SCHEDULER ACTIVATIONS:

Effective Kernel Support for User-Level Management of Parallelism

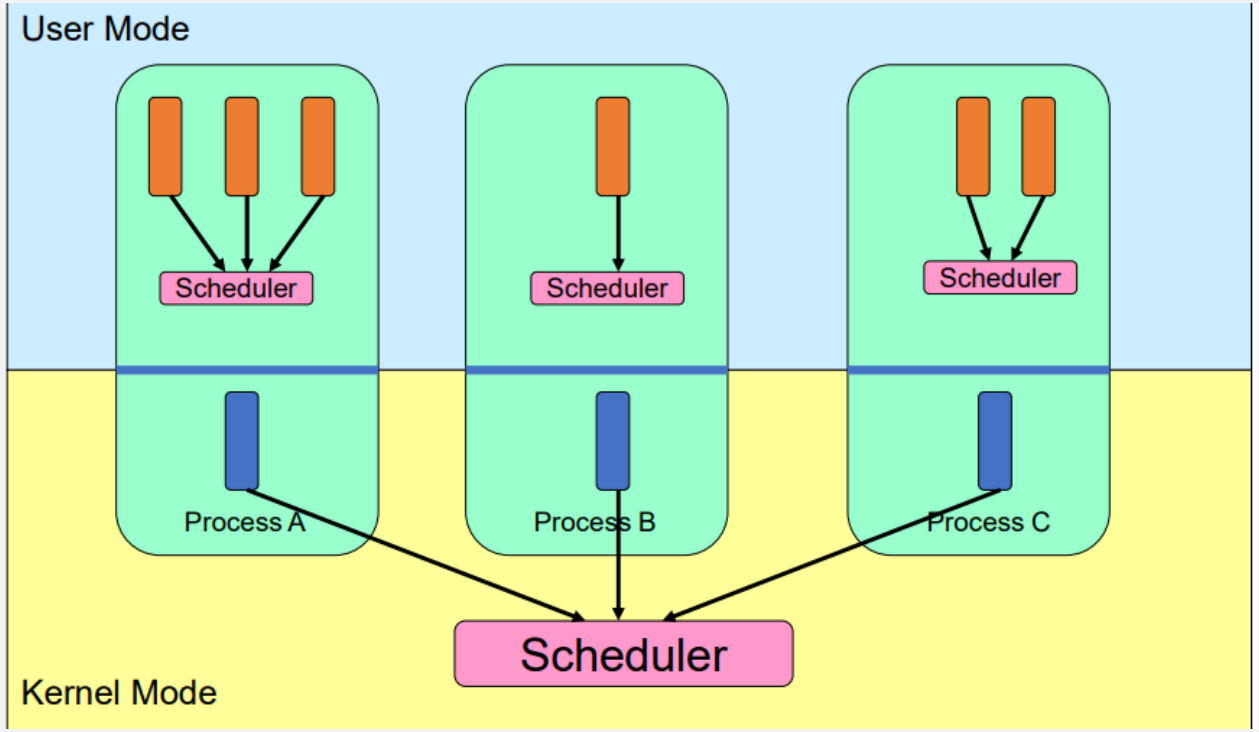
Nikolaos Emmanouil Tarasidis

What is a Thread?

- "A single flow of control within a process, with its own thread ID, scheduling priority and policy, errno value, floating-point environment, thread-specific key/value bindings, and the required system resources to support a flow of control." POSIX 3.109

User-Level Threads

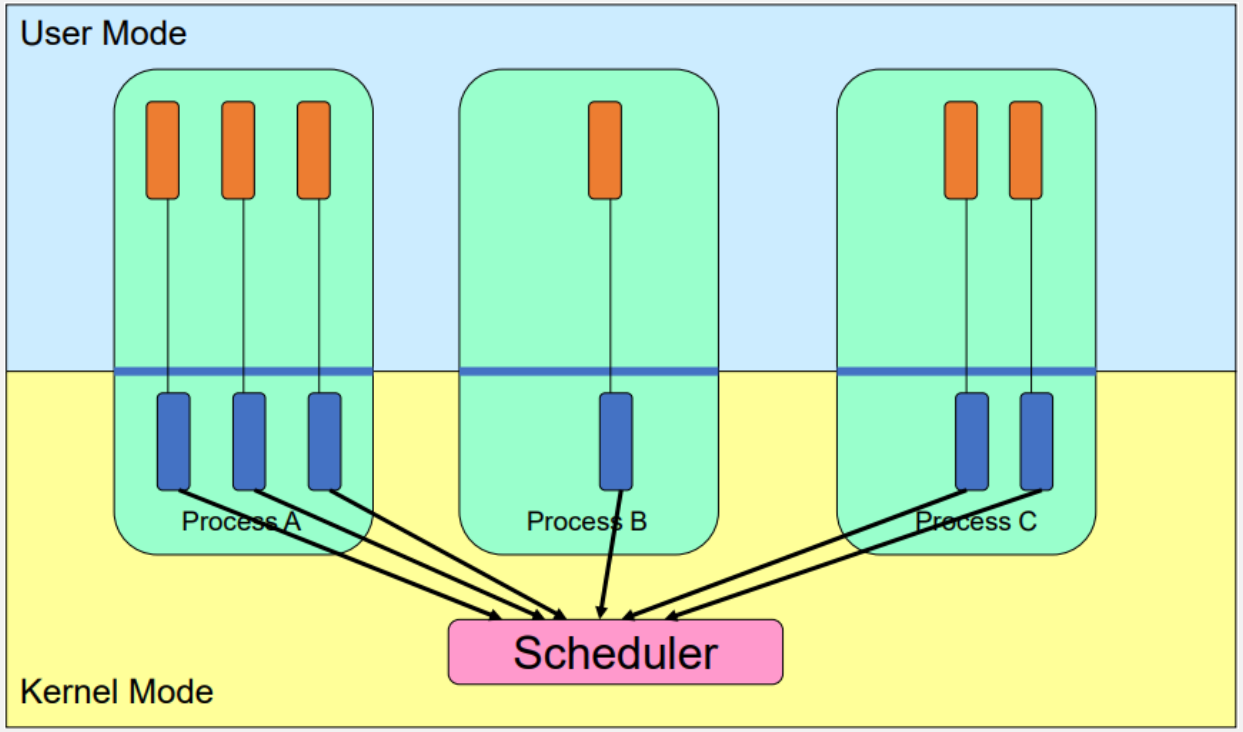
- Threads implemented in as a user-level library.
- Fast thread management
- Multiple threads mapped to the same schedulable kernel context



