Design of Real-Time Software

Part 1: Real-time Scheduling

Onno van Roosmalen
O.S.vanRoosmalen@chello.nl

Version July 2016

Content

- Introduction
- Static priority scheduling
  - RM scheduling without resource contention
    - Periodic tasks
    - Sporadic tasks
    - Aperiodic tasks
  - RM scheduling with resource contention
  - DM scheduling

Content

- Introduction

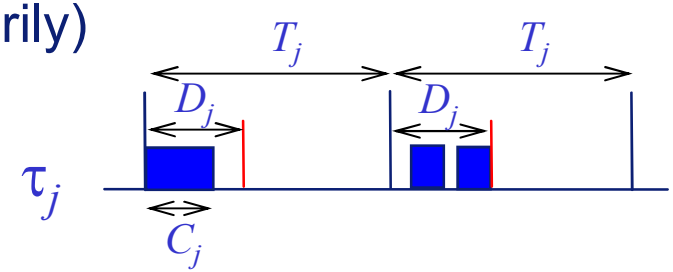
Recording external events

- The controlling software samples the external state with a certain frequency (time-driven)
  - typically: continuous external variables
    - temperature, humidity
    - water level
    - pressure
- The controlling software is informed about the external events (event-driven)
  - typically: changes in discrete environmental variables
    - a packet passes a sensor
    - a hazardous condition occurs
    - a frame must be placed on the screen
- The choice between these two approaches is made during design of the controlling system (SW and HW)

Tasks

- Task definition (again): *a sequence of actions that must be carried out to generate a response to an external event or a time event*

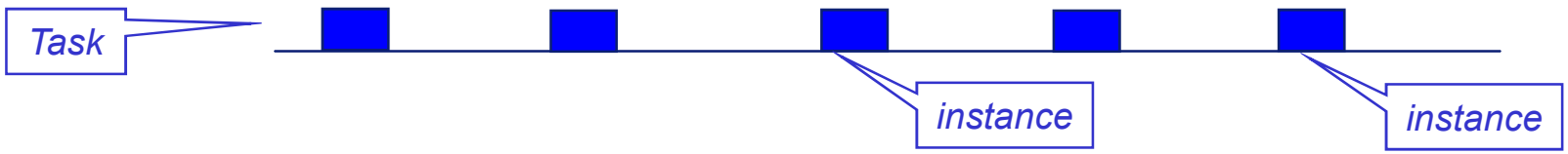
- Task types *(note: all cyclic):*
  - Periodic (time driven; released with period exactly T)
  - Sparse (event driven; released with distance larger than a certain T)
  - Aperiodic (event driven; released arbitrarily)



- A task has
  - a name/index (the j th task) τ_j
  - a (worst case) execution time (WCET) C_j
  - a period (periodic and sporadic tasks) T_j
  - a deadline within the period D_j

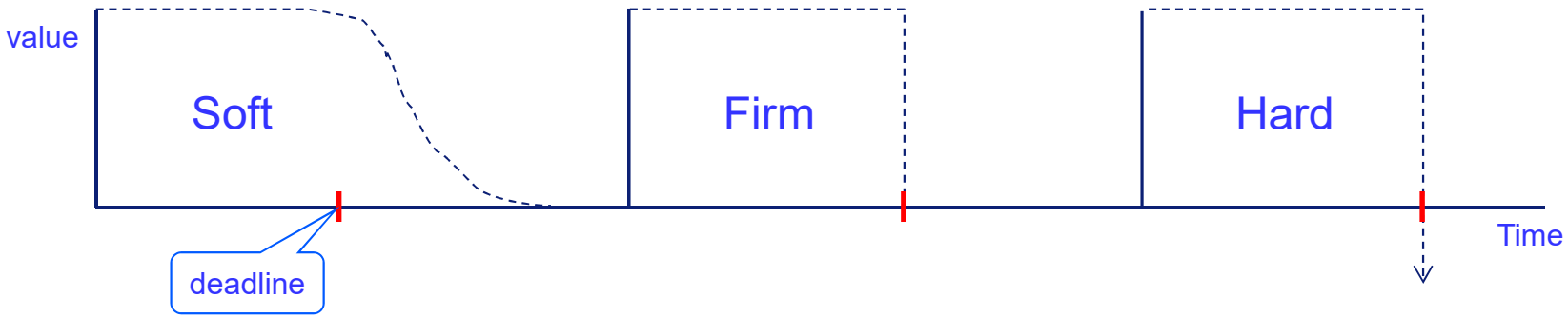
Types of tasks

- Software is most often a mix of time-driven and event driven tasks
  - depends on the choices on measuring environmental state
  - depends on the choice of HW implementation of sensors
- Both time-driven and event-driven tasks:
  - Tasks are cyclic
    - events recur
  - Cycles are started by
    - a clock (time driven)
    - an external event occurrence (event driven)
  - Each execution of the action-sequence is an *instance* of a task



Value after deadline

- **Soft**
  - A response is still valuable after the deadline, but value decreases steadily after that.
    - Example: interaction with human users. People get impatient.
- **Firm**
  - A response has no value after the deadline.
    - Example: a video frame that cannot be shown in time can be skipped.
- **Hard**
  - Damage is done if a response does not come in time.



Examples

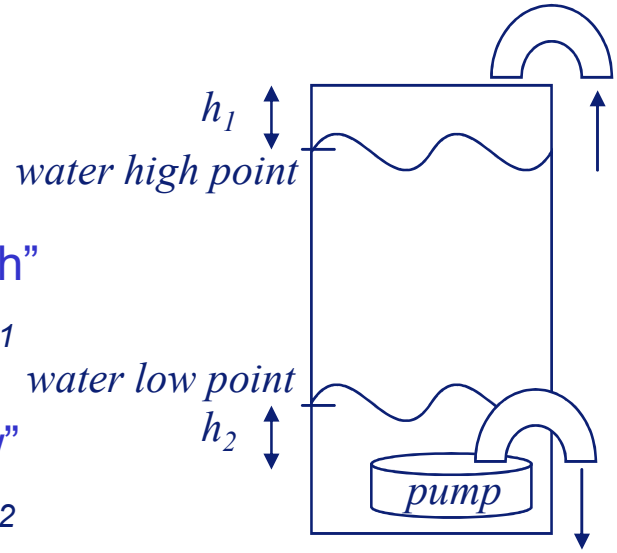
- Dependable real-time systems
  - High cost of failure
    - Possibly loss of life on failure
  - Guaranteed dependability (especially timeliness)
  - Example: Industrial control

- High performance real-time systems
  - Low probability of failure
    - Constant quality of service
  - High regularity in performance
  - Example: Consumer electronics

Example: water vessel

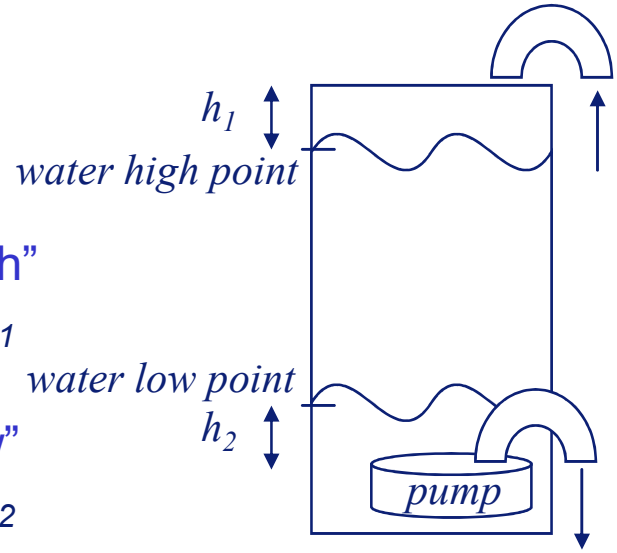
- Requirements:
  - Vessel may not overflow
  - Pump may not run dry
- Properties
  - Water level has a maximum rate of change (up and down)
  - Sensor positions are chosen

- Derived SW requirements
  - Response deadline can be calculated
    - The state "pump is off and the water is high" may not persist longer than a time span td_1
      - **only then risk of overflow**
    - The state "pump is on and the water is low" may not persist longer than a time span td_2
      - **only then risk of underflow**



Example: water vessel

- Requirements:
  - Vessel may not overflow
  - Pump may not run dry
- Properties
  - Water level has a maximum rate of change (up and down)
  - Sensor positions are chosen
- Derived SW requirements
  - Response deadline can be calculated
    - The state "pump is off and the water is high" may not persist longer than a time span td_1
      - **only then risk of overflow**
    - The state "pump is on and the water is low" may not persist longer than a time span td_2
      - **only then risk of underflow**



Requirements on tasks

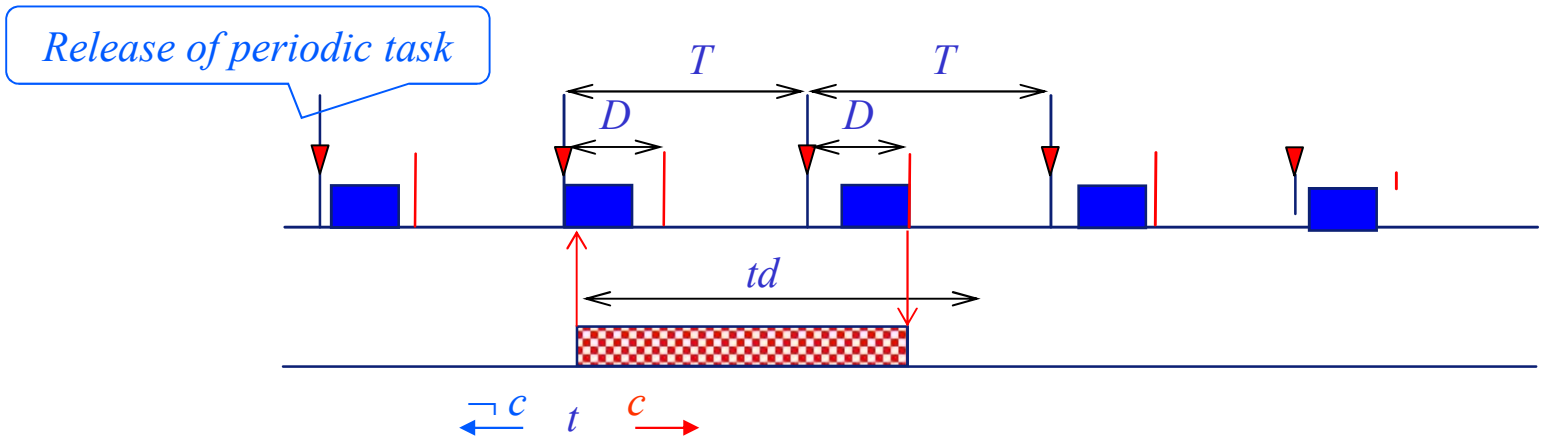
- Detection criterion: situations must be detected in which
  - the water is above (gets above) the high point
  - the water is below (gets below) the low point

- Response criterion: after detection the system must respond before
  - the water is above the high point for a time td_1 or longer
  - the water is below the low point for a time td_2 or longer

- These requirements can be met by a polling task (!)
  - Requirements like: “the system must respond when the water reaches the high point” can not be met by a polling task
    - Why?

Polling task

- Critical state c should not exist longer than a time span td without response that cancels this critical state
  - critical state c: water above/below sensor and pump off/on
- Periodic task is released with period T and satisfies deadline D within this period.
  - If water at low sensor: Task stops pump
  - If water at high sensor: Task starts pump
- Schedulability conditions (see diagram): $T + D < td$
  - If the task may finish anywhere within the period ($D = T$) : $2.T < td$



Predictable computation time

- Make sure that the task can finish within $D\ (\leq T\ )$
  - Worst case execution times C (WCET) should be calculable $(C \leq D\ )$
  - The load on the processor resource is not more than C per period T
- The predictability of this is endangered by
  - anomalies of the hardware and system software
    - use of DMA that locks the bus
    - cache behaviour
    - interrupts
    - memory management
  - constraints
    - finding required resources locked by other tasks
    - precedence constraints
  - absence of transparency in high level languages
    - dynamic variables, garbage collection (memory management)
    - repetition, recursion

Example: Video device

- Device characteristics
  - audio/video: perception is main concern
  - high volume turnover: cost constraints
- Real-time
  - high quality audio and video pose stringent real-time requirements
  - regularity is more important than latency
- Quality of service
  - “collective effect of service performances that determine the degree of satisfaction of the user of that service

Average case (⇐ concerns are Quality+Cost)

resource allocation is more important than

Worst case (⇐ concerns are Latency + Predictability)

Average case: why?

- Average case and worst case are far apart
  - worst case leads to over-dimensioning of resources
- High volumes: low bill of material is desired
- Also low power conflicts with over-dimensioning

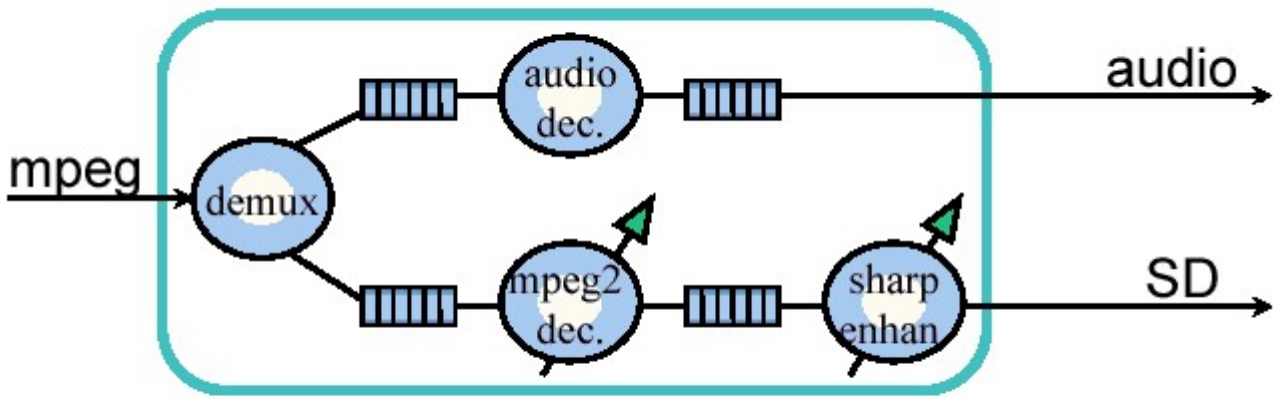
Average case: how?

