# Compositional Analysis Techniques for Multiprocessor Real-Time Scheduling

**Hennadiy Leontyev**

**University of North Carolina at Chapel Hill**

**Advisor: Prof. James H. Anderson**

**August 2009**

# Some Self-Promotion ☺

- I am a fifth-year PhD student at UNC-Chapel Hill
- Currently I am actively looking for a job
- Research interests
  - » Theory of multiprocessor soft real-time scheduling
  - » Component-based systems
  - » Analysis tools
- More at http://cs.unc.edu/~leontyev

# Outline

- Motivation/Background
  - » Recent trends in software and hardware development
  - » System models
  - » Research need
  - » Prior work
- My research
  - » More detailed outline will follow
- Research goals
- Concluding remarks

# Motivation

- Complex and distributed embedded systems
  - » CAN, FlexRay
- Proliferation of multiprocessor/multicore platforms
  - » Cost reduction
  - » Smaller energy consumption
- Real-time features in Linux:
  - » High-resolution timers, priority inheritance, short non-preemptive sections, …
- Containers in Linux:
  - » Encapsulate task groups (a little like RT “servers”)
  - » Can have a tree of containers of arbitrary depth
  - » Containers may be created, modified, etc. dynamically

# Motivation

- Verification of timing and performance cha
  - » R
  - » R
  - » Throughput

**Meet timing constraints and use *minimum resources*?**

- QUESTION: What algorithms and analysis tools allow embedded systems with multiprocessor components to be supported *efficiently*

# Background

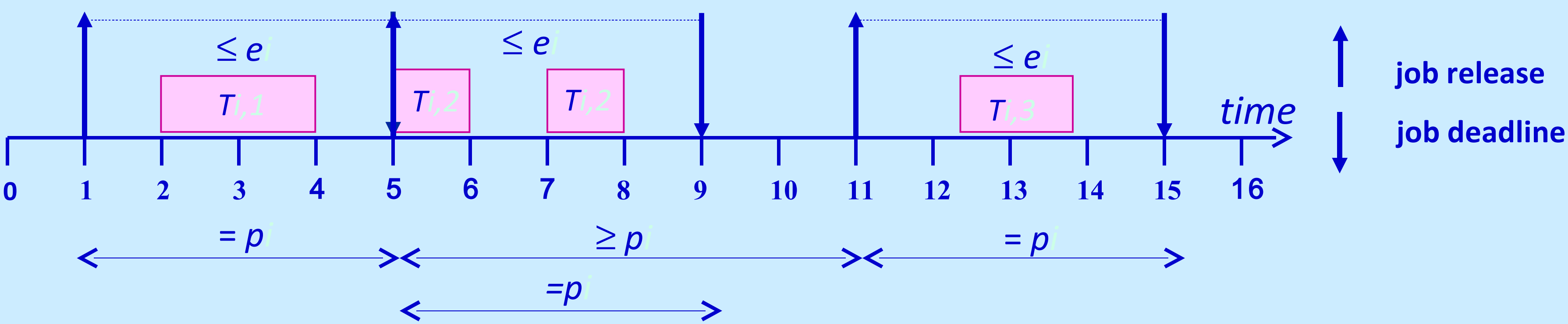
## (Sporadic Tasks)

**Task *Ti* is denoted *Ti* (*ei*,*pi*).**

**Period (min. inter-arrival separation, relative deadline)**

**worst-case execution cost**

**<u>Example:</u> *Ti* (2,4).**

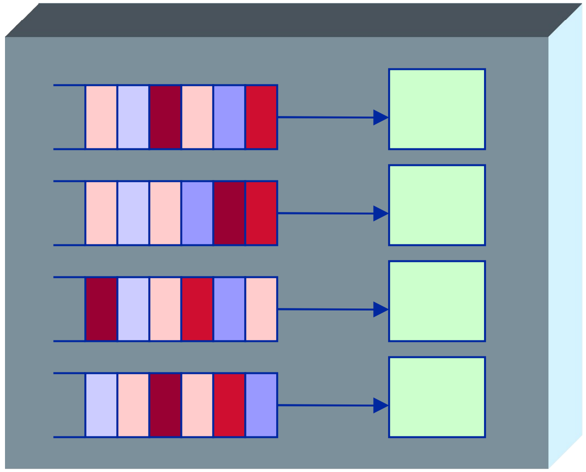



**ui = ei/pi ≡ utilization of Ti (ei ≤ pi & ui ≤ 1).**

# Background

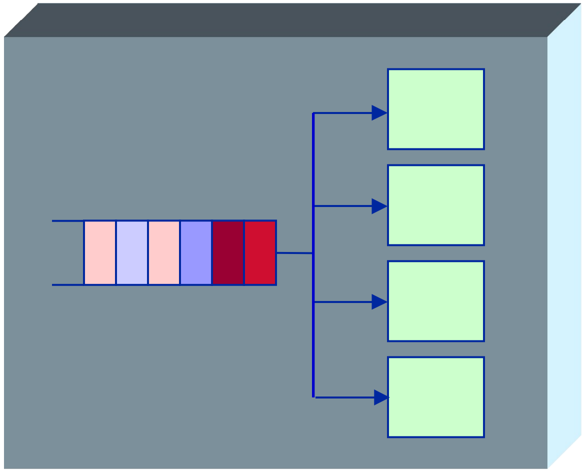
(Multiprocessor Scheduling)

## Partitioning



Statically assign tasks to processors

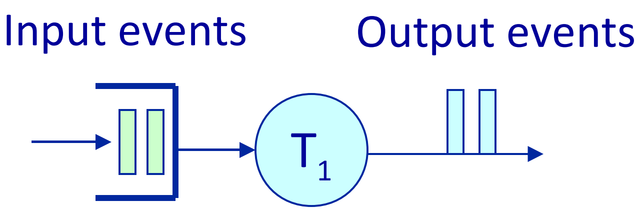
## Global Scheduling



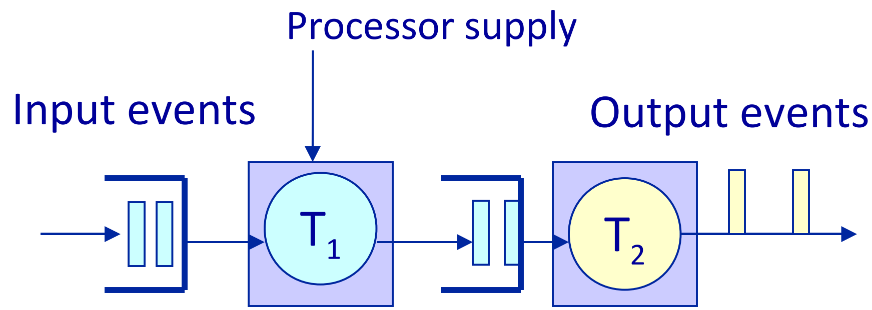
Use a single run queue

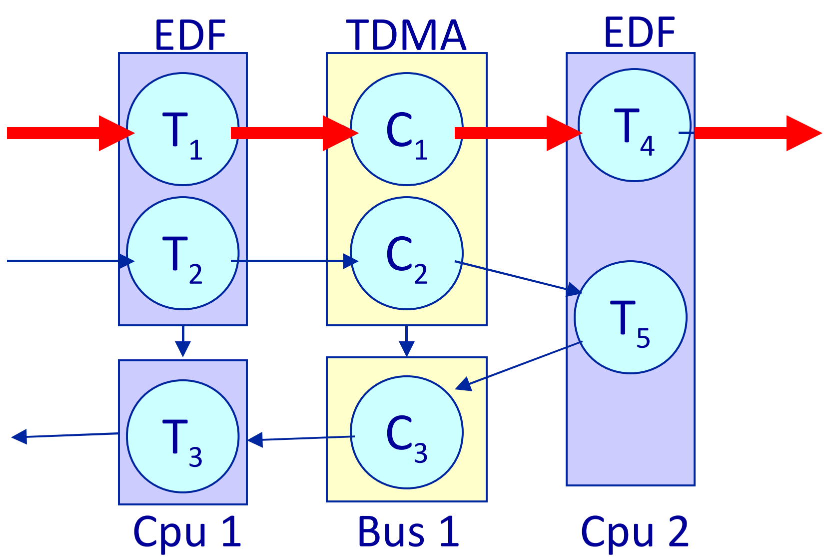
# Background

## (Streaming Task Model)




Streaming Task




# Real-Time Calculus Framework

http://www.mpa.ethz.ch

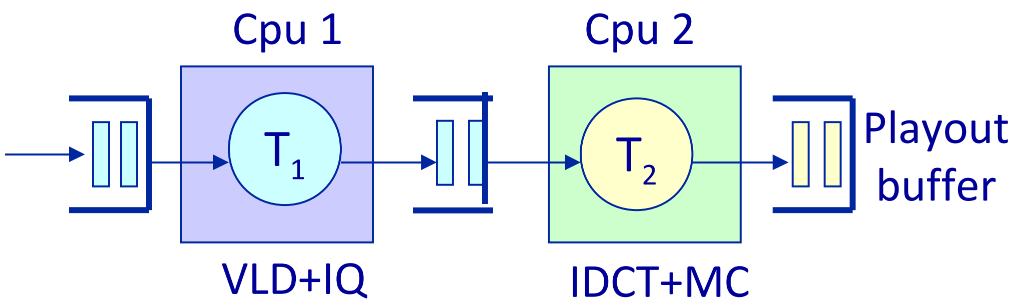
State-of-the-art analysis is for uniprocesor and partitioned systems only!

# Outline

- Motivation/Background
  - » Recent trends in software and hardware development
  - » System models
  - » Research need
  - » Prior work
- My research
  - » More detailed outline will follow
- Research goals
- Concluding Remarks

# Motivation

(MPEG-2 player)



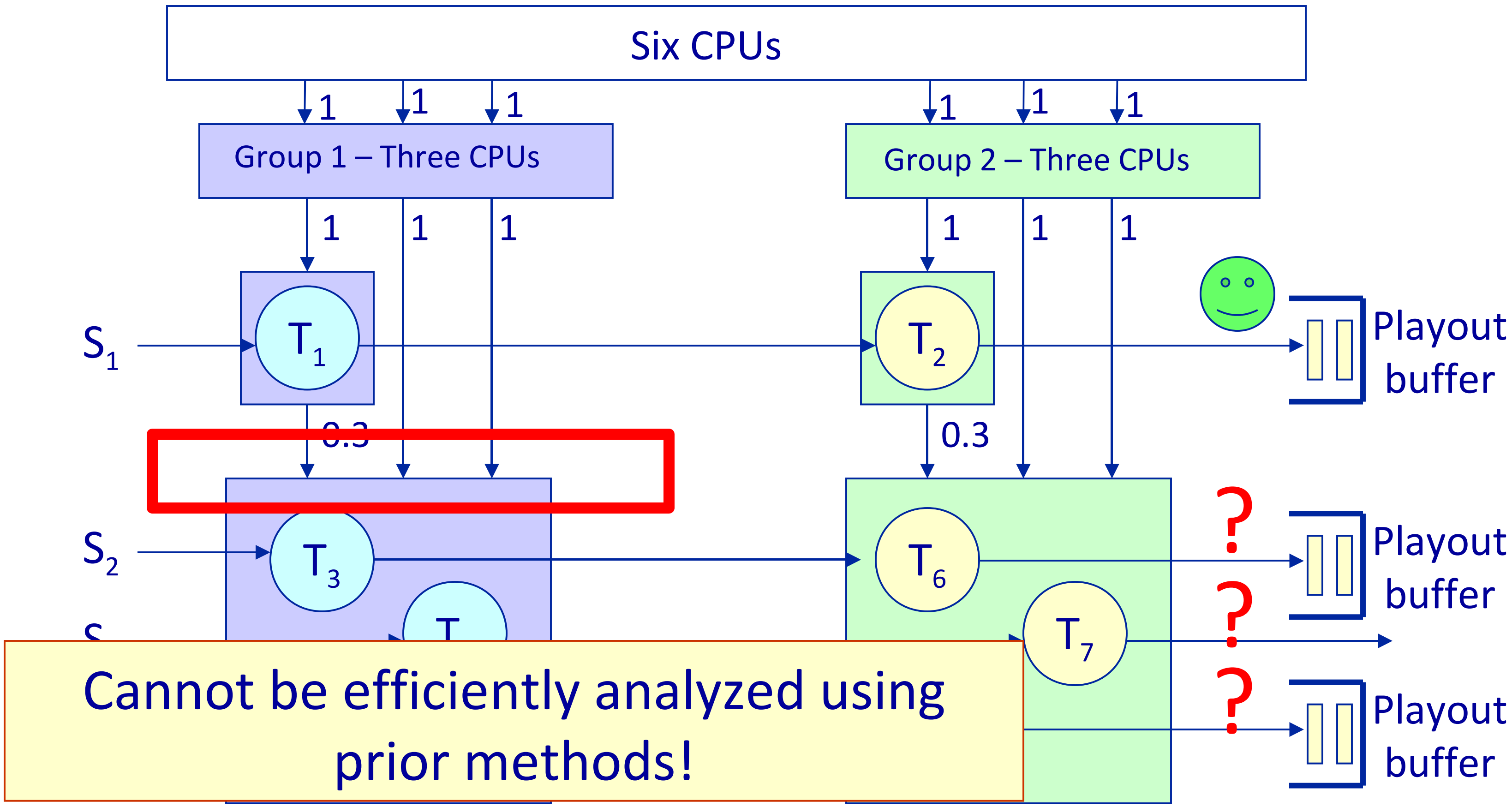

Each task consumes ~0.7 of available processor time (taken from measurements)

- 4 video streams with different criticality
- No two tasks can be placed on one processor
- 8 processors if traditional RTC is used
- Can do better with new multiprocessor analysis!

