Scheduler Activations

THE UNIVERSITY OF
NEW SOUTH WALES

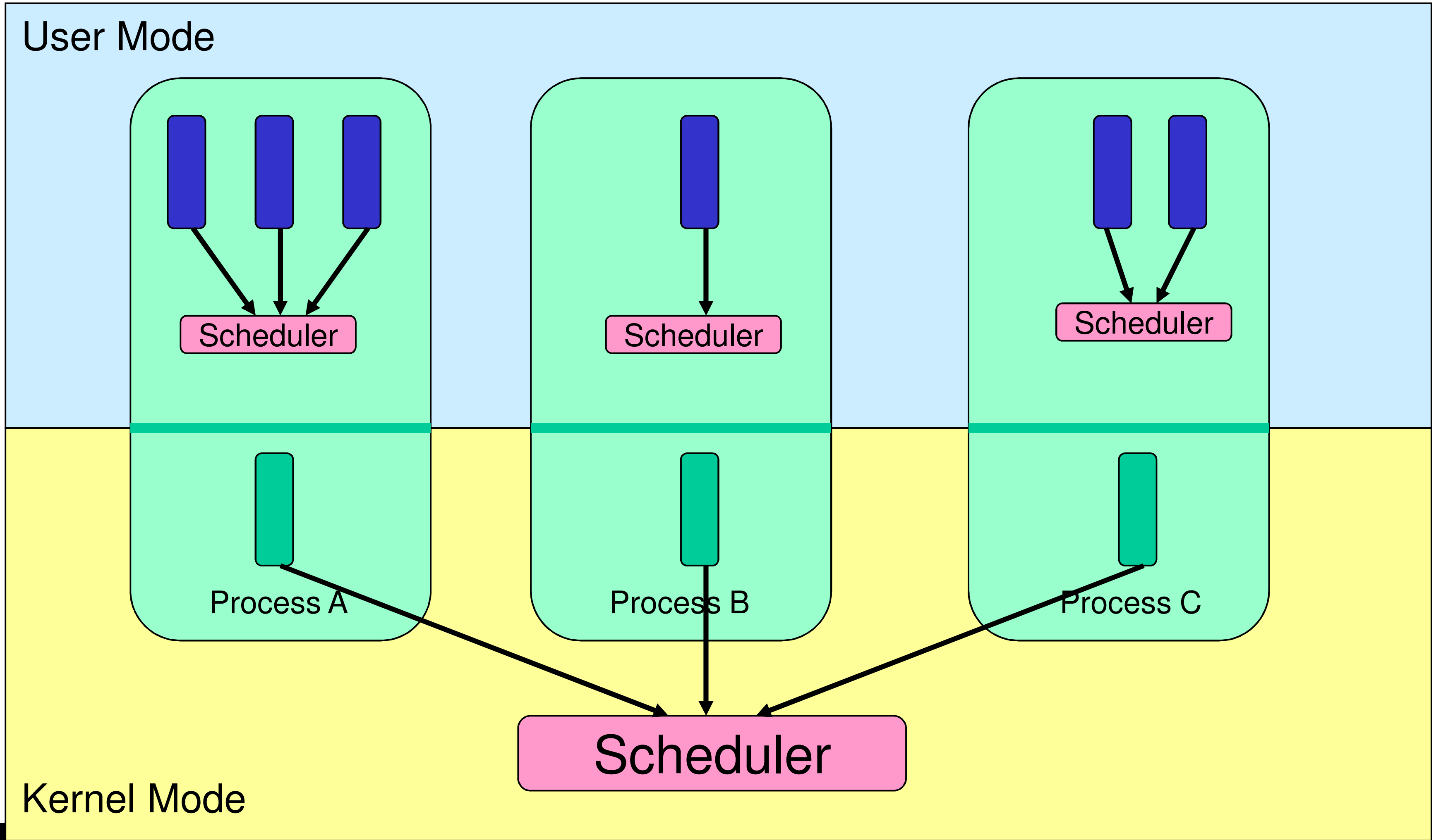
Slides modified from Raymond Namyst, U. Bordeaux cse

Learning Outcomes

- An understanding of hybrid approaches to thread implementation
- A high-level understanding of scheduler activations, and how they overcome the limitations of user-level and kernel-level threads.

- Thomas Anderson, Brian Bershad, Edward Lazowska, and Henry Levy. Scheduler Activations: Effective Kernel Support for the User-Level management of Parallelism. ACM Trans. on Computer Systems 10(1), Feburary 1992, pp. 53-79.

