AVICPS Workshop – December 2013

Towards a Safe Compositional Scheduling Theory for CPS

Linh Thi Xuan Phan

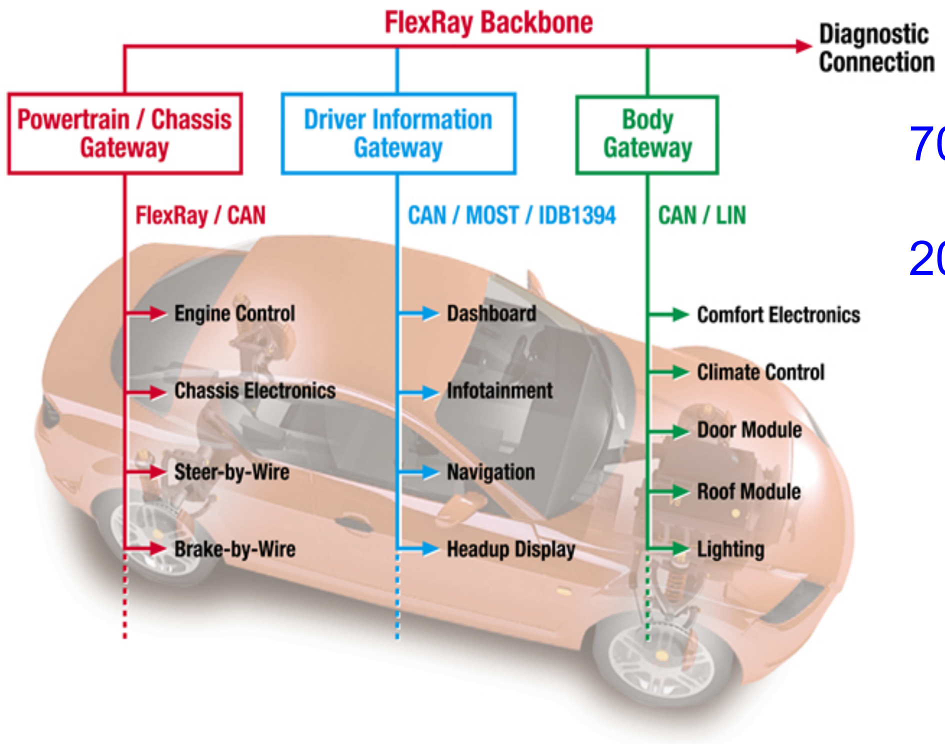
University of Pennsylvania

Penn Engineering

PRECISE

PENN RESEARCH IN EMBEDDED COMPUTING AND INTEGRATED SYSTEMS ENGINEERING

Trend: Complexity



70 to 100 electronic control units (ECUs)

2000 to 3000 software functions

- Increasing number of software components
- Increasing resource sharing, due to SWaP constraints
- Existing techniques: inefficient, pessimistic
- **Need a scalable analysis and resource-efficient design**

Approach: Compositional design

- Design and analyze compositionally via interfaces
  - Break the problem down into smaller problems
  - Perform the design and analysis locally
  - Create an interface that abstracts away details and exposes only the key properties
  - Reason about the composition of interfaces during system integration
- Traditional focus: functional and behavioral aspects
  - e.g., AADL interfaces
- CPS: Need abstraction of **timing** and **resource** aspects
  - CPS components manage their own resources
- **Idea**: Compositional scheduling and timing analysis via **resource-aware interfaces**

Compositional Analysis via Resource-Aware Interfaces

A brief introduction

CPS component

- **Workload**
  - Primitive component: real-time tasks, e.g., periodic tasks
  - Composite component: smaller components, or components + tasks

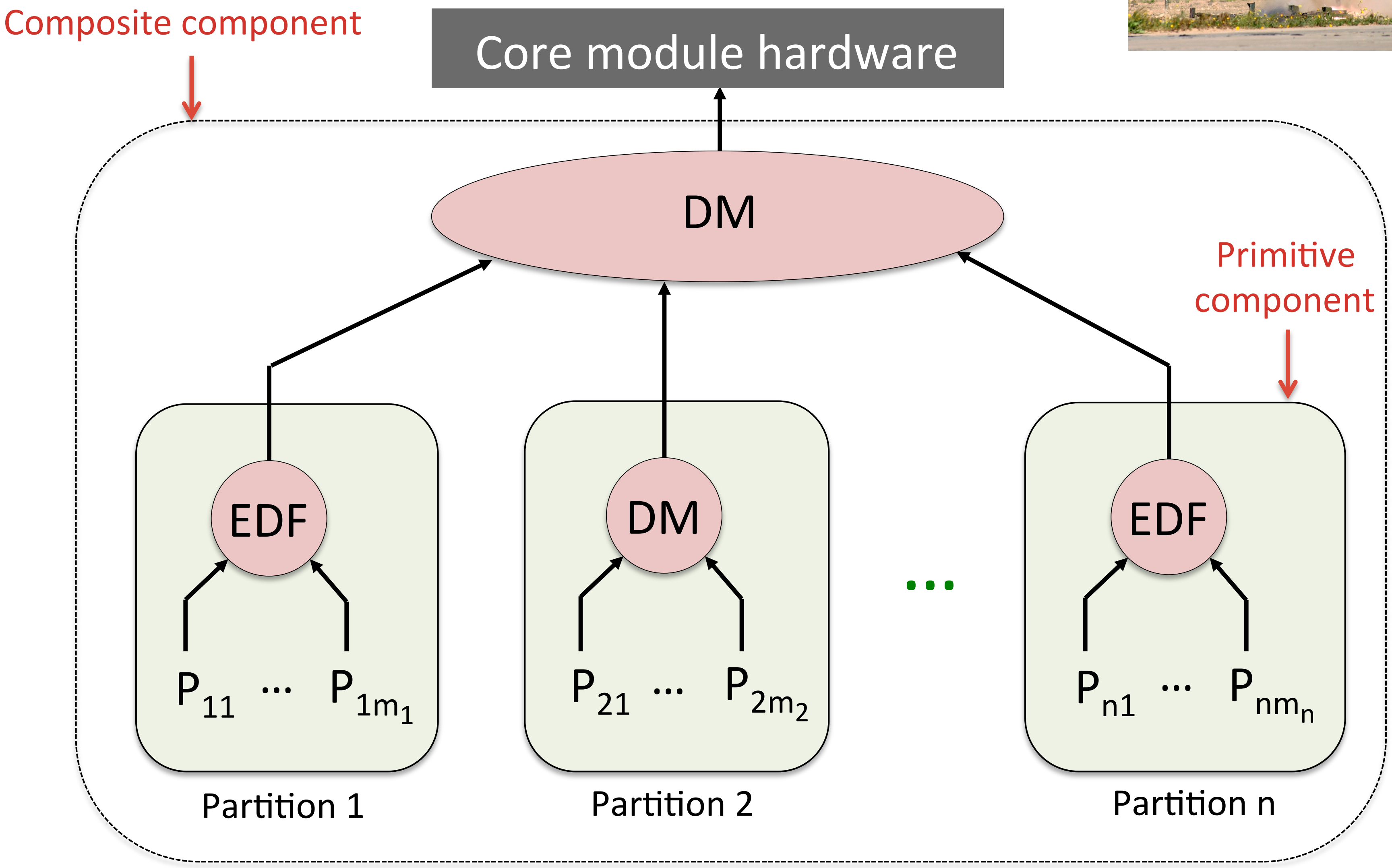
- **Scheduling algorithm**: any existing algorithms, e.g.,
  - Earliest deadline first (EDF), global EDF (gEDF) for multicore
    - Active job with the earliest absolute deadline is executed
  - Fixed-priority, e.g., Deadline monotonic (DM), gDM for multicore
    - Active job with the smallest relative deadline is executed

- In general: component's resource demand depends on both the workload and the scheduling algorithm

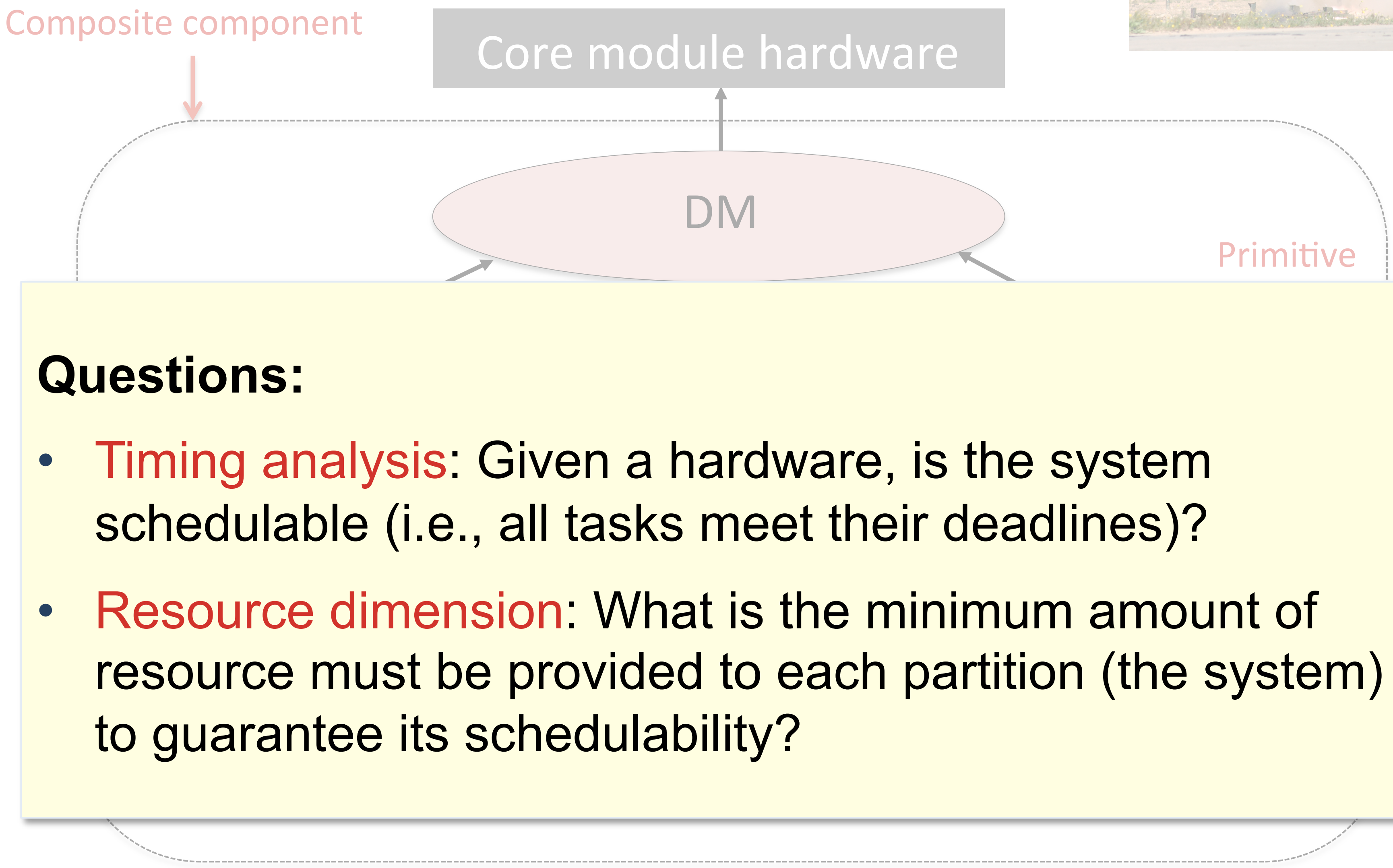
Resource-aware interface

- An abstraction of timing and resource aspects
  - Captures the minimum amount of resource supply must be given to the component to ensure its schedulability
  - A component is schedulable if its interface is satisfied!
- **How to represent an interface?**
  - **Using an interface model**
  - Example: Explicit Deadline Periodic (EDP) : (Π, Θ, Δ)
    - provides a budget of Θ resource units within a deadline Δ in each period Π
    - resource bandwidth = Θ/Π
- **How to compute the interface?**
  - **Intuition: Based on component schedulability test**
  - Example: An interface I can feasibly schedule all tasks of C under EDF iff its resource supply ≥ total resource demand of the tasks

