CISTER - Research Center in
Real-Time & Embedded Computing Systems

A Multi-DAG Model for Real-Time Parallel Applications with Conditional Execution

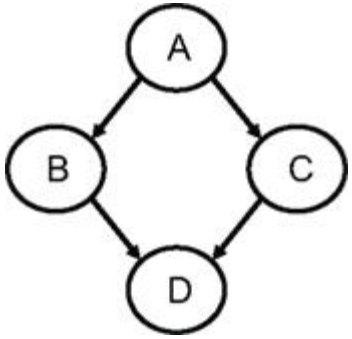
José Carlos Fonseca, Vincent Nélis,
Gurulingesh Raravi and Luís Miguel Pinho

Motivation

- New massively parallel architectures
  - Shift towards fine-grained parallel programming
  - Allow for HPC requirements/tighter deadlines

P-SOCRATES

- Current parallel task models do not explicitly account for conditional execution
  - Fork/join model
  - Synchronous parallel task model
  - Parallel DAG model



- It is unlikely to have a parallel application with no conditional statements

Motivation

- An application's code with a conditional statment

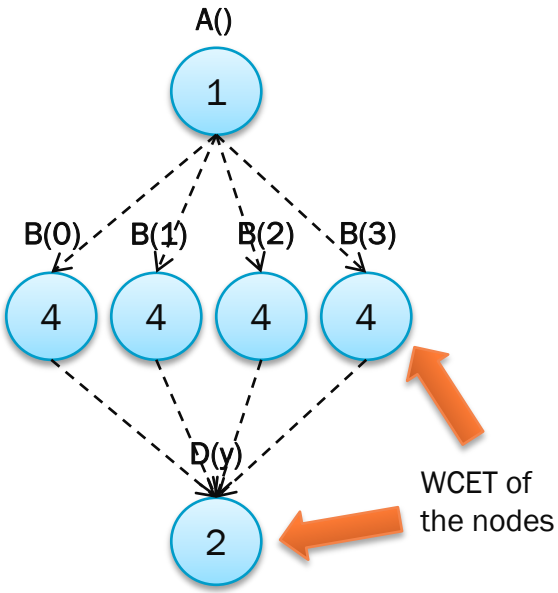
```
x = A();
y = 0;
if (x > 1) {
   #pragma omp parallel for reduction(+:y)
   for (i = 0; i < 4; i++)
      y += B(i);
} else {
   #pragma omp parallel for reduction(+:y)
   for (i = 0; i < 2; i++)
      y += C(i);
}
z = D(y)
```

Motivation

• An application's code with a conditional statment

```
x = A();
y = 0;
if (x > 1) {
   #pragma omp parallel for reduction(+:y)
   for (i = 0; i < 4; i++)
      y += B(i);
} else {
   #pragma omp parallel for reduction(+:y)
   for (i = 0; i < 2; i++)
      y += C(i);
}
z = D(y)
```

"if" path

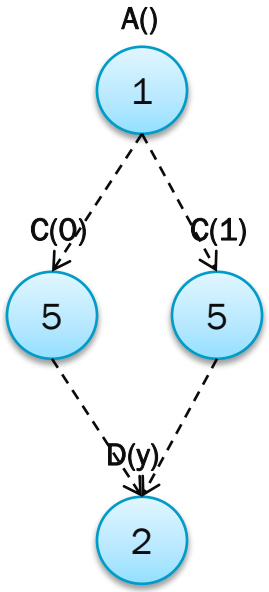


Motivation

- An application's code with a conditional statment

```
x = A();
y = 0;
if (x > 1) {
   #pragma omp parallel for reduction(+:y)
   for (i = 0; i < 4; i++)
      y += B(i);
} else {
   #pragma omp parallel for reduction(+:y)
   for (i = 0; i < 2; i++)
      y += C(i);
}
z = D(y)
```

"else" path

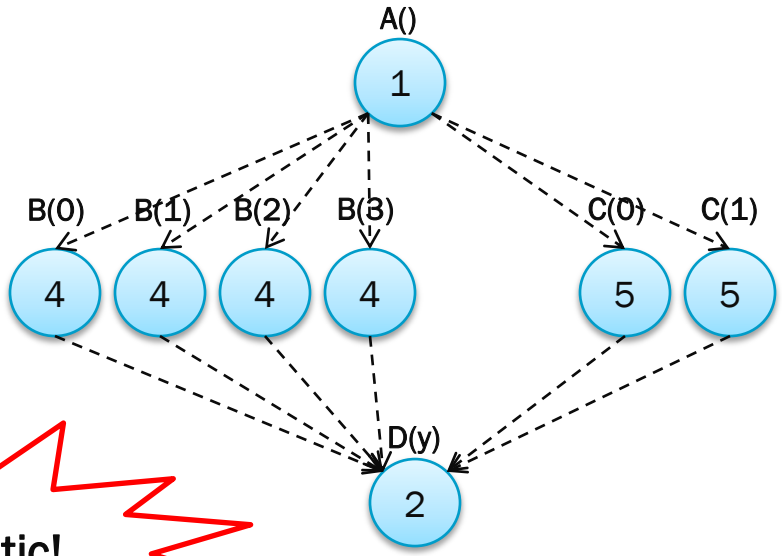


Motivation

- The two paths are mutual exclusive

- How would current parallel task models represent this application?

"if" path + "else" path =



Pessimistic!

Model

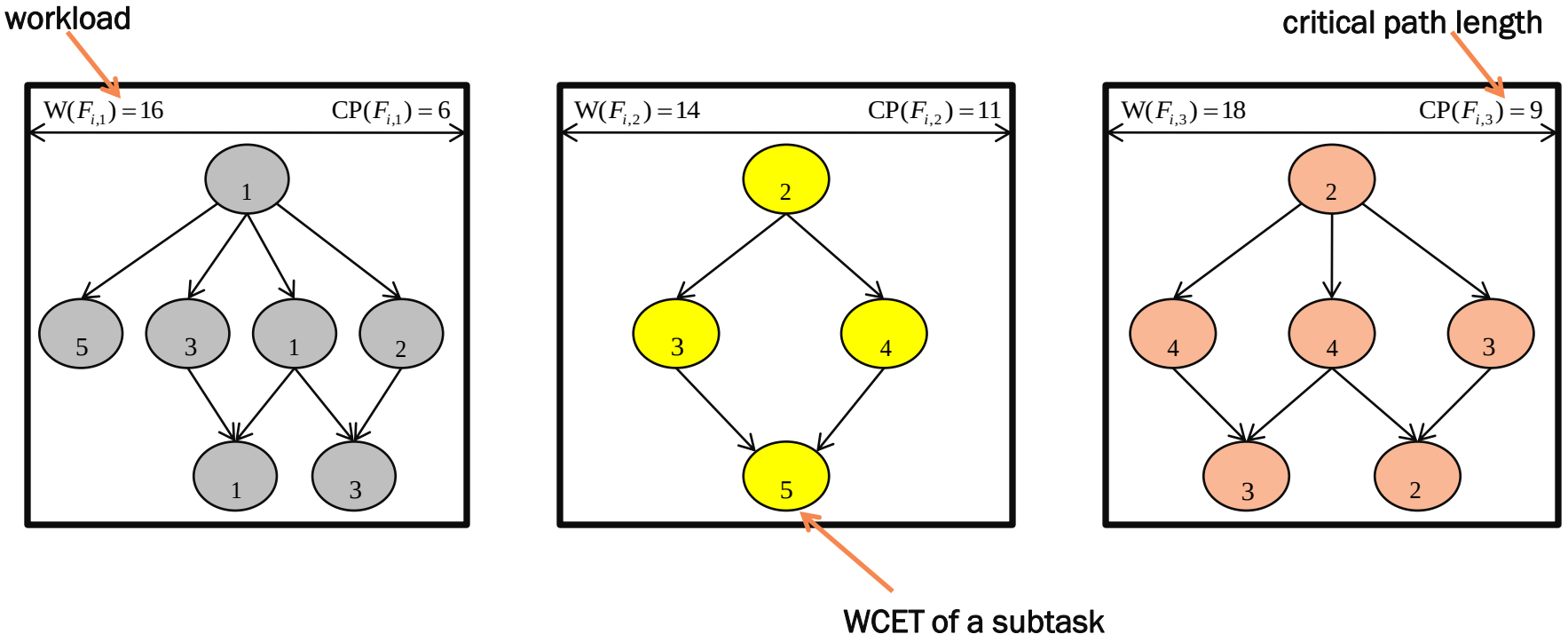
- Each task is comprised by a set of execution flows

Model

- Each task is comprised by a set of execution flows
- A execution flow is a DAG of computing units (subtasks), representing one of the paths that can be taken by a job throughout the task's execution