# Motivation

(Multiprocessor Execution of MPEG-2 player)

8x0.7=5.6 -> 6 processors are probably sufficient

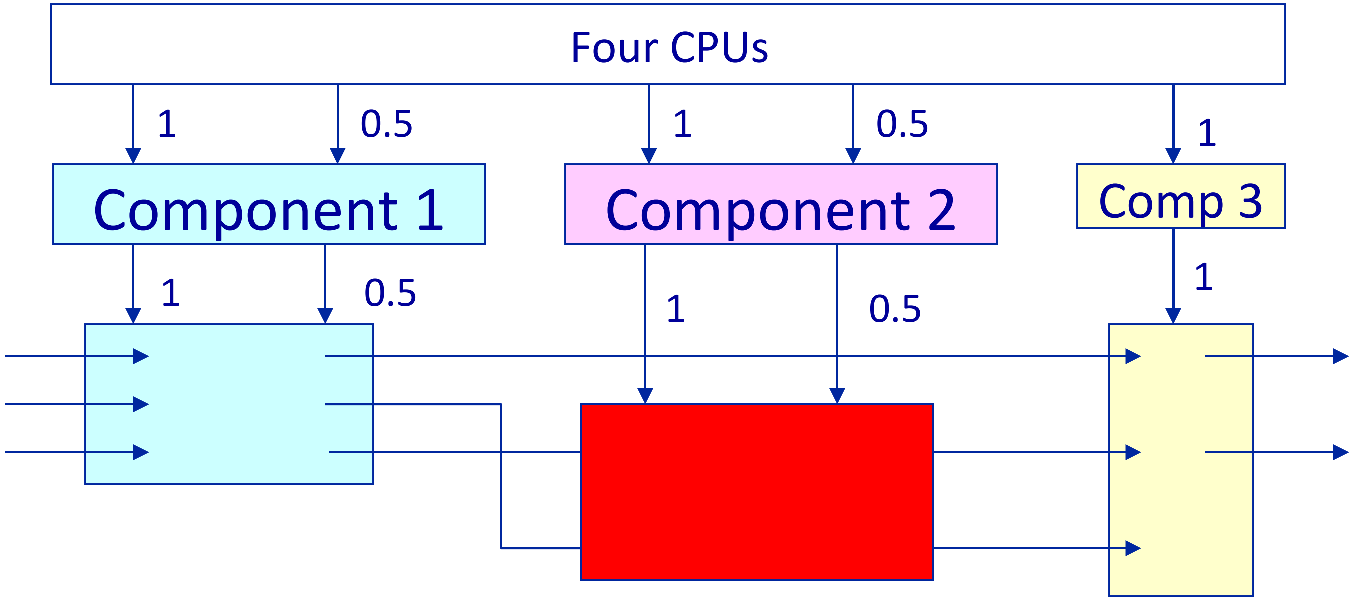


# Motivation

(Multicomponent Systems)

What if there are fractional requirements on supply?

How to isolate misbehaving components?



# Prior Work

| | | | | |
|---|---|---|---|---|
| | | Streaming Task Model | | |
| | | Sporadic Hard | Sporadic Soft | |
| Partitioning | | **Real-Time Calculus Theory** [Chakraborty, Wandeler, Thiele] | | |
| Global | UNR | [Bertogna et. al, Baruah, Fisher] | [Devi & Anderson] | |
| | RESTR | [Shin, Bini, Insup Lee, Eawarsan] | | My work |

Hard – all deadlines are met

Soft – bounded maximum deadline miss (tardiness)

# Motivation

(Directions for Research)

1. Develop a scheme for efficient distribution of multiprocessor capacity among components
   » Understand the behavior of recurring task sets (sporadic tasks) if multiprocessor capacity is restricted
2. Consider more advanced workload models (streaming tasks) under restricted capacity

# Outline

- Motivation/Background
- My research
  - » Distributing processing power among components
    - – Hierarchical bandwidth reservation scheme
  - » Analysis of a single component
    - – Multiprocessor extensions to real-time calculus
- Research goals
- Concluding remarks

# Problem Addressed

- **Given:** A characterization of the processing supply available to a container H.
- **Determine:** How to allocate processing time to its children.
  - If child is another container, must characterize its supply too.
- **Goal:** Would like little or no utilization loss throughout container hierarchy.

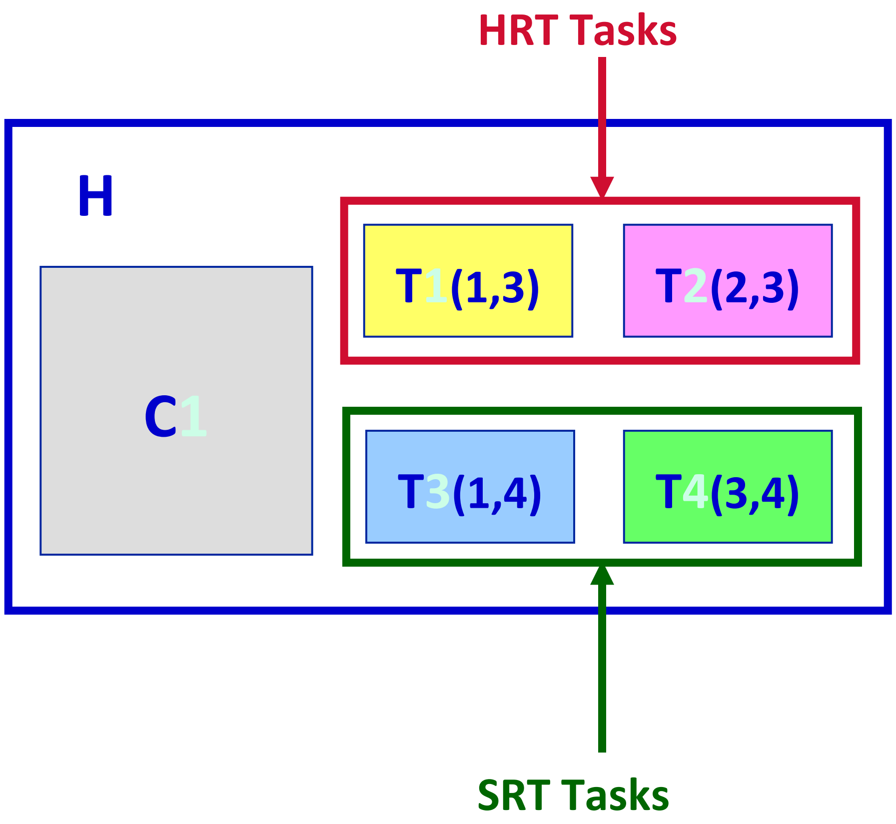
# Problem Addressed

- **Given:** A characterization of the processing supply available to a container H.
- **Determine:** How to allocate processing time to its children.
  - If child is another container, must characterize its supply too.
- **Goal:** Would like little or no utilization loss throughout container hierarchy.

- **Assumptions:**
  - No non-RT tasks.
  - No dynamic changes.
  - Most tasks are SRT (as opposed to HRT).
    - Motivated by focus on Linux and multiprocessors.
  - All (RT) tasks are sporadic with implicit deadlines.

# Problem Addressed

- **Given:** A characterization of the processing supply available to a container H.
- **Determine:** How to allocate processing time to its children.
  - If child is another container, must characterize its supply too.
- **Goal:** Would like little or no utilization loss throughout container hierarchy.



# Background

## SRT Tasks

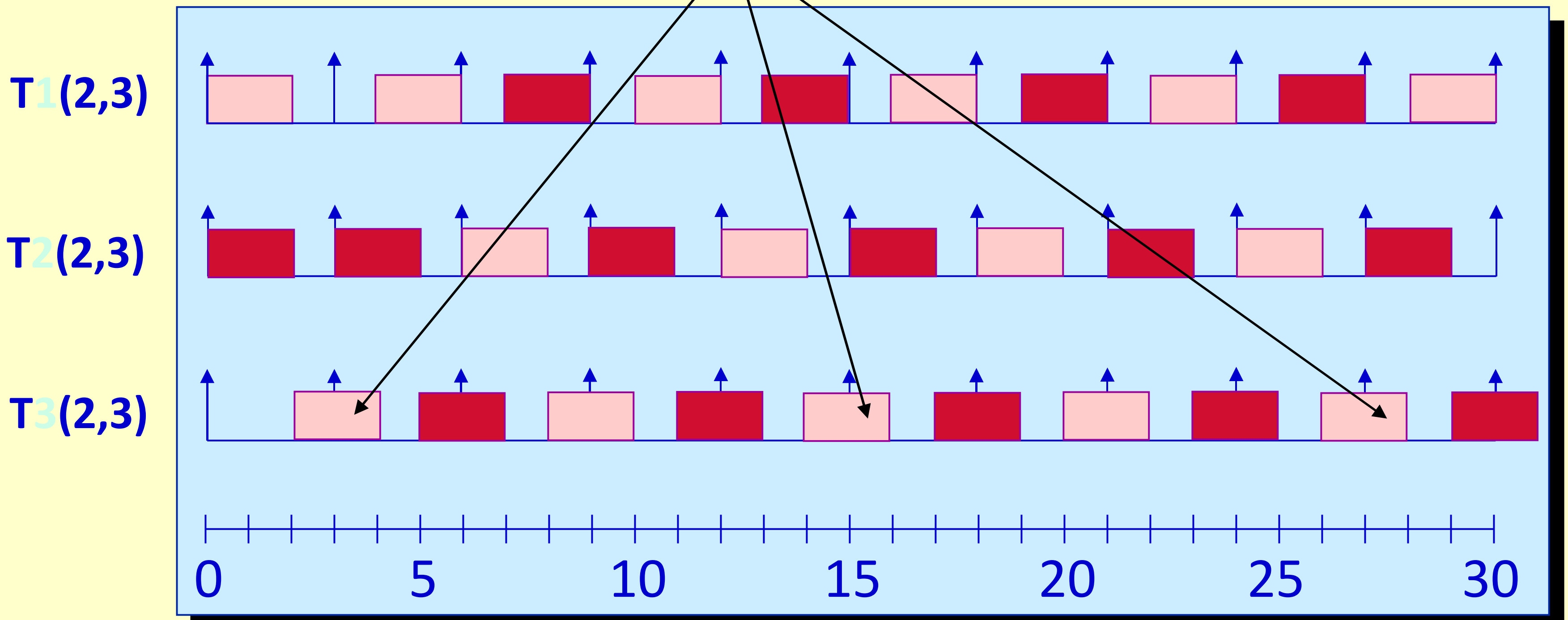
- SRT tasks may miss deadlines, but must have *bounded tardiness*.
- A variety of global scheduling algorithms can ensure bounded tardiness with no utilization loss [Leontyev & Anderson 2007].
  - More on this later…

# Background

## SRT Tasks

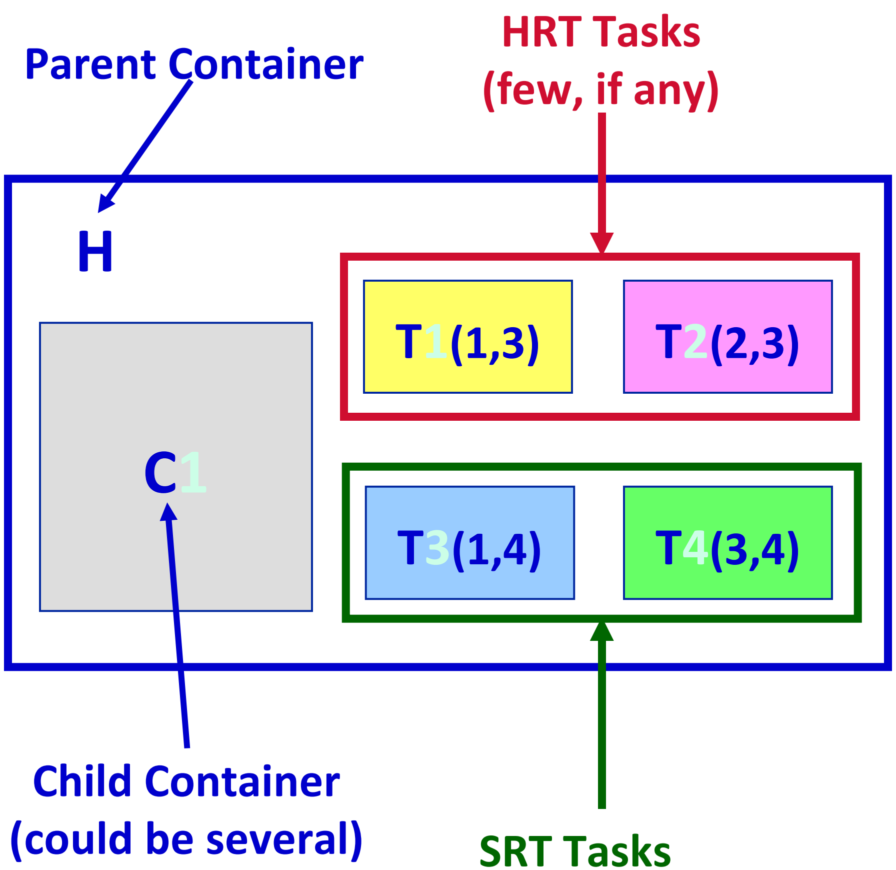
**Example:** Global EDF on two processors.

