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

Improved Response Time Analysis of Sporadic DAG Tasks for Global FP Scheduling

José Fonseca, Geoffrey Nelissen
and Vincent Nélis

Instituto Superior de Engenharia do Porto

INESCTEC

10/4/2017

Parallel task models

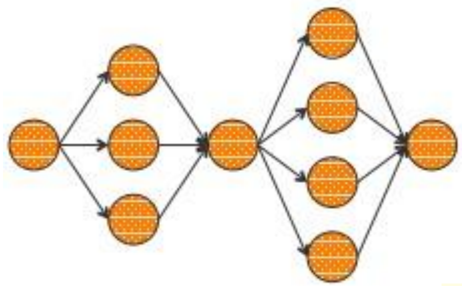
Exploit powerful multicore architectures

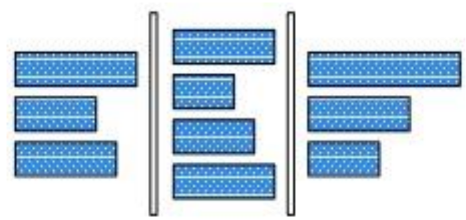
- Through task parallelism

Target modern applications

- Real-time and high-performance requirements

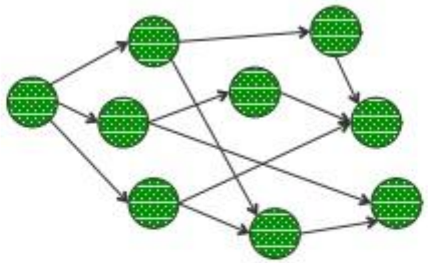
1. Fork-join

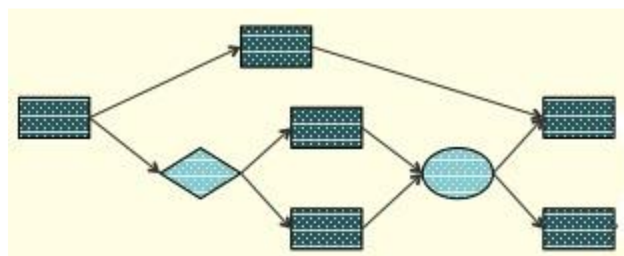




2. Synchronous parallel

3. DAG



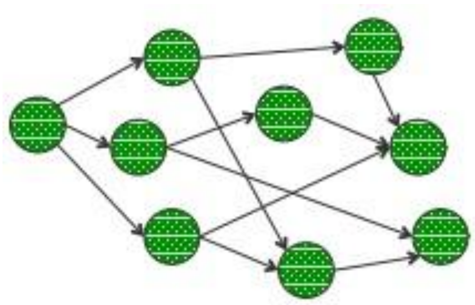


4. Conditional DAG

Most analysis **overlook** such rich **internal structures**

System Model

- Set of DAG tasks
- Sporadic arrivals
- Constrained deadlines
- Task-level fixed priorities
- Global scheduling
- Platform composed of *m* identifical cores



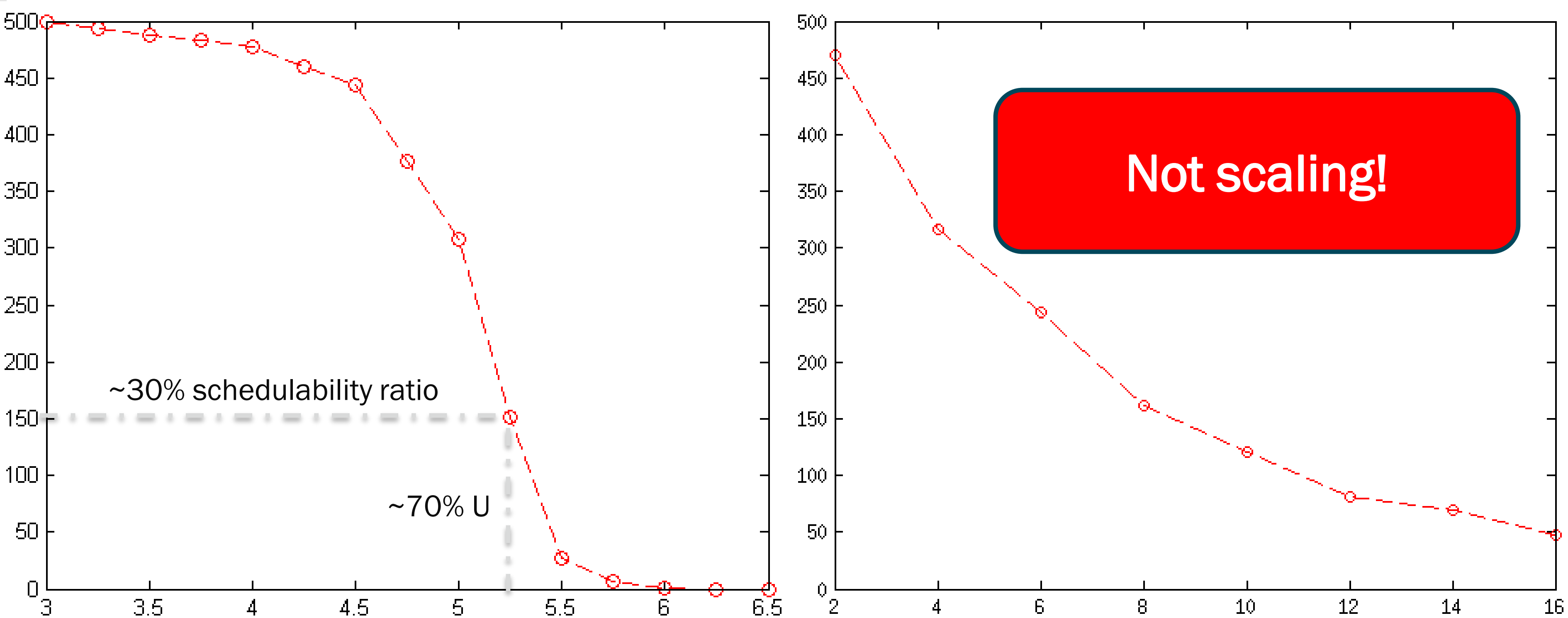
Overall Problem

Schedulability analysis for DAG tasks on a multiprocessor system under G-FP scheduling