Example: ARINC 653



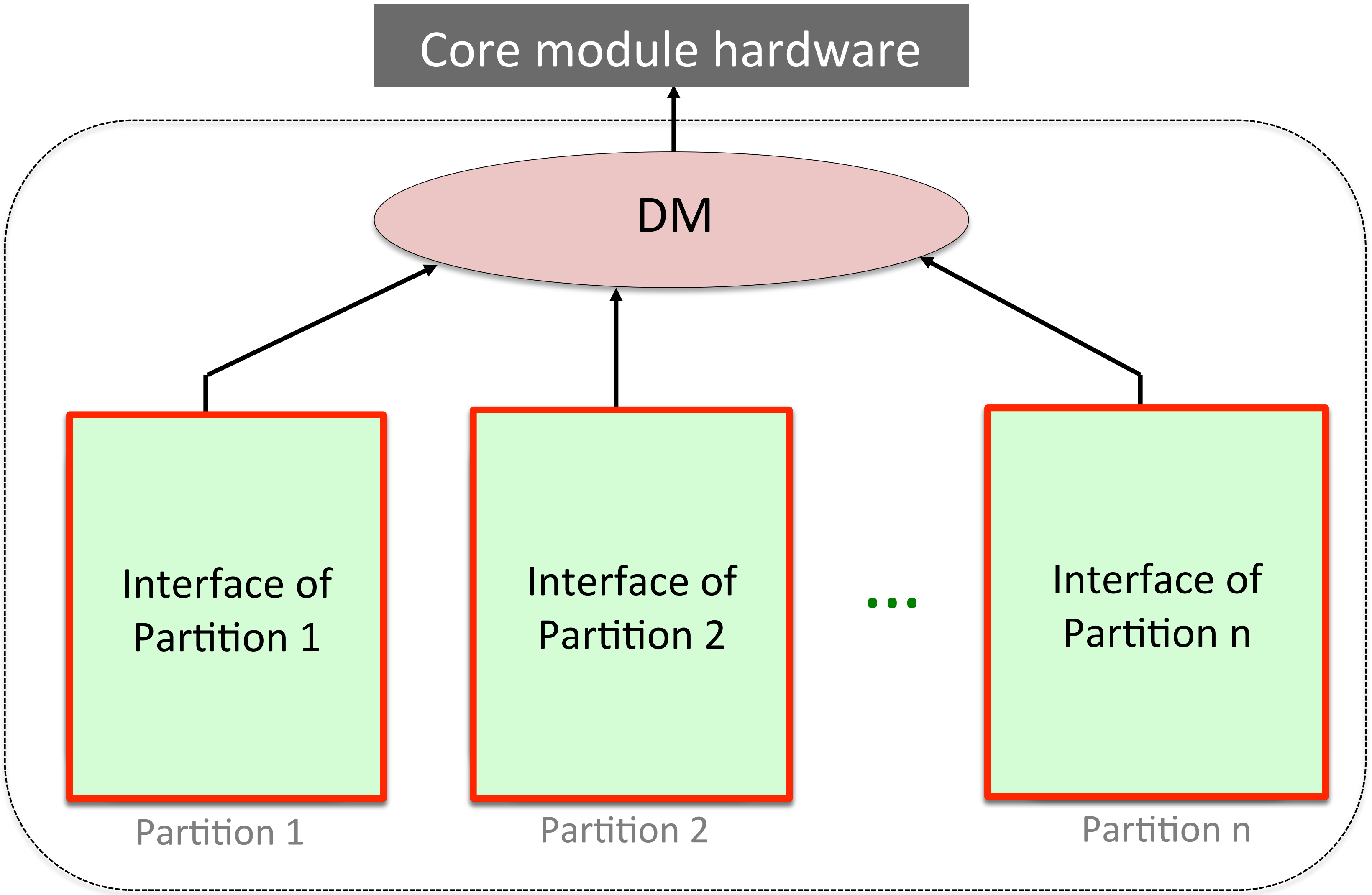
Example: ARINC 653



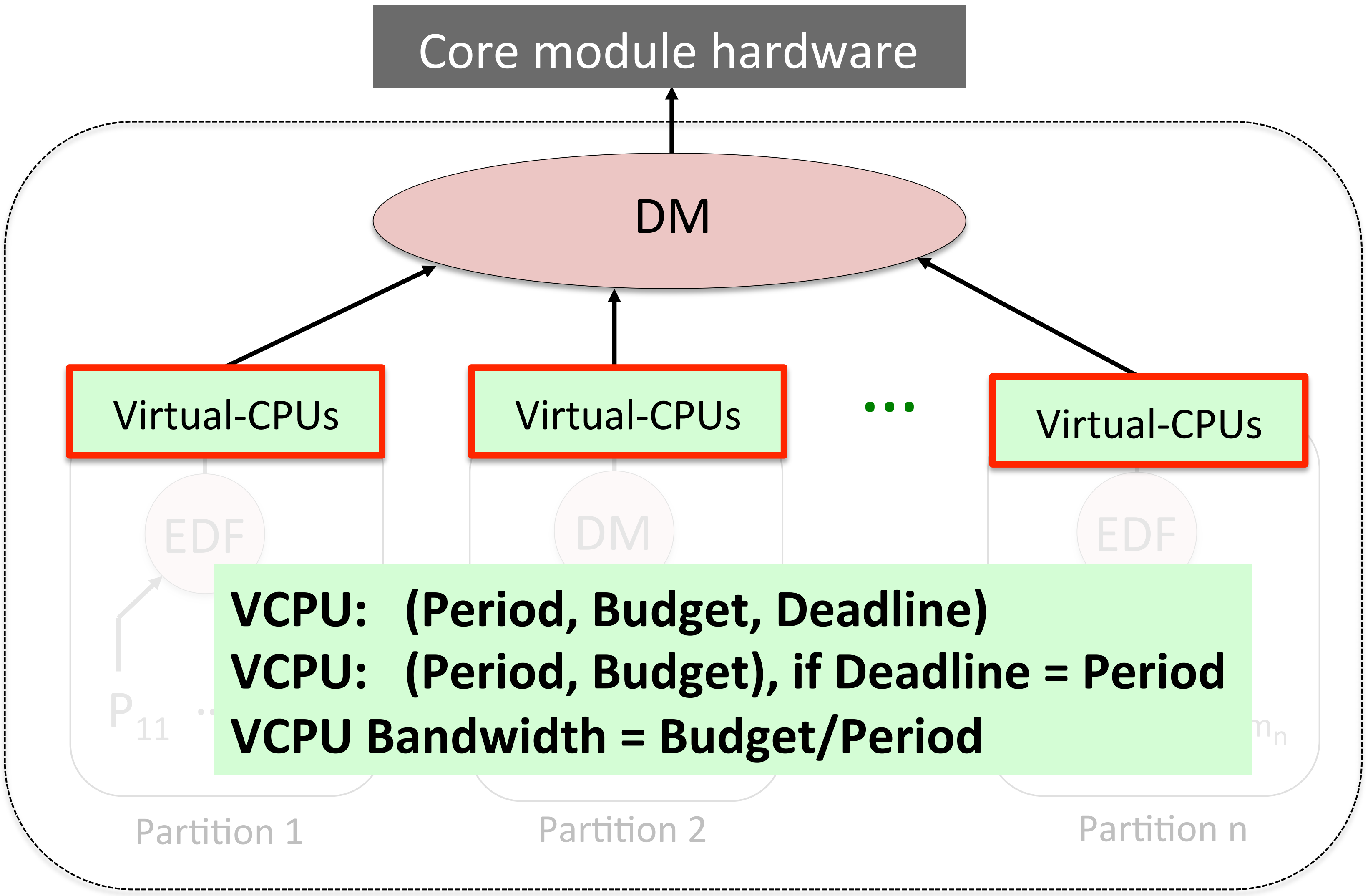
Questions:

- Timing analysis: Given a hardware, is the system schedulable (i.e., all tasks meet their deadlines)?
- Resource dimension: What is the minimum amount of resource must be provided to each partition (the system) to guarantee its schedulability?

