CFS Load balancer

Domain, group scheduling, Bandwidth control, PELT load tracking

국민대학교 임베디드 연구실
경 주 현

2017 LG MC 인재육성팀 교육

Outline

- Load balancer
- CFS load balancer
- Schedule domain
- Group scheduling
- Bandwidth control
- PELT load tracking

On multi-core systems

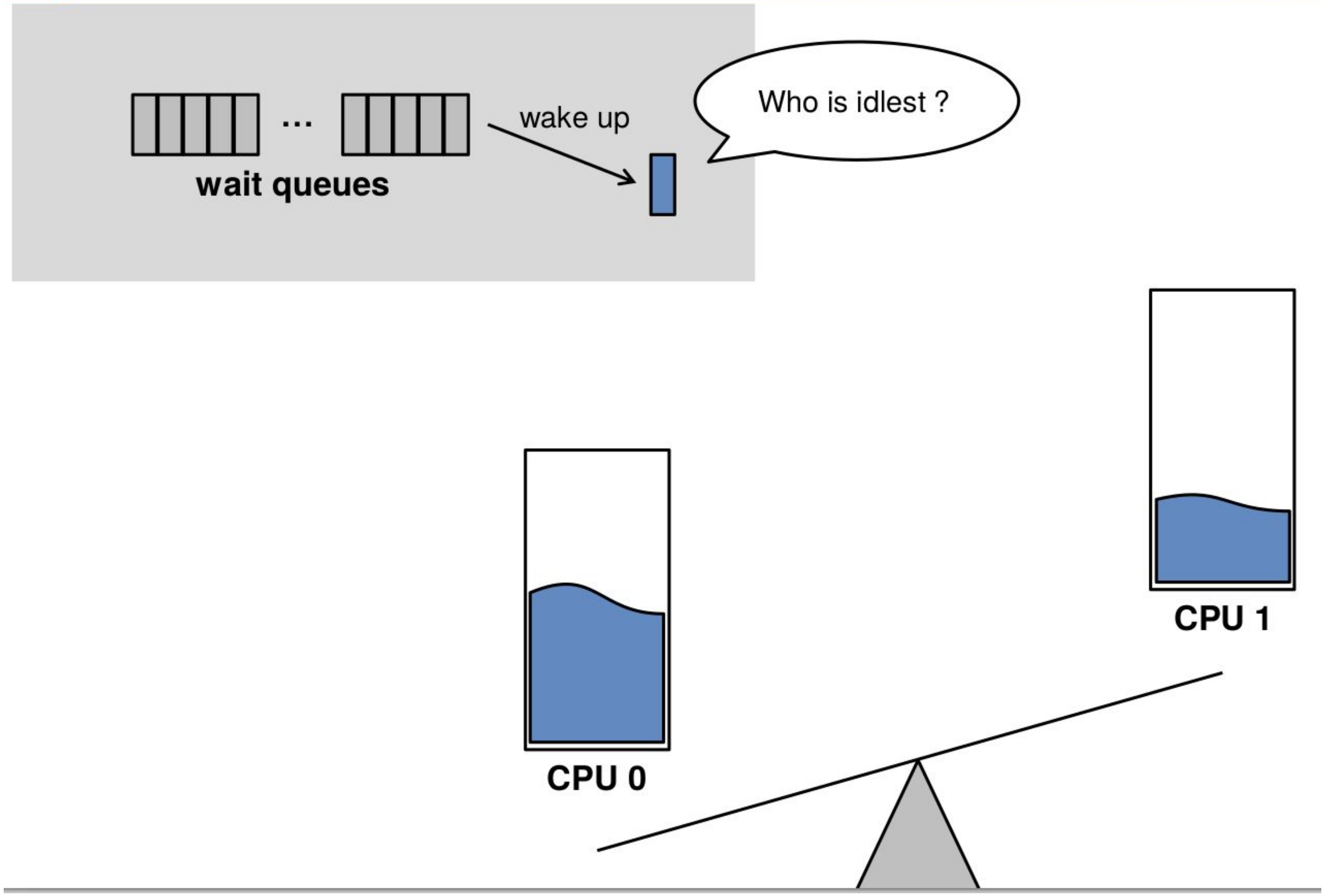
- **On a single-CPU system**

  CFS is very simple.

- **On multi-CPU systems**

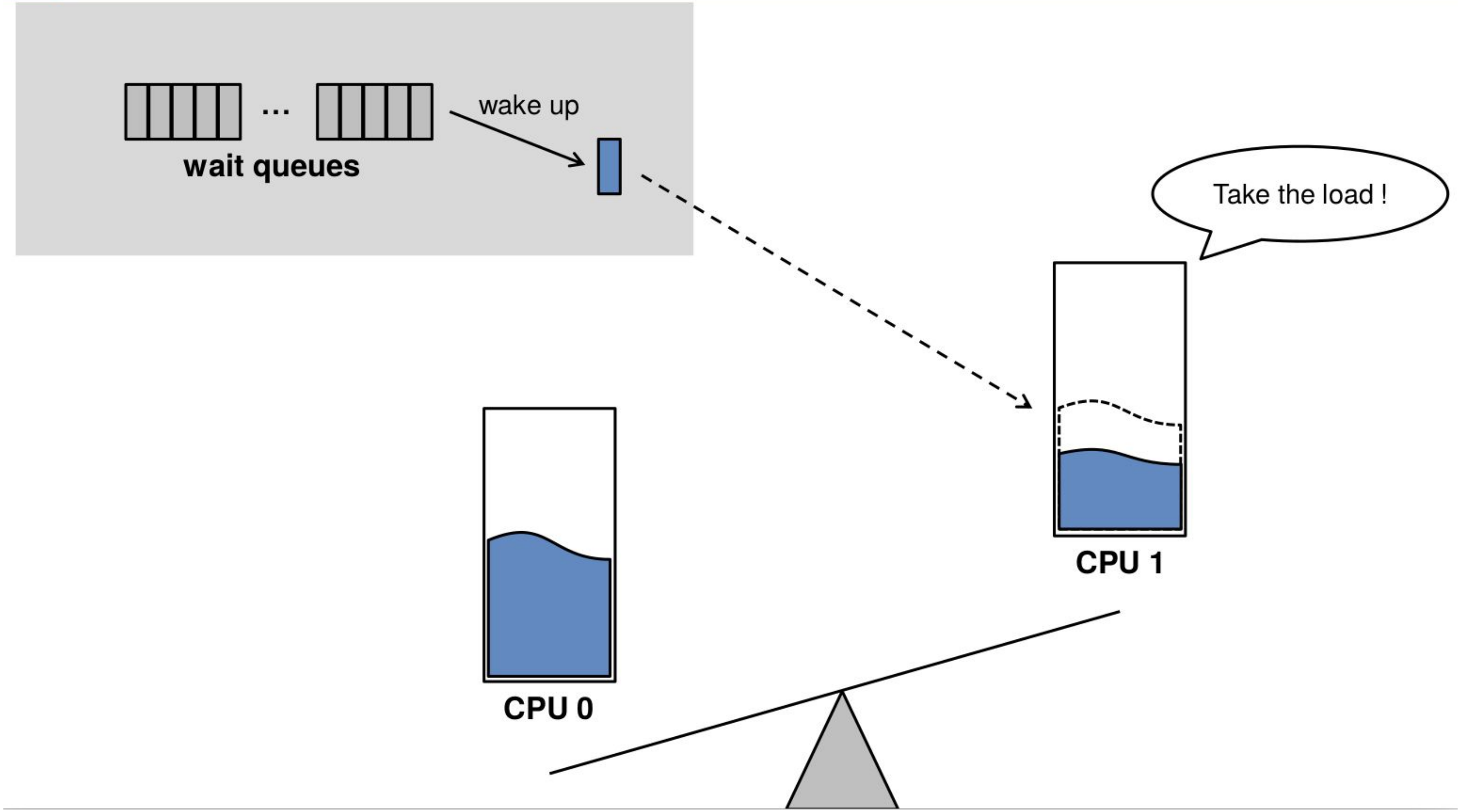
  CFS becomes quite complex

Load Balancing - review



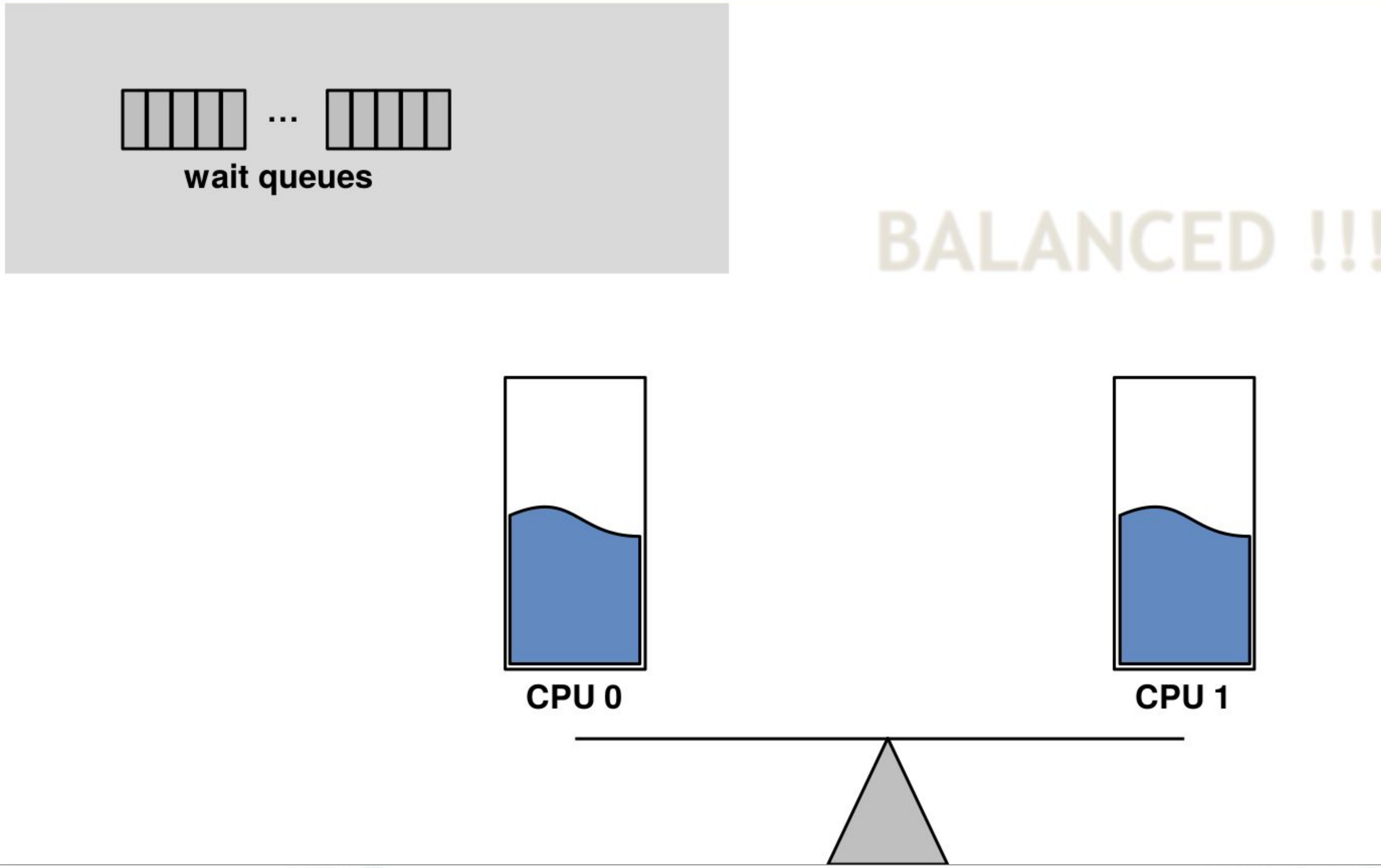
"Energy Aware Scheduling", Byungchul Park, LG Electronic

