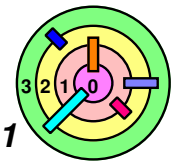
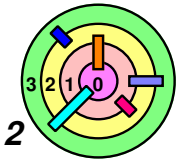
5.1.1.3 Scheduler Activations Model



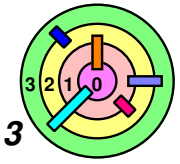
Problems With Two-level Model

- Two-level model does not solve the I/O blocking problem
  - if there are *N* kernel threads and if *N* user threads are blocked in I/O
    - no other user threads can make progress
    - Solaris solution basically goes back to one-level model
- Another problem: *Priority Inversion*
  - user-level thread schedulers are not aware of the kernel-level thread scheduler
    - it may know the number of kernel threads
  - how can the user-level scheduler talk to the kernel-level scheduler?
    - people have tried this, but it's complicated
  - it's possible to have a higher priority user thread scheduled on a lower priority kernel thread and vice versa

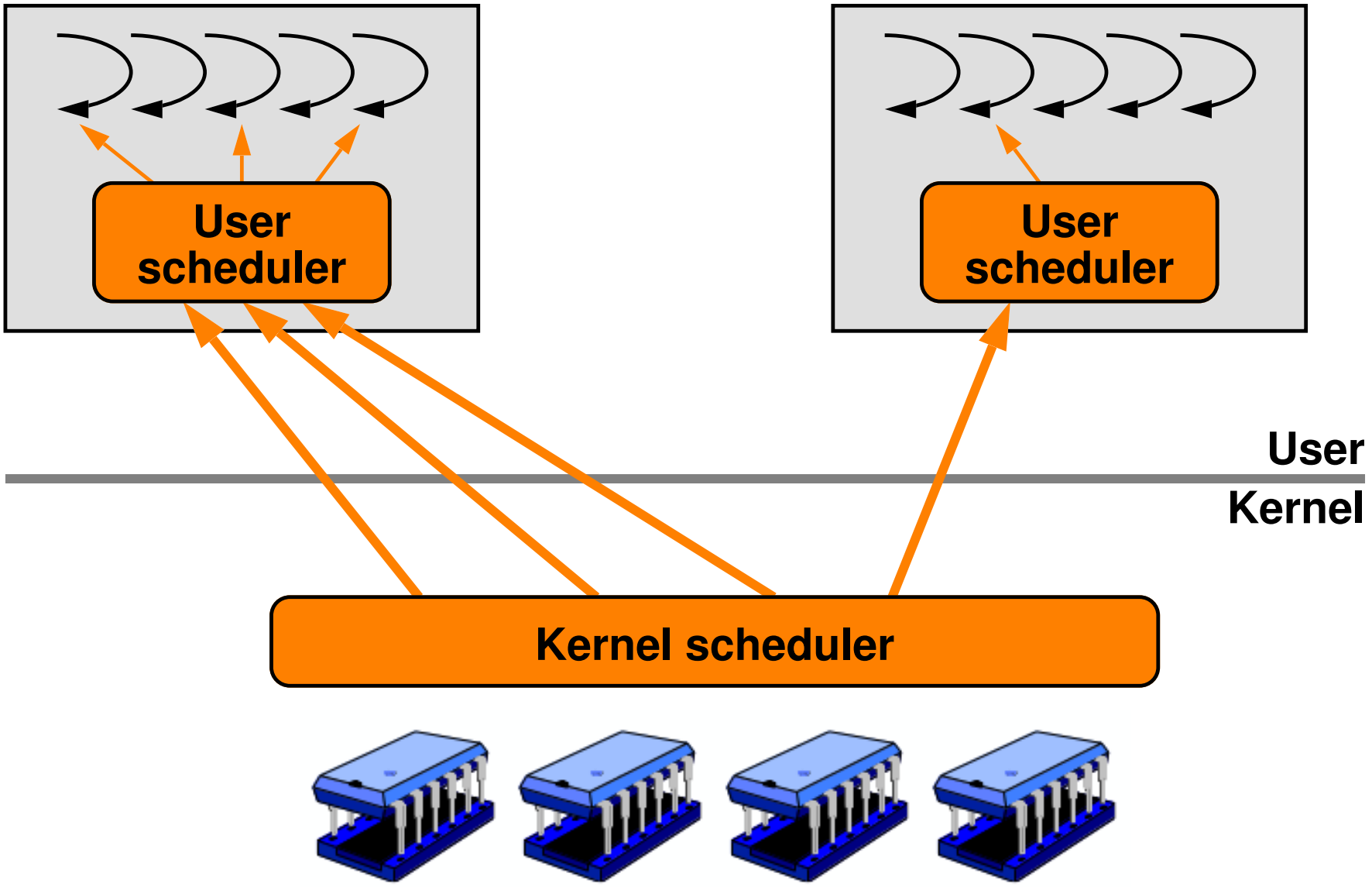


Scheduler Activations Model - Variation on Two-Level Model

- The scheduler activations model is radically different from the other models
  - in other models, we think of the kernel as providing some kernel thread contexts
    - then multiplexing these contexts on processors using the kernel's scheduler
  - in scheduler activations model, we divvy up processors to processes, and processes determine which threads get to use these processors
    - the kernel should supply however many kernel contexts it finds necessary

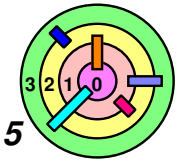


Scheduler Activations Model Example

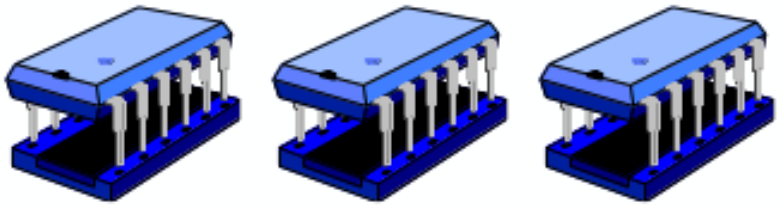


Scheduler Activations Model Example

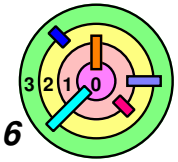
- Let's say a process starts up running a single thread
  - kernel scheduler assigns a processor to the process
  - if the thread blocks, the process gives up the processor to the kernel scheduler
- Suppose the user program creates a new thread and parallelism is desired
  - code in user-level library notifies the kernel that it needs two processors
  - when a processor becomes available, the kernel creates a new kernel context
    - the kernel places an upcall to the user-level library, effectively giving it the processor
    - the user-level library code assigns this processor to the new thread



