# Real-time Virtualization, Concurrency Platforms, and Middleware: from Devices to Edge Servers

TECoSA Seminar, Thursday, March 4, 2021
ITM School, KTH Royal Institute of Technology

Chris Gill
Professor of Computer Science and Engineering
Washington University in St. Louis

# From devices to edge servers

- Part I: a view of cyber-physical scalability
  - A spectrum of platforms and timing requirements
  - Some illustrative applications and their challenges
- Part II: progress so far, and its implications
  - From fixed priority to federated parallel real-time scheduling
  - Addressing capacity limitations using task details
  - Exploiting mixed-criticality & elasticity for utilization trade-offs
- Part III: emerging needs and opportunities
  - Integrating mixed-criticality, elasticity for graceful degradation
  - Towards scalable cyber-physical platform frameworks

# Part I: a view of cyber-physical scalability

- A spectrum of platforms and timing requirements
  - » Devices, micro-and-mini-boxes, servers, clusters, edge clouds
  - » Requirements and assumptions about tasks and time scales

- Some illustrative applications and their challenges
  - » Real-time hybrid simulation for earthquake engineering
  - » Airdrop (and other stress) testing
  - » Catoptric surfaces for redirecting light in occupied spaces

# A spectrum of platform resource granularity

From devices, to micro-boxes, to mini-boxes, to servers, to server clusters to server-cluster based edge clouds to clouds

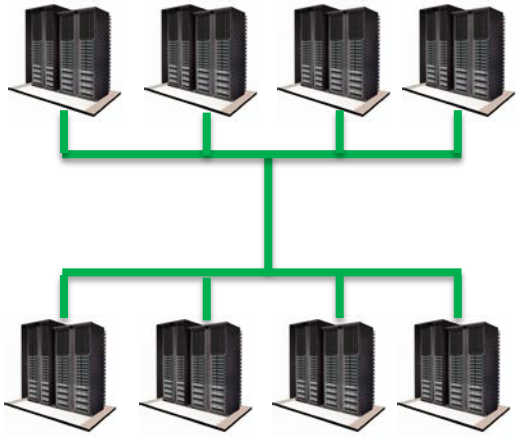
Micro-box: connects to device, receives & sends data (e.g., RPi 0)

Mini-box: processes data on e.g., Linux with ~4 cores + GPU (e.g., RPi 3 B+)

Server: Linux on 16 to 64 cores + GPU + FPGA, runs simulations, other

Cluster: multiple servers on shared memory, runs larger scale computations

Edge cloud: multiple server clusters on local network, runs even larger and/or more numerous computations



Virtualization enables even finer grained resource partitioning

M. Xu, L. Phan, O. Sokolsky, S. Xi, C. Lu, C. Gill, I. Lee, "Cache-Aware Compositional Analysis of Real-Time Multicore Virtualization Platforms", Real-Time Systems 51(6), 2015

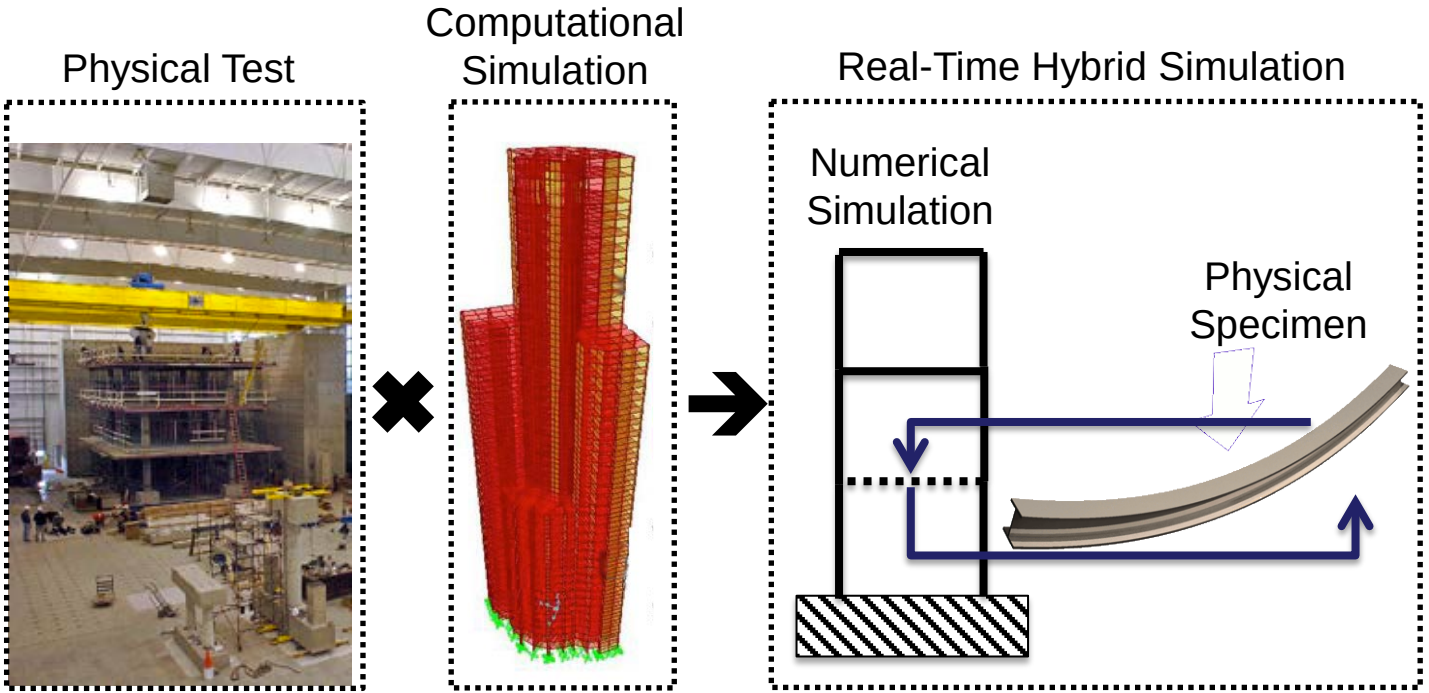
# Requirements and assumptions

- About tasks
  - Data and control flow dependences may form a general DAG
  - Some tasks are more critical than others (safety vs. mission)
  - Any performance degradation must be criticality-aware
  - Graceful degradation of lower-criticality performance matters

- About time-scales
  - Recurring deadlines down to milliseconds must be met end-to-end for cyber-physical integrity at the speed of sound
  - Multi-timescale measurement and enforcement is needed

# Application: real-time hybrid simulation (RTHS)

Finite element model simulation with **real-time** guarantees, integrated **safely** with control and physical sensing and actuation



# Application: airdrop (and other stress) testing

- Airdrop testing for integrity of equipment/supply cargo where size, weight, and power matter
- Arduino/RPi0 micro-boxes collect data on-board on-line
- Linux Raspberry Pi 3 mini-boxes coordinate test itself
  - Do some on-board processing
  - Stream results back to test crew
  - Communicate test status
  - Manage test configuration/modes

2010 Haiti earthquake relief airdrop
(from https://en.wikipedia.org/wiki/Airdrop)

# Application: catoptric surface redirects light

- Hundreds of mirrors on individual pan-tilt units
- Controlled by tens of Arduino micro-boxes
- Several Linux Raspberry Pi mini-boxes will be added
  - Overall system management
  - On-line self-calibration
  - Coordination/feedback loops
  - Scripted effects sequences
  - Personalized lighting services

https://samfoxschool.wustl.edu/news/13923

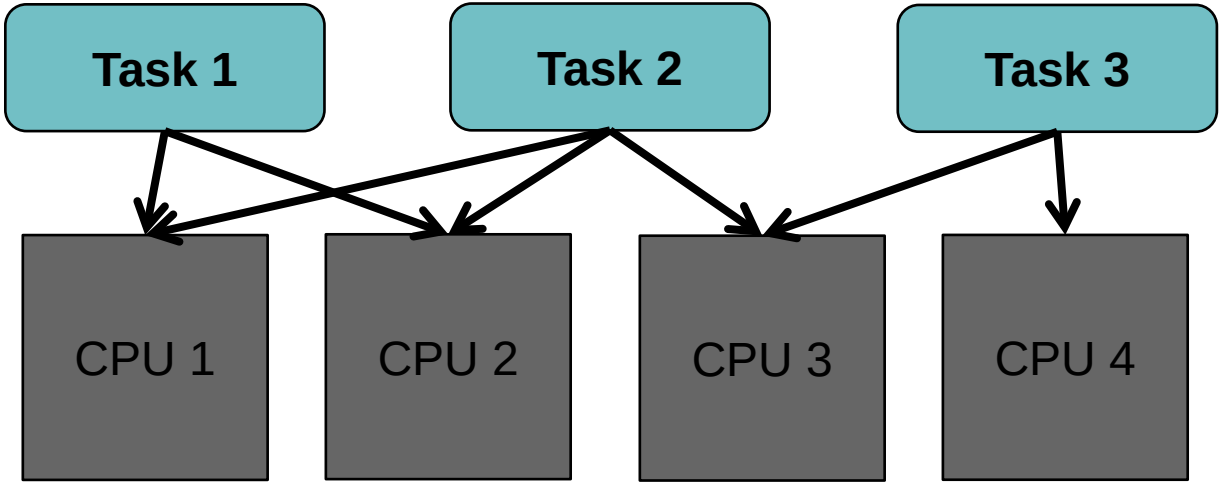
Steinberg Hall, Washington University in St. Louis

