9. Distributed and multiprocessor systems

UNIVERSIDAD DE CANTABRIA

9.1 Introduction

UNIVERSIDAD DE CANTABRIA

Analysis in a distributed environment:

- **end-to-end deadlines**
- **handling suspension**

Multiprocessor synchronization

Priority Inversion in the Client/Server Approach

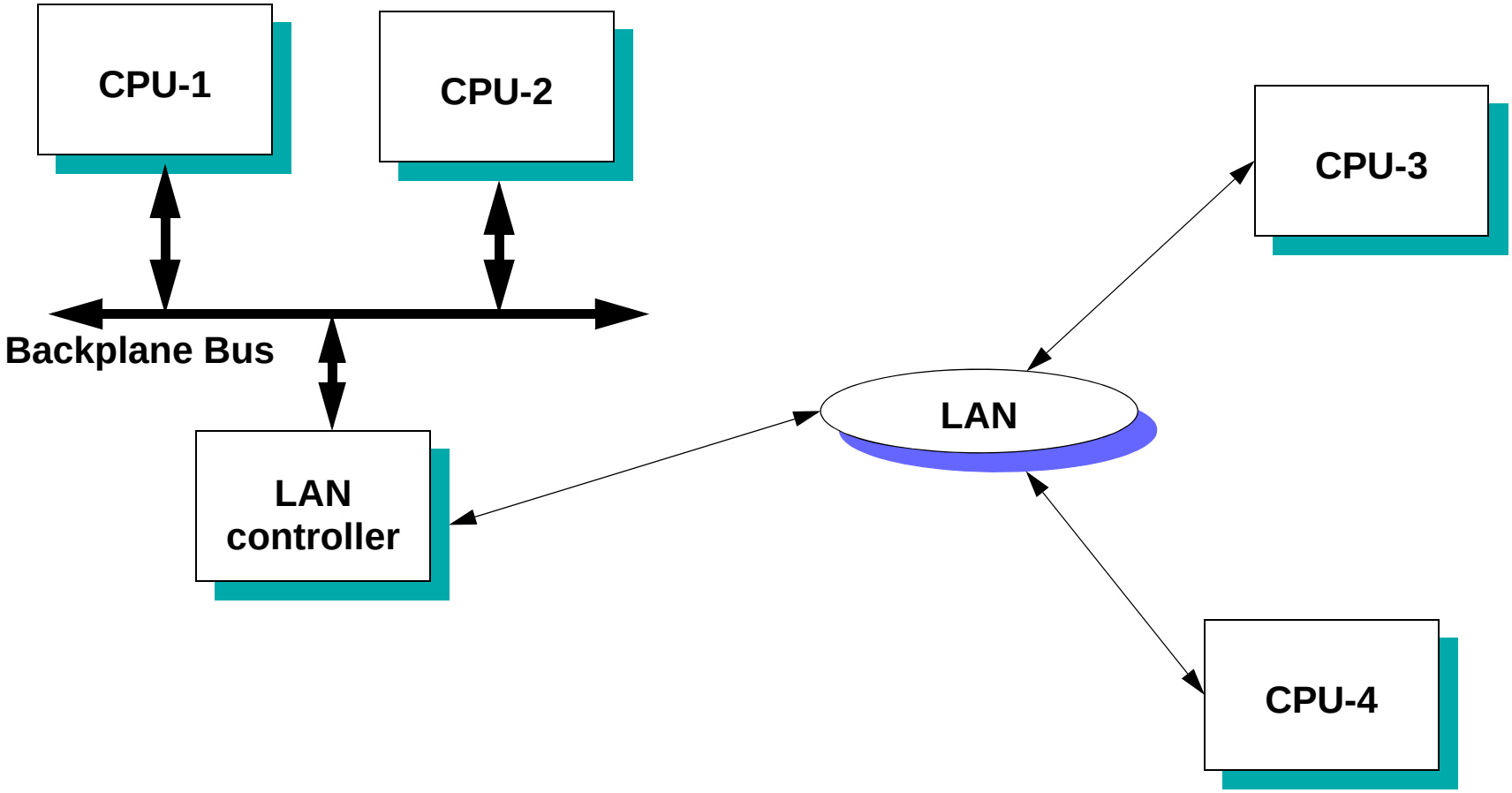
Notes:

UNIVERSIDAD DE CANTABRIA

These are the issues related to multiprocessor and distributed systems, that we want to discuss:

- Analysis in a distributed environment. We want to establish some basic concepts that will allow us to analyze a multiprocessor system using the techniques shown in this tutorial. In particular, we will focus on:
  - Analyzing end-to-end deadlines
  - Handling task suspension
- Multiprocessor synchronization. We will show a new kind of priority inversion that appears in shared memory multiprocessors, called remote blocking. We will also provide solutions to avoid this negative effect.
- Priority Inversion in the Client/Server Approach. The client/server approach is a very popular paradigm used in distributed systems. However, priority inversions may arise that are important for real-time systems. A solution to this problem is discussed here.

Analysis in a Distributed Environment



Notes:

Multiprocessor or distributed systems are different from the conventional uniprocessor systems that we have used until now in this tutorial, because there are multiple resources that can be used simultaneously, such as multiple CPUs, LANs, Backplane Buses, etc. In a distributed environment, many of the events have responses that are executed in several resources. The analysis in one resource influences the other. For example:

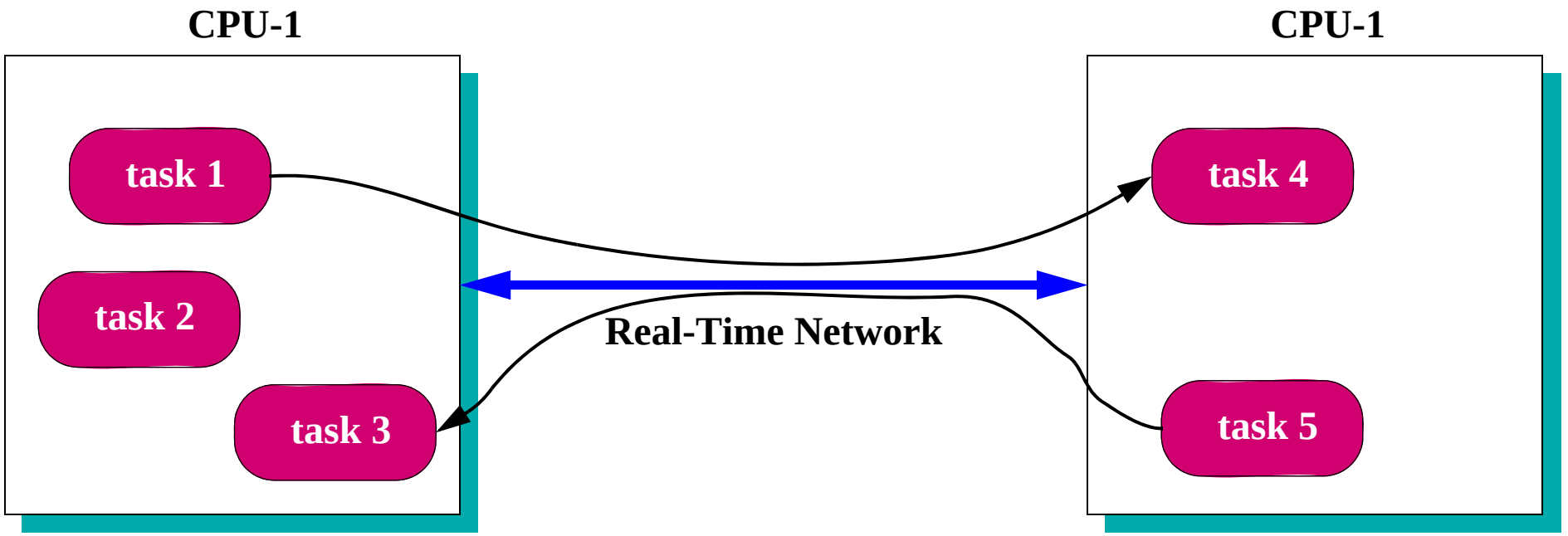
- A chain of responses: An event arrives to one CPU, which processes it; after processing, a message is sent over a LAN; once this message arrives to the destination, another task is activated in a second CPU. We have a chain of three responses, each executed in a different resource.
- A remote procedure call: A server task runs in CPU-1 and accepts requests coming from a LAN. A client task is activated by an event and runs in CPU-2. In the middle of its execution it sends a request to the server task, using the LAN, and suspends itself waiting for a response. When the response arrives from the LAN, the task resumes its execution. We have a chain of five responses in this case (client-1, request message, server, response message, and client-2).

The special characteristics of distributed environments introduce new issues such as:

- end-to-end deadlines
- task suspension
- task allocation and prioritization

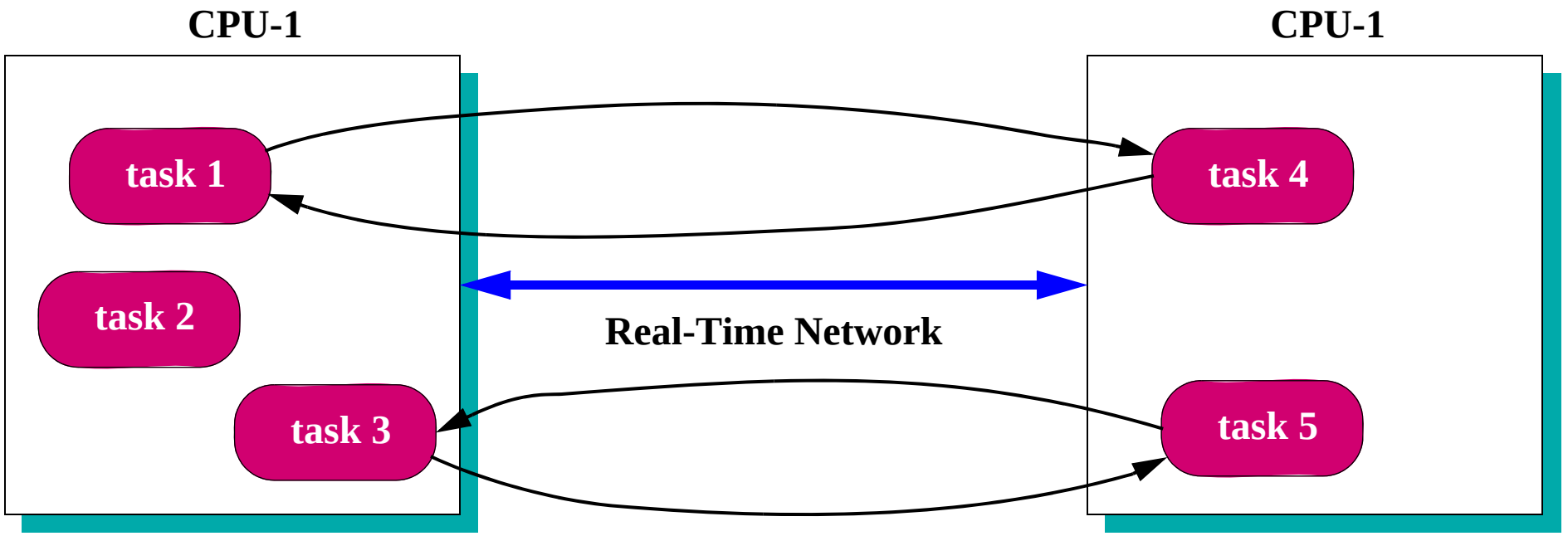
9.2 Message passing systems

Asynchronous message passing



Message passing systems (cont'd)

Remote procedure calls



9.3 Real-time networks

UNIVERSIDAD DE CANTABRIA

Very few networks guarantee real-time response

- many protocols are based on collisions
- no priorities or deadlines in the protocols

Some solutions

- CAN bus
- Priority-based token passing (e.g., RT-EP)
- Time-division multiplexed access
- Point to point networks
- ATM

We will assume a priority-based network

9.4 End-to-End Deadlines

UNIVERSIDAD DE CANTABRIA

It is possible to add response times in different resources to find out the worst-case end-to-end response time

The portions of the distributed response after the first are not initiated periodically. We can use two approaches:

- Analyze the effect of jitter
  - the analysis is complex, because jitter in one resource affects response times in other resources, and viceversa
- Or eliminate jitter by using purely periodic activation or a sporadic server

The analysis of the network is done like the analysis in a CPU, but:

- the scheduling policy is usually different than in the CPUs

Notes:

UNIVERSIDAD DE CANTABRIA

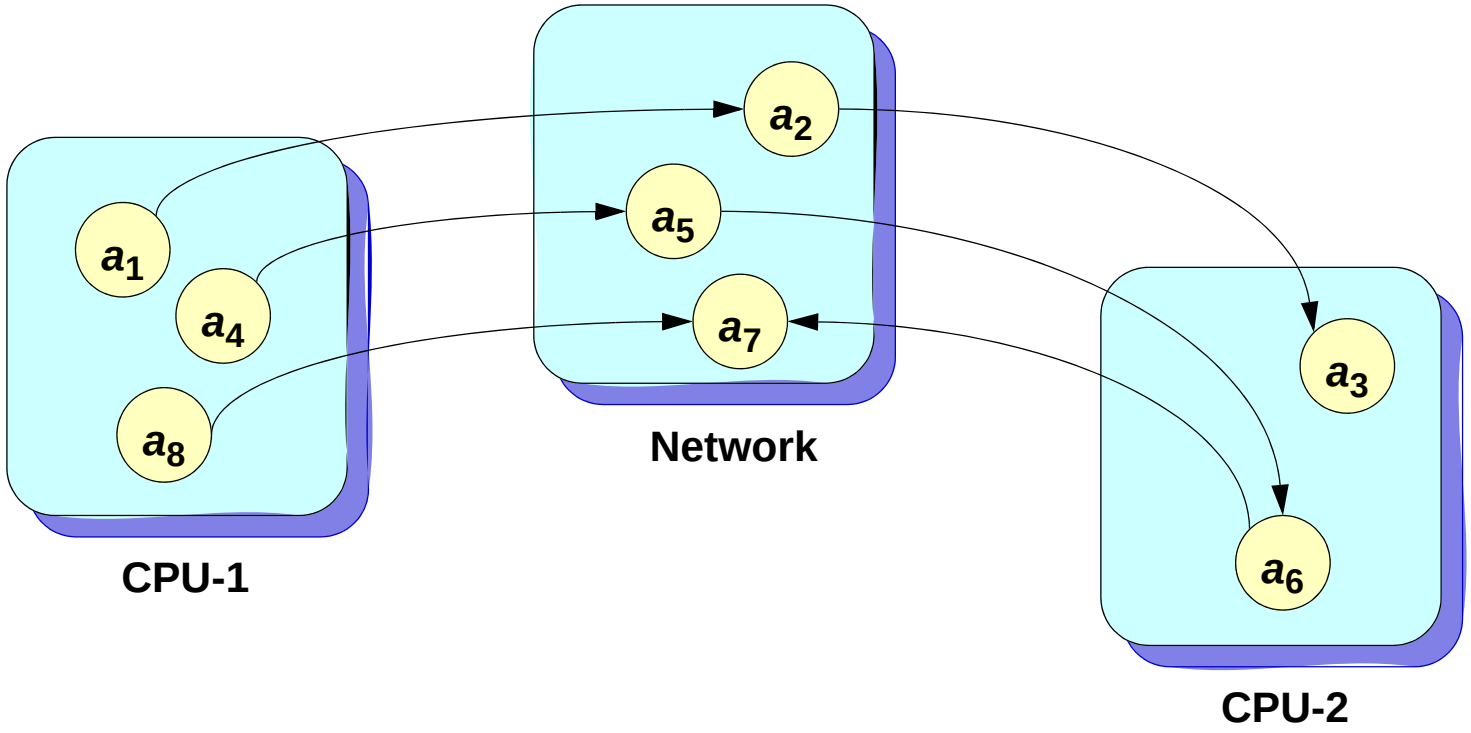
The main problem that appears in a distributed system when a response is comprised of portions that run in different resources, is that all portions except the first are not activated periodically. Since each portion is activated when its predecessor has finished, it has jitter in its activation, which influences the response time of lower priority tasks. We can use two approaches to handle jitter:

- Analyze the effect that jitter has on the schedulability of lower priority responses. We must take into account that the jitter in one resource influences the response times in that resource, which in turn influences the jitter in other resources. However, since the effect of increasing the jitter can only increase the response times, we can apply the schedulability equations iteratively, first assuming that there is no jitter, and then using the jitter from the previous iteration, until we reach a stable result.
- The second approach consists of eliminating jitter by activating each portion of the response at periodic intervals. These intervals are calculated so that when one portion is started, there is guarantee that the previous portion had already completed. The same effect can be used by scheduling each portion of the response with a sporadic server, that will guarantee that the effects on lower priority tasks are not worse than those of an equivalent periodic task.

The schedulability of LANs is usually determined in the same way as in a CPU: messages are treated like tasks, and message transmission times are like task execution times. However, we must be careful, because most LANs do not operate like CPUs, in a priority preemptive way. The analysis must take into account the scheduling policy of the LAN.

Release jitter in distributed systems

In the example below, actions a_2,a_3 and a_5,a_6,a_7,a_8 have jitter, even if a_1 and a_4 are purely periodic:



The problem

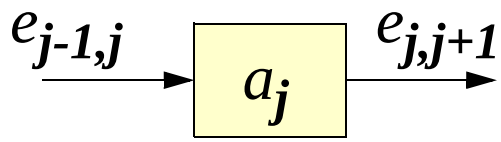
UNIVERSIDAD DE CANTABRIA

Release jitter in one resource depends on the response times in other resources

Response times depend on release jitter

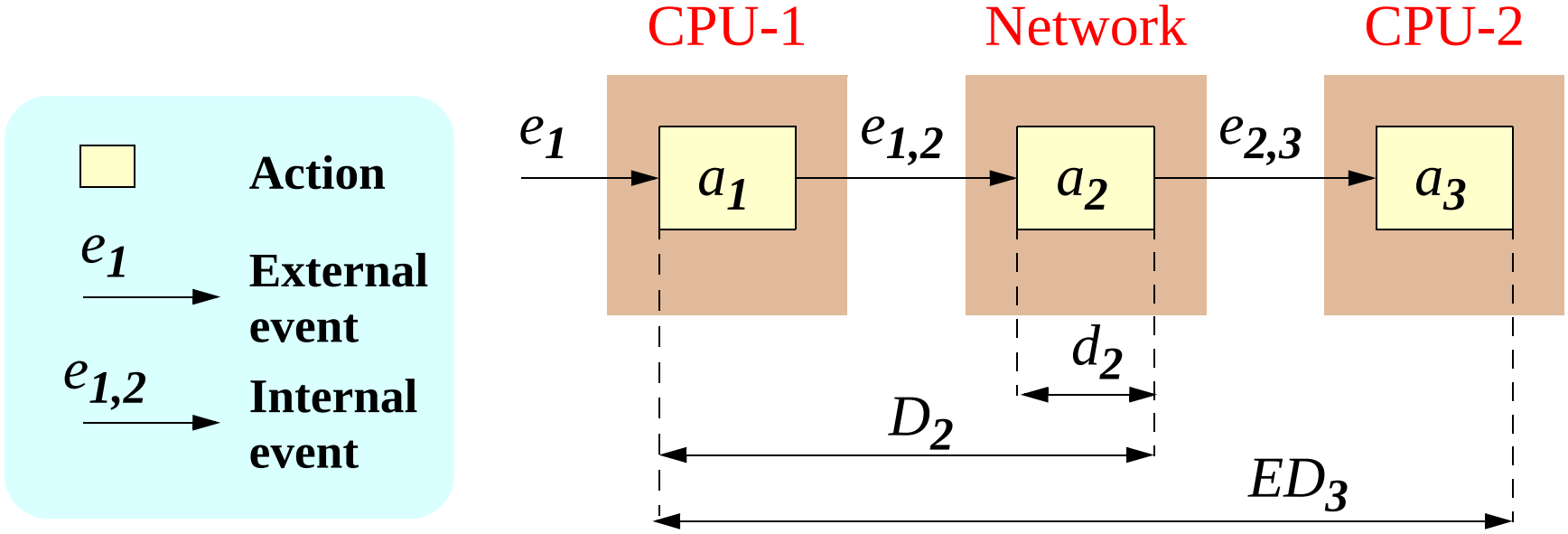
9.5 Transactional model of real-time systems