- Lesser picture quality often better than temporal incorrectness
  - deadline misses may lead to wrong picture
  - deadline misses tend to come in bursts
- Reducing quality of manipulating the video stream may reduce load in high load/overload situations
  - Quality fluctuations are perceived as non-quality
  - Regularly recurring errors are very visible and annoying
- Only video specialists can make trade-off

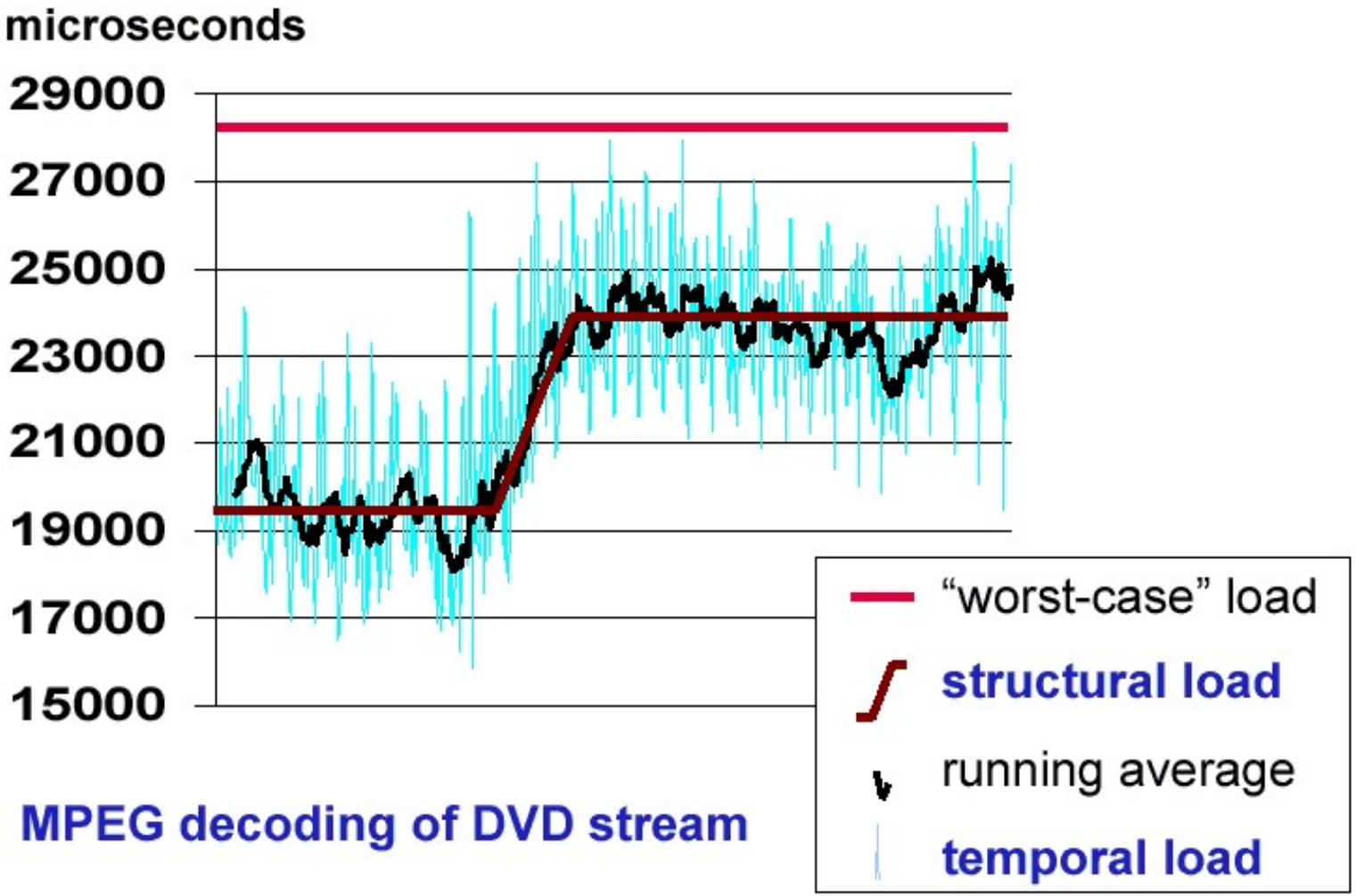
Quality of Service

Example of processing pipeline

- Various periodic tasks in a pipeline
- Some tasks can perform processing at various quality levels (green arrows)
- Buffers help to balance the load over time
- Periods can be different (compare audio with video decoding)



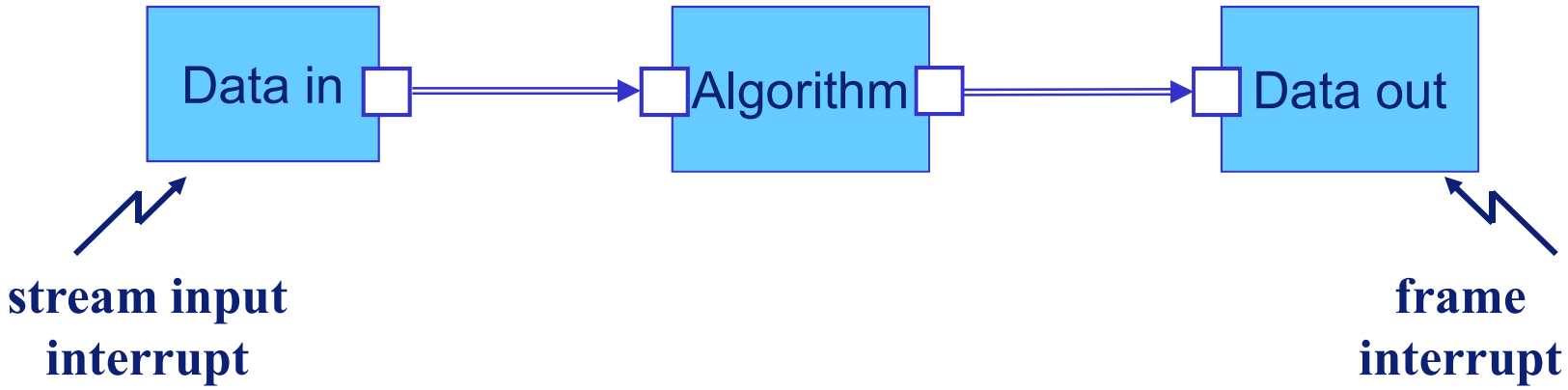
Structural and temporal load changes



MPEG decoding of DVD stream

Processing pipeline

$P_{Algorithm}$ = *while true do*
receive(i, frame);
process(frame);
send(o, frame);
od



Acceptable WCET's

$P_{Algorithm}$ = *while true do*

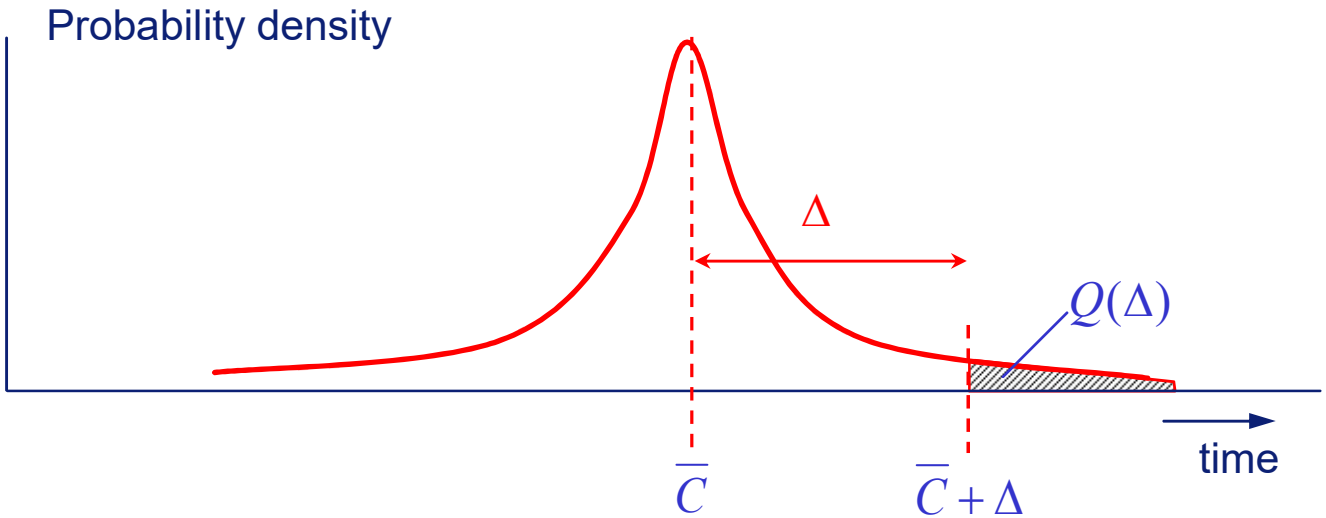
receive(i, frame);

process(frame);

send(o, frame);

od

- Average execution time to process frame $\overline{C}$
- Period T is given by frame rate
- Assume Gaussian probability distribution of execution times: $P(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(t-\overline{C})^2}{2\sigma^2}}$
- Probability of exceeding $C = \overline{C} + \Delta$: $Q(\Delta)$ Favors large Δ

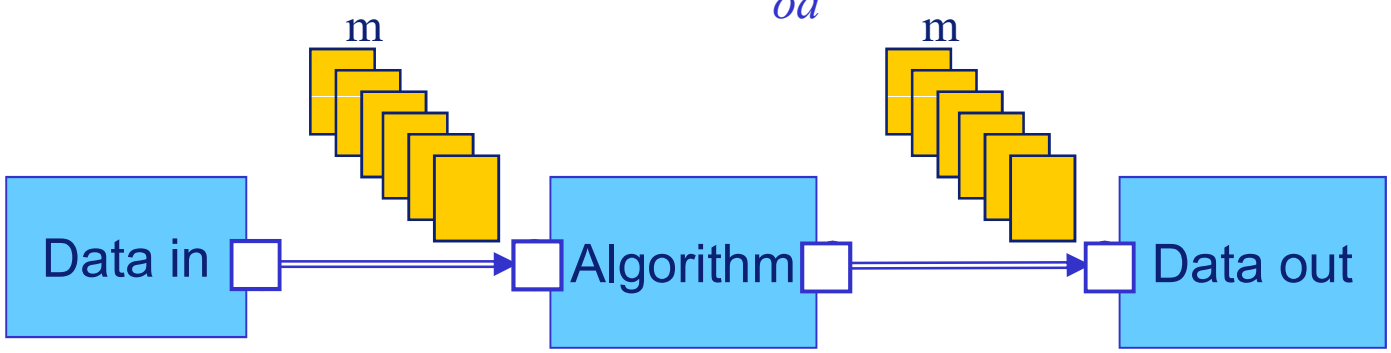


Buffering

Failures:

- Input buffer overflow
- Output buffer underflow

$P_{Algorithm}$ = *while true do*
do m times
receive(i, frame);
process(frame);
send(o, frame);
od
od



stream input interrupt

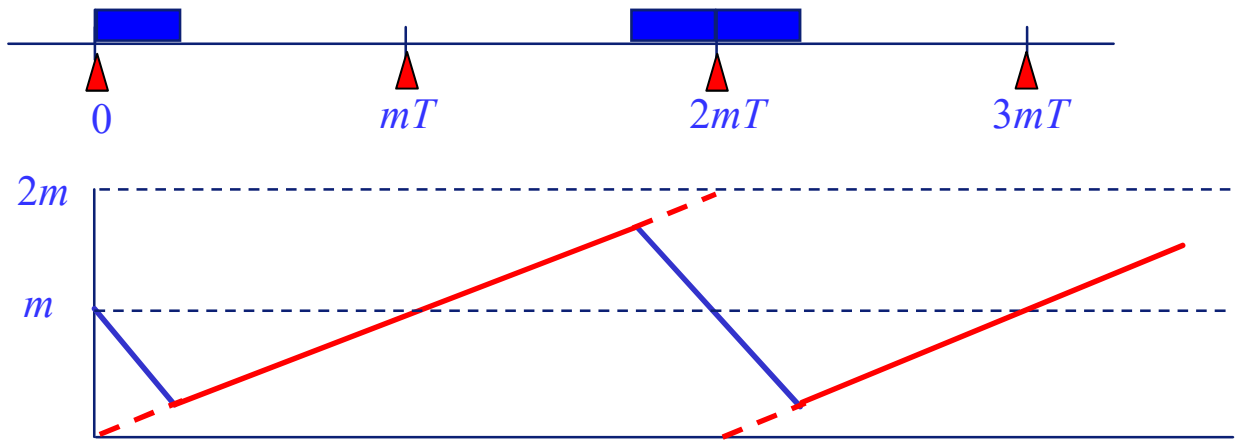
frame interrupt

- Buffer size ~ 2*m*
- Period : *mT*
- Latency : ~ 2*mT*

input m frames;
(process m frames // input m frames);

Buffer size

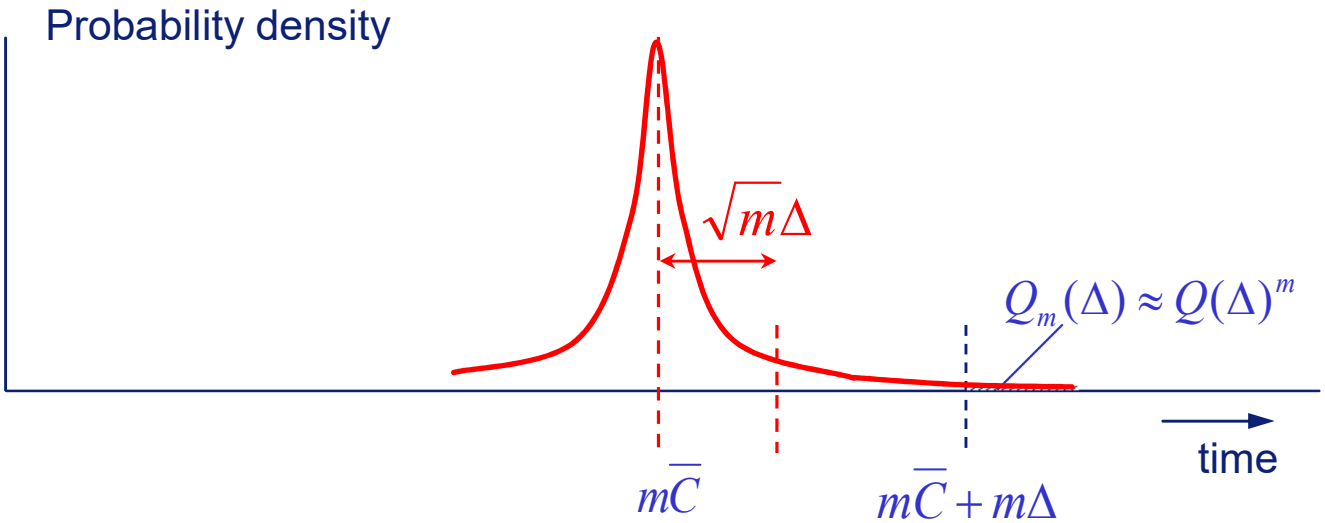
- Period of task instance is now m times the original period: mT
- Worst case fluctuations in actual start time of periodic process: $\sim mT$
- This requires both filled and empty buffer space of about m
- Total buffer size should be about $2m$



Processing m frames per task instance

```
P_Algorithm = while true do
                do m times
                    receive(i, frame);
                    process(frame);
                    send(o, frame);
                od
            od
```

- Probability of exceeding execution time $m(\overline{C}+\Delta)$: $Q_m(\Delta) \approx Q(\Delta)^m$
- With buffer 1 → m : failure probability e.g. 0.1 → 0.1^m



Correct Assumptions?