Compositional analysis overview

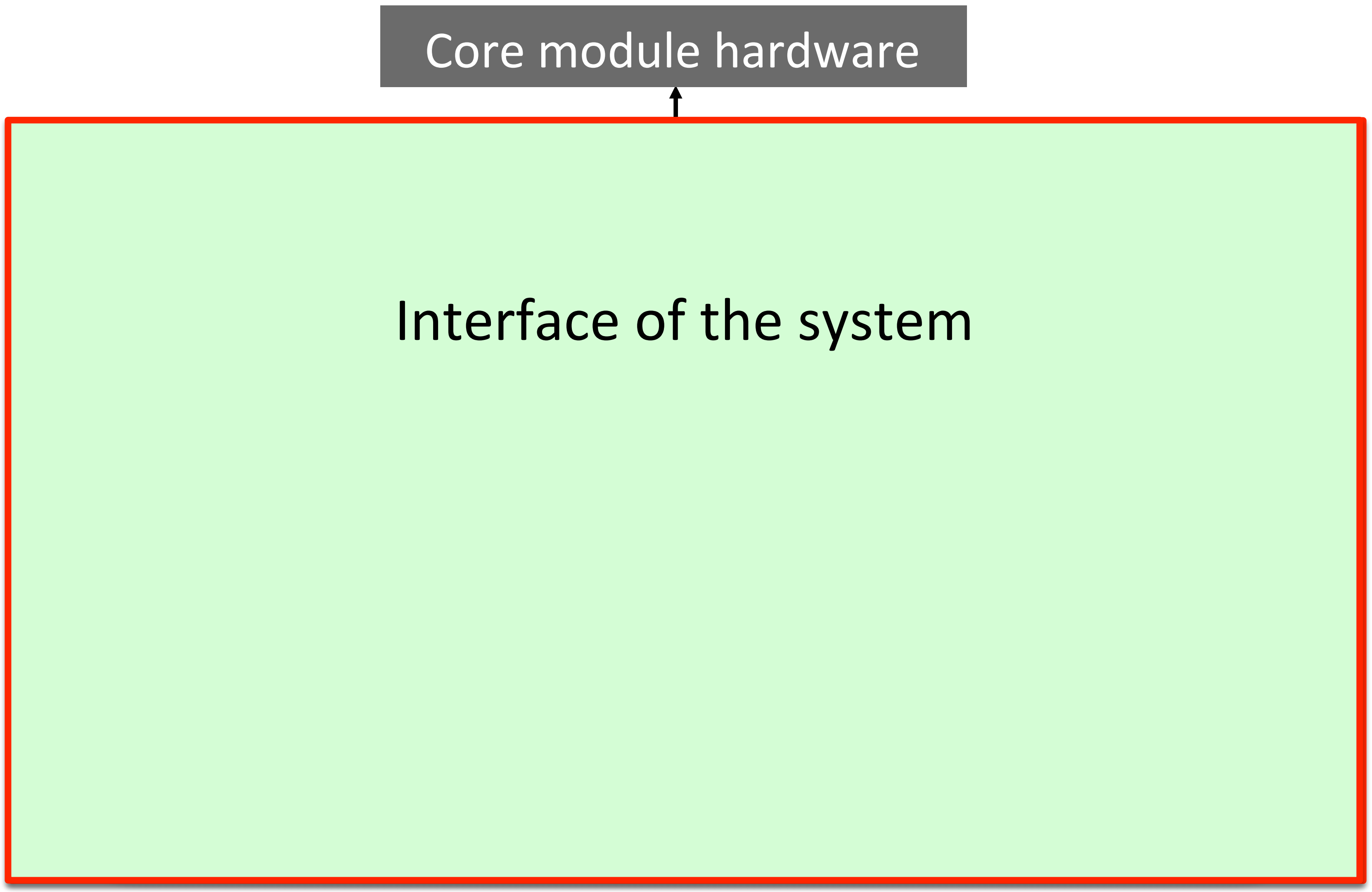


Compositional analysis overview

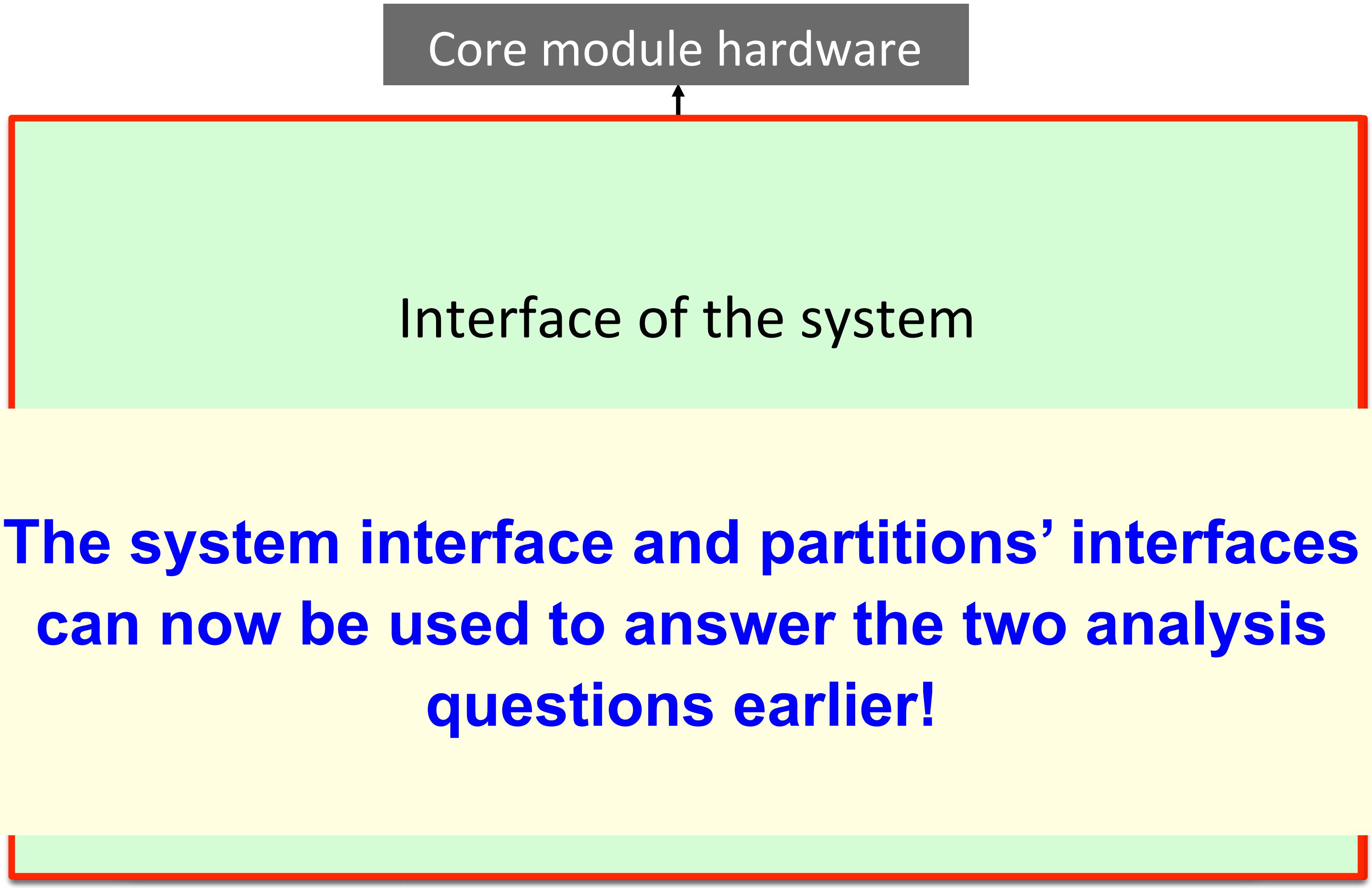


VCPU: (Period, Budget, Deadline)
VCPU: (Period, Budget), if Deadline = Period
VCPU Bandwidth = Budget/Period

Compositional analysis overview



Compositional analysis overview



The system interface and partitions' interfaces can now be used to answer the two analysis questions earlier!

State of the art

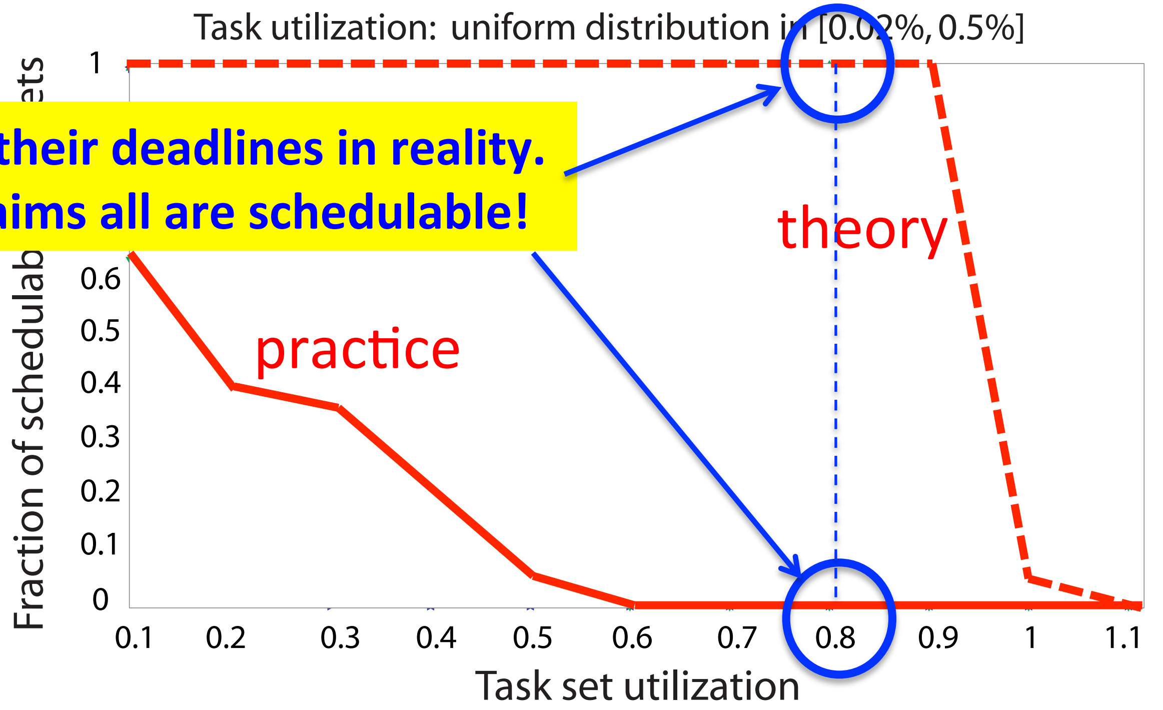
- Lots of existing work
  - a wide range of interface models and interface computation methods have been developed
    - see the paper for an incomplete list…
  - tools and implementations are available
    - e.g., CARTS, RTCToolbox, RT-Xen
- Many benefits
  - Enable efficient timing analysis of complex systems
  - Improve resource use efficiency
  - Can be used to perform resource dimensioning
  - Enable efficient integration and isolation of independently-developed cyber-physical components
- But…

We are not there yet!

- Existing theory has many limitations, e.g.,
  a) assumes unrealistic platforms, e.g., without overhead
  b) ignores semantics of interactions between the cyber and the physical aspects

- Result: Unsafe behavior!
  a) interfaces underestimate actual resource needs, leading to tasks missing their deadlines!
  b) undesirable component interactions via shared actuators
     - e.g., unintended simultaneous control of the steering shaft by the collision avoidance and lane centering control components

Existing compositional analysis



All tasks miss their deadlines in reality.
But, theory claims all are schedulable!

Fraction of schedulable task sets vs. workload utilization

Existing analysis: Unsafe!

Outline

- Introduction
- Challenges
  - Platform overhead
  - Data-dependent components
  - Scheduling theory vs. high-level formal models
  - Safety-aware interfaces
  - Clock synchronization
  - Analyzing state-based systems
- Conclusion

Outline

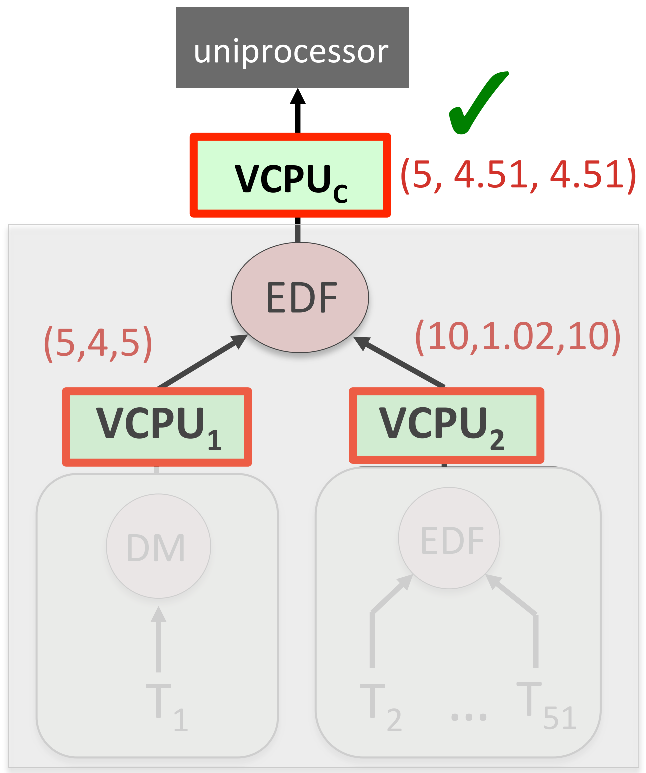
- Introduction
- **Challenges**
  - **Platform overhead**
  - Data-dependent components
  - Scheduling theory vs. high-level formal models
  - Safety-aware interfaces
  - Clock synchronization
  - Analyzing state-based systems
- Conclusion

Scenario #1: Task release delay

- Each job is released using an interrupt service routine (ISR)
- ISRs are typically serviced as soon as they are triggered
- Processing ISRs takes time!
  - e.g., up to 0.014ms to release a job on a Dell Optiplex-980 quad-core processor that runs LIMUS^{RT}
- Results:
  - Task execution will be delayed!
  - Overhead can accumulate if more jobs need to be released one after another

