Processing
(threads, agents, formalism)

How can processing activity be structured on a single processor?

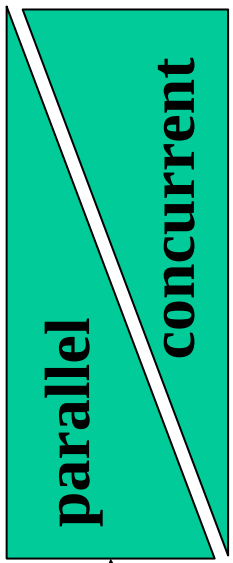
How can application-level information and system-level information be combined to provide efficient scheduling of processing activities?

Why is mobility of a processing activity desired and how can it be achieved?

How can the concepts of communication, processing, and mobility be represented in a formal model?

Context

Support for concurrent and parallel programming



conform to application semantics

respect priorities of applications

no unnecessary blocking

fast context switch

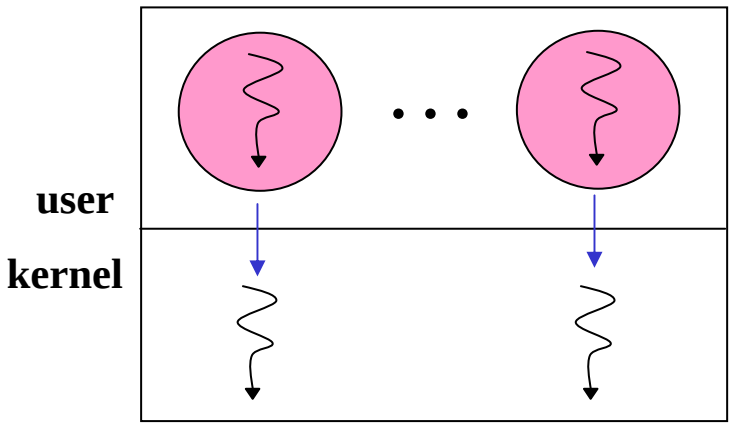
high processor utilization

functional

performance

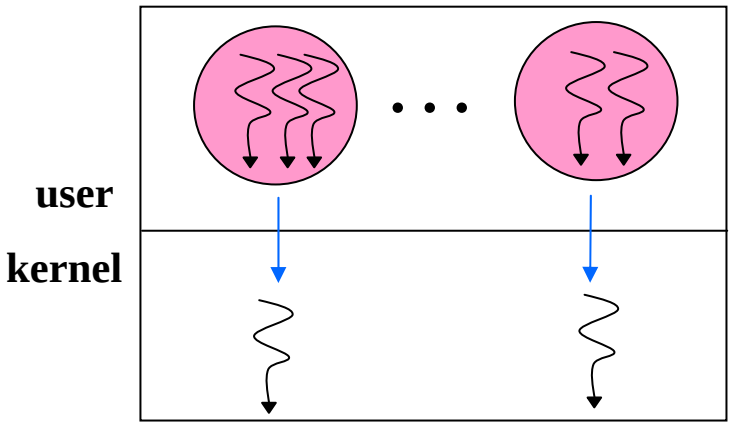
relative importance

“Heavyweight” Process Model



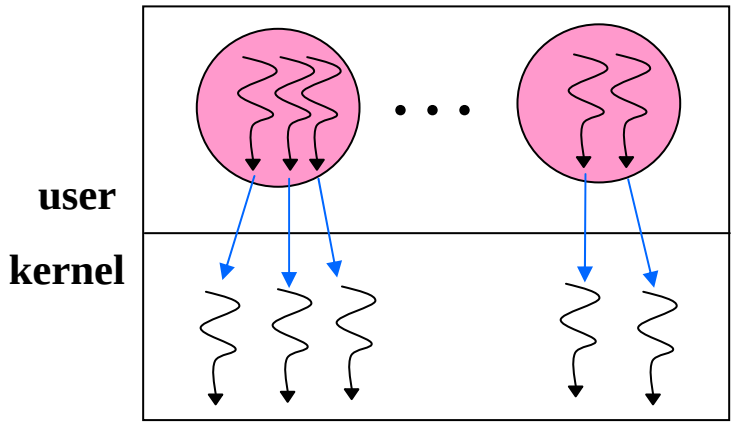
- **simple, uni-threaded model**
- **security provided by address space boundaries**
- **high cost for context switch**
- **coarse granularity limits degree of concurrency**