Load Balancing - review



"Energy Aware Scheduling", Byungchul Park, LG Electronic

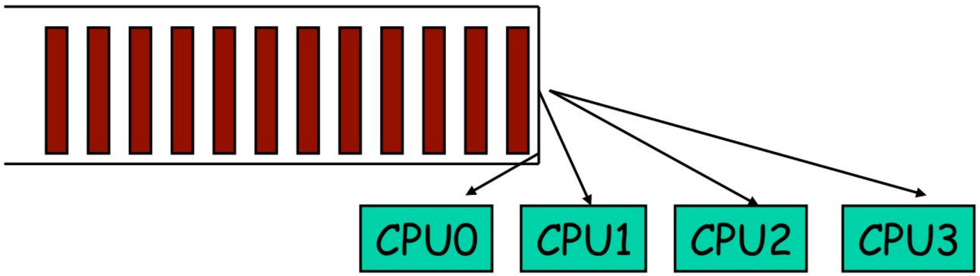
Load Balancing - review



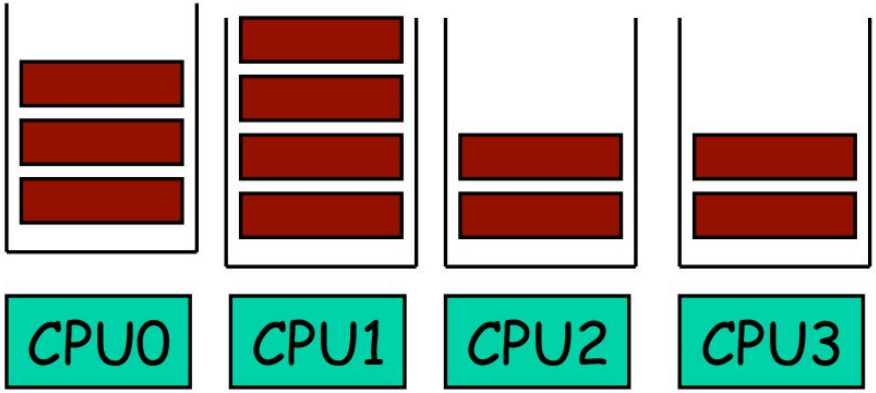
"Energy Aware Scheduling", Byungchul Park, LG Electronic

Run over key point again

- **Global shared runqueue.**
  - expensive synchronized access for global data structure



- **per-cpu runqueue.**
  - must be kept balanced -> Load balancing



KESL Kookmin Univ. Embedded System Lab

Load balancing

- **Conceptually, load balancing is simple.**
- **But, modern multicore systems is complex.**
  - Cache data structure, power, big.little, NUMA
- **To understanding the load balancing algorithm, the load tracking metric must be known.**

The load tracking metric.

- Load balancer uses to track load.
- Easy metric method

The load tracking metric.

- **Load balancer uses to track load.**
- **Easy metric method**
  - Balancing with **Number of threads**
- **Number of threads metric**
  - problem

The load tracking metric.

- **Load balancer uses to track load.**

- **Easy metric method**
  - Balancing with Number of threads

- **Number of threads metric**
  - problem :
    - queue 1 : high-priority thread queue
    - queue 2 : low-priority thread queue

The load tracking metric.

- **Load balancer uses to track load.**
- **Easy metric method**
  - Balancing with Number of threads
- **Number of threads based load-balancing**
  - problem :
    - queue 1 : high-priority thread queue
    - queue 2 : low-priority thread queue
- **Solution**
  - To balance the queues based on thread's **weights**, not their number.

Balancing with Number of threads

- **Problem**
  - One thread is high priority and nine threads are of low priority.
  - Queue 1: one thread (high priority)
  - Queue 2: nine threads (low priority
  - Work-stealing
    - When high priority thread often sleep.
    - It may frequently steal work from queue2 to queue1.

Balancing with thread's weights

- **Problem**
  - One thread is high priority and nine threads are of low priority.
  - Queue 1: one thread (high priority)
  - Queue 2: nine threads (low priority
  - Work-stealing
    - When high priority thread often sleep.
    - It may frequently steal work from queue2 to queue1.
- **Solution**

Balancing with thread's weights

- **Problem**
  - One thread is high priority and nine threads are of low priority.
  - Queue 1: one thread (high priority)
  - Queue 2: nine threads (low priority
  - Work-stealing
    - When high priority thread often sleep.
    - It may frequently steal work from queue2 to queue1.
- **Solution**
  - Do not just based on weights, but based on combination of **thread's weight and its average CPU utilization**.
- **CFS's load**
  - Thread's weight and its average CPU utilization

CFS load balancer

- **Problem**

  Consider
  - multithreading in different processes.
  - process 1 : lots of threads.
  - process 2 : few threads.

- **Result**

CFS load balancer

- **Problem**

  Consider
  - multithreading in different processes.
  - process 1 : lots of threads.
  - process 2 : few threads.

- **Result**
  - process 1 : receive a lot more CPU time than process 2
  - process 2 : starvation
  - This would be unfair.

CFS load balancer

- **Problem**

  Consider
  - multithreading in different processes.
  - process 1 : lots of threads.
  - process 2 : few threads.

- **Result**
  - process 1 : receive a lot more CPU time thad process 2
  - process 2 : starvation
  - This would be unfair.

- **Solution**
  - Linux added a **group scheduling feature** to bring fairness between groups of threads

Linux Group Scheduling

- **Group scheduling is enabled**
  - CONFIG_FAIR_GROUP_SCHED

- **A group of tasks is called a “scheduling entity”**

```
struct sched_entity {
        struct load_weight load;
        struct sched_entity *parent;
        struct cfs_rq *cfs_rq;
        struct cfs_rq *my_rq;
        struct sched_avg avg;
        /* ... */
   };
```

Group Scheduling Data Structure

- Scheduling is always at the granularity of sched_entity
- A single task becomes a scheduling entity

```
struct task_struct {
    struct sched_entity se;
    /* ... */
};
```

- Each scheduling entity contains a runqueue.

```
struct cfs_rq {
    struct load_weight load;
    unsigned long runnable_load_avg;
    unsigned long blocked_load_avg;
    unsigned long tg_load_contrib;
    /* ... */
};
```

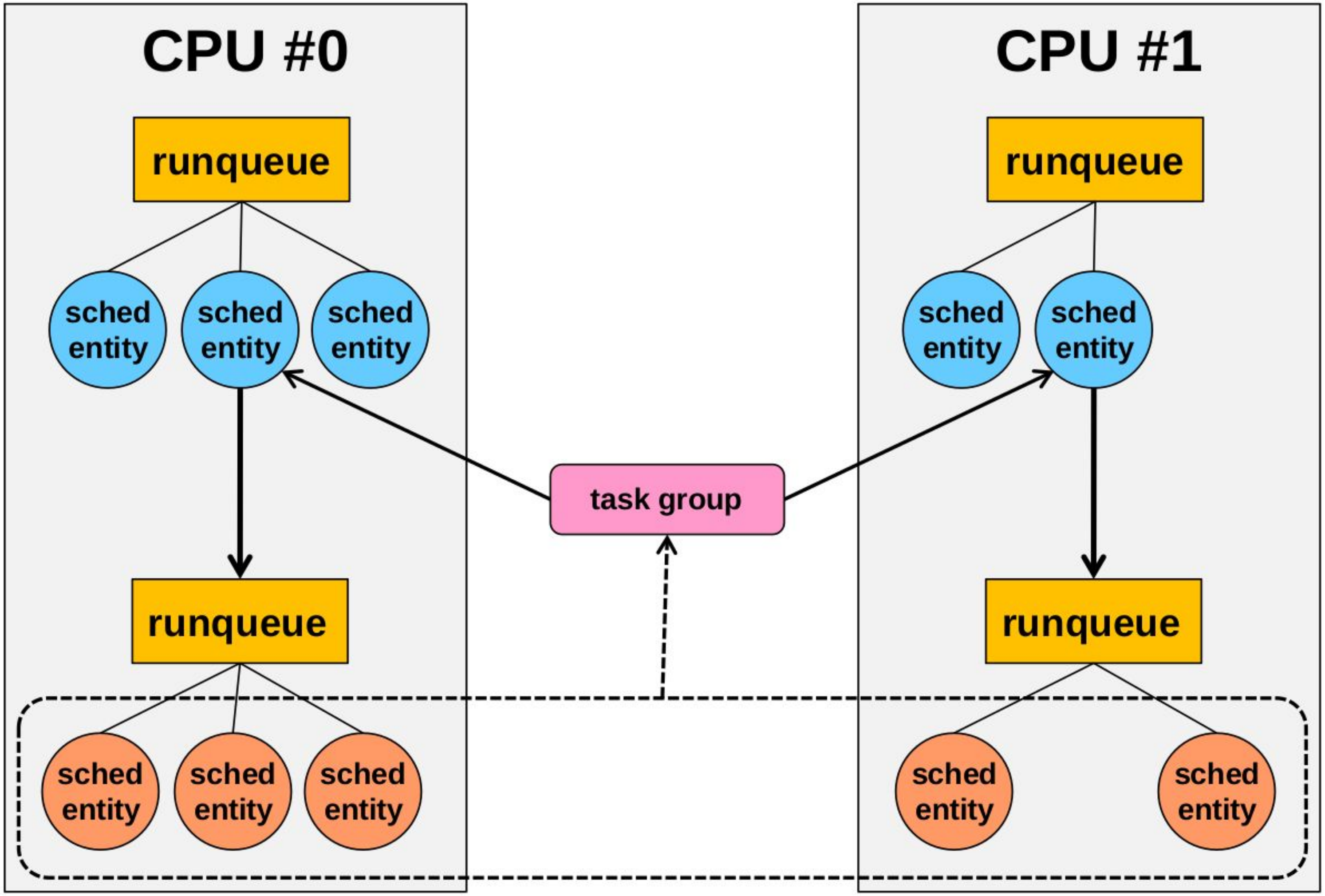
Group Scheduling Data Structure

- **Extend the concept to multiprocessor**
-
```
struct task_group {
      struct sched_entity **se;
      struct cfs_rq **cfs_rq;
      unsigned long shares;
      atomic_long_t load_avg;
      /* ... */
  };
```

