Embedded Systems

9. Power and Energy

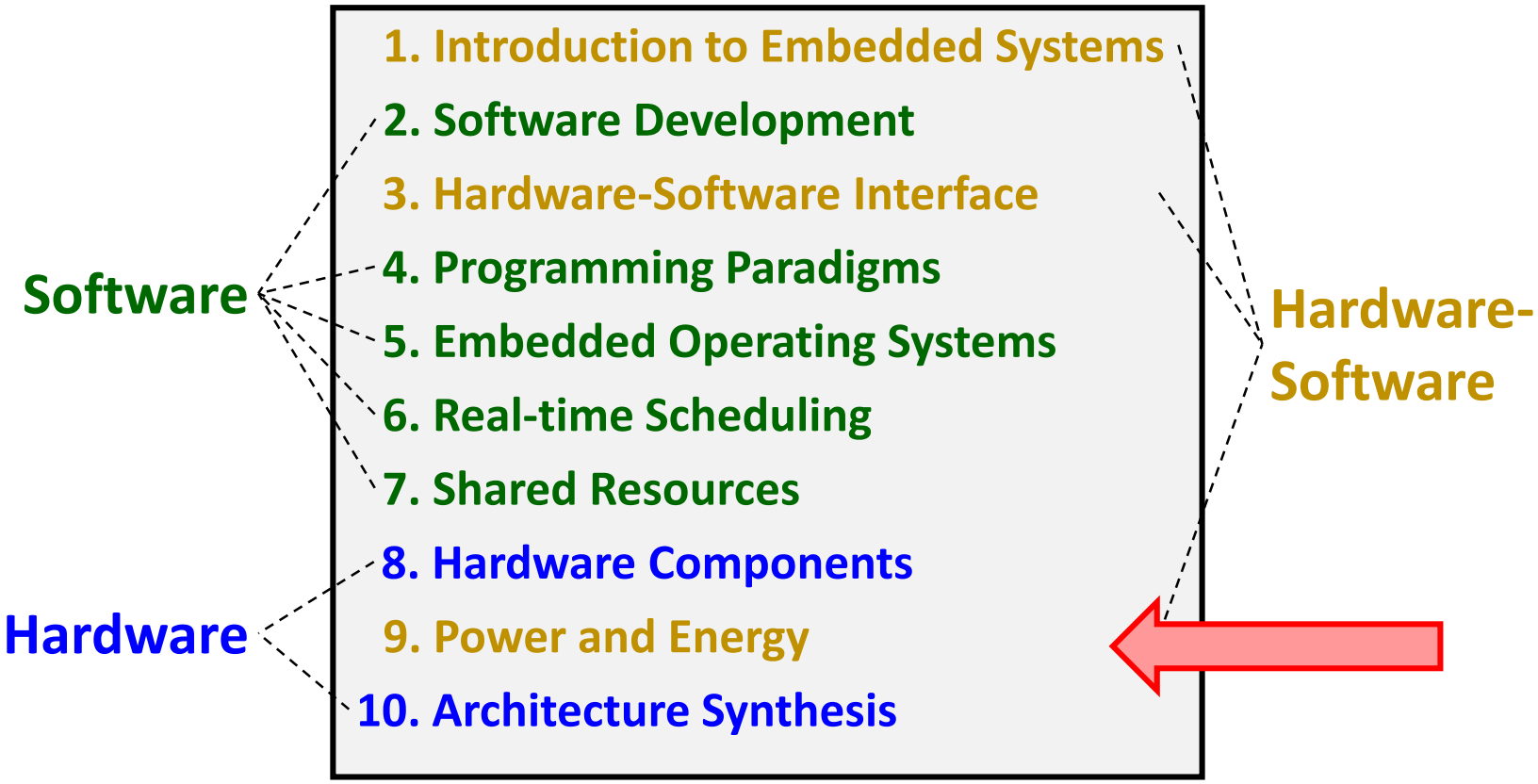
© Lothar Thiele

Computer Engineering and Networks Laboratory

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture Overview



General Remarks

Power and Energy Consumption

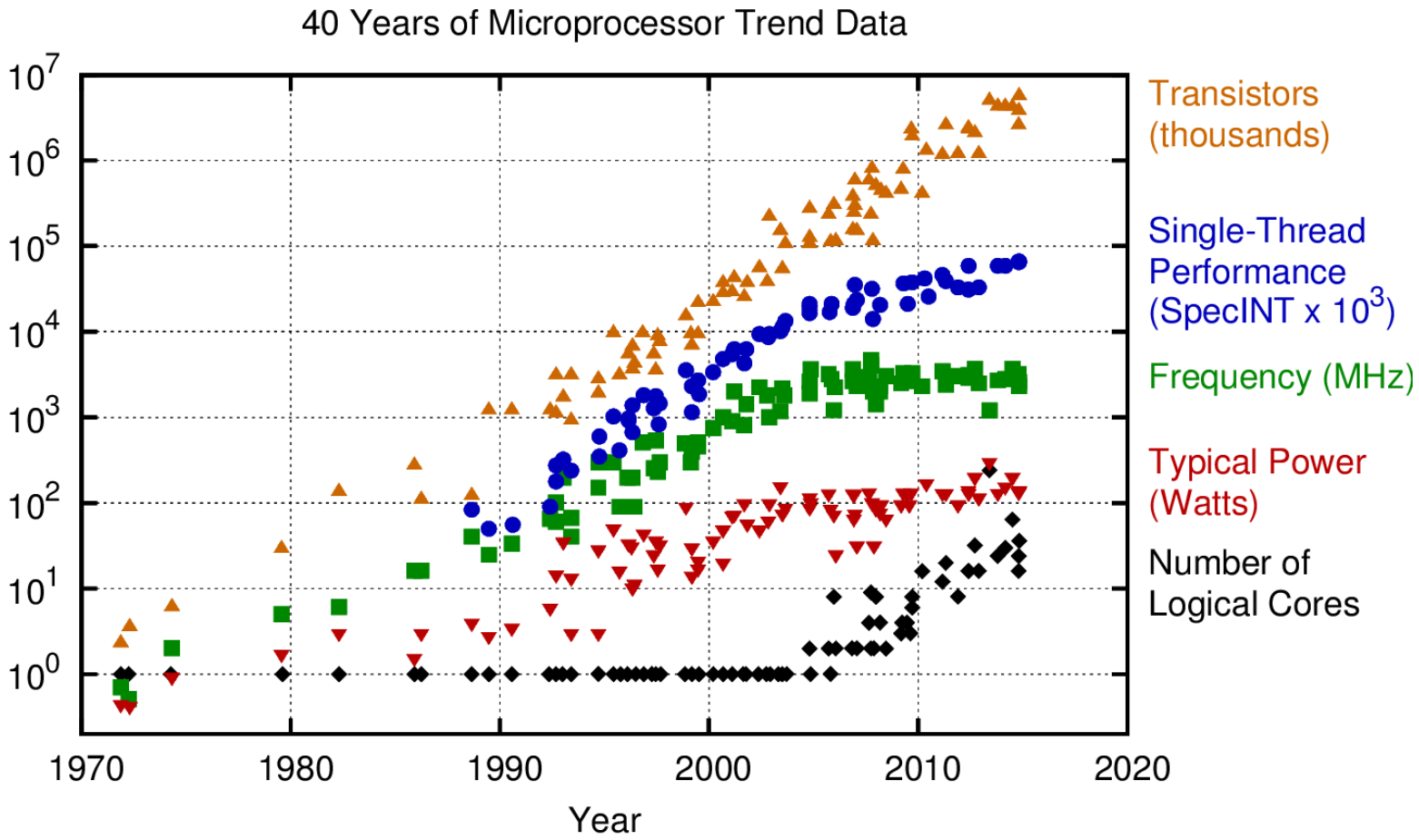
- Statements that are true since a decade or longer:

  „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]

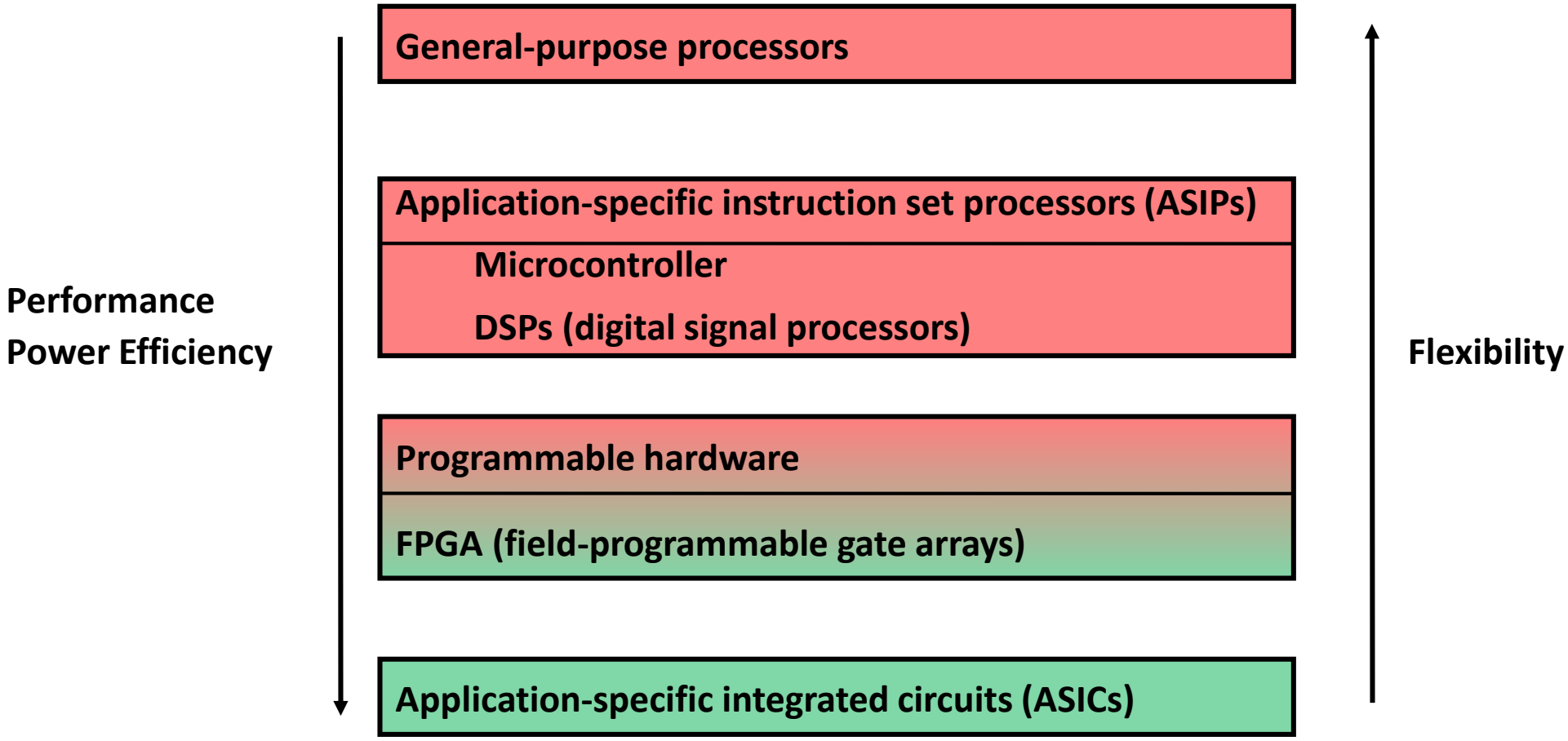
- Main ***reasons*** are:
  - power provisioning is expensive
  - battery capacity is growing only slowly
  - devices may overheat
  - energy harvesting (e.g. from solar cells) is limited due to the relatively low energy available density

Some Trends



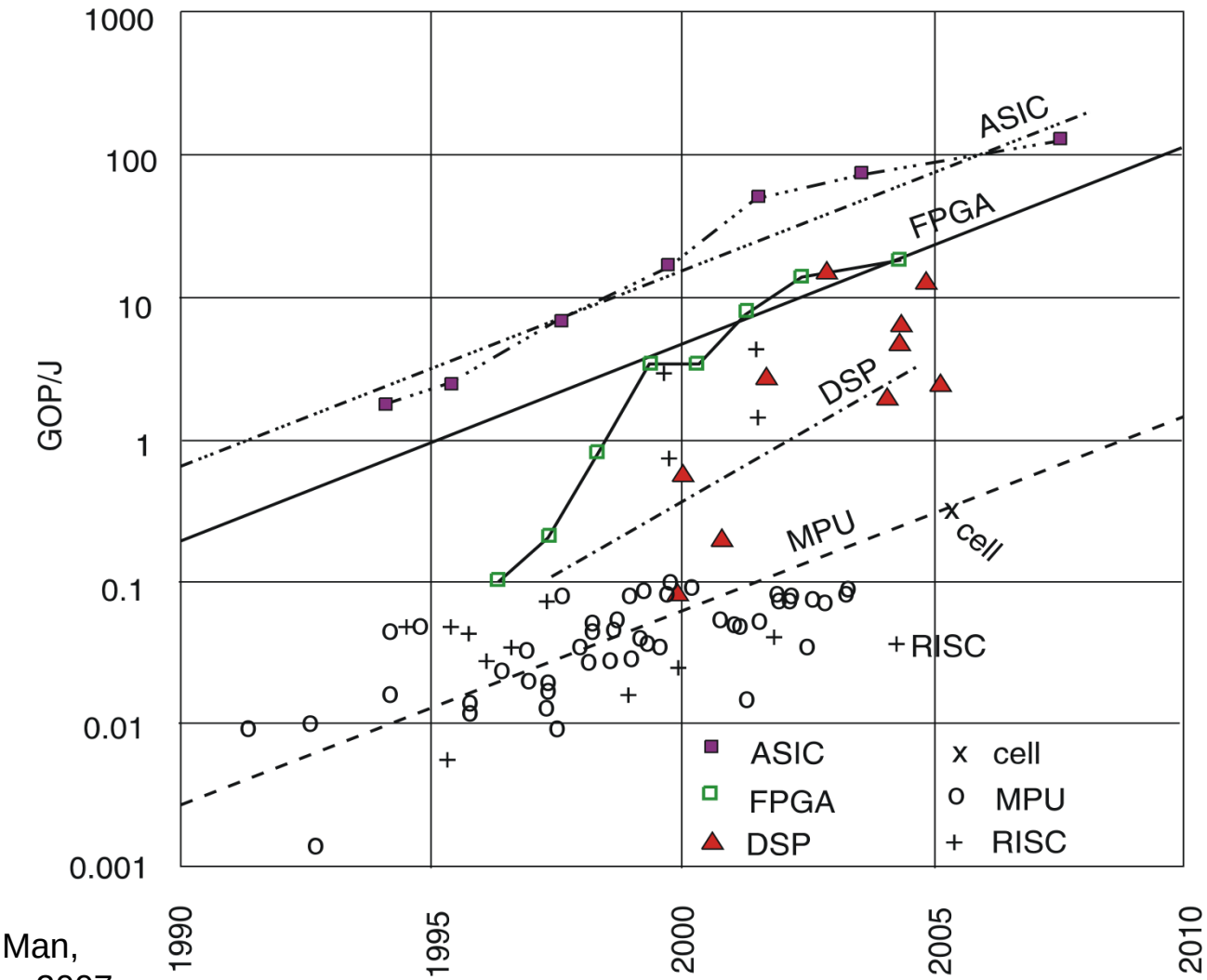
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Implementation Alternatives



Energy Efficiency

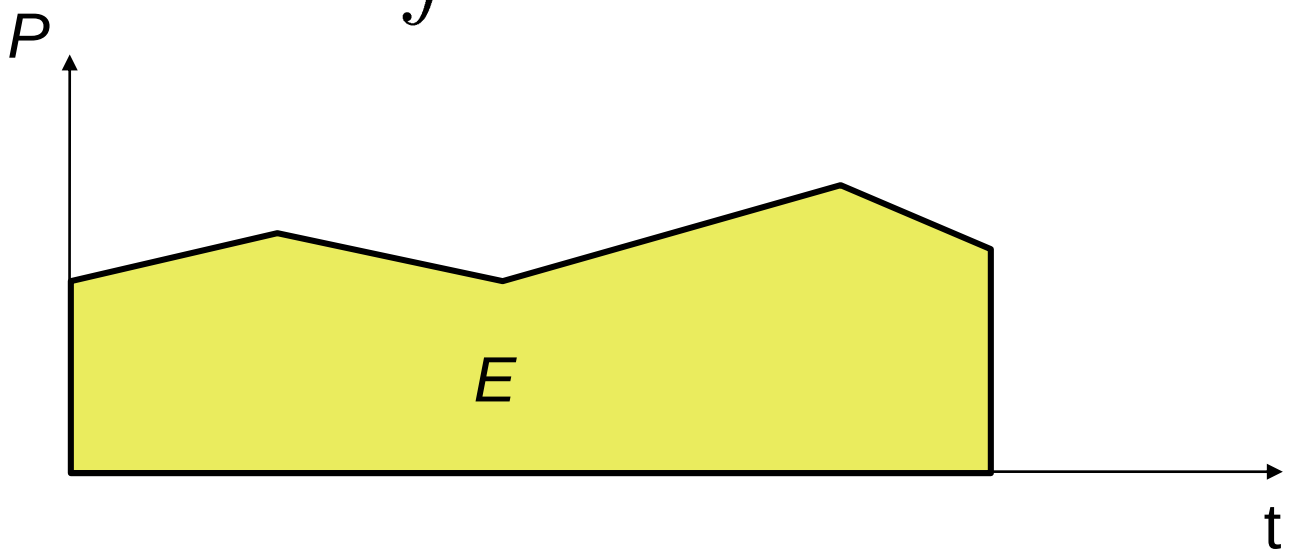
- It is necessary to ***optimize HW and SW.***
- Use ***heterogeneous architectures*** in order to adapt to required performance and to class of application.
- Apply ***specialization techniques***.



Power and Energy

Power and Energy

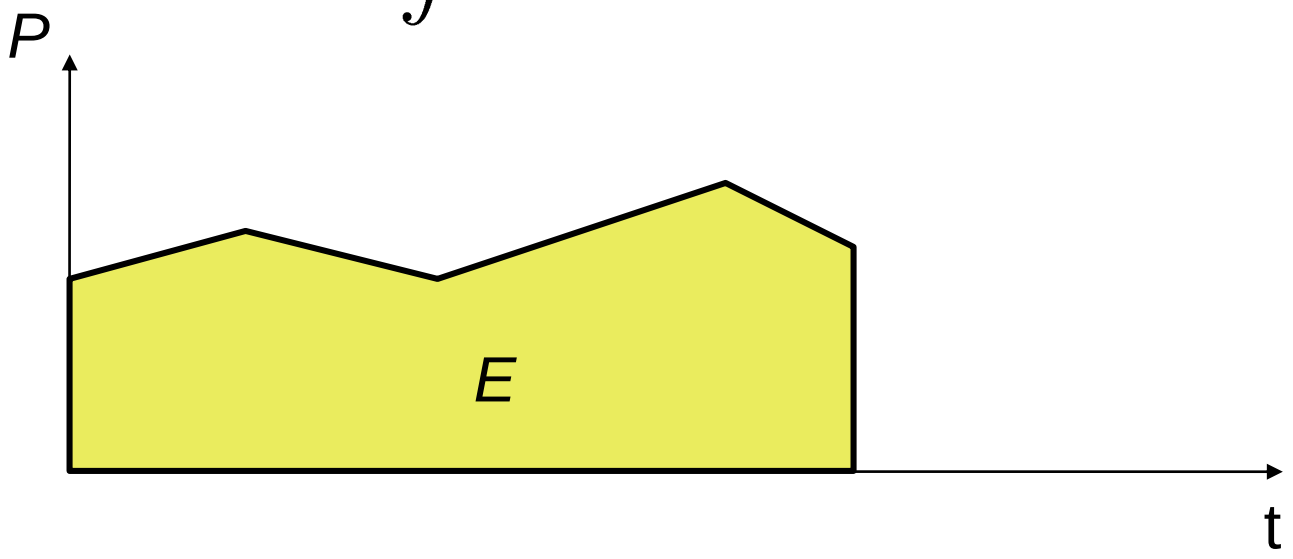
$$E = \int P(t)dt$$



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

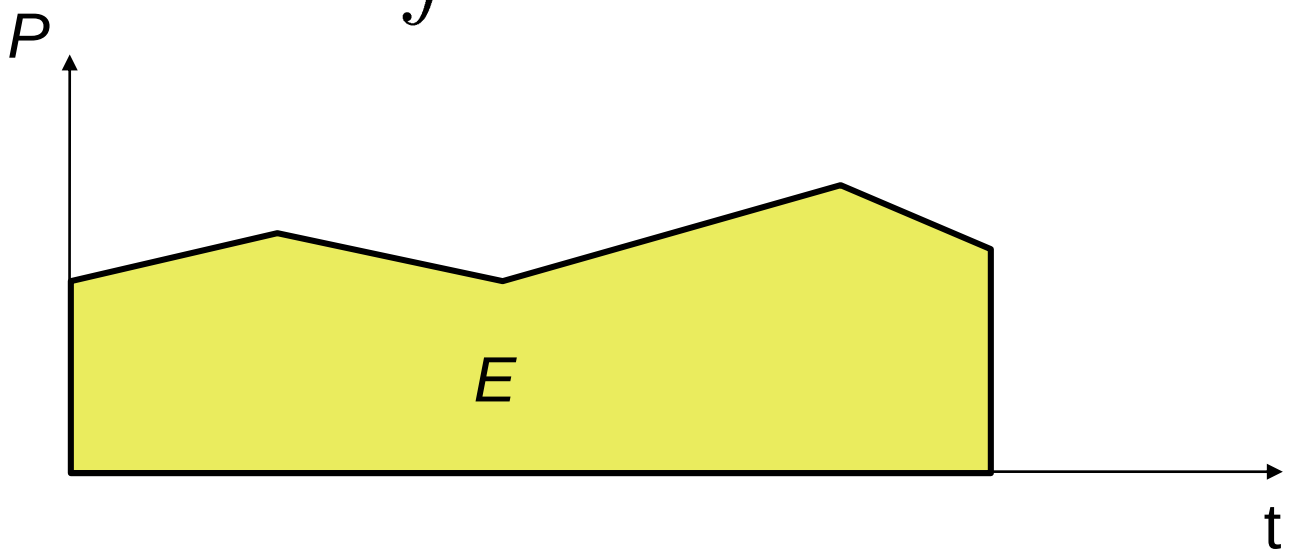
$$E = \int P(t)dt$$



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

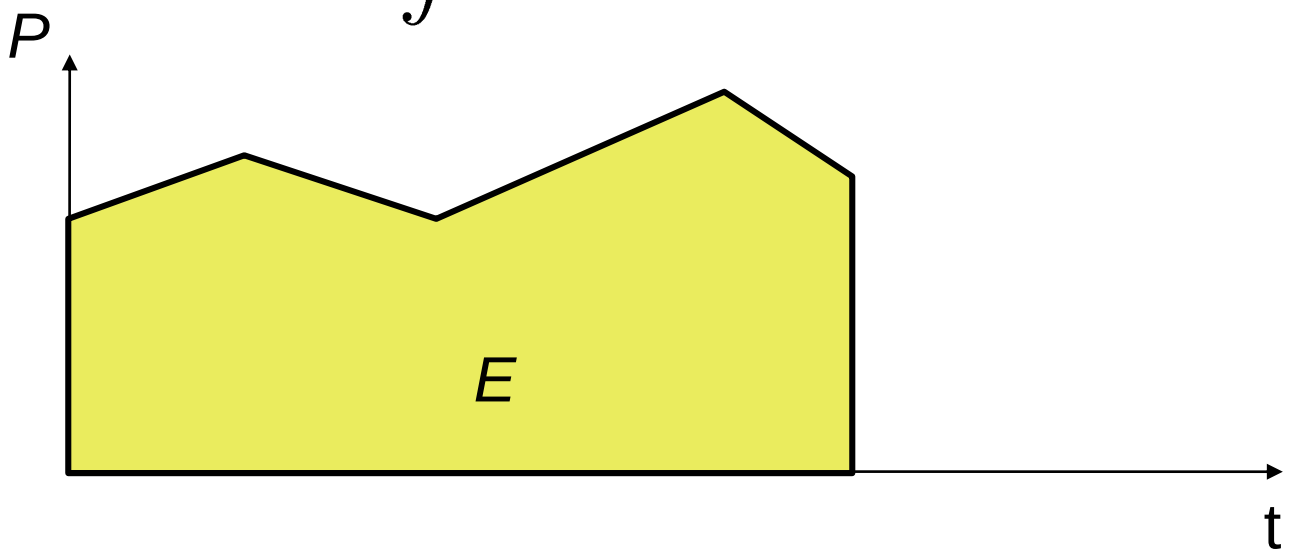
$$E = \int P(t)dt$$



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

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

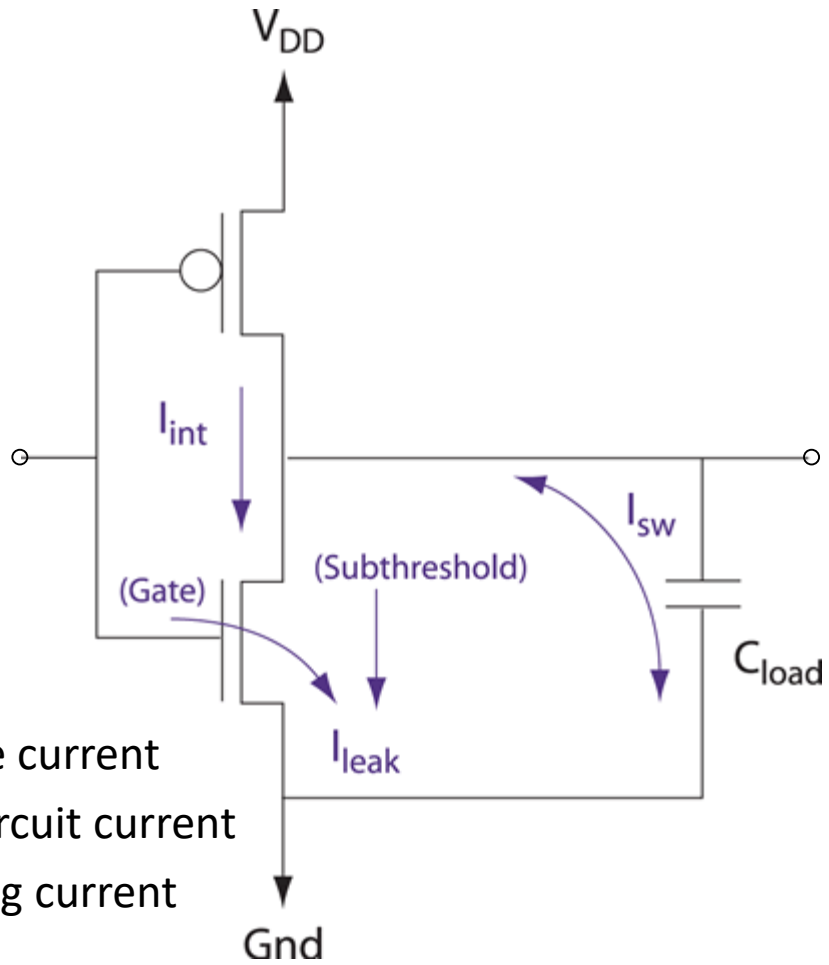


In some cases, faster execution also means less energy, but the opposite may be true if power has to be increased to allow for a faster execution.

