Une école de l'IMT

Shedding the Shackles of Time-Division Multiplexing

Farouk Hebbache[1]
with Florian Brandner,[2] Mathieu Jan,[1] Laurent Pautet[2]

[1]CEA List, L3S
[2]LTCI, Télécom ParisTech, Université Paris-Saclay

Multi-core Architectures

- Complex WCET[1] analysis:
  - Pessimistic WCET
  - Tasks rarely use their full time budget
  - Low resource utilization
  - How to improve the use of hardware resources?

- "Mixed-Criticality" scheduling:
  - Timing guarantees depends on criticality level
  - Non-critical tasks with best effort processing
  - Improves resource utilization, but only at scheduling level

[1] Worst-Case Execution Time

Motivation

Memory arbitration scheme in multi-core architectures

Time-Division Multiplexing (TDM)

- Fixed time windows
- Exclusive memory access
- Isolation between cores
- More precises analysis techniques compared to Round-Robin[2]
- Non-work-conserving

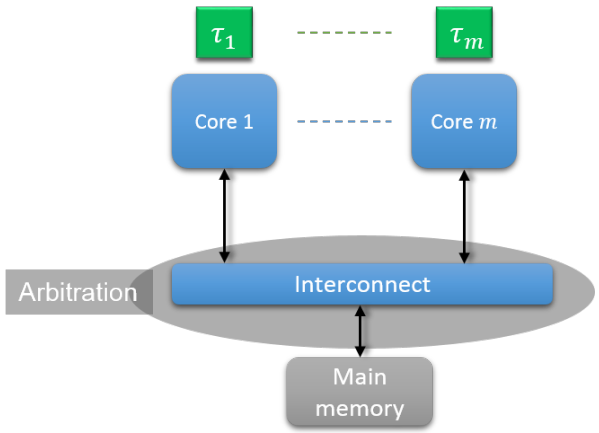
Goal: improve memory utilization while keeping TDM guarantees.

[2] H. Rihani et al, WCET Analysis in Shared Resources Real-Time Systems with TDMA Buses, RTNS 2015

System Model

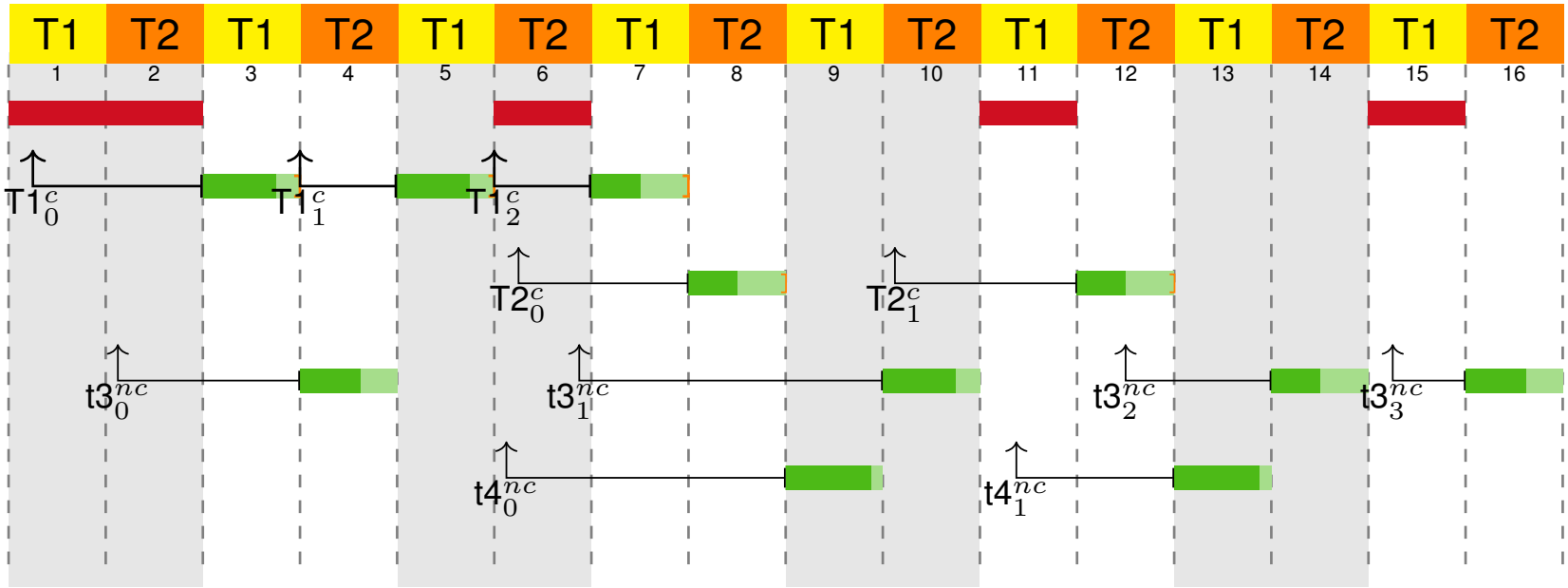
- *m* processing cores
- Each core executes a single task (critical or non-critical)

  $\Longrightarrow$ No tasks scheduling needed
  $\Longrightarrow$ Cores emit critical/non-critical memory requests



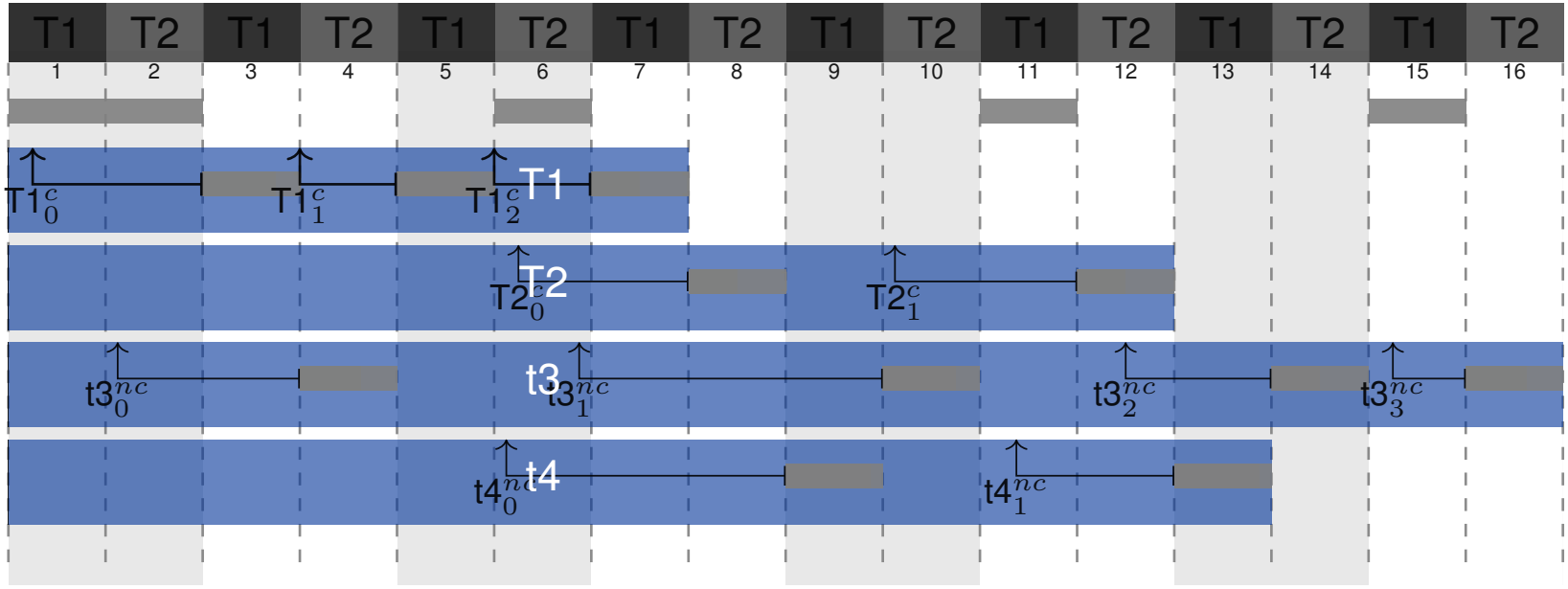
- Memory arbitration
  - Time-Division Multiplexing (TDM)
  - Dedicated slot for critical cores
  - No slots for non-critical cores

Example: Strict Time-Division Multiplexing



Critical tasks T1 and T2 with dedicated TDM slots
as well as non-critical tasks t3 and t4.