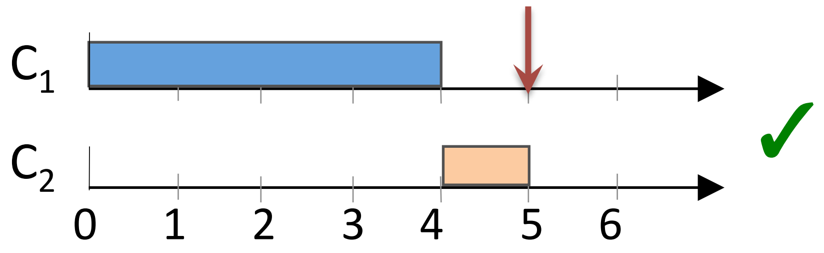
Scenario #1: Task release delay

- Existing theory: zero release delay

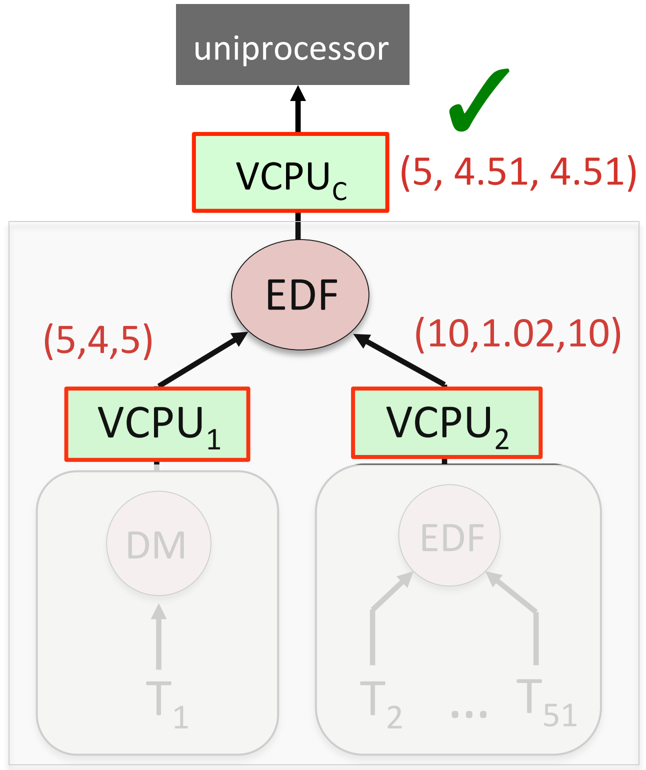


$T_1 = (5, 4, 5)$ $T_2 = (500, 1, 500)$

$T_2 = T_3 = \ldots = T_{51}$



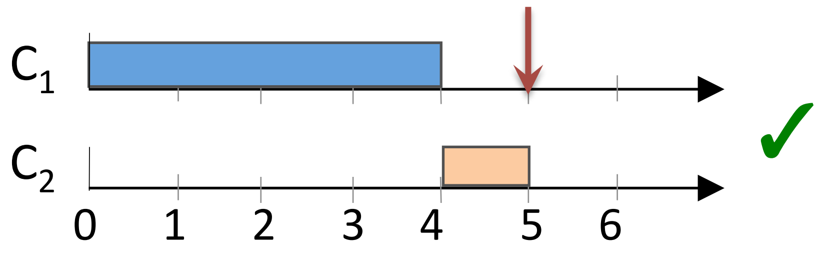
Scenario #1: Task release delay



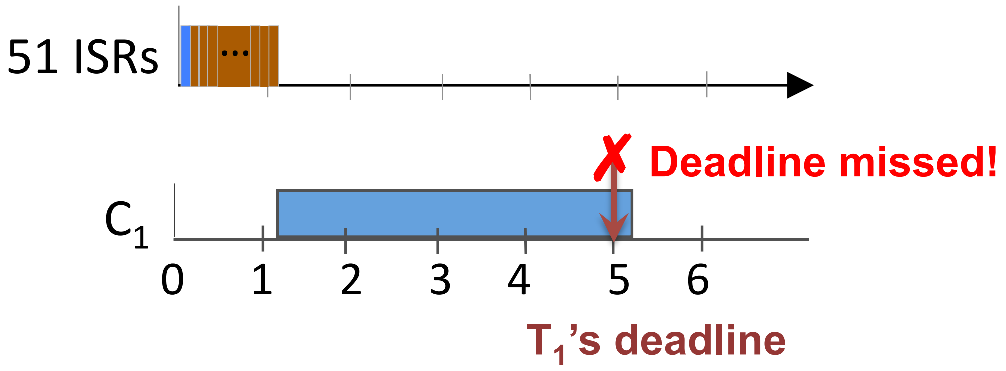
T_1= (5, 4, 5) **T_2 = (500, 1, 500)**

$T_2 = T_3 = \ldots = T_{51}$

- Existing theory: zero release delay

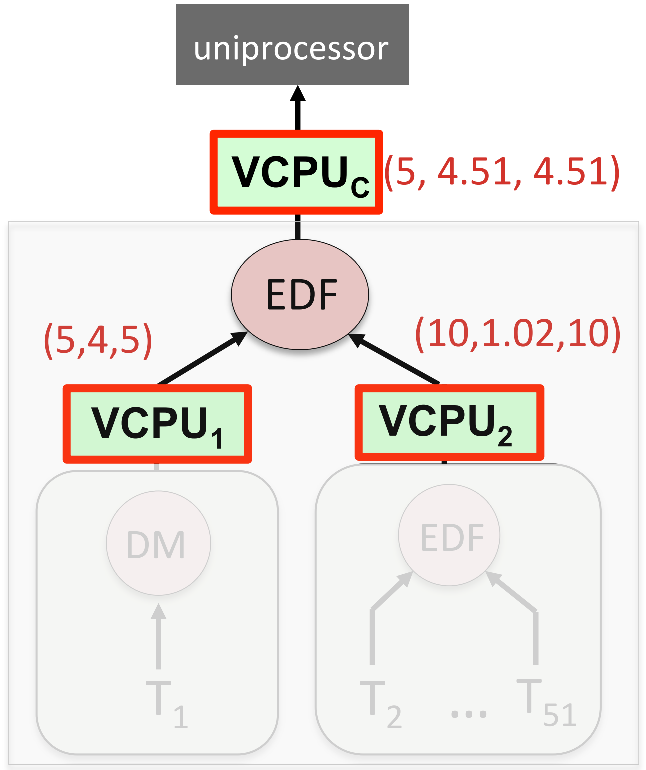


- In practice: release delay > 0
  - Each job is released using an ISR
  - When all tasks are released ε time units one after another: all 51 ISRs will be released first!



Strawman Solution #1:

Inflate task WCET with an ISR



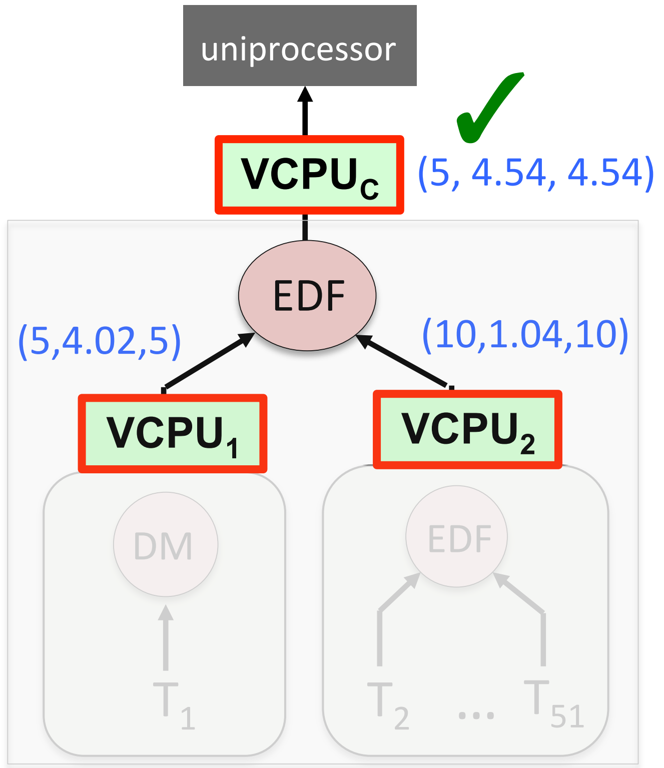
$T_1 = (5, 4, 5)$ $T_2 = (500, 1, 500)$

$T_2 = T_3 = \ldots = T_{51}$

Time unit: ms

$e' = e + \Delta^{rel}$

$\Delta^{rel} = 0.02$



$T'_1 = (5, 4.02, 5)$ $T_2 = (500, 1.02, 500)$

$T_2 = T_3 = \ldots = T_{51}$

Strawman Solution #1:

Inflate task WCET with an ISR

Unsafe!

Schedulable in theory but not in practice

T_1= (5, 4, 5) T_2 = (500, 1, 500)

$T_2 = T_3 = \ldots = T_{51}$

Time unit: ms

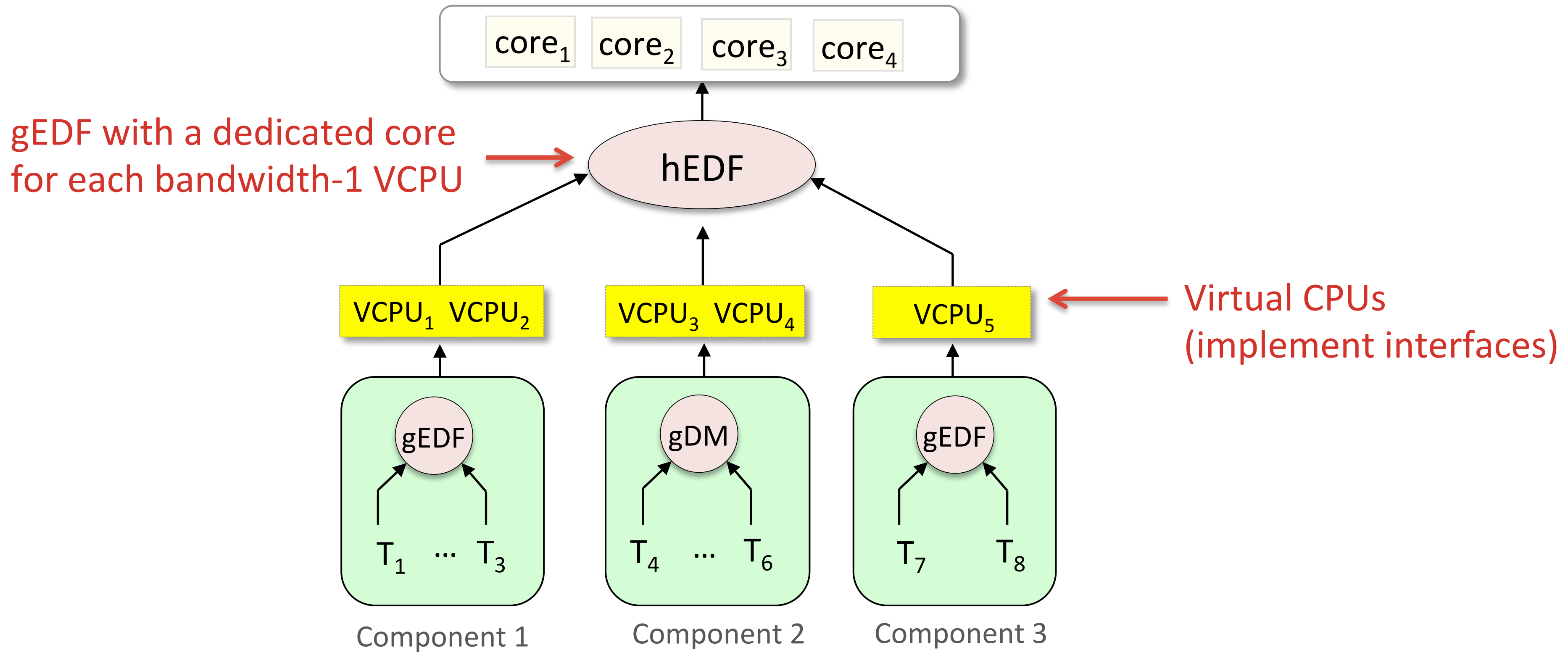
T'_1= (5, 4.02, 5) T_2 = (500, 1.02, 500)

$T_2 = T_3 = \ldots = T_{51}$

Strawman Solution #2:
Inflate task WCET with all ISRs?

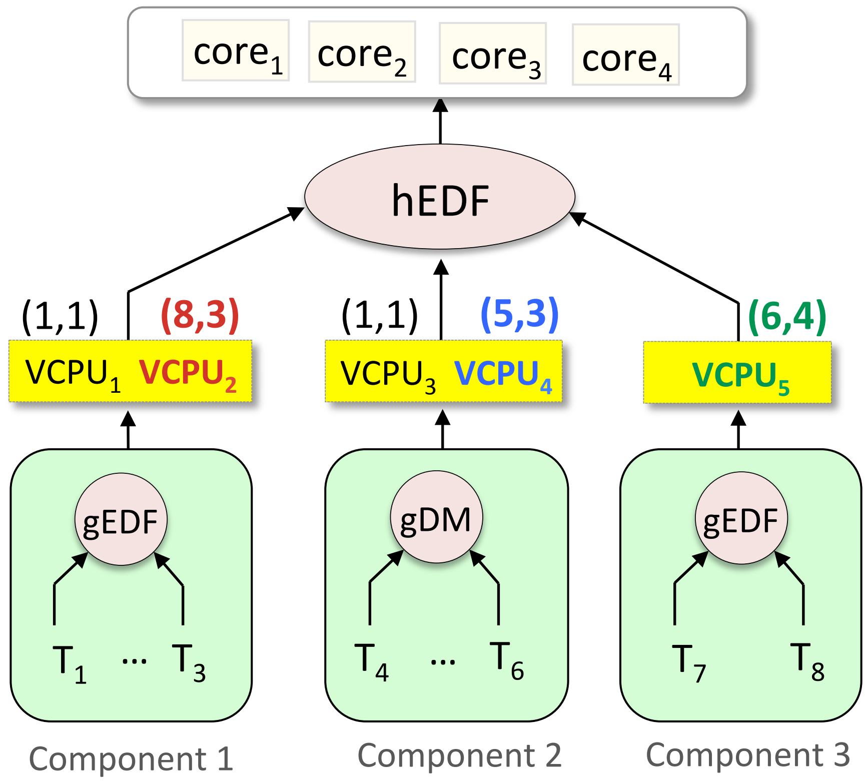
- $e' = e + n\Delta^{rel}$
  - n: number of tasks in the whole system
- Safe, but…
- **Impossible to obtain n**
  - The task information within one component is hidden from another component
- Also, overly pessimistic

Scenario #2: Cache interference

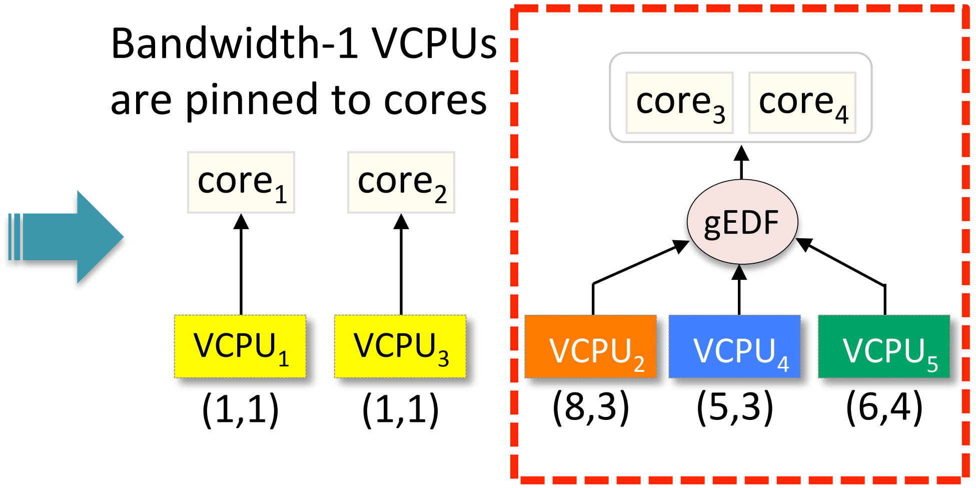


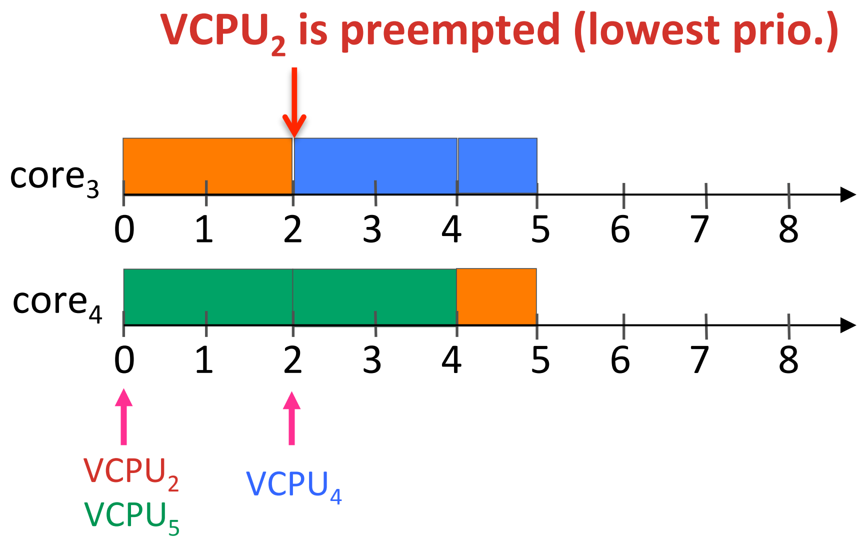
- Overhead due to cache misses depends on not only the interference between tasks but also the interference between VCPUs and between VCPUs and tasks

