THE UNIVERSITY of York

Department of Computer Science

Chapter 11: Scheduling Real-Time Systems

Lecture plan

- 1st hour: Introduce scheduling
  - Fixed priority scheduling
  - Utilization test
  - Response time test
  - Priority inversion
  - Priority inheritance/ceiling

- Rest: Guidance session ex. 9

Scheduling

- In general, a scheduling scheme provides two features:
  - An algorithm for ordering the use of system resources (in particular the CPUs)
  - A means of predicting the worst-case behaviour of the system when the scheduling algorithm is applied
- The prediction can then be used to confirm the temporal requirements of the application

Task-Based Scheduling

- Tasks exist at run-time
  - Supported by real-time OS or run-time
- Each task is:
  - Runnable (and possible running), or
  - Suspended waiting for a timing event
  - Suspended waiting for a non-timing event

Task-Based Scheduling

- Scheduling approaches
  - Fixed-Priority Scheduling (FPS)
  - Earliest Deadline First (EDF)
  - Value-Based Scheduling (VBS)

Fixed-Priority Scheduling (FPS)

- This is the most widely used approach and is the main focus of this course
- Each task has a fixed, static, priority which is computer pre-run-time
- The runnable tasks are executed in the order determined by their priority
- In real-time systems, the "priority" of a task is derived from its temporal requirements, not its importance to the correct functioning of the system or its integrity

Preemption

- With priority-based scheduling, a high-priority task may be released during the execution of a lower priority one
- In a preemptive scheme, there will be an immediate switch to the higher-priority task
- With non-preemption, the lower-priority task will be allowed to complete before the other executes
- Preemptive schemes enable higher-priority tasks to be more reactive, and hence they are preferred
- Cooperative dispatching (deferred preemption) is a half-way house
- Schemes such as EDF and VBS can also take on a pre-emptive or non pre-emptive form

Scheduling Characteristics

- Sufficient – pass the test will meet deadlines
- Necessary – fail the test will miss deadlines
- Exact – necessary and sufficient
- Sustainable – system stays schedulable if conditions 'improve'

Simple Task Model

- The application is assumed to consist of a fixed set of tasks
- All tasks are periodic, with known periods
- The tasks are completely independent of each other
- All system's overheads, context-switching times and so on are ignored (i.e, assumed to have zero cost)
- All tasks have a deadline equal to their period (that is, each task must complete before it is next released)
- All tasks have a fixed worst-case execution time

Standard Notation

C	Worst-case computation time (WCET) of the task
D	Deadline of the task
I	The interference time of the task
N	Number of tasks in the system
P	Priority assigned to the task (if applicable)
R	Worst-case response time of the task
T	Minimum time between task releases, jobs, (task period)
U	The utilization of each task (equal to C/T)
a-z	The name of a task

Rate Monotonic Priority Assignment

- Each task is assigned a (unique) priority based on its period; the shorter the period, the higher the priority
- i.e, for two tasks `i` and `j`,

$$T_i < T_j \Rightarrow P_i > P_j$$

- This assignment is optimal in the sense that if any task set can be scheduled (using pre-emptive priority-based scheduling) with a fixed-priority assignment scheme, then the given task set can also be scheduled with a rate monotonic assignment scheme
- Note, priority 1 is the lowest (least) priority

Example Priority Assignment

Process	Period, T	Priority, P
a	25	5
b	60	3
c	42	4
d	105	1
e	75	2

Utilization-Based Analysis

- For D=T task sets only
- A simple sufficient but not necessary schedulability test exists

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N(2^{1/N} - 1)$$

$$U \leq 0.69 \text{ as } N \to \infty$$

Utilization Bounds

N	Utilization bound
1	100.0%
2	82.8%
3	78.0%
4	75.7%
5	74.3%
10	71.8%

Approaches 69.3% asymptotically

Task Set A

Task	Period T	Computation Time C	Priority P	Utilization U
a	50	12	1	0.24
b	40	10	2	0.25
c	30	10	3	0.33

- $U = \frac{12}{50} + \frac{10}{40} + \frac{10}{30} \leq 3\left(2^{1/3} - 1\right)$

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N\left(2^{1/N} - 1\right)$$

Task Set A

Task	Period T	Computation Time C	Priority P	Utilization U
a	50	12	1	0.24
b	40	10	2	0.25
c	30	10	3	0.33