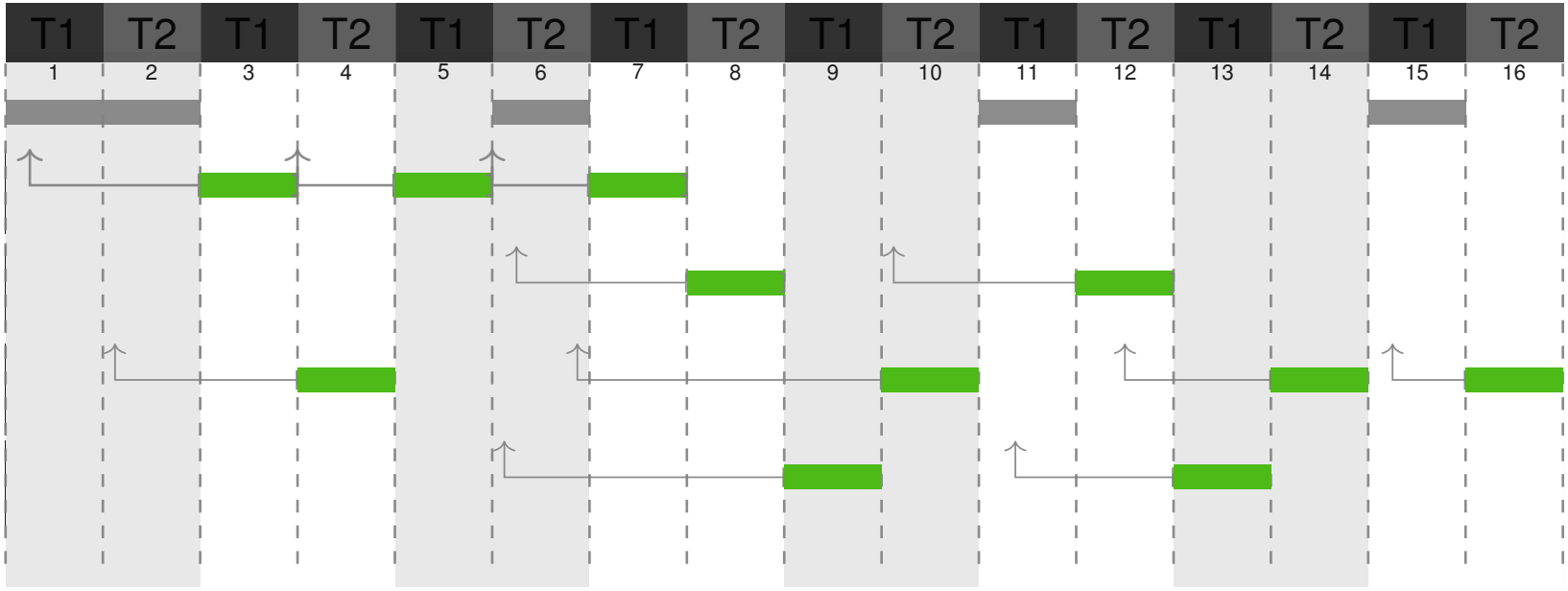
Example: Strict Time-Division Multiplexing



Critical tasks T1 and T2 with dedicated TDM slots
as well as non-critical tasks t3 and t4.

Rows: different tasks as they execute

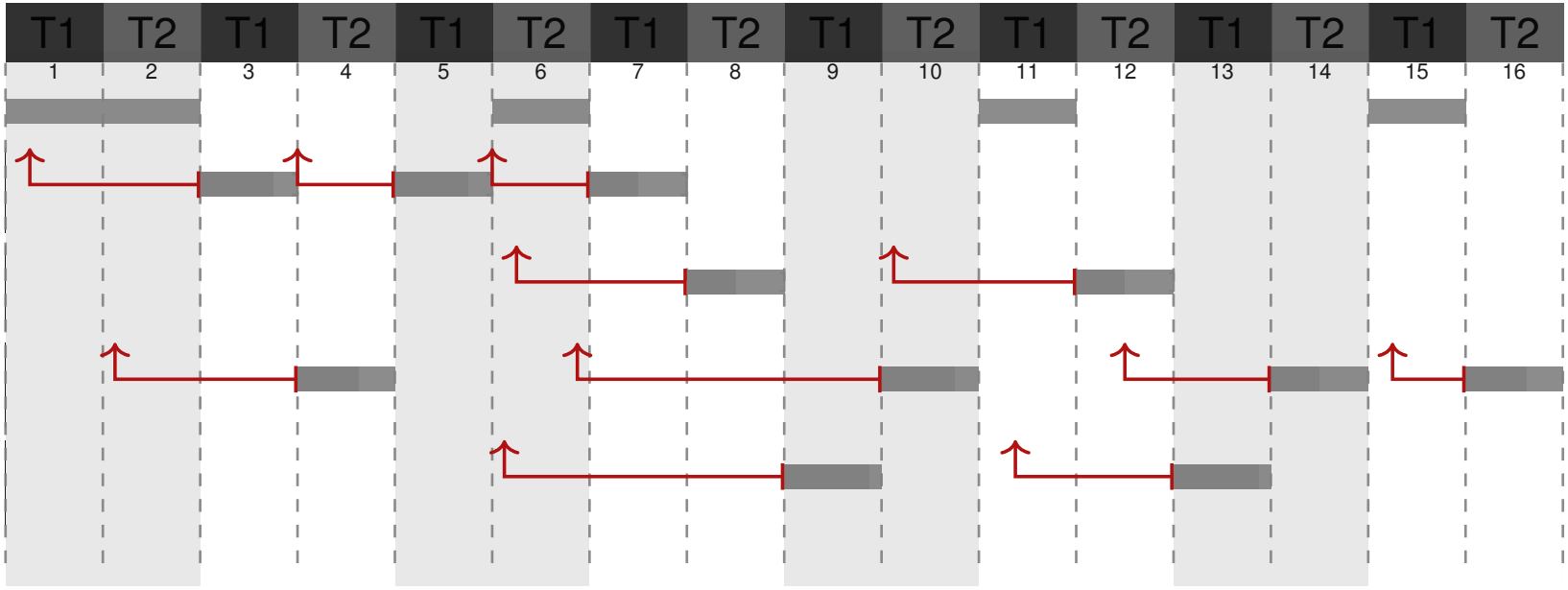
Example: Strict Time-Division Multiplexing



Critical tasks T1 and T2 with dedicated TDM slots
as well as non-critical tasks t3 and t4.

Green bars: requests being processed by the memory
(only a single active request at a time)

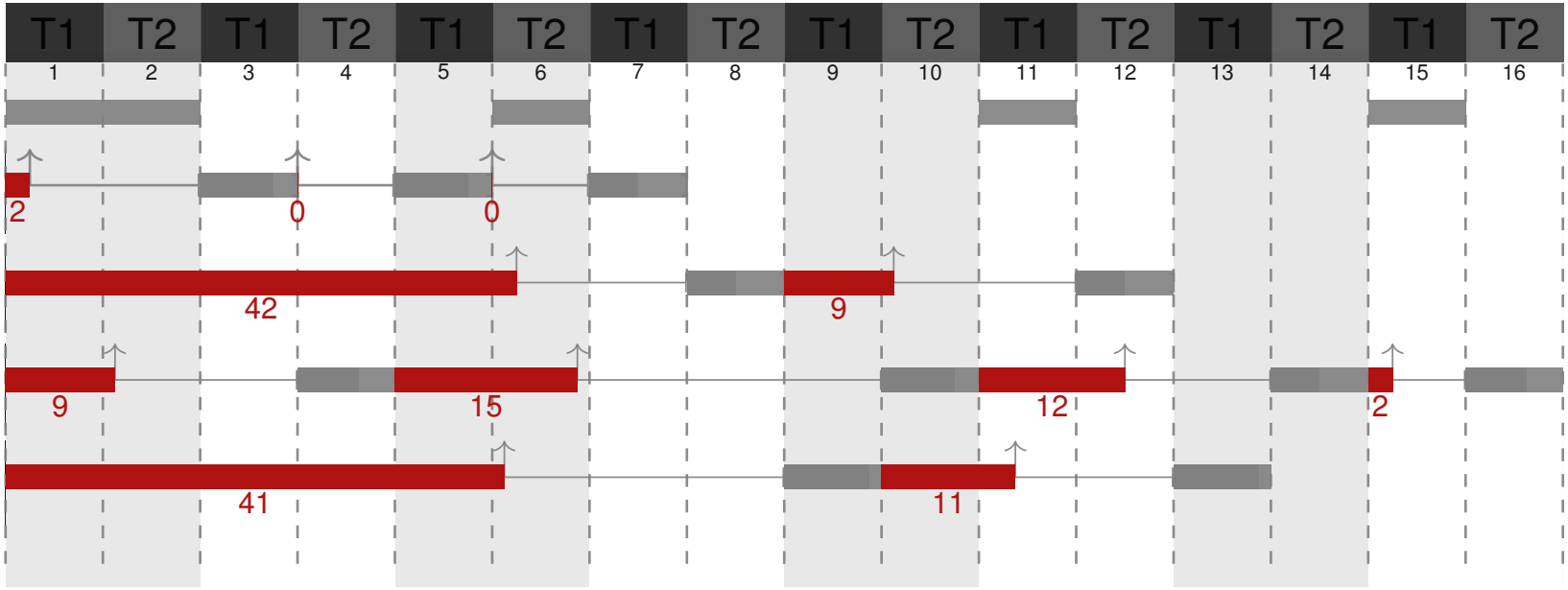
Example: Strict Time-Division Multiplexing



Critical tasks T1 and T2 with dedicated TDM slots
as well as non-critical tasks t3 and t4.

