Sistemi in tempo reale

Mode change

Giuseppe Lipari

Scuola Superiore Sant’Anna

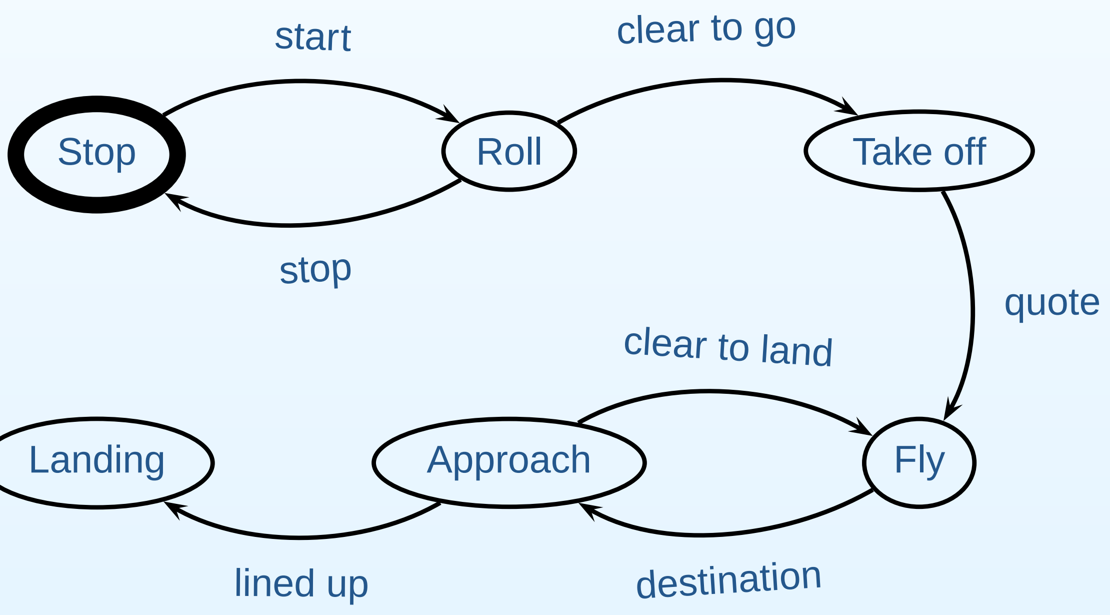
Pisa -Italy

Modes

- The system can have different working *modes*
- Each mode defines the same system under different working conditions;
  - As an example, consider an airplane;
  - Typical modes are take-off, cruise, and landing;
  - During each mode, the system has different control goals; and it must run different control algorithms.

Modes and transistions

- Modes can be represented by a state machine. For example, consider the previous example of airplane control:



Modes and transistions

- A mode is a node in the diagram (a *state*)
- A transition is an edge between two nodes:
  - A transition happens when certain conditions are verified;
  - For example, a user command, an internal condition, an external condition;
- Upon the occurrence of a transition:
  - terminate all tasks that are in the current mode and will not be active in the new mode;
  - call a *transition function*;
  - activate the new set of tasks to be executed.

Modes and tasks

- To implement modes:
- In general, there will be one global variable that identifies the current working mode (`currmode`);
- One *manager task* that identifies when modes must be changed;
- Modes can be implemented in two basic ways;
  1. **Type 1** A fixed set of tasks for all the modes; each task can execute different algorithms depending on the current mode;
  2. **Type 2** A different set of tasks for each mode.
- Of course, it is also possible to mix the two implementations.

Mode changes and consistency

- There are several problems the designer must deal with when designing an multi-mode real-time system;
- The main problem is what happens during the transition between two modes. In particular, we must deadl with
  1. How to change mode
  2. Consistency of variables
  3. Schedulability analysis
- Now we start dealing with problems 1 and 2.

Consistency

- Clearly, we cannot change the control algorithm at an arbitrary point while the algorithm is executing;
  - A control algorithm updates its internal state variables while executing;
  - we must ensure that the state variable does not remain in an inconsistent state when we change mode;
  - the same happens if the task is accessing a shared resource with a critical section protected by a mutex; we cannot interrupt it and change algorithm, otherwise the mutex remains locked!
- This means that the change of control algorithm must be *synchronized* with appropriate *checkpoints*;
  - A checkpoint is a point in the code when is *safe* to interrupt the algorithm maintaining the consistency of the data;
  - The "easiest" checkpoints are at the beginning and at the end of the task instance.

Implementation type 1

- Suppose we synchronize at the beginning of the task instance. The code for each task is something like the following;

```
while(1) {
  switch (currmode) {
  mode1 : // control algorithm
          // for mode 1
          break;
  mode2 : // control algorithm
          // for mode 2
          break;
  default : break;
  }
  task_endcycle();
}
```

Implementation type 1 - II

- The task cannot change mode while is executing. It can only change mode at the beginning of one of its istance;
- In this way we guarantee consistency of internal and external variables (state variables and output variables).
- To introduce other checkpoints, we could complicate the code by dividing each conrol algorithm in different blocks, and check the change of mode at the end of each block.

Implementation type 2

- In this case, each task can be active only in a subset of the modes.
- Define $\mathcal{T}_1$ the tasks active in mode 1, and $\mathcal{T}_2$ the task active in mode 2.
  - Suppose that the list of modes for which a task is active are stored in 2-dimension array `modes[task][mode]`.
  - If task `i` is active in mode `currmode`, then `modes[i][currmode]` is `true`, otherwise it is `false`.
- Typical code of the task;

```
while (1) {
  // control algorithm
  if (!mode[i][currmode]) task_disable();
  task_endcycle();
}
```

- The primitive `task_disable()` suspends the periodic activations; they will be enabled again by an explicit `task_activate()`

Type 1 vs. type 2

- In type 1, all tasks have the same parameters (period and priority) in every mode;
- In type 2, we have different tasks for different modes: therefore, from one mode to the other, we can change both the period, the priority and the computation time of a task.
- Type 2 is more general, whereas type 1 is more simple to implement.

Mode manager

- In both cases, we need a “mode manager” task that controls when the mode must be changed.
    - The mode manager can be a periodic or aperiodic task;
    - In the first case (periodic), it periodically observe the state of the system and of the external variables and decided a mode change;
    - In the second case (aperiodic), it is attached to an external interrupt (external condition) or it is explicitely activated by another task.
    - The mode manager implements the state machine and controls transition between modes.
- From now on, we consider only type 2 implementations.

Implementation type 2: manager

- The task manager is structured as follows

```
while (1) {
  if (modeIsChanged()) {
    old_mode = curr_mode;
    curr_mode = getNewMode();
    transition(old_mode, new_mode);
    for (i=0; i < NTASK; i++) {
      if (mode[i][curr_mode] && !mode[i][old_mode])
        task_activate(tid[i]);
    }
  }
  task_endcycle();
}
```

Mode Manager

- The manager is a periodic task that periodically checks for occurrence of mode changes.
- It waits for a change of mode (function `modeIsChanged()`)
- When it happens, performs transition functions and activates all tasks belonging to the new mode and not active in the old mode.

