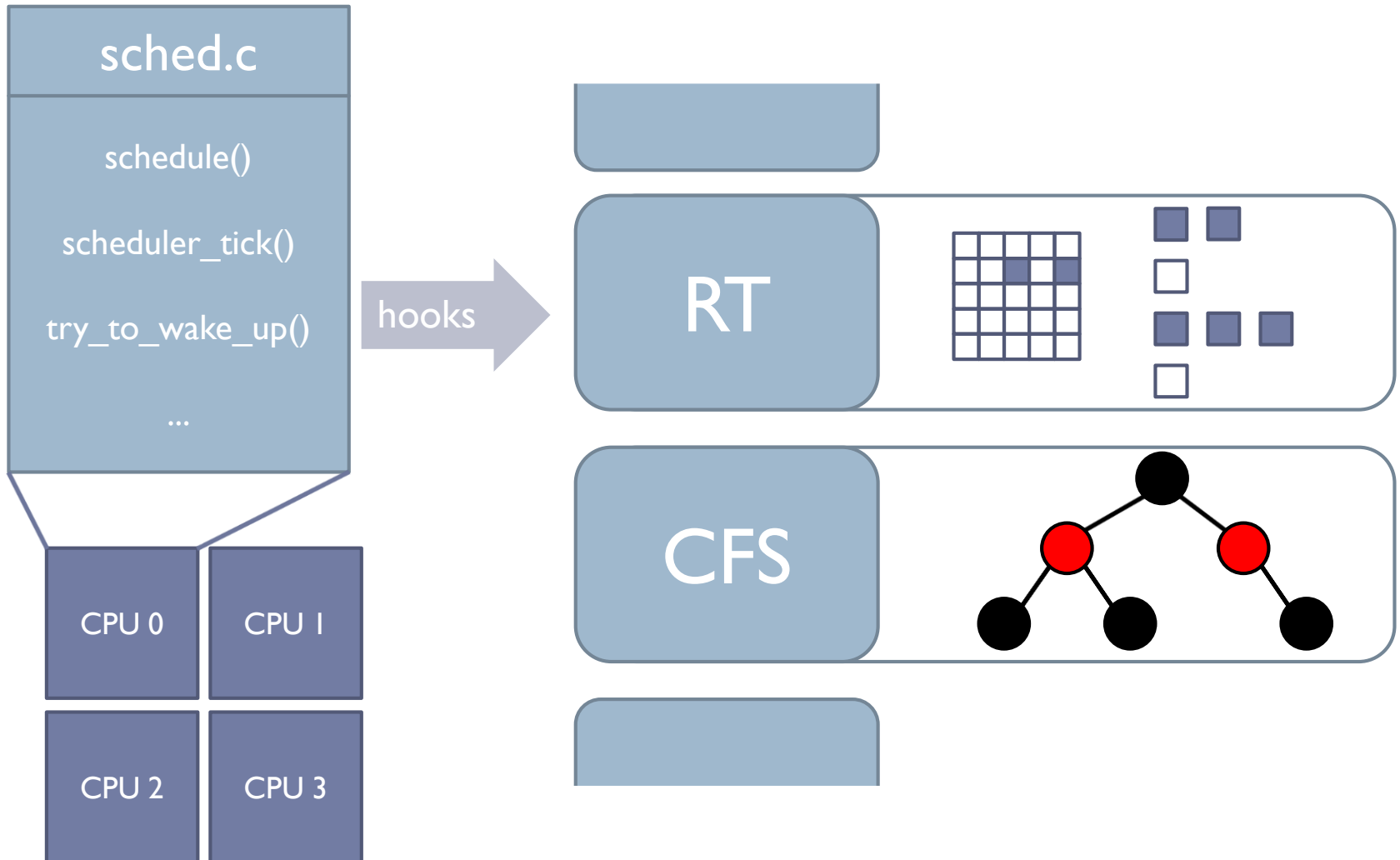
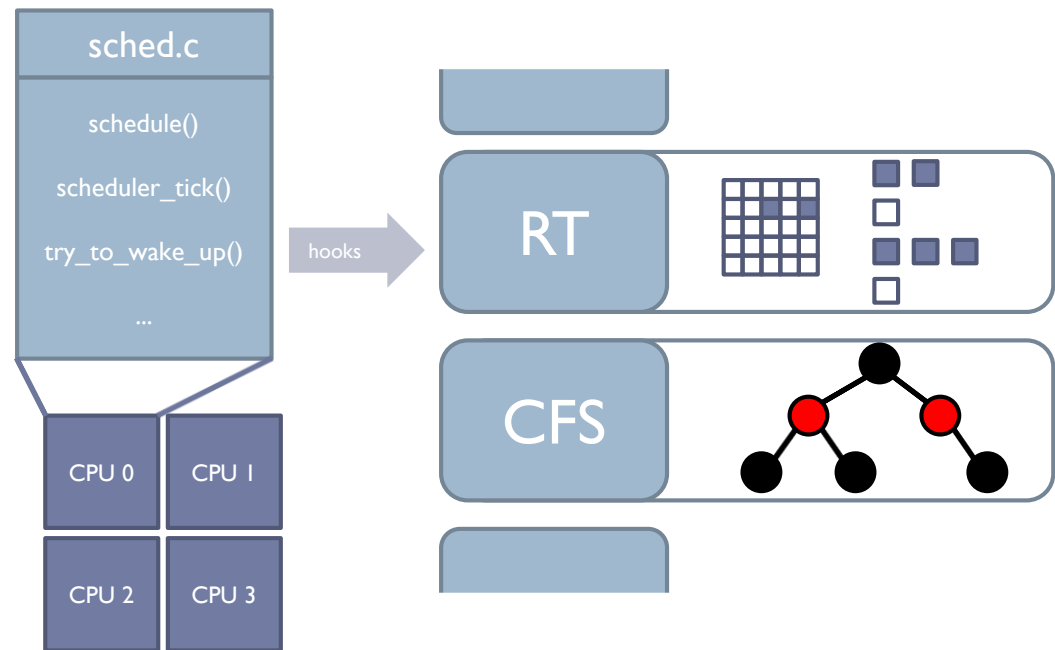


Linux Process Scheduling



Linux Process Scheduling

1. Task Classification
2. Scheduler Skeleton
3. Completely Fair Scheduler (CFS)
4. Real Time Scheduling (RT)
5. Load Balancing CFS
6. Load Balancing RT



1. Task Classification

Task Types

Real Time vs Normal/Other

CPU bound vs I/O bound

- Efficient vs Responsive
- Server vs Desktop vs HPC

1. Task Classification Scheduling Classes

include/linux/sched.h

struct sched_class

const struct sched_class *next

(*enqueue_task)
(*dequeue_task)
(*check_preempt_curr)
(*pick_next_task)
(*select_task_rq)
(*pick_next_task)
...

kernel/sched_rt.c

rt_sched_class

.next = &fair_sched_class

.enqueue_task = &enqueue_task_rt
.dequeue_task = &dequeue_task_rt
...