Arcs: memory wait time, from issuing to start of processing (depends on arbitration)

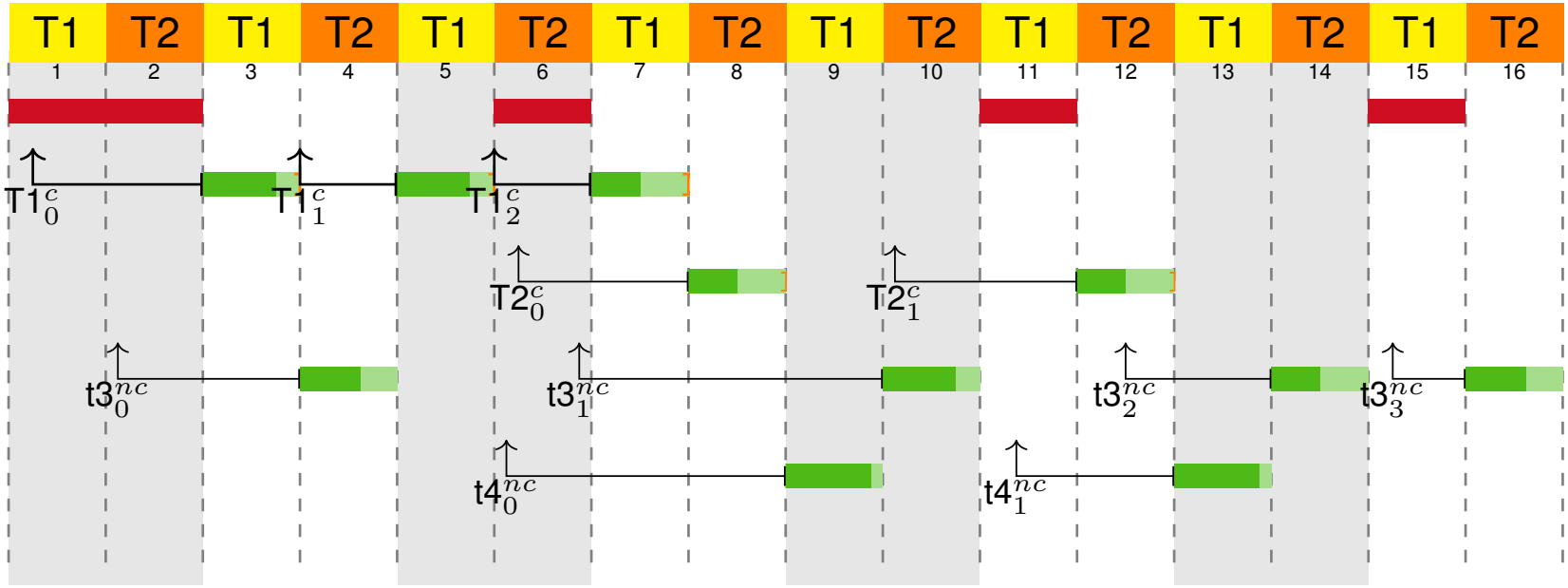
Example: Strict Time-Division Multiplexing



Critical tasks T1 and T2 with dedicated TDM slots
as well as non-critical tasks t3 and t4.

Gaps: computation time between requests
(these values remain unchanged)

Example: Strict Time-Division Multiplexing

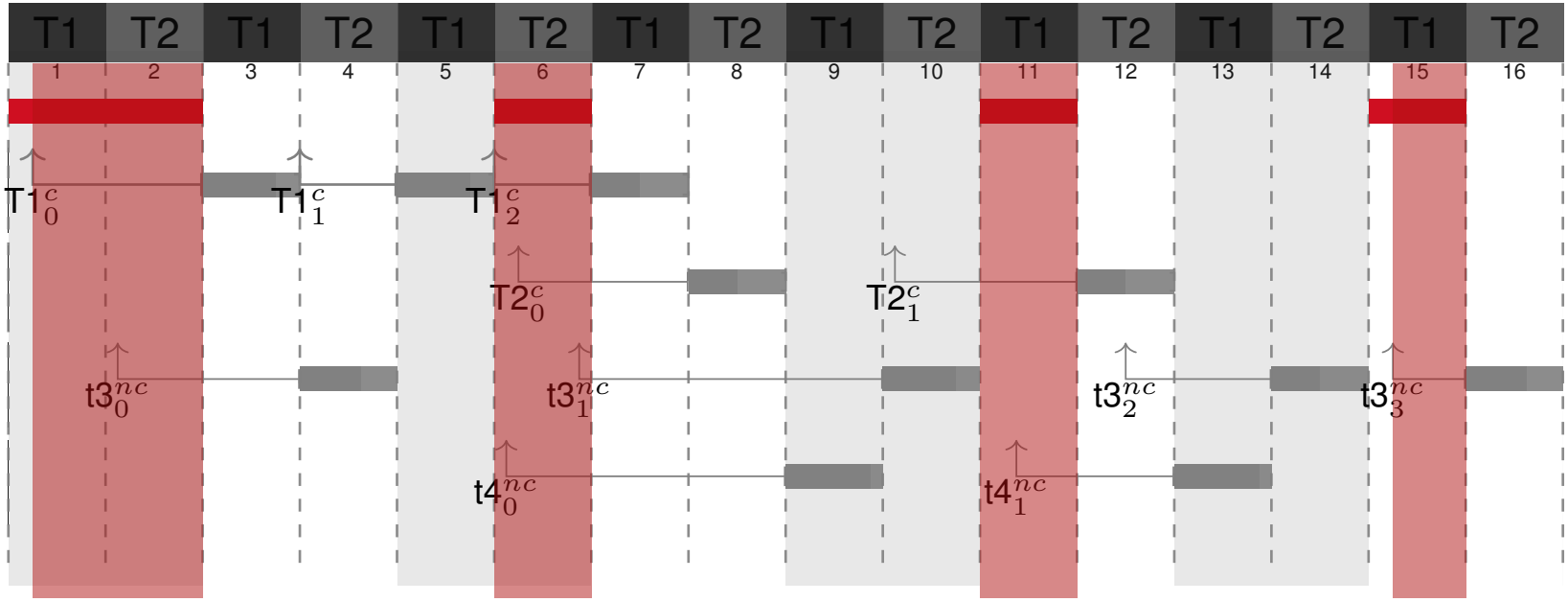


Critical tasks T1 and T2 with dedicated TDM slots
as well as non-critical tasks t3 and t4.

Non-critical requests can only reclaim unused TDM slots.

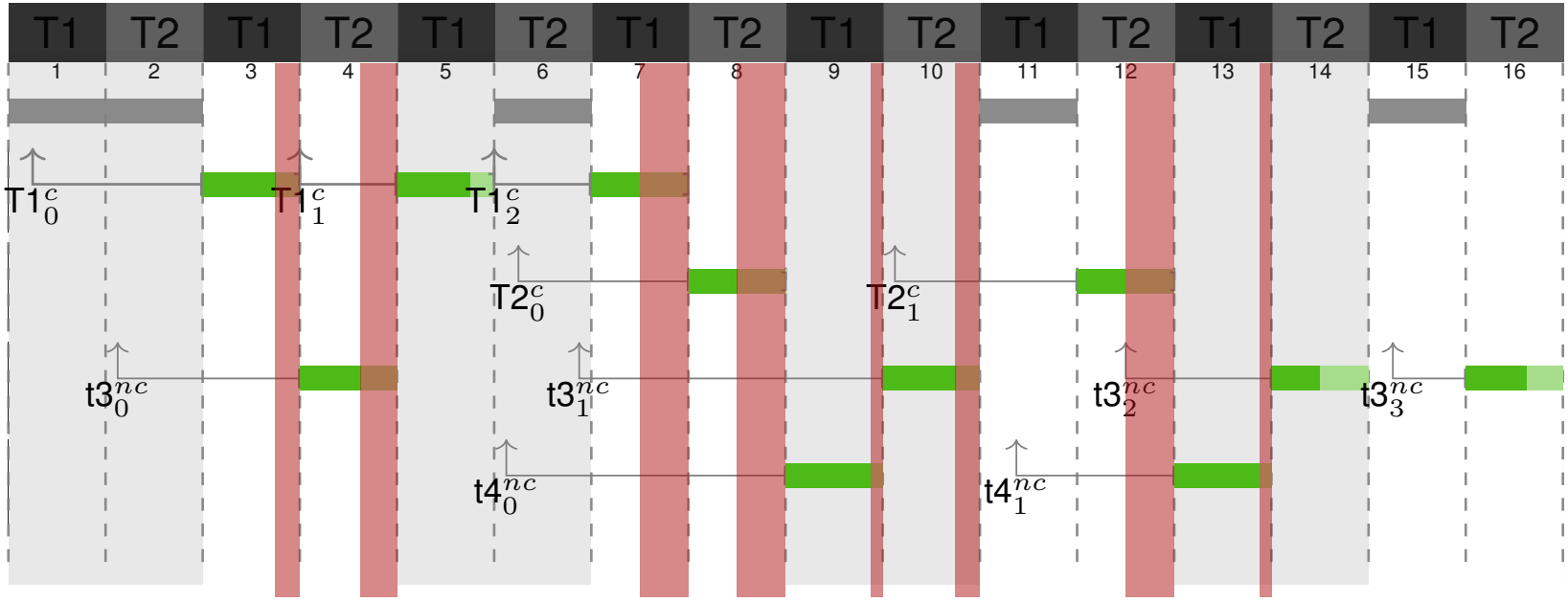
Issues with TDM

Issue with TDM (1)