***Tardiness*** is at most one quatum.



# Background

## Container Model




Container C (parent or child) receives $\sim L \cdot w(C)$ time units over an interval of length L.

w(C) = Bandwidth.

For task Ti, $w(T_i)=u_i=e_i/p_i$.

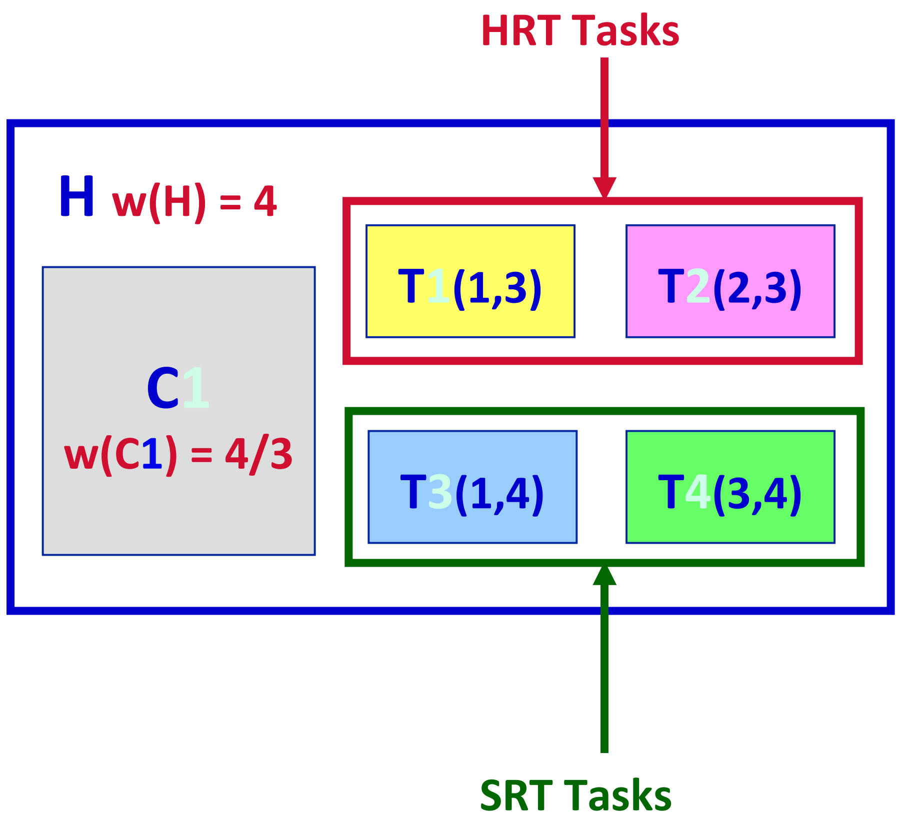
A task always executes on at most one processor

# Outline of Our Approach

- Determine how container supplies should be restricted.
- Given such a supply for the parent, determine how to schedule its children (tasks and containers).
  - We borrow heavily from prior work here.
- Show that supplies for child containers are correctly restricted.

# Running Example

We will use this example to illustrate the approach…



# Container Supply

## Two obvious approaches…

### ~~Maximize parallelism:~~

- ~~May be beneficial if there are a large number of HRT tasks.~~
- ~~Restricts task utilizations.~~
- ~~Difficult to analyze esp. with nesting.~~

### Minimize parallelism:

- Can ensure bounded tardiness w/o utilization restrictions.
  - See paper.
- Lessens tardiness bounds.
  - Again, see paper.
- May be difficult to ensure hard deadlines.

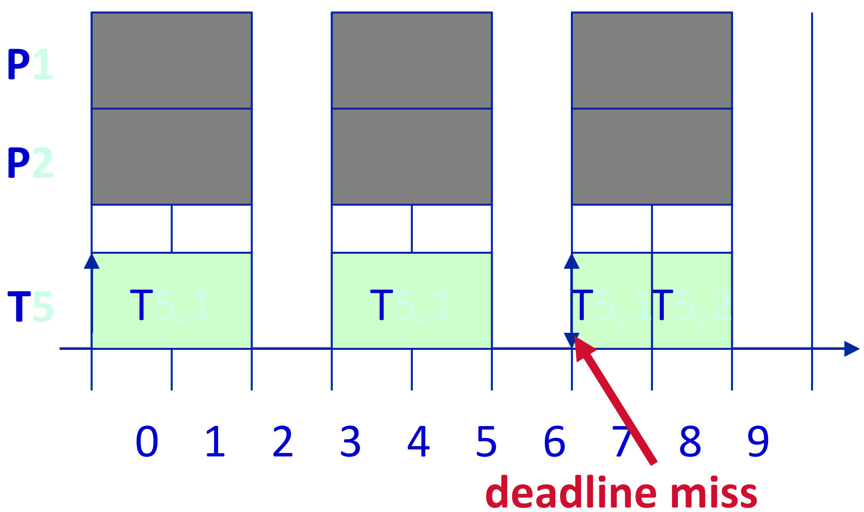
# Container Supply

## Example 1

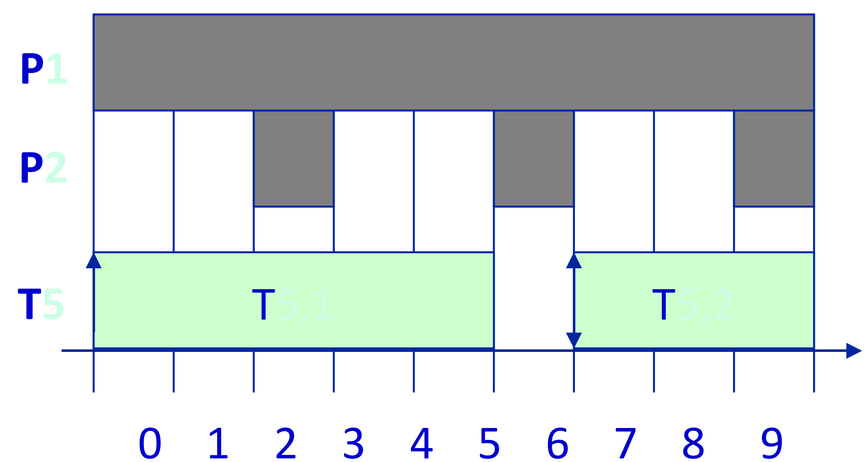
Suppose that C1 with w(C1) = 4/3 in running example has a task T5(5,6)…

Processor allocated to C1

Execution of T5




**<u>Maximum</u> Parallelism**

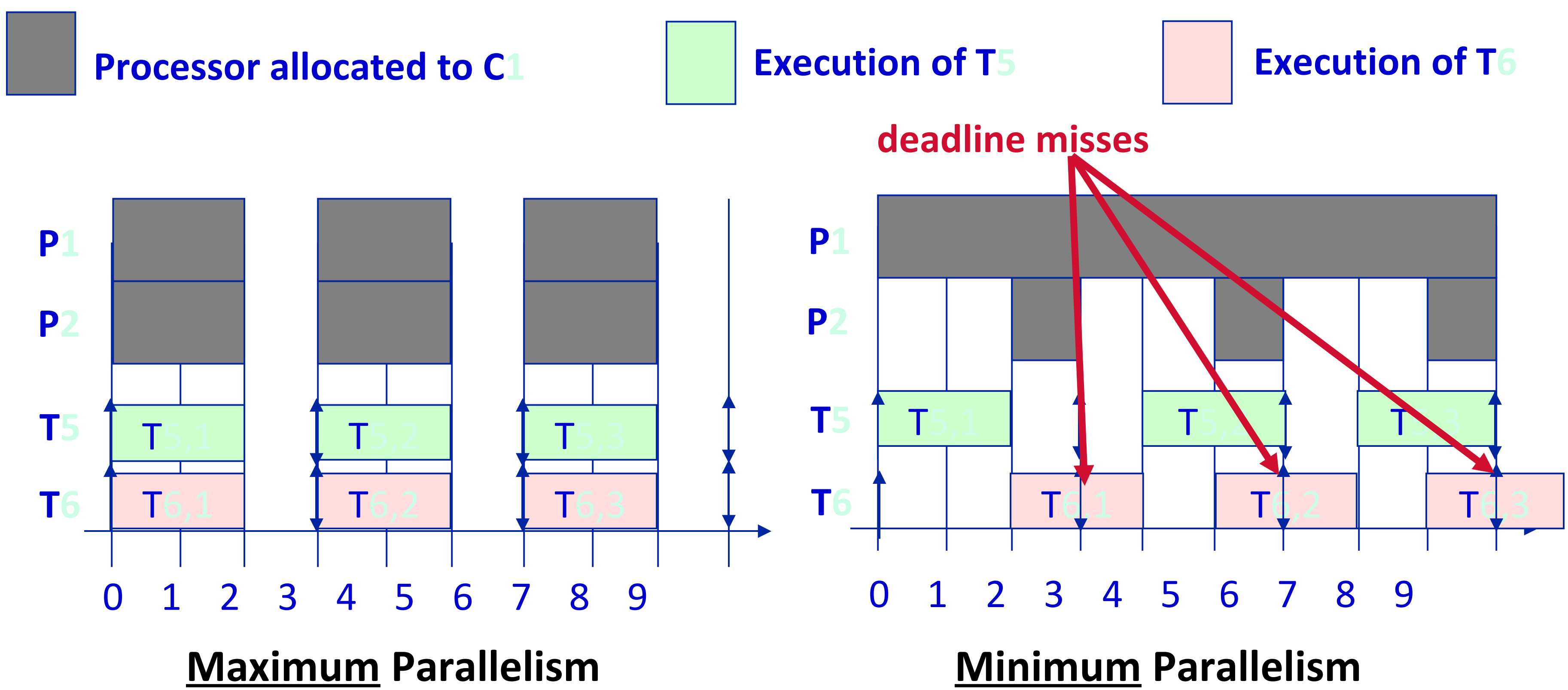



**<u>Minimum</u> Parallelism**

# Container Supply

## Example 2

Now suppose that C1 contains two tasks, T5(2,3) and T6(2,3)…



# Container Supply

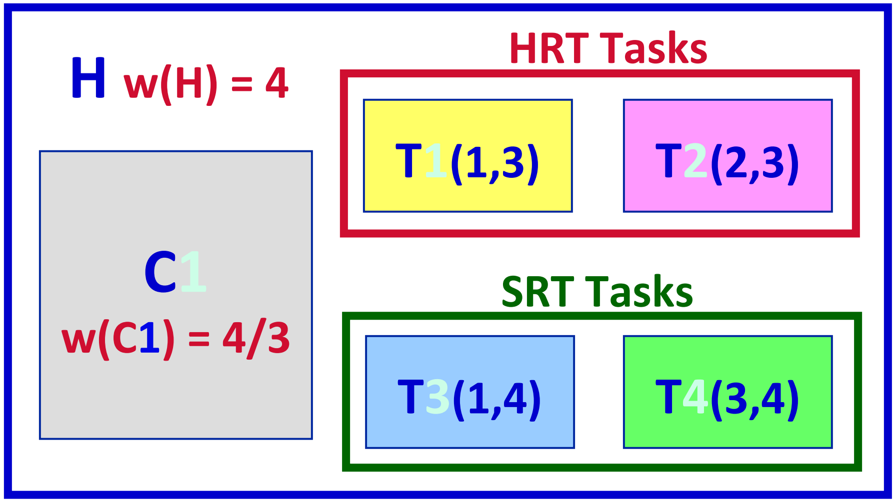
## MinPar Supply

- We require the supply for container C to satisfy the **MinPar Rule**:

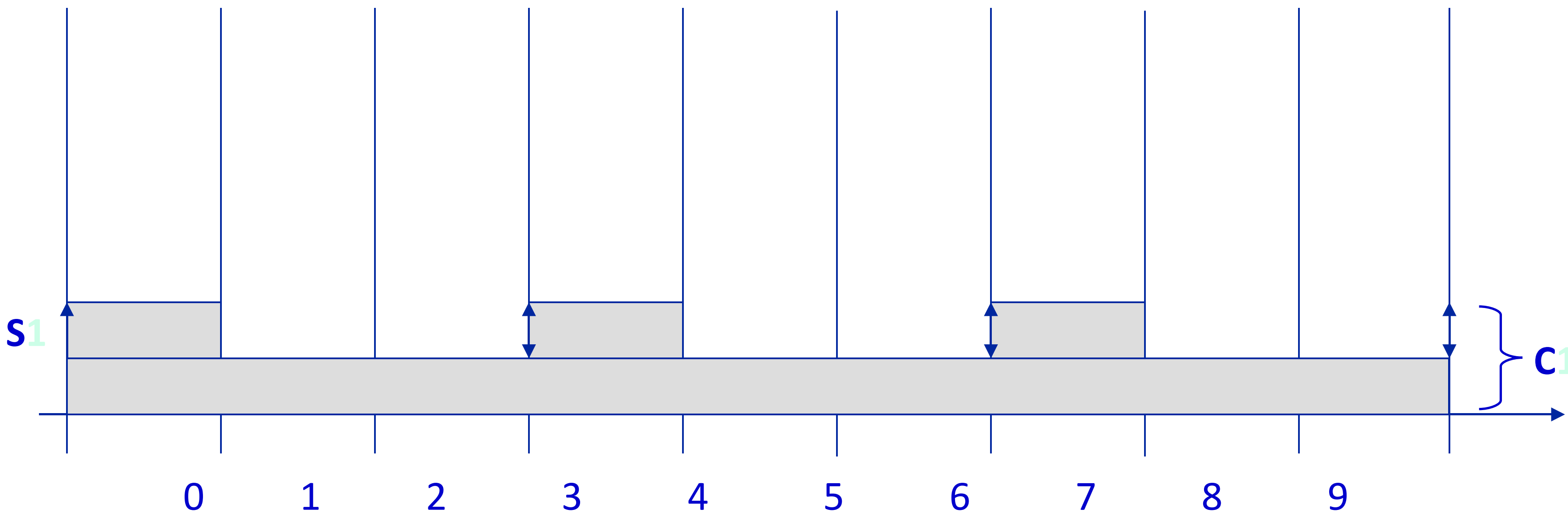
  > » C gets $\lfloor w(C) \rfloor$ dedicated processors plus an additional processor that is allocated at a rate of $f(C) = w(C) - \lfloor w(C) \rfloor$ (= fractional part of its bandwidth).

