The Real-Time Multi-Resource Task Model

RTSOPS'12

Pisa, Italy

July 10th, 2012

Cong Liu

The University of North Carolina at Chapel Hill

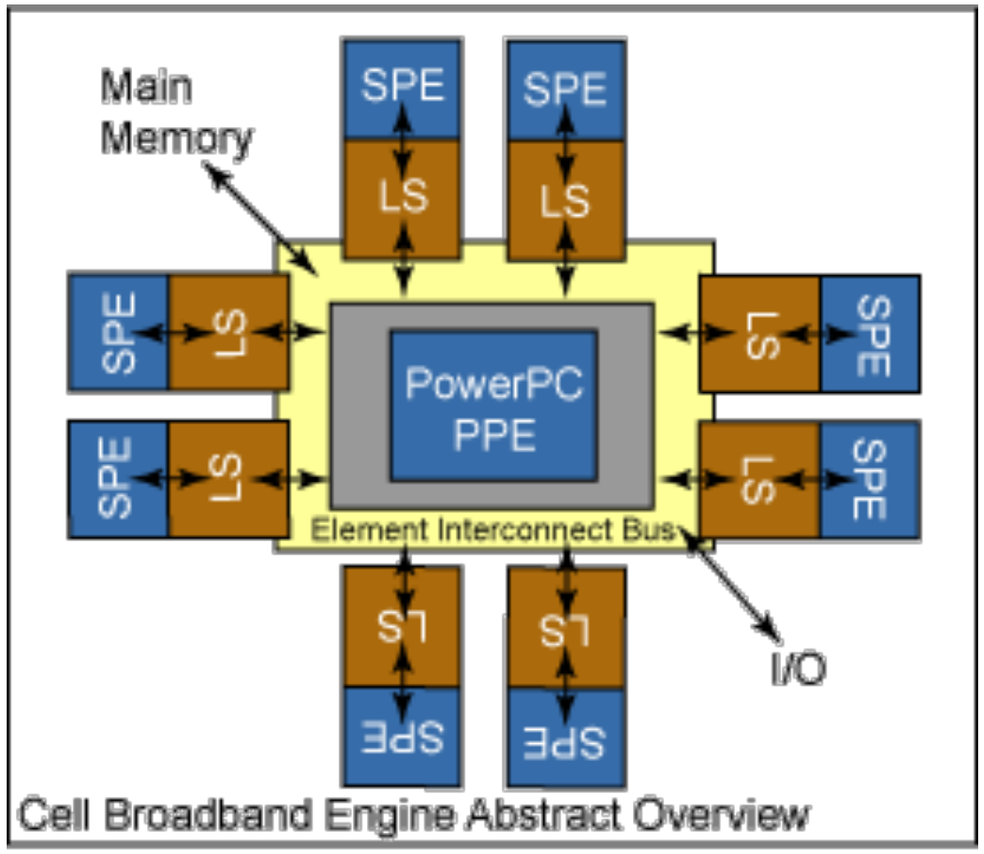
Work supported by NSF grants CNS 0834270, CNS 0834132, and CNS 1016954; ARO grant W911NF-09-1-0535; AFOSR grant FA9550-09-1-0549; and AFRL grant FA8750-11-1-0033

Heterogeneous Systems for Real-Time Computing

- Integrate additional specialized resources such as FPGA, GPU
  - high performance
  - energy efficiency
  - highly reactive systems that interact with other environments

Heterogeneous Systems for Real-Time Computing

- Integrate additional specialized resources such as FPGA, GPU
  - high performance
  - energy efficiency
  - highly reactive systems that interact with other environments

