Predictable Implementation of Real-Time Applications on Multiprocessor Systems on Chip

Petru Eles, Jakob Rosén, Alexandru Andrei, Zebo Peng

Department of Computer and Information Science (IDA)
Linköping University
http://www.ida.liu.se/~eslab/

We want systems to be predictable!

We want systems to be predictable!

Safety critical

We want systems to be predictable!

QoS should be guaranteed!

- **Real-Time:**

  Time constraints have to be satisfied!

  Safe W(B)CETs have to be known

☞ **Many current/future real-time applications need both**

- performance
- predictability

- Strong(er) processors
- Multiprocessors on Chip

☞ **Many current/future real-time applications need both**

- performance
- predictability

- Strong(er) processors
- Multiprocessors on Chip

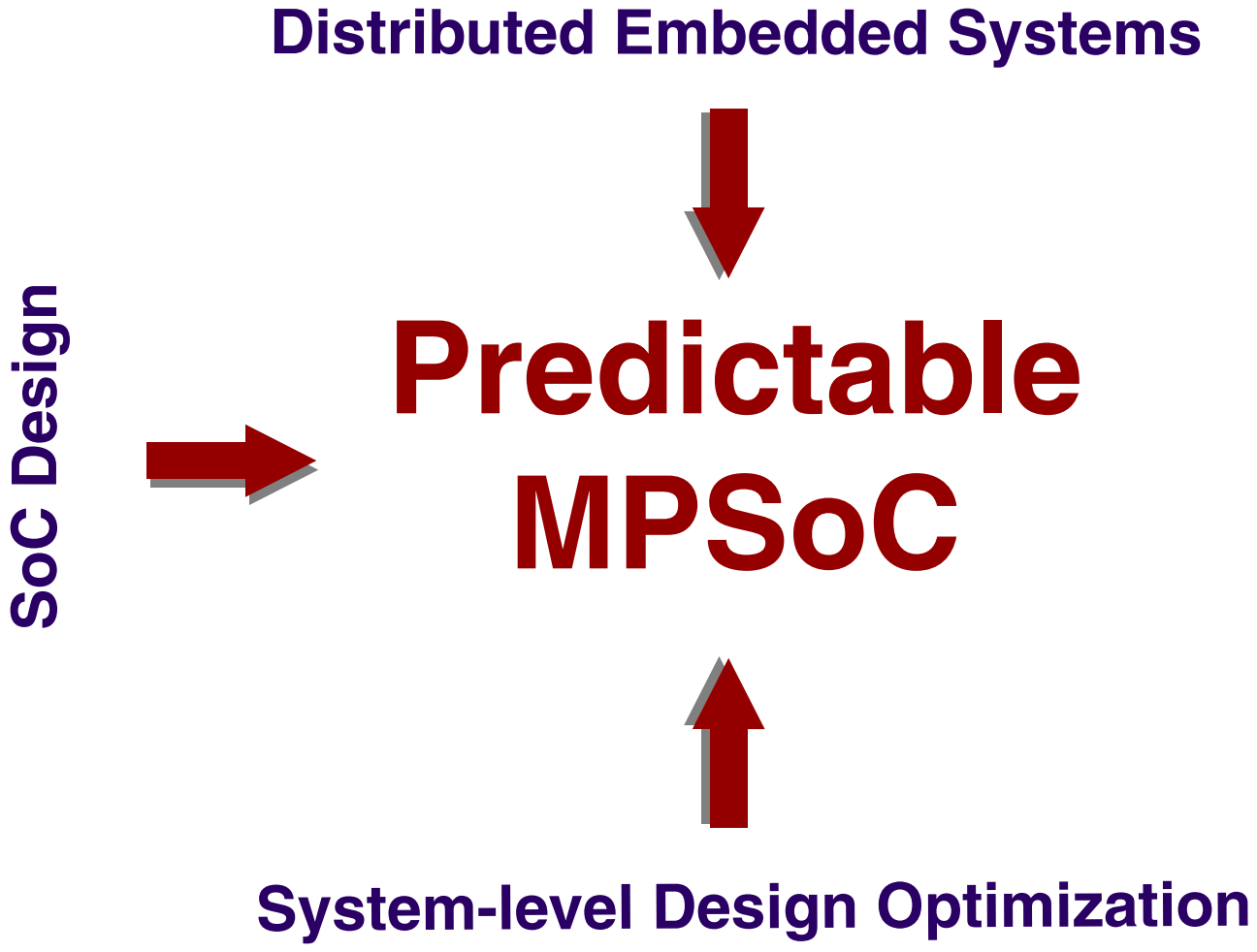
Can we still achieve predictability?

Outline

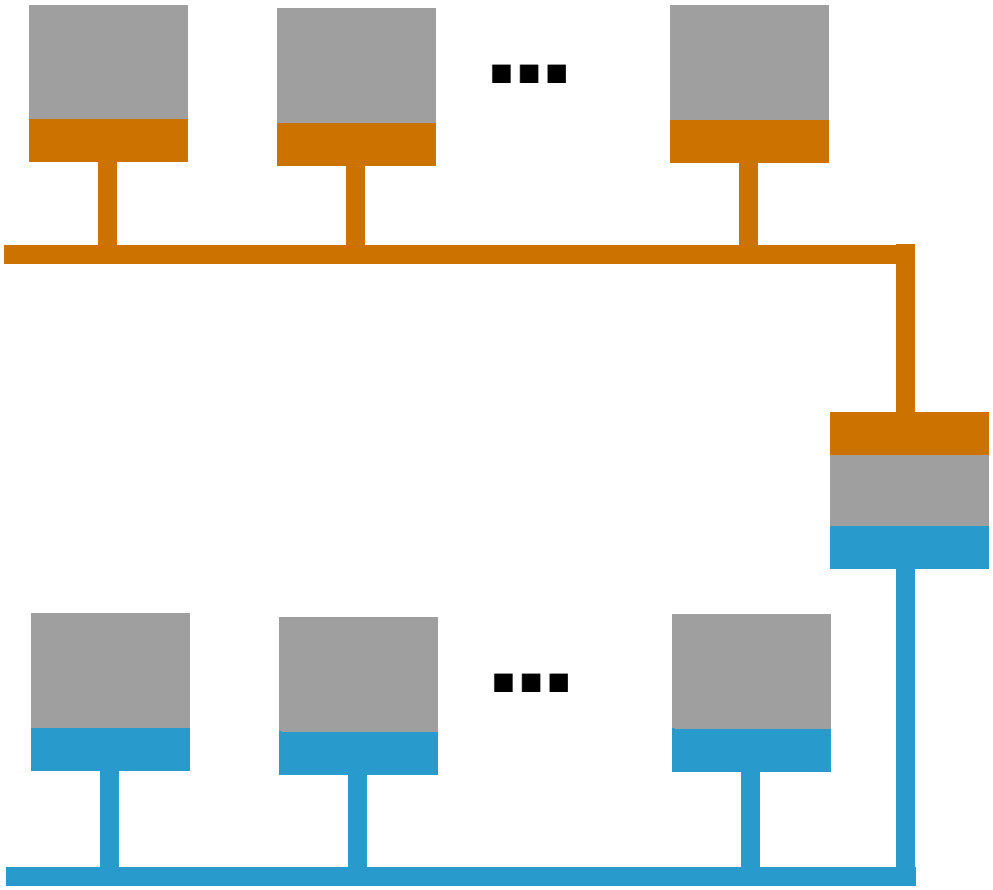
- Background
- Hardware Architecture & Application Model
- The Problem and Possible Solutions
- WCET Analysis with TDMA Bus Schedule
- TDMA Bus Scheduling Approaches
- Bus Schedule Generation
- Conclusions

How did we come to this?

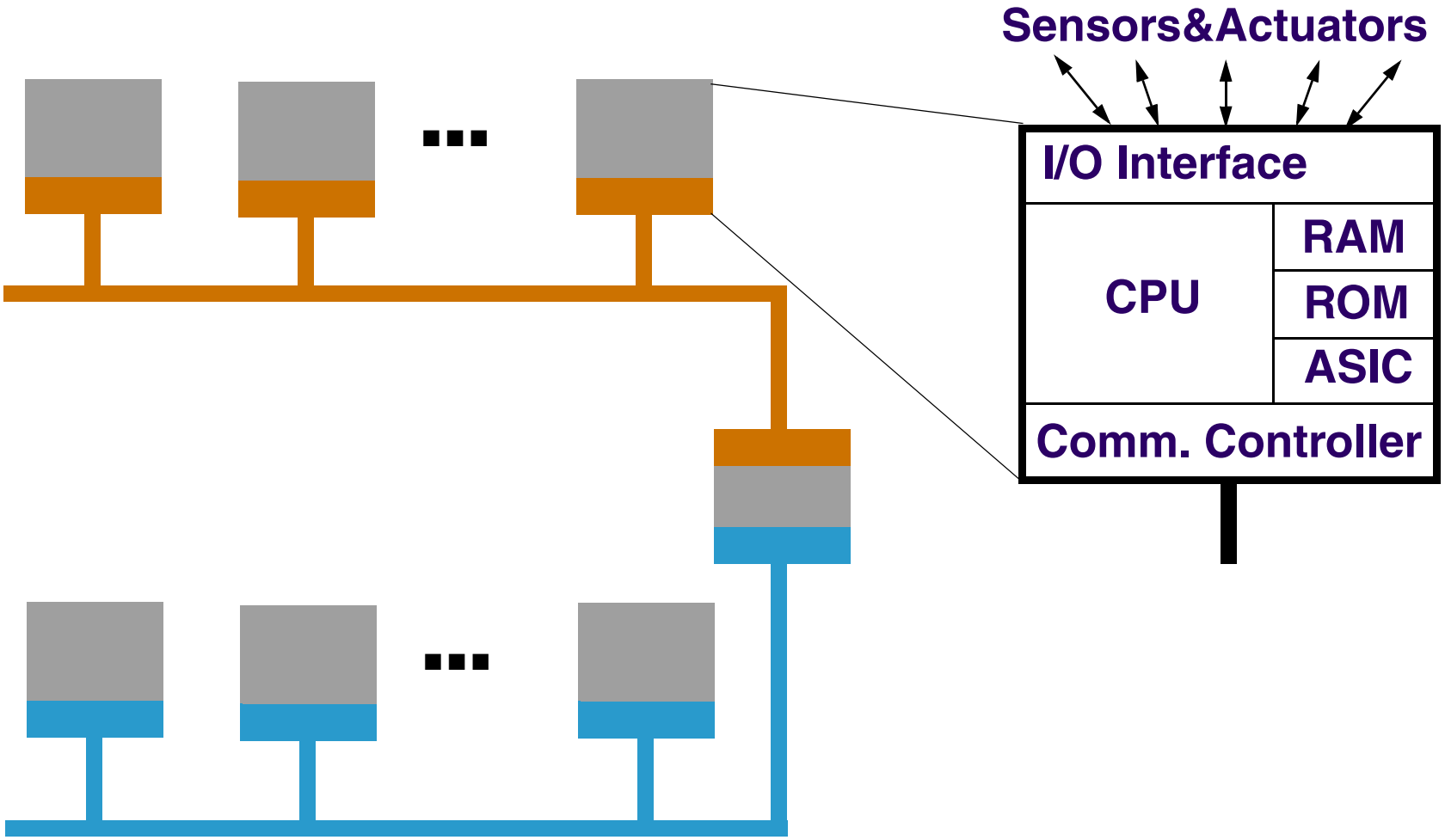
How did we come to this?



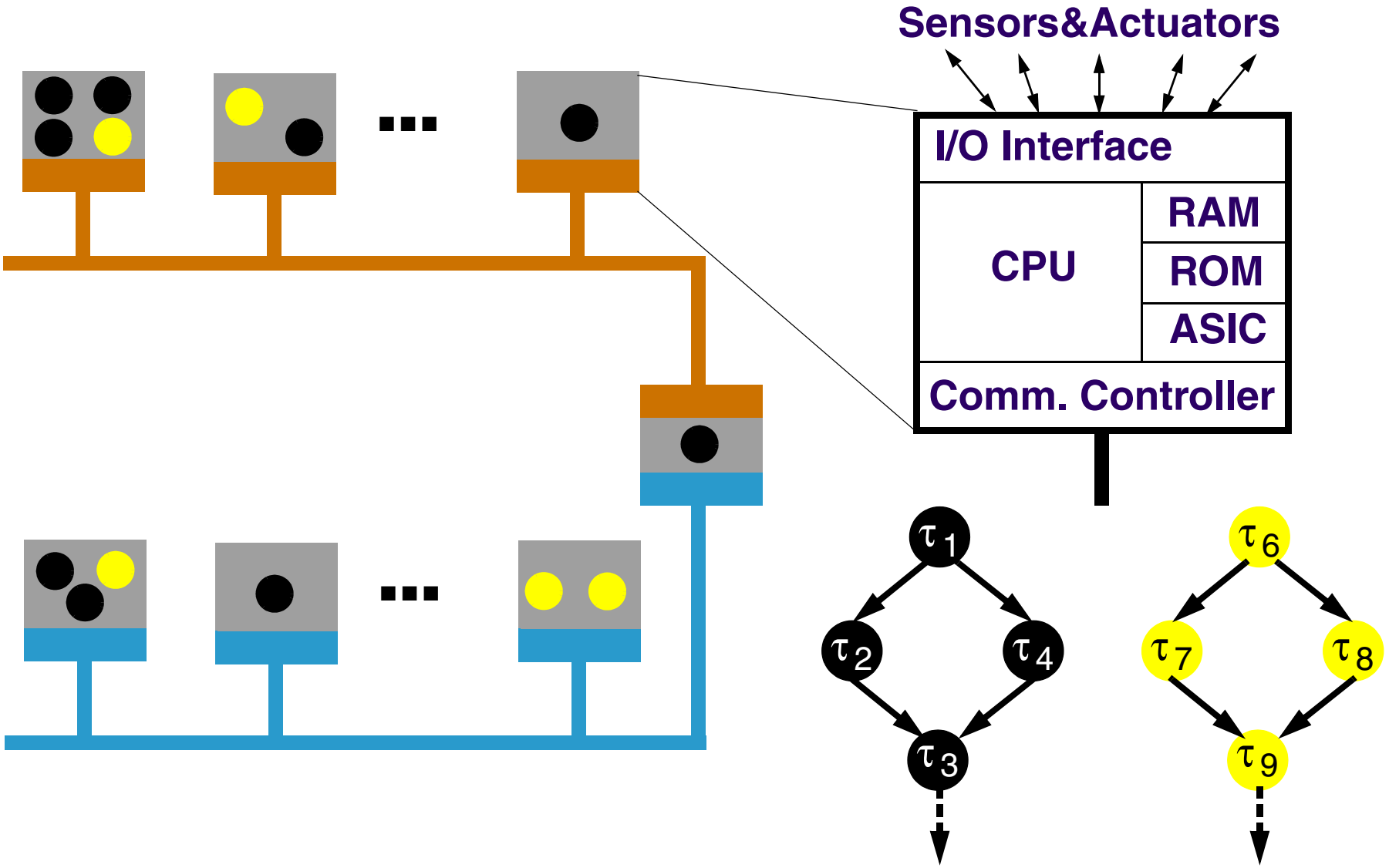
Distributed Embedded Systems



Distributed Embedded Systems



Distributed Embedded Systems

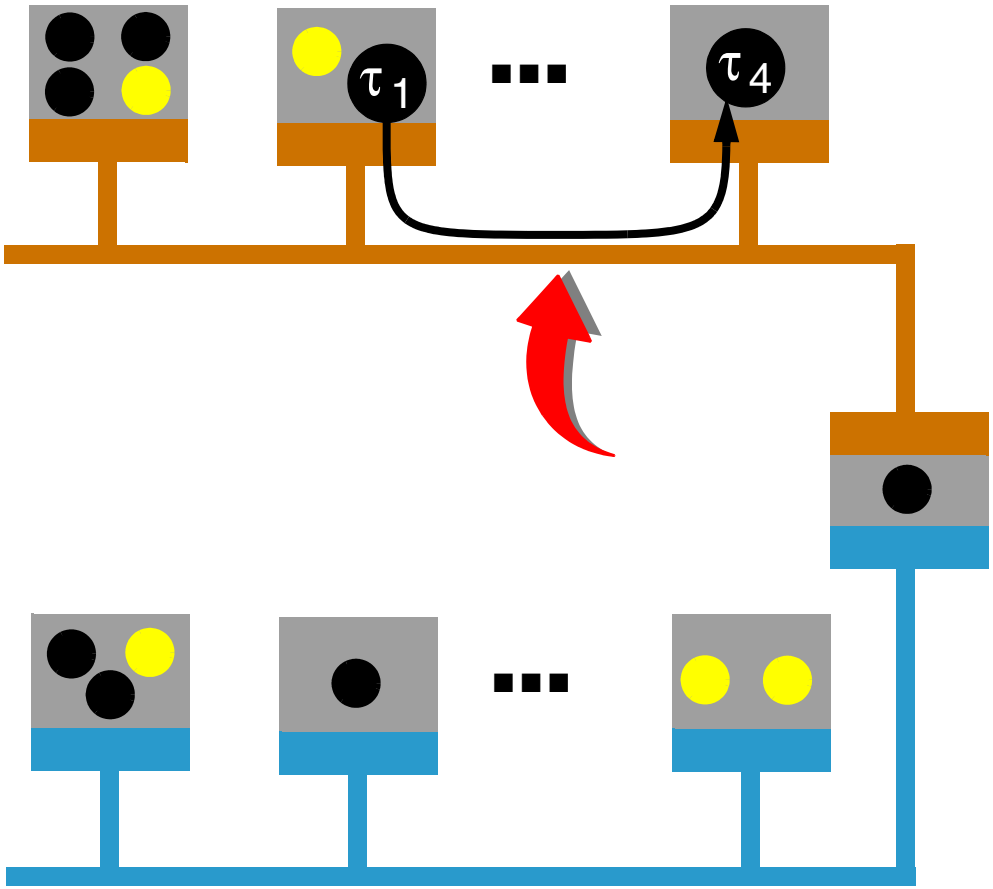


Distributed Embedded Systems

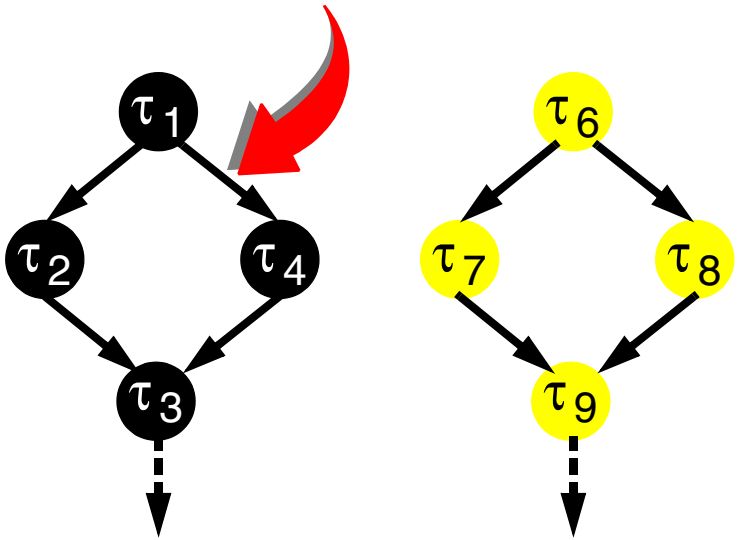
- Worst-case response time analysis
- Static-cyclic schedule generation
- Analysis of heterogeneous time/event triggered systems
- Static/Dynamic/heterogeneous communication (TTP, CAN, FlexRay)
- Scheduling with Fault-tolerance constraints

- Optimization problems:
  - schedule generation/priority assignment/voltage selection
  - task mapping
  - bus configuration
  - scheduling policy assignment
  - fault tolerance policy assignment

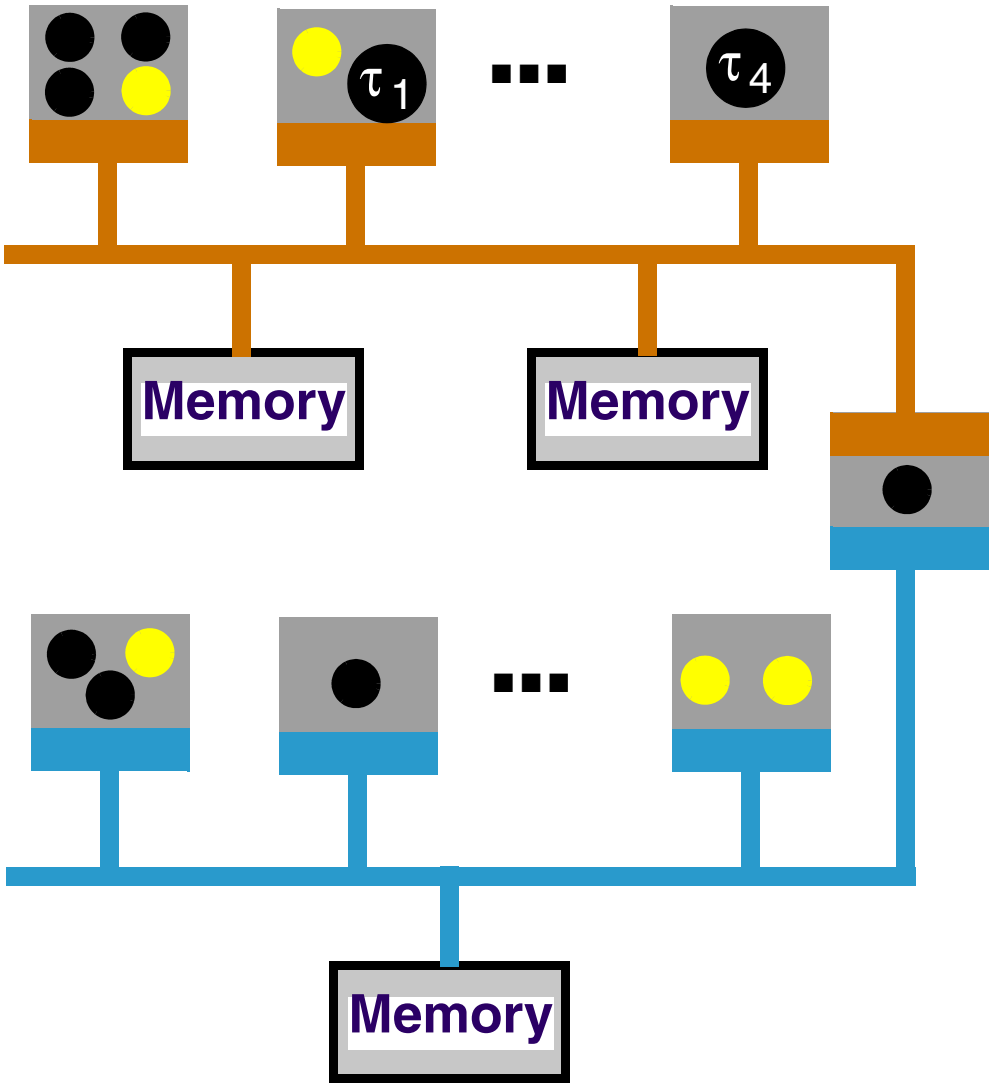
Distributed Embedded Systems



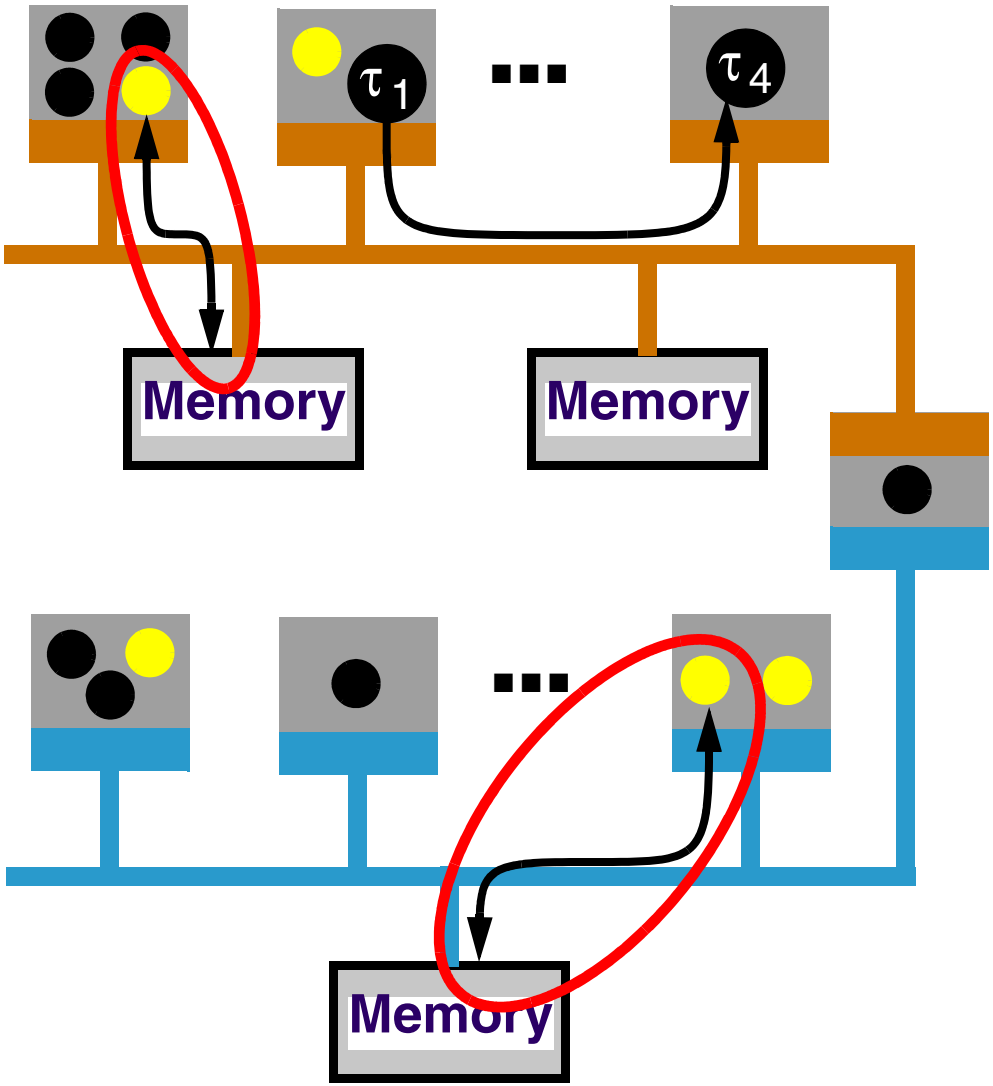
- Only message passing between tasks (explicit communication) is going over the shared bus.
- Bus sharing has NO impact on task WCET.



The MPSoC case

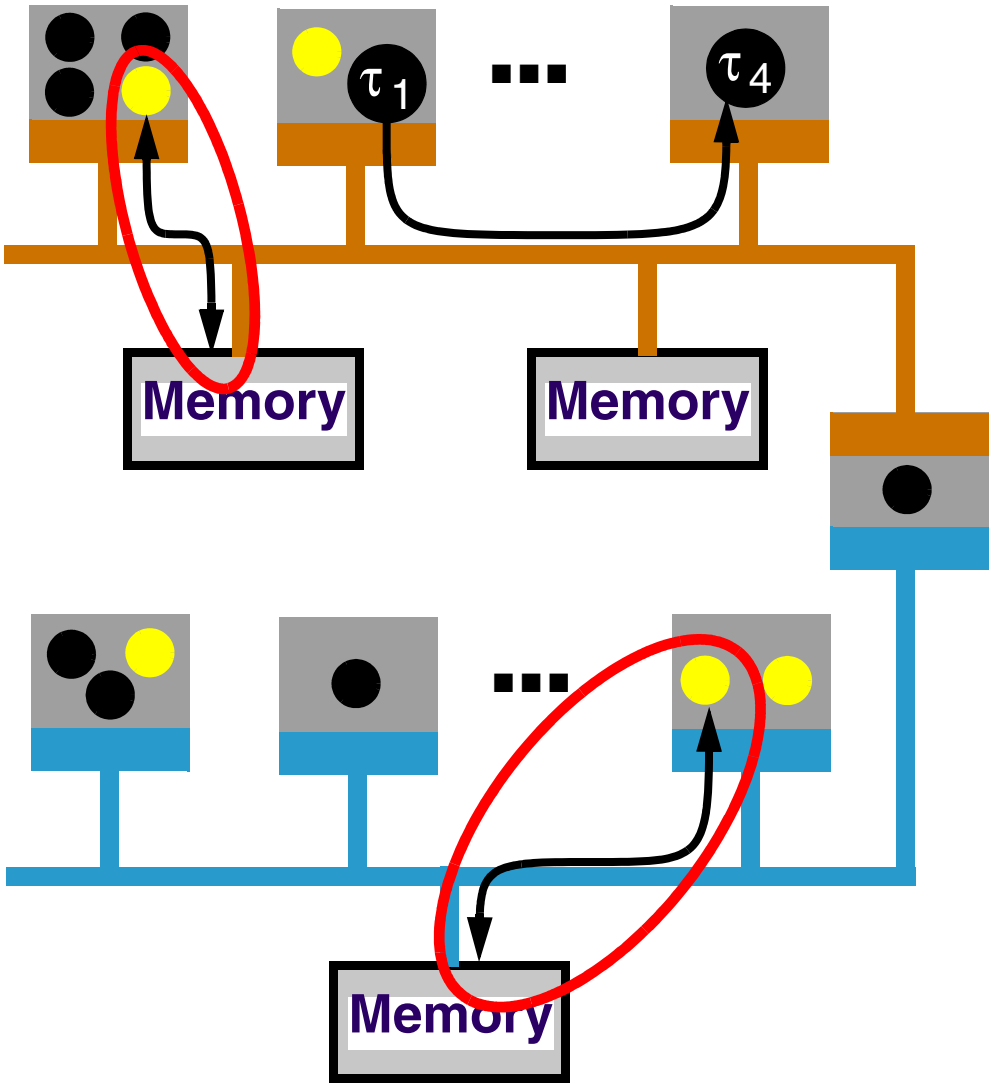


The MPSoC case



- Memory accesses (in case of cache miss) are going over the shared bus.
- Bus sharing has impact on task WCET!!!

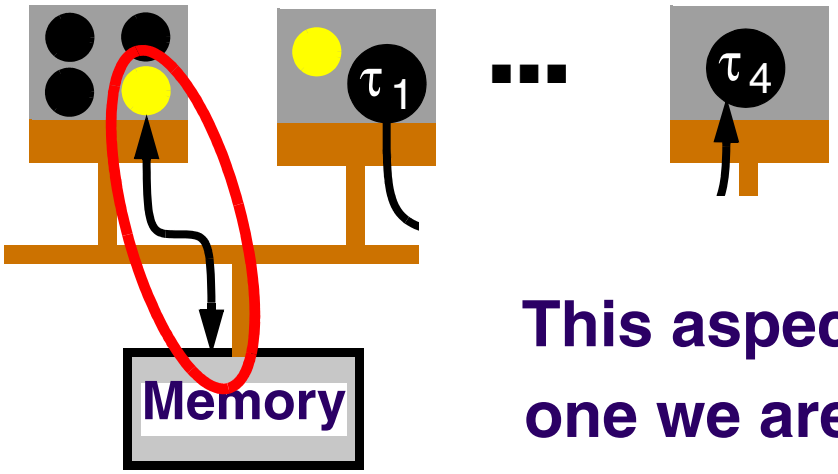
The MPSoC case



- Memory accesses (in case of cache miss) are going over the shared bus.
- Bus sharing has an impact on task WCET!!!

You ignore this: Your results are WRONG!

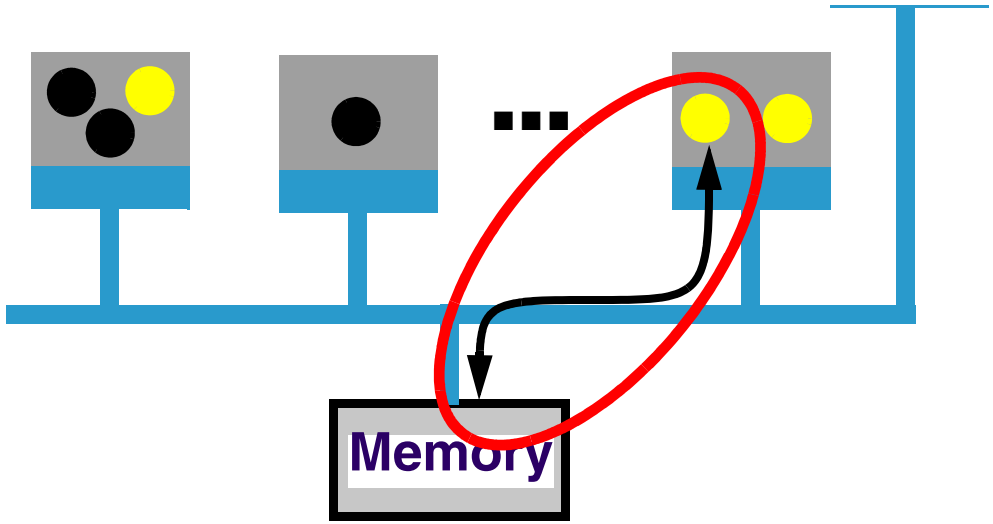
The MPSoC case



☞ Memory accesses (in case of cache miss) are going over the

This aspect of predictability is the one we are interested in here!

an impact on



You ignore this: Your results are WRONG!

Some Context

- Deriving WCET in the case of a *single processor* is a BIG problem:
  - Pipelining
  - Cache memory
  - Branch prediction
  - Out of order execution

Some Context

☞ Deriving WCET in the case of a *single processor* is a BIG problem:

- Pipelining
- Cache memory
- Branch prediction
- Out of order execution

Interference between processor components leeds to *timing anomalies*.

Execution time depends on system state!

Some Context

☞ Deriving WCET in the case of a *single processor* is a BIG problem

- Pipelining
- Cache memory
- Branch prediction
- Out of order execution

Interference between processor components leeds to *timing anomalies*.

Execution time depends on system state!

WCET analysis is
- complex
- potentially pessimistic

Some Context

- Eliminate “problematic” features?

Some Context

- Eliminate “problematic” features?

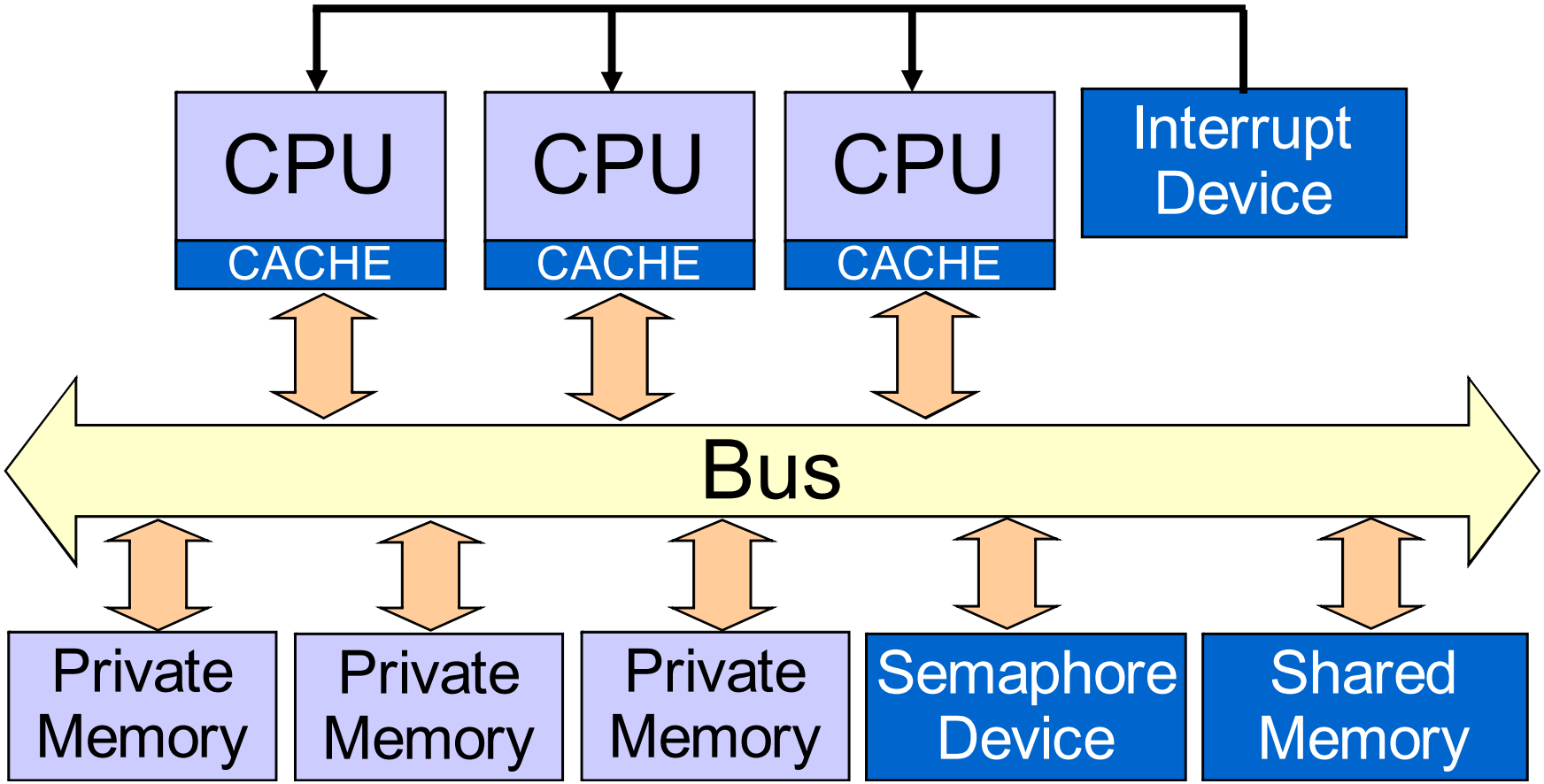
- The player is not allowed to run faster than the referee?!

