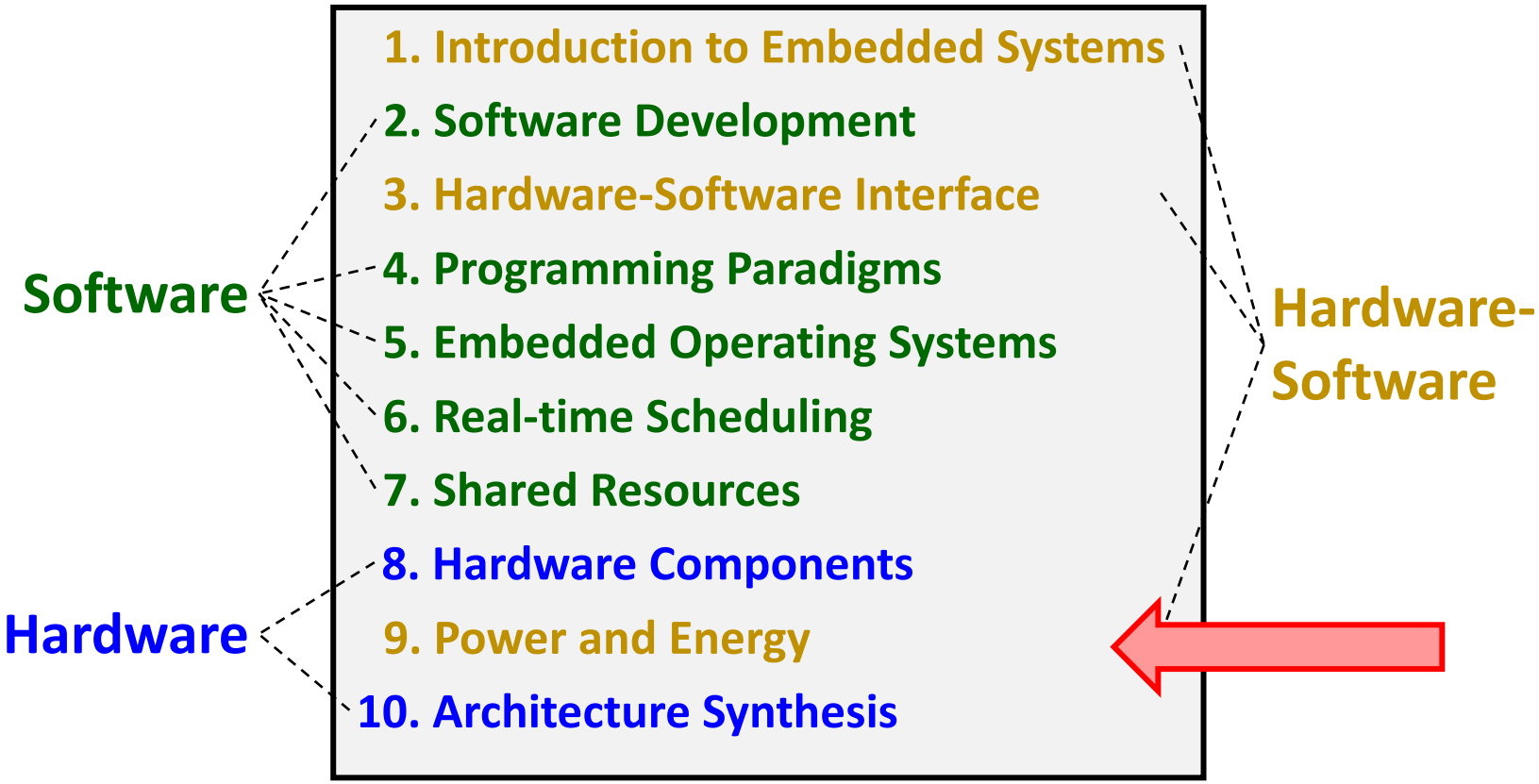
Introduction to Embedded Systems

9. Power and Energy

Prof. Dr. Marco Zimmerling

Where we are …


1. Introduction to Embedded Systems
2. Software Development
3. Hardware-Software Interface
4. Programming Paradigms
5. Embedded Operating Systems
6. Real-time Scheduling
7. Shared Resources
8. Hardware Components
9. Power and Energy
10. Architecture Synthesis
Software
Hardware
Hardware-
Software

General Remarks

Power and Energy Consumption

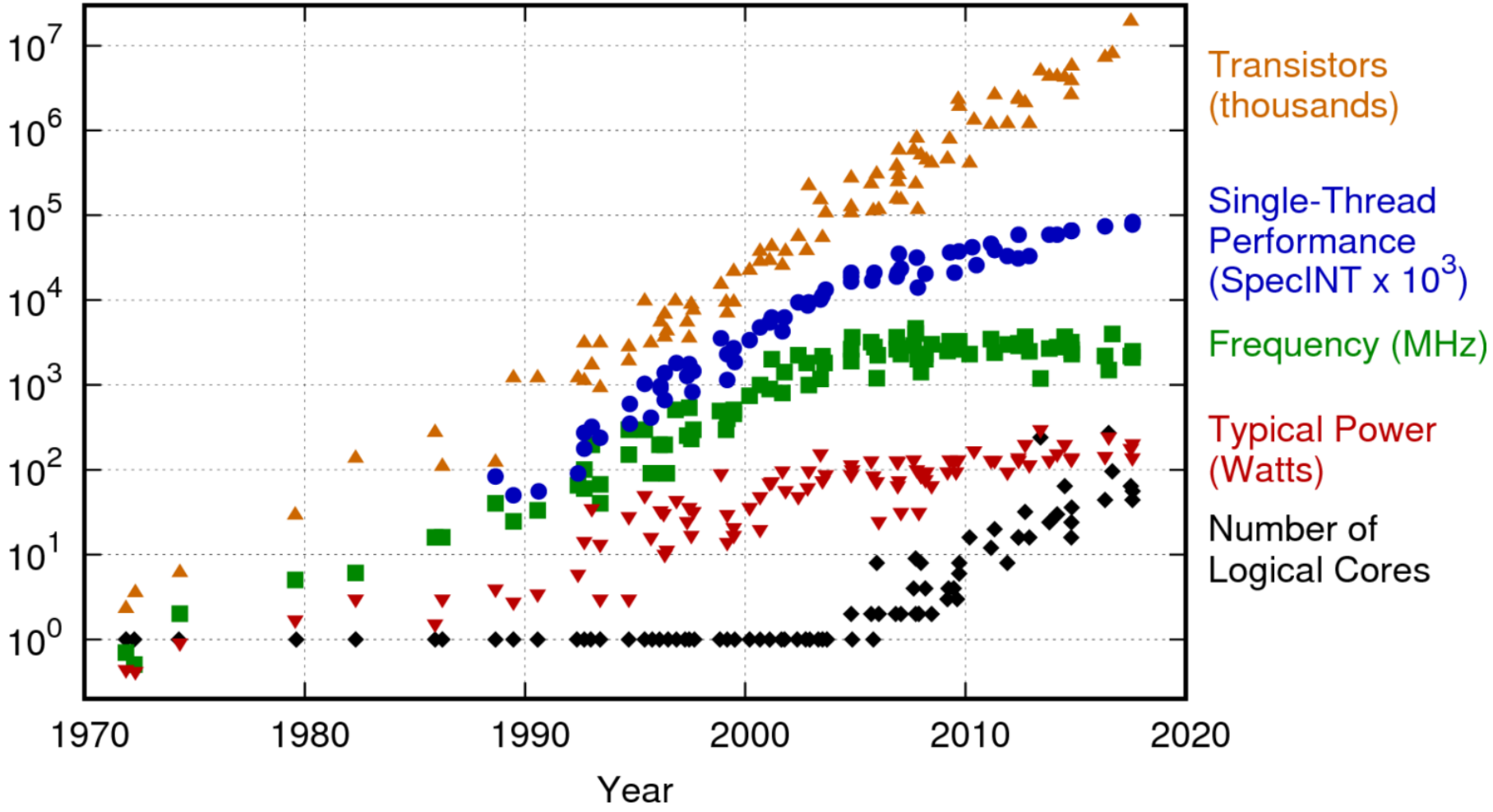
- Statements that are true since a decade or longer:

  „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]

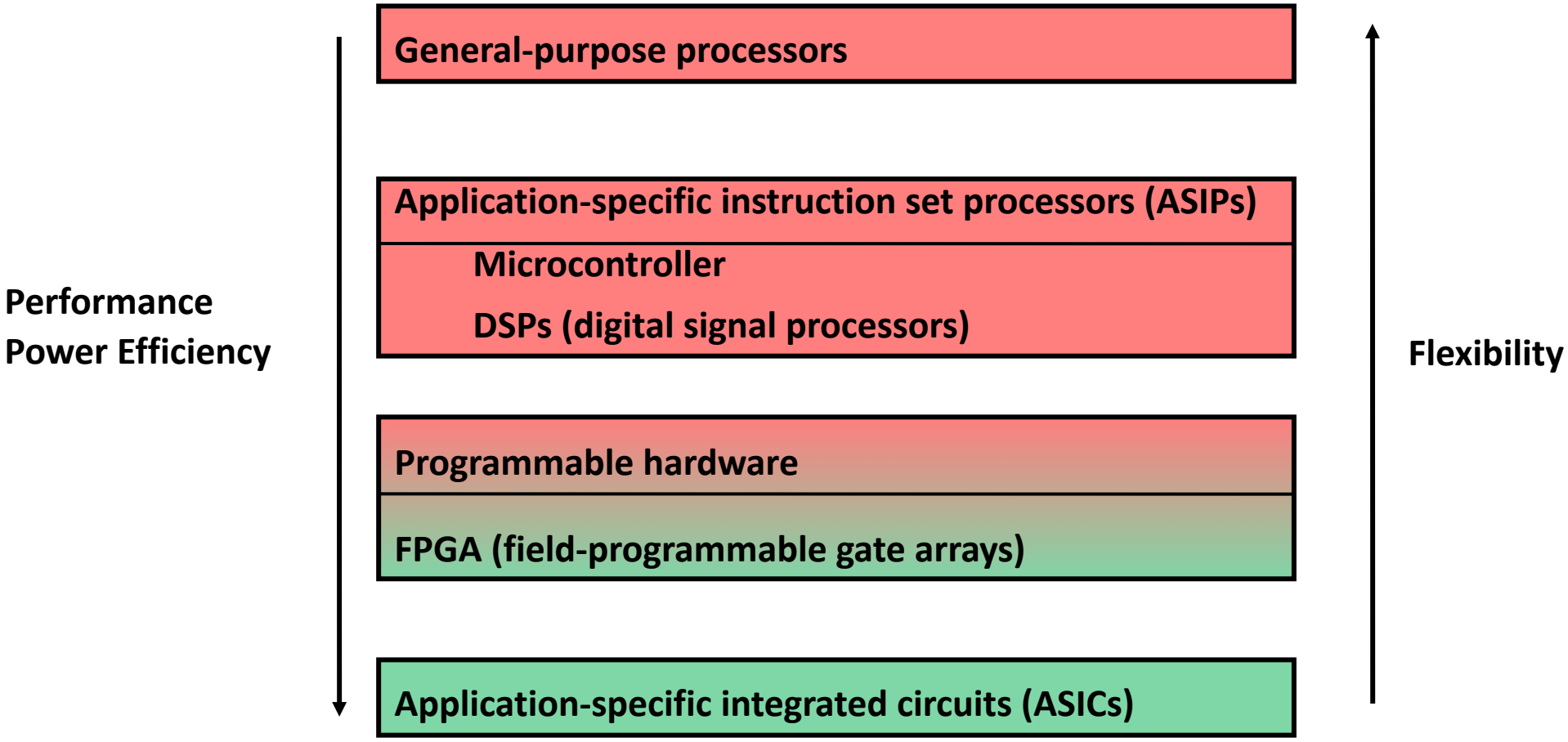
- *Main reasons* are:
  - power provisioning is expensive
  - battery capacity is growing only slowly
  - devices may overheat
  - energy harvesting (e.g., from solar cells) is limited due to low energy density

Some Trends



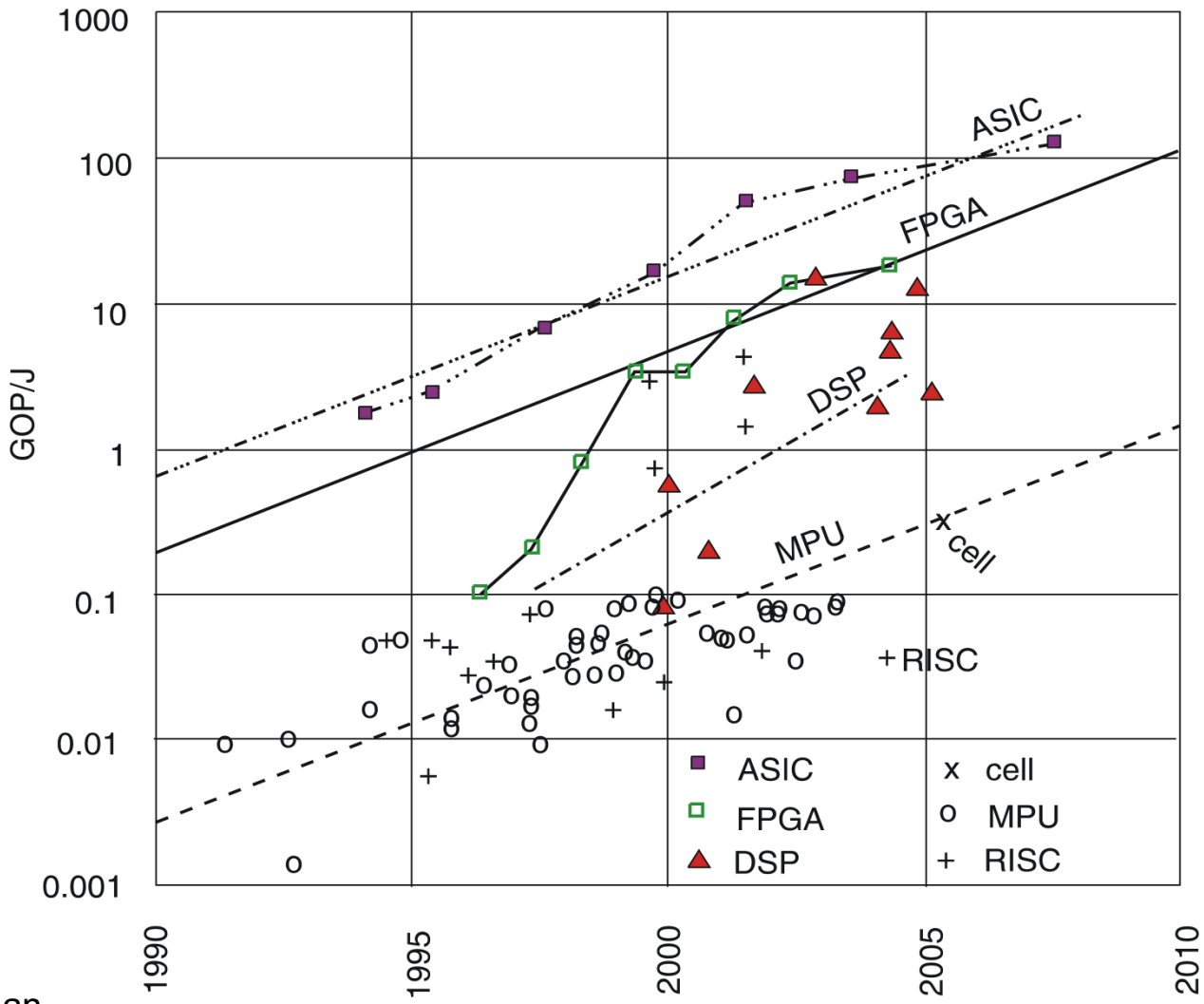
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Implementation Alternatives