Low Power vs. Low Energy

- Minimizing the ***power consumption*** *(voltage * current)* is important for
  - the design of the power supply and voltage regulators
  - the dimensioning of interconnect between power supply and components
  - cooling (short term cooling)
    - high cost
    - 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 (energy harvesting, solar panels, maintenance/batteries)
  - long lifetimes, low temperatures

Power Consumption of a CMOS Gate

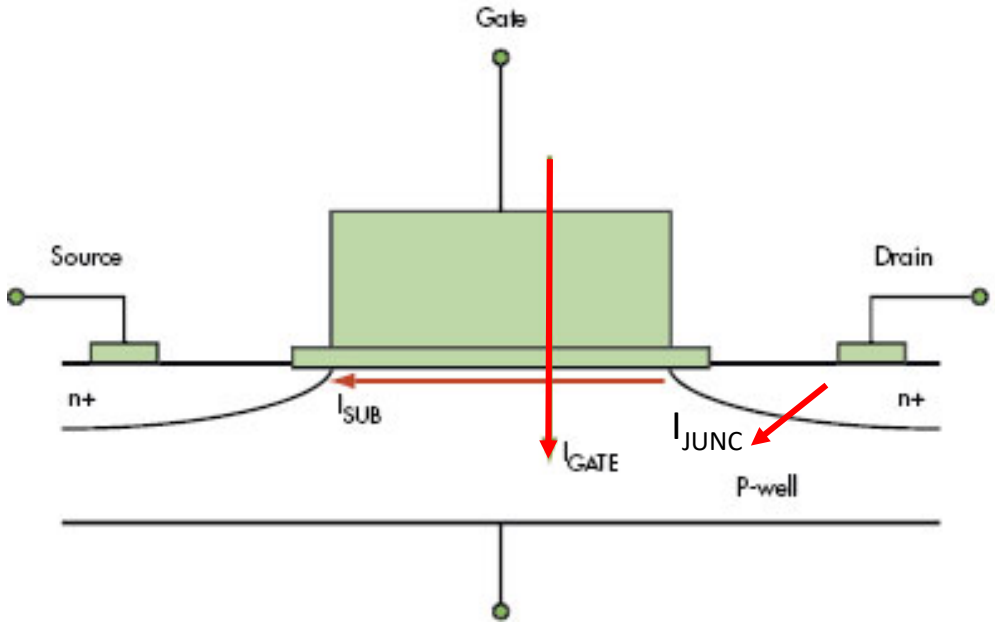


I_{leak} : leakage current

I_{int} : short circuit current

I_{sw} : switching current

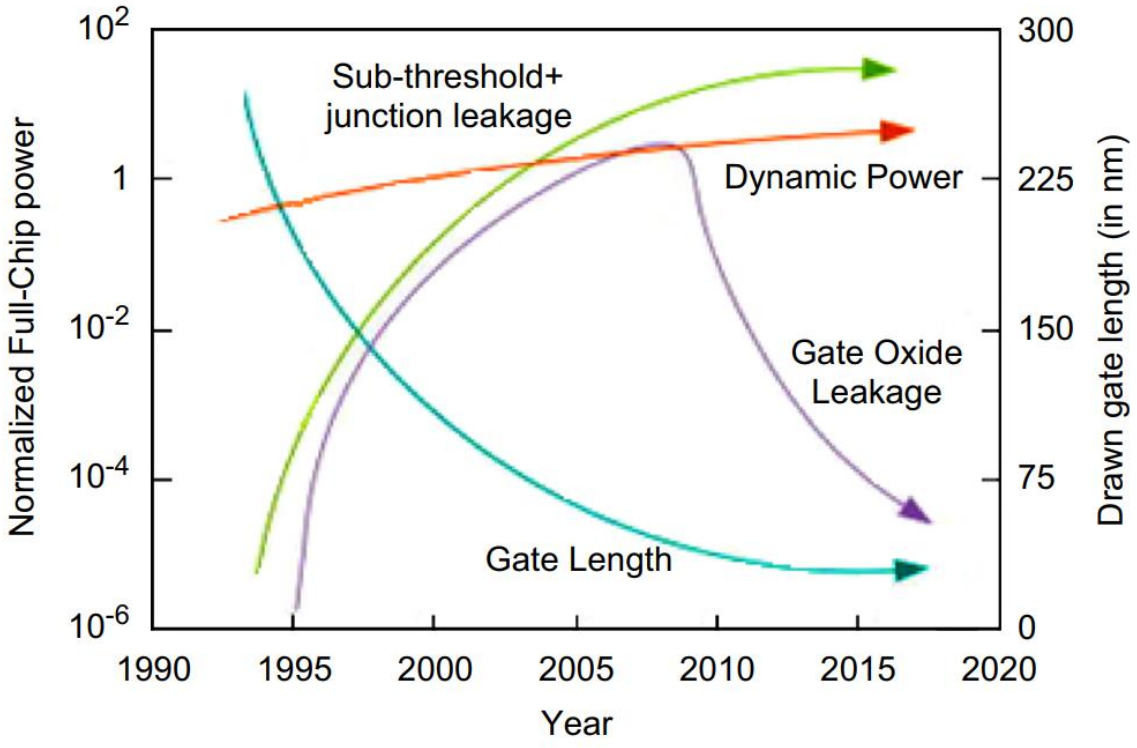
subthreshold (I_{SUB}), junction (I_{JUNC}) and gate-oxide (I_{GATE}) leakage



Power Consumption of a CMOS Processors

Main sources:

- Dynamic power consumption
  - charging and discharging capacitors
  - Short circuit power consumption: short circuit path between supply rails during switching
- Leakage and static power
  - gate-oxide/subthreshold/junction leakage
  - becomes one of the major factors due to shrinking feature sizes in semiconductor technology

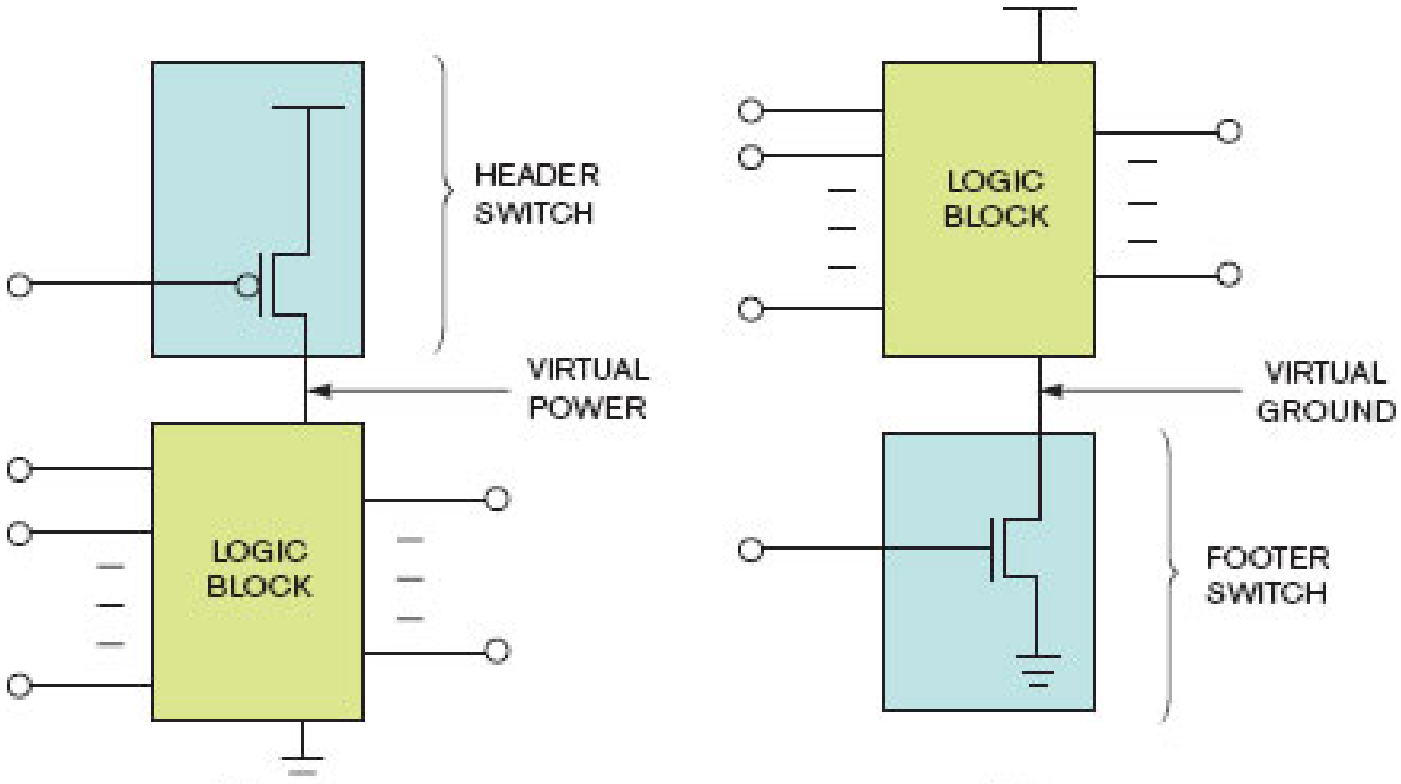


[J. Xue, T. Li, Y. Deng, Z. Yu, Full-chip leakage analysis for 65 nm CMOS technology and beyond, Integration VLSI J. 43 (4) (2010) 353–364]

Reducing Static Power - 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



Dynamic Voltage Scaling (DVS)

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:

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

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} .

Dynamic Voltage Scaling (DVS)

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

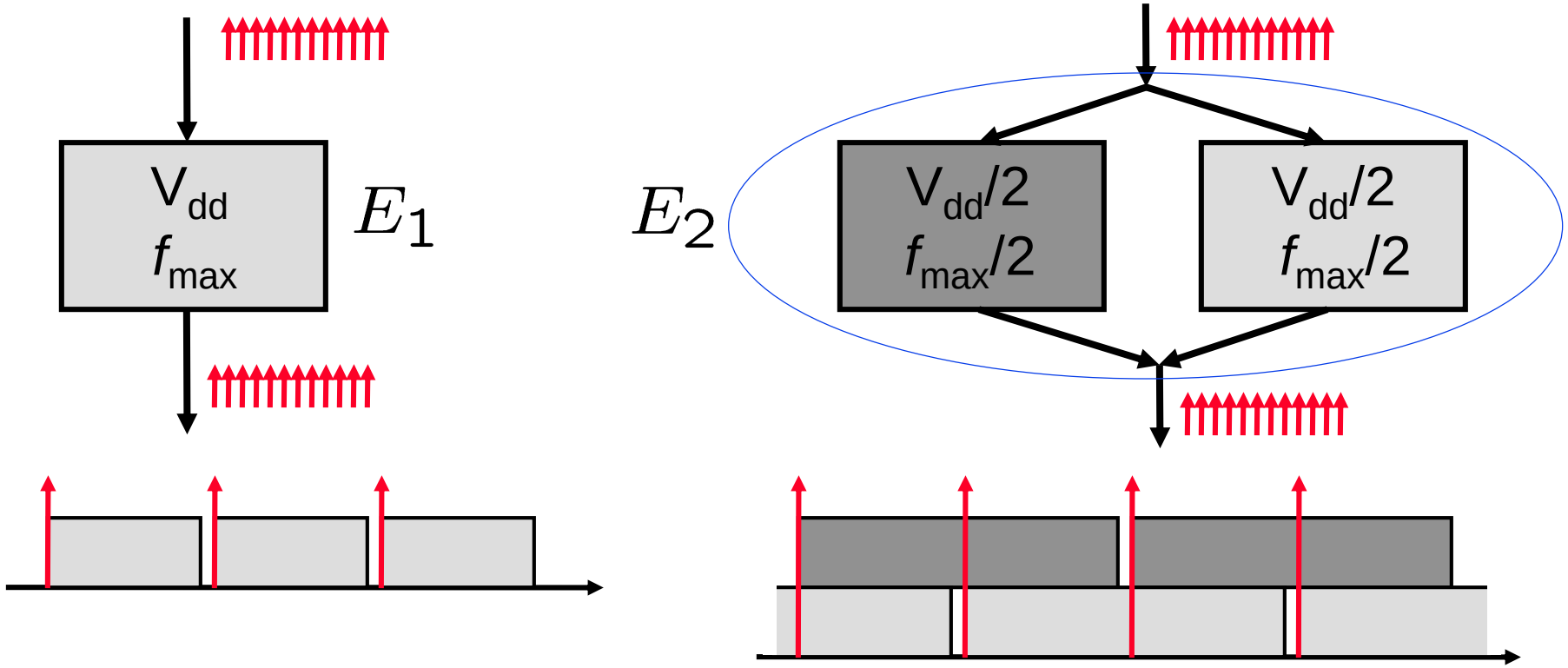
$$E \sim \alpha C_L V_{dd}^2 f t = \alpha C_L V_{dd}^2 \ (\#\text{cycles})$$

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*

Techniques to Reduce Dynamic Power

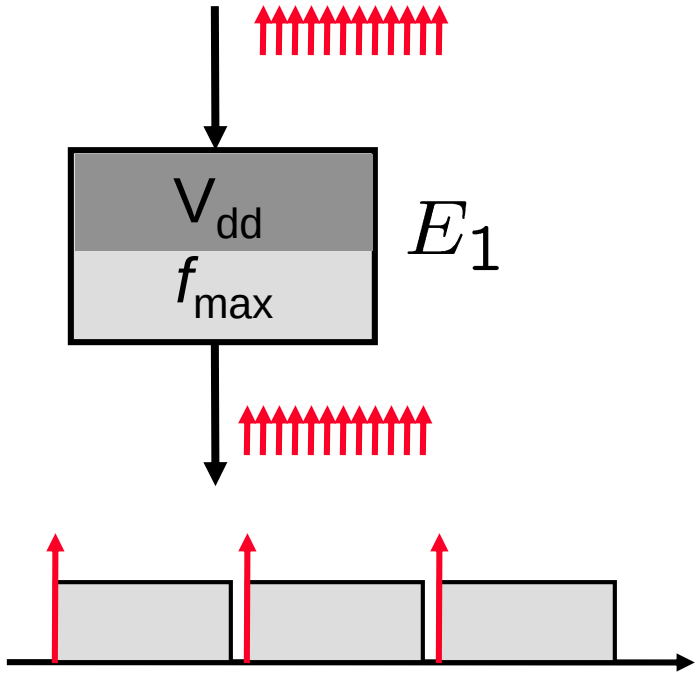
Parallelism

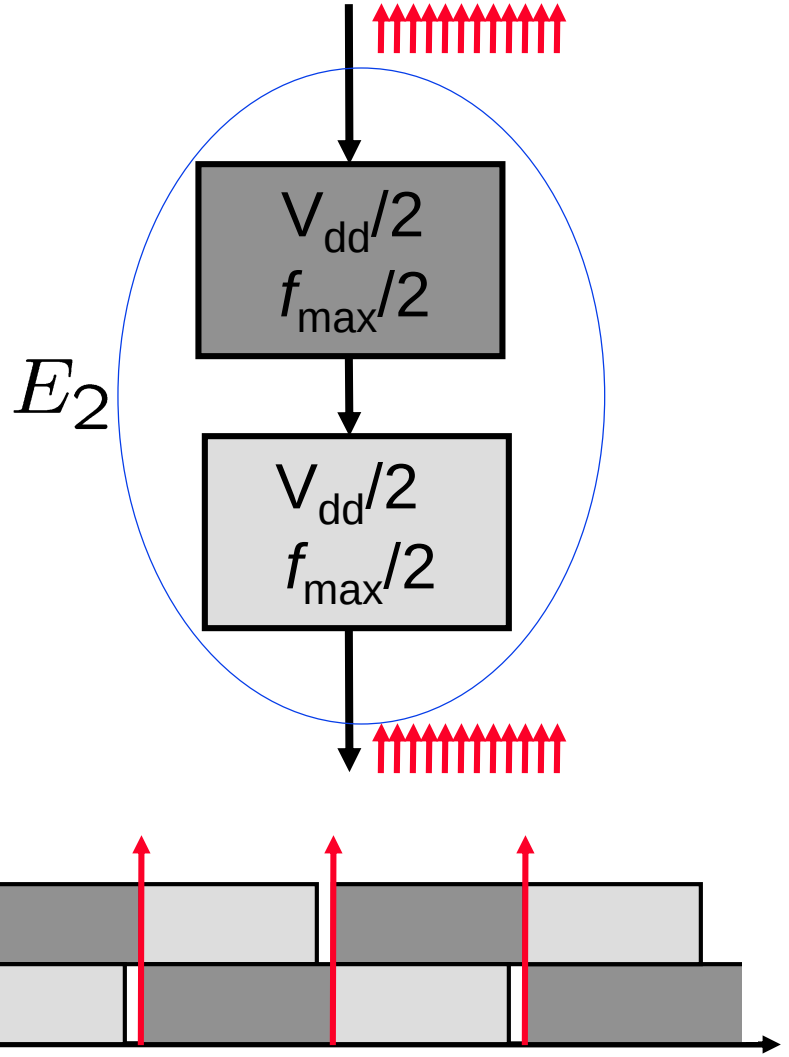


$$E \sim V_{dd}^2 \,(\#\text{cycles})$$

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

Pipelining



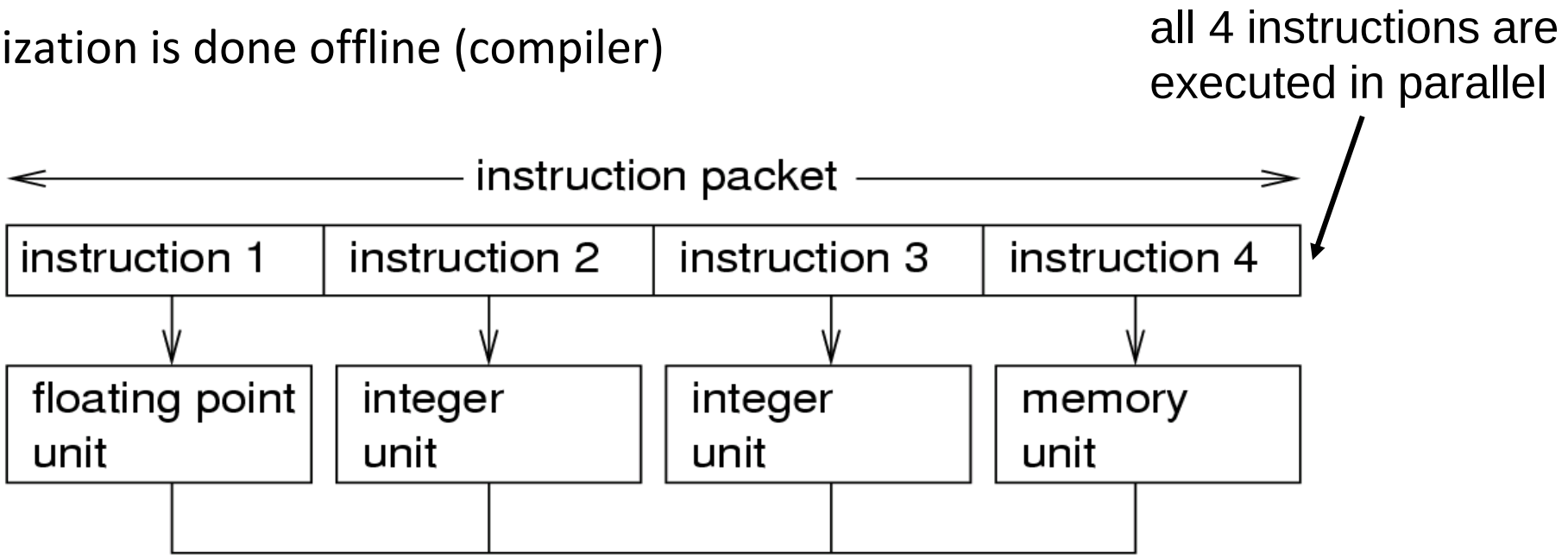


$$E \sim V_{dd}^2 \ (\#\text{cycles})$$

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

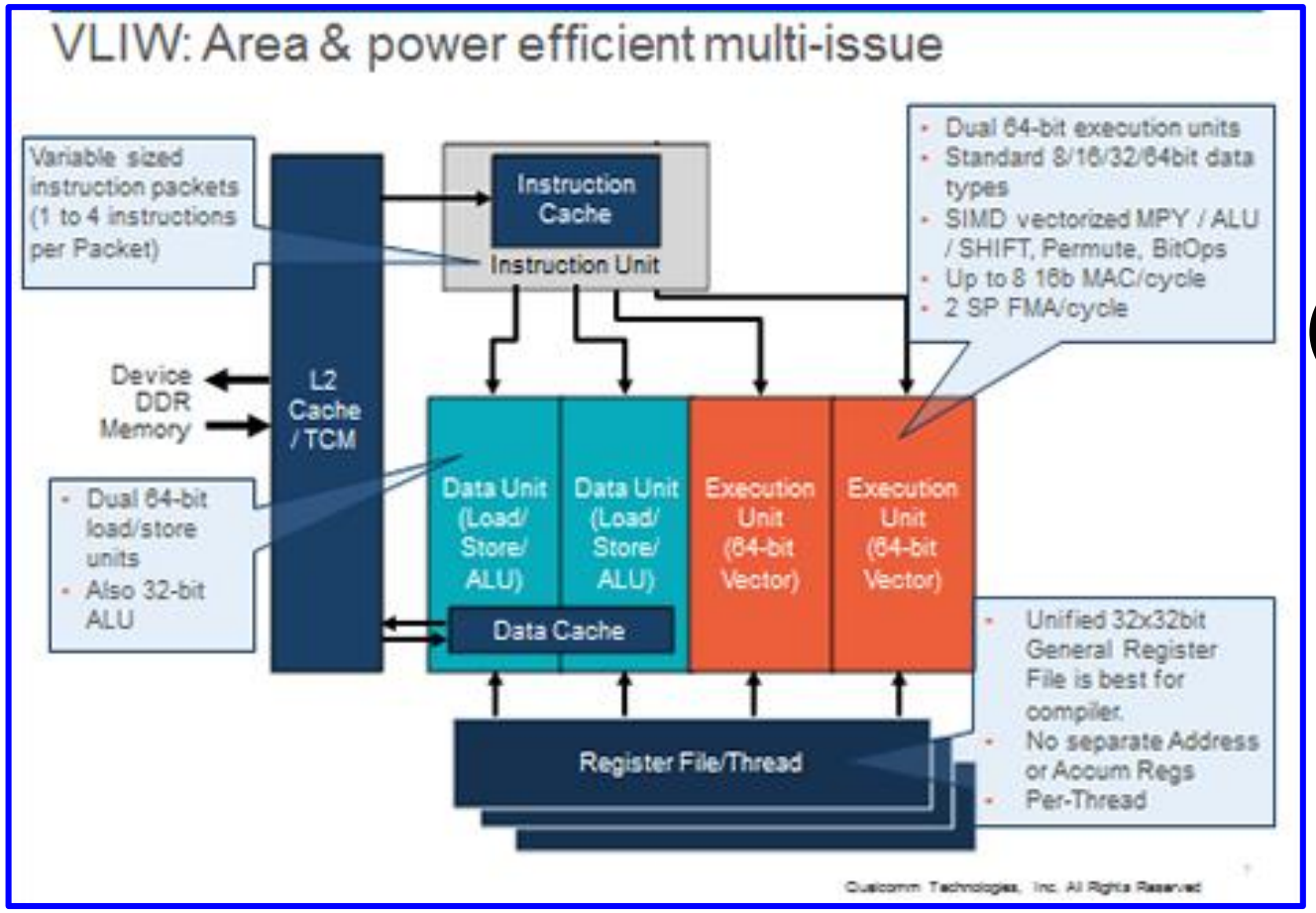
VLIW (Very Long Instruction Word) Architectures

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

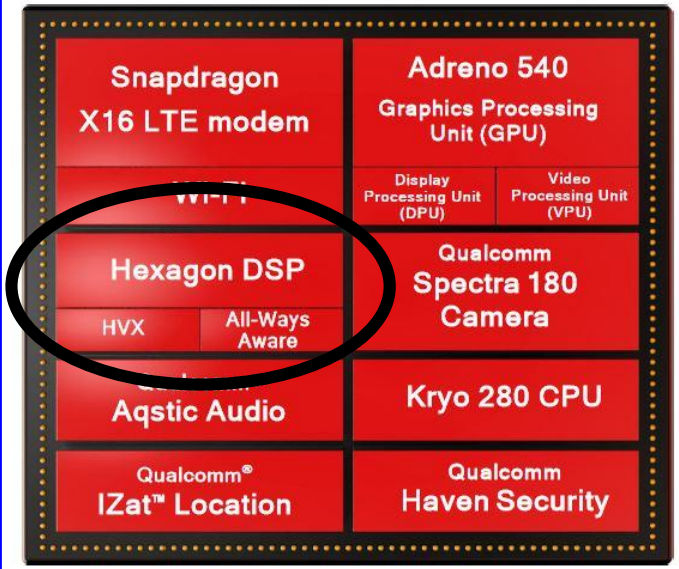


Example: Qualcomm Hexagon

Hexagon DSP



Snapdragon 835 (Galaxy S8)



