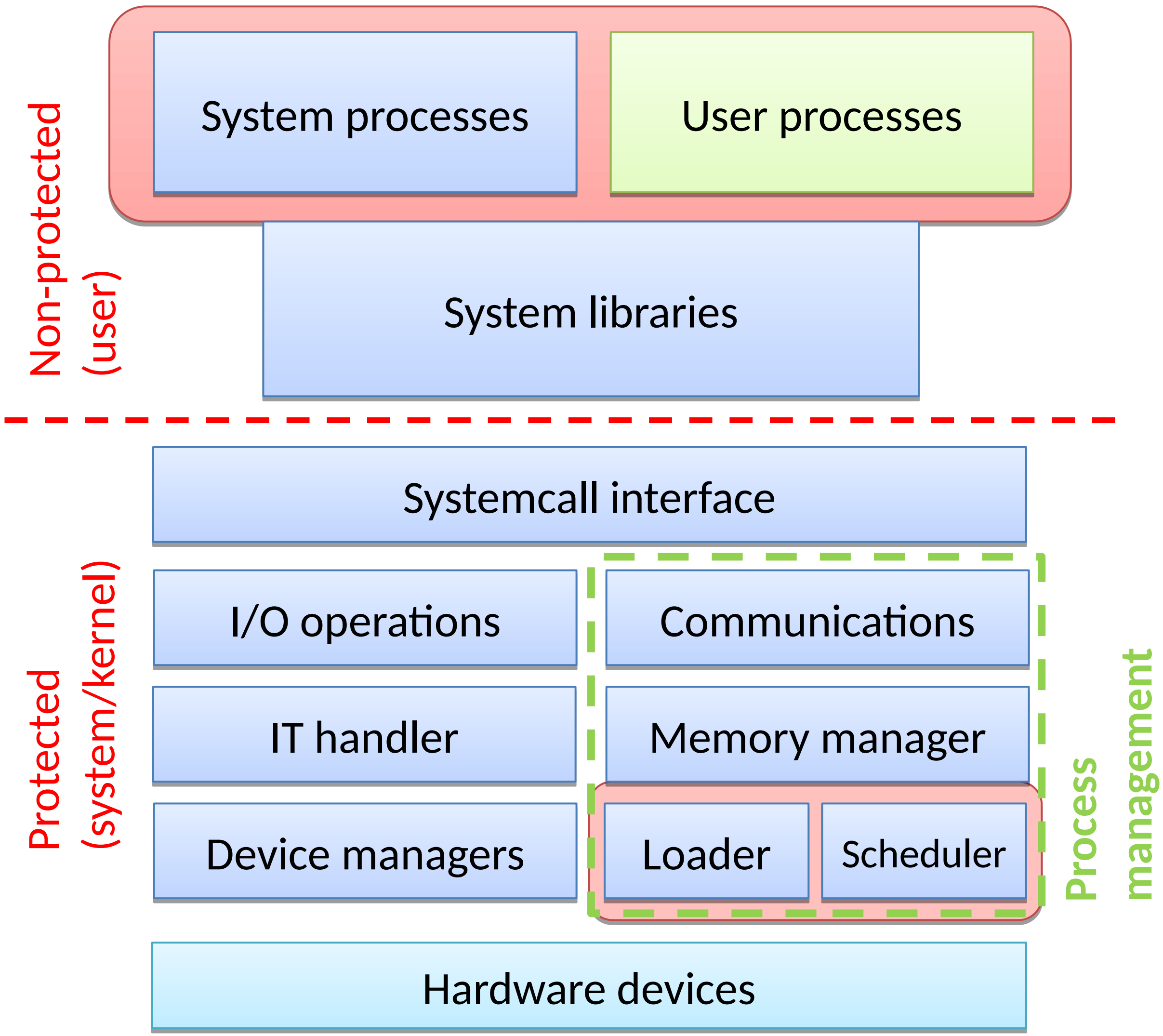
Operating Systems Internals – Task scheduling 3

Péter Györke
http://www.mit.bme.hu/~gyorke/
gyorke@mit.bme.hu

Budapest University of Technology and Economics (BME)
Department of Measurement and Information Systems (MIT)

