

Dynamic Mapping and Ordering Tasks of Embedded Real-Time Systems on Multiprocessor Platforms

P. Yang and F. Catthoor
IMEC/K.U. Leuven

SEEDS FOR
TOMORROW'S
WORLD



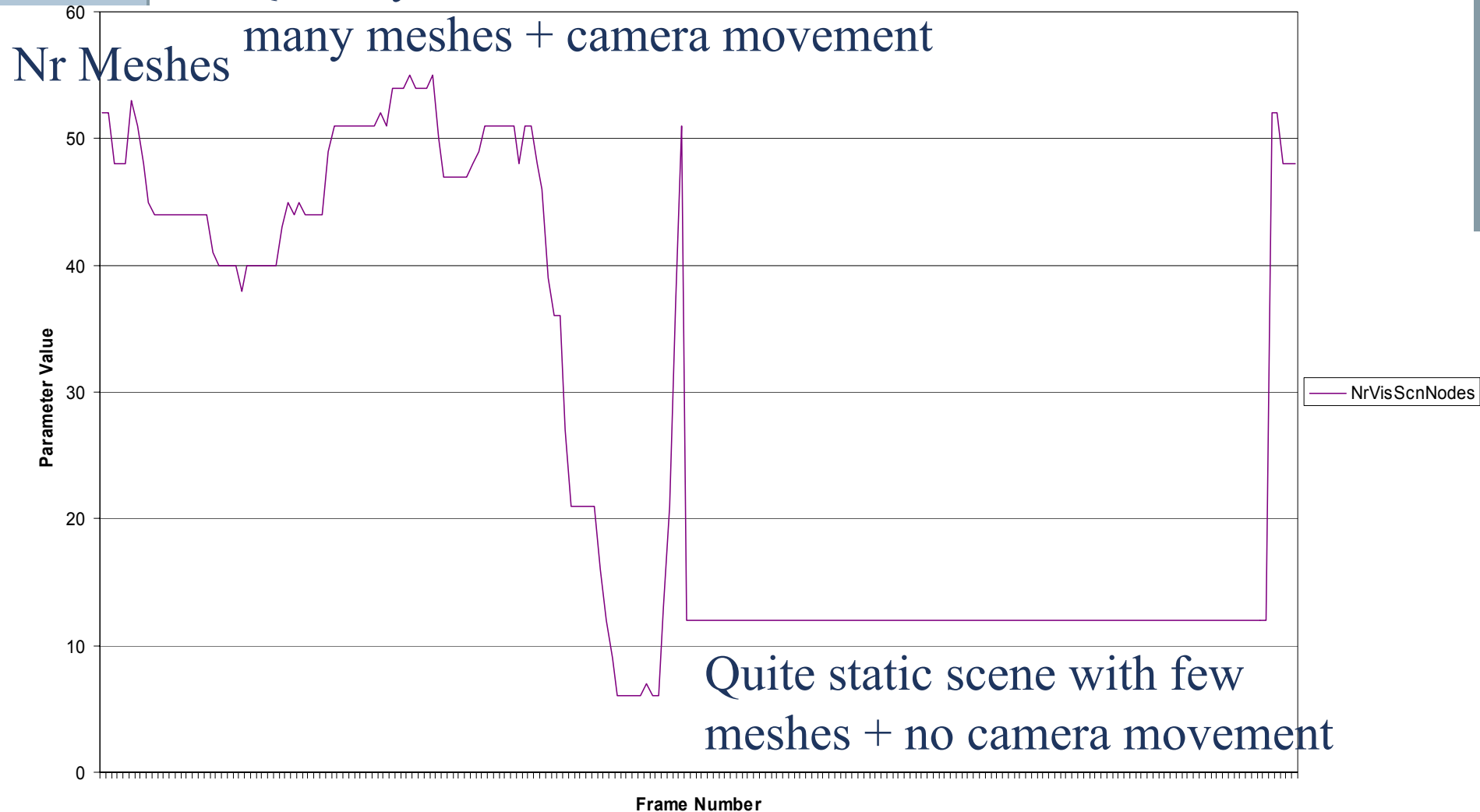
Terminal QoS (3D demonstrator)



Future multimedia applications tend to be highly dynamic

Quite dynamic scene with many meshes + camera movement

Quite static scene with few meshes + no camera movement



Goal of our research

A methodology to map *dynamic* and *concurrent* real-time applications on an embedded (heterogeneous) multi-processor platform



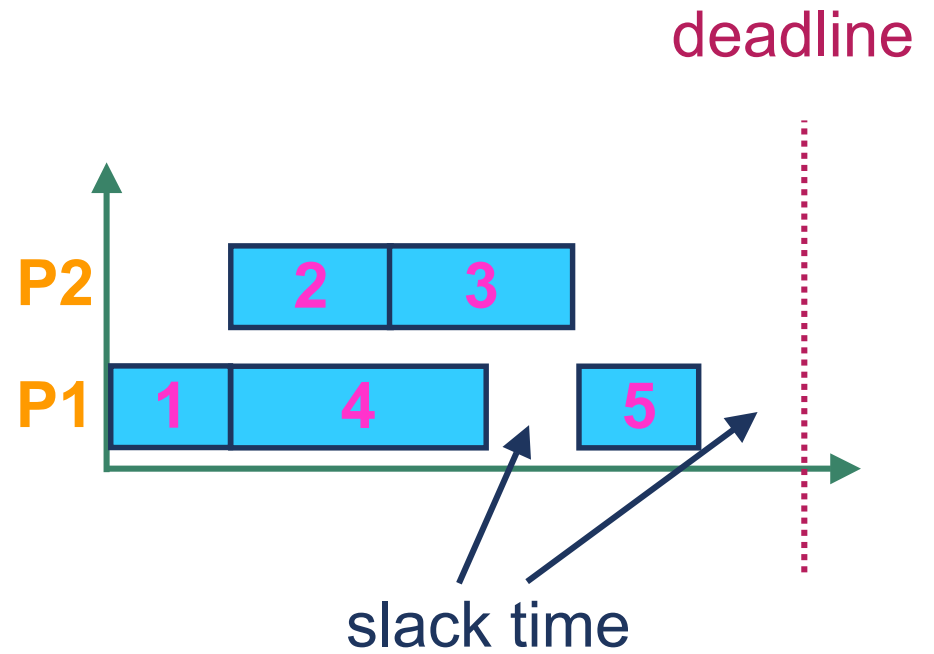
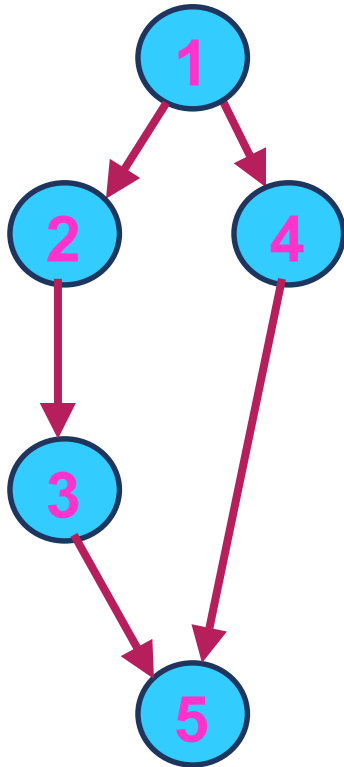
Goals of our research

- Satisfy the constraints on
 - deadline
 - quality
 - resources
 - dependencies
 - ...

- Cost can be any of the following or their combinations
 - energy, quality, latency, ...

- *Task Concurrency Management methodology*
Heuristic algorithm for run-time task scheduling
Mapping and ordering tasks dynamically
Conclusions

(Sub)Task scheduling



P1

P2

Task scheduling algorithms

- Traditional on-line task scheduling algorithms

- Round-robin, with or without time-slicing, for non-RT system

Rate Monotonic (RM)

No energy considerations!

- Variations of RM and EDF, considering dependency, resource, aperiodic tasks, ...