# Part II: progress so far, and its implications

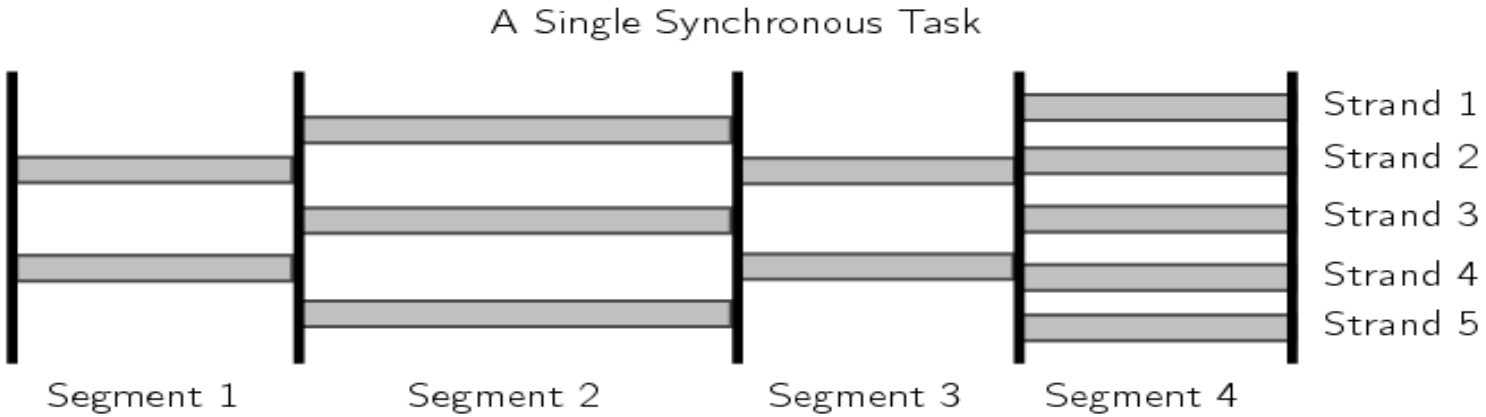
- From fixed priority to federated real-time scheduling
  - » Limitations of a narrow thread-scoped (middleware-like) view
  - » Scaling dedicated resources for timing assurance, to a limit
- Addressing capacity limitations using task details
  - » Avoids over-reservation of dedicated resources
  - » Exploits task structure, improves packing, allows splitting
- Mixed-criticality & elasticity for utilization trade-offs
  - » Mixed-criticality supports tiered degradation under overload
  - » Elasticity avoids overload by shifting rates and/or workloads

# Middleware-like parallel real-time systems view

- A task may need >1 processors to meet deadlines
  - » Precedence constraints among subtasks complicate this
- Schedulability analysis based on tasks' ***computational requirements*** and ***deadline constraints***

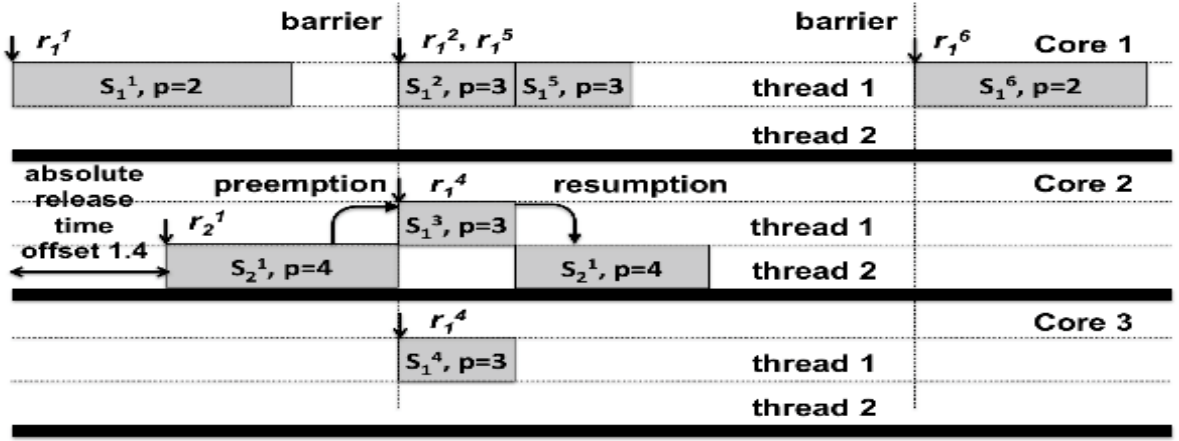


# Parallel synchronous (fork-join) task model




- Sequences of segments (each with parallel strands)
- Can enforce strands' priorities via Linux (with the RT_PREEMPT patch) real-time thread priorities
- Strands barrier synchronize at end of a segment
  - » Implemented using Linux threads and futexes

# Example execution trace: strands on cores

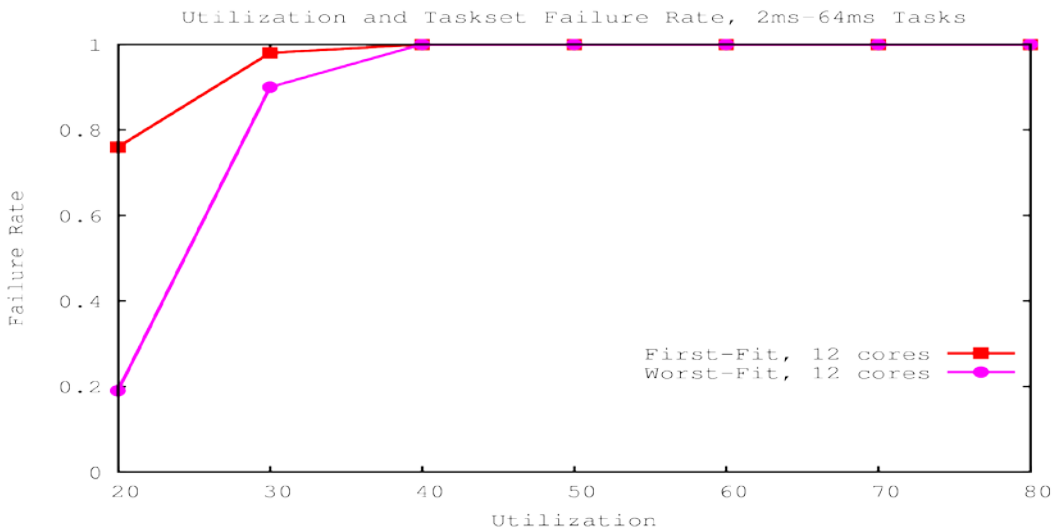



- Fischer Baruah Baker First Fit Decreasing (FBB-FFD) bin packing of stands onto cores (alternatively, worst fit)
  - » Prioritize strands by relative deadline (earlier➔higher)

A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core Real-Time Scheduling for Generalized Parallel Task Models", RTSS 2011

# Scalability limitations

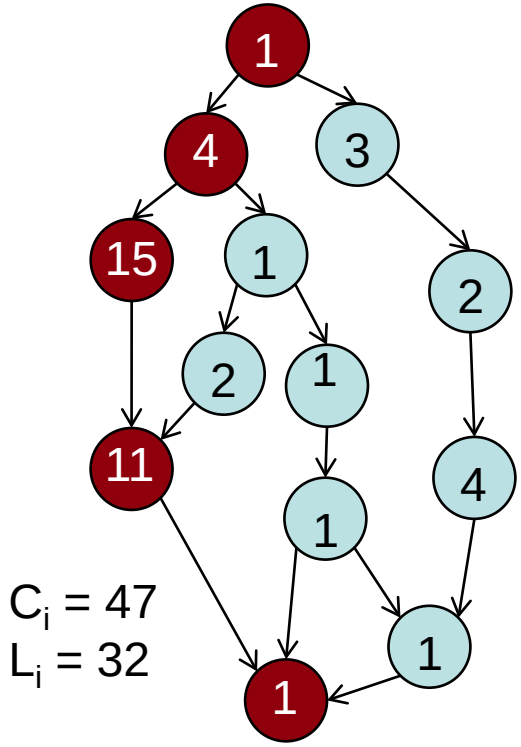
- At 500Hz (2 msec periods), 12-core task sets failed to meet theoretical bound
  - » Largely due to DAQ I/O bottlenecks
  - » Hand-crafted implementation ran tasks with 4ms periods feasibly on 12 cores




D. Ferry, J. Li, M. Mahadevan, K. Agrawal, C. Gill, and C. Lu,
"A Real-Time Scheduling Service for Parallel Tasks", RTAS 2013

# Towards general parallel real-time systems

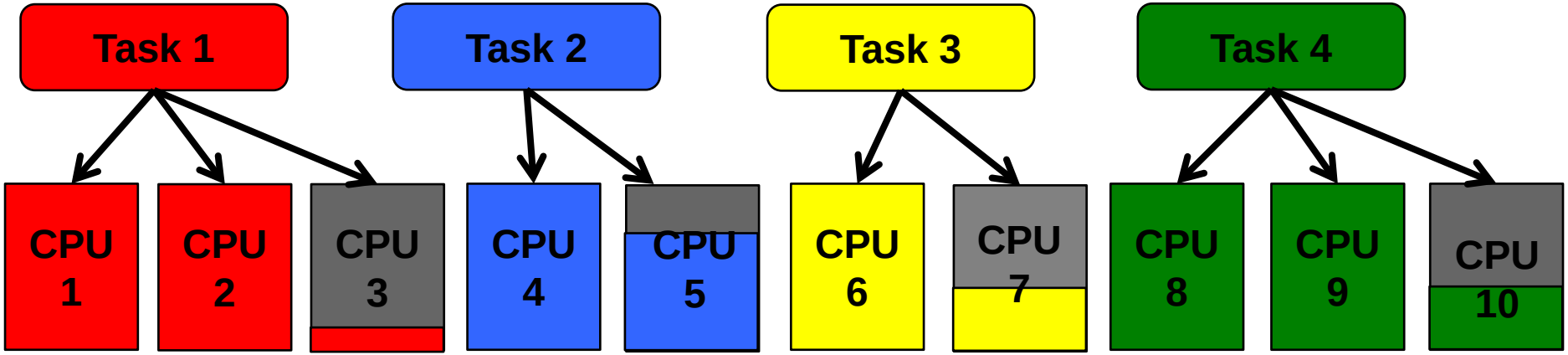
- General DAG Model for Task $\tau_i$
  - » Node: subtask $\tau_{i,j}$ (weight=$C_{i,j}$)
  - » Edge: dependence between nodes
- Span (Critical Path Length) $L_i$:
  - » Highest-weight chain of nodes
  - » Execution time on ∞ CPUs
- Work (Computation Time) $C_i$:
  - » Execution time on 1 CPU