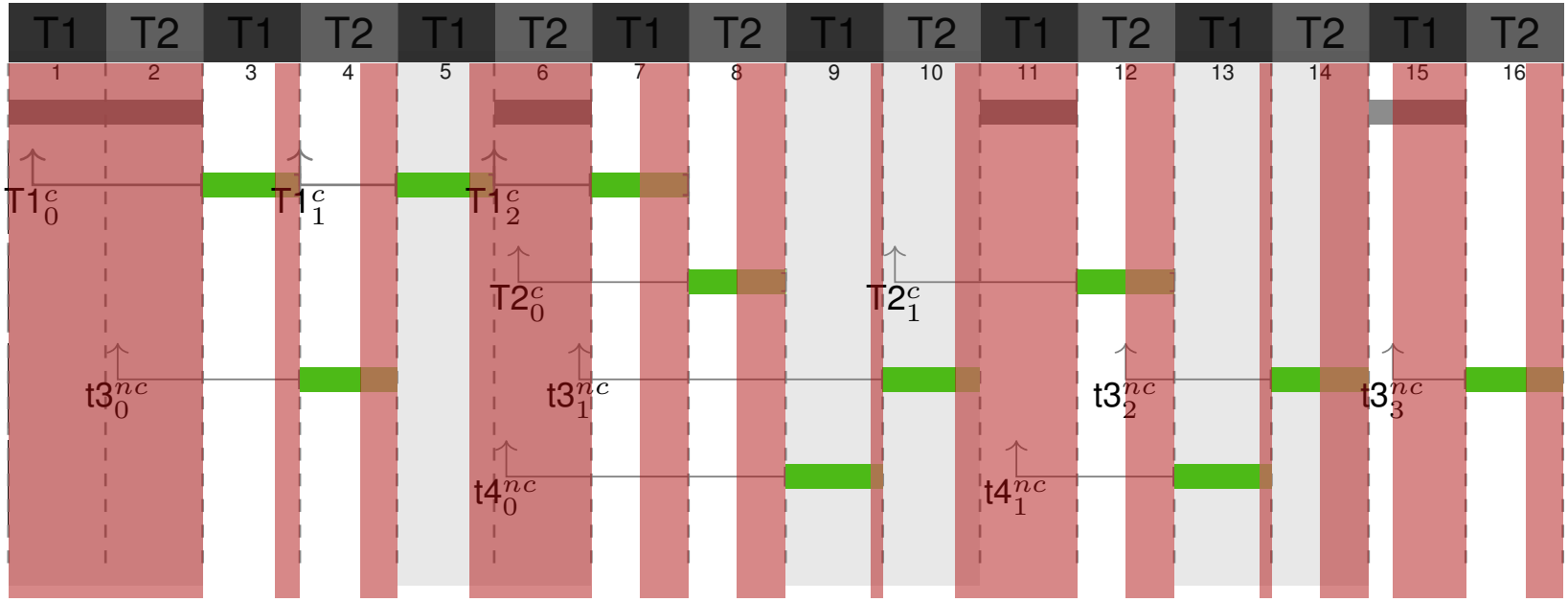
Issue Delay: Number of cycles during which the memory is idle, despite pending requests at the arbiter.

Issue with TDM (2)



Release Delay: Number of cycles that memory requests finish earlier than the TDM slot length.

Total Memory Idling



Number of cycles where the memory is idle

Issue & release delays $\Longrightarrow$ non-work-conserving

Goal: Eliminate issue & release delays.

Dynamic TDM-based Arbitration

Dynamic TDM-based Arbitration (TDMds)

- **Goal:** Reallocating free slots to other pending requests
  - F. Hebbache et al, “Dynamic arbitration of memory requests with tdm-like guarantees,” (Workshop CRTS’17)

- **How?**
  - Each critical request is associated with a deadline
    - Deadline is at end of a TDM slot of the request owner
    - Scheduled using *Earliest-Deadline-First* strategy (EDF)
  - Best-effort for non-critical requests
    - Scheduled using *First-In First-Out* strategy (FIFO)
      (other alternatives possible, e.g., fixed priorities)
  - Track slack when critical requests complete before deadline

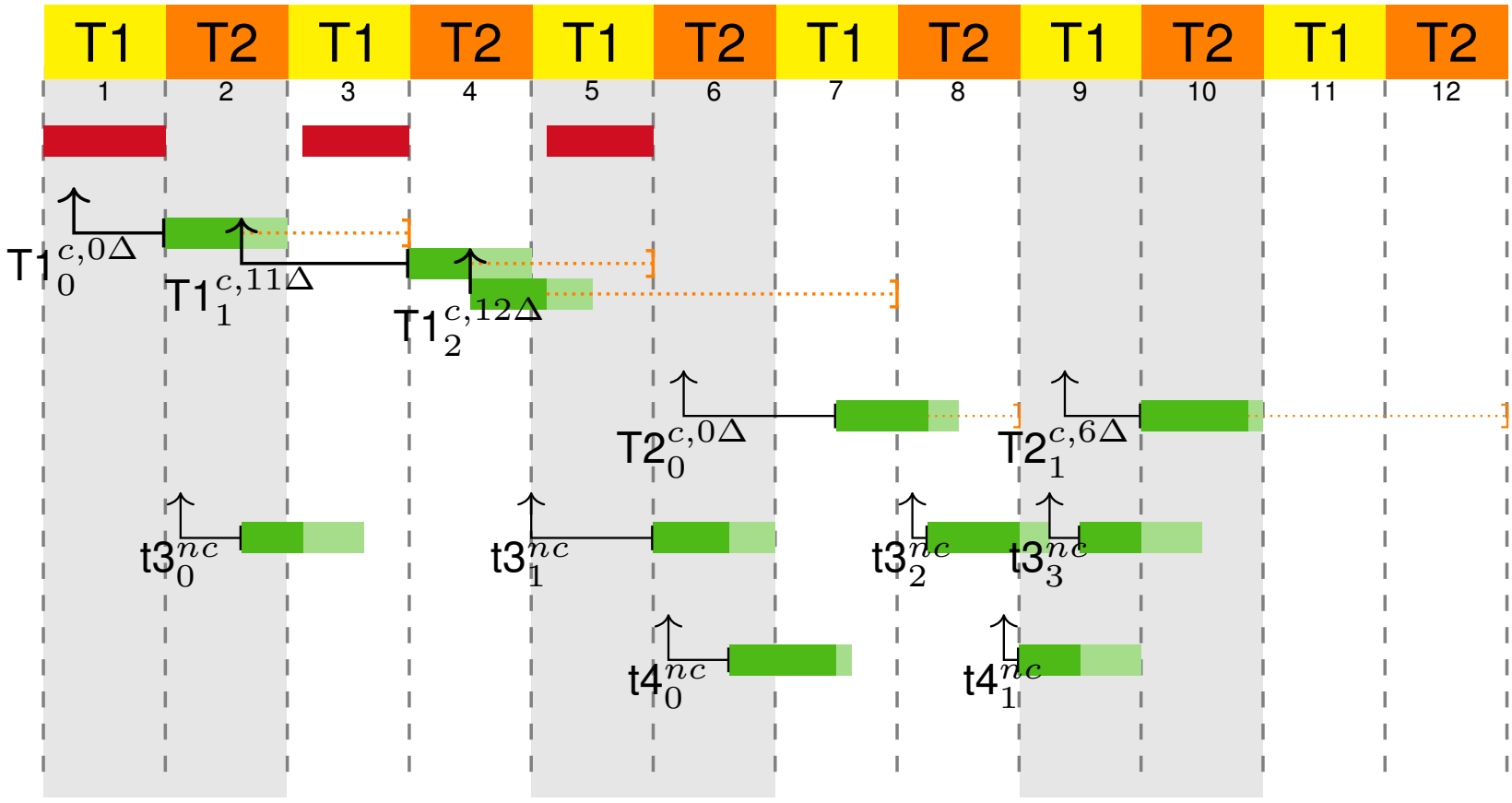
TDM-based Arbitration without slots (`TDMer`)

- **Goal:** Eliminate issue/release delays within slots . . .
- **How?**
  - Requests handled independently from TDM slots
  - Operates at the granularity of clock cycles
  - Request can complete within the next slot . . .

  $\Longrightarrow$ Owner of the next slot can miss its deadline!

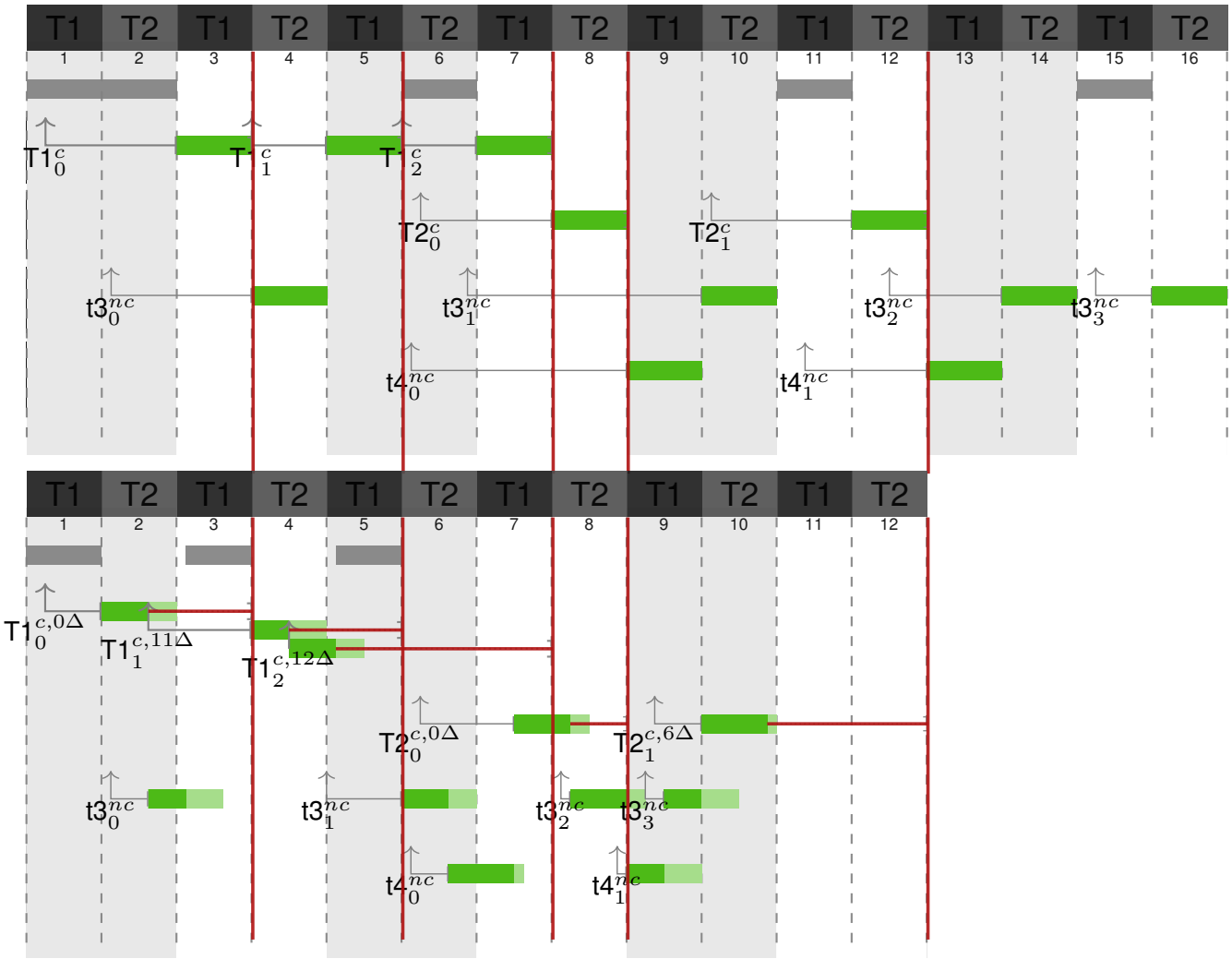
How to guarantee that critical tasks respects their deadlines?

