

## 2. Periodic Tasks

---

2.1 Basic concepts

2.2 Principles of fixed priority analysis

2.3 Rate monotonic priorities

2.4 Utilization test

2.5 Response time analysis

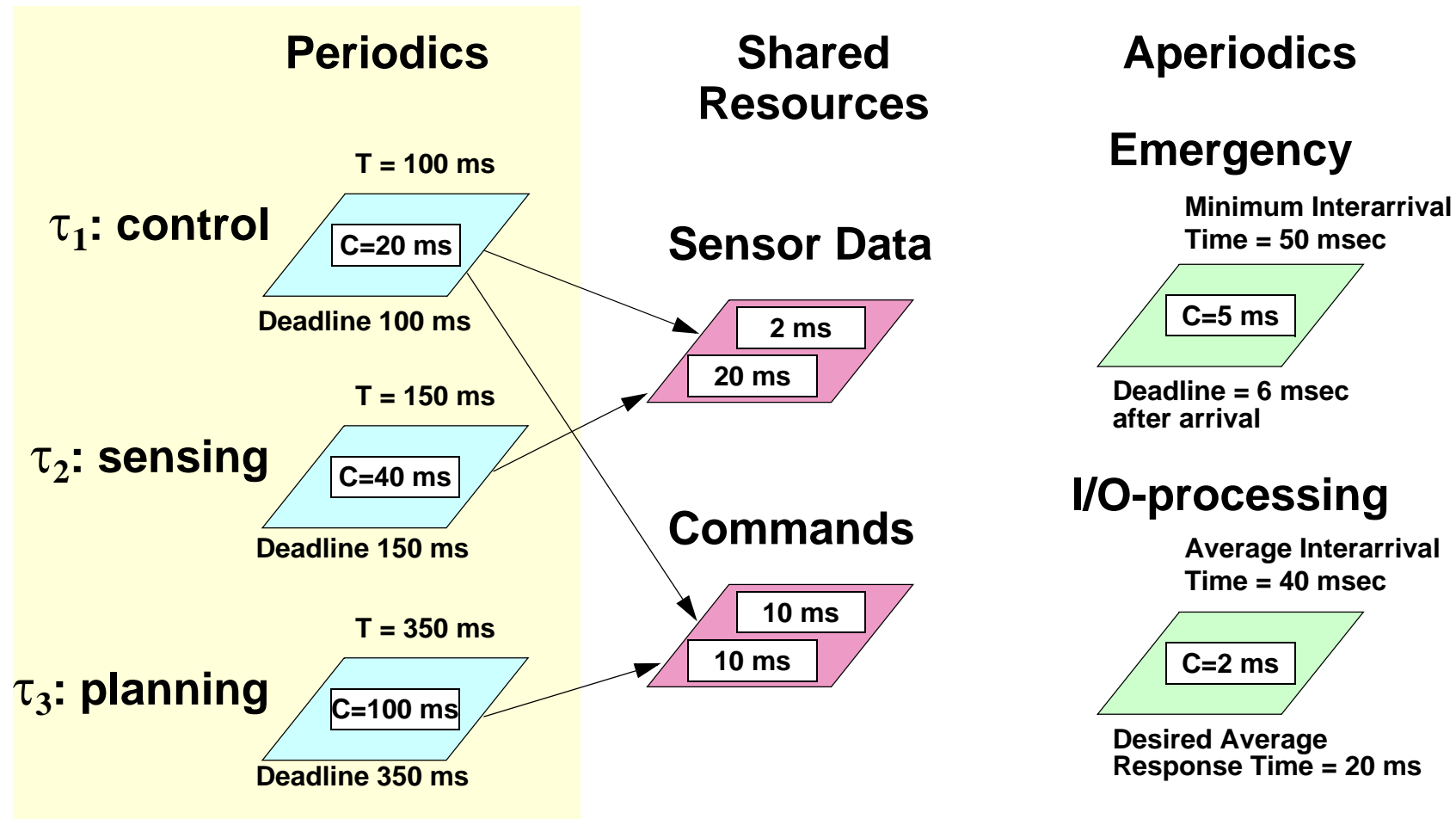
## Notes:



---

In this section, we will discuss the basic principles of Rate Monotonic Analysis. We will present methods to analyze the schedulability of a set of periodic independent tasks. This kind of task set is very unrealistic, but the techniques show in this chapter are easily extended to support all the issues that appear in practical real-time systems.

# 2.1 Basic Concepts: A Sample Problem - Periodics



## Notes:

---

This sample task set is simple and yet embodies many different types of real-time requirements. This task set is completely analyzable using the methods outlined in this presentation. For now we will concentrate on the periodic requirements and ignore any pre-period deadlines:

- Periodic task  $\tau_1$ : execution time = 20 msec; period = 100 msec; deadline is at the end of each period.  
(we consider the actual deadline of  $\tau_1$  later)
- Periodic task  $\tau_2$ : execution time = 40 msec; period = 150 msec; deadline is at the end of each period .
- Periodic task  $\tau_3$ : execution time = 100 msec; period = 350 msec; deadline is at the end of each period.