Linear Action:

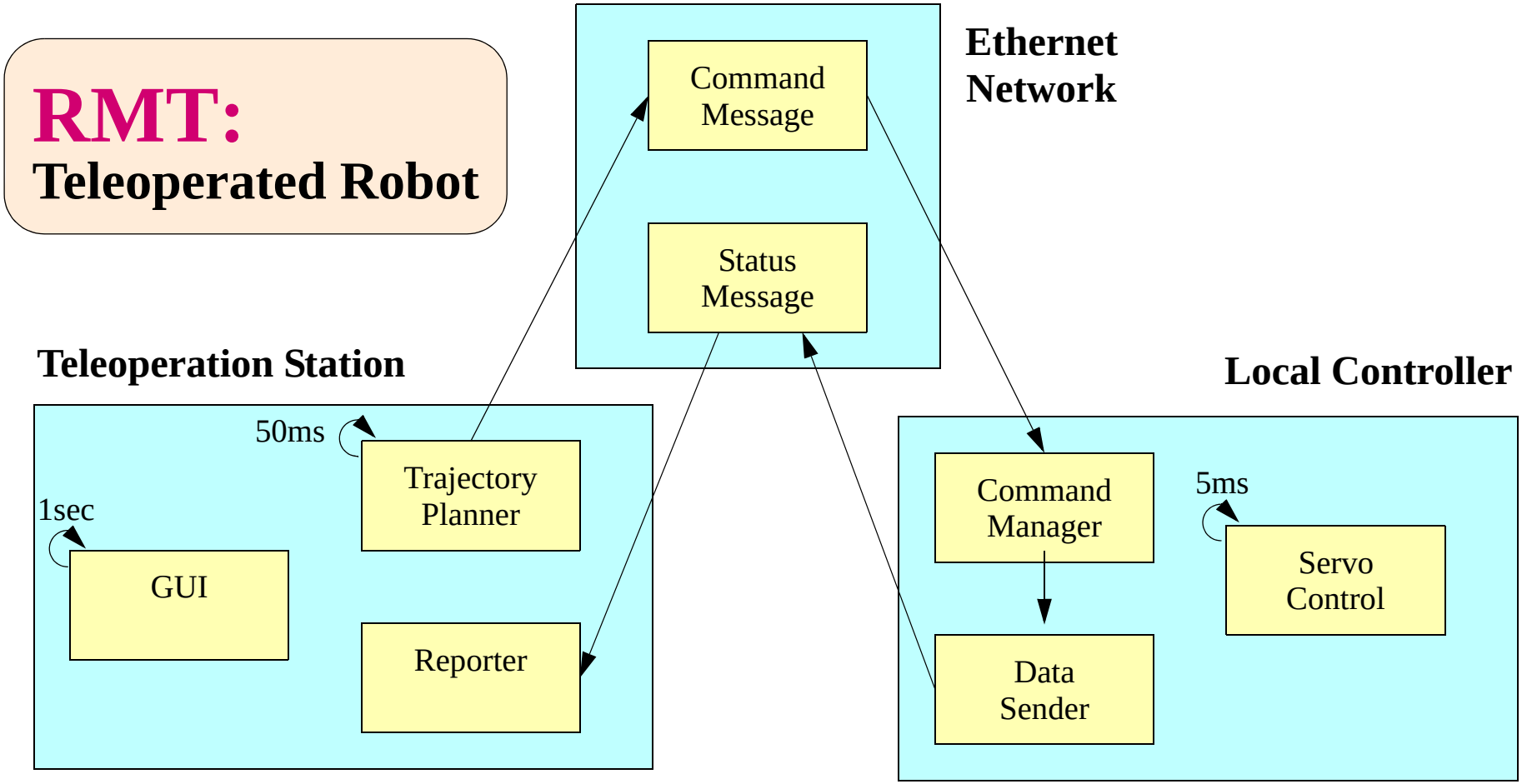


$$T_{j-1,j} = T_j = T_{j,j+1}$$

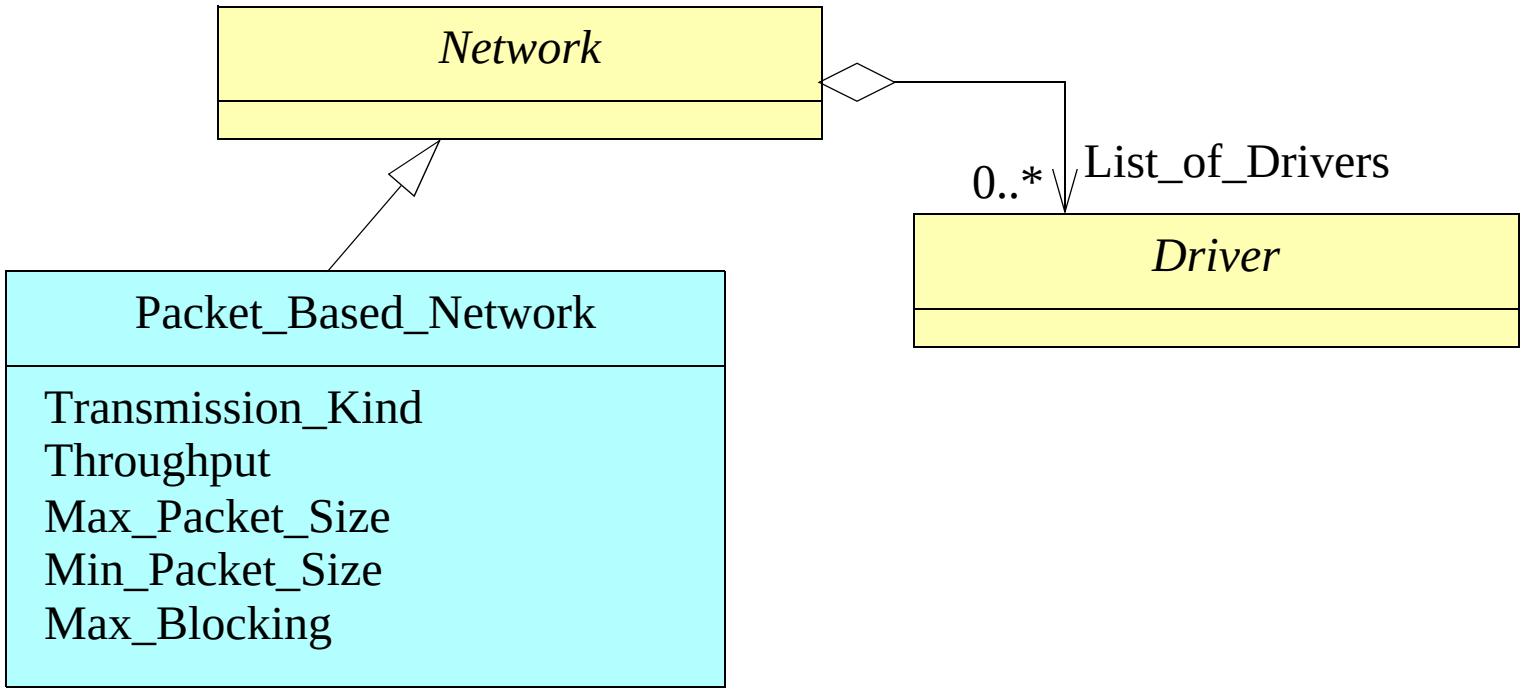
Linear Response to an Event:



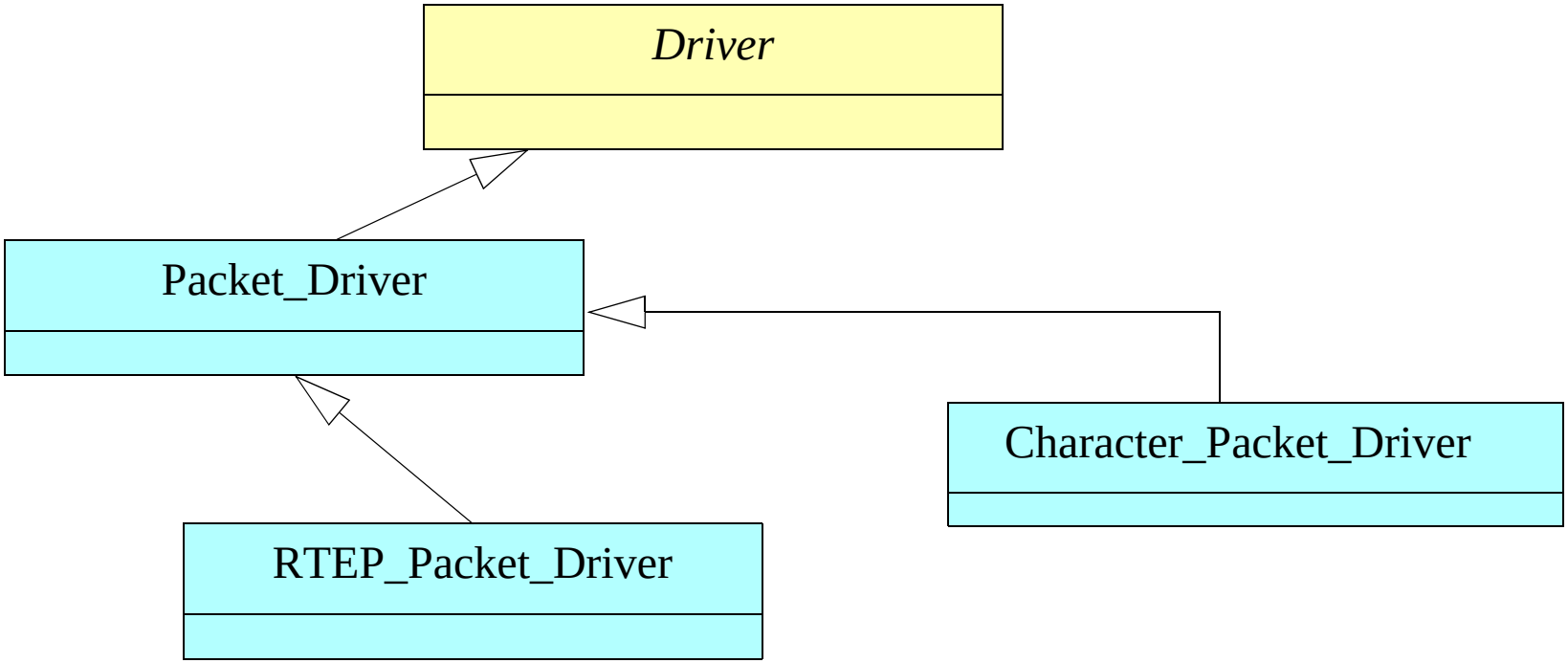
Example


RMT:
Teleoperated Robot
Ethernet
Network
Command
Message
Status
Message
Teleoperation Station
50ms
Trajectory
Planner
1sec
GUI
Reporter
Local Controller
Command
Manager
5ms
Servo
Control
Data
Sender

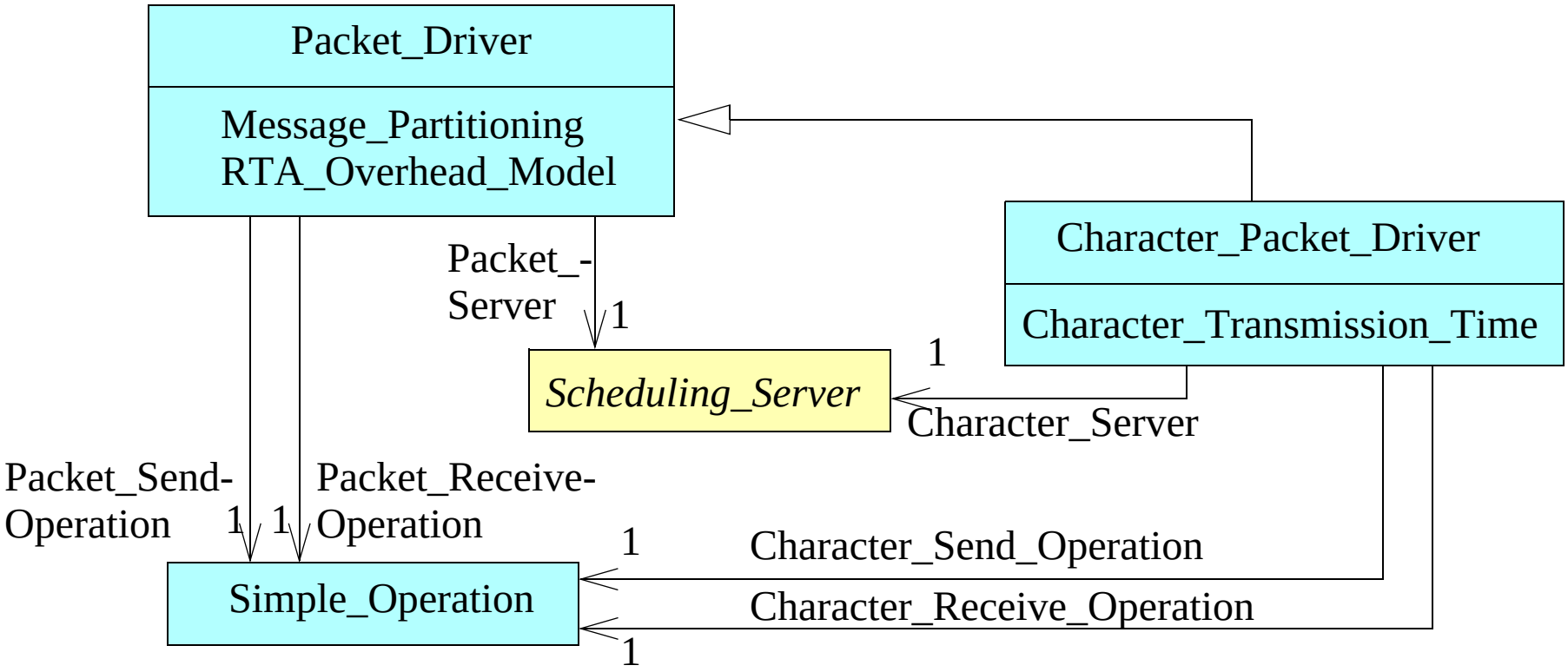
Networks


Network
0..*
List_of_Drivers
Driver
Packet_Based_Network
Transmission_Kind
Throughput
Max_Packet_Size
Min_Packet_Size
Max_Blocking

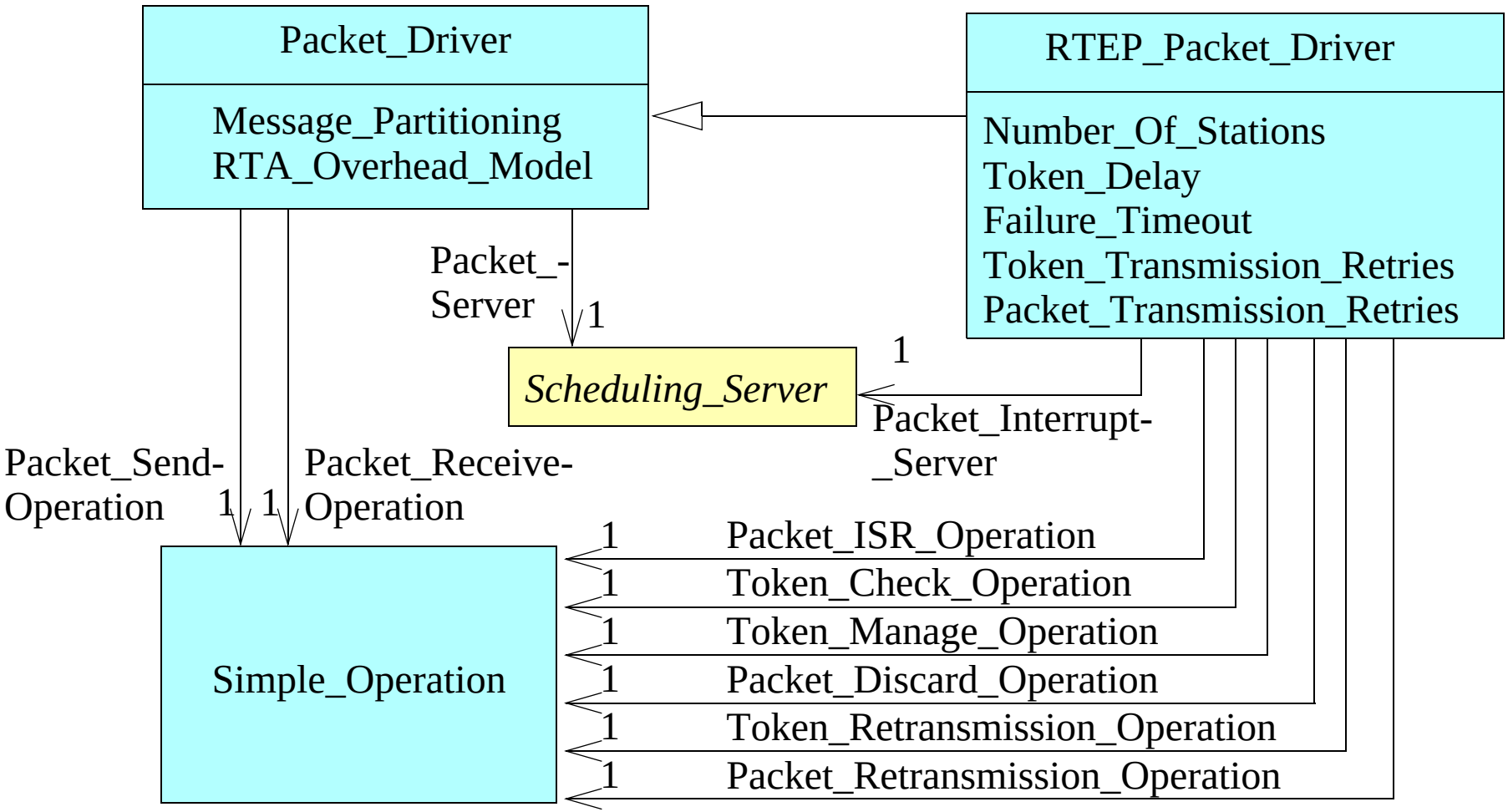
Network Drivers



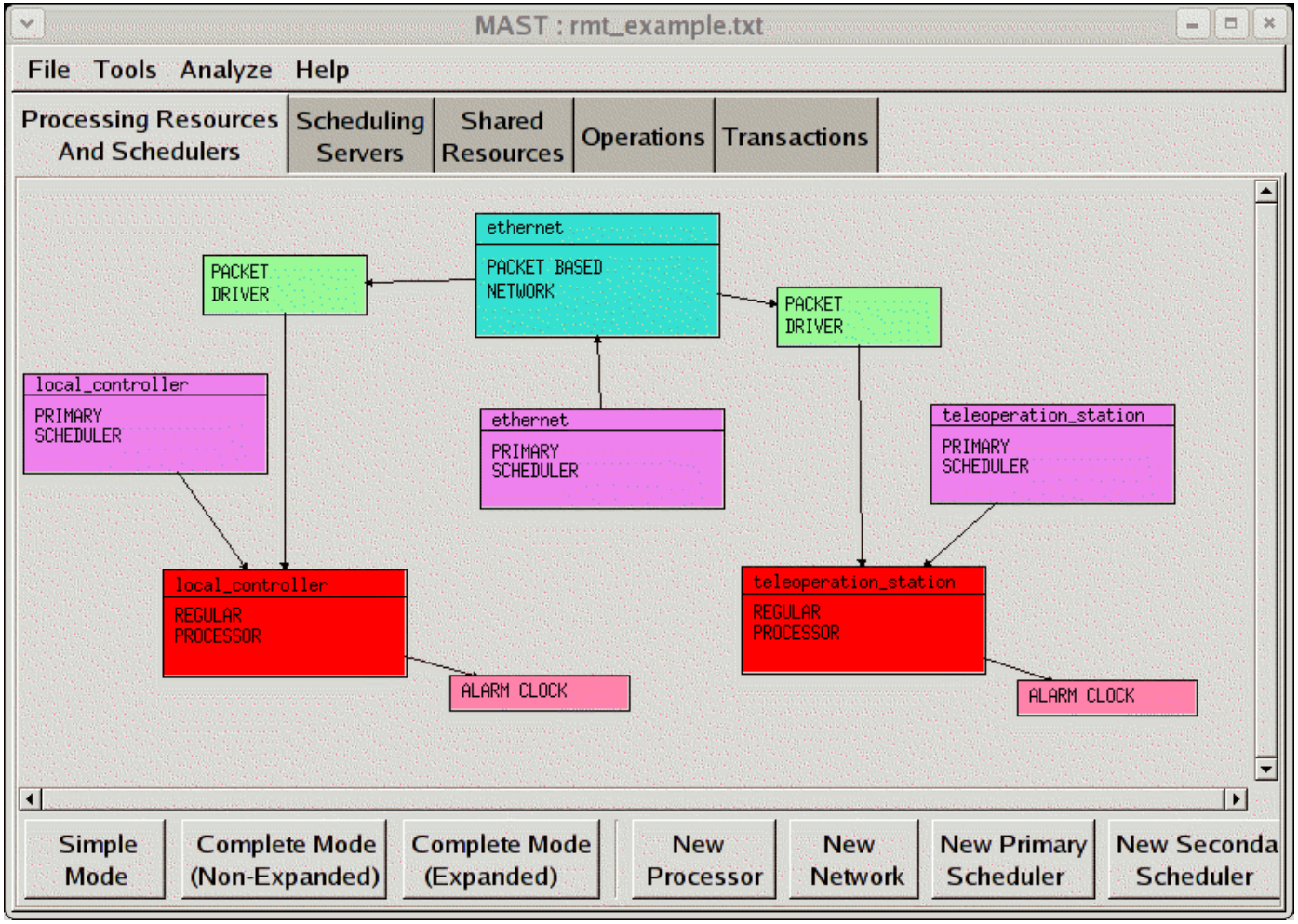
Network drivers (cont'd)