kernel/sched_fair.c

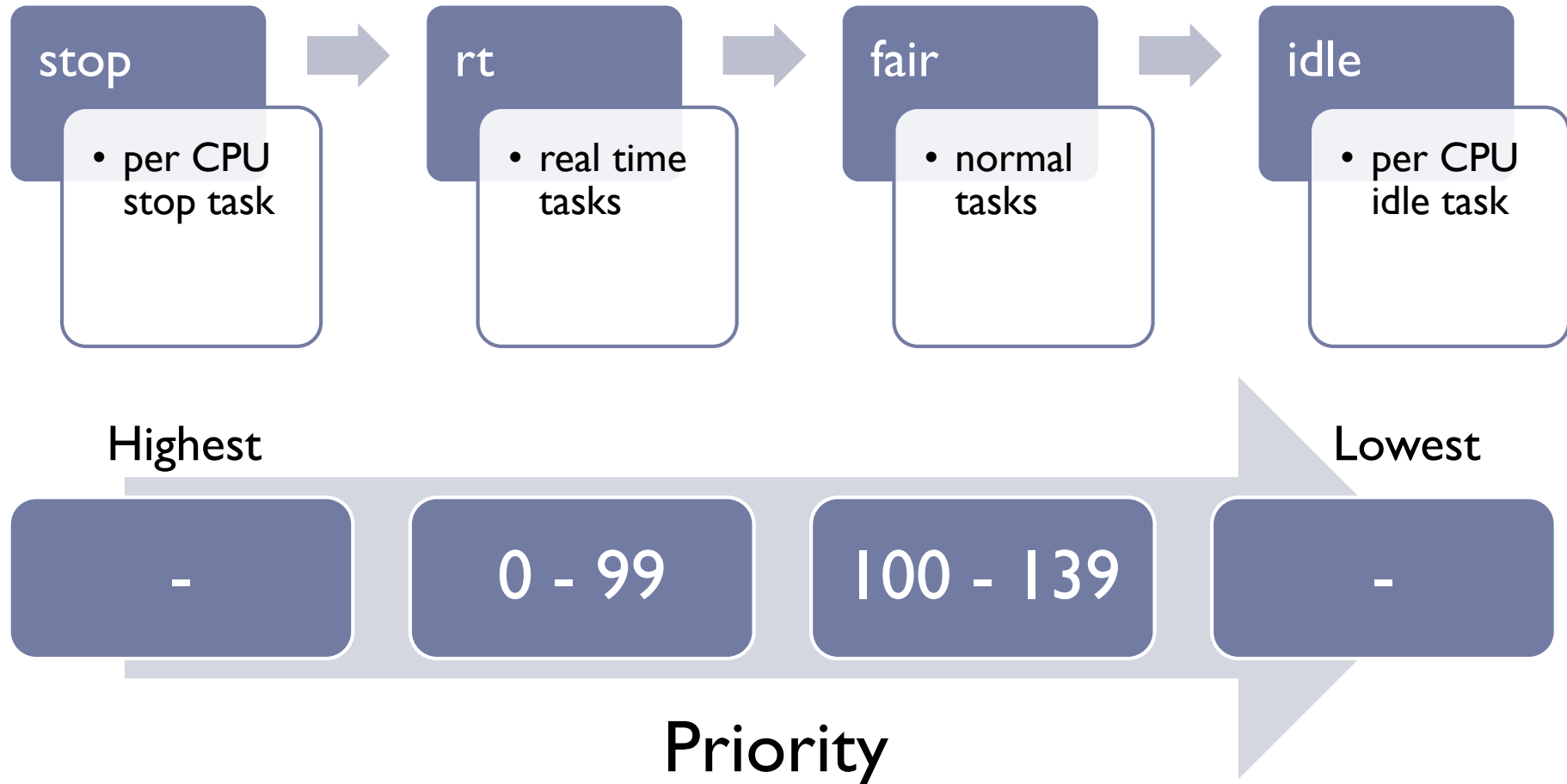
fair_sched_class

.next = &idle_sched_class

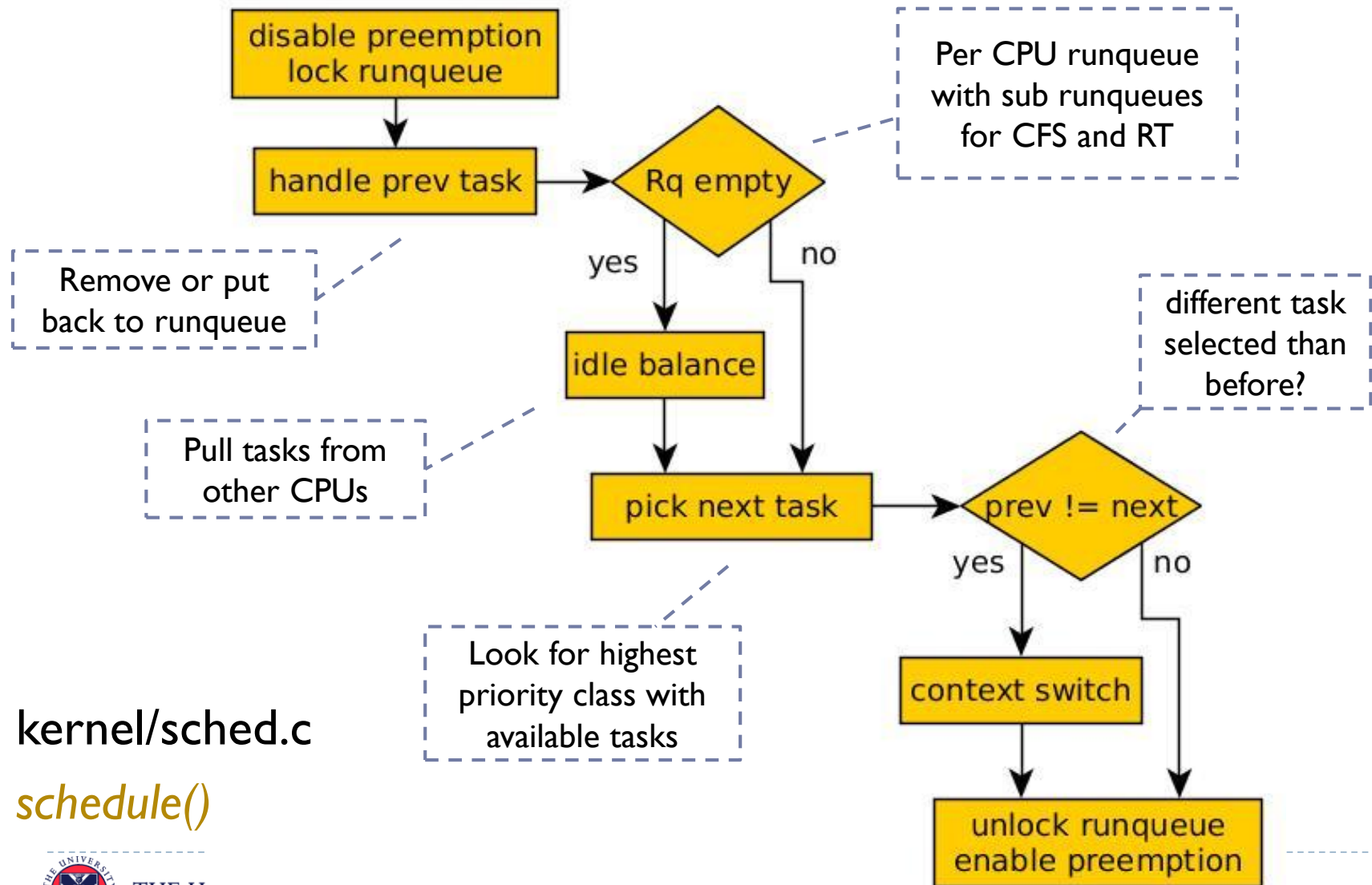
.enqueue_task = &enqueue_task_fair
.dequeue_task = &dequeue_task_fair
...

