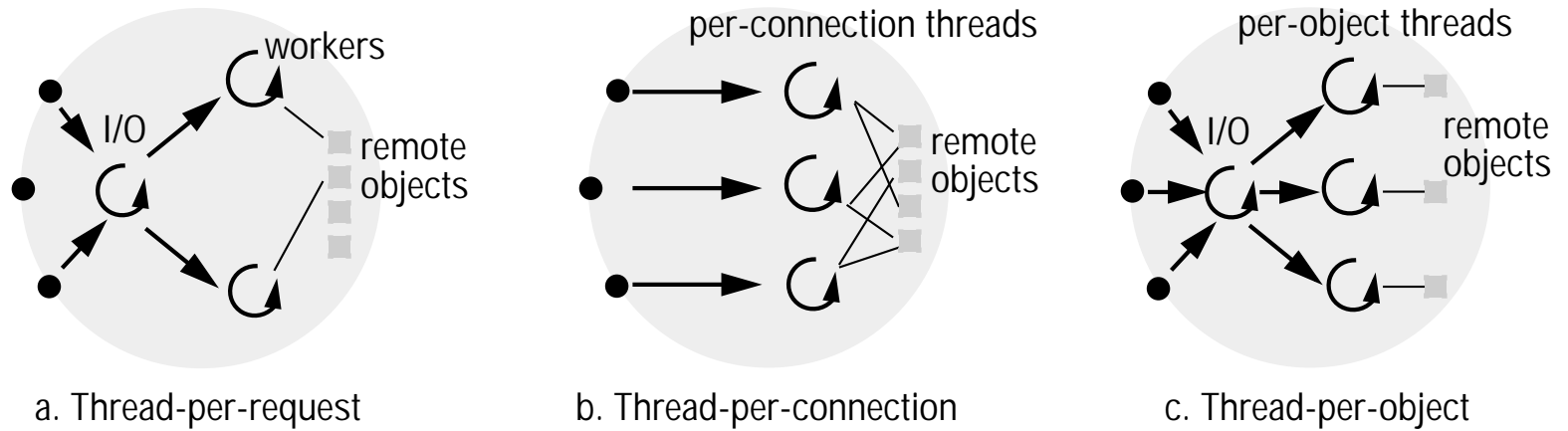


Figure 6.7 State associated with execution environments and threads

<i>Execution environment</i>	<i>Thread</i>
Address space tables	Saved processor registers
Communication interfaces, open files	Priority and execution state (such as <i>BLOCKED</i>)
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 6.8 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()

Enter the *READY* state and invoke the scheduler.

destroy()

Destroy the thread.

Figure 6.9 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()*.

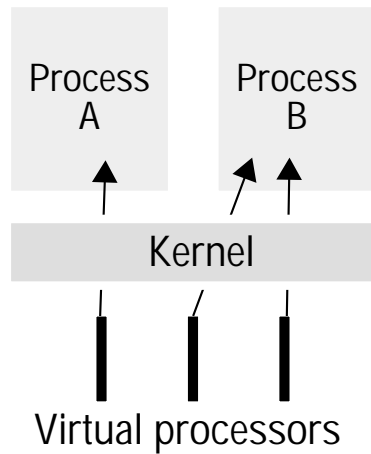
object.wait(long millisecs, int nanosecs)

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.

