

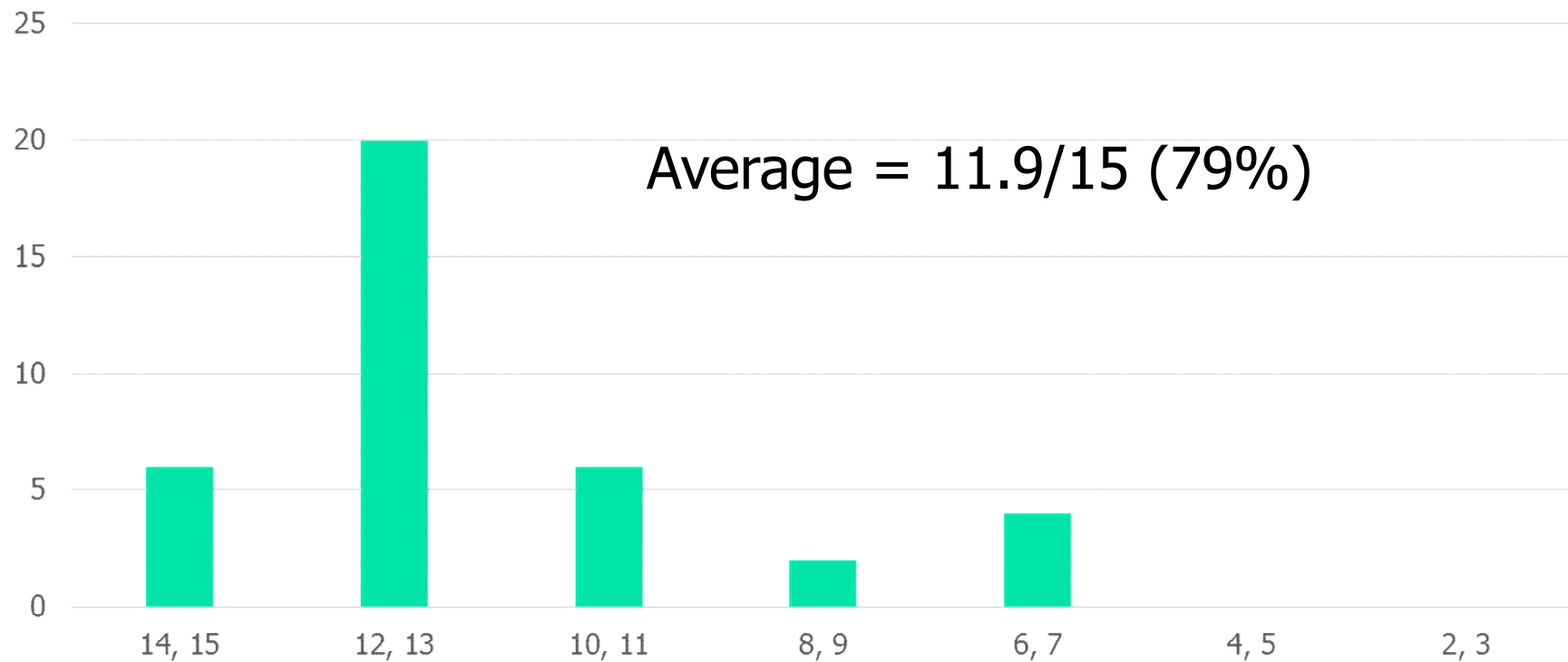


Final Review



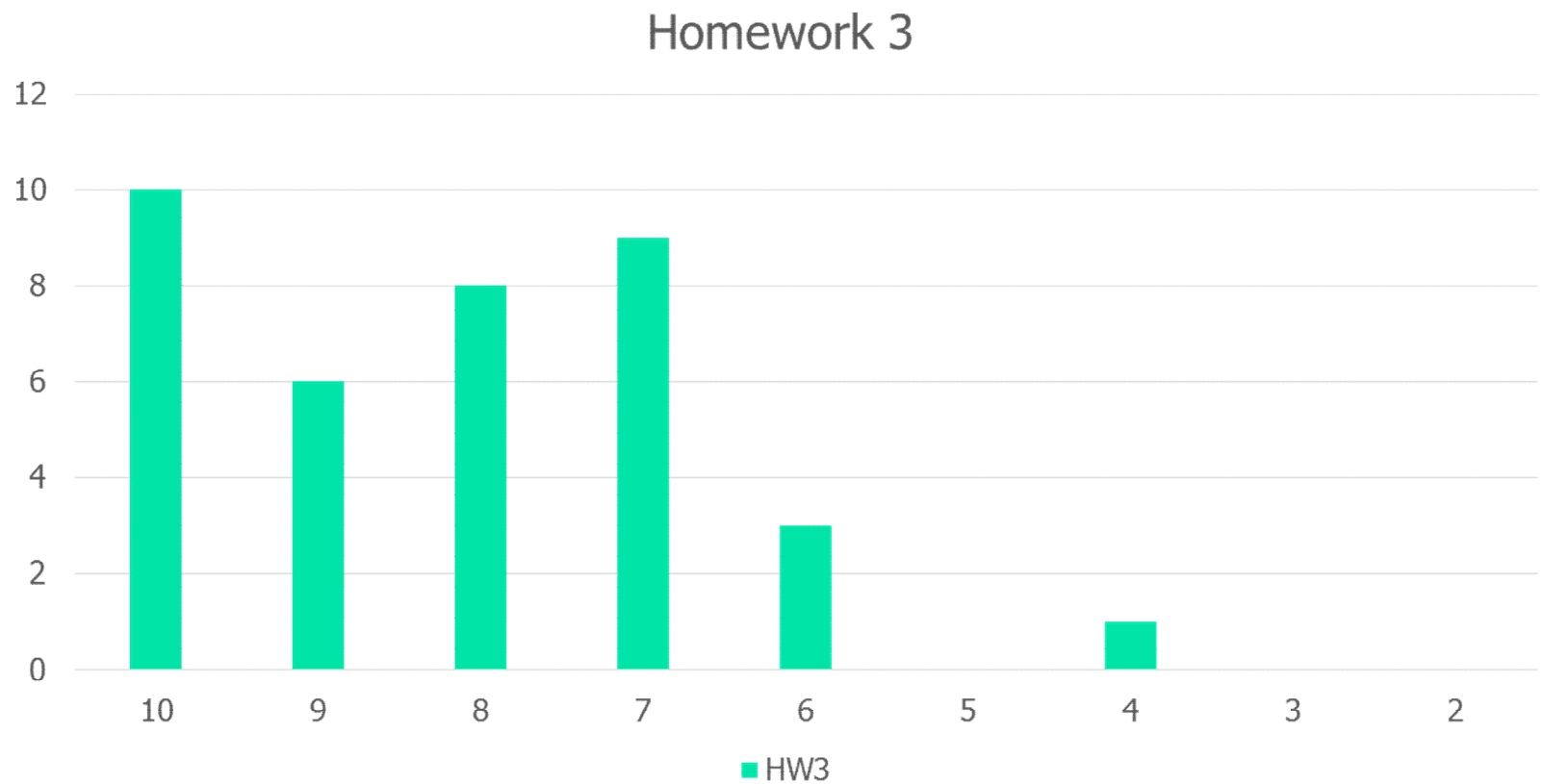
Midterm 2

Midterm

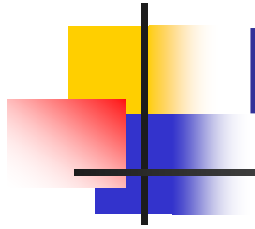




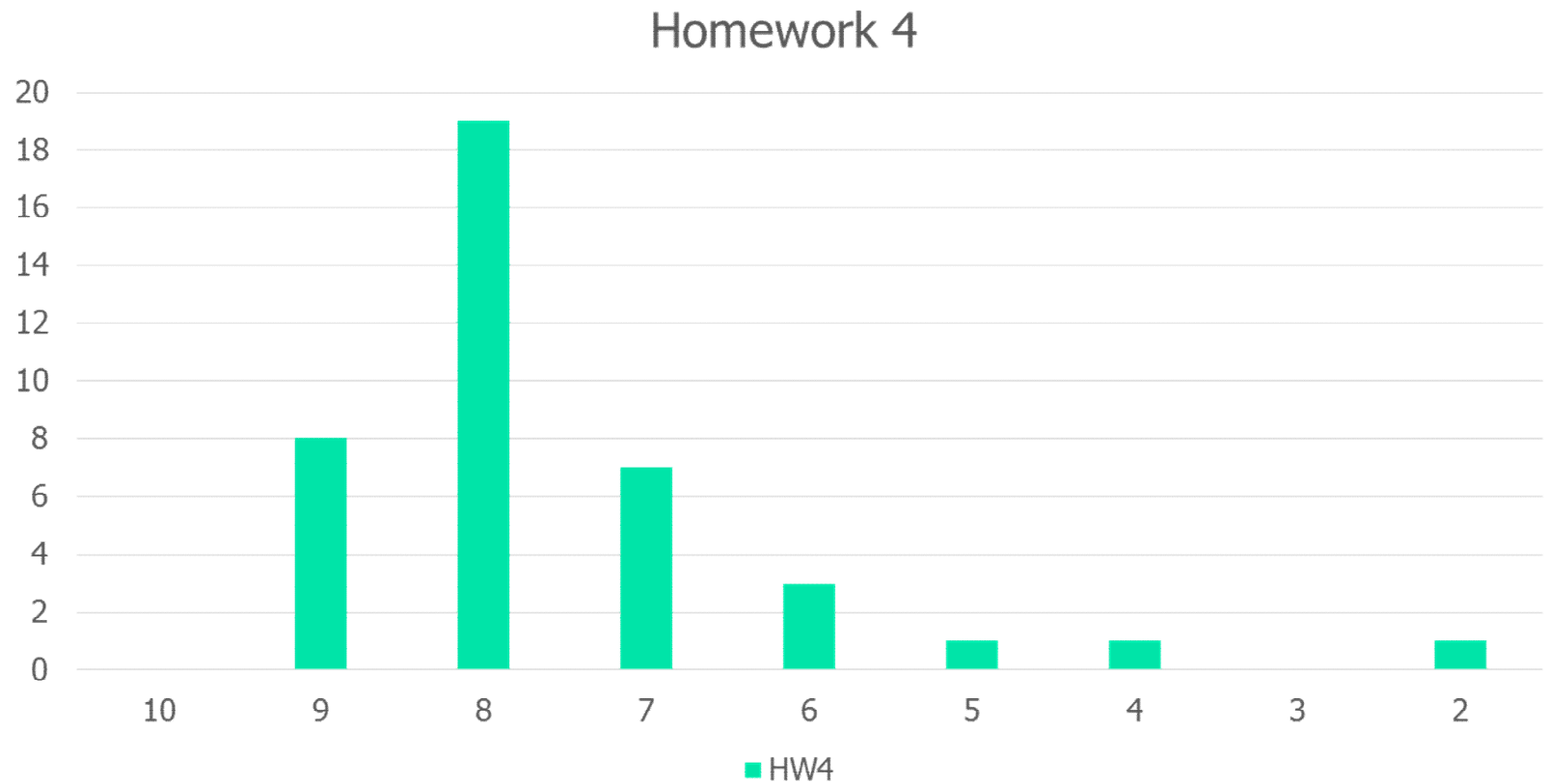
Homework 3



Average grade = 8.4/10



Homework 4



Average grade = 7.5/10



PART I

Reliability



Reliability

- Reliability for a giving mission duration t , $R(t)$, is the probability of the system working as specified (i.e., probability of no failures) for a duration that is at least as long as t .
- The most commonly used reliability function is the exponential reliability function:

$$R(t) = e^{-\lambda t}$$

where λ is the failure rate.