- Dynamic Voltage Scaling (DVS) aware algorithms

Mapping is fixed at design-time !

[Leng00], on-line, continuous/discrete voltage, single processor

- [Quan01], off-line, continuous voltage, multiprocessor

Design-time vs. run-time scheduling

- Design-time scheduling

- off-line
- assuming clairvoyance
- schedules planned for

- Run-time scheduling

- on-line
- no a priori knowledge

***Design-time scheduling alone
can not handle the dynamic features
of future applications!***

analyzable

✓ low overhead

✗ low utilization

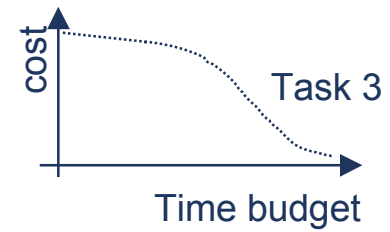
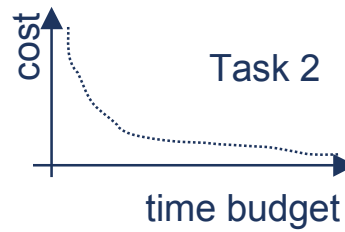
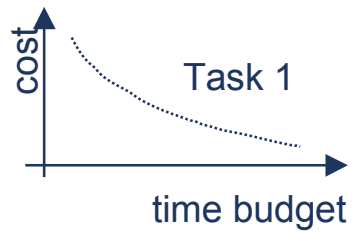
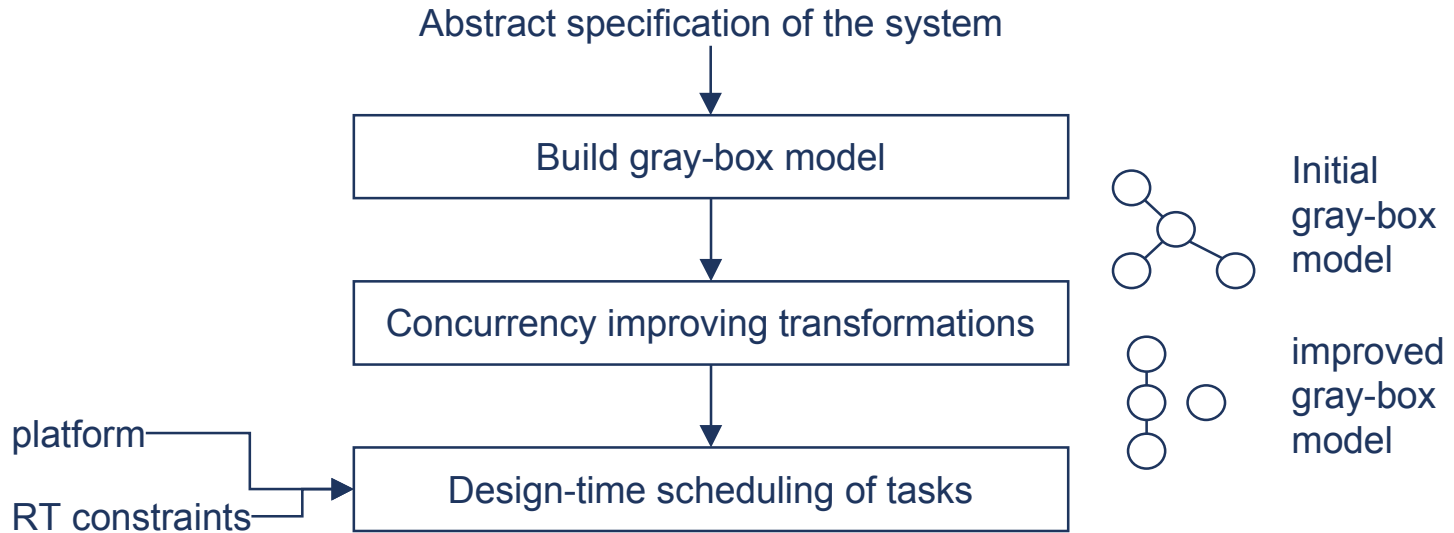
✗ less number of tasks
handled

✓ few assumptions about workload

✗ prone to overloads

✗ consumes processor time

✗ only statistical guarantees



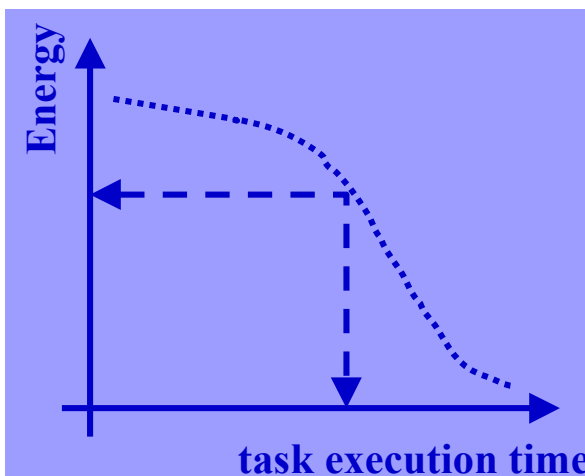
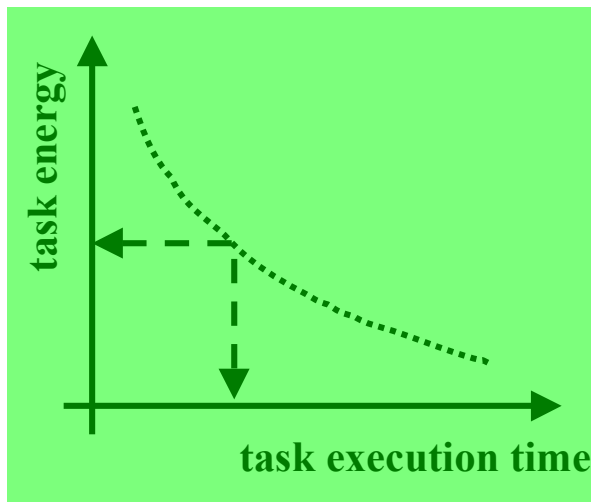
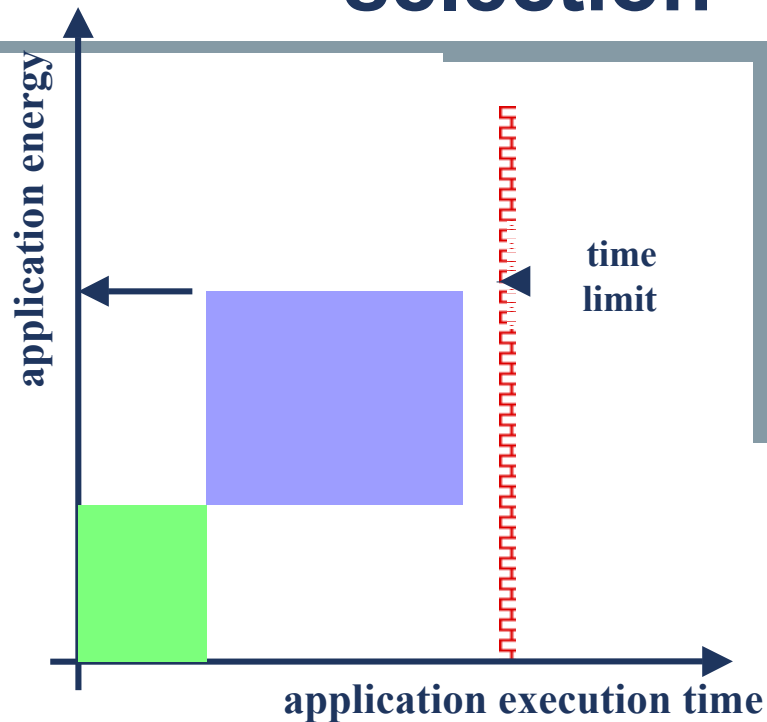
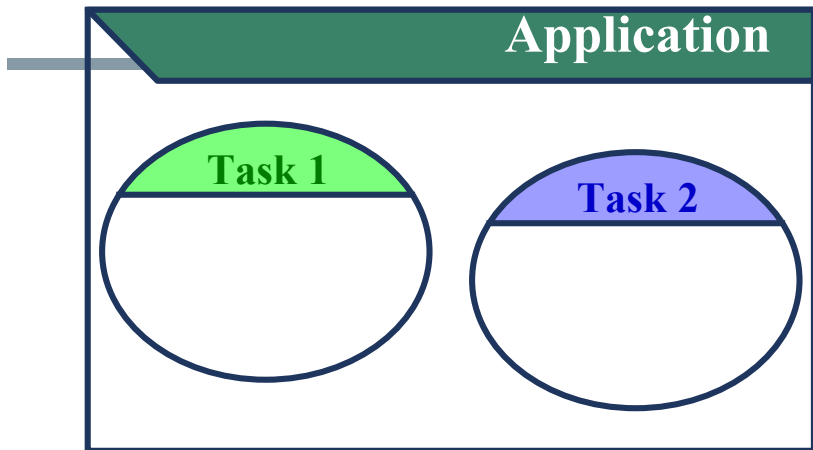
platform →

