

Research Challenges in Exploiting Multi-Core Platforms for Real-Time Applications

Giorgio Buttazzo



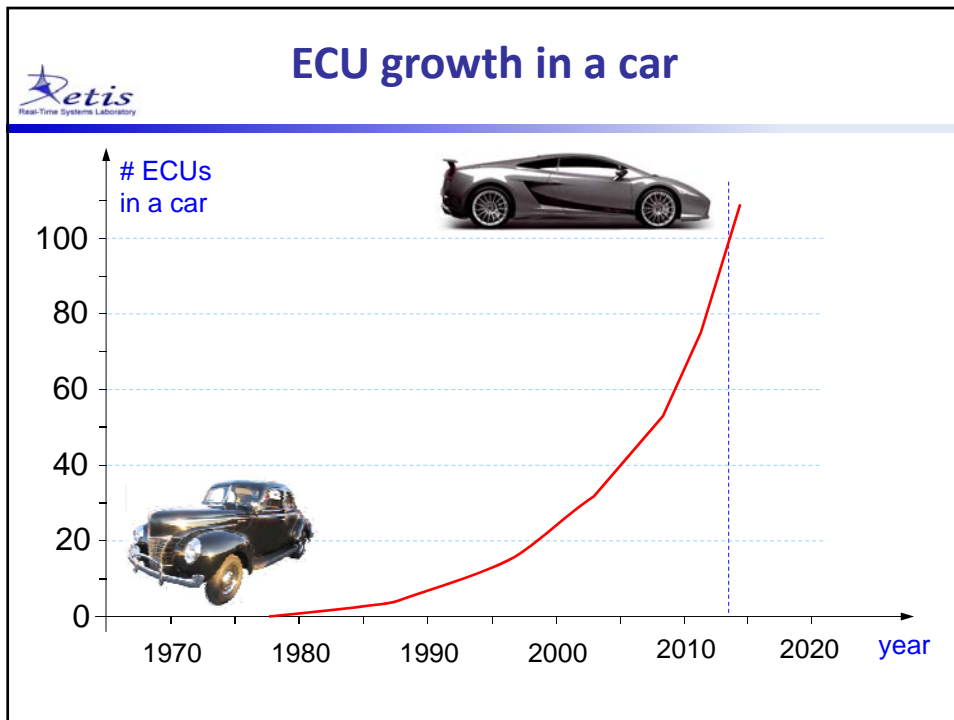
Scuola Superiore Sant'Anna, Pisa



Evolution of Embedded Systems

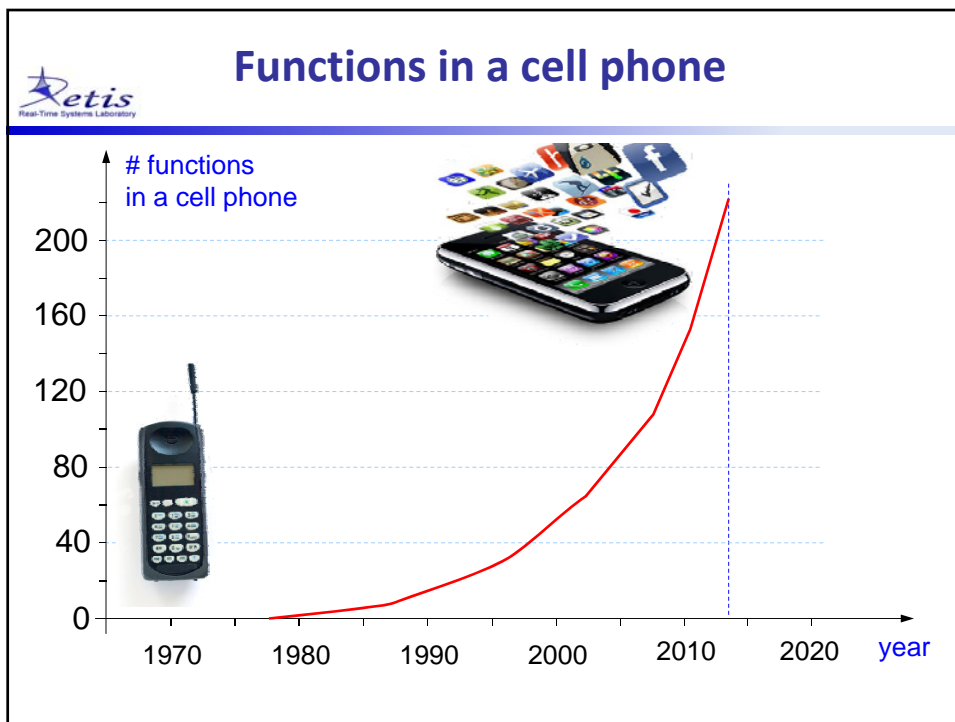
The complexity of embedded systems has grown exponentially in several application domains:






Software in a car

▪ Engine:	ignition, fuel pressure, water temperature, valve control, gear control,
▪ Dashboard:	engine status, message display, alarms
▪ Diagnostic:	failure signaling and prediction
▪ Safety:	ABS, ESC, EAL, CBC, TCS
▪ Assistance:	power steering, navigation, sleep sensors, parking, cruise control, collision detection
▪ Comfort:	fan control, air conditioning, music, regulations: steer/lights/sits/mirrors/glasses...



 **Observed trends**

Software:


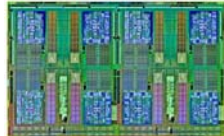

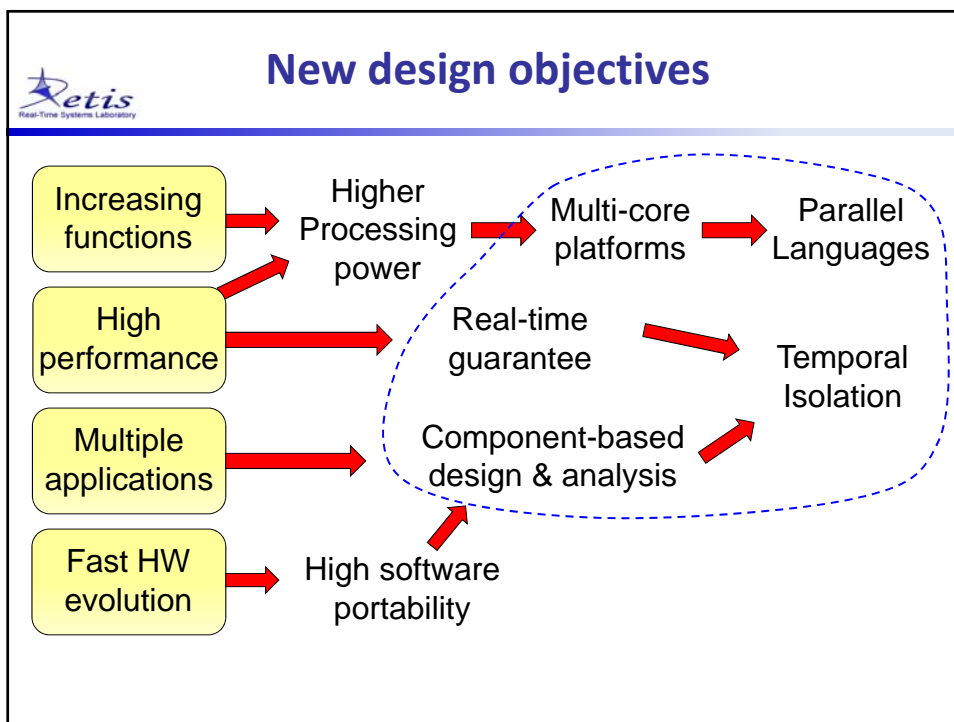
- increasing functionalities

Hardware:

- multi-core platforms and heterogeneous systems

Requirements:

- high-performance and real-time



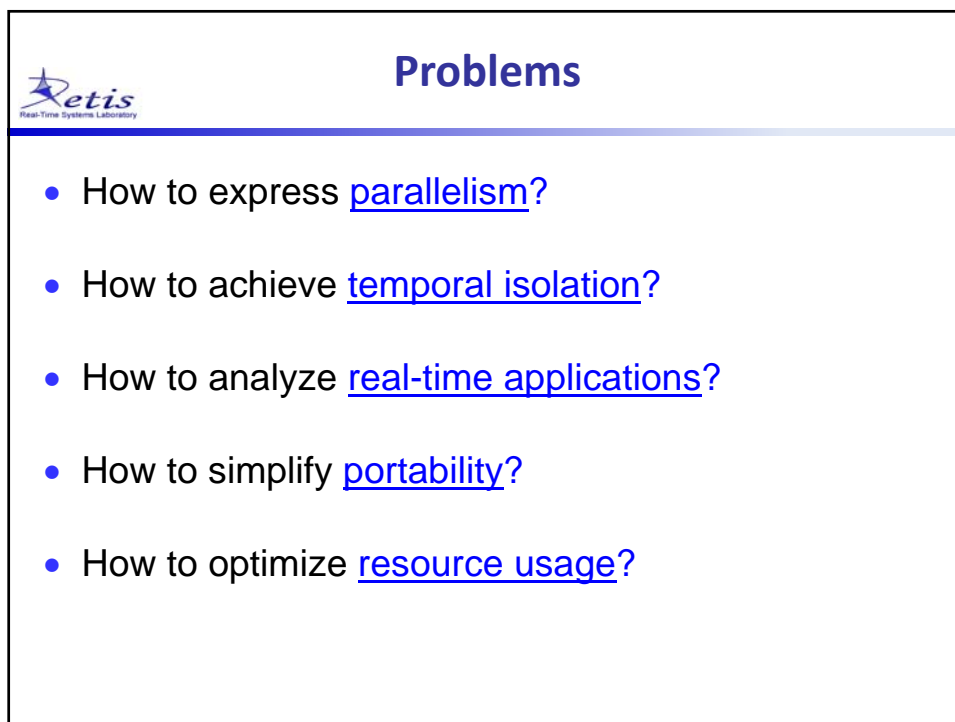
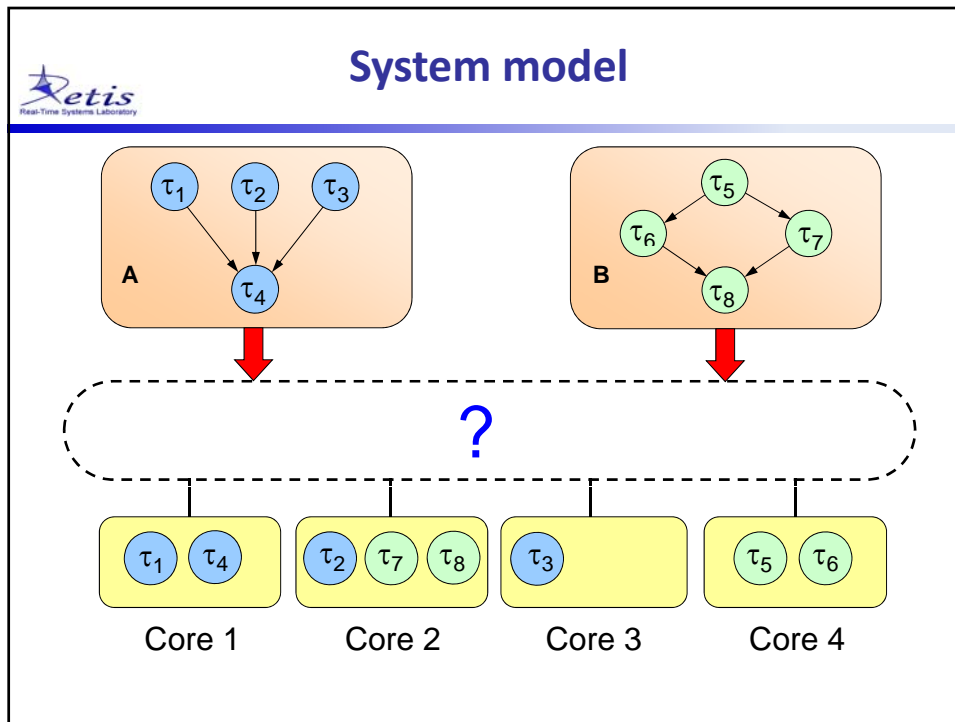
Required support

