CFS 스케줄러

국민대학교 임베디드 연구실
경 주 현

Outline

- CFS scheduler
- CFS Load balancer

Linux scheduler history

- **Linux 2.4: global queue, O(N)**
  - Simple
  - Poor performance on multiprocessor/core
  - Poor performance when n is large

- **Linux 2.5: O(1) scheduler, per-CPU run queue**
  - Solves performance problems in the old scheduler
  - Complex, error prone logic to boost interactivity

O(1)

- **The Linux scheduler was overhauled completely with the release of kernel 2.6**
- **O(1) scheduler relies on active and expired arrays of processes**
  - To achieve constant scheduling time
- **Problem**

O(1)

- **The Linux scheduler was overhauled completely with the release of kernel 2.6**
- **Time slice == nice value**
- **To achieve constant scheduling time**
- **Problem**
  - No guarantee of fairness
  - the complex heuristics
    - **interactive or non-interactive**

Linux scheduler implementations

- **Linux 2.6~4.x: completely fair scheduler (CFS)**
  - Fair
  - Naturally boosts interactivity
- **CFS scheduler defines a fixed time interval during which each thread in the system must run at least once.**

Scheduler Policy

- **The key decisions made in the scheduler are :**

*"how to determine a thread's **timeslice**? and how to pick the **next thread** to run"*

- **Previous studies : FIFO, Round Robin**
  - Problem : Starvation and unfair

On a single-CPU system

- CFS is very **simple**.
- An implementation of the weighted fair queueing(WFQ)
- CFS scheduler defines a fixed time interval during which each thread in the system **must run at least once.**
- The interval is divided among **thread's weights.**
  - We call the *time slice*.
  - Time slice <- period / weight's rate

"how to determine a thread's ***timeslice****? and how to pick the* ***next thread*** *to run"*

CFS's Time slice

- **Completely Fair Scheduler (CFS)**
- **Provide each task CPU time proportional to its weight**

time slice = (Weight of task / Total weight) x period

Task Weight

- **priority number must be mapped**
  - kernel by a number between 100 and 139
- **A priority number of 120(nice value 0) = 1024 load**
  - see prio_to_weight table

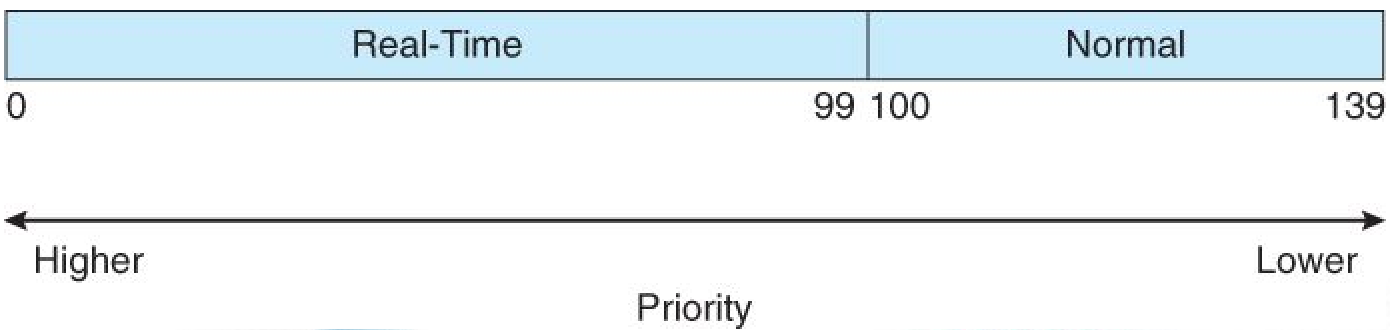
```
struct load_weight {
     unsigned long weight;
 }; // found in struct sched_entity
```

```
static const int prio_to_weight[40] = {
 /* -20 */  88761,    71755,    56483,    46273,    36291,
 /* -15 */  29154,    23254,    18705,    14949,    11916,
 /* -10 */  9548,     7620,     6100,     4904,     3906,
 /*  -5 */  3121,     2501,     1991,     1586,     1277,
 /*   0 */  1024,     820,      655,      526,      423,
 /*   5 */  335,      272,      215,      172,      137,
 /*  10 */       110,       87,       70,       56,       45,
 /*  15 */       36,        29,       23,       18,   15,
};
```

Nice Values and Task Priority

- **Non-real-time priority**
  - Nice value (-20~19, default 0)
  - A large nice value corresponds to a lower priority

- **Real-time priority**
  - Priority range: 0~99
  - Priority for real-time tasks
    - SCHED_FIFO, SCHED_RR
  - A smaller value corresponds to a higher priority



https://www2.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/6_CPU_Scheduling.html

Task Weight Example

- **A priority: 120, normal task, load: 1024**
- **A and B, running at a priority of 120**
- **Available CPU time**
  - 1024 / (1024*2) = 0.5
- **Increased by one level to 121**
  - 820 ((1024/1.25))
  - Task A : 820/(1024+820)) = ~0.45
  - Task B : (1024/(1024+820)) = ~0.55
- **10% decrease in the CPU time share for Task A.**

Time slice

- **se.load.weight**
  - weight of scheduling entity

- **cfs_rq.load.weight**
  - accumulation of weights of all tasks on its run queue

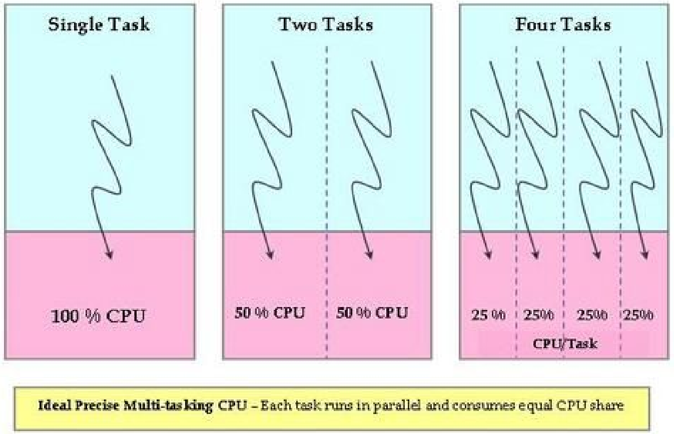
```
time_slice = (se.load.weight) / cfs_rq.load.weight) * sched_period();
```

sched_slice()

kernel/sched/fair.c

CFS problem

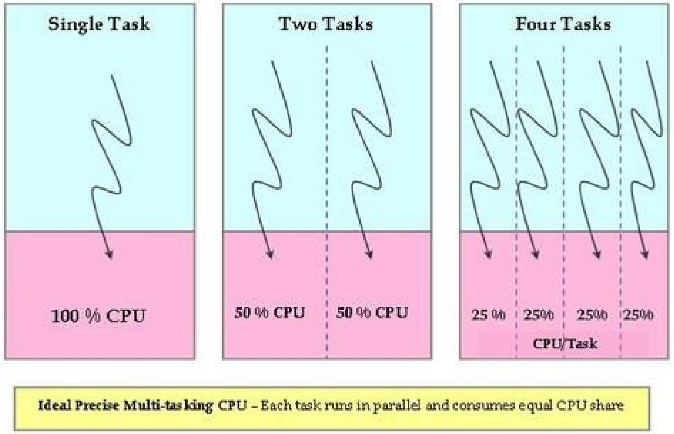
- **Real-world**
  - CFS basically models an “ideal, precise multitasking CPU” on real hardware.”
  - But in real-world, ideal CPU is nonexistent.
  - Time-sharing system



http://www.linuxjournal.com/magazine/completely-fair-scheduler?page=0,0

CFS problem

- **Real-world**
  - CFS basically models an “ideal, precise multitasking CPU” on real hardware.”
  - But in real-world, ideal CPU is nonexistent.



