Deep in Kernel Scheduler

Alex Shi

<Oct 2015, Nanjing>

http://www.linaro.org

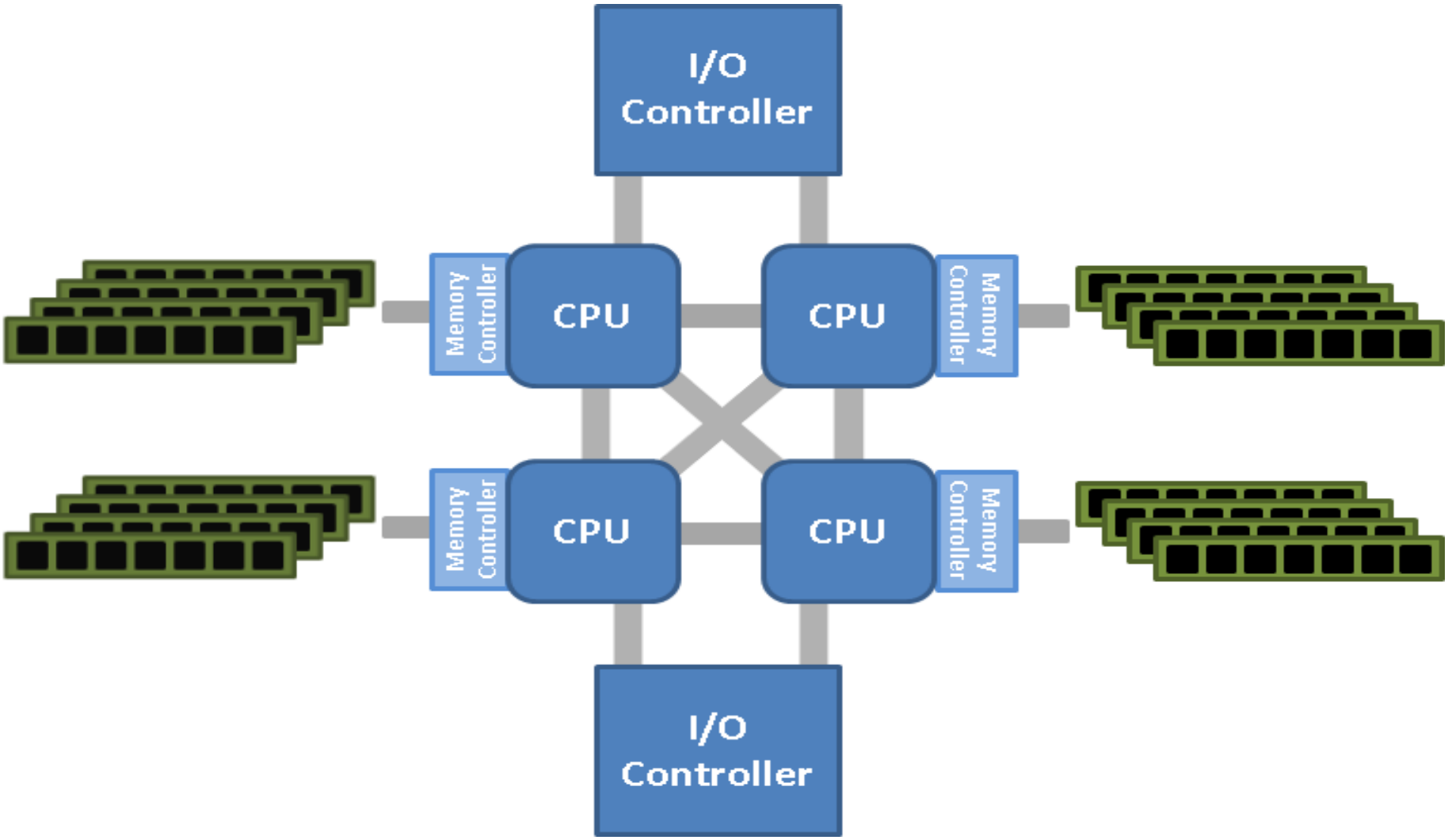
Linaro

Who am I?

