



# Mixed-Criticality Scheduling of Processing Pipelines

Presenter: Dionisio de Niz

Bjorn Andersson, Hyoseung Kim, Mark Klein, Linh Thi Xuan Phan, and Raj Rajkumar



Copyright 2018 Carnegie Mellon University, Hyoseung Kim and Linh Thi Xuan Phan. All Rights Reserved.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

**NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.**

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at [permission@sei.cmu.edu](mailto:permission@sei.cmu.edu).

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM18-0288



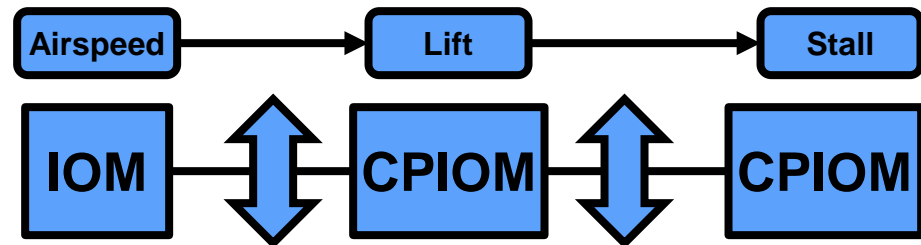
# Motivation

## Mixed-Criticality

- Certification standards (DO-178B/C)
  - Different assurance rigor depending on severity of failure (criticality)
  - If proven that low-criticality task cannot interfere with higher-criticality ones

## Distributed Scheduling

- End-to-end threads running across multiple processors



## Distributed Mixed-Criticality Scheduling



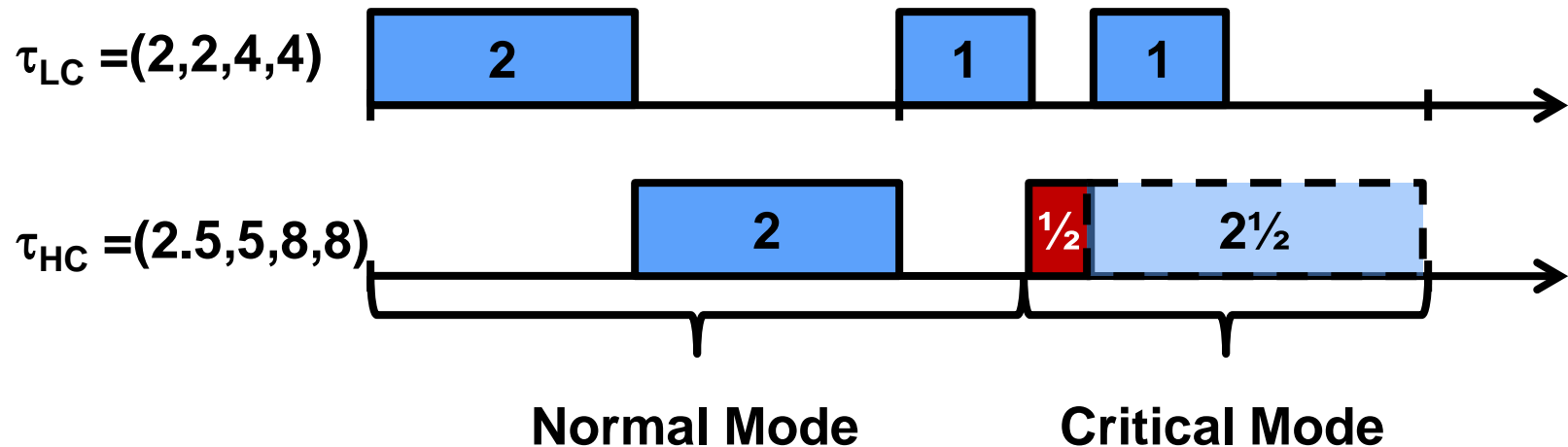
# Zero-Slack Scheduling (1)

Tasks :  $\tau_i = (C_i, C_i^o, T_i, D_i, \zeta_i)$

- $C_i$ : nominal WCET,  $C_i^o$ : overloaded WCET,  $T_i$ : period,  $D_i$ : deadline,  $\zeta_i$ :criticality