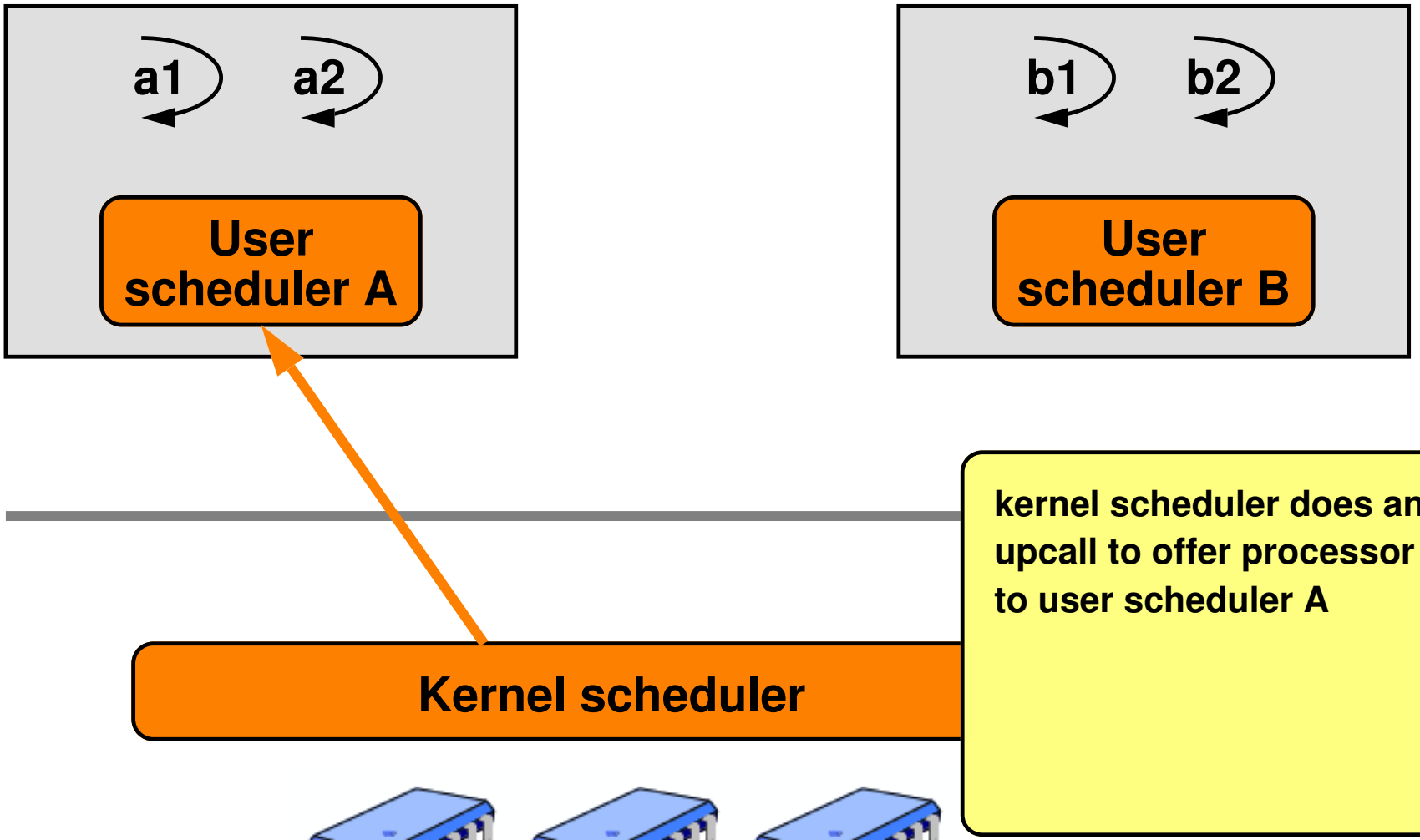
Scheduler Activations Model Example



Kernel scheduler does not schedule threads

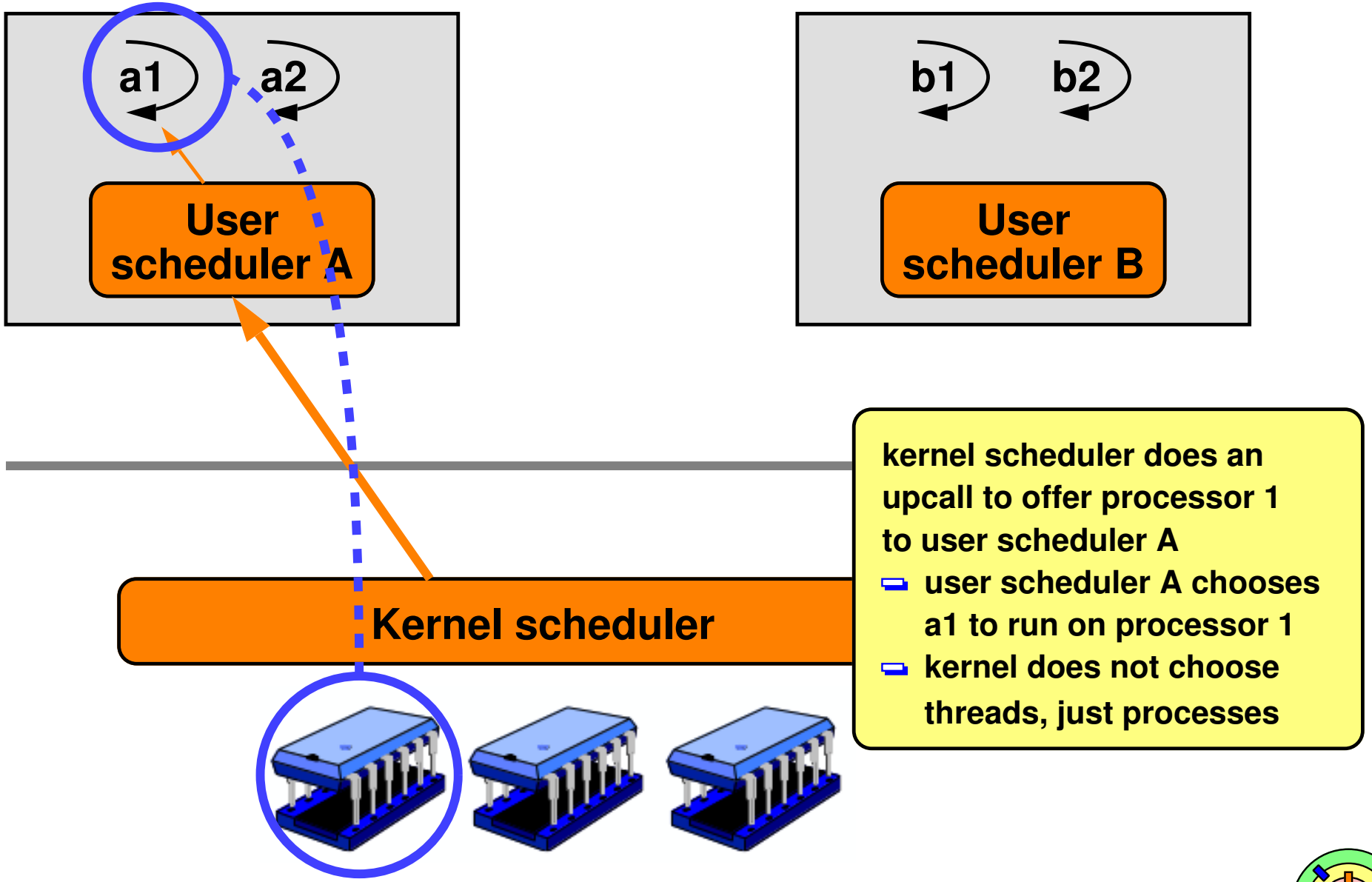