$C_i = 47$
$L_i = 32$

# Federated scheduling (FS)

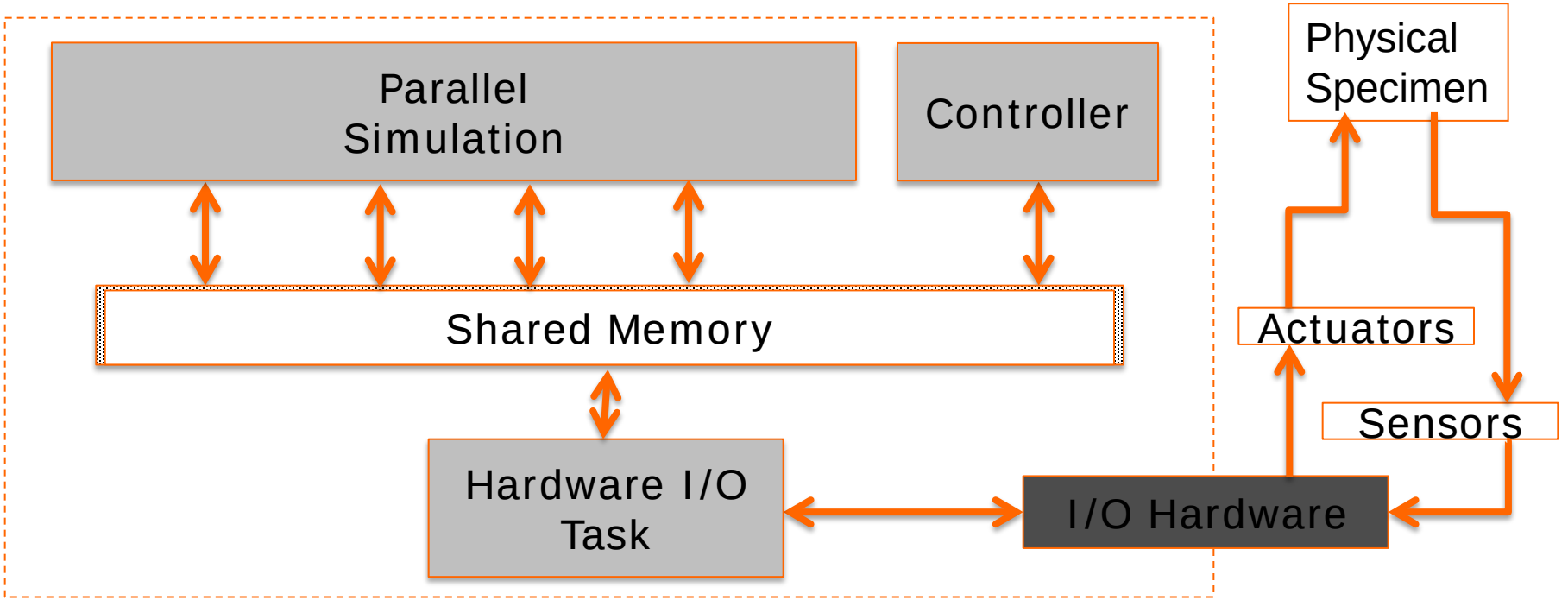
- In general a parallel task requires $\frac{C_i - L_i}{D_i - L_i} = \mathbf{A_i} + \boldsymbol{\varepsilon_i}$ CPUs to guarantee timely completion ($A_i$ is an integer, $0 \le \varepsilon_i \le 1$)
- Federated scheduling allocates $\left\lceil \frac{C_i - L_i}{D_i - L_i} \right\rceil = \begin{cases} A_i & \varepsilon_i = 0 \\ 1 + A_i & \varepsilon_i > 0 \end{cases}$ CPUs




J. Li, J.-J. Chen, K. Agrawal, C. Lu, C. Gill, A. Saifullah,
"Analysis of Federated and Global Scheduling for Parallel Real-Time Tasks," ECRTS 2014

# CyberMech parallel RTHS architecture

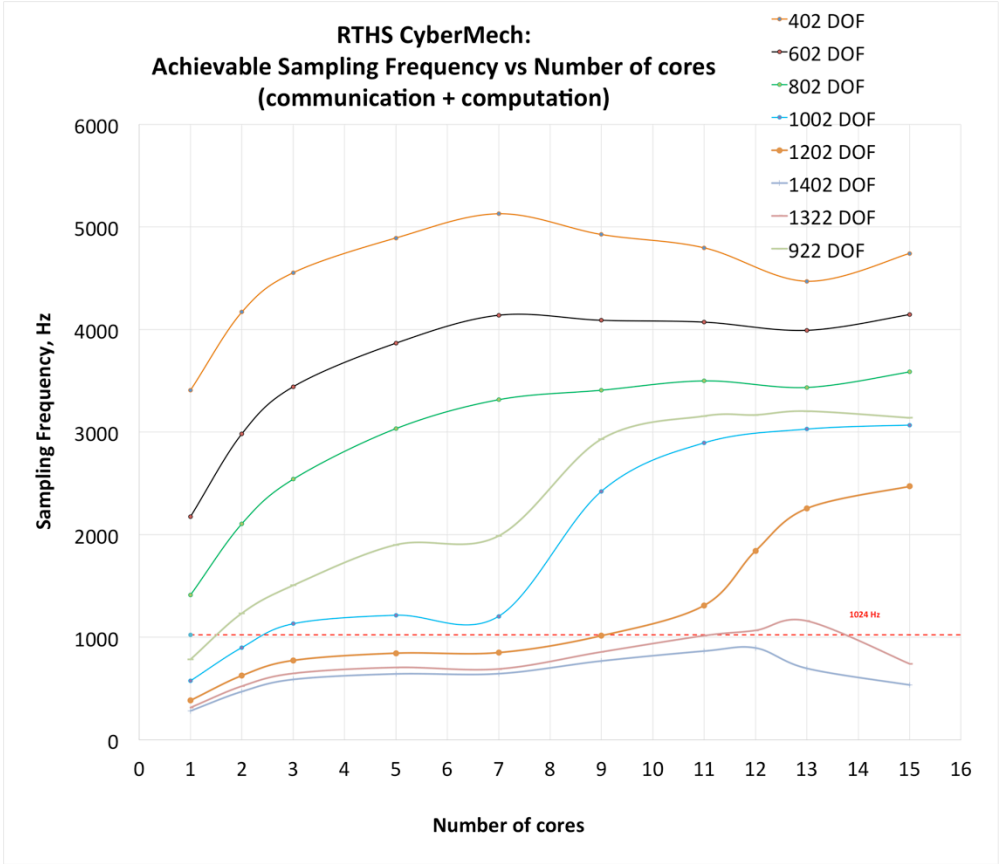
Federated scheduling in OpenMP/Cilk Plus, safe multithreading removes I/O bottlenecks




D. Ferry, G. Bunting, A. Maghareh, S. Dyke, A. Prakash, K. Agrawal, C. Gill, C. Lu, "Real-time System Support for Hybrid Structural Simulation," EMSOFT 2014

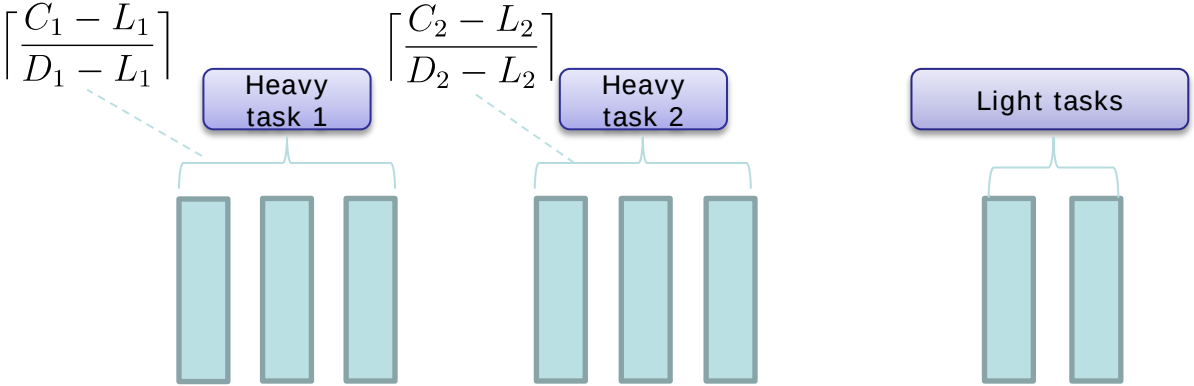
# Scalable up to a point

- Realistic scale experiments
  - » Using hydraulic actuators at Purdue University Bowen Lab
- First ever fully parallel RTHS
  - » Experiments with up to 1322 degree-of-freedom simulations on 13 (of 16) cores at 1024Hz
  - » Suitable for 9-story buildings
- Communication costs limit this
  - » Especially between threads on cores in different chip sockets



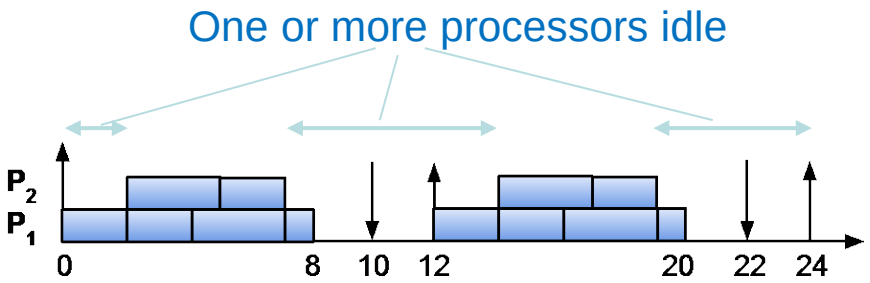
# Heavy versus light parallel real-time tasks

- Federated scheduling classifies tasks as
  - » Heavy tasks, $U_i \geq 1$, require parallel execution
  - » Light tasks, $U_i < 1$, can execute sequentially



# Federated scheduling can be inefficient

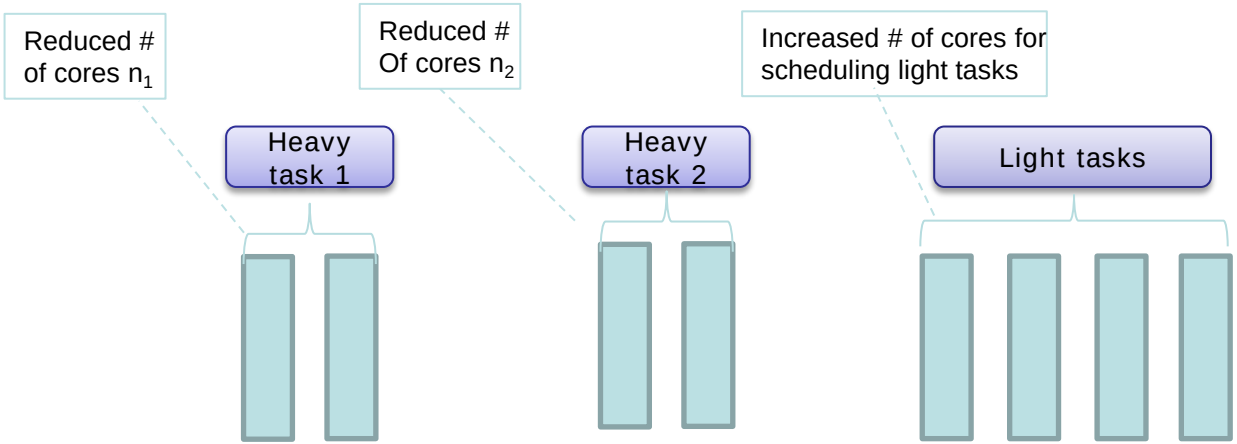
- Processors allocated to a heavy task can't be used by any other task when idling