Energy Efficiency

- It is necessary to *optimize HW and SW*.
- Use *heterogeneous architectures* in order to adapt to required performance and application.
- Apply *specialization techniques*:
  - Higher parallelism
  - Turn off unused components (e.g., low-power modes)
  - Higher heterogeneity
  - Voltage and frequency scaling



© Hugo De Man,
IMEC, Philips, 2007

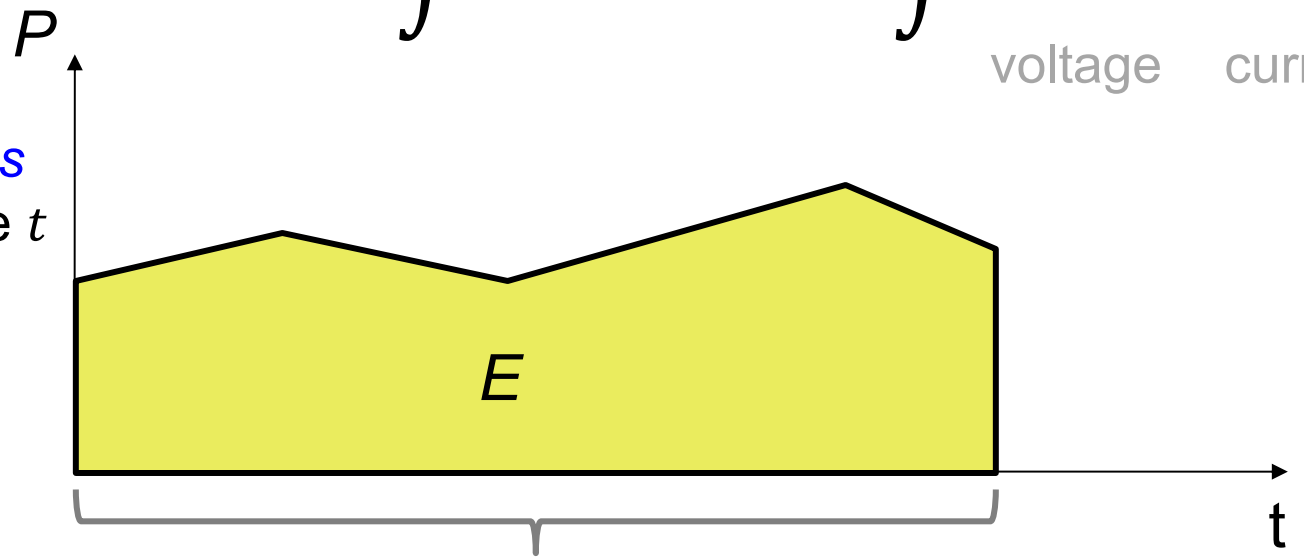
Power and Energy

Power and Energy

$$E = \int P(t)dt = \int V(t) \cdot I(t)dt$$

voltage current

Power is an *instantaneous quantity* measured at time t

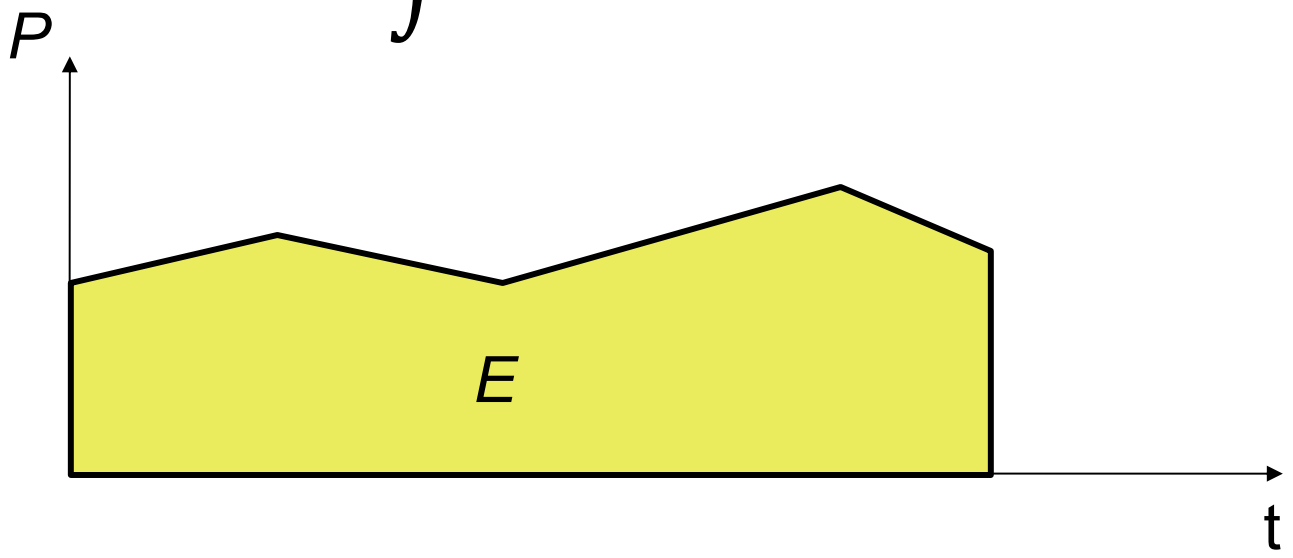


Energy is consumed *by a given task* over a certain time interval

Because energy is not an instantaneous quantity, it is important to specify for *what* a given amount of energy is used (context)!

Power and Energy

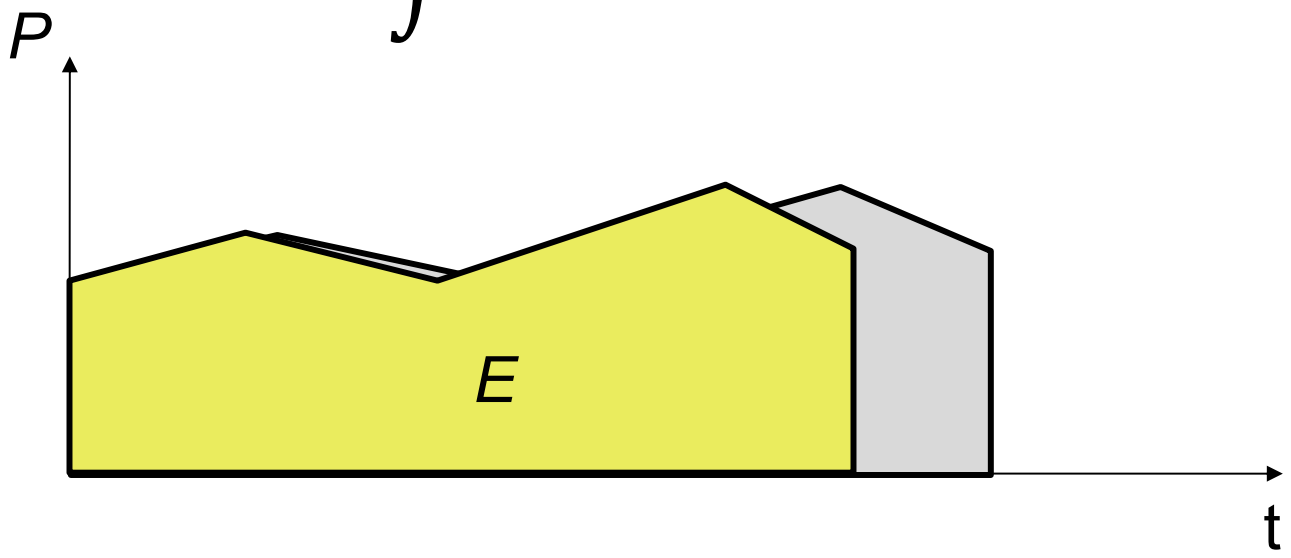
$$E = \int P(t)dt$$



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

Power and Energy

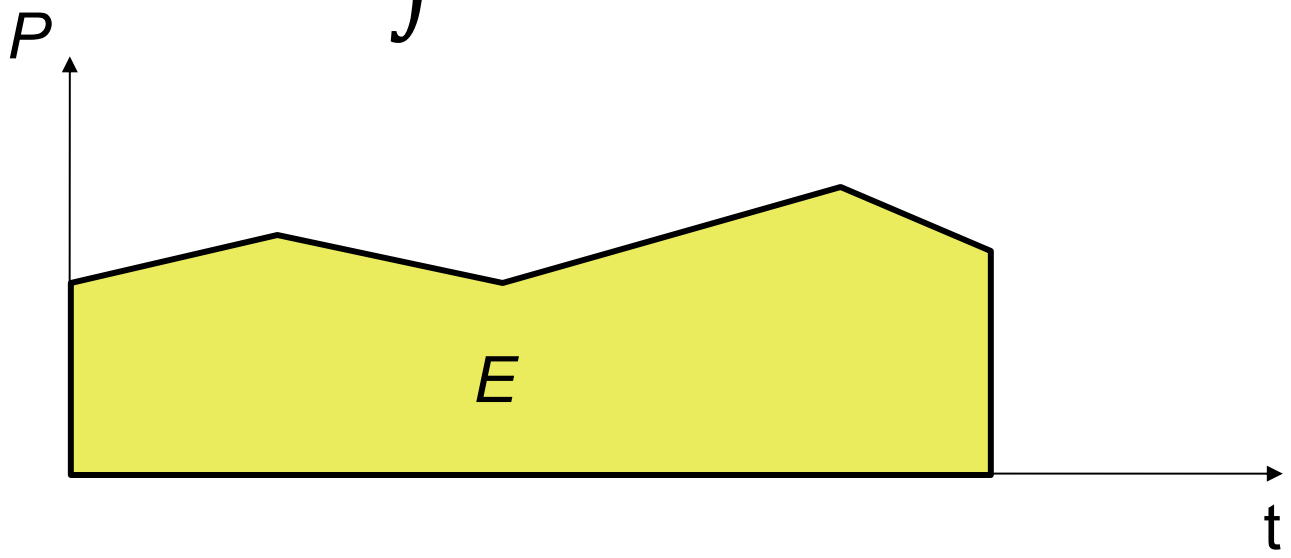
$$E = \int P(t)dt$$



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

Power and Energy

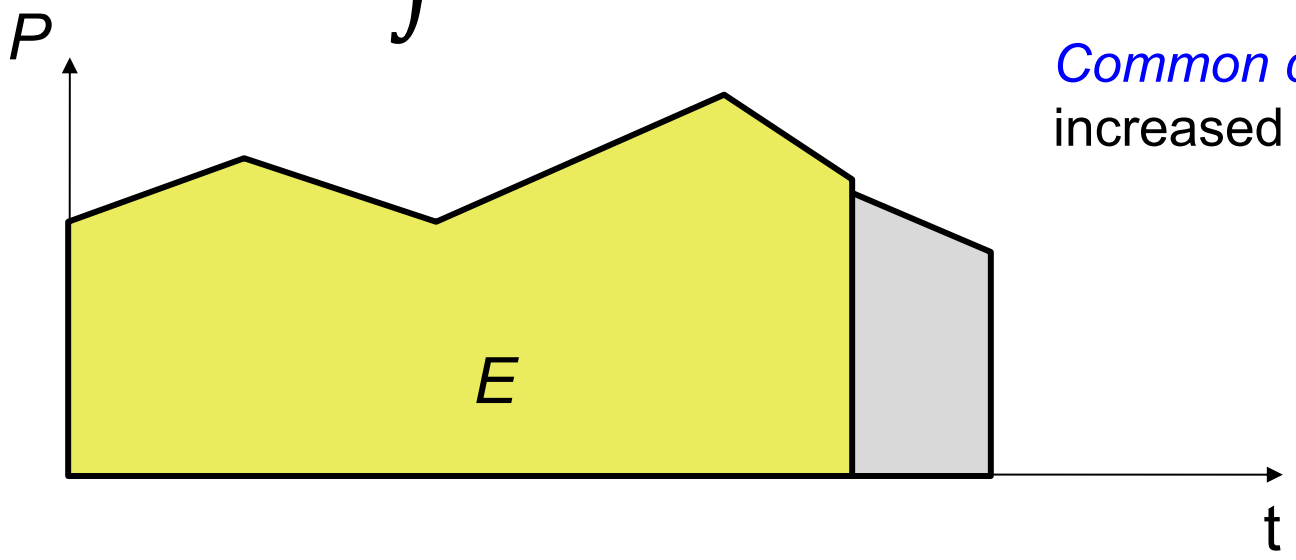
$$E = \int P(t)dt$$



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

Power and Energy

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



Common case: Power must be increased to execute faster.