- If MinPar holds for parent container, it can easily be ensured for any child container:
  - » Create a fictional **"server"** sporadic task of util. $f(C)$ to supply the fractional part.

# Running Example




- 4 processors for H.
- 1 reserved processor for C1.
- Server task S1(1,3) for fractional part of C1.



# Remaining Sub-Problems

- We view server tasks as SRT.
  - » SRT tasks don’t require utilization constraints.
- Thus, there are two remaining problems:
  - » Scheduling HRT tasks.
  - » Scheduling SRT tasks (which may be either “real” tasks or server tasks).

# Scheduling HRT Tasks

- Given our assumption that there are few (if any) such tasks, we use a *very simple* approach:

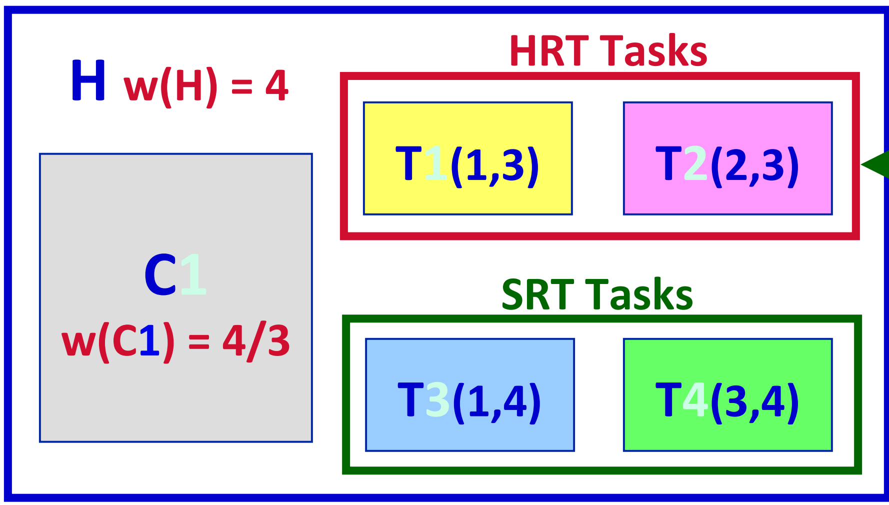
  » Assign HRT tasks to a new child container.

  » Schedule them within that container using partitioned EDF (PEDF).

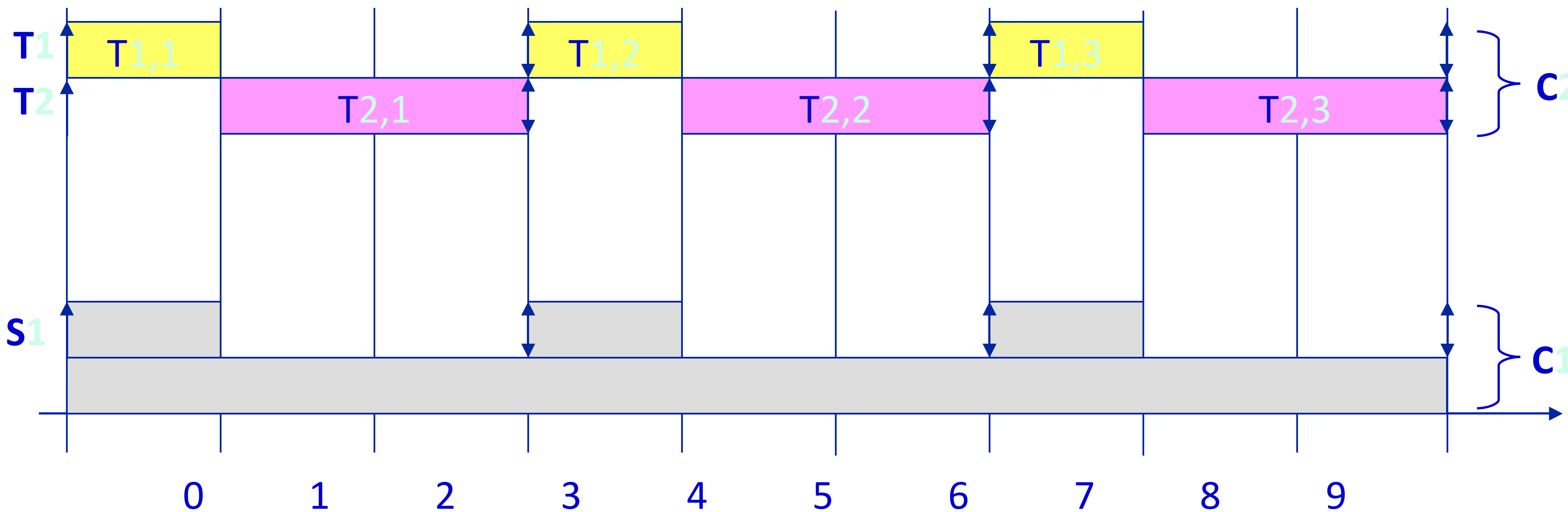
- Notes:
  » Some (small) utilization loss may result.

  » Other approaches are possible.

# Running Example




This is viewed as a new child container C2. It schedules its tasks using PEDF (in this case, on only one processor).



# Last Remaining Sub-Problem

- Need to determine how to schedule all SRT tasks.
  - » Such a task may either be a "real" task or a server task.
  - » Given the MinPar Rule and the design decisions so far, these tasks will be scheduled on $X \leq \lfloor w(H) \rfloor$ dedicated processors and at most one additional partially-available processor.
  - » **Our Goal:** Ensure bounded tardiness for these tasks.

# Last Remaining Sub-Problem

- Need to determine how to schedule all SRT tasks.
  - » Such a task may either be a "real" task or a server task.
  - » Given the MinPar Rule and the design decisions so far, … ed
  - »

This goal can be met using any window-constrained global algorithm [Leontyev & Anderson 2007].

# Window-Constrained Priorities

$$r(T_{i,j}) - \varphi_i \leq \chi(T_{i,j}, t) \leq d(T_{i,j}) + \psi_i$$

release time of job $T_{i,j}$

priority of job $T_{i,j}$ at time t

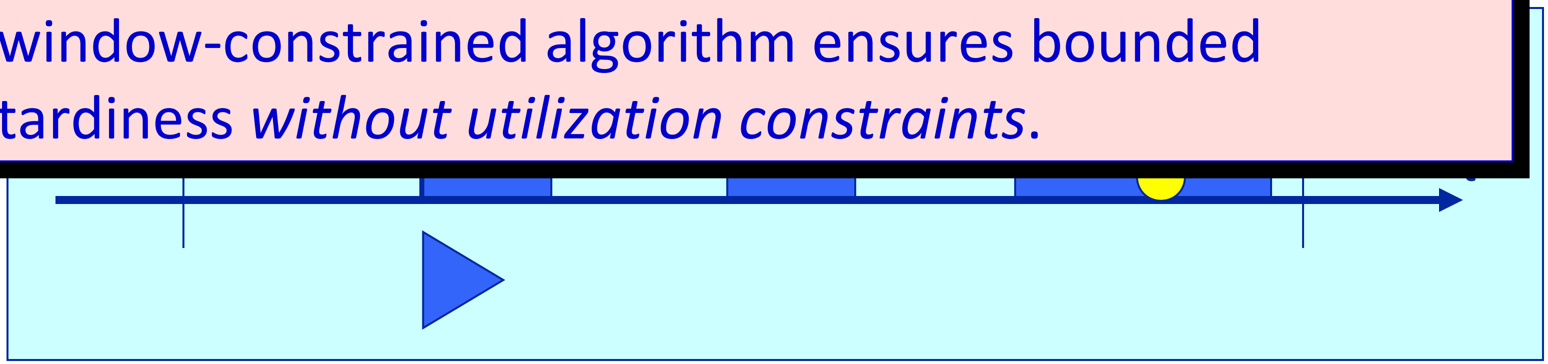
deadline of job $T_{i,j}$

two constants

# Window-Constrained Priorities

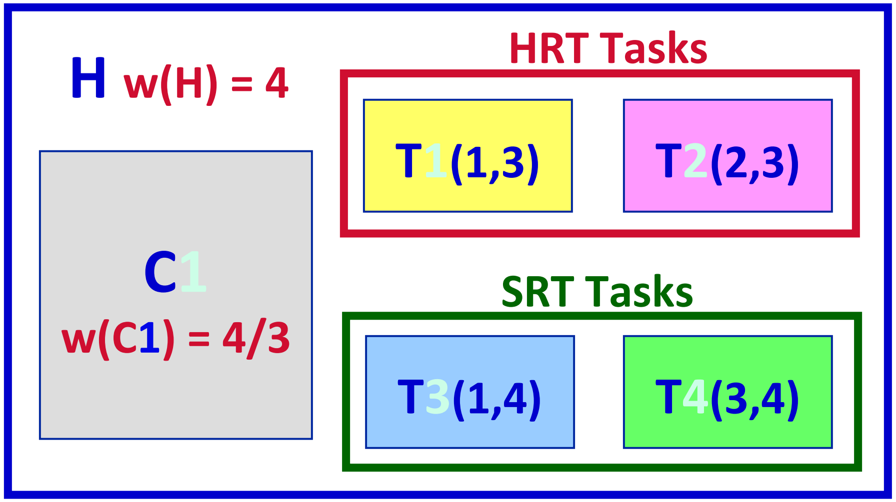
$$r(T) - \varphi \le \chi(T, t) \le d(T) + \psi$$

**Theorem [Leontyev & Anderson 2007]:** If processing time is supplied according to the MinPar Rule, then any window-constrained algorithm ensures bounded tardiness *without utilization constraints*.

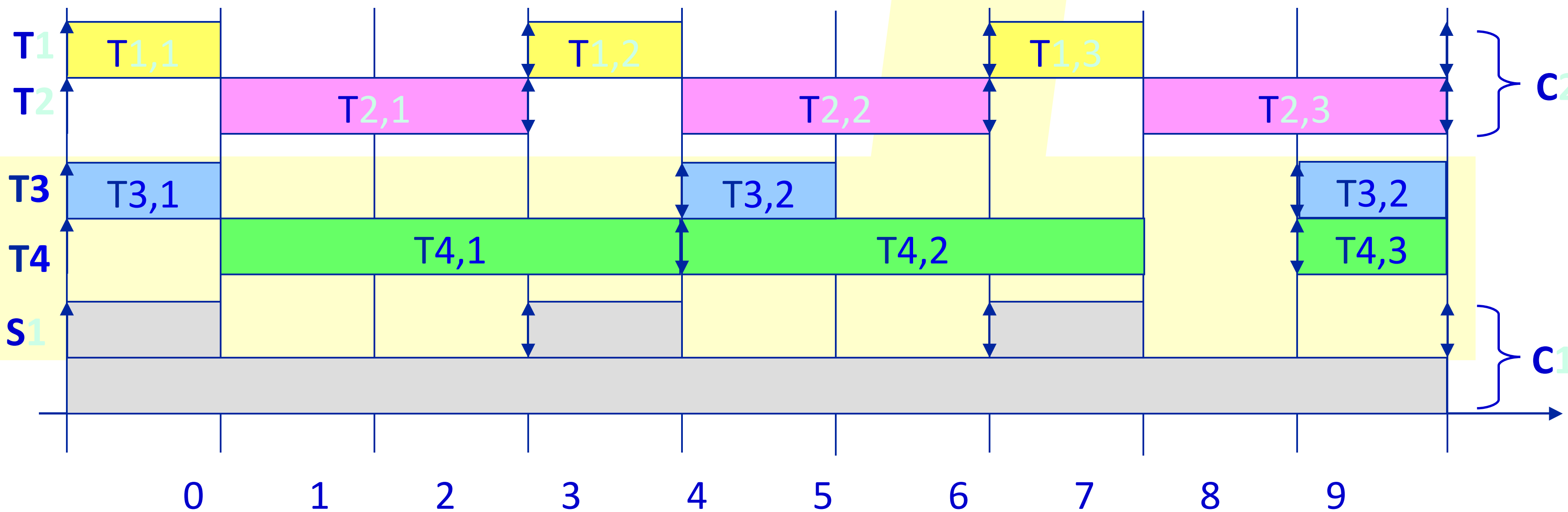


GEDF, FIFO, Pfair, EPDF, LLF, EDZL
are all window-constrained.

# Running Example




These tasks are scheduled on two processors using GEDF.