Scheduler Activations Model Example



Kernel scheduler does not schedule threads

Scheduler Activations Model Example

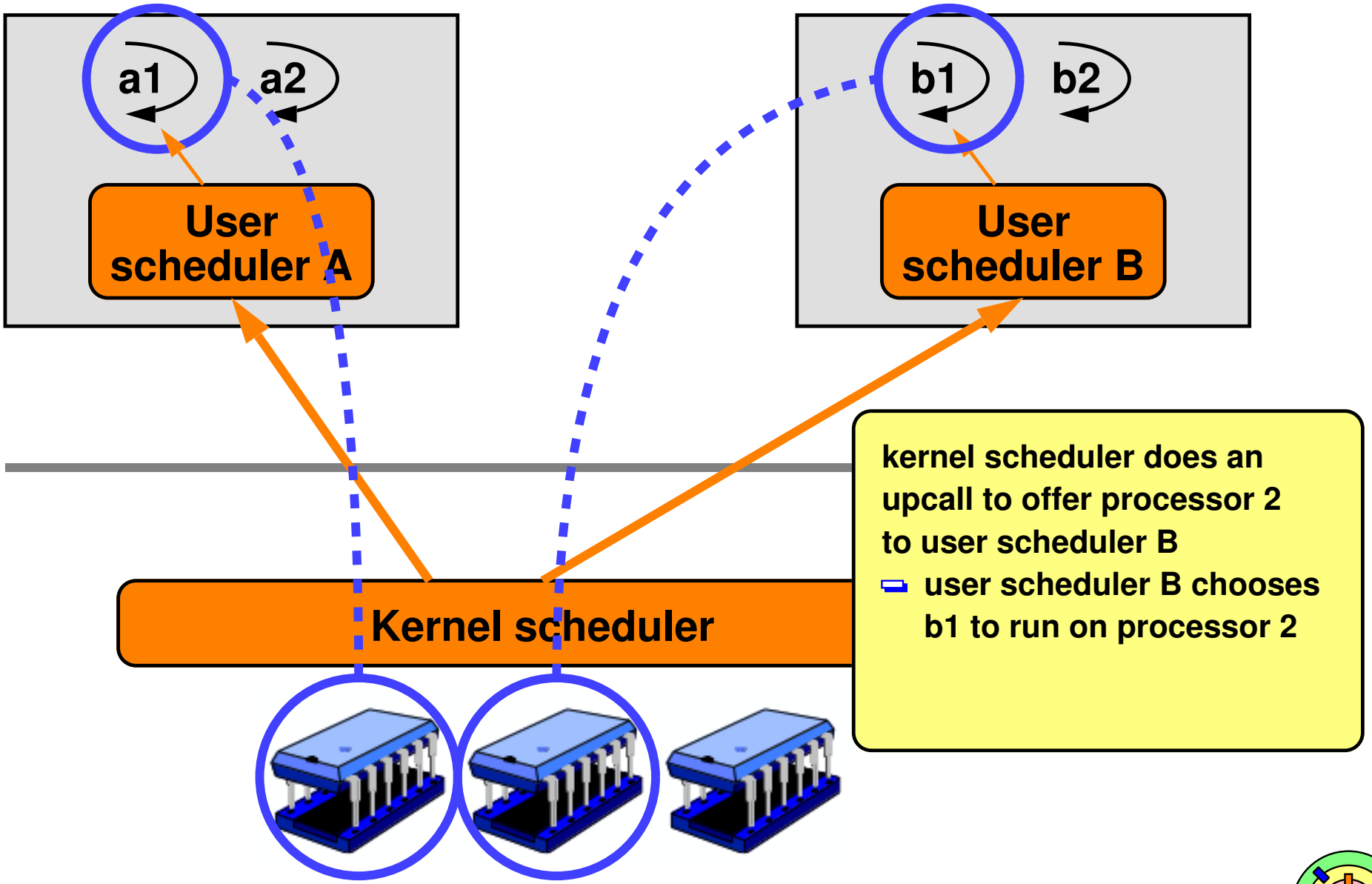


kernel scheduler does an upcall to offer processor 1 to user scheduler A

- user scheduler A chooses a1 to run on processor 1
- kernel does not choose threads, just processes