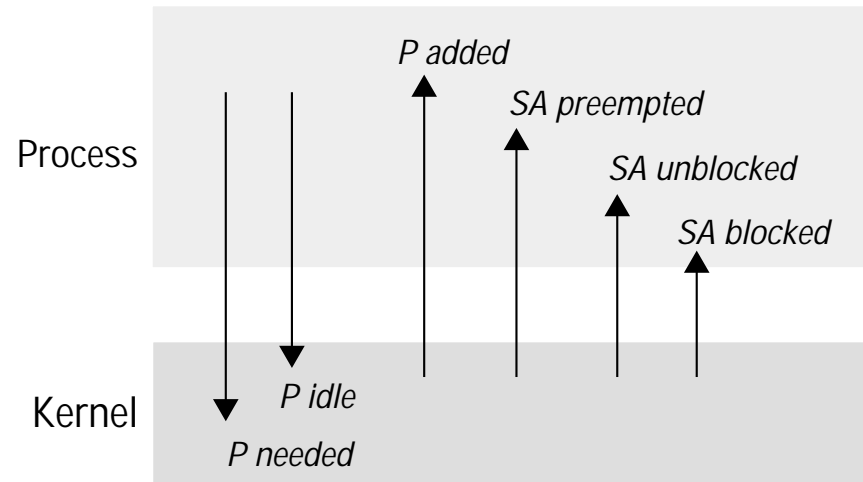
object.notify(), *object.notifyAll()*

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

Figure 6.10 Scheduler activations



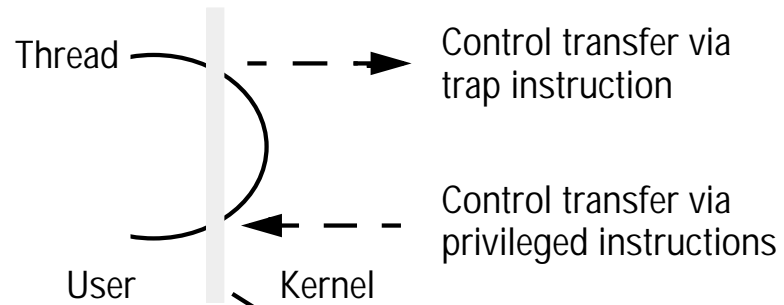
A. Assignment of virtual processors to processes



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

Figure 6.11 Invocations between address spaces (cont'd on next slide)

(a) System call



(b) RPC/RMI (within one computer)

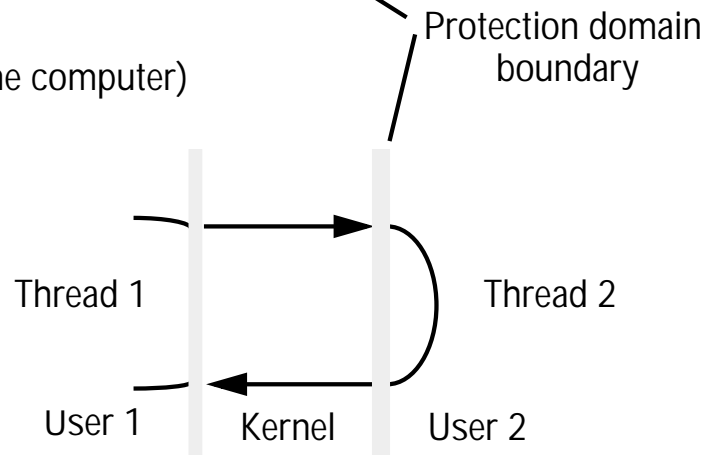


Figure 6.11 (cont'd) Invocations between address spaces

(c) RPC/RMI (between computers)

