

Embedded Systems Design

Maryline CHETTO

Master Temps Réel – Systèmes Embarqués (CORO ERTS)



Topics

- **Design of multiprocessing systems**
- **Design of overloaded systems**
- **Design of energy constrained systems**
- **Design of energy harvesting systems**

DESIGNING A MULTIPROCESSING SYSTEM

Two approaches

Global scheduling:

choose a task, and assign it to one of the idle processors (otherwise, preempt one of the running task).

On-line processor assignment. With migration.

Partitioning:

Initially choose a processor for all tasks, and then run local scheduler on each processor.

Off-line processor assignment. No migration.

Global scheduling

Few theoretical results (feasibility tests) compared to monoprocessor real-time scheduling.

Drawbacks:

task migrations.

Difficult to apply to heterogeneous systems: task migration, hardware, operating systems.

Advantages:

Well suited for multiprocessing architectures with shared memory.

Two types of scheduling algorithms

1) Adapt mono-processor scheduling algorithms:

- Global RM, global EDF, global DM, global LLF, ...
- Choose the level of migration.
- Apply the scheduling algorithm on all the set of the processors.
- At each time, assign m highest priority tasks to the m processors.
- Preemptions occur when a job/task has to be run since its priority is higher and when all processors are busy.

2. New algorithms: PFair, LLREF, EDF(k), SA, EDZL, ...

Pfair scheduler: identical processors + synchronous periodic tasks with deadline equal to period = optimal scheduler

Global scheduling

A global scheduler has to solve two problems:

- When and how to choose the processor to run tasks
- When and how to assign priority to tasks.

Remark: EDF is not optimal in the multiprocessor case

Two kinds of migration:

- **Job level migration:** each job of a task can be run on any processor. But a job started on a given processor can not be moved on another.
- **Task level migration:** a task can run at any time on any processor.

Exercise on Global scheduling

Assume that the tasks are scheduled with RM or EDF. Look at the schedule produced for the following case:

- $n = m+1$, $P_i = 1$, $C_i = 2\varepsilon$, $u_i = 2\varepsilon$ for all $1 \leq i \leq m$
- $P_{m+1} = 1+\varepsilon$, $C_{m+1} = 1$, $u_{m+1} = 1/(1+\varepsilon)$

Solution to Exercise on Global scheduling

Dhall's effect [Dhall & Liu, 1978]

- Periodic task sets with utilization close to 1 are unschedulable using global RM / EDF
- $n = m+1$, $P_i = 1$, $C_i = 2\varepsilon$, $u_i = 2\varepsilon$ for all $1 \leq i \leq m$
- $P_{m+1} = 1+\varepsilon$, $C_{m+1} = 1$, $u_{m+1} = 1/(1+\varepsilon)$
- Task $m+1$ misses its deadline although U very close to 1

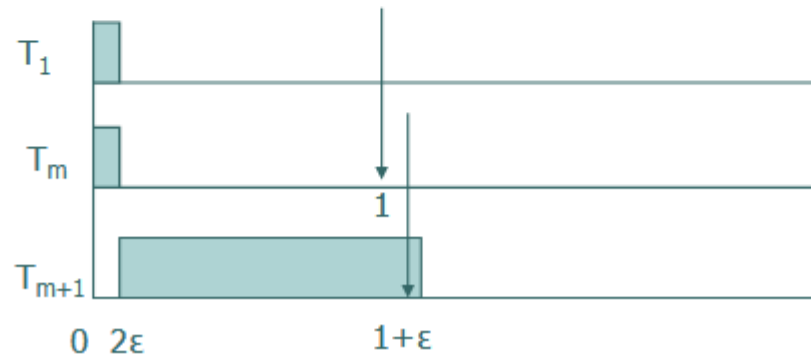
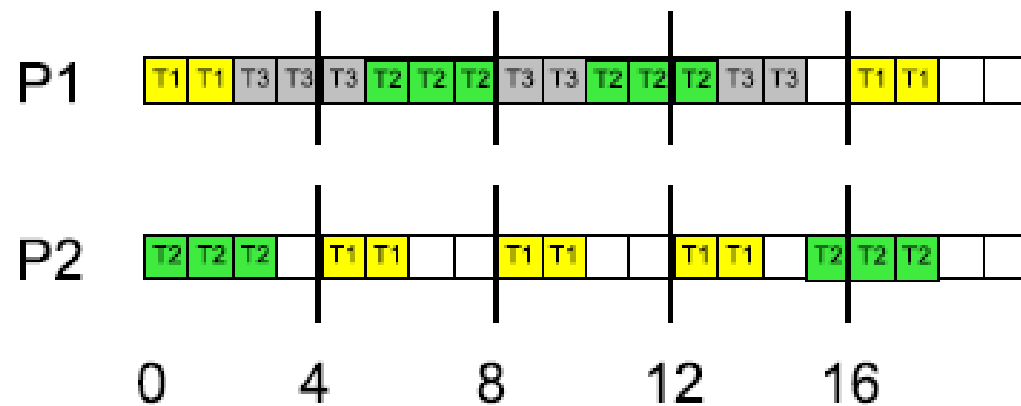


Illustration with Global Deadline Monotonic

- **Example:** global Deadline Monotonic

	Ci	Pi	Di
T1	2	4	4
T2	3	5	5
T3	7	20	20



• Priority assignment: $T_1 > T_2 > T_3$.

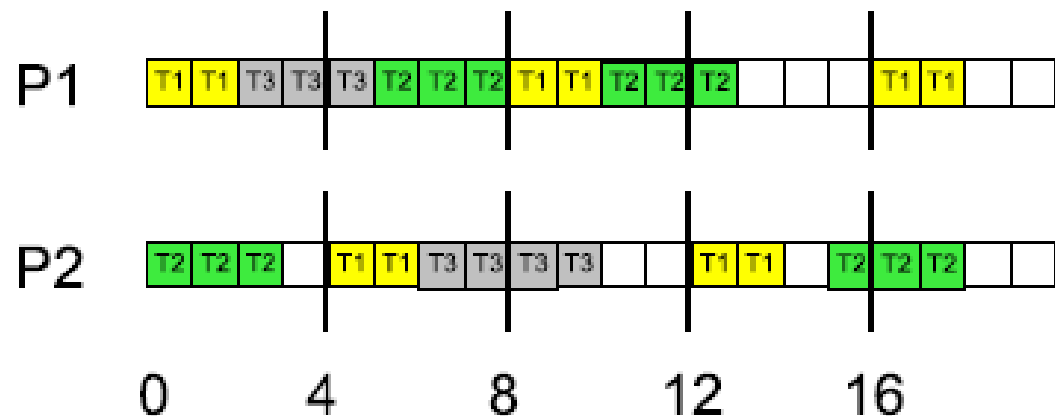
• Job level migration.

Thanks to F. Singhoff for this slide

Illustration with Global Deadline Monotonic

- **Example:** global Deadline Monotonic

	Ci	Pi	Di
T1	2	4	4
T2	3	5	5
T3	7	20	20



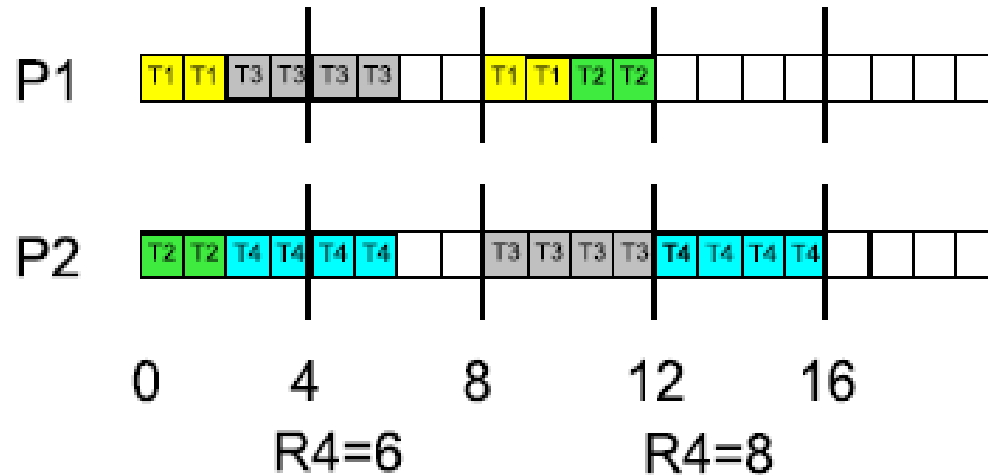
- Priority assignment: $T_1 > T_2 > T_3$.

- Task level migration.

Thanks to F. Singhoff for this slide

Anomalie: critical instant

	Ci	Di	Pi
T1	2	2	8
T2	2	4	10
T3	4	6	8
T4	4	7	8



- Critical instant:

In the monoprocessor case, the critical instant is the worst case on processor demand and occurs in the beginning of the scheduling (busy period).

Assumption used to compute the worst case response times.

It is not true in the multiprocessor case (see the example above).

Thanks to F. Singhoff for this slide

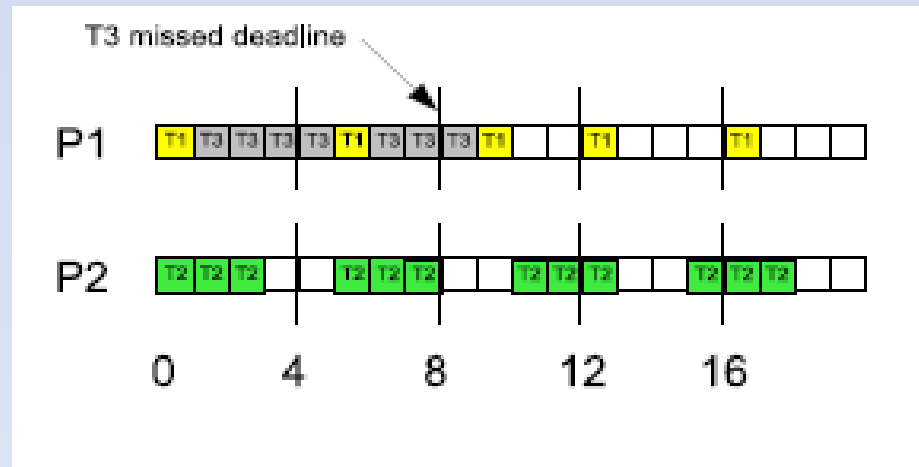
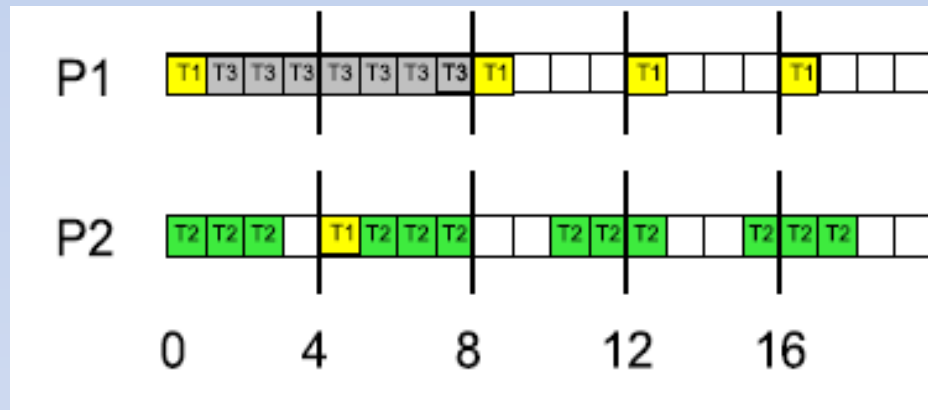
Other anomalie

A positive change on processor utilization (lower computation times or greater periods) on a feasible task set lead to an unfeasible task set.

Exercise: Let consider the following task set that is synchronous to time zero and consider two identical processors with job migration level.

	C_i	P_i	D_i
T1	1	4	2
T2	3	5	3
T3	7	20	8

Draw the schedule with global RM and verify that there is no deadline missing. Then, increase the period of task 1 from 4 to 5 and verify that there is a deadline missing.



The Pfair scheduling

We expect to find a schedule that leads to an equal processor utilization factor between all processors called "Proportionate Fair" (2005).

Pfair is an optimal scheduler with identical processors and synchronous periodic tasks with deadline equal to periods.

Task level migrations.

A lot of context switches.

Partitioning

Advantages:

- deterministic behavior in case of failure: failures of a task may only imply failure on its processor.
- Implementation: local schedulers are independent
- No migration costs
- Direct reuse of mono-processor schedulability tests
- Isolation between processors in case of overload

Drawbacks:

Partitioning is a NP-hard problem: Bin-packing.

Limits of Partitioning

Largest utilization bound for any partitioning algorithm
[Andersson, 2001]

$$\frac{m+1}{2}$$

Exercise:

Look at the following case : $m+1$ tasks of execution time $(1+\epsilon)$ and period 2

Partitionning

Two kinds of issue:

1) Number of processors to minimize: optimization problem (**bin-packing problem**)

- Bin = task with size = processor utilization
- Box = processor with size = ability to host tasks

2) Fixed number of processors: search problem (**knapsack problem**)

Both problems are NP-hard

Partitionning

- **Many parameters:**

Processors (identical or not), tasks (priority, period, capacity).
Take into account task communications, shared resources,...

- **Examples of objective functions:**

Minimize the number of processors,

Principles of a partitioning heuristic

1. Order tasks according to a given parameter (e.g. period, priority, processor utilization).
2. Do task assignment according to their order.
3. For each task, look for an available processor, according to a policy (e.g.: Best Fit, First Fit, Next Fit, ...).
4. An available processor is a processor that is able to run a task with no missed deadline: processors are selected according to a feasibility test.
5. Stop when all tasks are assigned.

Principles of a partitioning heuristic

Bin-packing heuristics: YY

- FF: First-Fit
- BF: Best-Fit
- WF: Worst-Fit, NF: Next-Fit
- FFD, BFD, WFD: First/Best/Worst-Fit Decreasing

Partitioning algorithms XX-YY

Practice: Rate Monotonic Next Fit

Tasks are increasingly ordered by their periods.

- We start with task $i = 0$ and processor $j = 0$.
- Assign task i on processor j if the feasibility tests is true (e.g. $U < 0.69$).
- Otherwise, put task i on processor $j + 1$.
- Assign the next task $i + 1$.
- Stop when all tasks have been assigned.
- $j =$ the number of required processors.

Comments on the heuristics

Trouble with Next Fit = sometimes leads to a low processor utilization factor.

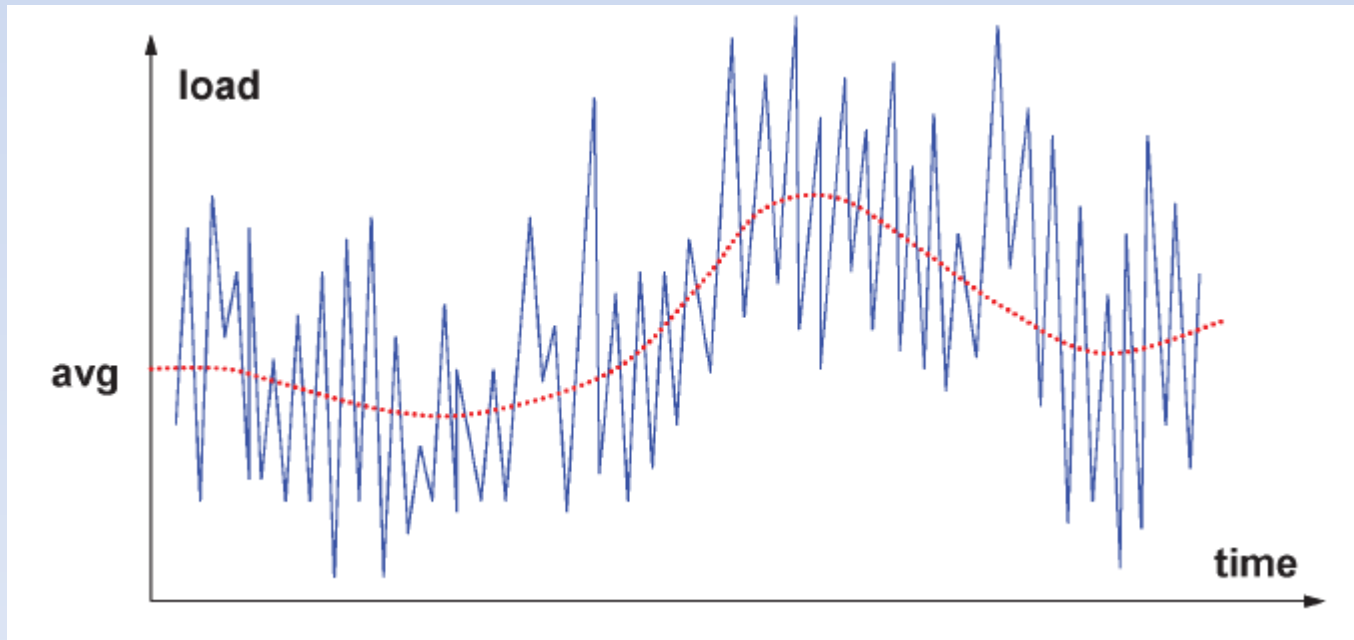
Others heuristics to increase processor utilization factor.:

First Fit : for each task i , start with processor $j = 0$ and assign task i on the first processor on which the feasibility test is true.

Best Fit : for each task i , start with processor $j = 0$ and assign task i on the processor on which the feasibility test is true and on which the processor utilization factor has the highest value.

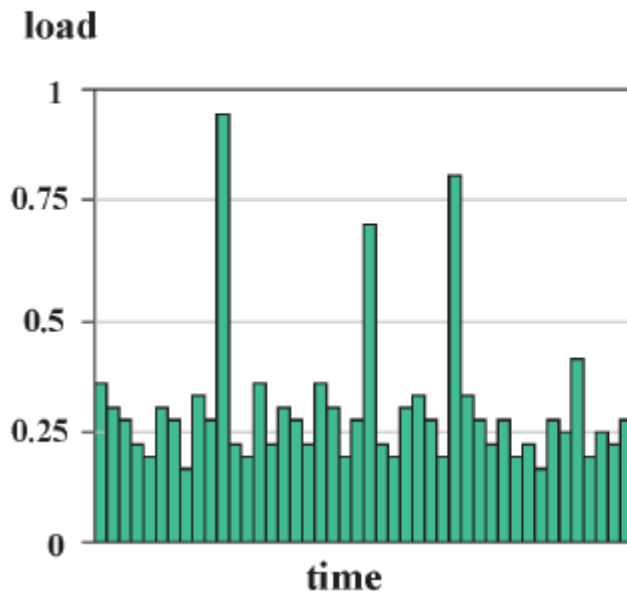
DESIGN OF AN OVERLOADED SYSTEM

Several applications are characterized by highly variable computing loads such as multimedia applications.