“Lightweight” (User-level) Threads



- **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



- **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**

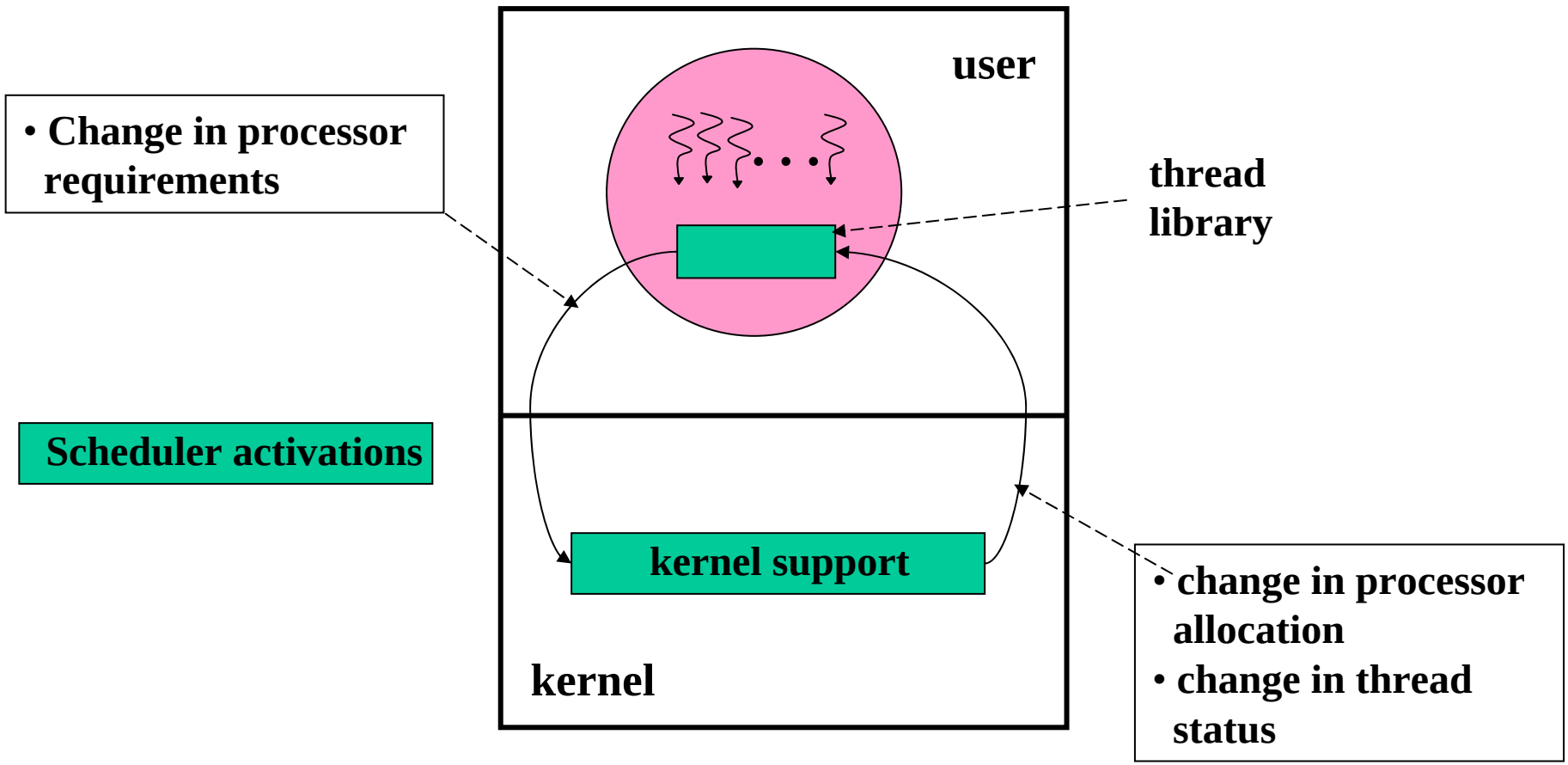
Problem

- 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