http://www.linuxjournal.com/magazine/completely-fair-scheduler?page=0,0

- **Solution**
  - Virtual Runtime

vruntime

- **How long a process has run**

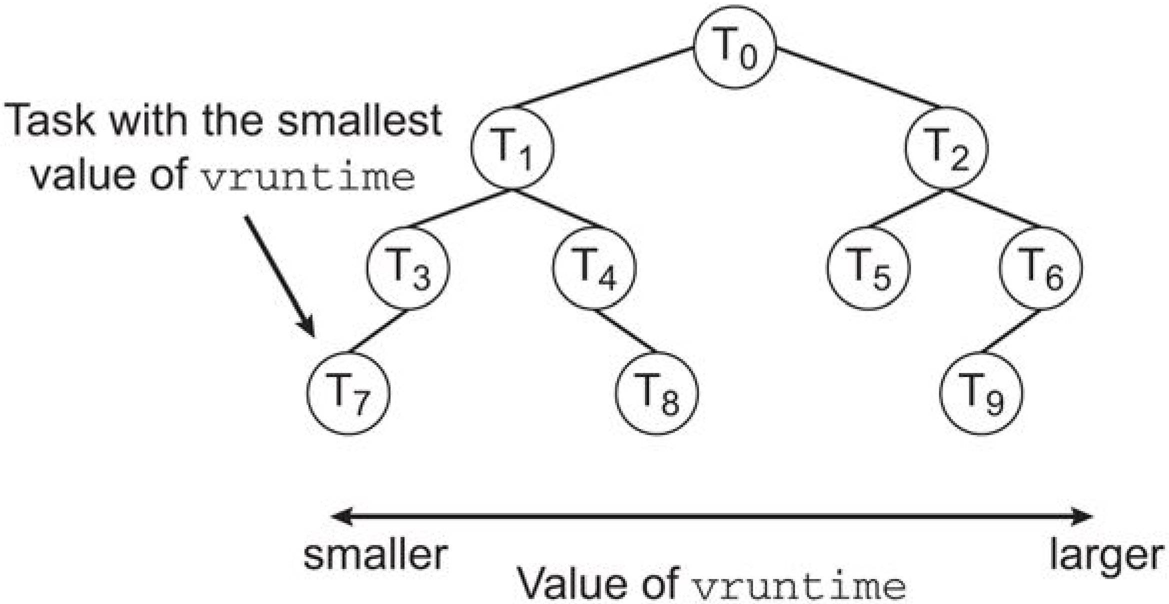
vruntime +=
delta_exec * (NICE_0_LOAD / curr->load.weight);

- **delta_exec**
  - The time spent by the task since the last time vruntime was updated.

- **CFS scheduler defines a fixed time interval during which each thread in the system must run at least once.**

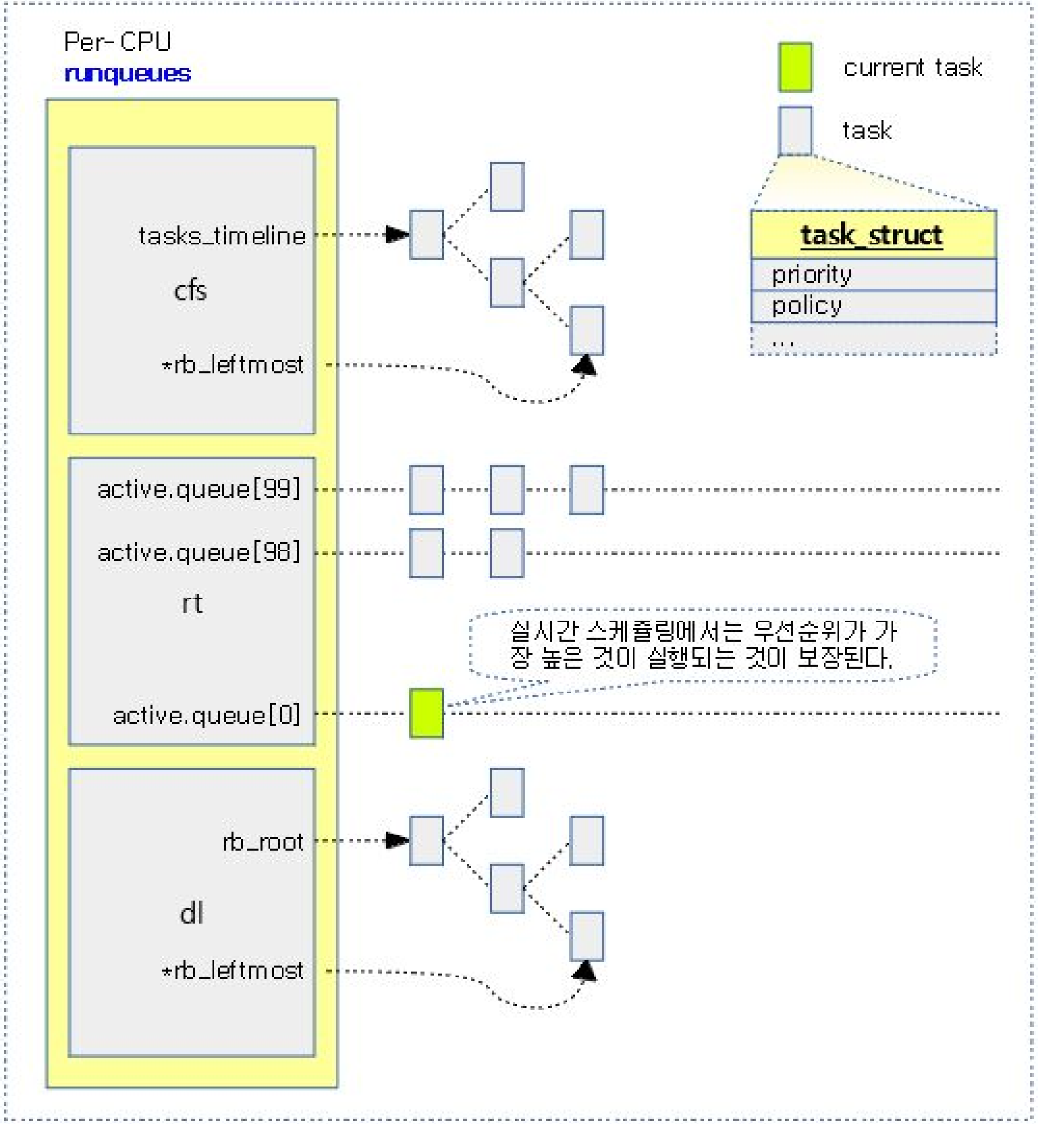
Red-black tree

- **Each runnable task is placed in a red-black tree**
  - A balanced binary search tree whose key is based on the value of vruntime
- **The leftmost node has the smallest key value**
  - the task with the highest priority