From queueing theory:
Probability of zero
independent arrivals in t
time units (Poisson
arrival process)



Reliability

- The most commonly used reliability function is the exponential reliability function:

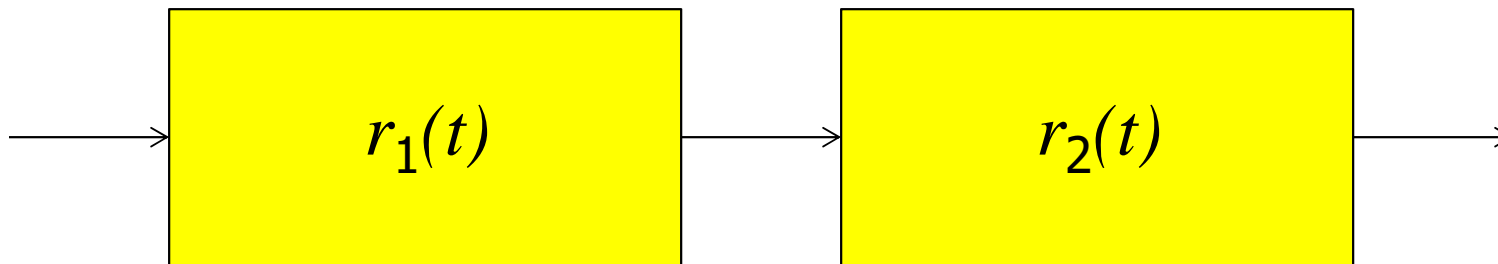
$$R(t) = e^{-\lambda t}$$

where λ is the failure rate.

- Mean time to failure (MTTF): $1 / \lambda$



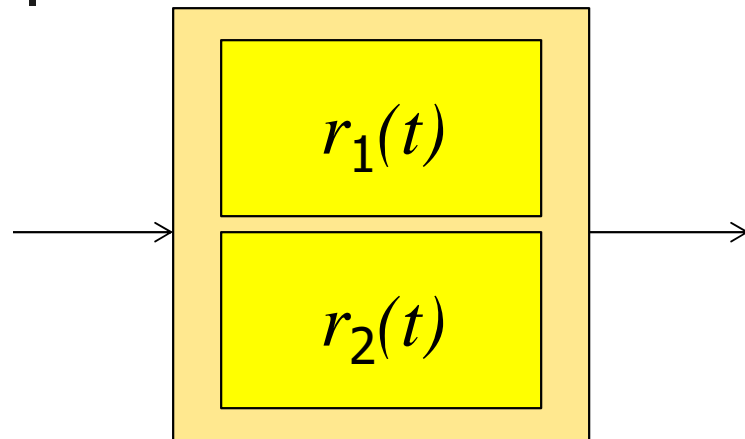
Simple Reliability Modeling



- Total failure rate = $\lambda_1 + \lambda_2$
- Mean time to failure = $1/(\lambda_1 + \lambda_2)$
- Total reliability:

$$R(t) = r_1(t)r_2(t) = e^{-(\lambda_1 + \lambda_2)t}$$

Simple Reliability Modeling

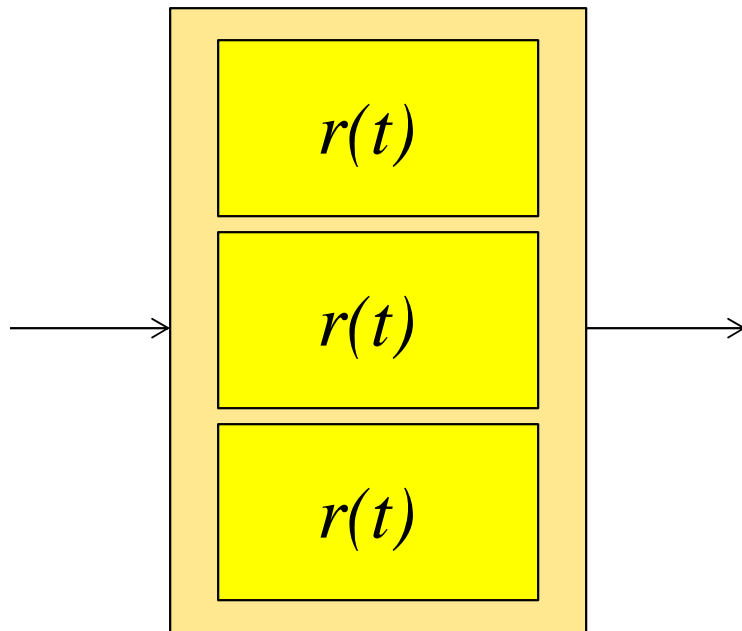


Note: This system needs at least one of the two components to function.

- Total reliability:

$$R(t) = 1 - (1 - r_1(t))(1 - r_2(t))$$

Triple Modular Redundancy



Note: This system needs at least two of the three components to function.

- Total reliability:

$$R(t) = r^3(t) + 3r^2(t)(1 - r(t))$$



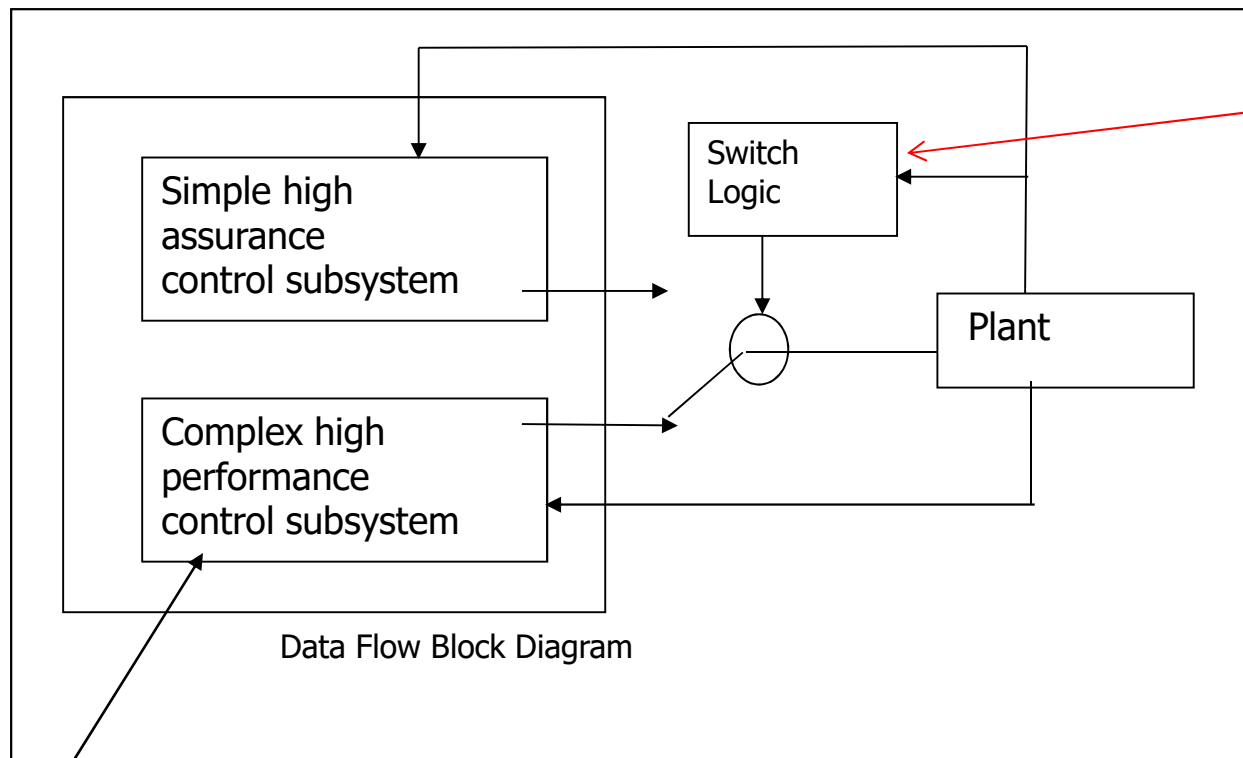
Other Implications

$$R(\textit{Effort}, \textit{Complexity}, t) = e^{-kC t/E}$$

- Note: splitting the effort greatly reduces reliability.

Simplex Architectural Pattern

A simple verifiable core; diversity in the form of 2 alternatives; feedback control of the software execution.