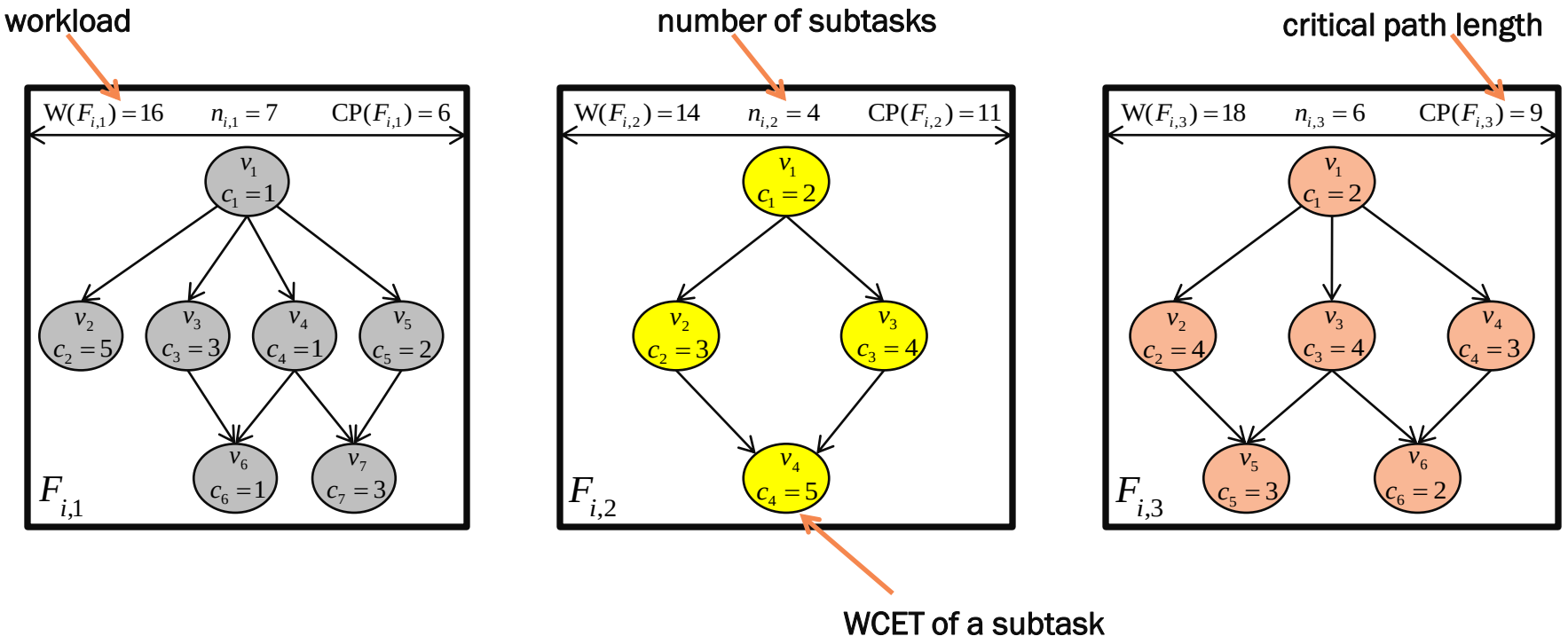
Model

- Each task is comprised by a set of execution flows
- A execution flow is a DAG of computing units (subtasks), representing one of the paths that can be taken by a job throughout the task's execution

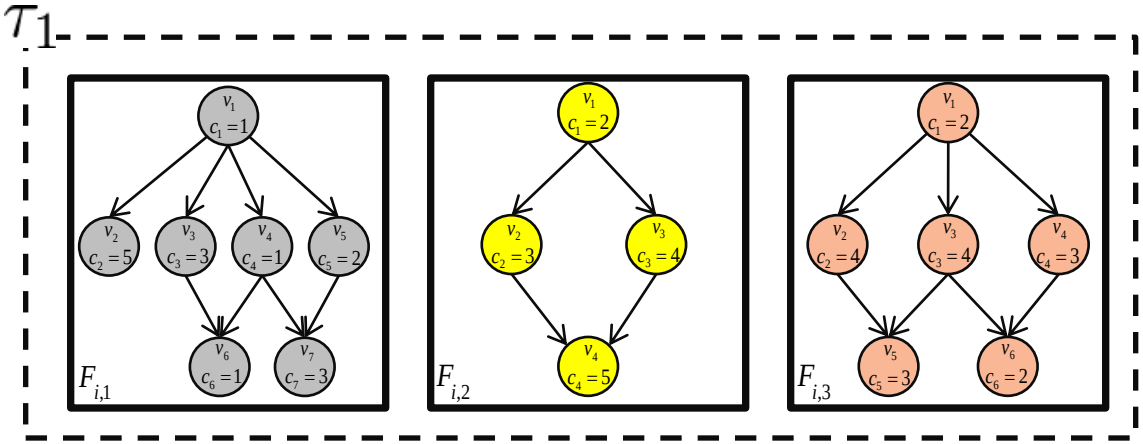


Model

- Each task is comprised by a set of execution flows
- A execution flow is a DAG of computing units (subtasks), representing one of the paths that can be taken by a job throughout the task's execution



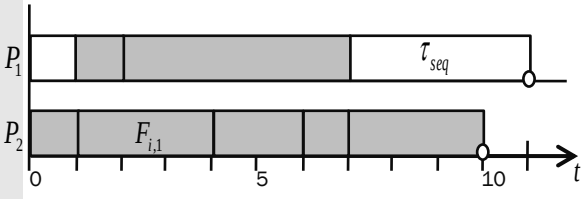
Scheduling Issue

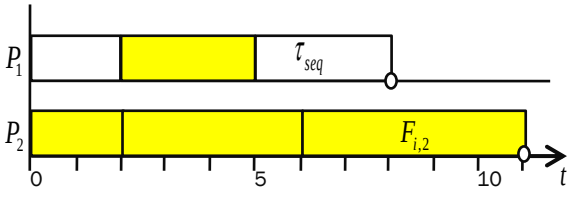


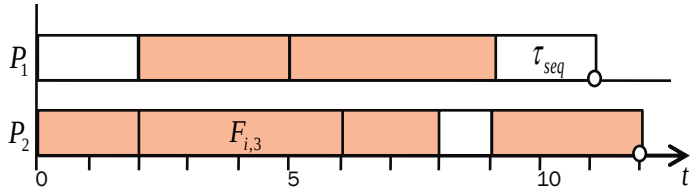
priority

$\tau_1 > \tau_2$

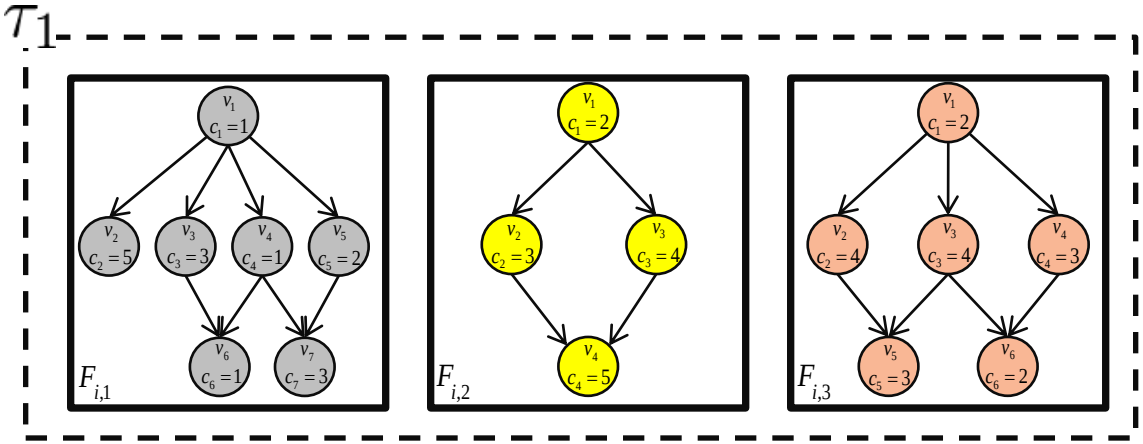
- A potential schedule when the number of cores is 2 :







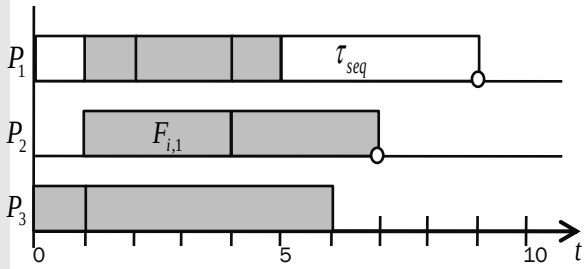
Scheduling Issue

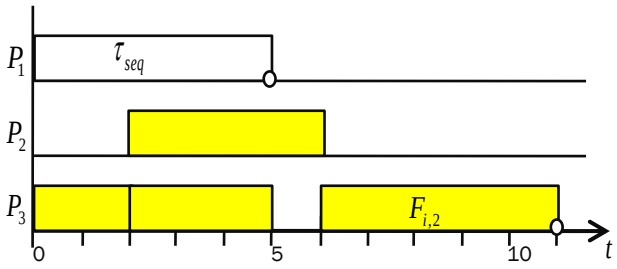


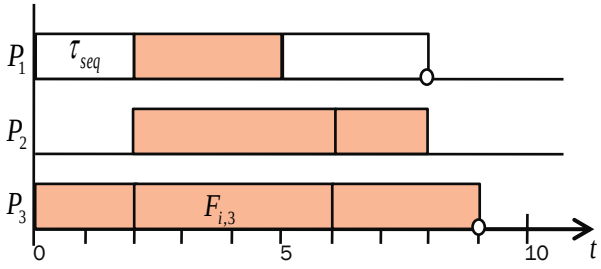
priority

$\tau_1 > \tau_2$

- A potential schedule when the number of cores is 3 :