Example: VCPU preemption

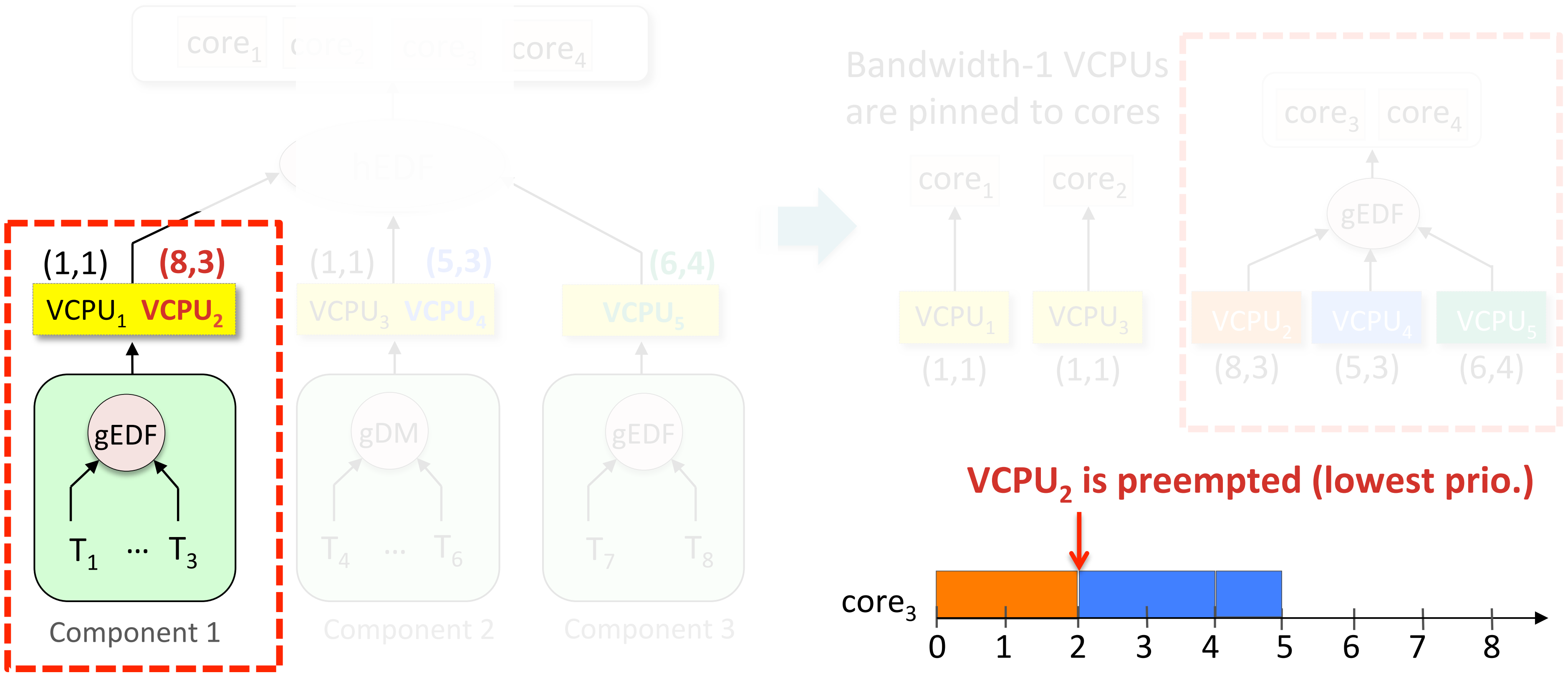


$T_1 = (8,4)$
$T_2 = (6,2)$
$T_3 = (10,1.5)$

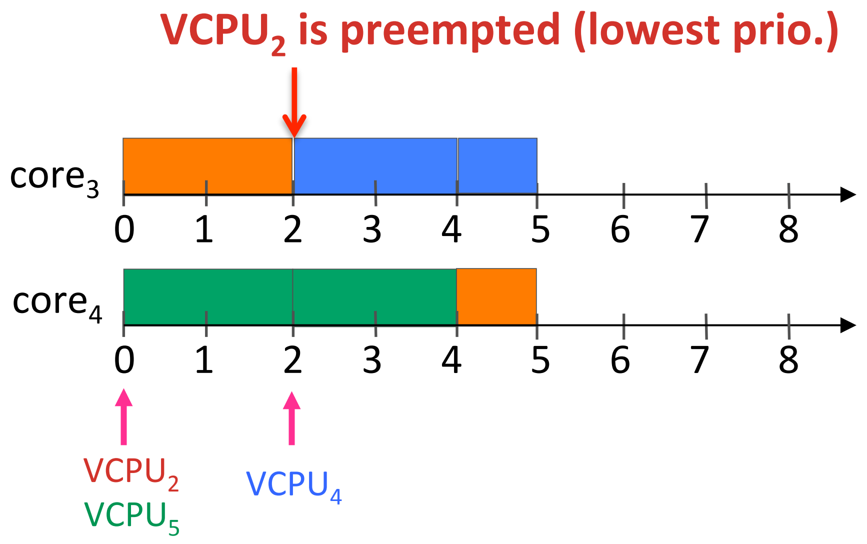




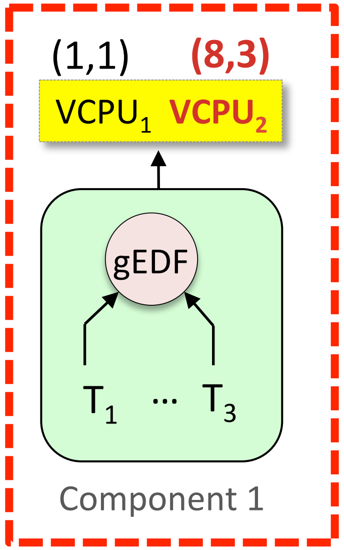
Example: VCPU preemption



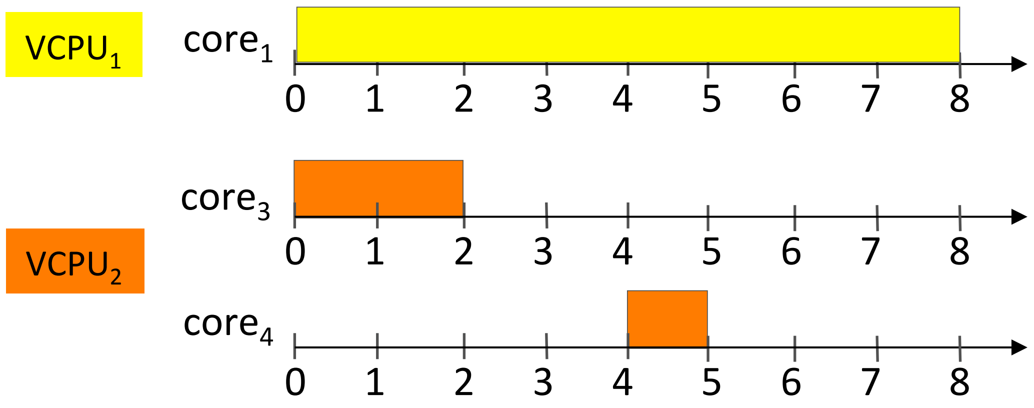
$T_1 = (8,4)$
$T_2 = (6,2)$
$T_3 = (10,1.5)$



Example: VCPU preemption

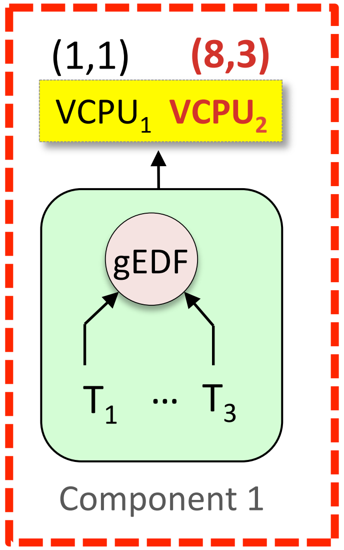


$T_1 = (8,4)$
$T_2 = (6,2)$
$T_3 = (10,1.5)$



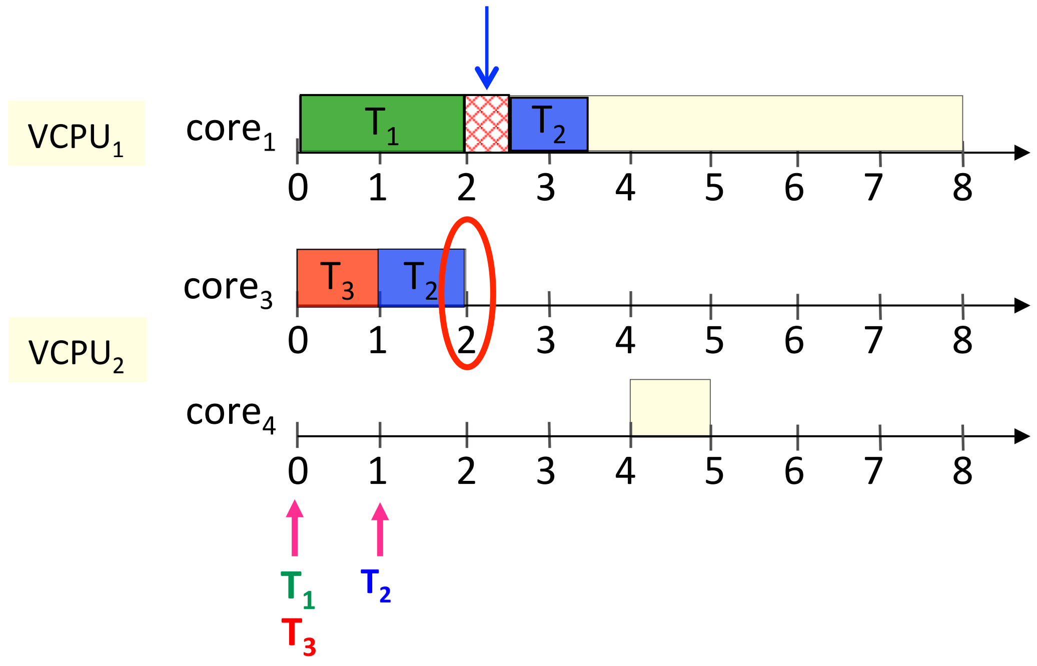
Example: VCPU preemption

VCPU$_2$ is preempted and becomes unavailable
→ T$_2$ migrates to VCPU$_1$ (core 1) and preempts the lower-priority task T$_1$



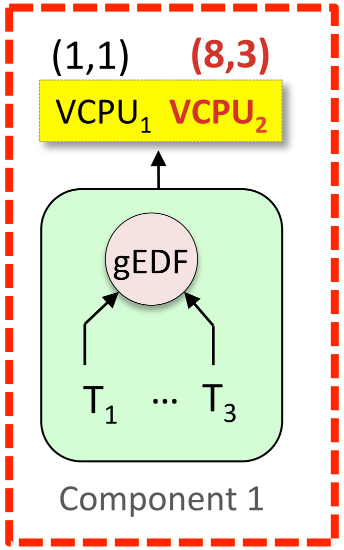
T$_1$ = (8,4)	🟩
T$_2$ = (6,2)	🟦
T$_3$ = (10,1.5)	🟥