Transitions

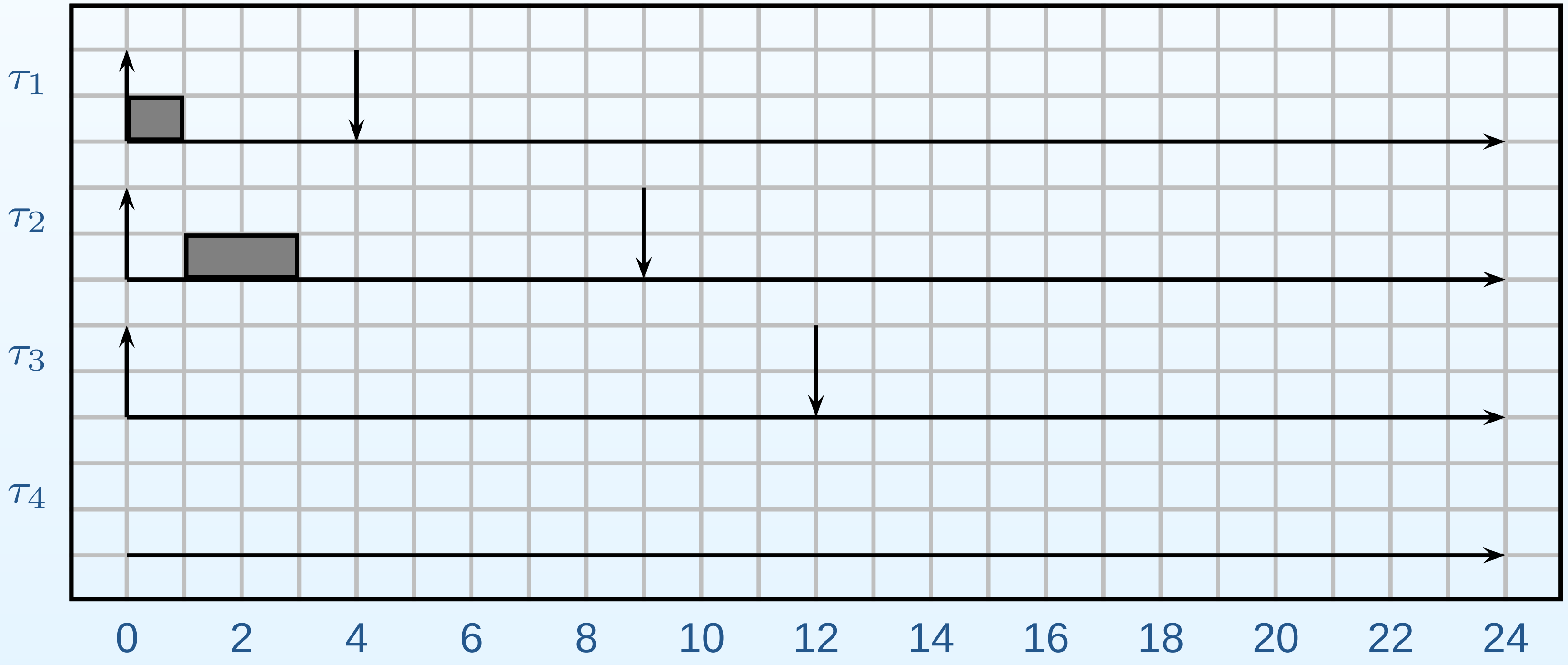
- Suppose the system must change from mode 1 to mode 2.
- To ensure a *smooth* transition between two modes, the states of control algorithms of mode 2 must be properly initialized;
- In other words, the initial conditions of mode 2 depend on the state conditions of mode 1.
  - Suppose, as an example, that we want to guarantee continuity of the signal and of the first derivative of the signal.
  - The, the internal conditions of the controller for mode 2 must be set so to ensure these two conditions;
- From a software point of view, for each transition we must call a set of functions to adjust the initial conditions of all control algorithms.

Scheduling analysis

- Another important problem is schedulability:
- Suppose we are changing from mode 1 to mode 2, and that $\mathcal{T}_1$ is the set of tasks active in mode 1 and $\mathcal{T}_2$ is the set of tasks that are active during mode 2.
  - Set $\mathcal{T}_1 \backslash \mathcal{T}_2$ is the set of tasks that *leave the mode*;
  - Set $\mathcal{T}_2 \backslash \mathcal{T}_1$ is the set of tasks that *enter the mode*.
- It is important to guarantee that the system continue to be schedulable;
- Even if $\mathcal{T}_1$ and $\mathcal{T}_2$, each one considered in isolation, are schedulable, if the transistion is not done properly, some deadline could be missed during the transitory.

Example of deadline miss during transition

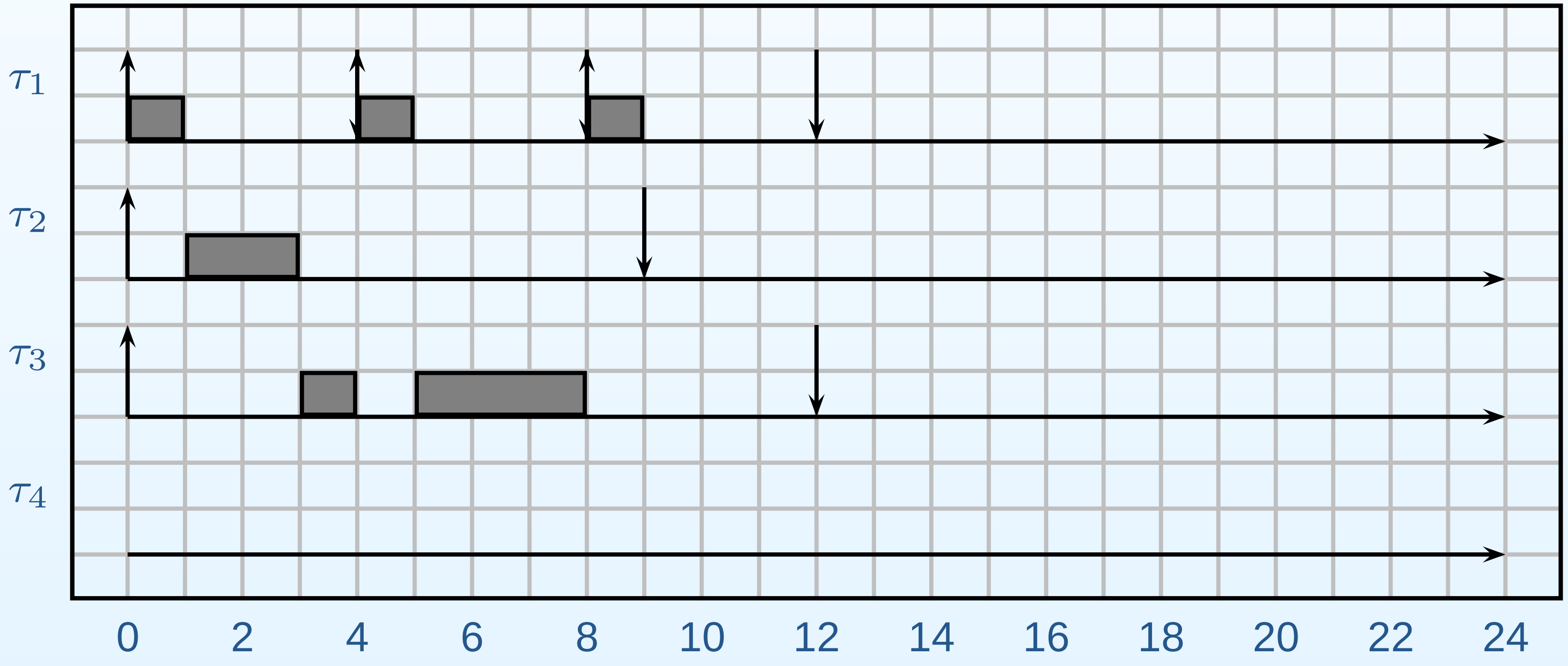
- Consider $\mathcal{T}_1 = \{\tau_1, \tau_2, \tau_3\}$ and $\mathcal{T}_2 = \{\tau_1, \tau_4, \tau_5\}$ with:
  - $\tau_1 = (1, 4)$, $\tau_2 = (2, 9)$, $\tau_1 = (5, 12)$, and $\tau_2 = (3, 9)$, $\tau_1 = (2, 12)$;
  - Transition starts at time $t = 2.5$;