- Not statistical
  - Increased execution times come in bursts
  - Therefore: execution times are not Gaussian distributed
- Other measures
  - Different functions (e.g. audio decoding and sharpness enhancing) may be independent
    - Instead of m cycles of a single process combine m cycles of different processes

➡ We have seen qualitative arguments to illustrate the issues

Guaranteeing Deadlines

- General approach:
  - Make hypothesis on behavior of environment.
    - E.g. water vessel:
      - The influx of water into the vessel will be less then x liter/second
    - E.g. video processing:
      - The frame processing time will be less than x milliseconds per frame
  - The correctness of the hypothesis may have a calculable probability
    - E.g. water vessel:
      - The influx on average may exceeds this value once in every 100 years.
    - E.g. video processing:
      - The probability distribution for frame processing times is $f(t)$
  - Within the hypothesis meeting deadlines is guaranteed
  - Failures that fall outside the hypothesis have a certain probability
    - E.g. water vessel:
      - The possibility of overflow is limited to once in every 100 years.
    - E.g. video processing:
      - The probability for missing a frame is 1 in 10^9

Scheduling theory

- Focus is on how we guarantee meeting deadlines

- Many conclusions are qualitative as well
  - What scheduling approach is optimal
  - Optimality depends on the boundary conditions
  - One can be optimal according to many criteria

Content

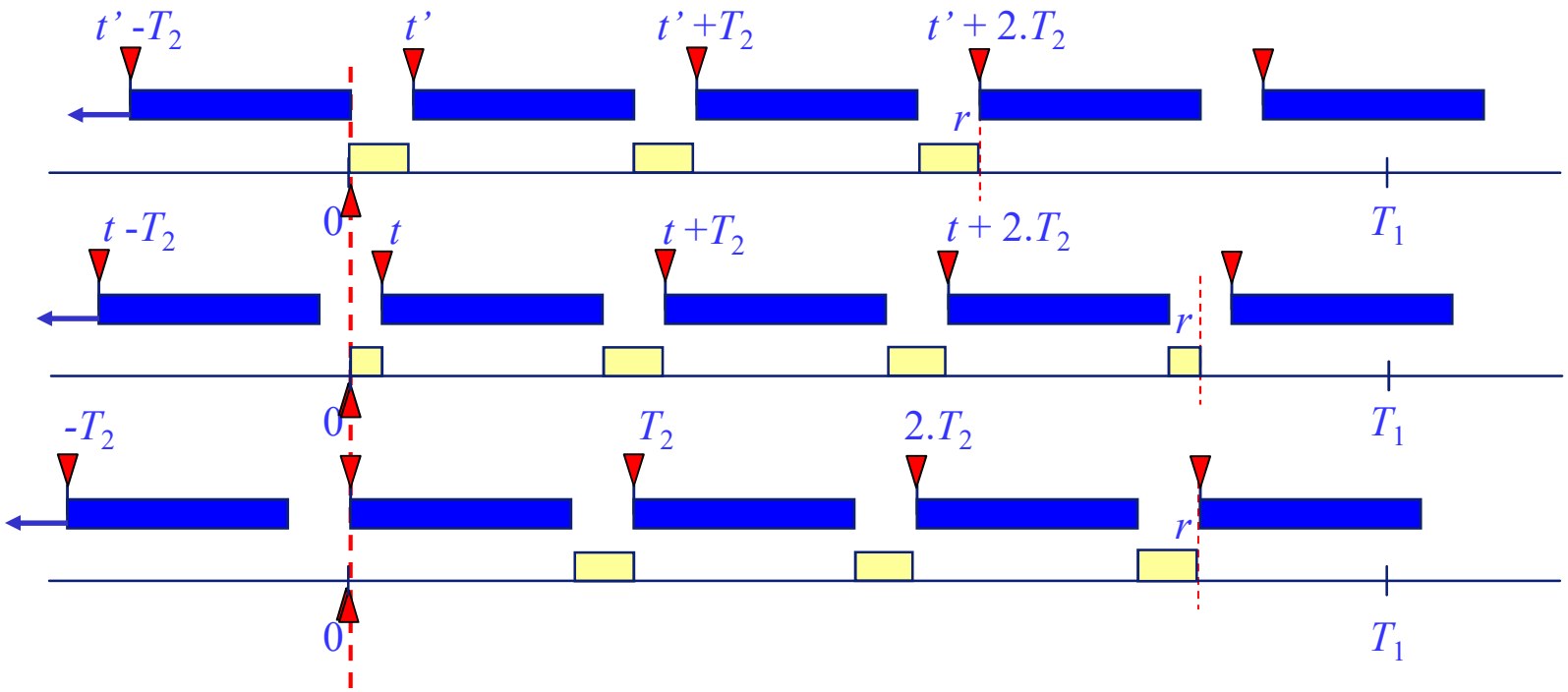
- Introduction
- Static priority scheduling
  - Rate Monotonic (RM) scheduling without resource contention
    - Periodic tasks

Rate-Monotonic Scheduling

- Rate monotonic=task priority proportional to rate (frequency)
- Basic assumptions:
  - One processor
  - Tasks are independent and periodically released (polling tasks)
  - Tasks have fixed individual priorities (τ_j has priority p_j)
  - Preemption allowed
    - High priority tasks may preempt low priority tasks.
  - Tasks have fixed (maximal) execution duration (WCET$=C_j$)
- For the moment
  - The processor is the only shared resource
  - Deadlines within periods are equal to periods ($D_j = T_j$)
- Terminology:
  - “There is a correct schedule” or “There is a feasible schedule” means “There is a schedule that satisfies all timing constraints”

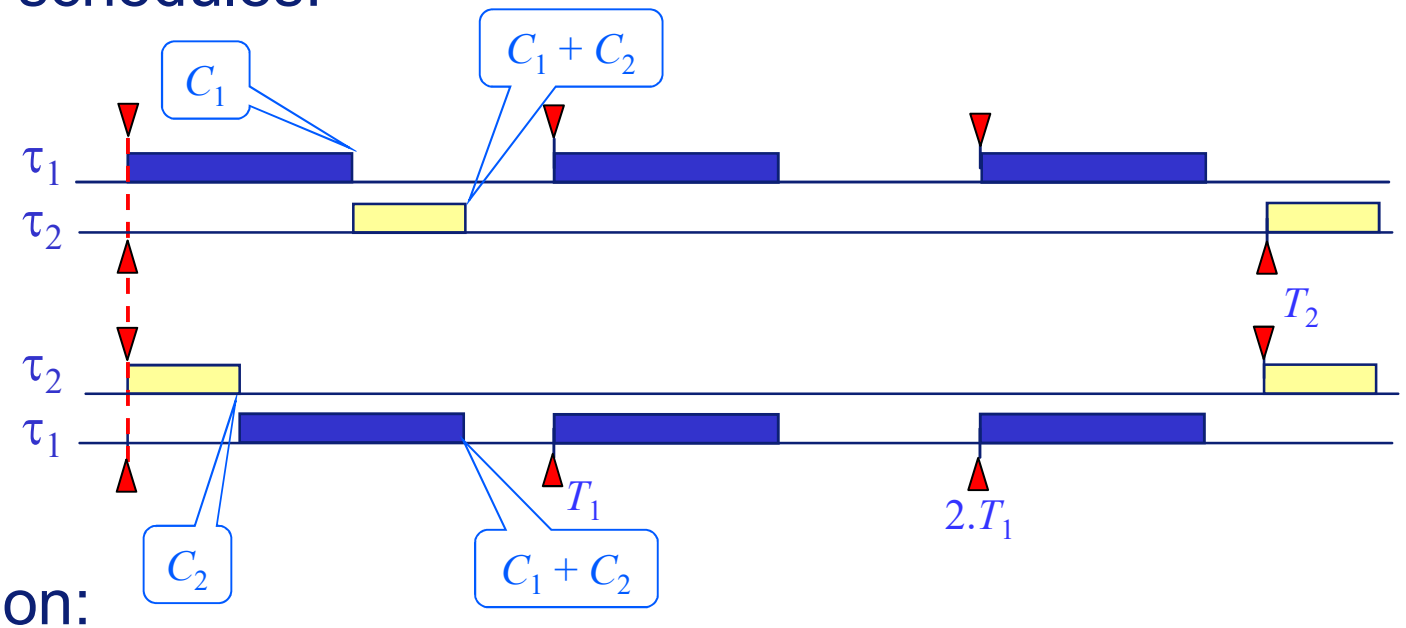
Critical Instant

- Critical instant of a task (fixed priority scheduling only!!) :
  - Definition: the situation that gives the largest response time
- Lemma:
  - The worst case response time of a task occurs when it is released at the same time with all higher priority tasks
- Example: shift blue task until critical instant is reached



Example

- Consider two tasks $\tau_1 = < C_1, T_1, p_1 >$ and $\tau_2 = < C_2, T_2, p_2 >$
- Assume:
  - $T_1 < T_2$
  - $C_1 + C_2 \leq T_1$
- Possible schedules:



- Conclusion:
  - If $C_1 + C_2 \leq T_1$ any priority assignment will lead to a correct schedule

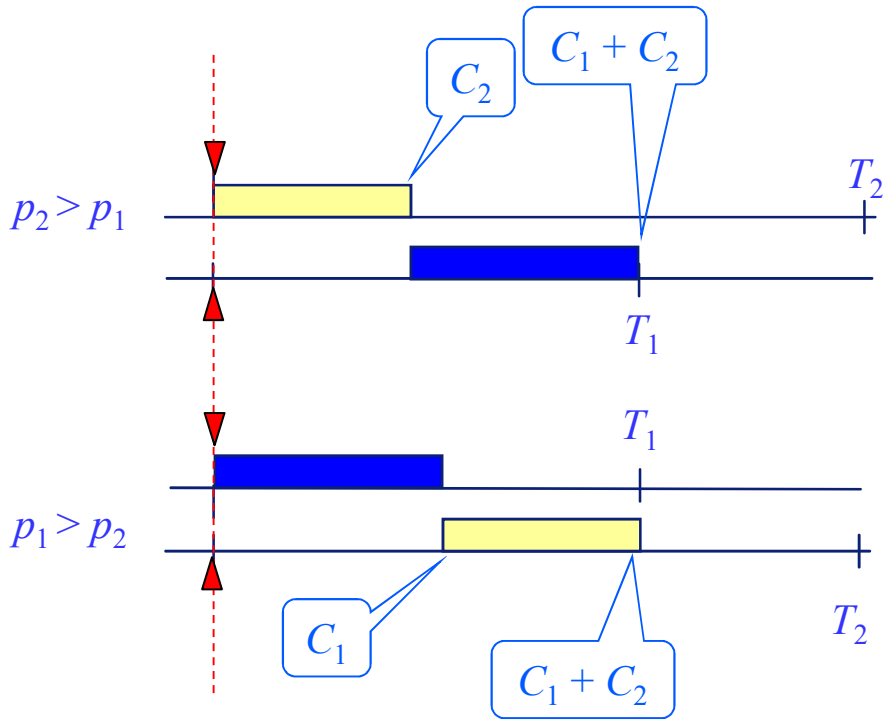
Exercise: arbitrary priority assignments

- Task set:
  - τ_1 : $C_1 = 2$ $T_1 = 6$
  - τ_2 : $C_2 = 2$ $T_2 = 10$
  - τ_3 : $C_3 = 4$ $T_3 = 12$

- Draw the six schedules for each of the possible priority assignments
- Determine the worst case response time for each task in each case and check if it is smaller than the period (=deadline) for that task.

Optimality of RM Priorities

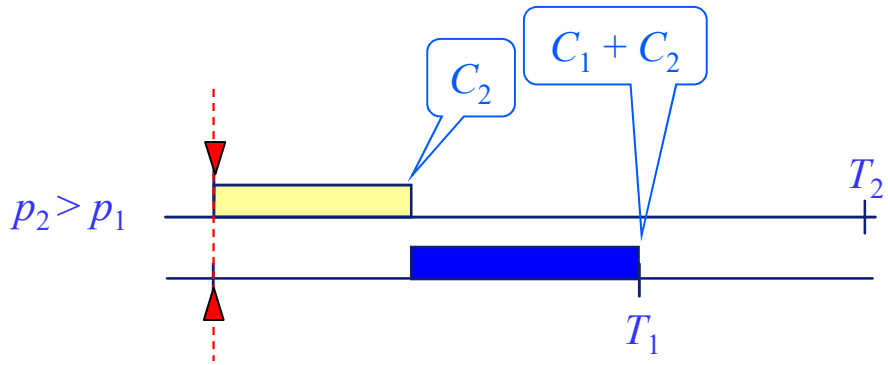
- Rate monotonic priority assignment
  - highest rate has highest priority
- If a priority assignment leads to a feasible schedule then the RM assignment leads to a feasible schedule as well.
- Example (two tasks):



Schedulability Condition: 2 Tasks

- Assume one period the larger of the two: $T_1 < T_2$
- We can have two possible priority assignments
  - non-RM: $p_2 > p_1$
  - RM: $p_1 > p_2$

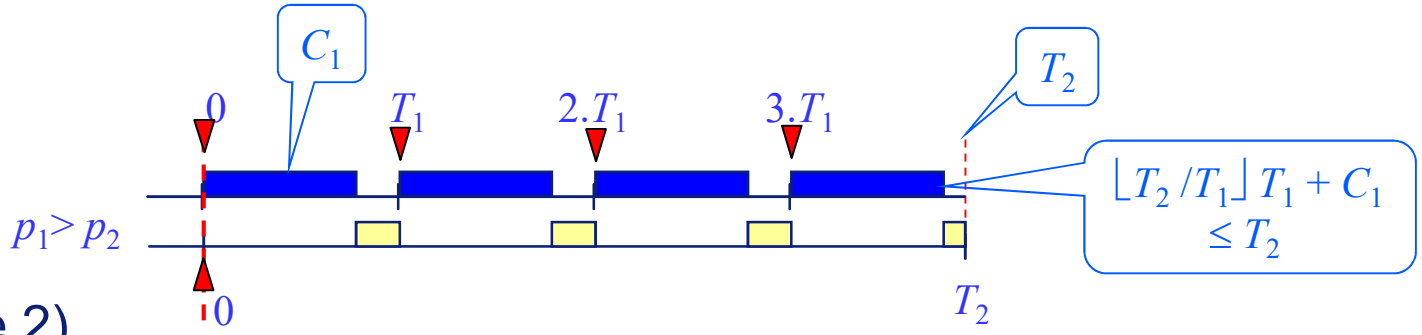
- non-RM:
  - At critical instant both tasks should finish in time:



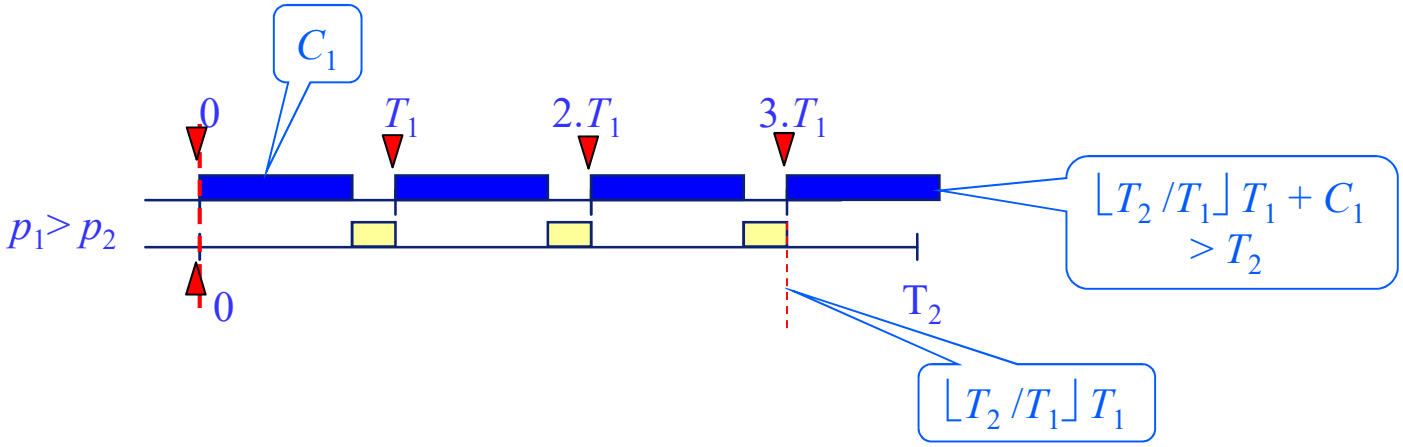
Schedulable iff $C_1 + C_2 \leq T_1$

Schedulability Condition: 2 Tasks

- RM:
  - More difficult case because τ_2 can be preempted by τ_1
  - Two situations depending on where period T_2 ends
  - RM: Case 1)