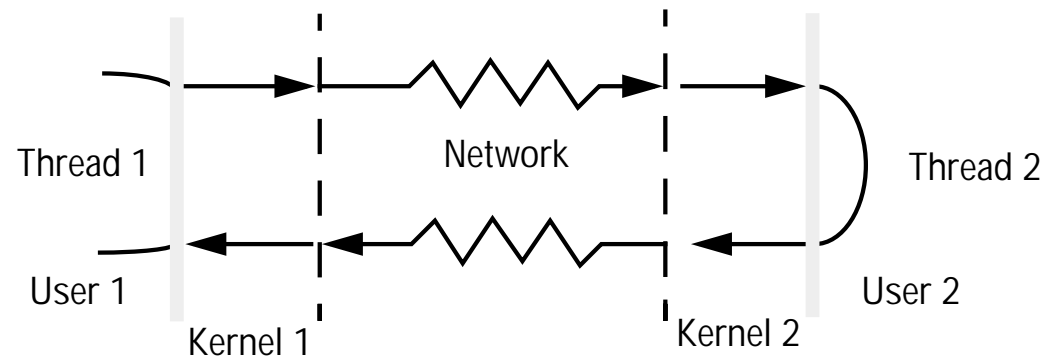


Figure 6.12 RPC delay against parameter size

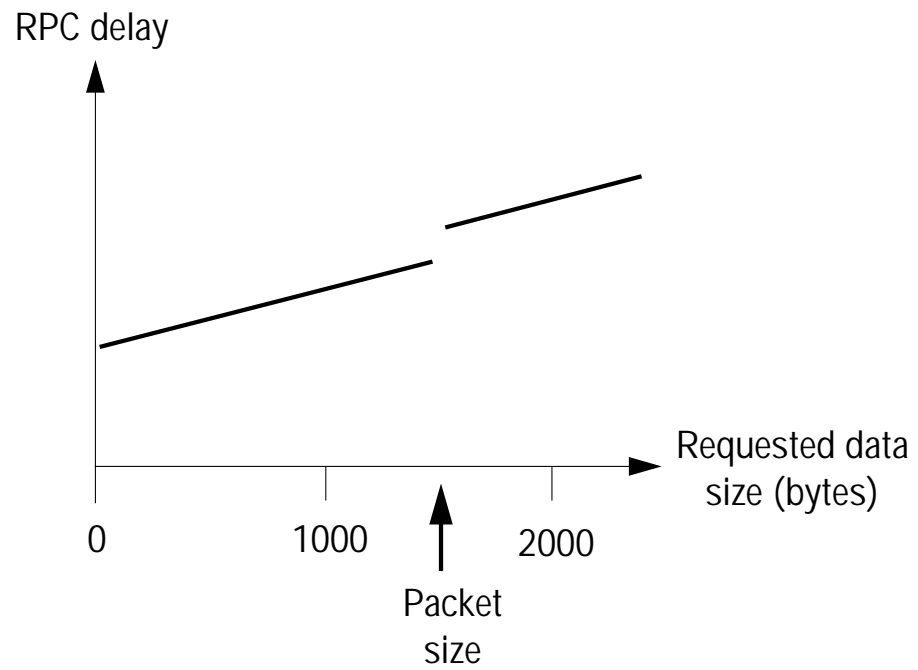


Figure 6.13 A lightweight remote procedure call