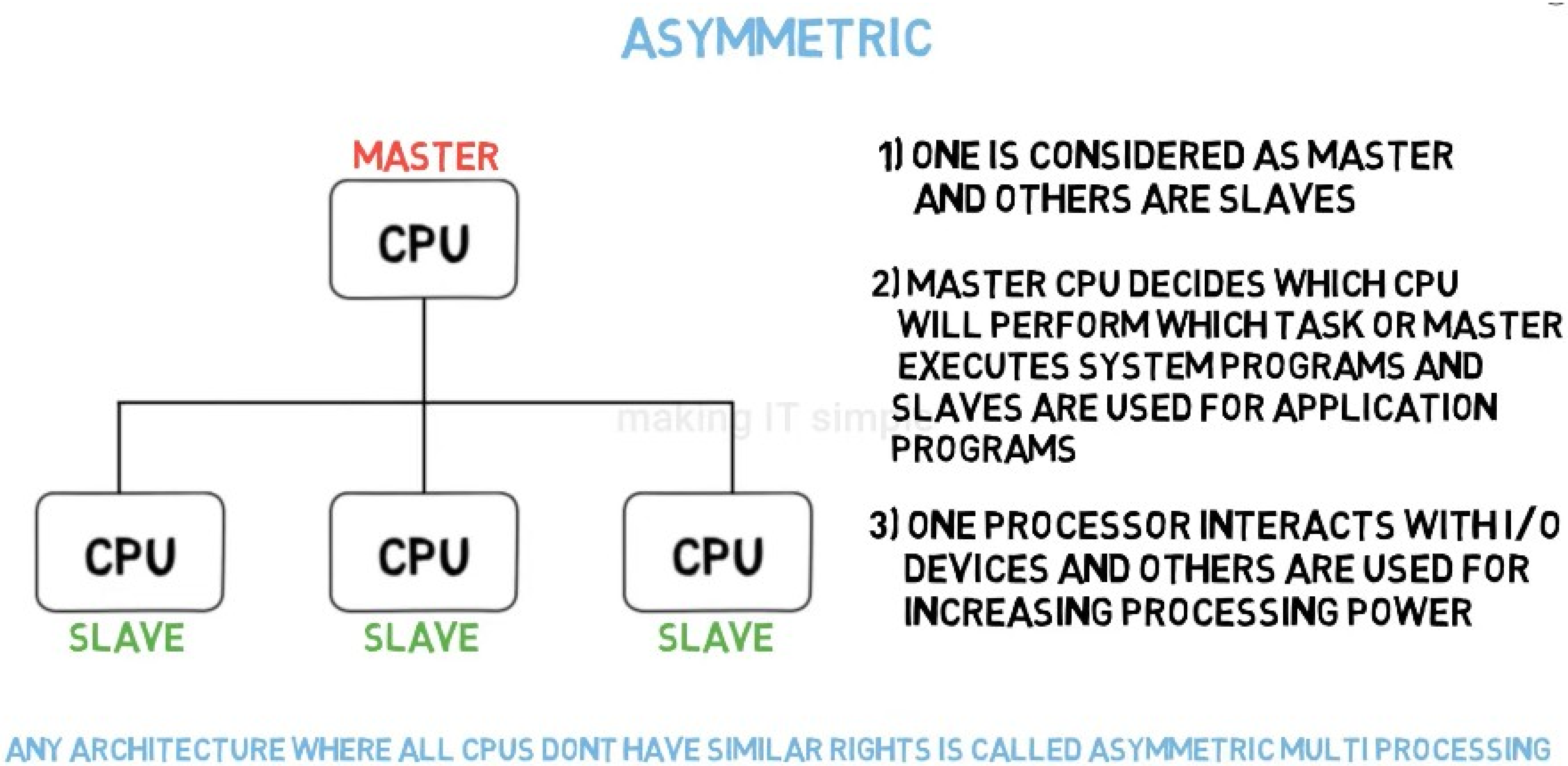
The slides of the latest lecture will be on the course page. (https://www.mit.bme.hu/eng/oktatas/targyak/vimiab00)