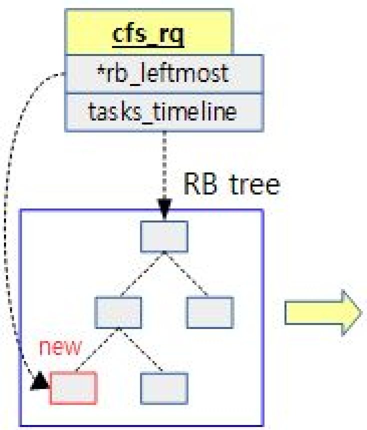
https://www2.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/6_CPU_Scheduling.html

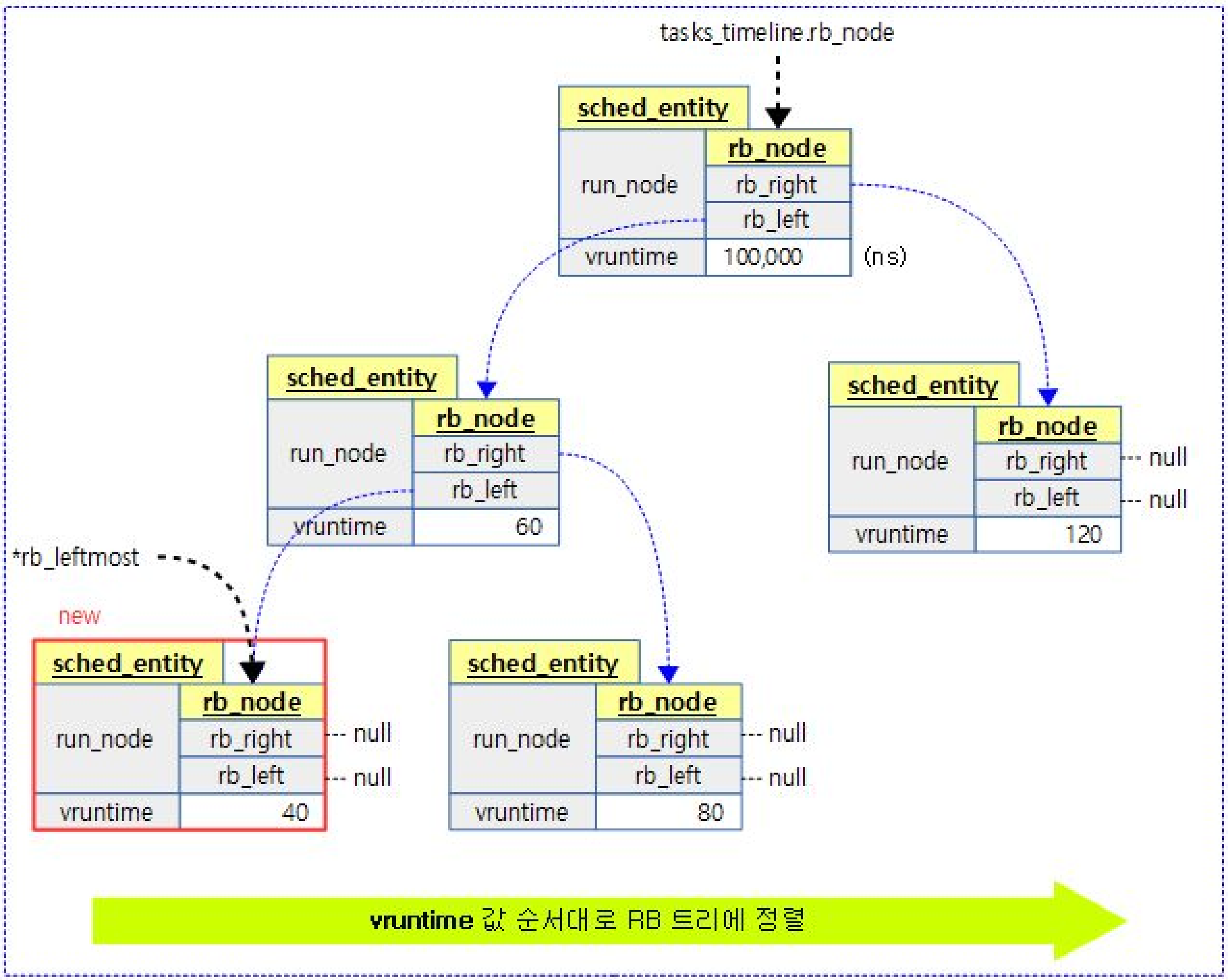
Red-black tree



- http://jake.dothome.co.kr/

Red-black tree





_enqueue_entity(cfs_rq, se)

Linux multiplexing

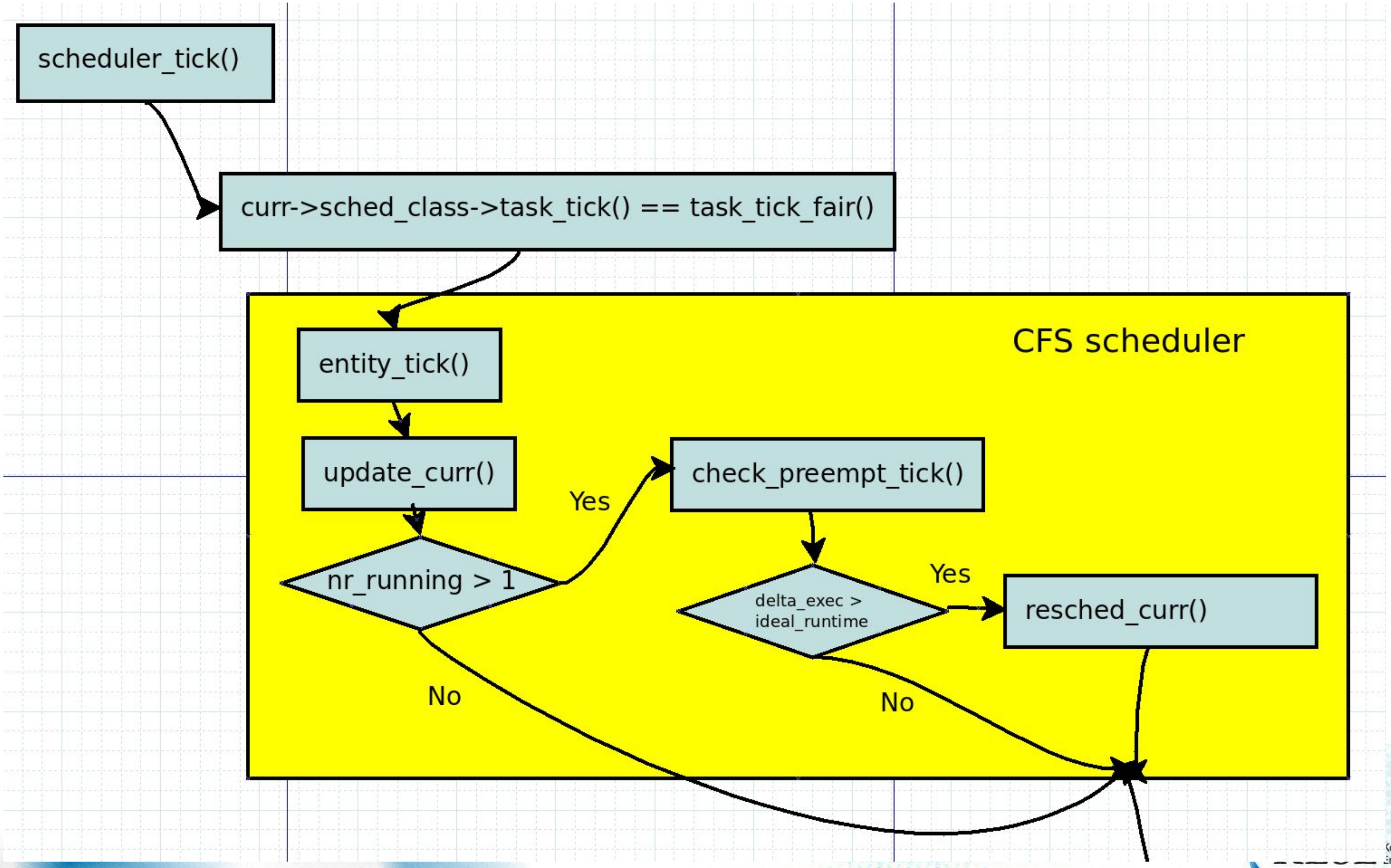
- **Linux multiplexes by two situations.**