- [Automatize allocation](#) of parallel applications
- Simplify [portability](#) by proper abstraction layers
- [Optimize resources](#) (processing, memory, energy)
- [Isolate](#) the timing behavior of different applications
- Provide [offline guarantee](#) for real-time applications



This talk

- Illustrates a [general framework](#) to address such multiple objectives;
- Presents a number of [consolidated methodologies](#) for handling specific problems;
- States a number of [open problems](#) for further research.





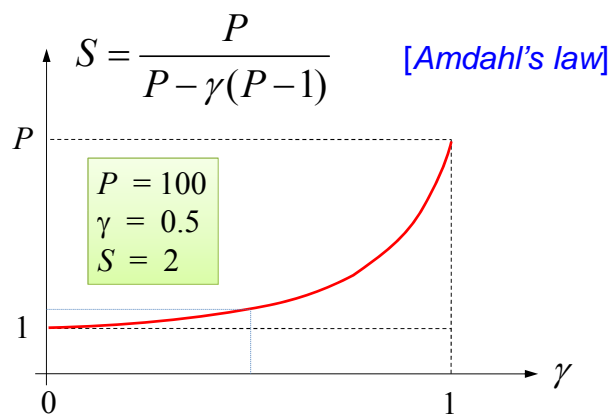
Exploiting parallelism

- As we are entering the multicore era, sequential languages (as C/C++) are no longer the most appropriate way to specify programs.
- In fact, a sequential language hides the intrinsic concurrency that must be exploited to improve the performance of the system.



Speed-up factor

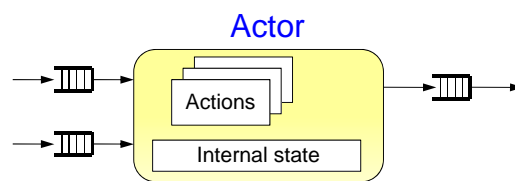
- γ fraction of parallel code
 $S(P)$ speed-up factor on P processors



Expressing parallelism

Parallelism can be expressed by using a suitable [dataflow language](#), like CAL [UC Berkeley, 2003].

- It describes algorithms through a set of modular components ([actors](#)), communicating through I/O ports:



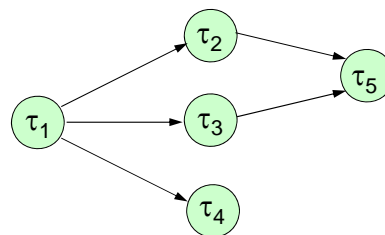
- Actions read input tokens, modify the internal state, and produce output tokens.

Application model


- An application can be modeled as a [task graph](#) with precedence relations:

Task τ_i

A sequential portion of code that cannot be further parallelized

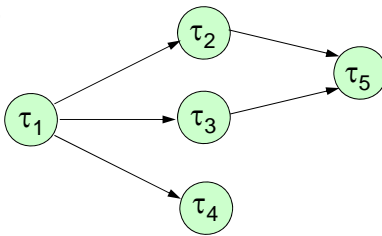



A task graph specifies the [maximum level of parallelism](#)

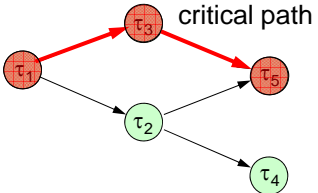
 **Assumptions and parameters**

- **Arrival pattern**
 - **Periodic** (activations exactly separates by a period T)
 - **Sporadic** (Minimum Interarrival Time T)
 - **Aperiodic** (no interarrival bound exists)
- Is **preemption** allowed at arbitrary times?
- Is task **migration** allowed?

Application parameters:
 $\{C_1, C_2, C_3, C_4, C_5\}, D, T$



 **Important factors**



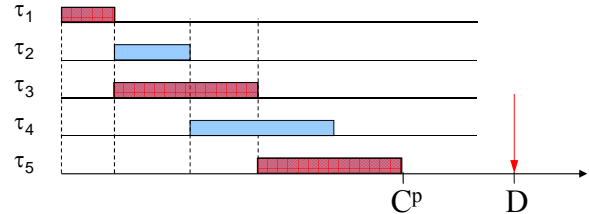
critical path

Sequential Computation time


$$C^s = \sum C_i$$

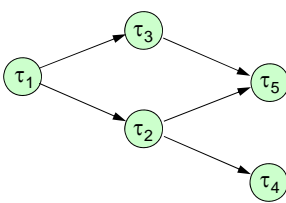
Parallel Computation time

$$C^p = \text{length of a critical path}$$



$(C^p > D) \Rightarrow A$ is not feasible in any number of cores
 $(C^s \leq D) \Rightarrow A$ is feasible on a single core

 **Important factors**




Sequential
Computation time $C^s = \sum C_i$


Parallel
Computation time $C^p = \text{length of a critical path}$

$U = \frac{C^s}{T}$ required bandwidth

$\lceil U \rceil \leq \text{Number of required cores} \leq \lceil 2U \rceil$

 **Outline**

- How to express parallelism
- How to achieve temporal isolation
- How to analyze real-time applications
- How to simplify portability
- How to optimize resource usage


 **Achieving Temporal Isolation**


Temporal Isolation

Property of a multi-application system in which the performance of an application does not depend on the execution of the others.

The performance of an application only depends on:

- Its own computational demand;
- The amount of allocated resources.

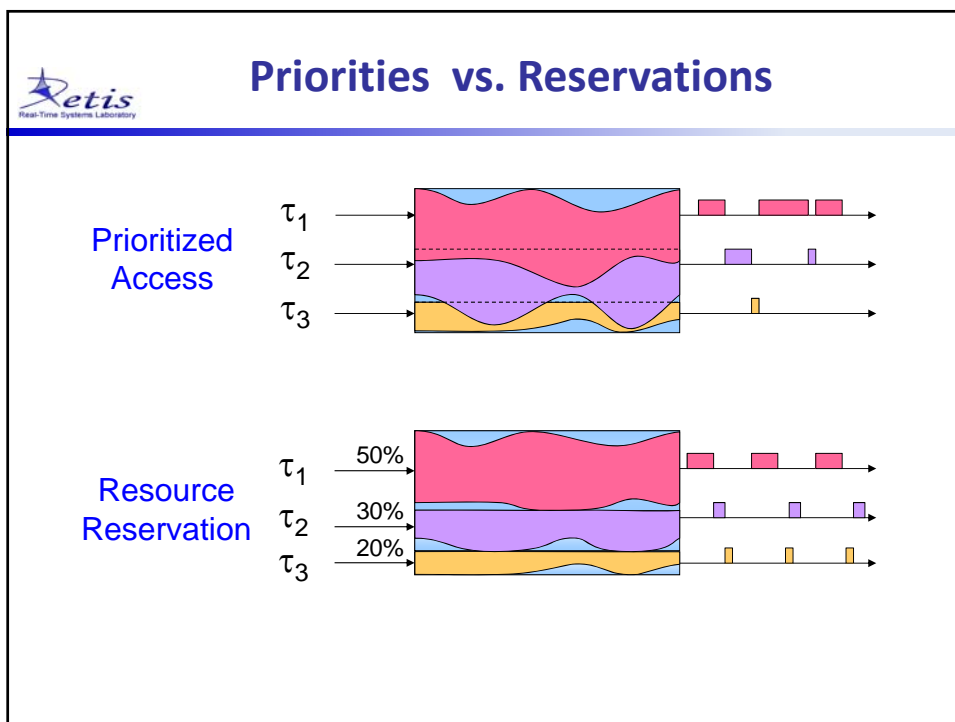
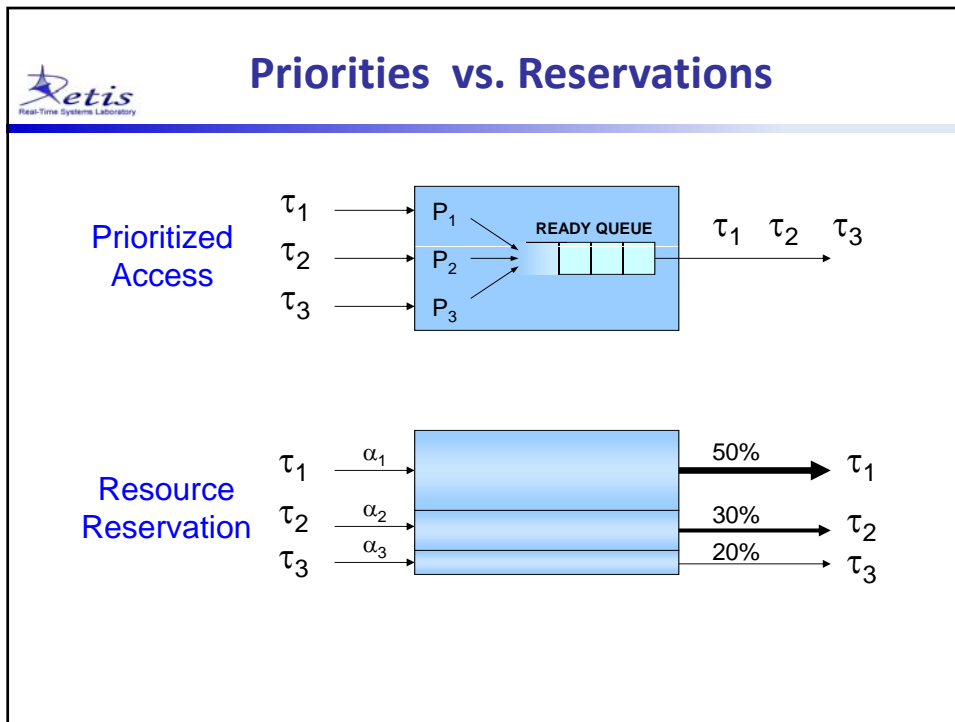
 Resource Reservation


 **Achieving Temporal Isolation**

An isolated application executes as it were executing alone on a slower dedicated processor of speed s equal to the reserved fraction.