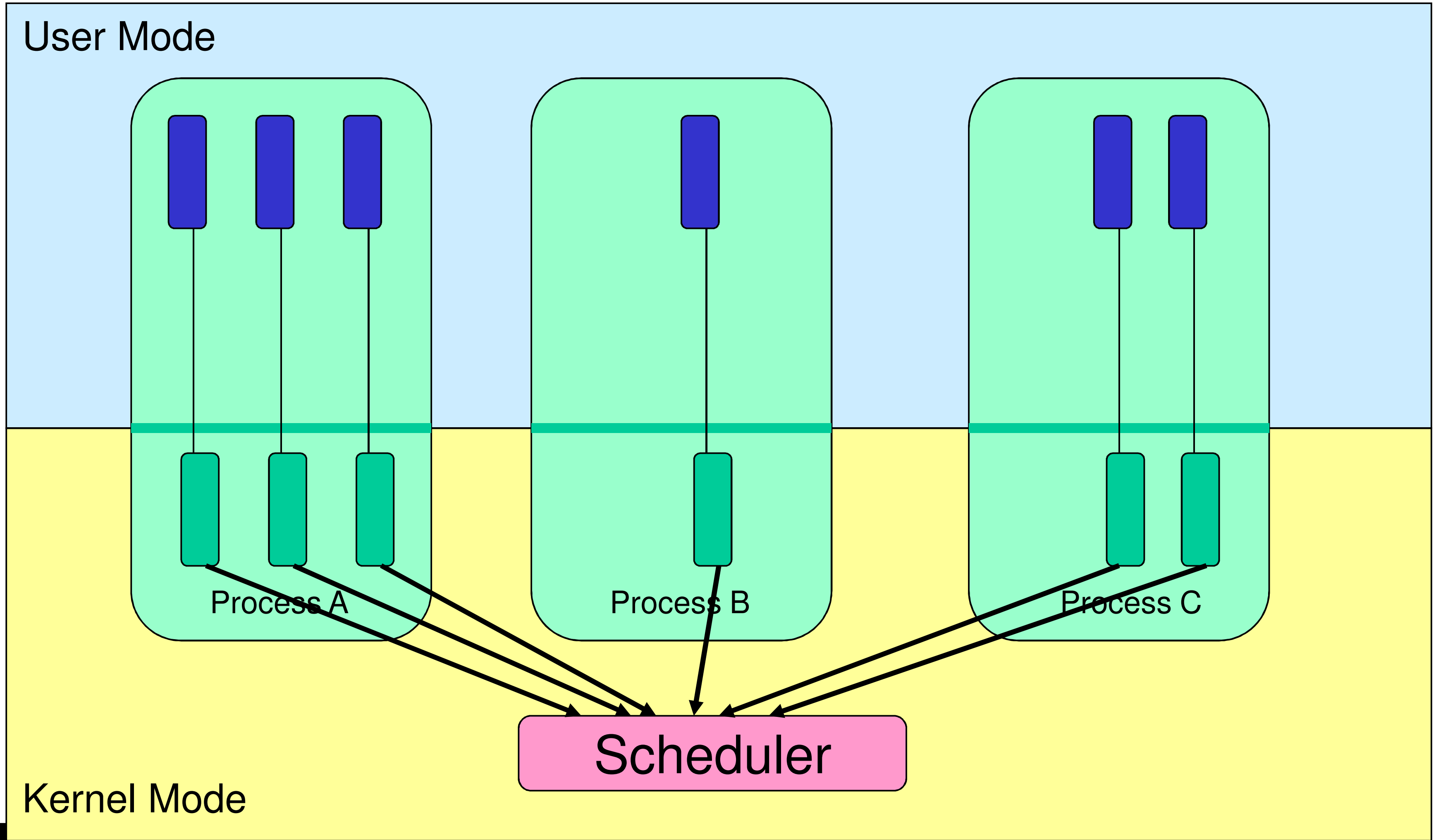
User-level Threads



User-level Threads

- ✓ Fast thread management (creation, deletion, switching, synchronisation…)
- ✘ Blocking blocks all threads in a process
  - Syscalls
  - Page faults
- ✘ No thread-level parallelism on multiprocessor

Kernel-Level Threads



Kernel-level Threads

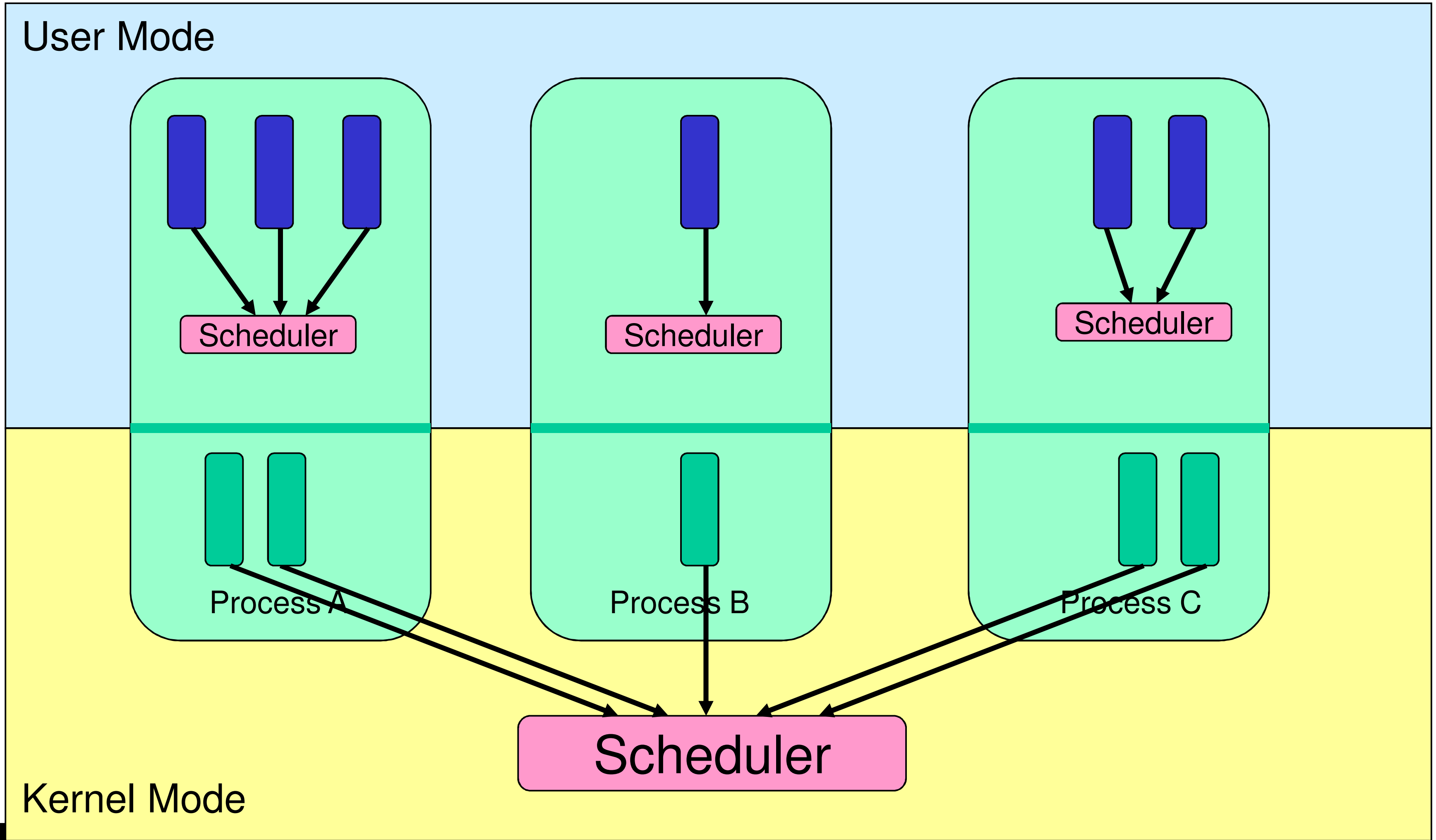
- ✘ Slow thread management (creation, deletion, switching, synchronisation…)
  - System calls
- ✓ Blocking blocks only the appropriate thread in a process
- ✓ Thread-level parallelism on multiprocessor

