Sharif University of Technology
Department of Computer Science and Engineering

Lec. 8:
Evaluation and optimizations

M. Ansari

Fall 2021

According to Peter Marwedel's Lectures

Validation and Evaluation

Definition: *Validation* is the process of checking whether or not a certain (possibly partial) design is appropriate for its purpose, meets all constraints and will perform as expected (yes/no decision).

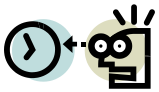
Definition: Validation with mathematical rigor is called *(formal) verification*.

Definition: *Evaluation* is the process of computing quantitative information of some key characteristics of a certain (possibly partial) design.

How to evaluate designs according to multiple criteria?

Many different criteria are relevant for evaluating designs:

- Average & worst case delay
- power/energy consumption
- thermal behavior
- reliability, safety, security
- cost, size
- weight
- EMC characteristics
- radiation hardness, environmental friendliness, ..





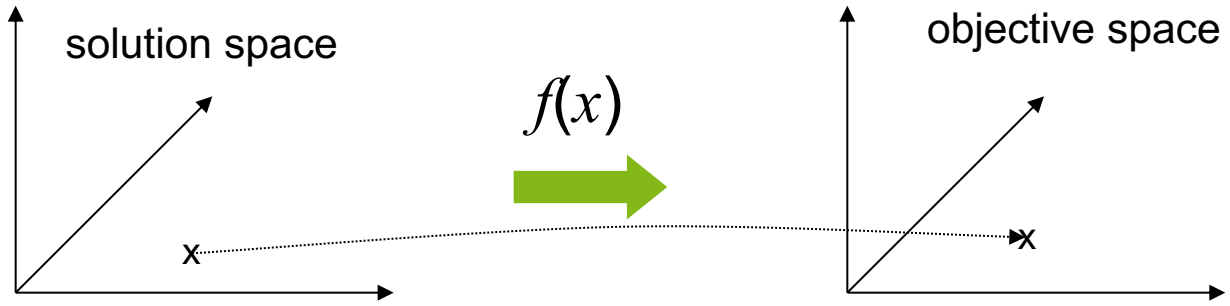




How to compare different designs?
(Some designs are “better” than others)

Definitions

- Let X: m-dimensional **solution space** for the design problem. Example: dimensions correspond to # of processors, size of memories, type and width of busses etc.
- Let F: n-dimensional **objective space** for the design problem. Example: dimensions correspond to average and worst case delay, power/energy consumption, size, weight, reliability, …
- Let $f(x)=(f_1(x),\ldots,f_n(x))$ where $x \in X$ be an **objective function.** We assume that we are using $f(x)$ for evaluating designs.



Pareto points

- We assume that, for each objective, an order $<$ and the corresponding order $\leq$ are defined.
- **Definition**:
  Vector $u=(u_1,\ldots,u_n)\in F$ **dominates** vector $v=(v_1,\ldots,v_n)\in F$
  $\Leftrightarrow$
  u is "better" than v with respect to one objective and not worse than v with respect to all other objectives:

$$\forall i \in \{1,\ldots,n\} : u_i \leq v_i \wedge$$
$$\exists i \in \{1,\ldots,n\} : u_i < v_i$$

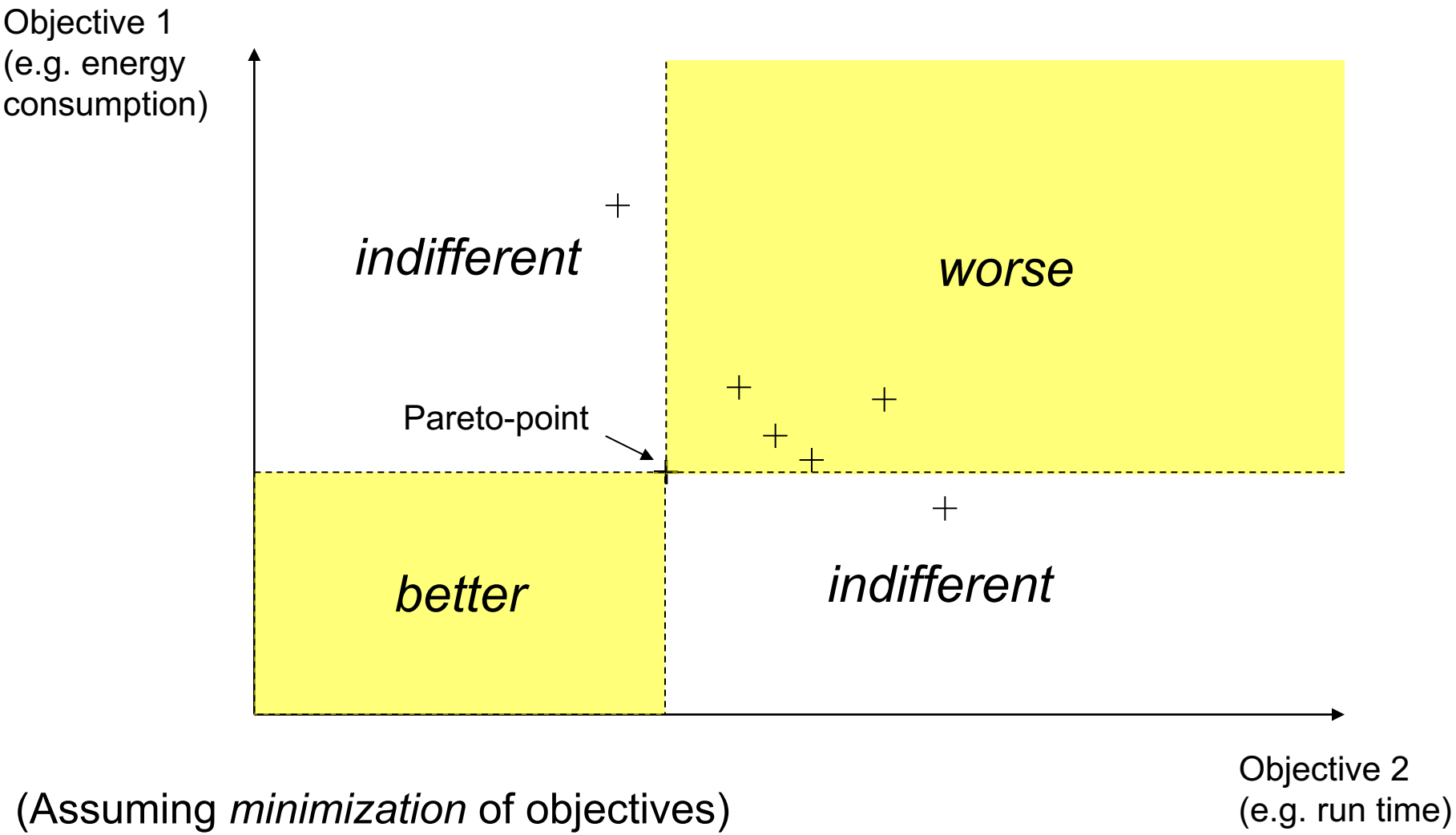
- **Definition:**
  Vector $u\in F$ is **indifferent** with respect to vector $v\in F$
  $\Leftrightarrow$ neither u dominates v nor v dominates u