BACKGROUND

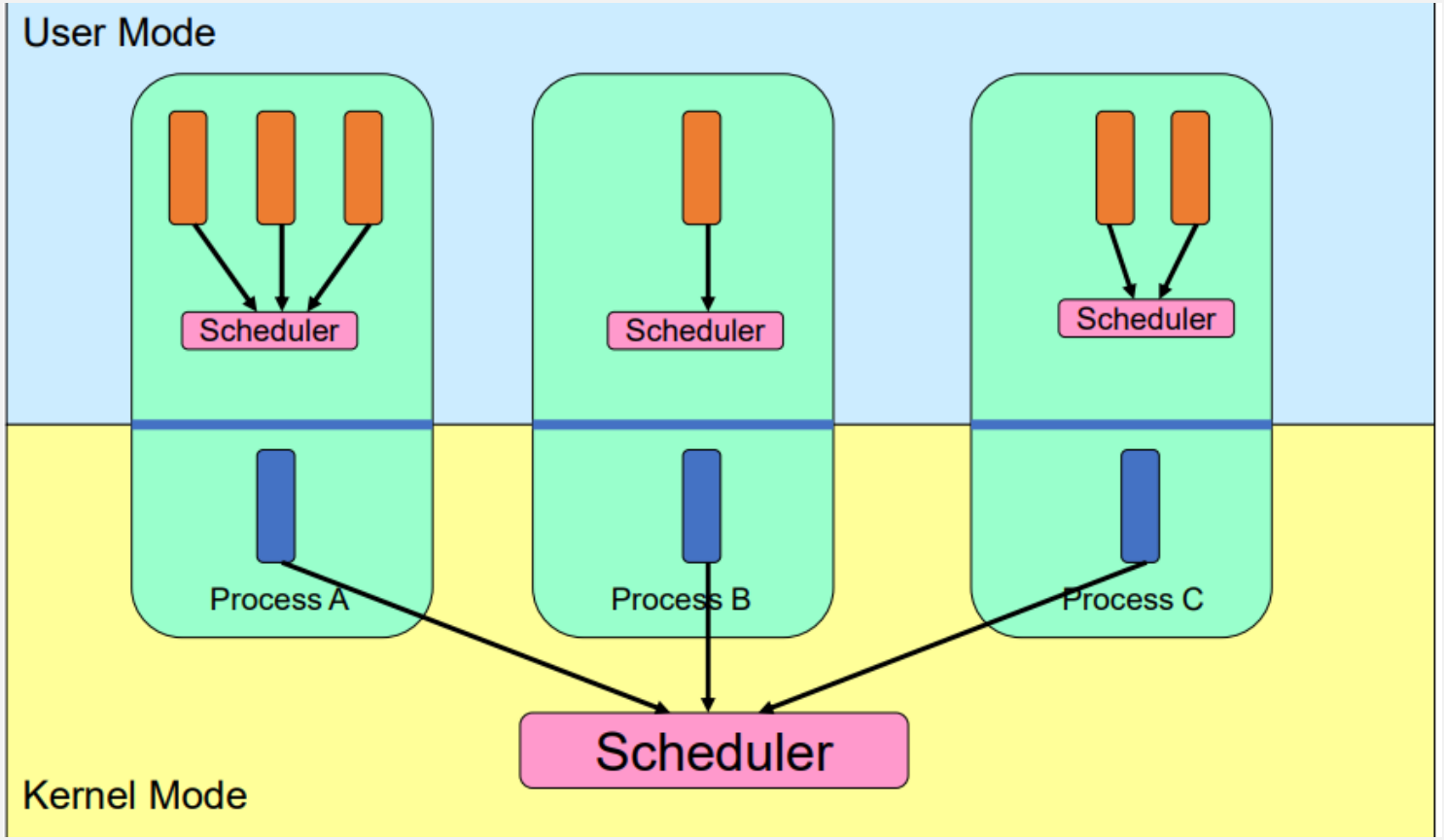
Kernel-Level Threads

- Threads implemented in the kernel (e.g. pthread)
- Slower to create and manage
- Each thread is mapped to one schedulable kernel context

