# Scheduling Containers Operations in **Cloud**

ECE 4150 – Cloud Computing

Spring 202 – Vijay Madisetti

(Material adapted from CMU's RMS Notes & Notes From the University of York (UK) Lectures – For Georgia Tech Internal Use Only – Do not distribute)

# Scheduling Containers in **DevSecOps**

***If you are running a data center and are DevSecOps operations's efficiency, you will need to master this module***

## Lecture aims

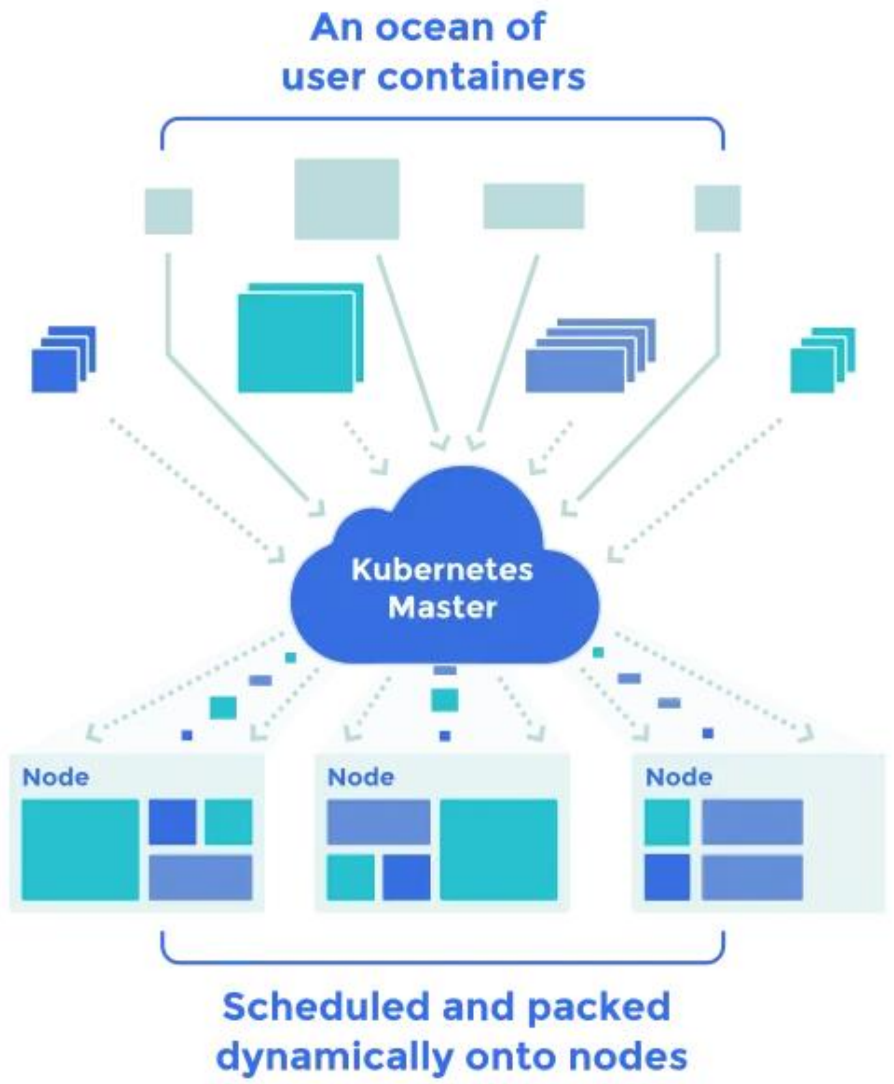
- To understand the role that scheduling and schedulability analysis plays in predicting that cloud-based containers meet their deadlines

## Topics

- Simple process model
- The cyclic executive approach
- Process-based scheduling
- Utilization-based schedulability tests
- Response time analysis for FPS and EDF
- Worst-case execution time
- Sporadic and aperiodic processes
- Process systems with $D < T$
- Process interactions, blocking and priority ceiling protocols
- Dynamic systems and on-line analysis