State-of-the-art Analysis

[Melani'15] A. Melanie, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela and G. C. Buttazzo, "Response Time Analysis of Conditional DAG Tasksin Multiprocessor Systems", ECRTS'15

Performance in terms of schedulable task sets



Utilization, m = 8

Cores, U = 70%

Understanding State-of-the-art Analysis [Melani'15]

[Melani'15] - RTA

Response time computation of a DAG task τ_k

$$R_k = L_k + \frac{1}{m}\sum_{\forall i \le k} I_{i,k}$$

Length of the interfered path

Interfering workload

Work-conserving property

Interference is spread over all *m* cores

Two types of interference

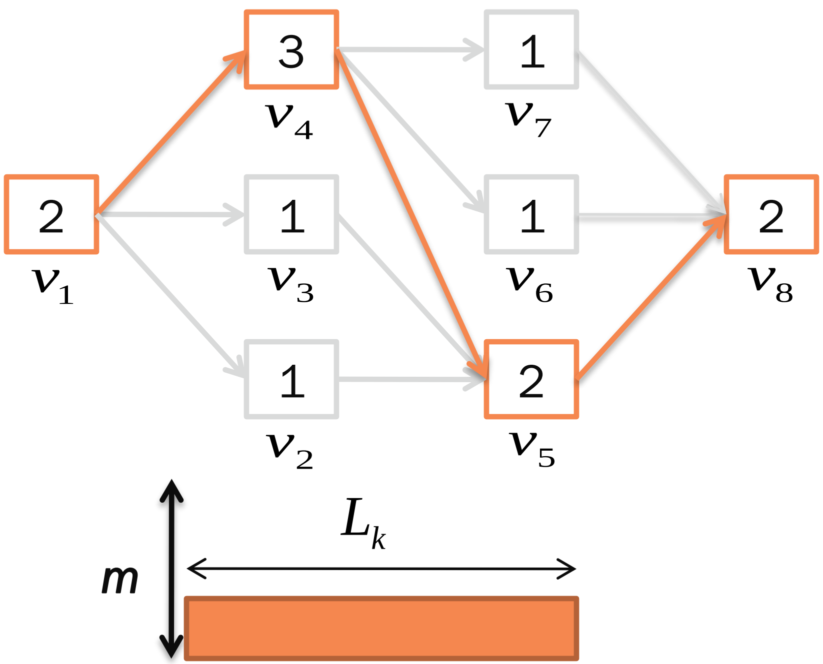
- Self interference
- Inter-task interference

[Melani'15] - Self Interference

It is the delay exerted on the RT of interfered path by the own DAG

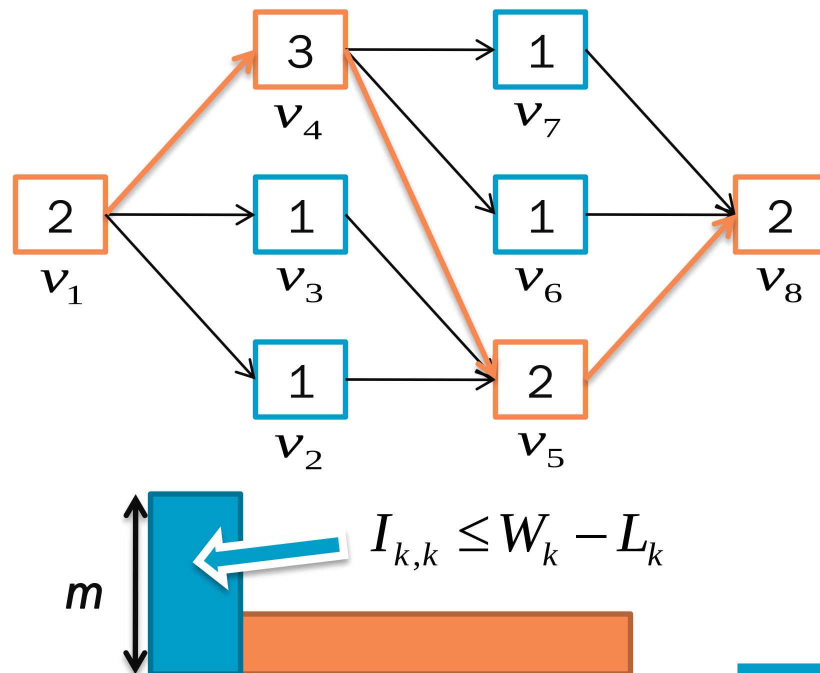
Who is interfered?

- Any critical path



Who interferes?