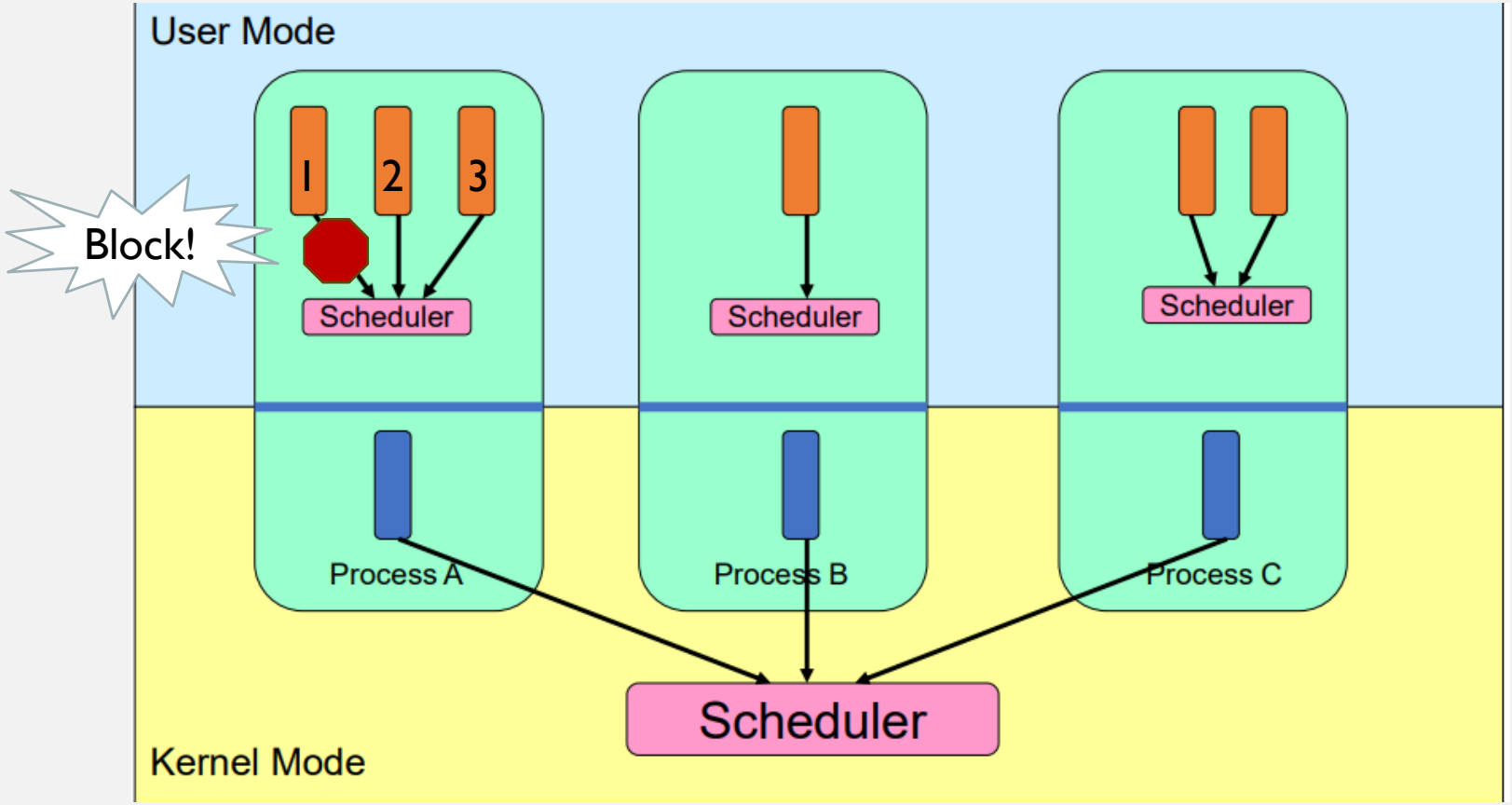


THE PROBLEM

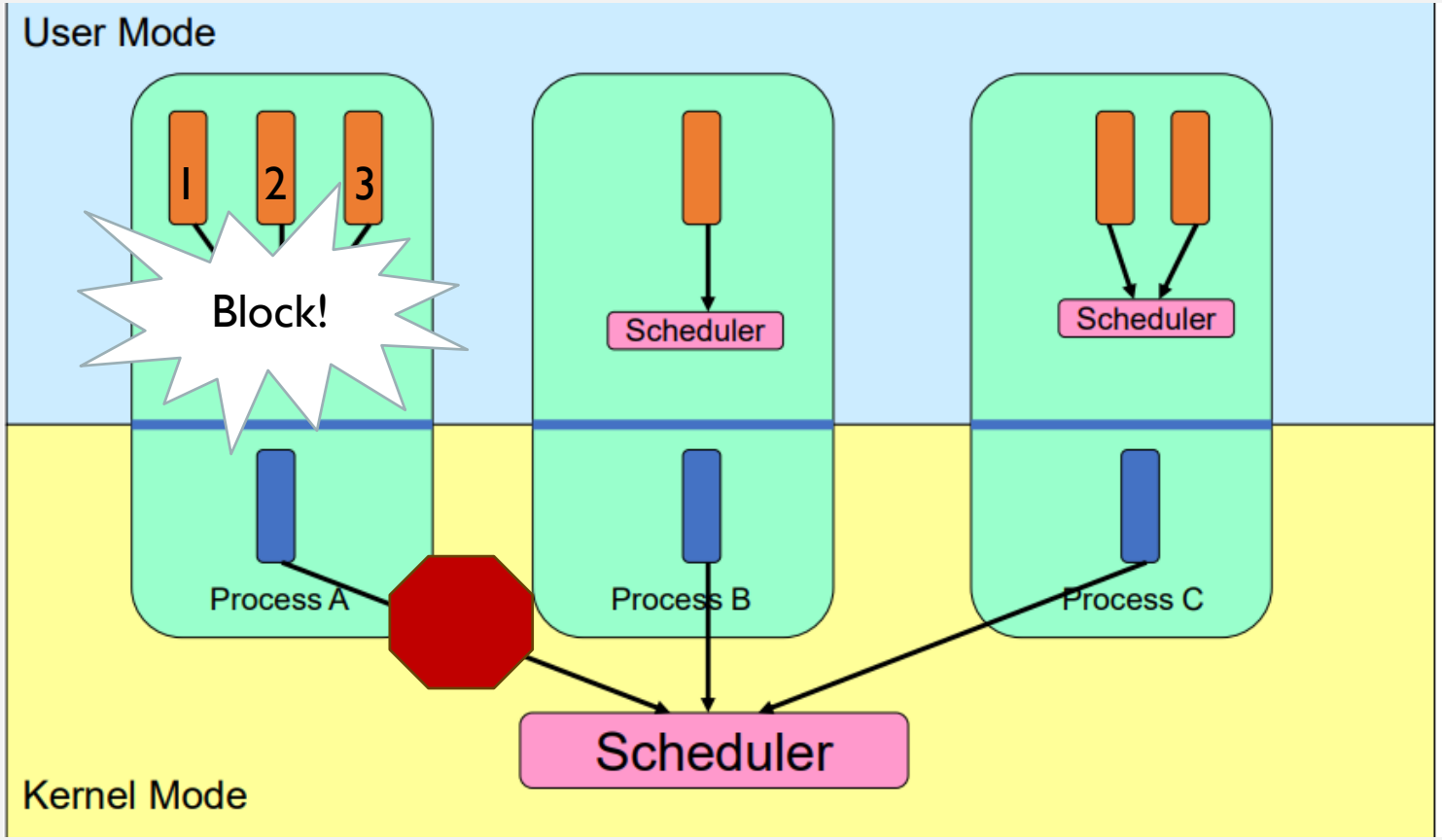
Start



THE PROBLEM

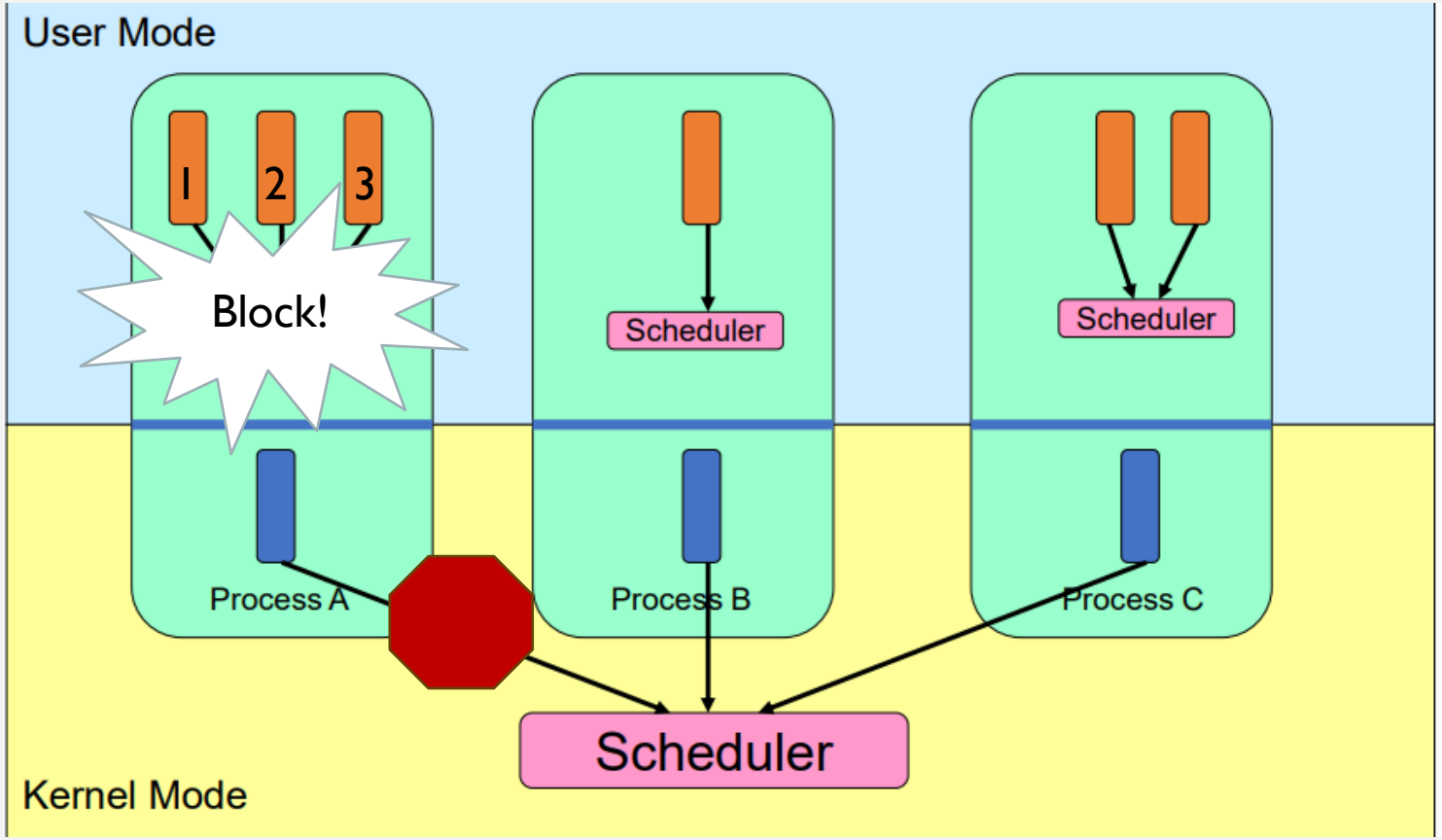

User Mode
1
2
3
Block!
Scheduler
Scheduler
Scheduler
Process A
Process B
Process C
Scheduler
Kernel Mode

THE PROBLEM


User Mode
1
2
3
Block!
Scheduler
Scheduler
Process A
Process B
Process C
Scheduler
