Power Aware Scheduling in Multicore Systems

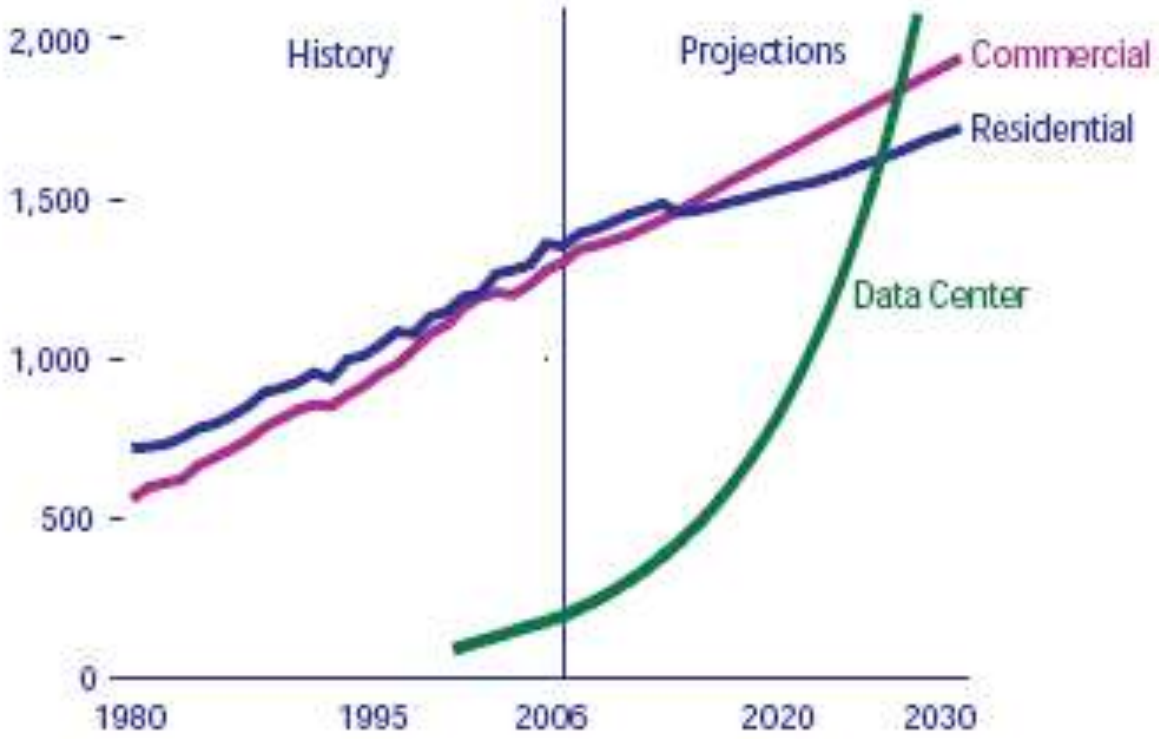
Rami Melhem

Department of Computer Science

The University of Pittsburgh

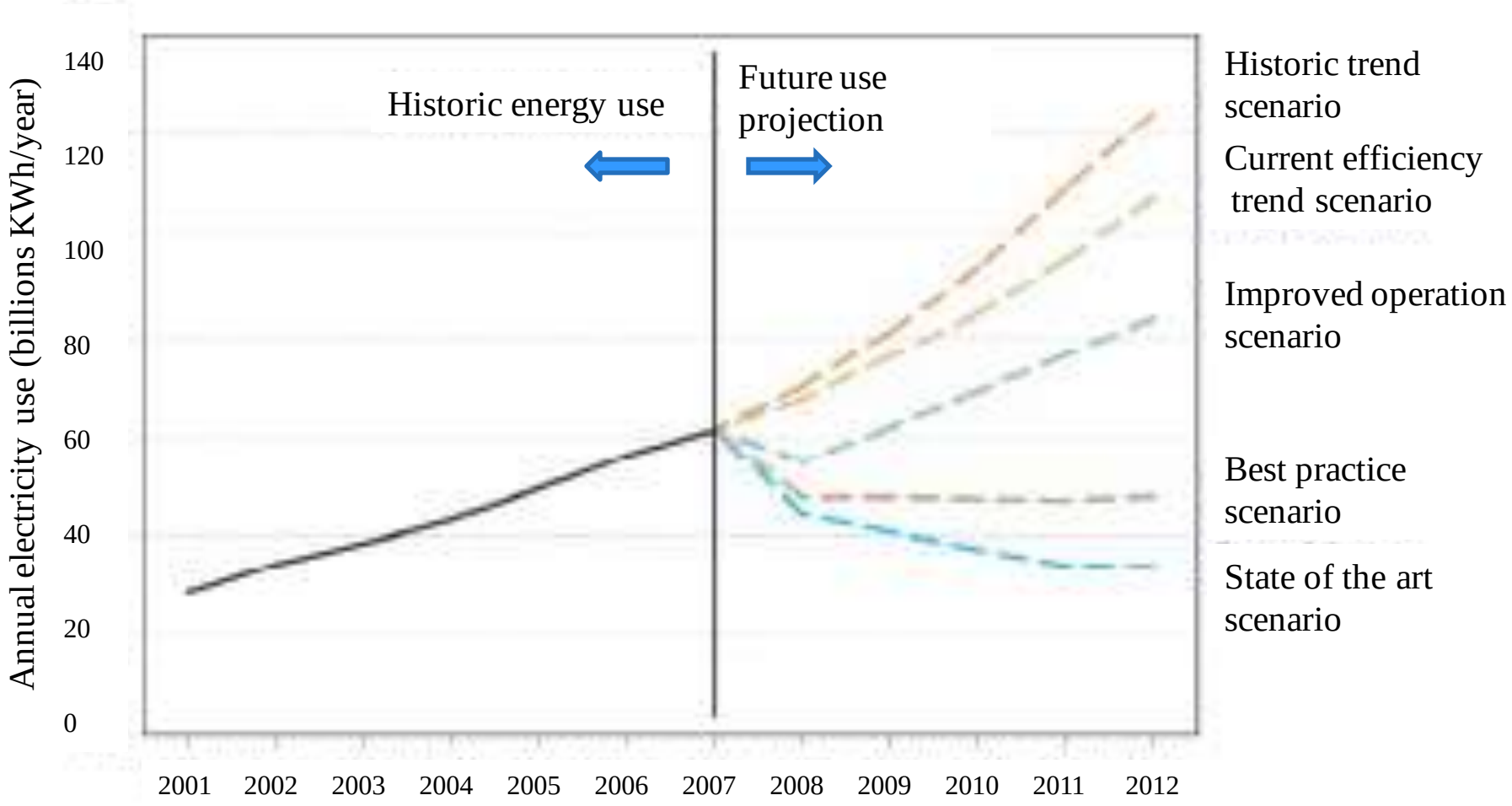
Why power aware scheduling?

It is estimated that 2% of the US energy consumption results from computer systems (including embedded and portable devices)



Electricity usage projection

Potential Data Center Electrical Usage[1]



- Introduction and Motivation
- Scheduling with dynamic voltage and frequency scaling
- Power management in multicores
  1) Assuming the Amdahl computational model
  2) Assuming structured applications
  3) Assuming unstructured applications
- Conclusion

Power Consumption in a Chip

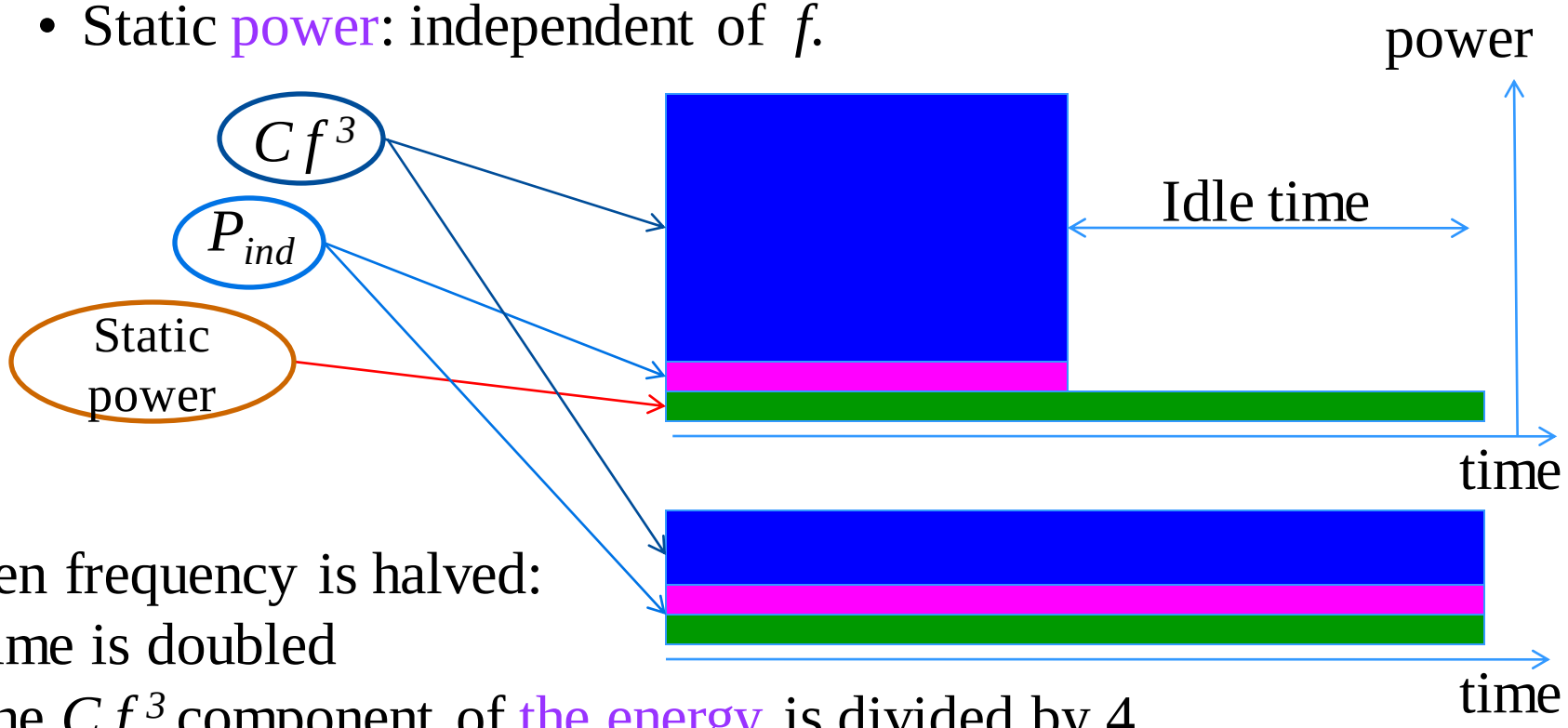
- Dynamic power: $P_{dynamic} \approx C\ V^2\ f + P_{ind}$
  - C: switch capacitance
  - V: supply voltage
  - f: operating frequency
- For a given technology, f and V are usually linearly related → $P_{dynamic}$ is cubically proportional to the processor's speed.
- P_{ind} actually depends on V but usually assumed a constant

- Static power: power components that are independent of f

- Two common management techniques:
  1) Processor throttling (turn off when not used)
  2) Frequency and voltage scaling

Frequency/voltage scaling

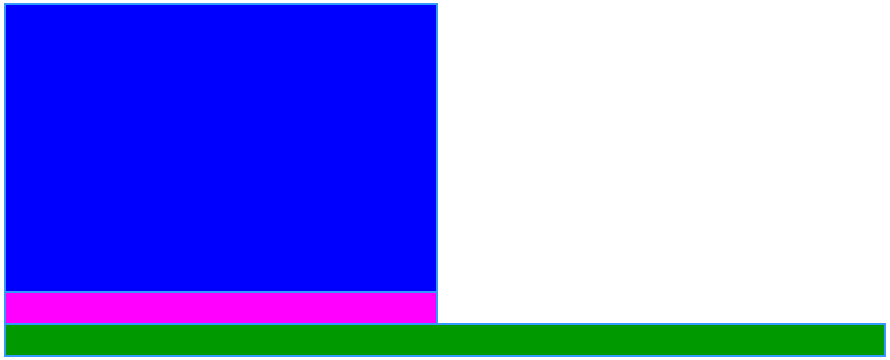
- Gracefully reduce performance
  - Dynamic power $P_d = C f^3 + P_{ind}$
  - Static power: independent of f.