Performance

Table I: Thread Operation Latencies (μsec.)

Operation	FastThreads	Topaz threads	Ultrix processes
Null Fork	34	948	11300
Signal-Wait	37	441	1840

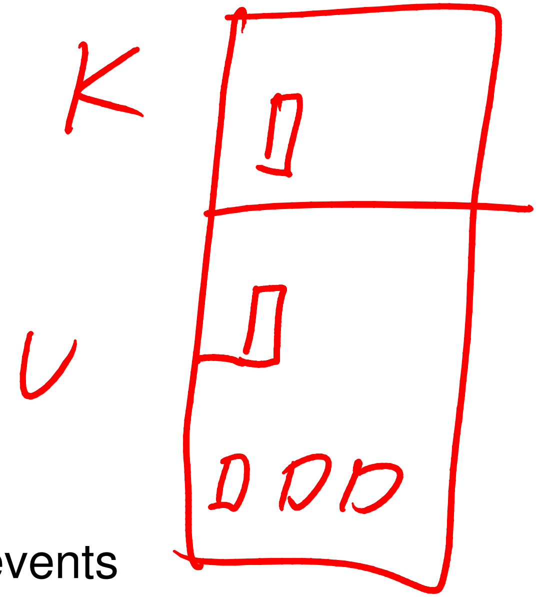
Hybrid Multithreading



Hybrid Multithreading

- ✓ Can get real thread parallelism on multiprocessor
- ✘ Blocking still a problem!!!