Start with DM

- Calculate last instant to stop lower-critical task  $Z_i$ : zero-slack instant
- Stop lower-criticality tasks



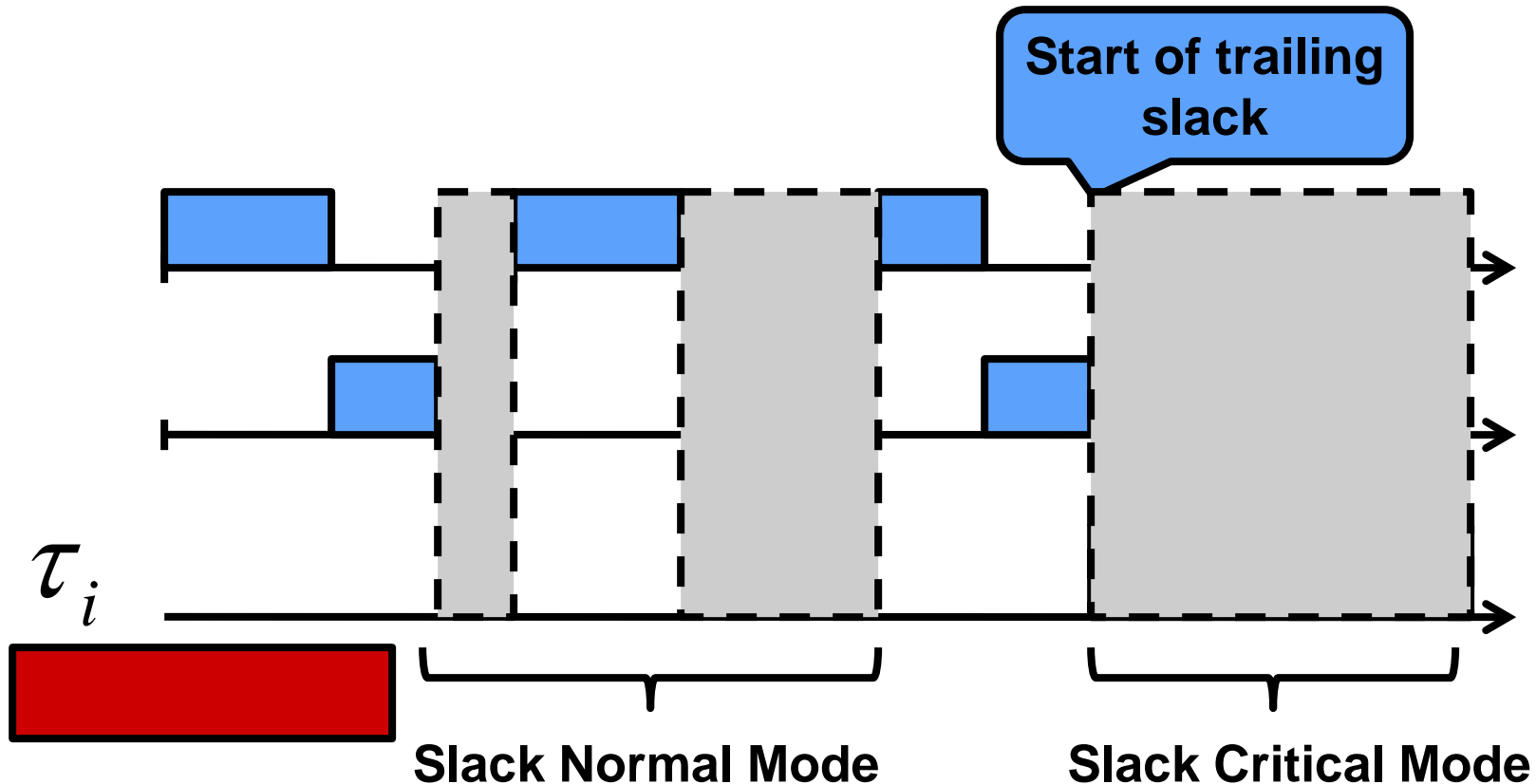
# Zero-Slack Scheduling (2)

