Temporal Protection in Real-Time Systems

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

November 2016

© 2016 Carnegie Mellon University

[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.

Copyright 2016 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

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.

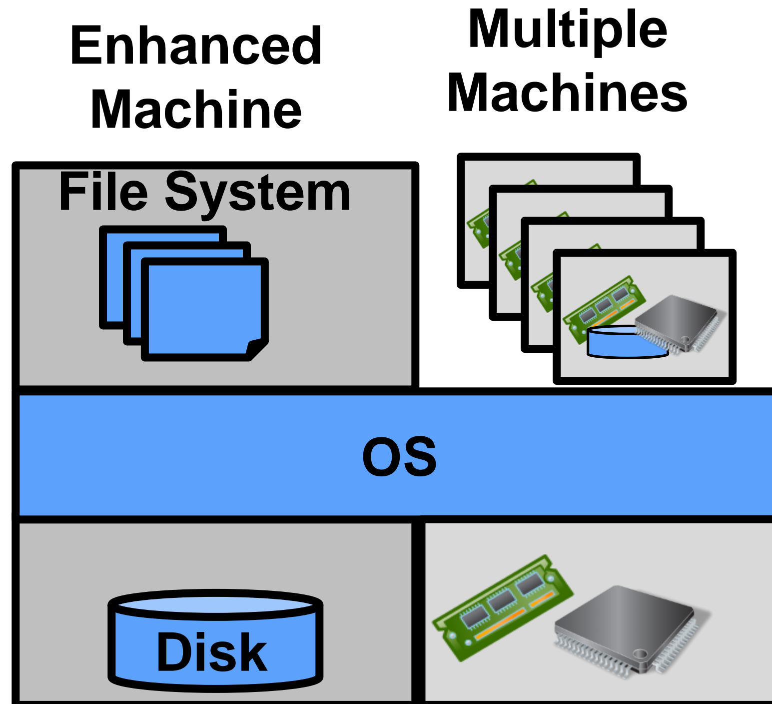
[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

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.

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

DM-0004174

OS Dual Objective

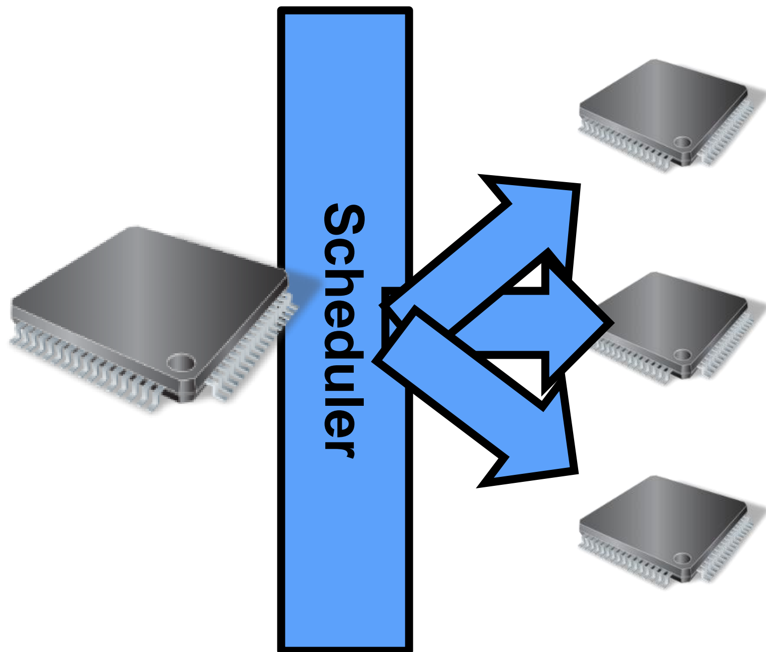


Icons credit: http://www.doublejdesign.co.uk

© 2016 Carnegie Mellon University

[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.

Time-Sharing CPU – Round robin

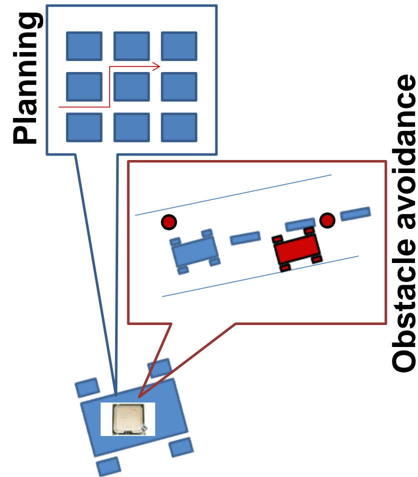


Same time requirement – Fair Scheduling

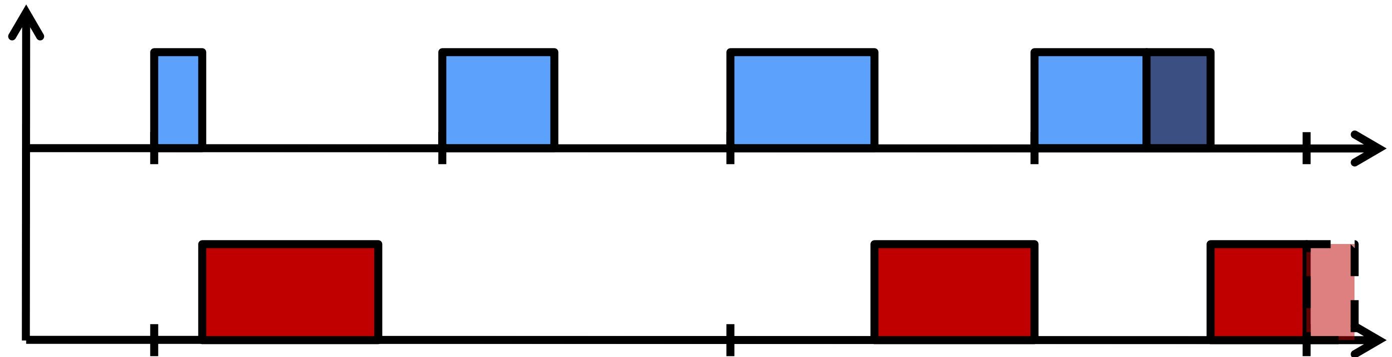
© 2016 Carnegie Mellon University

[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.

Consolidation of Mixed-Criticality Tasks

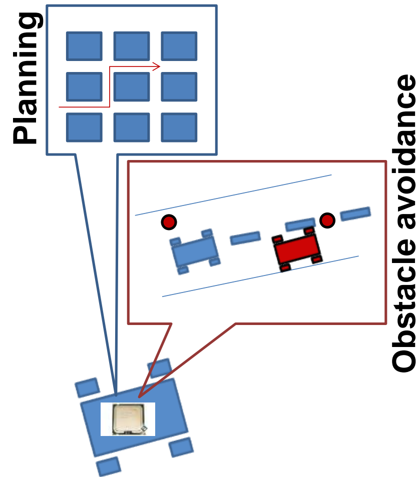


Shared Hardware
Can lead to cycle stealing

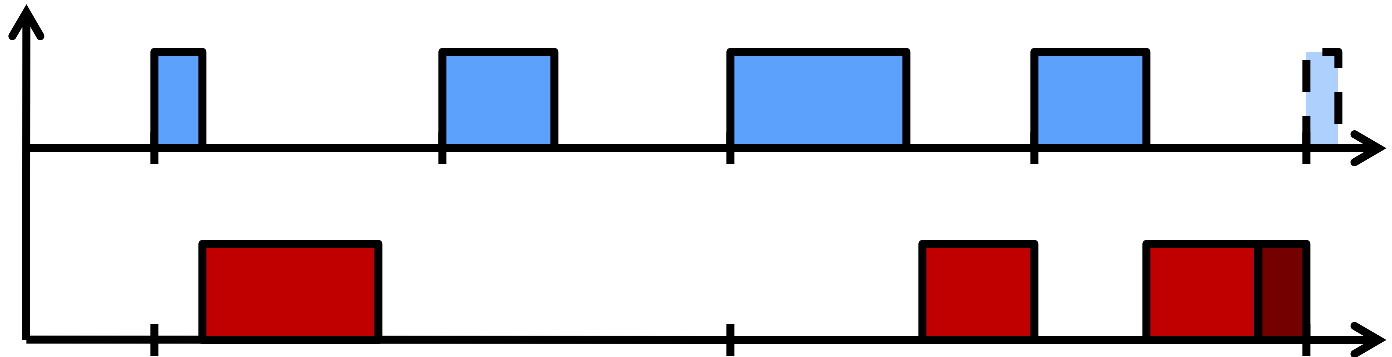


© 2016 Carnegie Mellon University
[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Consolidation of Mixed-Criticality Tasks