RT constraints →

Run-time scheduler

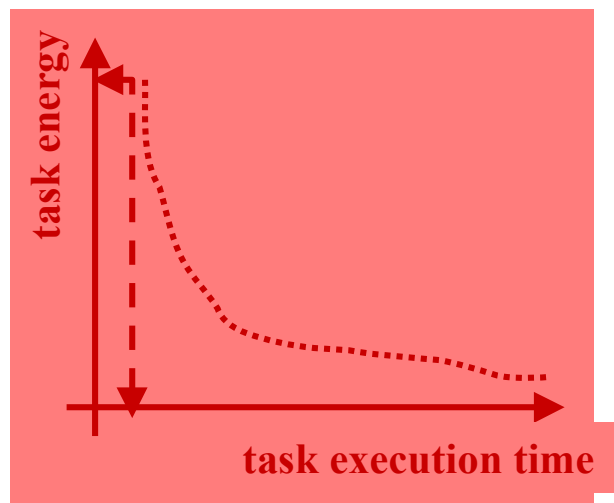
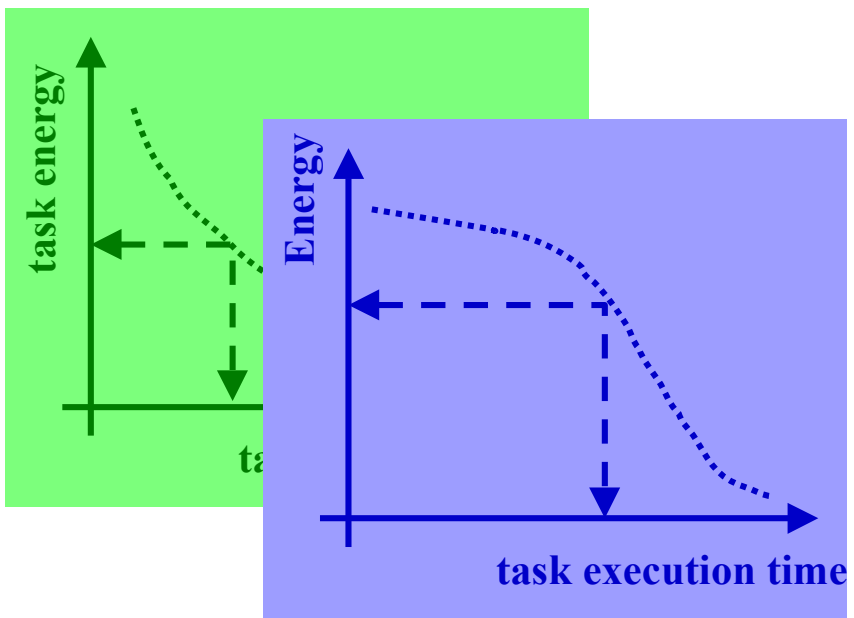
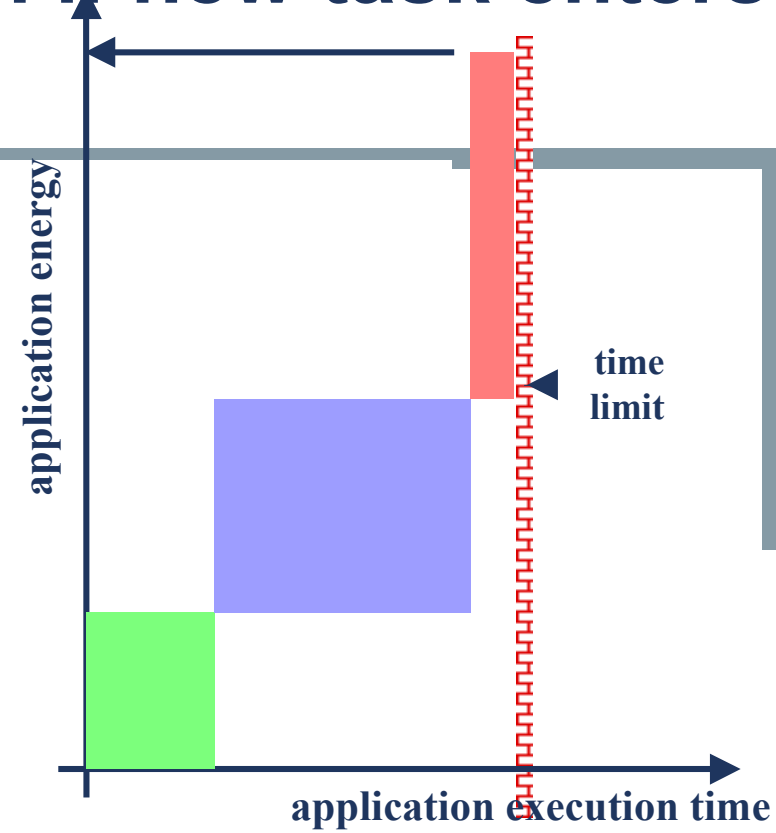
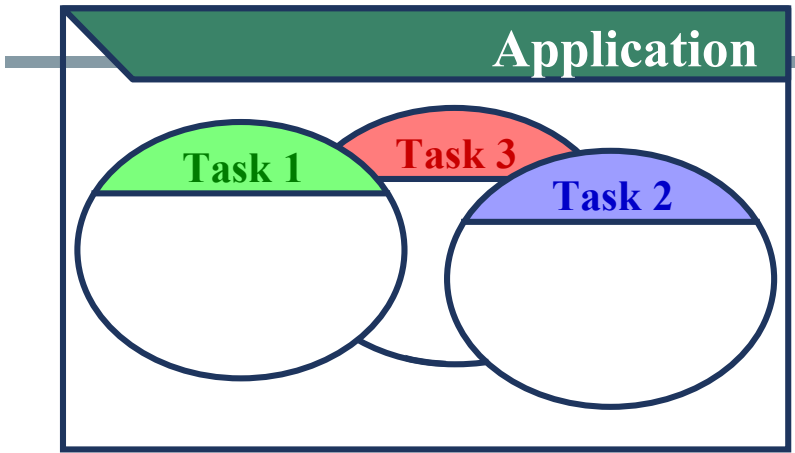