## Guarantee

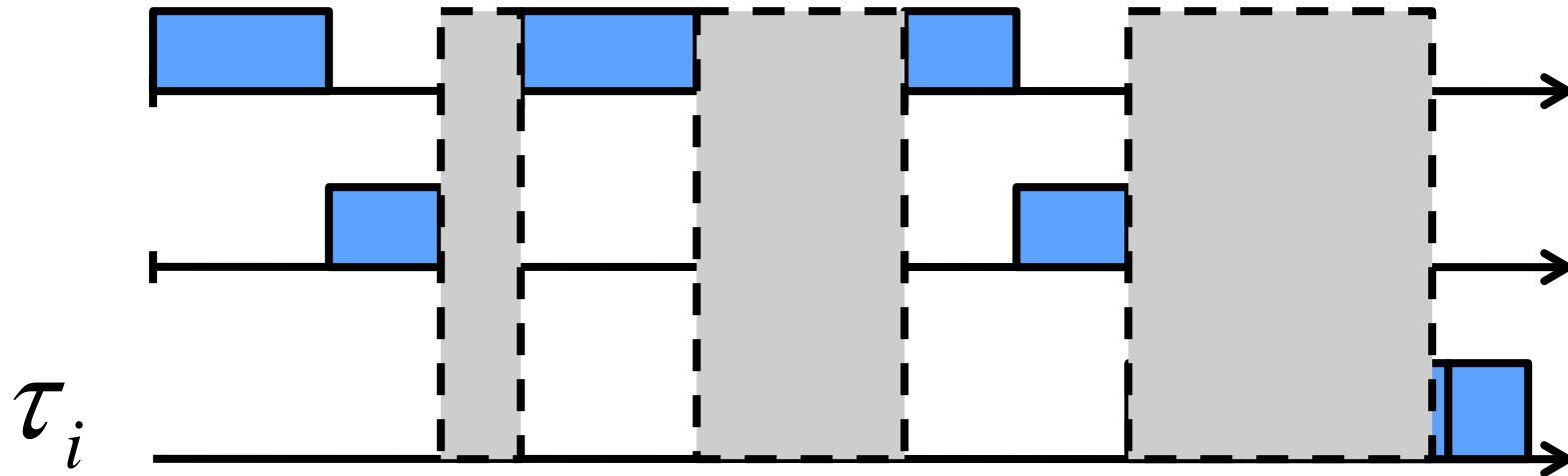
- $\tau_i$  guaranteed to execute for  $C_i^0$  before its deadline  $D_i$ 
  - If no higher criticality task  $\tau_j$  executes for more than  $C_j$



# Calculating The Zero-Slack Instant



# Calculating The Zero-Slack Instant



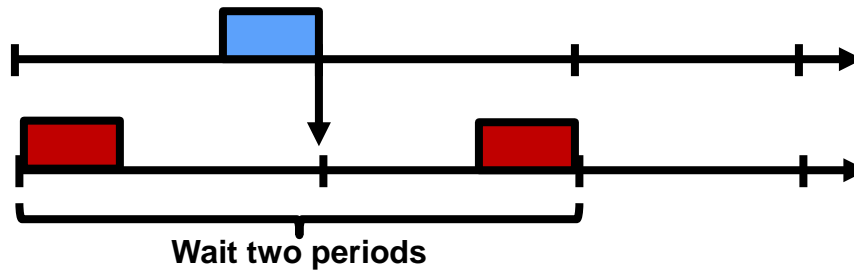
New slack can open after each iteration

Needs to repeat until no new slack opens

# Distributed Scheduling

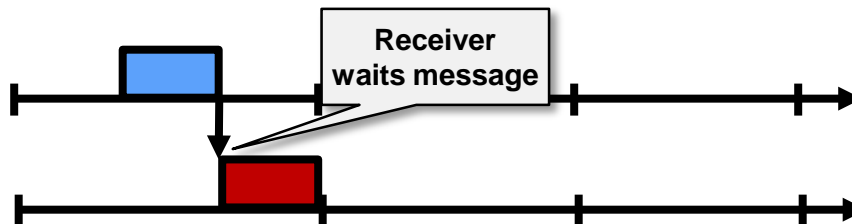
## Multi-period end-to-end deadlines

- Start of receiver task not synchronized
- Message reception guaranteed after two periods



