INSTITUTE OF COMMUNICATION, INFORMATION AND PERCEPTION TECHNOLOGIES

Scuola Superiore Sant'Anna

Retis Real-Time Systems Laboratory

Resource Reservation for Real-Time Self-Suspending Tasks: Theory and Practice

Alessandro Biondi, **Alessio Balsini**, Mauro Marinoni

Scuola Superiore Sant'Anna, Pisa

Outline

- Scenario
  - Common Use Case
  - Self-Suspending Tasks on H-CBS
- Solution
  - Naïve
  - H-CBS-SO (Newly Proposed Algorithm)
- Linux Implementation
- Conclusions and Future Work

Scenario

We consider systems running:

- Time sensitive tasks
- Best effort tasks (user interface, log, …)

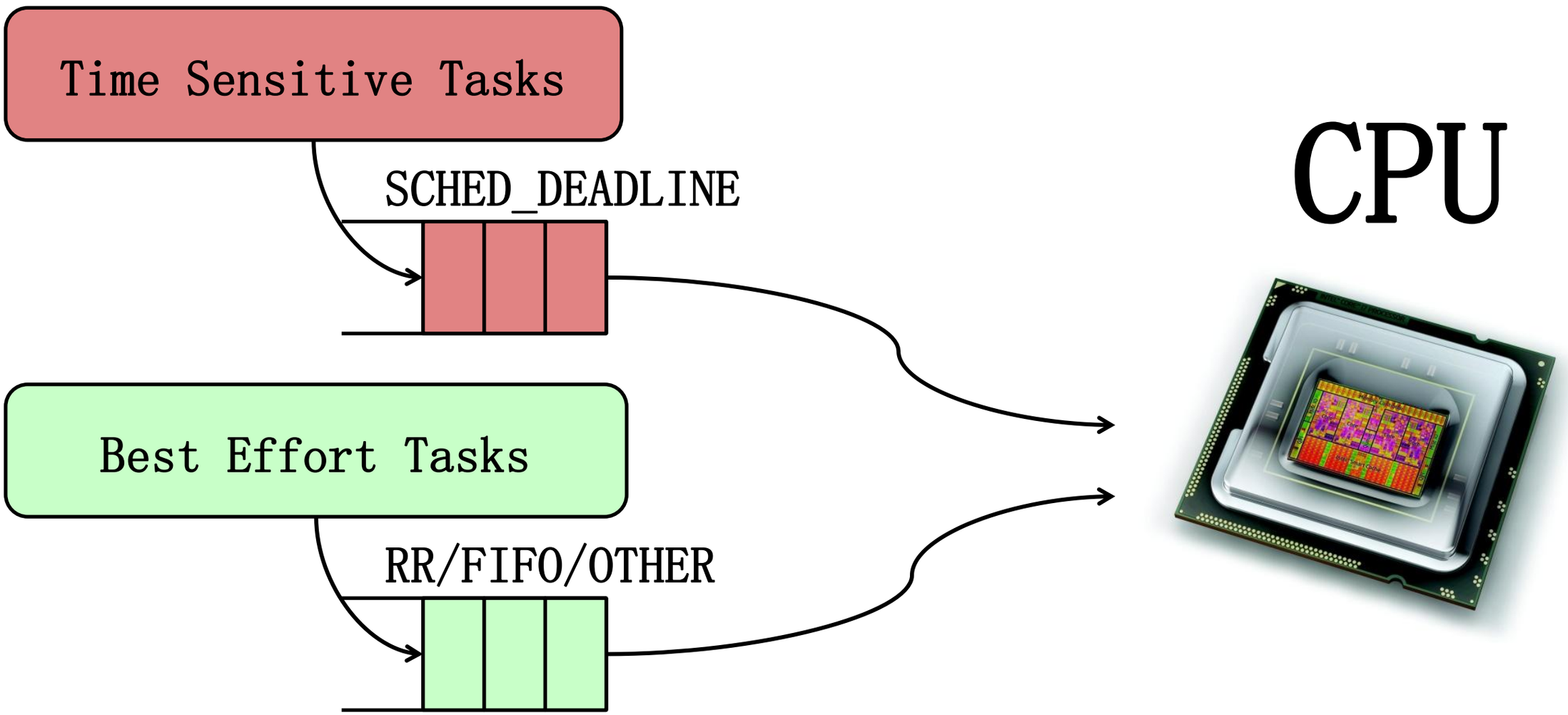
Objectives:

- Provide isolation among time sensitive tasks
- Real-time scheduling of time sensitive tasks

