ACM SIGBED International Conference on Embedded Software (EMSOFT 2025)

Rasco: Resource Allocation and Scheduling Co-design for DAG Applications on Multicore

Abigail Eisenklam, Robert Gifford, Georgiy A. Bondar*, Yifan Cai, Tushar Sial[T], Linh Thi Xuan Phan, Abhishek Halder[T]*

Penn University of Pennsylvania

UC SANTA CRUZ*

IOWA STATE UNIVERSITY[T]

The Rise of Data Intensive CPS Tasks

- Tasks in real-time, embedded, and CPS are increasingly **data intensive**
- Autonomous control, image processing, signal processing, etc.

Joby Acquires Xwing Autonomy Division, Looks Ahead to Autonomous Flight

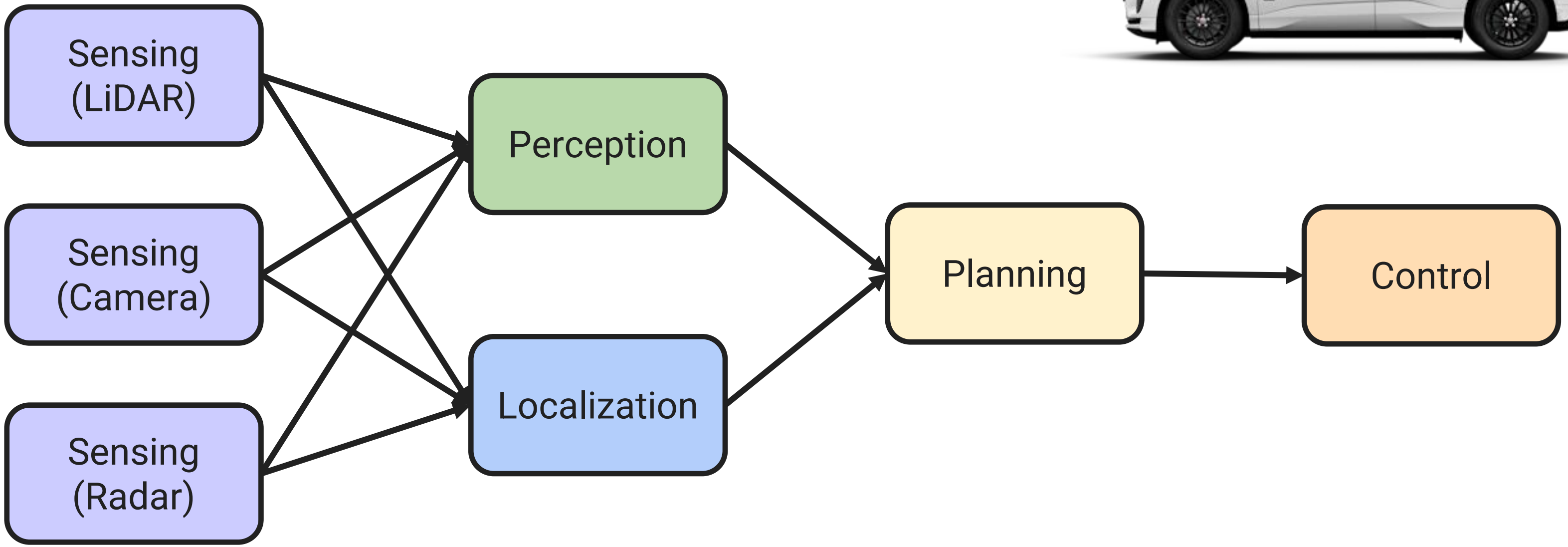
June 04, 2024 7:00am EDT

Download as PDF

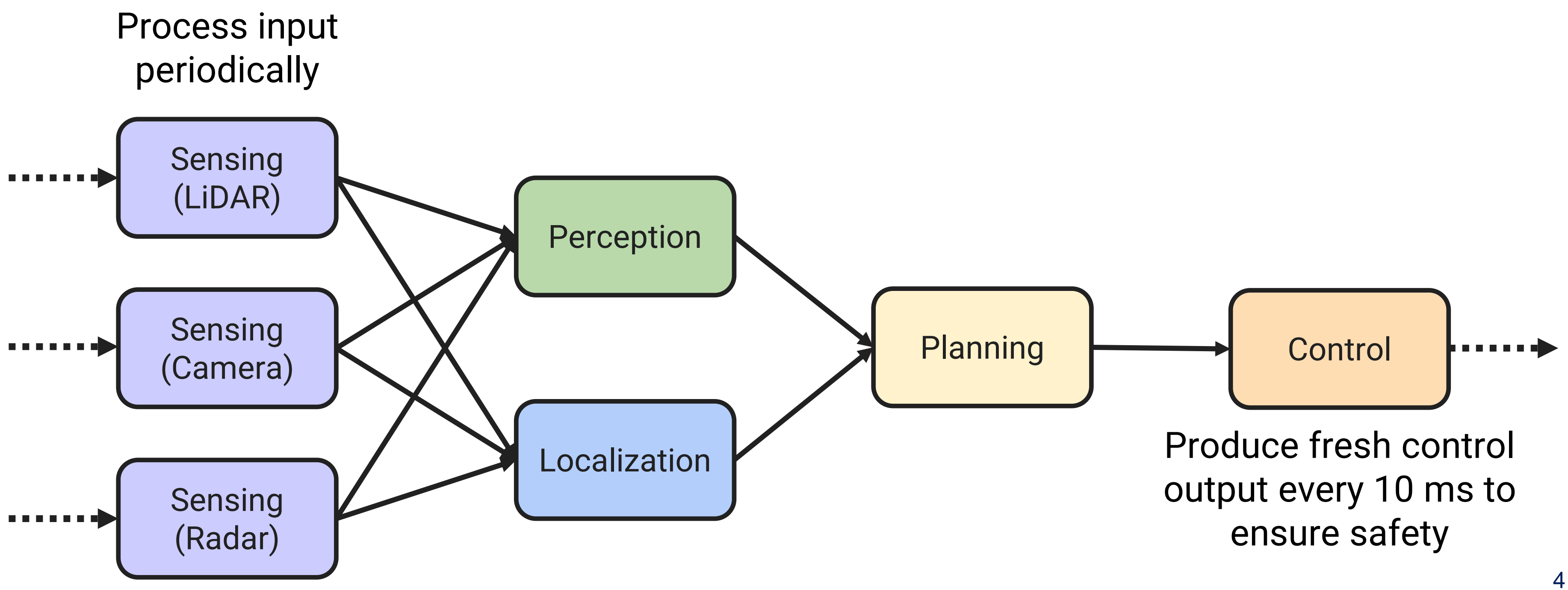
https://www.jobyaviation.com/

Data Dependencies in CPS

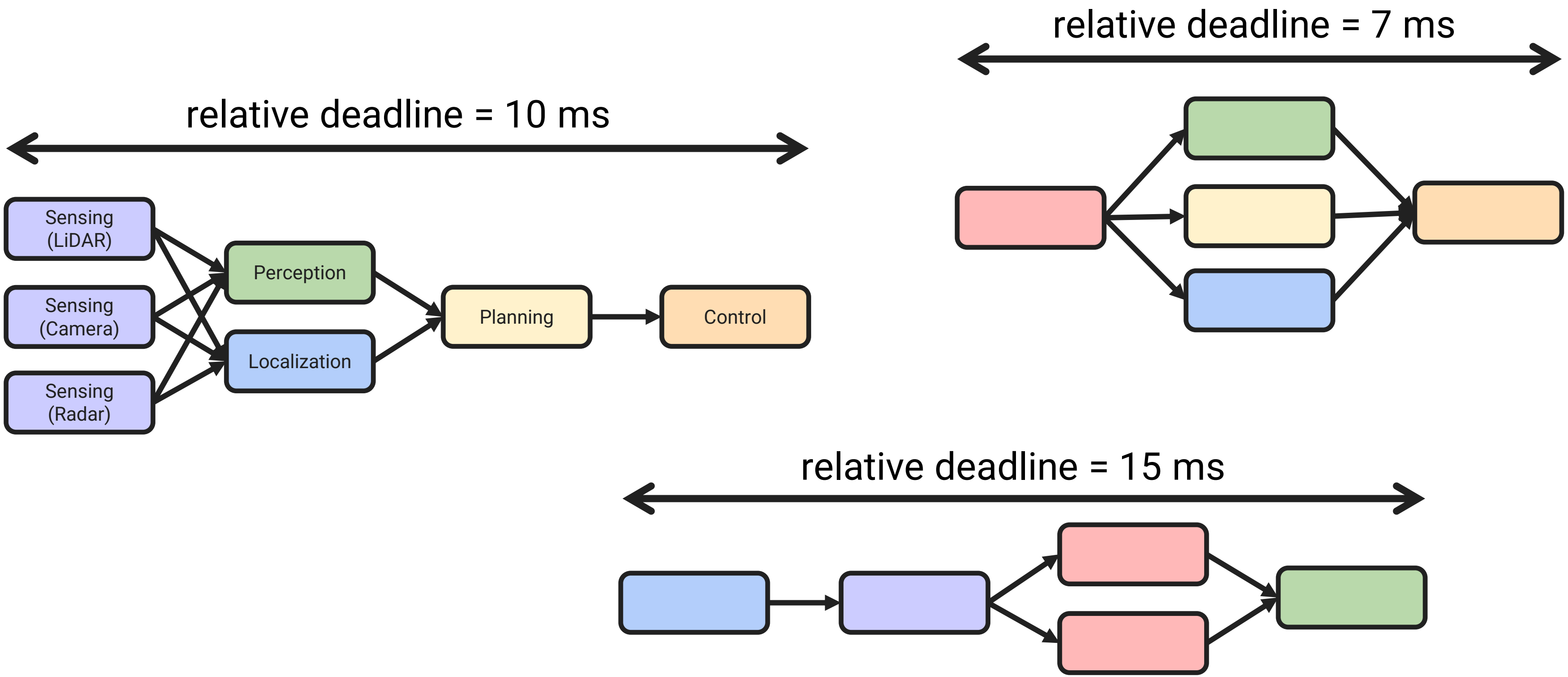
- CPS tasks have data dependencies
- Example: Autoware pipeline



Data Dependencies in CPS

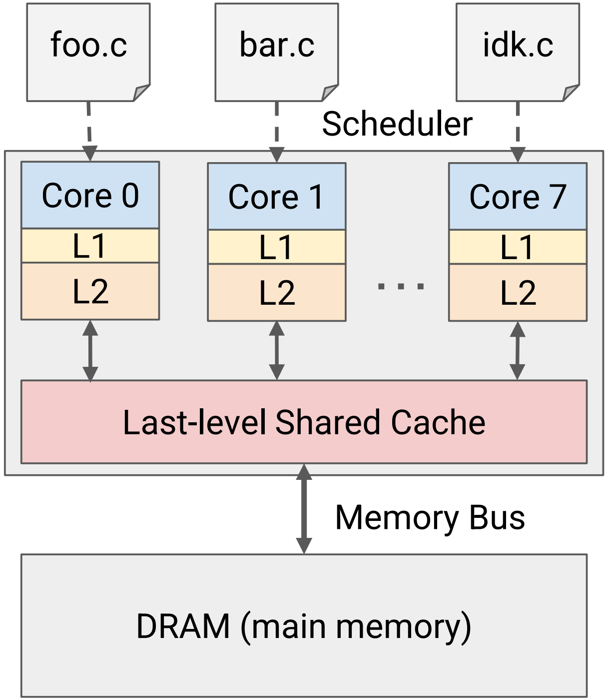


Model: Multiple Periodic DAG Tasks w/ Implicit Deadlines



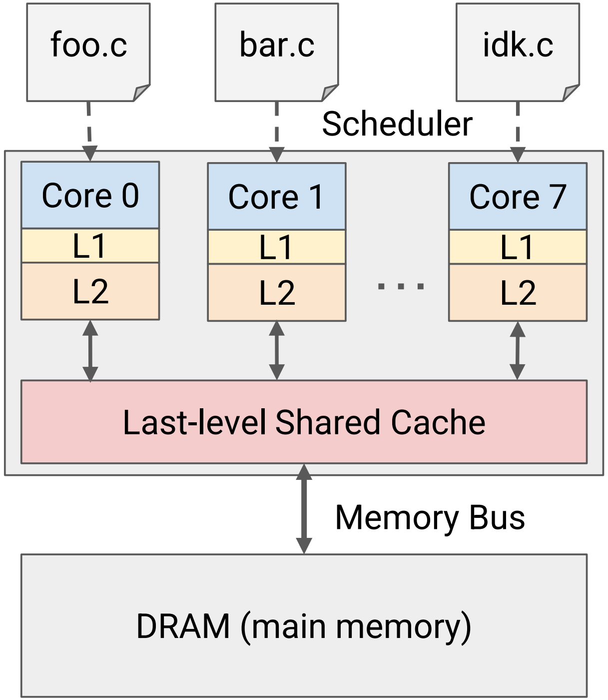
Leveraging Multicore Hardware in CPS

- Multicores exploit inter- and intra-DAG parallelism
  - Lower DAG latency
  - Higher system throughput



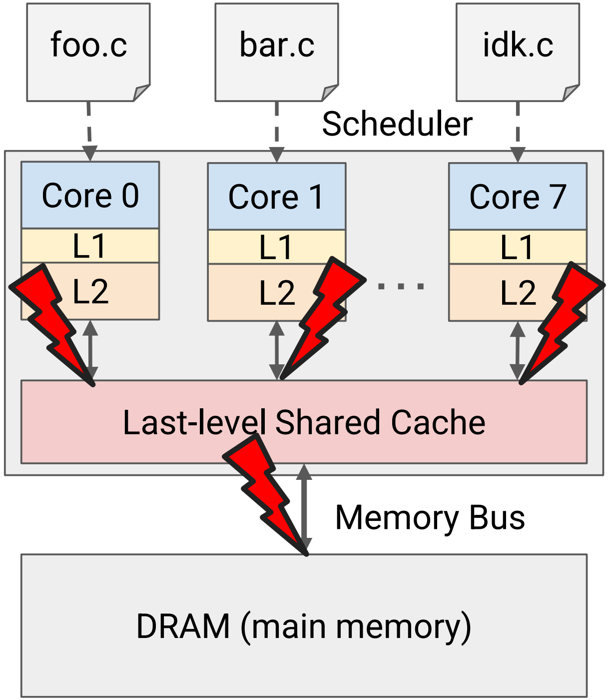
Leveraging Multicore Hardware in CPS

- Multicores exploit inter- and intra-DAG parallelism
  - Lower DAG latency
  - Higher system throughput
- Shared resources such as last-level cache and memory bandwidth are statistically multiplexed
  - Good average case performance



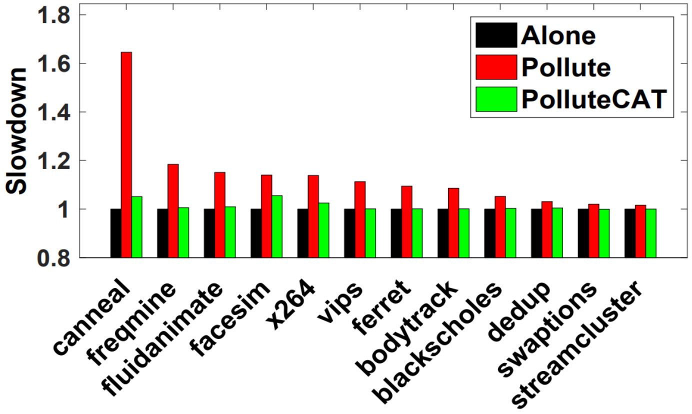
Challenges on Multicore

- Contention for shared resources can cause **interference** between tasks



Challenges on Multicore

- Contention for shared resources can cause **interference** between tasks
- Example: **WCET slowdown** of PARSEC benchmarks due to interference



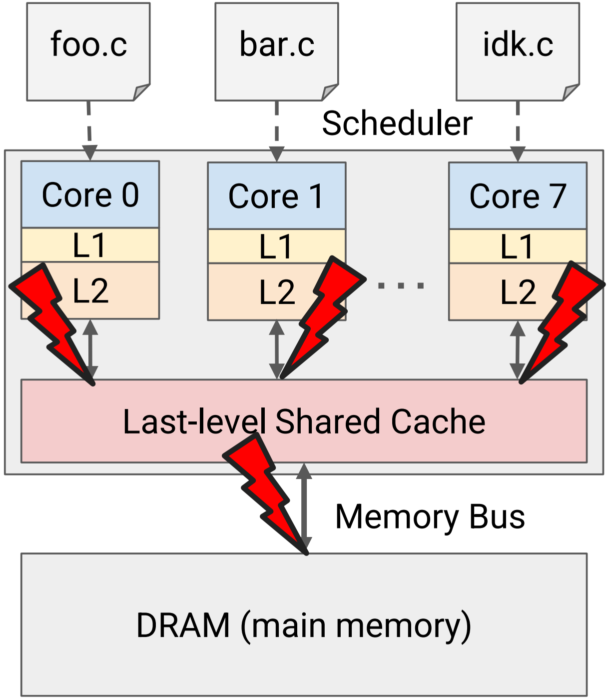
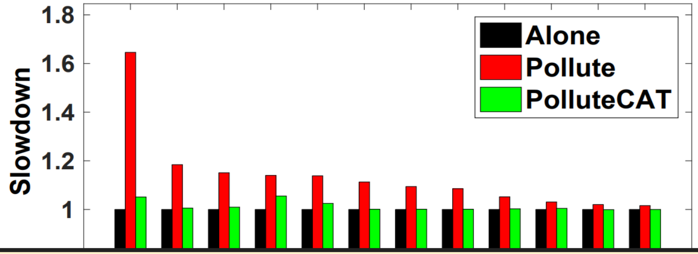
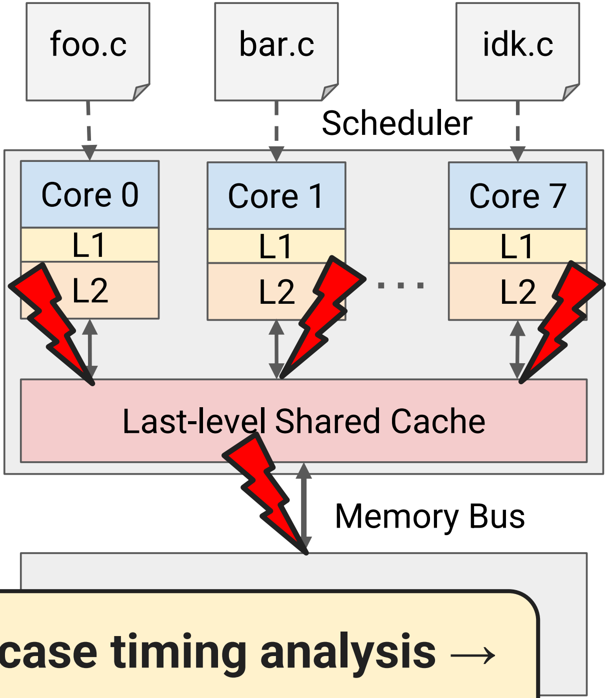


Image from: M. Xu *et al.*, "Holistic Resource Allocation for Multicore Real-Time Systems," *RTAS, 2019.*

Challenges on Multicore

- Contention for shared resources can cause **interference** between tasks
- Example: **WCET slowdown** of PARSEC benchmarks due to interference





Potential interference → overly-conservative worst-case timing analysis → over-provisioning of hardware resources

Image from: M. Xu *et al.*, "Holistic Resource Allocation for Multicore Real-Time Systems," *RTAS, 2019.*

Resource Contention: State of the Art

	Resource aware?	DAG support?	Resource control?	Dynamic control?	Timing analysis?	Takeaway
[Shi et al. RTAS '24] [Casini et al. RTAS '20] [Tessler et al. RTSS '23] [Zhao et al. RTNS '23]	✓	✓	✗	✗	✓	
CaM [Xu et al. RTAS '19] MMO [Sun et al. ECRTS '25]	✓	✗	✓	✗	✓	
DNA [Gifford et al. RTAS '20]	✓	✗	✓	✓	✗	

Resource Contention: State of the Art

	Resource aware?	DAG support?	Resource control?	Dynamic control?	Timing analysis?	Takeaway
[Shi et al. RTAS ‘24] [Casini et al. RTAS ‘20] [Tessler et al. RTSS ‘23] [Zhao et al. RTNS ‘23]	✓	✓	✗	✗	✓	No resource isolation, can still have interference
CaM [Xu et al. RTAS ‘19] MMO [Sun et al. ECRTS ‘25]	✓	✗	✓	✗	✓	
DNA [Gifford et al. RTAS ‘20]	✓	✗	✓	✓	✗	

Resource Contention: State of the Art

	Resource aware?	DAG support?	Resource control?	Dynamic control?	Timing analysis?	Takeaway
[Shi et al. RTAS '24] [Casini et al. RTAS '20] [Tessler et al. RTSS '23] [Zhao et al. RTNS '23]	✓	✓	✗	✗	✓	No resource isolation, can still have interference
CaM [Xu et al. RTAS '19] MMO [Sun et al. ECRTS '25]	✓	✗	✓	✗	✓	
DNA [Gifford et al. RTAS '20]	✓	✗	✓	✓	✗	

Resource Contention: State of the Art

	Resource aware?	DAG support?	Resource control?	Dynamic control?	Timing analysis?	Takeaway
[Shi et al. RTAS '24] [Casini et al. RTAS '20] [Tessler et al. RTSS '23] [Zhao et al. RTNS '23]	✓	✓	✗	✗	✓	No resource isolation, can still have interference
CaM [Xu et al. RTAS '19] MMO [Sun et al. ECRTS '25]	✓	✗	✓	✗	✓	Static allocation is resource inefficient
DNA [Gifford et al. RTAS '20]	✓	✗	✓	✓	✗	

Resource Contention: State of the Art

	Resource aware?	DAG support?	Resource control?	Dynamic control?	Timing analysis?	Takeaway
[Shi et al. RTAS '24] [Casini et al. RTAS '20] [Tessler et al. RTSS '23] [Zhao et al. RTNS '23]	✓	✓	✗	✗	✓	No resource isolation, can still have interference
CaM [Xu et al. RTAS '19] MMO [Sun et al. ECRTS '25]	✓	✗	✓	✗	✓	Static allocation is resource inefficient
DNA [Gifford et al. RTAS '20]	✓	✗	✓	✓	✗	

Resource Contention: State of the Art

	Resource aware?	DAG support?	Resource control?	Dynamic control?	Timing analysis?	Takeaway
[Shi et al. RTAS '24] [Casini et al. RTAS '20] [Tessler et al. RTSS '23] [Zhao et al. RTNS '23]	✓	✓	✗	✗	✓	No resource isolation, can still have interference
CaM [Xu et al. RTAS '19] MMO [Sun et al. ECRTS '25]	✓	✗	✓	✗	✓	Static allocation is resource inefficient
DNA [Gifford et al. RTAS '20]	✓	✗	✓	✓	✗	Dynamic allocation breaks timing guarantee

Rasco: Fine-grain Resource Control with Guarantees

	Resource aware?	DAG support?	Resource control?	Dynamic control?	Timing analysis?
Rasco [this work]	✓	✓	✓	✓	✓

We want the best of both worlds:

✓ Tight worst-case timing analysis via resource isolation

✓ Dynamic allocation of resources based on fine-grain needs

Research Questions

1. How do we design a task model that enables dynamic resource allocation
   and worst-case timing analysis?

2. Using this model, how do we allocate resources to improve the

   - resource efficiency,
   - average-case latency,
   - and hard real-time schedulability

   of DAG applications?