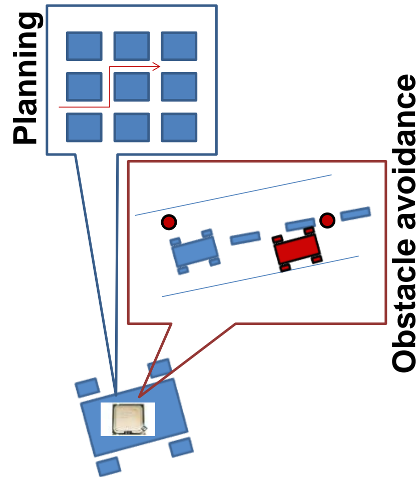


To avoid interference
add temporal protection

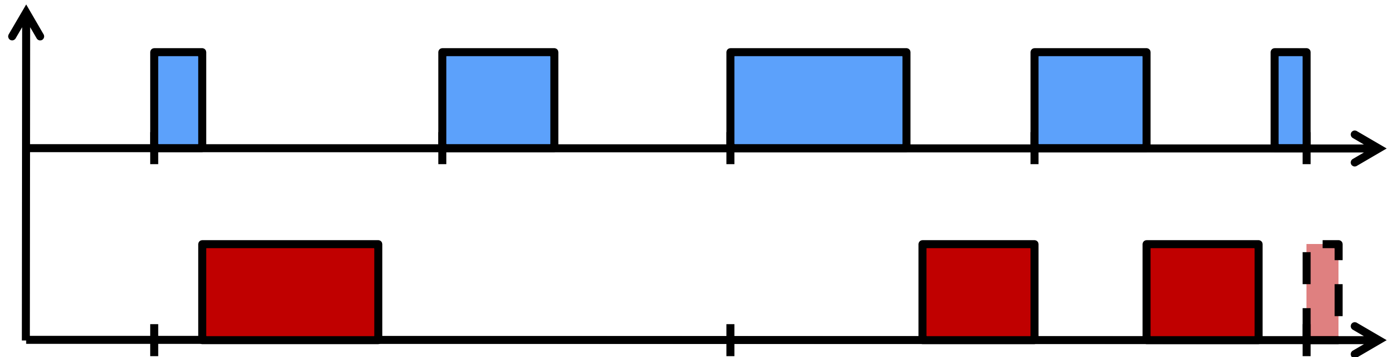


© 2016 Carnegie Mellon University
[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Consolidation of Mixed-Criticality Tasks



BUT
Symmetric protection
leads to *criticality inversion*



© 2016 Carnegie Mellon University

[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.

Criticality Inversion

A higher-criticality task waits for a lower-criticality task to release a resource

- Symmetric temporal protection
- Scheduling policy is aimed at maximizing utilization (RMS/DMS/EDF)

© 2016 Carnegie Mellon University

[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.

Rate-Monotonic Priority

Shorter Period → Higher Priority

- Ideal utilization

BUT: Poor Criticality Protection Due to Criticality Inversion

- If criticality order is opposite to rate-monotonic priority order

High Criticality

10/100

100

Low Criticality

1/10

© 2016 Carnegie Mellon University

[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.

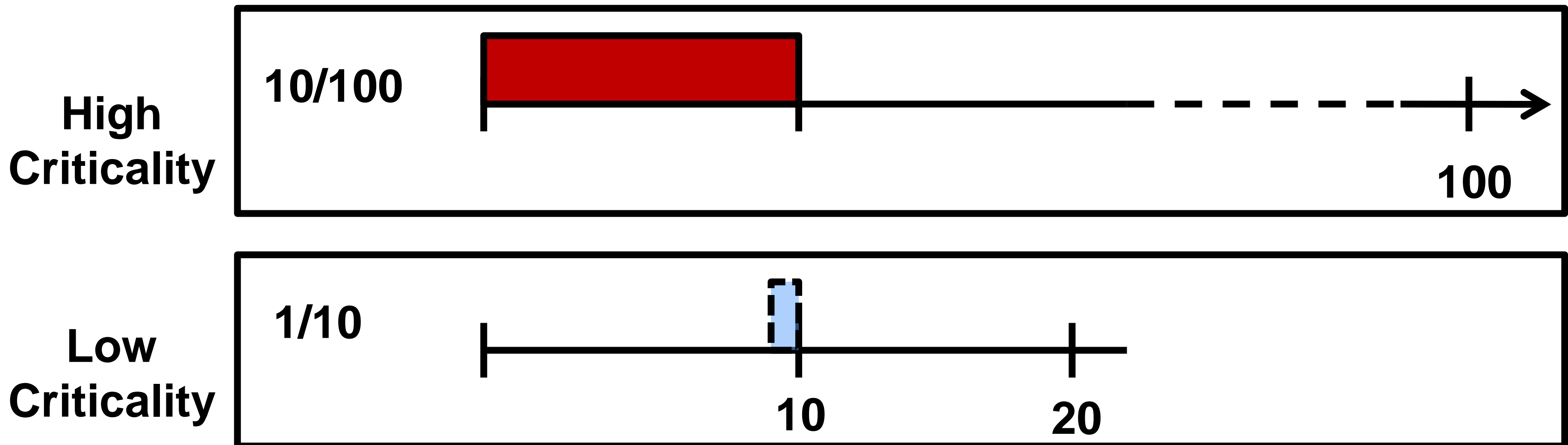
Criticality As Priority Assignment (CAPA)

Higher Criticality → Higher Priority

- Ideal criticality protection:
  - lower criticality cannot interfere with higher criticality

BUT: Poor Utilization Due to Priority Inversion

- If criticality order is opposite to rate-monotonic priority order



© 2016 Carnegie Mellon University
[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.

Task Model

$$\tau_i = \left(C_i, C_i^o, T_i, D_{i,} \zeta_i\right)$$

C_i **Normal Execution Budget of task *i***

C_i^o **Overload Execution Budget of task *i***

T_i **Period of task *i***

D_i **Deadline of task *i*** $D_i \leq T_i$

ζ_i **Criticality of task *i***

© 2016 Carnegie Mellon University

[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.

Zero-Slack Scheduling

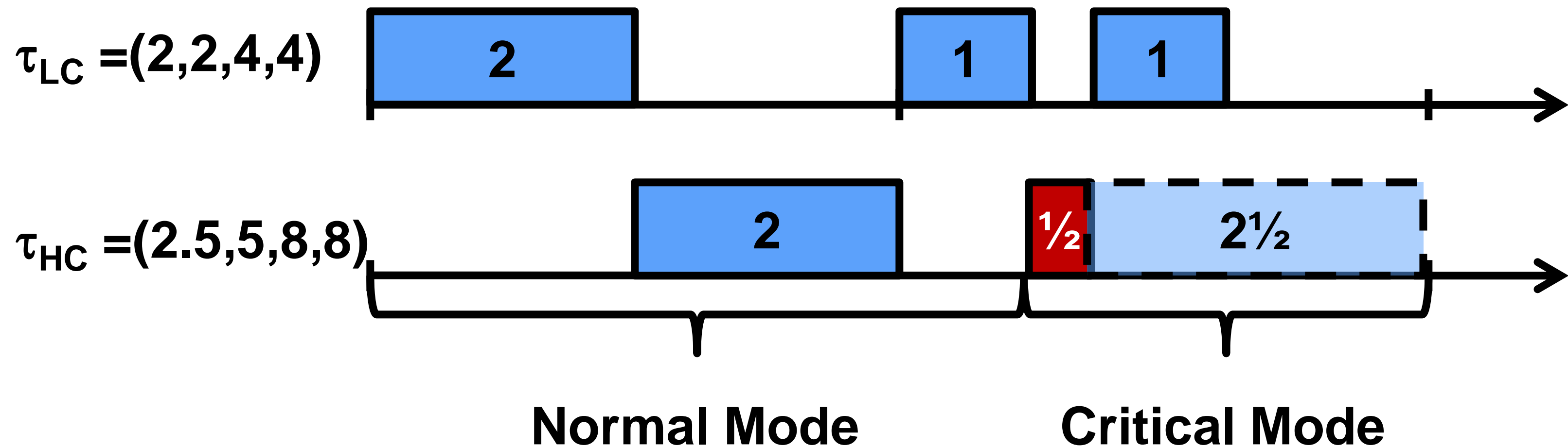
Start with RM

Calculate the last instant before τ_{HC} misses its deadline

- this is called the ***zero-slack*** instant

Switch to criticality-as-priority

- Splits the execution window into
  - Normal mode (RM)
  - Critical mode (CAPA)



© 2016 Carnegie Mellon University

[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.