Pareto points

- A solution $x \in X$ is called **Pareto-optimal** with respect to X $\Leftrightarrow$ there is no solution $y \in X$ such that $u=f(x)$ is dominated by $v=f(y)$. x is a **Pareto point**.
- **Definition**: Let $S \subseteq F$ be a subset of solutions. $v \in F$ is called a **non-dominated solution** with respect to S $\Leftrightarrow$ v is not dominated by any element $\in S$.
- v is called **Pareto-optimal** $\Leftrightarrow$ v is non-dominated with respect to all solutions F.
- A **Pareto-set** is the set of all Pareto-optimal solutions

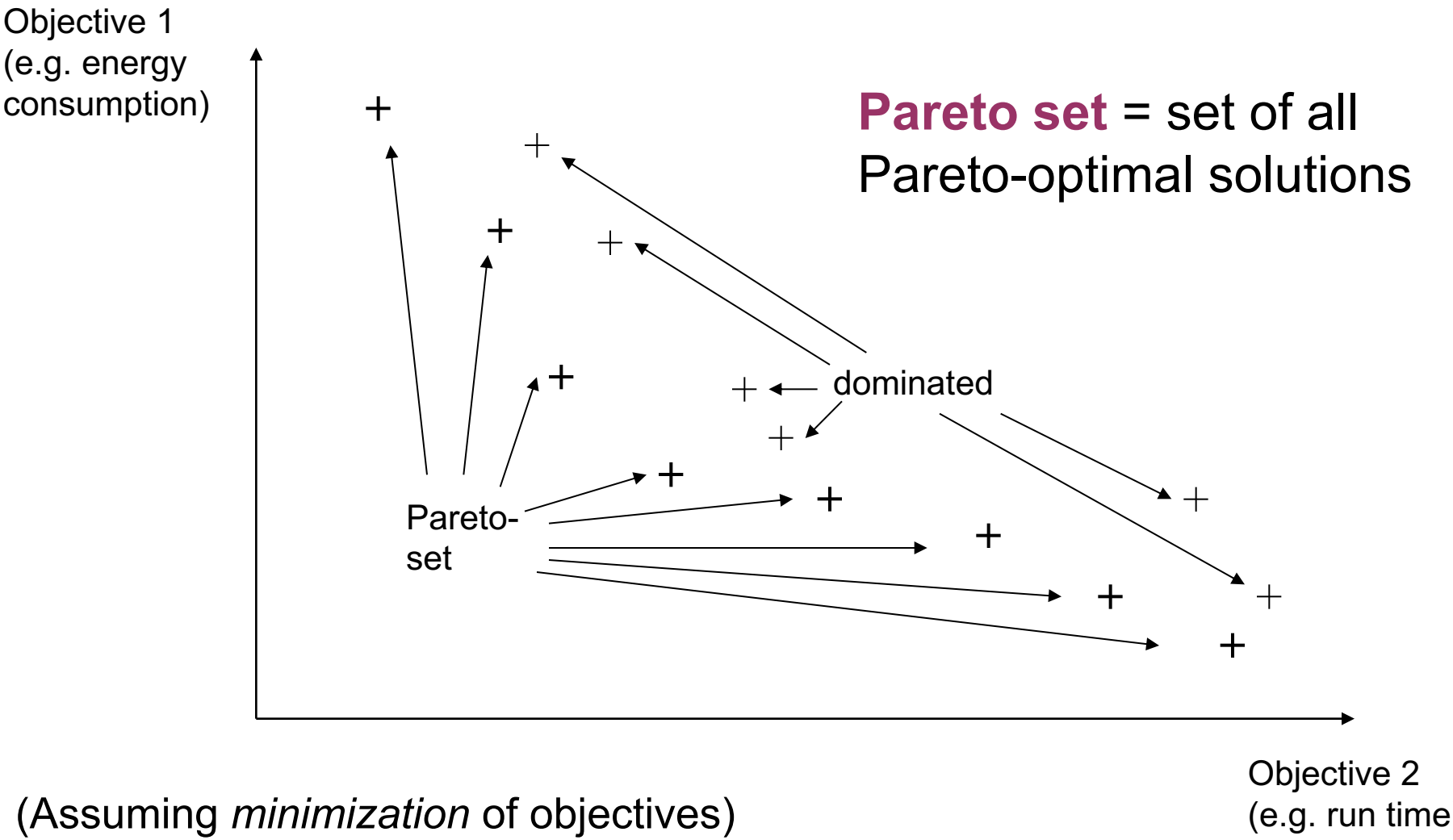
Pareto-sets define a **Pareto-front**
(boundary of dominated subspace)