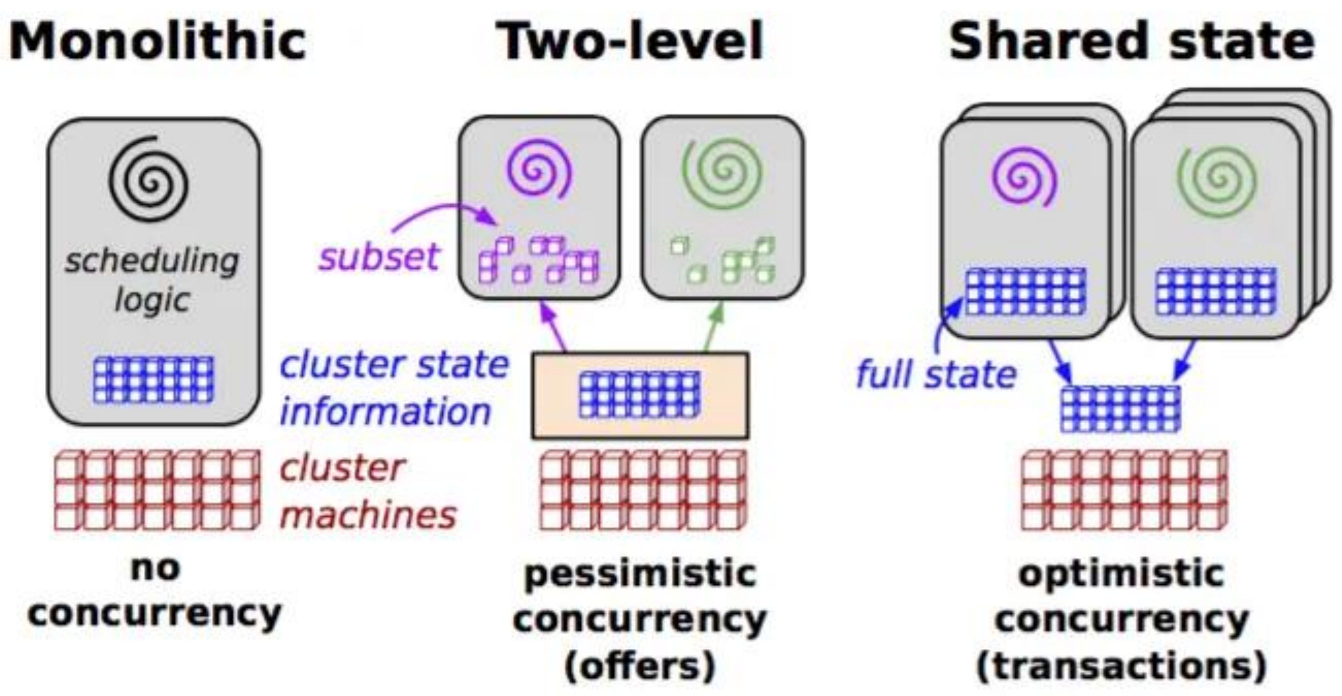
# Overall view of schedulers

- *Typical scale for a customer – 100 nodes/cpus with*
- *30 pods per node, and 2 containers per pod*
- *= 100*30*2 = 6000 pods*