- **c is CPU NUM**

```
tg->se[c] = &se;
tg->cfs_rq[c] = &se->my_q;
```

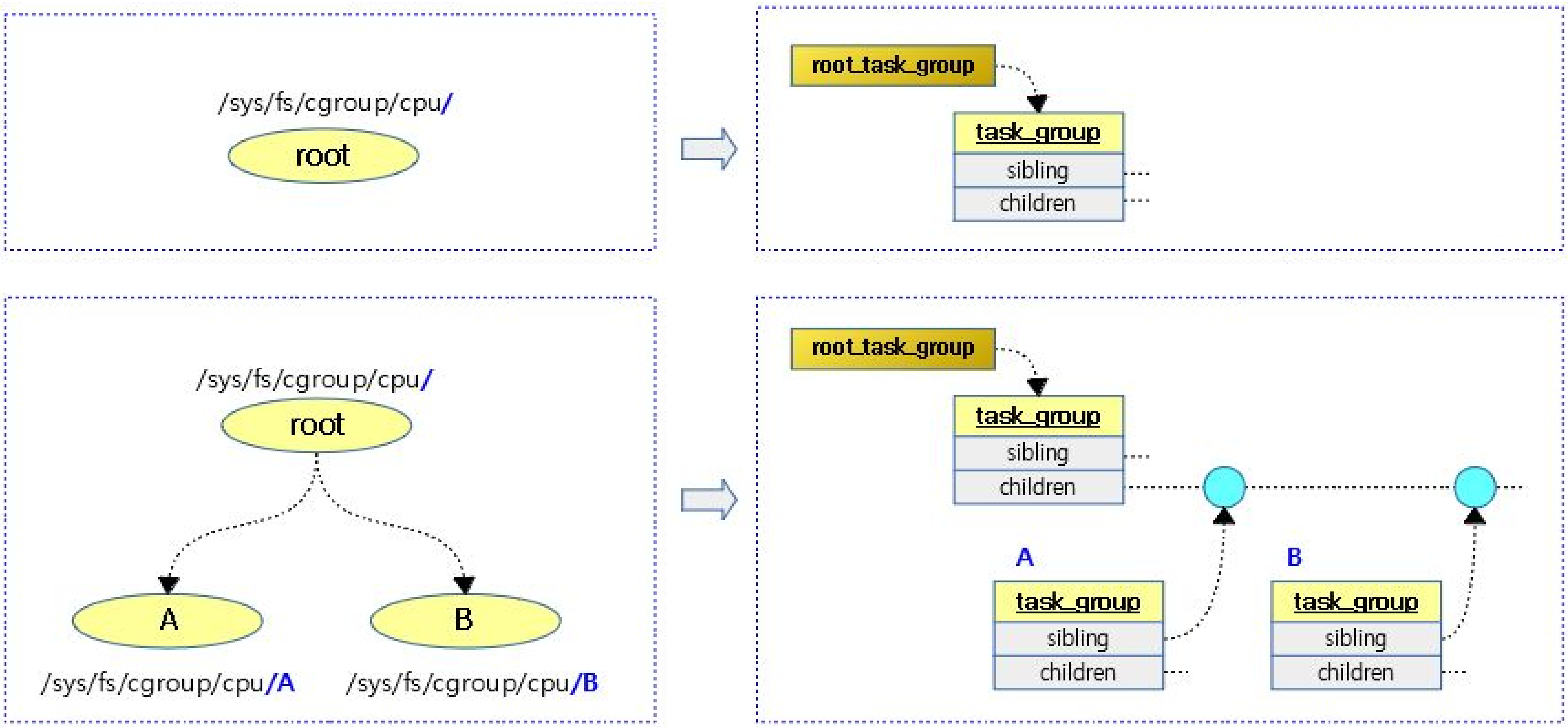
Group Scheduling Data Structure

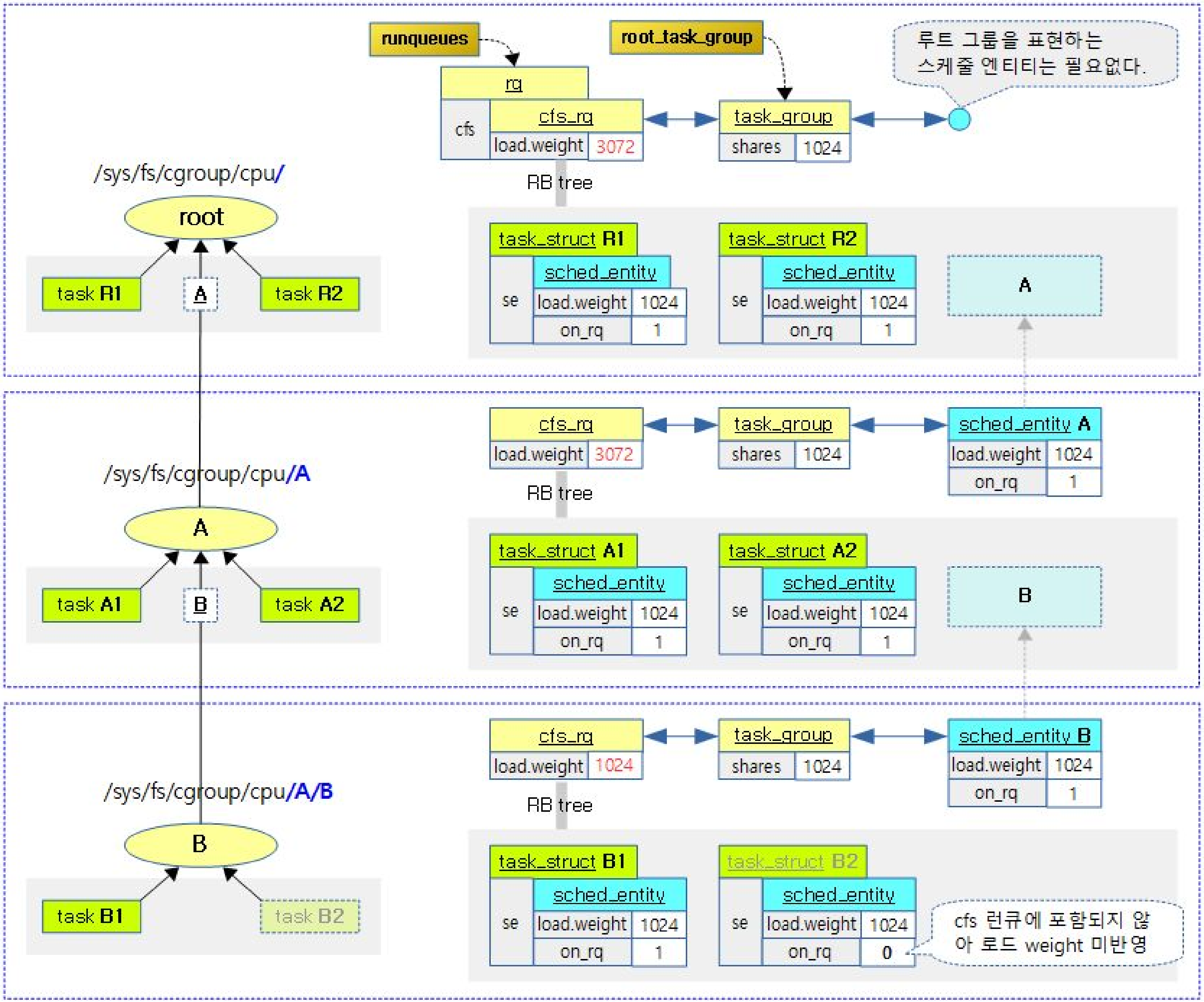


http://rtcc.hanyang.ac.kr/rtccw/?page_id=1745

KESL Kookmin Univ. Embedded System Lab

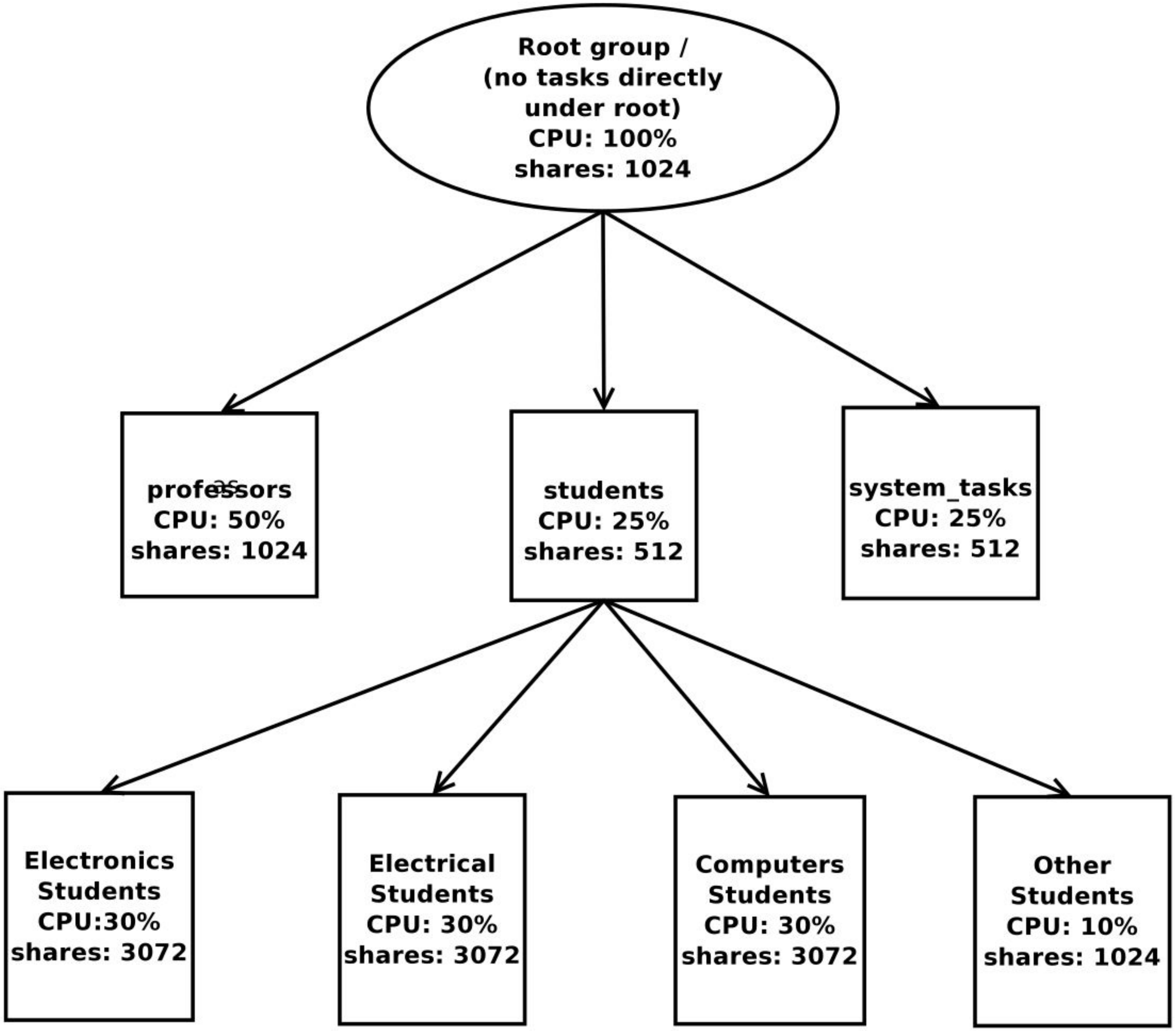
Group Scheduling Data Structure





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

Linux Group Scheduling



CFS load balancer

- **Problem**
  - When there is free CPU time available
    - The CPU go idle X
  - The CFS scheduler will give any left-over time to other processes
  - Sometimes system administrator may want to limit maximum share of CPU time
  - The CFS scheduler cannot limit CPU time.

CFS bandwidth control feature

- **Problem**
  - When there is free CPU time available
    - The CPU go idle X
  - The CFS scheduler will give any left-over time to other processes
  - Sometimes system administrator may want to limit maximum share of CPU time
  - The CFS scheduler cannot limit CPU time.

- **Solution**
  - Linux added a **CFS bandwidth control feature** to limit Limiting the maximum share of CPU time that a process (or group of processes).