T$_2$ experiences migration overhead due to VCPU preemption



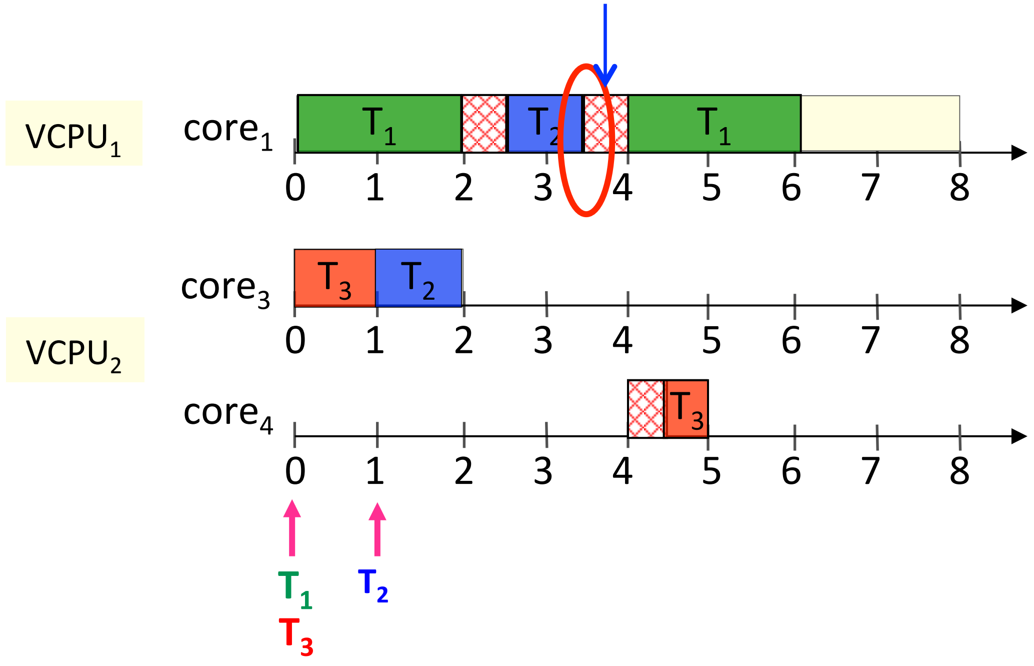
Example: VCPU preemption

T_2 finishes → T_1 resumes



$T_1 = (8,4)$
$T_2 = (6,2)$
$T_3 = (10,1.5)$

T_1 experiences preemption overhead due to VCPU preemption



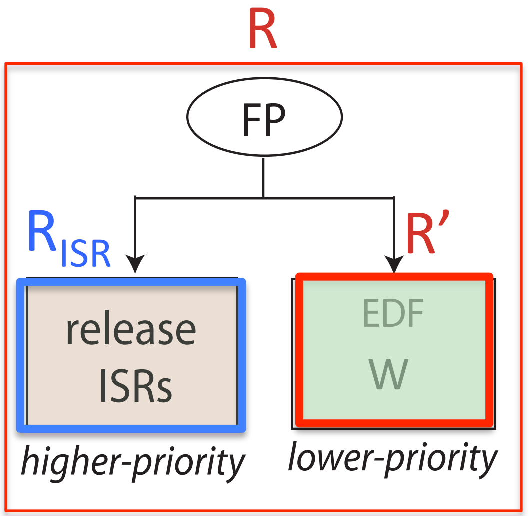
Approach: Overhead-aware analysis

- Incorporate platform overhead into components' interfaces and schedulability test
- **Inflatable overhead**
  - Examples: schedule function, context switch, tick, cache miss due to intra-component task preemption/migration
  - Accounted by inflating each task's WCET
- **Non-inflatable overhead**
  - Examples: release ISR delay, cache-related overhead due to VCPU preemption or completion
  - **Expose the combined overhead experienced by a component on its interface**

Release ISR overhead accounting

[RTAS'13]

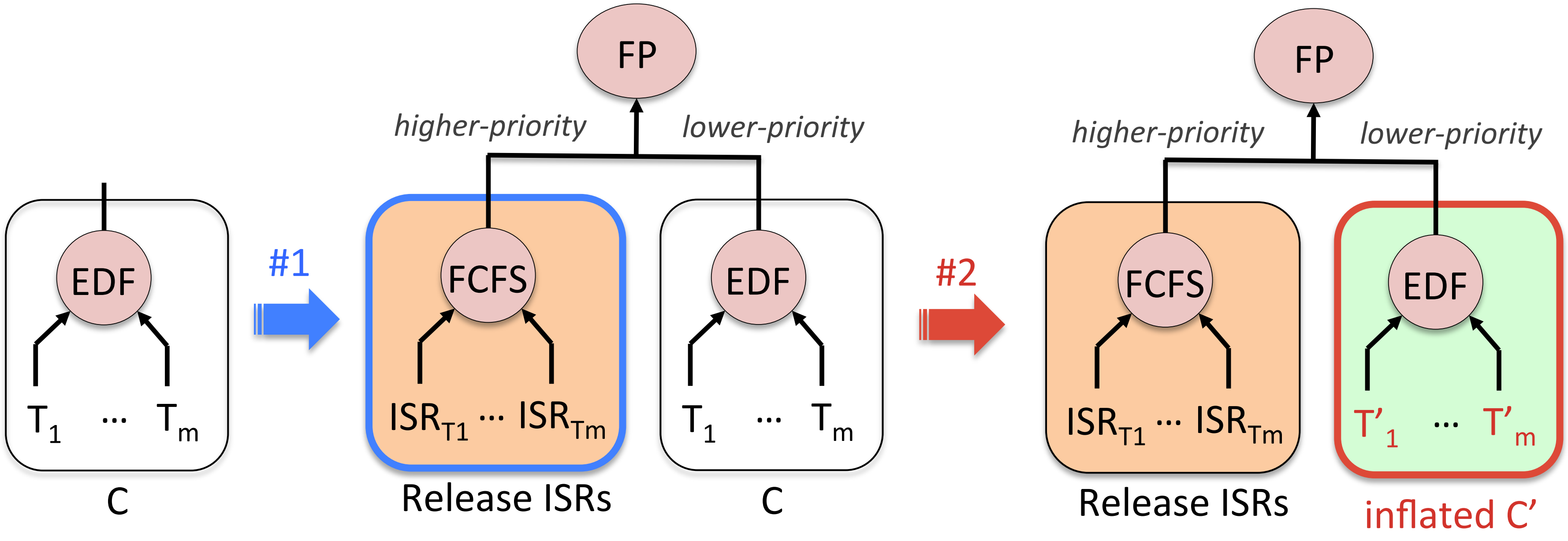
- Approach: Model overhead caused by ISRs
  - Using a compositional scheduling analogy
  - ISRs: higher-priority intra-component
  - Workload: lower-priority component
  - Scheduled under Fixed Priority



- Intuition:
  - The effective resource given to the tasks is the remaining resource after processing the higher-priority release ISRs component

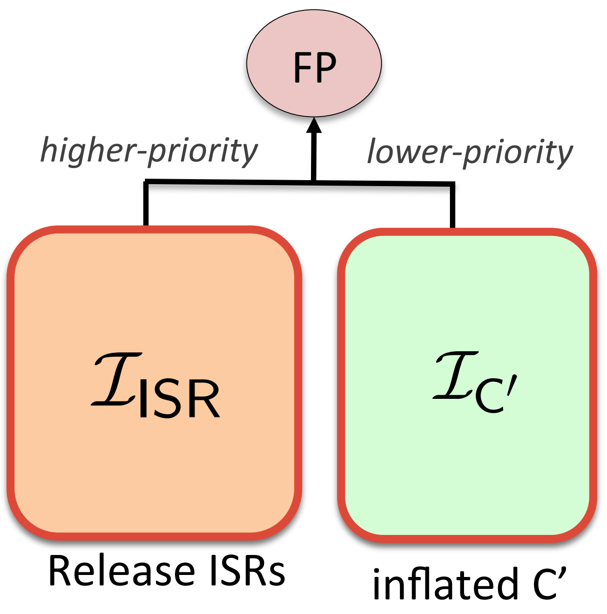
Overhead-aware comp. analysis

- Primitive component transformation
  - Step #1: Add a higher-priority release ISRs component
  - Step #2: Inflate the WCETs of tasks with inflatable overhead



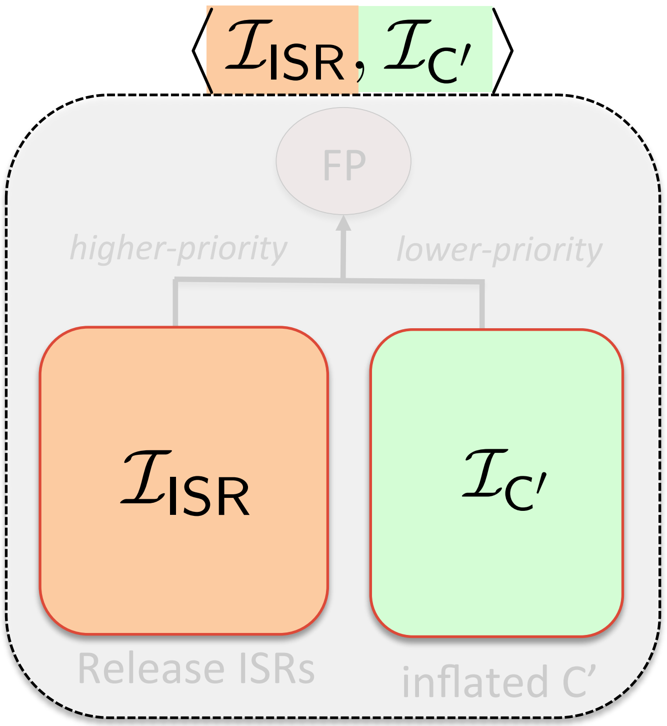
Overhead-aware comp. analysis

- Primitive component transformation
- Interface abstraction
  - Abstract each component into an interface



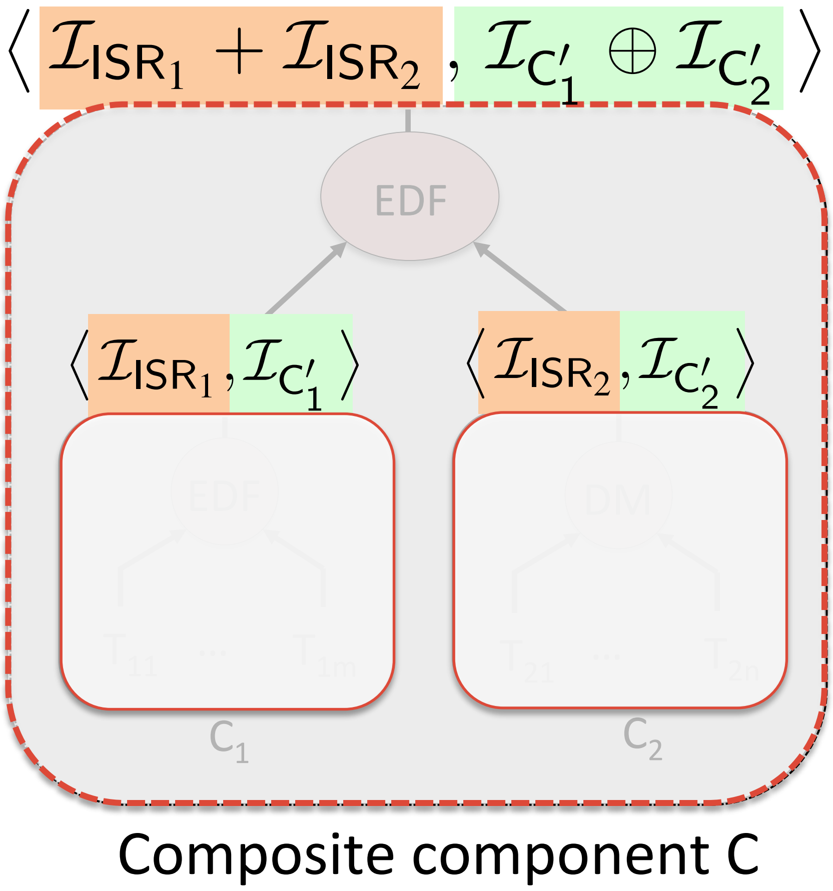
Overhead-aware comp. analysis

- Primitive component transformation
- Interface abstraction
  - Abstract each component into an interface
  - Overhead-aware interface of C:


$\langle \mathcal{I}_{\text{ISR}}, \mathcal{I}_{C'} \rangle$
FP
higher-priority
lower-priority
$\mathcal{I}_{\text{ISR}}$
$\mathcal{I}_{C'}$
Release ISRs
inflated C'

Overhead-aware comp. analysis

- Primitive component transformation
- Interface abstraction
- Interface composition


$\langle \mathcal{I}_{\mathsf{ISR}_1} + \mathcal{I}_{\mathsf{ISR}_2}, \mathcal{I}_{C'_1} \oplus \mathcal{I}_{C'_2} \rangle$
EDF
$\langle \mathcal{I}_{\mathsf{ISR}_1}, \mathcal{I}_{C'_1} \rangle$
$\langle \mathcal{I}_{\mathsf{ISR}_2}, \mathcal{I}_{C'_2} \rangle$
EDF
T_{11} … T_{1m}
DM
T_{21} … T_{2n}
C_1
C_2
Composite component C