1. Task Classification

Scheduling Class Priorities



2. Scheduler Skeleton Scheduler Entry Point



kernel/sched.c

schedule()

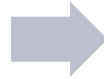
2. Scheduler Skeleton Calling the Scheduler

1. Timer Interrupt
2. Currently running task goes to sleep
3. Sleeping task wakes up

2. Scheduler Skeleton Calling the Scheduler - Timer

kernel/sched.c

```
scheduler_tick() {  
    ...  
    sched_class->task_tick()  
    ...  
}
```



kernel/sched_X.c

X_sched_class

```
task_tick_X() {
```

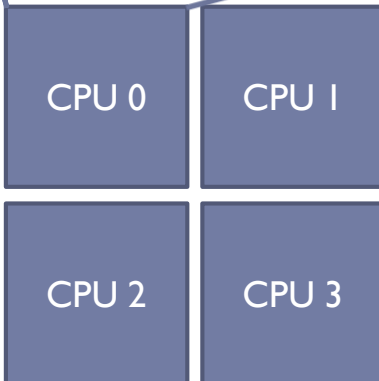
update task's
scheduling entity

is it
another
task's turn
yet?

Yes

Invoke scheduling

```
}
```



2. Scheduler Skeleton

Calling the Scheduler - Sleep

```
1  /* 'q' is the wait queue we wish to sleep on */
2  DEFINE_WAIT(wait);
3
4  add_wait_queue(q, &wait);
5  while (!condition) { /* condition is the event that we are waiting for */
6      prepare_to_wait(&q, &wait, TASK_INTERRUPTIBLE);
7      if (signal_pending(current))
8          /* handle signal */
9          schedule();
10 }
11 finish_wait(&q, &wait);
```

2. Scheduler Skeleton

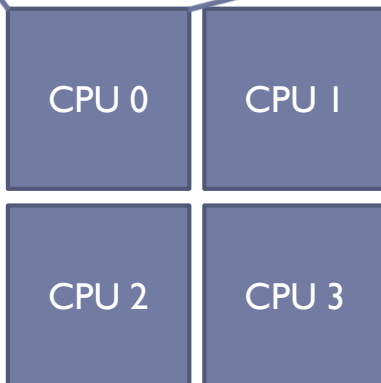
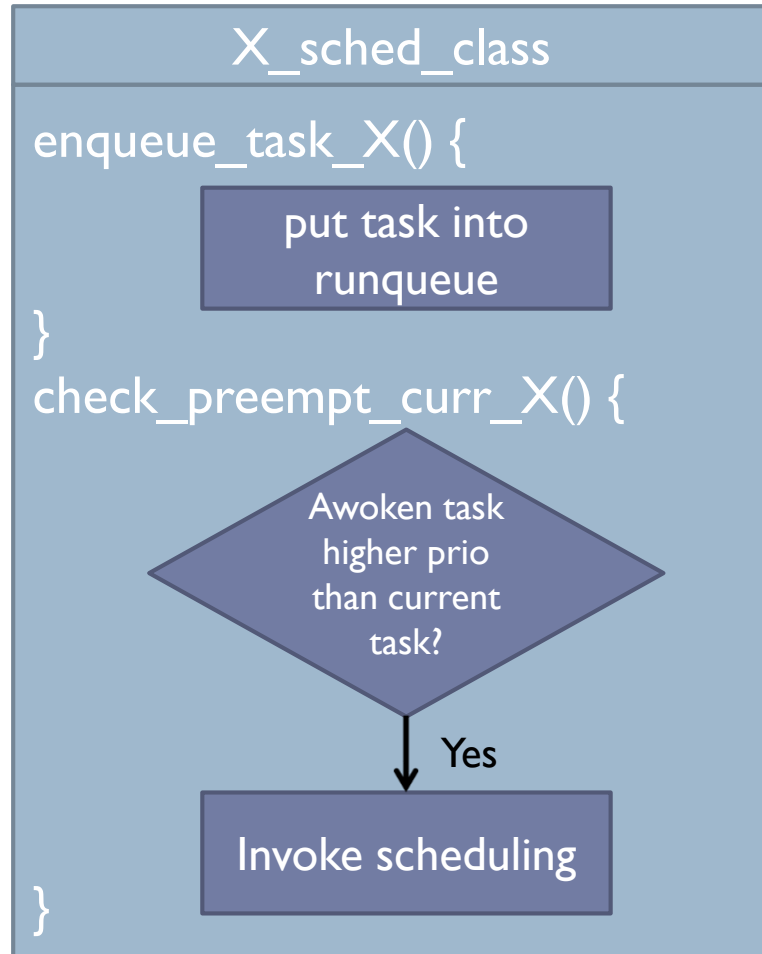
Calling the Scheduler – Wake up

kernel/sched.c

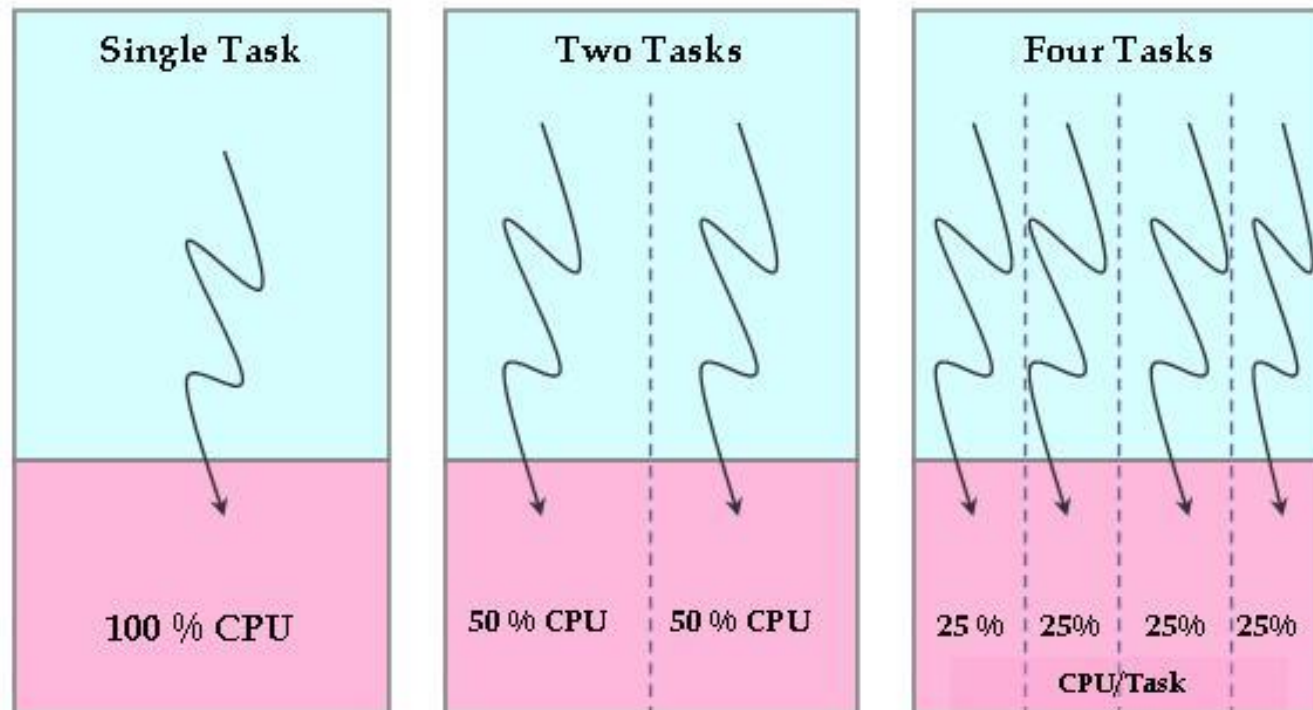
```
try_to_wake_up() {  
    ...  
    sched_class->enqueue_task()  
  
    sched_class->check_preempt_curr()  
  
    task.state = TASK_RUNNING  
    ...  
}
```