Some Context

☞ In the case of multiprocessors, on top of all those problems you have:

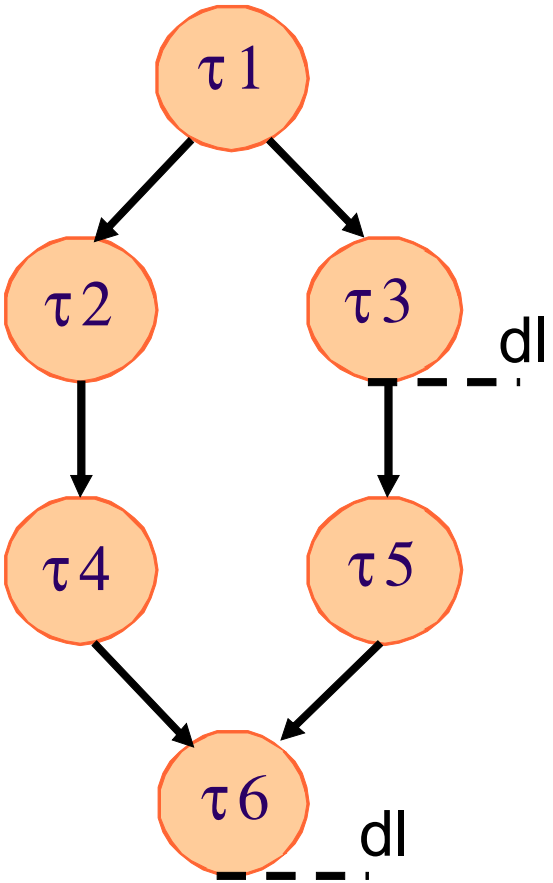
- Interference between active processes
- Additional shared resources (e.g. communication infrastructure)

Hardware Architecture



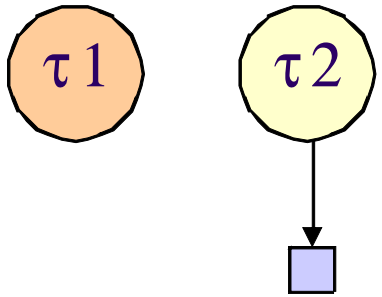
Application Model

- Task Graph
  - Application deadline
  - Individual Task deadlines

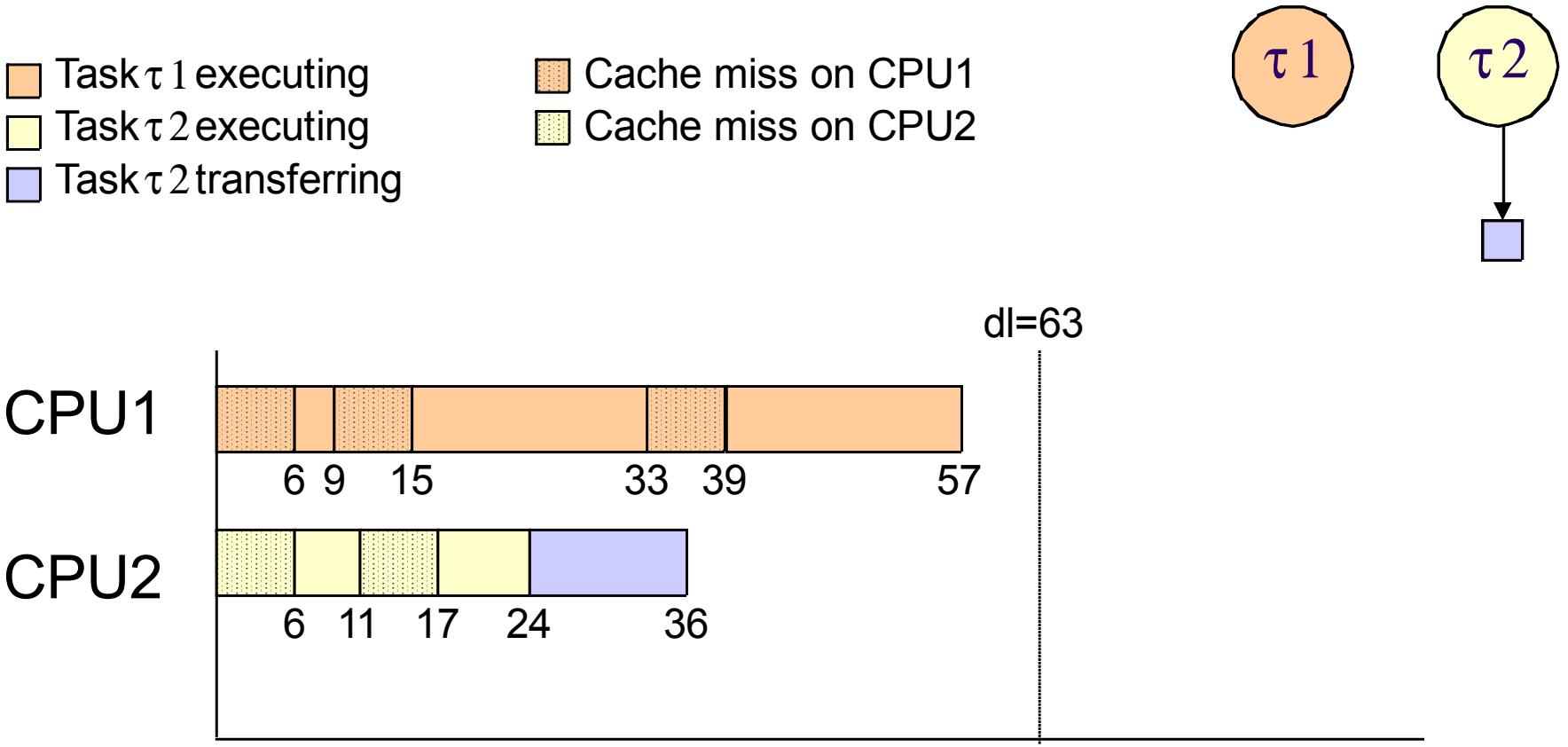


An Example

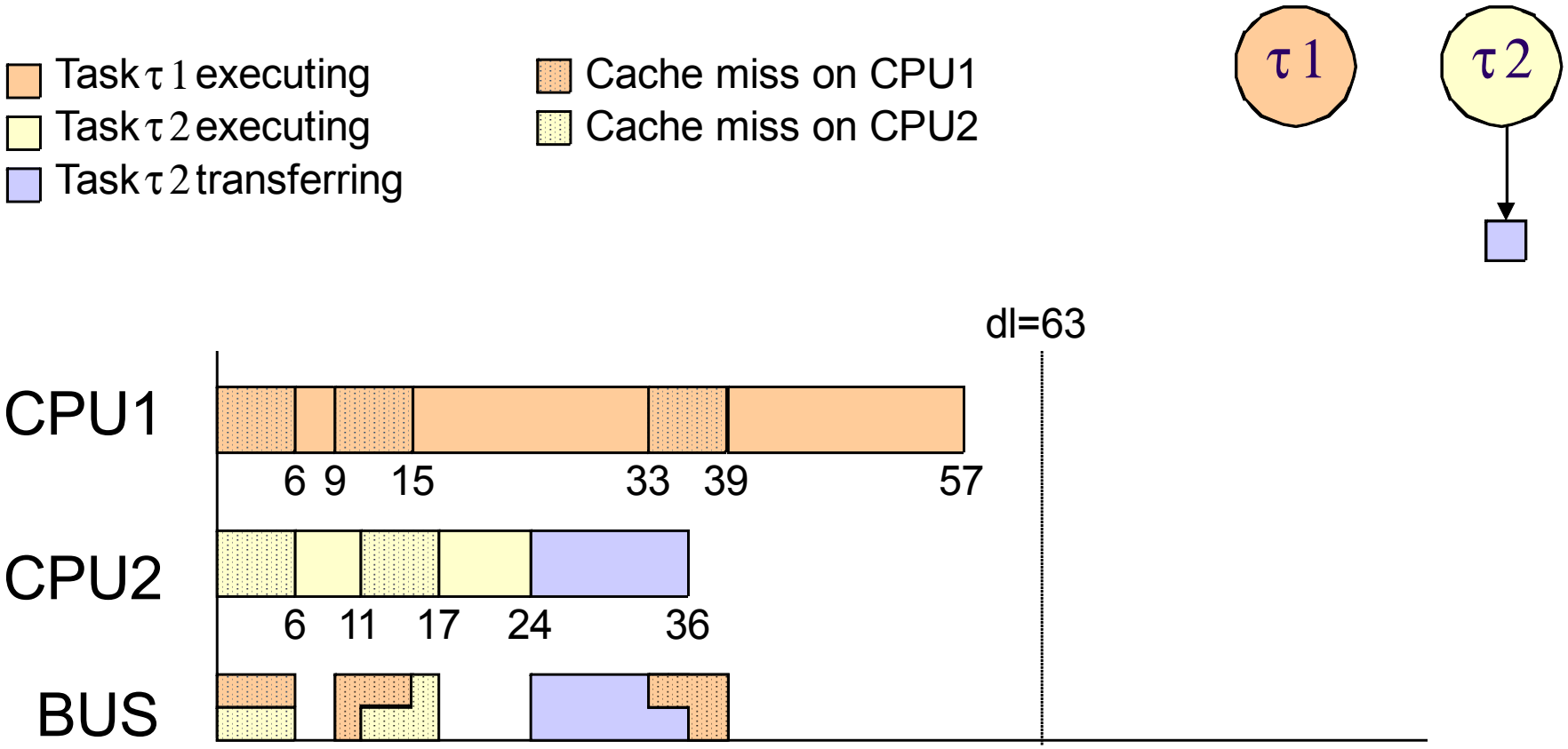
- WCETs - as from “traditional” WCET analysis:
  - t1: 57
  - t2: 24
  - tw: 12
- Deadline: 63



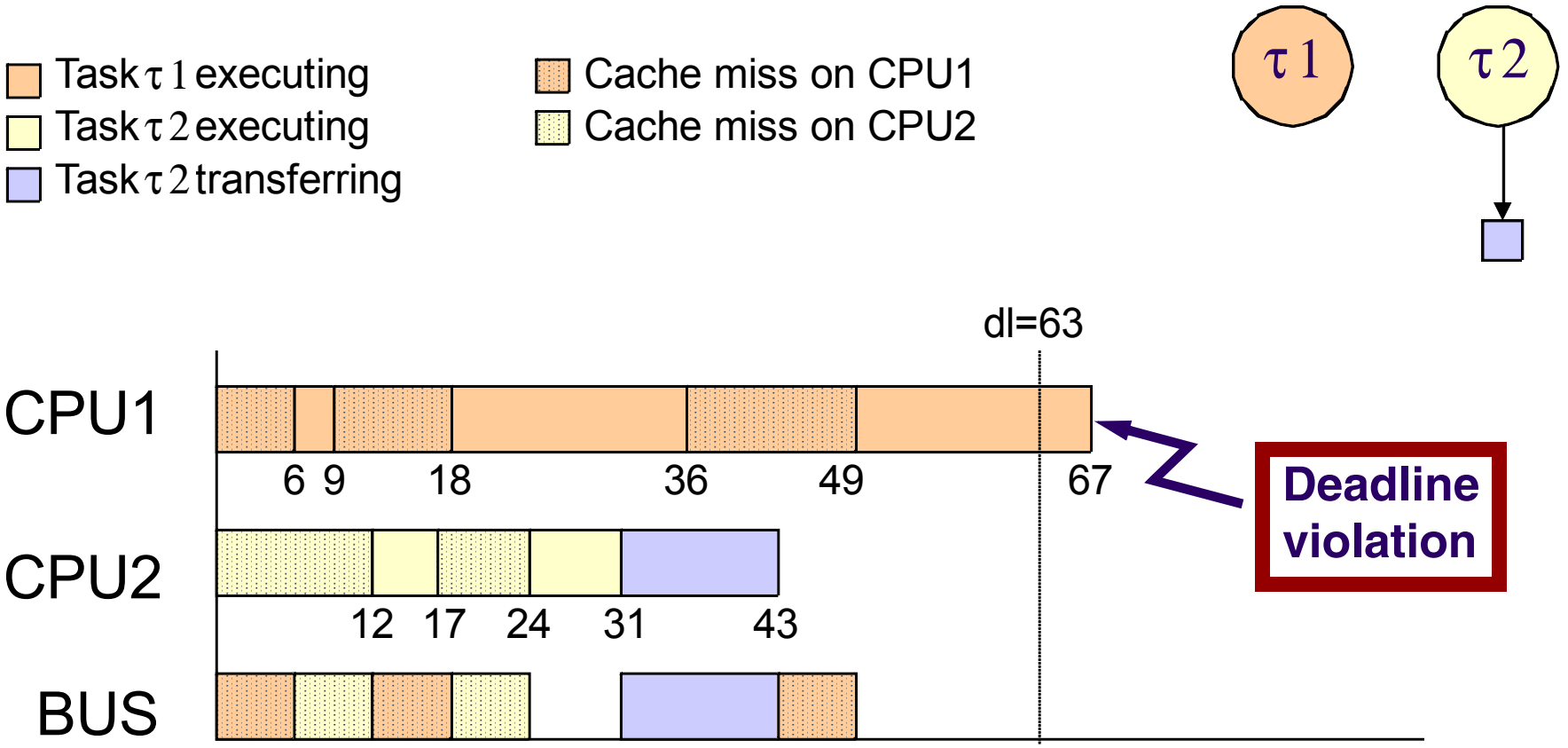
An Example


Task τ1 executing
Task τ2 executing
Task τ2 transferring
Cache miss on CPU1
Cache miss on CPU2
τ1
τ2
dl=63
CPU1
6 9 15 33 39 57
CPU2
6 11 17 24 36

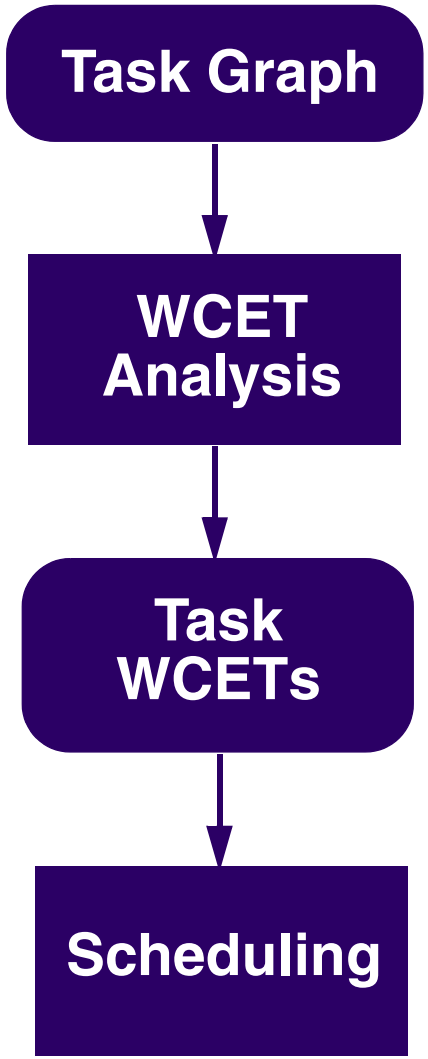
An Example


Task τ1 executing
Task τ2 executing
Task τ2 transferring
Cache miss on CPU1
Cache miss on CPU2
τ1
τ2
dl=63
CPU1
6 9 15 33 39 57
CPU2
6 11 17 24 36
BUS

An Example

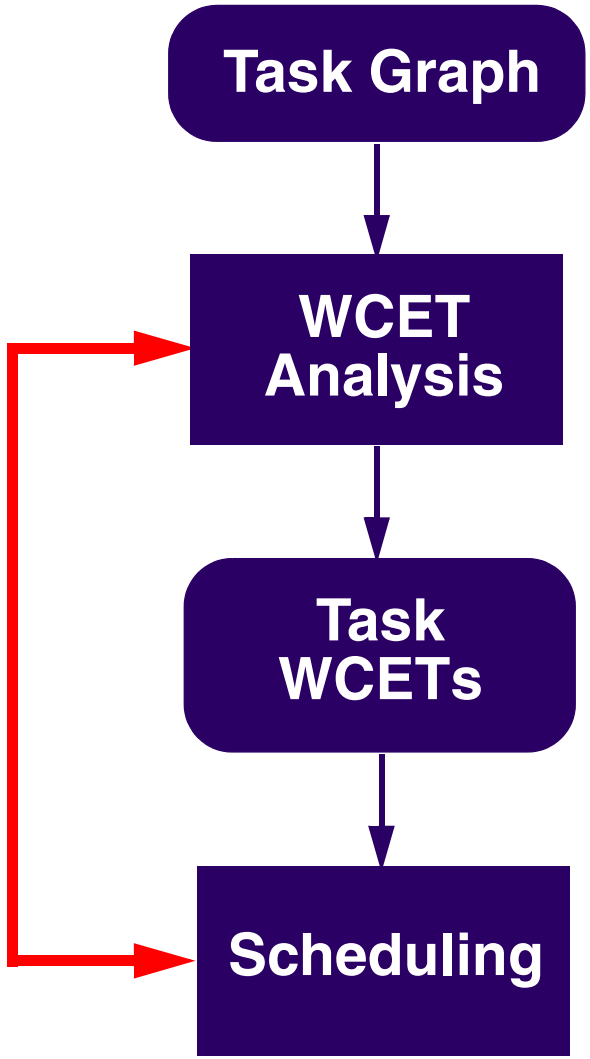

Task τ1 executing
Task τ2 executing
Task τ2 transferring
Cache miss on CPU1
Cache miss on CPU2
τ1
τ2
dl=63
CPU1
CPU2
BUS
6 9 18 36 49 67
12 17 24 31 43
Deadline violation

“Traditional” Approach

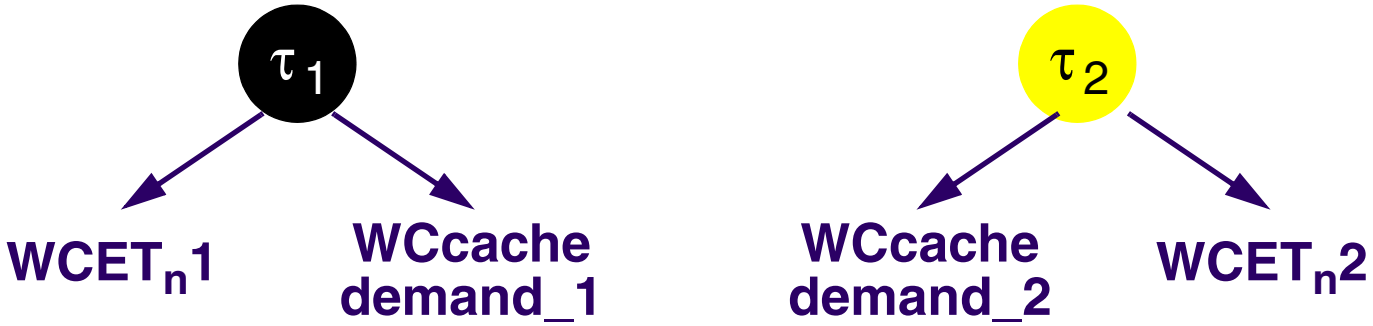


In The Multiprocessor Case

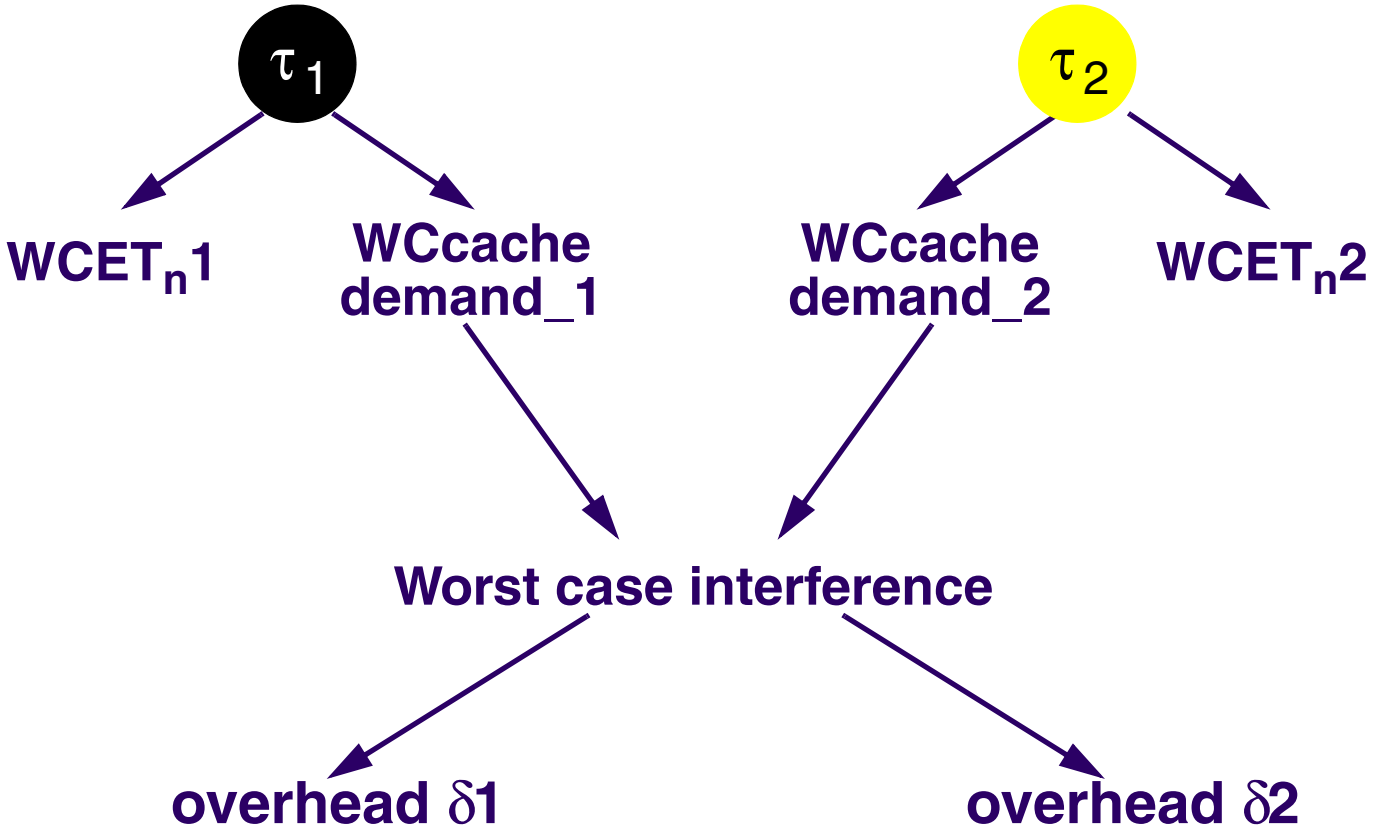
- The WCETs depend on the schedule (interference due to bus conflicts).
- The schedule depends on the WCETs.



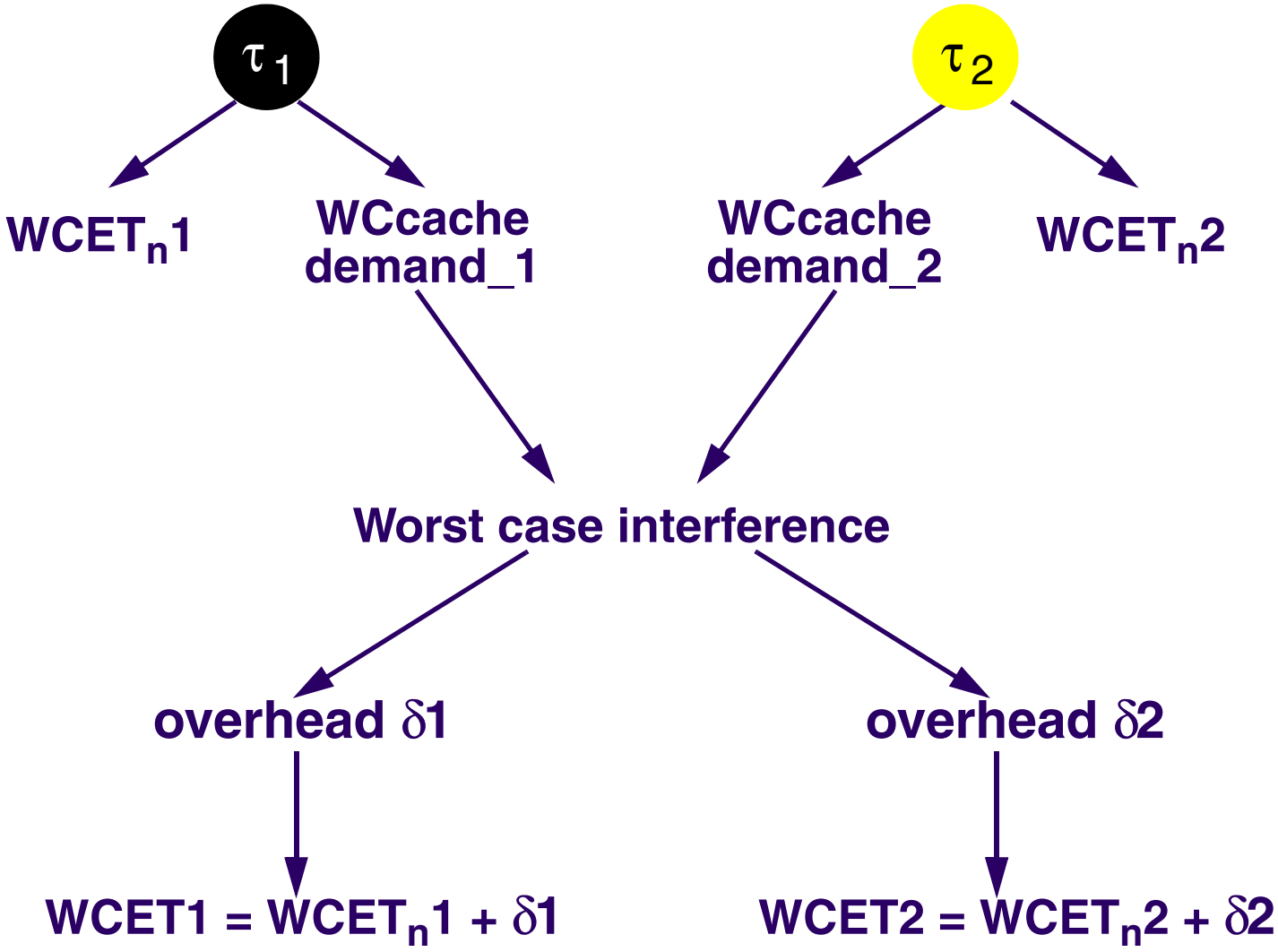
A Possible Way?



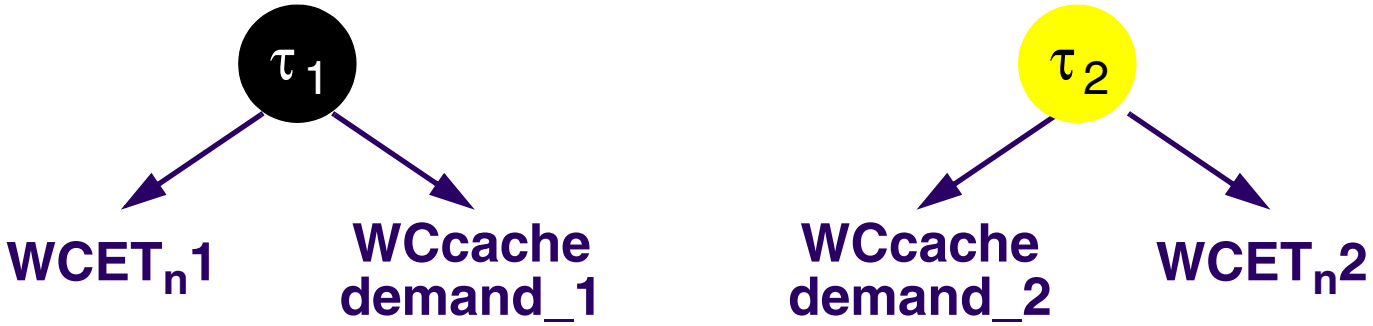
A Possible Way?



A Possible Way?



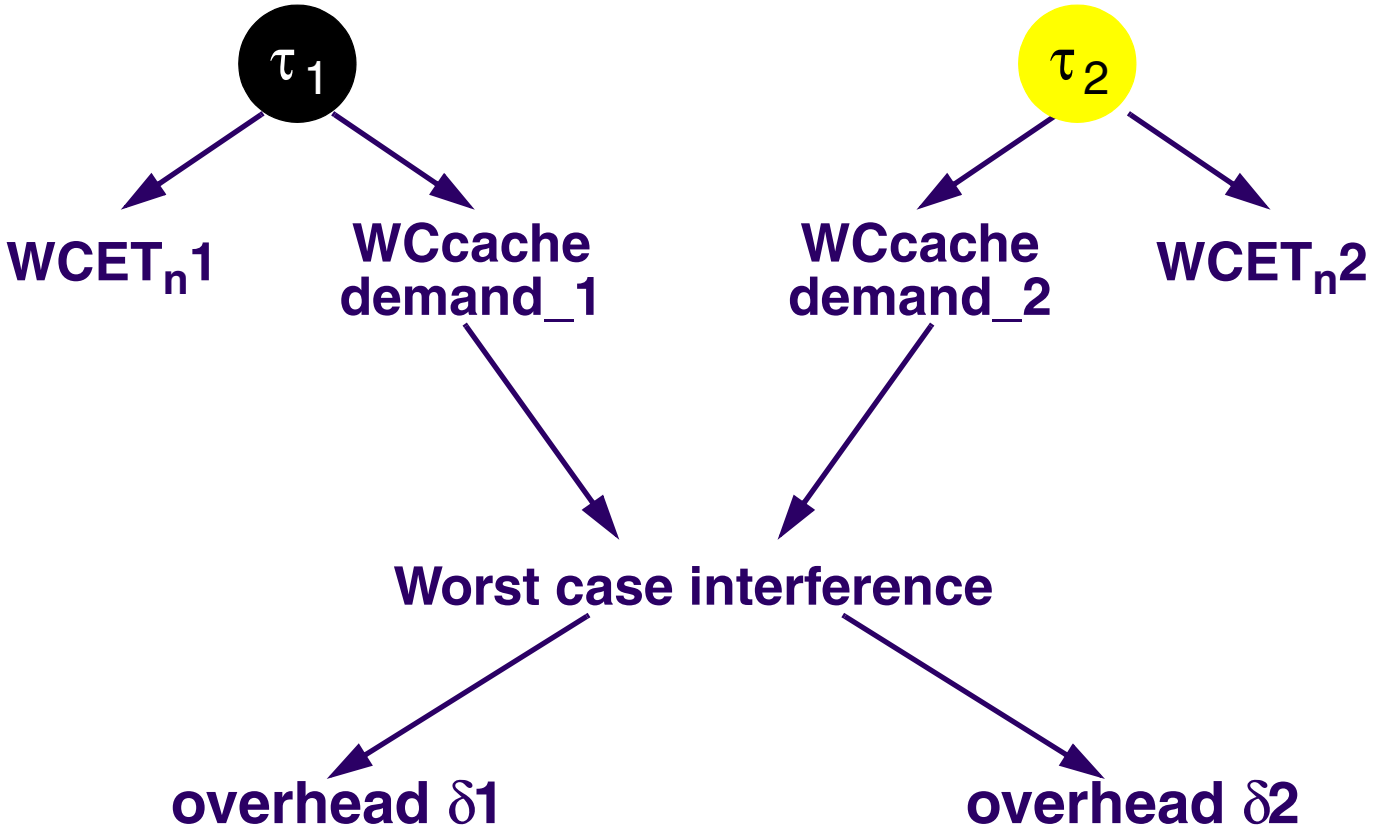
A Possible Way?



☞ **Problem**: What is the WCcache demand and $WCET_n$?

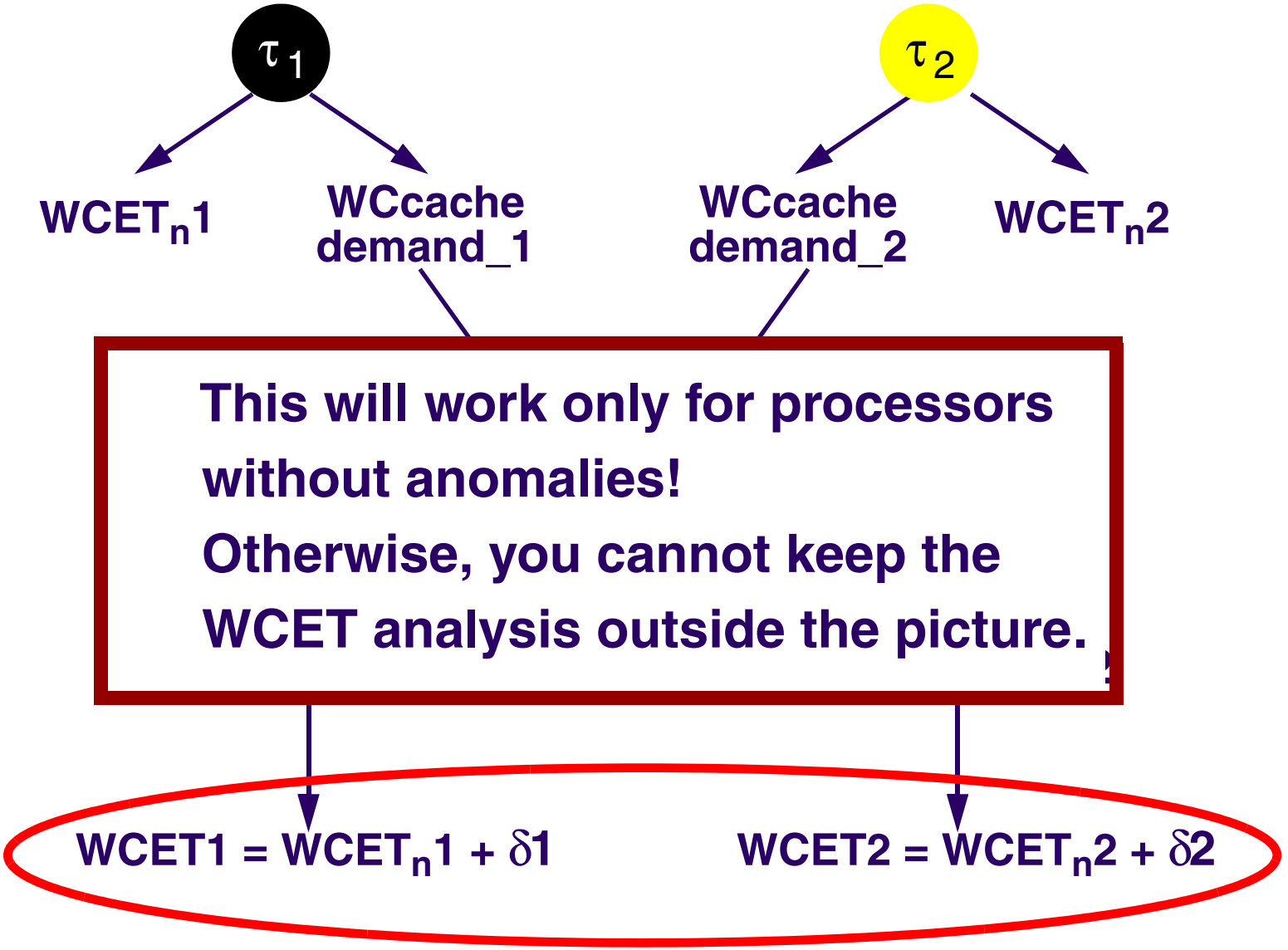
- The longest path is not necessarily the one with the most cache misses.

