Handling of Tasks in Operating Systems Supporting Multiprogramming

Tamás Kovácsházy, PhD

3rd topic

1. Example of simple scheduling
2. Complex scheduling algorithms and scheduling in multiprocessor systems

Méréstechnika és Információs Rendszerek Tanszék

Lesson 1.

- Comparison of simple CPU scheduling algorithms
- FIFO, SJF, SRTF, RR algorithms are compared
- Deterministic analytical modeling
  - Given arrival time for all tasks
  - Given CPU bursts for all tasks
    - For the sake of simplicity only the first CPU burst is considered
  - Overhead is considered 0ms
    - Scheduling and context switch take 0ms
- Assignment: Compute the typical metrics of scheduling algorithms

Lesson 1., Known properties of the system

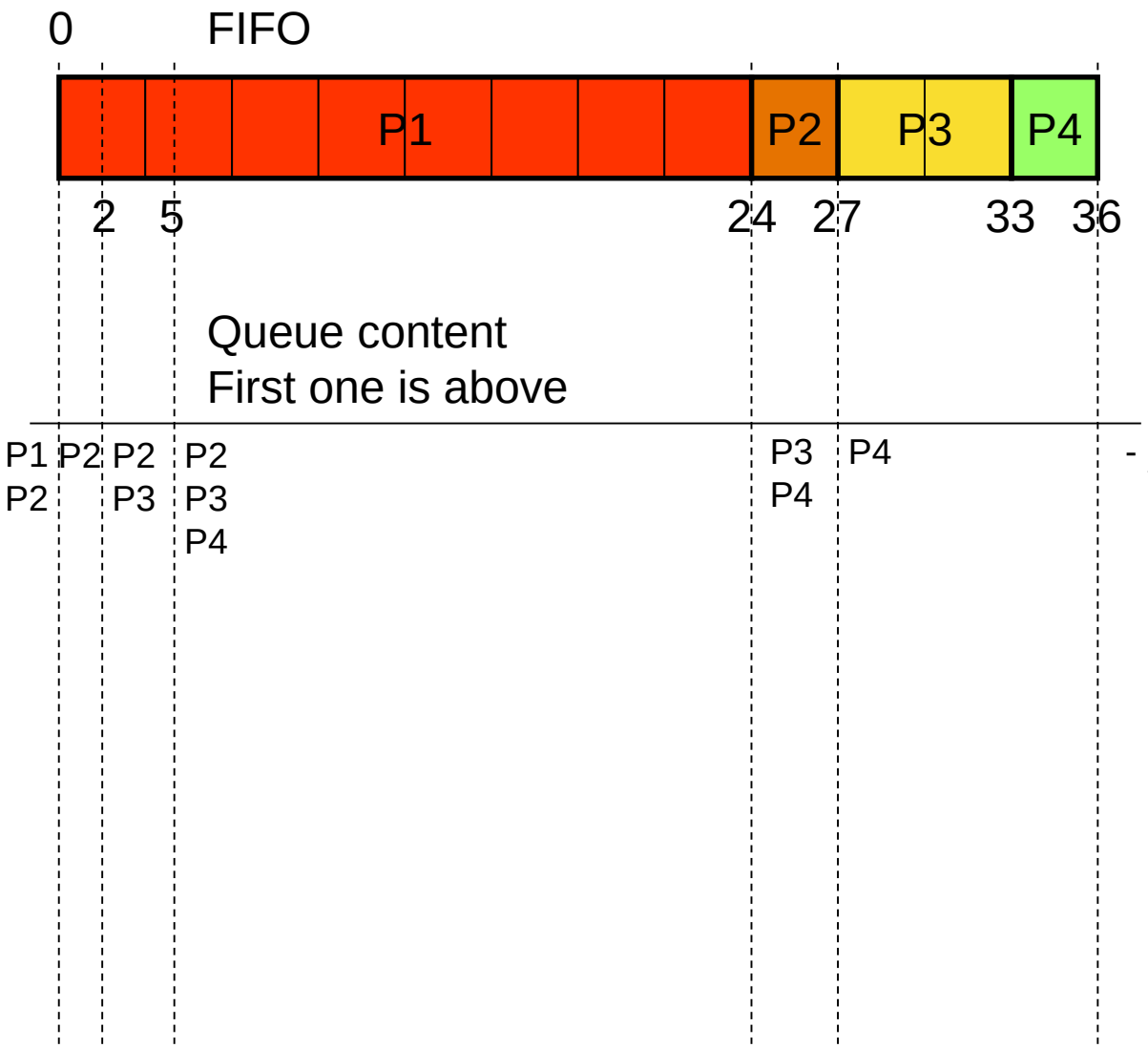
Task	Arrival time (ms)	CPU burst (ms)
P1	0	24
P2	0	3
P3	2	6
P4	5	3

RR time slice: 4 ms

Beware of errors

- The slides have been presented several times, but errors may be present in them even now.
- Sometimes I make errors while presenting them, and sometimes it is conscious to check your attention.
- So pay attention and try to follow me while I present the solution…

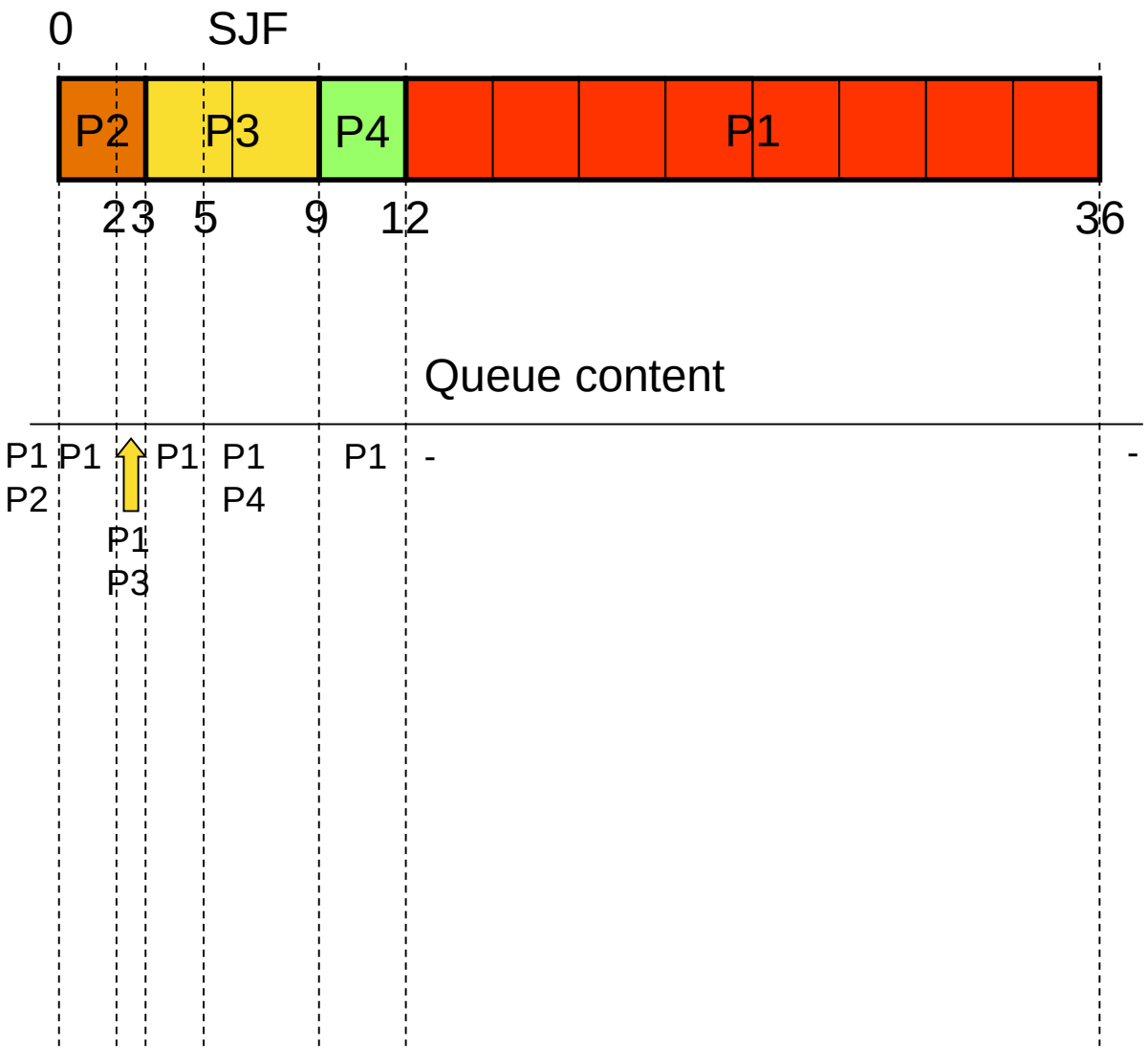
Lesson 1., FIFO Gannt diagram (chart)



Task	Arrival time (ms)	CPU burst (ms)
P1	0	24
P2	0	3
P3	2	6
P4	5	3

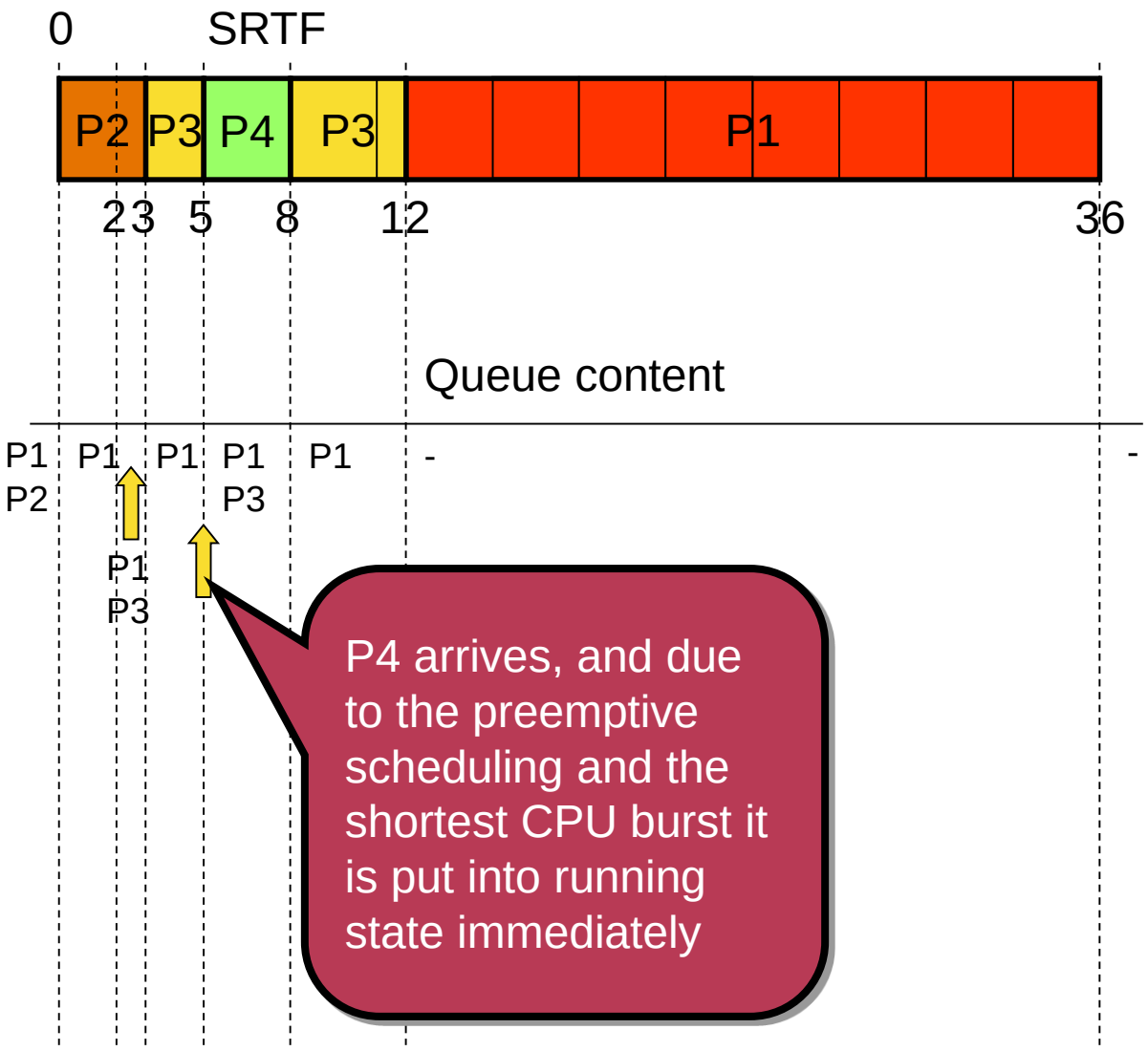
Empty queue

Lesson 1., SJF Gannt diagram (chart)