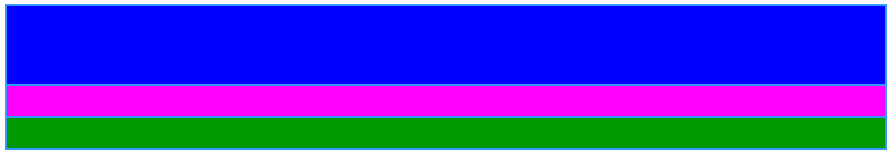


When frequency is halved:

- Time is doubled
- The $C f^3$ component of the energy is divided by 4
- The P_{ind} component of the energy is doubled

Frequency/voltage scaling





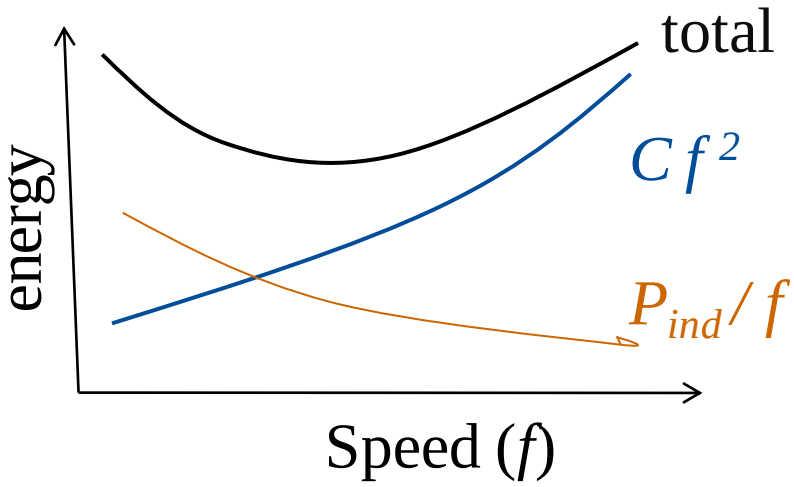
Slower speed

reduces the $C f^3$ component of the energy

increases the P_{ind} component of the energy

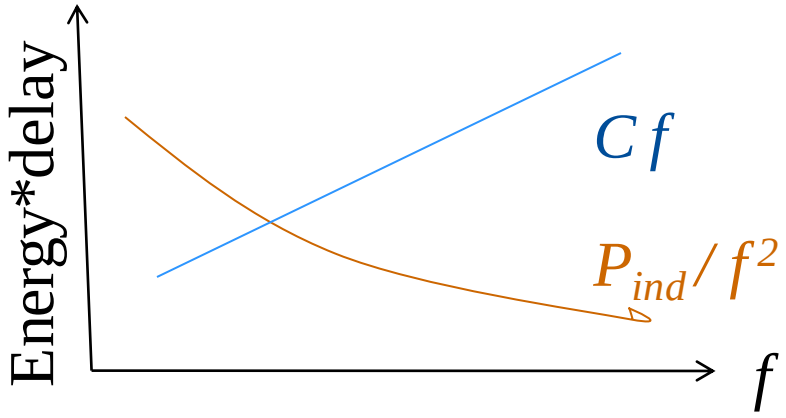
There is an optimal speed.

More complex when speeds are discrete



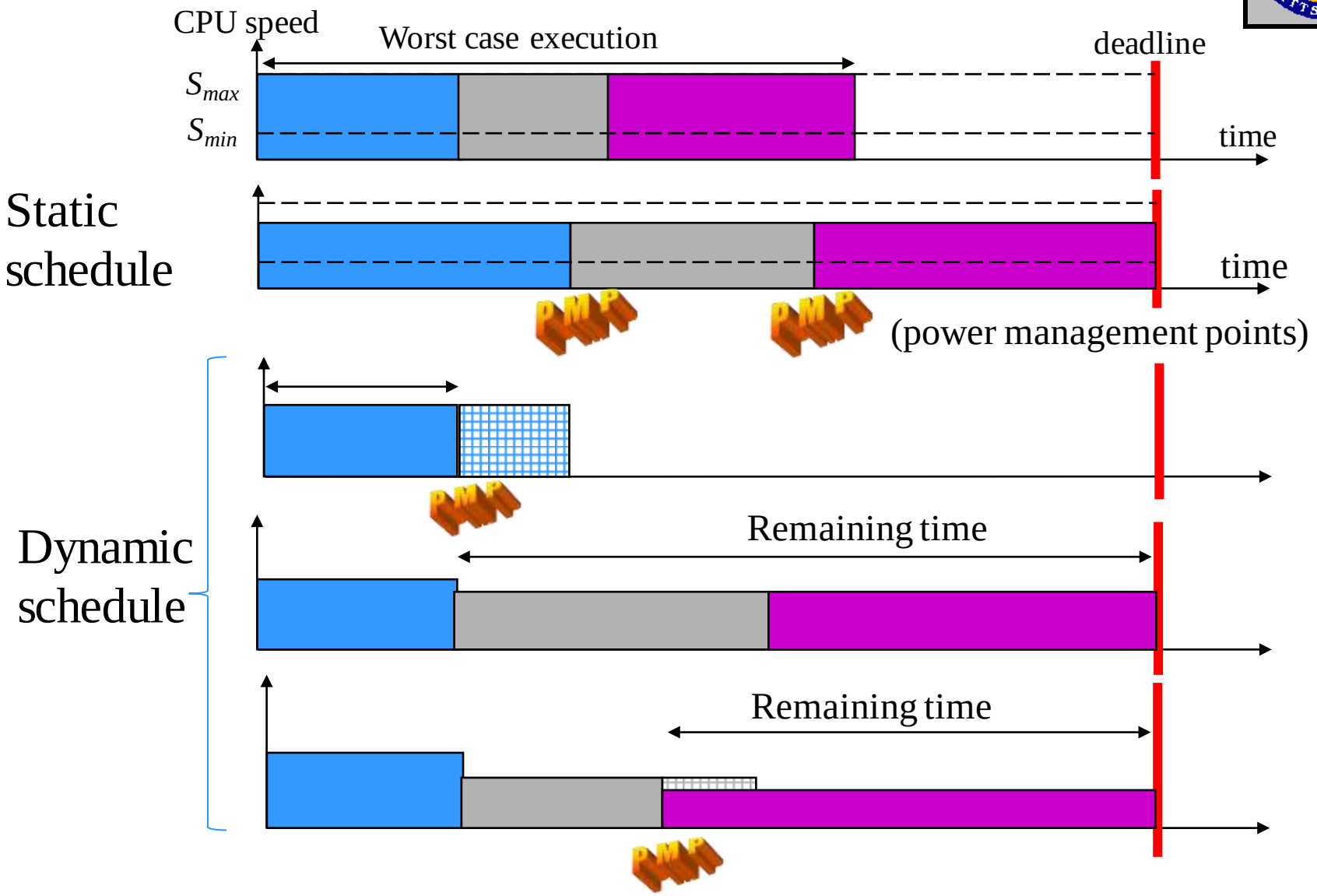
Different goals of power management

- Minimize total energy consumption
- Minimize the energy-delay product
  - Takes performance into consideration



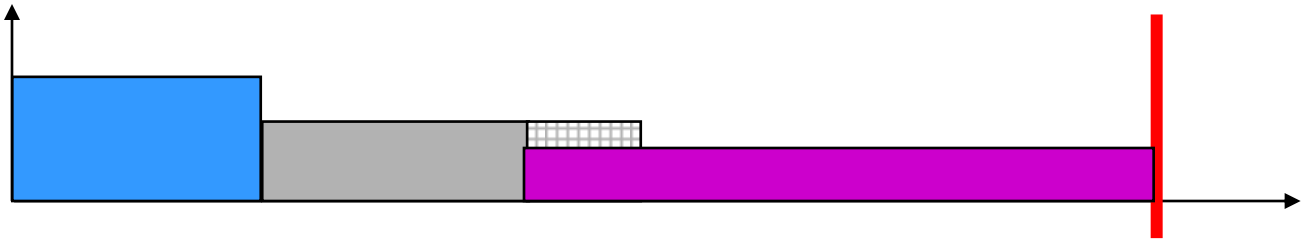
- Maximize performance given an energy (or power) budget
- Minimize energy given a deadline
- Minimize the maximum temperature