A Possible Way?



☞ **Problem:** How to determine the Worst case interference and the related penalty?

A Possible Way?

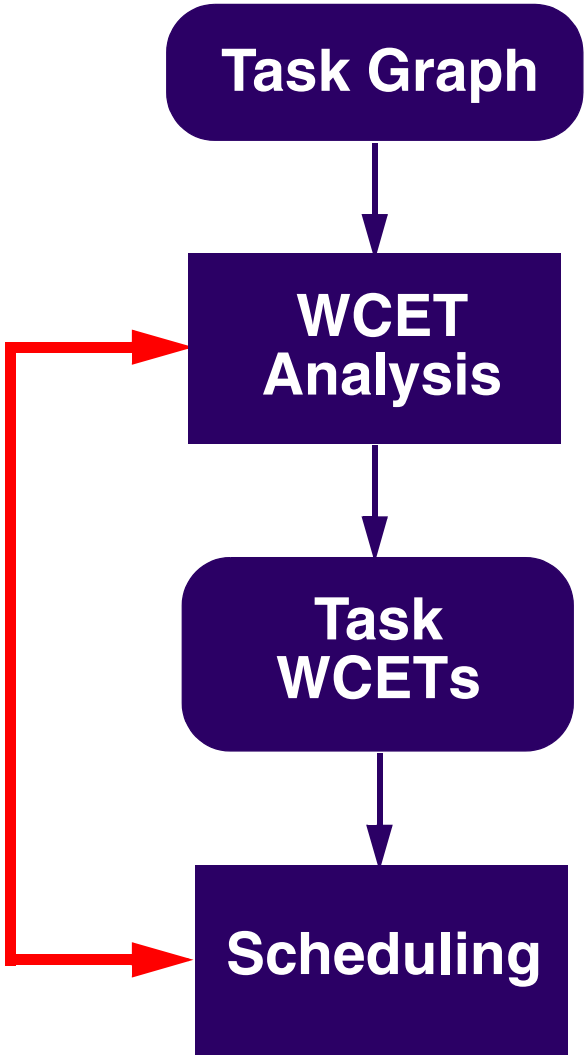


Proposed Approach

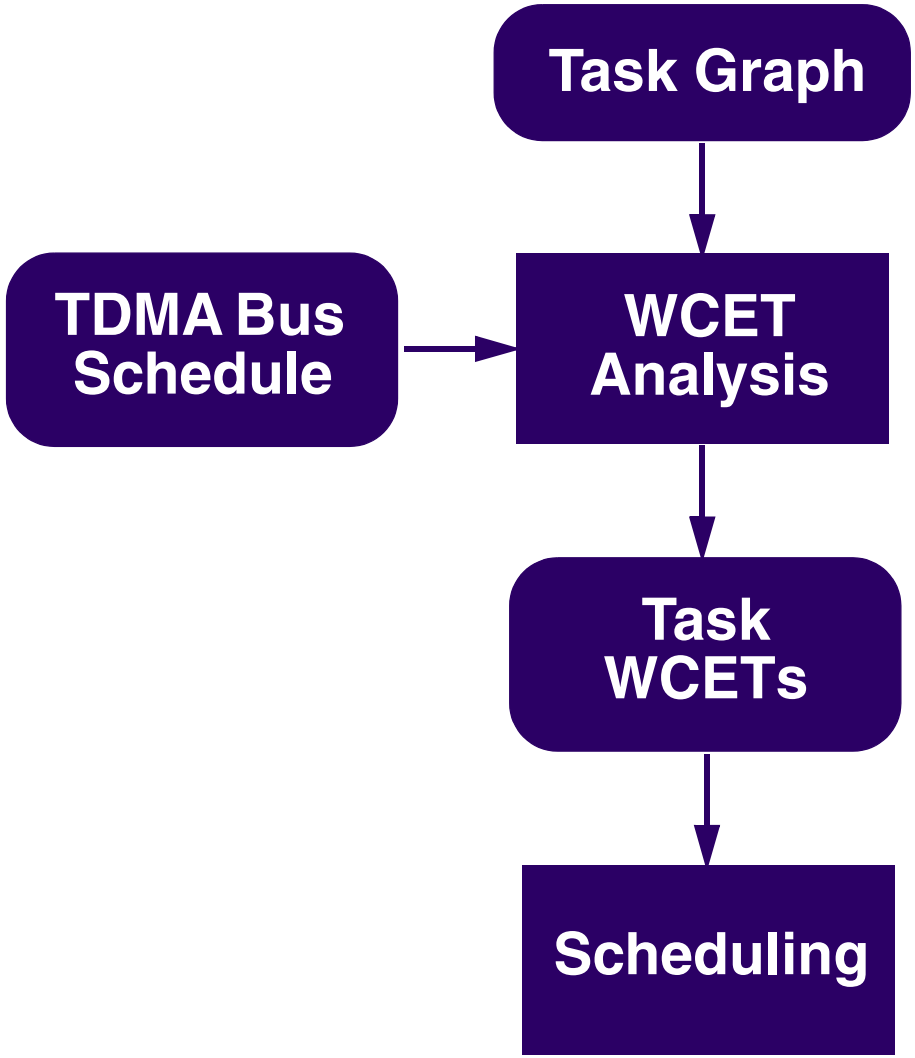
- The WCETs depend on the schedule (due to bus conflicts).
- The schedule depends on the WCETs.

☞ <u>Solution</u>:

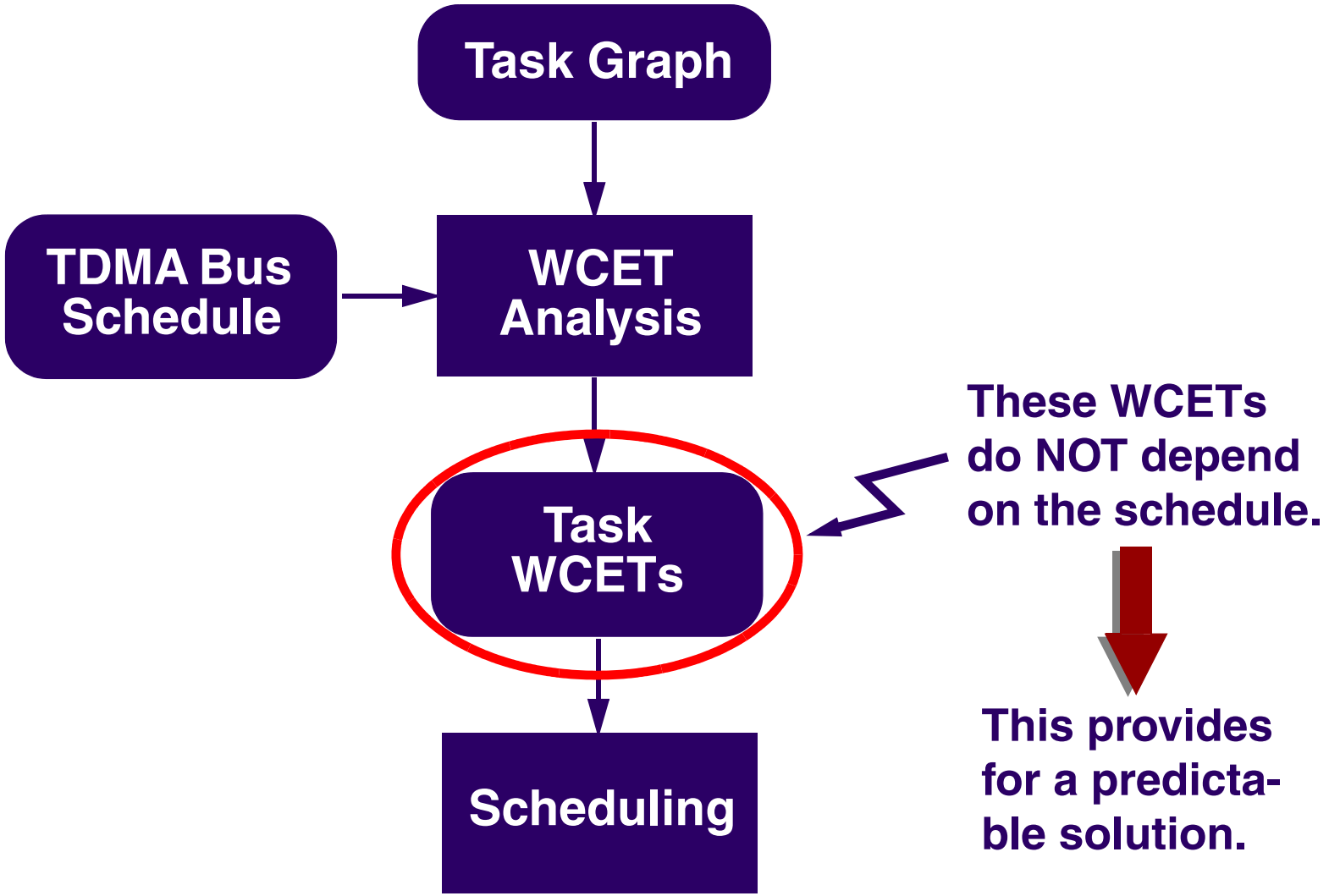
- Cut the circular dependency by using a *TDMA bus schedule.*



Proposed Approach

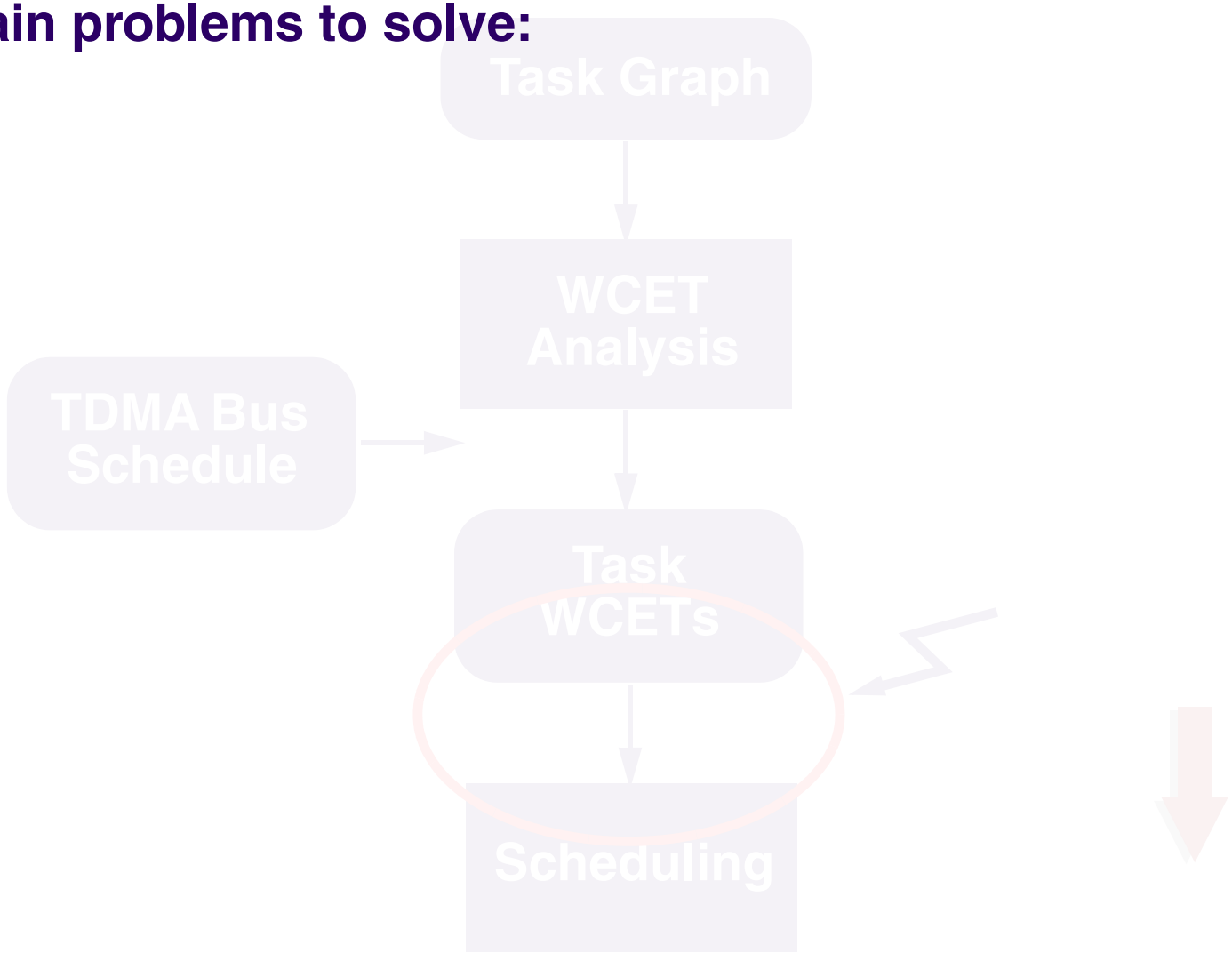


Proposed Approach



Proposed Approach

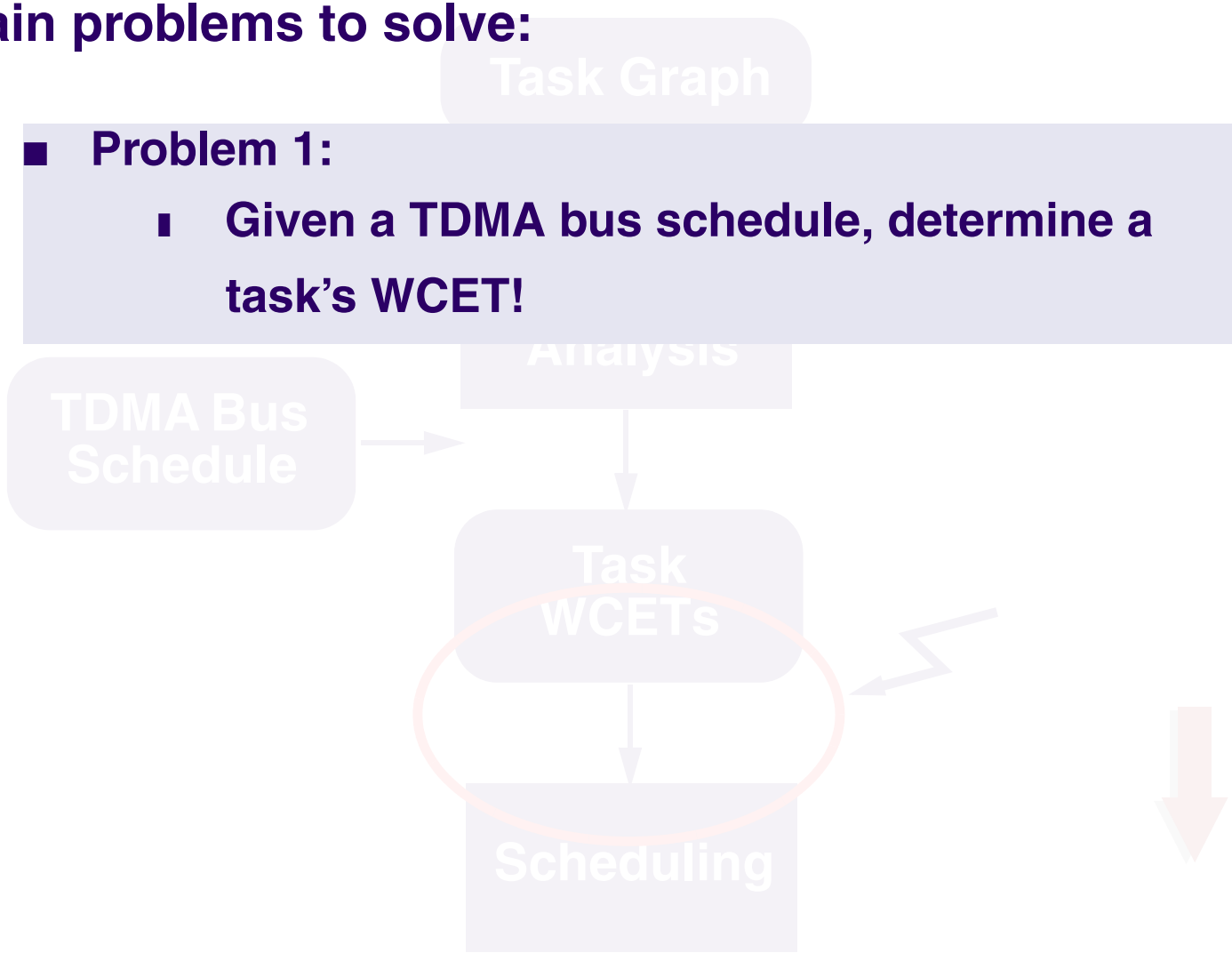
☞ **Three main problems to solve:**



Proposed Approach

☞ **Three main problems to solve:**

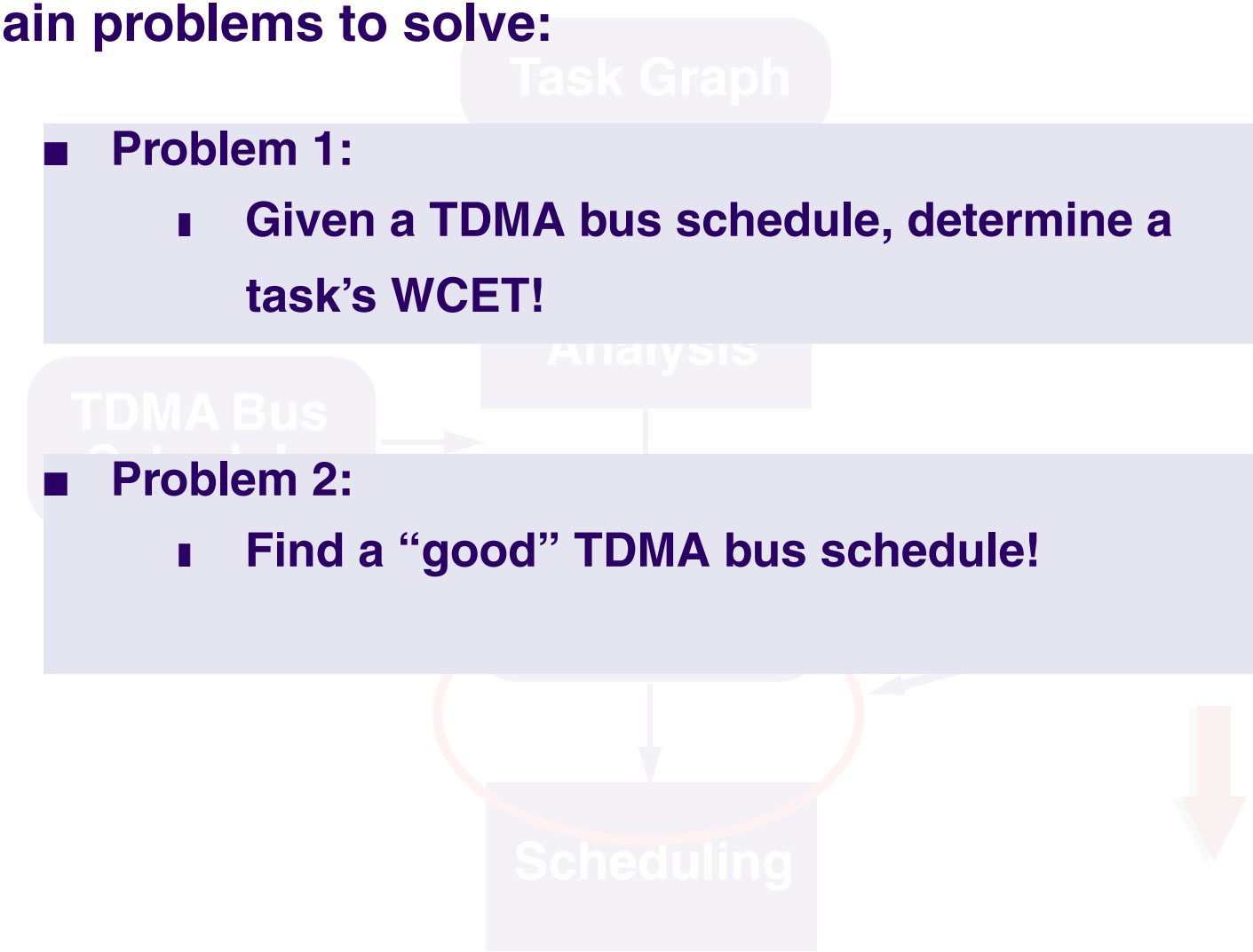
- **Problem 1:**
  - **Given a TDMA bus schedule, determine a task's WCET!**



Proposed Approach

☞ Three main problems to solve:

- Problem 1:
  - Given a TDMA bus schedule, determine a task's WCET!
- Problem 2:
  - Find a "good" TDMA bus schedule!



Proposed Approach

☞ Three main problems to solve:

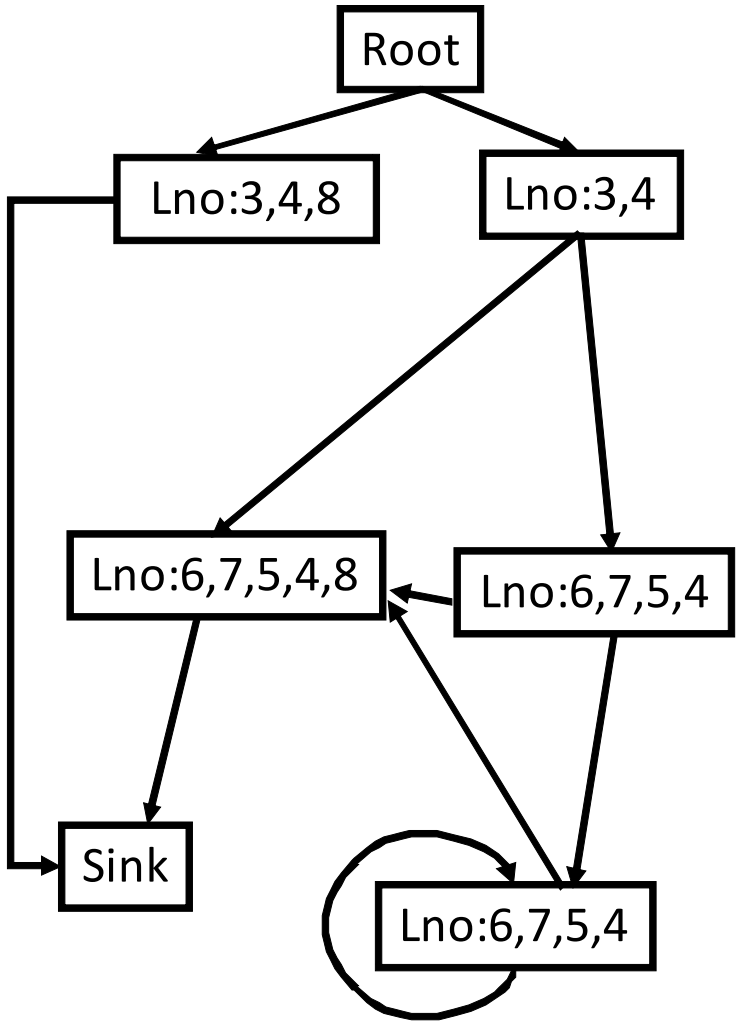
- Problem 1:
  - Given a TDMA bus schedule, determine a task's WCET!
- Problem 2:
  - Find a "good" TDMA bus schedule!
- Problem 3:
  - Produce a system schedule!

The WCET Analysis

```
1:void foo() {
2: int i, temp;
3: for (i=0;
4:       i<100;
5:       i++) {
6:   temp=a[i];
7:   a[temp]=0;
8: }
9:}
```

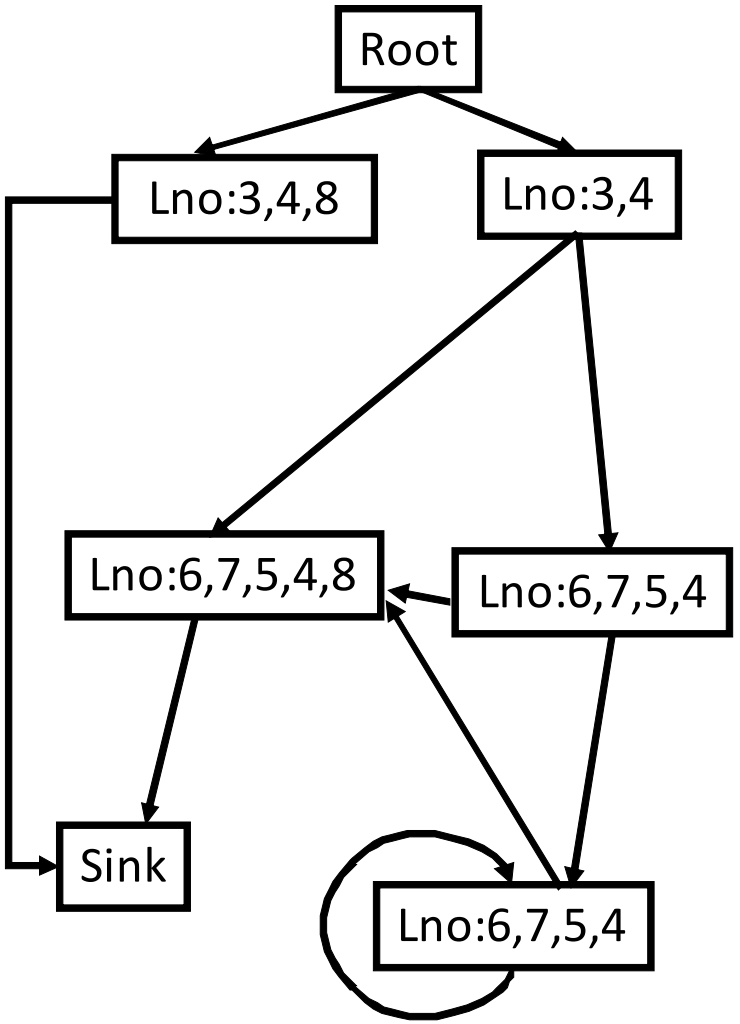
The WCET Analysis

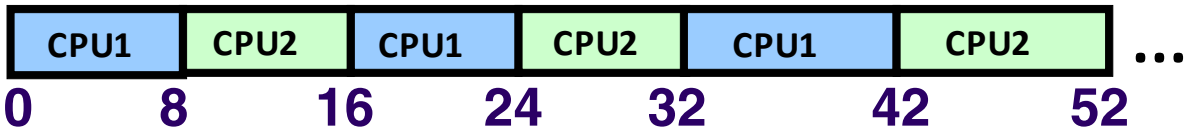
```
1:void foo() {
2: int i, temp;
3: for (i=0;
4:       i<100;
5:       i++) {
6:   temp=a[i];
7:   a[temp]=0;
8: }
9:}
```



The WCET Analysis

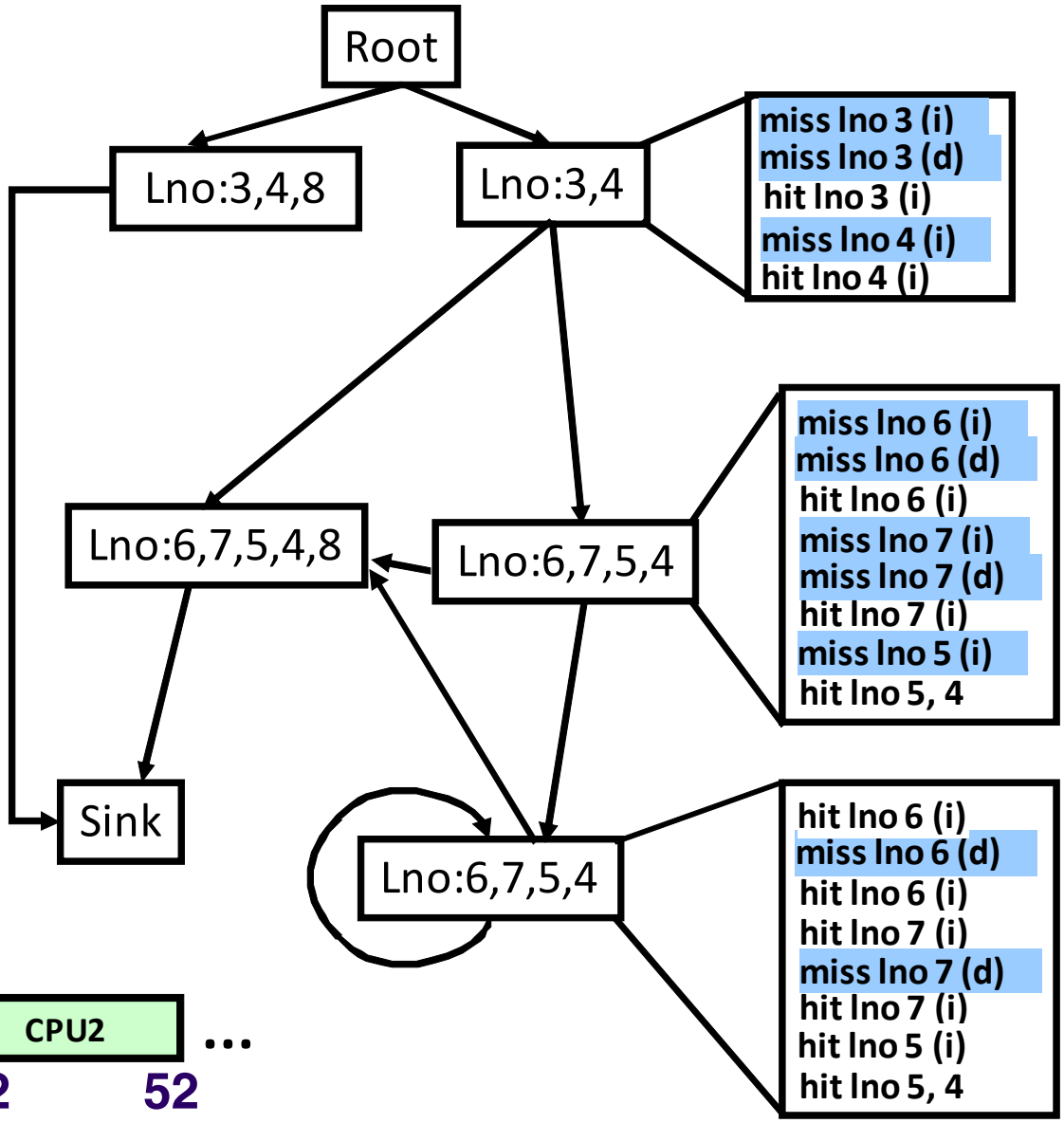
```
1:void foo() {
2: int i, temp;
3: for (i=0;
4:       i<100;
5:       i++) {
6:   temp=a[i];
7:   a[temp]=0;
8: }
9:}
```

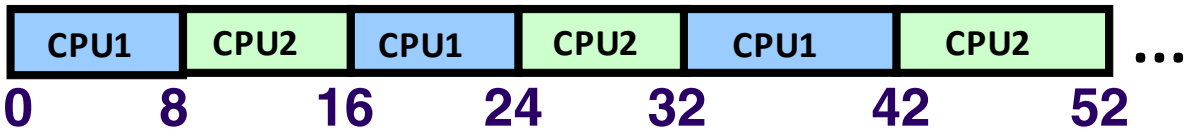




The WCET Analysis

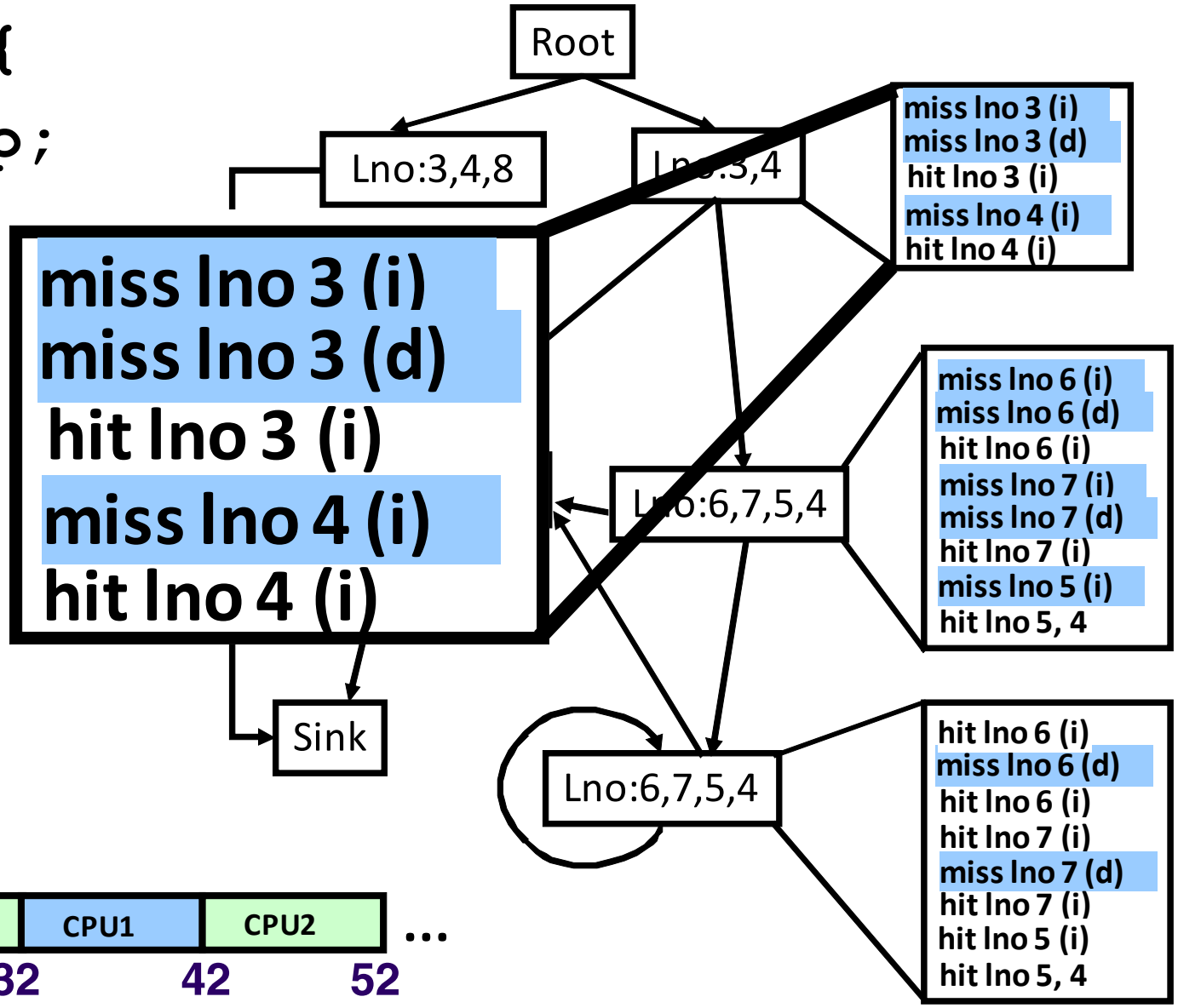
```
1:void foo() {
2:  int i, temp;
3:  for (i=0;
4:         i<100;
5:         i++) {
6:    temp=a[i];
7:    a[temp]=0;
8:  }
9:}
```

