Computer Vision for Embedded Systems

Yung-Hsiang Lu
Purdue University
yunglu@purdue.edu

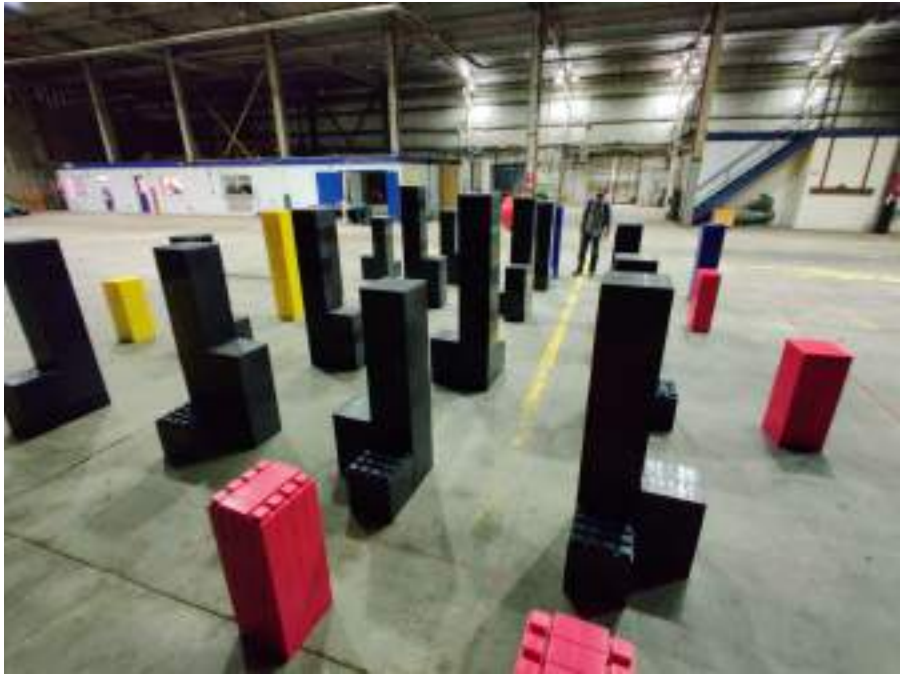
PURDUE UNIVERSITY

IEEE Autonomous UAV Challenge

@Purdue University

Purdue UAV Research and Test Facility (PURT)

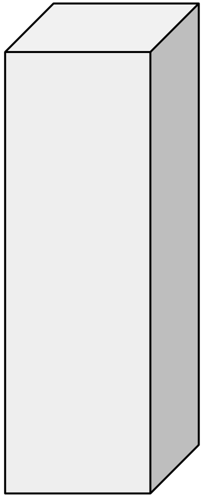
Indoor, with motion capture system, at Purdue Airport



Timing Requirements

$$t = \frac{d}{v}$$

$$t > t_{cv} + t_{ctrl} + t_r$$



https://www.dreamstime.com/stock-illustration-hand-draw-illustration-aerial-vehicle-quadrocopter-air-drone-hovering-drone-sketch-image69534927

Real-time Vision Definitions

- Before the next incoming data at a specified rate
- As fast as the slowest components in the system
- No later than pre-defined acceptable delays
- “***Fast***” is arbitrary, use-case dependent