Dynamic Voltage and Frequency Scaling - Optimization

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

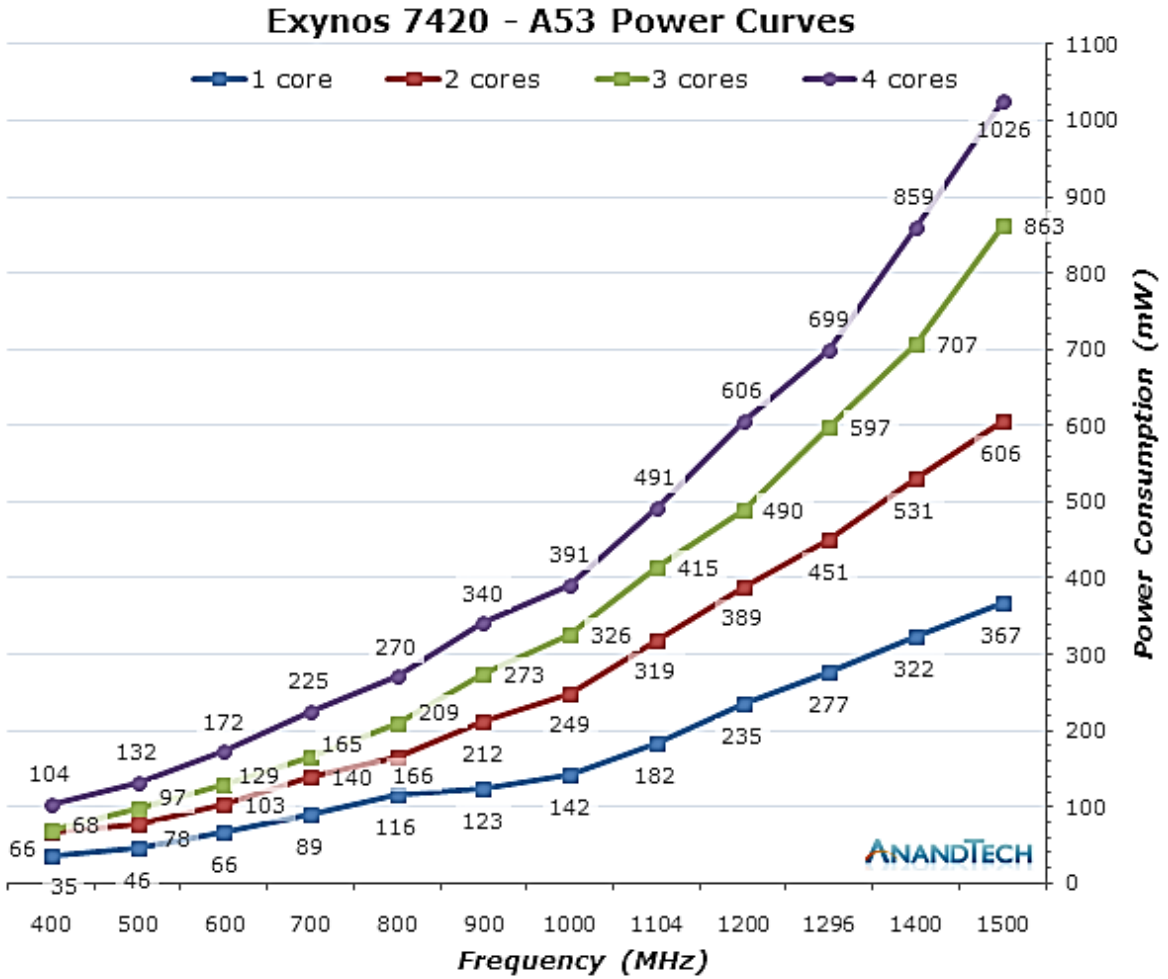
gate delay

Saving energy for a given task:

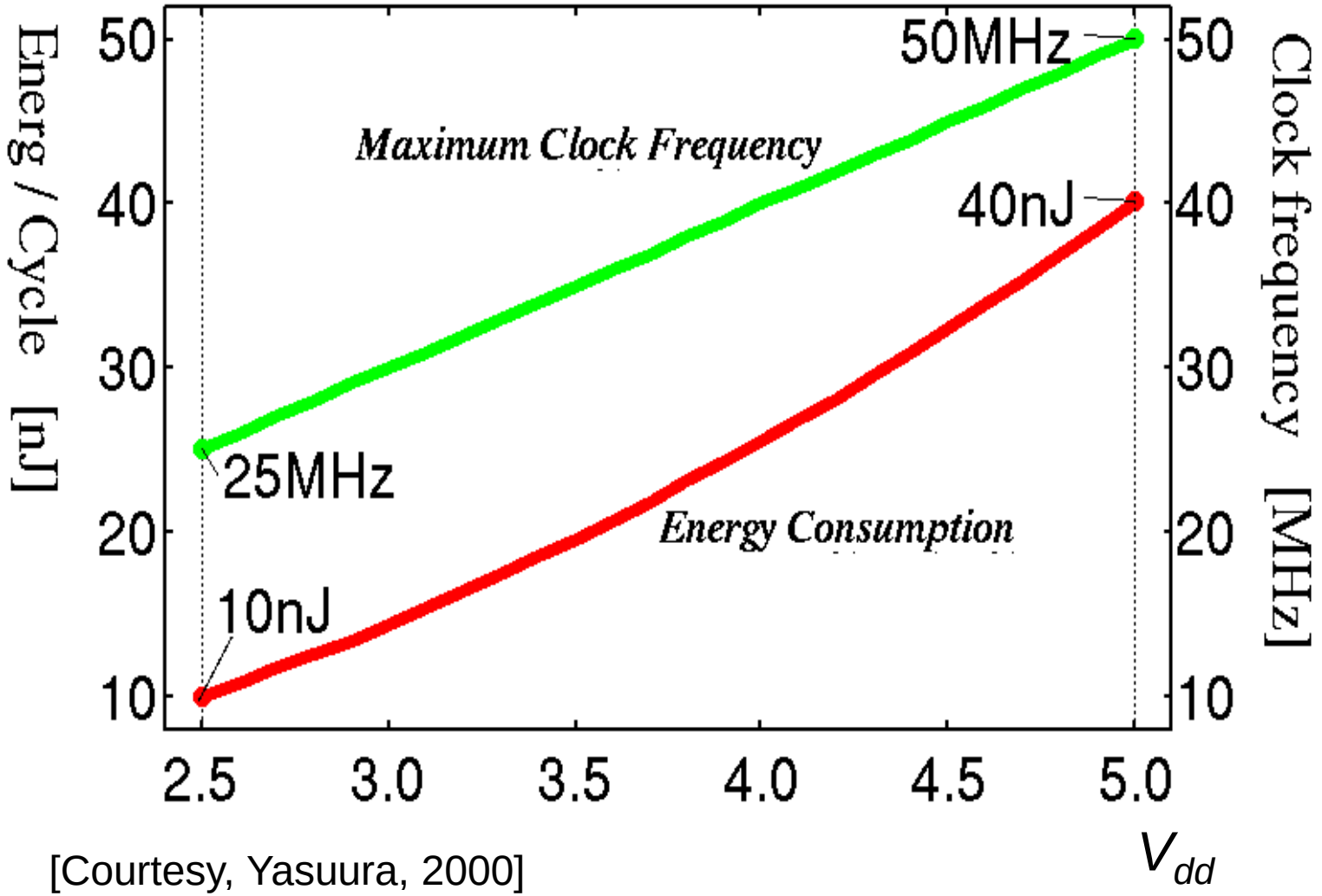
- reduce the supply voltage V_{dd}
- reduce switching activity α
- reduce the load capacitance C_L
- reduce the number of cycles *#cycles*

Example DVFS: Samsung Exynos (ARM processor)

ARM processor core A53 on the Samsung Exynos 7420 (used in mobile phones, e.g. Galaxy S6)



Example: Dynamic Voltage and Frequency Scaling

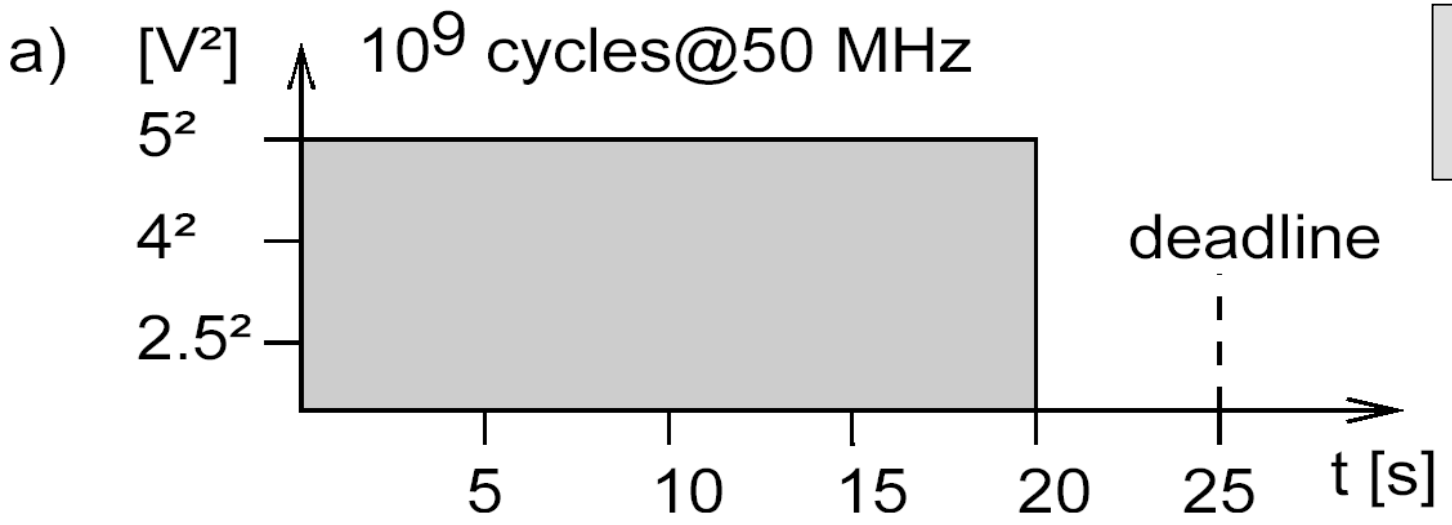


[Courtesy, Yasuura, 2000]

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.

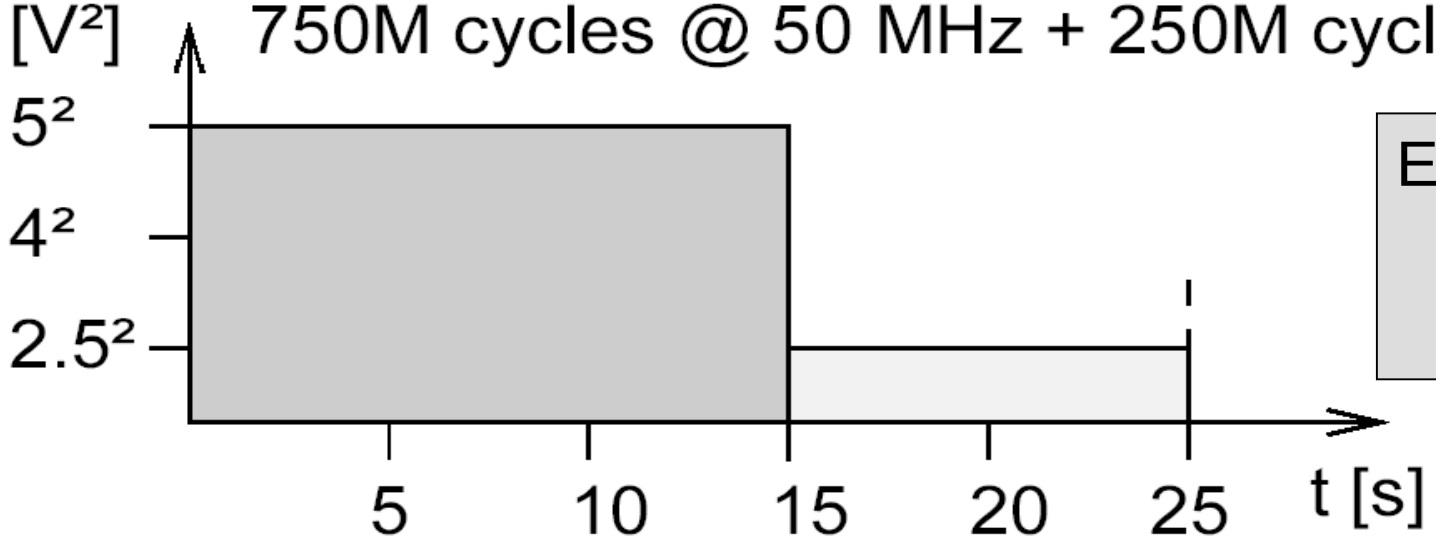


$E_a = 10^9 \times 40 \times 10^{-9}$
$= 40$ [J]

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



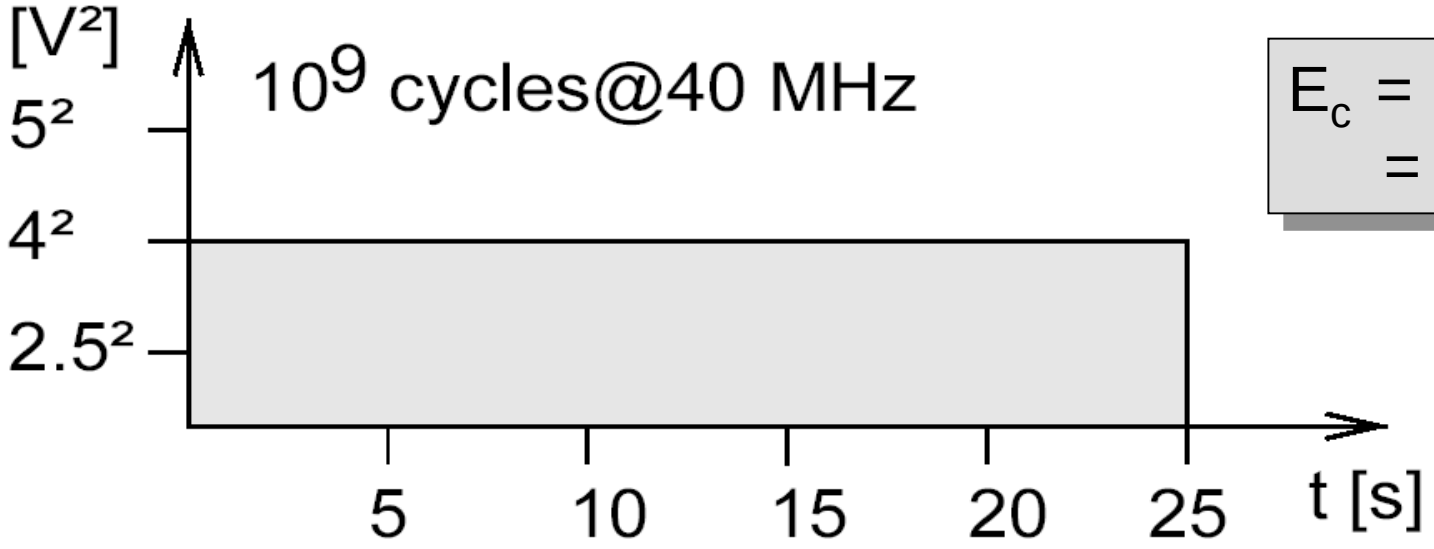
$E_b = 750\ 10^6 \times 40 \times 10^{-9} + 250\ 10^6 \times 10 \times 10^{-9}$

$= 32.5$ [J]

Example: DVFS – Use One Voltage

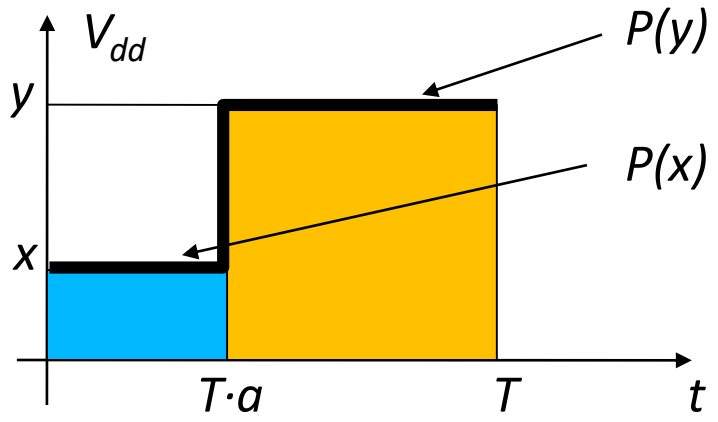
V_{dd} [V]	5.0	4.0	2.5
Energy per cycle [nJ]	40	25	10
f_{max} [MHz]	50	40	25
cycle time [ns]	20	25	40

c)



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

DVFS: Optimal Strategy



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

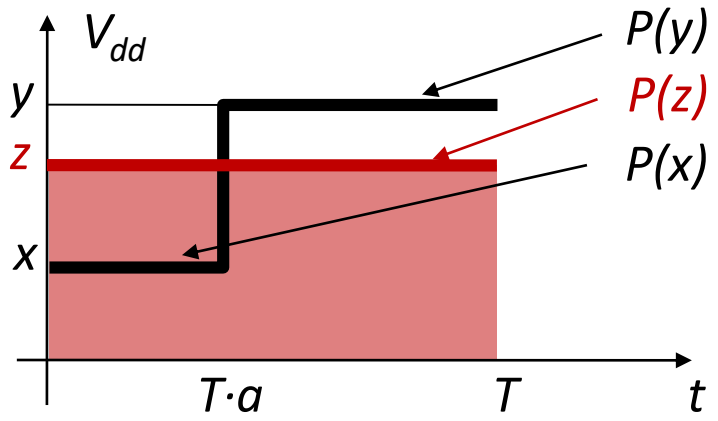
gate delay: $\tau \sim \frac{1}{V_{dd}}$

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

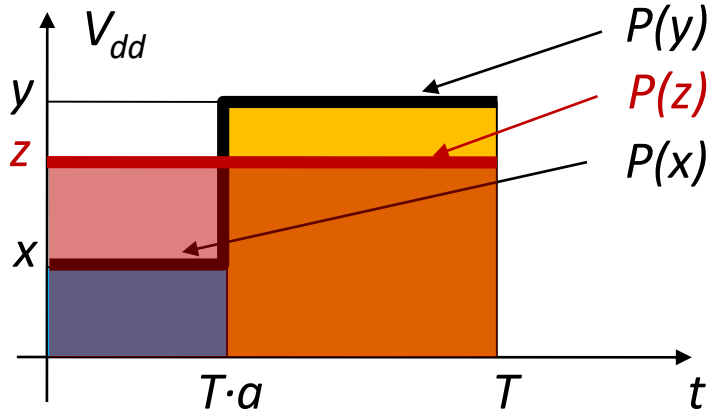
execution rate: $f(t) \sim V_{dd}(t)$

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

- ***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: $\tau \sim \frac{1}{V_{dd}}$

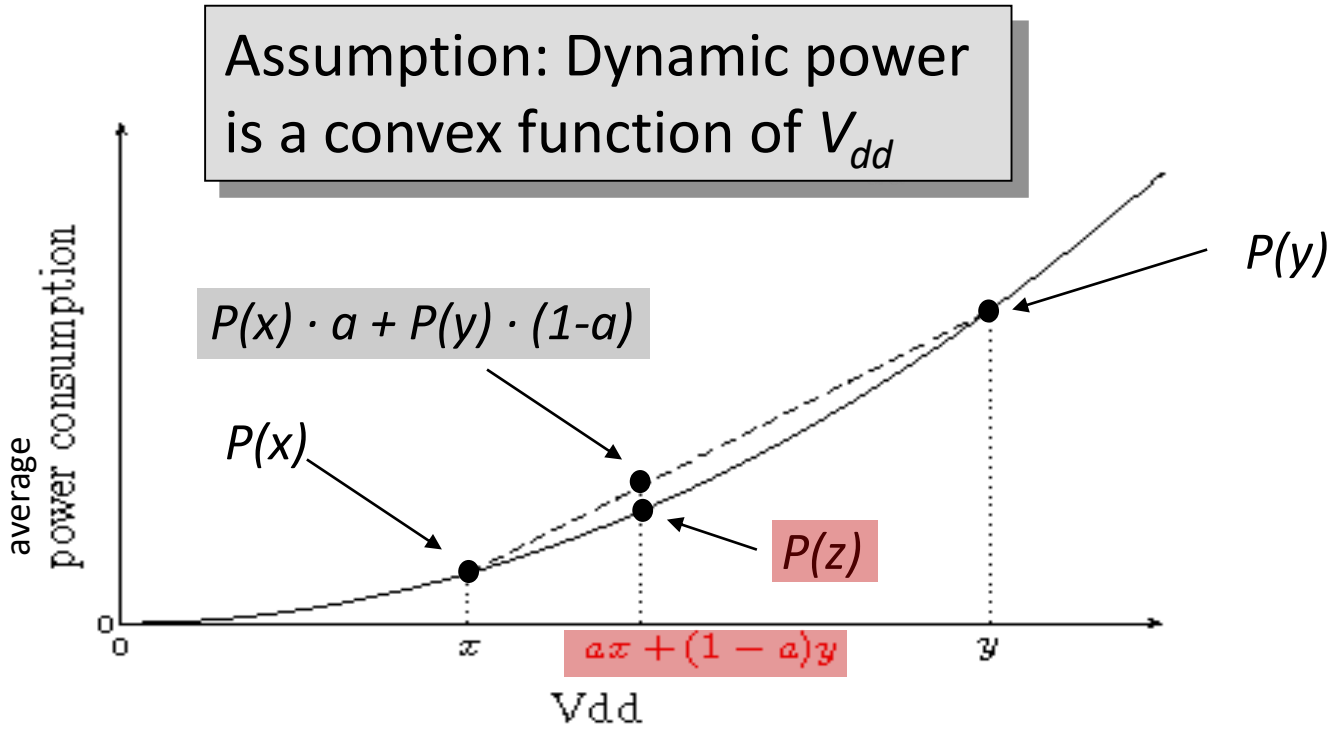
execution rate: $f(t) \sim V_{dd}(t)$

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

- ***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

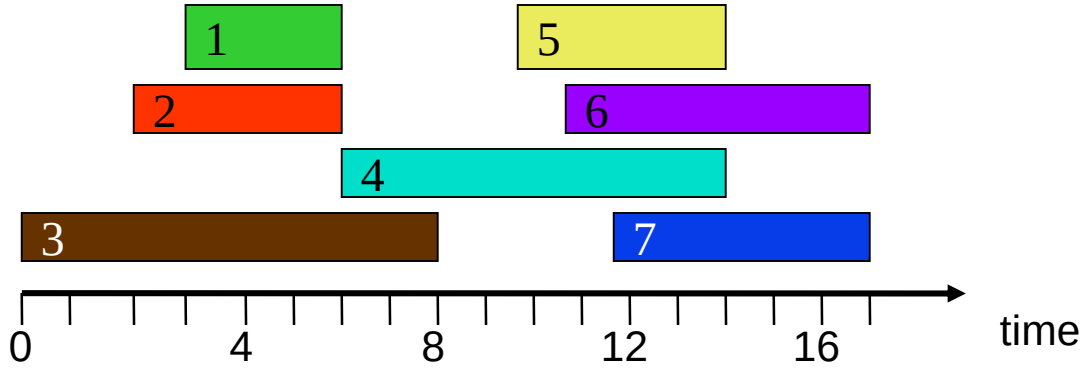
DVFS: Real-Time Offline Scheduling on One Processor

- 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 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 Optimal DVFS 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

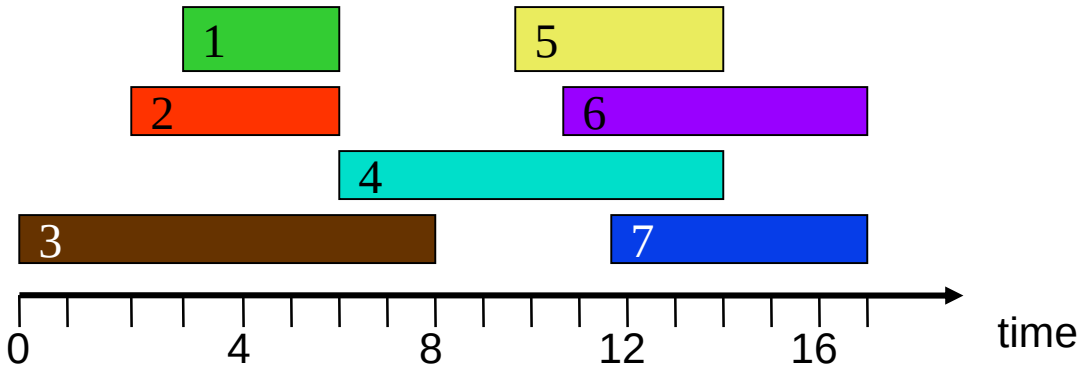
- Define ***intensity*** G([*z, z'*]) in some time interval [*z, z'*]:
  - average accumulated execution time of all tasks that have arrival and deadline in [*z, z'*] relative to the length of the interval *z'-z*