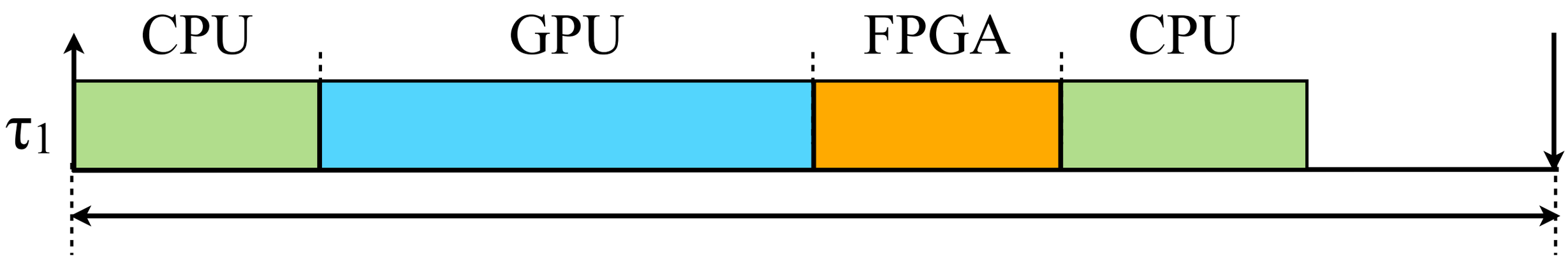


- IBM cell architecture
  - One "general" processor with eight "specialized" processors
  - Specialized processors designed for handling vectorized floating point code execution

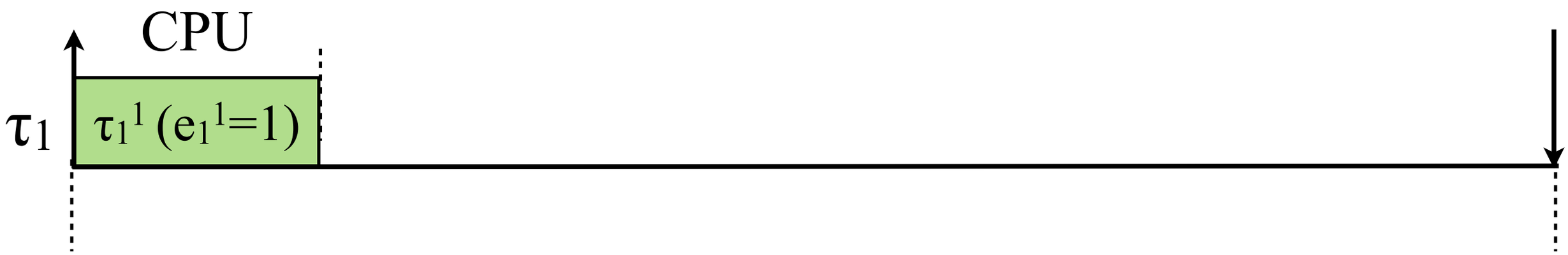
Real-Time Sporadic Multi-Resource (SMR) Task Model

- Model real-time tasks that access multiple resources during execution
  - ➡ Extend the sporadic task model
  - ➡ Sporadically release jobs
  - ➡ Each job contains several phases, each executed on a specific resource

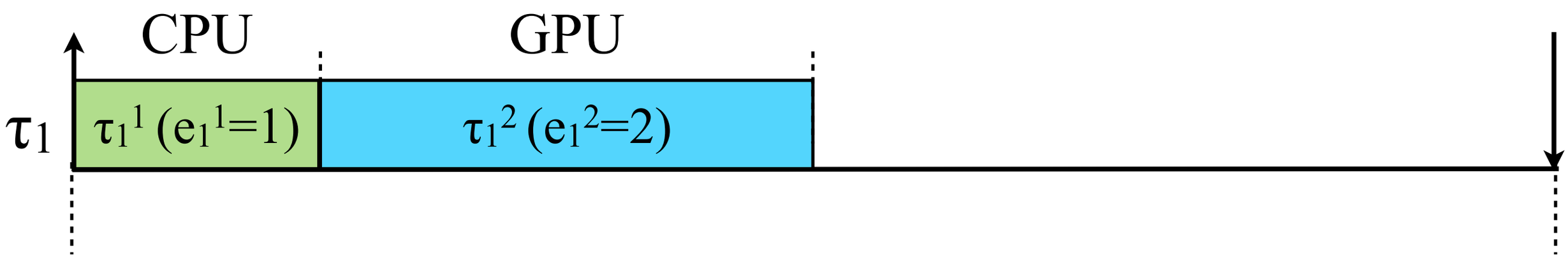
Real-Time Sporadic Multi-Resource (SMR) Task Model



