CIS 721 - Real-Time Systems
Lecture 9: Arbitrary Start Times

Mitch Neilsen
neilsen@cis.ksu.edu

Outline

- Clock-Driven Scheduling (Ch. 5)
- **Priority-Driven Scheduling**
  - **Periodic Tasks (Ch. 6)**
    - **Arbitrary Start Times**
      - **Leung’s Feasibility Test**
      - **Audsley’s Feasibility Test**
    - **Arbitrary Deadlines**
  - Aperiodic and Sporadic Tasks (Ch. 7)

Arbitrary Start Times (Phasing)

- N.C. Audsley, “Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times”, Tech. Report YCS 164, University of York, York, England, 1991.

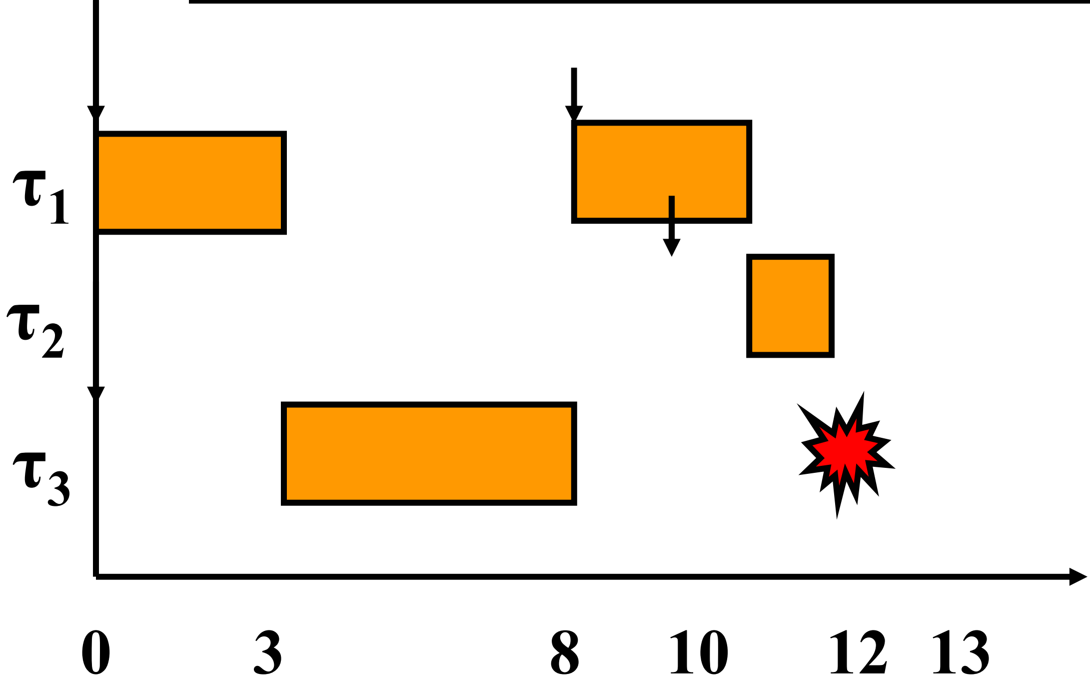
Example #1

- If phasing (φ_i or $\mathbf{O_i}$) is allowed to be greater than 0, then a Rate Monotonic (RM) priority assignment may not be optimal.

Task τ_i	Period T_i	Deadline D_i	Run-Time C_i	Phase φ_i
τ_1	8	8	3	0
τ_2	12	12	1	10
τ_3	12	12	6	0

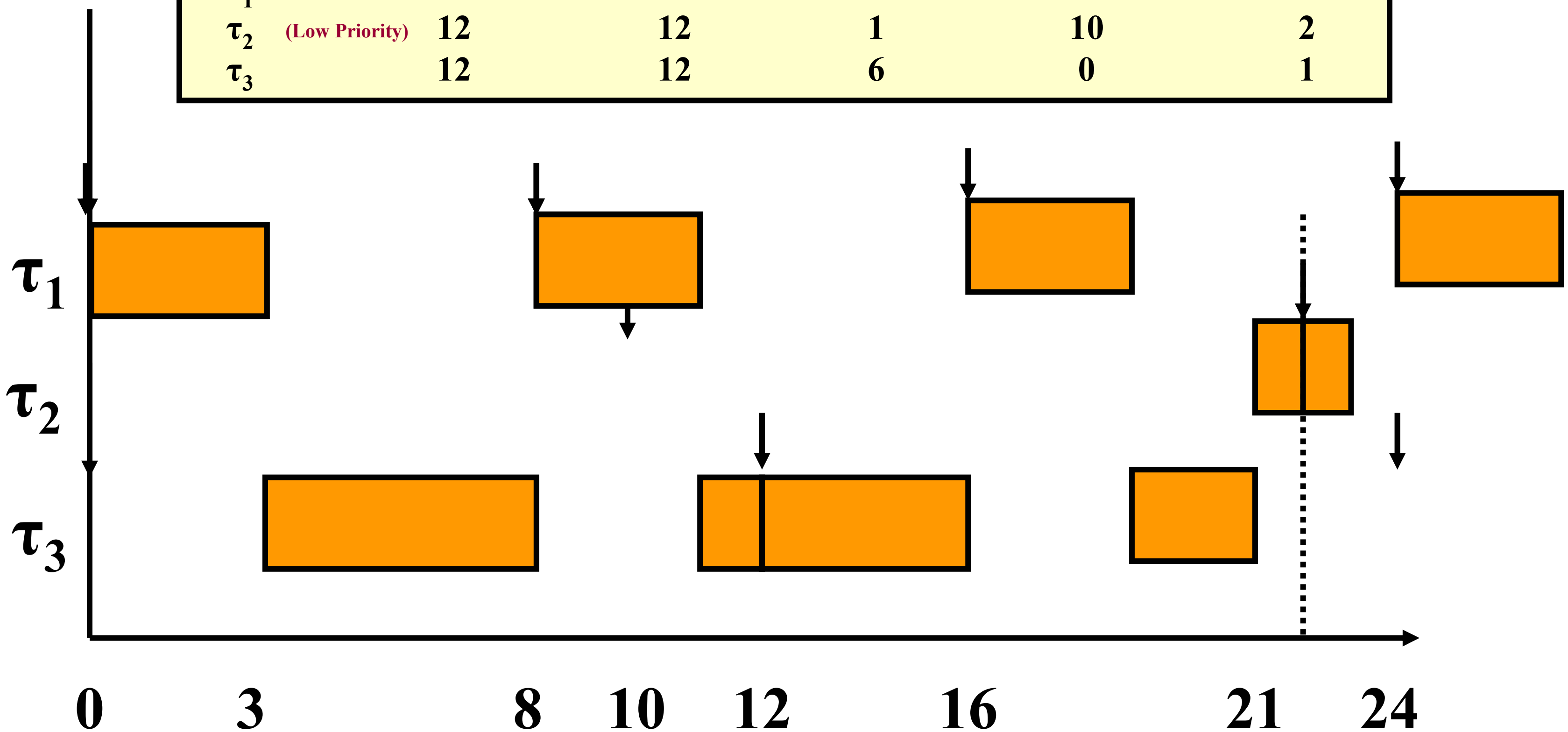
Example #1 (cont.)