Is safety
guaranteed?
If not, switch

Better performance, but less reliable



Well Formed Dependencies

- *Informal intuition:* A reliable component should not *depend* on a less reliable component (it defeats the purpose).
- Design guideline: **Use but do not depend** on less reliable components



Review of Important Theorems

- Total Probability Theorem:

$$P(A) = P(A|C_1) P(C_1) + \dots + P(A|C_n) P(C_n)$$

where C_1, \dots, C_n cover the space of all possibilities

- Bayes Theorem:

$$P(A|B) = P(B|A) \cdot P(A)/P(B)$$

- Other: $P(A,B) = P(A|B) P(B)$

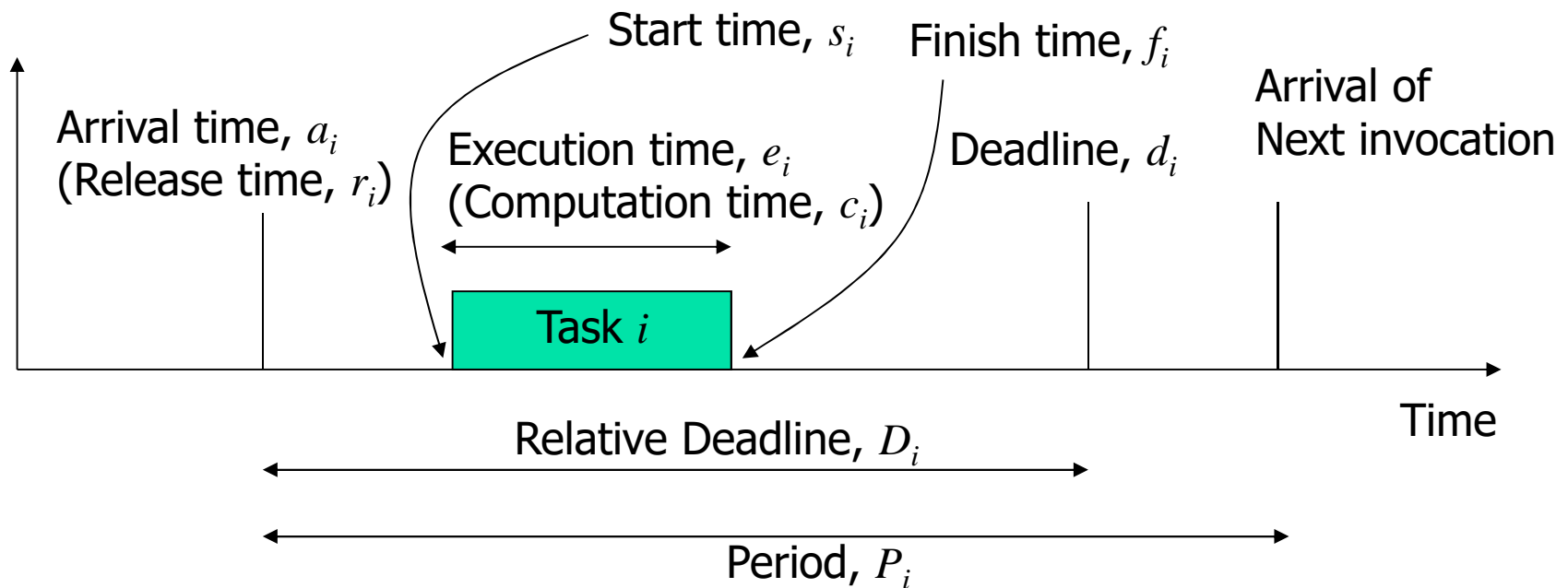


PART I

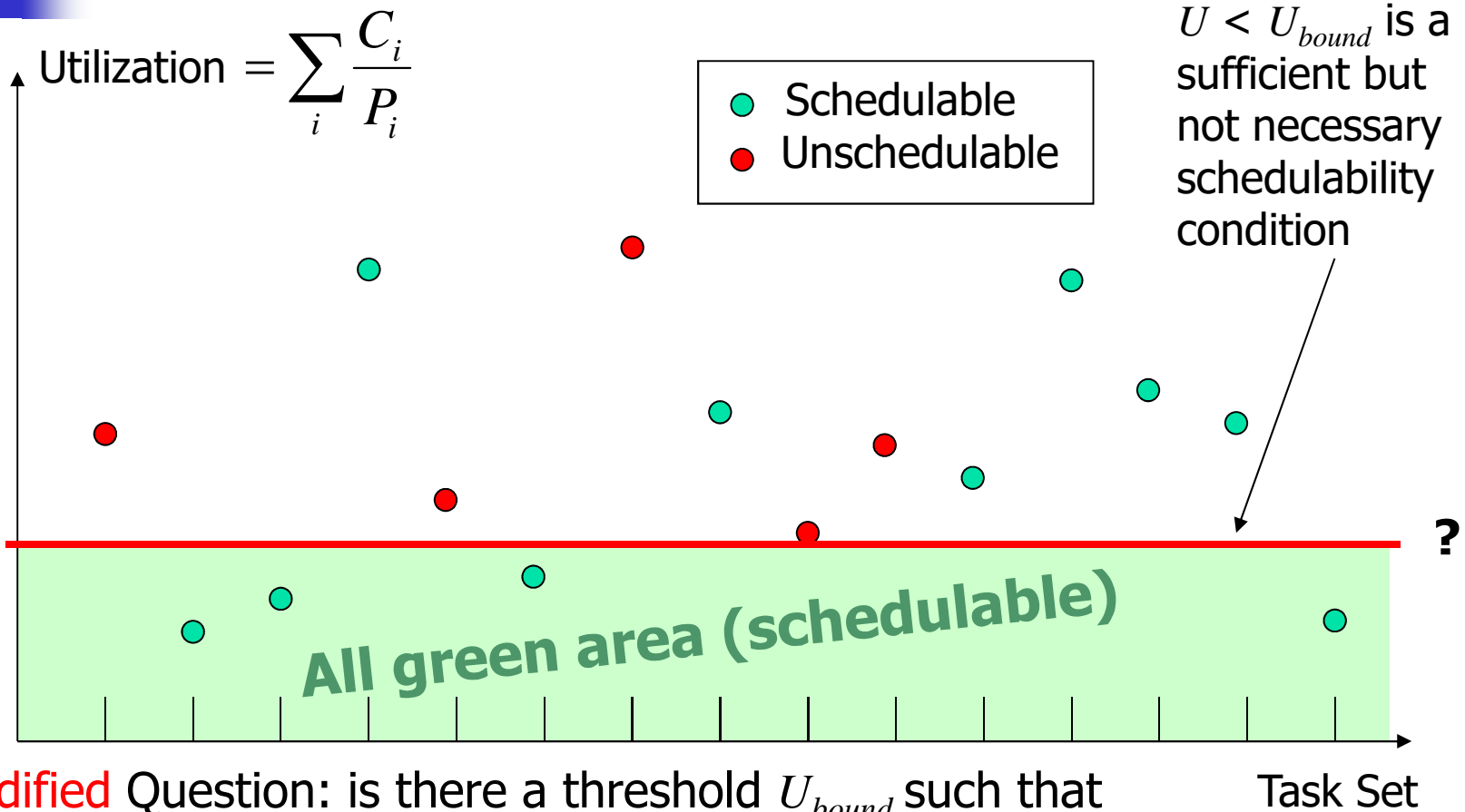
Timeliness

Some Terminology

- Tasks, periods, arrival-time, deadline, execution time, etc.



Utilization Bounds



- **Modified Question:** is there a threshold U_{bound} such that
 - When $U < U_{bound}$ deadlines are met
 - When $U > U_{bound}$ deadlines **may or may not be** missed



The Schedulability Condition

For n independent periodic tasks with periods equal to deadlines:

The utilization bound of EDF = 1.

The Utilization bound of RM is:

$$U = n \left(2^{1/n} - 1 \right)$$

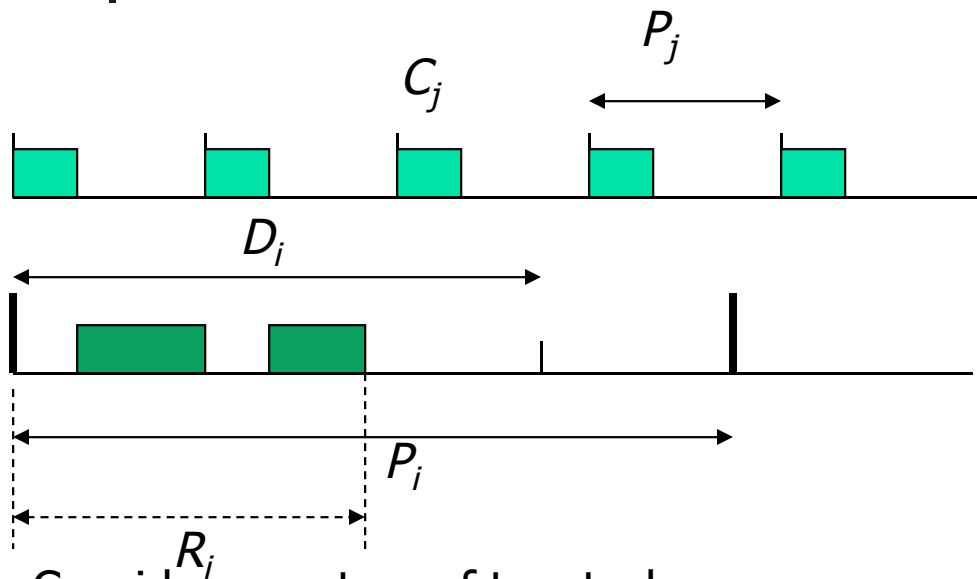
$$n \rightarrow \infty \quad U \rightarrow \ln 2$$

Example

$$I = \sum_{j \in HP} \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

$$R_i = I + C_i$$

I : Interference of higher priority tasks, HP with task i .