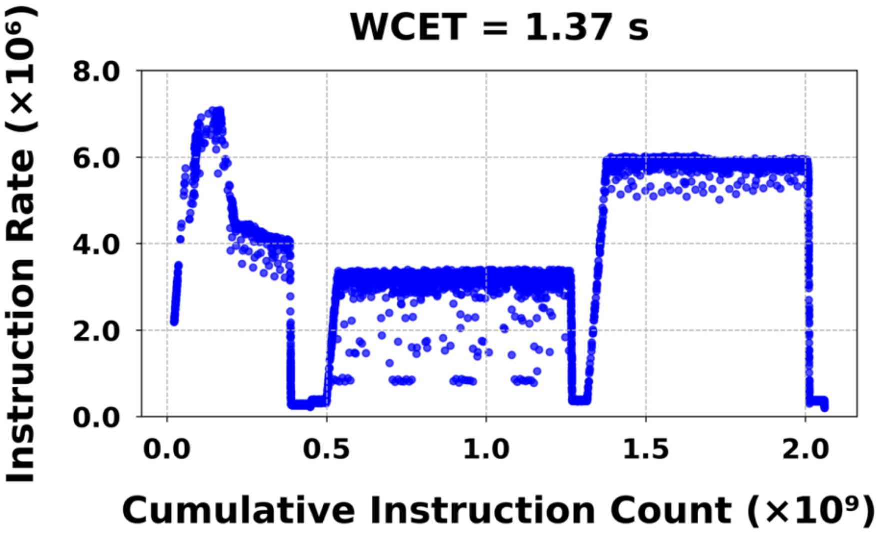
Contributions

- To answer RQ1, we propose a **resource-dependent multi-phase task model** which enables worst-case timing analysis under dynamic resource allocation
- To answer RQ2, we develop Rasco, a resource allocation and scheduling **co-design algorithm** for DAG applications on multicore
- We then implement a **prototype** of Rasco to evaluate the safety and utility of our approach in a real-time operating system

Talk Outline

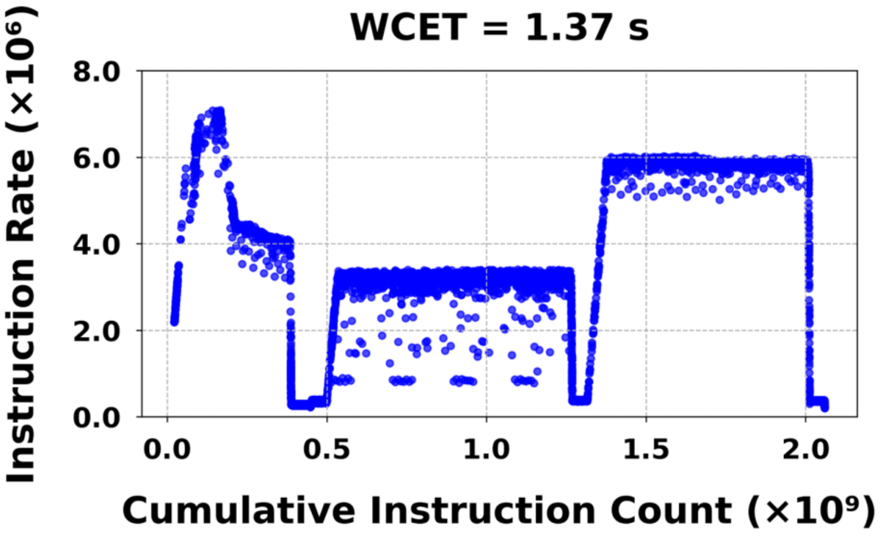
1. Introduction/Background
2. **The Resource-Dependent Multi-Phase Model**
3. Rasco: Resource Allocation and Scheduling Co-design
4. Numerical Evaluation
5. Prototype Evaluation and Overhead Accounting
6. Conclusion

Task Execution Phases

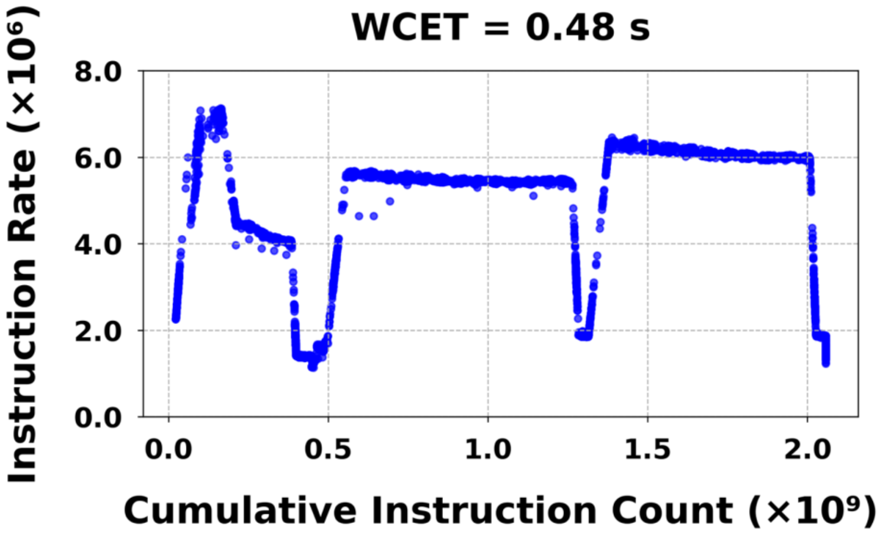


FFT with 10% of LLC and memory BW

Task Execution Phases

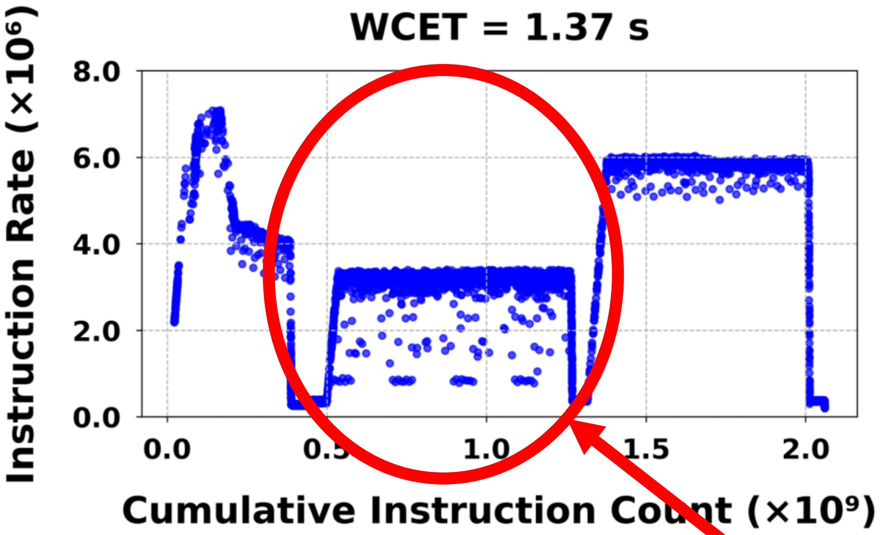


FFT with 10% of LLC and memory BW

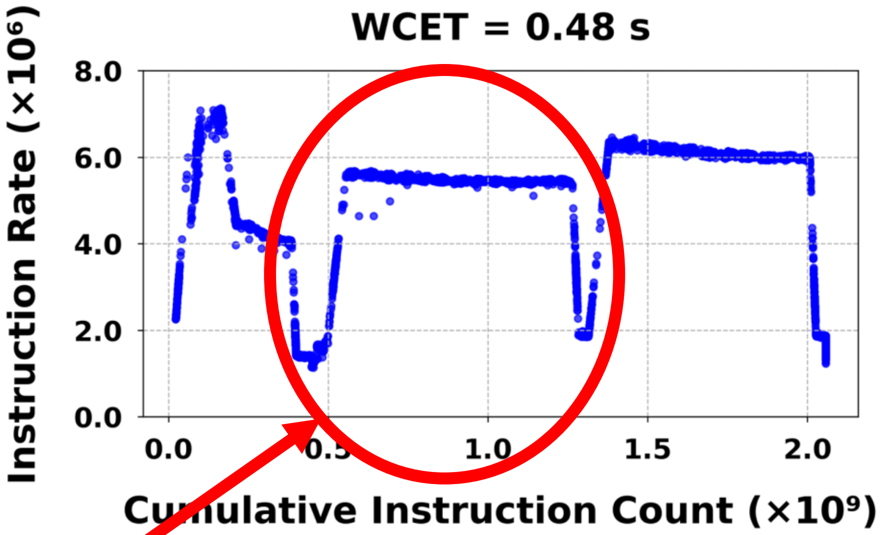


FFT with 50% of LLC and memory BW

Task Execution Phases



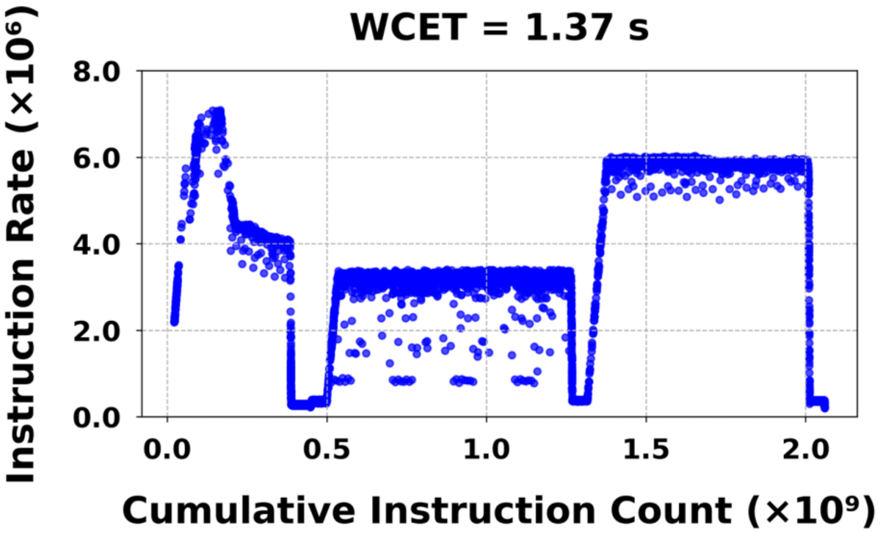
FFT with 10% of LLC and memory BW



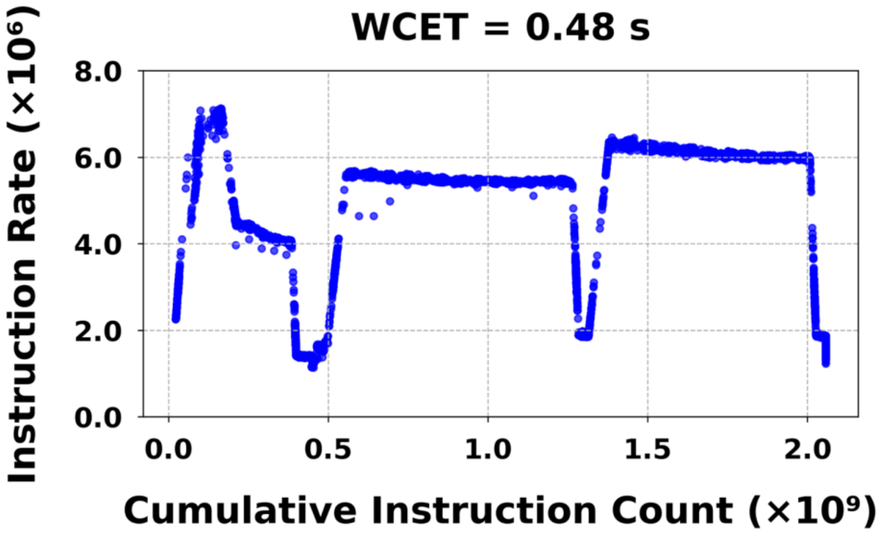
FFT with 50% of LLC and memory BW

Middle phase is highly resource intensive

Task Execution Phases



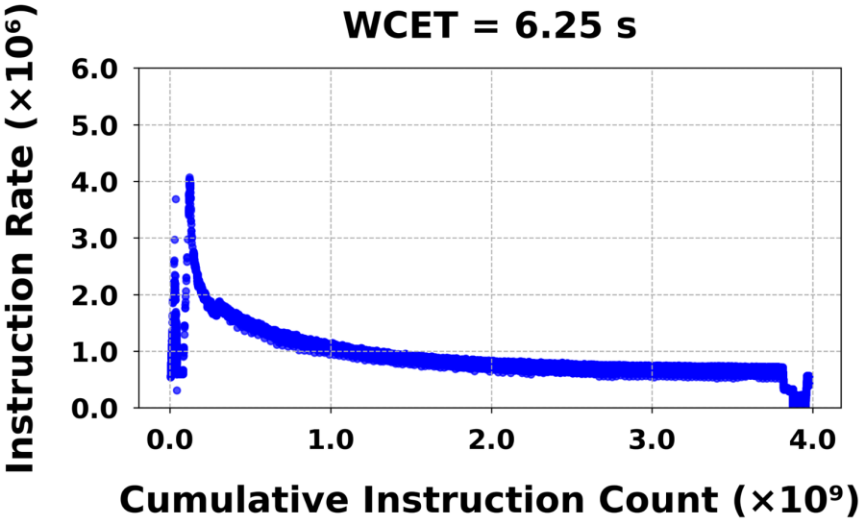
FFT with 10% of LLC and memory BW



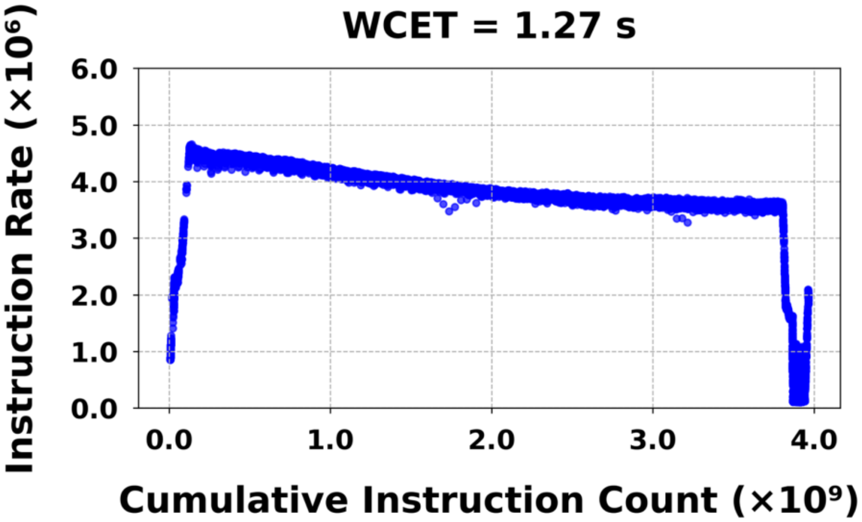
FFT with 50% of LLC and memory BW

- **Tasks have different execution phases**
- **Phases vary in resource intensity**

A General Model for Resource-Dependent Phases

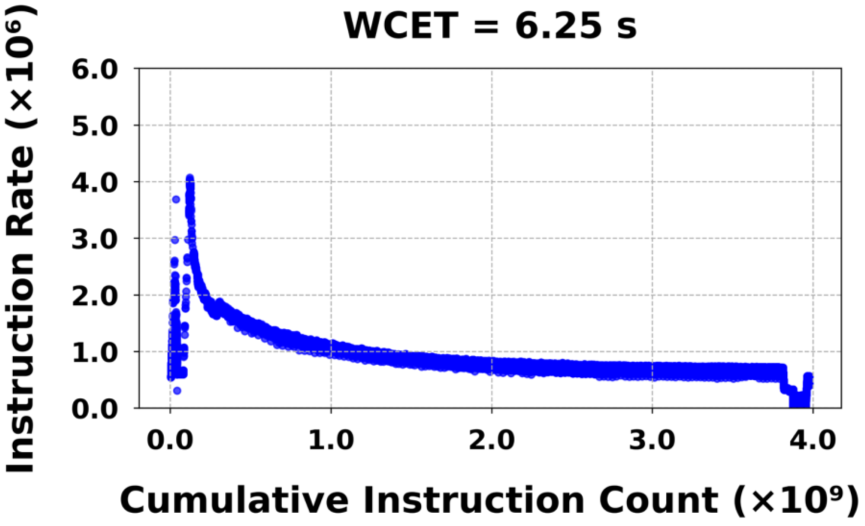


Canneal with 10% of LLC and mem BW

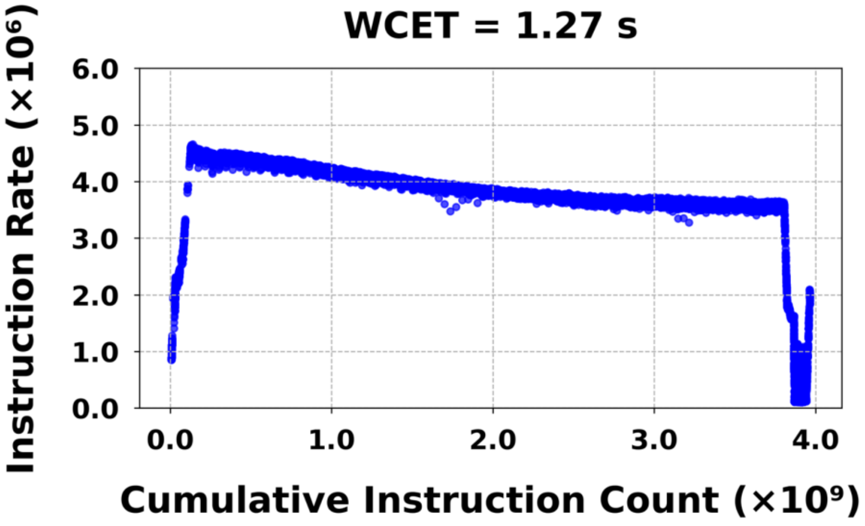


Canneal with 50% of LLC and mem BW

A General Model for Resource-Dependent Phases



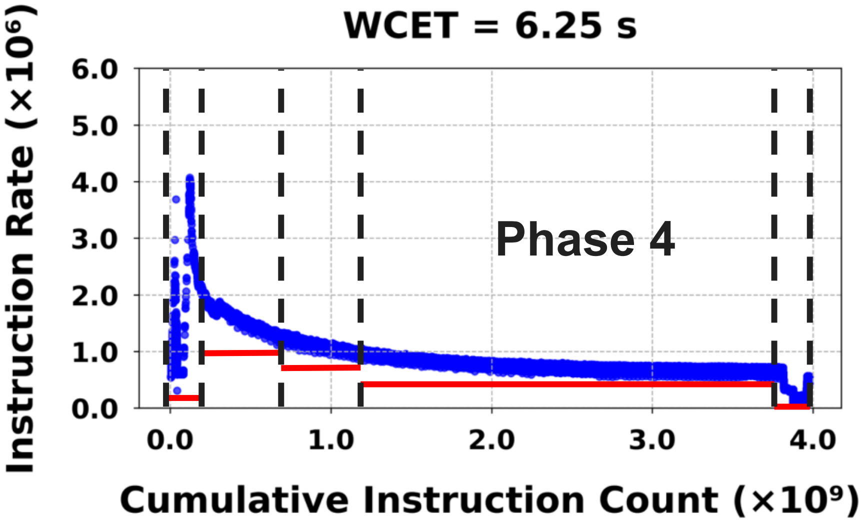
Canneal with 10% of LLC and mem BW



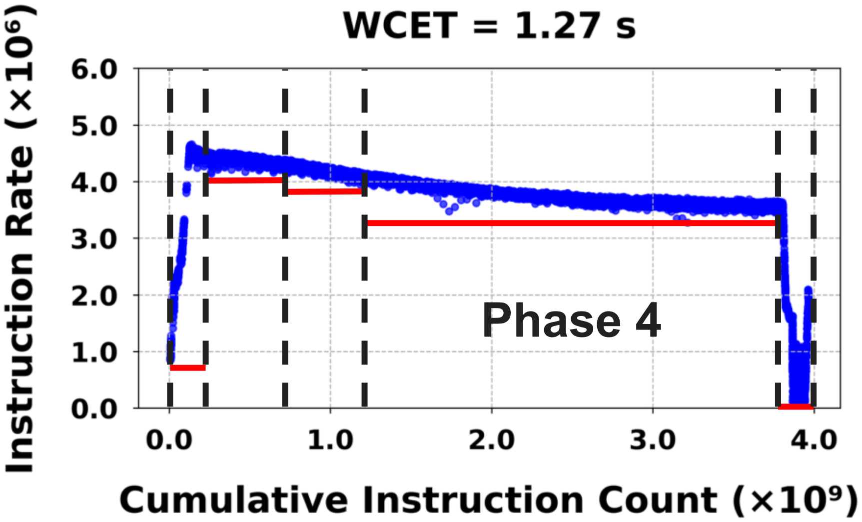
Canneal with 50% of LLC and mem BW

Need a resource-dependent task model that is generalizable, but still tight.