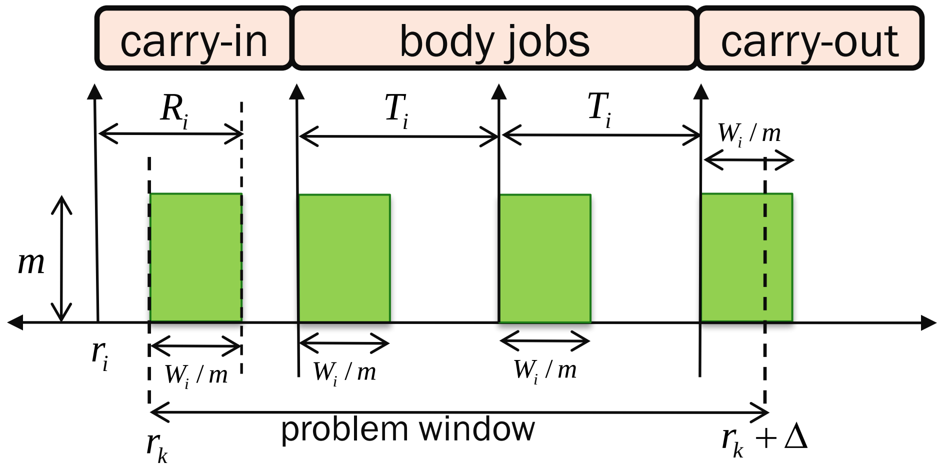
- Every node that does not belong to the selected critical path

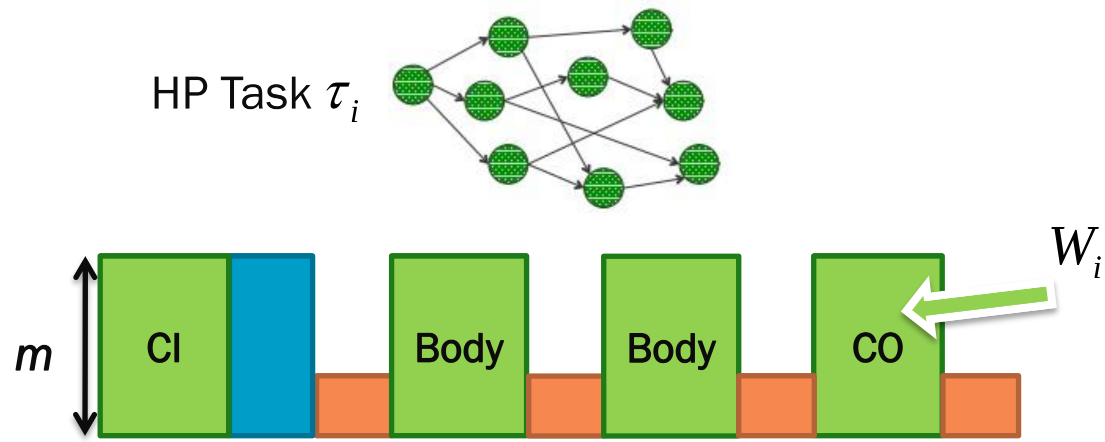


[Melani'15]: Inter-Task Interference

Accounts for the maximum **interfering workload** generated by the jobs of the **HP tasks**

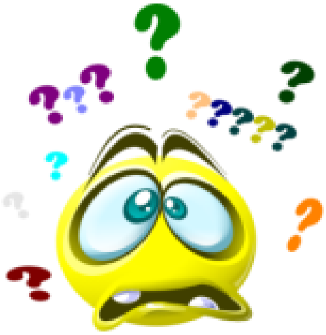
- Inter-task interference depends on the length of the interval
- Based on the concept of problem window





Lost all information about the DAG's internal structure!

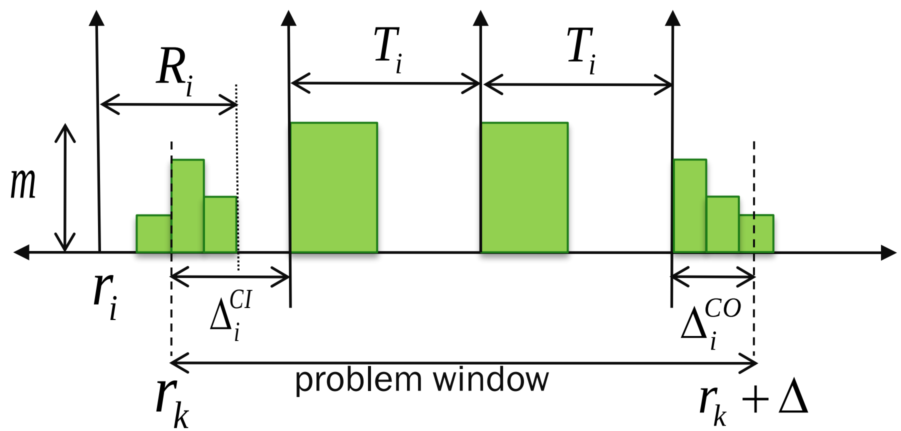
What Can We Do?



Problem Definition

Proposed worst-case scenario

➢ Explores the internal structure of each DAG to derive more accurate carry-in and carry-out contributions



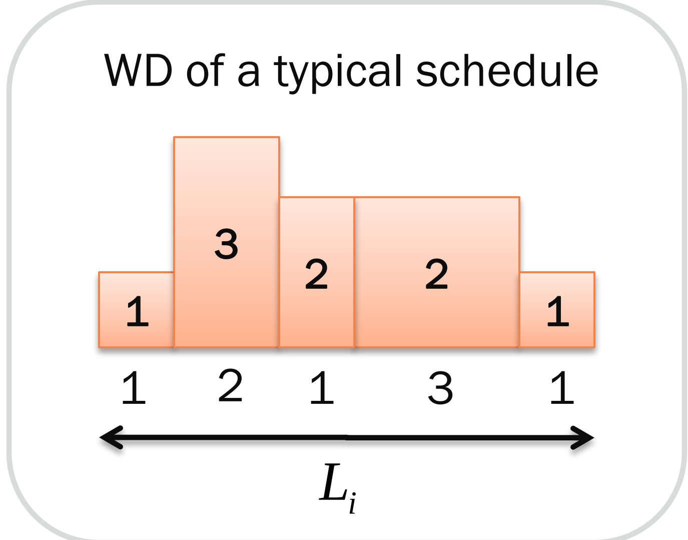
Challenges

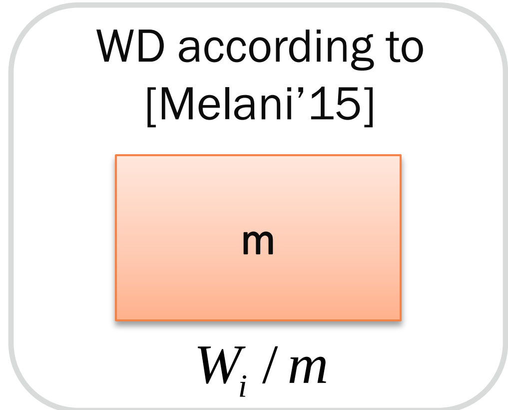
- Upper-bound the carry-in workload
- Upper-bound the carry-out workload
- Position the window such that interference is maximized

