Scheduler Activation

Thomas E. Anderson, Brian N. Bershad, Edward D. Lazowska, and Henry M. Levy

SOSP.1991

Thread

Background of the paper / Why

Why

- Why do we need threads?

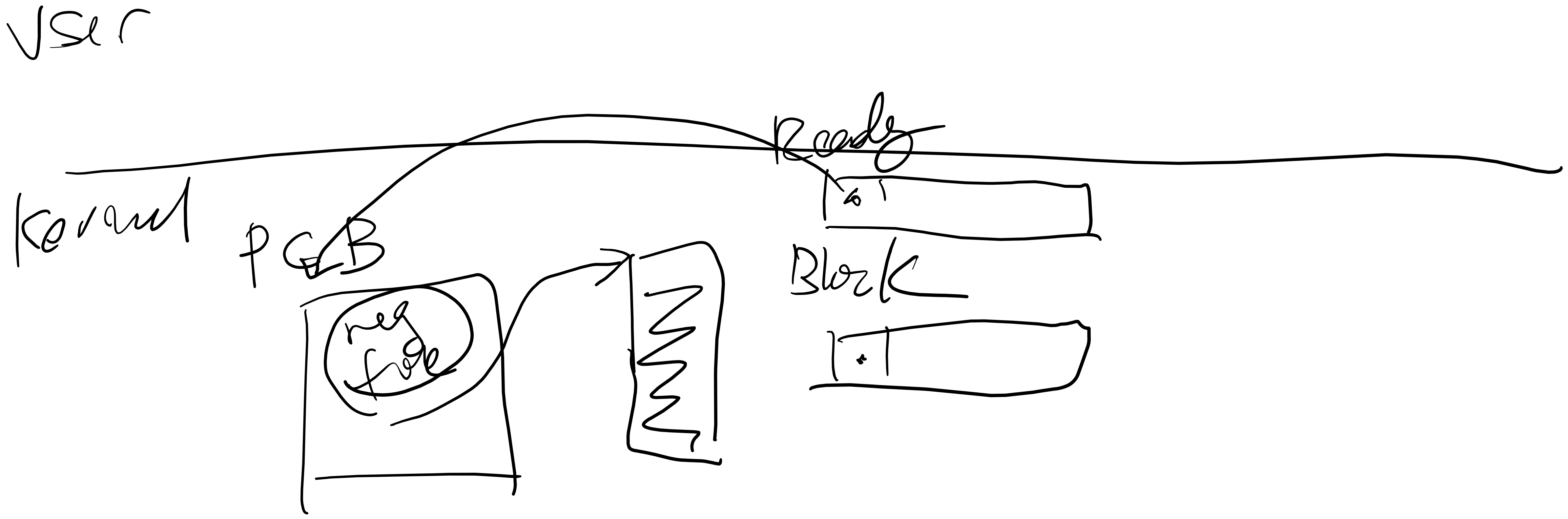
- Why do we still need processes after we have threads?

Why

- Why do we need threads?
  - Performance from parallel execution
  - Share memory
  - Cost with every process
- Why do we still need processes after we have threads?

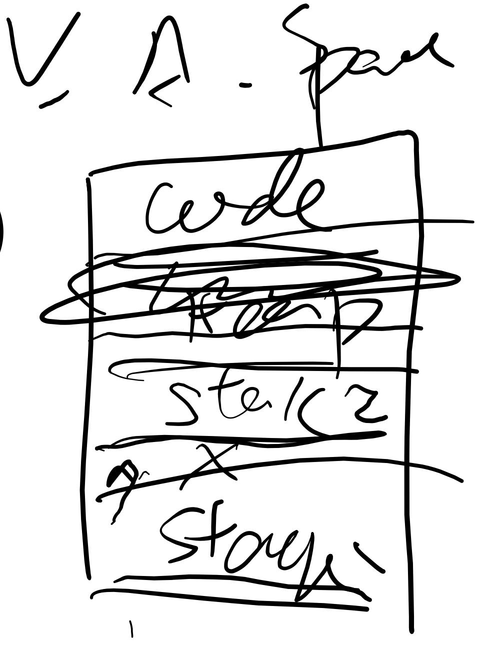
Process vs. Thread (roles, resources)