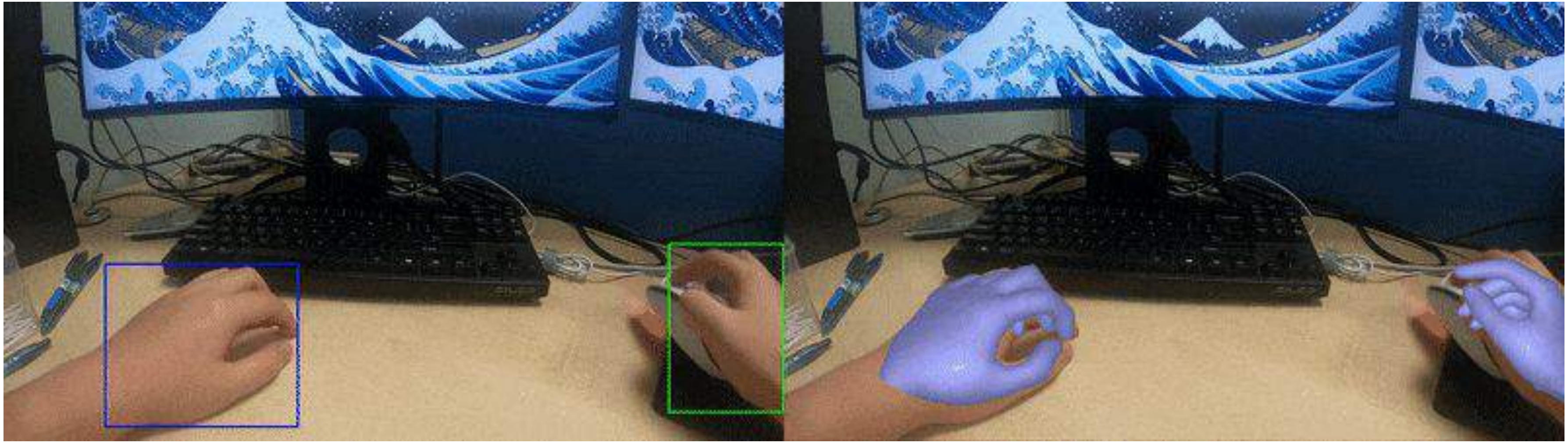



https://becominghuman.ai/how-to-improve-computer-vision-in-ai-drones-using-image-annotation-services-e67507457eb2
https://www.rsipvision.com/press-release-rsip-vision-ceo-ai-in-medical-devices/
https://www.inc.com/kevin-j-ryan/self-driving-cars-powered-by-people-playing-games-mighty-ai.html
https://www.ennomotive.com/computer-vision-in-manufacturing-opportunity-or-thread/

Interactions with the physical world
(real-time = meet deadline)

Real-time Vision Uses

Pose estimation for motion capture



References: https://github.com/facebookresearch/frankmocap

Why Real-Time in Computer Vision?

https://www.livescience.com/61596-animals-with-cameras-pbs.html
https://vicharkness.co.uk/2015/04/06/putting-cameras-on-birds/
https://medium.com/syncedreview/new-ttnet-table-tennis-model-accelerates-dl-in-sports-analysis-666dbfd142f1

DOI:10.1145/1506409.1506426

The passage of time is essential to ensuring the repeatability and predictability of software and networks in cyber-physical systems.

BY EDWARD A. LEE

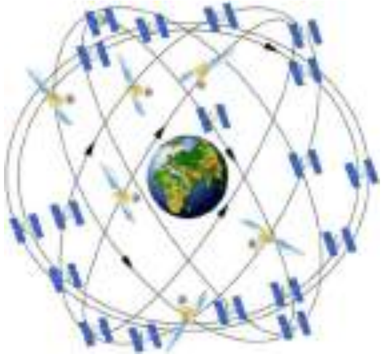
Computing Needs Time

Communications of the ACM Volume 52 Issue 5 May 2009 pp 70–79

https://theconversation.com/linking-self-driving-cars-to-traffic-signals-might-help-pedestrians-give-them-the-green-light-132952

Special Relativity

- The speed of light in vacuum (0.3Mkm/s) is the limit of information passing
- modern processor about 3GHz, or 0.3 ns/cycle
- 1km needs 3 us (not considering processing time)
- 1km = 10,000 clock cycles
- GPS (Global Positioning System) uses special relatively to calculate time.



