Scheduler Activations

Virginia Tech

Concurrent Processing

How can concurrent processing activity be structured on a single processor?

How can application-level information and system-level information be combined to provide efficient scheduling of processing activities?

Reasons for Concurrency
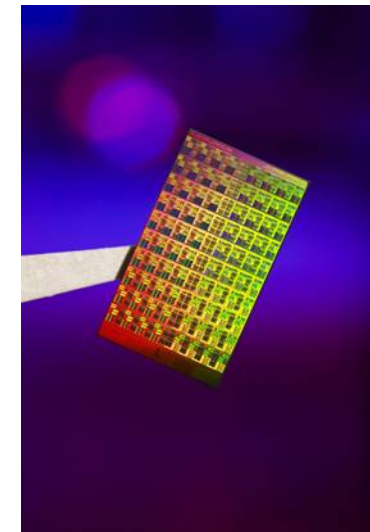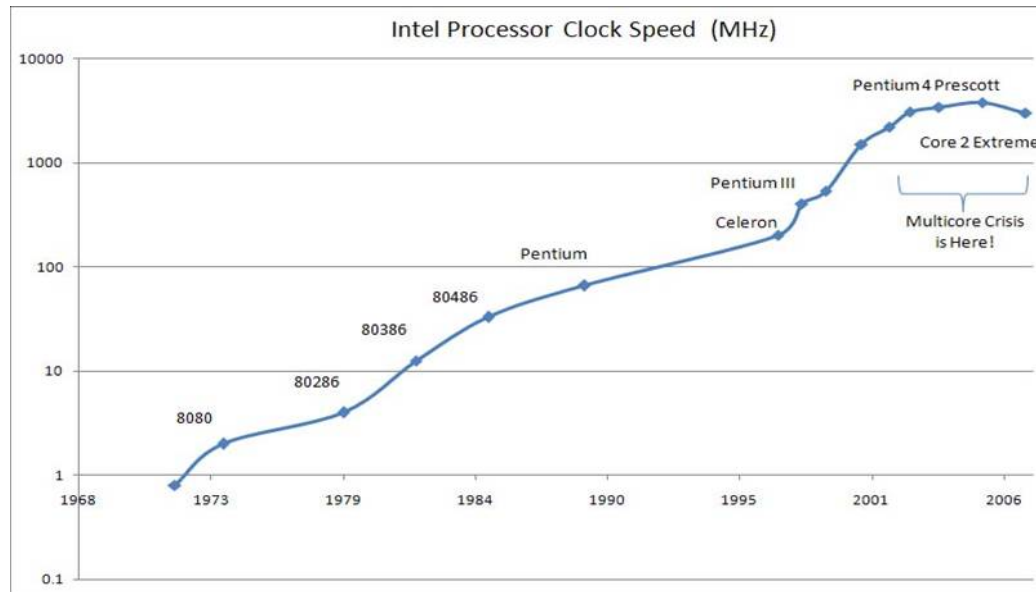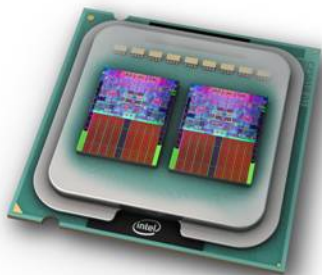

multitasking

parallelism

coordination

performance

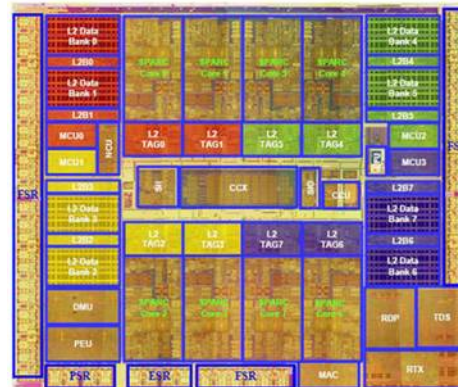# Technologies Driving Concurrent Programming

Intel Processor Clock Speed (MHz)

Intel: 80 core experimental system

Sun: 8 core chip

Niagara2 Chip Overview

- 8 Sparc cores, 8 threads each
- Shared 4MB L2, 8-banks, 16-way associative
- Four dual-channel FBDIMM memory controllers
- Two 10/1 Gb Enet ports
- One PCI-Express x8 1.0A port
- 342 mm^2 die size in 65 nm
- 711 signal I/O, 1831 total