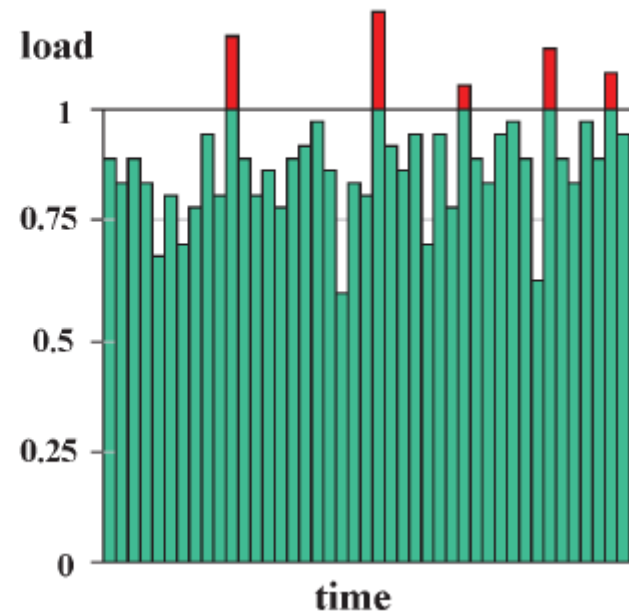


EXAMPLES

**System designed under
worst-case assumptions**



**System designed under
average-case assumptions**



Predictability vs Efficiency

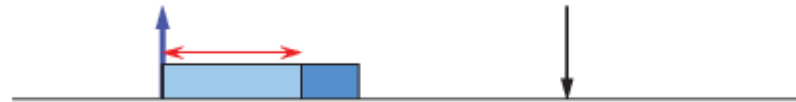
Worst case design i.e. pessimistic assumptions lead to predictability but no efficiency (in terms of hardware such as memory requirement, processor speed)

Average case design (average period, average execution times) lead to low predictability and high efficiency

Overrun

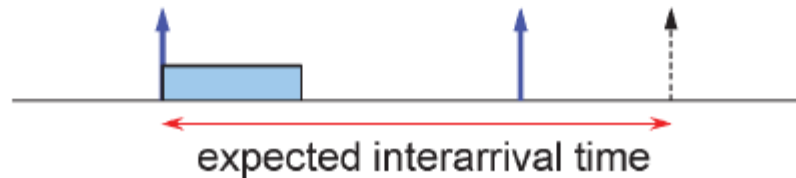
Execution Overrun

The task computation time exceeds its expected value:



Activation Overrun

The next task activation occurs before its expected time

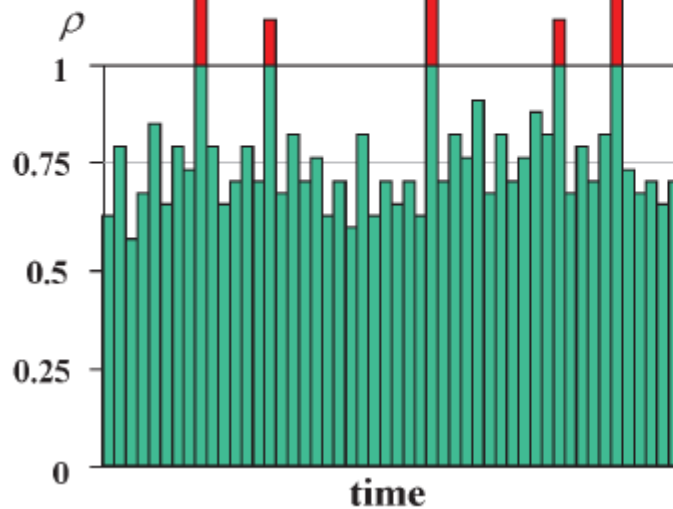


1

Overload

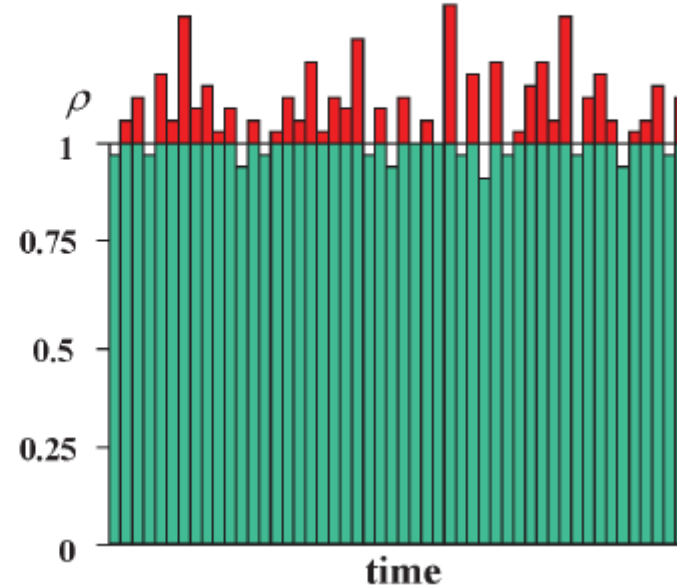
Transient Overload

$$\rho_{\max} > 1, \text{ but } \rho_{\text{avg}} \leq 1$$



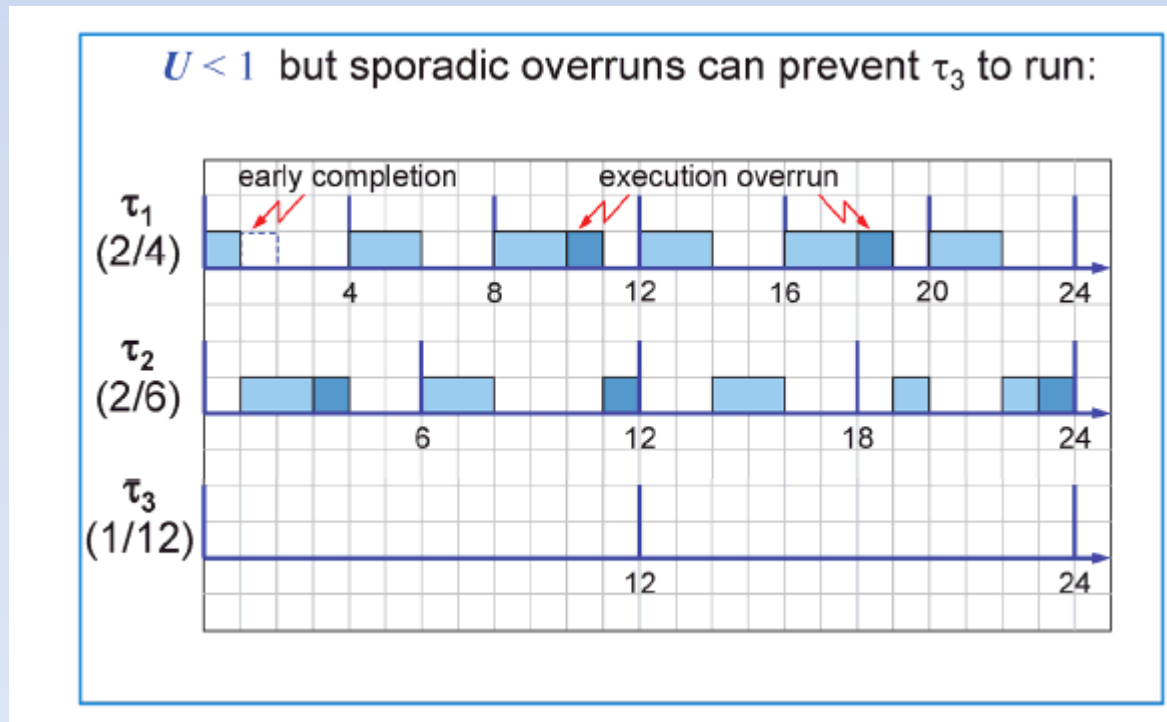
Permanent Overload

$$\rho_{\text{avg}} > 1$$



Consequences of overload

The overload can be transient (deadline missings) or permanent (starvation)



Handling permanent overload

There are two different approaches:

- **Value-based scheduling**
 - least importance tasks are rejected
 - important tasks receive full service
- **Performance degradation**
 - All tasks are executed
 - but with reduced requirements

Value based scheduling

- If $\rho > 1$, no all tasks can finish within their deadline.
 - To avoid domino effects, the load is reduced by rejecting the least important tasks.
 - To do that, the system must be able to handle tasks with both timing constraints and importance values.
-
- Under RM and EDF, the value of a task is implicitly encoded in its period or deadline.
 - However, in a chemical plant controller, a task reading the steam temperature every 10 seconds is more important than a task which updates the clock icon every second.

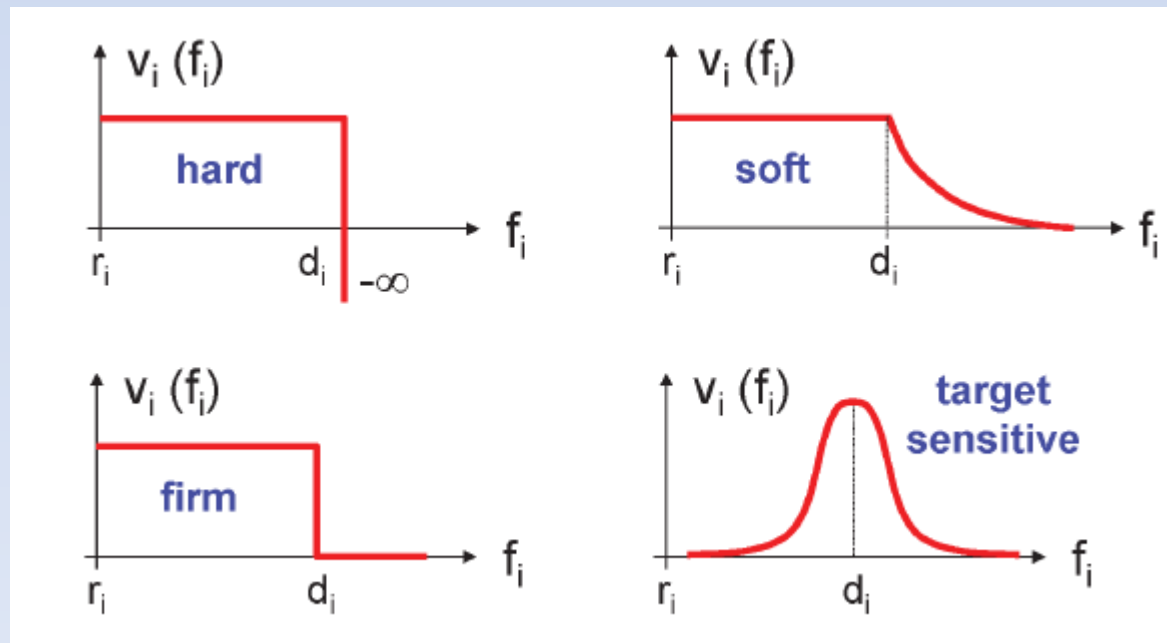
How to assign value ?

A task τ_i can be assigned a value v_i according to some criteria such as:

$v_i = V_i$	arbitrary constant
$v_i = C_i$	computation time
$v_i = V_i/C_i$	value density

Value as a function of time

A task τ_i can be assigned a value v_i which depends on its completion time (f_i) and its criticality:



Performance evaluation

The performance of a scheduling algorithm can be evaluated by the cumulative value:

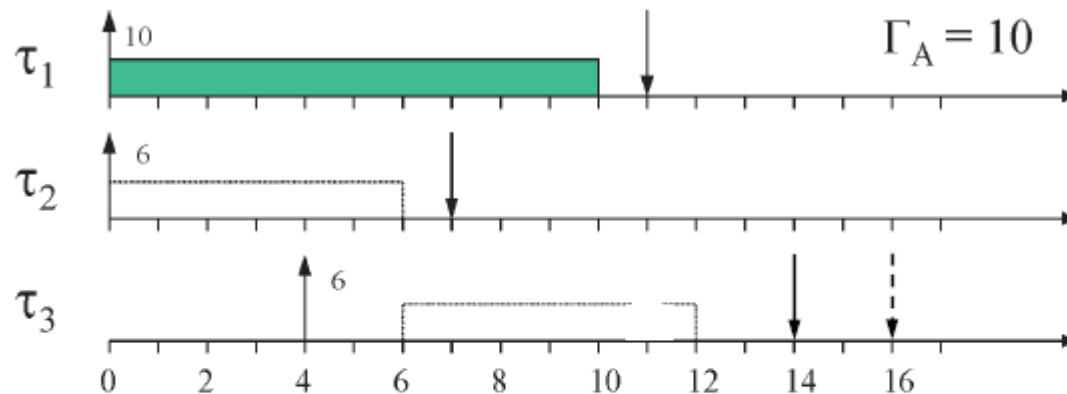
$$= \sum_{i=1}^n v_i(f_i)$$

Important note: The optimal cumulative value cannot be obtained by an online scheduling algorithm

Exercise: Give an illustrative example

Illustrative example

We assume that $V_i = C_i$



If at time $t = 0$, r_3 is not known, we cannot select the task that maximizes the cumulative value.

Competitive factor

- Let Γ^* be the maximum cumulative value achievable by an **optimal clairvoyant algorithm**.
- An algorithm **A** has a competitive factor φ_A , if it is guaranteed that, **for any task set**, it achieves:

$$\Gamma_A \geq \varphi_A \Gamma^*$$

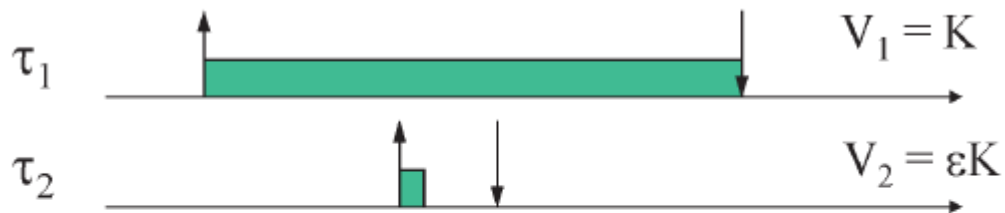
- Hence, $\varphi_A \in [0,1]$ and can be computed as:

$$\varphi_A = \min_{\tau} \frac{\Gamma_A(\tau)}{\Gamma^*(\tau)}$$

Exercise: Prove that EDF has a competitive factor equal to zero.

Solution of exercise

Prove that EDF has a competitive factor equal to zero.



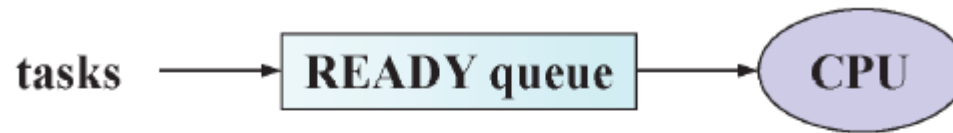
In such a situation, $\Gamma_{\text{EDF}} = V_2$ and $\Gamma^* = V_1$,
hence $\Gamma_{\text{EDF}} / \Gamma^* = V_2 / V_1 \rightarrow 0$ for $V_1 \gg V_2$

Important theoretical result

[Baruah et al., 91]

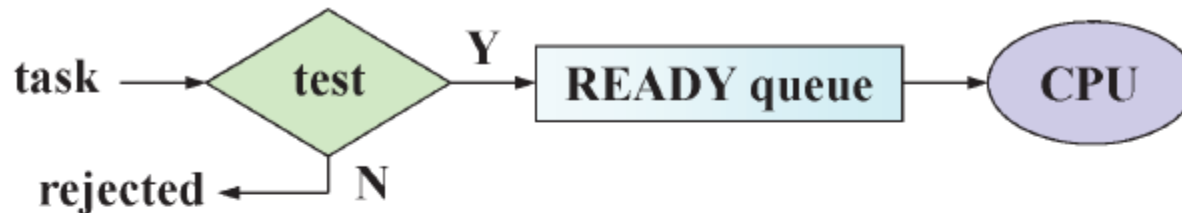
If $\rho \geq 2$ and task value is proportional to computation time, then no on-line algorithm can have a competitive factor greater than 0.25.

Solution 1: Handling overload with value based scheduling (Best Effort)



- Tasks are always accepted in the system.
- Performance is controlled through a suitable (value-based) priority assignment.
- **Problem:** domino effect.

Solution 2: Handling overload with Admission control



- Every task is subject to an acceptance test which keeps the load ≤ 1 . Check schedulability too
- It prevents domino effects, but does not take values into account.
- Low efficiency due to the worst-case guarantee (tasks may be unnecessarily rejected).

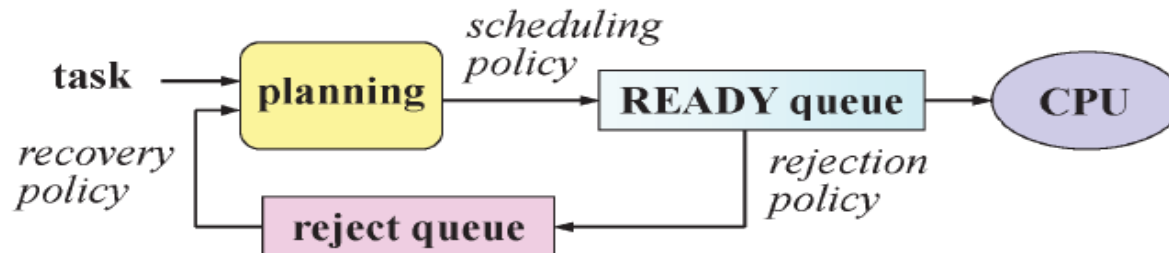
Solution 3: Handling overload with Robust scheduling

Best Effort Scheduling

... does not control **load**, but cares about **value**

Admission Control

... does not care about **value**, but controls **load**



- Task scheduling and task rejection are controlled by two separate policies.
- Tasks are scheduled by deadline, rejected by value.
- In case of early completions, rejected tasks can be recovered by a reclaiming mechanism.

