Towards Unit Testing Real-Time Schedulers in LITMUS^{RT}

Mac Mollison
Björn Brandenburg
James H. Anderson

Towards Unit Testing Real-Time Schedulers in LITMUS^{RT}

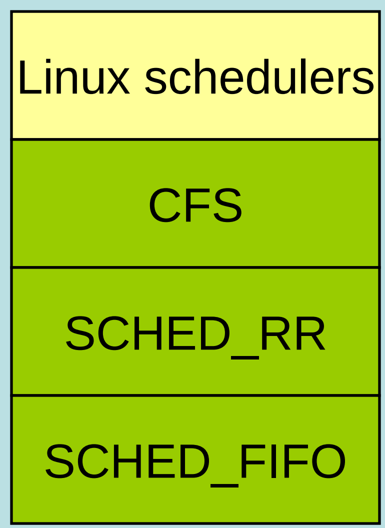
- What is LITMUS^{RT}?
- What is this talk about?
- What is a typical scheduling policy?
- Why do we need a test tool?
- How do we test? (Answer: Unit Testing)
- What are the specific tests?

LITMUS^{RT}

LInux **T**estbed for **MU**ltiprocesor **S**cheduling in **R**eal-**T**ime Systems

Patch to Linux 2.6.24 Kernel

Before the patch:



LITMUS^{RT}

LInux **T**estbed for **MU**ltiprocesor **S**cheduling in **R**eal-**T**ime **S**ystems

Patch to Linux 2.6.24 Kernel

After the patch:

Linux schedulers
CFS
SCHED_RR
SCHED_FIFO

+

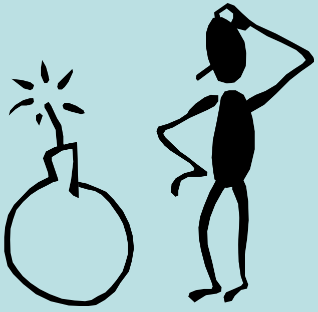
LITMUS^{RT} schedulers
G-EDF
C-EDF
P-EDF
PD^2

Towards Unit Testing Real-Time Schedulers in LITMUS^{RT}

- What is LITMUS^{RT}?
- What is this talk about?
- What is a typical scheduling policy?
- Why do we need a test tool?
- How do we test? (Answer: Unit Testing)
- What are the specific tests?

Overview

- What LITMUS^{RT} is
- Why we want to test LITMUS^{RT} schedulers
  - *Implementing real-time schedulers is nontrivial – bugs can be subtle*
- How to test LITMUS^{RT} schedulers
  - *Unit Testing - testing small pieces of code programmatically – with a twist*



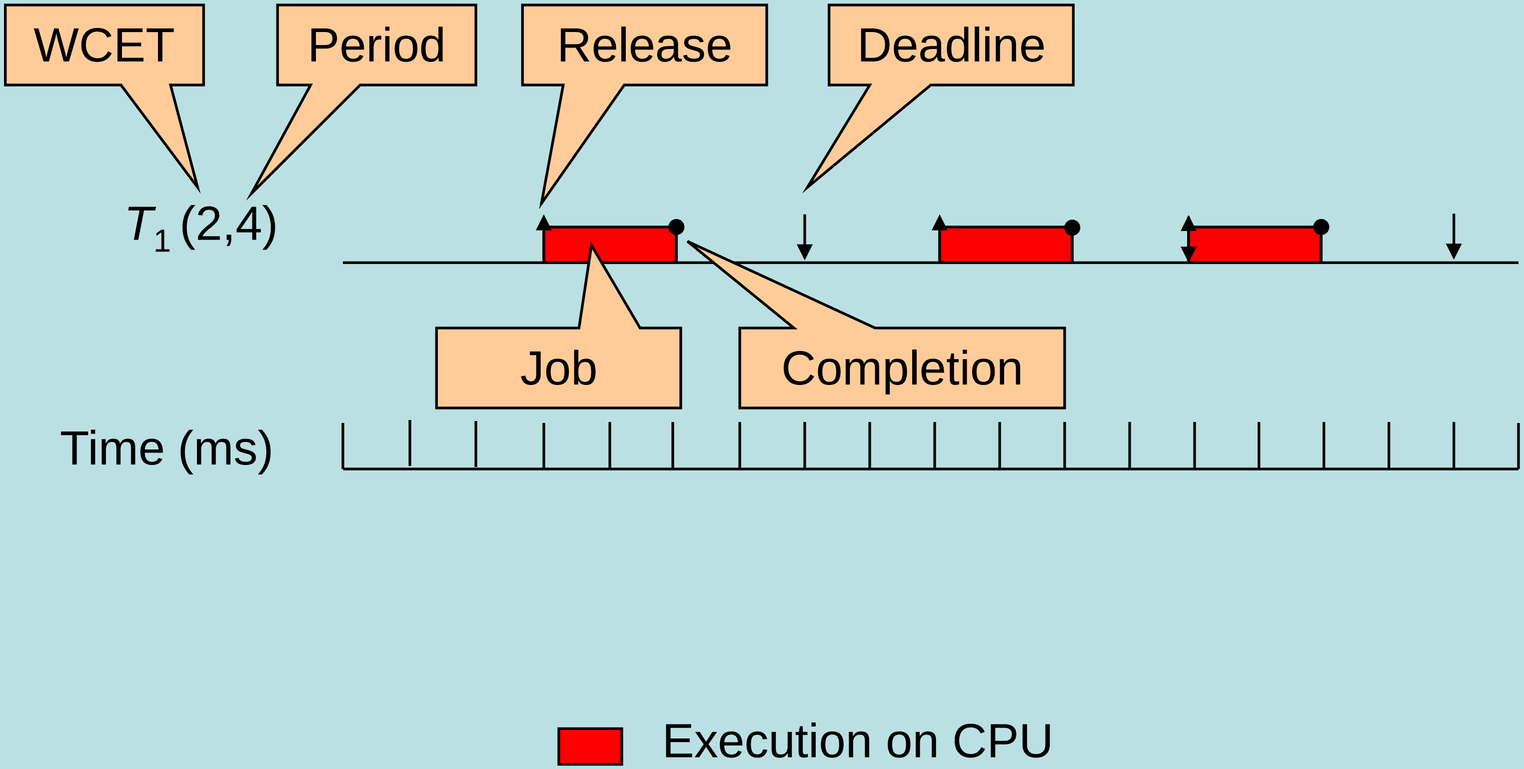
My Work

- Developed specification for a tool to test schedulers
- Implemented prototype of the tool for the G-EDF scheduling policy

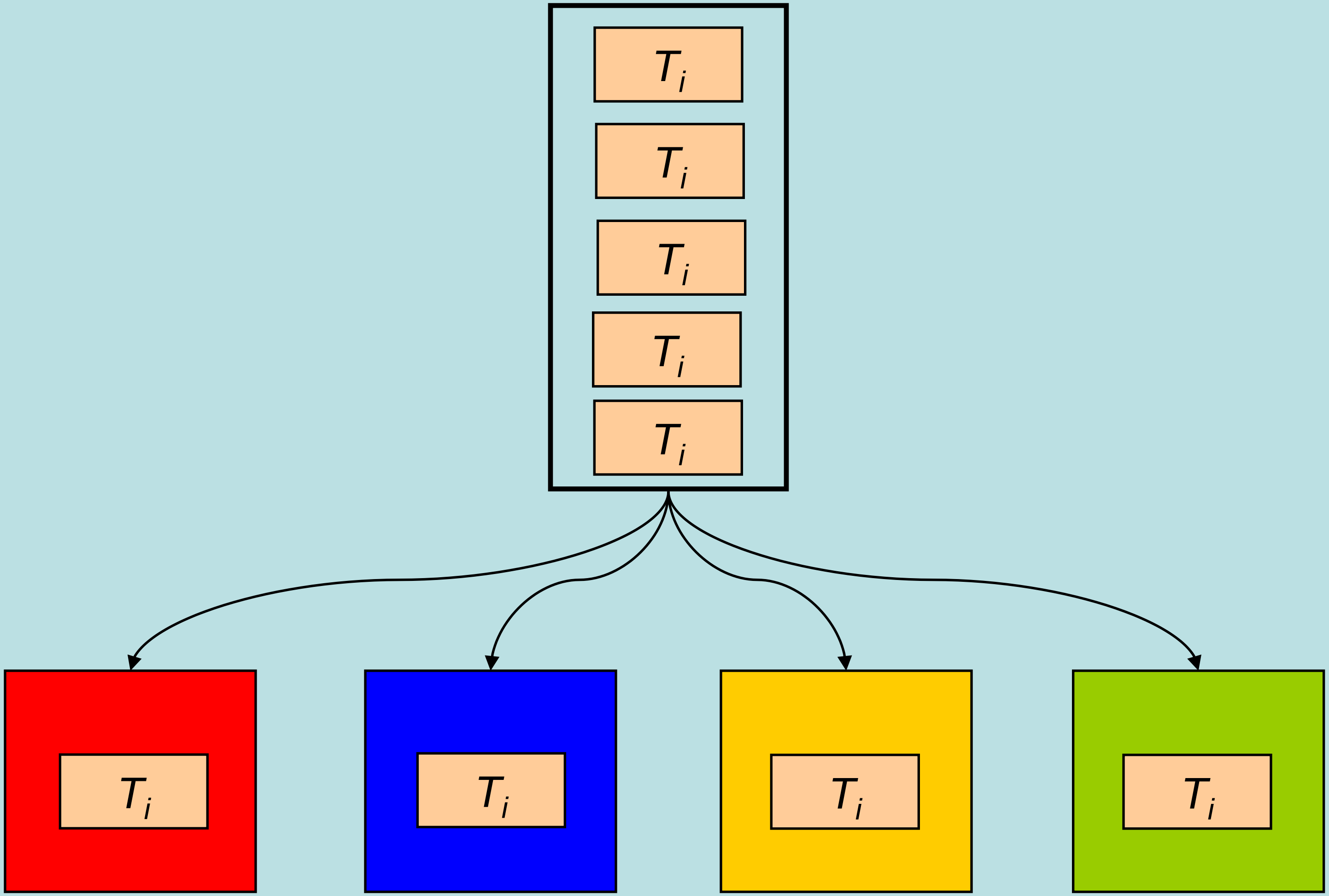
Towards Unit Testing Real-Time Schedulers in LITMUS^{RT}

- What is LITMUS^{RT}?
- What is this talk about?
- What is a typical scheduling policy?
- Why do we need a test tool?
- How do we test? (Answer: Unit Testing)
- What are the specific tests?

Sporadic Task Model

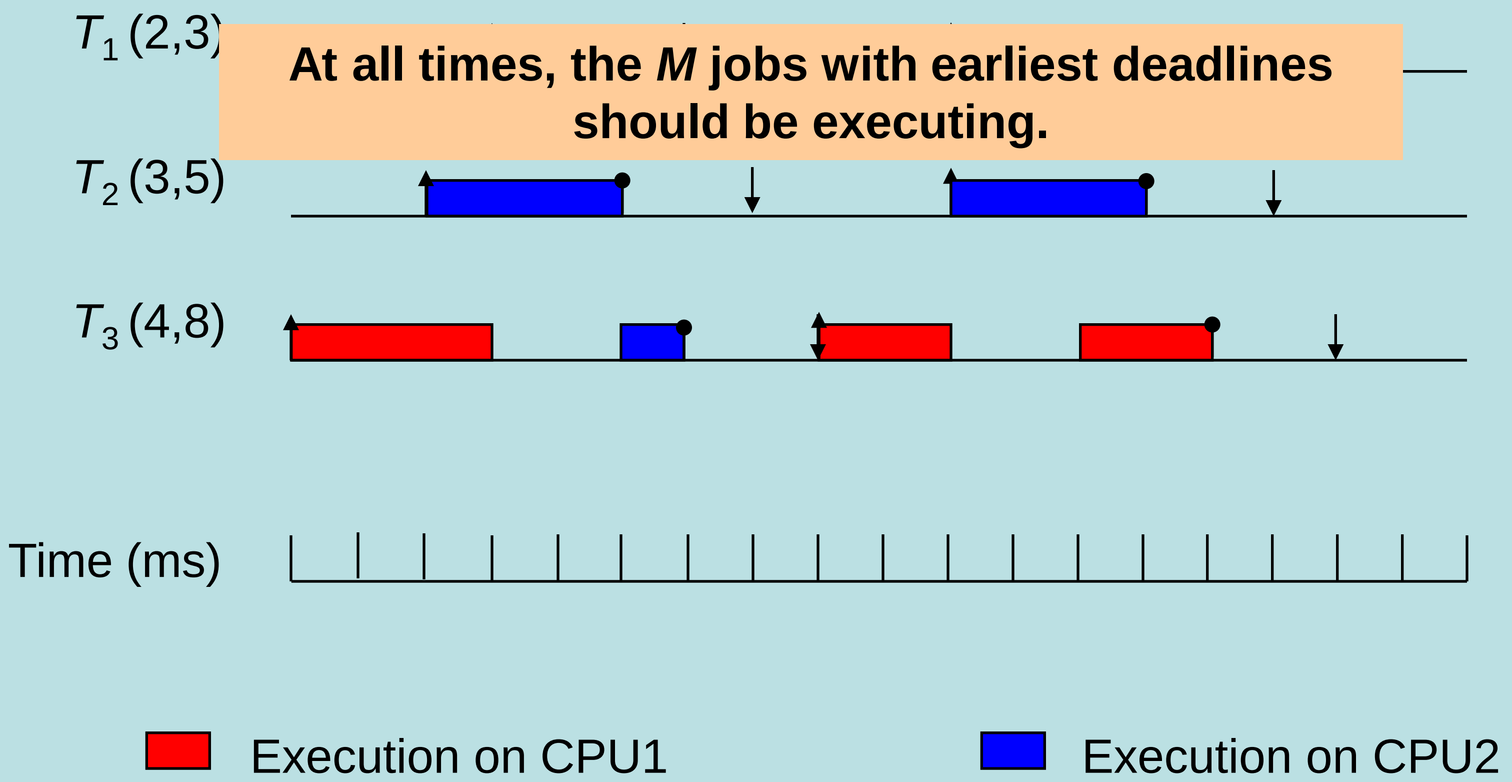


Global Scheduling Policies

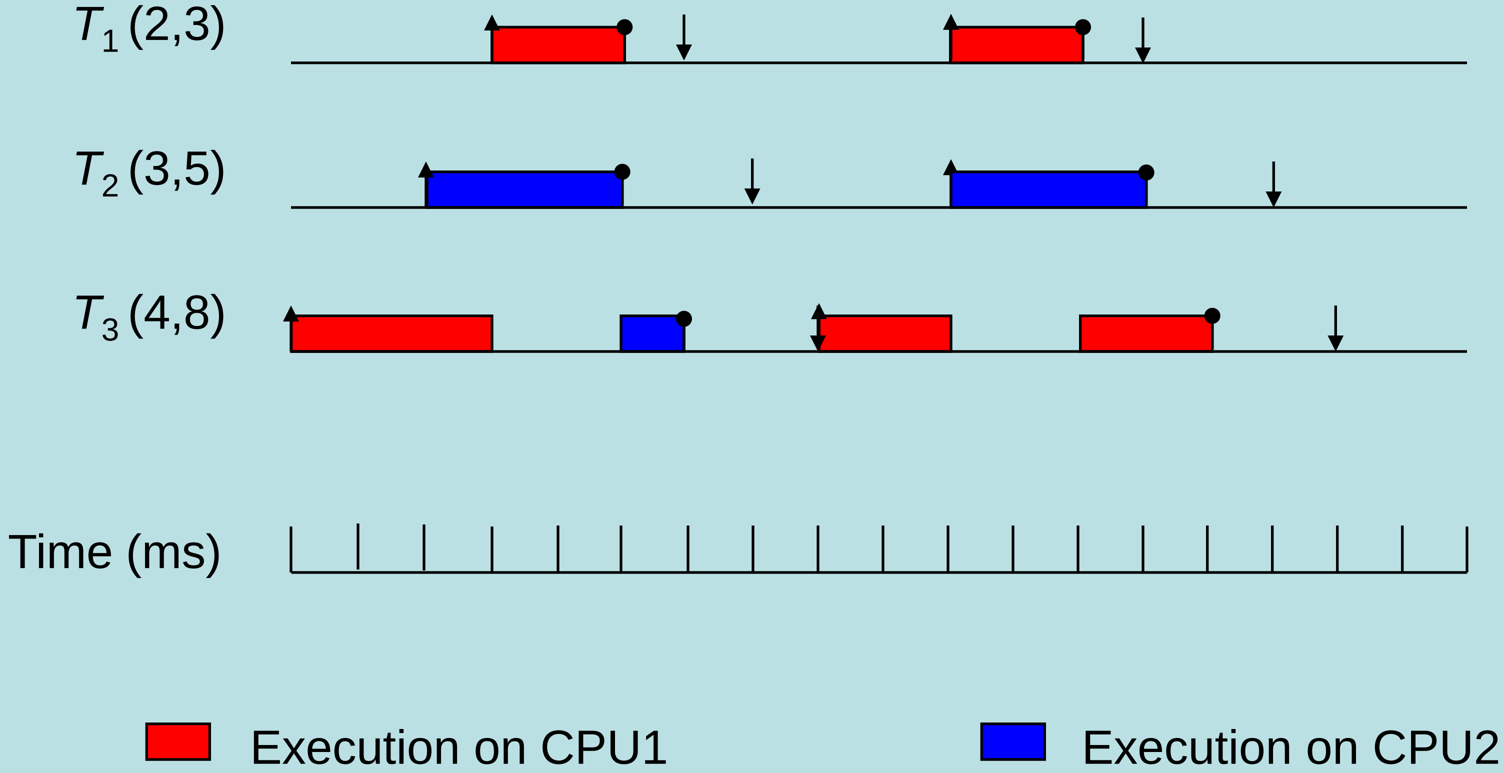


M CPUs CPU 1 CPU 2 CPU 3 CPU 4

Global Earliest Deadline First (G-EDF)



Global Earliest Deadline First (G-EDF)



Towards Unit Testing Real-Time Schedulers in LITMUS^{RT}

- What is LITMUS^{RT}?
- What is this talk about?
- What is a typical scheduling policy?
- Why do we need a test tool?
- How do we test? (Answer: Unit Testing)
- What are the specific tests?