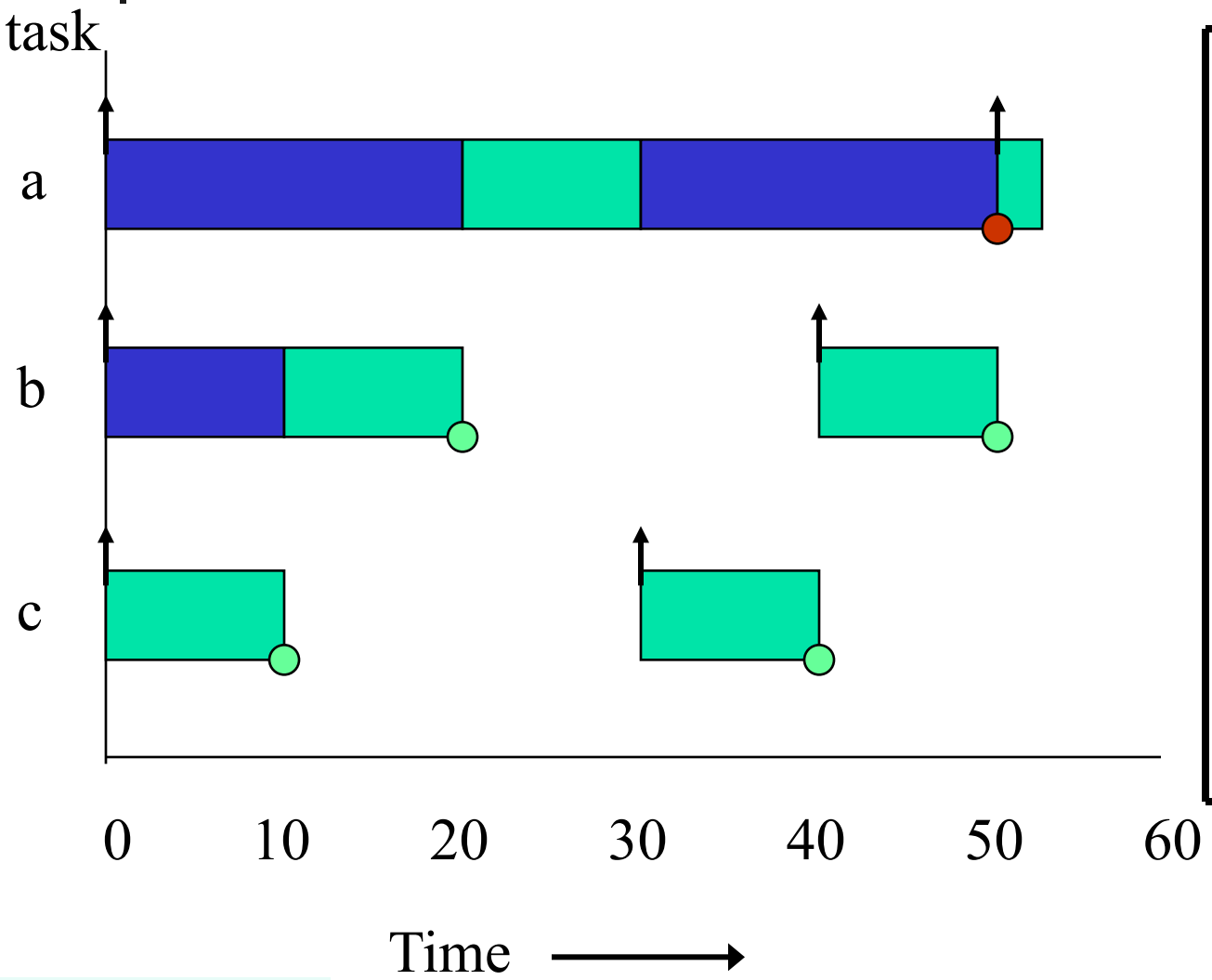
- $U = \frac{12}{50} + \frac{10}{40} + \frac{10}{30} \leq 3\left(2^{1/3} - 1\right)$
- $U = 0.82 \leq 0.78$