- **Periodically forces a switch**
  – Tick interrupt

- **Sleep and wakeup mechanism**
  – process wait for device or pipe I/O

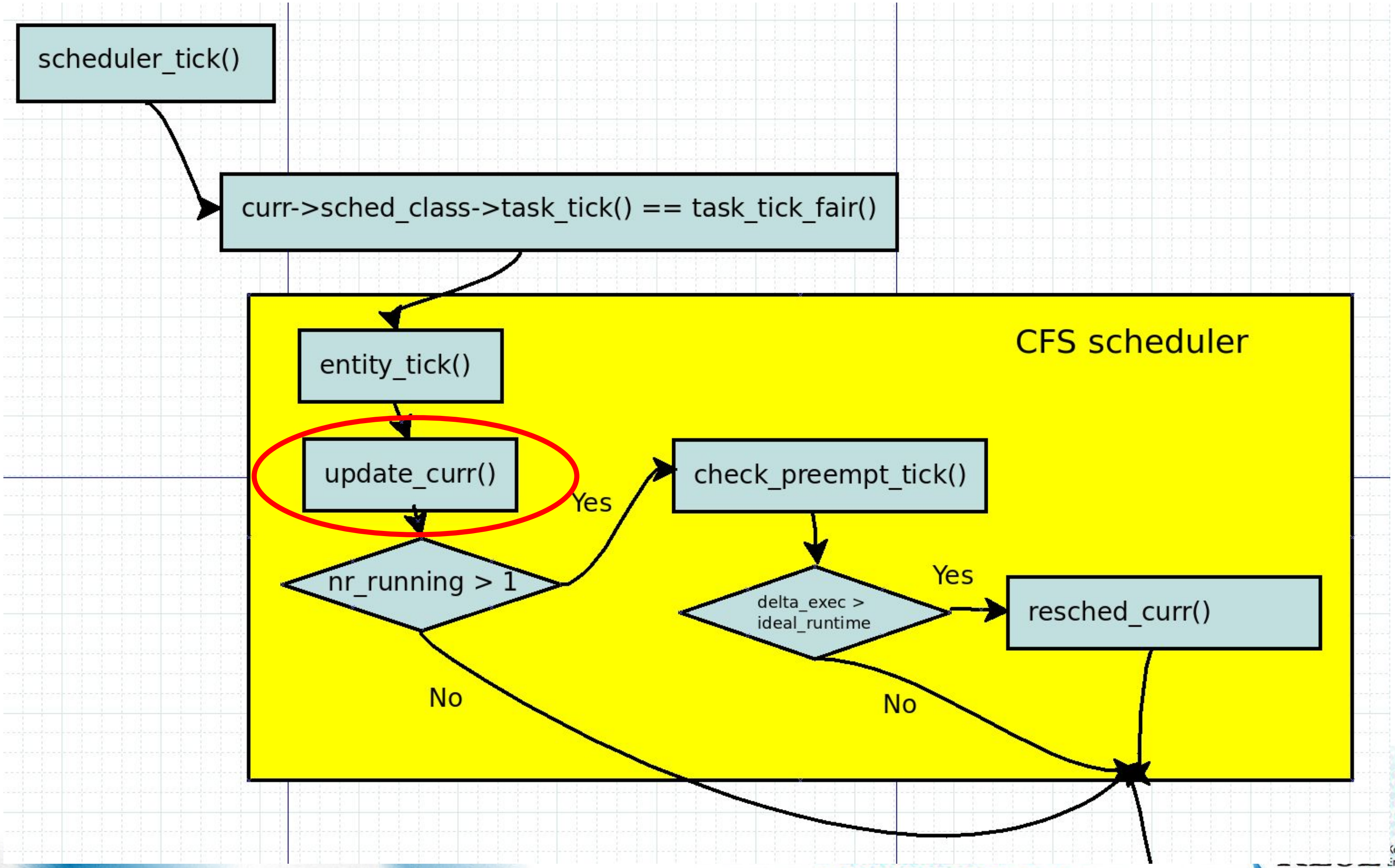
When schedule a task

- **Scheduler Tick**



When schedule a task