⇨ Kernel scheduler does not schedule threads

Scheduler Activations Model Example

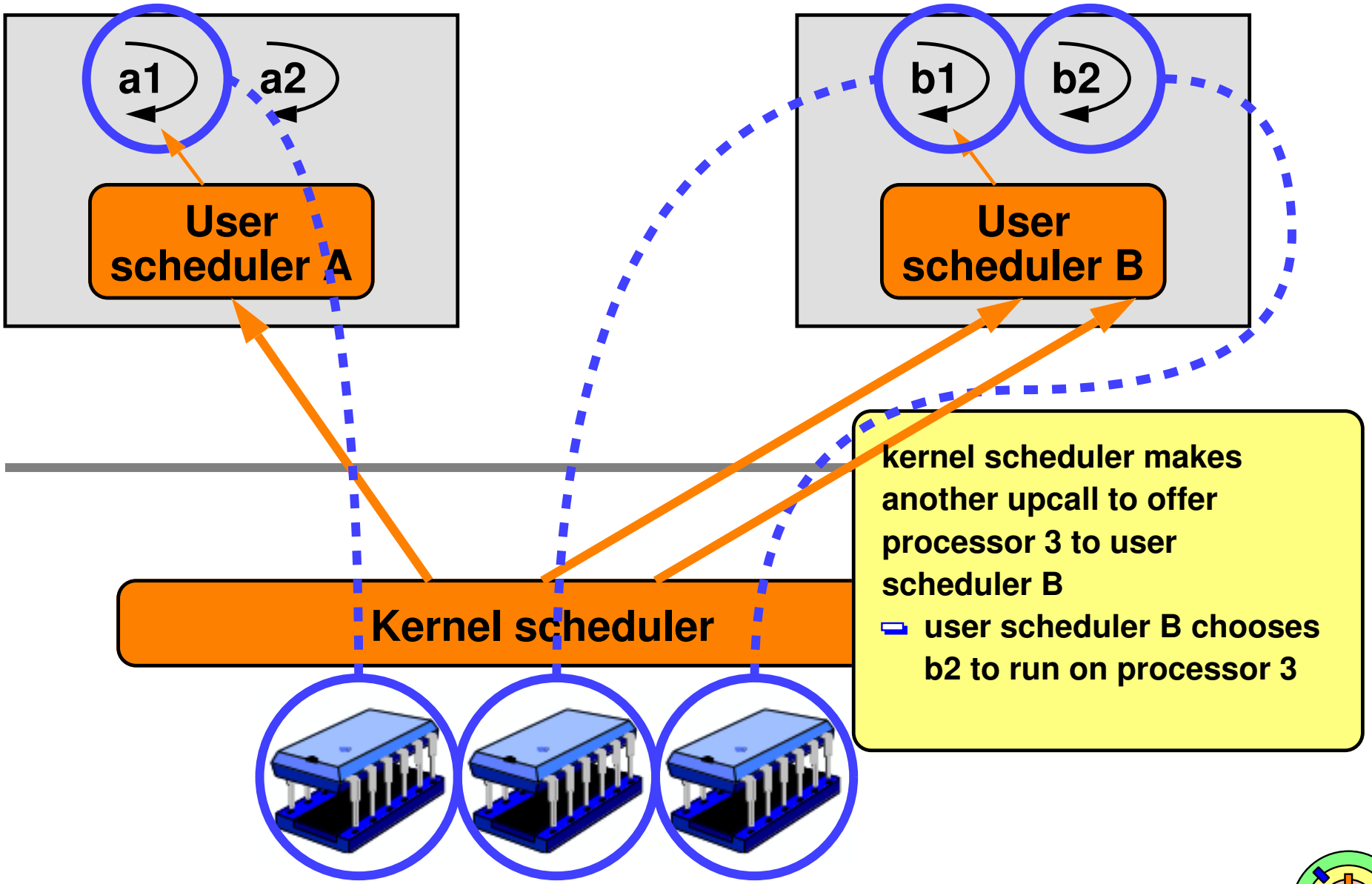


kernel scheduler does an upcall to offer