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

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

YDS 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

3,6,5

2,6,3

0,8,2

6,14,6

10,14,6

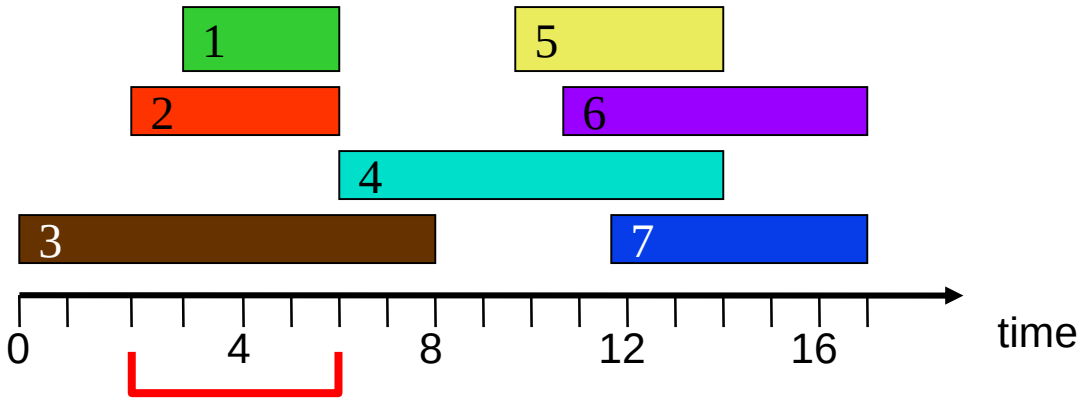
11,17,2

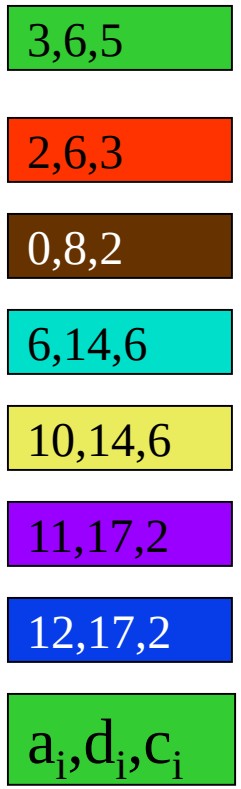
12,17,2

a_i,d_i,c_i

YDS Optimal DVFS Algorithm for Offline Scheduling

Step 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

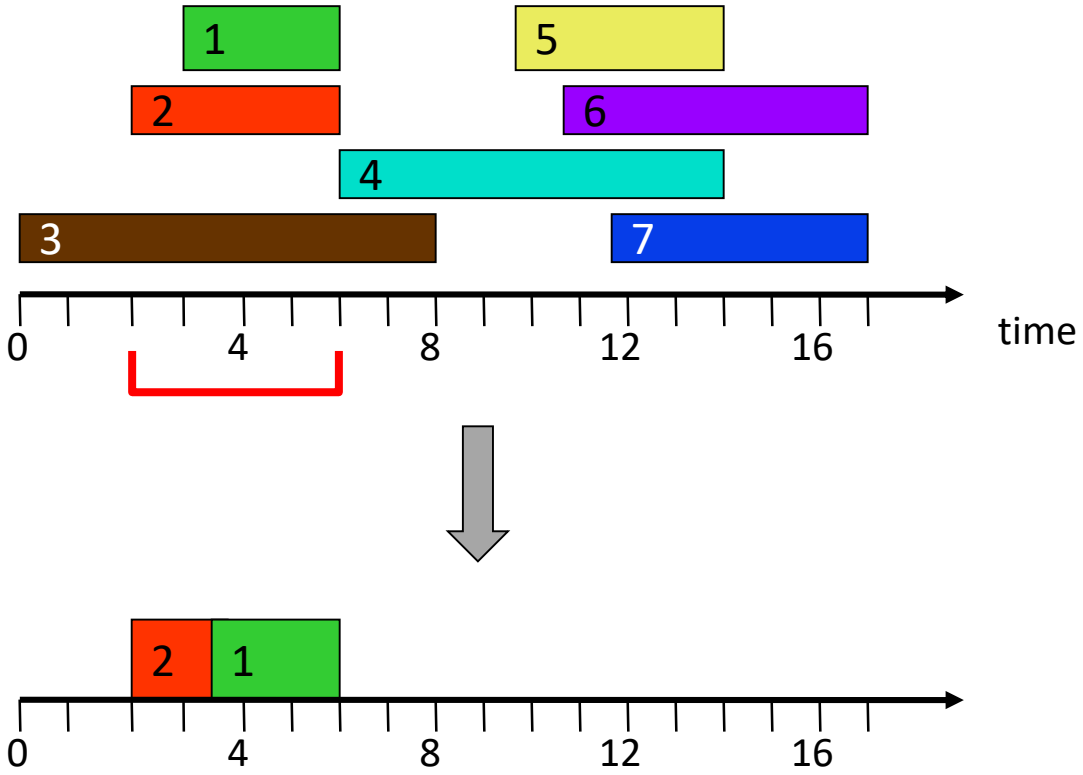
G([3,6]) =5/3, G([3,14]) = (5+6+6)/(14-3) = 17/11, G([3,17])=(5+6+6+2+2)/14=21/14

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

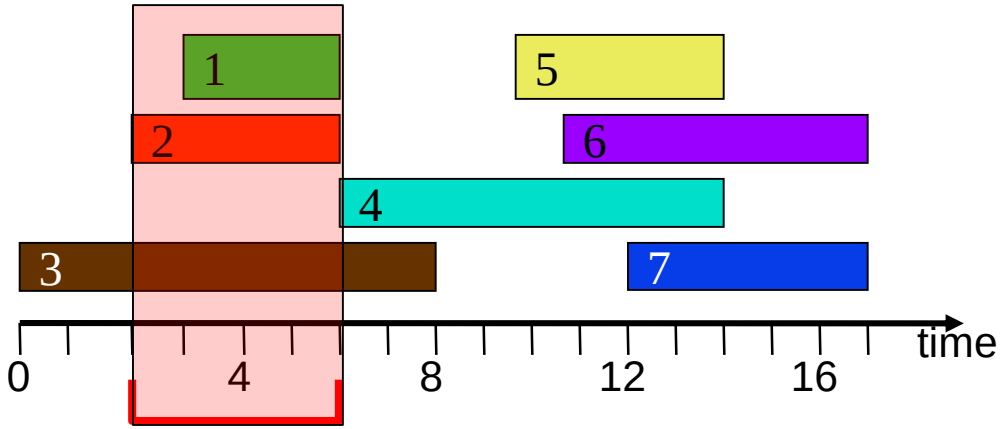
11,17,2

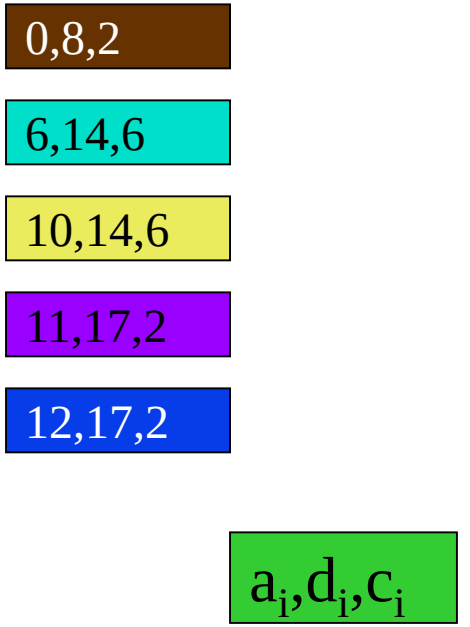
12,17,2

a_i,d_i,c_i

YDS Optimal DVFS Algorithm for Offline Scheduling

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

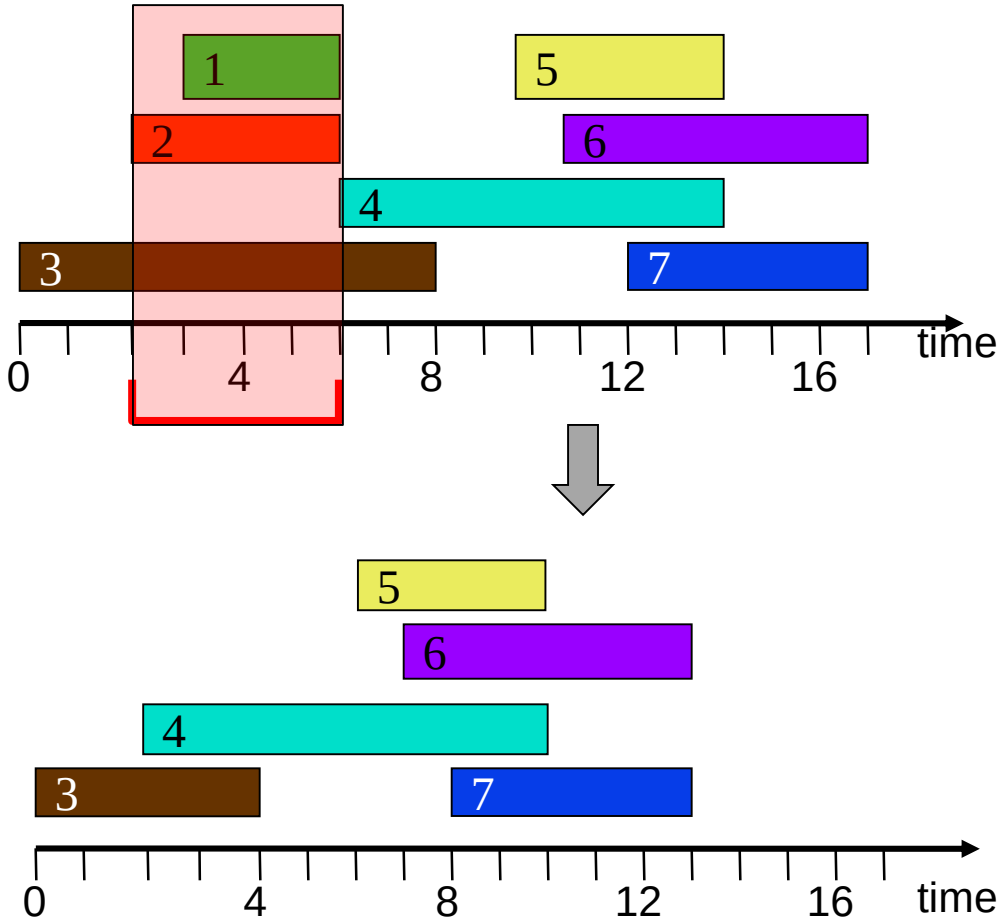


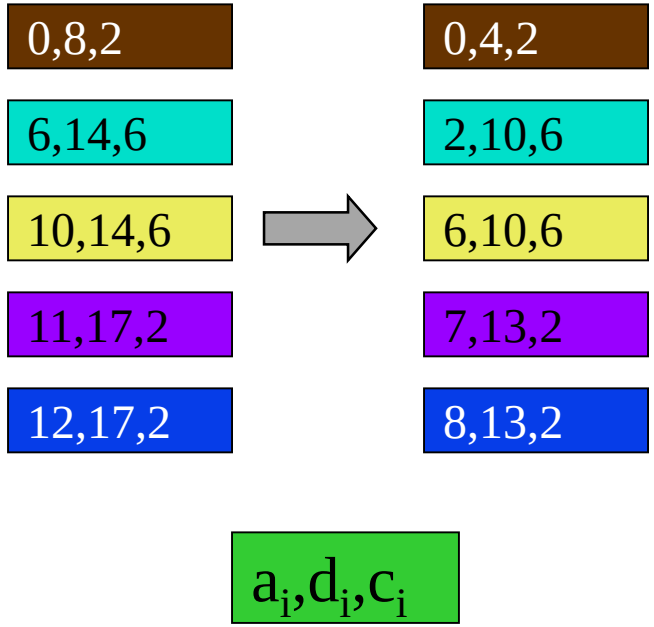


a_i,d_i,c_i

YDS Optimal DVFS Algorithm for Offline Scheduling

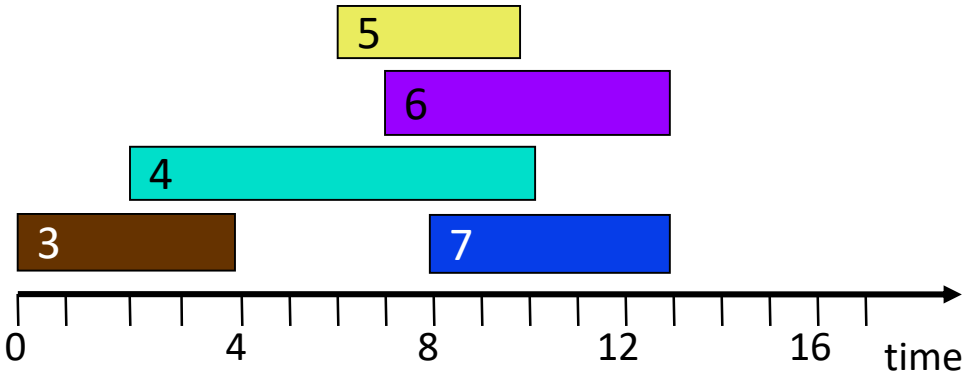
Step 2: Adjust the arrival times and deadlines by excluding the possibility to execute at the previous critical intervals.





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

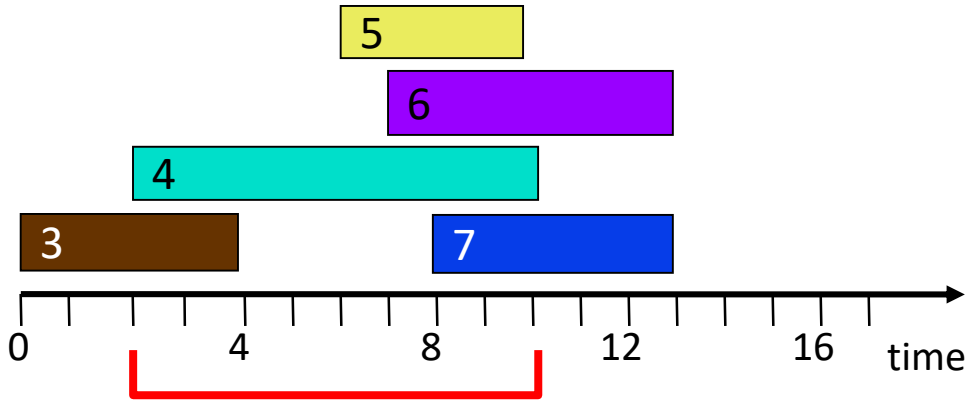
7,13,2

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

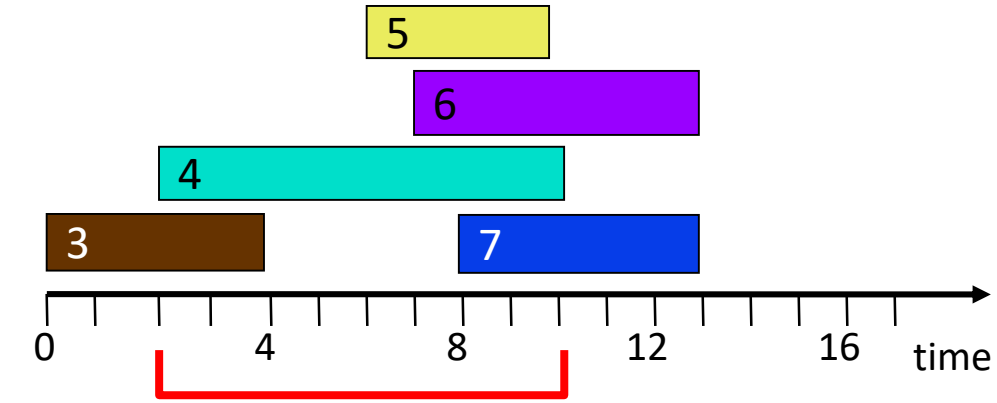
7,13,2

8,13,2

a_i, d_i, c_i

YDS Optimal DVFS Algorithm for Offline Scheduling

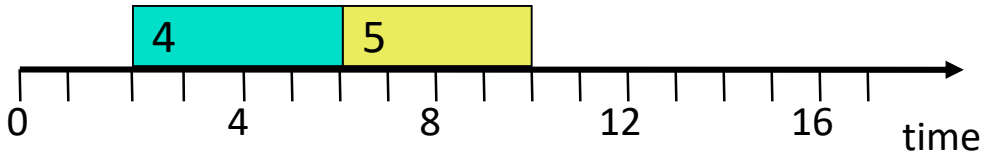
Step 3: Run the algorithm for the revised input again



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

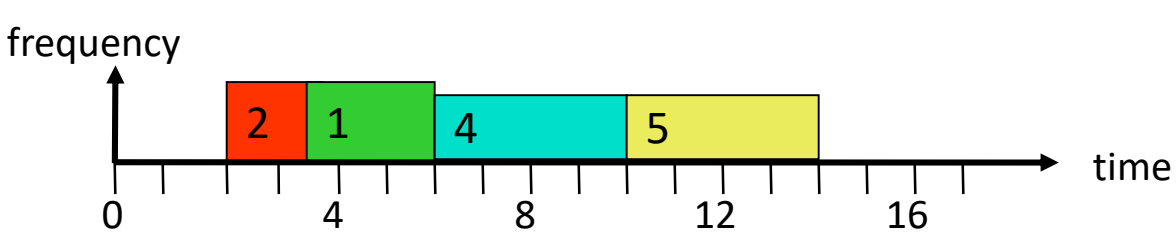
8,13,2

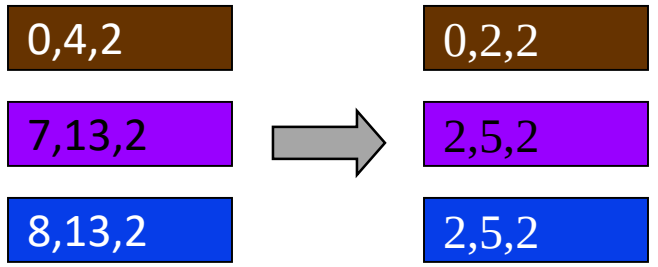
a_i, d_i, c_i

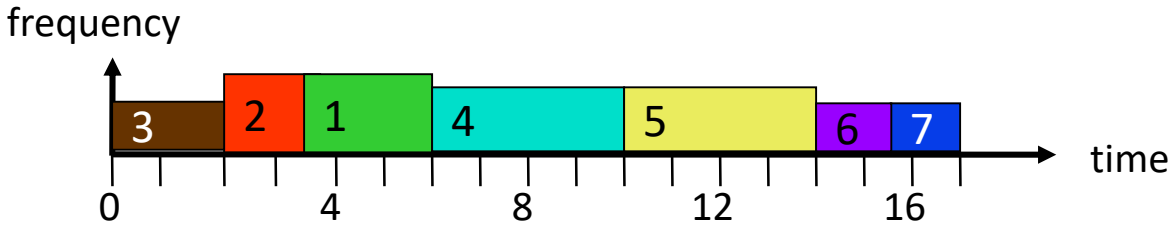
YDS Optimal DVFS Algorithm for Offline Scheduling

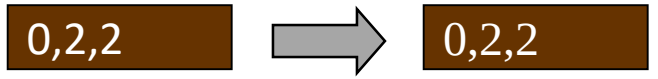
Step 3: Run the algorithm for the revised input again

Step 4: Put pieces together



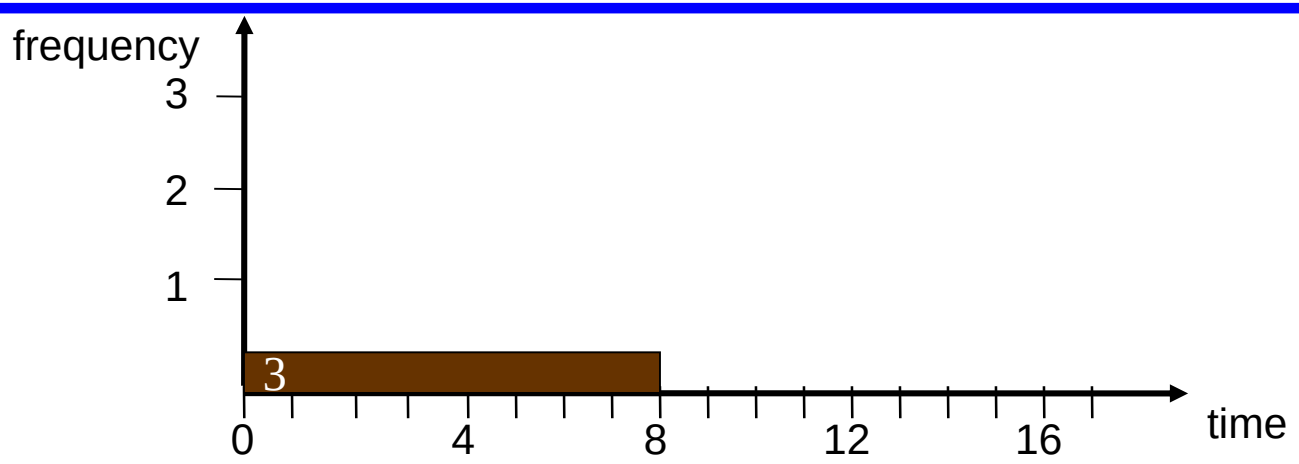






	v_1	v_2	v_3	v_4	v_5	v_6	v_7
frequency	2	2	1	1.5	1.5	4/3	4/3

YDS Optimal DVFS Algorithm for Online Scheduling



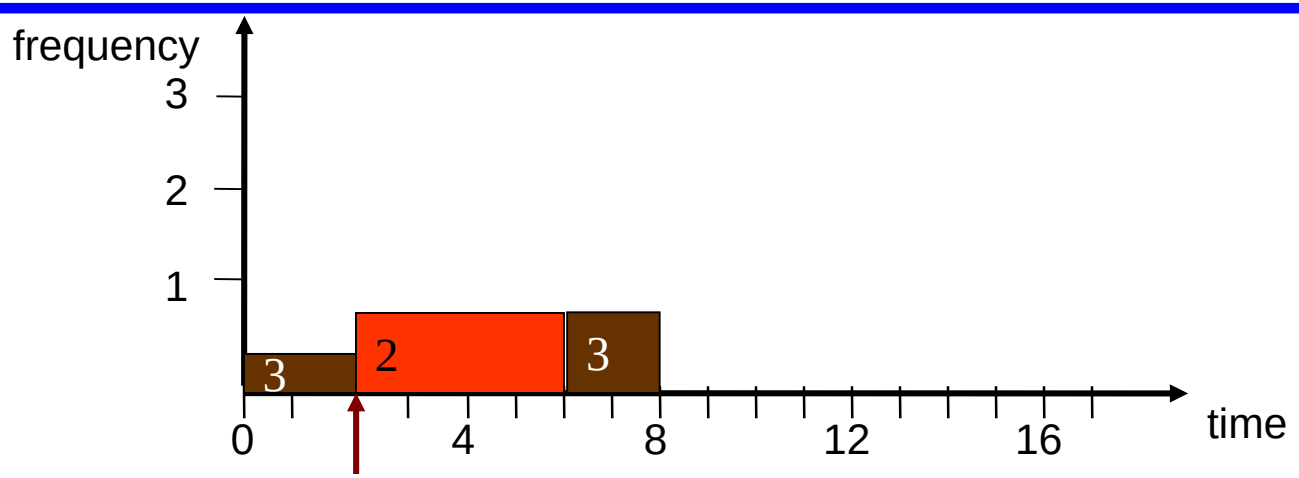
0,8,2

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:

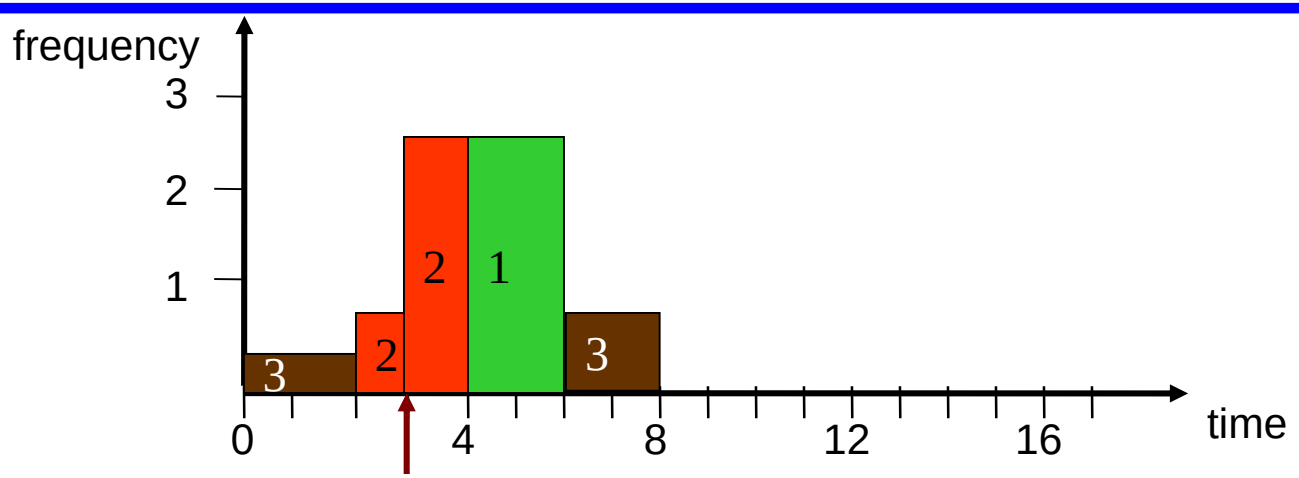
Time 0: task v_3 is executed at 2/8

Time 2: task v_2 arrives

- G([2,6]) = ¾, G([2,8]) = 4.5/6=3/4 => execute v_8 , 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

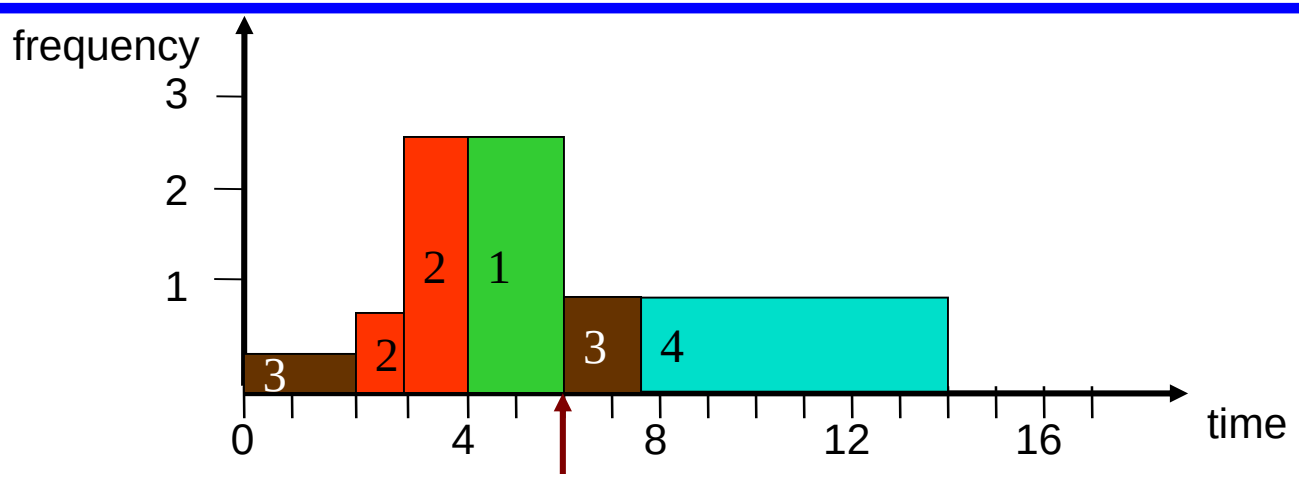
- G([2,6]) = ¾, G([2,8]) = 4.5/6=3/4 => execute v_8 , v_2 at ¾