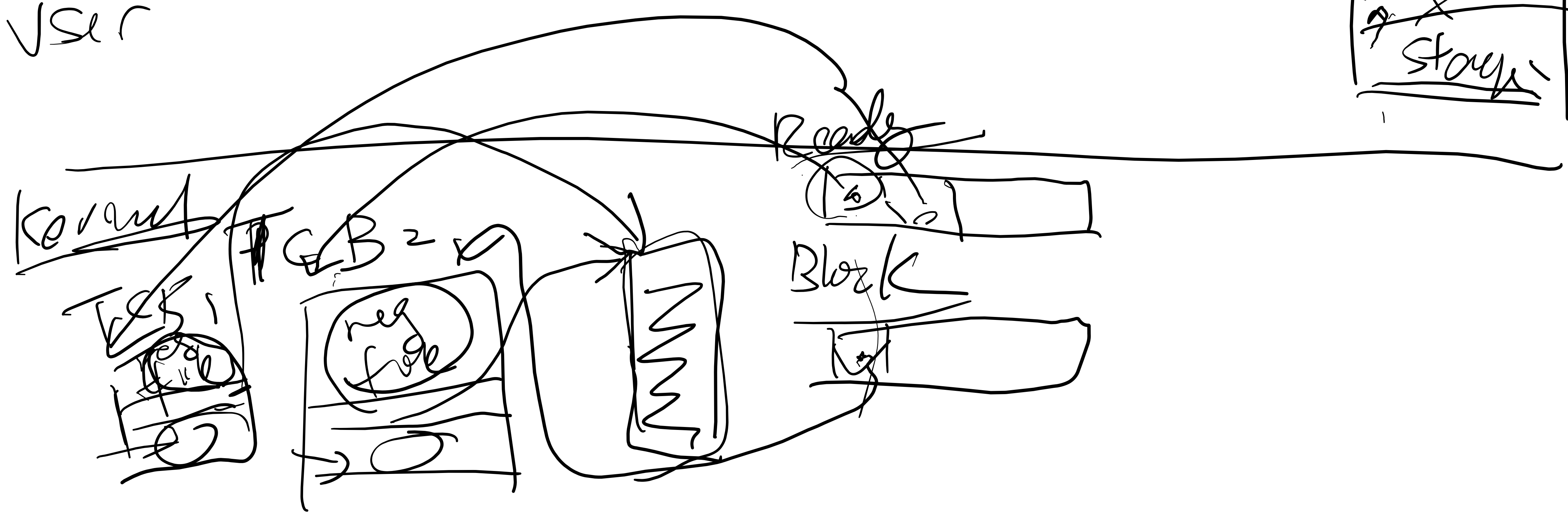
- Process (resource unit)
  - Page table
  - Open file table
- Thread (execution unit)
  - Register



Process vs. Thread (roles, resources)

- Process (resource unit)
  - Page table
  - Open file table
- Thread (execution unit)
  - Register