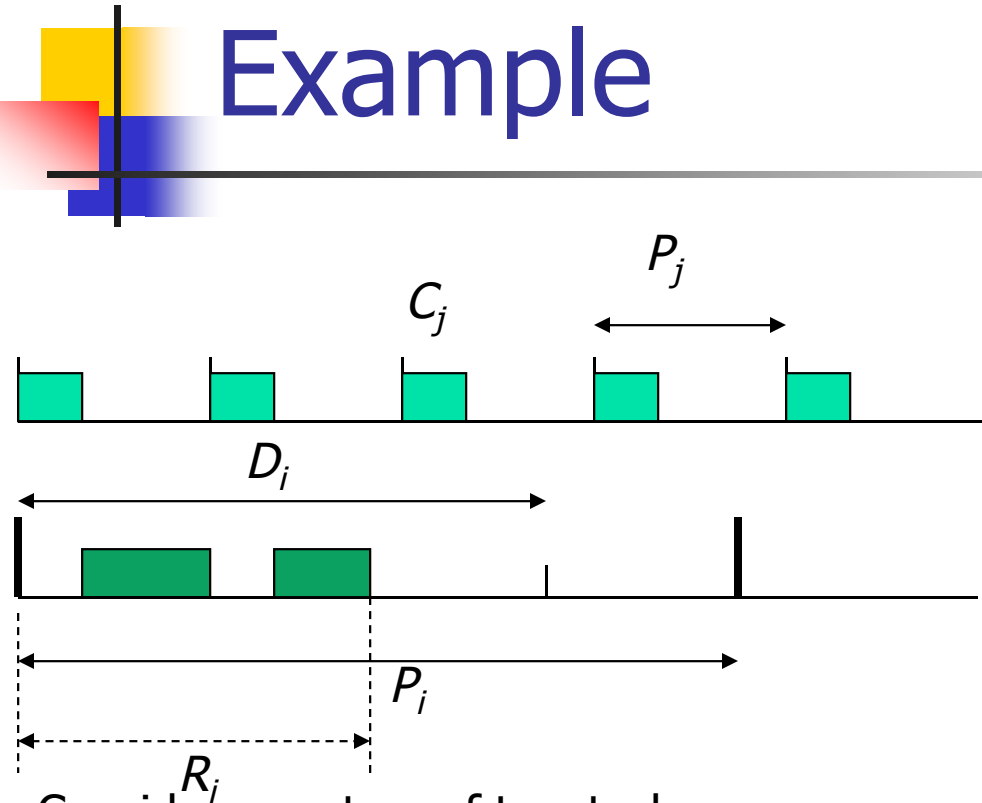


Consider a system of two tasks:

Task 1: $P_1=1.7$, $D_1=0.5$, $C_1=0.5$

Task 2: $P_2=8$, $D_2=3.2$, $C_2=2$

Example



Consider a system of two tasks:

Task 1: $P_1=1.7$, $D_1=0.5$, $C_1=0.5$

Task 2: $P_2=8$, $D_2=3.2$, $C_2=2$

$$I = \sum_{j \in HP} \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

$$R_i = I + C_i$$

I : Interference of higher priority tasks, HP with task i .

$$I^{(0)} = C_1 = 0.5$$

$$R_2^{(0)} = I^{(0)} + C_2 = 2.5$$

$$I^{(1)} = \left\lceil \frac{R_2^{(0)}}{P_1} \right\rceil C_1 = \left\lceil \frac{2.5}{1.7} \right\rceil 0.5 = 1$$

$$R_2^{(1)} = I^{(1)} + C_2 = 3$$

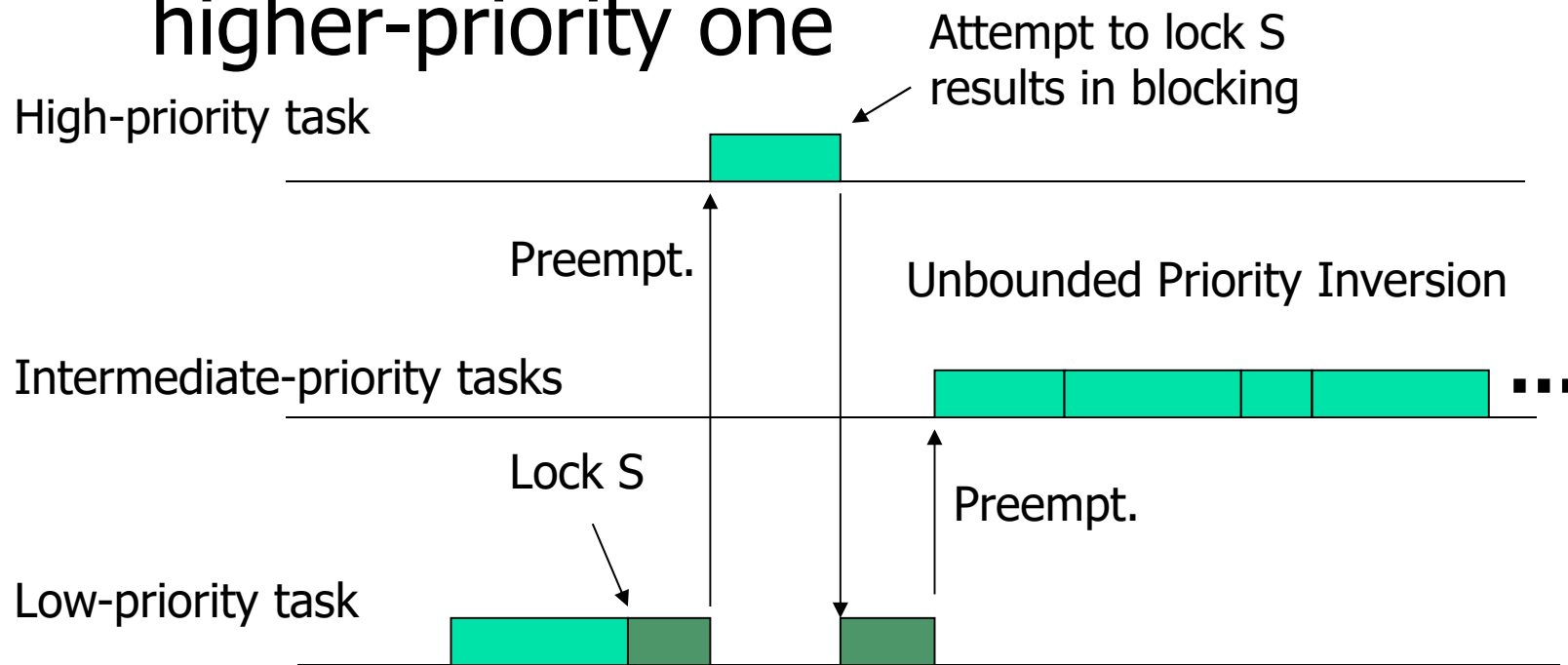
$$I^{(2)} = \left\lceil \frac{R_2^{(1)}}{P_1} \right\rceil C_1 = \left\lceil \frac{3}{1.7} \right\rceil 0.5 = 1$$

$$R_2^{(2)} = I^{(2)} + C_2 = 3$$

$3 < 3.2 \rightarrow \text{Ok!}$

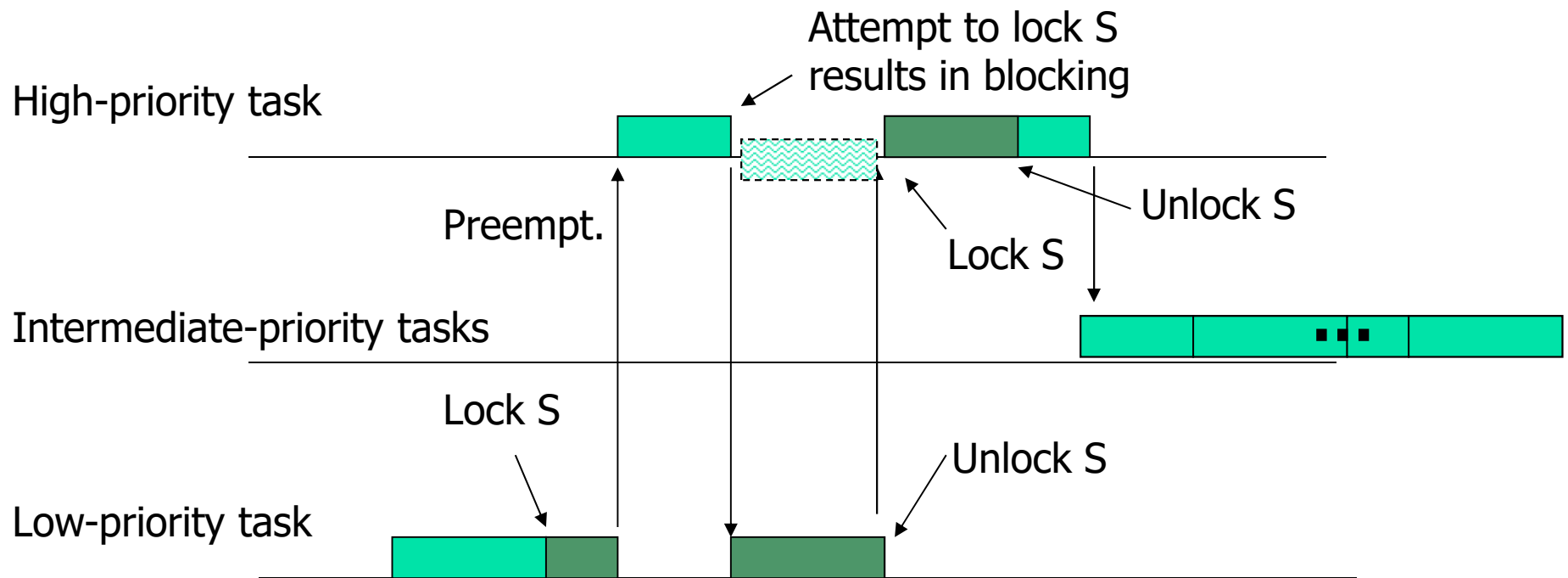
Blocking and Priority Inversion

- Consider the case below: a series of intermediate priority tasks is delaying a higher-priority one



Priority Inheritance Protocol

- Let a task inherit the priority of any higher-priority task it is blocking





Maximum Blocking Time

- If all critical sections are equal (of length B):
 - Blocking time = $B \min(N, M)$
(Why?)
- If they are not equal
 - Find the worst (maximum length) critical section for each resource
 - Add up the top $\min(N, M)$ sections in size
- The total priority inversion time for task i is called B_i