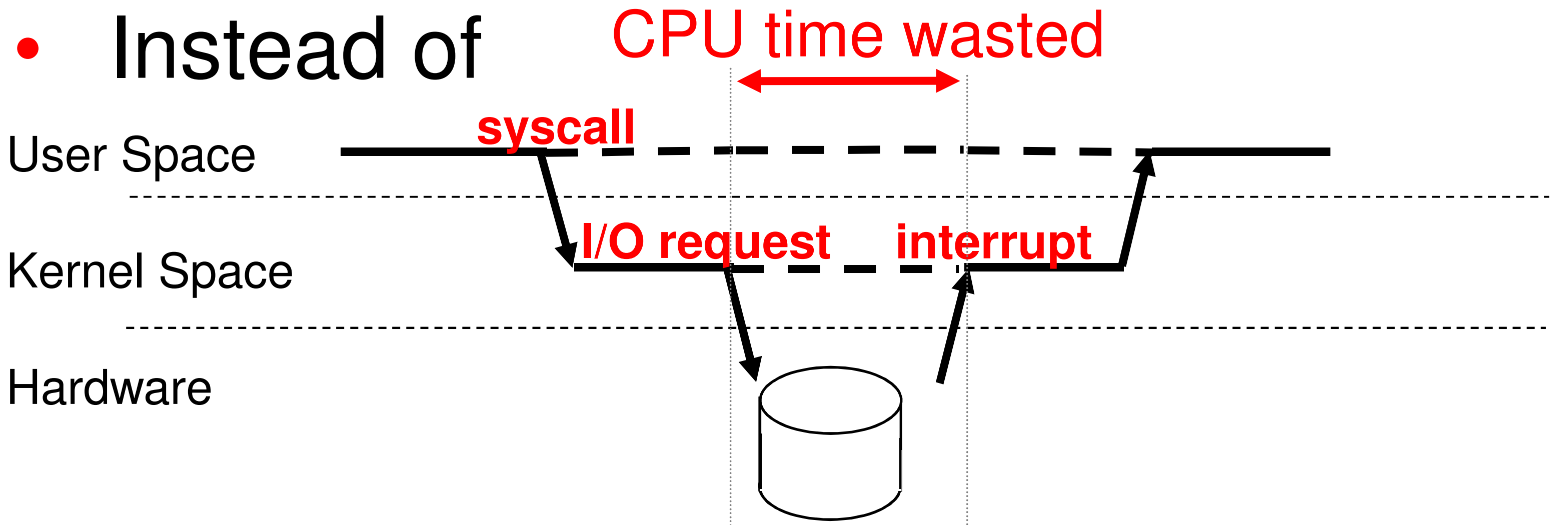
Scheduler Activations



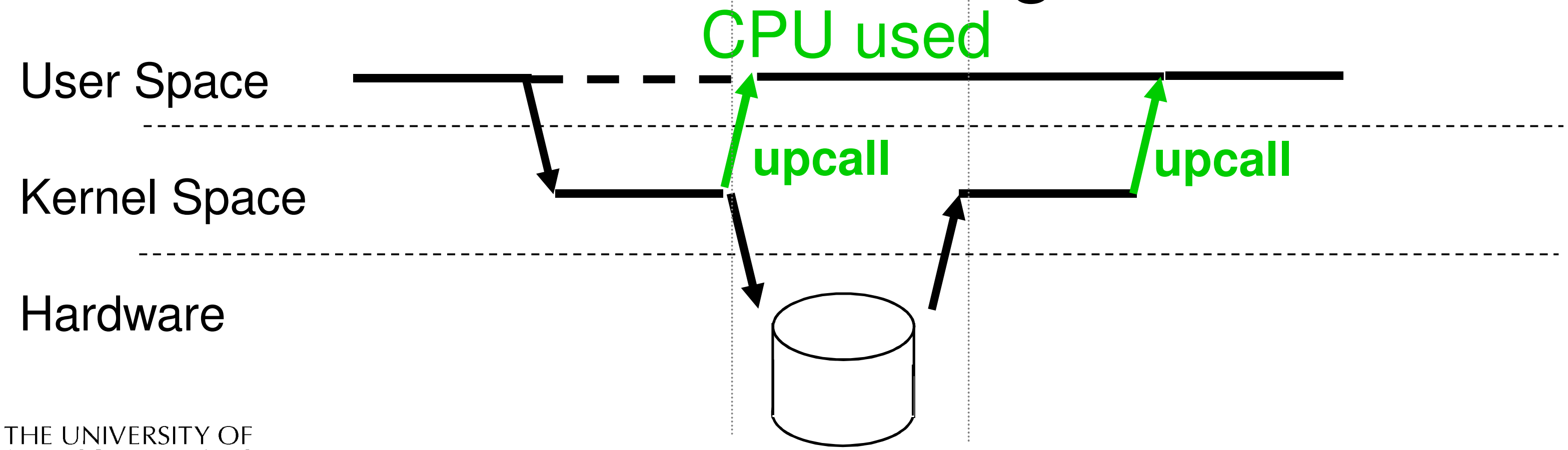
- First proposed by [Anderson et al. 91]
- Idea: Both schedulers co-operate
    - User scheduler uses system calls
    - Kernel scheduler uses upcalls!
- Two important concepts
    - Upcalls
        - Notify the user-level of kernel scheduling events
    - Activations
        - A new structure to support upcalls and execution
            - approximately a kernel thread
        - As many running activations as (allocated) processors
        - Kernel controls activation creation and destruction

Scheduler Activations

- Instead of



- …rather use the following scheme:

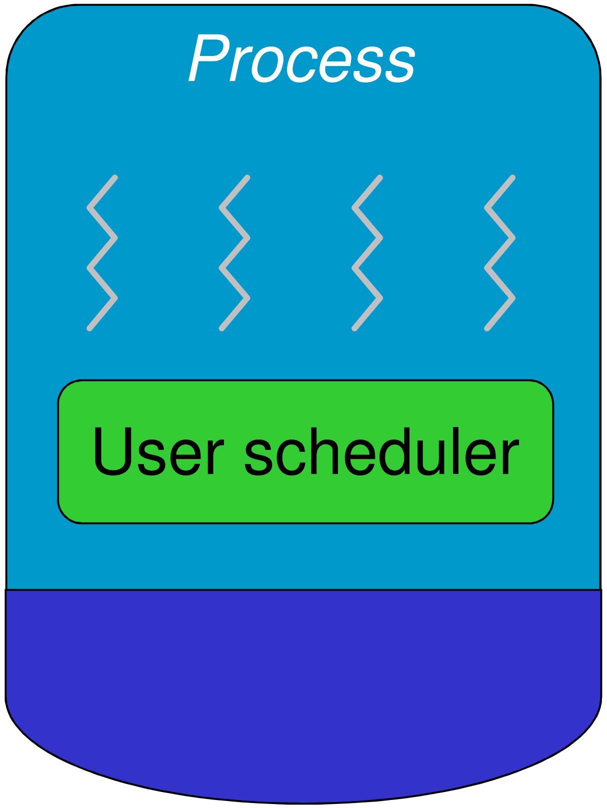


Upcalls to User-level scheduler

- New
  - Allocated a new virtual CPU
  - Can schedule a user-level thread
- Preempted
  - Deallocated a virtual CPU
  - Can schedule one less thread
- Blocked
  - Notifies thread has blocked
  - Can schedule another user-level thread
- Unblocked
  - Notifies a thread has become runnable
  - Must decided to continue current or unblocked thread