Pareto Point



(Assuming *minimization* of objectives)

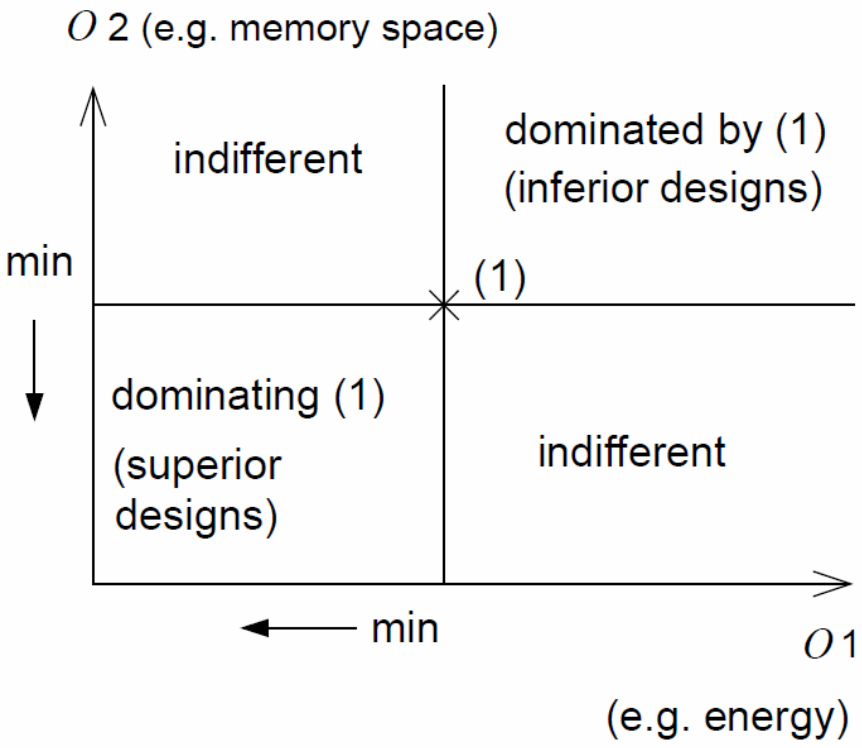
Pareto Set



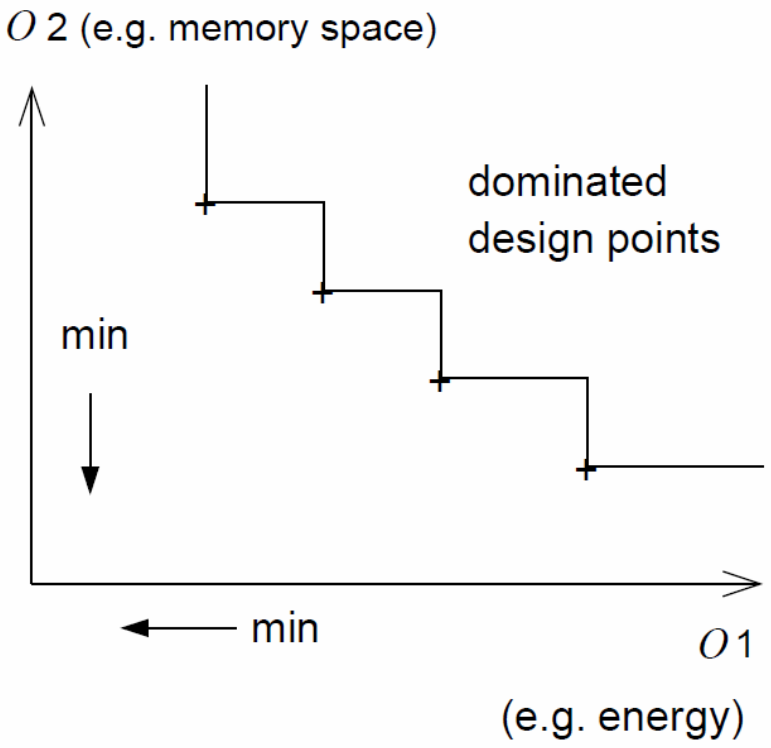
(Assuming *minimization* of objectives)

One more time …

Pareto point



Pareto front



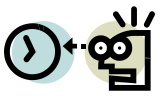
Design space evaluation

Design space evaluation (DSE) based on Pareto-points is the process of finding and returning a set of Pareto-optimal designs to the user, enabling the user to select the most appropriate design.

How to evaluate designs according to multiple criteria?

Many different criteria are relevant for evaluating designs:

- Average & worst case delay
- power/energy consumption
- thermal behavior
- reliability, safety, security
- cost, size
- weight
- EMC characteristics
- radiation hardness, environmental friendliness, ..





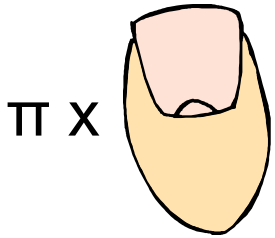




How to compare different designs?
(Some designs are “better” than others)

Average delays (execution times)

- **Estimated** average execution times **:**
  Difficult to generate sufficiently precise estimates;
  Balance between run-time and precision