A New Notion

Workload Distribution (WD)

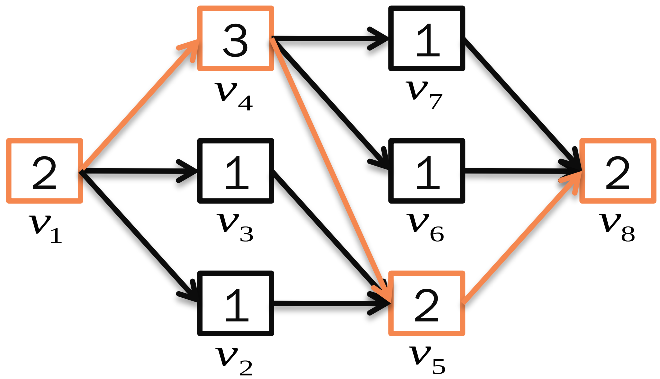
- A workload distribution describes a schedule S of a DAG task as a sequence of blocks (w,h)
- The height denotes the number of executing nodes
- The width determines the duration of such execution batch
- Total workload in function of a certain length is given by the areas
- It is not required for S to be valid





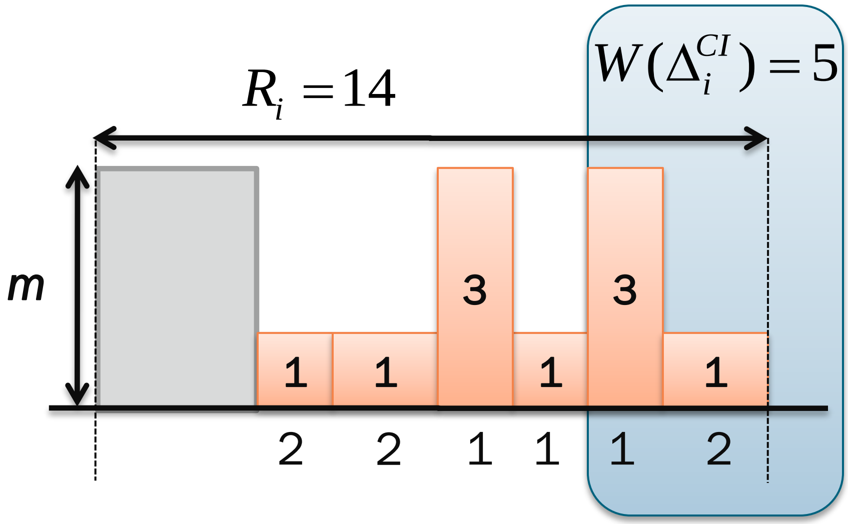
Carry-in Workload

How to model the carry-in job such that the interfering workload is maximized?



