Scuola Superiore Sant'Anna
di Studi Universitari e di Perfezionamento

API for Real-Time Scheduling with Temporal Isolation on Linux

Tommaso Cucinotta, Dhaval Giani, Dario Faggioli, Fabio Checconi

Real-Time Systems Lab (RETIS)

Center for Excellence in Information, Communication and Perception Engineering (CEIICP)

Scuola Superiore Sant'Anna, Pisa (Italy)

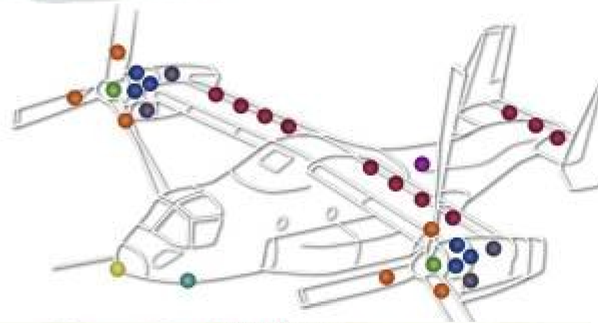
Motivations and background

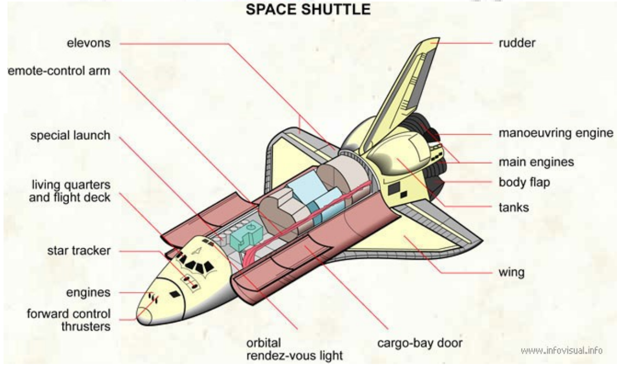
What is Real-Time

Drive assistance

- Engine control, brakes, stability, speed, parking
- Trajectory and set-up control

Defence, army, space



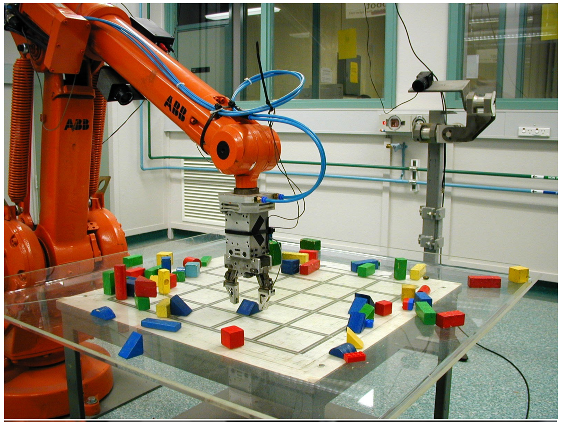


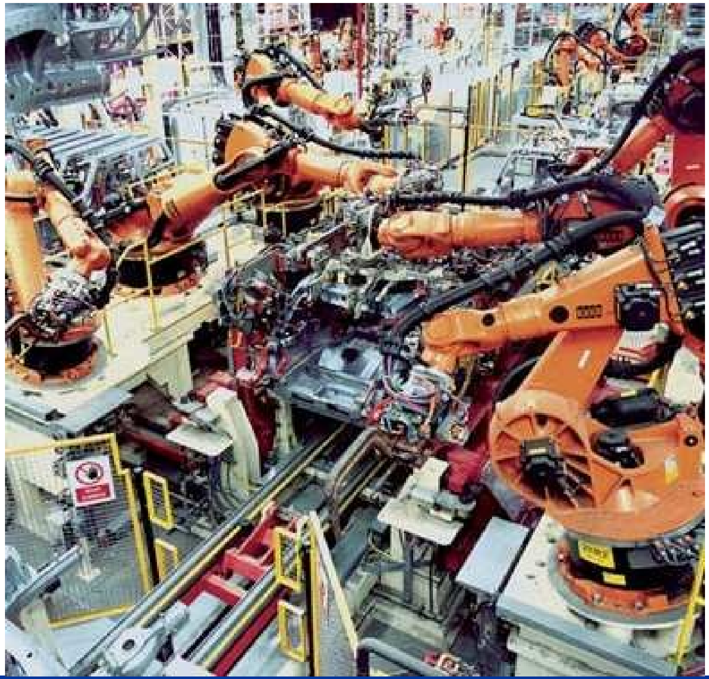
What is Real-Time

Control of chemical and nuclear plants

Control of productive processes and industrial automation

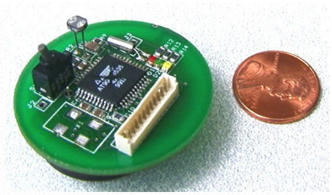
Traffic control

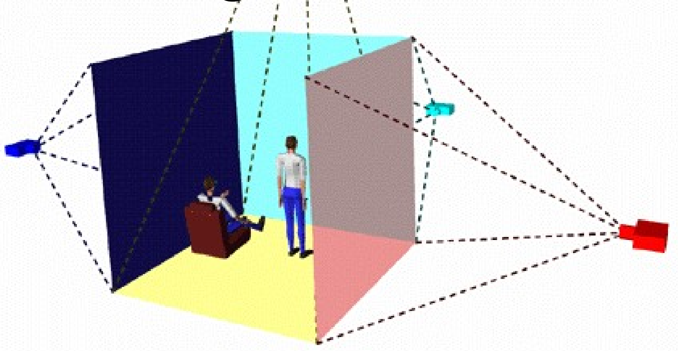




What is Real-Time

- Multimedia, videosurveillance
- Augmented virtual reality
- Telecommunications
- Environment monitoring





