# Schedulability in Real Time Systems

Paritosh Pandya

CS684, IIT Bombay
March 2022

# Real time, Embedded Systems

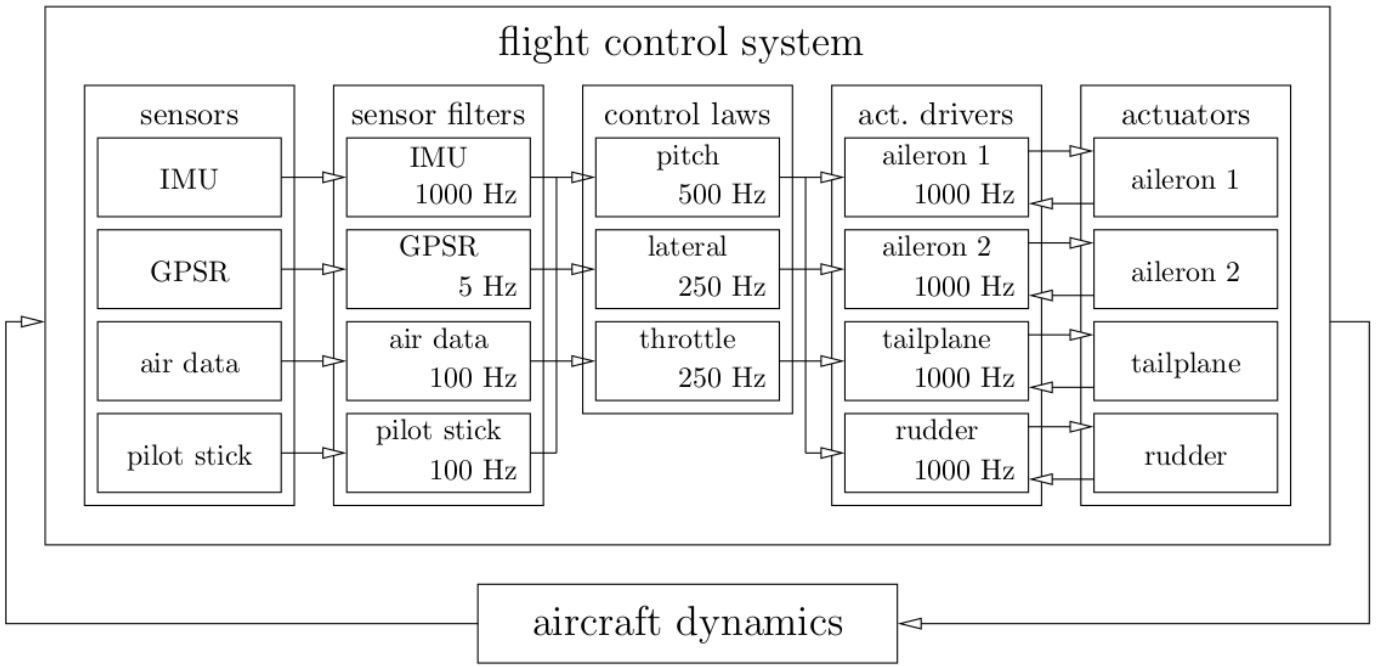
An assembly of electro-mechanical, optical, chemical components with sensors and actuators, connected to onboard computer.

- Program is typically orgainzed as a set of repeating tasks.
- A periodic task typically has the structure:

```
repeat every 10 ms
  { sense input;
    Compute;
    Actuate output;
  }
```

- (latancy) There are real-time requirements on delays between input and output.

# Example: Flight Control System



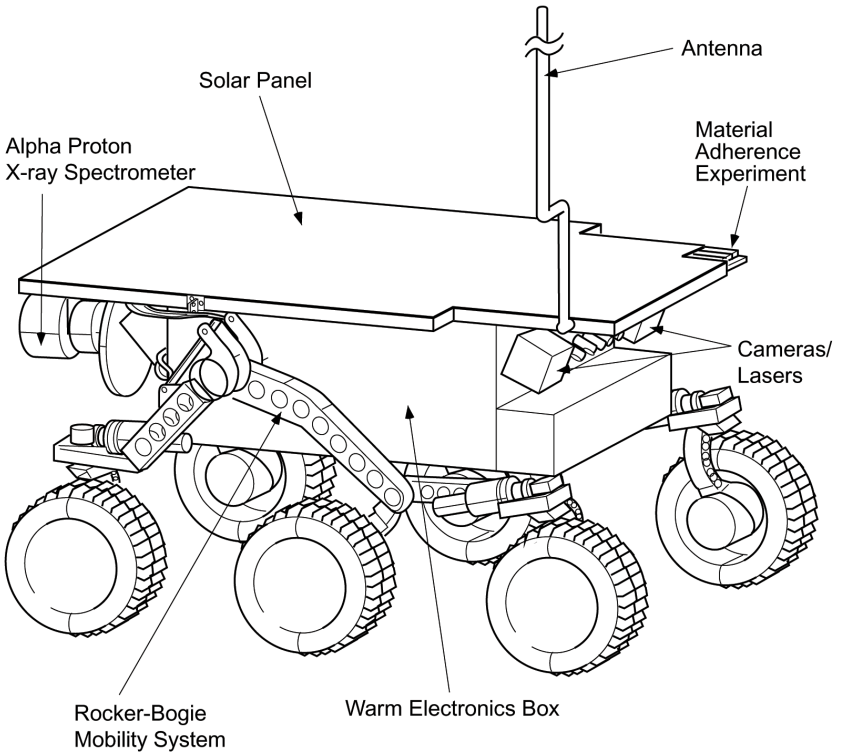

- A set of periodic processes.
- Interaction via shared memory.
- Execute on single microcontroller by sharing CPU
- Scheduling important to meet latency.

# Schedulability Analysis

Given set of tasks what kind of scheduling policy will allow all tasks to meet their deadlines (latency requirements) ?

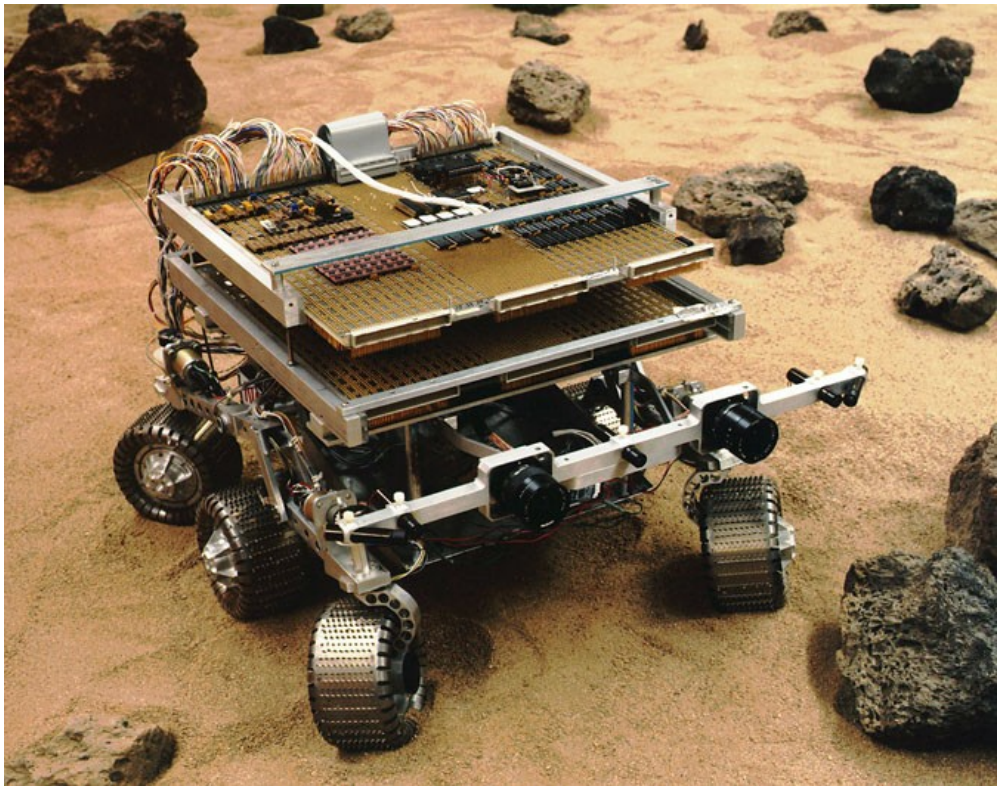
# Example: Mars Pathfinder (1)

NASA Mars Mission 4 July, 1997.