CFS bandwidth control

- **Example use cases**
  - Virtual Machines
  - Pay-per-use

CFS bandwidth control - two knobs

- **cpu.cfs_period_us :**
  - The period over which the group's CPU usage is to be regulated

- **cpu.cfs_quota_us :**
  - how much CPU time is available to the group over that period

CFS bandwidth control - two knobs

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

CFS load balancer

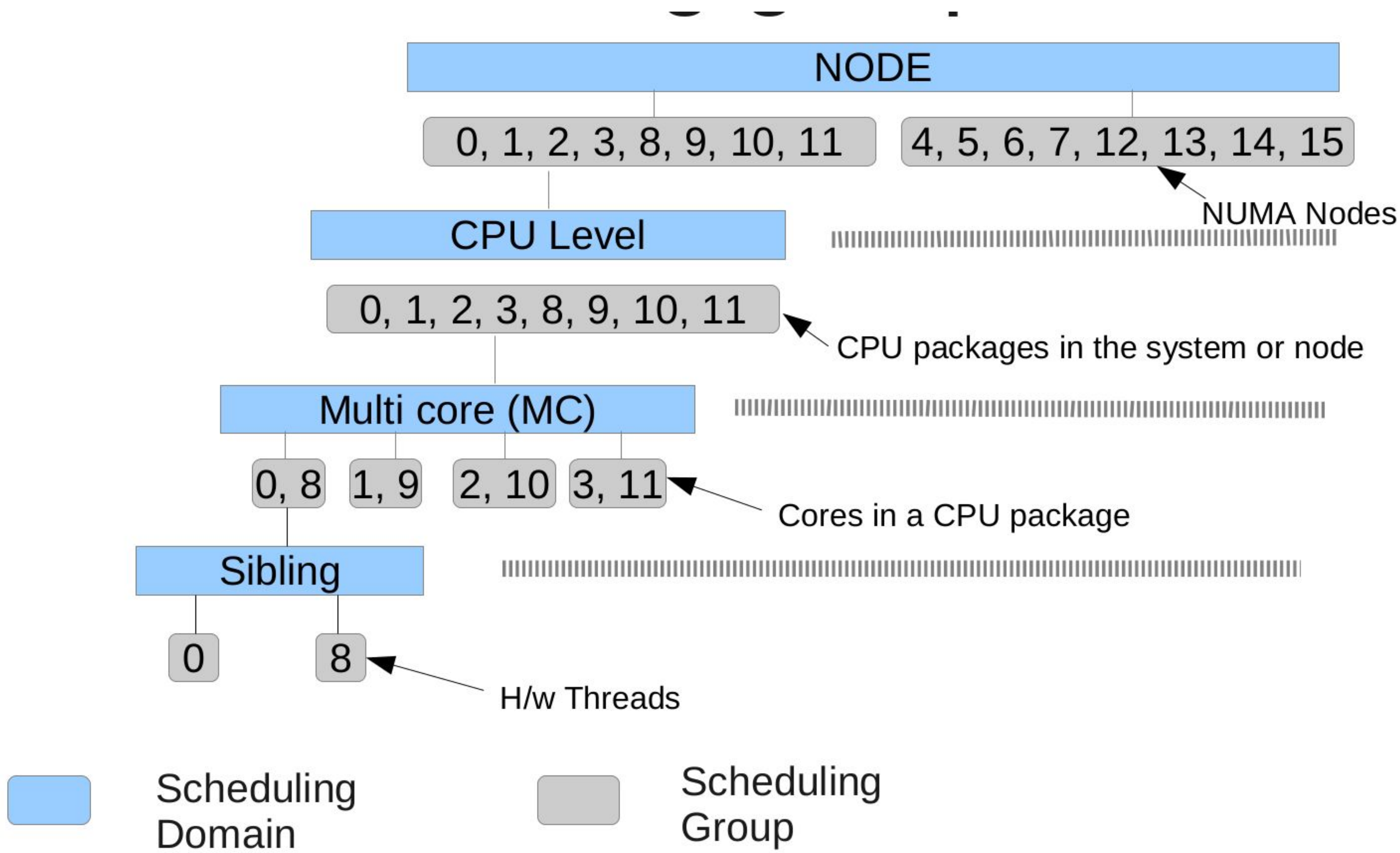
- **Problem**
  - Hardware topologies are becoming more varied, accommodating different power/performance budgets:
    - SMP, NUMA, ARM big.LITTLE technology.
  - Modern hardware composed of hierarchy levels of subsystems

Scheduling domain

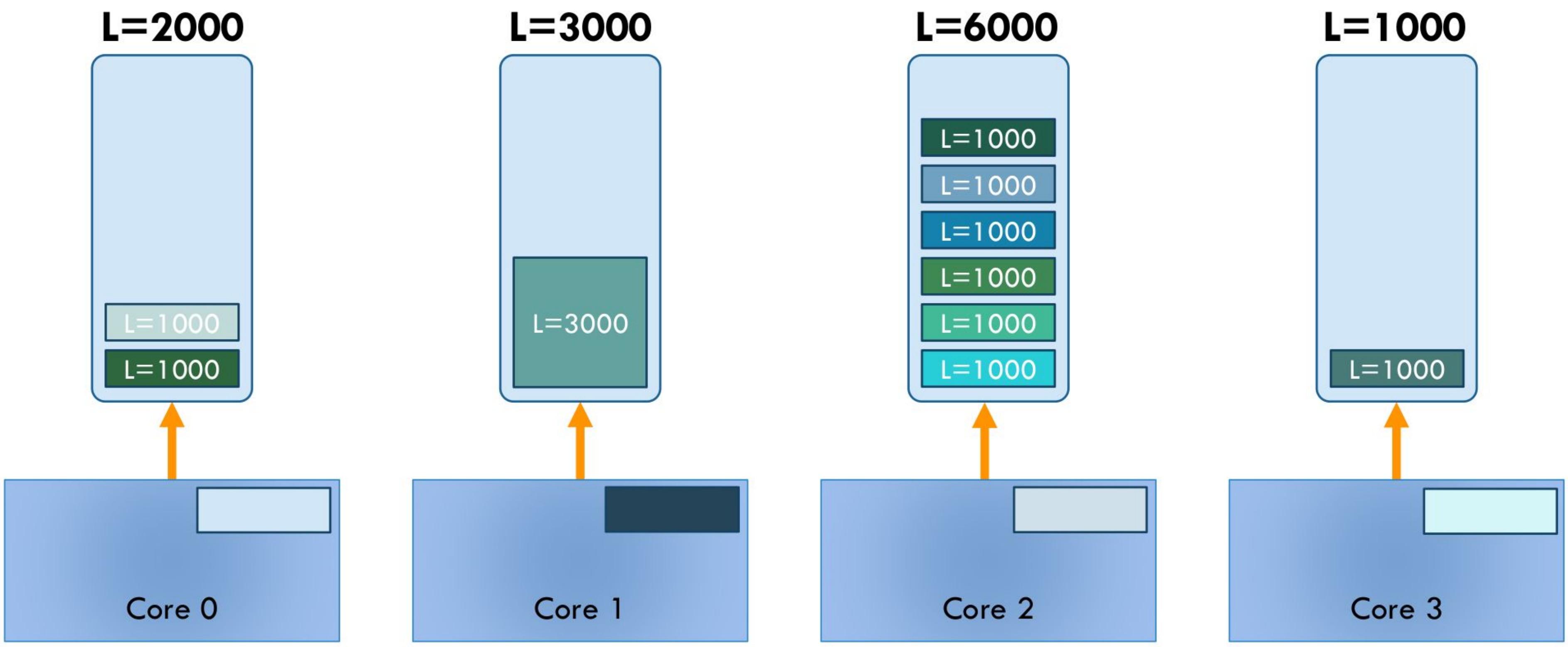
- **Problem**
  - Hardware topologies are becoming more varied, accommodating different power/performance budgets:
    - SMP, NUMA, ARM big.LITTLE technology.
  - Modern hardware composed of hierarchy levels of subsystems

- **Solution**
  - Linux added a **scheduling domain feature** to cover various multi-core systems such as cache architecture, NUMA, heterogeneous systems.

