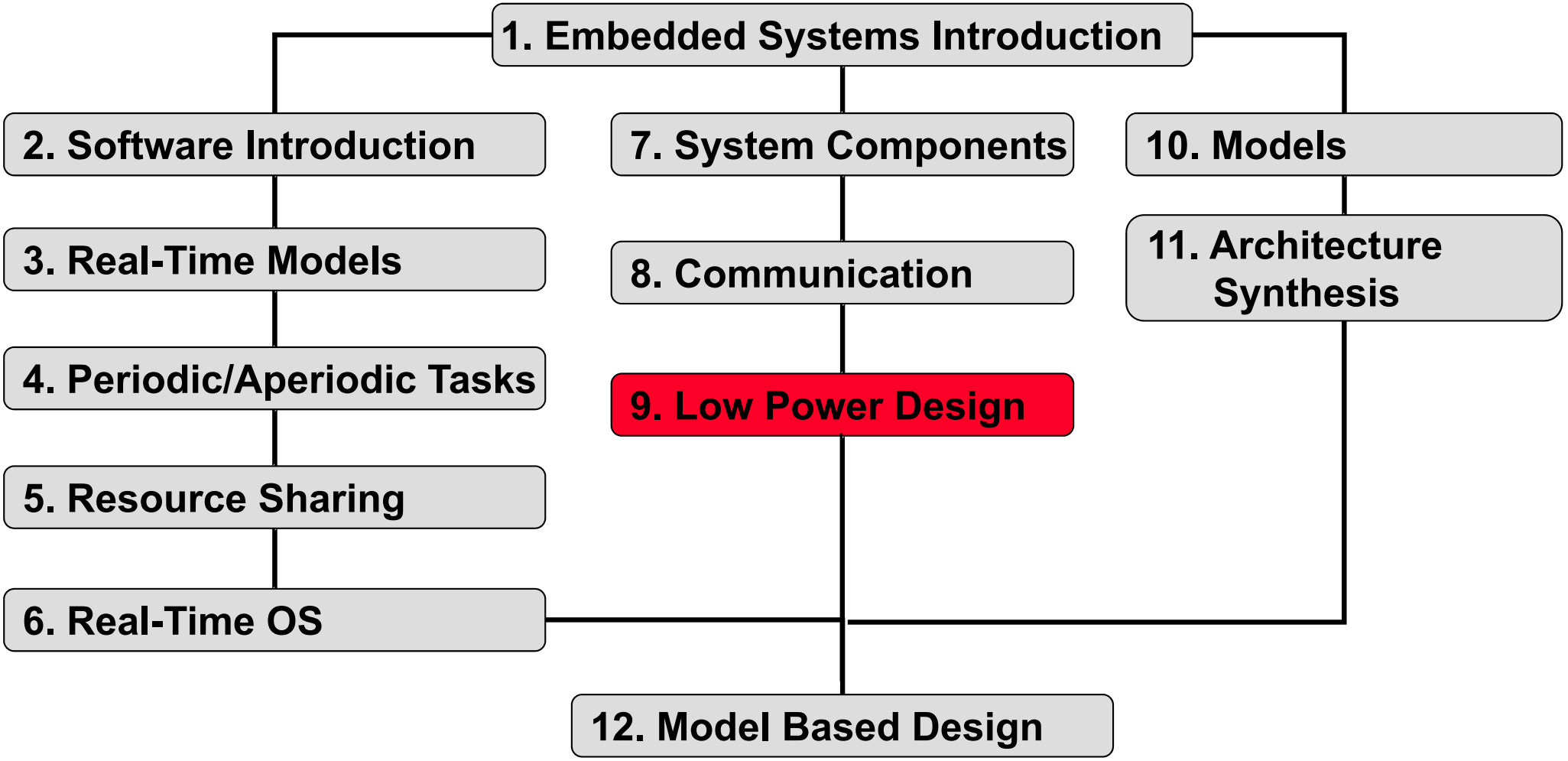
Embedded Systems

9. Low Power Design

Lothar Thiele

Contents of Course



Software and Programming

Processing and Communication

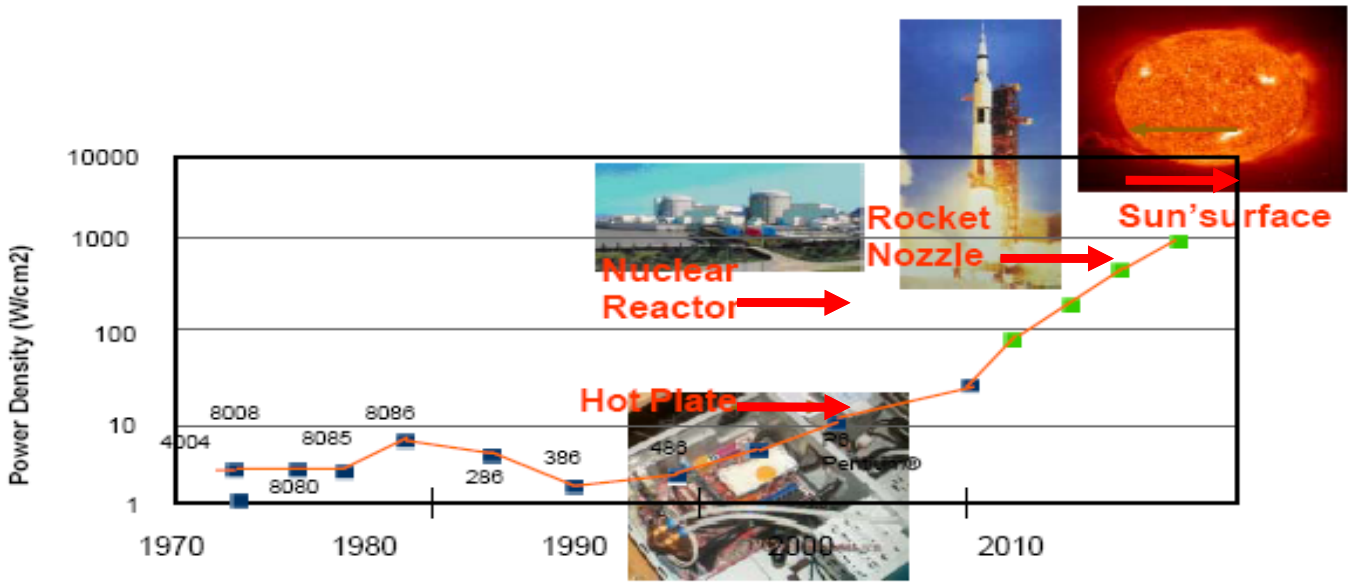
Hardware

Topics

- ***General Remarks***
- Power and Energy
- Basic Techniques
  - Parallelism
  - VLIW (parallelism and reduced overhead)
  - Dynamic Voltage Scaling
  - Dynamic Power Management

Power and Energy Consumption

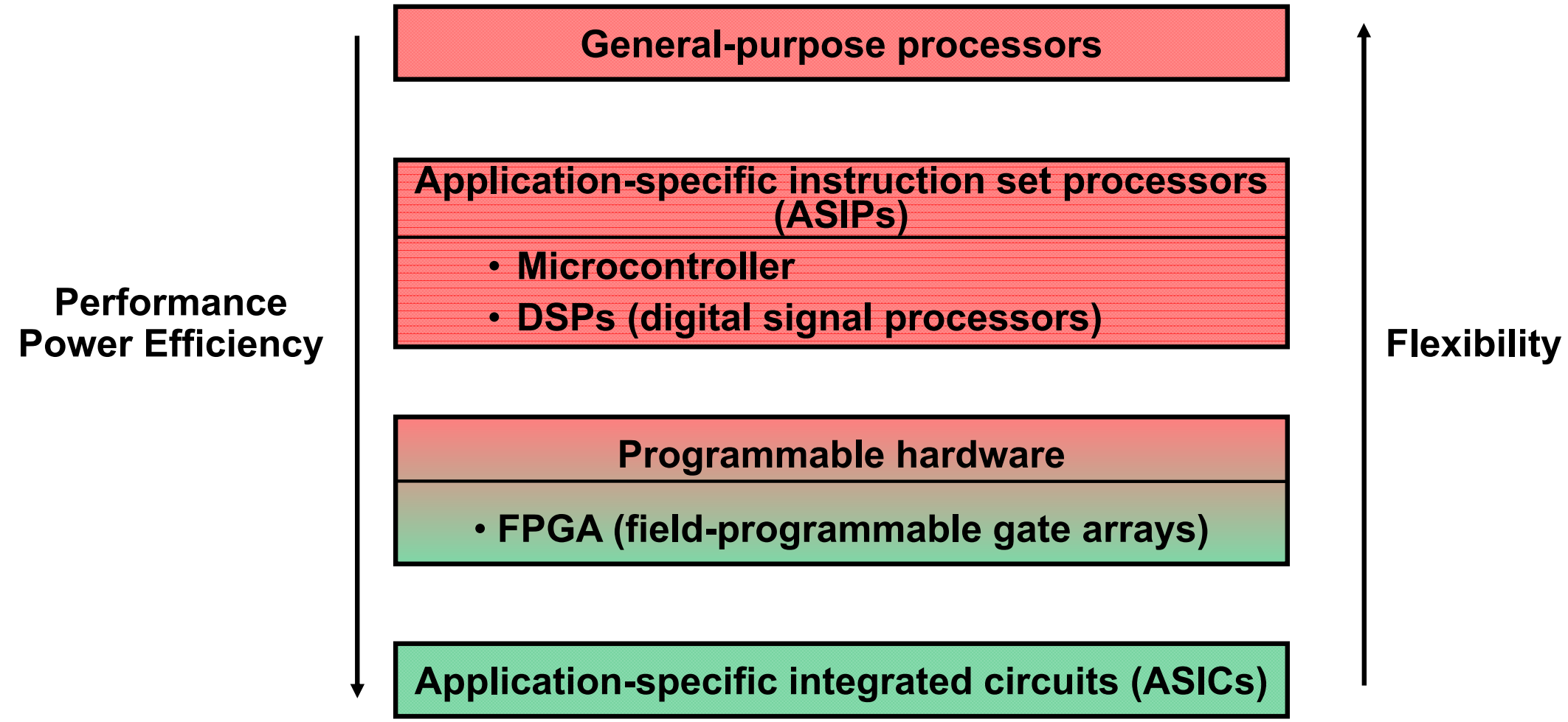
Need for efficiency (power and energy):



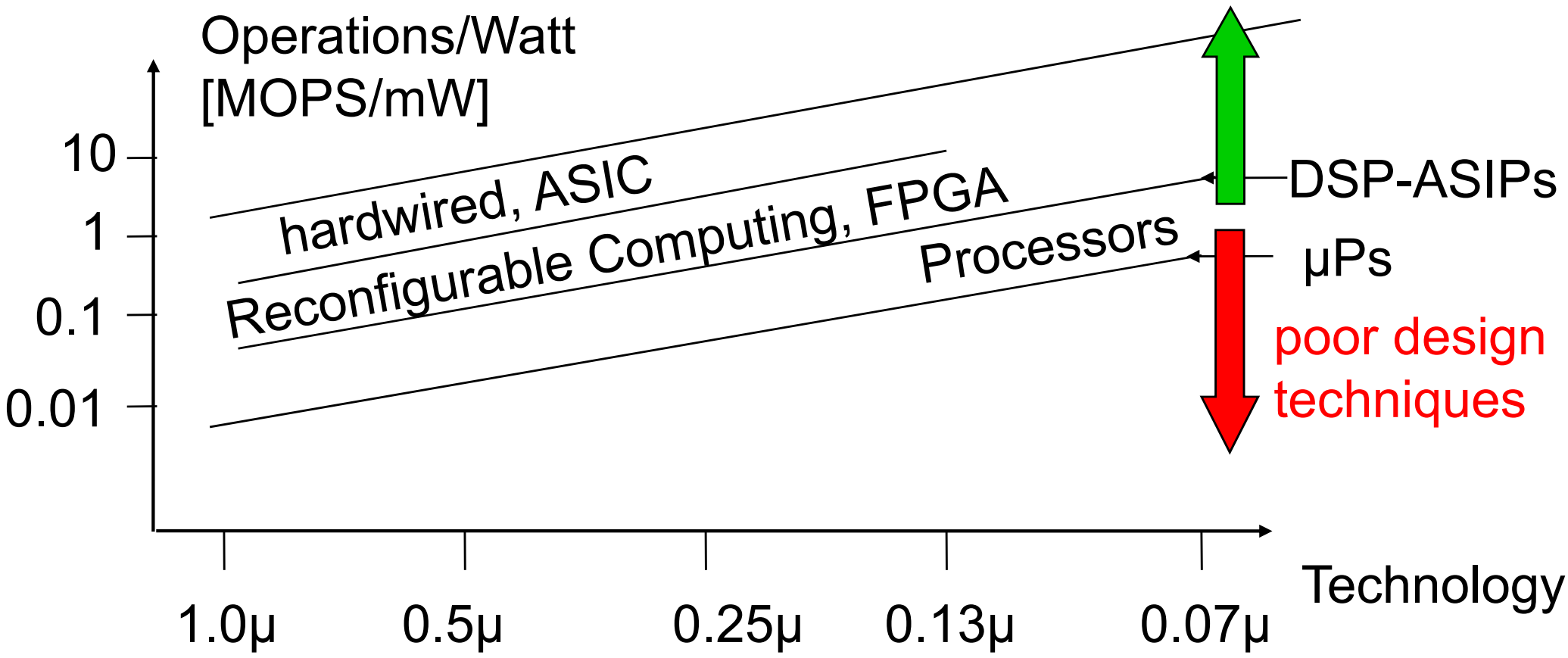
„Power is considered as the most important constraint in embedded systems." [in: L. Eggermont (ed): Embedded Systems Roadmap 2002, STW]

"Power demands are increasing rapidly, yet battery capacity cannot keep up." [in Diztel et al.: Power-Aware Architecting for data-dominated applications, 2007, Springer]

Implementation Alternatives

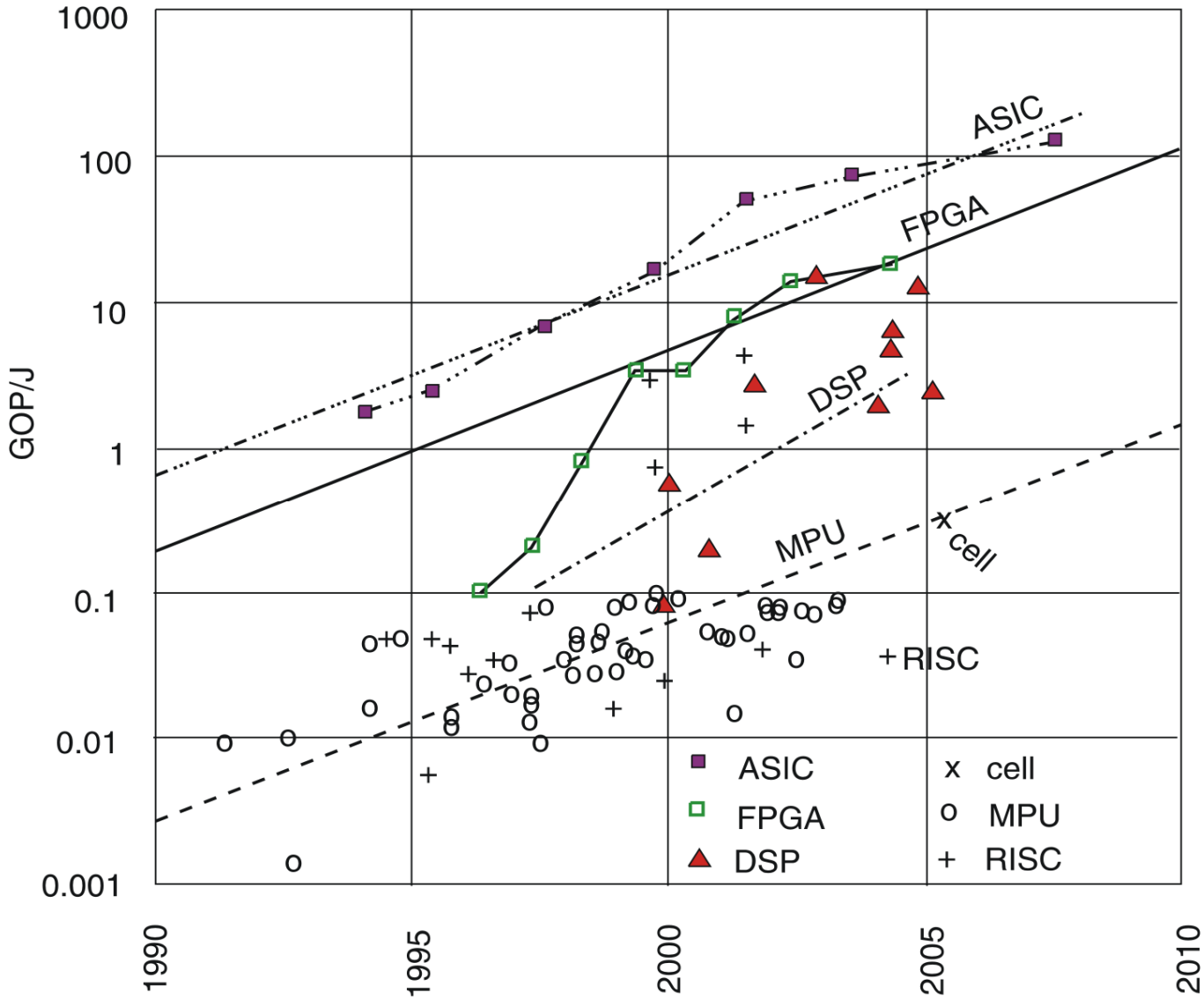


The Power/Flexibility Conflict



Necessary to ***optimize HW and SW.***
Use ***heterogeneous architectures***.
Apply ***specialization techniques***.