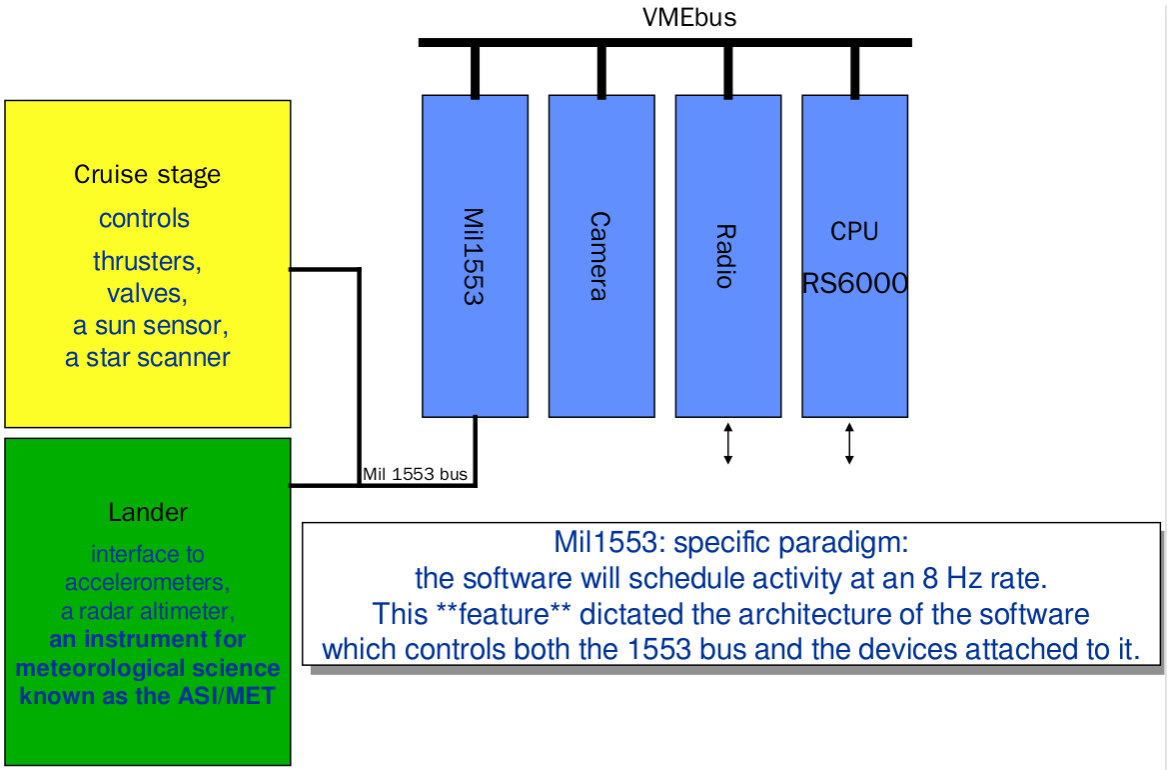



(Images courtsey NASA)

# Example: Mars Pathfinder (2)



# Example: Mars Rover (3)




Information Bus: A shared data structure for devices and processes of Lander and Cruise Control.

# Example: Mars Pathfinder (4)

## Pathfinder used VxWorks RTOS

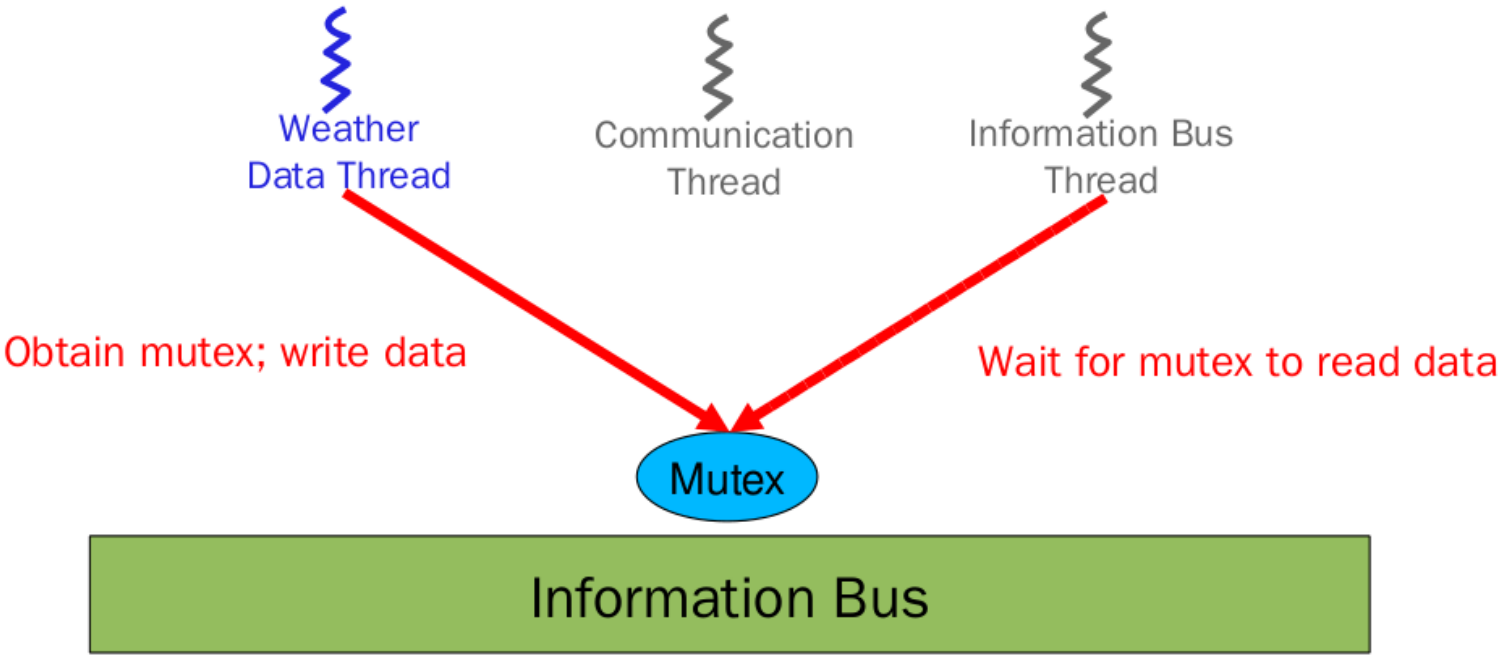
- Threads for the 1553 bus for data collection, scheduled on every 1/8th sec cycle.
- 3 periodic tasks
  - Task 1 – Information Bus Thread: Bus Manager

    high frequency, high priority
  - Task 2 – Communication Thread

    medium frequency / priority, high execution time
  - Task 3 – Weather Thread: Geological Data Gatherer

    low frequency, low priority
- Each checks if the other executed and completed in the previous cycle
  - If the check fails, this is a violation of a hard real-time guarantee and the system is reset

# Example: Mars Pathfinder (5)



# Mars Pathfinder Bug

NASA sent mars pathfinder Sojourn on 4 July 1997.

- Stopped working after 87 sol due to software error.

# Mars Pathfinder Bug

NASA sent mars pathfinder Sojourn on 4 July 1997.

- Stopped working after 87 sol due to software error.
- Diagnosed as a rare schedulability problem called Priority Inversion.

# Mars Pathfinder Bug

NASA sent mars pathfinder Sojourn on 4 July 1997.