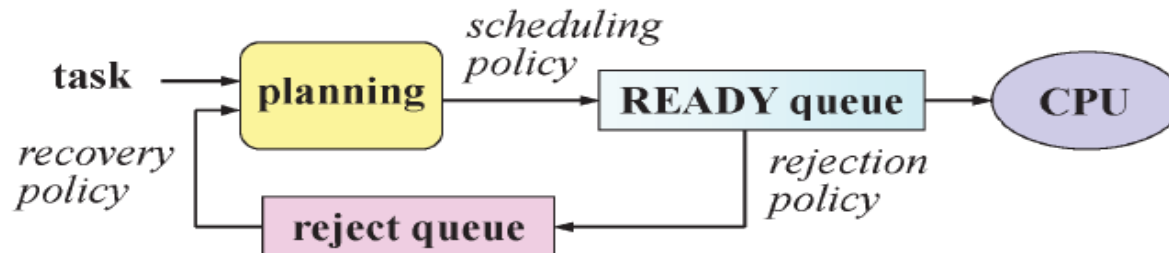
Robust EDF

Best Effort Scheduling

... does not control **load**, but cares about **value**

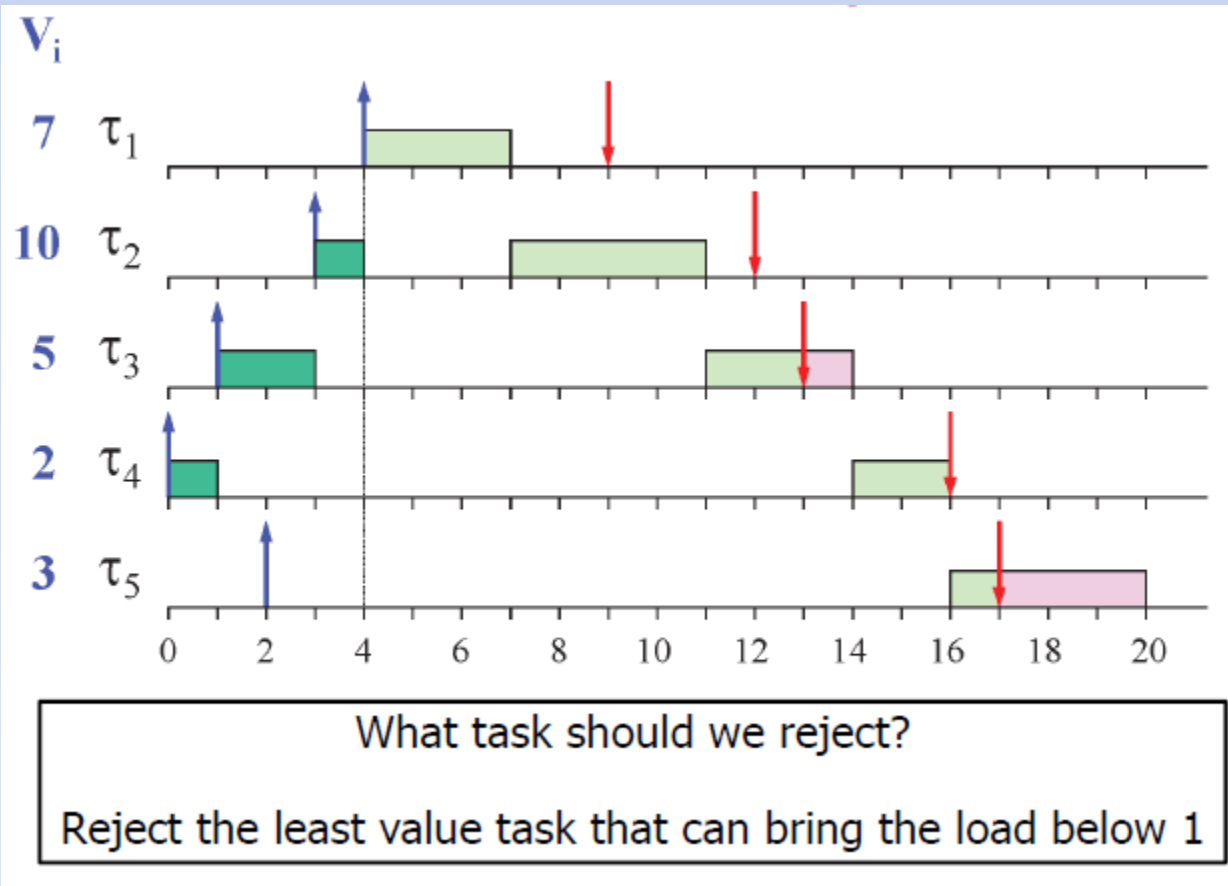
Admission Control

... does not care about **value**, but controls **load**

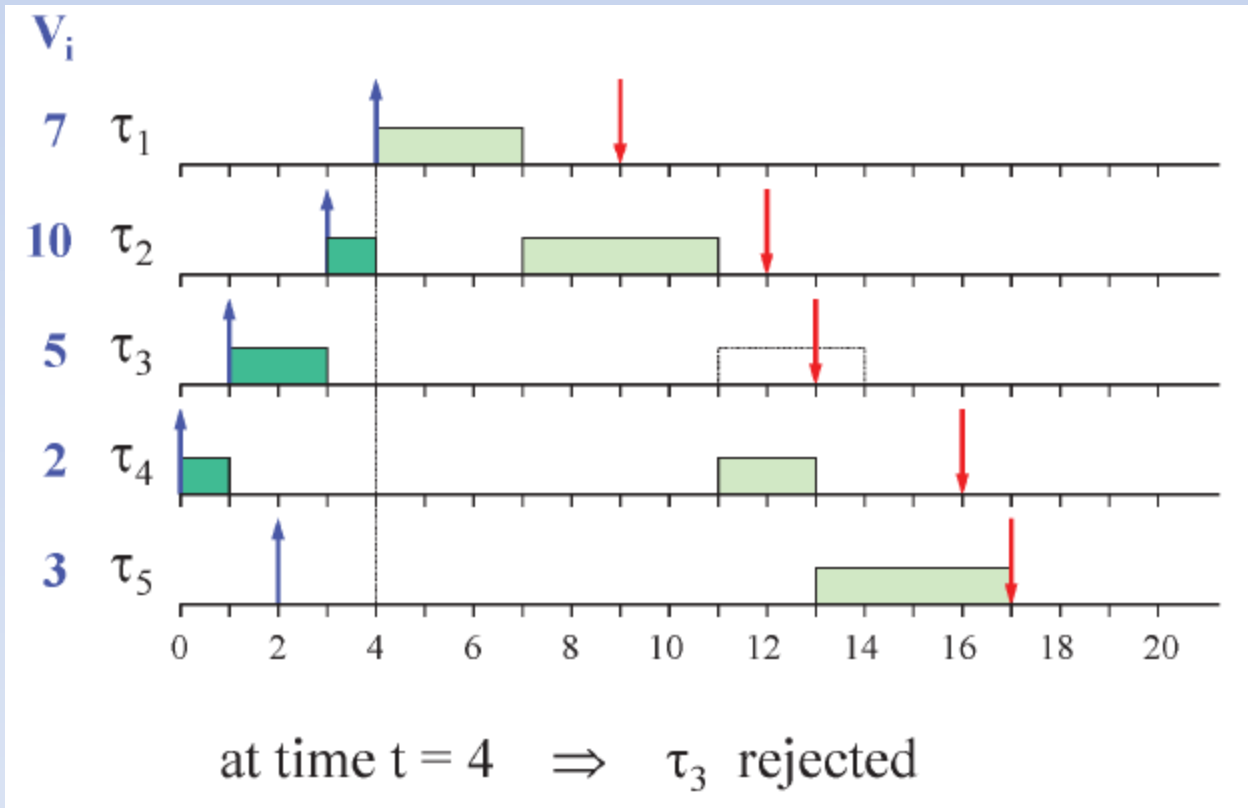


- Task scheduling and task rejection are controlled by two separate policies.
- Tasks are scheduled by deadline, rejected by value.
- In case of early completions, rejected tasks can be recovered by a reclaiming mechanism.

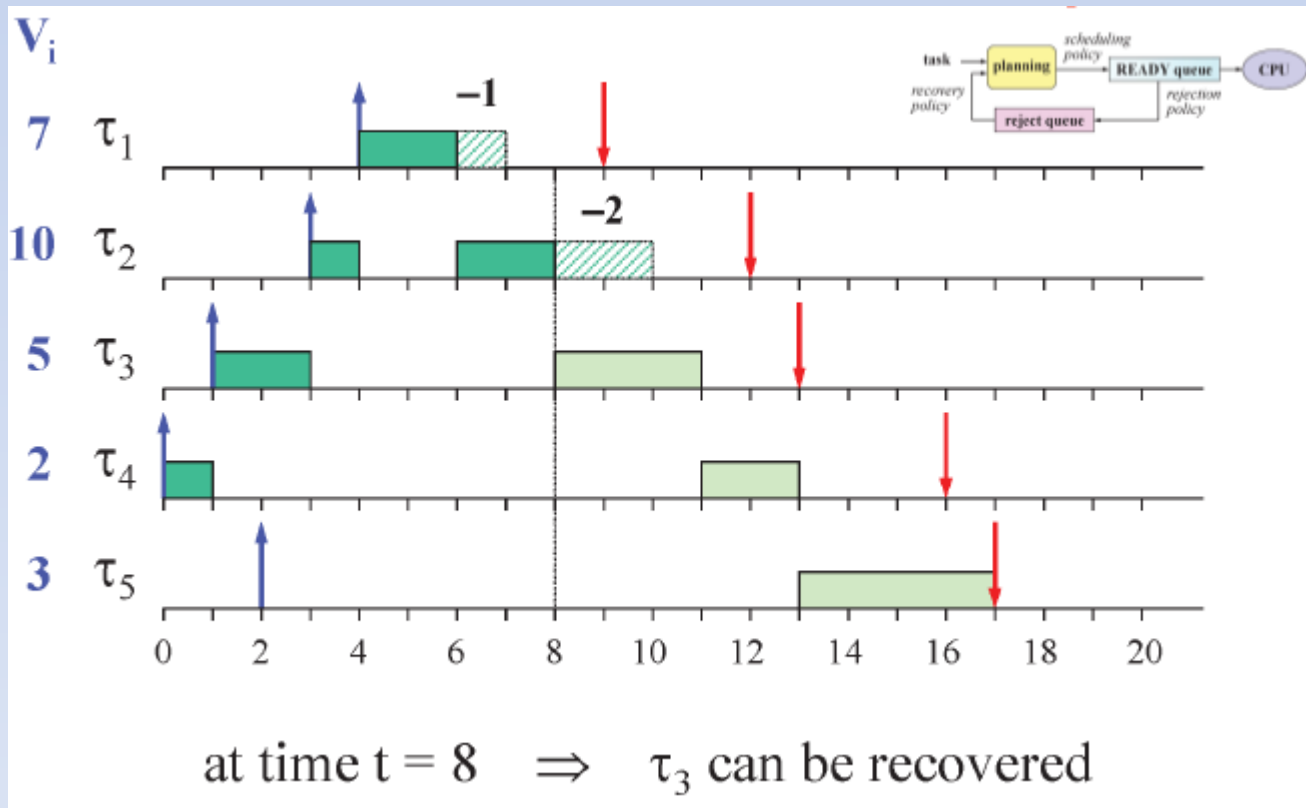
Robust EDF: example



Robust EDF: example



Robust EDF: example



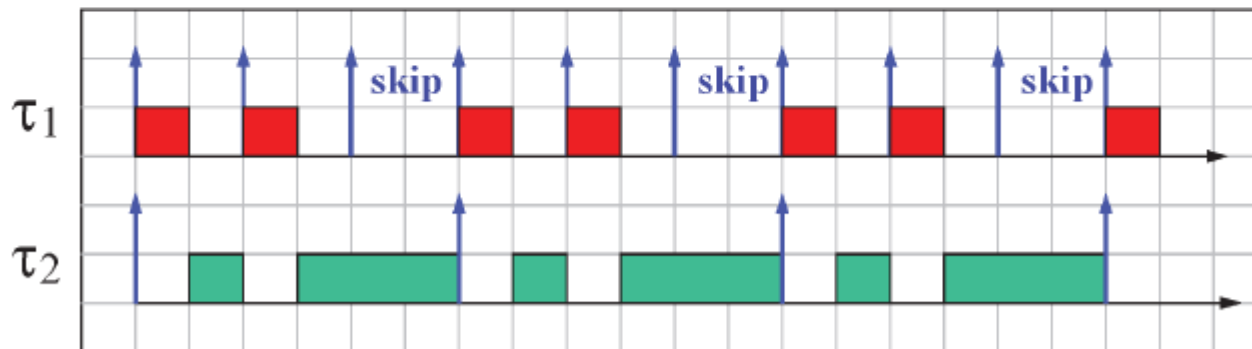
Other techniques to handle overloads

- Value based scheduling techniques
- Performance degradation
 - Job skipping
 - Computation time reducing
 - Period increasing

Job skipping to handle overloads

The system is overloaded, but tasks can be schedulable if τ_1 skips one instance every 3:

$$U_p = \frac{1}{2} + \frac{4}{6} = 1.17 > 1$$



The Skip Over model to handle overloads

Exercise (2 h 30):

You have to analyze the following paper:

Skip-Over: Algorithms and Complexity for Overloaded Systems That Allow Skips by *Gilad Koren* and *Dennis Shasha*, *Real Time Systems Symposium*, 1996.

Describe the assumptions of the Skip-Over model, explain the necessary feasibility condition, explain and illustrate the two basic scheduling approaches RTO and BWP, propose if possible a better scheduling approach.

Information about the exam of EMBEDD

The exam of Embedd will consist of two parts:

- 1) The first one will be questions and exercises about the course (1h15)
- 2) In the second one, you will have 45 mn to submit me one of the four paper analyzes you have done (Overload, energy minimization, energy harvesting)

DESIGN OF AN ENERGY EFFICIENT SYSTEM

Dynamic voltage frequency scaling (DVFS)

Introduction

- Energy consumption is an important issue in embedded systems.
 - Mobile and portable devices.
 - Laptops, PDAs.
 - Mobile and Intelligent systems: Digital camcorders, cellular phones, and portable medical devices.

Power consumption model

Frequency (MHz)	Voltage (V)	Normalized Frequency	Normalized power consumption
100	0.85	25%	11%
200	1	50%	30%
300	1.1	75%	54%
400	1.3	100%	100%

Intel PXA250

Frequency (MHz)	Voltage (V)	Normalized Frequency	Normalized power consumption
300	0.8	30%	11%
433	0.875	43%	20%
533	0.95	53%	28%
667	1.05	67%	44%
800	1.15	80%	63%
900	1.25	90%	83%
1000	1.3	100%	100%

Transmeta Crusoe TM5800

Consommation du DSP TMS 320C6201

Paramètre	Valeur
Tension d'alimentation	3.3 V
Configuration du Cache de données	4 KB, 2-way, 32 bytes
Temps d'accès à la ligne du cache de données (<i>Hit Latency</i>)	1 cycle
Accès à la mémoire (<i>Memory Access Latency</i>)	100 cycles
Energie par accès lecture au Cache de données	0.20 nJ
Energie par accès écriture au Cache de données	0.21 nJ
Energie par accès mémoire	4.95 nJ
Energie due aux Transactions des bus interne	0.04 nJ
Energie due aux Transactions des bus externes	3.48 nJ
Energie par Cycle d'horloge	0.18 nJ
Technologie	0.35 micron

Consumption of the different parts

- 58%for the core,
- 33% for memory cache,
- 7% for processor bus,
- 2% for internal SRAM

Case study (smart phones)

- Practical multi-core processors

Device	Chip maker	Processor name	Frequency	# cores
iPhone 5	Apple	A6	1.02GHz	2
Galaxy S III	Samsung	Exynos 4412	1.44GHz	4
Motorola RAZR	Ti	OMAP 4430	1.2GHz	2
HTC one S	Qualcomm	MSM8260A	1.2GHz	2
Asus transformer	NVIDIA	Tegra 3	1.3GHz	4
New iPad	Apple	A5X	1 GHz	2

- Contemporary multi-core processors have more than 2 cores at about 1 GHz.

Power consumption

- **Power in CMOS Circuits:**

$$P = \boxed{k_2 C_L V_{dd}^2 f} + \boxed{I_{leak} V_{dd}} + k_1 I_{direct} V_{dd}$$

Switching power Leakage power

- **Delay in CMOS Circuits:**

$$t_p = k \frac{V_{dd}}{(V_{dd} - V_{th})^\alpha}, \alpha = [2,3]$$

- **Power is a convex function of clock frequency (speed)**

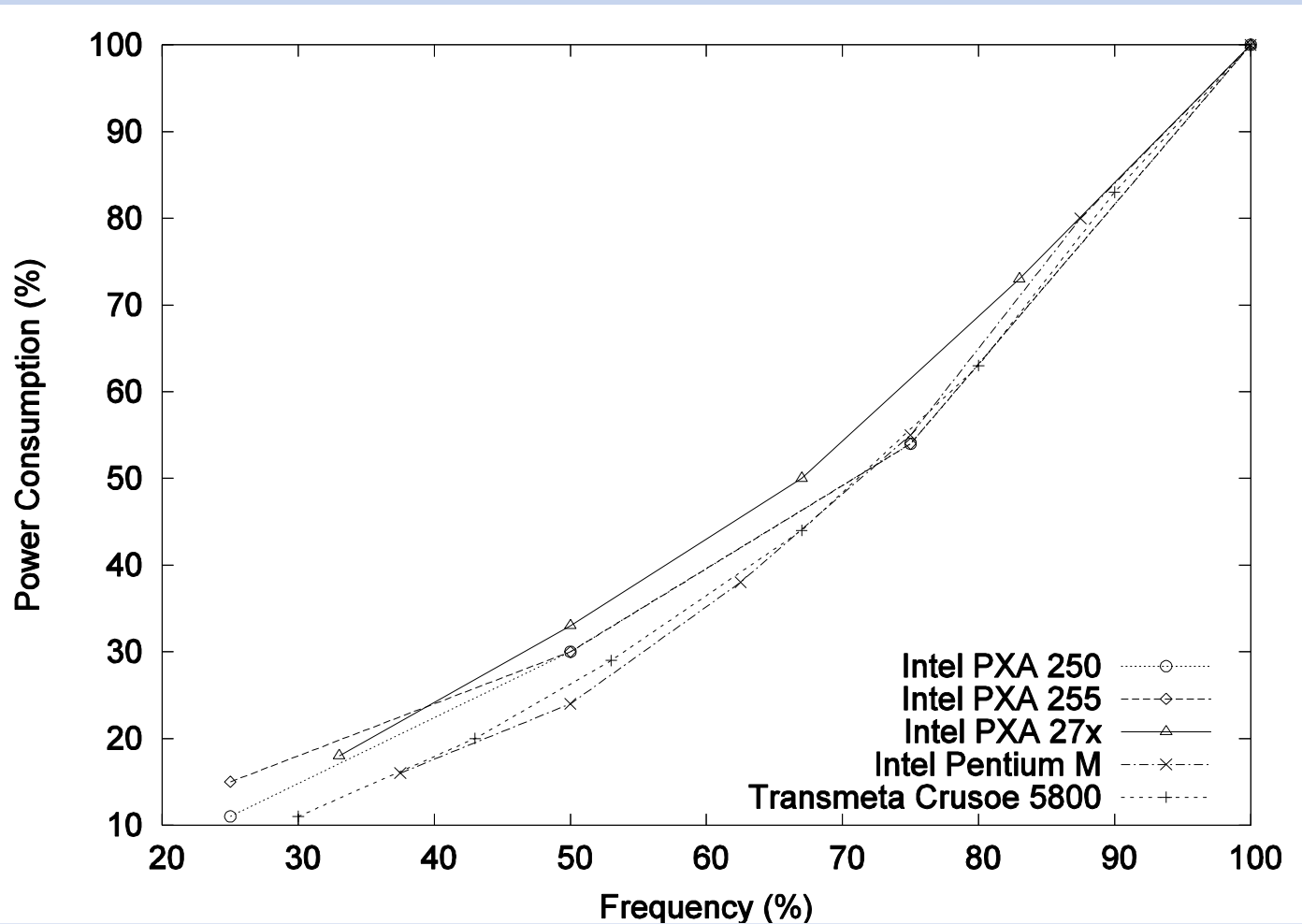
Important Facts (2)

- **Processors are based on CMOS logic**
 - **Static power + Dynamic power**

Dynamic power (due to switching activity)

- $P \propto V^2 \cdot f$
- $V \propto f$ V: voltage; P: power; E: Energy
- $E = P * T_{cc}$ $T_{cc} = CC/f$ (execution time)
- $E_i = K \cdot CC_i \cdot f^2$;
 CC_i : # clock cycles of task T_i .
 f : frequency at which T_i is run.