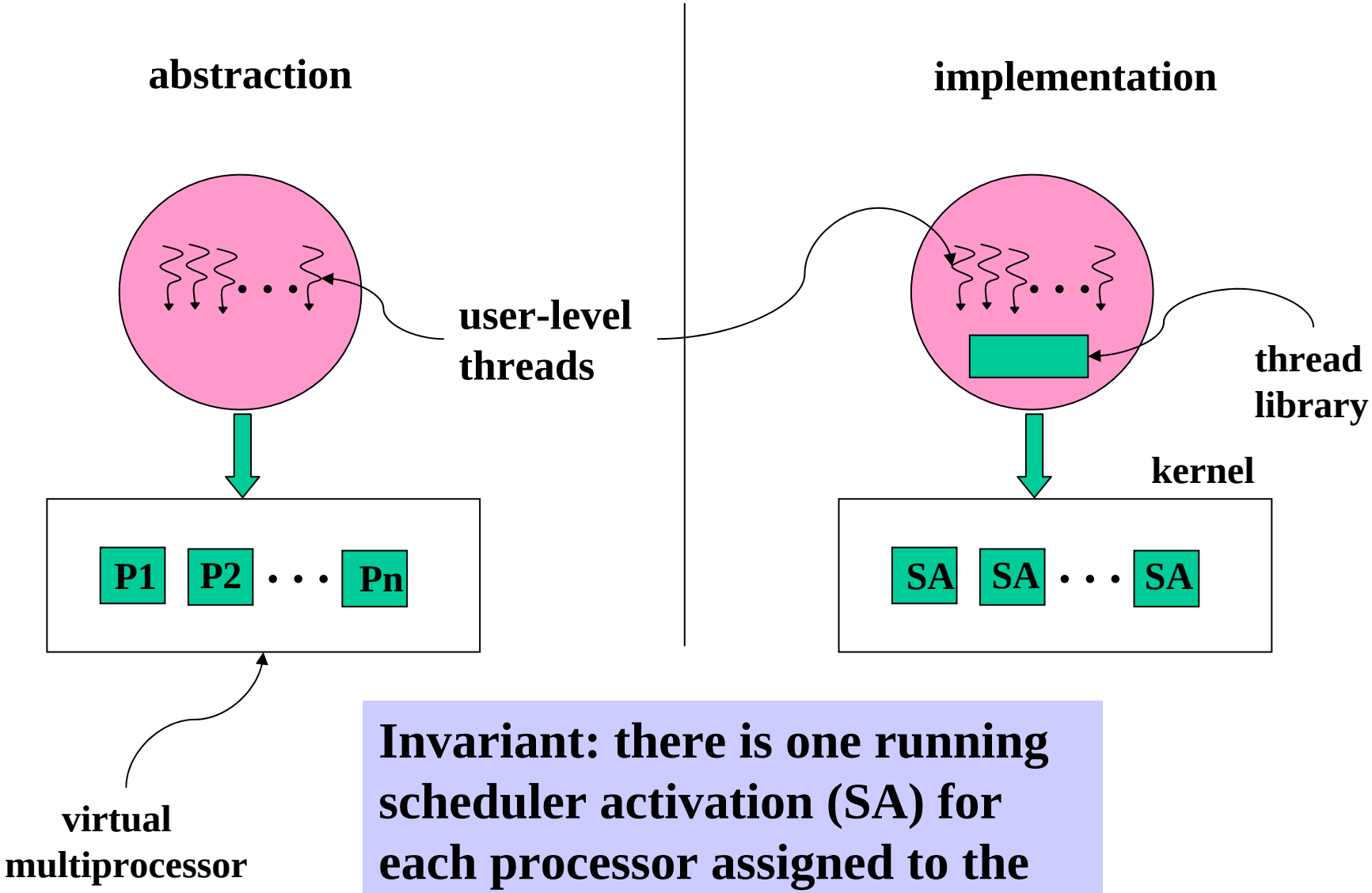
Communication via Upcalls

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)

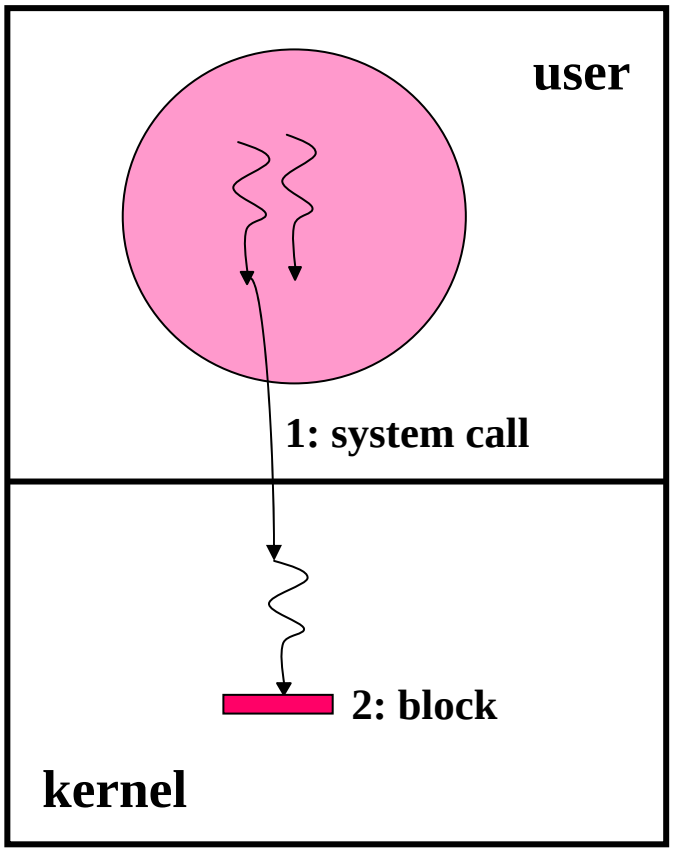
The thread library must maintain the association between a thread's identity and thread's scheduler activation number.