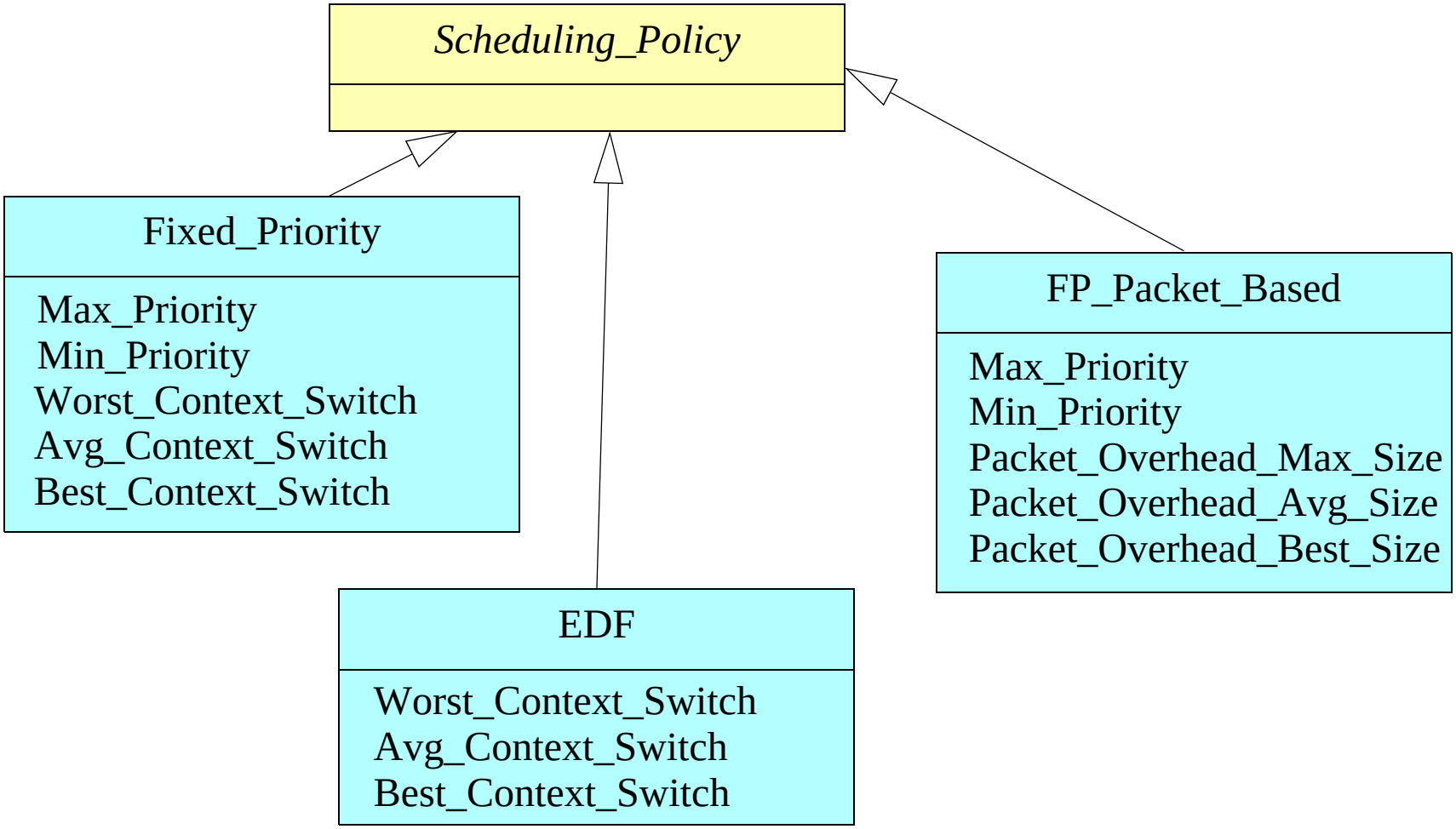
Network drivers (cont'd)



Processing resources, schedulers, drivers, and timers



Scheduling Policies



9.6 Analysis of End-to-End Deadlines

UNIVERSIDAD DE CANTABRIA

The analysis can be done in two different ways:

- **Holistic analysis: use the schedulability analysis for single processor systems, as if all resources were independent:**
  - **Jitter depends on response times**
  - **Response times depend on jitter**
  - **Because the dependency of the response time on jitter is monotonic, we can start with zero jitter and then iterate over the analysis until a stable solution is achieved**
  - **This analysis is pessimistic**
- **Using task offsets:**
  - **More complex analysis**
  - **Much less pessimistic**

Notes:

UNIVERSIDAD DE CANTABRIA

The analysis can be done in two different ways:

- Using the schedulability analysis for single processor systems, as if all resources were independent. The problem with this analysis is that jitter depends on response times, and the response times in turn depend on jitter. Because the dependency of the response time on jitter is monotonic, we can start with zero jitter and then iterate over the analysis until a stable solution is achieved. This analysis is pessimistic, because we are assuming that tasks encounter a critical instant in all CPUs and networks, but this may not be possible in practice because the execution of tasks is not independent.

Using task offsets:

- This is a much more complex analysis, that takes into account the interactions among tasks through the mechanism of considering task offsets. The results of this analysis are much less pessimistic than with the assumption of independent tasks.

Holistic analysis technique

UNIVERSIDAD DE CANTABRIA

Mainly developed at the University of York

Each resource is analyzed separately (CPU's and communication networks):

- **all activations after the first have "*jitter*"**
- ***jitter* in one action is considered equal to the worst-case response time of the previous actions**
- **analysis in one resource affects *response times* in other resources**
- **the analysis is repeated *iteratively*, until a stable solution is achieved**
- **the method converges because of the *monotonic* relation between jitters and response times**

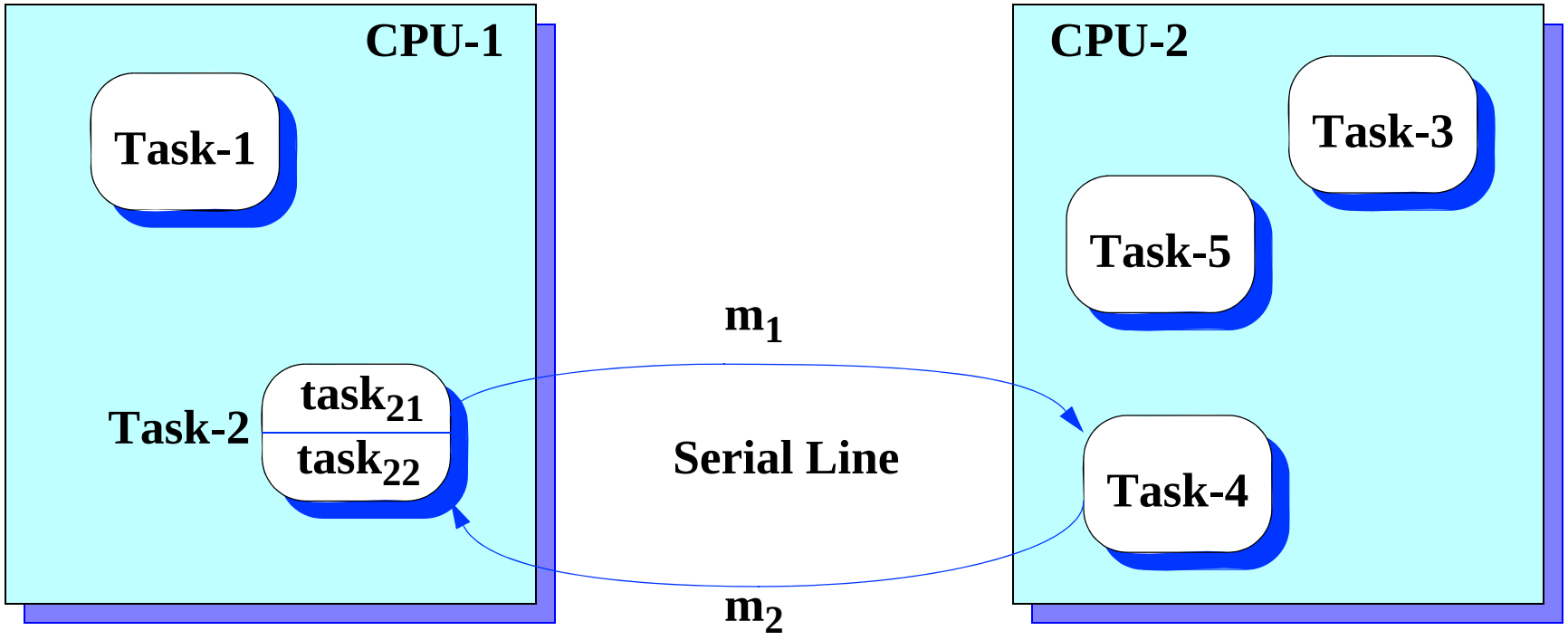
Analysis in the distributed system

```
algorithm WCRT is
begin
  initialize jitter terms to zero
  loop
    calculate worst-case response times;
    calculate new jitters, equal to response
      times of preceding actions
    exit when not schedulable or
      no variation since last iteration;
  end loop;
end WCRT
```

Assumption: $J_i = R_i - R^b_i$, R^b_i = best-case response time=0

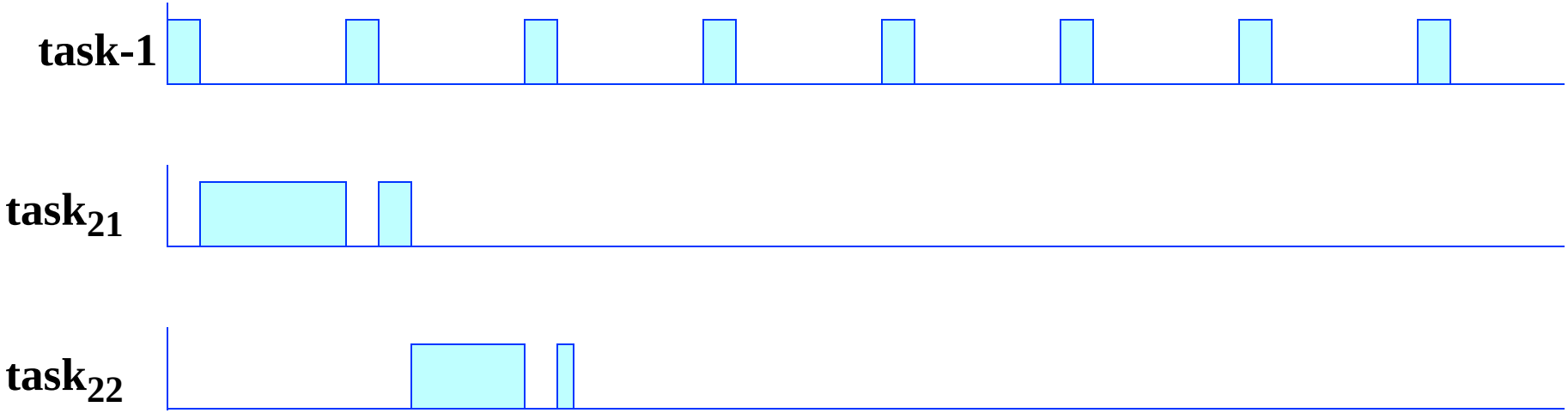
Pessimism in holistic analysis

Holistic analysis technique assumes independent task activations in each resource



Independent task analysis

Execution timeline for $task_{22}$ in previous example:

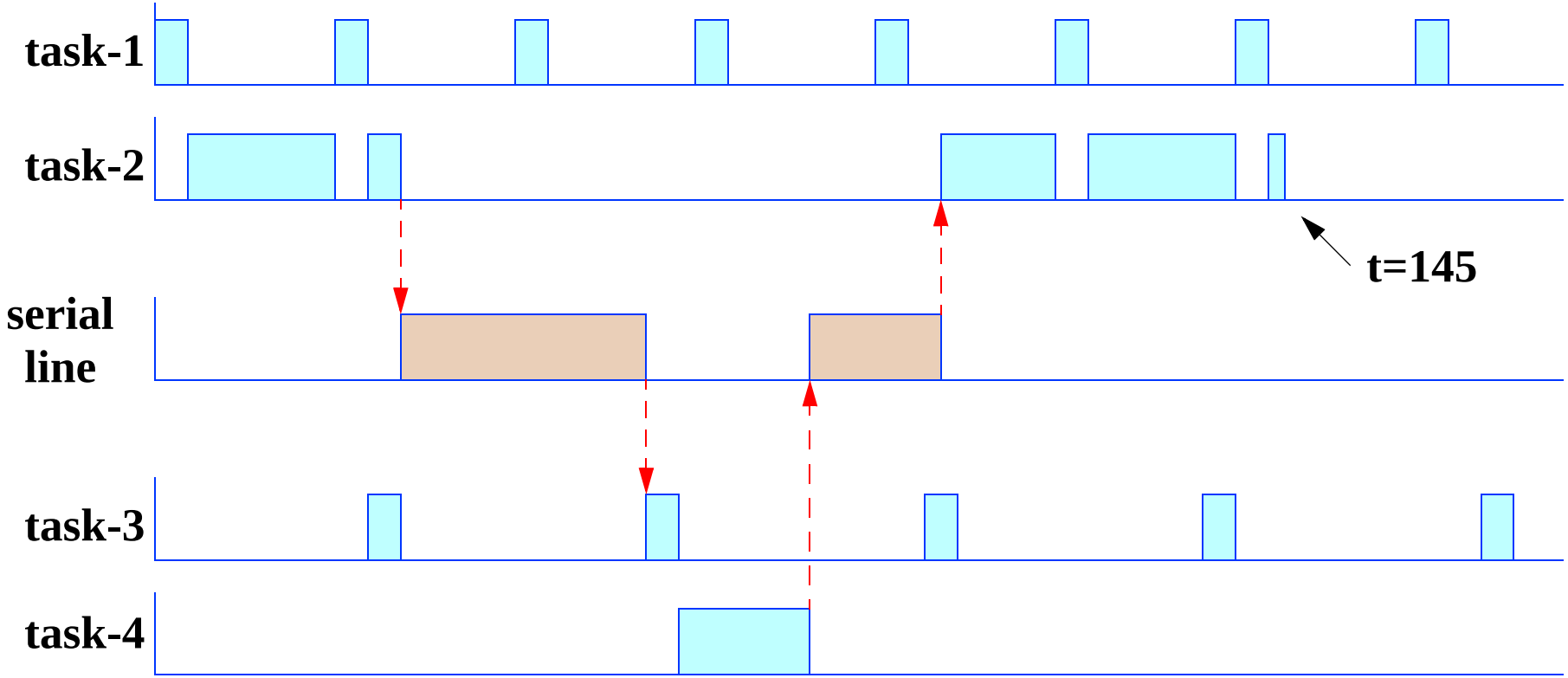


Response time for task-2:

- includes times for $task_{21}$, m_1, task-4, m_2 and $task_{22}$
- total is 270

Using offsets to reduce pessimism

Execution timeline for analysis of task-2:



Objectives of the offset-based techniques

UNIVERSIDAD DE CANTABRIA

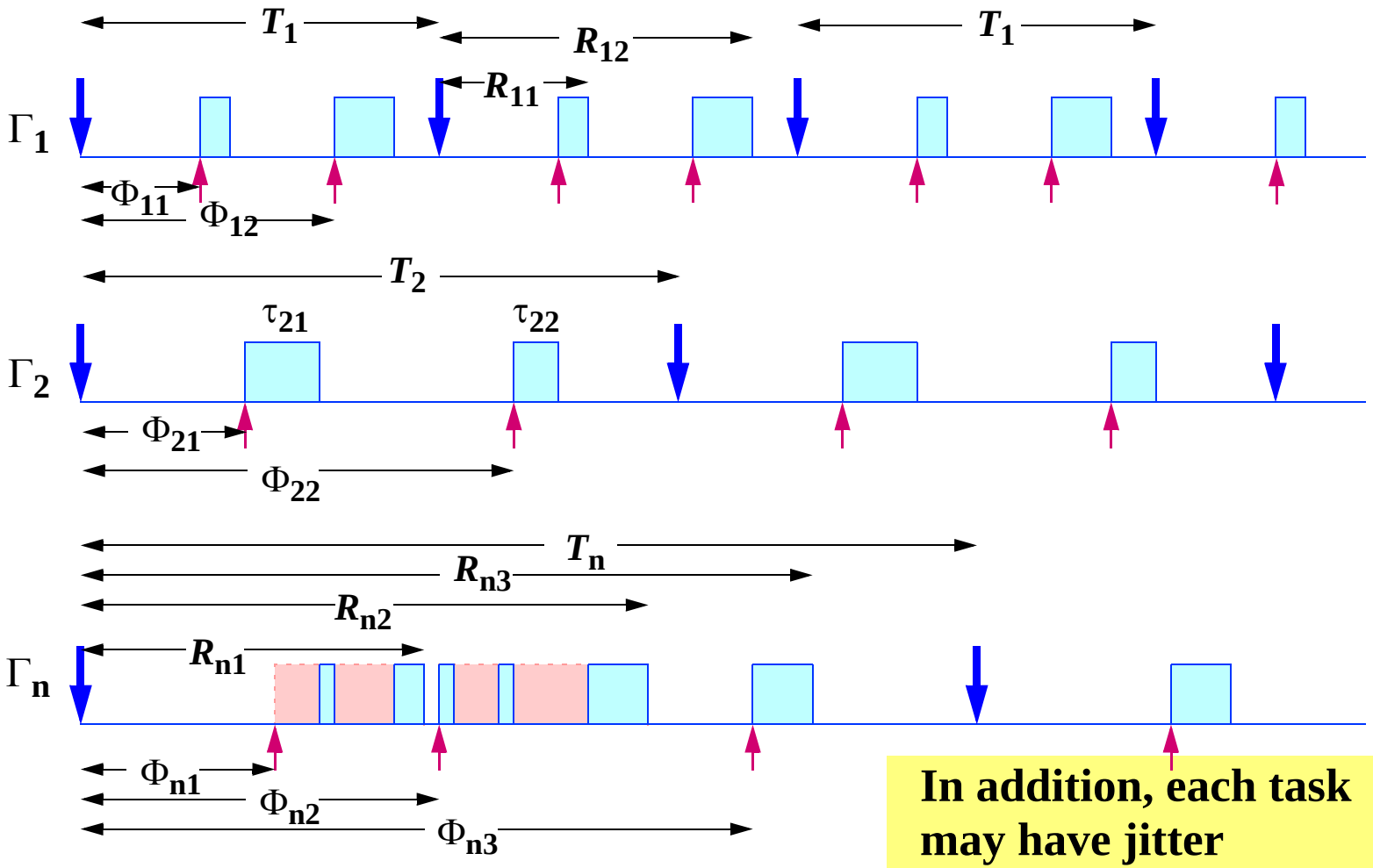
To reduce the pessimism of the worst-case analysis in multiprocessor and distributed systems:

- **by considering offsets in the analysis,**
- **offsets can be larger than task periods;**
  - **this is important if deadlines > task periods**
- **offsets can be static or dynamic:**
  - **offsets are dynamic in distributed systems**
  - **also in tasks that suspend themselves**

This enhancement comes "for free", as there is no change to the application,

- **although better results can be obtained if best-case execution times are measured**

System model with offsets



Analysis with offsets

UNIVERSIDAD DE CANTABRIA

Developed by Tindell, at the University of York:

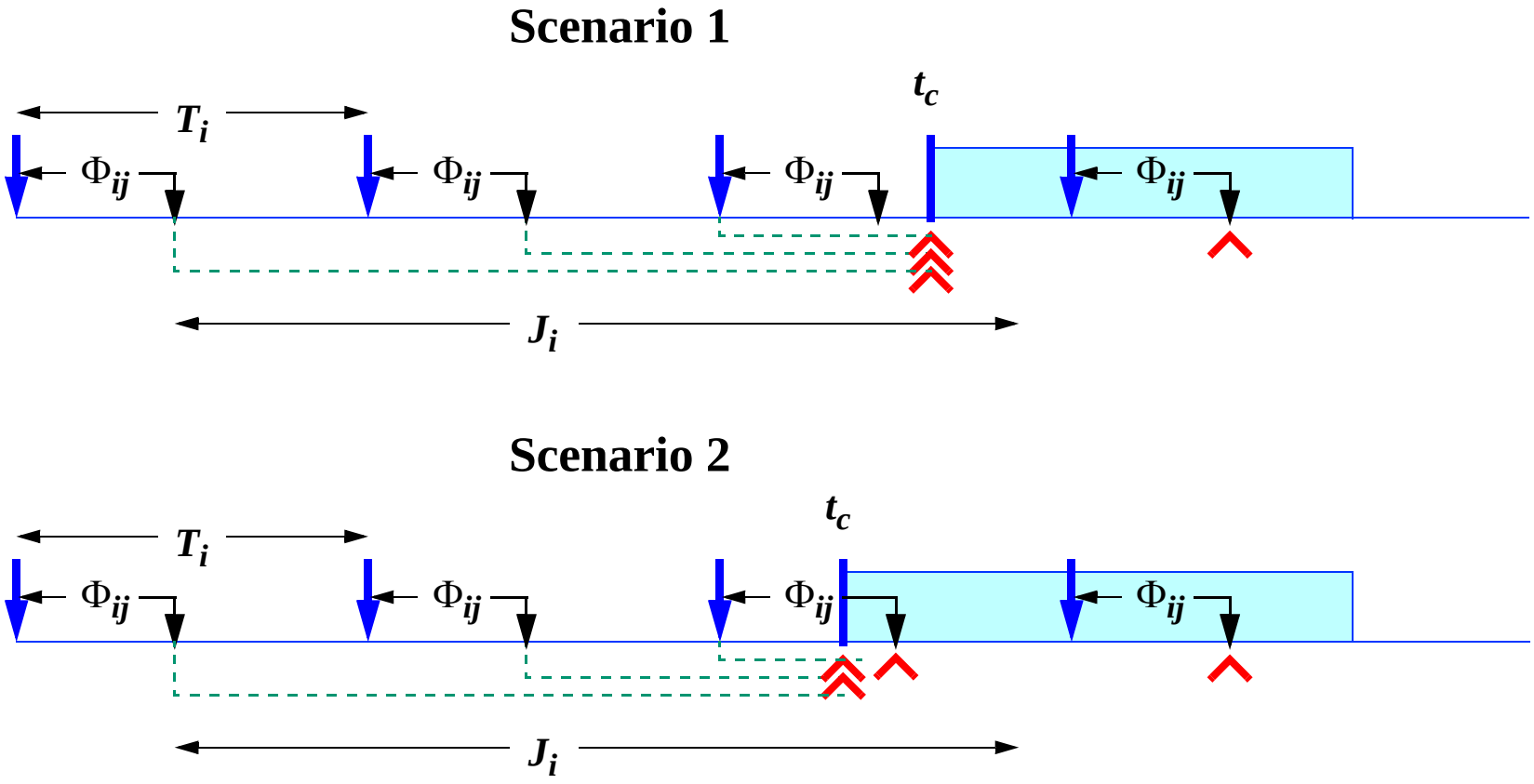
- **The exact analysis is intractable**
- **An upper-bound approximation provides good results (equal to exact analysis in 93% of tested cases)**

Main limitations:

- **Offsets are static:**
  - **not applicable to general distributed transactions**
- **Offsets are less than the task periods:**
  - **for distributed systems, deadlines would need to be smaller than or equal to the task periods**

Extended by Palencia (1998) to address these limitations

Scenarios for calculating the worst-case contribution of τ_{ij}



Exact analysis with static offsets

UNIVERSIDAD DE CANTABRIA

Contribution of task τ_{ij} to the response time of lower priority tasks:

- ***Set 0***: activations that occur before the critical instant and that cannot occur inside the busy period
- ***Set 1***: activations that occur before or at the critical instant, and that may occur inside the busy period
  - **Theorem 1**: worst-case when they all occur at the critical instant
  - **Theorem 2**: worst-case when the first one experienced its maximum jitter
- ***Set 2***:activations that occur after the critical instant
  - **Theorem 1**: worst-case when they have zero jitter

Upper bound approximation for worst-case analysis

UNIVERSIDAD DE CANTABRIA

Exact analysis is intractable:

- The task that generates the worst-case contribution of a given transaction is unknown.
- The analysis has to check all possible combinations of tasks

Tindell developed an upper bound approximation:

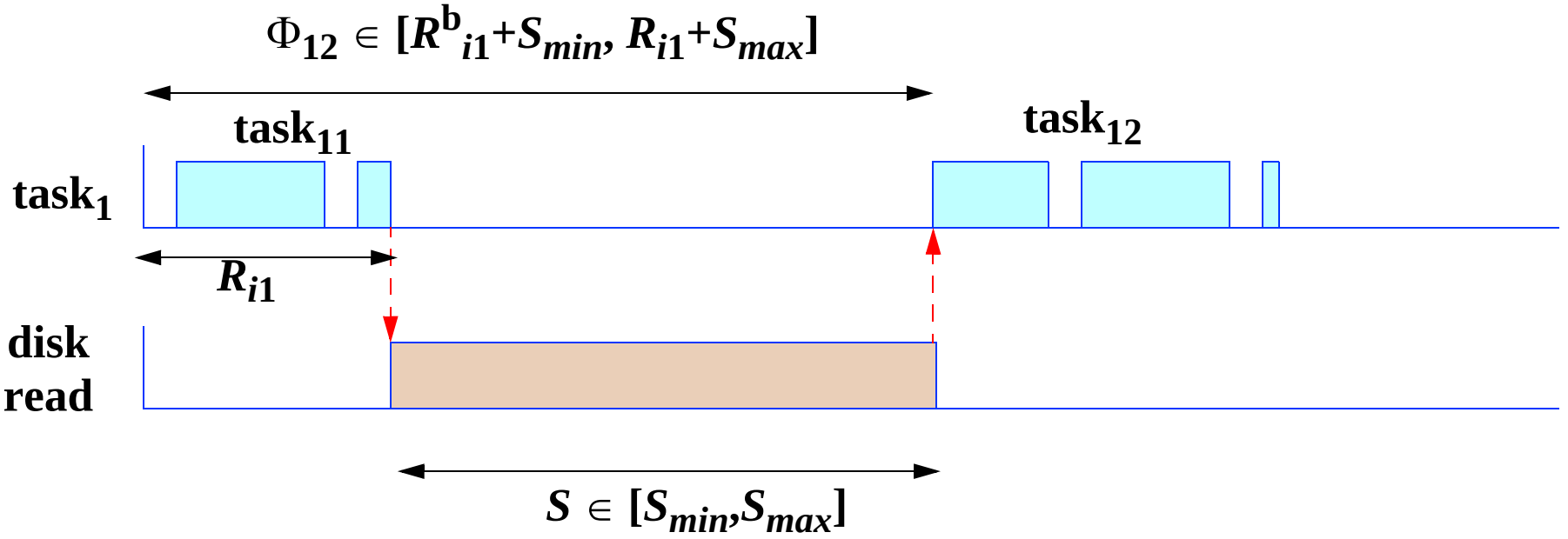
- For each transaction we consider a function that is the maximum of all the worst-case contributions considering each of the tasks of the transaction to be initiating the critical instant
- This technique is pessimistic, but polynomial
- In 93% of the tested cases, the response times were exact

Analysis with dynamic offsets

In many systems the offset may be dynamic:

$$\Phi_{ij} \in [\Phi_{ij,\,min}, \Phi_{ij,\,max}]$$

Example: task with a suspending operation



Analysis with dynamic offsets (cont'd)

UNIVERSIDAD DE CANTABRIA

Dynamic offsets can be modeled with static offsets and jitter:

- **Equivalent offset:**

$$\Phi'_{ij} = \Phi_{ij,min}$$

- **Equivalent jitter:**

$$J'_{ij} = J_{ij} + (\Phi_{ij,max} - \Phi_{ij,min})$$

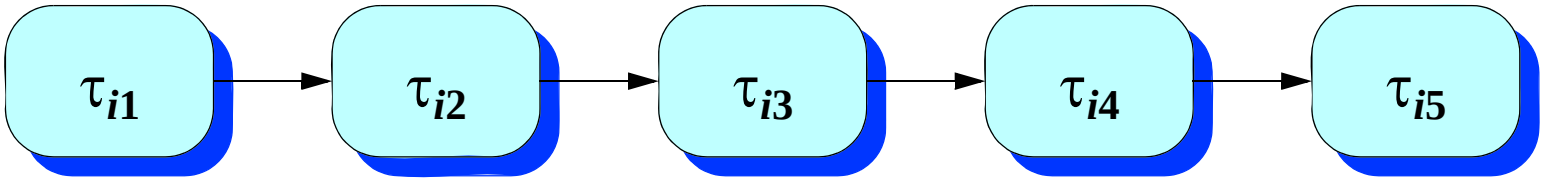
The problem is that now offsets depend on response times, and response times depend on offsets

- **The solution is to apply the analysis iteratively, starting with response times = zero, until a stable solution is achieved**
- **We call this algorithm WCDO**

Analysis of multiprocessor and distributed systems

UNIVERSIDAD DE CANTABRIA

Distributed transaction Γ_i



Dynamic offsets in distributed transactions can also be modeled with static offsets and jitter:

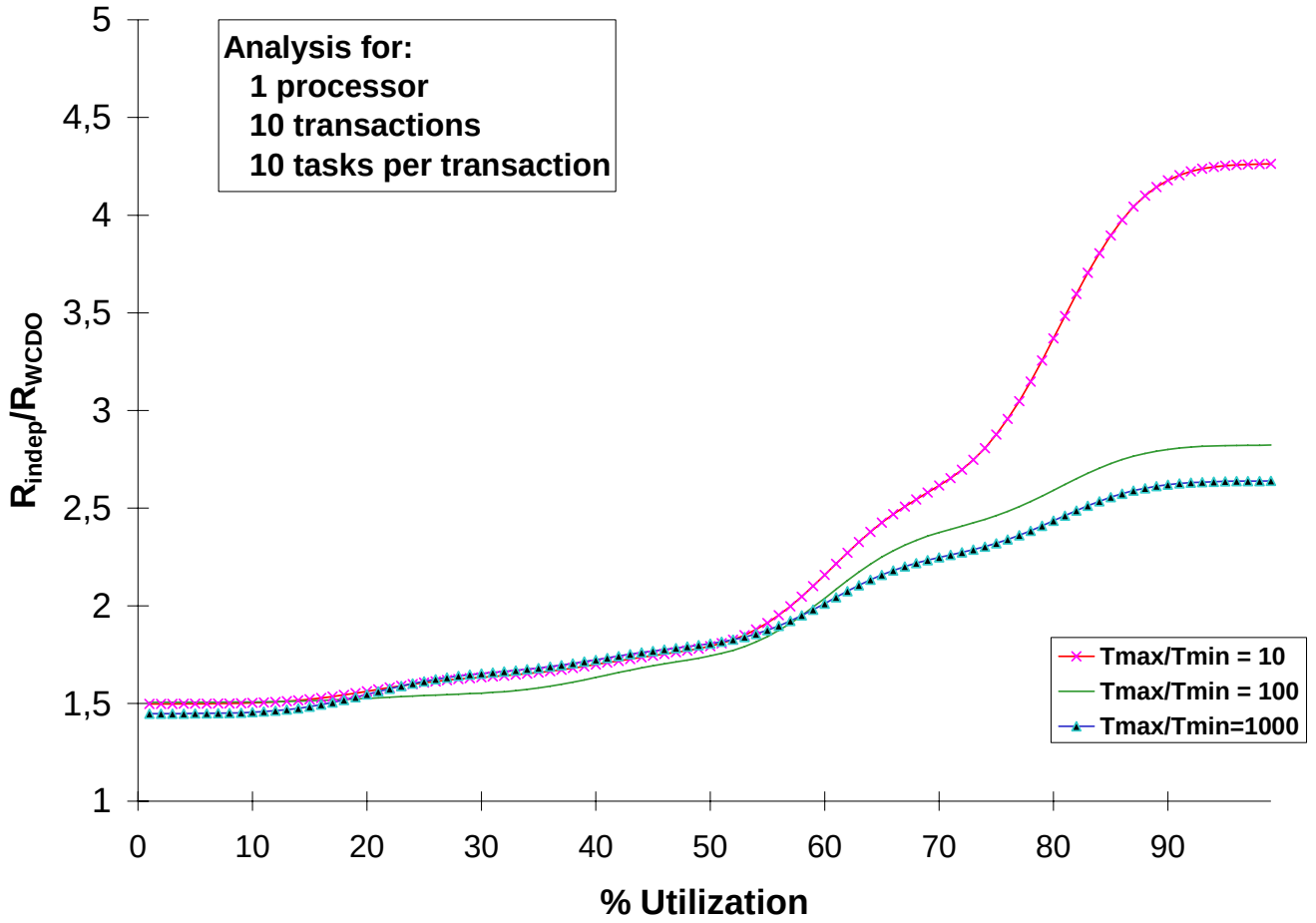
- **Equivalent offset:**

$$\Phi'_{ij} = \Phi_{ij,min} = R^b_{ij-1}$$

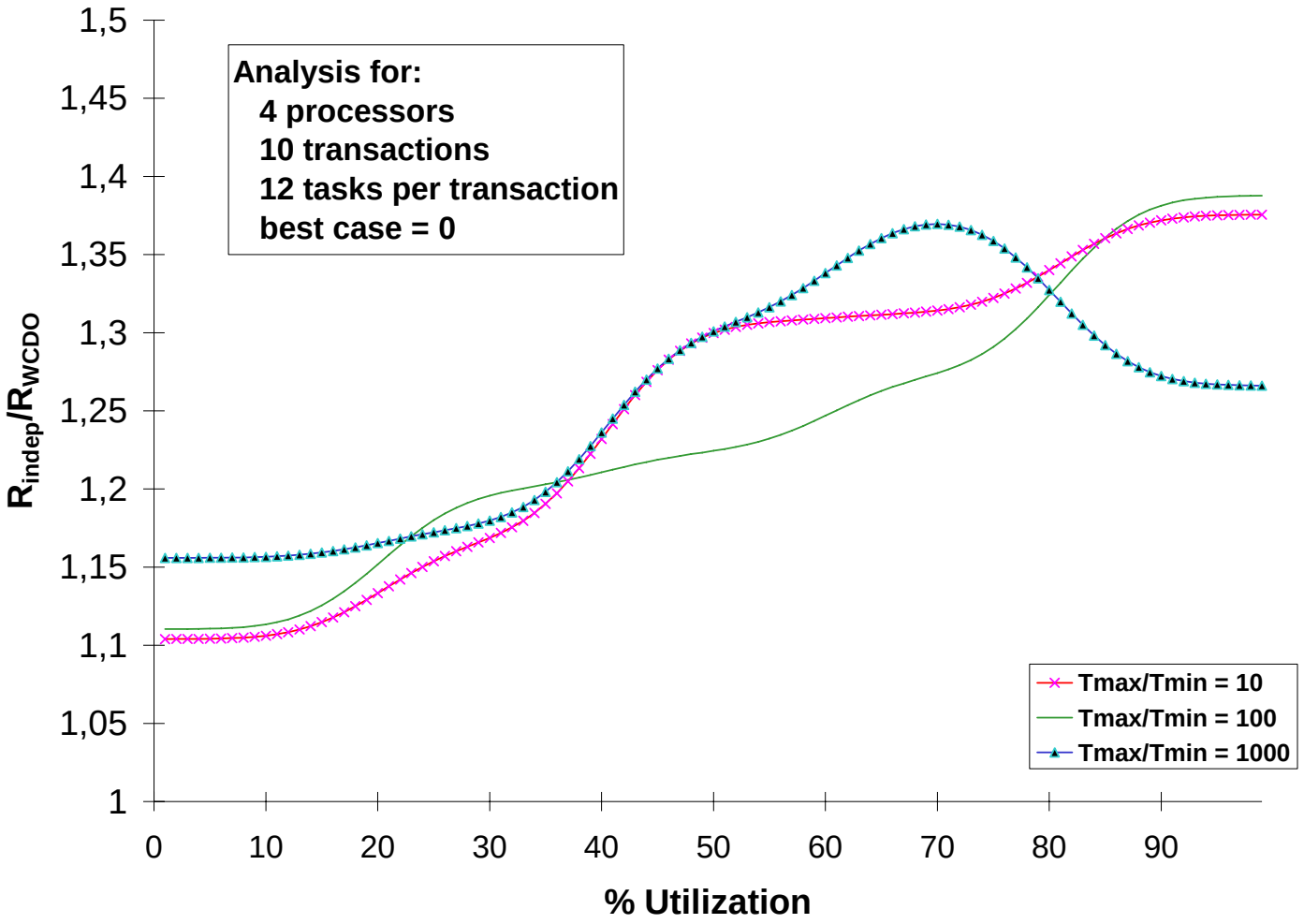
- **Equivalent jitter:**

$$J'_{ij} = J_{ij} + (\Phi_{ij,max} - \Phi_{ij,min}) = R_{ij-1} - R^b_{ij-1}$$

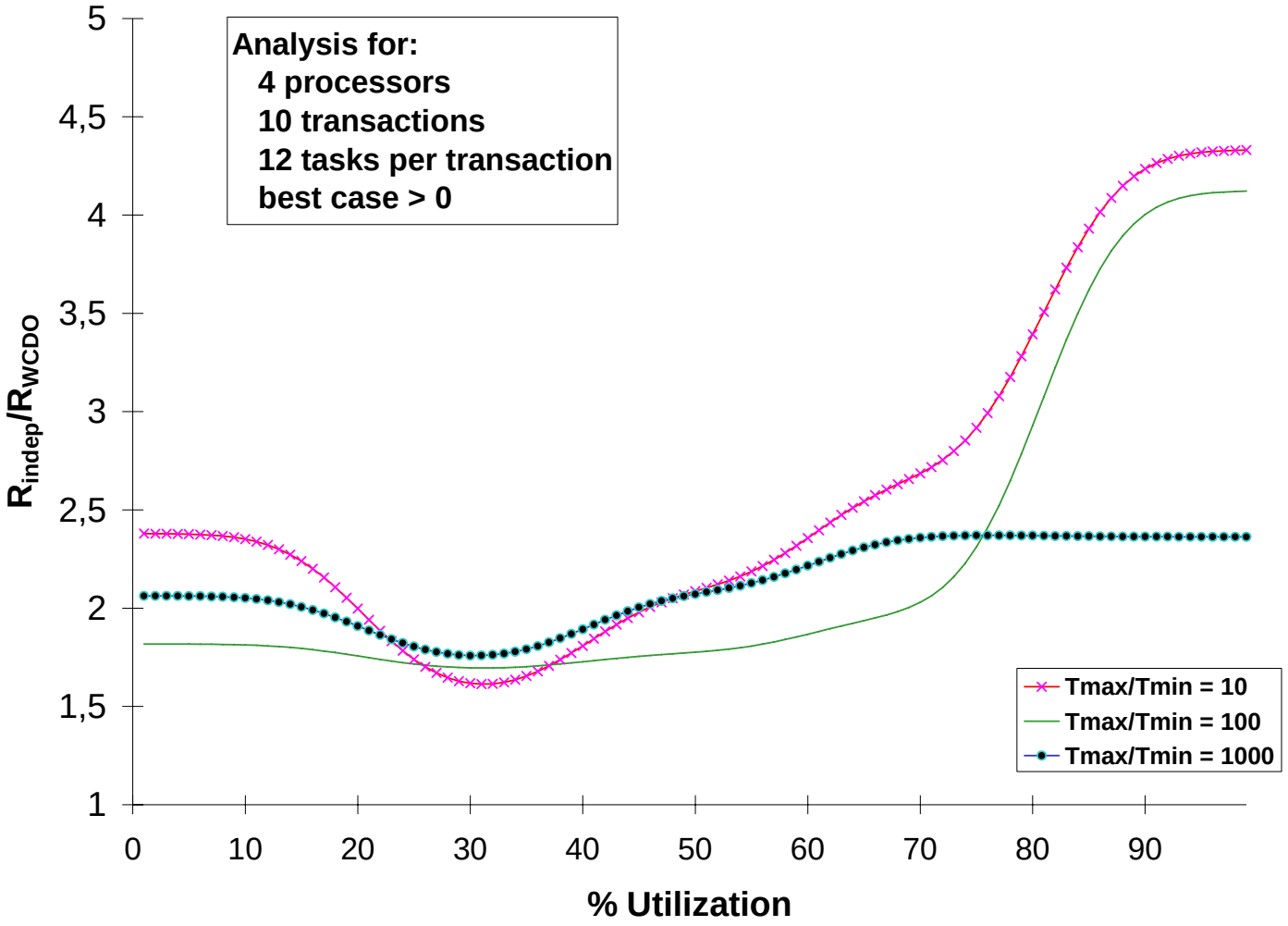
Comparison with holistic analysis: 1 processor



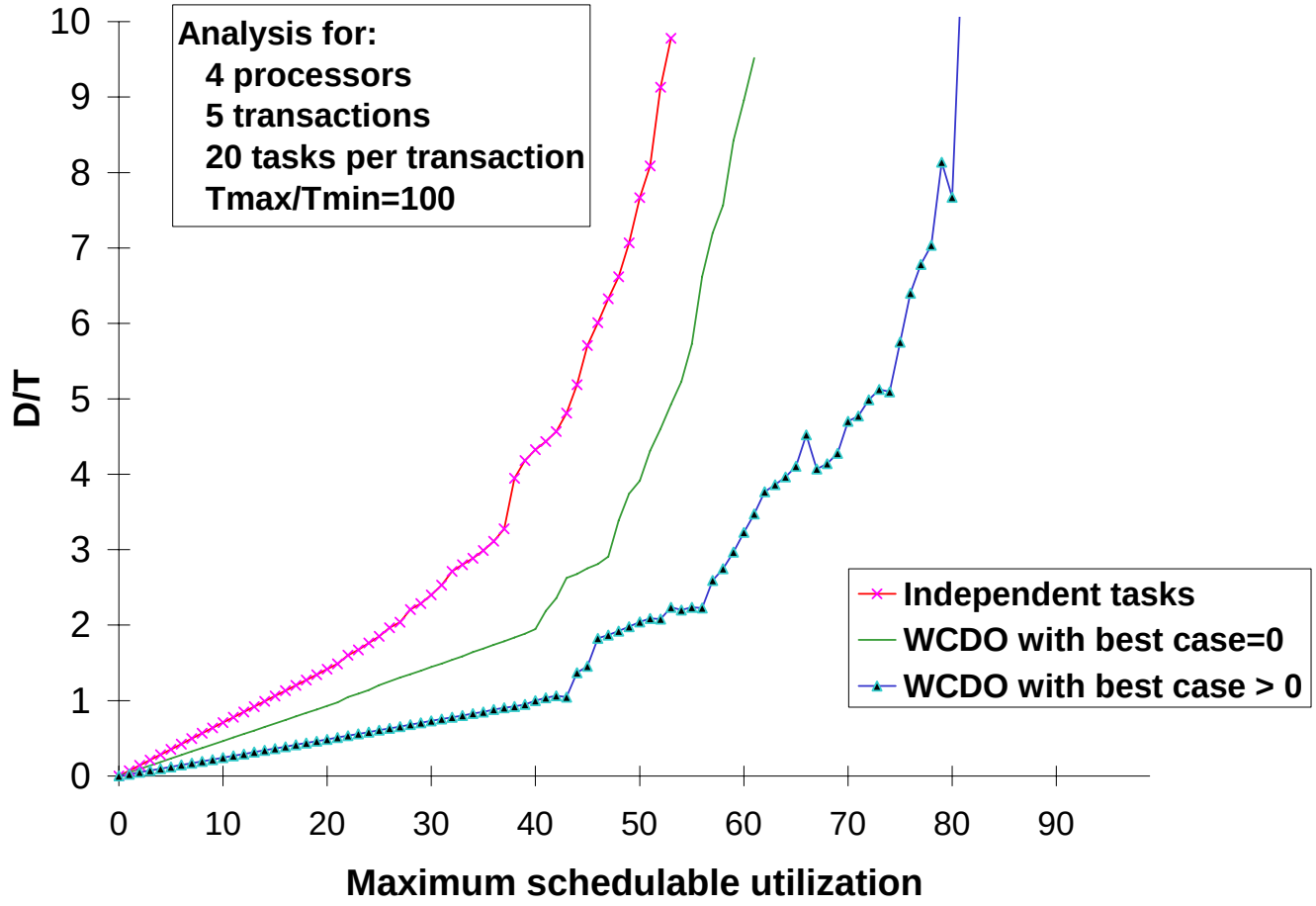
Comparison with four processors, and best-case=0



Comparison with four processors and best-case>0



Maximum utilization with 20 tasks per transaction

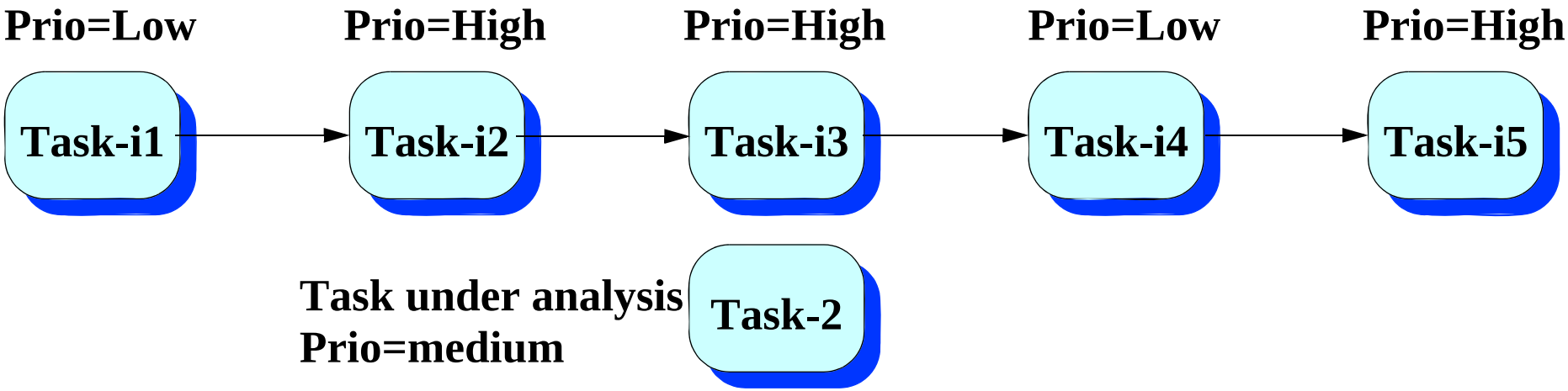


Room for improvement in offset-based analysis

Offset-based analysis produces results that are much less pessimistic than other methods (i.e., holistic analysis)

But offset-based analysis still has room for improvement

- a high priority task that is preceded by a low priority task may not be able to execute before a medium priority task under analysis



Objectives of optimized offset-based analysis

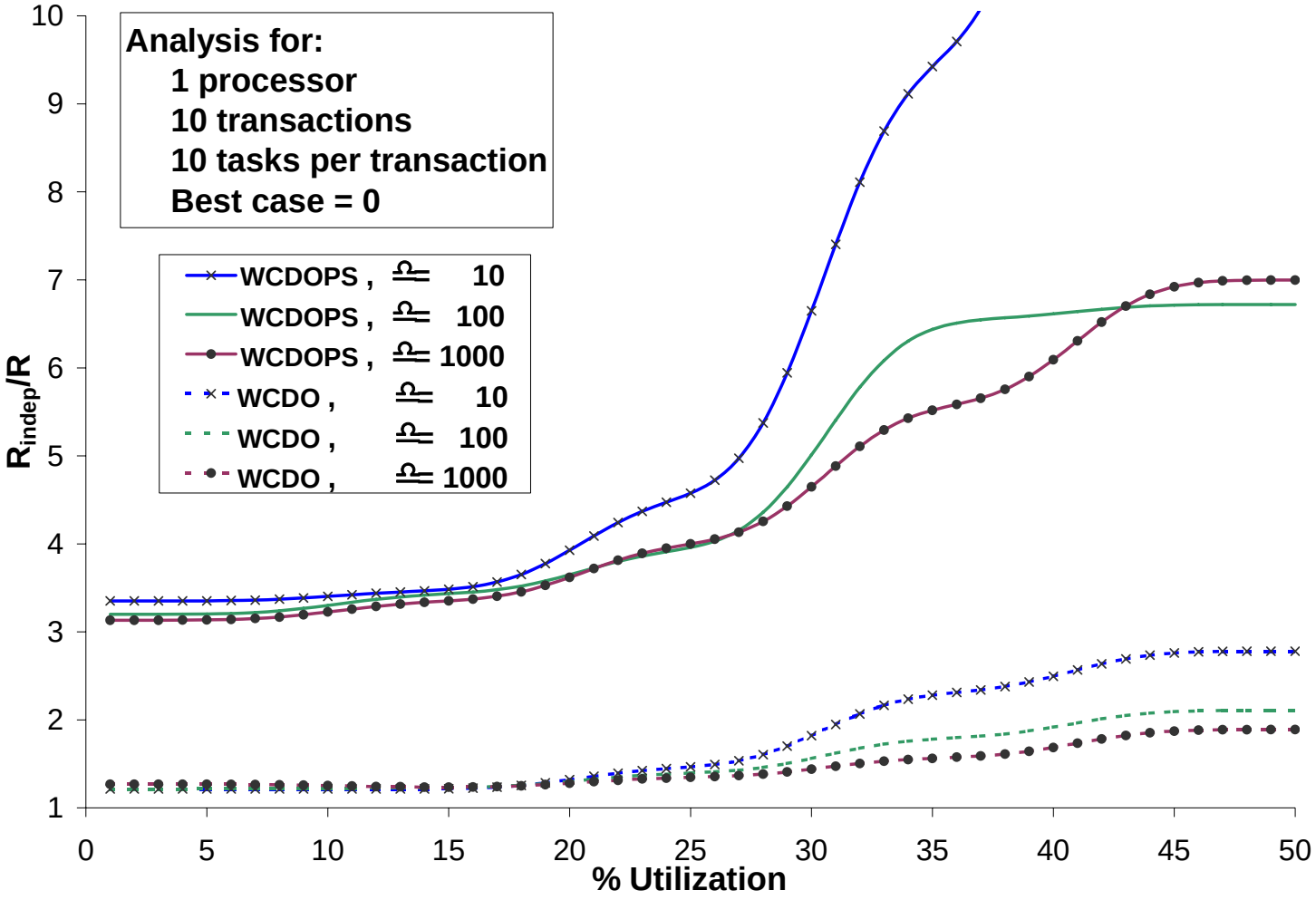
UNIVERSIDAD DE CANTABRIA

To enhance the offset-based schedulability analysis by:

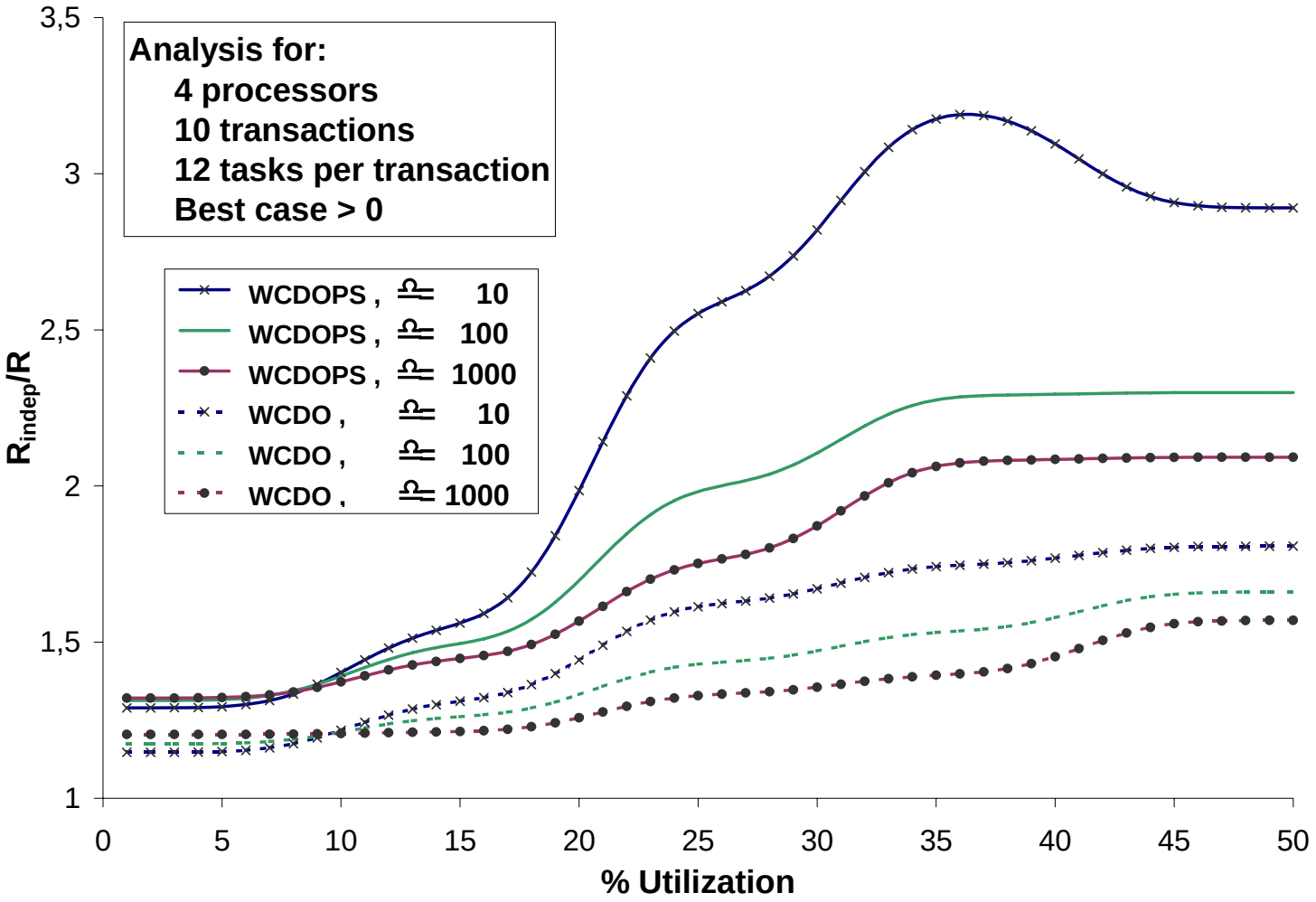
- **Eliminating from the analysis the effects of higher priority tasks that cannot execute due to precedence constraints**
- **Eliminating the effects of the tasks that are preceded by the task under analysis**

These enhancements reduce much of the pessimism in the analysis of distributed systems