kernel/sched_X.c



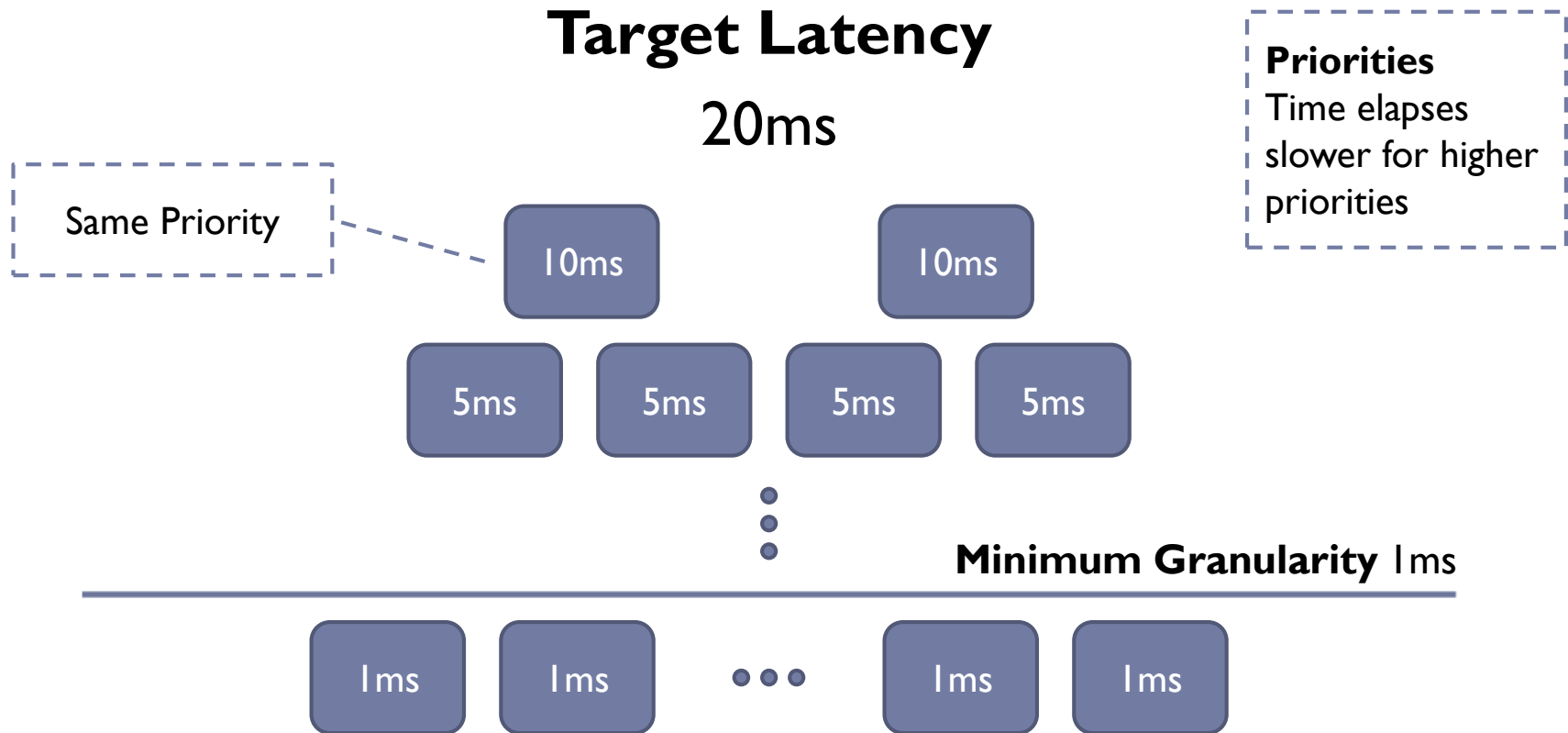
3. Completely Fair Scheduler Concept



Ideal Precise Multi-tasking CPU - Each task runs in parallel and consumes equal CPU share

Virtual Runtime CFS tries to maintain an equal virtual runtime for each task in a CPU's runqueue at all time.