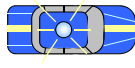
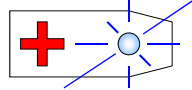
Advantages

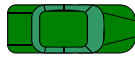
- **Predictability:** A misbehavior of an application does not affect the others.
- **Modular analysis:** RT constraints can be verified independently of the knowledge of other applications.

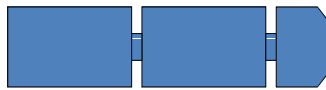



 **Priorities vs. Reservations**

Tasks as vehicles


High priority  

Medium priority 

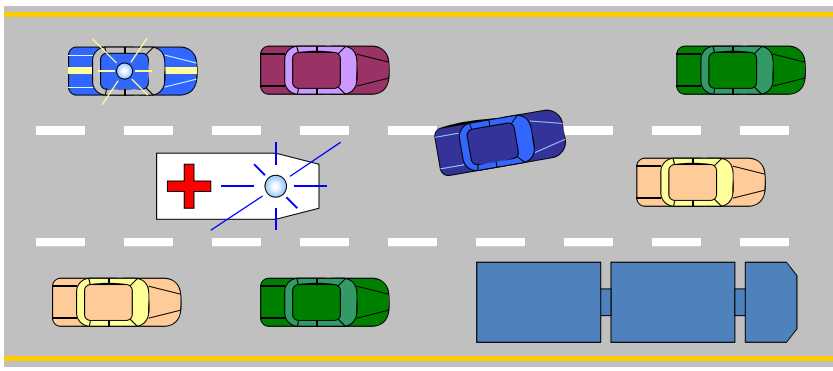
Low priority 

Shared resource 


25

 **Priorities vs. Reservations**

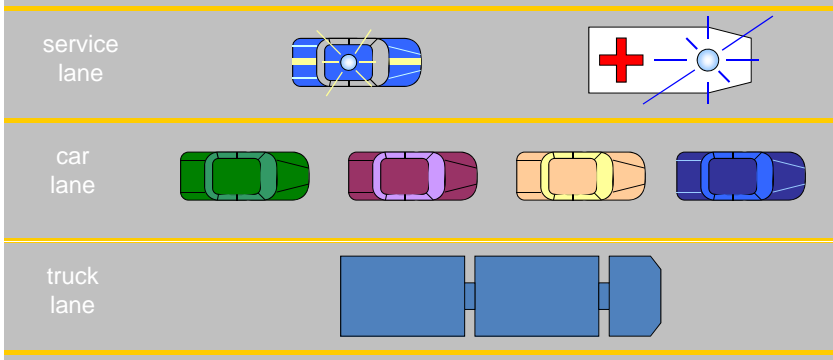
Priority: Problems in overload conditions



26

 **Priorities vs. Reservations**

Reservation: Less interference




service lane

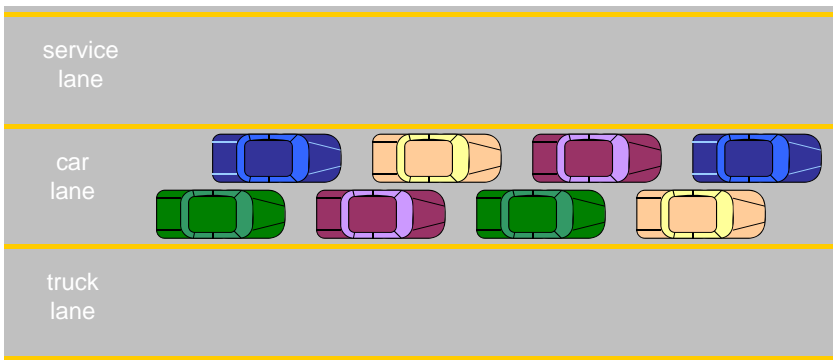
car lane

truck lane

27

 **Priorities vs. Reservations**

Reservation: but a resource is wasted if not used

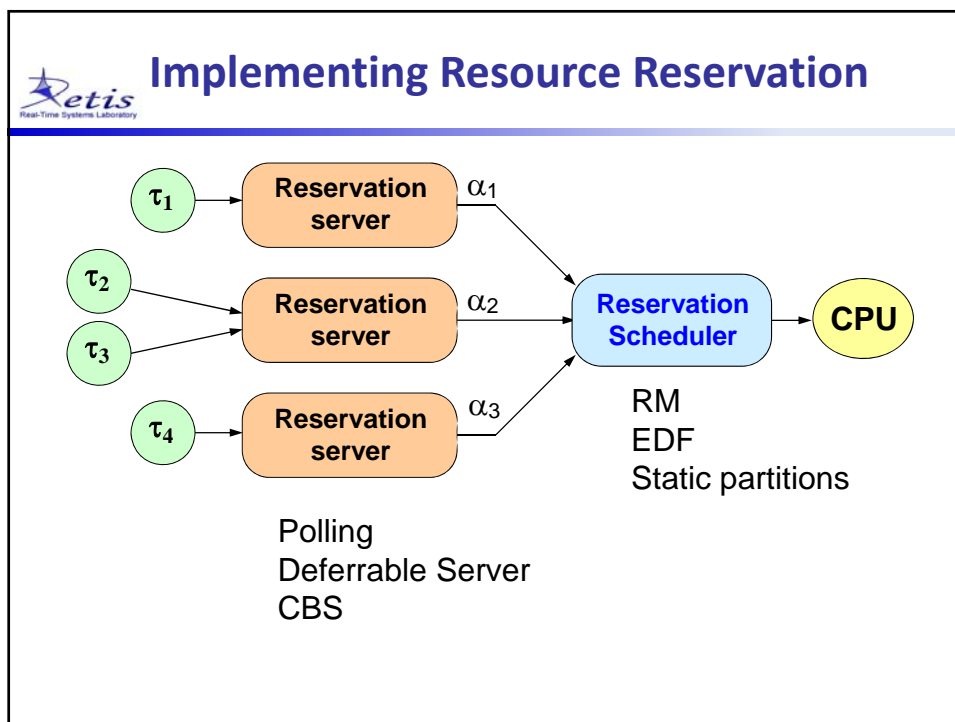
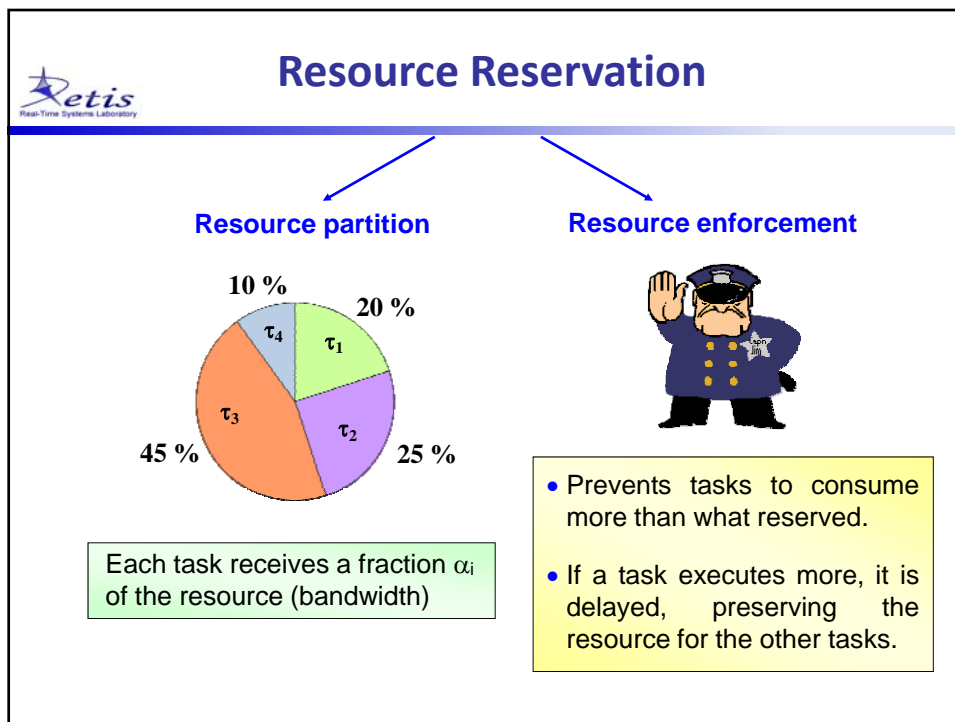



service lane

car lane

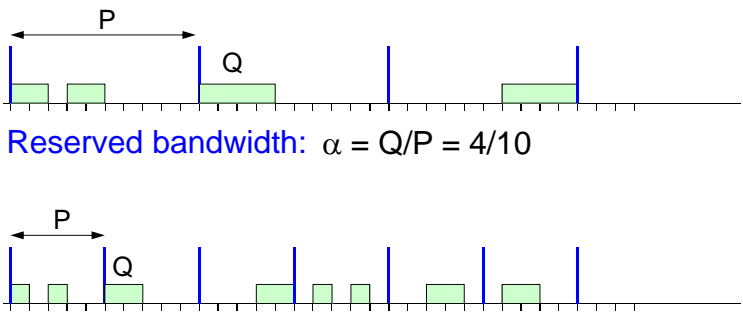
truck lane

28



 **Reservation server**


A way to implement a reservation is through a periodic server providing a **budget Q** every **period P**:



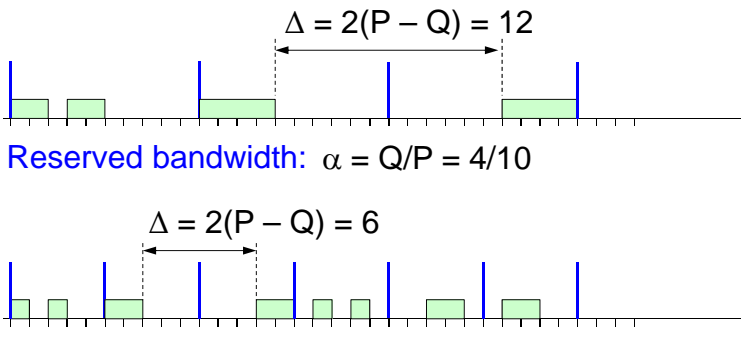
Reserved bandwidth: $\alpha = Q/P = 4/10$

Reserved bandwidth: $\alpha = Q/P = 2/5$

Which one is better?