Static and Dynamic voltage scaling (DVS)*

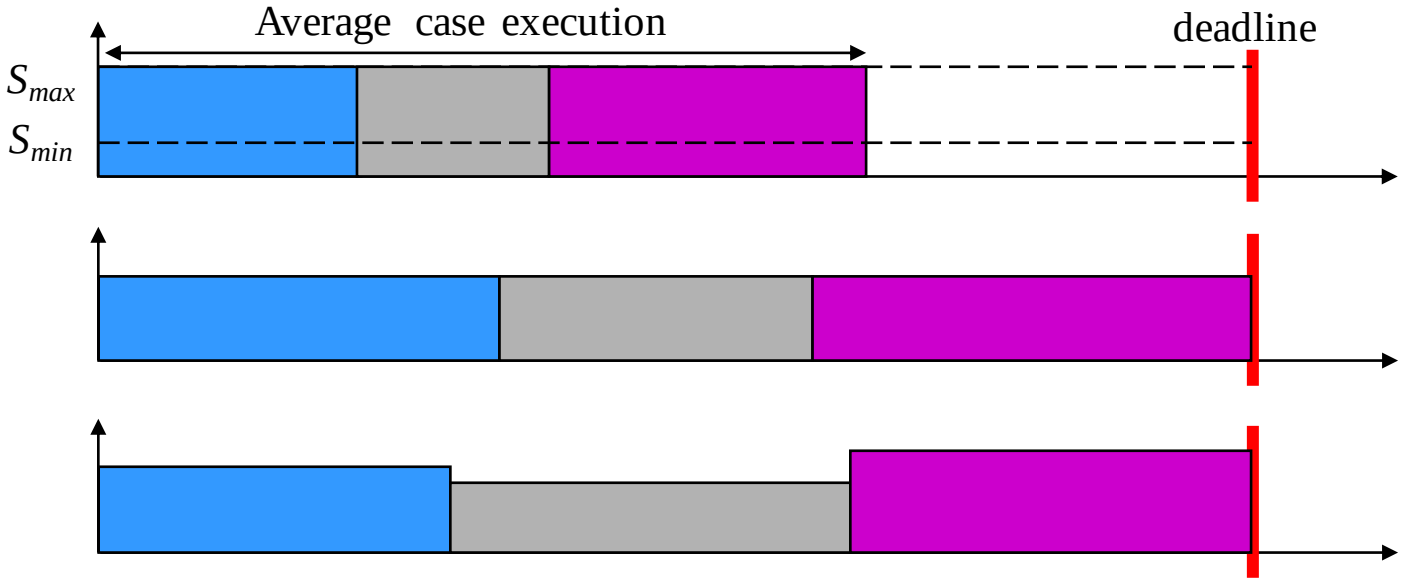


*) COLP 2000

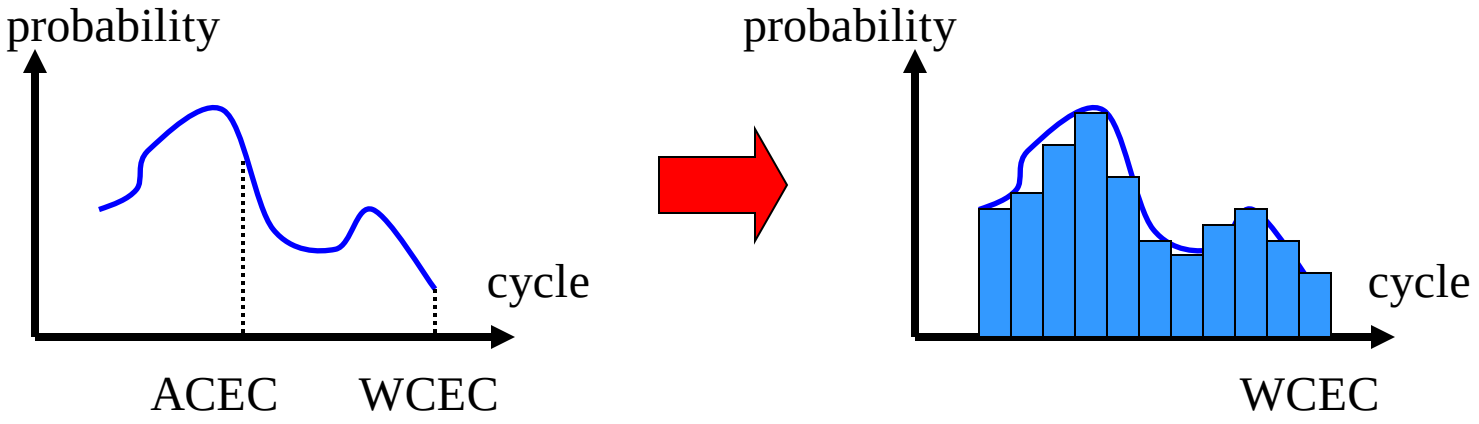
Static and Dynamic voltage scaling (DVS)



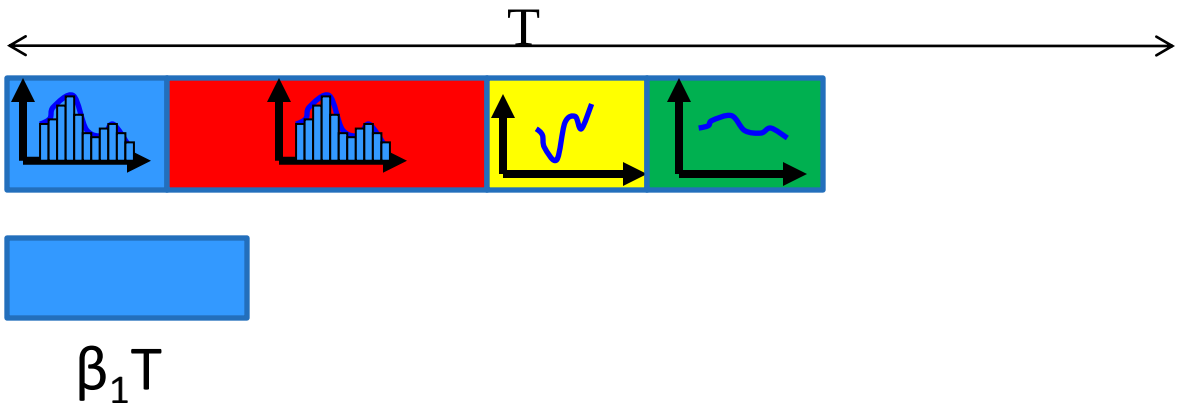
- Energy is minimum when execution speed is uniform
- Use statistical knowledge of execution time for the static scheduling rather than worst case execution time



Probability Distribution of Execution Cycles (a histogram for each task)*



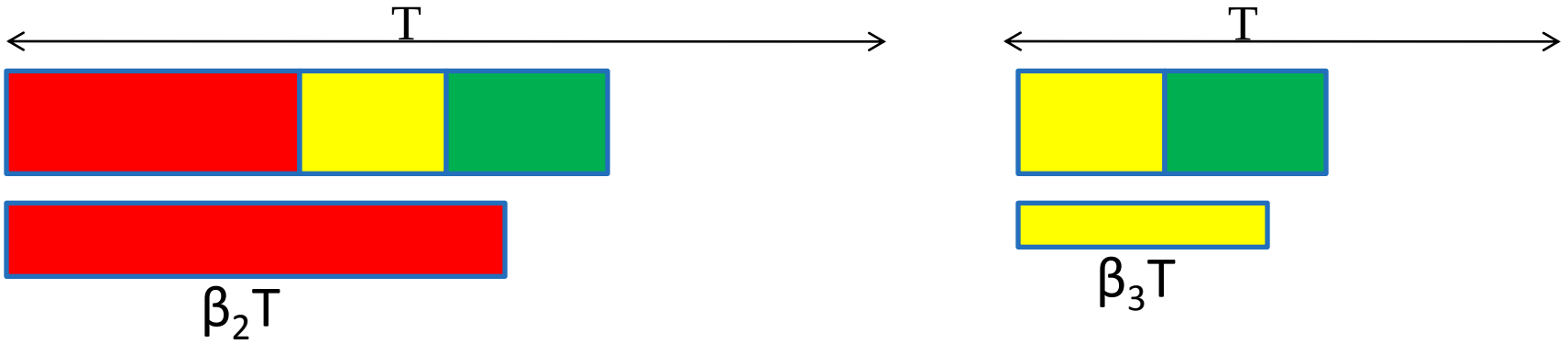
- Can use this knowledge to determine the fraction, β_i , of the remaining time, T, to allocate to the i^{th} task for **minimum expected energy consumption.**



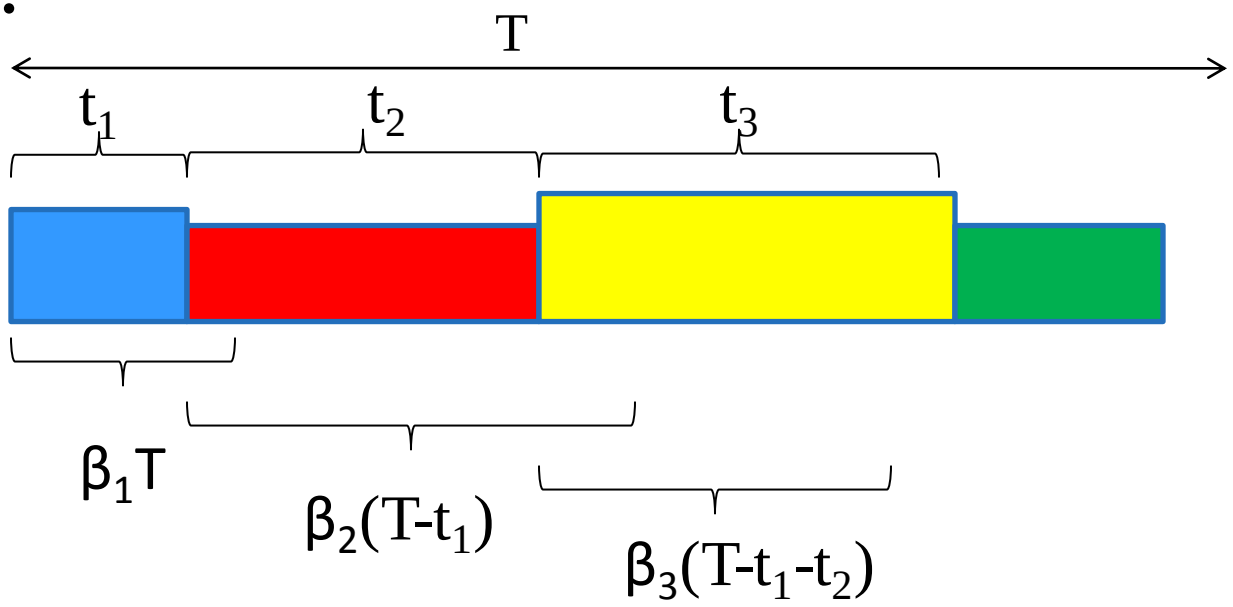
*) ACM Transactions on Computer Systems (2007)

DVS to minimize expected energy consumption

Offline:



At run time:



Power Management in Multicores

We will consider three task models:

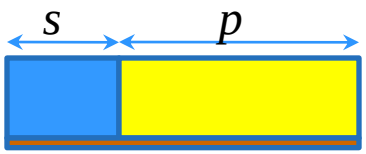
1) The Amdahl model (perfect parallel sections)
2) Structured computation (streaming applications)
3) Computations with unknown structures

DVS for multiple cores*

To derive a simple analytical model, assume Amdahl's law:

- *p* % of computation can be perfectly parallelized.

One core



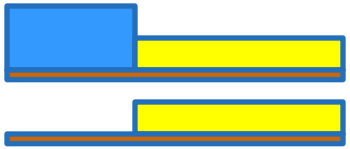
Two cores

Manage energy by determining:

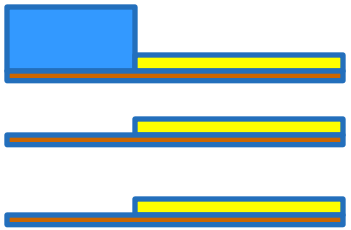
- The speed for the serial section
- The number of cores used in the parallel section
- The speed in the parallel section

Slow down the cores

Slow down the parallel section



Using more cores

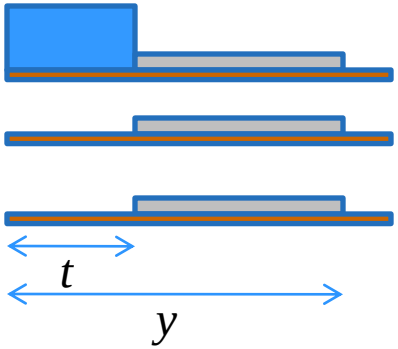


*) TPDS 2010

A model to Study Parallelism, Performance & Energy Consumption

- Initial assumptions
  - *P*rocessor cores consume static power (cannot turn off completely)
  - Dynamic power proportional to f^{α}
  - Maximum "relative" processor speed = 1, at which
    - Dynamic power = 1
    - static power = λ

- Question 1:
  - Find processors' speeds for minimum energy consumption?
- To find optimal speeds
  - Write energy expression, E in terms of t and y
  - Solve for $\partial E / \partial t = 0$ and $\partial E / \partial y = 0$