- Stopped working after 87 sol due to software error.
- Diagnosed as a rare schedulability problem called Priority Inversion.
- Fixed by reloading patched code.

# Mars Pathfinder Bug

NASA sent mars pathfinder Sojourn on 4 July 1997.

- Stopped working after 87 sol due to software error.
- Diagnosed as a rare schedulability problem called Priority Inversion.
- Fixed by reloading patched code.

## Schedulability Analysis

Subsequent schedulability analysis found the problem in original design. It proved the correctness of modified design.

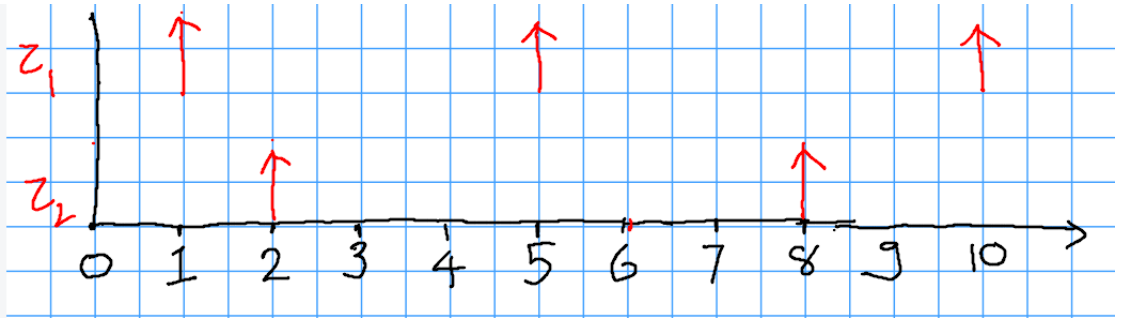
# IEEE TCRTS Test Of Time Awards 2020

Instituted by IEEE in 2020 for papers having lasting impact on the field.

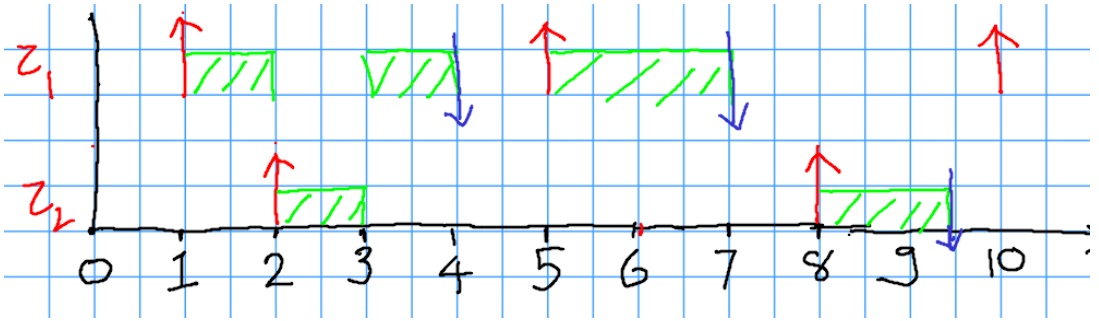
- Chung. L. Liu and James W. Layland
  **Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment**
  Journal of the Association for Computing Machinery, Vol. 20, No. 1, pp. 46-61, January 1973.
  *For pioneering the way towards a formal analysis of real-time scheduling algorithms.*
- Mathai Joseph and Paritosh Pandya
  **Finding Response Times in a Real-Time System**
  The Computer Journal, Vol. 29, Issue 5, pp. 390–395, 1986.
  *For first proposing an exact method for computing worst-case response times under fixed priority pre-emptive scheduling.*
- John A. Stankovic
  **Misconceptions about Real Time Computing: A Serious Problem for Next Generation Systems**
  IEEE Computer, Vol. 21, pp. 10-19, October 1988.
  *For clearly highlighting the unique characteristics of real-time computing and motivating research in this field.*
- Lui Sha, Raj Rajkumar and John P. Lehoczky
  **Priority Inheritance Protocols: an Approach to Real-Time Synchronization**
  IEEE Transactions on Computers, Vol. 39, No. 9, pp. 1175-1185, September 1990.
  *For introducing the now-standard methods for controlling priority inversion on uniprocessors and their impact on the Mars Pathfinder mission.*

# Framework for Schedulability

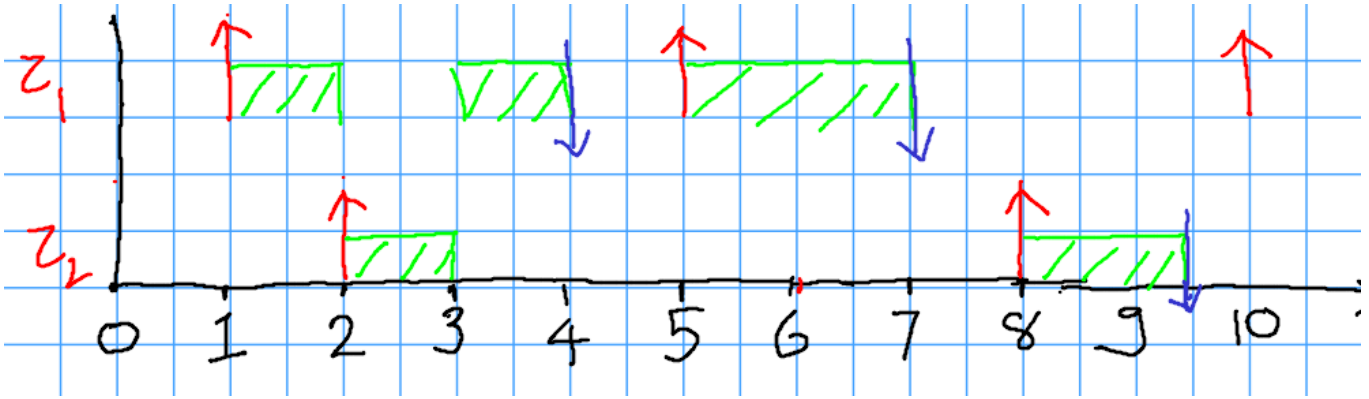
- A set of repeating tasks $\tau_1, \ldots, \tau_n$
- Arrival Pattern $\sigma = (\Gamma_i, \Theta_i)$




- $\Gamma_i(j)$ gives time of arrival of $j$th instance of task $i$.
  $\Theta_i(j)$ gives cpu time needed to execute $j$th instance of task $i$.
- Tasks are executed on Single CPU under the control of a scheduler.



# Framework for Schedulability (2)




$pr(\tau_2) > pr(\tau_1)$

- Preemptive scheduling versus Non-preemptive scheduling.
- Priority Based Preemptive Scheduling Each process $\tau_i$ has a unique priorty $pr(\tau_i)$. Processes can be ordered by their priority.
- For a taskset $\tau_1, \ldots, \tau_n$, arrival pattern blue $\sigma = (\Gamma_i, \Theta_i)$ and priority assignment $pr(\tau_i)$ there is unique execution diagram.
- Response time (local) $RTL_i(j)$ is the time between release and completion of $j$th instance of task $i$.
  Example: $RTL_1(1) = 3$, $RTL_2(1) = 1$, and $RTL_2(1) = 2$.
- Deadline $D_i$ maximum permitted response time.
  Execution meets deadline $D_i$ if $\forall i, j.\ \ RTL_i(j) \leq D_i$.

# Sporadic Tasks

A set of sporadic tasks $\tau_1, \ldots, \tau_n$.

$\tau_i = (T_i, C_i, D_i)$ with $D_i \leq T_i$.

- (Period $T_i$) Each task $\tau_i$ is repeatedly invoked with a minimum period of $T_i$.
  $\Gamma_i(j+1) - \Gamma_i(j) \geq T_i$ for all $i, j$.
- (Load $C_i$) Each invocation needs at most $C_i$ seconds processor time.
  $\Theta_i(j) \leq C_i$ for all $i, j$.
- (Deadline $D_i$) Each invocation must finish within $D_i$ seconds of its arrival.