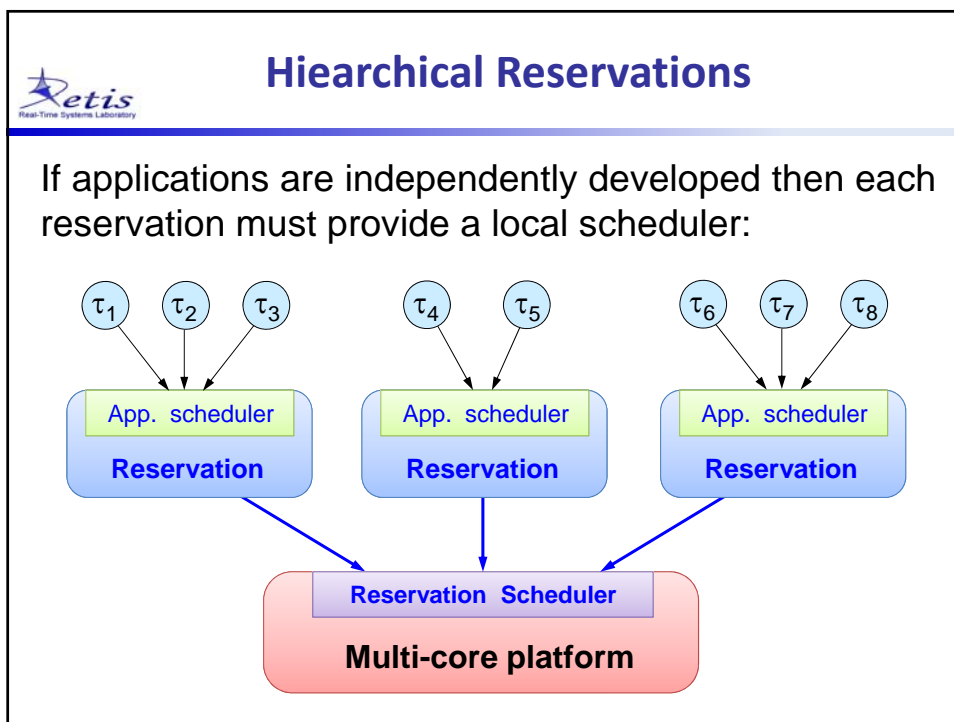
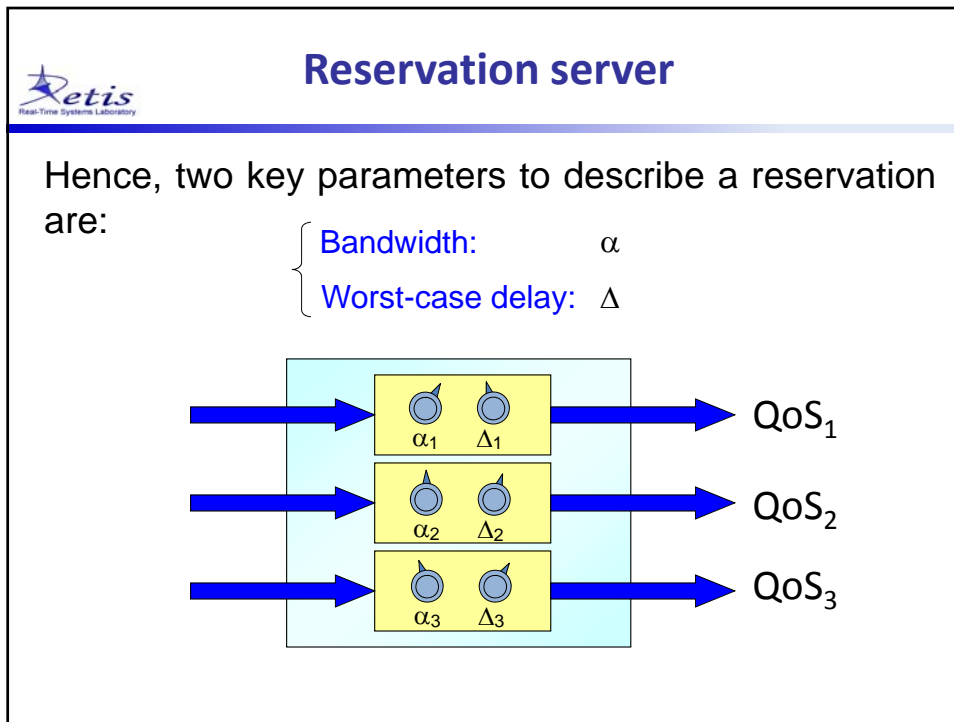
 **Reservation server**

Observe the worst-case **service delay Δ** :



Reserved bandwidth: $\alpha = Q/P = 4/10$

Reserved bandwidth: $\alpha = Q/P = 2/5$





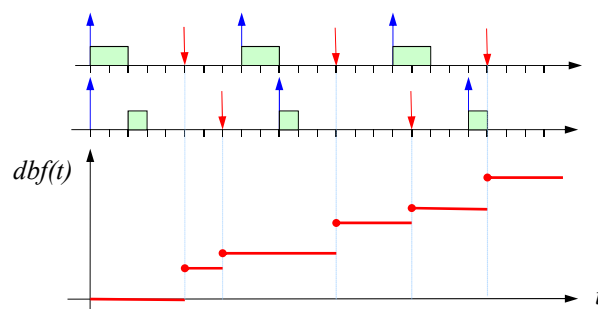
Outline

- How to express parallelism
- How to achieve temporal isolation
- How to analyze real-time applications
- How to simplify portability
- How to optimize resource usage



Schedulability Analysis

- Application demand is described through the demand bound function:

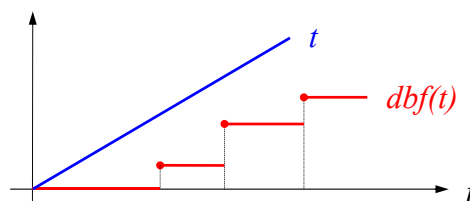


$$dbf(t) = \sum_{i=1}^n \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i$$

Schedulability Analysis

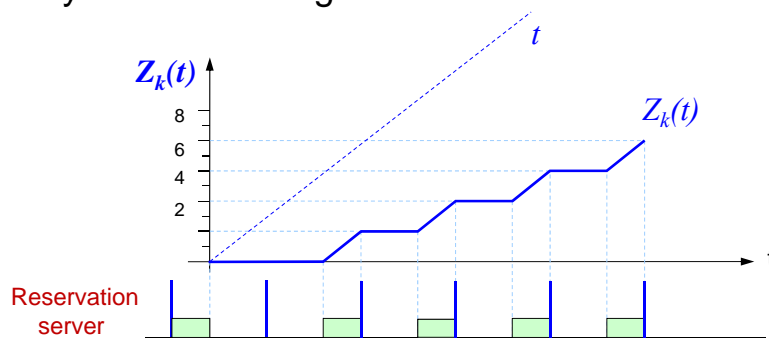
- On a single processor, feasibility is guaranteed under EDF if and only if:

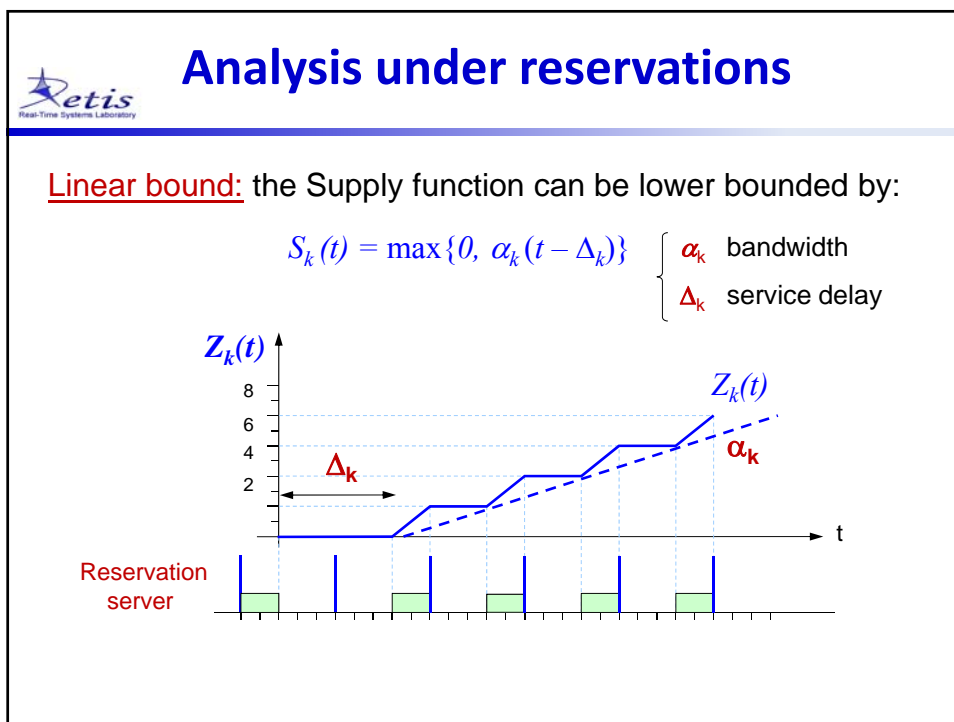
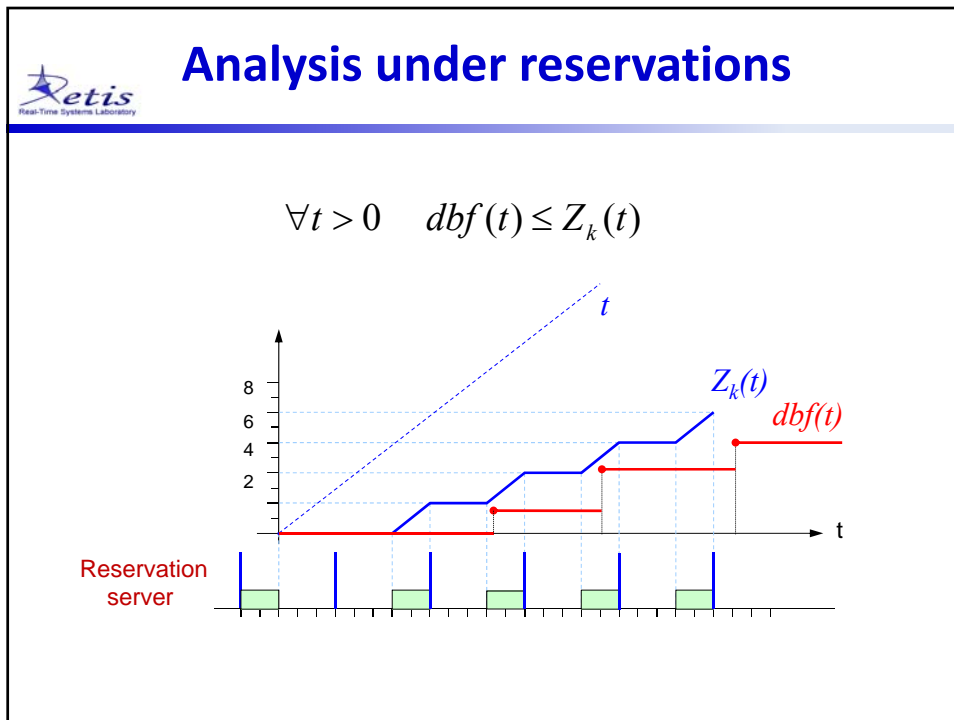
$$\forall t > 0 \quad dbf(t) \leq t$$