processor 2 to user scheduler B

- user scheduler B chooses b1 to run on processor 2

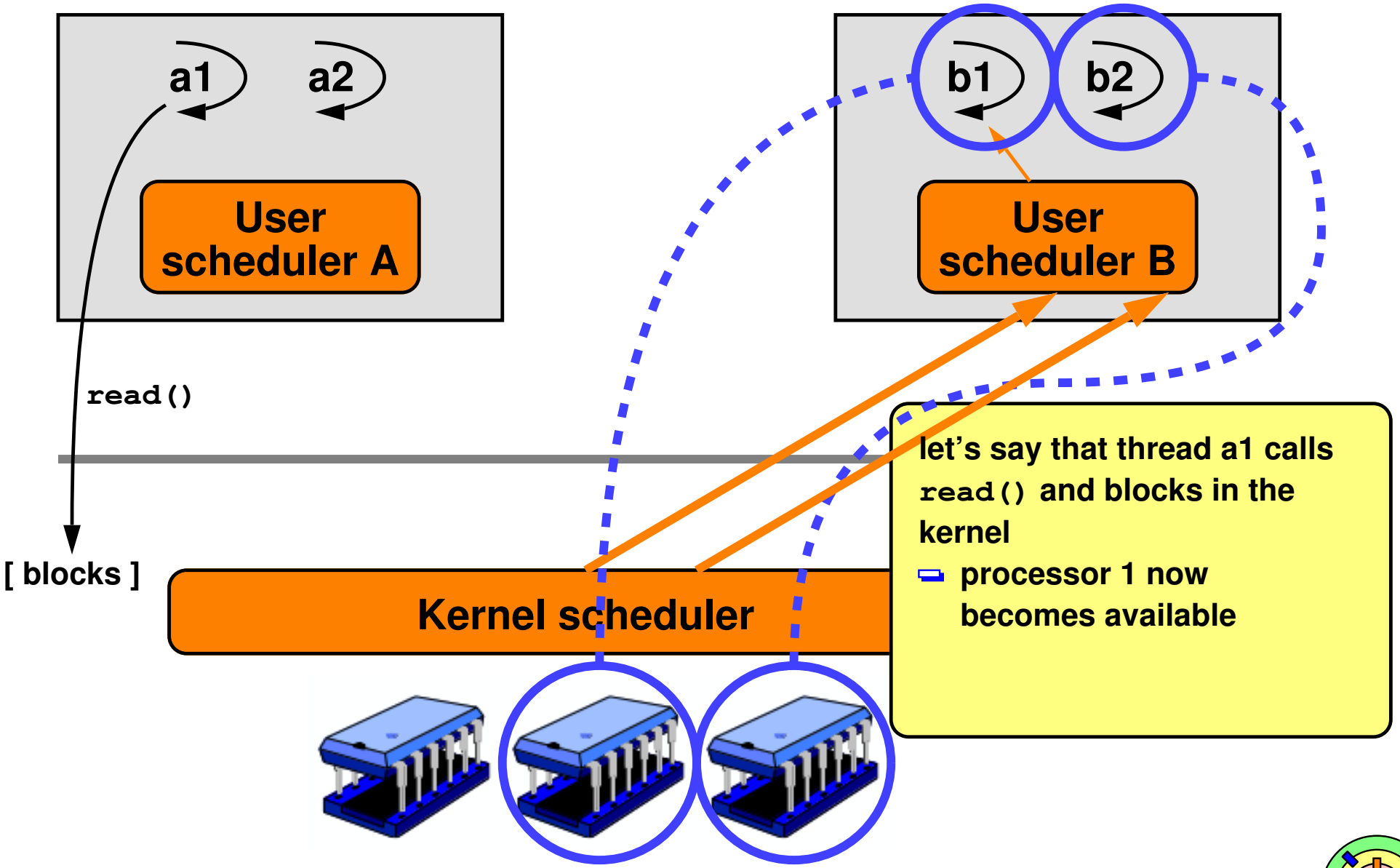
Kernel scheduler does not schedule threads