Why Develop a Test Tool?

1) Code is very complex
   - We cannot easily tell if the code is correct
   - We need help debugging

1 sched_litmu... sched_gsn_... 00:44

sched_litmus.c (~/kernels/litmus2008-mod/litmus) - GVIM

File Edit Tools Syntax Buffers Window Help

```c
    litmus->tick(p);
}

#define NO_CPU -1

static void litmus_schedule(struct rq *rq, struct task_struct *prev)
{
    struct rq* other_rq;
    long prev_state;
    lt_t _maybe_deadlock = 0;
    /* WARNING: rq is _not_ locked! */
    if (is_realtime(prev))
        update_time_litmus(rq, prev);

    /* let the plugin schedule */
    rq->litmus_next = litmus->schedule(prev);

    /* check if a global plugin pulled a task from a different RQ */
    if (rq->litmus_next && task_rq(rq->litmus_next) != rq) {
        /* we need to migrate the task */
        other_rq = task_rq(rq->litmus_next);
        TRACE_TASK(rq->litmus_next, "migrate from %d\n", other_rq->cpu

        /* while we drop the lock, the prev task could change its
         * state
         */
        prev_state = prev->state;
        mb();
        spin_unlock(&rq->lock);

        /* Don't race with a concurrent switch.
         * This could deadlock in the case of cross or circular migrat
         * It's the job of the plugin to make sure that doesn't happen
         */
        TRACE_TASK(rq->litmus_next, "stack_in_use=%d\n",
                   rq->litmus_next->rt_param.stack_in_use);
        if (rq->litmus_next->rt_param.stack_in_use != NO_CPU) {
            TRACE_TASK(rq->litmus_next, "waiting to deschedule\n");
            _maybe_deadlock = litmus_clock();
        }
        while (rq->litmus_next->rt_param.stack_in_use != NO_CPU) {
            cpu_relax();
```

36,42-45

sched_gsn_edf.c (~/kernels/litmus2008-mod/litmus) - GVIM1

File Edit Tools Syntax Buffers Window Help

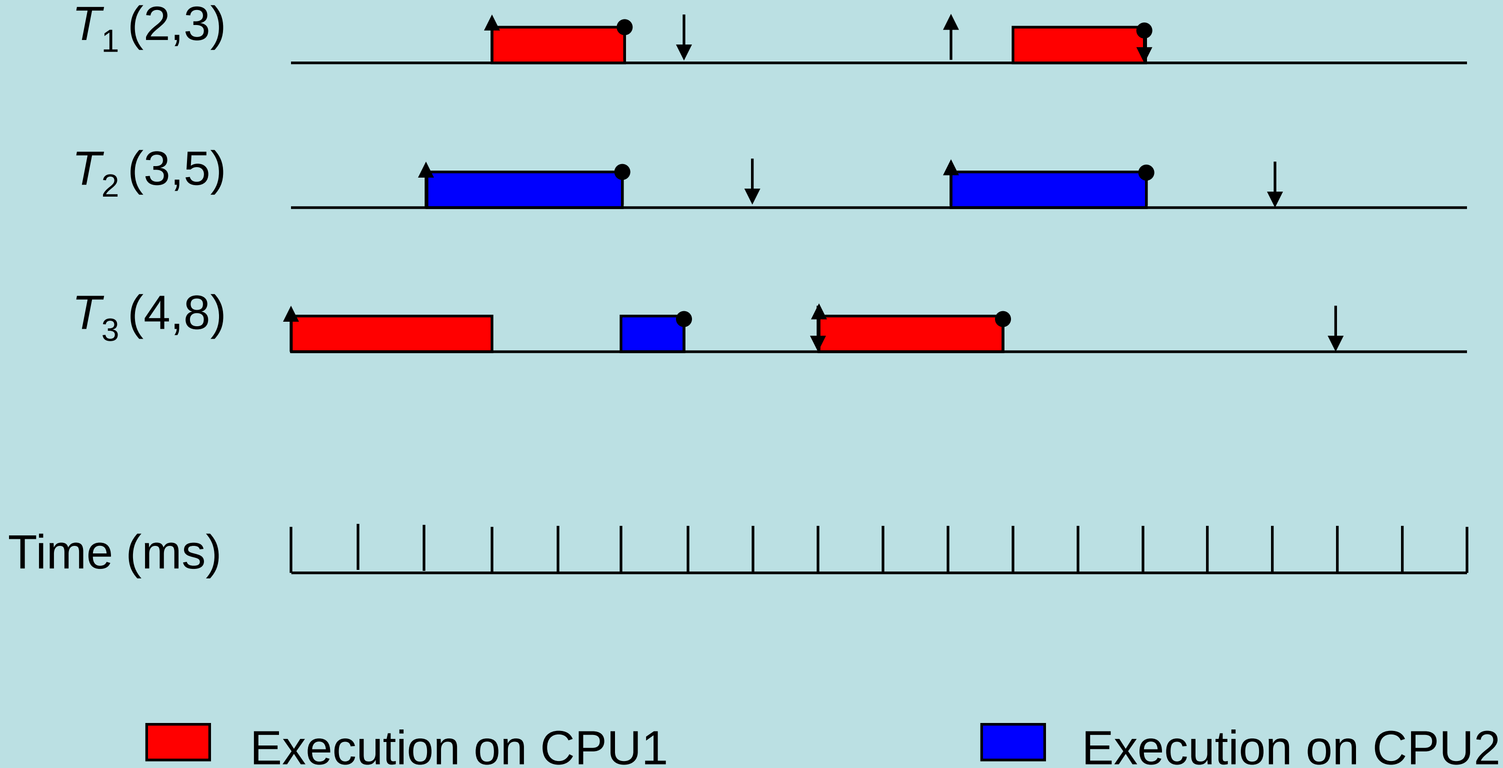
```c
    for(last = lowest_prio_cpu();
        edf_preemption_needed(&gsnedf, last->linked);
        last = lowest_prio_cpu()) {
        /* preemption necessary */
        task = __take_ready(&gsnedf);
        TRACE("check_for_preemptions: attempting to link task %d to %
              task->pid, last->cpu);
        if (last->linked)
            requeue(last->linked);
        link_task_to_cpu(task, last);
        preempt(last);
    }
}

/* gsnedf_job_arrival: task is either resumed or released */
static noinline void gsnedf_job_arrival(struct task_struct* task)
{
    BUG_ON(!task);

    requeue(task);
    check_for_preemptions();
}

static void gsnedf_release_jobs(rt_domain_t* rt, struct heap* tasks)
{
    unsigned long flags;

    spin_lock_irqsave(&gsnedf_lock, flags);

    __merge_ready(rt, tasks);
    check_for_preemptions();

    spin_unlock_irqrestore(&gsnedf_lock, flags);
}

/* caller holds gsnedf_lock */
static noinline void job_completion(struct task_struct *t, int forced
{
    BUG_ON(!t);

    sched_trace_task_completion(t, forced);
```

286,2-5

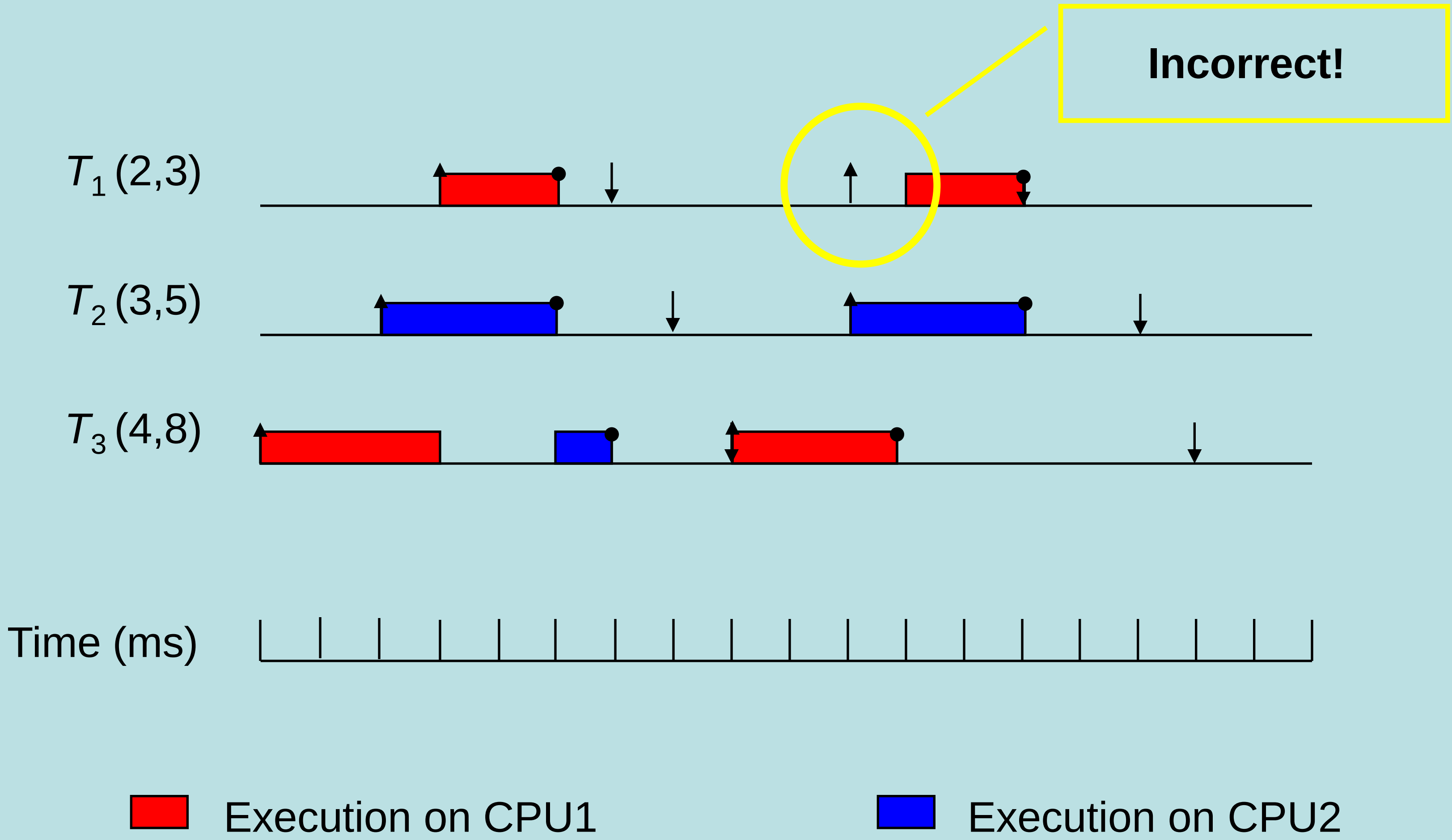
Why Develop a Test Tool?

1) Code is very complex
   - We cannot easily tell if the code is correct
   - We need help debugging
2) Resulting schedules are very complex
   - We cannot easily tell if correct schedules are produced

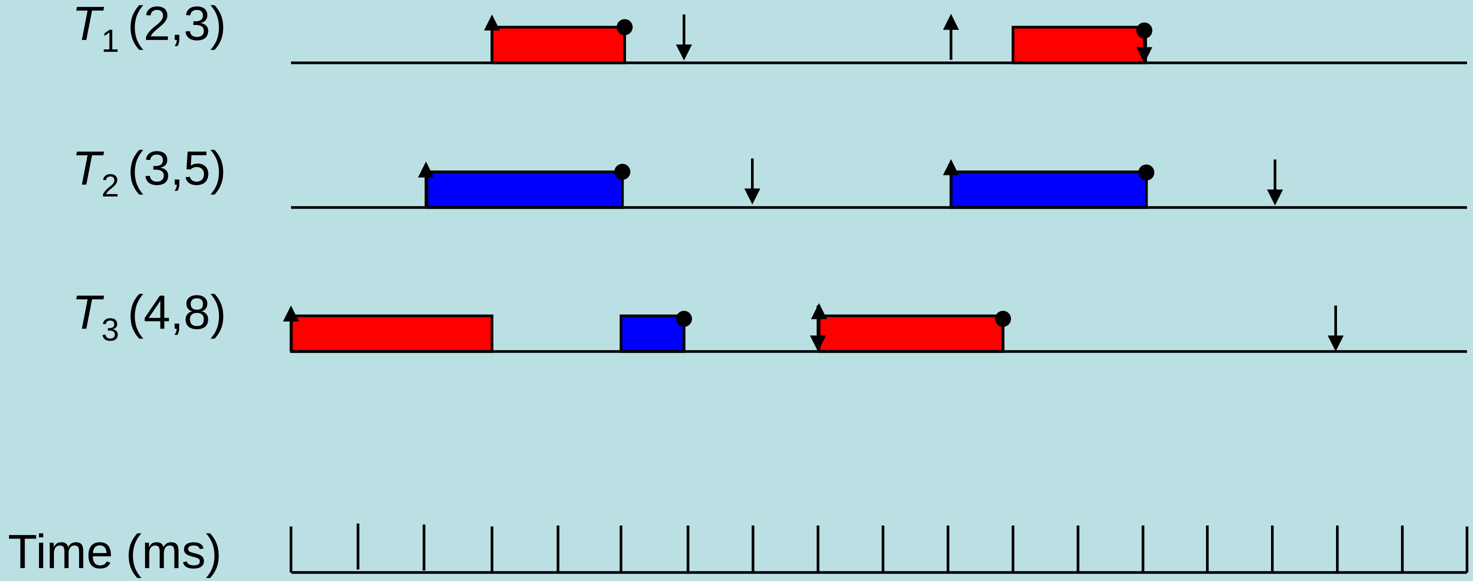
Global Earliest Deadline First (G-EDF) ???