3. Completely Fair Scheduler Virtual Runtime



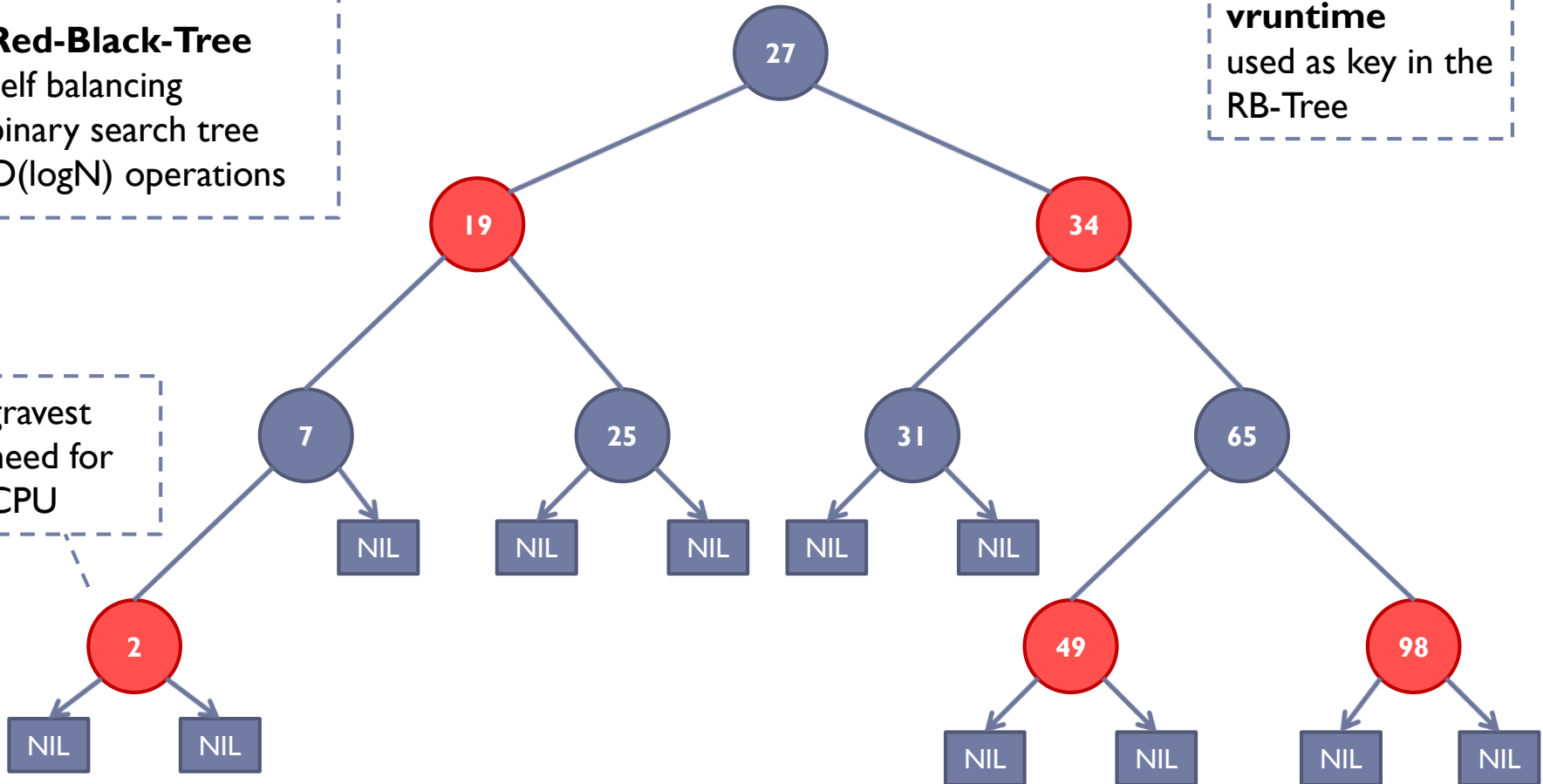
- How does it work out for I/O bound and CPU bound Tasks?

3. Completely Fair Scheduler Runqueue - Red-Black-Tree

Red-Black-Tree
self balancing
binary search tree
 $O(\log N)$ operations

vruntime
used as key in the
RB-Tree

gravest
need for
CPU

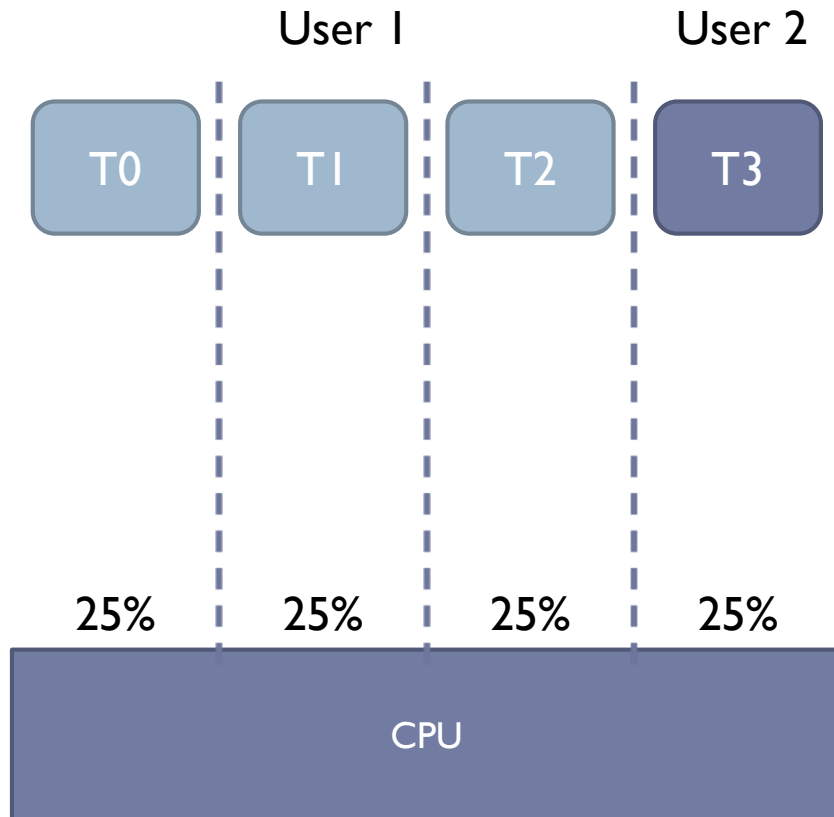


virtual runtime

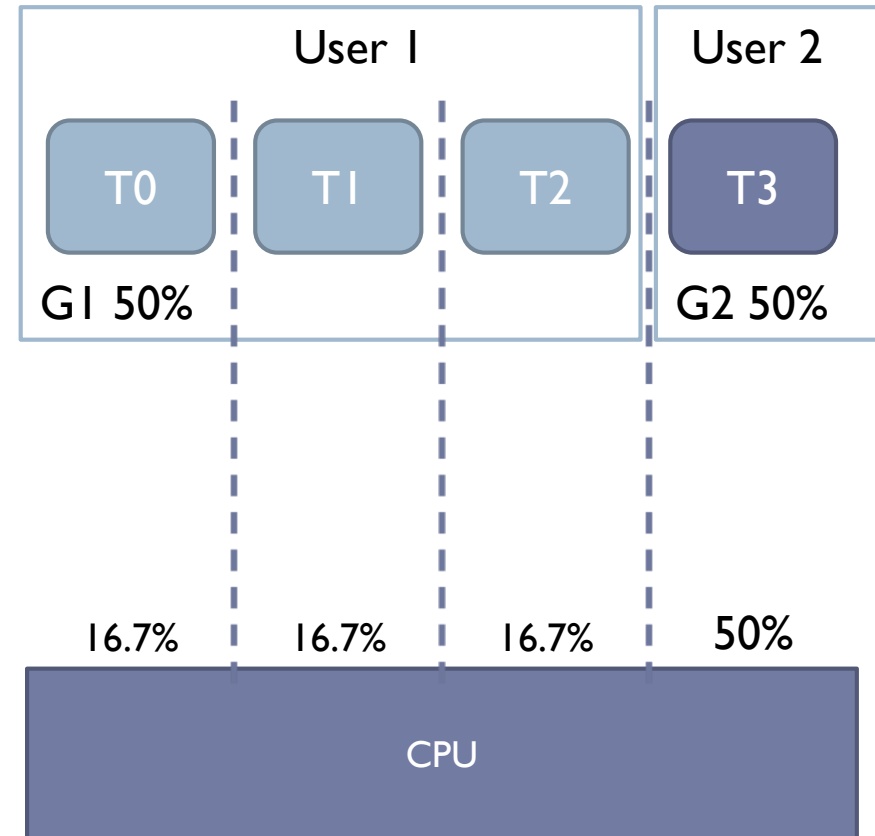
3. Completely Fair Scheduler

Fair Group Scheduling

Without Group Scheduling



With Group Scheduling



First, be fair to groups – then, be fair to tasks.