Task τ_i		Period T_i	Deadline D_i	Run-Time C_i	Phase φ_i	Priority π_i
τ_1	(High Priority)	8	8	3	0	3
τ_2		12	12	1	10	2
τ_3	(Low Priority)	12	12	6	0	1



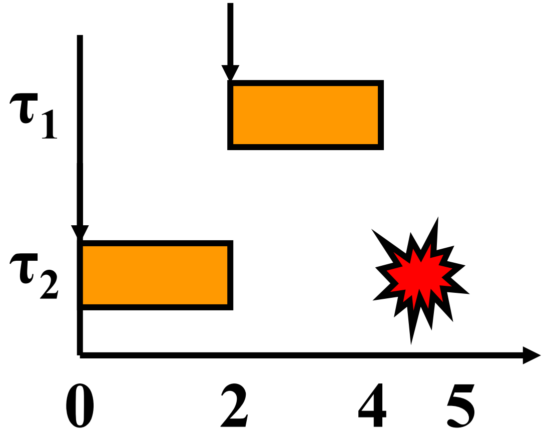
Example #1b

Task τ_i		Period T_i	Deadline D_i	Run-Time C_i	Phase φ_i	Priority π_i
τ_1	(High Priority)	8	8	3	0	3
τ_2	(Low Priority)	12	12	1	10	2
τ_3		12	12	6	0	1



Example #2

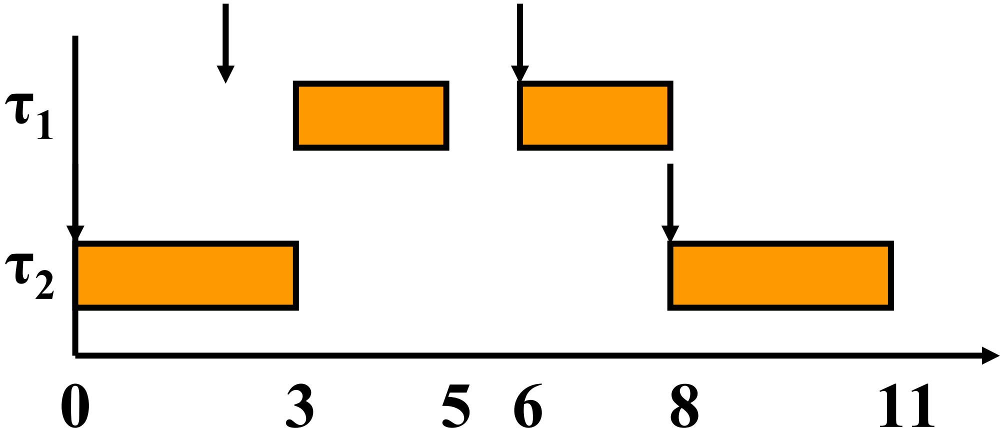
Task τ_i	Period T_i	Deadline D_i	Run-Time C_i	Phase φ_i
τ_1 (High Priority)	4	3	2	2
τ_2 (Low Priority)	8	4	3	0



If phasing (φ_i) is allowed to be greater than 0, then a Deadline Monotonic (DM) priority assignment may not be optimal.

Example #3

Task τ_i	Period T_i	Deadline D_i	Run-Time C_i	Phase φ_i
τ_1 (Low Priority)	4	3	2	2
τ_2 (High Priority)	8	4	3	0



Leung's Test

- J. Leung and J. Whitehead, "On the Complexity of Fixed Priority Scheduling of Periodic Real-Time Tasks", Performance Evaluation, 2(4):237-250, 1982.
- A task set is **feasible** if all deadlines are met in the interval **[*s, 2P*)** where
  - $\boldsymbol{s = \max\{\varphi_1, \varphi_2, \ldots, \varphi_n\}}$
  - $\boldsymbol{P = lcm\{T_1, T_2, \ldots, T_n\}}$
- Implicit assumption: $\boldsymbol{s \leq P}$

Simple Test

- Compute $P = lcm\ \{ T_1, T_2, \ldots, T_n \}$.
- Set $s = \max \{\varphi_1, \varphi_2, \ldots, \varphi_n\}$.
- If ($s \leq P$), set $S = 0$; otherwise, set $S = \lfloor s/P \rfloor P$.
- Construct a schedule for the interval $[S, S + 2P)$.
- Check the schedule to see if all deadlines are met.

Example #4

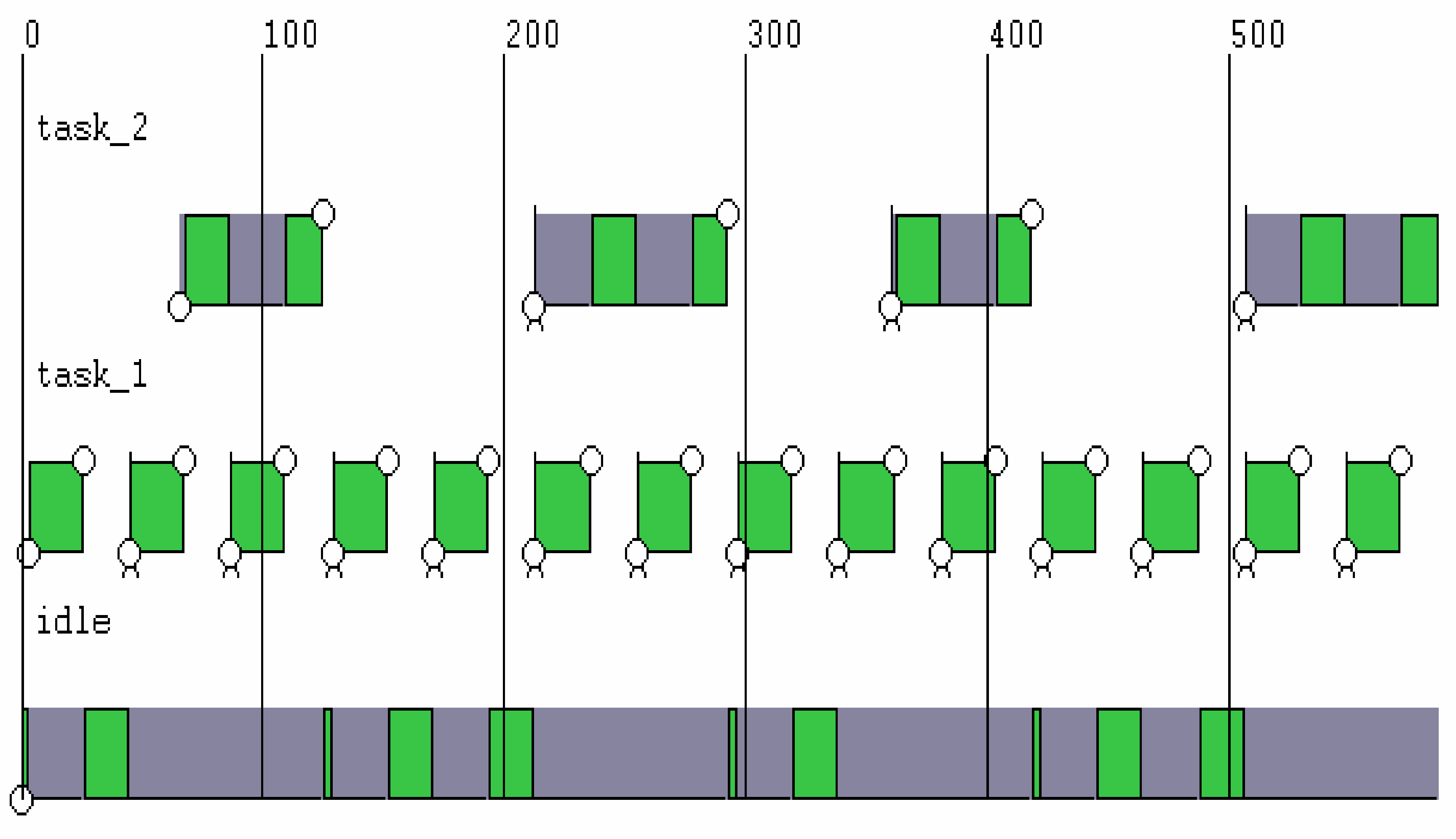
Task τ_i	Period T_i	Deadline D_i	Run-Time C_i	Phase φ_i
τ_1	42	42	23	3
τ_2	147	147	34	66

- Since $gcd(42, 147) = 21$, $P = lcm(42, 147) = 294$.
- Also, $s = 66$, so $S = 0$.
- Check for missed deadlines in [0, 588).