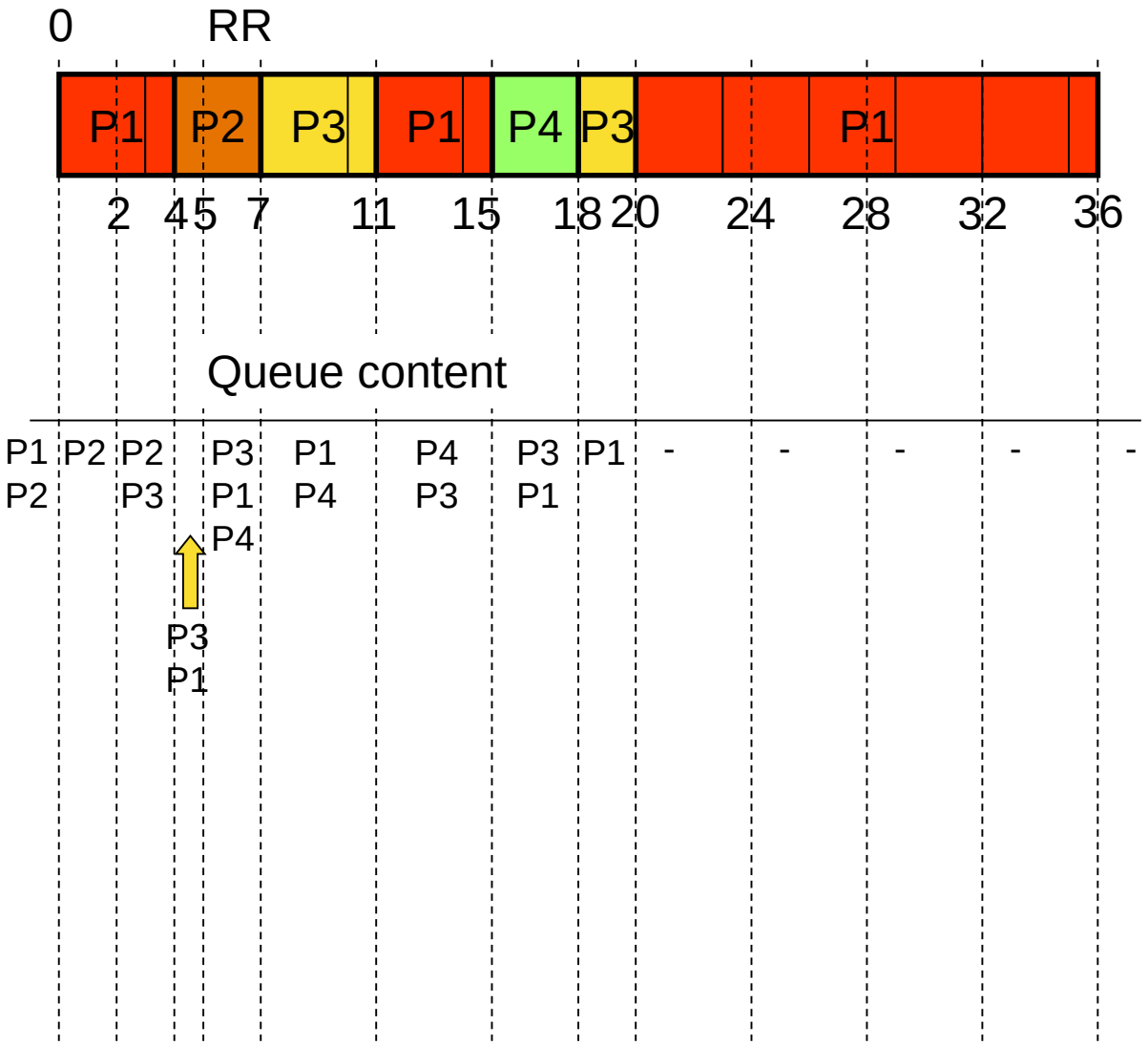
Task	Arrival time (ms)	CPU burst (ms)
P1	0	24
P2	0	3
P3	2	6
P4	5	3

Lesson 1., SRTF Gannt diagram (chart)



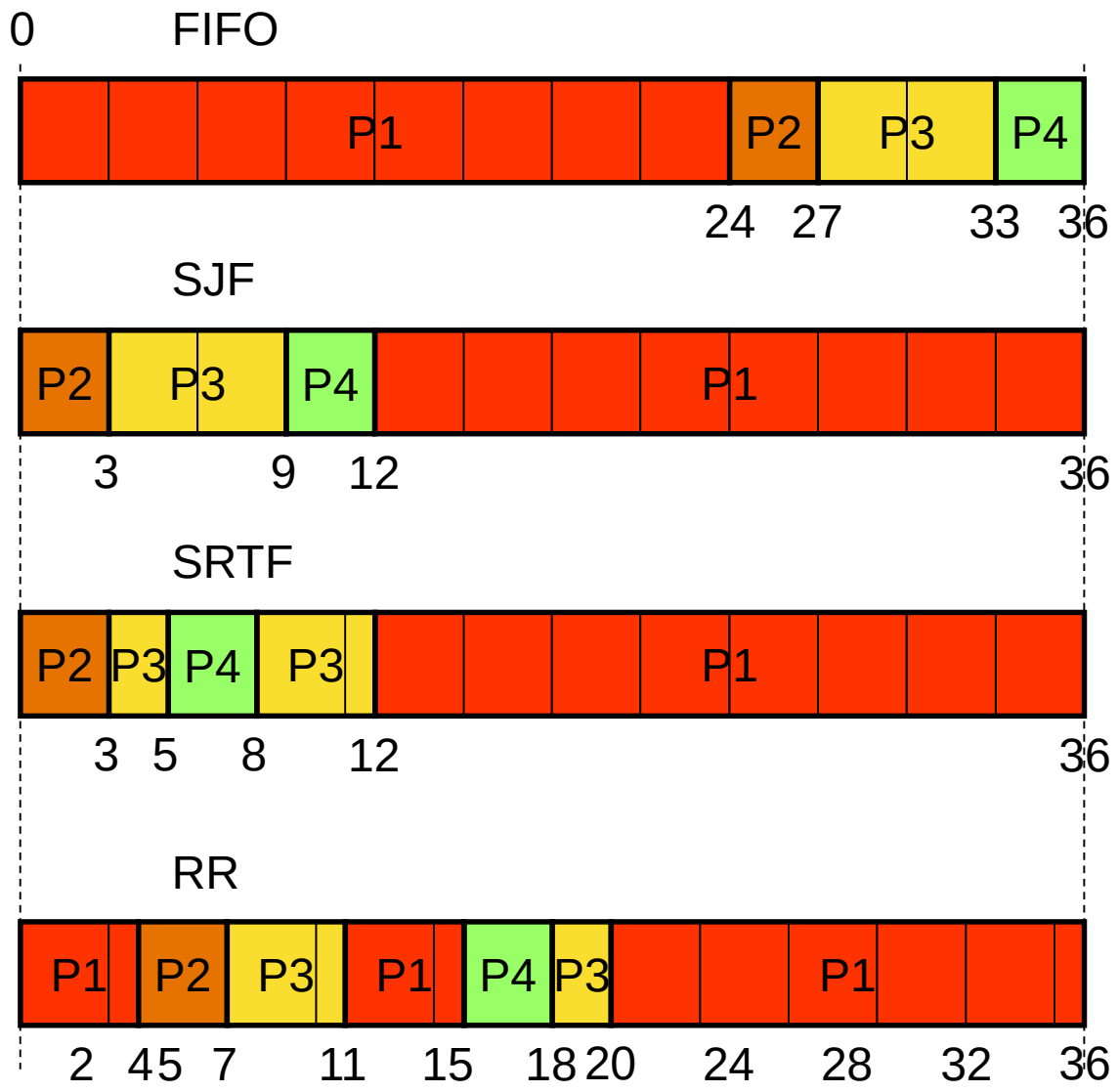
Task	Arrival time (ms)	CPU burst (ms)
P1	0	24
P2	0	3
P3	2	6
P4	5	3

Lesson 1., RR Gannt diagram (chart)