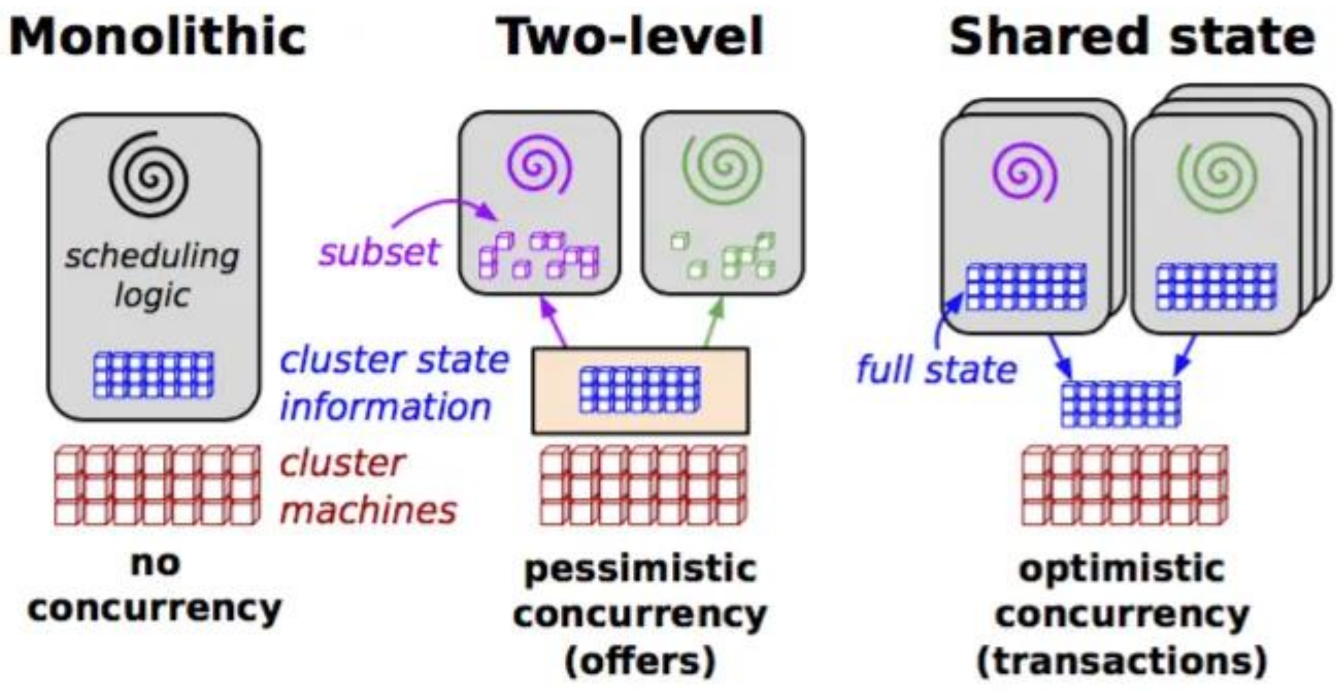



Kubernetes architecture (containers in grey, pods in color), © Google Inc. [31]

# Schedulers' Architecture



# Schedulers' Architecture




Mesos

Omega

# General Approach

- Spread tasks to unallocated and unassigned nodes (“spread” approach)
- Use constraints to customize scheduler
- User reservations and placement requests for special consideration
- Treat failure of nodes differently than performance optimization.

# Scheduling

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

# Simple Process Model for Containers

The cloud server is assumed to consist of a fixed set of processes corresponding to containers/pods

All processes are periodic, with known periods (or execution times that keep repeating)

The container processes are completely independent of each other

All server system's overheads, context-switching times and so on are ignored (i.e, assumed to have zero cost)

All container processes have a deadline equal to their period (that is, each process must complete before it is next released)

All processes have a fixed worst-case execution time (WCET)

# Standard Notation

- Worst-case blocking time for the process (if applicable)
- Worst-case computation time (WCET) of the process
- Deadline of the process
- The interference time of the process
- Number of processes in the system
- Priority assigned to the process (if applicable)
- Worst-case response time of the process
- Minimum time between process releases (process period)
- The utilization of each process (equal to C/T)
- The name of a process