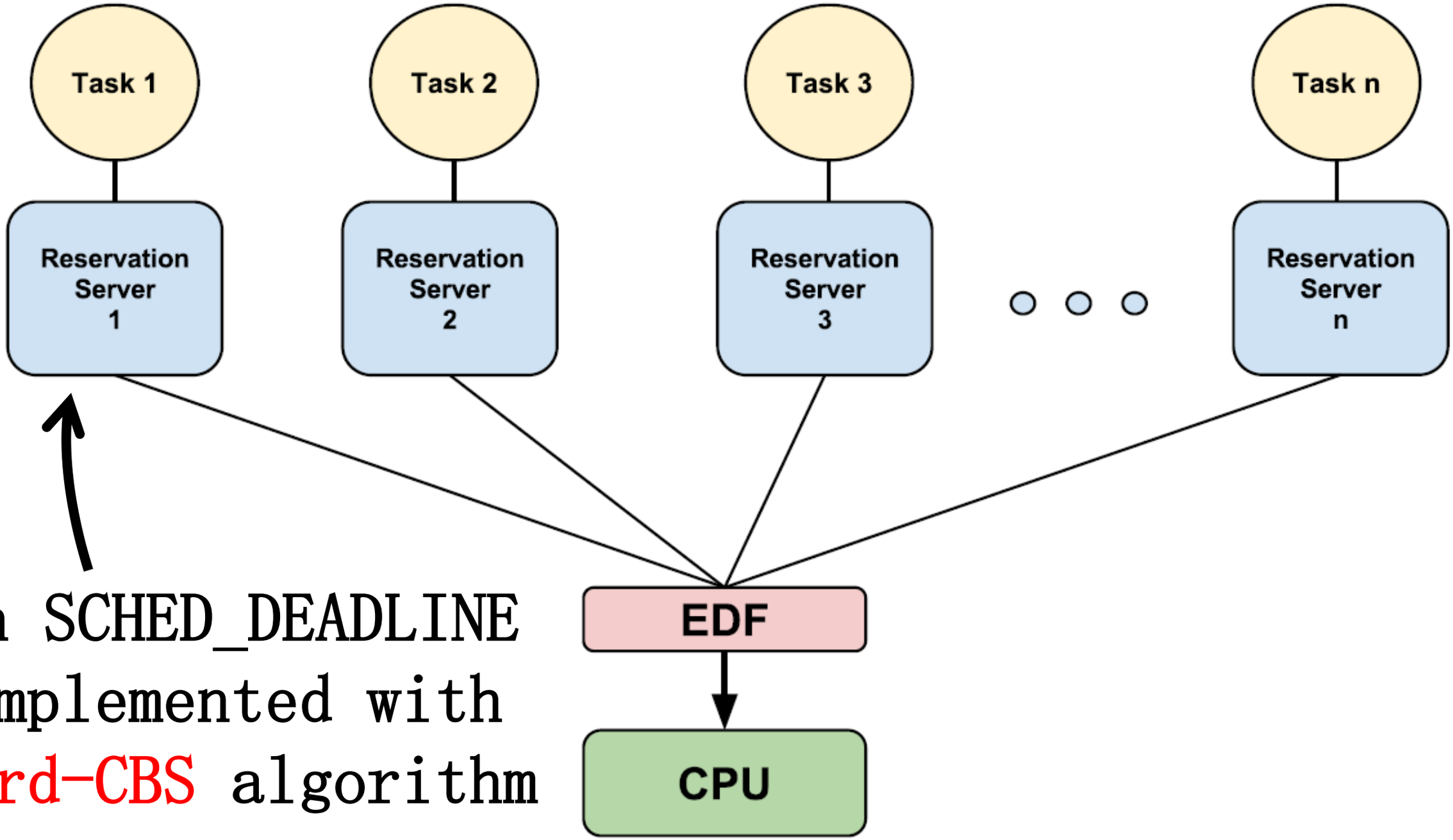
Diffused Example

Scheduling in Linux

SCHED_DEADLINE in mainline since v3.14



EDF + Resource Reservation



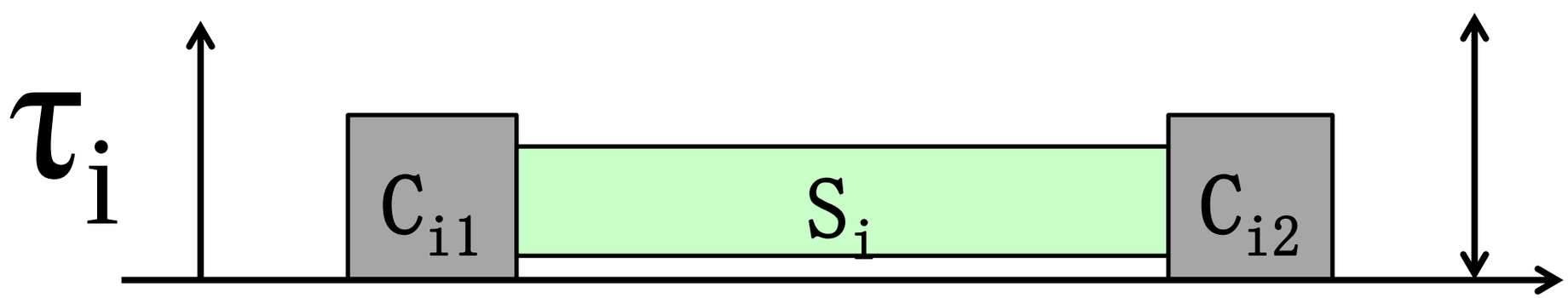
In SCHED_DEADLINE implemented with Hard-CBS algorithm

Self-Suspending Tasks

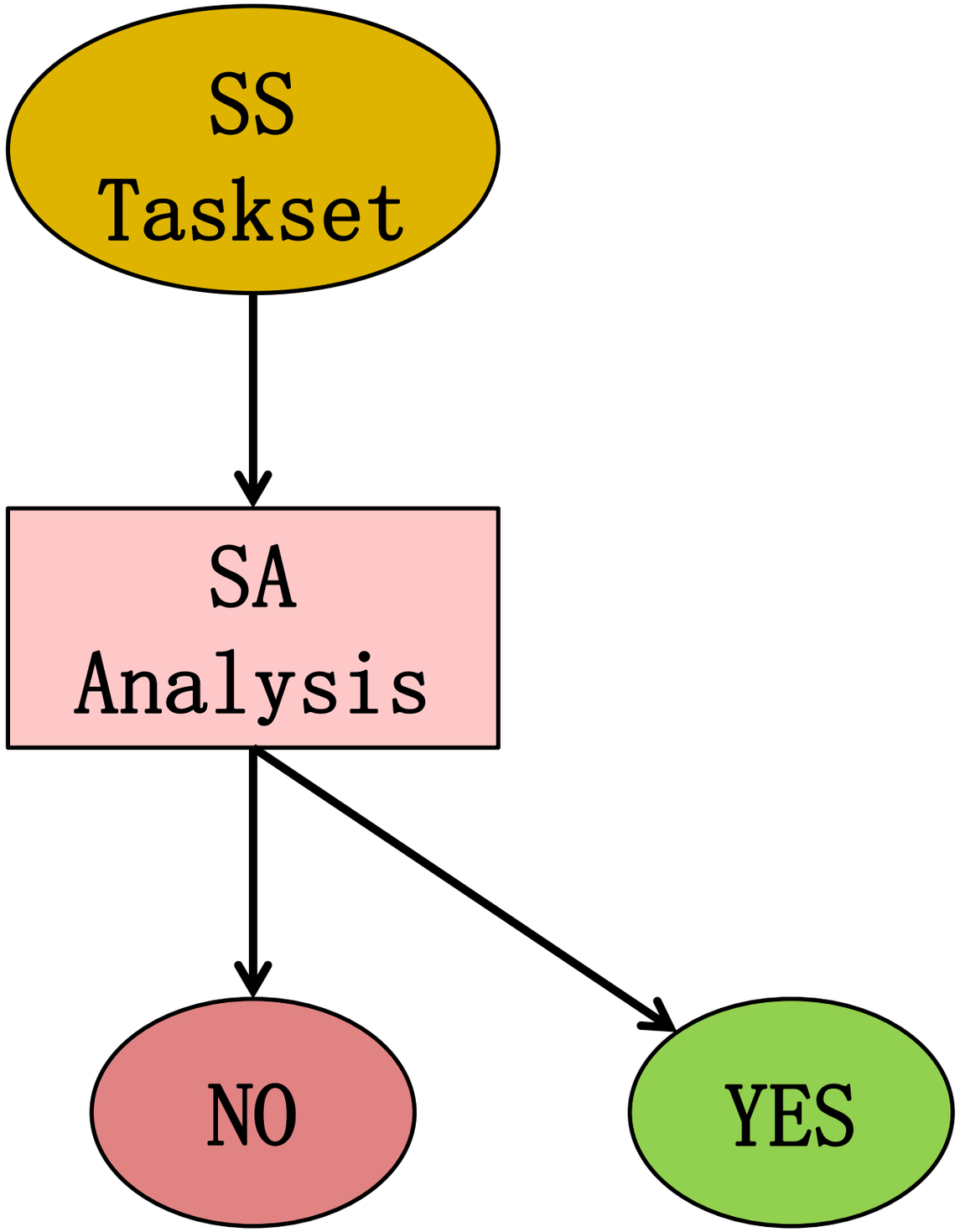
Typical computations executed in Linux cannot be precisely represented with classical task models (e.g., Liu&Leyland model)

- Tasks perform self-suspensions due to:
  - I/O operations, accelerators, semaphores, …