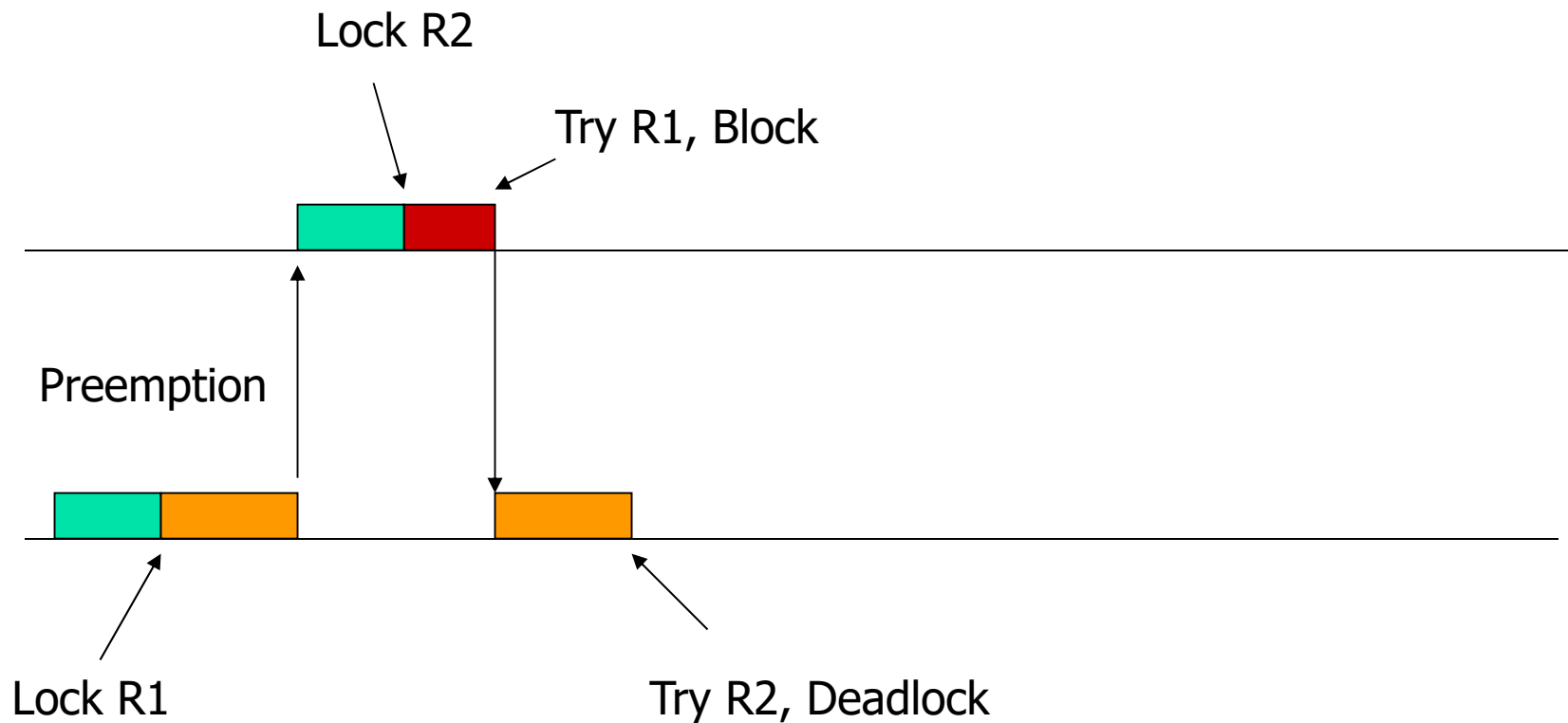
Schedulability Test

$$\forall i, 1 \leq i \leq n,$$

$$\frac{B_i}{P_i} + \sum_{k=1}^i \frac{C_k}{P_k} \leq i(2^{1/i} - 1)$$

Problem: Deadlock

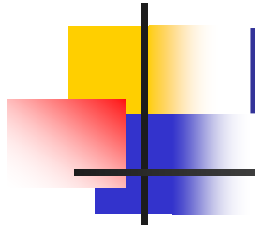
Deadlock occurs if two tasks locked two semaphores in opposite order





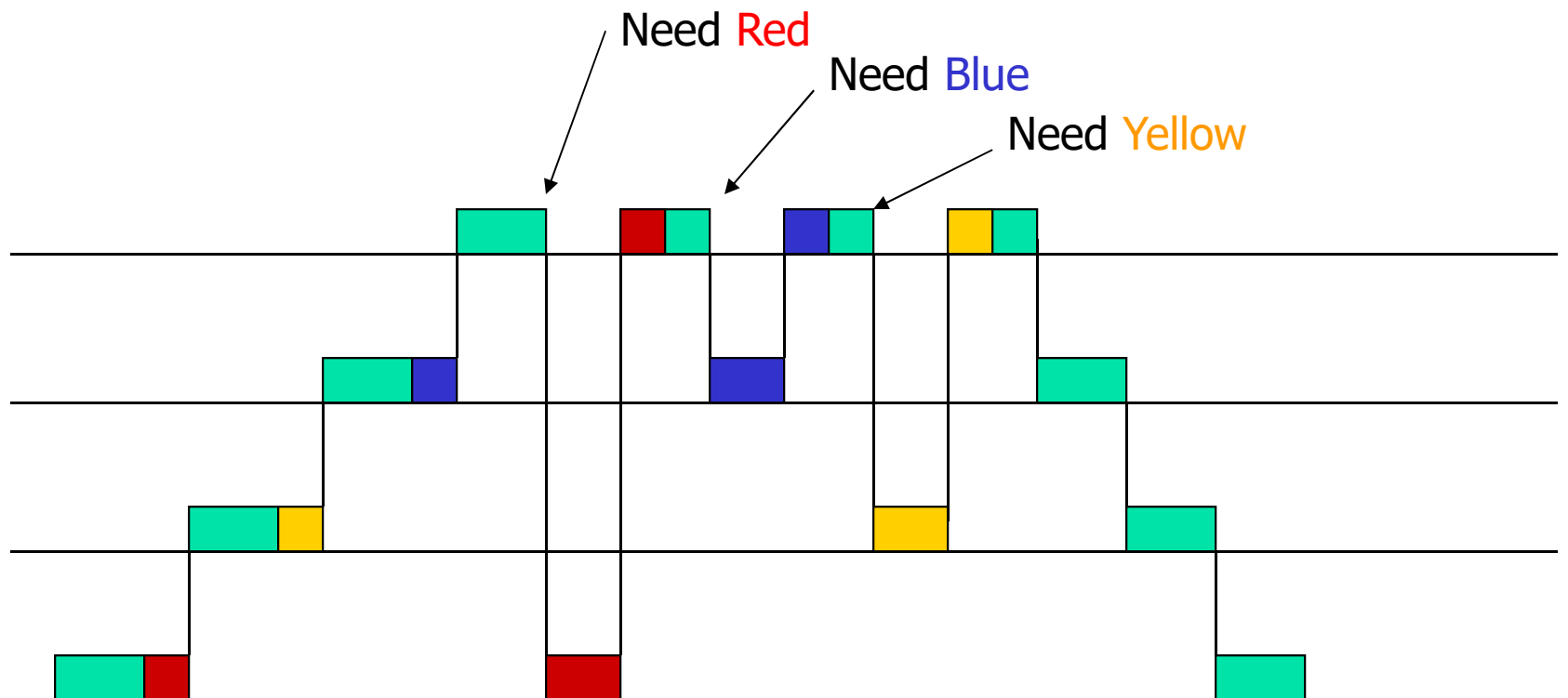
Priority Ceiling Protocol

- Definition: The priority ceiling of a semaphore is the highest priority of any task that can lock it
- A task that requests a lock R_k is denied if its priority is not higher than the highest priority ceiling of all currently locked semaphores (say it belongs to semaphore R_h)
 - The task is said to be blocked by the task holding lock R_h
- A task inherits the priority of the top higher-priority task it is blocking