# Cyclic Executives

- One common way of implementing scheduling containers is to use a cyclic executive
- Here the design is concurrent but the code is produced as a collection of procedures
- Procedures are mapped onto a set of minor cycles that constitute the complete schedule (or major cycle)
- Minor cycle dictates the minimum cycle time
- Major cycle dictates the maximum cycle time

Has the advantage of being fully deterministic

# Properties

No actual processes exist at run-time; each minor cycle is just a sequence of procedure calls

The procedures share a common address space and can thus pass data between themselves. This data does not need to be protected (via a semaphore, for example) because concurrent access is not possible

All “*process*” periods must be a multiple of the minor cycle time

# Problems with Cycle Executives

- The difficulty of incorporating processes with long periods; the major cycle time is the maximum period that can be accommodated without secondary schedules
- Sporadic activities are difficult (impossible!) to incorporate
- The cyclic executive is difficult to construct and difficult to maintain — it is a NP-hard problem
- Any “process” with a sizable computation time will need to be split into a fixed number of fixed sized procedures (this may cut across the structure of the code from a software engineering perspective, and hence may be error-prone)
- More flexible scheduling methods are difficult to support
- *Determinism is not required, but predictability is*

# Process-Based Scheduling

- Scheduling approaches for containers
  - 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 lecture

Each process has a fixed, static, priority which is computer pre-run-time

The runnable processes are executed in the order determined by their priority

In real-time systems, the “priority” of a process is derived from its temporal requirements, not its importance to the correct functioning of the system or its integrity

# FPS and Rate Monotonic Priority Assignment

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

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

- This assignment is optimal in the sense that if any process set can be scheduled (using pre-emptive priority-based scheduling) with a fixed-priority assignment scheme, then the given process 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