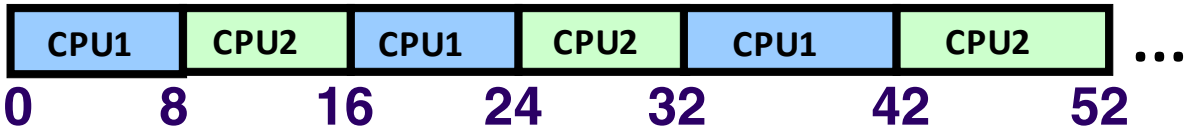




The WCET Analysis

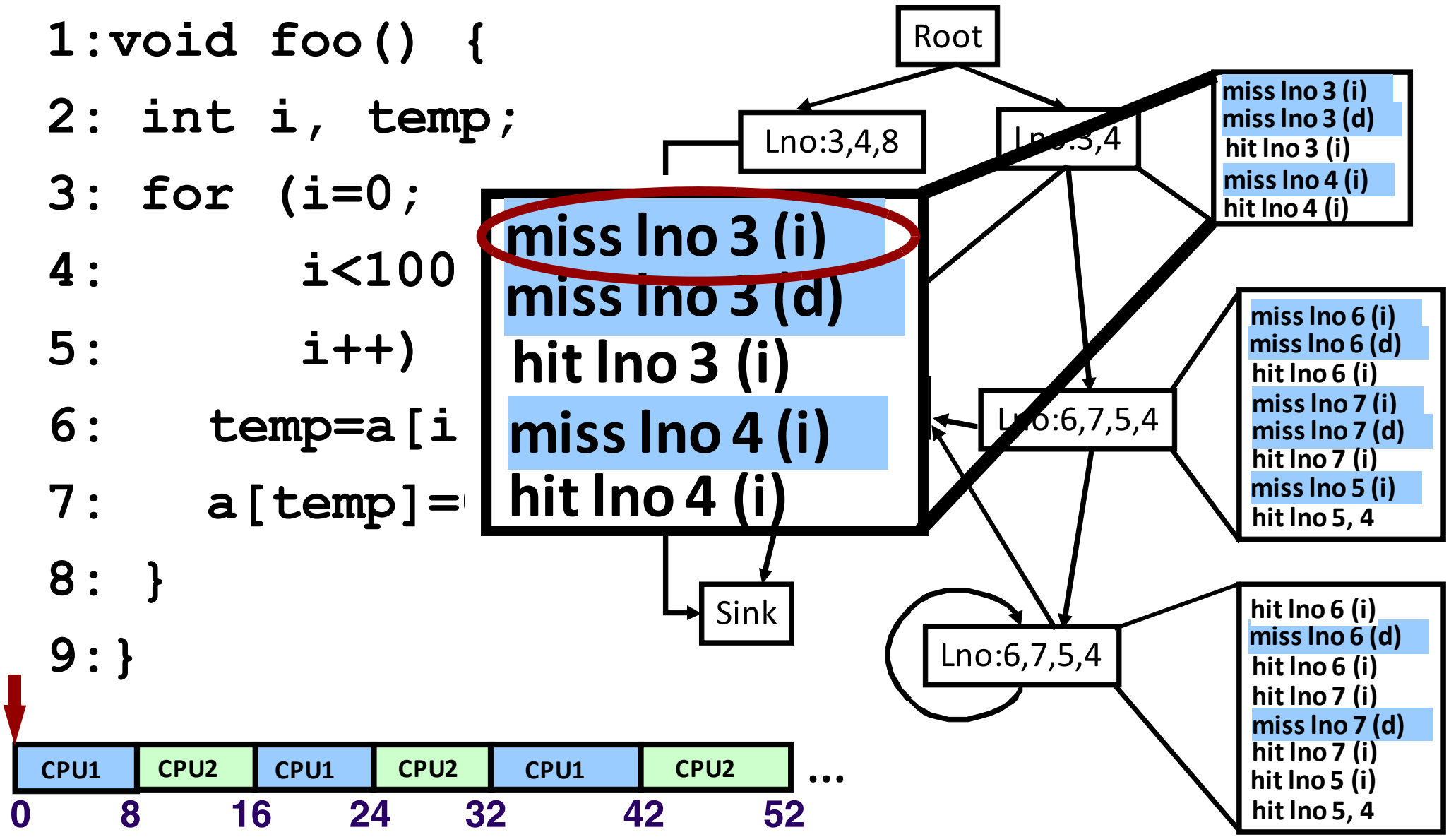
```
1:void foo() {
2:  int i, temp;
3:  for (i=0;
4:         i<100
5:         i++)
6:     temp=a[i
7:     a[temp]=
8:  }
9:}
```





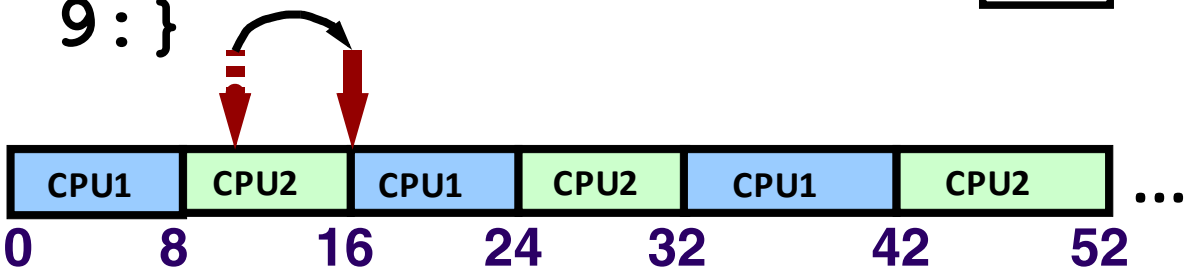
The WCET Analysis

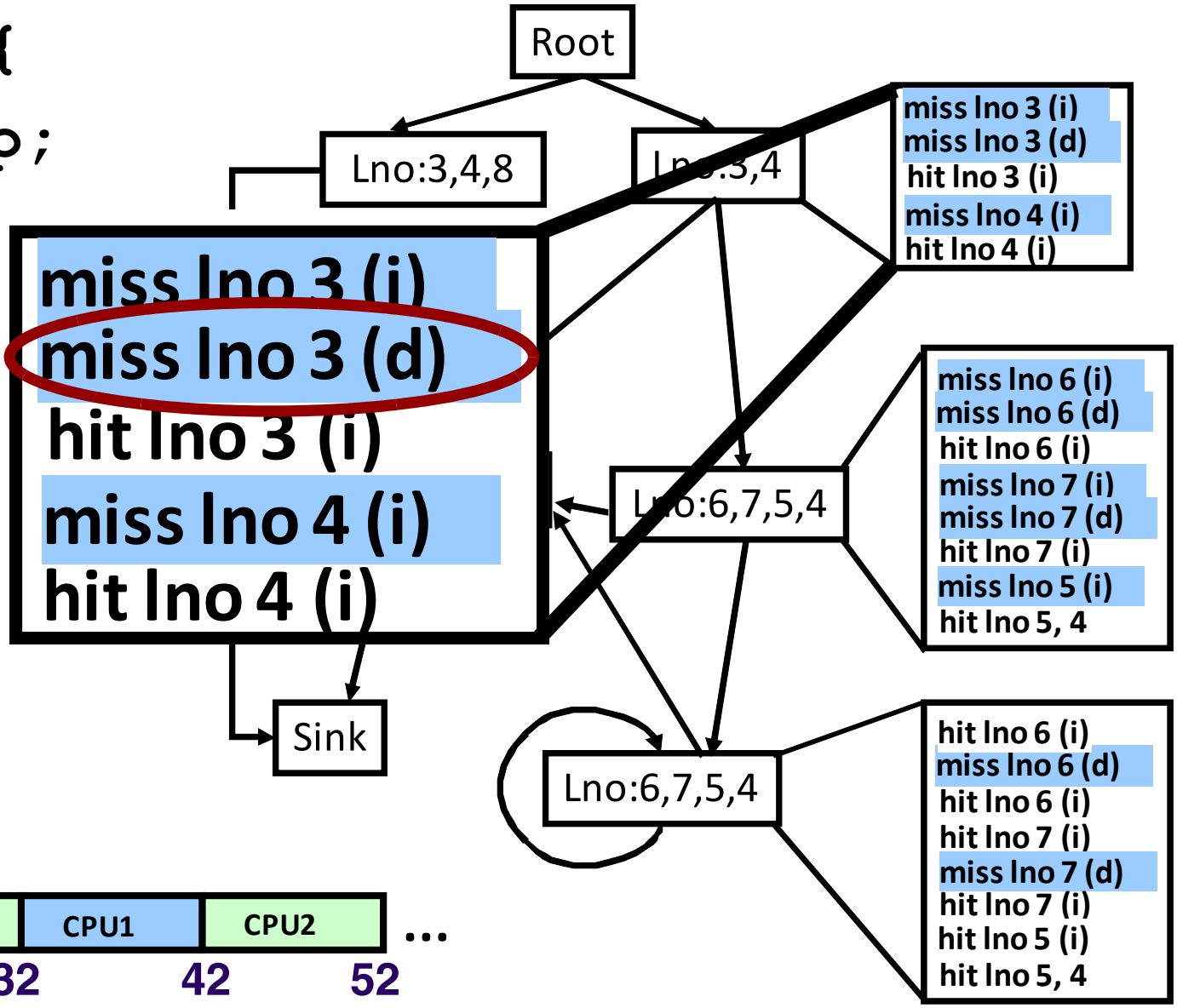
```
1:void foo() {
2:  int i, temp;
3:  for (i=0;
4:         i<100
5:         i++)
6:     temp=a[i
7:     a[temp]=
8:  }
9:}
```



The WCET Analysis

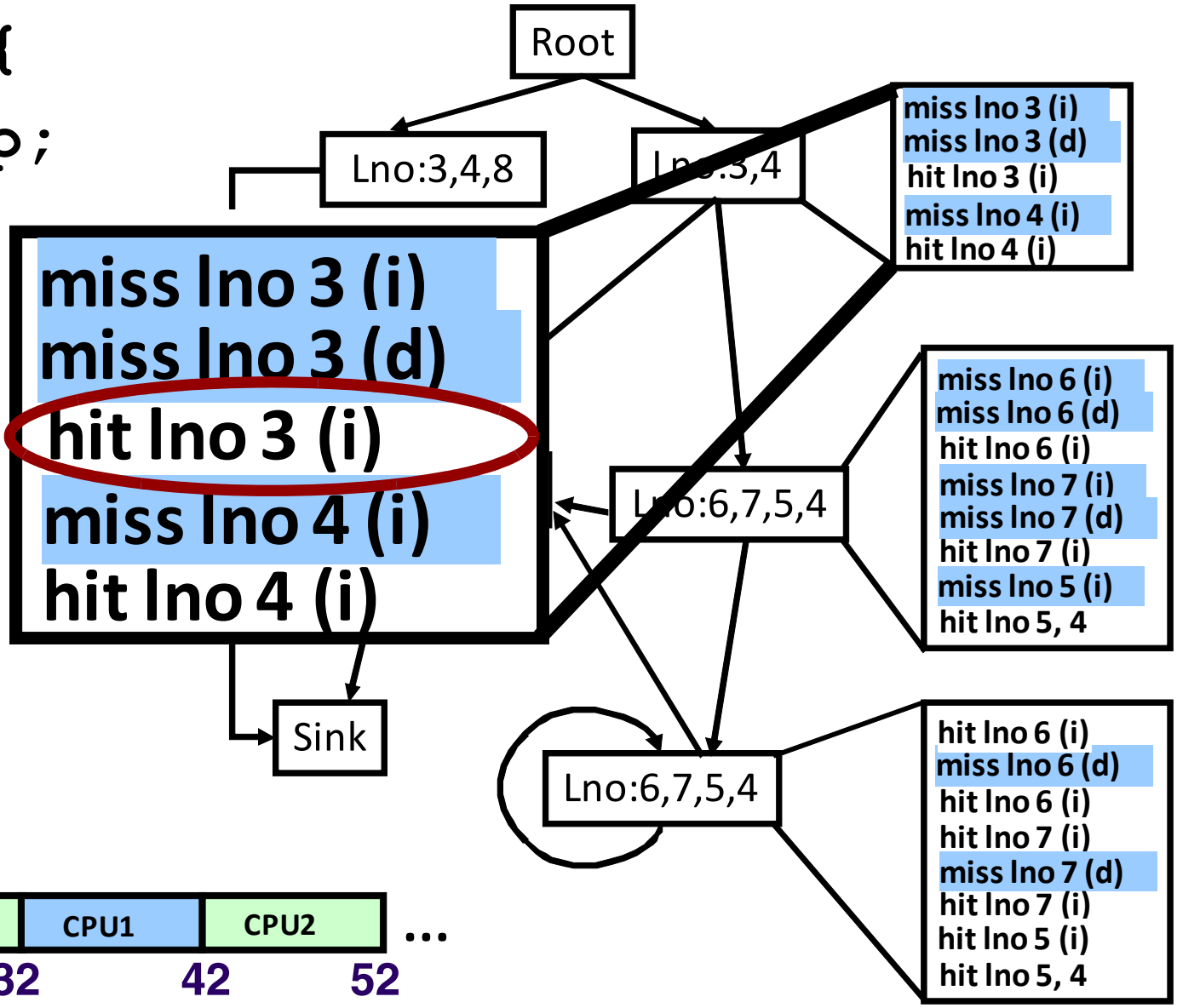
```
1:void foo() {
2:  int i, temp;
3:  for (i=0;
4:         i<100
5:         i++)
6:     temp=a[i
7:     a[temp]=
8:  }
9:}
```

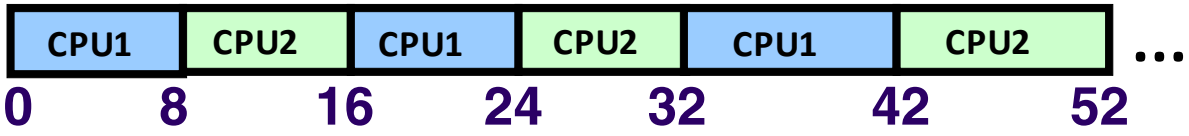




The WCET Analysis

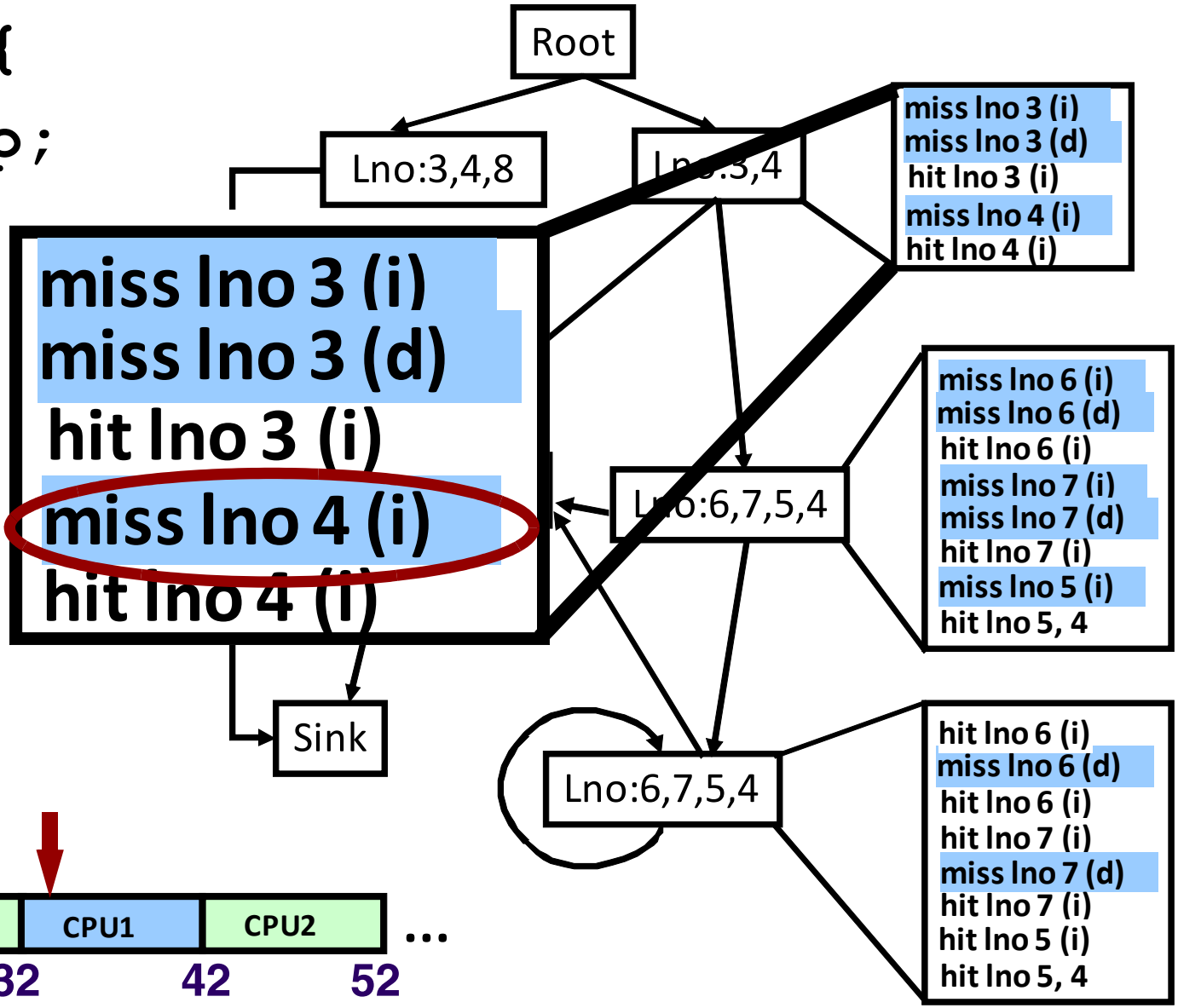
```
1:void foo() {
2:  int i, temp;
3:  for (i=0;
4:         i<100
5:         i++)
6:     temp=a[i
7:     a[temp]=
8:  }
9:}
```



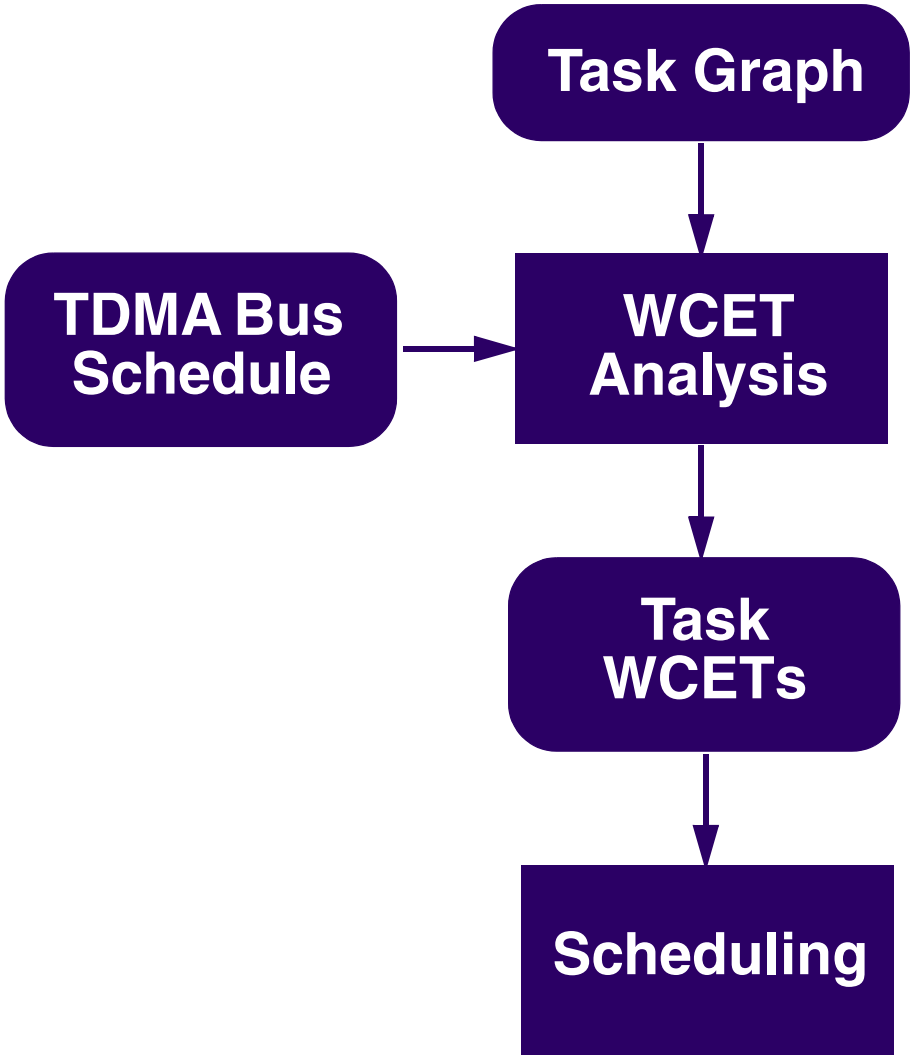


The WCET Analysis

```
1:void foo() {
2:  int i, temp;
3:  for (i=0;
4:         i<100
5:         i++)
6:     temp=a[i
7:     a[temp]=
8:  }
9:}
```

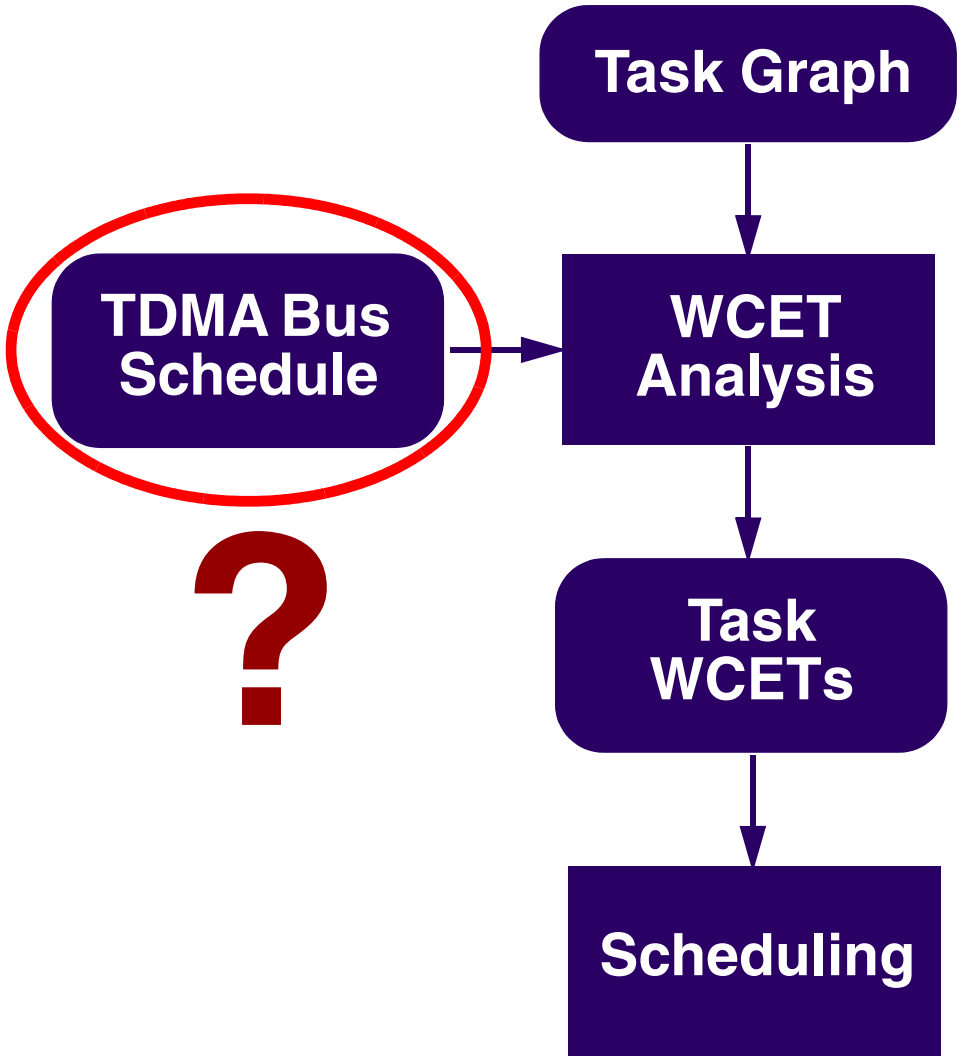


Back To The Big Picture



- Using the WCETs calculated with the given TDMA Bus schedule we can construct a safe system schedule.

Back To The Big Picture

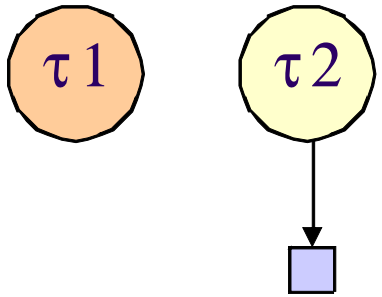


- Using the WCETs calculated with the given TDMA Bus schedule we can construct a safe system schedule.

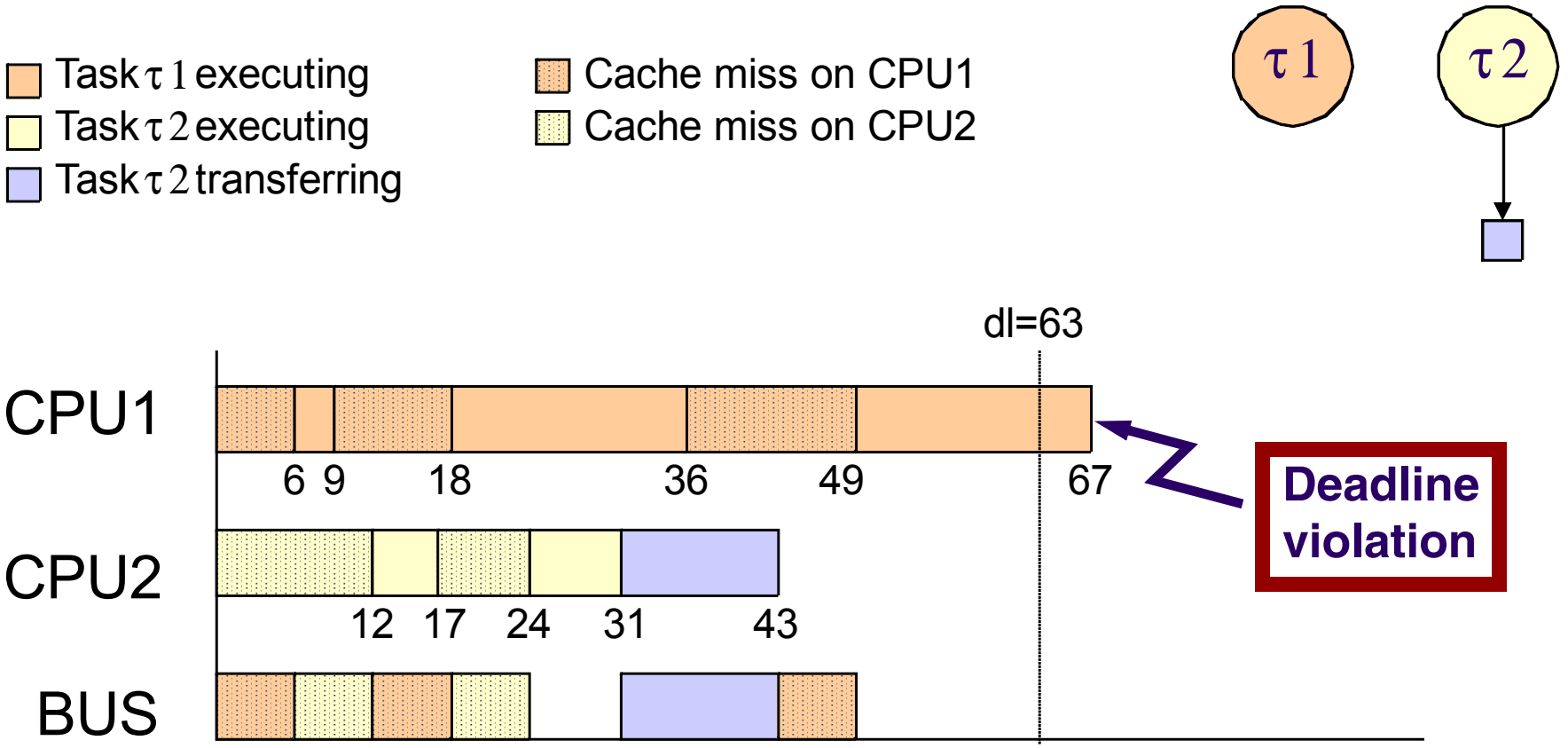
- How to determine the TDMA Bus schedule?

TDMA Bus Schedule

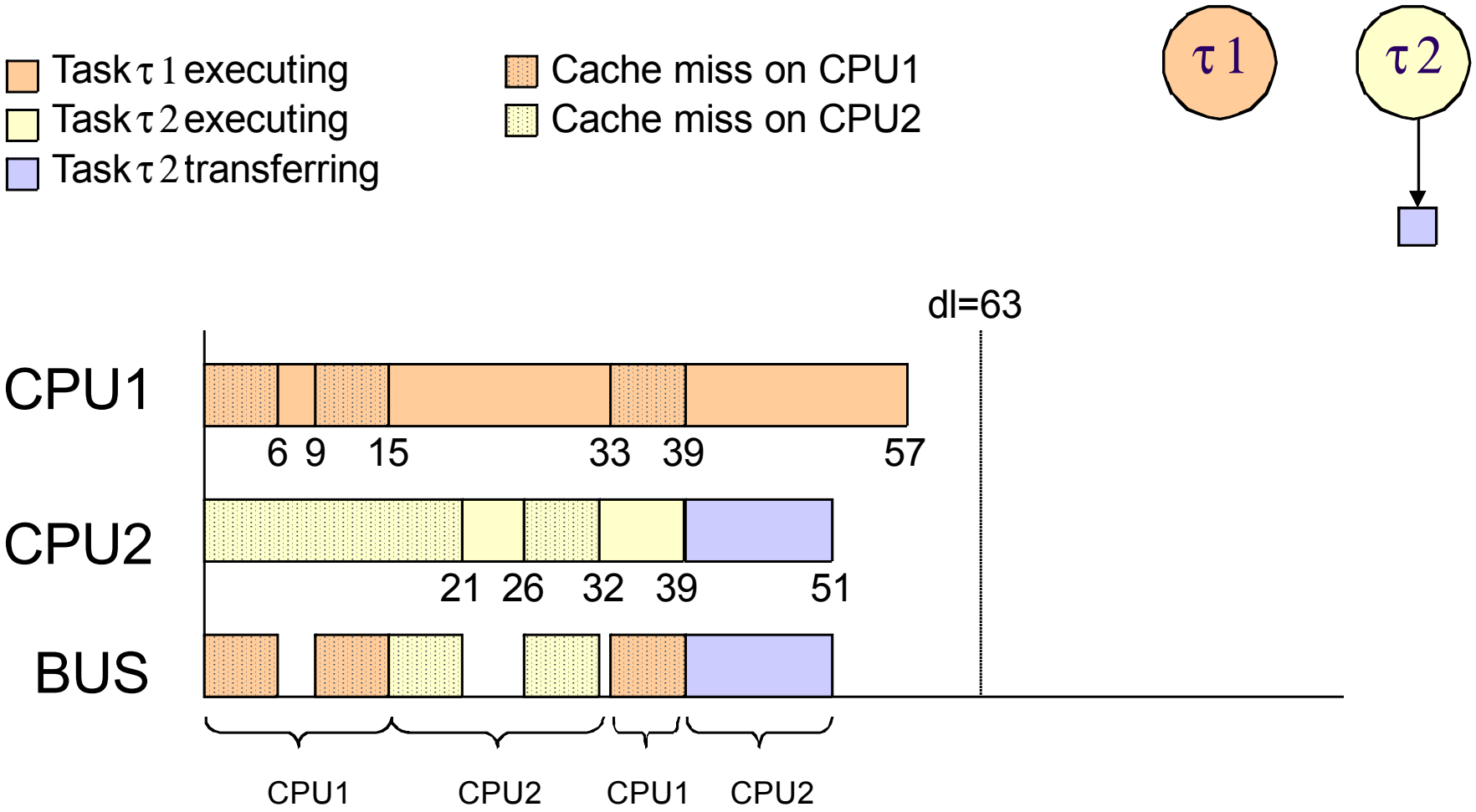
- WCETs - as from “traditional” WCET analysis:
  - t1: 57
  - t2: 24
  - tw: 12
- Deadline: 63



TDMA Bus Schedule



TDMA Bus Schedule



TDMA Bus Schedule

- Any TDMA Bus schedule provides a predictable, safe schedule.

- The bus schedule matters in terms of worst case schedule length!

TDMA Bus Schedule

- Any TDMA Bus schedule provides a predictable, safe schedule.

- The bus schedule matters in terms of worst case schedule length!

  - Problem:
    - Find a TDMA bus schedule which minimises the worst case schedule length!

TDMA Bus Schedule

- An ideal bus schedule would serve each miss as fast as possible.

- Adapt the bus schedule to the distribution of the cache misses as detected by the WCET analysis!

TDMA Bus Schedule

- An ideal bus schedule would serve each miss as fast as possible.

- Adapt the bus schedule to the distribution of the cache misses as detected by the WCET analysis!

- Tasks which are simultaneously active have bus access requirements that are conflicting.

TDMA Bus Schedule

- An ideal bus schedule would serve each miss as fast as possible.

- Adapt the bus schedule to the distribution of the cache misses as detected by the WCET analysis!

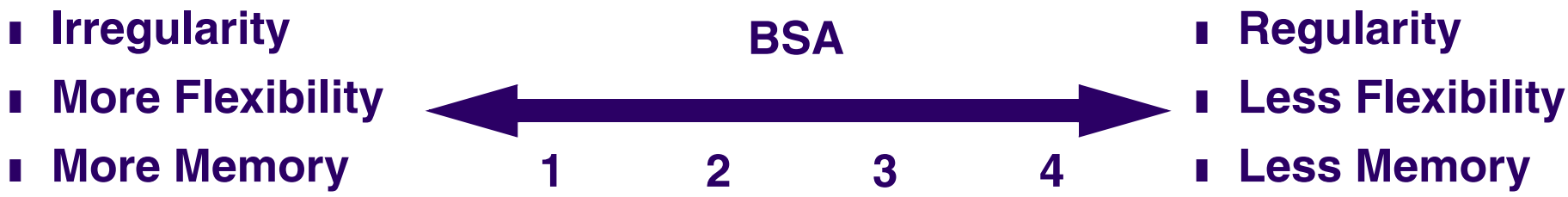
- Tasks which are simultaneously active have bus access requirements that are conflicting.

- Cache misses are distributed irregularly.

- A bus schedule following this irregularity would consume very much memory in the bus controller.