Example: Dynamic TDM without slots (TDMer)



Same task set using dynamic TDM-based arbitration (TDMer).

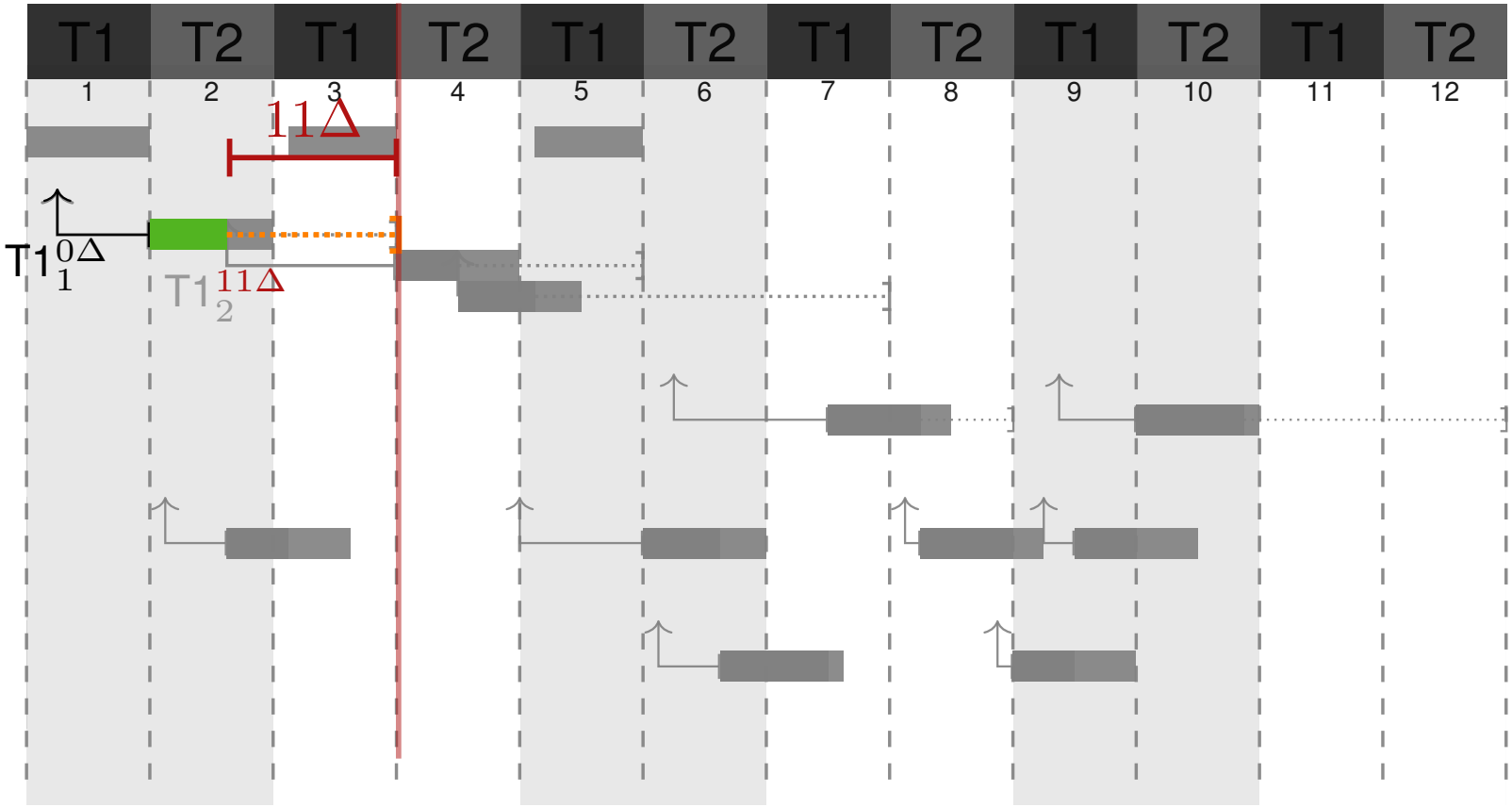
Strict TDM vs. TDMer



Critical requests complete earlier than under strict TDM.

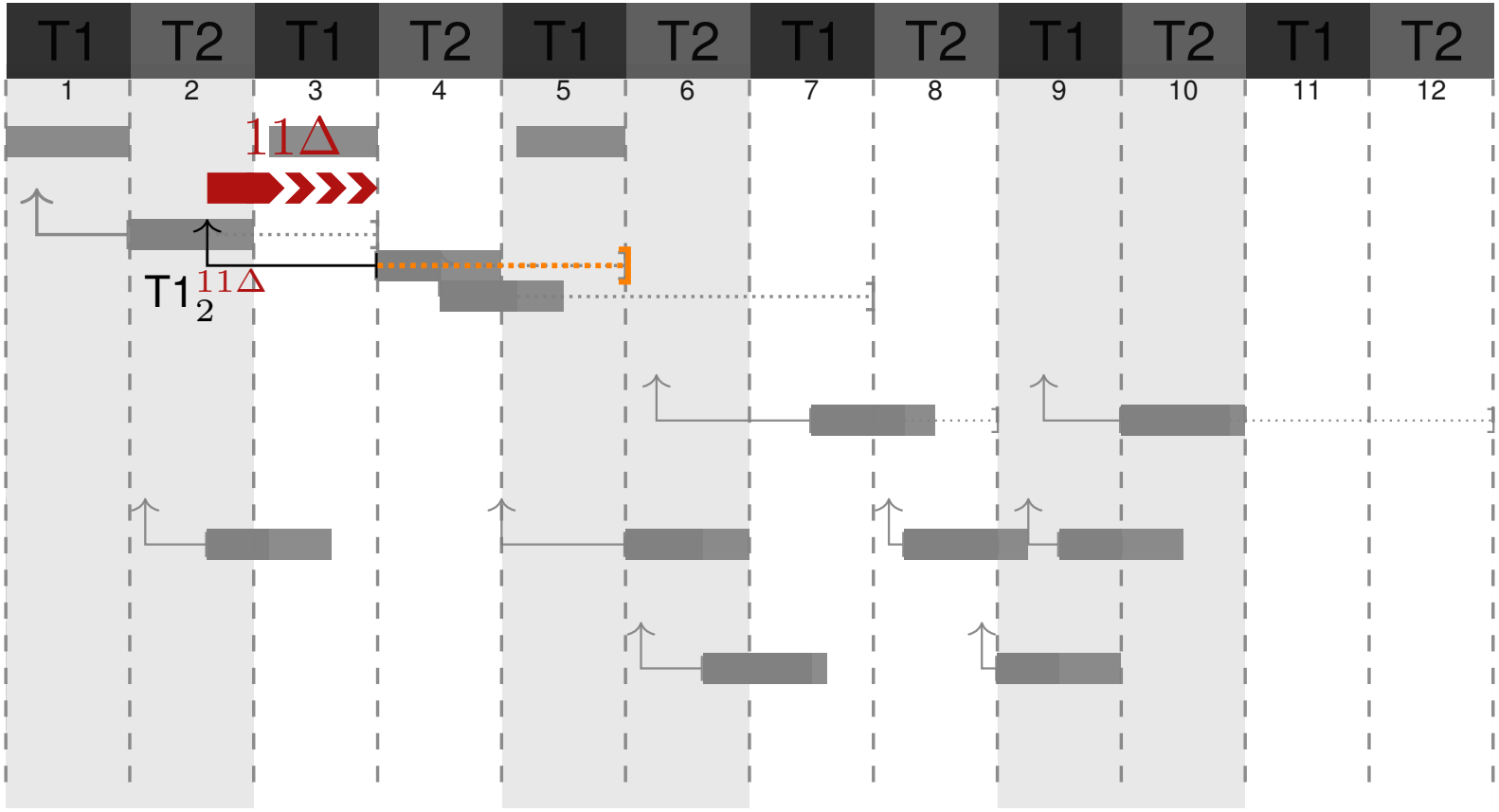
Making deadlines match

Slack Counters



Critical tasks accumulate slack whenever a request finishes before its deadline (i.e., earlier than strict TDM).