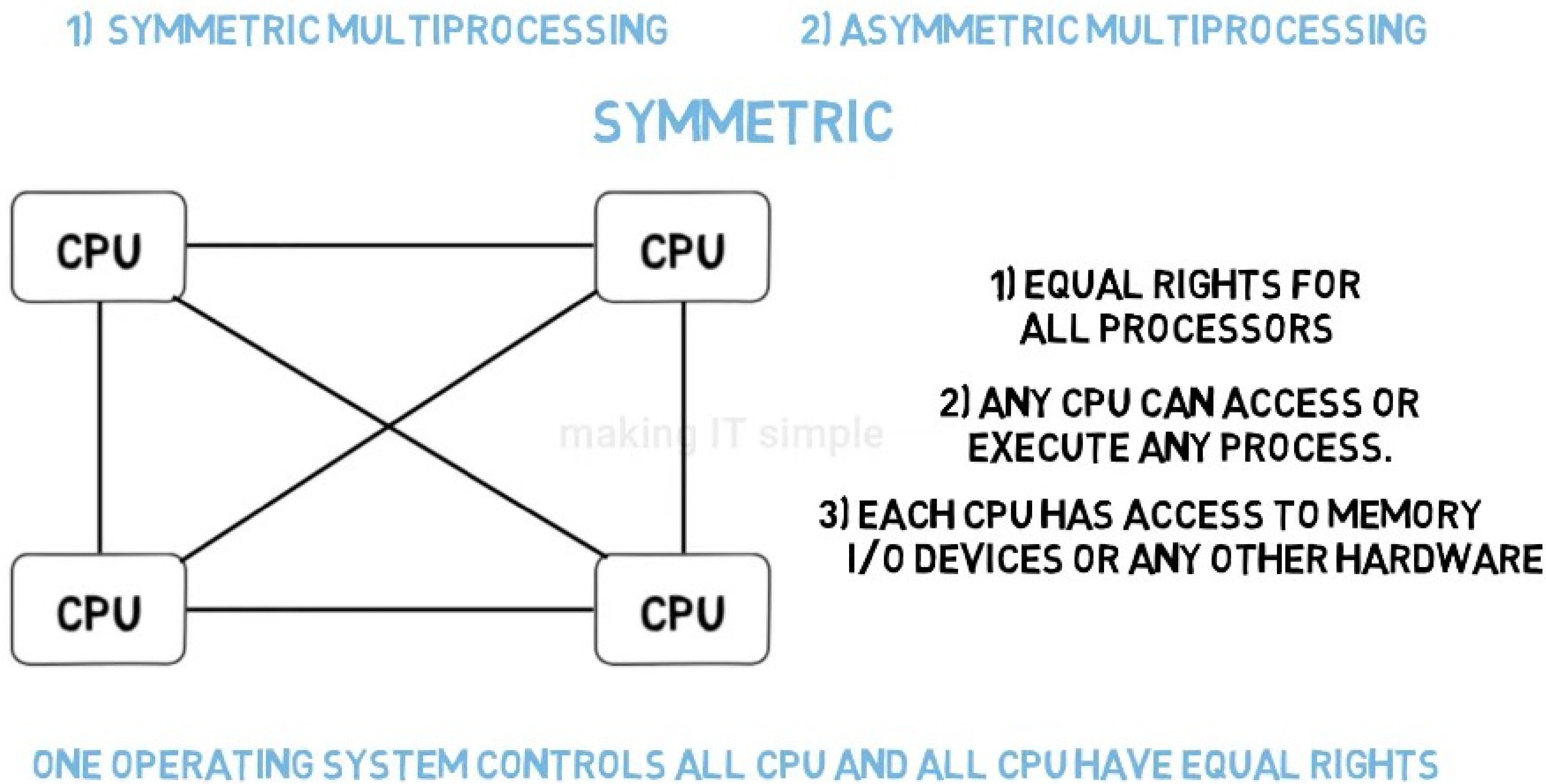
The main blocks of the OS and the kernel (recap)



Task scheduling (recap)

- Single level scheduling
  - FCFS, RR, SJF, SRTF, Priorities
  - Convey, Starvation
  - Measures: Avg. waiting time, turnaround time
- Multilevel scheduling
  - Multilevel static scheduling: fixed priorities (no queue change)
  - Multilevel dynamic scheduling: dynamic priorities (better in practice, because the tasks may change their nature)
- Multiprocessor scheduling
  - Processor affinity
  - Symmetric
    - Local queues for every processor
  - Asymmetric
    - Kernel gets a whole CPU


ASYMMETRIC
MASTER
CPU
CPU
CPU
CPU
SLAVE
SLAVE
SLAVE
1) ONE IS CONSIDERED AS MASTER AND OTHERS ARE SLAVES
2) MASTER CPU DECIDES WHICH CPU WILL PERFORM WHICH TASK OR MASTER EXECUTES SYSTEM PROGRAMS AND SLAVES ARE USED FOR APPLICATION PROGRAMS
3) ONE PROCESSOR INTERACTS WITH I/O DEVICES AND OTHERS ARE USED FOR INCREASING PROCESSING POWER
ANY ARCHITECTURE WHERE ALL CPUS DONT HAVE SIMILAR RIGHTS IS CALLED ASYMMETRIC MULTI PROCESSING