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

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_8 , v_2 at ¾

Time 3: task v_1 arrives

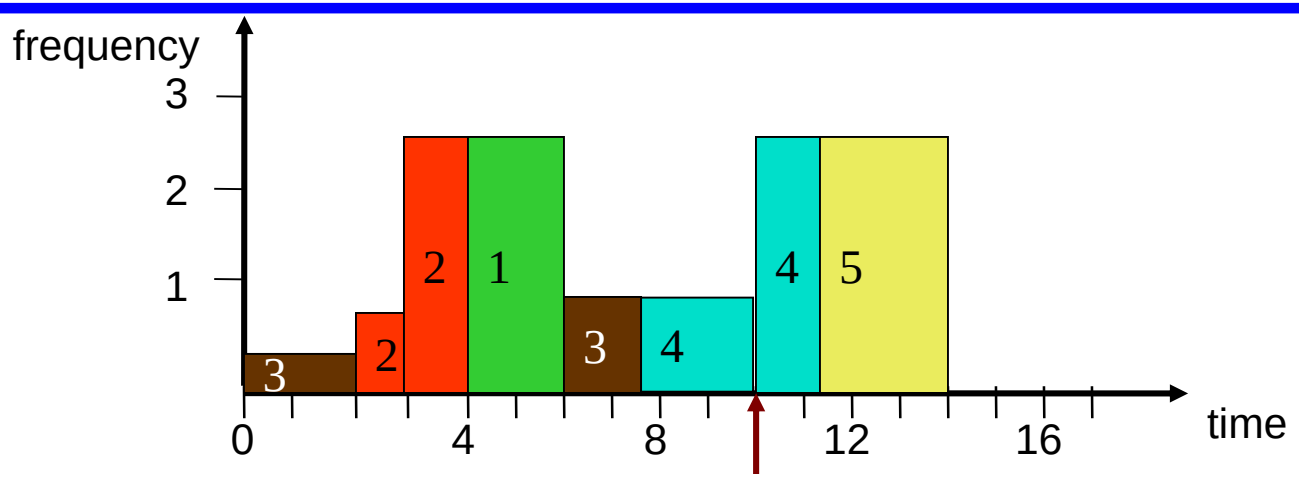
- G([3,6]) = (5+3-3/4)/3=29/12, G([3,8]) < G([3,6]) => execute v_2 and v_1 at 29/12

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

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

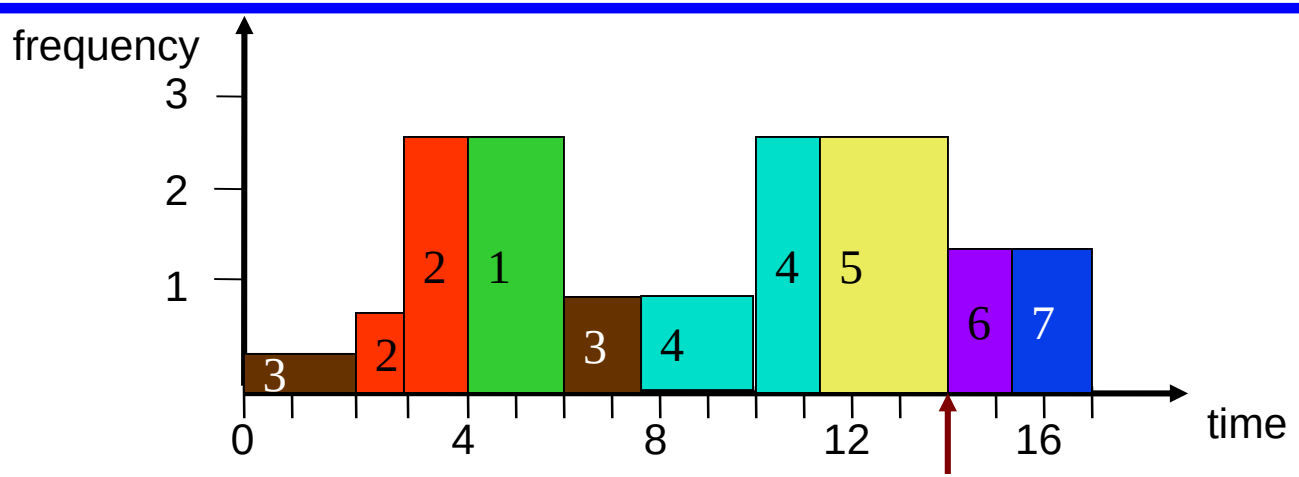
- G([6,8]) = 1.5/2, G([6,14]) = 7.5/8 => execute v_3 and v_4 at 15/16

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.

Dynamic Power Management

Dynamic Power Management (DPM)

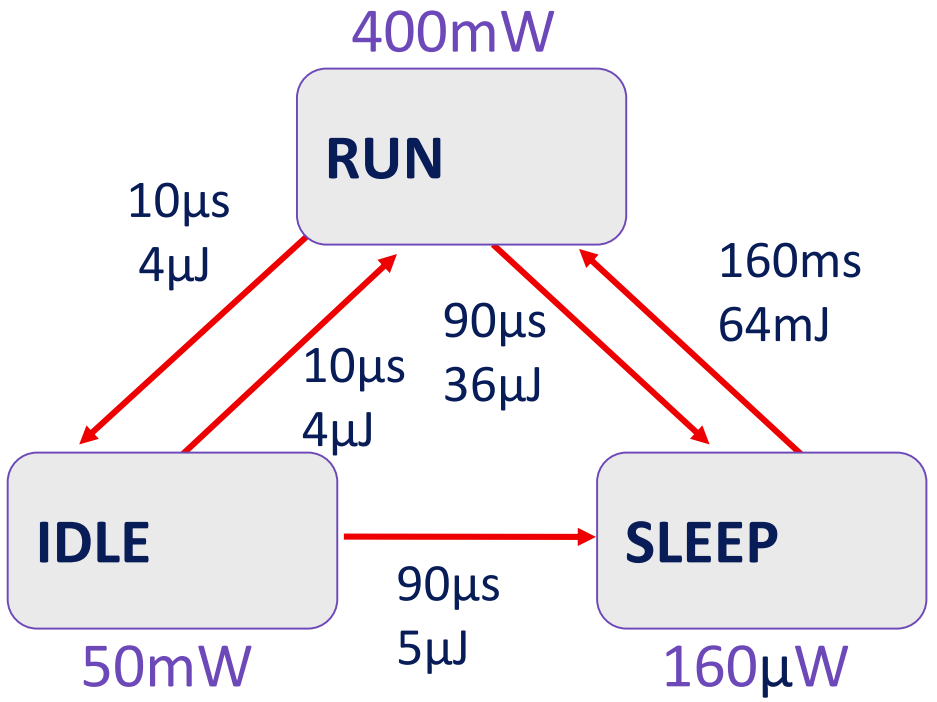
- Dynamic power management tries to assign optimal **power saving states during program execution**
- DPM requires hardware and software support

Example: StrongARM SA1100

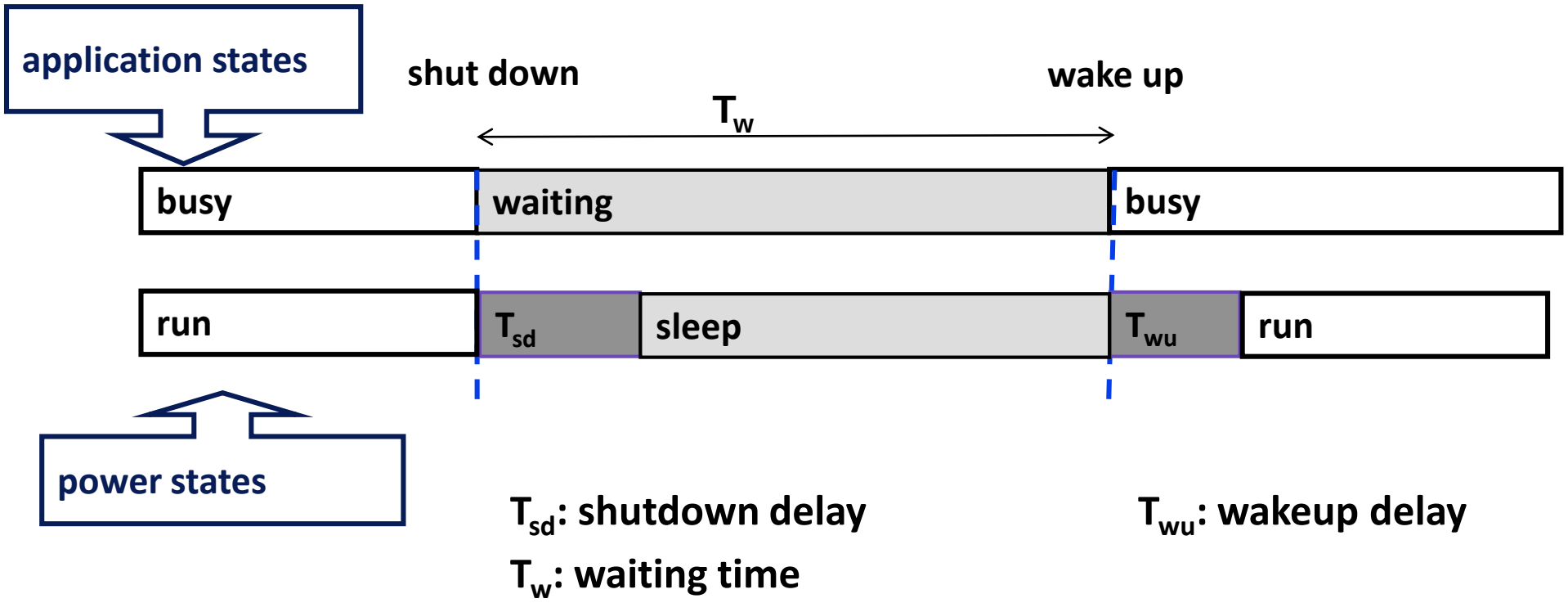
RUN: operational

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

SLEEP: Shutdown of on-chip activity



Dynamic Power Management (DPM)

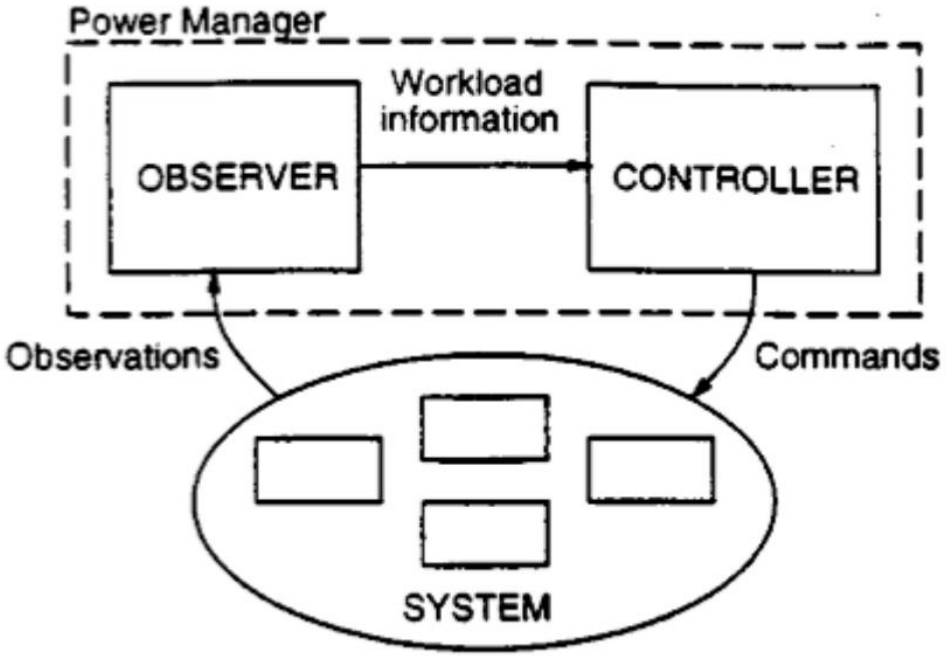


Desired: Shutdown only during long waiting times. This leads to a tradeoff between energy saving and overhead.

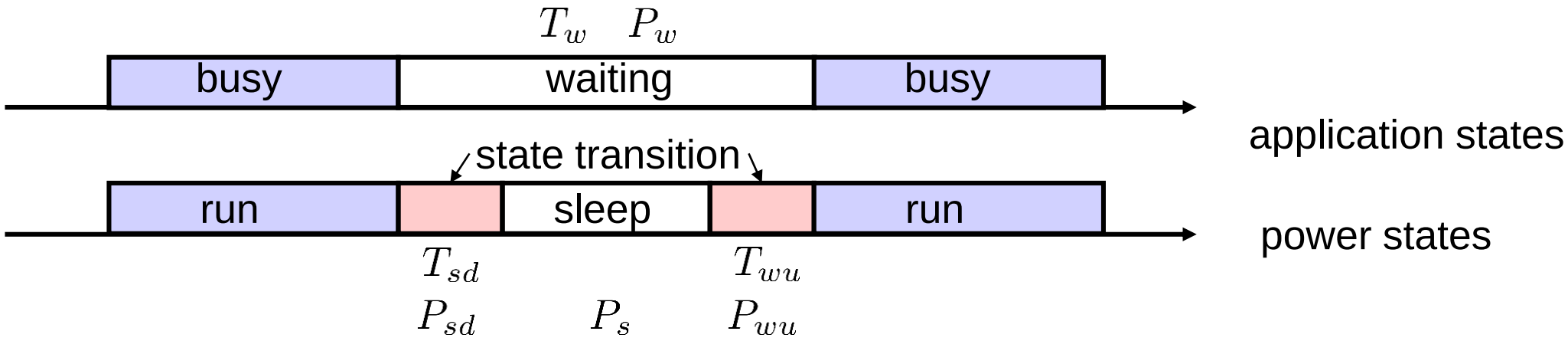
Break-Even Time

Definition: The minimum waiting time required to compensate the cost of entering an inactive (sleep) state.

- Enter an inactive state is beneficial only if the waiting time is longer than the break-even time
- Assumptions for the calculation:
  - No performance penalty is tolerated.
  - An ideal power manager that has the *full* knowledge of the future workload trace. On the previous slide, we supposed that the power manager has *no* knowledge about the future.



Break-Even Time



Scenario 1 (no transition): $E_1 = T_w \cdot P_w$

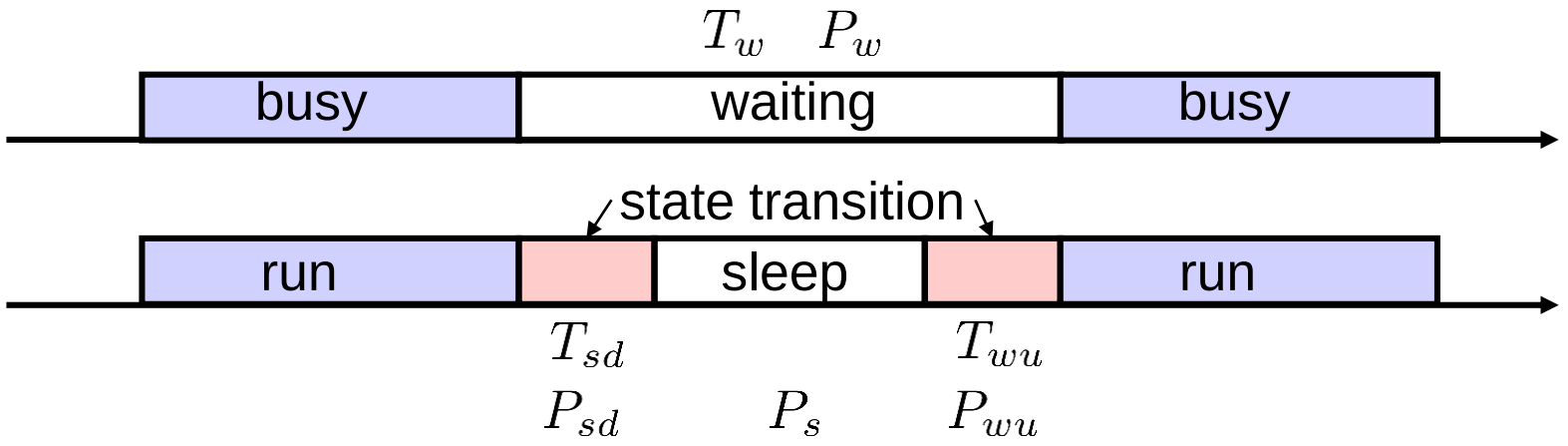
Scenario 2 (state transition): $E_2 = T_{sd} \cdot P_{sd} + T_{wu} \cdot P_{wu} + (T_w - T_{sd} - T_{wu}) \cdot P_s$

Break-even time: Limit for T_w such that $E_2 \leq E_1$

Break-even constraint: $T_w \geq \dfrac{T_{sd} \cdot (P_{sd} - P_s) + T_{wu} \cdot (P_{wu} - P_s)}{P_w - P_s}$ ← break-even time

Time constraint: $T_w \geq T_{sd} + T_{wu}$

Break-Even Time



Scenario 1 (no transition): $E_1 = T_w \cdot P_w$

Scenario 2 (state transition): $E_2 = T_{sd} \cdot P_{sd} + T_{wu} \cdot P_{wu} + (T_w - T_{sd} - T_{wu}) \cdot P_s$

Break-even time: Limit for T_w such that $E_2 \leq E_1$

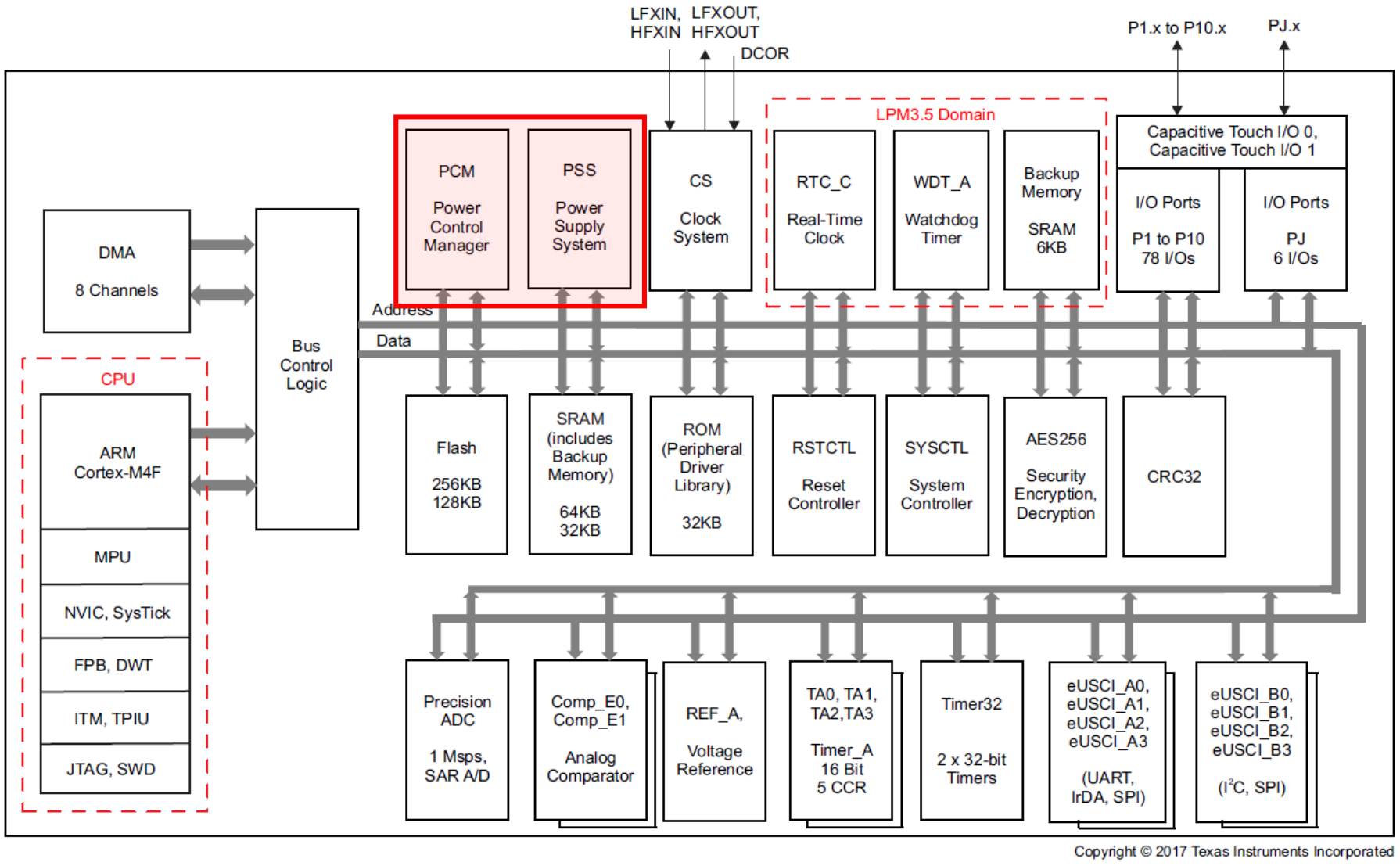
Break-even constraint: $T_w \geq \dfrac{T_{sd} \cdot (P_{sd} - P_s) + T_{wu} \cdot (P_{wu} - P_s)}{P_w - P_s}$

Time constraint: $T_w \geq T_{sd} + T_{wu}$

remove, if power manager has no knowledge about future

break-even time

Power Modes in MSP432 (Lab)



The MSP432 has one active mode in 6 different configurations which all allow for execution of code.

It has 5 major low power modes (LP0, LP3, LP4, LP3.5, LP4.5), some of them can be in one of several configurations.

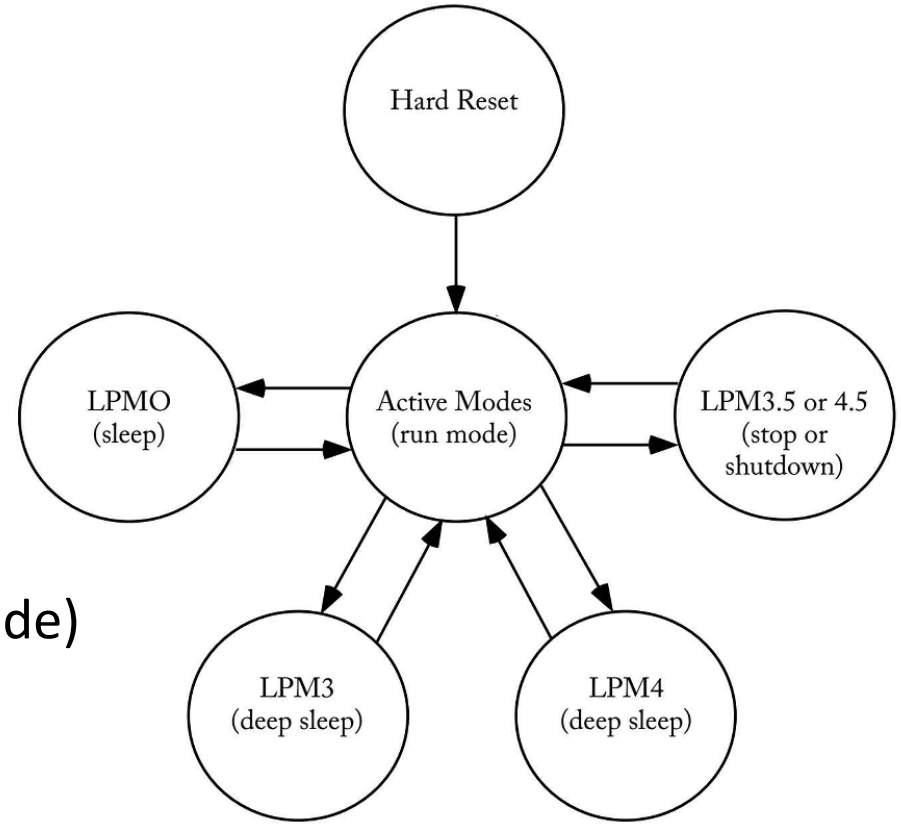
In total, the MSP432 can be in 18 different low power configurations.

active mode (32MHz): 6 - 15 mW ; low power mode (LP4): 1.5 – 2.1 µW

Power Modes in MSP432 (Lab)

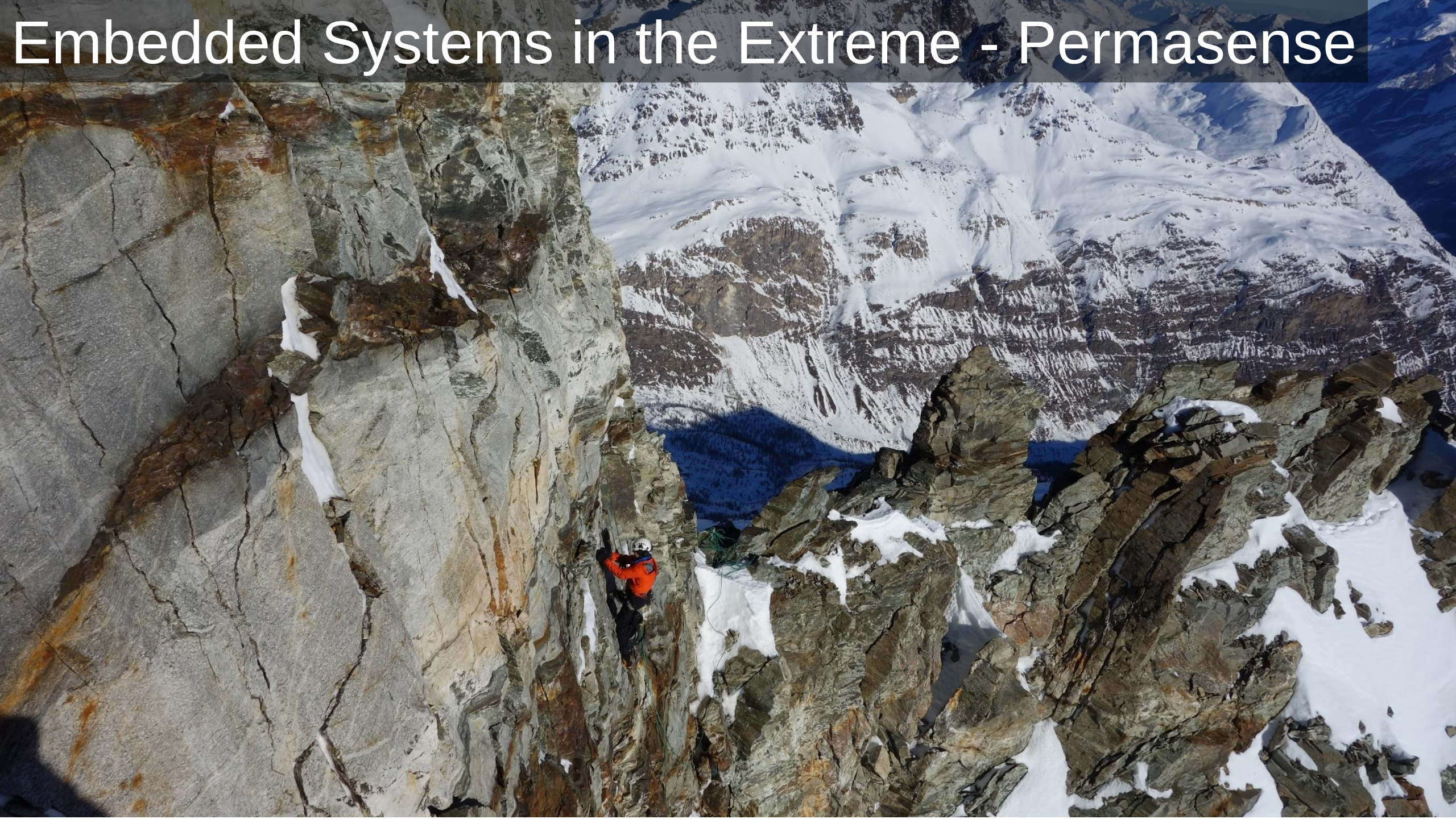
- Transition between modes can be handled using C-level interfaces to the power control manger.