A General Model for Resource-Dependent Phases



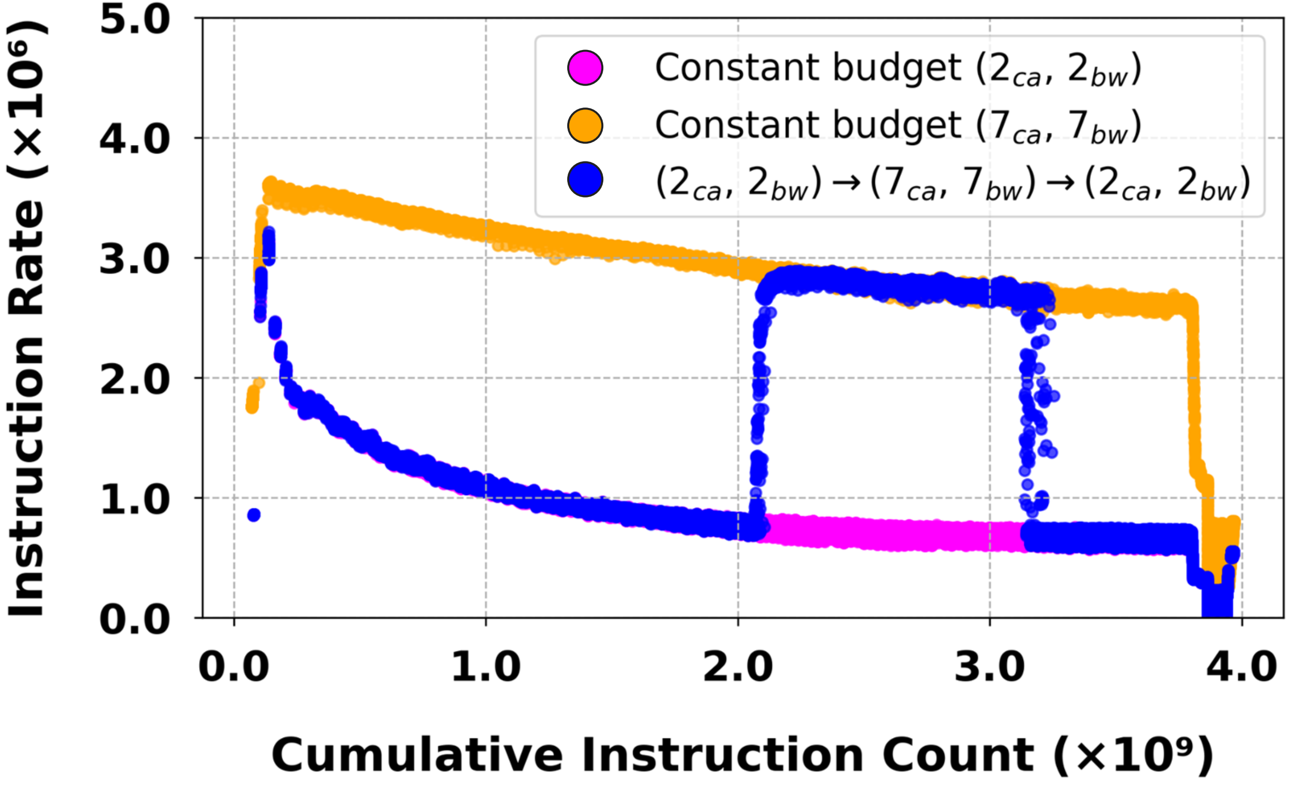
Canneal with 10% of LLC and mem BW



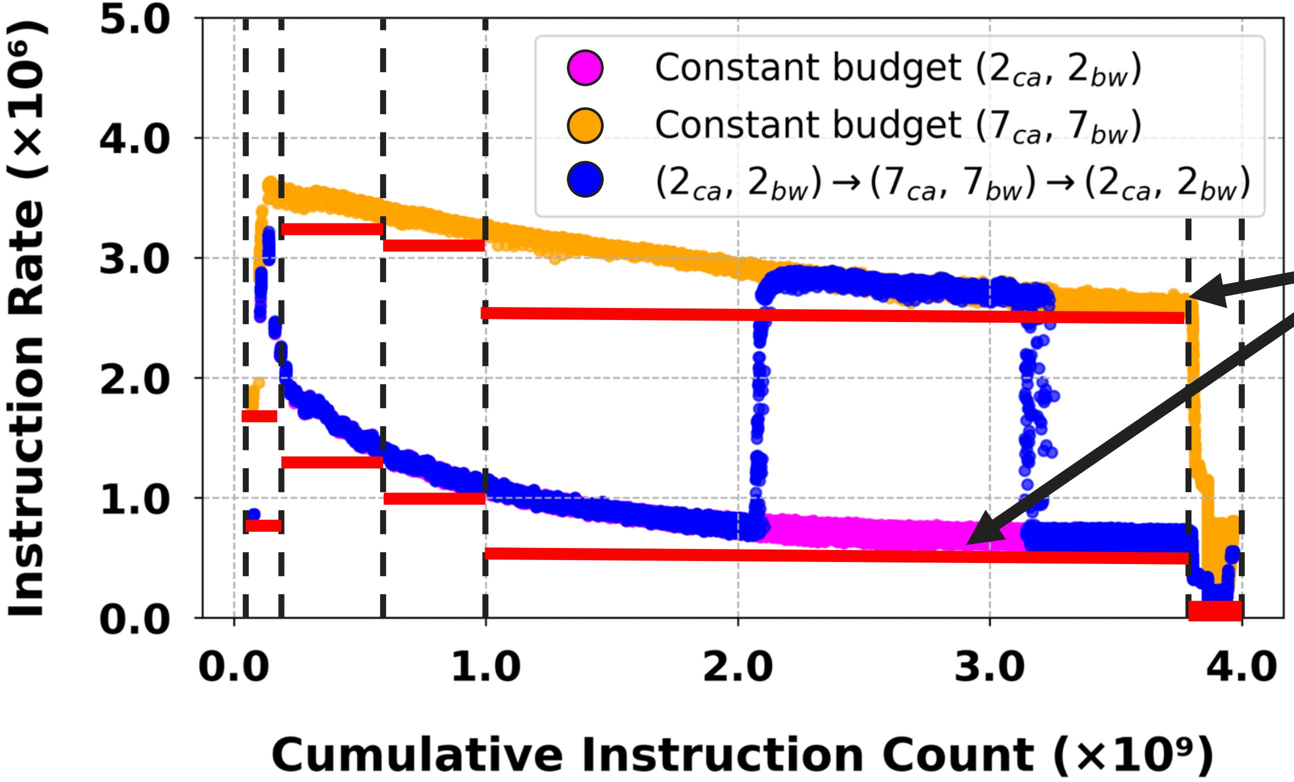
Canneal with 50% of LLC and mem BW

Step 1: use changepoint detection to identify phases
Step 2: compute worst-case instruction rates for each phase

WCET Under Dynamic Resource Allocation

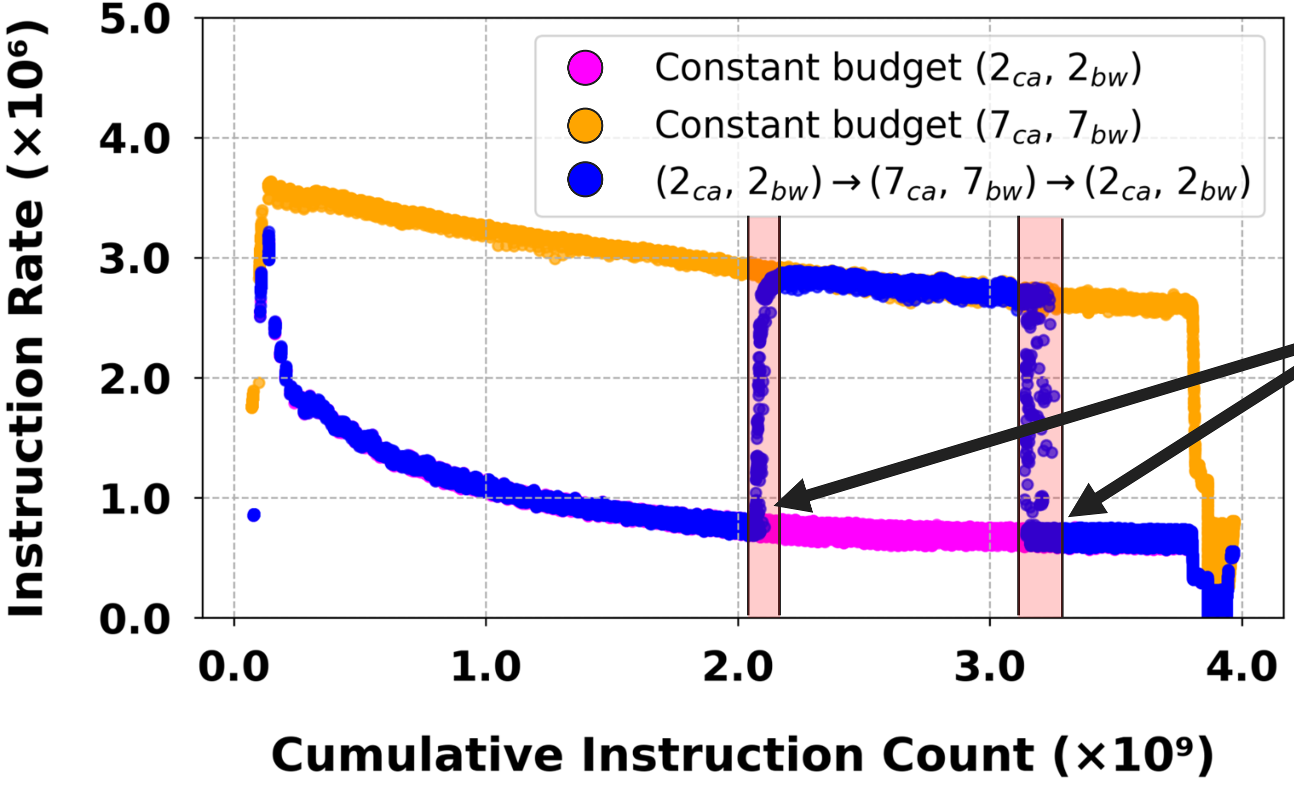


WCET Under Dynamic Resource Allocation



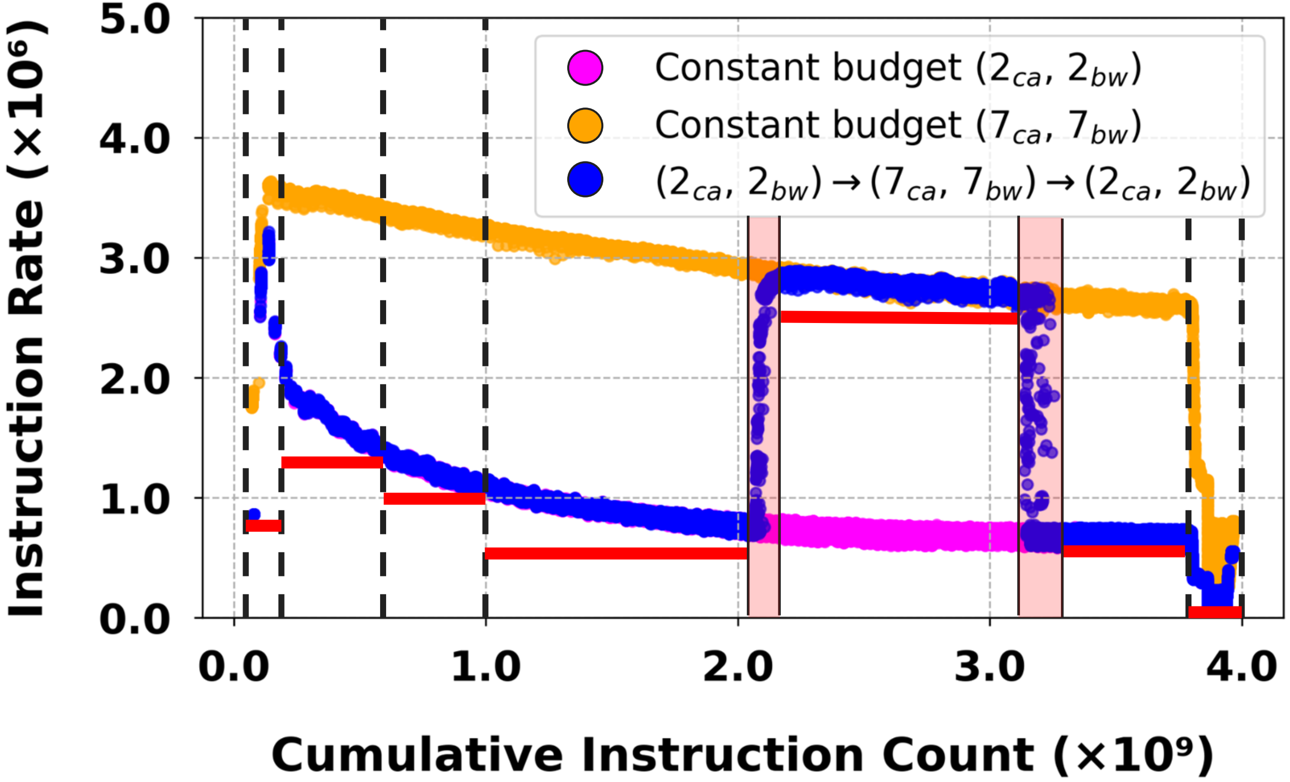
Phases and worst-case rates from multi-phase model

WCET Under Dynamic Resource Allocation

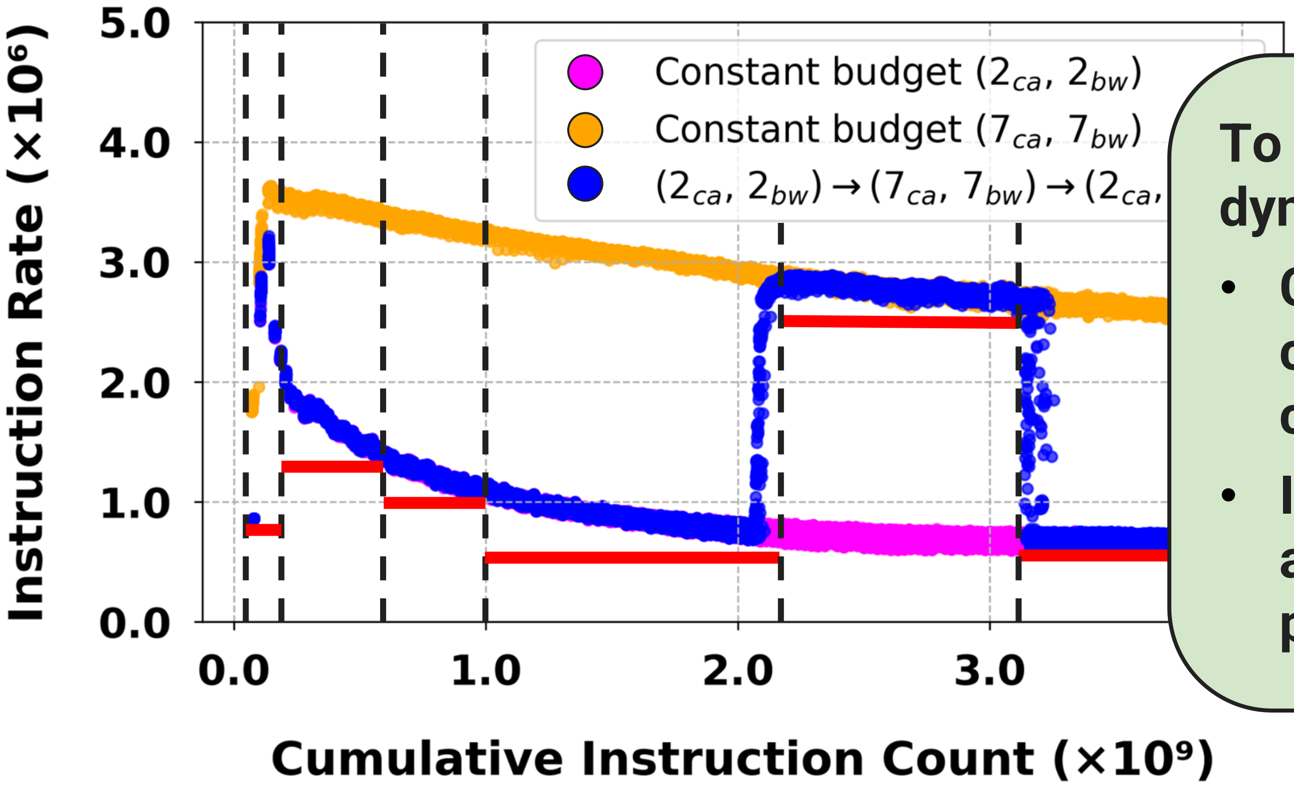


Delay between resource reconfiguration and rate change

WCET Under Dynamic Resource Allocation



WCET Under Dynamic Resource Allocation



To get WCET under dynamic budget:

- **Compose worst-case rates from constant budgets**
- **Incorportate delay at reconfiguration points**

Research Questions

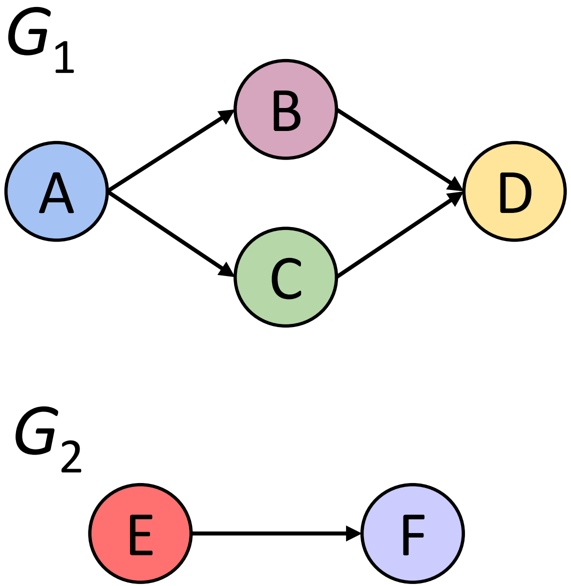
1. How do we design a task model that enables dynamic resource allocation and worst-case timing analysis? ✓

2. Using this model, how do we allocate resources to improve the

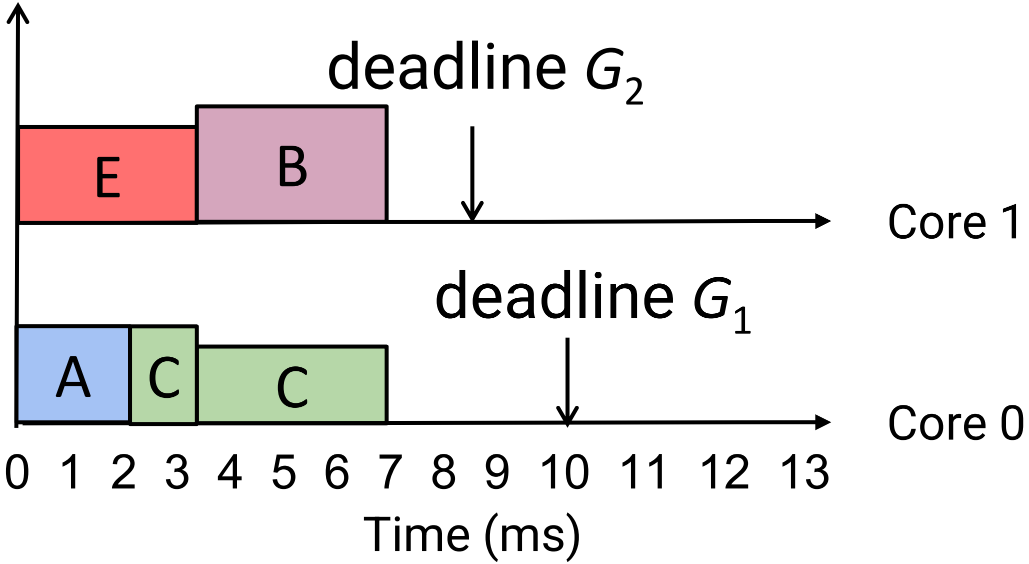
   - resource efficiency,
   - average-case latency,
   - and hard real-time schedulability

   of DAG applications?