Energy Efficiency



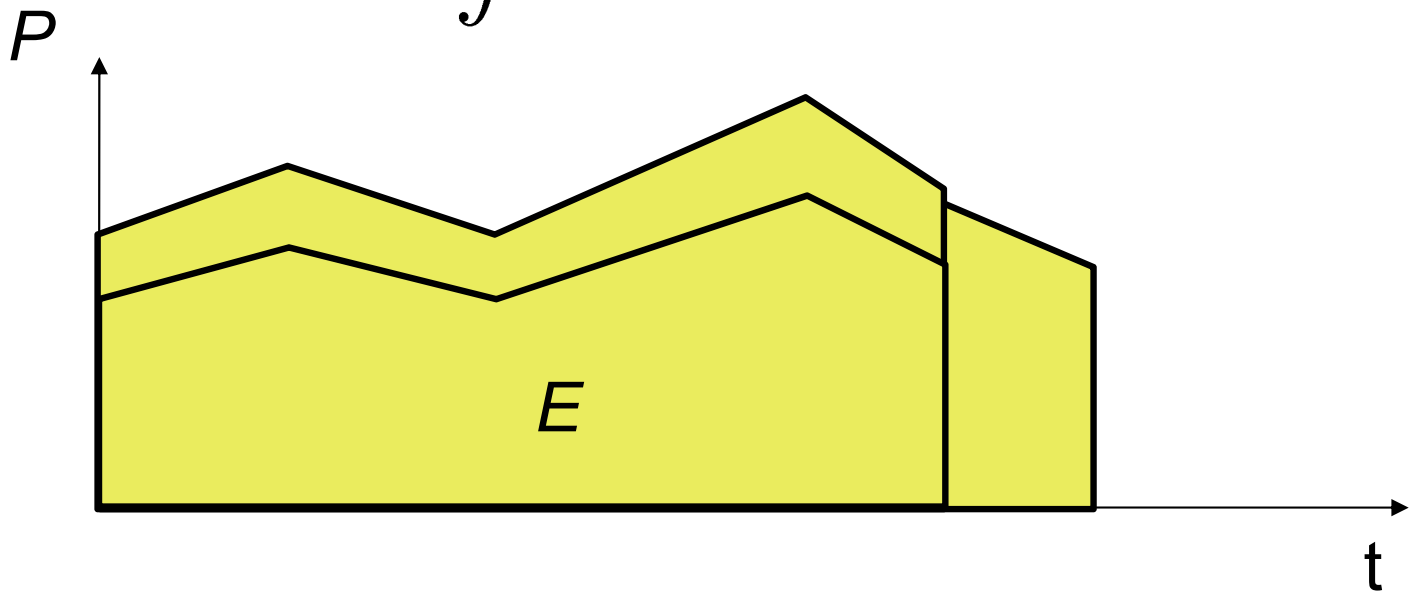
© Hugo De Man,
IMEC, Philips, 2007

Topics

- General Remarks
- ***Power and Energy***
- Basic Techniques
  - Parallelism
  - VLIW (parallelism and reduced overhead)
  - Dynamic Voltage Scaling
  - Dynamic Power Management

Power and Energy are Related

$$E = \int P(t)dt$$

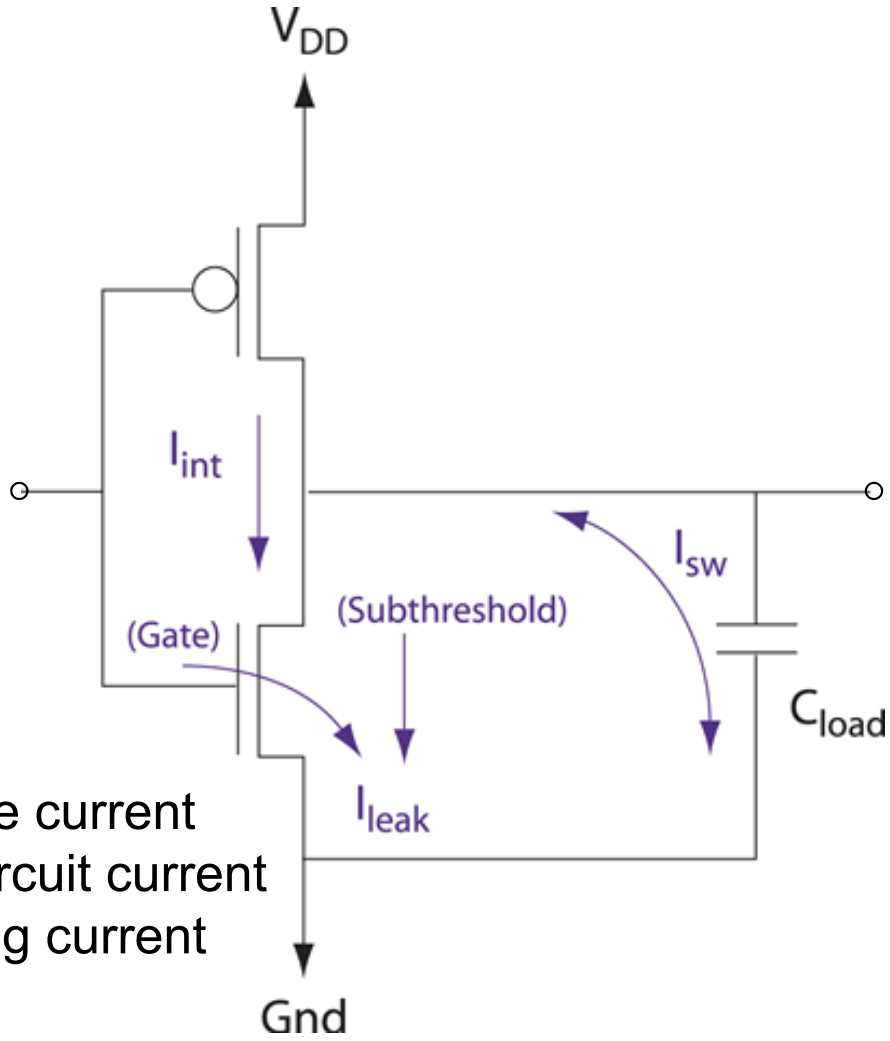


In many cases, faster execution also means less energy, but the opposite may be true if power has to be increased to allow faster execution.

Low Power vs. Low Energy

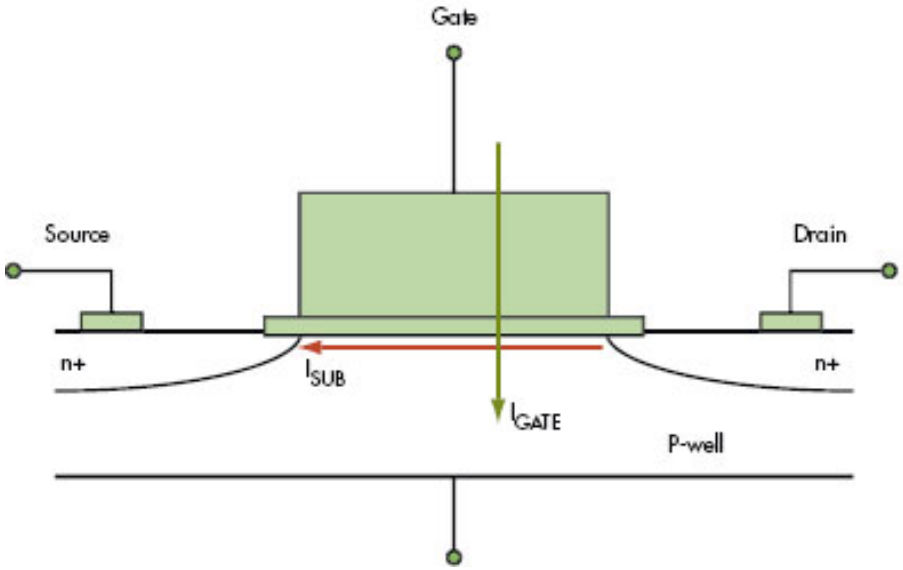
- Minimizing the ***power consumption*** is important for
  - the design of the power supply
  - the design of voltage regulators
  - the dimensioning of interconnect
  - cooling (short term cooling)
    - high cost (estimated to be rising at $1 to $3 per Watt for heat dissipation [Skadron et al. ISCA 2003])
    - limited space
- Minimizing the ***energy consumption*** is important due to
  - restricted availability of energy (mobile systems)
  - limited battery capacities (only slowly improving)
  - very high costs of energy (solar panels, in space)
  - long lifetimes, low temperatures

Power Consumption of a CMOS Gate



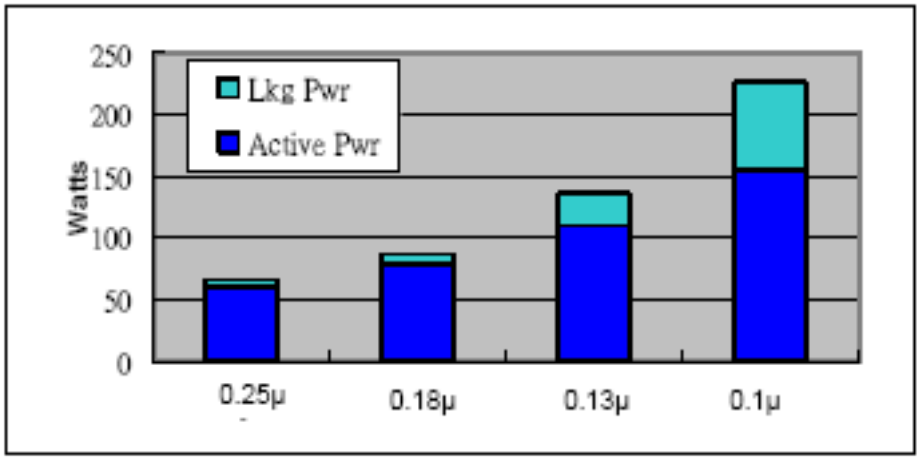
I_{leak} : leakage current
I_{int} : short circuit current
I_{sw} : switching current

subthreshold and gate-oxide leakage



Power Consumption of CMOS Processors

- ***Main sources***:
  - Dynamic power consumption
    - charging and discharging capacitors
  - Short circuit power consumption
    - short circuit path between supply rails during switching
  - Leakage
    - leaking diodes and translators
    - becomes one of the major factors due to shrinking feature sizes in semiconductor technology



(Micro32 Keynotes by Fred Pollack)

Dynamic Voltage Scaling (DVS)

Power consumption of CMOS circuits (ignoring leakage):

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

V_{dd} : supply voltage
α : switching activity
C_L : load capacity
f : clock frequency

Delay for CMOS circuits:

$$\tau \sim C_L \frac{V_{dd}}{(V_{dd} - V_T)^2}$$

V_{dd} : supply voltage
V_T : threshold voltage
$V_T \ll V_{dd}$

Decreasing V_{dd} reduces P quadratically (f constant).

The gate delay increases reciprocally with decreasing V_{dd} .

Maximal frequency f_{max} decreases linearly with decreasing V_{dd} .

Potential for Energy Optimization: DVS

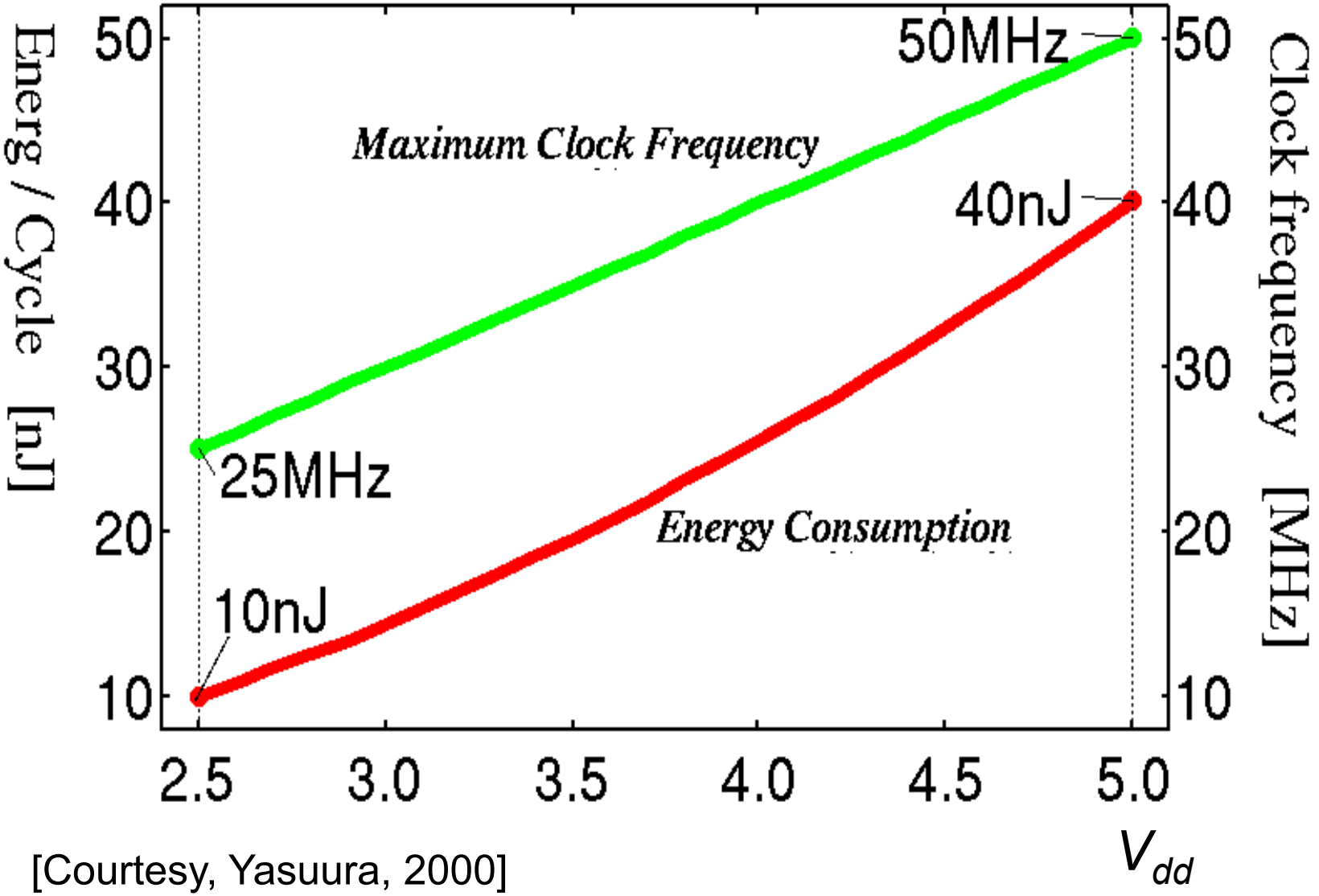
$$P \sim \alpha C_L V_{dd}^2 f$$

$$E \sim \alpha C_L V_{dd}^2 f t = \alpha C_L V_{dd}^2 \ (\#\text{cycles})$$

Saving energy for a given task:

- Reduce the supply voltage V_{dd}
- Reduce switching activity α
- Reduce the load capacitance C_L
- Reduce the number of cycles *#cycles*

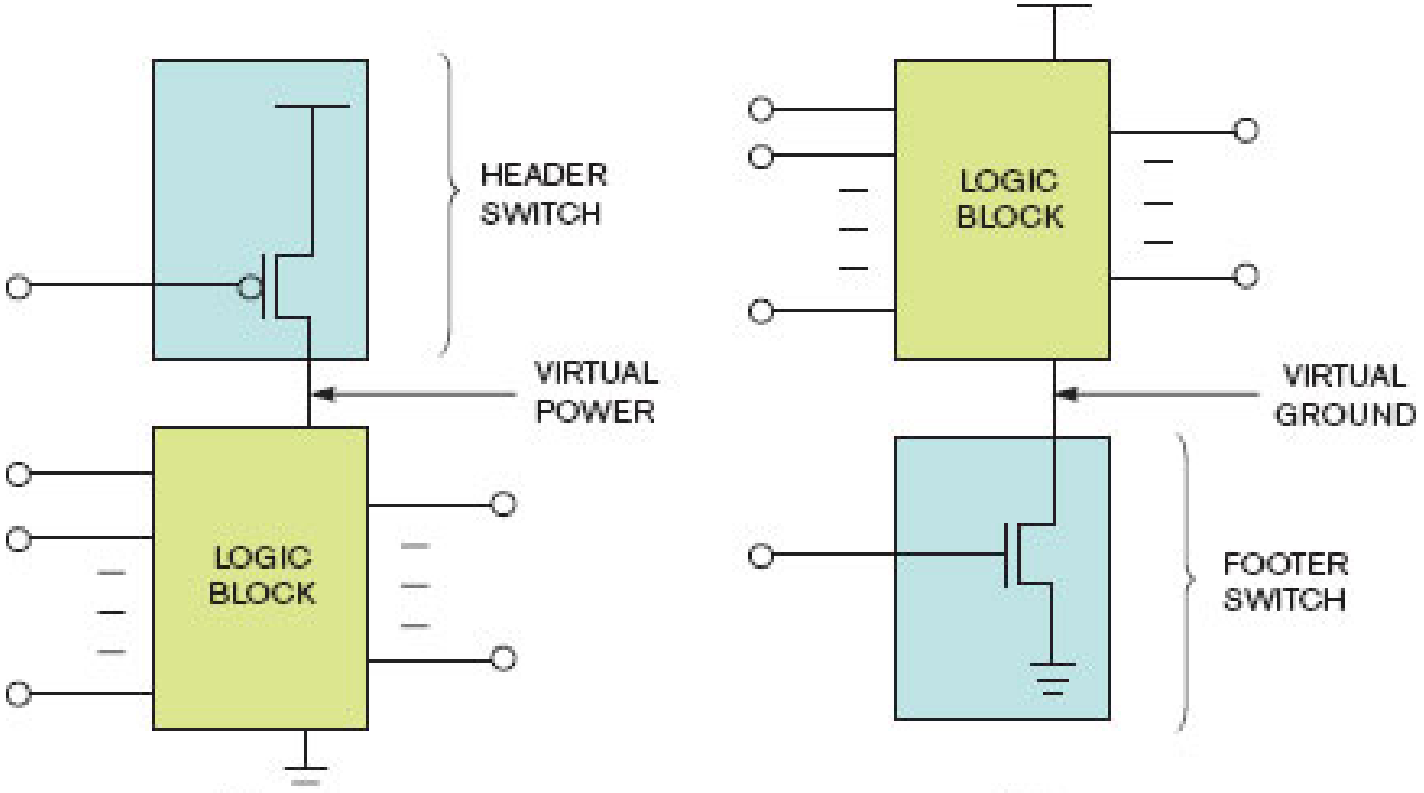
Example: Voltage Scaling



[Courtesy, Yasuura, 2000]

Power Supply Gating

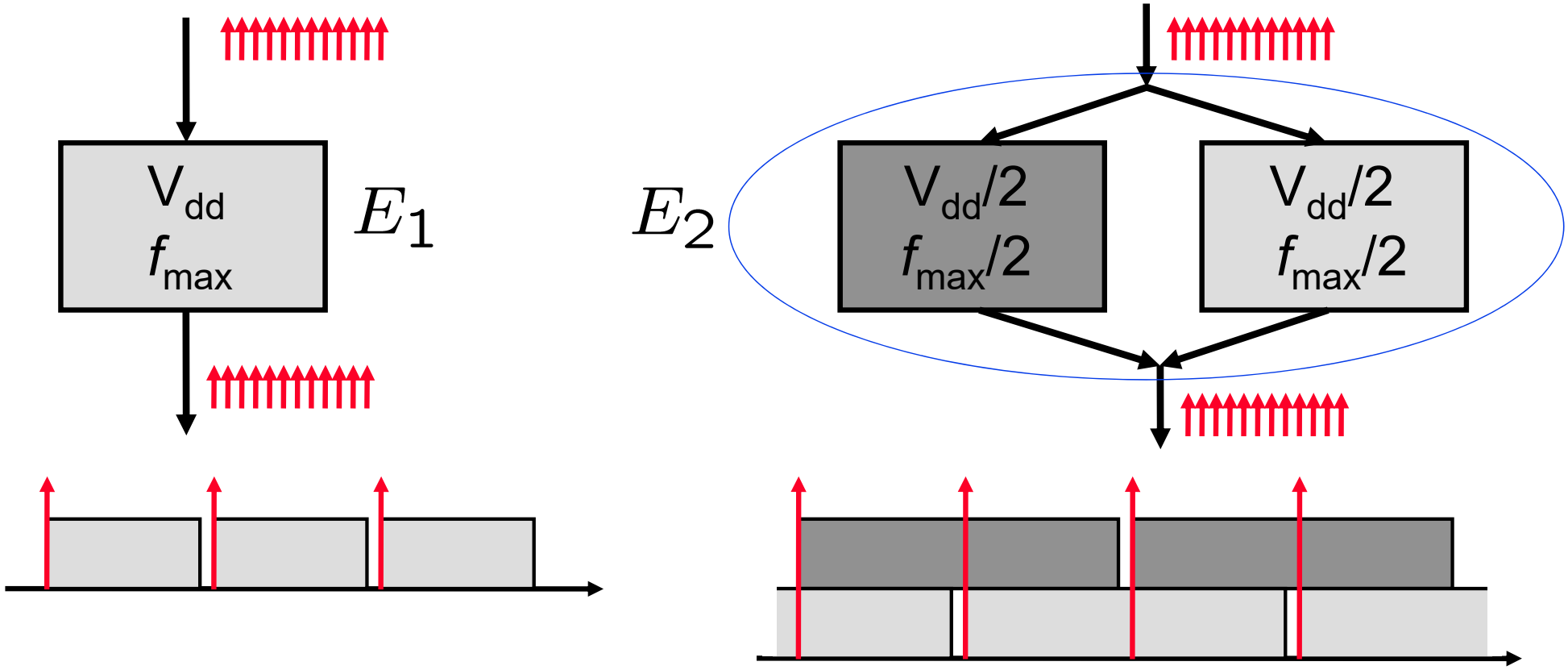
- Power gating is one of the most effective ways of minimizing static power consumption (leakage)
  - Cut-off power supply to inactive units/components
  - Reduces leakage



Topics

- General Remarks
- Power and Energy
- Basic Techniques
  - ***Parallelism***
  - VLIW (parallelism and reduced overhead)
  - Dynamic Voltage Scaling
  - Dynamic Power Management

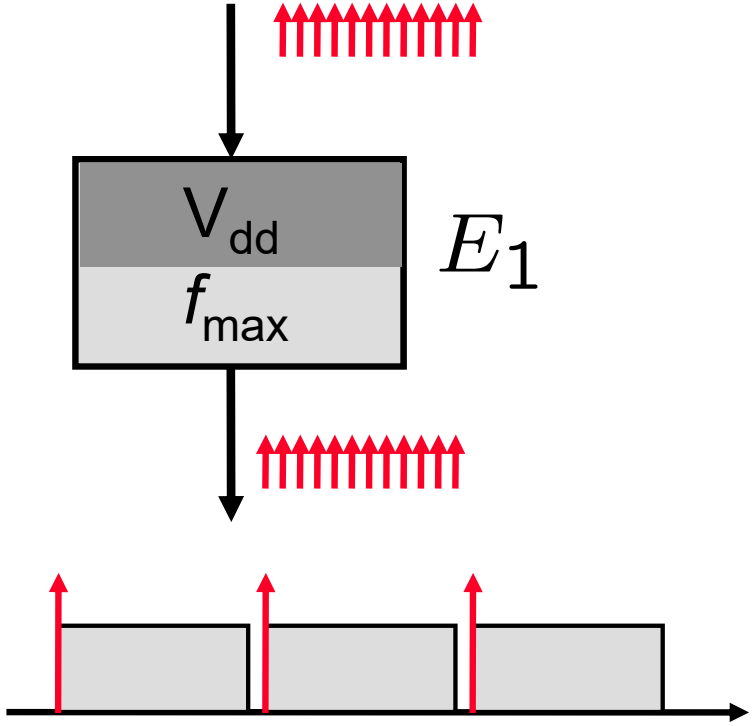
Use of Parallelism

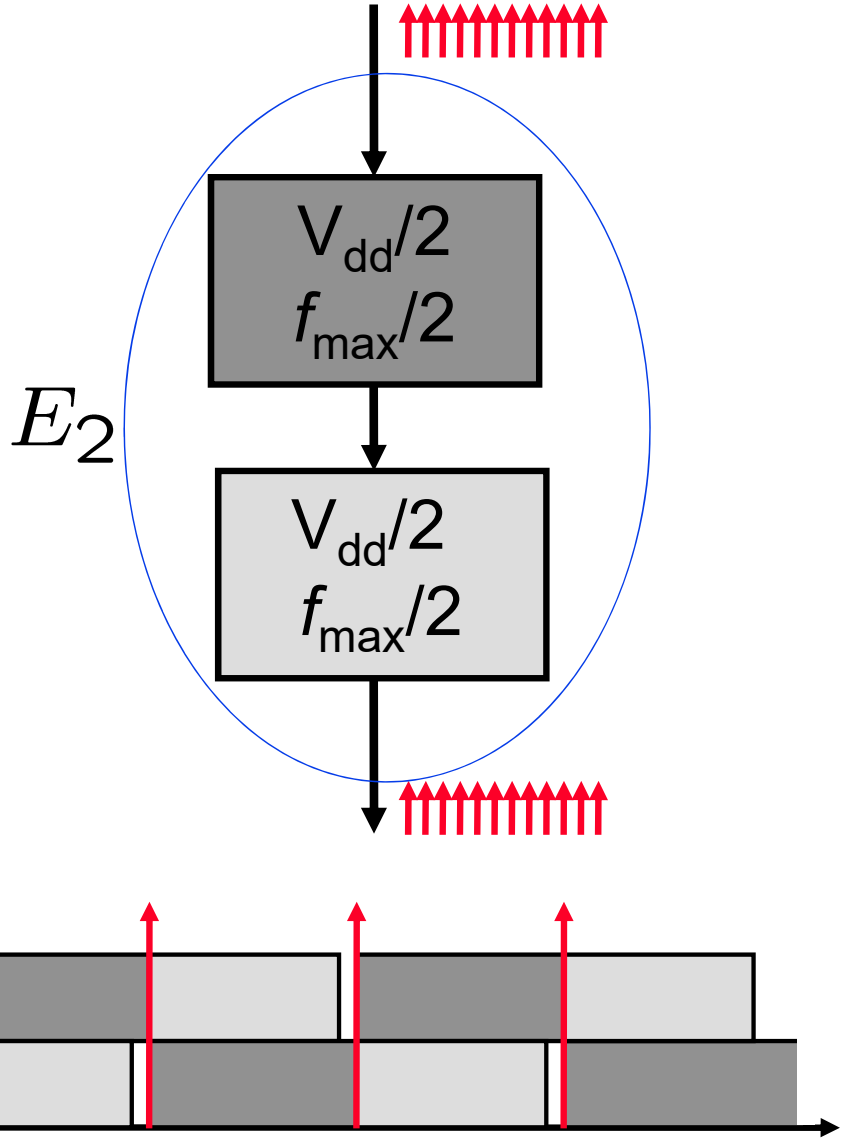


$$E \sim V_{dd}^2 \ (\#\text{cycles})$$

$$E_2 = \frac{1}{4} E_1$$

Use of Pipelining





$$E \sim V_{dd}^2 \ (\#\text{cycles})$$

$$E_2 = \frac{1}{4} E_1$$

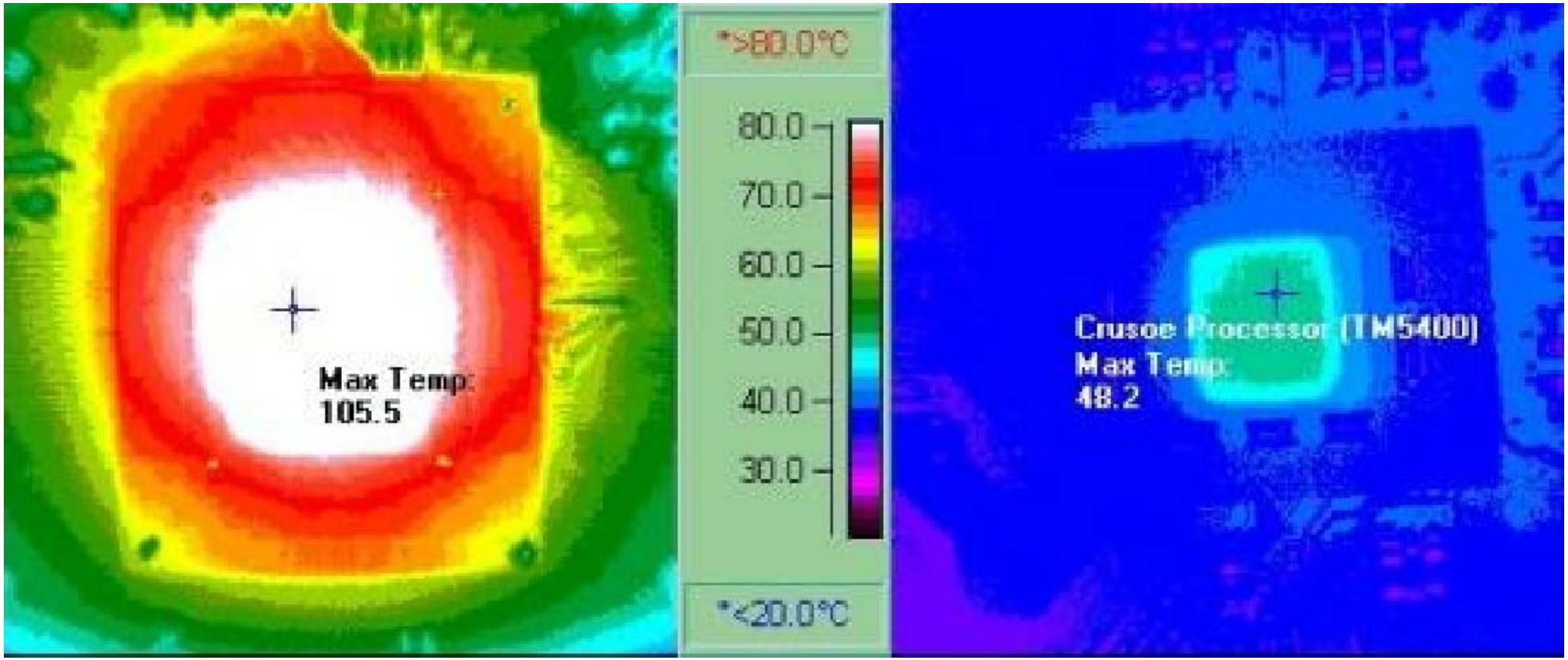
Topics

- General Remarks
- Power and Energy
- Basic Techniques
  - Parallelism
  - ***VLIW (parallelism and reduced overhead)***
  - Dynamic Voltage Scaling
  - Dynamic Power Management