Critical Instant of a Task τ_i

Higher Priority Higher Criticality

Suspended

Higher Priority Lower Criticality

τ_i

N mode

C mode

Higher Criticality Lower Priority

© 2016 Carnegie Mellon University

[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.

Interference in Zero-Slack Scheduling

Task Set Divided into

- H^{lc} : Higher priority, lower criticality
- H^{hc} : Higher priority, higher criticality
- L^{lc} : Lower priority, lower criticality
- L^{hc} : Lower priority, higher criticality

Interfering tasks in normal mode (Normal mode)

- $H^{lc} + H^{hc} + L^{hc}$

Interfering tasks in critical mode (C mode)

- $H^{hc} + L^{hc}$

© 2016 Carnegie Mellon University

[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.

Scheduling Guarantee

A task τ_i is guaranteed C_i^o before D_i

if no $\tau_j \mid \zeta_j < \zeta_i$

executes beyond its C_j

© 2016 Carnegie Mellon University

[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.

Calculating The Zero-Slack Instant

Start of trailing slack

τ_i

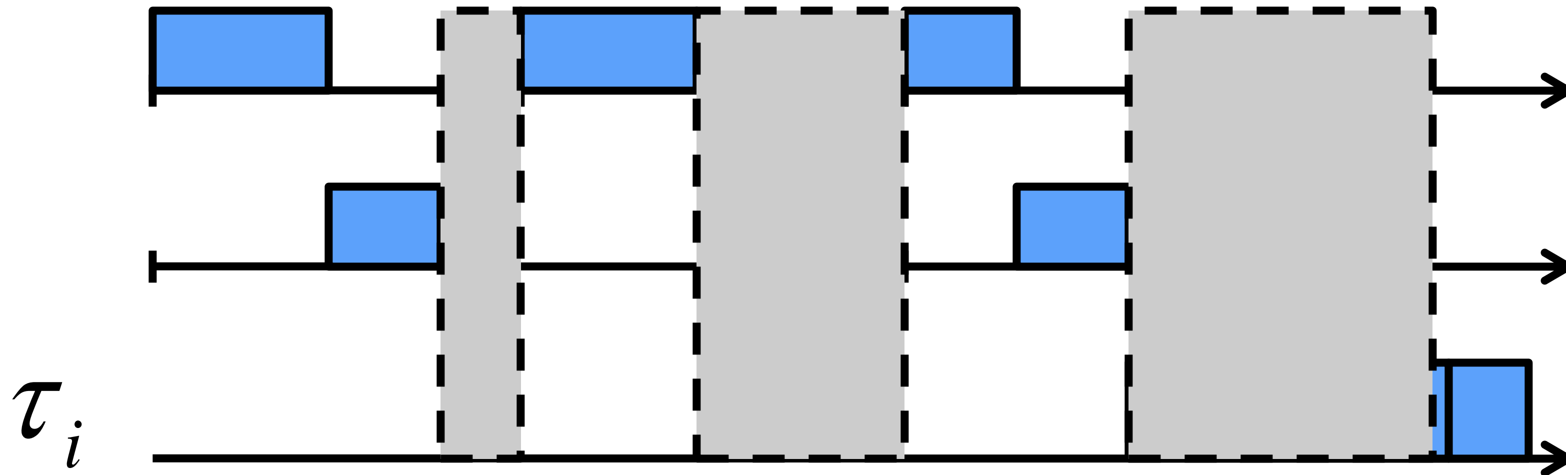
Slack Normal Mode

Slack Critical Mode

© 2016 Carnegie Mellon University

[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.

Calculating The Zero-Slack Instant



New slack can open after each iteration

Needs to repeat until no new slack opens

© 2016 Carnegie Mellon University

[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.

ZSRM Properties

Subsumes RM

- If criticalities are aligned to priorities
- No critical mode

Subsumes CAPA

- If not enough slack, only critical mode

Graceful Degradation

- In overloads, deadlines are missed in reverse criticality order

© 2016 Carnegie Mellon University
[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.

Implementation

ZSRM

Scheduling algorithm calculates zero-slack instants offline

Linux/ RK

- Resource reservation in Linux
  - CPU, Net, Mem, Disk
- Bundled into resource sets that provide a form of virtual machine
- Multiple implementations
  - Nano/RK for sensor networks

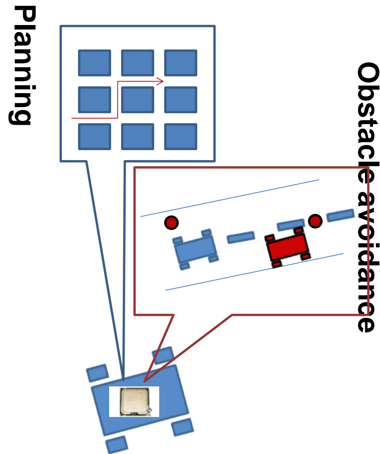
Special Zero-Slack Reserves

- Switch to critical mode
  - Stop lower-criticality tasks on zero-slack instant
- Tasks in critical mode in stack

© 2016 Carnegie Mellon University

[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.

What about Shared Resources?



- Consider a new medium priority and Medium criticality task (say v2v task)
- Let Planning and Obstacle Avoidance share an Obstacle Map Data Structure

Potentially leads to unbounded Criticality Inversion!!!

© 2016 Carnegie Mellon University

[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.

Priority and Criticality Inversion

Priority Inversion

Low Criticality

High Priority

Arrival

Obtain Lock

Release Lock

Medium Criticality

Medium Priority

Arrival

Zero-Slack Instant

Deadline

Criticality Inversion

High Criticality

Low Priority

Arrival

Zero-Slack Instant

Wait on Lock

© 2016 Carnegie Mellon University

[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.

Blocking in Zero-Slack Scheduling

A job J_h waiting for a job J_l to exit critical section $Z_{l,k}$ is considered to be *blocked* at time t, *if and only if* one of the following conditions is satisfied at t:

1) The priority of J_l is lower than J_h's priority and J_l is running in its ***normal*** mode.

2) The criticality of J_l is lower than J_h's criticality and J_h is running in its ***critical*** mode.

© 2016 Carnegie Mellon University
[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.

Priority and Criticality Inheritance Protocol (PCIP)

© 2016 Carnegie Mellon University
[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.