Stress Program Input

```
/* audsley1.str: Example From Audsley's
Paper */
system
  node node_1
    processor proc_1
      periodic task_1
        period 42 deadline 42 offset 3
        priority 1
        [23,23]
      endper
      periodic task_2
        period 147 deadline 147 offset 66
        priority 2
        [34,34]
      endper
    endpro
  endnod
endsys
```

Example: No Missed Deadlines



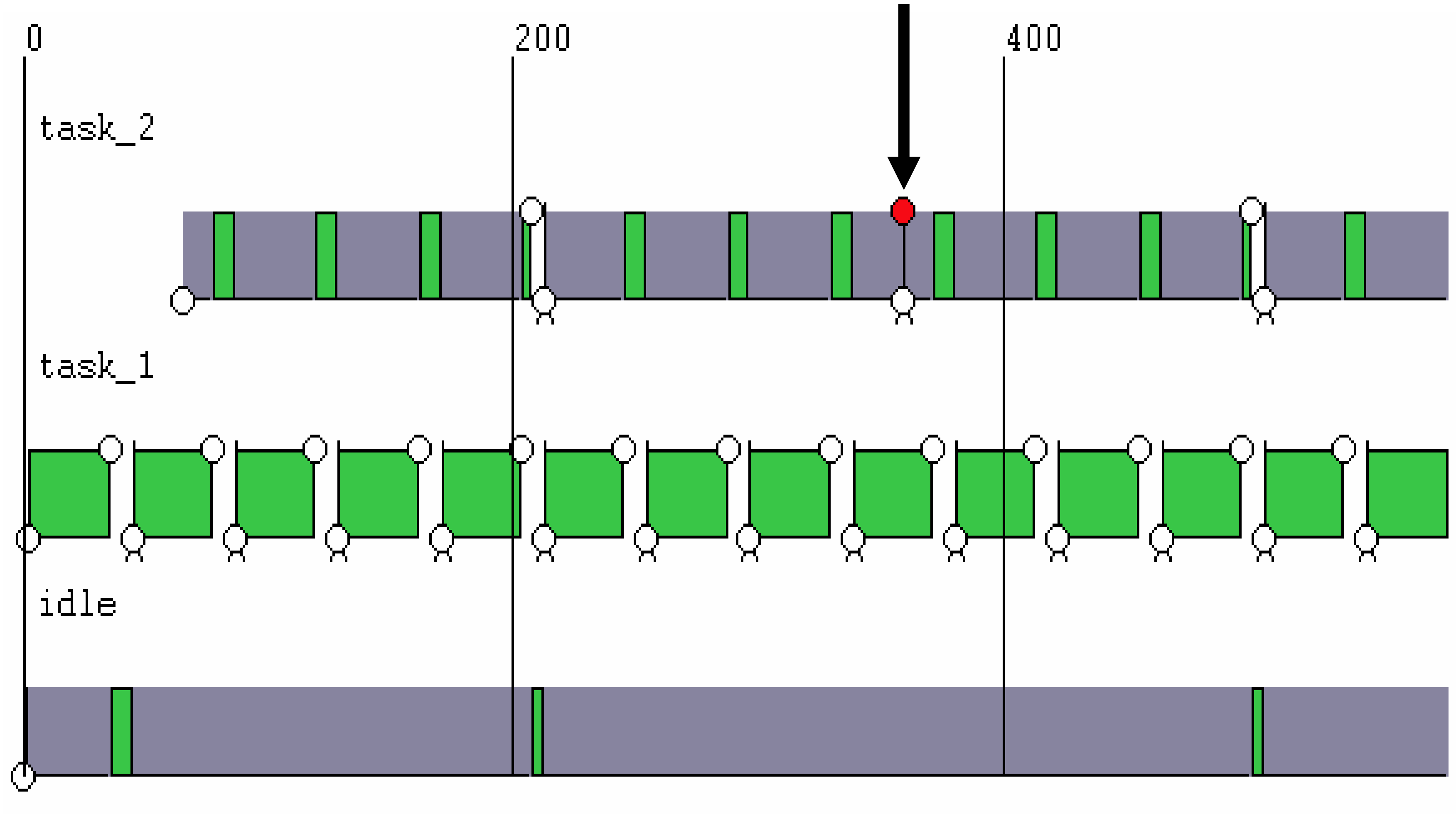
Example #5

Task τ_i	Period T_i	Deadline D_i	Run-Time C_i	Phase φ_i
τ_1	42	42	33	3
τ_2	147	147	31	66

Stress Program Input

```
/* audsley2.str: Example From Audsley's Paper */
system
  node node_1
    processor proc_1
      periodic task_1
        period 42 deadline 42 offset 3
        priority 1
        [33,33]
      endper
      periodic task_2
        period 147 deadline 147 offset 66
        priority 2
        [31,31]
      endper
    endpro
  endnod
endsys
```

Example: Missed Deadline



Audsley's Feasibility Test

- Construct a method to determine if tasks share a **common release time**, and combine such tasks.
- Construct a method to compute an **optimal priority assignment**.
- Construct a **feasibility test**.
- Combine optimal priority assignment method and feasibility test.

Special Case: Two Tasks

- **Determine if the two tasks τ_1 and τ_2 share a common release time.**
- We may assume that $\boldsymbol{\varphi_i < T_i}$ for all i:
  - if ($\boldsymbol{\varphi_i = T_i}$), we can set $\boldsymbol{\varphi_i} = 0$, and
  - if ($\boldsymbol{\varphi_i > T_i}$), we can reduce the offset to $\boldsymbol{\varphi_i = \varphi_i - \lfloor \varphi_i / T_i \rfloor T_i}$.
- A critical instant (common release time) will occur when:

$$\boldsymbol{\varphi_1} + a\mathbf{T_1} = \boldsymbol{\varphi_2} + b\mathbf{T_2} \qquad \mathbf{(1)}$$

Two Tasks (continued)

- W.O.L.O.G., w.m.a., $\boldsymbol{\varphi_1 < \varphi_2}$. Subtract $\boldsymbol{\varphi_1}$ from both sides of equation (1) to obtain:

  $$a\mathbf{T_1} = \boldsymbol{\varphi'_2} + b\mathbf{T_2} \text{ where } \boldsymbol{\varphi'_2 = \varphi_2 - \varphi_1} \qquad \mathbf{(2)}$$

- Equation (2) holds iff $\boldsymbol{\varphi'_2} = \mathbf{h}\ \mathit{gcd}\mathbf{(\ T_1, T_2\ )}$ for some integer **h**.
- The greatest common divisor, $\mathit{gcd}\mathbf{(\ T_1, T_2\ )}$, can be computed using Euclid's Algorithm in $O(\log(\max\{T_1, T_2\}))$ - time.

Euclid's Algorithm

```
Euclid( a, b )
    if  ( b = 0 ) then
        return ( a )
    else
        return ( Euclid( b, a mod b ) )
    end if
```

Example

Euclid(30,21) --- 30 mod 21 = 9

$\rightarrow$ Euclid(21,9) --- 21 mod 9 = 3

$\rightarrow$ Euclid(9,3) --- 9 mod 3 = 0

$\rightarrow$ Euclid(3,0)

$\rightarrow$ 3

GCD Recursion Theorem

- Let a and b denote positive integers, then *gcd*(a,b) = *gcd*(b, a mod b).
- Proof: Left as an exercise.

Many Tasks

- **Determine if many tasks, $\tau_1, \tau_2, \ldots, \tau_n$, share a common release time.**
- **Naive Approach** - compare every release of τ_i $(1 \leq i \leq n)$ with every release of τ_j $(i+1 \leq j \leq n)$. This approach has exponential time complexity.
- **More Efficient Approach -** use Euclid's Algorithm to determine if tasks share a common release time, and combine those tasks.

Efficient Approach - Two Tasks

- Consider two tasks, τ_A and τ_B.
- Assume that $0 = \varphi_A < \varphi_B < \tau_B$.
- We can form a **hybrid task** τ_{AB} to represent critical instants of the two tasks with φ_{AB} as the first critical instant. The period, T_{AB} is the least common multiple of T_A and T_B.

$$T_{AB} = \frac{T_A T_B}{\gcd(T_A, T_B)}$$

Adapted Euclid's Algorithm

- Adapt Euclid's Algorithm to find integers x and y such that $x\,T_A + y\,T_B = gcd(T_A, T_B)$.
- Then, multiply both sides by h to obtain:

  $(hx)\,T_A + (hy)\,T_B = h\,gcd(T_A, T_B) = \varphi_B$.
- Either $x < 0$ or $y < 0$. Assume that $x < 0$.
- Then, common release times occur w/ period T_{AB}; add and subtract $k\,T_{AB}$ to terms on the LHS:

  $(\,hx\,T_A + k\,T_{AB}\,) + (\,hy\,T_B - k\,T_{AB}\,) = \varphi_B$

  with $k = \lceil\, |x|\,h\,T_A / T_{AB} \,\rceil$.