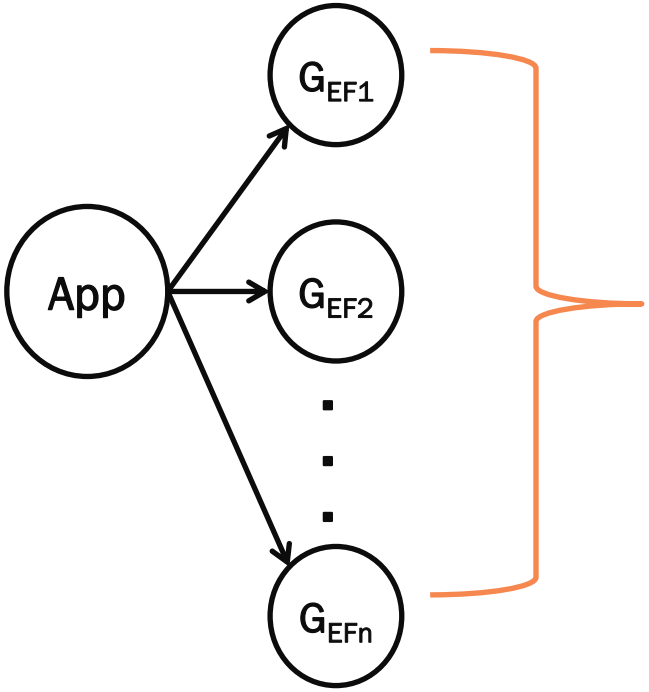




Scheduling Issue

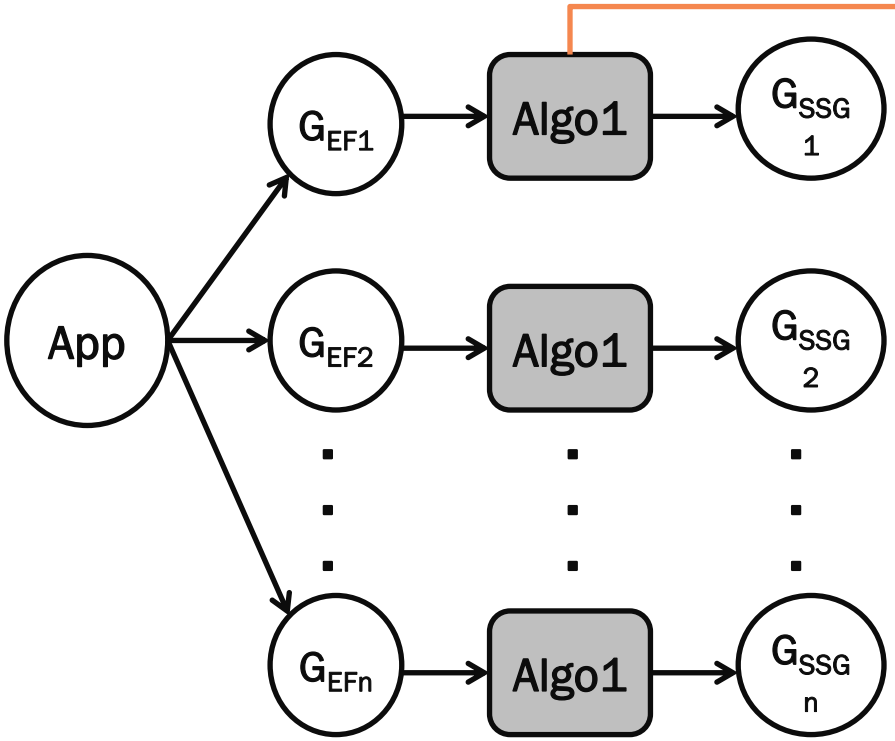
- In general, no execution flow dominates the others in terms of interference
  - Different execution flows may contribute to the WCRT of different lower priority tasks
  - Different settings (cores, priority, etc.) affect the outcome as well
  - Contrary to sequential tasks
- Considering all the flow combinations is intractable!
- Current analysis are still safe if
  1. Different execution flows are fully merged together
  2. Or, schedulability analysis does not rely on the DAG structure

Overview of the Solution



Exhaustive set of feasible executions flows (Multi-DAG) provided by adapted WCET Tools