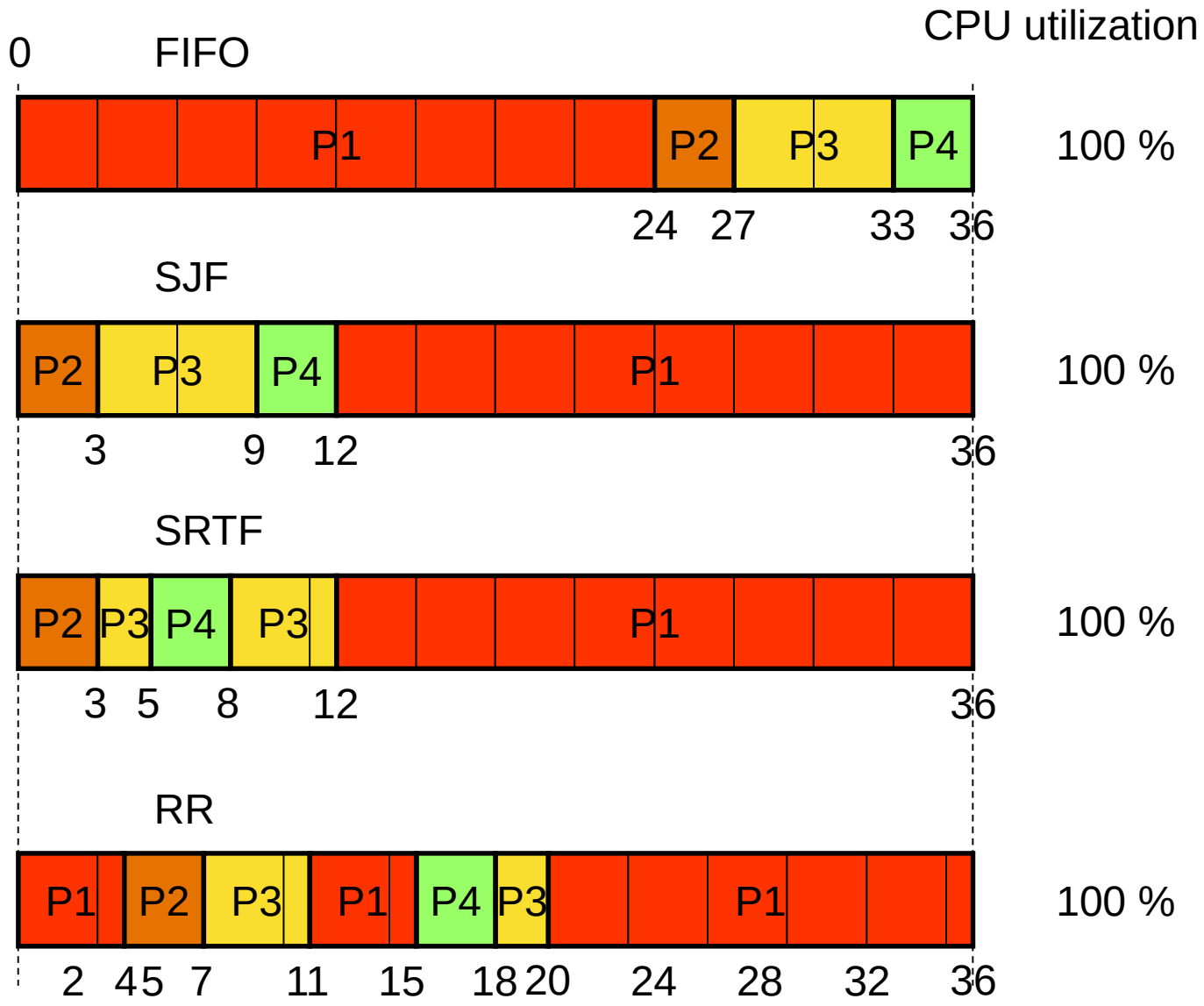


Task	Arrival time (ms)	CPU burst (ms)
P1	0	24
P2	0	3
P3	2	6
P4	5	3

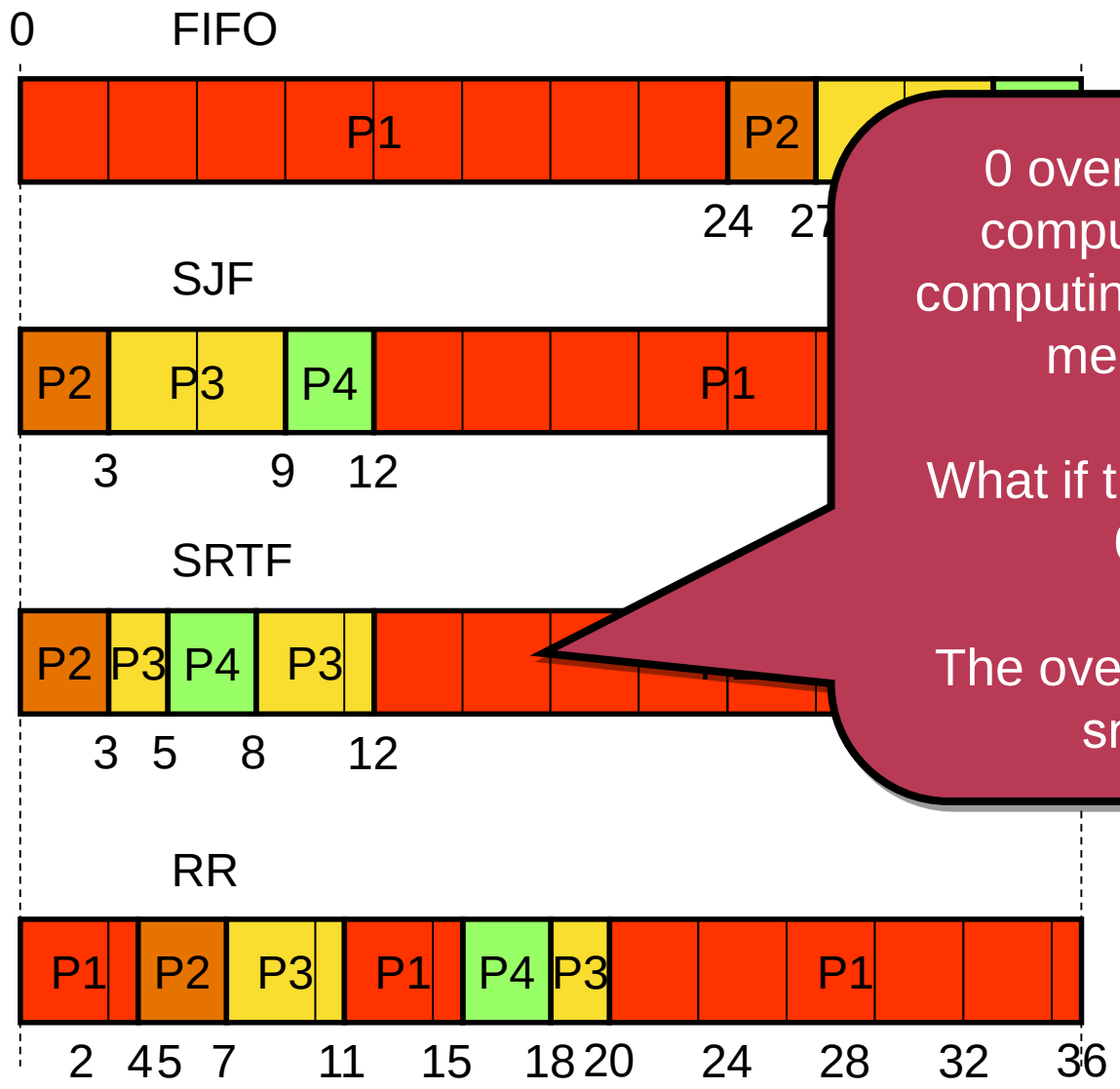
Lesson 1., Gannt diagram (chart)



Lesson 1., CPU Utilization



Lesson 1., CPU Utilization



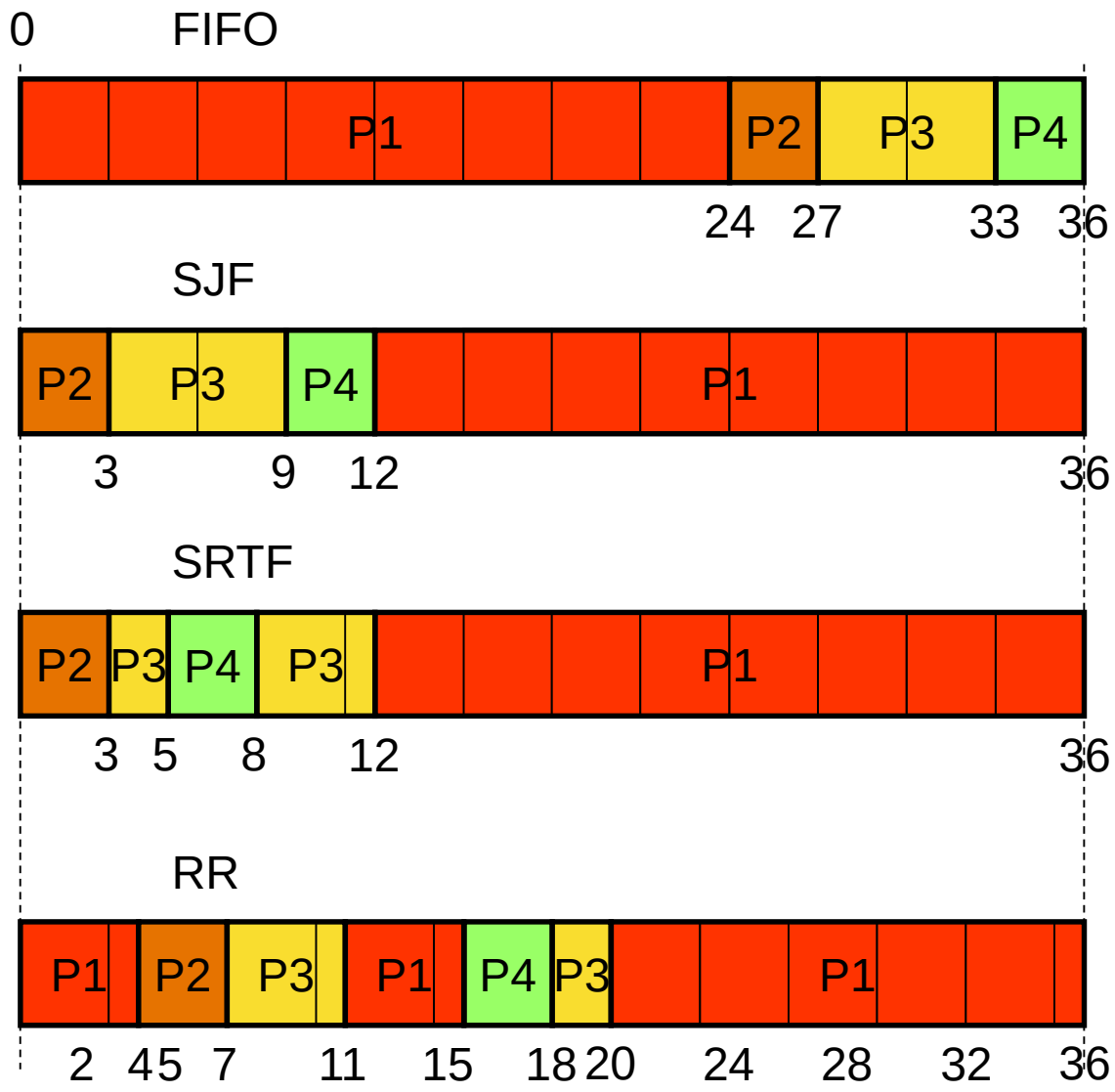
0 overhead based computation, even computing throughput is meaningless