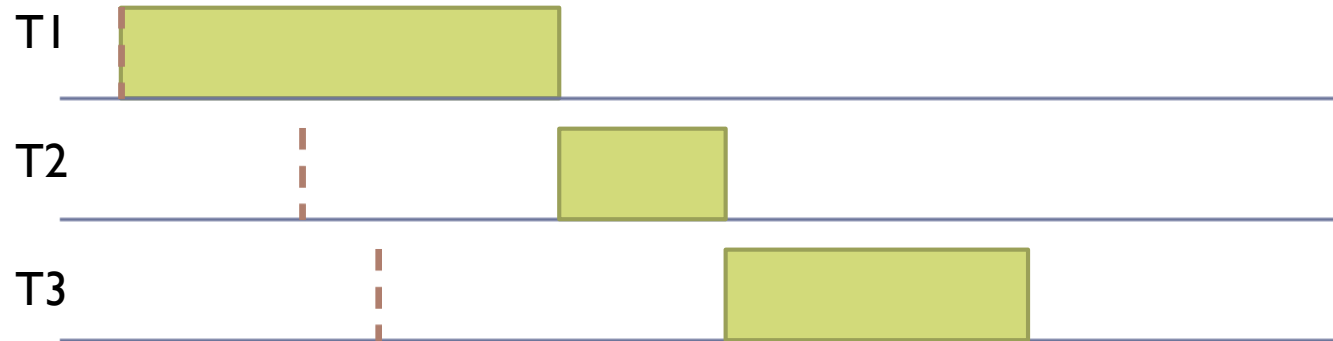
4. Soft-Real-Time Scheduling Scheduling Modes

Scheduling of tasks with strict timing requirements.

RT scheduling is reliable but kernel does not guarantee that deadlines will be met.

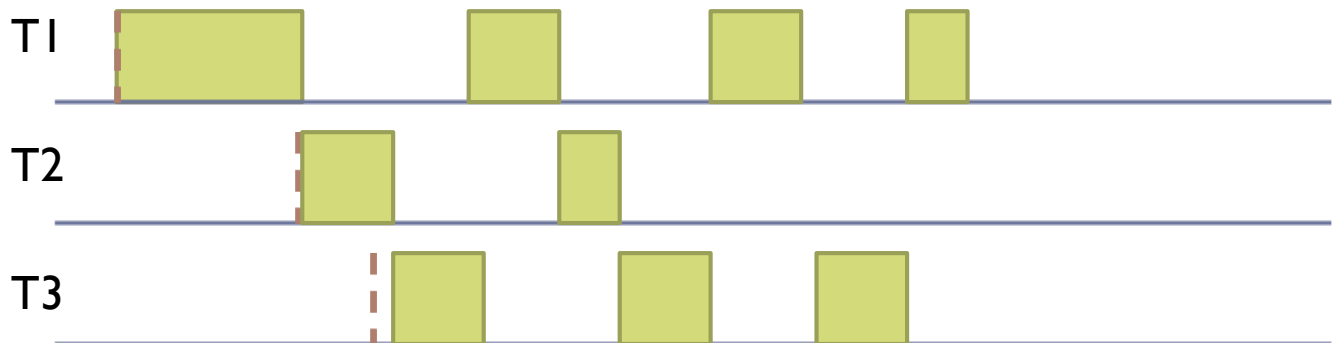
SCHED_FIFO

no time slices
until termination
or yielding CPU



SCHED_RR

fixed time slice
round robin
scheduling

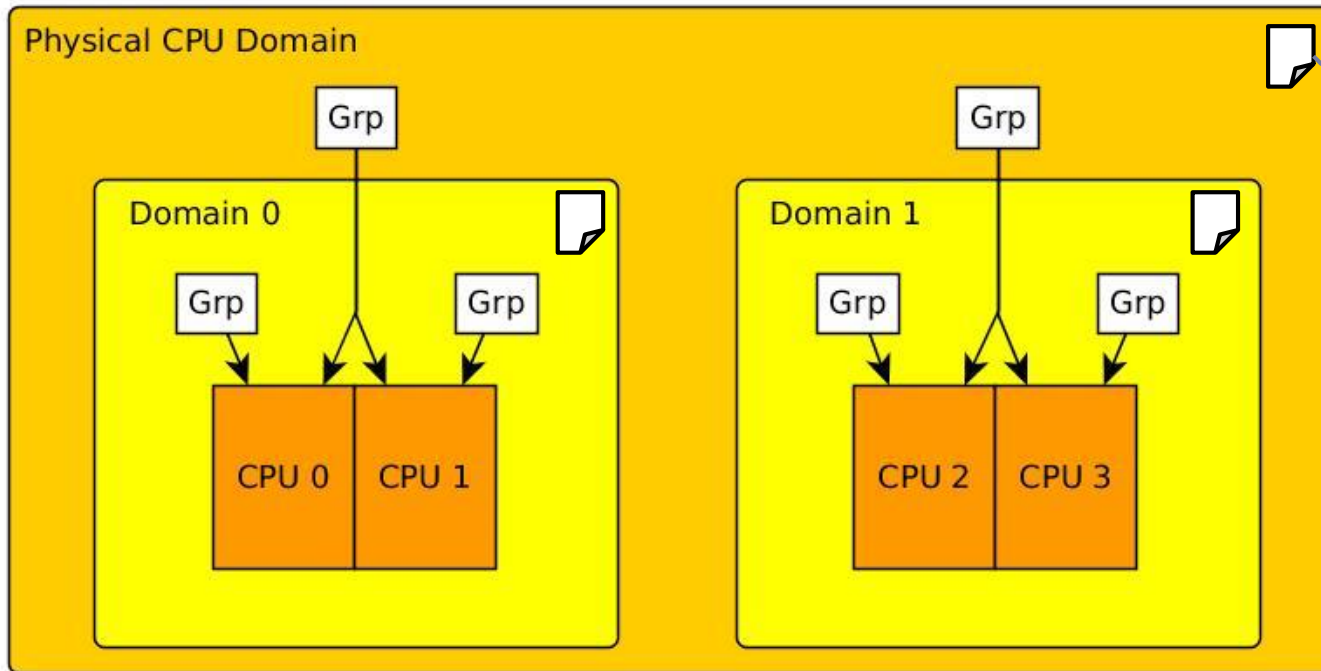


criticalblue
Accelerating Embedded Software



5. Load Balancing CFS Scheduling Domains

CFS load balancing is used to offload task from busy CPUs to less busy or idle ones on SMP Systems.



Domain Specific Balancing Policy

- how often to do balancing
- how far to move tasks
- how long before cache cools down
- ...