Kernel Mode

THE PROBLEM

Threads 2 & 3 still have work to do.
Quite unfortunate!



SCHEDULER ACTIVATIONS

WHAT WE WANT

- Kernel allocates processors to address spaces.
- Kernel notifies the user-level thread system of relevant events.
- User-level thread system notifies the kernel when it needs more or fewer processors
- Fully transparent

SCHEDULER ACTIVATIONS

- A vessel or execution context for running user-level threads.
- Notifies the user-level thread system of kernel events.
- Provides space for saving thread context.

SIMPLE EXAMPLE

- Two processors allocated to the address space.

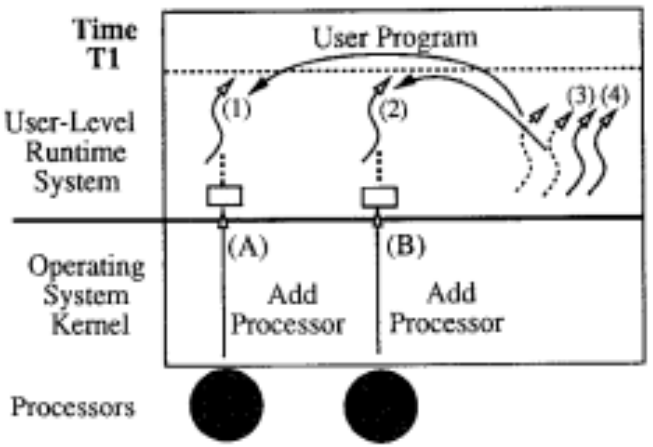


Fig. 1. Example: I/O request/completion.

SIMPLE EXAMPLE

- Thread I blocks!