Power consumption



Two ways to save power

- **Shutdown** through choice of right system & device states

Multiple sleep states Also known as **Dynamic Power Management (DPM)**

- **Slowdown** through choice of right system & device state

Multiple active states Also known as **Dynamic Voltage/Frequency Scaling (DVS)**

- DPM + DVS

Choice between amount of slowdown and shutdown

DVS-example

- Consider a task with a computation time 20 units.

- Energy of T_i without DVS:

- $E1 = K * 20 * F^2.$

Time taken =
 $t1$ (say)

- Energy of T_i with DVS:

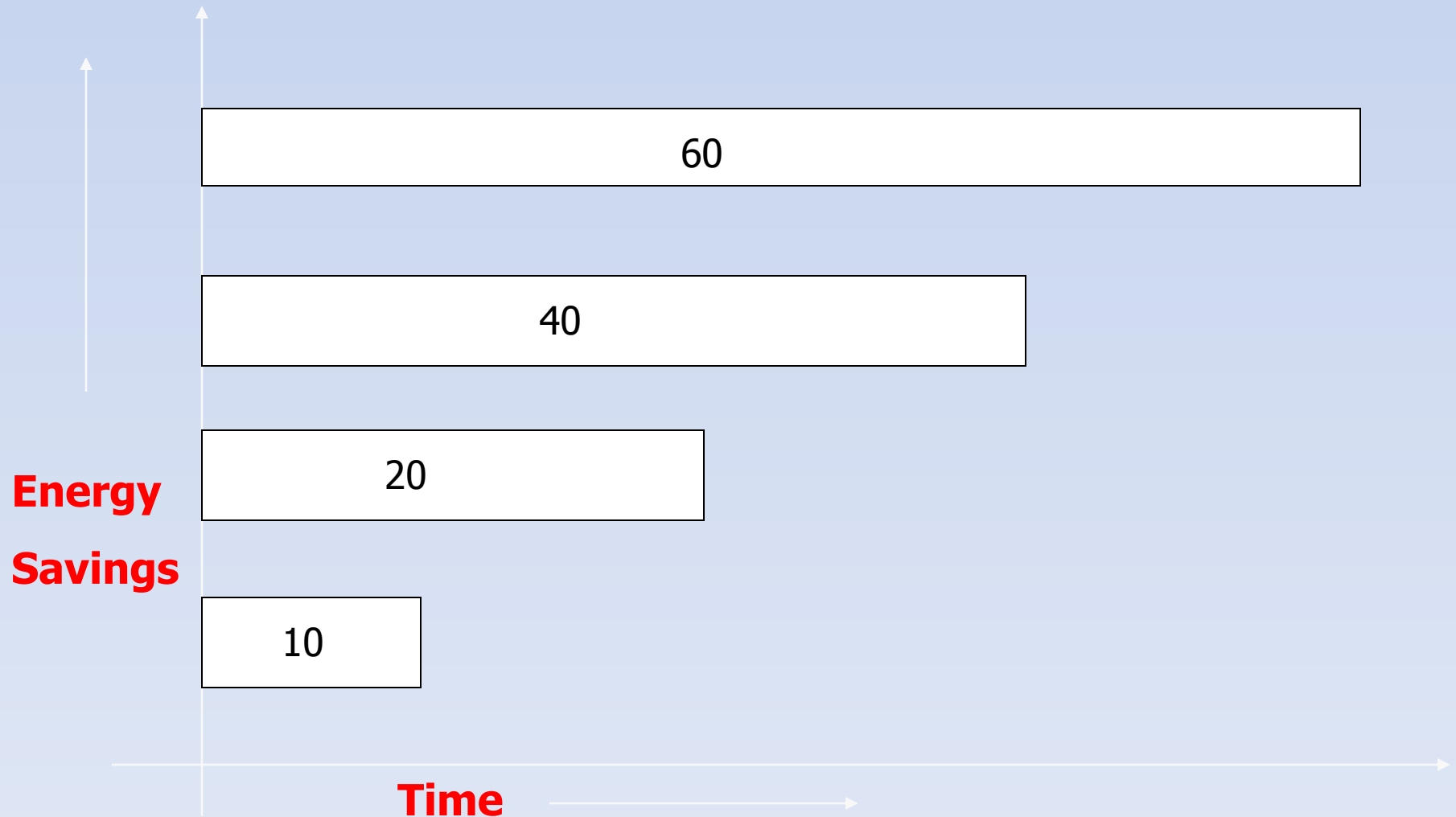
- $E2 = K * 20 * (F/2)^2.$

- Clearly, $E2 = (E1)/4$

Time taken =
 $t2 = 2 * t1$

Therefore, if we reduce the frequency we save energy but, we spend more time in performing the same computation

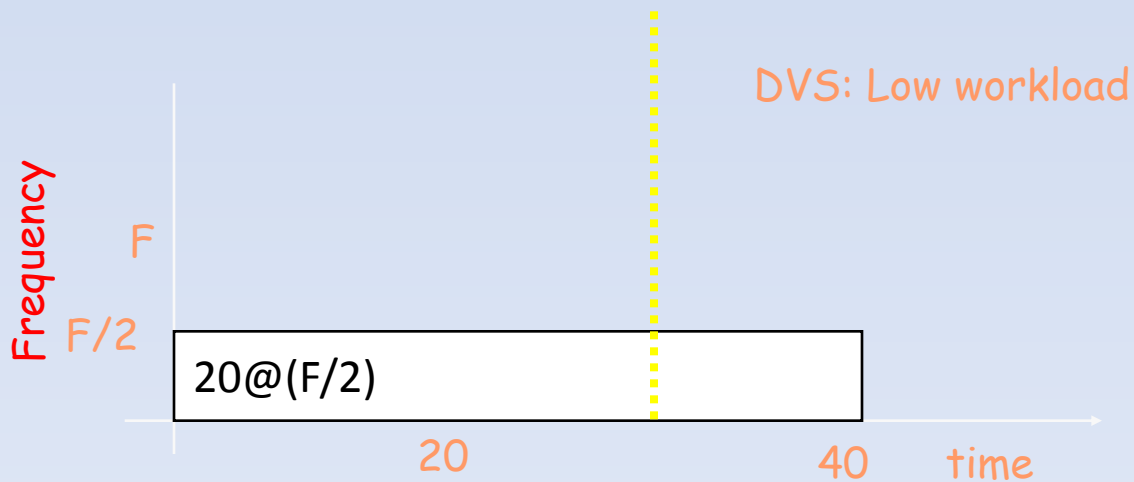
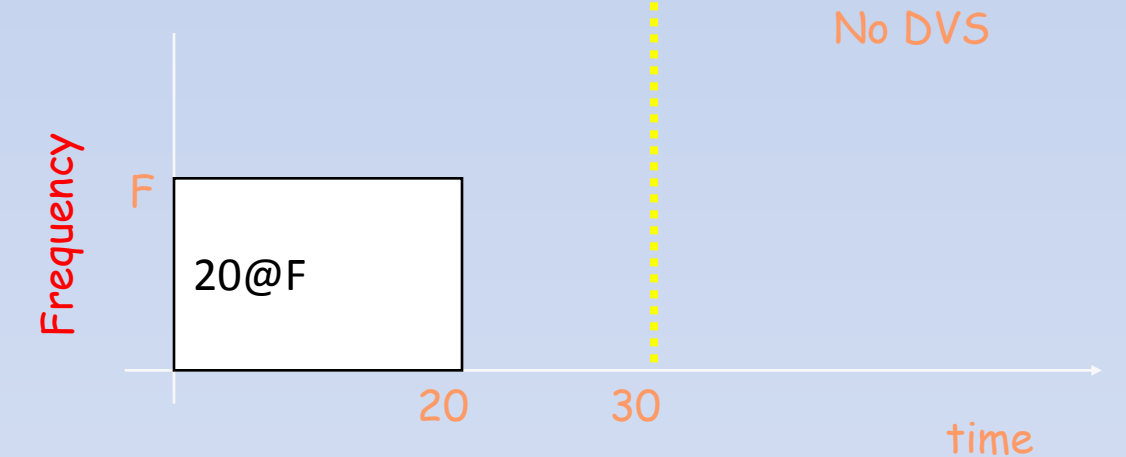
Energy-Time Tradeoffs



Simple DVS scheme handling RT-task

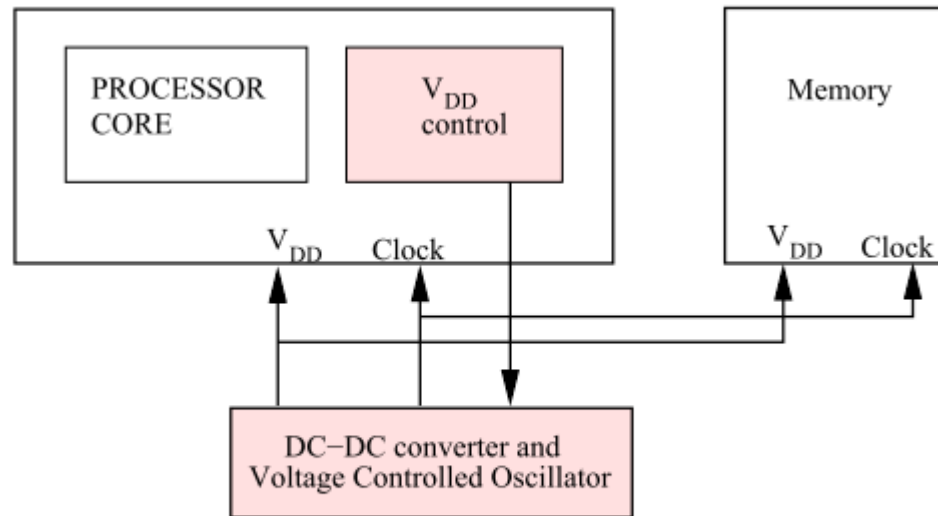
- Consider a real-time task $T1 = (20, 30)$
- Applying the simple DVS scheme discussed earlier:
 - If there is peak load $T1$ runs at maximum frequency (F) and meets the deadline with no energy savings
 - If there is less load $T1$ runs at half the maximum frequency ($F/2$) and completes at time = 40 thereby missing its deadline

Simple DVS scheme handling RT-task



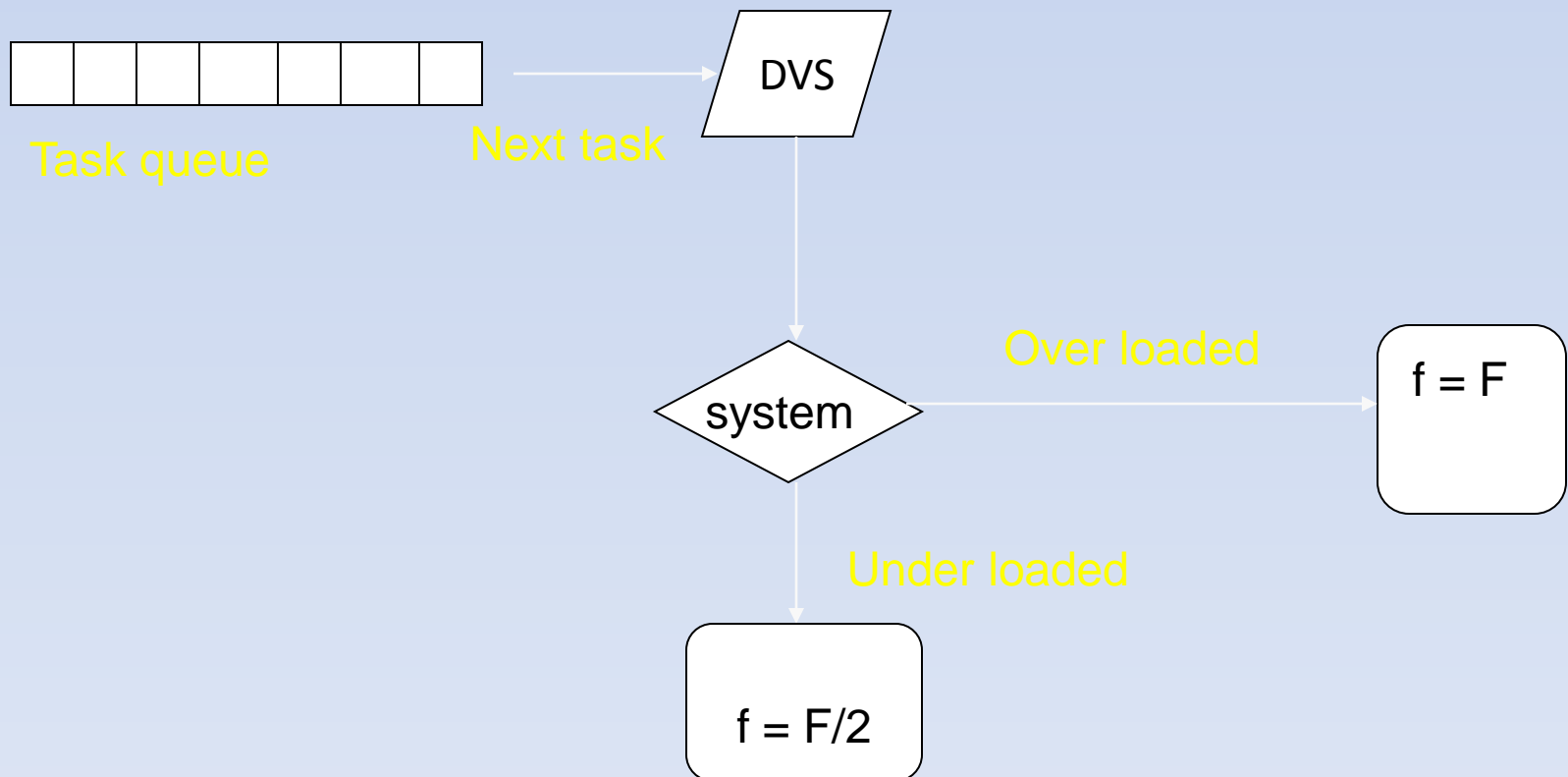
Inference:
DVS cannot be
blindly applied to
real-time
embedded
systems

Implementing DVS



Augmented processor blocks to implement dynamic voltage/frequency scaling

Simple DVS-Scheme



Exercice

- Soit un programme constitué de 10^9 cycles d'instruction et devant s'exécuter en un temps inférieur ou égal à 25s. Le modèle de consommation de la machine est le suivant:

Tension	Fréquence	Energie/Cycle
2.5 V	25Mhz	10nJ
5.0 V	50Mhz	40nJ

1. Quelle est la consommation à 50MHz ?
2. Trouver la fréquence de fonctionnement idéale.

Problem Definition

- ■ **A set of RT tasks:** $J = \{J_1, J_2, \dots, J_n\}$
 - release times r_i
 - deadlines d_i
 - worst case execution cycles c_i
- ■ **A processor**
 - Constant Time Transition Overhead: Δt
 - Energy Transition overhead: $\Delta E(V_1, V_2)$
 - A set of valid speeds (normalized): $S = \{S_1, S_2, \dots, 1.0\}$
 - A set of corresponding voltages: $V = \{V_1, V_2, \dots, V_m\}$
 - Instructions are not executed during speed transitions
- **Goal:** Produce a valid schedule
- **Secondary Goal:** Minimize Energy

Taxonomy of power-aware algorithms

- **Static (off-line) algorithms:**
 - Typically applied to periodic tasks
 - Make use of off-line parameters (e.g. periods, WCETs)
 1. Fixed System Voltage
 2. Fixed Task Voltage
- **Dynamic (on-line) algorithms:**
 - Take advantage of early completions
 1. Not based on prediction
 2. Based on prediction

Static algorithms

- Two techniques:

1. Fixed System Voltage

- All tasks have the same speed
- Low energy saving, no overhead
- *RTDVS-Static for EDF* (Pillai, 2001)
 - Selects the lowest possible speed (optimal)

2. Fixed Task Voltage

- Assigns a different speed to each task

Main approaches

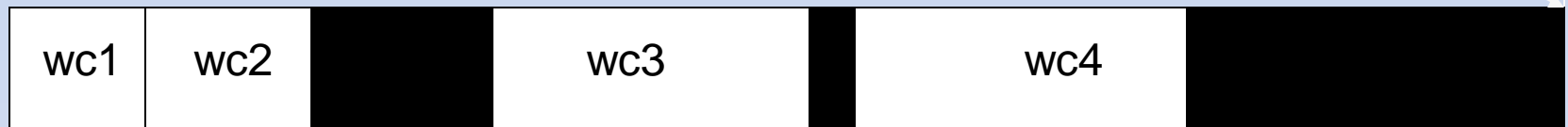
- Static voltage scaling EDF (static)
- Cycle conserving RT-DVS (dynamic)
- Look-Ahead RT-DVS (dynamic)

Static Voltage Scaling EDF: Motivation

Pre-run schedule with holes

WC_i = worst case computation time @ F_{\max}

Next arrival
of T1



Holes in the pre-run schedule imply:

EDF Test:

$$\sum(wc_i/p_i) < 1 \quad \text{at frequency} = F_{\max}$$

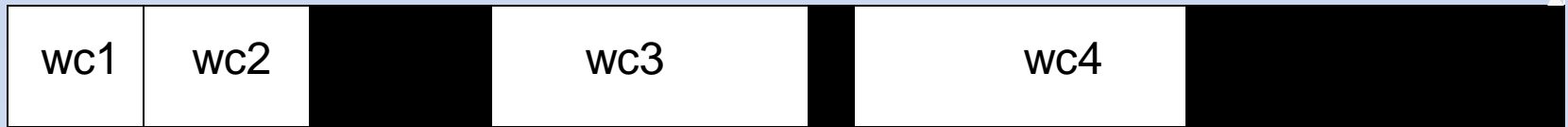
In other words, whenever $\sum(wc_i/p_i) < 1$ there are holes in the EDF schedule

Static Voltage Scaling EDF: exploiting holes

Pre-run schedule with holes

WC_i = worst case computation time @ F_{\max}

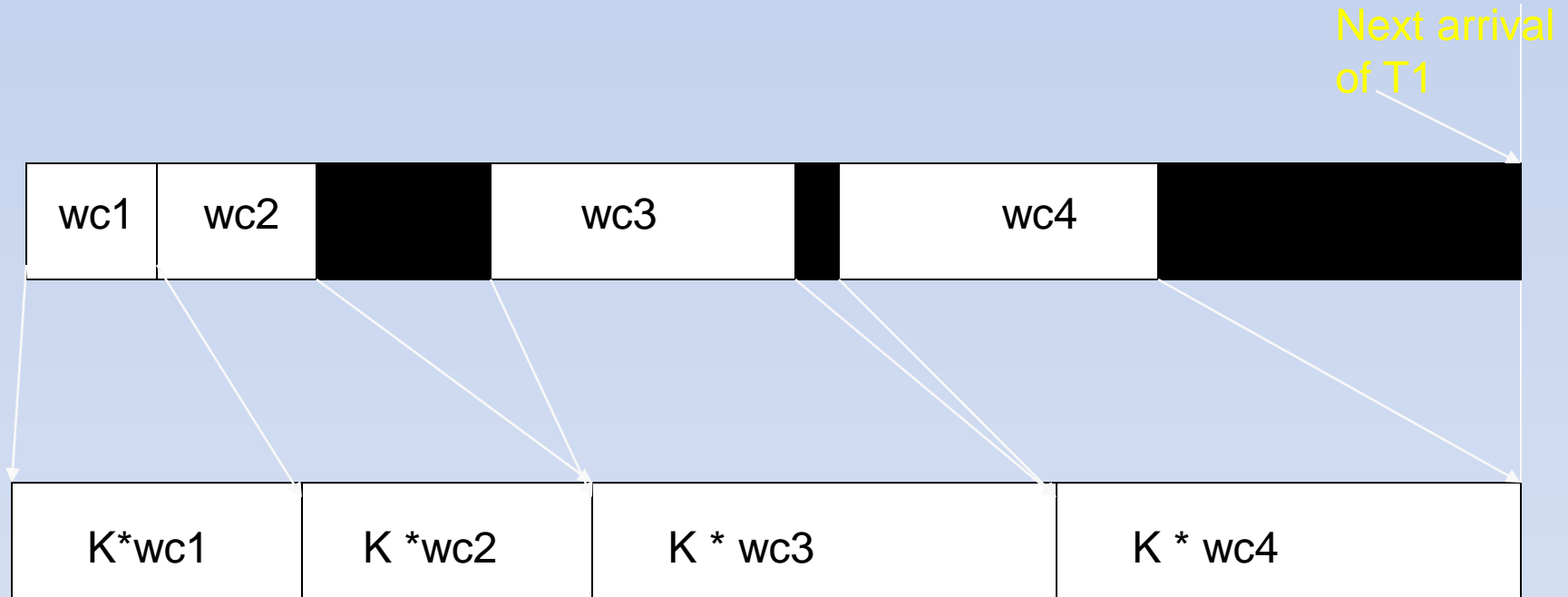
Next arrival
of T1



Processor typically idles
during holes.