TDMA Bus Schedule Approaches

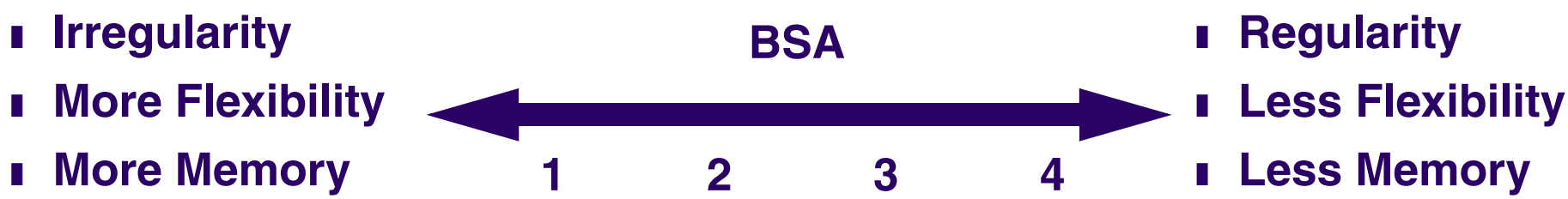
- Four TDMA Bus Schedule Approaches:
  - BSA1
  - BSA2
  - BSA3
  - BSA4



TDMA Bus Schedule Approaches

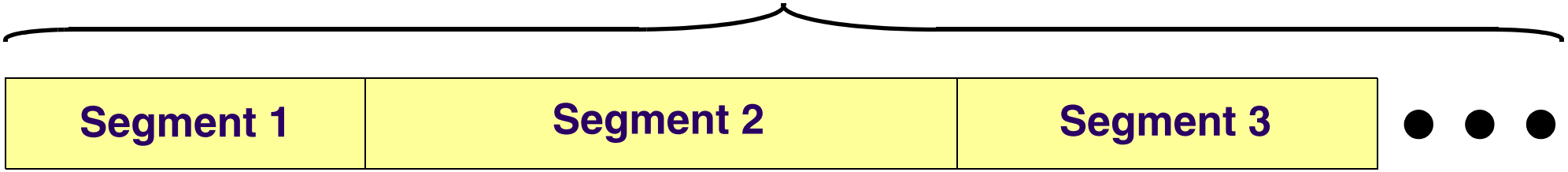
- Four TDMA Bus Schedule Approaches:

  - You have to pay for predictability:
    - Performance
    - Hardware overhead

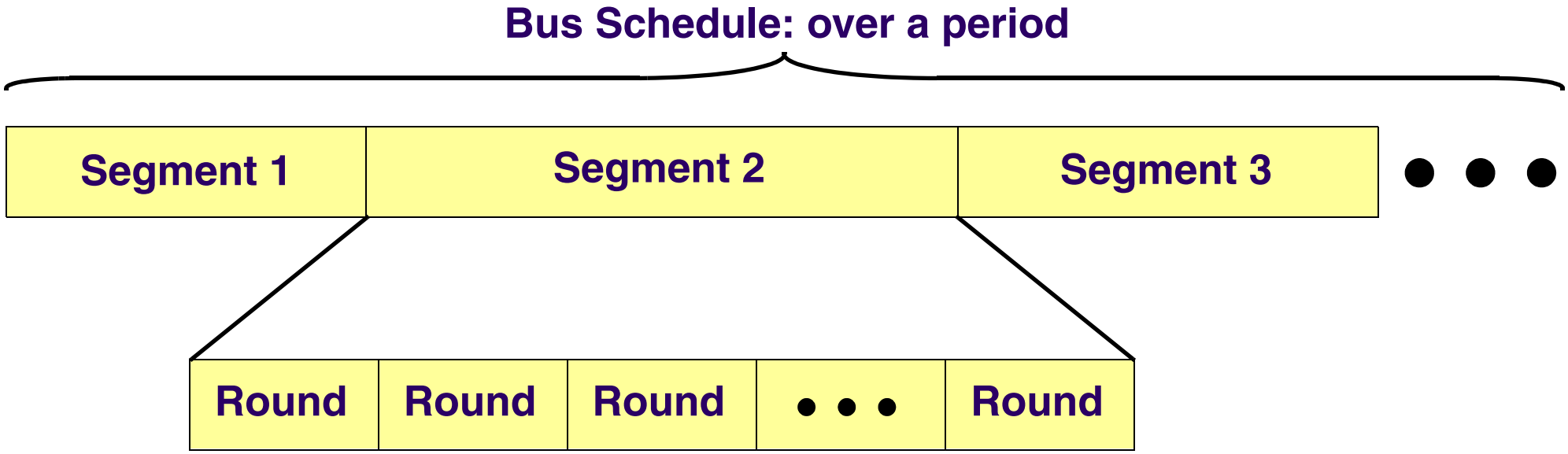


Bus Schedule

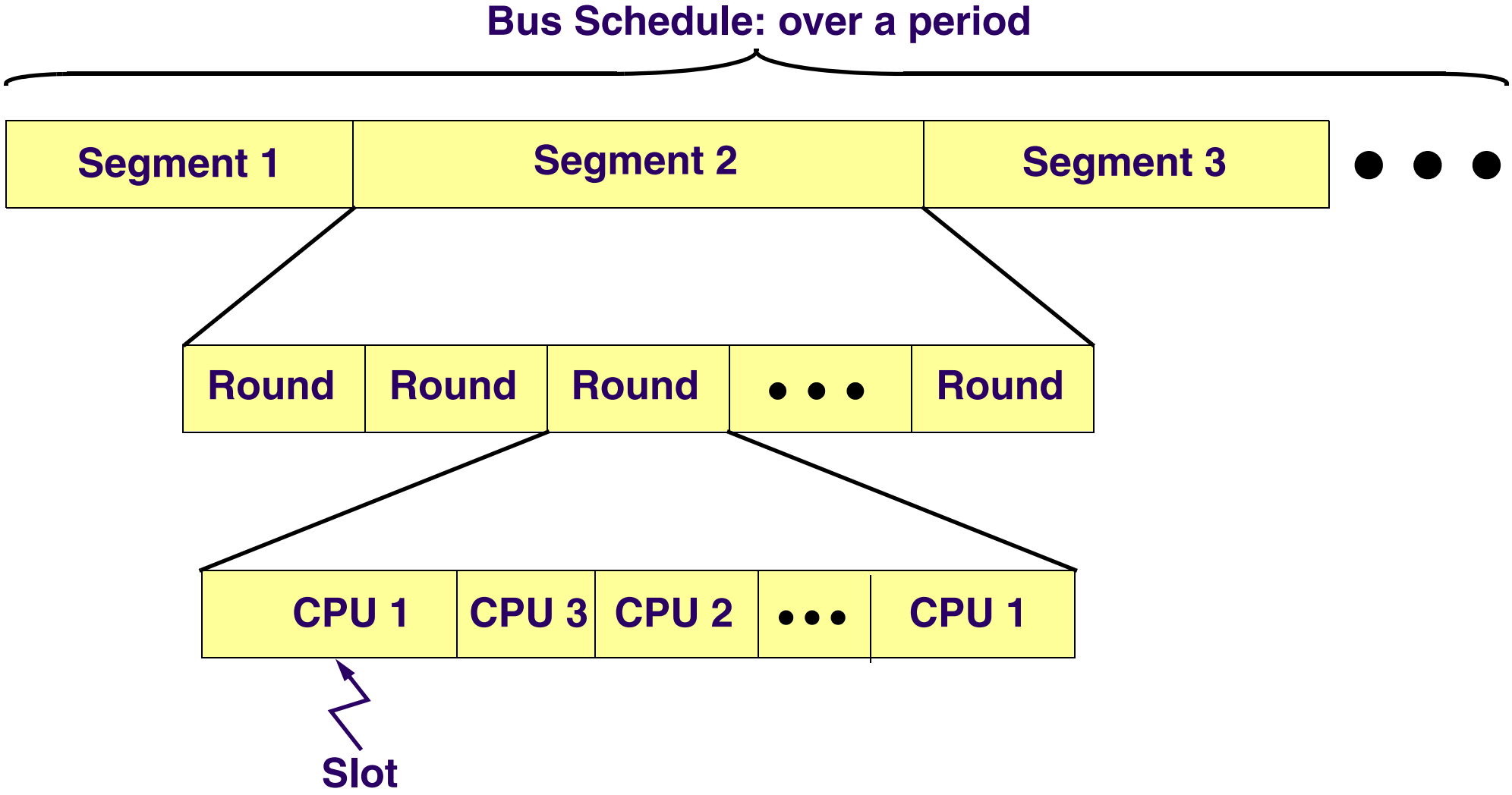
Bus Schedule: over a period



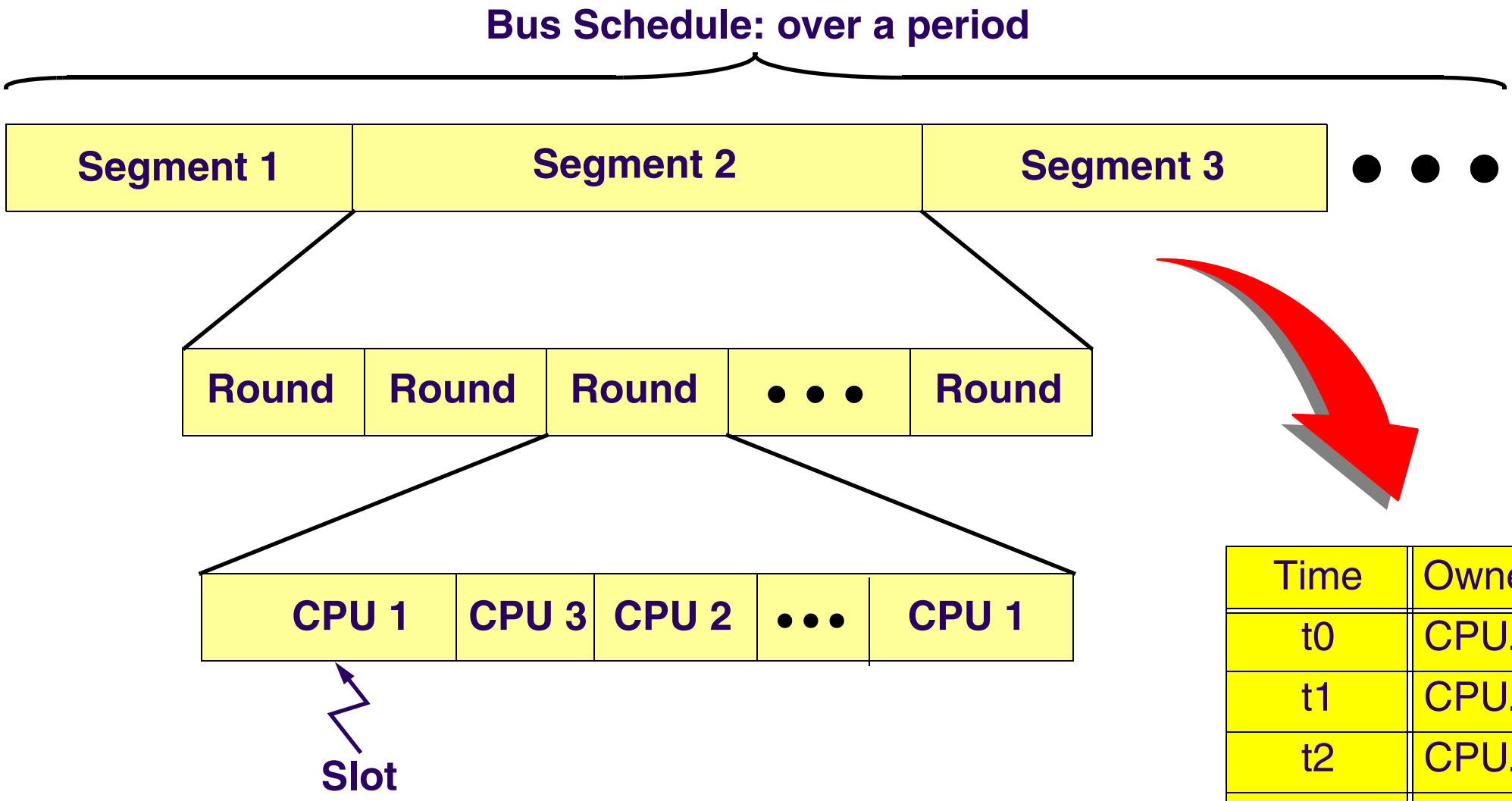
Bus Schedule



Bus Schedule



Bus Schedule



Time	Owner
t0	CPU...
t1	CPU...
t2	CPU...
...	...

☞ The most general, unrestricted, bus schedule:

- Composed of one single segment
  - Consisting of one single round
    - Consisting of an arbitrary sequence of slots of arbitrary size.

- The most general, unrestricted, bus schedule:

- Composed of one single segment
  - Consisting of one single round
    - Consisting of an arbitrary sequence of slots of arbitrary size.

- Can, potentially, be very well customised to the actual cache miss profile.
- Huge memory requirements!

- Not realistic but a good reference for comparison

- **Restriction:**
  - Each processor has maximum one slot in a round.

☞ Slot sizes in a round and their order are arbitrary.

☞ Rounds belonging to different segments are different.

- **Restriction:**
  - Each processor has maximum one slot in a round (BSA2).
  - All slots in a round must have the same size.

☞ **Rounds belonging to different segments are different.**

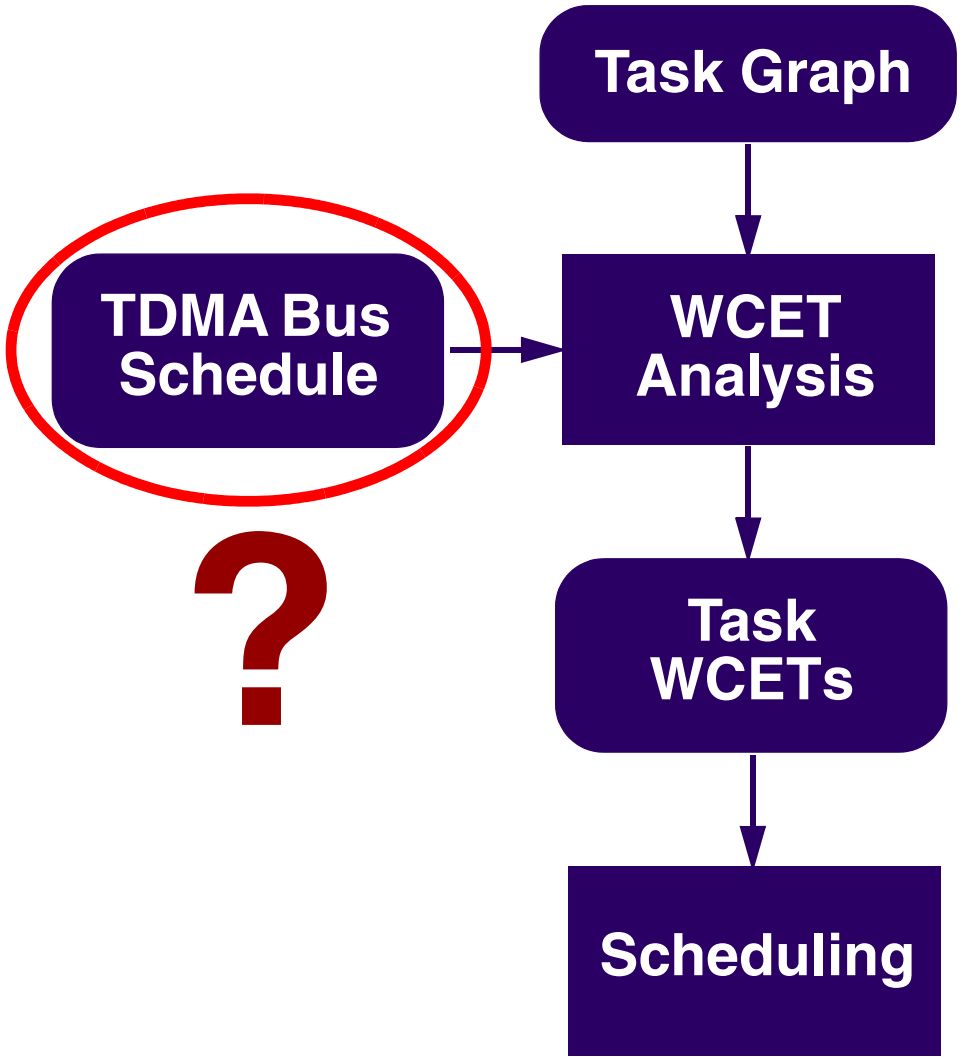
BSA4

- Restriction:
  - Each processor has maximum one slot in a round (BSA2).
  - All slots in a round must have the same size (BSA3).
  - All rounds, in all segments, are identical.

☞ The same round repeats over the whole bus schedule.

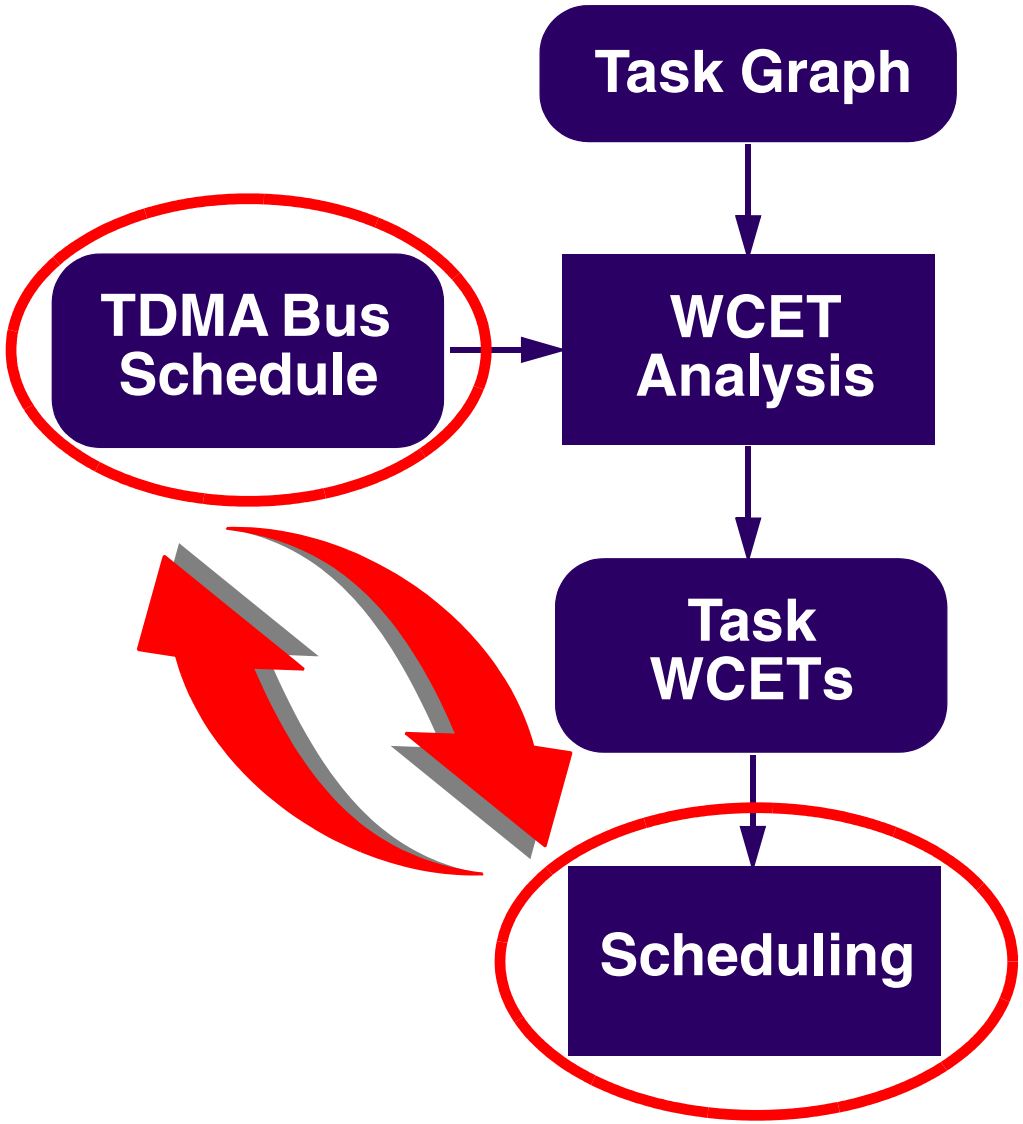
☞ This is an *extremely* constrained approach!

Back To The Big Picture



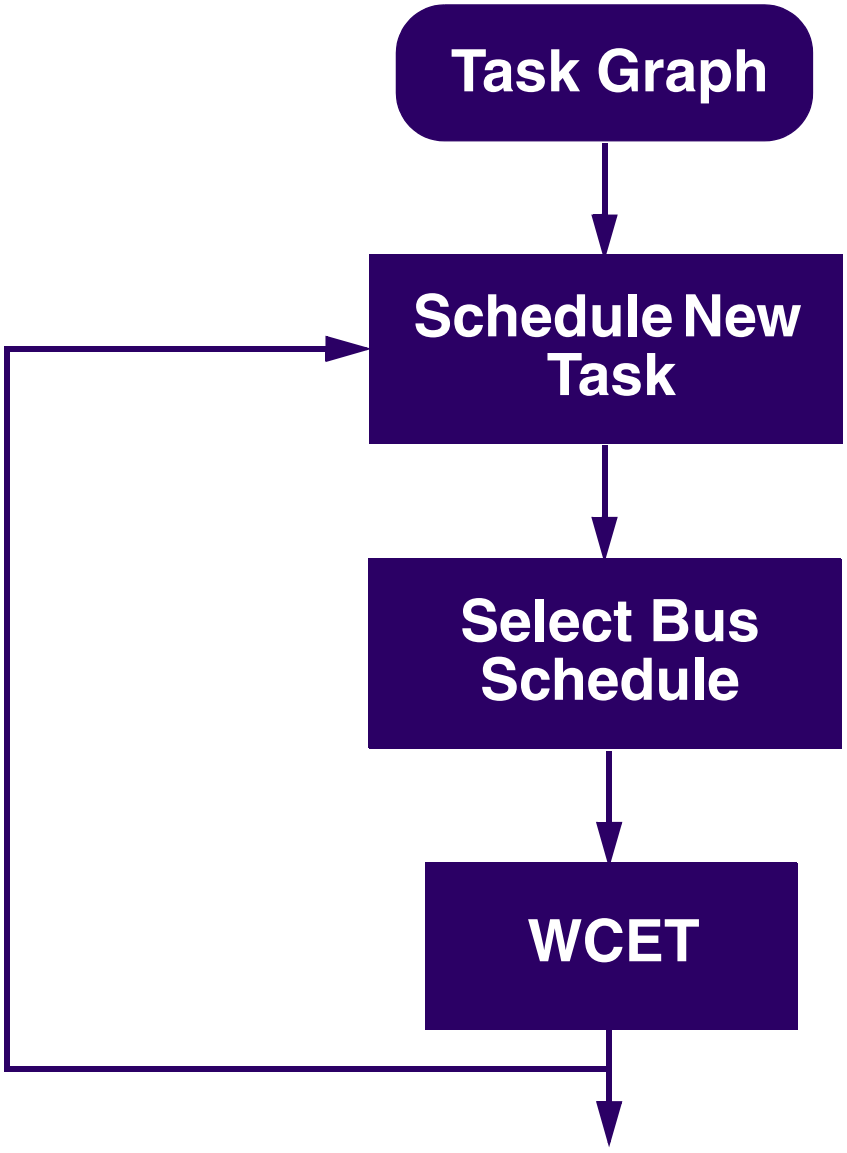
- Using the WCETs calculated with the given TDMA Bus schedule we can construct a safe system schedule.
- How to determine the TDMA Bus schedule?

Back To The Big Picture



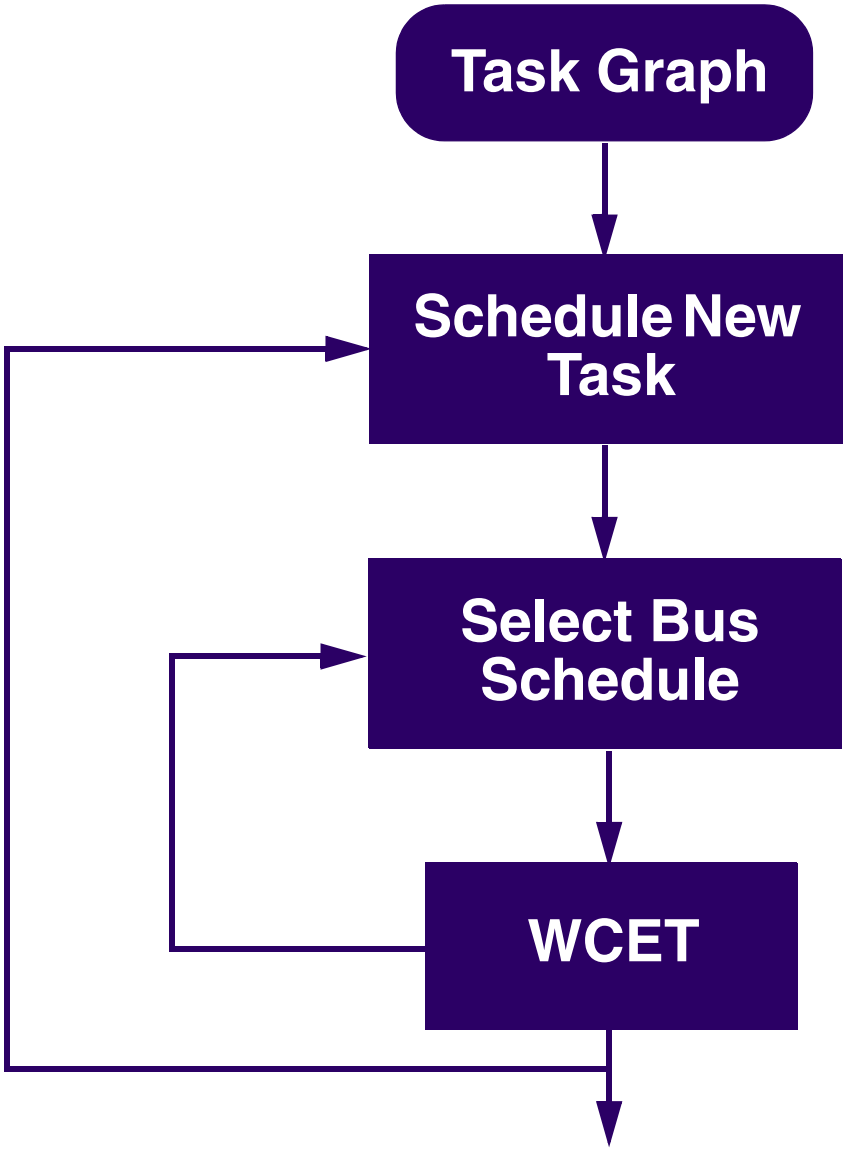
- Using the WCETs calculated with the given TDMA Bus schedule we can construct a safe system schedule.
- How to determine the TDMA Bus schedule?
- The bus schedule is determined simultaneously with the task graph scheduling.

Back To The Big Picture



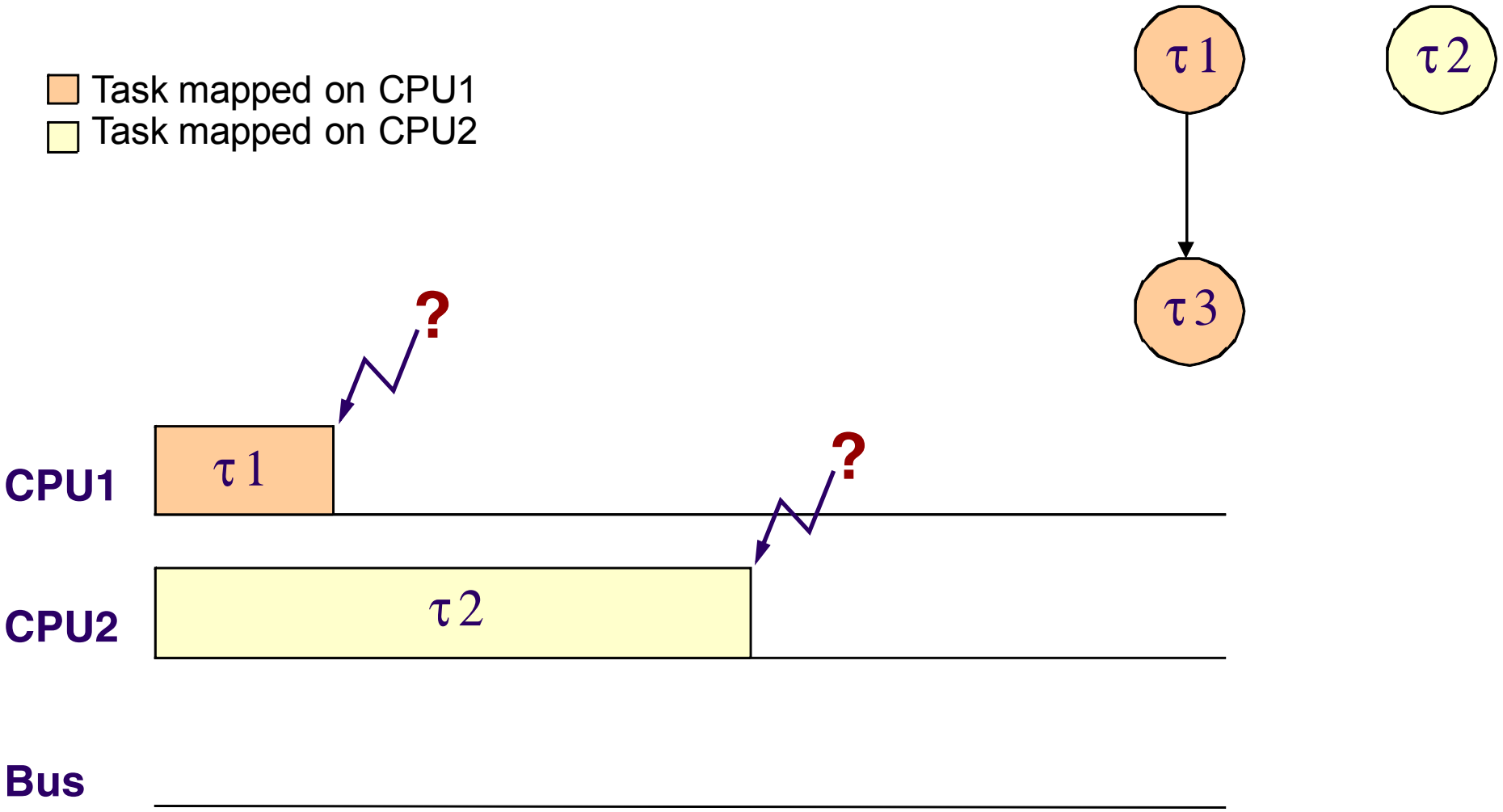
☞ The bus schedule is determined simultaneously with the task graph scheduling.

Back To The Big Picture

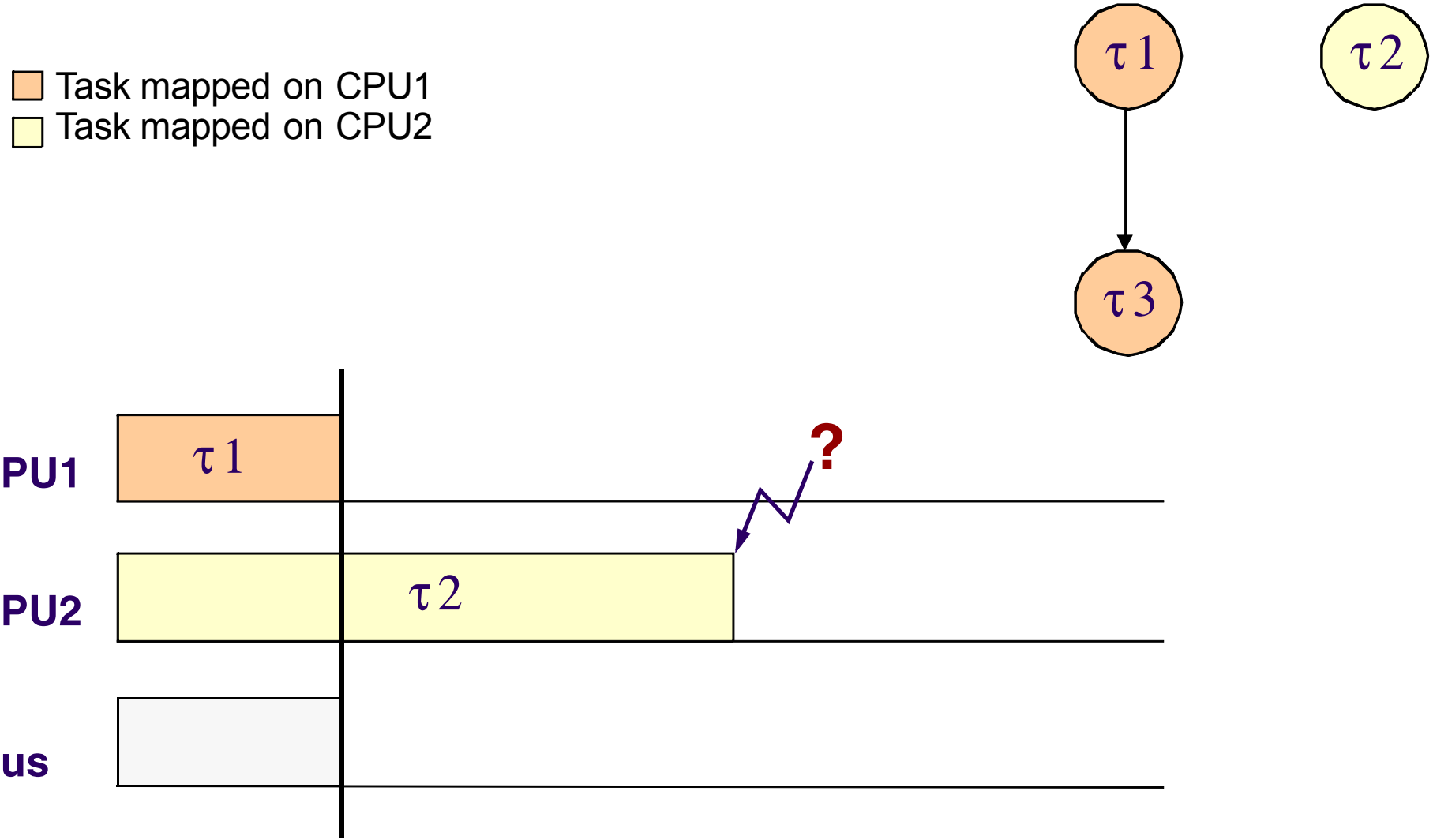


☞ The bus schedule is determined simultaneously with the task graph scheduling.