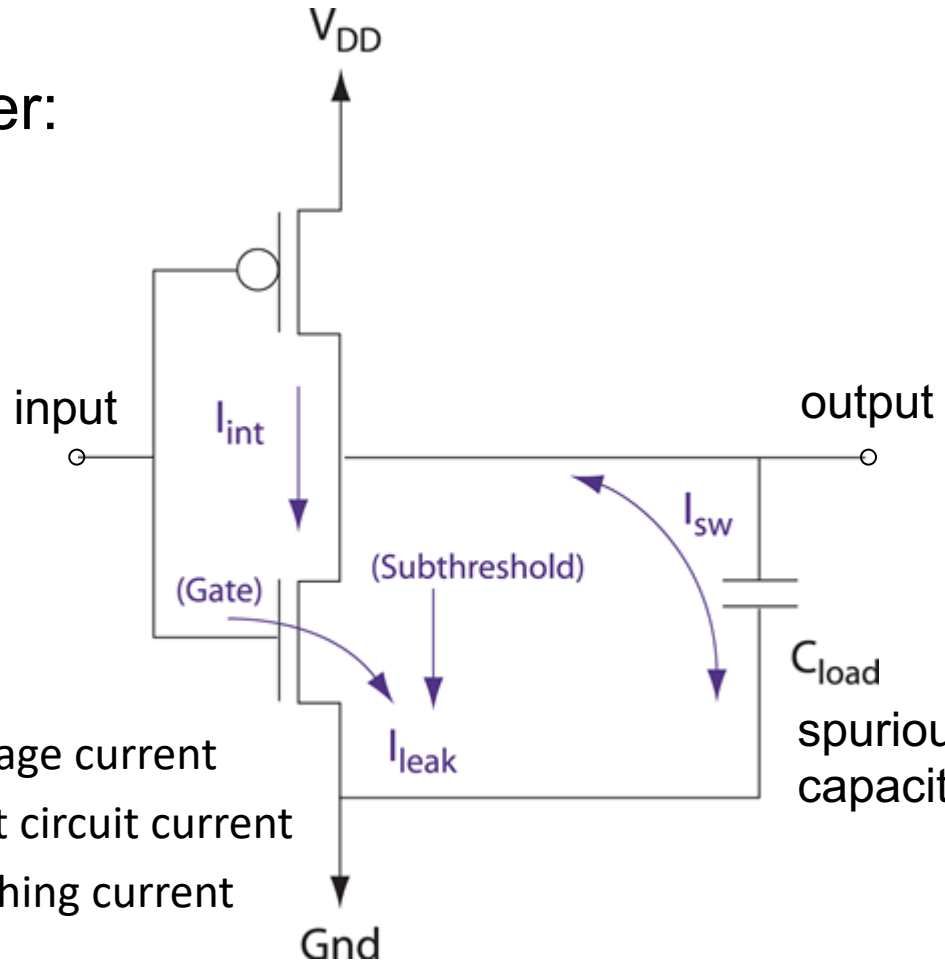
In some cases, faster execution also means less energy, but the opposite may be true if power has to be increased to allow for a faster execution.

Low Power vs. Low Energy

- Power and energy impact system design in different ways
- Minimizing the *power consumption (= voltage x current)* is important for
  - the design of the power supply and voltage regulators
  - the dimensioning of interconnect between power supply and components (e.g., reduced power enables the use of thinner wires)
  - cooling (short term cooling)
    - high cost, limited space
- Minimizing the *energy consumption* is important due to
  - restricted availability of energy (e.g., in mobile systems or IoT devices)
  - limited battery capacities (only slowly improving)
  - very high costs of energy (energy harvesting, solar panels, maintenance/batteries)
  - long lifetimes, low temperatures

Power Consumption of a CMOS Gate

Inverter:

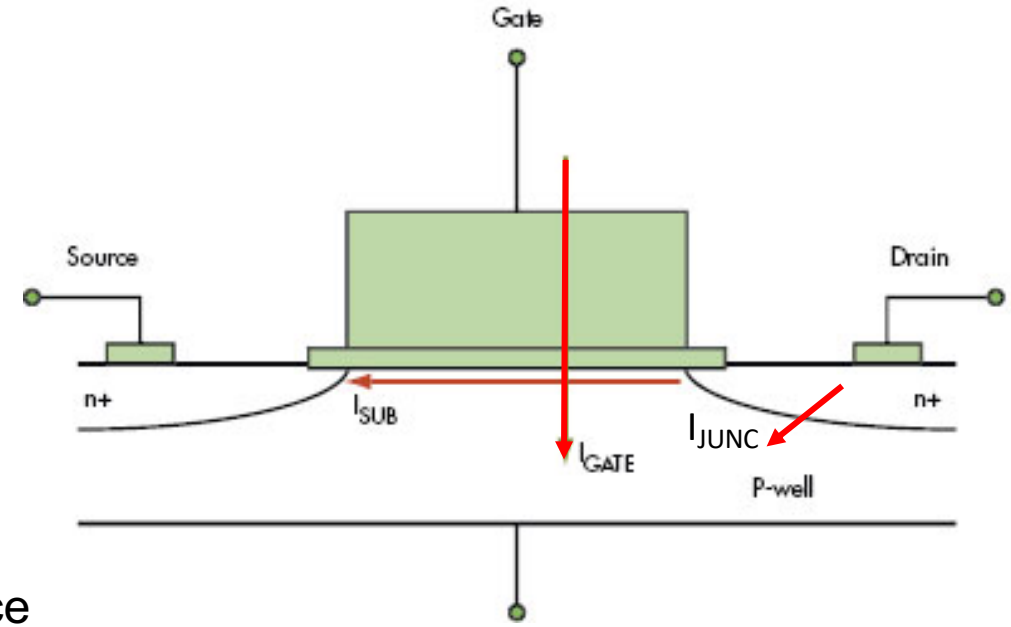


I_{leak} : leakage current

I_{int} : short circuit current

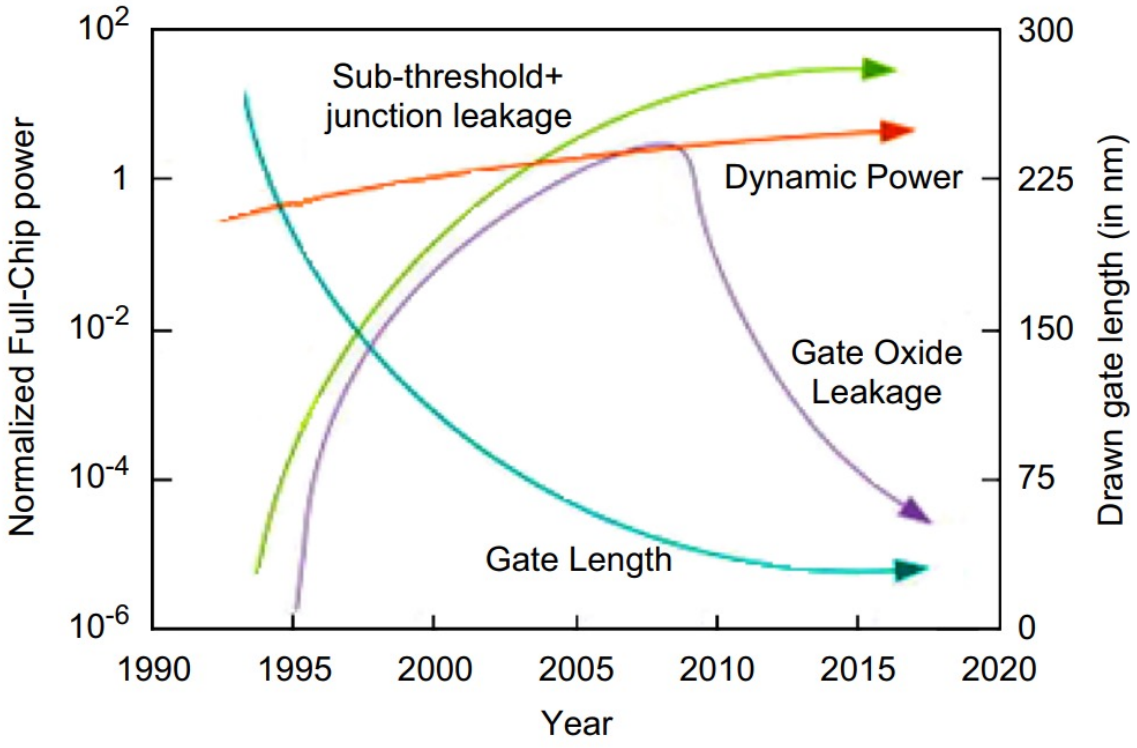
I_{sw} : switching current

subthreshold (I_{SUB}), junction (I_{JUNC}) and gate-oxide (I_{GATE}) leakage



Main Sources of Power Consumption of a CMOS Processors

- *Dynamic* power consumption:
  - (Dis)charging capacitances while switching
  - Short-circuit power consumption due to a short-circuit path between supply rails while switching
- *Static* power consumption:
  - Gate-oxide, subthreshold, and junction leakage while nothing happens (i.e., no clock, no switching of gate inputs)



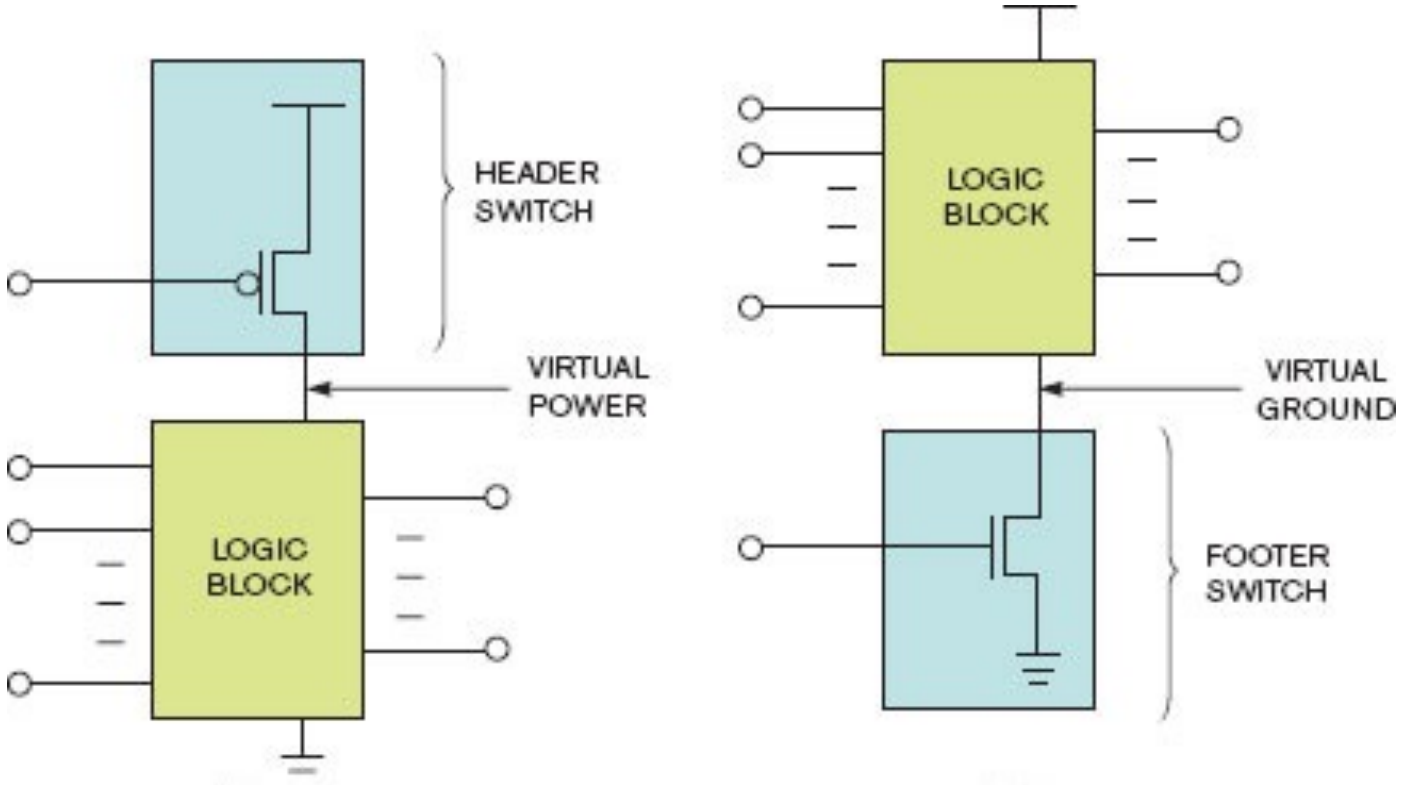
[J. Xue, T. Li, Y. Deng, Z. Yu, Full-chip leakage analysis for 65 nm CMOS technology and beyond, Integration VLSI J. 43 (4) (2010) 353–364]

Techniques to Reduce Static Power

Power Supply Gating

Power gating is one of the most effective ways to reduce static power consumption (leakage)

- Idea: Cut power supply off when components are unused (duty cycling, low-power modes)



Techniques to Reduce Dynamic Power

Voltage Scaling: A Simple Analytical Model

Average 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 of CMOS circuits:

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

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

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

But decreasing V_{dd} also increases the gate delay τ reciprocally.

Maximal frequency f_{max} decreases linearly with decreasing V_{dd} (i.e., "reducing the voltage makes the system slower").

$$f_{max} \sim \frac{1}{\tau} \sim \frac{V_{dd}}{C_L}$$

Voltage Scaling: A Simple Analytical Model

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

This is the energy per cycle.