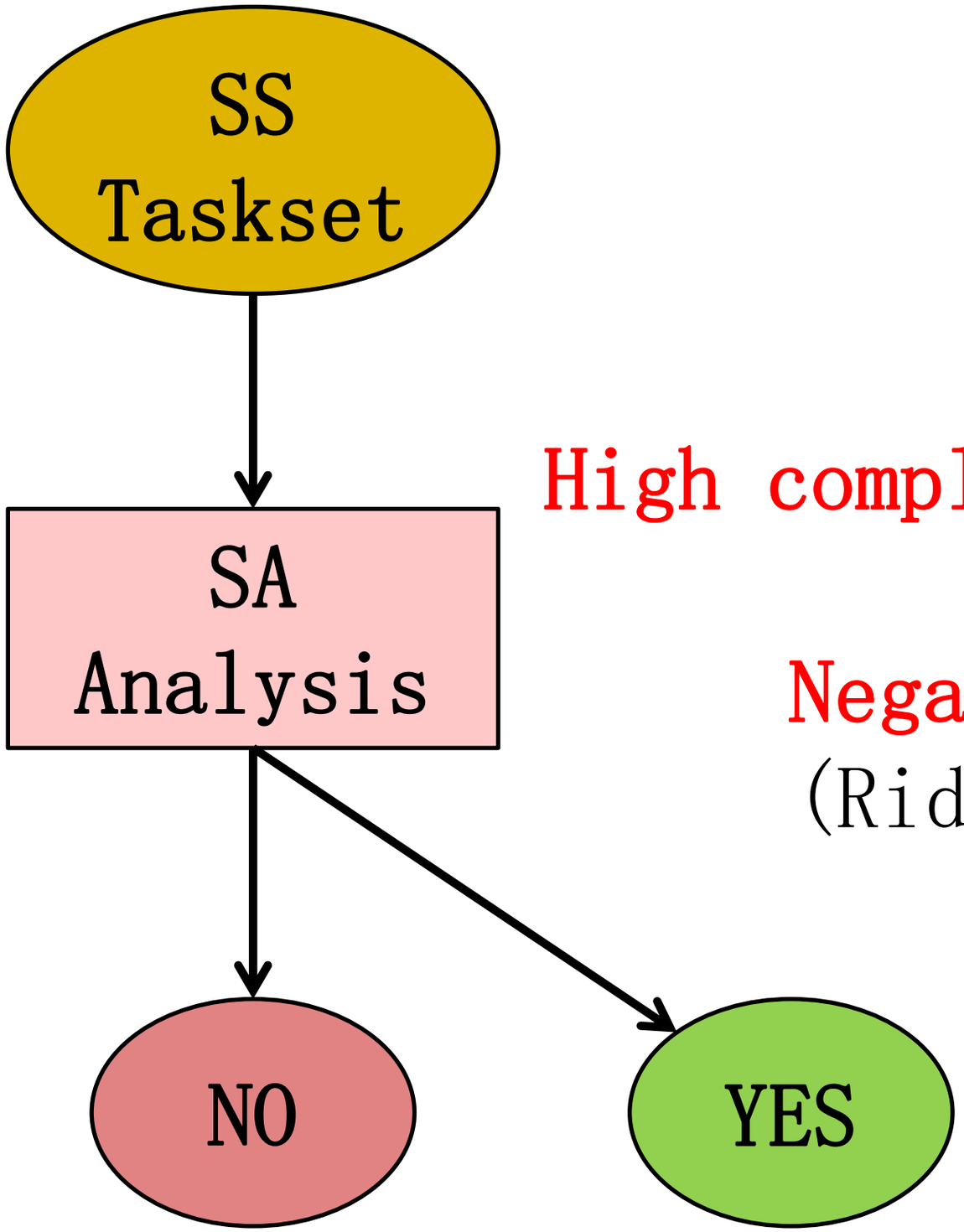
Self-suspending task model



Suspension Oblivious Analysis



Suspension Oblivious Analysis



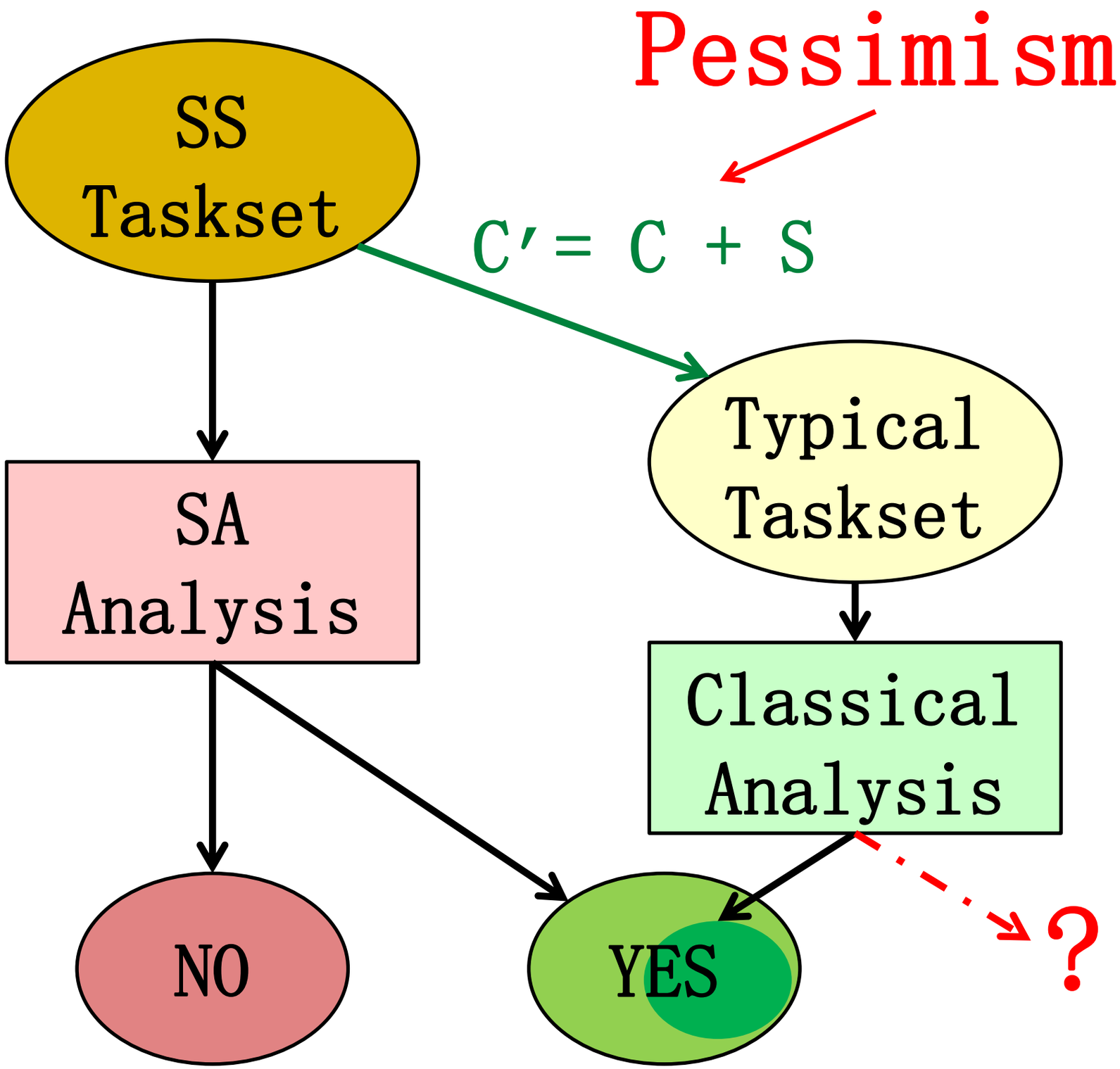
High complexity of analysis

+

Negative results

(Ridouard et al.)

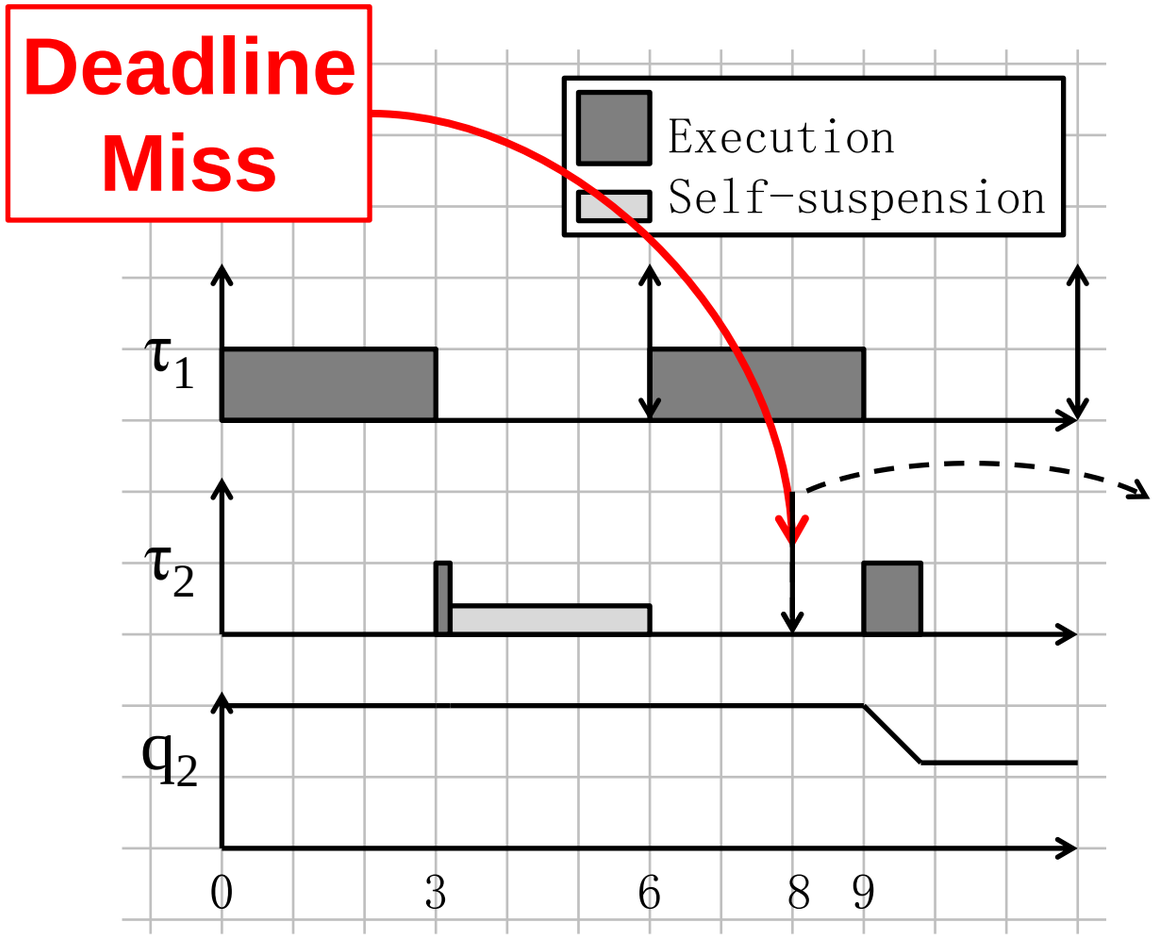
Suspension Oblivious Analysis



Problem With Current H-CBS

- A taskset resulting schedulable with SO analysis

  - $C_1 = 3$, $D_1 = T_1 = 6$;
  - $C_2 = 1$, $S_2 = 3$, $D_2 = T_2 = 8$;
  - $U = \frac{3}{6} + \frac{3+1}{8} = 1 \leq 1$
    - ✓ Schedulable under EDF with SO Analysis