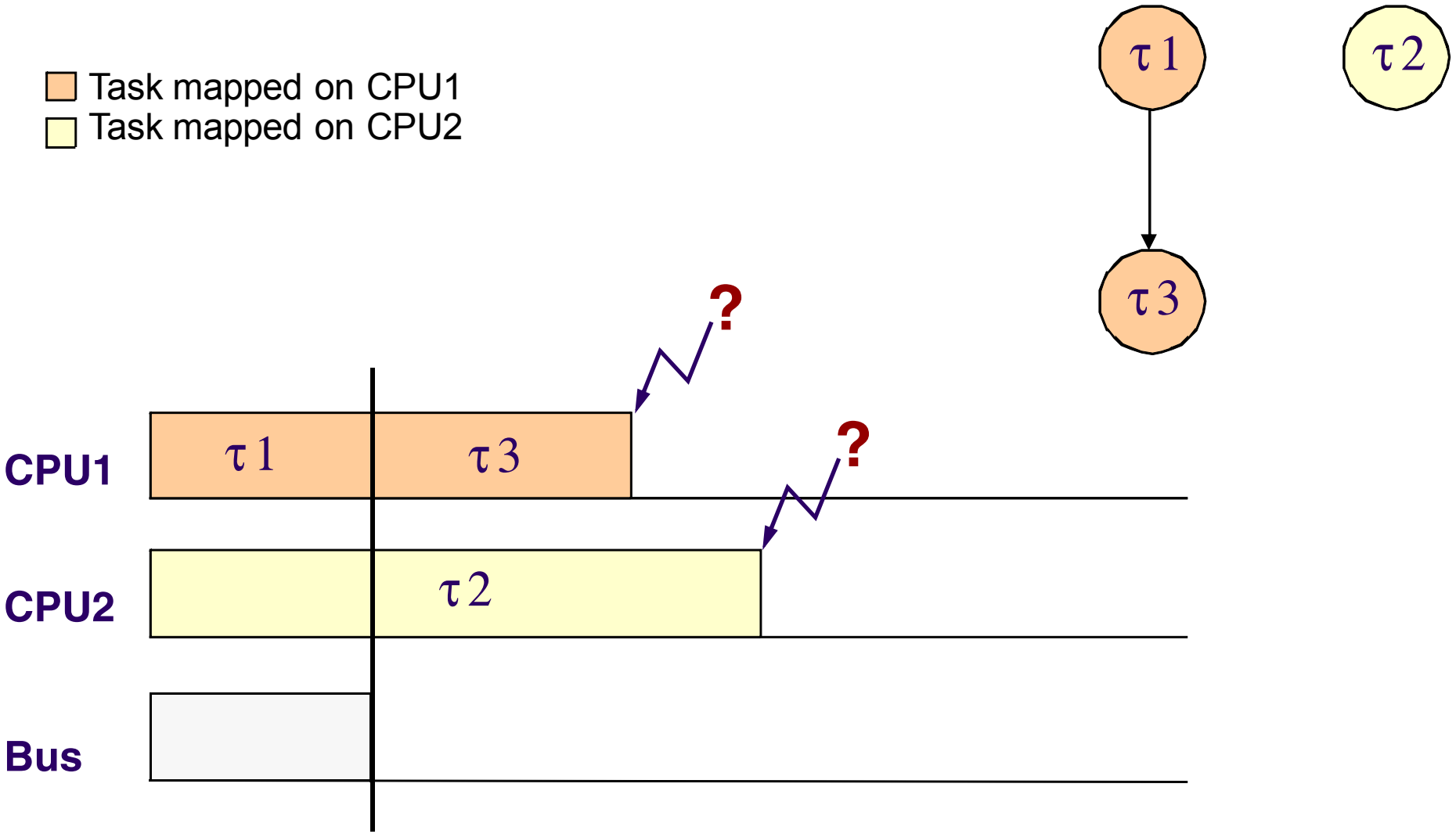
The Big Picture: Example



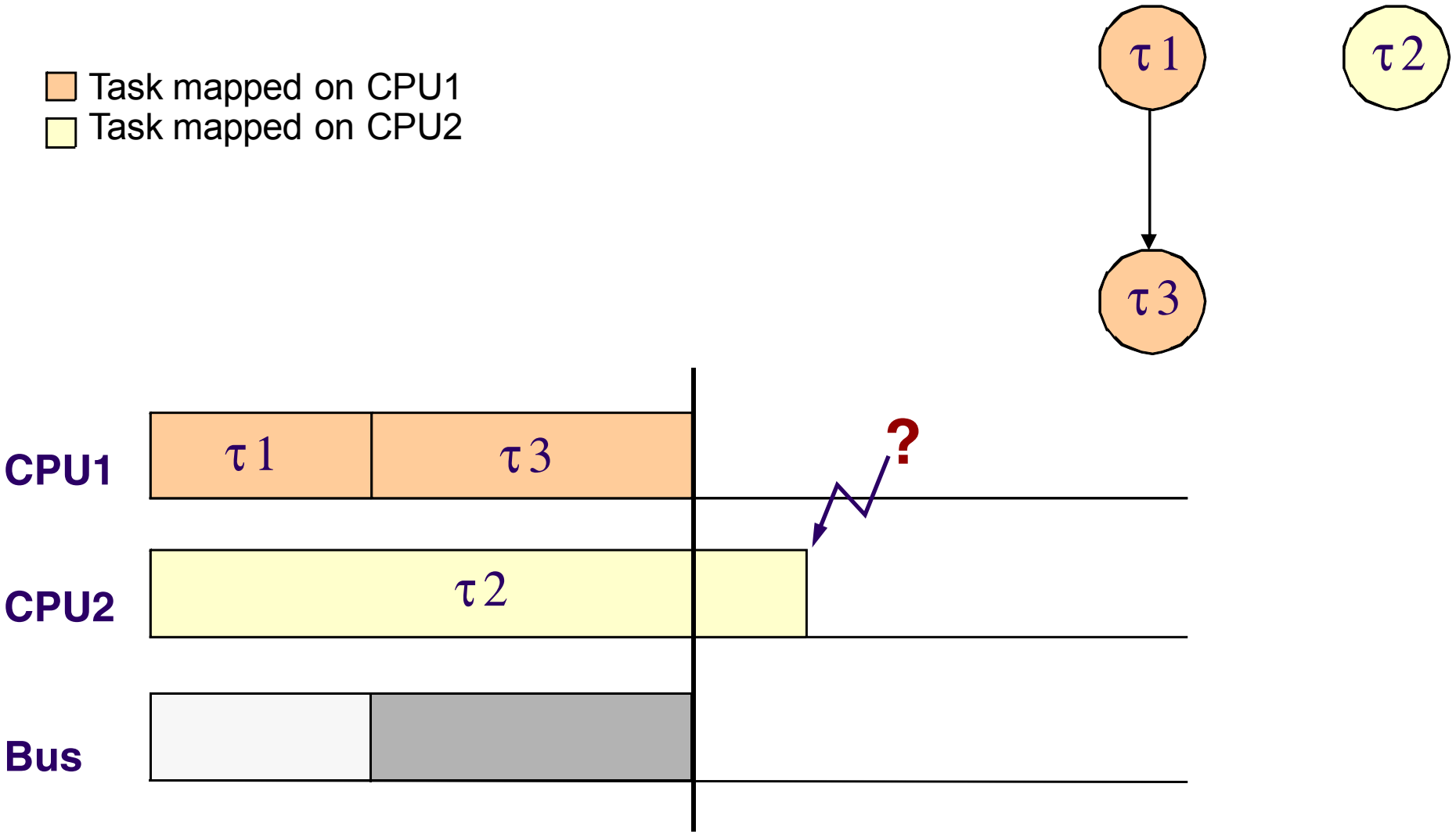
The Big Picture: Example



The Big Picture: Example

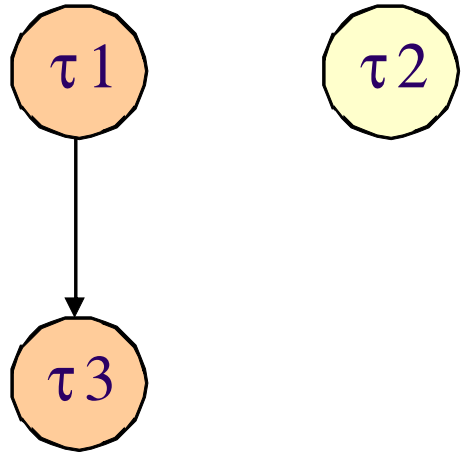


The Big Picture: Example

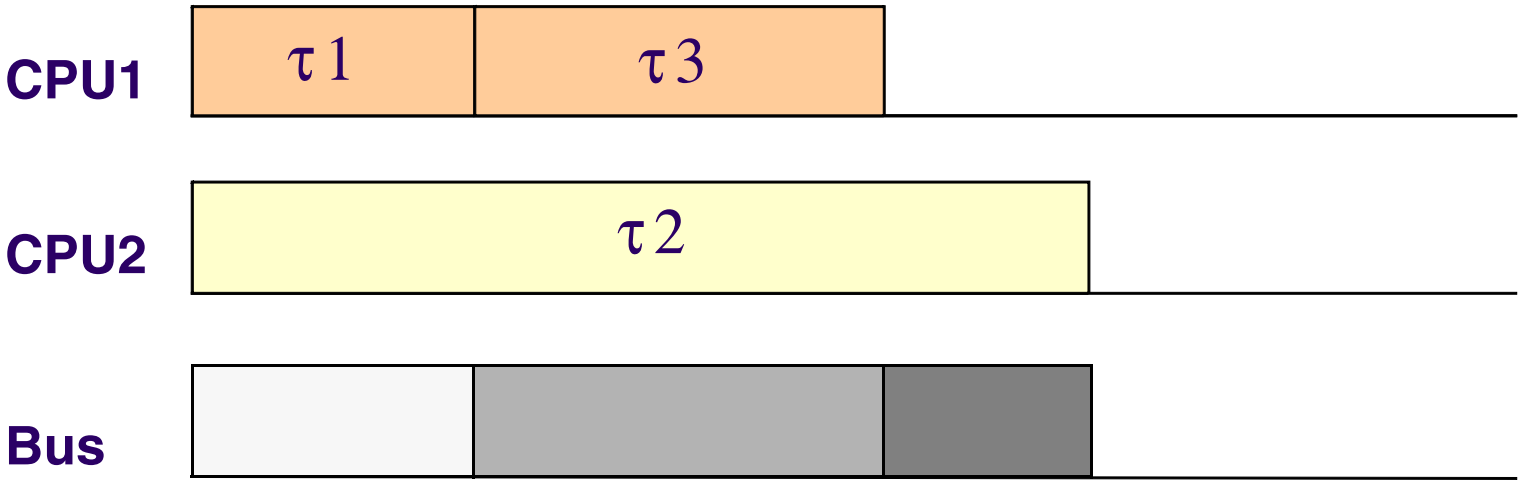


The Big Picture: Example

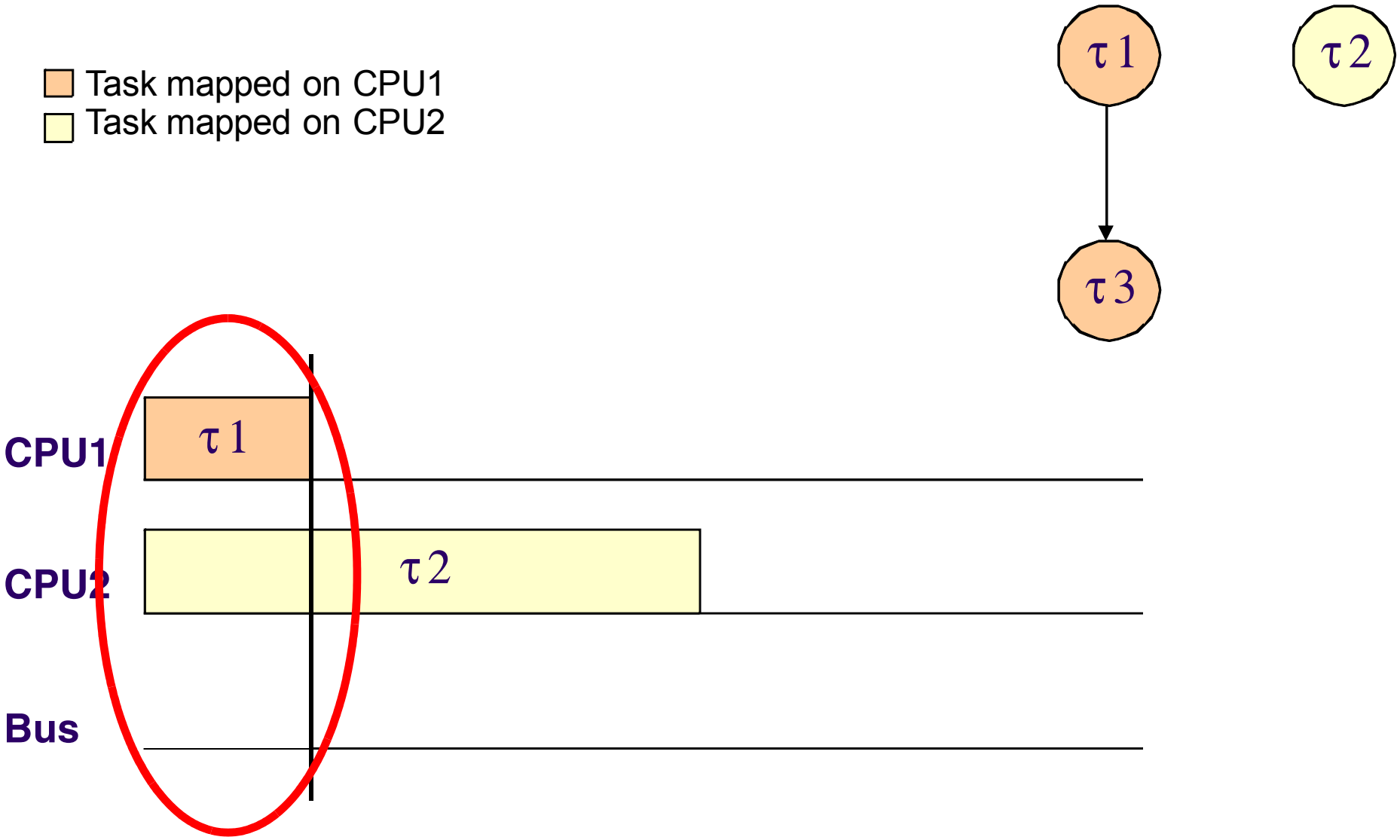
Task mapped on CPU1
Task mapped on CPU2



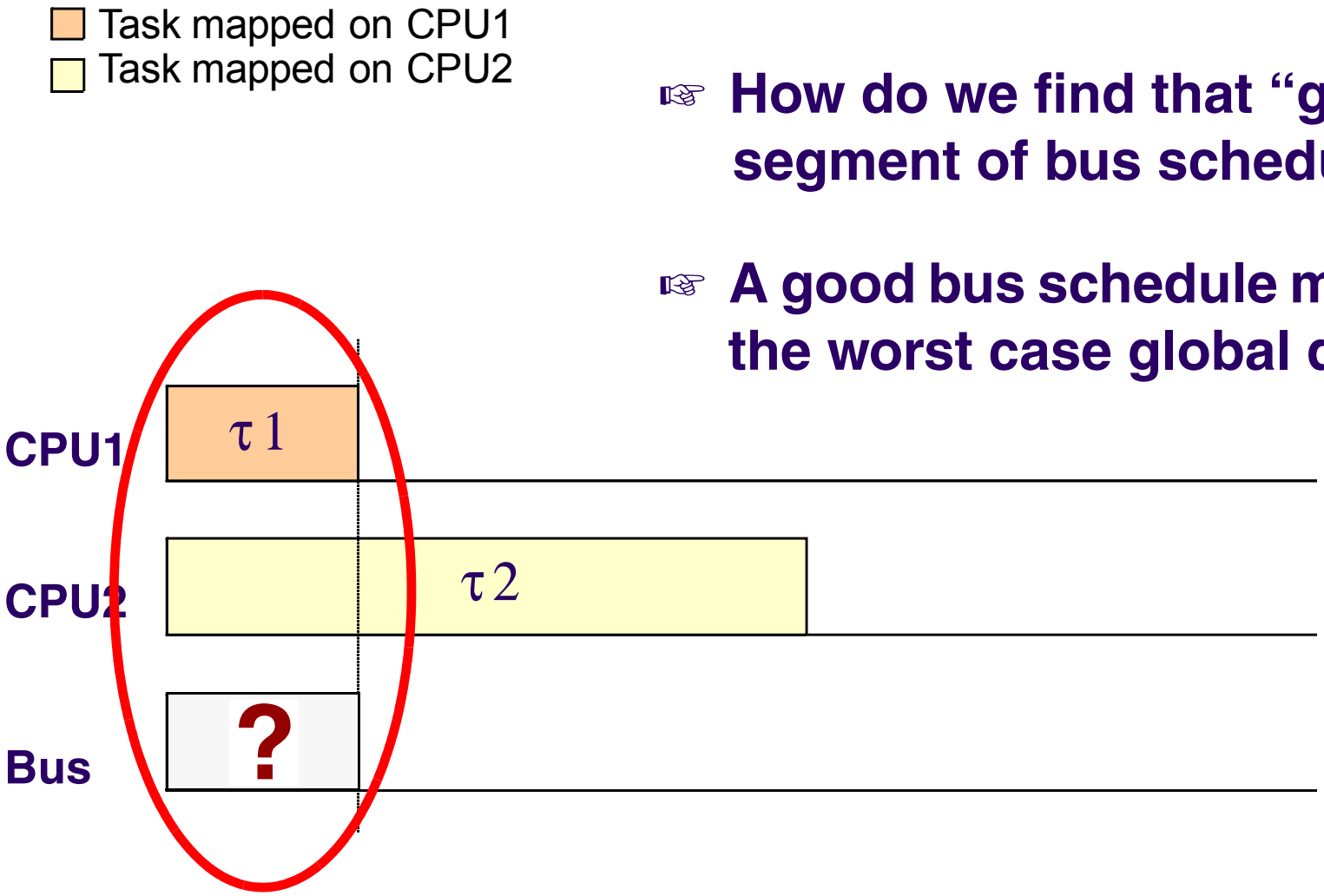
The final Bus and Task schedule



Bus Schedule Generation



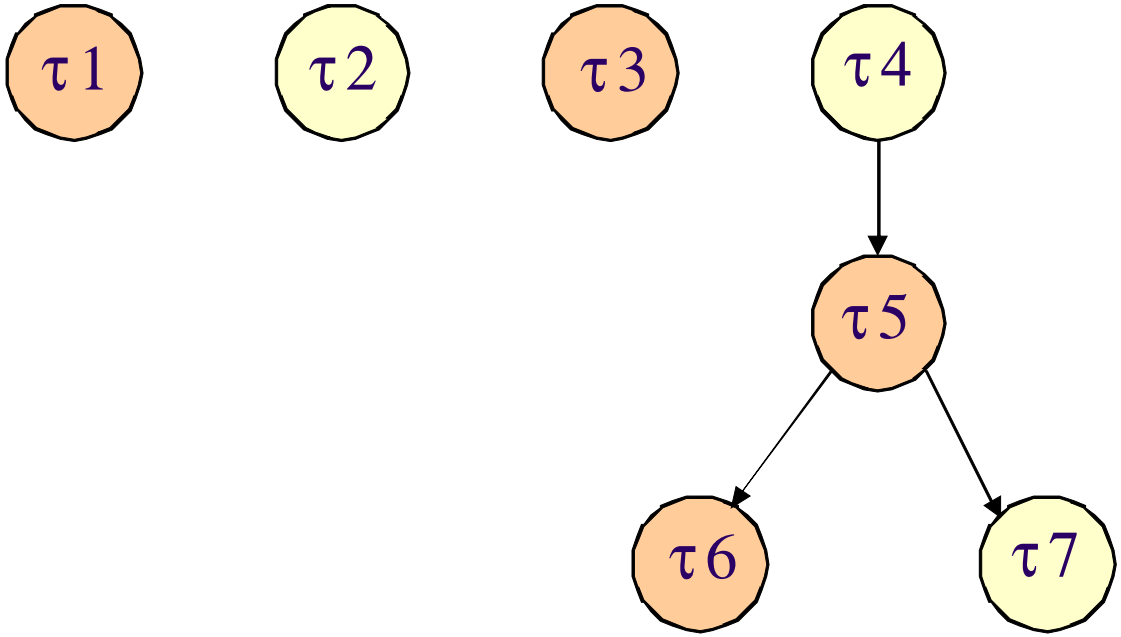
Bus Schedule Generation



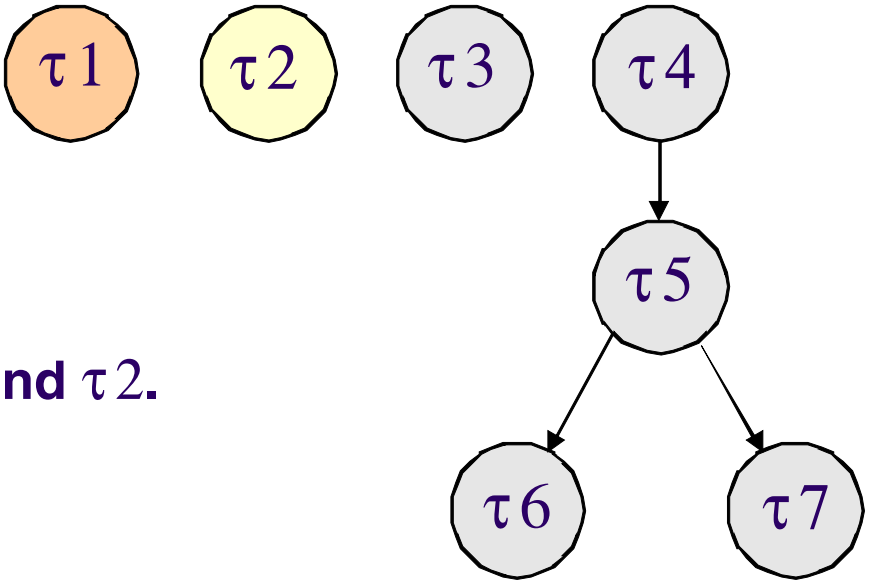
- How do we find that "good" segment of bus schedule?
- A good bus schedule minimizes the worst case global delay!

Bus Schedule Generation

Task mapped on CPU1
Task mapped on CPU2

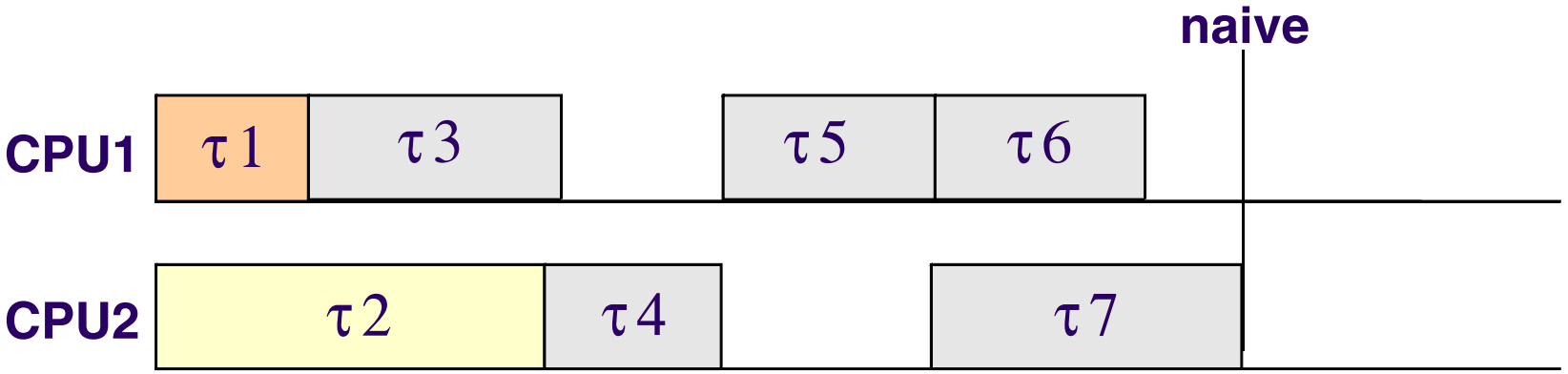


Bus Schedule Generation

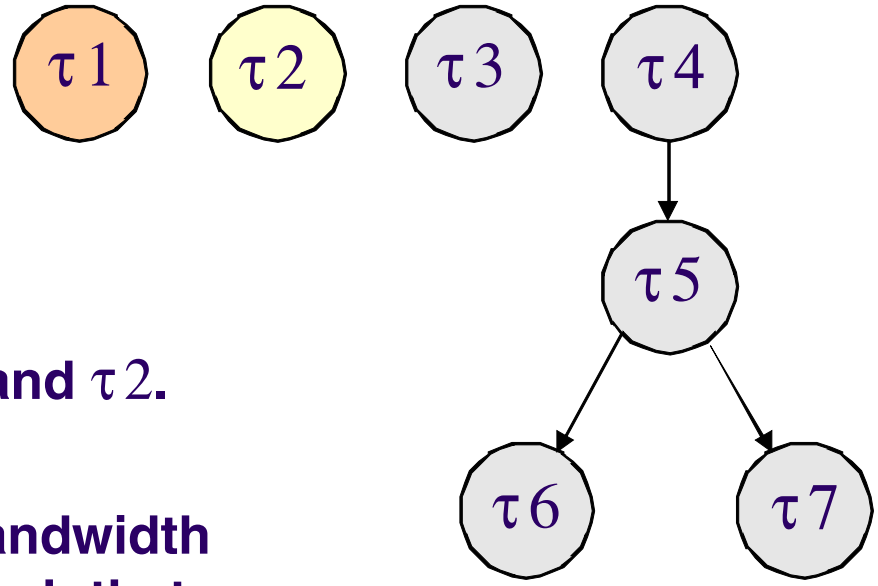


- Task being processed on CPU1
- Task being processed on CPU2
- Task not yet analyzed

☞ **We are looking at tasks τ1 and τ2.**

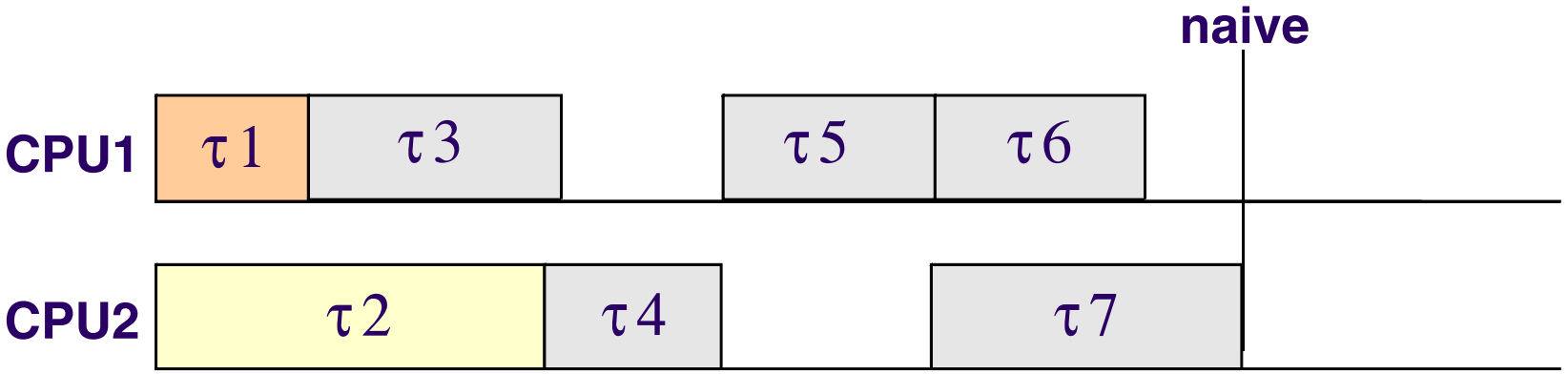


Bus Schedule Generation

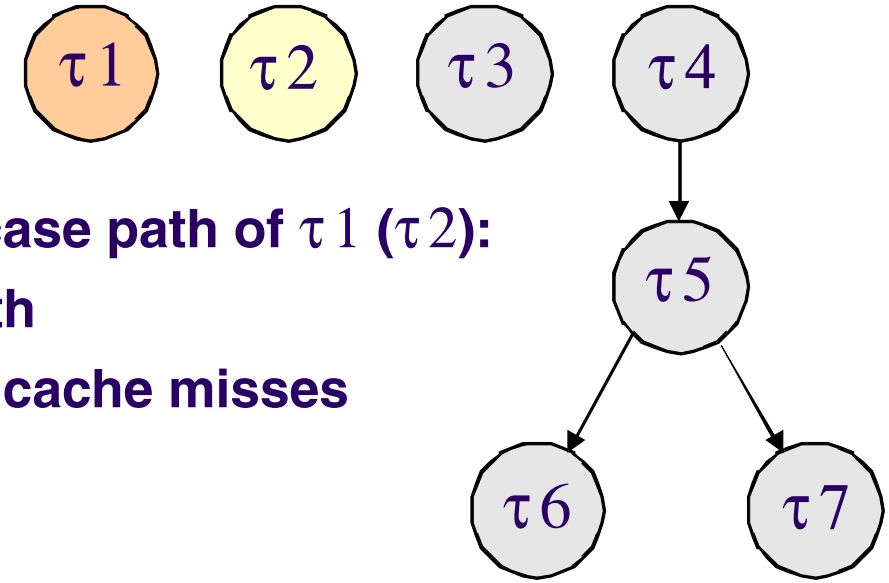


- Task being processed on CPU1
- Task being processed on CPU2
- Task not yet analyzed

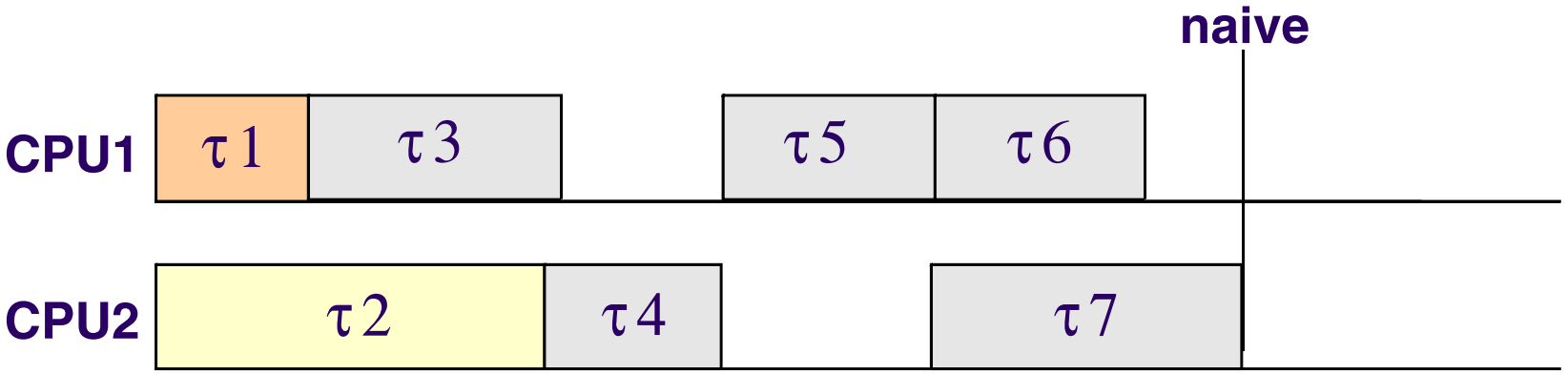
- **We are looking at tasks τ1 and τ2.**
- **Question:**
  How to distribute the bus bandwidth between CPU1 and CPU2 such that the global delay is reduced?



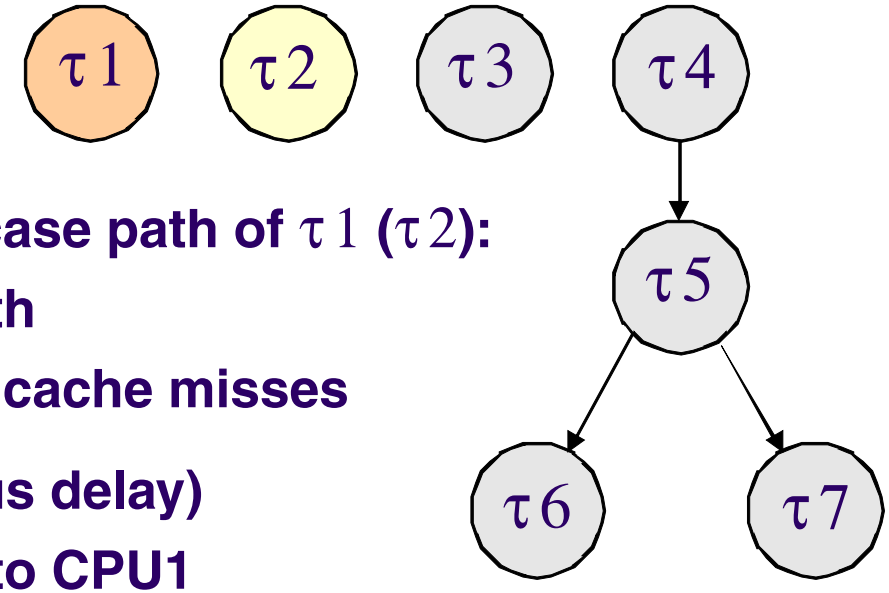
Bus Schedule Generation



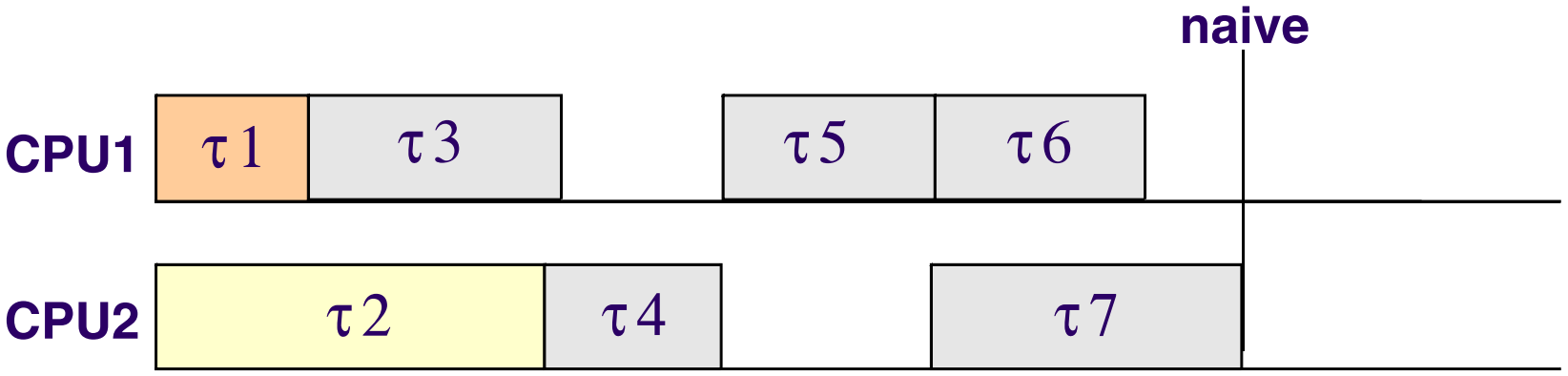
- *Traditional* WCET analysis detects a worst case path of τ1 (τ2):
  - *m1*: number of cash misses on this path
  - *l1*: total time spent on this path *except* cache misses



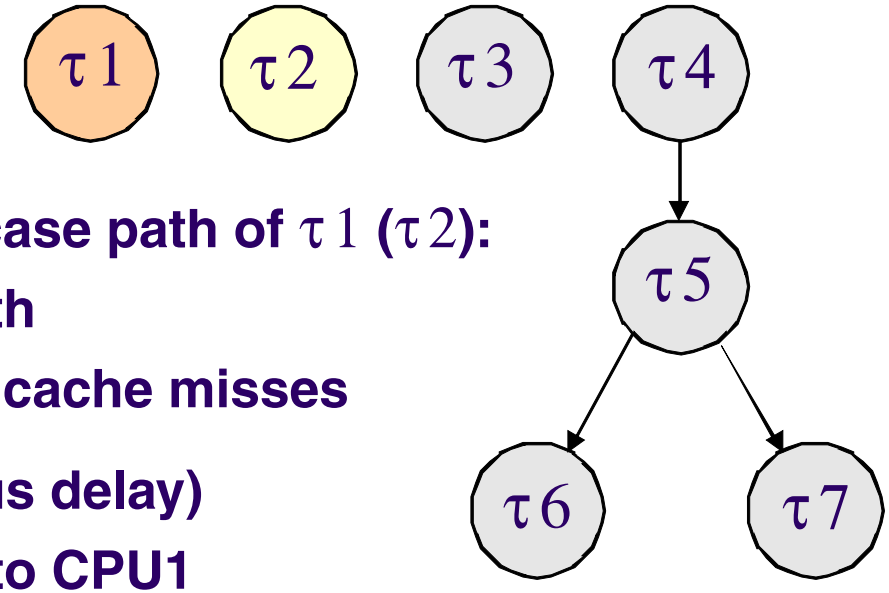
Bus Schedule Generation



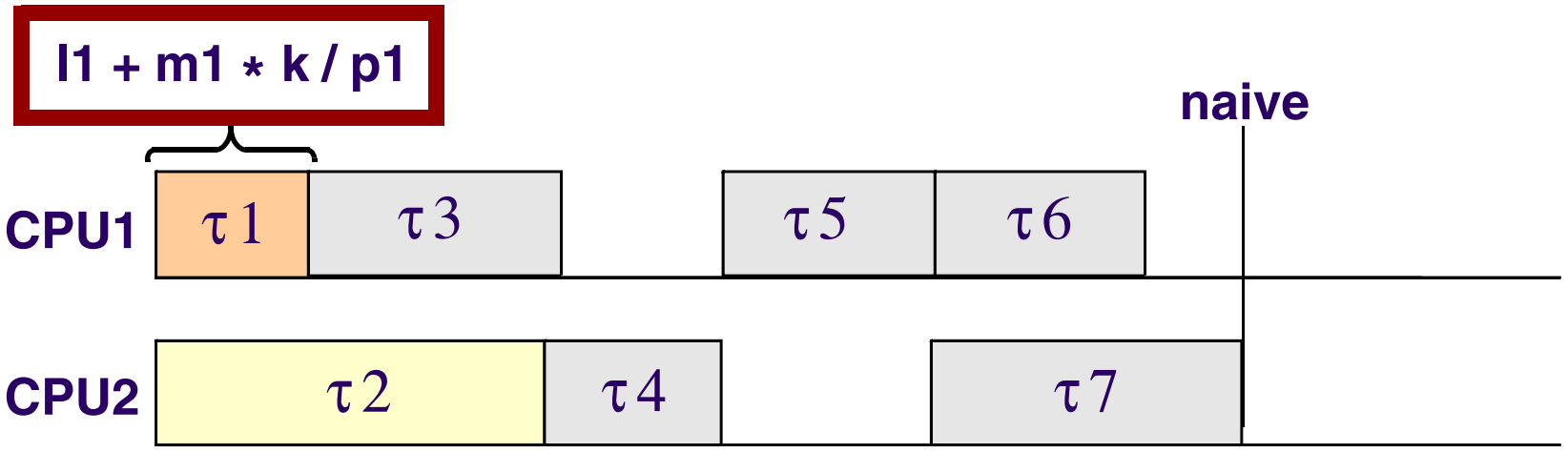
- *Traditional* WCET analysis detects a worst case path of τ1 (τ2):
  - *m1*: number of cash misses on this path
  - *l1*: total time spent on this path *except* cache misses
- *k*: time needed to solve a cache miss (no bus delay)
- p1: percentage of bus bandwidth allocated to CPU1