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

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

# Utilization Bounds

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

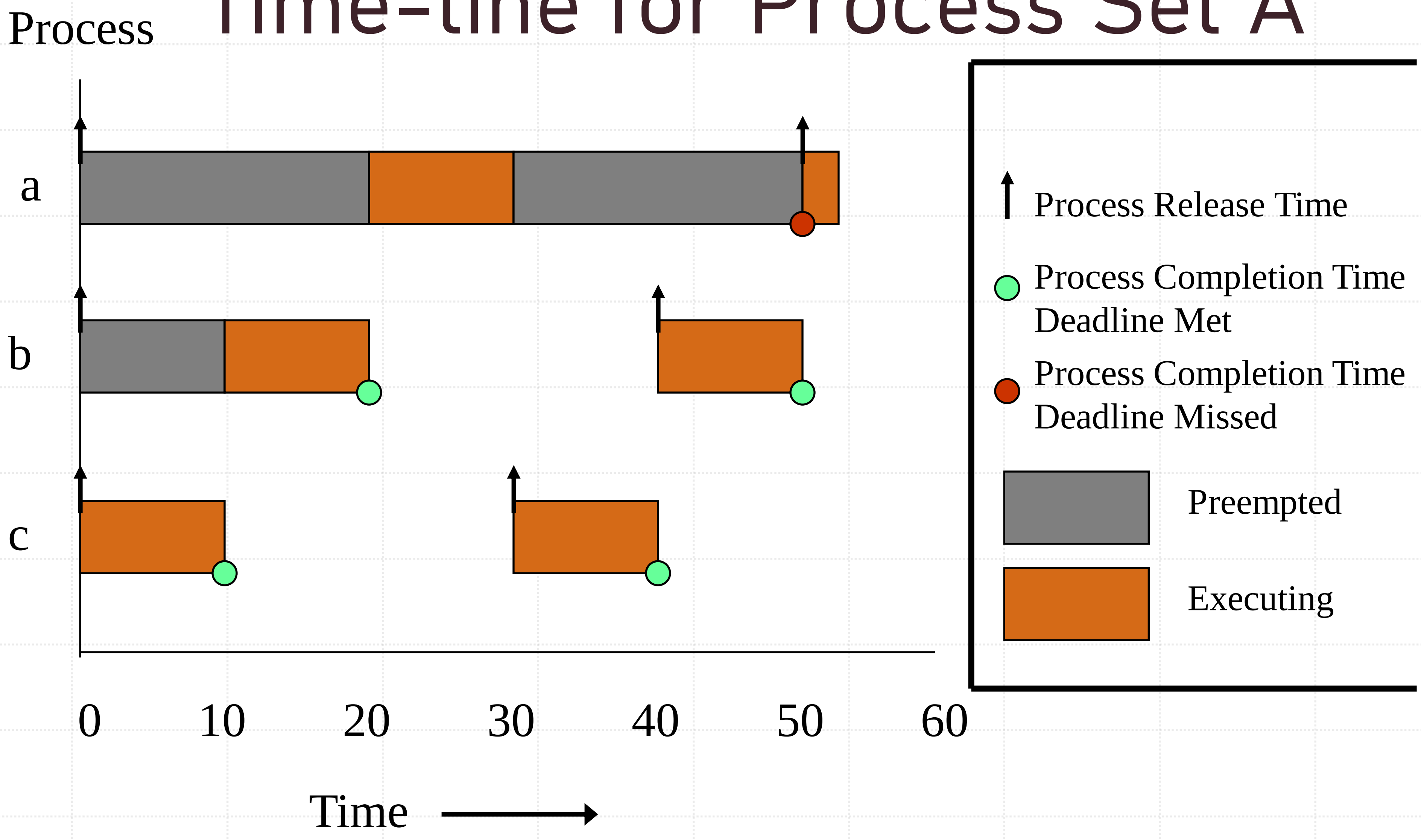
Approaches 69.3% asymptotically

# Process Set A

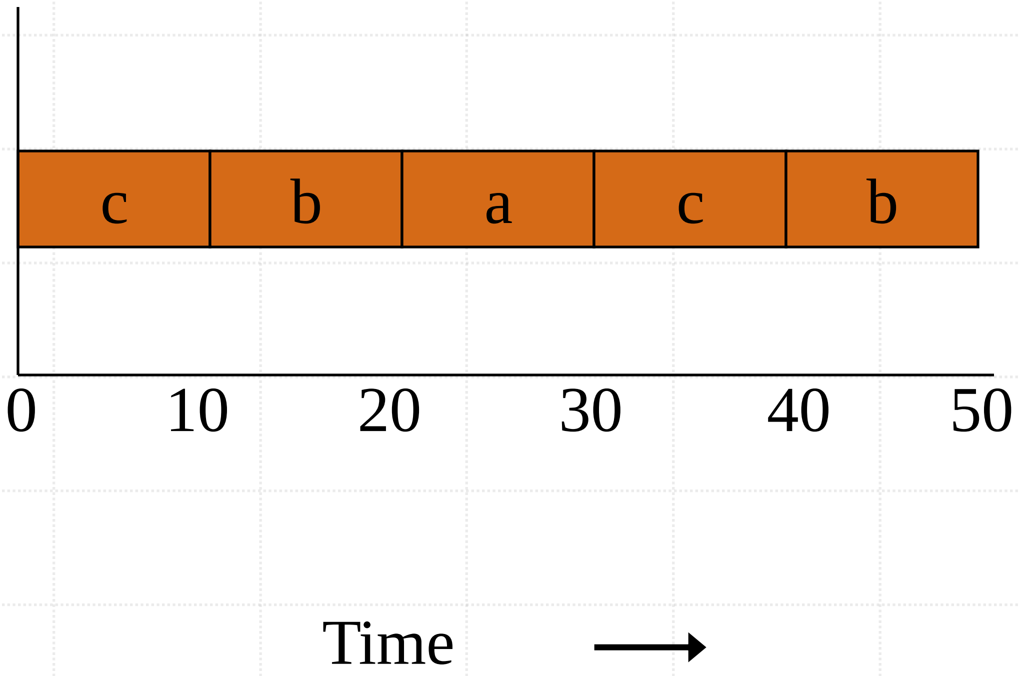
| Process | Period T | ComputationTime C | Priority P | Utilization U |
|---|---|---|---|---|
| a | 50 | 12 | 1 | 0.24 |
| b | 40 | 10 | 2 | 0.25 |
| c | 30 | 10 | 3 | 0.33 |