Adapted Algorithm (cont.)

- Let $t = (hx\ T_A + k\ T_{AB})$.
- Since, t may be greater than T_{AB},
  $\varphi_{AB} = t - \lfloor t / T_{AB} \rfloor T_{AB}$
- Note: $\varphi_{AB} > \varphi_B$ because $T_{AB} > T_B$ and $\varphi_B < T_B$
- Successively apply the same technique to the remaining tasks in the task set Γ.
  - Compare hybrid task τ_{AB} with $\tau_C \in \Gamma - \{\tau_A , \tau_B\}$.
  - If they share a common release time, form τ_{ABC} .
  - Since there are at most n-1 hybrid tasks, the time complexity is $O(n \log(\max\{T_1, \ldots , T_n\}))$.

Example #6

Task τ_i	Period T_i	Deadline D_i	Run-Time C_i	Phase φ_i
τ_A	42	42	23	0
τ_B	147	147	34	63

- $gcd(\boldsymbol{\tau}_A , \boldsymbol{\tau}_B) = 21 \Rightarrow T_{AB} = lcm(\boldsymbol{\tau}_A , \boldsymbol{\tau}_B) = 294$, $h = 3$.
- $k = \lceil |x|\, h\, T_A / T_{AB} \rceil = \lceil 3*3*42/294 \rceil = 2$.
- $t = (\, hx\, T_A + k\, T_{AB}\,) = 3*(-3)*42 + 2*294 = 210$.
- $\varphi_{AB} = t - \lfloor t / T_{AB} \rfloor T_{AB} = 210 - \lfloor 210 / 294 \rfloor 210 = 210$.

Optimal Priority Assignment

- Task sets whose tasks are never released at the same time are called **non-critical-instant task sets**, denoted Γ^* or Δ^*.
- There are n! different priority assignment functions possible $\phi = \{\phi_1, \phi_2, \ldots, \phi_{n!}\}$.
  - $\phi_i(\boldsymbol{\tau_A}) = j$: the ith assignment function maps $\boldsymbol{\tau_A}$ to priority j, and $\phi^{-1}_i(j) = \boldsymbol{\tau_A}$ (ϕ_i is 1-1 and onto).
- Task $\boldsymbol{\tau_A}$ is schedulable iff $C_A + I_A \leq D_A$.

run-time

interference

Theorems

- **Theorem 1:** If τ_A is assigned the lowest priority and is **not** feasible, then no priority assignment that assigns τ_A the lowest priority produces a feasible schedule.
- **Theorem 2:** If τ_A is assigned the lowest priority and is feasible, then if a feasible priority assignment for Γ^* exists, an ordering with τ_A assigned the lowest priority exists.

Theorem 2

If τ_A is assigned the lowest priority, n, and is feasible, then if a feasible priority ordering for Δ^* exists, an ordering with τ_A assigned the lowest priority exists.

Proof:

Let us assume that an assignment function Φ_y produces the feasible assignment:

$$\Phi_y(\tau_B) = 1, \Phi_y(\tau_C) = 2, \ldots\ldots, \underline{\Phi_y(\tau_A) = i}, \Phi_y(\tau_D) = i + 1, \ldots\ldots, \Phi_y(\tau_E) = n$$

We note that τ_A is feasible at priority level $i < n$. A second priority assignment function Φ_x defines:

$$\Phi_x(\tau_B) = 1, \Phi_x(\tau_C) = 2, \ldots\ldots, \Phi_x(\tau_D) = i, \ldots\ldots, \Phi_x(\tau_E) = n - 1, \underline{\Phi_x(\tau_A) = n}$$

Since τ_A is feasible if assigned priority level n (by the theorem), we can assign τ_A to level n. The tasks originally assigned priority levels $i + 1 \cdots n$ in Φ_x are promoted 1 place (i.e. the task at priority level $i + 1$ is now assigned priority i). Clearly, the tasks assigned levels $1 \cdots i \cancel{+} 1$ remain feasible as nothing has changed to affect their feasibility. The tasks originally assigned levels $i + 1 \cdots n$ also remain feasible as the interference on them has decreased with τ_A now being of the lowest priority. Since τ_A is feasible at the lowest priority level, at least one feasible priority assignment exists with τ_A as the lowest priority task. The theorem is proved.

■

Theorem 3:

Let the tasks assigned priority levels $i, i+1,.., n$ by assignment function Φ_x be feasible under that priority ordering. If there exists a feasible priority ordering for Δ^*, there exists a feasible priority ordering that assigns the same tasks to levels $i..n$ as Φ_x.

Proof:

We prove the theorem by showing that a feasible priority assignment function Φ_y can be transformed to assign the same tasks to priority levels $i, i+1,.., n$ as Φ_x, whilst preserving the feasibility of Φ_y. The proof is by induction: Φ_y is transformed successively moving tasks $\Phi_x^{-1}(n), \Phi_x^{-1}(n-1),.., \Phi_x^{-1}(i)$ to priority levels $n, n-1,..,i$ under Φ_y.

Base

Let $\Phi_x^{-1}(n) = \tau_A$ and $\Phi_y(\tau_A) = m$, where $m \le n$. By theorem 2 we can move τ_A to the assigned level (n) under Φ_k without altering the feasibility of Δ^*.

Inductive Hypothesis

We assume that the tasks assigned to priority levels $n-1, n-2, .., i+1$ under Φ_x are moved to levels $n-1, n-2, .., i+1$ under Φ_y. Δ^* is assumed to remain feasible.

Inductive Step

Let $\Phi_x^{-1}(i) = \tau_B$ and $\Phi_y(\tau_B) = m$, where $m \le i$ (since the reassignment of priority levels $n, n-1, .., i+1$ has promoted τ_B to have a priority of between 1 and i). Under both orderings, the tasks assigned to priority levels $i+1, .., n$ are identical. Task τ_B is reassigned in Φ_y to level i. We know (by Φ_x) that τ_B is feasible at this level (assuming that tasks assigned to levels $i+1..n$ are identical under Φ_x and Φ_y).

After the reassignment, tasks at levels $1..i-1$ remain feasible, as their respective interferences are no greater than before the reassignment and therefore must remain feasible. This proves the theorem.

■

Algorithm 1: Optimal Priority Assignment:

```
PriorityAssignment ()
  begin
    Δ = Δ*   -- copy the non-critical instant task set
    for j in (n..1)   -- priority level j
        unassigned = TRUE
        for τA in Δ
            if (feasible(τA, j)) then -- if τA is fesaible
                                       -- for priority level j
                ψ(j) = τA    -- assign τA to priority level j
                Δ = Δ - τA
                unassigned = FALSE
            endif
            if (unassigned)
                exit  -- no feasible priority assignment exists
            endif
        endfor
    endfor
  end
```

■

Analysis

- Priorities are assigned from lowest to highest priorities. Once assigned, the priority will never need to be changed.
- Let B denote the maximum number of feasibility tests required. Then,

  $B = n + (n-1) + (n-2) + ... + 1 = n(n+1)/2 \in O(n^2)$.
- Recall that the Naive Approach requires O(n!) calls to the feasibility test.

Feasibility Test

- **Theorem 6:** Task τ_i is feasible iff the deadlines corresponding to releases of the task in the interval $[\, S_i, S_i+P_i \,)$ are met where S_i is the initial stabilisation time and $P_i = lcm\,\{ T_1, T_2, \ldots , T_i \}$.
- **Idea:** Try to reduce the length of the interval to be tested. In practice, not much better than Leung’s Test.