# Computing Next-Level Supply

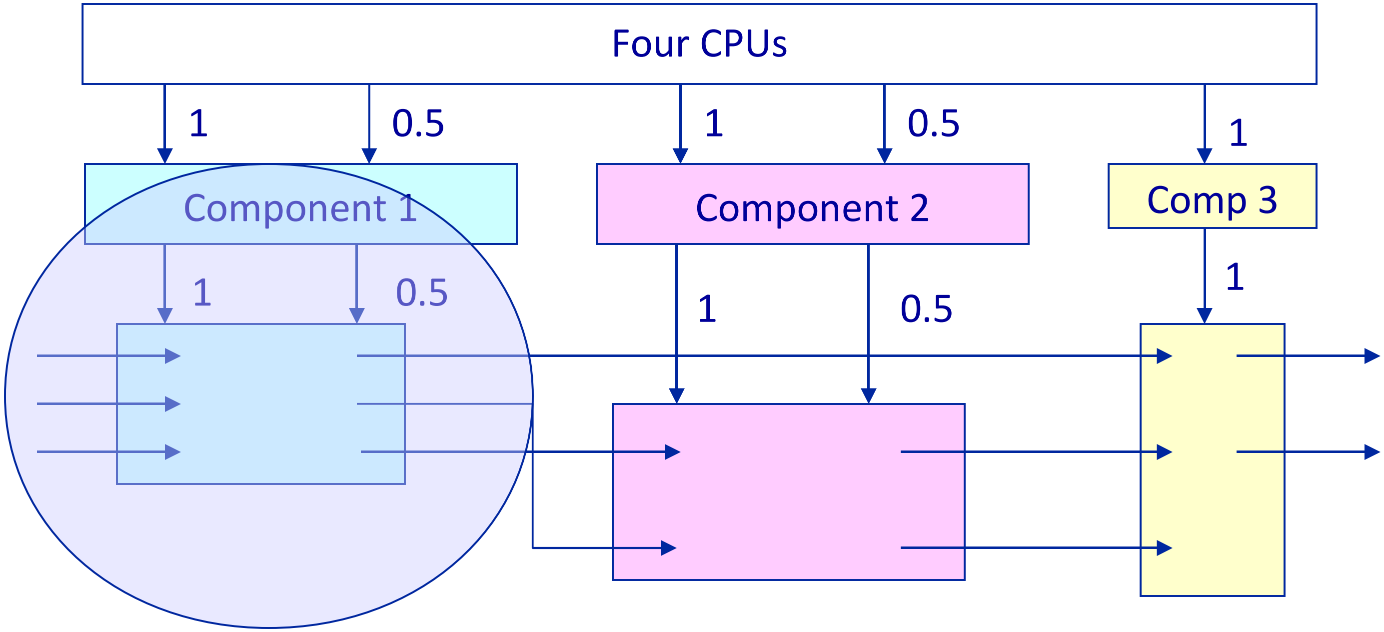
- This is pretty easy:
  - » Each child container is allocated:
    - – some set S of fully-available processors;
    - – at most on partially-available processor P.
    - – The allocation rate of P can be formally characterized.
      - This is based on corresponding server task's execution cost, period, and tardiness bound.
      - See the paper
        - » [Real-Time Systems Journal ECRTS'08 special issue].

# Hierarchical Scheduling Summary

- Scalable multiprocessor hierarchical scheduling scheme
  - » Theoretically unlimited container tree depth
  - » Bounded job response times
  - » No utilization loss in fully SRT case
- Relevance to embedded systems
  - » Distribute the processing power of a multiprocessor among multiple components

# Motivation

## (Multicomponent Systems)

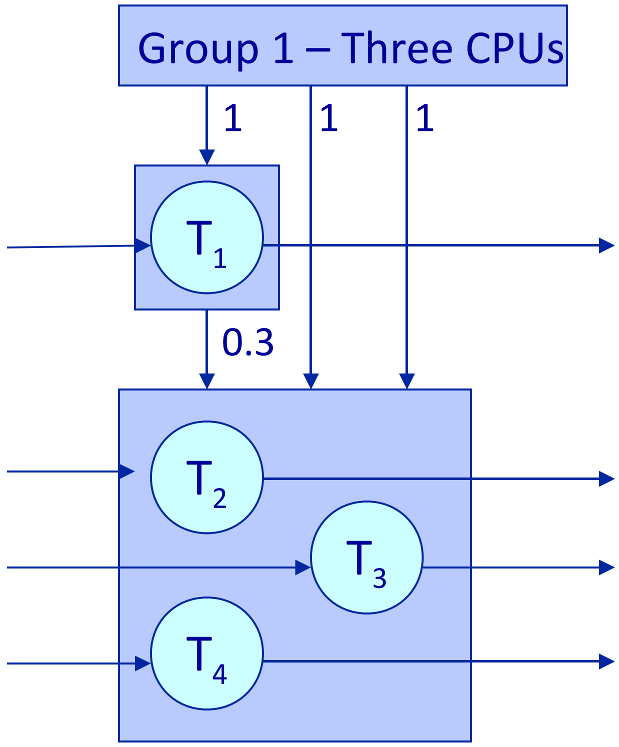


# Outline

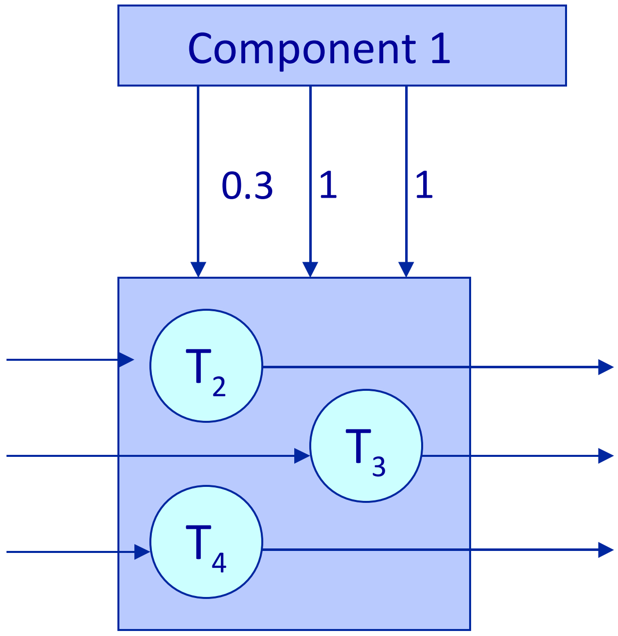
- Motivation/Background
- My research
  - » Distributing processing power among components
    - – Hierarchical bandwidth reservation scheme
  - » Analysis of a single component
    - – Multiprocessor extensions to real-time calculus (joint work with Prof. Samarjit Chakraborty)
- Research goals
- Concluding remarks

# Multiprocessor RTC
(System Model)

## MPEG-2 Example



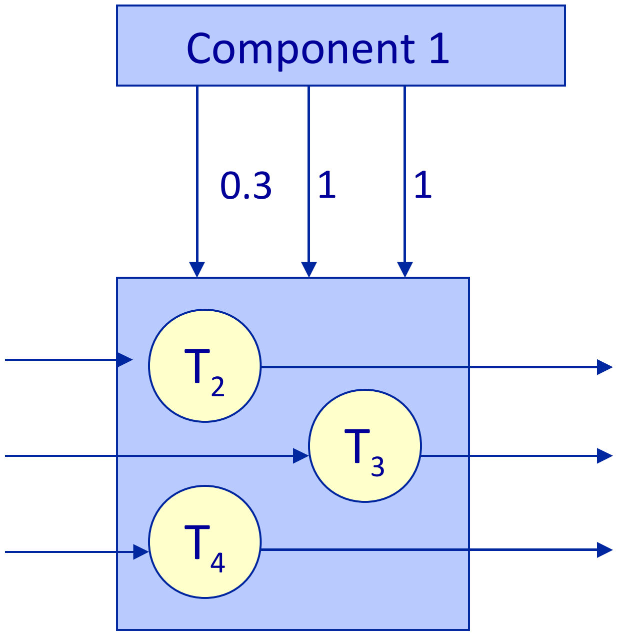

## Abstraction I



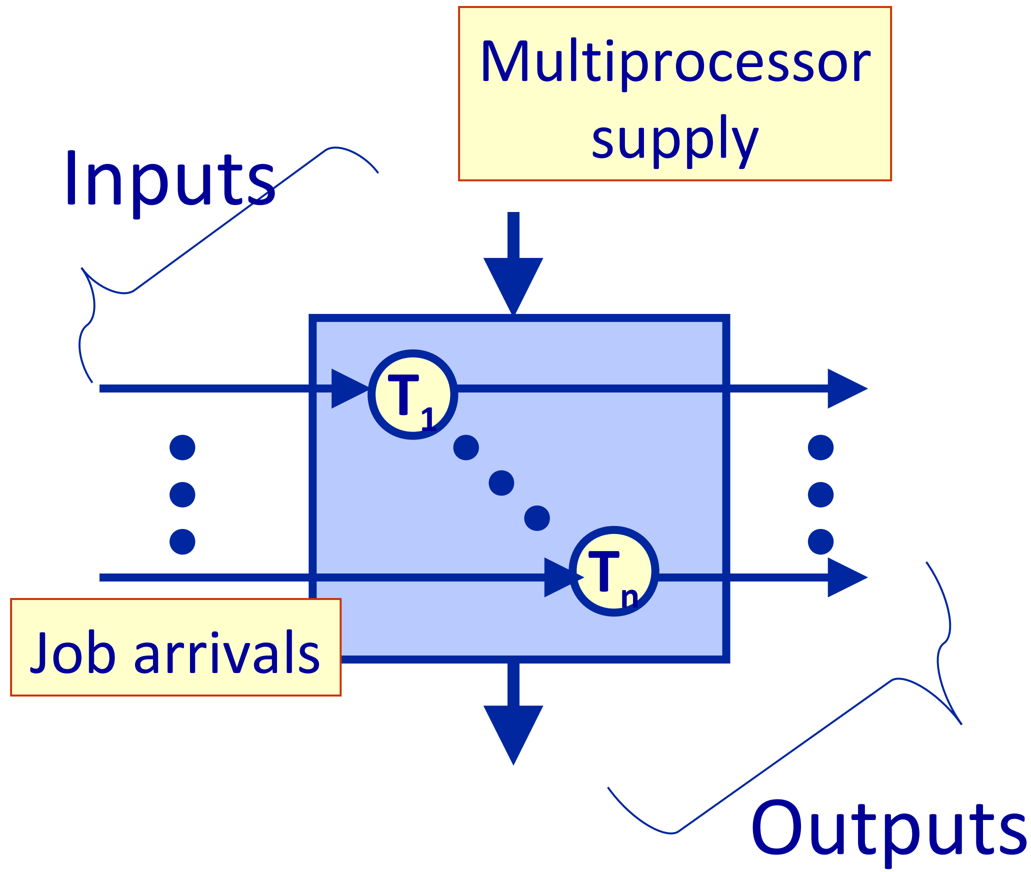
# Multiprocessor RTC

(System Model)

## Abstraction Step I




## Abstraction Step II



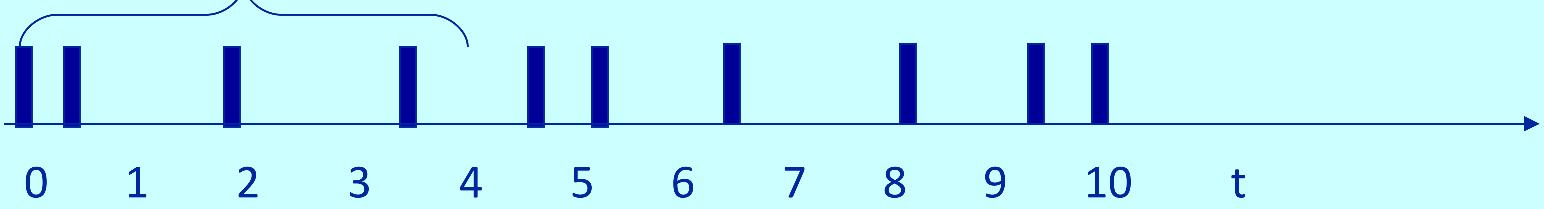
# Multiprocessor RTC

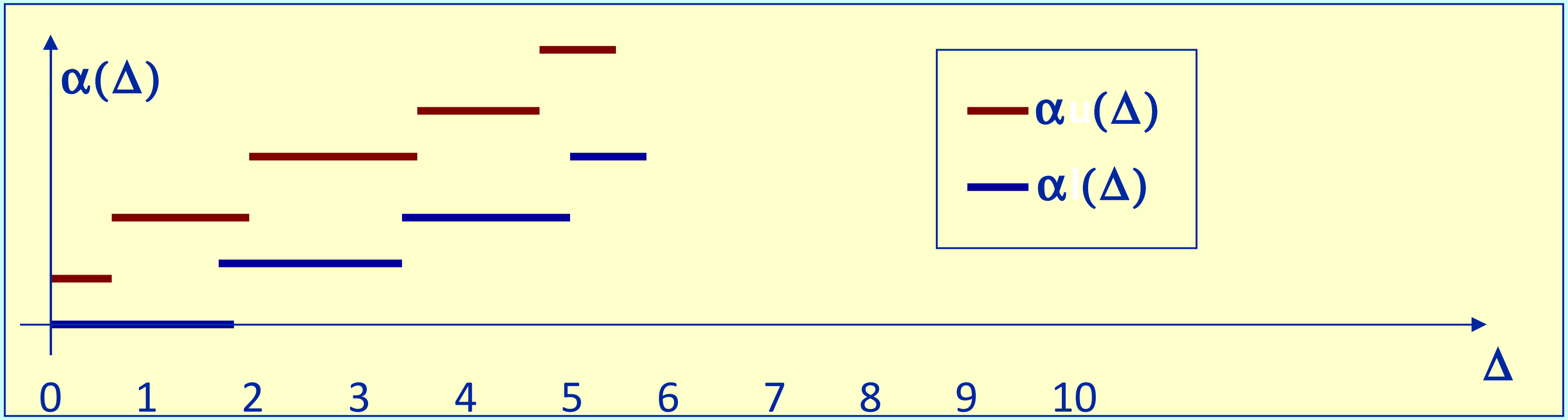
(Job Arrival Functions [Wandeler])