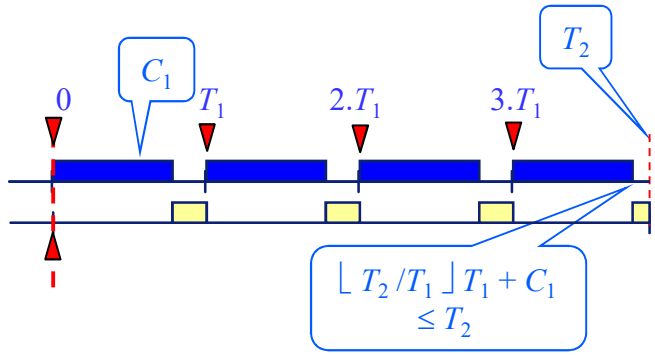


  - RM: Case 2)



RM: Case 1

- Assumption: $C_1 \le T_2 - \lfloor T_2 / T_1 \rfloor T_1$



- Time occupied by τ_1 in period T_2

  is $\lceil T_2 / T_1 \rceil C_1$

- Maximally available for τ_2 in period T_2

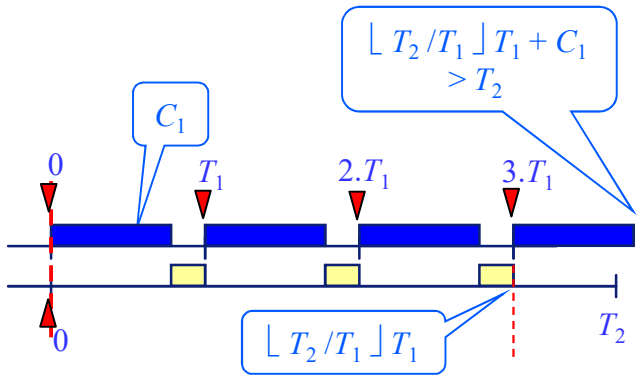
  is $T_2 - \lceil T_2 / T_1 \rceil C_1$

- Thus

Schedulable iff $C_2 \le T_2 - \lceil T_2 / T_1 \rceil C_1$

RM: Case 2

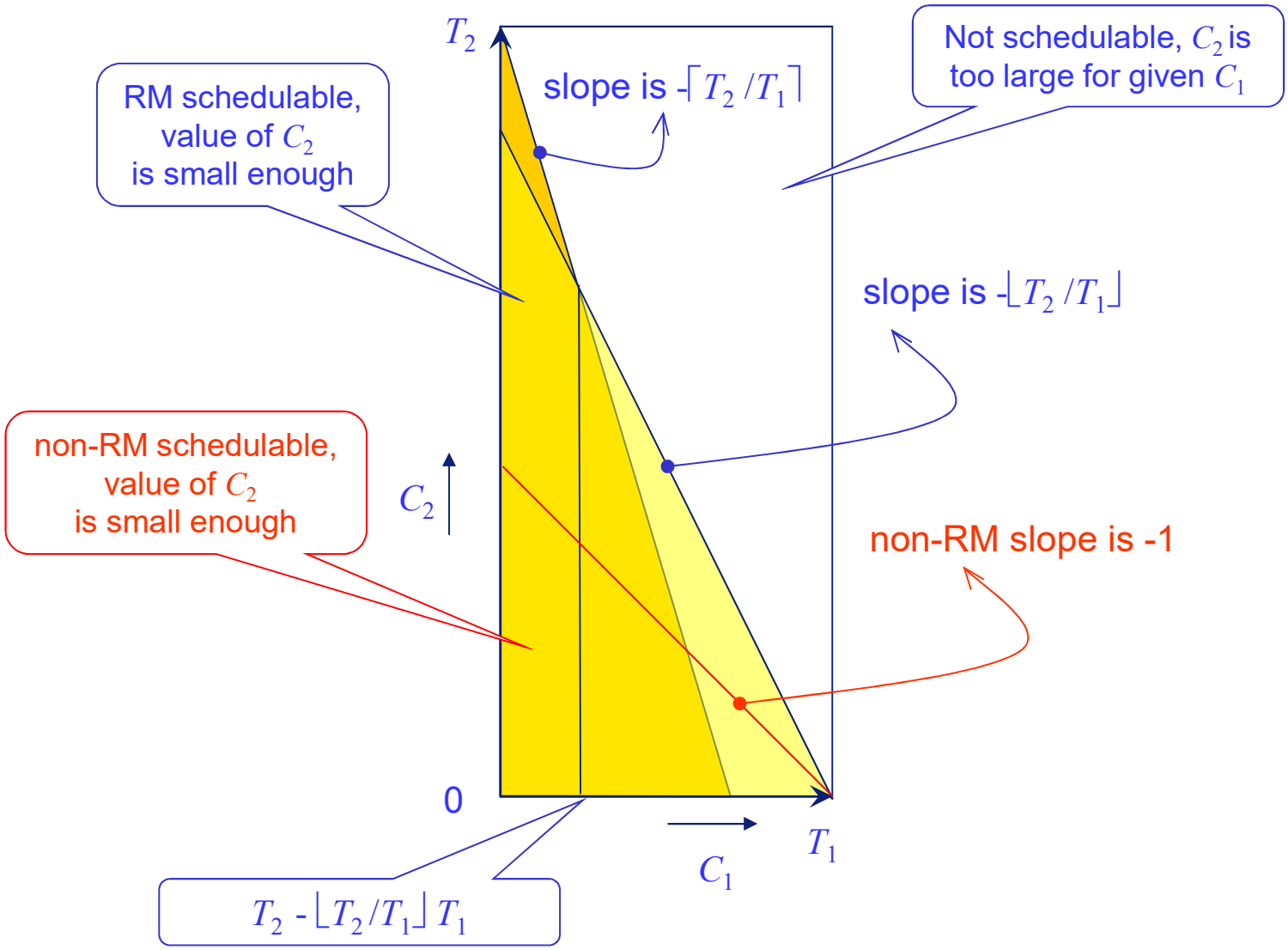
- Assumption: $C_1 > T_2 - \lfloor T_2/T_1 \rfloor T_1$



- Time occupied by τ_1 in period $\lfloor T_2/T_1 \rfloor T_1$ is $\lfloor T_2/T_1 \rfloor C_1$
- Maximally available for τ_2 in period $\lfloor T_2/T_1 \rfloor T_1$ is $\lfloor T_2/T_1 \rfloor T_1 - \lfloor T_2/T_1 \rfloor C_1$
- Thus

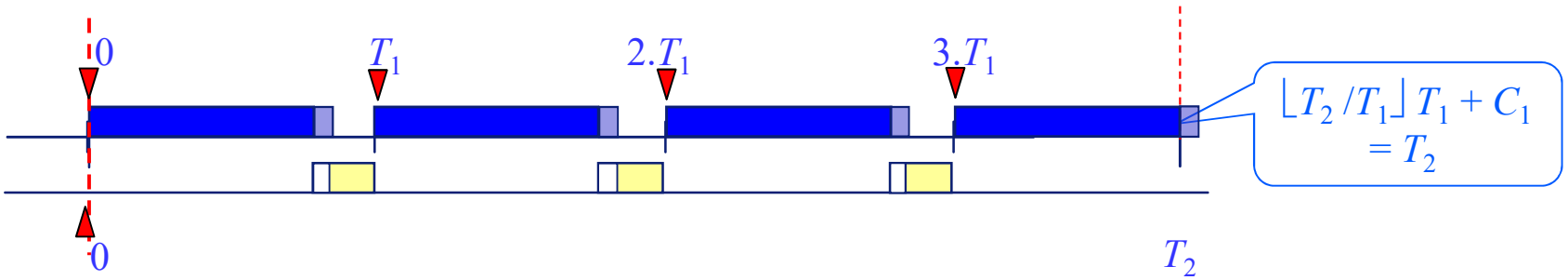
Schedulable iff $C_2 \leq \lfloor T_2/T_1 \rfloor T_1 - \lfloor T_2/T_1 \rfloor C_1$

Graphically

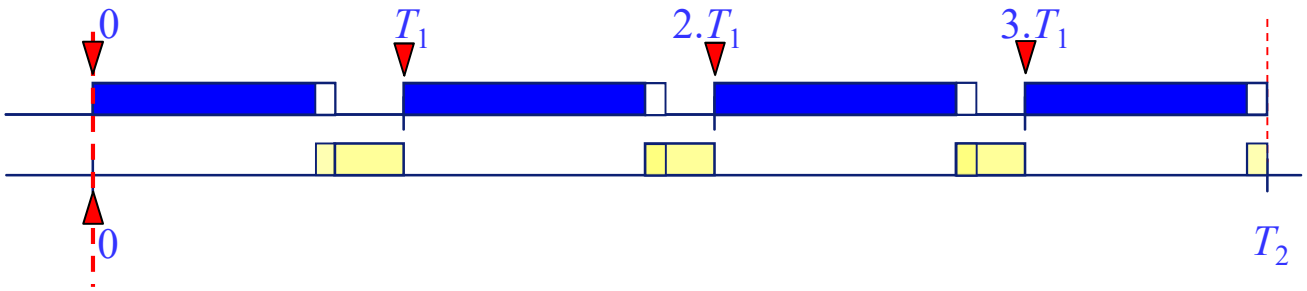


Slope Change

- Consider variations around magic point $C_1 = T_2 - \lfloor T_2 / T_1 \rfloor T_1$
  - Increasing C_1 by Δ can be compensated by decreasing C_2 by $\lfloor T_2 / T_1 \rfloor \Delta$



  - Decreasing C_1 by Δ can be compensated by increasing C_2 by $\lceil T_2 / T_1 \rceil \Delta$



Utilization Bound for Schedulability

- Consider two tasks $\tau_1 = < C_1, T_1, p_1 >$ and $\tau_2 = < C_2, T_2, p_2 >$
  - Rate monotonic priorities: p_i depends on T_i
  - C_i / T_i is worst case time-fraction that τ_i requires processor
  - $C_i / T_i > 1$ can not be guaranteed (no feasible schedule)

  C_i, T_i are all known: a precise schedulability criterion can be derived (see later)

  – If only the utilization is known: $$U = \sum_{i=1}^{m} \frac{C_i}{T_i}$$

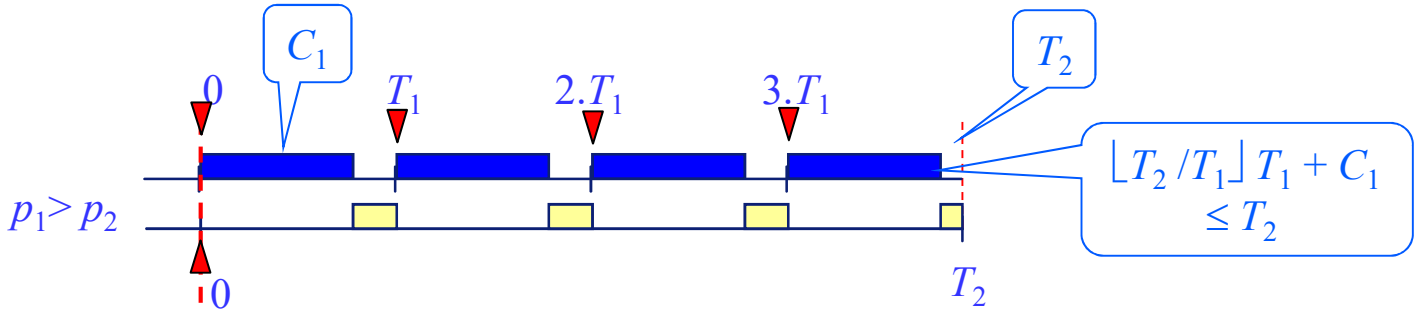
  We can look for a $U_{\min}$ for which
  - $U \leq U_{\min}$ the tasks always meet their deadlines
  - $U > U_{\min}$ the tasks can not be guaranteed to meet their deadlines

Determining $U_{\min}$ (two tasks)

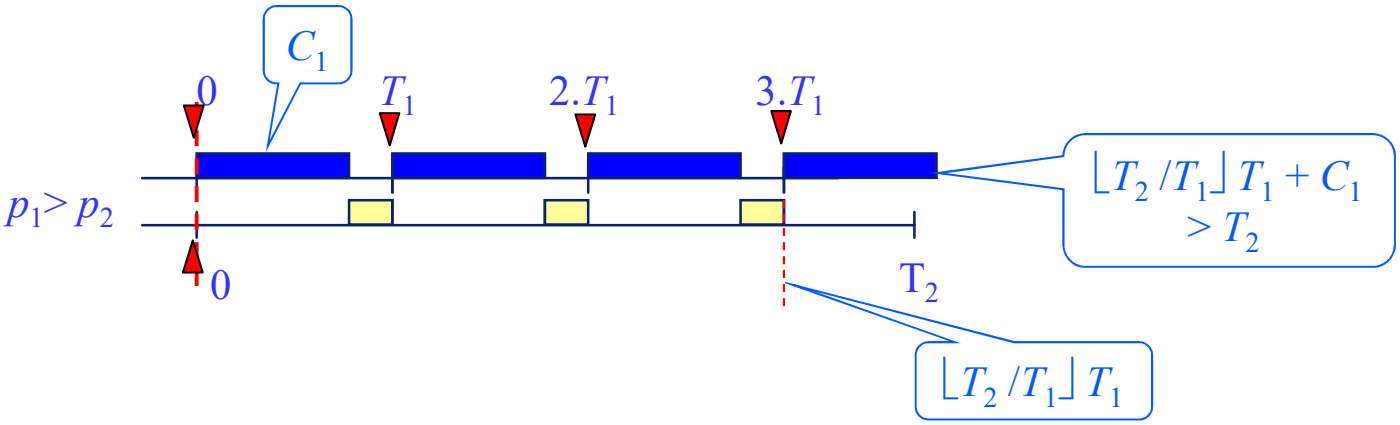
- Terminology:
  - Tasks are fully utilizing the processor if no increase of C_1 or C_2 possible
    - Critical instant
    - At critical instant: tasks fully occupy the processor until T_2
    - Schedulable but increasing either C_1 or C_2 breaks schedulability
  - Note: this does not imply $U_{full} = 1$
- Approach
  - Consider free variables in utilization: C_1, C_2, T_1, T_2
  - Consider largest feasible value for C_2 given C_1, T_1, T_2
    - C_2 can be eliminated giving full utilization $U_{full}(C_1, T_1, T_2)$
  - Find C_1 for which full utilization is minimal, this is $U_{\min}$
  - Then, for any C_1 and C_2 that yield a utilization below $U_{\min}$ schedule is feasible.
  - Minimize $U_{\min}$ with respect to T_1 and T_2 as well.