Instead, the holes can be
exploited to slowdown the
processor to save energy

Static Voltage Scaling EDF



EDF Test:

$$\sum(wc_i/p_i) < 1 \quad \text{at maximum frequency} = F_{\max}$$

Static-VS EDF Test:

$$K* [\sum(wc_i/p_i)] = 1 \quad \text{at frequency} = F_{\max}/K$$

Static voltage scaling: Exercise

- Task set: $T1 = (1, 4)$ and $T2 = (2, 8)$
- $U = 1/4 + 2/8 = 0.5 (< 1) @ F_{\max}$

Questions:

- What is the “k” at which the task set is still schedulable @ (F_{\max} / k) :
- What is the energy gain ?

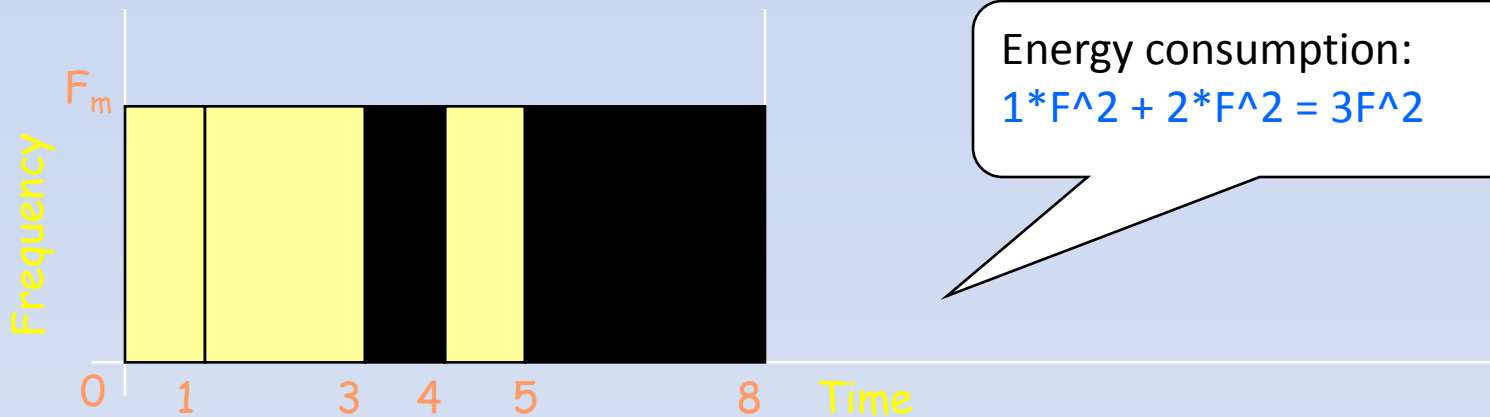
Static voltage scaling: **Solution**

- Task set: $T1 = (1, 4)$ and $T2 = (2, 8)$
- $U = 1/4 + 2/8 = 0.5 (< 1) @ F_{\max}$
- What is the “k” at which the task set is still schedulable @ (F_{\max} / k) :
 - Let $K = x$
 - $U = (1*x)/4 + (2*x)/8 = x*(0.5) = 1$
 - $X = 2$, that is $k = 2$
 - Therefore, we can operate at $f = F_{\max} / 2$ and still meet the deadlines

Static voltage scaling: Solution

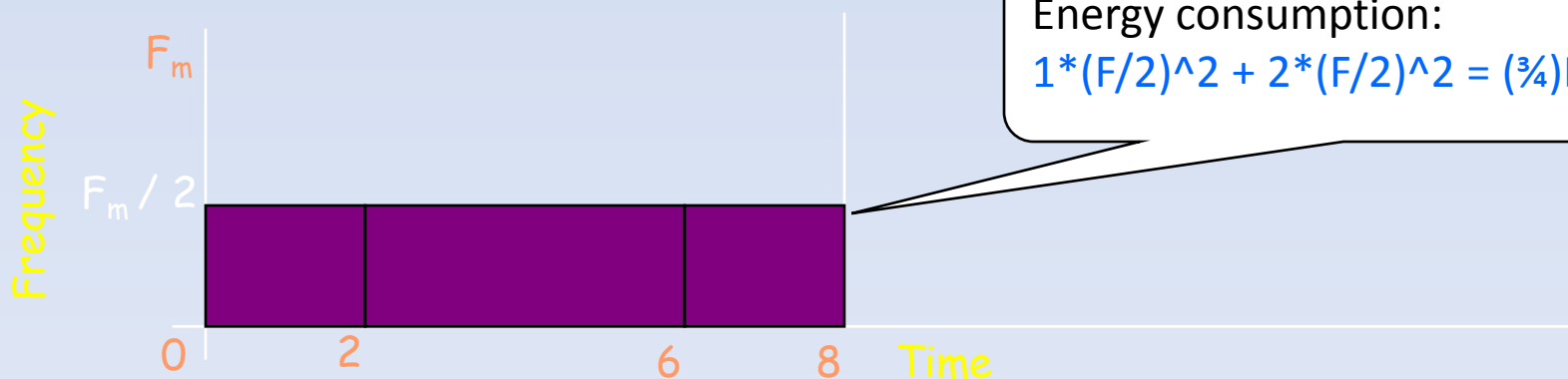
Modified Task set @ ($F_{\max}/2$): T1 = (2, 4) and T2 = (4, 8)

$U = 2/4 + 4/8 = 1$ @ ($F_{\max}/2$)



Energy consumption:

$$1 * F^2 + 2 * F^2 = 3F^2$$



Energy consumption:

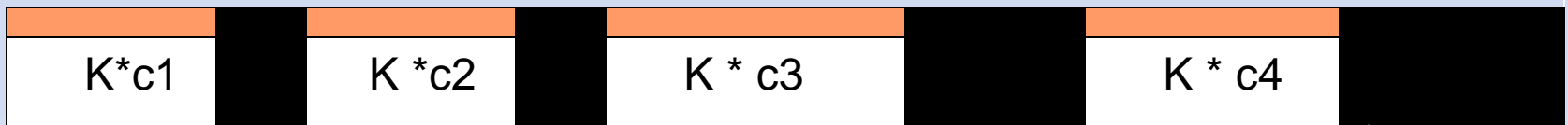
$$1 * (F/2)^2 + 2 * (F/2)^2 = (\frac{3}{4})F^2$$

What if $(C_i < WC_i)$??

Actual
computation
time

Worst case
computation
time

Next arrival
of T1

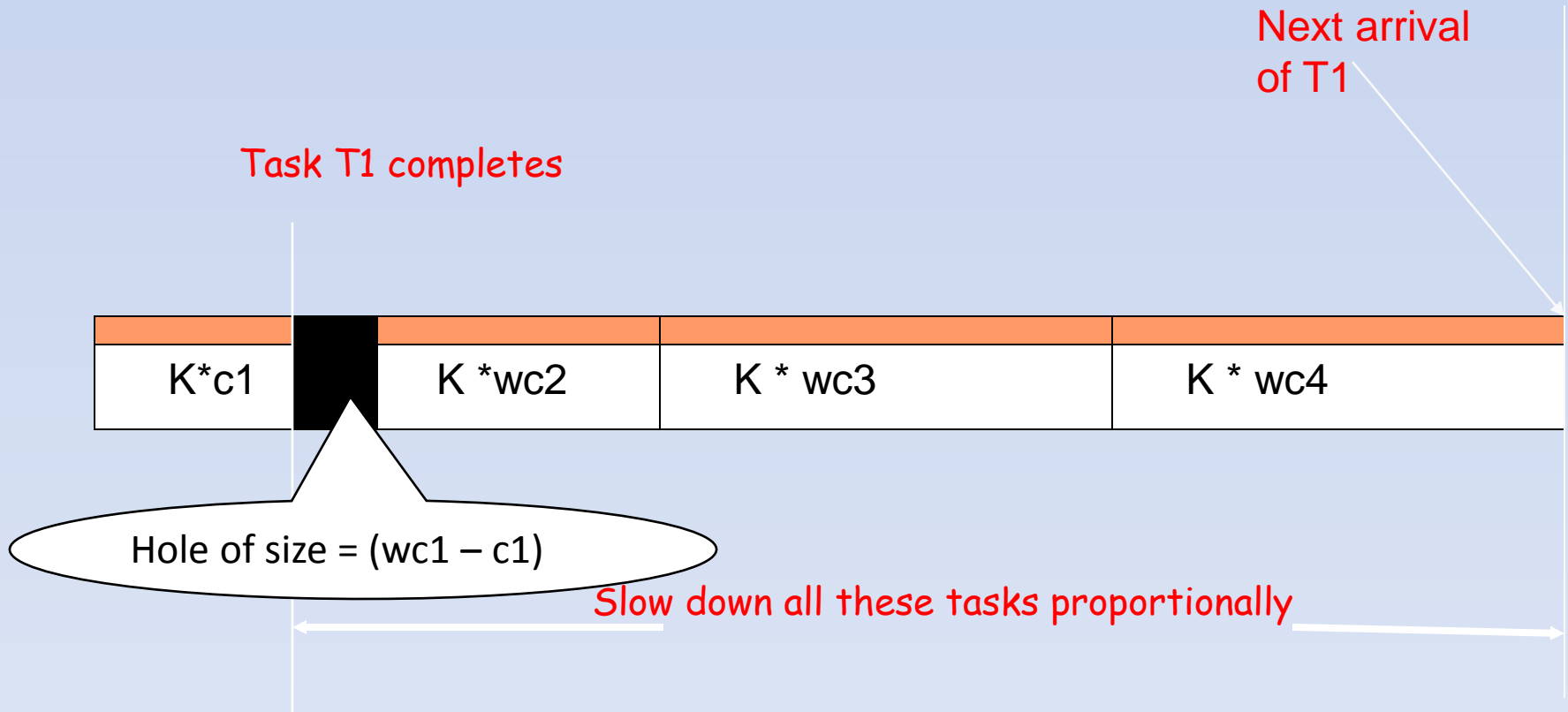


More holes left unexploited

Dynamic algorithms

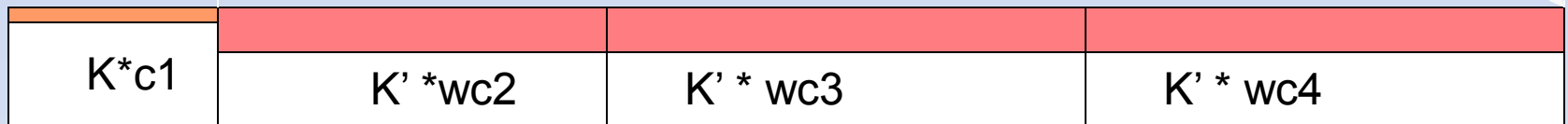
- **Take advantage of all jobs having $C_{ik} < WCET_i$**
 - For real-world tasks, C_{ik} may vary up to 87% wrt $WCET_i$
- More overhead than in static algorithms
- Two techniques:
 1. Not based on prediction
 - No assumptions on task duration
 2. Based on prediction
 - Try to foresee future load requirements
 - Performance: adaptability to ever-changing workloads

What if $(C_i < WC_i)$??



What if $(C_i < WC_i)$?

Next arrival
of T1



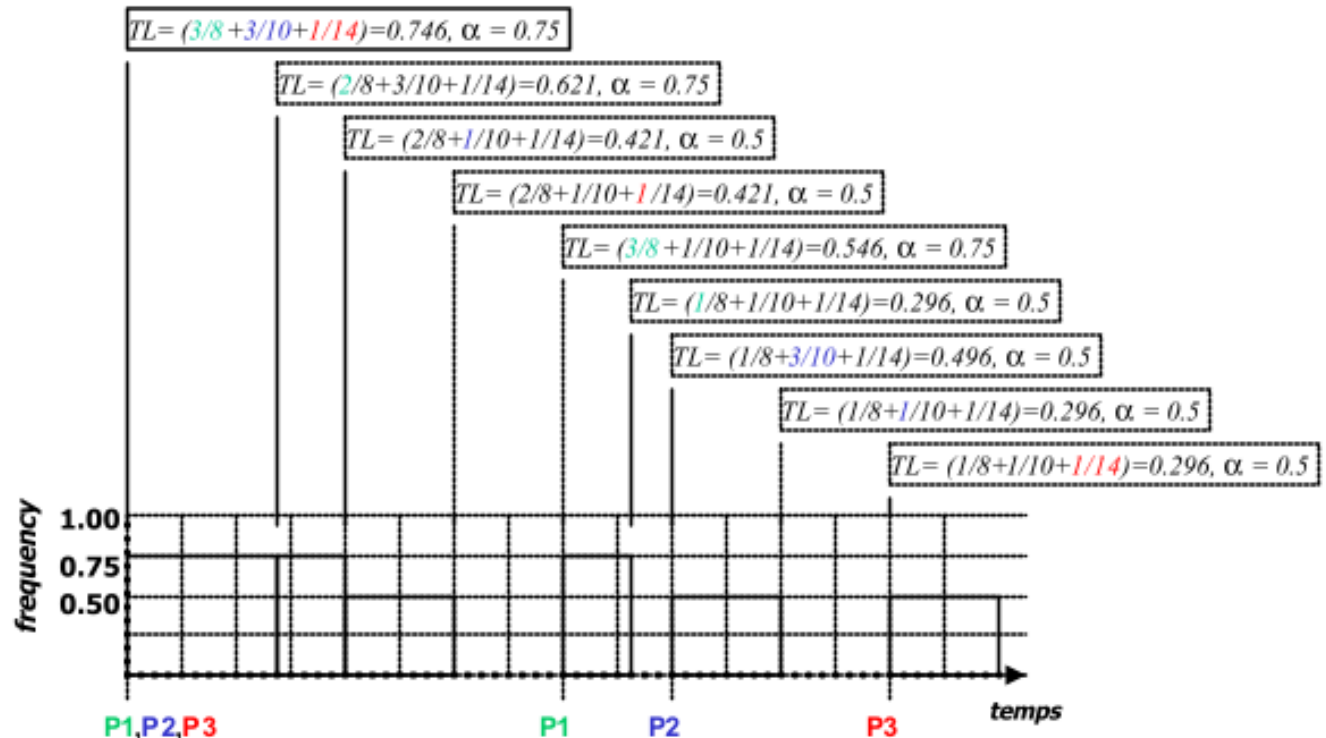
CPU Cycles are conserved by slowing down the remaining tasks

Cycle conserving EDF: Example

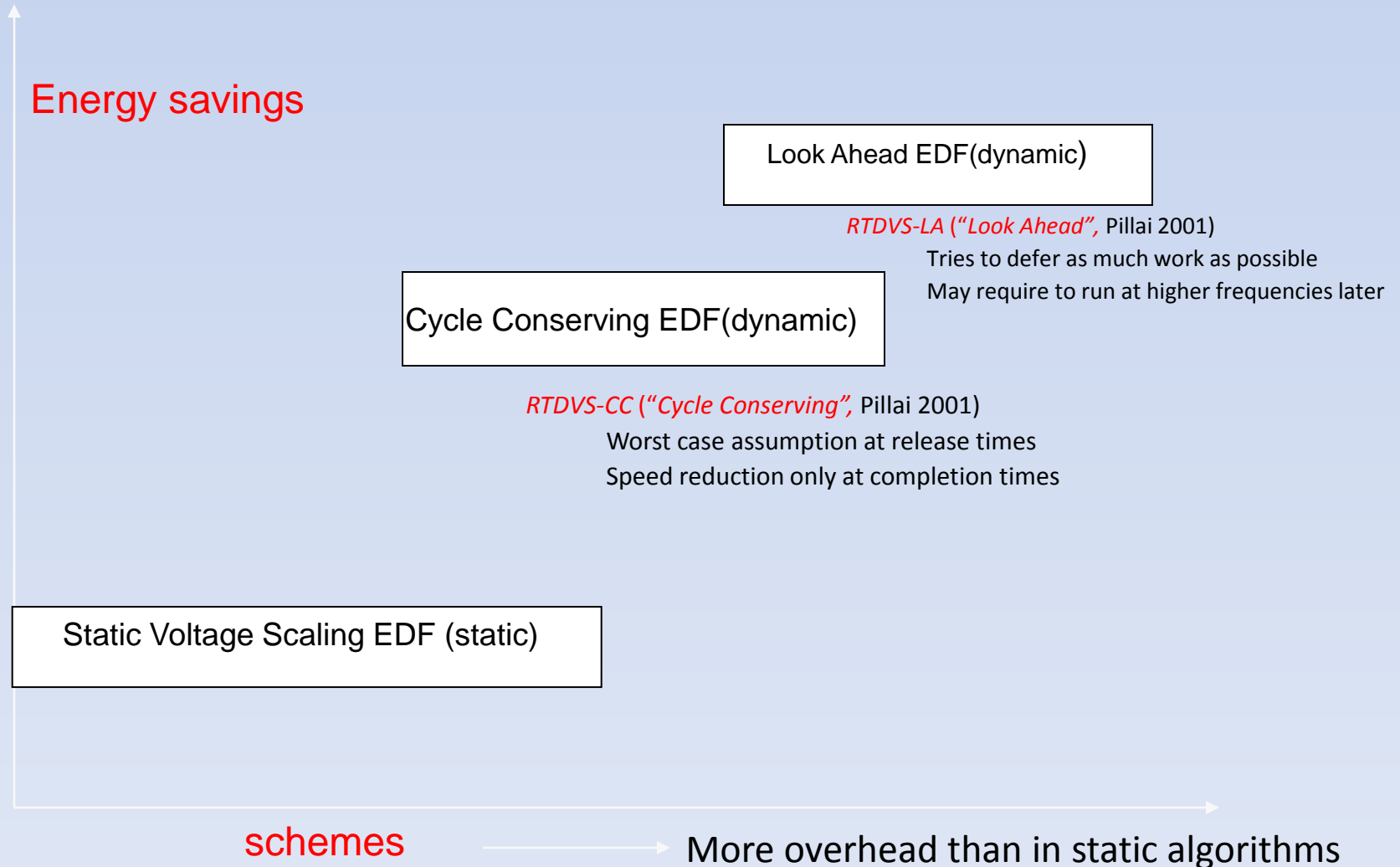
- Task set: $T1 = (3, 6)$ and $T2 = (6, 12)$
- $U = 3/6 + 6/12 = 1 @ F_{\max}$
- What is the “k” at which the task set is still schedulable @ (F_{\max} / k) :
 - Let $K = x$
 - $U = (3*x)/6 + (6*x)/12 = x*(1.0) = 1$
 - $X = 1$, that is $k = 1$
 - Therefore, we should operate at $f = F_{\max}$ in order to meet all the deadlines