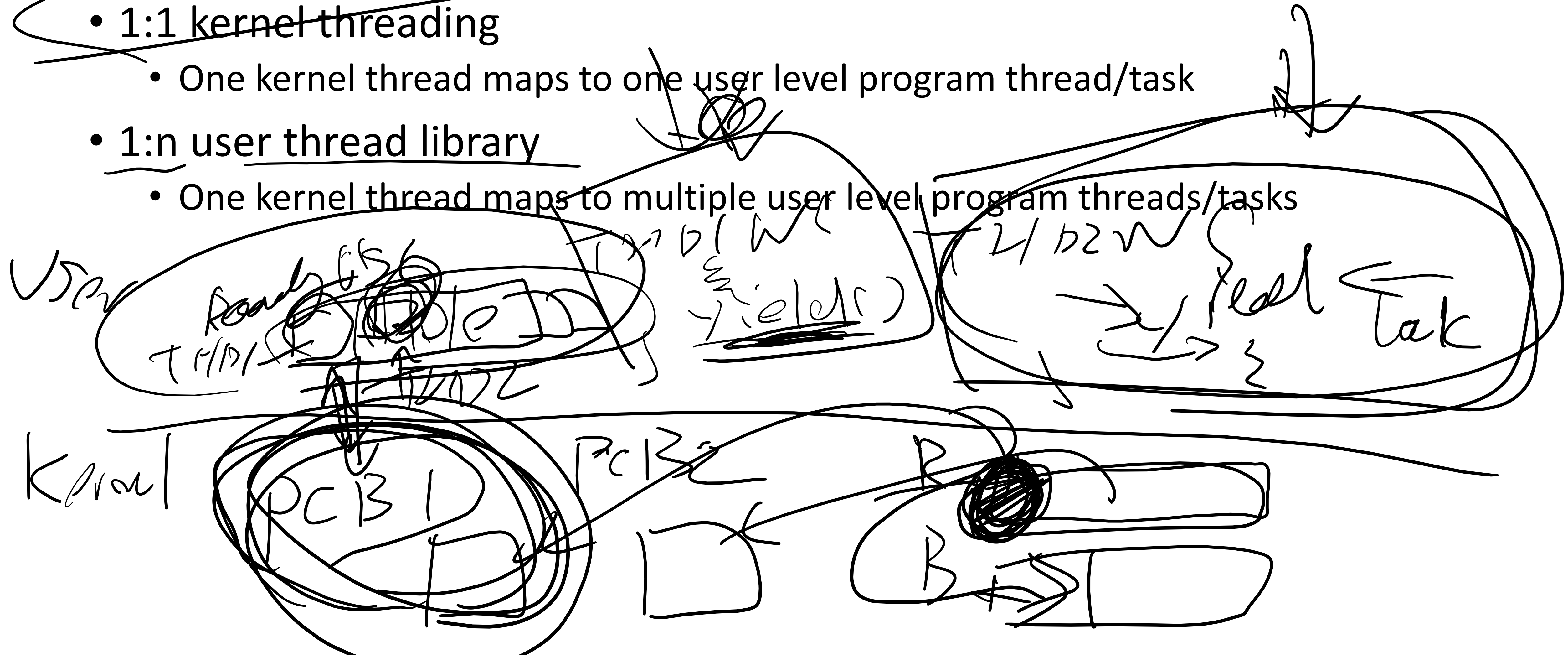
Background: process implementation

- Scheduling in the era of processes

- I/O blocking in the era of processes

Background: thread implementation before this paper

- 1:1 kernel threading
  - One kernel thread maps to one user level program thread/task
- 1:n user thread library
  - One kernel thread maps to multiple user level program threads/tasks



1:1 kernel thread implementation

- How to implement
- How to create a thread
- How to do context switch
- How to do synchronization?
- What happened at an I/O blocking?

1:n user level thread implementation

- How to implement
- How to create a thread
- How to do context switch
- How to do synchronization
- What happened at an I/O blocking
- What happened at an I/O unblocking

Kernel threading vs. User threading

- User threading
    - Disadvantage
    - Advantage
- Kernel threading
    - Advantage
    - Disad …