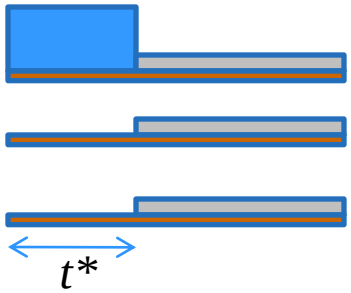


- May do the same for find the speeds that minimize the Energy-Delay product

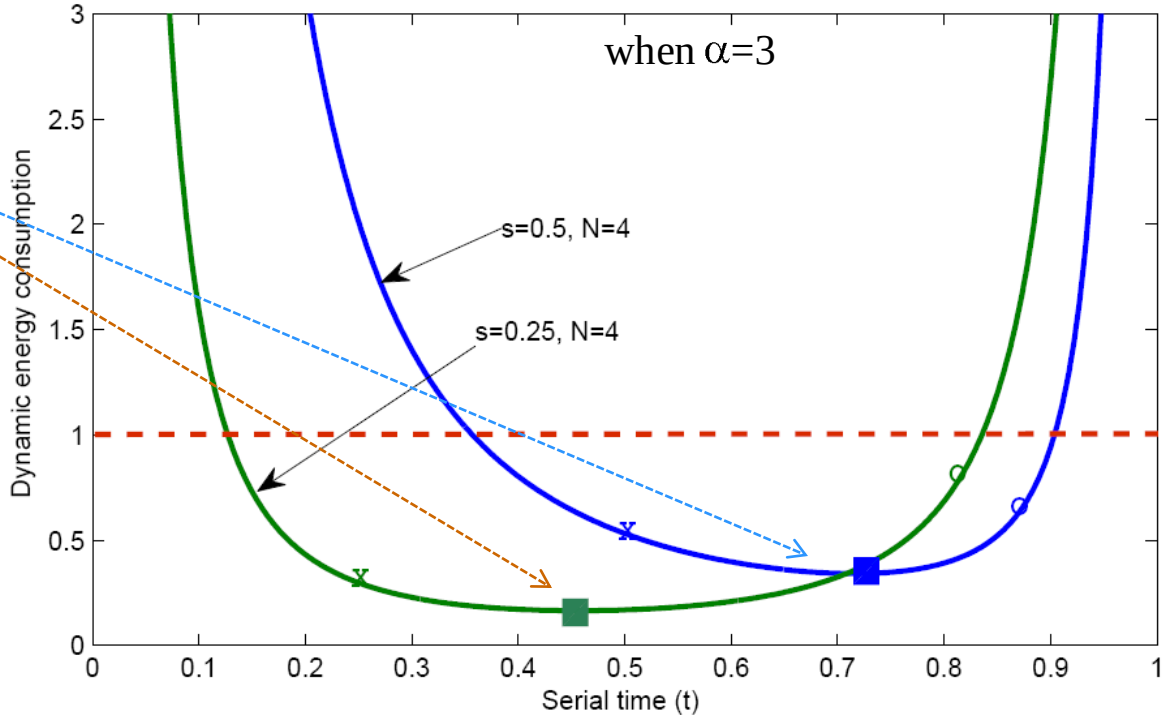
Example: parallelism is used for energy (parallel speedup = 1)

$$t^* = \left(\frac{\alpha - 1}{\lambda N}\right)^{\frac{1}{\alpha}} \cdot s$$

s = % of the serial computation
N = # of processors



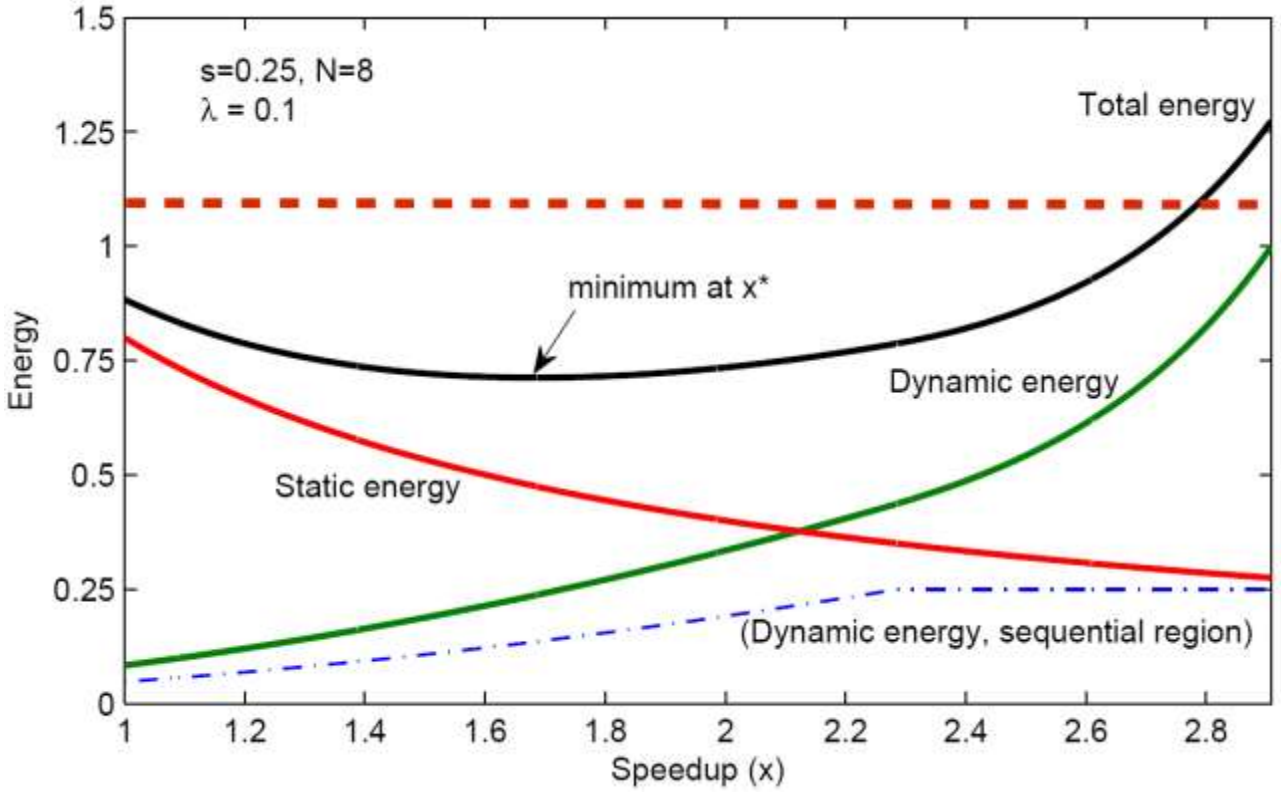
Energy consumption



$$Improvement\ in\ Dynamic\ Energy = \frac{1}{\left(s + \frac{p}{N^{(\alpha-1)/\alpha}}\right)^{\alpha}}$$

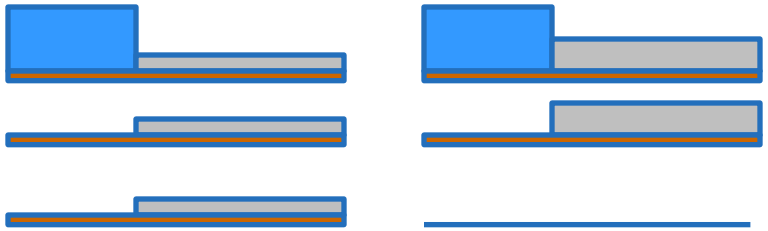
Usage of the model

- Find processor speeds for minimum energy consumption
- Find effect of static power on optimal energy consumption
- Optimize energy for a given speedup (performance)

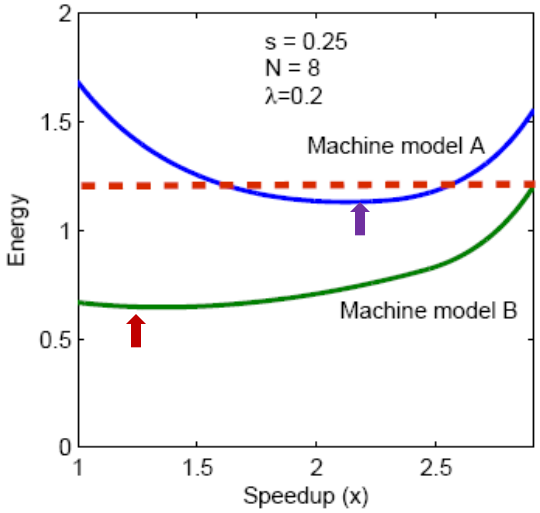


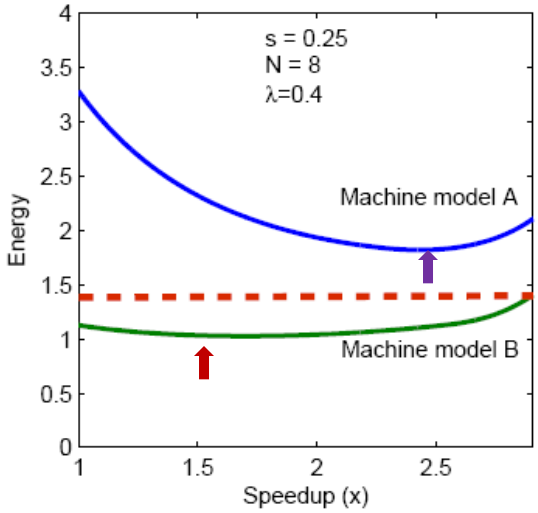
An Alternate system model

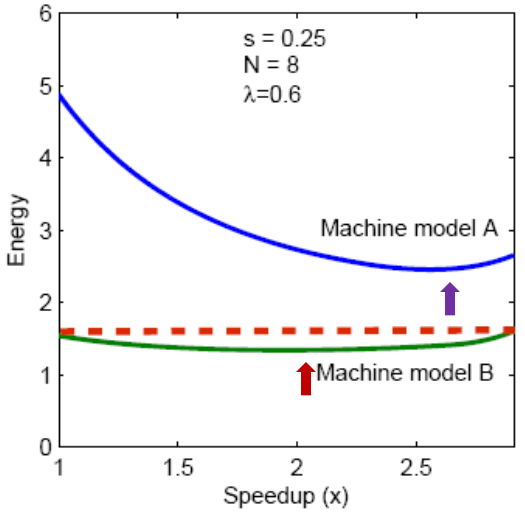
- Model B: can turn off individual processors
- To minimize energy (or energy-delay), we now need to
  - Find the number of processors to use, and
  - The processors' speeds



Minimum Energy at Different Speed Targets







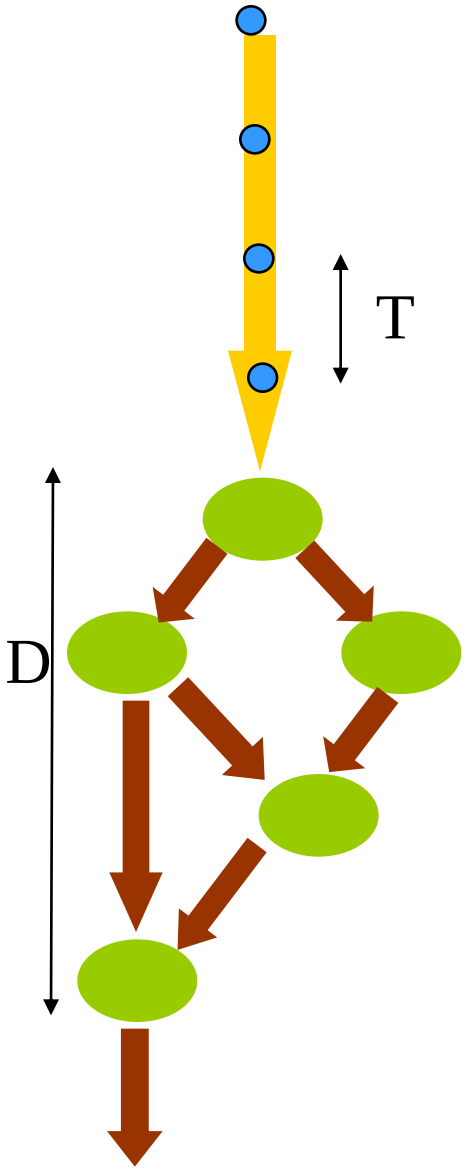
Machine model B always achieves smaller energy than a sequential machine.
Larger λ forces the processor to achieve the lowest energy at a higher speed.