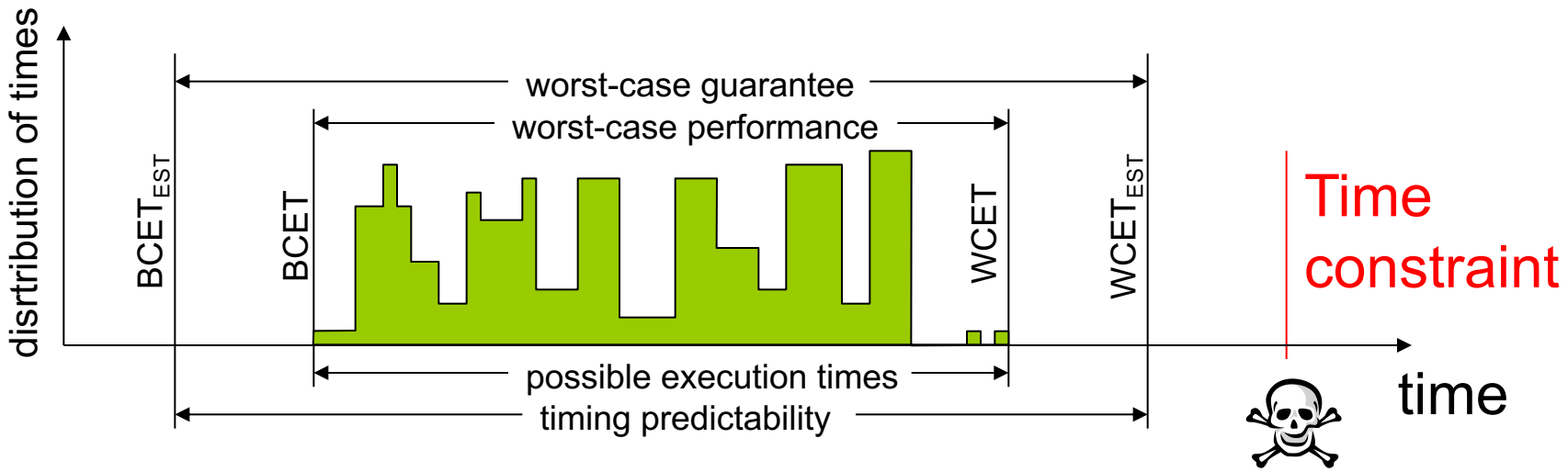


- **Accurate** average execution times**:**
  As precise as the input data is.

We need to compute **average** and **worst case** execution times

Worst case execution time (1)

Definition of worst case execution time:



WCET$_{EST}$ must be

1. safe (i.e. ≥ WCET) and
2. tight (WCET$_{EST}$-WCET≪WCET$_{EST}$)

Worst case execution times (2)

Complexity:

- in the general case: undecidable if a bound exists.
- for restricted programs: simple for “old“ architectures, very complex for new architectures with pipelines, caches, interrupts, virtual memory, etc.

Approaches:

- for hardware: requires detailed timing behavior
- for software: requires availability of machine programs; complex analysis (see, e.g., www.absint.de)

Summary

Evaluation and Validation

- In general, multiple objectives
- Pareto optimality
- Design space evaluation (DSE)
- Execution time analysis
  - Trade-off between speed and accuracy
  - Computation of worst case execution times

Standard Optimization: Integer linear programming models

Ingredients:

- Cost function
- Constraints

} Involving linear expressions of integer variables from a set X

Cost function

$$C = \sum_{x_i \in X} a_i x_i \text{ with } a_i \in \mathbb{R}, x_i \in \mathbb{N} \quad (1)$$

Constraints: $\forall j \in J : \sum_{x_i \in X} b_{i,j}\, x_i \geq c_j$ with $b_{i,j}, c_j \in \mathbb{R}$ (2)

Def.: The problem of minimizing (1) subject to the constraints (2) is called an **integer linear programming (ILP) problem**.

If all x_i are constrained to be either 0 or 1, the ILP problem is said to be a **0/1 integer linear programming problem**.