- Mode Change at time $t = 2.5$; τ_2 suspended at time $t = 3$

Example of deadline miss during transition

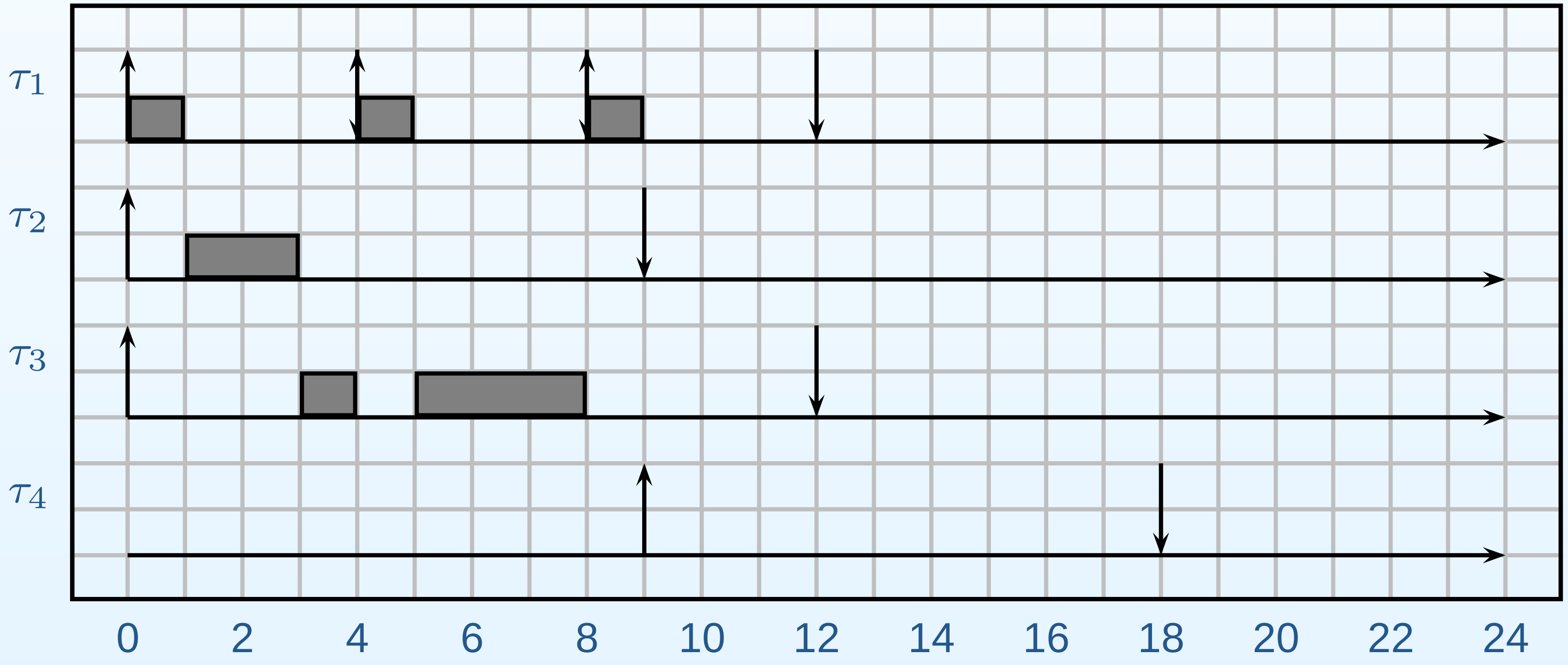
- Consider $\mathcal{T}_1 = \{\tau_1, \tau_2, \tau_3\}$ and $\mathcal{T}_2 = \{\tau_1, \tau_4, \tau_5\}$ with:
  - $\tau_1 = (1, 4)$, $\tau_2 = (2, 9)$, $\tau_1 = (5, 12)$, and $\tau_2 = (3, 9)$, $\tau_1 = (2, 12)$;
  - Transition starts at time $t = 2.5$;



- Mode Change at time $t = 2.5$; τ_2 suspended at time $t = 3$

Example of deadline miss during transition

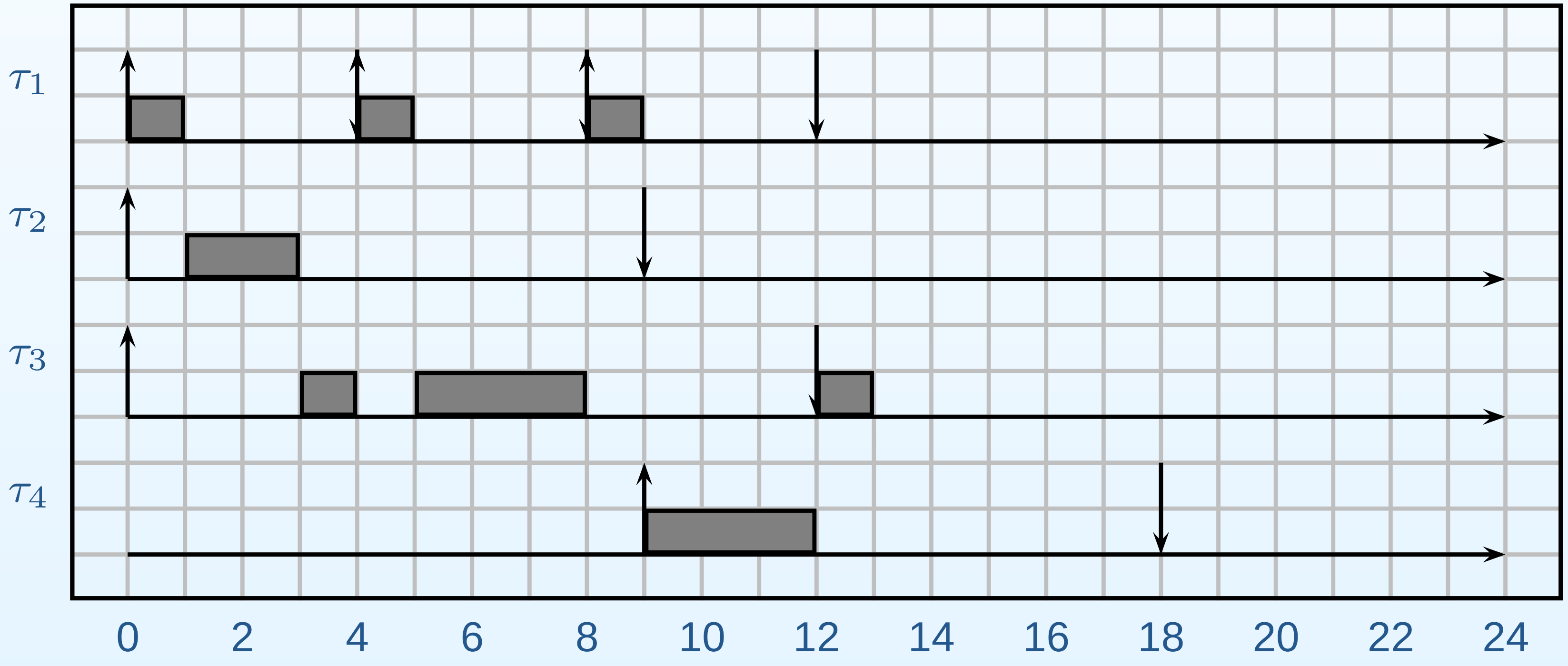
- Consider $\mathcal{T}_1 = \{\tau_1, \tau_2, \tau_3\}$ and $\mathcal{T}_2 = \{\tau_1, \tau_4, \tau_5\}$ with:
    - $\tau_1 = (1, 4)$, $\tau_2 = (2, 9)$, $\tau_1 = (5, 12)$, and $\tau_2 = (3, 9)$, $\tau_1 = (2, 12)$;
    - Transition starts at time $t = 2.5$;