Analysis under reservations

Given a reservation R_k , the supply function $Z_k(t)$ gives the minimum amount of service time available in any interval of length t :

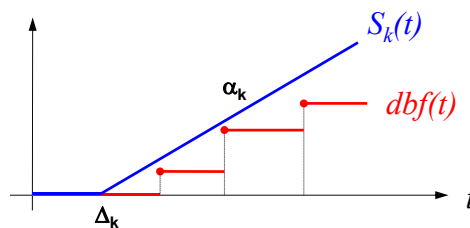




Schedulability Analysis

- On a reservation R_k , feasibility is guaranteed under EDF if:

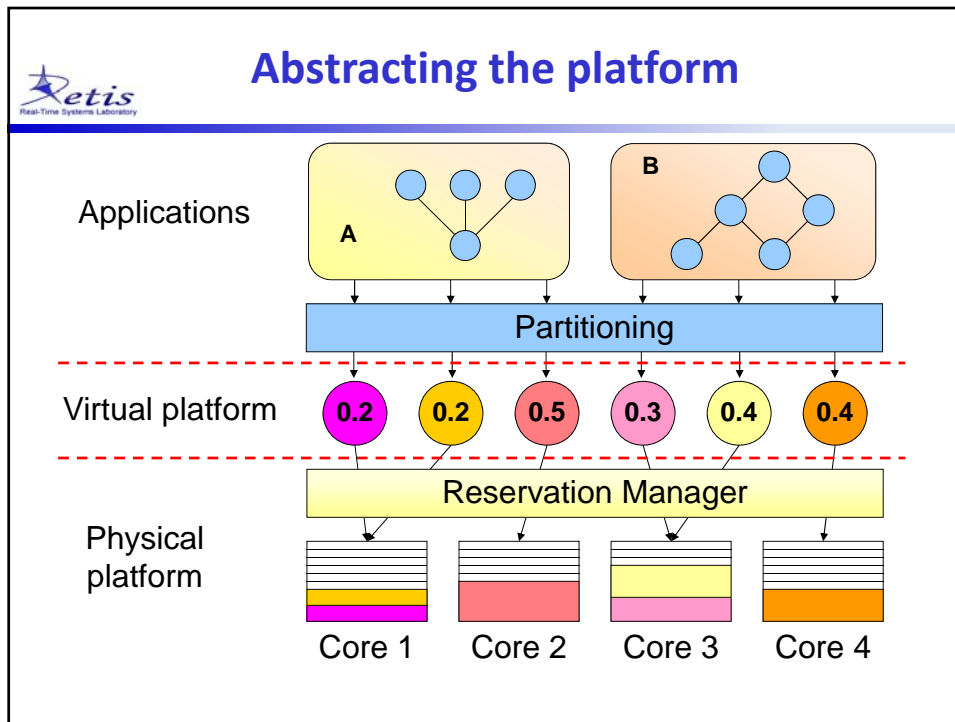
$$\forall t > 0 \quad dbf(t) \leq S_k(t)$$



$$dbf(t) = \sum_{i=1}^n \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i$$

Outline

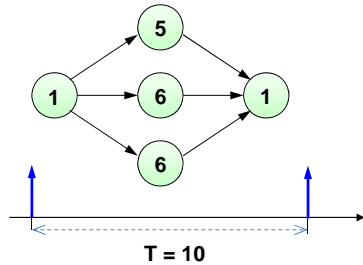
- How to express parallelism
- How to achieve temporal isolation
- How to analyze real-time applications
- How to simplify portability
- How to optimize resource usage



Multicore Reservations

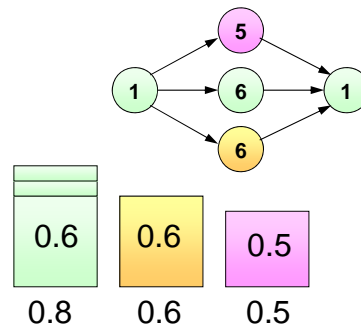
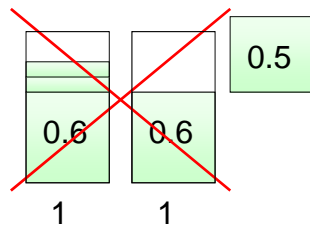
- How to extend reservations on multicore platforms?
- Does it make sense to define reservations with bandwidth $\alpha > 1$?

Partitioning is not trivial



$$C_s = \sum C_i = 19$$

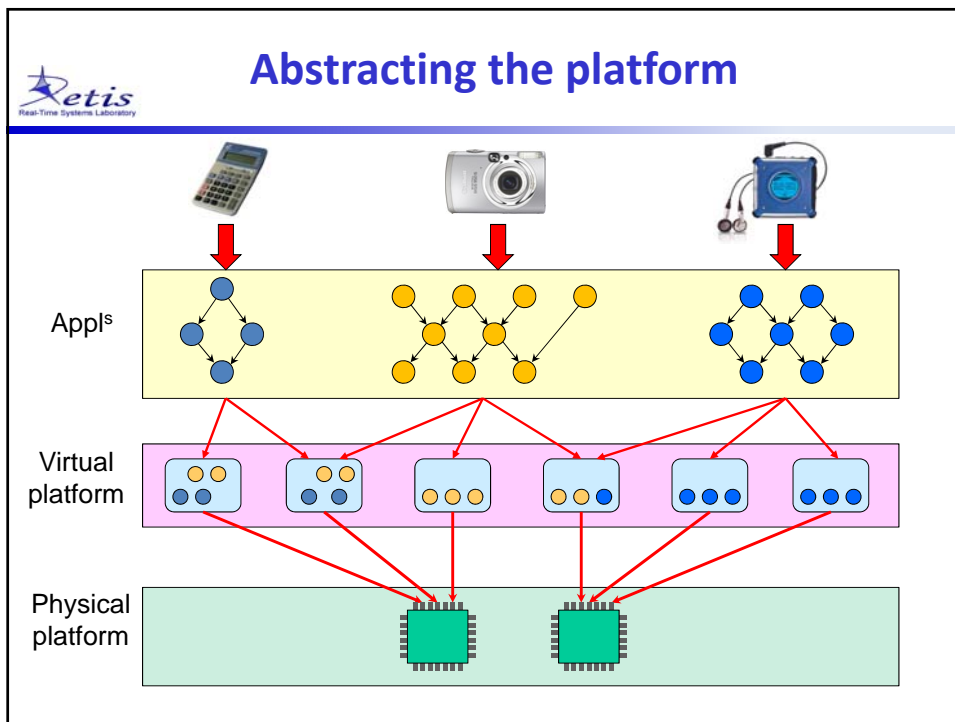
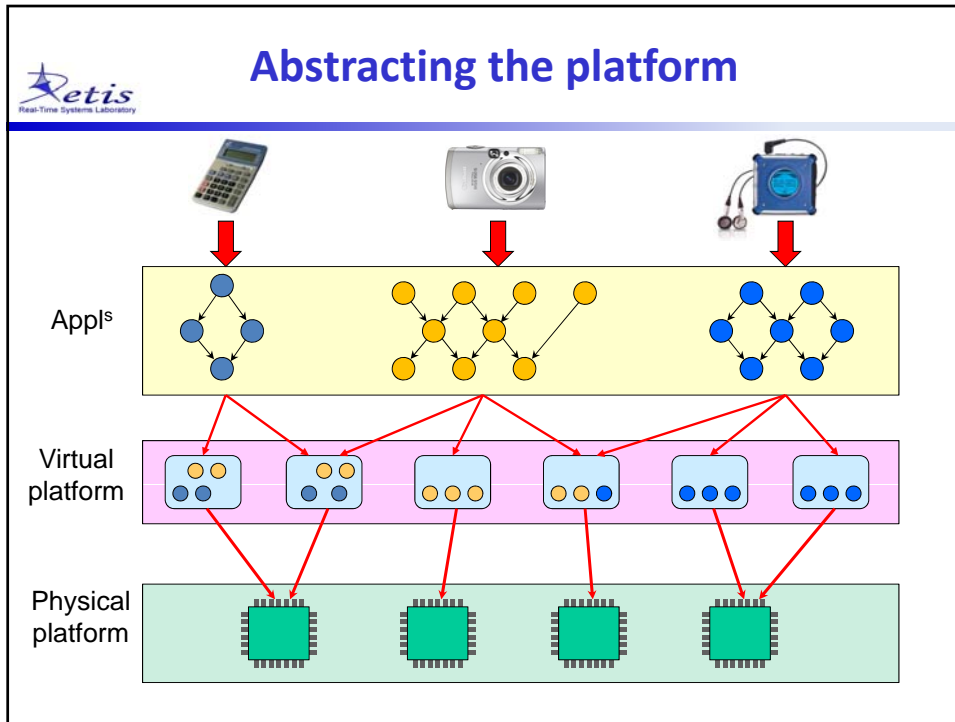
$$U = \frac{C^s}{T} = \frac{19}{10} = 1.9 < 2$$