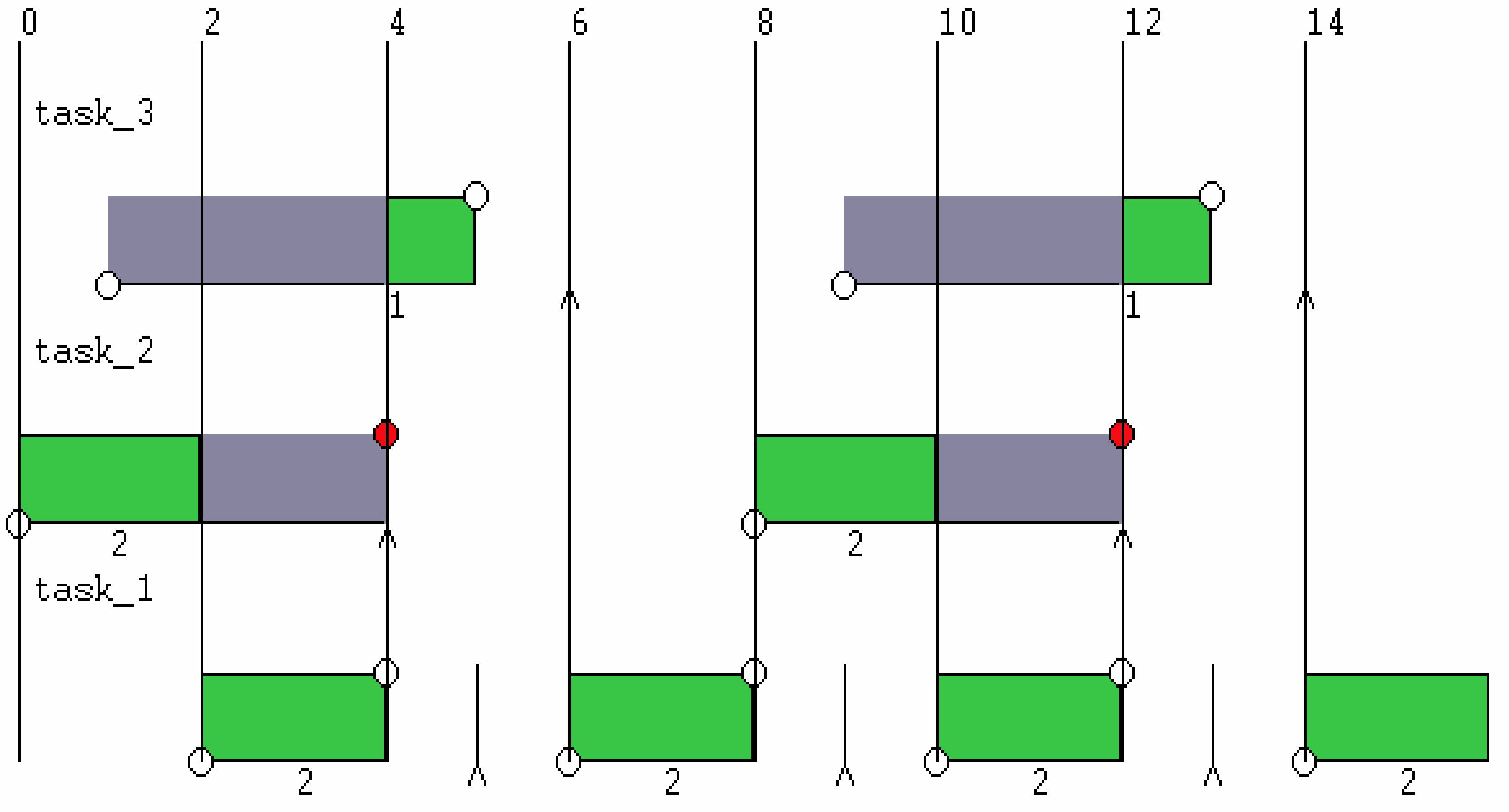
Example #7

Task τ_i		Period T_i	Deadline D_i	Run-Time C_i	Phase φ_i
τ_A	High Priority (1)	4	3	2	2
τ_B		8	4	3	0
τ_C	Low Priority (3)	8	5	1	1

$P = lcm \{ 4, 8, 8 \} = 8.$

Test Feasibility over interval [0, 16).

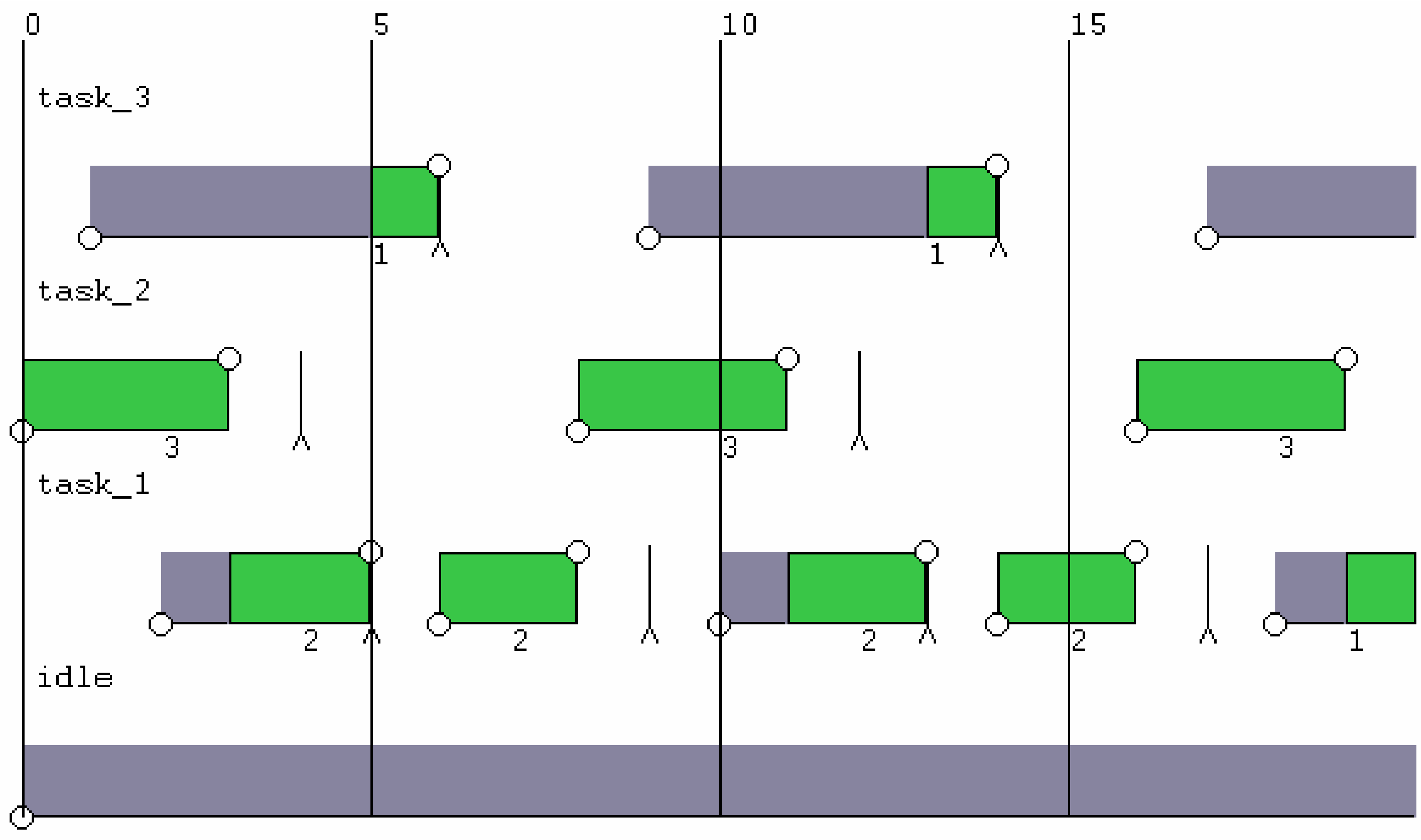
Example #7 - Schedule



Example #7 - Observations

- Task τ_C (task_3) is feasible in the feasibility interval [0, 16) (with priority = 3 = lowest).
- Task τ_B (task_2) is **not** feasible in the feasibility interval [0, 16).
- Idea: Test feasibility of τ_A with a priority of 2.