What if the overhead is 0.1 ms

The overhead is much smaller…

Lesson 1., CPU utilization

CPU utilization with 0.1 ms scheduling and context switch overhead



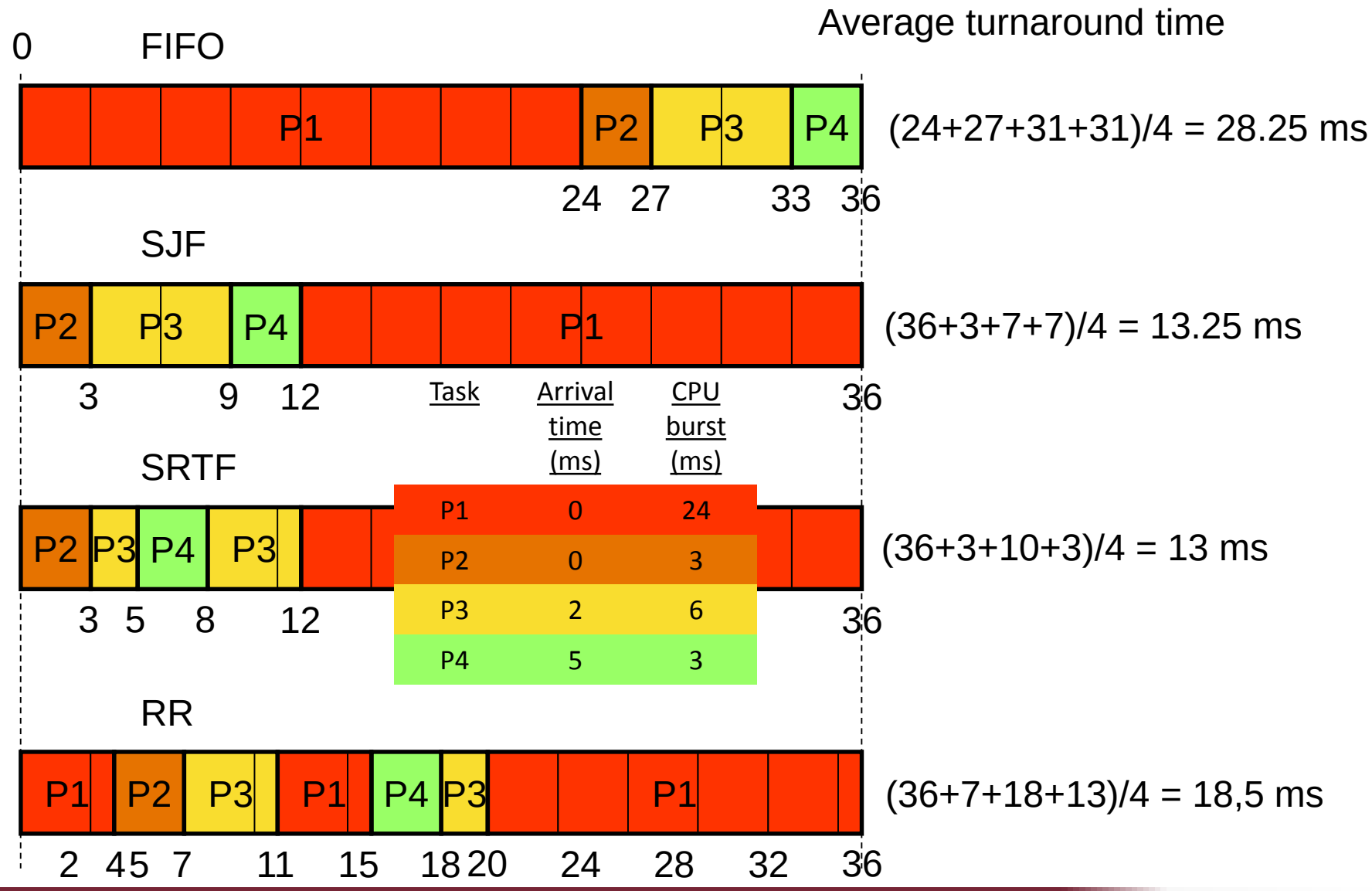
4 cs: 36,4-0.4/36,4 = 98.9 %

4 cs: 36,4-0.4/36,4 = 98.9 %

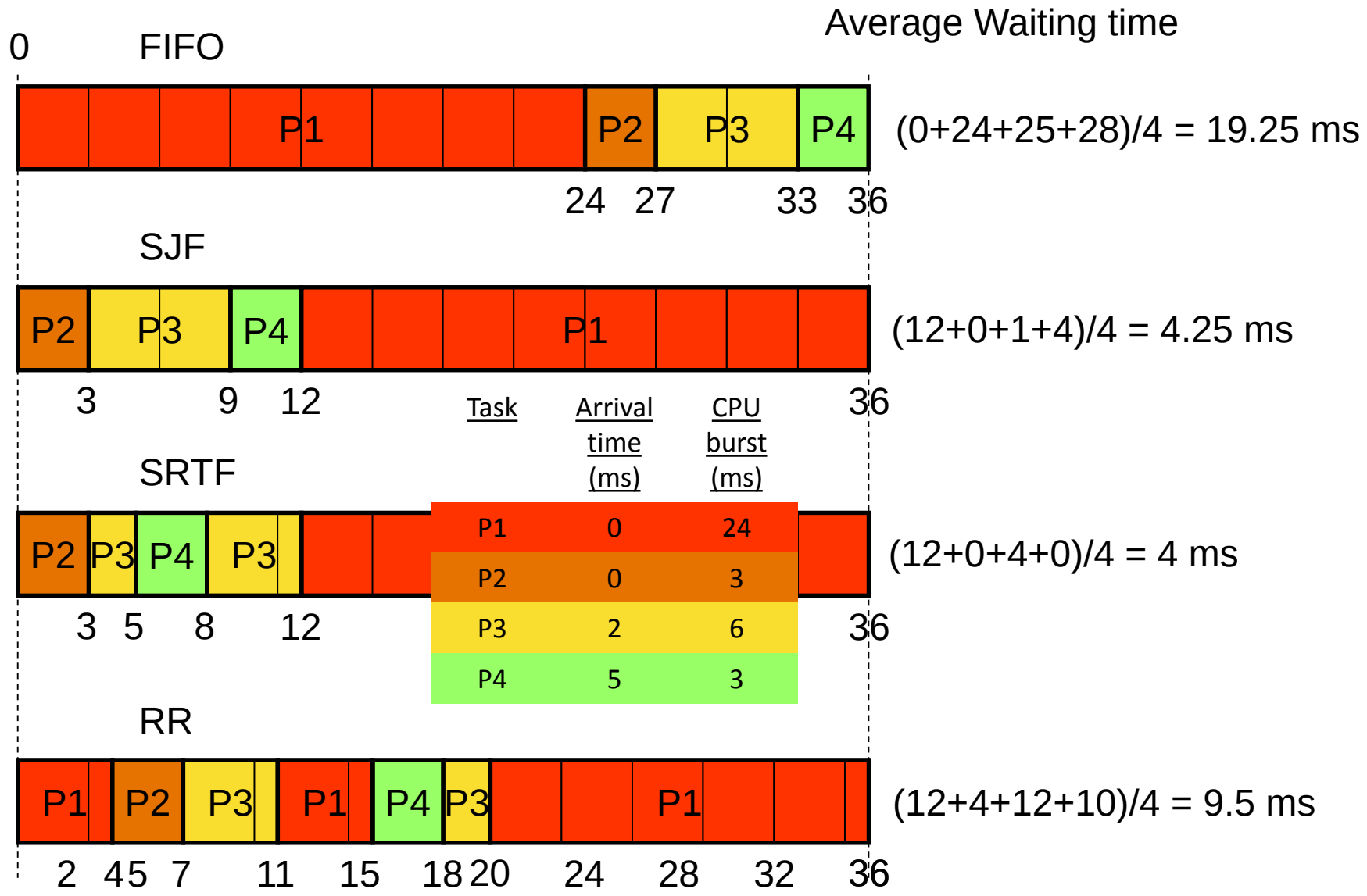
5 cs: 36,5-0.5/36,5 = 98.6 %

7 cs + 3 sc : 37-1/37 = 97.3 %

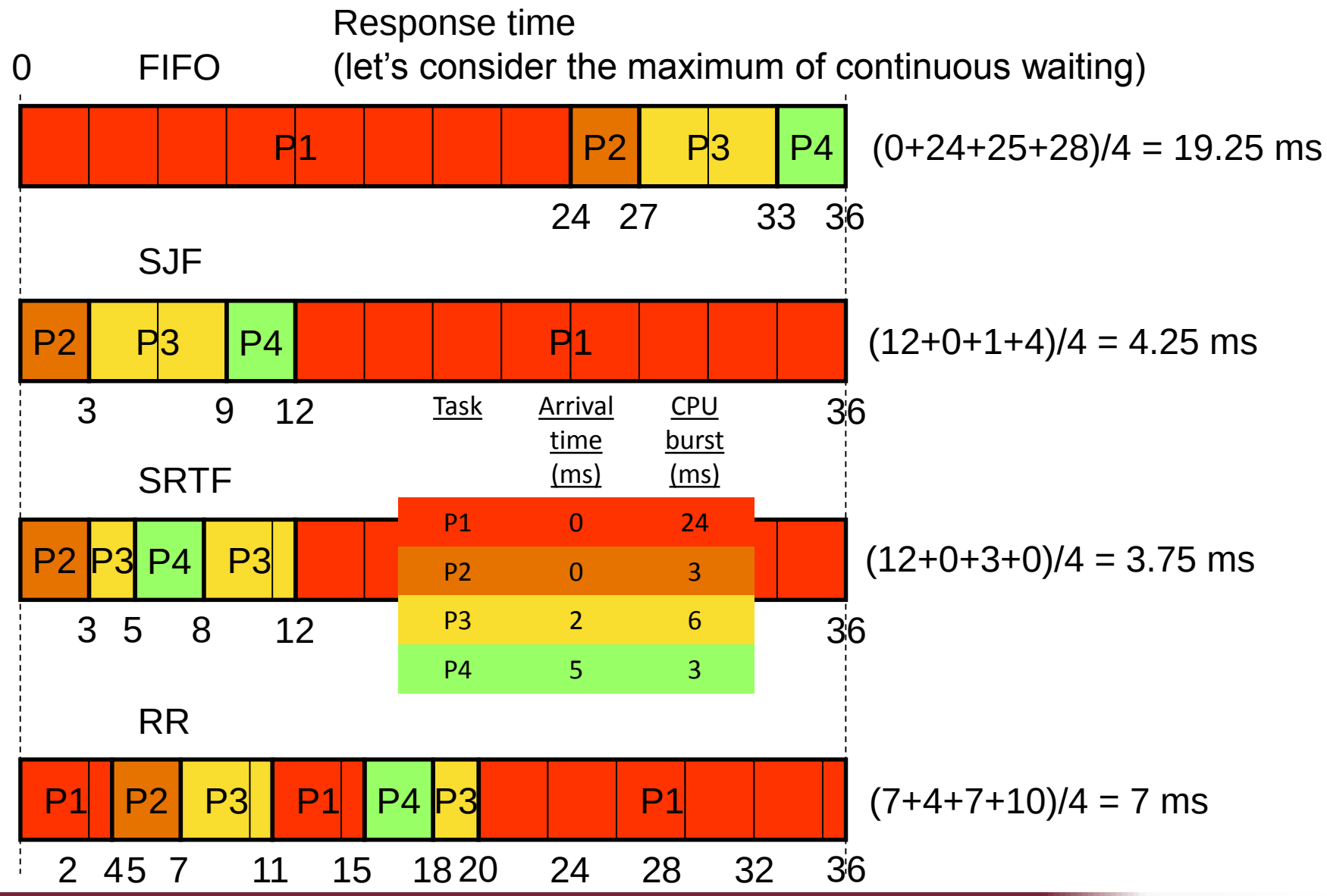
Lesson 1., Turnaround time



Példa 1., Waiting time



Lesson 1., Response time



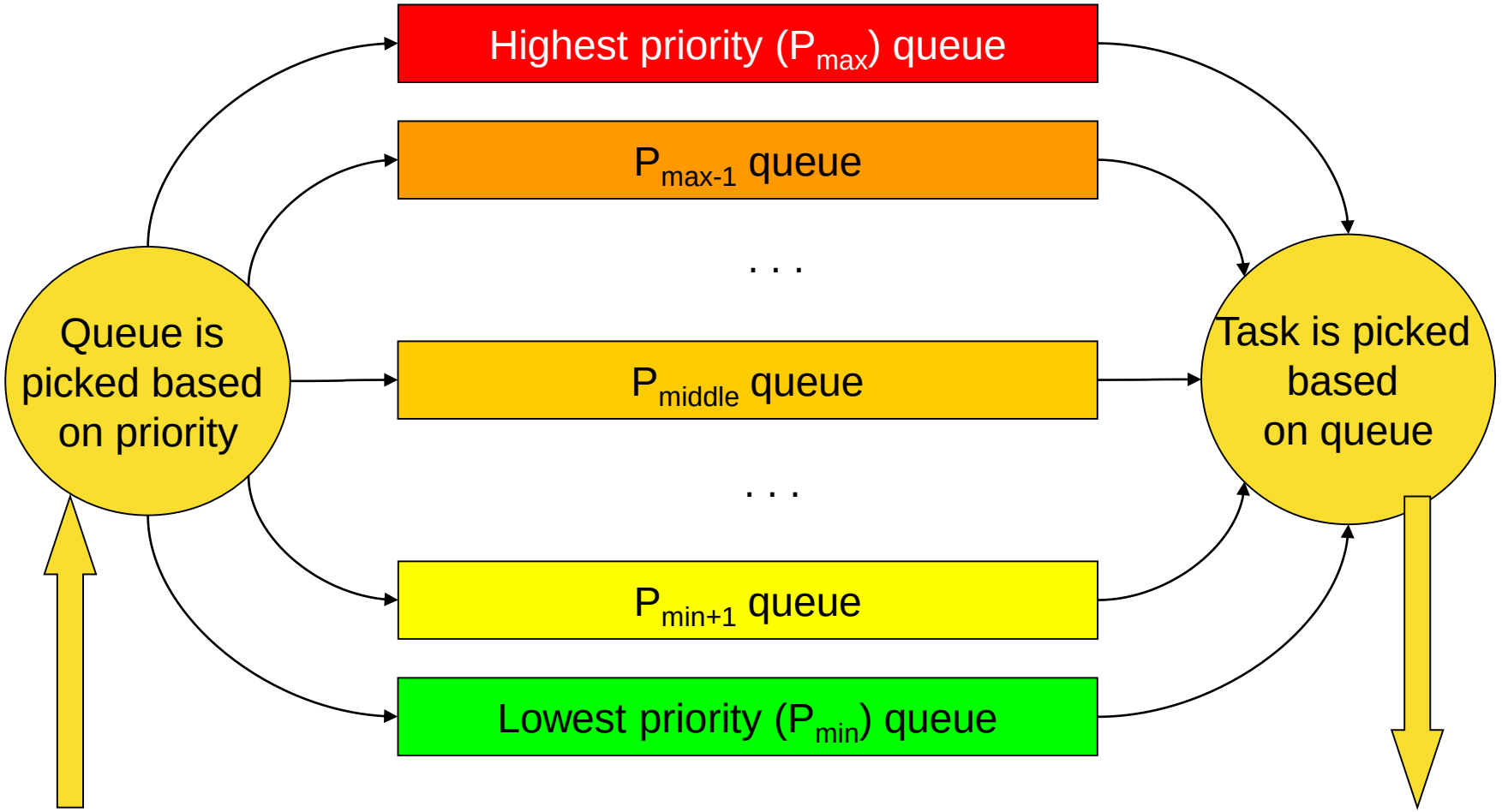
Lesson 1., Conclusions

- The task pool defines the results
  - This task pool has demonstrated the convoy effect and the fine differences in between the SJF and SRTF schedulers
  - SRTF is the best, the SJF follows it closely, but in real life situations we cannot use them because we do not know future CPU bursts
  - The operation of the RR algorithm can be presented also, it parameters are better than the FIFO, and it does not require unavailable run-time information
- Homework (as preparation for the midterm?):
  - 4 identical, 9 ms CPU burst task arriving at 0 ms
  - 4 identical, 9 ms CPU burst tasks arriving at 0, 3, 6, 11 ms