Power Management in Multicores

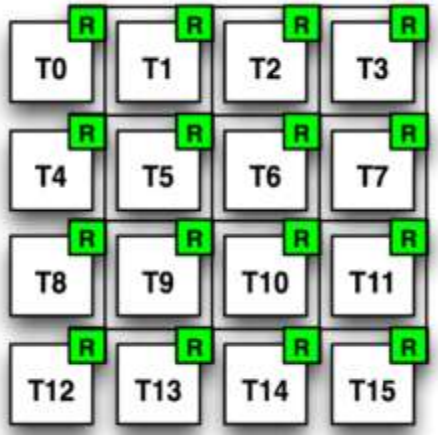
We will consider three possible task models:

1) The Amdahl model (perfect parallel sections)
2) Structured computation (streaming applications)
   - static mapping based on worst case execution
   - DVS based on statistical properties and dynamic slack reclamation
3) Computations with unknown structures

Mapping streaming applications to CMPs

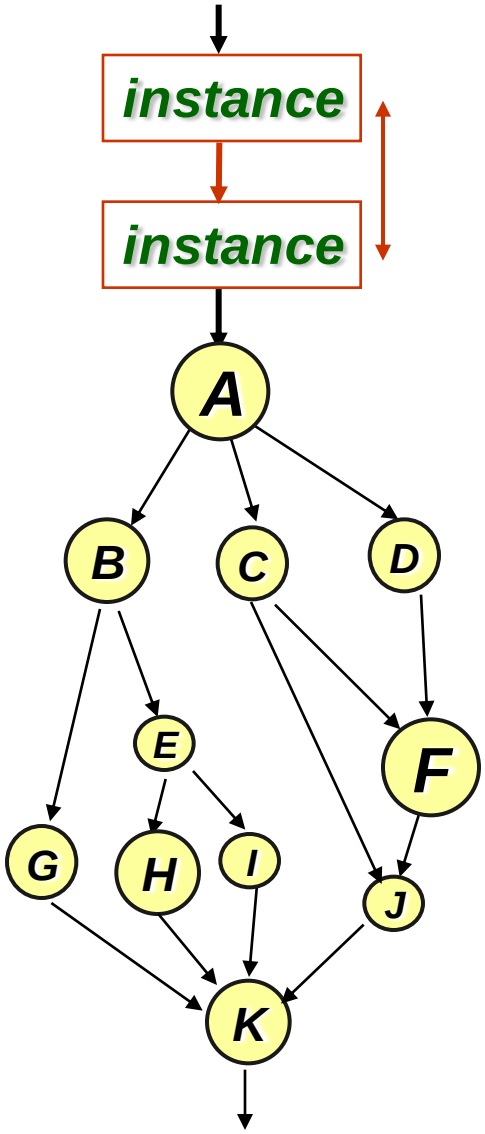


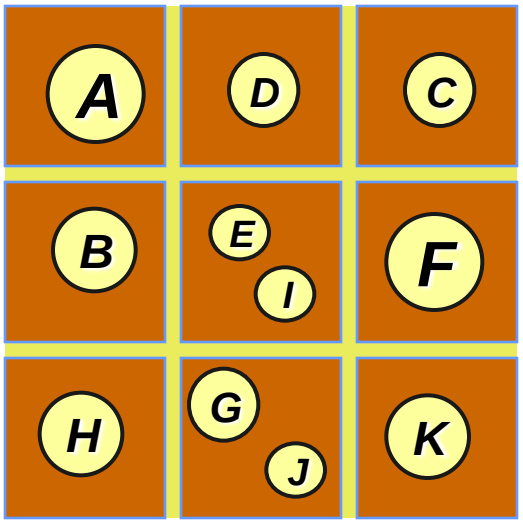
- Streaming applications are prevalent
  - Audio, video, real-time tasks, cognitive applications



- Constrains:
  - Inter-arrival time (T)
  - End-to-end delay (D)
- Power aware mapping to CMPs
  - Determine the number of cores to use
  - Determine speeds
  - Account for communication

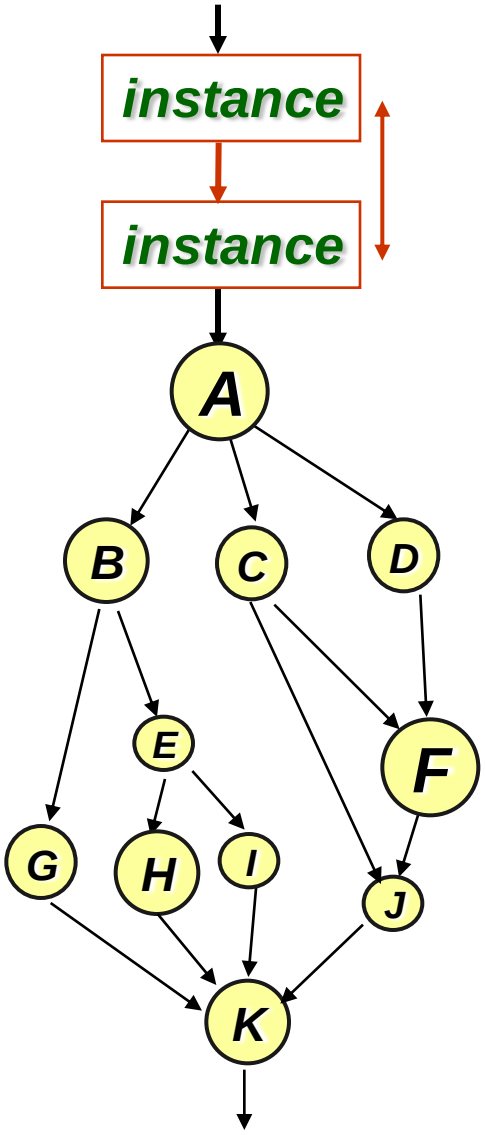
Mapping a task graph onto a CMP

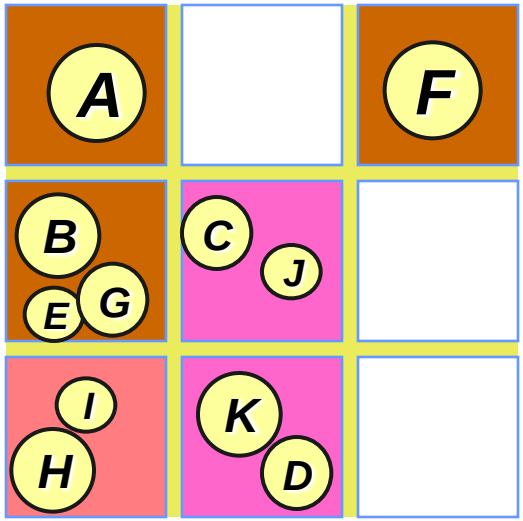




- Timing constraints are conventionally satisfied through load balanced mapping
- The mapping problem is NP hard even without considering energy
- NP hard when considering energy
  - Minimize energy consumption
  - Maximize performance for a given energy budget

Turn OFF some cores and use DVFS





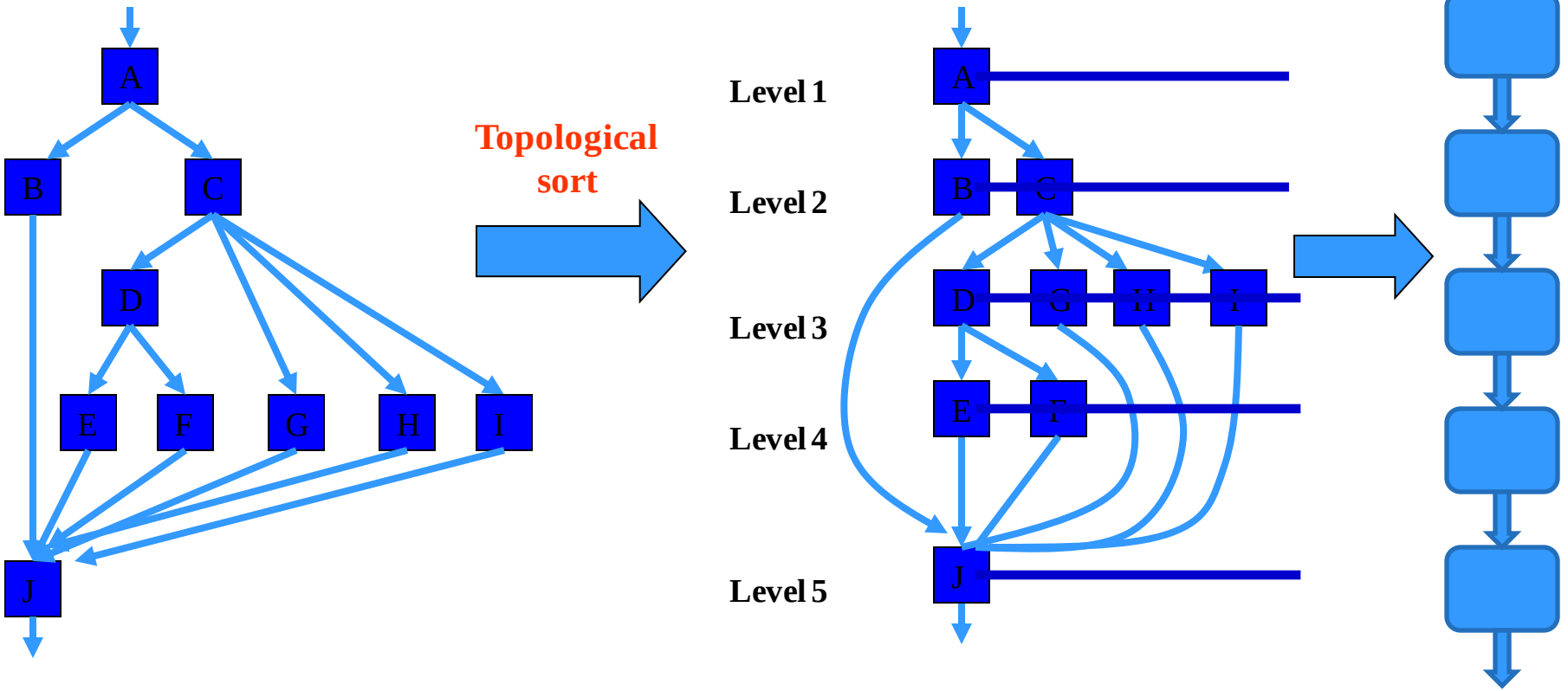
Maximum speed/voltage (f_{max})

Medium speed/voltage

Minimum speed/voltage (f_{min})