Example

$$C = 5x_1 + 6x_2 + 4x_3$$

$$x_1 + x_2 + x_3 \geq 2$$

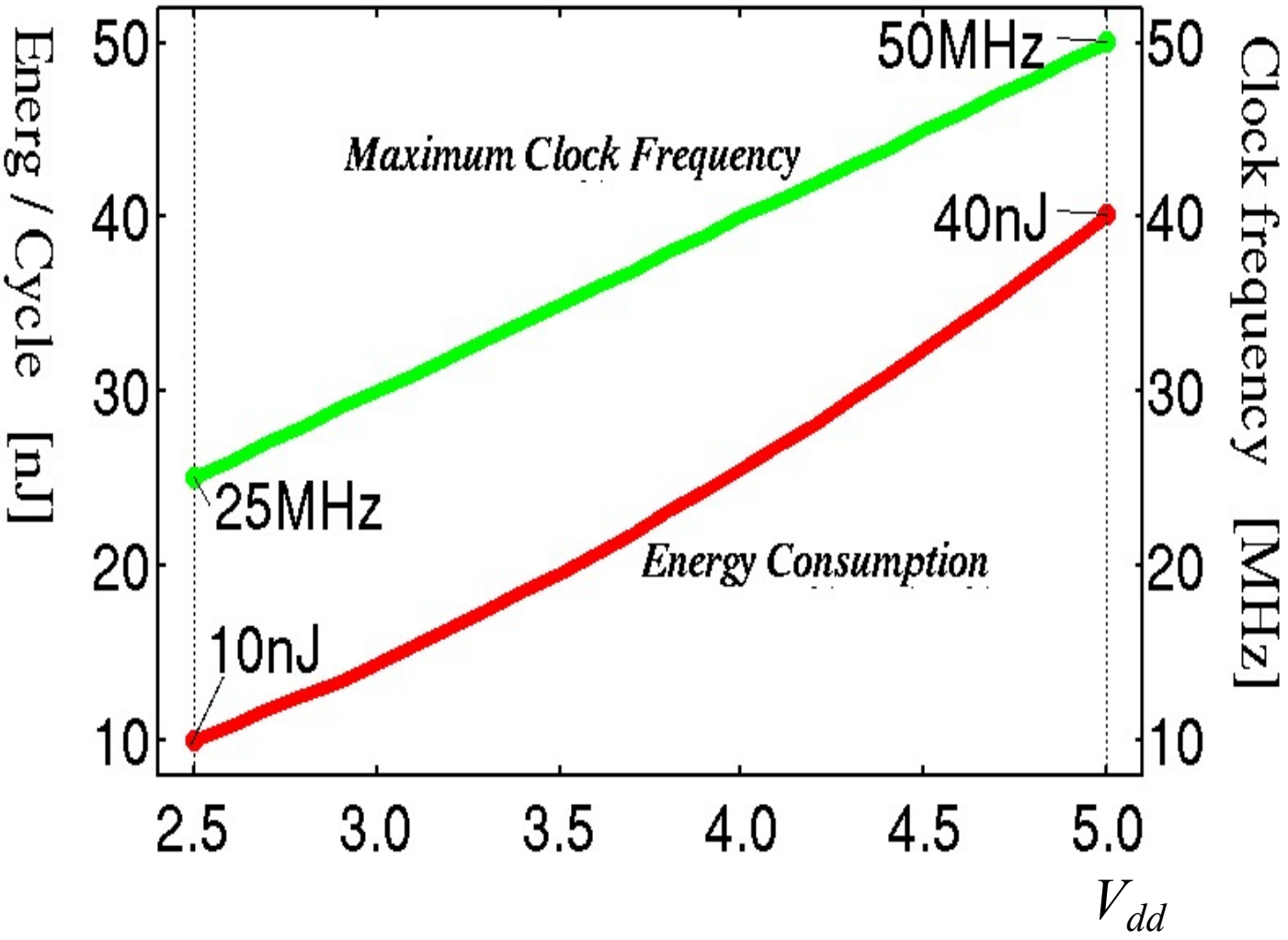
$$x_1, x_2, x_3 \in \{0,1\}$$

x_1	x_2	x_3	C	
0	1	1	10	
1	0	1	9	← Optimal
1	1	0	11	
1	1	1	15	

Remarks on integer programming

- Maximizing the cost function: just set C‘=-C
- Integer programming is NP-complete.
- Running times depend exponentially on problem size, but problems of >1000 vars solvable with good solver (depending on the size and structure of the problem)
- The case of $x_i \in \mathbb{R}$ is called *linear programming* (LP). Polynomial complexity, but most algorithms are exponential, in practice still faster than for ILP problems.
- The case of some $x_i \in \mathbb{R}$ and some $x_i \in \mathbb{N}$ is called *mixed integer-linear programming.*
- ILP/LP models good starting point for modeling, even if heuristics are used in the end.
- Solvers: lp_solve (public), CPLEX (commercial), …

Voltage Scaling and Power Management
Dynamic Voltage Scaling



Recap from chapter 5: Fundamentals of dynamic voltage scaling (DVS)

Power consumption of CMOS circuits (ignoring leakage):

$P = \alpha\ C_L V_{dd}^2\ f$ with

α : switching activity

C_L : load capacitance

V_{dd} : supply voltage

f : clock frequency

Delay for CMOS circuits:

$$\tau = k\, C_L \frac{V_{dd}}{(V_{dd} - V_t)^2}$$ with

V_t : threshhold voltage

(V_t substancially < than V_{dd})

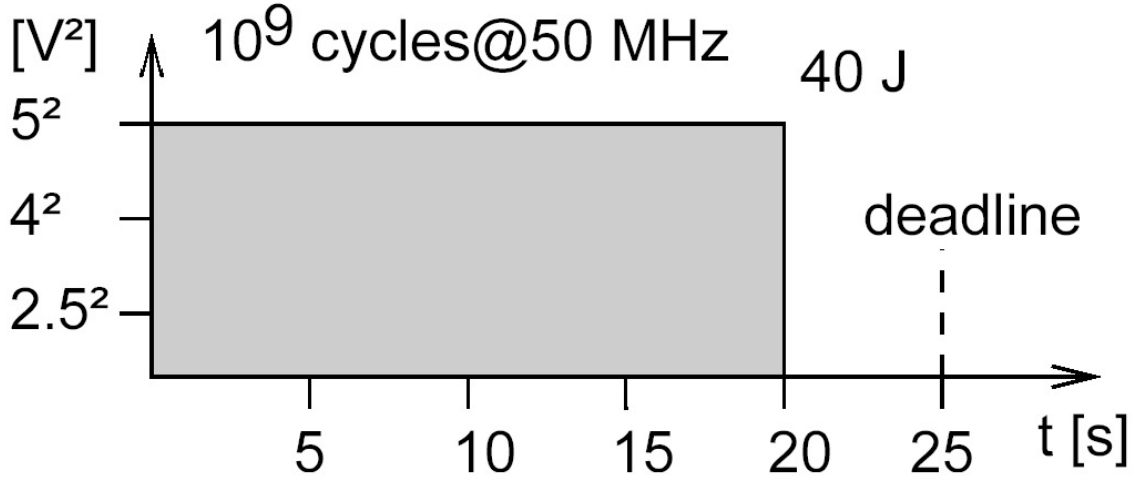
☞ Decreasing V_{dd} reduces P quadratically,
while the run-time of algorithms is only linearly increased
(ignoring the effects of the memory system).

Example: Processor with 3 voltages
Case a): Complete task ASAP

Task that needs to execute 10^9 cycles within 25 seconds.

V_{dd} [V]	5.0	4.0	2.5
Energy per cycle [nJ]	40	25	10
f_{max} [MHz]	50	40	25
cycle time [ns]	20	25	40

a)

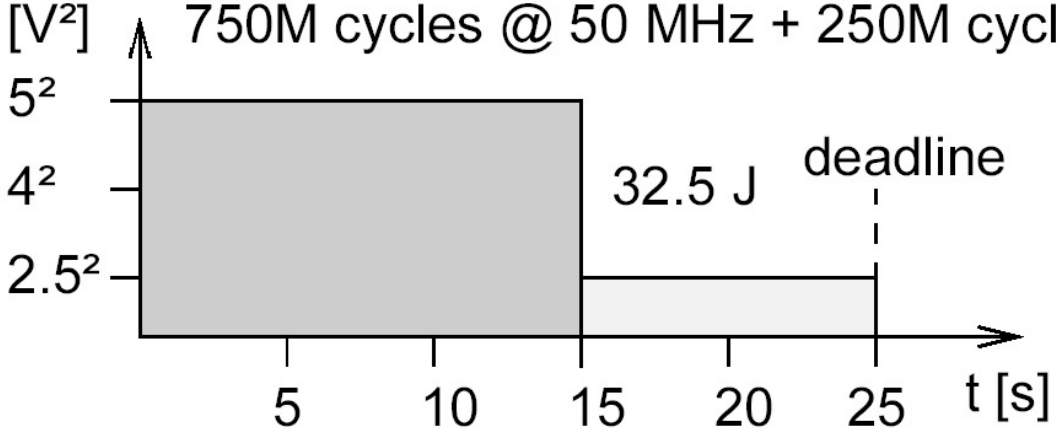


$E_a = 10^9 \times 40 \times 10^{-9}$ [J]
$= 40$ [J]