- **Scheduler Tick**

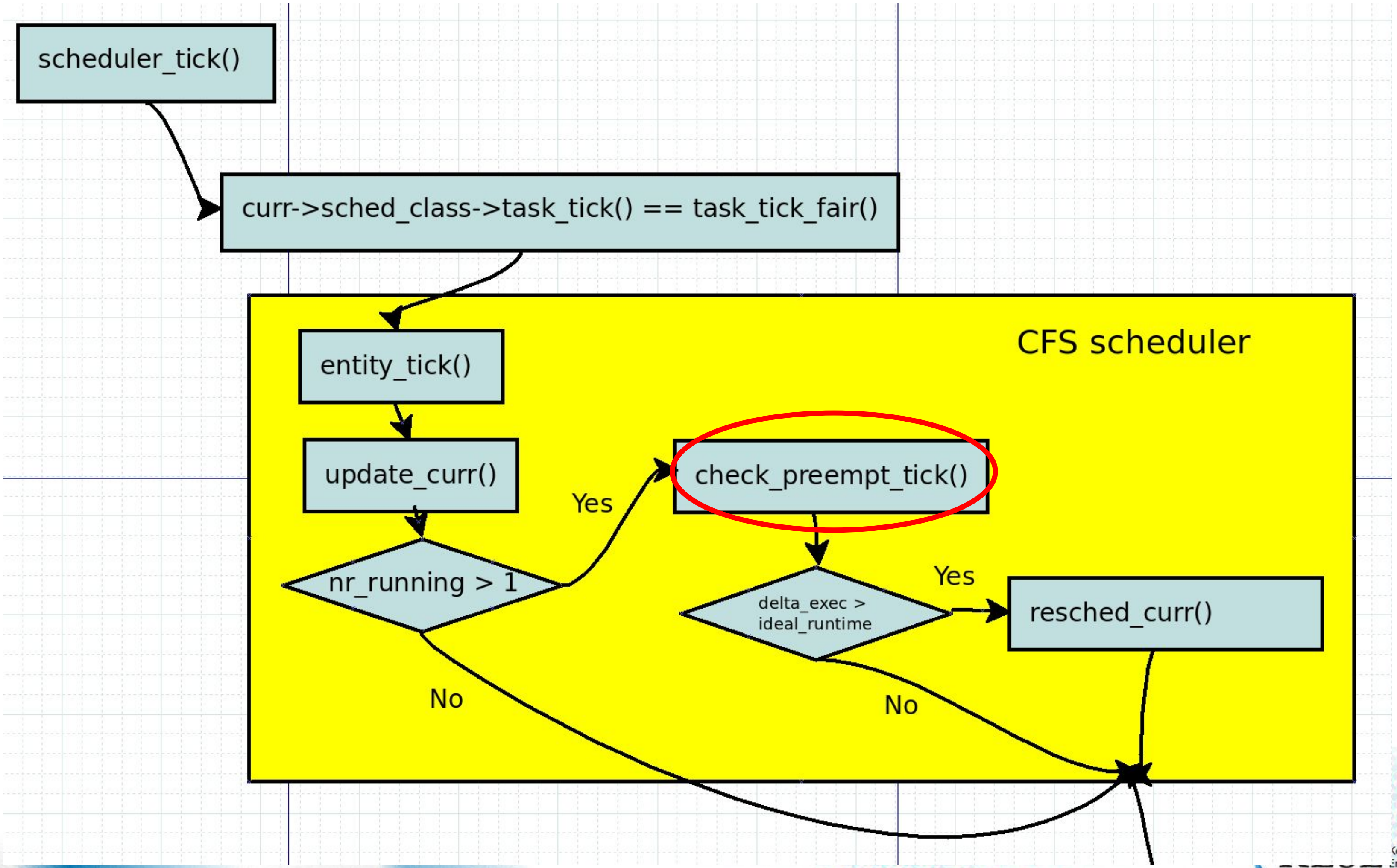


update_curr()

kernel/sched/fair.c

When schedule a task

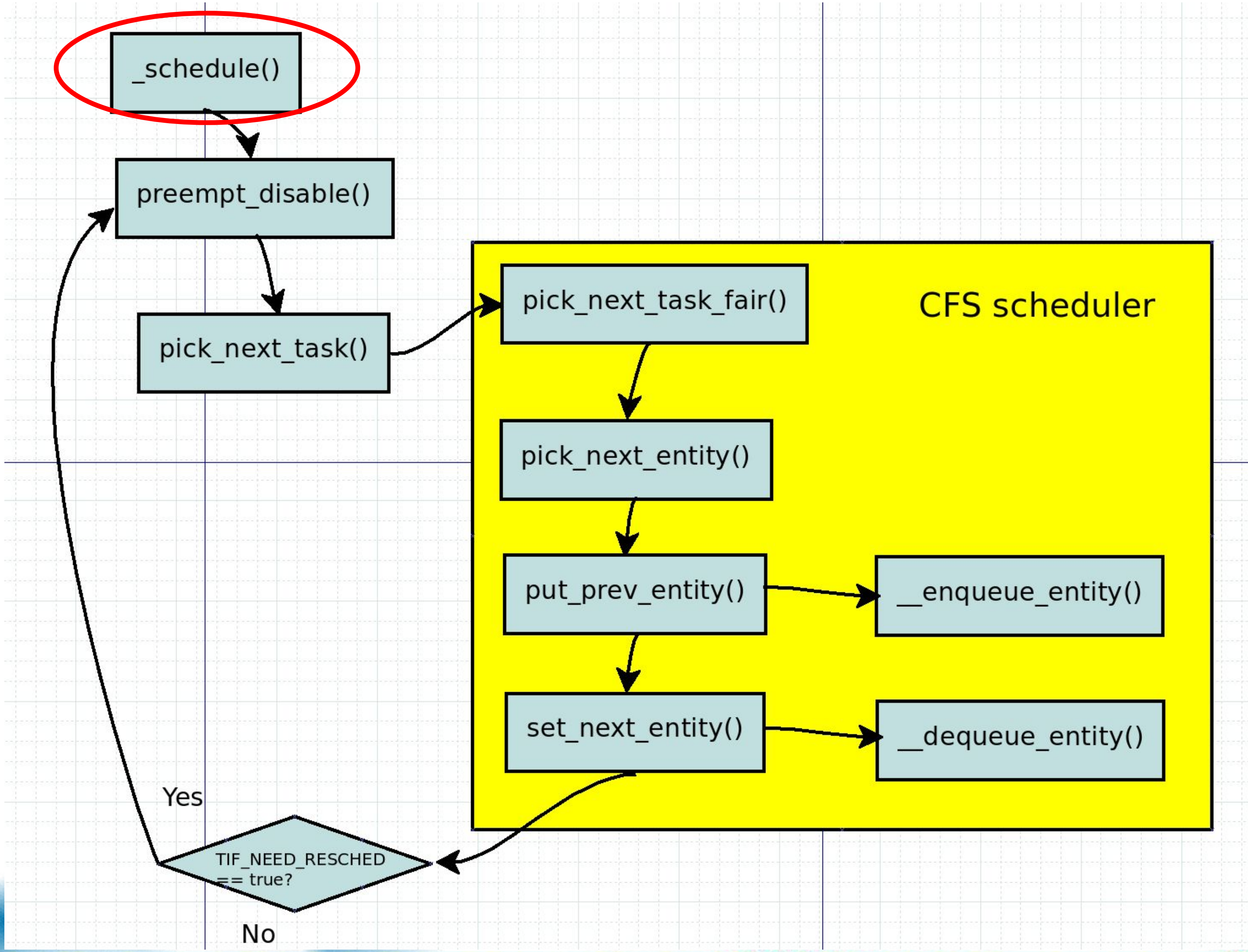
- **Scheduler Tick**



check_preempt_tick()

kernel/sched/fair.c

Scheduling by schedule()