Criticality of time requirements

Systems with critical timing requirements

- e.g., defence, army, space

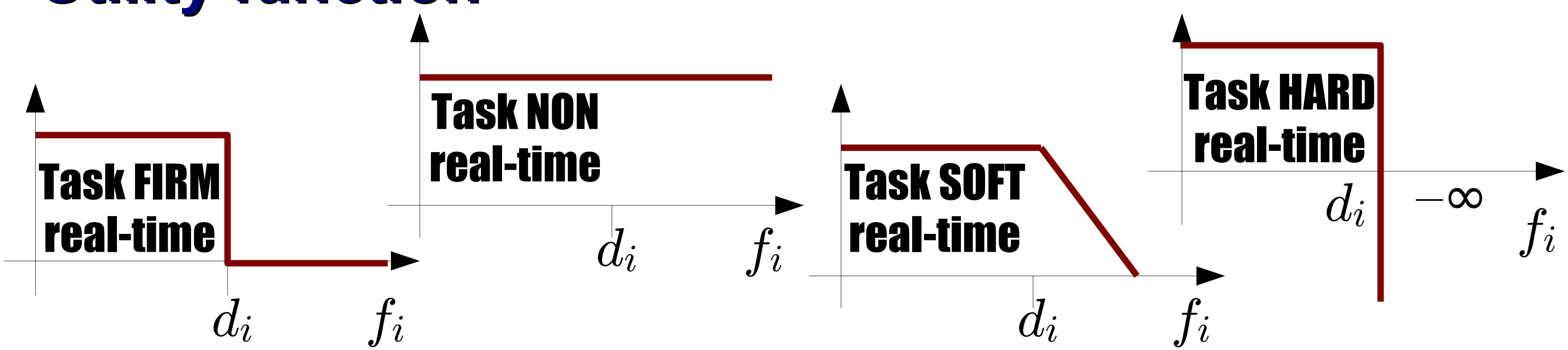
Systems with lower criticality timing requirements

- e.g., industrial automation

Systems with non-critical timing requirements

- e.g., multimedia, virtual reality, telecommunications

Utility function



Our Focus

We focus on systems

- With **non-critical soft real-time** requirements
- Where the use of a **GPOS is desirable and feasible**
  - As opposed to a **traditional RTOS**

Application scenarios

- Multimedia
- Embedded systems
- QoS-enabled Cloud Computing
- Web servers with QoS assurance

Problem Presentation

General-Purpose Operating Systems

- Very effective for storing & managing multimedia contents
- Designed for
  - **average-case** performance
  - serving applications on a **best-effort** basis
- They are <u>not</u> the best candidate for serving *real-time applications* with **tight timing constraints**
  - nor for **real-time multimedia**
  - nor for computing with **precise QoS assurance**

Possible Solutions

Overcoming limitations of a GPOS for multimedia

- **Large buffers** used to compensate *unpredictability*
  - ==> **poor** real-time **interactivity** and no low-latency multimedia
- **One-application one-system** paradigm
  - For example, for low-latency real-time audio processing (jack), gaming, CD/DVD burning, plant control, etc...
- **POSIX real-time extensions**
  - Priority-based, **no temporal isolation**
  - Not appropriate for deploying the multitude of (soft) real-time applications populating the systems of tomorrow
- **Linux Real-Time Throttling** extension
  - Designed for **limiting**, not **guaranteeing**

Proposed Solution

Real-Time Schedulers in Linux

Recently Proposed Real-Time Scheduler(s)

Features (schedulers implement)

- **Temporal isolation** among **tasks** and **task groups**
- Need for provisioning of reservation parameters (**sporadic real-time task model**)
  - **runtime** every **period**
  - Optional allowance to use more CPU if available
- Simple **admission control** scheme
  - May be **disabled** if custom user-space policy needed
  - Optional **over-subscription** possibility with **graceful, controlled** management of **overloads**
- **Priority**-based, **Deadline**-based, **mixed scheduling**
- **Hierarchical scheduling**
  - Attach **more tasks** as a whole to a **single reservation**
  - **Nesting** of groups and subgroups at arbitrary levels

Recently proposed schedulers and their APIs

EDF RT Throttling (a.k.a., The IRMOS Scheduler)

- Parameters: **runtime, period, cpu mask, tasks**
  - **RT priorities** of real-time tasks
- **cgroup**-based interface
  - Problem of **atomic changes** to scheduling parameters

SCHED_SPORADIC

- Parameters: **runtime**, **period**, **priority**, **low-priority**
- POSIX standard system call: sched_setscheduler()
  - **Breaks binary interface** & compatibility
- Alternative system call: **sched_setscheduler_ex()**

SCHED_DEADLINE

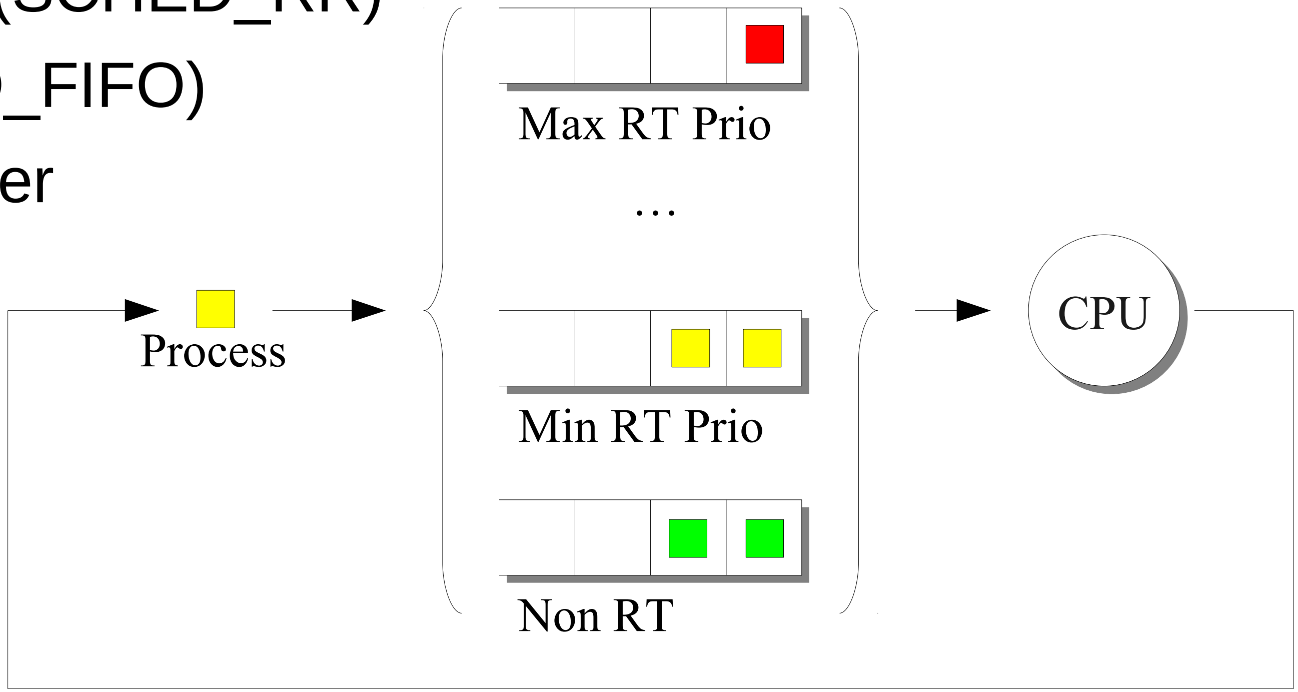
- Parameters: **runtime, period, flags**
- system call: **sched_setscheduler_ex()**