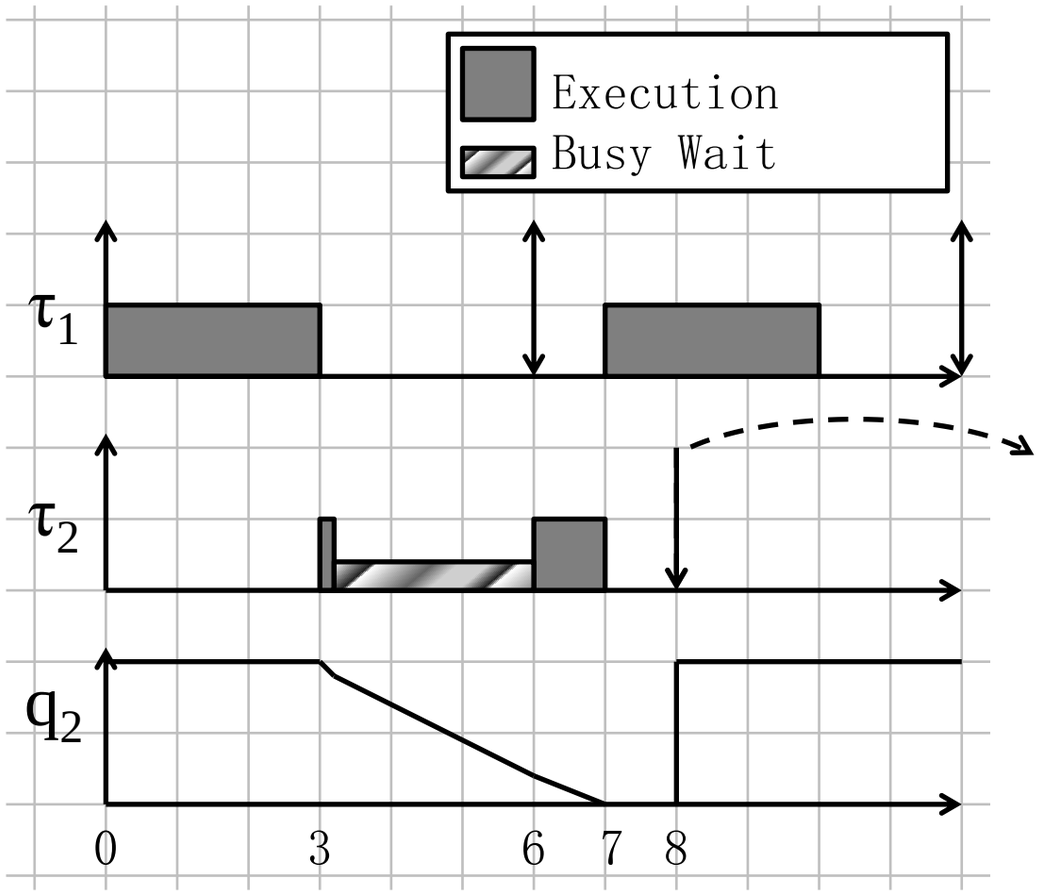


The server was not able to guarantee the reservation

Naïve Solution

- Substitute the self-suspension with busy waiting

  - $C_1 = 3$, $D_1 = T_1 = 6$;
  - $C_2 = 1$, $S_2 = 3$, $D_2 = T_2 = 8$;
  - $U = \frac{3}{6} + \frac{3+1}{8} = 1 \leq 1$
    - ✓ Schedulable under EDF with SO Analysis



- This solution **guarantees the isolation**, but
- RT tasks **steal CPU time** to best effort tasks
  - **CPU time is wasted** because of busy waiting

Needs

Desired solution:

- ✓ Takes the complexity advantages given by the **SO analysis**
- ✓ Guarantees **isolation** from **overrun** and **over-self-suspension**
- ✓ **No busy waiting**

H-CBS extension which guarantees isolation of self-suspending tasks schedulable with SO analysis:

H-CBS-SO algorithm

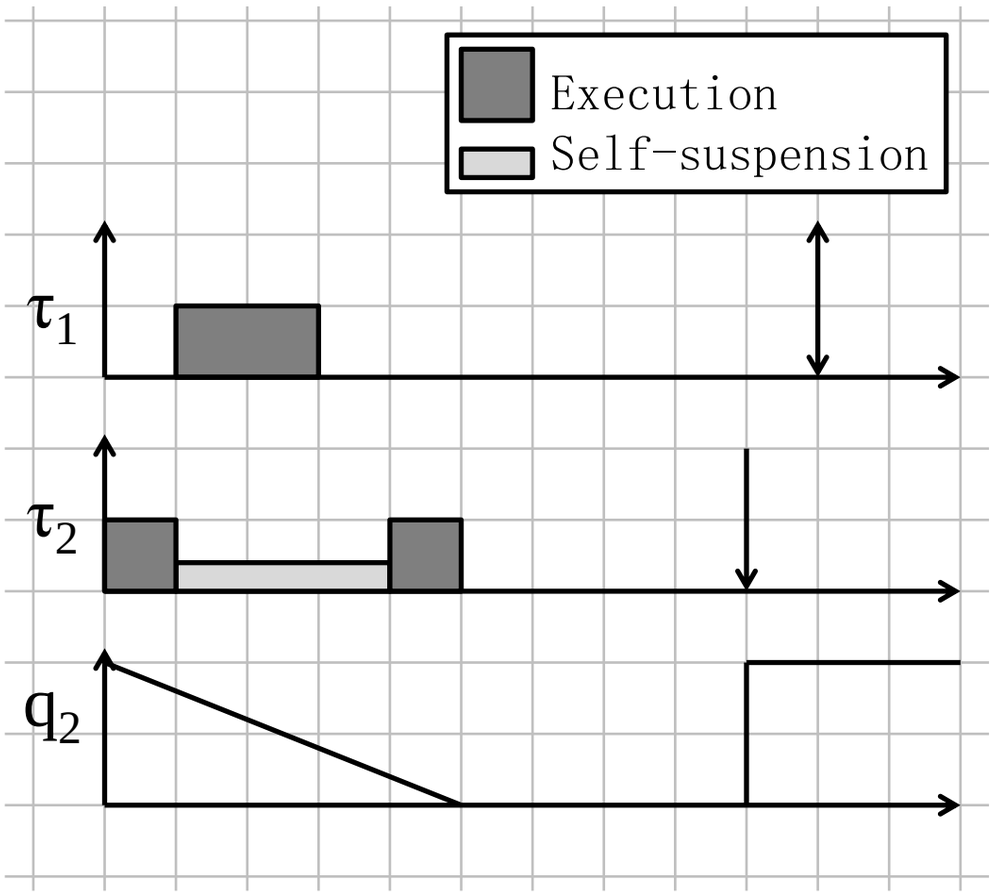
H-CBS-SO Algorithm

Solution: create an algorithm where

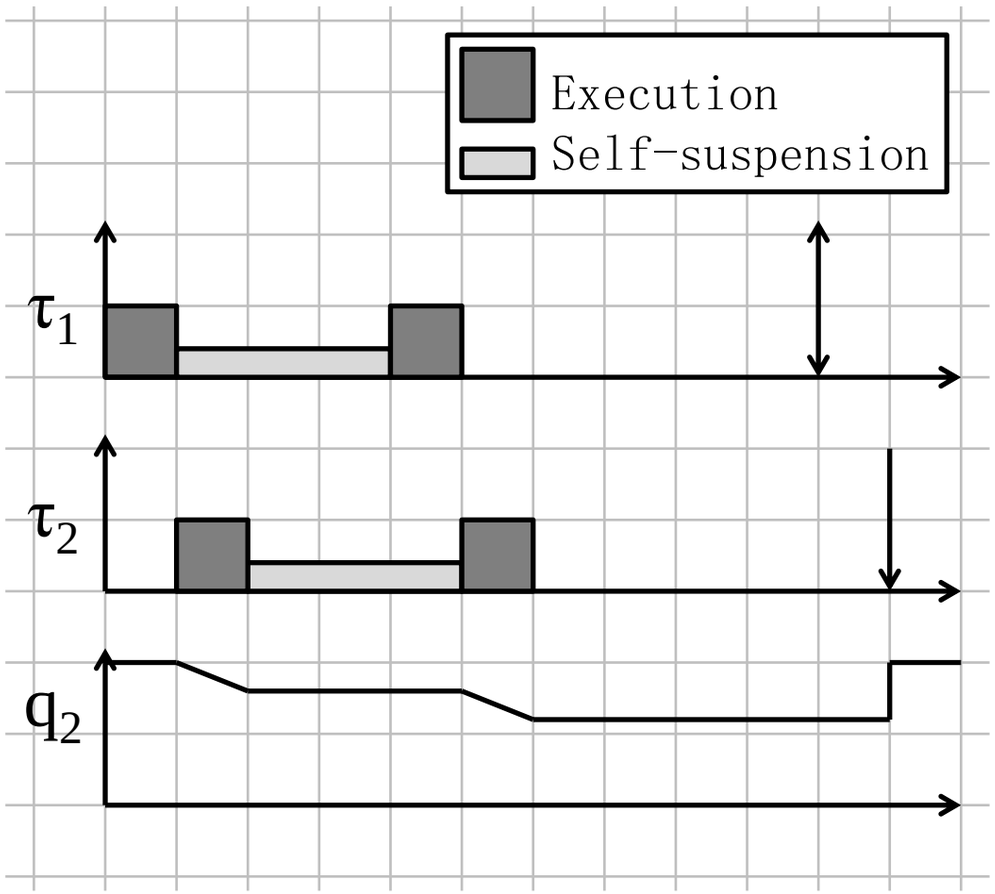
- The tasks are still able to **self-suspend**
- The SO analysis works thanks to a new budget accounting mechanism
- It is possible to use the self-suspension time to
  - Reduce the **response times** of other tasks
  - Execute **best-effort tasks**

