




Adaptive
Real-Time Scheduling:
Why and How

Giorgio Buttazzo
University of Pavia

Motivations

Handling complexity

- Most modern embedded systems are characterized by
 - ⇒ high degree of complexity
 - ⇒ dynamic behavior
- 
- High complexity increases the risk of low reliability
 - ⇒ Proper support at different levels (OS, network, middleware, ...)
 - Dynamic behavior causes unpredictable overloads
 - ⇒ Efficient resource management
 - ⇒ Adaptive strategies

Examples: Consumer Electronics

- Mobile terminals are getting more advanced by the hour: new features added at a high speed.
- SW code is constantly increasing. Today a phone consists of several million lines of code involving large number of concurrent activities.
- Getting all this to work is not trivial task and it is not getting any easier.

Risks

Due the increased SW complexity, CE products (traditionally robust, predictable and easy to use) may loose these features.

General Issues

- Support for complexity control
 - ⇒ Modularity
 - ⇒ Composability
- Increasing SW dimension requires
 - ⇒ optimal use of resources
 - ⇒ overload management
- Rapid systems evolution requires platforms to be: highly configurable, reusable, scalable, and easily portable
- Other important issues include: power consumption and size and security.

Challenges

1. To control complexity we need to better specify system activities

If systems are better specified, their properties can be verified more easily

Actions

- Get rid of priorities
- Allow specification of resource usage

Get rid of priorities

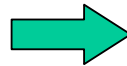
1a. Drawbacks of priorities

- Priorities are often not realistic and mapping QoS specs to priorities is often not trivial
 - ⇒ QoS specs are multi-dimensional (information loss)
 - ⇒ Priorities introduce constraints that affect schedulability
- System analysis is difficult (except for trivial cases)
- Enforcing a priority order does not mean satisfying performance requirements
 - ⇒ They depend on the execution behavior
- Risk of starvation of low pr. tasks during overloads

Explicit Resource Reservation

A much better approach is to use a programming model that enables the designer to explicitly control the resources assigned to a given activity.

**Resource
Reservation
(RR)**

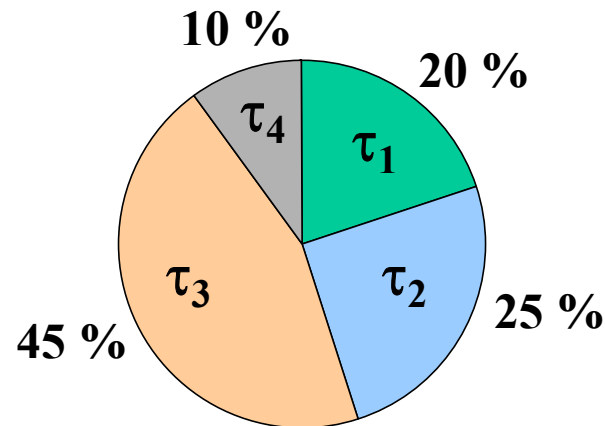


Resource partition

Resource enforcement

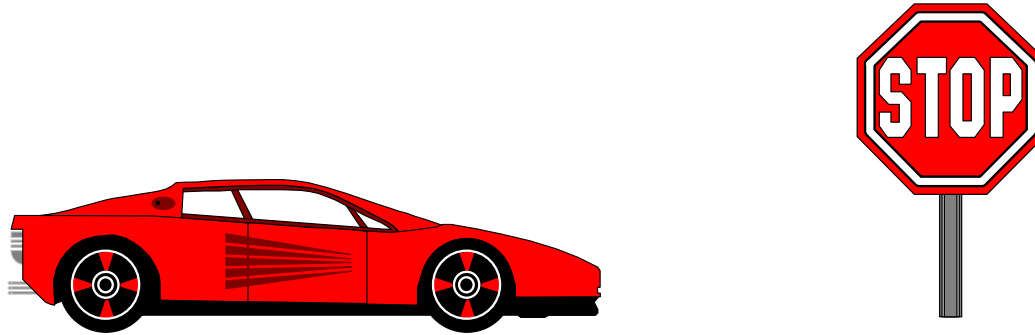
Resource Partition

- Partition each resource among system activities, based on performance and execution requirements:



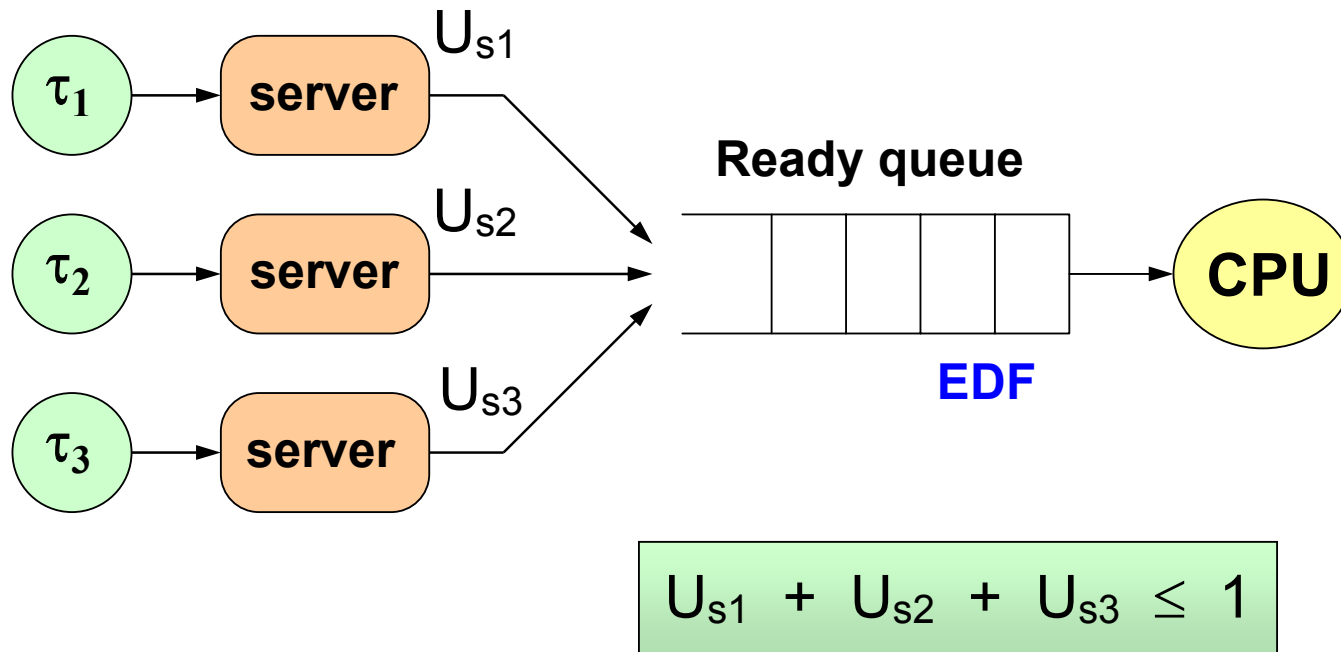
Each task receives a fraction U_i of the processor and behaves as it were executing alone on a slower processor of speed U_i

Resource Enforcement



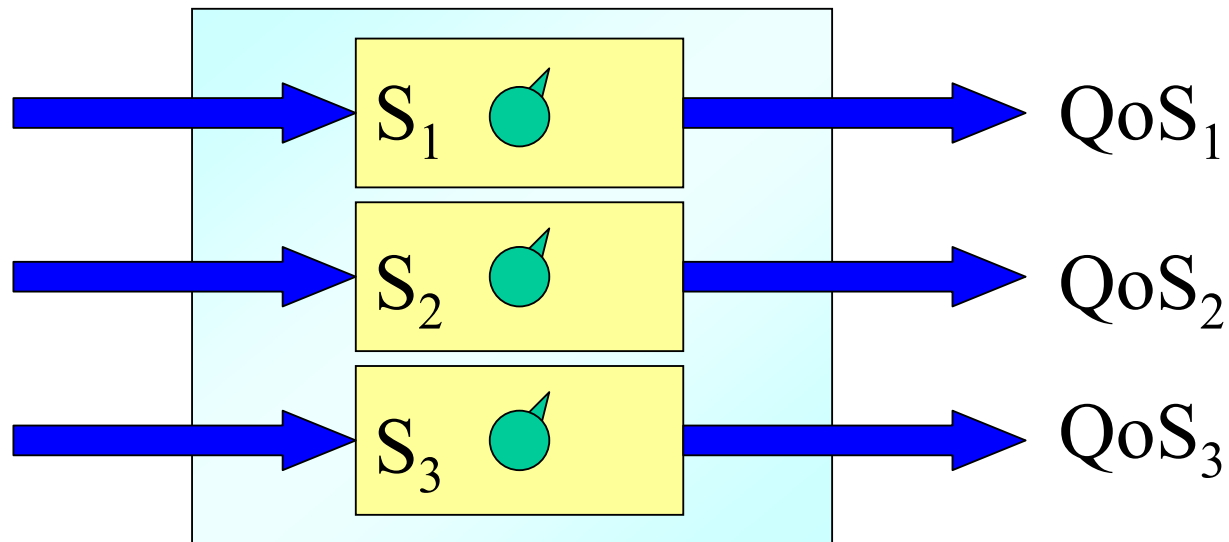
- It is a mechanism that makes sure that each task does not consume more than its reserved amount.
- If a task attempts to execute more, it must be delayed, preserving the resource for the other tasks.