POSIX Real-Time Scheduling

Multi-queue priority-based scheduler

Processes at same priority

- Round-Robin (SCHED_RR)
- FIFO (SCHED_FIFO)
- Sporadic Server
  (see later)



Traditional RT Systems (and Priority Scheduling)

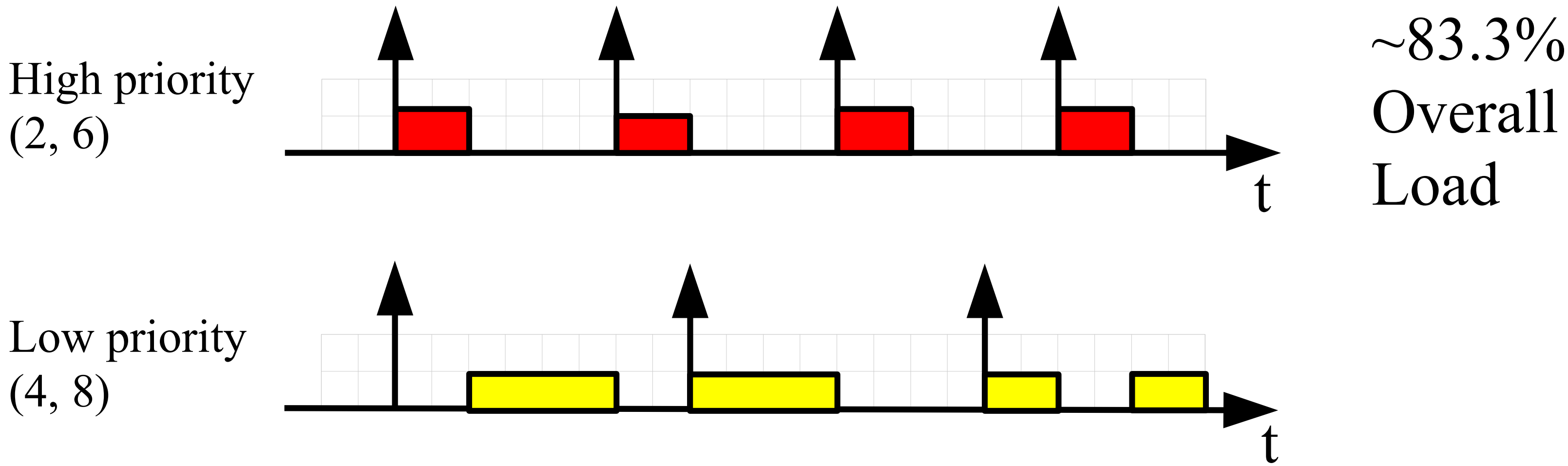
All deadlines respected as far as system behaves as foreseen at design time

- Traditional (C, T) task model
  - C: Worst-Case Execution Time (WCET)
  - T: Minimum inter-arrival period

Admission Control, e.g., for RM:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \left(\sqrt[n]{2} - 1 \right)$$

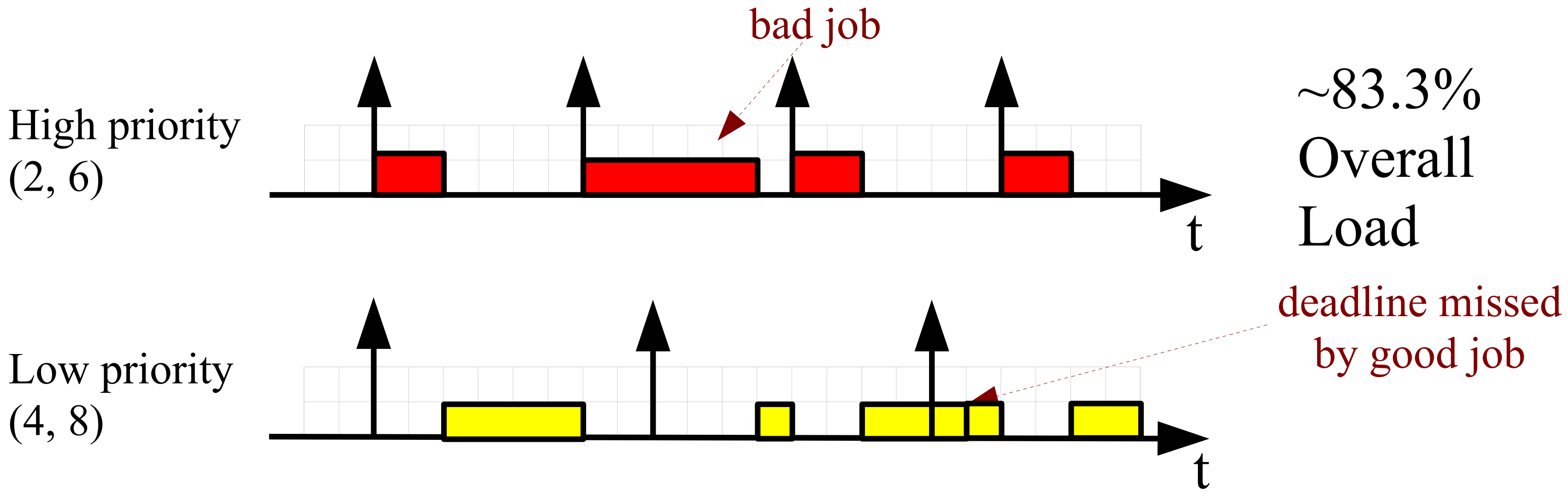
$$\prod_{i=1}^{n} \left(\frac{C_i}{T_i} + 1 \right) \leq 2$$