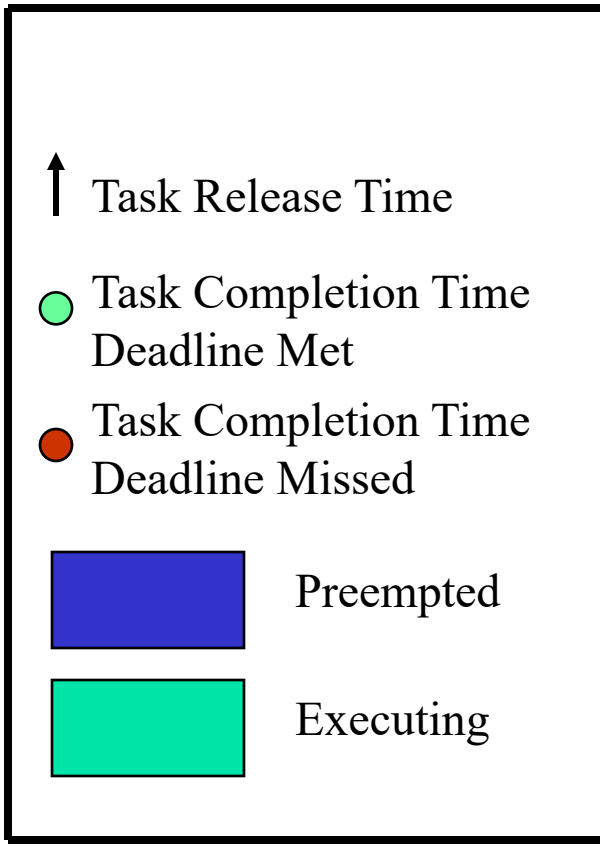
Task Set A

Task	Period T	Computation Time C	Priority P	Utilization U
a	50	12	1	0.24
b	40	10	2	0.25
c	30	10	3	0.33

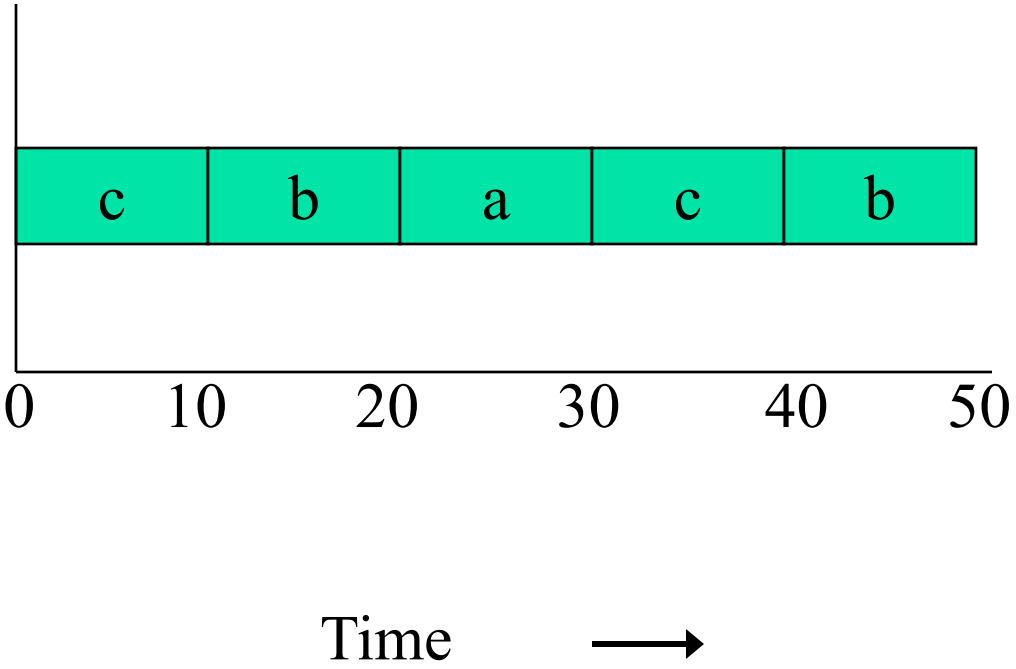
- $U = \frac{12}{50} + \frac{10}{40} + \frac{10}{30} \leq 3\left(2^{1/3} - 1\right)$
- $U = 0.82 \leq 0.78$
- Test fails

Time-line for task Set A





Gantt Chart for Task Set A



Task Set B

Task	Period T	Computation Time C	Priority P
a	80	32	1
b	40	5	2
c	16	4	3

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N(2^{1/N} - 1)$$

Task Set B

Task	Period T	Computation Time C	Priority P	Utilization U
a	80	32	1	0.400
b	40	5	2	0.125
c	16	4	3	0.250

- $U = \frac{32}{80} + \frac{5}{40} + \frac{4}{16} \leq 3\left(2^{1/3} - 1\right)$
- $U = 0.775 \leq 0.78$
- Test passes

Task Set C

Task	Period T	Computation Time C	Priority P	Utilization U
a	80	40	1	
b	40	10	2	
c	20	5	3	