Core turned OFF

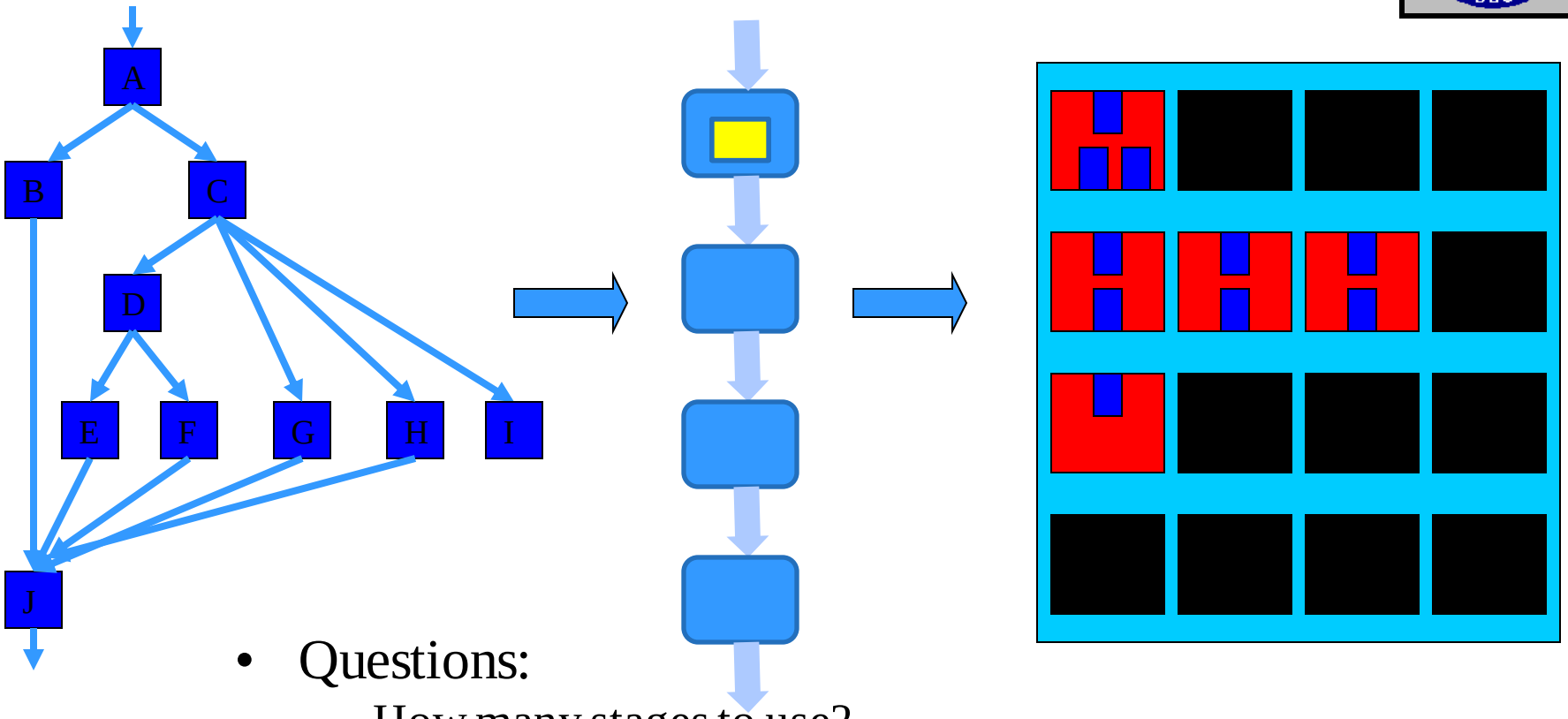
Scheduling General Task Graphs*



- Treating each level as a task in linear task graphs, we can use the linear pipeline schedule as a heuristics for general task graphs

*) ACM Tran Comp. Systems 2007

Scheduling General Task Graphs



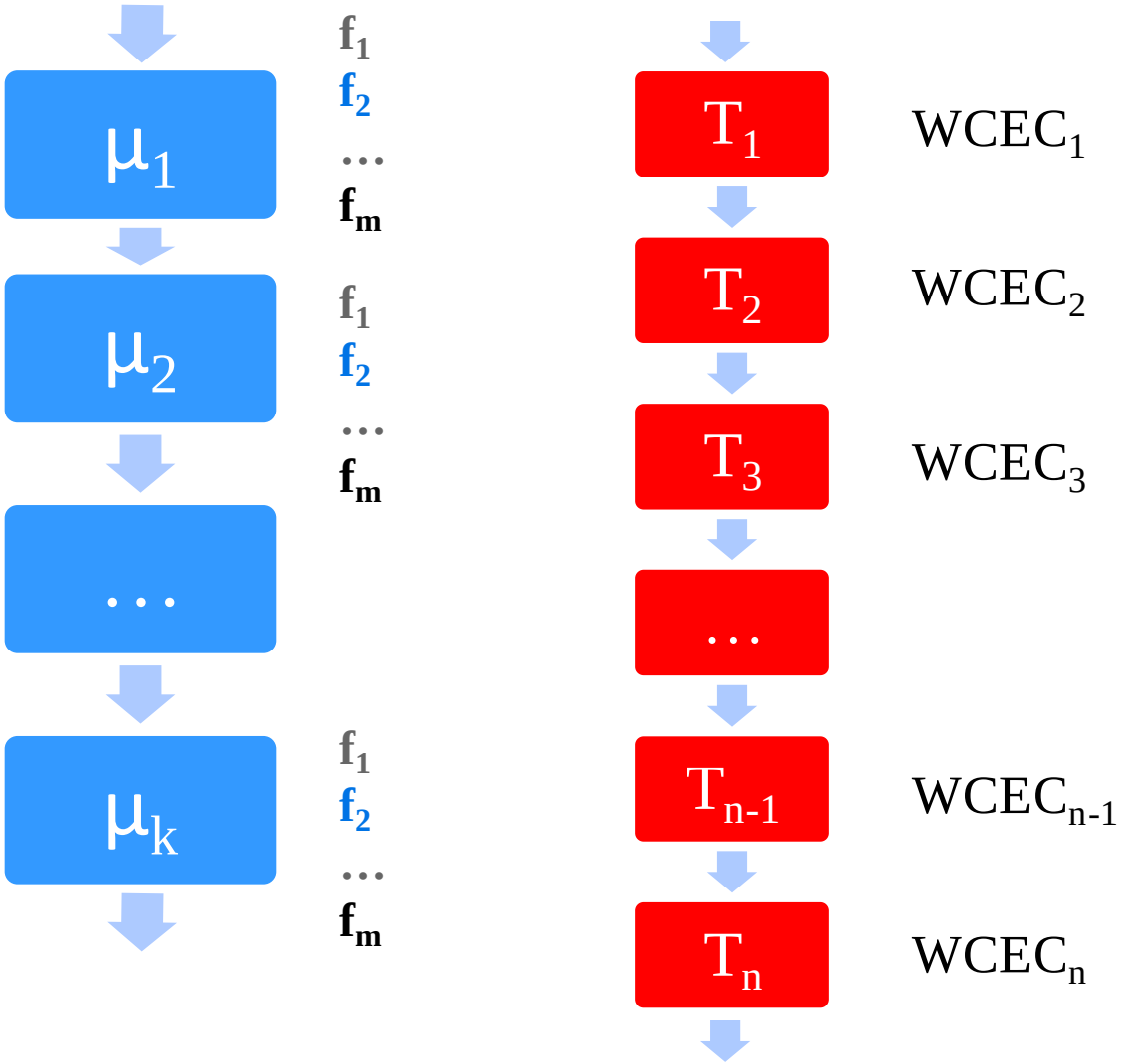
- Questions:
  - How many stages to use?
  - Allotted time for each stage?
  - For each stage, how many processors to use?
  - For each stage, what's the mapping?
  - For each stage, what's the speed for each task?