- The combined utilization is 0.82 (or 82%)
- This is above the threshold for three processes (0.78) and, hence, this process set fails the utilization test

# Time-line for Process Set A



# Gantt Chart for Process Set A



# Process Set B

| Process | Period T | ComputationTime C | Priority P | Utilization U |
|---|---|---|---|---|
| a | 80 | 32 | 1 | 0.400 |
| b | 40 | 5 | 2 | 0.125 |
| c | 16 | 4 | 3 | 0.250 |

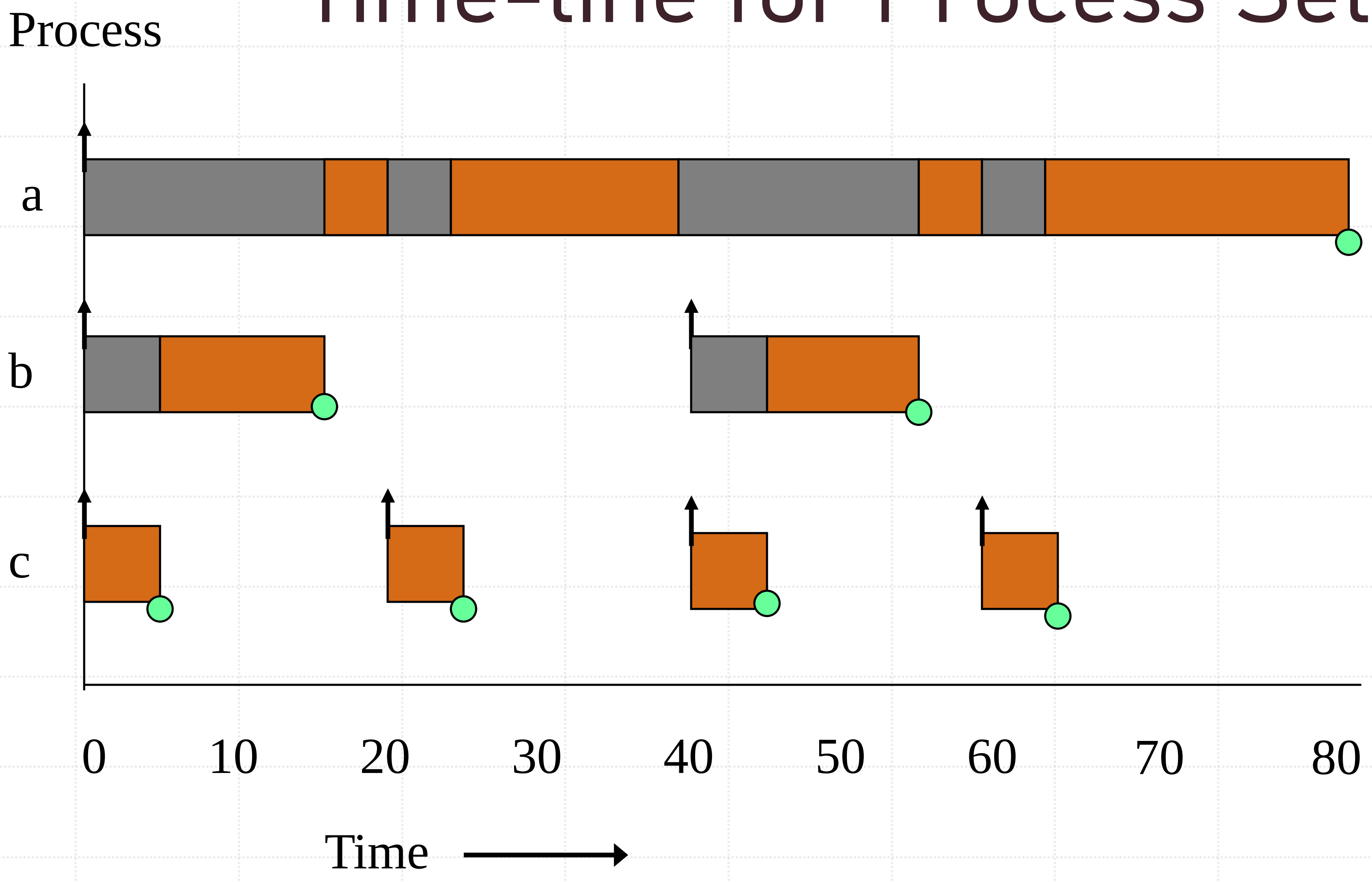
- The combined utilization is 0.775
- This is below the threshold for three processes (0.78) and, hence, this process set will meet all its deadlines