New ideas help …

Pentium

Crusoe

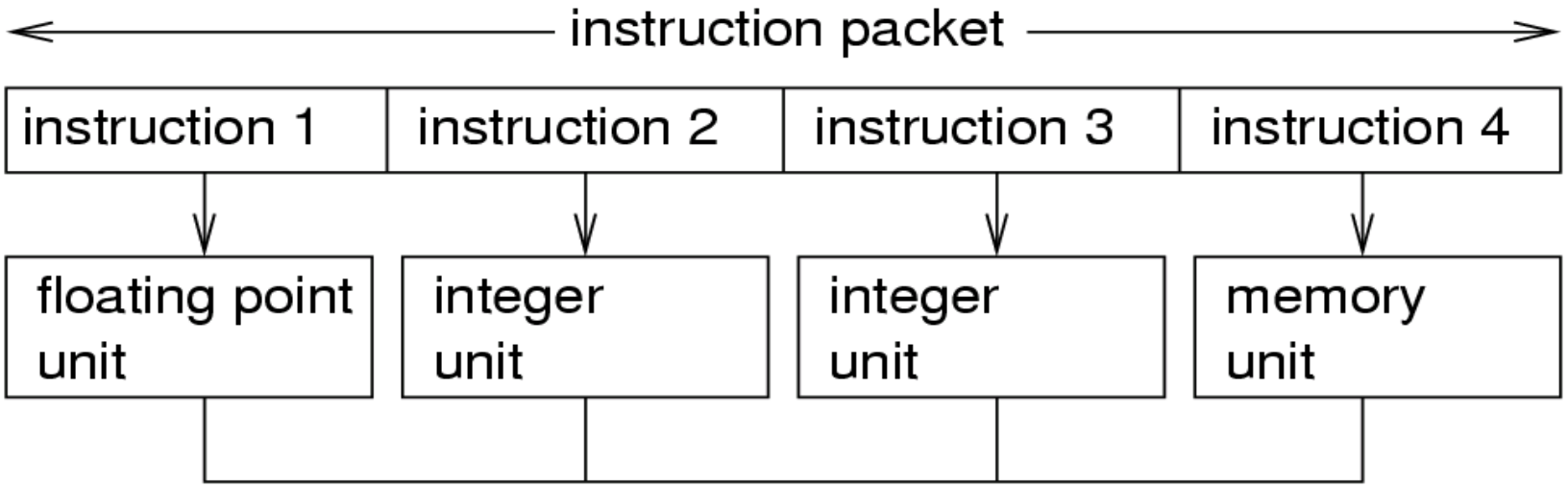


Running the same multimedia application.

As published by Transmeta [www.transmeta.com]

VLIW Architectures

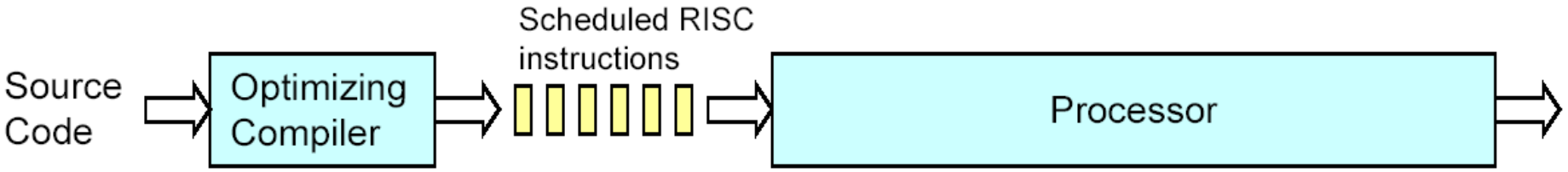
- ***Large degree of parallelism***
  - many computational units, (deeply) pipelined
- ***Simple hardware architecture***
  - explicit parallelism (parallel instruction set)
  - parallelization is done offline (compiler)



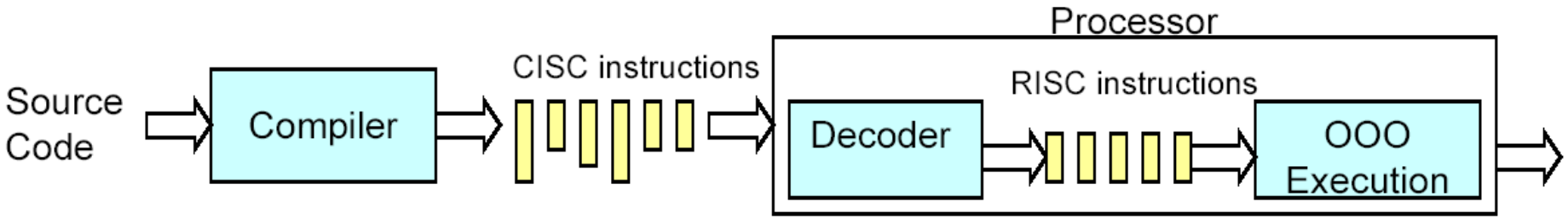
Transmeta was a typical VLIW Architecture

- 128-bit instructions (bundles):
  - 4 operations per instruction
  - 2 combinations of instructions allowed
- Register files
  - 64 integer, 32 floating point
- Some interesting features
  - 6 stage pipeline (2x fetch, decode, register read, execute, write)
  - x86 ISA execution using software techniques
    - Skip the binary compatibility problem!!
    - Interpretation and just-in-time binary translation
  - Speculation support

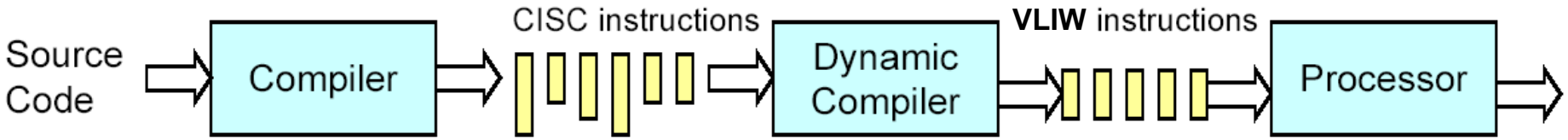
Transmeta



e.g. Sun, MIPS, ARM



e.g. Intel, AMD

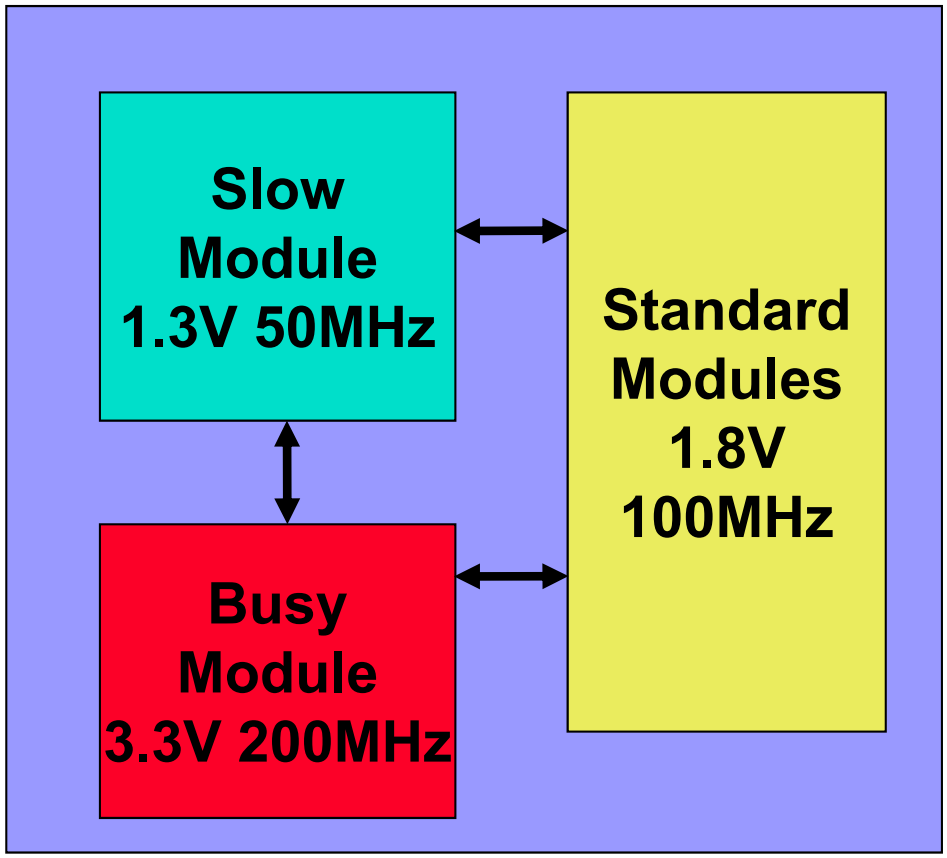


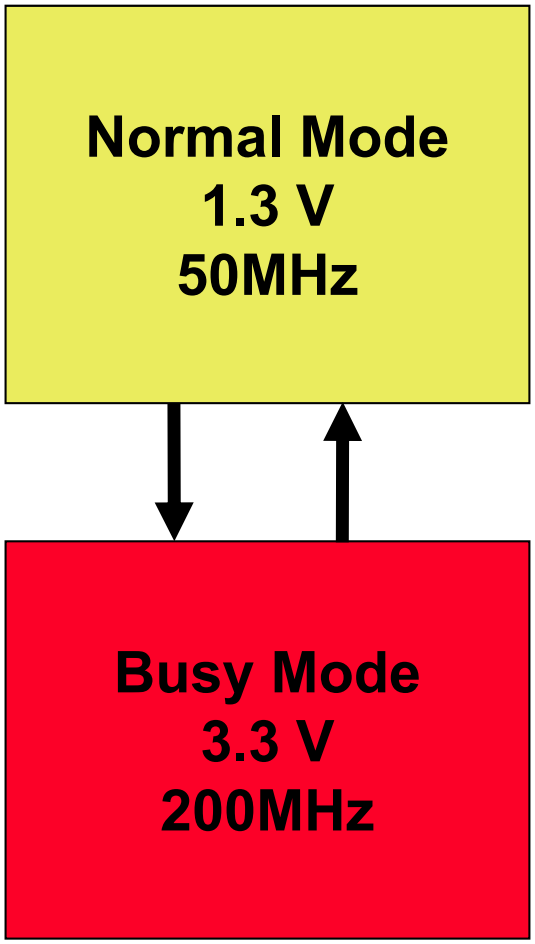
e.g. Transmeta
(VLIW)

Topics

- General Remarks
- Power and Energy
- Basic Techniques
  - Parallelism
  - VLIW (parallelism and reduced overhead)
  - ***Dynamic Voltage Scaling***
  - Dynamic Power Management

Spatial vs. Dynamic Voltage Management





Not all components require same performance.

Required performance may change over time

Potential for Energy Optimization: DVS

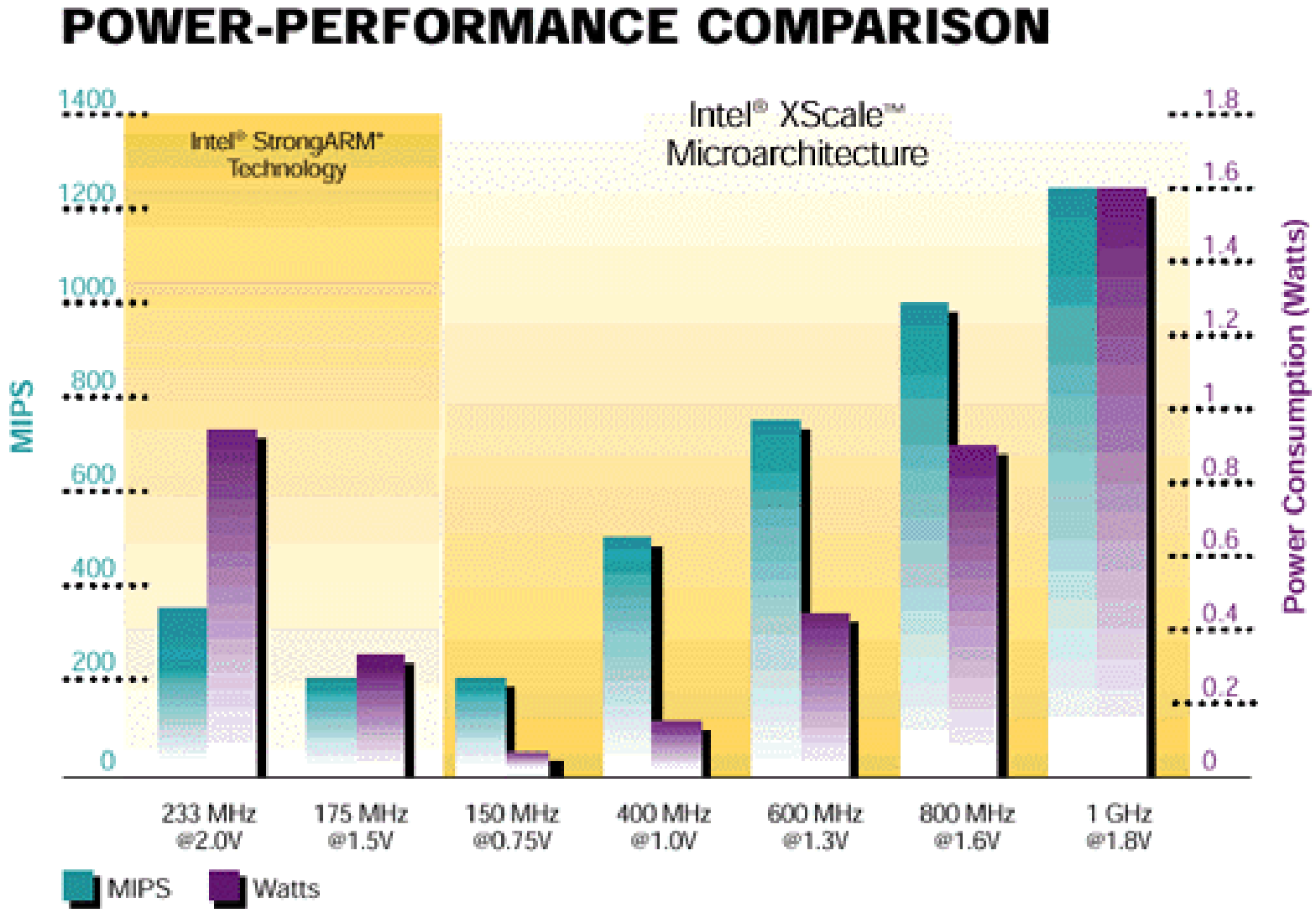
$$P \sim \alpha C_L V_{dd}^2 f$$

$$E \sim \alpha C_L V_{dd}^2 f t = \alpha C_L V_{dd}^2 \ (\#\text{cycles})$$

Saving energy for a given task:

- Reduce the supply voltage V_{dd}
- Reduce switching activity α
- Reduce the load capacitance C_L
- Reduce the number of cycles *#cycles*

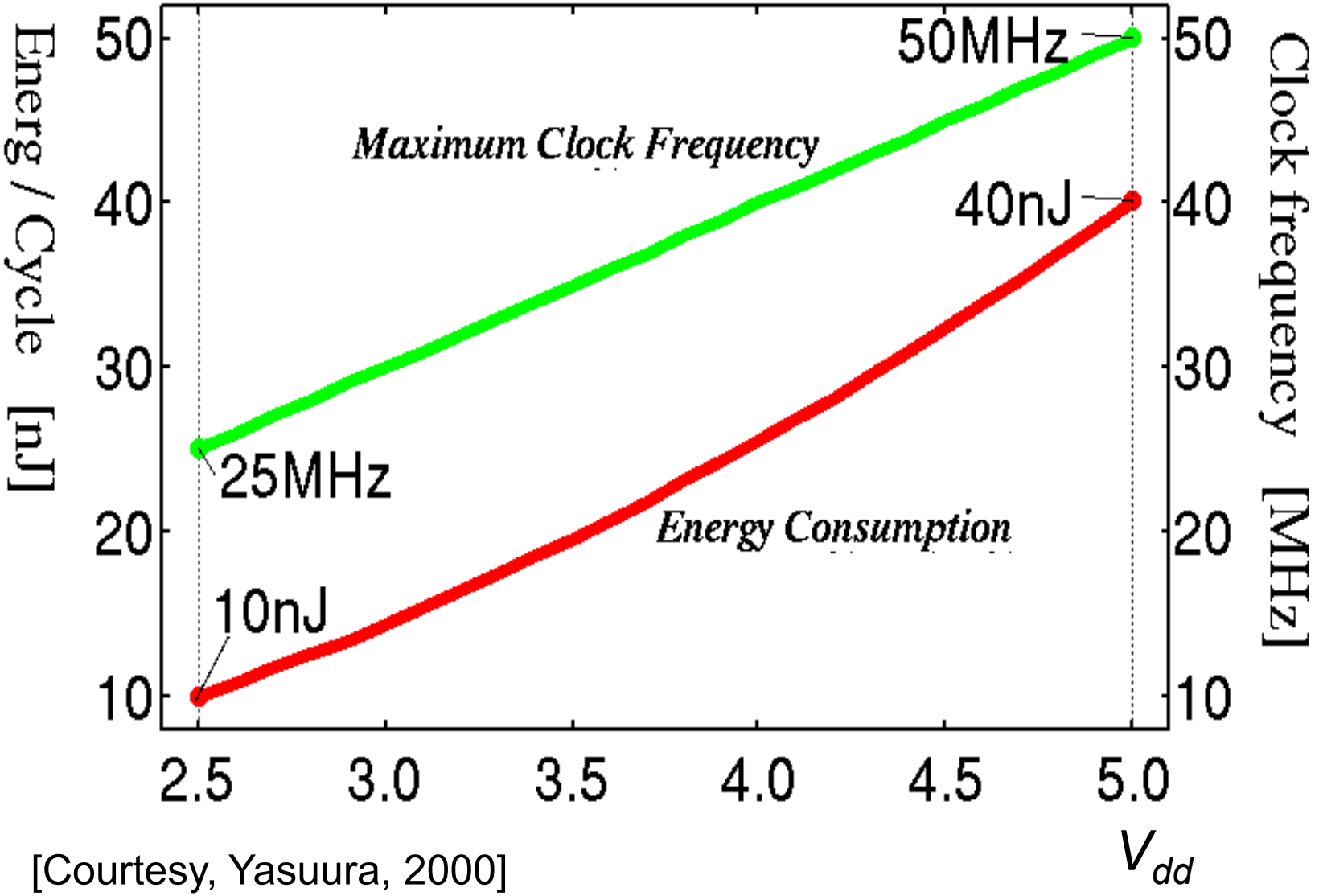
Example: INTEL Xscale



OS should schedule distribution of the energy budget.

From Intel's Web Site

Example: Voltage Scaling



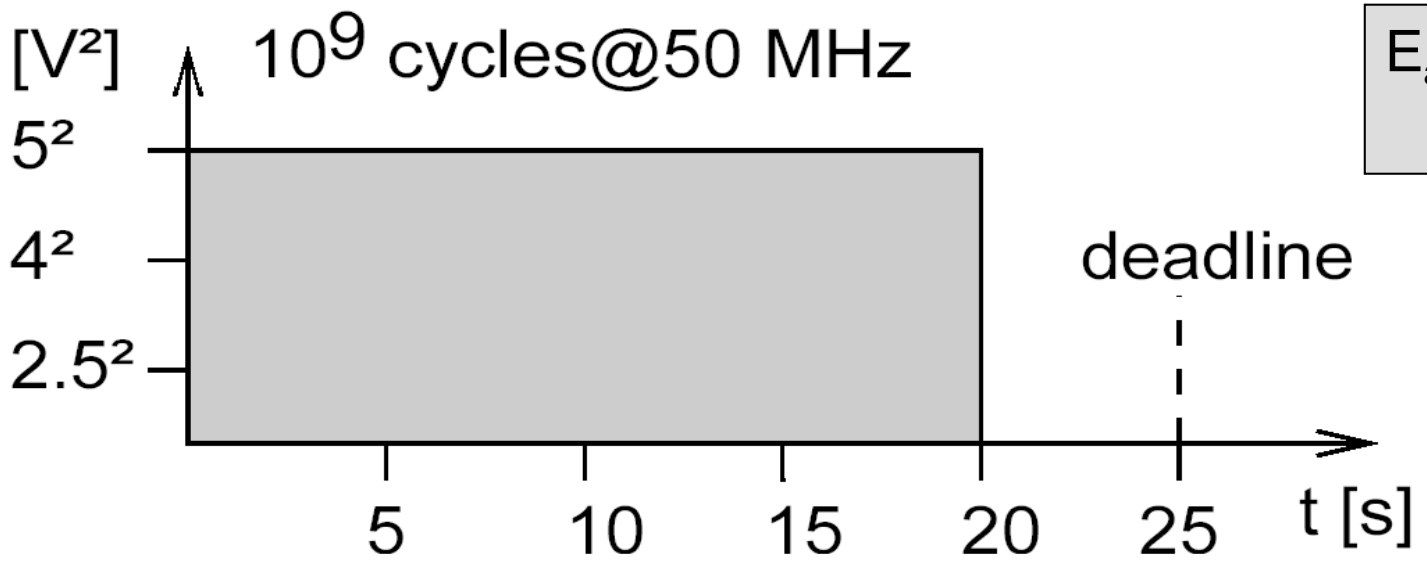
[Courtesy, Yasuura, 2000]

DVS Example: a) Complete task ASAP

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

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

a)

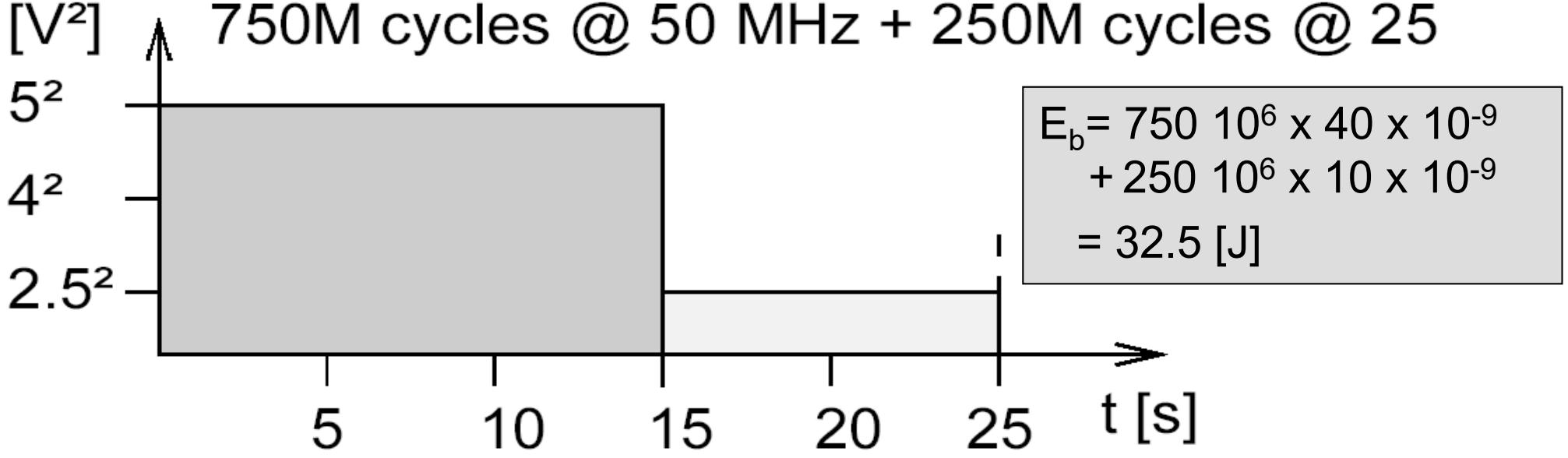


$E_a = 10^9 \times 40 \times 10^{-9}$
$= 40$ [J]

DVS Example: 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)

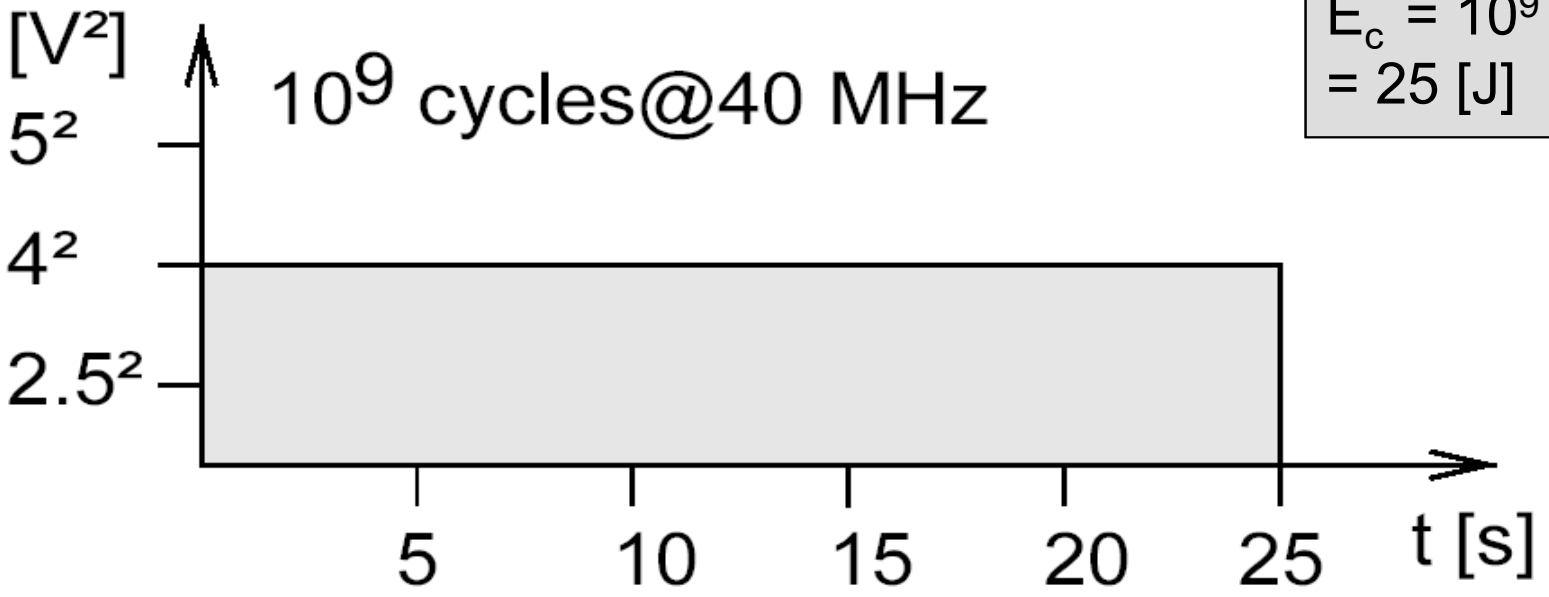


E_b= 750 10^6 x 40 x 10^{-9}
+ 250 10^6 x 10 x 10^{-9}
= 32.5 [J]

DVS Example: c) Optimal Voltage

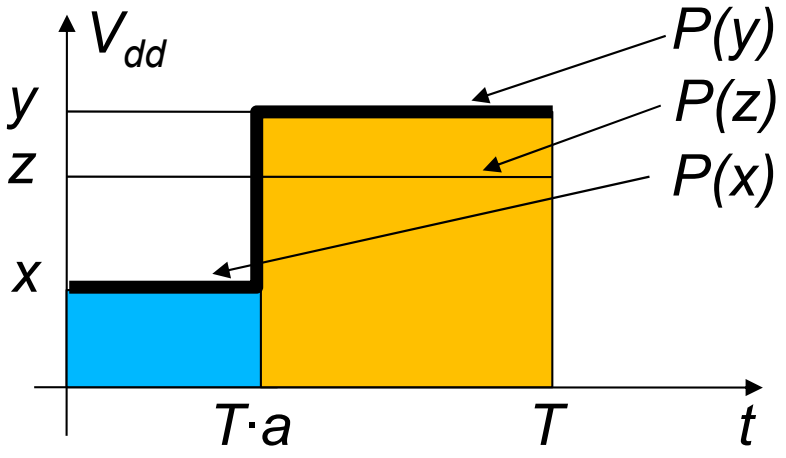
V_{dd} [V]	5.0	4.0	2.5
Energy per cycle [nJ]	40	25	10
f_{max} [MHz]	50	40	25
cycle time [ns]	20	25	40

c)



$E_c = 10^9 \times 25 \times 10^{-9}$
$= 25$ [J]

DVS: Optimal Strategy



$z = a \cdot x + (1\text{-}a) \cdot y$

Execute task in fixed time T with variable voltage $V_{dd}(t)$:

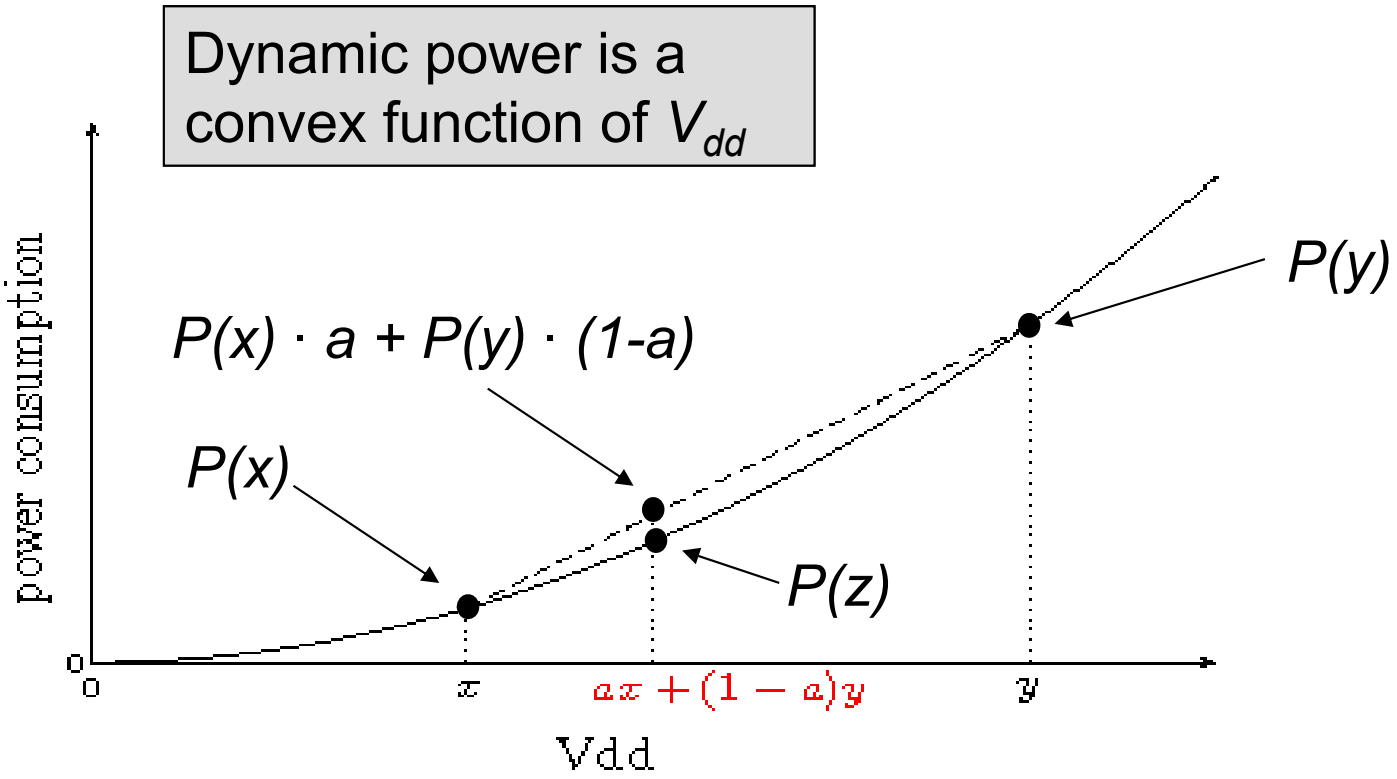
gate delay: $\tau \sim \frac{1}{V_{dd}}$

execution rate: $f(t) \sim V_{dd}(t)$

invariant: $\int V_{dd}(t)dt = \text{const.}$

- ***case A***: execute at voltage x for $T \cdot a$ time units and at voltage y for $(1\text{-}a) \cdot T$ time units;
  energy consumption $T \cdot (P(x) \cdot a + P(y) \cdot (1\text{-}a))$
- ***case B***: execute at voltage $z = a \cdot x + (1\text{-}a) \cdot y$ for T time units;
  energy consumption $T \cdot P(z)$

DVS: Optimal Strategy



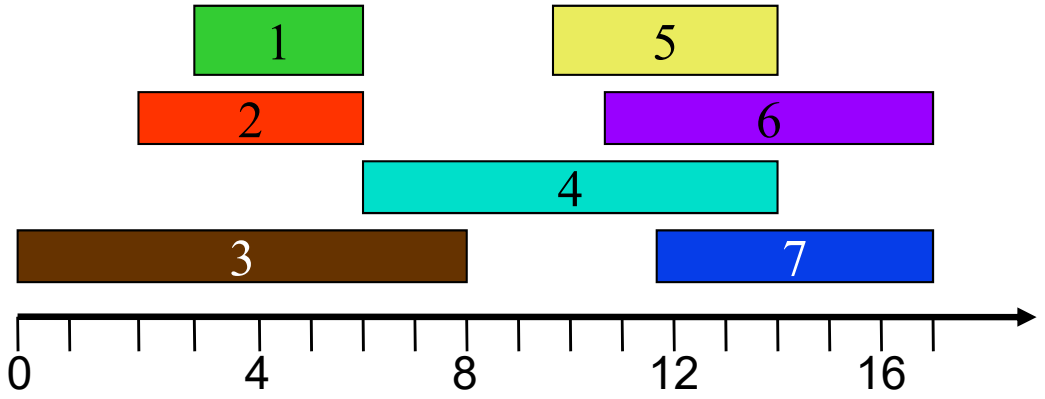
- If possible, running at a constant frequency (voltage) minimizes the energy consumption for dynamic voltage scaling:
  - **case A** is always worse if the power consumption is a convex function of the supply voltage

DVS: Offline Scheduling on One Processor

- Let us model a set of independent tasks as follows:
  - We suppose that a task $v_i \in V$
    - requires c_i computation time at normalized processor frequency 1
    - arrives at time a_i
    - has (absolute) deadline constraint d_i
- How do we schedule these tasks such that all these tasks can be finished ***no later than their deadlines*** and the energy consumption is ***minimized***?
  - YDS Algorithm from "A Scheduling Model for Reduce CPU Energy", Frances Yao, Alan Demers, and Scott Shenker, FOCS 1995."

If possible, running at a constant frequency (voltage) minimizes the energy consumption for dynamic voltage scaling.

YDS Algorithm for Offline Scheduling



3,6,5

2,6,3

0,8,2

6,14,6

10,14,6

11,17,2

12,17,2

a_i,d_i,c_i

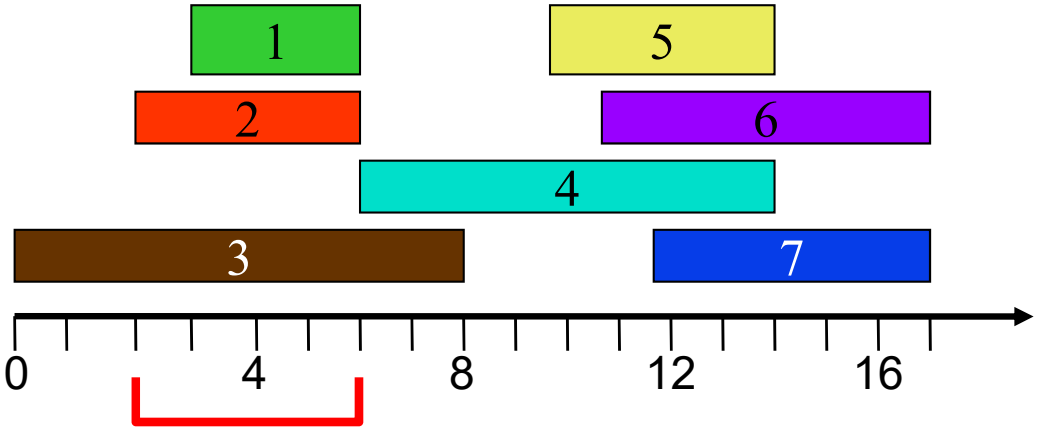
- Define ***intensity*** G([*z*, *z*‘]) in some time interval [*z*, *z*‘]:
  - average accumulated execution time of all tasks that have arrival and deadline in [*z*, *z*‘] relative to the length of the interval *z*‘-*z*

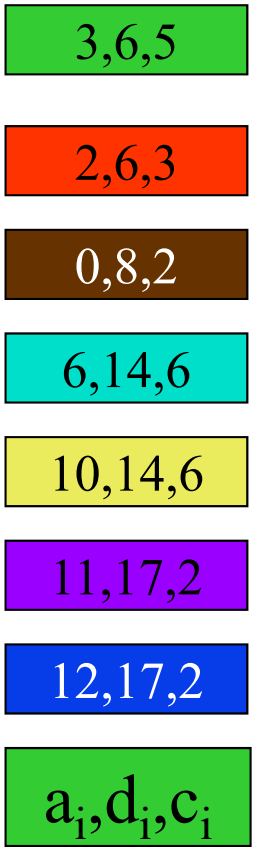
$$V'([z,z']) = \{v_i \in V \ : \ z \leq a_i < d_i \leq z'\}$$

$$G([z,z']) = \sum_{v_i \in V'([z,z'])} c_i/(z'-z)$$

YDS Algorithm for Offline Scheduling

- ***Step 1:*** Execute jobs in the interval with the highest intensity by using the earliest-deadline first schedule and running at the intensity as the frequency.





G([0,6]) = (5+3)/6=8/6, G([0,8]) = (5+3+2)/ (8-0) = 10/8,

G([0,14]) = (5+3+2+6+6)/14=11/7, G([0,17]) = (5+3+2+6+6+2+2)/17=26/17

G([2, 6]) = (5+3)/(6-2)=2, G([2,14]) = (5+3+6+6) / (14-2) = 5/3,

G([2,17]) = (5+3+6+6+2+2)/15=24/15

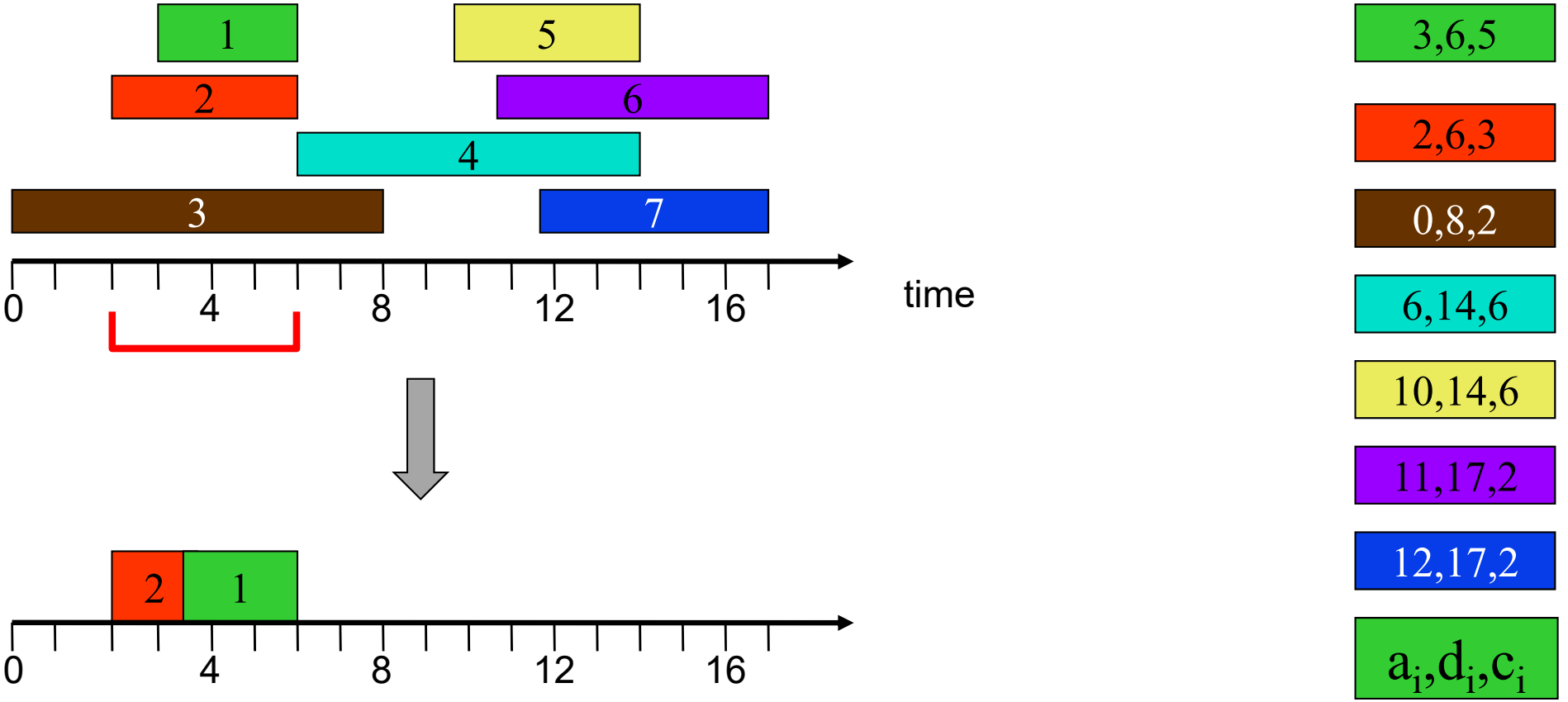
G([3,6]) =5/3, G([3,14]) = (5+6+6)/(14-3) = 17/11, G([3,17])=(5+6+6+2+2)/14=21/14

G([6,14]) = 12/(14-6)=12/8, G([6,17]) = (6+6+2+2)/(17-6)=16/11

G([10,14]) = 6/4, G([10,17]) = 10/7, G([11,17]) = 4/6, G([12,17]) = 2/5

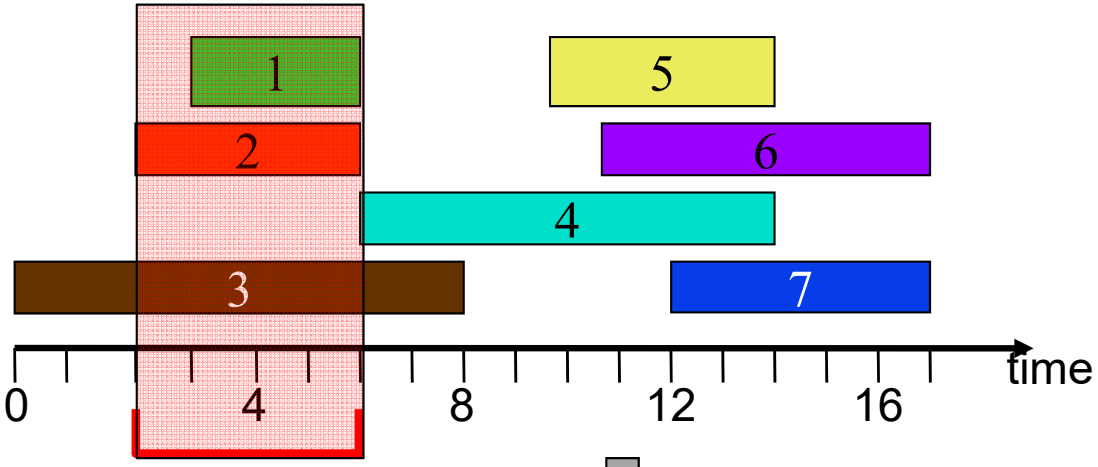
YDS Algorithm for Offline Scheduling

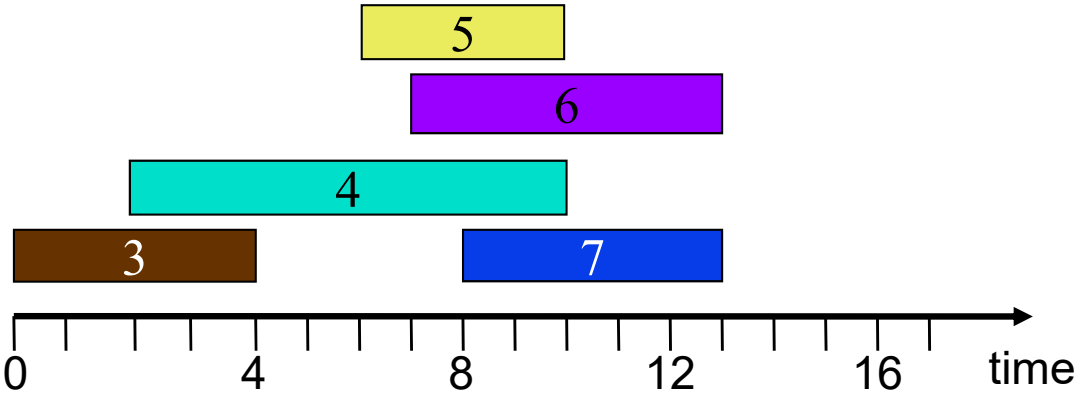
- ***Step 1:*** Execute jobs in the interval with the highest intensity by using the earliest-deadline first schedule and running at the intensity as the frequency.