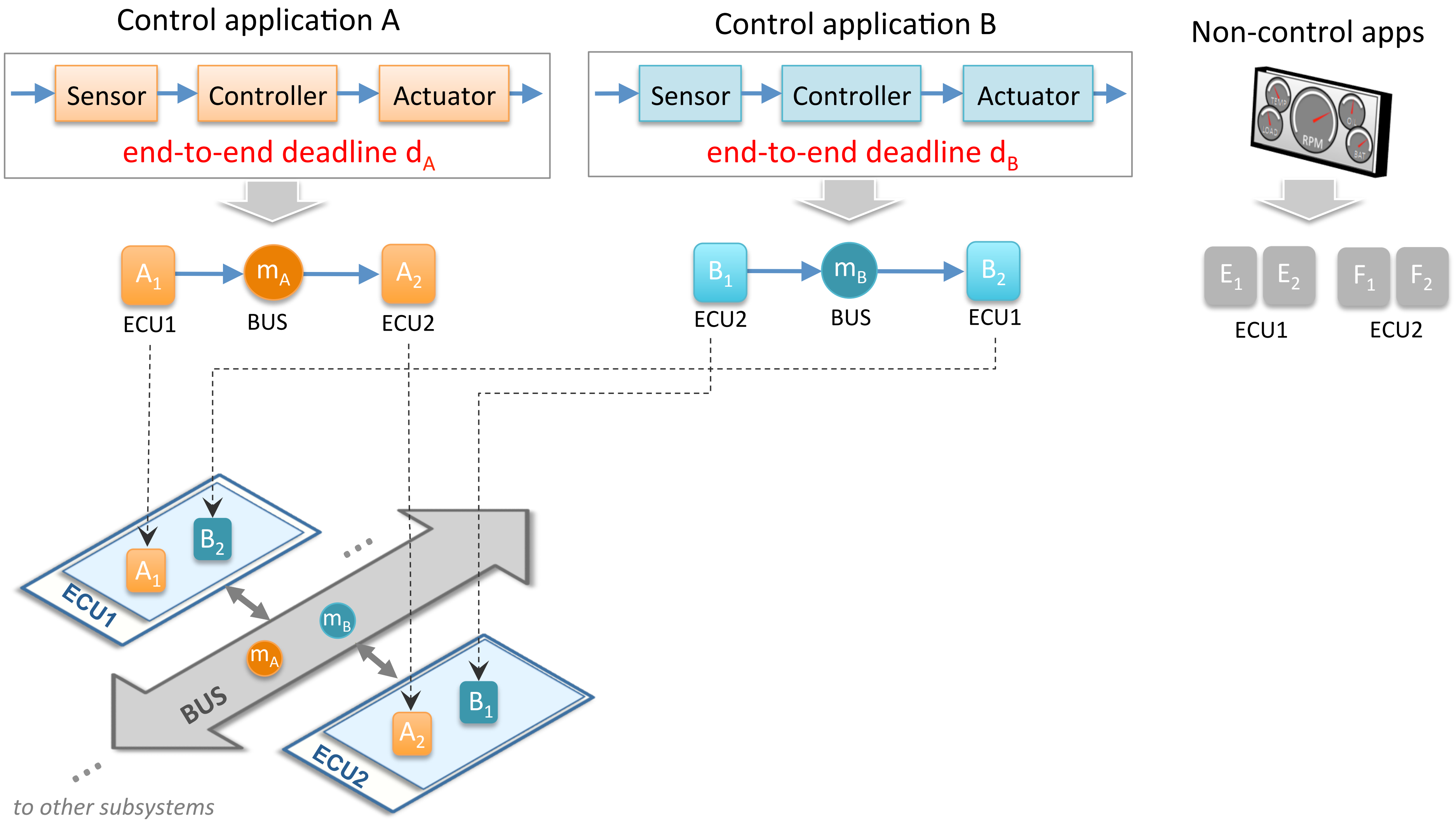
Overhead-aware compositional analysis on multi-core

- Cache-aware: Tomorrow's talk!

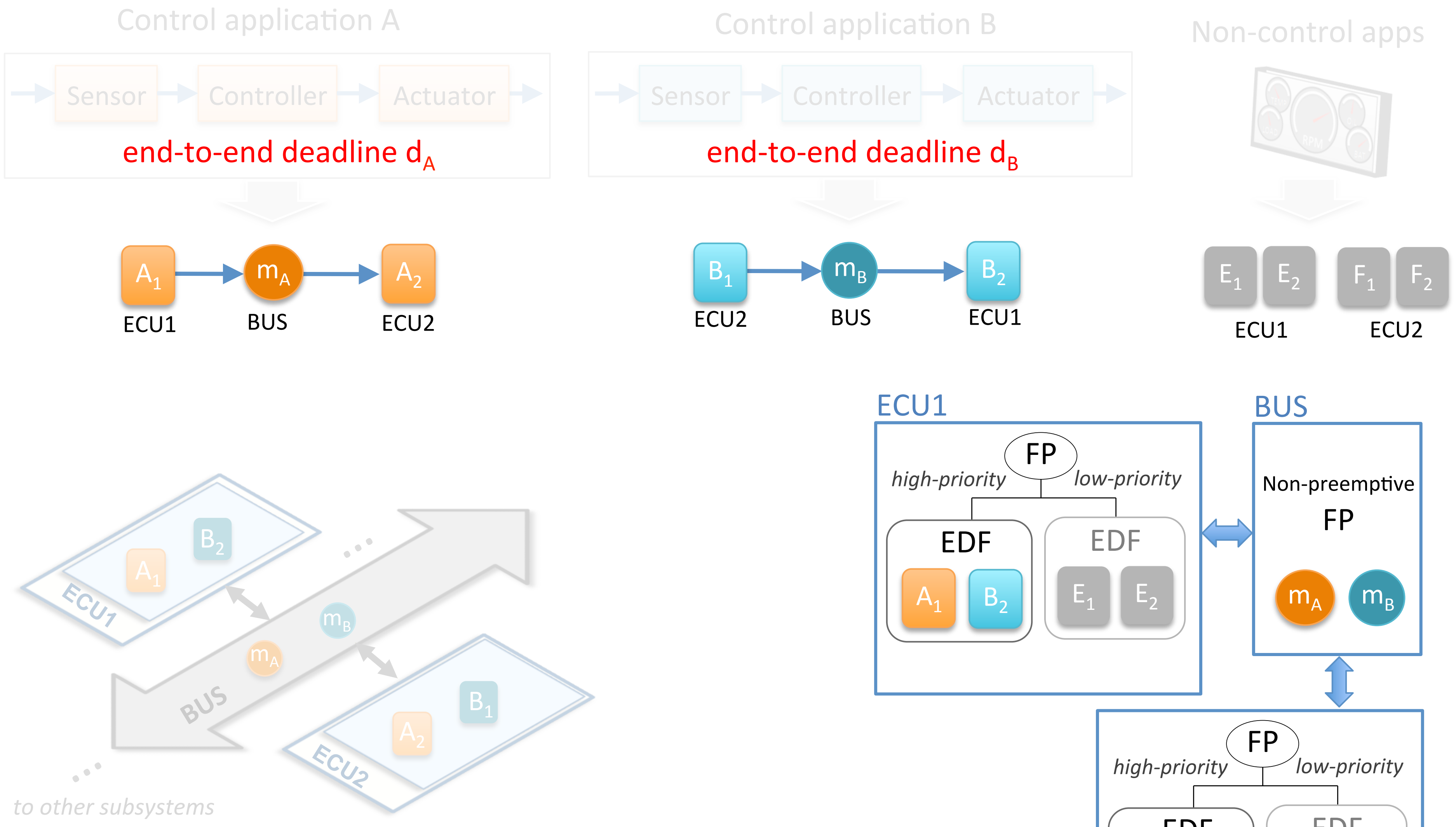
- Open questions
  - Complex cache hierarchy with shared cache
  - Improvement of schedulability analysis under multicore scheduling (e.g., global EDF)
  - Global optimal interfaces
  - Cache control to reduce interference
  - …

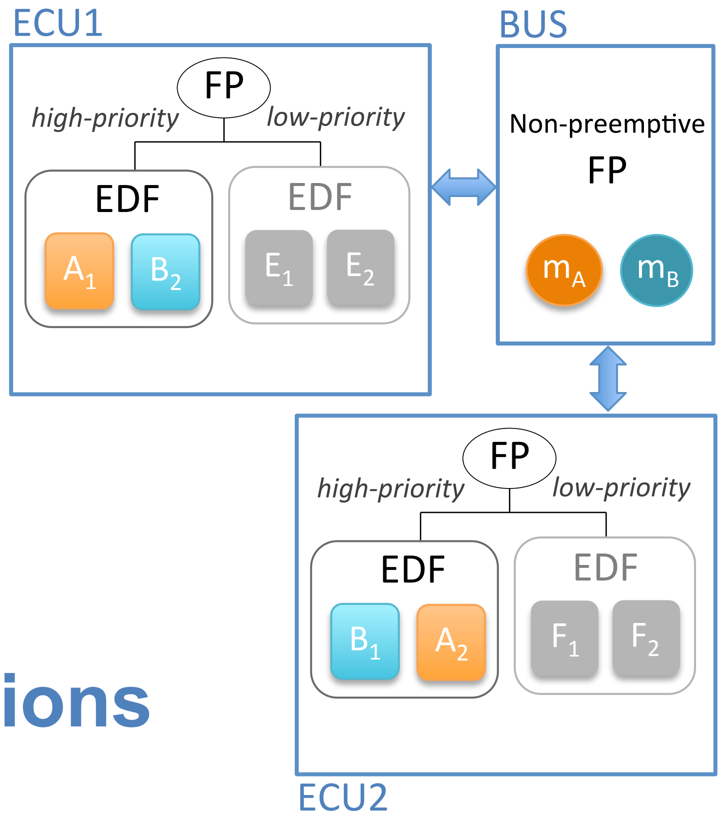
Outline

- Introduction
- **Challenges**
  - Platform overhead
  - **Data-dependent components**
  - Scheduling theory vs. high-level formal models
  - Safety-aware interfaces
  - Clock synchronization
  - Analyzing state-based systems
- Conclusion



Scenario: Automotive applications



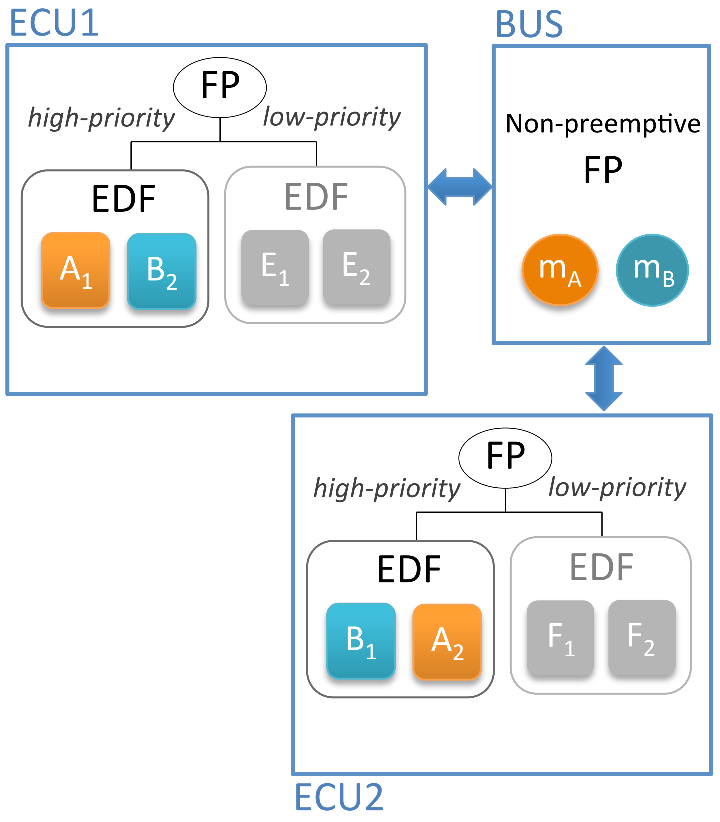


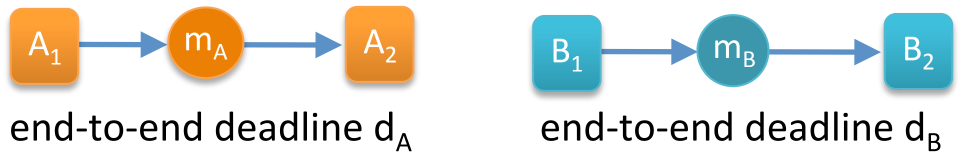
Scenario: Automotive applications

Existing compositional theory

- Assume all tasks are independent
- Assume the deadline of each task is given
- Result:

  Cannot be directly applied!