- The number of processors allocated to each heavy task may be pessimistic
  - » Computed based on its worst-case DAG: $m_i = \lceil \frac{C_i - L_i}{D_i - L_i} \rceil$

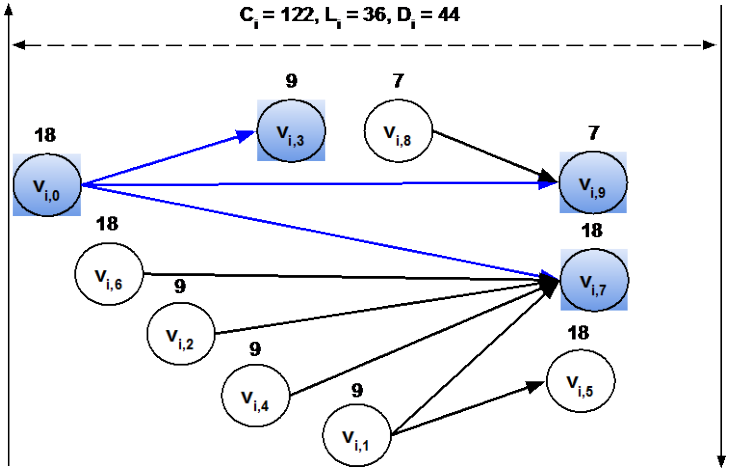
# Idea: exploit DAG structure to improve utilization

- Use internal graph structure, not just basic parameters $C_i$, $L_i$, $D_i$, for a more exact calculation of # cores needed by heavy tasks
- Reallocate extra cores for light tasks[1] (or elastic or mixed-critical use)




[1] Brandenburg et al., “Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations”, RTSS 2016

# New algorithm for scheduling heavy tasks



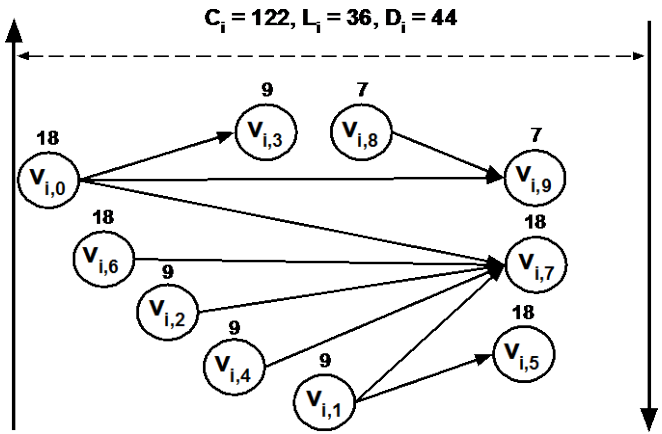

***Subgraph*** of $v_{i,0}$: $\{v_{i,0}, v_{i,3}, v_{i,7}, v_{i,9}\}$

***Subgraph work*** of $v_{i,0}$ = 18 + 9 + 18 + 7 = 52

1. For a given number $m_i$ (starts at $\lceil C_i/D_i \rceil$) cores for $\tau_i$; for each iteration
   a. Choose $m_i$ vertices with ***greatest subgraph work*** to schedule
   b. Schedule them until some unchosen vertex has subgraph work greater than that of a chosen one
      - ***Vertices can be preempted ("task splitting")***
2. If successful, return the schedule. If failed, increase $m_i$ by 1 core and go to step 1

S. Dinh, C. Gill, K. Agrawal, "Efficient Deterministic Federated Scheduling for Parallel Real-time Tasks," RTCSA 2020

# Comparison with other federated approaches



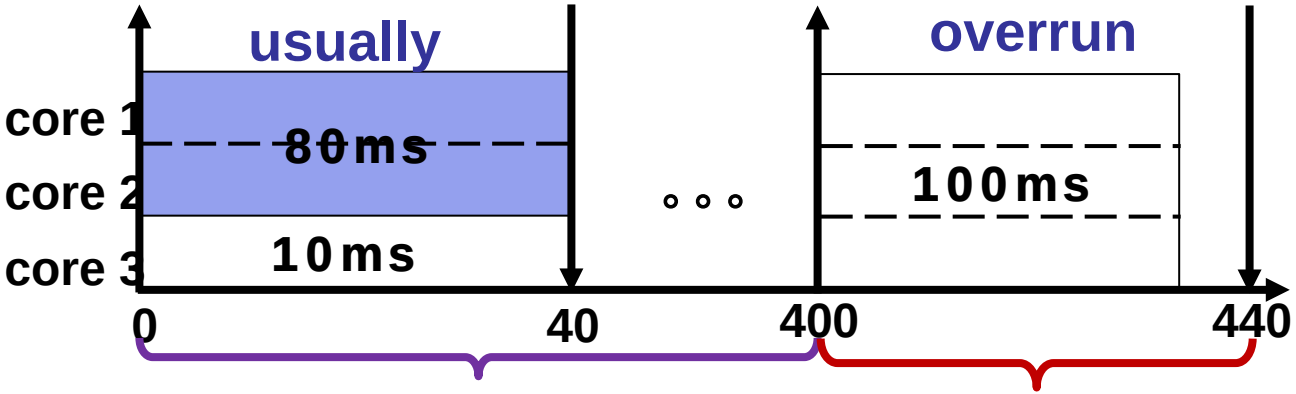

- The new algorithm needs **3** processors
  - Fully exploits tasks' DAGs
- Original federated scheduling needs $\lceil (C_i - L_i) / (D_i - L_i) \rceil = 11$ processors
  - Only considers tasks' parameters $C_i$, $L_i$, and $D_i$
- Semi-federated scheduling[2] needs $\lfloor (C_i - L_i) / (D_i - L_i) \rfloor = 10$ processors
  - Partially consider tasks' DAGs for its runtime dispatcher

[2] Jiang et al., "Semi-Federated Scheduling of Parallel Real-Time Tasks on Multiprocessors," RTSS 2017

# Mixed-criticality improves resource efficiency

**Mixed-criticality scheduling of parallel real-time tasks**

Provides different levels of real-time guarantees



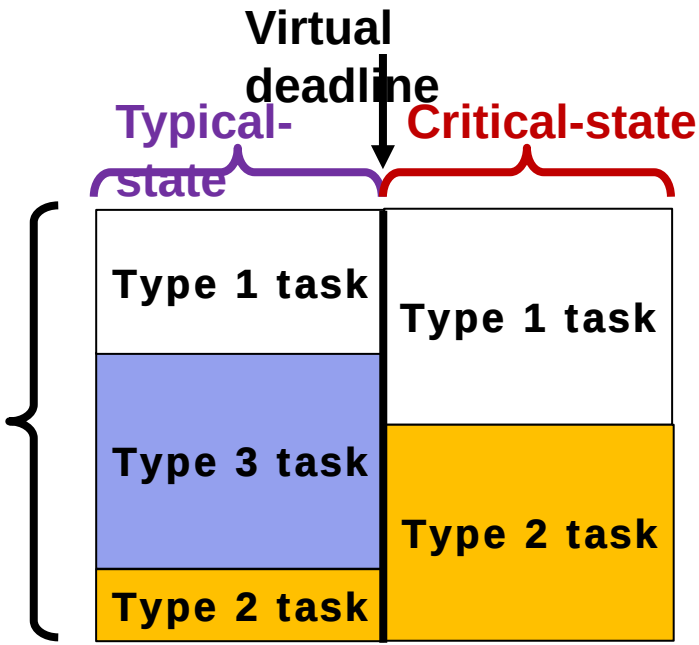

**Most cases:**
guarantee that **both high and low-criticality tasks** meet deadlines

**Very rare cases:**
only guarantee that **high-criticality tasks** meet deadlines

# Mixed-criticality federated scheduling

Classify tasks into N types (by criticality)

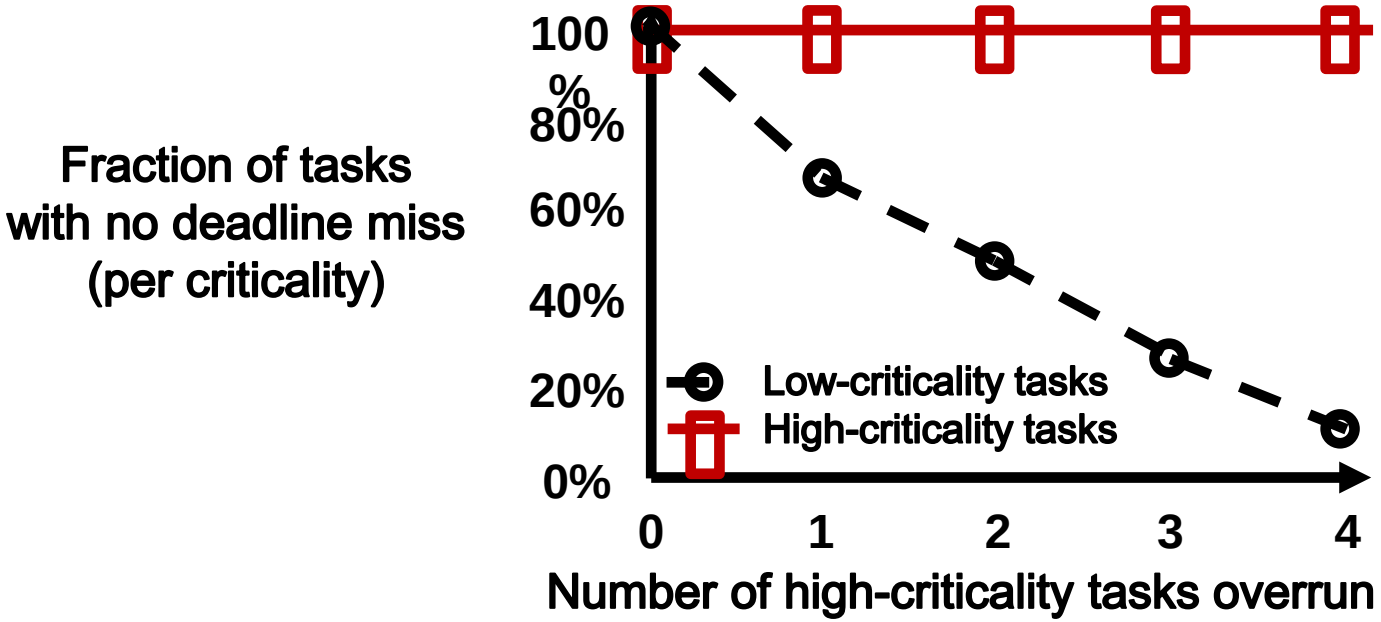
For each task in each type, calculate/assign
(1) virtual deadline for finishing
(2) dedicated cores in typical state
(3) dedicated cores in critical state