Global Earliest Deadline First (G-EDF)



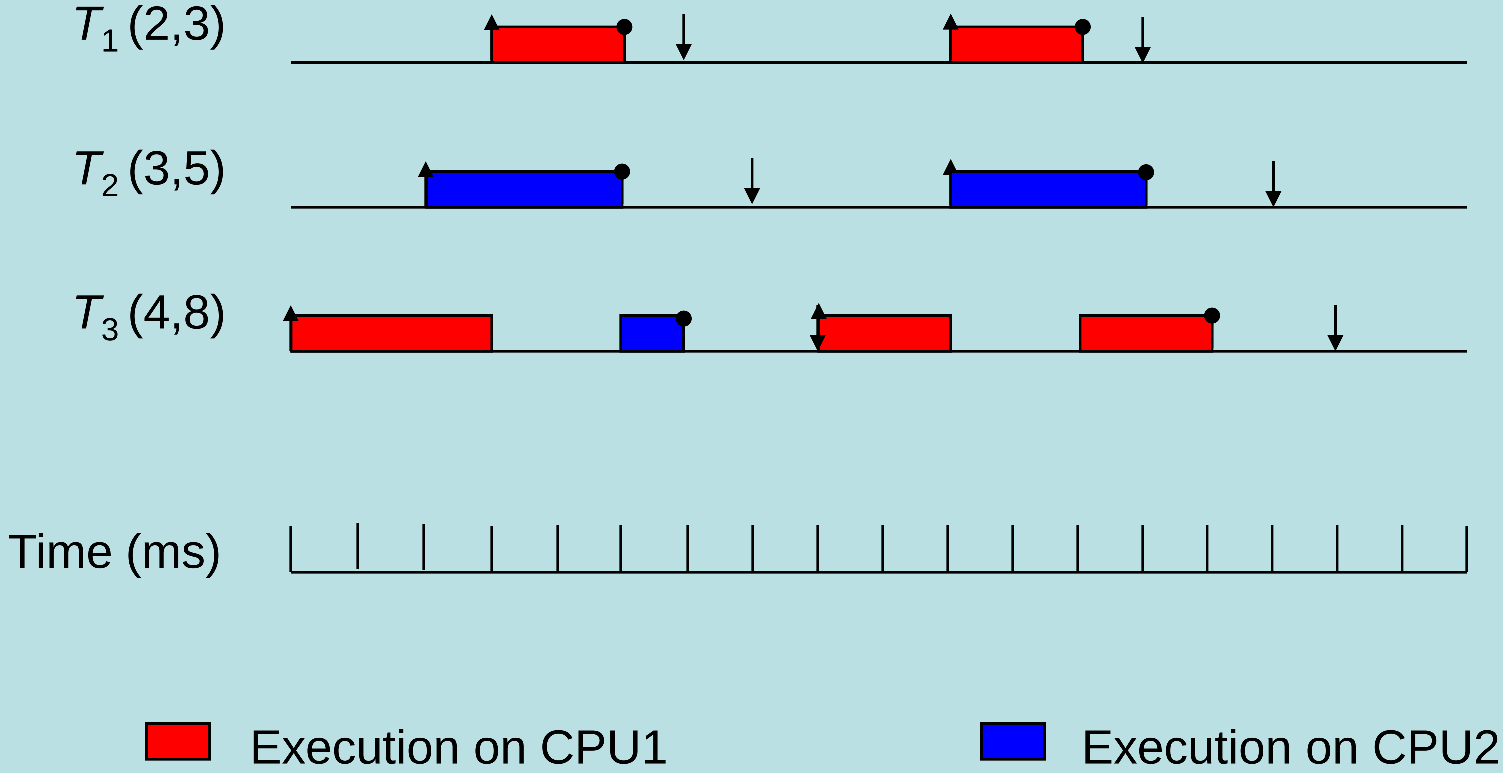
Global Earliest Deadline First (G-EDF)



Execution on CPU1

Execution on CPU2

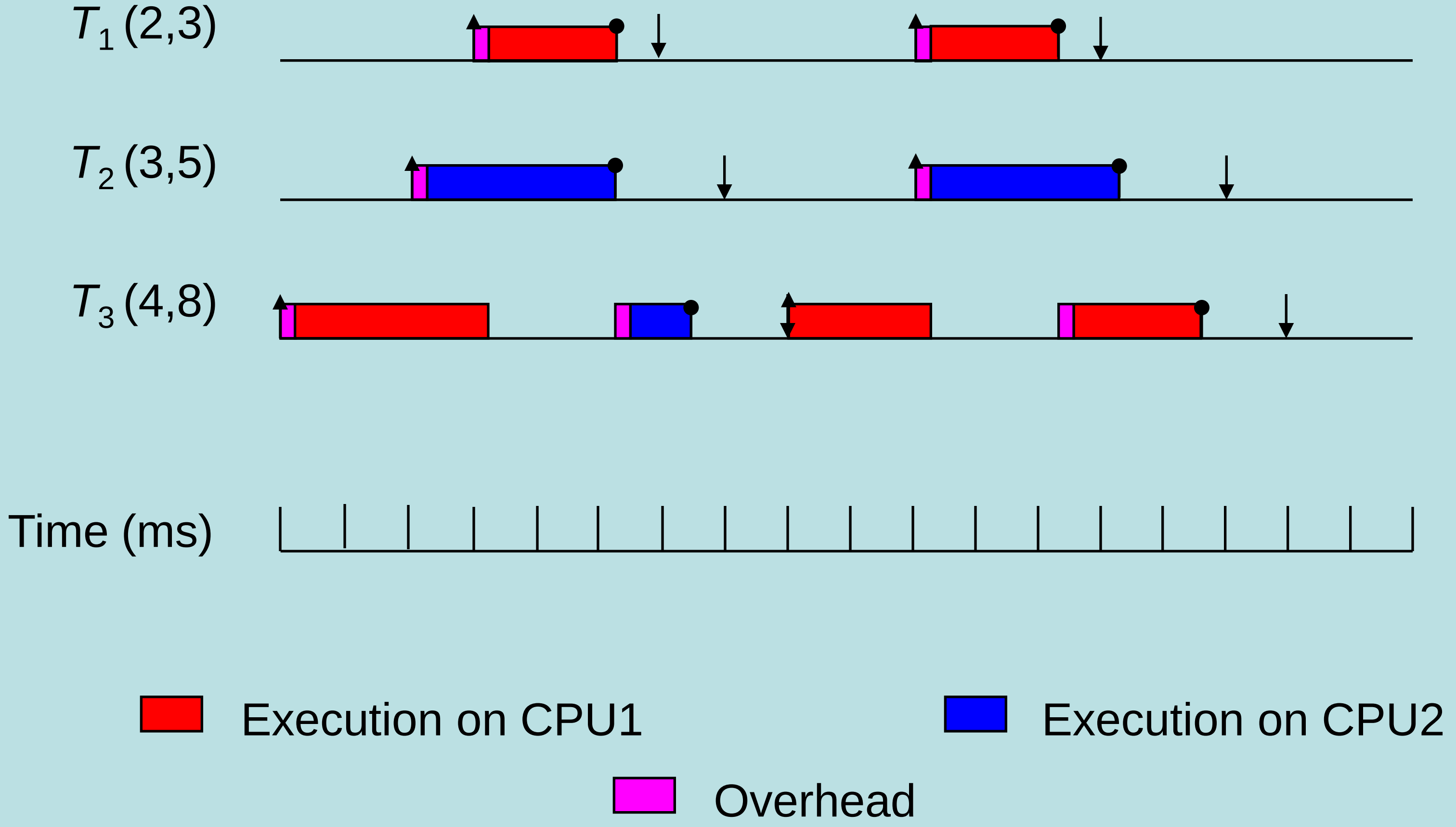
Global Earliest Deadline First (G-EDF)



Why Develop a Test Tool?

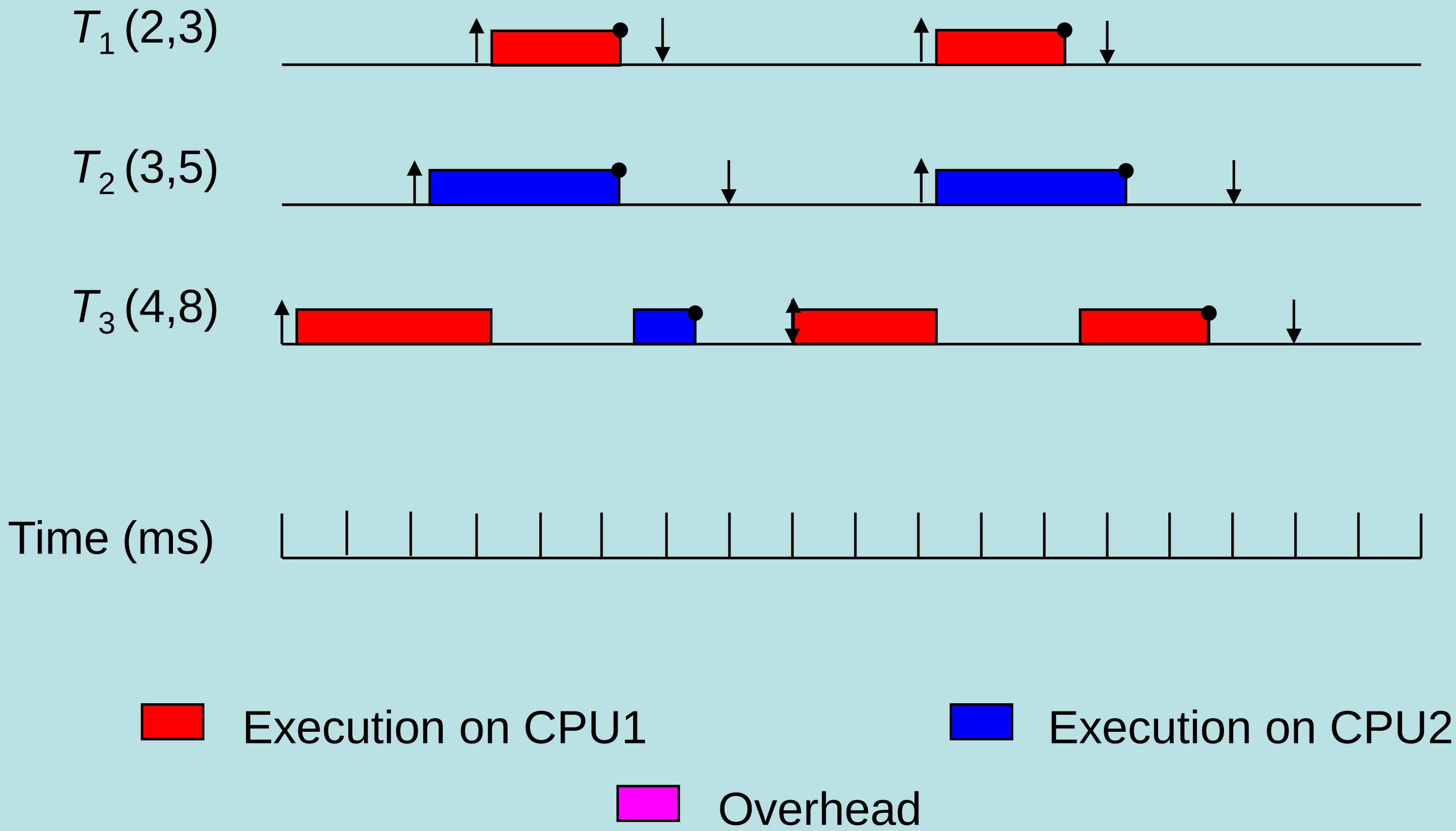
1) Code is very complex
   - We cannot easily tell if the code is correct
   - We need help debugging
2) Resulting schedules are very complex
   - We cannot easily tell if correct schedules are produced
3) We need to minimize overhead
   - Detailed regression testing is necessary

Overhead

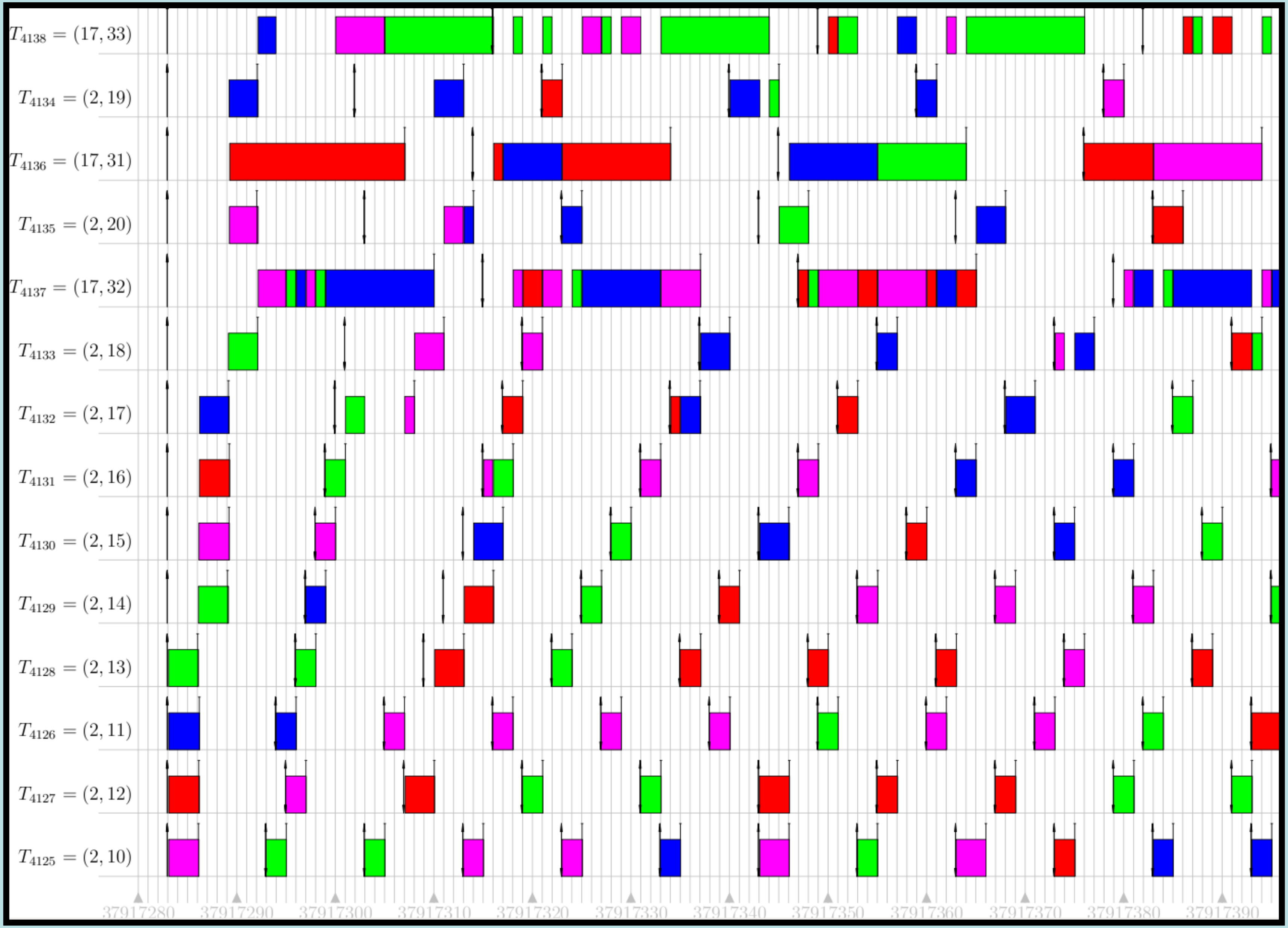


T_1 (2,3)
T_2 (3,5)
T_3 (4,8)
Time (ms)
Execution on CPU1
Execution on CPU2
Overhead

Overhead



T_1 (2,3)
T_2 (3,5)
T_3 (4,8)
Time (ms)
Execution on CPU1
Execution on CPU2
Overhead


$T_{4138} = (17, 33)$
$T_{4134} = (2, 19)$
$T_{4136} = (17, 31)$
$T_{4135} = (2, 20)$
$T_{4137} = (17, 32)$
$T_{4133} = (2, 18)$
$T_{4132} = (2, 17)$
$T_{4131} = (2, 16)$
$T_{4130} = (2, 15)$
$T_{4129} = (2, 14)$
$T_{4128} = (2, 13)$
$T_{4126} = (2, 11)$
$T_{4127} = (2, 12)$
$T_{4125} = (2, 10)$
37917280
37917290
37917300
37917310
37917320
37917330
37917340
37917350
37917360
37917370
37917380
37917390

Towards Unit Testing Real-Time Schedulers in LITMUS^{RT}

- What is LITMUS^{RT}?
- What is this talk about?
- What is a typical scheduling policy?
- Why do we need a test tool?
- How do we test? (Answer: Unit Testing)
- What are the specific tests?