A dynamic programming algorithm*

Periodic Job

Inter-arrival time: T

Deadline: D > T

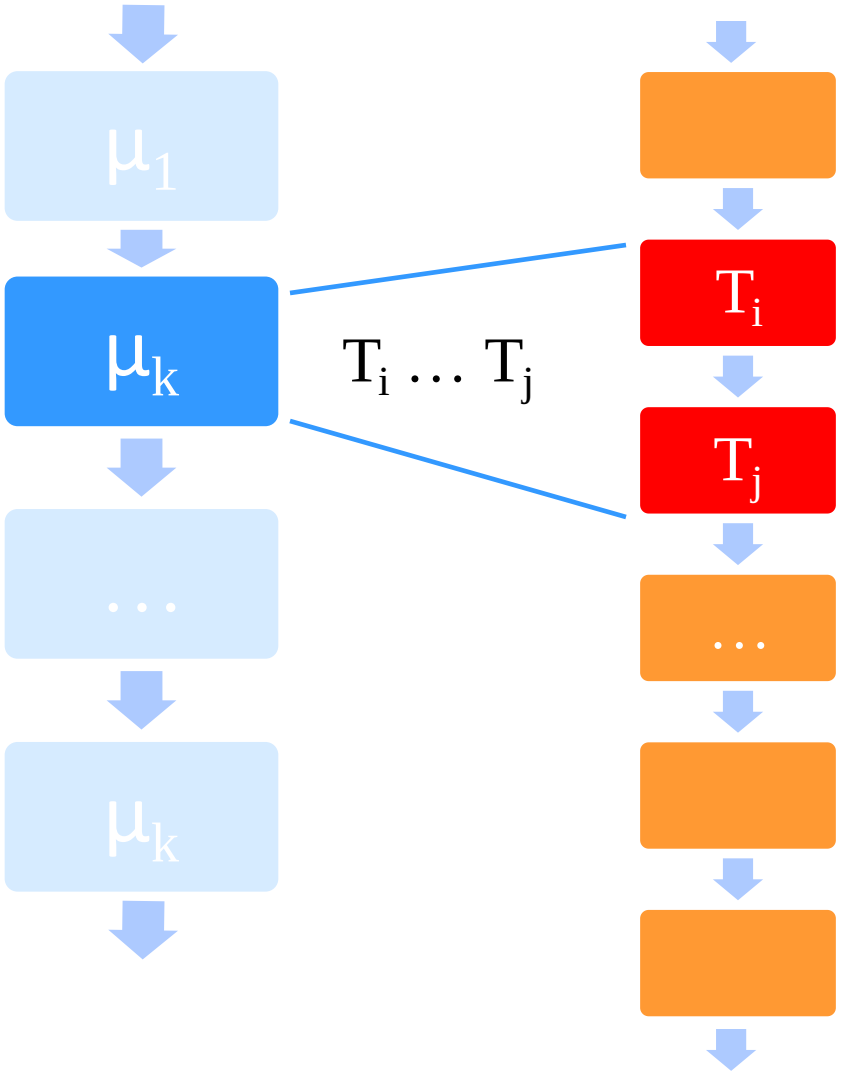


*) ACM Tran Comp. Systems 2007

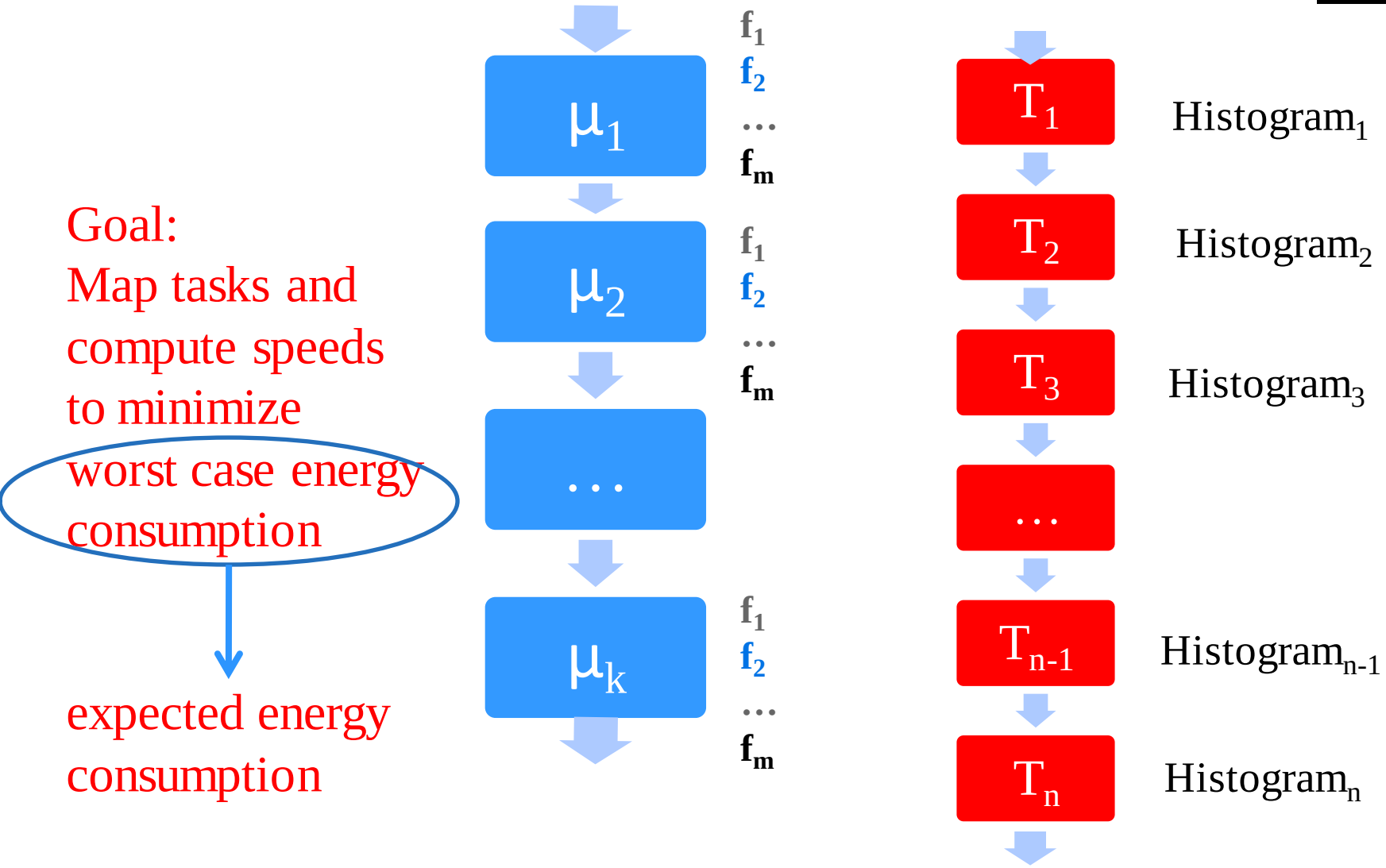
A dynamic programming algorithm

Compute energy and delay when T_i , … T_j are mapped to one processor

Use recursion to propagate this information



May also apply statistical analysis*



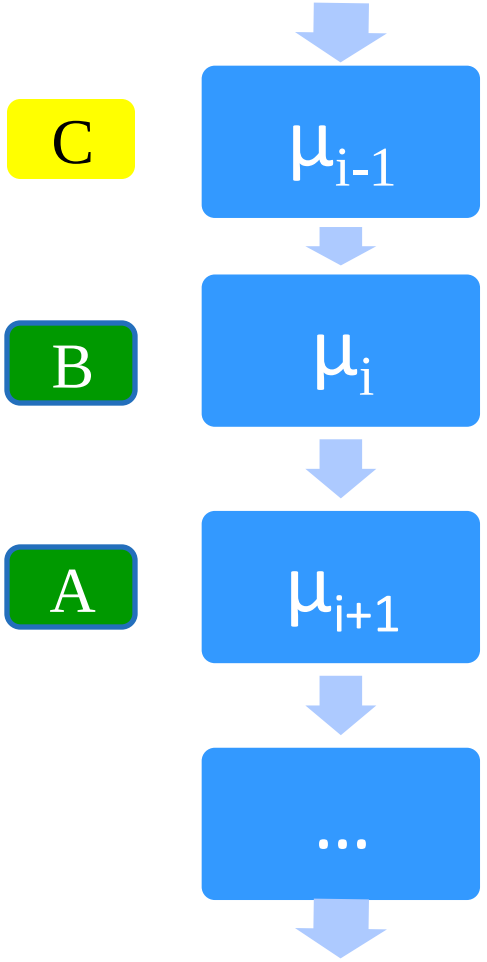
*) IEEE Symposium on Industrial Embedded Systems (SIES) 2009

Dynamic slack (idle time) reclamation

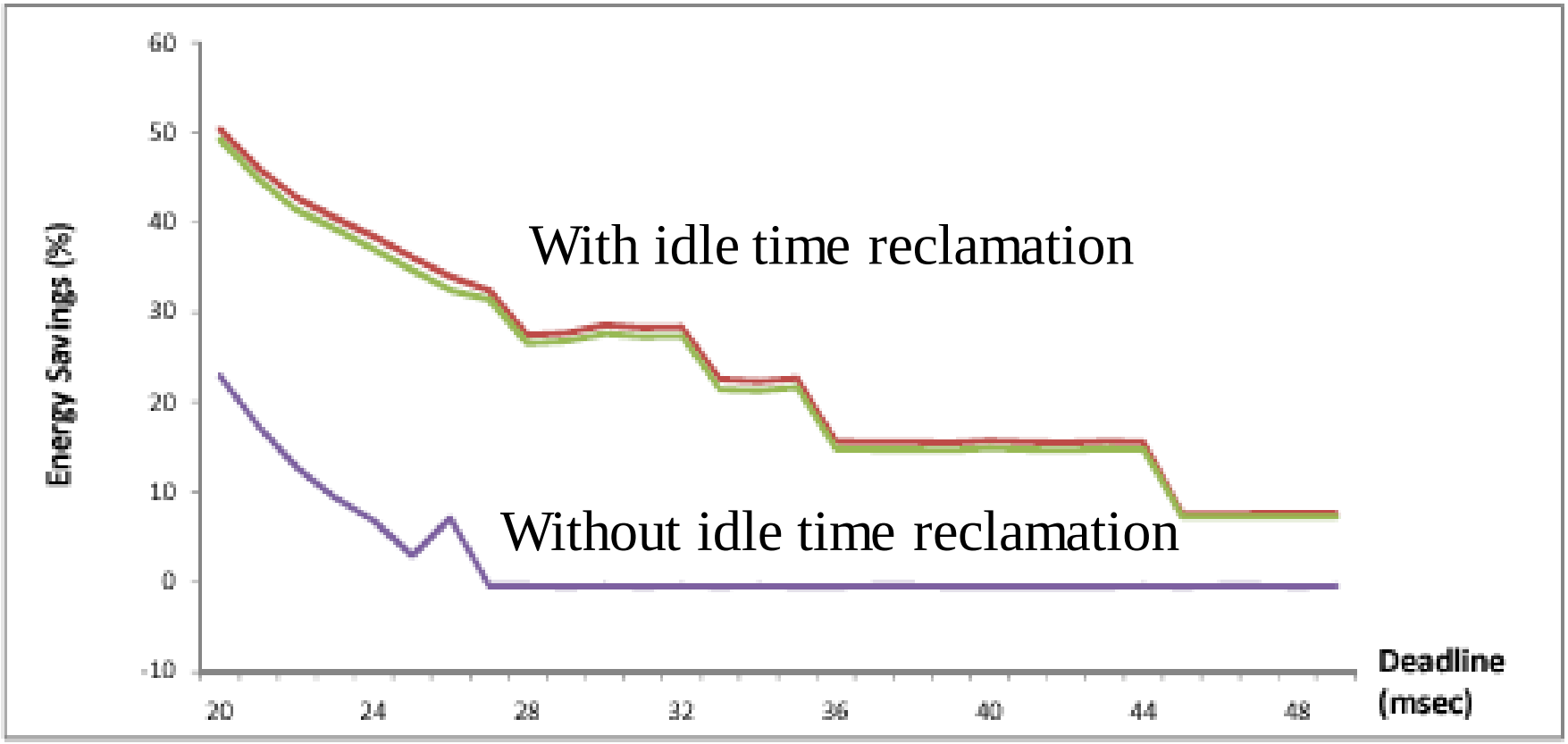
It is not always possible to reclaim the idle time to slow down the processing in a pipeline.

Example:

- If B finishes early, we cannot use the idle time
- unless C finishes early and moves into μ_i (can slow down the computation of C).



Effect of Cross-Stage Idle Time Reclamation



Experiments when initial mapping is done using the worst case execution time

Power Management in Multicores

We will consider three possible task models:

1) The Amdahl model (perfect parallel sections)
2) Structured computation (streaming applications)
   - static mapping based on worst case execution
   - DVS based on statistical properties and dynamic slack reclamation
3) Computations with unknown/unspecified structures

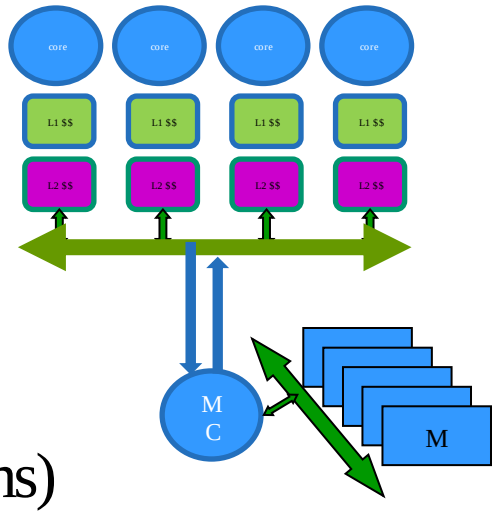
DVS using Machine Learning*

Characterize the execution state of a core by parameters such as

- Rate of instruction execution (IPC)
- # of memory accesses per instruction
- Average memory access time (depends on other threads)

Learn for each state of a core

- The frequency that optimizes your goal (example goal is energy consumption)



During execution, periodically (every 50μs -10ms)

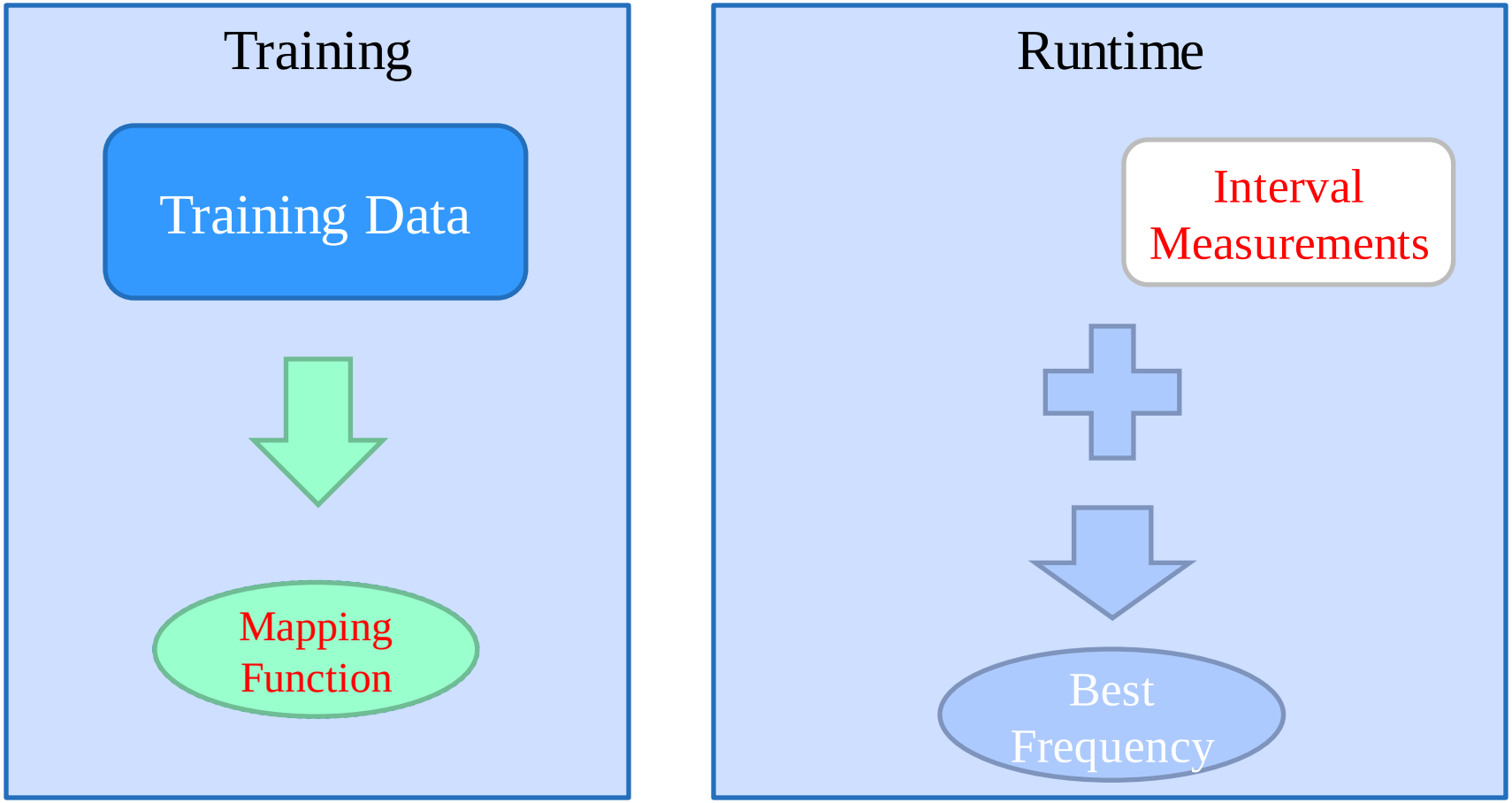
Estimate the current state (through run-time measurements)

Assume that the future is a continuation of the present

Set the frequency to the best recorded during training

*) Computer Frontiers 2010

Machine Learning Approach



For training, we use representative workloads and set the frequencies randomly in each interval to learn as much as possible.