Example #7 - Revised



Definitions

- The **interference** suffered by task τ_i in the interval $[\, t, t + D_i \,)$ is denoted I^t_i .
- Interference $I^t_i = R^t_i + K^t_i$ where:

  R^t_i = **remaining interference** due to higher priority tasks that have not completed their executions at time t, but released before time t.

  K^t_i = **created interference** due to higher priority tasks released in the interval $[\, t, t + D_i \,)$.

Audsley's Feasibility Test

- For each release of task τ_i at time t, τ_i is feasible for that release iff $R^t_i + K^t_i + C_i \leq D_i$.
- Let $B_i = \{ S_i, S_i + T_i, S_i + 2T_i, \ldots , S_i + P_i \}$.
- If each release of each task τ_i at time $t \in B_i$ is feasible, then the entire task set is feasible.

Definition 1:

The initial stabilisation time, S_j, of task τ_j, is the time after which the execution of the task set repeats exactly every P_j with respect to tasks $\tau_1, \tau_2, \ldots, \tau_j$.

∎

```
FeasibilityTest ()
  begin
    for τ_i in Δ*        -- taken in order τ_1, τ_2, ...
        t = 0;
        L_i^t = 0
        while (t < S_i + P_i)

            -- Calculate R_i^t - create and order β
            RemainingInterference ()

            -- Calculate K_i^t - create and order η
            CreatedInterference ()

            if (C_i + R_i^t + K_i^t > D_i)
                exit     -- τ_i not feasible so quit
            endif

            t = t + T_i    -- go to next release of τ_i
        endwhile
    endfor
  end
```

Algorithm 2: Exact Remaining Interference.

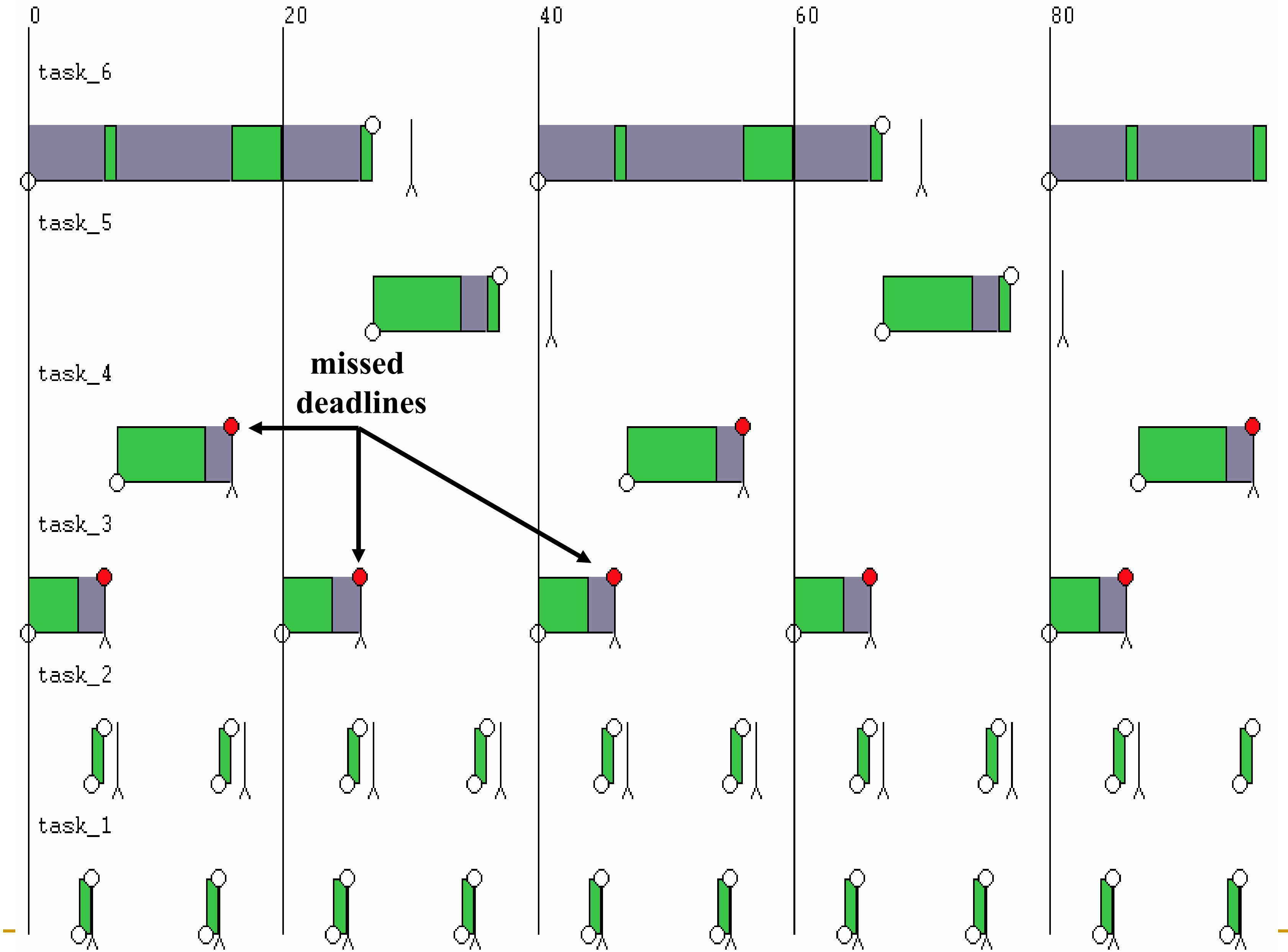
```
RemainingInterference ()
  begin
    time  =  t - T_i + D_i
    R_i^t   =  0
    -- create and order tuple set β
    -- for this release of τ_i
    for (C, t_r) in β
        if (t_r > time + R_i^t) then
            R_i^t = 0
        endif
        time = t_r
        R_i^t = R_i^t + C
    endfor
    R_i^t = R_i^t - (t - t_r)
    if (R_i^t < 0) then
        R_i^t = 0
    endif
  end
```

Algorithm 3: Exact Created Interference.