Implementation



Resource Reservation

The QoS depends on the amount of resources reserved for the service.

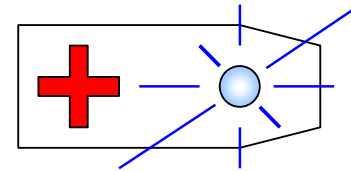
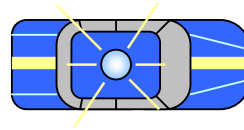


Benefits of resource reservation

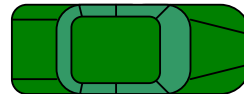
1. Resource allocation is easier than priority mapping
2. Temporal isolation (temporal protection)
 - Important for modularity and scalability
3. Simpler schedulability analysis
4. Easier probabilistic approach

Priority vs. *RR*: tasks as vehicles

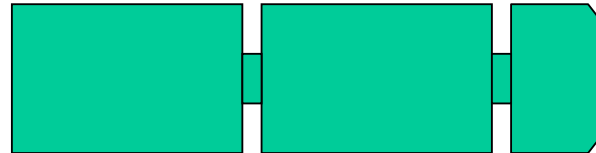
High priority



Medium priority



Low priority

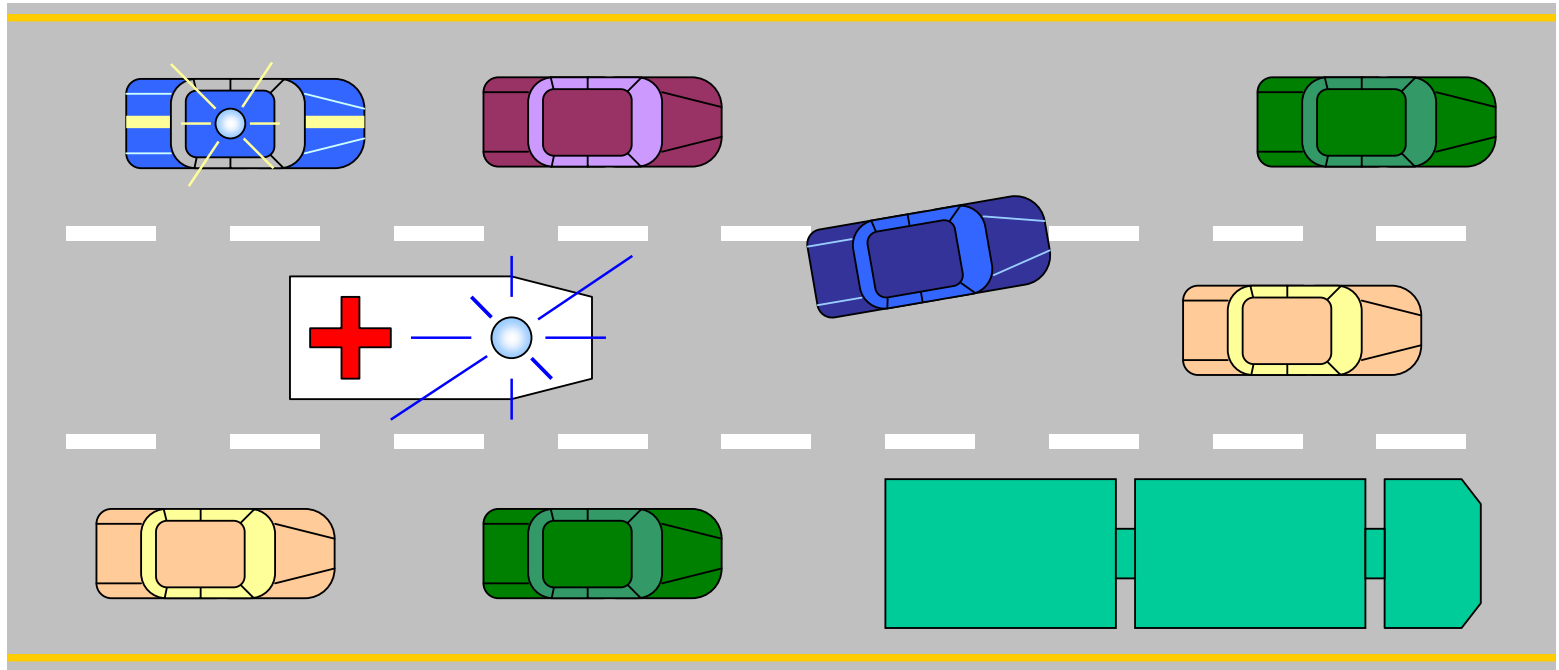


Shared resource

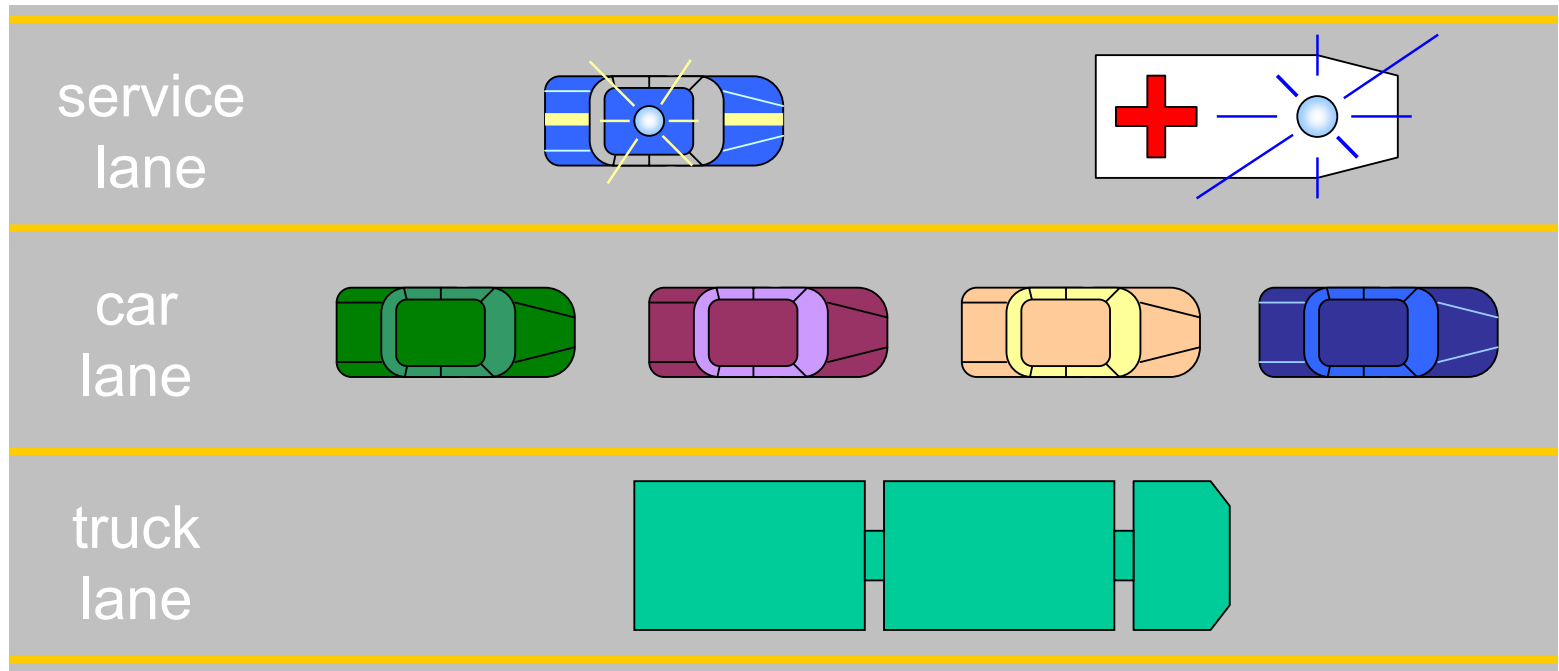


The priority approach

Problems in overload conditions

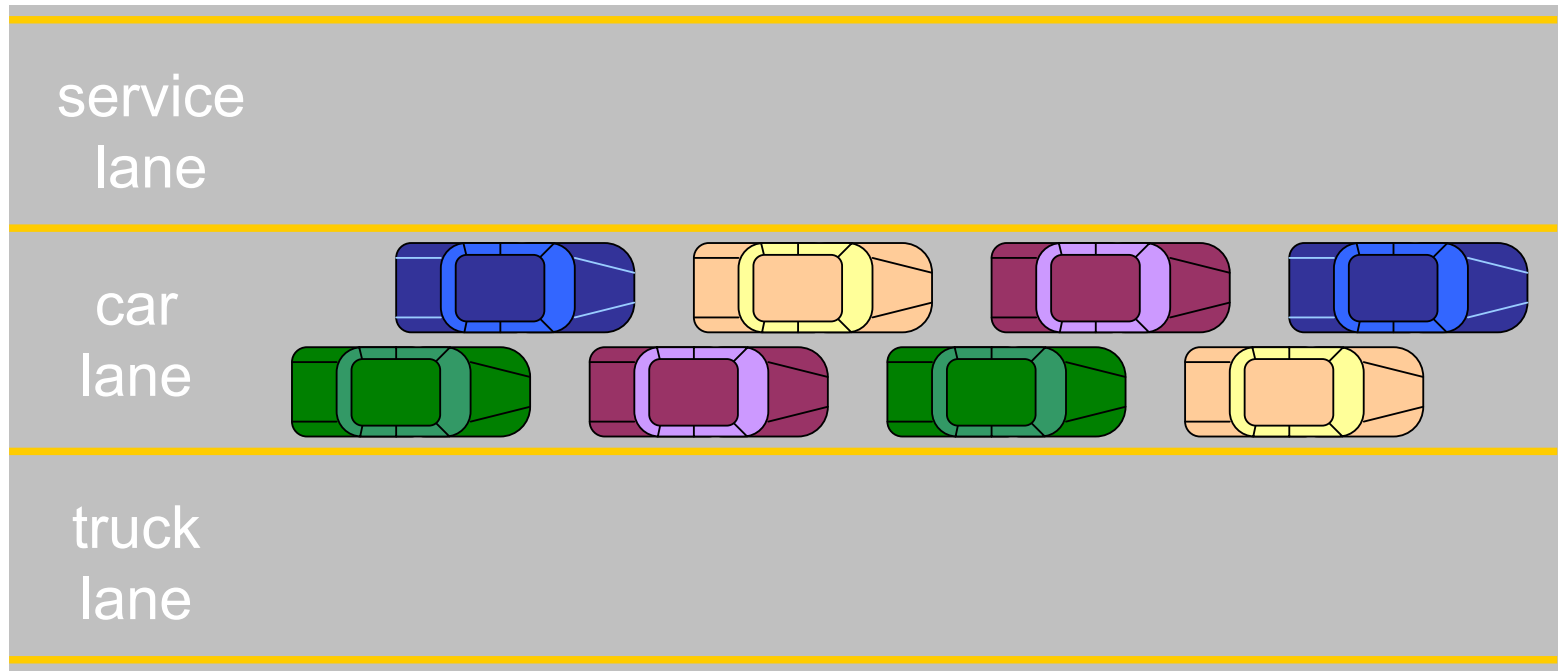


The *RR* approach



RR: potential problems

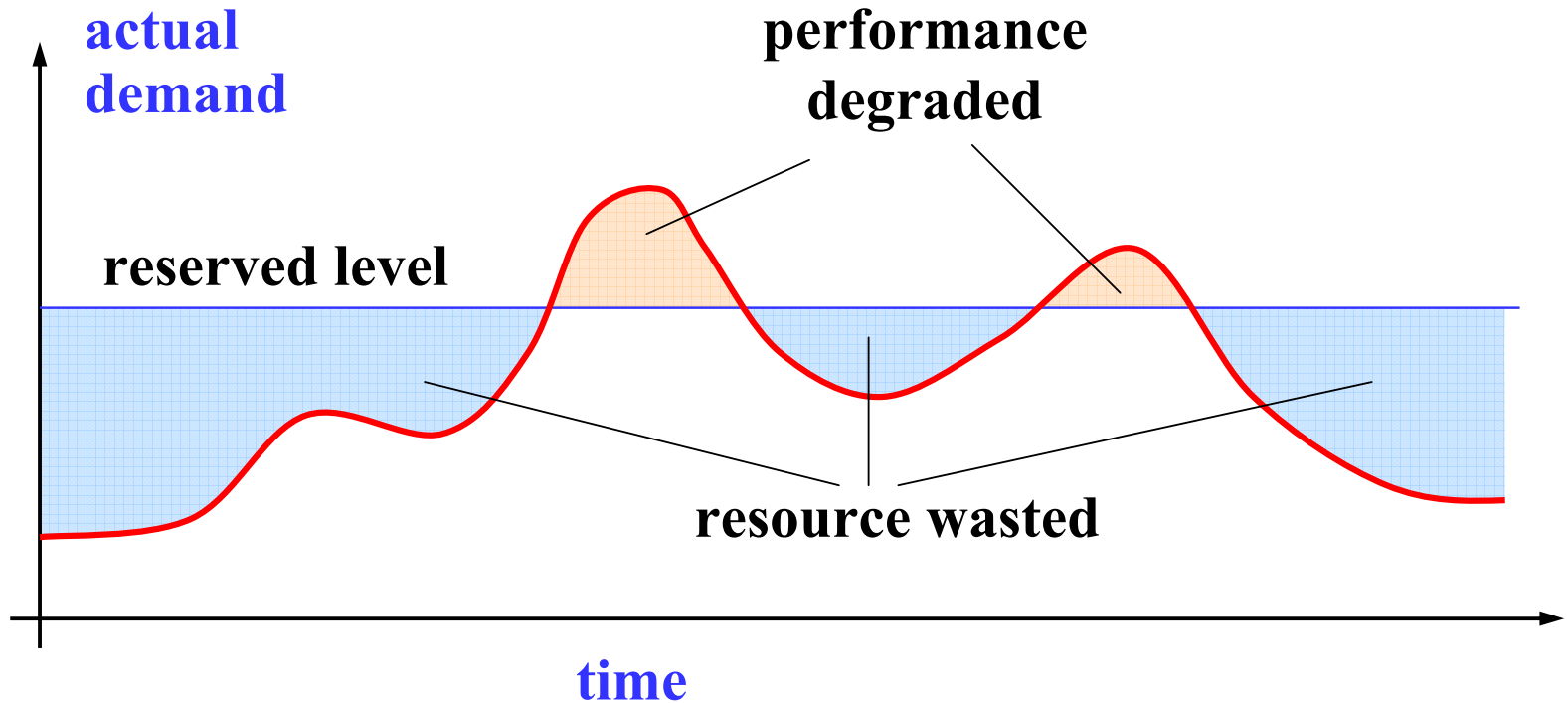
A resource is wasted if not used



Potential problems

1. A reserved resource is wasted if not used
2. How to cope with wrong reservations?
 - Not enough reservation \Rightarrow Performance degradation
 - Too much reservation \Rightarrow resource waste
3. How about dynamic changes?