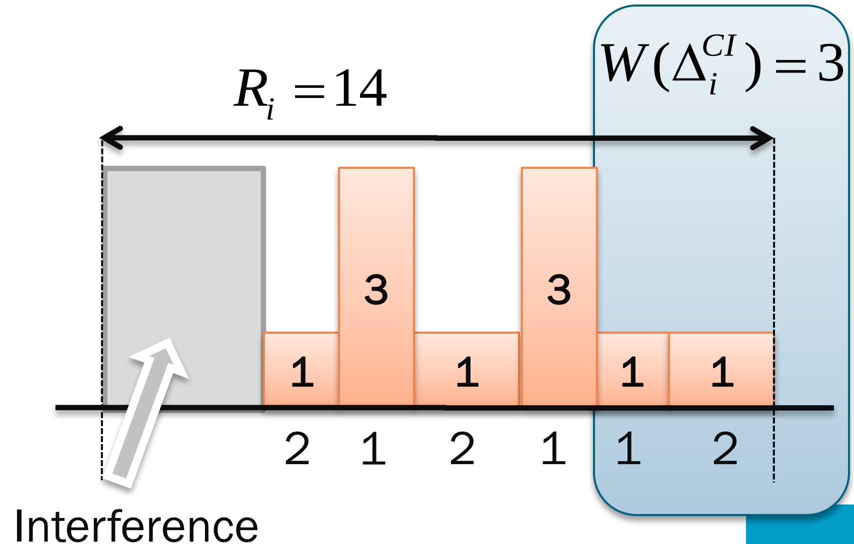
Intuitive approach

- Nodes execute as late as possible



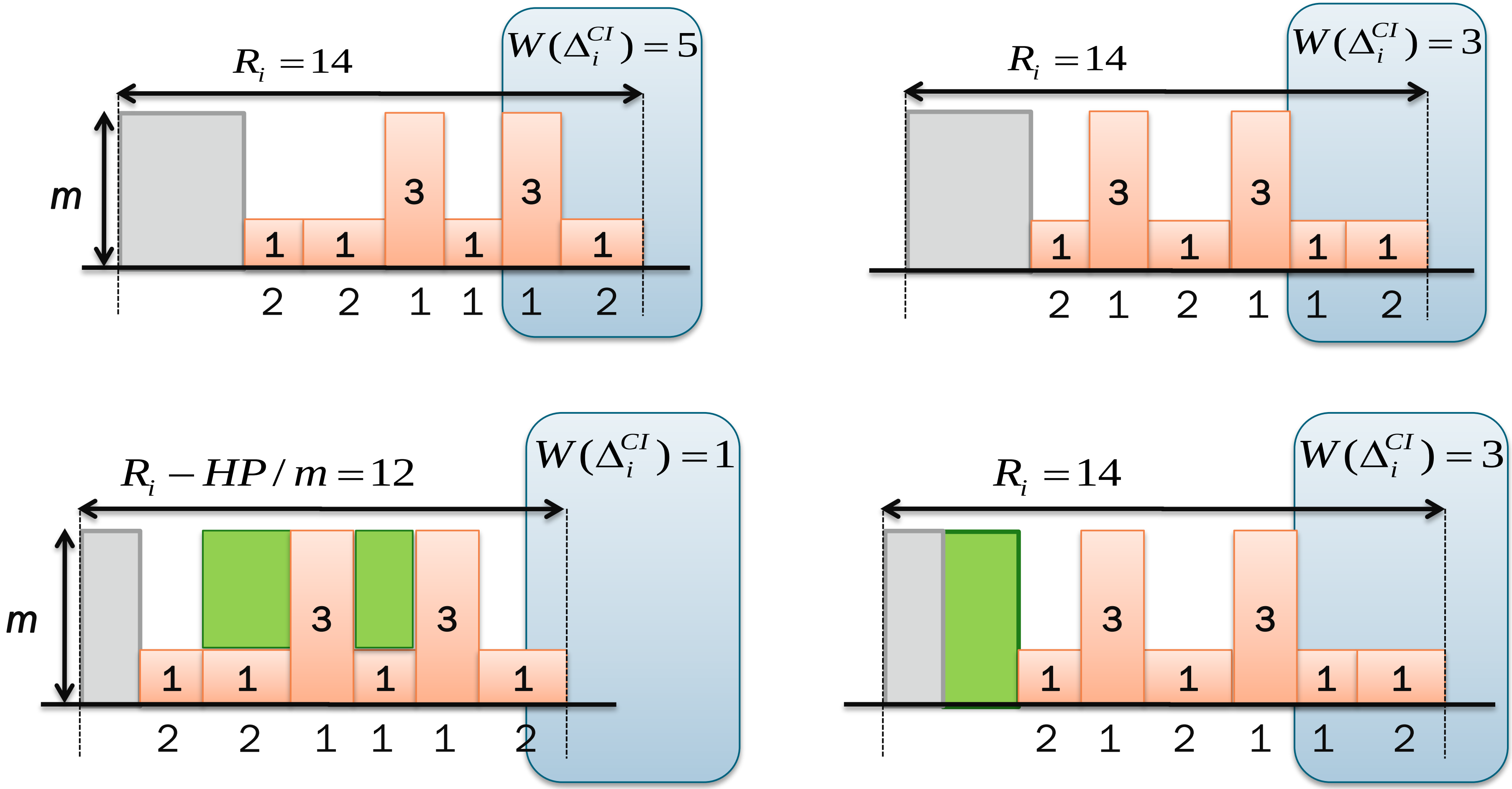
Our approach

- Nodes execute as soon as possible



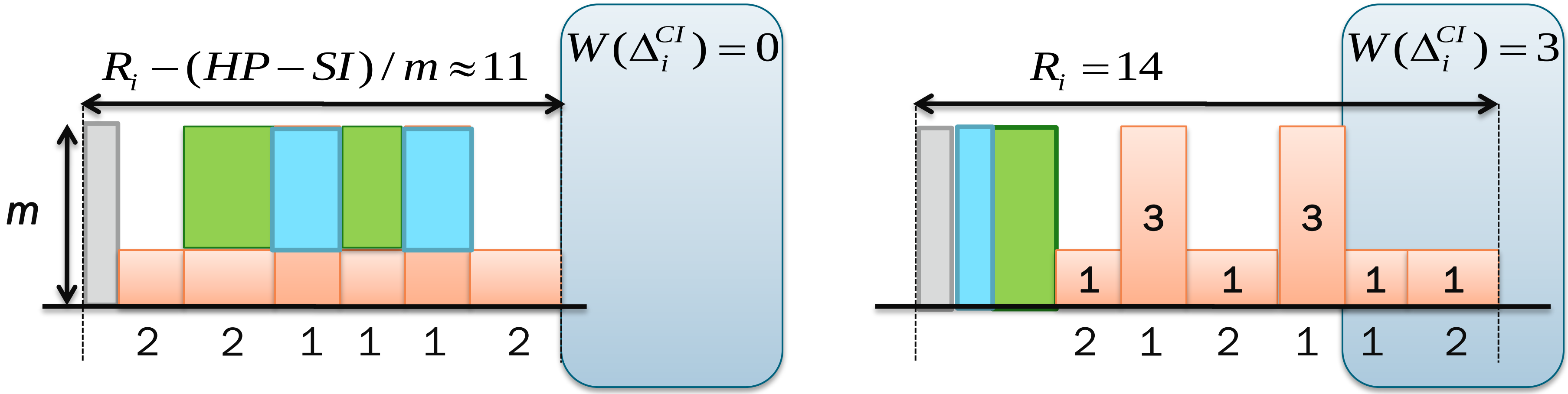
Carry-in Workload

What happens to the actual WCRT when we check the inter-task interference?