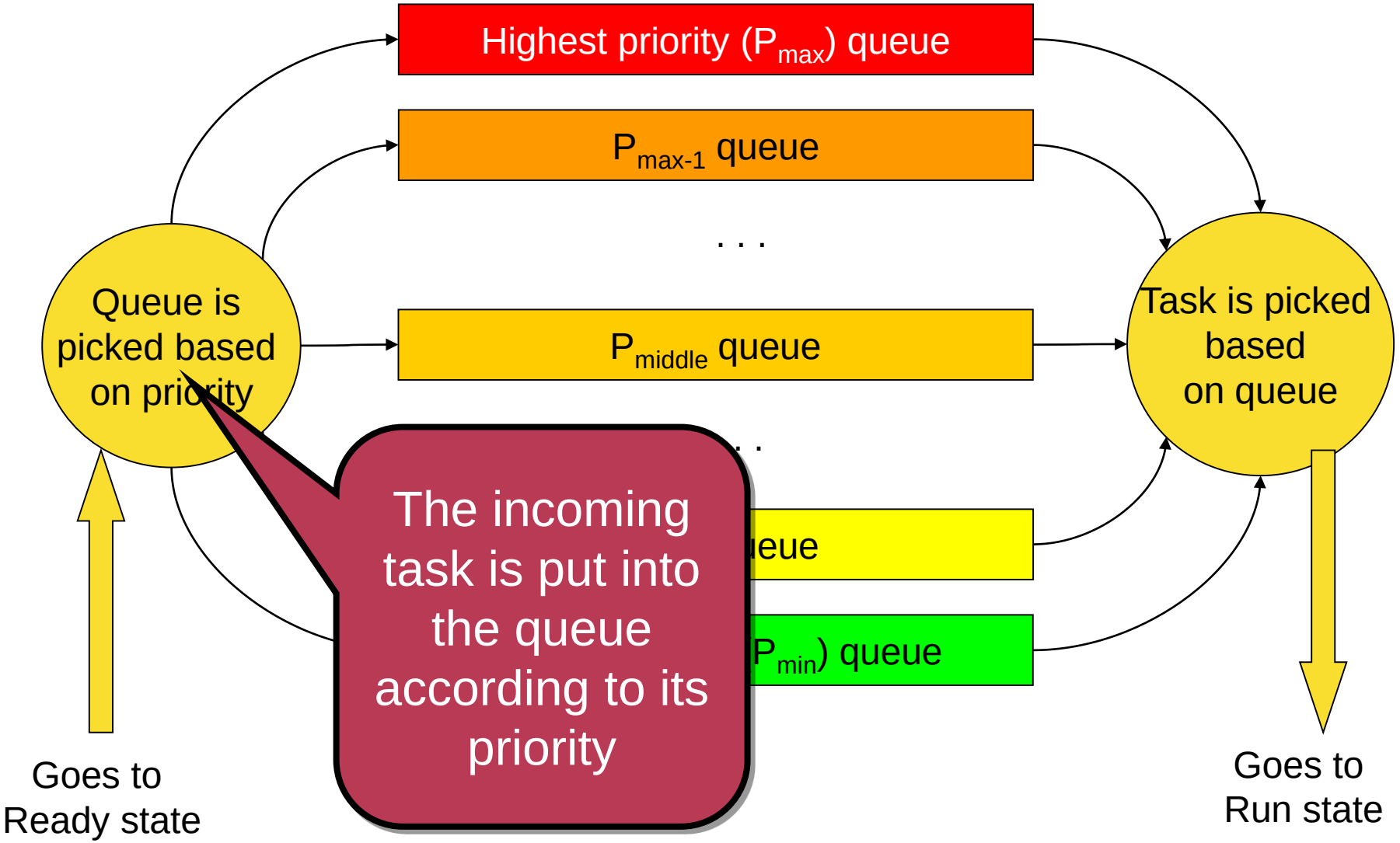
Multilevel queue

- For all priority levels we have a ready queue
  - It does not say anything determining the priorities...
  - It does not say anything about the scheduling algorithms used on a priority level (typically FIFO, RR)
    - It depends on implementation
  - How many priority levels are needed?
    - Definitely not too much (8-16-32).
    - In case of too many priority levels it transforms itself to the simple priority based scheduler (1 task on a level)
    - Because it is required to have a queue for all priority levels by increasing the number of priorities the complexity of implementation grows

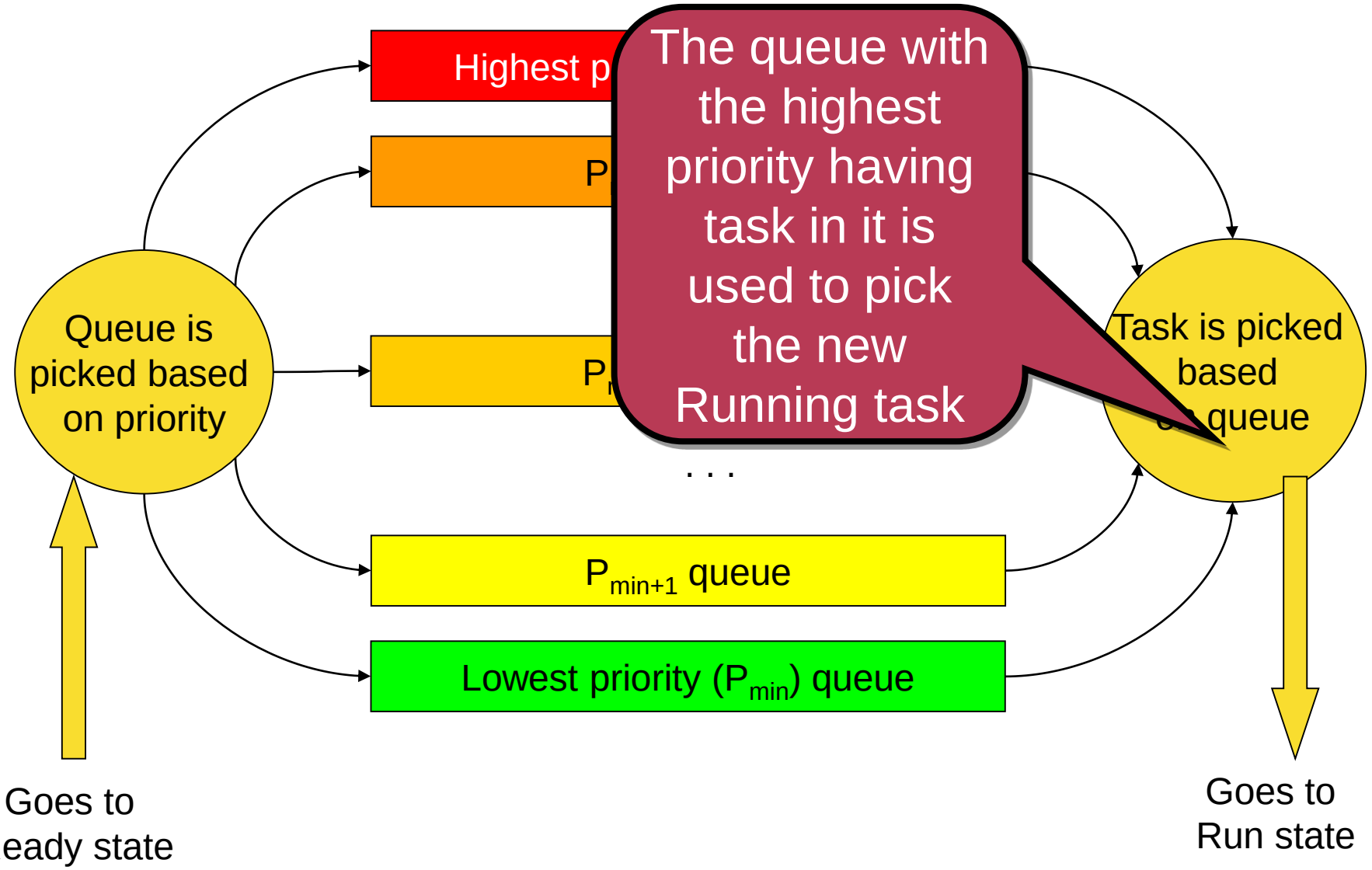
Multilevel queue 1.



Multilevel queue 1.



Multilevel queue 1.

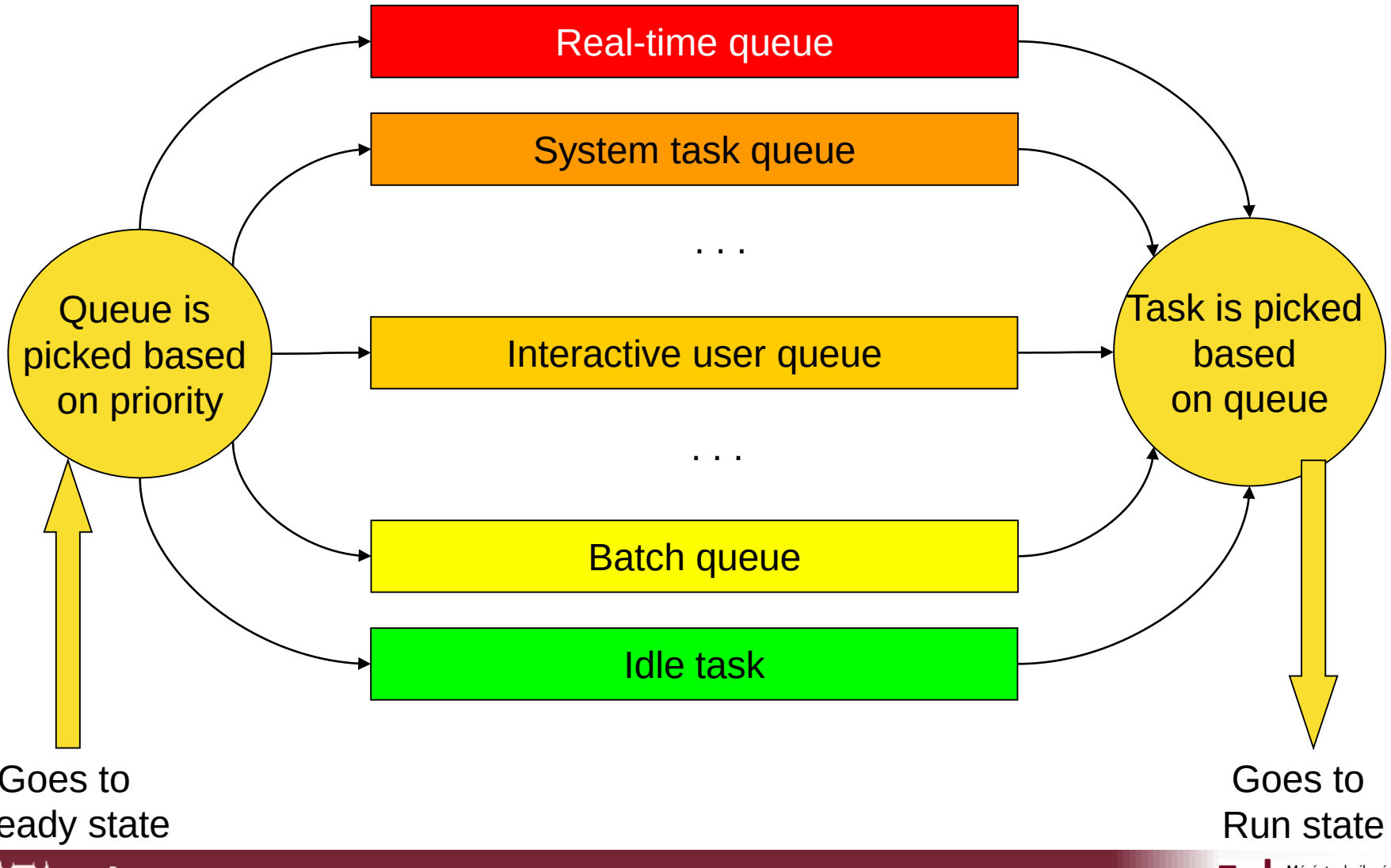


Assignment of priorities

- Most cases priority is determined based on the task, e.g.:
  - Batch tasks have low priority
  - On-line (interactive) tasks have medium priority
  - Some system tasks have high priority
  - Real-time tasks have highest priority
- Typically RR scheduling is used for the queues
  - Maybe FIFO is used for the batch tasks

Multilevel queue 2.