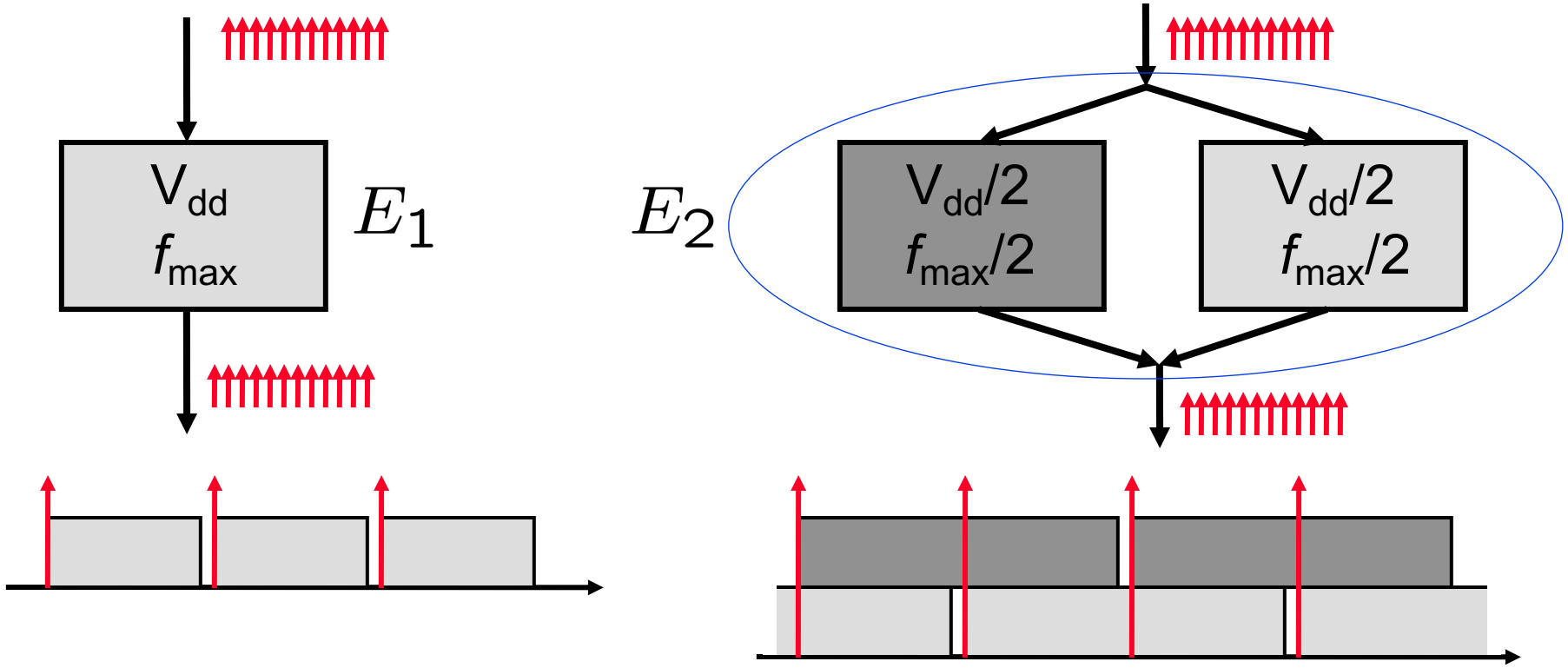
$$E \sim \alpha C_L V_{dd}^2 f t = \overbrace{\alpha C_L V_{dd}^2}\ (\#\text{cycles})$$

This is the energy needed for executing a task that requires #cycles (i.e., number of 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*

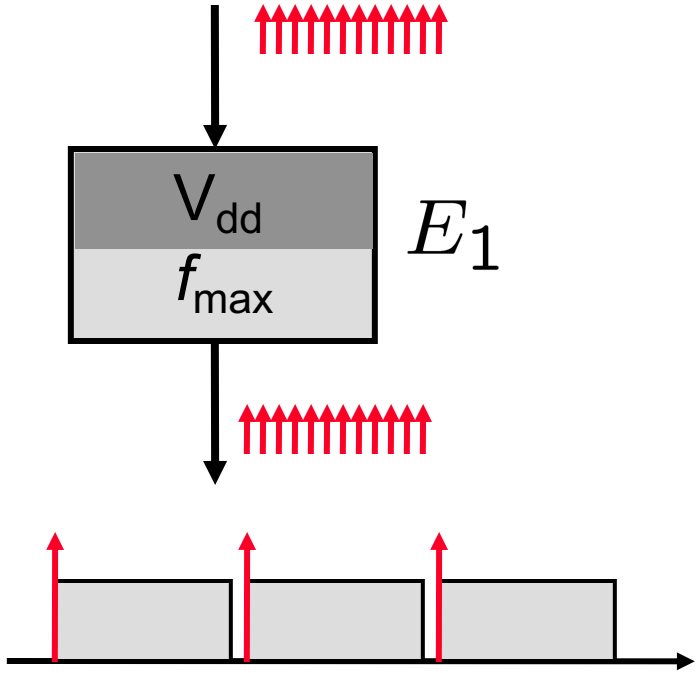
Parallelism

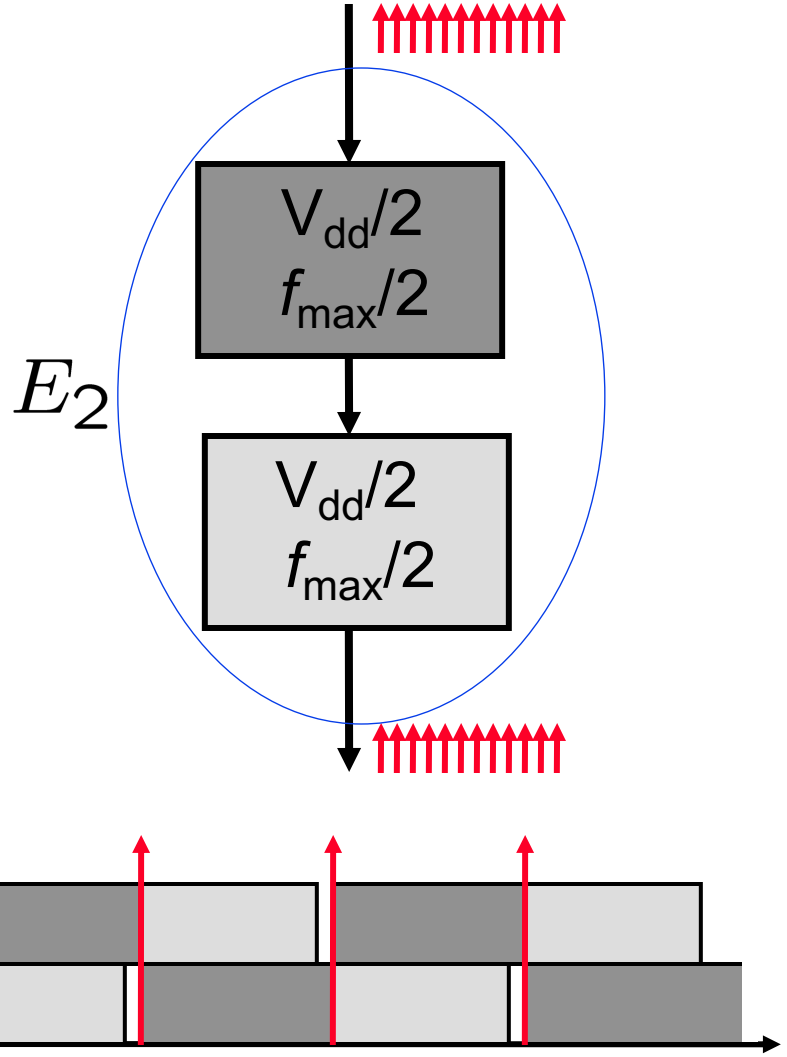


$$E \sim V_{dd}^2 \,(\#\text{cycles})$$

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

Pipelining



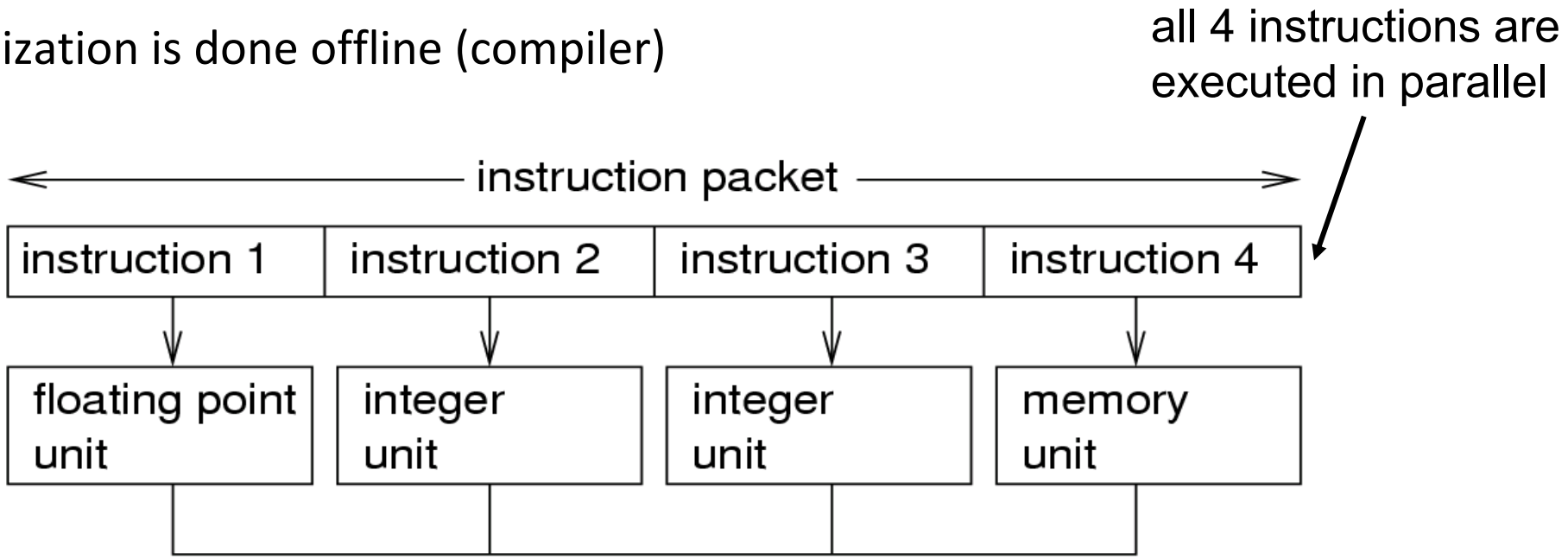


$$E \sim V_{dd}^2 \ (\#\text{cycles})$$

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

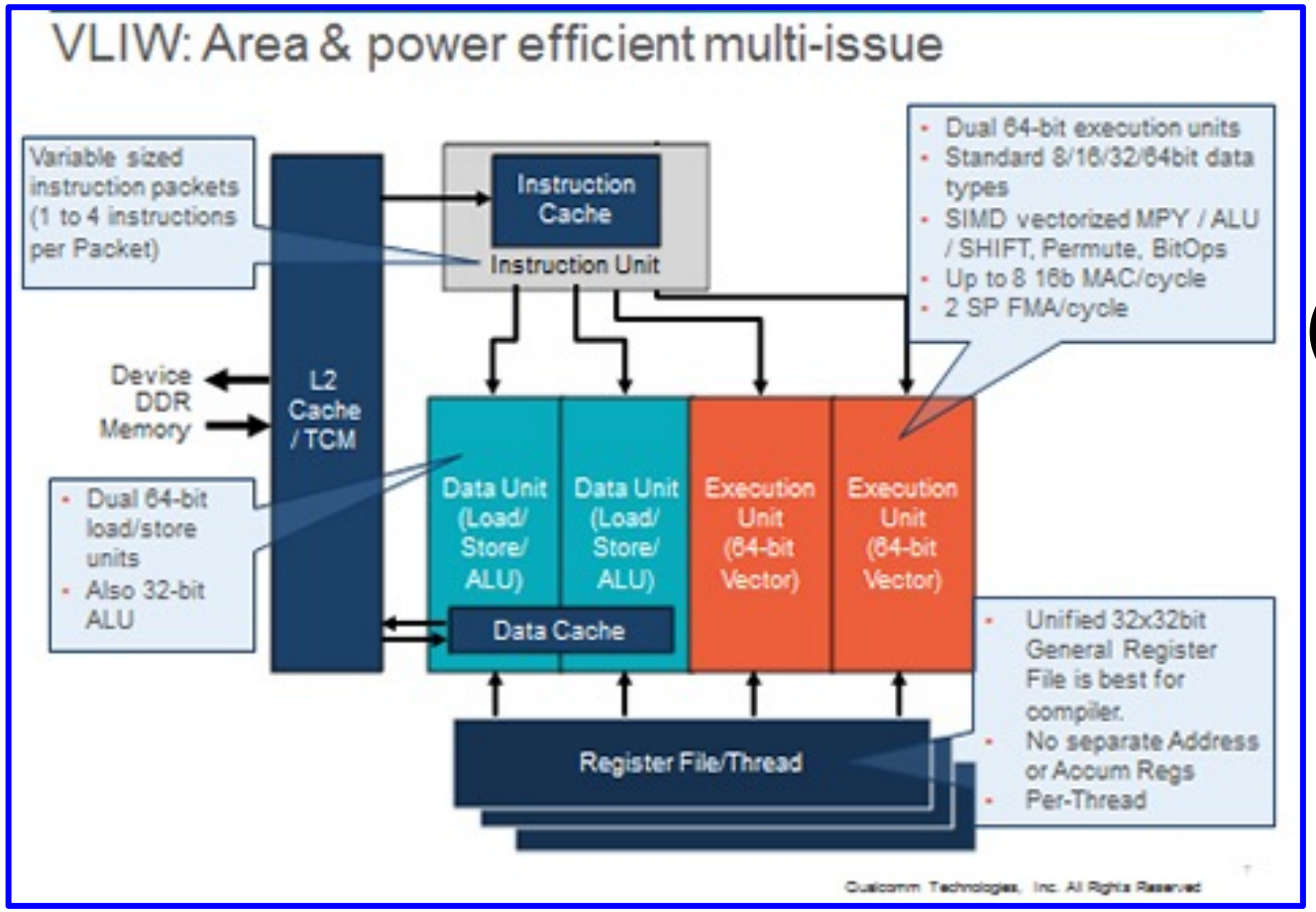
VLIW (Very Long Instruction Word) Architectures

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



Example: Qualcomm Hexagon

Hexagon DSP



Snapdragon 835 (Galaxy S8)

So far: *Statically* decrease voltage and frequency

Next: *Dynamically* change voltage and frequency

Dynamic Voltage and Frequency Scaling (DVFS)