- Alex Shi 时奎亮
  - Linaro – Intel – Linpus
  - Linaro stable kernel maintainer

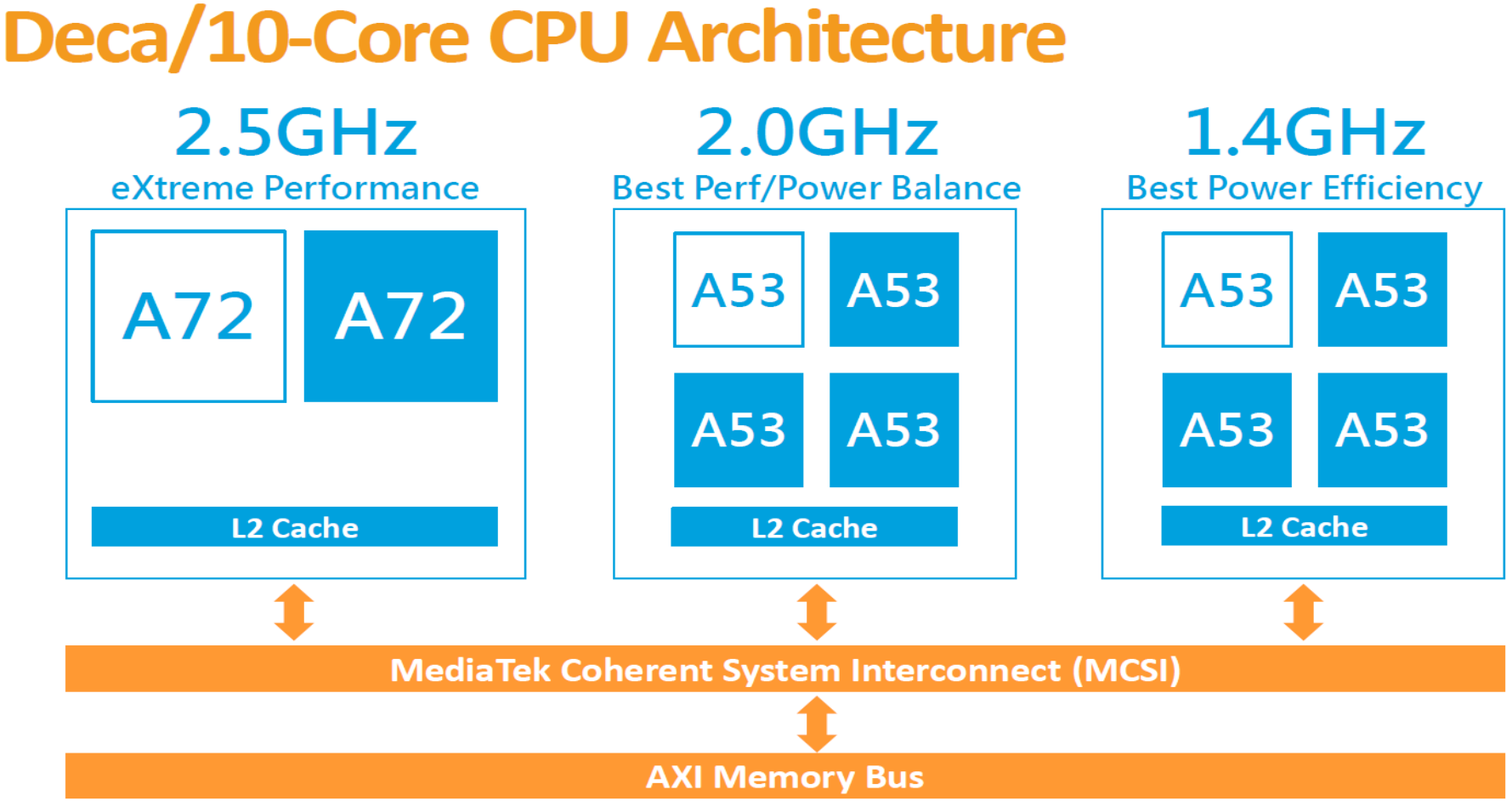
Linaro

Modern Computer Arch – Server



Linaro

Modern Computer Arch – Mobile



Scheduler

Linaro

Scheduler Service for Tasks

- Batch/Interactive Tasks
  - Complete Fair Scheduler
- Real Time Task
  - DeadLine
  - FIFO, RR

Dead Line Scheduler

- Task Organize
  - Per CPU RB tree
- Algorithm
  - Runtime, period, deadline
    - A SCHED_DEADLINE task should receive "runtime" microseconds of execution time every "period" microseconds, and these "runtime" microseconds are available within "deadline" microseconds from the beginning of the period.