Resource Allocation and Scheduling

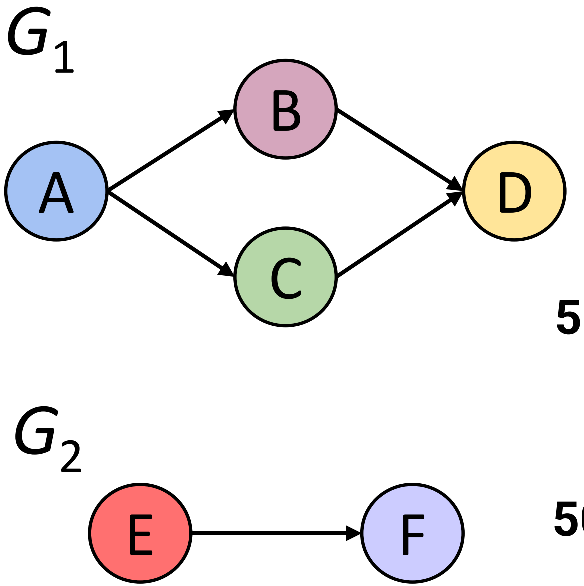


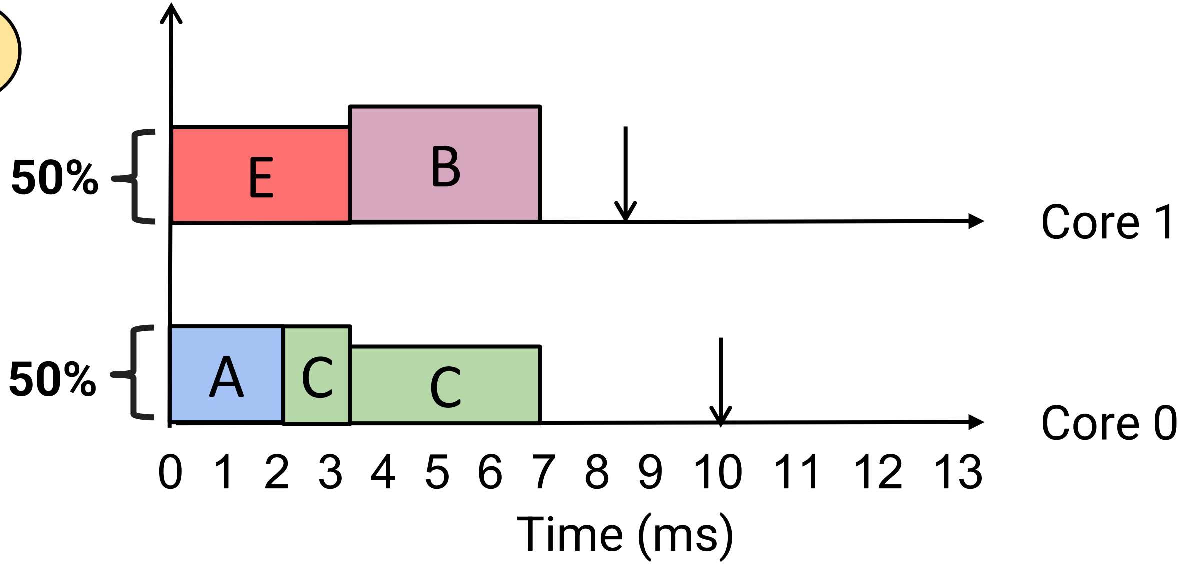
Suppose we are at time t = 7
and need to schedule D and F



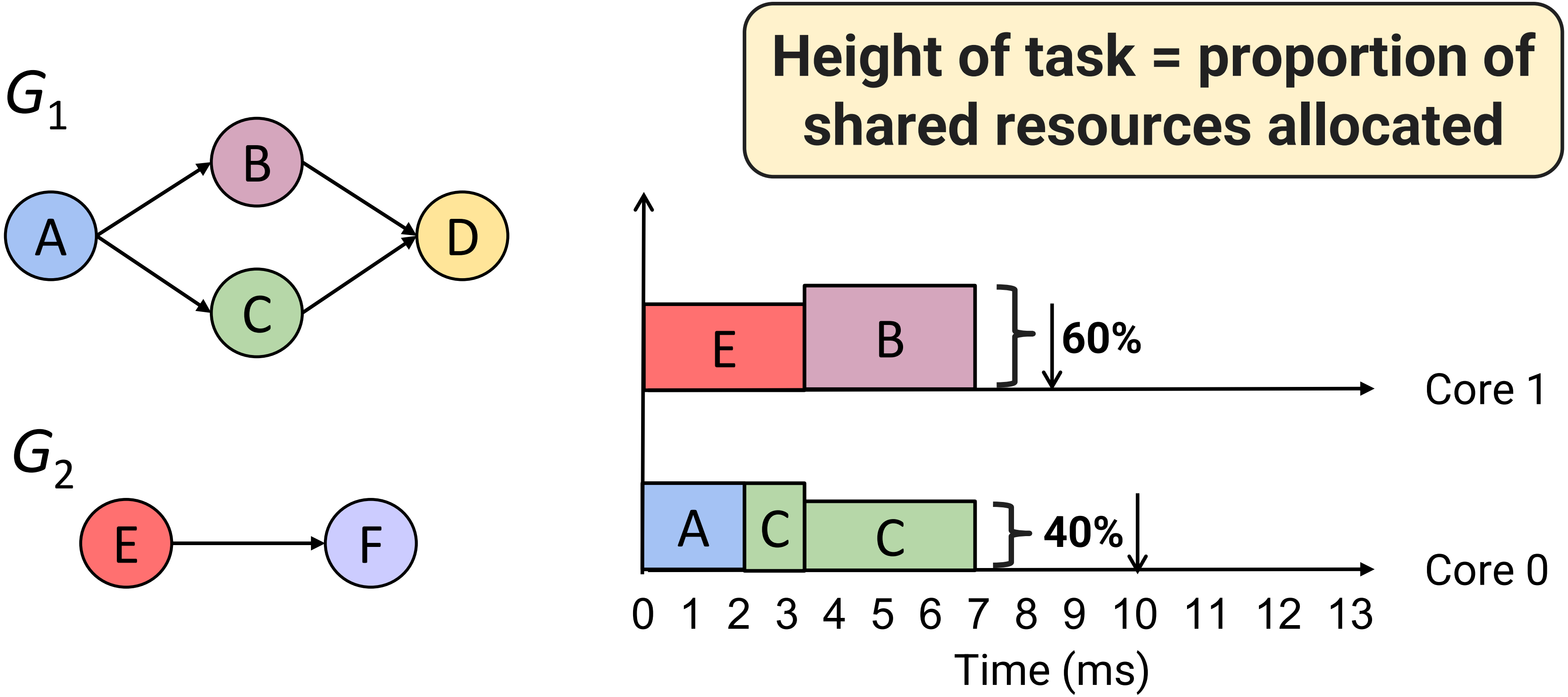
Resource Allocation and Scheduling

Height of task = proportion of shared resources allocated

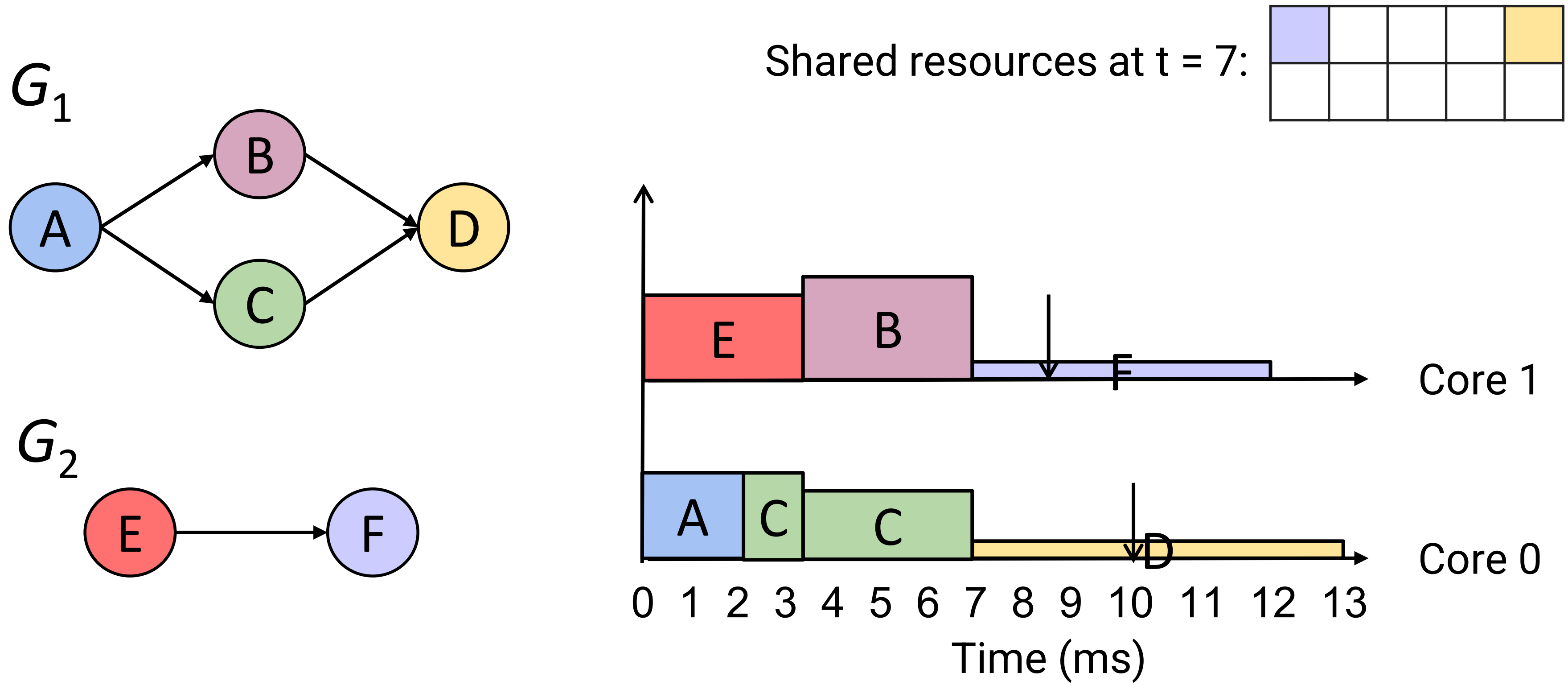




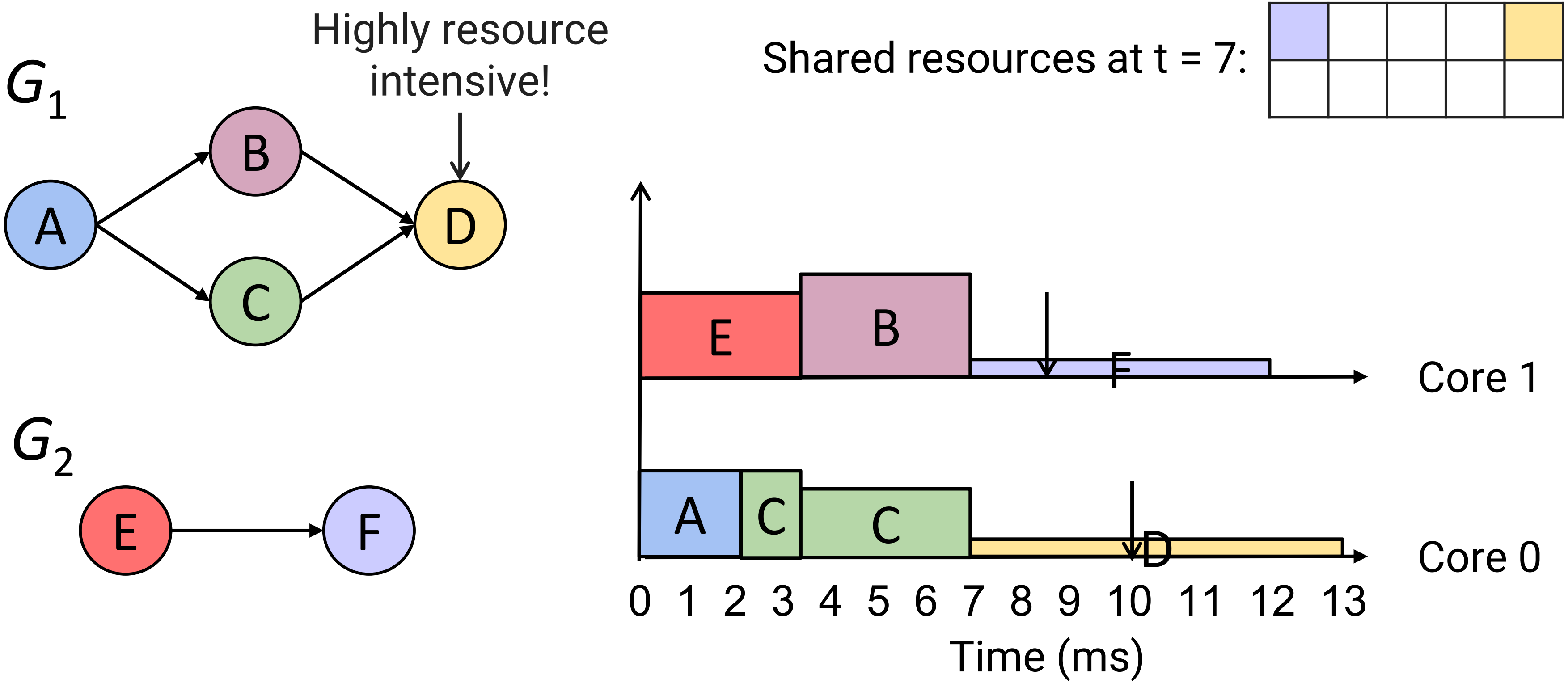
Resource Allocation and Scheduling



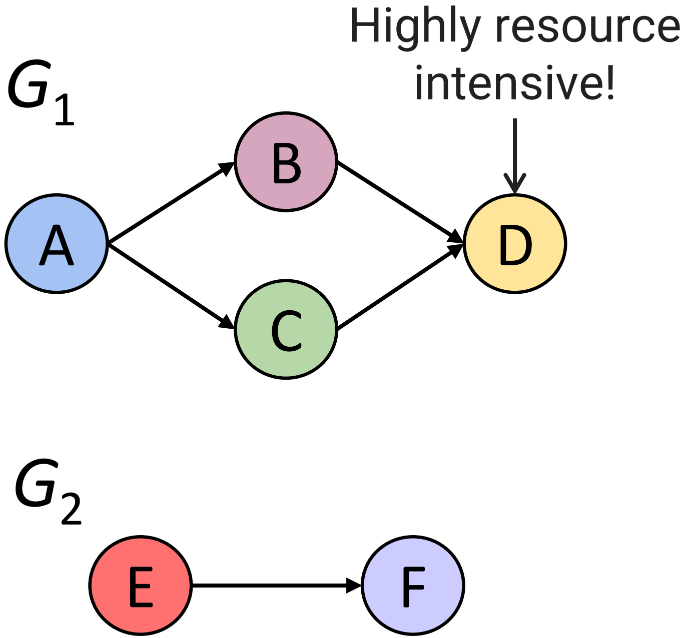
Resource Allocation and Scheduling

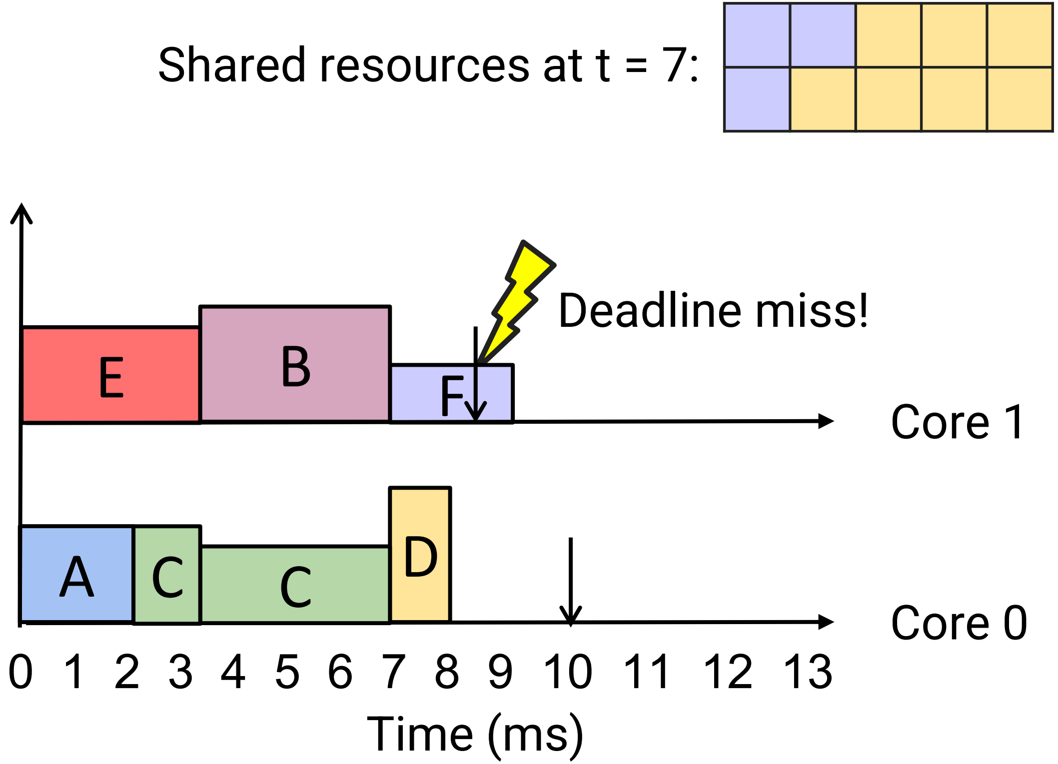


Resource Allocation and Scheduling



Resource Allocation and Scheduling





The Case for Co-Design

Maximizing resource efficiency ≠ maximizing schedulability

The Case for Co-Design

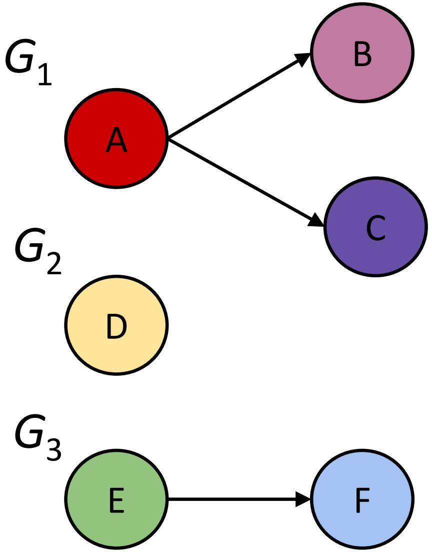
Maximizing resource efficiency ≠ maximizing schedulability

Resource allocation and scheduling must be co-designed.