Kernel threading vs. User threading

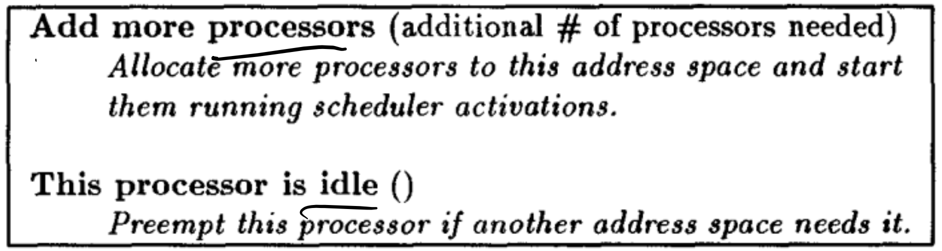
- User threading
  - Disadvantage
    - Poor integration with system events (I/O, …)
      - I/O blocking event
  - Advantage
    - Fast in thread creation, synchronization
    - Flexible, customized
- Kernel threading
  - Advantage
    - Great integration with system events
  - Disad …

A simple m:n implementation

- M kernel threads for n user level threads
- What happens at an I/O blocking
- What happens at an I/O unblocking?

Scheduler activation

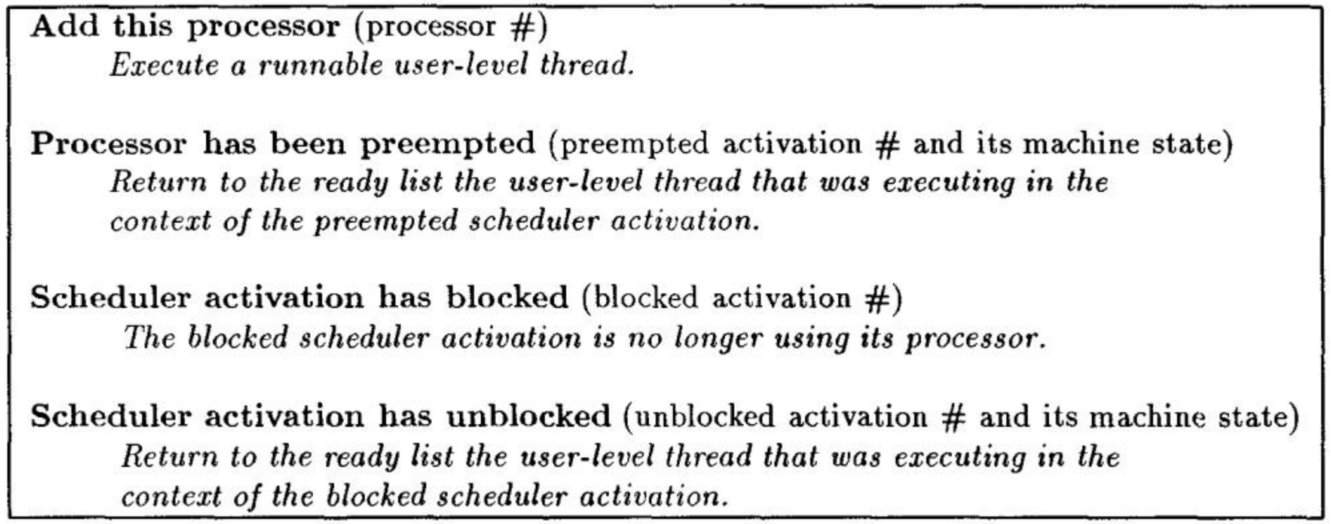
- M:N with communication
  - **Up Calls** and system calls
  - Kernel offers mechanisms
  - User-level offers policies



Add more processors (additional # of processors needed)
Allocate more processors to this address space and start them running scheduler activations.

This processor is idle ()
Preempt this processor if another address space needs it.

Table 3: Communication from the Address Space to the Kernel



Add this processor (processor #)
Execute a runnable user-level thread.