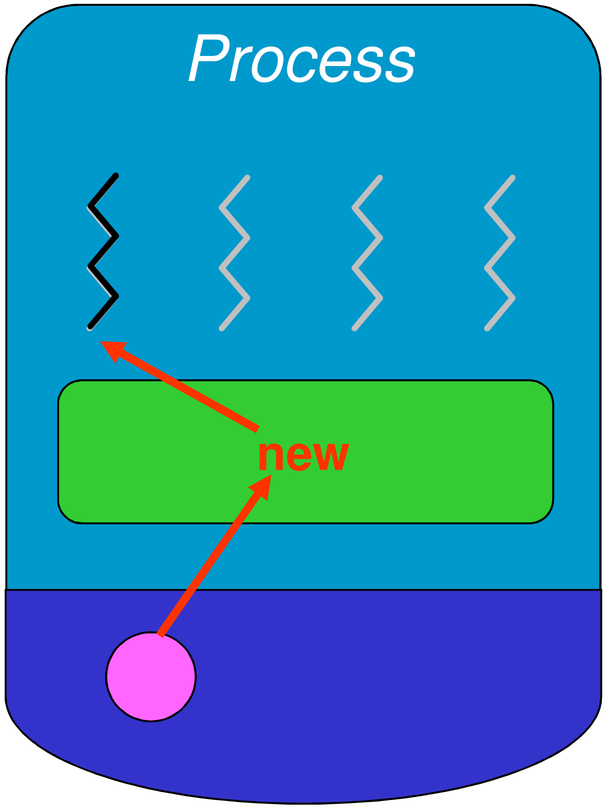
Working principle

- Blocking syscall scenario on 2 processors



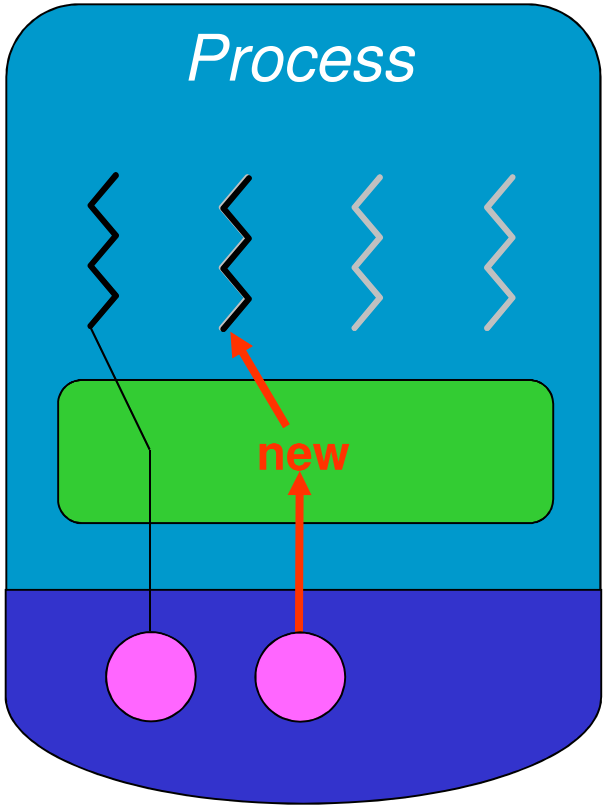
Working principle

- Blocking syscall scenario on 2 processors



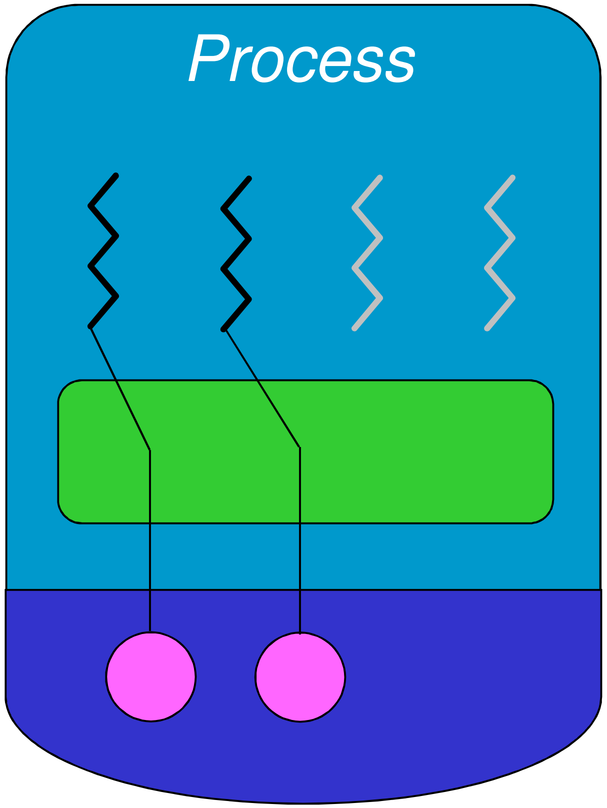
Working principle

- Blocking syscall scenario on 2 processors



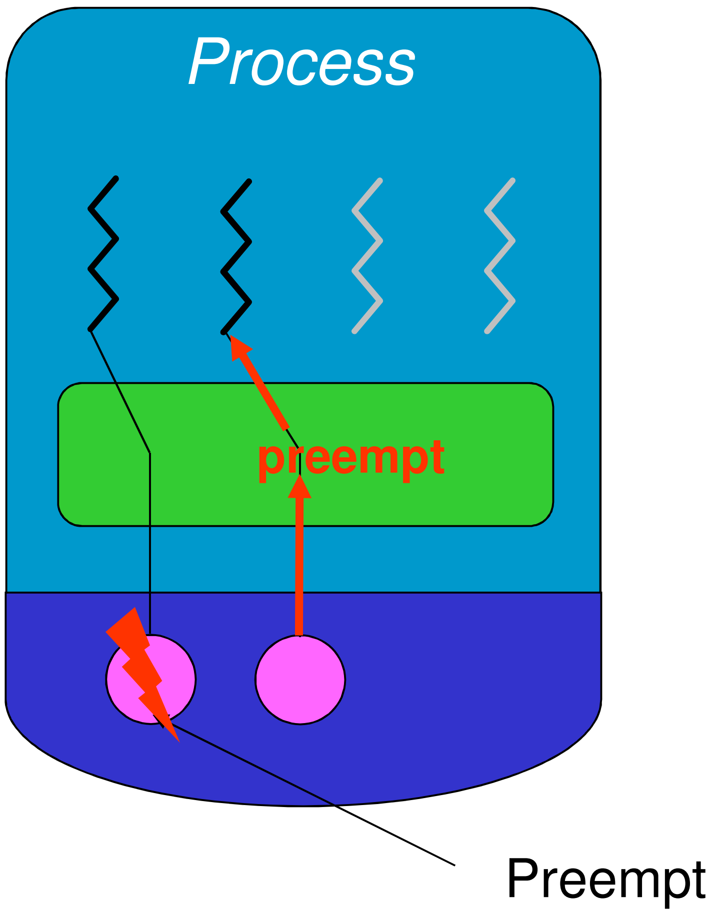
Working principle

- Blocking syscall scenario on 2 processors



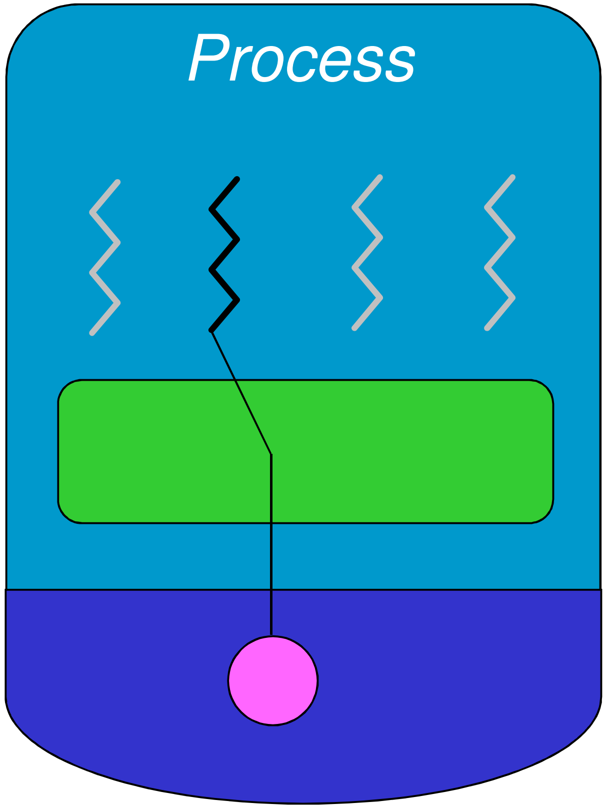
Working principle