https://www.sciencedirect.com/topics/computer-science/global-positioning-system

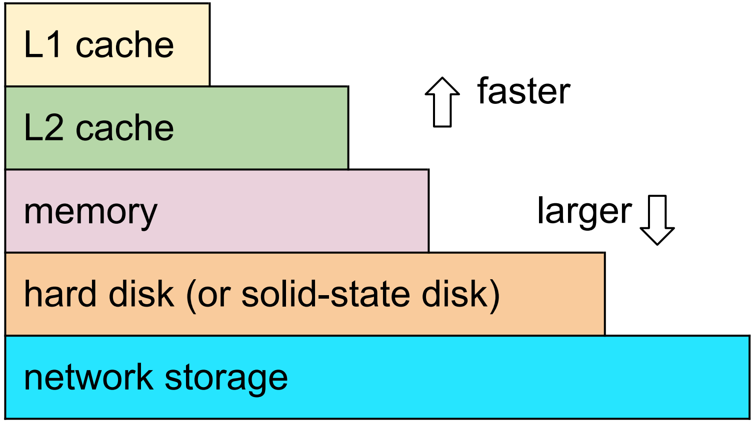
Hard vs Soft Real-Time

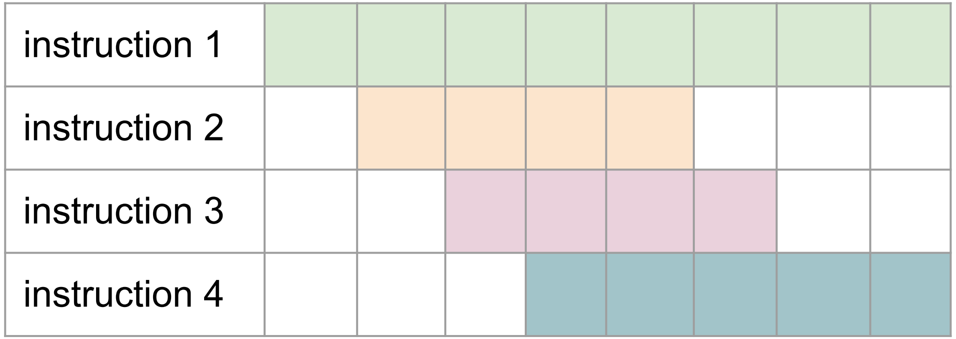
https://www.allaboutcircuits.com/technical-articles/introduction-to-real-time-embedded-systems/

https://www.allaboutcircuits.com/technical-articles/introduction-to-real-time-embedded-systems/

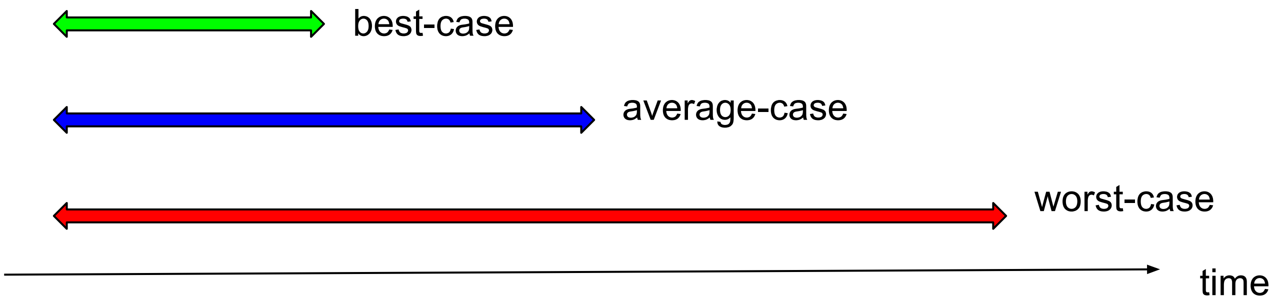
Why timing is difficult in modern computers?