# Concepts and Definitions - Periodics

## Periodic task

- initiated at fixed intervals
- must finish before start of next cycle

Task's CPU utilization:  $U_i = C_i/T_i$

- $C_i$  = compute time (execution time) for task  $\tau_i$
- $T_i$  = period of task  $\tau_i$
- $P_i$  = priority of task  $\tau_i$
- $D_i$  = deadline of task  $\tau_i$
- $\phi_i$  = phase of task  $\tau_i$
- $R_i$  = response time of task  $\tau_i$

CPU utilization for a set of tasks:  $U = U_1 + U_2 + \dots + U_n$

## Notes:

A periodic task becomes ready to execute at fixed intervals. The time between task initiations is called the *period* of the task. The deadline of a task is the maximum time at which the execution of the task must have been completed, counting from the instant of task initiation. For the moment, we will consider task deadlines to coincide with the start of the next period.

$C_i$  and  $T_i$  represent the worst-case execution time and period, respectively, for task  $\tau_i$ . A task's utilization of the CPU is  $C_i/T_i$ . Total CPU utilization is the sum of the utilizations of all the tasks. The deadline of task  $\tau_i$  is  $D_i$ . The priority of task  $\tau_i$  is  $P_i$ . The initial phase of task  $\tau_i$  is  $\phi_i$ ; the phase is the time at which the task is first activated. Since phases may drift, we will always assume the worst-case phase in the analysis.

The worst-case response time of task  $\tau_i$  is  $R_i$ . This is the maximum difference between the activation time (i.e., the beginning of the task's period) and the response time.

## 2.2 Principles of Fixed Priority Analysis

Two concepts help building the worst-case condition under fixed priorities:

- **Critical instant.** The worst-case response time for all tasks in the task set is obtained when all tasks are activated at the same time
- **Checking only the deadlines in the worst-case busy period.**
  - for task: interval during which the processor is busy executing  $\tau_i$  or higher priority tasks

Based on these concepts, several results arise:

- Optimality of rate monotonic priorities
- Utilization bound test
- Exact test

## Notes:

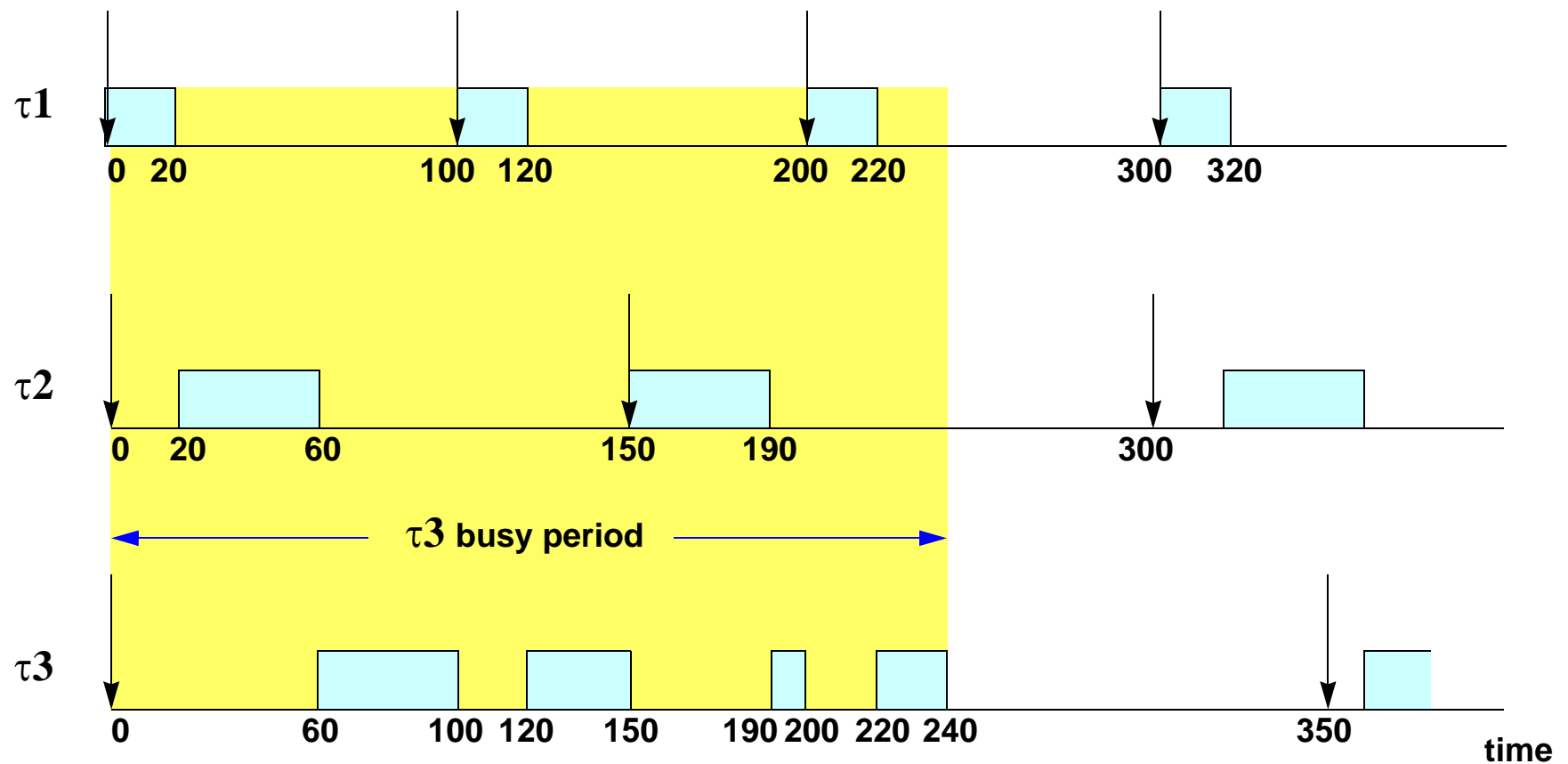
The analysis problem that we have to solve is determining if a set of periodic independent tasks with deadlines coincident with the end of their periods is schedulable under a fixed-priority preemptive scheduler. To determine schedulability, one must determine that the deadlines will be met under all possible circumstances.