Scheduling Domains

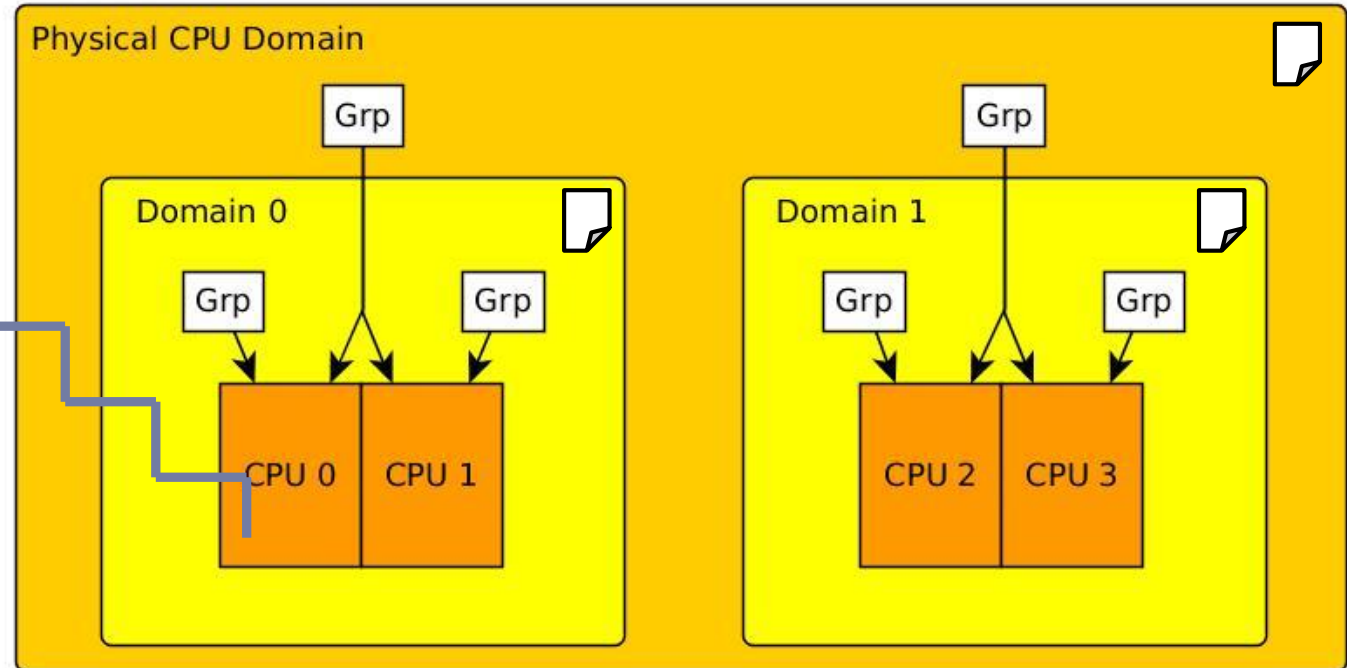
Handle the topology variety of modern processor.

5. Load Balancing CFS

Active and Idle Balancing

kernel/sched_fair.c

Balancing
walks up the
domain hierarchy
and looks for the
busiest group



Active Balancing

regularly by `scheduler_tick()`
pulls tasks over from busiest group
per domain

Idle Balancing

as soon as CPU goes idle in `schedule()`
checks if average idle time is larger than
migration cost
pulling tasks like active balancing

5. Load Balancing CFS

Where To Put a New Task

kernel/sched.c

```
select_task_rq(flag) {  
    ...  
    sched_class->select_task_rq(flag)  
    ...  
}
```

kernel/sched_fair.c

fair_sched_class

```
select_task_rq_fair(flag) {  
    ...  
}
```



Returns optimal
CPU to put the
task on

Looks for idlest CPU
considering only those
domains that have *flag*
set in their balancing
policy

SD_BALANCE_EXEC

used in *sched_exec()* upon starting a new task with *exec()*

SD_BALANCE_FORK

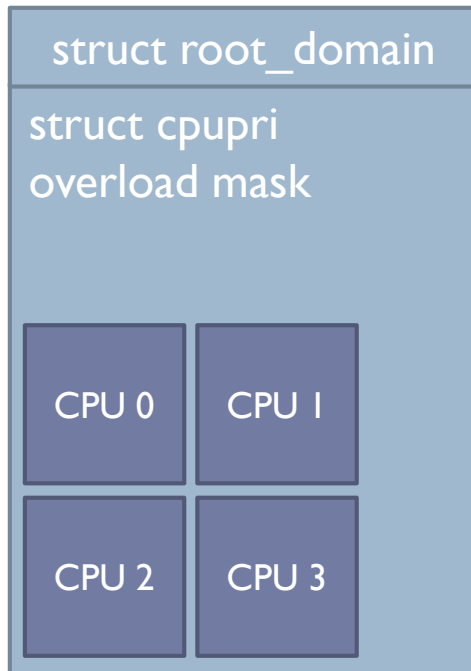
used in *wake_up_new_task()* upon a fork command

SD_BALANCE_WAKE

used in *try_to_wake_up()* upon waking up a task that already ran

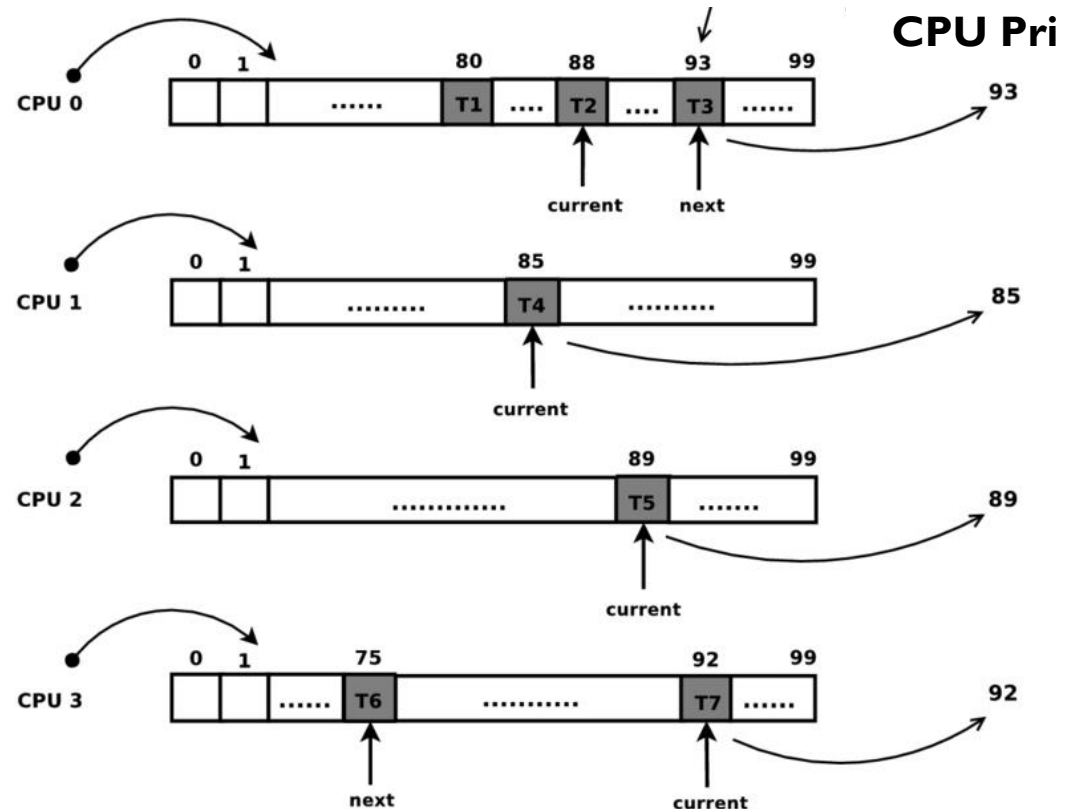
6. Load Balancing RT Root Domains and CPU Pri

RT load balancing aims to make sure that the N highest priority tasks on the system are running at all time where N is the number of CPUs.