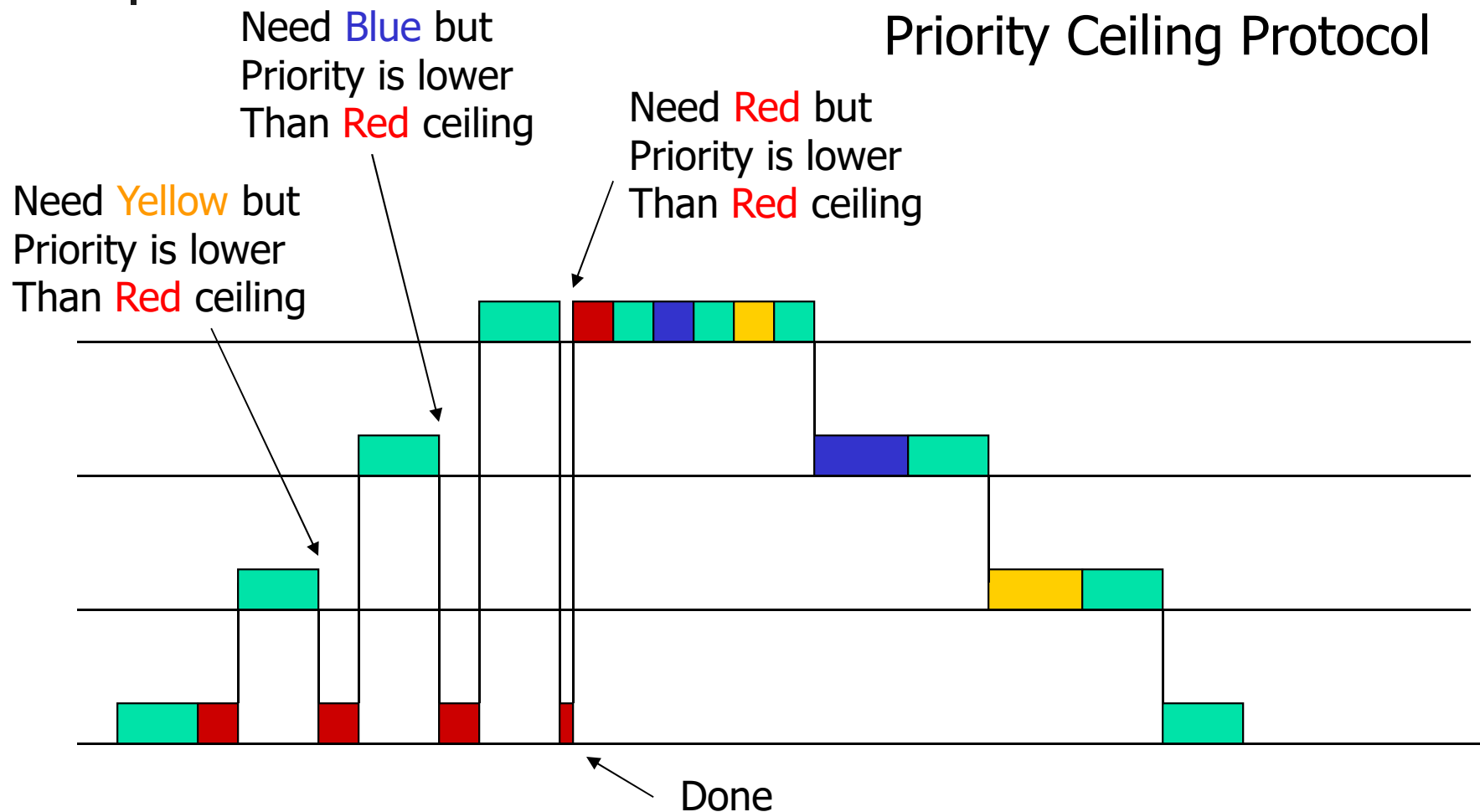
Maximum Blocking Time

Priority Inheritance Protocol



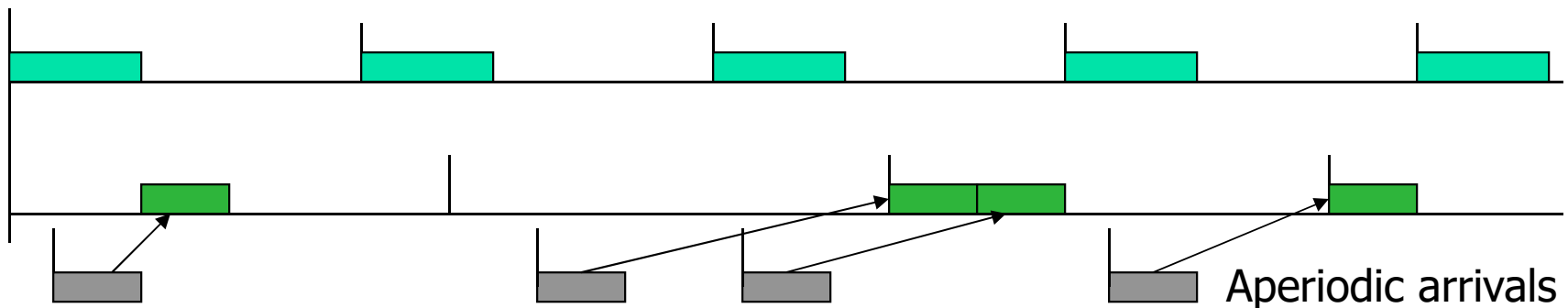
Maximum Blocking Time

Priority Ceiling Protocol



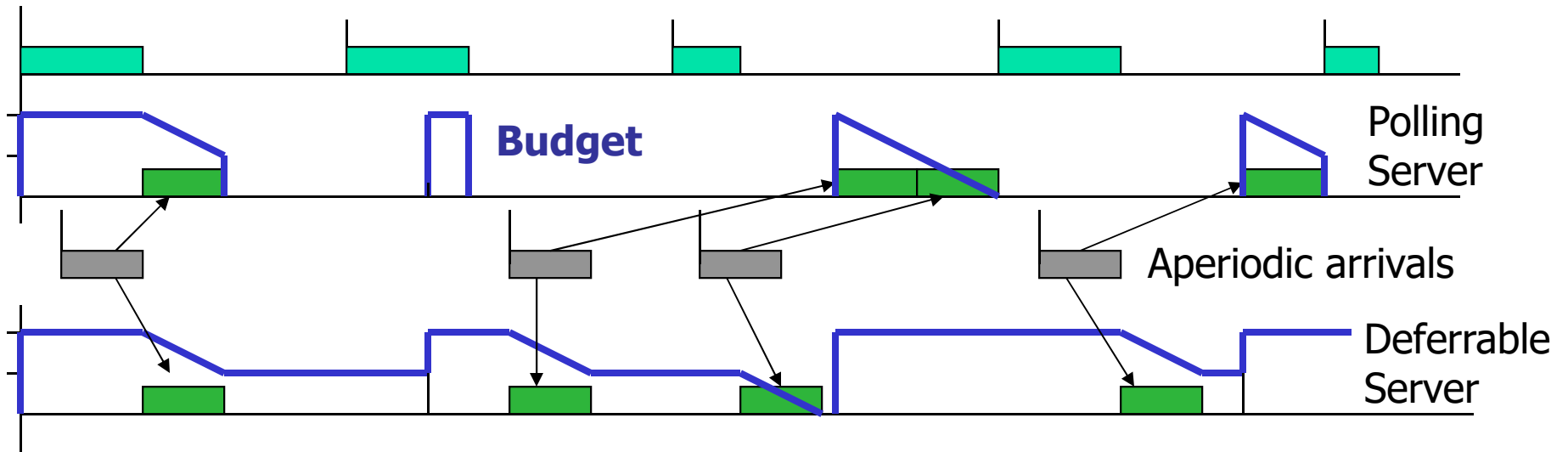
Example of a Polling Server

- Polling server:
 - Period $P_s = 5$
 - Budget $B_s = 2$
- Periodic task
 - $P = 4$
 - $C = 1.5$
- All aperiodic arrivals have $C=1$



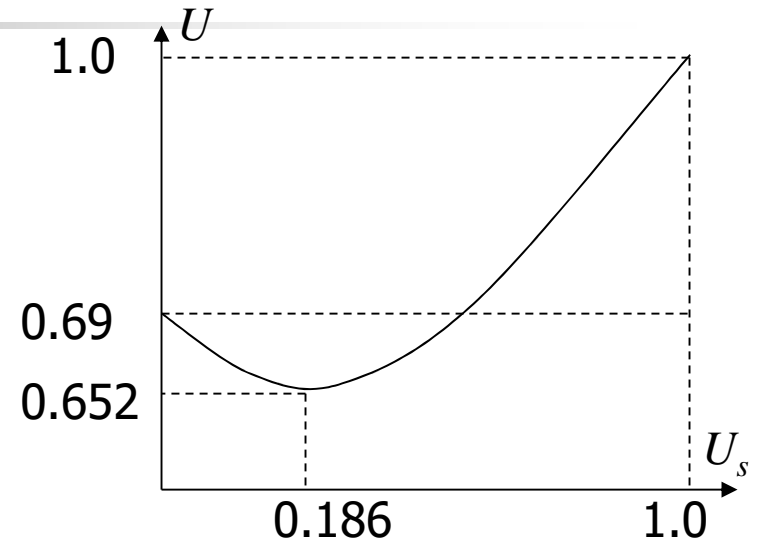
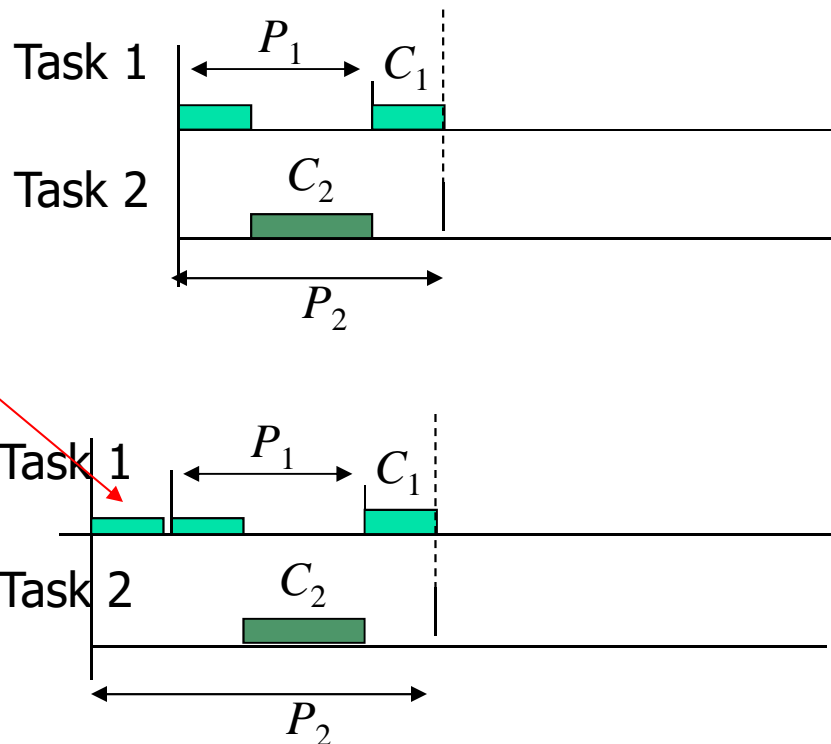
Deferrable Server

- Keeps the balance of the budget until the end of the period
- Example (continued)



Worst-Case Scenario

Deferred
Previous
Invocation



$$U_p \leq \ln \left(\frac{U_s + 2}{2U_s + 1} \right)$$

Exercise: Derive the utilization bound for a deferrable server plus one periodic task