J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Lu, C. Gill,
"Mixed-Criticality Federated Scheduling for Parallel Real-time Tasks," RTAS 2016

# Selective degradation by criticality

MCFS scheduler for OpenMP tasks performs similarly in both numerical and empirical evaluations



# Discretely elastic parallel real-time tasks

- Task $\tau_i$ has elastic coefficient $E_i$ and $k_i$ unique modes of operation
- Each mode of operation $j$ $(1 \leq j \leq k_i)$ has a triple:
  - Period (and implicit deadline) $T_i^{(j)}$
  - Work $C_i^{(j)}$
  - Span (critical path length) $L_i^{(j)}$

# Discretely elastic parallel real-time tasks

- Selecting mode of operation per task is weakly NP Hard
- Pseudo-polynomial time dynamic programming algorithm selects one mode of operation per task to

$$\textbf{minimize} \sum_{i=1}^{n} \frac{1}{E_i} \left( U_i^{(max)} - U_i \right)^2$$

$$\textbf{such that } \forall_i \left( U_i^{(min)} \leq U_i \leq U_i^{(max)} \right) \wedge m \geq \sum_{i=1}^{n} \left\lceil \frac{C_i - L_i}{T_i - L_i} \right\rceil$$

n is the number of tasks and m is the number of processors

# An expanding model of elasticity

- Rate-elastic parallel real-time tasks
  - » Minimum inter-arrival time (period) can be varied elastically
  - » Task's work and span are fixed
- Computationally elastic parallel real-time tasks
  - » Sum of subtask execution times (work) can be varied elastically
  - » Task's span, period, and (implicit) deadline are fixed

J. Orr, C. Gill, K. Agrawal, S. Baruah, C. Cianfarani, P. Ang, C. Wong
"Elasticity of Workloads and Periods of Parallel Real-Time Tasks," RTNS 2018

- **Combined elasticity of parallel real-time tasks**
  - » Tasks' periods, execution times (work), spans can ***all*** be varied

J. Orr, J. Condori, C. Gill, S. Baruah, K. Agrawal, S. Dyke, A. Prakash, I. Bate, C. Wong, S. Adhikari, "Elastic Scheduling of Parallel Real-Time Tasks with Discrete Utilizations," RTNS 2020

# Benefits of combined elasticity

- A wider range of adaptive tasks can be modeled
- Encodes specific task details (e.g., harmonic rates)

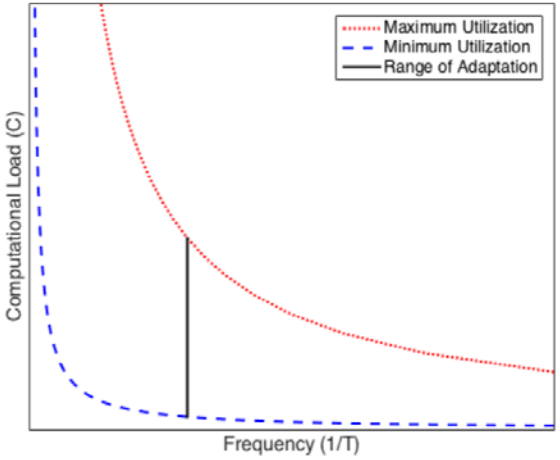



Figure 1: Continuous Computationally-Elastic Task

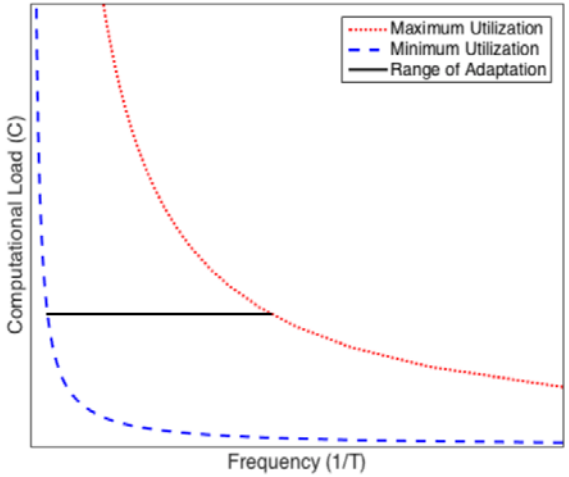



Figure 2: Continuous Period-Elastic Task

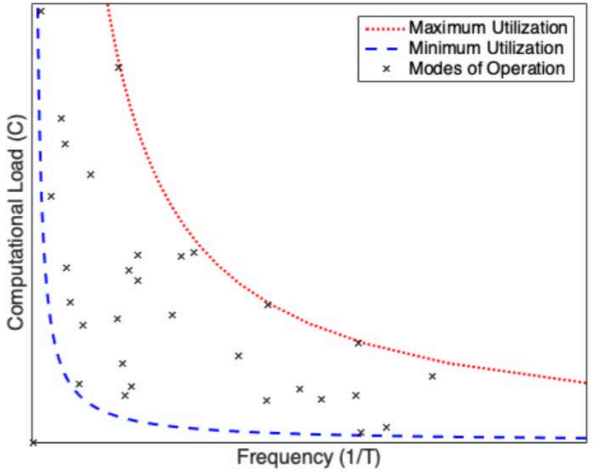
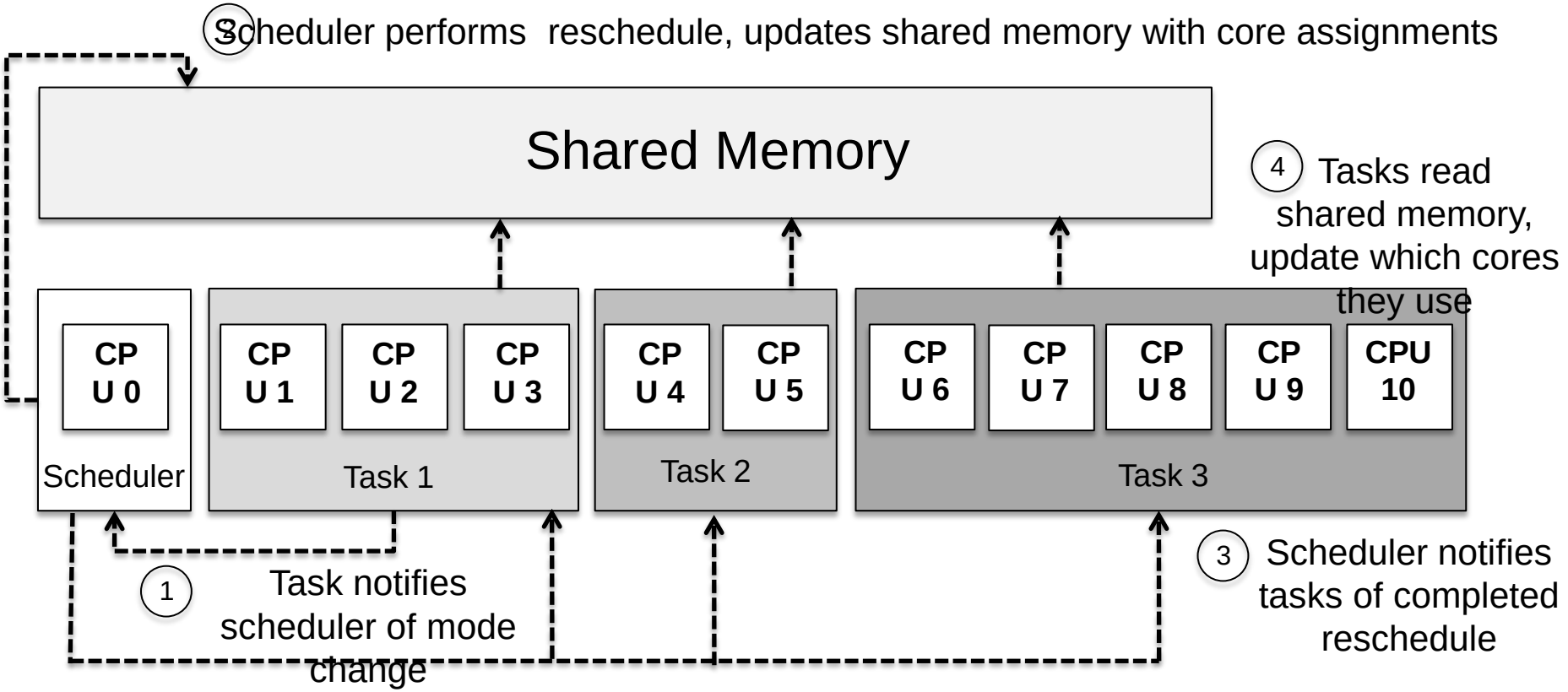