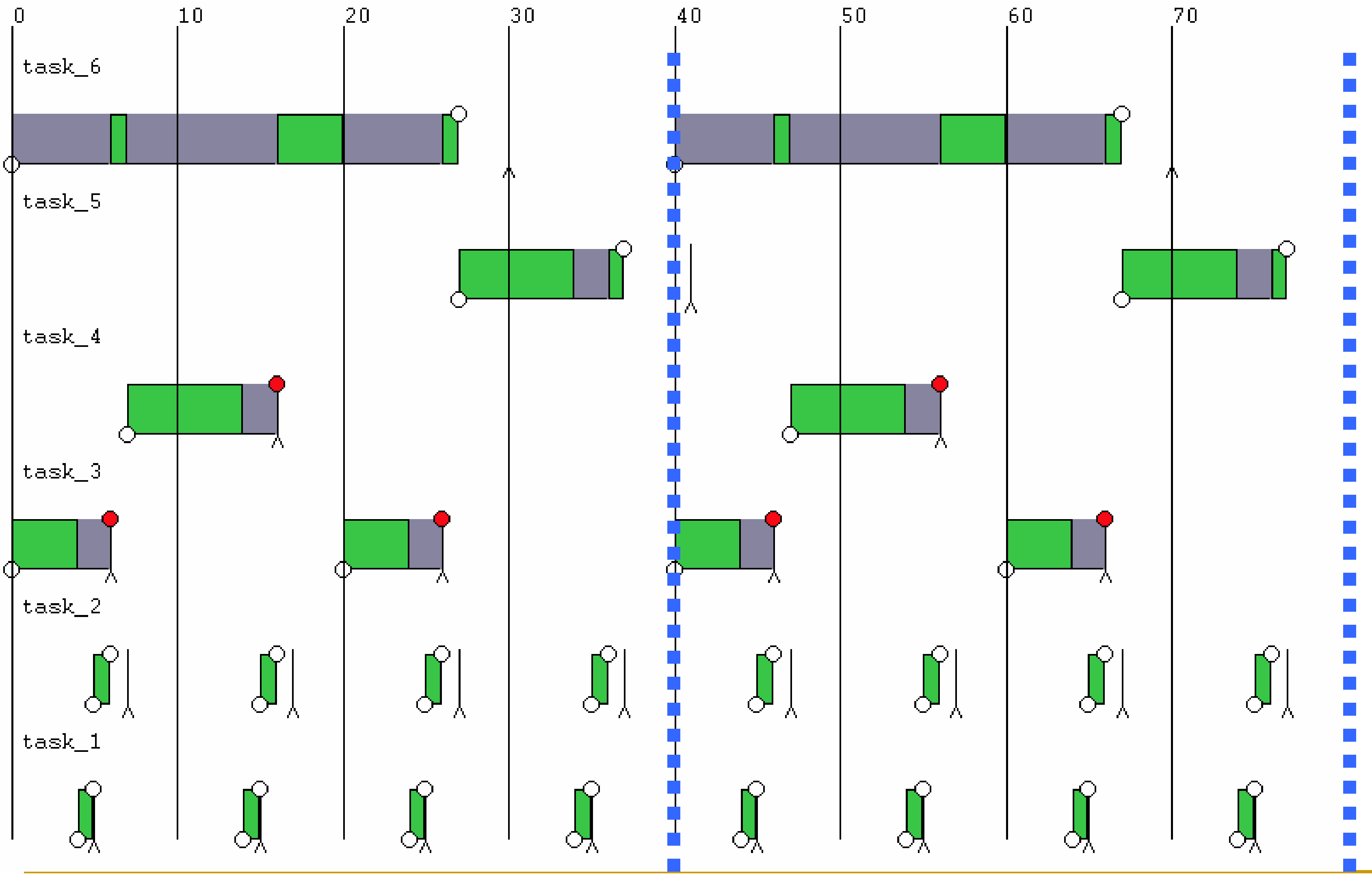
```
CreatedInterference ()
  begin
    next_free = R_i^t + t
    K_i^t = 0
    total_created = R_i^t
    -- create and order tuple set η
    -- for this release of τ_i
    for (C, t_r) in η
        total_created = total_created + C
        if (next_free < t_r) then
            next_free = t_r
        endif
        K_i^t = K_i^t + min (t + D_i - next_free, C)
        next_free = min (t + D_i, next_free + C)
    endfor
    L_i^{t+Ti} = total_created - create - max(D_i, R_i^t)
  end
```

Example #8

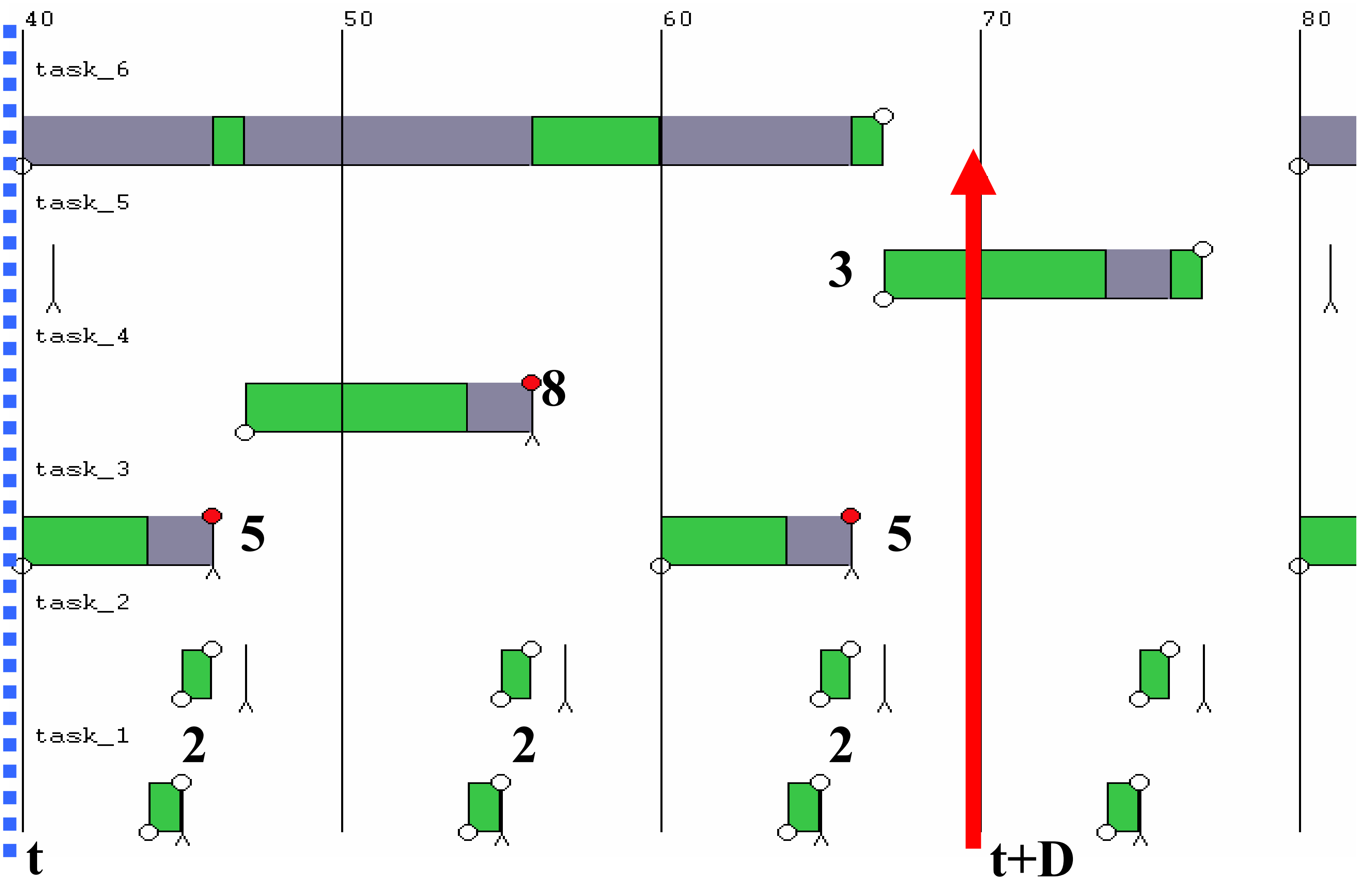
Task τ_i	Period T_i	Deadline D_i	Run-Time C_i	Phase φ_i
τ_A	10	1	1	4
τ_B	10	2	1	5
τ_C	20	6	5	0
τ_D	40	9	8	7
τ_E	40	14	8	27
τ_F	40	30	6	0


0
20
40
60
80
task_6
task_5
task_4
task_3
task_2
task_1
missed
deadlines

Remaining Interference ($R_i^t = 0$)



Created Interference ($K_i^t = 27$)