# Process Set C

| Process | Period T | ComputationTime 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 processes (0.78) but the process set will meet all its deadlines

# Time-line for Process Set C



# Criticism of Utilization-based Tests

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

The test is said to be sufficient but not necessary

# Response-Time Analysis (CMU's RTA)

- 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 $w_i^0, w_i^1, w_i^2, \ldots, w_i^n, \ldots$ 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 $C_i$ )

# Process Set D

| Process | Period T | ComputationTime C | Priority P |
|---|---|---|---|
| a | 7 | 3 | 3 |
| b | 12 | 3 | 2 |
| c | 20 | 5 | 1 |

$$R_a = 3$$

$$w_b^0 = 3$$

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

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

$$R_b = 6$$

$$w_c^0 = 5$$

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

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

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

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

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

$$R_c = 20$$

# Revisit: Process Set C

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

- The combined utilization is 1.0
- This was above the ulilization threshold for three processes (0.78), therefore it failed the test
- The response time analysis shows that the process set will meet all its deadlines
- RTA is necessary and sufficient

# Response Time Analysis

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

# Worst-Case Execution Time – WCET

- Obtained by either measurement or analysis
- The problem with measurement is that it is difficult to be sure when the worst case has been observed
- The drawback of analysis is that an effective model of the processor (including caches, pipelines, memory wait states and so on) must be available

# WCET— Finding C

Most analysis techniques involve two distinct activities.

- The *first* component takes the process and decomposes its code into a directed graph of basic blocks
- These basic blocks represent straight-line code
- The *second* component of the analysis takes the machine code corresponding to a basic block and uses the processor model to estimate its worst-case execution time
- Once the times for all the basic blocks are known, the directed graph can be collapsed

# Need for Semantic Information

```
for I in 1.. 10 loop
  if Cond then
    -- basic block of cost 100
  else
    -- basic block of cost 10
  end if;
end loop;
```

- Simple cost 10*100 (+overhead), say 1005.
- But if `Cond` only true 3 times, then cost is 375

# Sporadic Processes

- Sporadic processes have a minimum inter-arrival time
- They also require D<T

$$W_i^{n+1} > D_i$$

- The response time algorithm for fixed priority scheduling works perfectly for values of D less than T as long as the stopping criteria becomes
- It also works perfectly well with any priority ordering — hp(i) always gives the set of higher-priority processes

# Hard and Soft Processes

- In many situations the worst-case figures for sporadic processes are considerably higher than the averages
- Interrupts often arrive in bursts and an abnormal sensor reading may lead to significant additional computation
- Measuring schedulability with worst-case figures may lead to very low processor utilizations being observed in the actual running system

# General Guidelines

Rule 1 — all processes should be schedulable using average execution times and average arrival rates