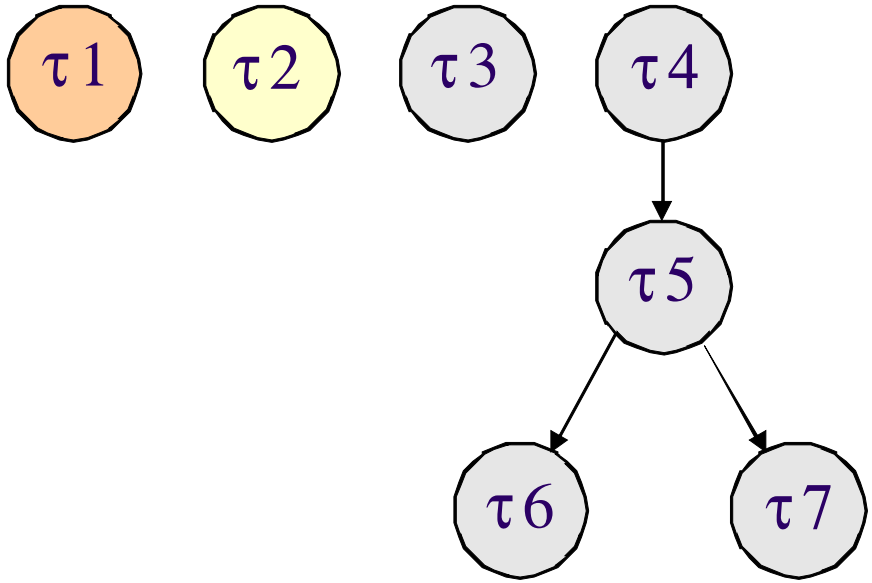
Bus Schedule Generation



- *Traditional* WCET analysis detects a worst case path of τ1 (τ2):
  - *m1*: number of cash misses on this path
  - *l1*: total time spent on this path *except* cache misses
- *k*: time needed to solve a cache miss (no bus delay)
- p1: percentage of bus bandwidth allocated to CPU1

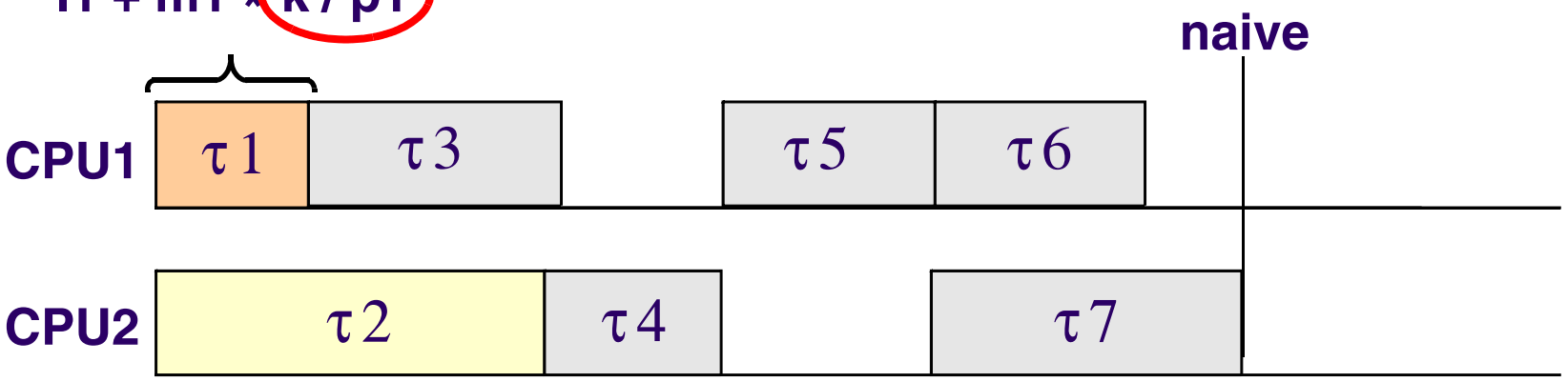


Bus Schedule Generation

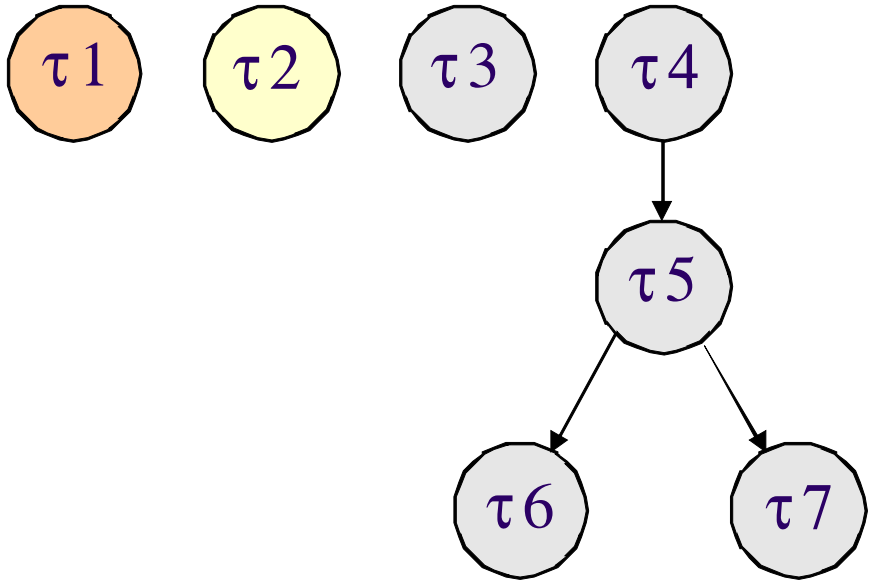


Average time to solve a cache miss

l1 + m1 * k / p1



Bus Schedule Generation



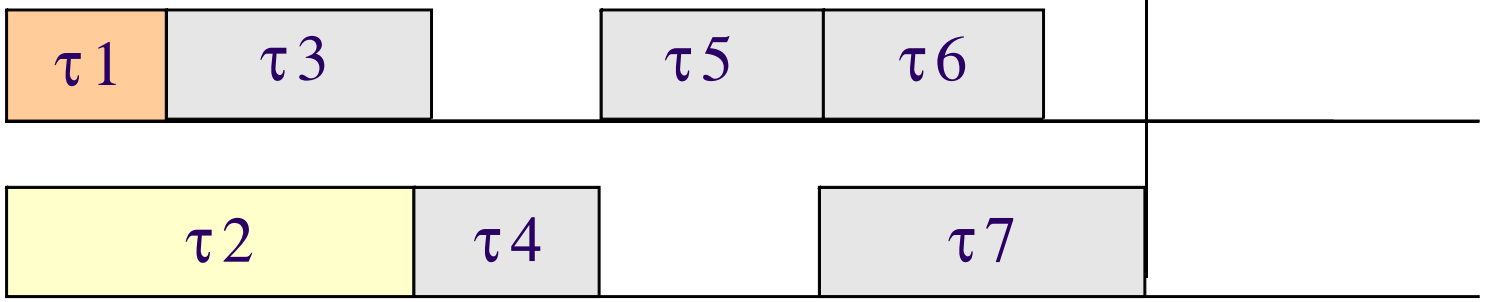
Total time to solve cache misses

l1 + m1 * k / p1

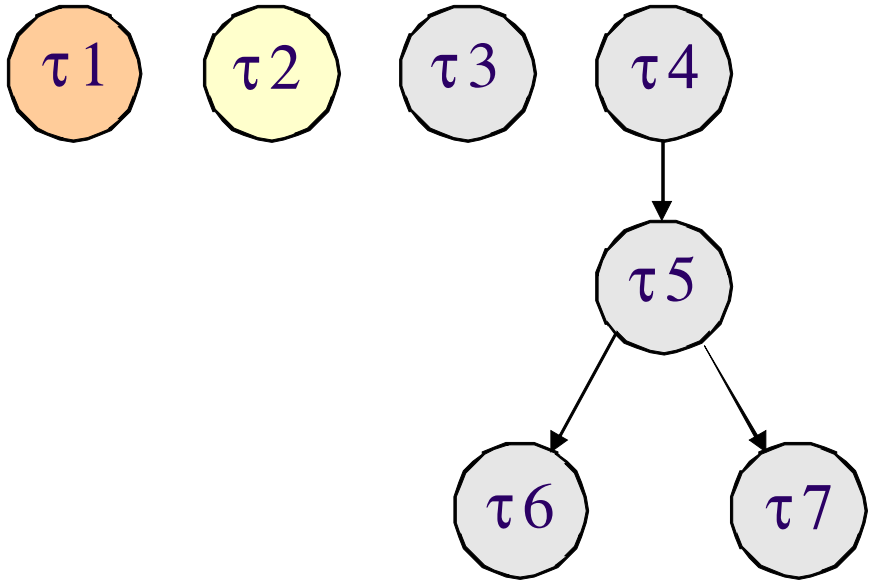
naive

CPU1

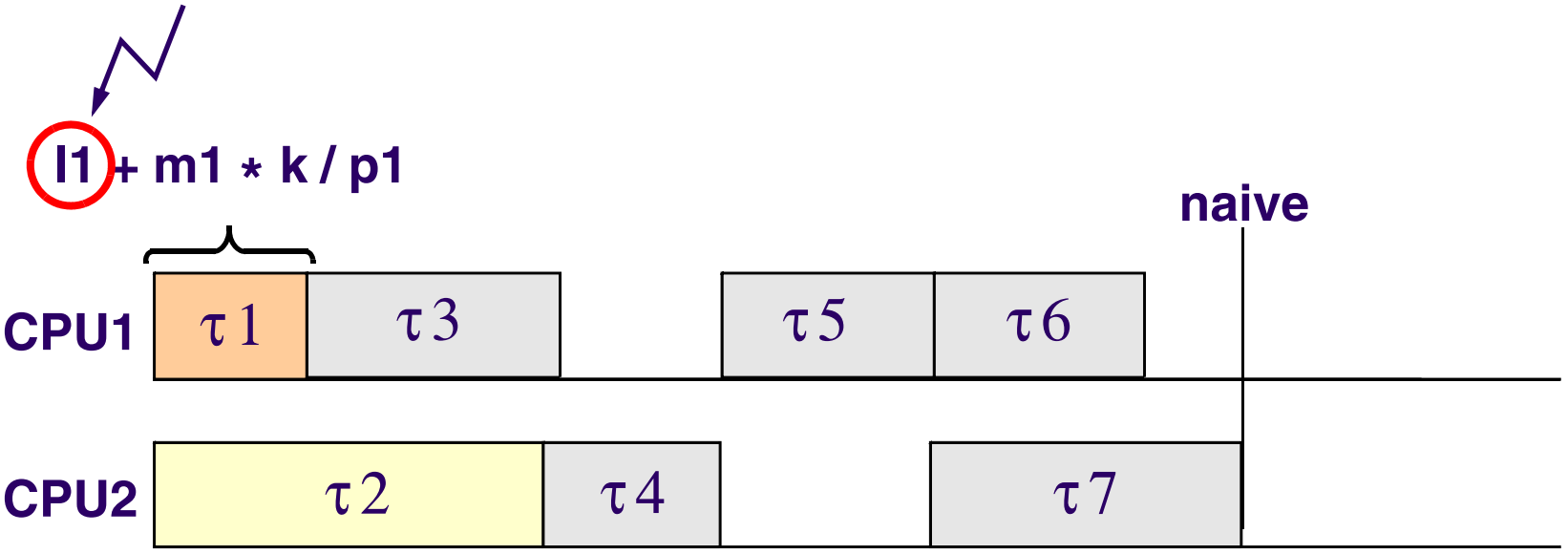
CPU2



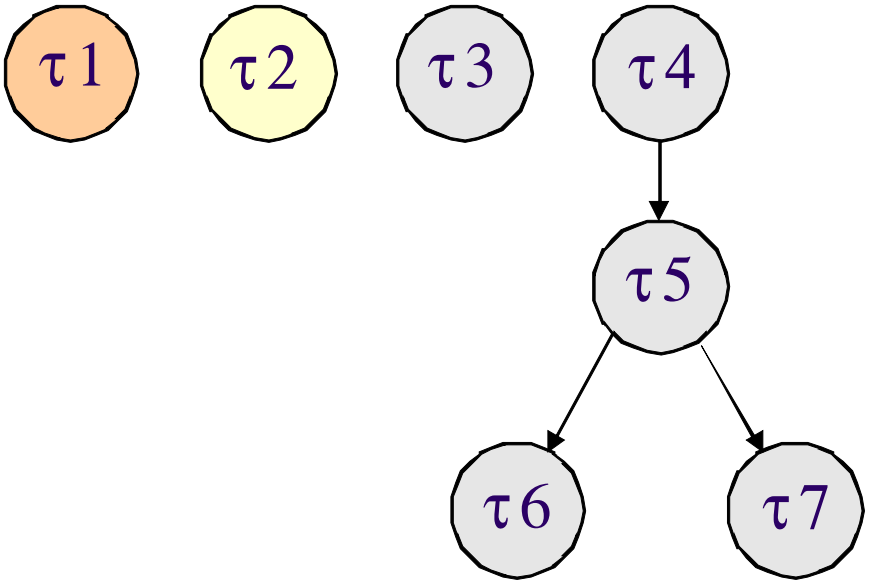
Bus Schedule Generation

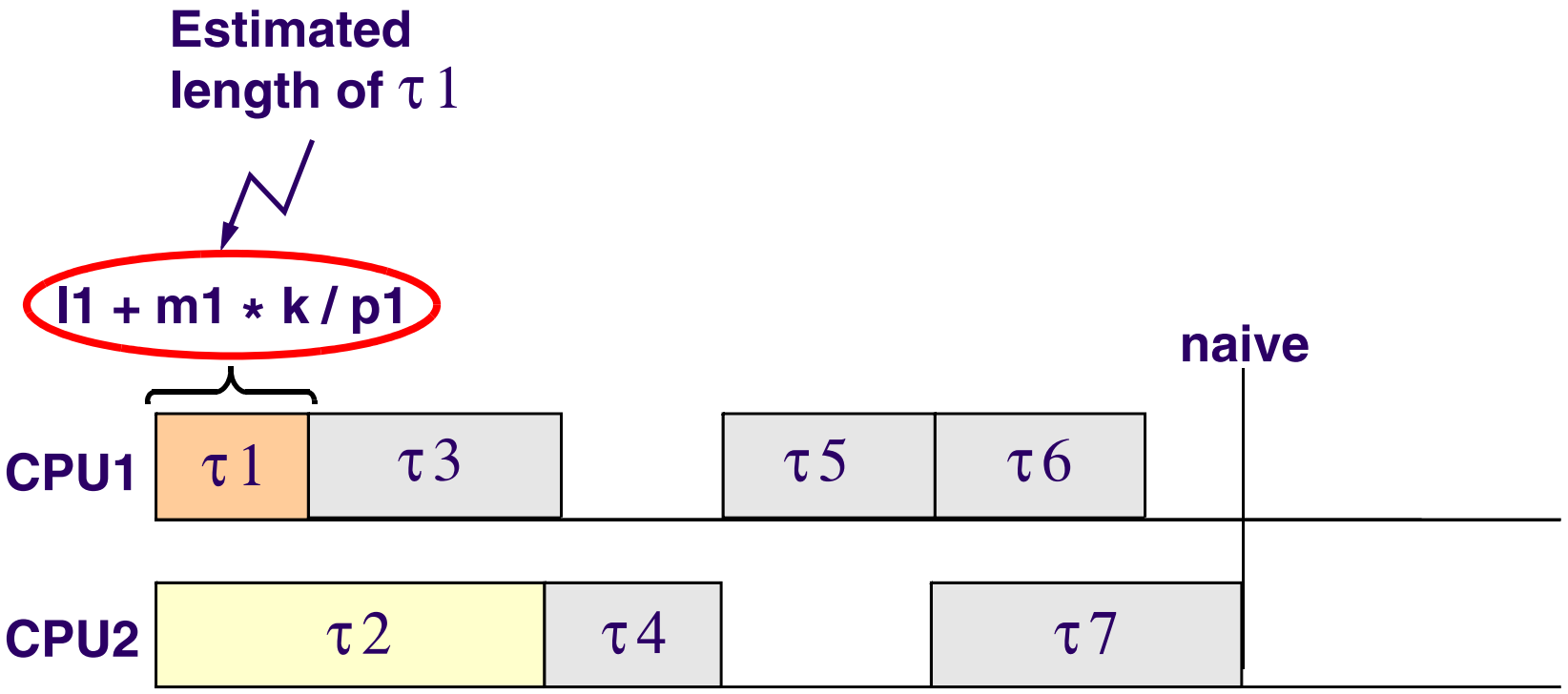


Ex. time, except cache misses

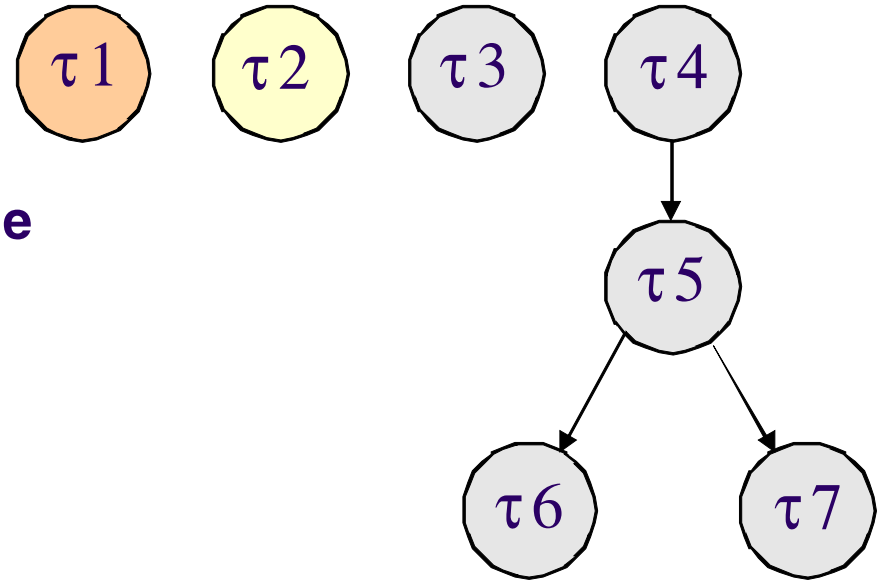


Bus Schedule Generation

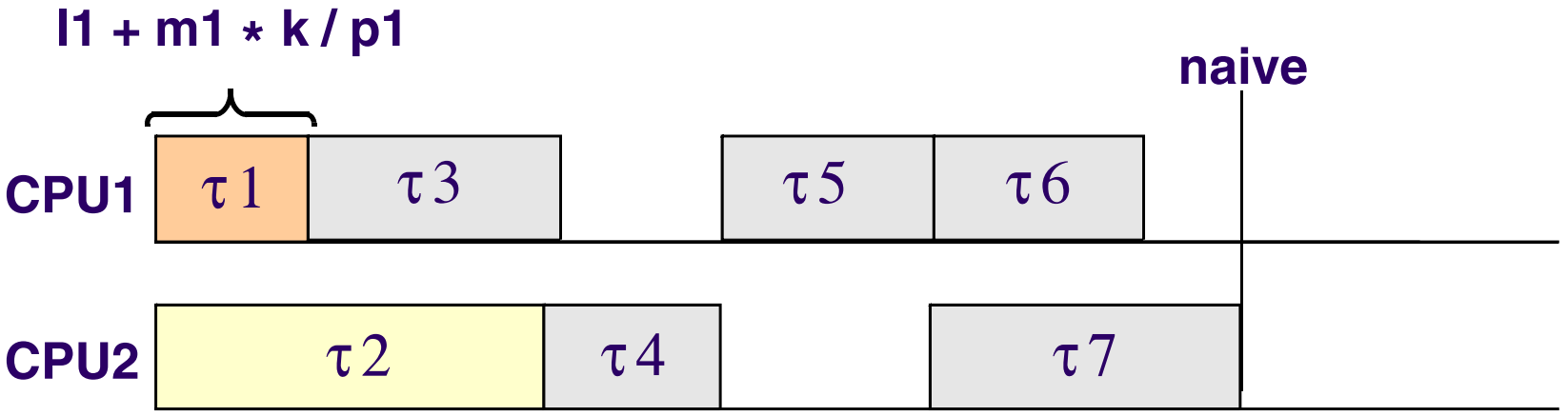




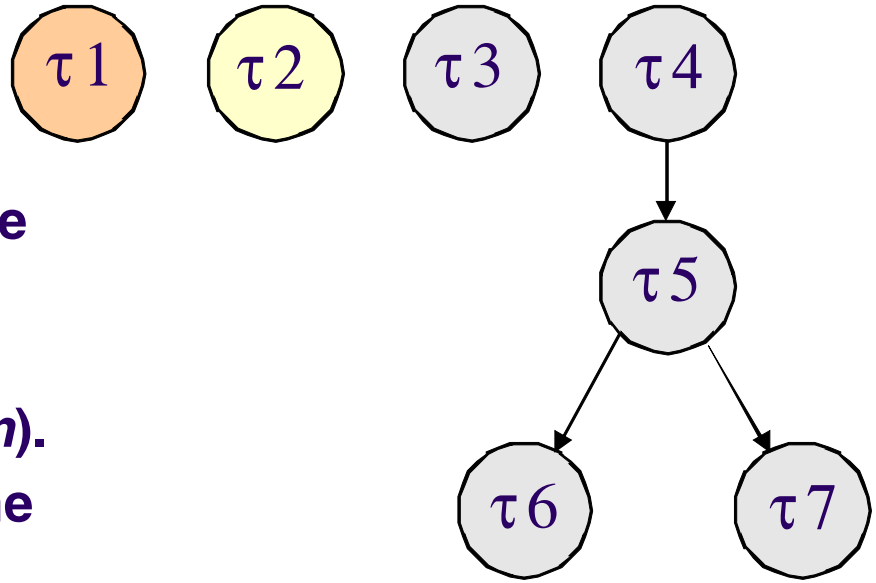
Bus Schedule Generation



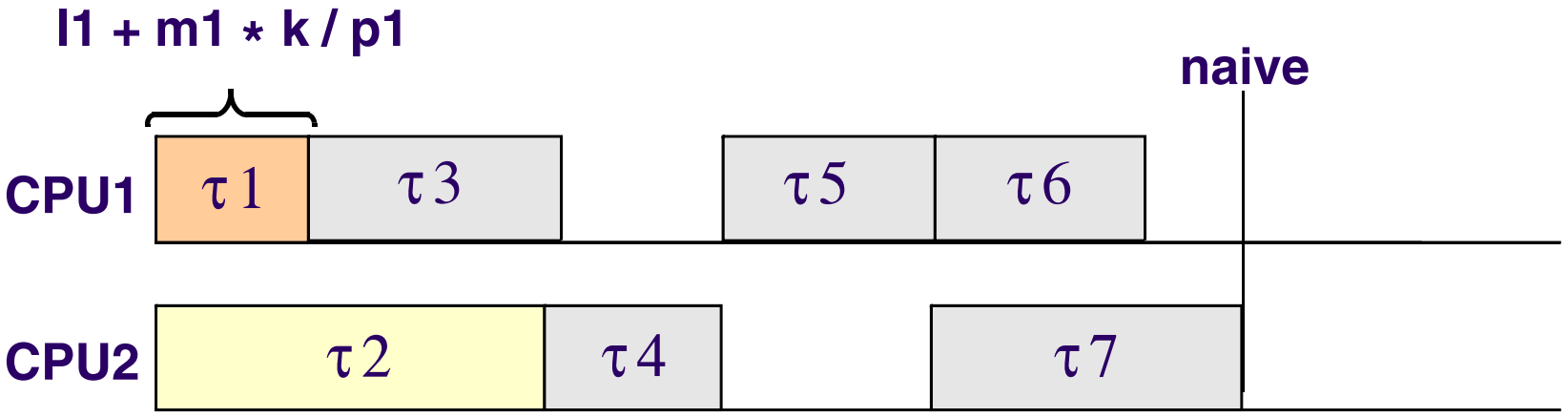
☞ **Question:**
How to fix *p1* and *p2* such that the global delay is reduced?



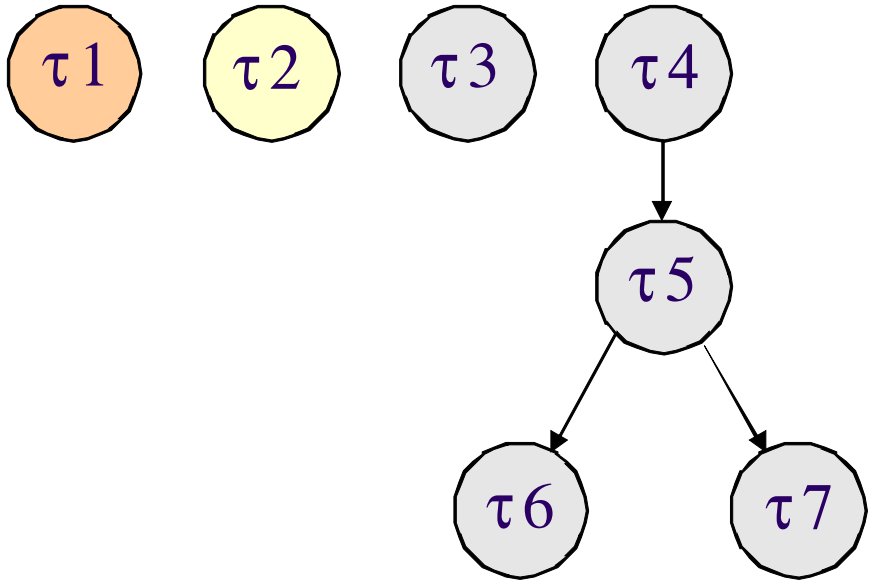
Bus Schedule Generation



- ☞ **Question:**
  How to fix *p1* and *p2* such that the global delay is reduced?
- **Give more bandwidth to the task with more cache misses (larger *m*).**
- **Tasks have different impact on the global delay!**

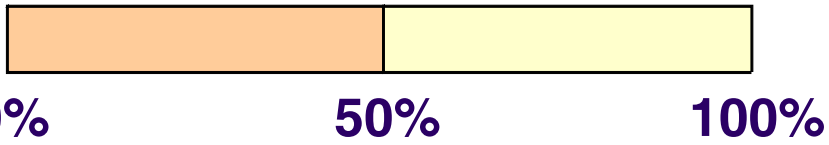


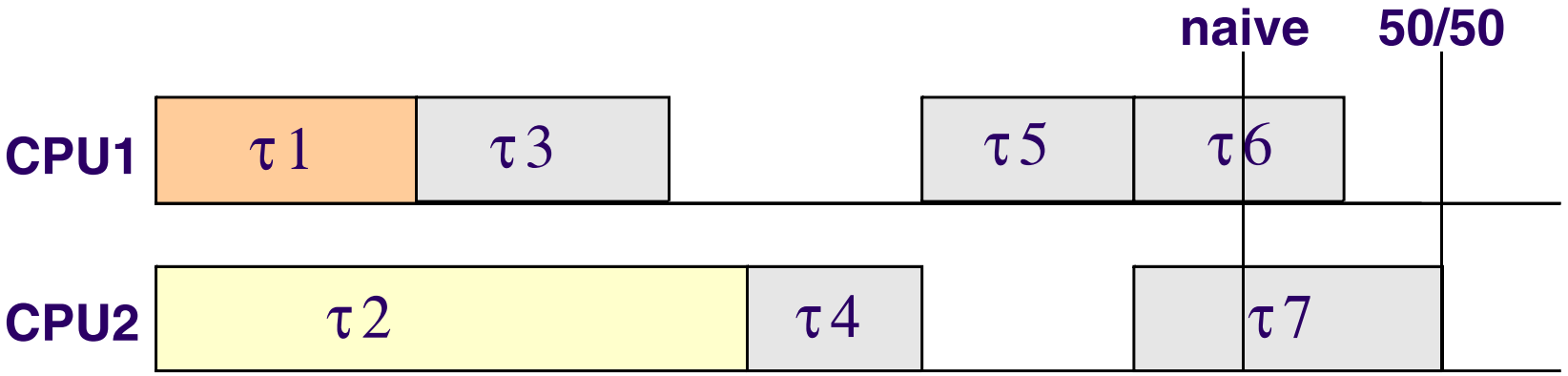
Bus Schedule Generation



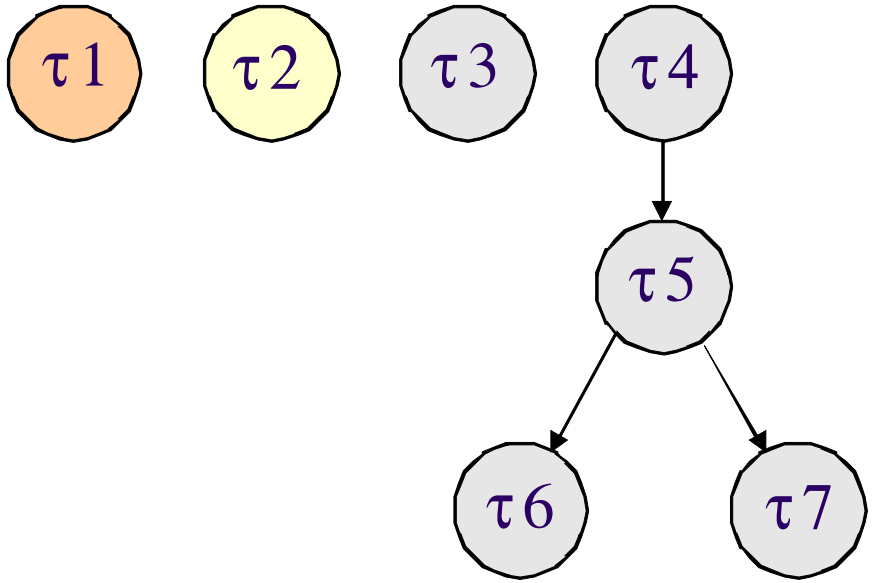
Task being processed on CPU1
Task being processed on CPU2
Task not yet analyzed

Bandwidth Distribution



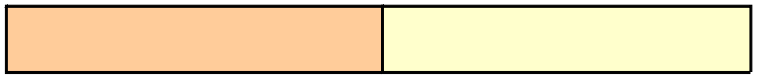


Bus Schedule Generation



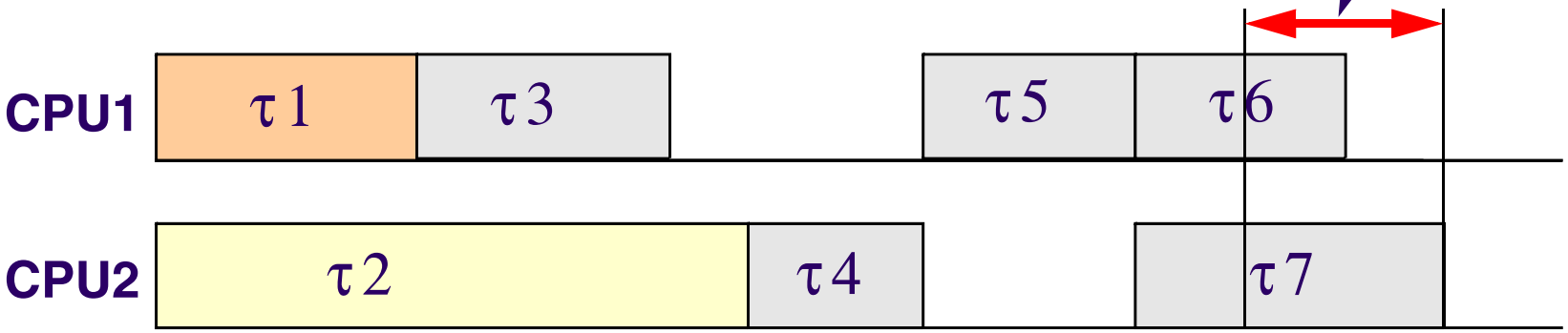
Task being processed on CPU1
Task being processed on CPU2
Task not yet analyzed

Bandwidth Distribution

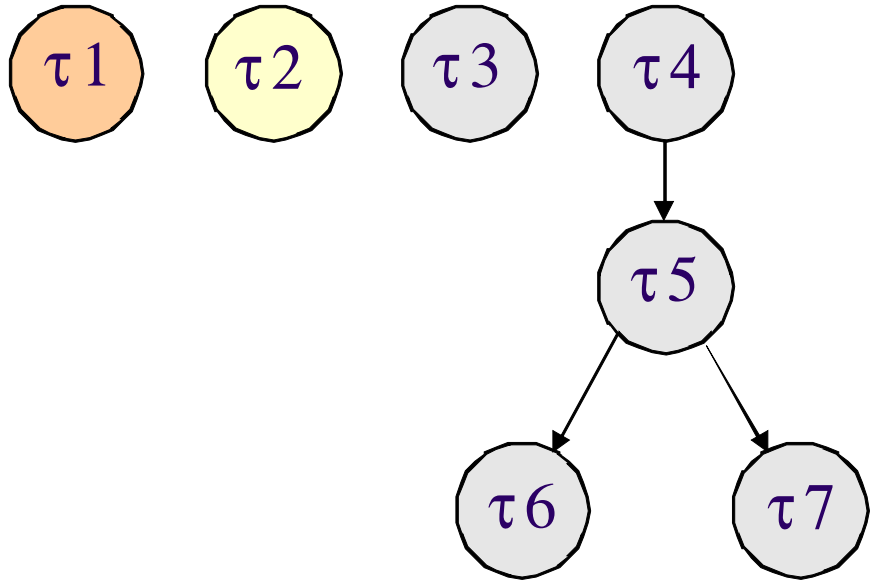


0% **50%** **100%**

This has to be minimized

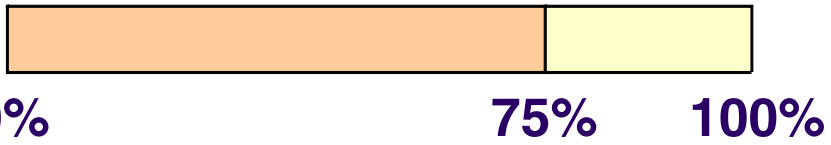


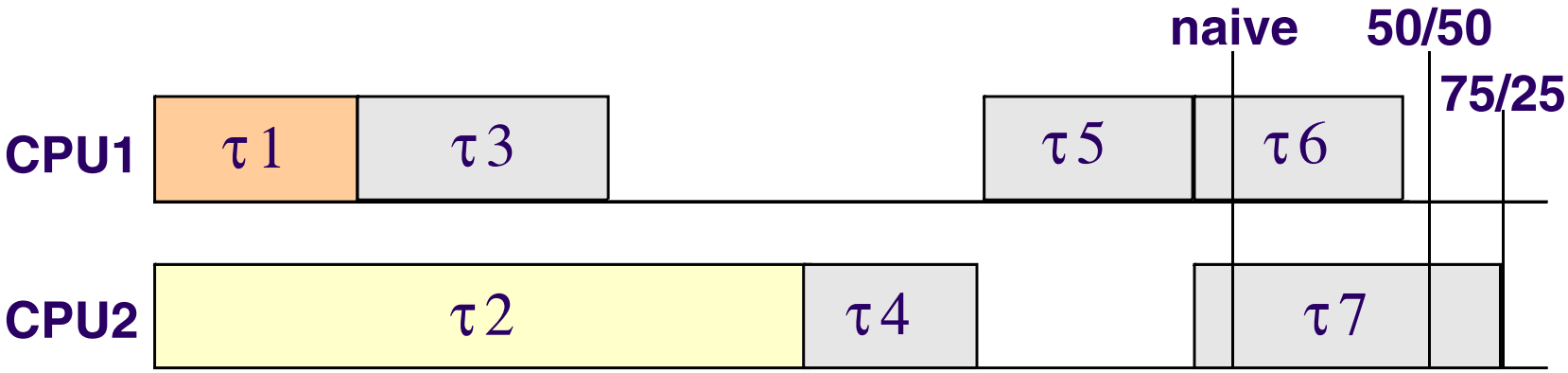
Bus Schedule Generation



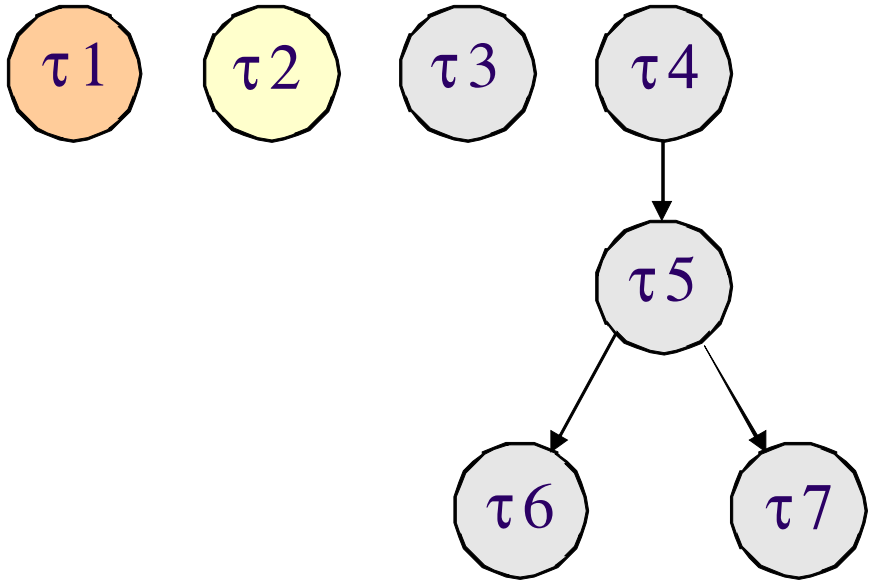
Task being processed on CPU1
Task being processed on CPU2
Task not yet analyzed