Overview of the Solution

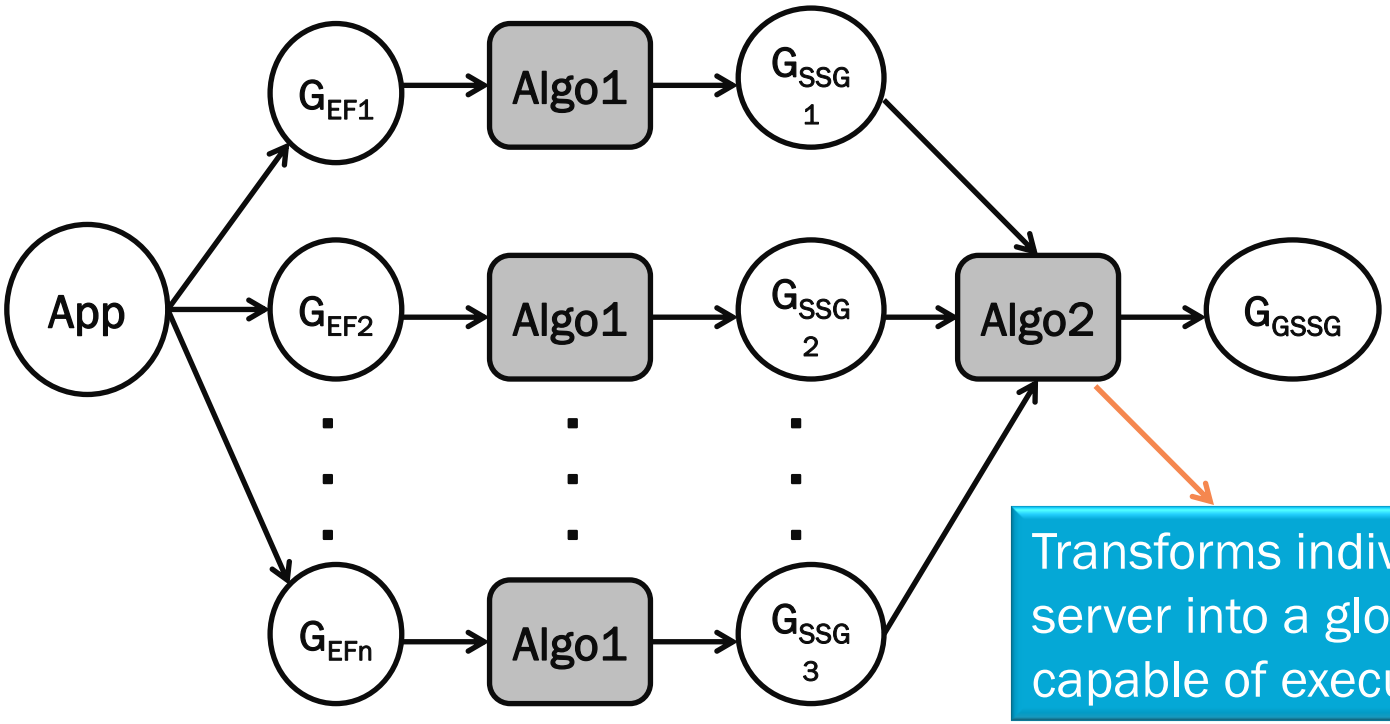


Converts each EF into a DAG of servers with well-defined behaviour

+

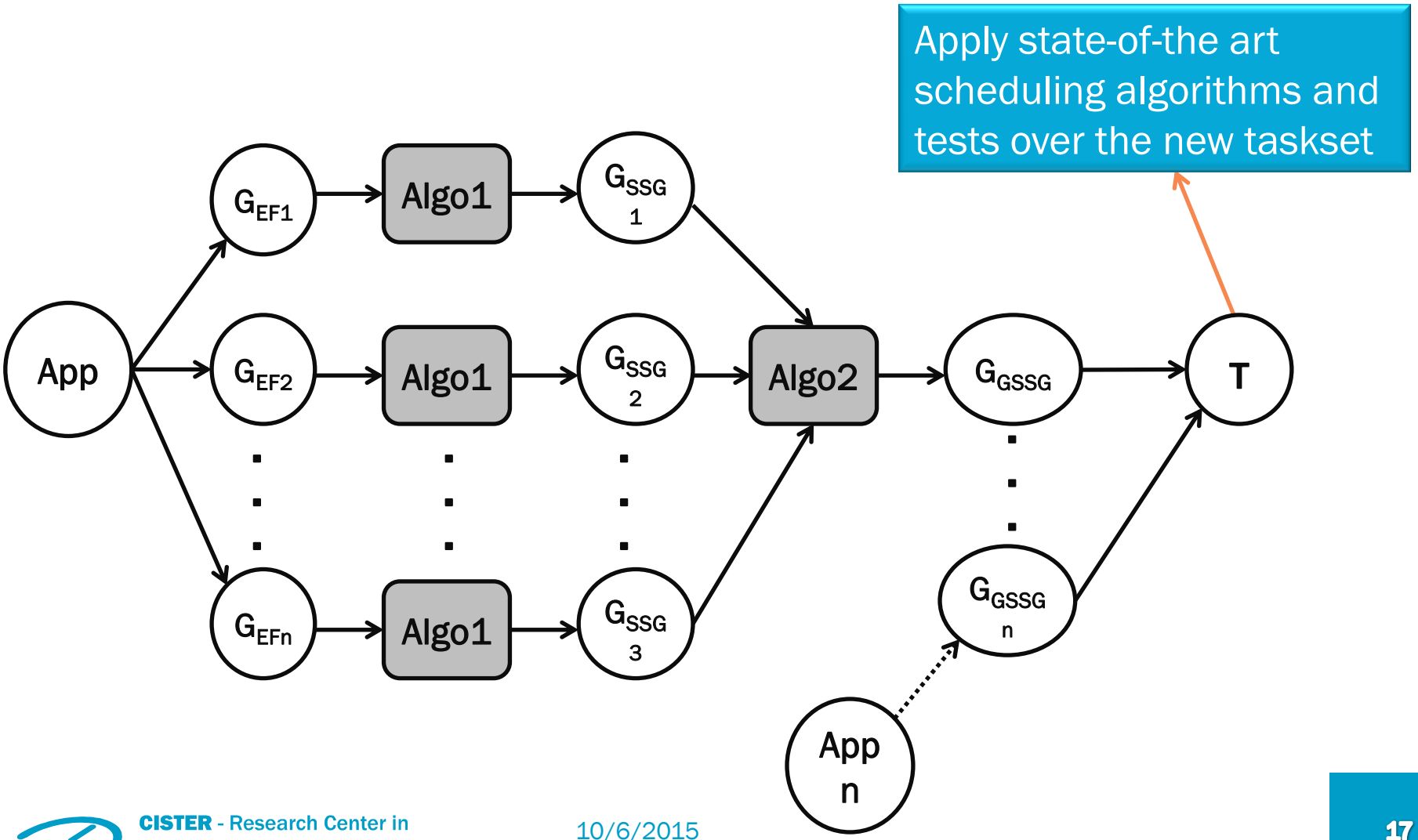
Rule to assign sub-tasks to servers

Overview of the Solution



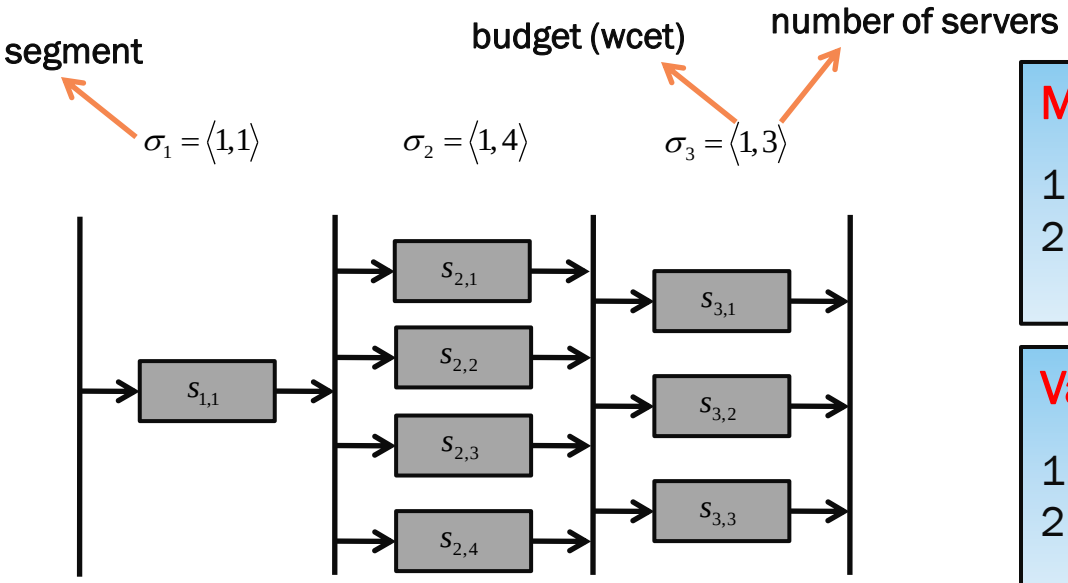
Transforms individual DAG of server into a global DAG of servers capable of executing any EF

Overview of the Solution



Synchronous Server Graph

- A SSG is a synchronous DAG of servers
  - Servers are grouped in segments
  - Servers provide budget to the subtasks
  - Segments release their servers when the previous segment finishes
  - Must satisfy the validity property



Mapping rule

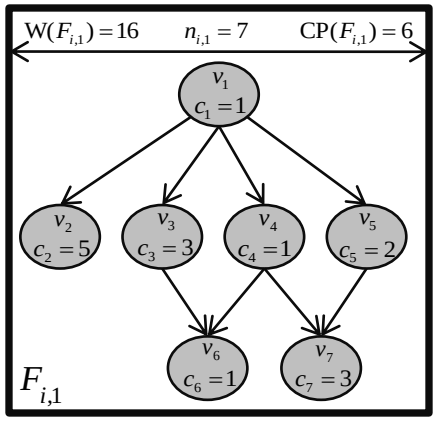
1. Each server accepts only 1 subtask
2. A subtask cannot execute on more than 1 server per segment

Validity

1. All subtasks dependencies are met
2. There is always enough budget to complete the execution flow

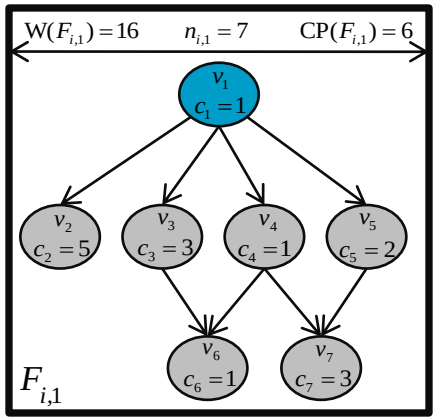
Per-flow SSG

- A valid SSG is derived for each execution flow
  1. Get the set of ready subtasks
  2. Find the minimum execution requeriment
  3. Create a segment with budget equal to 2) and with as many servers as subtasks in 1)
  4. Subtract the budget from the subtasks (remove if complete)
  5. Go back to 1) if the execution flow is not empty



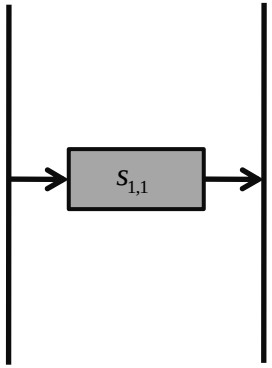
Per-flow SSG

- A valid SSG is derived for each execution flow
  1. Get the set of ready subtasks
  2. Find the minimum execution requeriment
  3. Create a segment with budget equal to 2) and with as many servers as subtasks in 1)
  4. Subtract the budget from the subtasks (remove if complete)
  5. Go back to 1) if the execution flow is not empty



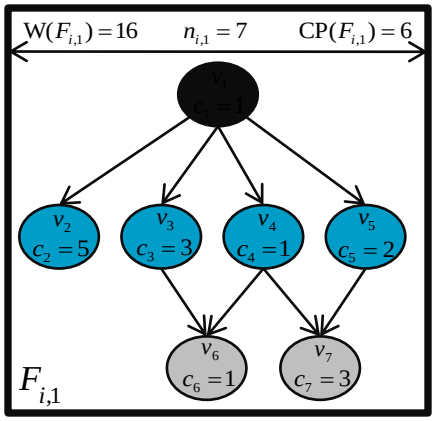
$\sigma_1 = \langle 1,1 \rangle$

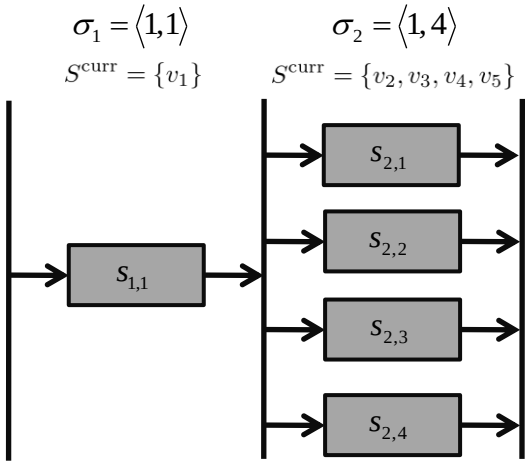
$S^{\text{curr}} = \{v_1\}$



Per-flow SSG

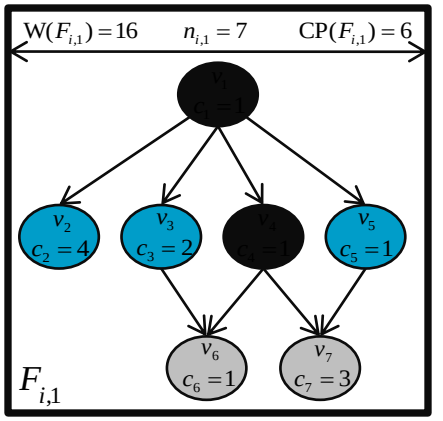
- A valid SSG is derived for each execution flow
  1. Get the set of ready subtasks
  2. Find the minimum execution requeriment
  3. Create a segment with budget equal to 2) and with as many servers as subtasks in 1)
  4. Subtract the budget from the subtasks (remove if complete)
  5. Go back to 1) if the execution flow is not empty

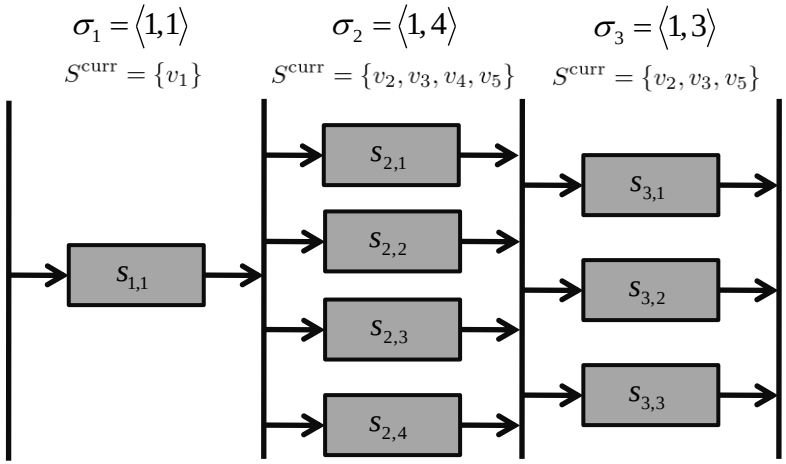




Per-flow SSG

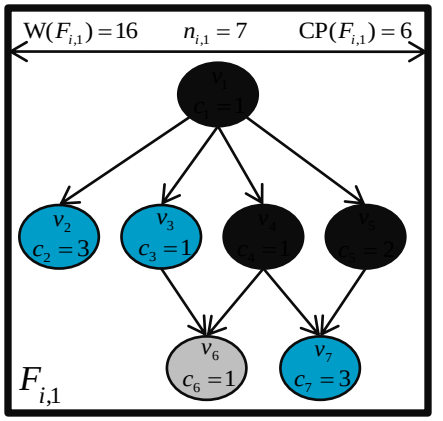
- A valid SSG is derived for each execution flow
  1. Get the set of ready subtasks
  2. Find the minimum execution requeriment
  3. Create a segment with budget equal to 2) and with as many servers as subtasks in 1)
  4. Subtract the budget from the subtasks (remove if complete)
  5. Go back to 1) if the execution flow is not empty