CFS Scheduling Domains

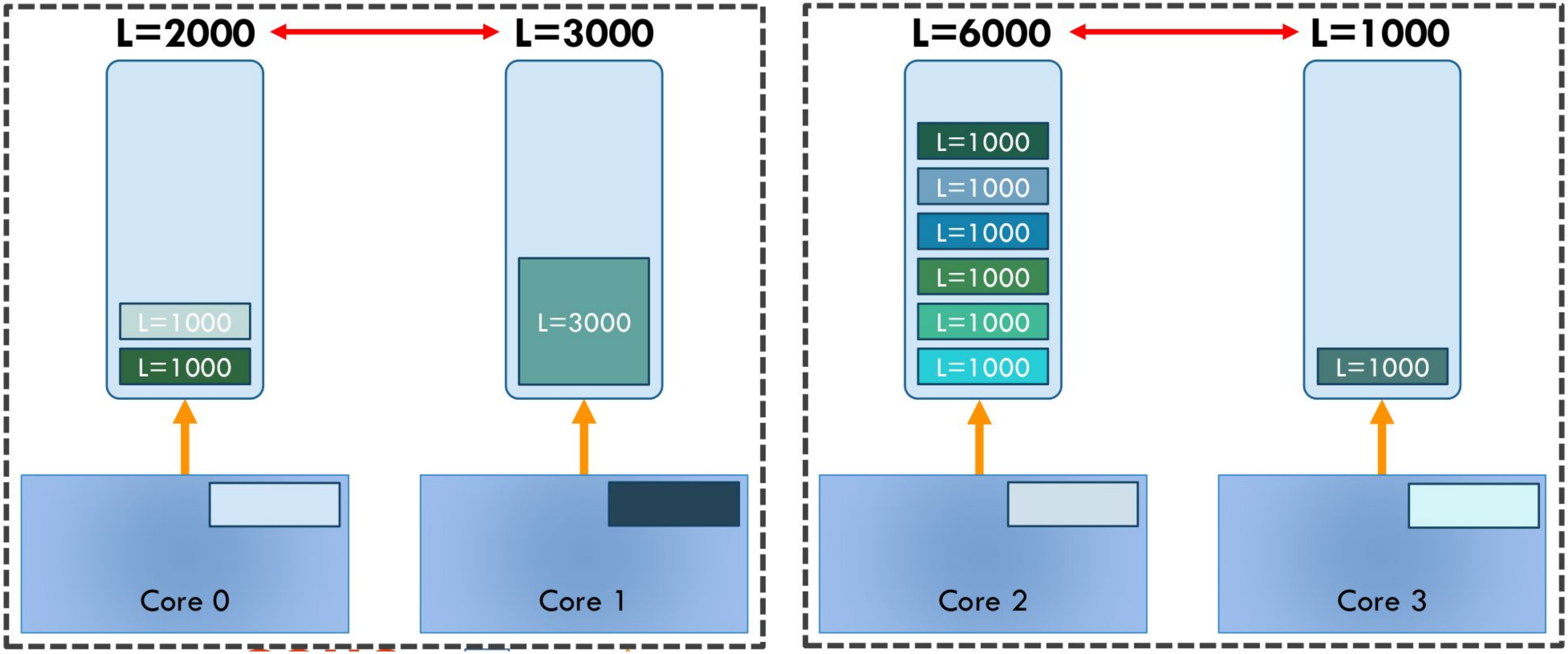


HIERARCHICAL LOAD BALANCING



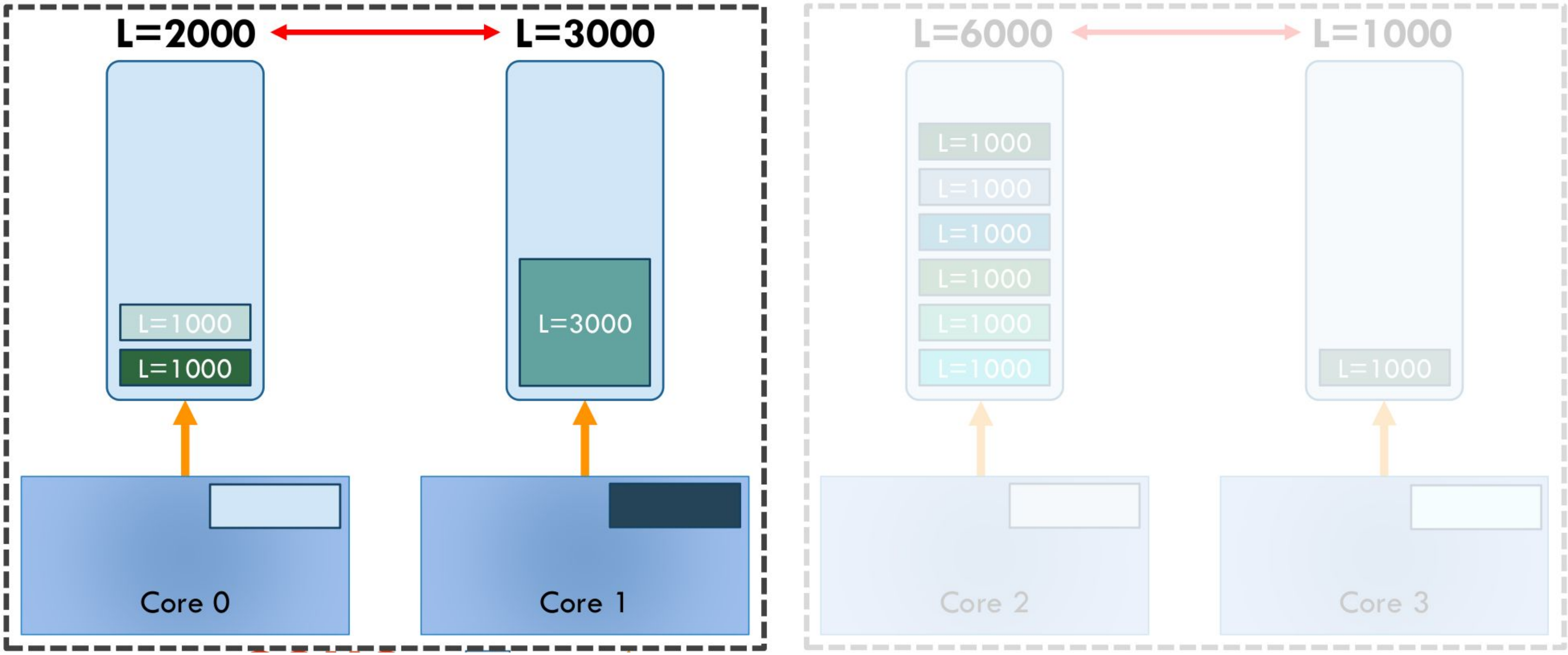
"THE LINUX SCHEDULER:
A DECADE OF WASTED CORES", EuroSys'16, Jean-Pierre Lozi, et,.al.

HIERARCHICAL LOAD BALANCING



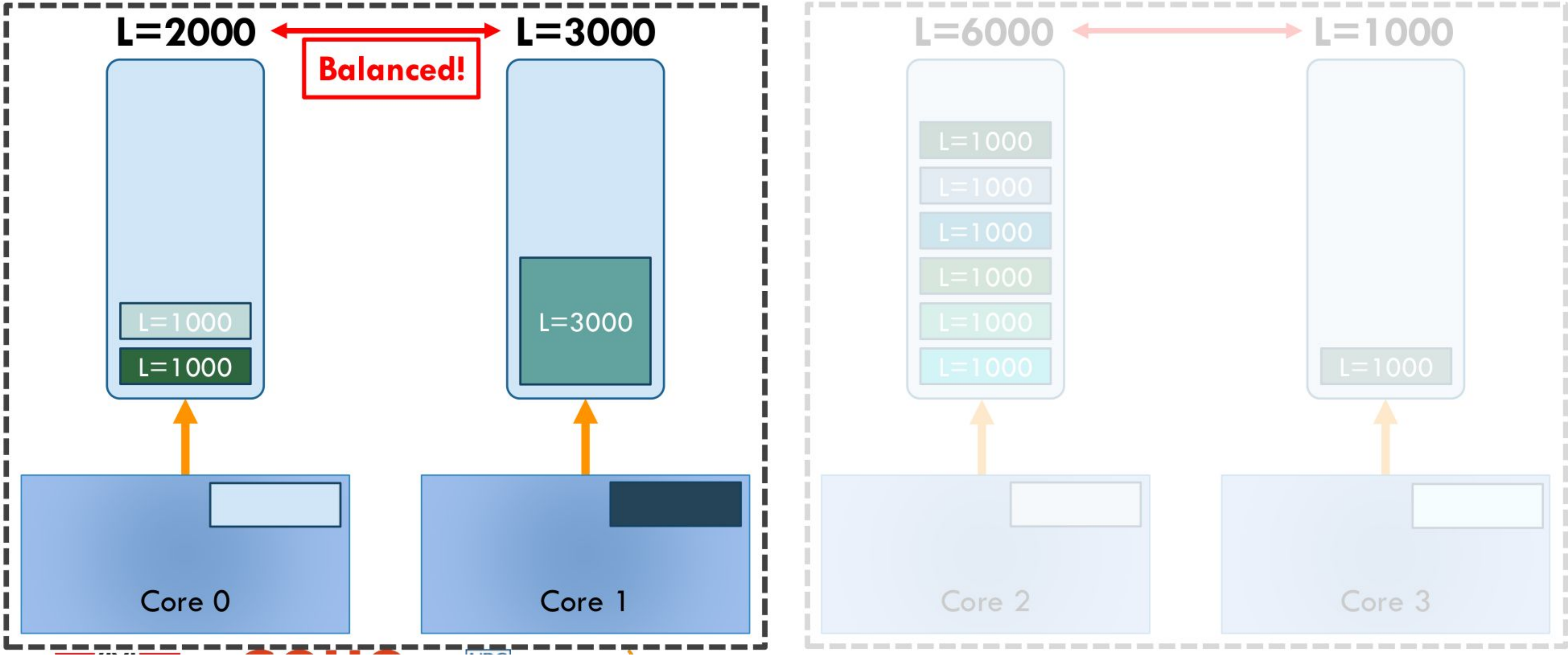
“THE LINUX SCHEDULER:
A DECADE OF WASTED CORES”, EuroSys’16, Jean-Pierre Lozi, et,.al.

