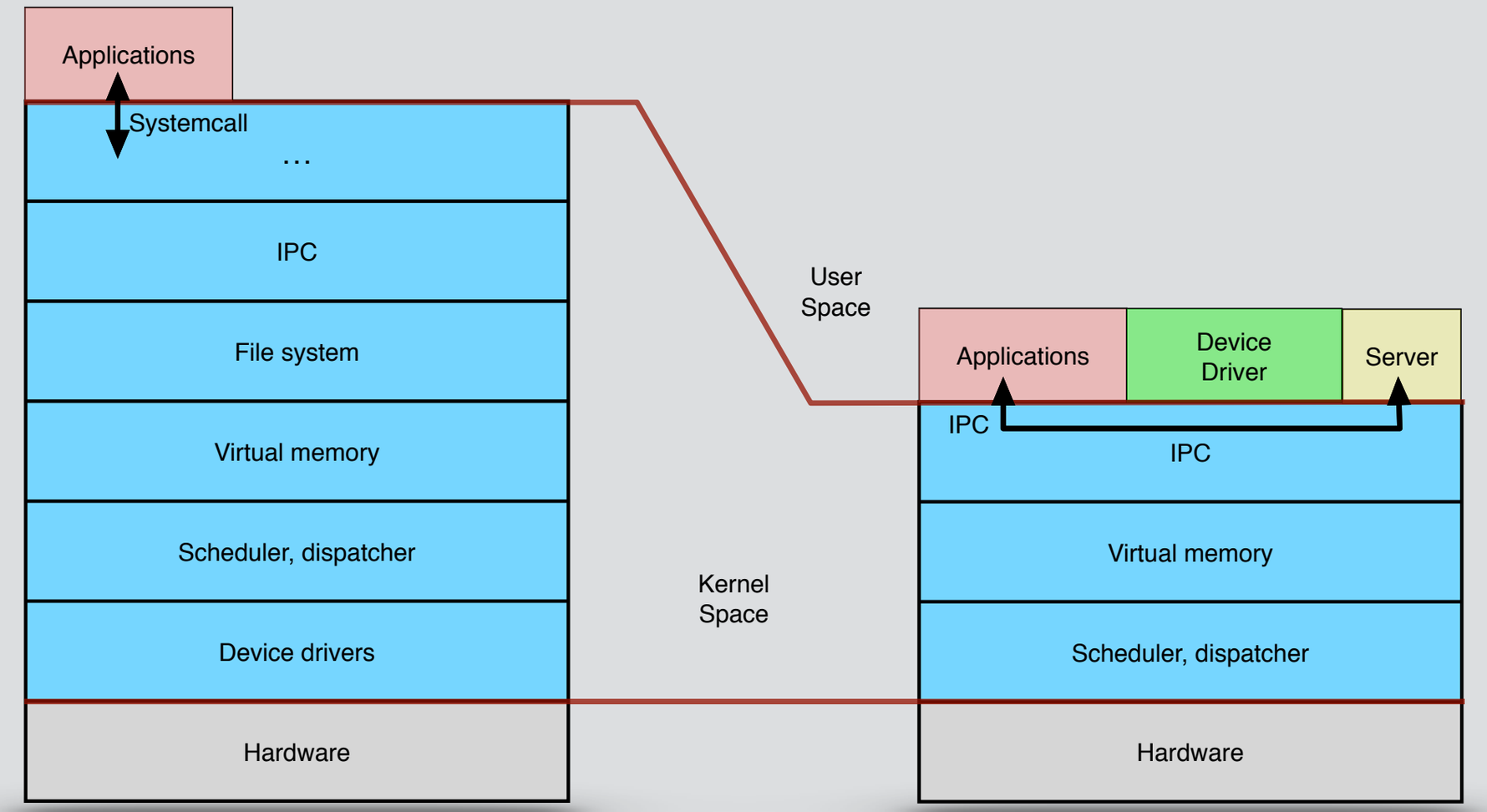
User-level CPU Inheritance Scheduling

Sergej Bomke

6.08.2015

Microkernel

- Minimize the kernel part of the system
- More modularity, flexibility, reliability and trustworthiness.
- Implement the smallest set of functionality