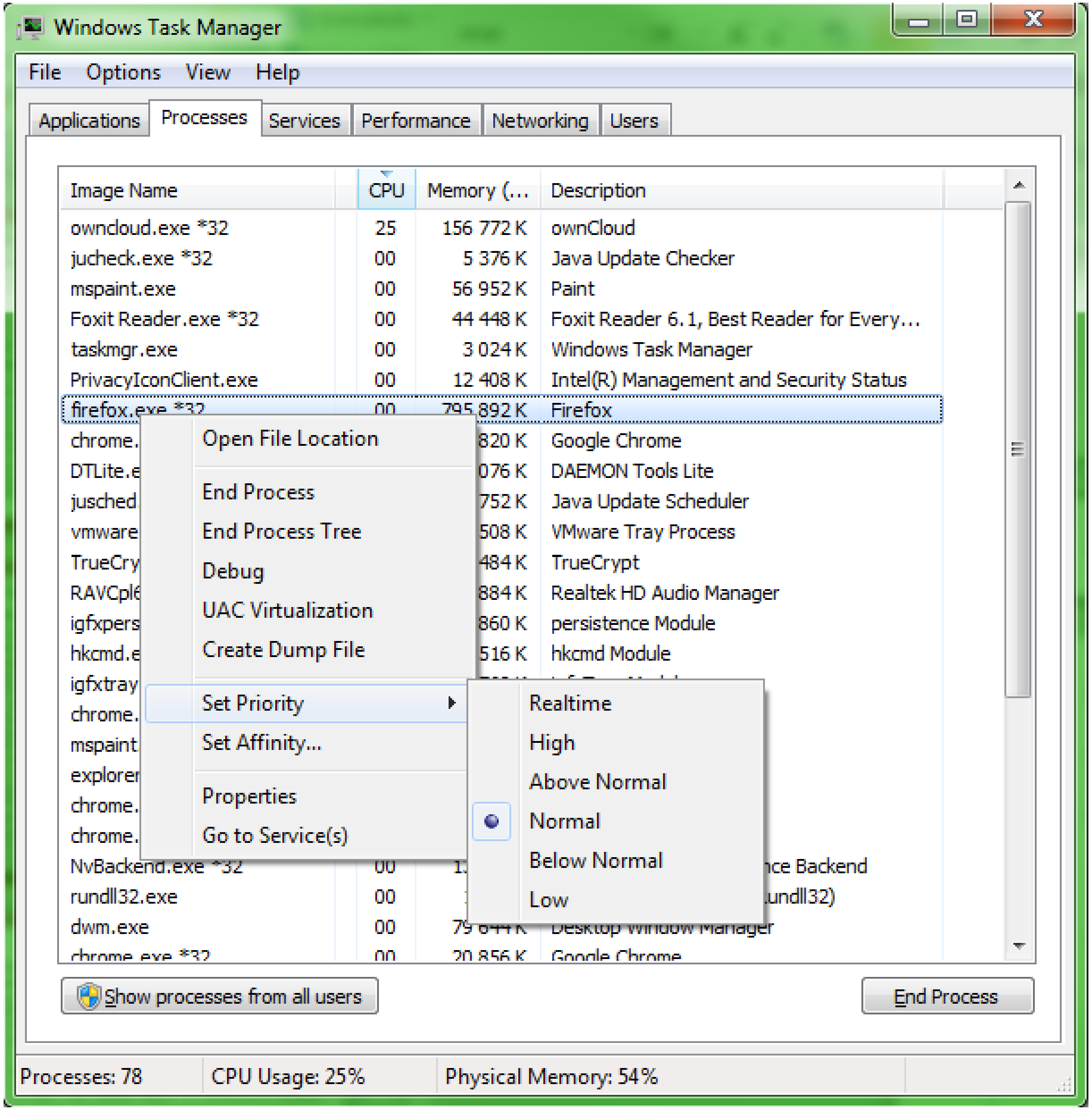
Advantages of Multi-processing:

Increased: 1- Throughput, 2- Reliability

3- Cost saving, 4- Parallel Processing

Schedulers in practice: Windows

- Symmetrical, multiprocessor with local RTR queues
  - It manages the soft and hard affinity
- Multilevel, priority, time sharing, fully preemptive scheduler
  - 16 „real-time" levels (priorities: 16-31)
  - 16 dynamic level (0-15, 0 is reserved for system)
  - The user can slightly modify the priorities
    - Low – below normal – normal – above normal – high – realtime
  - The scheduler may boost the priority in some cases
    - If a task finishes waiting for an I/O operation (better response time)
    - If a tasks gets an input from the user interface (better response time)
    - If a task is waiting for a long time (to avoid starvation)
    - Priority inversion is managed (see random boost)(Locks)
    - Multimedia Class Scheduler Service boosts the MM and gaming processes