Linux multiplexing

- **Linux multiplexes by two situations.**

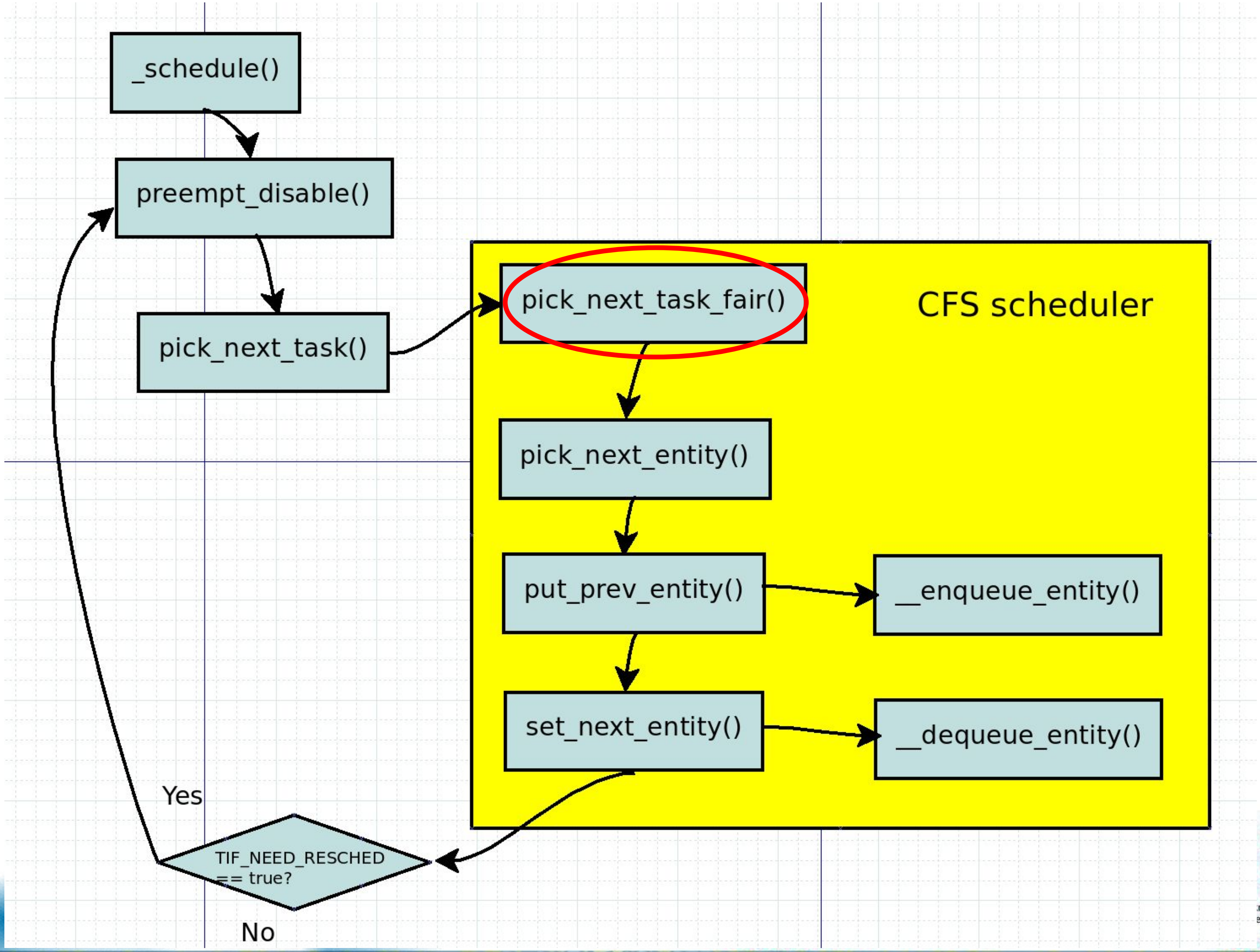
- **Periodically forces a switch**
  - Tick interrupt

- **Sleep and wakeup mechanism**
  - process wait for device or pipe I/O

“how to determine a thread’s ***timeslice****? and how to pick*

the ***next thread*** *to run”*

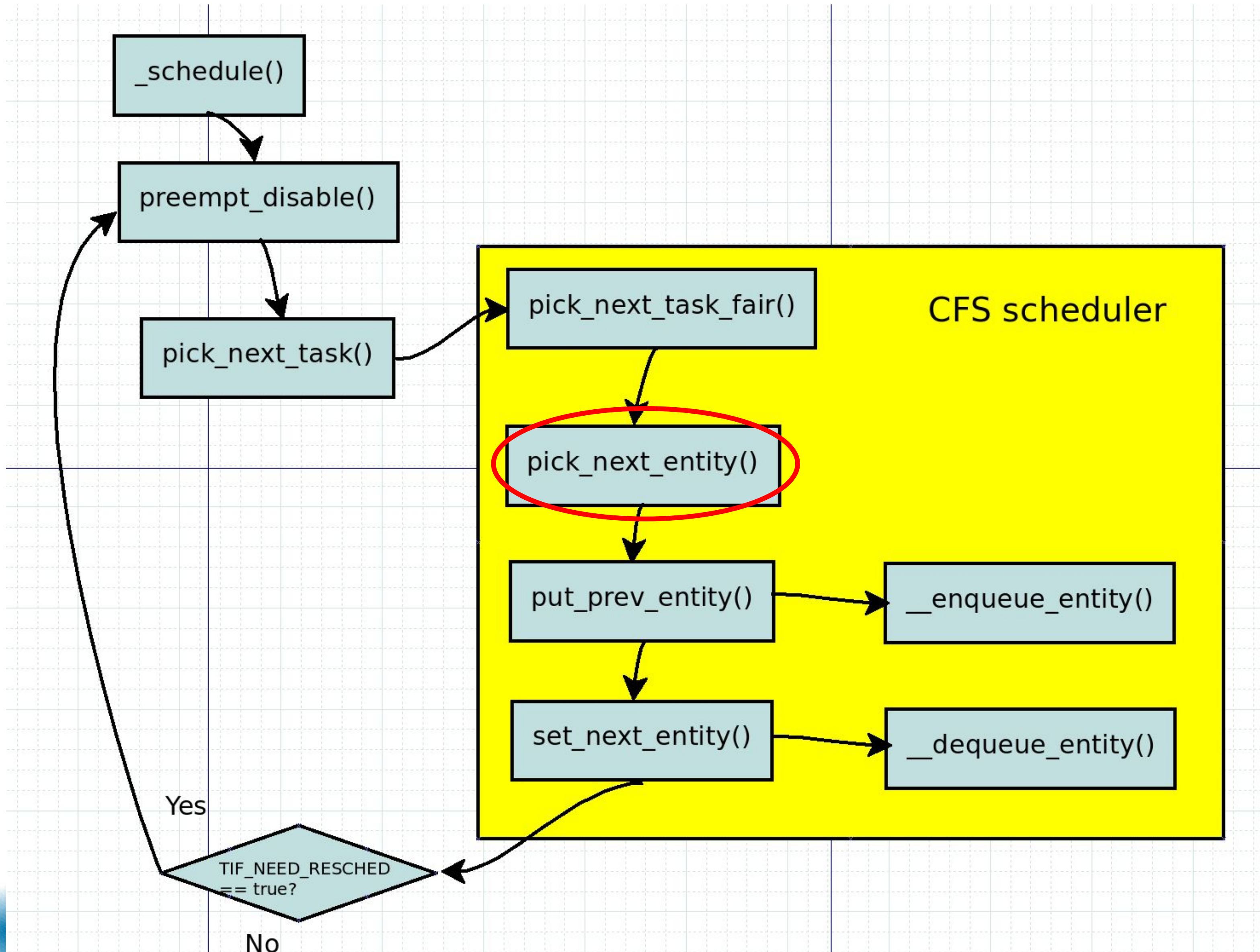
Scheduling by schedule()