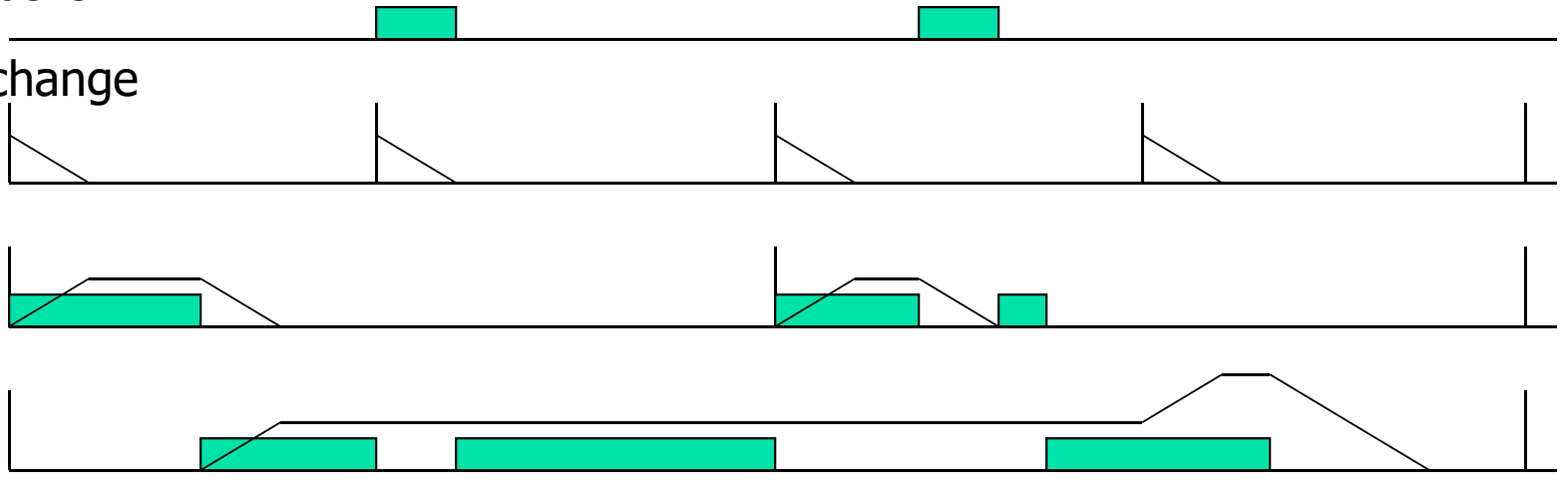
Priority Exchange Server

Example

Aperiodic tasks

Priority Exchange
Server

Periodic
Tasks





Sporadic Server

- Server is said to be *active* if it is in the *running* or *ready* queue, otherwise it is *idle*.
- When an aperiodic task comes and the budget is not zero, the server becomes active
- Every time the server becomes *active*, say at t_A , it sets replenishment time one period into the future, $t_A + P_s$ (but does not decide on replenishment amount).
- When the server becomes idle, say at t_I , set replenishment amount to capacity consumed in $[t_A, t_I]$

$$U_p \leq \ln \left(\frac{2}{U_s + 1} \right)$$



Slack Stealing Server

- Compute a slack function $A(t_s, t_f)$ that says how much total slack is available
- Admit aperiodic tasks while slack is not exceeded



PART III

Energy



Power of Computation

- Terminology

- R : Power spent on computation
- V : Processor voltage
- f : Processor clock frequency
- R_0 : Leakage power

- Power spent on computation is:

- $R = k_v V^2 f + R_0$
where k_v is a constant



Energy of Computation

- Power spent on computation is:
 - $R = k_v V^2 f + R_0$
- Consider a task of length C clock cycles and a processor operating at frequency f
- The execution time is $t = C/f$
- Energy spent is:
 - $E = R t = (k_v V^2 f + R_0)(C/f)$



Reducing Processor Frequency

Good or Bad?

- Does it make sense to operate the processor at a reduced speed to save energy? Why or why not?

Possible Answer:

$$E = R t = (k_v V^2 f + R_0)(C/f) = k_v V^2 C + R_0 C/f$$

- Conclusion: E is minimum when f is maximum.
 - Operate at top speed
- Is this really true? What are the underlying assumptions?

Dynamic Voltage Scaling (DVS):

Reducing Voltage and Frequency

- Processor voltage can be decreased if clock frequency is decreased
 - Voltage and frequency can be decreased roughly proportionally.
 - In this case (where $V \sim f$):

$$R = k_f f^3 + R_0$$

$$E = (k_f f^3 + R_0)(C/f) = k_f f^2 C + R_0 C/f$$

Dynamic Voltage Scaling (DVS):

Reducing Voltage and Frequency

- Processor voltage can be decreased if clock frequency is decreased
 - Voltage and frequency can be decreased roughly proportionally.

$$R = k_f f^3 + R_0$$

$$E = (k_f f^3 + R_0)(C/f) = k_f f^2 C + R_0 C/f$$

- Question: Does reducing frequency (and voltage) increase or decrease total energy spend on a task?

Dynamic Voltage Scaling (DVS):

The Critical Frequency

- There exists a minimum frequency below which no energy savings are achieved

$$E = k_f f^2 C + R_0 C / f$$

$$dE/df = 2k_f f C - R_0 C / f^2 = 0$$

$$f = \sqrt[3]{\frac{R_0}{2k_f}}$$



DVS Algorithm 1: Static Voltage Scaling

1. Calculate the critical frequency
2. Calculate the minimum frequency at which the task set remains schedulable
 - Example: If EDF is used and the utilization is 60% at the maximum frequency f_{max} , then the frequency can be decreased to $0.6 f_{max}$.
3. Let f_{opt} be the larger of the above two
4. Operate the system at the smallest frequency at or above f_{opt} .



DVS Algorithm 2: Cycle-conserving DVS

- What if a task finishes early?
 - Re-compute the utilization based on the reduced execution time.
 - Calculate the minimum frequency at which the task set is schedulable using the new utilization.
 - Update task execution times to the WCET when new invocations are released.

Practical Consideration:

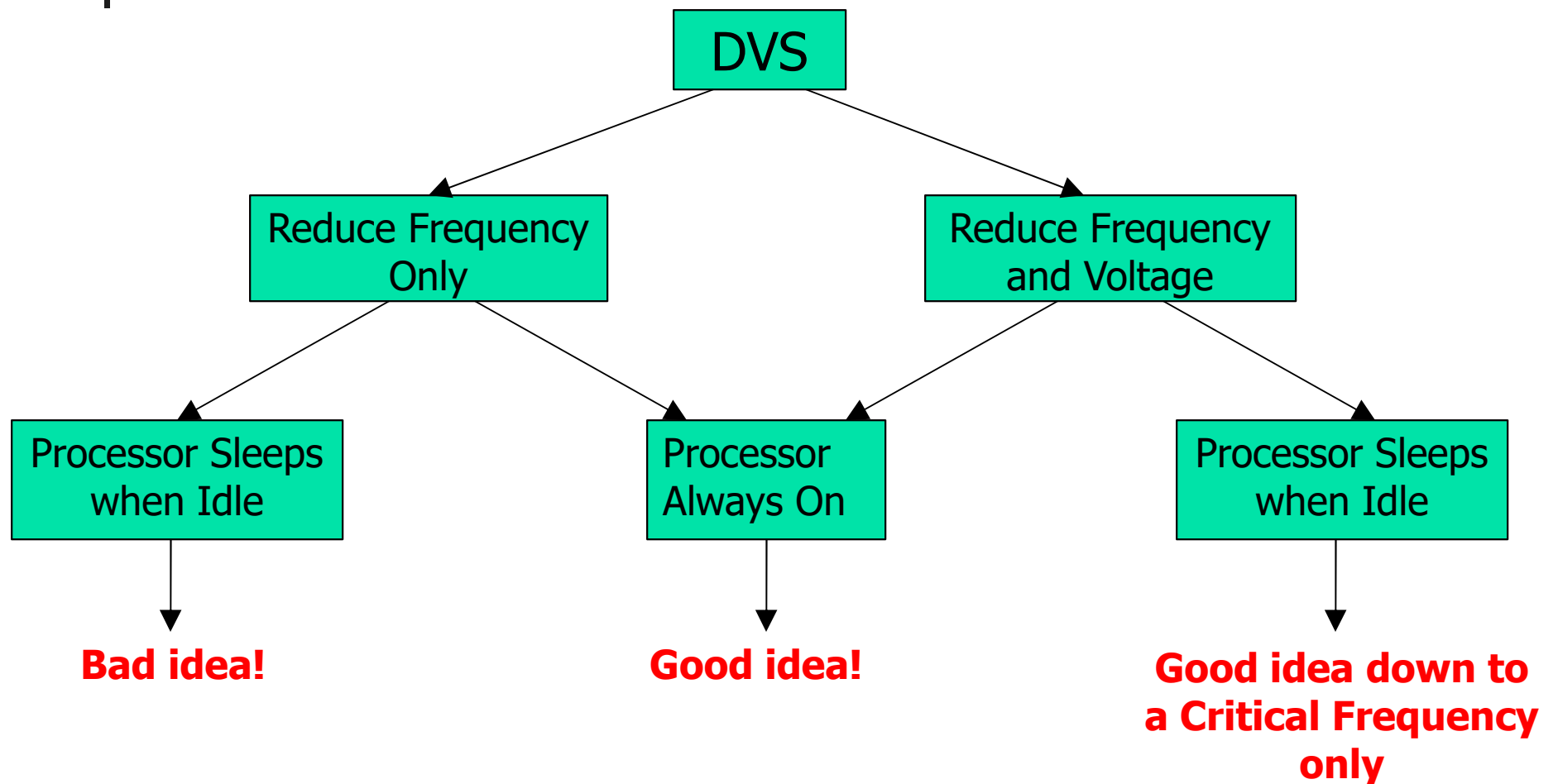
Accounting for Off-chip Overhead

- In the preceding discussion, we assumed that task execution *time* at frequency f is C/f , where C is the total cycles needed
- In reality some cycles are lost waiting for memory access and I/O (Off-chip cycles).
 - Let the number of CPU cycles used be C_{cpu} and the time spent off-chip be $C_{off-chip}$
 - Execution time at frequency f is given by

$$C_{cpu}/f + C_{off-chip}$$



Recap





Processor Performance States (P-States)

- **P0** max power and frequency
- **P1** less than P0, voltage/frequency scaled
- **P2** less than P1, voltage/frequency scaled
- ...
- **P_n** less than $P(n-1)$, voltage/frequency scaled



Processor “Sleep” States (C-states)

- **C0**: is the operating state.
- **C1** (often known as *Halt*): is a state where the processor is not executing instructions, but can return to an executing state instantaneously. All ACPI-conformant processors must support this power state.
- **C2** (often known as *Stop-Clock*): is a state where the processor maintains all software-visible state, but may take longer to wake up. This processor state is optional.
- **C3** (often known as *Sleep*) is a state where the processor does not need to keep its cache, but maintains other state. This processor state is optional.



Turning Processors Off

The Cost of Wakeup

- Energy expended on wakeup, E_{wake}
- To sleep or not to sleep?