Cycle Conserving EDF

Processus	Période	WCET $\alpha = 1$	Instance 1 $\alpha = 1$	Instance 1 $\alpha = 0.75$	Instance 1 $\alpha = 0.5$	Instance 2 $\alpha = 1$	Instance 2 $\alpha = 0.75$	Instance 2 $\alpha = 0.5$
1	8 ms	3 ms	2 ms	2.67 ms	4 ms	1 ms	1.33 ms	2 ms
2	10 ms	3 ms	1 ms	1.33 ms	2 ms	1 ms	1.33 ms	2 ms
3	14 ms	1 ms	1 ms	1.33 ms	2 ms	1 ms	1.33 ms	2 ms



Relative performance



Other Dynamic algorithms

- *DRA-Standard* (Aydin 2001)
 - Earliness of tasks computed through a comparison with the worst-case static schedule
- *DRA-OTE (“One Task Extension”*, Aydin 2001)
 - Further slows down the speed when there is only one task and its WCET does not extend beyond the next event
- For sporadic hard tasks:
 - *DVSST* (Qadi 2003)
 - Variable U keeps track of the total used bandwidth
 - The processor speed is set depending on U

Other dynamic algorithms based on prediction

- *DRA-AGGR* (“*Aggressive*”, Aydin 2001)
 - Speculative assumption that future jobs will present a computational demand as previous ones
 - May have to increase the speed later to guarantee the feasibility of future tasks

Non exhaustive list of references

- [Power reduction techniques for microprocessor systems](#) Vasanth Venkatachalam, Michael Franz , **ACM Computing Surveys (CSUR)**, Volume 37 Issue 3 , Sept. 2005.
- **Power management for energy-aware communication systems, ACM Transactions on Embedded Computing Systems (TECS)**, Volume 2 , Issue 3 (August 2003) , Pages: 431 - 447
- [1] **Real-Time Dynamic voltage scaling for Low-Power Embedded Operating Systems**, P. Pillai and K. G. Shin, in ACM SOSP, pages 89-201, 2001.
- [2] **Intra-task Voltage Scheduling on DVS-Enabled Hard Real-Time Systems**, D. Shin and J. kim, IEEE Design and Test of Computers, March 2001.
- [3] **Enhanced fixed-priority scheduling with (m,k)-firm guarantee**, G. Quan and X. (Sharon) Hu *IEEE Real-Time Systems Symposium*, pp 79-88, 2000.

DESIGN OF AN ENERGY HARVESTING SYSTEM

Energy harvesting

- Energy harvesting is the capture of ambient energy to provide electricity for small and or mobile equipment
- Energy **harvesting** = energy **scavenging**
- can derive energy from a variety of sources including solar, vibration, and temperature variation

Example: At an average power consumption of 100 mW, with 1 cm³ of lithium battery volume, operation during 1 year → not always acceptable

Energy harvesting provides an average of 100 mW/cm³ **indefinitely**

Objectives:

- long life equipment
- Wired power has limitations in out-door applications
- reducing the need for batteries
- Maintenance-free

Design objectives

To execute, **each task requires only processor time** under the previous assumptions.

Our goal was **to meet deadlines requirements** taking into account **demands of processing time** of all the tasks

- **Given the limited availability of the processor** (limited speed and limited number)
- **Energy is assumed to be non limited**

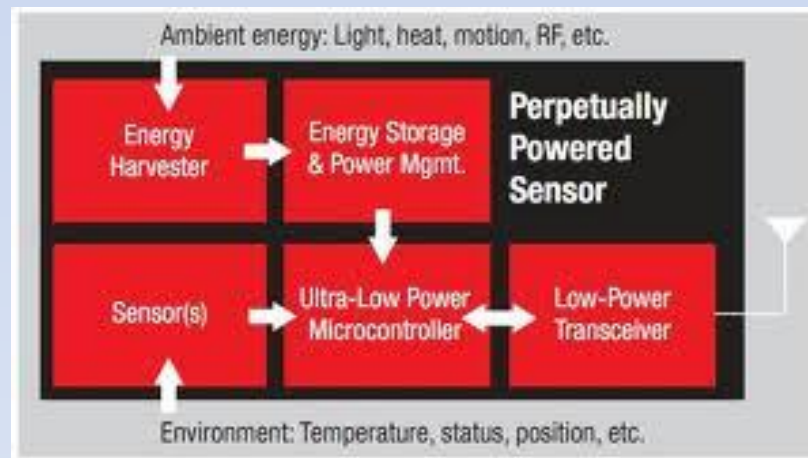
Our goal will **to meet deadlines requirements** taking into account **demands of processing time AND demands of energy**

the energy availability will be limited

Central Issues

- Classical task scheduling, including EDF and RM only accounts for *timing* parameters of the tasks and consequently is not suitable when considering *energy* constraints.
- The problems we have to deal with are:
 - *How to extend classical schedulers so as to suitably exploit both the processor and the available ambient energy ?*
 - *What is the feasibility test ?*
 - *How to choose the size of the energy storage?*
- We will present a scheduling strategy for time critical tasks that run on a computing system that is powered through a renewable energy storage device.

Anatomy of an energy harvesting system



(Source: Texas Instruments)

Characteristics of Wireless Sensors

- In a Wireless Sensor , a task periodically performs the following job:
 - reads data on sensor (temperature, humidity, vehicular movement, lightning condition, pressure, noise levels, ...)
 - stores locally the data and perform quick local data processing
 - and sends the data to a base station

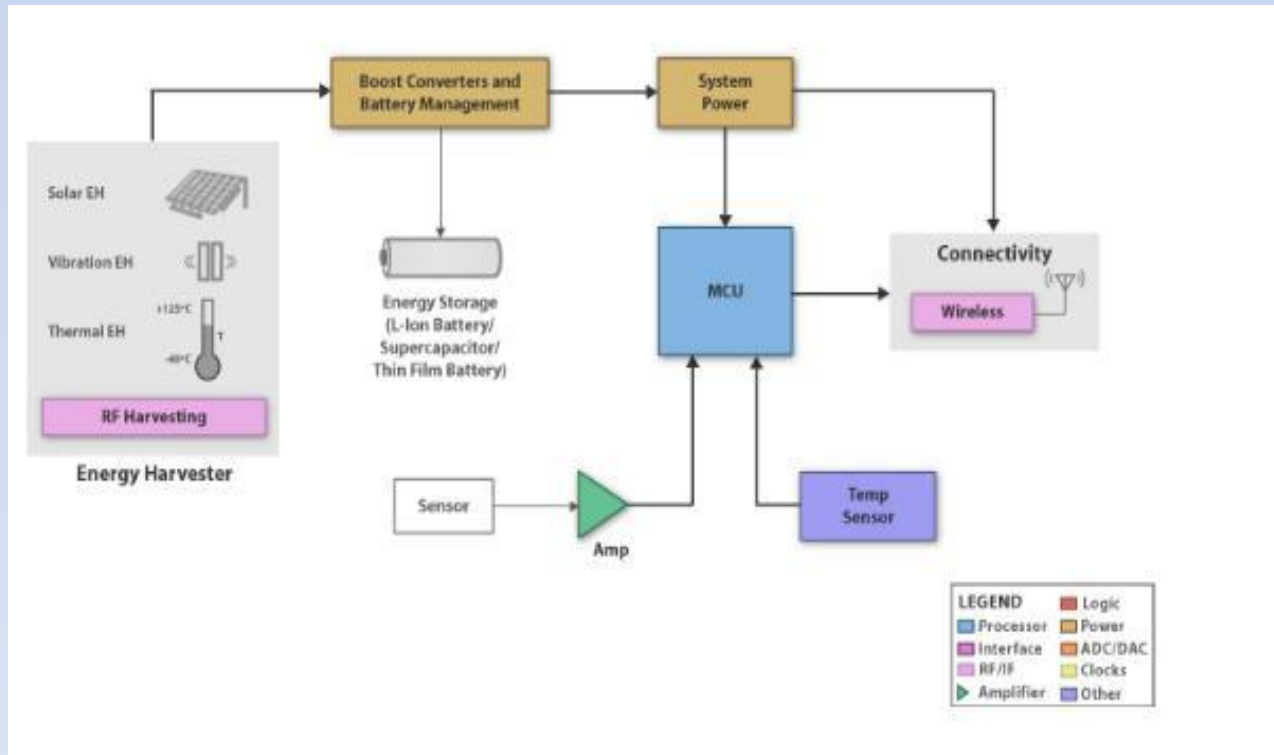
Characteristics:

- limited in power, computational capacities, and memory
- dense deployment of disposable and low-cost sensor nodes
- Sensor node lifetime shows a strong dependence on battery lifetime.

→ Will benefit from energy harvesting



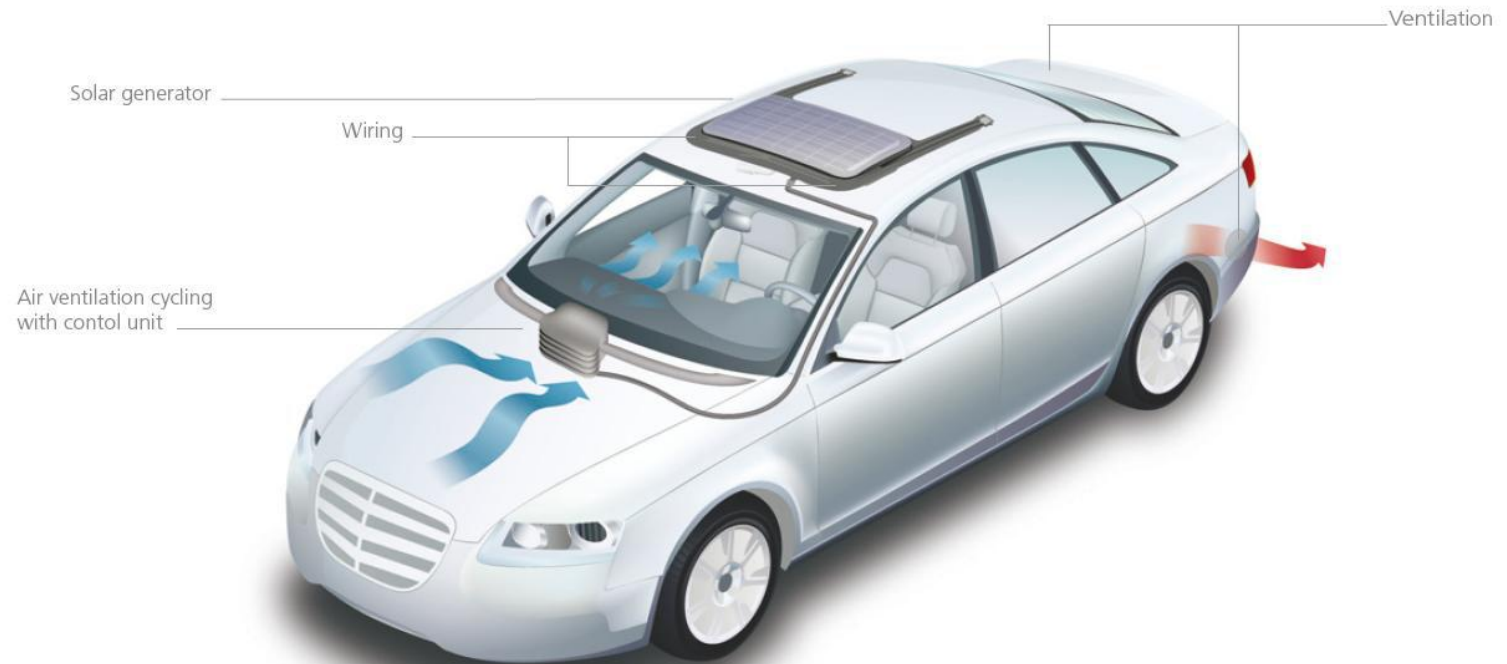
Energy Harvesting block diagram



(Source: Texas Instruments)

Main application sectors

- **Military** (Battlefield surveillance, Reconnaissance of opposing forces and terrain, Battle damage assessment)
- **Environmental** (tracking the movements of animals, Forest fire detection, observation of small size biodiversity, level of air pollution,...)
- **Health** (Telemonitoring of human physiological data,...)



Heterogeneous Multifunction Integration



**Incandescent
lamps 135 mm**



**Integrated
inorganic LED
15mm**



2004

2008

2015

Source: Fiat Research Centre

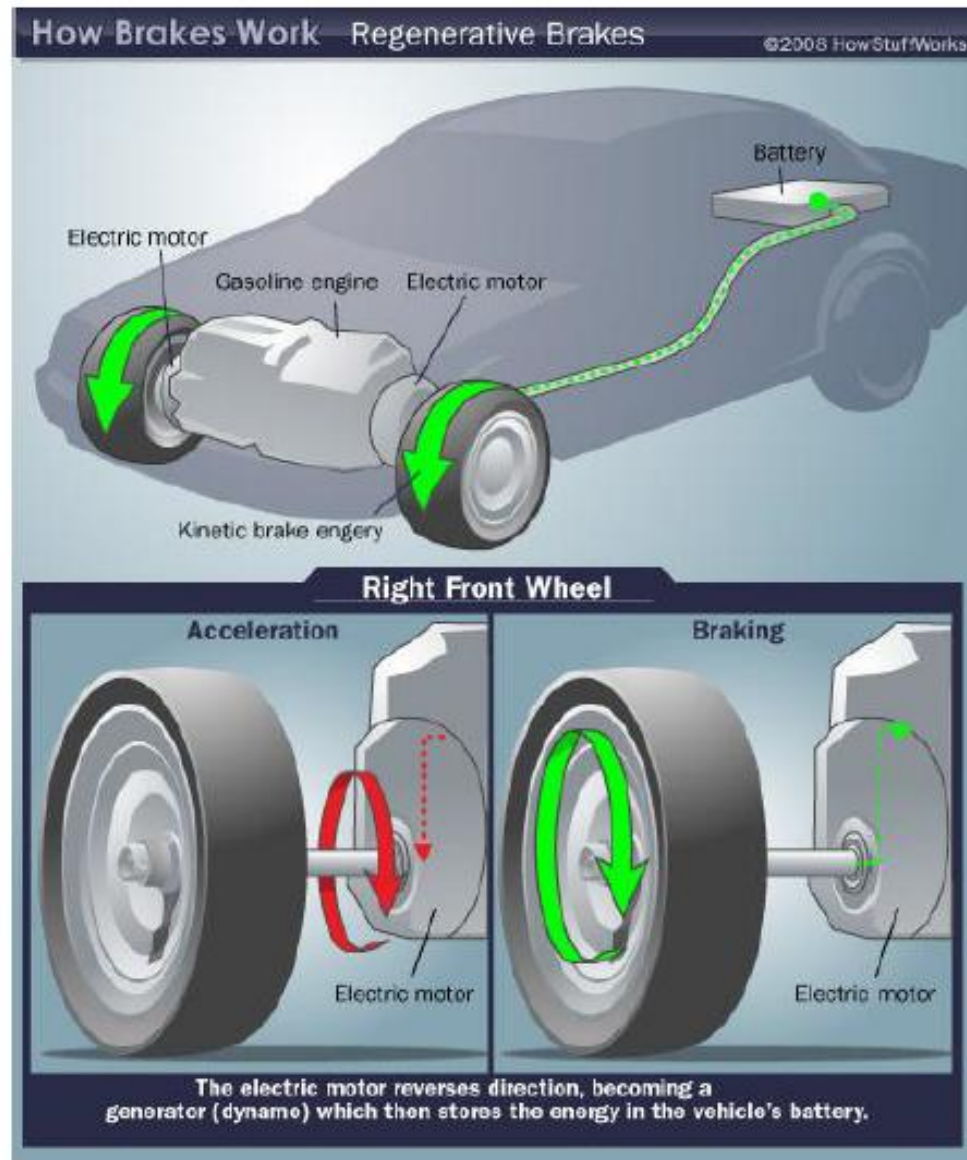
Fig. 3.2 *Thermoelectric generators were first demonstrated by BMW in 2007*



Source: BMW

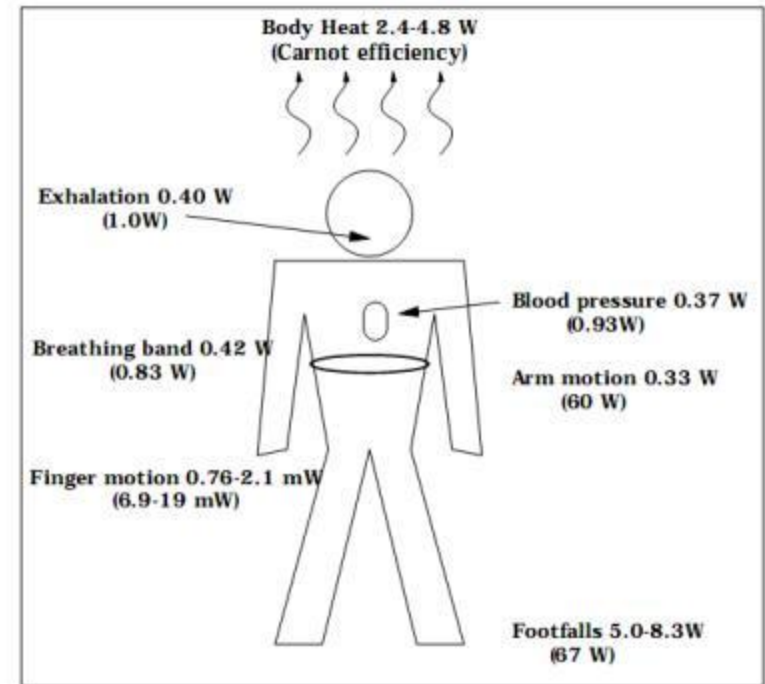
As a side note, continuing its efforts to improve fuel efficiency BMW is also working on the development of turbosteamers as well as brake energy regeneration. The turbosteamer is essentially a miniature gas turbine generator that will use the heat from the exhaust to create steam and in the process, turn a gas turbine. Further development is needed in order to reduce its size but demonstrators back in 2005 increased engine performance by 15%. BMW says the system would only add around 10 to 15kg to the car's weight while providing enough electric power to run all auxiliaries during highway and country road driving conditions. This would free up the engine reducing fuel consumption by up to 10%.