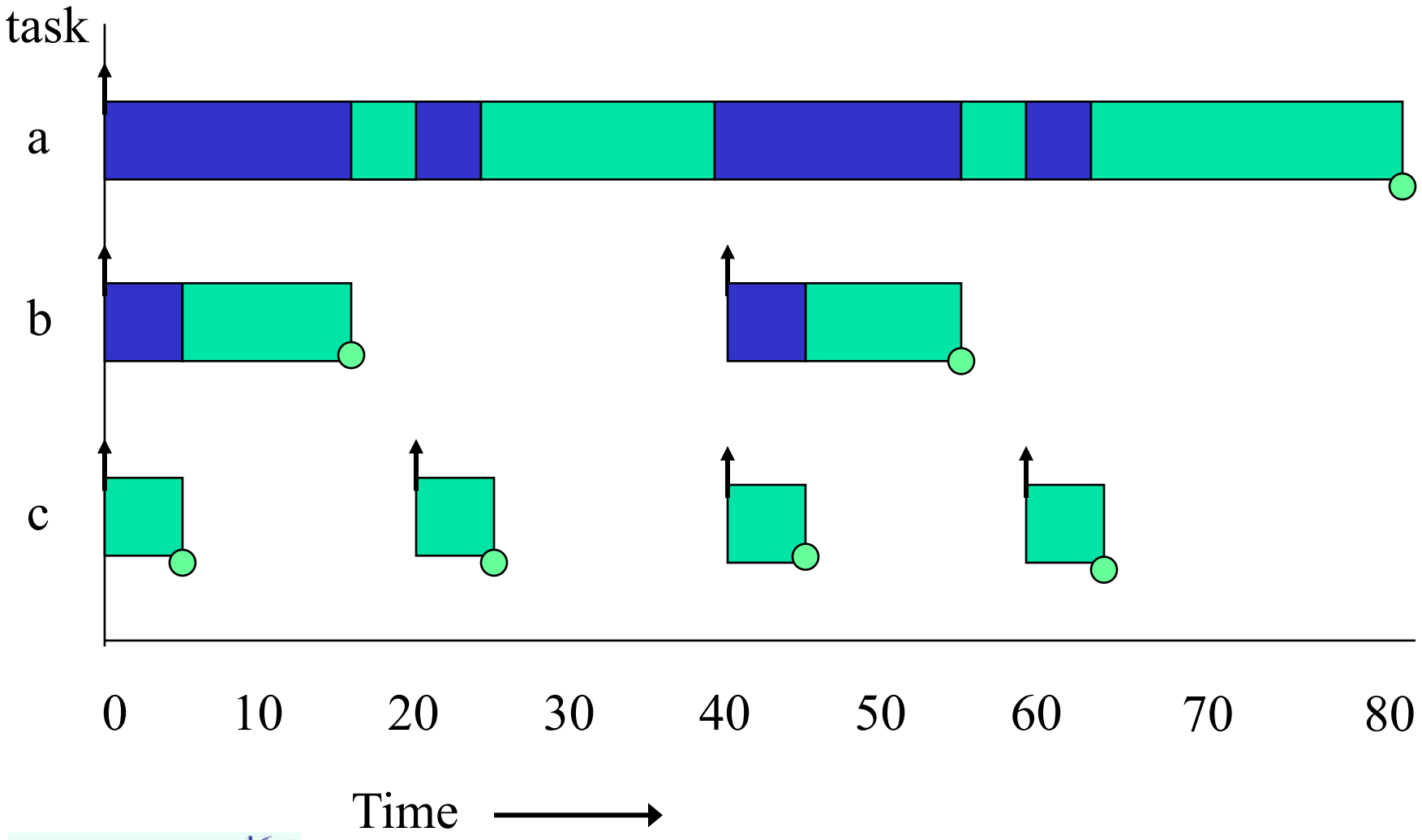
$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N(2^{1/N} - 1)$$

Task Set C

Task	Period T	Computation Time C	Priority P	Utilization U
a	80	40	1	0.50
b	40	10	2	0.25
c	20	5	3	0.25

- The combined utilization is 1.0
- This is above the threshold for three tasks (0.78), test fails

Time-line for Task Set C



Criticism of Tests

- Not exact
- Not general
- BUT it is O(N)

The test is sufficient but not necessary

Response-Time Analysis

- Here task i's worst-case response time, R, is calculated first and then checked (trivially) with its deadline

$$R_i \leq D_i$$

$$R_i = C_i + I_i$$

Where I is the interference from higher priority tasks

Calculating R

During R, each higher priority task j will execute a number of times:

$$\text{Number of Releases} = \left\lceil \frac{R_i}{T_j} \right\rceil$$

The ceiling function $\lceil \ \rceil$ gives the smallest integer greater than the fractional number on which it acts. So the ceiling of 1/3 is 1, of 6/5 is 2, and of 6/3 is 2.