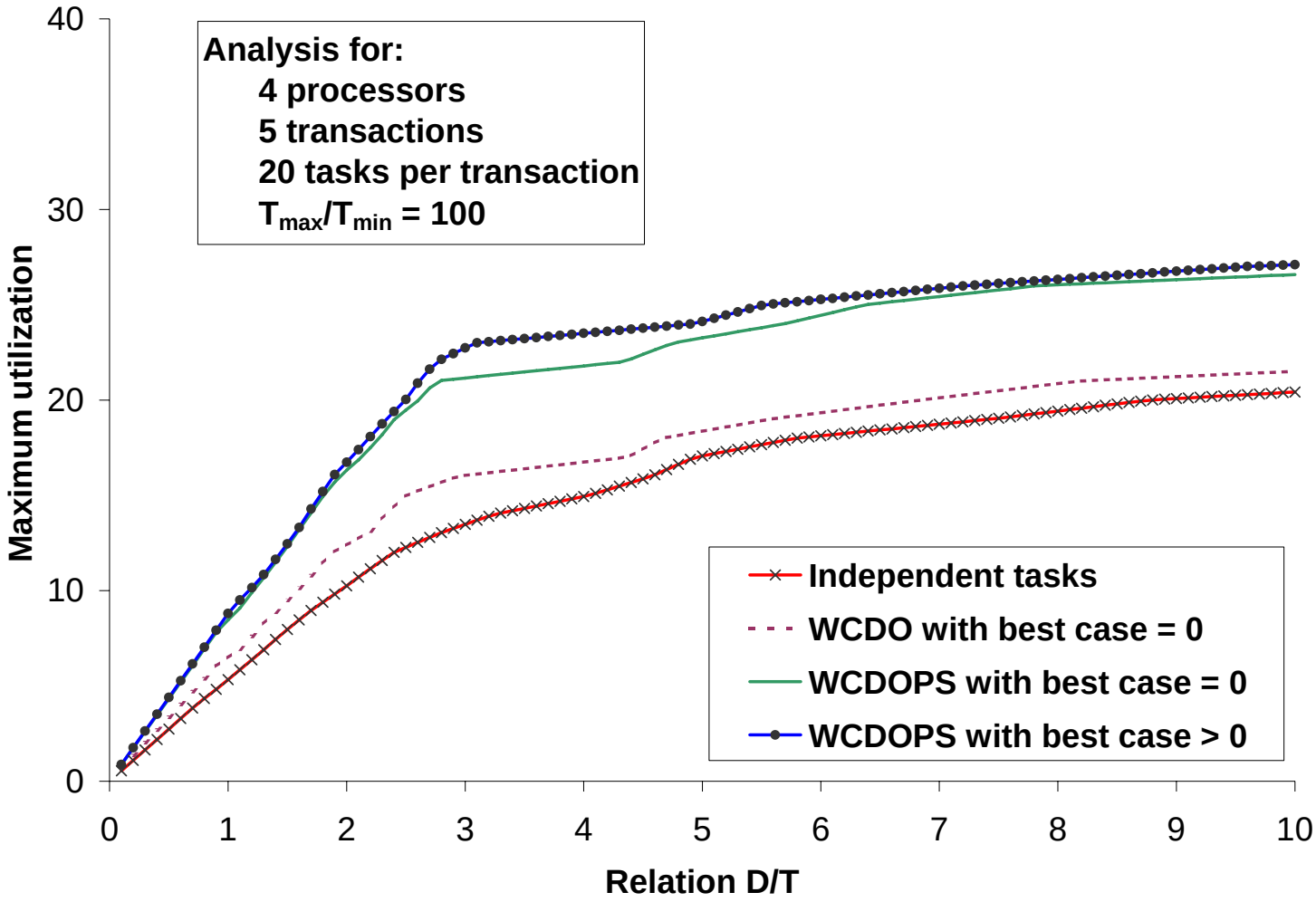
Simulation results: response times



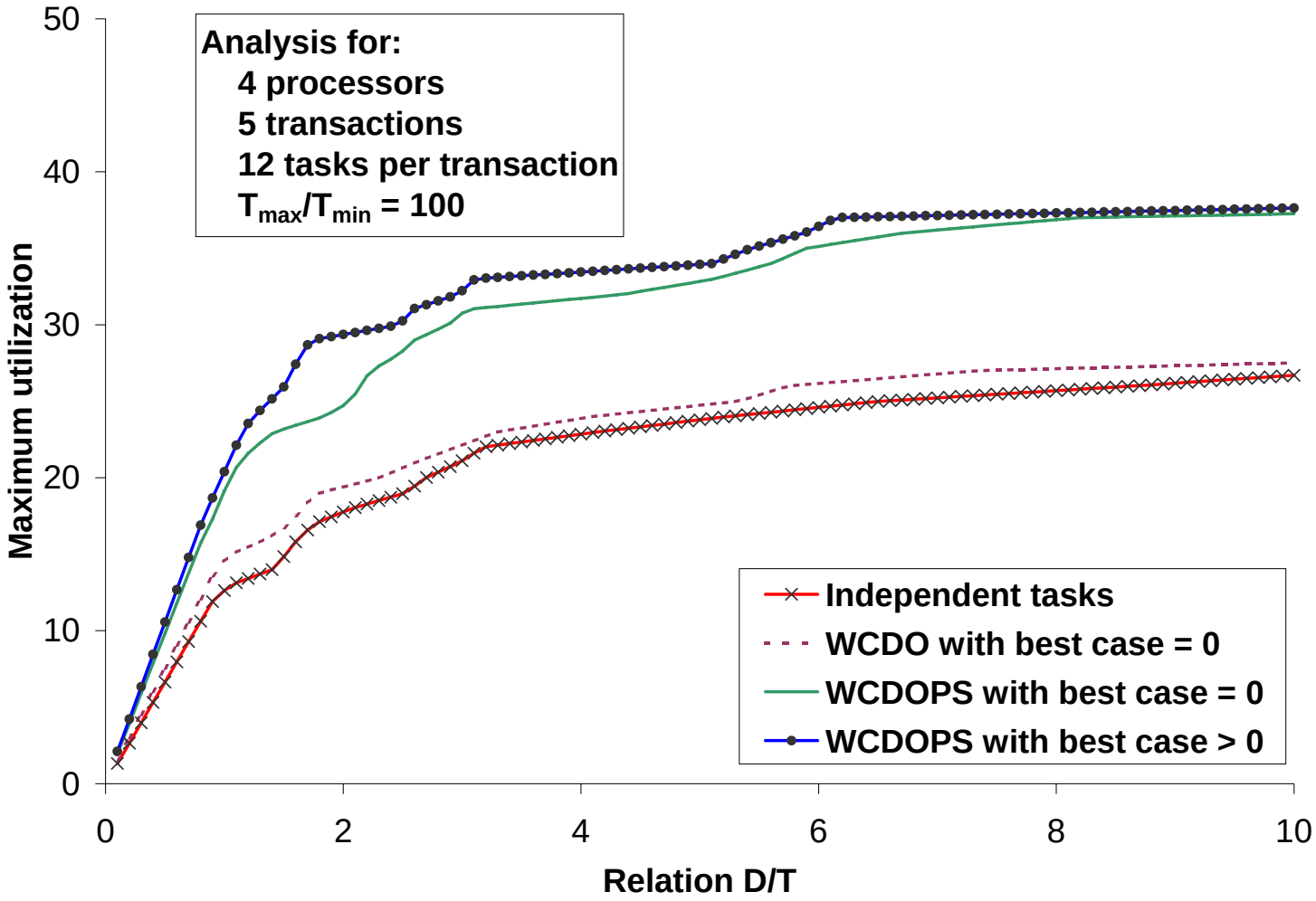
Simulation results: response times



Simulation results: utilization



Simulation results: utilization



Holistic analysis for EDF systems

UNIVERSIDAD DE CANTABRIA

Developed by *Spuri*

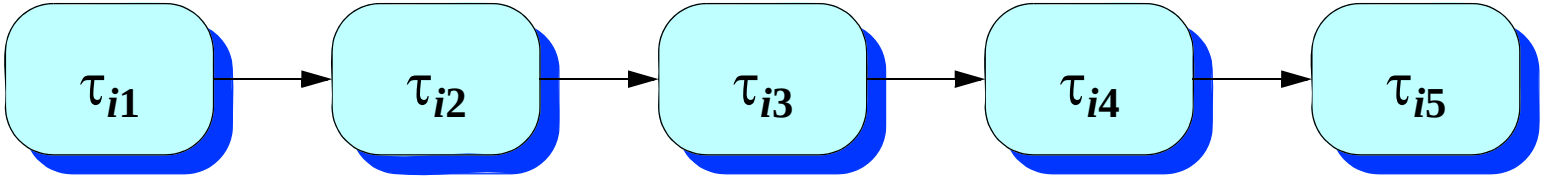
- similar to the holistic analysis developed for fixed priorities at the University of York

Each resource is analyzed separately (CPU's and networks):

- all activations after the first present "*jitter*"
- *jitter* in one action is considered equal to the worst-case response time of the previous actions
- analysis in one resource affects response times in other resources
- the analysis is repeated, until a stable solution is achieved
- the solution is pessimistic

Offset-based analysis for EDF (Palencia, 2003)

Distributed transaction Γ_i



Dynamic offsets in distributed transactions can also be modeled with static offsets and jitter:

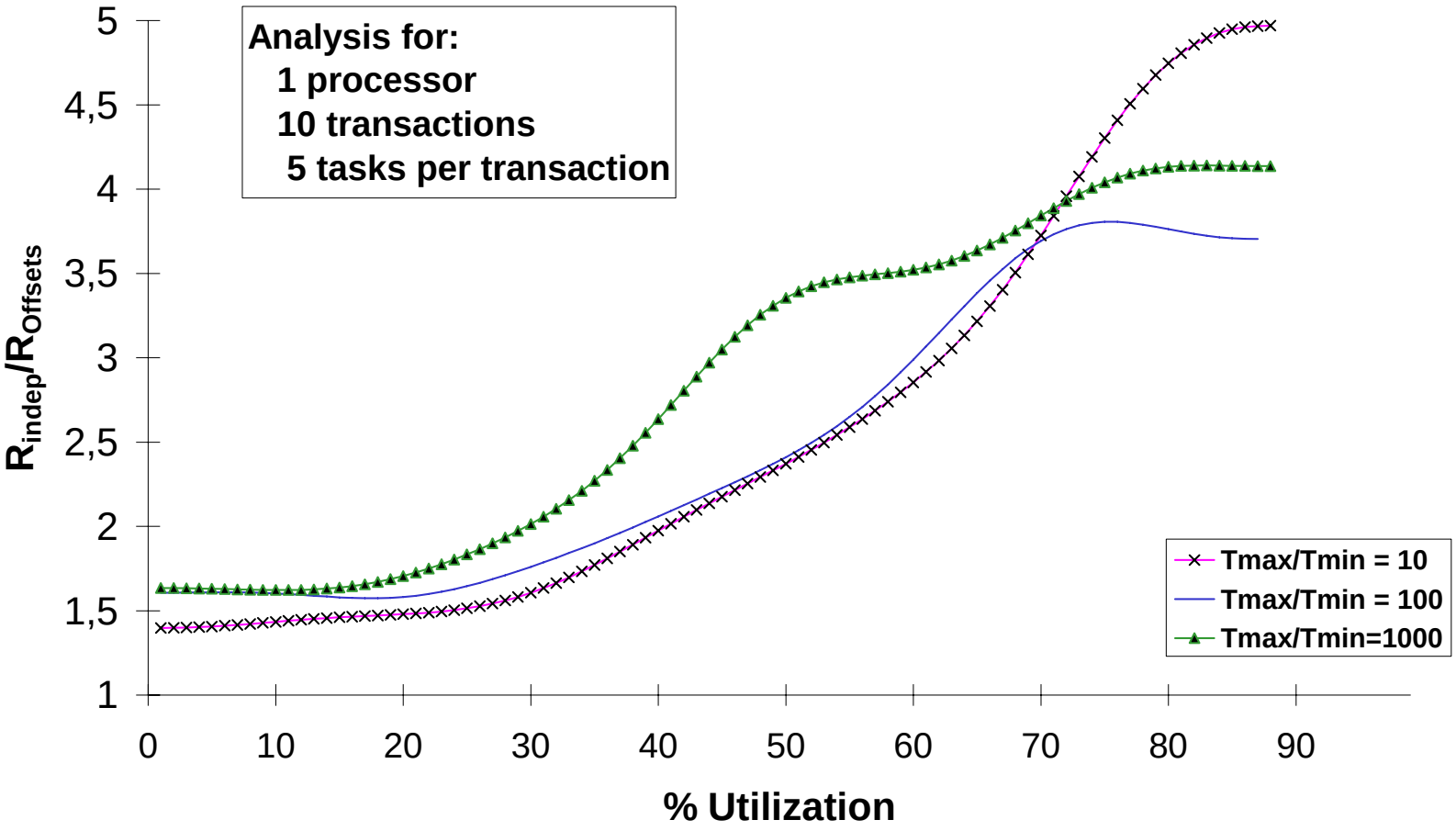
- **Equivalent offset:**

$$\Phi'_{ij} = \Phi_{ij,min} = R^b_{ij-1}$$

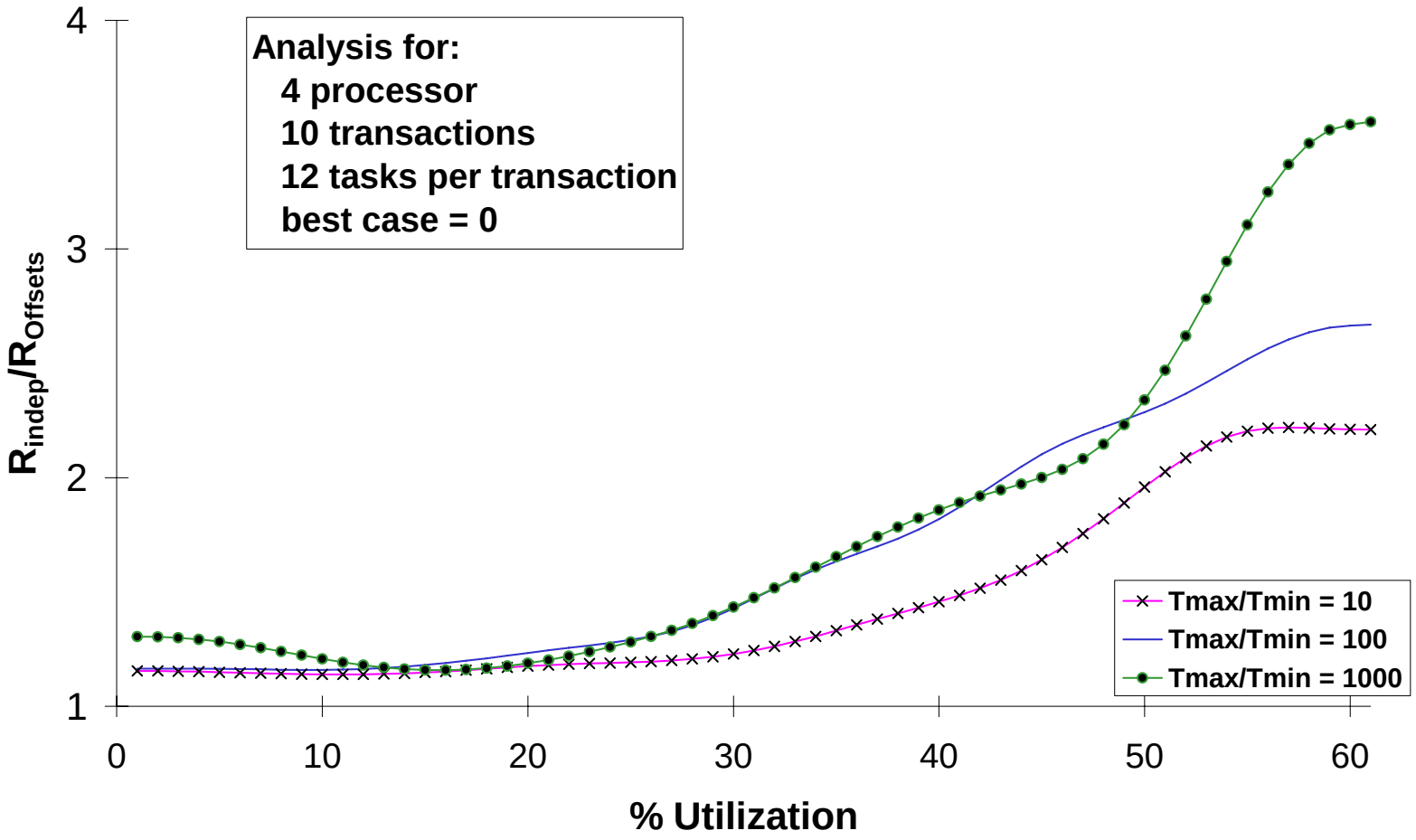
- **Equivalent jitter:**

$$J'_{ij} = J_{ij} + (\Phi_{ij,max} - \Phi_{ij,min}) = R_{ij-1} - R^b_{ij-1}$$

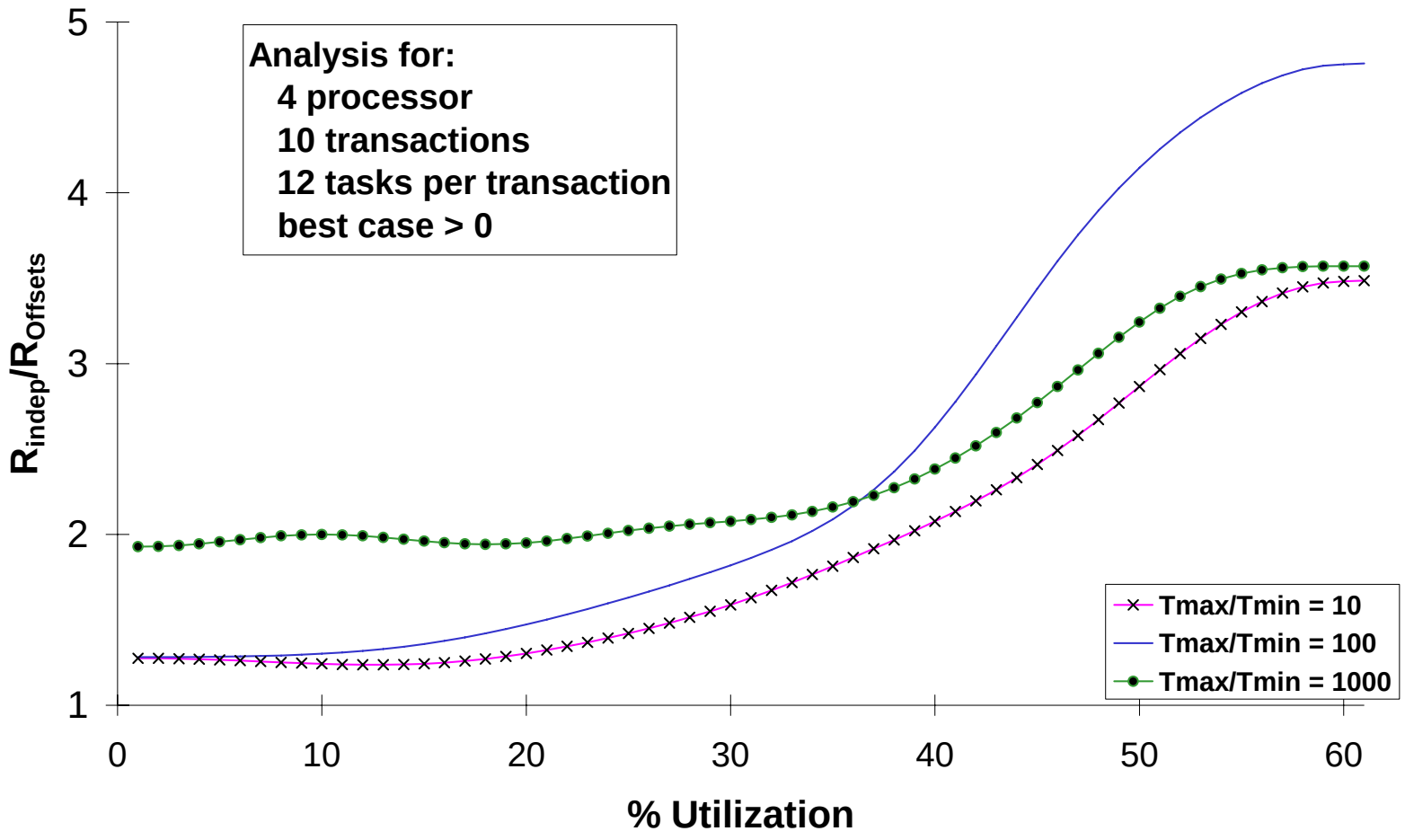
Comparison with holistic analysis: 1 processor



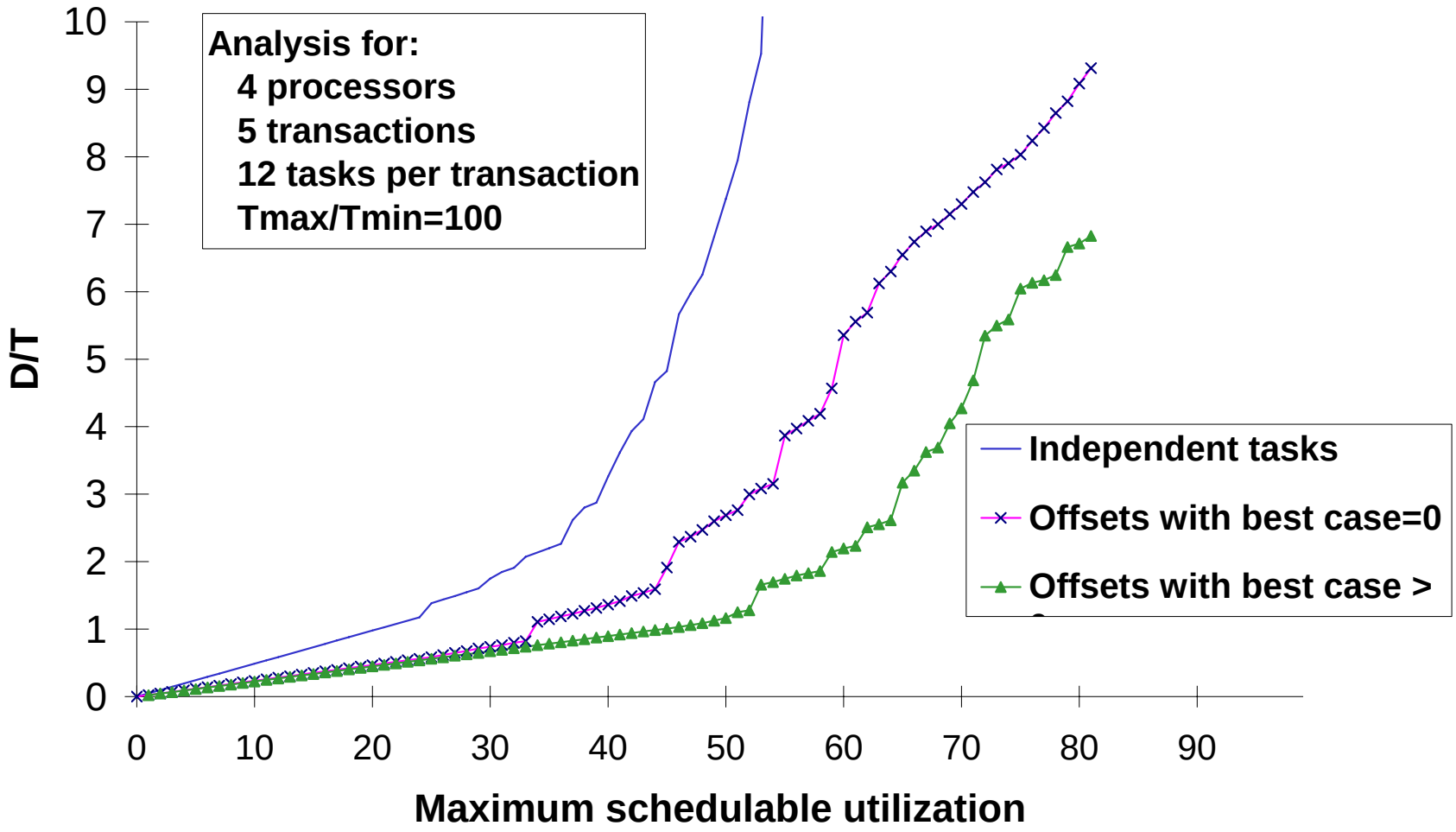
Comparison with four processors, and best-case=0



Comparison with four processors and best-case>0



Maximum utilization with 12 tasks per transaction



Summary of Analysis Techniques

Kind of Analysis	Deadlines	Number of processors	Fixed Priorities	EDF
Utilization test	D=T	1	☑	☑
Response Time Analysis	Arbitrary	1	☑	☑
Holistic Analysis	Arbitrary	Many	☑	☑
Offset-Based Analysis	Arbitrary	Many	☑	☑

Distributed model also applicable to:

- signal & wait synchronization
- activities that suspend themselves (i.e., delays, I/O, ...)