- Not to sleep (for time t):

$$E_{no-sleep} = (k_v V^2 f + R_0) t$$

- To sleep (for time t) then wake up:

$$E_{sleep} = P_{sleep} t + E_{wake}$$

- To save energy by sleeping: $E_{sleep} < E_{no-sleep}$

$$t > \frac{E_{wake}}{k_v V^2 f + R_0 - P_{sleep}}$$

DPM and the Problem with Work-conserving Scheduling

- No opportunity to sleep ☹️

Task 1 ($C=2, P=12$)



Task 2 ($C=1, P=16$)

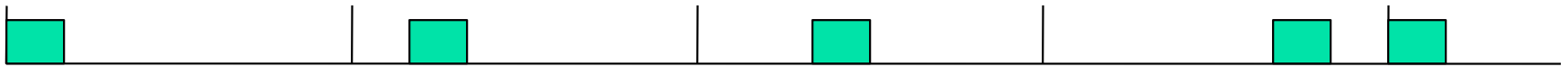


Minimum sleep period

DPM and the Problem with Work-conserving Scheduling

- Must batch! 😊

Task 1 (C=2, P=12)



Task 2 (C=1, P=16)



Minimum sleep period

Device Forbidden Regions

- Treat sleep periods like the *highest-priority* sporadic task. Use *response time analysis* for schedulability. Problems?
 - A Valid solution, but pessimistic.
(Called: Device Forbidden Regions. Published in RTAS 2008.)

Task 3 (C=11, P=16)



Task 1 (C=2, P=12)



Task 2 (C=1, P=16)





How Many Processors to Use?

- Consider using one processor at frequency f versus two at frequency $f/2$
- Case 1: Total power for one processor
 - $k_f f^3 + R_0$
- Case 2: Total power for two processors
 - $2 \{k_f (f/2)^3 + R_0\} = k_f f^3 / 4 + 2 R_0$
- The general case: n processors
 - $n \{k_f (f/n)^3 + R_0\} = k_f f^3 / n^2 + n R_0$

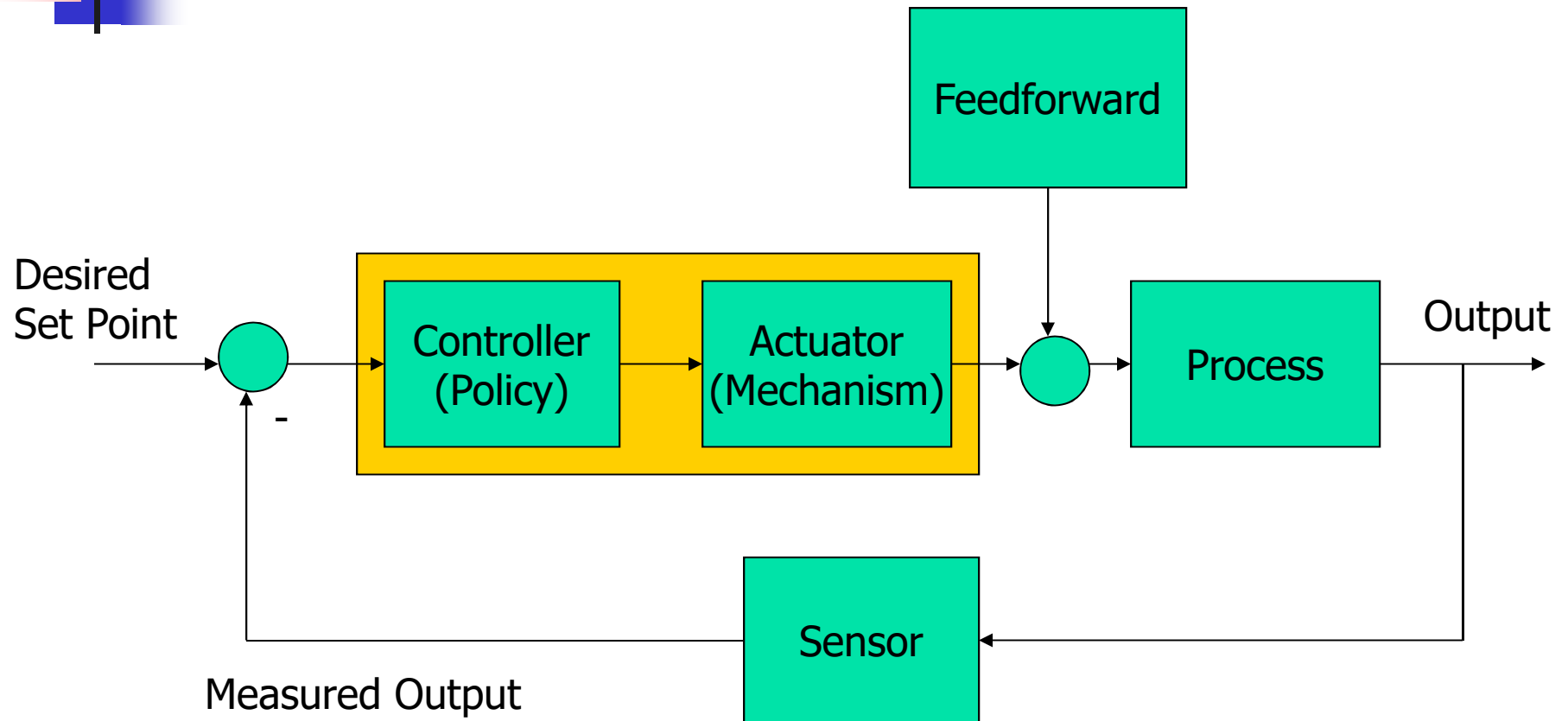


How Many Processors to Use?

- The general case: n processors
 - $Power = n \{k_f (f/n)^3 + R_0\} = k_f f^3 / n^2 + n R_0$
 - $dPower/dn = -2 k_f f^3 / n^3 + R_0 = 0$

$$n = \sqrt[3]{\frac{2k_f f^3}{R_0}}$$

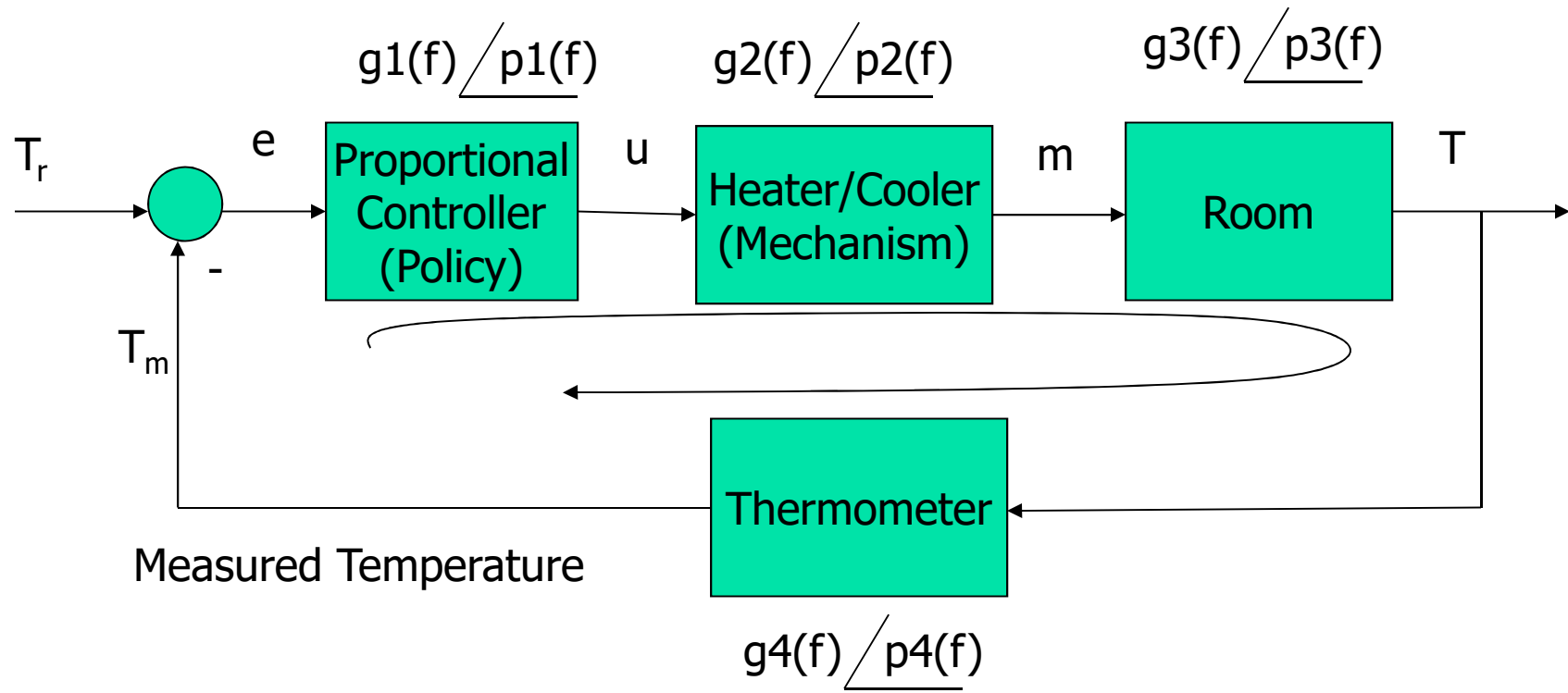
Classical Feedback Control Loops



Stability – Recap

Phase equation: $\sum_i p_i(f) = \pi$ \longrightarrow f is obtained

Gain equation: $\prod_i g_i(f)$ must be less than 1 for stability





Summary of Basic Elements

Input = $\sin(\omega t)$

Element	Gain	Phase
Integrator	$1/\omega$	$-\pi/2$
Differentiator	ω	$\pi/2$
Pure delay element (Delay = D)	1	$-\omega D$
First order lag (time constant = τ)	$K / \sqrt{1 + (\tau\omega)^2}$	$-\tan^{-1}(\omega\tau)$
Pure gain (Gain = K)	K	0

Note:

$$\omega = 2\pi f_{osc}$$

Where f_{osc} is
the loop
frequency of
oscillation

Steady State Error

At steady state the system “catches up” – phase shift is zero.

$$T_r - e \cdot g_1(f) \cdot g_2(f) \cdot g_3(f) \cdot g_4(f) = e$$

$$e = \frac{T_r}{1 + \prod_i g_i(f)}$$