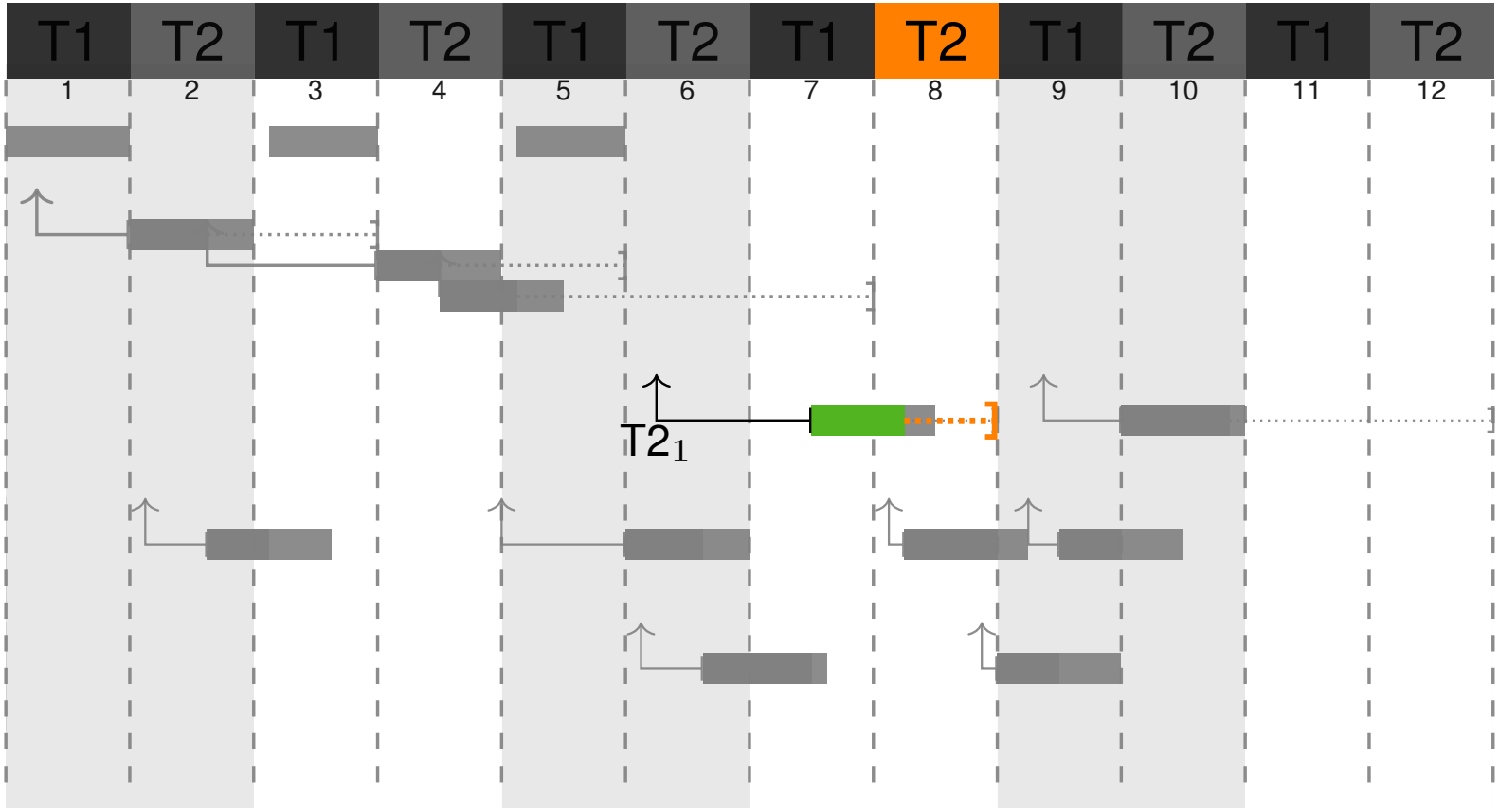
Deadlines computation



Deadlines are derived by finding the next TDM slot after the **delayed issue date** (*now* + slack).

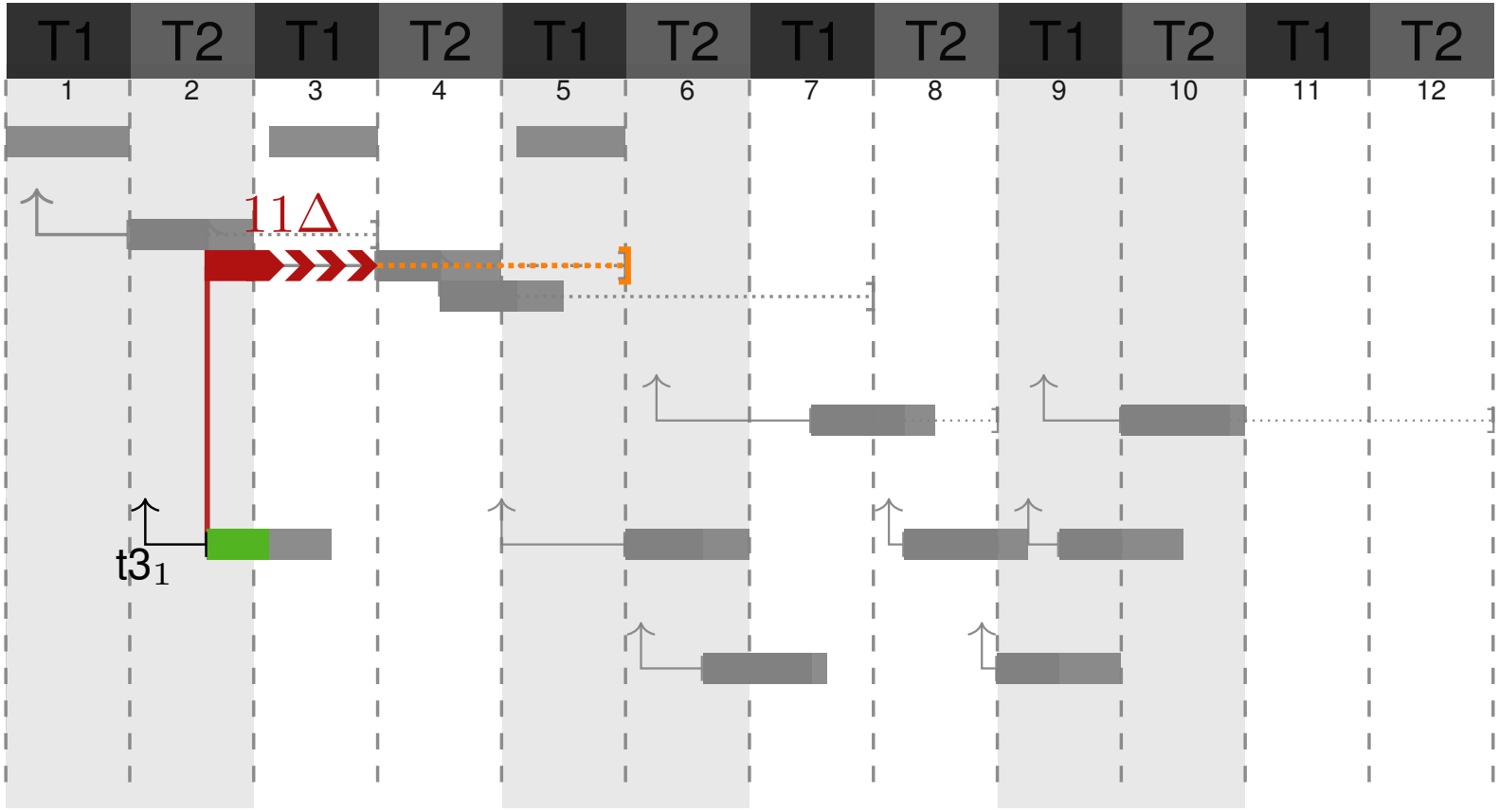
Scheduling requests at any moment

Early requests processing (1)



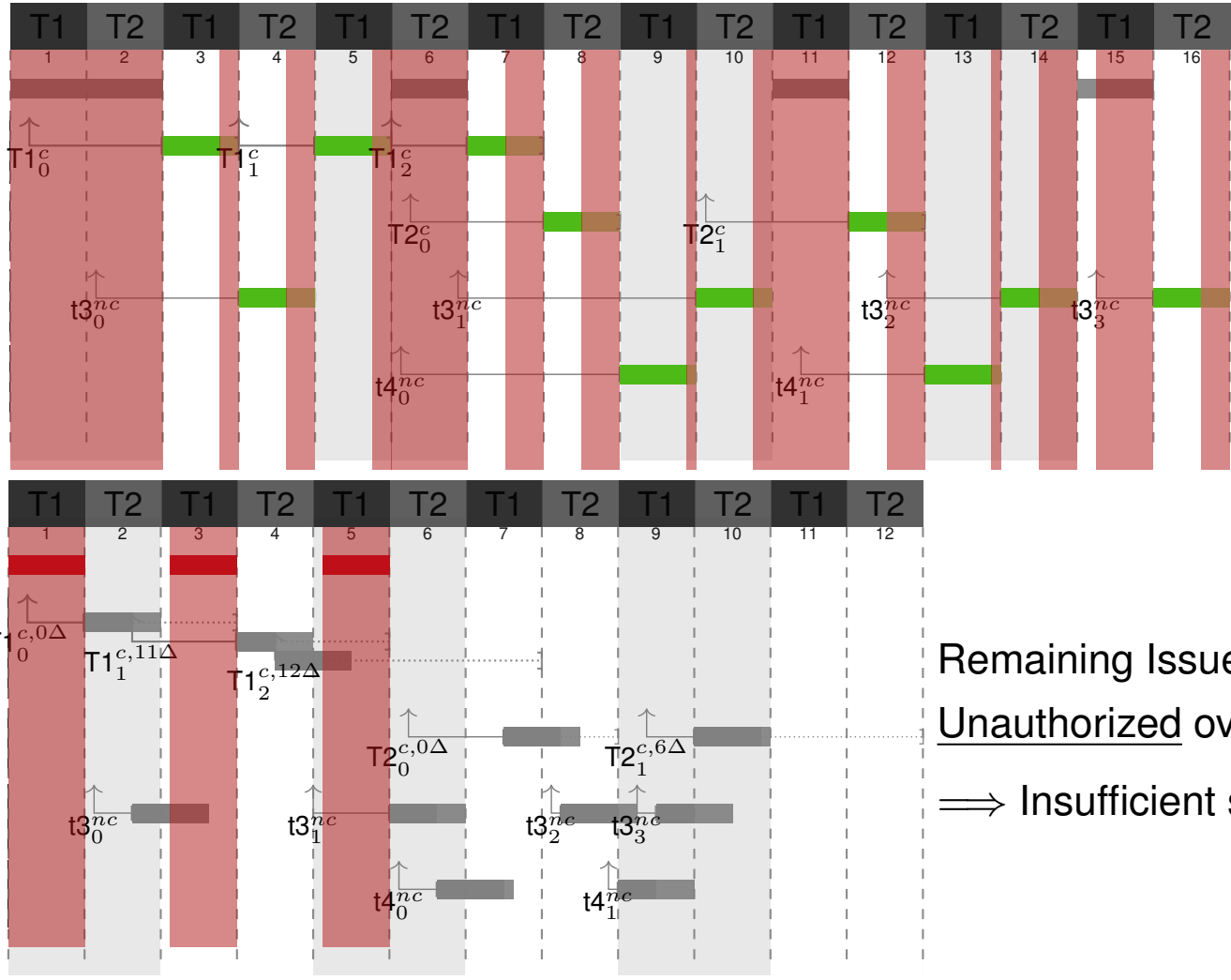
Critical requests may be processed when the upcoming slot belongs to the request owner.