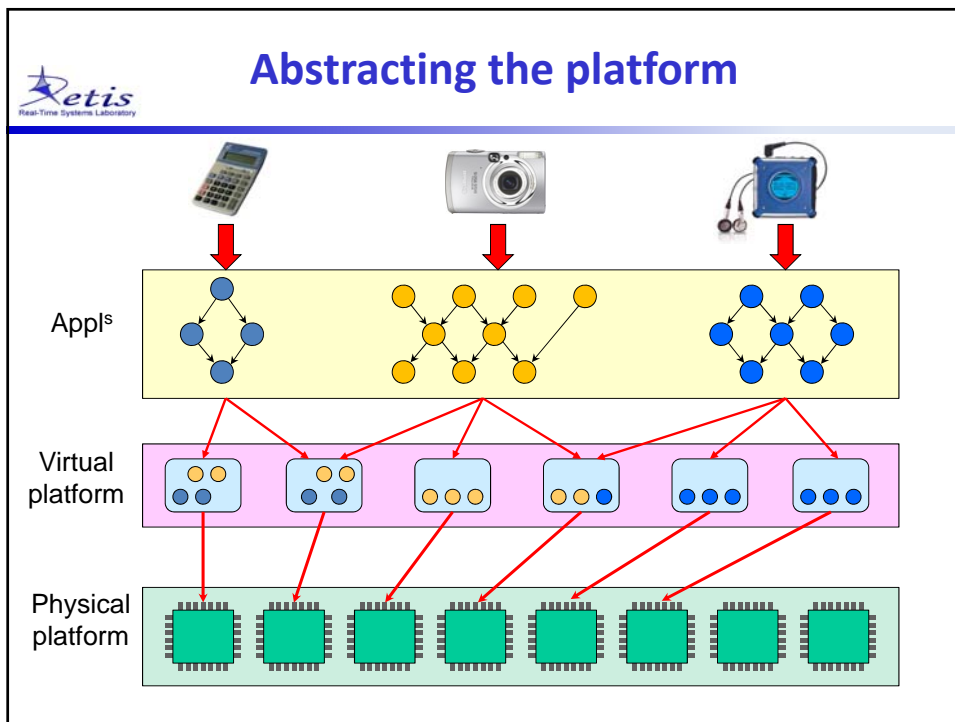
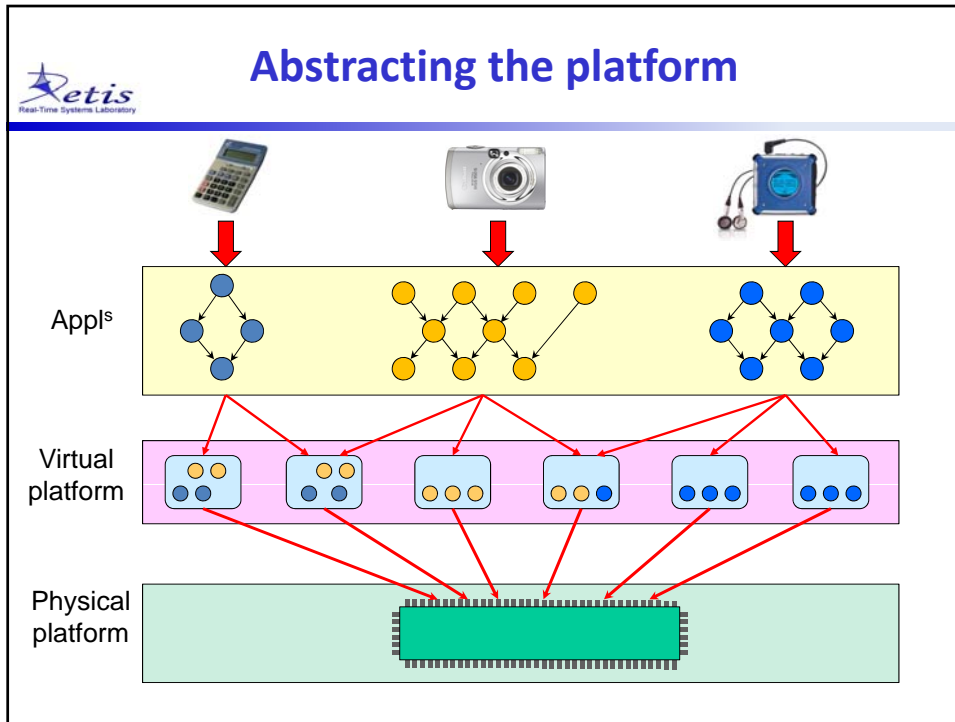


Multicore Reservations

- Hence, a multicore reservation cannot be specified by the overall supplied bandwidth.

A multicore reservation must be specified as a set of uniprocessor reservations

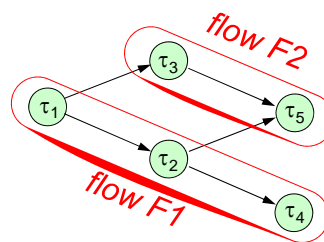




Partitioning

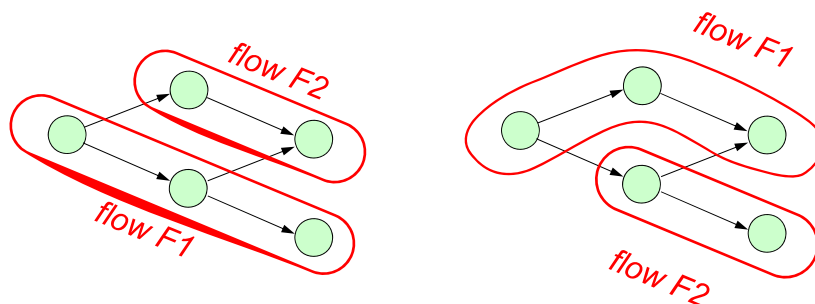
- To partition an application into a set of reservations, we identify a set of **flows**.
- Each flow is a sequential execution to be allocated on a virtual uniprocessor (i.e., a reservation)


For instance:



Selecting the best flows


- Different partitions have different bandwidth consumption
- Which one gives the best results?





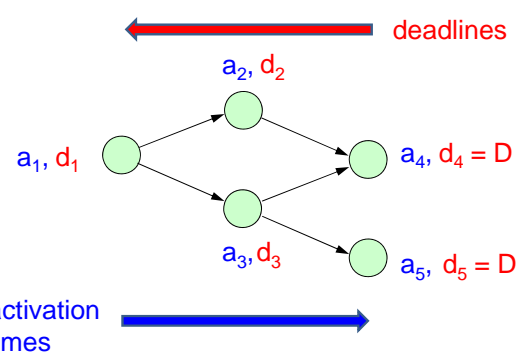
Outline

- How to express parallelism
- How to achieve temporal isolation
- How to perform real-time guarantees
- How to achieve portability by a proper abstraction (the virtual multiprocessor)
- How to optimize resource usage



Dealing with precedence constraints

To compute the bandwidth required by a flow, we have to assign intermediate **activation times** and **deadlines**:





Dealing with precedence constraints

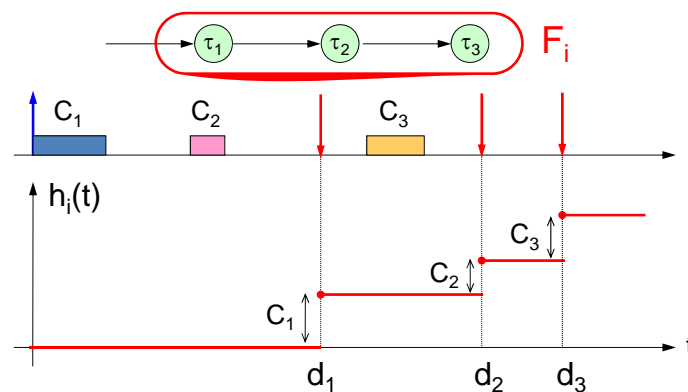
Once **activation times** and **deadlines** are assigned to each task, we can execute them according to EDF, forgetting the precedence relations:


- a_1, d_1
- a_2, d_2
- a_3, d_3
- a_4, d_4
- a_5, d_5



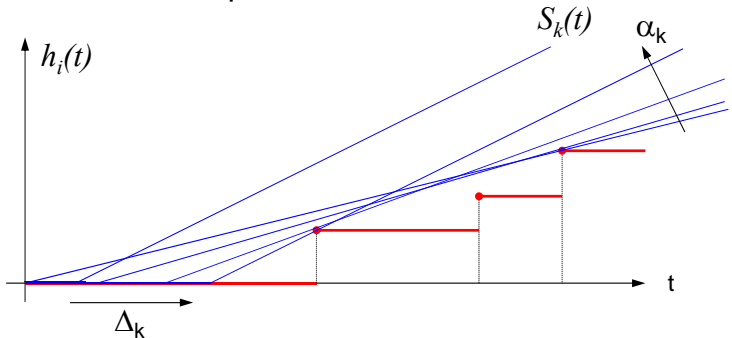
Computational demand of a flow

- Then, the processor demand required by a flow F_i is computed in each interval of time:




 **Real-time guarantee**

- To guarantee flow F_i on VP_k it must be $\forall t \ h_i(t) \leq S_k(t)$
- Several solutions are possible:




The higher the delay, the higher the bandwidth required.

 **Resource Optimization**

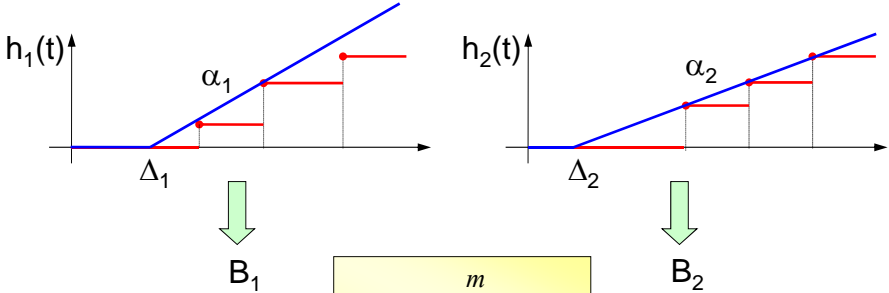
$\left\{ \begin{array}{l} \alpha = Q/P \\ \Delta = 2(P - Q) \end{array} \right.$	Overhead: σ / P
	Actual Bandwidth: $B = \alpha + \sigma/P$

$$B = \alpha + 2\sigma \frac{1 - \alpha}{\Delta}$$


Taking overhead into account, it is possible to compute the (α, Δ) that minimizes B .