- Mode Change at time $t = 2.5$; τ_2 suspended at time $t = 3$
- Task τ_4 executes instead of task τ_2 from time $t = 9$

Example of deadline miss during transition

- Consider $\mathcal{T}_1 = \{\tau_1, \tau_2, \tau_3\}$ and $\mathcal{T}_2 = \{\tau_1, \tau_4, \tau_5\}$ with:
  - $\tau_1 = (1, 4)$, $\tau_2 = (2, 9)$, $\tau_1 = (5, 12)$, and $\tau_2 = (3, 9)$, $\tau_1 = (2, 12)$;
  - Transition starts at time $t = 2.5$;



- Mode Change at time $t = 2.5$; τ_2 suspended at time $t = 3$
- Task τ_4 executes instead of task τ_2 from time $t = 9$

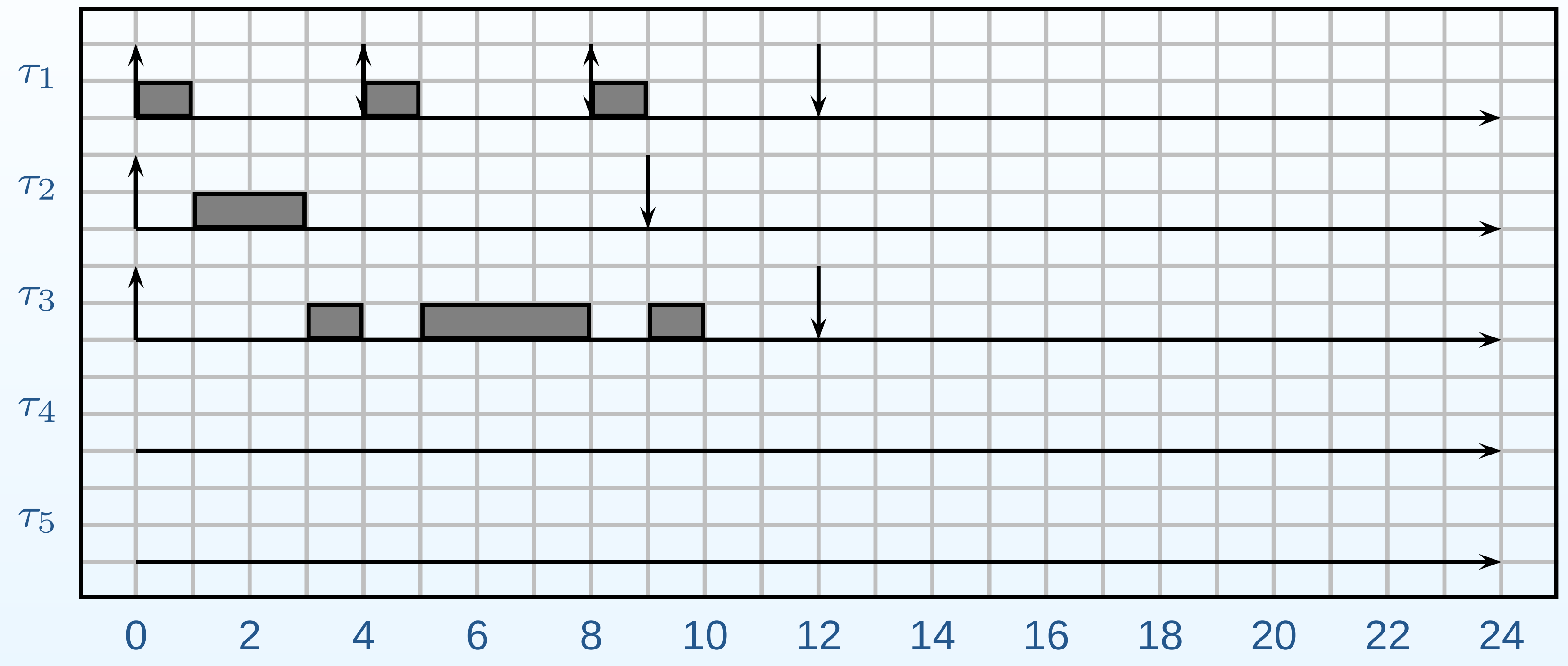
Rules

- The only way to avoid this problem is to allow the transition only in certain istants of time;
- We must ensure that all tasks that leave the system have completed, before activating the new tasks.
- General rule: *first de-activate all tasks that leave the mode, then activate the tasks that enter the mode*
    - In the previous example, this rule was not respected: task τ_4 is activated before task τ_3 is de-activated.
    - Therefore, the earliest instant at which the transition can be done is time 12, when τ_3 has completed.
- The rule above can be re-expressed as: *the earliest time at which the new tasks can be activated is the largest absolute deadline among all tasks that leave the system*
- This means that the transition has a delay.