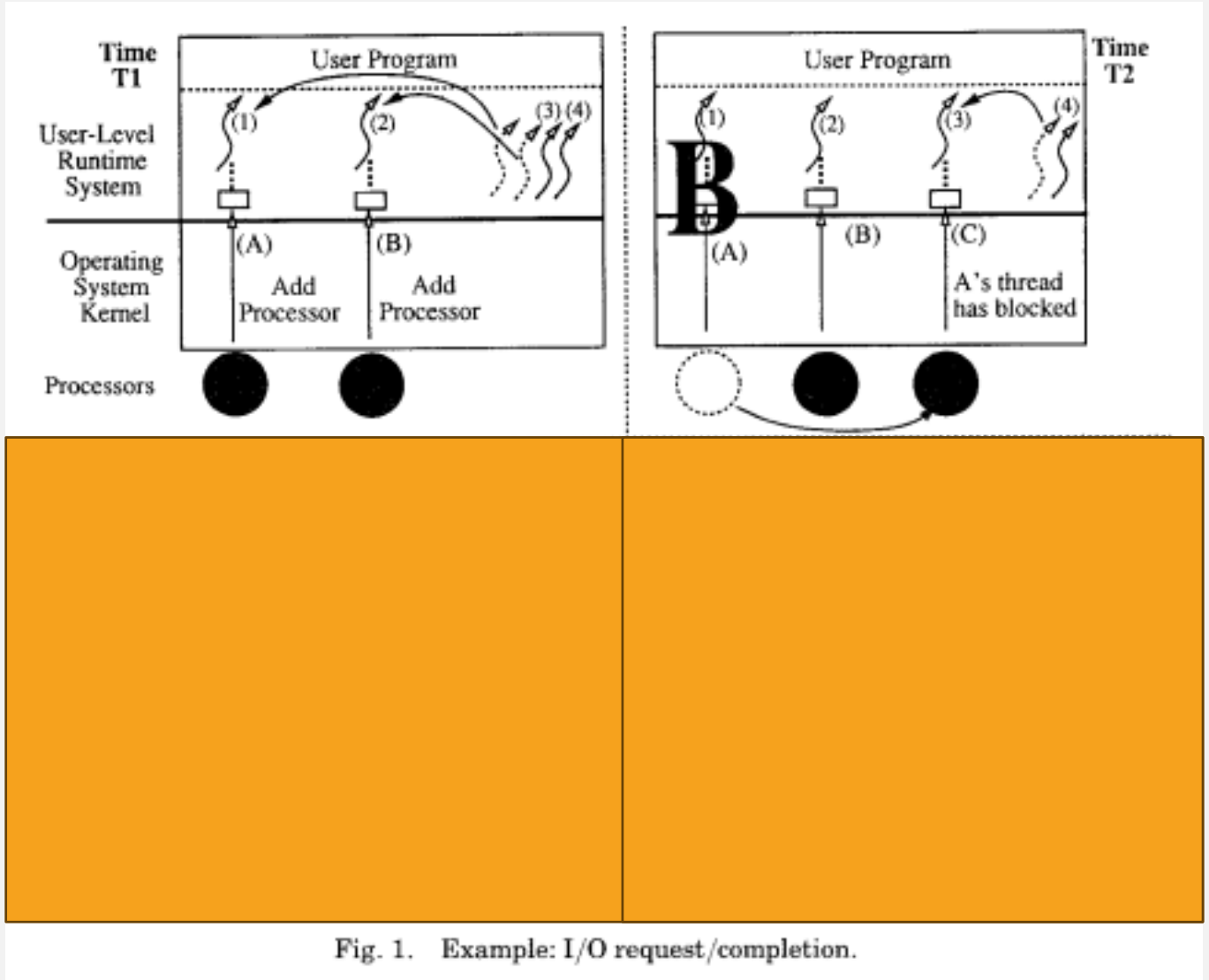


Fig. 1. Example: I/O request/completion.

SIMPLE EXAMPLE

- Thread I/O completes.