Early request processing (2)



Requests are processed any time, independent from TDM slots, if critical tasks have enough slack (here T1).

Memory idling considerably reduced (`TDMer`)



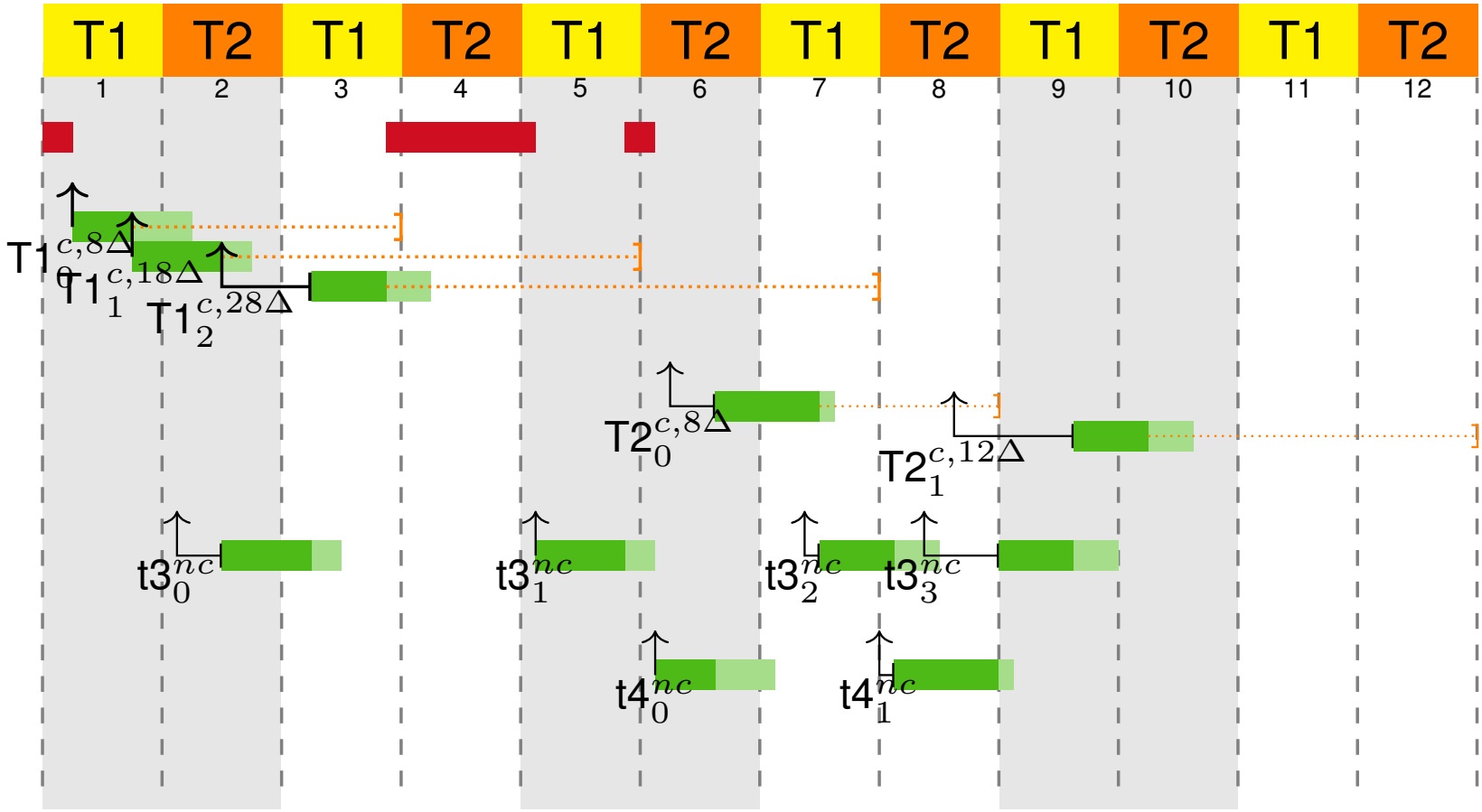
Remaining Issue delay . . .

Unauthorized overflow on slot **T2**

$\Longrightarrow$ Insufficient slack?

The total memory idling is considerably improved

Initial Slack (`TDMeri`)



Providing initial slack of a single TDM slot (8Δ) eliminates (almost) all issue delays …

Experiments

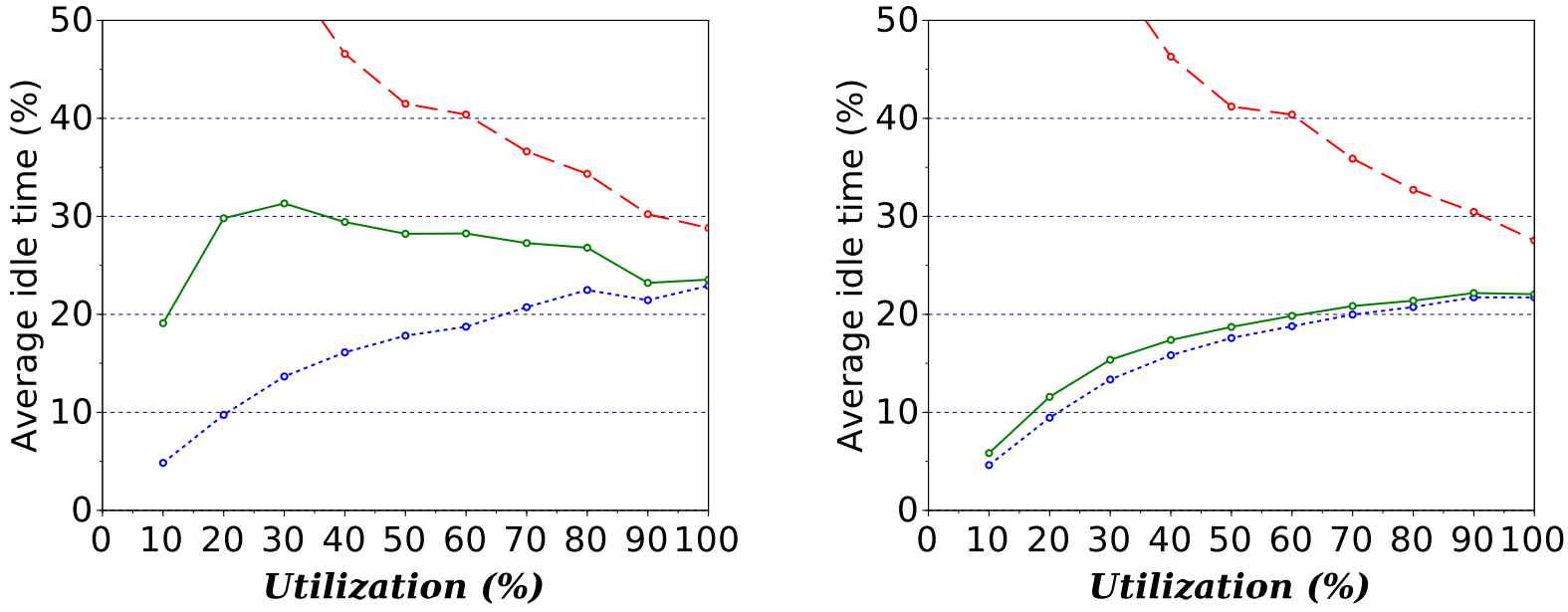
Benchmark Setup

- Up to 24 Patmos cores simulated
- TDM slot length set to 40 cycles
  - Access latencies between $[21, 40]$ cycles
- Various TDM-based arbitration schemes
  (`TDMfs`, `TDMds`, `TDMer`, and `TDMeri`)

- Overall 4320 simulation runs
  - Based on randomized memory traces
    (calibrated from actual traces of MiBench on Patmos)
  - Ratio of critical/non-critical tasks: 1/3 and 1/1

Evaluation metric: memory utilization
(issue & release delays, memory idling)