**Worst Case Response time $RT_i$ under priority assignment $pr$**

Let $\Sigma$ be set of arrival patterns satisfying sporadic constraints.

$$RT_i = \max_{\sigma \in \Sigma} \max_j RTL_i(j)$$

Thus worst case response time $RT_i$ is the maximum of $RTL_i(j)$ over all instances and all permitted arrival patterns $\Sigma$.

Priority assignment $pr$ is feasible if $RT_i \leq D_i$ for all $i$

# Hard Real Time Systems [Liu and Layland 1973]

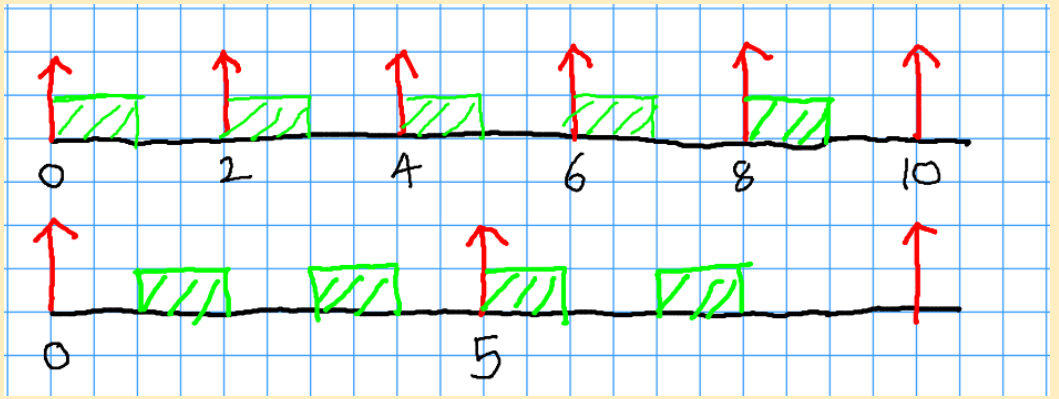
- A set of sporadic tasks $\tau_1, \ldots, \tau_n$ with $\tau_i = (T_i, C_i, D_i)$
- (Period $T_i$) Each task $\tau_i$ is repeatedly invoked at a minimum period of $T_i$.
  (Load $C_i$) Each invocation needs atmost $C_i$ seconds processor time.
- (Deadline $D_i$) Each invocation must finish within $D_i$ seconds of its arrival.
- Tasks are independant (no synchronization). ←
- Tasks execute on a single processor. CPU is shared between tasks.
- Priority based pre-emptive scheduling:
  Tasks are assigned unique priorities.
  Invocation of higher priority task switches processor to it from currently executing lower priority task.

A priorty assignment is feasible if for all possible task arrival patterns all deadlines are met. Taskset is feasible if there exists a feasible priority assignment.

# Hard Real-time System Example

Task set $\tau_1 = (2, 1, 2)$ and $\tau_2 = (5, 2, 4)$

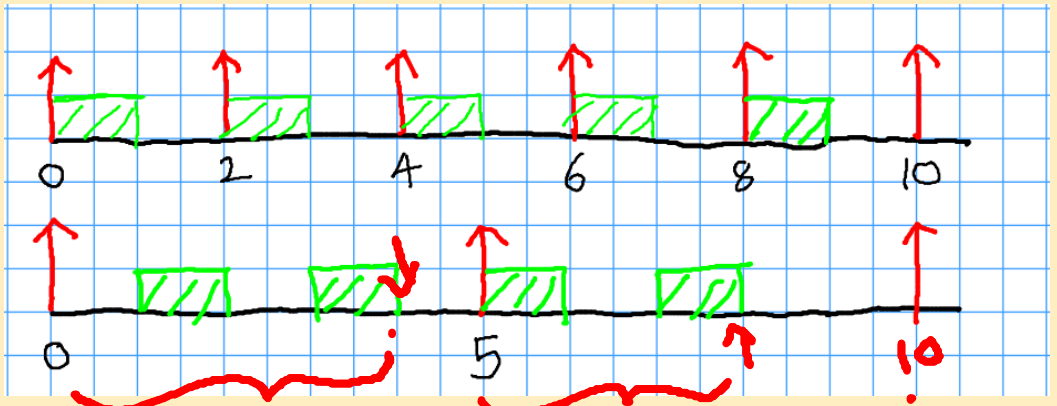
## Execution with $pr(\tau_1) > pr(\tau_2)$



# Hard Real-time System Example

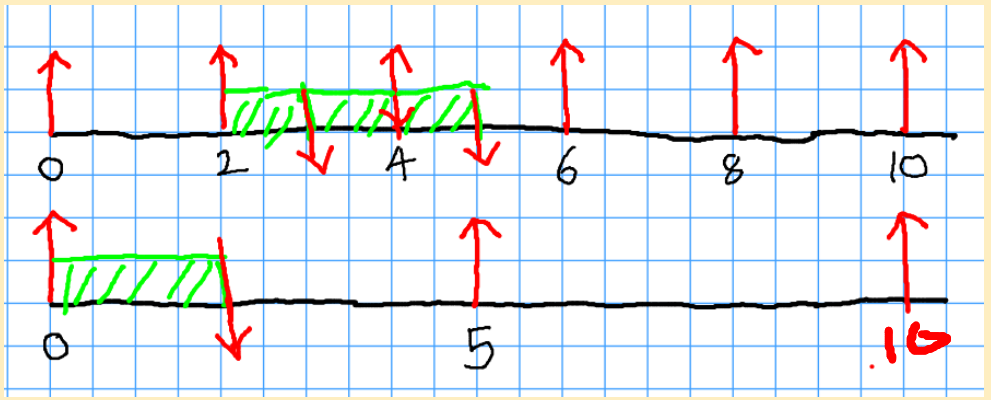
Task set $\tau_1 = (2, 1, 2)$ and $\tau_2 = (5, 2, 4)$

## Execution with $pr(\tau_1) > pr(\tau_2)$




Feasible

## Execution with $pr(\tau_2) > pr(\tau_1)$




Infeasible

# Analysis of Hard Real Time Systems

[Liu and Layland 1973]
For a given sporadic task set

## Feasibility

Given a priority assignment *pr*, how to check feasibility (i.e. deadlines are always met under all permitted task arrival patterns)?

## Priority Assignment

How to assign priorities to the tasks to ensure feasibility? How to compute *pr* which is feasible?

# Critical Instance: Planning for the worst

Critical Arrival

Given a taskset $\tau_1, \ldots, \tau_n$ with $\tau_i = (T_i, C_i, D_i)$, the critical instance is the unique arrival pattern $\sigma = (\Gamma_i, \Theta_i)$ where

- All tasks are invoked simultaneously at time $= 0$. Thus, $\Gamma_i(1) = 0$
- All tasks always arrive exactly after period $T_i$. Thus, $\Gamma_i(j+1) - \Gamma_i(j) = T_i$ for all $i, j$.
- Each task invocation takes maximum permitted load $C_i$. Thus, $\Theta_i(j) = C_i$ for all $i, j$.

Under a given priority assignment, the critical instance has unique execution.

# Critical Instance: Planning for the worst

Critical Execution

Given a taskset $\tau_1, \ldots, \tau_n$ with $\tau_i = (T_i, C_i, D_i)$, the critical instance is the unique arrival pattern $\sigma = (\Gamma_i, \Theta_i)$ where

- All tasks are invoked simultaneously at time = 0. Thus, $\Gamma_i(1) = 0$
- All tasks always arrive exactly after period $T_i$. Thus, $\Gamma_i(j+1) - \Gamma_i(j) = T_i$ for all $i, j$.
- Each task invocation takes maximum permitted load $C_i$. Thus, $\Theta_i(j) = C_i$ for all $i, j$.

Under a given priority assignment, the critical instance has unique execution.