The Challenge

Without overhead, and if we did not need very detailed feedback, we could check the invariant: At all times, the *M* jobs with earliest deadlines should be executing.

The Challenge

Instead, we have to use a series of specific tests to detect anomalies in scheduling and to measure overhead.

Unit Testing

A series of specific tests
with detailed feedback
that can be produced after each code revision
suggests
Unit Testing

Unit Testing

A **series of specific tests**
with **detailed feedback**
that can be **produced after each code revision**
suggests
Unit Testing

Unit Testing: programmatically testing small modules of *code* after each revision

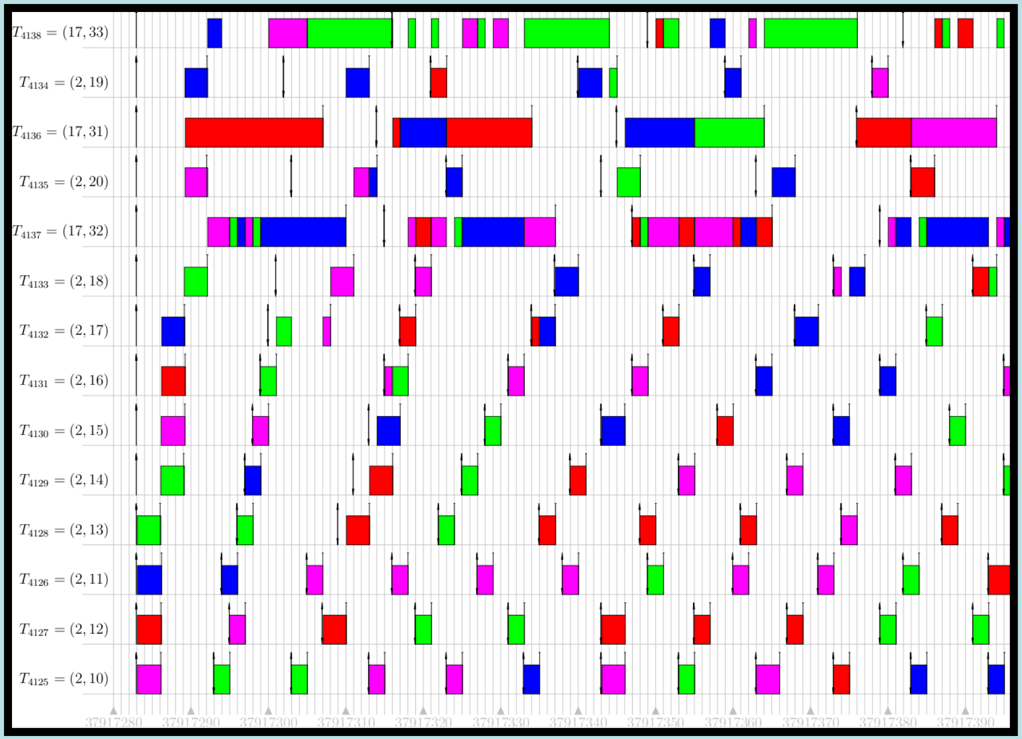
Unit Testing

A **series of specific tests**
with **detailed feedback**
that can be **produced after each code revision**
suggests
Unit Testing

Unit Testing: programmatically testing small modules of *code* after each revision

We test recorded schedule “traces” instead

Feather-Trace



Event records

Release

Switch To

Switch Away

Completion

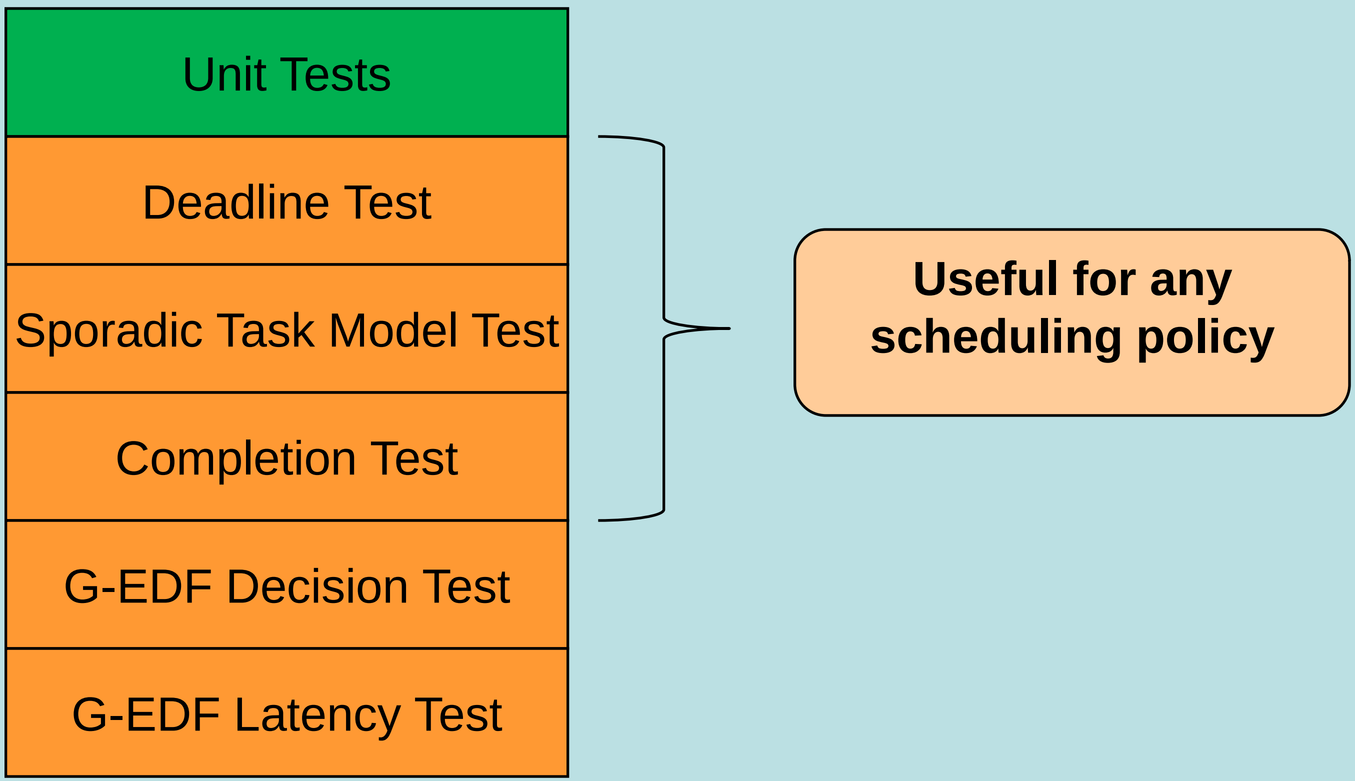
Towards Unit Testing Real-Time Schedulers in LITMUS^{RT}

- What is LITMUS^{RT}?
- What is this talk about?
- What is a typical scheduling policy?
- Why do we need a test tool?
- How do we test? (Answer: Unit Testing)
- What are the specific tests?

Unit Tests

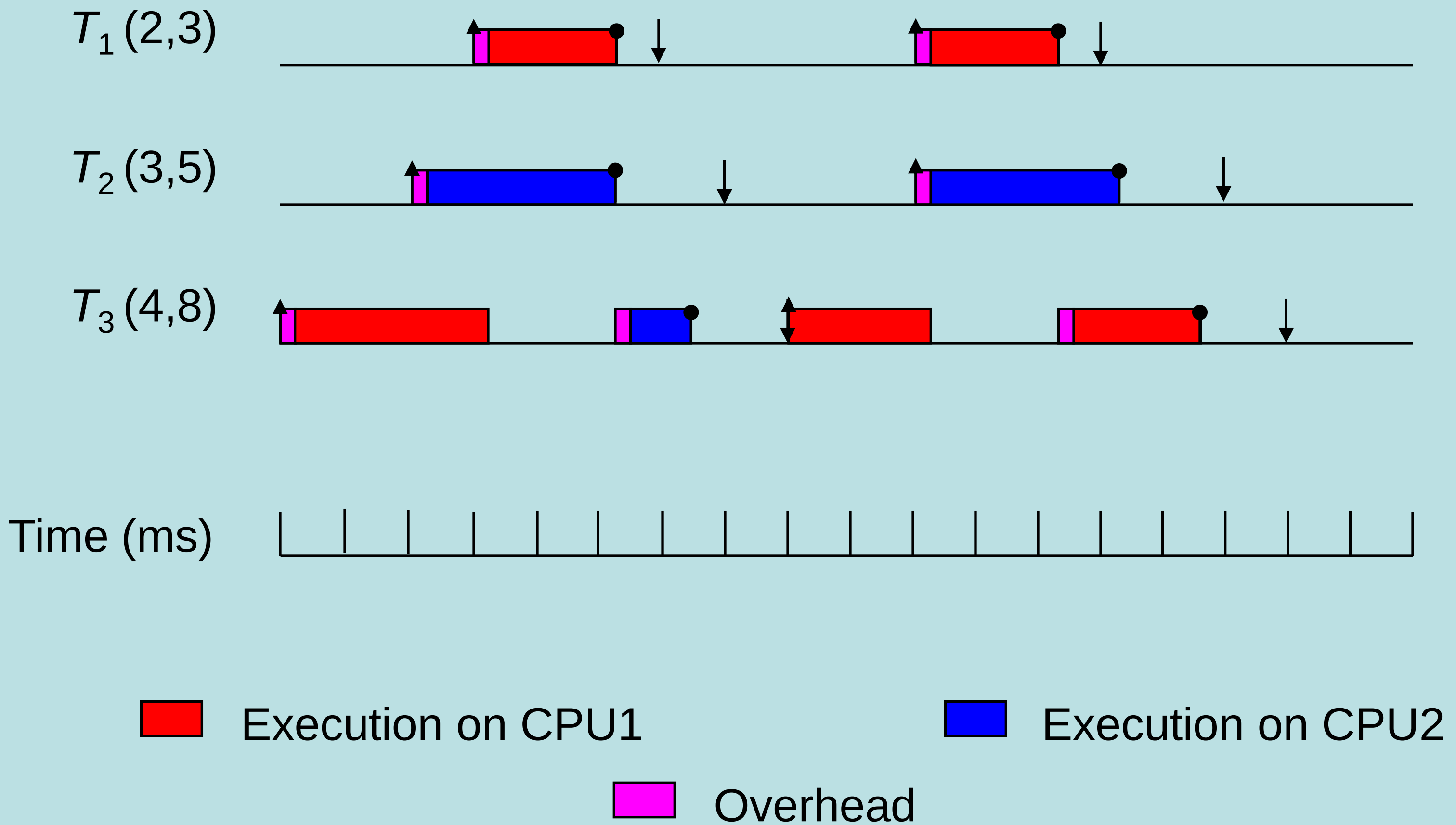
Unit Tests
Deadline Test
Sporadic Task Model Test
Completion Test
G-EDF Decision Test
G-EDF Latency Test

Unit Tests


Unit Tests
Deadline Test
Sporadic Task Model Test
Completion Test
G-EDF Decision Test
G-EDF Latency Test
Useful for any scheduling policy

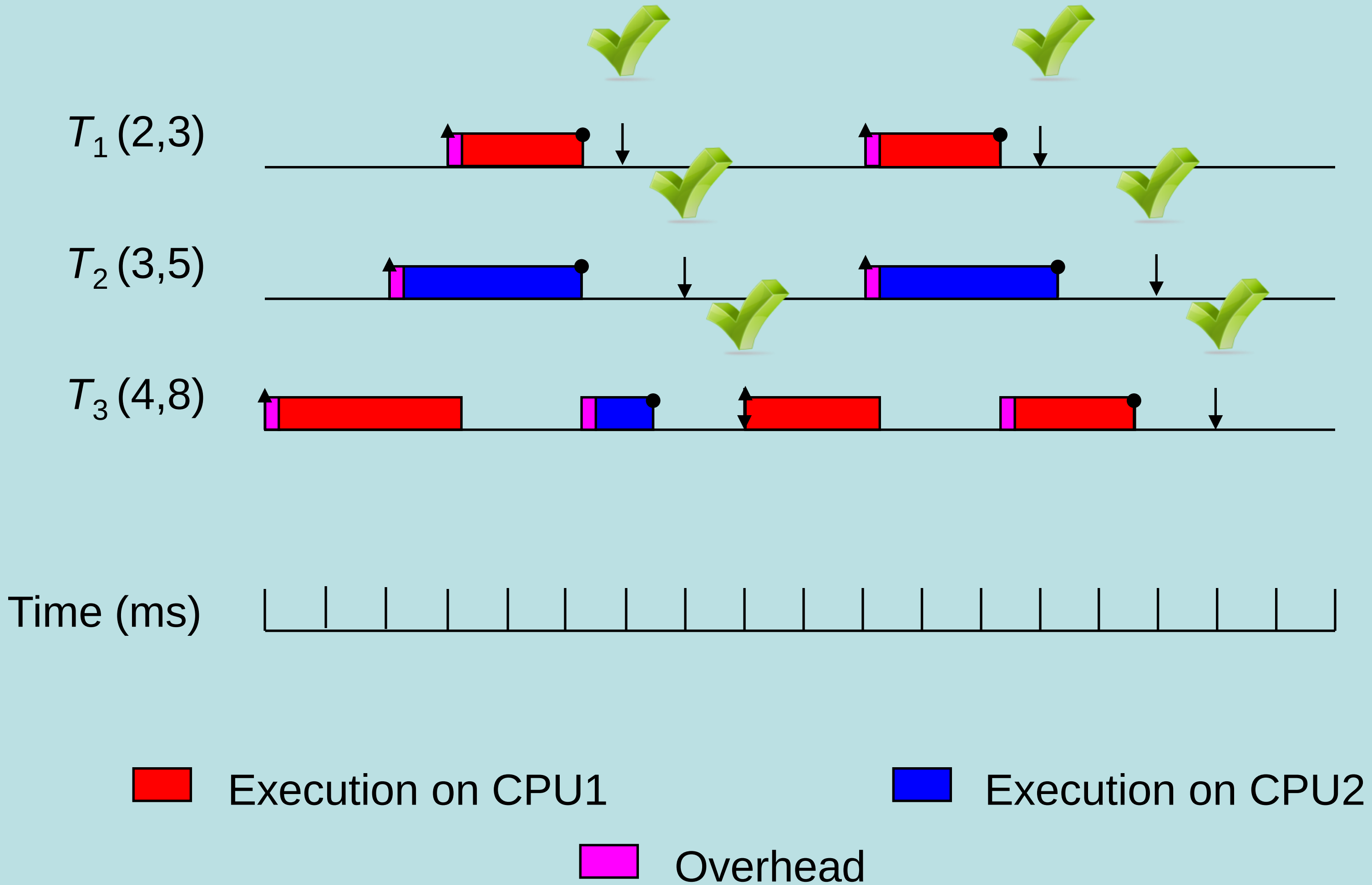
Deadline Test

Did all jobs complete by their deadlines?