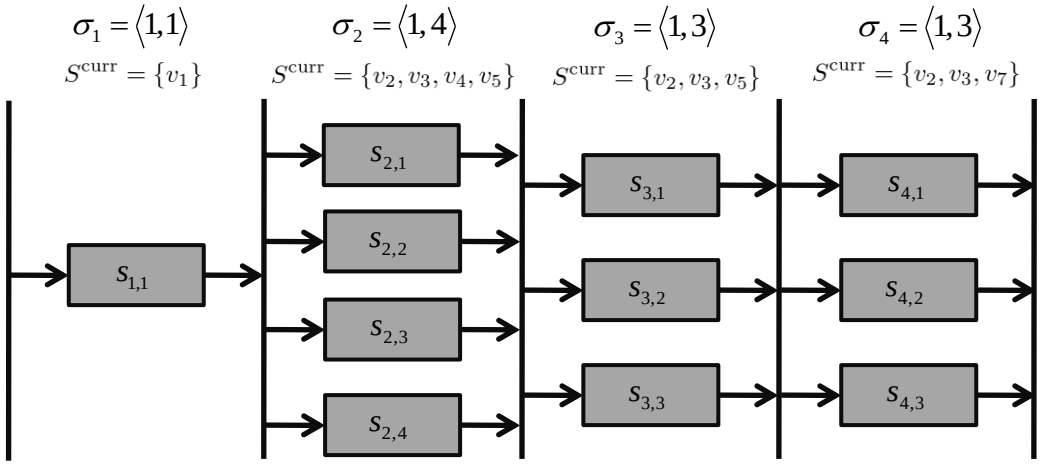




Per-flow SSG

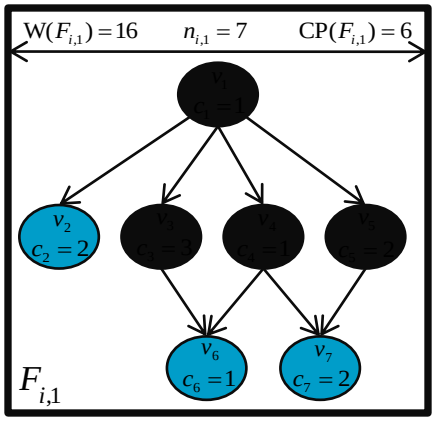
- A valid SSG is derived for each execution flow
  1. Get the set of ready subtasks
  2. Find the minimum execution requeriment
  3. Create a segment with budget equal to 2) and with as many servers as subtasks in 1)
  4. Subtract the budget from the subtasks (remove if complete)
  5. Go back to 1) if the execution flow is not empty

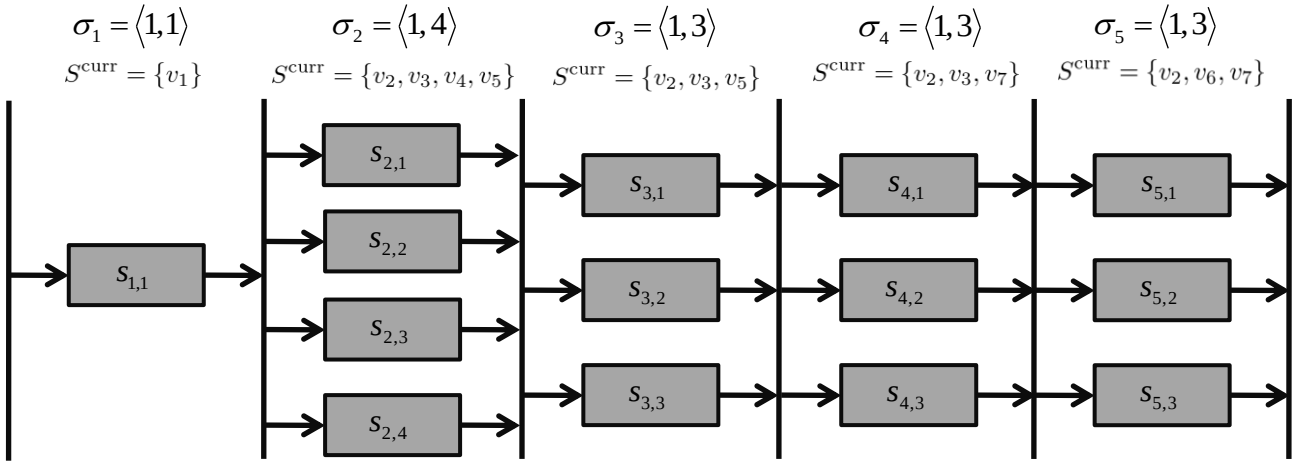




Per-flow SSG

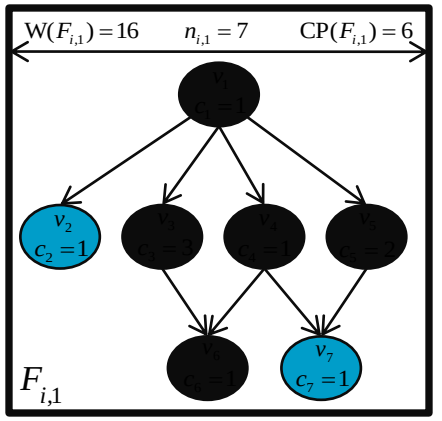
- A valid SSG is derived for each execution flow
  1. Get the set of ready subtasks
  2. Find the minimum execution requeriment
  3. Create a segment with budget equal to 2) and with as many servers as subtasks in 1)
  4. Subtract the budget from the subtasks (remove if complete)
  5. Go back to 1) if the execution flow is not empty

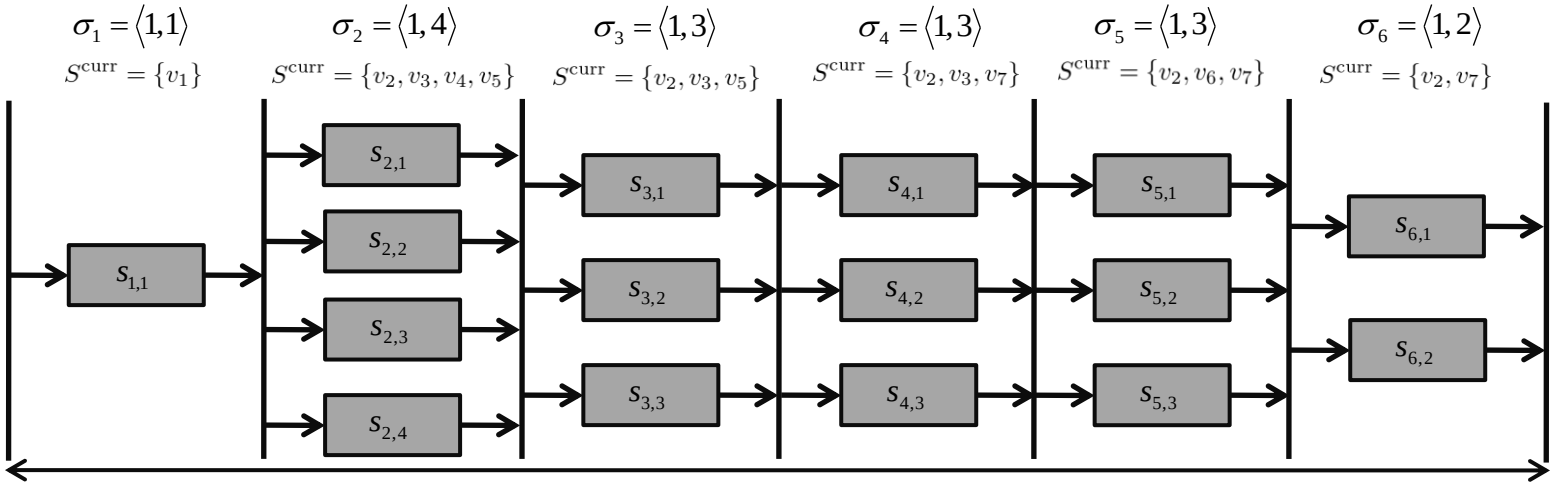




Per-flow SSG

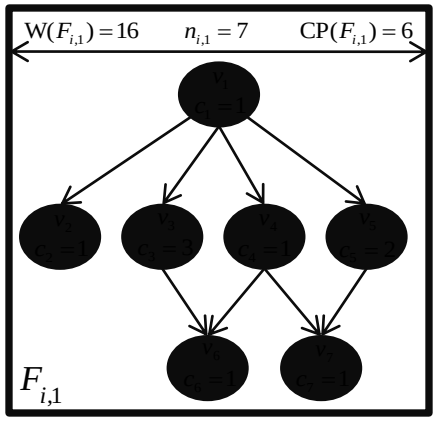
- A valid SSG is derived for each execution flow
  1. Get the set of ready subtasks
  2. Find the minimum execution requeriment
  3. Create a segment with budget equal to 2) and with as many servers as subtasks in 1)
  4. Subtract the budget from the subtasks (remove if complete)
  5. Go back to 1) if the execution flow is not empty