The basic RMA results for solving this problem are based on two very simple concepts, that help to build the worst-case condition for that set of tasks:

- *Critical instant.* The worst-case response time for any task in the task set is obtained when all tasks are activated at the same time, i.e., when the phase of each task is zero.
- *Checking the first deadline.* When all tasks are activated at the same time, if a task meets its first deadline, it will always meet all of its deadlines



# Example of a Critical Instant



## Notes:

The figure above shows how to interpret the critical instant results. It shows the execution sequence for the set of three tasks from our example. Task  $\tau_1$  has been assigned the highest priority, and preempts any other activity running in the system. Its response time is equal to its worst-case execution time (assuming no context switch overhead). Task  $\tau_2$  can sometimes be preempted by  $\tau_1$  and thus its worst-case response time is 60 time units, which is the response time to the first activation. Task  $\tau_3$  can be preempted both by  $\tau_1$  and  $\tau_2$ . The figure shows that it is preempted multiple times before it completes its response to the first activation, at  $t=240$  time units; this represents its worst-case response time.

The critical-instant and checking-the-first-deadline concepts show a very simple way to construct the worst case that can ever happen for a given task. It corresponds to the first activation, when all the other tasks are activated at the same time. If the task set meets its deadlines for the worst case, then it will always meet all deadlines. Based upon the proofs of these concepts, several analytical results have been developed that help in analyzing a system composed of periodic tasks; these results are discussed in the following slides.

## 2.3 Rate Monotonic Priorities

---

For a set of tasks with the following characteristics:

- periodic,
- independent,
- deadlines = periods,
- fixed-priority preemptive scheduling,

the optimum priority assignment is called rate monotonic:

- tasks with smaller periods get higher priorities

Optimum means that if the task set is schedulable with a given fixed-priority assignment it is also schedulable with rate monotonic priorities

## Notes:

---

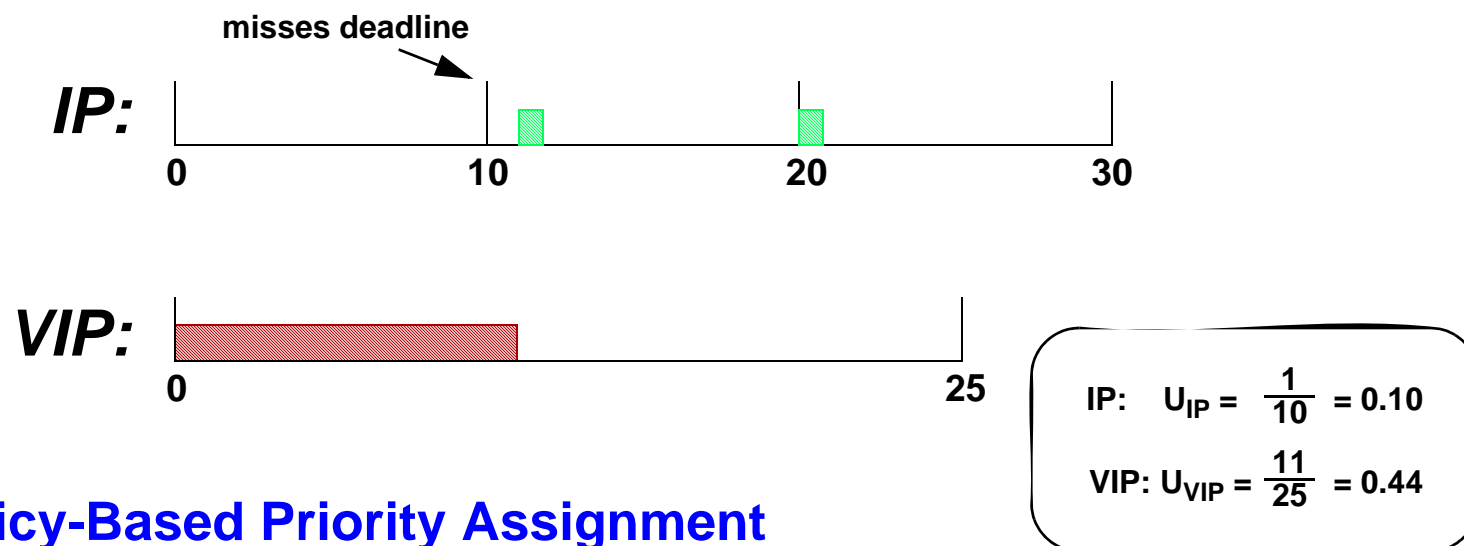
Rate Monotonic Scheduling is the name of a fixed-priority preemptive scheduling policy in which priorities are assigned ordered according to the task's rate: tasks with smaller periods get higher priorities.