H-CBS-SO Algorithm

Key idea: mimic the behavior of the H-CBS with busy waiting



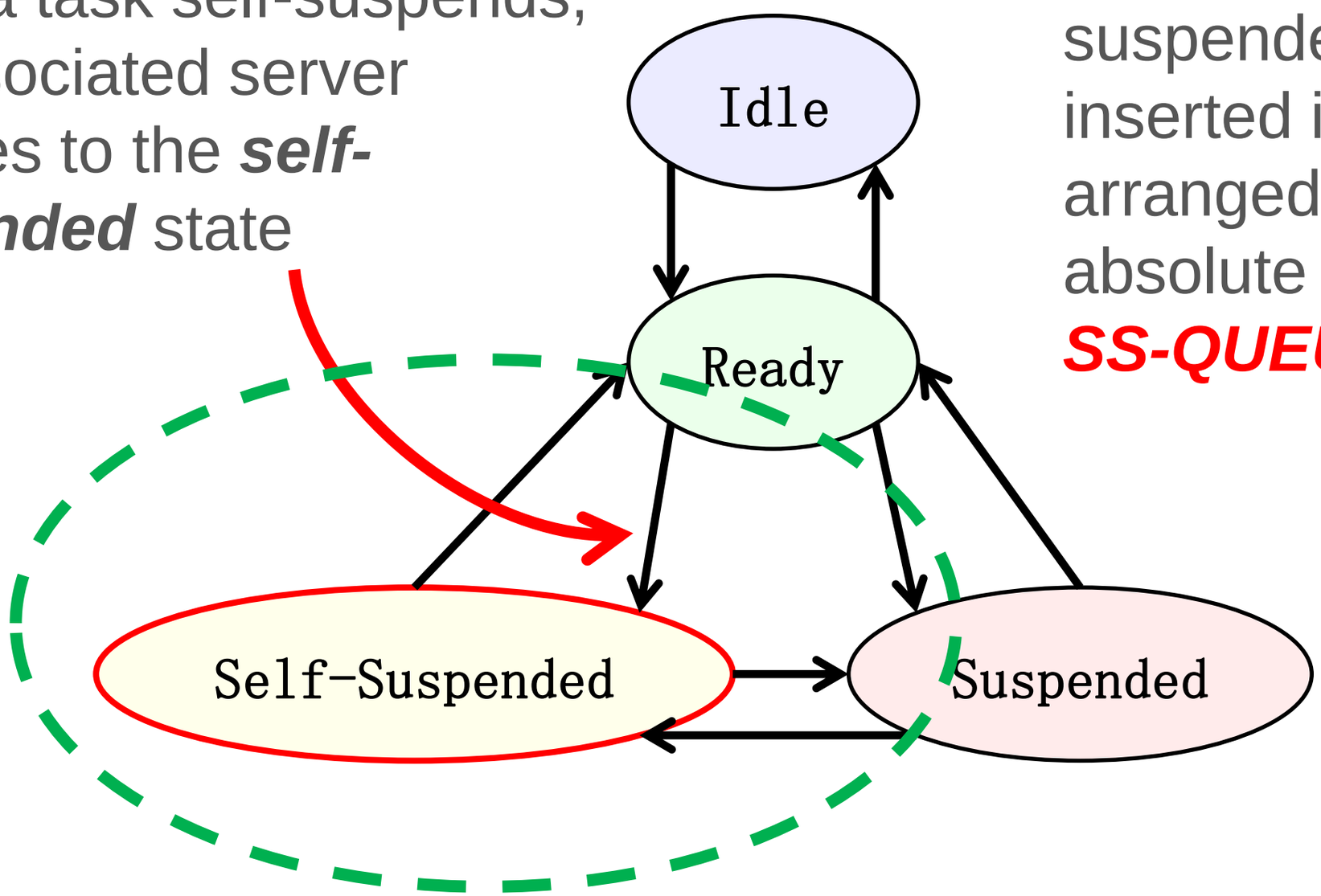
Budget Decreased



Budget Held

H-CBS-SO Algorithm

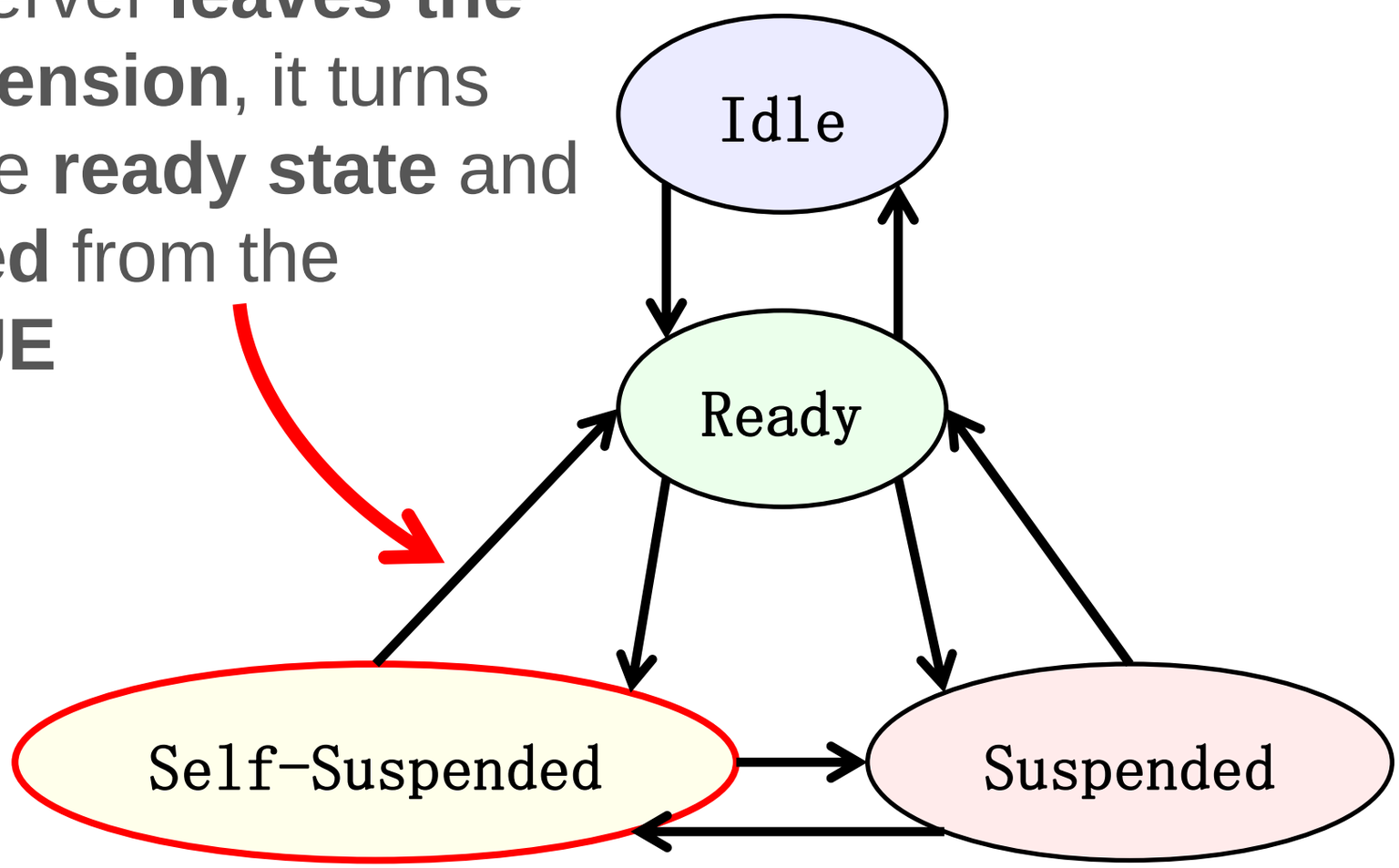
When a task self-suspends, the associated server switches to the ***self-suspended*** state