Carry-in Workload

And now also the self interference…



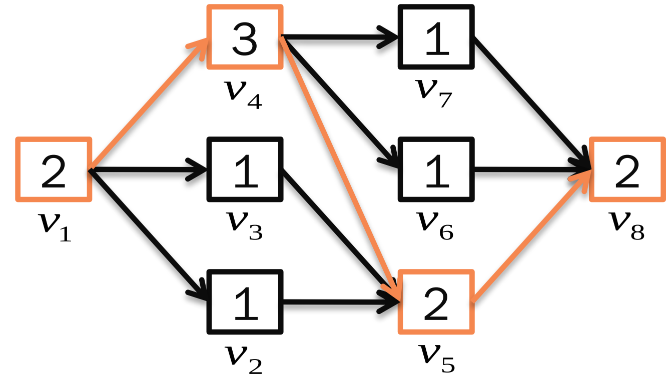
The makespan WD upper-bounds the interfering workload generated by the carry-in job when

- The WD is aligned with the WCRT
- The WCRT is computed according to the pessimistic method described
- Any other WD generates less workload due to the discrepancy between its actual RT and the WCRT

Carry-out Workload

How to model the carry-out job such that the interfering workload is maximized?

- Execute as much workload as possible, as soon as possible
- Maximum cumulative parallelism



Can we construct such schedule for any value of the CO length?

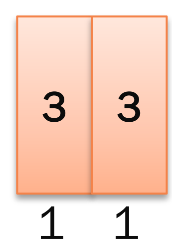
if $\Delta_i^{CO} = 1$

node	v_1	v_2	v_3	v_4	v_6	v_7
exec time	0	1	1	0	1	1

$W(\Delta_i^{CO}) = 4$

if $\Delta_i^{CO} = 2$

node	v_1	v_2	v_3	v_4	v_5	v_6	v_7
exec time	0	1	1	1	1	1	1



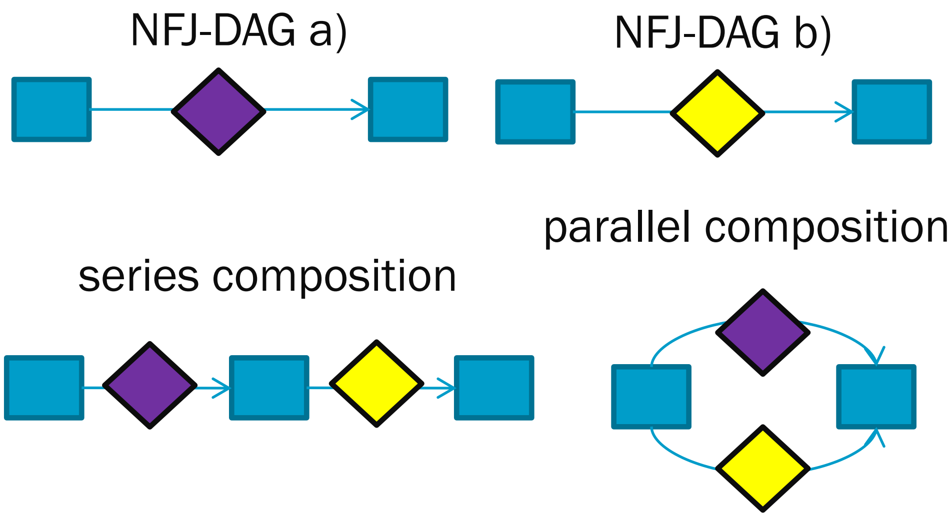
$W(\Delta_i^{CO}) = 6$

Carry-out Workload

We solve the problem by transforming the DAG into a nested fork-join DAG

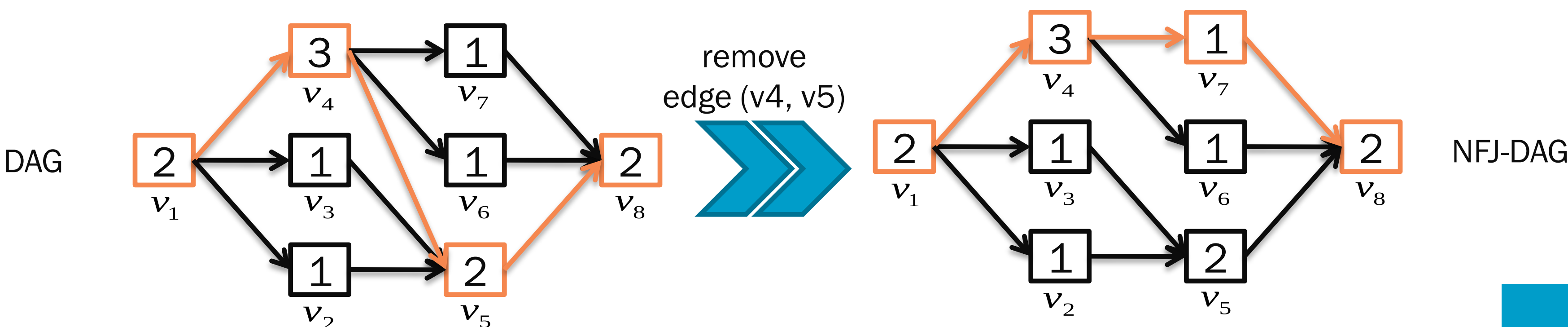
- Well-structured parallelism
- More general than SP model
- More concurrency

NFJ-DAG construction



Transformation

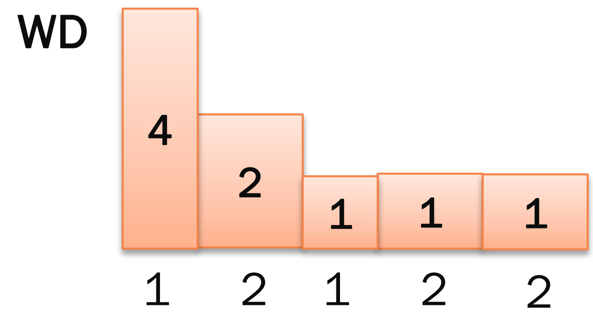
- Identify conflicting edges
- Remove minimum number of such edges to resolve the issue