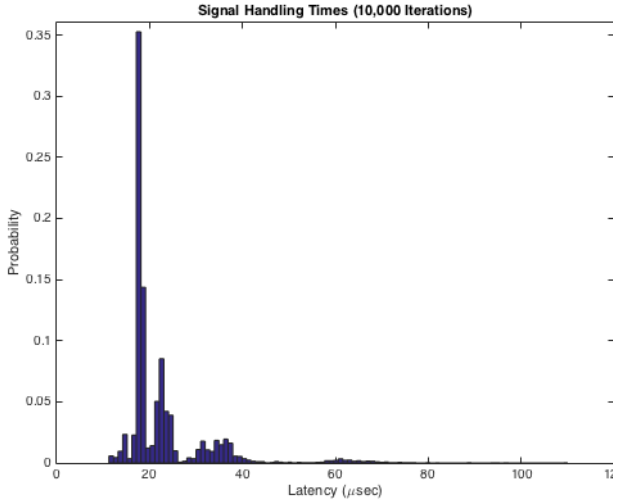

Figure 3: Discrete Combined-Elastic Task

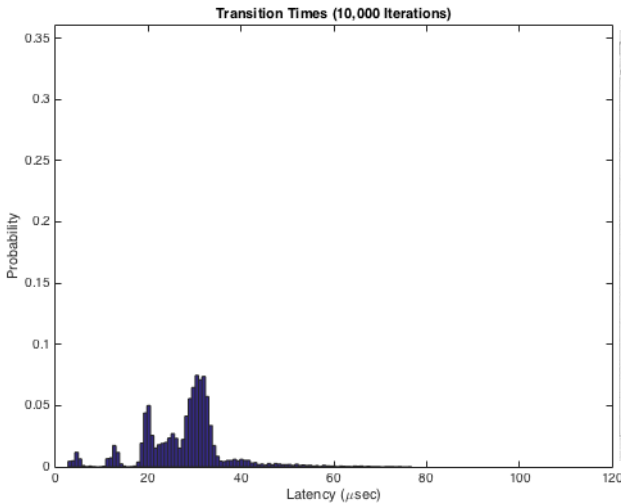
# Adaptive rescheduling design & implementation



# Adaptation mechanism overheads are acceptable

- Task notification via POSIX RT signals
  - » Ranged from 11.23 µsec to 110.03 µsec, often around 18 µsec
- Thread priority change (and possible core migration)
  - » Ranged from 2.67 µsec to 76.77 µsec, often around 30 µsec



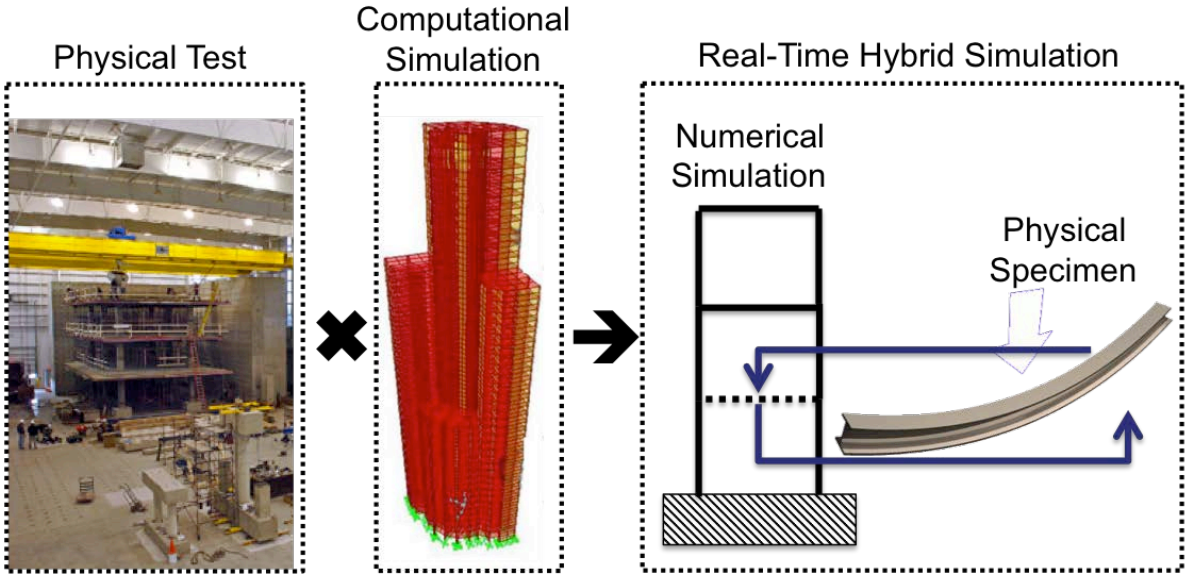

# Part III: emerging needs and opportunities

- Integrating mixed-criticality and elasticity
  - » Supporting nuanced graceful degradation during overload
  - » Increasing common-case utilization, even on mini-boxes

- Toward scalable cyber-physical platform frameworks
  - » Configurable, factory-driven, pattern-oriented software
  - » Necessary analysis, mechanisms, and coordination woven in
  - » Linux as an extensible (soft?) real-time operating system

# Why elasticity in mixed-criticality PRT systems

Can **"re-size" workloads** and/or **periods** adaptively, to **degrade lower-criticality components gracefully when higher-criticality utilizations surge**, e.g., if a *parallel* **high-criticality component** must switch to using a **more demanding computation** in a *given* **RTHS** with millisecond **real-time** guarantees, and a *parallel* 1000 degree-of-freedom **FEM** integrated **safely** with control, sensing, and actuation



# Key idea: above $Z_i$ *decrease* cores Used

| Task | $E_i$ | $Z_i$ | $m_i[0]$ | $m_i[1]$ | $m_i[2]$ |
|---|---|---|---|---|---|
| $T_1$ | 5 | 0 | 8 | 6 | 4 |
| $T_2$ | 4 | 0 | 6 | 4 | 2 |
| $T_3$ | 3 | 1 | 4 | **6** | 2 |
| $T_4$ | 2 | 1 | 4 | **5** | 2 |
| $T_5$ | 1 | 2 | 2 | 2 | **6** |

$$U_i[j] = \frac{C_i[j]}{D_i[j]}$$

$$m_i[j] = \left\lceil \frac{C_i[j] - L_i[j]}{D_i[j] - L_i[j]} \right\rceil$$

*degraded*

**overload**

nominal

- Nominal, overload, degraded ranges of utilization
  - » ***Same*** nominal utilization below designated criticality level
  - » Overload utilization at designated criticality level (if it's > 0)
  - » Degrading (non-increasing) utilizations at even higher levels

# Compress lower-criticality PRT tasks elastically

$$\textbf{minimize} \sum_{\forall i | Z_i < j} \frac{1}{E_i} (U_i[\alpha_i] - U_i)^2$$

(**criticality <u>insensitive</u> compression of lower criticality tasks**, $\alpha_i$ is $Z_i$ if task $\tau_i$ has overrun its virtual deadline, or 0 if it has not)

$$\textbf{such that } \forall_i \, (U_i[Z^{MAX}] \leq U_i \leq U_i[\alpha_i]) \wedge m \geq \sum_{k=1}^{n} m_k[j]$$

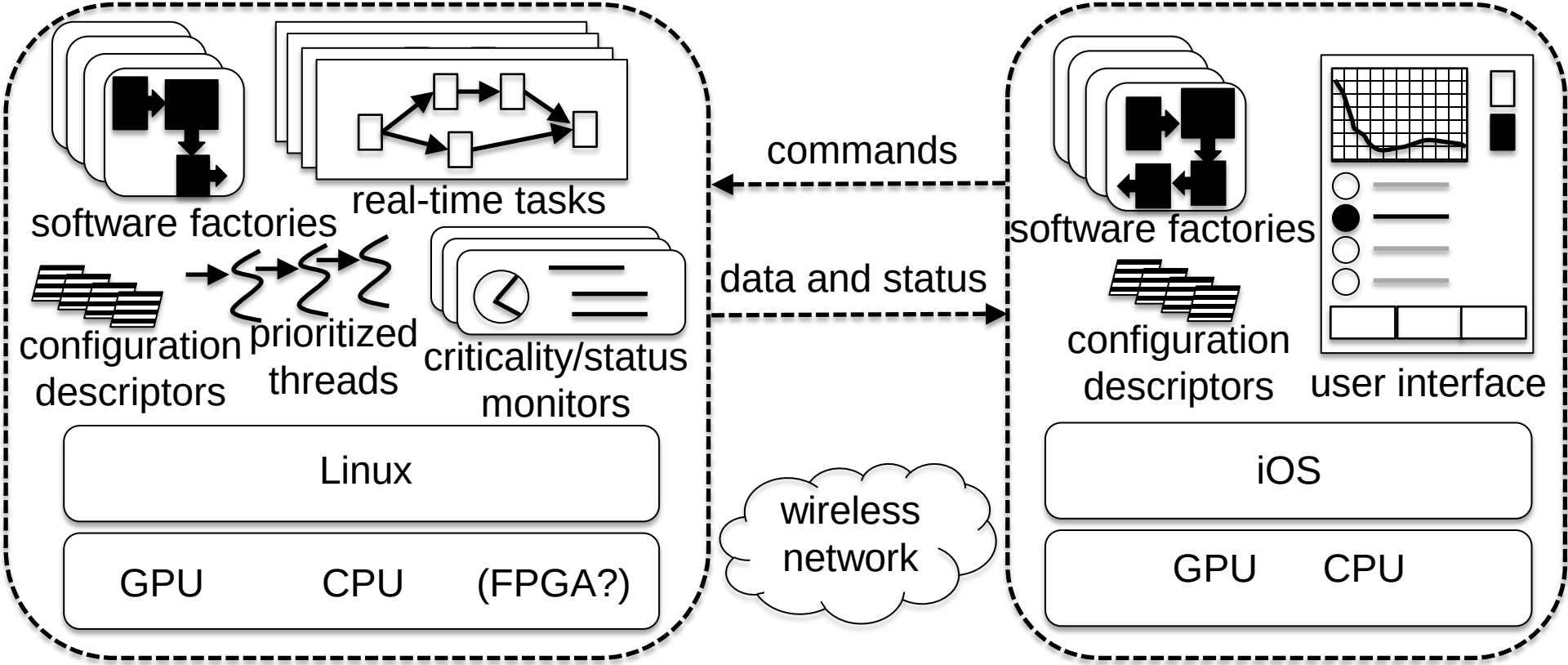
- In each mode, can vary e.g., $D_i$, $L_i$, and $C_i$
  - » Only compress tasks whose criticality $Z_i$ is **below** current level **j**
  - » Still checks whether **all** tasks are schedulable on $m$ cores

# Elastic compression strategies

- Both strategies preserve criticality invariant
  - » Higher criticality tasks meet deadlines, even in overload
  - » If necessary, can drop most demanding lowest criticality tasks
- Criticality *insensitive* strategy
  - » Compresses **all** tasks whose $Z_i$ is below current criticality level j
  - » Exploits elasticity of entire compressible set at once
- Criticality *sensitive* strategy
  - » Compress level-0 tasks, then level-1, etc. up to j-1
  - » Maintains strict criticality ordering, but still degrades gracefully