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

energy per cycle

reduce voltage -> reduce energy per task

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

$$f \sim \frac{1}{\tau} \sim V_{dd}$$

reduce voltage -> reduce clock frequency

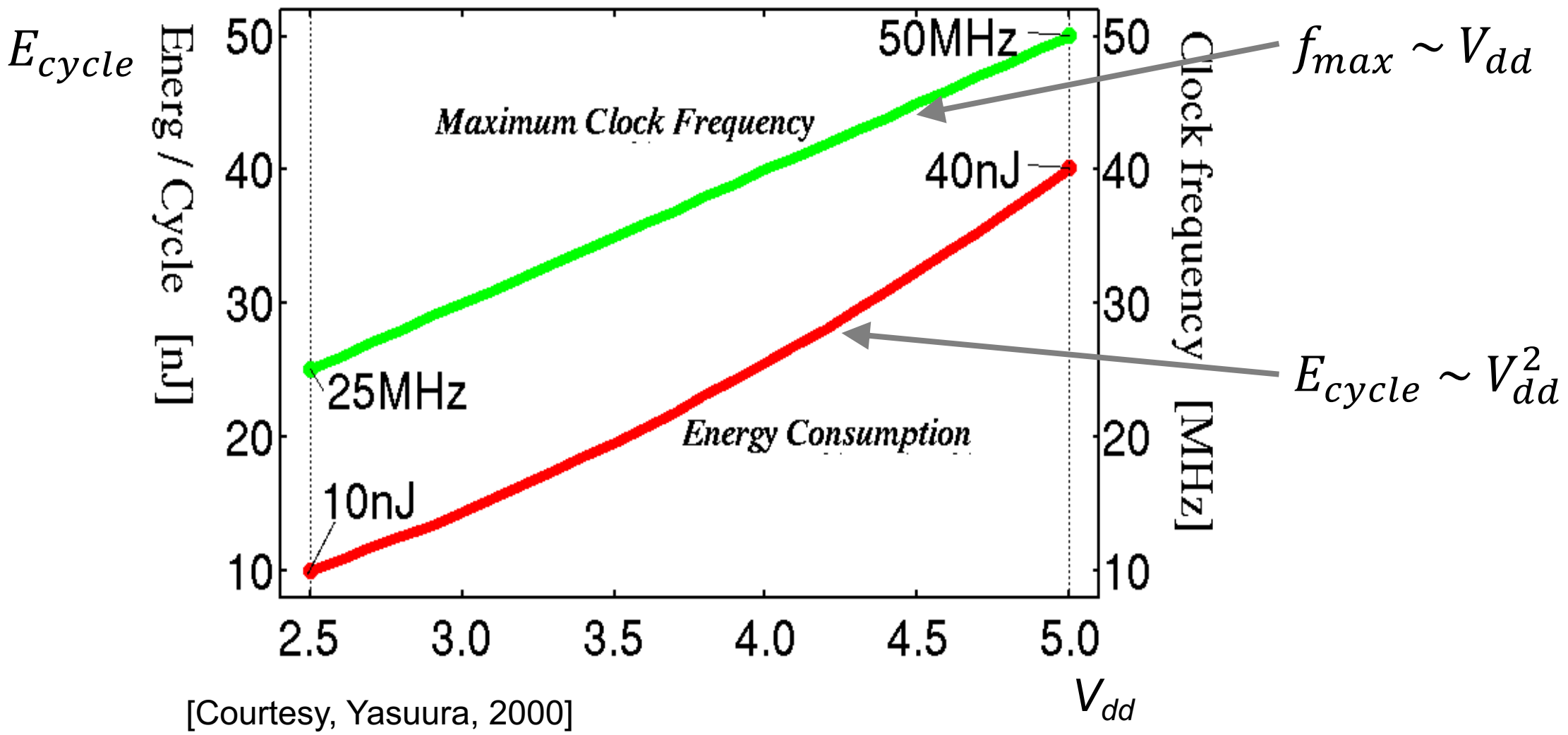
maximum frequency of operation

gate delay

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: Dynamic Voltage and Frequency Scaling

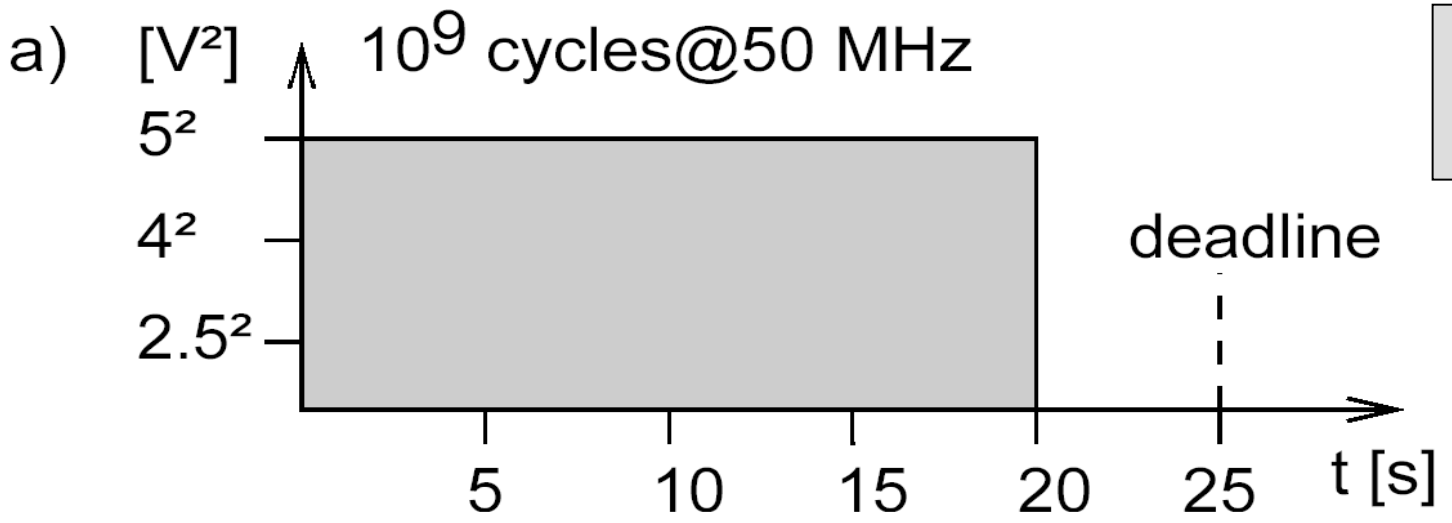


[Courtesy, Yasuura, 2000]

Example: DVFS – Complete Task as Early as Possible

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

We suppose a task that needs 10^9 cycles to execute within 25 seconds.

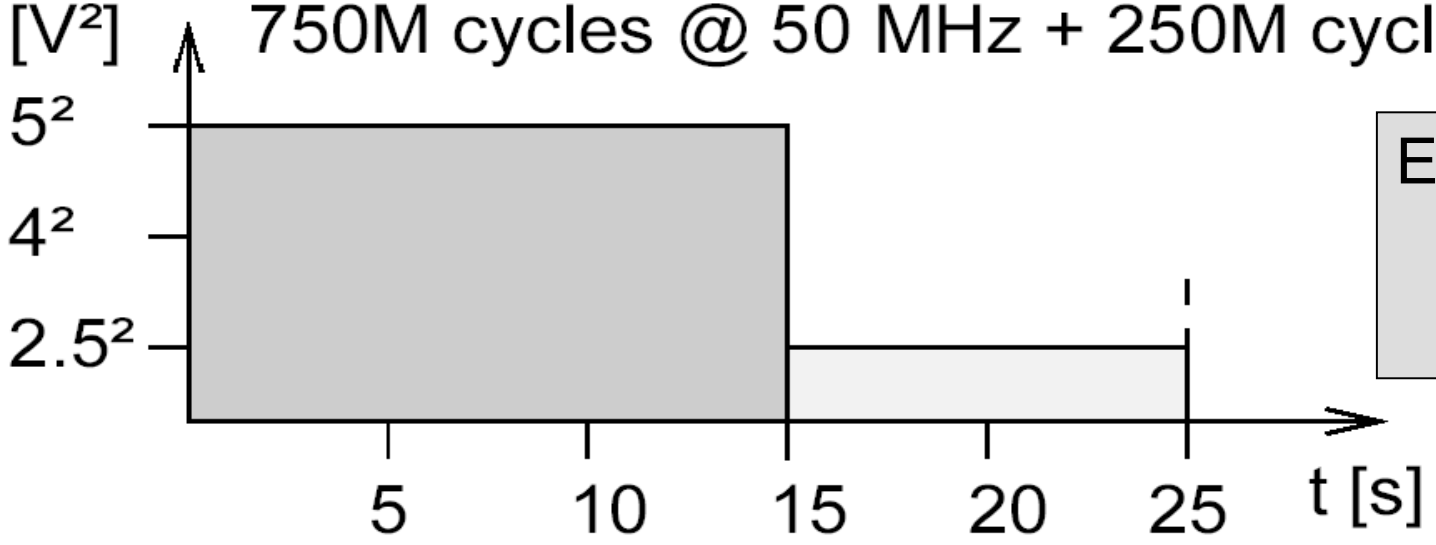


$E_a = 10^9 \times 40 \times 10^{-9}$
$= 40$ [J]

Example: DVFS – Use 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) 750M cycles @ 50 MHz + 250M cycles @ 25 MHz



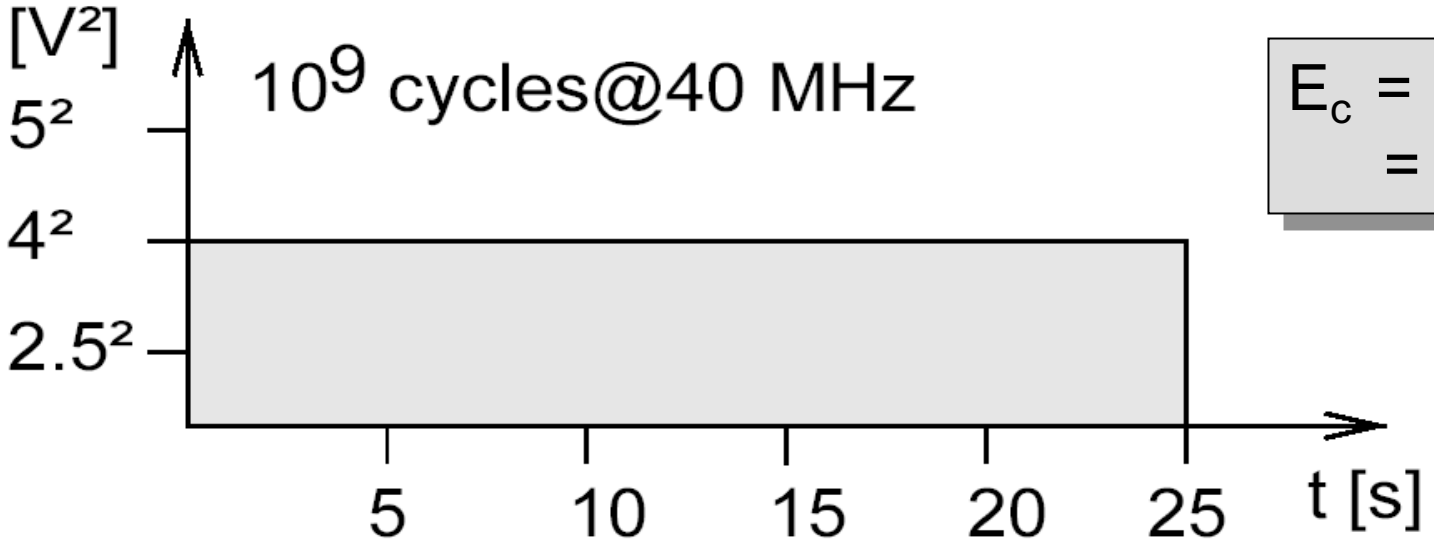
$E_b = 750\ 10^6 \times 40 \times 10^{-9} + 250\ 10^6 \times 10 \times 10^{-9}$

$= 32.5$ [J]

Example: DVFS – Use One Voltage

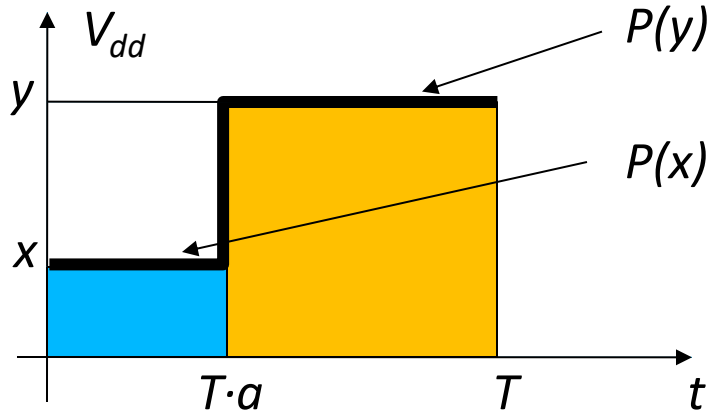
V_{dd} [V]	5.0	4.0	2.5
Energy per cycle [nJ]	40	25	10
f_{max} [MHz]	50	40	25
cycle time [ns]	20	25	40

c)



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

DVFS: Optimal Strategy



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

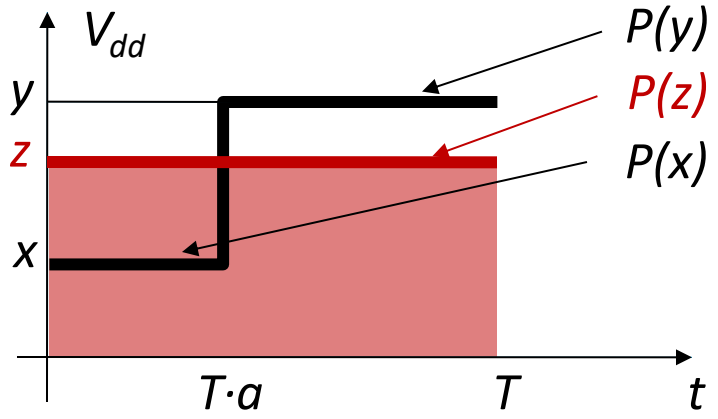
gate delay: $\tau \sim \frac{1}{V_{dd}}$

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: $T \cdot (P(x) \cdot a + P(y) \cdot (1\text{-}a))$