**Theorem (Liu, Layland 73)**

*If critical instance gives feasible execution, then the priority assignment is feasible.*

# Naive Feasibility Test

Given sporadic task set $\tau_1, \ldots, \tau_n$ with $\tau_i = (T_i, C_i, D_i)$, the hyper-period $HP = lcm(T_1, \ldots, T_n)$.

Given priority assignment $pr$ to check if it is feasible,

- Observation: Execution of the critical instance under $pr$ for the interval $[0 : HP)$ repeats without any change.
- Simulate the execution of critical instance only upto $HP$ and compute observed worst case reponse times for each task.
- If each of these $RT_i \leq D_i$ then priority assingment is feasible.

Difficulty Hyperperiod can grow exponentially with number of tasks and hence simulation is often not practicable.

# Static Priority Assignment Schemes (scheduling policies)

## Rate Monotonic Scheduling

Assign priorities in the order of rate (inverse of period). Shortest period gets highest priority.

## Theorem (Liu, Layland 73)

*For tasksets where $T_i = D_i$ for all $i$, rate monotonic scheduling is optimal. If any arbitrary priority assignment is feasible then so is rate monotonic assignment.*

## Deadline Monotonic Scheduling

Assign priorities in order of inverse of deadlines. Shortest deadline gets highest priority.

## Theorem (Leung, Whitehead, 1982)

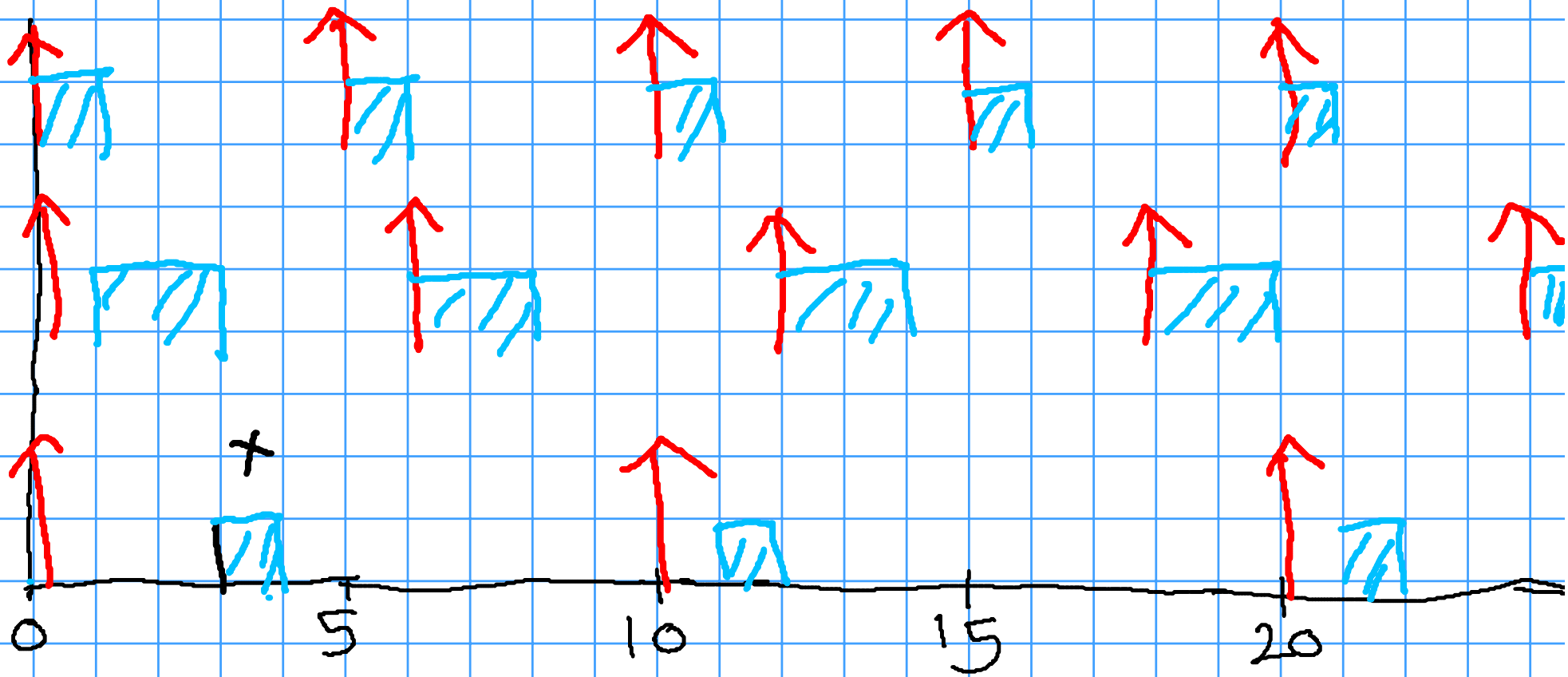
*For tasksets with $D_i \leq T_i$ forall $i$, Deadline monotonic scheduling is optimal. If any arbitrary fixed priority assignment is feasible (meets deadlines) then so is deadline monotonic priority assignment.*

# Example: Rate and Deadline Monotonic Priority Assignments

| *Task* | *T* | *C* | *D* |
|---|---|---|---|
| $\tau_1$ | 10 | 1 | 3 |
| $\tau_2$ | 5 | 1 | 5 |
| $\tau_3$ | 6 | 2 | 4 |

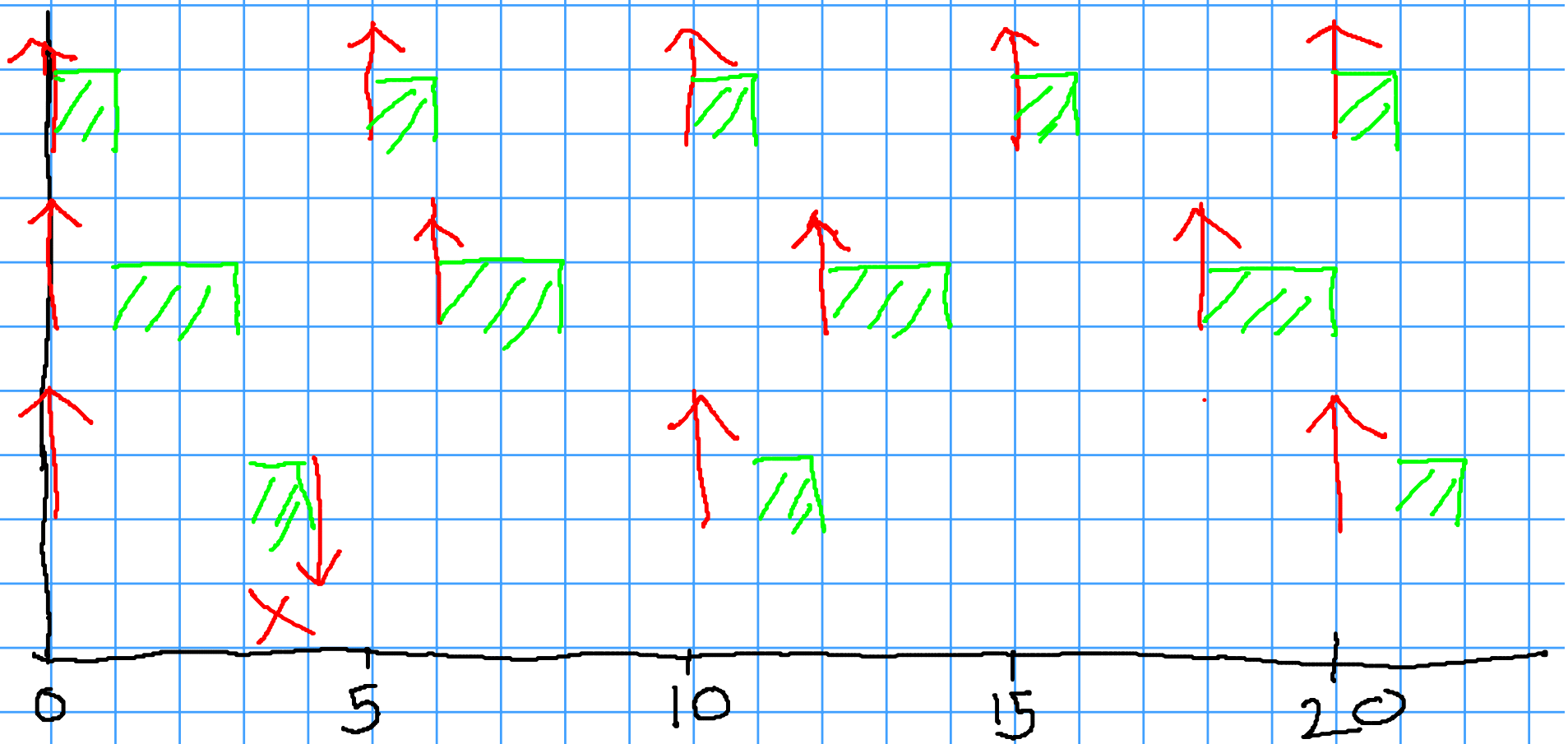
- Rate monotonic Priorities: $\tau_2 > \tau_3 > \tau_1$.
  Infeasible by naive test as it violates deadline.
- Deadline monotonic priorities: $\tau_1 > \tau_3 > \tau_2$
  Feasible by Naive test.