Determining U_{min}

- Consider same two cases as before
  - Case 1: maximal C_2 slopes at $-\lceil T_2/T_1 \rceil C_1$

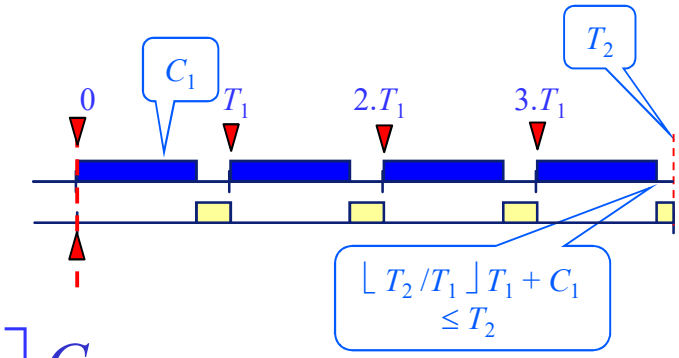


  - Case 2: maximal C_2 slopes at $-\lfloor T_2/T_1 \rfloor C_1$



Case 1

- Assumption: $C_1 \le T_2 - \lfloor T_2/T_1 \rfloor T_1$



- With full utilization: $C_2 = T_2 - \lceil T_2/T_1 \rceil C_1$
- Thus

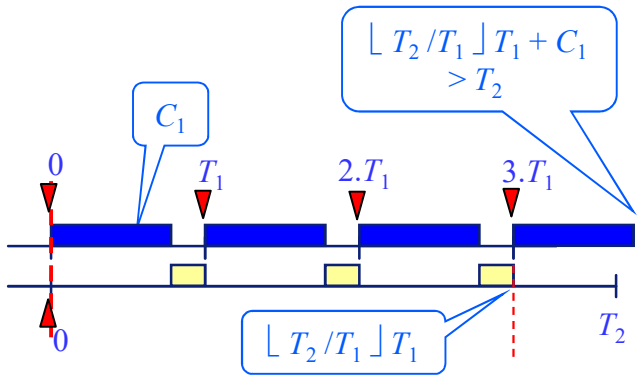
$$U_{full}^{(1)}\left(\frac{C_1}{T_1}\right) = \frac{C_1}{T_1} + \frac{T_2 - \left\lceil T_2/T_1 \right\rceil C_1}{T_2} = 1 + \frac{C_1}{T_1}\cdot\left(1 - \frac{T_1}{T_2}\left\lceil \frac{T_2}{T_1} \right\rceil\right)$$

constant

negative slope

Case 2

- Assumption: $C_1 > T_2 - \lfloor T_2/T_1 \rfloor T_1$



- With full utilization: $C_2 = \lfloor T_2/T_1 \rfloor T_1 - \lfloor T_2/T_1 \rfloor C_1$
- Thus

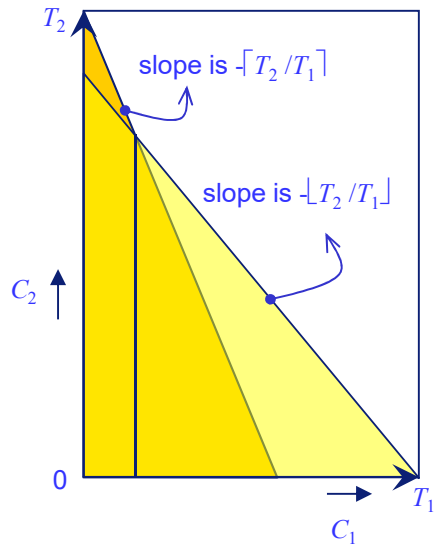
$$U_{full}^{(2)}\left(\frac{C_1}{T_1}\right) = \frac{C_1}{T_1} + \frac{(T_1 - C_1)\left\lfloor T_2/T_1 \right\rfloor}{T_2} = \frac{T_1}{T_2}\left\lfloor \frac{T_2}{T_1} \right\rfloor + \frac{C_1}{T_1}\cdot\left(1 - \frac{T_1}{T_2}\left\lfloor \frac{T_2}{T_1} \right\rfloor\right)$$

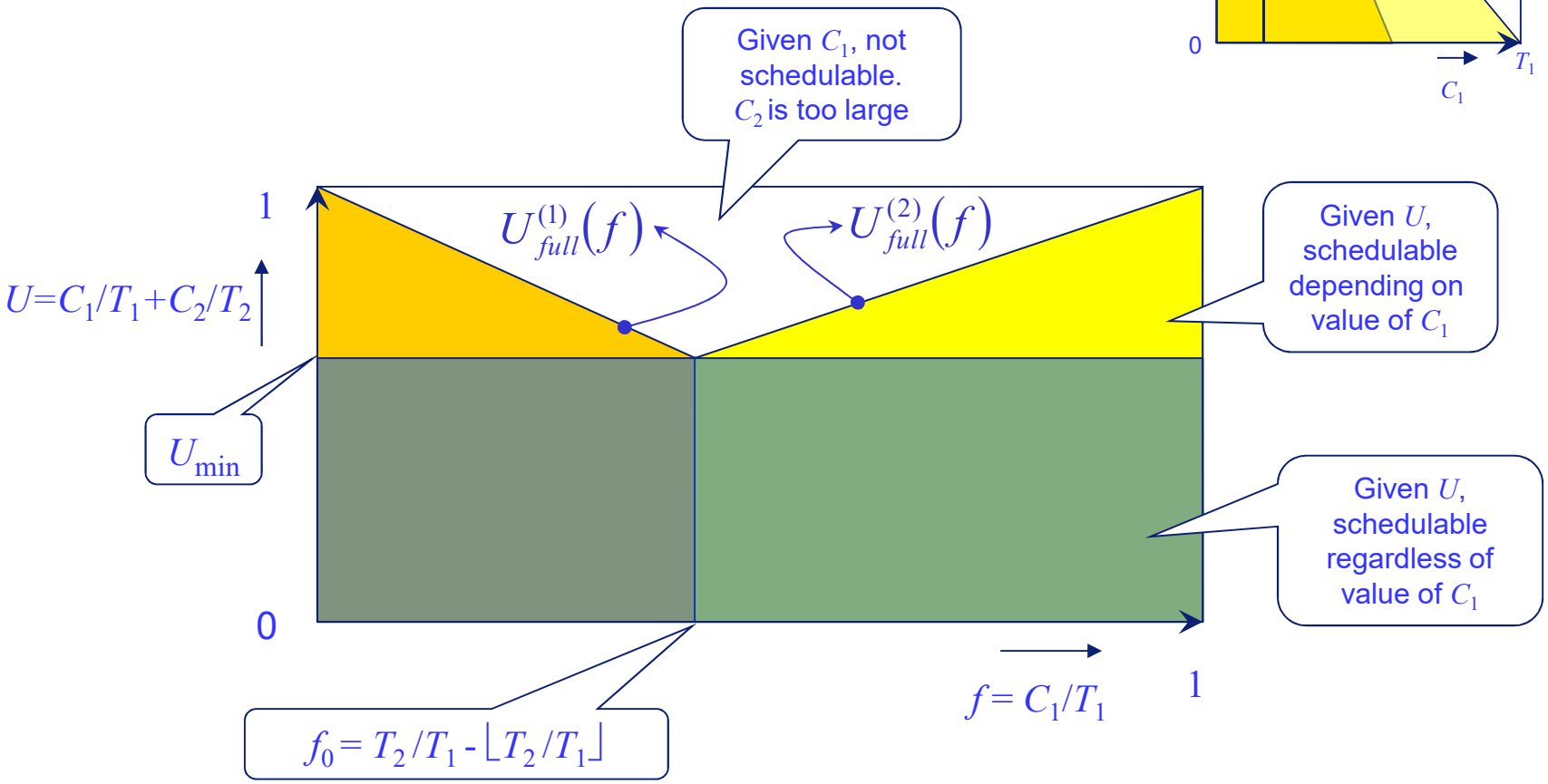
constant smaller 1

positive slope

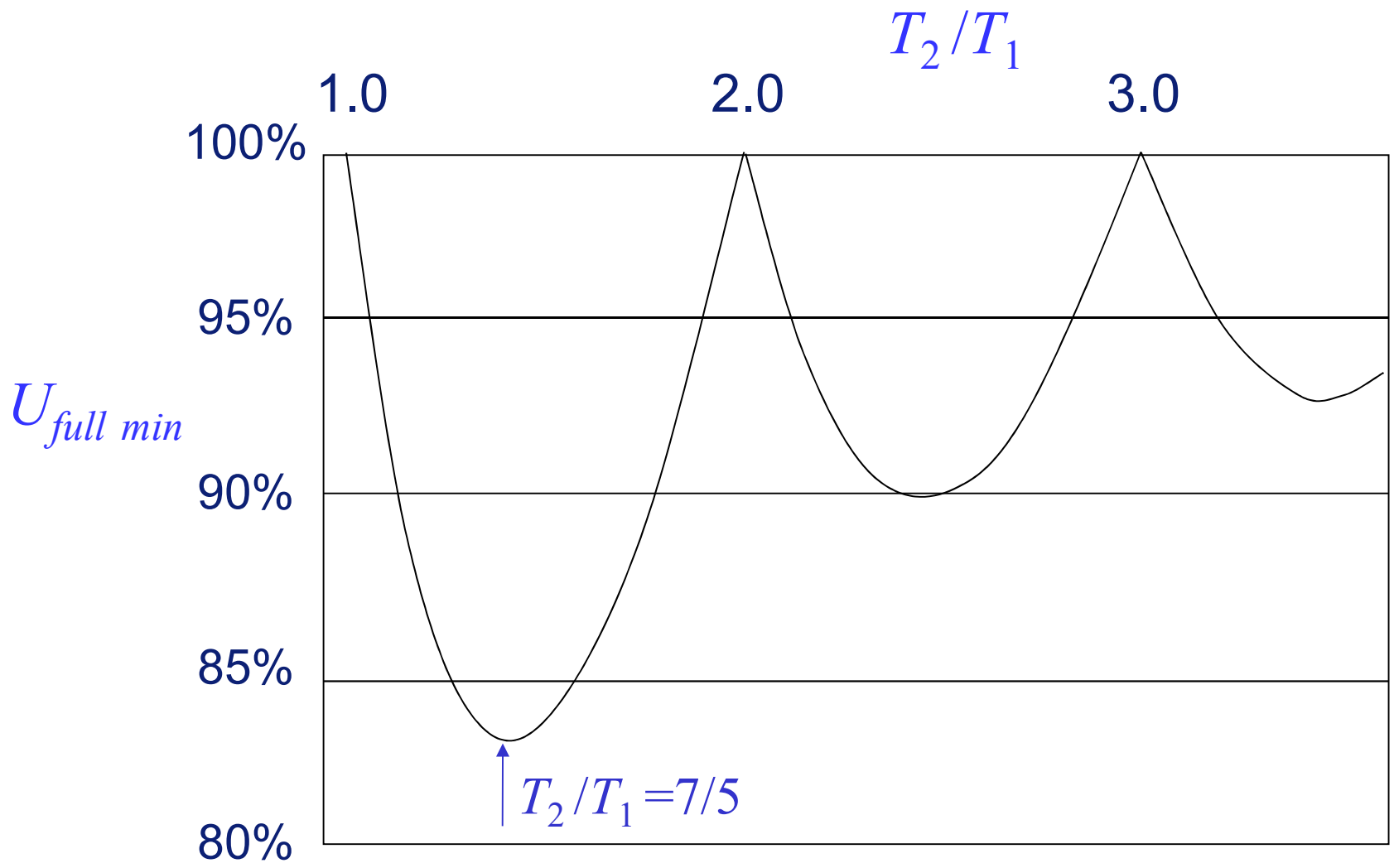
Graphically

- Introduce: $f = C_1/T_1 \quad (0 \le f \le 1)$





Minimal Value of Full Utilization varying T_2/T_1



Result

- Combine: $U_{\min} = U_{full}^{(1)}(f_0) = U_{full}^{(2)}(f_0) = 1 + f_0\left(1 - \frac{T_1}{T_2}\left\lceil \frac{T_2}{T_1} \right\rceil\right)$

- With: $f_0 = T_2/T_1 - \lfloor T_2/T_1 \rfloor$

- Introduce: $I = \left\lfloor \frac{T_2}{T_1} \right\rfloor \quad (1 \le I \le \infty)$

- Then: $T_2/T_1 = I + f_0 \quad (0 \le f_0 \le 1)$

- And thus:

$$\boxed{U_{\min}(f, I) = 1 - f(1-f)/(I+f)}$$

Further simplification

- We can even eliminate f and I
  - Minimal value of U_{min} when $I = 1$
  - Minimize for values of $0 \le f \le 1$

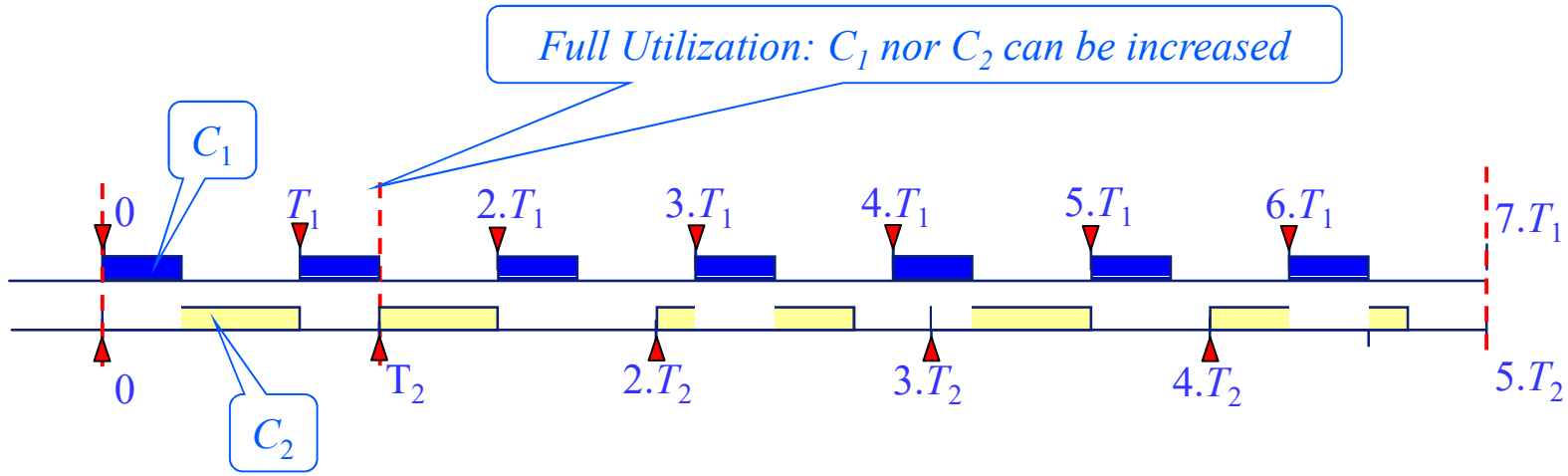
$$\left.\frac{dU_{\min}(f)}{df}\right|_{I=1} = 0 = \frac{d}{df} f\,(f-1)/(1+f) \longrightarrow f = \sqrt{2}-1$$

$$T_2/T_1 = \lfloor T_2/T_1 \rfloor + f = \sqrt{2}$$

$$\Rightarrow \boxed{U_{\min} = 2(\sqrt{2}-1) \cong 0.829}$$

Example: Full Utilization

- $\tau_1 : \; C_1 = 2 \; , \; T_1 = 5$
- $\tau_2 : \; C_2 = 3 \; , \; T_2 = 7$ (U_{full} minimal at $T_2 = \sqrt{2} \; . \; T_1 = 1.41 \; T_1$)



- Worst case response times:

  $\tau_1 : \; r_1 = 2$

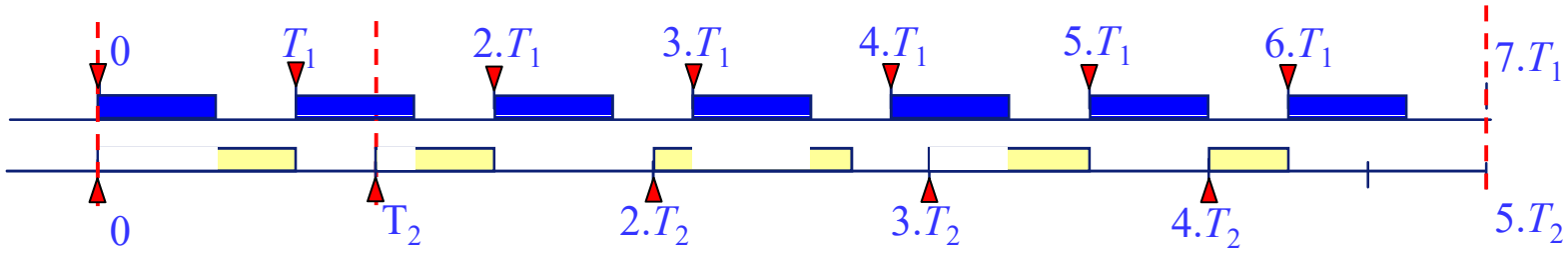
  $\tau_2 : \; r_2 = 5$

- Utilization

  $U_{full} = (2/5 + 3/7) \; = \; (29/35) = 0.828$

Example: Increase C_1 but keep Full Utilization

- $\tau_1 : \; C_1 = 2+1 = 3 \; , \; T_1 = 5$
- $\tau_2 : \; C_2 = 3-1 = 2 \; , \; T_2 = 7$



- Worst case response times:

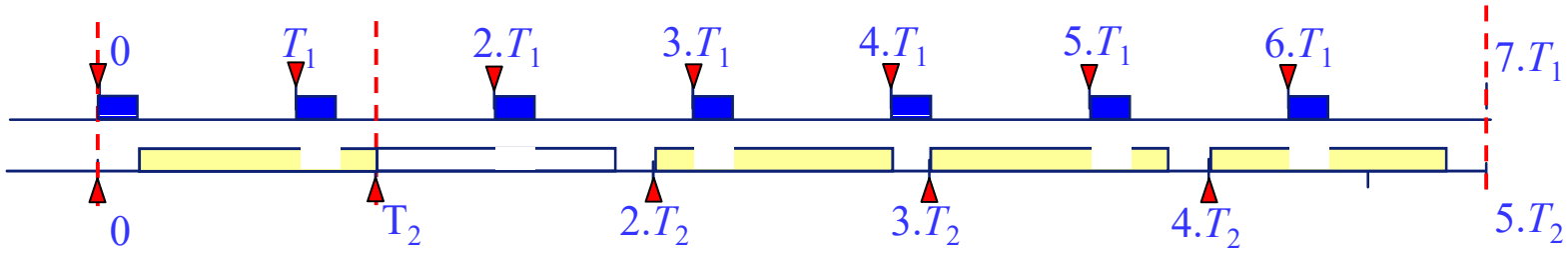
  $\tau_1 : \; r_1 = 3$

  $\tau_2 : \; r_2 = 5$
- Utilization

  $U_{full} = (3/5 + 2/7) \; = \; (31/35) = 0.89$

Example: Decrease C_1 but keep Full Utilization

- $\tau_1 : \; C_1 = 2 - 1 = 1 \; , \; T_1 = 5$
- $\tau_2 : \; C_2 = 3 + 2 = 5 \; , \; T_2 = 7$



- Worst case response times:

  $\tau_1 : \; r_1 = 1$

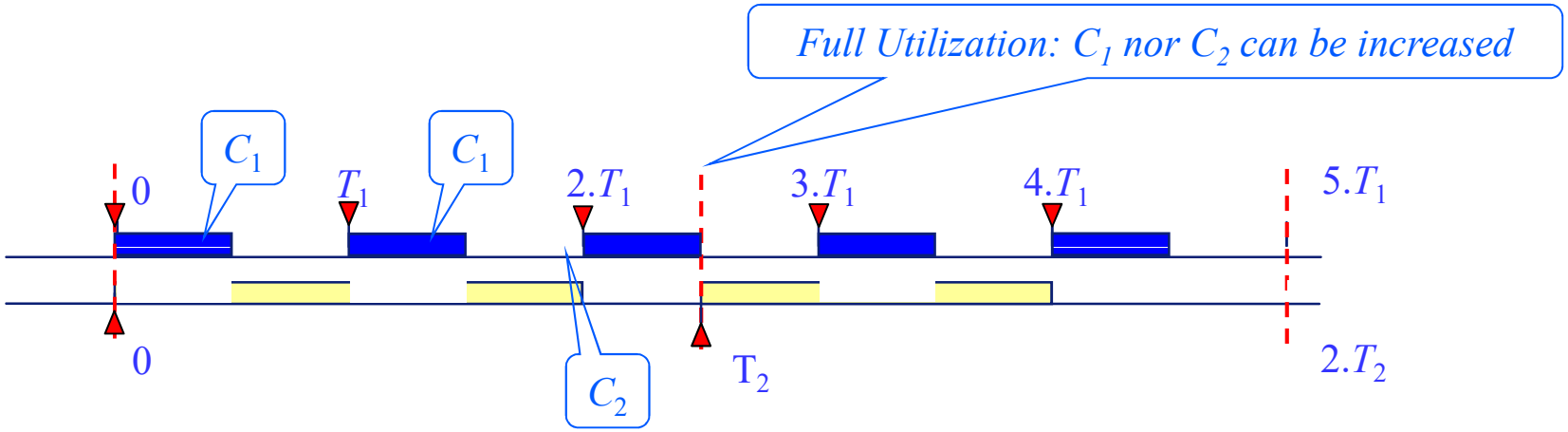
  $\tau_2 : \; r_2 = 7$

- Utilization

  $U_{full} = (1/5 + 5/7) \; = \; (32/35) = 0.91$

Example: Full Utilization

- $\tau_1 : \; C_1 = 2 \; , \; T_1 = 4$
- $\tau_2 : \; C_2 = 4 \; , \; T_2 = 10 \quad (U_{full}$ minimal at $T_2 = 6^{1/2}\, T_1 = 2.45\, T_1 \,)$



- Worst case response times:

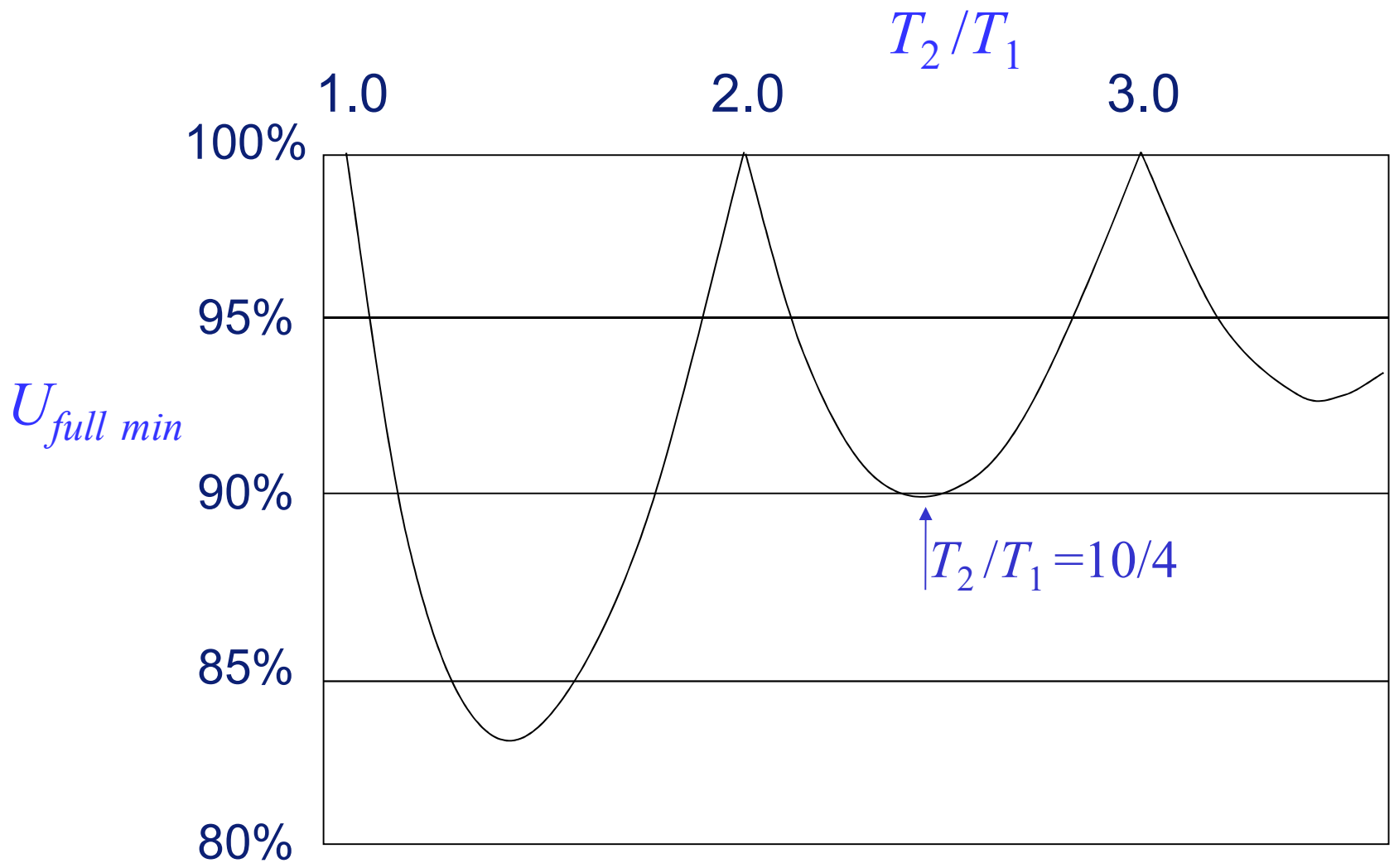
  $\tau_1 : \; r_1 = 2$

  $\tau_2 : \; r_2 = 8$

- Utilization

  $U_{full} = (2/4 + 4/10) \; = \; (9/10) = 0.90$

Minimal Value of Full Utilization varying T_2/T_1



Arbitrarily many tasks

- Approach
  - Consider free variables in utilization: $U\,(C_1\,,T_1,\ldots\,,C_m\,,T_m)$
  - Eliminate C_m to obtain full utilization: $U_{full}\,(C_1\,,T_1,\ldots\,,C_{m\text{-}1}\,,T_{m\text{-}1}\,,T_m)$
  - Find the C_i's for which the value of U_{full} is minimal.
  - Minimize also with respect to the T_i's

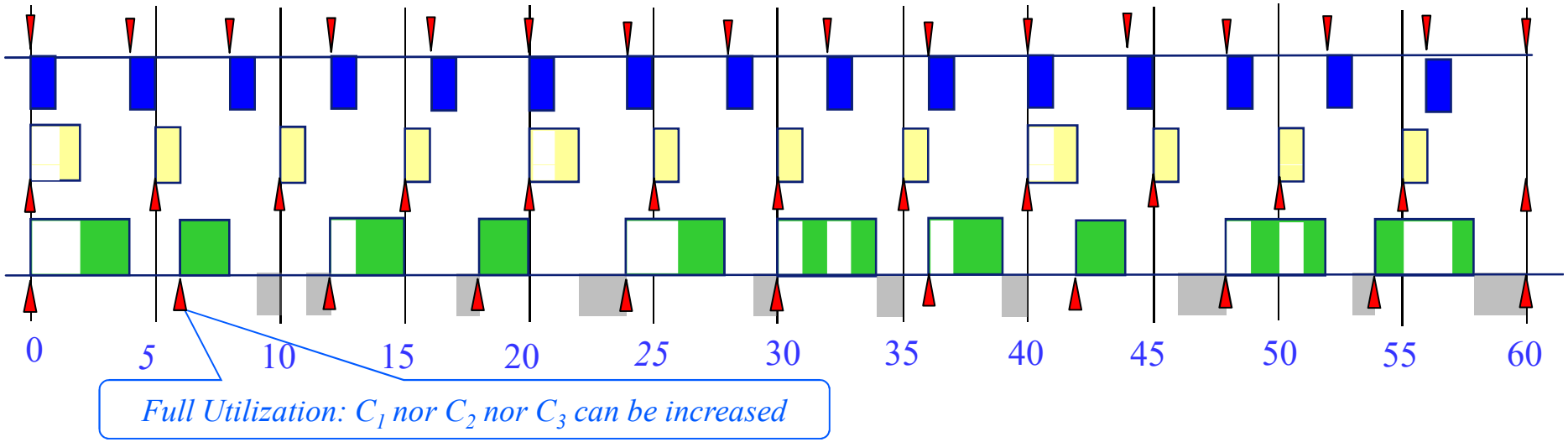
- Result:

$$\boxed{U_{\min} = m(2^{1/m} - 1)}$$

$$U_{\min}(3) \approx 0.780$$
$$U_{\min}(\infty) \approx 0.696$$

Example: Full Utilization for 3 Tasks

- $\tau_1 : \; C_1 = 1 \, , \, T_1 = 4$
- $\tau_2 : \; C_2 = 1 \, , \, T_2 = 5$ (U_{full} minimal at $T_2 = 2^{1/3} \, T_1 = 1.26 \, T_1$)
- $\tau_3 : \; C_3 = 2 \, , \, T_3 = 6$ (U_{full} minimal at $T_3 = 2^{2/3} \, T_1 = 1.59 \, T_1$)



- Worst case response times:

  $\tau_1 : \; r_1 = 1 \quad ; \tau_2 : \; r_2 = 2 \quad ; \tau_3 : \; r_3 = 4$

- Utilization

  $U_{full} = (1/4 + 1/5 + 2/6) \; = \; (47/60) = 0.7833$

Exercises

- Calculate RM schedulability of the following task sets
  - Use the utilization bound
  - Exercise 1. Task set:
    - $\tau_1 : \quad C_1 = 1 \quad T_1 = 3$
    - $\tau_2 : \quad C_2 = 1 \quad T_2 = 4$
  - Exercise 2. Task set:
    - $\tau_1 : \quad C_1 = 2 \quad T_1 = 5$
    - $\tau_2 : \quad C_2 = 4 \quad T_2 = 15$
    - $\tau_3 : \quad C_3 = 5 \quad T_3 = 20$
  - Exercise 3. Task set:
    - $\tau_1 : \quad C_1 = 1 \quad T_1 = 4$
    - $\tau_2 : \quad C_2 = 2 \quad T_2 = 6$
    - $\tau_3 : \quad C_3 = 3 \quad T_3 = 10$

Calculation of Response Times

- Utilization bound is sufficient but not necessary
  - Also task sets that do not satisfy the utilization bound may be schedulable even if $U > U_{min}$
- A more precise bound can be given for each specific task set
- This is done on the basis of response times.