 **Optimal bandwidth**

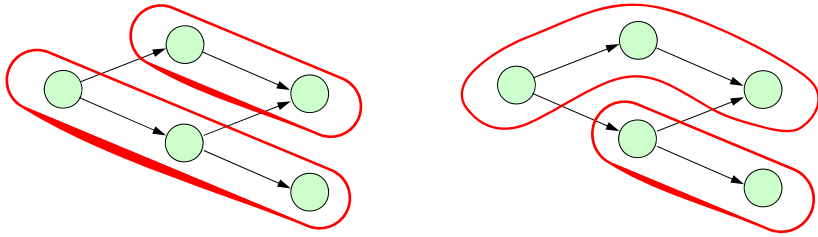
- Once the best (α, Δ) have been computed for each flow, the total bandwidth required by the application is:




$$B = \sum_{k=1}^m B_k$$

 **Selecting the best partition**

- Different partitions require different bandwidth:



How to find the partition that minimizes B?


 **Search algorithm**

Pruning is used to cut

- unfeasible branches ($B_k > 1$)
- redundant branches ($m > M$)

$$M = \left\lceil \delta \frac{C^s}{T} \right\rceil$$

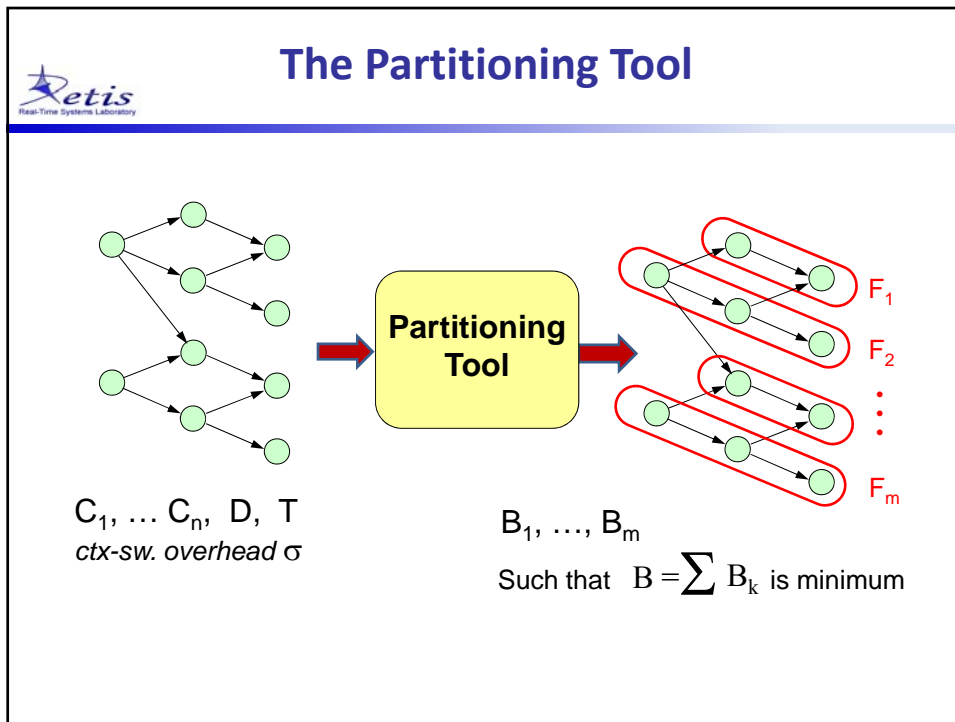
Exponential complexity (tractable for $n < 20$)

 **Heuristics needed**

Although the pruning, the branch and bound algorithm has exponential complexity and the method is not usable with more than 15 tasks.

For this reason,

- heuristic algorithms are needed to partition the applications into flows.
- simple tools are crucial to make such methods applicable in practice.



Partitioning Tool for Multi-core Reservations

If the graph does not show. Please download [svg support plugin](#).
*Browsers natively support svg: Firefox 1.5+, Opera 8.5+, Safari 3.0+, Chrome 1.0+

Precedence Graph

```

    graph LR
      Task0((Task0)) --> Task1((Task1))
      Task0 --> Task3((Task3))
      Task1 --> Task2((Task2))
      Task1 --> Task4((Task4))
      Task3 --> Task4
  
```

Timeline Representation/Parallel Number Function

Task0: [0, 4.0]

Task1: [4.0, 5.0]

Task2: [5.0, 10.0]

Task3: [4.0, 6.0]

Task4: [6.0, 9.0]

Demand Bound Function/Alpha-Delta Server

Flow0: $\alpha=0.89, \Delta=0.29, \text{Cost}=0.93$

Flow1: $\alpha=0.75, \Delta=0.53, \text{Cost}=0.80$

Application Parameters

Arrival time: 0, Period: 12, Deadline: 12, sigma: 0.1, sequentialC: 15, parallelC: 10, max Parallel Number: 2

Configuration

Deadline Assignment Method: Chetto* Chetto

Application

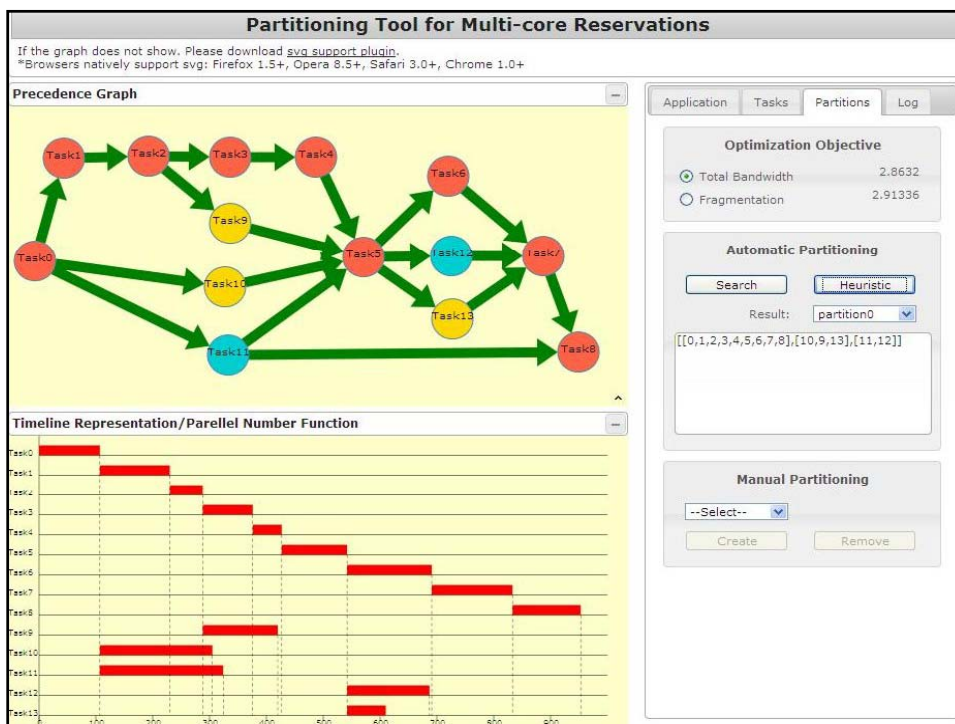
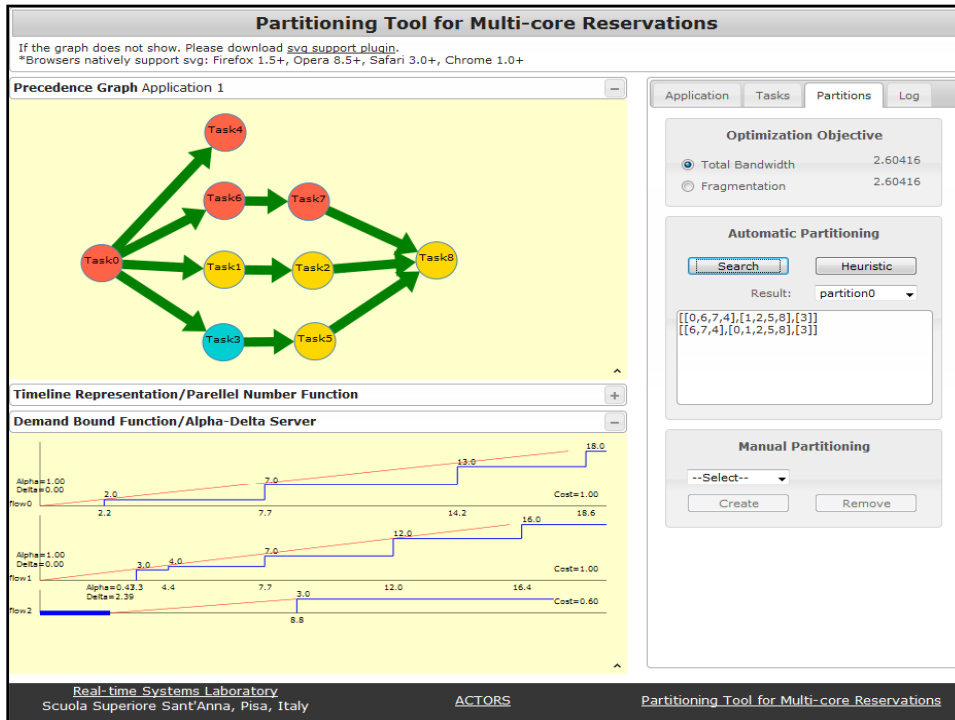
Input Format: 1 2


Buttons: Set Params, Create, Save

Real-time Systems Laboratory
Scuola Superiore Sant'Anna, Pisa, Italy

ACTORS


Partitioning Tool for Multi-core Reservations





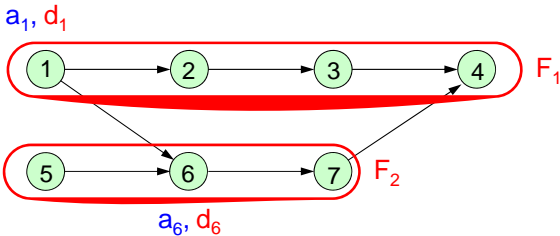
Open Problems

- Communication delays
- Shared Resources
- Adaptivity



Accounting for communication delays


- Delays depend on the allocation on the physical cores.
- They affect task activation times