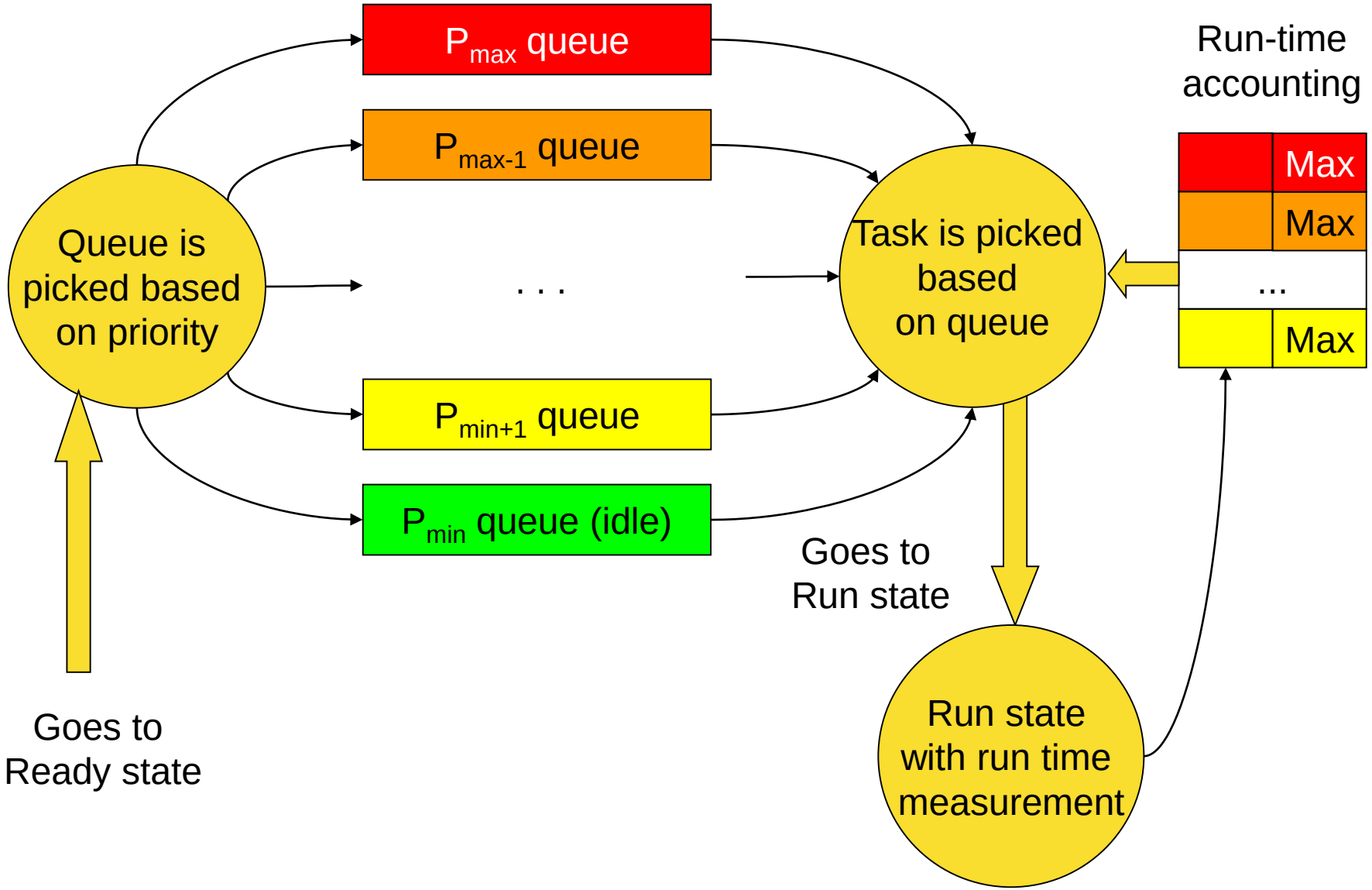
Priority determined based on tasks



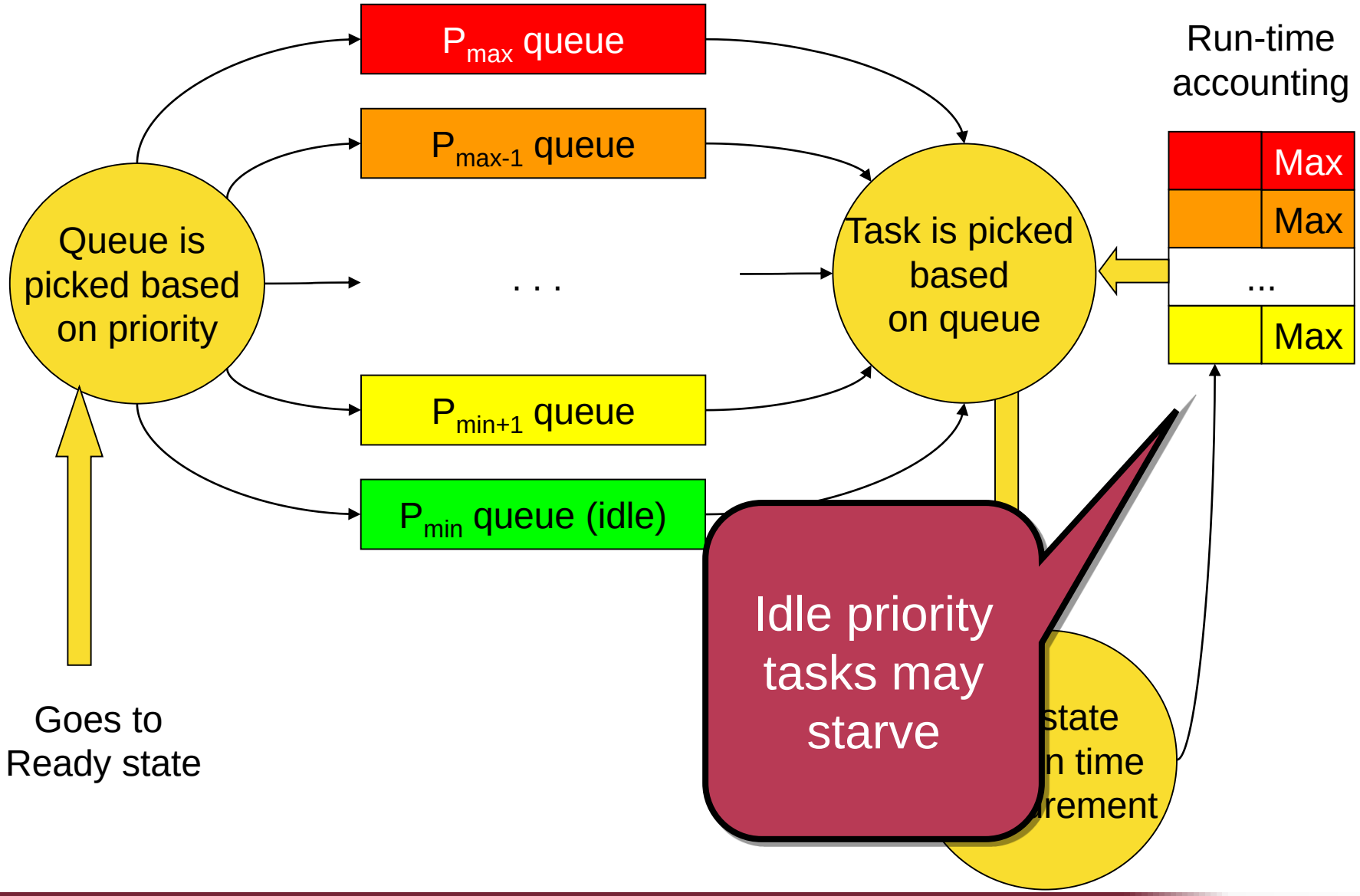
Problems of priority based systems

- Low priority tasks may starve in priority based systems
  - Too many task on higher priority levels running for long period of time
  - It is an overload situation, the system has insufficient resources to run all the tasks
- Time sharing among priority levels (fairness, no starvation)
  - A given % of time is assigned to priorities
    - If there are tasks Ready on that level they get the CPU time available
    - They get more only if there is no lower priority level task Ready
  - Requires more complex administration
    - Time must be measured and distributed for priority level?
  - Weighted fair queuing, Weighted Round-robin scheduling