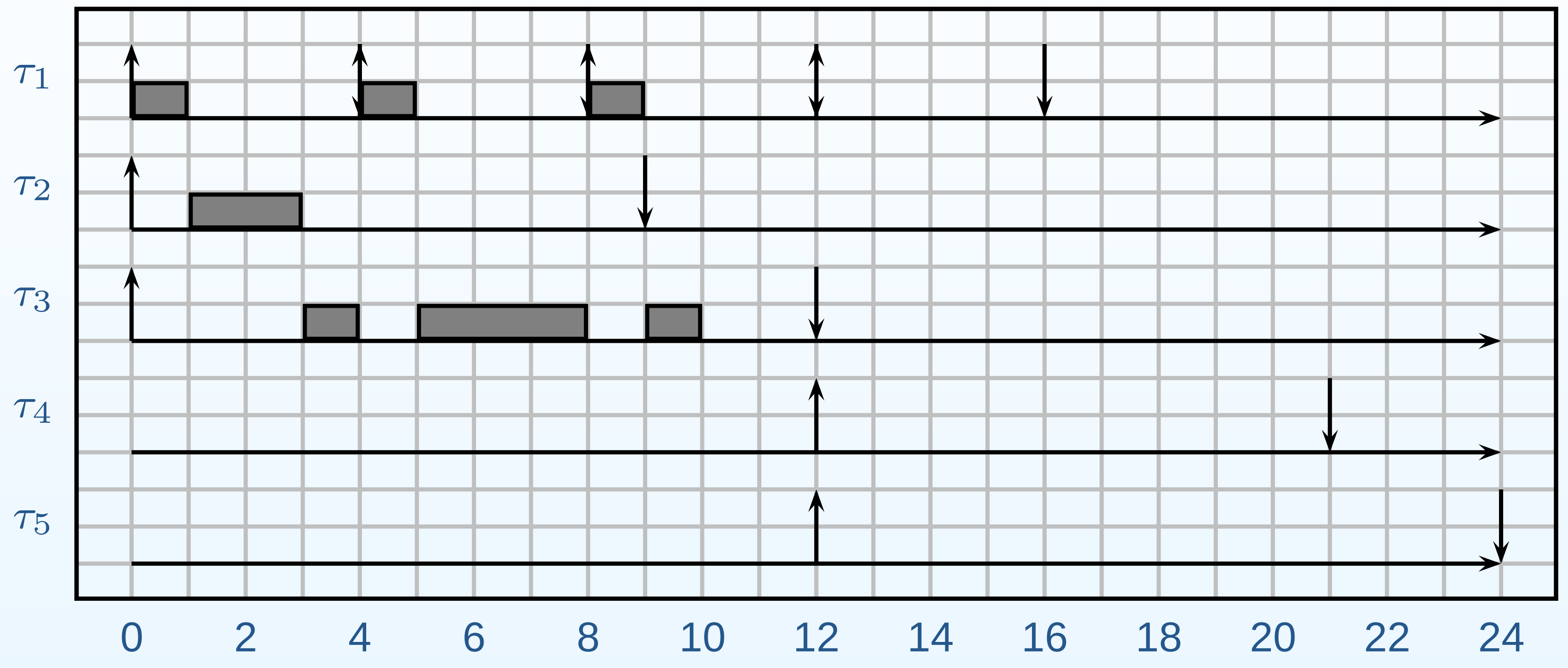
Type I and type II

- The same problem is for implementations of type 1 and of type 2
  - In type 1, each task can be considered as a different task in each mode, with a different computation time.