Intel: Quad Core

Synchronization

- Difficulty in controlling concurrency via blocking
  - ☐ **Deadlock**
  - ☐ **Priority inversion**
  - ☐ **Convoying**

- A variety of "correct" answers
  - ☐ **Sequential consistency (Lamport): individual operations on a shared resource (e.g., memory).**
  - ☐ **Serializability: a group of operations (transaction) which may be interleaved with operations of another group (transaction) each operating on a shared resource (e.g., a database).**
  - ☐ **Linearizability (Herlihy, Wing): a group of operations (transaction) not interleaved with operations of another group (transaction) each operating on a shared object.**

Context

Support for concurrent and parallel programming
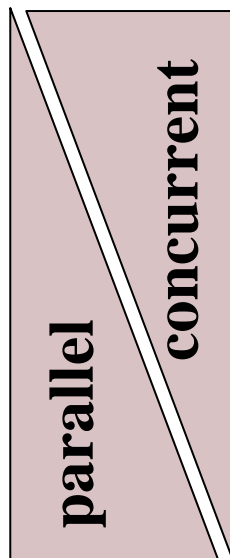
conform to application semantics

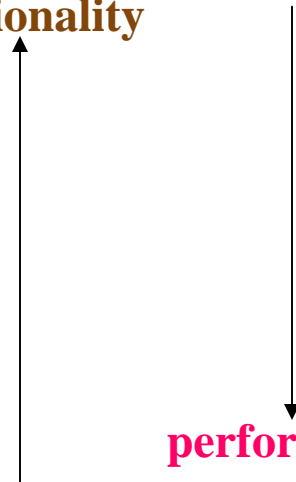respect priorities of applications

no unnecessary blocking
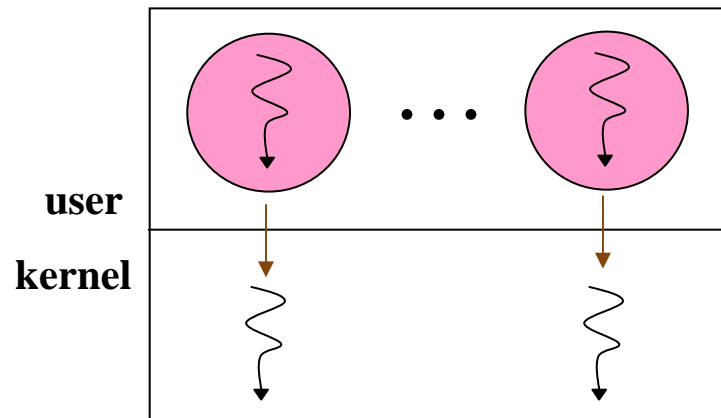
fast context switch

high processor utilization

parallel / **concurrent**

relative importance

functionality

performance

Virginia Tech

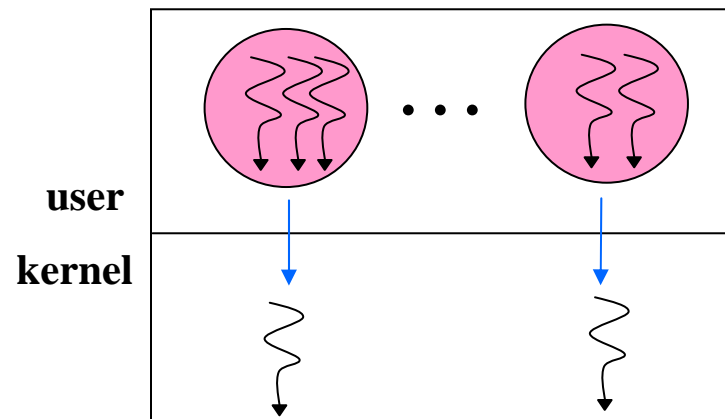# "Heavyweight" Process Model

- simple, uni-threaded model
- security provided by address space boundaries
- high cost for context switch
- coarse granularity limits degree of concurrency