HIERARCHICAL LOAD BALANCING



“THE LINUX SCHEDULER:
A DECADE OF WASTED CORES”, EuroSys’16, Jean-Pierre Lozi, et,.al.

HIERARCHICAL LOAD BALANCING

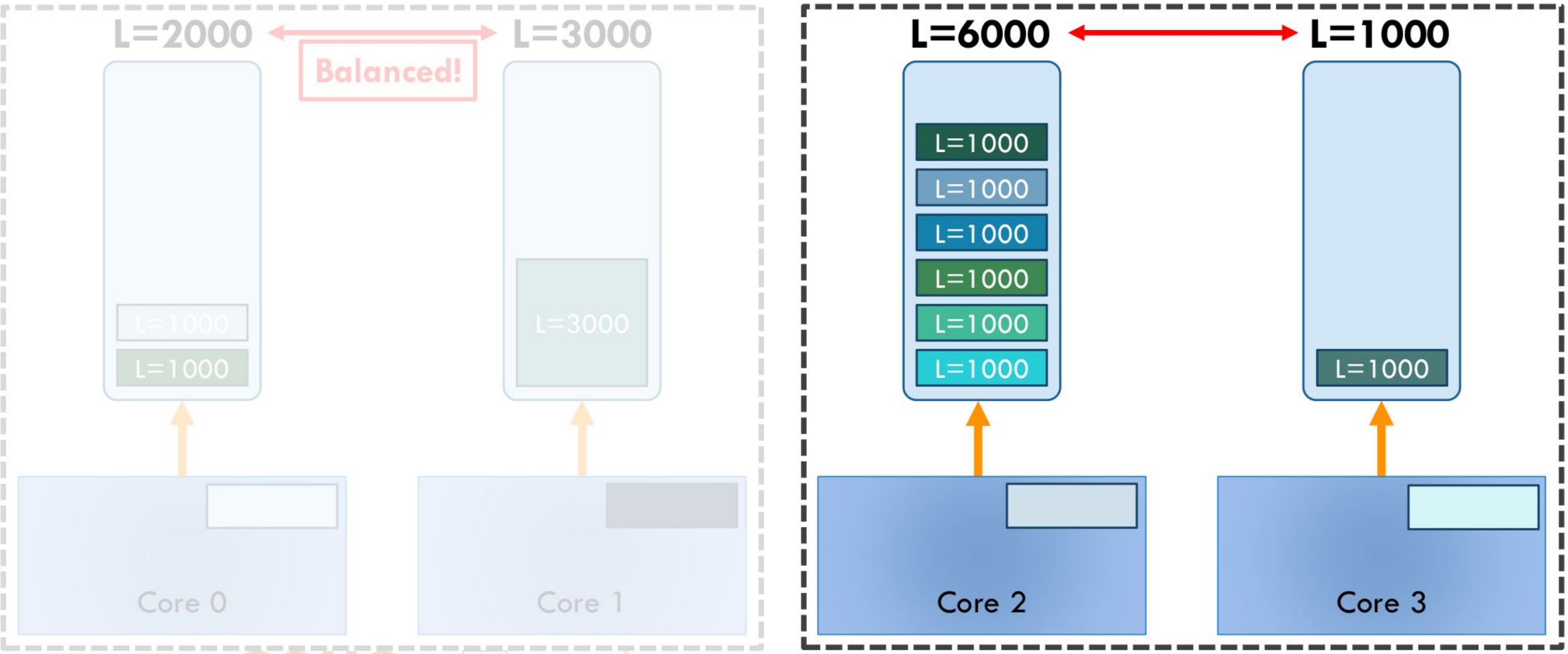


“THE LINUX SCHEDULER:
A DECADE OF WASTED CORES”, EuroSys’16, Jean-Pierre Lozi, et,.al.

rebalance_domains()

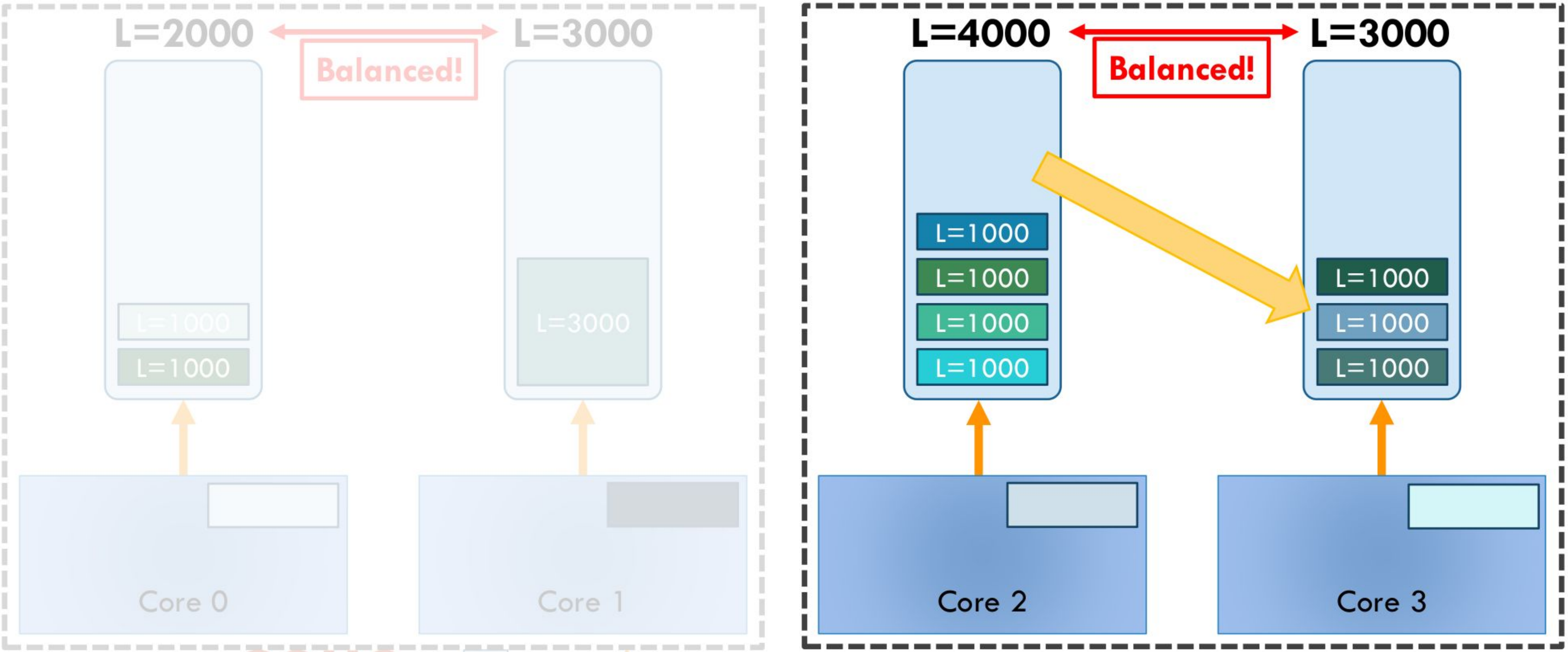
kernel/sched/fair.c

HIERARCHICAL LOAD BALANCING



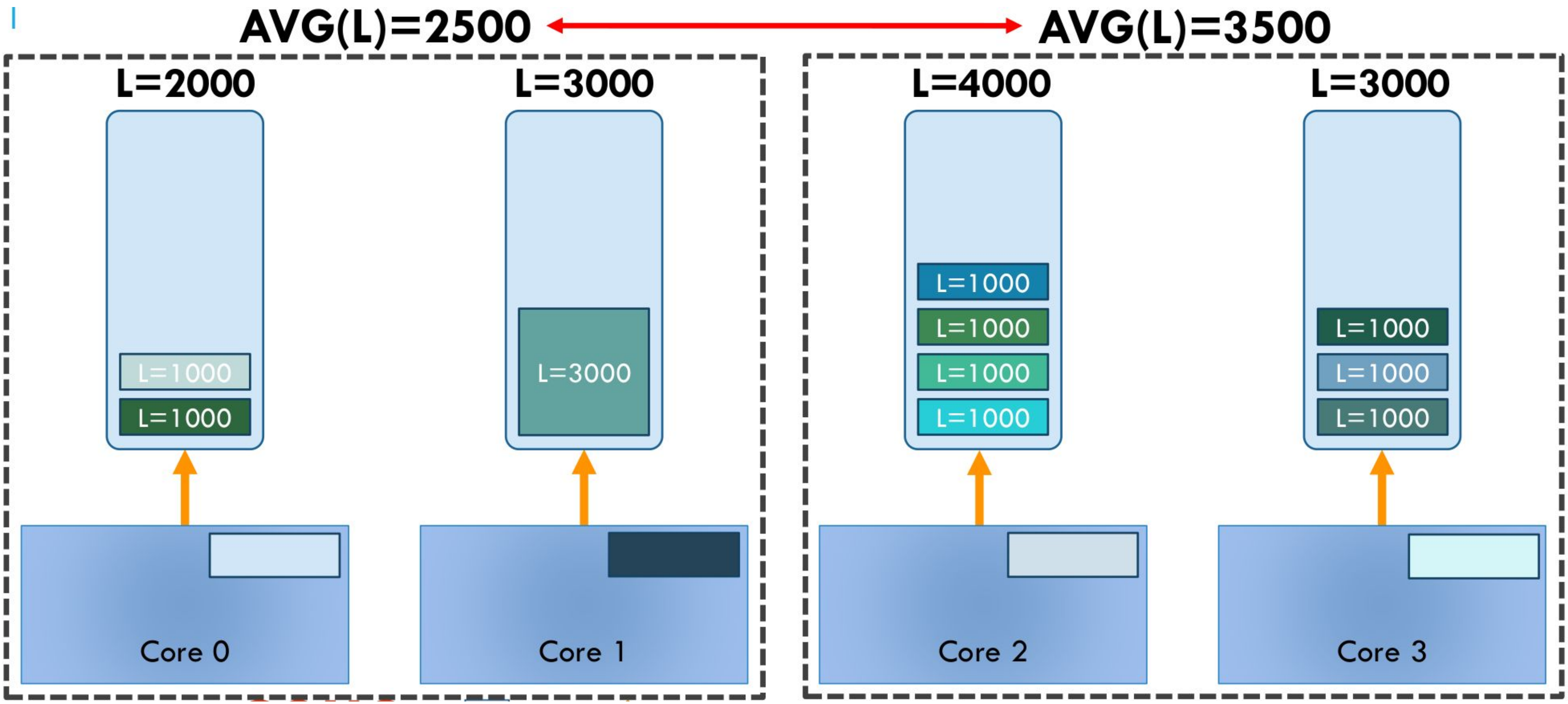
"THE LINUX SCHEDULER:
A DECADE OF WASTED CORES", EuroSys'16, Jean-Pierre Lozi, et,.al.