- Note: we can simply draw a schedule and calculate worst case task response times.
- This calculation by construction can be formalized

Response-Time Approach

- The worst case response time r_i of a τ_i happens at the critical instant and satisfies

$$r_i = \sum_{j=1}^{i-1} C_j \left\lceil \frac{r_i}{T_j} \right\rceil + C_i$$

Execution time of task itself

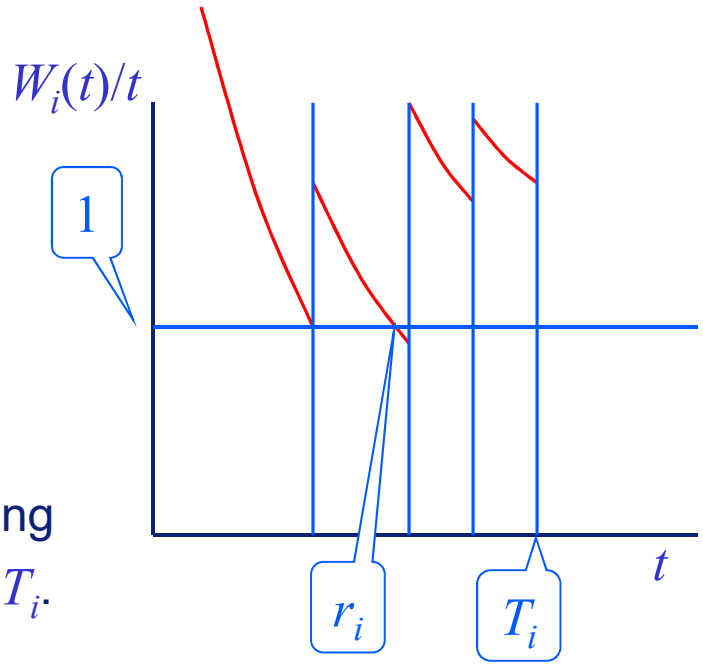
Worst case waiting time for execution of higher priority tasks

Maximum number of times higher priority task τ_j may be executed in period r_i

Schedulability of Task τ_i

- Consider: $$\frac{W_i(t)}{t} = \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil$$
  - Task τ_i can be scheduled if a solution for t in $W_i(t)/t = 1$ exists
  - The response time r_i equals the smallest of such solution.

- Take: $$\boxed{\min_{0<t\le T_i} \frac{W_i(t)}{t} \le 1}$$
  - Then there is a solution for r_i :
    - $\lim_{t \downarrow 0} W(t)/t = \infty$
    - $W_i(t)/t$ is continuous and strictly decreasing
    - Except when t is a multiple of any of the T_i.
    - Then $W_i(t)/t$ jumps up discontinuously



Single Schedulability Criterion

- A solution r_i must exist for each task τ_i
- Therefore all tasks are schedulable iff

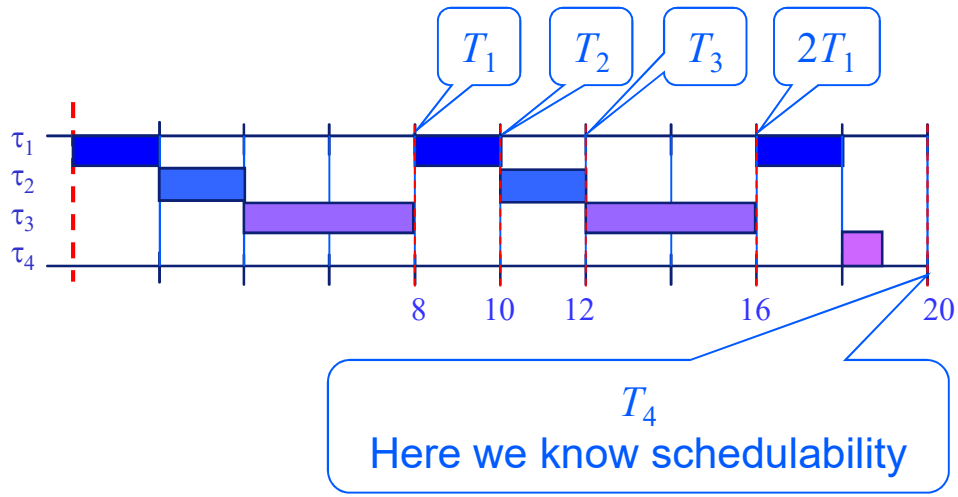
$$\boxed{\max_{1 \le i \le m} \left[\min_{0 < t \le T_i} \frac{W_i(t)}{t} \right] \le 1}$$

  – Reason: $\max \le 1$ only if each $\min_i \le 1$

- We only need to consider $W_i(t)$ at integer multiples of the periods T_i (see graph)

Example

- Task set:
  - τ_1 : $C_1 = 2$ $T_1 = 8$
  - τ_2 : $C_2 = 2$ $T_2 = 10$
  - τ_3 : $C_3 = 4$ $T_3 = 12$
  - τ_4 : $C_4 = 1$ $T_4 = 20$



  - Utilization bound: $U = (2/8 + 2/10 + 4/12 + 1/20) = 0.8333$
    - $U(4) \approx 0.757$; $U(3) = 0.780$ so last two tasks have no guarantee

  - Consider: $t = 8, 10, 12, 16, 20$
    - $W_4(8) = C_1 + C_2 + C_3 + C_4 = 9$ not ok (1 short: length of τ_4)
    - $W_4(10) = 2C_1 + C_2 + C_3 + C_4 = 11$ not ok (still 1 short: length of τ_4
    - $W_4(12) = 2C_1 + 2C_2 + C_3 + C_4 = 13$ not ok (still 1 short: length of τ_4)
    - $W_4(16) = 2C_1 + 2C_2 + 2C_3 + C_4 = 17$ not ok (still 1 short: length of τ_4)
    - $W_4(20) = 3C_1 + 2C_2 + 2C_3 + C_4 = 19$ ok r_4 =19

Exercises

- Calculate RM schedulability of the following task sets
  - Use the response time formula
  - Exercise 1. Task set:
    - $\tau_1 : \quad C_1 = 1 \quad T_1 = 3$
    - $\tau_2 : \quad C_2 = 1 \quad T_2 = 4$
  - Exercise 2. Task set:
    - $\tau_1 : \quad C_1 = 2 \quad T_1 = 5$
    - $\tau_2 : \quad C_2 = 4 \quad T_2 = 15$
    - $\tau_3 : \quad C_3 = 5 \quad T_3 = 20$
  - Exercise 3. Task set:
    - $\tau_1 : \quad C_1 = 1 \quad T_1 = 4$
    - $\tau_2 : \quad C_2 = 2 \quad T_2 = 6$
    - $\tau_3 : \quad C_3 = 3 \quad T_3 = 10$

Content

- Introduction
- Static priority scheduling
  - RM scheduling without resource contention
    - Periodic tasks
    - Sporadic tasks

Sporadic tasks

- Sporadic tasks
  - Released irregularly
  - Have a maximum rate at which they are released
  - Thus assuming a minimum inter-arrival time T
- Sporadic tasks can be treated as periodic tasks with period T

Content

- Introduction
- Static priority scheduling
  - RM scheduling without resource contention
    - Periodic tasks
    - Sporadic tasks
    - Aperiodic tasks

Aperiodic Tasks

- True aperiodic tasks
  - Can be arbitrarily released
  - No guarantee can be given if released too often
  - Deal with them without spoiling schedulability of periodic and sporadic tasks
- Several approaches among which:
  - Run as background tasks
  - Reserving space with a high priority periodic task
  - Deferred server
  - Sporadic server
  - ...

Run at Background

- Run the aperiodic tasks at lowest priority.
  - Periodic and sporadic tasks experience no interference.
  - Aperiodic tasks may run when the processor is not allocated to the periodic/sporadic ones.
  - Possibly no timeliness of any of the aperiodic task-instances.

Run in High-Priority Task

- Introduce a fictitious high priority server task
  - Period T_s
  - Execution time C_s
  - Aperiodic tasks run in slots of this high priority one.
  - If no aperiodic task are released before or within this slot the processor is available
  - Outside these slots no running of the aperiodic tasks
- Schedulability of the periodic tasks is determined including the fictitious task
- If execution time of an aperiodic task is smaller C_s then such task can finish within time $T_s + C_s$
  - At least, if aperiodic tasks are not released too often

Deferred Server

- Slight modification
  - Avoid idle processor if no aperiodic tasks are released
  - Slot for server process starts every period T_d
    - If aperiodic processes have been released
      - **Execute them for period C_d**
    - If no aperiodic processes are released: defer server execution to moment such tasks are released
      - **Preempt other tasks**
      - **Execute them for period C_d**
      - **Capacity reset to C_d at start of period T_d**
  - Utilization bound
    - Aperiodic tasks handled by the deferred server have utilization U_d
    - Periodic tasks have utilization U_p

$$\boxed{U_p + U_d < 0.652}$$

Sporadic Server

- Slight modification
  - To avoid server to execute two periods back-to-back
  - Slot for server process starts every period T_s and has capacity C_s
    - When aperiodic processes are released
      - **Execute them for requested period $C_c < C_s$**
      - **Capacity C_c is added to remaining capacity after T_s time from start of consumption of C_c**
      - **Sporadic server behaves like a sporadic task with period T_s**
  - Utilization bound
    - Aperiodic tasks handled by the deferred server have utilization U_s
    - Periodic tasks have utilization U_p

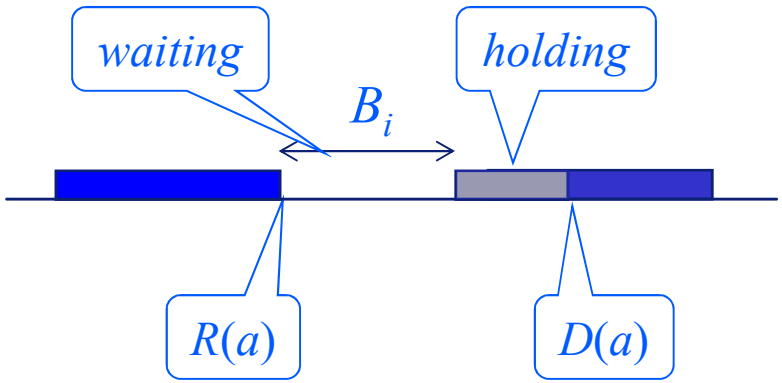
$$\boxed{U_p \leq \ln\left(\frac{2}{U_s + 1}\right)}$$

Content

- Introduction
- Static priority scheduling
  - RM scheduling without resource contention
    - Periodic tasks
    - Sporadic tasks
    - Aperiodic tasks
  - RM scheduling with resource contention

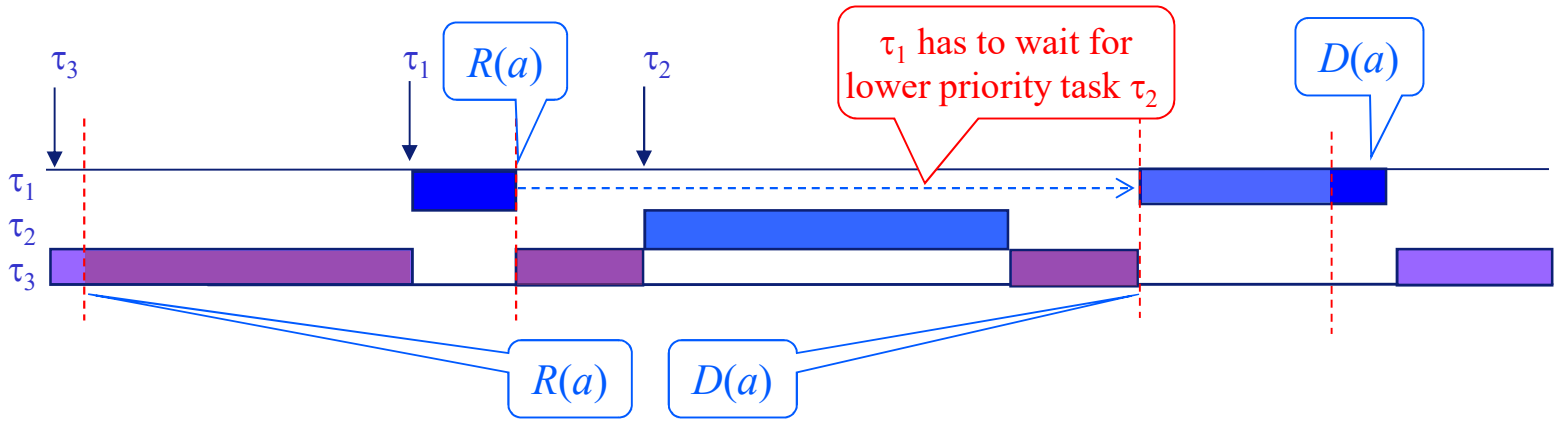
Shared Resources

- Tasks may need other resources apart from processor
  - Typical example: mutual exclusive access to data
- Important instances
  - Reserve the resource: $R(a)$
  - Wait for the resource (blocked). Blocking period: B_i
  - Hold the resource
  - Release (de-reserve) the resource: $D(a)$



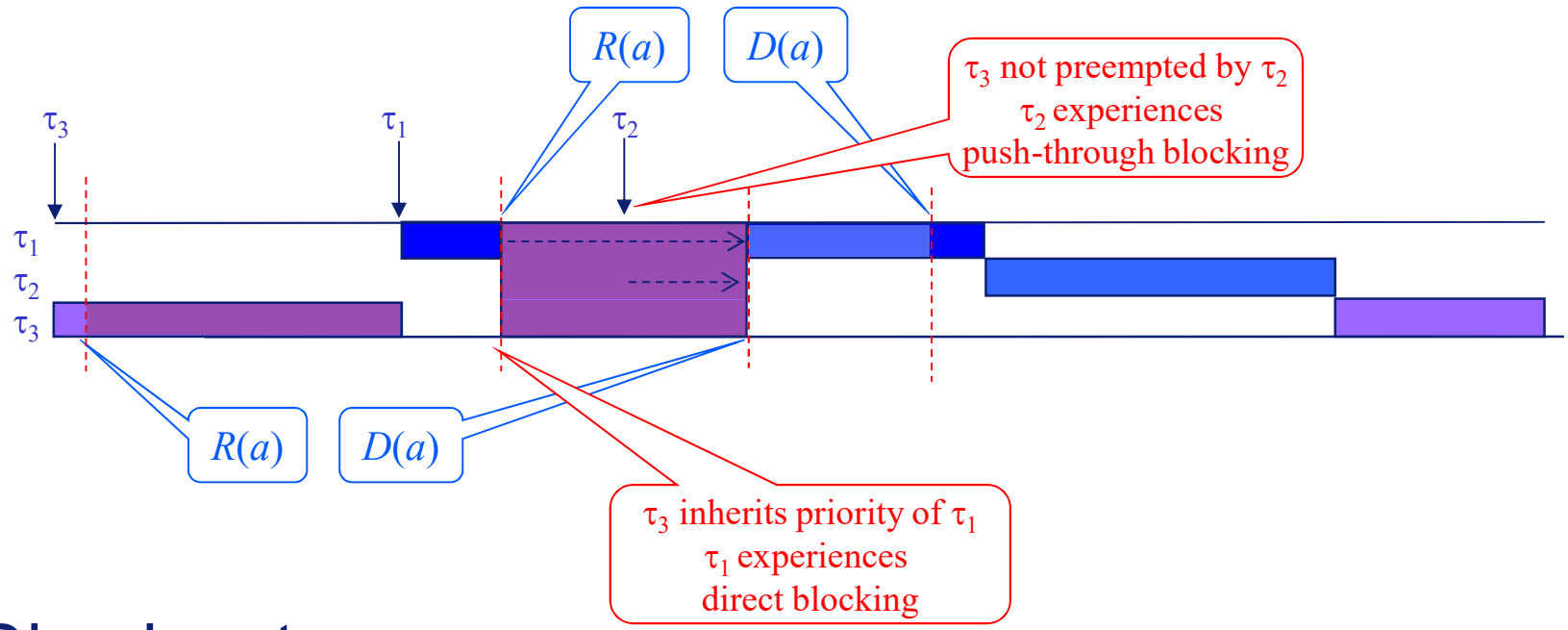
Problem: Priority Inversion

- Consider three tasks
  - Priorities $p_1 > p_2 > p_3$
  - Share a single resource a
  - Events
    - τ_3 holds resource
    - τ_1 reserves resource and waits
    - τ_2 starts, executes and finishes
    - τ_3 releases resource
    - τ_1 gets resource but executes after τ_2 !!!