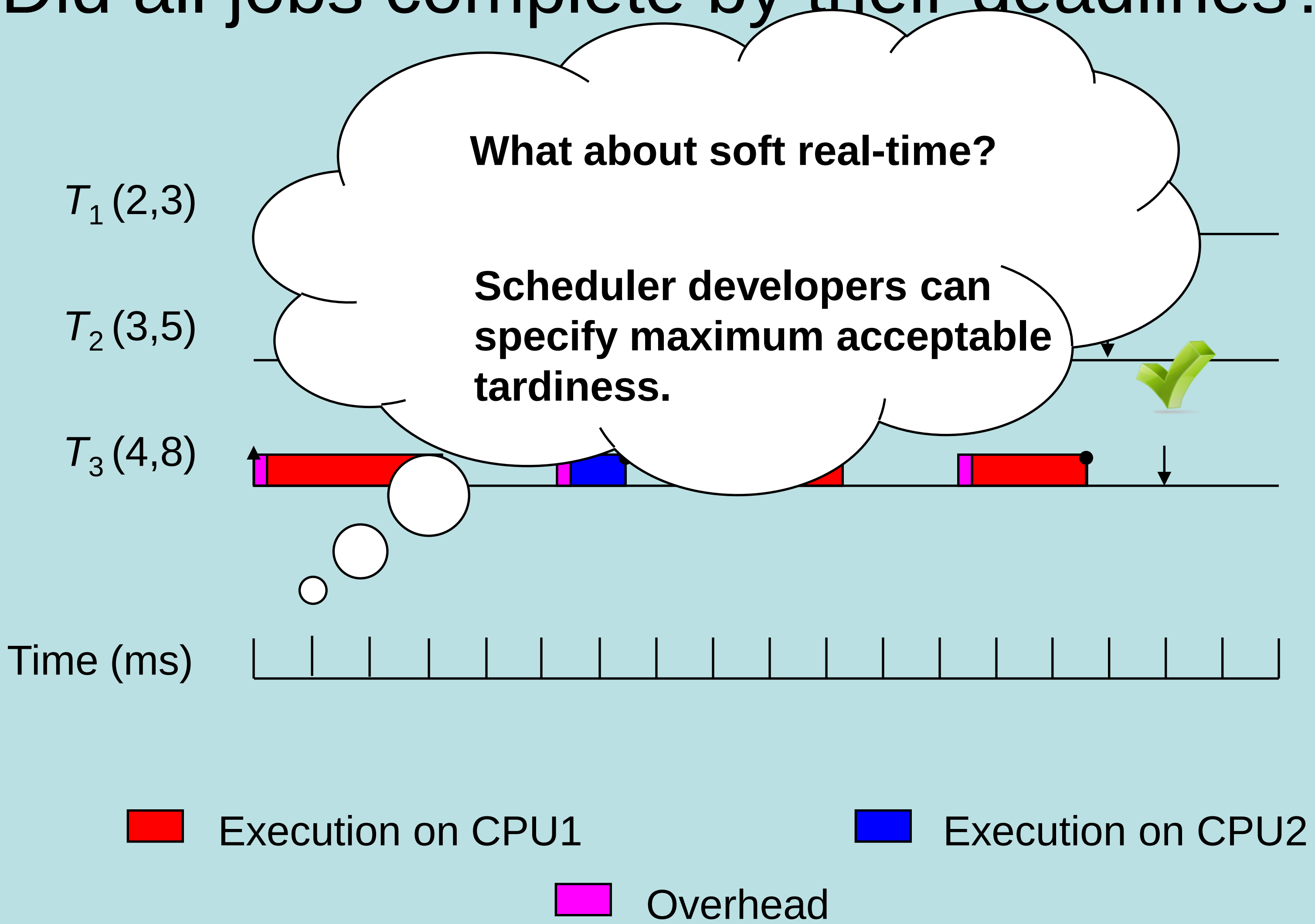
Deadline Test

Did all jobs complete by their deadlines?



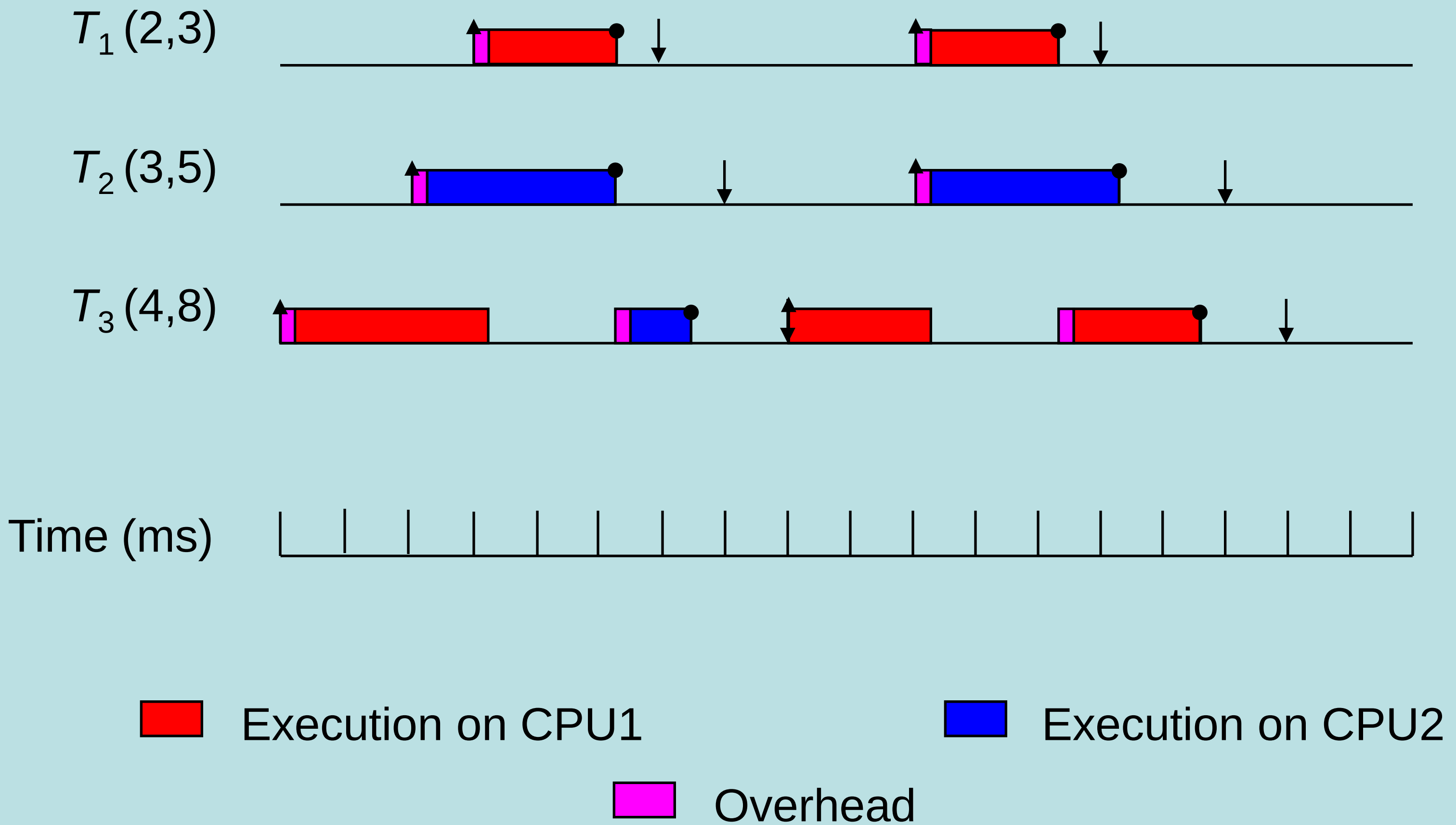
Deadline Test

Did all jobs complete by their deadlines?

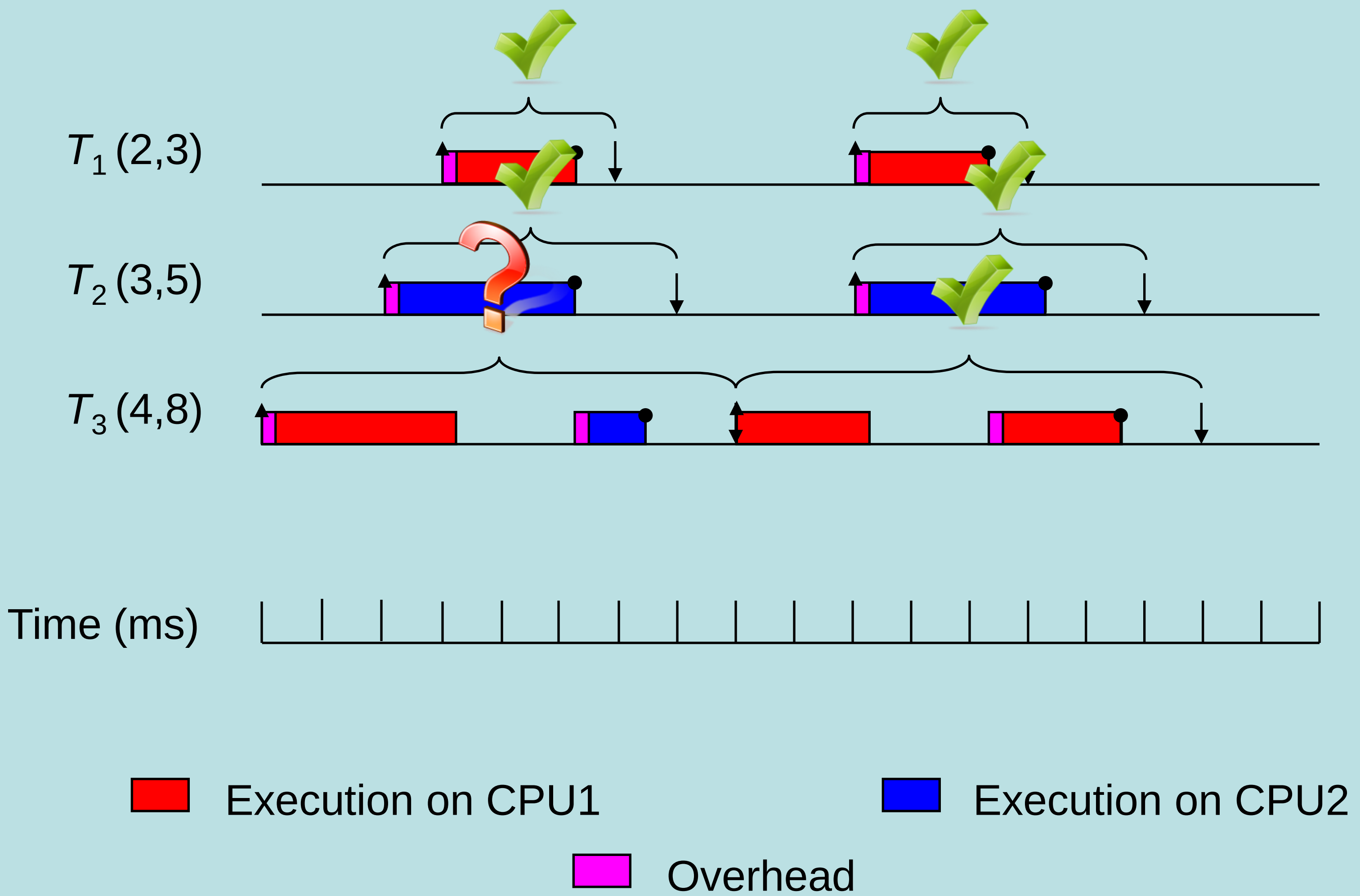


Sporadic Task Model Test

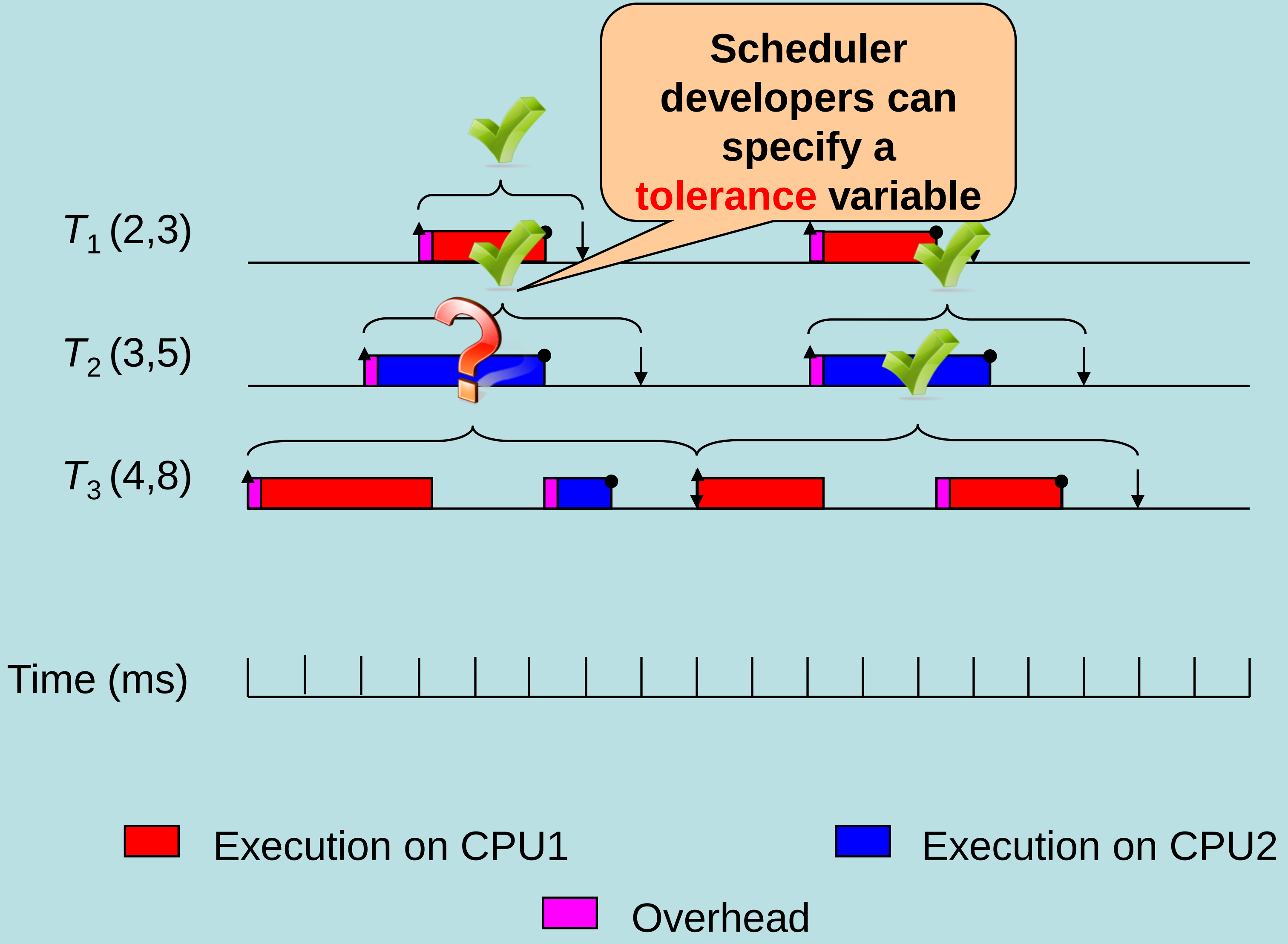
Were job releases separated by at least the period of the task?