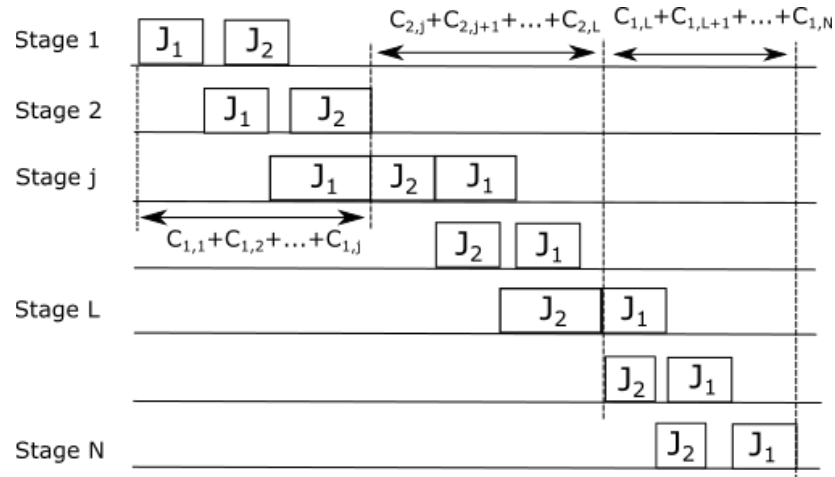
## Pipelines to reduce wait

- Synchronize start of receiver task





# Fixed-Priority Pipelines



$$Delay(J_i) \leq \sum_{j=i}^n C_{eqj} + \sum_{s=1}^{N-1} \max_{j=i}^n (C_{j,s})$$

$$C_{eqj} = \begin{cases} C_{j,max1} + C_{j,max2} & \text{if } A_i < A_j \\ C_{j,max1} & \text{otherwise} \end{cases}$$



# Real-Time Pipelines Response Time

## Synthetic Taskset

- $\tau_i$  execution:

$$- C_e^*(i) = \sum_{(j|\rho_j \geq \rho_i)} C_{j,max1} + \sum_{s=1}^{N-1} \max_{(j|\rho_j \geq \rho_i)} (C_{j,2})$$

- Preemption on others

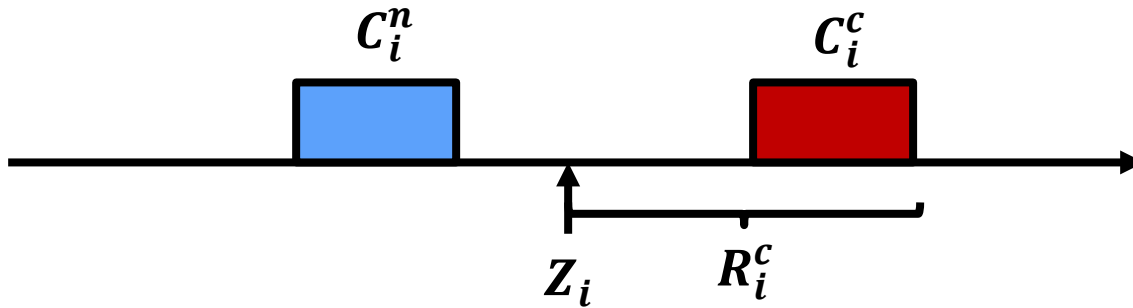
$$- C_j^* = C_{j,max1} + C_{j,max2}$$

- Response time

$$- R_i^{(k)} = C_e^*(i) + \sum_{(j|\rho_j > \rho_i)} \left\lceil \frac{R_i^{k-1}}{T_j} \right\rceil C_j^*$$