Problems of Priority Scheduling

High-priority processes may indefinitely delay low-priority ones

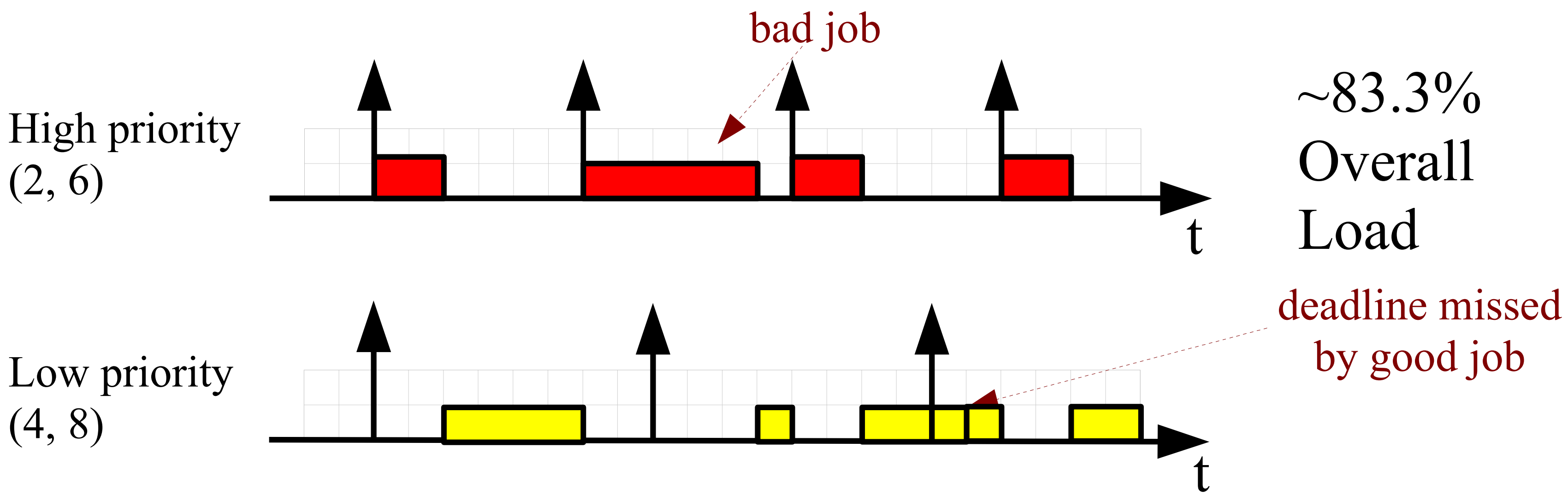
- Coherent with the typical real-time/embedded scenario
  - Higher-priority processes are **more important** (e.g., safety critical)



Problems of Priority Scheduling

High-priority processes may indefinitely delay low-priority ones

- Coherent with the typical real-time/embedded scenario
  - Higher-priority processes are **more important** (e.g., safety critical)
- What if processes have **same importance/criticality** ?



Deadline-based Scheduling

Optimum for single-processor systems

- Necessary and sufficient admission control test for simple task model:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

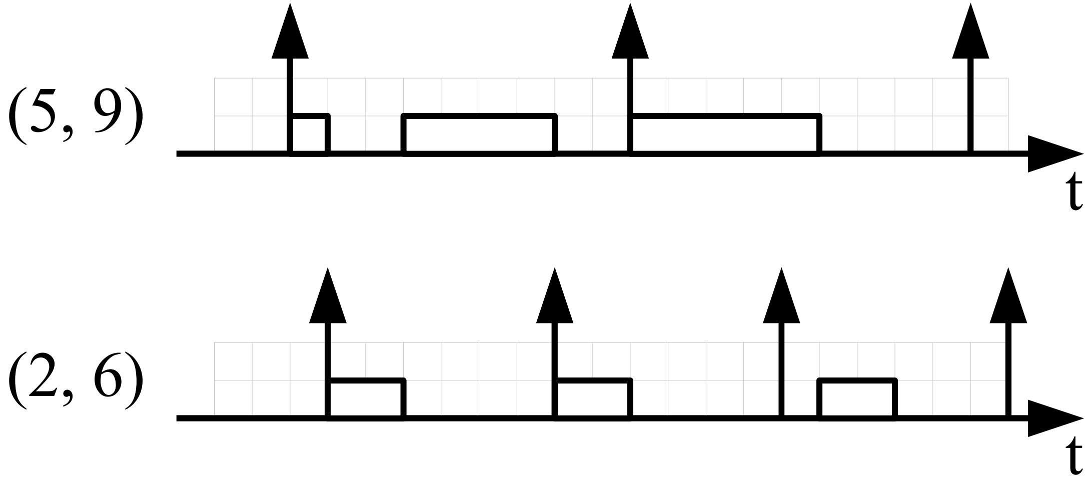
Same problems of PS

- Deadlines respected as far as the WCETs are respected
- Things may go bad when
  - One or more tasks exhibit higher computation times than foreseen
  - One or more tasks behaves differently than foreseen
    - e.g., it blocks on a critical section for more than foreseen
- The task that suffers may not be the misbehaving one

Real-time theory

Reservation-based scheduling: (Q_i, P_i)

- "Q_i time units **guaranteed** on a CPU every P_i time units"