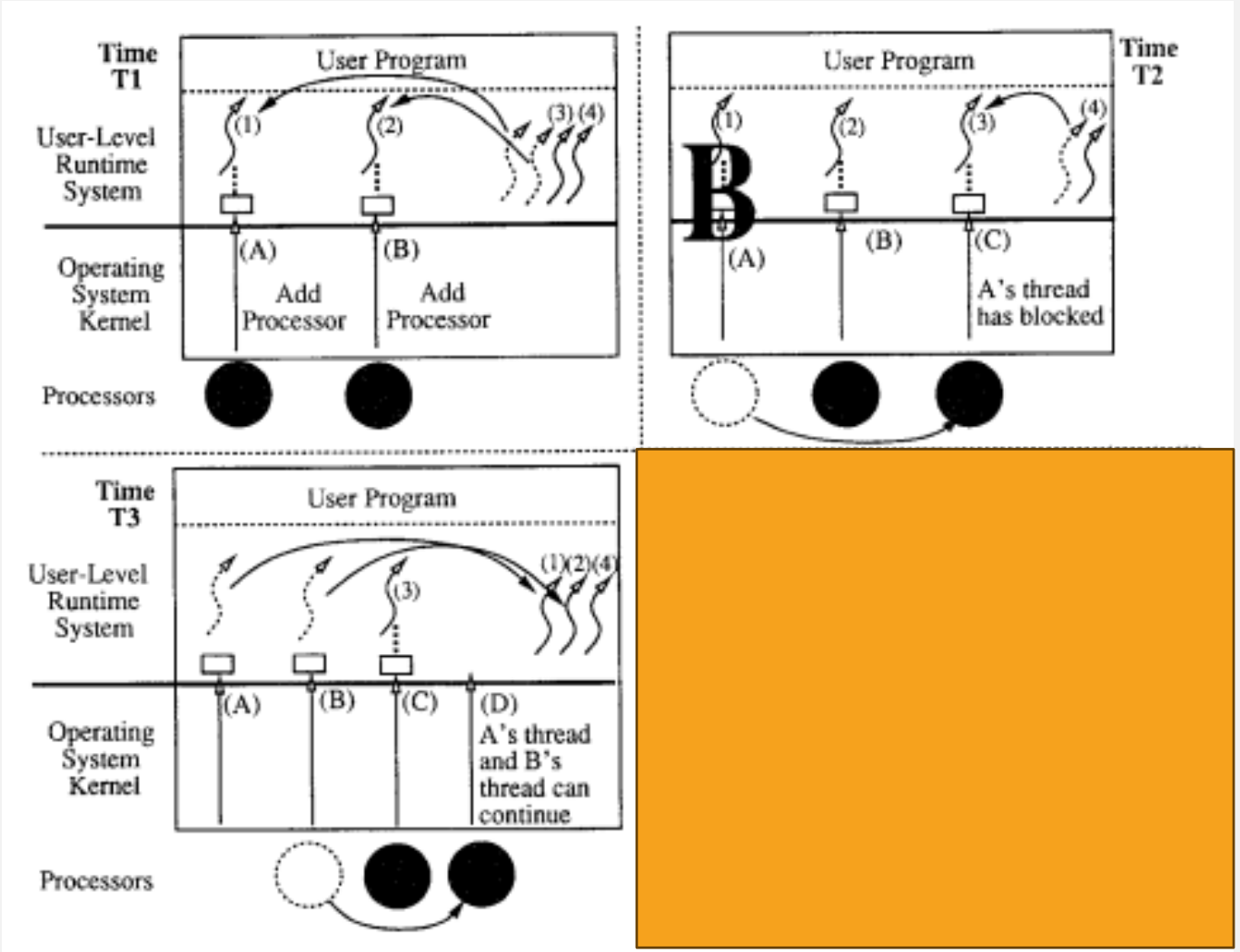


Fig. 1. Example: I/O request/completion.

SIMPLE EXAMPLE

- Execution resumes normally.

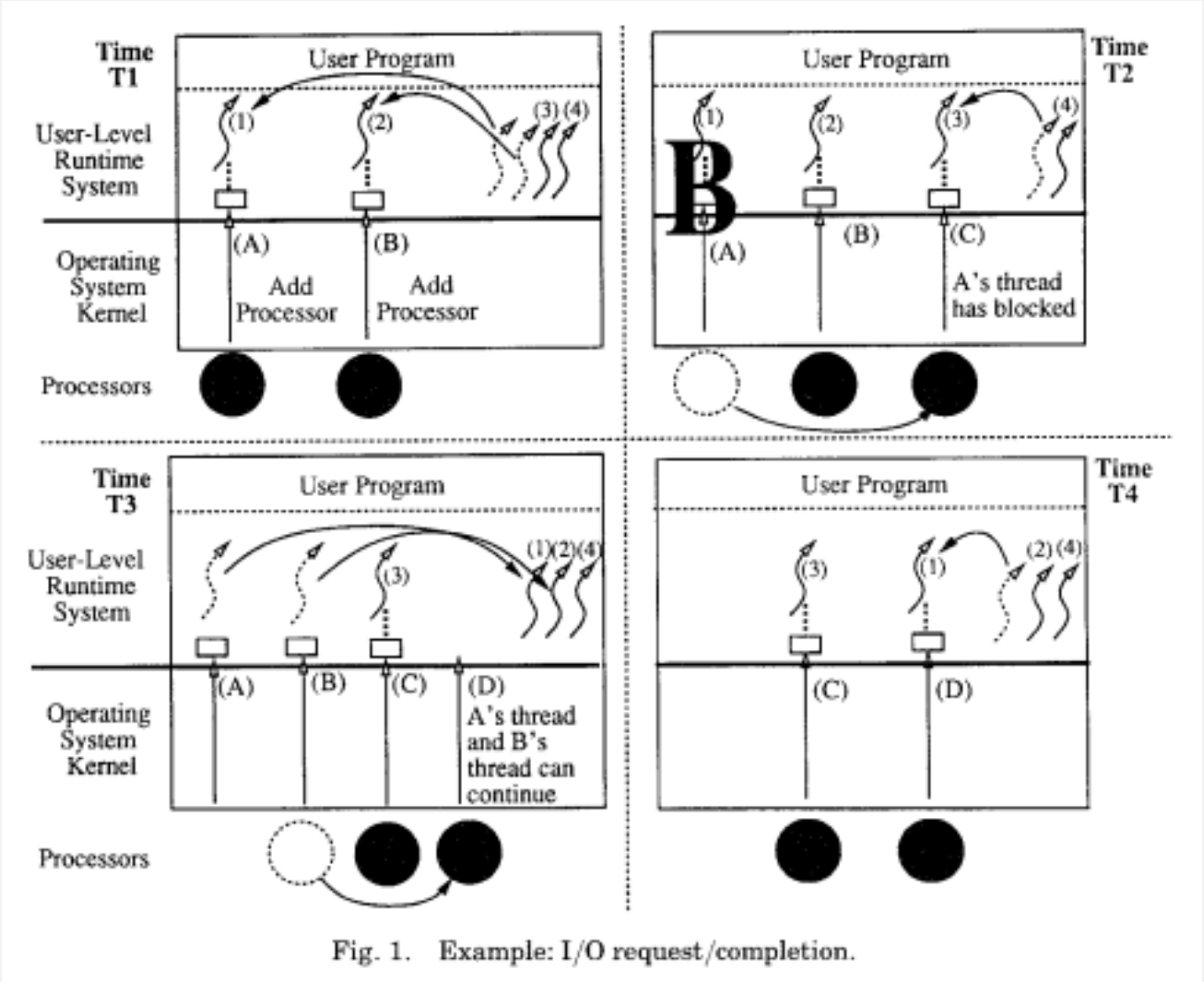


Fig. 1. Example: I/O request/completion.