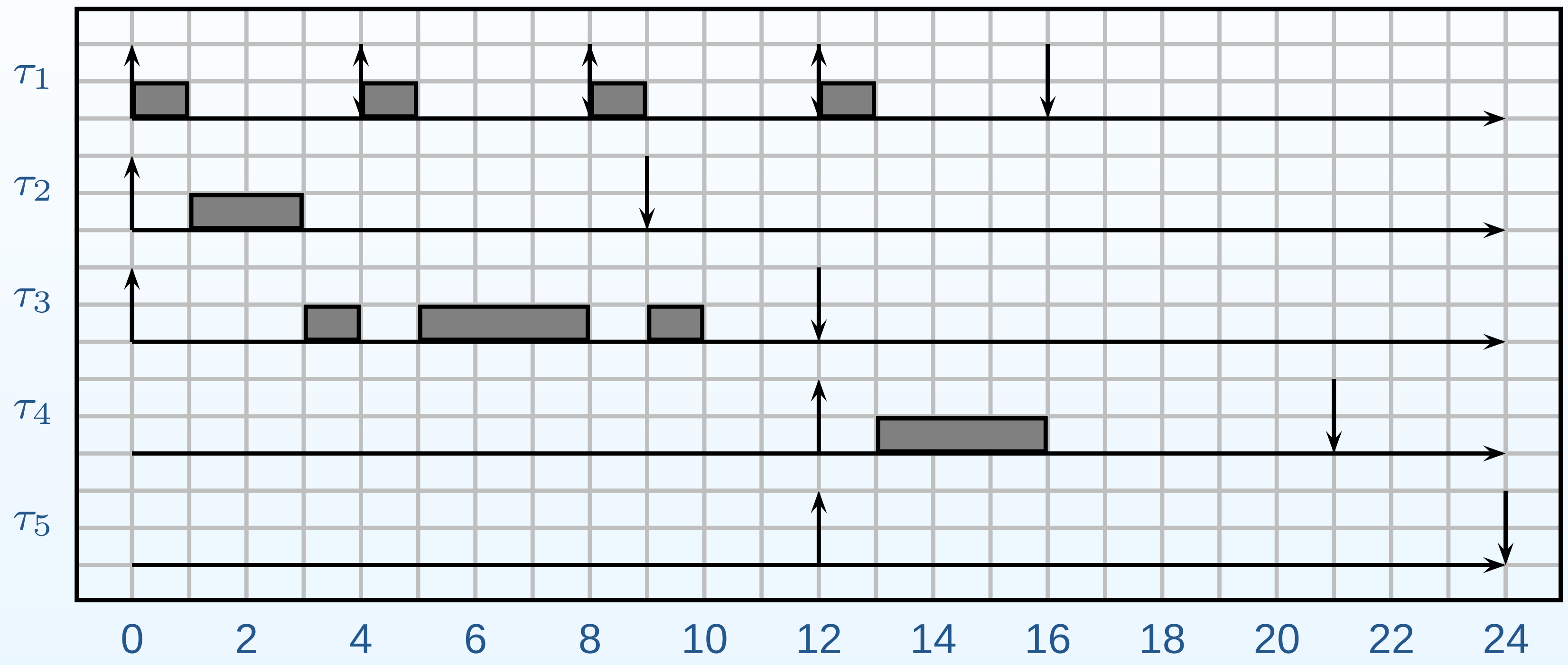
Example revised



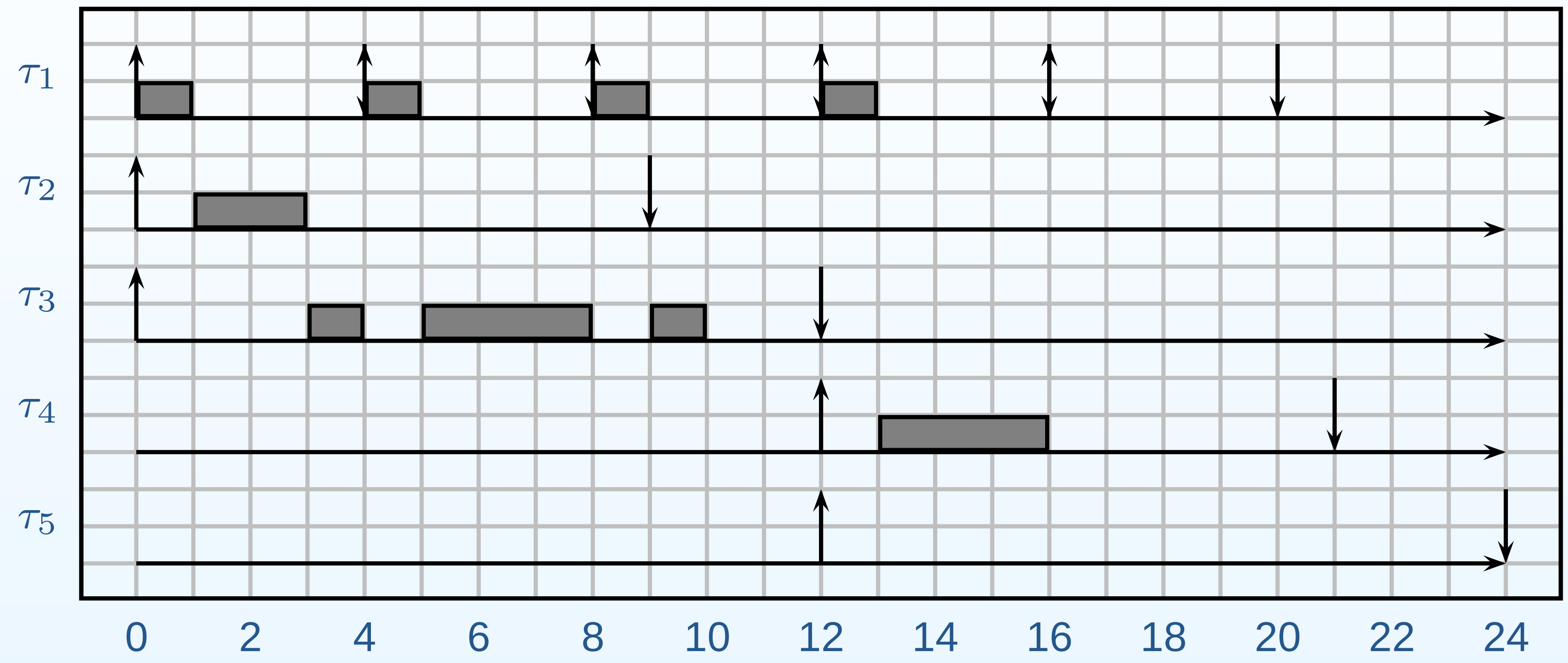
Example revised



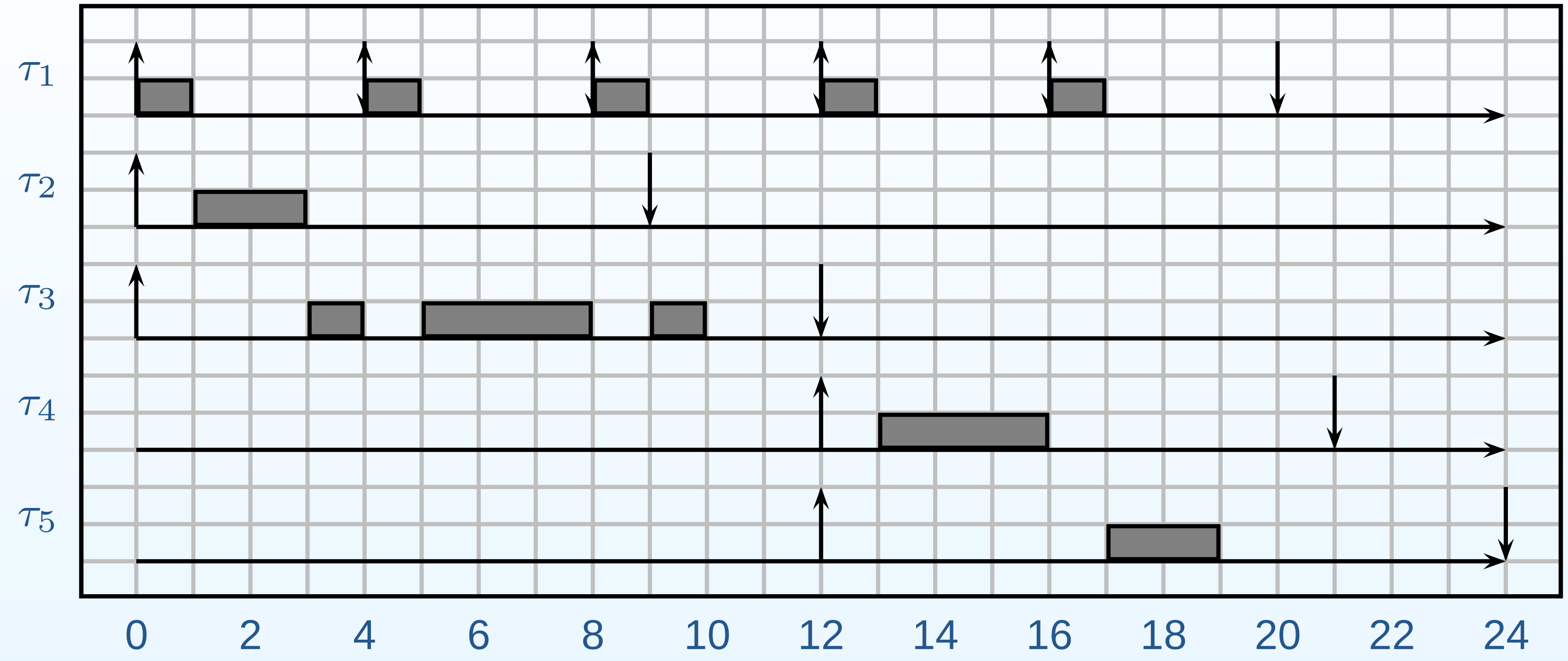
Example revised



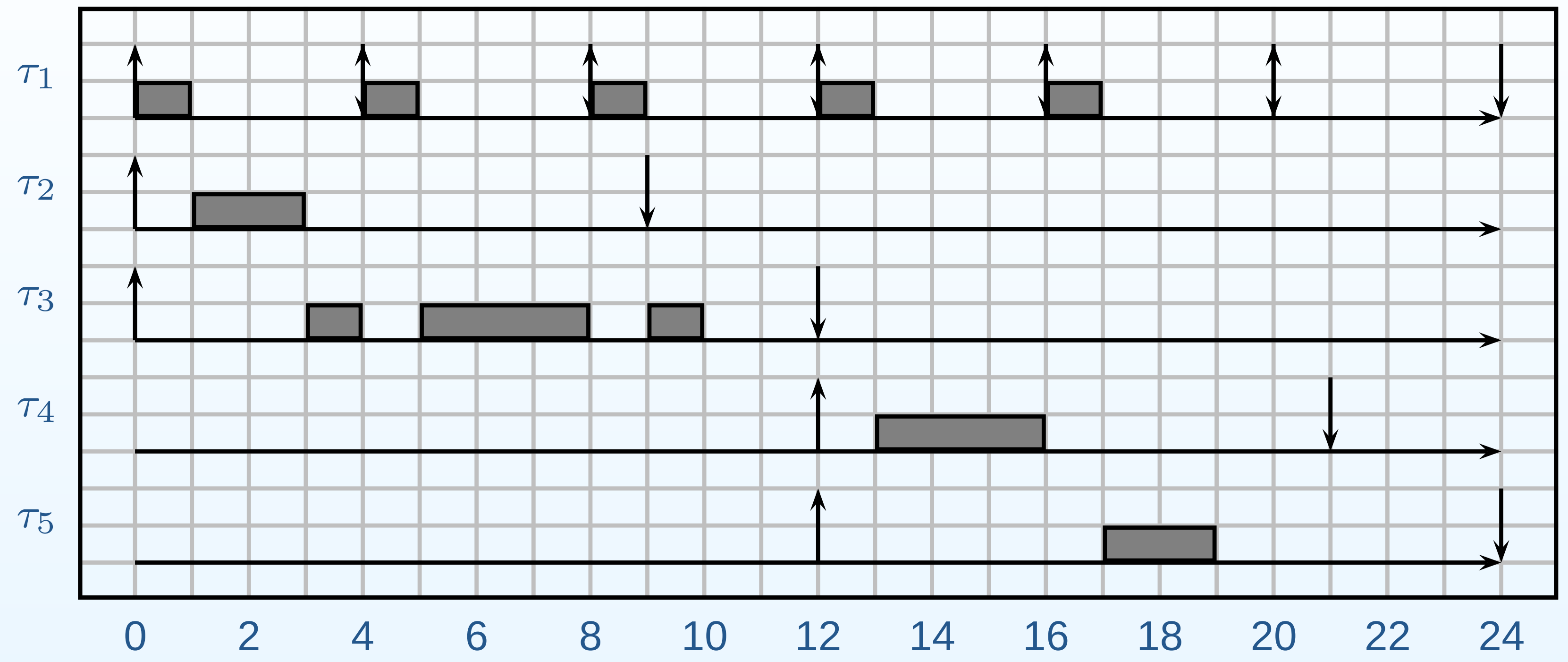
Example revised



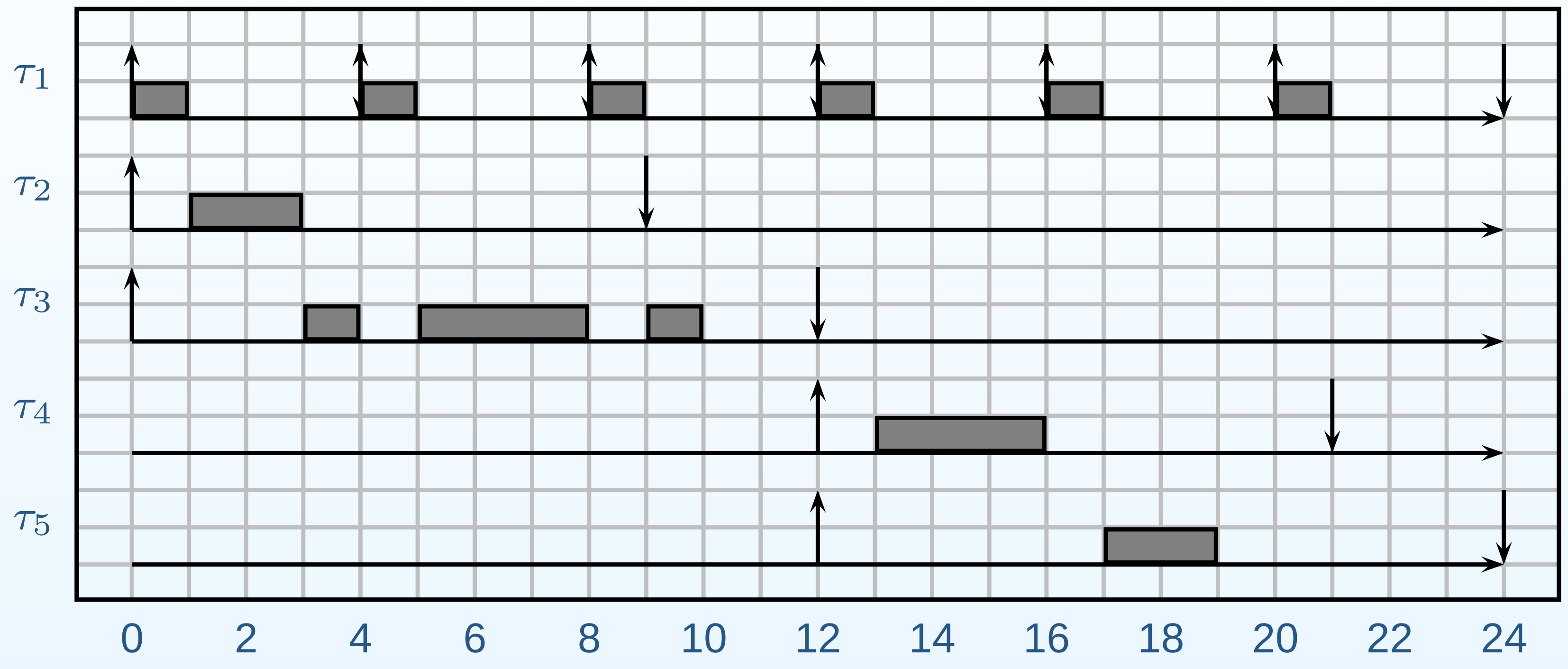
Example revised



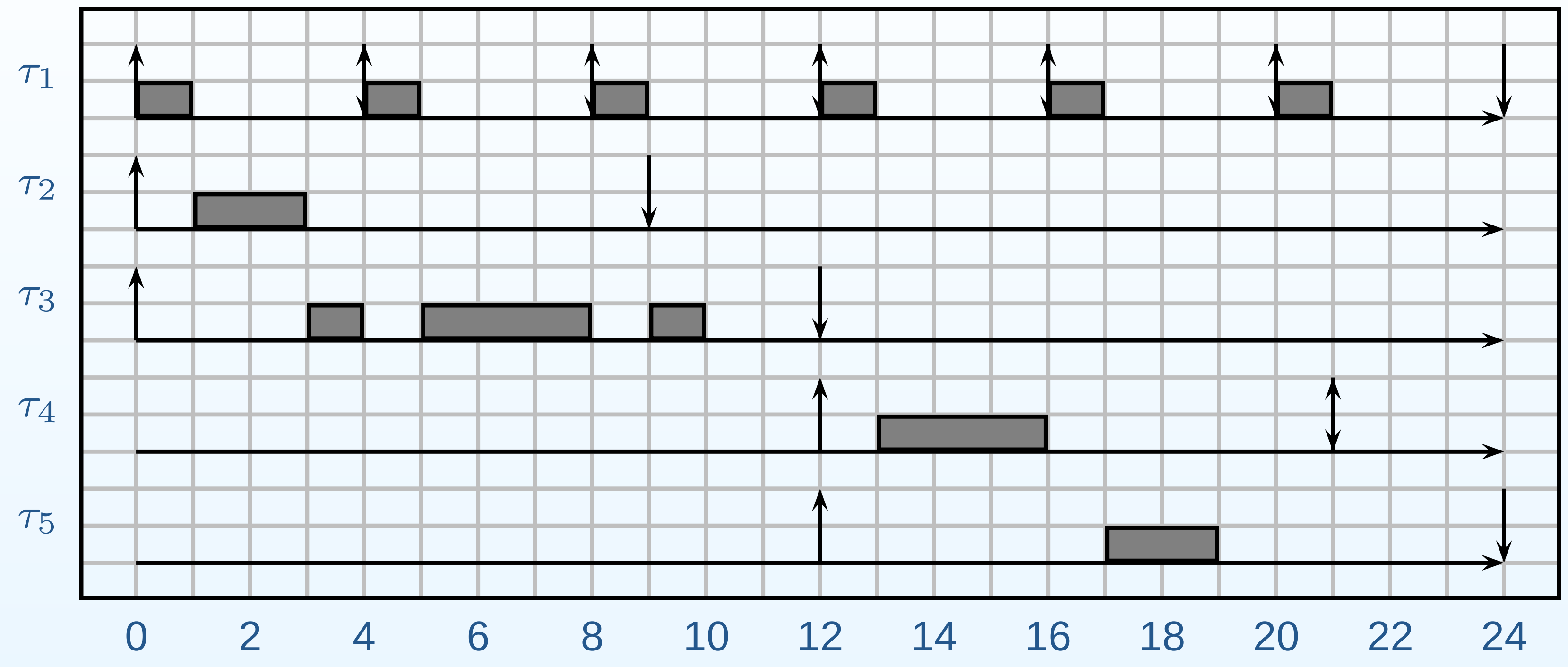
Example revised



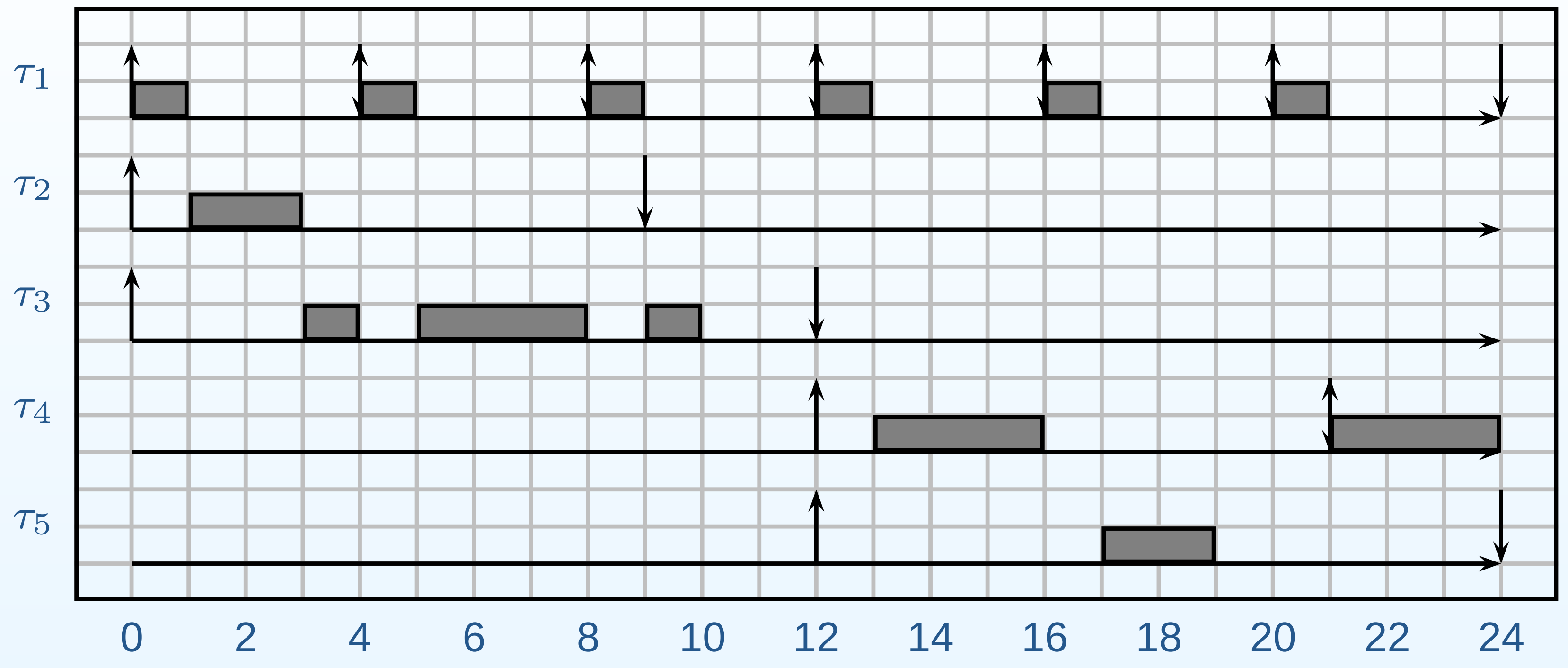
Example revised



Example revised



Example revised



Maximum transition delay

- In the worst-case the delay is equal to the length of the longest period among all tasks that leave the mode.
- Other possibilities;
  - Another simple assumption is to make the transition at the hyperperiod;
    - $\longrightarrow$ In fact, at the hyperperiod, all task have completed;
    - $\longrightarrow$ however, the delay in this case may be larger;
  - Another possibility is to wait for the first idle time;
    - $\longrightarrow$ While the delay may be shorter in this case, it may be difficult to calculate it a priori.