(5,1,5), (6,2,4), (10,1,3) in priority order



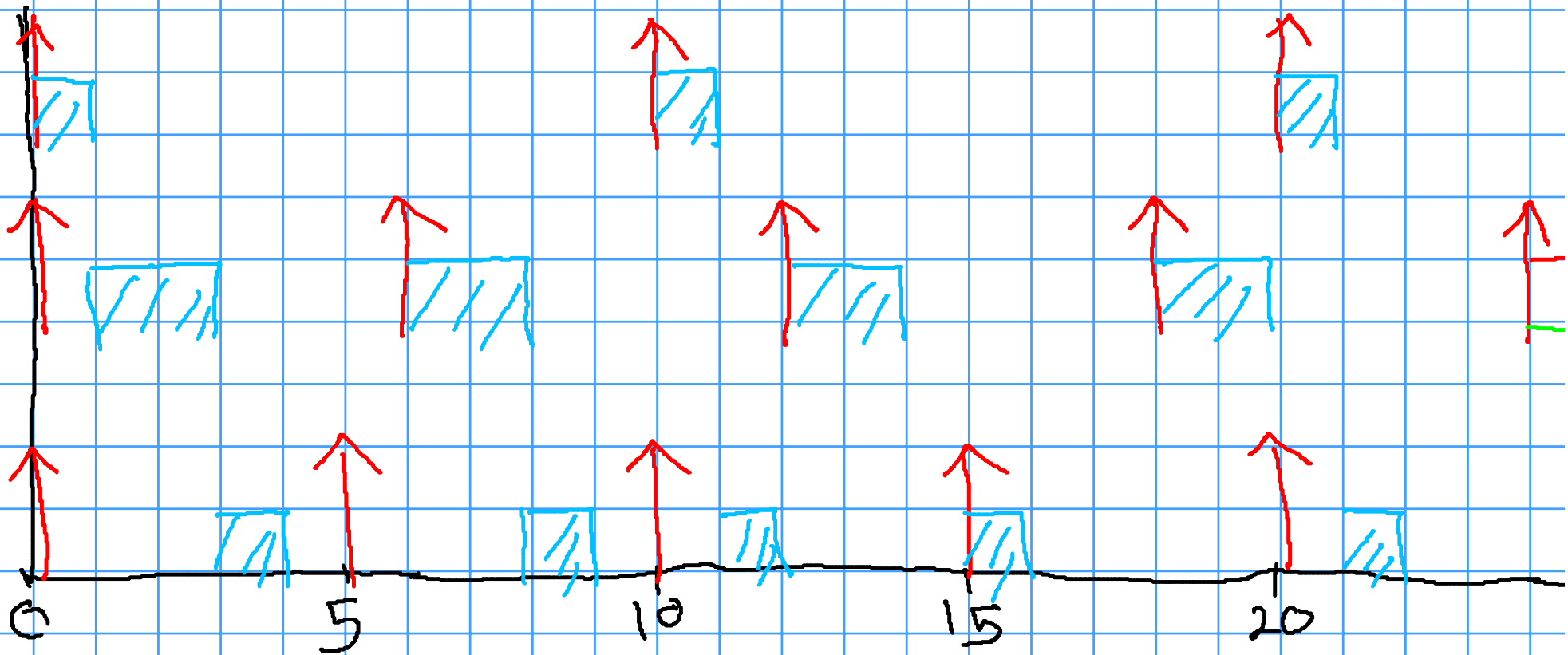

$RT(2,1) = 1$ $RT_2 = 3$ $RT_3 = 4$

(5, 1, 5) (6, 2, 1) (10, 1, 3) priority-ordered.




Infeasible

(10,1,3) (6,2,4) (5,1,5) DM priority order.




Feasible

# Fesibility Checking

Utilization based

[Liu and Layland 73]

$T_i$ 10 $C_2 = 2$ 0.2

## Utilization

CPU Utilization by task $i$ is $U_i = C_i/T_i$

Total Untilization $U = C_1/T_1 + C_2/T_2 + \ldots + C_n/T_n$

Total utilization gives the fraction of time the CPU is kept busy.

## Necessary condition

$$U \leq 1$$

Necessary but not sufficient.

## Sufficient Condition: For Tasksets with $D_i = T_i$ for all $i$.

$$U \leq B(n) \text{ where } B(n) = n \times (2^{1/n} - 1)$$

$B(n)$ has limit $\ln(2)$ as $n \to \infty$.

Sufficent but not necessary.

# Utilization Bound Table

| | | |
|---|---|---|
| B(1)=1.0 | B(4)=0.756 | B(7)=0.728 |
| B(2)=0.828 | B(5)=0.743 | B(8)=0.724 |
| B(3)=0.779 | B(6)=0.734 | $U(\infty)=0.693$ |

Note that $U(\infty)=0.693$ !

# Examples: Utilization based feasibility

$T_i = D_i$

- ($U > 1$) Infeasible: Taskset $(12, 8)$, $(6, 3)$. $U = 8/12 + 3/6$
- ($U < ln(2)$) Feasible: Taskset $(12, 2)$, $(6, 1)$. $U = 2/12 + 1/6 = 1/3$
- ($U = 1$) Inconclusive: Taskset $(12, 4), (6, 4)$. Consider naive test with rate monotonic assignment.
- Taskset $(100, 20)$, $(150, 40)$, $(350, 10)$.

# Examples: Utilization based feasibility

$T_i = D_i$

- $(U > 1)$ Infeasible: Taskset $(12, 8),\ (6, 3)$.
- $(U < ln(2))$ Feasible: Taskset $(12, 2),\ (6, 1)$.
- $(U = 1)$ Inconclusive: Taskset $(12, 4), (6, 4)$. Consider naive test with rate monotonic assignment.
- Taskset $(100, 20),\ (150, 40),\ (350, 10)$.
  $U = 20/100 + 40/150 = 0.753 > ln(2)$.
  But, $B(3) = 3 * (2^{1/3} - 1) = 0.779$. Hence, $U < B(3)$.
  Feasible by rate monotonic.

$D_2 < T_2$

# Exact test for feasibility: The Response Time Approach

[Joseph, Pandya 86]

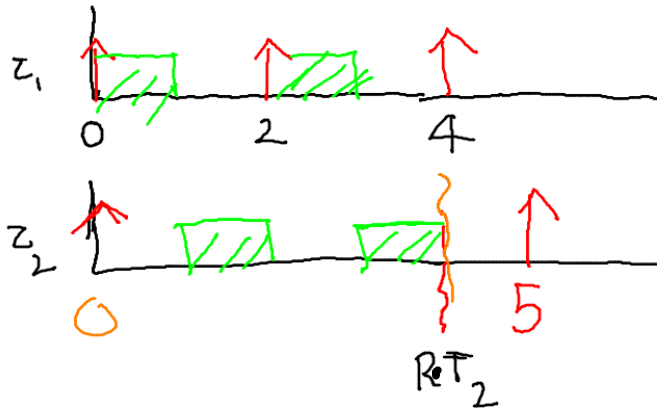
For a given priority assignment, let $RT_i$ denote the worst case response time of task $\tau_i$.