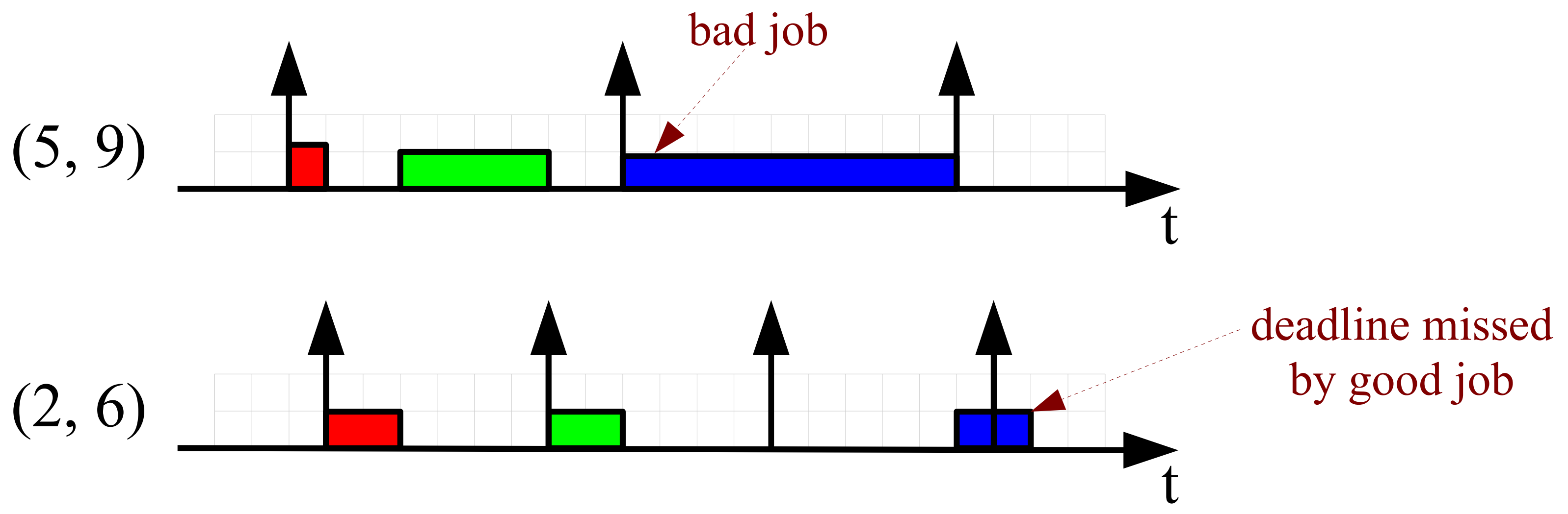
~88.9% Overall Load

- Independently of how others behave (**temporal isolation**)

Temporal Isolation

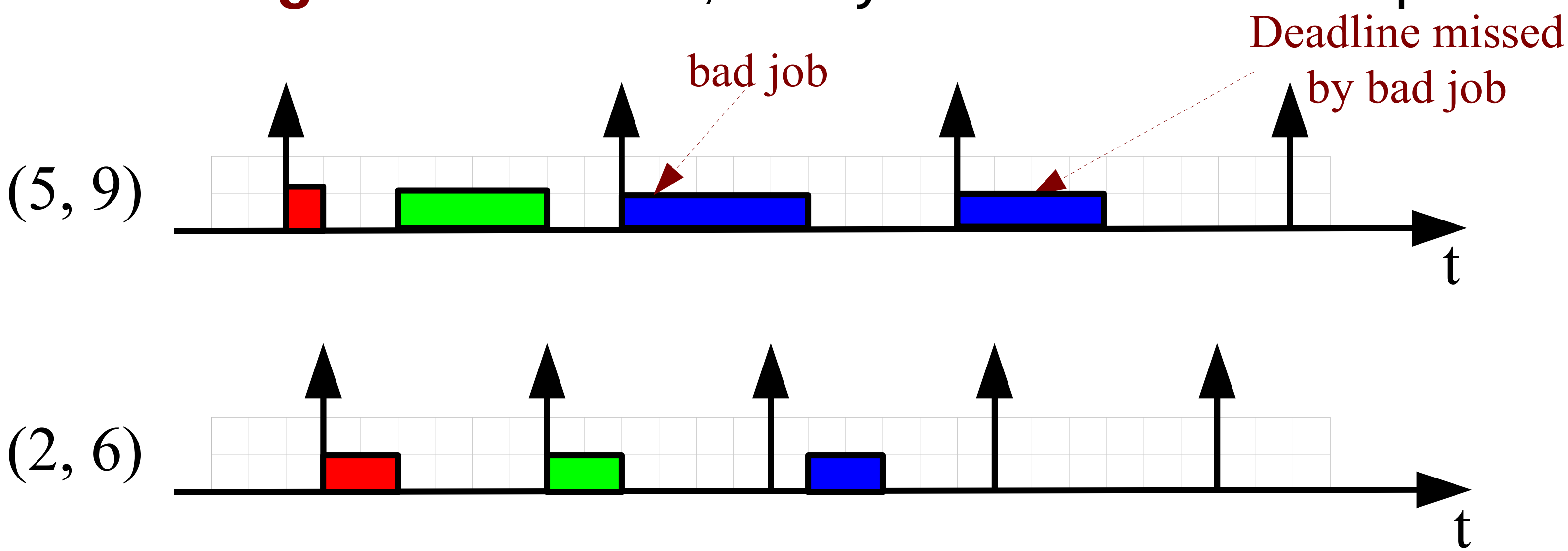
Enforcement of temporal isolation

- Not only EDF scheduling



Enforcement of temporal isolation

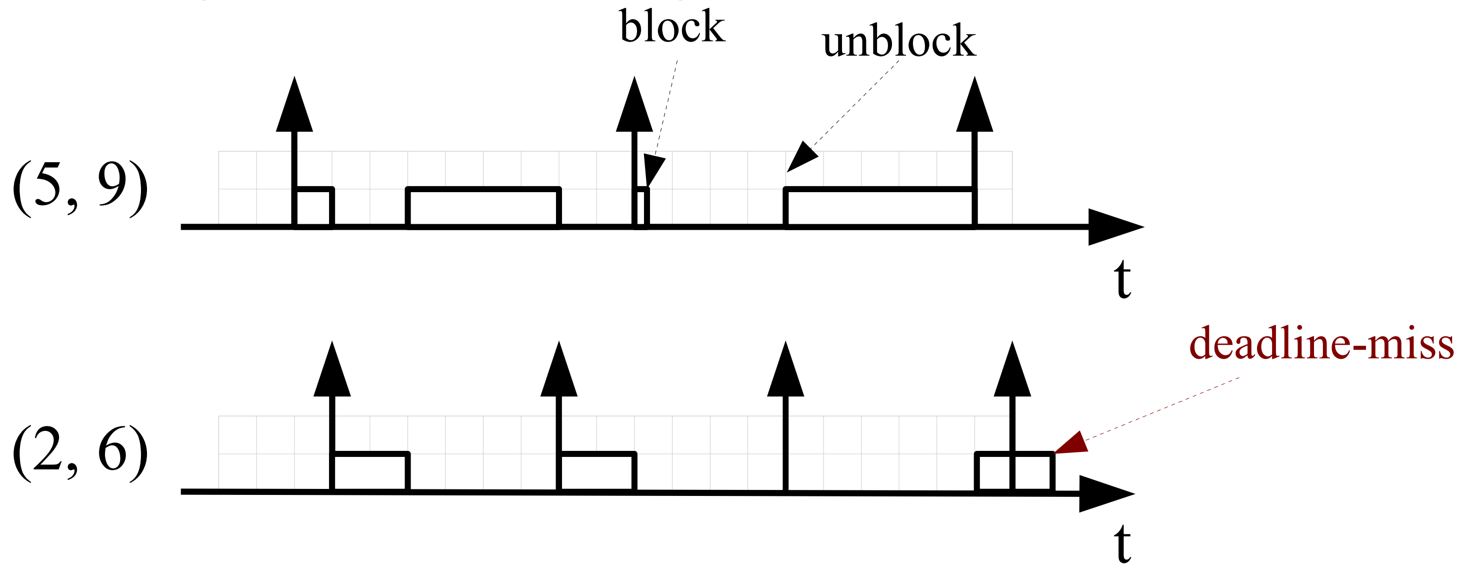
- Once **budget exhausted**, delay to next activation period