C. Gill, J. Orr, S. Harris, "Supporting Graceful Degradation through Elasticity in Mixed-Criticality Federated Scheduling," 6th WMC at RTSS, December 2018

# Why scalable cyber-physical platform frameworks



# Acknowledgements

PURDUE UNIVERSITY
=Gregory Bunting
Johnny Condori
Hugo Esquivel
Amin Maghareh
Prof. Shirley Dyke
Prof. Arun Prakash

BROWN UNIVERSITY
Christian Cianfarani
Christopher Wong

TEXAS The University of Texas at Austin
Phyllis Ang

STEPHEN F. AUSTIN STATE UNIVERSITY NACOGDOCHES, TEXAS
Sabina Adhikari

Washington University in St.Louis
James Orr
Steven Harris
-David Ferry
+Jing Li
*Abusayeed Saifullah
Prof. Kunal Agrawal
Prof. Chenyang Lu
Prof. Sanjoy Baruah
Prof. Roger Chamberlain
Prof. Chandler Ahrens

=now at Sandia National Labs
-now at St. Louis University
+now at New Jersey Institute of Technology
*now at Wayne State University

NSF
Supported in part by NSF grants CNS-1136073, CNS-1136075, CCF-1337218

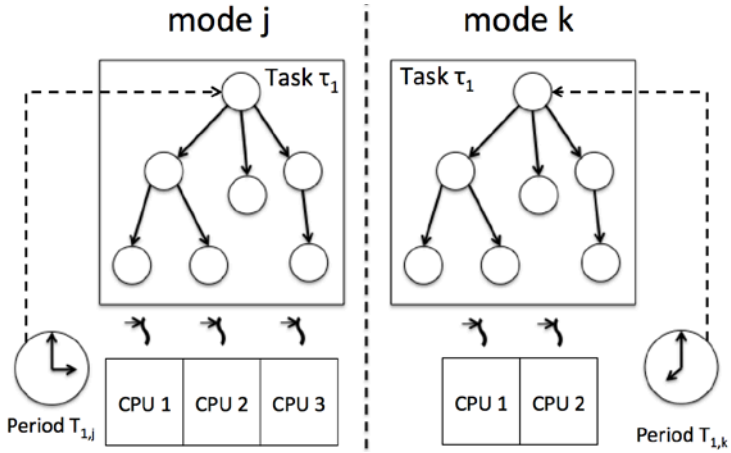
# Thank you for your interest in this work!

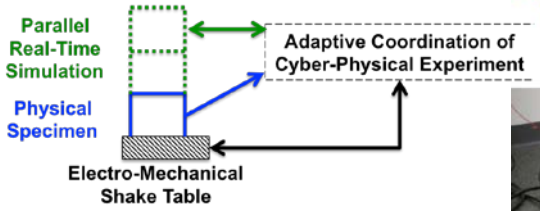
# Any questions?

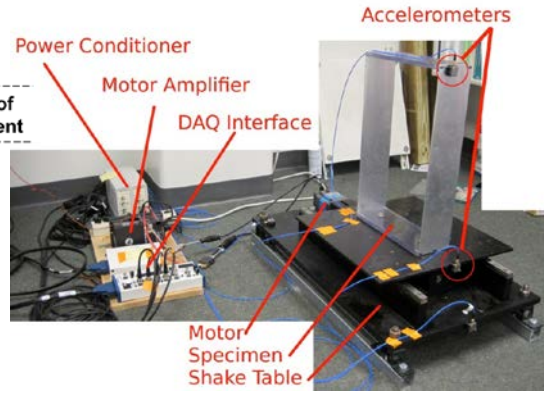
# Backup Slides

# RTHS platform design and implementation

- Per-task *objective functions* for application-specific adaptive mode change behavior
  - » *Event-driven or timing-based*
- System-wide communication for coordination and scheduling
- *Managers* maintain period, deadline, CPU allocation; and track the objective function
- C++ in OpenMP on Linux with RT patch, using shared memory and POSIX RT signals

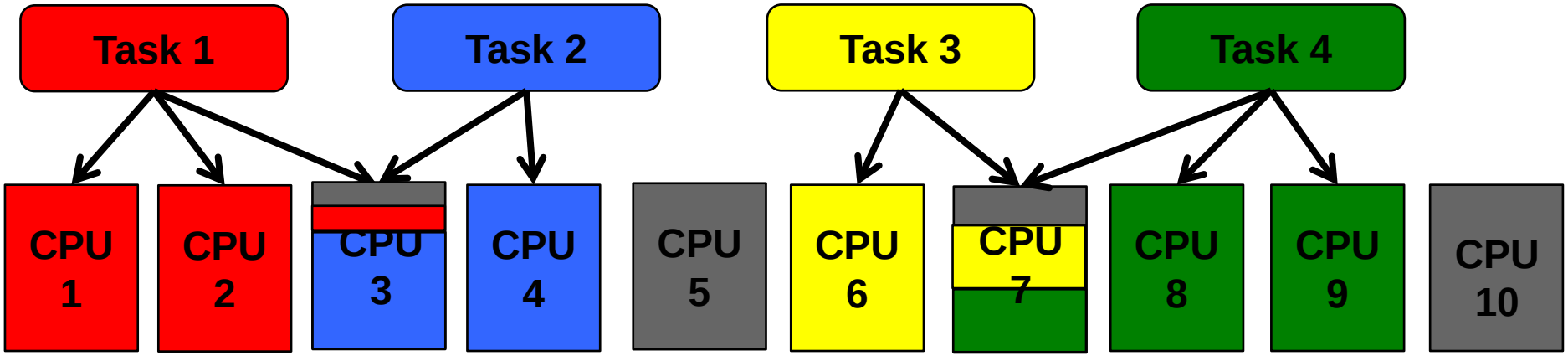


# Semi-federated scheduling

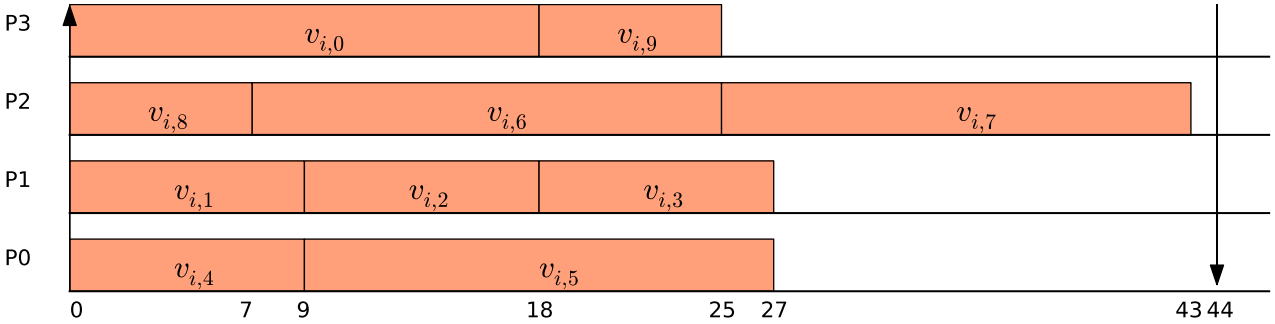
- In general a parallel task requires $\frac{C_i - L_i}{D_i - L_i} = \mathbf{A_i} + \boldsymbol{\varepsilon_i}$ ($A_i$ is integer, $0 \le \varepsilon_i \le 1$) CPUs to guarantee completion
- Semi-federated scheduling first allocates $\left\lfloor \frac{C_i - L_i}{D_i - L_i} \right\rfloor = \boldsymbol{A_i}$ CPUs
  - » Remaining $\varepsilon_i$ scheduled as sequential tasks on remaining CPUs (e.g. via partitioned EDF)




X. Jiang, N. Guan, X. Long, and W. Yi,
"Semi-Federated Scheduling of Parallel Real-Time Tasks on Multiprocessors," RTSS 2017

# Comparison: Dinh's vs other approaches, cont.

- Reservation-based federated scheduling[3] with $m_i = \lceil (C_i - L_i)/(D_i - L_i) \rceil = 11$
  - » Reservation servers require **total budget ≥ $C_i + L_i(m_i - 1)$** = 122 + 36*10 = 482
  - » Does not consider tasks' DAGs
- List scheduling[4] needs 4 processors
  - » Considers tasks' DAGs, but nodes are scheduled non-preemptively

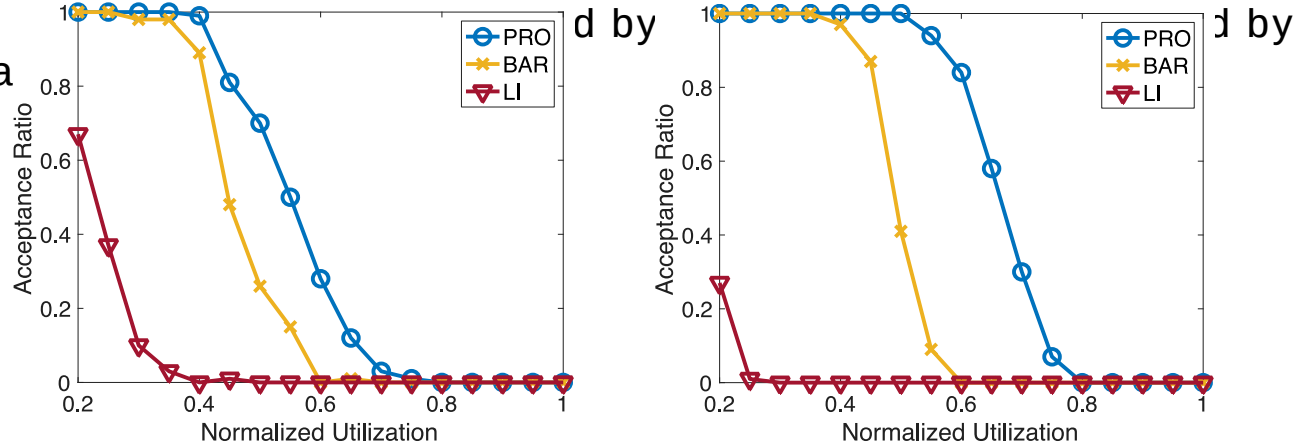



[3] Ueter et al., Reservation-Based Federated Scheduling for Parallel Real-Time Tasks, RTSS 2018
[4] Baruah et al., Federated Scheduling of Sporadic DAG Task Systems, IPDPS 2015