Talk Outline

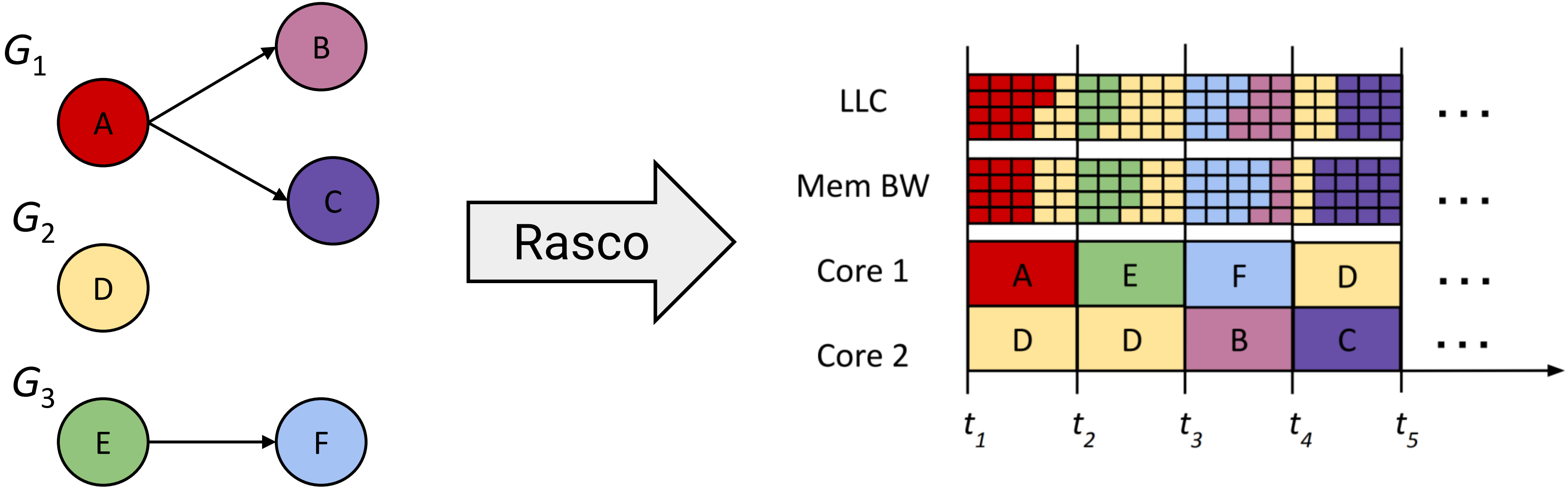
1. Introduction/Background
2. The Resource-Dependent Multi-Phase Model
3. Rasco: Resource Allocation and Scheduling Co-design
4. Numerical Evaluation
5. Prototype Evaluation and Overhead Accounting
6. Conclusion

Rasco Algorithm Overview



Input: periodic DAG tasks and the multi-phase model for each task

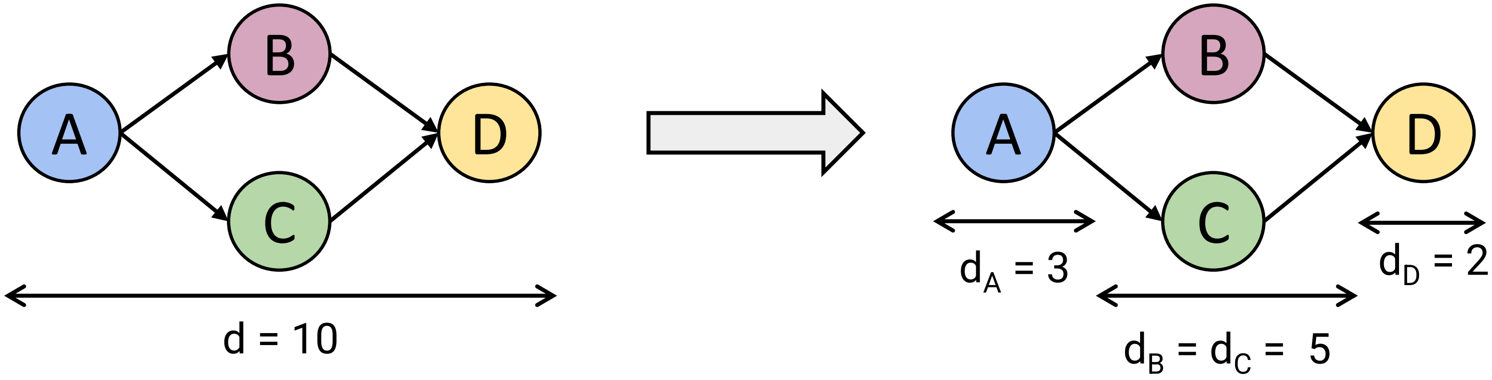
Rasco Algorithm Overview



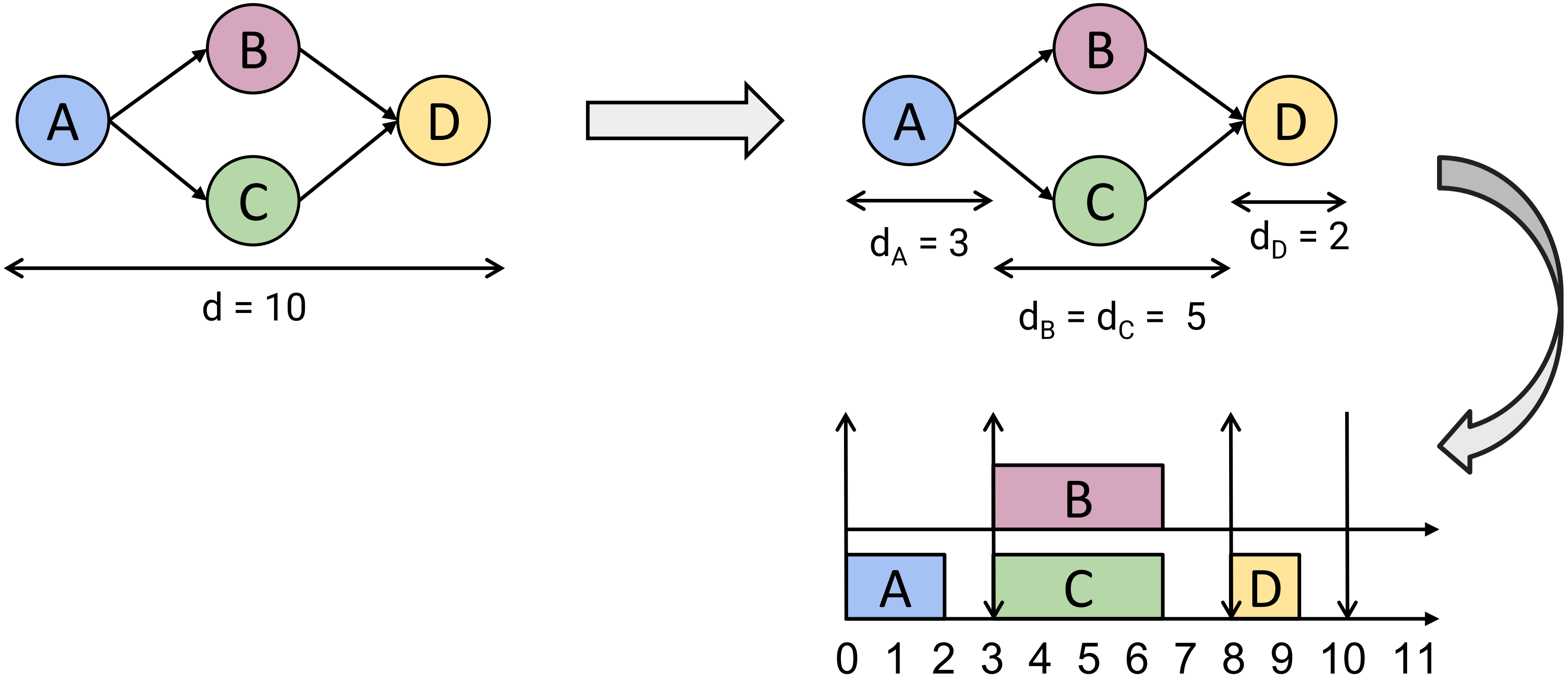
Input: periodic DAG tasks and the multi-phase model for each task

Output: task schedule with partition of shared resources at each scheduling point

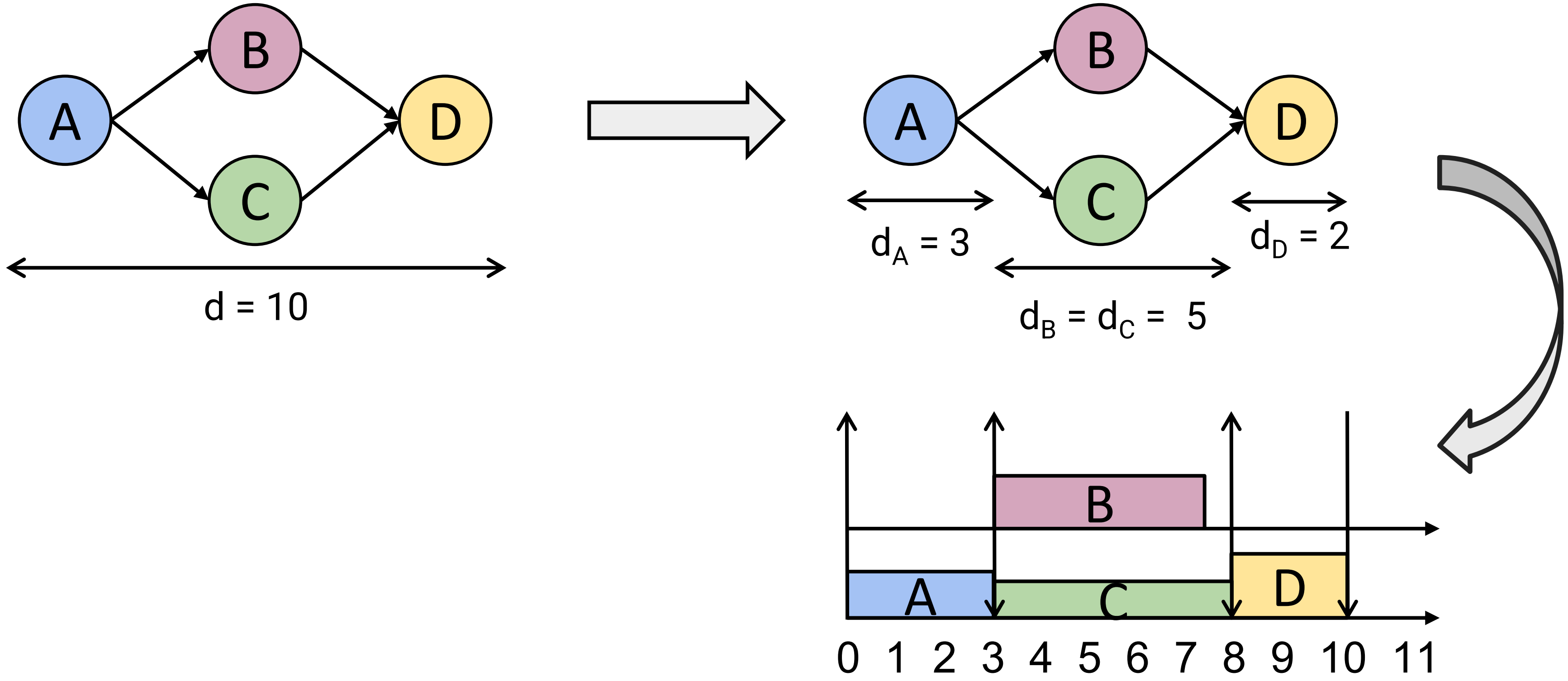
Rasco Pre-Processing: Deadline decomposition



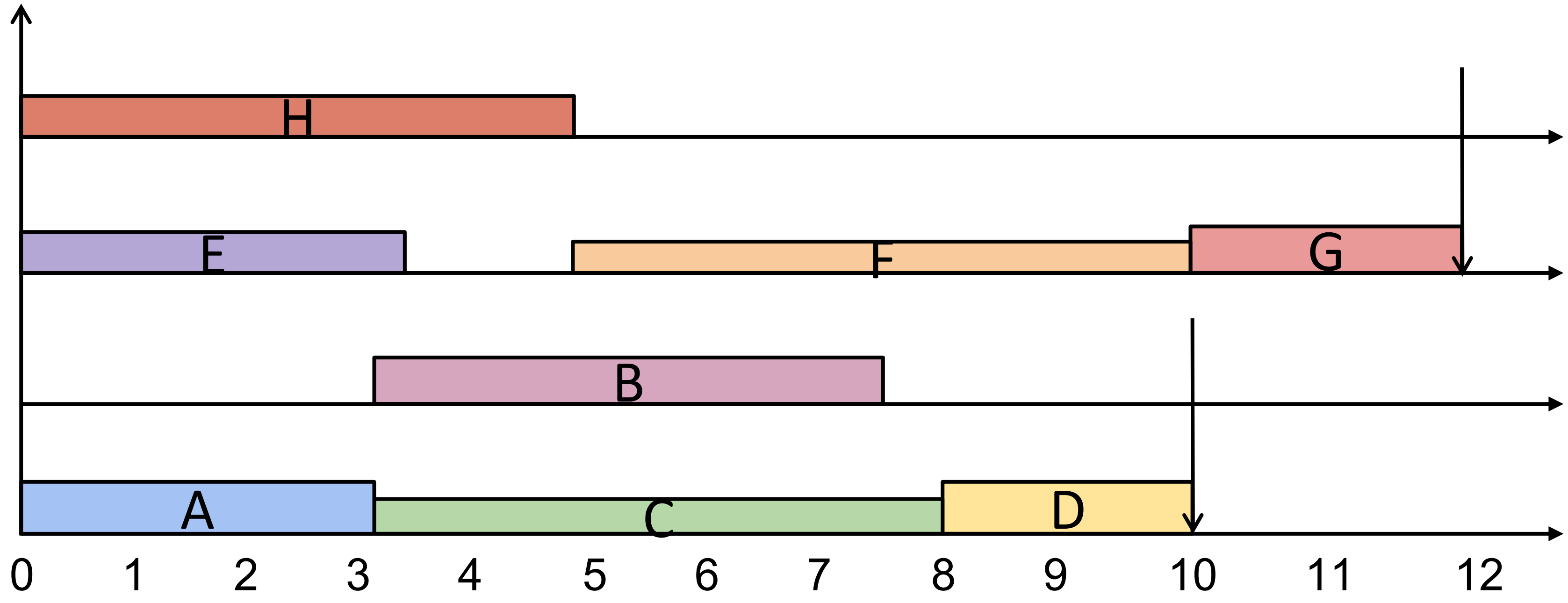
Rasco Pre-Processing: Deadline decomposition



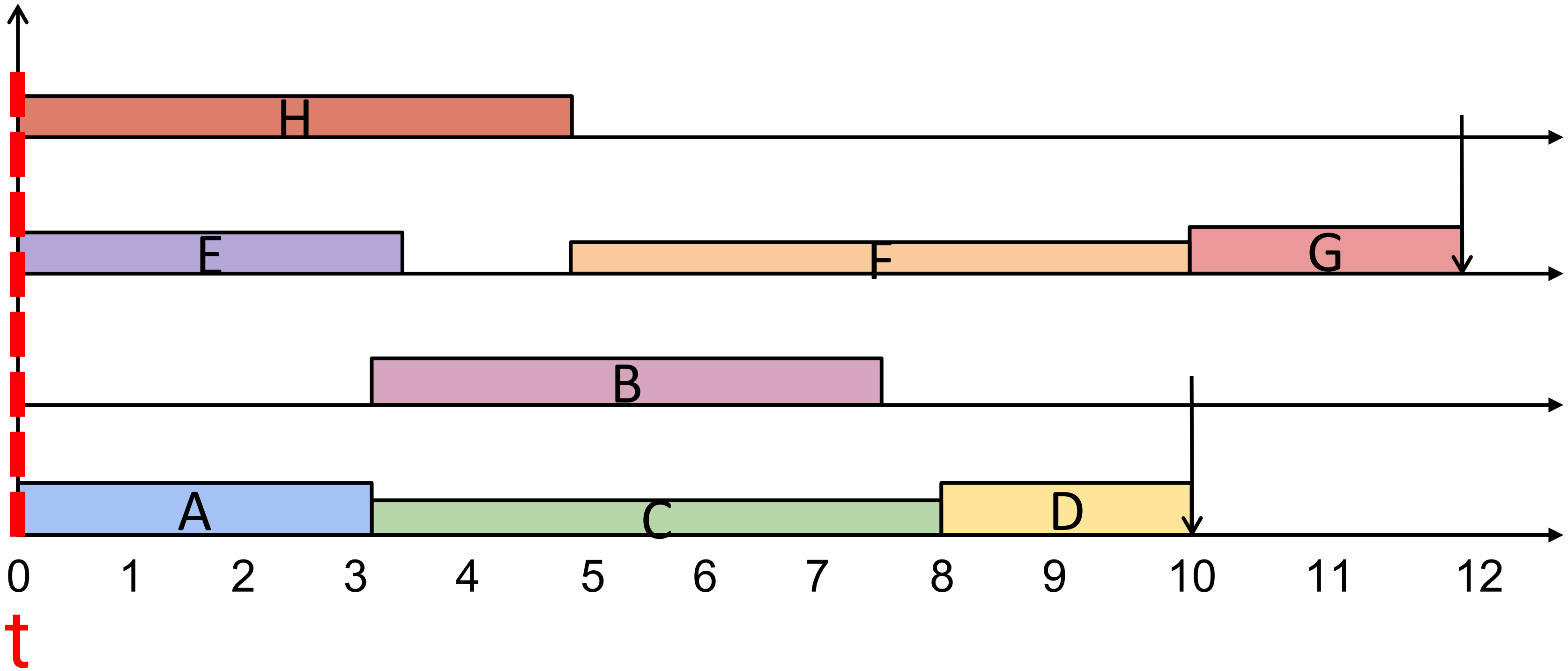
Rasco Pre-Processing: Deadline decomposition



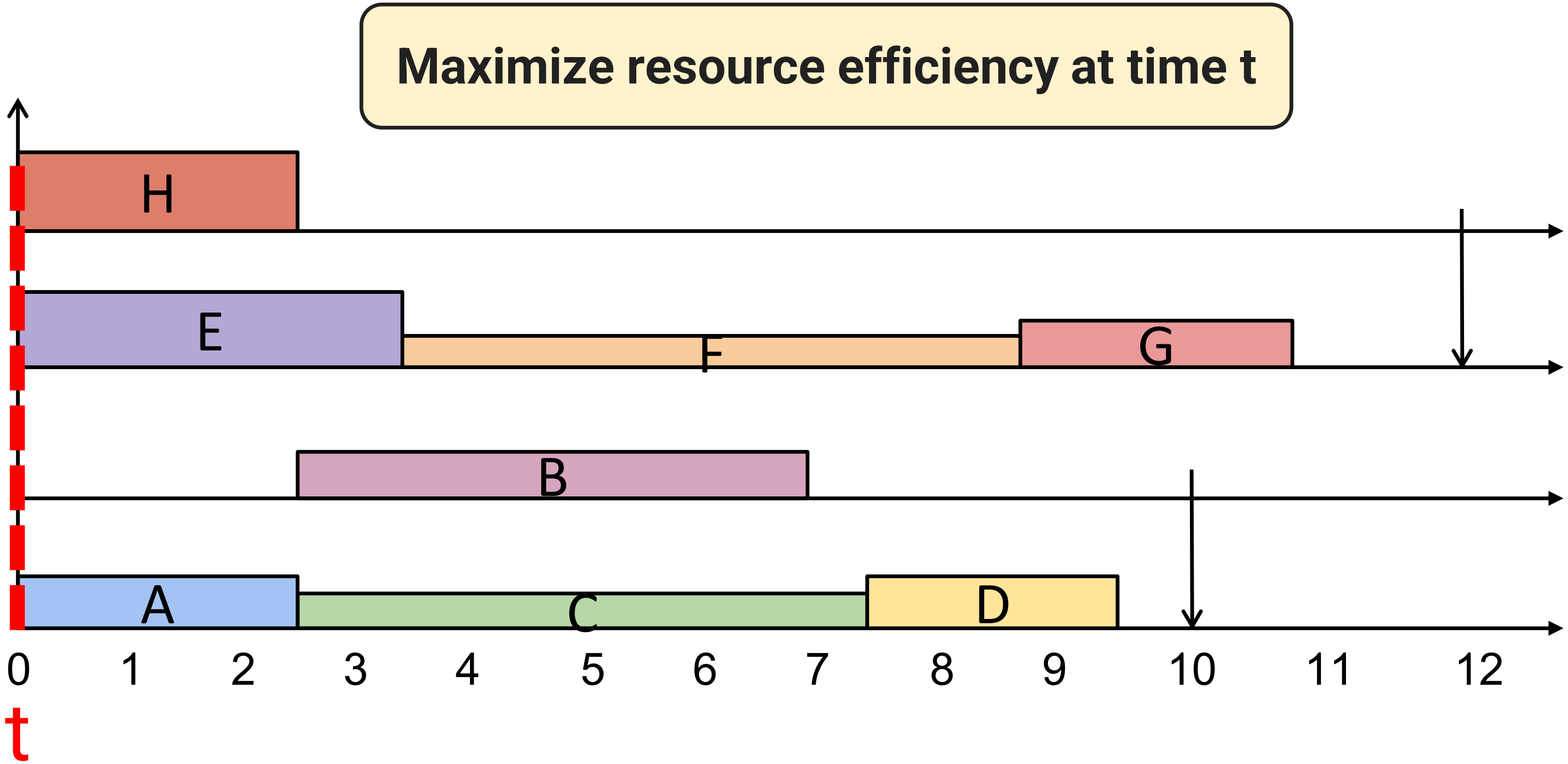
Rasco Pre-Processing: Stack all DAG tasks (G_1, G_2)