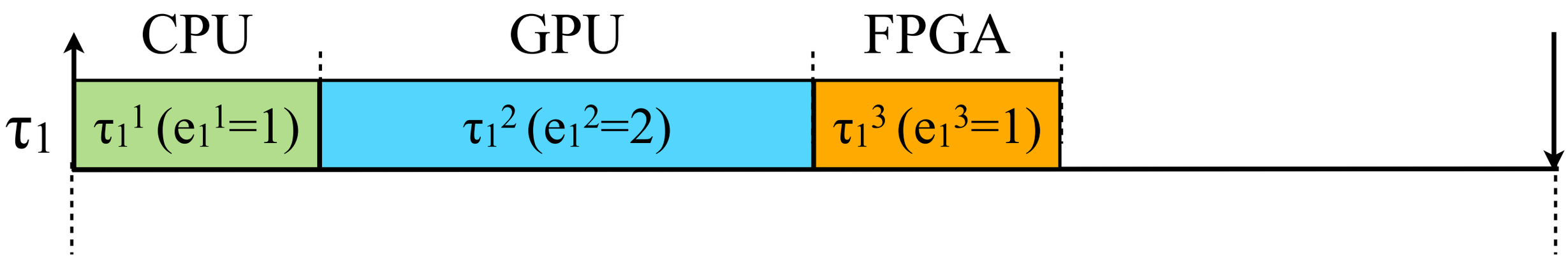
Real-Time Sporadic Multi-Resource (SMR) Task Model



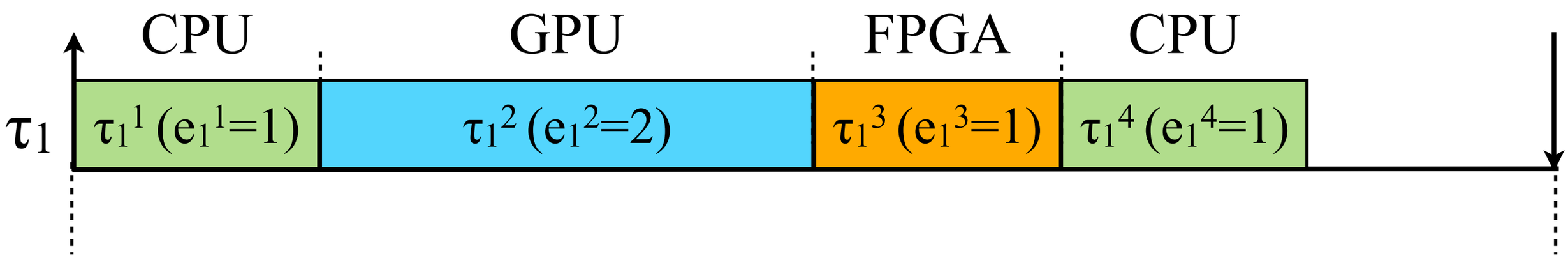
Real-Time Sporadic Multi-Resource (SMR) Task Model



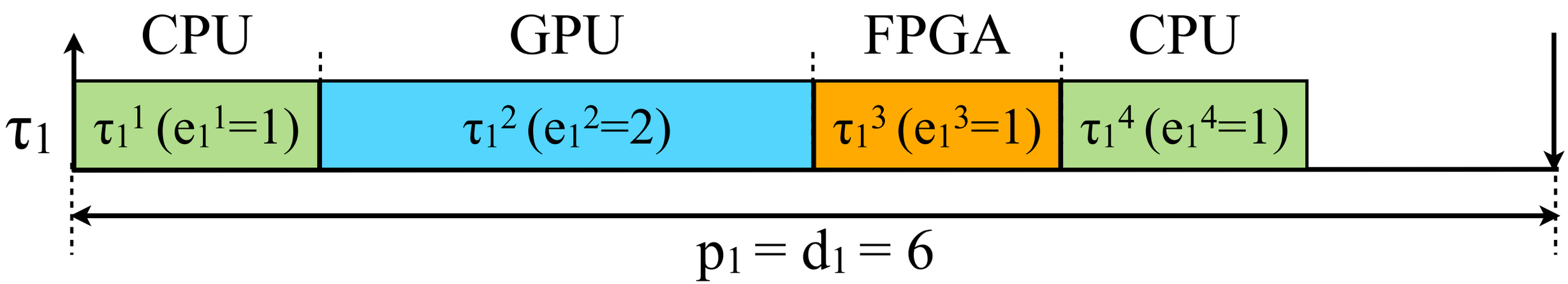
Real-Time Sporadic Multi-Resource (SMR) Task Model