- Blocking syscall scenario on 2 processors



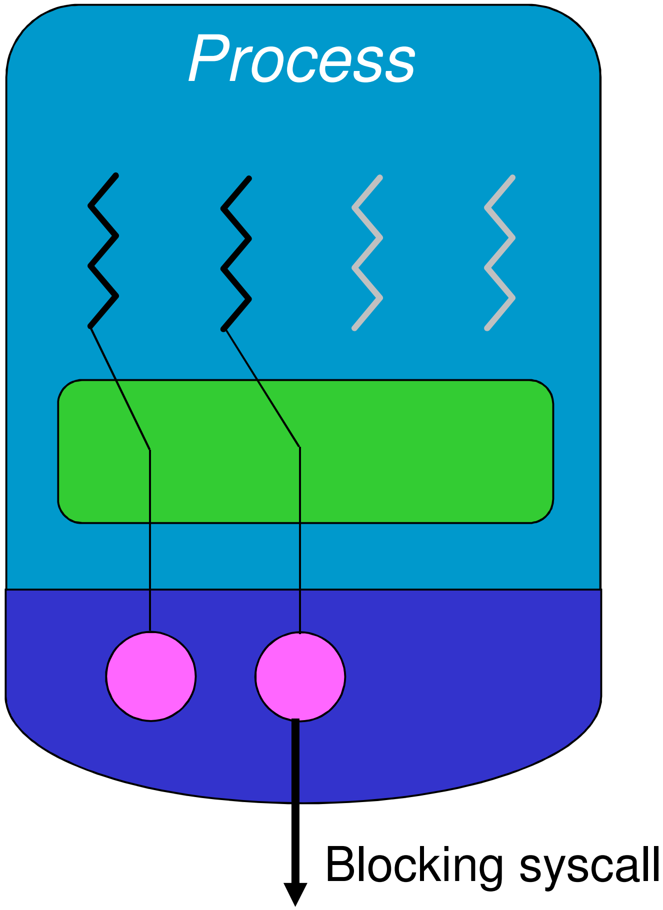
Working principle

- Blocking syscall scenario on 2 processors



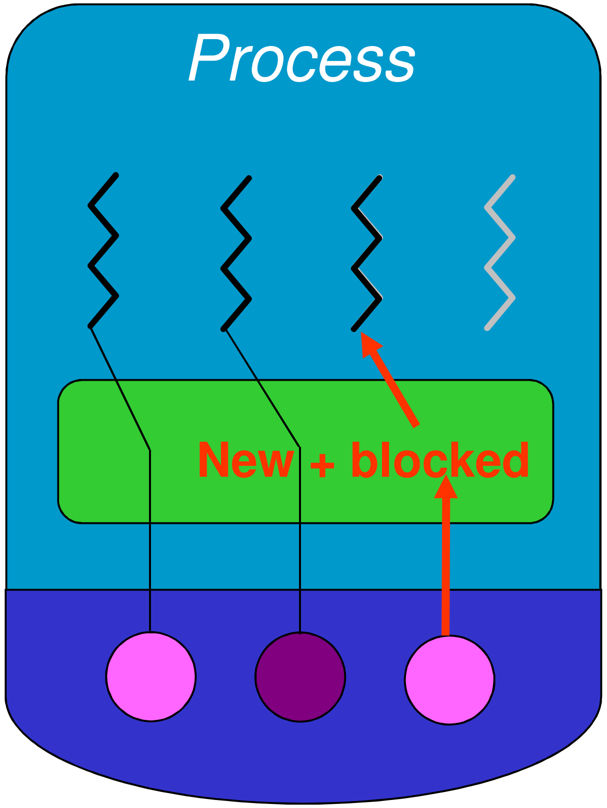
Working principle

- Blocking syscall scenario on 2 processors



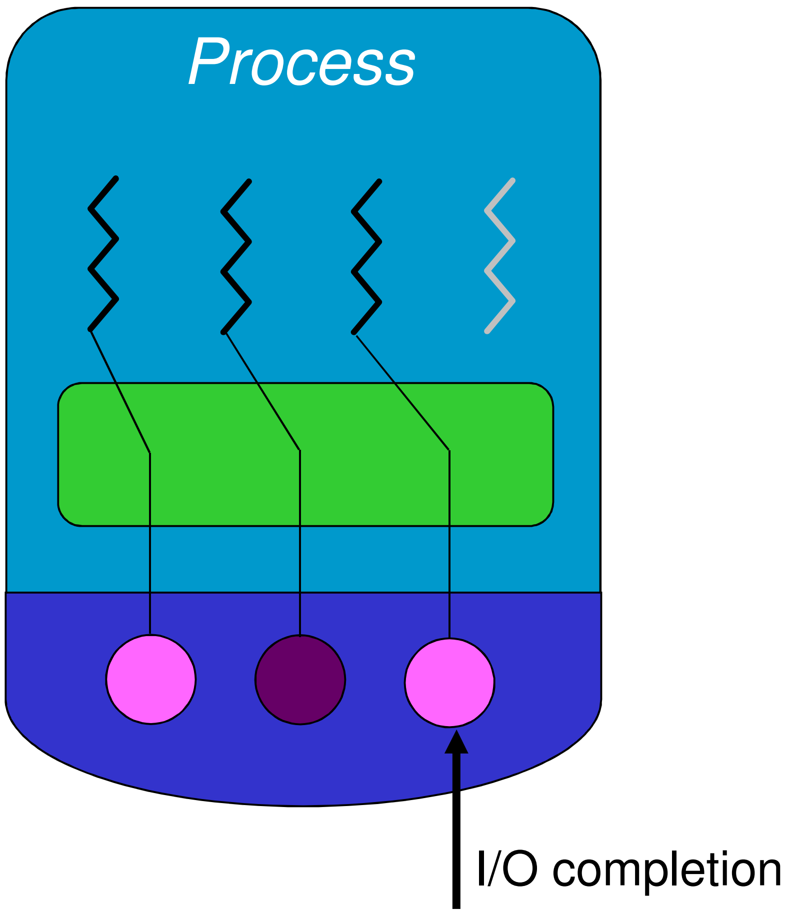
Working principle

- Blocking syscall scenario on 2 processors



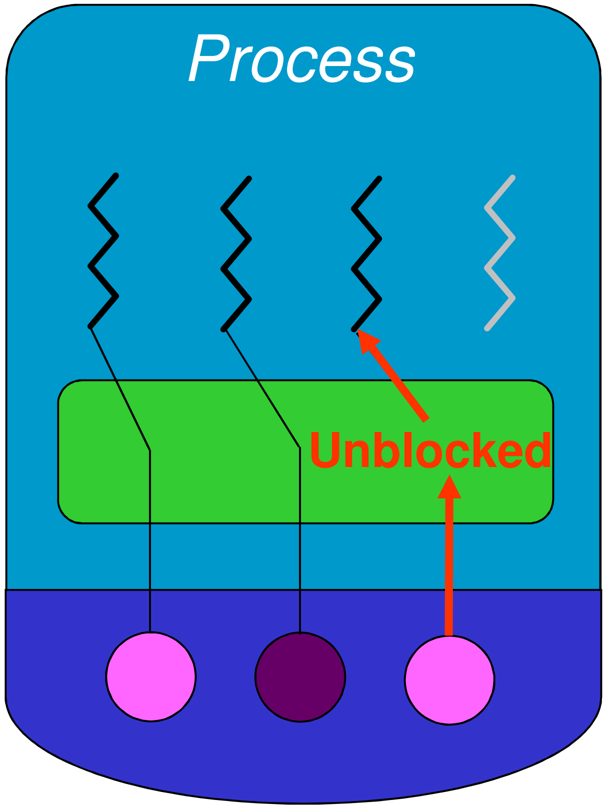
Working principle

- Blocking syscall scenario on 2 processors



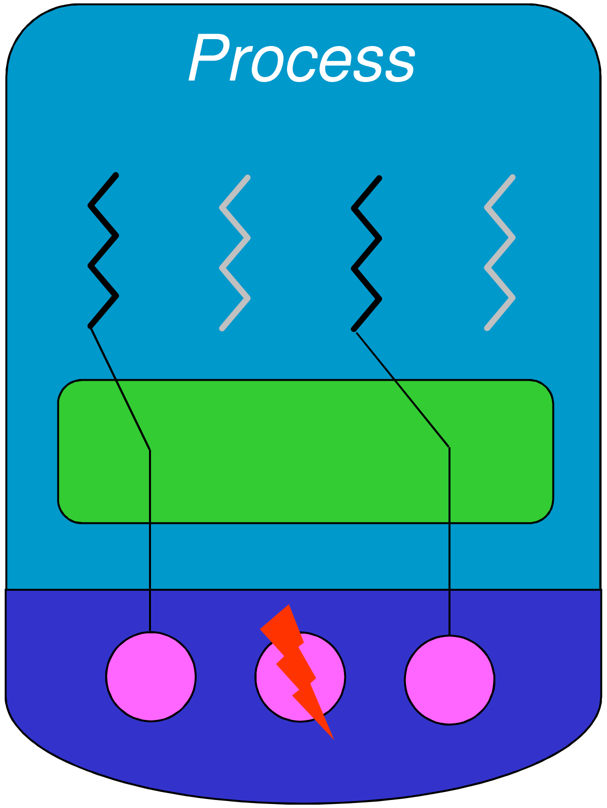