# ZSRM Response Time Equations



$$C_i^c = C_i^o ; C_i^n = 0$$

$$R_i^c = C_i^c + \sum_{j \in H_i^{hc}} \left\lceil \frac{R_i^c}{T_j} \right\rceil C_j + \sum_{j \in L_i^{hc}} \left\lceil \frac{R_i^c}{T_j} \right\rceil (C_j - C_j^n) + \sum_{j \in H_i^{sc}} \left\lceil \frac{R_i^c}{T_j} \right\rceil C_j^o$$

$$Z_i = D_i - R_i^c$$

$$I_i = \sum_{j \in H_i^{hc}} \left\lceil \frac{Z_i}{T_j} \right\rceil C_j + \sum_{j \in H_i^{lc} \cup H_i^{sc}} \left\lceil \frac{Z_i}{T_j} \right\rceil C_j^o$$

$$S_i^n = \max(0, Z_i - I_i - C_i^n)$$

$$C_i^c = \max(0, C_i^c - S_i^n) ; C_i^n = \min(C_i^o, C_i^n + S_i^n)$$

## Interfering Tasksets

Set	Prio	Crit
$H_i^{hc}$	H	H
$H_i^{lc}$	H	L
$H_i^{sc}$	H	S
$L_i^{lc}$	L	L
$L_i^{hc}$	L	H



# ZSRM Pipeline Response Time (1)

Stages in normal and critical mode

- $\sigma_i^z$  : stage where  $Z_i$  occurs
- $\Pi_i^c = \{\pi_j | \sigma_i^z \leq j \leq N\}$  ;  $\Pi_i^n = \{\pi_j | 1 \leq j < \sigma_i^z\}$

Critical mode synthetic task

- $C_e^c(i) = C_e^K(i)$
- $C_e^K(i) - \sum_{\tau_i \in \Gamma_i^c \cup \tau_i} C_{j,max1}^{C_i} + \sum_{\pi_{i,s} \in \Pi_i^c \setminus \pi_N} \max_{\tau_j \in \Gamma_i^c \cup \tau_i} C_{j,s}^{e_i}$
- To calculate which stages overload  $C_{j,s}^{e_i}$  is calculated solving optimization

– Maximize  $\sum_{s \in \Pi_i^c \setminus \pi_N} \max_{j \in \Gamma_i^c \cup \tau_i} x_{j,s}$

• Subject to:

•  $\forall \langle j, s \rangle$  s.t.  $(j \in \Gamma_i^c \cup \tau_i) \wedge (s \in \Pi_i^c), x_{j,s} \leq C_{j,s}^o$

•  $\forall j$  s.t.  $(j \in \Gamma_i^c \cup \tau_i) \wedge (\zeta_j > \zeta_i), \sum_{s \in \Pi_i^c} x_{j,s} \leq \sum_{s \in \Pi_i^c} C_{j,s}$

•  $C_j^{*c_i} = C_{j,max1}^{C_i} + C_{j,max2}^{C_i}$

•  $R_i^c = C_e^c(i) + \sum_{j \in \Gamma_i^c} \left\lceil \frac{R_i^c}{T_j} \right\rceil C_j^{*c_i}$