Windows Task Manager
File Options View Help
Applications Processes Services Performance Networking Users
Image Name CPU Memory (... Description
owncloud.exe *32 25 156 772 K ownCloud
jucheck.exe *32 00 5 376 K Java Update Checker
mspaint.exe 00 56 952 K Paint
Foxit Reader.exe *32 00 44 448 K Foxit Reader 6.1, Best Reader for Every...
taskmgr.exe 00 3 024 K Windows Task Manager
PrivacyIconClient.exe 00 12 408 K Intel(R) Management and Security Status
firefox.exe *32 00 795 892 K Firefox
Google Chrome
DAEMON Tools Lite
Java Update Scheduler
VMware Tray Process
TrueCrypt
Realtek HD Audio Manager
persistence Module
hkcmd Module
Open File Location
End Process
End Process Tree
Debug
UAC Virtualization
Create Dump File
Set Priority
Set Affinity...
Properties
Go to Service(s)
Realtime
High
Above Normal
Normal
Below Normal
Low
rundll32.exe 00
dwm.exe 00
Show processes from all users
End Process
Processes: 78 CPU Usage: 25% Physical Memory: 54%

Schedulers in practice: Linux

- Before v2: traditional UNIX scheduler (as seen before)
- Before v2.4 kernel
  - Scheduling classes: real-time, non-preemptive, normal
  - Complex scheduling with O(N) (linear) complexity
  - Single scheduling queue (no multiprocessor support)
  - Kernel is non-preemptive
- Kernel v2.6
  - O(1) scheduler (as the call it)
  - Distributed queues for each processor
  - Multilevel feedback with priorities and time-sharing
  - 140 priority level: 0-99 „real-time” static, 100-139 time-sharing, dynamic
  - „active” has time slice, „expired” (preempted) queues on every level
  - Priority is recalculated when moving from active to expired
  - If the active queue is empty, the expired become active
  - Tasks waiting for a long time gets a little „bonus”
- From kernel v2.6.23 CFS (Completely fair scheduler)
  - See next slide...

O(1) --- constant complexity – not depending on n

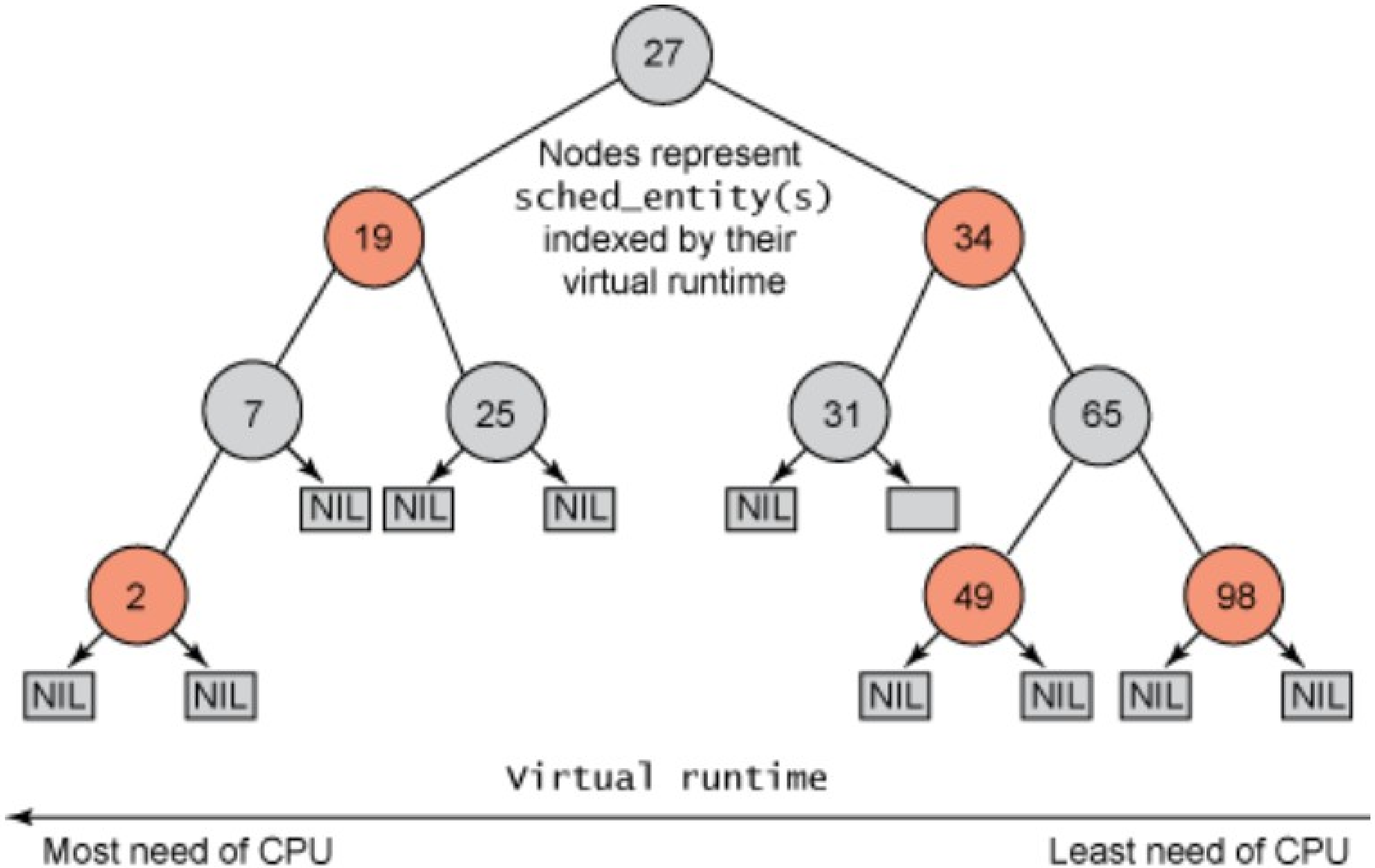
Linear O(n) — performance becomes less efficient as data grows.