Working principle

- Blocking syscall scenario on 2 processors



Working principle

- Blocking syscall scenario on 2 processors



Scheduler Activations

- Thread management at user-level
  - Fast
- Real thread parallelism via activations
  - Number of activations (virtual CPU) can equal CPUs
- Blocking (syscall or page fault) creates new activation
  - User-level scheduler can pick new runnable thread.
- Fewer stacks in kernel
  - Blocked activations + number of virtual CPUs

Performance

Table IV. Thread Operation Latencies (μsec.)

Operation	FastThreads on Topaz Threads	FastThreads on Scheduler Activations	Topaz threads	Ultrix processes
Null Fork	34	37	948	11300
Signal-Wait	37	42	441	1840

Adoption

- Adopters
  - BSD “Kernel Scheduled Entities”
    - Reverted back to kernel threads
  - Variants in Research OSs: K42, Barrelfish
  - Digital UNIX
  - Solaris
  - Mach
  - Windows 7
- Linux -> kernel threads

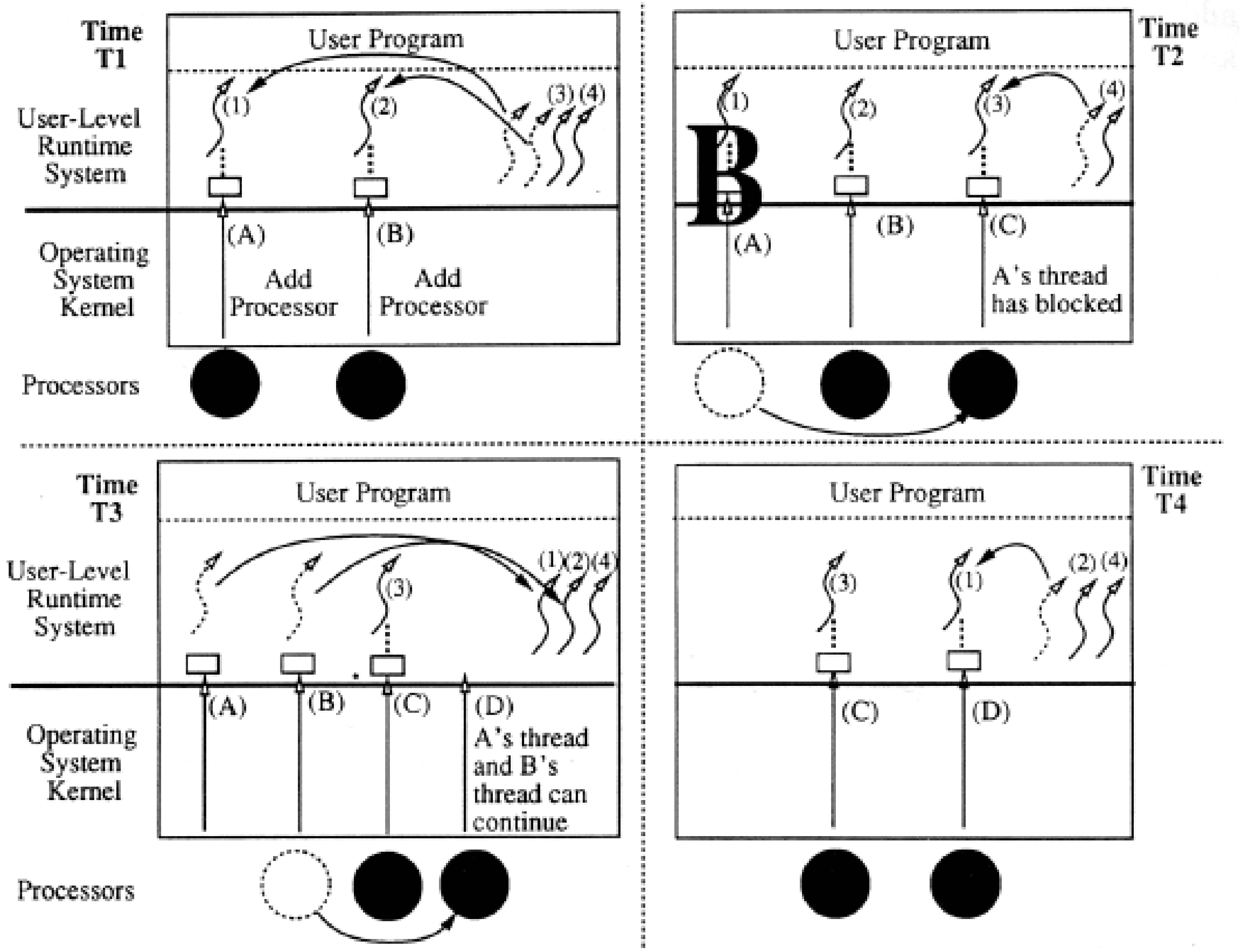


Fig. 1. Example: I/O request/completion.