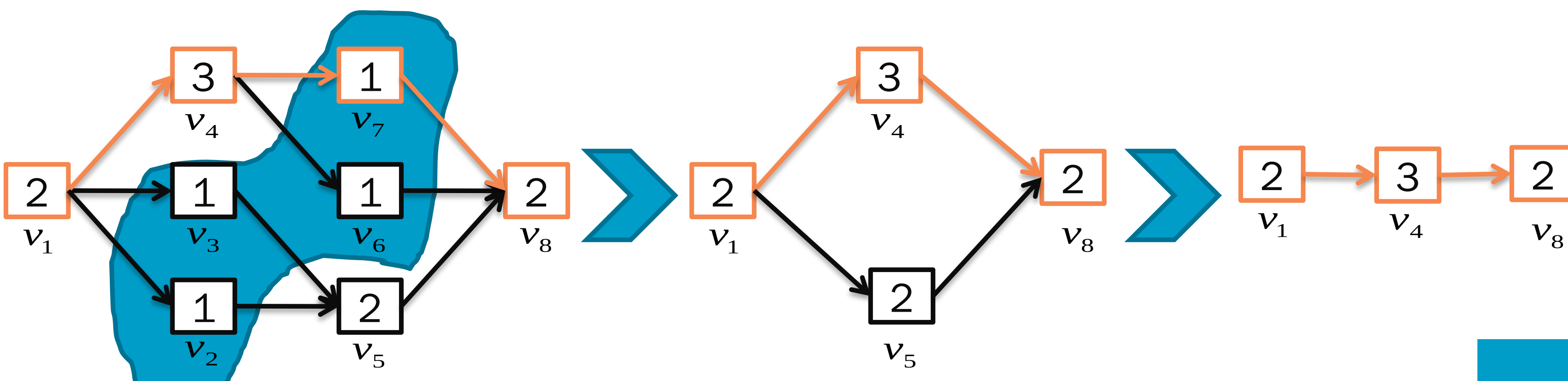
Carry-out Workload

Constructing WD

- Find the set yielding maximum parallelism in the NFJ-DAG (uses a binary tree)
- The height is the number of elements in the set
- The width is the minimum (remaining) WCET among the elements
- Subtract this value from the selected nodes; remove exhausted nodes
- Repeat until NFJ-DAG is empty

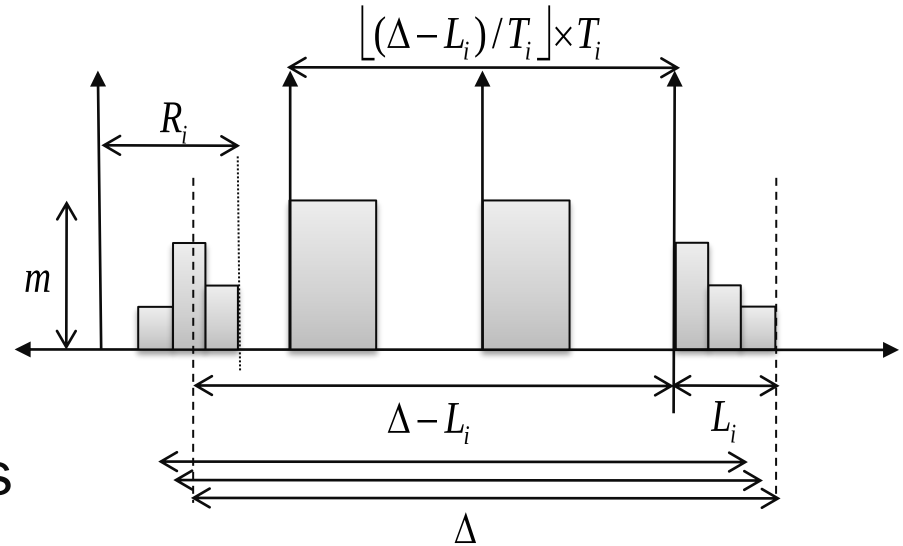


max cumulative becomes max at each step



Response Time Analysis

How to align the problem window in order to upper-bound the interfering workload of both carry-in and carry-out jobs?



The problem can be formulated as

$$\textbf{max} \quad CI_i(x1) + CO_i(x2)$$
$$\textbf{s.t.} \quad x1 + x2 = \Delta_i^C$$

- The solution to this optimization problem is the desired upper-bound
- The values of x1 and x2 correspond to the length of the carry-in and carry-out windows

We proposed an algorithm to solve this sliding window problem with complexity linear to the number of blocks in the WDs

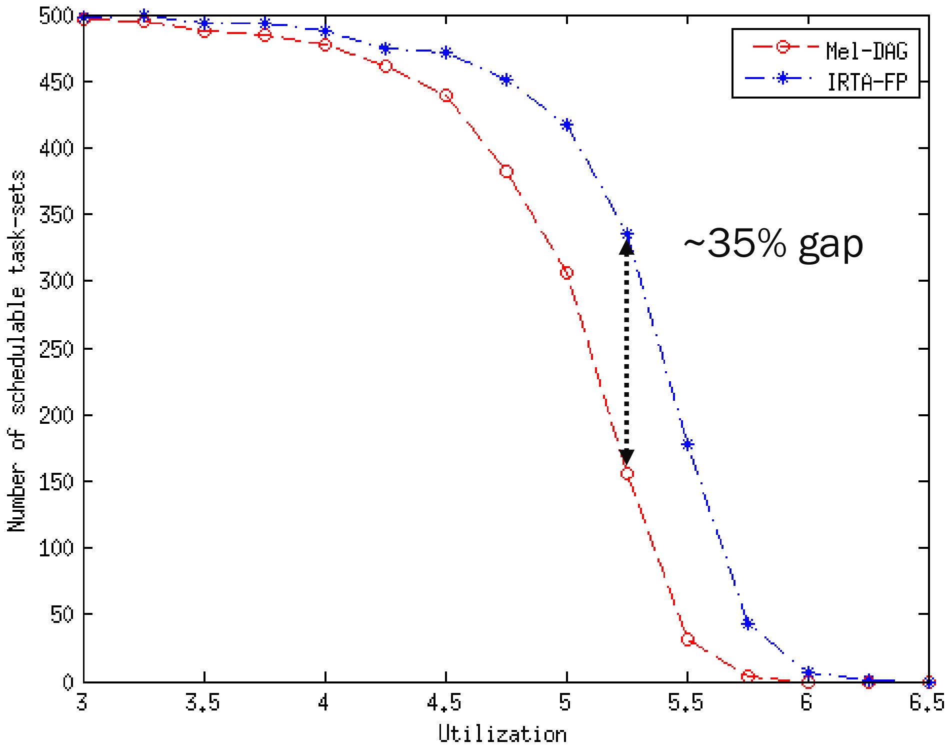
Experimental Results

Comparison with the state-of-the-art G-FP analysis [Melani'15]