Scheduler Activations Model Example



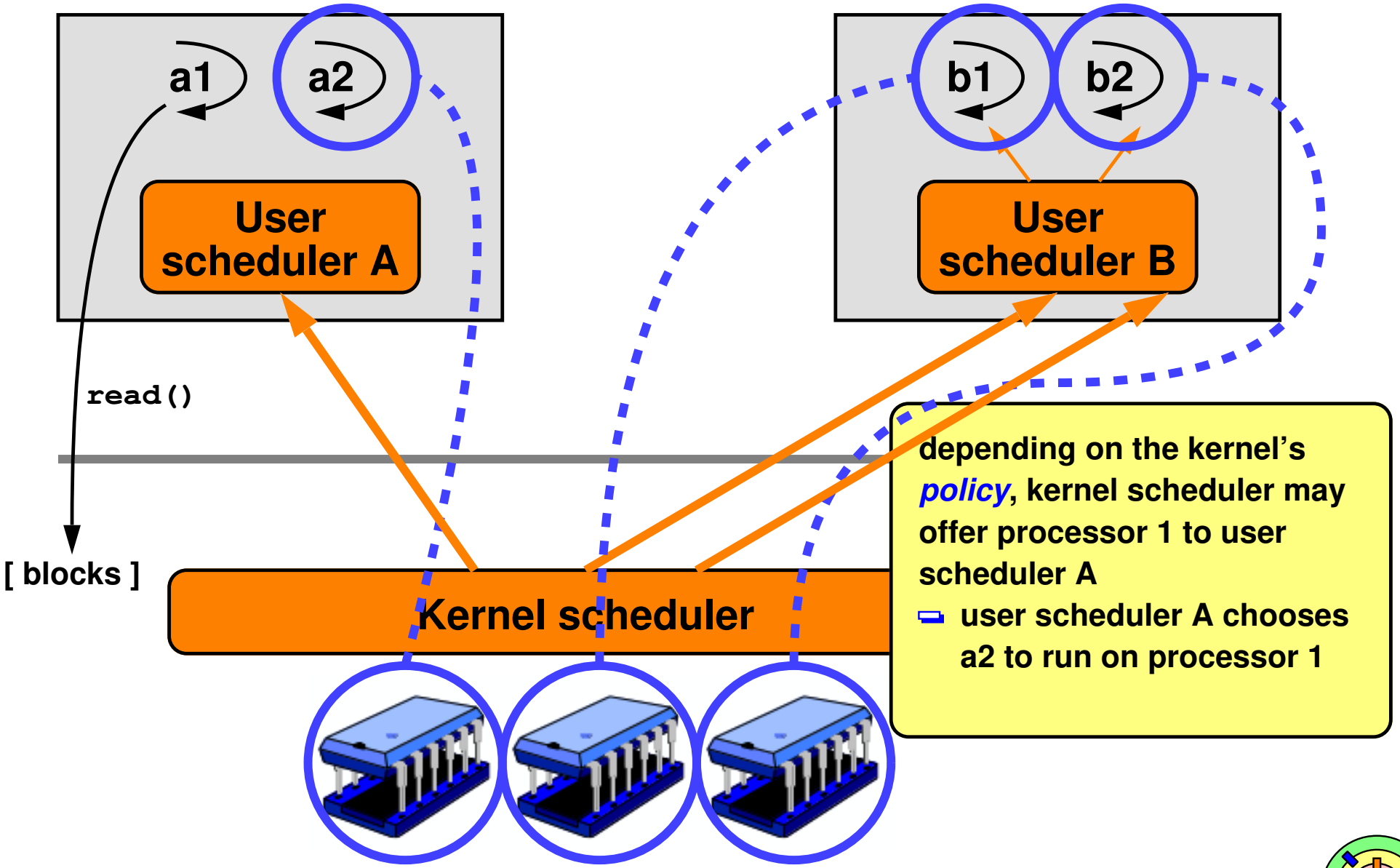
Kernel scheduler does not schedule threads