Mode Manager

- Two more global variables are needed:
- `transition_time` is the time after which the tasks that enter the mode can be activated;
- `transitory` is a boolean variable that is true when the system is changing from one mode to the other;
- We group these variables in a structure, and protect the structure with a mutex;

```
struct _mode_struct {
  int curr_mode;
  int old_mode;
  int transitory;
  TIME transition_time;
  pthread_mutex_t m;
} ms;
```

Example of code for the mode manager

```
TASK mode_manager (void *arg) {
  //initialization
  while(1) {
    pthread_mutex_lock(&ms.m);
    if (ms.transitory && sys_gettime(&mytime) >= ms.transition_time) {
      for (i=0; i<N; i++) if (mode[i][ms.curr_mode]) task_activate(pid[i]);
      ms.transitory = 0;
      pthread_mutex_unlock(&ms.m);
    }
    else if (isModeChanged(ms.curr_mode) {
      ms.old_mode = ms.curr_mode; ms.curr_mode = getNewMode(ms.curr_mode);
      ms.transitory_time=getTransitoryTime(ms.old_mode, ms.curr_mode);
      ms.transitory=true;
      pthread_mutex_unlock(&ms.m);
      transition(ms.old_mode, ms.curr_mode);
    }
    else pthread_mutex_unlock(&ms.m);
  }
  task_endcycle();
}
```

Considerations

- In the previous example of code, we suppose that the mode manager task is a periodic task;
    - The mode manager must execute at high priority;
    - If it executes at low priority, the transition delay could increase due to the response time of the mode manager task;
    - Additional delay is due to the period of the mode manager task; The period must be quite small, otherwise the delay increases too much.
- The mode manager can also be an aperiodic task;
    - The mode manager task is activate only when the condition happens, from an external interrupt, of from one of the other tasks;
    - In this case, it is necessary to understand which is the maximum frequency of a mode change (minimum interarrival time);
    - Again, the priority of the mode manager task should be as high as it is possible.