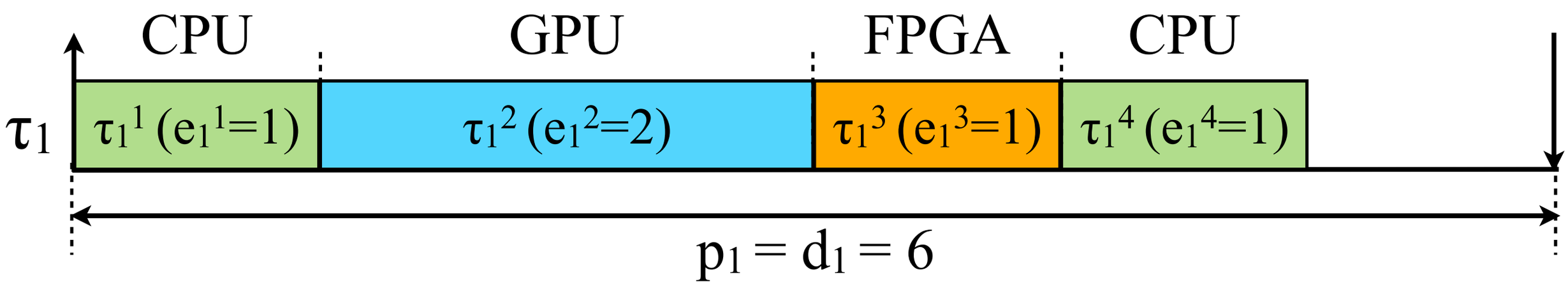
Real-Time Sporadic Multi-Resource (SMR) Task Model



Real-Time Sporadic Multi-Resource (SMR) Task Model



Real-Time Sporadic Multi-Resource (SMR) Task Model



- Utilization on CPU = 2 / 6

Scheduling Restrictions on Certain Resources

- Non-preemptivity or non-job-migration restrictions
  - GPU: non-preemptive
  - Job migrations may cause significant overheads on resources such as GPU and FPGA