User-Level Scheduling

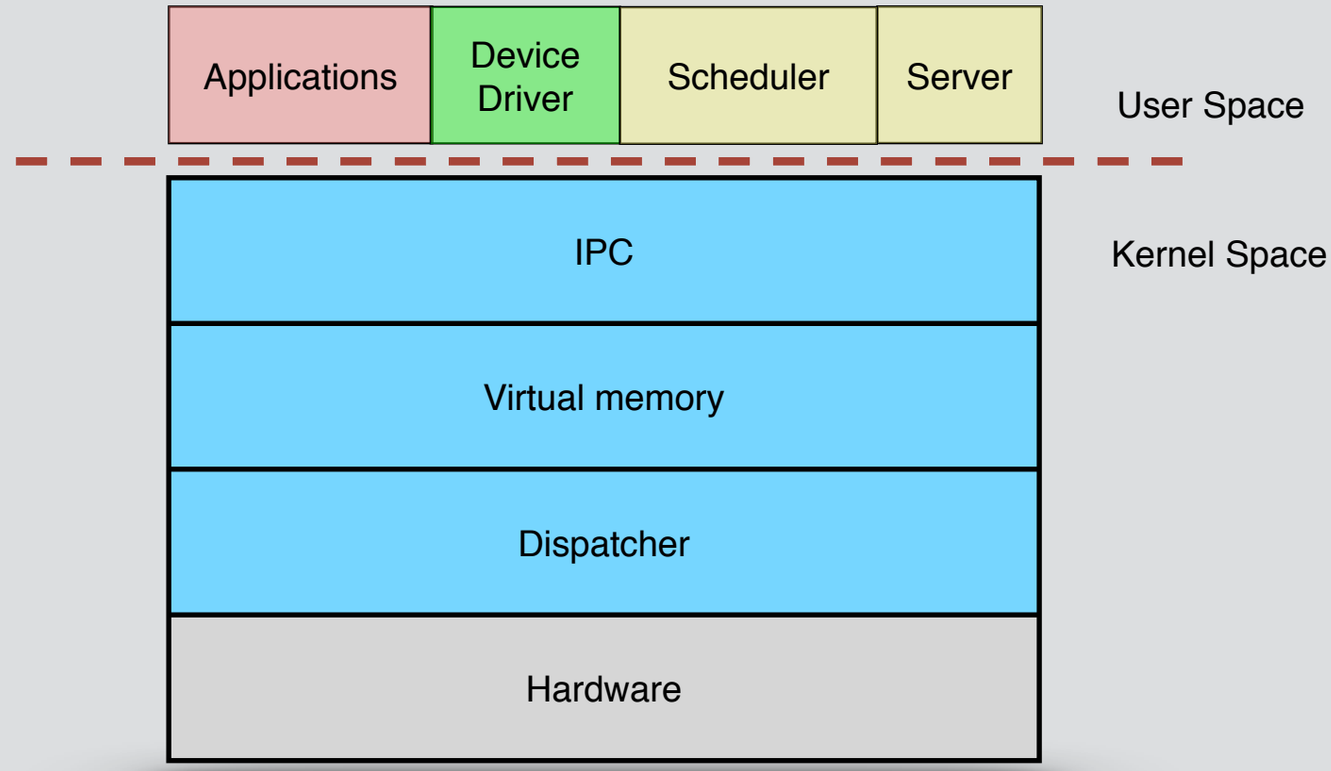
- Export scheduling policies form the kernel
- Makes the system more generic and flexible
- Allows to implement different scheduling policies without any kernel code adjustment
- Allows to build environments with different scheduling requirements

CPU Inheritance Scheduling

- Processor scheduling framework
  - by Bryan Ford and Sai Susarla
- Idee:
  - Applications & OS can implement customize local scheduling policies
  - Threads donate CPU to others threads
  - Threads can act as schedulers for other threads
  - Hierarchy of schedulers

CPU Inheritance Scheduling

- Export scheduler from kernel to the user level.
- Dispatcher stays in kernel space