"Lightweight" (User-level) Threads

user

kernel

- thread semantics defined by application
- fast context switch time (within an order of magnitude of procedure call time)
- system scheduler unaware of user thread priorities
- unnecessary blocking (I/O, page faults, etc.)
- processor under-utilization

Kernel-level Threads

user
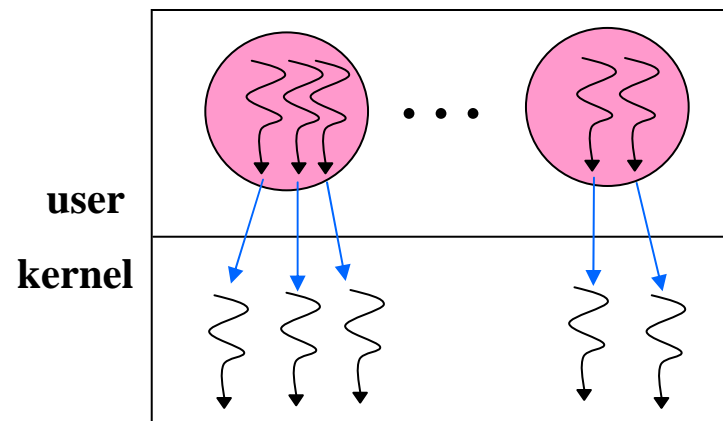
kernel

- thread semantics defined by system
- overhead incurred due to overly general implementation and cost of kernel traps for thread operations
- context switch time better than process switch time by an order of magnitude, but an order of magnitude worse than user-level threads
- system scheduler unaware of user thread state (e.g, in a critical region) leading to blocking and lower processor utilization
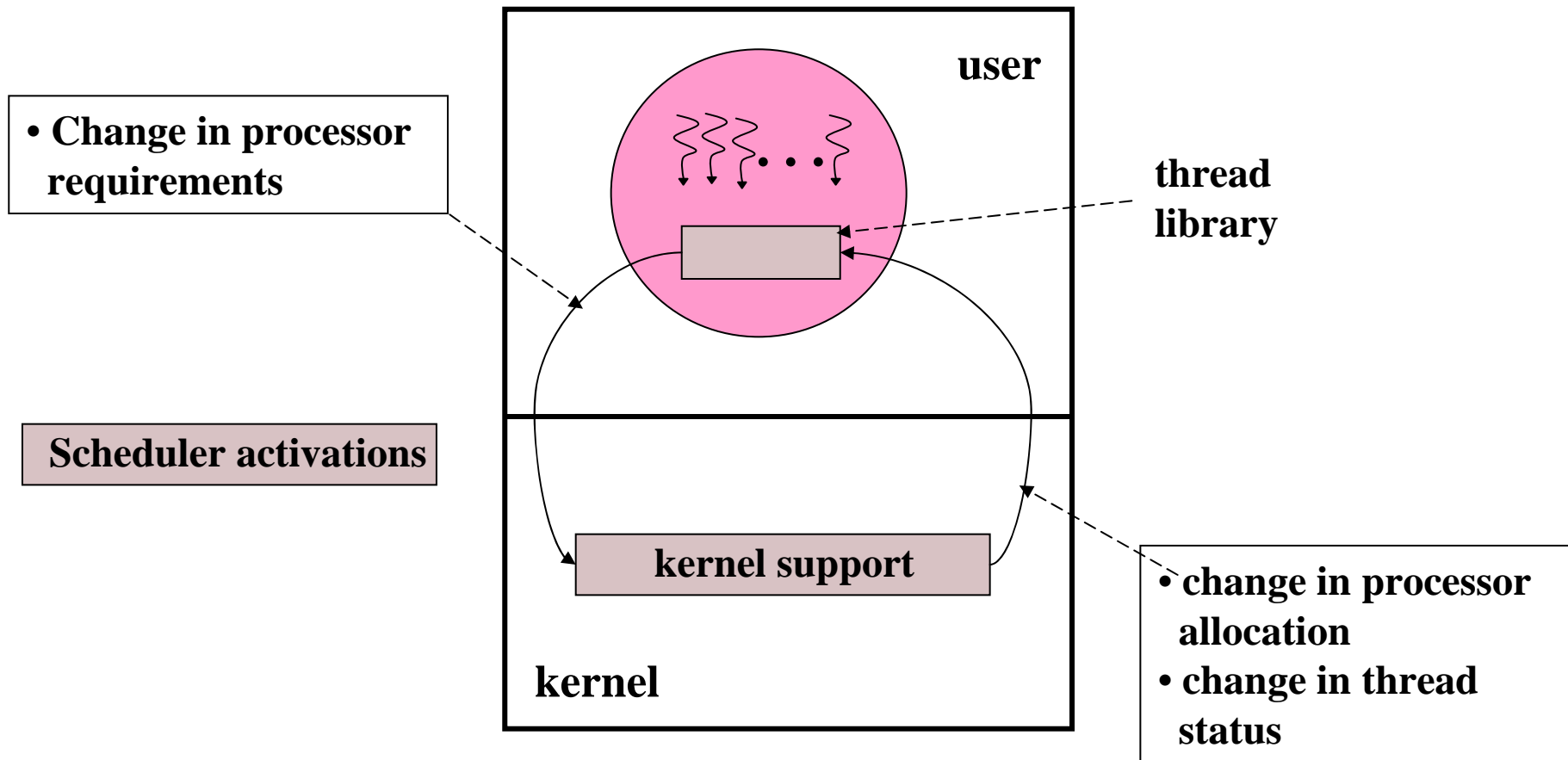
Problem