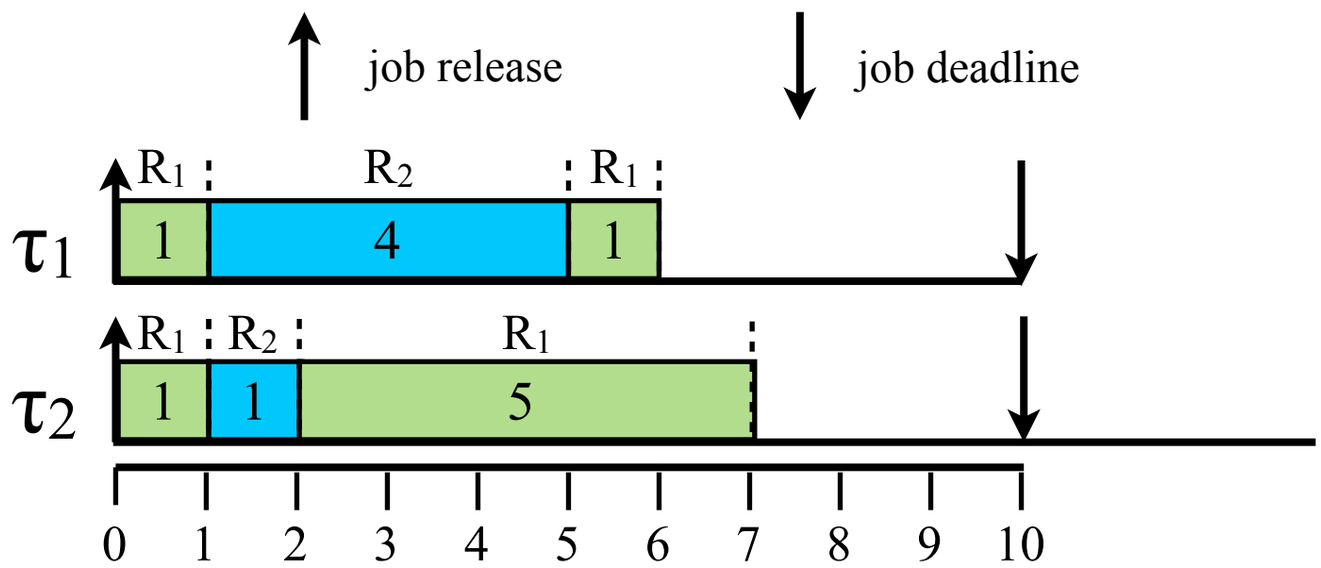
The Open Problem

How to schedule a set of hard real-time SMR tasks on a heterogeneous platform consisting of multiple types of resources, where each type of resource may have multiple processors and require certain scheduling restrictions?

The Challenge

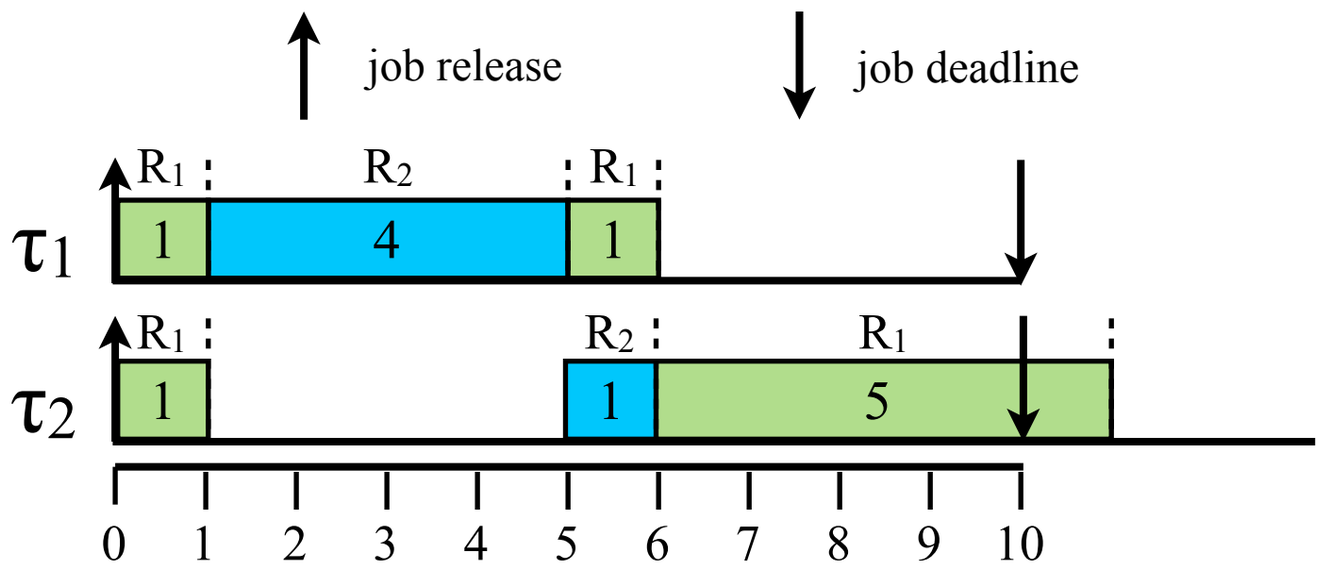
- Difficult due to precedence constraints and interferences between executions of tasks on multiple resources