Case b): Two voltages

V_{dd} [V]	5.0	4.0	2.5
Energy per cycle [nJ]	40	25	10
f_{max} [MHz]	50	40	25
cycle time [ns]	20	25	40

b) [V²] 750M cycles @ 50 MHz + 250M cycles @ 25

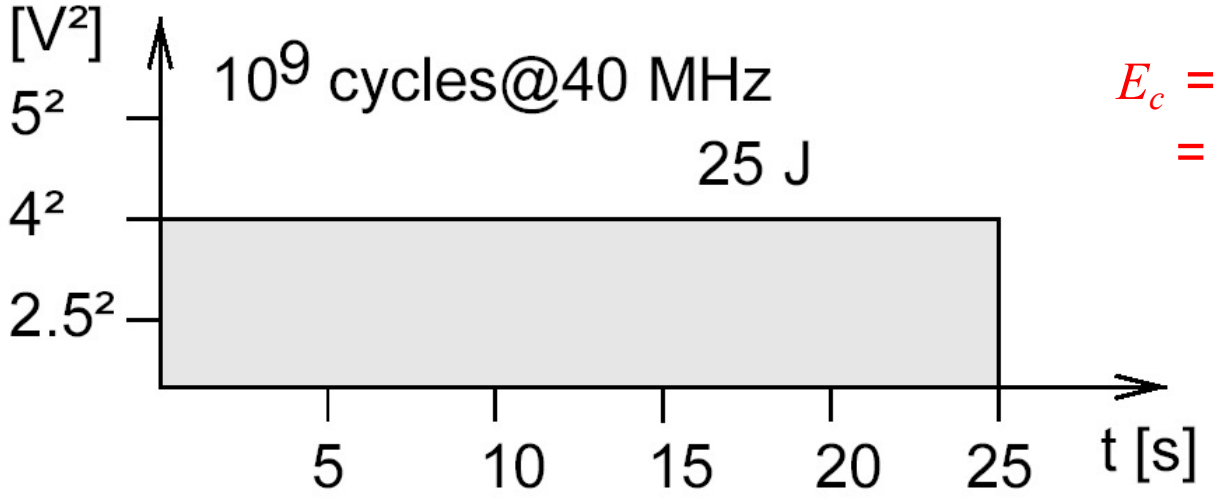


E_b= 750 10^6 x 40 10^{-9} +
250 10^6 x 10 10^{-9} [J]
= 32.5 [J]

Case c): Optimal voltage

V_{dd} [V]	5.0	4.0	2.5
Energy per cycle [nJ]	40	25	10
f_{max} [MHz]	50	40	25
cycle time [ns]	20	25	40

c)



E_c = 10^9 x 25 x 10^{-9} [J]
= 25 [J]

Observations

- A minimum energy consumption is achieved for the ideal supply voltage of 4 Volts.
- In the following: **variable voltage processor =** processor that allows **any** supply voltage up to a certain maximum.
- It is expensive to support truly variable voltages, and therefore, actual processors support only a few fixed voltages.

Generalisation

Lemma [Ishihara, Yasuura]:

- If a variable voltage processor completes a task before the deadline, the energy consumption can be reduced.
- If a processor uses a single supply voltage V and completes a task T just at its deadline, then V is the unique supply voltage which minimizes the energy consumption of T.
- If a processor can only use a number of discrete voltage levels, then the two voltages which (almost*) minimize the energy consumption are the two immediate neighbors of the ideal voltage V_{ideal} possible for a variable voltage processor.

  ---

  * Except for small amounts of energy resulting from the fact that cycle counts must be integers.

The case of multiple tasks: Assignment of optimum voltages to a set of tasks

N : the number of tasks

EC_j : the number of executed cycles of task j

L : the number of voltages of the target processor

V_i : the i^{th} voltage, with $1 \leq i \leq L$

F_i : the clock frequency for supply voltage V_i

T : the global deadline at which all tasks must have been completed

$X_{i,j}$: the number of clock cycles task j is executed at voltage V_i

SC_j : the average switching capacitance during the execution of task j (SC_j comprises the actual capacitance CL and the switching activity α)

Designing an ILP model

Simplifying assumptions of the ILP-model include the following:

- There is one target processor that can be operated at a limited number of discrete voltages.
- The time for voltage and frequency switches is negligible.
- The worst case number of cycles for each task are known.