DVFS: Optimal Strategy



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

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

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

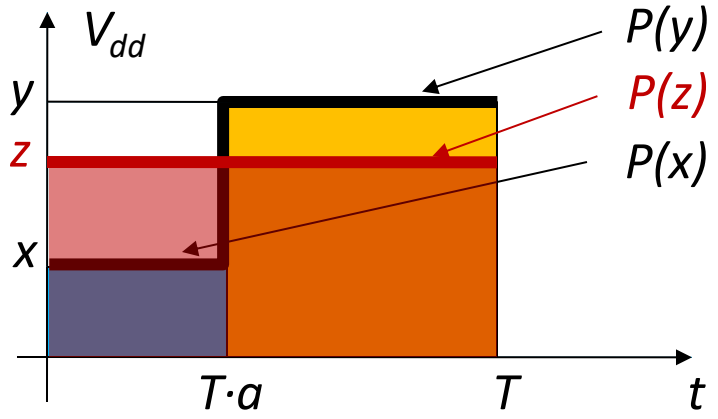
invariant: $\int V_{dd}(t)dt = \text{const.}$

Ensures that we compare the energies consumed by executing the same algorithm with the same #cycles

- ***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: $T \cdot P(z)$

DVFS: Optimal Strategy



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

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

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

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

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

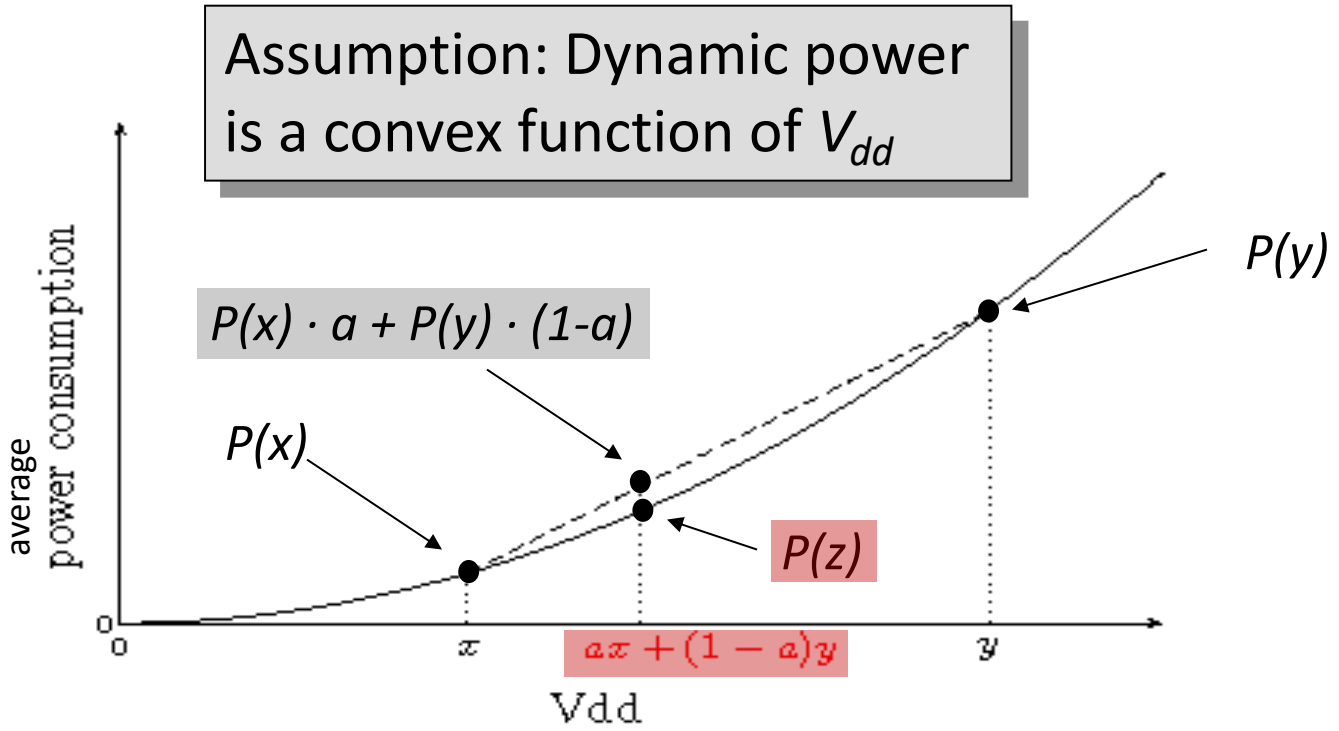
invariant: $\int V_{dd}(t)dt = \text{const.}$

Ensures that we compare the energies consumed by executing the same algorithm with the same #cycles

- ***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: $T \cdot P(z)$

DVFS: 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

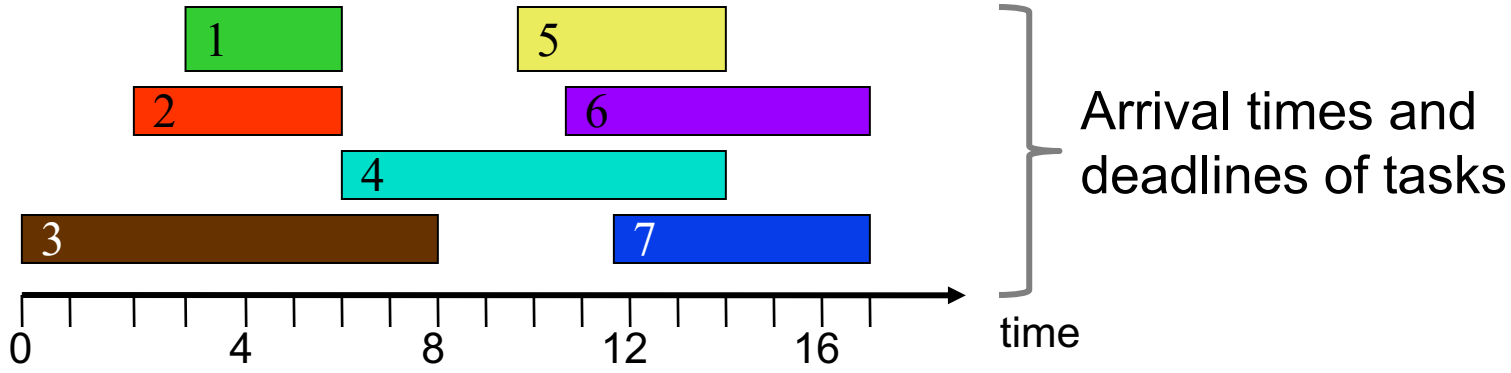
DVFS: Real-Time Offline Scheduling on One Processor

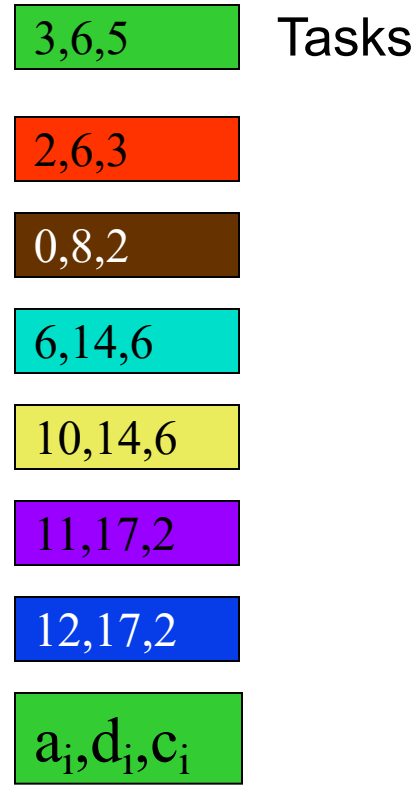
- Let us model a set of independent tasks as follows:
  - We suppose that a task $v_i \epsilon V$
    - requires c_i computation time at normalized processor frequency 1
    - arrives at time a_i
    - has (absolute) deadline constraint d_i

- 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 Reduced 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 Optimal DVFS Algorithm for Offline Scheduling





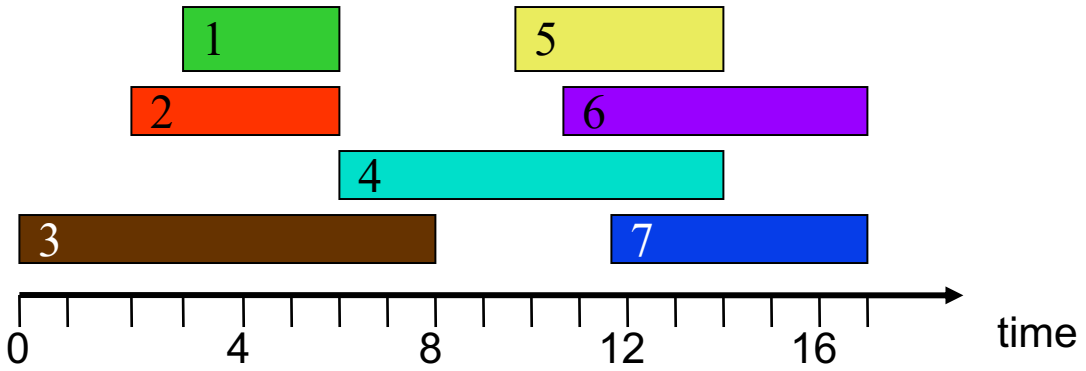
- Define ***intensity*** G([*z, z'*]) in some time interval [*z, z'*]:
  - average accumulated execution time of all tasks that have arrival and deadline in [*z, z'*] relative to the length of the interval *z'-z*

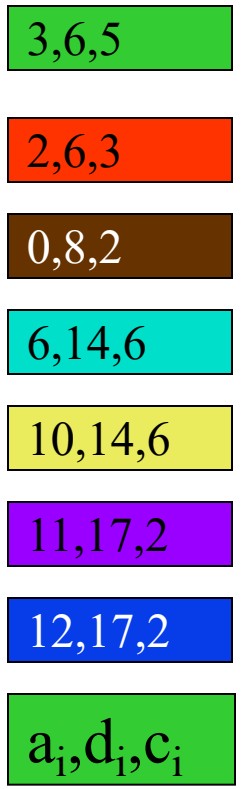
$$V'([z,z']) = \{v_i \in V \ : \ z \leq a_i < d_i \leq z'\}$$

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

YDS Optimal DVFS 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

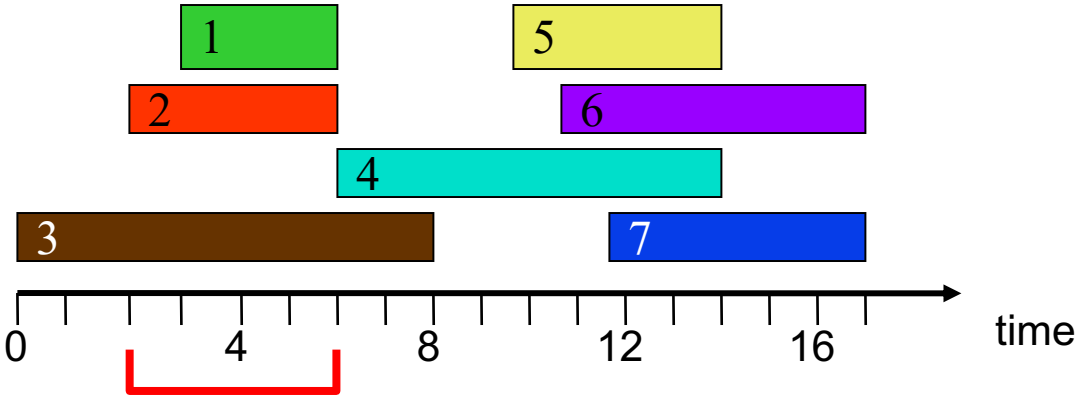
G([3,6]) =5/3, G([3,14]) = (5+6+6)/(14-3) = 17/11, G([3,17])=(5+6+6+2+2)/14=21/14

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 Optimal DVFS 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.



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

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

critical interval

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

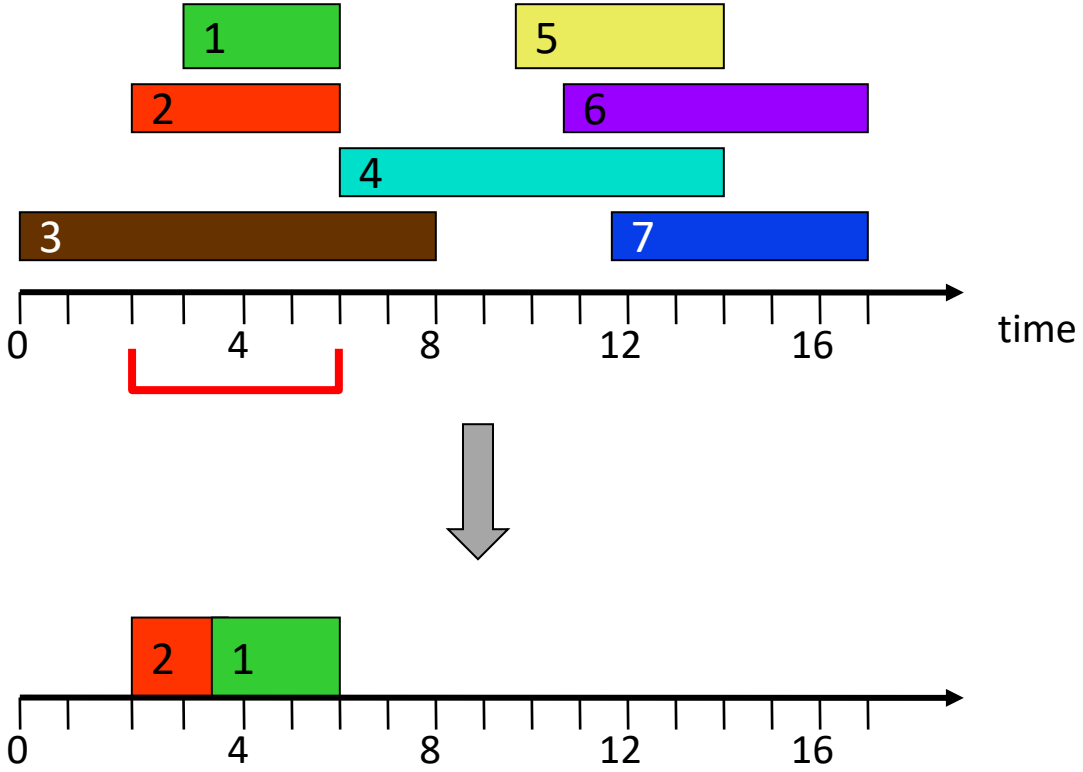
G([3,6]) =5/3, G([3,14]) = (5+6+6)/(14-3) = 17/11, G([3,17])=(5+6+6+2+2)/14=21/14

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 Optimal DVFS 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.