PCIP Definition

A task τ_i that holds a lock to a resource can inherit the priority from a task τ_j and the criticality from a task τ_k (τ_k can be the same as τ_j), both requesting a lock to the resource held by τ_i as follows:

- τ_i *inherits the priority* of τ_j if τ_j's priority is higher.
  - This inherited priority has an immediate effect on the scheduling of τ_i
- τ_i *inherits the criticality* of τ_k if τ_k's criticality is higher.
  - This criticality is used by τ_i immediately as soon as τ_k requests the lock held by τ_i

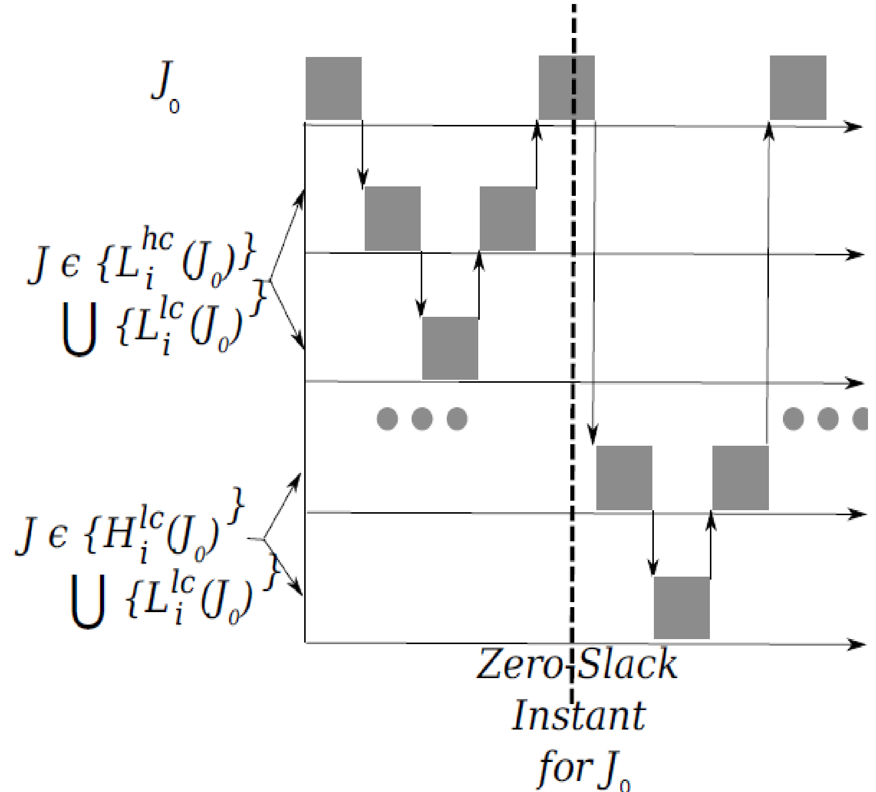
© 2016 Carnegie Mellon University

[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.

PCIP Possible Blocking

Consider a Job J_0

- $L_i^{hc}(J_0)$ is the set of jobs with **lower priority and higher criticality**
- $L_i^{lc}(J_0)$ is the set of jobs with **lower priority and lower criticality**
- $H_i^{lc}(J_0)$ is the set of jobs with **higher priority and lower criticality**



ther than $H_i^{hc}(J_0)$ all other sets can lead to priority or criticality inversio

$H_i^{hc}(J_0)$ is the set of jobs with

© 2016 Carnegie Mellon University

[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.

PCIP Properties

- Under PCIP, given a job J_0 for which there are n jobs $\{J_1, J_2, \ldots, J_n\}$, with J_i in $\{ L_i^{hc}(J_0) \cup L_i^{lc}(J_0) \cup H_i^{lc}(J_0) \}$, job J_0 can be blocked for at most the duration of one critical section in each of $\beta^*_{0,i}$.
  - where,

    $\beta^*_{0,i}$ is the set of critical sections of J_i that can block J_0

- Under PCIP, if there are *"m"* locks which can block job *J*, then *J* can be blocked at most *"m"* times in its *normal* mode and blocked at most *"m"* times in its *critical* mode.

© 2016 Carnegie Mellon University
[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.

PCIP Illustration

Task τ_1 Inherits Criticality of τ_2

Task τ_0 Inherits Criticality of τ_1 and τ_2

Low Criticality (P_1 V_1) *High Priority* — τ_0

Medium Criticality (P_1 P_2 V_2 V_1) *Medium Priority* — τ_1

High Criticality (P_2 V_2) *Low Priority* — τ_2

Task	C	C^o	T	D	Criticality	Priority
τ_0	10	70	100	100	2	0
τ_1	20	100	200	200	1	1
τ_2	40	200	400	400	0	2

© 2016 Carnegie Mellon University
[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.

Priority and Criticality Ceiling Protocol (PCCP)

© 2016 Carnegie Mellon University
[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.

PCCP Definition

- Each lock is assigned <u>both</u> a *priority ceiling* and a *criticality ceiling*
  - Priority ceiling is the highest possible priority of any locker of the lock
  - Criticality ceiling is the highest possible criticality of any locker of the lock
- Both the *priority ceiling* and the *criticality ceiling* of a lock are acquired by task whenever it holds the lock

© 2016 Carnegie Mellon University
[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.

PCCP – Maximum Blocking

- Each job J can only be blocked twice
  - At most once in ***Normal*** execution mode
  - At most once in ***Critical*** execution mode

- Each job J_w can block job J only once
  - Otherwise, J_w is $L_i^{lc}(J_0)$
  - And, Job J has to be blocked by J_w once in Normal mode
  - However, J_w cannot obtain the processor again as it is in $L_i^{lc}(J_0)$!!!

© 2016 Carnegie Mellon University
[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.

PCCP - No Deadlocks

- Under PCCP, no job J_k can preempt another job J_i while J_i holds a lock (i.e. is inside the critical section) that is also accessed by J_k.
- PCCP prevents *Transitive Blocking*
- PCCP *prevents Deadlocks*

© 2016 Carnegie Mellon University
[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.

PCCP Illustration

Task τ_0 acquires the Priority and Criticality Ceiling of Lock (P_1, V_1)

Task τ_1 acquires the Priority and Criticality Ceiling of Lock (P_2, V_2)

Low Criticality / *High Priority*	(P_1 V_1)	τ_0	
Medium Criticality / *Medium Priority*	(P_1 P_2 V_2 V_1)	τ_1	$P_2 P_1 V_1 V_2$
High Criticality / *Low Priority*	(P_2 V_2)	τ_2	$P_2 V_2$

ZS_1 ZS_2

50 70 80 190

Task	C	C^o	T	D	Criticality	Priority
τ_0	10	70	100	100	2	0
τ_1	20	100	200	200	1	1
τ_2	40	200	400	400	0	2

© 2016 Carnegie Mellon University

[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.

PCIP Blocking Term Analysis

- PCIP Blocking Term B_i for Task τ_i

$$B_i = \min\left(\sum_{\tau_j \in \{H_i^{lc} \cup L_i^{lc} \cup L_i^{hc}\}} \lambda(\beta^*_{i,j}), \sum_{\Psi_{i,k} \in \Psi_i} 2\Lambda(\Psi_{i,k}) \right)$$

where,

- $\beta^*_{i,j}$ is the set of critical sections of τ_j that can block τ_i
- $\lambda(\beta^*_{i,j})$ is the length of the *longest critical sections* of $\beta^*_{i,j}$ that can block task τ_i
- $\Lambda(\Psi_{i,j})$ is the length of the critical section protected by lock $\Psi_{i,j}$

© 2016 Carnegie Mellon University

[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.