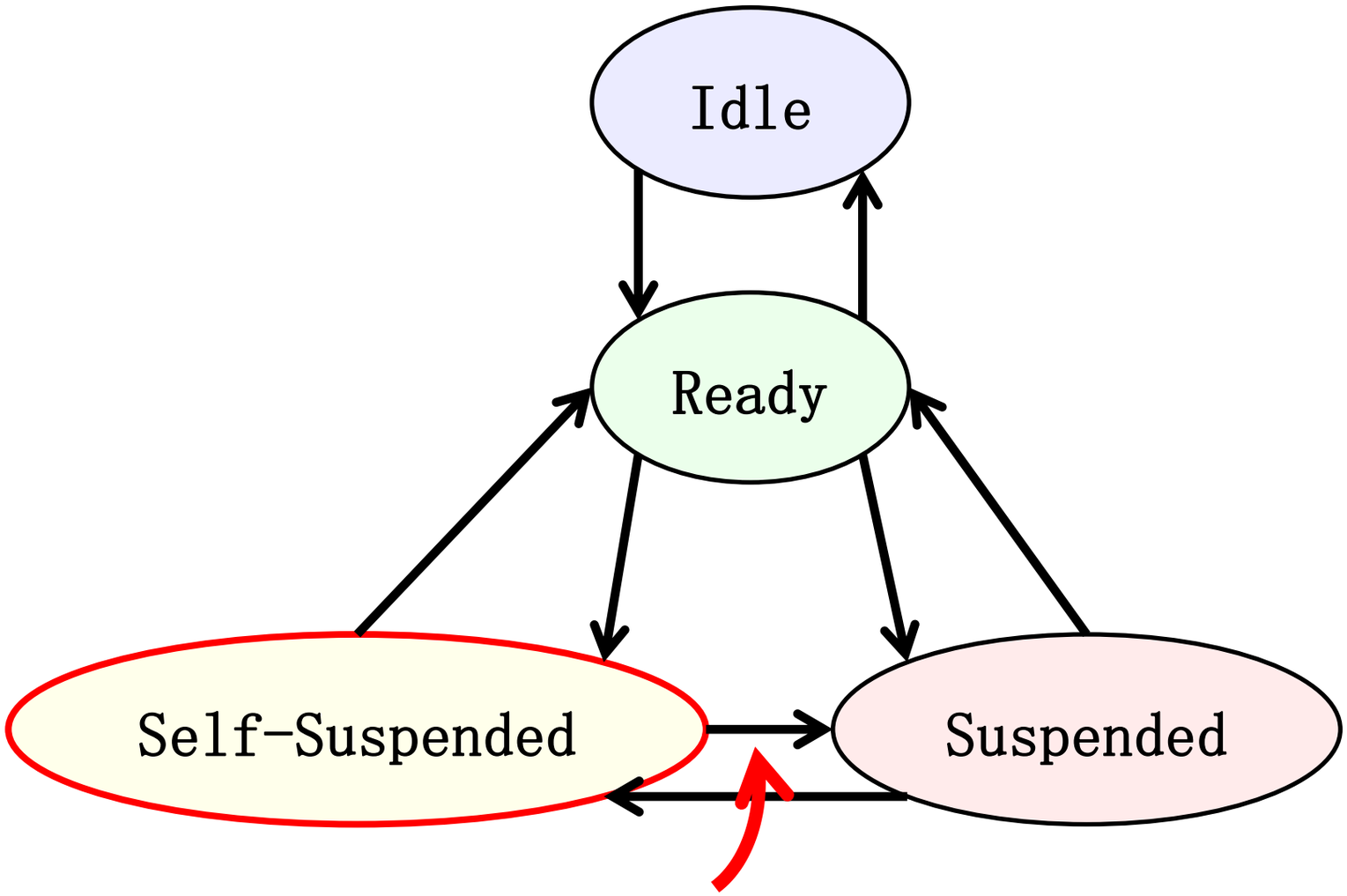
Servers in self-suspended state are inserted in a queue arranged by ascending absolute deadline, called ***SS-QUEUE***

H-CBS-SO Algorithm

When a server **leaves the self-suspension**, it turns back to the **ready state** and is **removed** from the **SS-QUEUE**

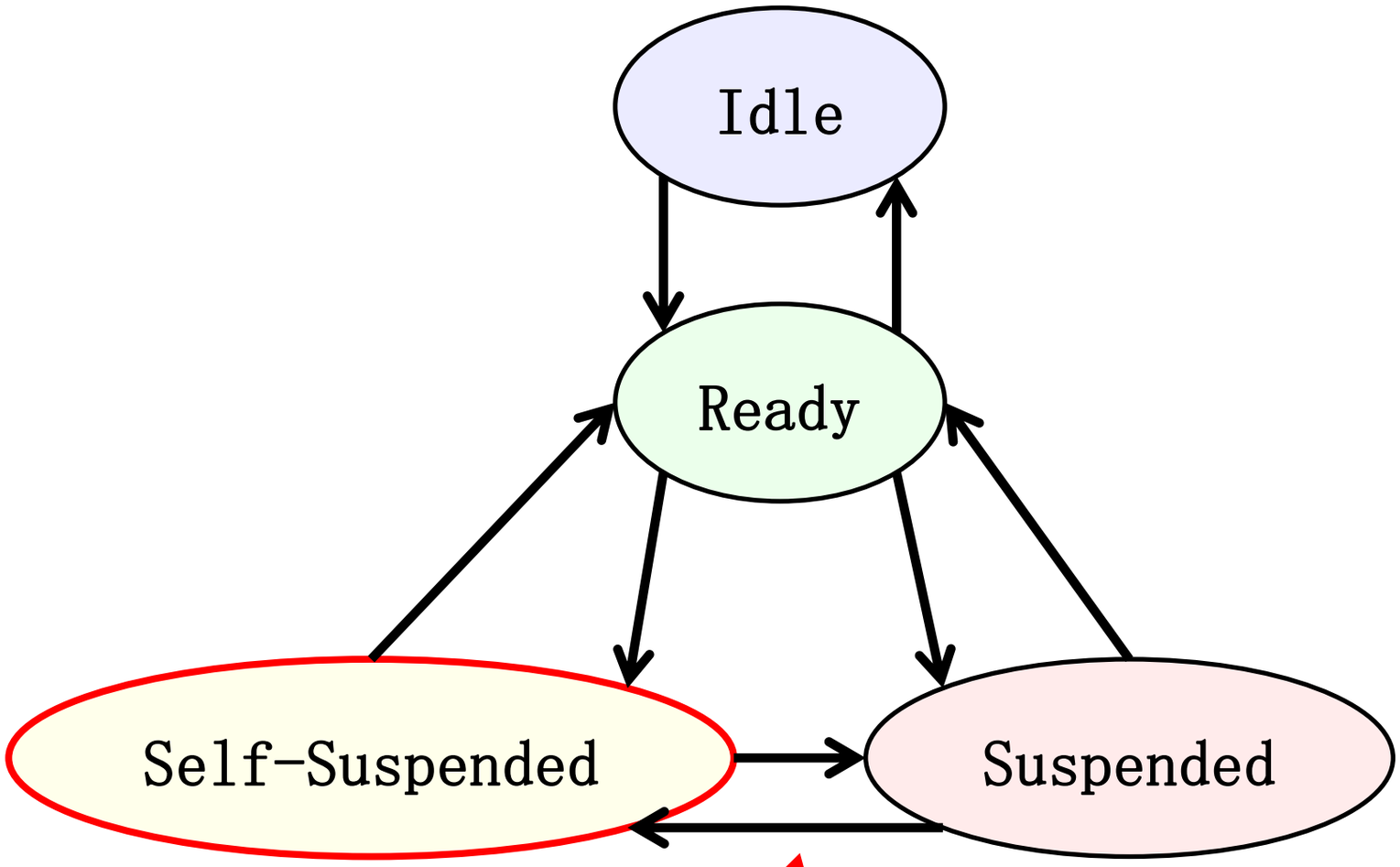


H-CBS-SO Algorithm



While a **server is self-suspended**, its **budget** is decreased and may **finish**, causing the removal from SS-QUEUE and a switch to the **Suspended** state.