Problem #1: reservation \Rightarrow waste

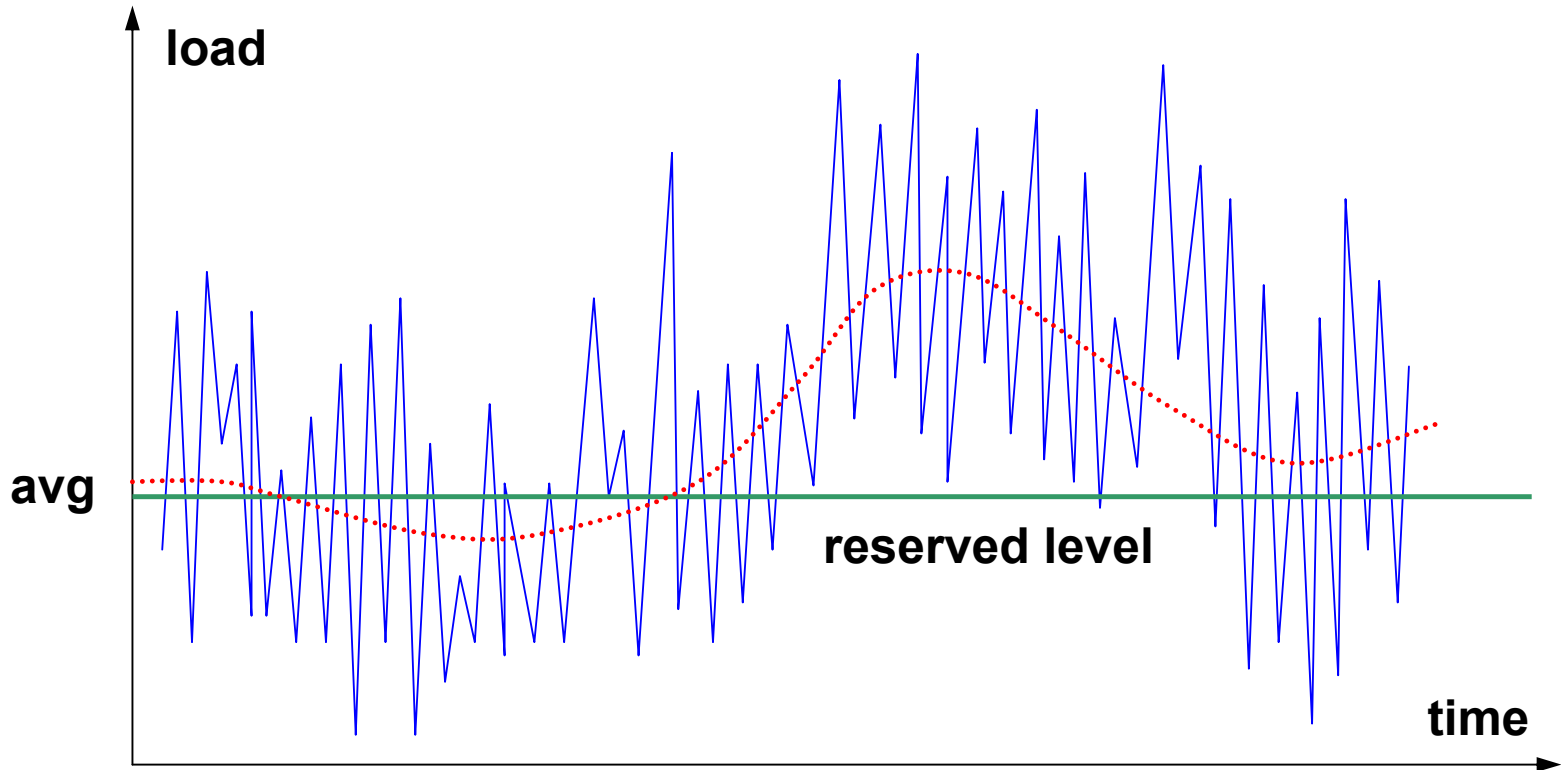


Solution #1

If task requirements vary significantly over time:

Efficient resource reclaiming is needed to exploit unused resources in favor of the most demanding tasks.

Problems #2-3: wrong reservations and dynamic behavior



Solution #2-3

To cope with wrong reservations or dynamic changes, resources need to be adapted on line:

- **A monitoring mechanism is needed to probe the actual demand**
- **A feedback mechanism is required to perform resource adaptation**

Summarizing

Predictability + high efficiency

can be achieved by:

- ⇒ Resource Reservations in place of priorities
- ⇒ Resource Reclaiming to avoid waste
- ⇒ Feedback scheduling to cope with wrong reservations and dynamic changes