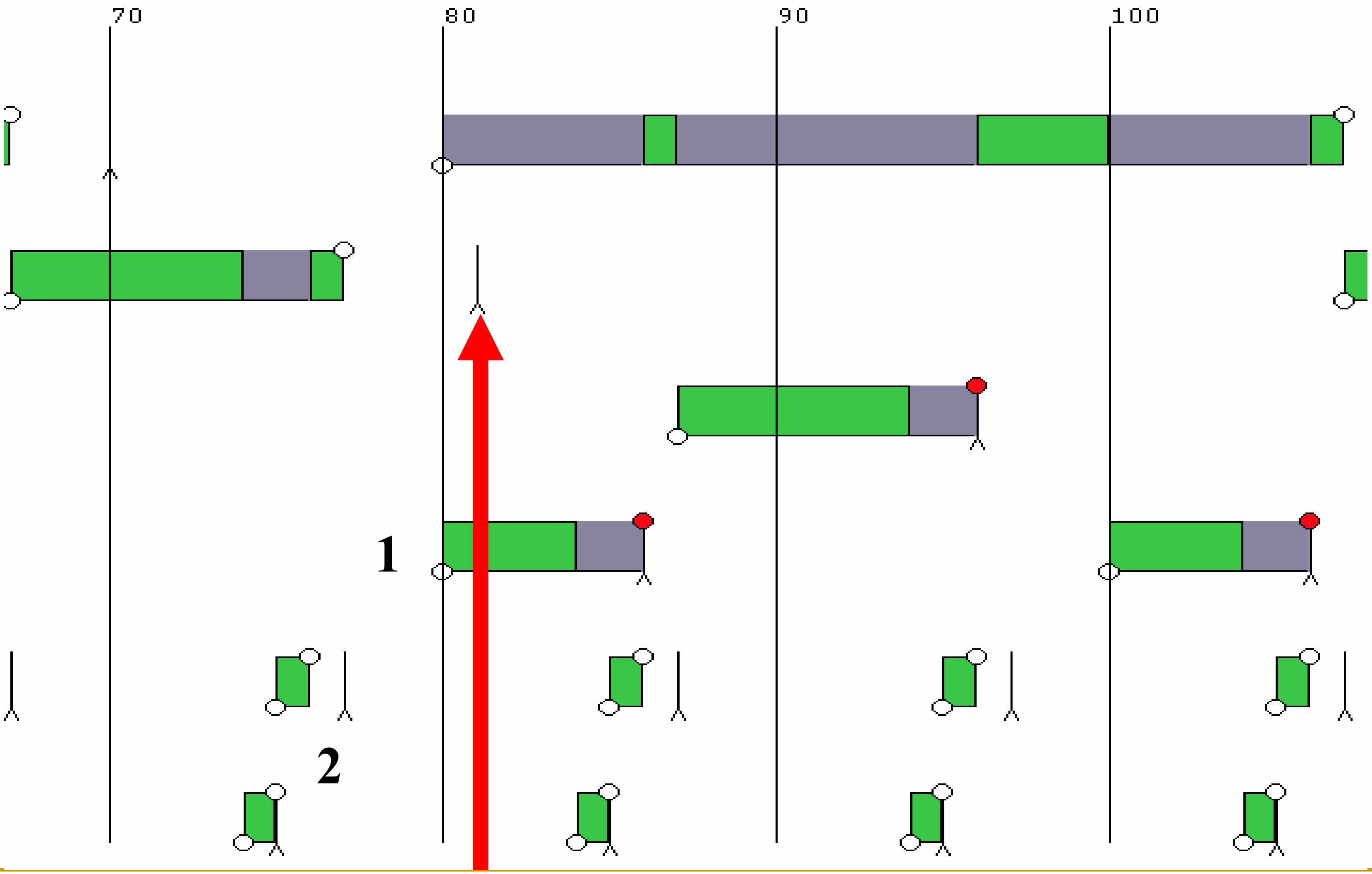
Try task_5 (T_E)

```
periodic task_4
  period 40 deadline 9 offset 7
  priority 4
  [8,8]
endper

periodic task_5
  period 40 deadline 14 offset 27
  priority 6
  [8,8]
endper

periodic task_6
  period 40 deadline 30 offset 0
  priority 5
  [6,6]
endper
```

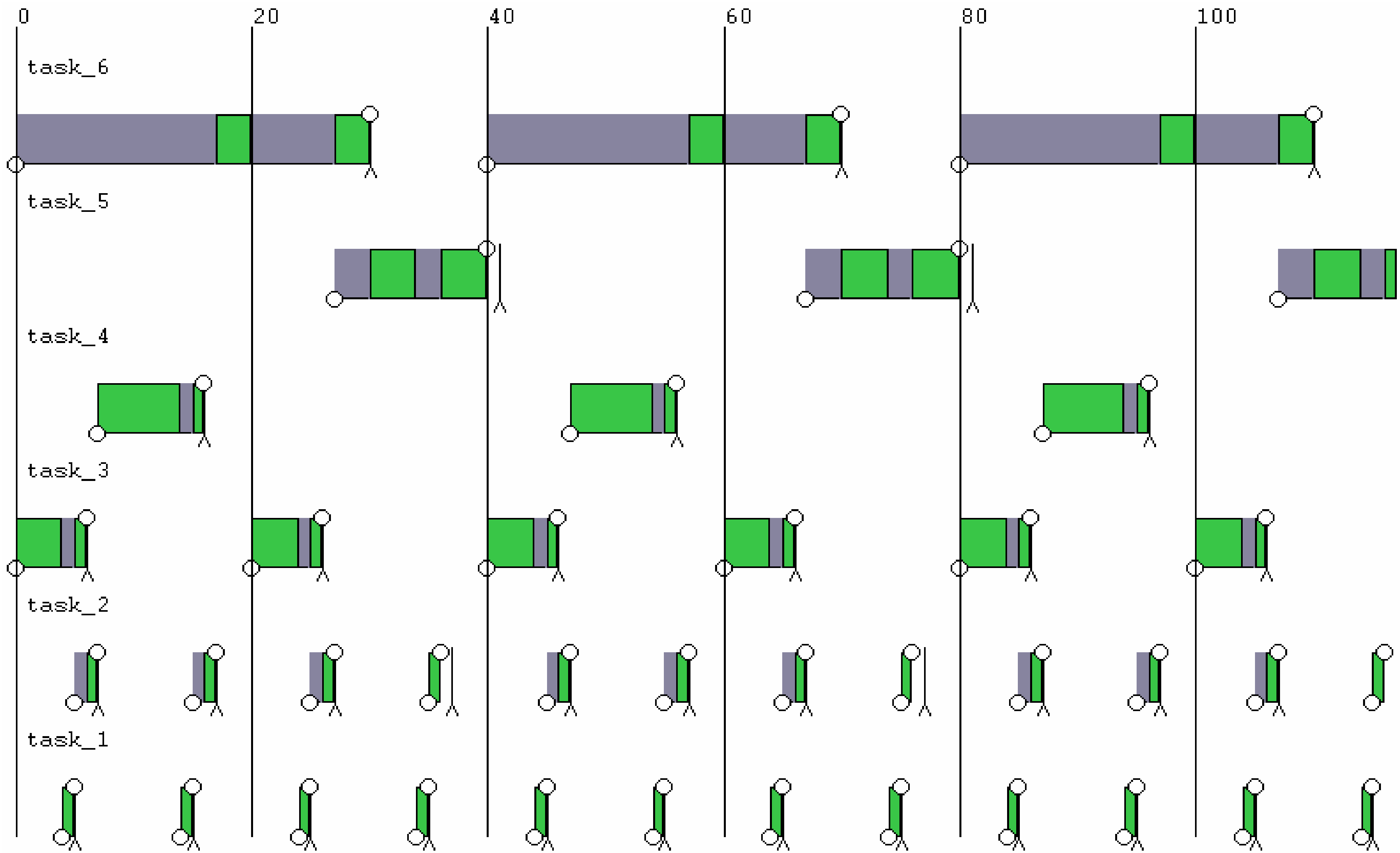
Interference ($R_i^t = 3$, $K_i^t = 3$)



Example #8 – Updated Priorities

Task(Prio) τ_i (π_i)	Period T_i	Deadline D_i	Run-Time C_i	Phase φ_i
τ_A (1)	10	1	1	4
τ_B (4)	10	2	1	5
τ_C (2)	20	6	5	0
τ_D (3)	40	9	8	7
τ_E (6)	40	14	8	27
τ_F (5)	40	30	6	0

Feasible Assignment



Arbitrary Deadlines

- J.P. Lehoczky, “Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines”, In Proceedings of IEEE Real-Time Systems Symposium, pp. 201-209, December, 1990.
- K. Tindell, A. Burns, and A.J. Wellings, “An extensible approach for analysing fixed priority hard real-time tasks”, Real-Time Systems, 6 (2), pp. 133-151, 1994.

Summary

- Program #1
- Homework #2
- Read Audsley’s paper on scheduling with arbitrary start times, and Lehoczky’s paper on scheduling with arbitrary deadlines.