Run-time: original Pareto point selection

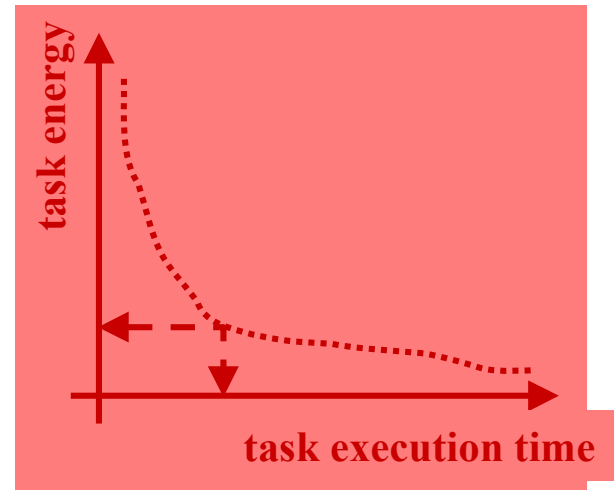
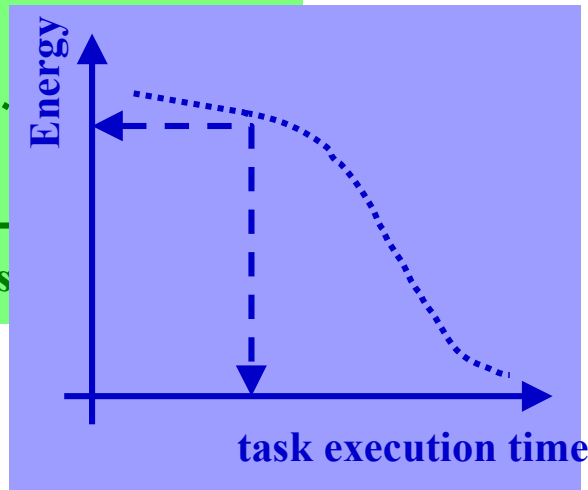
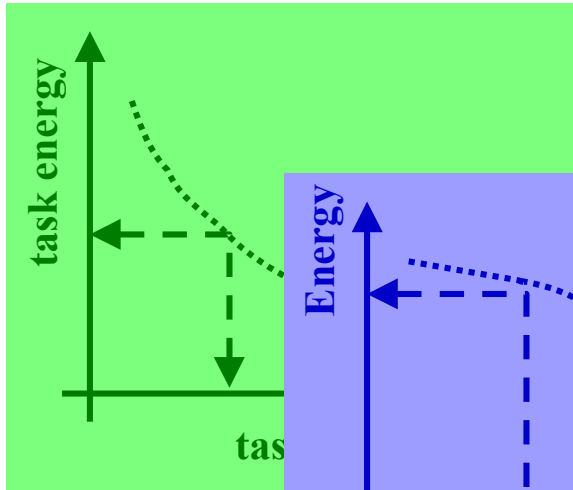
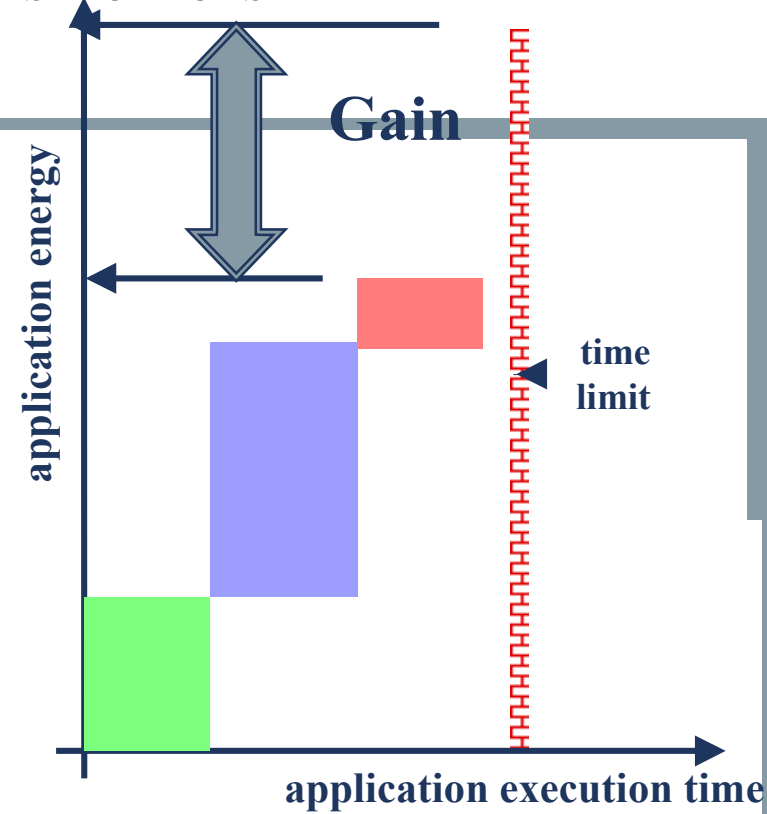
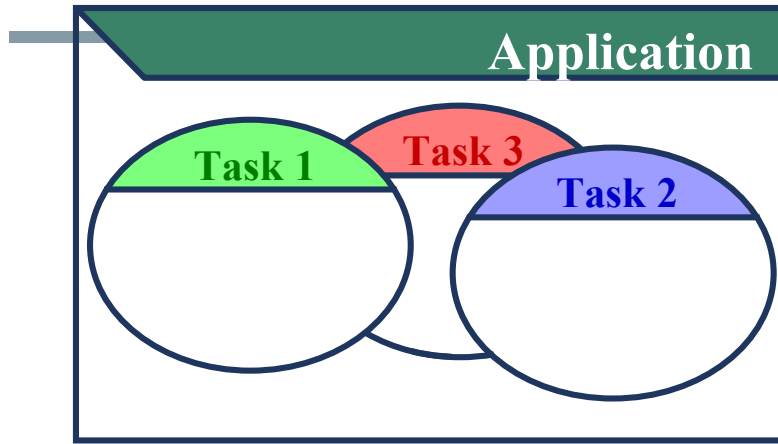




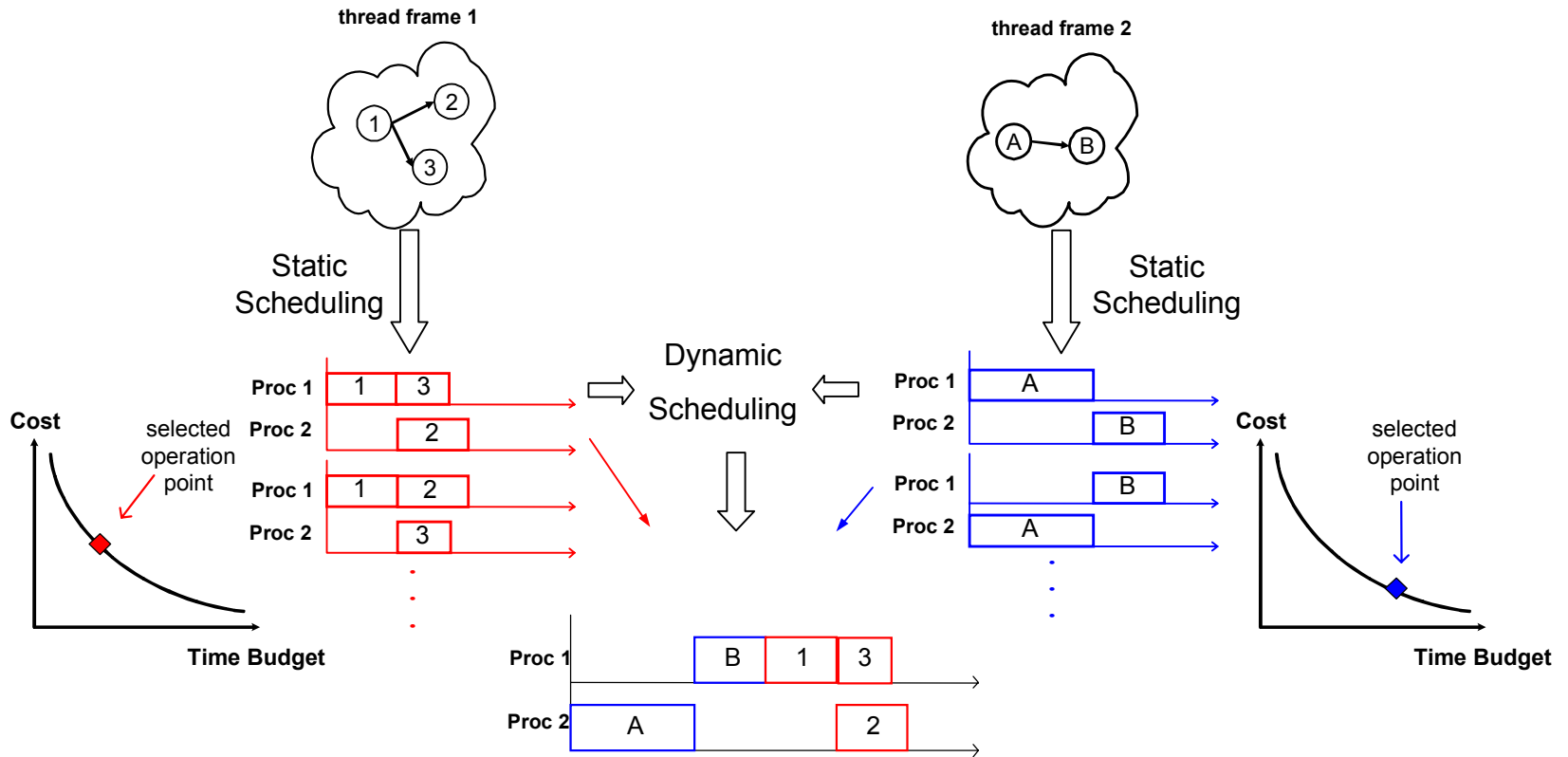
Run-time: one selection if new task enters