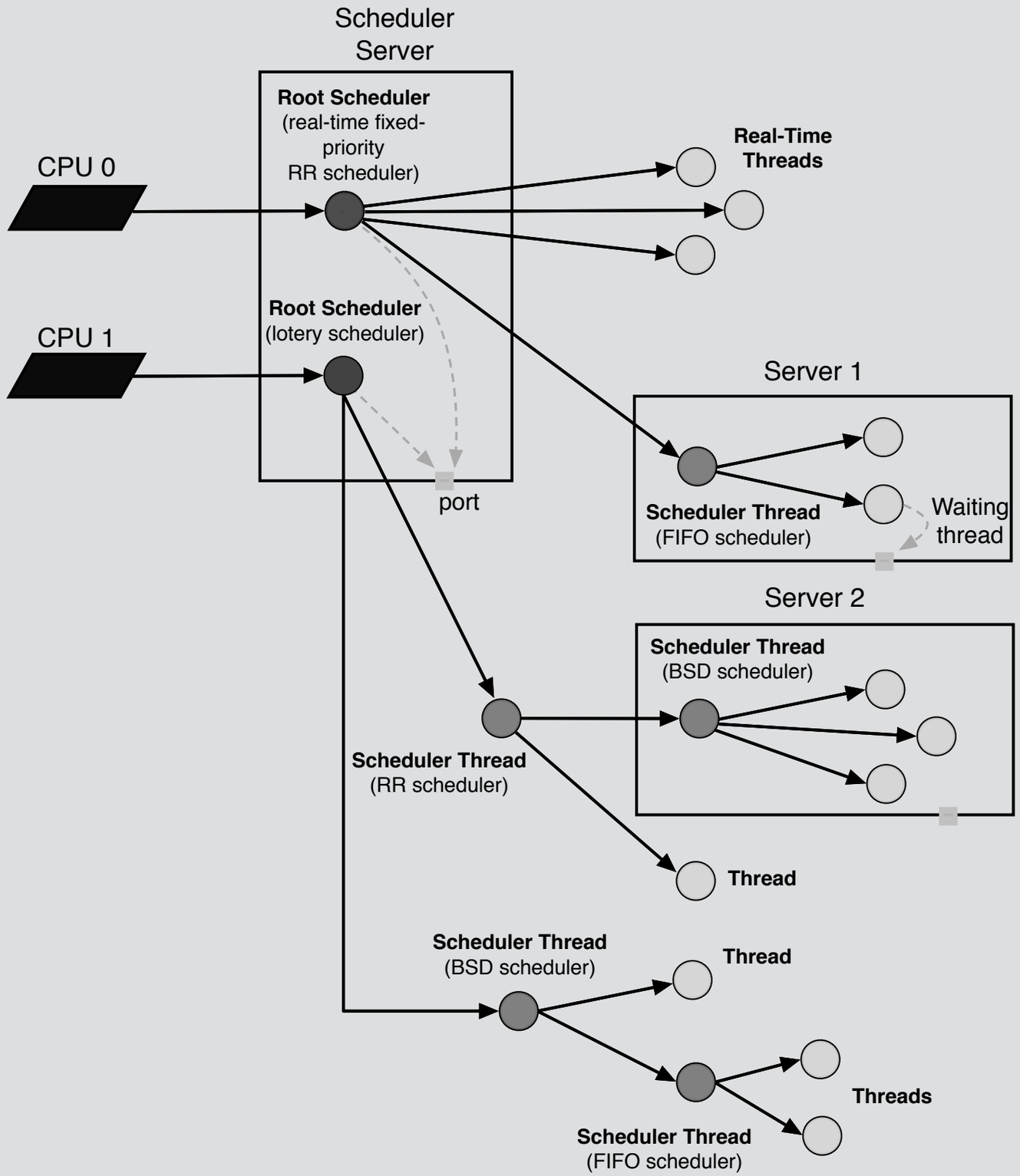


CPU Inheritance: Threads

- Root scheduler thread
  - owns real CPU time
  - donates its available time to other threads
  - one root scheduler thread for each real CPU
- Client thread
  - inherits some CPU resources
- Scheduler thread
  - a client thread
  - has its own clients
  - spends most of its time to donates own CPU resources

CPU Inheritance: Threads

- Logical hierarchy of schedulers
- Each root and scheduler thread can implement different scheduling policies
- Root scheduler of a CPU determines the base scheduler policy for the assigned CPU

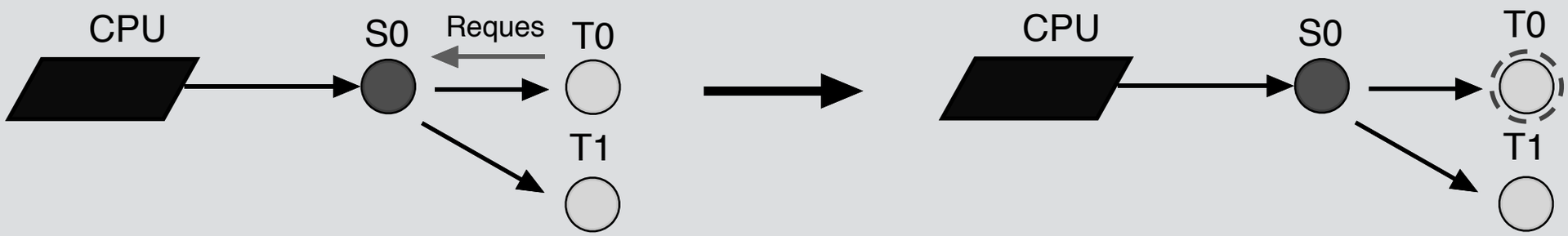


CPU Inheritance: Requesting CPU Time

- No thread can run unless another thread donates some CPU resources to it
  - except the root scheduler thread
- Thread becomes state ready
- Dispatcher notifies the scheduler thread
- Several handling options
  - scheduler thread has some CPU time
  - scheduler thread already donating its CPU time
  - scheduler thread doesn’t have any CPU time left
  - scheduler thread actually preempted