Rule 2 — all hard real-time processes should be schedulable using worst-case execution times and worst-case arrival rates of all processes (including soft)

- A consequent of Rule 1 is that there may be situations in which it is not possible to meet all current deadlines
- This condition is known as a transient overload
- Rule 2 ensures that no hard real-time process will miss its deadline
- If Rule 2 gives rise to unacceptably low utilizations for "normal execution" then action must be taken to reduce the worst-case execution times (or arrival rates)

# Aperiodic Processes

- These do not have minimum inter-arrival times
- Can run aperiodic processes at a priority below the priorities assigned to hard processes, therefore, they cannot steal, in a pre-emptive system, resources from the hard processes
- This does not provide adequate support to soft processes which will often miss their deadlines
- To improve the situation for soft processes, a server can be employed.
- Servers protect the processing resources needed by hard processes but otherwise allow soft processes to run as soon as possible.
- POSIX supports Sporadic Servers

# Process Sets with D < T

- For D = T, Rate Monotonic priority ordering is optimal
- For D < T, Deadline Monotonic priority ordering is optimal

$$D_i < D_j \Rightarrow P_i > P_j$$

# D < T Example Process Set

| Process | Period T | Deadline D | ComputationTime C | Priority P | Response time R |
|---|---|---|---|---|---|
| a | 20 | 5 | 3 | 4 | 3 |
| b | 15 | 7 | 3 | 3 | 6 |
| c | 10 | 10 | 4 | 2 | 10 |
| d | 20 | 20 | 3 | 1 | 20 |

# Process Interactions and Blocking

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

# Response Time and Blocking

$$R_i = C_i + B_i + I_i$$

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

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

# Dynamic Systems and Online Analysis

- There are dynamic soft real-time applications in which arrival patterns and computation times are not known *a priori*
- Although some level of off-line analysis may still be applicable, this can no longer be complete and hence some form of on-line analysis is required
- The main task of an on-line scheduling scheme is to manage any overload that is likely to occur due to the dynamics of the system's environment
- EDF is a dynamic scheduling scheme that is an optimal
- During transient overloads EDF performs very badly. It is possible to get a cascade effect in which each process misses its deadline but uses sufficient resources to result in the next process also missing its deadline

# Admission Schemes

To counter this detrimental domino effect, many on-line schemes have two mechanisms:

- an admissions control module that limits the number of processes that are allowed to compete for the processors, and
- an EDF dispatching routine for those processes that are admitted

An ideal admissions algorithm prevents the processors getting overloaded so that the EDF routine works effectively

# Values

If some processes are to be admitted, whilst others rejected, the relative importance of each process must be known

This is usually achieved by assigning value

Values can be classified

- Static: the process always has the same value whenever it is released.
- Dynamic: the process's value can only be computed at the time the process is released (because it is dependent on either environmental factors or the current state of the system)
- Adaptive: here the dynamic nature of the system is such that the value of the process will change during its execution

To assign static values requires the domain specialists to articulate their understanding of the desirable behaviour of the system

# Summary

- A scheduling scheme defines an algorithm for resource sharing and a means of predicting the worst-case behaviour of a cloud application when that form of resource sharing is used.
- With a cyclic executive, the application code must be packed into a fixed number of minor cycles such that the cyclic execution of the sequence of minor cycles (the major cycle) will enable all system deadlines to be met
- The cyclic executive approach has major drawbacks many of which are solved by priority-based systems
- Simple utilization-based schedulability tests are not exact

# Summary

- Response time analysis is flexible and caters for:
  - Periodic and sporadic processes
  - Blocking caused by IPC
  - Cooperative scheduling (not covered)
  - Arbitrary deadlines (not covered)
  - Release jitter (not covered)
  - Fault tolerance (not covered)
  - Offsets (not covered)
- Containers support preemptive priority-based scheduling
- Containers address dynamic systems with the potential for on-line analysis