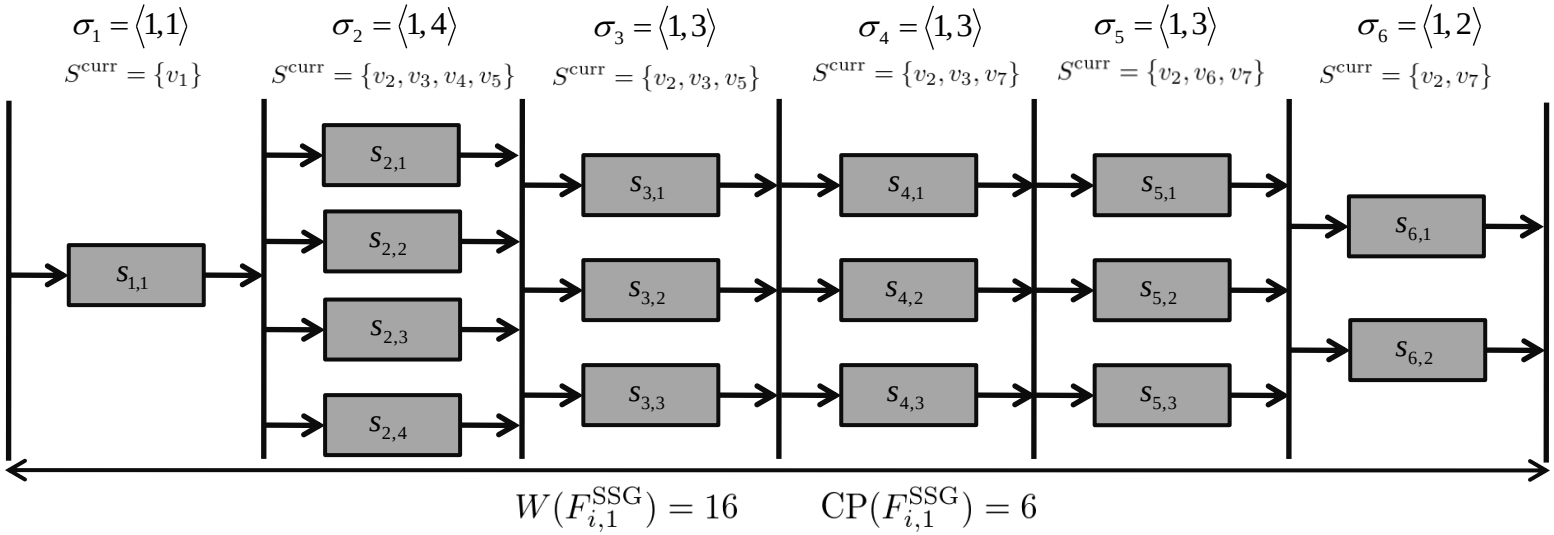




Per-flow SSG

- A valid SSG is derived for each execution flow
  - Optimal workload
  - Optimal critical path length



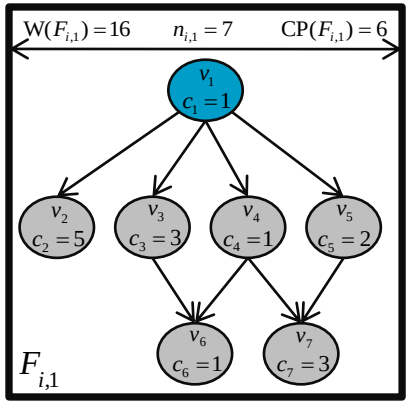


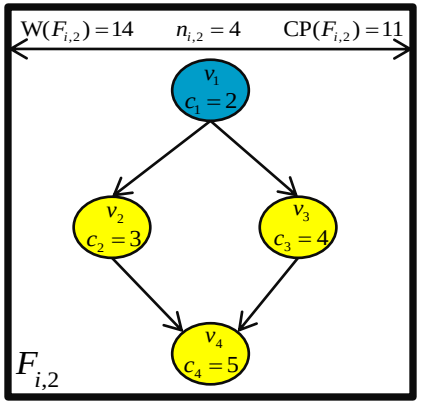
Per-Task GSSG

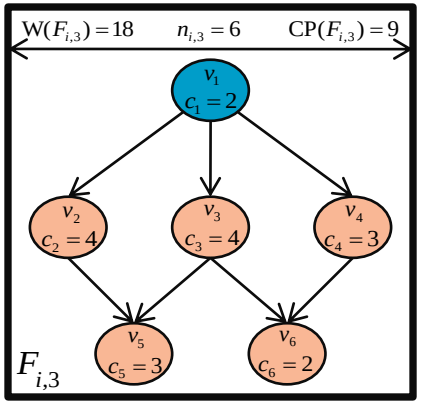
- A valid global SSG is derived for each task based on all its SSGs
  - Any job of the task is guaranteed to complete, irrespective of which execution flow is taken at run-time
  - Inherites task deadline and period
  - Converts the multi-DAG model into a parallel synchronous task model
  - Allows schedulability analysis to be performed over the GSSGs

- Algorithm
  1. Get the ready segments from every SSG
  2. Find the minimum budget
  3. Find the maximum number of servers
  4. Create a segment with budget equal to 2) and number of servers equal to 3)
  5. Subtract the budget from the segments in 1) (remove if complete)
  6. Go back to 1) until all SSGs are empty