- Branches
- Memory hierarchy
- Pipeline and superscalar in processors
- Speculative execution
- Interrupts
- Network delays





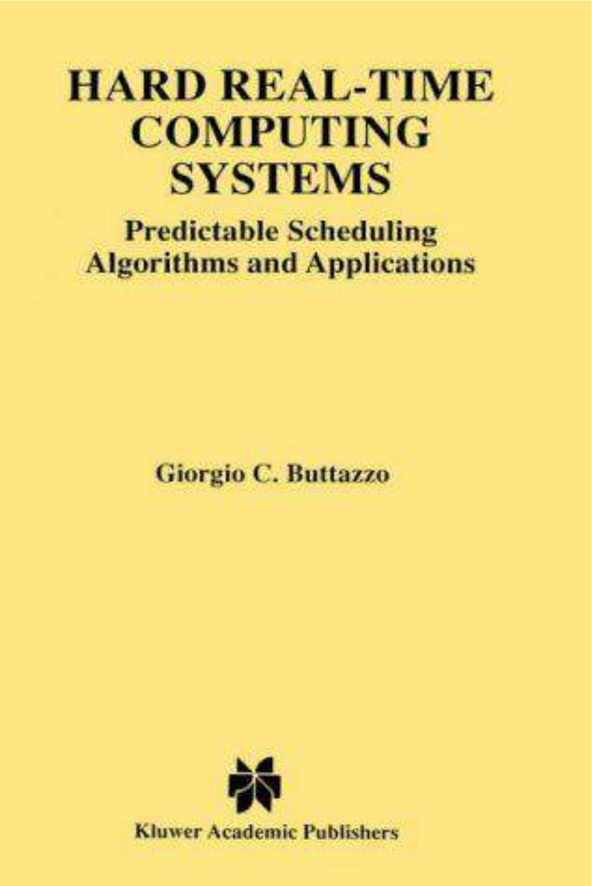
WCET (Worst-Case Execution Time)



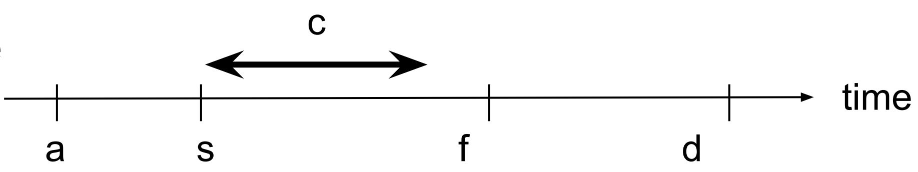
Scheduling for Real-Time

Most systems have # processors << # tasks e.g., a laptop processor has only several cores for running dozens of programs (web browser, email reader, text editor, background music, camera, spam checker …)