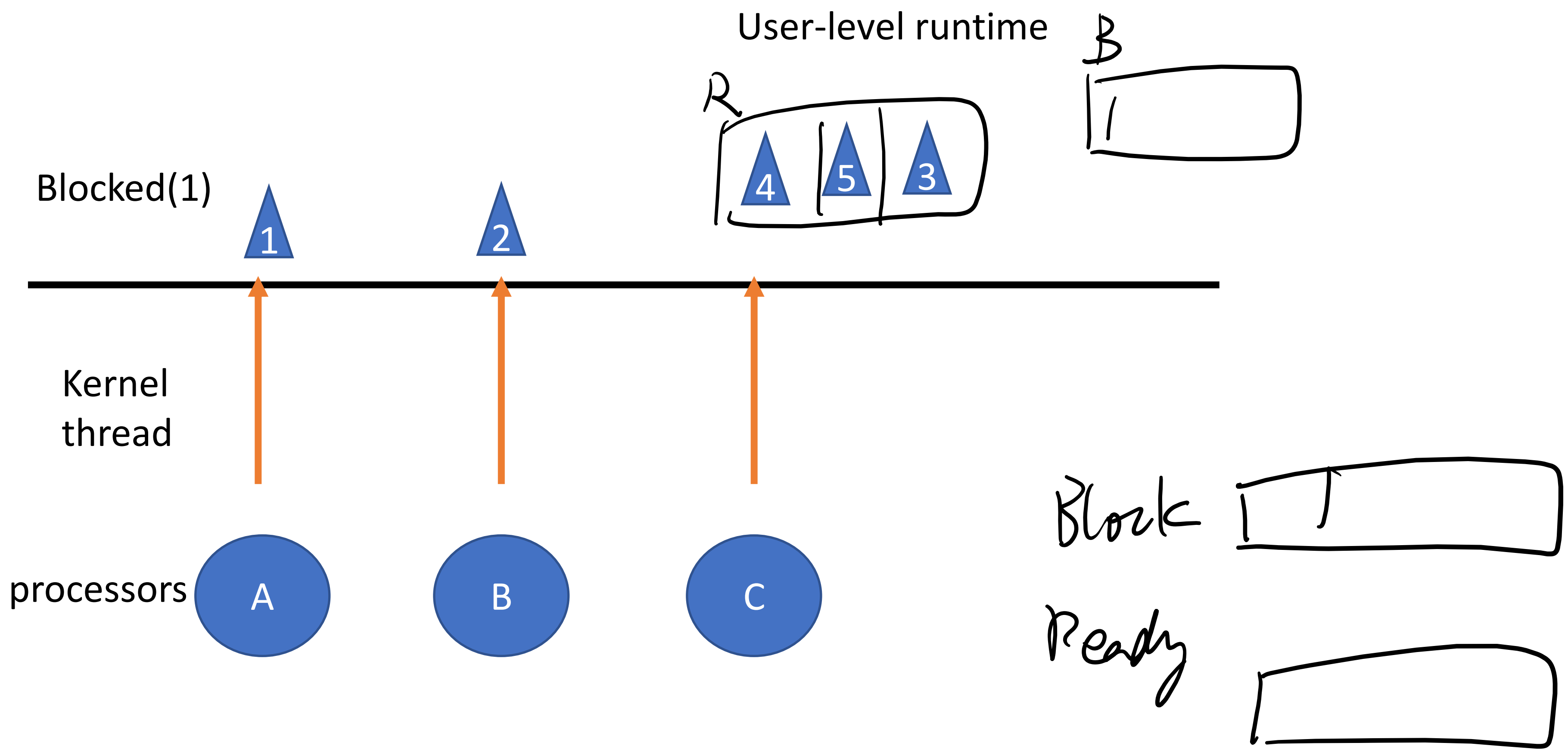
Processor has been preempted (preempted activation # and its machine state)
Return to the ready list the user-level thread that was executing in the context of the preempted scheduler activation.

Scheduler activation has blocked (blocked activation #)
The blocked scheduler activation is no longer using its processor.