Real Time Scheduler

- Task Organize
  - Per CPU links
- Priority
  - Round Robin
  - FIFO

Complete Fair Scheduler

- Ingo Molnar introduced into 2.6.23
  - Algorithm complexity
	Average	Worst case
Space	O(n)	O(n)
Search	O(log n)	O(log n)
Insert	O(log n)	O(log n)
Delete	O(log n)	O(log n)
- Inspired from Con Koliva 'fair scheduling' O(1)

Complete Fair Scheduler

- CPU Topology
  - Hierarchy Domain and Group
- Task Organize
  - Per CPU Red-black Tree

CFS CPU Topology

- CPU Domain & Group
  - Hierarchy, Tree like tree structure
  - Domain consists of CPU groups
  - Every CPU belongs to basic domain and group
  - Balancing occurs between groups in a domain
  - Per CPU

CFS CPU Organize

- CPU Domain & Group
  - On a real NUMA computer. The sched_domain, sched_group like the following:

CPU0 attaching sched-domain:

domain 0: span 0,12 **level SIBLING**

groups: 0 (cpu_power = 589) 12 (cpu_power = 589)

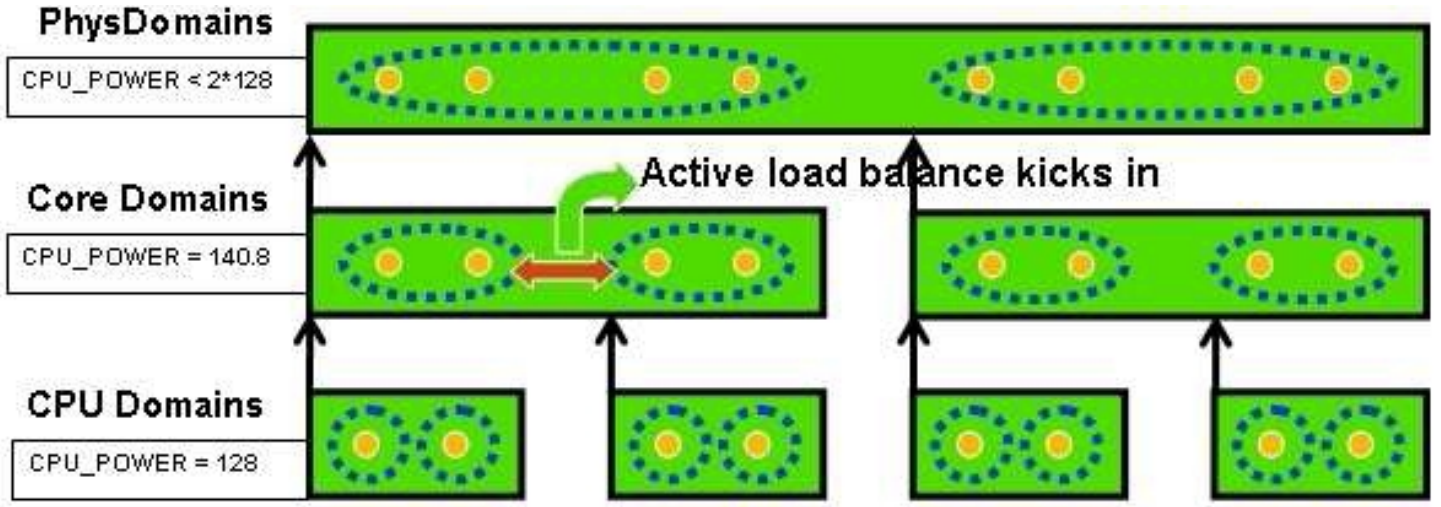
domain 1: span 0-5,12-17 **level MC**

groups: 0,12 (cpu_power = 1178) 1,13 (cpu_power = 1178) 2,14 (cpu_power = 1178) 3,15 (cpu_power = 1178) 4,16 (cpu_power = 1178) 5,17 (cpu_power = 1178)

domain 2: span 0-23 **level NODE**

groups: 0-5,12-17 (cpu_power = 7068) 6-11,18-23 (cpu_power = 7068)