Total interference is given by:

$$\left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Response Time Equation

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Where $hp(i)$ is the set of tasks with priority higher than task i

Solve by forming a recurrence relationship:

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

The set of values $w_i^0, w_i^1, w_i^2, ..., w_i^n, ..$ is monotonically non decreasing. When $w_i^n = w_i^{n+1}$ the solution to the equation has been found; w_i^0 must not be greater that R_i (e.g. 0 or C_i)

Task Set D

Task	Period T	Computation Time C	Priority P
a	7	3	3
b	12	3	2
c	20	5	1

$$R_a = 3$$

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

Task Set D

Task	Period T	Computation Time C	Priority P
a	7	3	3
b	12	3	2
c	20	5	1

$$w_b^0 = 3$$

$$w_b^1 = 3 + \left\lceil \frac{3}{7} \right\rceil 3 = 6$$

$$w_b^2 = 3 + \left\lceil \frac{6}{7} \right\rceil 3 = 6$$

$$R_b = 6$$

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

$$w_c^0 = 5$$

$$w_c^1 = 5 + \left\lceil \frac{5}{7} \right\rceil 3 + \left\lceil \frac{5}{12} \right\rceil 3 = 11$$

$$w_c^2 = 5 + \left\lceil \frac{11}{7} \right\rceil 3 + \left\lceil \frac{11}{12} \right\rceil 3 = 14$$

$$w_c^3 = 5 + \left\lceil \frac{14}{7} \right\rceil 3 + \left\lceil \frac{14}{12} \right\rceil 3 = 17$$

$$w_c^4 = 5 + \left\lceil \frac{17}{7} \right\rceil 3 + \left\lceil \frac{17}{12} \right\rceil 3 = 20$$

$$w_c^5 = 5 + \left\lceil \frac{20}{7} \right\rceil 3 + \left\lceil \frac{20}{12} \right\rceil 3 = 20$$

$$R_c = 20$$

Revisit: Task Set C

Process	Period T	Computation Time C	Priority P	Response time R
a	80	40	1	
b	40	10	2	
c	20	5	3	

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

Revisit: Task Set C

Process	Period T	Computation Time C	Priority P	Response time R
a	80	40	1	80
b	40	10	2	15
c	20	5	3	5

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

Response Time Analysis

- Is sufficient and necessary (exact)
- If the task set passes the test they will meet all their deadlines; if they fail the test then, at run-time, a task will miss its deadline (unless the computation time estimations themselves turn out to be pessimistic)

Task Interactions and Blocking

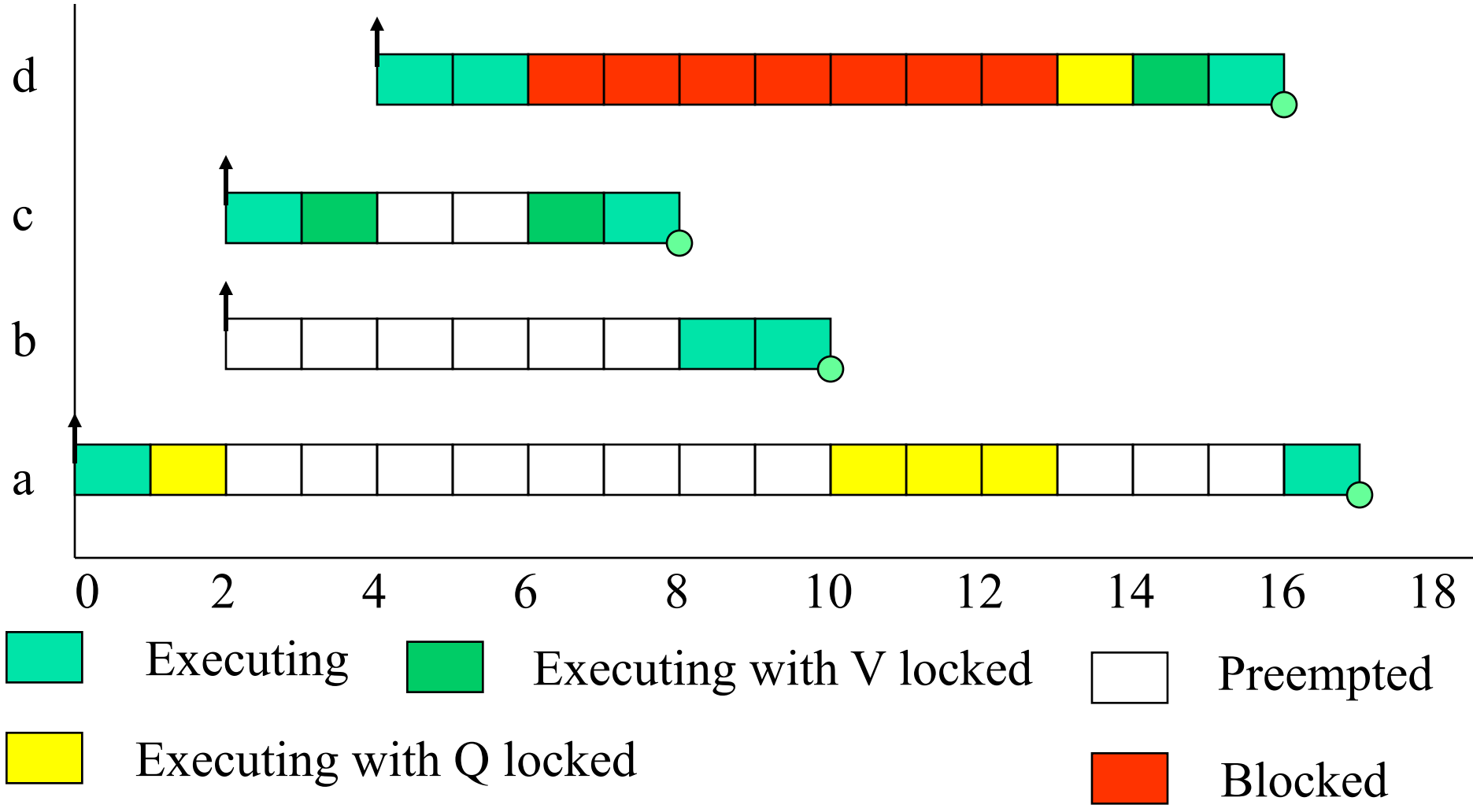
- If a task is suspended waiting for a lower-priority task to complete some required computation then the priority model is, in some sense, being undermined
- It is said to suffer priority inversion
- If a task is waiting for a lower-priority task, it is said to be blocked