Sporadic Task Model Test

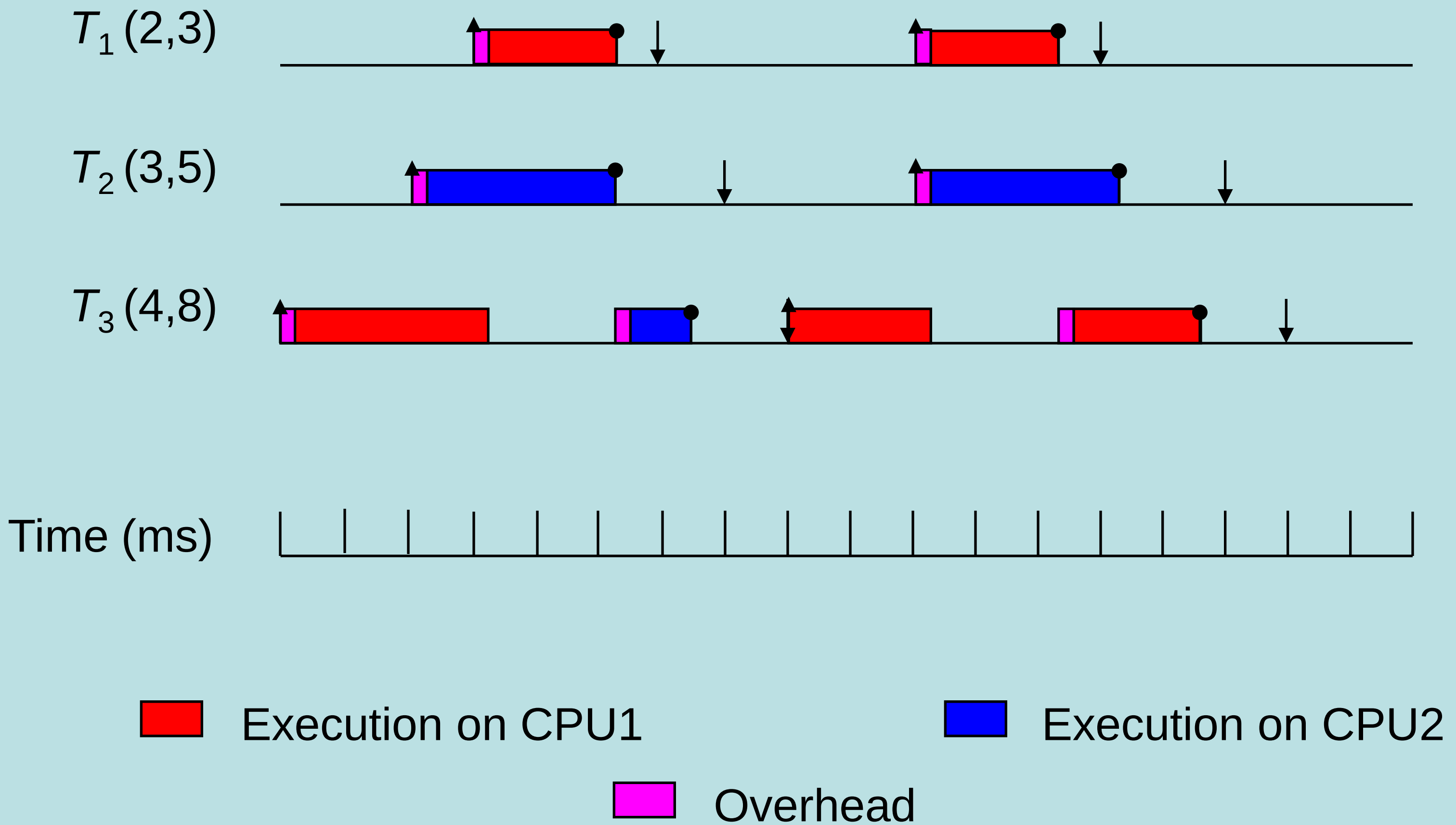


Sporadic Task Model Test



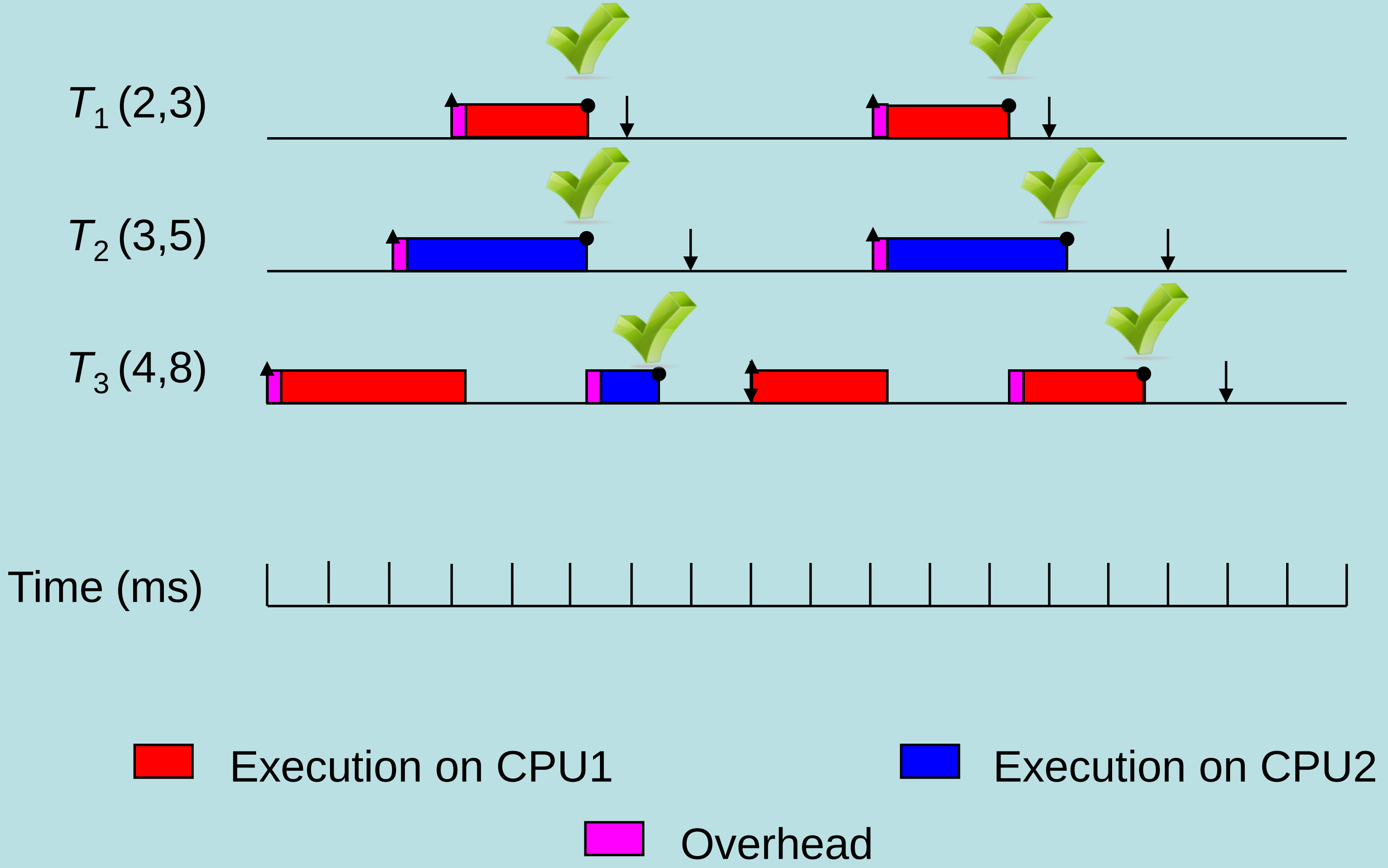
Completion Test

Did all released jobs actually complete?

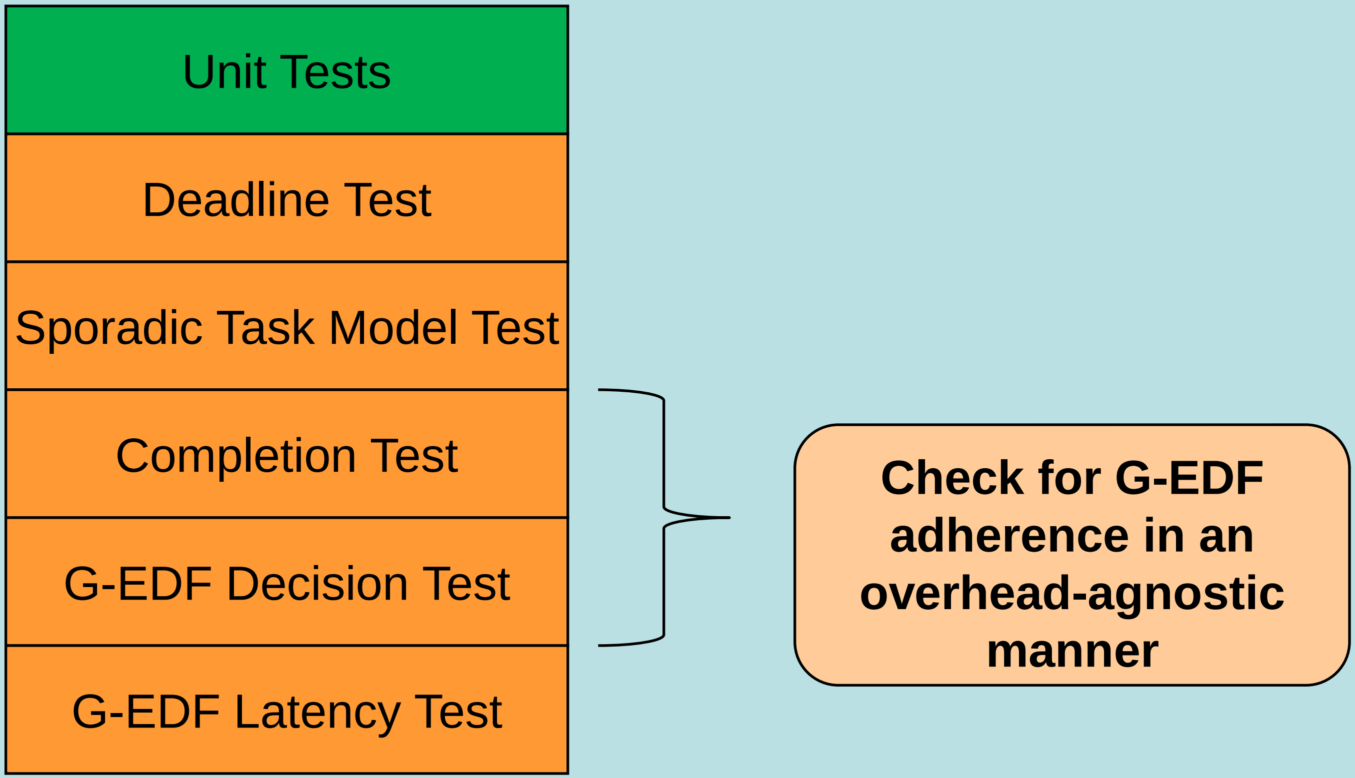


Completion Test

Did all released jobs actually complete?

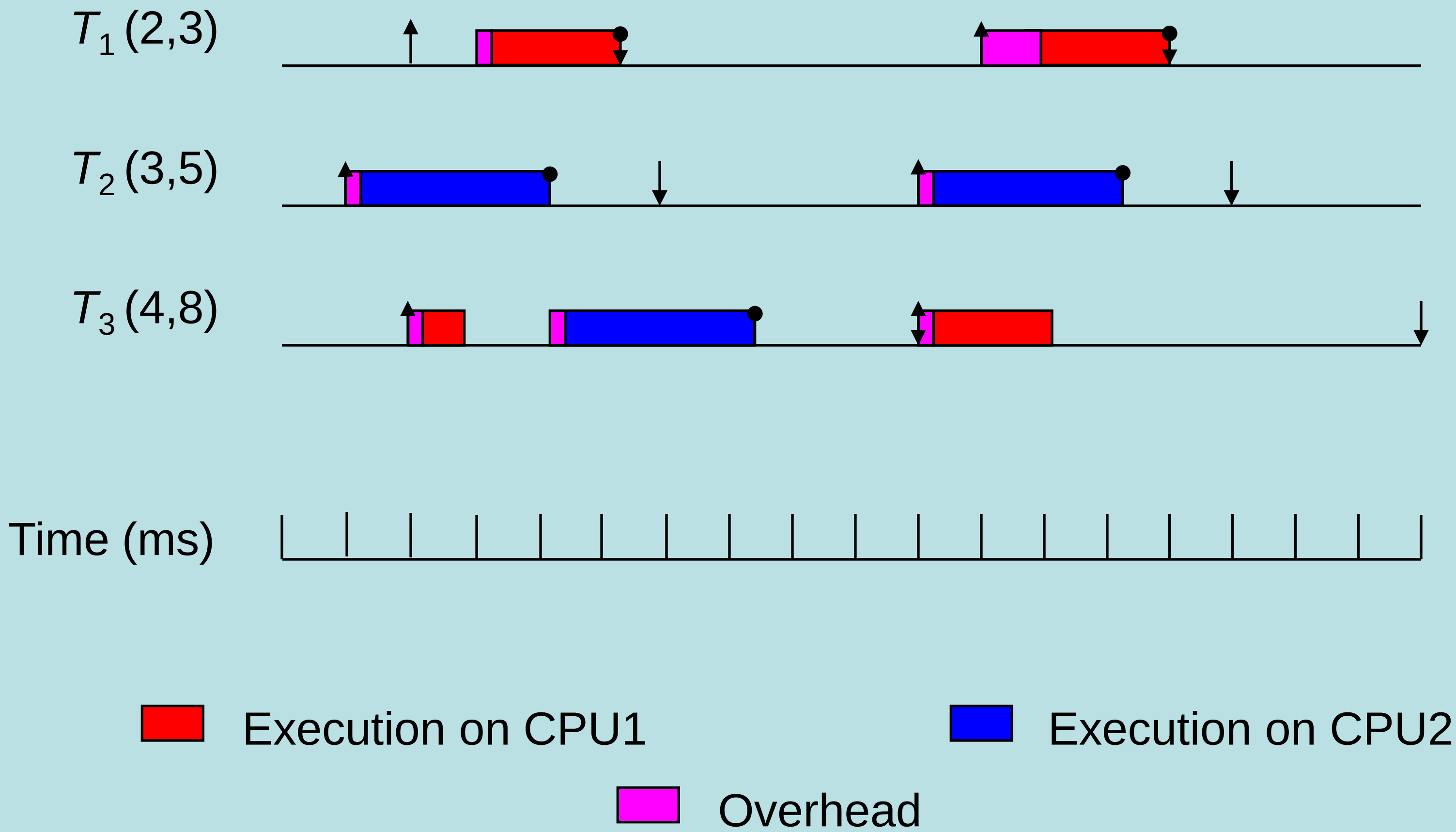


Unit Tests


Unit Tests
Deadline Test
Sporadic Task Model Test
Completion Test
G-EDF Decision Test
G-EDF Latency Test
Check for G-EDF adherence in an overhead-agnostic manner

G-EDF Decision Test

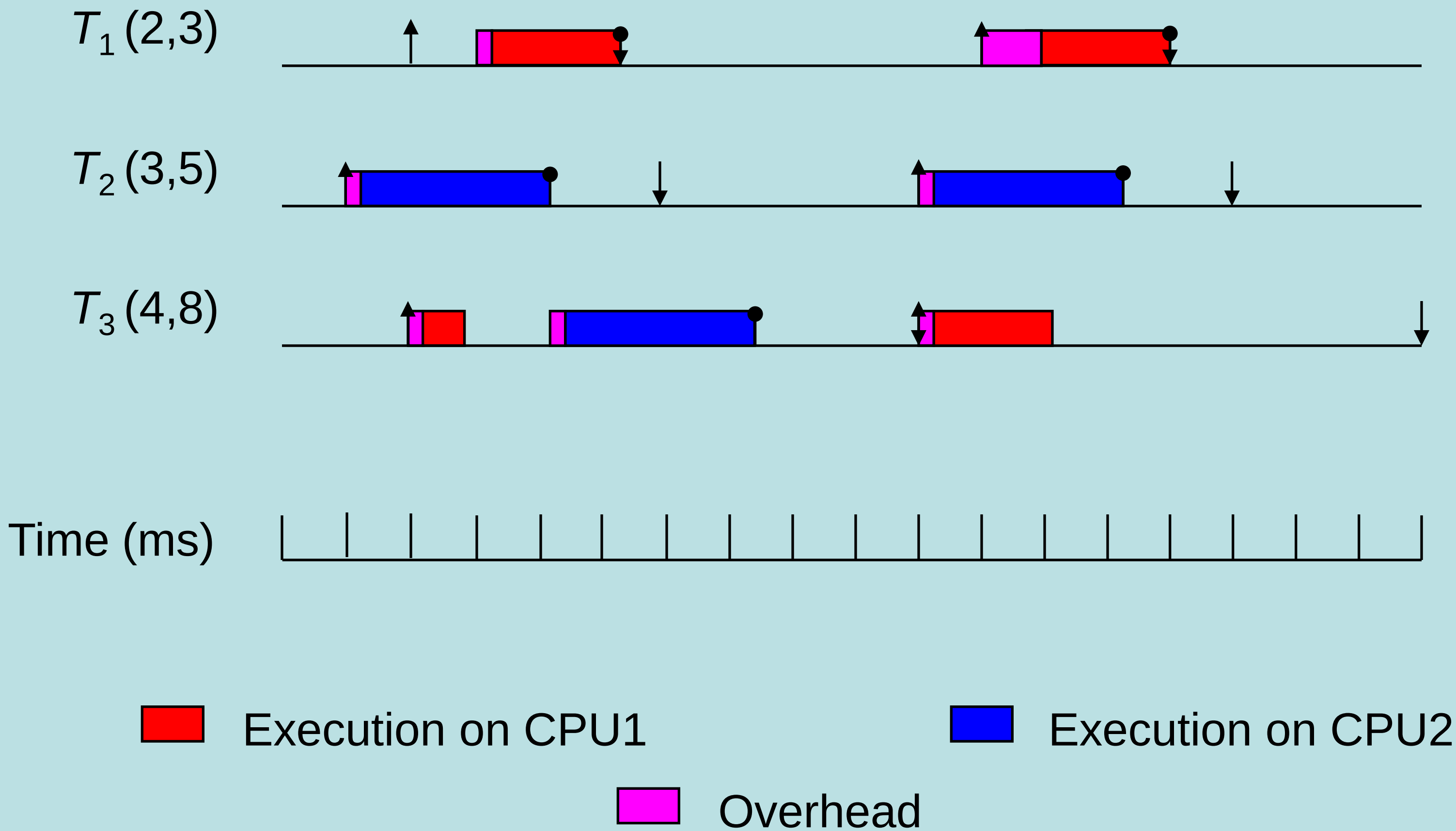
Are jobs switched to execution in EDF order?