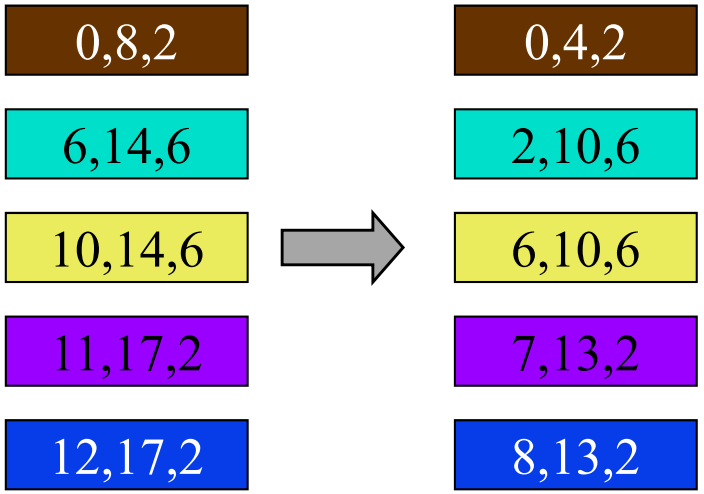


YDS Algorithm for Offline Scheduling

- ***Step 2:*** Adjust the arrival times and deadlines by excluding the possibility to execute at the previous critical intervals.



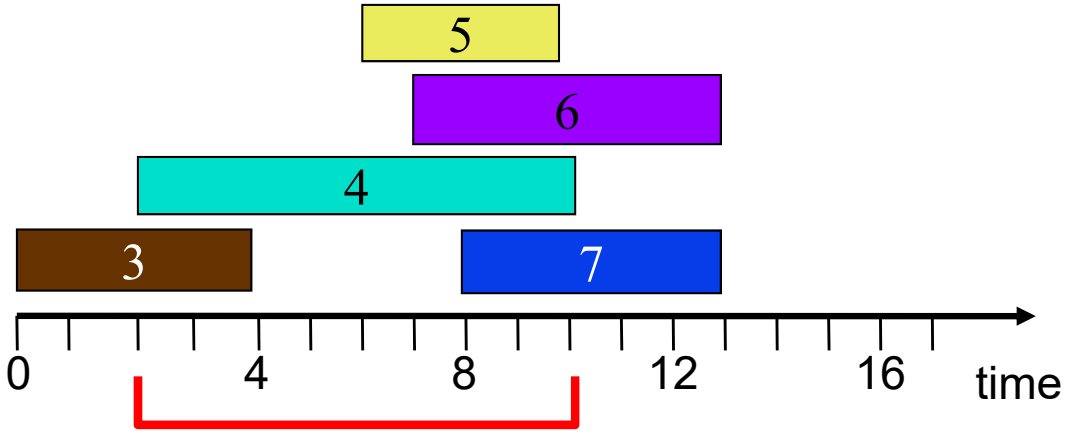




a_i, d_i, c_i

YDS Algorithm for Offline Scheduling

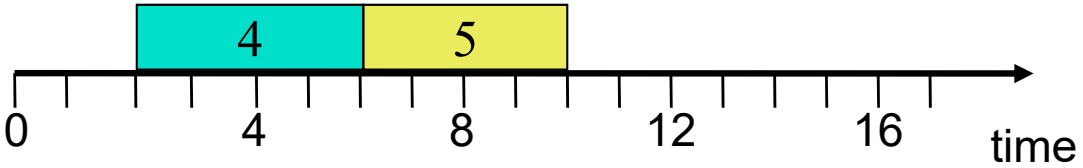
- ***Step 3:*** Run the algorithm for the revised input again



G([0,4])=2/4, G([0,10]) = 14/10, G([0,13])=18/13

G([2,10])=12/8, G([2,13]) = 16/11, G([6,10])=6/4

G([6,13])=10/7, G([7,13])=4/6, G([8,13])=4/5



0,4,2

2,10,6

6,10,6

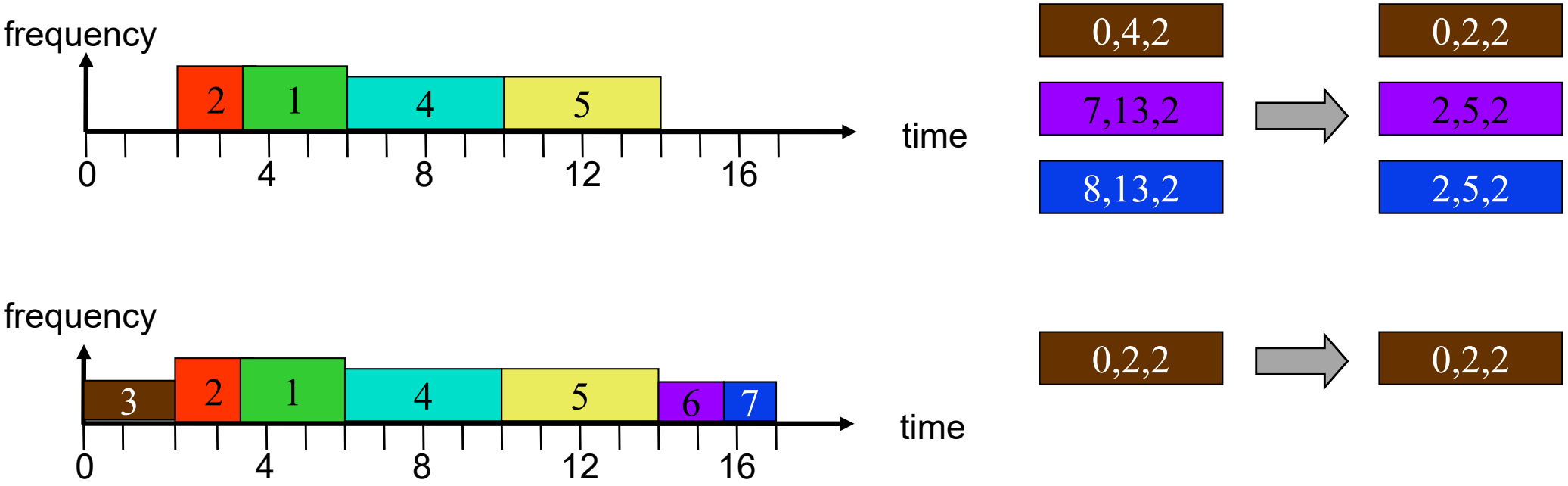
7,13,2

8,13,2

a_i,d_i,c_i

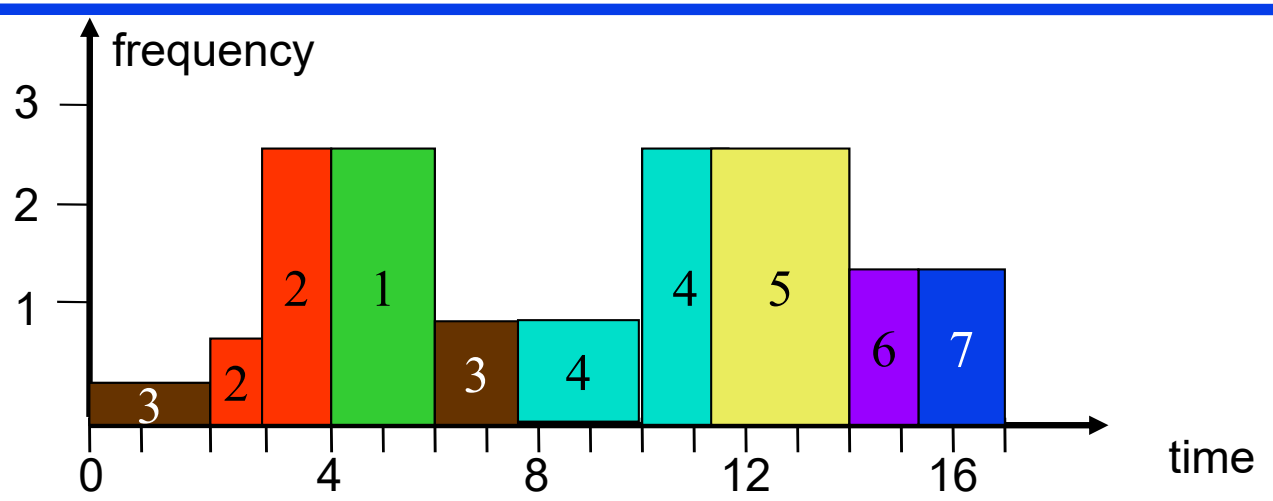
YDS Algorithm for Offline Scheduling

- ***Step 3:*** Run the algorithm for the revised input again
- ***Step 4:*** Put pieces together



	v_1	v_2	v_3	v_4	v_5	v_6	v_7
frequency	2	2	1	1.5	1.5	4/3	4/3

DVS: Online Scheduling on One Processor



3,6,5

2,6,3

0,8,2

6,14,6

10,14,6

11,17,2

12,17,2

a_i,d_i,c_i

- **Continuously update to the best schedule for all arrived tasks**
  - Time 0: task v_3 is executed at 2/8
  - Time 2: task v_2 arrives
    - G([2,6]) = ¾, G([2,8]) = 4.5/6=3/4 => execute v_2 at ¾
  - Time 3: task v_1 arrives
    - G([3,6]) = (5+3-3/4)/3=29/12, G([3,8]) < G([3,6]) => execute v_2 and v_1 at 29/12
  - Time 6: task v_4 arrives
    - G([6,8]) = 1.5/2, G([6,14]) = 7.5/8 => execute v_3 and v_4 at 15/16
  - Time 10: task v_5 arrives
    - G([10,14]) = 39/16 => execute v_4 and v_5 at 39/16
  - Time 11 and Time 12
    - The arrival of v_6 and v_7 does not change the critical interval
  - Time 14:
    - G([14,17]) = 4/3 => execute v_6 and v_7 at 4/3

Remarks on YDS Algorithm

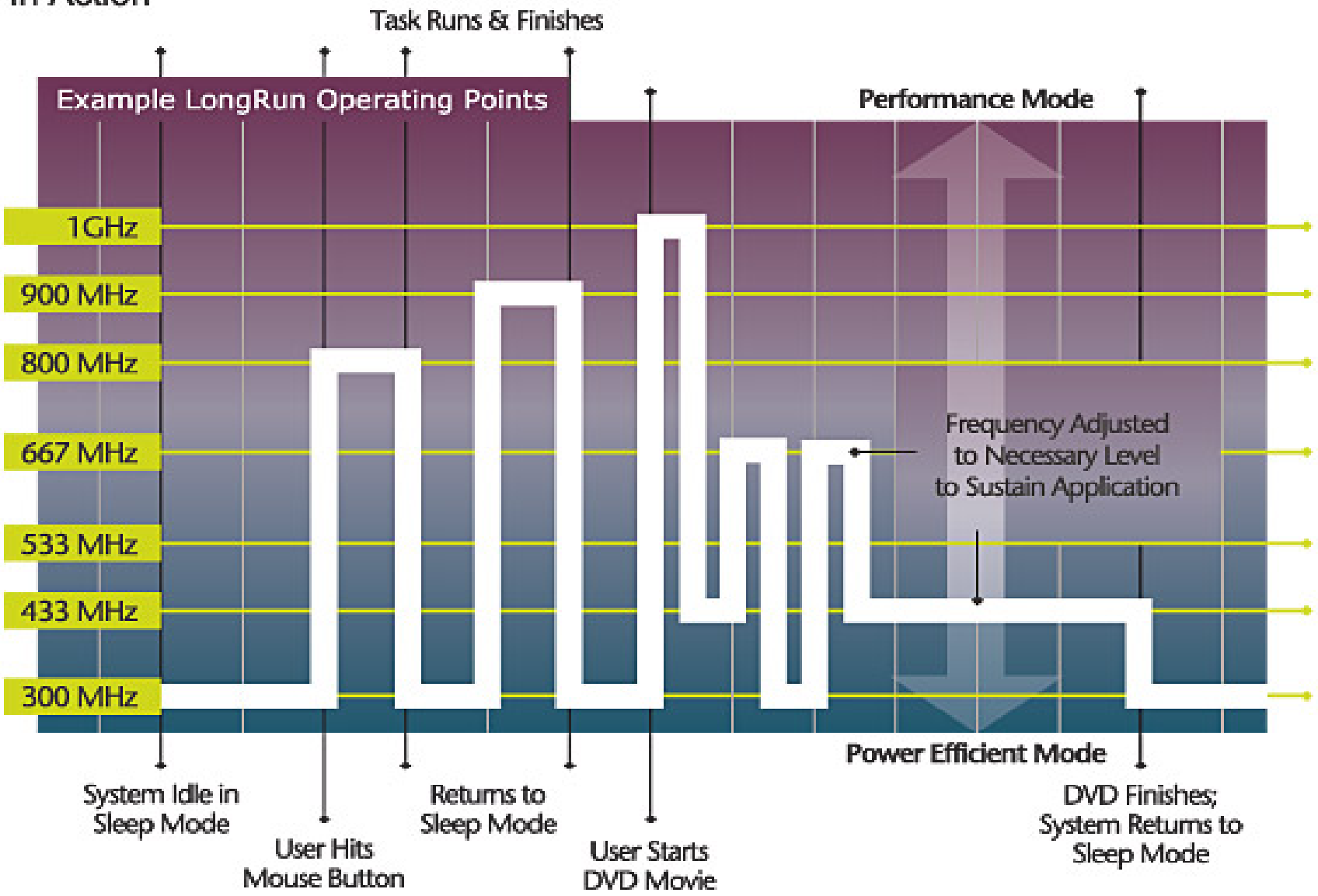
- ***Offline***
  - The algorithm guarantees the minimal energy consumption while satisfying the timing constraints
  - The time complexity is $O(N^3)$, where N is the number of tasks in V
    - Finding the critical interval can be done in $O(N^2)$
    - The number of iterations is at most N
  - Exercise:
    - For periodic real-time tasks with deadline=period, running at ***constant speed with 100% utilization*** under EDF has minimum energy consumption while satisfying the timing constraints.
- ***Online***
  - Compared to the optimal offline solution, the on-line schedule uses at most 27 times of the minimal energy consumption.