Solution: Priority Inheritance

- Blocking task inherits higher priority of blocked task



- Disadvantage:
  - Deadlocks remains possible
    - If tasks reserves resources while holding others (hold and block)
    - Creating a circular wait condition

Utilization bound

- A given task τ_i can be scheduled if

$$\sum_{j=1}^{i} \frac{C_j}{T_j} + \frac{B_i}{T_i} \leq i\left(2^{1/i} - 1\right)$$

- For m tasks:

$$\sum_{j=1}^{m} \frac{C_j}{T_j} + \max_{i=1}^{m-1} \frac{B_i}{T_i} \leq m\left(2^{1/m} - 1\right)$$

Response-Time Approach

- Assume a single resource shared by tasks τ_i
- Blocking delay B_i for task τ_i
  - Direct blocking
  - Push-through blocking
- The worst case response time r_i of a τ_i and should now include these blocking contributions

$$r_i = \sum_{j=1}^{i-1} C_j \left\lceil \frac{r_i}{T_j} \right\rceil + C_i + \underbrace{B_i}_{\text{Blocking}}$$

Blocking Contribution

- Single resource a
- $C_{a,i}$ is execution time of resource a by task τ_i
  - Note: execution times C_i include such contributions!
- p_a is the highest priority among tasks possibly reserving a
- Two cases
  - $p_a \geq p_i > p_m$ then $B_i = \max_{j=i+1}^{m} C_{a,j}$
  - $p_a < p_i \vee i = m$ then $B_i = 0$

τ_i only blocked by lower priority tasks

τ_i does not reserve a

τ_m does not experience blocking

- Computation intensive for more resources due to chained blocking.

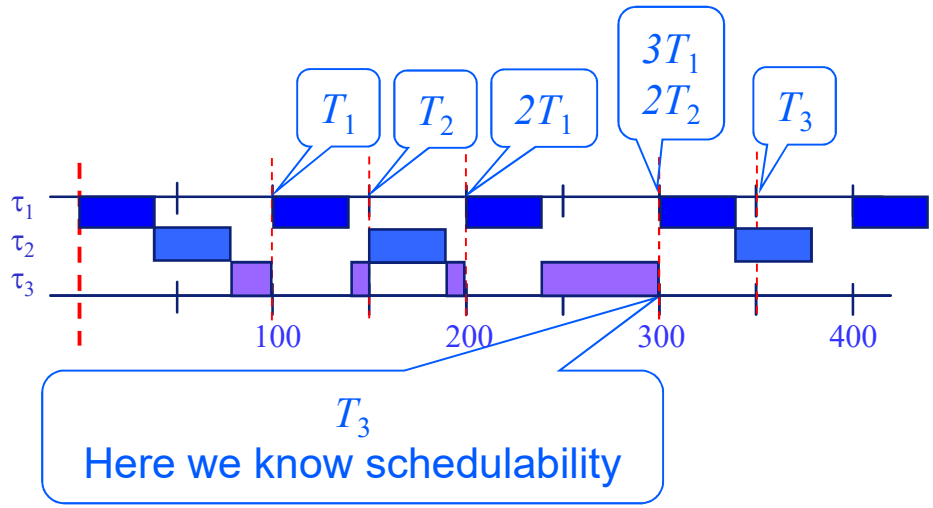
Schedulability of Task τ_i

- Consider: $$W_i(t) = \sum_{j=1}^{i} C_j \left\lceil \frac{t}{T_j} \right\rceil + B_i$$
  - Again task τ_i can be scheduled if a solution for t in $W_i(t)/t = 1$ exists
  - The response time r_i equals the smallest of such solution.

- We only need to consider $W_i(t)$ at integer multiples of the periods T_i

Example

- Task set:
  - τ_1 : $C_1 = 40$ $T_1 = 100$ $B_1 = 40$
  - τ_2 : $C_2 = 40$ $T_2 = 150$ $B_2 = 30$
  - τ_3 : $C_3 = 100$ $T_3 = 350$ $B_3 = 0$



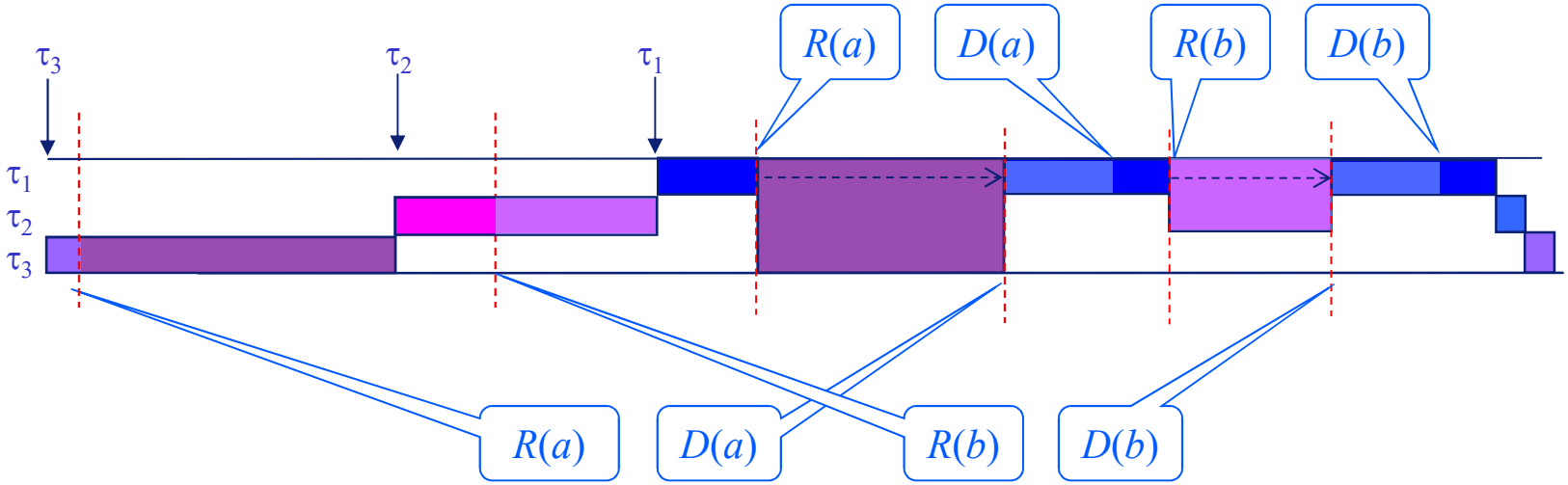
  – Consider: $t = 100, 150, 200, 300, 350$
    - $W_1(100) = C_1 + B_1$ $= 80$ ok
    - $W_2(100) = C_1 + C_2 + B_2$ $= 110$ not ok
    - $W_2(150) = 2C_1 + C_2 + B_2$ $= 150$ ok
    - $W_3(100) = C_1 + C_2 + C_3$ $= 180$ not ok
    - $W_3(150) = 2C_1 + C_2 + C_3$ $= 220$ not ok
    - $W_3(200) = 2C_1 + 2C_2 + C_3$ $= 260$ not ok
    - $W_3(300) = 3C_1 + 2C_2 + C_3$ $= 300$ ok
    - $W_3(350) = 4C_1 + 3C_2 + C_3$ $= 380$ not ok

Exercises

- Calculate RM schedulability of the following task sets
  - Exercise 4. Task set:
    - $\tau_1 : \quad C_1 = 4 \quad T_1 = 10 \quad B_1 = 3$
    - $\tau_2 : \quad C_2 = 3 \quad T_2 = 15 \quad B_2 = 2$
    - $\tau_3 : \quad C_3 = 4 \quad T_3 = 20 \quad B_3 = 0$

Problem: Chained Blocking

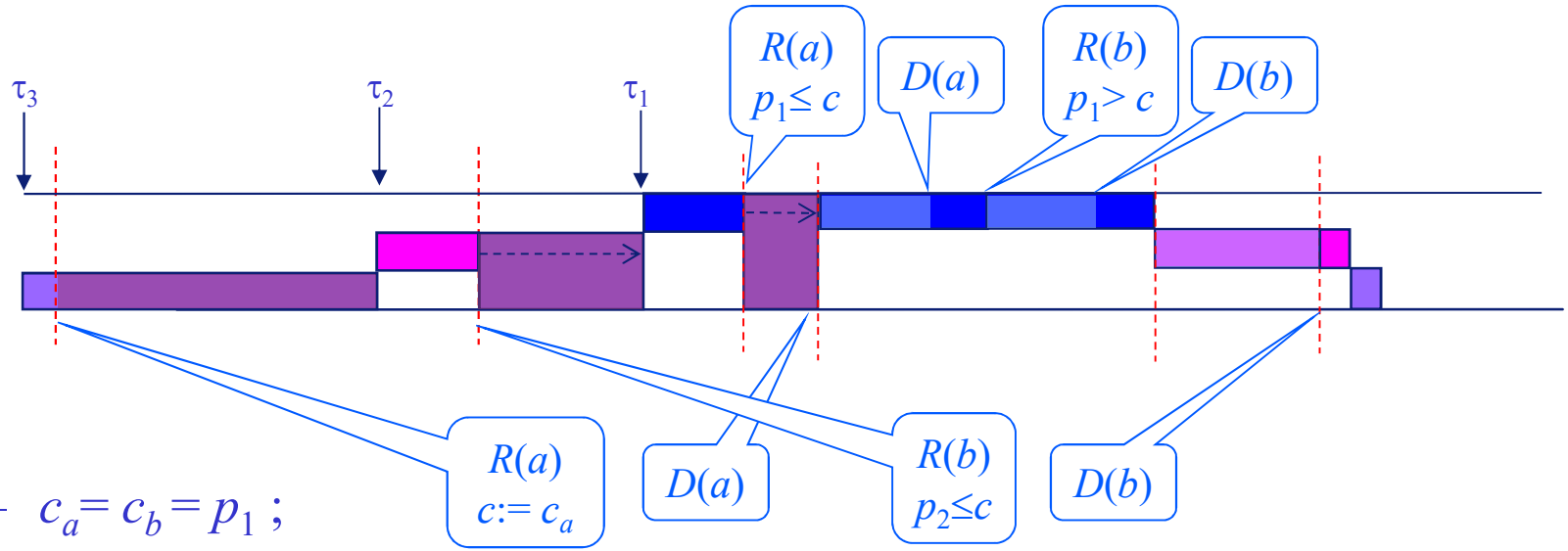
- Chained blocking when using multiple resources:



- Disadvantage:
  - Not just a single low-priority task can block the higher one.

Solution: Priority Ceiling

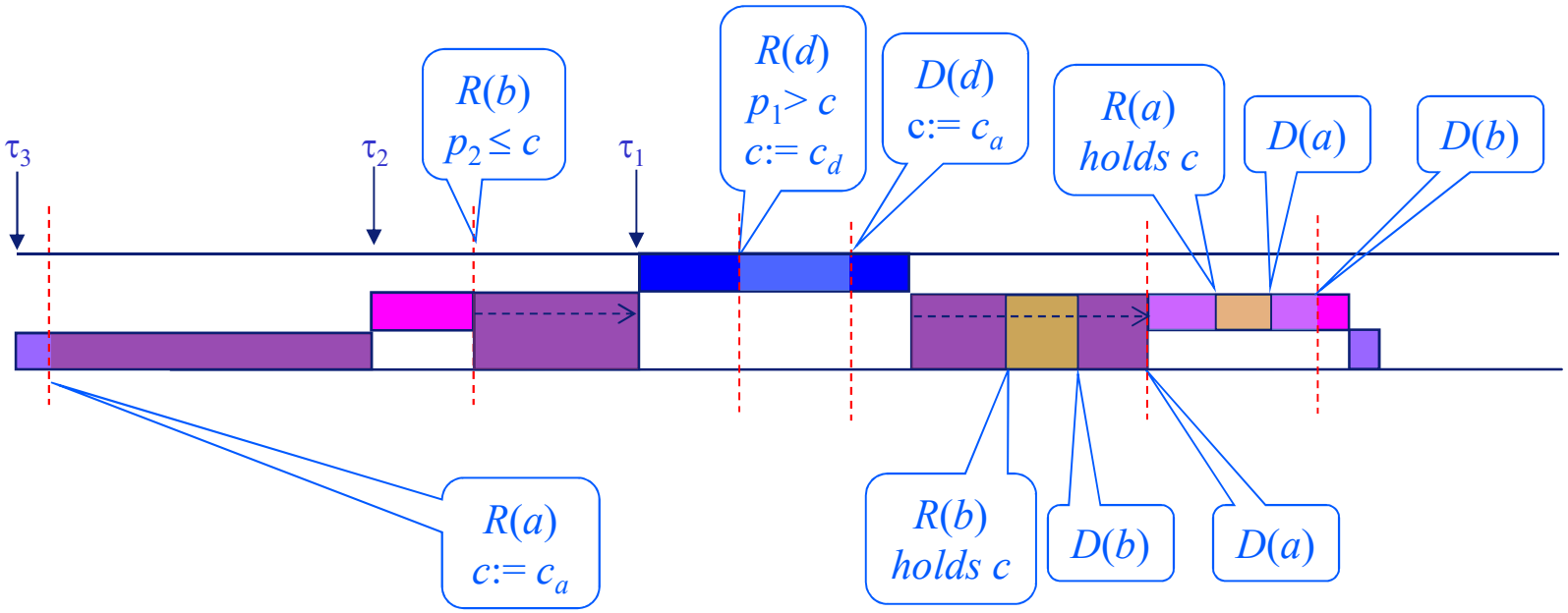
- To avoid chained blocking
  - Ceiling c_r of a resource r is highest priority found among the potentially reserving tasks
  - System ceiling c is highest ceiling of all currently held resources
  - A task τ_i may hold a resource r if $p_i > c$ or τ_i already holds a resource with $c_r = c$
  - Priority inheritance like before when tasks block because of resource usage by other tasks



  - $c_a = c_b = p_1$;

Priority Ceiling

- Also avoids deadlock



Blocking Contribution

- $C_{a,i}$ is execution time of resource a by task τ_i
- BR_i are the resources that can block τ_i
  - $BR_i = \{ \mathrm{r} \mid c_r \geq p_i \}$
- Blocking for task τ_i

$$B_i = \max_{j=i+1}^{m} \left(\max_{r \in BR_i} C_{r,j} \right)$$

τ_i only blocked by lower priority tasks

Content

- Introduction
- Static priority scheduling
  - RM scheduling without resource contention
    - Periodic tasks
    - Sporadic tasks
    - Aperiodic tasks
  - RM scheduling with resource contention
  - DM scheduling

Deadline $\neq$ Period

- Up to now the assumption with regard to deadline $D_i = T_i$
- If $D_i < T_i$, two possibilities
  - DM assignment of priorities: $D_i < D_j \leftrightarrow p_i > p_j$
  - RM assignment of priorities
- In either case, the response time formula is applicable
  - Modified use of the response time formula
    - Compute response time r_i for each task
    - Compare r_i with D_i