Run-time: better selection if new task enters



2-phase scheduling, example



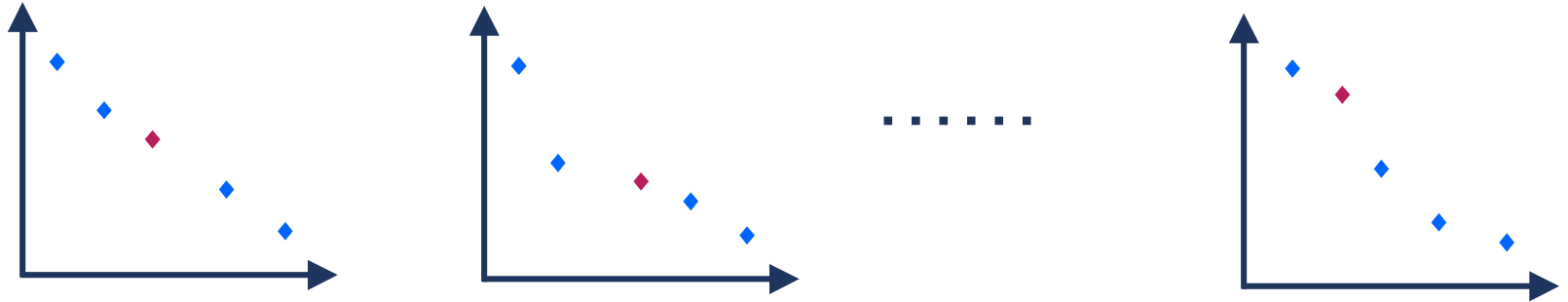
Task Concurrency Management methodology

➤ *Heuristic algorithm for run-time task scheduling*

Mapping and ordering tasks dynamically

Conclusions

Problem to solve



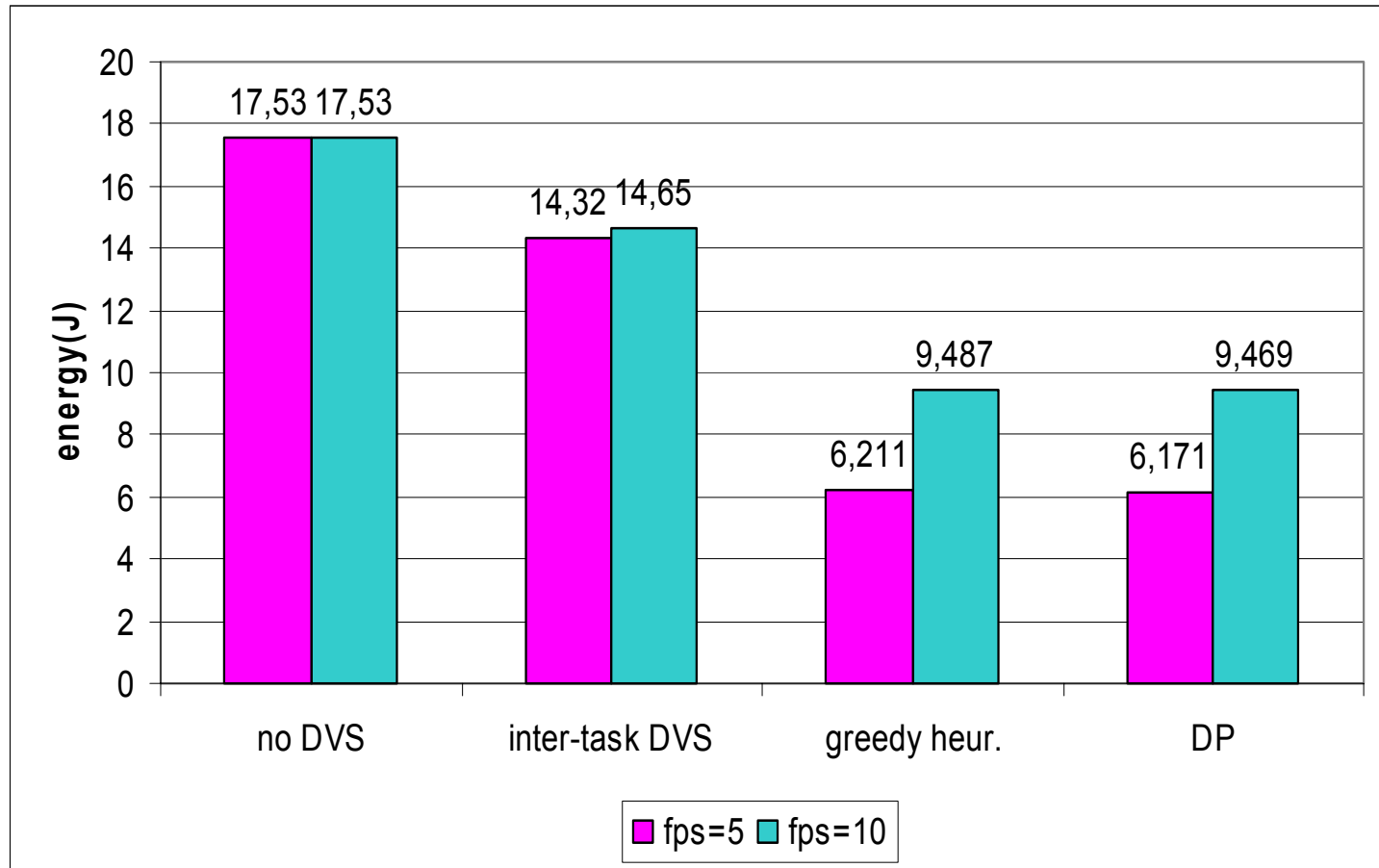
Multiple Choice Knapsack Problem!

[ISSS 2003]

Why do we need a new heuristic?