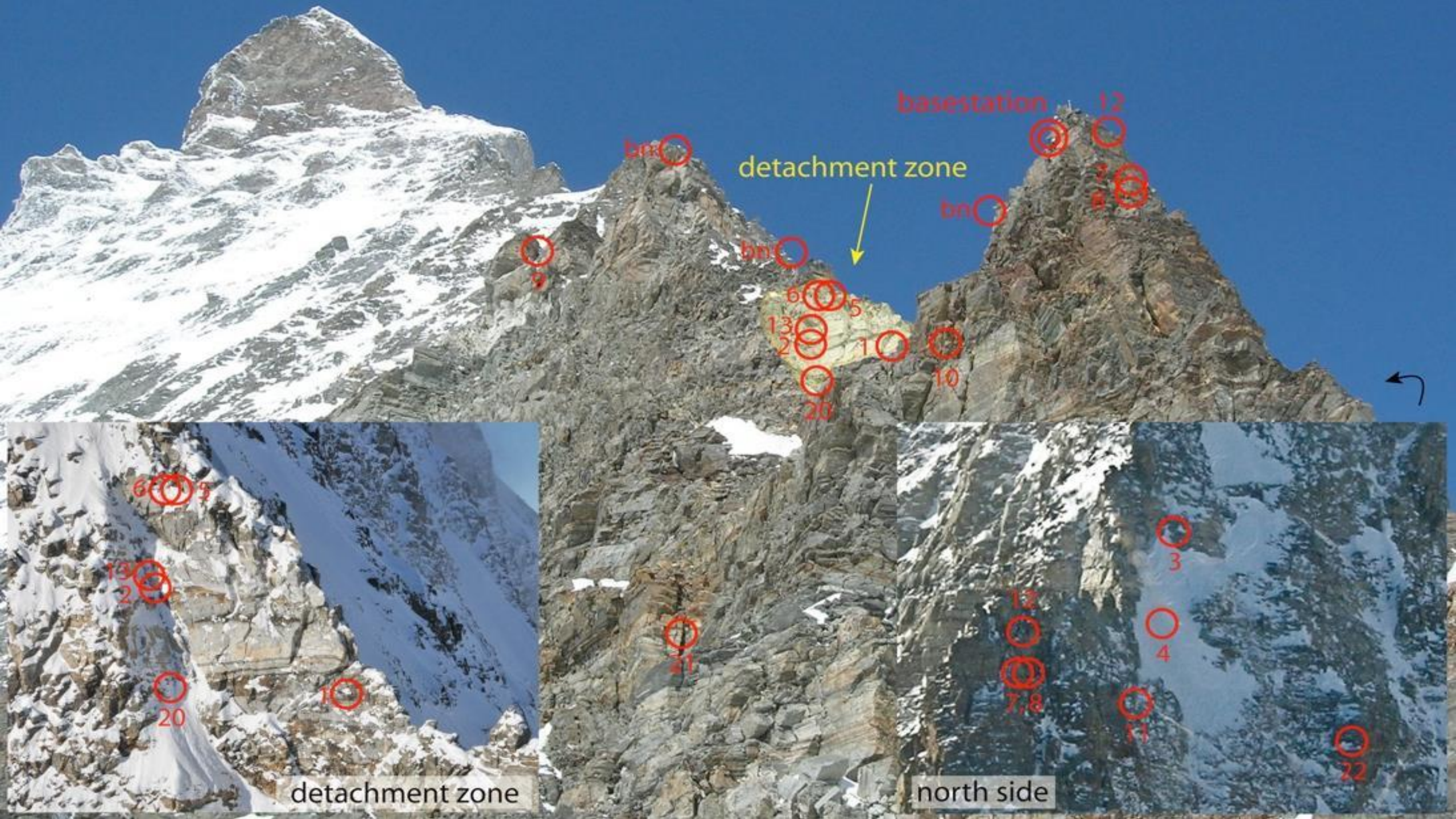
- Examples of interface functions:
  - uint8_t PCM_getPowerState (void)
  - bool PCM_gotoLPM0 (void)
  - bool PCM_gotoLPM3 (void)
  - bool PCM_gotoLPM4 (void)
  - bool PCM_shutdownDevice (uint32_t shutdownMode)



Battery-Operated Systems and Energy Harvesting

Embedded Systems in the Extreme - Permasense

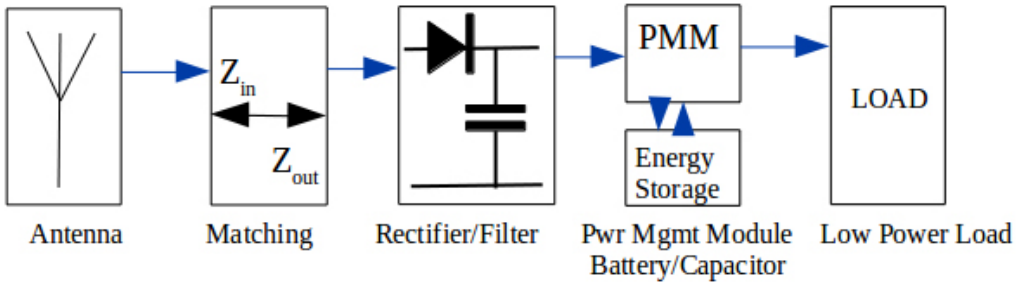



basestation
12
bm
detachment zone
bm
bm
9
6
5
13
2
1
10
20
21
3
4
12
7,8
11
22
detachment zone
north side

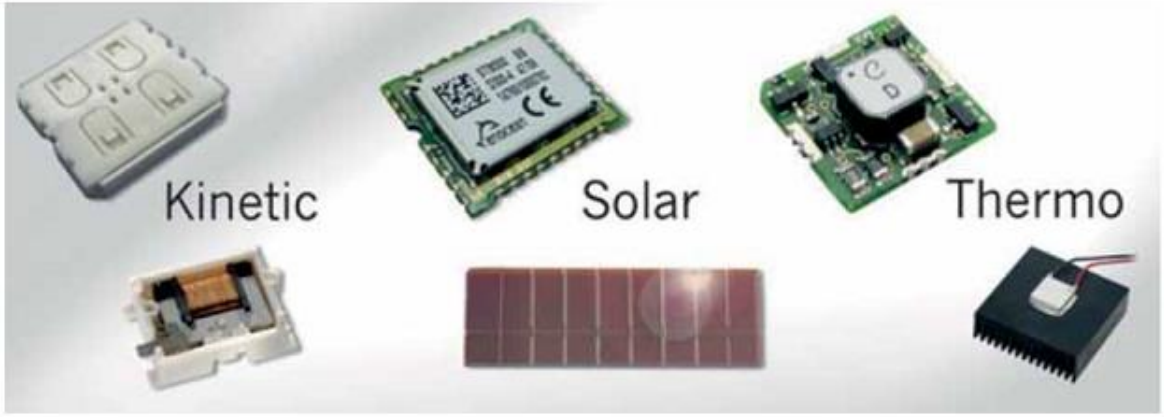
Reasons for Battery-Operated Devices and Harvesting

- Battery operation:
  - no continuous power source available
  - mobility

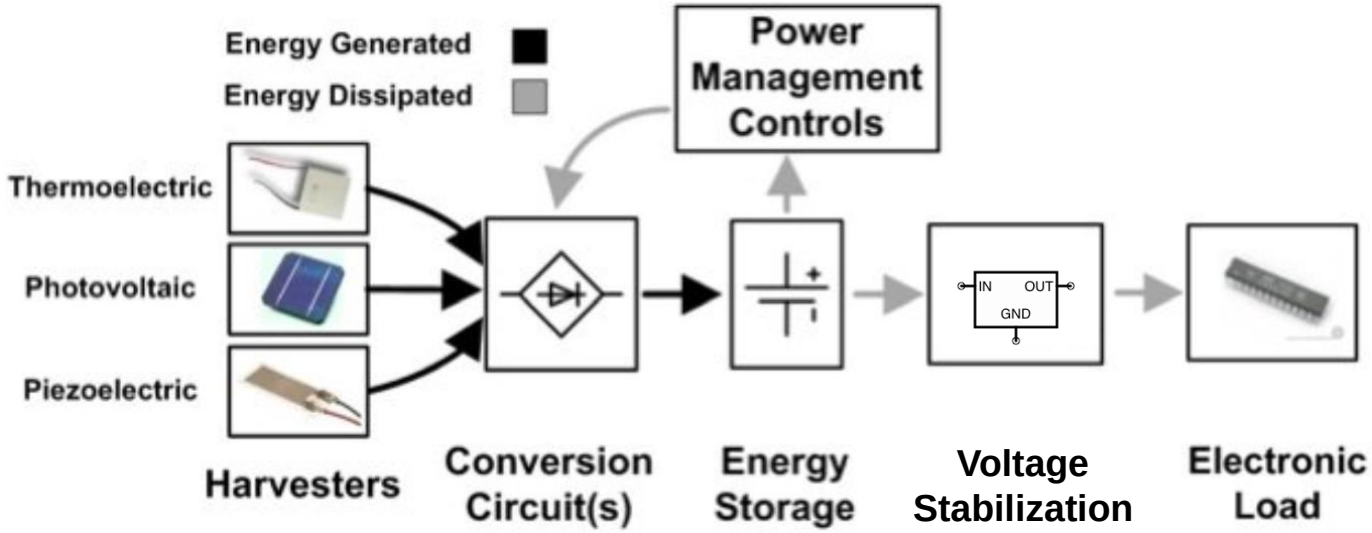
- Energy harvesting:
  - prolong lifetime of battery-operated devices
  - infinite lifetime using rechargeable batteries
  - autonomous operation



radio frequency (RF) harvesting



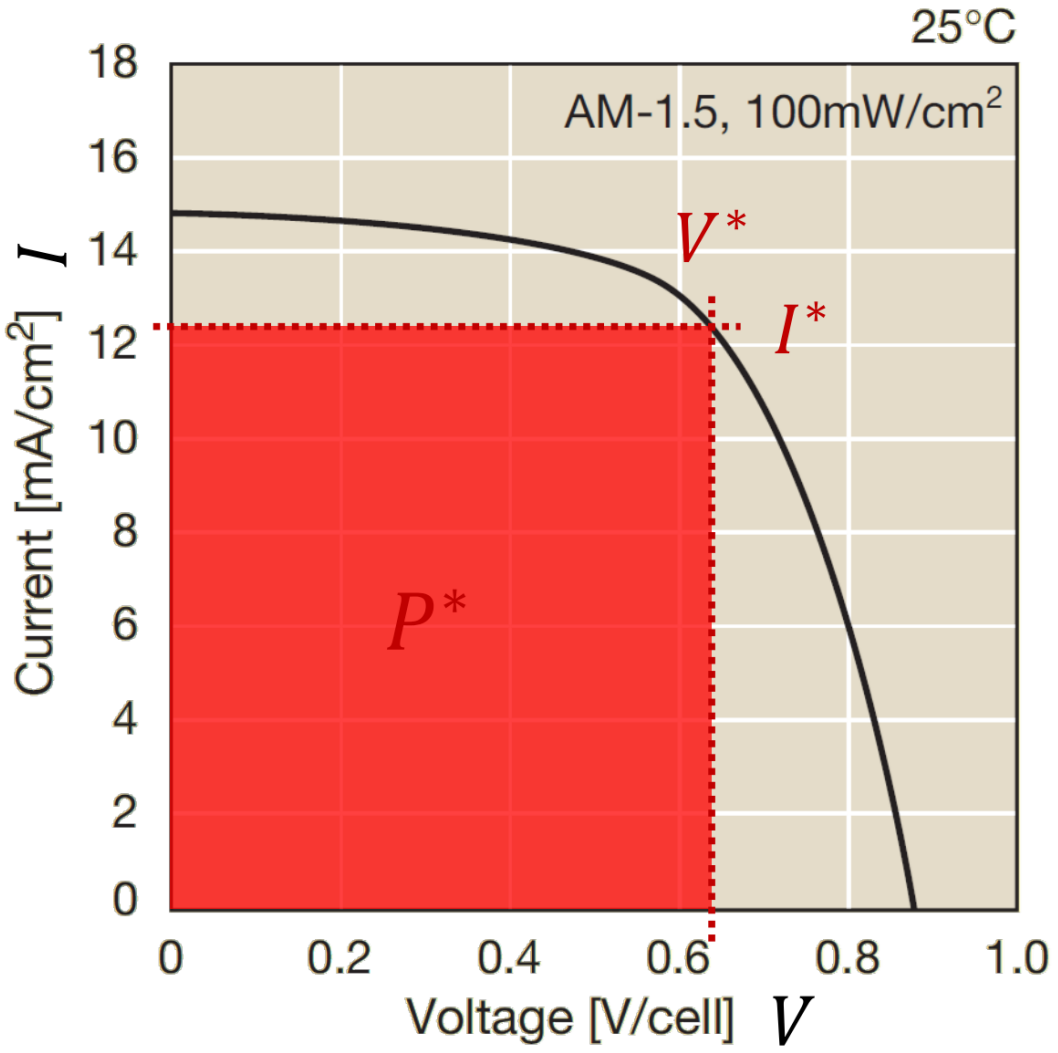
Typical Power Circuitry – Power Point Tracking



power point tracking / impedance matching; conversion to voltage of energy storage

rechargeable battery or supercapacitor

Solar Panel Characteristics

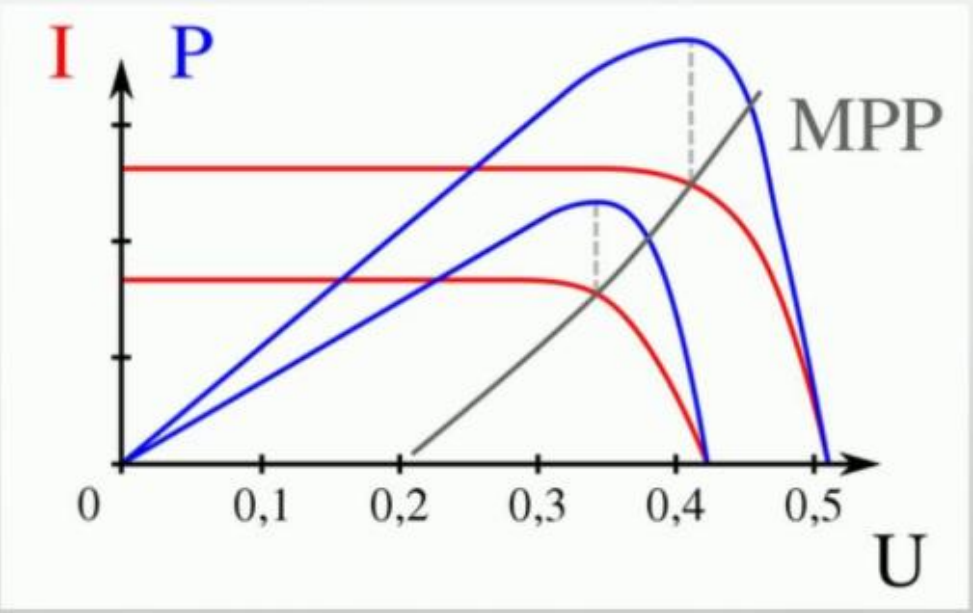


- Variable output power
  - Illuminance level
  - Electrical operation point
  - (Temperature, age, …)
- I-V-Characteristics
  - Non-linear
  - Dependent on ambient
- Maximum Power Point Tracking
  - Dynamic algorithm to find P^*

Diagram: *Amorton Amorphous Silicon Solar Cells Datasheet,* © Panasonic

Typical Power Circuitry – Maximum Power Point Tracking

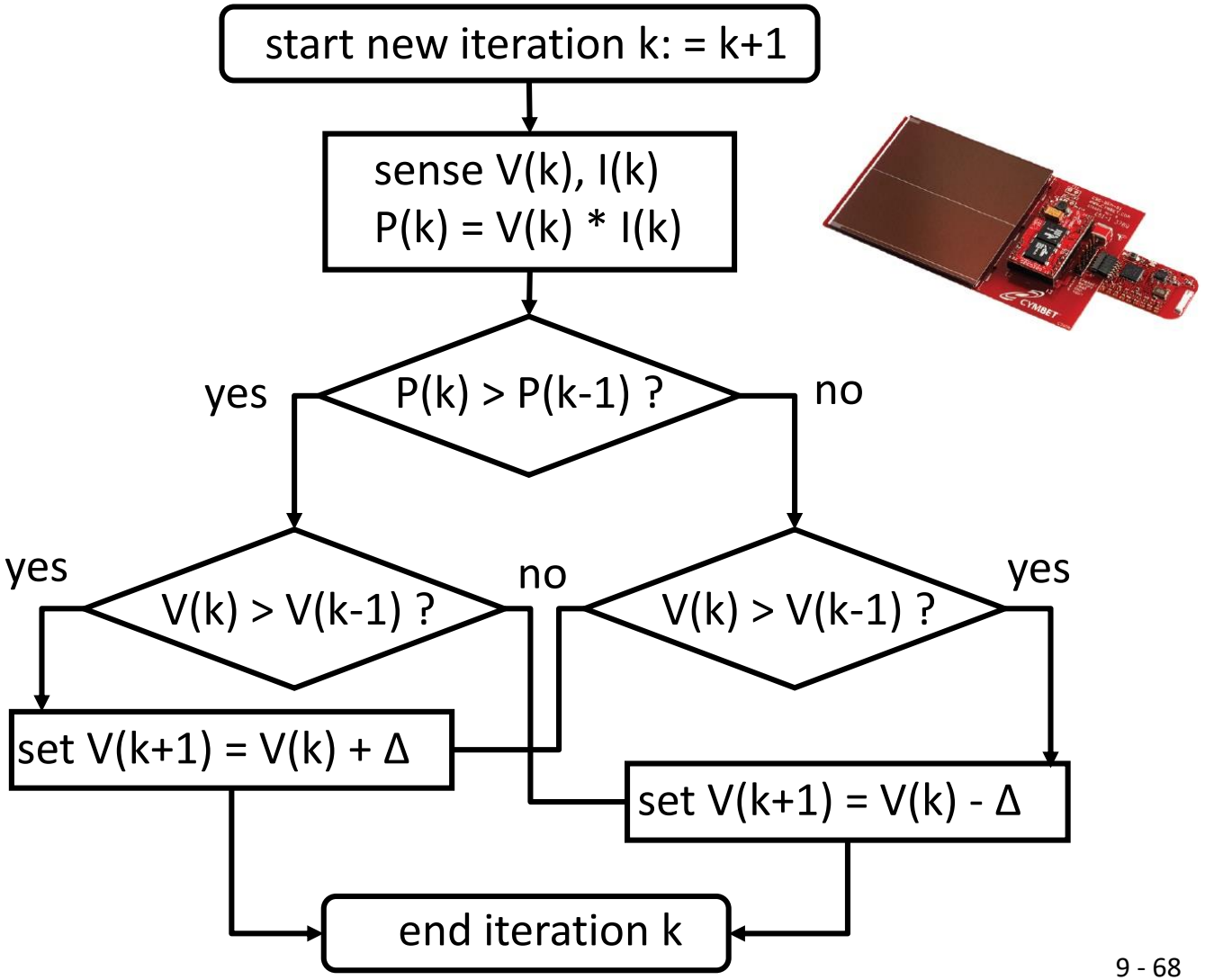
U/I curves of a typical solar cell:



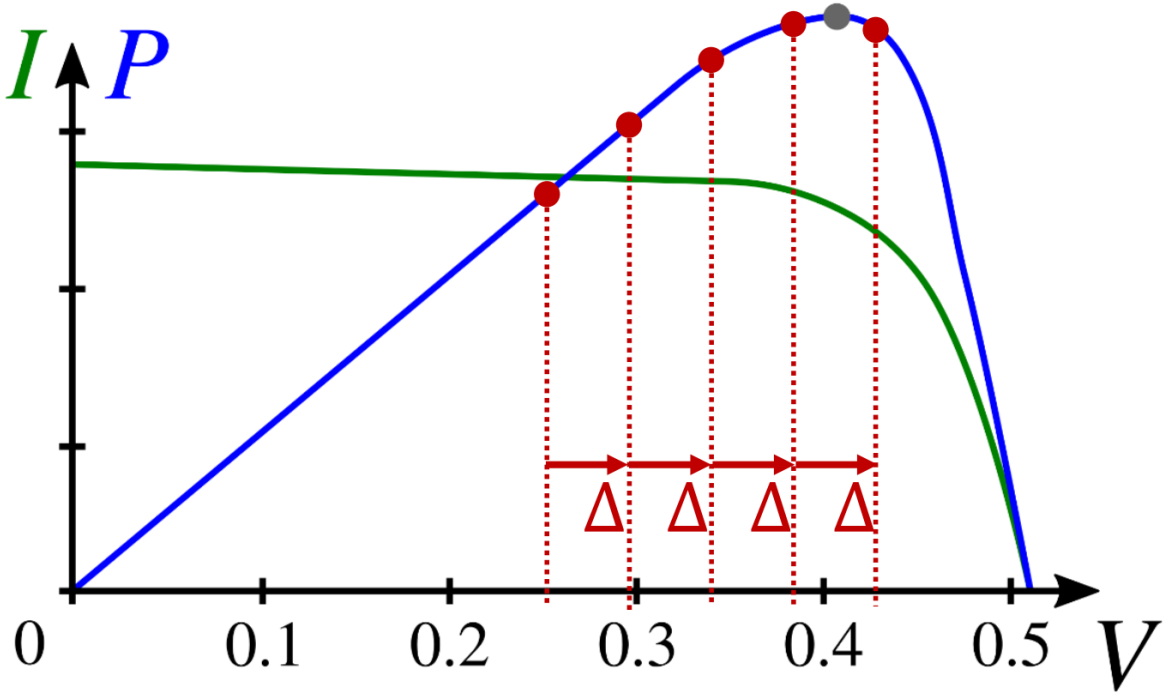
red: current for different light intensities
blue: power for different light intensities
grey: maximal power