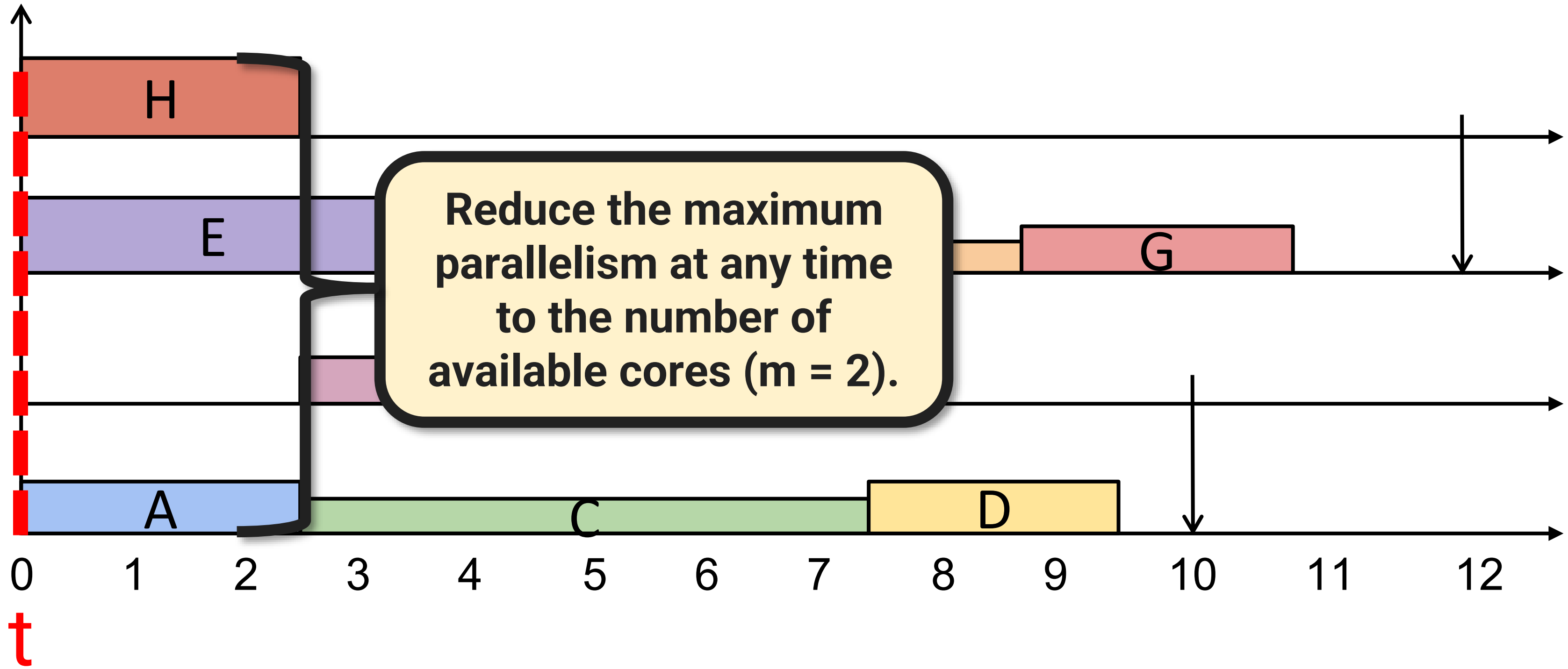
Step 1: Start at t = 0, give out remaining resources at t



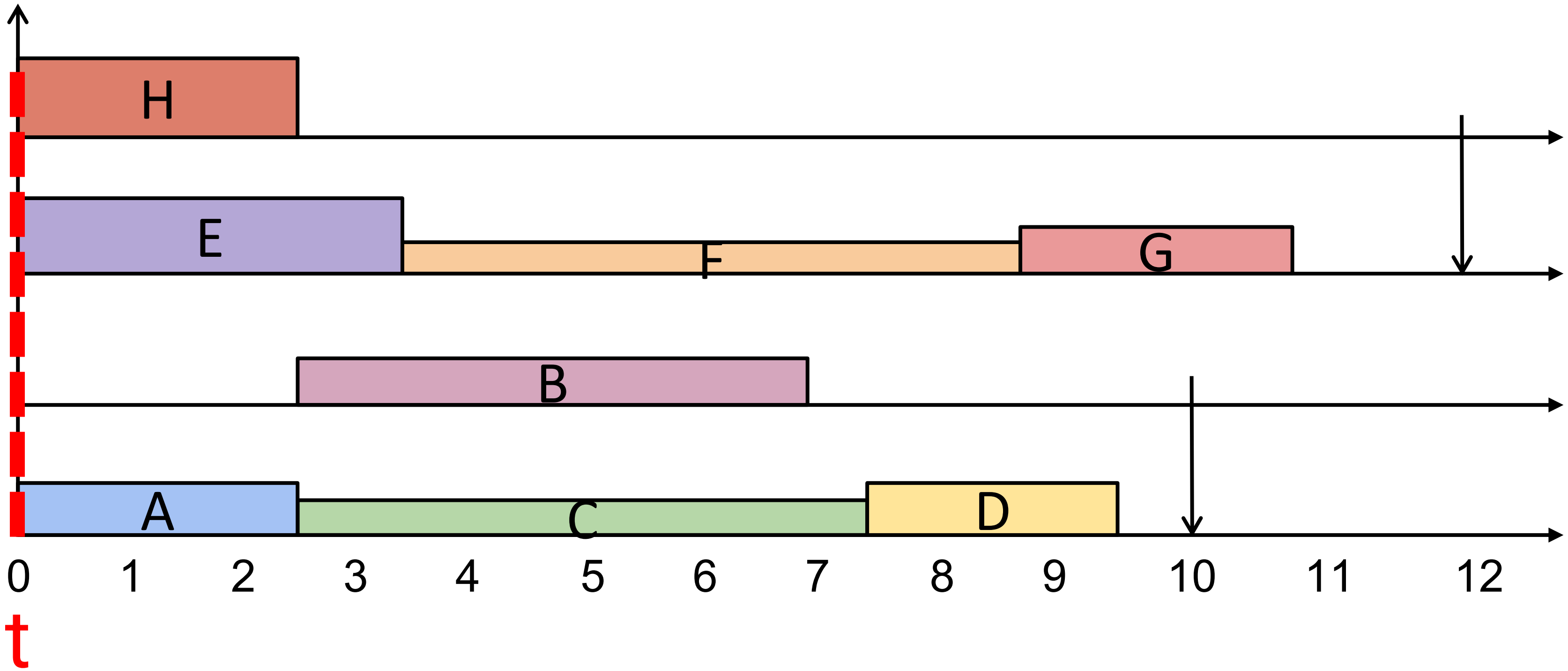
Step 1: Start at t = 0, give out remaining resources at t



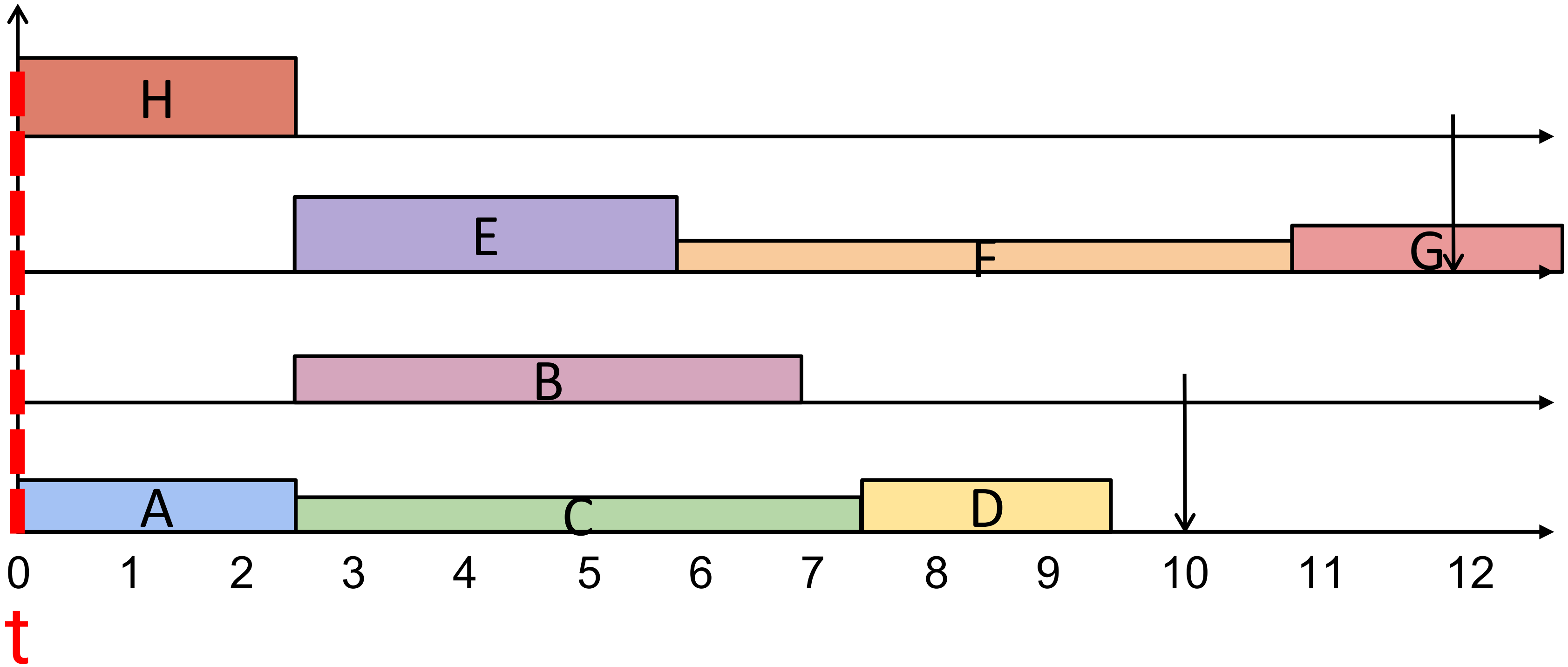
Step 1: Start at t = 0, give out remaining resources at t



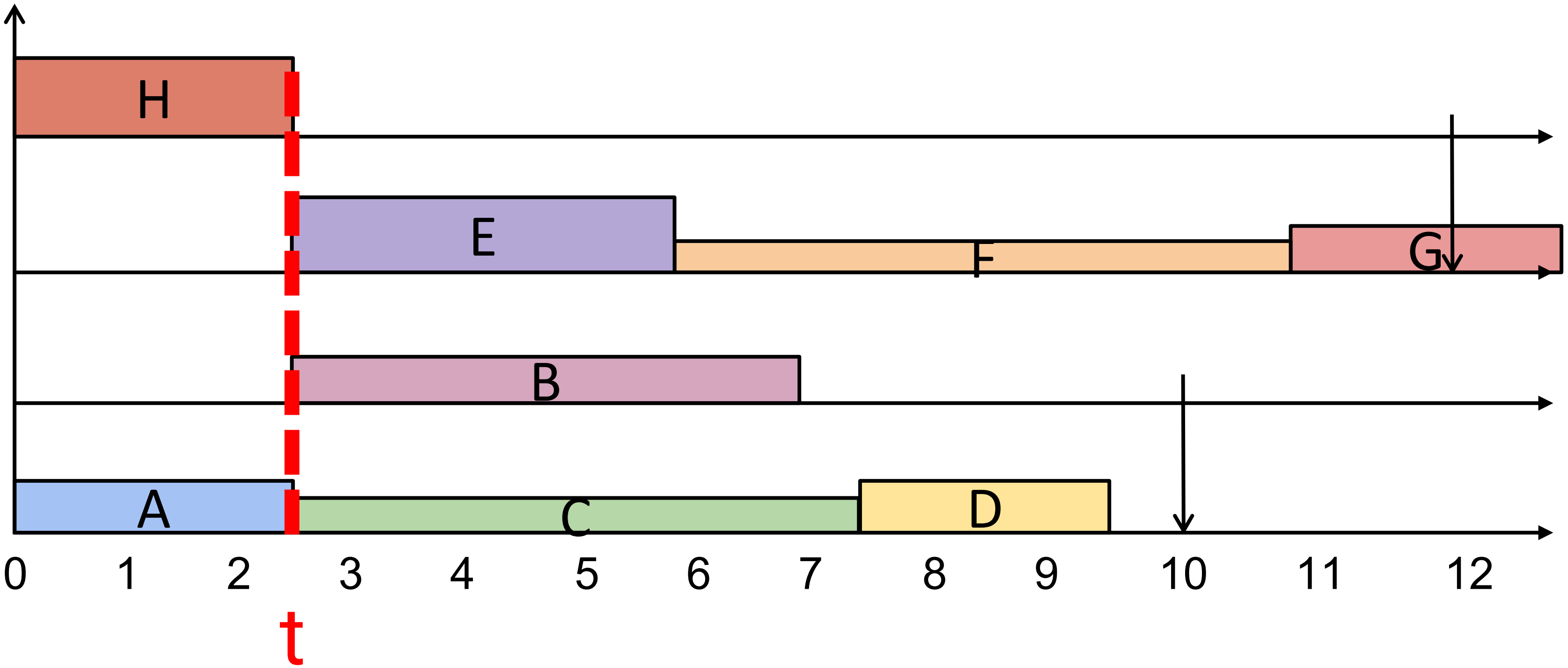
Step 2: Schedule m = 2 tasks with smallest deadlines



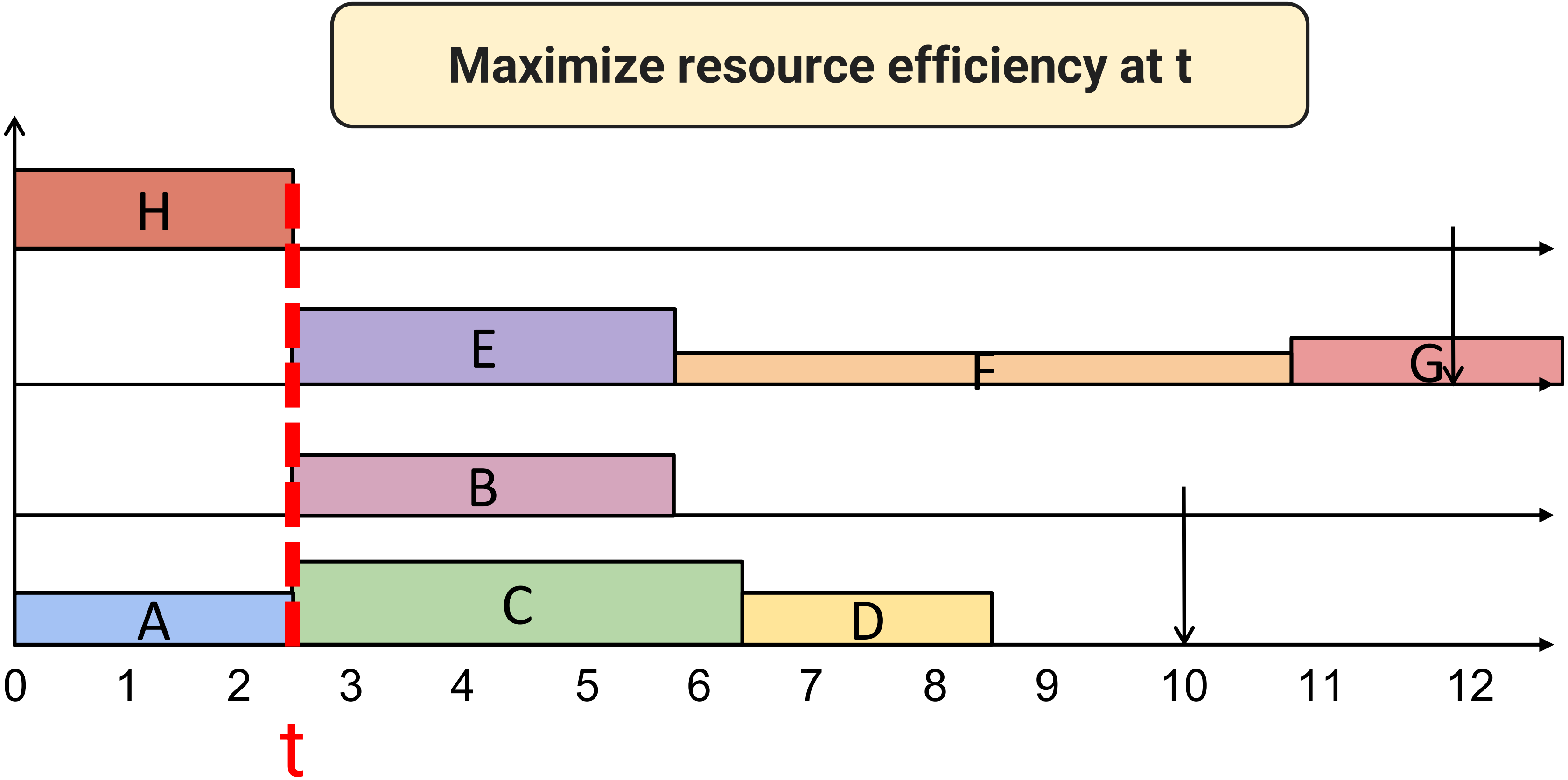
Step 2: Schedule m = 2 tasks with smallest deadlines