H-CBS-SO Algorithm



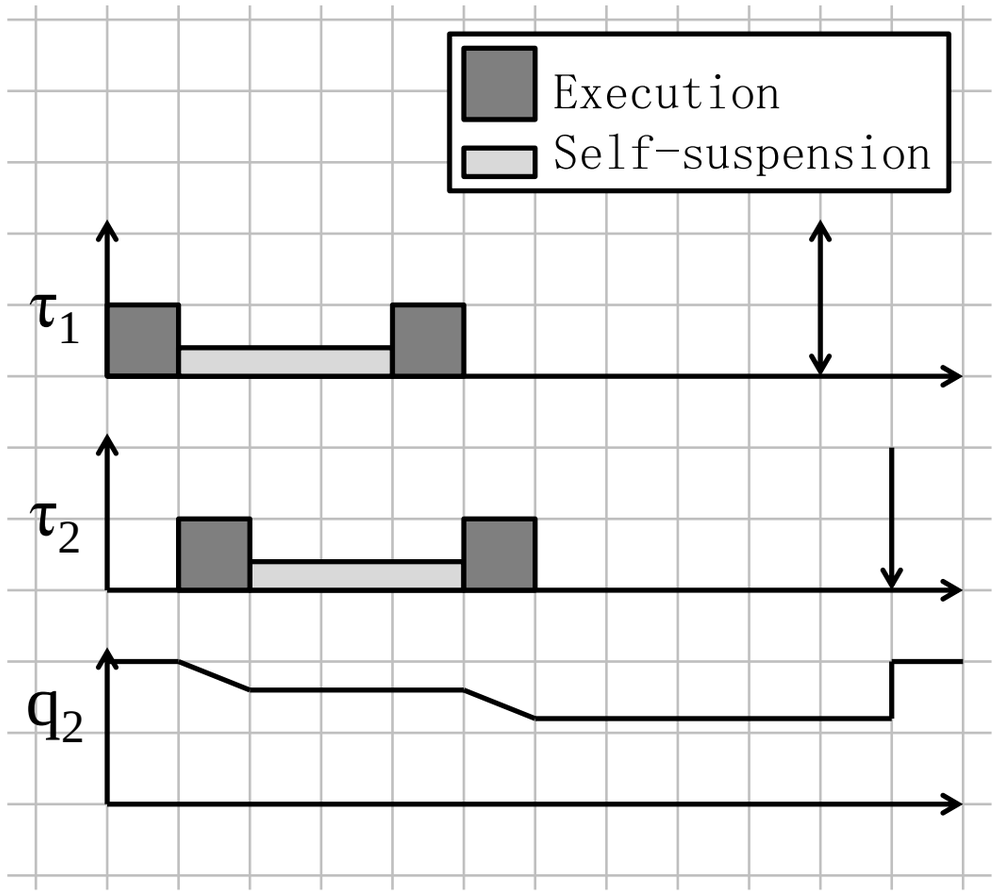
A **budget replenishment** happened to a Suspended server which was **self-suspended**

Nested Self-Suspension

The budget is decremented from the self-suspended task that would have executed in case of busy waiting, that is when:

- Another server with greater or equal deadline is running
- No other server is in the ready state (idle state)

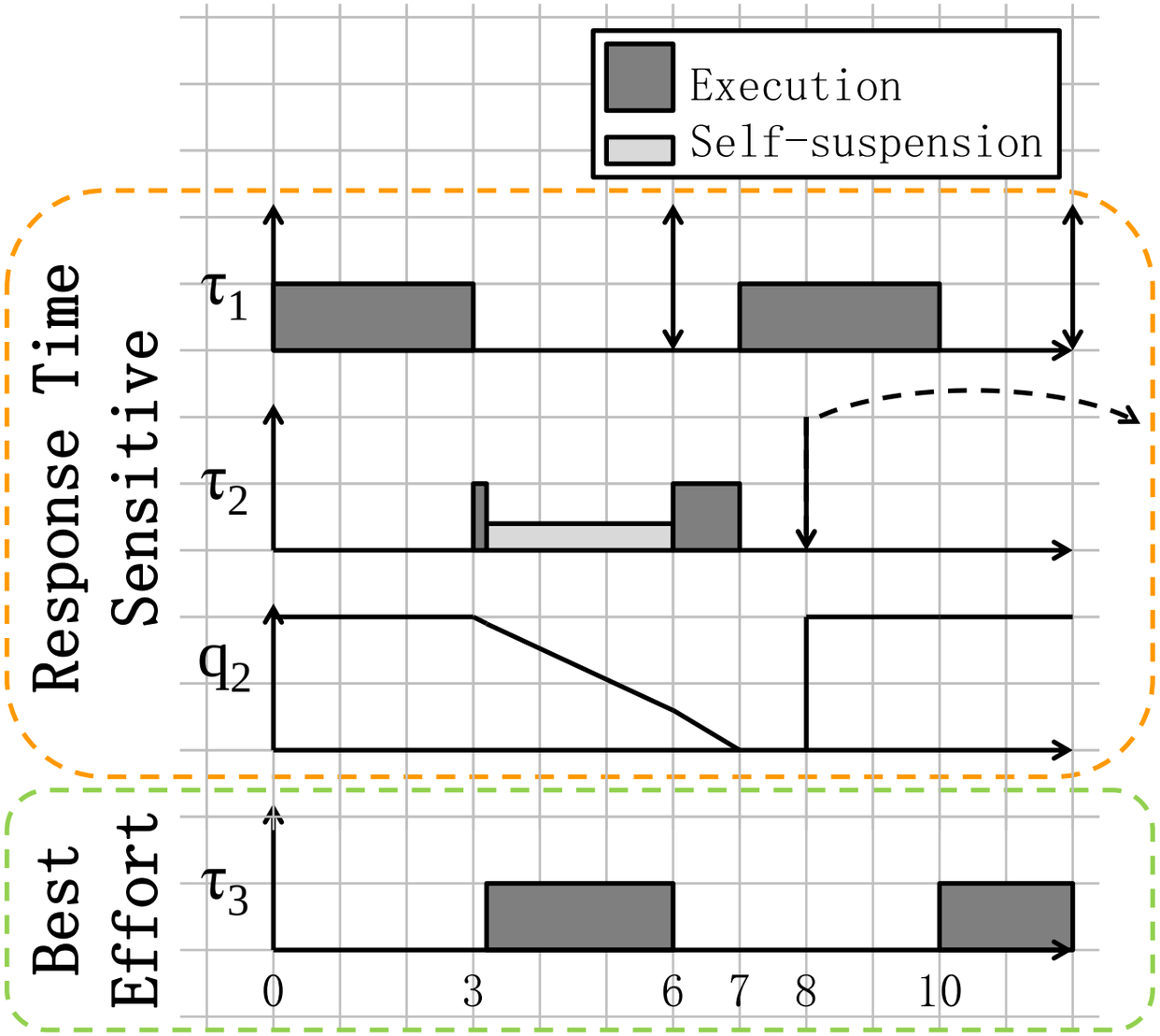
The behavior is the same of H-CBS + busy waiting



H-CBS-SO Algorithm

- **The budget is consumed while self-suspended**

- $C_1 = 3$, $D_1 = T_1 = 6$;
- $C_2 = 1$, $S_2 = 3$, $D_2 = T_2 = 8$;
- $U = \frac{3}{6} + \frac{3+1}{8} = 1 \leq 1$
  - ✓ Schedulable under EDF with SO Analysis



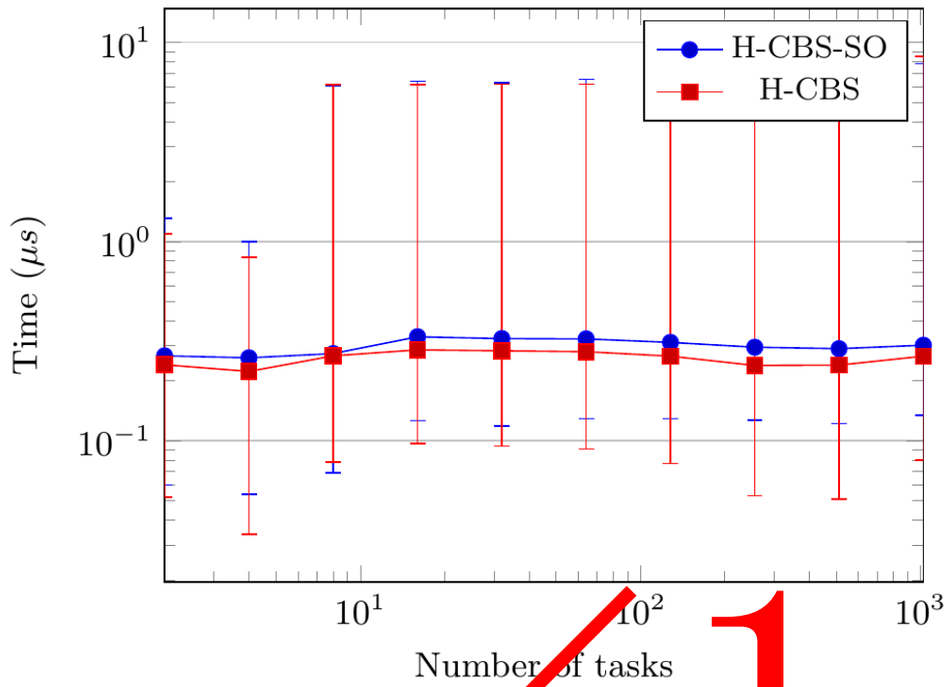
- **Guarantees back the schedulability + isolation**
- **Other tasks can execute during self-suspensions**