Priority Inversion

- To illustrate an extreme example of priority inversion, consider the executions of four periodic tasks: `a`, `b`, `c` and `d`; and two resources: `Q` and `V`

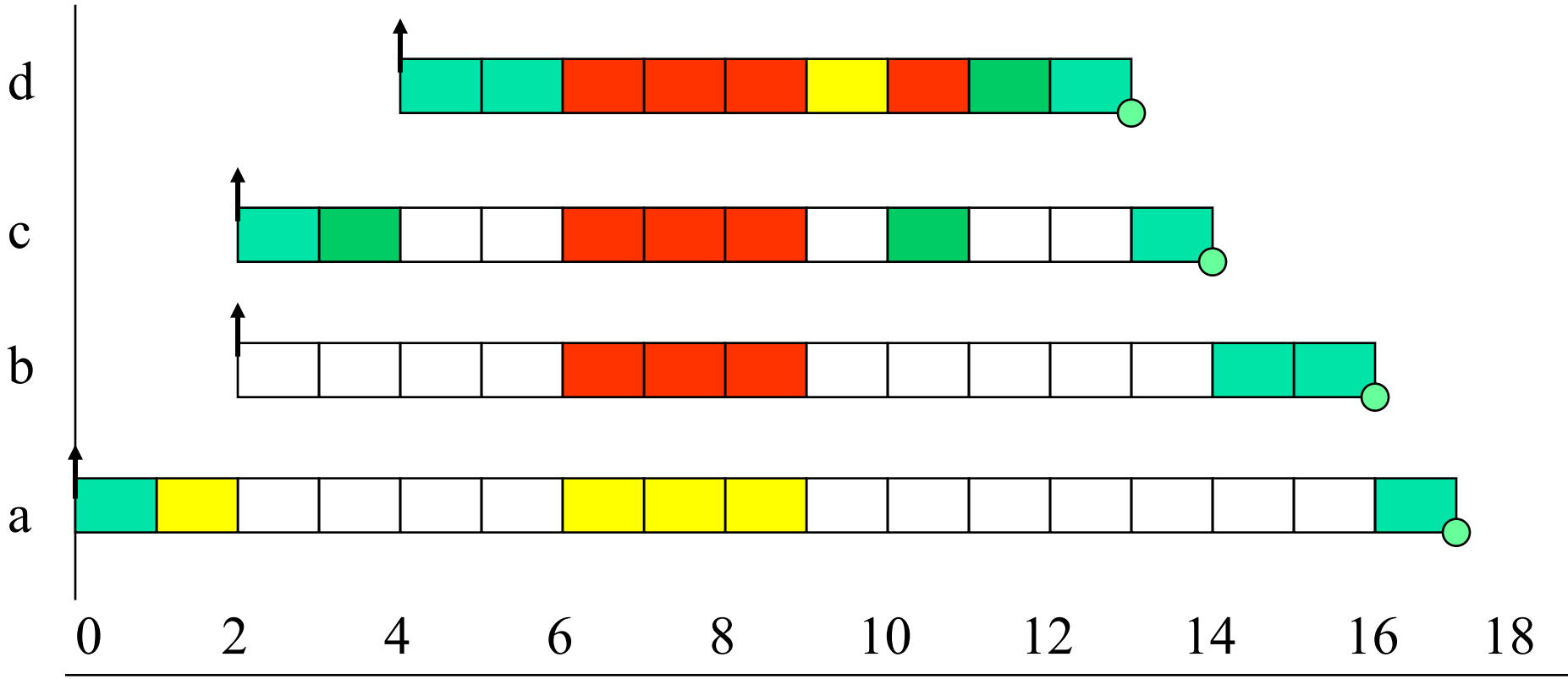
Task	Priority	Execution Sequence	Release Time
a	1	EQQQQE	0
b	2	EE	2
c	3	EVVE	2
d	4	EEQVE	4