The Challenge



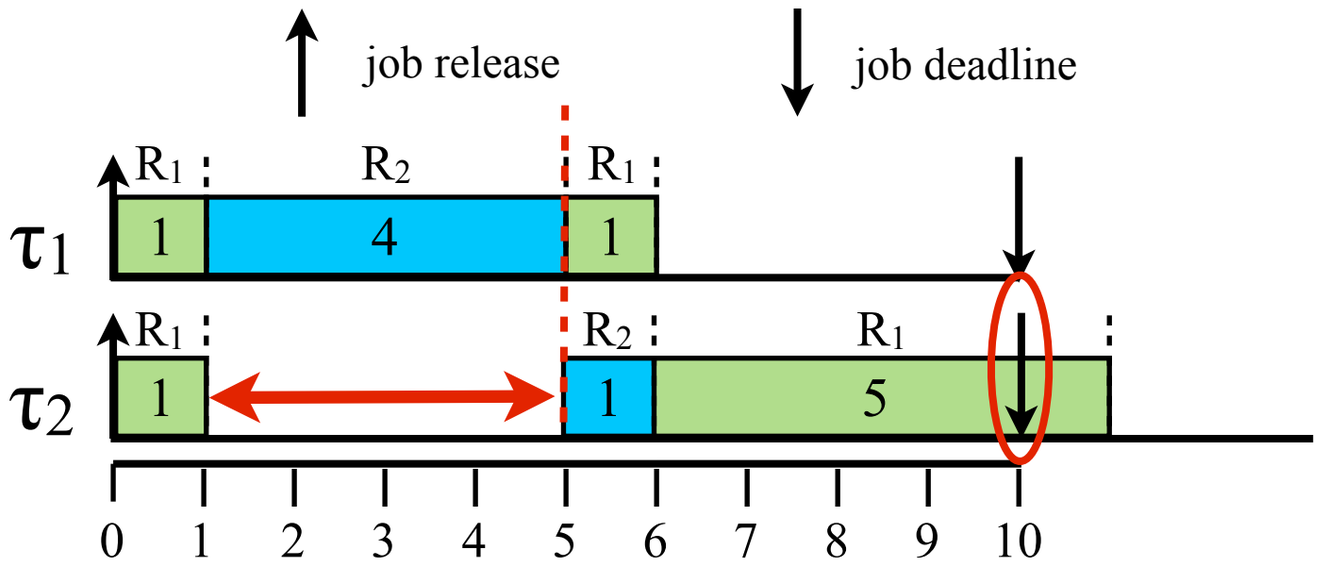
- Two SMR tasks executed on two resources, where R_1 has two processors and R_2 has one processor
- Lightly loaded system: 0.8 and 0.5 for the total utilization on R_1 and R_2, respectively

The Challenge



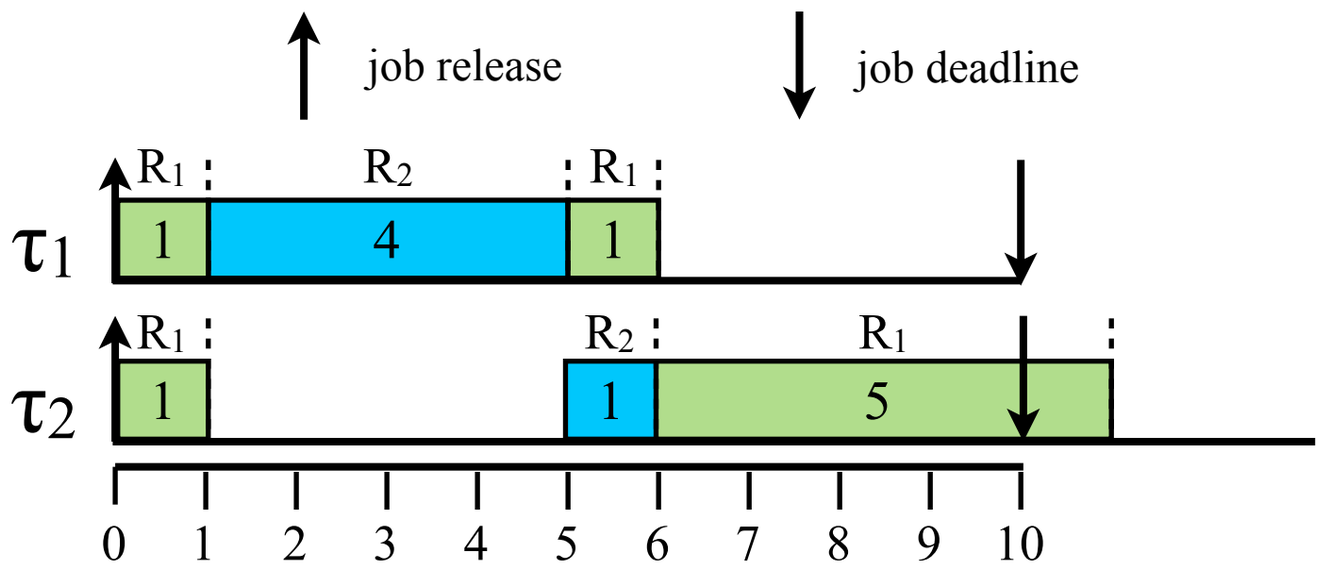
- Two SMR tasks executed on two resources, where R_1 has two processors and R_2 has one processor
- Lightly loaded system: 0.8 and 0.5 for the total utilization on R_1 and R_2, respectively