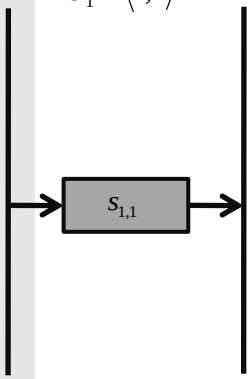
Per-Task GSSG



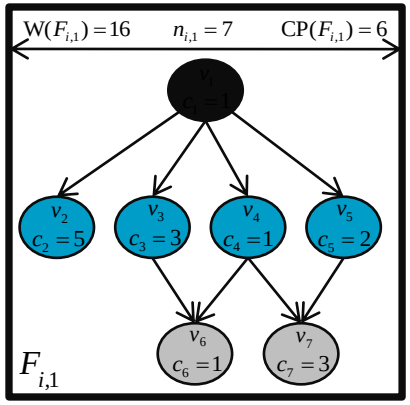


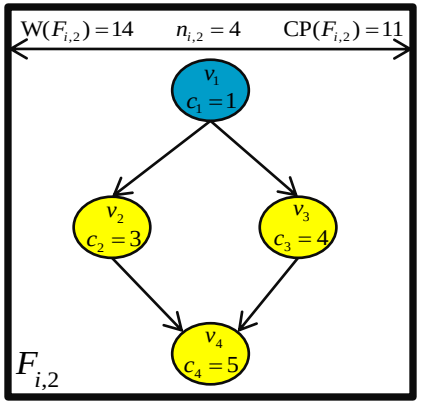


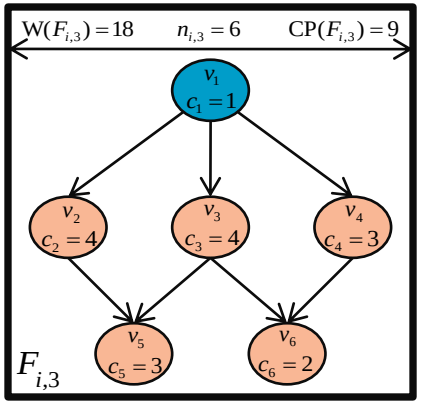
$\sigma_1 = \langle 1,1 \rangle$

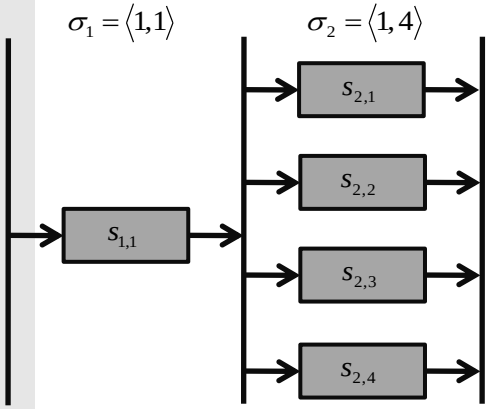


Per-Task GSSG

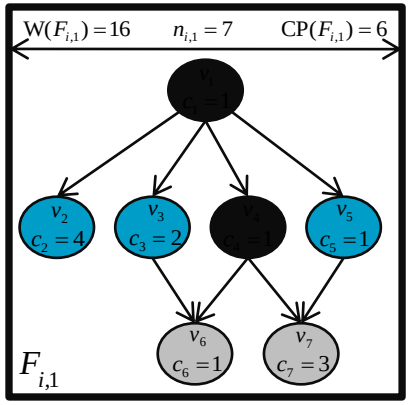


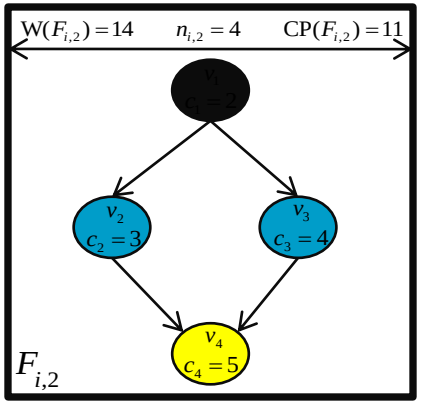


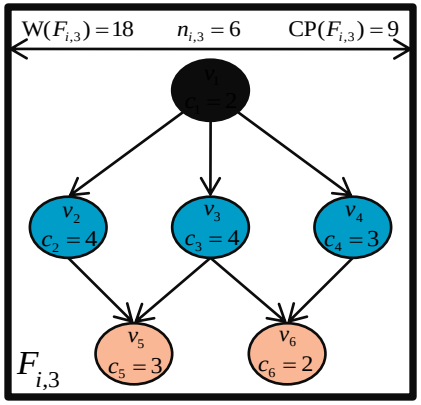


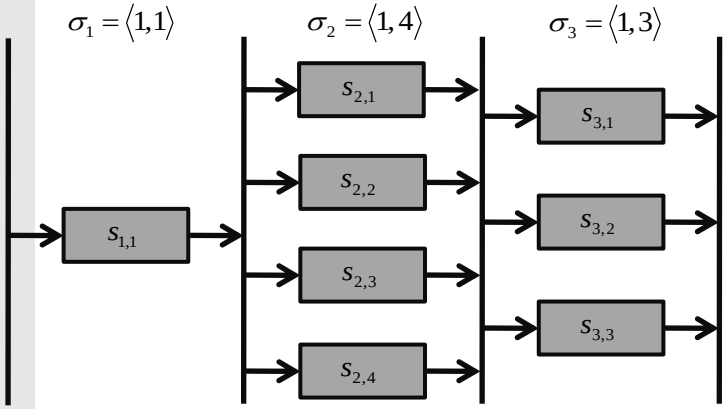


Per-Task GSSG

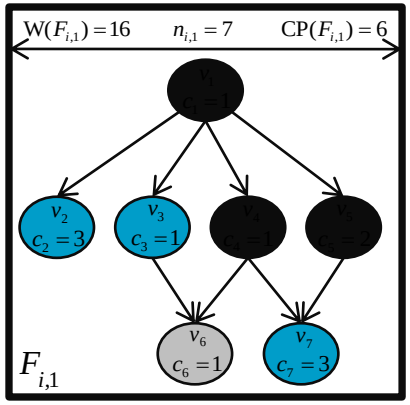


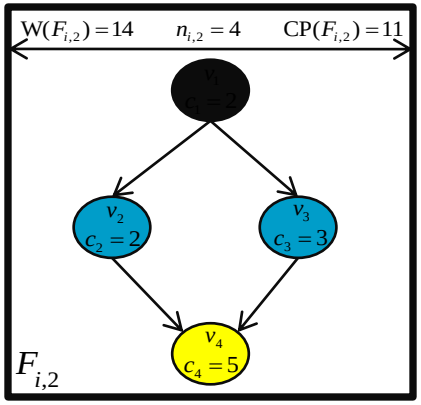


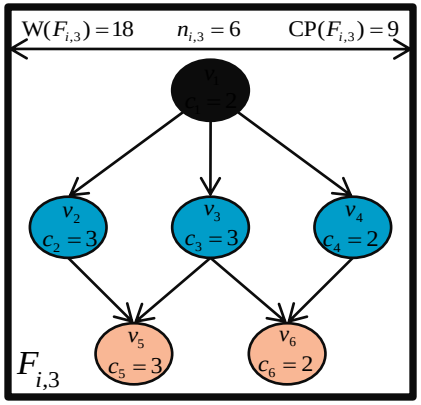


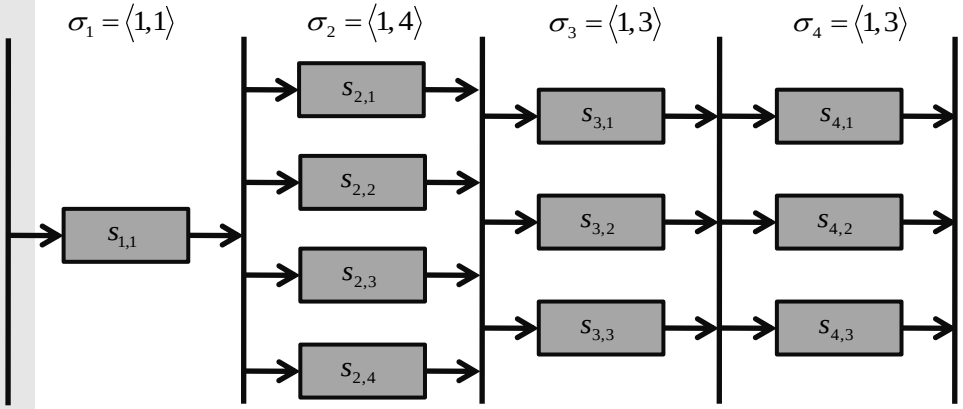


Per-Task GSSG

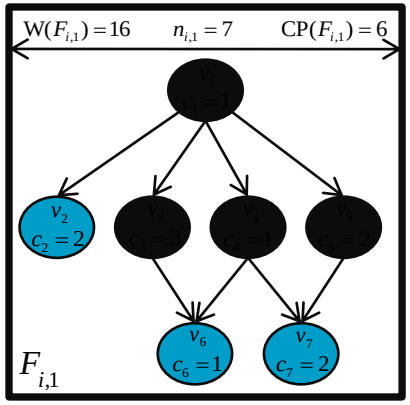


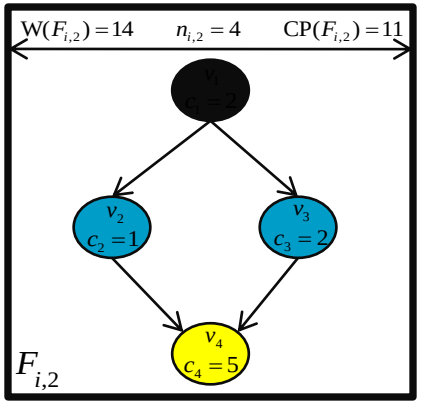


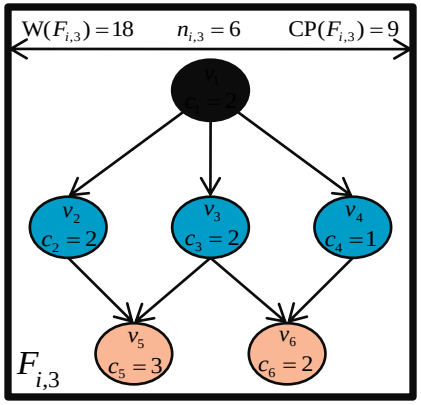


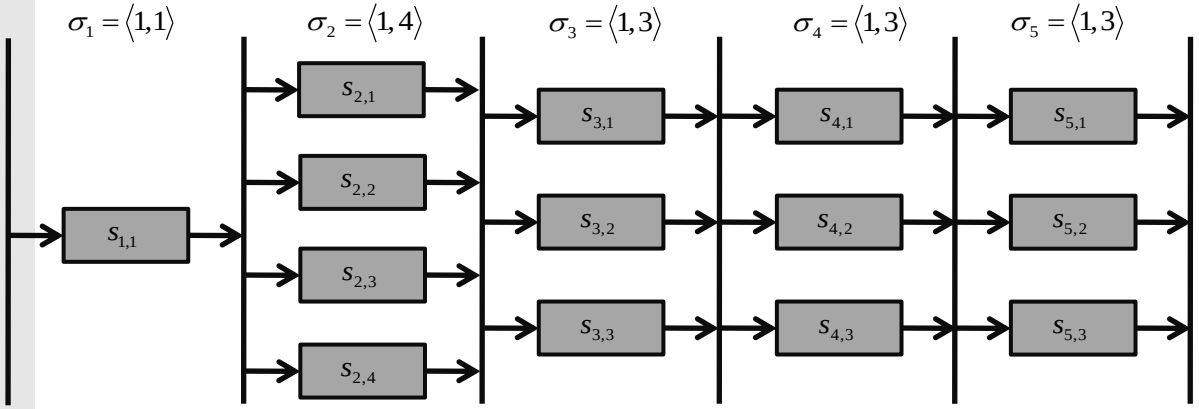


Per-Task GSSG

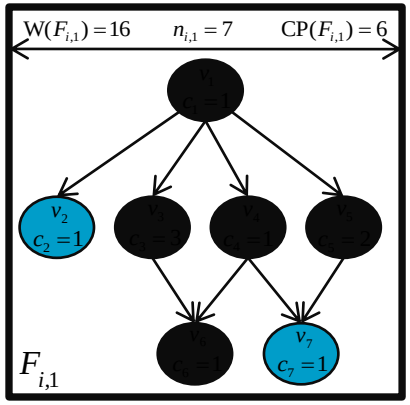


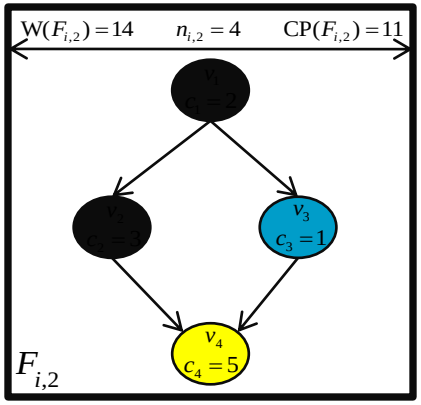


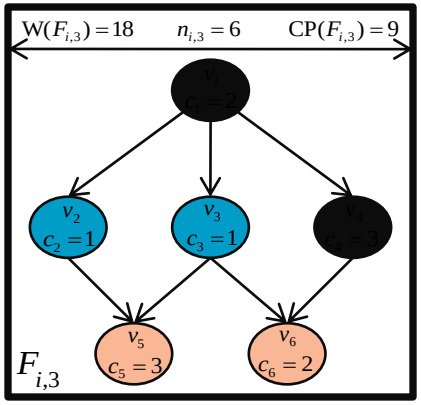


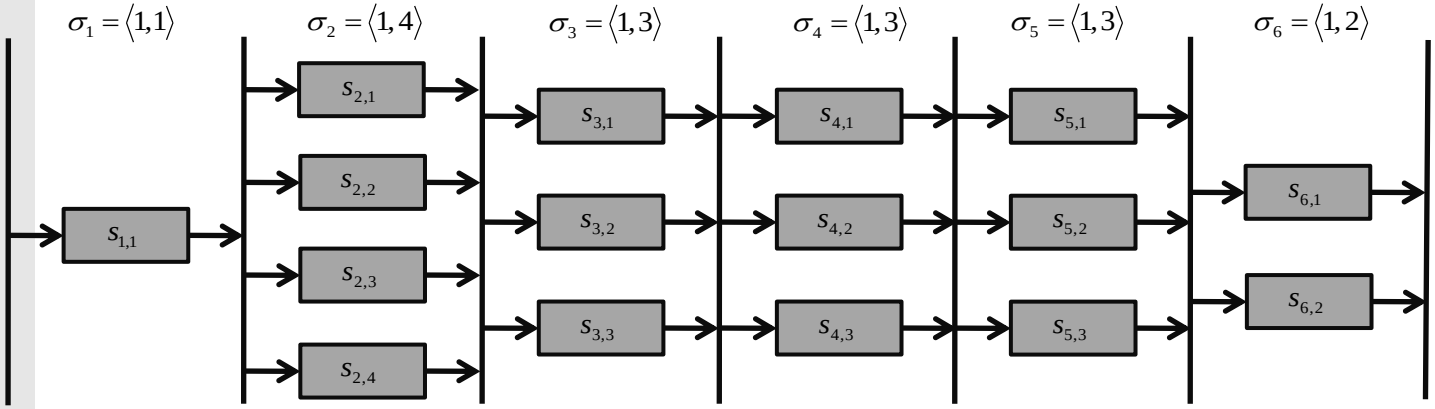


Per-Task GSSG