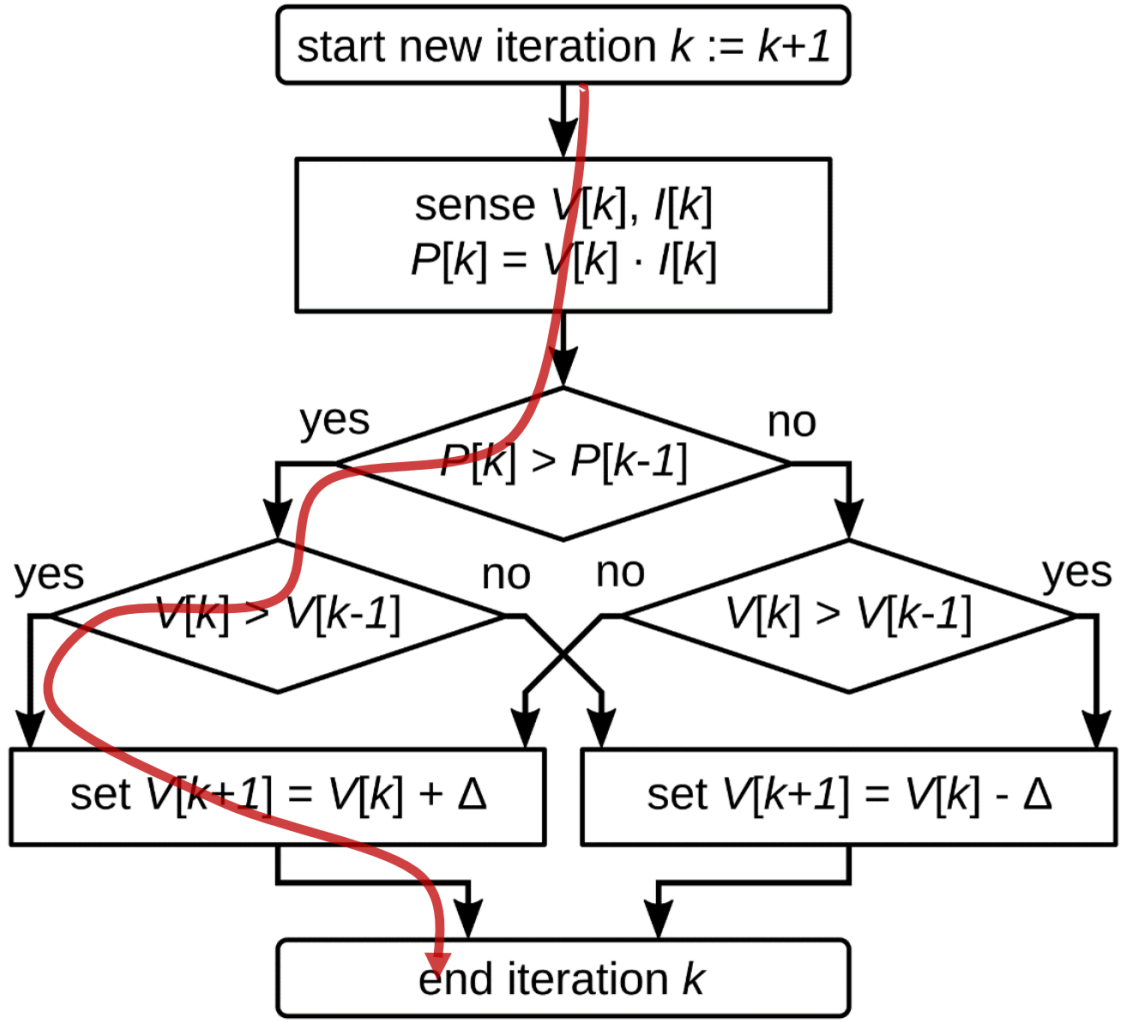
tracking: determine optimal impedance seen by the solar panel

simple tracking algorithm (assume constant illumination) :

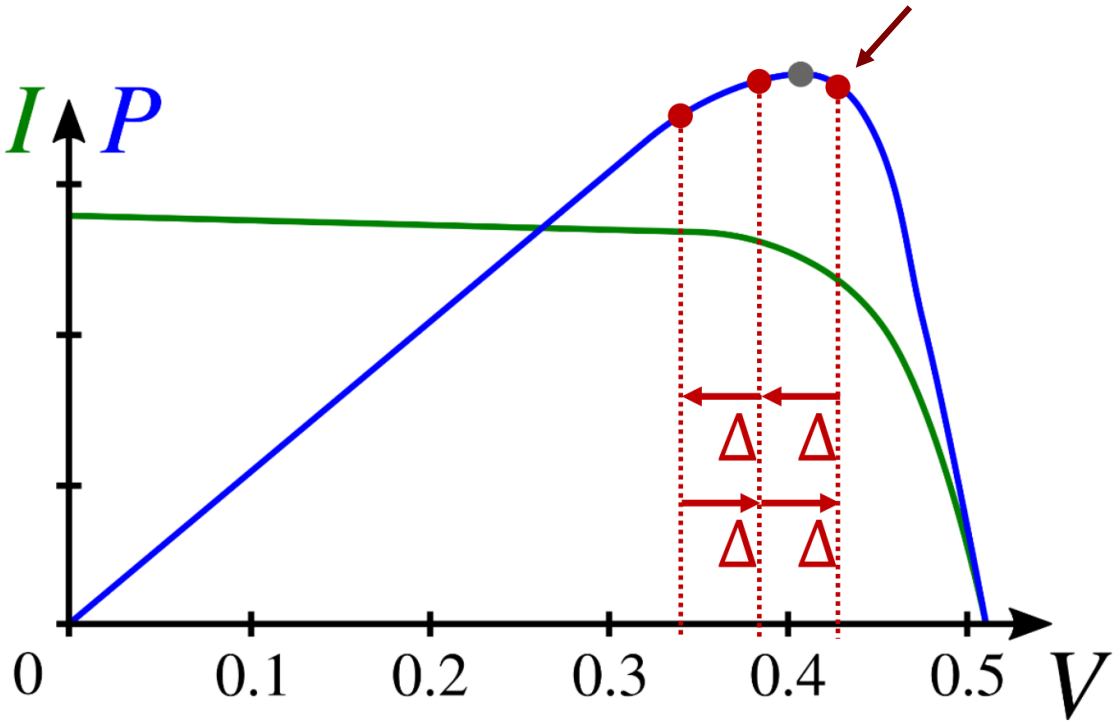


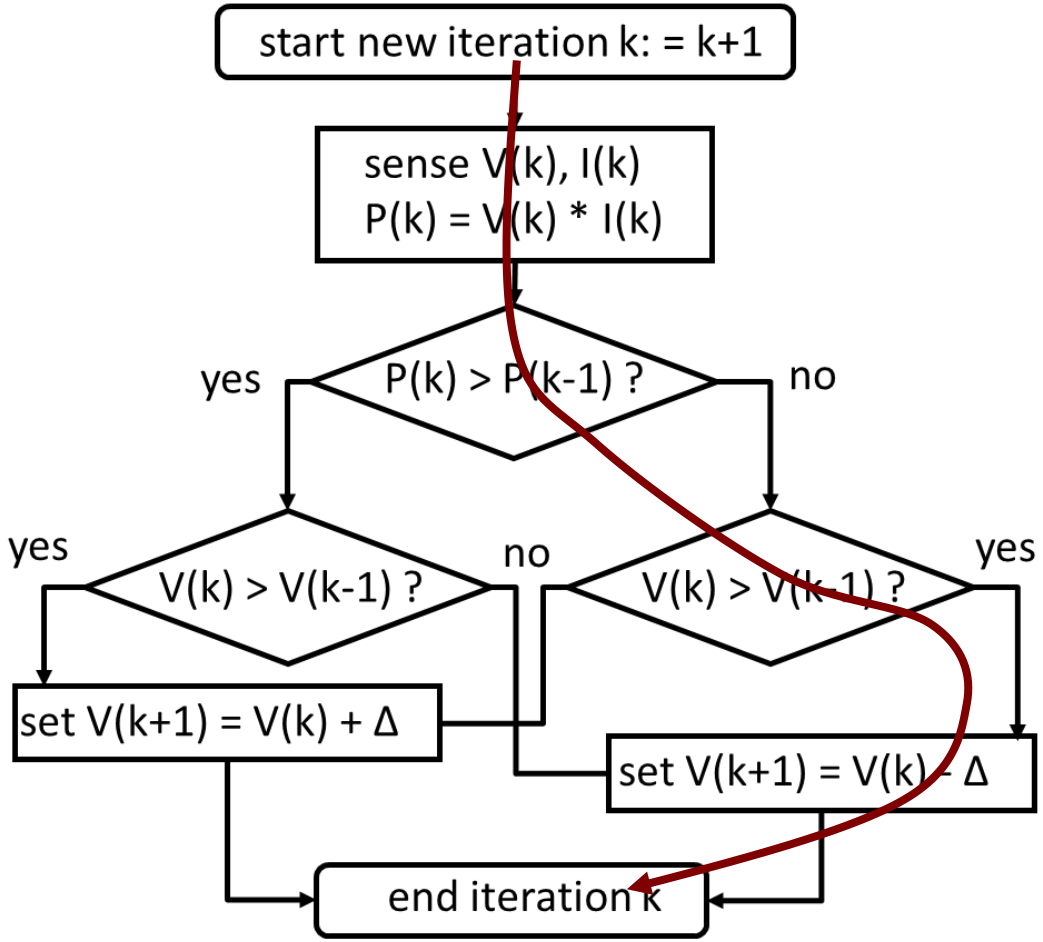
Maximal Power Point Tracking



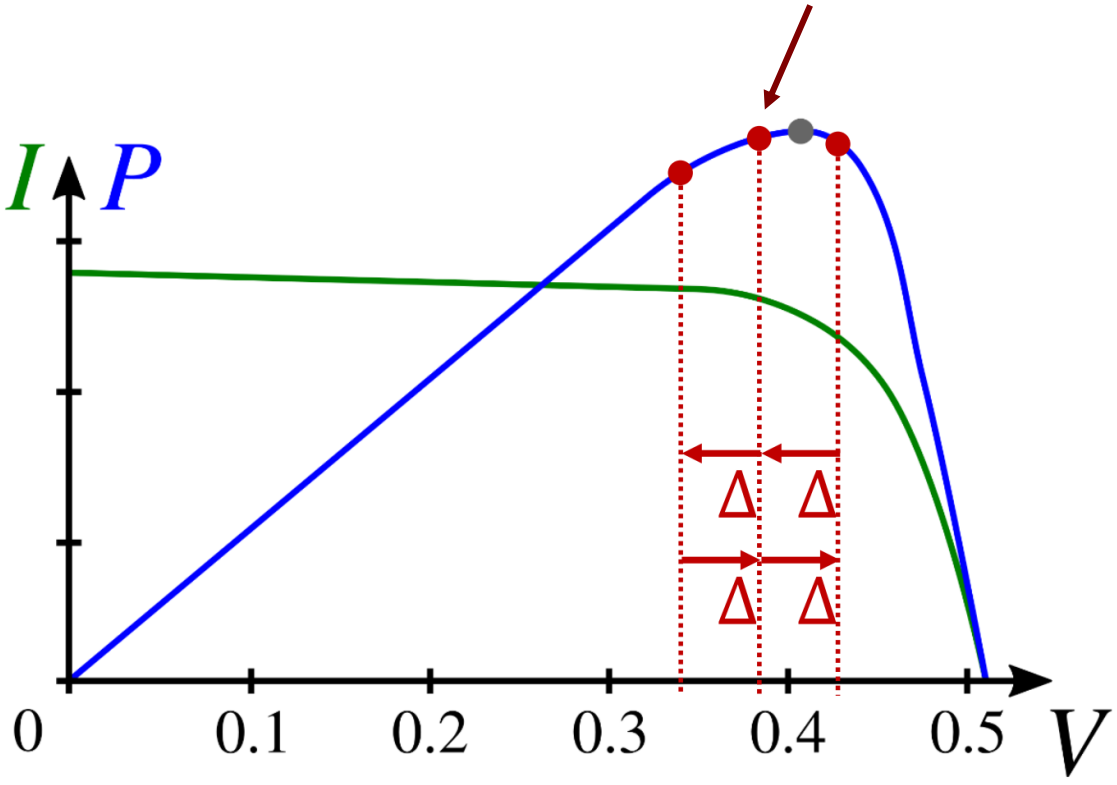


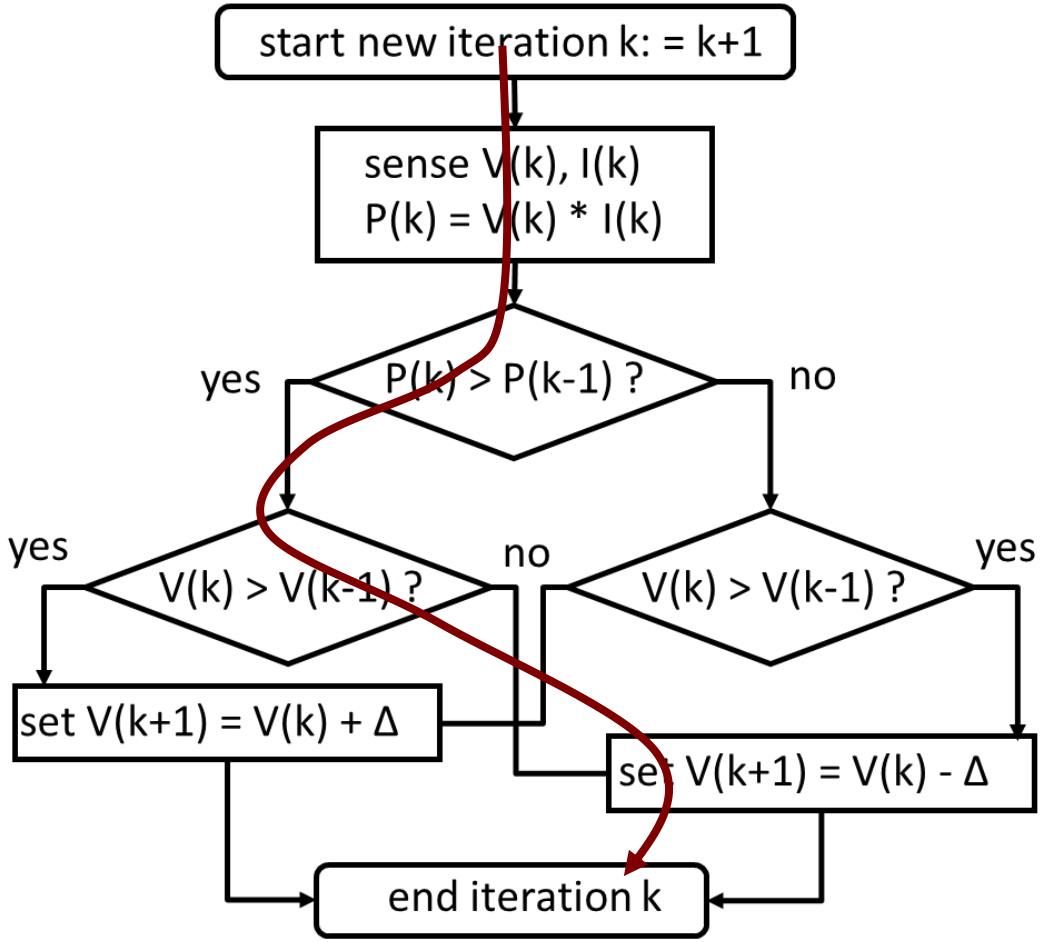
Maximal Power Point Tracking



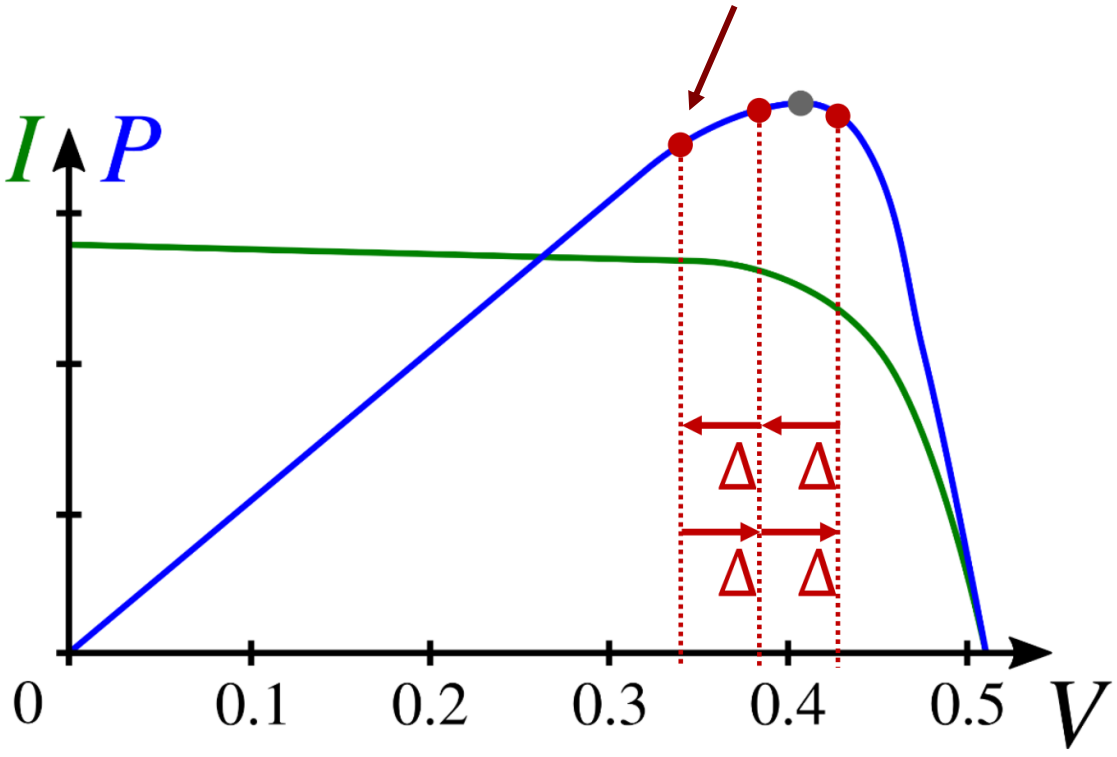


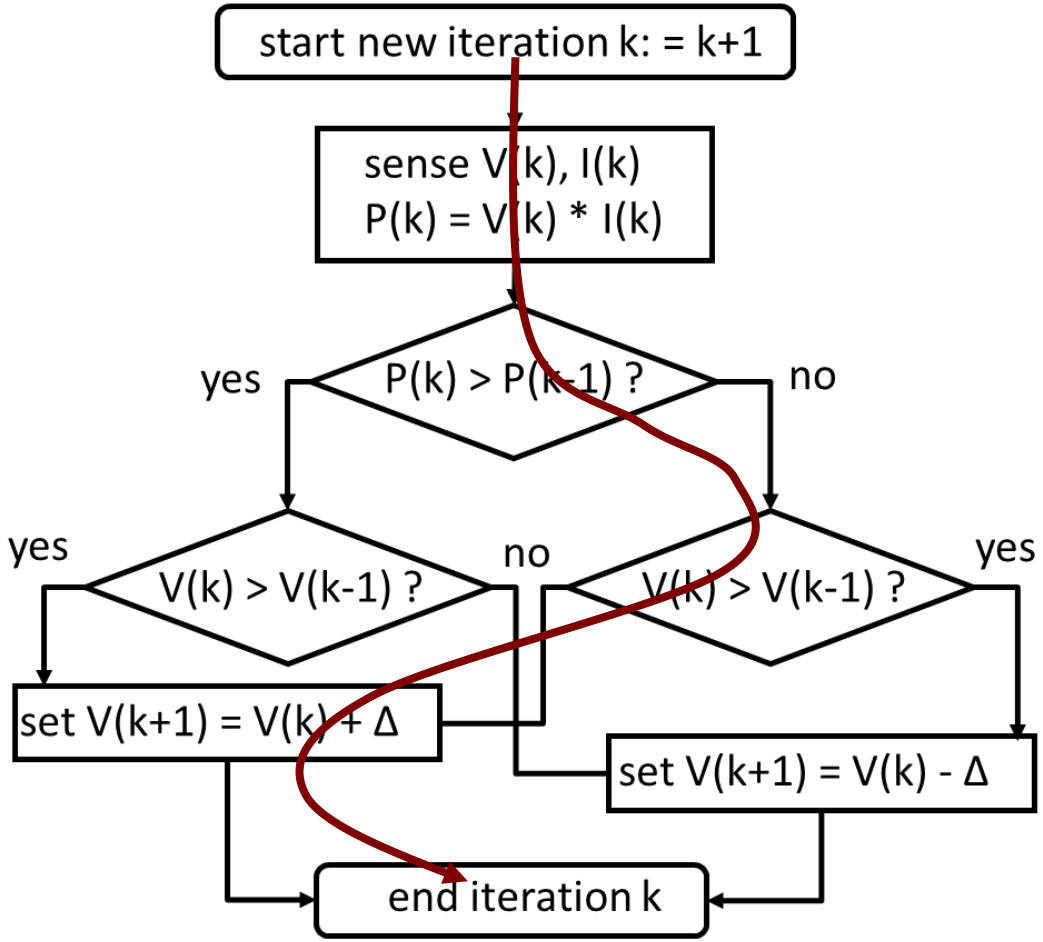
Maximal Power Point Tracking



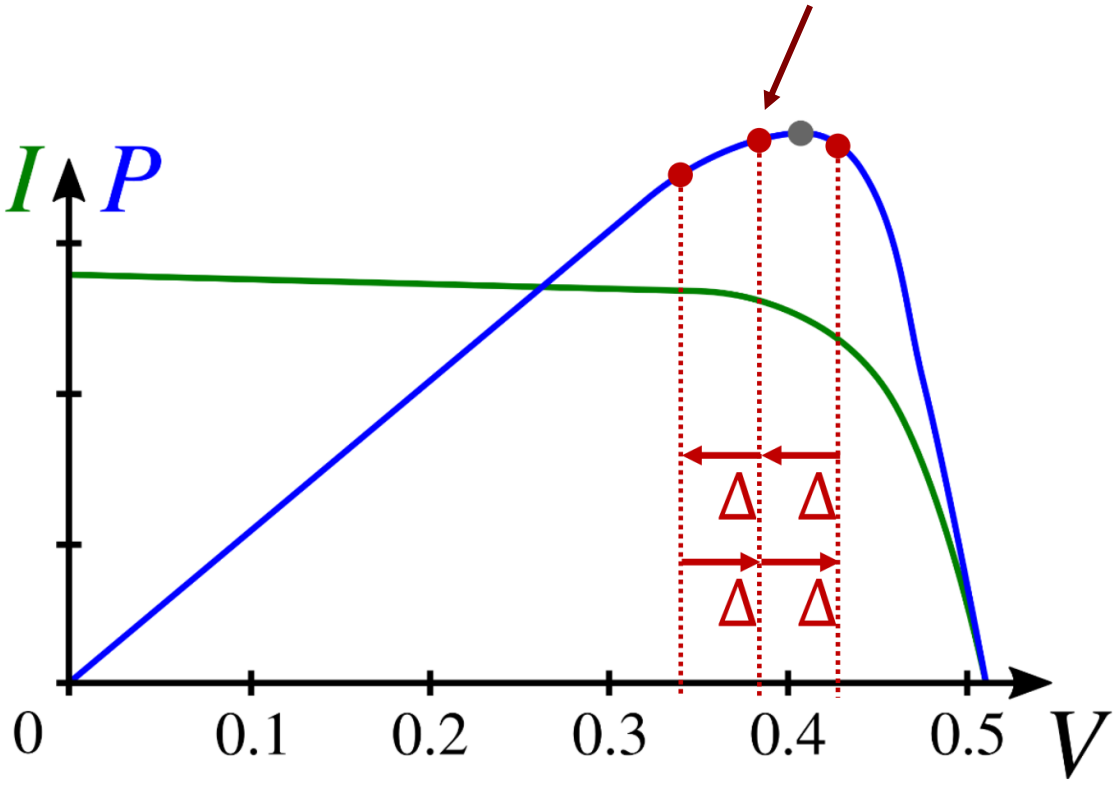


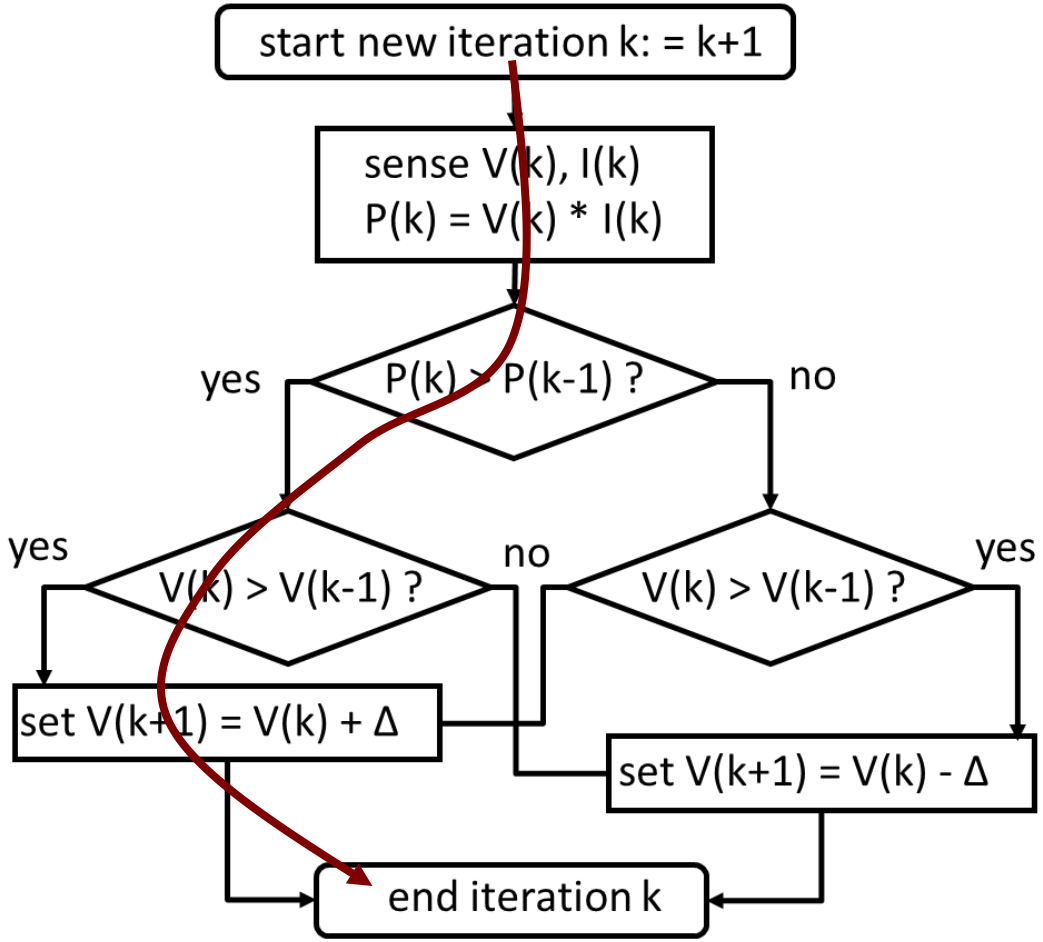
Maximal Power Point Tracking





Maximal Power Point Tracking



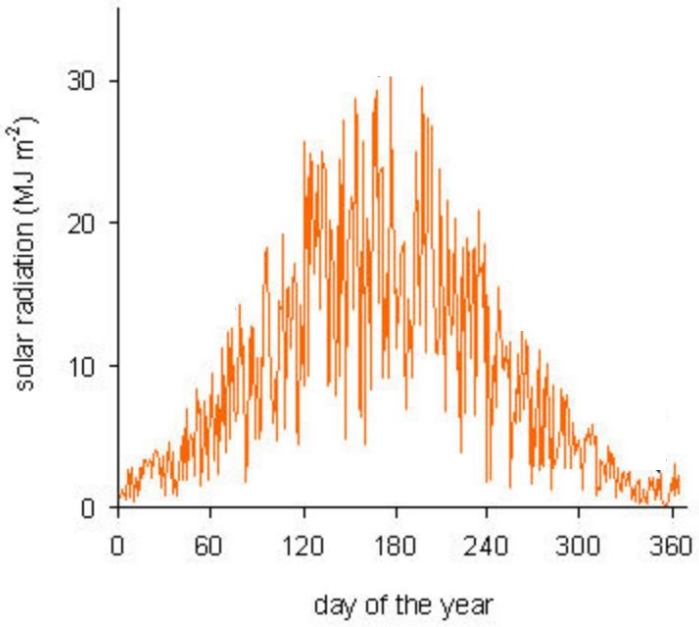


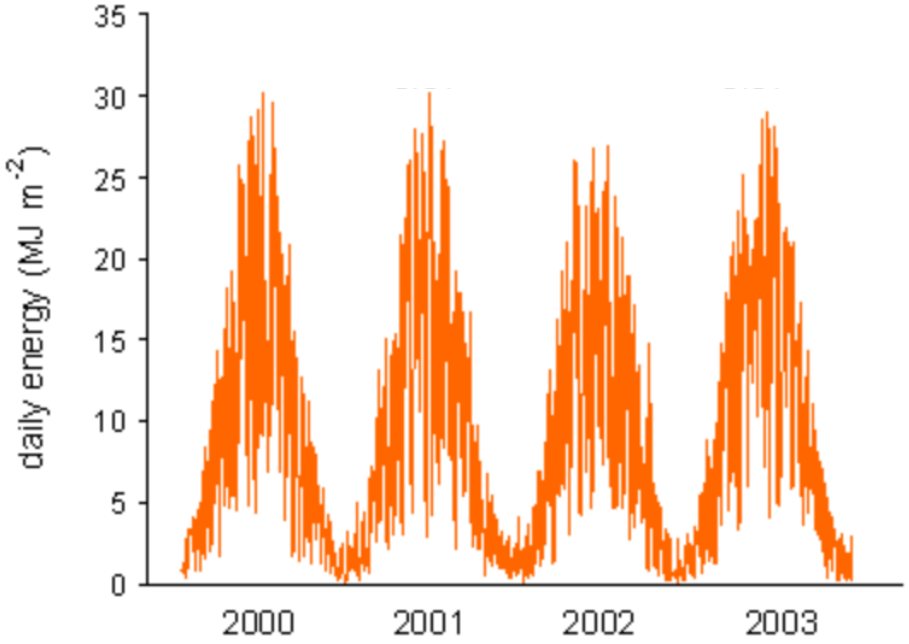
Typical Challenge in (Solar) Harvesting Systems

Challenges:

- What is the optimal maximum capacity of the battery?
- What is the optimal area of the solar cell?
- How can we control the application such that a continuous system operation is possible, even under a varying input energy (summer, winter, clouds)?