## Equational characterization of $RT_i$

Let $hp(i)$ denote the set of tasks with priority higher than $i$.

$$RT_i = C_i + \Sigma_{j \in hp(i)} (\lceil RT_i / T_j \rceil \times C_j) \quad (1)$$

Task set $\tau_1 = (2, 1, 2)$ and $\tau_2 = (5, 2, 4)$



# Exact test for feasibility: The Response Time Approach

[Joseph, Pandya 86]

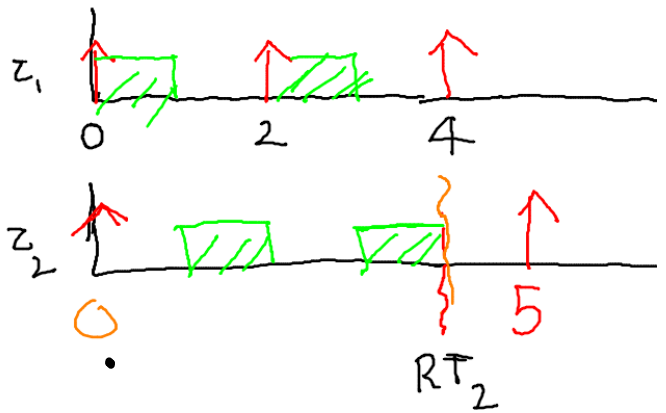
For a given priority assignment, let $RT_i$ denote the worst case response time of task $\tau_i$.

**Equational characterization of $RT_i$**

Let $hp(i)$ denote the set of tasks with priority higher than $i$.

$$RT_i = C_i + \Sigma_{j \in hp(i)} (\lceil RT_i / T_j \rceil \times C_j) \quad (1)$$

Task set $\tau_1 = (2, 1, 2)$ and
$\tau_2 = (5, 2, 4)$




$$RT_1 = C_1 + 0$$

# Exact test for feasibility: The Response Time Approach

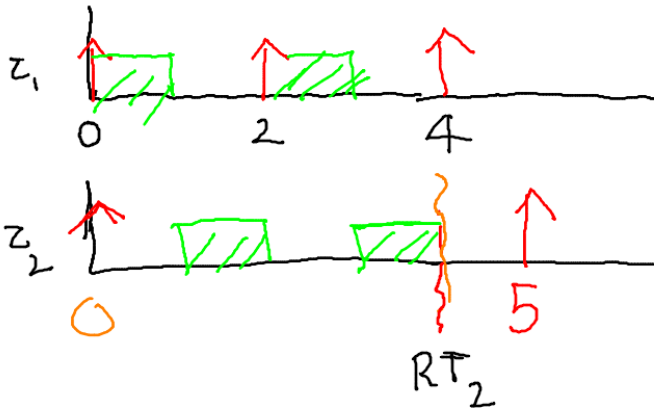
[Joseph, Pandya 86]
For a given priority assignment, let $RT_i$ denote the worst case response time of task $\tau_i$.

**Equational characterization of $RT_i$**

Let $hp(i)$ denote the set of tasks with priority higher than $i$.

$$RT_i = C_i + \Sigma_{j \in hp(i)} (\lceil RT_i / T_j \rceil \times C_j) \quad (1)$$

Task set $\tau_1 = (2, 1, 2)$ and $\tau_2 = (5, 2, 4)$




$$RT_2 = C_2 + I_1$$

$$I_1 = (\text{Count of } \tau_1 \text{ in } [0, RT_2)) * C_1$$
$$= (\lceil RT_2 / T_1 \rceil) * C_1$$

$$RT_2 = 2 + (\lceil RT_2 / 2 \rceil) * 1$$

Interference due to $\tau_j$ in $RT_i$ is

$$I_j^i = (\lceil RT_i / T_j \rceil) * C_j$$

# Solving the equation iteratively

$$r_i^1 = C_1 + C_2 + \ldots + C_i$$

$$r_i^{n+1} = C_i \; + \; \Sigma_{j \in hp(i)} \left( \lceil r_i^n / T_j \rceil \times C_j \right)$$

# Solving the equation iteratively

$$r_i^1 = C_1 + C_2 + \ldots + C_i$$

$$r_i^{n+1} = C_i \; + \; \Sigma_{j \in hp(i)} \left( \lceil r_i^n / T_j \rceil \times C_j \right)$$

- Start with $r_i^1$ as above. (An under-estimate of $RT_i$).
- Iteratively compute $r_i^{n+1}$ from $r_i^n$.

# Solving the equation iteratively

$$r_i^1 = C_1 + C_2 + \ldots + C_i$$

$$r_i^{n+1} = C_i \; + \; \Sigma_{j \in hp(i)} \left( \lceil r_i^n / T_j \rceil \times C_j \right)$$

- Start with $r_i^1$ as above. (An under-estimate of $RT_i$).
- Iteratively compute $r_i^{n+1}$ from $r_i^n$.
- (Convergence) Stop when $r_i^{n+1} = r_i^n$. This $r_i^n$ gives the worst case response time $RT_i$ for Task $\tau_i$.

# Solving the equation iteratively

$$r_i^1 = C_1 + C_2 + \ldots + C_i$$

$$r_i^{n+1} = C_i \; + \; \Sigma_{j \in hp(i)} \left( \lceil r_i^n / T_j \rceil \times C_j \right)$$

- Start with $r_i^1$ as above. (An under-estimate of $RT_i$).
- Iteratively compute $r_i^{n+1}$ from $r_i^n$.
- (Convergence) Stop when $r_i^{n+1} = r_i^n$. This $r_i^n$ gives the worst case response time $RT_i$ for Task $\tau_i$.
- If $U \leq 1$ then the computation will converge.

# Solving the equation iteratively

$$r_i^1 = C_1 + C_2 + \ldots + C_i$$

$$r_i^{n+1} = C_i \; + \; \Sigma_{j \in hp(i)} \left( \lceil r_i^n / T_j \rceil \times C_j \right)$$

- Start with $r_i^1$ as above. (An under-estimate of $RT_i$).
- Iteratively compute $r_i^{n+1}$ from $r_i^n$.
- (Convergence) Stop when $r_i^{n+1} = r_i^n$. This $r_i^n$ gives the worst case response time $RT_i$ for Task $\tau_i$.
- If $U \leq 1$ then the computation will converge.
- Fail if for any $n$, we get $r_i^n > D_i$. Priority assignment is infeasible.

# Example: Exact feasibility Test

Taskset $(10, 1, 3),\ (6, 2, 4),\ (5, 1, 5)$.

- $RT_1 = C_1 + 0 = 1$

# Example: Exact feasibility Test

Taskset $(10, 1, 3),\ (6, 2, 4),\ (5, 1, 5)$.

- $RT_1 = C_1 + 0 = 1$
- $RT_2^1 = C_1 + C_2 = 1 + 2 = 3.$
  $RT_2^2 = C_2 + \lceil RT_2^1 / T_1 \rceil * C_1 = 2 + \lceil 3/10 \rceil * 1 = 2 + 1 * 1 = 3$

# Example: Exact feasibility Test

Taskset $(10, 1, 3),\ (6, 2, 4),\ (5, 1, 5)$.

- $RT_1 = C_1 + 0 = 1$
- $RT_2^1 = C_1 + C_2 = 1 + 2 = 3.$
  $RT_2^2 = C_2 + \lceil RT_2^1 / T_1 \rceil * C_1 = 2 + \lceil 3/10 \rceil * 1 = 2 + 1 * 1 = 3$
- $RT_3^1 = C_1 + C_2 + C_3 = 1 + 2 + 1 = 4$
  $RT_3^2 = C_3 + (\lceil RT_3^1 / T_1 \rceil * C_1) + (\lceil RT_3^1 / T_2 \rceil * C_2)$
  $= 1 + (\lceil 4/10 \rceil * 1) + (\lceil 4/6 \rceil * 2)$
  $= 1 + (1 * 1) + (1 * 2) = 4$

# Complexity

- Feasibility is in NP.
- Feasibility has pseudo-polynomial upper bound.
  - Each iteration takes constant amount of time.
  - Number of iterations is bounded by the value $\Sigma_i \ T_i$.