CFS CPU Topology

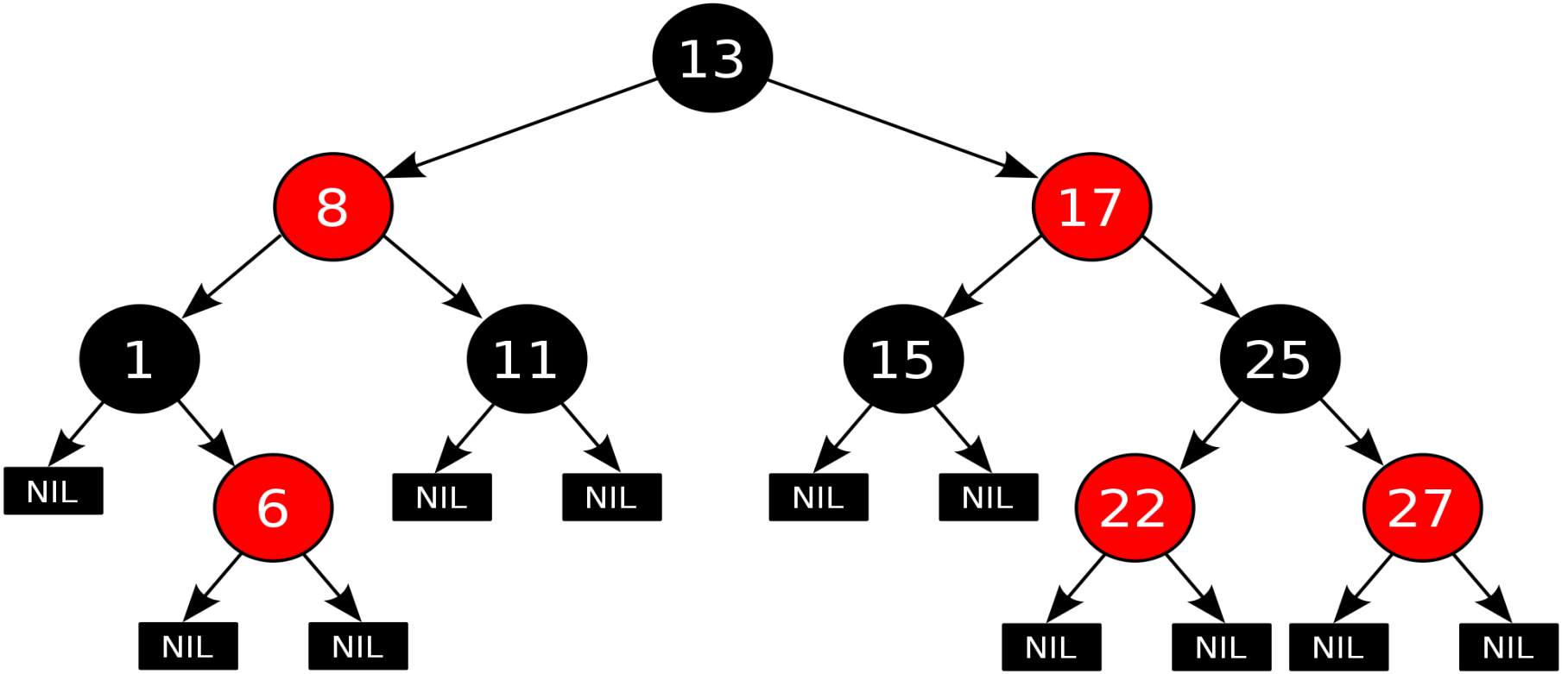


CFS Task Organize

- Task Organize
  - Per CPU Red-Black tree
  - Schedule Entities as RB tree node
  - Least Vruntime entity get running priority