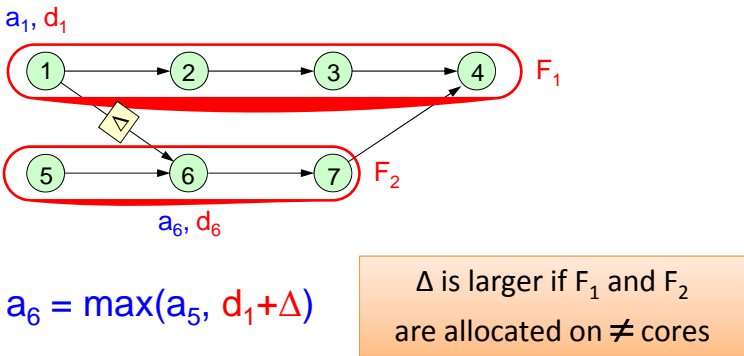
a_1, d_1

a_6, d_6

$$a_6 = \max(a_5, d_1)$$


 **Accounting for communication delays**

- Delays depend on the allocation on the physical cores.
- They affect task activation times

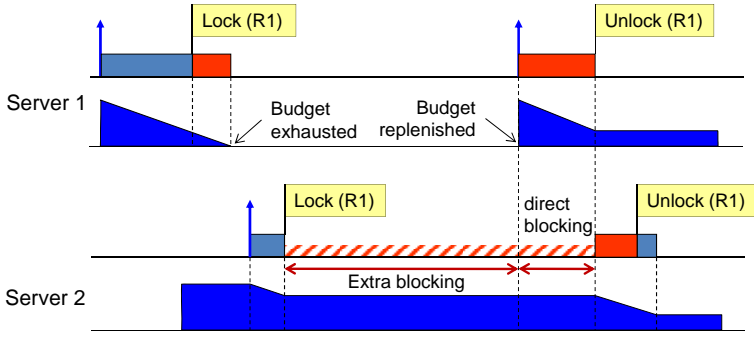



$a_6 = \max(a_5, d_1 + \Delta)$

Δ is larger if F_1 and F_2 are allocated on \neq cores

 **Problems with shared resources**

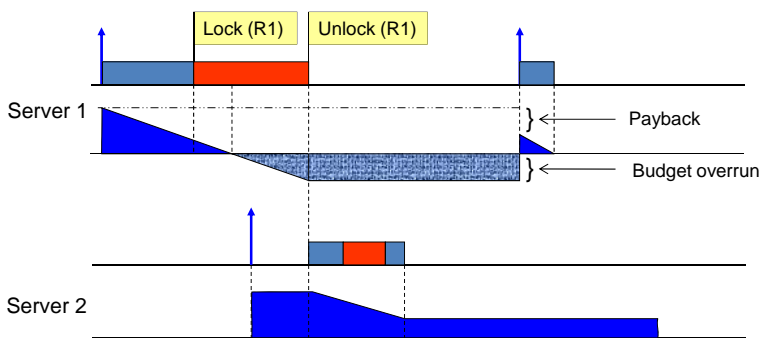
If a server exhausts its budget while locking a shared resource, extra blocking can be introduced in other servers needing the same resource:




 **Possible solution**

Budget overrun (with payback) [Davis & Burns, 2006]

Effective, but it breaks isolation and requires using extra bandwidth.

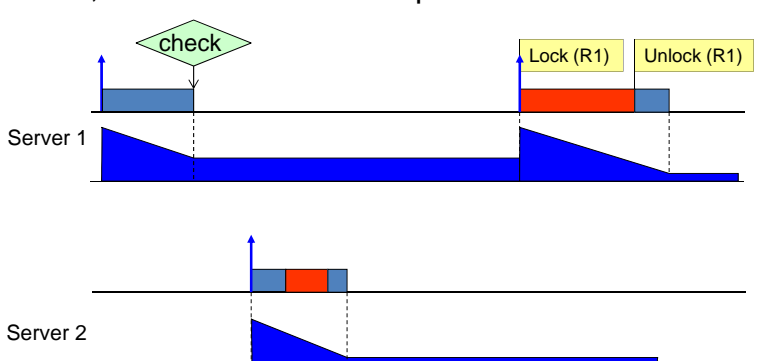


The diagram illustrates the budget management for two servers, Server 1 and Server 2, over time. Server 1's budget is represented by a blue area that starts at a high level, drops during a 'Lock (R1)' period (indicated by a yellow box), and then recovers during an 'Unlock (R1)' period. A shaded area below the budget line indicates a 'Budget overrun' during the lock period, and a bracket labeled 'Payback' shows the time taken for the budget to return to its original level. Server 2's budget is shown as a blue area that is lower during the lock period, indicating that it is not isolated from the budget changes on Server 1.


 **A better solution**

Budget check and wait [Behnam et al. 2007]

Before locking a resource, the budget is checked: if it is not sufficient, the server waits for replenishment ⇒ **SIRAP**

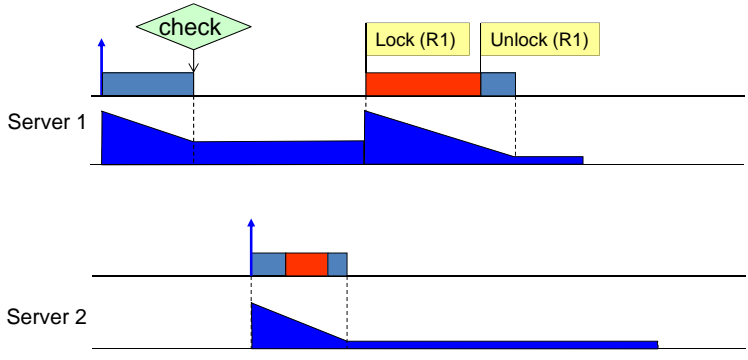


The diagram illustrates the budget management for two servers, Server 1 and Server 2, over time. Server 1's budget is represented by a blue area that starts at a high level, drops during a 'check' period (indicated by a green diamond), and then recovers. A yellow box indicates 'Lock (R1)' and another yellow box indicates 'Unlock (R1)'. Server 2's budget is shown as a blue area that is lower during the lock period, indicating that it is not isolated from the budget changes on Server 1.


 **A better solution**

Budget check and recharge [Bertogna et al., 2009]

Before locking a resource, the budget is checked: if it is not sufficient, the server recharges ASAP \Rightarrow **BROE**

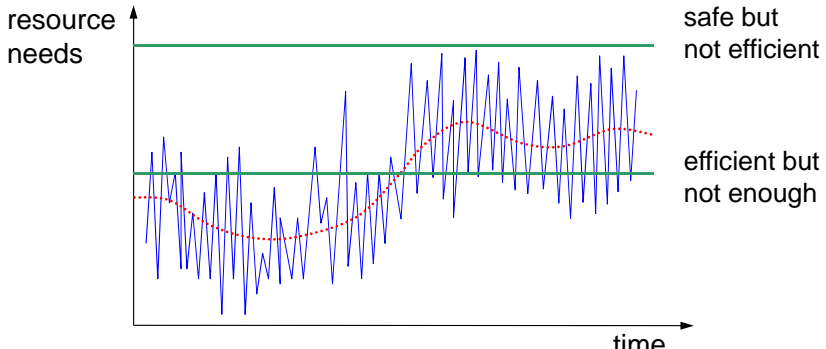


The diagram shows two server timelines. Server 1 starts with a budget (blue area) that decreases over time. A green diamond labeled 'check' points to the budget level. When the budget is low, a blue bar indicates recharging. After recharging, the budget is sufficient to lock resource R1 (red bar). Once R1 is unlocked (yellow box), the budget continues to decrease. Server 2's budget (blue area) is shown below, with its resource usage (red bar) occurring after Server 1's lock.


 **Dynamic applications**

- Some applications have highly variable behavior difficult to predict (e.g. multimedia players, visual tracking)

\Rightarrow any reservation is not appropriate

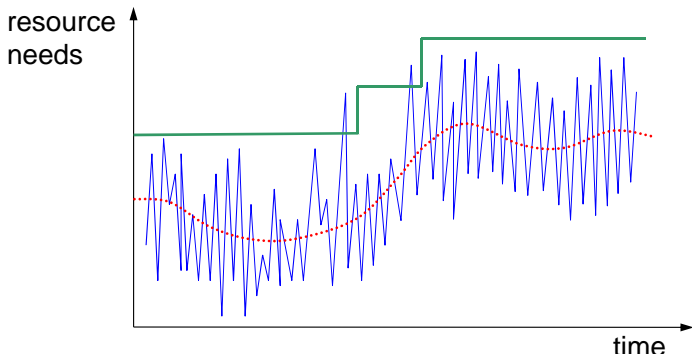



The graph plots resource needs against time. A blue line shows highly variable resource needs. A green horizontal line represents a constant reservation, labeled 'safe but not efficient'. A red dashed line represents a dynamic reservation, labeled 'efficient but not enough'.

 **Need for adaptivity (modes)**

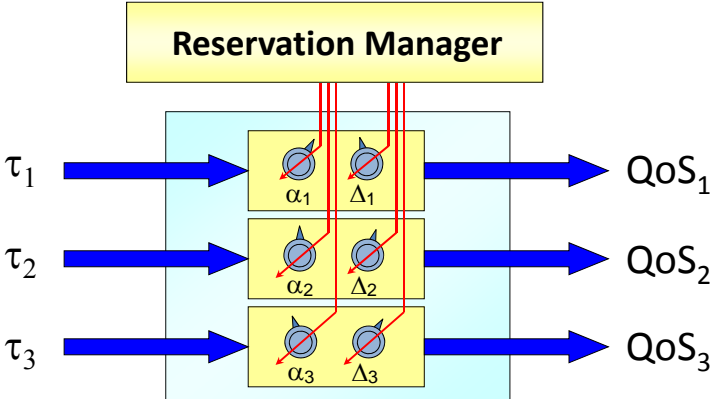
- Reservations should be adapted to the application needs based on runtime requirements:

Off line guarantee \rightarrow $\left\{ \begin{array}{l} \text{Feasible modes} \\ \text{Feasible transitions} \end{array} \right.$



 **Adaptive QoS Management**

- A Reservation Manager must decide how and when changing a reservation to satisfy the application needs, as well as to preserve the other reservations.



Thank you