Rate Monotonic priorities are optimum for sets of periodic independent tasks, when deadlines coincide with the end of the periods. Optimality means that if the task set is schedulable under any fixed priority arrangement, so it is under rate monotonic scheduling.

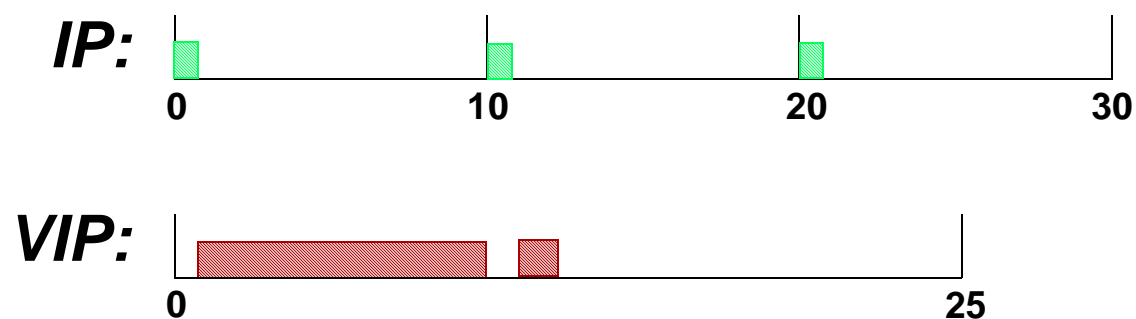
Notice that under the restrictions mentioned above it is not appropriate to assign priorities based upon semantic importance, but it is better to use the optimum priority assignment.

# Example of Priority Assignment

## Semantic-Based Priority Assignment



## Policy-Based Priority Assignment



## Notes:

---

This example provides some motivation for using the rate monotonic scheduling algorithm. In the example we have 2 tasks, named IP and VIP, with the following descriptions:

Task IP (Important Task):  $C_{IP} = 1$ ;  $T_{IP} = 10$ ;  $U_{IP} = 0.10$

Task VIP (Very Important Task):  $C_{VIP} = 11$ ;  $T_{VIP} = 25$ ;  $U_{VIP} = 0.44$

Total utilization: 54%

If we assign higher priority to task VIP, then task IP will miss its deadline, even though the total utilization is only 54%. But if priorities are assigned according to the rate monotonic algorithm, then both tasks will meet their deadlines. Notice that, as VIP's period increases, total utilization decreases to 10%, since  $U_{VIP}$  approaches zero for large  $T_{VIP}$ ; yet deadlines are still missed.

As an aside, note that many other scheduling policies (such as earliest deadline first, or least laxity first) result in the same priority assignment and the same schedule. The point of this example is that there are potential drawbacks to assigning priorities on the basis of semantic importance (degree of application criticality).

## 2.4 Utilization Bound Test

**Utilization Bound Test:** A set of  $n$  independent periodic tasks, with deadlines at the end of the periods, scheduled by the rate monotonic algorithm will always meet its deadlines, for all task phasings, if

$$\frac{C_1}{T_1} + \dots + \frac{C_n}{T_n} \leq U(n) = n(2^{1/n} - 1)$$

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

For harmonic task sets, the utilization bound is  $U(n)=1.00$  for all  $n$ .

## Notes:

We say a set of tasks is **schedulable** if the task set is guaranteed to meet all its deadlines. The utilization bound test says that a task set is schedulable if its total utilization is less than a certain bound. The utilization bound test provides the basic formula underlying the theory. More complex formulas provide better bounds. This test is the application of a theorem, which was proved by Liu and Layland in “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment”, *JACM*, 1973.