CFS Task Organize

- RB tree example



Linaro

CFS Running Summary

- The left most node of the scheduling tree is chosen (as it will have the lowest spent execution time), and sent for execution.
- If the process simply completes execution, it is removed from the system and scheduling tree.
- If the process reaches its maximum execution time or is otherwise stopped (voluntarily or via interrupt) it is reinserted into the scheduling tree based on its new spent execution time.
- The new left-most node will then be selected from the tree, repeating the iteration.

CFS Details

- Se.vruntime keep growing in all life
  - Get a init value when task created
  - Increase when do entity_tick etc events
  - Modify when task enqueue/dequeue to a new cpu
  - Vruntime reflect the task's exec time in nanosec and weighted by priority

Key functions of CFS

- Fork a new task
  - Do_fork() // create a task and add it into rq->cfs_rq

copy_process-> sched_fork->__sched_fork: init p->se/numa/dl valuables

wake_up_new_task->set_task_cpu Pick a CPU for this task

wake_up_new_task->activate_task->enqueue_task(_fair)

enqueue_entity:

adding a new task P into cfs_rq and give it's initial timeslice: cfs->min_vruntime

Update_curr(cfs_rq): curr->sum_exec_runtime += now – curr->exec_start (0);

curr->vruntime += now – curr->exec_start;

__equeue_entity(): adding se into RB tree according to the vruntime of the se, the smallest vruntime put at leftmost. And rq->min_vruntime = max(rq->min_vruntime, se->vruntime);

Update_cfs_load/update_cfs_shares for all entities on this cfs_rq.

Key functions of CFS

- Wakeup tasks
  - wake_up_process()

    Spin_lock p->pi_lock,

    if p is already on_rq, just change the state as task_running, then finished.

    Set p->state = TASK_WAKING

    Call p->sched_class->task_waking(p); // vruntime -= cfs_rq->min_vruntime.

    Then select a cpu and set task on it: select_task_rq(p, SD_BALANCE_WAKE);

    Finally, call ttwu_queue(p, cpu) to added into a rq and set state to task_running.

    Collect sched statistic, if want to

    Spin_unlock p->pi_lock.

Key functions of CFS

- Schedule() //select next task to run or do load balance
  - schedule()

    try to do blk_flush_plug(prev), if need // flush block to device then release io mem.

    preempt_disable(); raw_spin_lock_irq(rq->lock);

    keep prev task in queue if it is pending on signal, else deactivate prev.

    pick_next_task(rq, prev) from highest priority sched class to lower: stop/dl_rt/fair/idle

    if next !=prev, context_switch(); //unlocks the rq

    else raw_spin_unlock_irq(&rq->lock);

    balance_callback(); //post schedule

    Preempt_enable_no_resched();

Key functions of CFS

- Time count for tasks
  - scheduler_tick() // update task slice(runtime) and check whether other action needed, load_balance, preempt

    be called by timer code(HZ frequency) or fork code, by the following func tick_nohz_hander/tick_sched_timer/tick_periodic

    Scheduler_tick

    Update clock in sched_clock_tick()

    call task_tick() ;

    If smp; trigger_load_belance

    ->task_tick(rq, curr,0) ->entity_tick:

    Update_curr();