Energy Minimization using an Integer Linear Programming Model

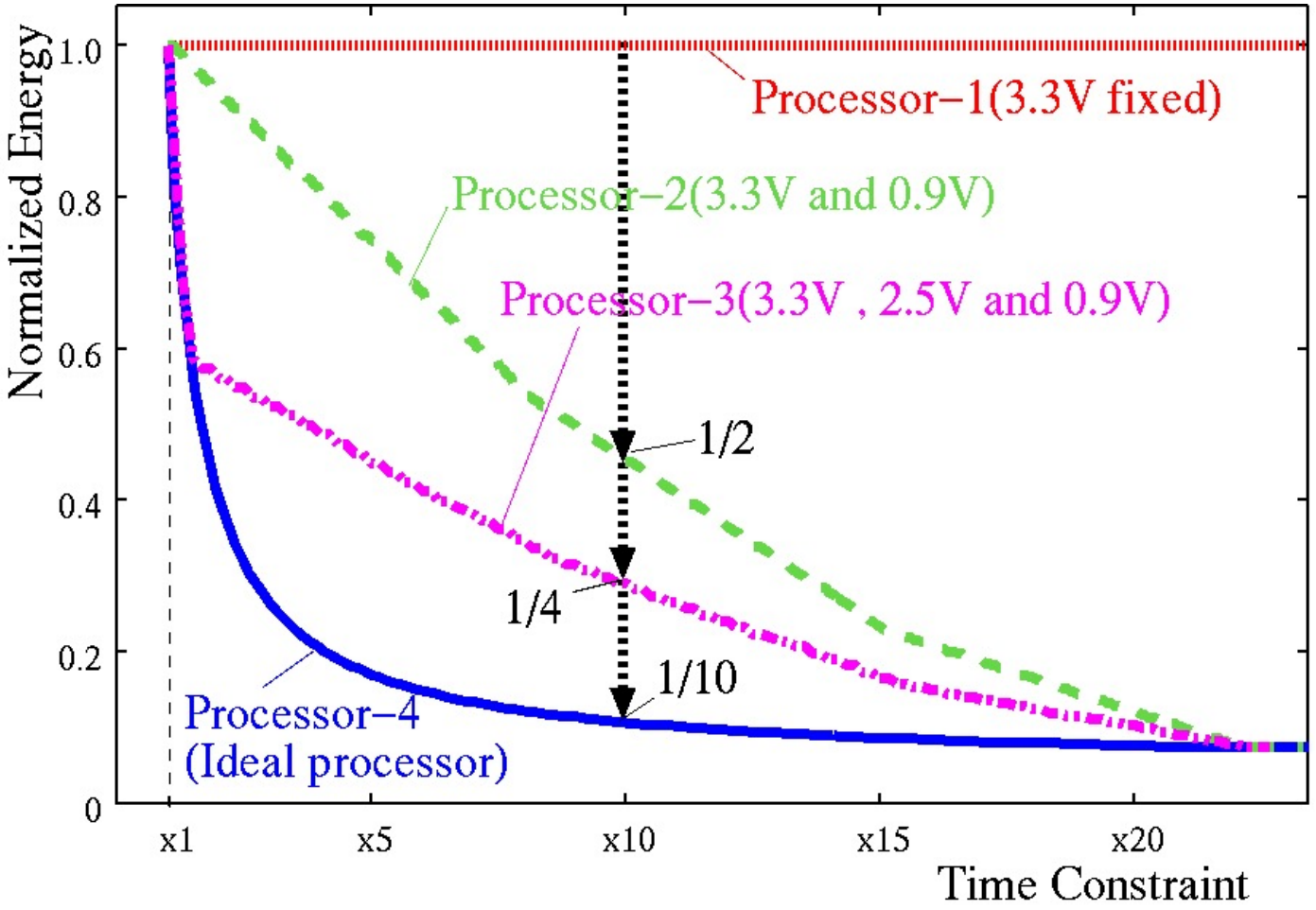
Minimize $$E = \sum_{j=1}^{N} \sum_{i=1}^{L} SC_j \cdot x_{i,j} \cdot V_i^2$$

subject to $$\forall j : \sum_{i=1}^{L} x_{i,j} = EC_j$$

and $$\sum_{i=1}^{L} \sum_{j=1}^{N} \frac{x_{i,j}}{F_i} \leq T$$

Experimental Results

3 tasks; EC_j=50 10^9; SC_1=50 pF; SC_2=100 pF; SC_3=150 pF



Dynamic power management (DPM)

Dynamic Power management tries to assign optimal power saving states.

- Questions: When to go to an power-saving state? Different, but typically complex models:
- Markov chains, renewal theory , ….

Task-level concurrency management

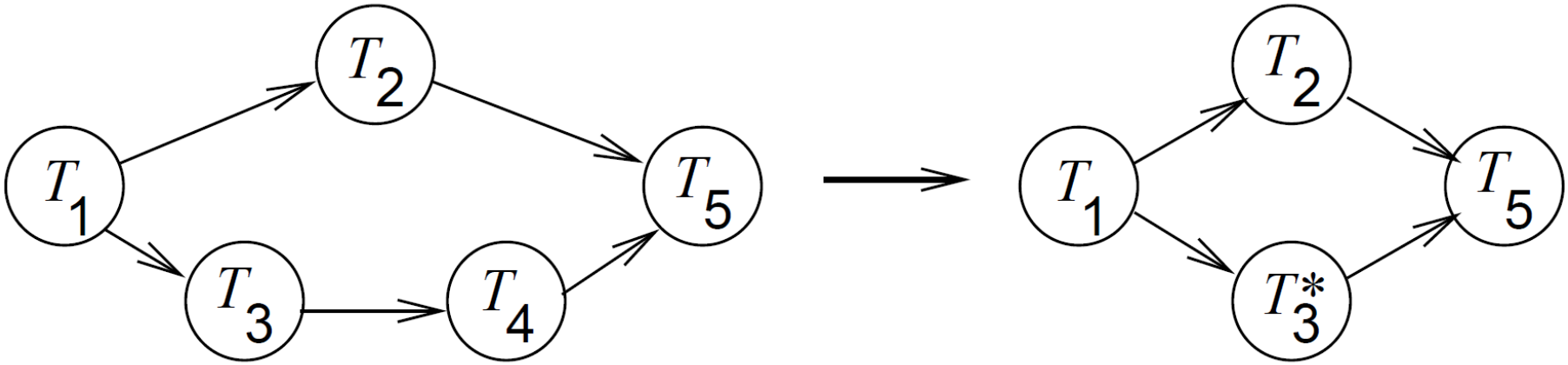
Book section 7.1

Granularity: size of tasks (e.g. in instructions)

Readable specifications and efficient implementations can possibly require different task structures.