- Application has knowledge of the user-level thread state but has little knowledge of or influence over critical kernel-level events (by design! to achieve the virtual machine abstraction)
- Kernel has inadequate knowledge of user-level thread state to make optimal scheduling decisions

Solution: a mechanism that facilitates exchange of information between user-level and kernel-level mechanisms.

A general system design problem: communicating information and control across layer boundaries while preserving the inherent advantages of layering, abstraction, and virtualization.

Scheduler Activations: Structure

• **Change in processor requirements**

Scheduler activations

user

thread library

kernel support

kernel

• change in processor allocation
• change in thread status

Communication via Upcalls

The kernel-level scheduler activation mechanism communicates with the user-level thread library by a set of upcalls:
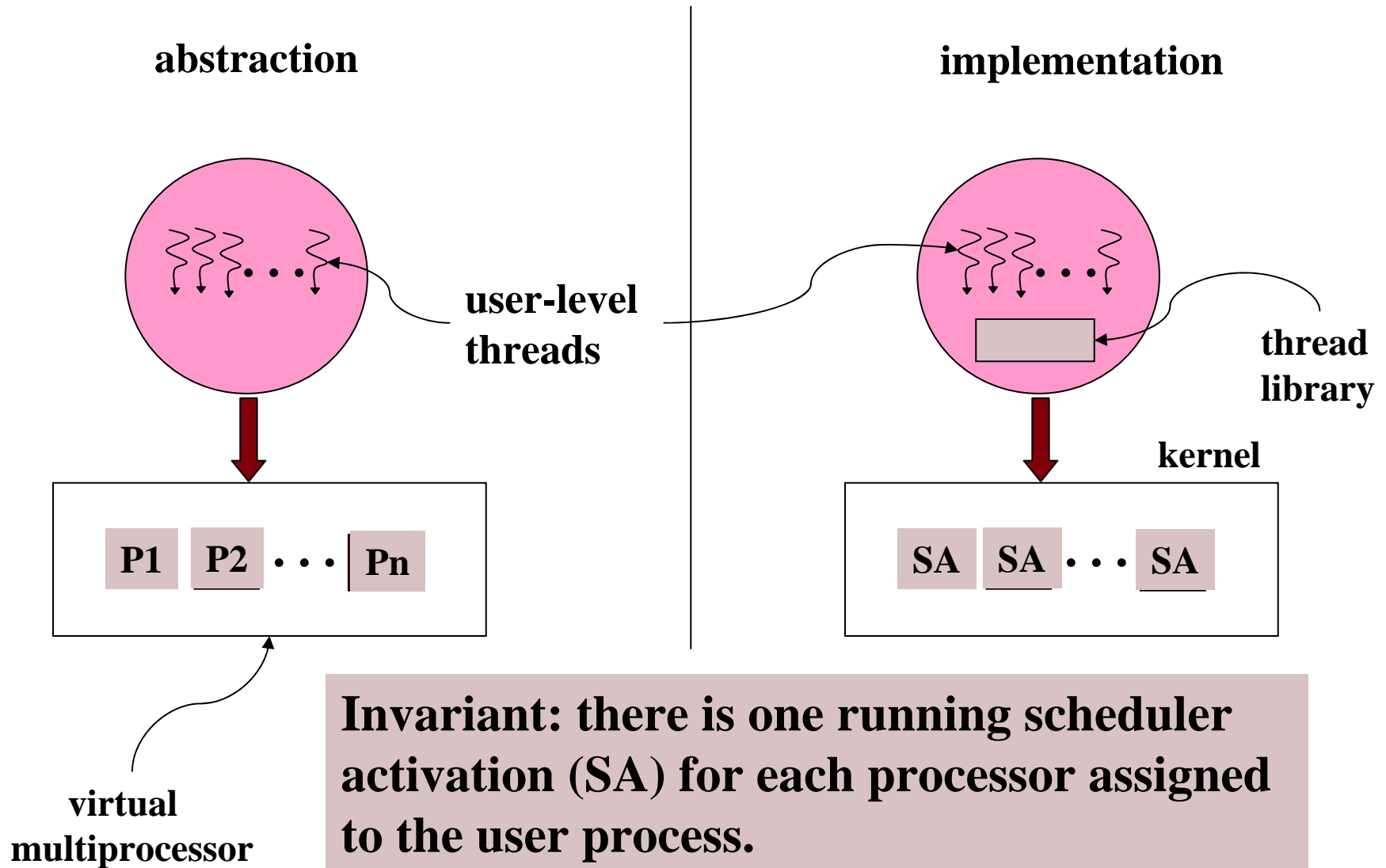
Add this processor (processor #)
Processor has been preempted (preempted activation #, machine state)
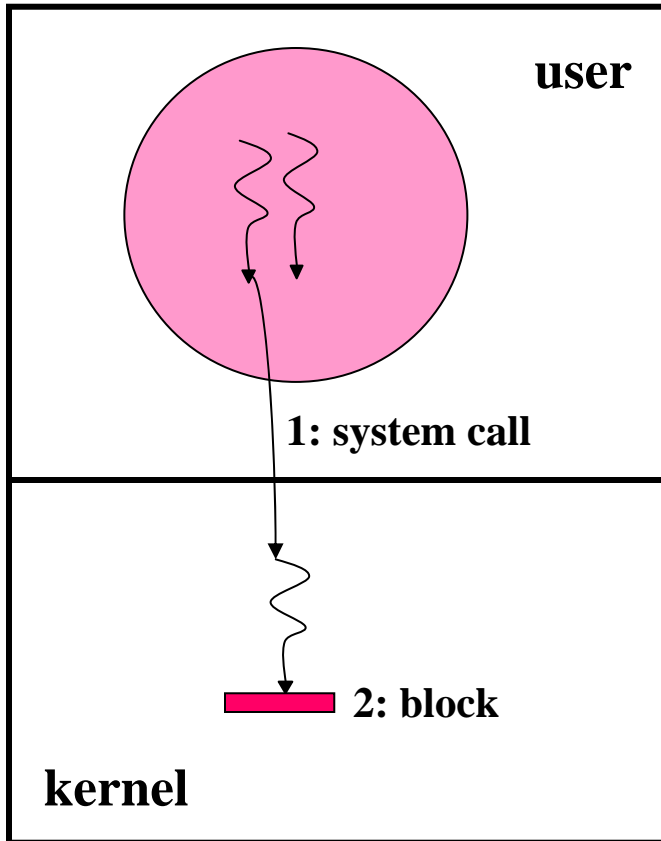Scheduler activation has blocked (blocked activation #)
Scheduler activation has unblocked (unblocked activation #, machine state)