Bandwidth Distribution



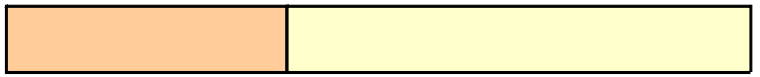


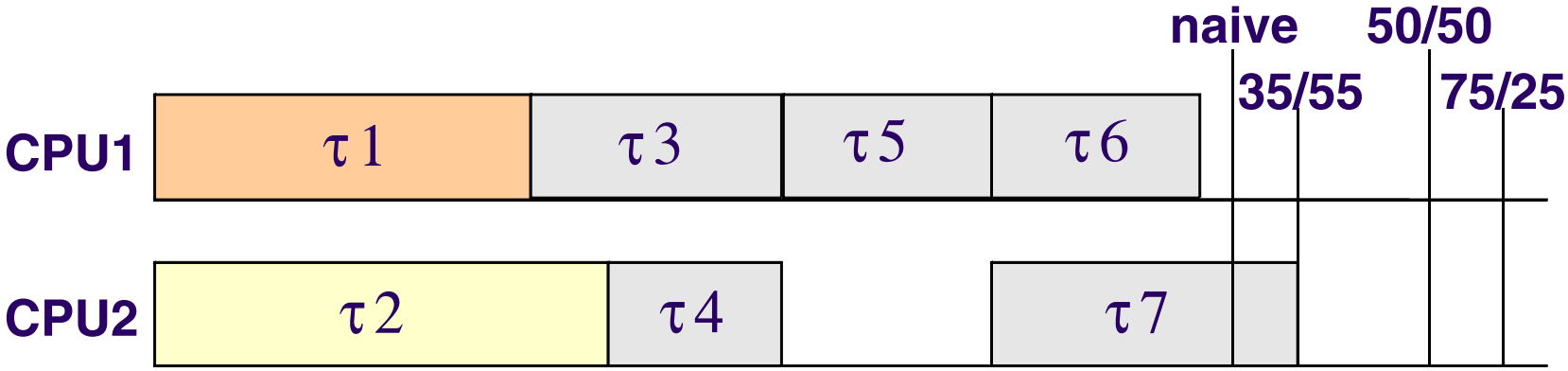
Bus Schedule Generation



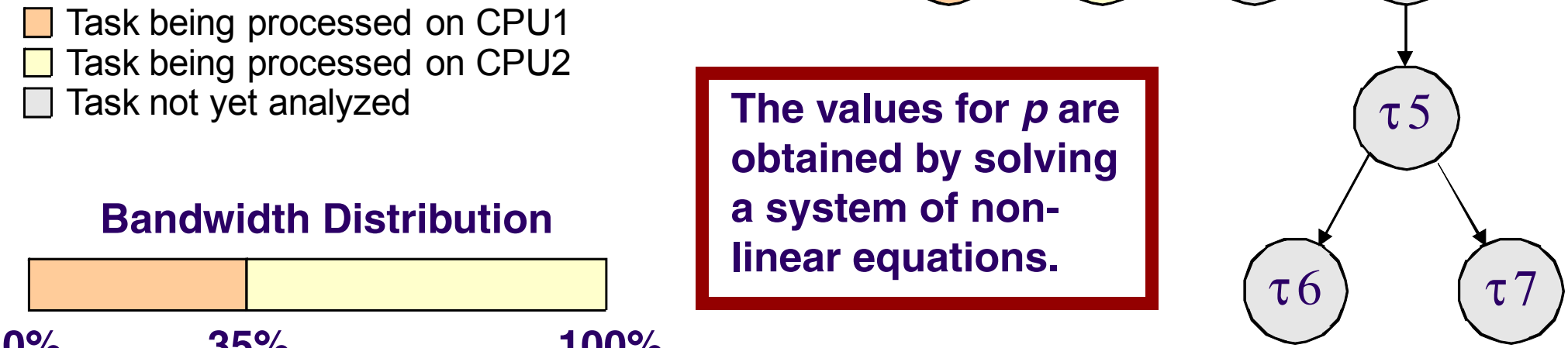
Task being processed on CPU1
Task being processed on CPU2
Task not yet analyzed

Bandwidth Distribution

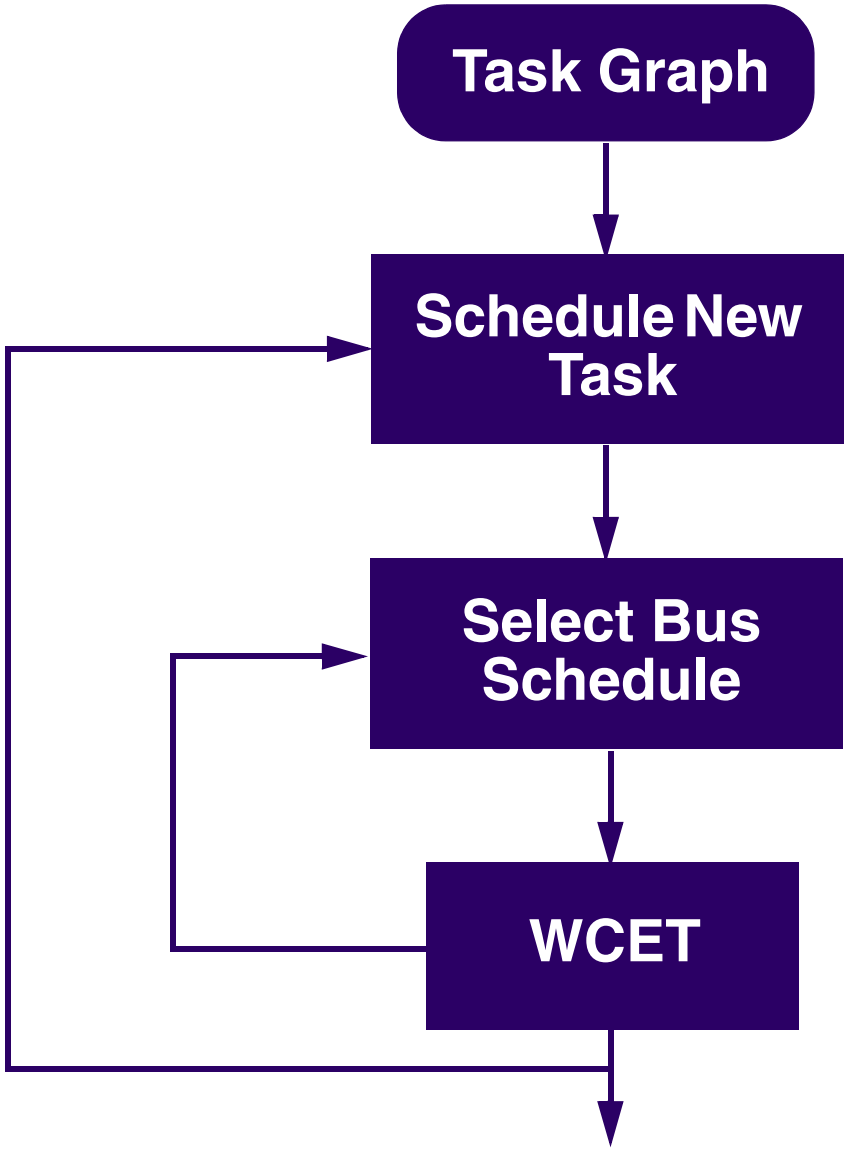




Bus Schedule Generation

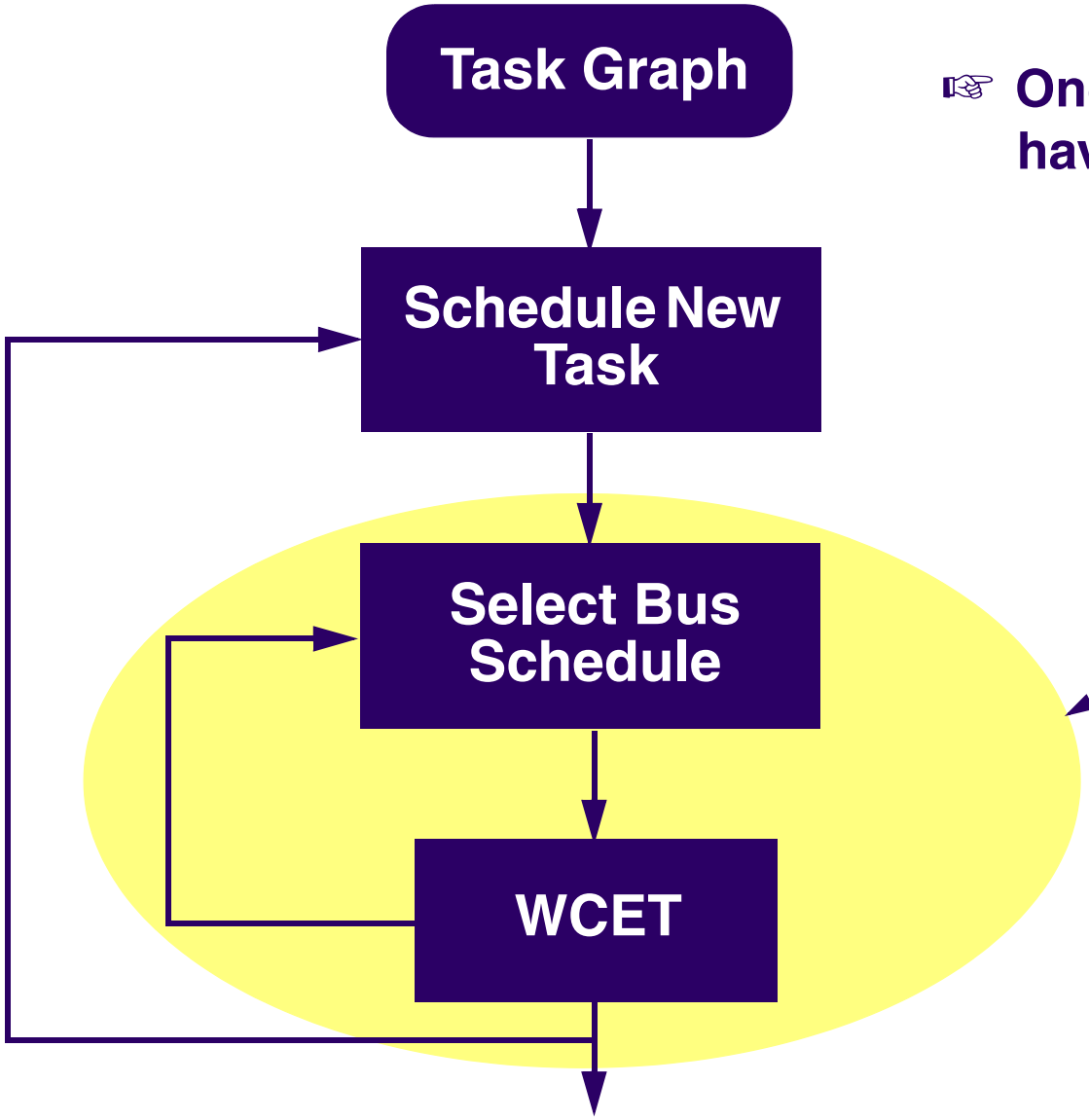


Bus Schedule Generation



- Once the values for *p* are obtained we have to fix:
  - Size of slots
  - Order of the slots

Bus Schedule Generation



- Once the values for *p* are obtained we have to fix:
  - Size of slots in the segment
  - Order of the slots in the segment

Density Regions

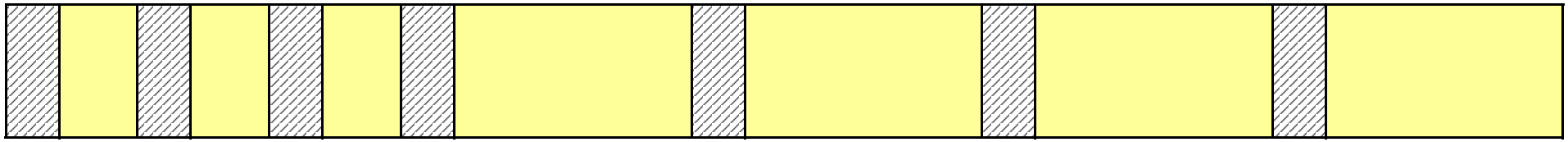
- When calculating the bandwidth distributions *p* we have assumed that misses are uniformly distributed throughout the task's critical path.

Density Regions

- When calculating the bandwidth distributions *p* we have assumed that misses are uniformly distributed throughout the task's critical path.

☞ In reality this is NOT the case.

Task τ_j

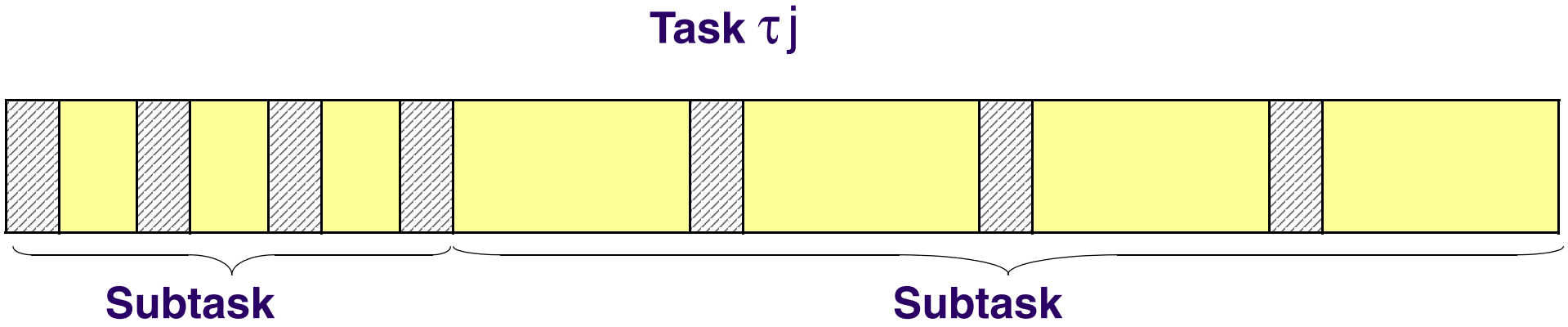


Density Regions

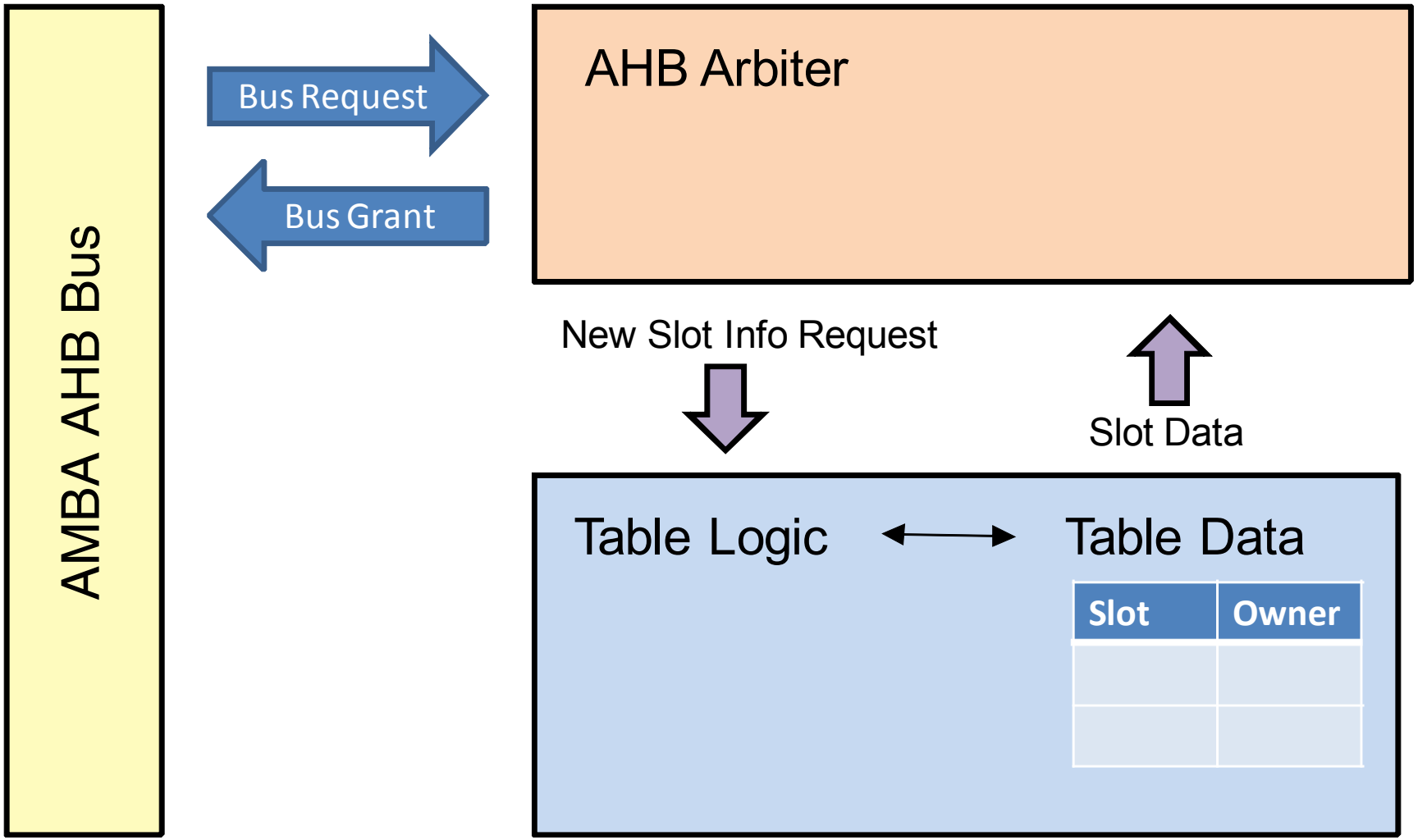
- When calculating the bandwidth distributions *p* we have assumed that misses are uniformly distributed throughout the task's critical path.

☞ In reality this is NOT the case.

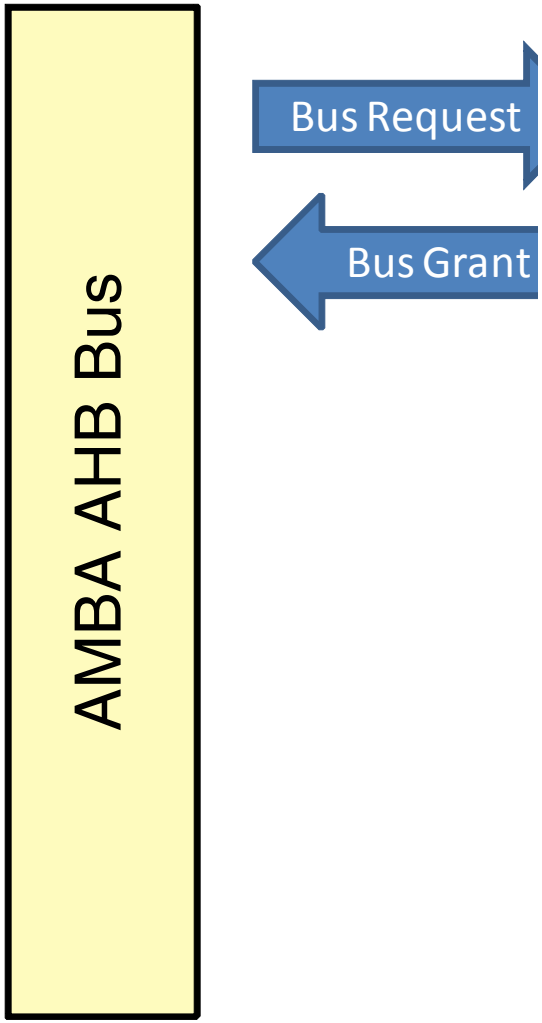
- We divide a task into subtasks according to *density regions*.



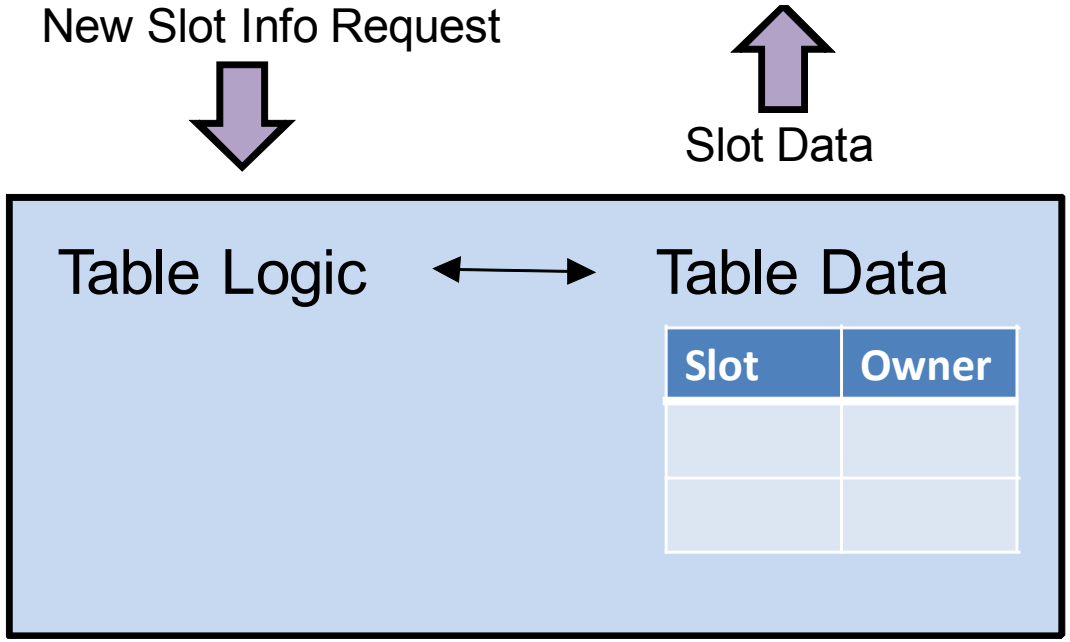
The Bus Arbiter



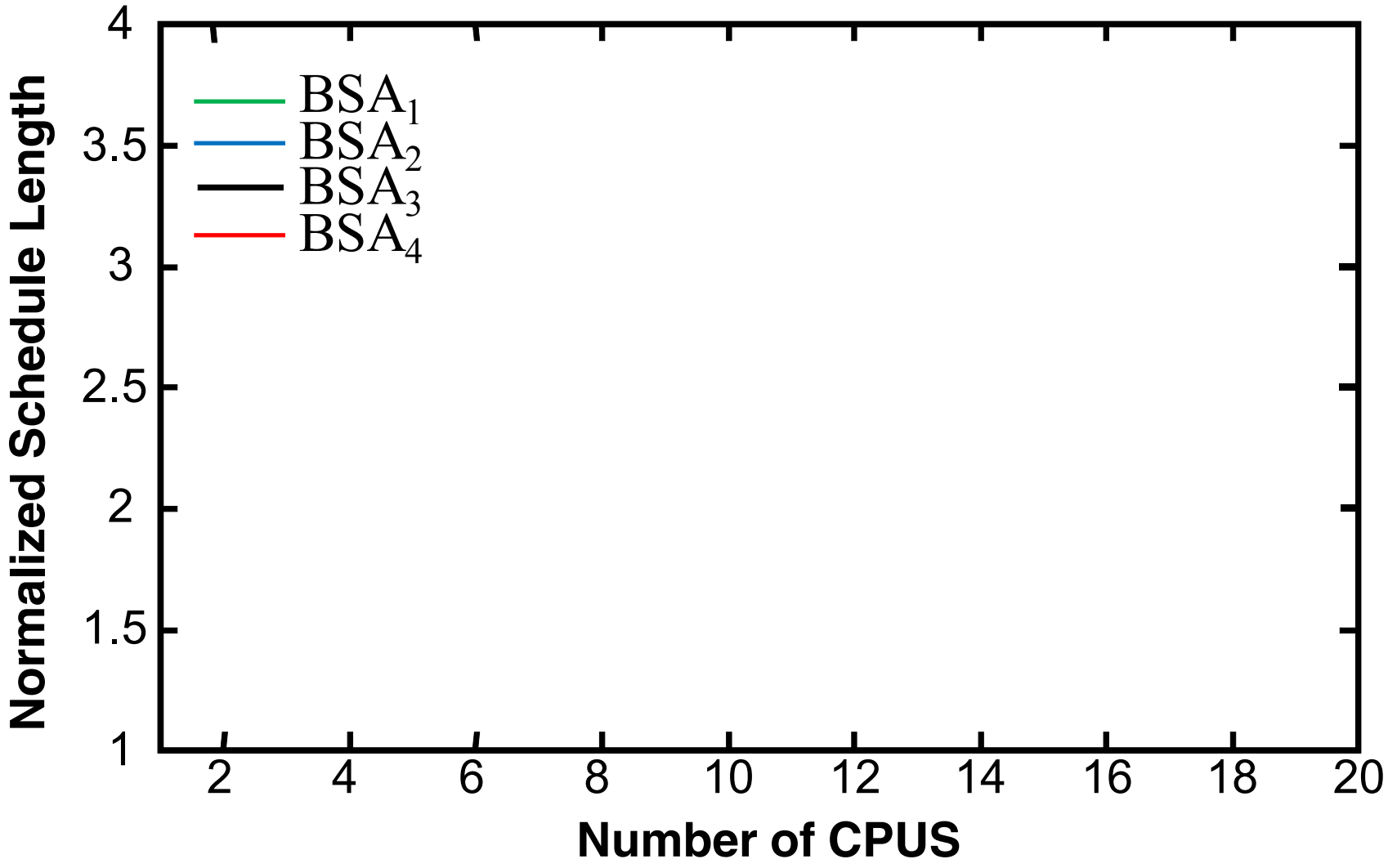
The Bus Arbiter



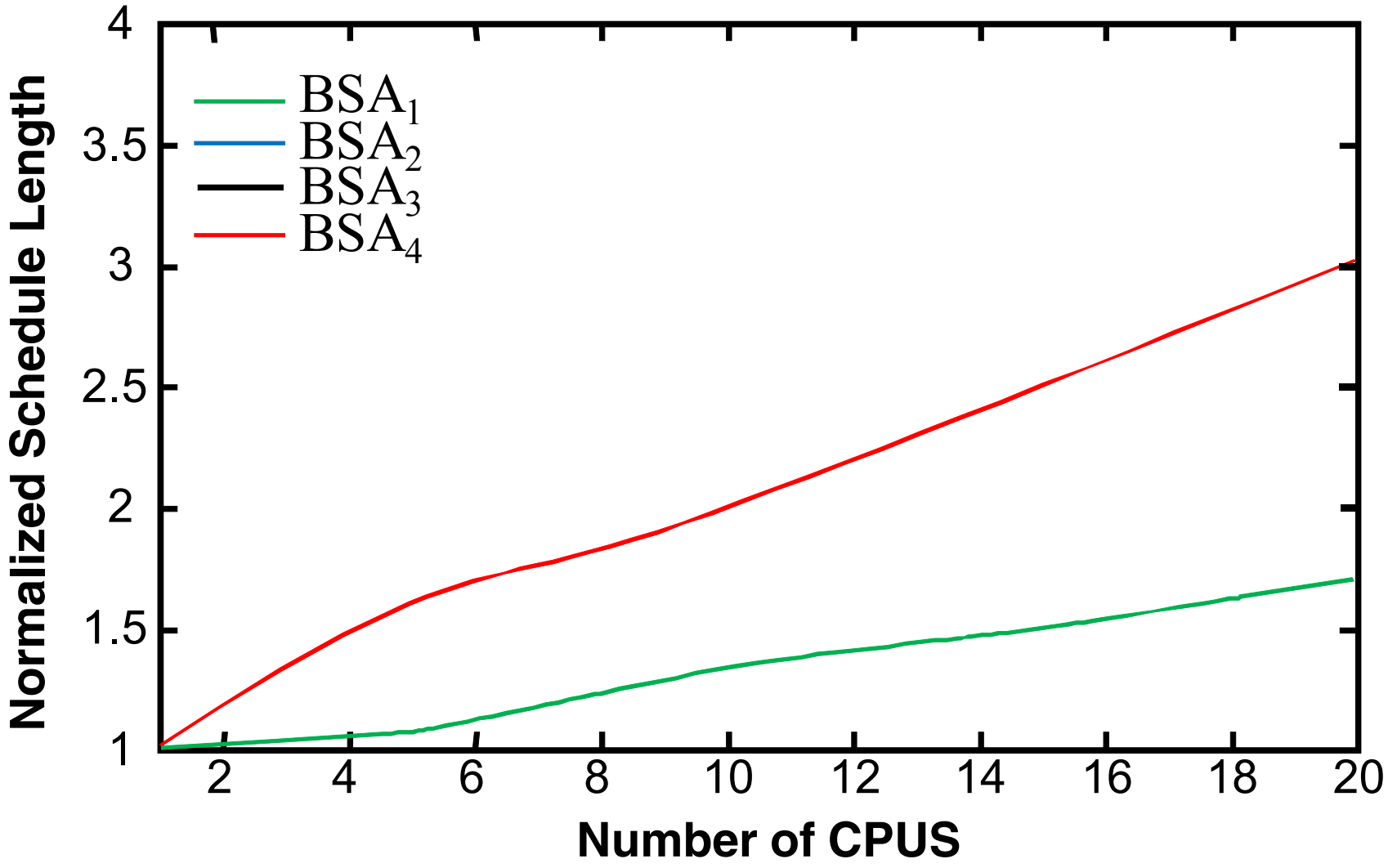
- Designed, in cooperation with University of Bologna (Luca Benini, Paolo Burgio)
- Synthesised in 90 nm technology (very low overhead!)



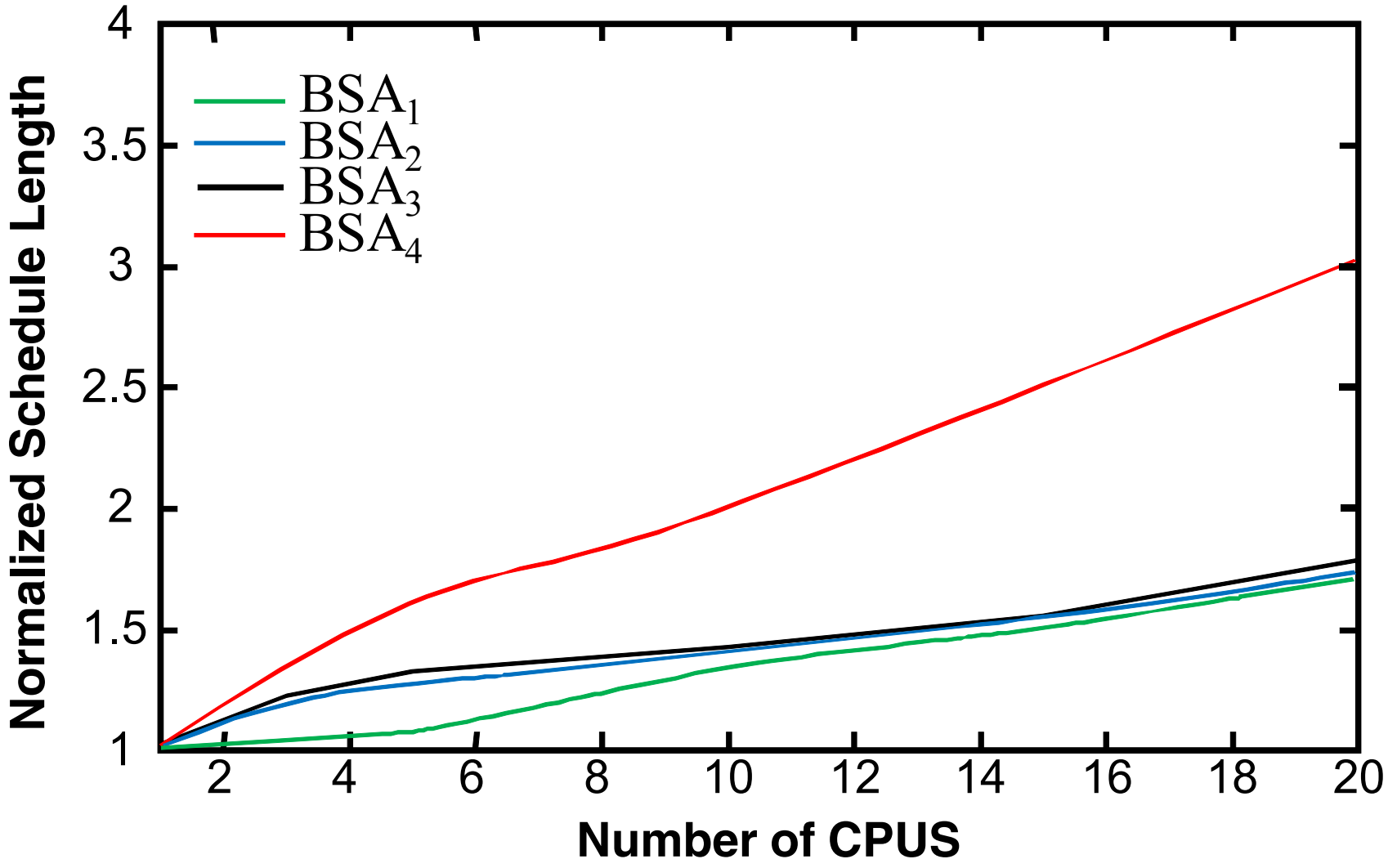
Experiments



Experiments



Experiments



Conclusions

- An Approach to Predictable Implementation on Multiprocessors
  - TDMA Bus Schedule
  - WCET Analysis with TDMA
  - Bus schedule generation

Conclusions

- An Approach to Predictable Implementation on Multiprocessors
  - TDMA Bus Schedule
  - WCET Analysis with TDMA
  - Bus schedule generation

Predictable implementation on MPSoC is possible without excessive overhead.