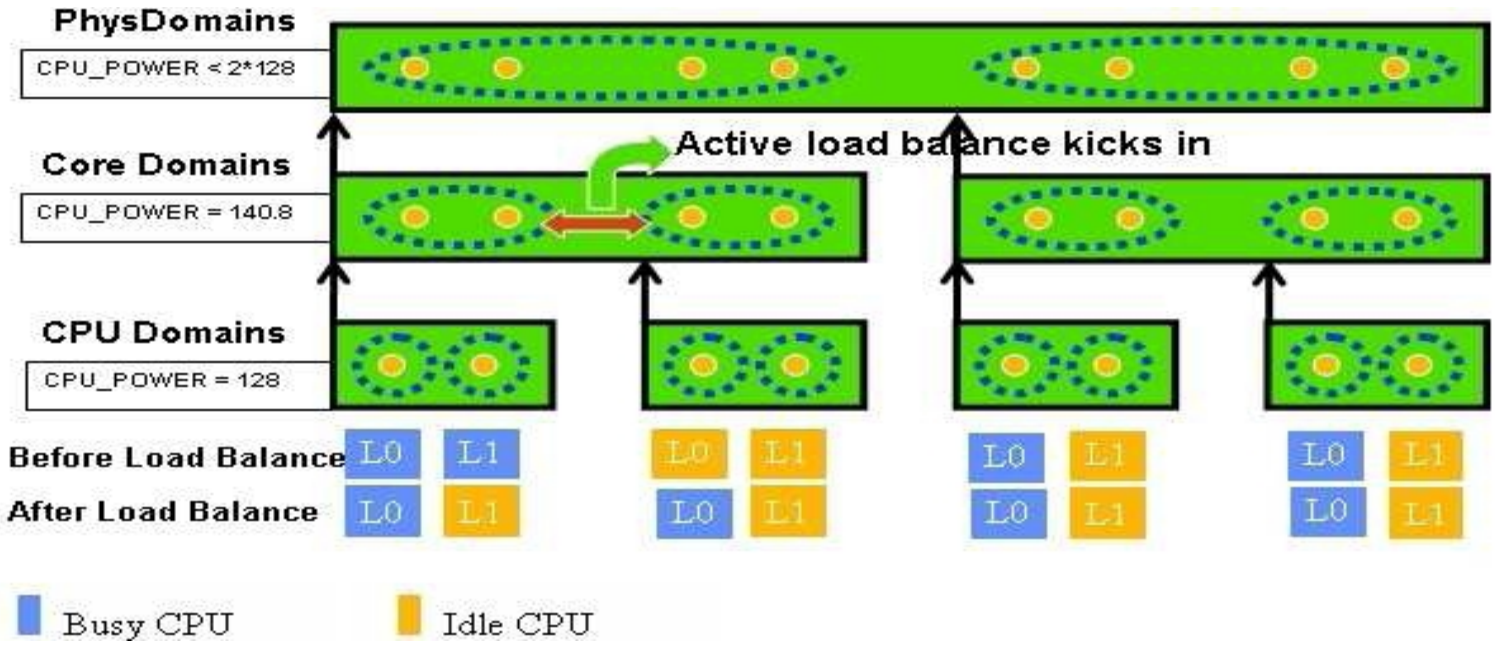
    If WAKEUP_PREEMPT enabled, try preempt the current task with a newly woken task if needed; check_preempt_tick() : if curr->sum_exec_runtime - curr->prev_sum_exec_runtime > ideal_runtime of this cfs_rq; resched_task();

Scheduler System Call

- Schedule related system call
  - nice
    - Set user nice
  - sched_setscheduler
    - set/change the scheduler policy and RT priority
  - sched_setaffinity
    - Pin task to specific CPUs
  - sched_yield (no real meaning now)