$\alpha^u(\Delta)$ – an upper bound on the number of arrivals for any interval of length $\Delta$
$\alpha^l(\Delta)$ – an lower bound on the number of arrivals for any interval of length $\Delta$

R(t) – the number of jobs in [0,t)

$$\forall s < t : \alpha^l(t-s) \le R(t) - R(s) \le \alpha^u(t-s)$$



# Multiprocessor RTC

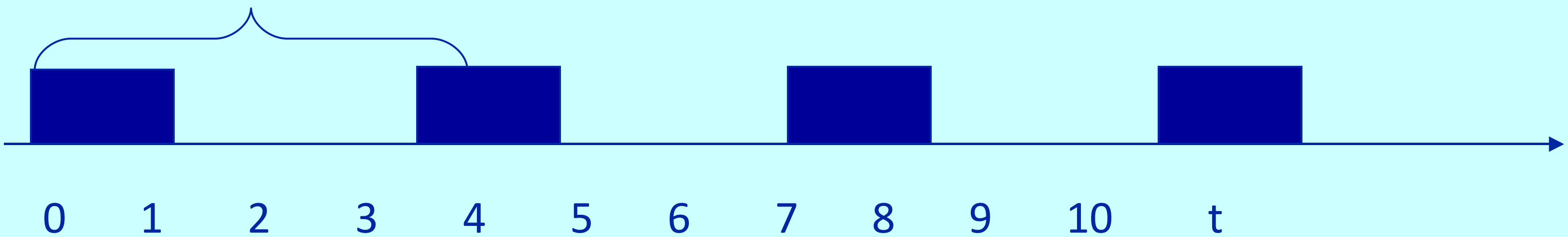
## (Supply Functions)

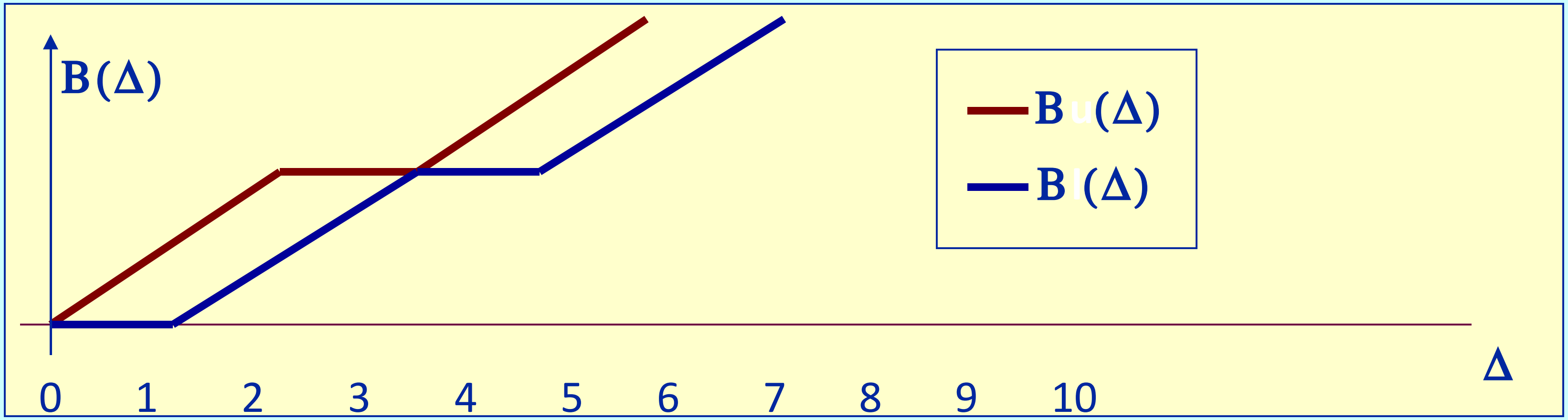
$B^u(\Delta)$ – an upper bound on the available time for any interval of length $\Delta$

$B^l(\Delta)$ – a lower bound on the available time for any interval of length $\Delta$

R(t) – the available time in [0,t)

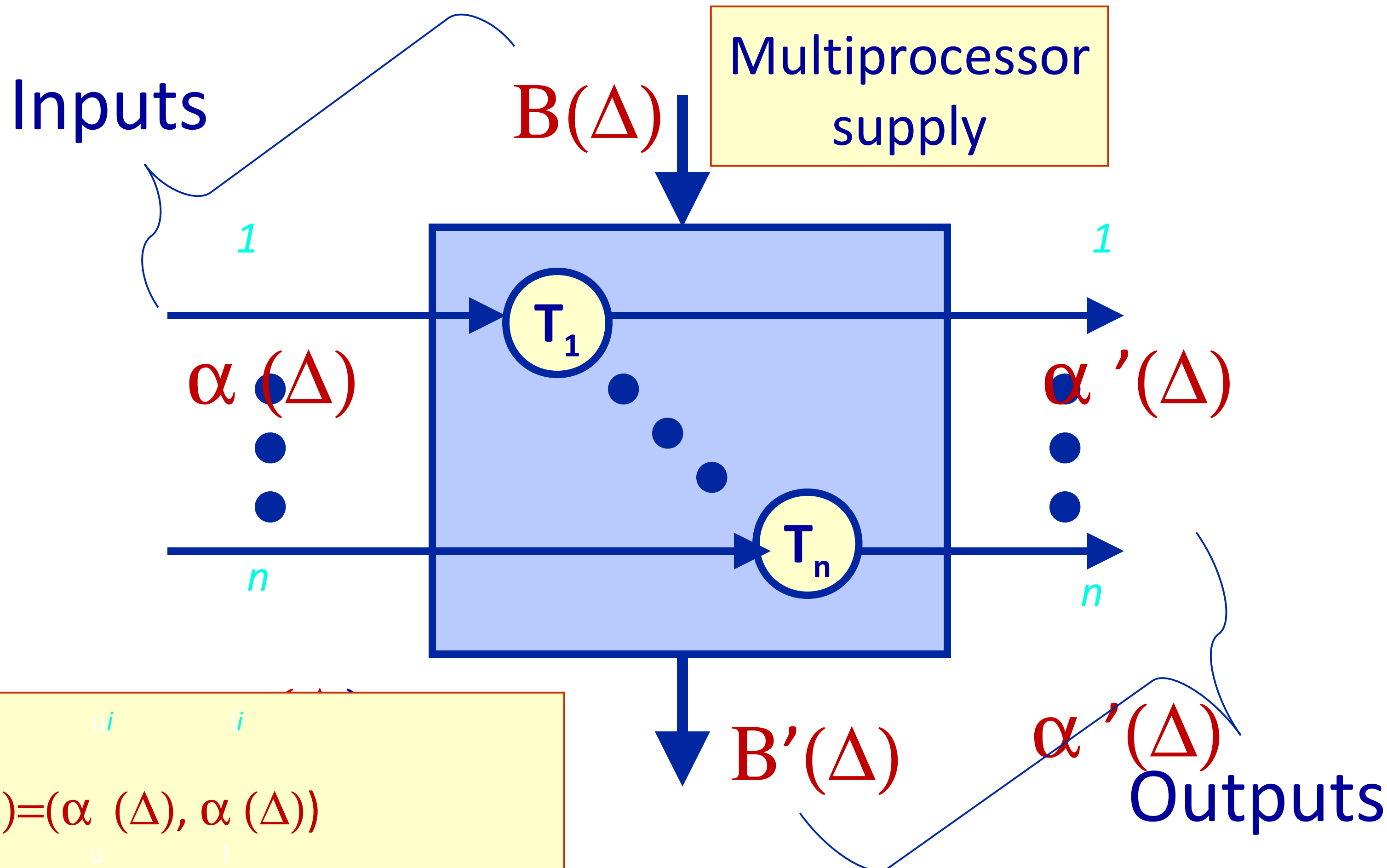
$$\forall s \le t : B^l(t-s) \le R(t) - R(s) \le B^u(t-s)$$



# Multiprocessor RTC

(System Model)




$\alpha(\Delta)=(\alpha(\Delta), \alpha(\Delta))$

$B(\Delta)=(B(\Delta), B(\Delta))$

$\gamma(k)$ execution requirement for

# Multiprocessor RTC

(Overview)

Prior work: [Chakraborty, Wandeler et. al]

Outputs are computed directly from inputs using min+ algebra, e.g.

This work:

$$\beta^{l'}(\Delta) = \sup_{0 \le \lambda \le \Delta} \left\{ \beta^{l}(\lambda) - \alpha^{u}(\lambda) \right\}$$

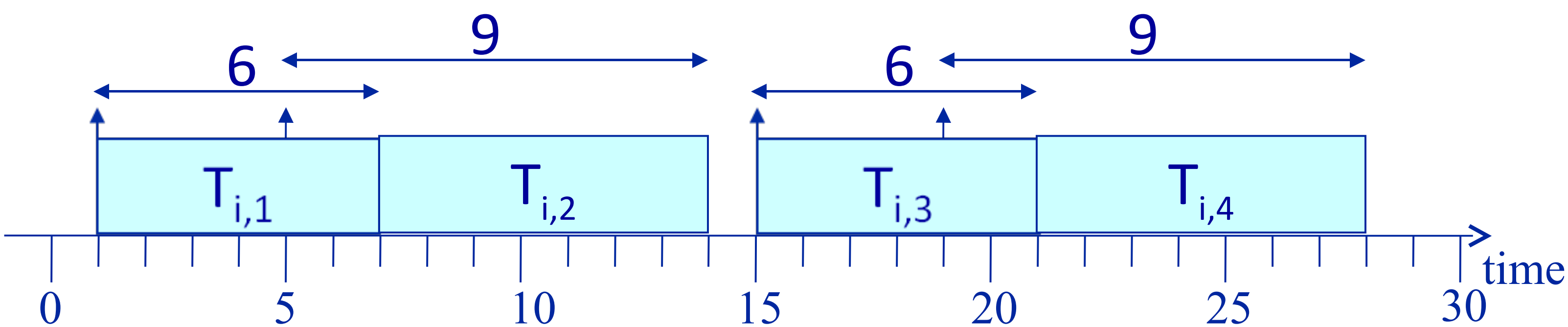
Calculate job response-time upper bounds $\Theta$

Use $\Theta$ to calculate $\alpha'(\Delta)$ and $\beta'(\Delta)$

# Multiprocessor RTC

(What is the response time?)

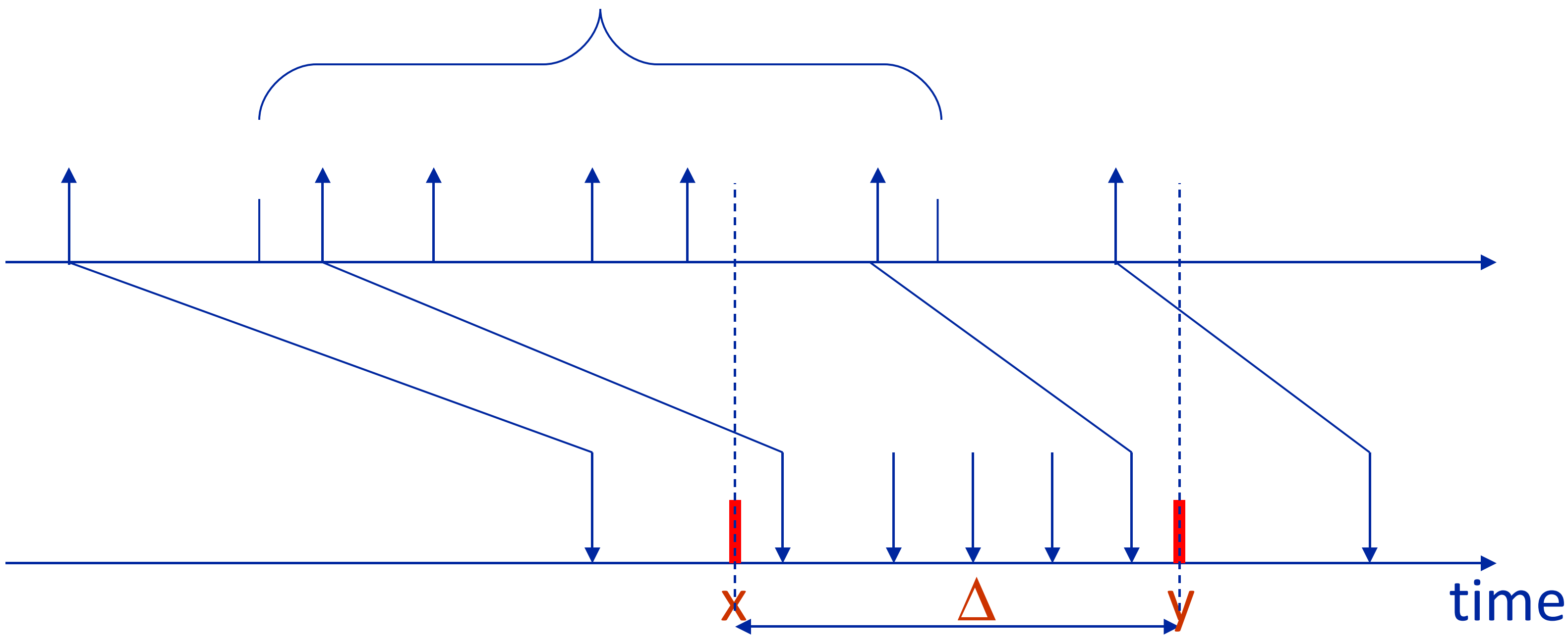
All jobs execute sequentially and in order!