Step 3: Get next t, repeat

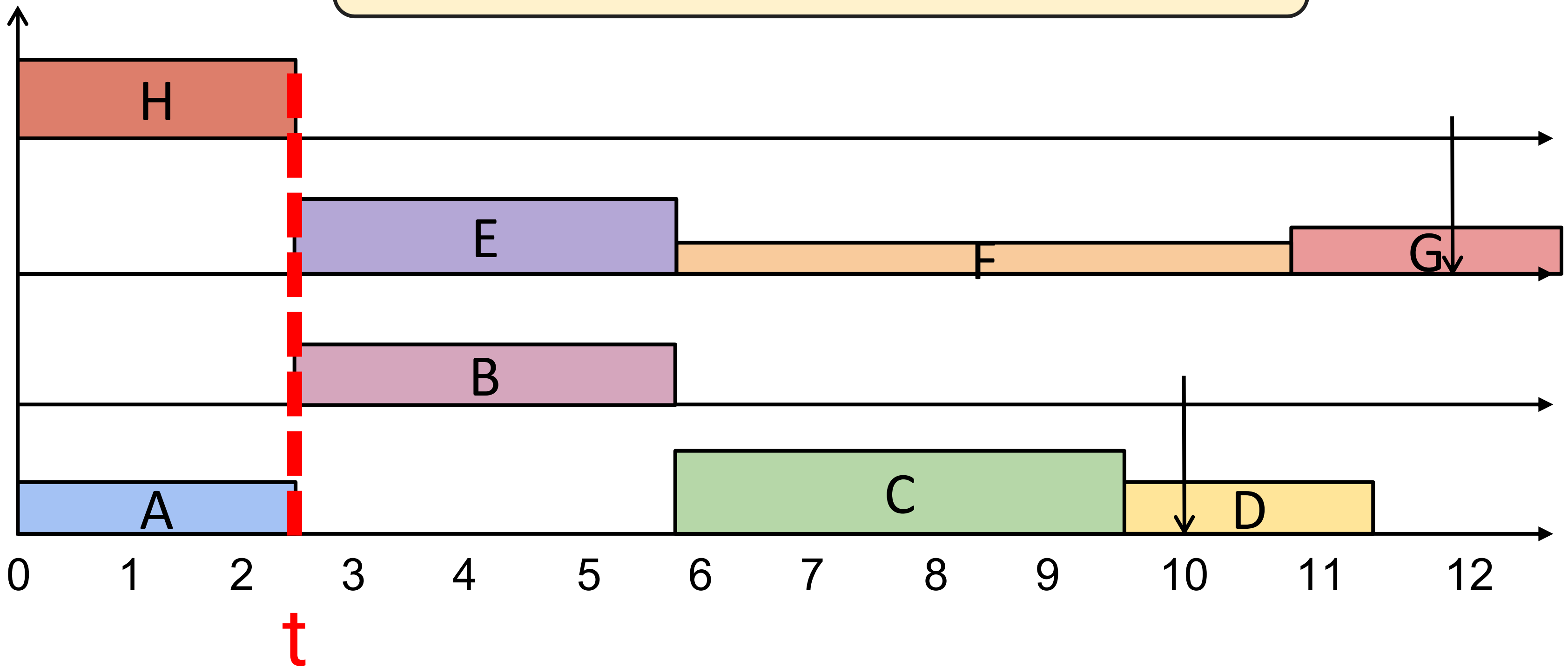


Step 3: Get next t, repeat

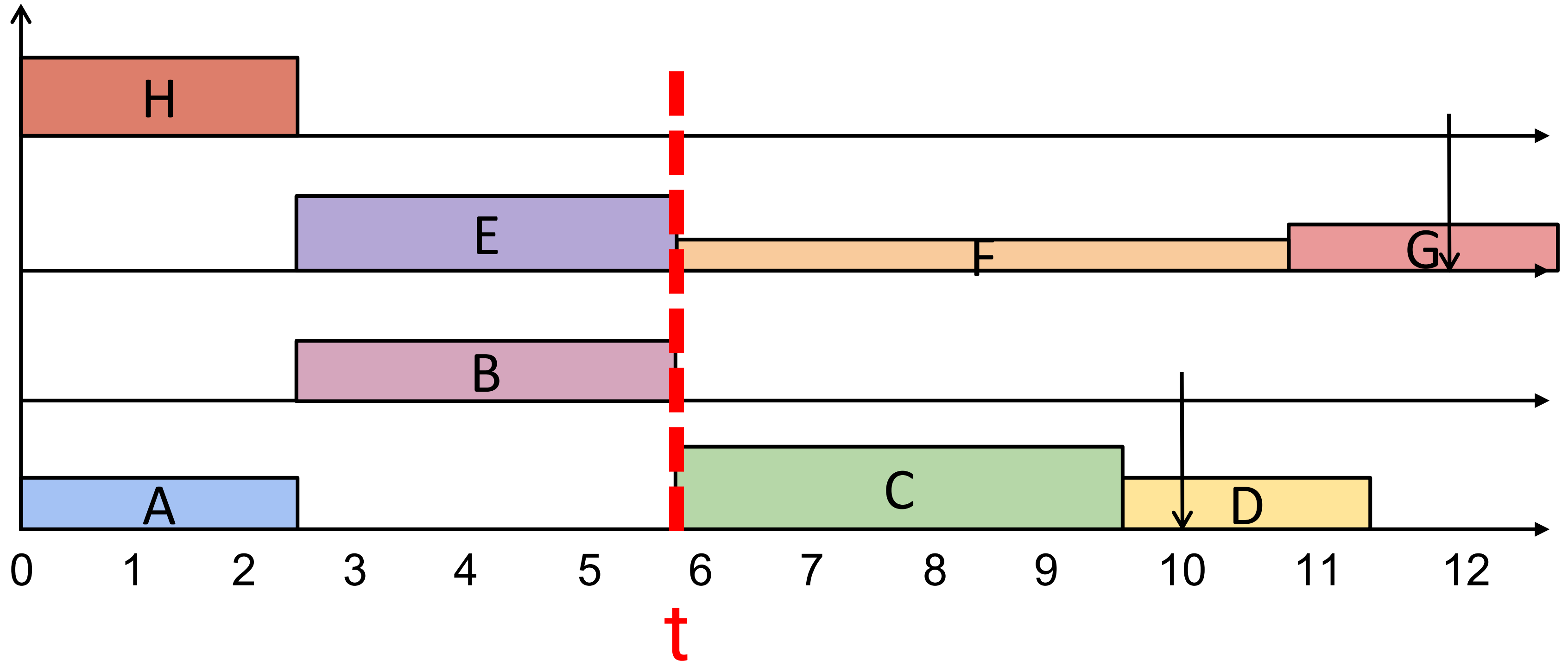


Step 3: Get next t, repeat

Schedule m tasks with earliest deadlines

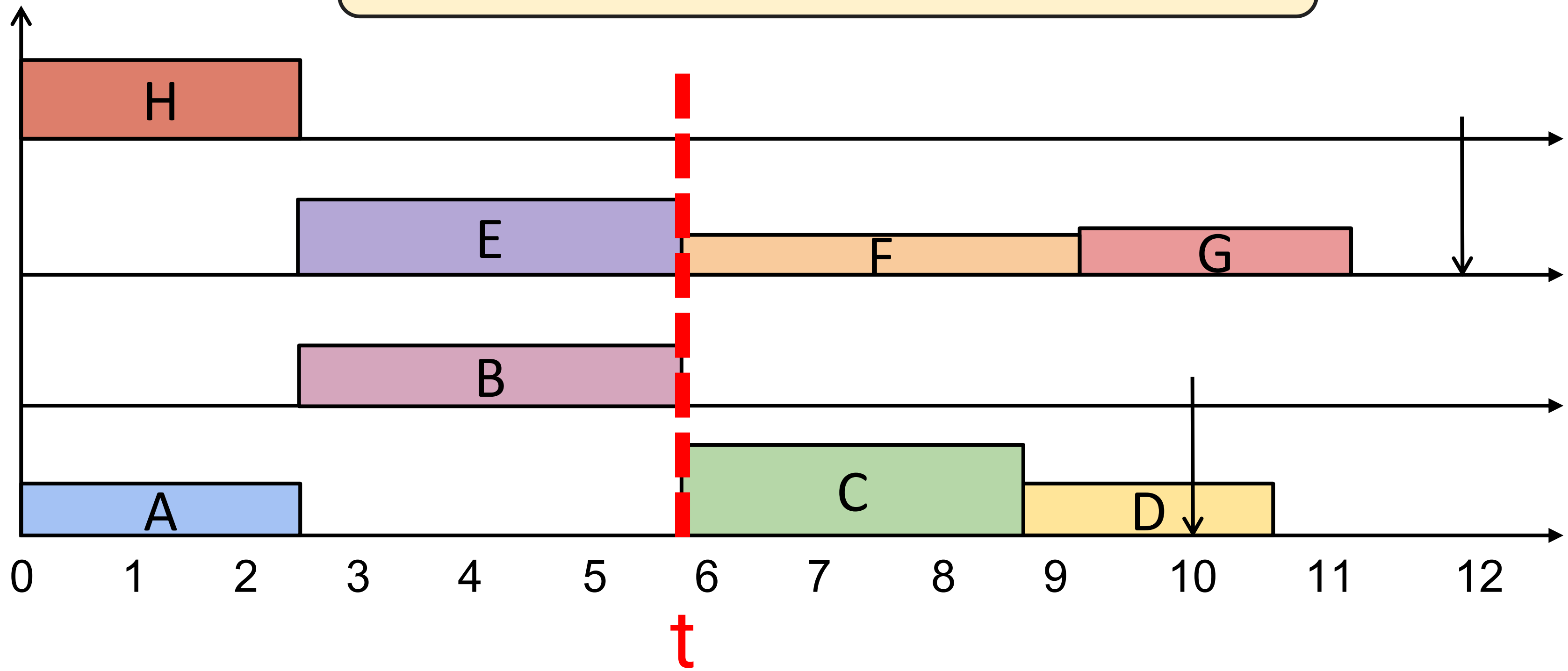


Step 3: Get next t, repeat

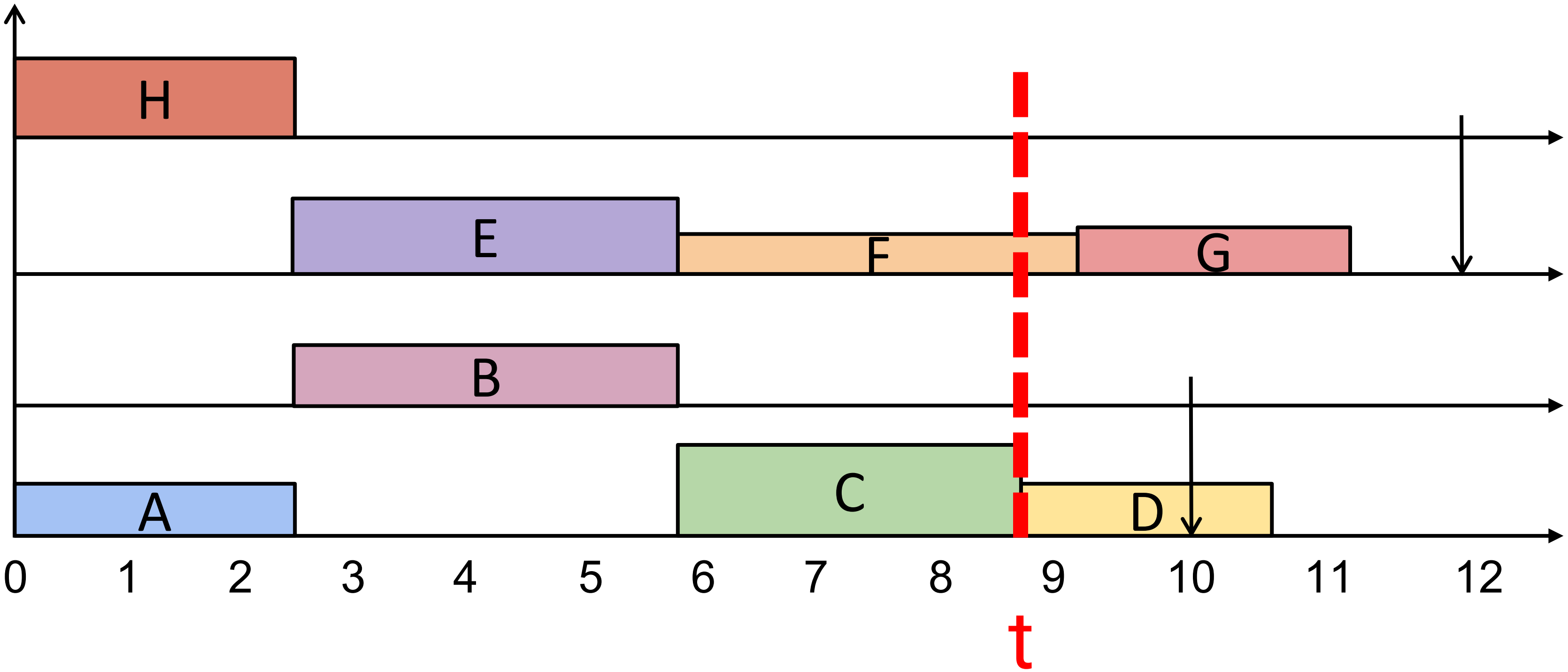


Step 3: Get next t, repeat

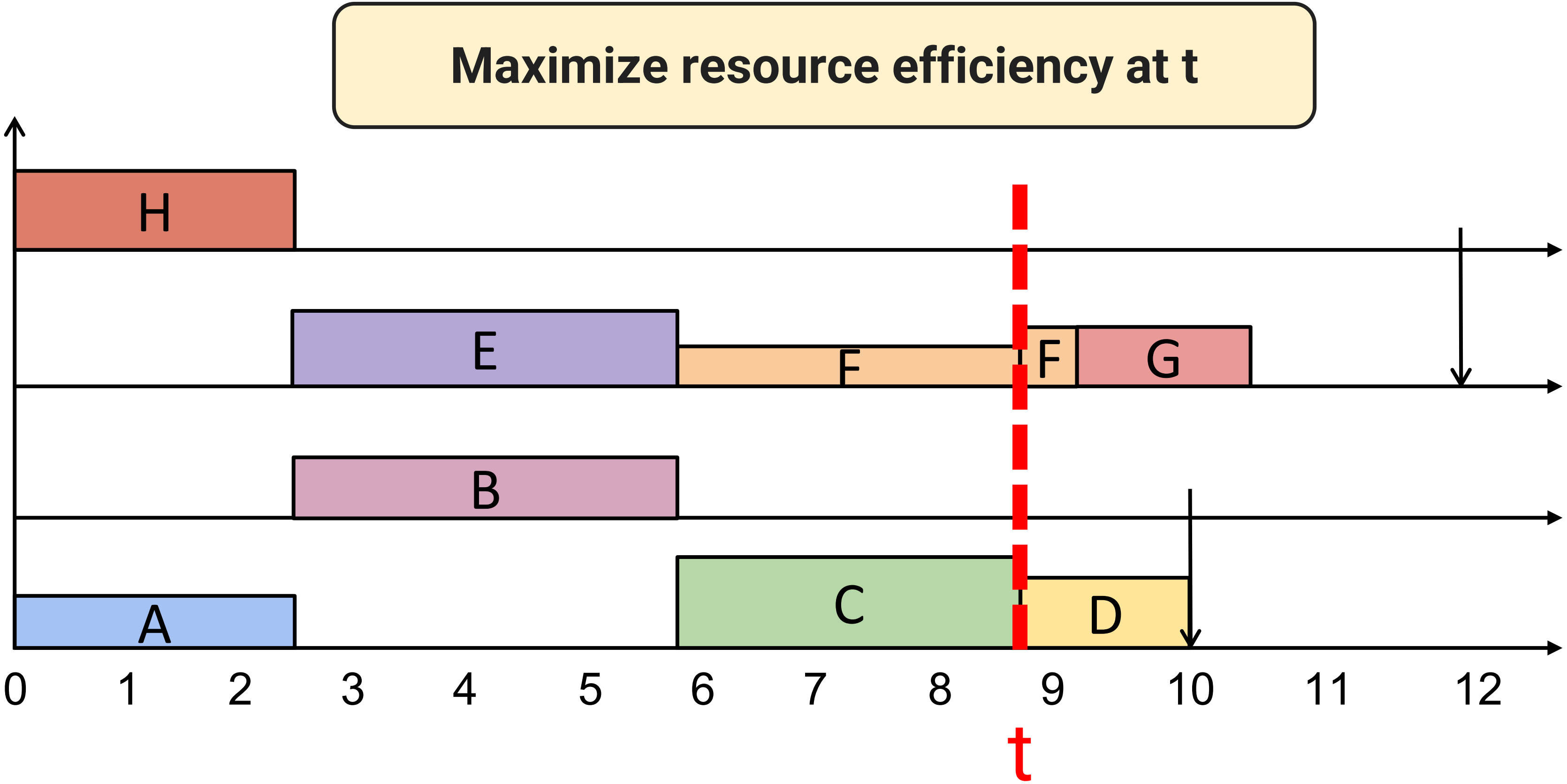
Maximize resource efficiency at t



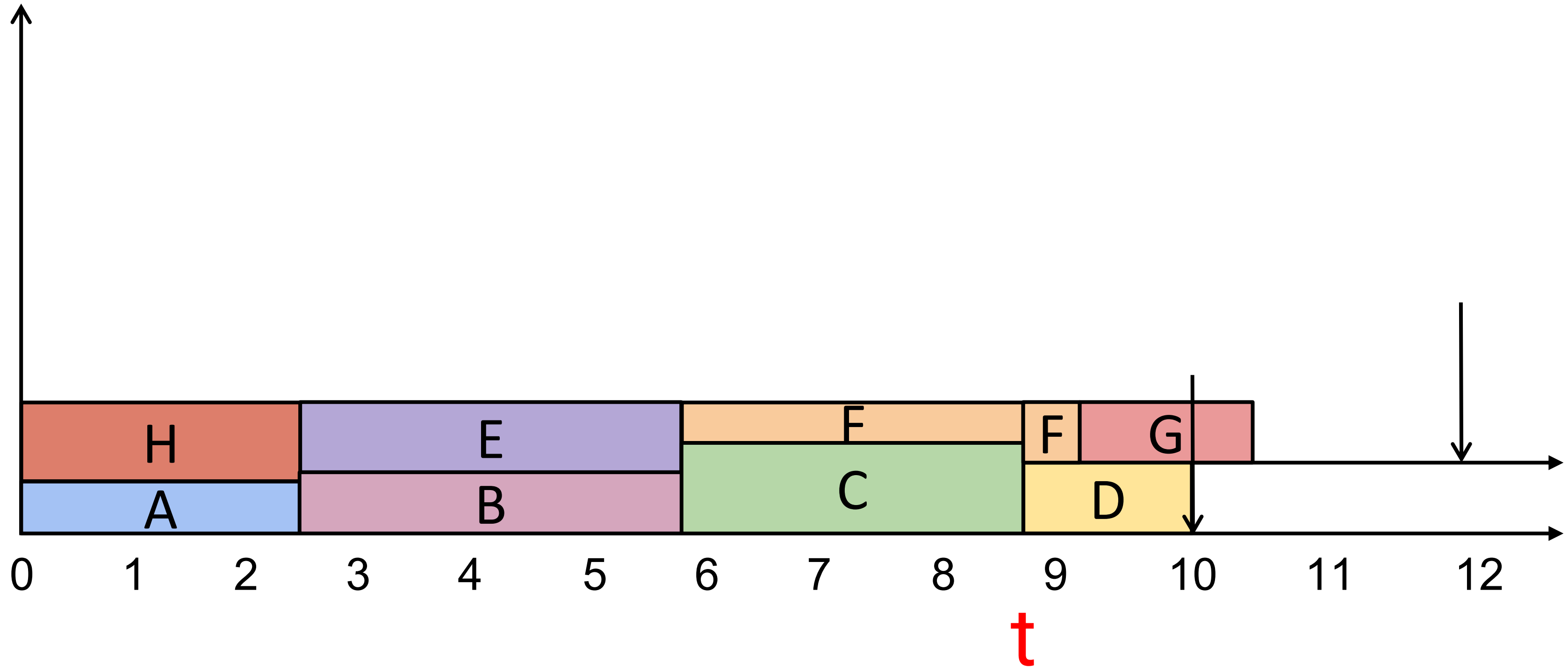
Step 3: Get next t, repeat



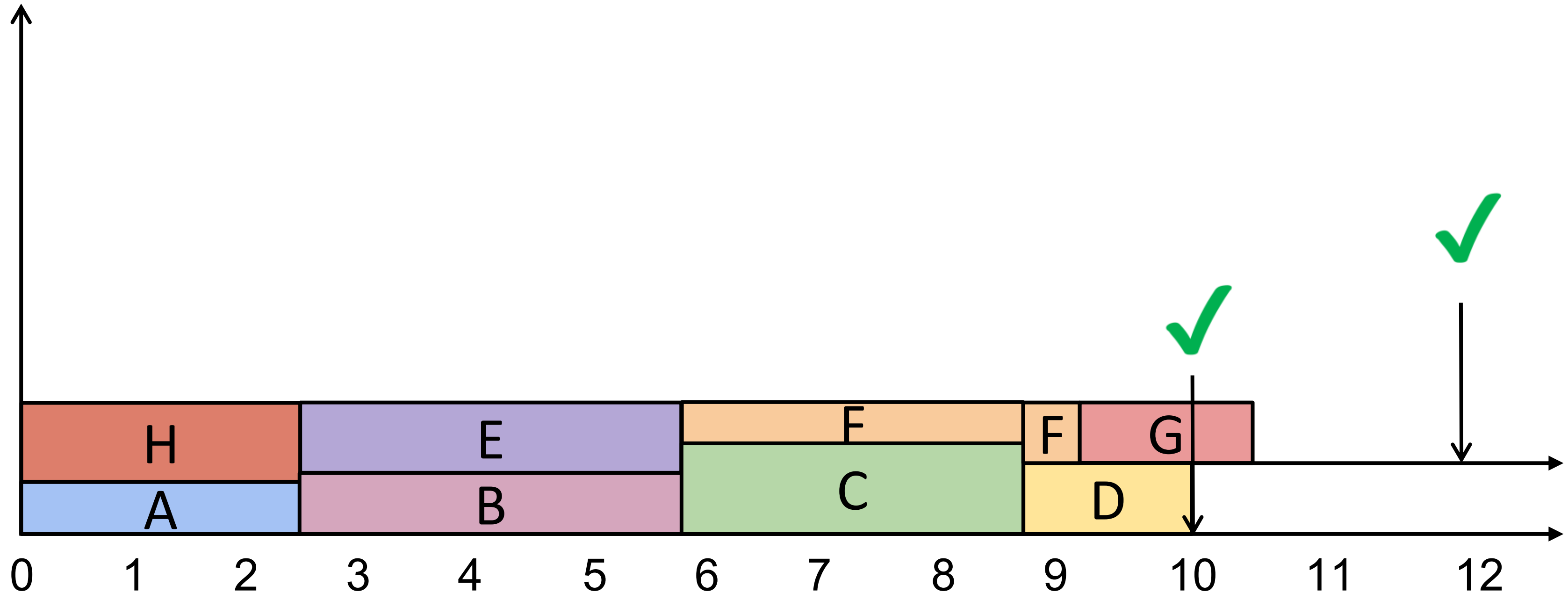
Step 3: Get next t, repeat



Step 4: Finish and squash onto 2 cores



Step 5: Check if schedulable



Talk Outline

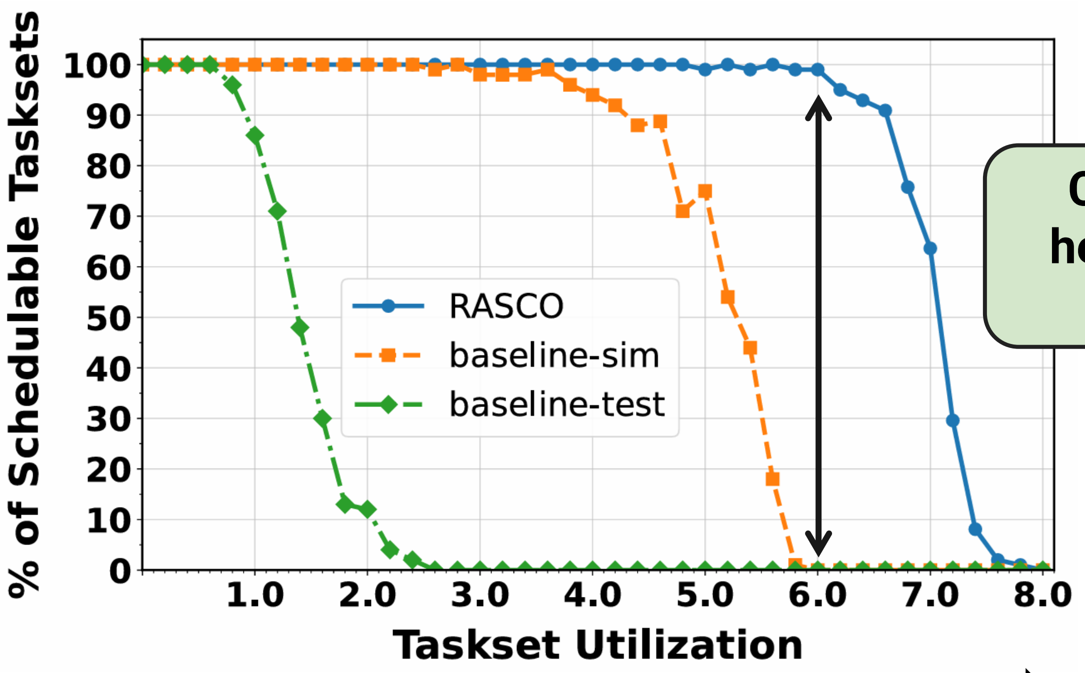
1. Introduction/Background
2. The Resource-Dependent Multi-Phase Model
3. Rasco: Resource Allocation and Scheduling Co-design
4. **Numerical Evaluation**
5. Prototype Evaluation and Overhead Accounting
6. Conclusion