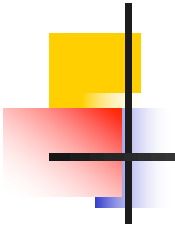
Example of Priority Inversion



Priority Inheritance

- If task `p` is blocking task `q`, then `p` runs with `q`'s priority





Priority Ceiling Protocols

Two forms

- Original ceiling priority protocol
- Immediate ceiling priority protocol

On a Single Processor

- A high-priority task can be blocked at most once during its execution by lower-priority tasks
- Deadlocks are prevented
- Transitive blocking is prevented
- Mutual exclusive access to resources is ensured (by the protocol itself)

OCPP

- Each task has a static default priority assigned (perhaps by the deadline monotonic scheme)
- Each resource has a static ceiling value defined, this is the maximum priority of the tasks that use it
- A task has a dynamic priority that is the maximum of its own static priority and any it inherits due to it blocking higher-priority tasks
- A task can only lock a resource if its dynamic priority is higher than the ceiling of any currently locked resource (excluding any that it has already locked itself)

ICPP

- Each task has a static default priority assigned (perhaps by the deadline monotonic scheme)
- Each resource has a static ceiling value defined, this is the maximum priority of the tasks that use it
- A task has a dynamic priority that is the maximum of its own static priority and the ceiling values of any resources it has locked

ICPP - Properties

- As a consequence of ICPP, a task will only suffer a block at the very beginning of its execution
- Once the task starts actually executing, all the resources it needs must be free; if they were not, then some task would have an equal or higher priority and the task's execution would be postponed