3,6,5

2,6,3

0,8,2

6,14,6

10,14,6

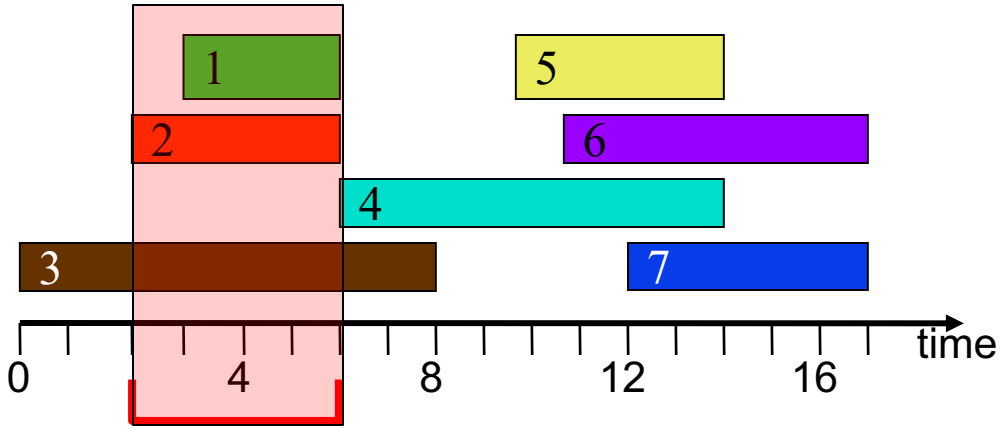
11,17,2

12,17,2

a_i,d_i,c_i

YDS Optimal DVFS Algorithm for Offline Scheduling

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



0,8,2

6,14,6

10,14,6

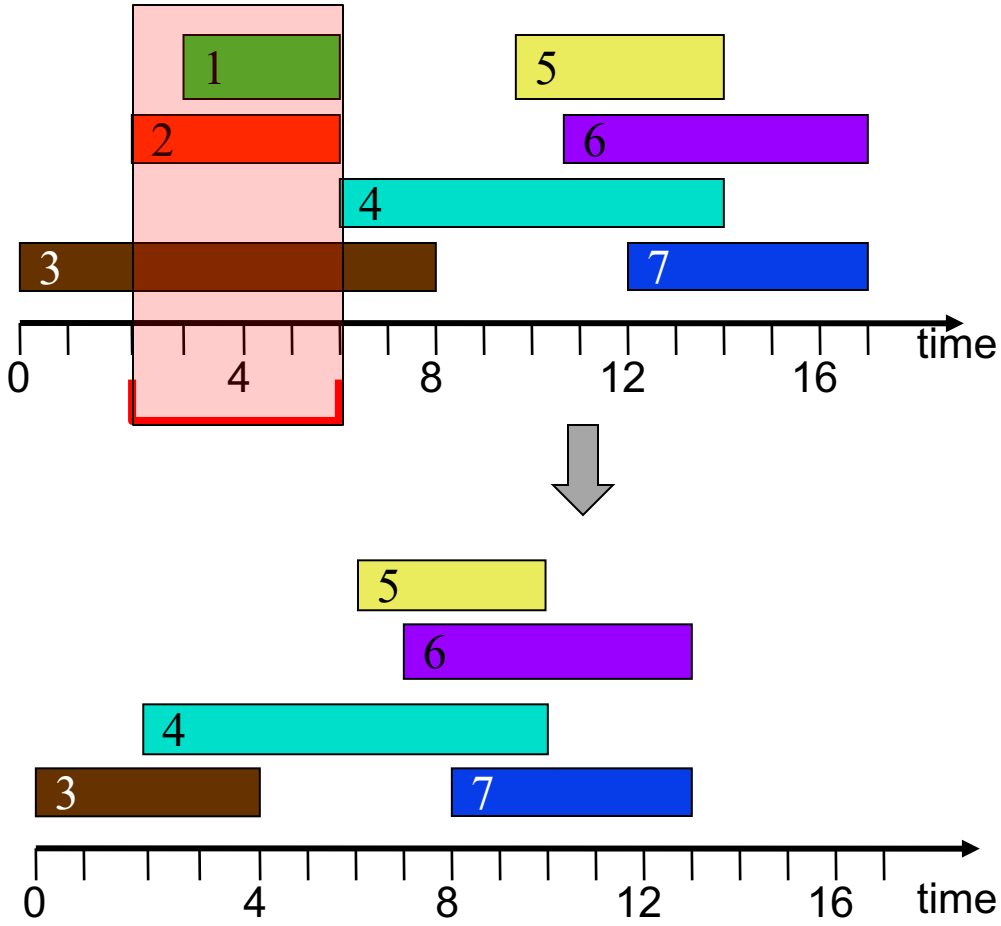
11,17,2

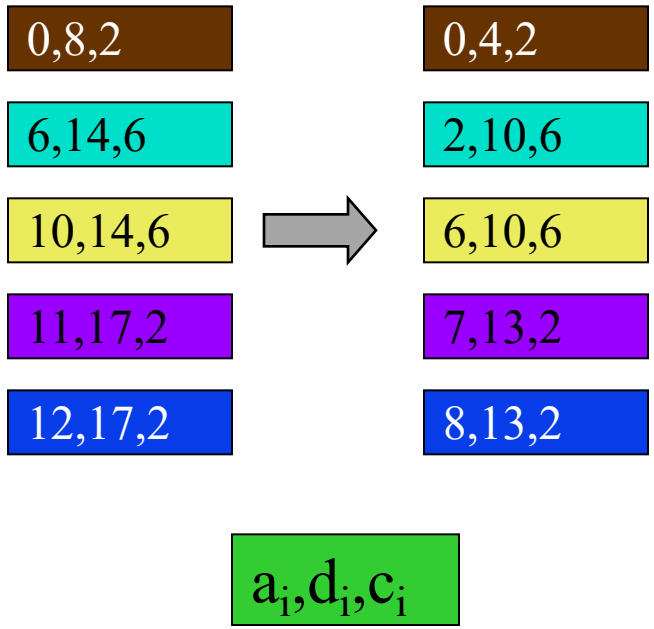
12,17,2

a_i,d_i,c_i

YDS Optimal DVFS Algorithm for Offline Scheduling

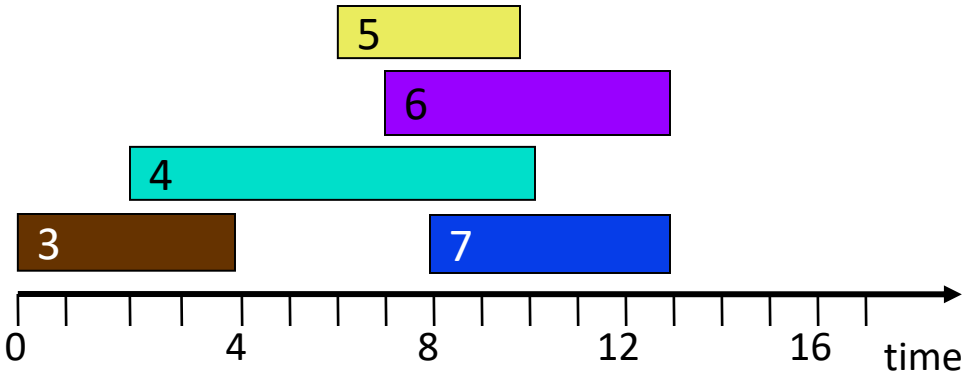
Step 2: Adjust the arrival times and deadlines by excluding the possibility to execute within the previous critical intervals.





YDS Optimal DVFS 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

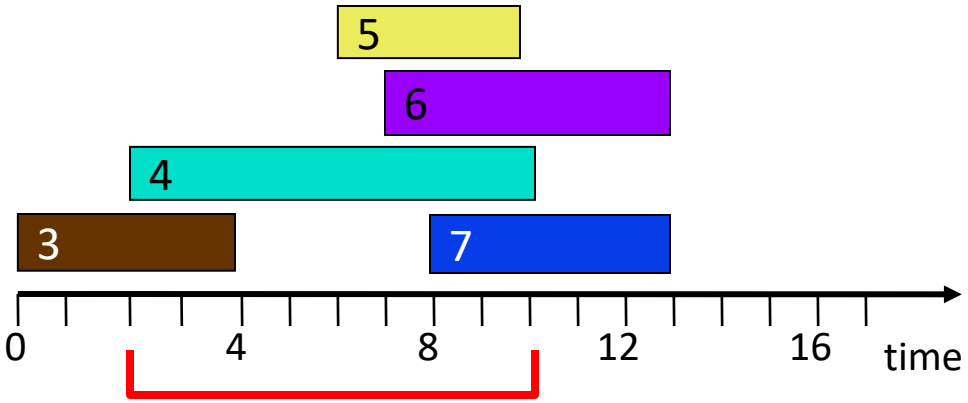
7,13,2

8,13,2

a_i, d_i, c_i

YDS Optimal DVFS 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

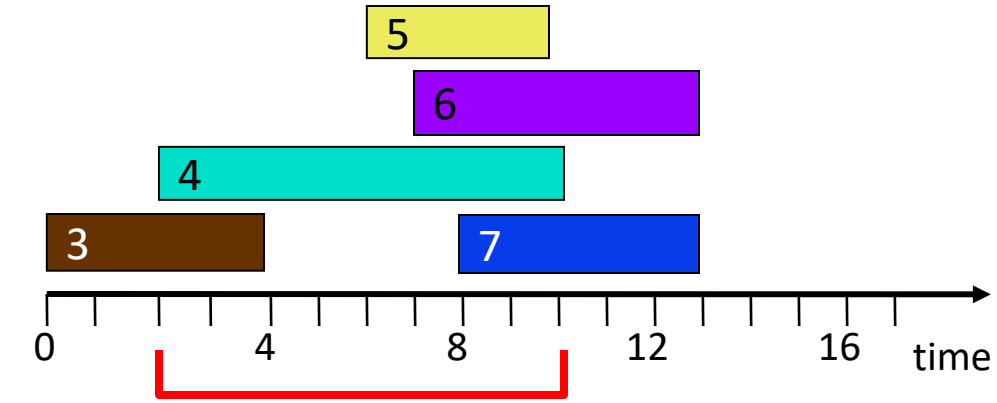
7,13,2

8,13,2

a_i, d_i, c_i

YDS Optimal DVFS Algorithm for Offline Scheduling

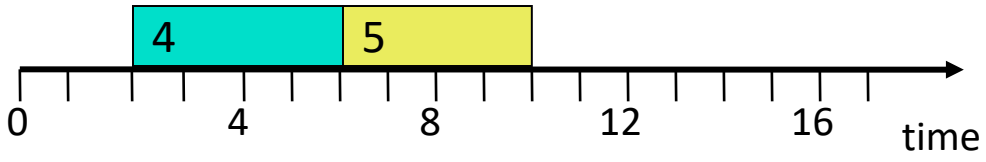
Step 3: Run the algorithm for the revised input again



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

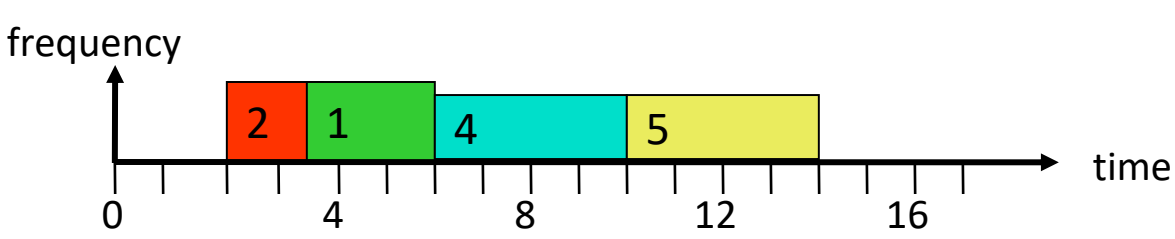
8,13,2

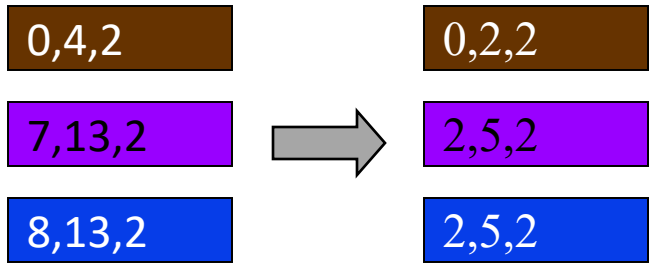
a_i, d_i, c_i

YDS Optimal DVFS Algorithm for Offline Scheduling

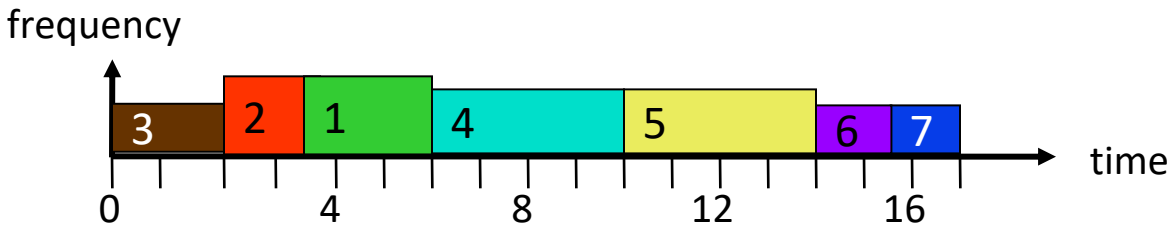
Step 3: Run the algorithm for the revised input again