HIERARCHICAL LOAD BALANCING



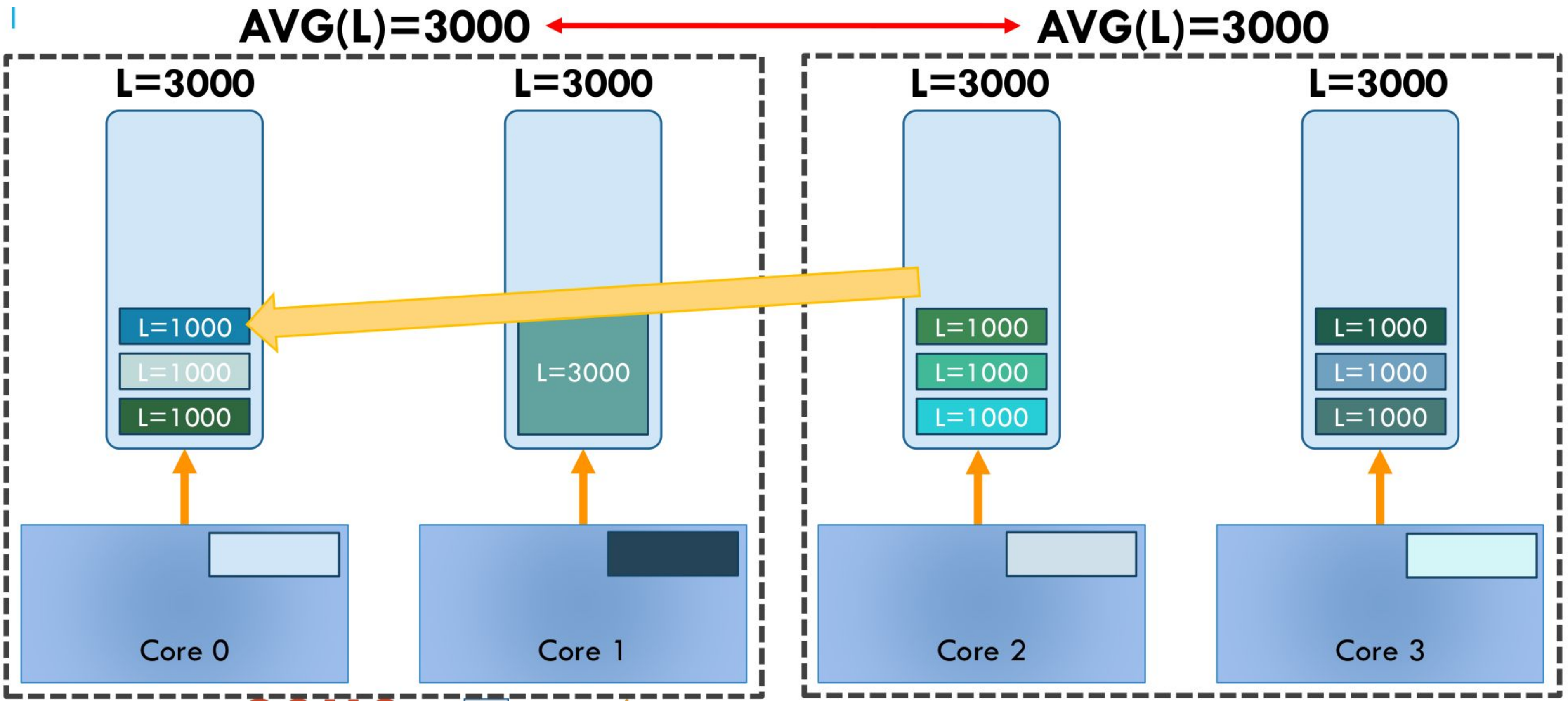
"THE LINUX SCHEDULER:
A DECADE OF WASTED CORES", EuroSys'16, Jean-Pierre Lozi, et,.al.

HIERARCHICAL LOAD BALANCING



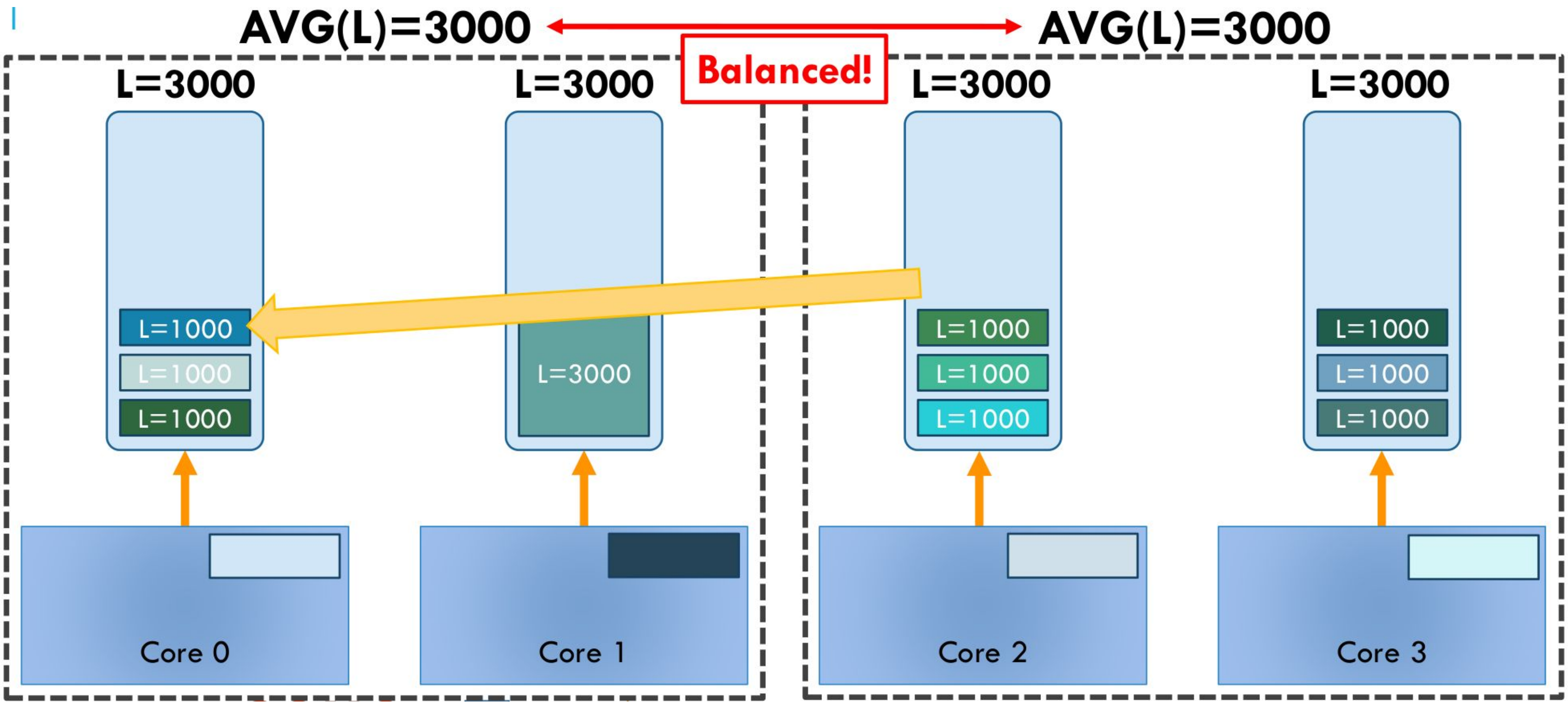
"THE LINUX SCHEDULER:
A DECADE OF WASTED CORES", EuroSys'16, Jean-Pierre Lozi, et,.al.

HIERARCHICAL LOAD BALANCING



"THE LINUX SCHEDULER:
A DECADE OF WASTED CORES", EuroSys'16, Jean-Pierre Lozi, et,.al.

HIERARCHICAL LOAD BALANCING



“THE LINUX SCHEDULER:
A DECADE OF WASTED CORES”, EuroSys’16, Jean-Pierre Lozi, et,.al.

KESL Kookmin Univ. Embedded System Lab

load_balance()

kernel/sched/fair.c

CFS load balancer

- **Problem**
  - How much is a process loading the system **right now**?
  - a bursty or a steady task?
  - a CPU-intensive or an I/O-bound task?
  - It does not matter for scheduling
  - It does matter for **load balancing**

CFS Per-entity load tracking feature

- **Problem**
  - How much is a process loading the system **right now**?
  - a bursty or a steady task?
  - a CPU-intensive or an I/O-bound task?
  - It does not matter for scheduling
  - It does matter for **load balancing**

- **Solution**
  - Linux added a **CFS Per-entity load tracking feature** to estimate a task load.

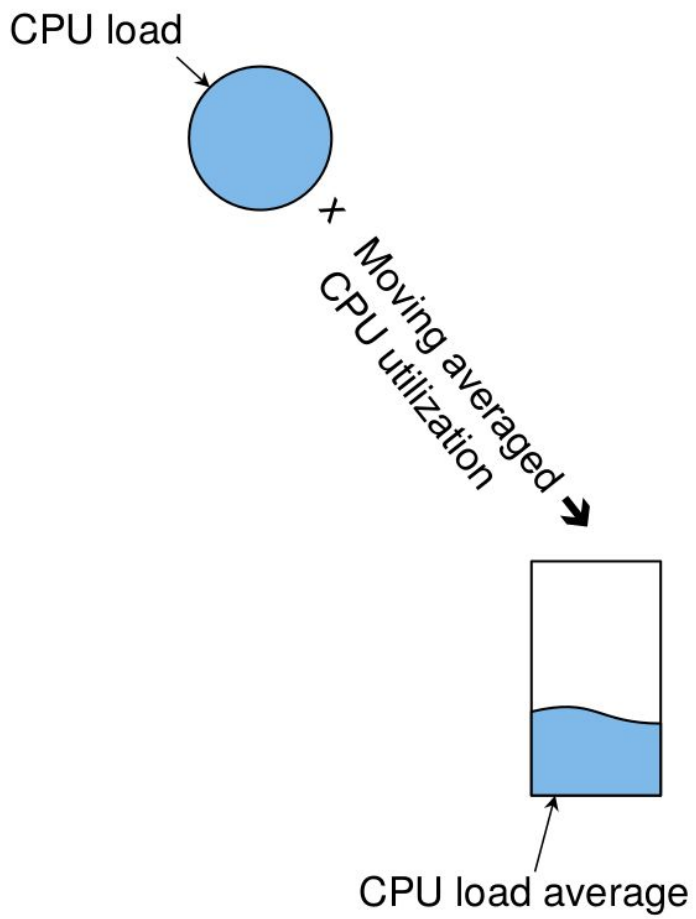
CFS load

- **Load of a process is important in scheduling**
  - which is required during load balancing.