Scheduler Activations Model Example



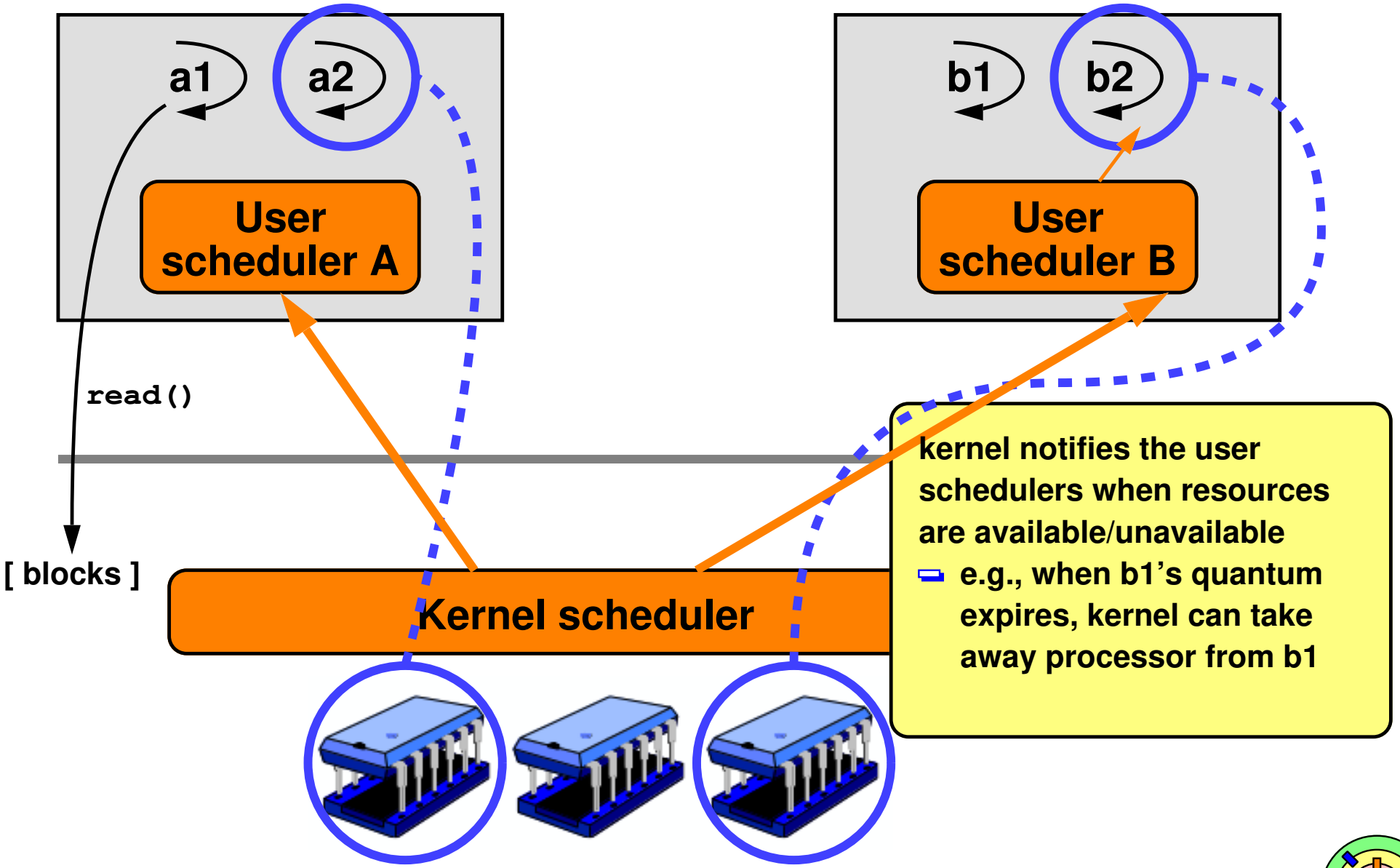
Kernel scheduler can have various scheduling policies

Scheduler Activations Model Example



Kernel scheduler can have various scheduling policies

Scheduler Activations Model Example



Kernel scheduler can have various scheduling policies

Scheduler Activations Model

- Scheduler Activations Model seems like a good solution for two-level models
  - it has not been adopted by a major OS vendor
    - it can take many many years to take something in research and move it into the real world
      - need to conduct extensive experiments to know about all the pluses and minuses of a new approach
      - it may not solve all types of priority inversion problems