Role of Scheduler Activations

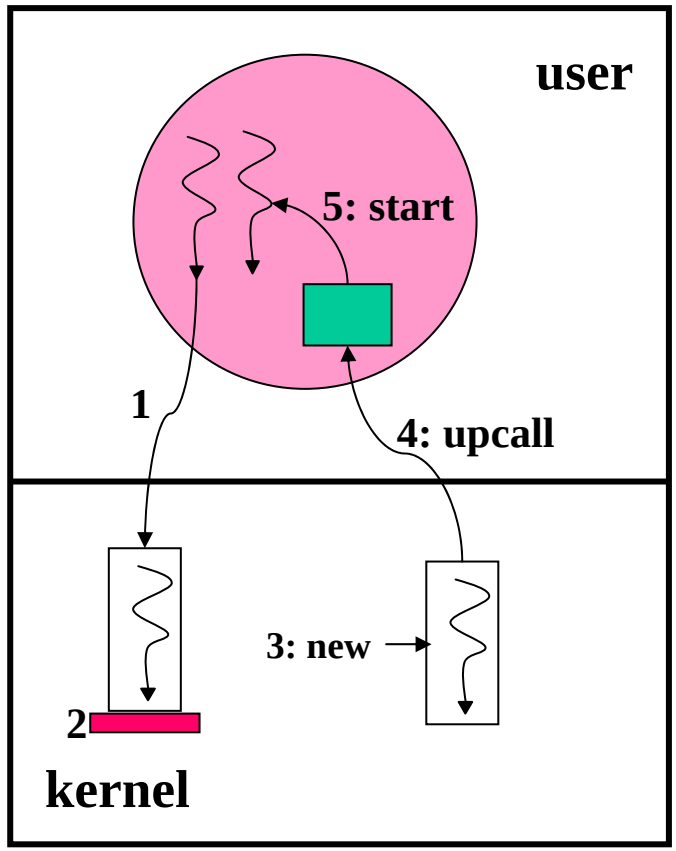


Invariant: there is one running scheduler activation (SA) for each processor assigned to the user process.

Avoiding Effects of Blocking

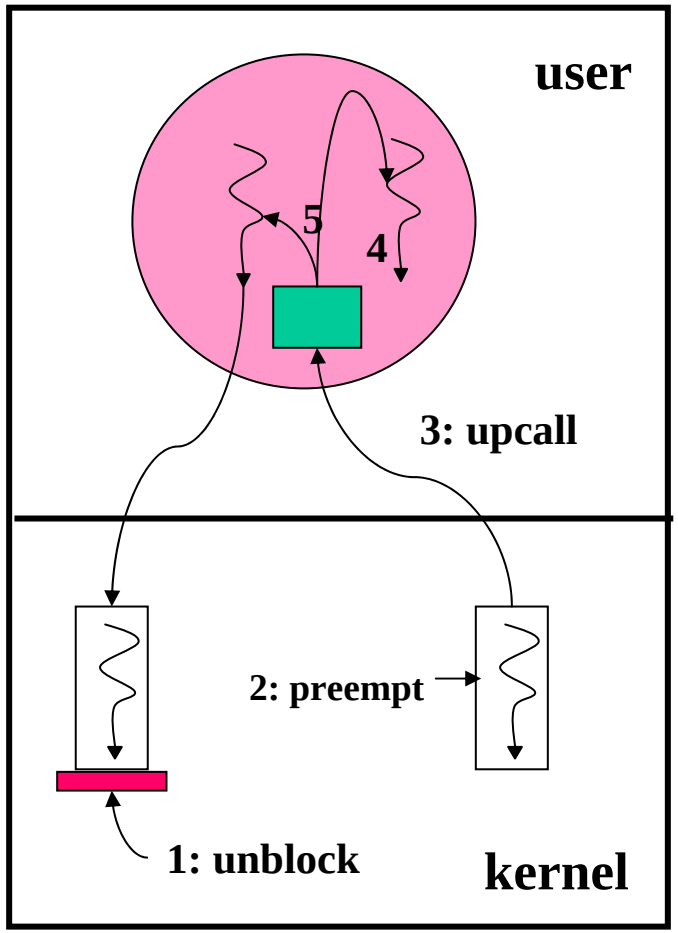


Kernel threads



Scheduler Activations

Resuming Blocked Thread



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**