Linux CFS (Completely Fair Scheduler)- Ingo Molnár

- Instead of queues, the tasks are stored in a special tree structure
  - Red-black tree: self balancing search tree
    - Smaller values on the left, higher values on the right
    - O(log *n*) Complexity
- Priority calculation
  - Virtual runtime (vruntime) assigned to each task
    - It tries to distribute runtime in a fair way
    - During execution vruntime is increasing, depending of the **priority**
  - The red-black tree is constructed based on vruntime
  - The tasks with
    - Small vruntime: gets the CPU earlier

- if process has run for t ms, then

  vruntime += t * (weight based on nice of process)

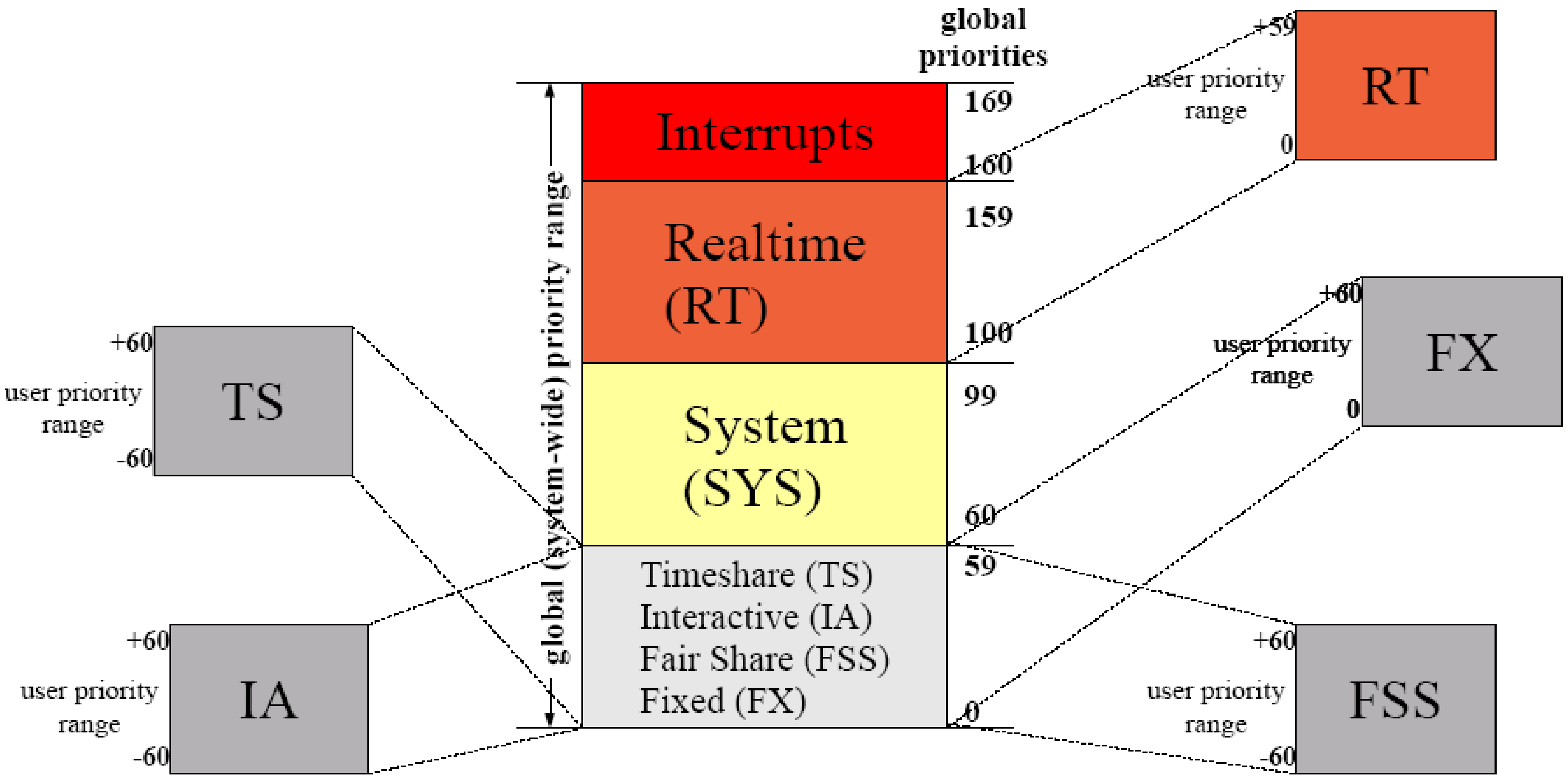

27
Nodes represent
sched_entity(s)
indexed by their
virtual runtime
19
34
7
25
31
65
NIL
NIL
NIL
NIL
2
49
98
NIL
NIL
NIL
NIL
NIL
NIL
Virtual runtime
Most need of CPU
Least need of CPU

Solaris UNIX

- Properties
  - The scheduling is thread based
  - Fully preemptive kernel
  - Multiprocessor systems and virtualization are supported

- Multiple scheduling classes
  - Time-sharing (TS): scheduling by waiting/running times
  - Interactive (IA): like above, but enhance the priority of the active window
  - Fixed priority (FX)
  - Fair share (FSS): CPU resources are assigned to a process group
  - Real-time (RT): this level provides the shortest latency
  - Kernel threads (SYS): highest level except the RT

Scheduling classes of Solaris UNIX



Computation examples for simple schedulers

- Schedulers
  - FCFS: simple, based on FIFO (cooperative)
  - RR: time-sharing (after the time slice is up, gets the next task from FIFO)
  - SJF: ordering tasks by estimated CPU-burst (cooperative)
  - SRTF: preemptive SJF
  - PRI: ordering tasks by priority

- Measuring numbers for evaluation
  - Response time
  - Waiting time – elapsed time in **non** running state (waiting, ready to run)
  - Execution time – elapsed time in running state

Simple scheduler examples

- Show the operation of simple schedulers on Gantt chart and calculate the avg. waiting time!
  - Methods: FCFS, SJF, RR (TS=2, TS=6)
  - Tasks [Start, CPU-burst]
    - A [0, 6]
    - B [0, 3]
    - C [0, 3]