Basic principle of regenerative braking.



Available human power sources

Activity	Kilocal/hr	Watts
sleeping	70	81
lying quietly	80	93
sitting	100	116
standing at ease	110	128
conversation	110	128
eating meal	110	128
strolling	140	163
driving car	140	163
playing violin or piano	140	163
housekeeping	150	175
carpentry	230	268
hiking, 4 mph	350	407
swimming	500	582
mountain climbing	600	698
long distance run	900	1,048
sprinting	1,400	1,630



Possible power recovery from body-centered sources.
Total power for each action is included in parentheses

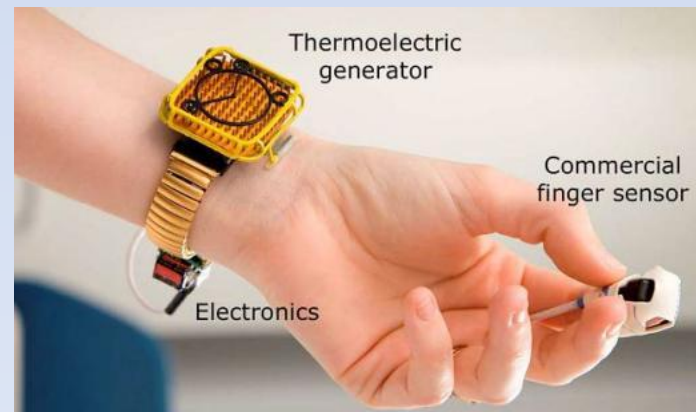
Source: "Human Generated Power for Mobile Electronics," Shad Starner, Joseph A. Paradiso

Available human power sources

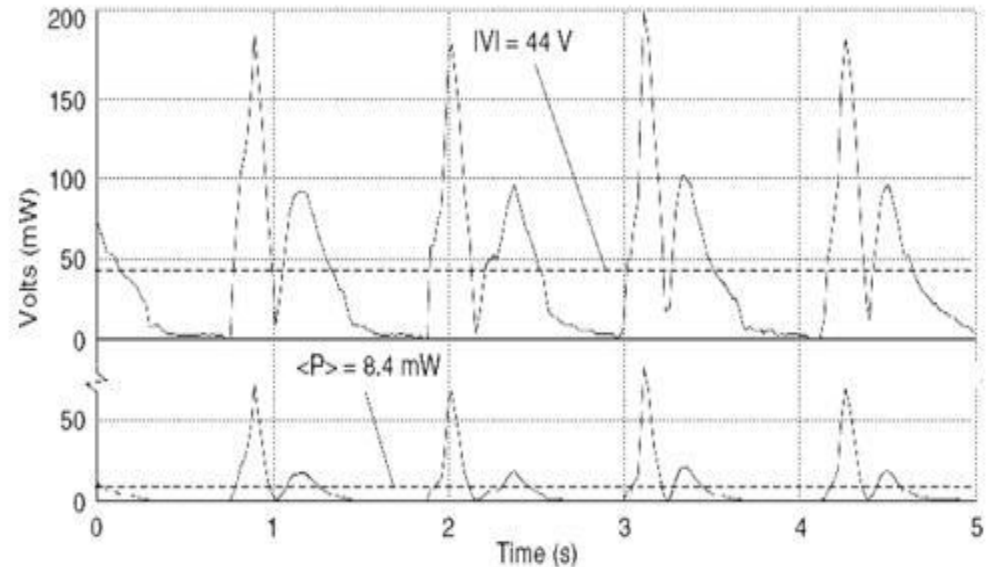
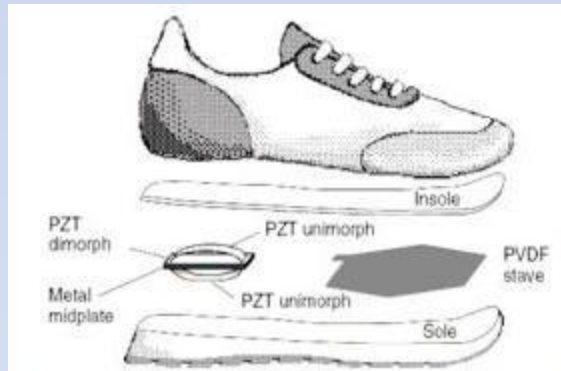
Thermal energy harvesting is usually achieved through the thermoelectric effect. It requires a thermal gradient. It can be achieved in the form of a wearable device.

Output of a “normal” adult body is around 100W

Source: Seiko Thermic watch,
a commercially available timepiece
(www.seikowatches.com)





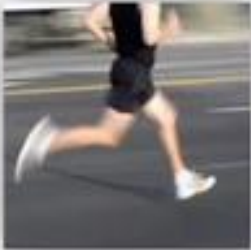

Energy from walking



Example of voltage and power generated

*) Nathan S. Shenck, Joseph A. Paradiso, "Energy Scavenging With Shoe-Mounted Piezoelectrics", Publications IEEE, 2001

Energy harvesting energy sources

				
	PHOTOVOLTAIC	THERMAL	KINETIC	RF
	Outdoor Indoor	Man Machine	Piezoelectric Electrostatic Electromagnetic	GSM WiFi
Harvested Power Potential	Up to 100,000 $\mu\text{W}/\text{cm}^2$ (100 mW/cm^2)	Up to 10,000 $\mu\text{W}/\text{cm}^2$ (10 mW/cm^2)	Up to 1,000 $\mu\text{W}/\text{cm}^2$ (1 mW/cm^2)	Up to 0.1 $\mu\text{W}/\text{cm}^2$

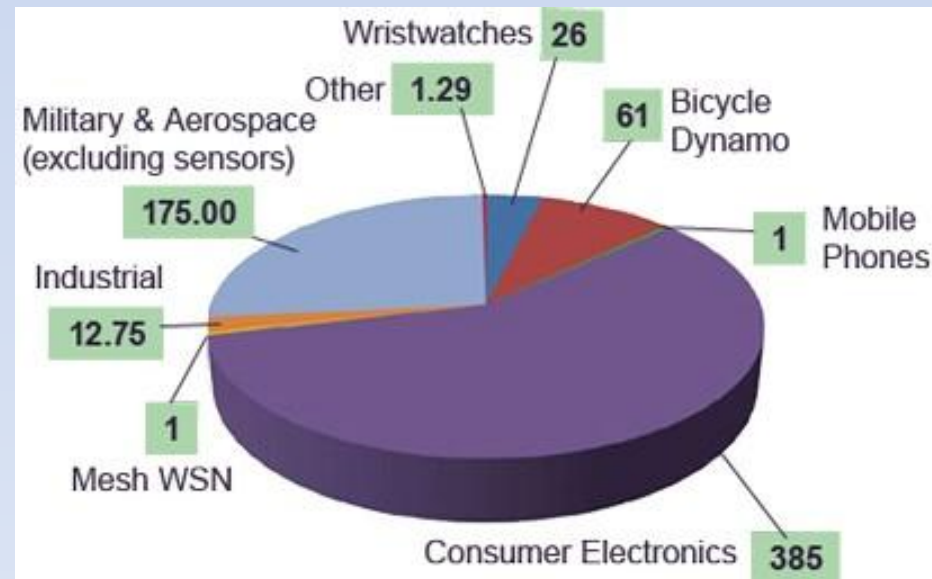
Market Segments using Energy Harvesting

- Mobile phones, Wireless sensor networks, Healthcare Implants,...

250 million wireless sensors
deployed in 2021,
powered by energy harvesting
(1.5 million in 2011)

(Numbers in millions of \$)

(Market=\$4.4 billion by 2021)



Source: IDTechEx (Energy Harvesting & Storage for Electronic Devices, 2011-2012)

Key specifications for energy harvesting



How much energy can I collect?

- Indoor: 50–1000 lux
- Direct Sunlight: ~100,000 lux
- Solar cell will convert 200 lux to ~42 μA

Key Specifications



How much energy can I store?

- Thin film battery rated 0.7 mAh
- Starts charging with 1 μA input
- 30 mA discharge current capability



How much energy is needed for the wireless sensor node?

- **Wireless MCU**
 - Sleep = 35 nA
 - Wireless MCU RTC (including transceiver standby): 800 nA
 - MCU Wake Time: 2 μs
- **Total Circuit**
 - Active mode = 3 μA
 - RX current (max) = 19 mA at -121 dBm
 - TX current (max) = 29 mA at +13 dBm

Source: Silicon Labs

Energy Harvesting becomes reality

New companies with names like *AdaptivEnergy*, *EnOcean*, *Cymbet* and *Perpetuum*, among others are specialized in energy harvesting systems.

And big electronics' manufacturers, such as *Texas Instrument* and *Analog Devic Inc*, are building microcontrollers, digital signal processors and sensors for these applications.

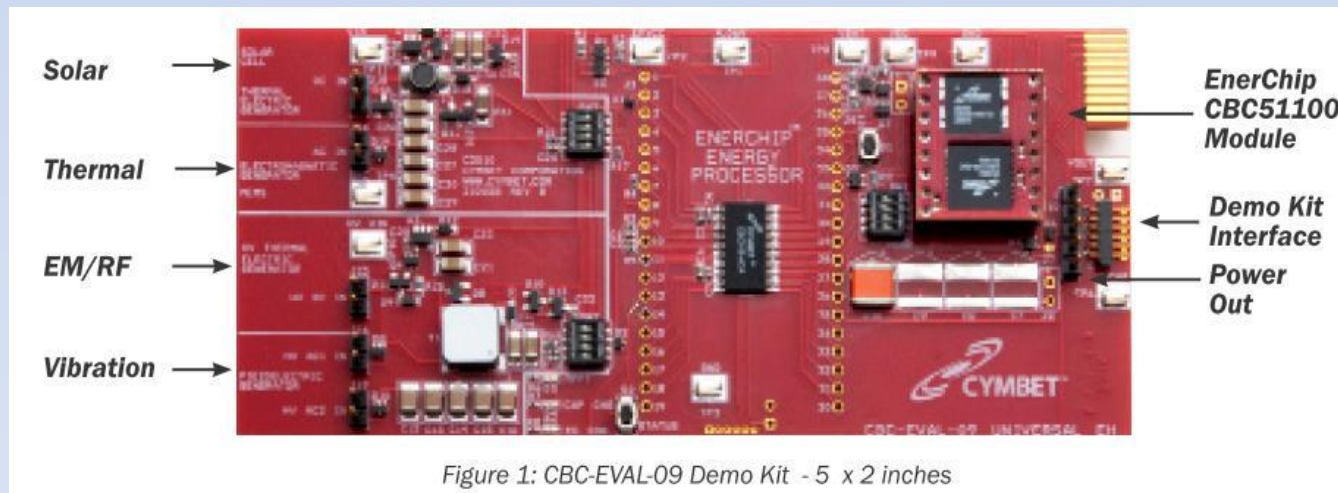
Example of existing renewable energy applications

- Bridges and buildings monitor structural health with battery-less sensors powered by photovoltaic cells
- Farmers check soil moisture with solar powered devices
- Mechanical vibrations can be used for monitoring a vehicle (for example: door lock switches)

Characteristics of these applications: they do not use wires or batteries, while at the same time they save power

Energy Harvesting becomes reality

CBC-EVAL-09 is a universal energy harvesting (EH) evaluation kit that combines any one of multiple EH transducers with the EnerChip™ EP CBC915-ACA Energy Processor and the EnerChip CBC51100 100uAh solid state battery module that has two 50μAh EnerChip solid state batteries connected in parallel.



Source:
Cymbet Corporation

The purpose of this evaluation platform is to enable designers to quickly develop Energy Harvesting applications.

Two kinds of energy harvesting systems

Energy harvesting can be divided into two architectures:

- **Harvest-Use:** Energy is harvested just-in-time for use

Example:

The switches draw power from the mechanical energy that's generated when people press the light switch. Wireless signals are transmitted over an RF frequency

- **Harvest-Store-Use:** Energy is harvested whenever possible and stored for future use.

Energy storage is useful when the harvested energy available is more than the current usage

Example:

During the daytime, energy is used for work and also stored for later use. During night, the stored energy is conservatively used to power the sensor node.

Energy storage

Almost all energy-harvesting systems require an energy storage element or buffer.

Even if the power consumed by an embedded application is so low as to be run directly on power captured or scavenged from the environment, such power would not be produced in a constant way.

This means that an energy storage (**reservoir**) is needed.

Storage is implemented in the form of

- a **capacitor**
- or/ and a rechargeable **battery**,

What kind of energy storage is needed depends greatly on the application.

Thin-film batteries

Some applications require power for only a very short period of time, as short as the RC time constant discharge rate of a capacitor.

Other applications require relatively large amounts of power for an extended duration, which dictates the use of a traditional AA or a rechargeable lithium battery.

Still other applications need the small footprint benefit of the capacitor and the low energy leakage advantage of a traditional battery.

This is where the thin-film batteries are gaining acceptance

	Li-Ion Battery	Thin Film Battery	Super Cap
Recharge cycles	Hundreds	Thousands	Millions
Self-discharge	Moderate	Negligible	High
Charge Time	Hours	Minutes	Sec-minutes
Physical Size	Large	Small	Medium
Capacity	0.3-2500 mAHr	12-1000 μ AHr	10-100 μ AHr
Environmental Impact	High	Minimal	Minimal

Characteristics of typical energy storage options (Courtesy of TI)

(Courtesy of TI)

Framework of an energy harvesting system

A typical energy harvesting system has four components,

- the **Energy source**: ambient source of energy to be harvested
- the **Harvesting architecture** : mechanisms to harness and convert the input ambient energy to electrical energy
- The **Energy storage**: to temporarily store the harvested energy
- and the **Load**: activity that consumes energy .

Weakness of current systems

Most embedded systems constructed to date do not extract power efficiently from the source. Consequently, this approach has disadvantages, such as high costs and large space.

They use a much larger harvester (e.g. solar panel) than necessary to yield the same level of power as a more efficient one.

They rely on a larger, more expensive, higher capacity battery than needed in order to sustain extended operation.

Issues:

- Need for a **methodology** for designing a Real-Time Energy Harvesting System in order to determine
 - ☐The best suitable energy storage unit
 - ☐The best suitable harvester

Energy consumption of WSN

The main task of a sensor node in a sensor field is to detect events, perform quick local data processing, and then transmit the data. Power consumption can hence be divided into three domains:

sensing, communication, and data processing.

A sensor node expends maximum energy in data communication.
This involves both data transmission and reception.

It can be shown that for short-range communication with low radiation power, transmission and reception energy costs are nearly the same.

Energy consumption in data processing

Energy expenditure in data processing is much less compared to data communication.

Example: (*I.F. Akyildiz, W. Su , Y. Sankarasubramaniam, E. Cayirci Computer Networks 38 (2002) 393–422*)

“Energy cost of transmitting 1 KB a distance of 100 m is approximately the same as Energy cost of executing 3 million instructions by a 100 million Instructions per second (MIPS) processor”.

Design objectives

Power management considerations are very different from those of maximizing lifetime i.e.
Power-aware \neq low-power

To execute, **each task requires time and energy.**

Our goal is **to meet deadlines requirements** taking into account:

1. **demands of energy and processing time** of all the tasks
2. **Limited availability of energy and processing time** due to
 - the **energy reservoir** (bounded capacity)
 - the **energy source** (variable and limited charging power)
 - the **processor** (limited speed)

Our primary goal is not to minimize energy consumption.

Design objectives

1) to operate in an *energy neutral mode*, consuming only as much energy as harvested.

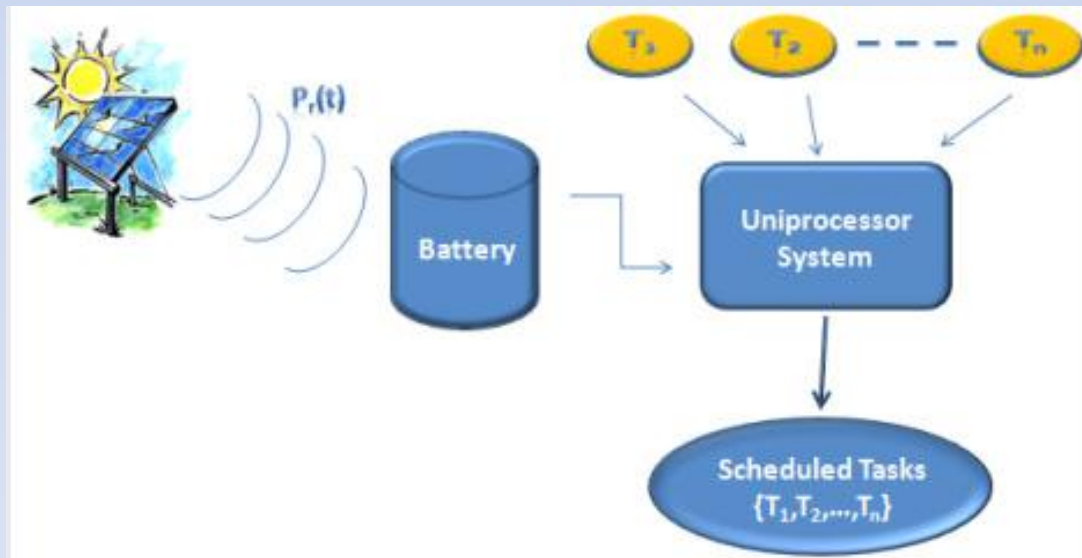
Such a mode of operation raises the possibility of indefinitely long lifetime, limited only by the hardware longevity.

Question: How to operate such that the energy used is always less than the energy harvested?

2) to deal with the real-time tasks (with timing constraints) under the strong variation of energy source with respect to time

Question: What is the maximum performance level that can be supported (in terms of deadline success) ?

Real-time energy harvesting system



Assumptions

Energy Storage

A nominal capacity :At any time, the stored energy is no more than the storage capacity, that is

$$E(t) < E$$

$$E = E_{\max} - E_{\min}$$

- Recharging and discharging may overlap

Energy Source

- Energy produced with fluctuating power $P_r(t)$
- We define the WCCR (Worst Case Charging Rate), namely $P_r(t)$, which is a lower bound on the harvested source power output.