H-CBS-SO on Linux

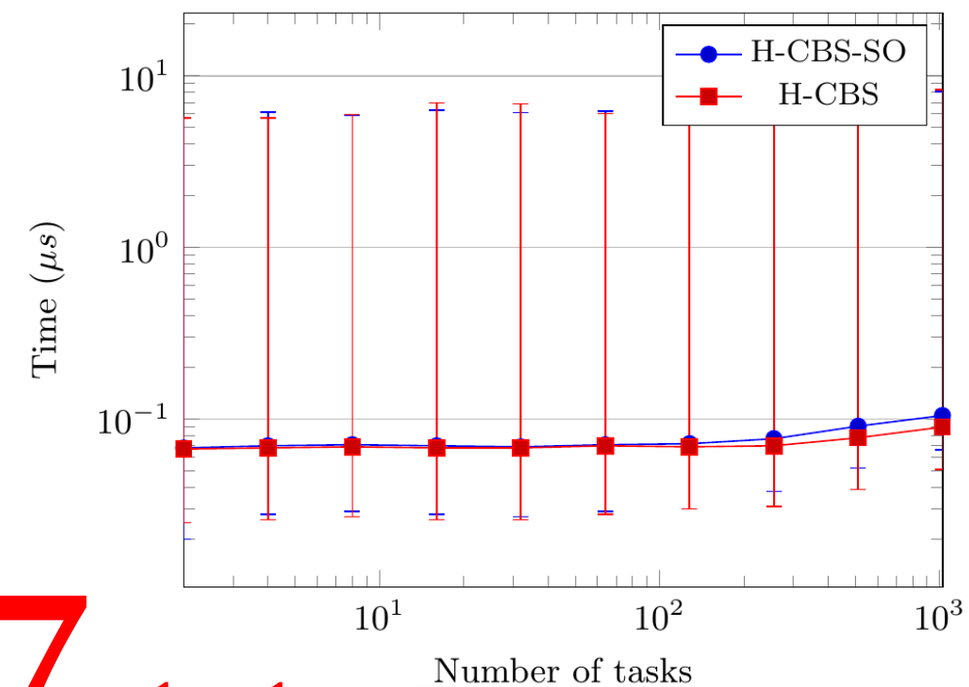
Based on SCHED_DEADLINE

- 1 server exists for 1 and only 1 task
- **Data structures**
  - SS_QUEUE implemented as RB-tree in task descriptor
    - ✓ ***O(log(n))*** for insertion and removal
    - ✓ ***O(1)*** complexity for accessing the head
- **Modified functions**
  - Detecting entrance and exit from self-suspension
  - Budget accounting for self-suspended servers

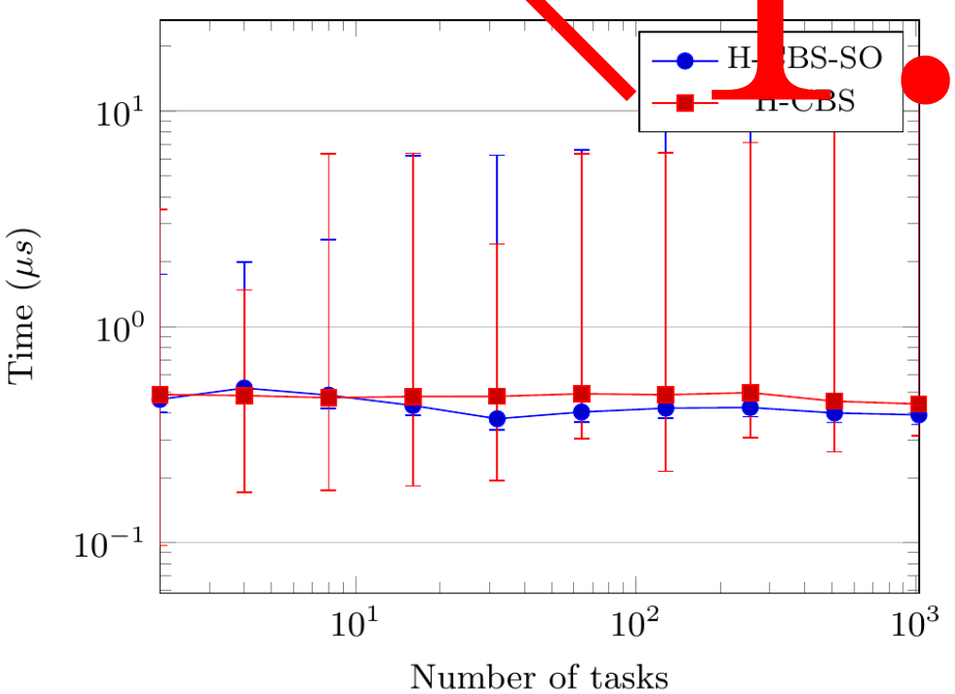
Linux Overhead



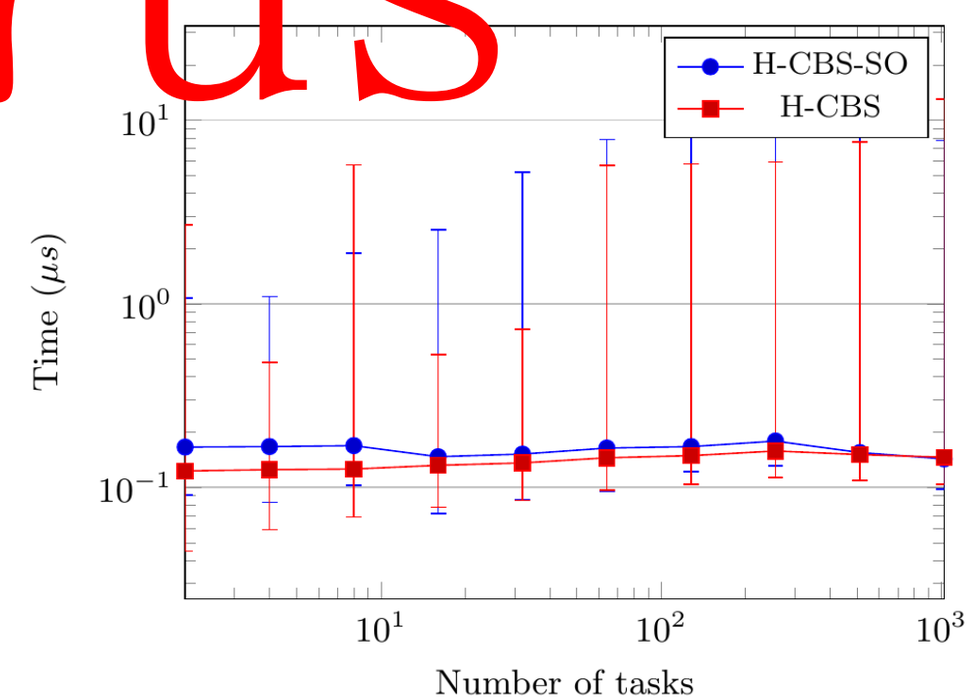
dequeue_task_dl() execution times.



update_curr_dl() execution times.



dl_task_timer() execution times.



enqueue_task_dl() execution times.

<< 1.7us

Intel Core 2 Duo at 3 GHz

Conclusions

Identified issue with H-CBS and SO analysis

Proposed new algorithm: H-CBS-SO

- ✓ Linux supports guaranteed reservation of self-suspending tasks under SO analysis providing temporal isolation among response time sensitive tasks
- ✓ Does not use busy waiting
- ✓ Does not significantly impact the system performance

Repository

The source code of SCHED_DEADLINE with H-CBS-SO is publicly available at

https://github.com/balsini/linux

Future Work

- Multicore H-CBS-SO extension and its implementation in SCHED_DEADLINE
- H-CBS extension for SA analysis
- Idle time reclaiming mechanism for H-CBS-SO

Thank you!

Alessio Balsini
a.balsini@sssup.it