# Mars Pathfinder Bug

NASA sent mars pathfinder Sojourn on 4 July 1997.

- Stopped working after 87 sol due to software error.
- Diagnosed as a rare schedulability problem called Priority Inversion
- Fixed by reloading patched code.

# Example: Mars Pathfinder (4)

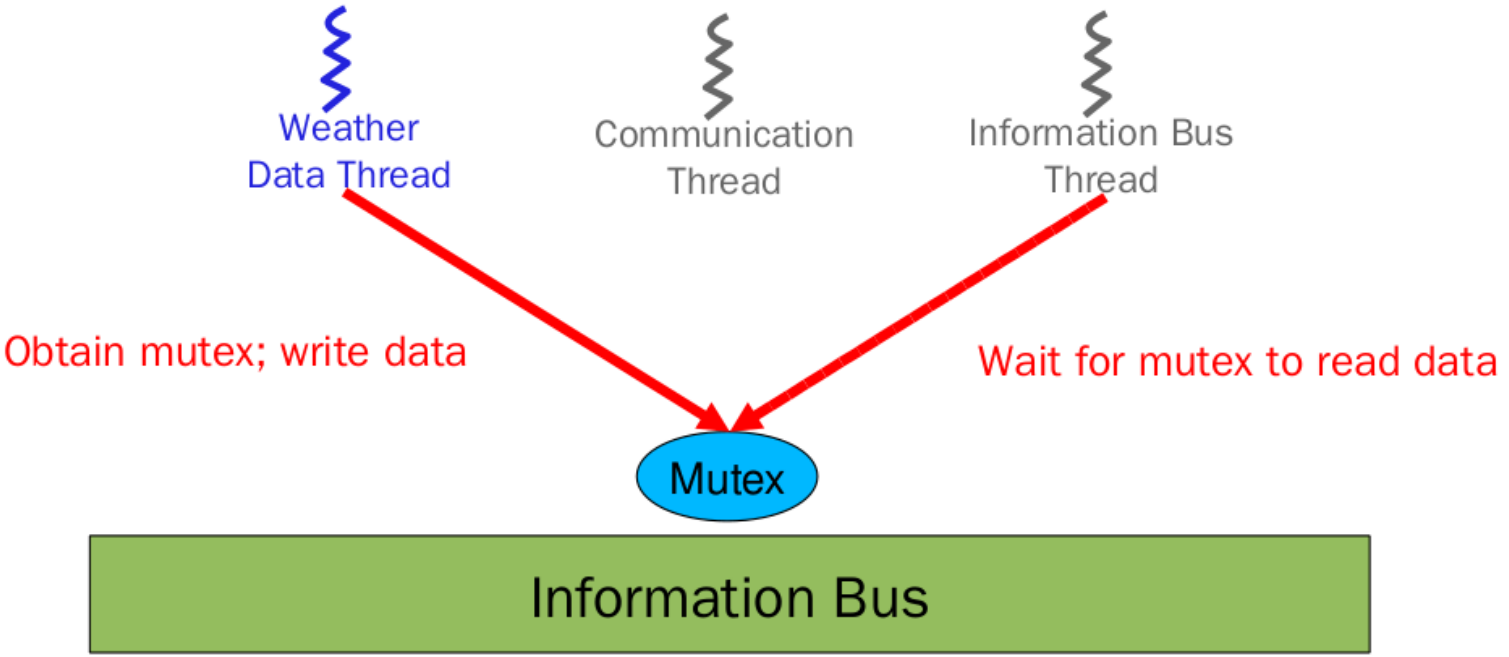
## Pathfinder used VxWorks RTOS

- Threads for the 1553 bus for data collection, scheduled on every 1/8th sec cycle.
- 3 periodic tasks
  - Task 1 – Information Bus Thread: Bus Manager
    high frequency, high priority
  - Task 2 – Communication Thread
    medium frequency / priority, high execution time
  - Task 3 – Weather Thread: Geological Data Gatherer
    low frequency, low priority
- Each checks if the other executed and completed in the previous cycle
  - If the check fails, this is a violation of a hard real-time guarantee and the system is reset

Information Bus: A shared data structure for devices and processes of Lander and Cruise Control.

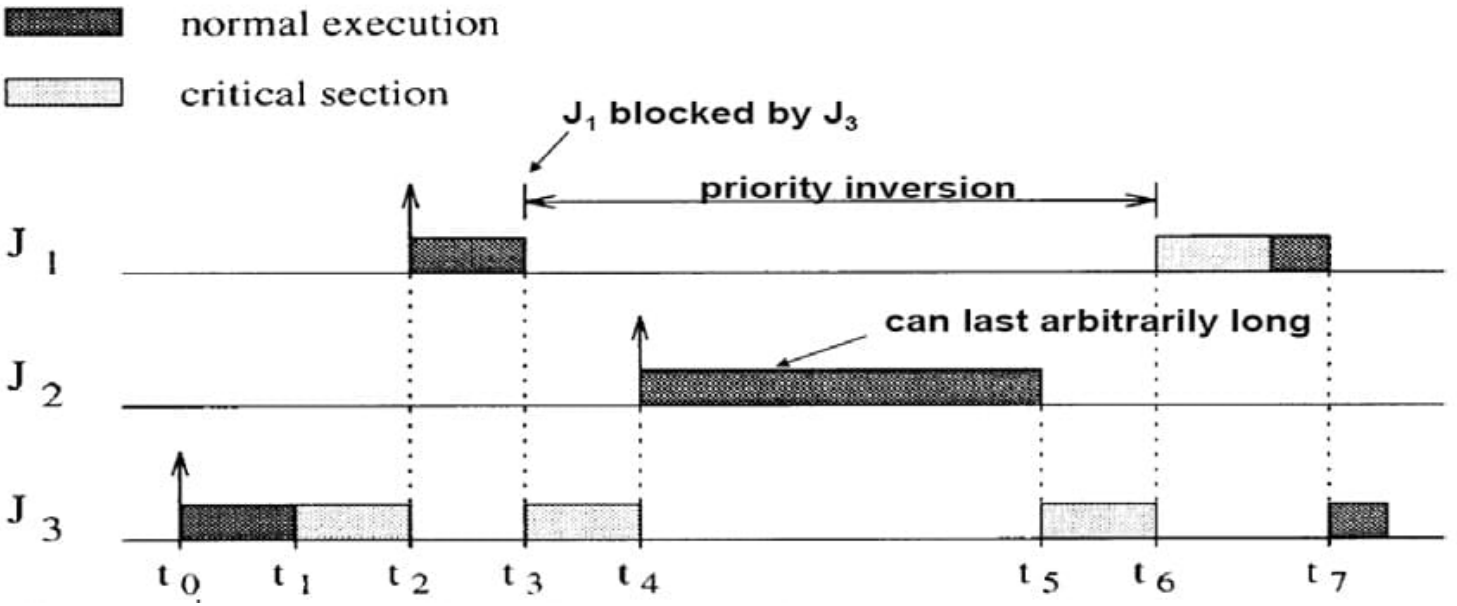
# Processes with Shared Resources

- Shared resources with mutually exclusive access
- Tasks block waiting for shared resource.



# Priority Inversion

[Lampson and Redell, 1980]



# Priority Inheritence Protocol (PIP)

- Temporarily increase the priority of task acquiring resource to high (ceiling) level.
- The critical section gets executed at high priority without blocking.
- Revert the task to original low priority on exiting the critical section.

Implemented in all major Kernels including POSIX threads, Java and VxWORKS.