Temporal Isolation

Is needed despite blocks/unblocks

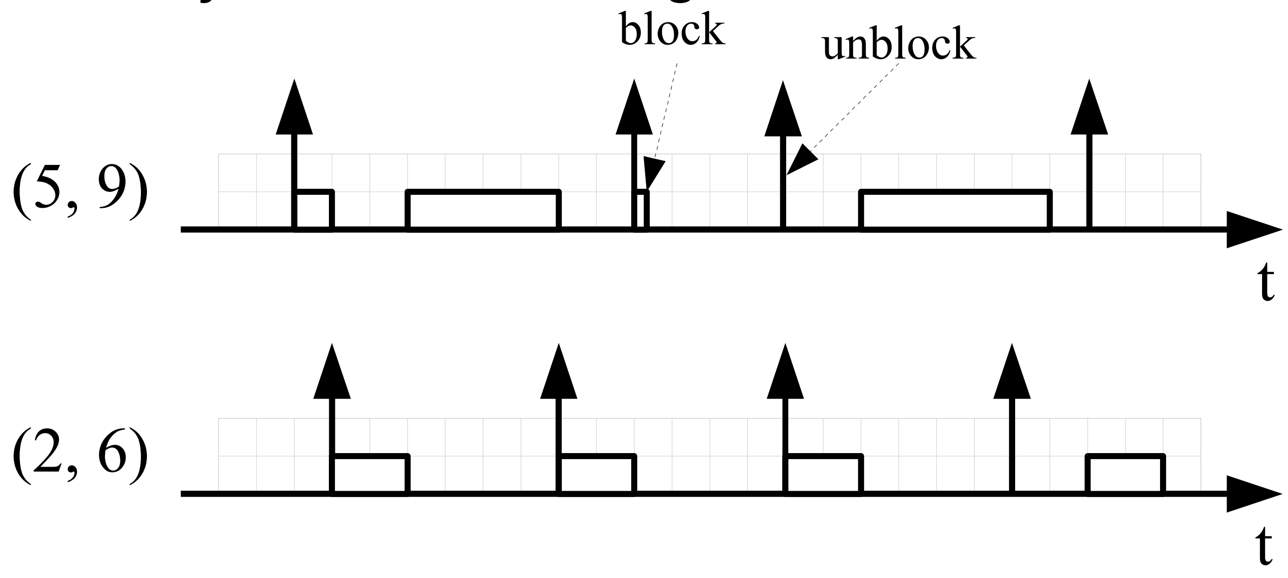
- Not only EDF scheduling



Temporal Isolation

Is needed despite blocks/unblocks

- Not only EDF scheduling

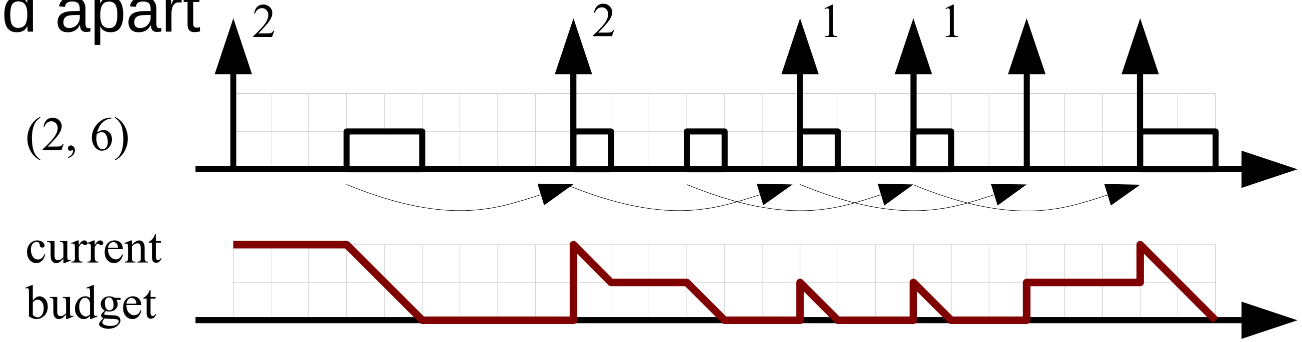


See CBS “unblock rule”

POSIX Sporadic Server

SCHED_SS

- Provides a form of temporal isolation
- Parameters: (Q, P, RT Priority, Low RT Priority)
- Budget exhausted => lower the priority till next recharge
- For every time interval in which the task executes, post a recharge of budget equal to the consumed CPU time one period apart



SCHED_SS may be analysed using FP techniques

- Patching the standard for getting rid of the "bug"

Usage Patterns

SCHED_DEADLINE

```
struct sched_param_ex sp = {
  .sched_runtime = runtime_ts;      // struct timespec
  .sched_deadline = deadline_ts;    // struct timespec
  .flags = 0;
};
sched_setscheduler_ex(pid, SCHED_DEADLINE, &sp);
/* Now you get runtime_ts every deadline_ts guaranteed */
```

Usage Patterns

IRMOS Scheduler

Pre-requisite at run-time: mount cgroups

- mkdir /cg
- mount -t cgroup -o cpu,cpuacct cgroup /cg

Reduce runtime for root-level tasks

- echo 200000 > /cg/cpu.rt_rt_task_runtime_us
  (root-group runtime remains at default of 950000)

Create group, with reservation of 10ms every 100ms

- mkdir /cg/g1
- echo 100000 > /cg/g1/cpu.rt_period_us
- echo 10000 > /cg/g1/cpu.rt_runtime_us
- echo 100000 > /cg/g1/cpu.rt_task_period_us
- echo 10000 > /cg/g1/cpu.rt_task_runtime_us