G-EDF Decision Test

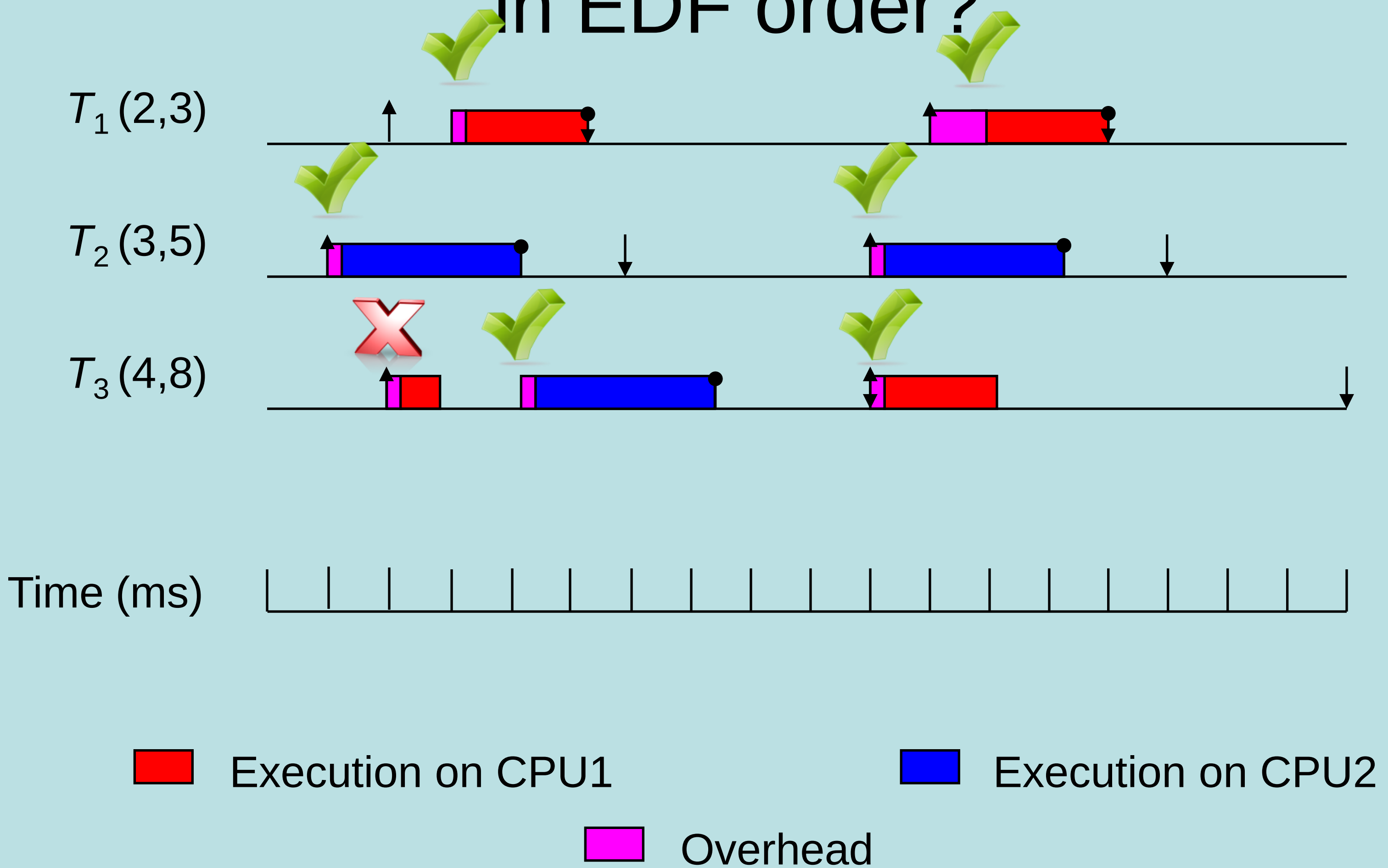
Are jobs … ecution

Test algorithm models execution state to check for correct decisions.



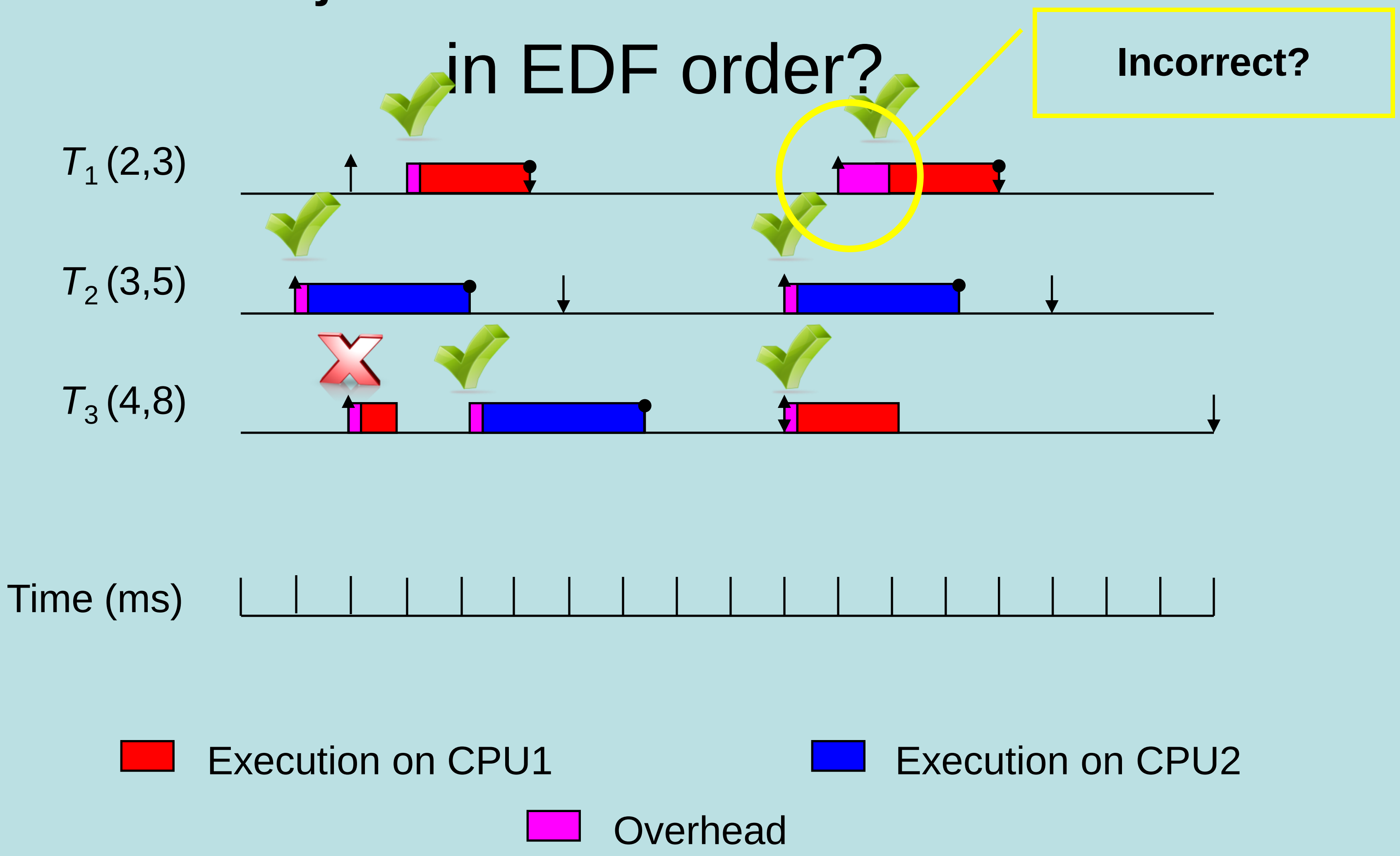
G-EDF Decision Test

Are jobs switched to execution in EDF order?



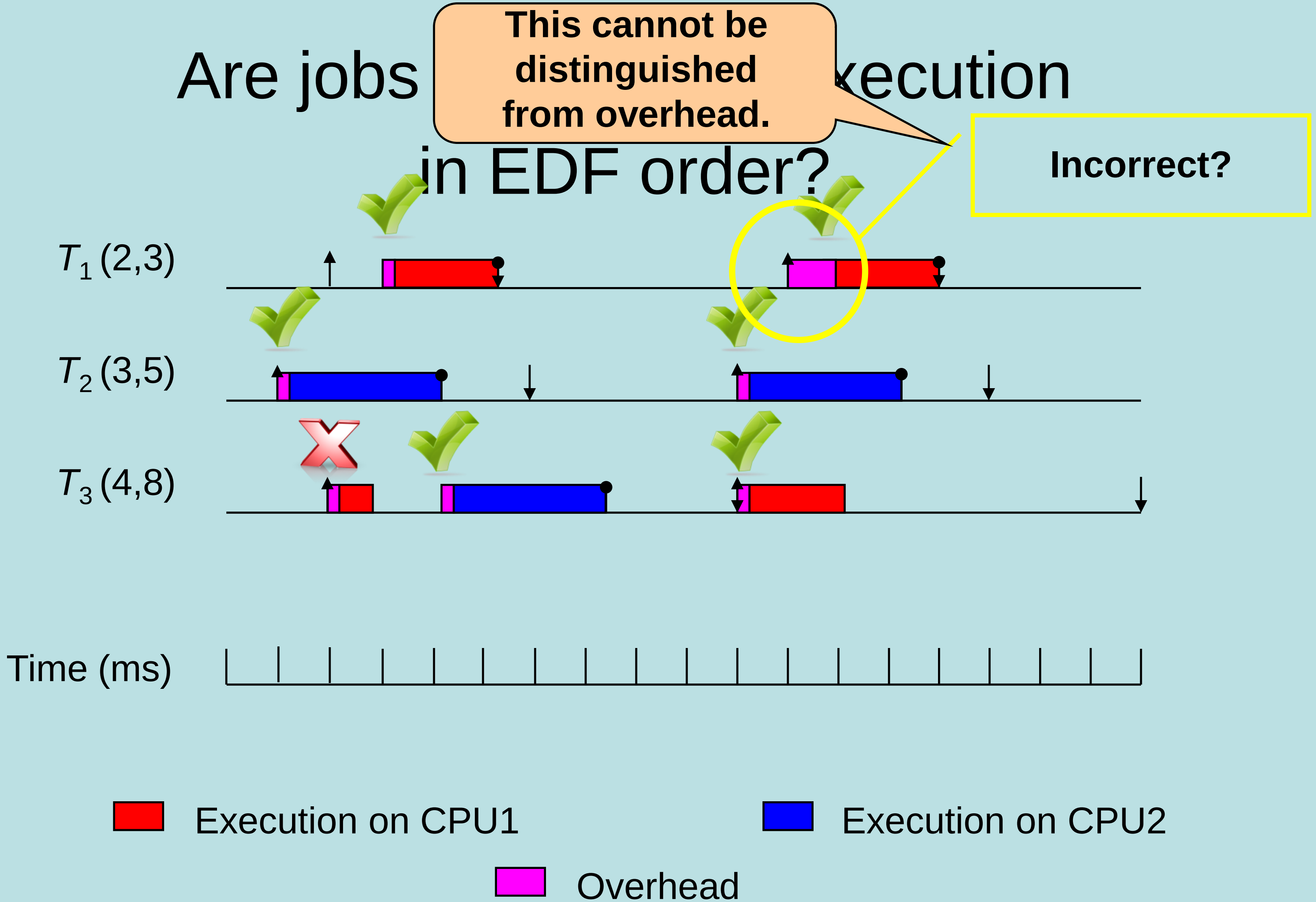
G-EDF Decision Test

Are jobs switched to execution in EDF order?



G-EDF Decision Test

Are jobs [...]xecution in EDF order?