scheduling = decide which task to run and when



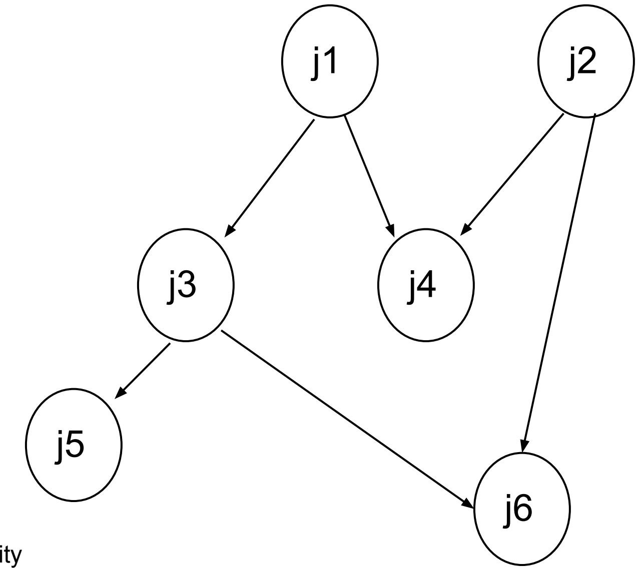
Task (Job) Model

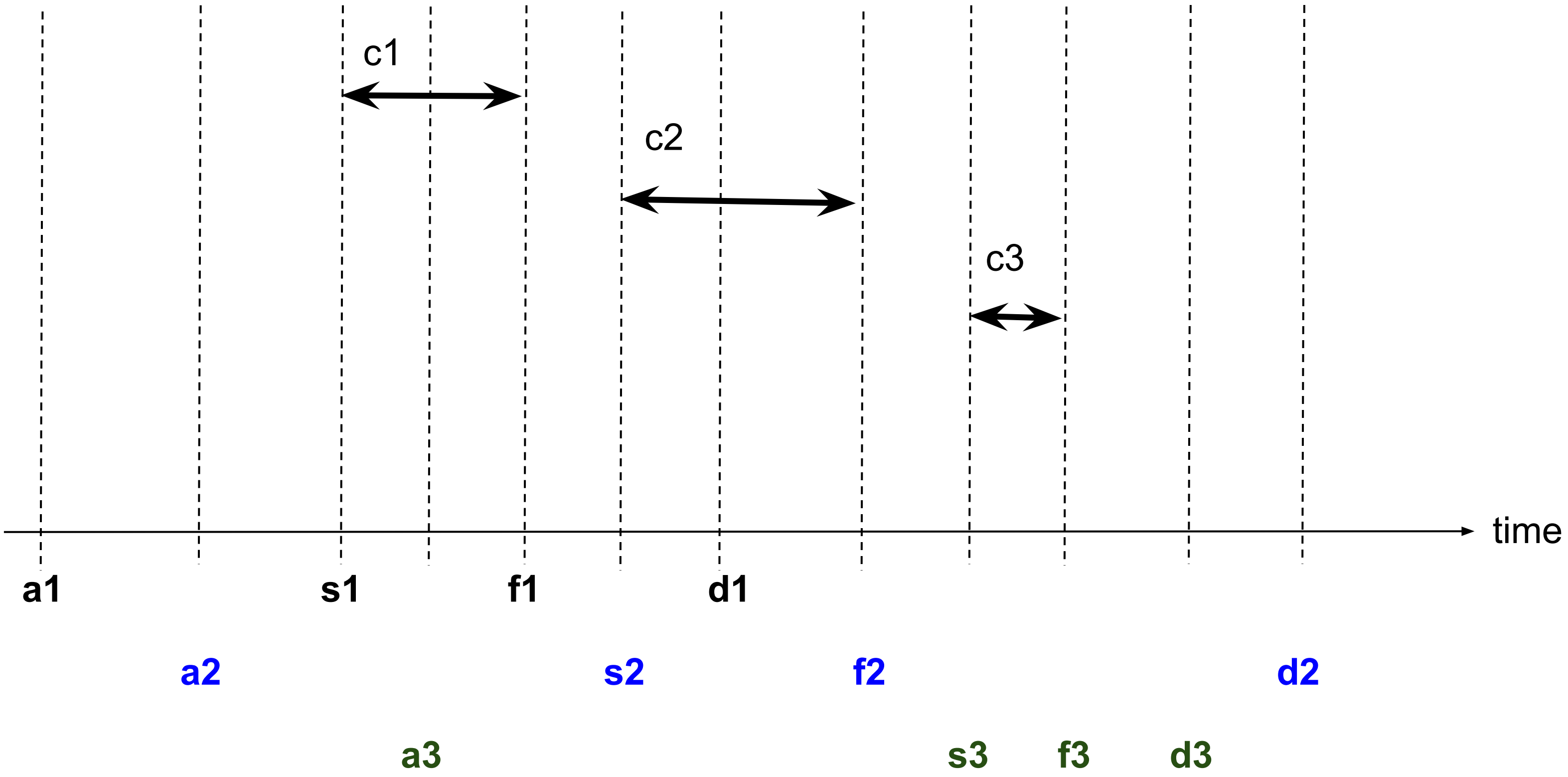


- a: task arrival time; d: task deadline
- s: starting time; f: finishing time
- c: computation time

Precedence:

- j1 and j2 can start immediately
- j3 can start after j1 finishes
- j4 can start after both j1 and j2 finish
- j5 can start after j3 finishes
- j6 can start after j2 and j3 finish




c1
c2
c3
time
a1
s1
f1
d1
a2
s2
f2
d2
a3
s3
f3
d3

Performance Metrics for n jobs

response time for j_i: $f_i - a_i$

average response time $\frac{1}{n}\sum_{i=1}^{n}(f_i - a_i)$

lateness: $f_i - d_i$

total execution time: $\sum_{i=1}^{n} c_i = \sum_{i=1}^{n}(f_i - s_i)$

number of deadline misses: U is the unit step function

$$\sum_{i=1}^{n} U(f_i - d_i)$$

Scheduler Characteristics

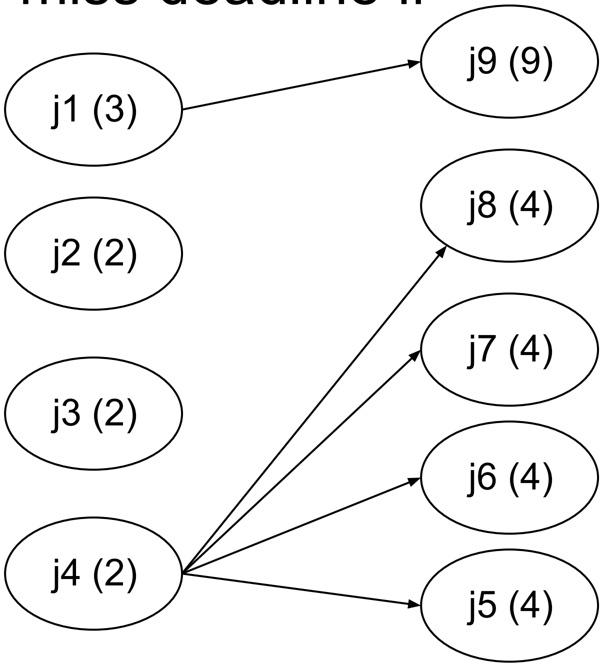
- preemptive: a running job can be interrupted and resumed
- static: scheduling decisions do not respond to run-time conditions
- dynamic: scheduling decisions respond to run-time conditions
- on-line: scheduling decisions are made while jobs are running
- off-line: scheduling decisions are made before jobs start running
- optimal: decisions are the best possible
- heuristic: decisions "make sense" but not necessarily optimal
- switching cost: time needed when changing jobs
- best-case, average-case, worst-case

Scheduling Anomalies

Scheduling Anomalies

Suppose a set of tasks can be scheduled (i.e., all deadlines are met) on a multi-core processor (Raspberry 4 uses Broadcom BCM2711 with quad cores). The tasks may miss deadline if

- more cores are available
- execution time becomes shorter
- precedence constraints are removed



Core 1	J1			J9											
Core 2	J2		J4		J5				J7						
Core 3	J3				J6				J8						
Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