PCCP Blocking Term Analysis

- PCCP Blocking Term B_i for Task τ_i

$$B_i = \max_{\tau_j \in \{H_i^{lc} \cup L_i^{lc} \cup L_i^{hc}\}} 2\lambda(\beta^*_{i,j})$$

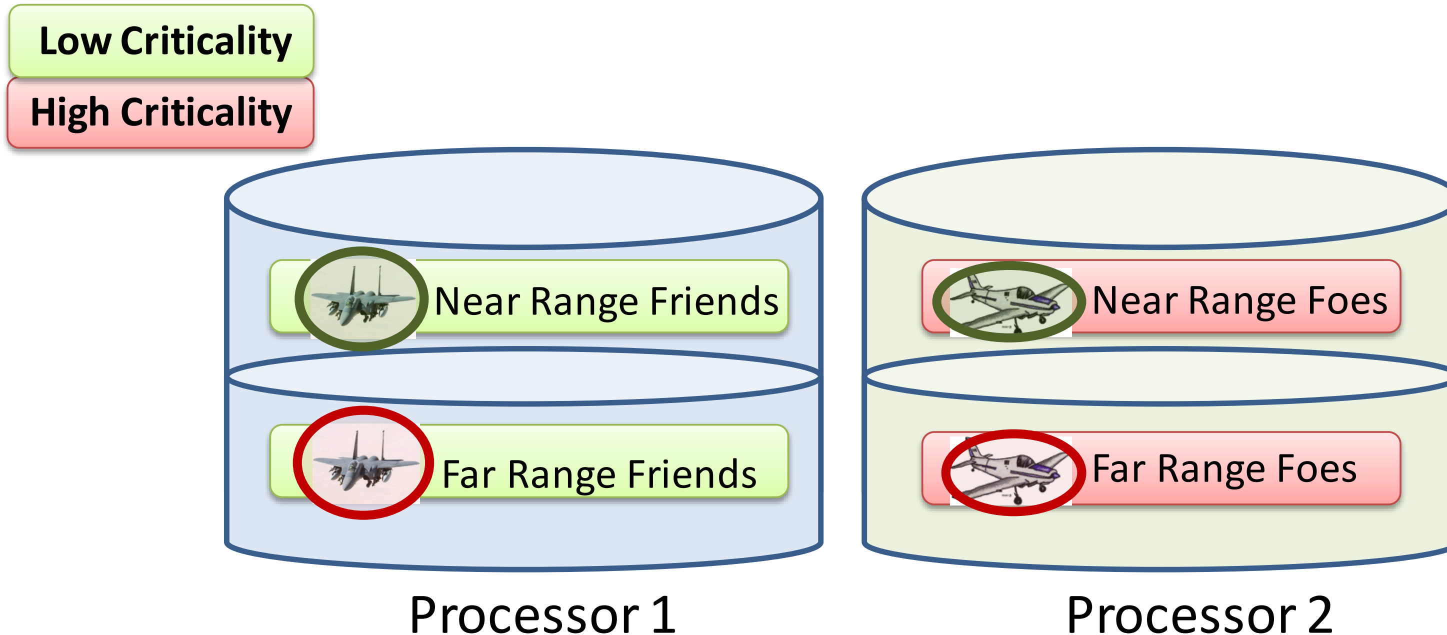
where,

- $\beta^*_{i,j}$ is the set of critical sections of τ_j that can block task τ_i
- $\lambda(\beta^*_{i,j})$ is the length of the longest critical sections of $\beta^*_{i,j}$ that can block task τ_i

© 2016 Carnegie Mellon University

[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.

Criticality isolation strategy (S)



	H	L
Both overload	0	0
High criticality overload	0	1
Low criticality overload	1	0
No overload	1	1

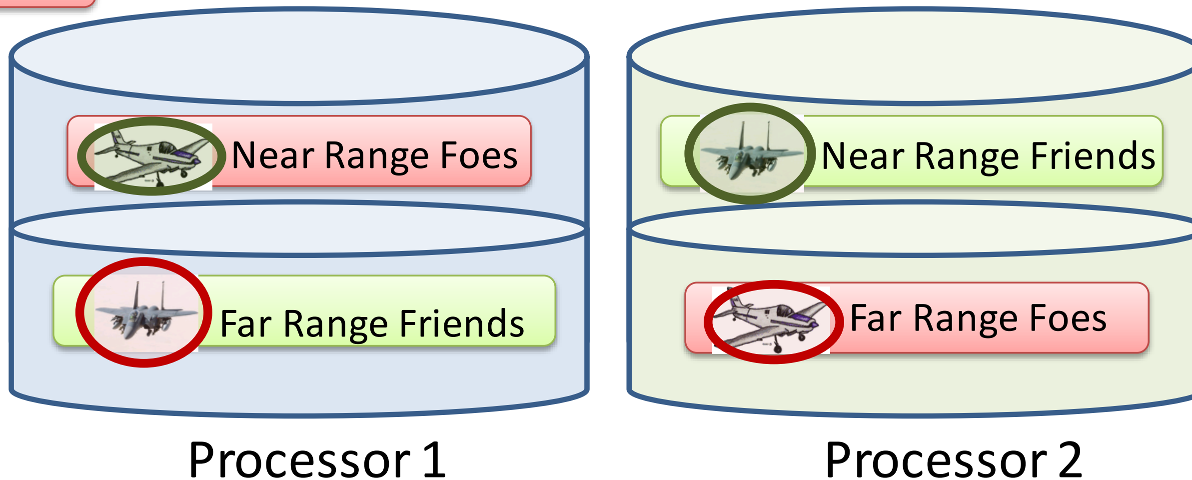
Assume that the system is schedulable without overloads

Under overloads only one task can meet its deadline

© 2016 Carnegie Mellon University

[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.

Criticality mixture strategy (T)



Processor 1

Processor 2

	H	L
Both overload	1	0
High criticality overload	1	0
Low criticality overload	1	0
No overload	1	1

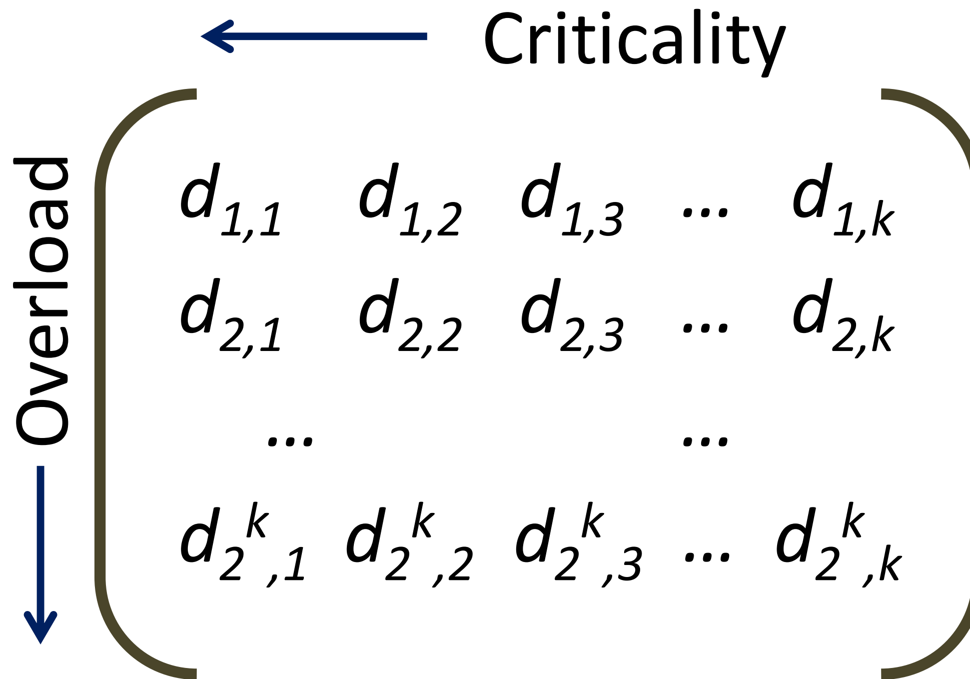
T is better than S

Assume that the system
is schedulable without overloads
Under overloads only one
task can meet its deadline
Assume that a uniprocessor
mixed-criticality scheduling algorithm
like ZRSM is used within each processor

© 2016 Carnegie Mellon University

[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.

Generalization: Ductility Matrix



Say we have ‘k’ criticality levels

2^k possible overload scenarios

- *All criticality levels overload* to *No overload*

© 2016 Carnegie Mellon University

[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.

Quantification of Ductility

$$P_d(\mathrm{D}) = \sum_{c=1}^{k} \left\{ \frac{1}{2^c} \frac{\sum_{r=1}^{2^k} d_{r,c}}{2^k} \right\}$$

*Scheme **S***

	H	L
Both overload	0	0
High criticality overload	0	1
Low criticality overload	1	0
No overload	1	1

*Scheme **T***

	H	L
Both overload	1	0
High criticality overload	1	0
Low criticality overload	1	0
No overload	1	1

For S, P_d (D) = 0.375

For T, P_d (D) = 0.5625

Shows that ***T*** is better than ***S***

Other Projection functions can be used

P_d(D) favors the more critical tasks **exponentially** over the lower criticality tasks

© 2016 Carnegie Mellon University

[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.