We assessed the schedulability of 500 task sets per configuration as a function of:

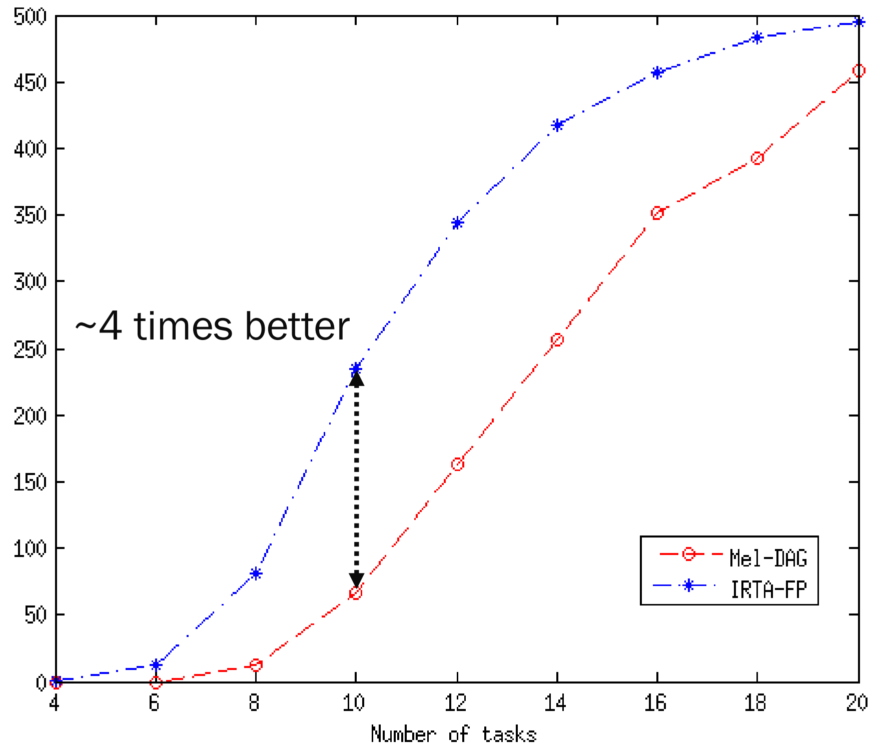
- System utilization *U*
- Number of tasks *n*
- Number of cores *m*

m = 8



Experimental Results

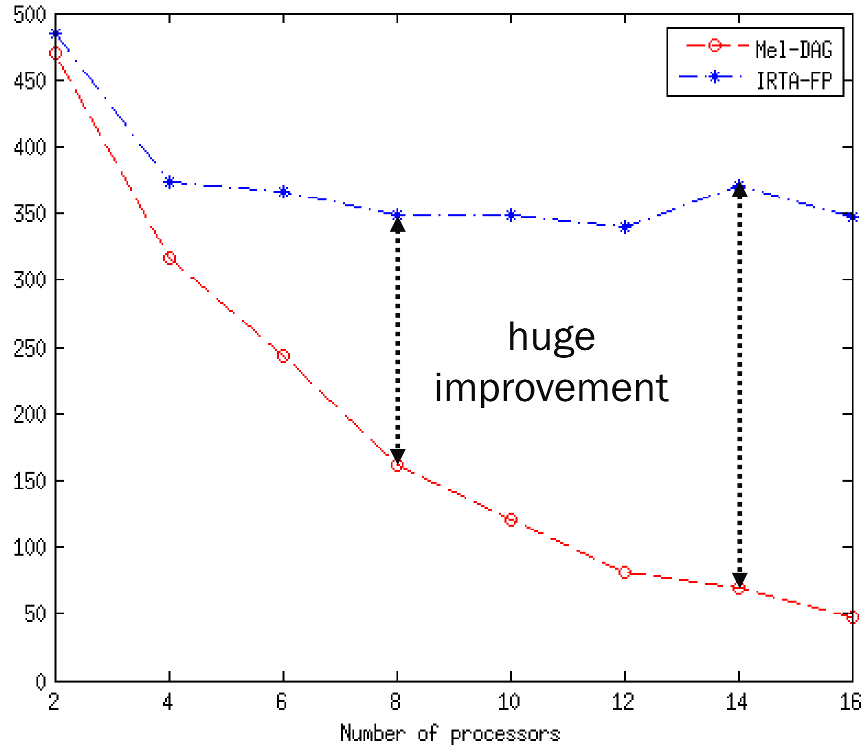
m = 8, U = 70%



Substantial schedulability improvements

Experimental Results

U = 70%, n = 1.5m



Robust to systems with increased number of cores

Summary

Addressed DAG tasks under G-FP scheduling

Introduced the notion of workload distribution

- Models the shapes of different schedules

Proposed two techniques to more accurately characterize the worst-case carry-in and carry-out workload

- DAG’s **internal structure** is explored

Experimental results reported significant gains in terms of schedulabity and effectiveness for large multiprocessor systems

Future work

- Address the pessimism in the self interference

Thank you!

jcnfo@isep.ipp.pt