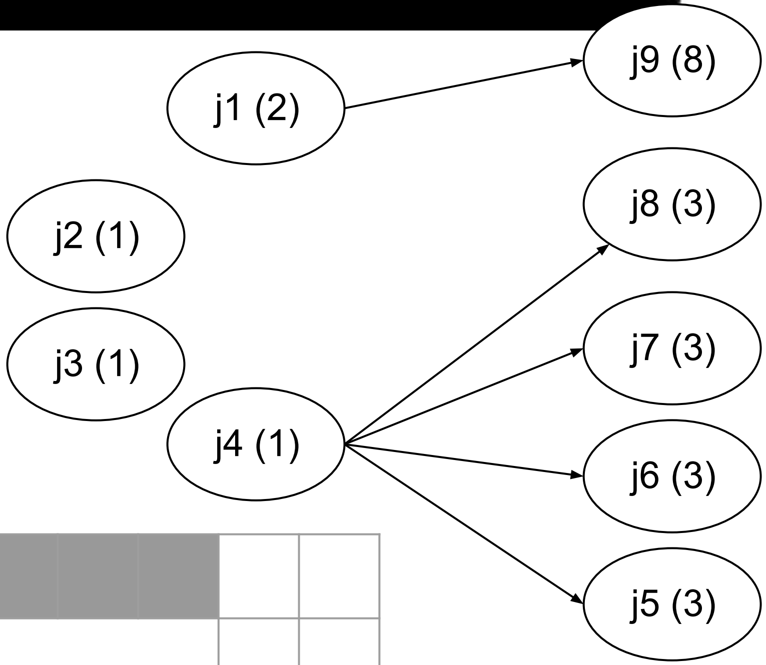
All jobs finished
at time 12

Add one more core

<table>
<tr><td>Core 1</td><td colspan="3">J1</td><td colspan="4">J8</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Core 2</td><td colspan="2">J2</td><td colspan="4">J5</td><td colspan="9">J9</td></tr>
<tr><td>Core 3</td><td colspan="2">J3</td><td colspan="4">J6</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Core 4</td><td colspan="2">J4</td><td colspan="4">J7</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Time</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr>
</table>

All jobs finished at time 15

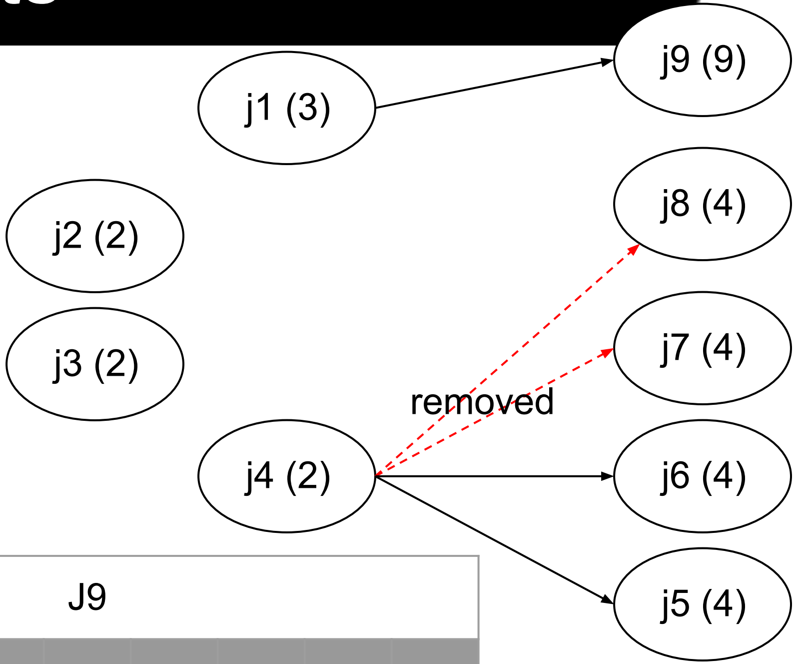
Reduce Execution Time by One



Core 1	J1		J5			J8									
Core 2	J2	J4	J6			J9									
Core 3	J3		J7												
Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Yung-Hsiang Lu, Purdue University

Remove precedence constraints



<table>
<tr><td>Core 1</td><td colspan="3">J1</td><td colspan="4">J8</td><td colspan="9">J9</td></tr>
<tr><td>Core 2</td><td colspan="2">J2</td><td colspan="2">J4</td><td colspan="4">J5</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Core 3</td><td colspan="2">J3</td><td colspan="4">J7</td><td colspan="4">J6</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Time</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td></tr>
</table>

Yung-Hsiang Lu, Purdue University

Scheduling Periodic Jobs

- A set of periodic jobs {J1, J2, …, Jn} with periods {T1, T2, …, Tn}.
- Each job's arrival time / deadline is the beginning / end of the period.
- Job Ji takes Ci to compute.
- No precedence constraint. No switching cost.
- processor utilization = $\sum_{i=1}^{n} \frac{C_i}{T_i}$