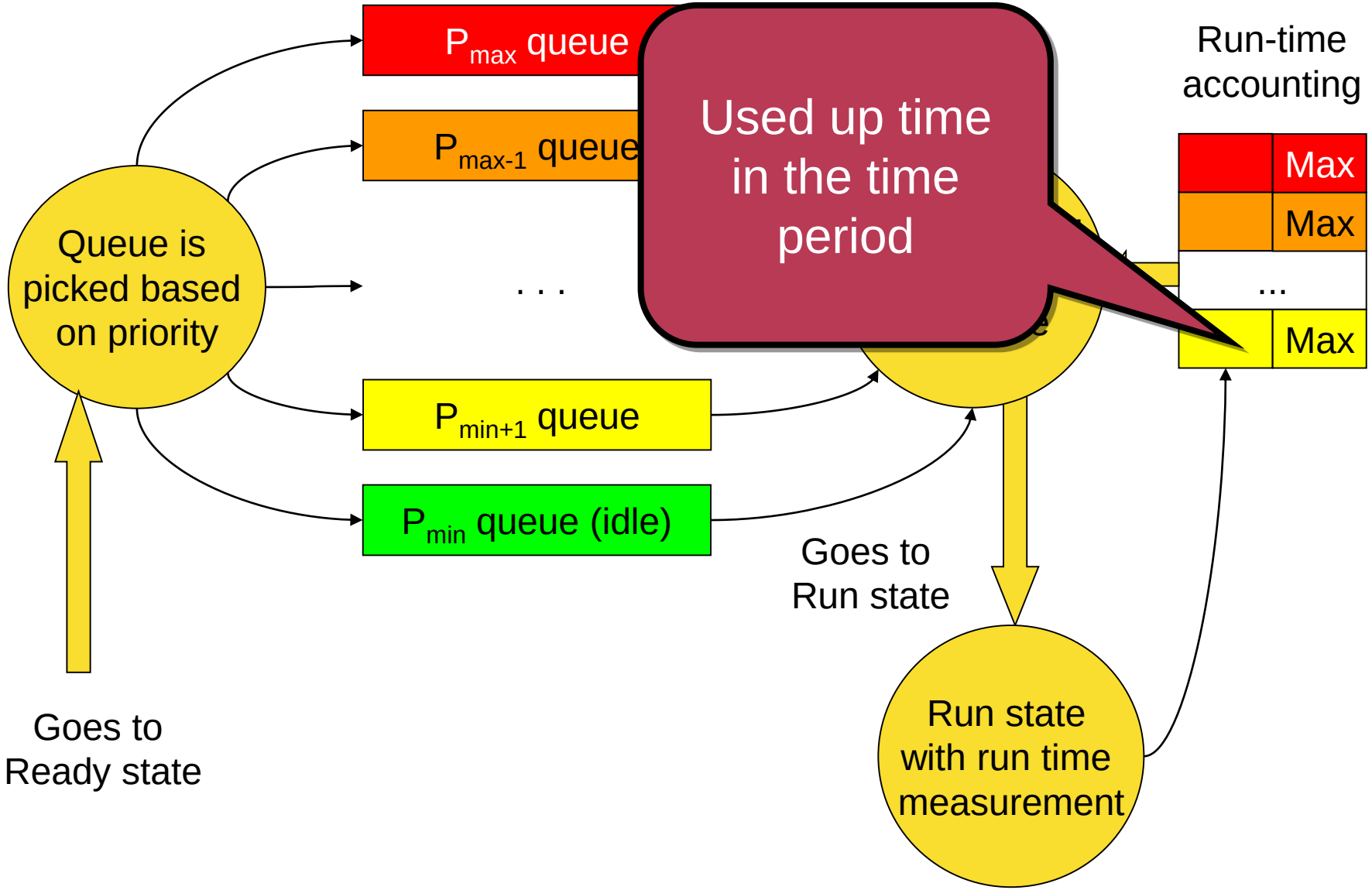
Time sharing among priority levels



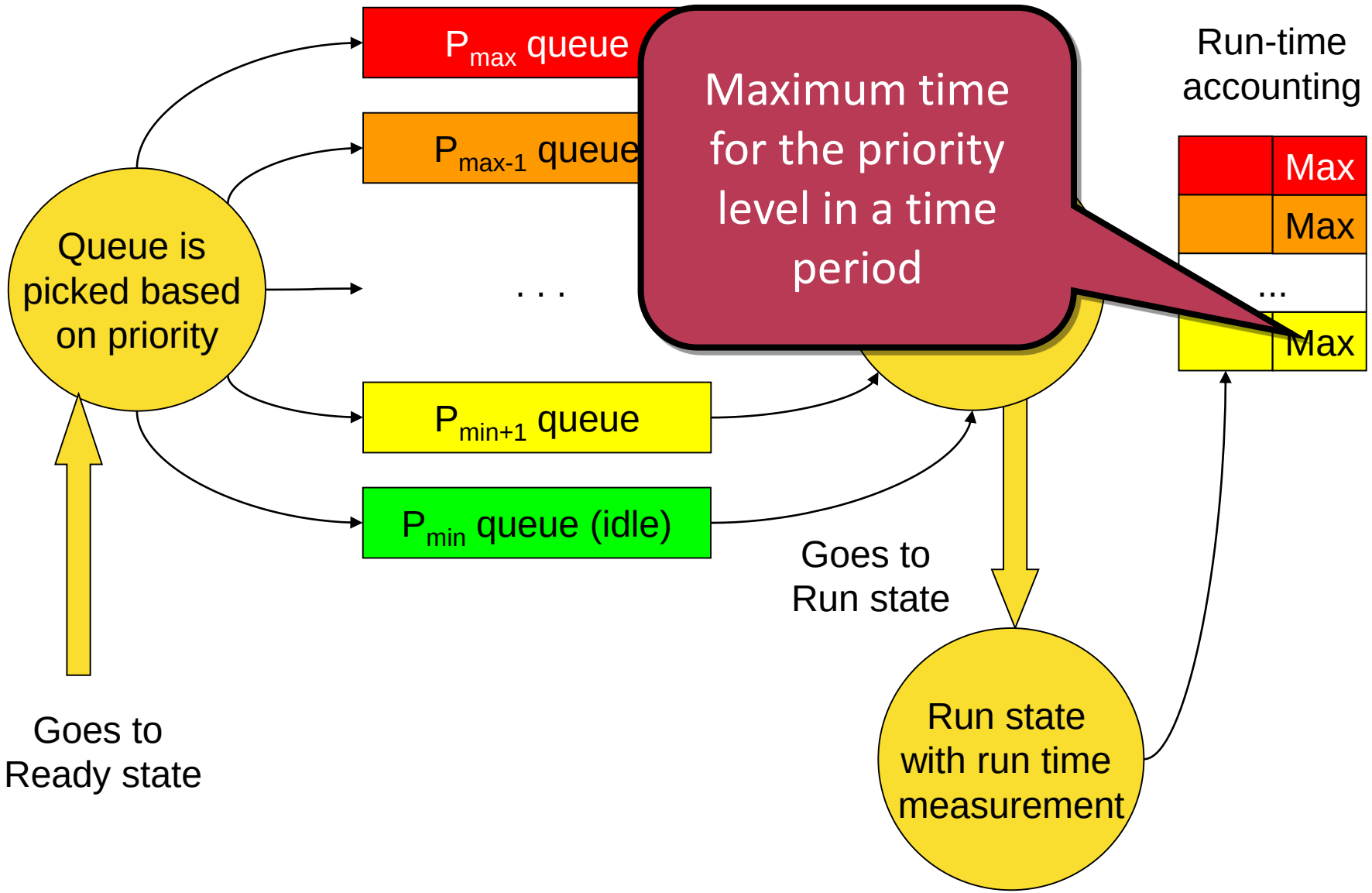
Time sharing among priority levels



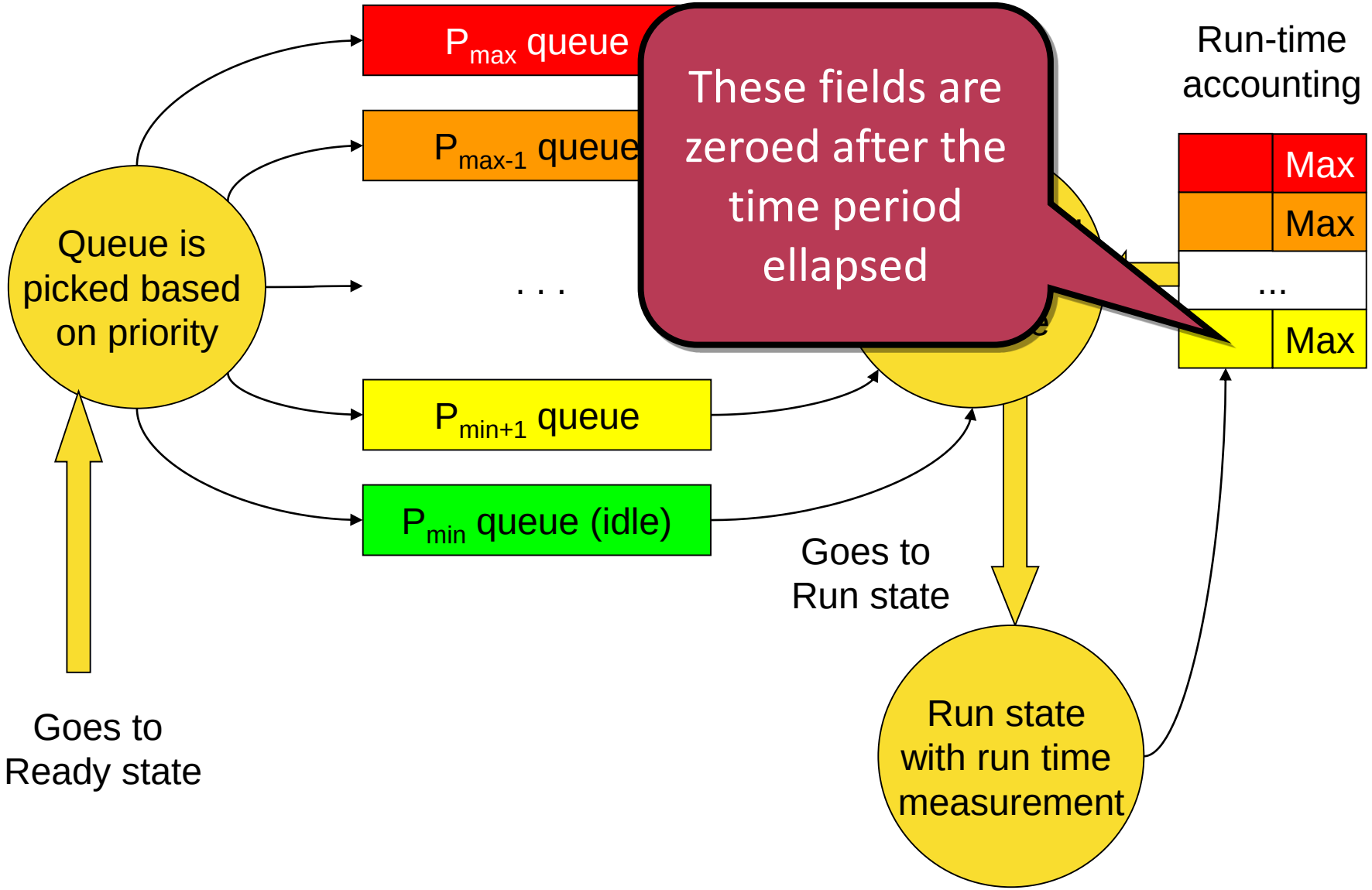
Time sharing among priority levels



Time sharing among priority levels



Time sharing among priority levels



Determining the time period

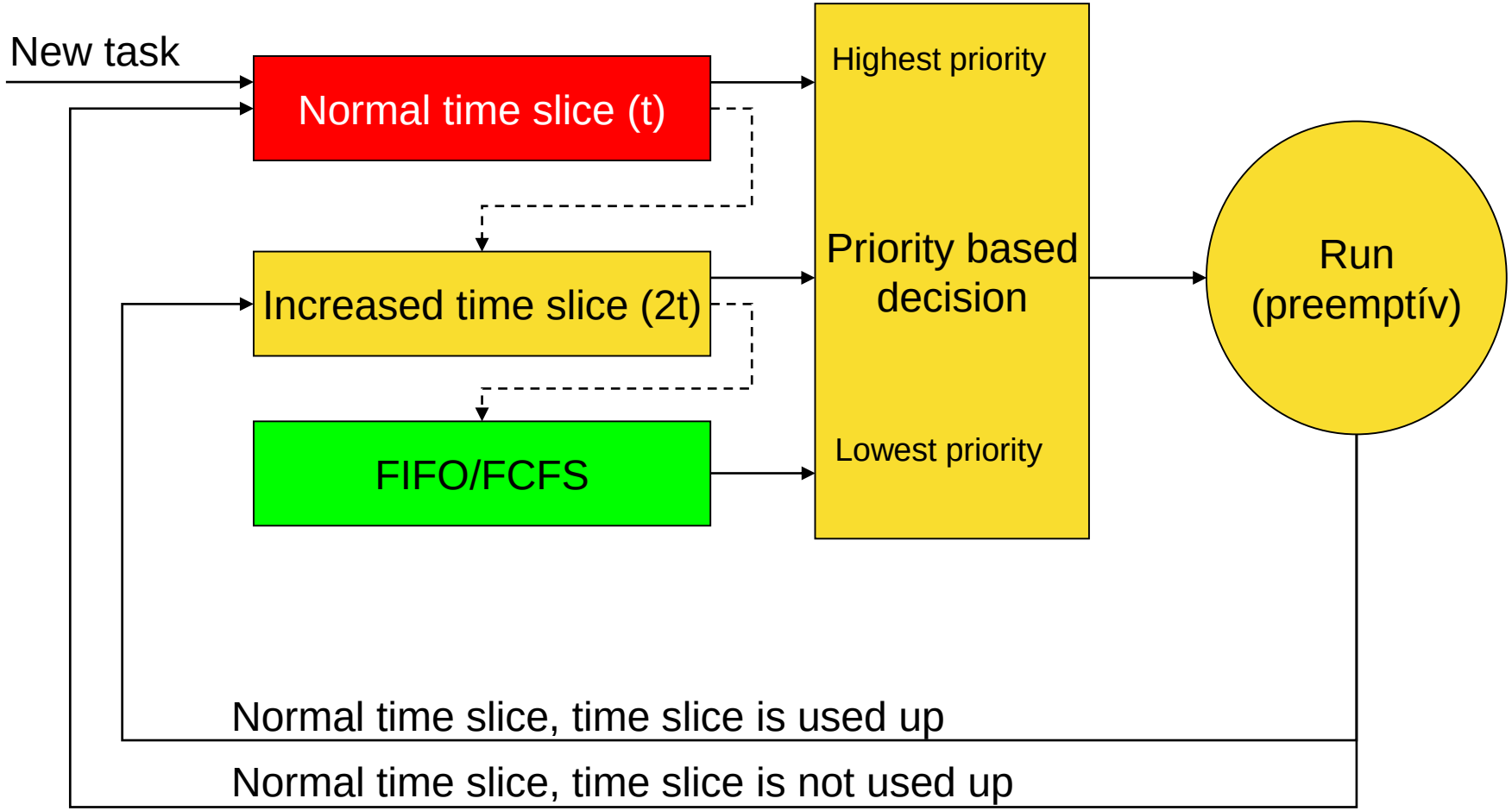
- It is a design decision in which time period we need to make sure that all priority levels get some CPU time
  - How long they can tolerate starving?
  - Reminder: RR time slice is typically10-20 ms
  - Starving may be allowed for much longer periods of time, it can be seconds.