The utilization bound test assumes that:

1. The processor always executes the highest priority ready task (fixed-priority preemptive).
2. Task priorities are assigned according to rate monotonic policy.
3. Tasks do not synchronize with each other.
4. Each task’s deadline is at the end of its period.
5. Tasks do not suspend themselves in the middle of computations.

The test takes into account the preemption time and execution time for each task under a worst-case combination of periods and execution times. As the number of tasks goes to infinity, the bound approaches  $\ln(2) = 0.693$ ; in other words, any number of independent periodic tasks will meet their deadlines if the total system utilization is under 69%.

For a harmonic task set (in which the period of each task is a multiple of all higher-frequency tasks), the utilization bound is 1.0 for all task sets.



# Sample Problem: Applying UB Test

	<b>C</b>	<b>T</b>	<b>U</b>
<b>Task <math>\tau_1</math>:</b>	20	100	0.200
<b>Task <math>\tau_2</math>:</b>	40	150	0.267
<b>Task <math>\tau_3</math>:</b>	100	350	0.286

Total utilization is  $.200 + .267 + .286 = .753 < U(3) = .779$

The periodic tasks in the sample problem are schedulable according to the UB test.

## Notes:

Since we are interested in worst-case behavior, tasks utilizations are always rounded up. For example,  $U_2 = 40/150 = .2666\dots$  becomes  $.267$  and  $U_3 = 100/350 = .2857143\dots$  becomes  $.286$ .

In this example, since the total utilization is under the bound for three tasks, all tasks are guaranteed to meet their deadlines, even under worst-case conditions. An additional 24.7% CPU capacity is available for lower-priority tasks that have no deadlines.

Remember that there are several assumptions associated with the utilization bound test:

- Zero context switch overhead.
- Deadlines are at end of period.
- No interrupts are used.
- Priorities are assigned in rate monotonic order.
- Tasks do not interact with one another.
- Tasks do not suspend themselves.

# 2.5 Response Time Analysis Toward a More Precise Test

**UB test has three possible outcomes.**

$$0 \leq U \leq U(n) \Rightarrow \textit{Success}$$

$$U(n) < U < 1,00 \Rightarrow \textit{Inconclusive}$$

$$1,00 < U \Rightarrow \textit{Overload}$$

**UB test is conservative.**

**A more precise test can be applied.**

## Notes:

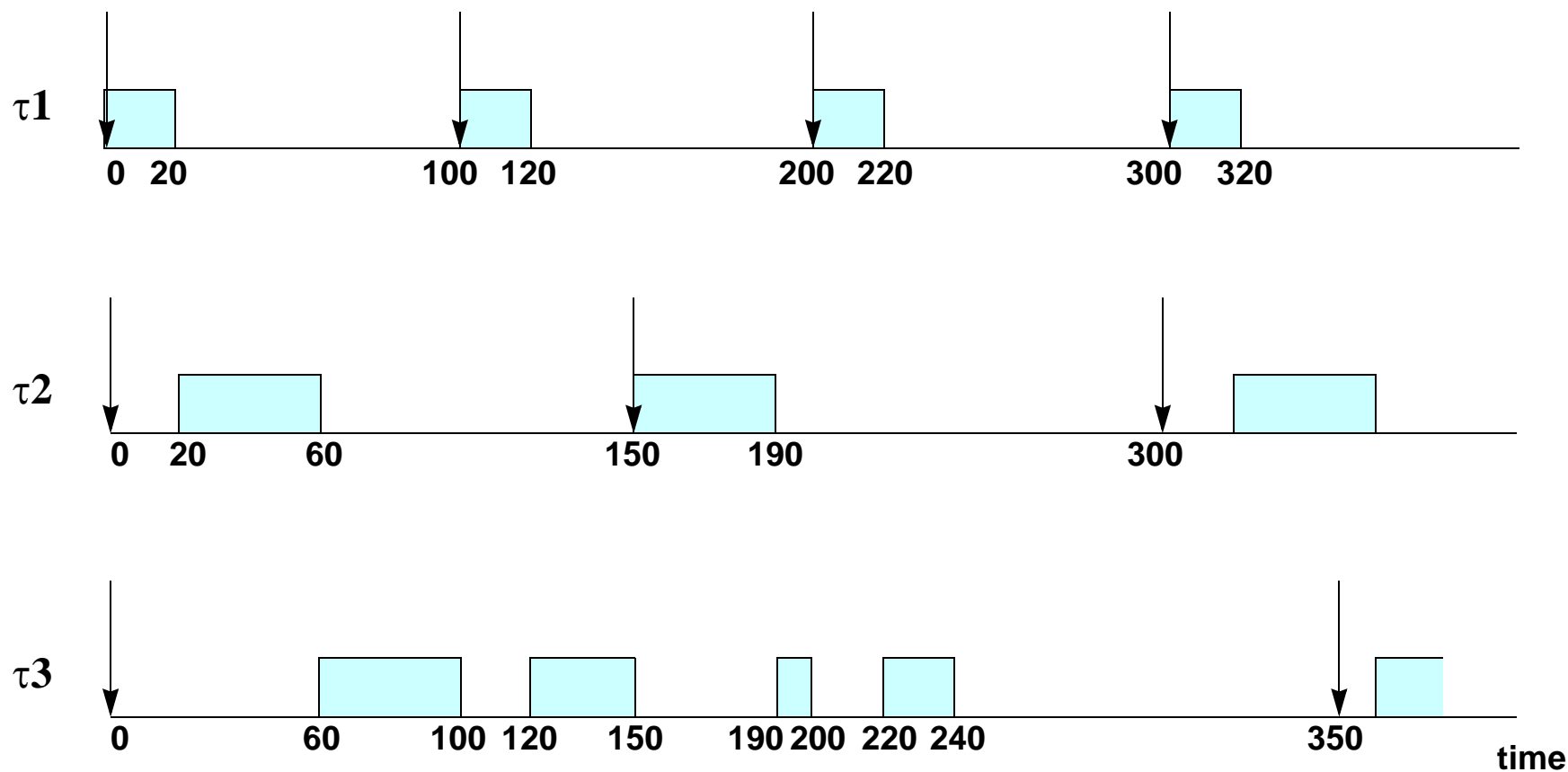
The results of applying the UB test can be:

<b><math>0 &lt; U \leq U(n)</math></b>	<b>success! task set is schedulable</b>
<b><math>U(n) &lt; U &lt; 1.0</math></b>	<b>inconclusive, task set may or may not be schedulable</b>
<b><math>1.0 &lt; U</math></b>	<b>overload, task set exceeds capacity</b>

When the UB test is inconclusive, a more precise test can be applied.

The previous schedulability test (utilization bound test) is very conservative. Schedulable system utilization can often exceed 90%. In fact, for tasks with harmonic periods, where each period is evenly divisible into all longer periods (e.g. 100 ms, 200 ms, 800 ms, 1600 ms), the utilization bound is 100%.

# Timeline for Sample Problem



## Notes:

---

This is the timeline for the periodic tasks of the sample problem (shown on the previous page), in which tasks are lined up in worst-case phasing (i.e. all task are ready to execute at time  $t=0$ ).

Timelines show one possible execution schedule and provide a graphical view of schedule analysis. We will draw timelines according to the following conventions:

- Tasks are arranged and numbered in rate monotonic order, highest frequency at the top.
- We assume Liu and Layland “worst-case” phasing, where all tasks start at time  $t=0$ .
- Execution time for  $\tau_1$  is plotted on its line.
- Execution time for  $\tau_2$  is then plotted on its line, accommodating preemption from  $\tau_1$ 's execution; then this process is repeated for remaining tasks.
- If any task is preempted, its execution time block is divided with a hole in the middle representing the preemption.

# Response time analysis (Harter, 1984; Joseph and Pandya, 1986)



For a set of  $n$  periodic independent tasks, with any fixed priority assignment we can apply the *Response Time Analysis (RTA)*:

- Number the tasks according to priority (highest priority= $\tau_1$ , lowest priority= $\tau_n$ )
- Under a critical instant condition, the amount of work  $W_i(t)$  of priority  $P_i$  or higher started before  $t$  is:

$$W_i(t) = \left\lceil \frac{t}{T_1} \right\rceil C_1 + \dots + \left\lceil \frac{t}{T_{i-1}} \right\rceil C_{i-1} + C_i$$

## Notes:

The critical-instant and check-the-first-deadline principles allow us to make an exact calculation of the worst-case response time of any given task in a periodic independent task set, independently of the priority assignment used.

To perform this calculation, we first provide an equation that allows us to obtain, at any given time  $t$ , the amount of work of priority  $P_i$  or higher that has been initiated in the system. Time  $t=0$  corresponds to the start or the critical instant. The amount of work initiated,  $W_i(t)$ , is calculated by multiplying the number of activations of each applicable task, by the respective execution times.

The number of activations is obtained using a ceiling function  $\lceil x \rceil$ , which means rounding  $x$  to the next integer.

The important issue about this equation is that task  $\tau_i$  will complete its execution when there is no pending work of priority  $P_i$  or higher, which is the same as finding the first instant  $t$  at which  $W_i(t) = t$ , i.e., all work has been completed. This instant can be found by using the iterative method shown in the next slide.