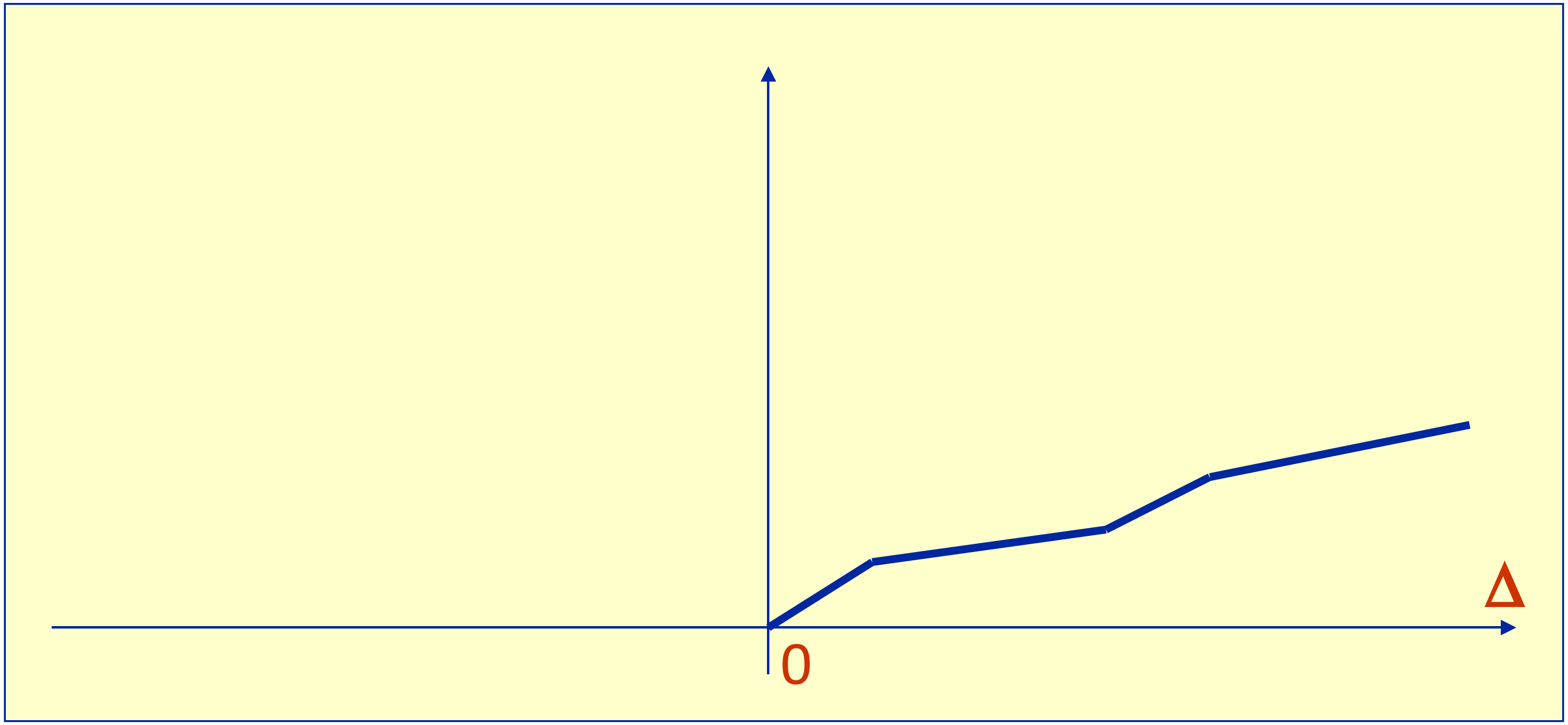

If maximum response time is bounded then job completion rate is the same as job arrival rate.

# Multiprocessor RTC

(Calculating job completions)

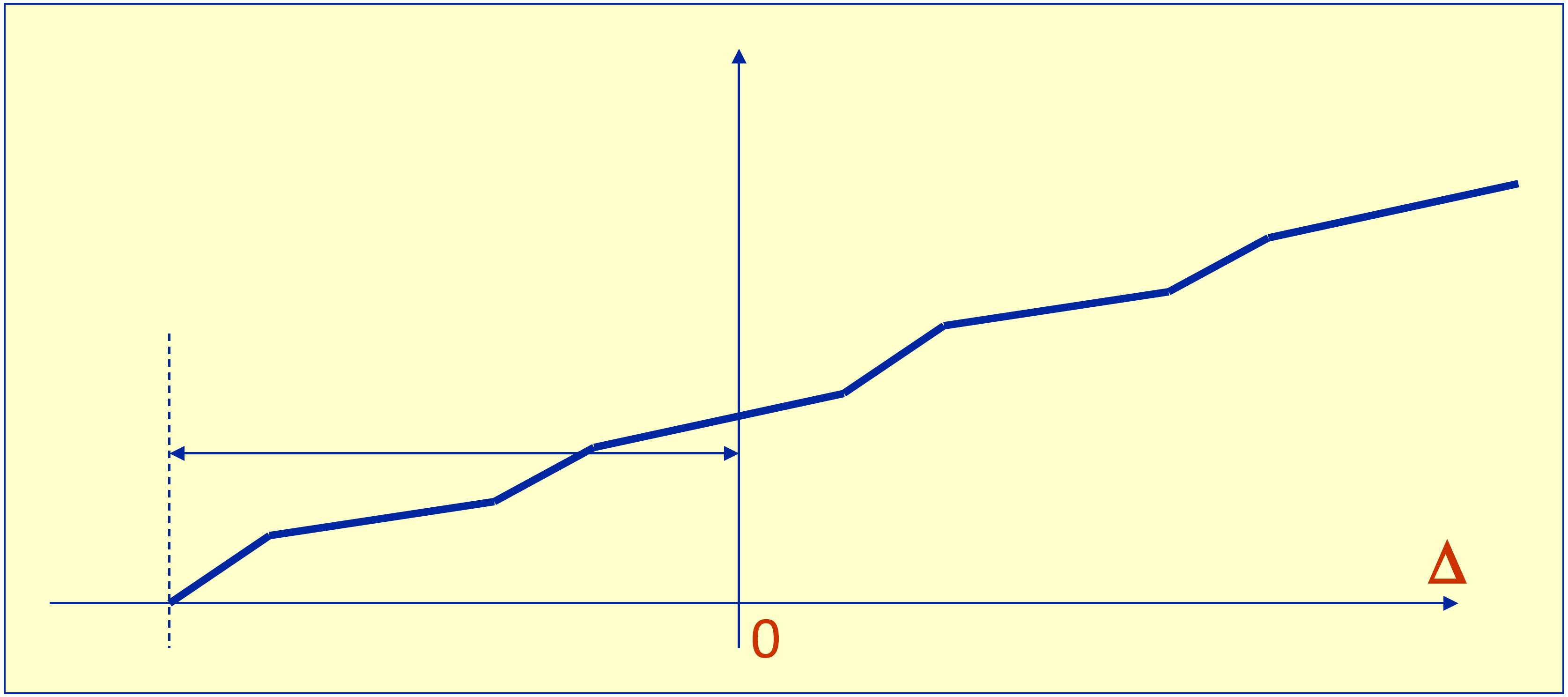


# Multiprocessor RTC
(Calculating Job Completion Curves)



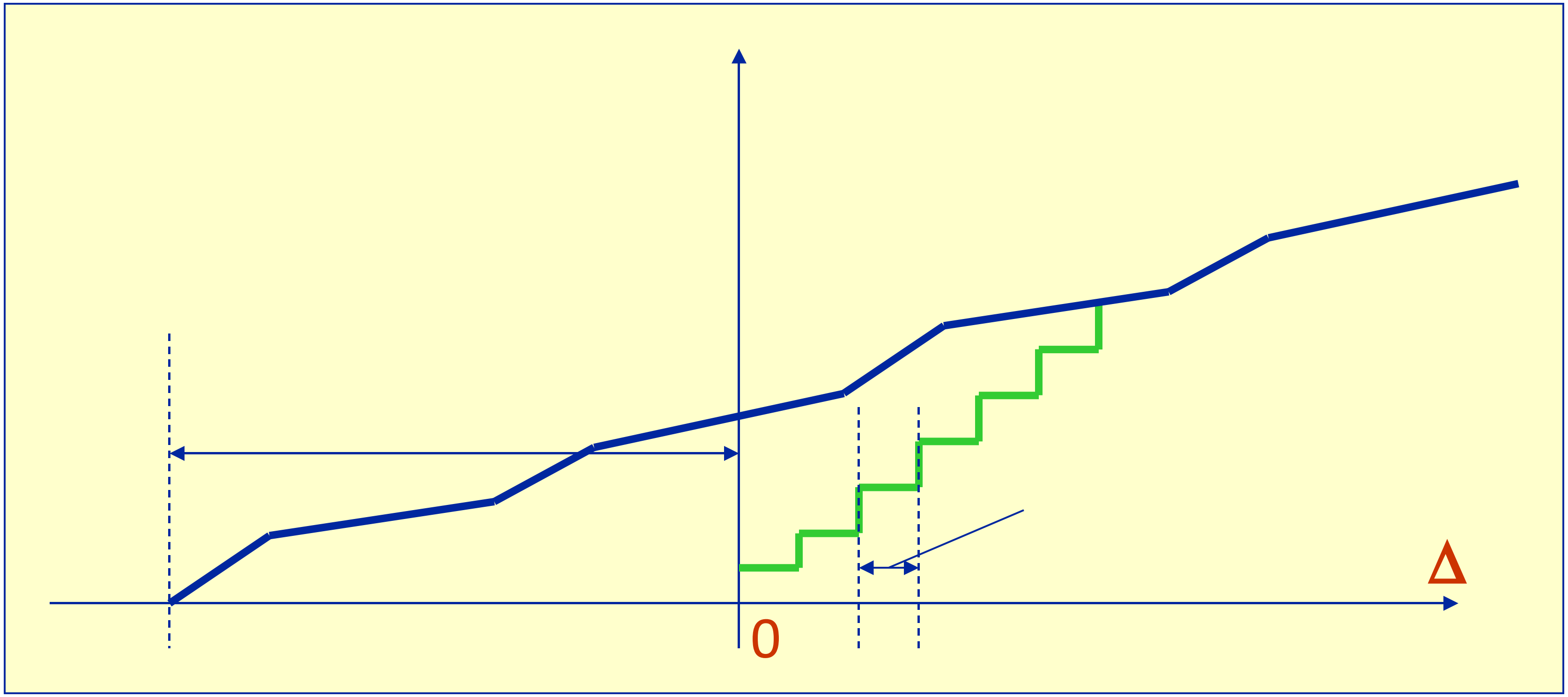
# Multiprocessor RTC

## (Calculating Job Completion Curves)



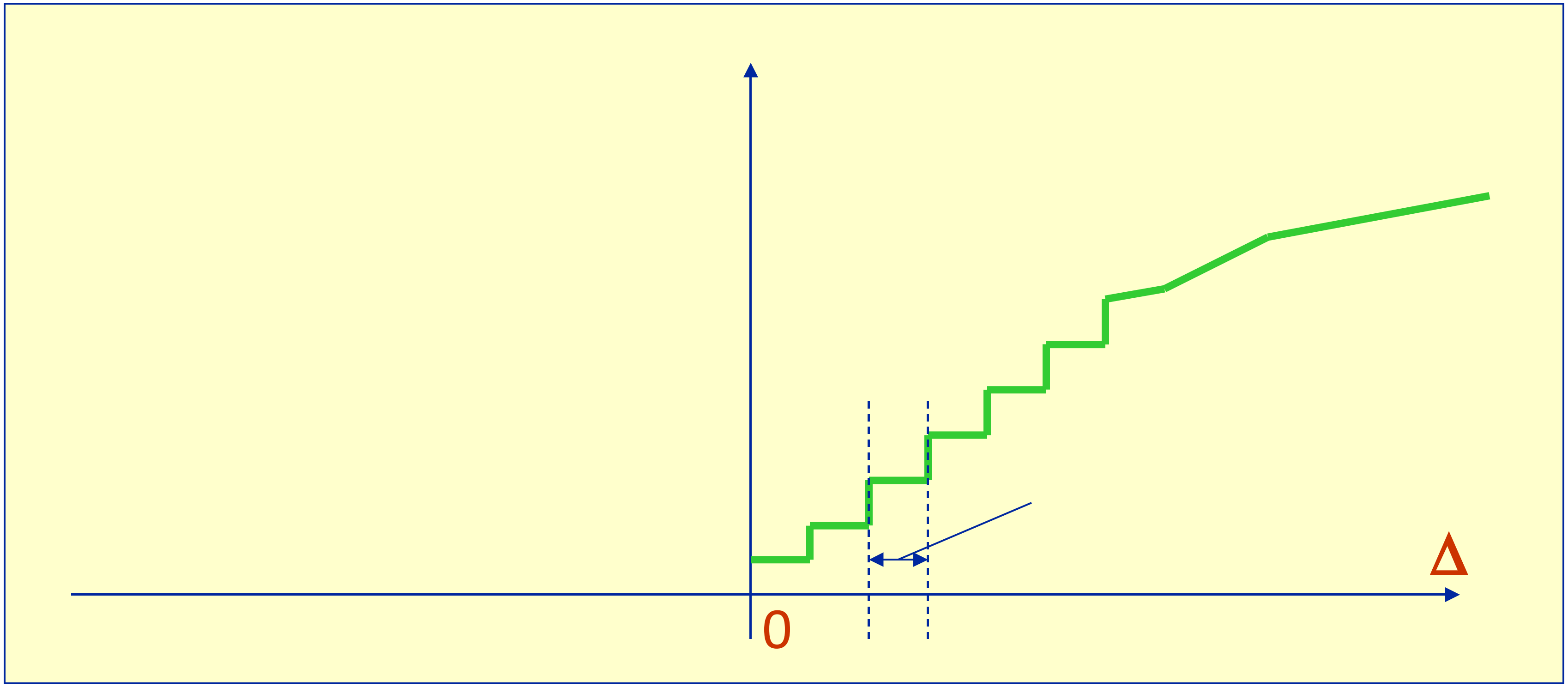
# Multiprocessor RTC

## (Calculating Job Completion Curves)



# Multiprocessor RTC

(Calculating Job Completion Curves)