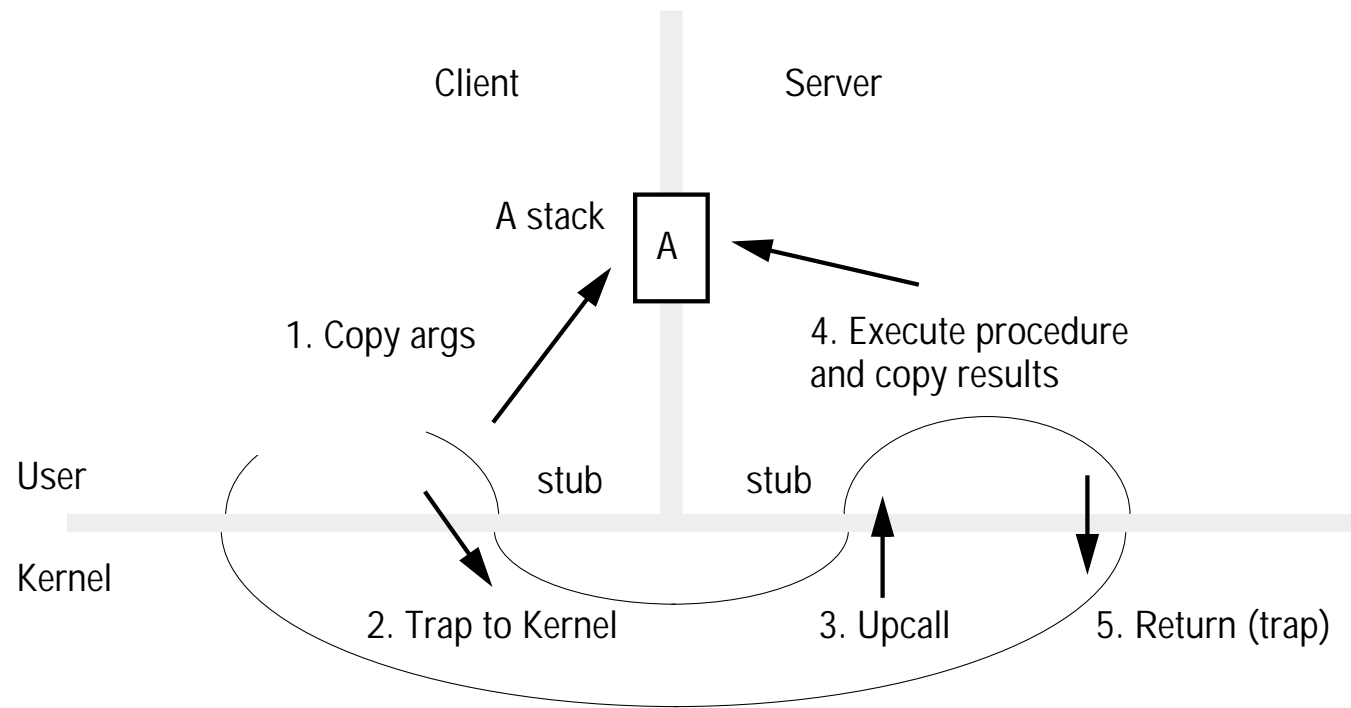


Figure 6.14 Times for serialised and concurrent invocations

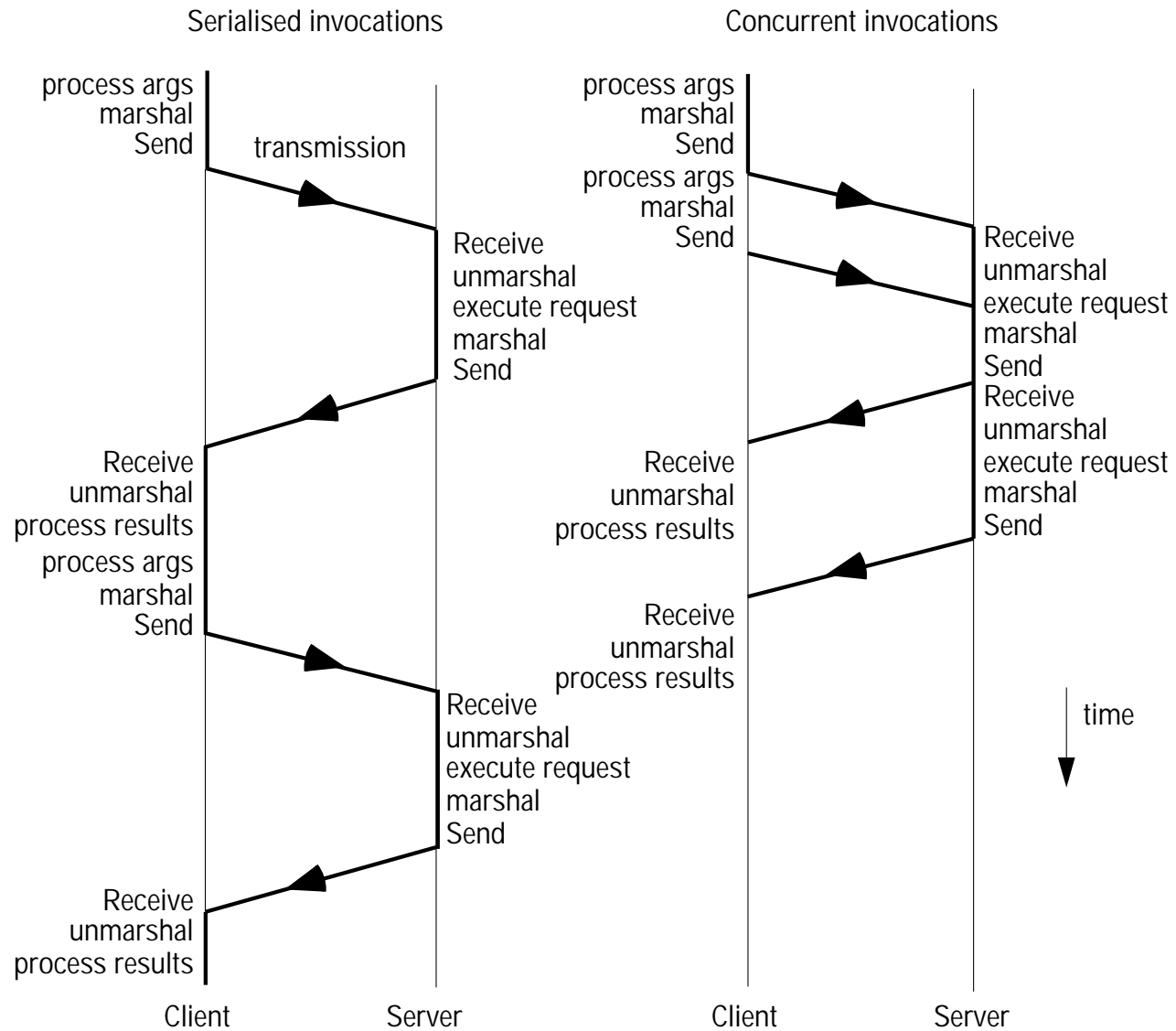


Figure 6.15 Monolithic kernel and microkernel