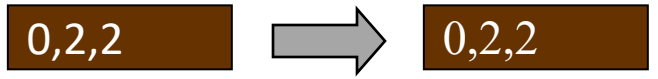
Step 4: Put pieces together





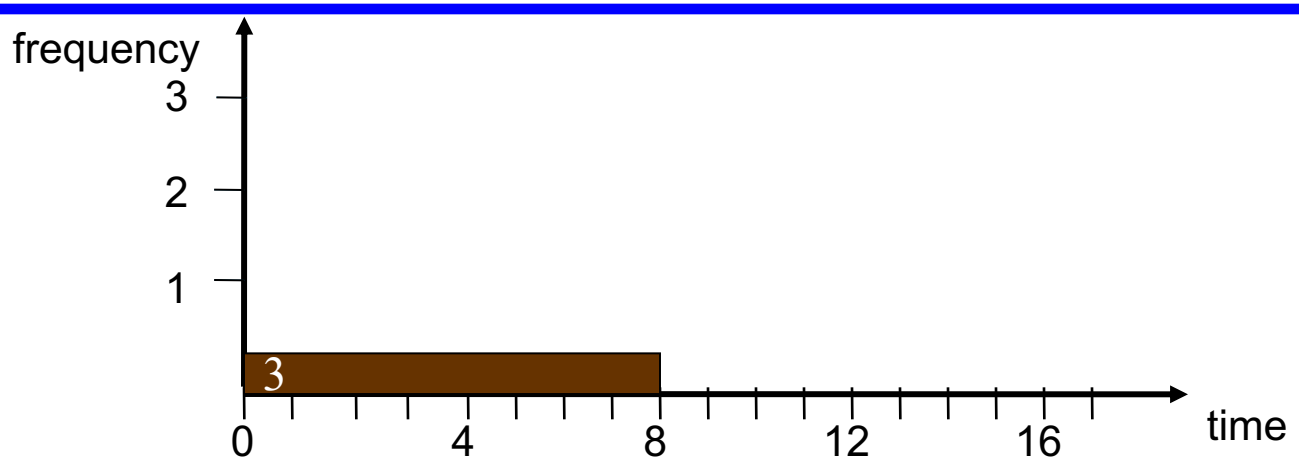
energy-optimal schedule





task	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

YDS Optimal DVFS Algorithm for Online Scheduling



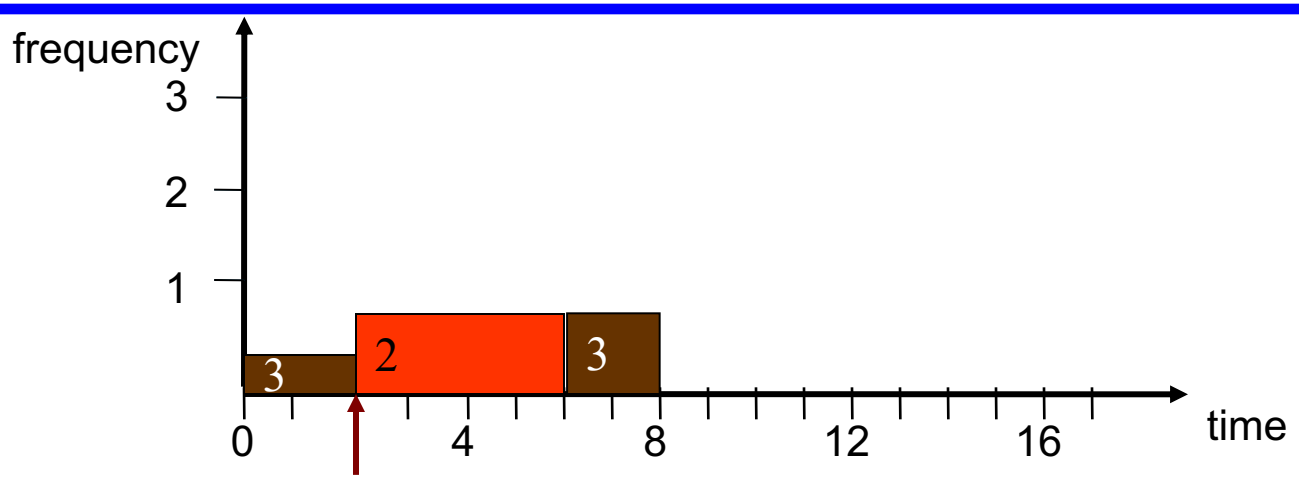
0,8,2

Continuously update to the best schedule for all arrived tasks:

Time 0: task v_3 is executed at 2/8

a_i,d_i,c_i

YDS Optimal DVFS Algorithm for Online Scheduling



2,6,3

0,8,2

Continuously update to the best schedule for all arrived tasks:

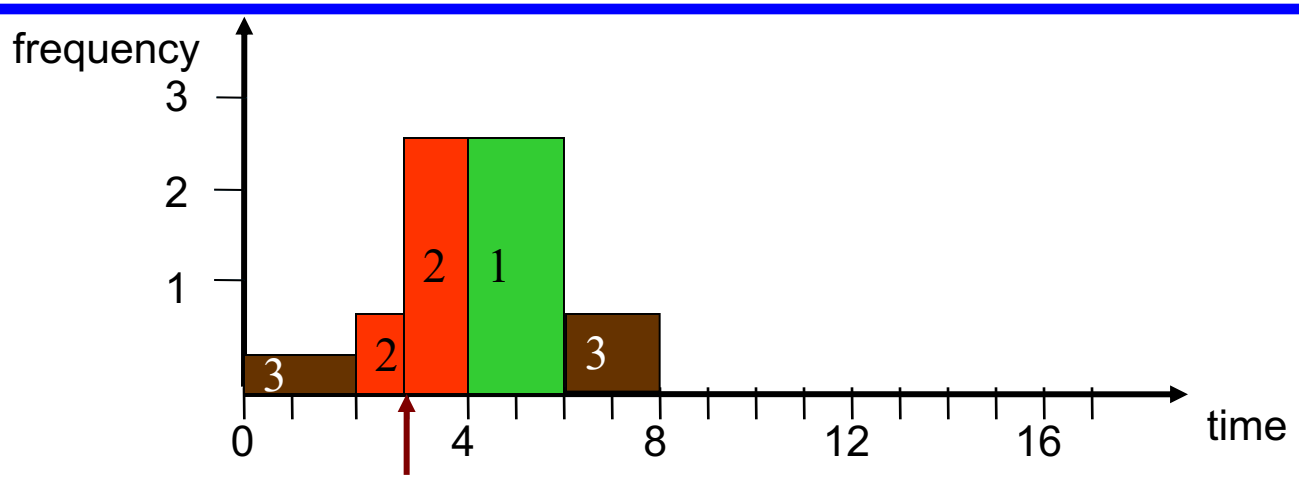
Time 0: task v_3 is executed at 2/8

Time 2: task v_2 arrives

- G([2,6]) = ¾, G([2,8]) = 4.5/6=3/4 => execute v_3 , v_2 at ¾

a_i,d_i,c_i

YDS Optimal DVFS Algorithm for Online Scheduling



3,6,5

2,6,3

0,8,2

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

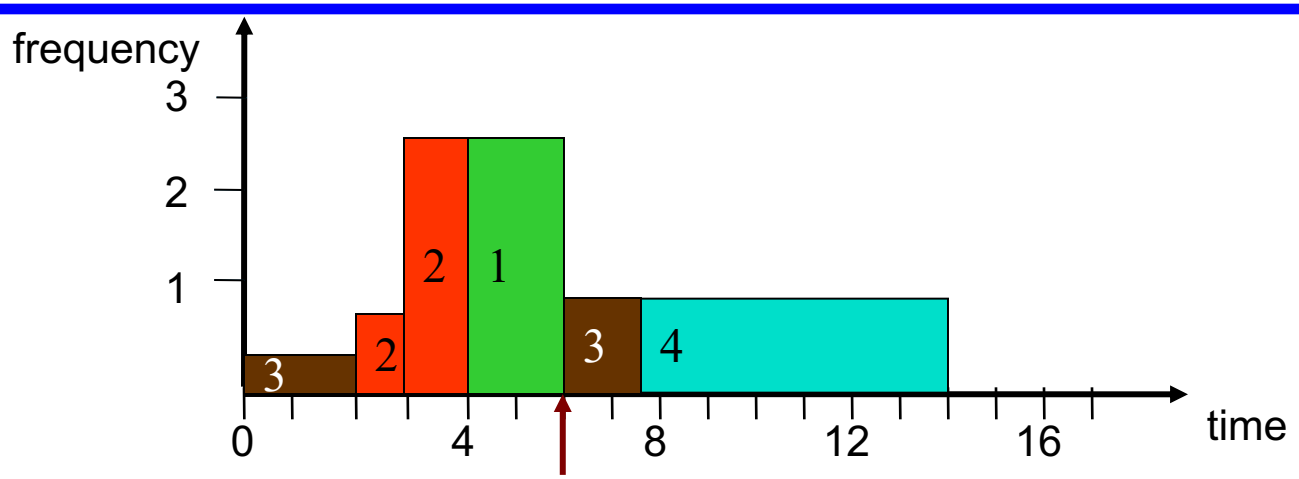
- G([2,6]) = ¾, G([2,8]) = 4.5/6=3/4 => execute v_3 , v_2 at ¾

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

a_i,d_i,c_i

YDS Optimal DVFS Algorithm for Online Scheduling



3,6,5

2,6,3

0,8,2

6,14,6

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_3 , v_2 at ¾

Time 3: task v_1 arrives

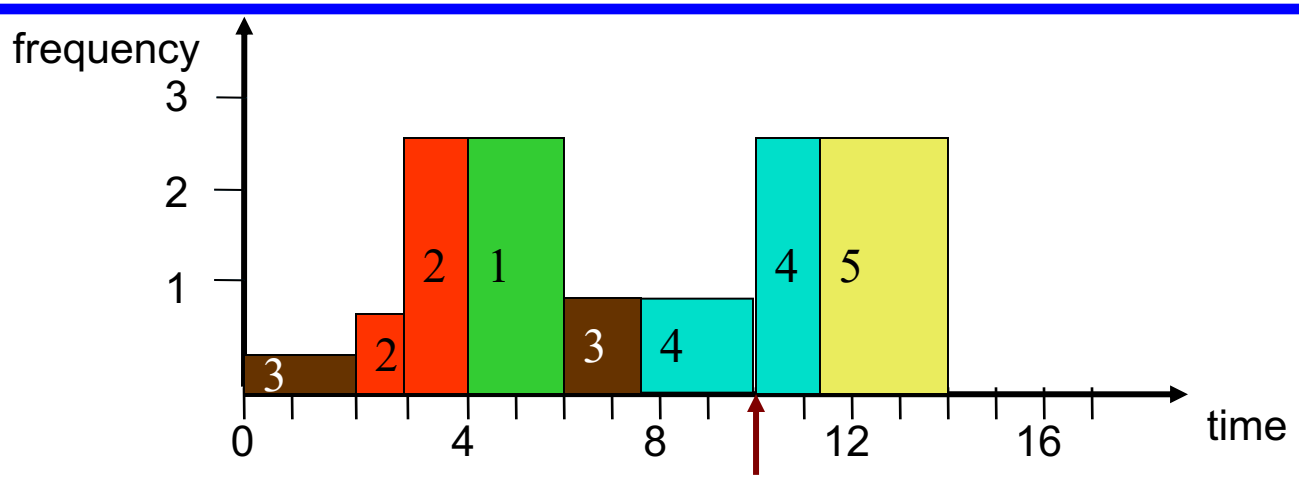
- G([3,6]) = (5+3-3/4)/3=29/12, G([3,8]) < G([3,6]) => execute v_2 and v_1 at 29/12

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

a_i,d_i,c_i

YDS Optimal DVFS Algorithm for Online Scheduling



3,6,5

2,6,3

0,8,2

6,14,6

10,14,6

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_8, 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

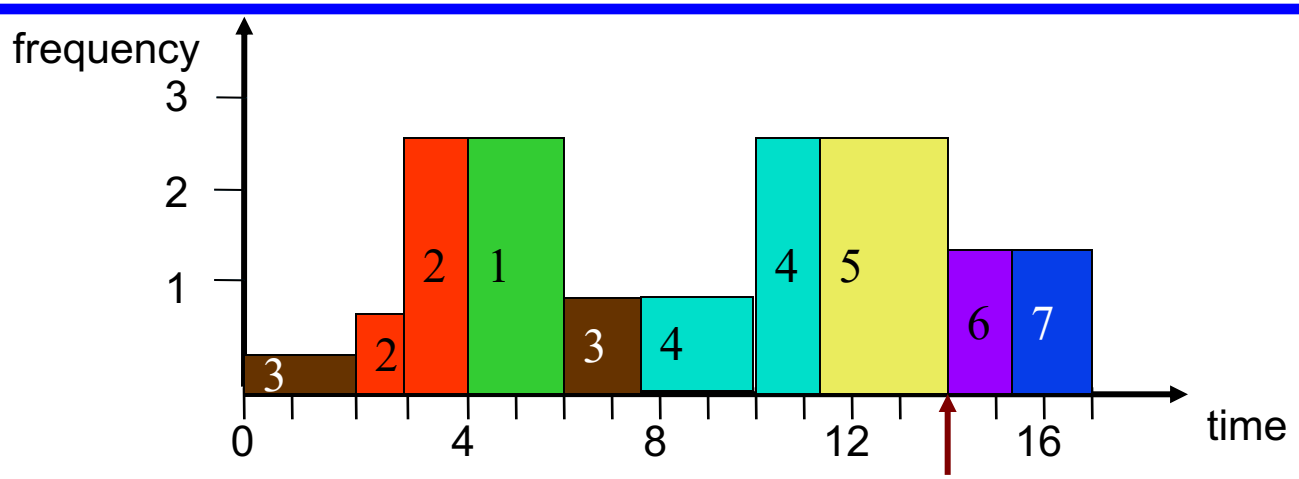
- G([6,8]) = 1.5/2, G([6,14]) = 7.5/8 => execute v_3 and v_4 at 15/16

Time 10: task v_5 arrives

- G([10,14]) = 39/16 => execute v_4 and v_5 at 39/16

a_i,d_i,c_i

YDS Optimal DVFS Algorithm for Online 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

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_8, 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 the 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.