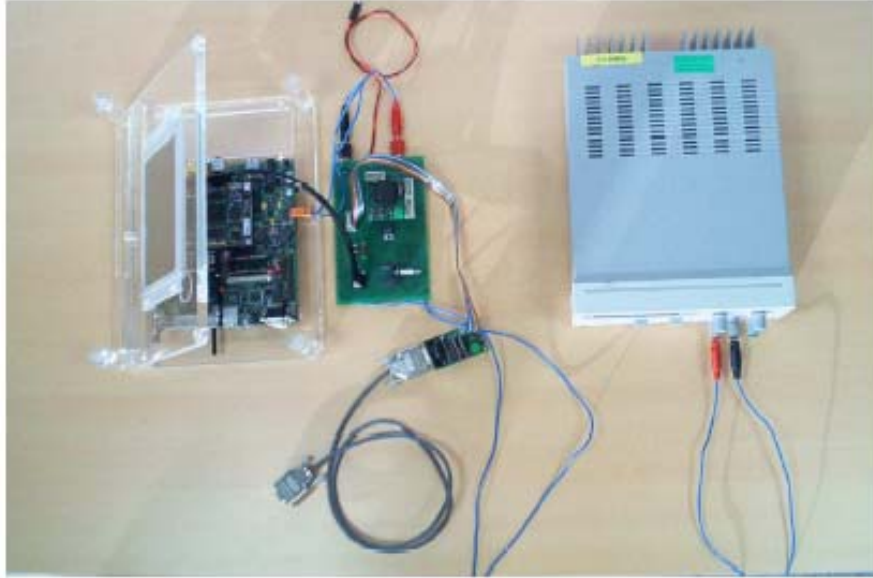
- Exact solver: not practical due to the NP hard feature
- Existing heuristics:
 - Are designed for big problems;
 - Rival each other in
 - which can get a solution closer to the optimal solution
 - which can solve a bigger (more difficult) problem
- Goals of our heuristic:
 - Find good **enough** solution in as **short** as possible time (10-100 K cycles) for a **reasonable** problem size (10-20 thread frames);
 - Should allow iterative improvement when it is possible.

3D Quality of Service (QoS) result



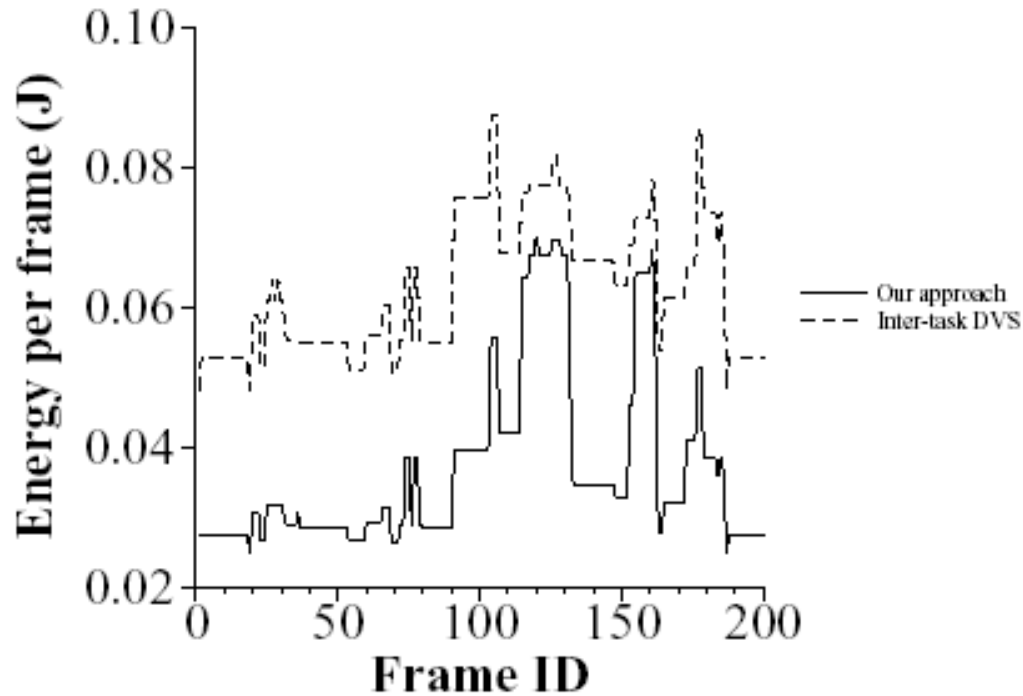
65% energy saving for 5 fps, 46% for 10 fps

PocketGL 3D demonstrator on Xscale board



Acunia Xingu development board with Xscale 80200
Hardware DC-DC converter
Modified Linux 2.4.16 for DVS control
Roundabout 3D rendering application

Significant energy saving with low implementation overhead



	No DVS	Inter-task DVS	Our approach
Max. frame rate at 1.5V	11.75FPS	11.71FPS	11.62FPS
Perf. overhead	0%	0.34%	1.1%
Executable size	2.43MB	2.44MB	2.46MB
Overhead	0%	0.57%	1.14%