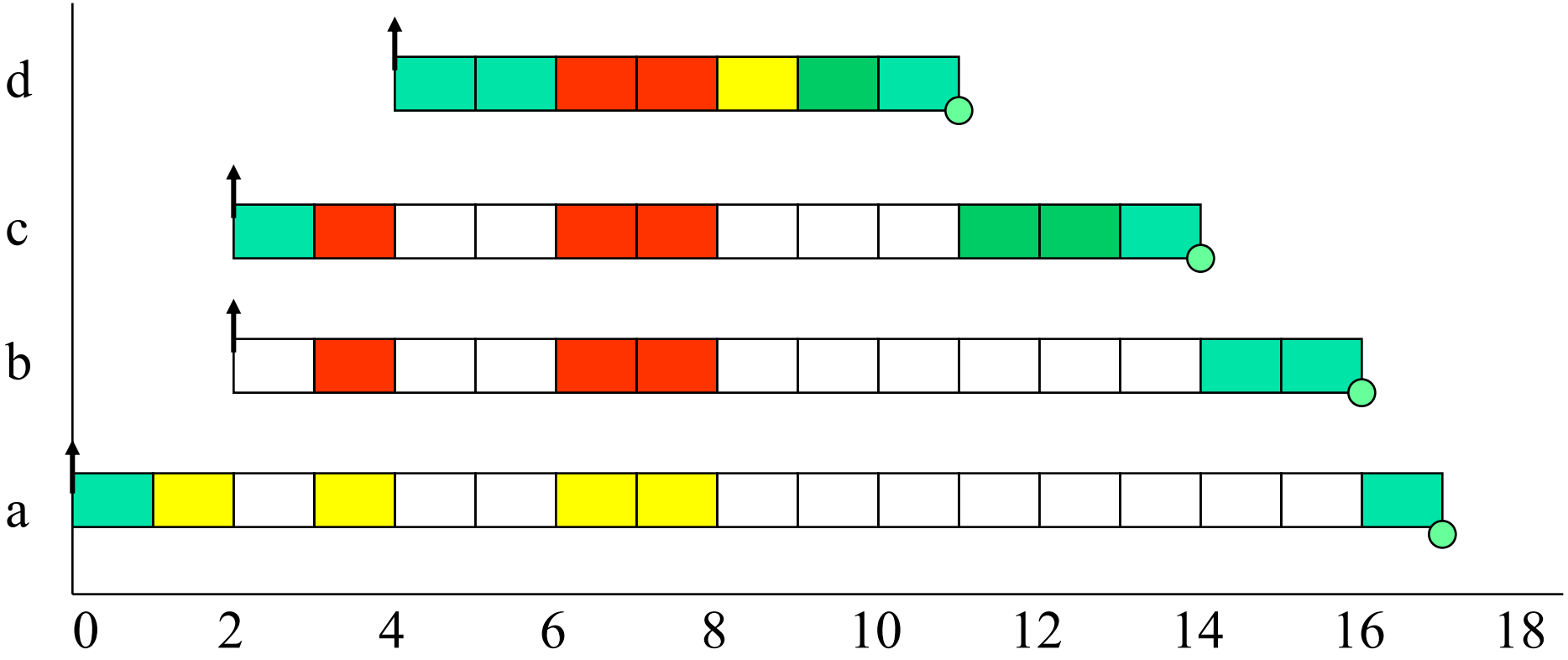
Priority ceiling protocols

- Protocol, not synchronization ensures mutual exclusive access
- Deadlock free due to priority ceiling

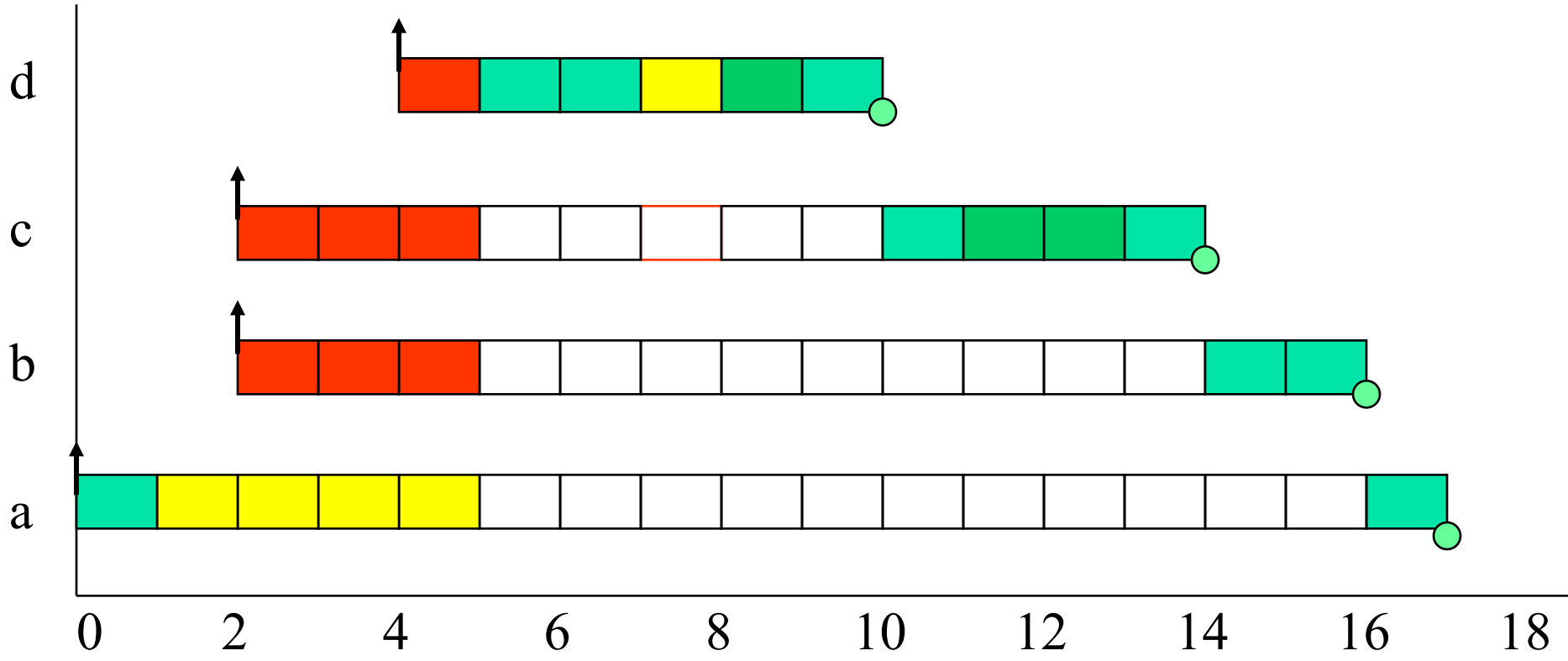
OCPP versus ICPP

- Although the worst-case behaviour of the two ceiling schemes is identical (from a scheduling view point), there are some points of difference:
  - ICCP is easier to implement than the original (OCPP) as blocking relationships need not be monitored
  - ICPP leads to less context switches as blocking is prior to first execution
  - ICPP requires more priority movements as this happens with all resource usage
  - OCPP changes priority only if an actual block has occurred

OCPP



ICPP Inheritance



Further reading

- Chapter 11 - Real-Time Systems and Programming Languages (Fourth Edition)