Root Domain

scope for RT scheduling decisions
overall overload and priority state



6. Load Balancing RT Push

A Low Priority Task is Pushed to Another CPU If

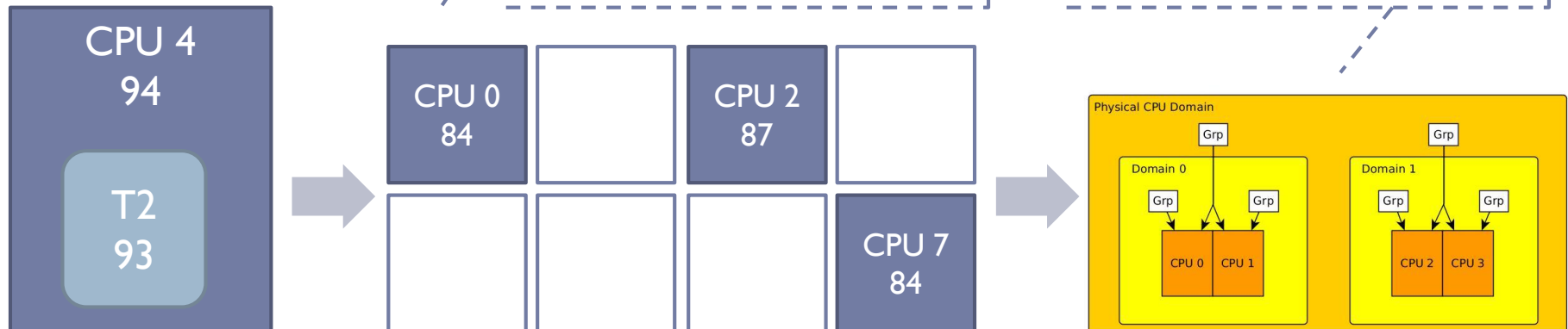
post_schedule()

- lower prio task wakes up on CPU with higher prio task running
- higher prio task wakes up on CPU and preempts lower prio task

T2 needs to be pushed away

get CPU Lowest Mask
represents the lowest priority tasks in the system

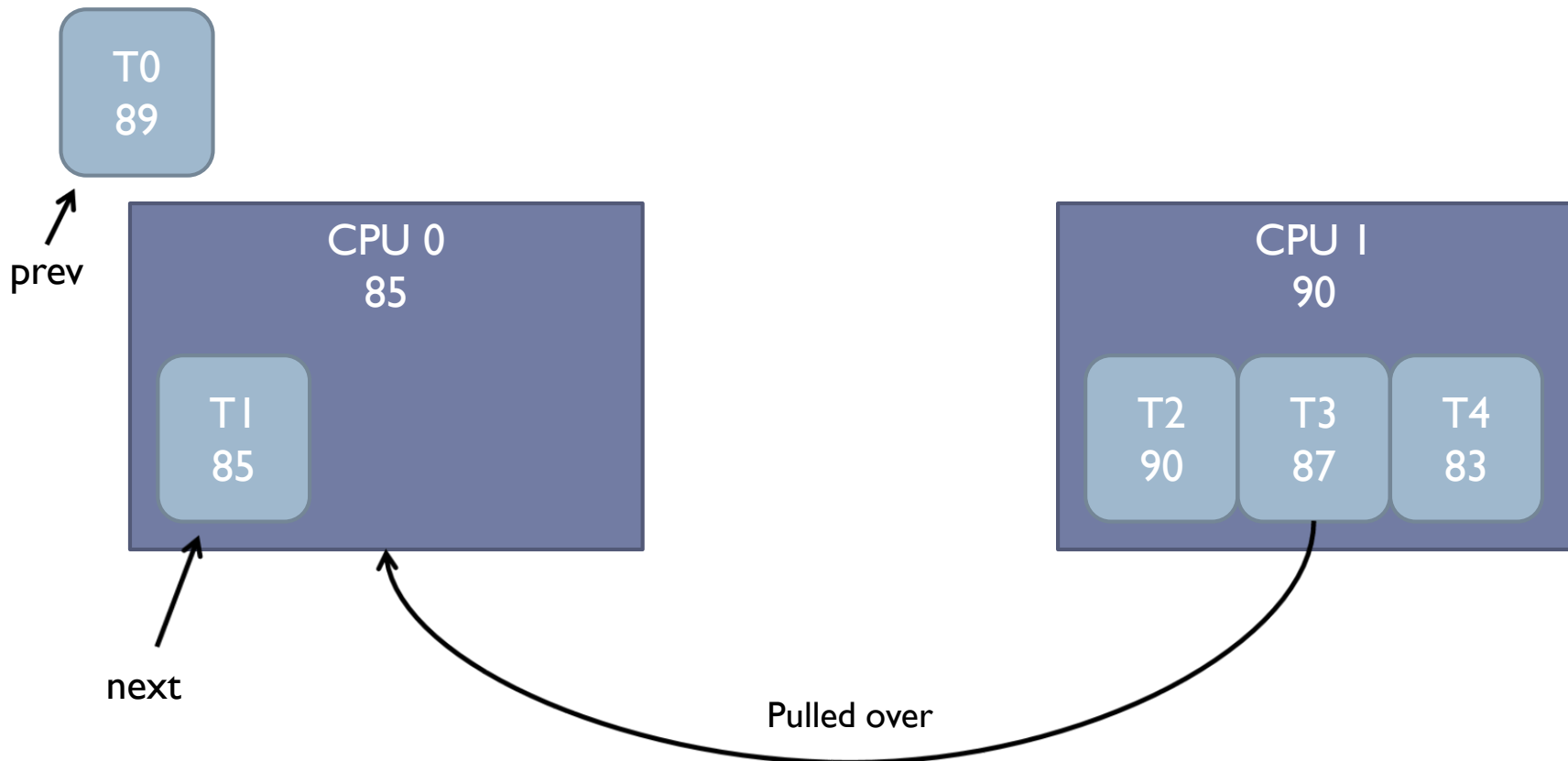
Elect best target CPU based on cache affinity and topology



6. Load Balancing RT Pull

A High Priority Task is Pulled from Another CPU If *pre_schedule()*

- priority of task to be scheduled would be lower than previously one's



Linux Process Scheduling

Thanks, Questions?