Priority assignment techniques

UNIVERSIDAD DE CANTABRIA

No known optimum priority assignment for distributed systems

Simulated Annealing

- Standard optimization technique
- Finds solutions by successively exchanging the priorities of a pair of tasks, and reapplying the analysis to determine if results are worse or better
- The probability of the change surviving is a function of the results
- Does not guarantee finding the solution

Priority assignment techniques (cont'd)

UNIVERSIDAD DE CANTABRIA

HOPA

- Heuristic algorithm based on successively applying the analysis
- Much faster than simulated annealing
- Usually finds better solutions

None of them guarantees finding the solution

Fixed Priority Assignment Techniques in MAST

Technique	Single-Processor	Multi-Processor
Monoprocessor	☑	
HOPA	☑	☑
Simulated Annealing	☑	☑

Monoprocessor:

- If deadlines within periods, DM
- Otherwise, Audsley's algorithm: iteratively apply analysis, successively ordering tasks by priority: $O(n^2)$ times the analysis

9.7 Handling Task Suspension

UNIVERSIDAD DE CANTABRIA

In many systems a task suspends itself waiting for a response from a remote resource.

Analysis of the suspending task: we can treat the suspension interval in two different (pessimistic) ways:

- treat the suspension time as execution time,
- or, treat each portion of the response as a different event.

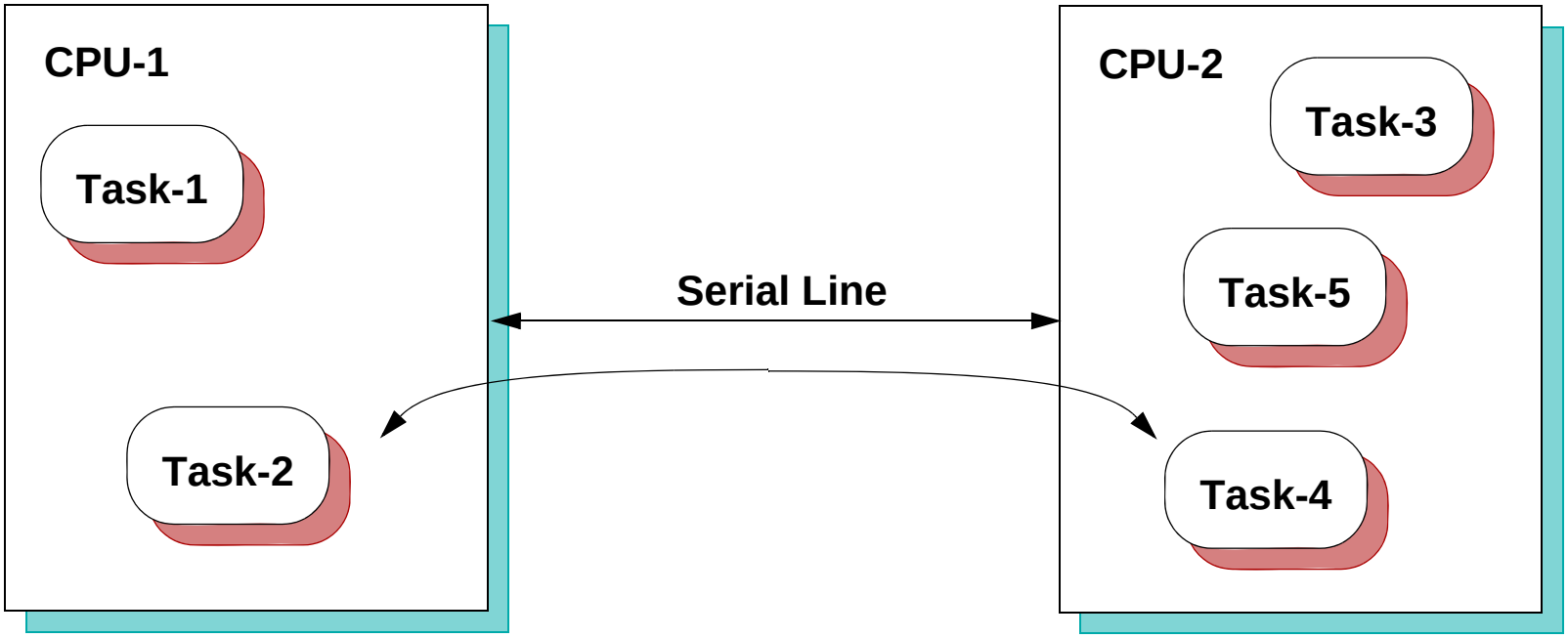
Analysis of lower priority tasks: treat each portion of the response as a different event, but:

- include the jitter effect of the second portion, or eliminate jitter.

Analysis with offsets can also be used for suspending tasks

- less pessimism introduced

Example of a Suspending Task:



Task parameters

UNIVERSIDAD DE CANTABRIA

Tasks 1, 2, 3, and 5 are periodic, and task 4 is a server task

Task	C_i	T_i	D_i	P_i
1	4	20	20	High
2	20+30	150	150	Low
3	5	30	30	High
4	15	-	-	Medium
5	100	200	200	Low

The transmission times in the serial line: 25 and 34 ms., respectively for the request and the reply message.

The end-to-end deadline for task-2 is 150 ms

Solution with Suspending Task

Analysis in CPU-1:

- treat suspension as execution time:

$$E_2 = \left\lceil \frac{a_k}{T_1} \right\rceil C_1 + C_{21} + M_1 + E_4 + M_2 + C_{22} = 165ms$$

- treat each portion as a different task:

$$E_{21} = \left\lceil \frac{a_k}{T_1} \right\rceil C_1 + C_{21}$$

$$E_{22} = \left\lceil \frac{a_k}{T_1} \right\rceil C_1 + C_{22}$$

$$E_2 = E_{21} + M_1 + E_4 + M_2 + E_{22} = 145ms$$

Notes:

UNIVERSIDAD DE CANTABRIA

The response time for task-1 is equal to C_1=4 ms.

For the response time of task-2 we must take into account the suspension time, which is the transmission time of the request message (M_1), plus the response time to task-4 (E_4), and plus the transmission time for the reply message (M_2). M_1 and M_2 are 25 and 34 ms. respectively, because the serial line is not used by any other task; otherwise a complete analysis should be performed for the communications line. E_4 is 20 ms.; this value was obtained from the analysis of CPU-2, which is shown in the following slide.

The response time for task-2 can be determined using two different approaches:

- Treating the suspension time as execution time: in this case the result is 165 ms.
- Treating each portion of task-2 as a different task, and adding the two response times plus the suspension time: in this case the result is 145 ms.

Both results may be pessimistic, i.e., they represent upper bounds to the worst-case execution time. Therefore, we can take the smallest (145 ms.) as the worst-case response time.

Solution (continued)

Analysis in CPU-2:

$$E_3 = C_3 = 5ms$$

$$E_4 = \left\lceil \frac{a_k}{T_3} \right\rceil C_3 + C_4 = 20ms$$

$$E_5 = \left\lceil \frac{a_k}{T_3} \right\rceil C_3 + \left\lceil \frac{a_k + E_{21} + M_1}{T_4} \right\rceil C_4 + C_5 = 160ms$$

The term $E_{21}+M_1$ incorporates the effect of jitter in the activation of task-4.

Notes:

The analysis of CPU-2 is conventional, except that task-4 is activated with jitter. This jitter has a schedulability penalty on lower priority tasks (i.e., on task-5). The amount of jitter that the activation of task-4 suffers is the variability of the first portion of task-2, and the request message. Supposing that both could sometimes be very short (near to zero) and in other occasions the worst case could happen (E_{21}=28 ms. and M_1=25 ms.), the total jitter could be 53 ms.

Notice that if we had not included the jitter effect in the equations, the result for task 5 would have been 140 ms., less that the actual worst-case that is 160 ms.

Analysis with offsets

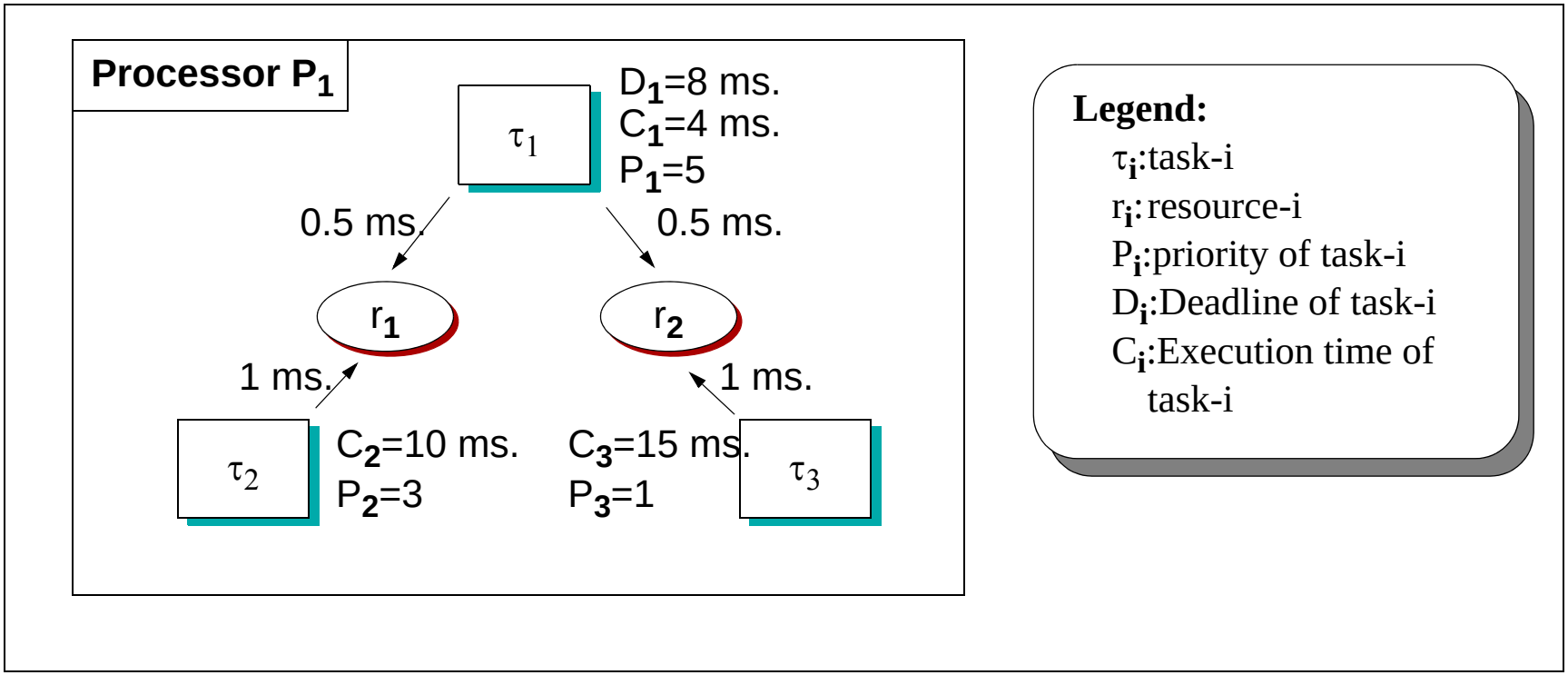
UNIVERSIDAD DE CANTABRIA

If we apply the analysis with offsets we get the following results:

Task	D_i	R_i
1	20	4
2	150	145
3	30	5
4	-	-
5	200	140

9.8 Multiprocessor Synchronization

Example: synchronization in a single processor



Notes:

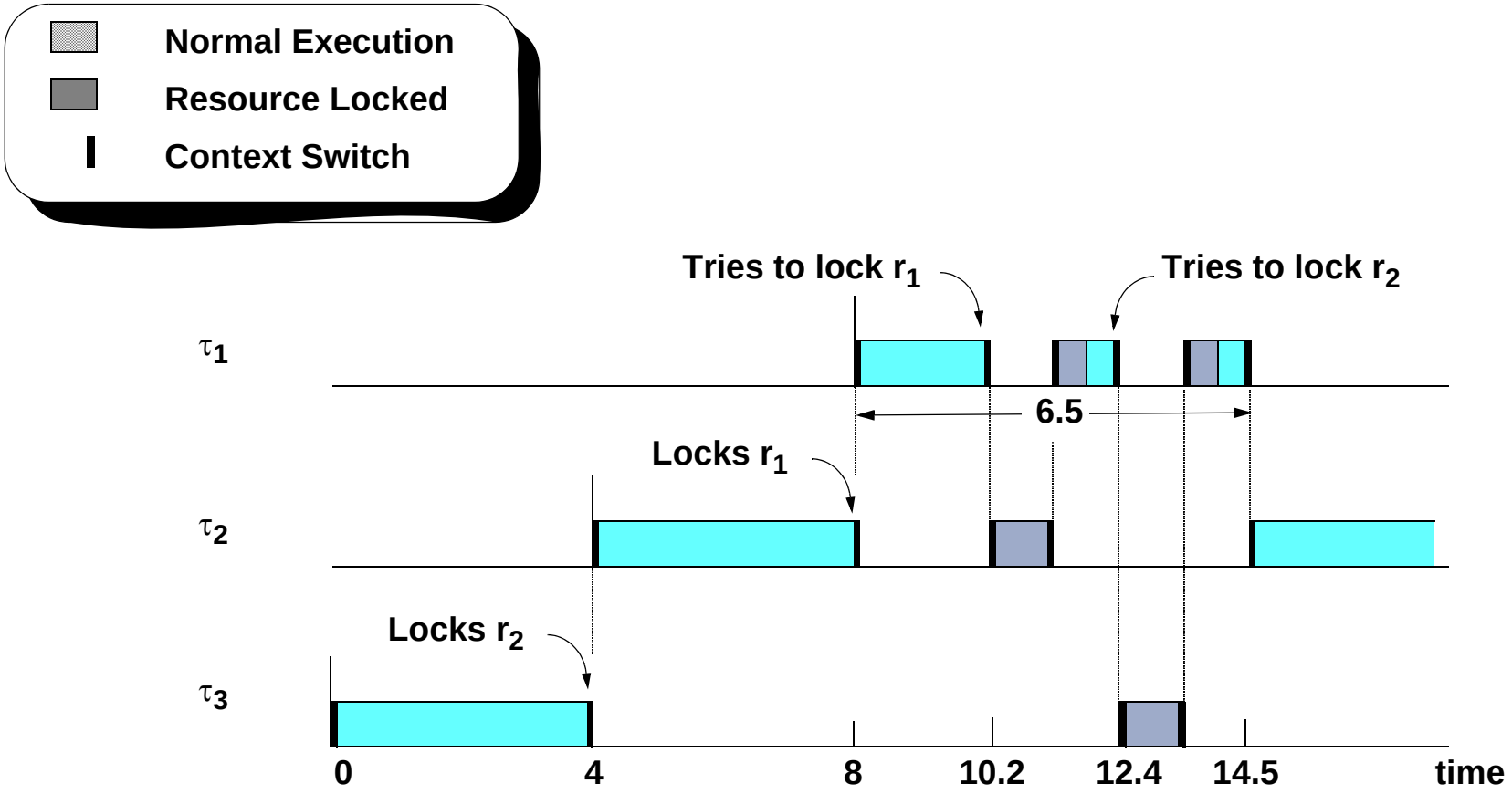
UNIVERSIDAD DE CANTABRIA

To illustrate multiprocessor synchronization, we will show an example of synchronization in a uniprocessor, and we will then extend it to a shared memory multiprocessor. Suppose the time-critical system with three periodic tasks that is shown in the figure above. One of the tasks, τ_1, executes at high priority and has a deadline or worst-case latency requirement of 8 ms. The task takes 4 ms. to execute, including the time spent using two data structures that are shared with another two tasks. Two resources are used, r_1 and r_2, one for each shared data structure; their purpose is to allow a mutually exclusive access to the data structures from the different tasks. The figure shows the time spent by each task with the resource locked. The task periods are considered to be large.

We will assume that tasks τ_1, τ_2, and τ_3 have been assigned priorities 5, 3, and 1 respectively, where a higher number corresponds to a higher priority. When the system is analyzed we find that unbounded priority inversion can appear when τ_1 attempts to lock resource r_2 while it is being owned by τ_3. Under these circumstances τ_1 has to wait for τ_3 to complete its critical section but, because of its low priority level, τ_3 may be preempted by τ_2 and by any other task with priority between 1 and 5. One would have expected that the worst-case time that τ_1 had to wait to acquire resource m_2 was 1 ms., a function only of the duration of the critical section (which is usually short). However, unbounded priority inversion makes this time also depend on the duration of the execution of complete tasks. In our example, if τ_2 takes10 ms. to execute, τ_1 would have to wait for 11 ms. in the worst case due to this effect, and would certainly miss its critical deadline.