Multilevel Feedback Queues (MFQ)

- Tasks are moved in-between queues based on actually executed CPU bursts
  - Short CPU burst tasks are preferred
    - They stay in the high priority queue with short RR time slice
  - Longer CPU burst tasks will get longer time slice but lower priority
  - Tasks are reevaluated dynamically
    - If the actually executed CPU burst decreases it may go back to a higher priority shorter time slice queue
  - Tasks waiting long times may get increased priority (ageing)
- Can be combined with other scheduling algorithms
  - E.g. interactive tasks are scheduled with multilevel feedback queue scheduler
  - For other tasks (RT, system, batch, idle, stb.) multilevel queue scheduler

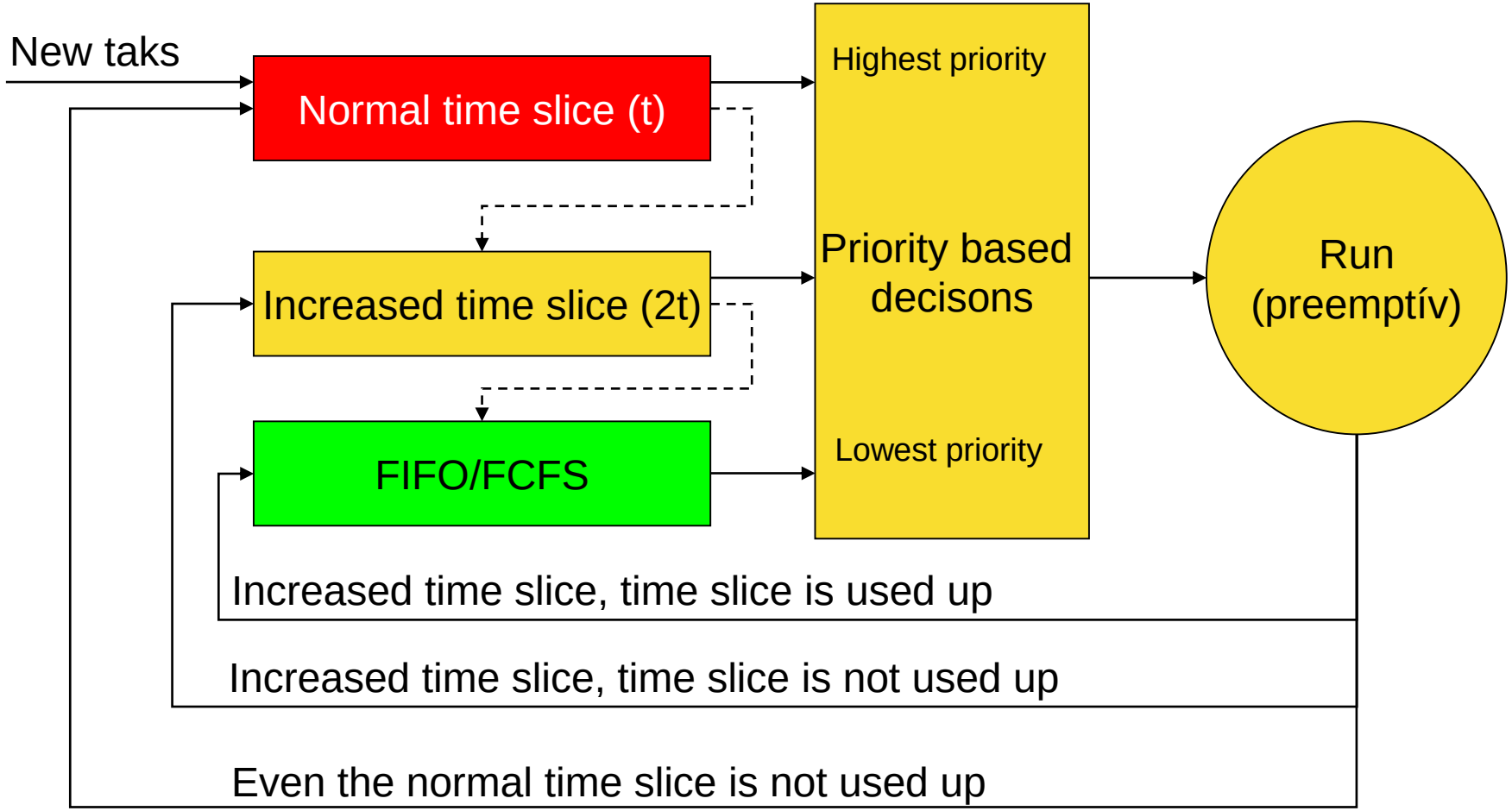
Multilevel Feedback Queues figure 1.

- 3 queues, task coming from the normal time slice queue.



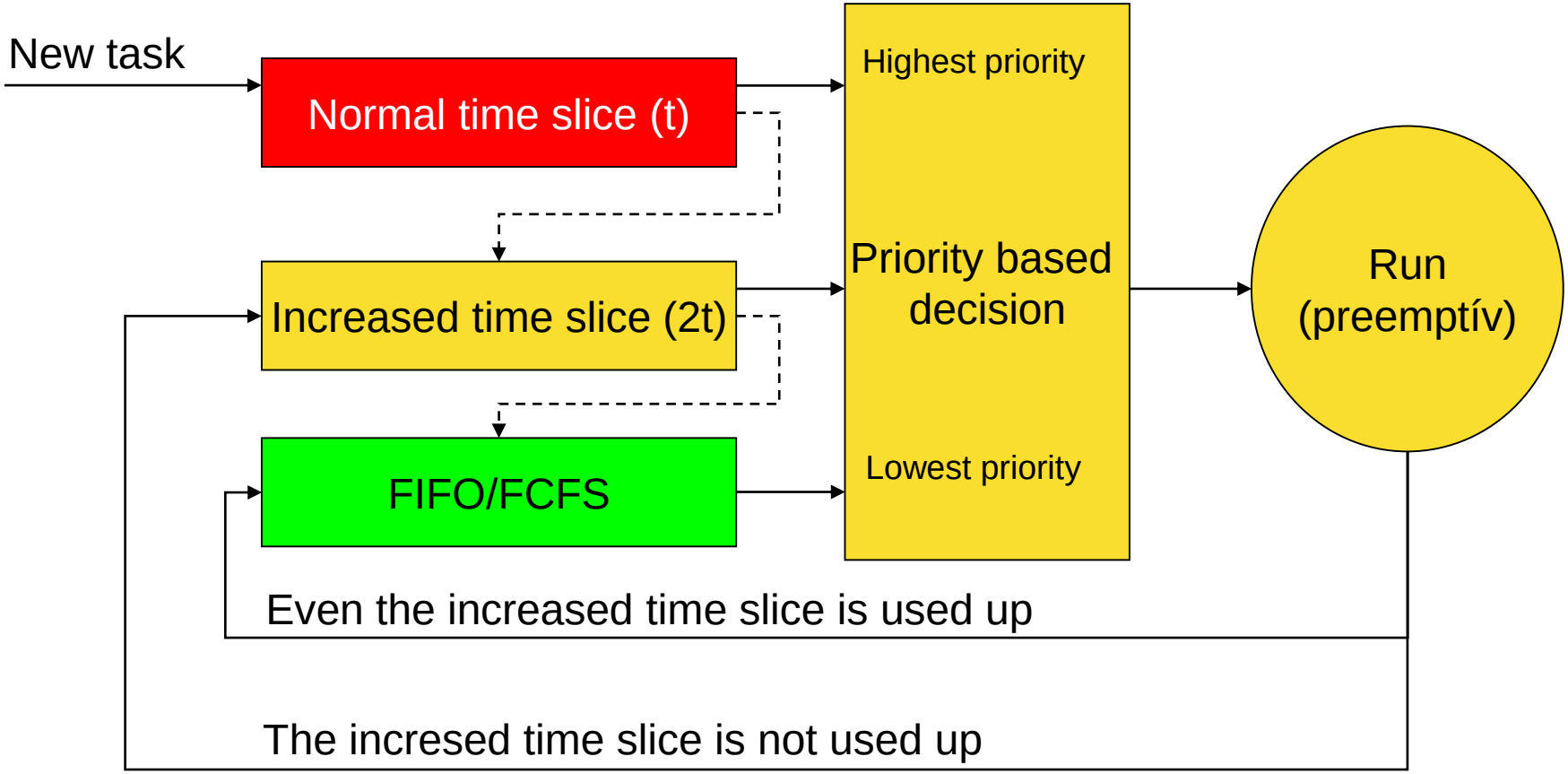
Multilevel Feedback Queues figure 2.

- 3 queues, task coming from the increased time slice queue



Multilevel Feedback Queues figure 3.

- 3 queues, task is coming from the FIFO/FCS queue



Multilevel Feedback Queues

- It is widely used with some extensions in generic operating systems
  - The scheduler of Windows, Linux, etc., show some quite similar internal operation as we detailed here

Multiple-processor scheduling

- Homogeneous multiprocessor system
  - SMP or NUMA architecture (or the mixture of that).
  - I/O peripherals are assigned to a physical processor
  - Solutions used:
    - Master and slaves (one CPU distributes tasks to the others)
    - Self-scheduling / peering (all CPUs have their own scheduler)

Processor Affinity

- The cache holds some parts of the instructions and data of the running task
  - The cache content of processors or processor cores are different (the difference depends on the architecture and the running tasks).
  - Cache coherency is for data in more than one cache, it is not reasonable to store the same data in all caches (they run different tasks with different instructions and data)
  - If the task is put on another execution unit, its performance may be reduced drastically (its data cannot be found the cache of the new execution unit)
    - The new execution unit builds its cache content while running the task…
- Aim: Keep the task on the same execution unit if possible
- Soft or hard processor affinity.
  - Soft: The OS tries to keep task on the same execution unit but no guarantee is given
  - Hard: The task is assigned to an execution unit, it cannot be moved to another execution unit
  - Affinity can be set by a system call

Processor Affinity

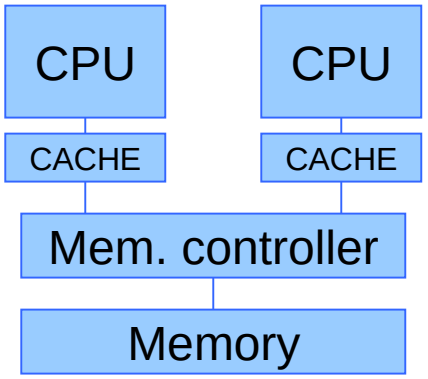
- The cache holds some parts of the instructions and data of the running task
  - The cache content of processors or processor cores are different (the difference depends on the architecture and the running tasks).
  - Cache coherency is fo… reasonable to store … tasks with different …
  - If the task is put on … reduced drastically … execution unit)
    - The new execution …
- Aim: Keep the task on the … ecution unit if possible
- Soft or hard processor a… y:
  - Soft: The OS tries to k… p task on the same execution unit but no guarantee is given
  - Hard: The task is assigned to an execution unit, it cannot be moved to another execution unit
  - Affinity can be set by a system call

Demonstration using Task Manager:
Intel Core i3-2350 (2 core + HT)

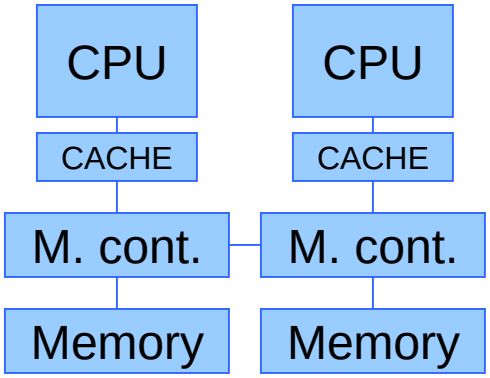
SMP v. NUMA

- In case of SMP only cache matters from the point of view of affinity
  - Sometimes marginal difference is observed
- In case of NUMA the location of the task memory and execution unit matter lot more
  - If possible, task should run from memory local to the execution unit
  - Remote memory use should be minimzed
  - Complexity of scheduling grows…

SMP



NUMA



Load balancing

- Global Ready queue or CPU local Ready queue?
- CPU local Ready queue
  - Push and/or Pull
  - Push: OS kernel task moves the tasks in between CPUs to balance Ready queues
  - Pull: In idle state any processor tries to get tasks from overloaded CPUs
  - The combination of the two can be also used
- Optimization of interdependent , parallel running tasks pl.
  - Gang scheduler
  - In case of large number of tasks and CPUs close to linear scaling can be provided