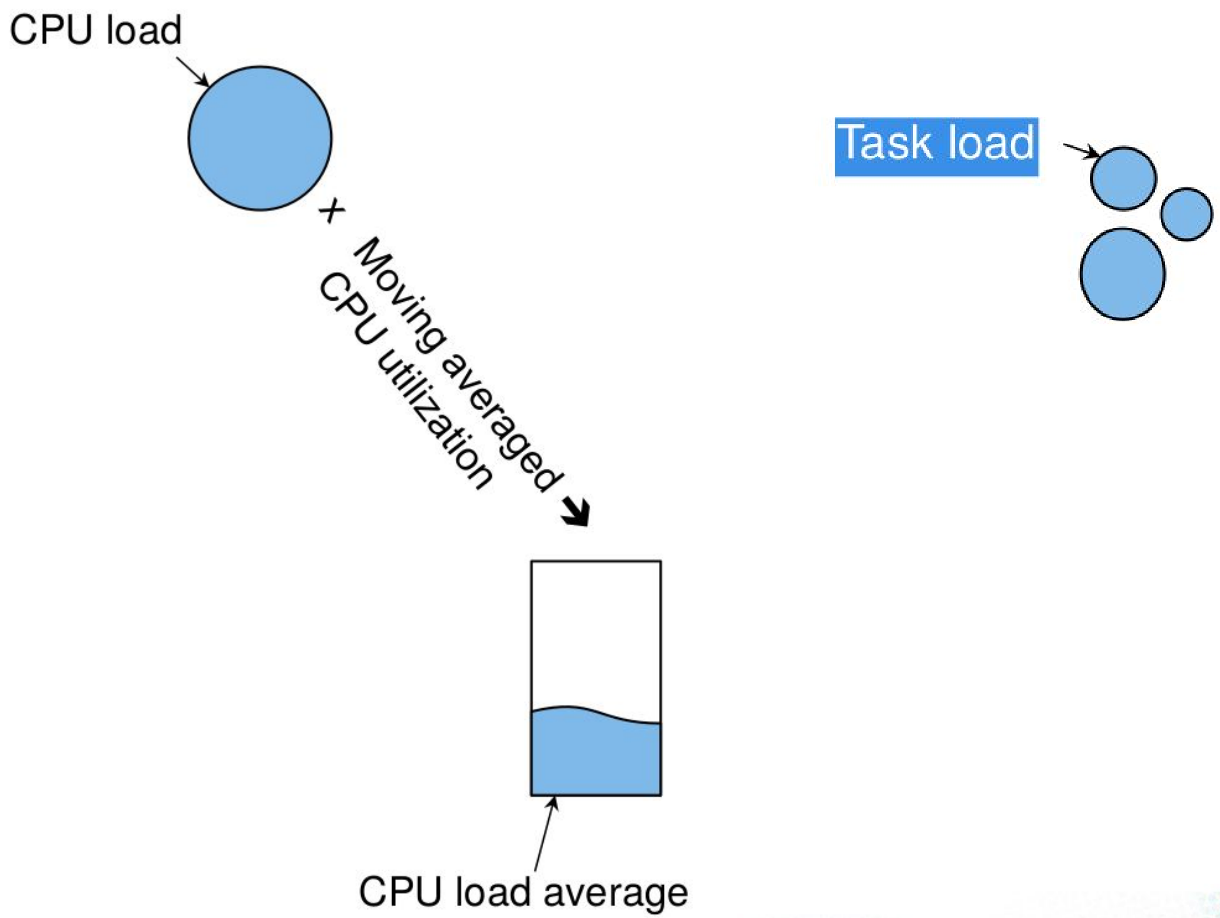
- **CFS's load**
  - Thread's **weight** and its **average CPU utilization**

- **Load of CPU have been the sum of the load of all the scheduling entities.**

CFS load



CFS load



"Energy Aware Scheduling", Byungchul Park, LGE

Per-entity load avg

- **Scheduling is always at the granularity of sched_entity**

- **A single task becomes a scheduling entity**

```
struct sched_avg {
    u64 last_update_time, load_sum;
    u32 util_sum, period_contrib;
    unsigned long load_avg, util_avg;
};
```

- **Each scheduling entity contains a runqueue.**

load tracking

- Formula gives the most weight to the most recent load
- *y* has been chosen so that y^{32} (32ms) is equal to 0.5
  - $y^0 = 1,$
  - $y^1 = 0.97852$
  - ...
  - $y^{32} = 0.5$
- Time is viewed as a sequence of 1ms (actually, 1024μs) periods
-

$$L = L_0 + L_1{*}y + L_2{*}y^2 + L_3{*}y^3 + \ldots$$

load tracking

- **PELT**

$$L = L_0 + L_1 * y + L_2 * y^2 + L_3 * y^3 + \ldots$$

- **CFS's load**
  - Thread's **weight** and its **average CPU utilization**
- Old version

sa.load_avg_contrib =

(sa.runnable_sum * se.load.weight) / sa.runnable_period;

-> Rewrite runnable load and utilization

- New version

  load_avg : PELT(running time + runnable time) * weight

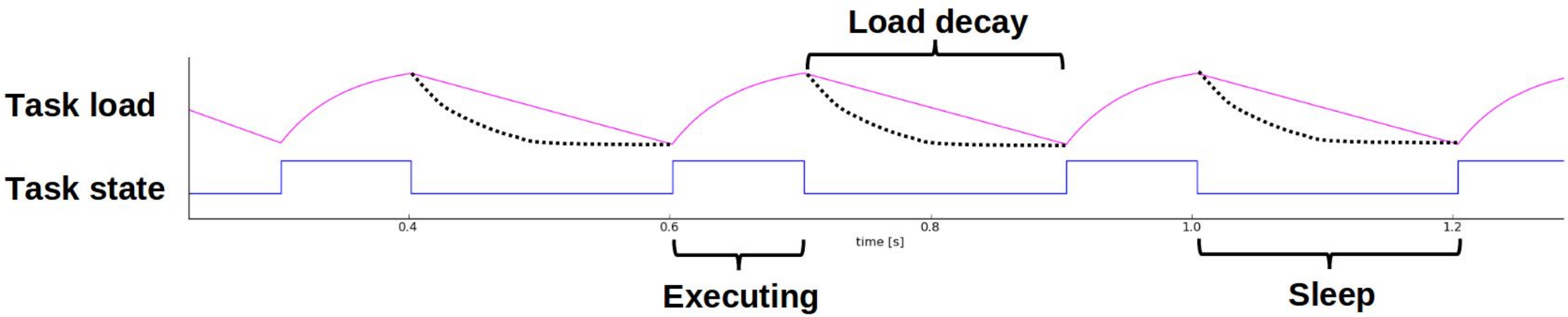
  util_avg : PELT(running time) * CPU invariant

__update_load_avg()

kernel/sched/fair.c

Per-entity load tracking

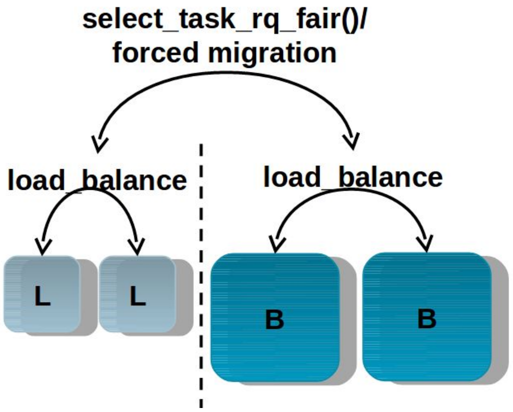
- The load is a history of time spent on the runqueue

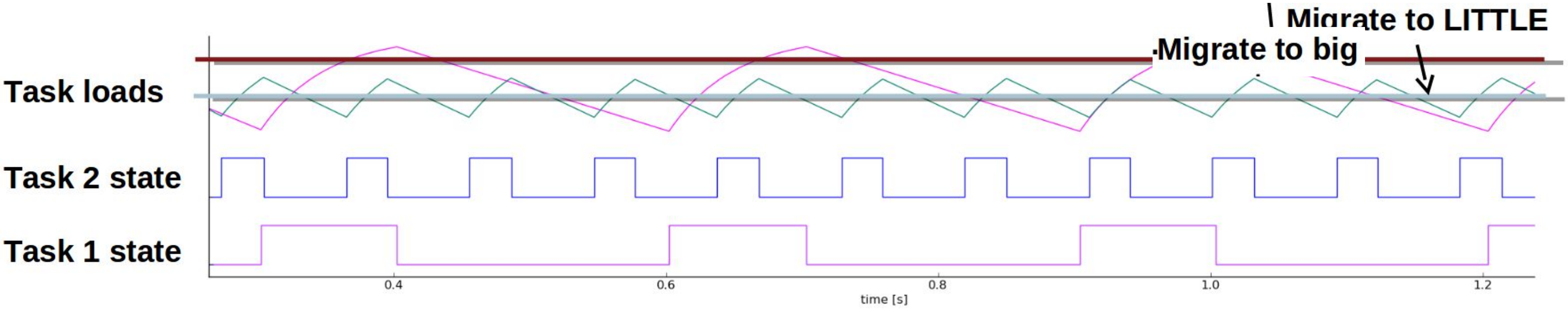


"Update on big.LITTLE scheduling experiments", ARM

Example use cases

- **HMP scheduler**





Next Step.

Energy-aware scheduling: EAS

1. **EAS features**

KESL Kookmin Univ. Embedded System Lab

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