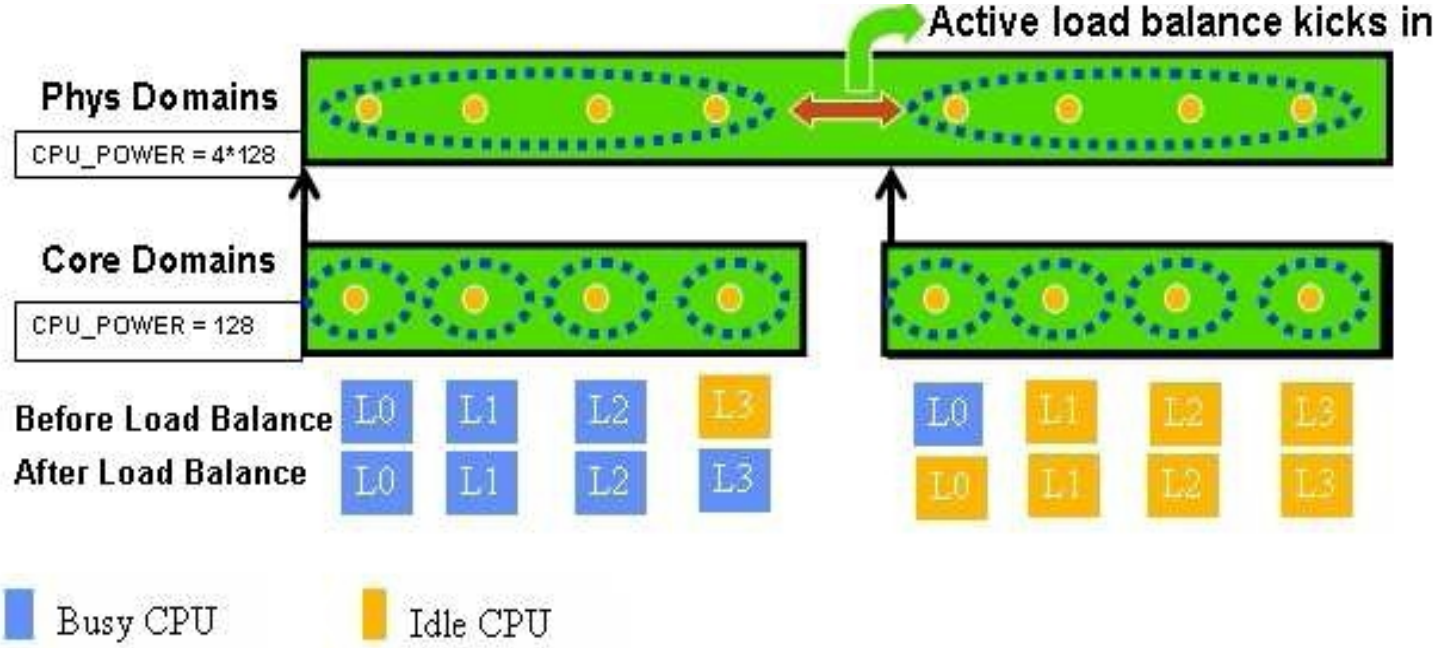
CFS Load Balance

- Performance Orientation Scheduler
  - Sparse tasks on CPUs
  - NUMA Schedule



CFS Load Balance

- Power Saving Orientation Scheduler
  - Little CPU first schedule
  - Packing first schedule



CFS Load Balance

- Balance Timing
  - Regular tick
    - Balance for self
    - Wakeup idle CPU
  - Before a CPU going to idle
  - Wakeup task

CFS Load Average

- Load decay on time.
  - L = L0 + L1*y + L2*y^2 + L3*y^3 + ...
  - Y^32 = ½; interval is 1ms
- Advantage
  - Small task packing
  - Add load predication in scheduler
- Issue
  - Predication incorrect
    - Like, a task run 100ms than idle 100ms...

CPU Idle and Schedule

- Idle balance
  - Do balance before a CPU idle, done after get tasks
- Wakeup tasks from shallowest idle CPU
  - Notice scheduler CPU idles state
  - Wakeup tasks from shallowest idle CPU

CPU Freq and Schedule

- Current issue
  - Scheduler treat low or high freq load as same
  - Same CPU determine the freq by themselves and scheduler can't know the running freq
- Freq aware scheduler
  - Get the freq-invariant task load
  - Lead to better load balance

CFS Load Balance Issue

- Current state
  - Per CPU balance
  - Bottom-Up mode, diffuse from smallest domain
  - Pull all load to self CPU and then rebalance internal
  - Issue :
    - Unnecessary task travel before a reasonable balance
    - CPU cache stain

New CFS Load Balance?

- Top-Down mode
  - Got all CPU info and load info before balance
  - Only one load balancer VS per CPU;
  - Balanced on only one time task moving
  - Task migrate times, O(1) vs O(logn), n is CPU number
- Questions
  - Decide on whole system load view
    - Power or performance orientation
  - Algorithm for all CPU load redistribution?

Thanks!