Topics

- General Remarks
- Power and Energy
- Basic Techniques
  - Parallelism
  - VLIW (parallelism and reduced overhead)
  - Dynamic Voltage Scaling
  - ***Dynamic Power Management***

Transmeta™ LongRun™ Power Management

In Action



Dynamic Power Management (DPM)

Dynamic Power management tries to assign optimal **power saving states**

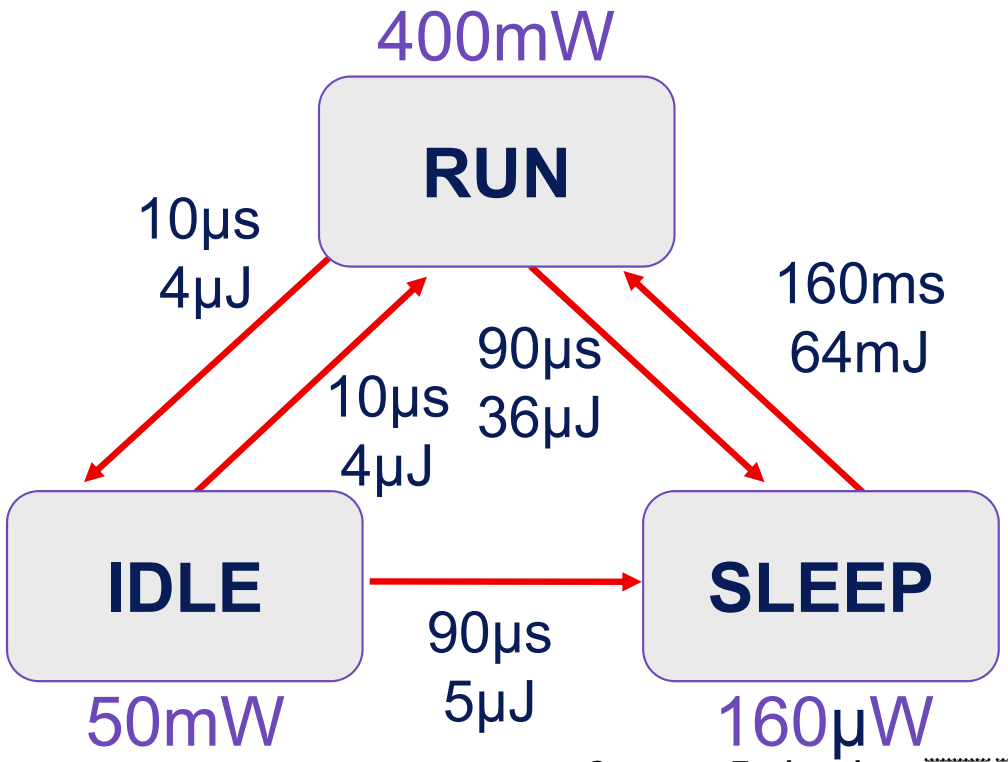
Requires Hardware Support

Example: StrongARM SA1100

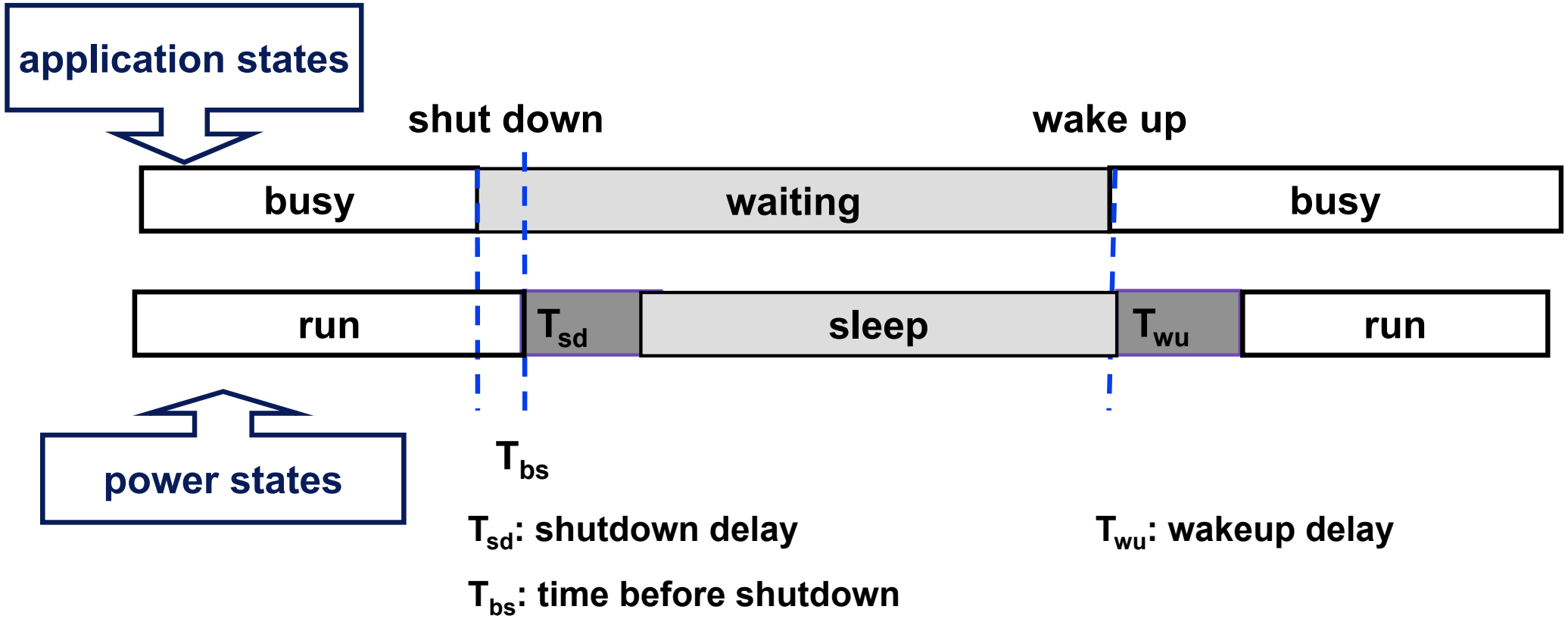
RUN: operational

IDLE: a SW routine may stop the CPU when not in use, while monitoring interrupts

SLEEP: Shutdown of on-chip activity



Reduce Power According to Workload



Desired: Shutdown only during long idle times

→ Tradeoff between savings and overhead

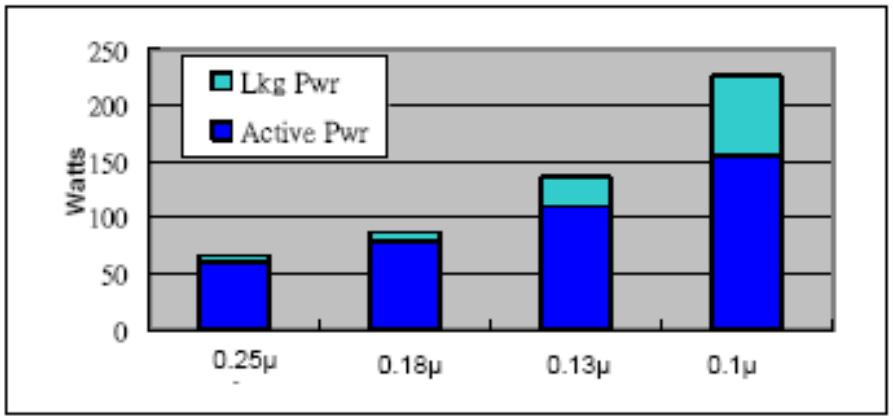
The Challenge

Questions:

- When to go to a power-saving state?
- Is an idle period long enough for shutdown?

→ Predicting the future

Combining DVFS and DPM

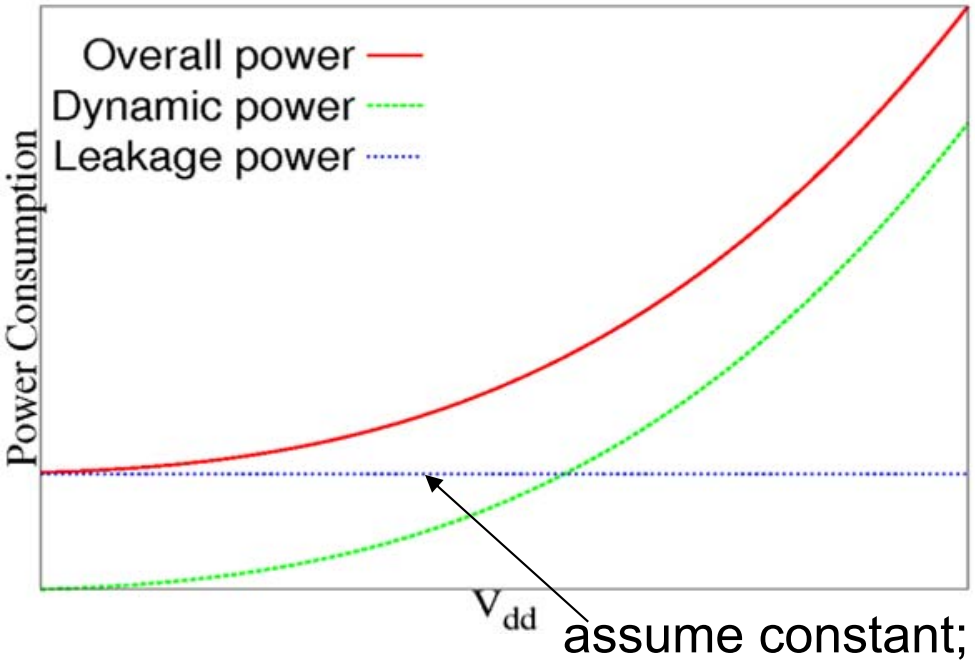


(Micro32 Keynotes by Fred Pollack)

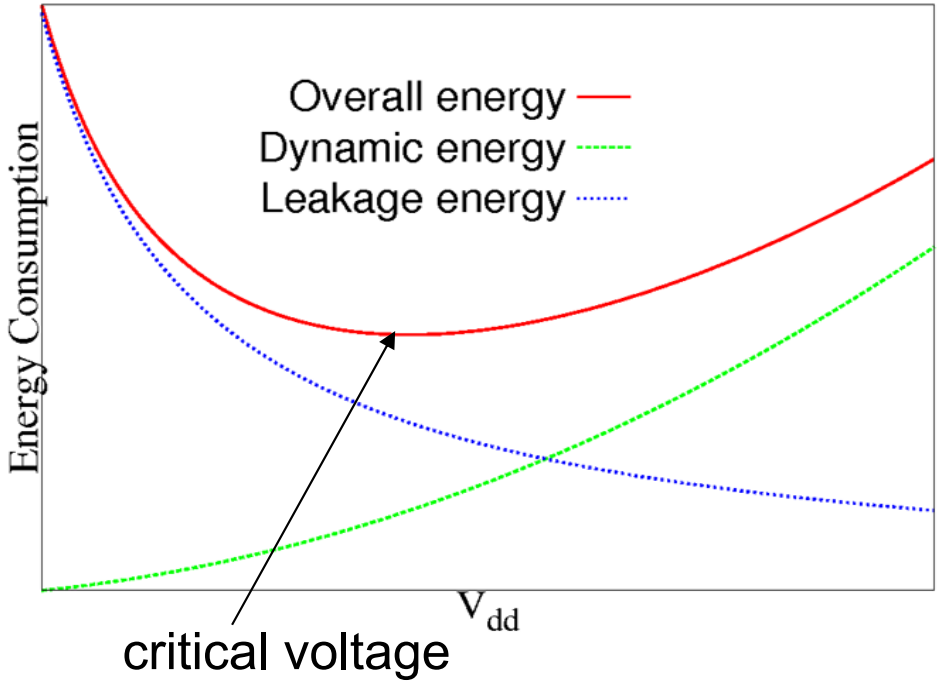
DVS Critical frequency (voltage):

- Running at any frequency/voltage lower than this frequency is not worthwhile for execution.

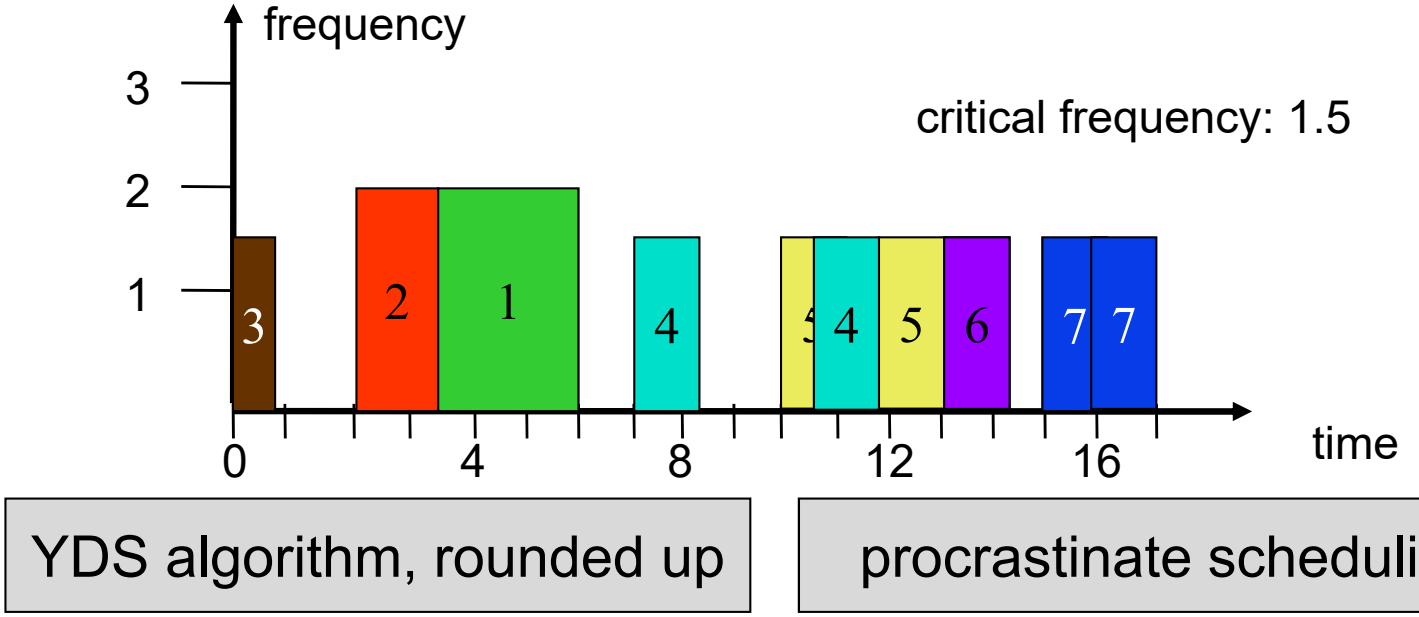
power during „run task“ using voltage and frequency scaling

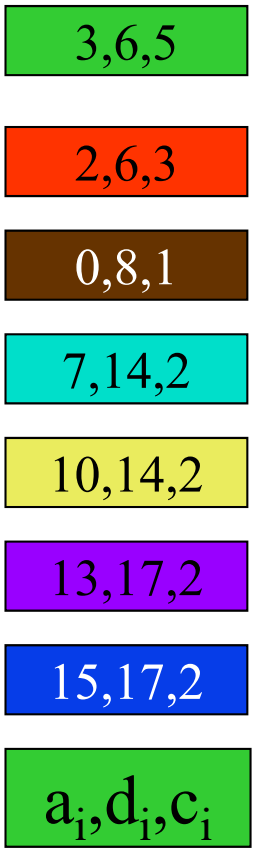


energy for executing task



Procrastination Schedule





- Execute by using voltages higher or equal to the critical voltage only
  - apply YDS algorithm
  - round up voltages lower than the critical voltage
- Procrastinate the execution of tasks to aggregate enough time for sleeping
  - Try to reduce the number of times to turn on/off
  - Sleep as long as possible

Operating System Services

= Hardware power management

= Application Software

= RTOS Power Management Framework