FCFS example

SJF example

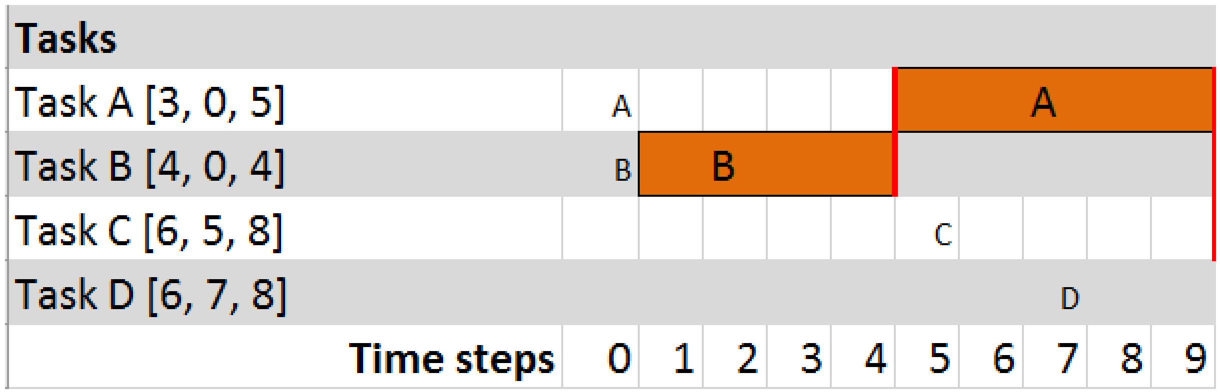
RR (TS=2) example

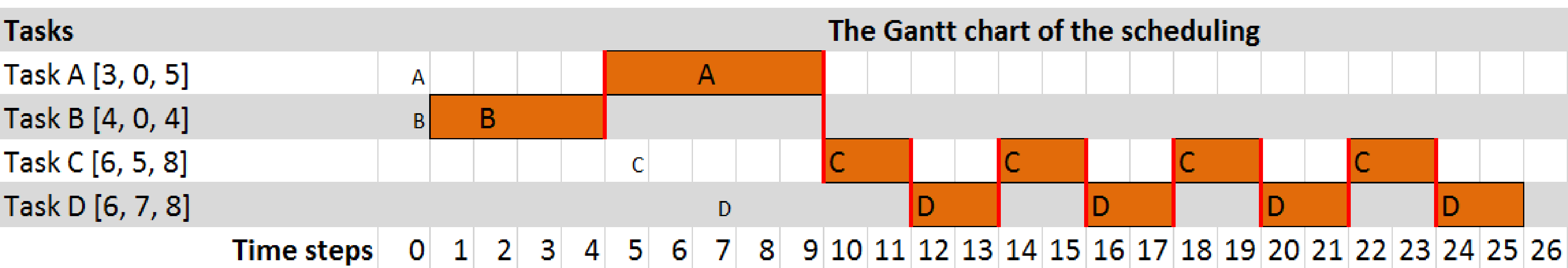
RR (TS=6) example

Practice example for static multilevel scheduler

- The scheduler
  - Static priorities: 0..9 (0 is the highest)
  - Levels:
    - Level 1: 0-4 non preemtive SJF
    - Level 2: 5-9 preemtive RR, time slice: 2
  - To calculate: running order and avg. waiting time