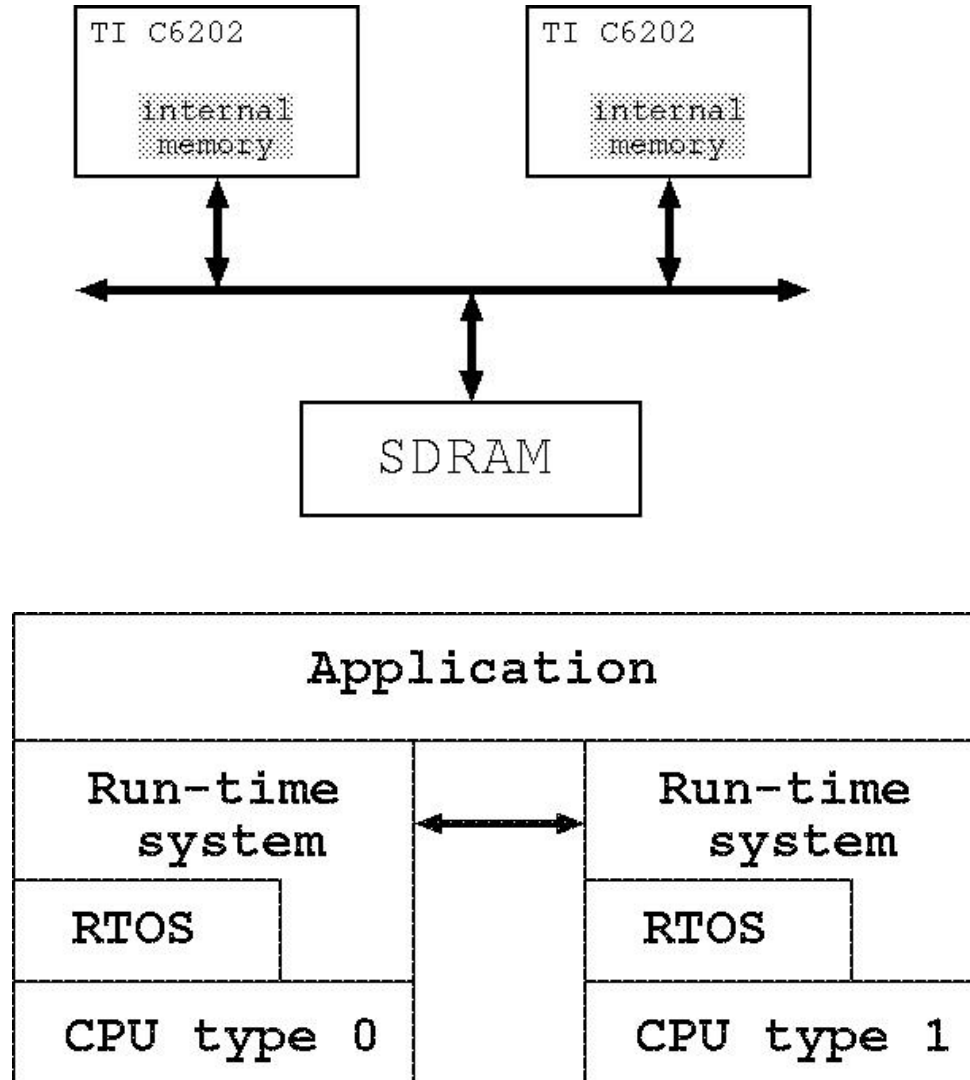
Task concurrency management methodology

Heuristic algorithm for run-time task scheduling

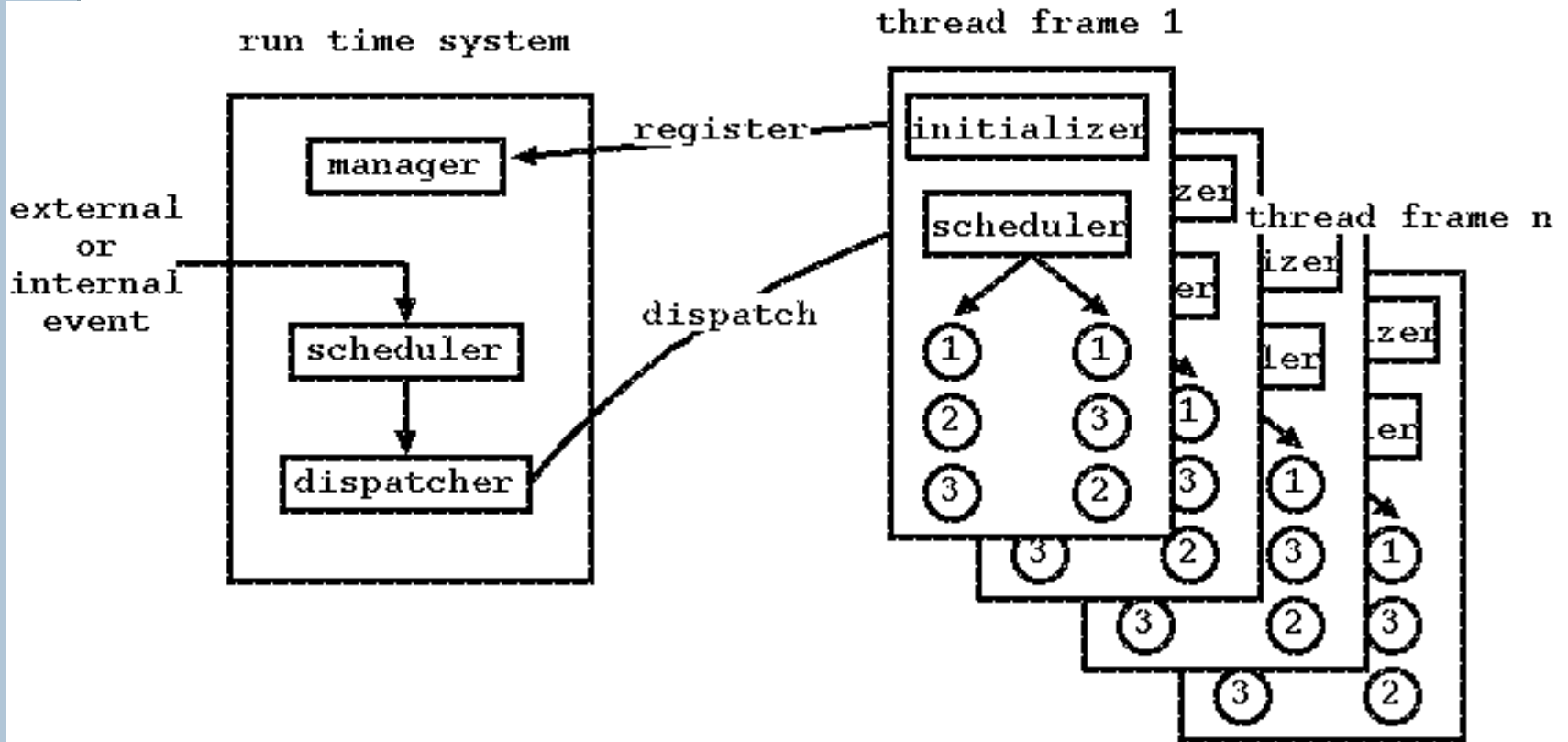
➤ *Mapping and ordering tasks dynamically*