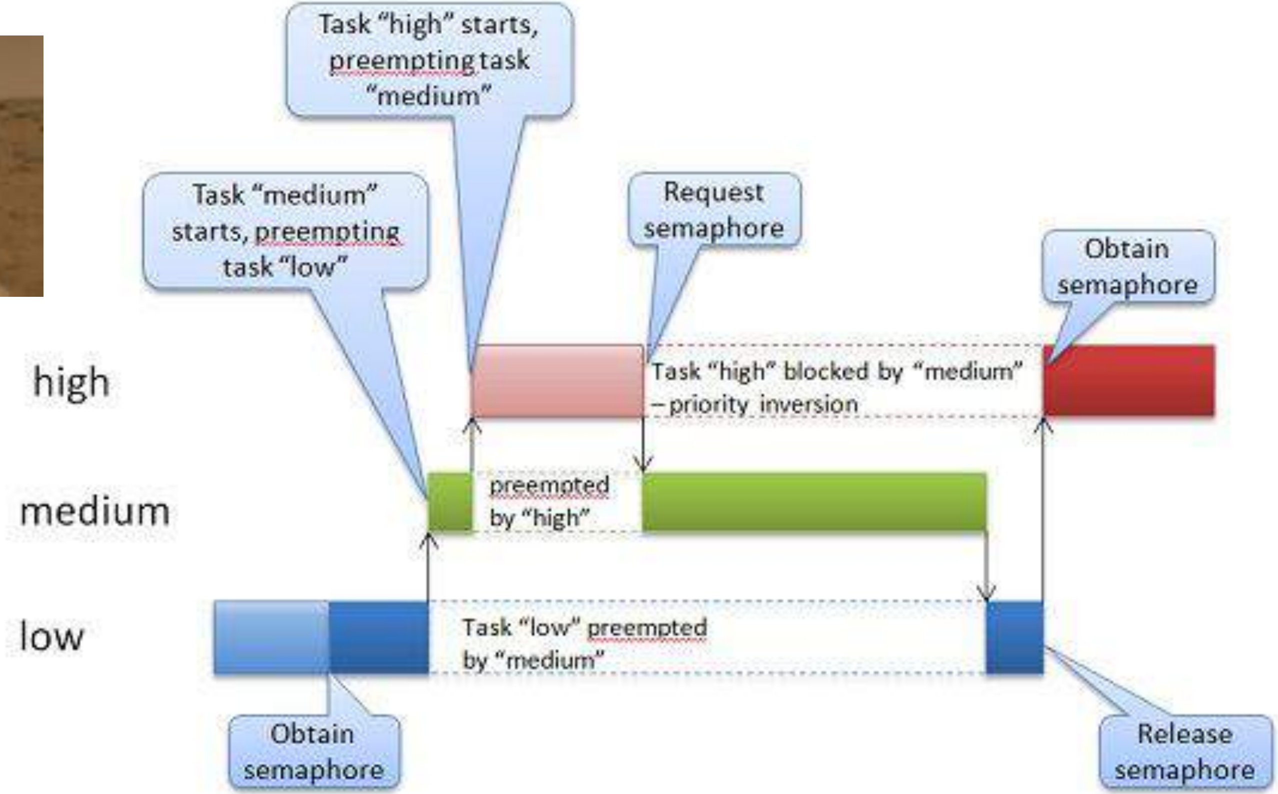
Rate Monotonic Scheduling of Periodic Tasks

- static scheduling
- fixed priority (more frequent tasks have higher priorities)
- preemptive (a running task stops for a new task of a higher priority)
- optimal among fixed-priority static scheduling
- guarantee schedulability if processor utilization is below ln 2 (about 0.69)

Earliest Deadline First

- dynamic scheduling
- dynamic priority (earlier deadline has a higher priority)
- preemptive (a running task stops for a new task of a higher priority)
- guarantee schedulability if processor utilization is below one

Priority Inversion



https://www.rapitasystems.com/blog/what-really-happened-software-mars-pathfinder-spacecraft

Yung-Hsiang Lu, Purdue University

Limitations of Existing Theories

- do not distinguish tasks of different severity of consequences
- zero switching cost
- known execution time
- static environment