The thread library must maintain the association between a thread's identity and thread's scheduler activation number.
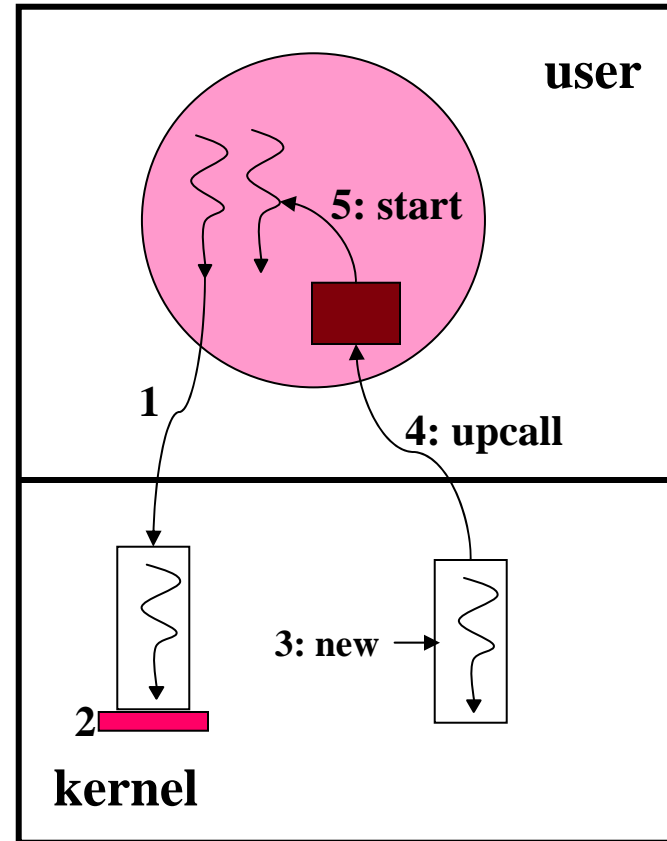
Role of Scheduler Activations

abstraction

implementation

user-level
threads

thread
library

kernel

| P1 | P2 | · · · | Pn |

| SA | SA | · · · | SA |

virtual
multiprocessor

Invariant: there is one running scheduler activation (SA) for each processor assigned to the user process.

Avoiding Effects of Blocking

Kernel threads

Scheduler Activations

Resuming Blocked Thread

user

4
5

3: upcall

2: preempt

1: unblock   kernel
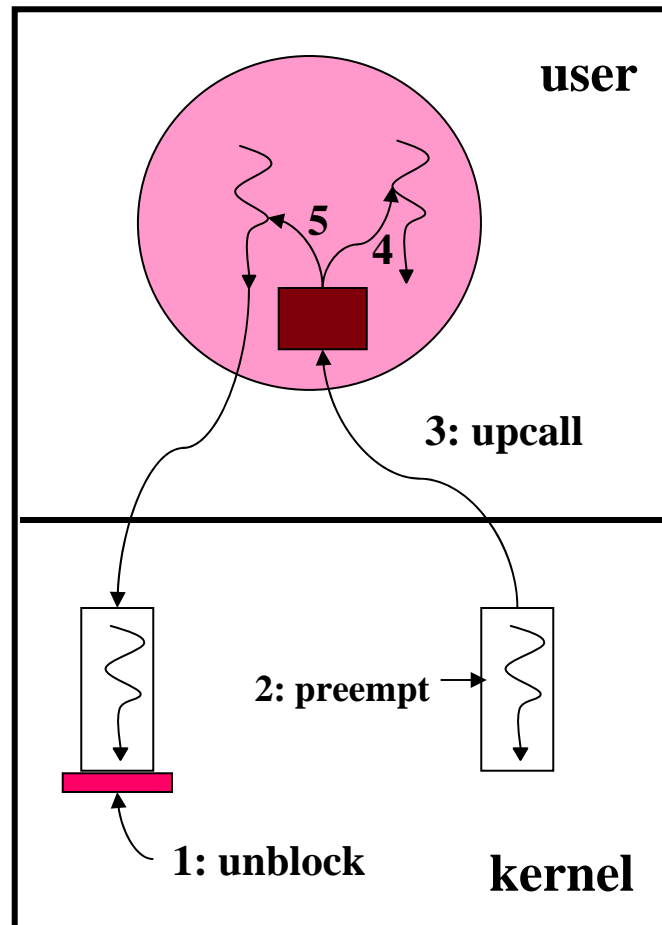
4: preempt
5: resume

Performance

Operation	FastThreads on Topaz Threads	FastThreads on Scheduler Activations	Topaz Threads	Ultrix process
Null fork	34	37	948	11300
Signal-Wait	37	42	441	1840