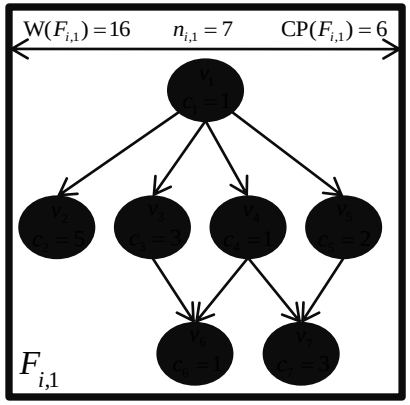


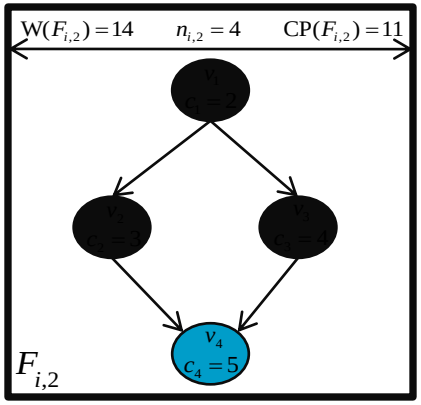


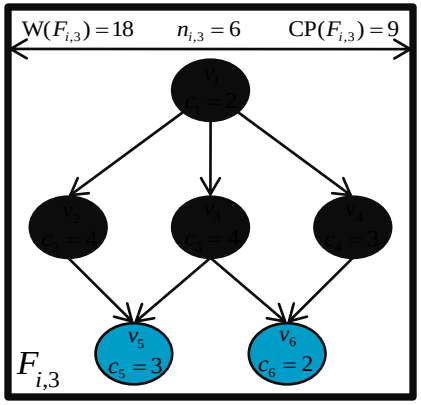


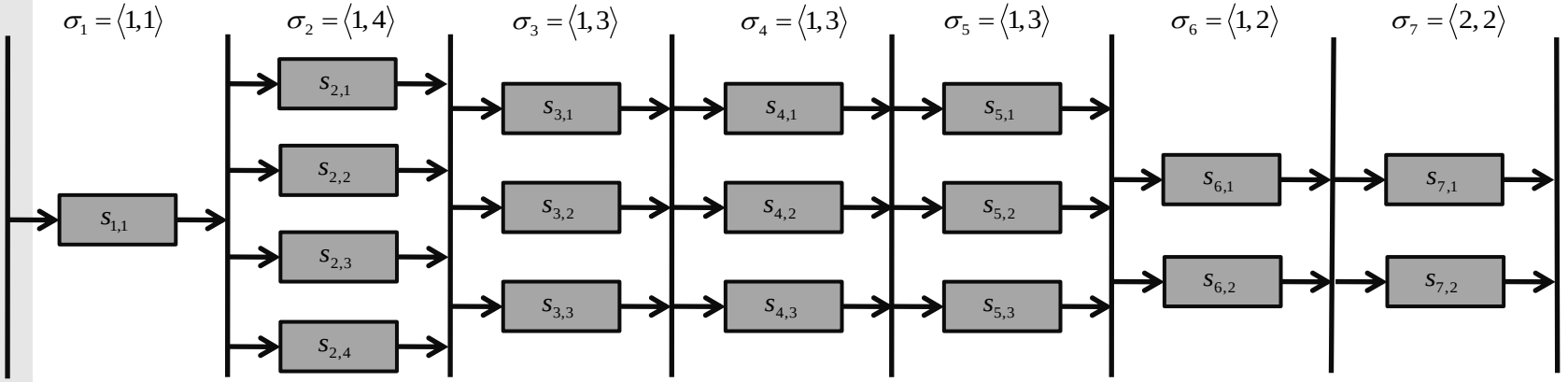


Per-Task GSSG

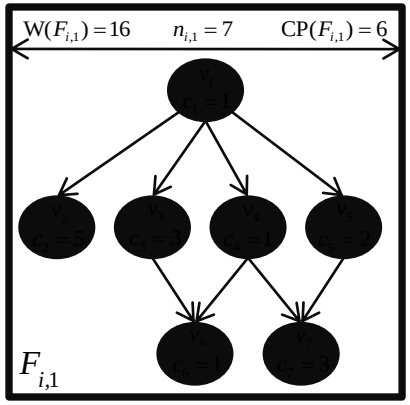


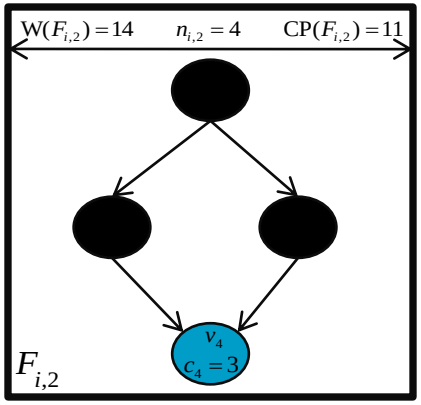


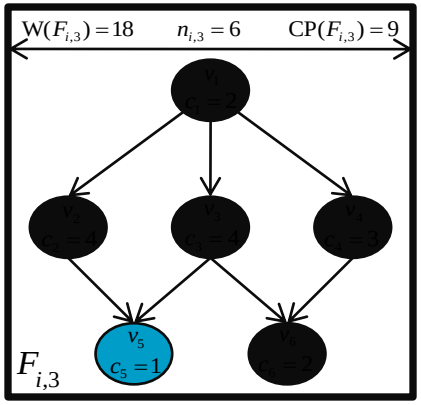


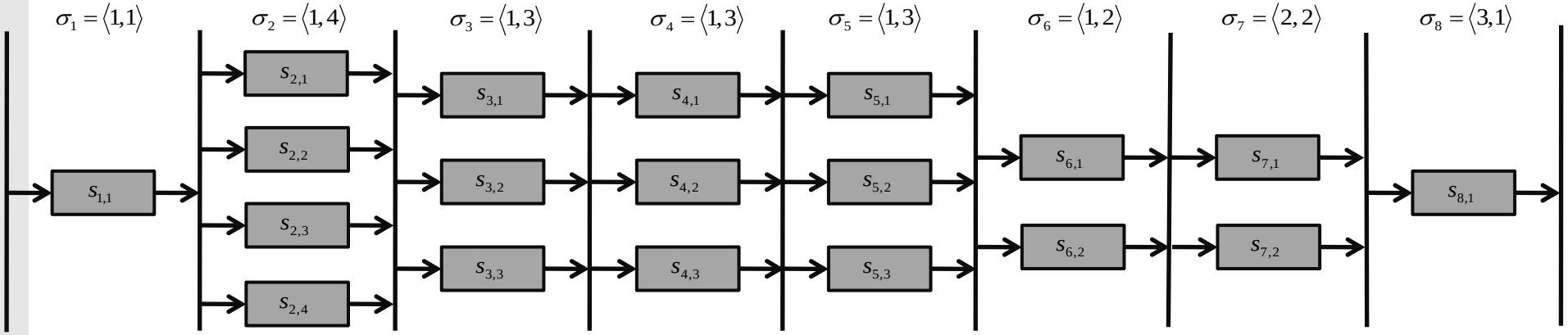


Per-Task GSSG



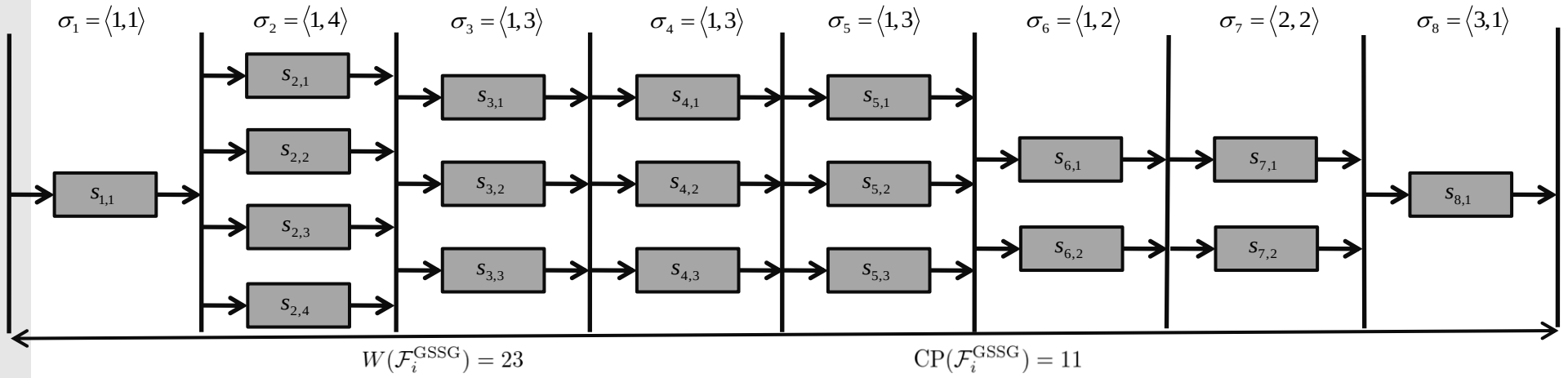






Per-Task GSSG

- A valid global SSG is derived for each task based on all its SSGs
  - Optimal critical path length
  - Inflated workload



Conclusions

- A first attempt to explicitly model and address the schedulability of parallel tasks with conditional execution

- The DAG of servers requires support from the Operating Systems

- Algorithm 2 can be improved to yield tighter workloads
  - A trade-off between workload and critical path length exists

- GSSGs allow schedulability tests to leverage from the DAG structure
  - Comparison to works that neglect the DAG structure is focus of our future work

Thank you for your attention!