Attach task with tid=1421

- echo 1421 > /cg/g1/tasks

Detach task

- echo 1421 > /cg/tasks

Attach process with pid=1700

- for tid in `ls /proc/1700/task`; do echo $tid > /cg/g1/tasks; done

Destroy group

- rmdir /cg/g1

IRMOS RT Scheduler Design Goals

Replace real-time throttling

Tight integration in Linux kernel

- Modification to the Linux RT scheduler

Reuse as many Linux features as possible

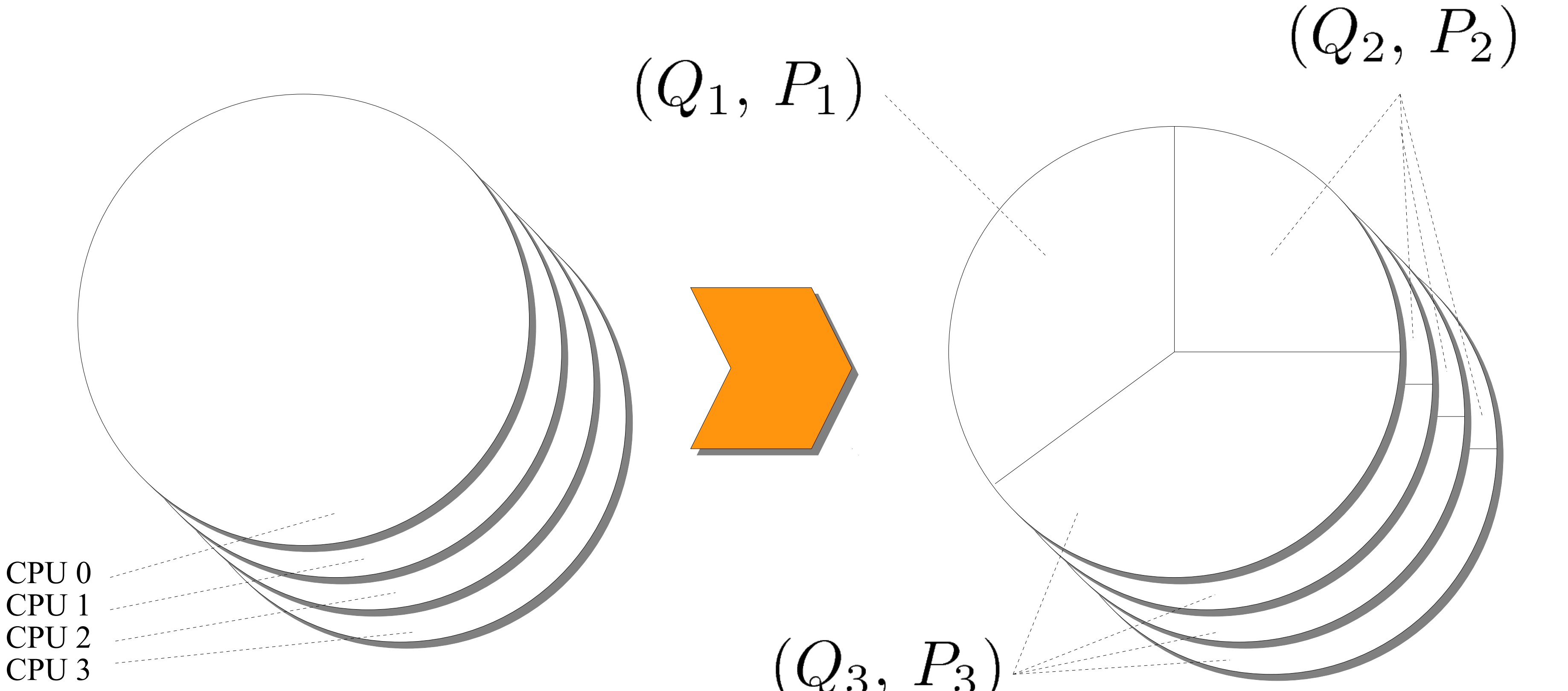
- Management of task hierarchies and scheduling parameters via **cgroups**
- **POSIX compatibility** and API