CPU Inheritance: Requesting CPU Time

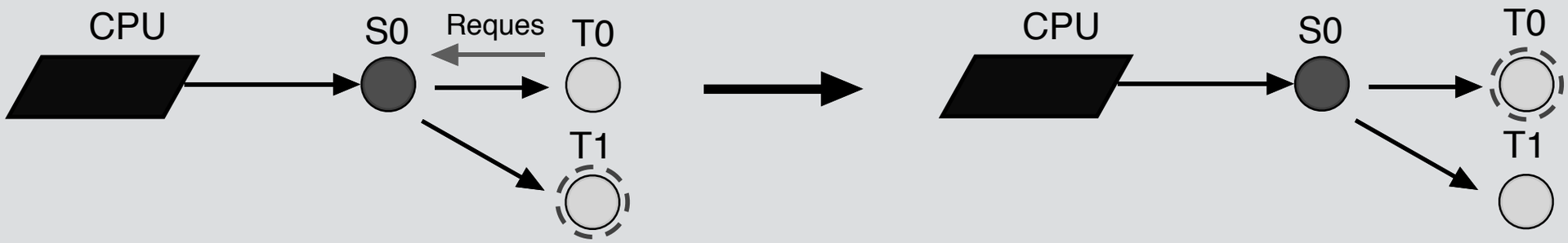
- Scheduler thread has some CPU time
  - scheduler thread donates CPU time to client



- Scheduler thread actually preempted
  - event is irrelevant for scheduling at this moment
  - the dispatcher resumes the currently running thread