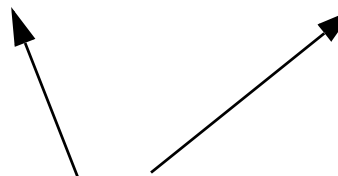
# Response time analysis (Harter, 1984; Joseph and Pandya, 1986)

Iterative test (pseudopolynomial time):

$$a_0 = C_1 + C_2 + \dots + C_i$$

$$a_{k+1} = W_i(a_k) = \left\lceil \frac{a_k}{T_1} \right\rceil C_1 + \dots + \left\lceil \frac{a_k}{T_{i-1}} \right\rceil C_{i-1} + C_i + B_i$$

preemption



execution



blocking



Finish when two consecutive results are the same

# Response time analysis (cont'd)

---

- The iteration stops when

$$a_{k+1} = a_k = R_i$$

- Task  $\tau_i$  is schedulable if:

$$R_i \leq D_i$$

For randomly generated task sets, the exact average utilization bound is 88%

## Notes:

The response time  $R_i$  of task  $\tau_i$  can be obtained by using the iterative equation that appears above. Each iteration consists of calculating the amount of work  $W_i(t)$  activated at the time of the previous iteration. The iteration stops when two successive steps yield the same result, which means that the amount of work initiated until that time is already complete. If the utilization is less than 100%, the iteration is guaranteed to be finite.

The method is called the response time analysis (RTA), or the exact test, because it allows obtaining the exact worst-case response time of a given task. It can be applied by hand for small task sets, but can also be easily implemented in a computer. Besides, it works for any fixed-priority assignment, not only rate monotonic priorities, like the utilization bounds test. The iterative formula for task response time was introduced by Harter in 1984, and later by Joseph and Pandya, "Finding Response Times in a Real-Time System," *BCS Computing Journal*, October 1986 (vol 29 no 5) pp 390-395.

The paper *The Rate Monotonic Scheduling Algorithm -- Exact Characterization and Average Case Behavior*, by Lehoczky, Sha, and Ding, Technical Report, Department of Statistics, Carnegie Mellon University, 1987 presents the fact that for randomly generated task sets, the actual utilization bound is 88% rather than the worst-case bound of 69% presented in the UB test.

# Example: Applying RTA-1

---

Taking the sample problem, we increase the compute time of  $\tau_1$  from 20 to 40; is the task set still schedulable?

Utilization of first two tasks:  $0.667 < U(2) = 0.828$

- first two tasks are schedulable by utilization bound test

Utilization of all three tasks:  $0.953 > U(3) = 0.779$

- utilization bound test is inconclusive
- need to apply response time test

## Notes:

The following data is the same as the sample problem. However, we have changed  $C_1$  from 20 to 40 to illustrate the use of the RTA test.

Task  $\tau_1$ :  $C_1 = 40$ ;  $T_1 = 100$ ;  $U_1 = 0.4$

Task  $\tau_2$ :  $C_2 = 40$ ;  $T_2 = 150$ ;  $U_2 = 0.267$

Task  $\tau_3$ :  $C_3 = 100$ ;  $T_3 = 350$ ;  $U_3 = 0.286$

Utilization of first two tasks:  $.4 + .267 = .667 < U(2) = .828$

Total utilization:  $.4 + .267 + .286 = .953 > U(3) = .779$

When we change  $C_1$  from 20 to 40, CPU utilization changes. Since the utilization of the first two tasks is under the utilization bound of the UB test, these tasks will always meet their deadlines. When the third task is considered, the total utilization is above the general bound, so we must look further to see if the third task can nonetheless meet its deadline.

We will now apply the response time test to task  $\tau_3$ .

# Example: Applying RTA-2

Use RTA test to determine if  $\tau_3$  meets its first deadline

$$a_0 = \sum_{j \leq 3} C_j = C_1 + C_2 + C_3 = 180$$

$$a_1 = \sum_{j < 3} \left\lceil \frac{180}{T_j} \right\rceil C_j + C_3$$

$$??? = \left\lceil \frac{180}{100} \right\rceil (40) + \left\lceil \frac{180}{150} \right\rceil (40) + 100 = 260$$

$$? a_2 = \left\lceil \frac{260}{100} \right\rceil (40) + \left\lceil \frac{260}{150} \right\rceil (40) + 100 = 300$$

## Notes:

To apply the RTA test, we do two steps

1. Use the iterative formula to compute the response time  $R_3$  of  $\tau_3$ .
2. Compare  $R_3$  to the first deadline for  $\tau_3$ , namely  $T_3$ .

If  $R_3 < T_3$ , then  $\tau_3$  is schedulable according to the RTA test.

We start with an initial guess  $a_0 = C_1 + C_2 + C_3$ . This is labeled as the “zero-th” iteration. To compute the first iteration,  $a_1$ , we use the iteration formula:

$$a_1 = \sum_{j < 3} \left\lceil \frac{a_0}{T_j} \right\rceil C_j + C_3$$

So the first iteration is

$$a_1 = 180.$$

Next we apply the formula to compute,  $a_2$ . Since each successive iteration is different, the computation has not yet converged and we must continue.

# Example: Applying RTA-3

---

$$a_2 = 300$$

$$a_3 = \left\lceil \frac{300}{100} \right\rceil (40) + \left\lceil \frac{300}{150} \right\rceil (40) + 100 = 300 \Rightarrow \textit{Done!}$$

$$R_3 = 300 < T_3 = 350$$

Task  $\tau_3$  is schedulable using the RTA test.