Timeline of Example (BIP)

Worst-case latency, single processor, basic inheritance



Notes:

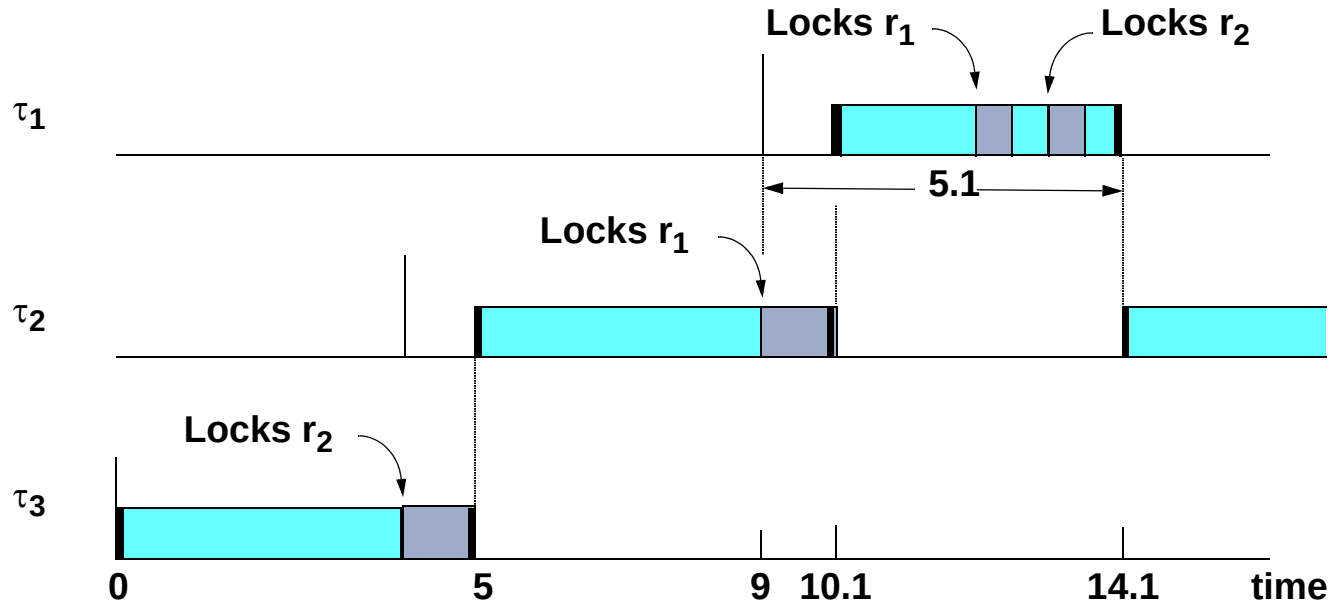
Basic priority inheritance could be used to avoid unbounded priority inversion. Whenever τ_1 is waiting to lock resource r_2, its owner (τ_3) inherits τ_1's priority level (5) and, therefore, it cannot be preempted by intermediate-priority tasks until it unlocks the resource. Consequently, the maximum time spent by task τ_1 waiting for resources to become unlocked becomes only a function of the duration of critical sections. For this example this time is1 ms. for each resource, totaling 2 ms and allowing τ_1 to meet its critical deadline. If we assume a non-negligible context switch time, in the worst case we would have to add two additional context switches for each of the resources that may be found locked. That makes in our example a total of one regular context switch needed to activate the task plus four additional context switches due to synchronization. If for example the context switch time is 0.1 ms., the total latency would be

$$Latency(\tau_1) = C_1 + C_S + \sum_{r_1, r_2} (B_{r_i} + 2C_S) = 4 + 0{,}1 + (1 + 0{,}2) + (1 + 0{,}2) = 6{,}5 \leq 8$$

where C_i is the worst case execution time of task-i, C_S is the context switch time, and B_{r_i} is the maximum time that τ_1 will spend waiting for lower priority tasks to unlock resource r_i. An execution sequence that leads to this worst-case latency is shown in the figure above.

Timeline of Example (IPC)

Worst-case latency, single processor, immediate priority ceiling:



Notes:

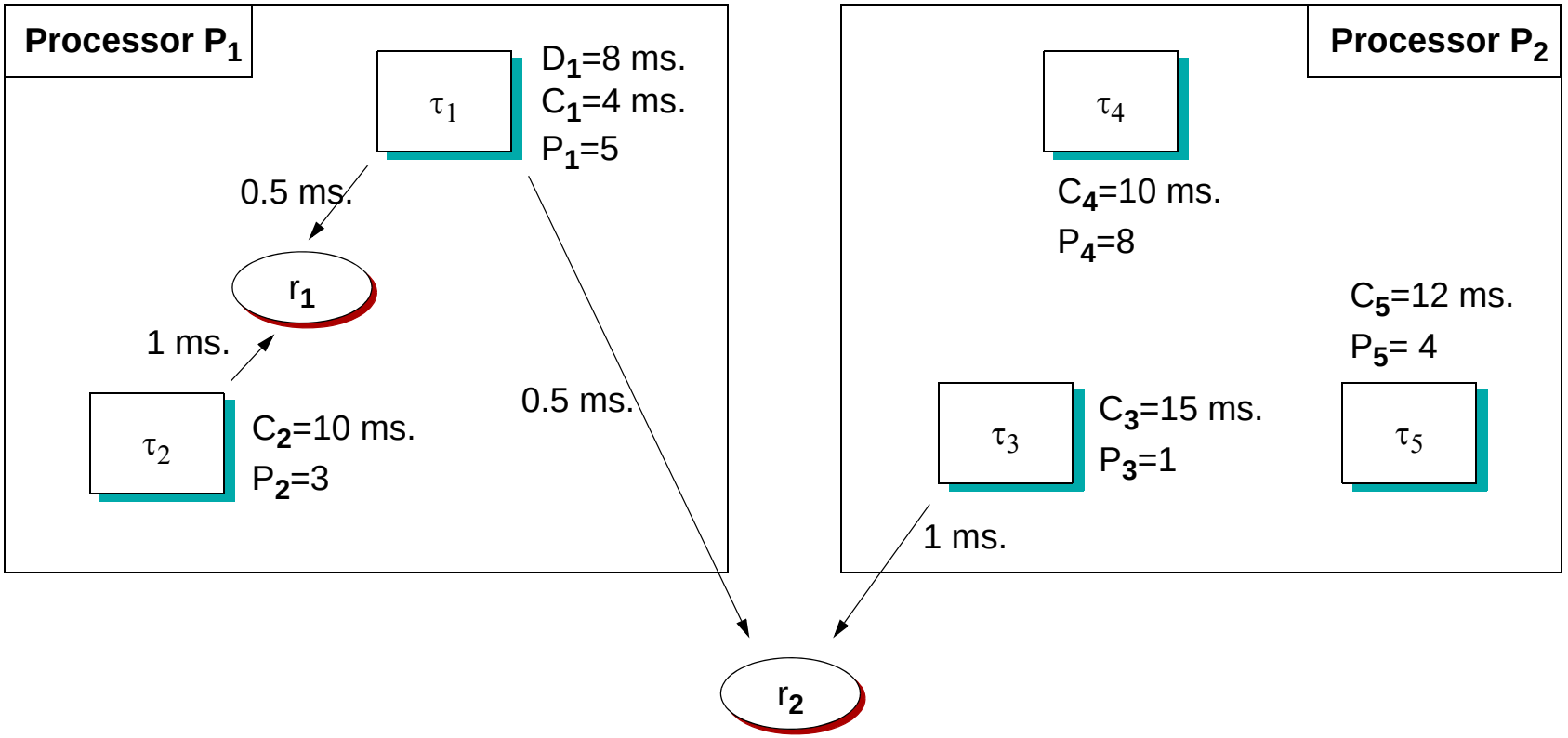
On the other hand, if we use immediate priority ceiling protocol for this example, two advantages would immediately be achieved, if we assume no suspension inside critical sections. First, τ_1 has to wait for at most one resource held locked by lower priority tasks. Second, the worst-case number of context switches due to synchronization becomes zero (only one remains for the activation of the task). This makes the total latency be:

$$Latency(\tau_1) = C_1 + C_S + Max(B_{r_1}, B_{r_2}) = 4 + 0{,}1 + Max(1, 1) = 5{,}1 \leq 8$$

An execution sequence that leads to this worst-case latency is shown in the figure above. It can also be seen that the immediate priority ceiling protocol preserves the blocked-at-most-once property by not letting τ_2 preempt τ_3 during its critical section

Multiprocessor Example

Synchronization in a multiprocessor



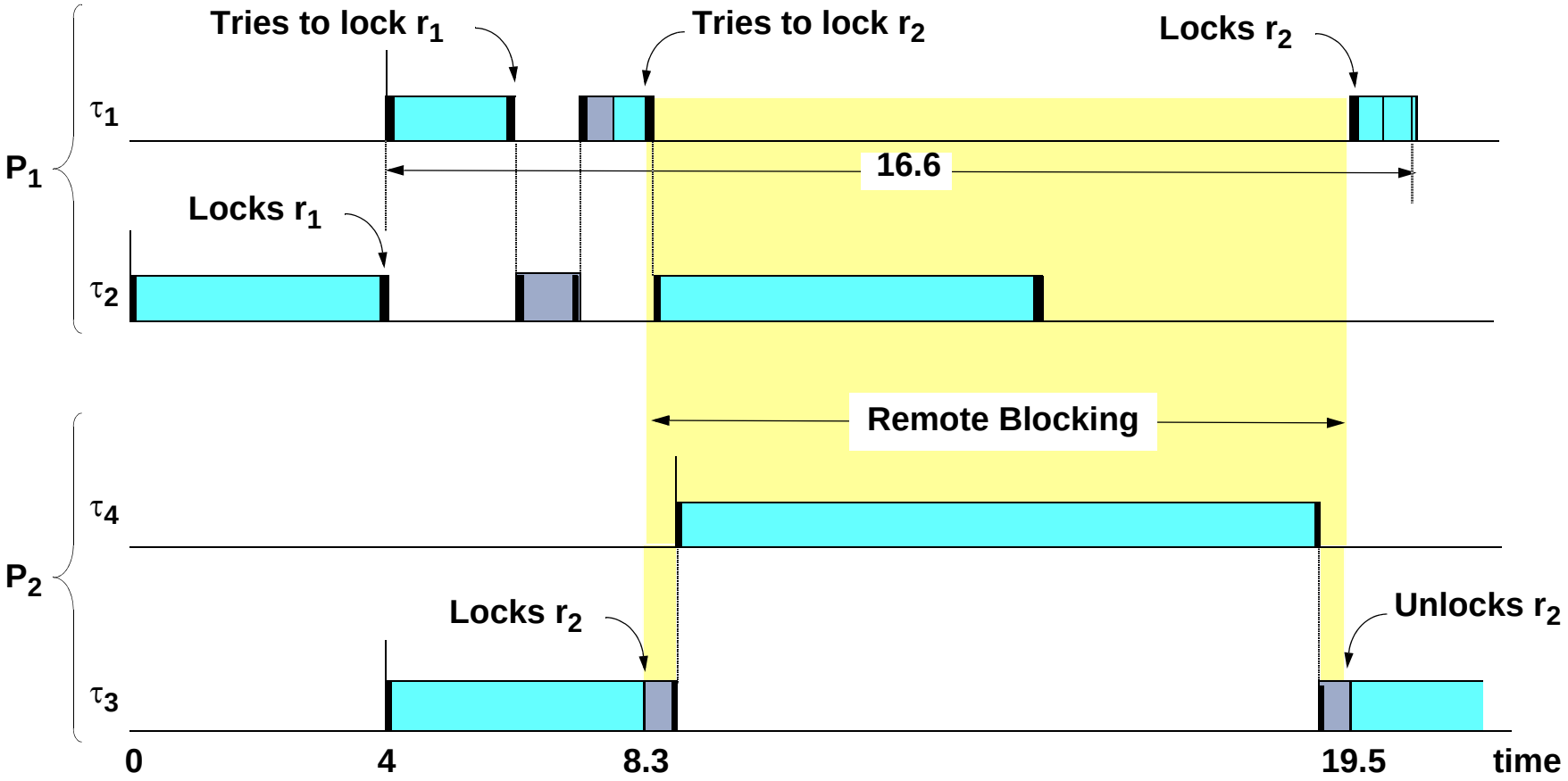
Notes:

UNIVERSIDAD DE CANTABRIA

Suppose that the system that appears in our example has growing demands for CPU utilization and a decision is made to move τ_3 to another processor in the system. This processor had two other tasks assigned to it, τ_4 and τ_5, which have worst case execution times of 10 and 12 ms., respectively. We will assume a shared memory system and that no changes to the code of the different tasks needs to be made. We will also neglect, for simplicity, the effects of contention in the access to shared memory. The figure above shows the new structure of the system.

Remote Blocking in a Multiprocessor

Worst-case latency, multiprocessor, basic inheritance



Notes:

Assuming that basic inheritance is being used we want to determine how much time does τ_1 have to spend waiting for resources r_1 or r_2 to become unlocked. We can see that, because of priority inheritance, τ_5 cannot influence this time, as its priority level is not high enough to preempt τ_3 if inheritance has taken effect. However, τ_4 is always able to preempt τ_3, and the total time spent by τ_1 waiting for r_2 could be 11 ms. in the worst case. Task τ_1 will miss its deadline because now its blocking time is again a function not only of the duration of critical sections but also of the execution of complete tasks. This effect is called remote blocking, and it has been shown that it can severely degrade the degree of CPU utilization in time-critical multiprocessor systems. The potential gain in computation power expected from the use of multiple processors is nullified because of this preemption effect across processors. The figure above shows an execution sequence that leads to this behavior.

Global Priority Ceiling Protocol

UNIVERSIDAD DE CANTABRIA

Works the same as the priority ceiling protocol

The concept of priority ceiling is redefined:

- **Local resources (used from a single processor):**
  - **Same as before (immediate priority ceiling)**
- **Global resources (used from multiple processors):**
  - **Above the priorities of any task in the system**
  - **Ordered according to the uniprocessor ceilings**

This protocol avoids remote blocking as well as unbounded priority inversion

Notes:

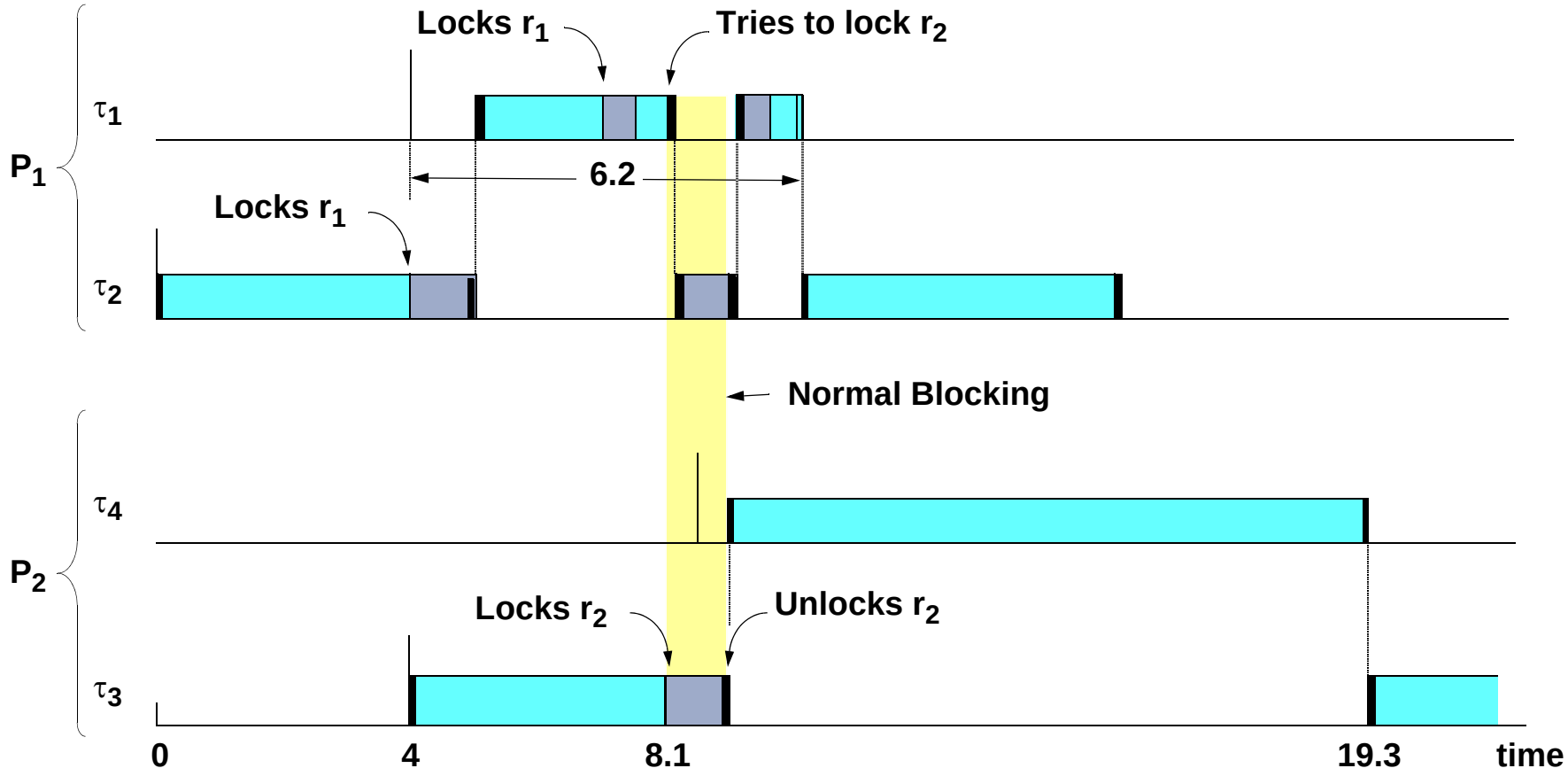
Basic inheritance is clearly inadequate for synchronization in multiprocessors. Its properties are based on the existence of a single scheduler, so when multiple schedulers are present it fails to preserve the invariant of synchronization waiting times being only a function of the duration of critical sections. However, both unbounded priority inversion and remote blocking can be avoided by using the immediate priority ceiling protocol, and setting the priority ceilings of each resource to the appropriate levels. These levels are:

- *Local resources*, only used by tasks in the same processor: The local ceiling of a resource is defined as the highest of the priorities of all the tasks that may lock that resource.
- *Global resources*, used by tasks in at least two different processors. The ceiling of a global resource must be above the priorities of any of the tasks in the system. In this class of higher priority levels, the ceilings will be assigned such that resources with higher local ceiling (as defined above) are assigned a higher global ceiling.

The resultant protocol is called the global priority ceiling protocol.

Avoiding Remote Blocking

Worst-case latency, multiprocessor, immediate priority ceiling



Notes:

Suppose that we use global priority ceiling protocol for our multiprocessor example. Resource r_1 is local and, therefore, it is assigned a ceiling of 5. Resource r_2 is global and is assigned a priority ceiling higher than the priority of any other task in the system, for example priority 9. Immediate priority ceiling causes τ_3's critical section to be executed at priority level 9, thus avoiding any preemption effects from τ_4 during the critical section. task τ_4 now potentially experiences a new blocking effect due to the high priority execution of this critical section but this is still a function of the duration of critical sections. task τ_1 is now blocked only for the duration of critical sections: its worst case waiting time is 1 ms. for each, r_1 and r_2, totaling a latency of 6 ms. and therefore meeting its deadline.

Summary of Example

Worst-case latency results for τ_1

	No Protocol	Basic Inheritance	Global priority ceiling
Uniprocessor	15.0+other tasks[1]	6.0	5.0
Uniprocessor & context switch[4]	15.5+other tasks[1]	6.5	5.1
Multiprocessor	28.0+other tasks[2]	16.0+other tasks[3]	6.0
Multiprocessor & context switch[4]	28.8+other tasks[2]	16.6+other tasks[3]	6.2

[1] Other tasks with priorities between 1 and 5.
[2] Other tasks in processor 2 with priorities above 3.
[3] Other tasks in processor 2 with priorities above 5.
[4] The actual worst-case number of context switches may slightly depend on the implementation or the timing requirements.

Notes:

UNIVERSIDAD DE CANTABRIA

In summary:

- In *uniprocessors*, the immediate priority ceiling protocol represents a better approach for the worst-case response time than basic inheritance. It reduces the amount of time spent by higher priority tasks waiting for lower priority tasks to complete critical sections, and also reduces the worst-case amount of context switches.
- In *multiprocessors*, the global priority ceiling protocol represents a significantly better approach than basic inheritance for the worst-case response time. In fact, basic inheritance is unusable for most time-critical multiprocessor systems if high levels of CPU utilization are desired. It also makes the system be very sensitive to changes, because the unbounded nature of the latency causes a change made to one task to affect the latencies of tasks in other processors, even if the modified task does not synchronize with any other tasks. The global priority ceiling protocol solves the remote blocking problem and continues to minimize the number of worst-case context-switches due to synchronization.