Challenge #1: Deadline correlation

Larger deadline for A_1 $\Rightarrow$ Smaller C1's interface bandwidth

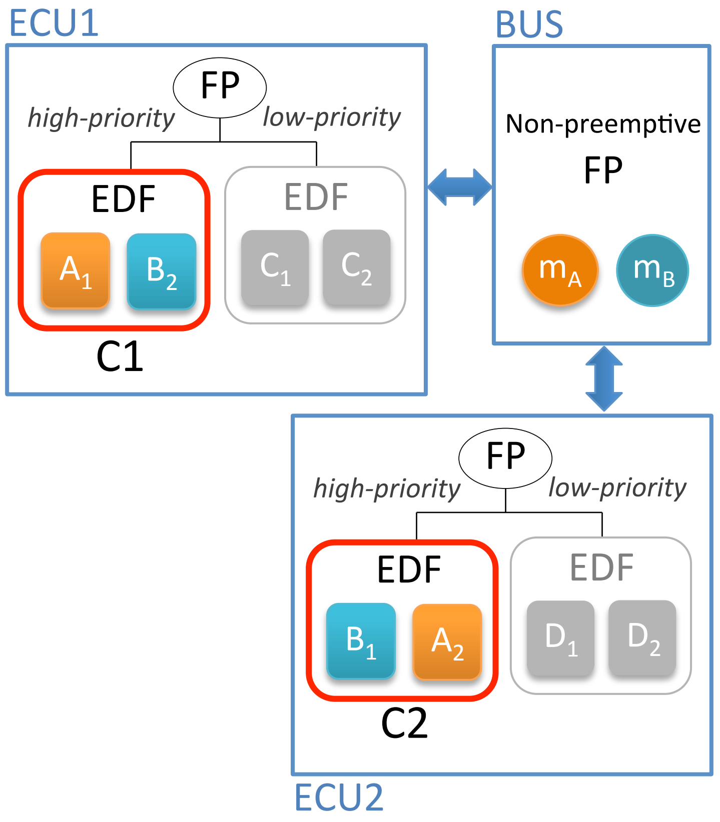
Smaller ECU1's frequency

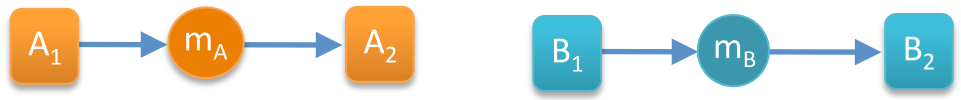
$\Downarrow$

Smaller deadline for A_2

Larger C2's interface bandwidth

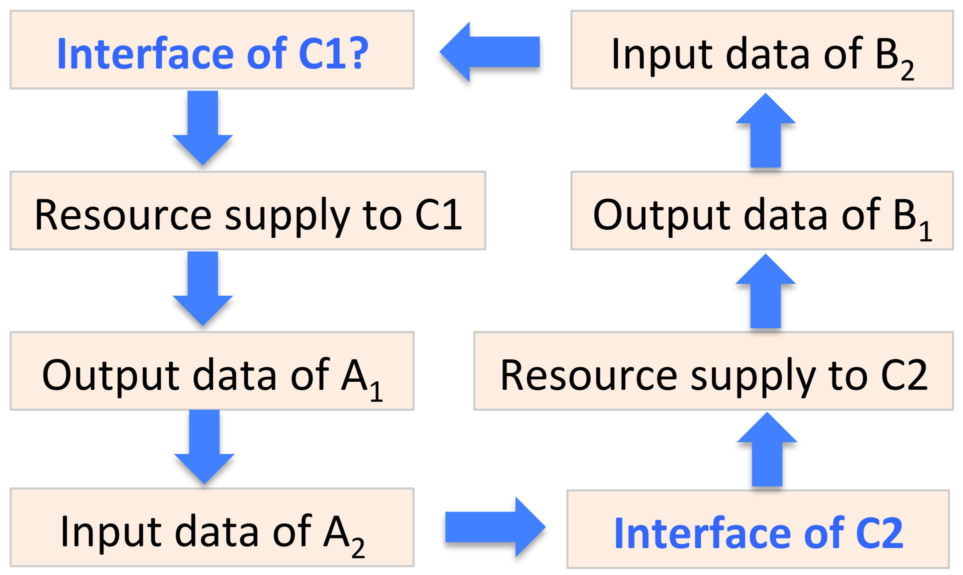
Larger ECU2's frequency

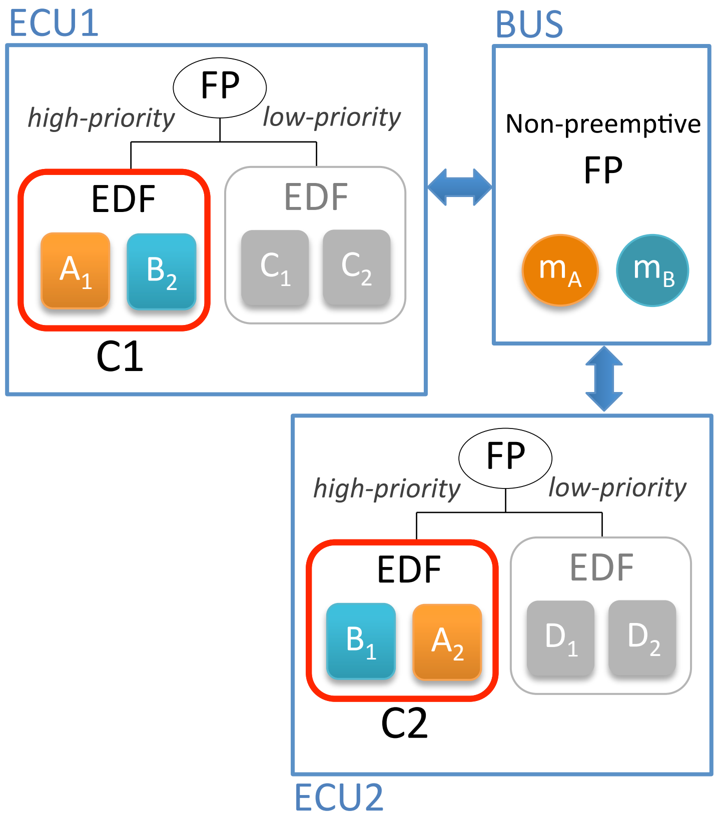


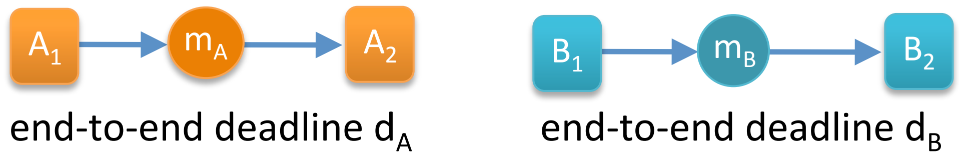


end-to-end deadline d_A end-to-end deadline d_B

Challenge #2: Cyclic dependency







Approach: Deadline decomposition

- Advantages: Existing compositional theory can be used
  - Existing deadline decomposition methods need to be extended!
- What is a meaningful notion of interface optimality?
  - **Idea**: use a partial order of resource use
- How to capture interactions between I/O composition via data dependence and hierarchical composition via scheduling?
  - **Idea**: combine assume/guarantee interfaces with compositional schedulability analysis
- How to tackle complexity due to data dependence?
  - **Idea**: transform arrival patterns of input/output data to restricted forms (e.g., periodic) while preserving end-to-end timing properties
  - **Approach**: adapt synchronization protocols and/or shaping techniques

Approach: Parametric interfaces

- Basic idea
  - Interface: a function of variables representing unknown tasks' parameters (e.g., local deadlines)
  - Symbolic computation of interfaces
  - Concrete interfaces are realized at the top level based on end-to-end timing constraints
- Advantage: accuracy!
- Challenge: the size of composed interface grows with more composition steps
  - **Idea:** refine intermediate interfaces based on safe approximations of the functions

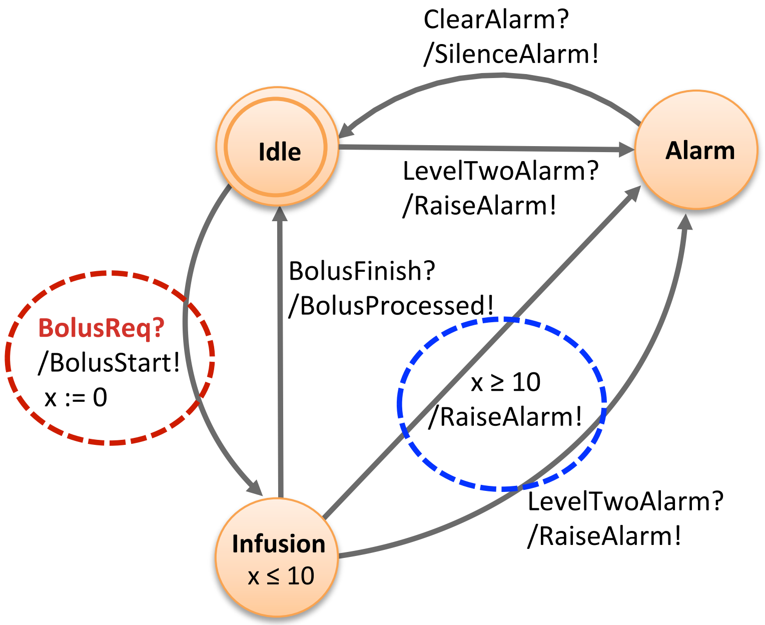
Outline

- Introduction
- **Challenges**
  - Platform overhead
  - Data-dependent components
  - **Scheduling theory vs. high-level models**
  - Safety-aware interfaces
  - Clock synchronization
  - Analyzing state-based systems
- Conclusion

The modeling gap

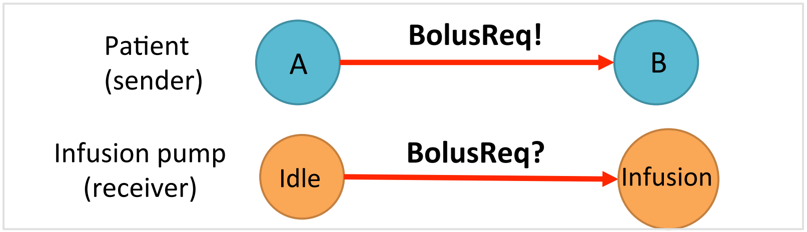
- High-level models of computation
  - Ex: timed automata, I/O automata, process algebra
  - Focus on high-level specifications of timed interactions, communications, synchronization
  - Ignore platform aspects, e.g., communication/scheduling delay
- Real-time task/resource models
  - Ex: periodic, concurrent task models, Real-Time Calculus
  - Focus on implementation-level information, e.g., execution time, deadline, resource sharing semantics
  - Do not consider high-level semantics, e.g., synchronization, time-dependent behavior

Example: Infusion pump



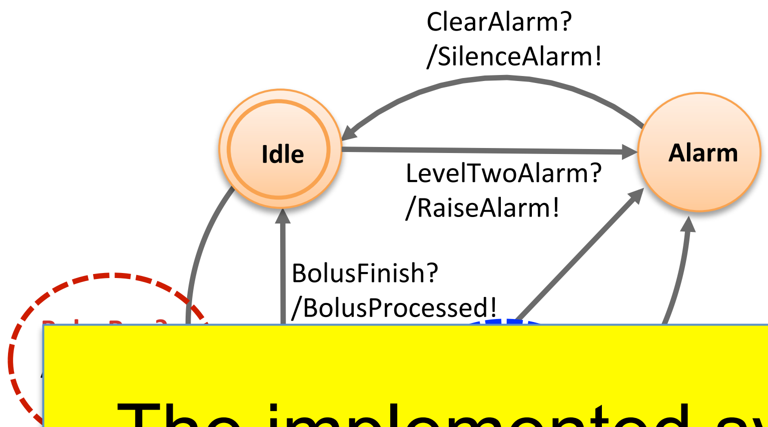
(partial) timed automation model of infusion pump software

- Abstract model
  - captures the software behavior independently of the target platform
  - makes implicit assumptions, e.g.,
    - synchronous communication is instantaneous
    - processing takes zero time



- Platform and task models
  - ignores time-dependent behavior
    - e.g., alarm raised after 10 time units of infusion

Example: Infusion pump



- Abstract model
  - captures the software behavior independently of the target platform
  - makes implicit assumptions, e.g.,
    - synchronous communication is instantaneous

The implemented system is unsafe, even if

- system properties are verified at the high-level model
- the system is schedulable at the platform level

- Platform and task models
  - ignores time-dependent behavior
    - e.g., alarm raised after 10 time units of infusion

LifeCare PCA

Bridging the gap

- Several existing approaches can serve as basis, e.g.,
  - Adding platform aspects to high-level models
    - e.g., new timed-automata semantics (almost ASAP, time-triggered, sampling-based, probabilistic, etc.)
  - Combining scheduling with high-level models
    - e.g., TIMES tool, resource-based process algebra
  - Automata- and actor-oriented scheduling interfaces
- However, existing work is expensive in timing analysis and assumes very simplistic platforms
  - Need more efficient approaches!
- Idea: use a glue-layer that connects both classes of models
  - Captures assumptions high-level models make about the platform
  - Can be used to mechanically verify that a given platform satisfies the assumptions
  - Our very initial work: RSP'12, CASES'13

Outline

- Introduction
- **Challenges**
  - Platform overhead
  - Data-dependent components
  - Scheduling theory vs. high-level formal models
  - **Safety-aware interfaces**
  - Clock synchronization
  - Analyzing state-based systems
- Conclusion

Conclusion

- Cyber-physical systems are increasingly complex
  - Need accurate and scalable analysis and design
- Compositional approach is an effective method
  - Can handle complexity
  - Can help optimize resources
- Many interesting open challenges remain
  - **This talk:** some important challenges towards a safe compositional scheduling and timing analysis
  - Research opportunities for you!!

linhphan@cis.upenn.edu