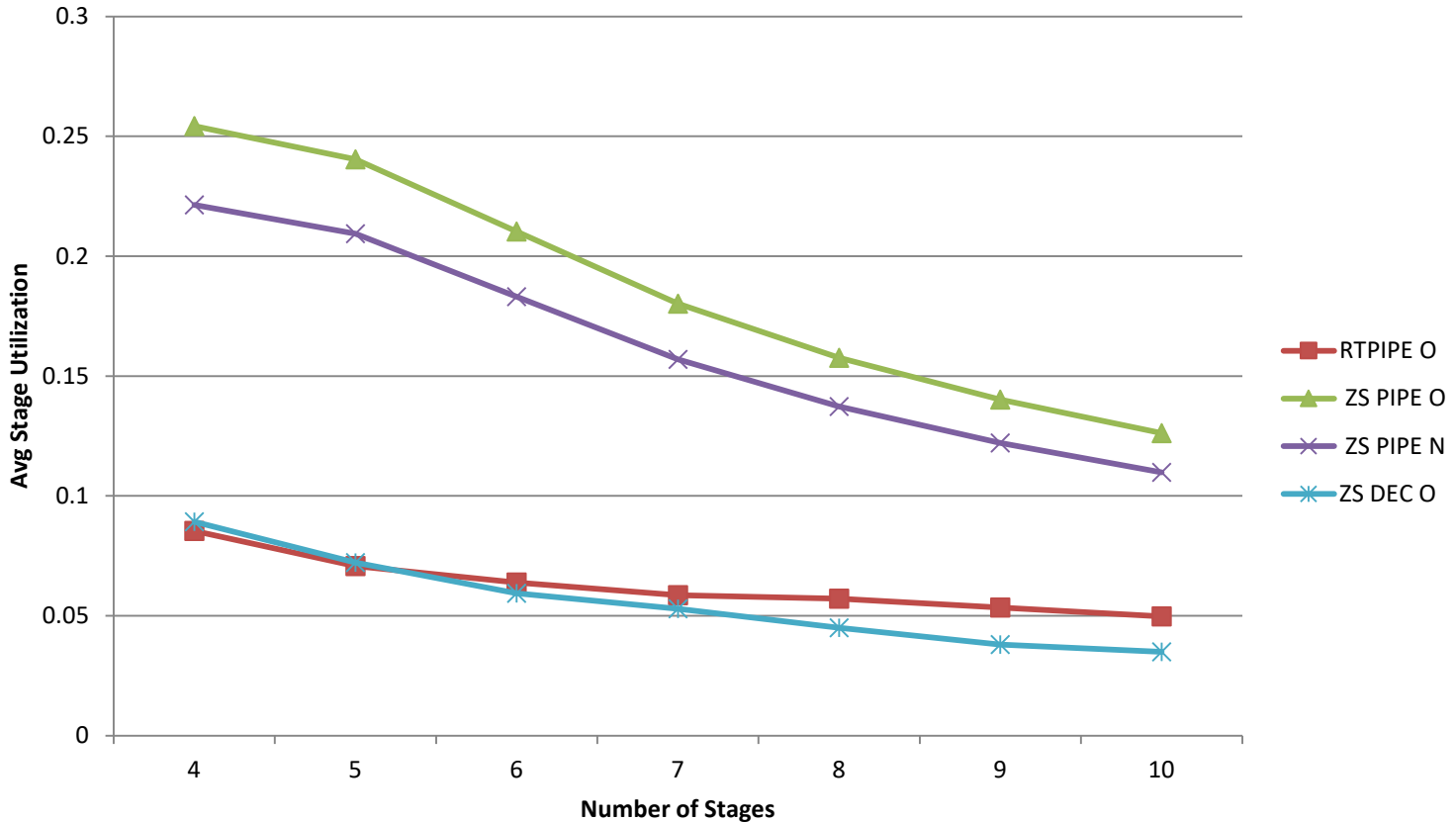
# ZSRM Pipeline Response Time (2)