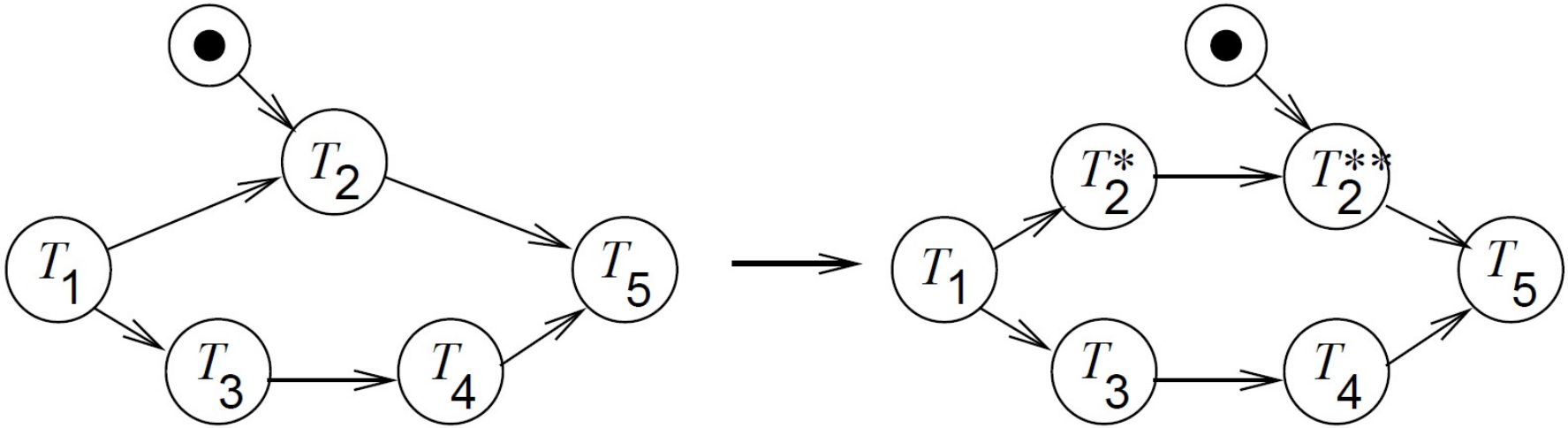
☞ Granularity changes

Merging of tasks



Reduced overhead of context switches,
More global optimization of machine code,
Reduced overhead for inter-process/task communication.

Splitting of tasks



No blocking of resources while waiting for input,
more flexibility for scheduling, possibly improved result.

Merging and splitting of tasks

The most appropriate task graph granularity depends upon the context ☞ merging and splitting may be required.

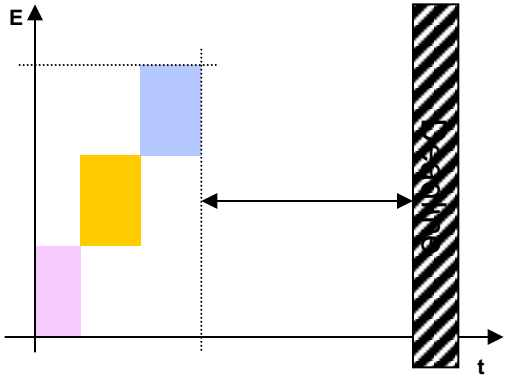
Merging and splitting of tasks should be done automatically, depending upon the context.

Task-level concurrency management

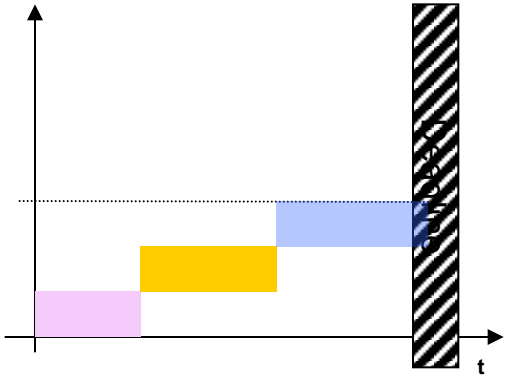
- The dynamic behavior of applications getting more attention.
- Energy consumption reduction is the main target.
- Some classes of applications (i.e. video processing) have a considerable variation in processing power requirements depending on input data.
- Static design-time methods becoming insufficient.
- Runtime-only methods not feasible for embedded systems.

→ How about mixed approaches?

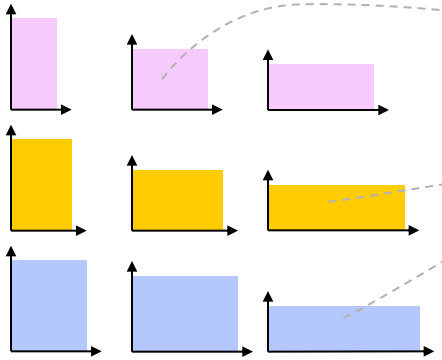
Example of a mixed TCM



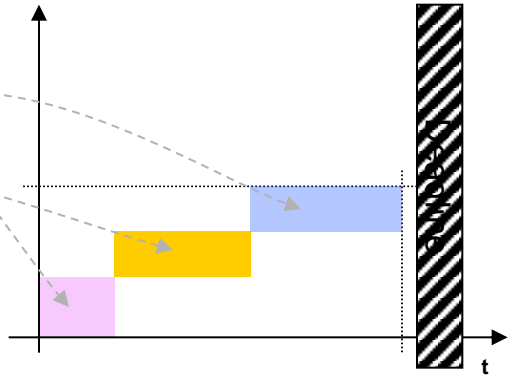
Static (compile-time) methods can ensure WCET feasible schedules, but waste energy in the average case.



…or they can define a probability for violating the deadline.



Mixed methods use compile-time analysis to define a set of possible execution parameters for each task.



Runtime scheduler selects the most energy saving, deadline preserving combination.

[IMEC, Belgium, http://www.imec.be/]