UPCALL POINTS

- Add this processor (processor #)
- Processor has been preempted (preempted activation # and its machine state)
- Scheduler activation has blocked (blocked activation #)
- Scheduler activation has unblocked (unblocked activation # and its machine state)

DOWNCALL* POINTS

- Add more processors (additional # of processors needed)
- This processor is idle ()

CRITICAL SECTIONS

- Thread is preempted while executing in a critical section. Problem!
- Poor performance.
- Deadlock!

SOLUTION?

- Approach based on recovery.
- When preempted thread returns, check if it's running in a CS.
- If so continue until it exits CS.

EVALUATION

IMPLEMENTATION

- Modified Topaz native OS of Firefly multiprocessor workstation.
- Changed FastThreads user-level thread library.
- Processors are allocated fairly.
- Thread scheduling is completely up to the user level.

OPTIMIZATIONS

- Reusing discarded scheduler activations.
- Lock overhead is handled by copying CS code and make the copy yield the processor after the thread exits that section.

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

PERFORMANCE

- N-body problem in O(NlogN)
- Fixed memory

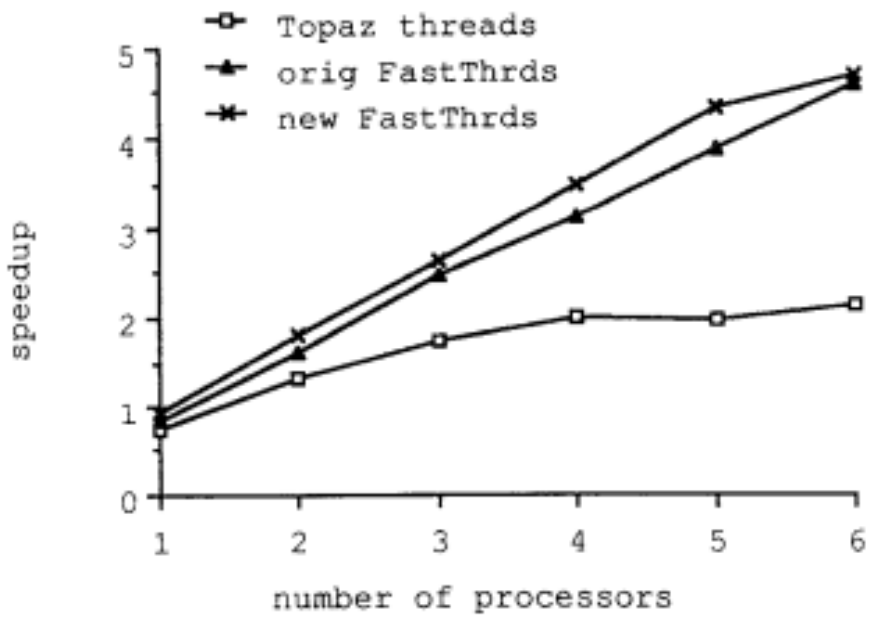


Fig. 2. Speedup of N-Body application versus number of processors, 100% of memory available.

PERFORMANCE

- N-body problem in O(NlogN)
- 6 processors (fixed)

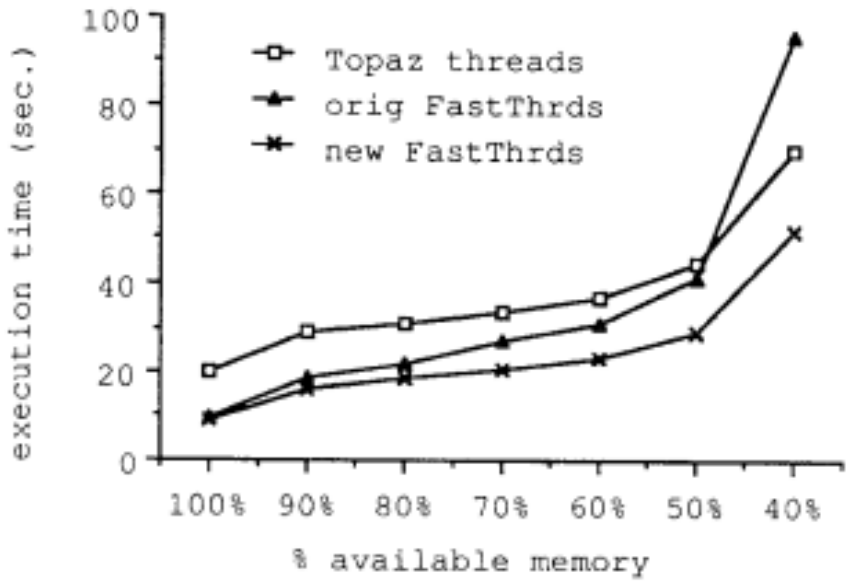


Fig. 3. Execution time of N-Body application versus amount of available memory, 6 processors.

PERFORMANCE

- N-body problem in O(NlogN)
- Two executions at the same time

Table V. Speedup of N-Body Application, Multiprogramming Level = 2, 6 Processors, 100% of Memory Available

Topaz threads	Original FastThreads	New FastThreads
1.29	1.26	2.45

THANK YOU!

Any questions?