pick_next_task_fair()

kernel/sched/fair.c

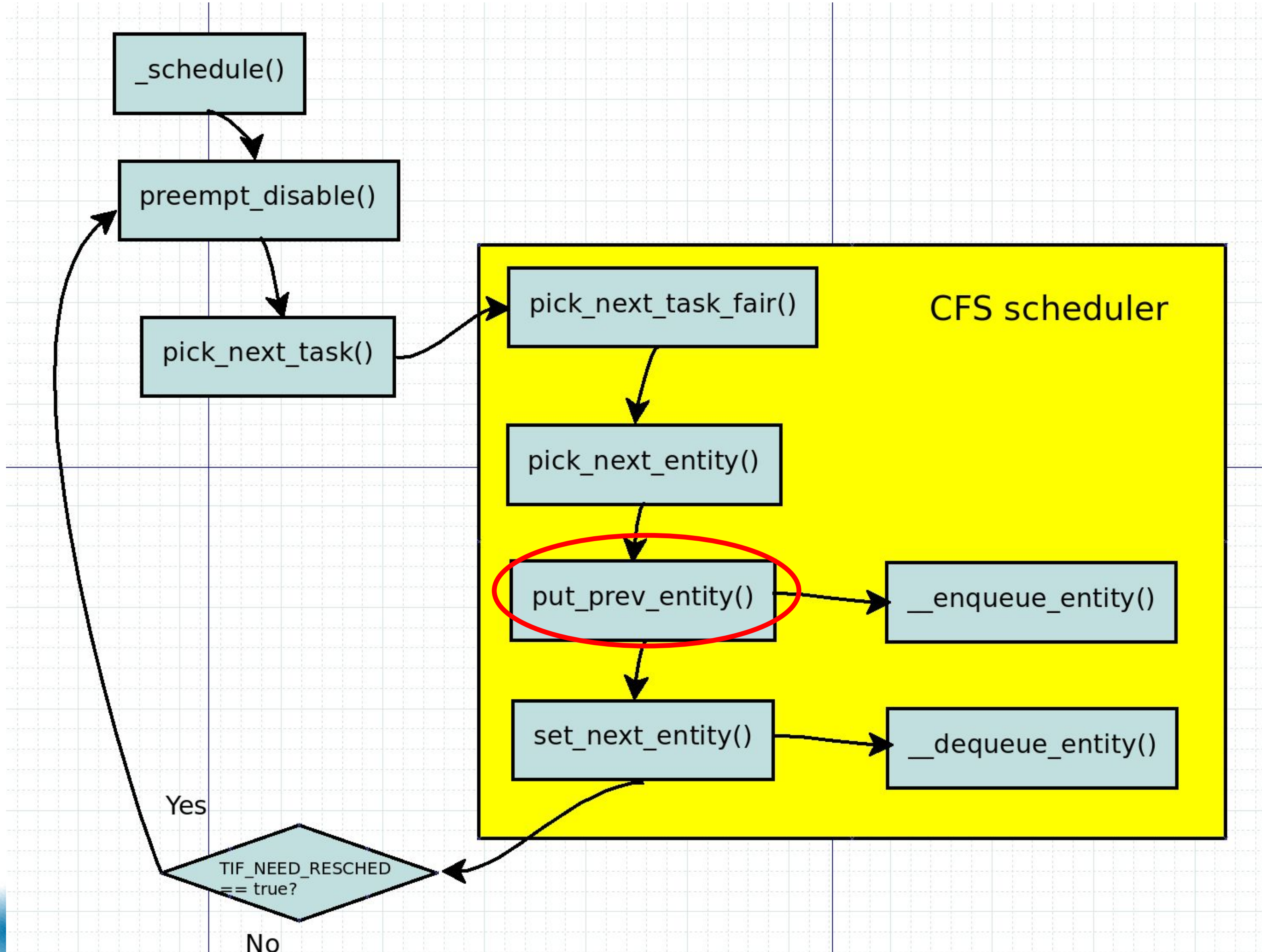
Scheduling by schedule()



pick_next_entity()

kernel/sched/fair.c

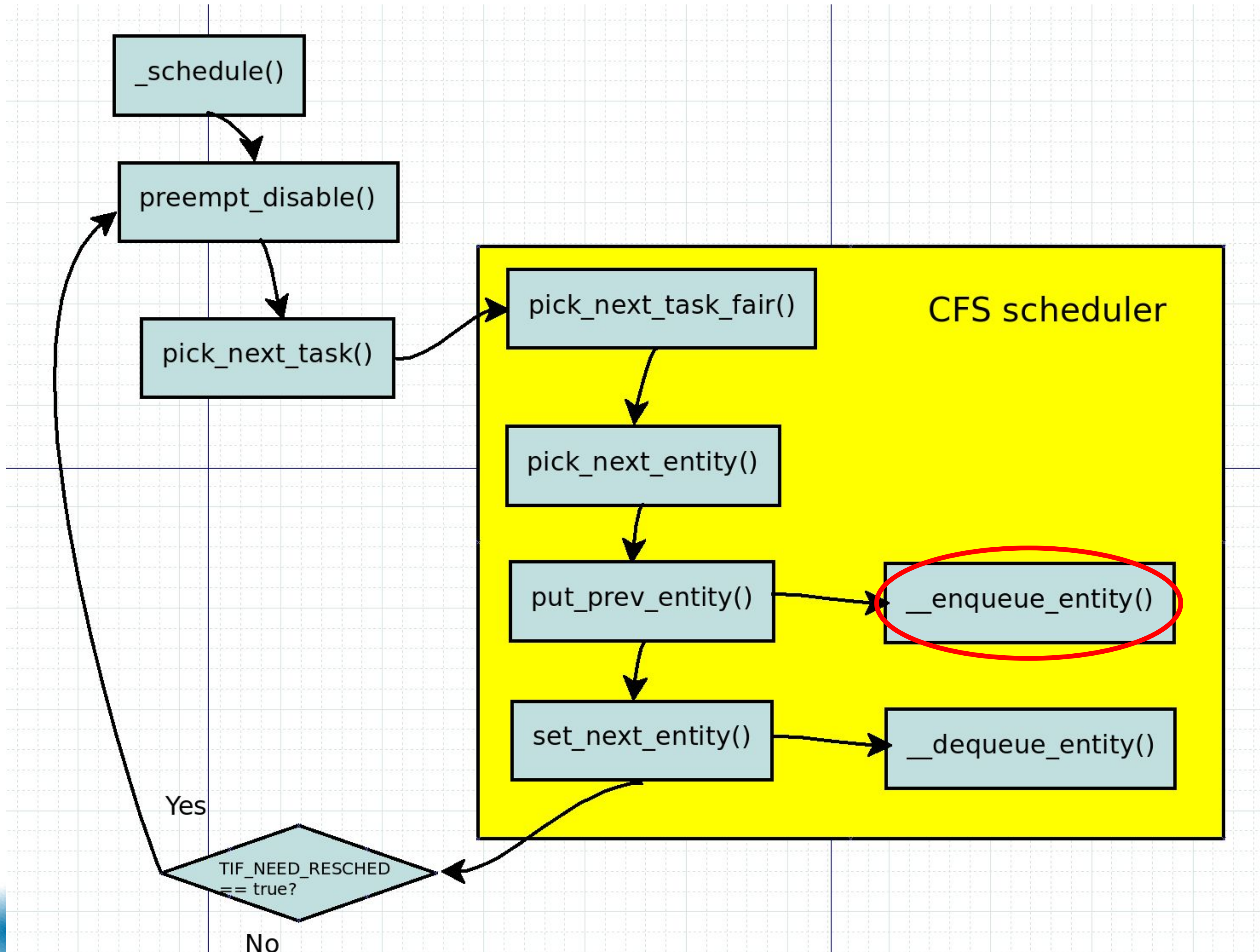
Scheduling by schedule()