Unit Tests

Unit Tests

Deadline Test

Sporadic Task Model Test

Completion Test

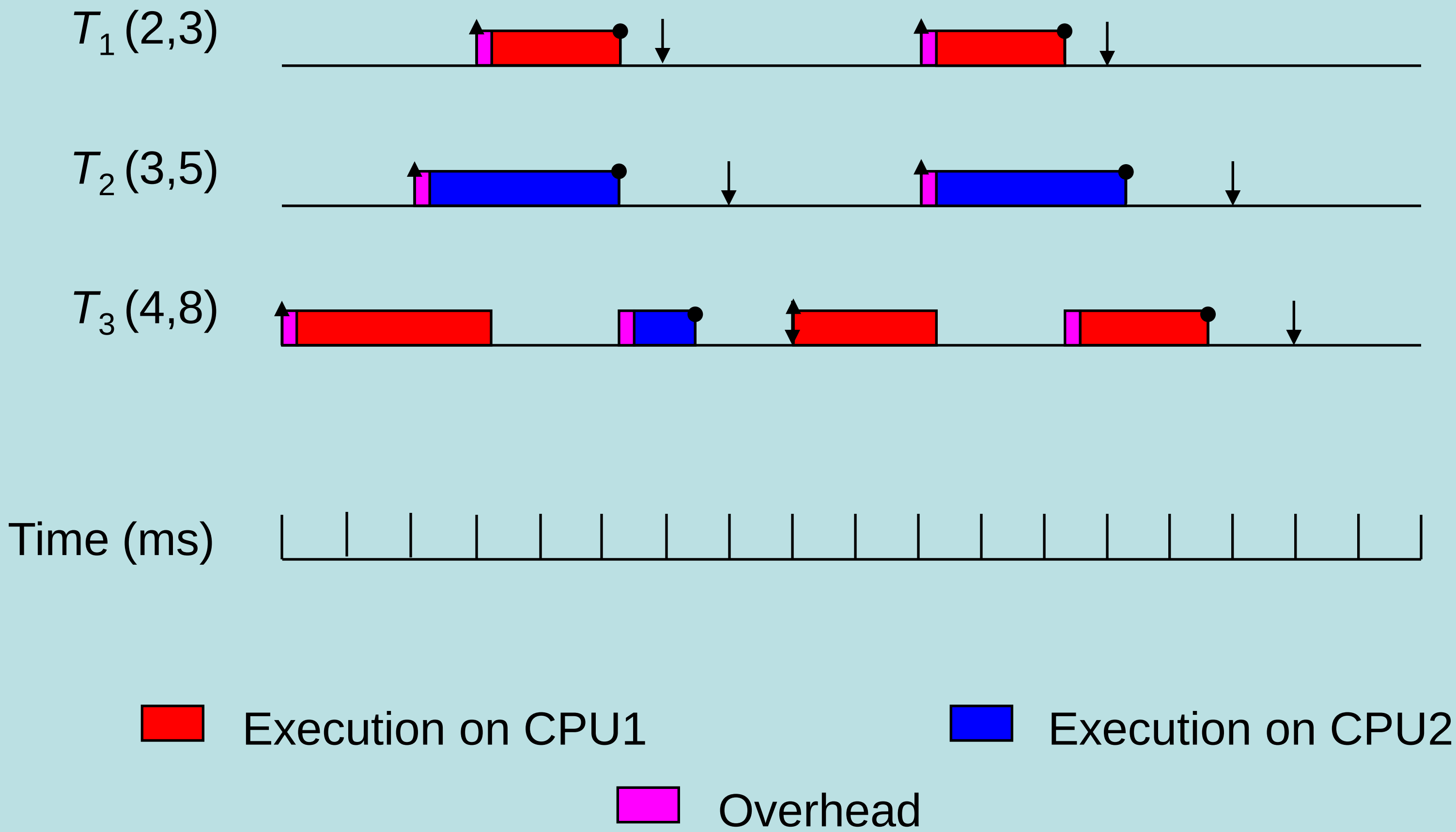
G-EDF Decision Test

G-EDF Latency Test

Helps ensure acceptable amounts of overhead

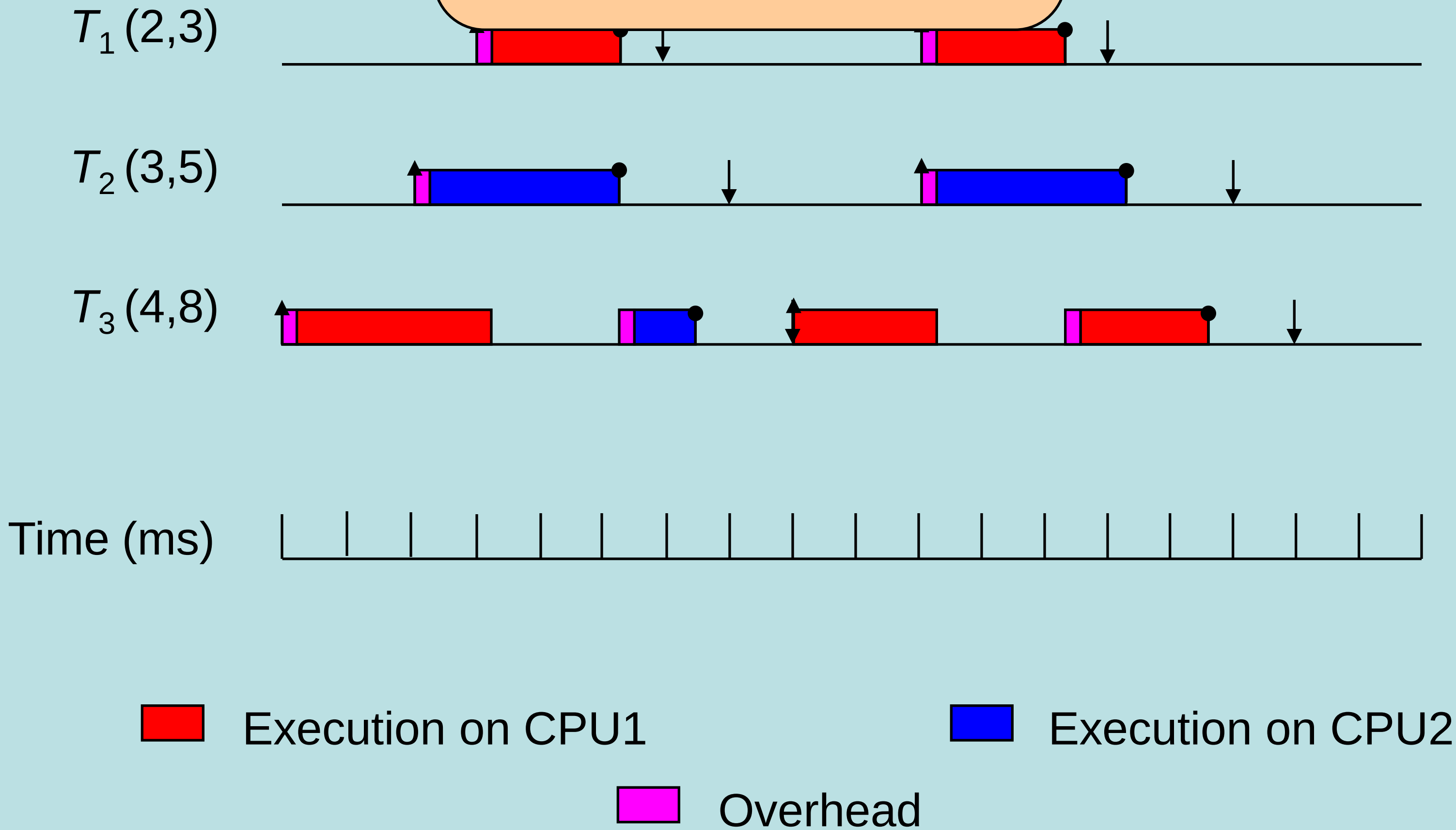
G-EDF Latency Test

Measures latency

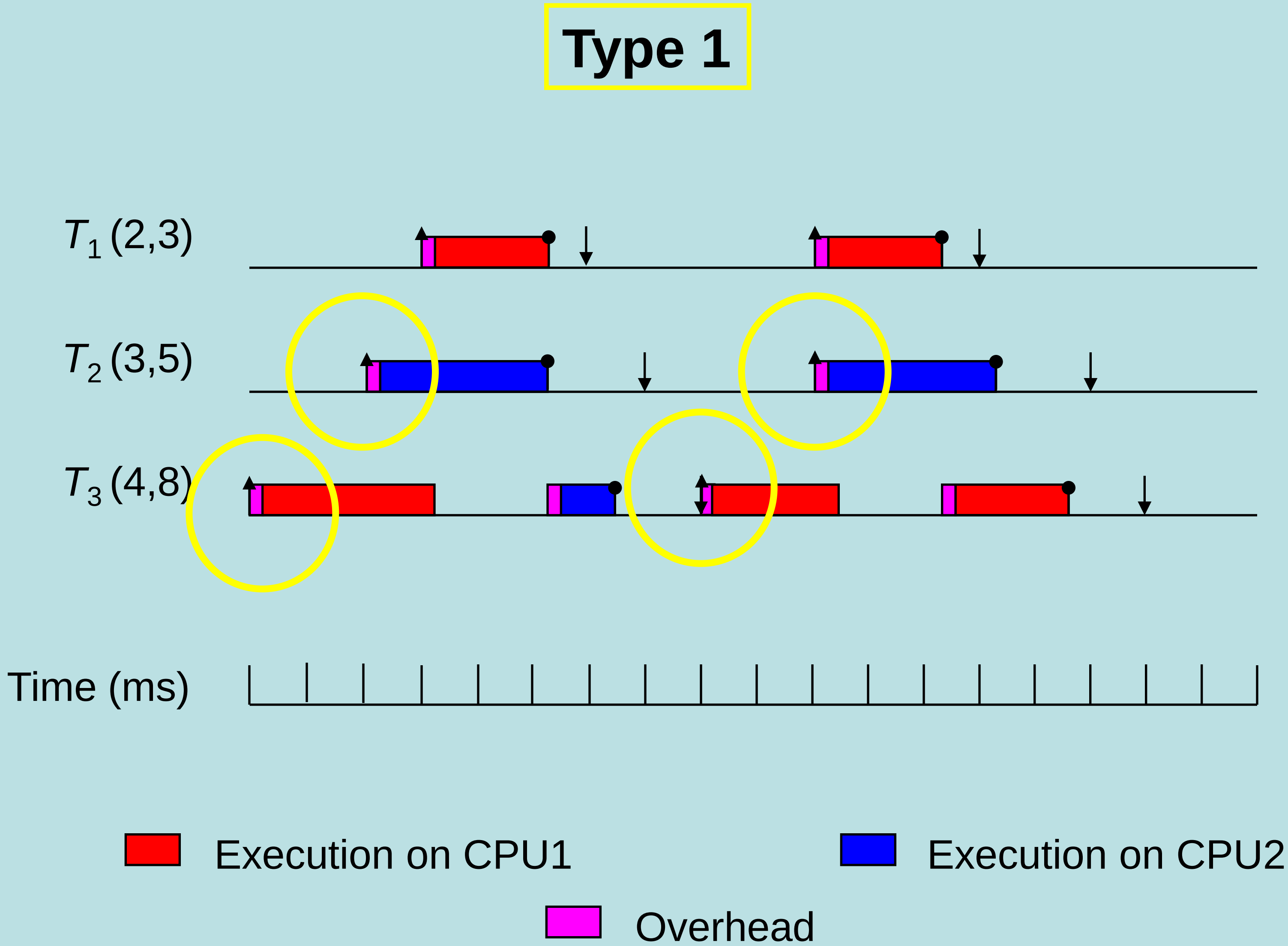


G-EDF Latency Test

Test algorithm can use sequence of events on each processor to determine type of latency

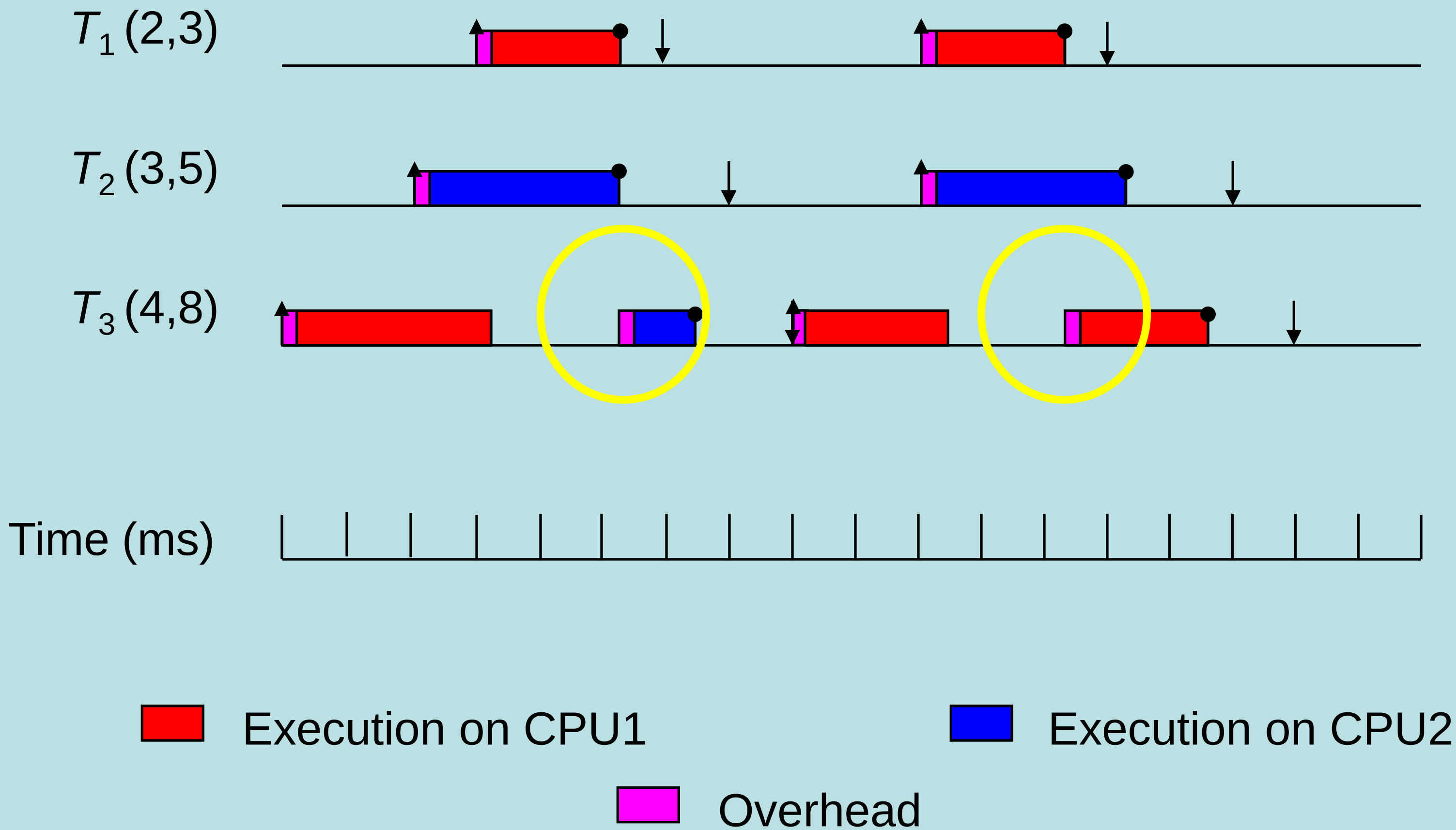


G-EDF Latency Test



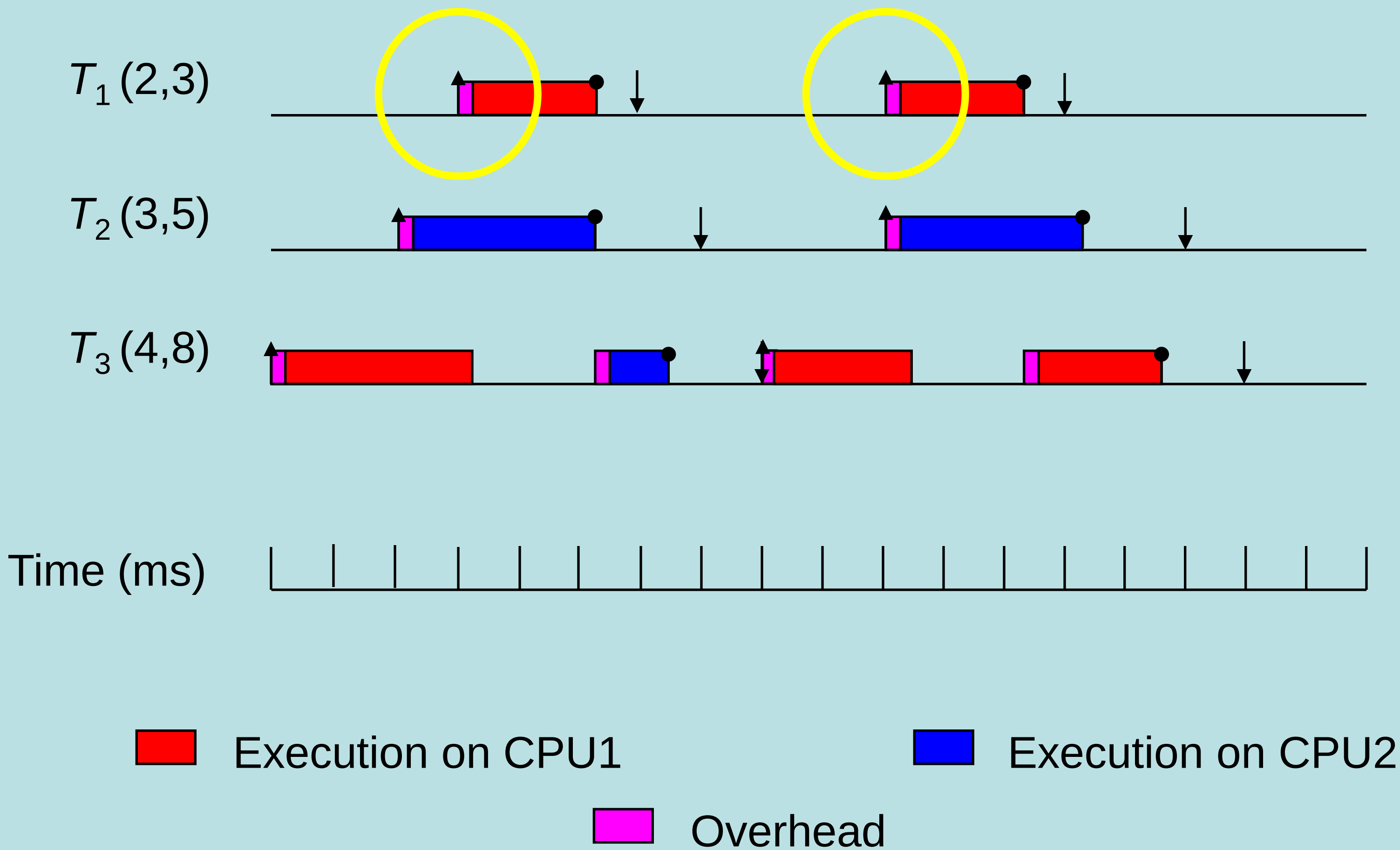
G-EDF Latency Test

Type 2



G-EDF Latency Test

Type 3



Summary

- What LITMUS^{RT} is
- Why we want to test LITMUS^{RT} schedulers
  - *Implementing real-time schedulers is nontrivial – bugs can be subtle*
- How to test LITMUS^{RT} schedulers
  - *Unit Testing - testing small pieces of code programmatically – with a twist*

Towards Unit Testing Real-Time Schedulers in LITMUS^{RT}

Questions?