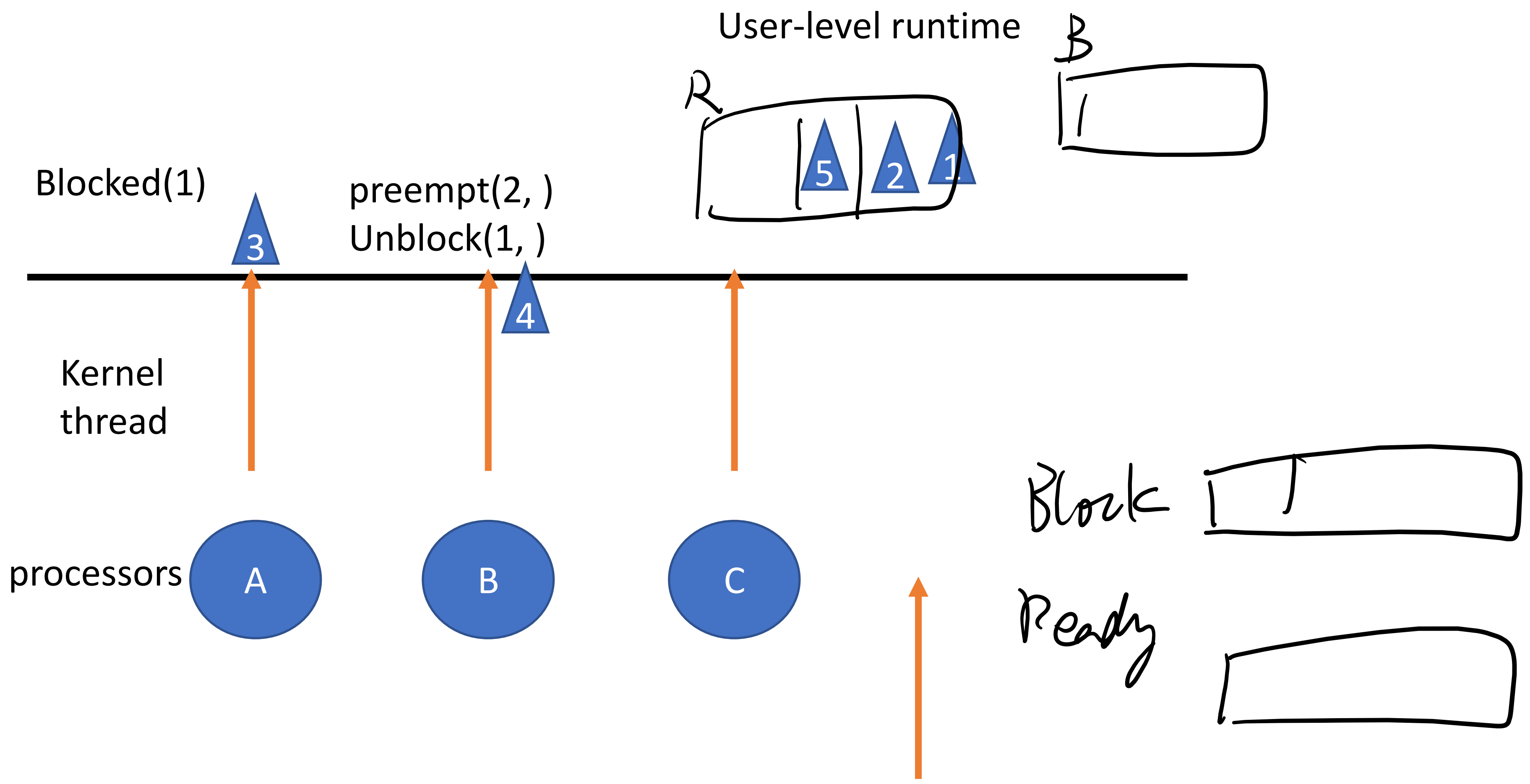
Scheduler activation has unblocked (unblocked activation # and its machine state)
Return to the ready list the user-level thread that was executing in the context of the blocked scheduler activation.

Table 2: Scheduler Activation Upcall Points

Example: when an I/O blocking happens



When the I/O is unblocked



Impact of SA

- Kernel thread has advanced
    - Creation faster
    - Synchronization faster