Outline

Mixed-criticality task scheduling problem

Zero-slack scheduling for uni-processors

- Zero-slack metrics & properties

Generalizing resource allocation to distributed mixed criticality tasks

- Generalized metric: Ductility matrix

Compress-on-Overload Packing (COP)

- COP Performance

Radar surveillance case study

Conclusions

© 2016 Carnegie Mellon University

[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.

Compress-on-Overload Packing (COP)

More critical

Less critical

$\frac{C}{T}$

$\frac{C^o}{T}$

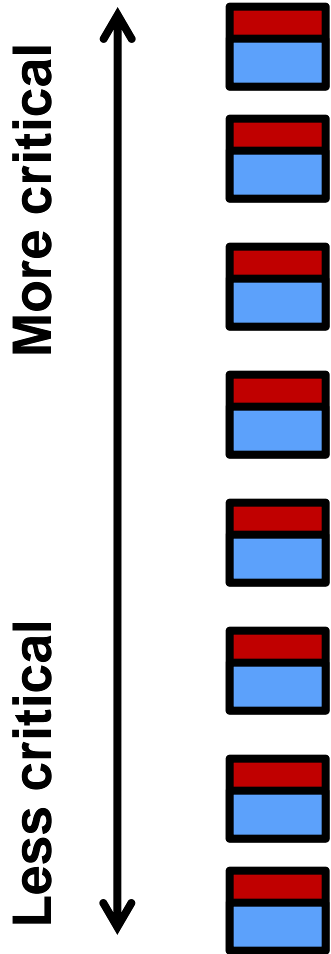
Proc 1 Proc 2

$$\sum \frac{C_i}{T_i}$$

© 2016 Carnegie Mellon University

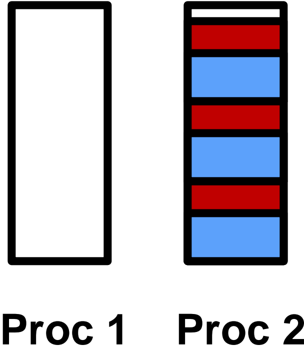
[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Compress-on-Overload Packing (COP)



Phase 1: Pack by criticality then size

object size = $\frac{C_i^o}{T_i}$



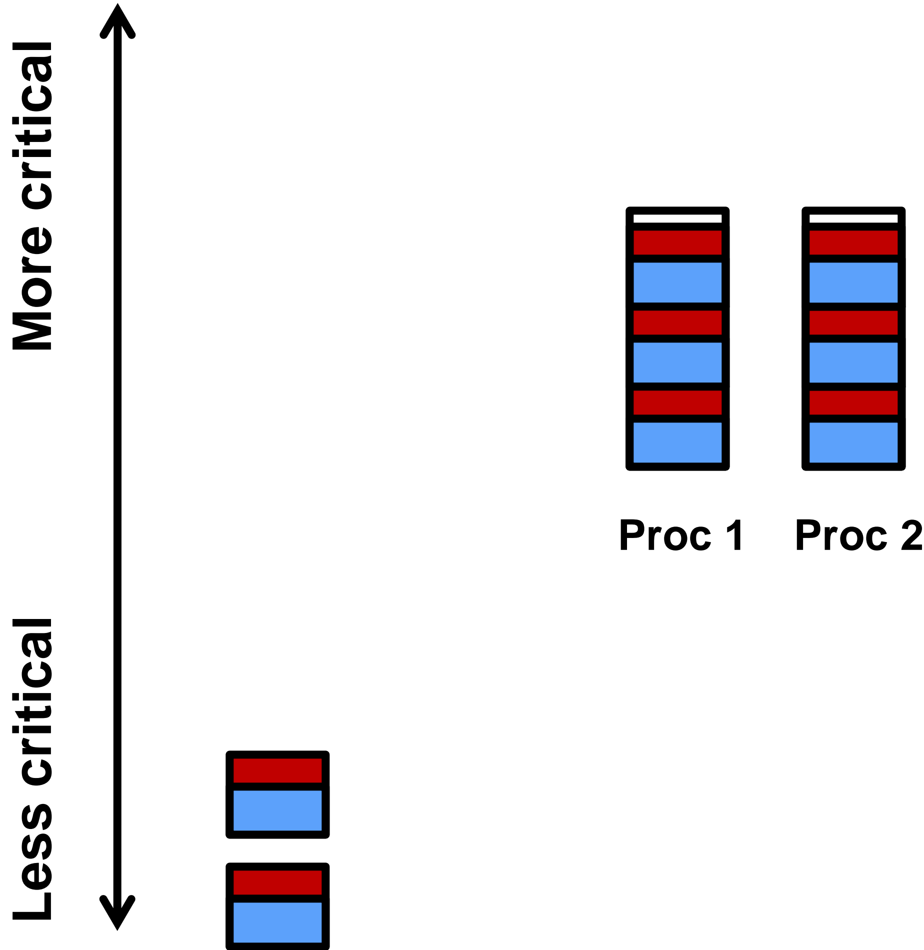
© 2016 Carnegie Mellon University

[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.

Compress-on-Overload Packing (COP)

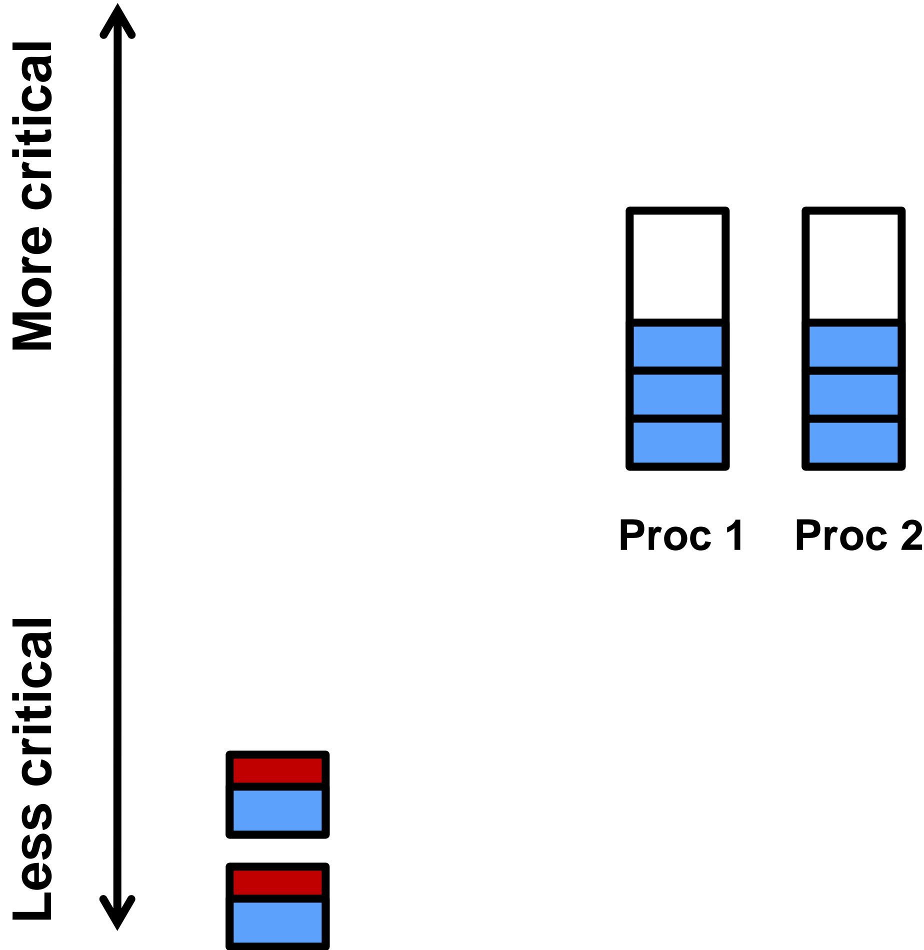
Phase 2: Pack by criticality then size

object size = $\frac{C_i}{T_i}$



© 2016 Carnegie Mellon University
[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.