put_prev_entity()

kernel/sched/fair.c

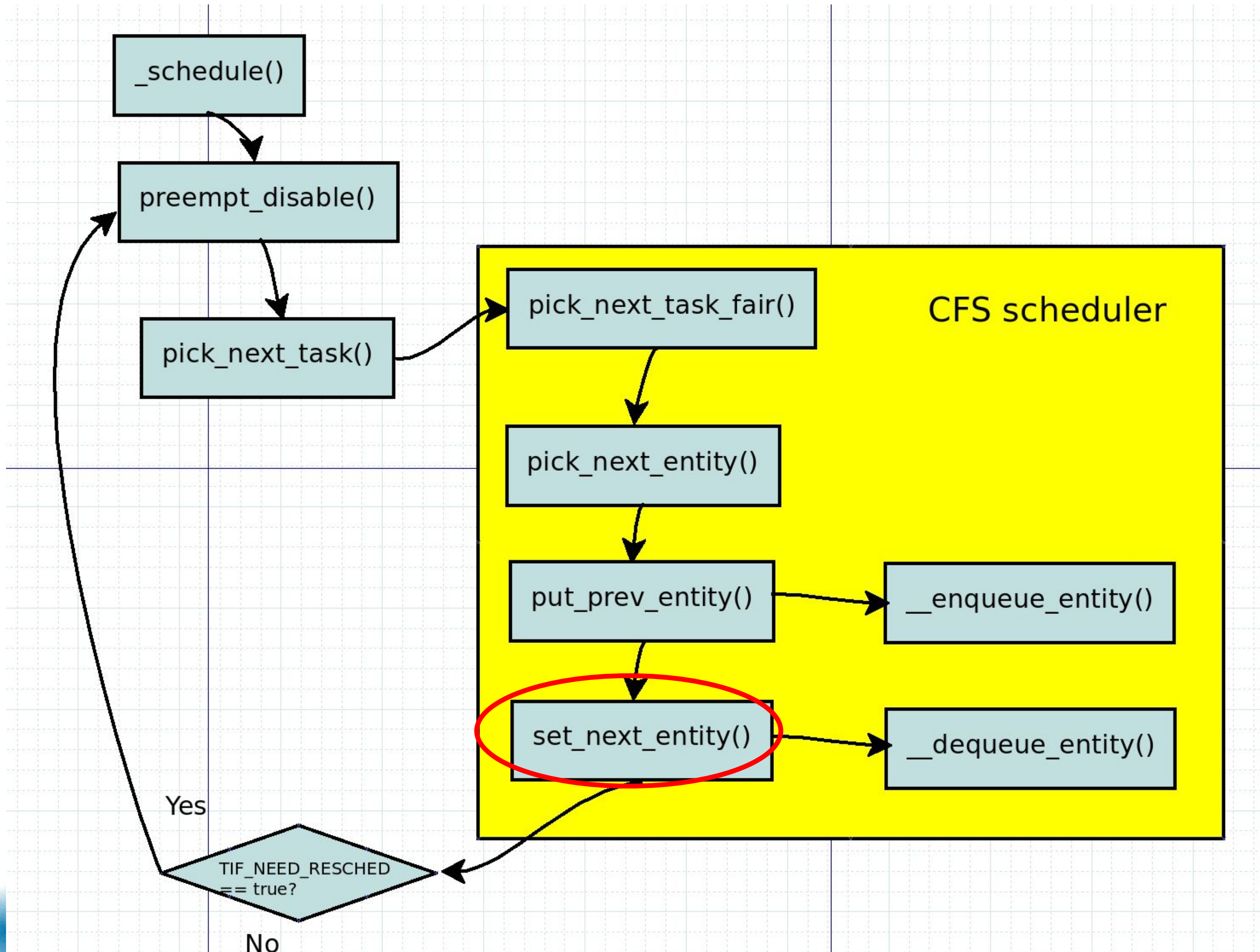
Scheduling by schedule()



__enqueue_entity()

kernel/sched/fair.c

Scheduling by schedule()



set_next_entity()

kernel/sched/fair.c

Next Step.

Energy-aware scheduling: EAS

1. **CFS scheduler - Kernel level**
2. **Load Balancer(Group Scheduling, Bandwidth Control, PELT)**
3. **EAS features**

Reference

- https://pdos.csail.mit.edu/6.828/2016/schedule.html
- http://web.mit.edu/6.033
- http://www.rdrop.com/~paulmck/
- "**Is Parallel Programming Hard, And If So, What Can You Do About It?**"
- Davidlohr Bueso. 2014. Scalability techniques for practical synchronization primitives. *Commun. ACM* 58

http://queue.acm.org/detail.cfm?id=2698990

- **"CPUFreq and The Scheduler Revolution in CPU Power Management", Rafael J. Wysocki**
- **https://sites.google.com/site/embedwiki/oses/linux/pm/pm-qos**
- **https://intl.aliyun.com/forum/read-916**
- **User-level threads : co-routines**

  http://www.gamedevforever.com/291

  https://www.youtube.com/watch?v=YYtzQ355_Co

- **Scheduler Activations**
  - https://cgi.cse.unsw.edu.au/~cs3231/12s1/lectures/SchedulerActivations.pdf
- **https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)**
- **http://jake.dothome.co.kr/**
- **http://www.linuxjournal.com/magazine/completely-fair-scheduler?page=0,0**
- **https://www2.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/6_CPU_Scheduling.html**
- **"Energy Aware Scheduling", Byungchul Park, LG Electronic**
- **"Update on big.LITTLE scheduling experiments", ARM**
- **"EAS Update" 2015 september ARM**
- **"EAS Overview and Integration Guide", ARM TR**
- **"Drowsy Power Management", Matthew Lentz, SOSP 2015**
- **https://www.slideshare.net/nanik/learning-aosp-android-hardware-abstraction-layer-hal**
- https://www.youtube.com/watch?v=oTGQXqD3CNI
- https://www.youtube.com/watch?v=P80NcKUKpuo
- https://lwn.net/Articles/398470/
- "SCHED_DEADLINE: It's Alive!", ARM, 2017

KESL Kookmin Univ. Embedded System Lab