Efficient for SMP

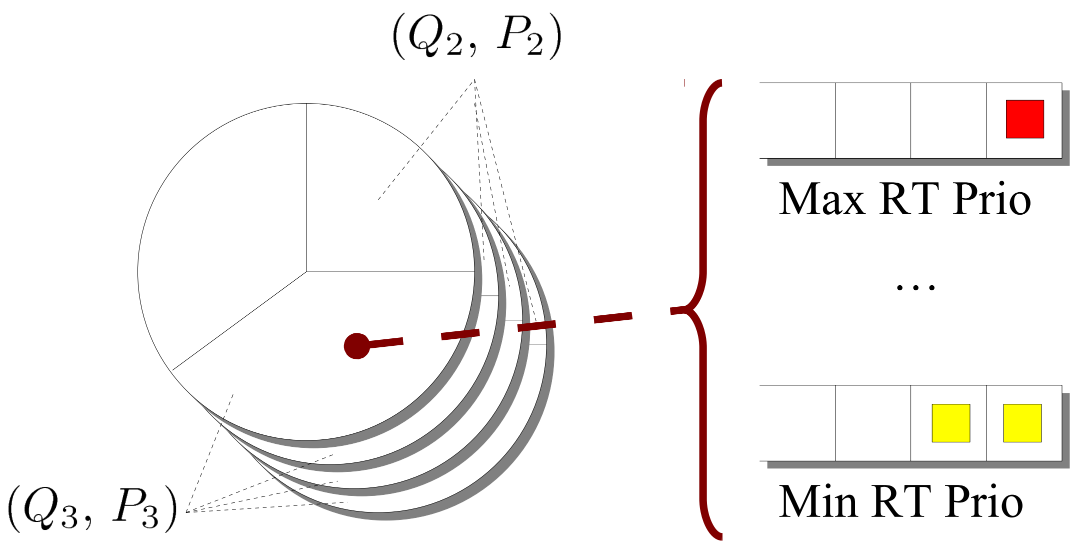
- Independent runqueues

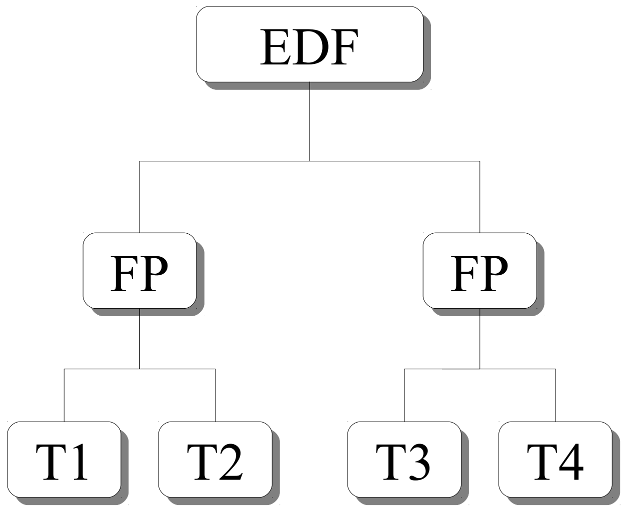
IRMOS Scheduler

Slice the available computing power into reservations



Hierarchical Scheduling





Needed operations

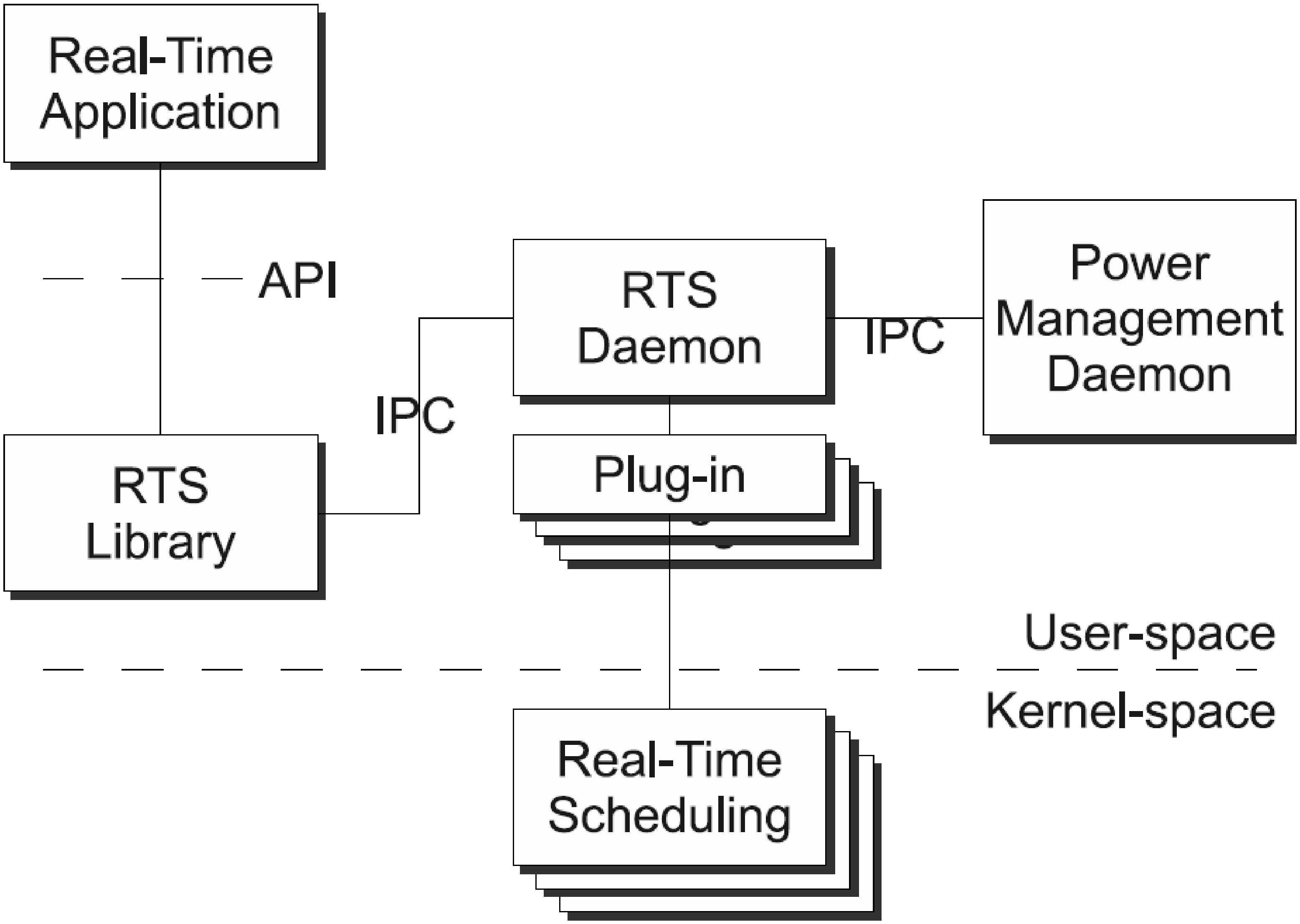
- **create** & **destroy** reservations
- **attach** & **detach** tasks ↔ reservations
- **list tasks** attached to reservations (and **list reservations**)
- Standard operations: **get & set parameters**

Other Features

Warning: features & parameters may easily grow

- Addition of parameters, such as
  - **deadline**
  - **desired** vs **guaranteed** runtime (for **adaptive reservations**)
- Set of **flags** for controlling variations on behaviour
  - **work conserving** vs **non-conserving** reservations
  - what happens at **fork()** time
  - what happens on tasks **death** (**automatic reclamation**)
  - **notifications** from kernel (e.g., **runtime exhaustion**)
- **Controlled access** to RT scheduling by **unprivileged applications** (e.g., per-user “quotas”)
- **Monitoring** (e.g., residual runtime, available bandwidth)
- Integration/interaction with **power management**

Proposed API for applications



Related Publications

- *Hierarchical Multiprocessor CPU Reservations for the Linux Kernel*
  F. Checconi, T. Cucinotta, D. Faggioli, G. Lipari
  OSPERT 2009, Dublin, Ireland, June 2009
- *An EDF Scheduling class for the Linux kernel*
  D. Faggioli, F. Checconi, M. Trimarchi, C. Scordino
  RTLWS 2009, Dresden, October 2009
- *Access Control for Adaptive Reservations on Multi-User Systems*
  T. Cucinotta
  RTAS 2008, St. Louis, MO, United States, April 2008
- *Self-tuning Schedulers for Legacy Real-Time Applications*
  T. Cucinotta, F. Checconi, L. Abeni, L. Palopoli
  EuroSys 2010, Paris, April 2010
- *Respecting temporal constraints in virtualised services*
  T. Cucinotta, G. Anastasi, L. Abeni
  RTSOAA 2009, Seattle, Washington, July 2009

Thanks for your attention

Help!!!

http://retis.sssup.it/people/tommaso