Numerical Evaluation Setup

- Profiled benchmarks from PARSEC and SPLASH2x
- Used Intel's CAT and MemGuard to partition shared resources
- Constructed multi-phase models using changepoint detection
- Randomly generated 100 tasksets per utilization step [X. Dai. dag-gen-rnd]
- Ran Rasco on each taskset
- Compared **schedulability** and **latency** against a state-of-the-art deadline decomposition method using even partition of resources to cores
  - Schedulability test: baseline-test
  - Simulated schedule under global EDF: baseline-sim

Schedulability Results

Better results

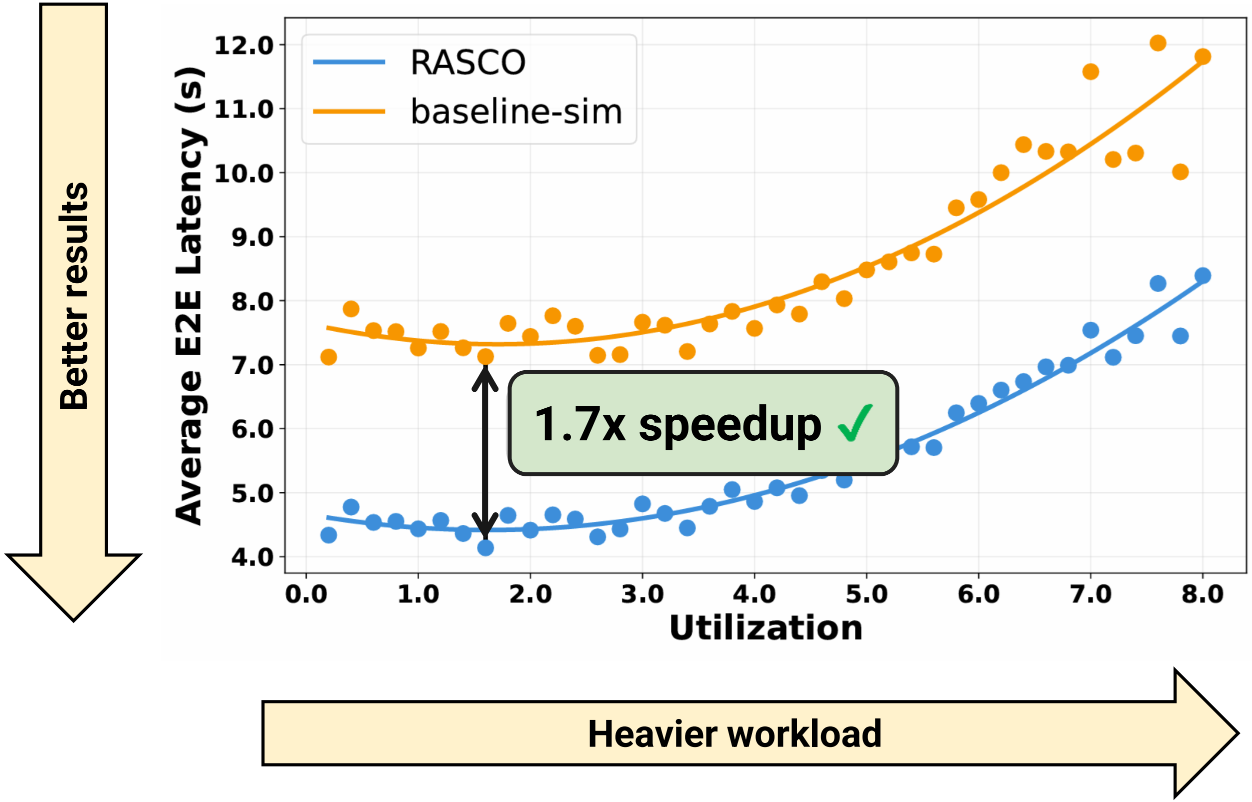


Heavier workload

Can schedule much heavier workloads on m=6 cores ✓

See paper for additional results.

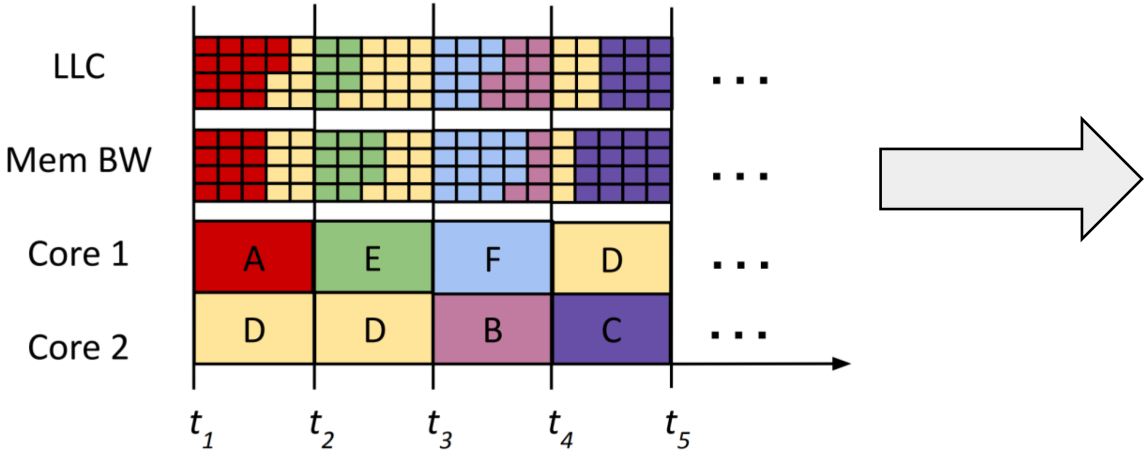
Latency Results



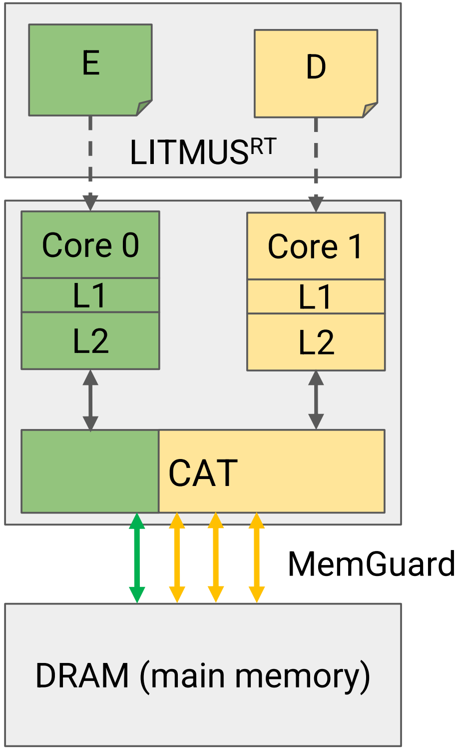
Talk Outline

1. Introduction/Background
2. The Resource-Dependent Multi-Phase Model
3. Rasco: Resource Allocation and Scheduling Co-design
4. Numerical Evaluation
5. **Prototype Evaluation and Overhead Accounting**
6. Conclusion

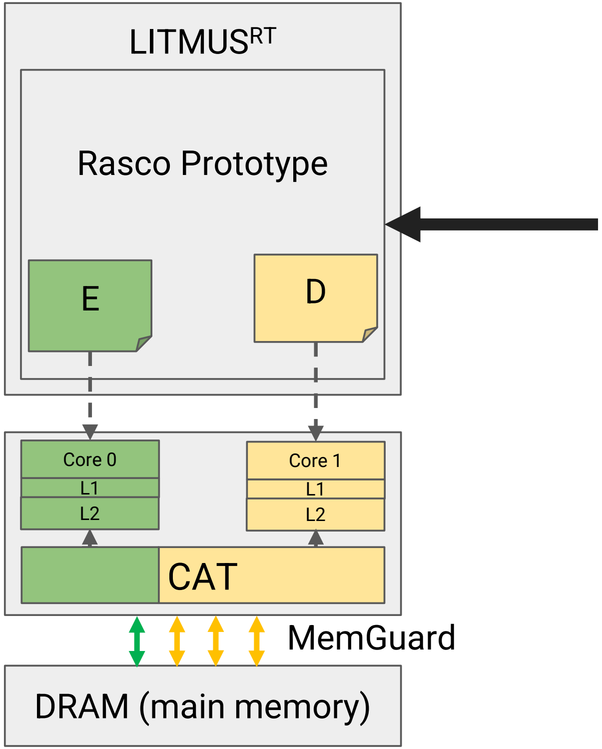
Prototype in LITMUS^{RT}



Prototype state at t_2



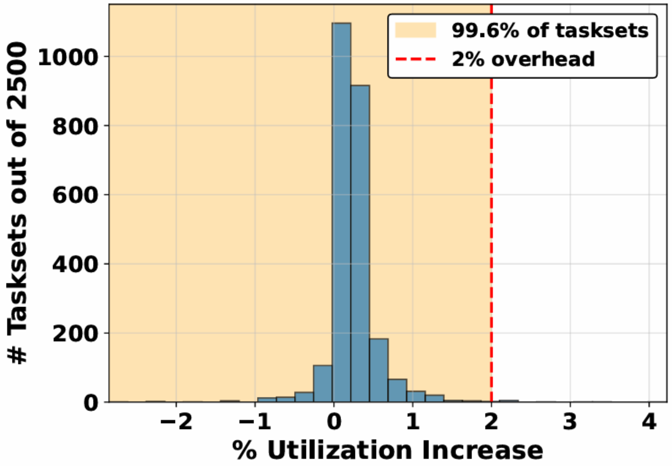
Rasco Prototype Features



1. Time-triggered table-driven scheduling in LITMUS^{RT}
2. Runtime reconfiguration of CAT + MemGuard in LITMUS^{RT}
3. Logic for handling job under-runs
4. Early releasing and work stealing

Overhead-Aware Rasco Extension

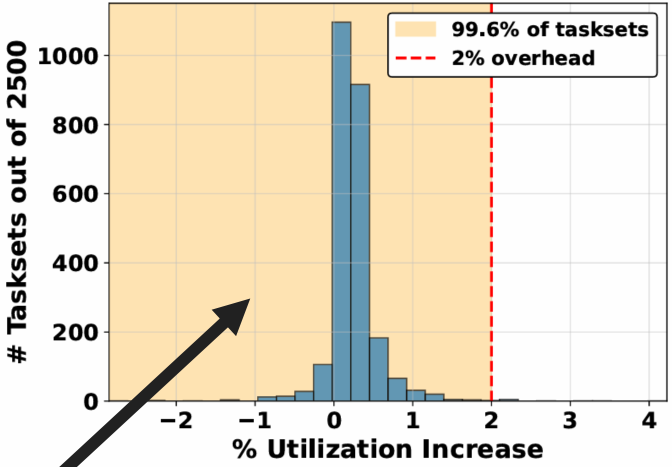
	Min (μs)	Mean (μs)	99th (μs)	Max (μs)
Rasco Scheduling	0.02	0.03	0.05	18.10
CAT + MemGuard	1.66	2.53	5.80	23.30



Observed small runtime overheads ✓

Overhead-Aware Rasco Extension

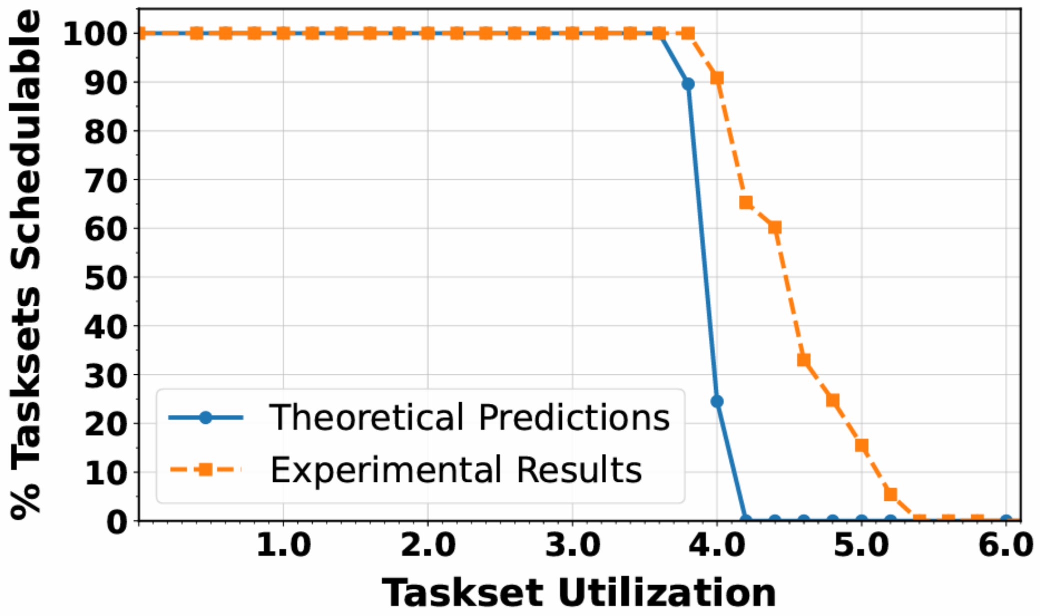
	Min (μs)	Mean (μs)	99th (μs)	Max (μs)
Rasco Scheduling	0.02	0.03	0.05	18.10
CAT + MemGuard	1.66	2.53	5.80	23.30



99.6% of the tasksets had less than 2% overhead ✓

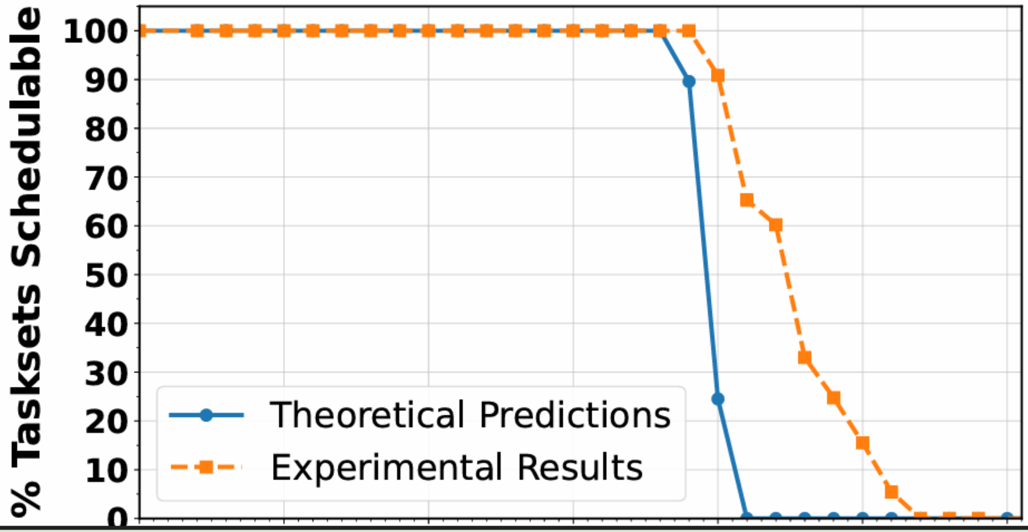
Empirical Evaluation on Prototype

- Ran Rasco's output schedules on our prototype



Empirical Evaluation on Prototype

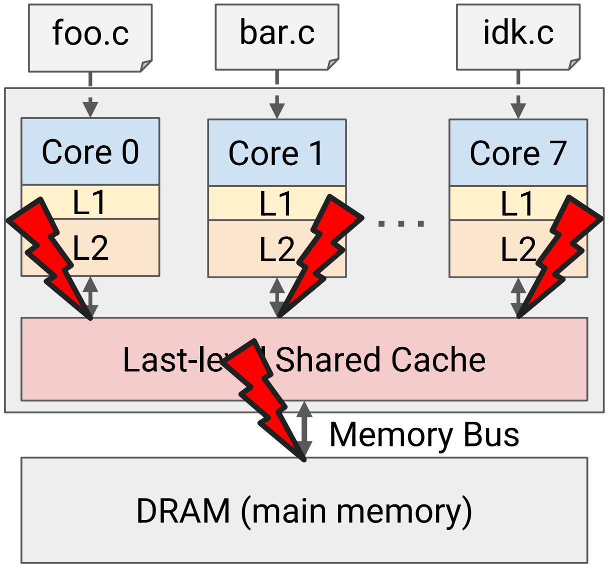
- Ran Rasco’s output schedules on our prototype

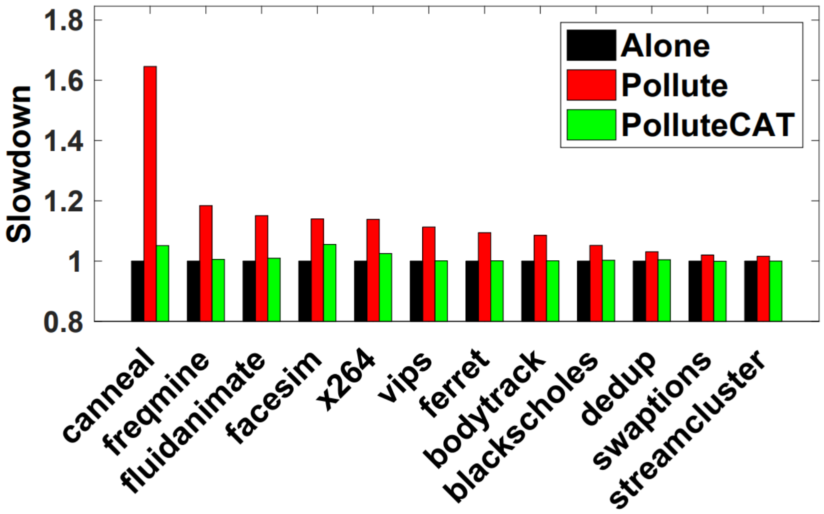


Empirical schedulability always exceeds theoretical guarantee ✓

Recap

- The move to multicore introduces challenges for timing analysis
- Resource contention → overly-conservative analysis → over-provisioning





Recap

- No prior work had achieved the **resource efficiency** of fine-grained **dynamic resource allocation** while providing **hard real-time guarantees**

	Resource aware?	DAG support?	Resource control?	Dynamic control?	Timing analysis?
Rasco	✓	✓	✓	✓	✓

Conclusion

- Proposed a **resource-dependent multi-phase model** which enables worst-case timing analysis under dynamic resource allocation
- Developed a resource allocation and scheduling **co-design algorithm** for DAG applications on multicore that improves
  - resource efficiency,
  - latency, and
  - schedulability
- Implemented a **prototype** of Rasco to evaluate the safety and utility of our approach in a real-time operating system

Conclusion

- Proposed a **resource-dependent multi-phase model** which enables worst-case timing analysis under dynamic resource allocation
- Developed a resource allocation and scheduling **co-design algorithm** for DAG applications on multicore that improves
  - resource efficiency,
  - latency, and
  - schedulability
- Implemented a **prototype** of Rasco to evaluate the safety and utility of our approach in a real-time operating system

Thank you!

https://github.com/abbyeisenklam/Rasco