Results: reduced issue delay



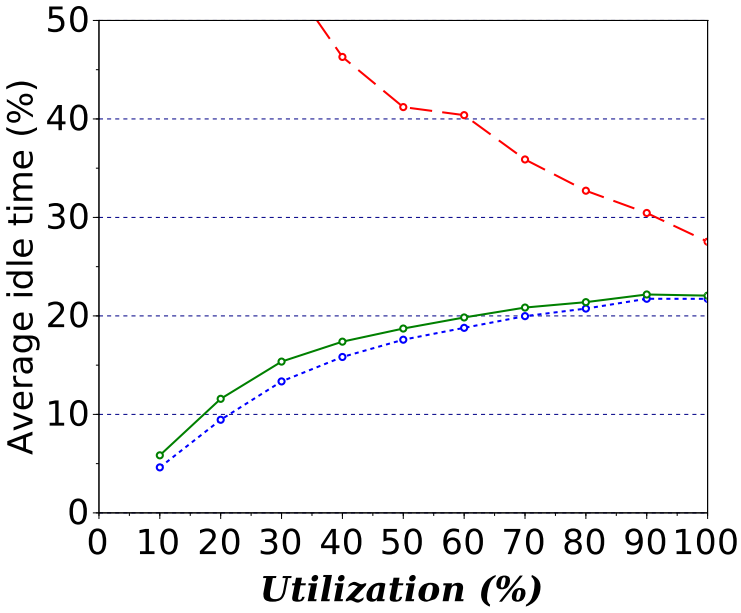
(a) Strict TDM (`TDMfs`)

(b) Dynamic TDM respecting slots (`TDMds`)

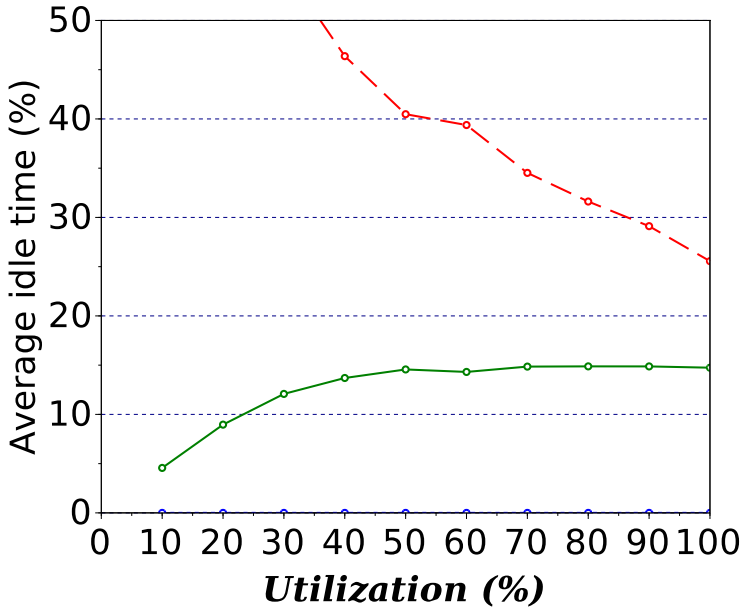
Dynamic TDM arbitration largely eliminates issue delays

However: Up to 23% of release delay . . .

Results: release delay vanishes



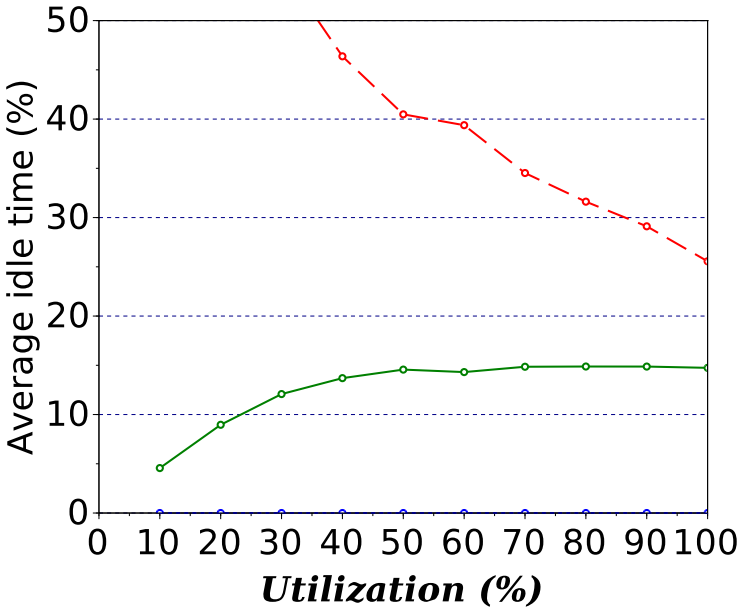
(b) Dynamic TDM respecting slots (`TDMds`)



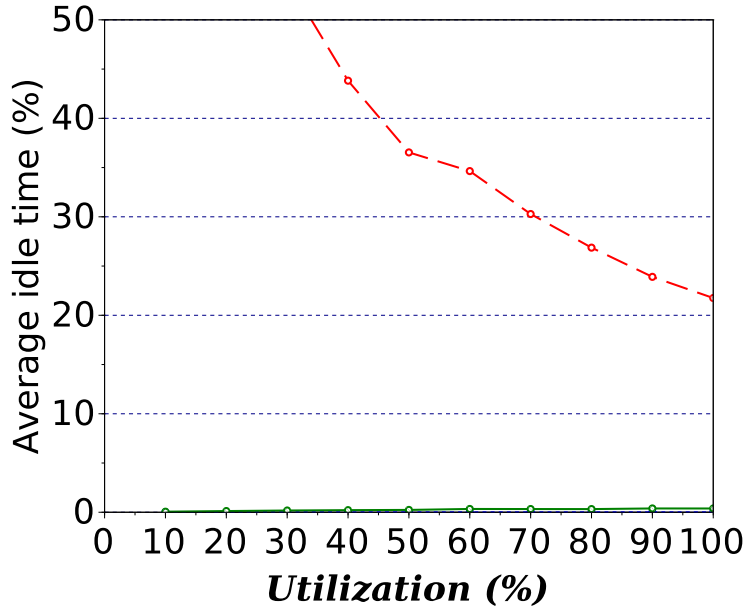
(a) Dynamic TDM without slots (`TDMer`)

`TDMer` arbitration noticeably better under high load – but little change in total memory idling …

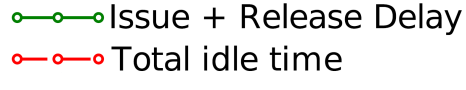
Results: work-conserving?



(a) Dynamic TDM without slots (`TDMer`)



(b) Dynamic TDM without slots with Initial Slack (`TDMeri`)



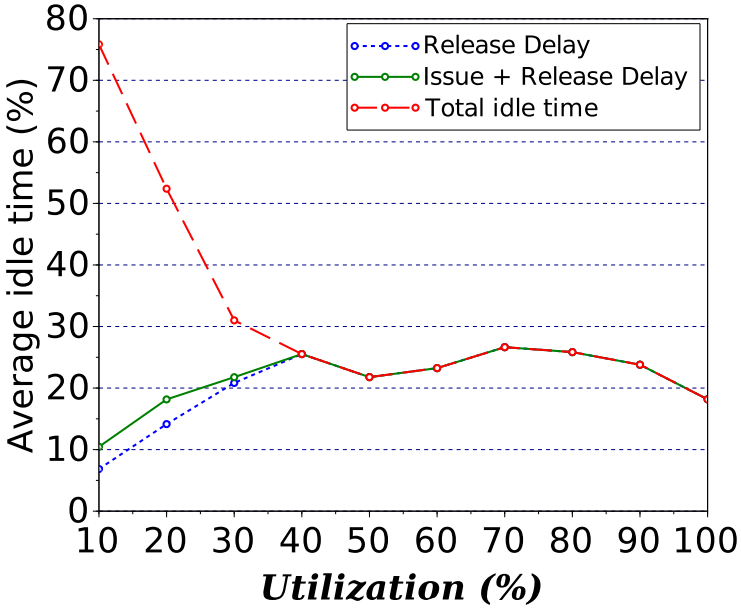
Initial slack (40 cycles) results in **work-conserving** arbitration — total memory idling improves considerably under high load

Conclusion

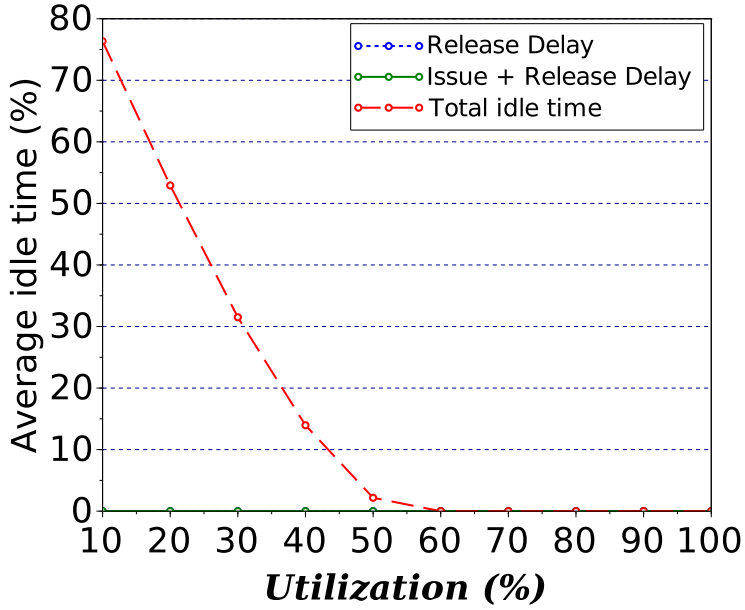
- Dynamic TDM-based arbitration
  - Improve resource utilization (work-conserving)
  - Preserves TDM's guarantees (proof in the paper)
  - Preserves precise analysis techniques

- Future work
  - Multiple tasks on a single core $\Longrightarrow$ preemption
  - Hardware implementation
  - Other forms of slack to be accumulated

TDMeri under high load



(a) Strict TDM (TDMfs)



(b) Dynamic TDM without slots With initial slack (40 cycles) (TDMeri)

- Memory idle time considering 24 tasks
- 6 critical and 18 non-critical tasks