## Notes:

---

We continue by applying the iteration formula for the next steps

Finally, the computed value for  $a_3$  is the same as  $a_2$  (300). The iteration has converged, and the worst-case response time for  $\tau_3$  is

$$R_3 = 300.$$

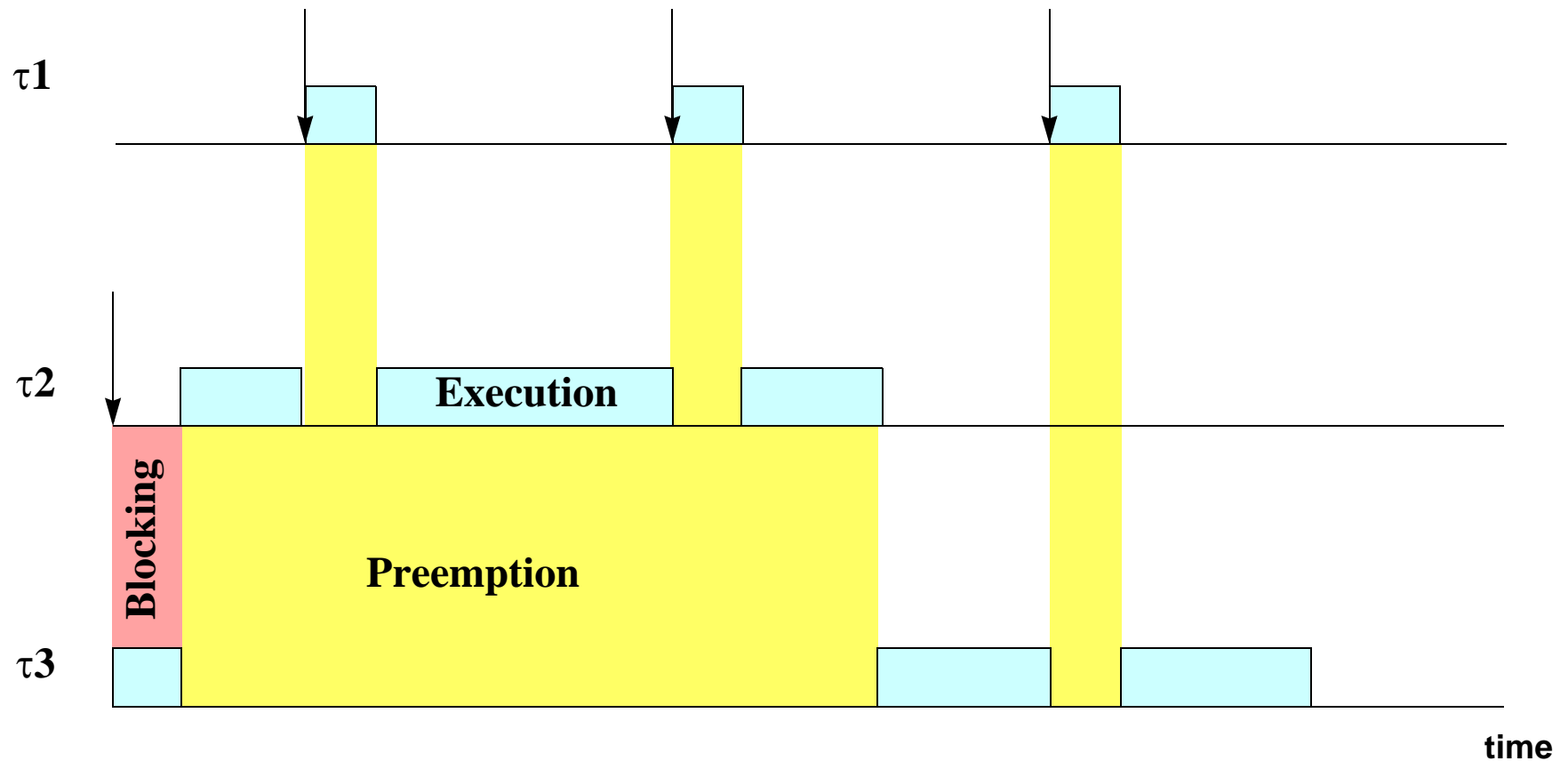
This must be compared to the first deadline for  $\tau_3$ .

We have  $R_3 < T_3$  since  $300 < 350$ .

So  $\tau_3$  is schedulable using the response time test.

Notice that the result is the same obtained when we sketched the timeline for the three tasks. In practice, the response time test is a numerical way to perform the graphical check represented by the timeline.

# Elements that influence the response time



# Summary

Utilization bound test is simple but conservative.

Response time test is more exact but also more complicated.

To this point, UB & RTA tests share the same limitations:

- all tasks run on a single processor
- all tasks periodic and non interacting
- deadlines always at the end of the period
- no interrupts
- zero context switch overhead
- tasks do not suspend themselves

In addition, the UB test has the following limitation:

- rate monotonic priorities assigned