Normal mode

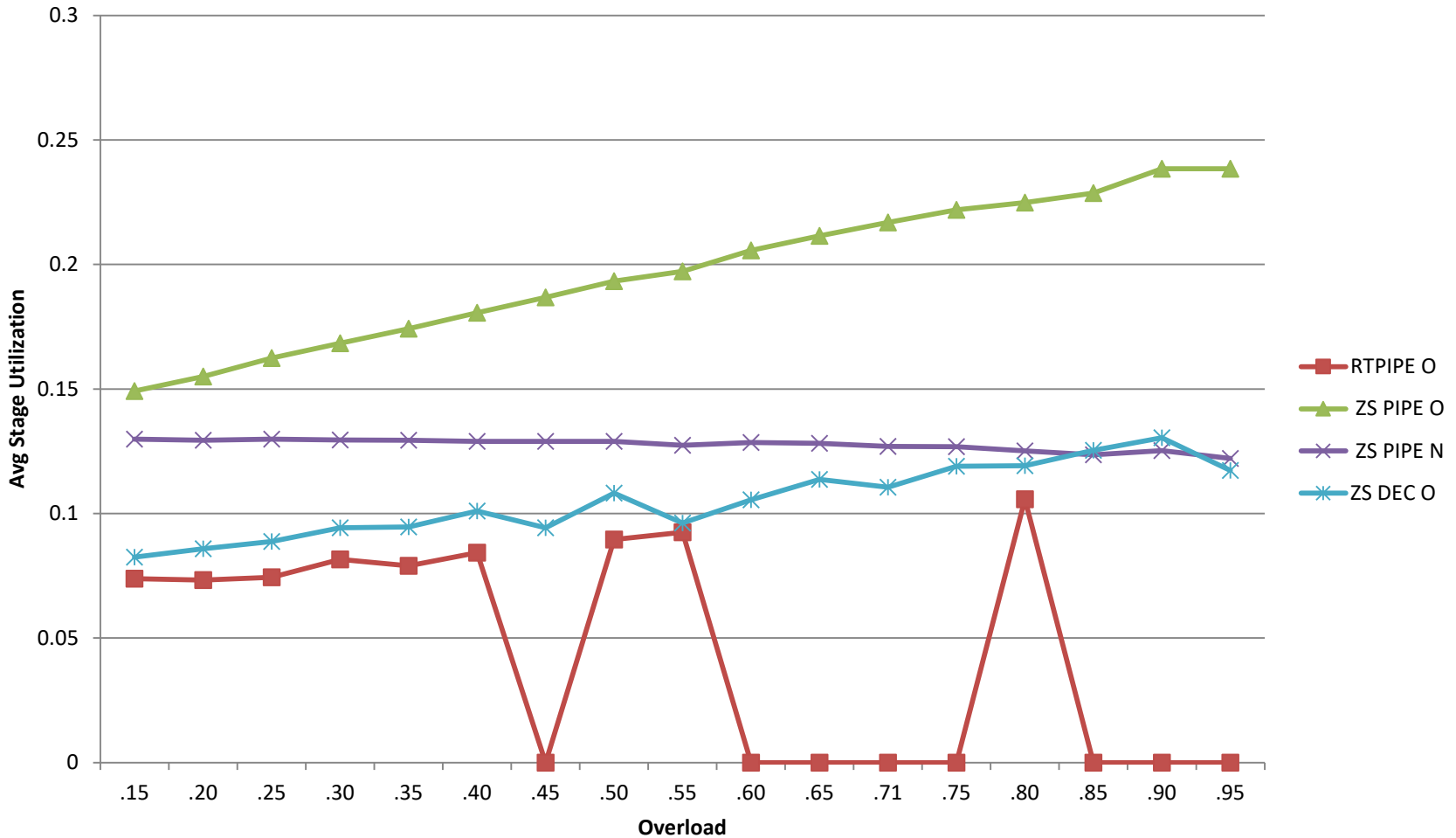
- $Z_i = D_i - R_i^c$
- $I_i = \sum_{j \in \Gamma_i^n} \left\lfloor \frac{Z_i}{T_j} \right\rfloor C_j^{*n_i}$
- $S_i^n = \max(0, Z_i - I_i - C_e^n(i))$
- $C_e^c(i) = \max(0, C_e^c(i) - S_i^n)$
- $C_e^n(i) = \min(C_i^o, C_e^n(i) + S_i^n)$
- $\sigma_i^z = \min_{1 \leq j \leq N} \{j \mid \sum_{s=1}^j C_{i,s}^o > C_e^n(i)\}$



# Average Utilization With Increasing Stages



# Increasing Overload



# Implementation

## Linux Kernel Module

- Periodic Activation of Root Stage
  - `wait_next_root_period()`
  - Wait for periodic timer to activate
- Middle Stage Activation
  - `wait_next_stage_arrival()`
  - Wait for message and enforce minimum inter-arrival
  - Execute stage computation
  - Send message to next stage
- Leaf Stage Activation
  - `wait_next_leaf_arrival()`
  - Wait for minimum interarrival
  - Execute stage computation
- End-to-end overload monitoring
- Defer after overload enforcement