Compress-on-Overload Packing (COP)



Phase 2: Pack by criticality then size

object size = $\frac{C_i}{T_i}$

© 2016 Carnegie Mellon University
[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.

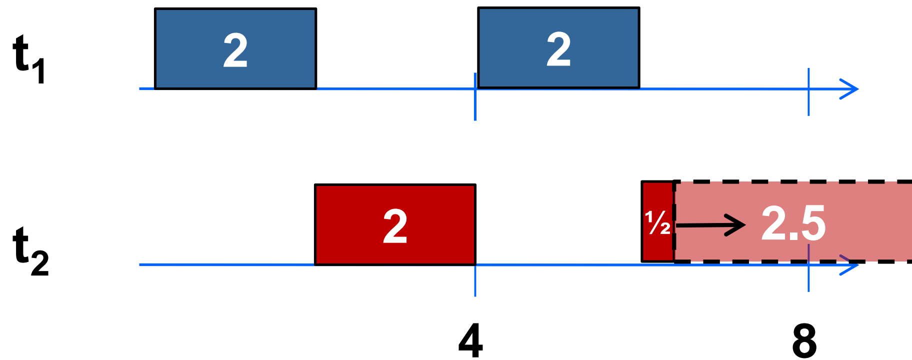
COP Performance

1.2
1
0.8
0.6
0.4
0.2
0
Ductility
4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Number of Processors
COP(FFD)
COP(BFD)
COP(WFD)
WFD

© 2016 Carnegie Mellon University
[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.

Overloading in Mixed-Criticality Systems

Task	Period	Criticality	WCET	NCET
t_1 Surveillance Cov.	4	Mission	2	2
t_2 Collision Avoid.	8	Safety	5	2.5

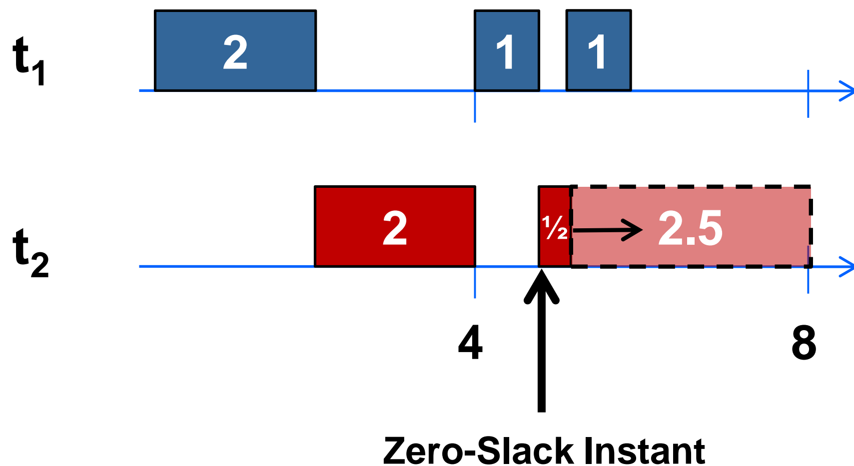


© 2016 Carnegie Mellon University

[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.

Zero-Slack Rate Monotonic

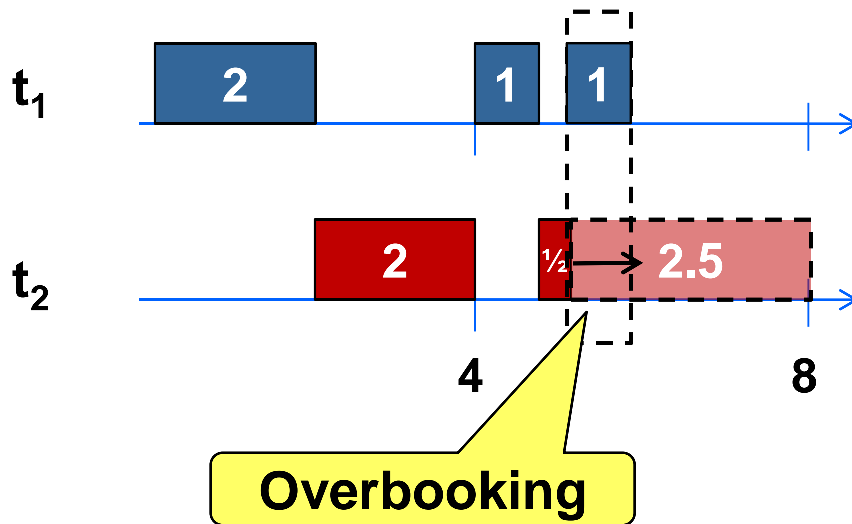
Task	Period	Criticality	WCET	NCET
t_1 Surveillance Cov.	4	Mission	2	2
t_2 Collision Avoid.	8	Safety	5	2.5



© 2016 Carnegie Mellon University
[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.

Zero-Slack Rate Monotonic

Task	Period	Criticality	WCET	NCET
t_1 Surveillance Cov.	4	Mission	2	2
t_2 Collision Avoid.	8	Safety	5	2.5

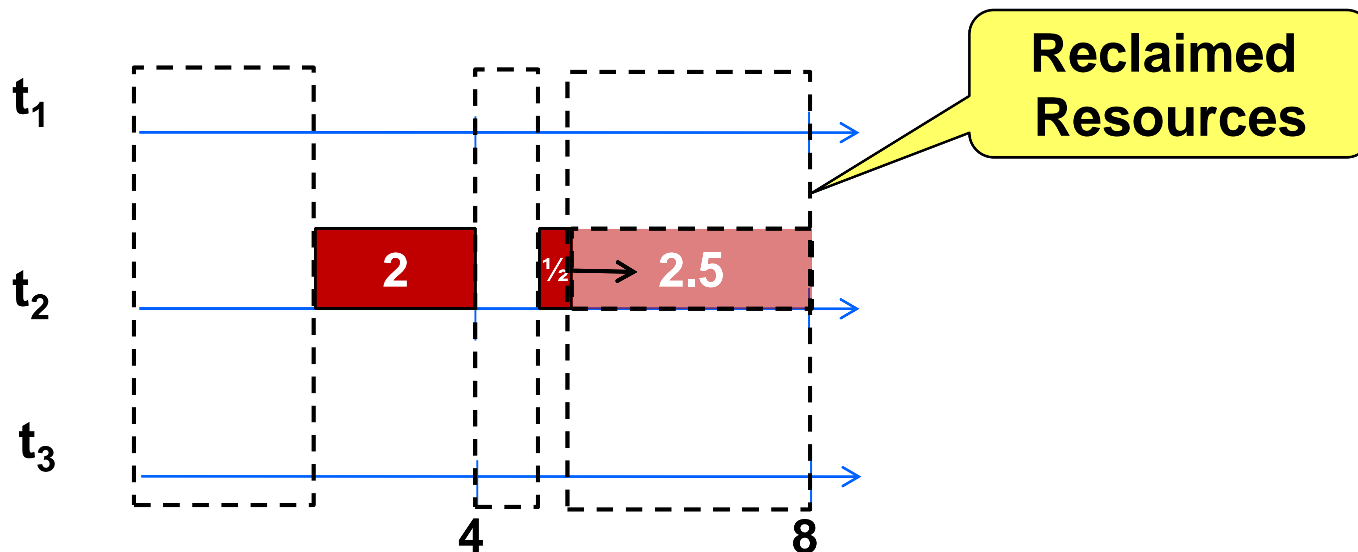


© 2016 Carnegie Mellon University

[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.