- Paramters of the tasks
  - Priority
  - Starting (arrival) time
  - CPU-burst
- Tasks [Pri, Sta, CPU]
  - A [3, 0, 5]
  - B [4, 0, 4]
  - C [6, 5, 8]
  - D [6, 7, 8]

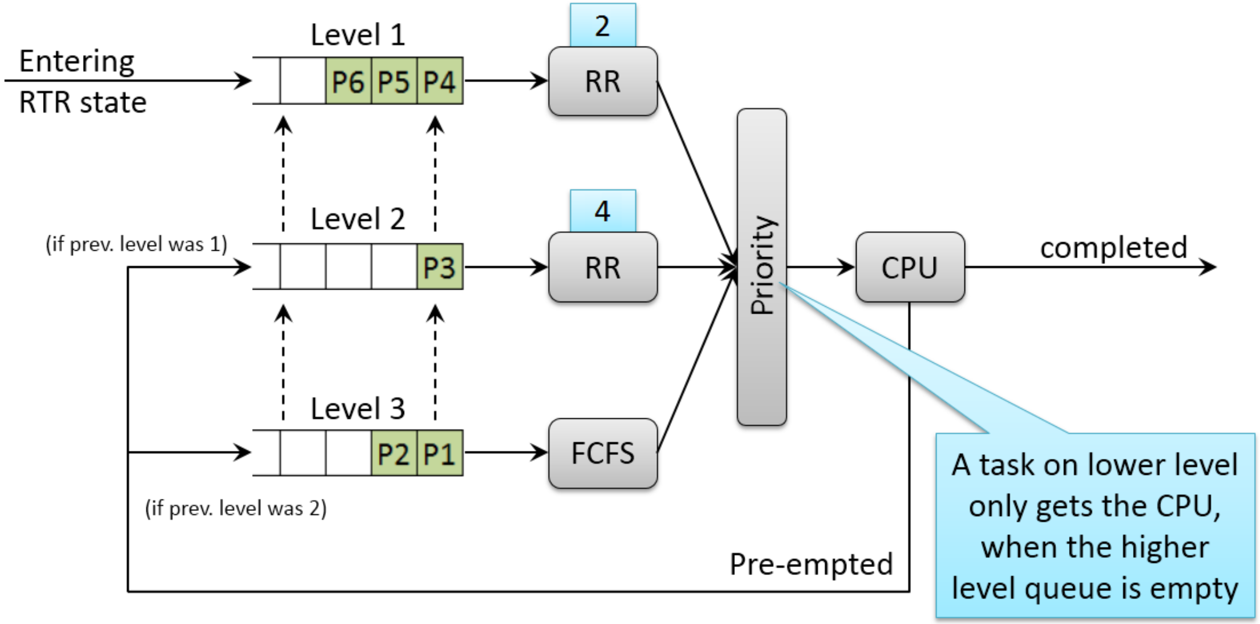
Gantt chart (static multilvel)





MFQ (Multilevel Feedback Queue) example

- 3 levels:



- Tasks [Start, CPU-burst]
  - A [0, 5]
  - B [0, 5]
  - C [3, 13]
  - D [10, 11]

Gantt chart (MFQ)