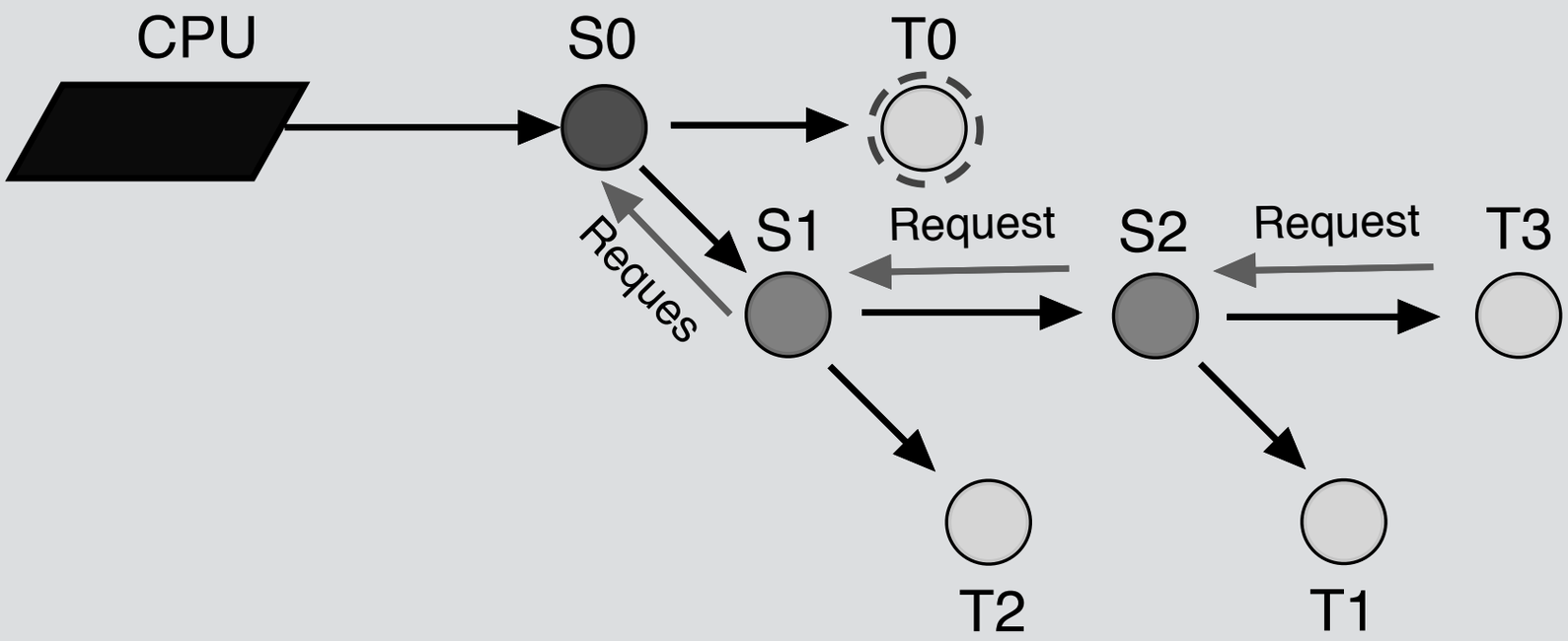
CPU Inheritance: Requesting CPU Time

- Scheduler thread already donating its CPU time
  - currently running thread will be preempted
  - control is given back to scheduler



CPU Inheritance: Requesting CPU Time

- Scheduler thread doesn't have any CPU time left
  - scheduler notifies responsible scheduler
  - leads to a chain where different scheduler threads are woken up



CPU Inheritance: Relinquishing the CPU

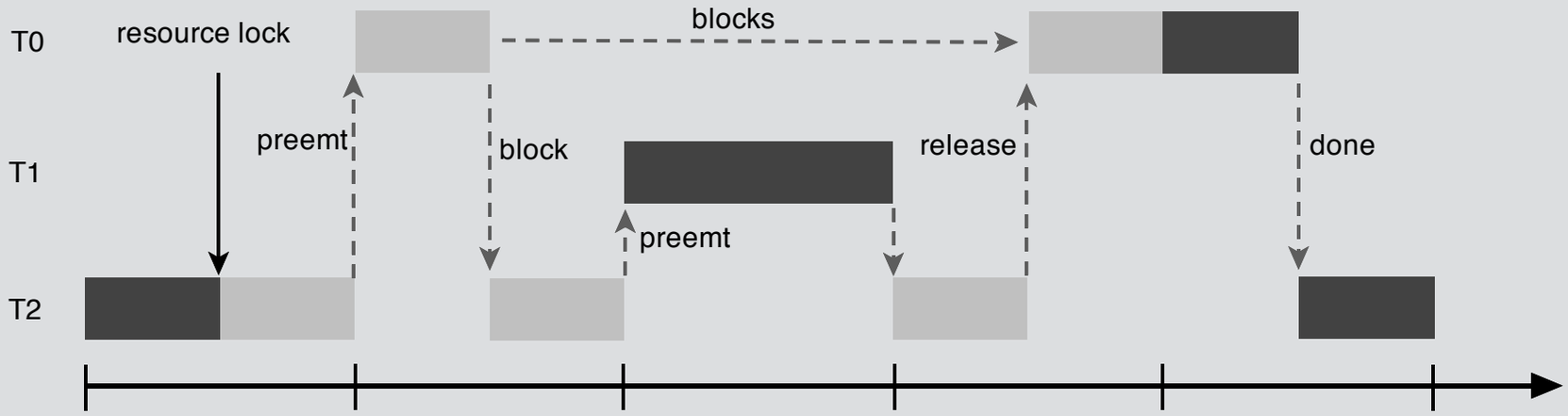
- Running thread may block to wait for an event
- Dispatcher returns control of the CPU to its scheduler thread
- Scheduler has choice
  - donates CPU time to another thread
  - returns control of the CPU to its scheduler

CPU Inheritance: Donation CPU

- Blocked thread donates the rest of its CPU time to another thread
- If an event occurs
  - donation ends
  - CPU is passed to the donator thread
- Possibility to inherit CPU time from more than one source

CPU Inheritance: Priority Inheritance

- High-priority thread is blocked while waiting on a resource that a lower-priority thread holds



- Solution: thread that holds the resource inherits the priority of the blocked thread

CPU Inheritance: Overhead

- Dispatcher costs
  - specifying the thread to switch after an occurred event
  - iteration through trees

Scheduling Hierarchy Depth	Dispatcher Time (μs)
Root scheduling only	8.0
2-level scheduling	11.2
3-level scheduling	14.0
4-level scheduling	16.2
8-level scheduling	24.4

- Context switch costs
  - additional context switches
  - heavily differs from system to system

CPU Inheritance: Problems

- Avalanche Effect
  - consumption of CPU time from multiple donators
- High depth of the scheduling hierarchy
  - can cause a large number of requests
- CPU accounting
  - scheduler threads use virtual time

Conclusion

- Allows to implement different scheduling policies
- Has low overhead
  - context switch costs are low
  - limited depth of the scheduling hierarchy
- Some optimizations are needed

Questions