# Multiprocessor RTC

(Calculating Remaining Supply)

Minimum guaranteed residual supply

Minimum guaranteed input supply

$$\beta^{l'}(\Delta) = \sup_{0 \le y \le \Delta} \left\{ \beta^{l}(y) - \alpha^{u}(y) \right\}$$

Maximum possible demand

## Uniprocessor RTC

## Multiprocessor RTC

Minimum guaranteed residual supply

Minimum guaranteed input supply

$$B'(\Delta) = \sup_{0 \le y \le \Delta} \left\{ B(y) - \sum_{T_i \in \tau} \min(y, \gamma_i(\alpha_i^{u}(y + \Theta_i))) \right\}$$

Maximum possible demand of $T_i$

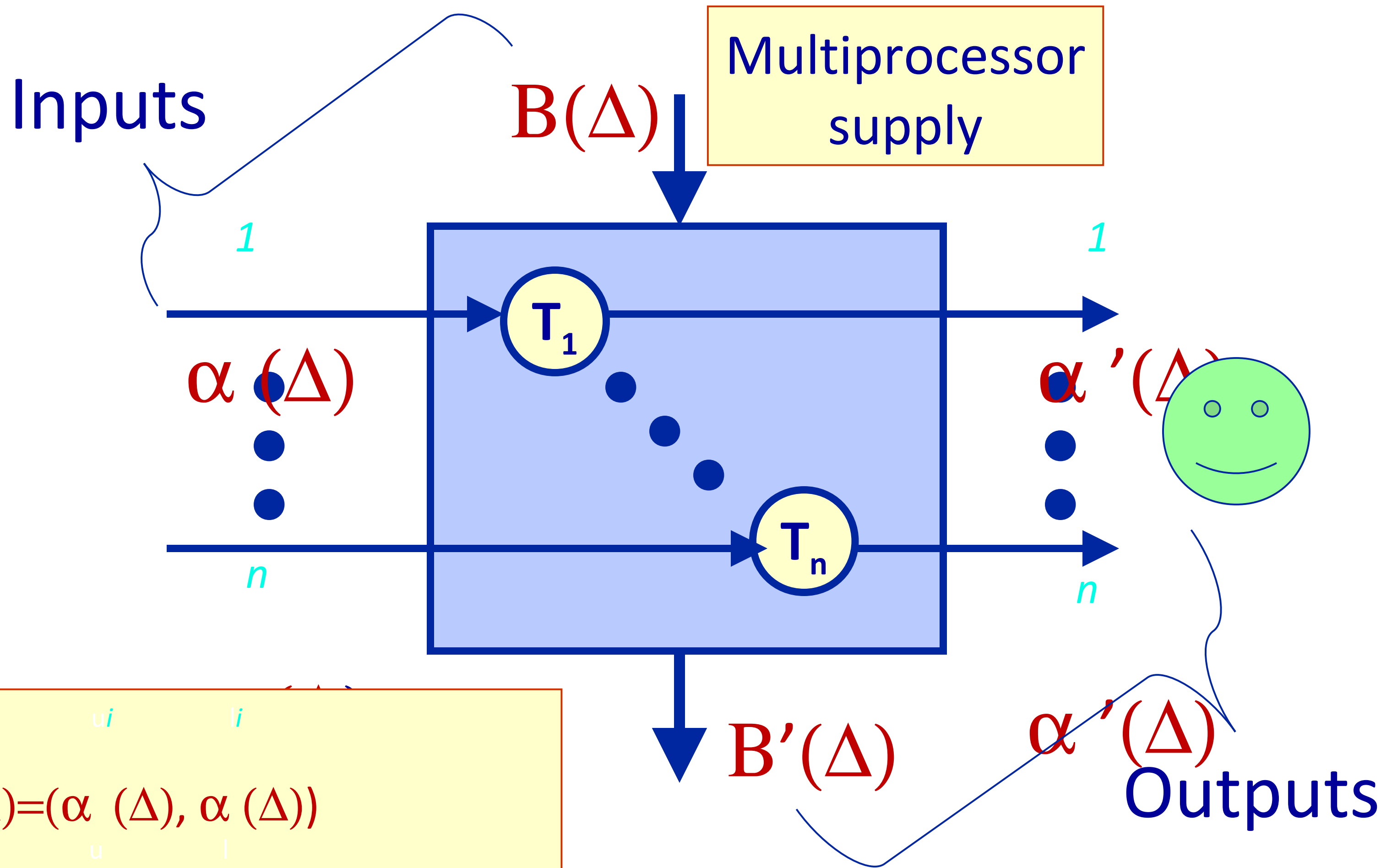
# Multiprocessor RTC

(Finding Response-Time Bounds)

- Pseudopolynomial time procedure for checking response-time bounds under global EDF
- Based on prior work by Baruah [RTSS'07] and Leontyev and Anderson [RTSS'08]
- Bounds are computed by iterative checking
- Currently working on finding closed-form expressions

# Multiprocessor RTC

(System Model)



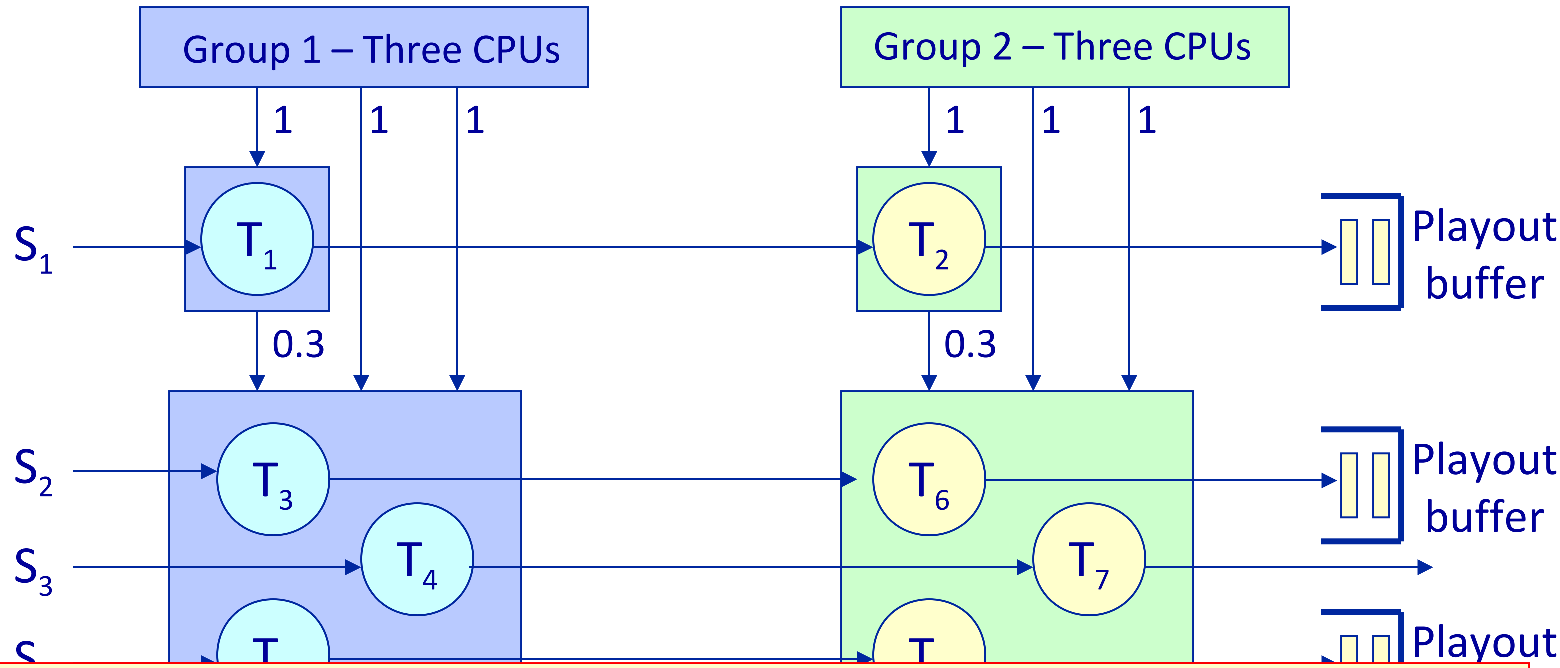

$\alpha(\Delta)=(\alpha_u(\Delta), \alpha_l(\Delta))$

$B(\Delta)=(B(\Delta), B(\Delta))$

$\gamma(k)$ execution requirement for

# Example

## (Multiprocessor Execution of MPEG-2 player)




6 CPUs are sufficient (vs. 8 under conventional RTC)
Buffer requirements for S2, S3, S4 (non-critical video) are higher (2 seconds) than those under partitioning

# Multiprocessor RTC

(Summary)

- New type of building blocks for multiprocessor systems
- Wider range of supported workloads
- High computational complexity
- Future work:
  - » Improving analysis accuracy
  - » Improving computational complexity

# Outline

- Motivation/Background
- My research
  - » Distributing processing power among components
    - – Hierarchical bandwidth reservation scheme
  - » Analysis of a single component
    - – Multiprocessor extensions to real-time calculus
- Research goals
- Concluding Remarks

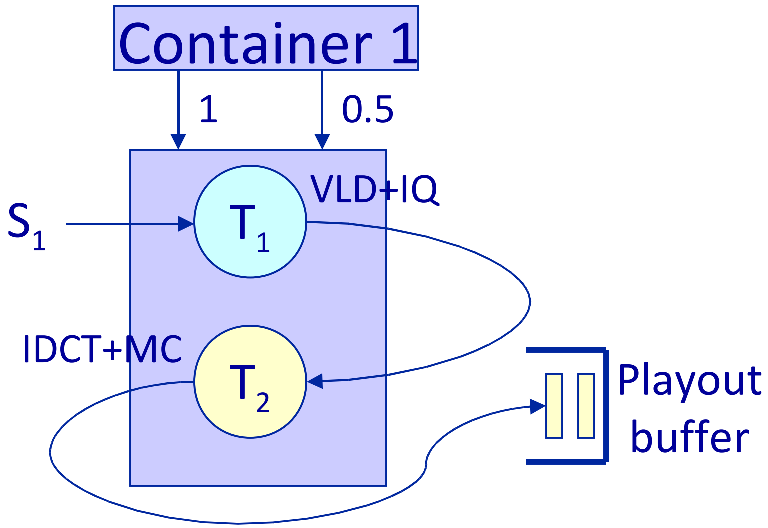
# Research Goals

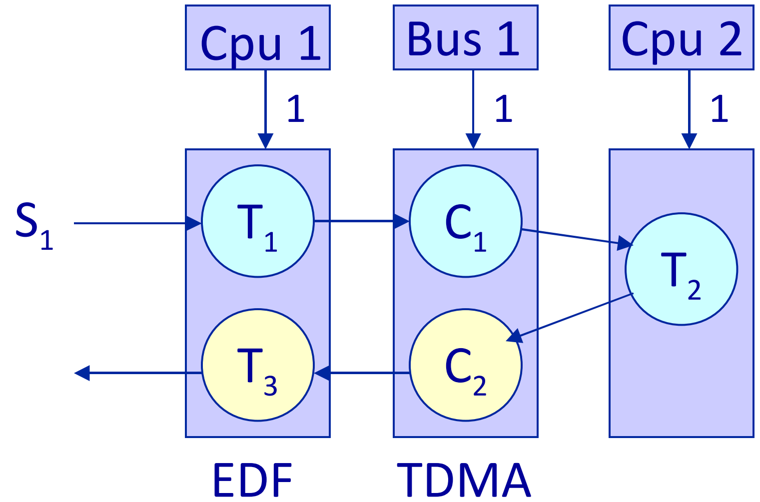
- Non-preemptive case
- Synchronization across containers
- Task interference
- Cyclic-dependencies between tasks/Pipelines
- Mutual exclusion

# Research Goals

- Non-preemptive case
- Synchronization across containers
- Task interference
- Cyclic-dependencies between tasks/Pipelines

# Research Goals
## (Cyclic dependencies/Pipelines)




Execution of $T_1$ and $T_2$ may overlap

No theory even for partitioned case

# Concluding Remarks

- Two approaches that extend state-of-the-art analysis
  - » Hierarchical container-based scheme
  - » Multiprocessor RTC
- New type of systems can be analyzed
- Compatible with previously developed theory
- Many promising directions for future work

Thank you!