Conclusions

Experimental Platform



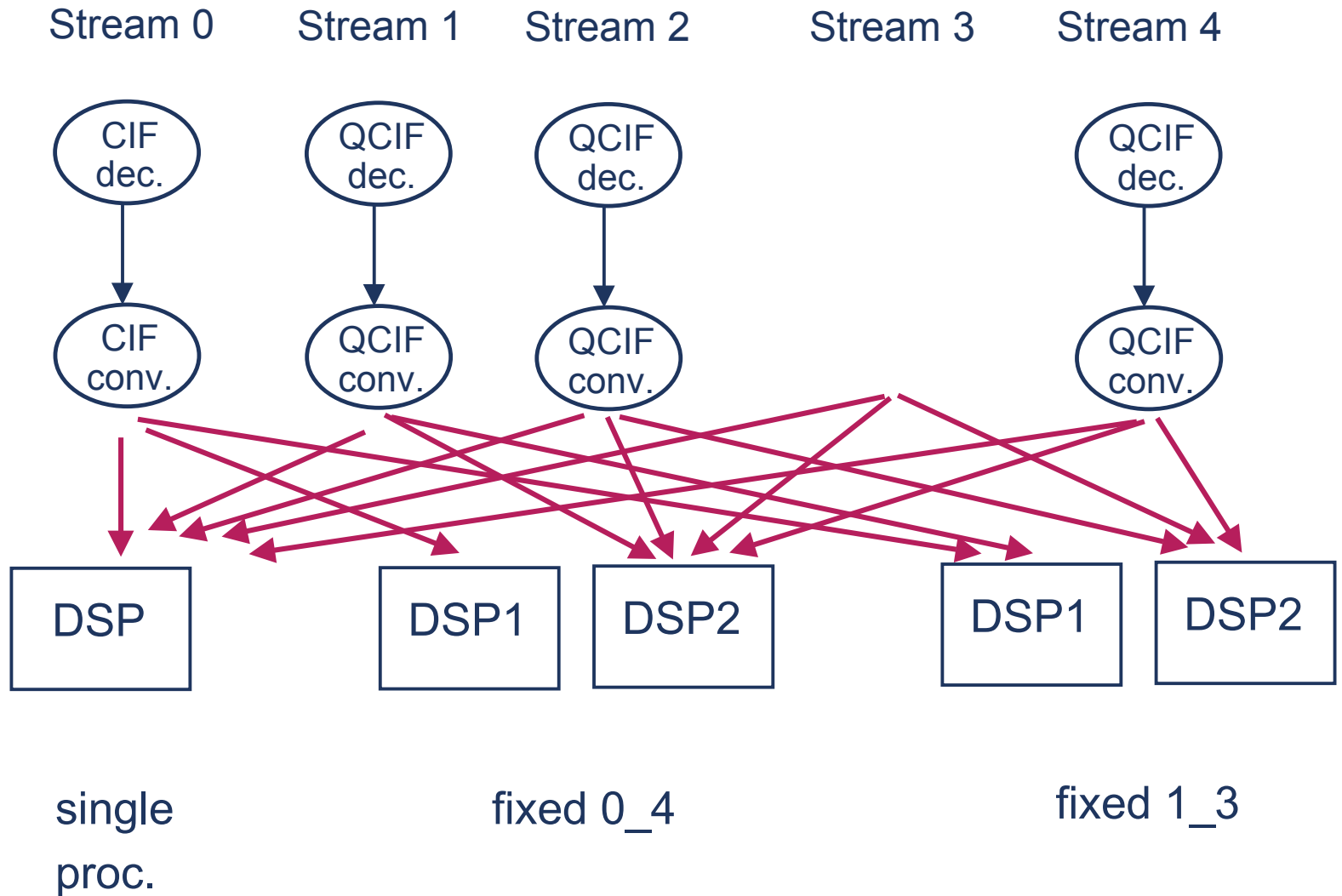
The Run-time System



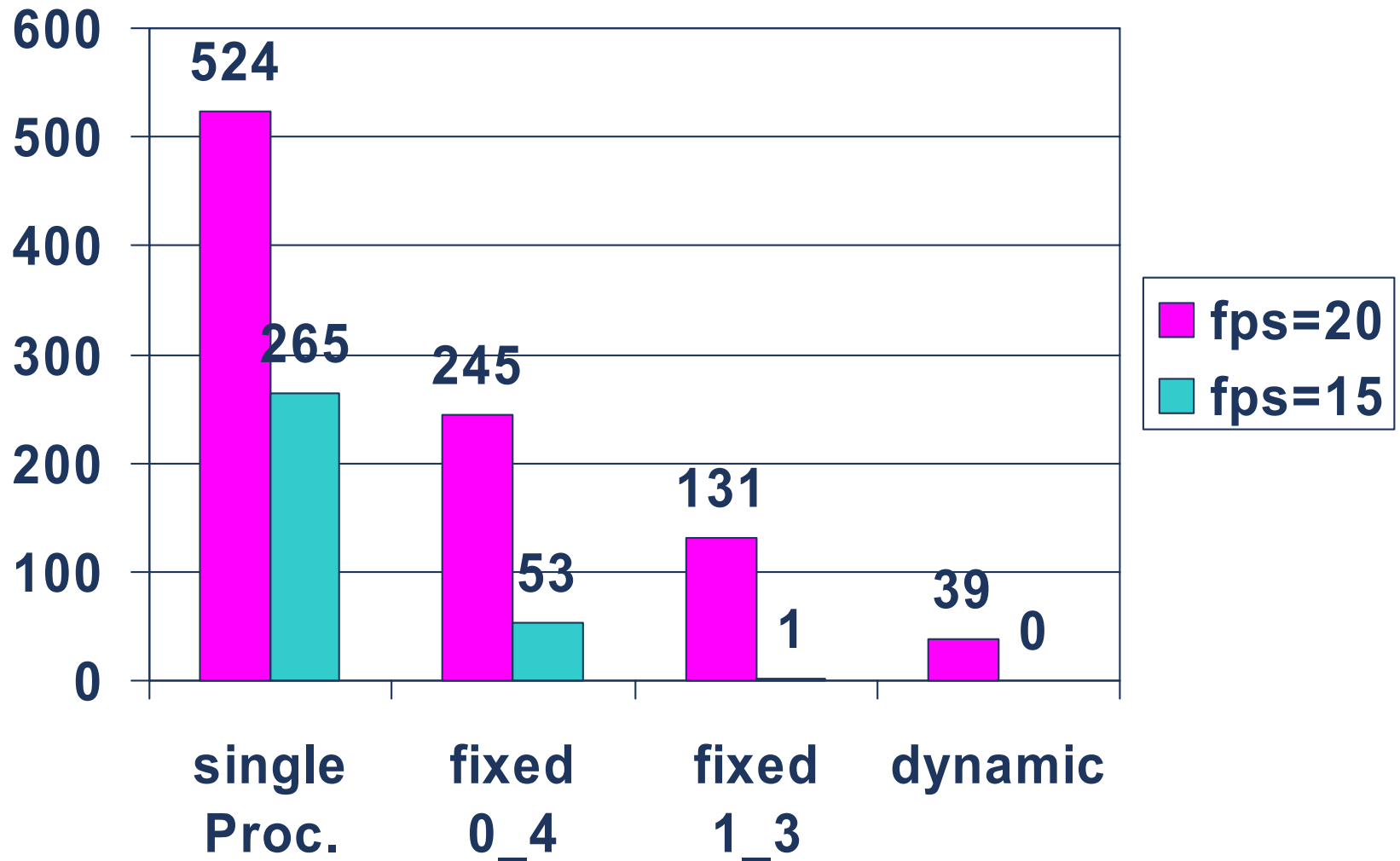
The realistic H.263 test case

- Telenor C reference code, tmn-1.7 decoder
- Support I, P and PB frames
- Clips from the standard test stream
 - Akiyo, Coastguard, Container, Forman and Hall
- Decoding 5 streams simultaneously
 - one CIF stream, each clip 100 frames
 - four QCIF stream, 5-50 frames each clip, randomly separated with 2-12 idle frames
- Simulated for 1000 periods

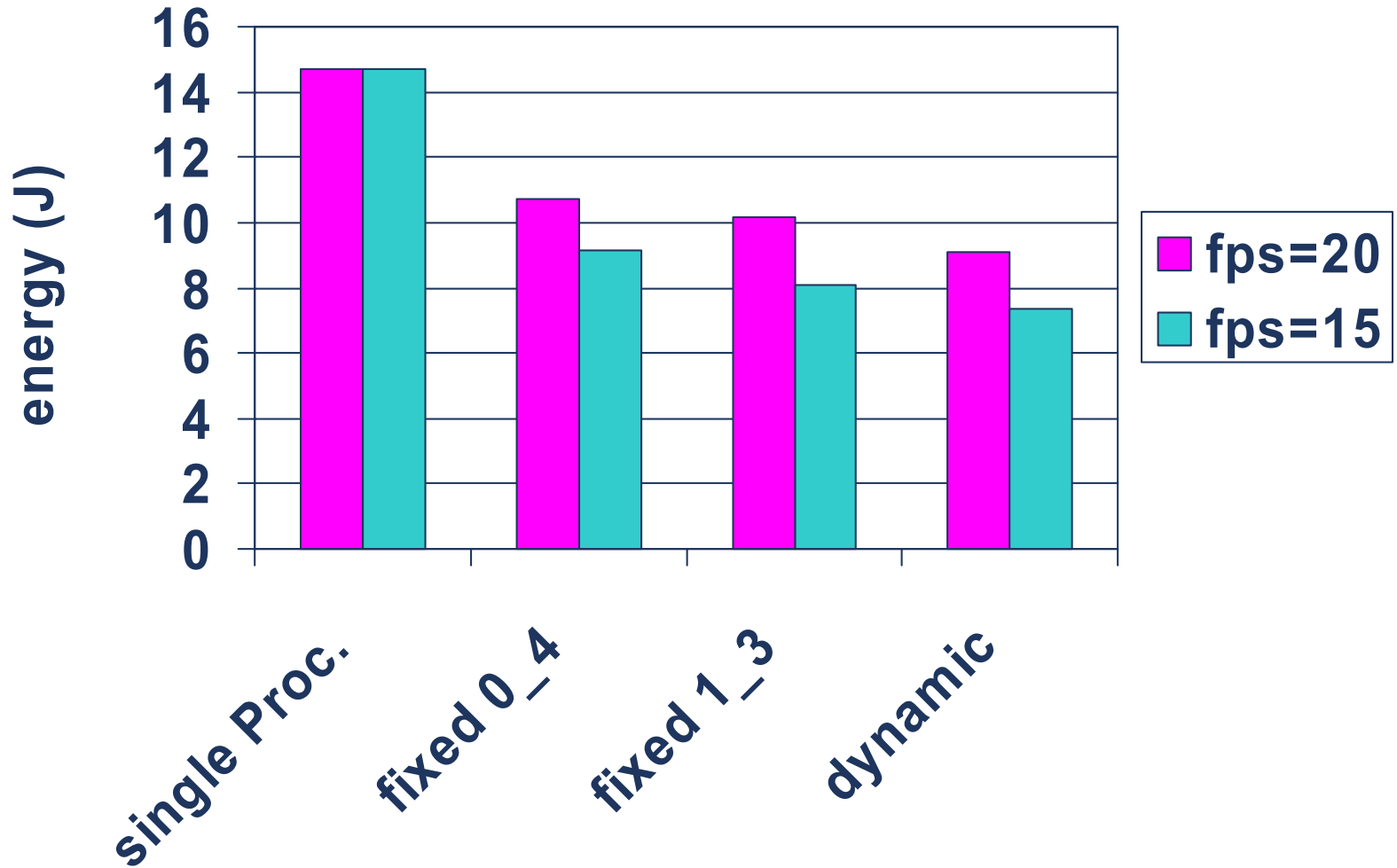
Reference cases



H.263 result: deadline miss



H.263 result: energy



50% energy saving compared with single Proc.(no DVS)

9% more energy saving compared with fixed 1_3 (fps=15)

- The TCM methodology enables us to explore the run-time tradeoffs
- Our run-time heuristic gives a fast and scalable solution within a reasonable error range
- Our middleware-like run-time system can map and order the tasks dynamically at a low overhead.