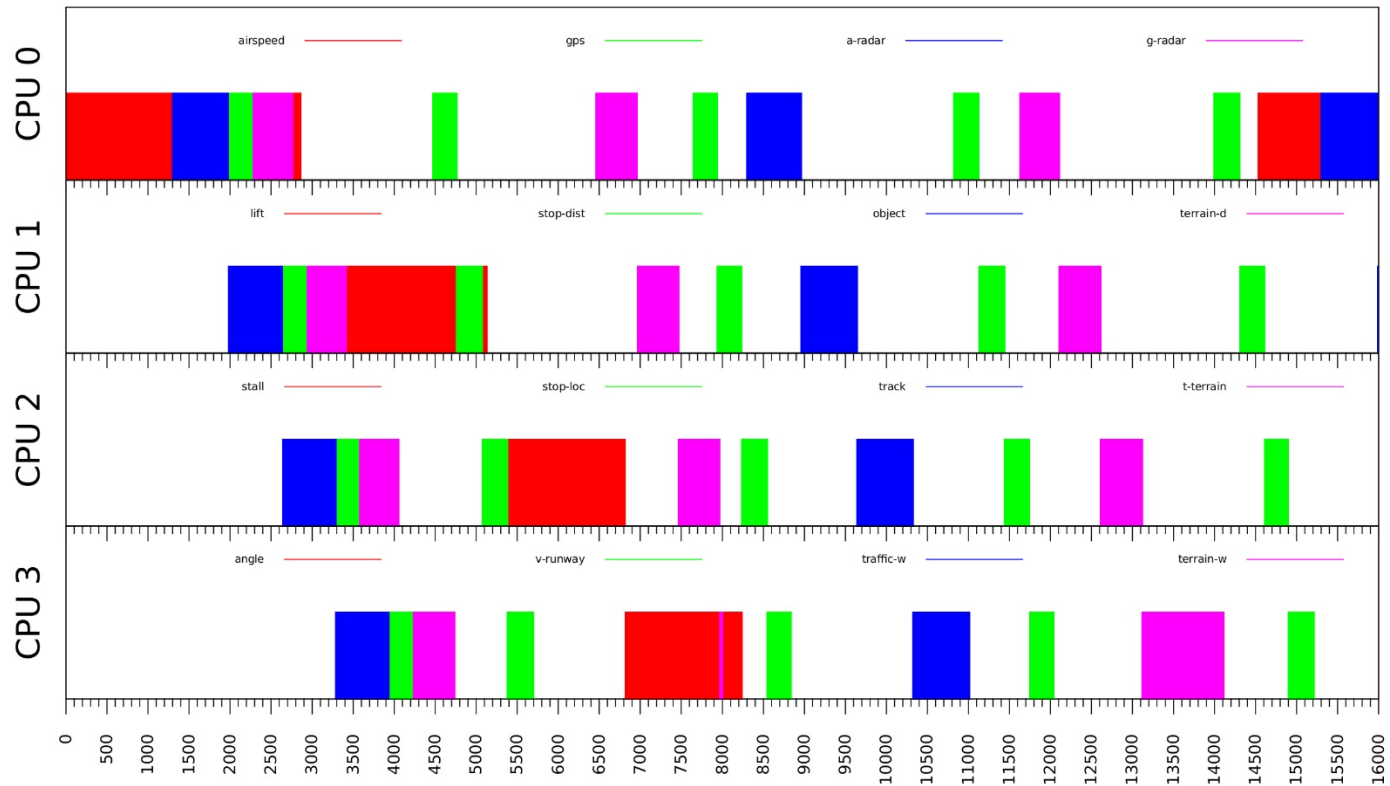


# Synthetic Avionics Taskset

Stage\Task	Stall Warning $\zeta = 4, \rho = 1$	Runway Overrun $\zeta = 2, \rho = 3$	Traffic Aware $\zeta = 1, \rho = 4$	Terrain Aware $\zeta = 3, \rho = 2$
IOM	Airspeed	GPS Pos.	Air Radar	Ground Radar
CPIOM1	Lift	Stop Distance	Object Ident.	Terrain Distance
CPIOM2	Stall threshold	Stop location	Track build.	Time to terrain
CPIOM3	Angle attack	Virtual runway	Traffic warn.	Terrain warn.
T	14534(ms)	3174(ms)	7002(ms)	5164(ms)
C	1090(ms)	238(ms)	525(ms)	387(ms)
C°	1362(ms)	297(ms)	656(ms)	484(ms)
Z	8000(ms)	3174(ms)	7002(ms)	5164(ms)



# Avionics Taskset Execution Timeline



# Conclusions

## Pipeline Scheduling Improves Utilization

- In contrast to unsynchronized activation
- Response-Time Pipeline Model by Jayachandran and Abdelzaher

## Created First Mixed-Criticality Scheduling

- ZSRM
  - Reformulated Response Time Z Calculation
  - End-to-end overload
  - End-to-end enforcement

## Improved Performance

- 4X average state utilization
- Increase overload tolerance