What defines the state of a core? (Feature Selection)

Start with Raw Measurements
Cycles
L1 Access
L1 Miss
Average Stall
Instructions
User Instructions
…

Generate Inverses
Cycles
$Cycles^{-1}$
L1 Access
$L1\ Access^{-1}$
L1 Miss
$L1\ Miss^{-1}$
…

First Order Metrics

Multiply Together
Cycles * L1 Access
Cycles * $L1\ Access^{-1}$
$Cycles^{-1}$ * L1 Access
$Cycles^{-1}$ * $L1\ Access^{-1}$
L1 Access * L1 Miss
L1 Access * $L1\ Miss^{-1}$
…

Second Order Metrics

Feature Selection: Correlation Study

	Correlation	Second Ord	Second Order Metrics	on
m1	0.1	Cycles * L1 A	Cycles * L1 Access	
m2	0.3	Cycles^{-1} * L1	Cycles * L1 Access^{-1}	
m3	-0.2	L1 Access *	Cycles^{-1} * L1 Access	
	0.05	Cycles * L1 A	Cycles^{-1} * L1 Access^{-1}	
	0.02	Cycles^{-1} * L1	L1 Access * L1 Miss	
	0.15	L1 Access *	L1 Access * L1 Miss^{-1}	

Goal Metric
Energy per User Instruction

The Mapping Function

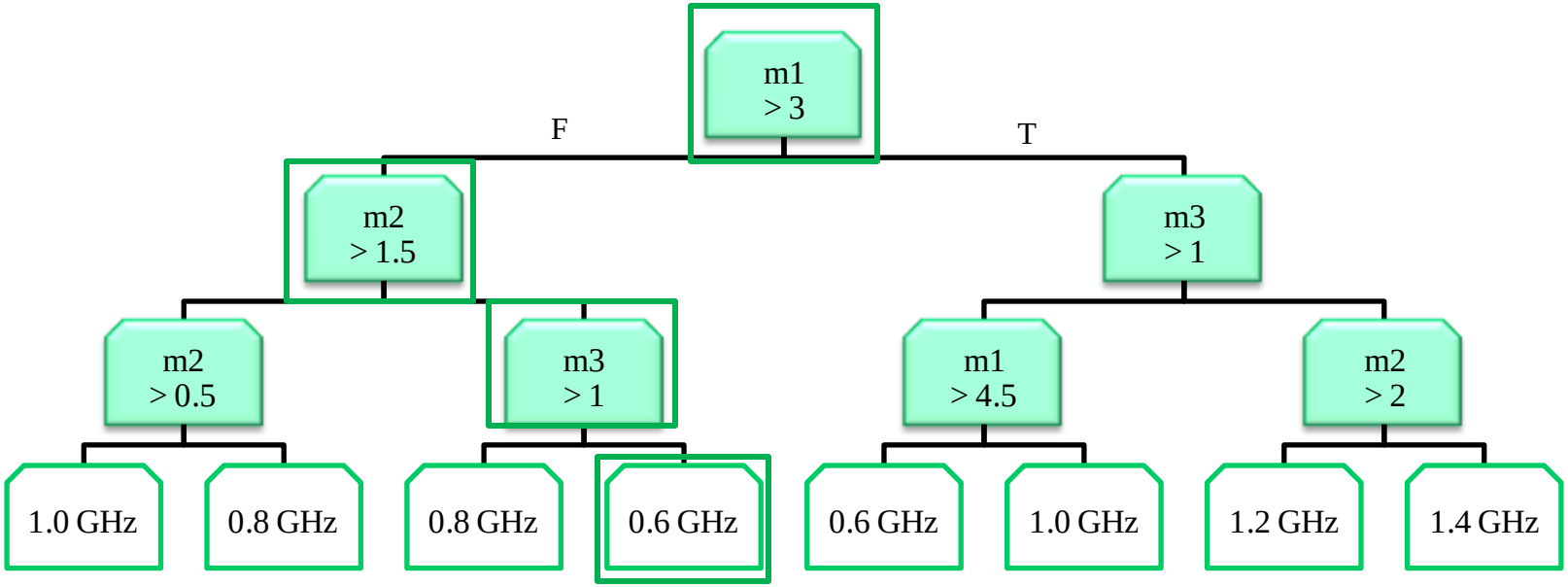
- The mapping function can be expressed as a table
- Each table entry represents a unique set of measurements
  - Tells us which frequency to choose

(m1,m2,m3)	Freq (GHz)
(2.1,3.5,1.8)	0.6
(4.0,1.0,4.0)	1.2

- Tow problems with the table
  - Too large (depends on discretization of the measurements)
  - Has empty entries (situation not encountered during training)
- Transform the table into a decision tree

Decision Tree: Example

ex: (m1,m2,m3) = (2.1,3.5,1.8)

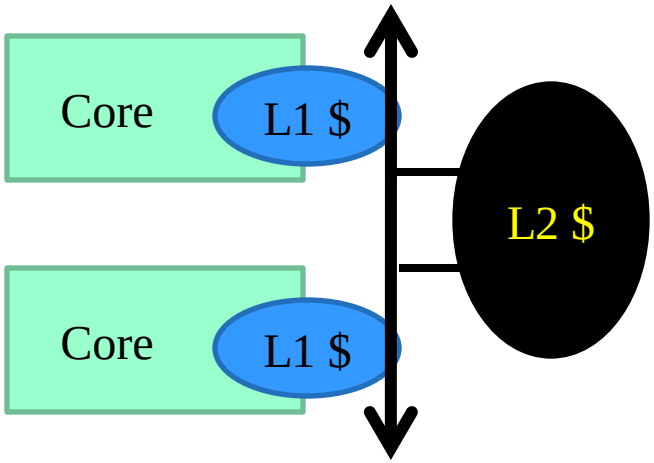


- Much smaller
- No blank entries

Experimental Validation

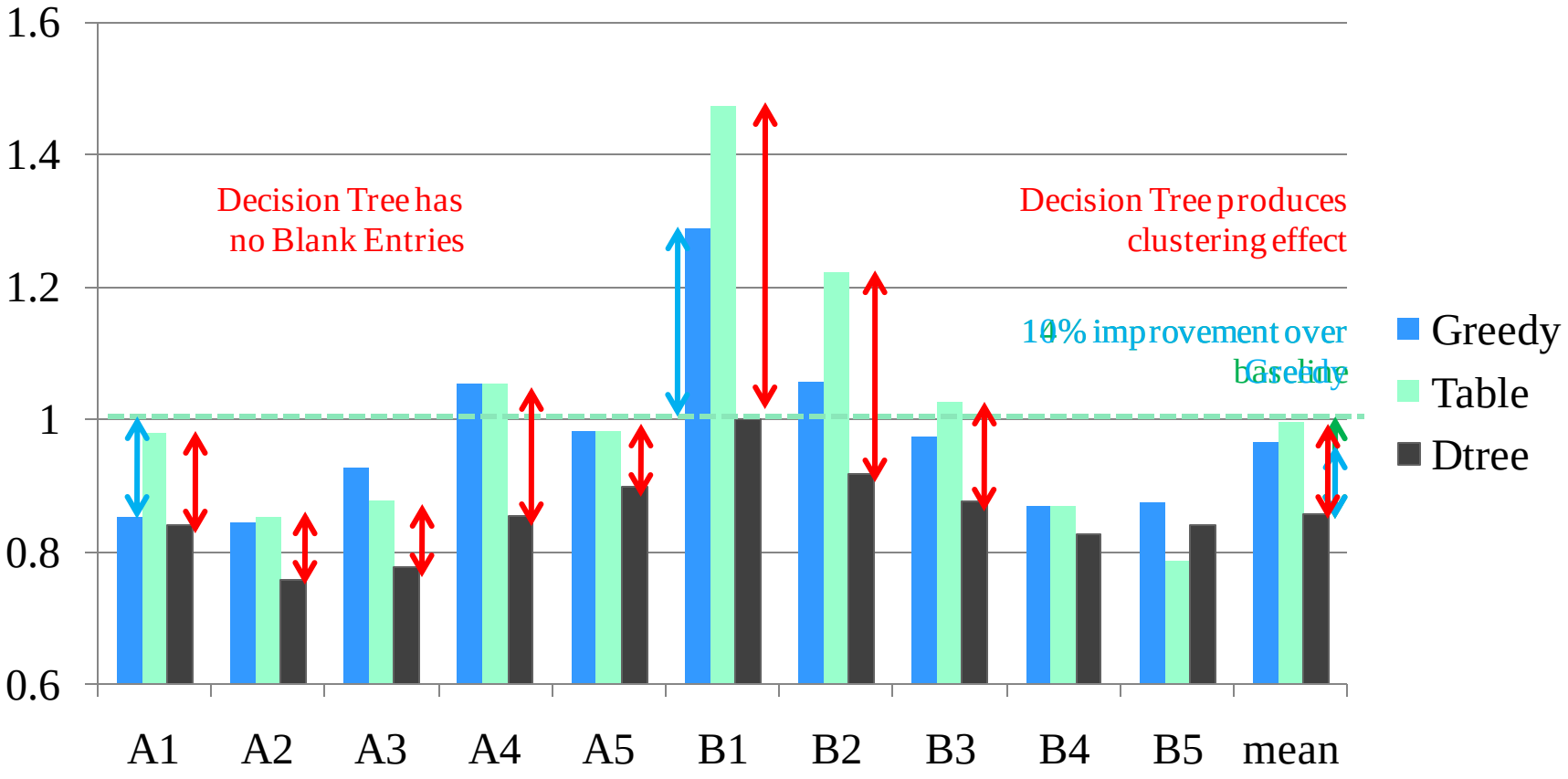
- Simics running Ubuntu
- Sample 2s execution

- Simulation parameters:
  - 16 in-order cores



- Power Parameters
  - 5 VF settings
  - 50µs to 1 ms intervals
  - Power =
    - Dynamic = αf^3
    - Static = βf
    - Background = γ
- Policies
  - Table
  - Decision tree
  - Greedy (HPCA '09)

Energy per (user-instruction)^2



Conclusion

- Scheduling processor speeds for multiple cores is challenging!
- Usually has to resort to heuristics to do the initial static scheduling in realistic settings
- Dynamic slack reclamation is not trivial due to computation dependences
- Machine learning techniques deal with a complex problem using statistical methods, rather than heuristics