The Challenge



- Two SMR tasks executed on two resources, where R_1 has two processors and R_2 has one processor
- Lightly loaded system: 0.8 and 0.5 for the total utilization on R_1 and R_2, respectively

The Challenge

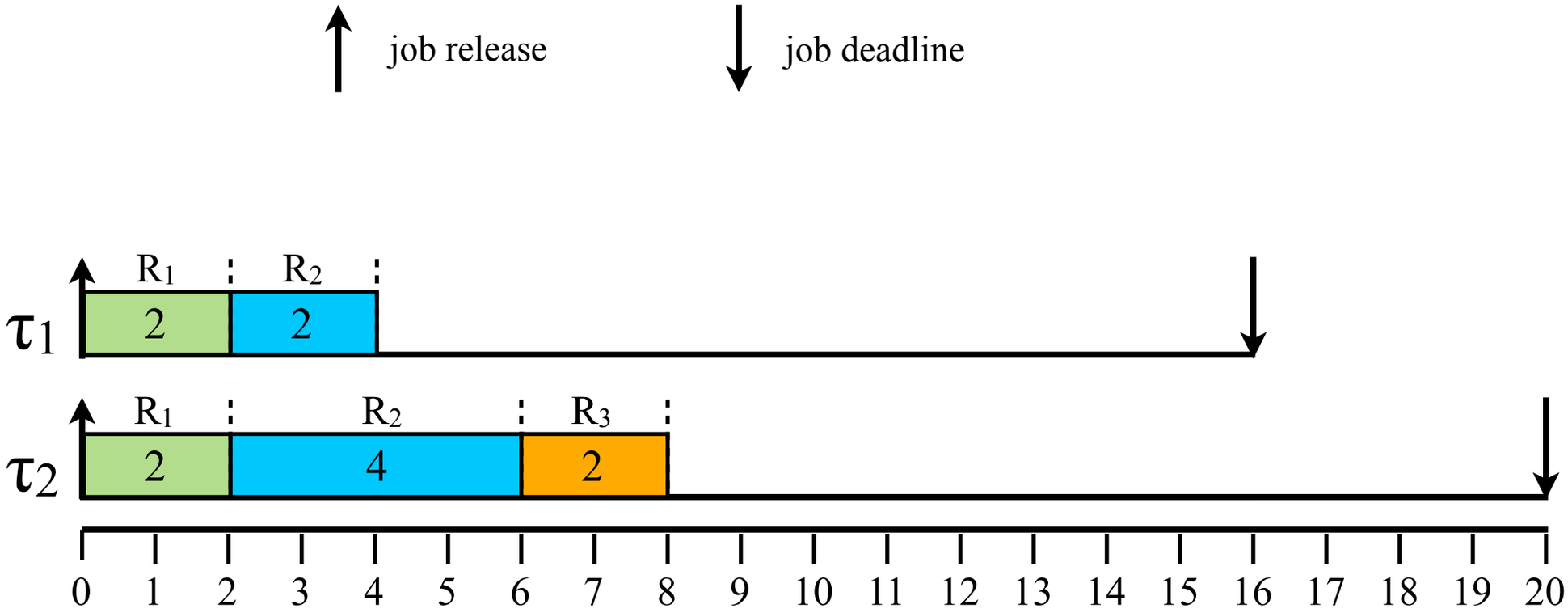


The precedence constraints among phases belonging to the same SMR task plus the interferences brought by other tasks quite negatively impact schedulability