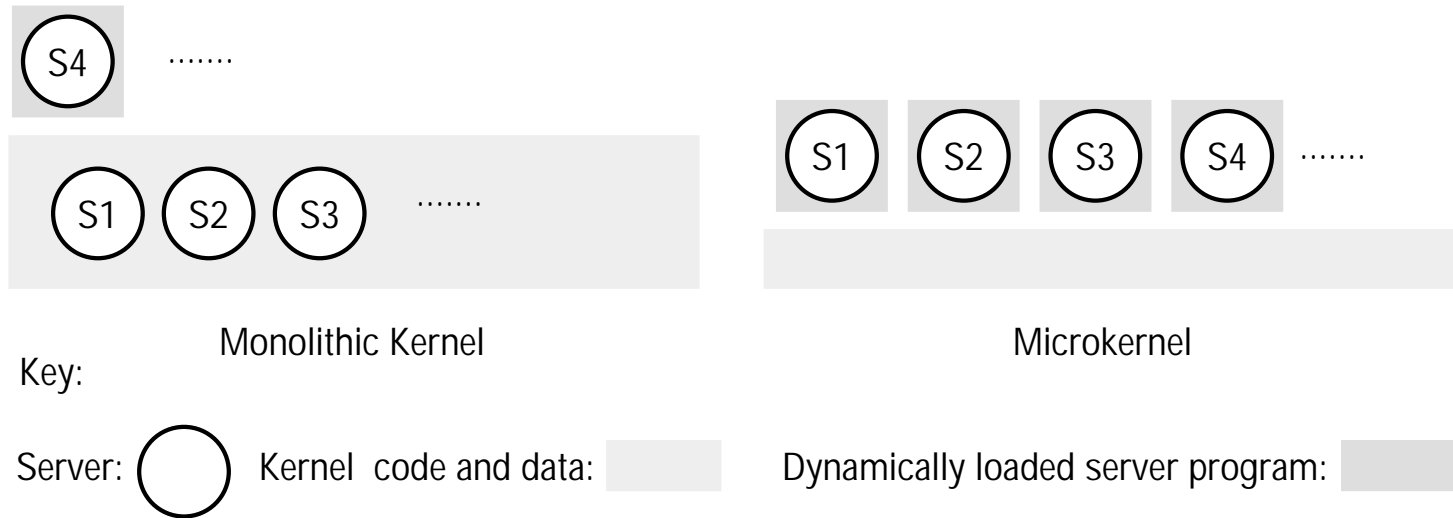
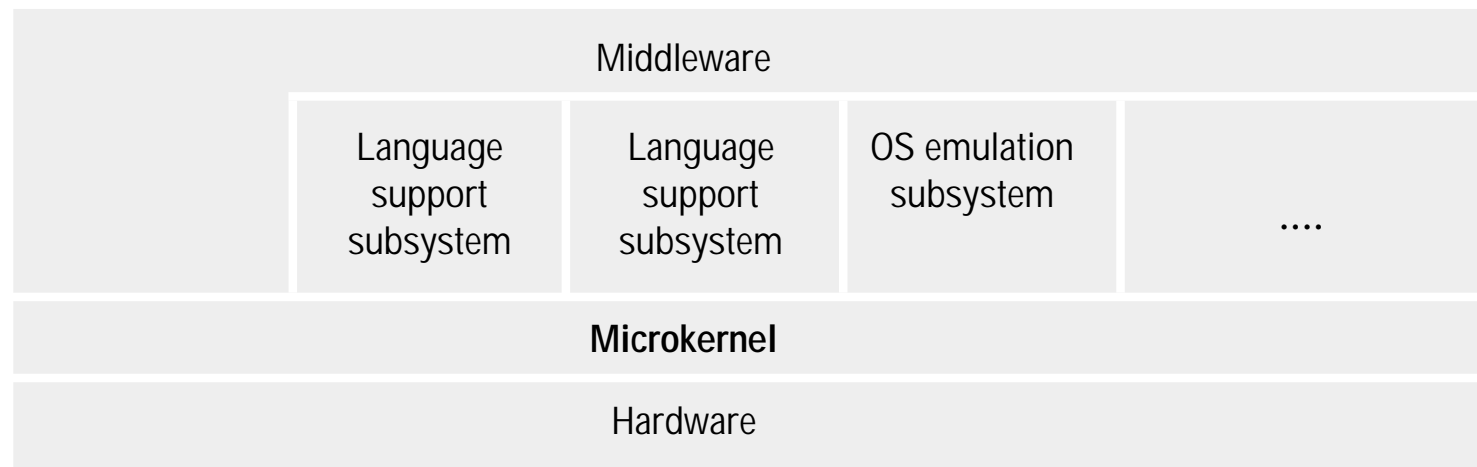


Figure 6.16 The role of the microkernel



The microkernel supports middleware via subsystems