# Dinh's results for heavy tasks only

- Synthetic task sets contain only heavy tasks
  - » RandFixedSum is used to generate individual utilization of heavy tasks
  - » Deadlines are set equal to periods
  - » For each value of U, 100 task sets were generated

- Compare the d by LI) and Barua d by

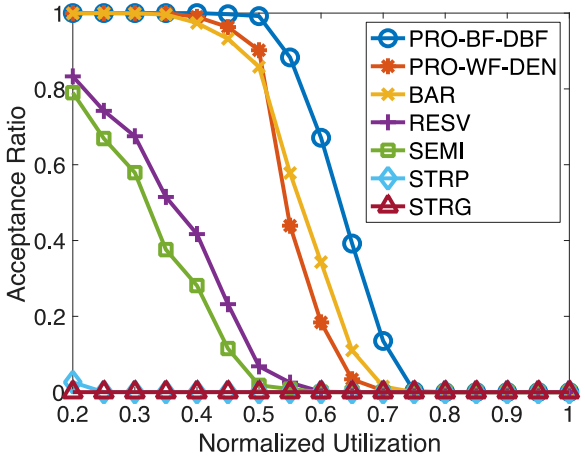


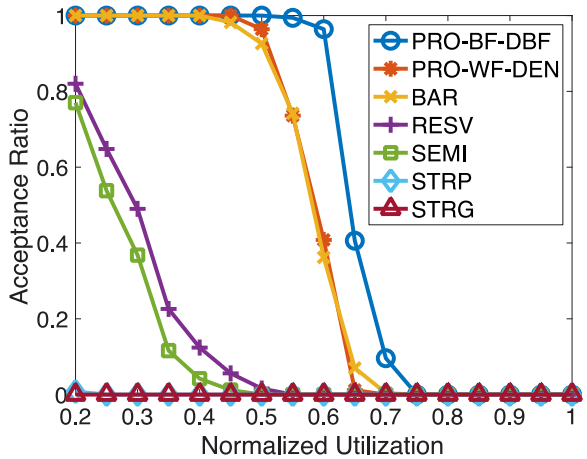

**m = 32 cores, n = 10 tasks  m = 64 cores, n = 15 tasks**

# Dinh's results for heavy and light tasks

- Synthetic task sets contain both heavy and light tasks
  - » Control utilization ratio of heavy tasks in each task set, 500 task sets generated per data point
  - » Use RandFixedSum to generate individual utilizations for heavy and light tasks
  - » Consider Worst-Fit, First-Fit, Best-Fit and 2 tests for partitioning sequential tasks to processors
  - » BAR and RESV use Best Fit with DBF test, SEMI uses Worst Fit with DEN test
- Compare also with Jiang et al. (denoted as SEMI), Ueter et al. (denoted as RESV), and a stretching algorithm by Qamhieh et al.[5] (denoted by STRP and STRG)

**m = 32, $U^h$ = 0.5, $n^h$ = [2, 7], $n^l$ = 40**



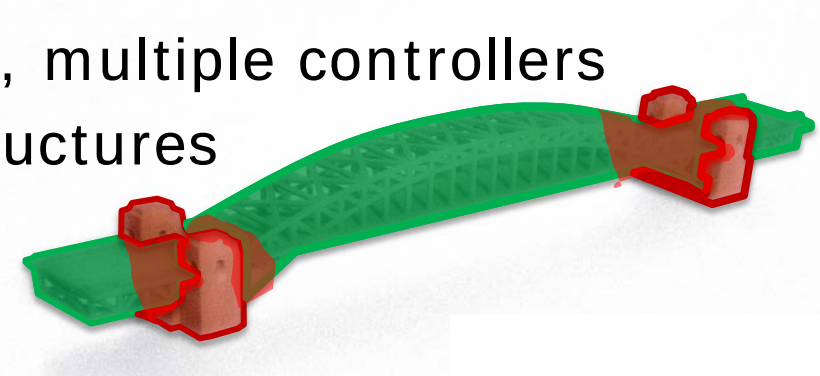

**m = 64, $U^h$ = 0.5, $n^h$ = [4, 14], $n^l$ = 80**

[5] Qamhieh et al., A Stretching Algorithm for Parallel Real-Time DAG Tasks on Multiprocessor Systems, RTNS 2014

# Motivation for adaptive RTHS

- Non-linear physical behavior
  - » Variable computational loads (particle vs. Kalman filters)
  - » Diversity of components' behaviors during an experiment

- More aggressive experiments
  - » Multiple physical components, multiple controllers
  - » Larger, more detailed sub-structures
  - » Variability of computations' rates and resource demands



# Elastic scheduling as constrained optimization

- Chantem et al. defined this as an optimization problem
  - » Minimize a weighted sum of squares of the differences between the chosen utilization for each task and its maximum utilization
  - » Subject to utilizations being between minimum and maximum values and the sum not exceeding the available utilization

$$\textbf{minimize} \sum_{i=1}^{n} \frac{1}{E_i} \left( U_i^{(max)} - U_i \right)^2$$

$$\textbf{such that } \forall_i \left( U_i^{(min)} \leq U_i \leq U_i^{(max)} \right) \wedge U_d \geq \sum_{i=1}^{n} U_i$$

T. Chantem, X. S. Hu, and M. D. Lemmon,
"Generalized Elastic Scheduling for Real-Time Tasks," RTSS 2006

# Elastic compression of parallel real-time tasks

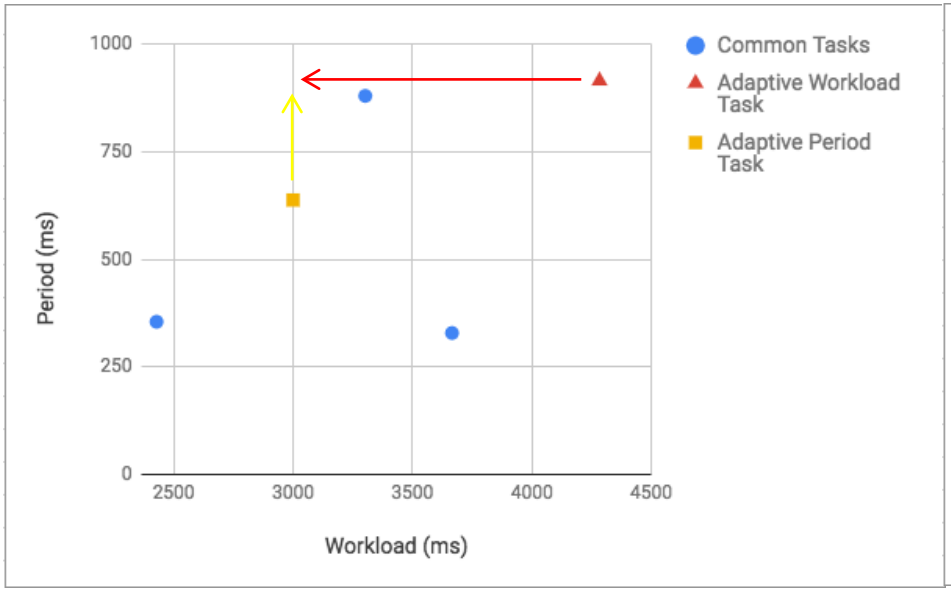
$$\textbf{minimize} \sum_{i=1}^{n} \frac{1}{E_i} \left( U_i^{(max)} - U_i \right)^2$$

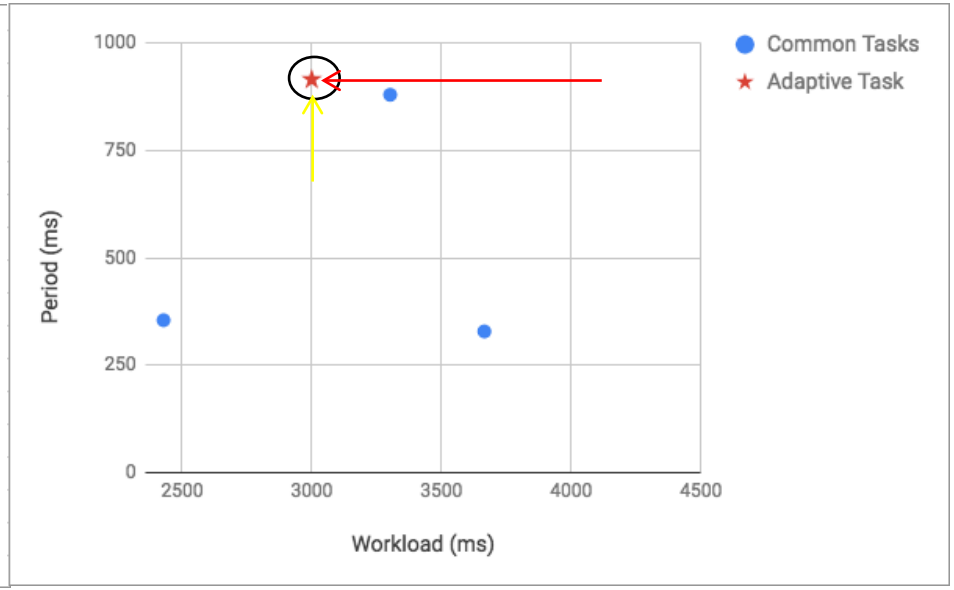
$$\textbf{such that } \forall_i \left( U_i^{(min)} \leq U_i \leq U_i^{(max)} \right) \wedge m \geq \sum_{i=1}^{n} \left\lceil \frac{C_i - L_i}{T_i - L_i} \right\rceil$$

- Updates optimization from Chantem et al. (RTSS 2006)
  - » Uses utilization definition for parallel real-time tasks
  - » Allows either period or work to be compressed elastically
  - » Checks schedulability under Federated Scheduling on $m$ cores

# Equivalence of rate versus workload compression

- Experiments compared varying a task's $D_i$ vs. its $C_i$
- Comparable tasks compressed to the same utilization
  - » Temporally vs. computationally elastic tasks reached same point



# Total cores available matters for EMCFS

- Upper bound (overprovisioning)
  - » Sum of max cores each task needs:
  - » All tasks ***could*** run in overload

$$\sum_{i=1}^{n} m_i\,[Z_i]$$

- Sufficient cores available (design target)
  - » Tasks utilizations can be met at each level: $\max_j \left( \sum_{i=1}^{n} m_i\,[j] \right)$
- Lower bound (underprovisioning)
  - » Below this tasks are dropped at ***every*** level: $\min_j \left( \sum_{i=1}^{n} m_i\,[j] \right)$