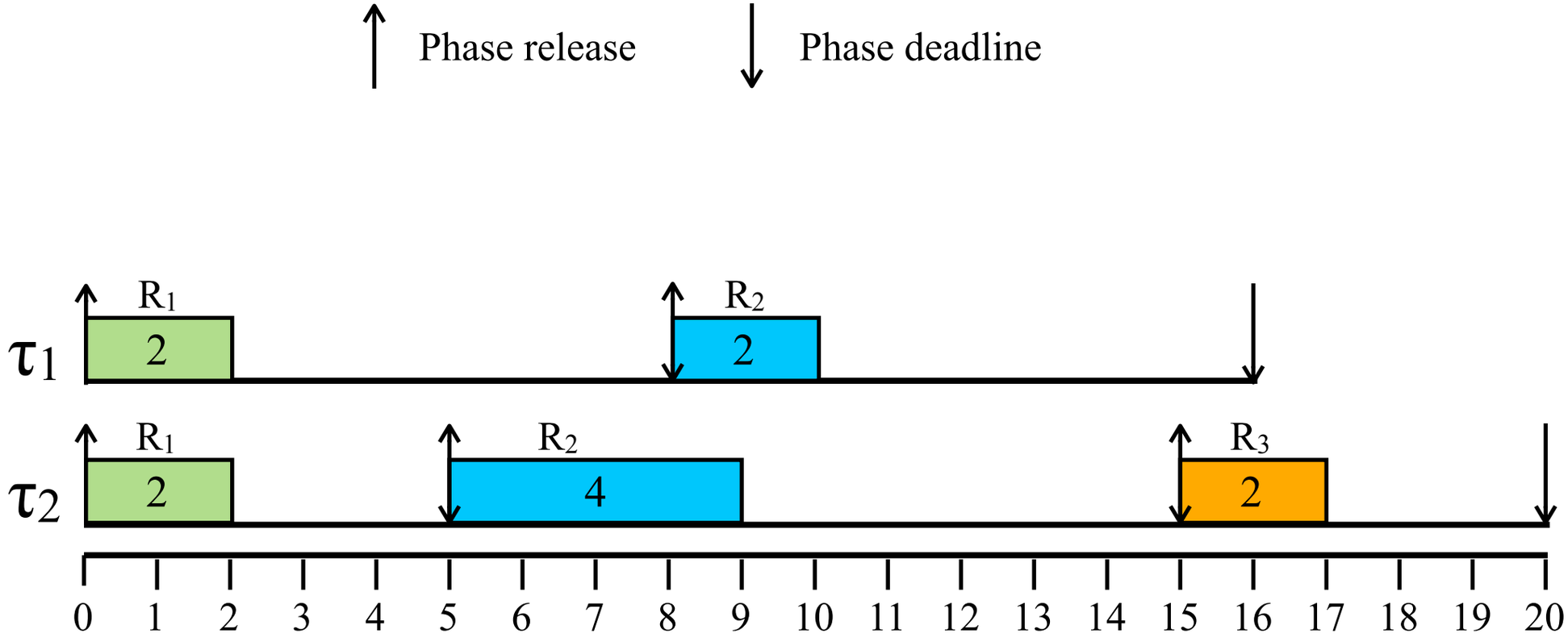
An Intuitive Approach: Assigning Intermediate Releases and Deadlines

- Each phase of an SMR task is transformed into a constrained-deadline subtask
  - ➡ Each subtask requests a single resource
  - ➡ Apply corresponding existing schedulability tests on each resource
  - ➡ The original SMR task system is schedulable if the transformed subtasks on every resource are schedulable

An Example


job release
job deadline
τ_1
R_1
R_2
2
2
τ_2
R_1
R_2
R_3
2
4
2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

An Example



- If all phases of an SMR task can meet their assigned intermediate deadlines, they become independent from each other

Other Insights

- Study special cases
  - ➡ SMR task systems that only request two resources each of which contains a single processor
  - ➡ Execution times of all phases of all tasks are identical