Reclaiming Resources in Mixed-Criticality Systems

Task	Period	Criticality	WCET	NCET	Utility
t_1 Surveillance Cov.	4	Mission	2	2	{2,2.5}
t_2 Collision Avoid.	8	Safety	5	2.5	
t_3 Amount of Intelligence	4	Mission	2	2	{2,2.5}

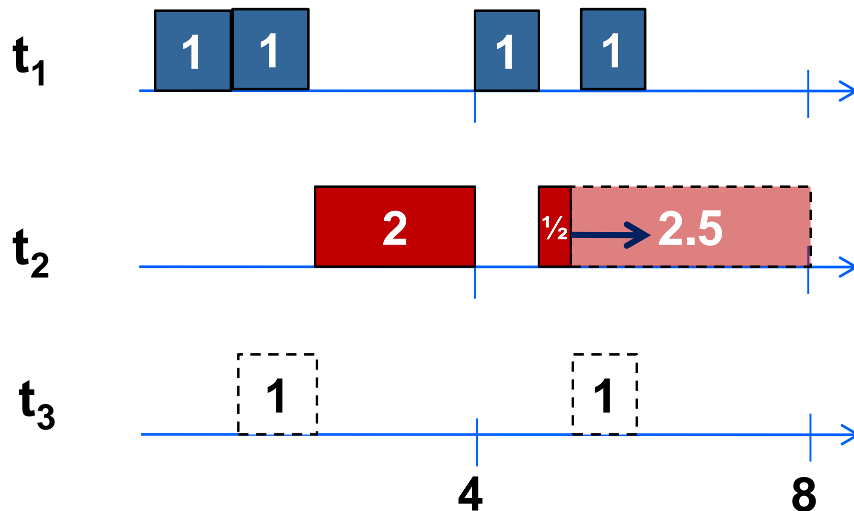


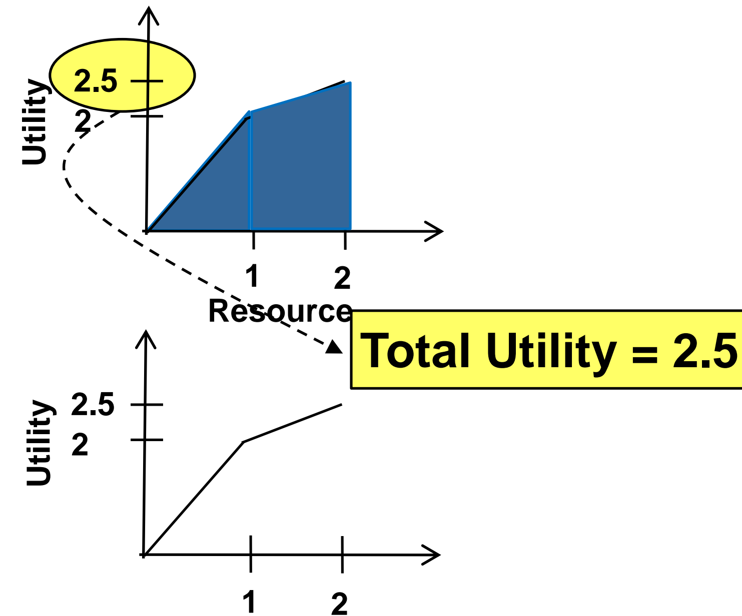
© 2016 Carnegie Mellon University

[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.

Using Reclaimed Resources to Maximized Utility

Task	Period	Criticality	WCET	NCET	Utility Levels
t_1 Surveillance Cov.	4	Mission	2	2	{2,2.5}
t_2 Collision Avoid.	8	Safety	5	2.5	
t_3 Amount of Intelligence	4	Mission	2	2	{2,2.5}



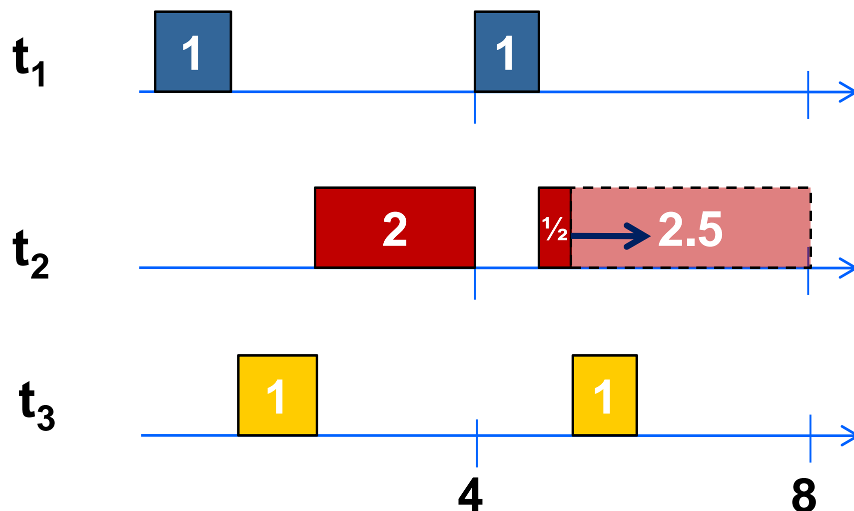


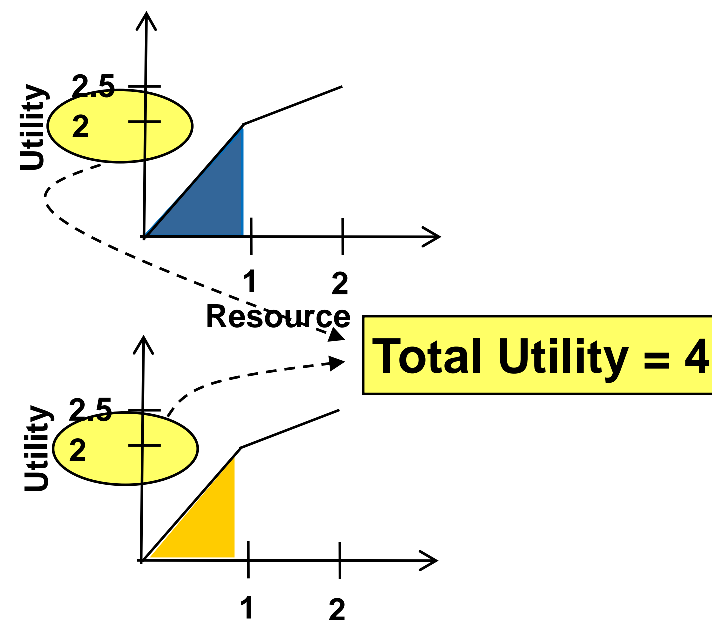
Utility Diminishes: Utility ≠ Criticality

© 2016 Carnegie Mellon University
[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.

Using Reclaimed Resources to Maximized Utility

Task	Period	Criticality	WCET	NCET	Utility Levels
t_1 Surveillance Cov.	4	Mission	2	2	{2,2.5}
t_2 Collision Avoid.	8	Safety	5	2.5	
t_3 Amount of Intelligence	4	Mission	2	2	{2,2.5}





ZS-QRAM: More mission-critical utility from same resources

[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.

ZSQRAM: Period Degradation

$$\frac{C_i = .25}{T_i = 2}$$

τ_i: ¼ ¼ ¼

0 1 2 3 4

$$\frac{C_j = 2}{T_{j,1} = 4, T_{j,2} =}$$

τ_j: 1¾ ¼

0 1 2 3 4

$$\frac{C_k = 2.5, C_k^o = 5}{T_k = 8}$$

τ_k: 1½ 1 ¾ 1¾

0 1 2 3 4 5 6 7 8

ZS Instant

10 8 6 4 2

$\frac{2.5}{16}$ $\frac{2.5}{8}$ $\frac{5}{16}$ $\frac{5}{8}$ 100%

© 2016 Carnegie Mellon University

[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.

ZSQRAM: Period Degradation

$$\frac{C_i = .25}{T_i = 2}$$

τ_i: ¼ ¼ ¼ (time axis 0 1 2 3 4)

$$\frac{C_j = 2}{T_{j,1} = 4, T_{j,2}}$$

τ_j: 1¾ ¼ (time axis 0 1 2 3 4)

$$\frac{C_k = 2.5, C_k^o = 5}{T_k = 8}$$

τ_k: 1½ 1 ¾ 1¾ (time axis 0 1 2 3 4 5 6 7 8)

ZS Instant

Chart: y-axis 2, 4, 6, 8, 10; x-axis 100%; x labels $\frac{2.5}{16}$, $\frac{2}{8}$, $\frac{2.5}{8}$, $\frac{5}{16}$, $\frac{2}{4}$, $\frac{5}{8}$

© 2016 Carnegie Mellon University

[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.