The energy drawn from the environment is **uncontrolled but predictable**

(**for example**, the intensity of direct sunlight cannot be controlled but it is predictable with daily and seasonal patterns and we can use prediction algorithms)

Assumptions

Task Set: each task is characterized by :

- **Worst Case Execution Time** (WCET).
- **Worst Case Energy Consumption** (WCEC)
- the **deadline** and **period** respectively.

Characteristics of the tasks are known in advance.

Preemption is allowed and tasks execute with fluctuating consumption power .

WCEC is not necessary proportional to WCET

Objective: To satisfy timing constraints on software execution

A new terminology

A scheduler S is said to be **optimal** if, given an energy source and an energy reservoir, if S cannot produce a valid schedule then no other one can do.

A scheduler S is said to be **energy-clairvoyant** if it needs the energy profile for the future

A schedule Γ for τ is said to be **valid** if the deadlines of all tasks of τ are met in Γ , starting with a storage fully charged.

A task set τ is said to be **timely-feasible** if there exists a valid schedule for τ without considering its energy constraints.

A task set τ is said to be **feasible** if there exists a valid schedule for τ with considering its energy constraints.

Summary of initial related works

DVS Approach (University of Pittsburg, USA)

- A. Allavena and D. Mossé, Scheduling of Frame based Embedded Systems with Rechargeable Batteries, Workshop on Power Management for Real-Time and Embedded Systems **2001**.
- C. Rusu, R. Melhem and D. Mossé, Multi-version Scheduling in Rechargeable Energy aware Real-time Systems, **ECRTS 2003**

Non DVS Approach

- A. Allavena and D. Mossé, Scheduling of Frame based Embedded Systems with Rechargeable Batteries, Workshop on Power Management for Real-Time and Embedded Systems **2001**.

(Swiss Federal Institute of Technology, ETH Zurich)

- C. Moser, D. Brunelli and L. Benini Real-time Scheduling with Regenerative Energy. **ECRTS 2006**

Related Work (Non DVS)

- Mosse et al. (**2001**) proposed an off-line scheduling algorithm
- Specifically, this algorithm is designed to schedule a set of independent periodic tasks with identical periods (**frame based system**).
- **Problem:** to find a schedule which is able to execute all the tasks within the deadline D , starting with a battery fully charged, ending at the same energy level.
- this approach is based on an unpractical assumption that the harvested energy from the ambient energy source is **constant**.
- **Drawback:**
 - Restrictive model (on energy consumption, task timing parameters)
 - Off-line scheduler

Related work

- An **optimal** real-time scheduling algorithm called **lazy scheduling (LSA)** has been proposed (Moser et al, **2006**)

Assumptions:

- Tasks may be periodic or aperiodic.
- Energy loss insignificant

Properties:

- LSA is a variant of EDF but is an idling energy-clairvoyant scheduler

Reference: Performance Evaluation of Real-Time Scheduling Heuristics for Energy Harvesting Systems , Maryline Chetto, Hui Zhang, **GreenCom'10**

The LSA scheduler

- The processor executes all tasks at **full power** (one frequency);
- The system starts executing a task if the following three conditions are met simultaneously:
 - 1) The task is ready;
 - 2) The task has the earliest deadline among all ready tasks
 - 3) The system is able to keep on running at the maximum power until the deadline of the task.

The LSA scheduler

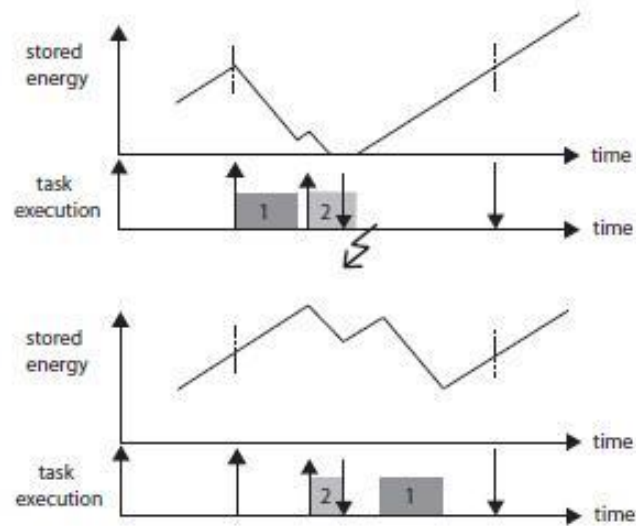


Figure 1. Greedy vs. lazy scheduling

The LSA scheduler

Exercice: Démontrer que la date de démarrage optimale est donnée par s_i

$$s_i^* = d_i - \frac{E_C(a_i) + E_S(a_i, d_i)}{P_{max}}.$$

$$E_S(a_i, s'_i) - C = E_S(a_i, d_i) + (s'_i - d_i)P_{max}.$$

$$s_i = \max(s'_i, s_i^*).$$

The LSA scheduler

Lazy Scheduling Algorithm LSA

Require: maintain a set of indices $i \in Q$ of all ready but not finished tasks J_i

$P_D(t) \leftarrow 0$;

while (true) **do**

$d_j \leftarrow \min\{d_i : i \in Q\}$;

 calculate s_j ;

 process task J_j with power $P_D(t)$;

$t \leftarrow$ current time;

if $t = a_k$ **then** add index k to Q ;

if $t = f_j$ **then** remove index j from Q ;

if $E_C(t) = C$ **then** $P_D(t) \leftarrow P_S(t)$;

if $t \geq s_j$ **then** $P_D(t) \leftarrow P_{max}$;

Drawbacks v.s. advantages of LSA

Advantages:

- Optimality
- A general schedulability test
- On-line
- Energy-clairvoyant

Drawbacks:

- WCEC is proportional to WCET. The ratio is given by the speed of the processor
- Optimality is true only if the processor is able to adjust its consumption power to the source power

→ Proposition of a novel on-line scheduler

ED-H : A new scheduler

Definition 15: The energy demand of a job set τ on the time interval $[t_1, t_2)$ is

$$g(t_1, t_2) = \sum_{t_1 \leq r_k, d_k \leq t_2} E_k \quad (8)$$

Definition 16: The static slack energy of a job set τ on the time interval $[t_1, t_2)$ is

$$SSE_\tau(t_1, t_2) = C + E_p(t_1, t_2) - g(t_1, t_2) \quad (9)$$

ED-H : A new scheduler

Definition 20: The slack energy of a job τ_i at current time t_c is

$$SE_{\tau_i}(t_c) = E(t_c) + E_p(t_c, d_i) - g(t_c, d_i) \quad (13)$$

Definition 21: Let d be the deadline of the active job at current time t_c . The preemption slack energy of a job set τ at t_c is

$$PSE_{\tau}(t_c) = \min_{t_c < r_i < d_i < d} SE_{\tau_i}(t_c) \quad (14)$$

ED-H Algorithm

- **Rule 1:** The EDF priority order is used to select the future running job in $L_r(t_c)$.
- **Rule 2:** The processor is imperatively idle in $[t_c, t_c + 1)$ if $L_r(t_c) = \emptyset$.
- **Rule 3:** The processor is imperatively idle in $[t_c, t_c + 1)$ if $L_r(t_c) \neq \emptyset$ and one of the following conditions is satisfied:
 - 1) $E(t_c) \approx 0$.
 - 2) $PSE_\tau(t_c) \approx 0$
- **Rule 4:** The processor is imperatively busy in $[t_c, t_c + 1)$ if $L_r(t_c) \neq \emptyset$ and one of the following conditions is satisfied:
 - 1) $E(t_c) \approx C$.
 - 2) $ST_\tau(t_c) = 0$
- **Rule 5:** The processor can equally be idle or busy in $[t_c, t_c + 1)$ if $L_r(t_c) \neq \emptyset$, $0 < E(t_c) < C$, $ST_\tau(t_c) > 0$ and $PSE_\tau(t_c) > 0$.

Properties of ED-H Scheduler

- **Two central rules:**
 - 1.The highest priority task is executed as long as there is slack energy.**
 - 2.The processor is idle (for recharging energy) as long as there is slack time or the reservoir is full**
- Slack time $(t) = 0$ means that the processor has to be busy from t so as to guarantee all future tasks before deadline.
- We only waste recharging power when there are no pending tasks and the storage unit is full.

Optimality of ED-H Scheduler

Theorem 1: The ED-H scheduling algorithm is optimal for the RTEH model.

Optimality signifies that ED-H can produce a valid schedule as long as there is no time interval with a length lower than the processor demand and no time interval where the energy demand is greater than the available energy. In other words, any job set which is neither processor-overloaded nor energy-overloaded should be feasible i.e. schedulable by ED-H.

Schedulability test of ED-H Scheduler

Theorem 3: τ is feasible if and only if

$$SST_{\tau} \geq 0 \text{ and } SSE_{\tau} \geq 0$$

In other terms, Theorem 3 states that τ is feasible if and only if τ is time-feasible and energy-feasible.

Illustrative example

Let the periodic task set given by

$\tau_1 = (3, 6, 9, 8)$, $\tau_2 = (3, 8, 12, 8)$ and $\tau_3 = (3, 12, 18, 8)$

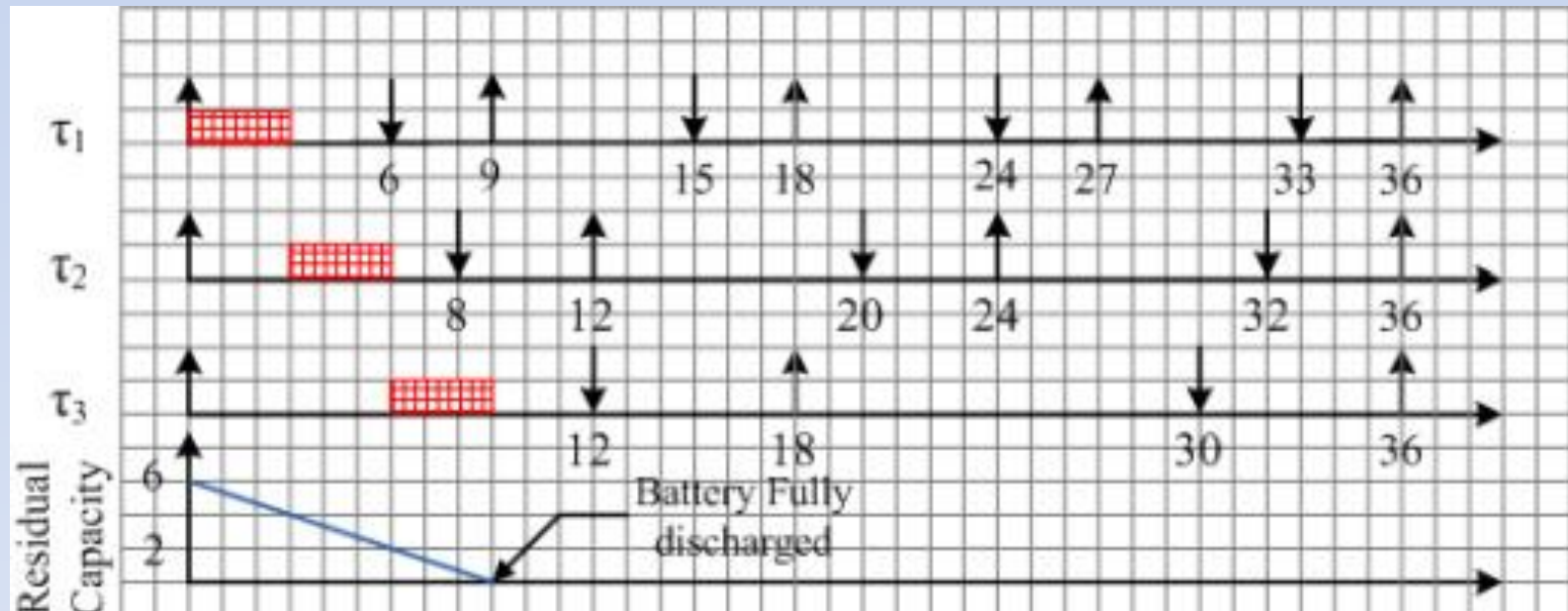
We assume that the energy storage capacity is $E = 6$.

For simplicity, the rechargeable power, P_r is constant along time and is equal to 2.

We note that $U_p = 0.75$ and $U_e = 2$.

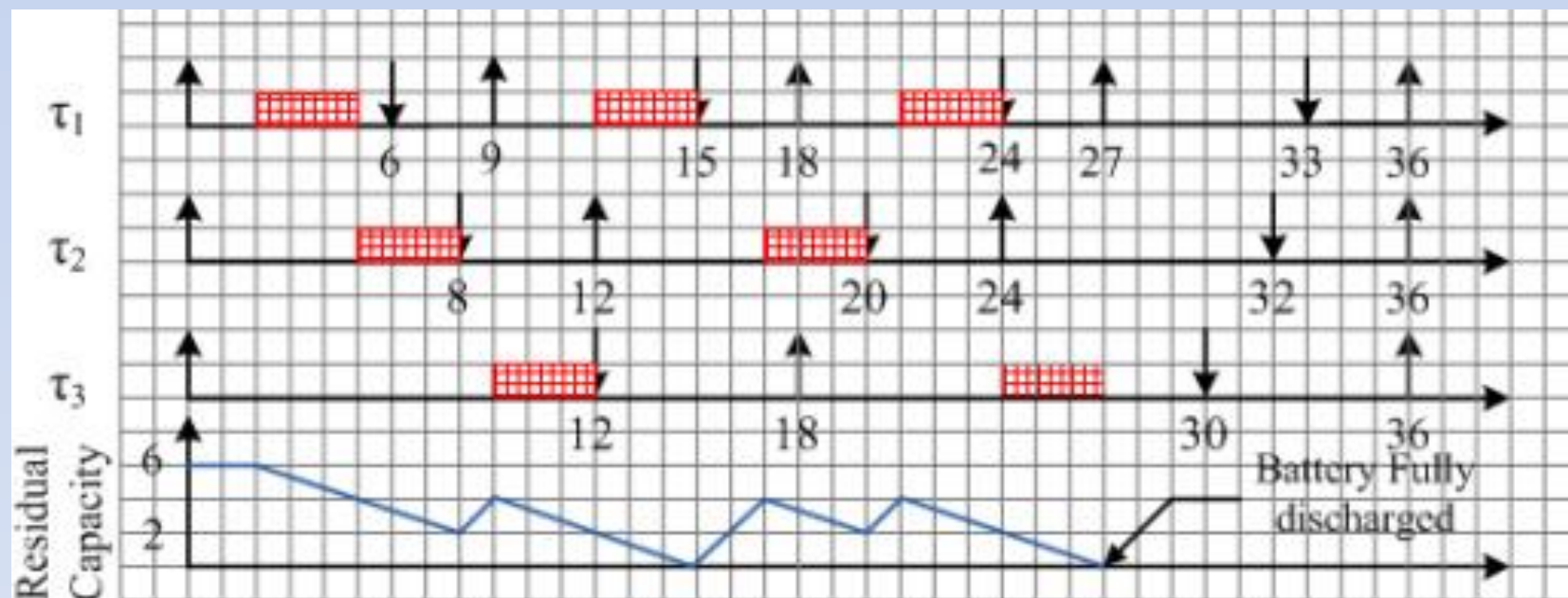
As $U_p \leq 1$ and $U_e \leq P_r$ the necessary feasibility condition related to time and energy are satisfied.

Illustrative example



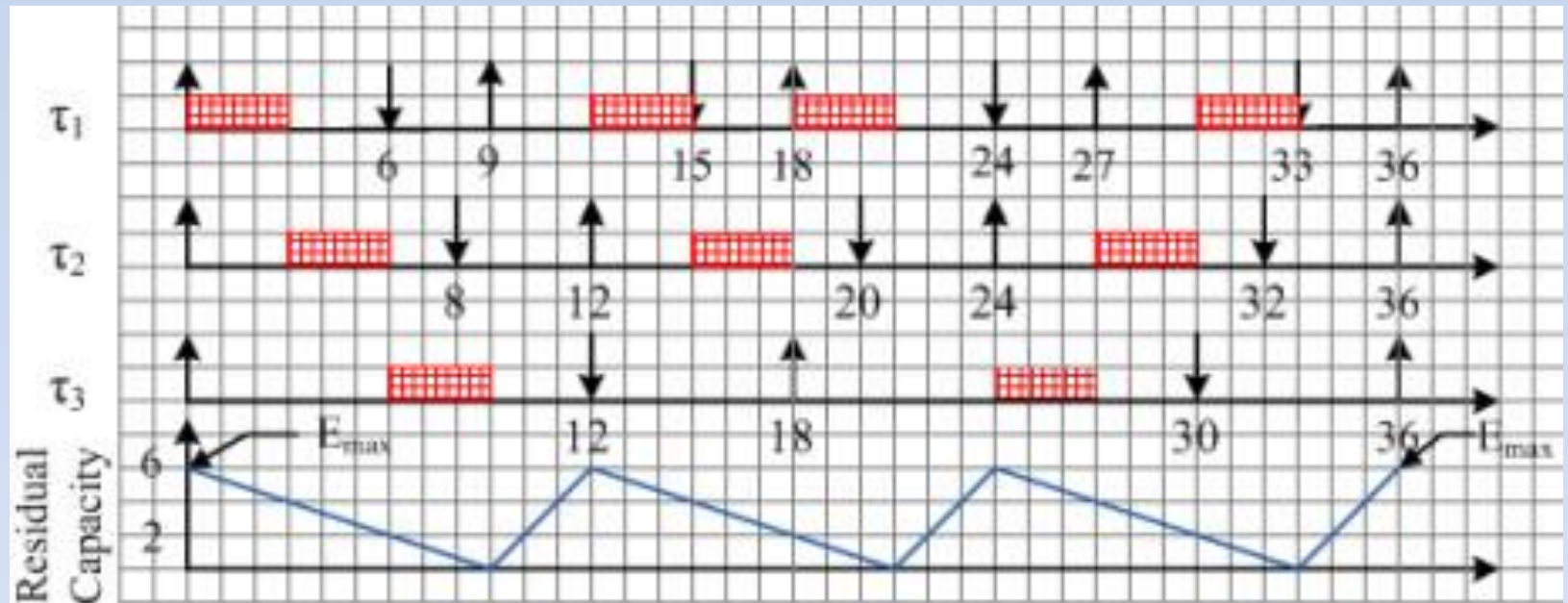
Weakness of the EDS (ASAP) scheduling strategy

Illustrative example



Weakness of the EDL (ALAP) scheduling strategy

Illustrative example



ED-H scheduling strategy

Comments on ED-H

ED-H has been designed to schedule :

- any set of time critical tasks (periodic or not)
- given any energy source profile (with constant production power or not)
- and given an energy storage unit with limited capacity.

BUT

ED-H is a **CLAIRVOYANT scheduler**:

- Energy clairvoyant (must know future energy profile)
- Processing clairvoyant (must know future processor demand)

RTOS Requirements

Adding Energy harvesting aware features to a RTOS requires the knowledge of:

- the current available energy
- the estimation of the future harvested power to tune the system behavior
- the energy requirement of every task.

Here are the main technological obstacles !!!