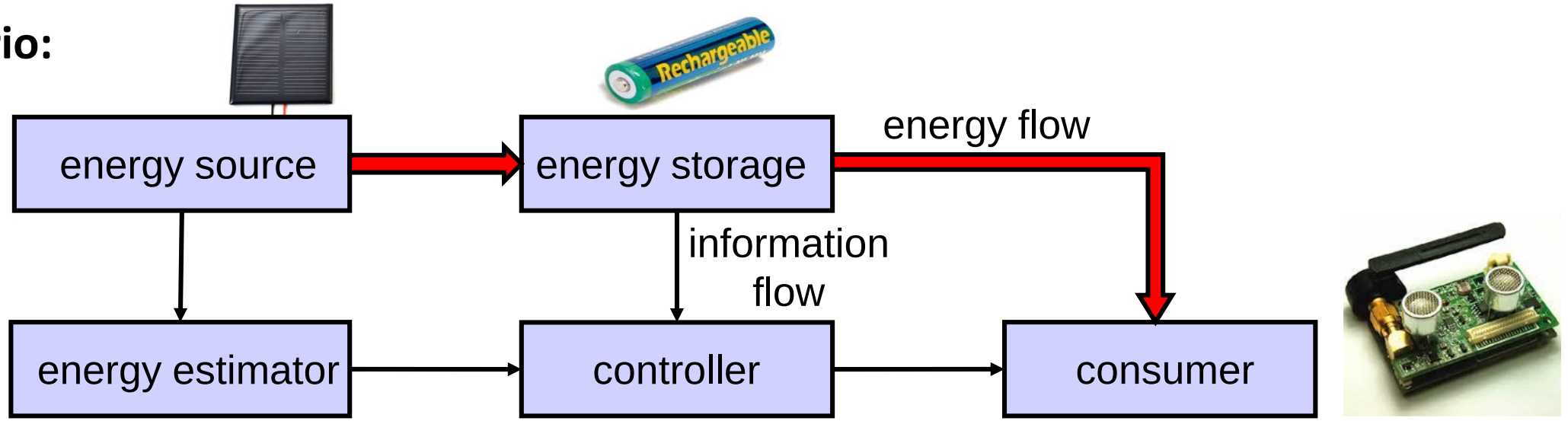
Example of a solar energy trace:





Example: Application Control

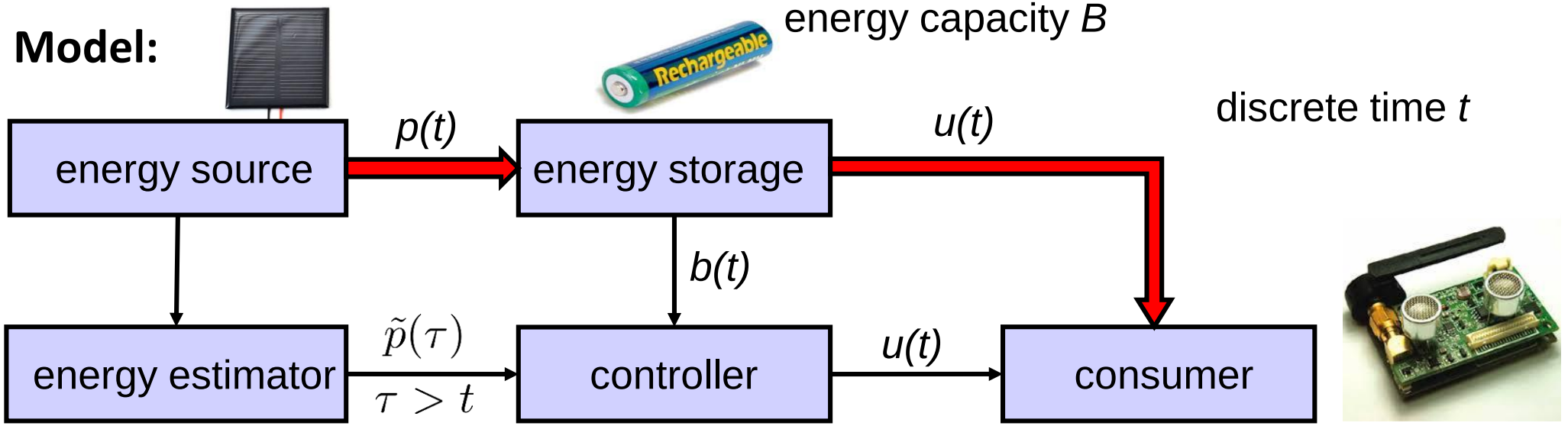
Scenario:



- The controller can adapt the service of the consumer device, for example the sampling rate for its sensors or the transmission rate of information. As a result, the power consumption changes proportionally.
- **Precondition for correctness** of application control: Never run out of energy.
- **Example for optimality criterion**: Maximize the lowest service of (or equivalently, the lowest energy flow to) the consumer.

Application Control

Formal Model:



- harvested and used energy in *[t, t+1)*: *p(t), u(t)*
- battery model: $b(t+1) = \min\{b(t) + p(t) - u(t), B\}$
- failure state: $b(t) + p(t) - u(t) < 0$
- utility:

$$U(t_1, t_2) = \sum_{t_1 \leq \tau < t_2} \mu(u(\tau))$$

μ is a strictly concave function; higher used energy gives a reduced reward for the overall utility.

Application Control

- **What do we want?** We would like to determine an optimal control *u*(t)* for time interval *[t, t+1)* for all *t in [0, T)* with the following properties:
  - $\forall\ 0 \leq t < T\ :\ b^*(t) + p(t) - u^*(t) \geq 0$
  - There is no feasible use function u(t) with a larger minimal energy:

    $$\forall u\ :\ \min_{0 \leq t < T}\{u(t)\} \leq \min_{0 \leq t < T}\{u^*(t)\}$$

  - The use function maximizes the utility *U(0, T).*
  - We suppose that the battery has the same or better state at the end than at the start of the time interval, i.e., *b*(T) ≥ b*(0).*
- We would like to answer two questions:
  - Can we say something about the characteristics of *u*(t)* ?
  - How does an algorithm look like that efficiently computes *u*(t)* ?

Application Control

Theorem: Given a use function u*(t), $t \in [0, T)$ such that the system never enters a failure state. If u*(t) is optimal with respect to maximizing the minimal used energy among all use functions and maximizes the utility U(t, T), then the following relations hold for all $\tau \in (0, T)$:

$$u^*(\tau - 1) < u^*(\tau) \Longrightarrow b^*(\tau) = 0$$

empty battery

$$u^*(\tau - 1) > u^*(\tau) \Longrightarrow b^*(\tau) = B$$

full battery

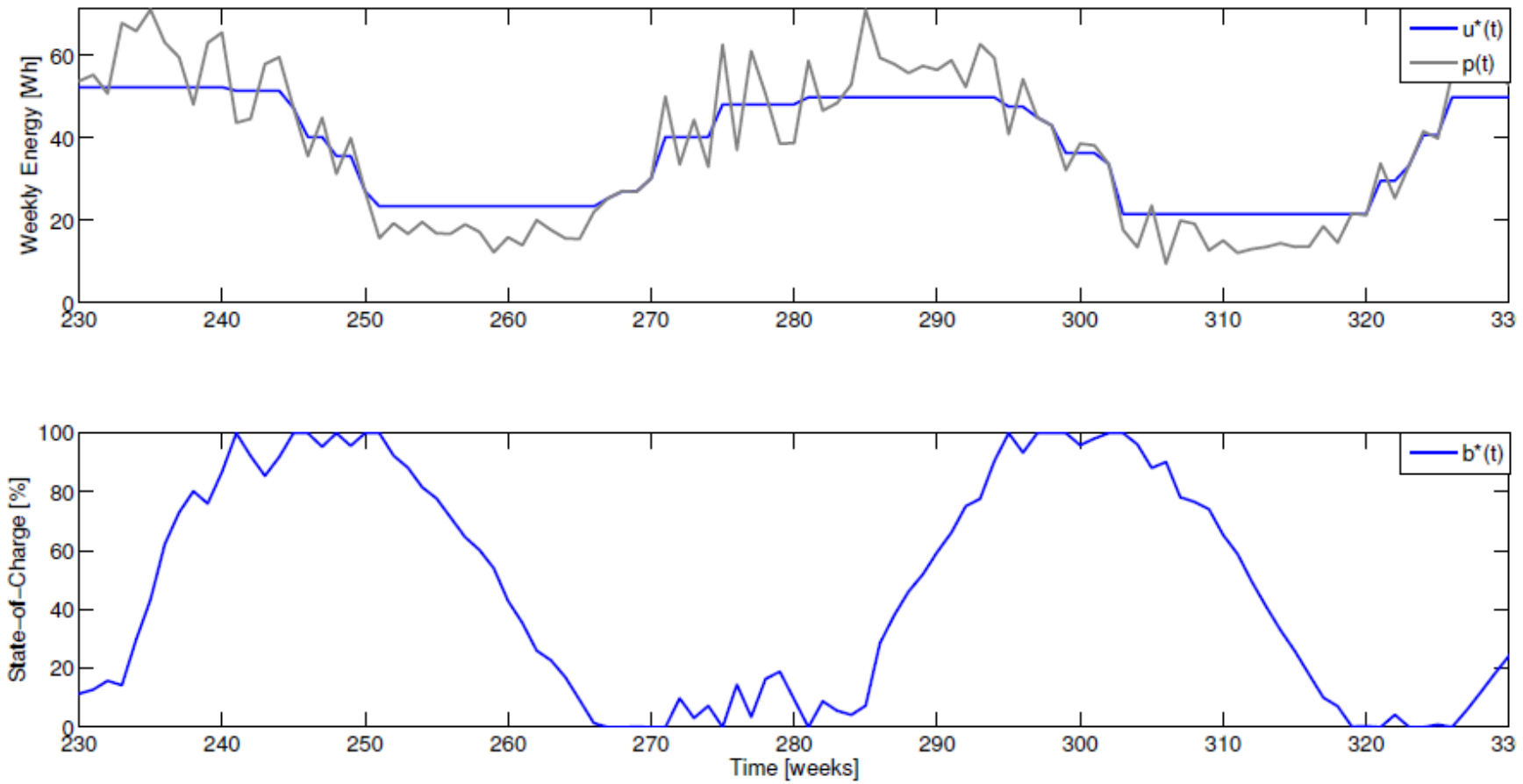
Sketch of a proof: First, let us show that a consequence of the above theorem is true (just reverting the relations):

$$\forall \tau \in (s, t] \ : \ 0 < b^*(\tau) < B \implies \forall \tau \in [s, t] \ : \ u^*(\tau) = u^*(t)$$

In other words, as long as the battery is neither full nor empty, the optimal use function does not change.

Application Control

- Proof sketch cont.:

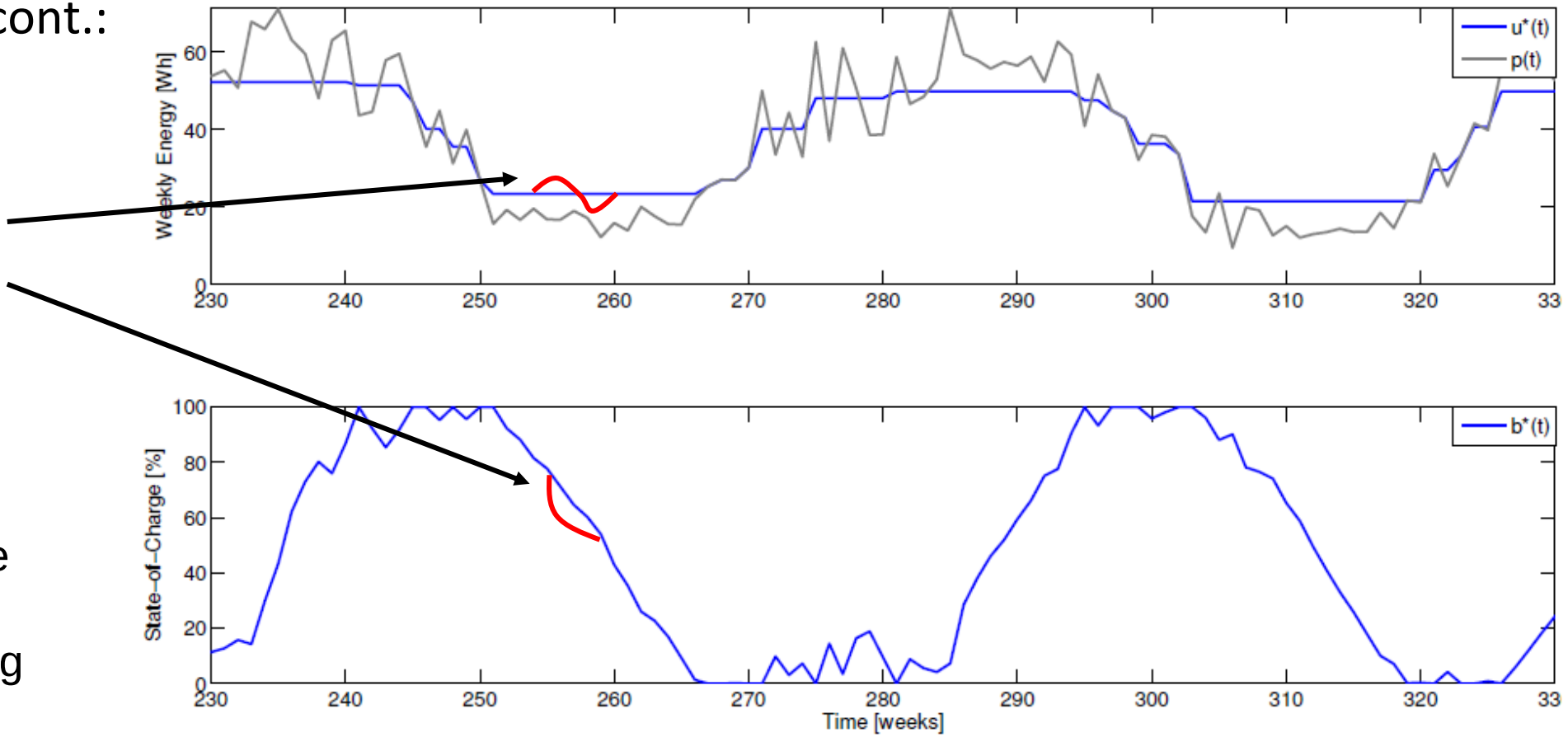


(top) Example of an optimal use function $u^*(t)$ for a given harvest function $p(t)$ and (bottom) the corresponding stored energy $b^*(t)$.

Application Control

- Proof sketch cont.:

suppose we change the use function locally from being constant such that the overall battery state does not change



(top) Example of an optimal use function $u^*(t)$ for a given harvest function $p(t)$ and (bottom) the corresponding stored energy $b^*(t)$.

then the utility is worse due to the concave function μ: diminishing reward for higher use function values; and the minimal use function is potentially smaller

Application Control

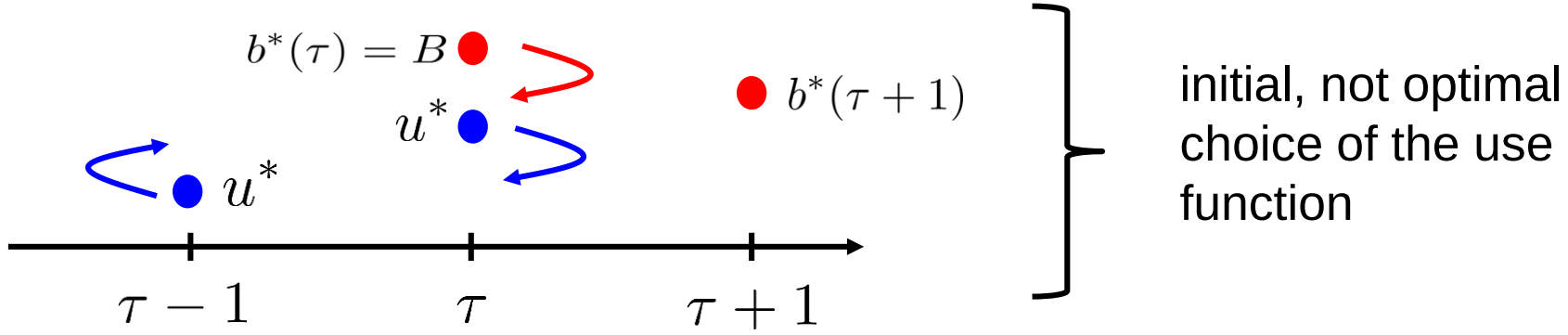
- Proof sketch cont.: Now we show that for all $\tau \in (t, T)$

$$u^*(\tau - 1) < u^*(\tau) \Longrightarrow b^*(\tau) = 0$$

or equivalently

$$b^*(\tau) > 0 \Longrightarrow u^*(\tau - 1) \geq u^*(\tau)$$

We already have shown this for $0 < b^*(\tau) < B$. Therefore, we only need to show that $b^*(\tau) = B \Longrightarrow u^*(\tau - 1) \geq u^*(\tau)$. Suppose now that we have $u^*(\tau - 1) < u^*(\tau)$ if the battery is full at τ. Then we can increase the use at time $\tau - 1$ and decrease it at time τ by the same amount without changing the battery level at time $\tau + 1$. This again would increase the overall utility and potentially increase the minimal use function.



Application Control

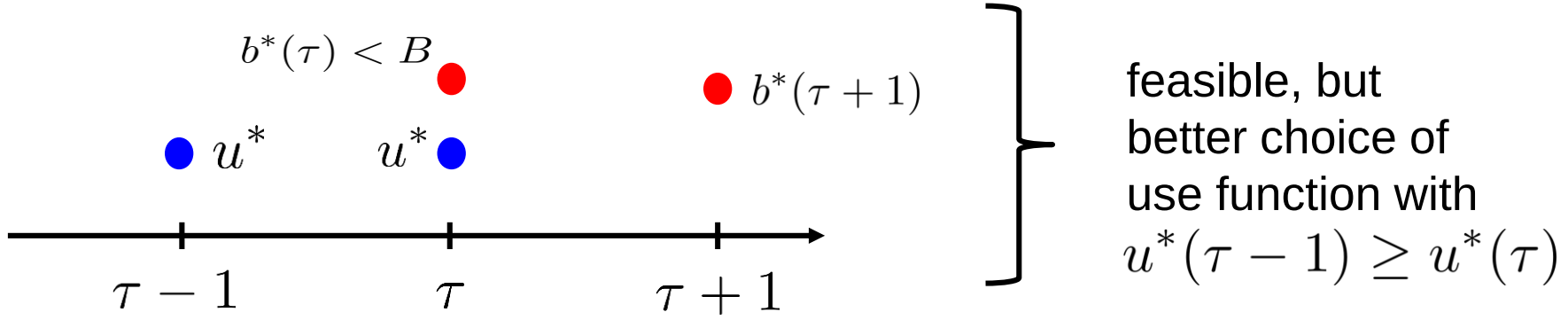
- Proof sketch cont.: Now we show that for all $\tau \in (t, T)$

$$u^*(\tau - 1) < u^*(\tau) \Longrightarrow b^*(\tau) = 0$$

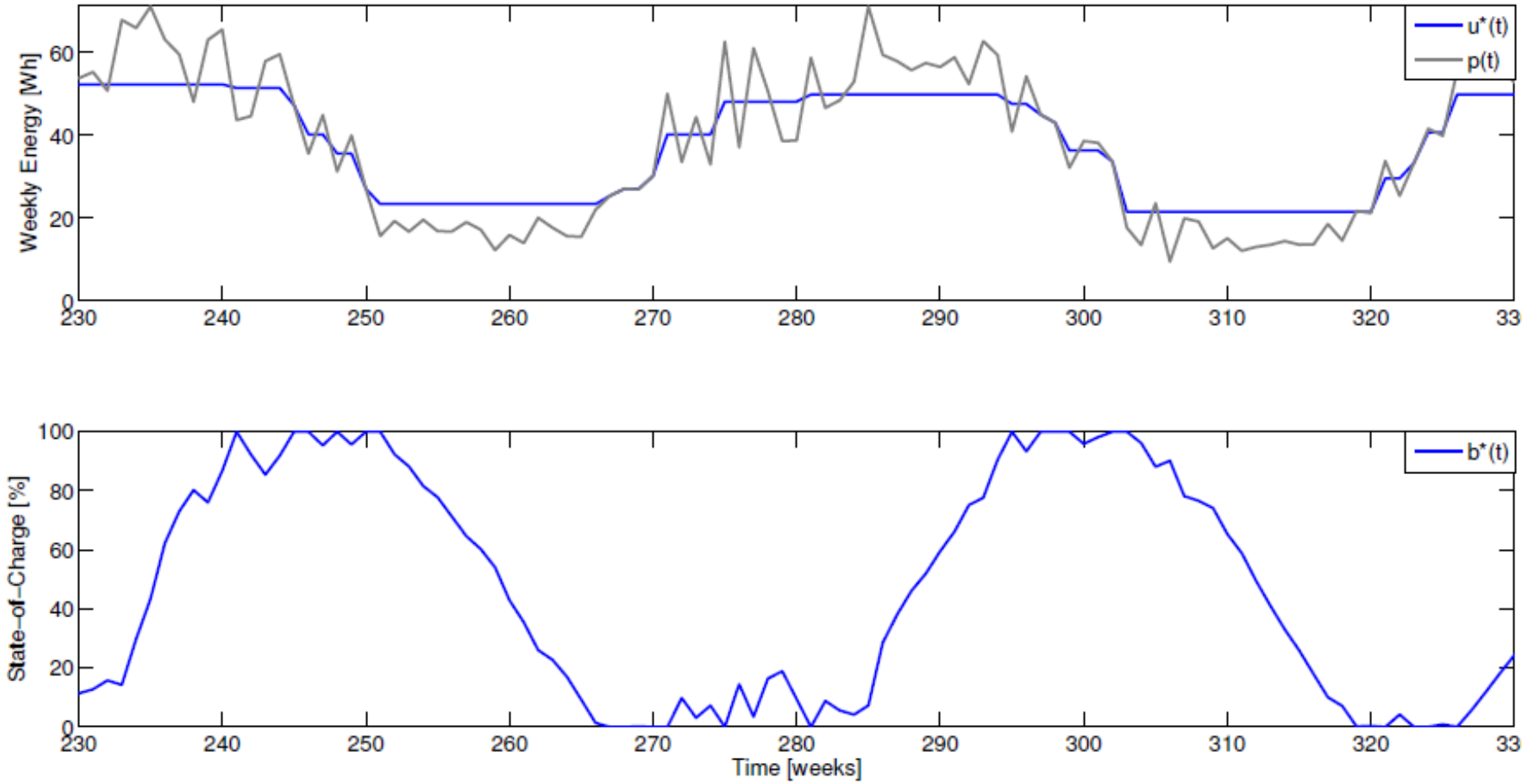
or equivalently

$$b^*(\tau) > 0 \Longrightarrow u^*(\tau - 1) \geq u^*(\tau)$$

We already have shown this for $0 < b^*(\tau) < B$. Therefore, we only need to show that $b^*(\tau) = B \Longrightarrow u^*(\tau - 1) \geq u^*(\tau)$. Suppose now that we have $u^*(\tau - 1) < u^*(\tau)$ if the battery is full at τ. Then we can increase the use at time $\tau - 1$ and decrease it at time τ by the same amount without changing the battery level at time $\tau + 1$. This again would increase the overall utility and potentially increase the minimal use function.



Application Control



(top) Example of an optimal use function $u^*(t)$ for a given harvest function $p(t)$ and (bottom) the corresponding stored energy $b^*(t)$.

Application Control

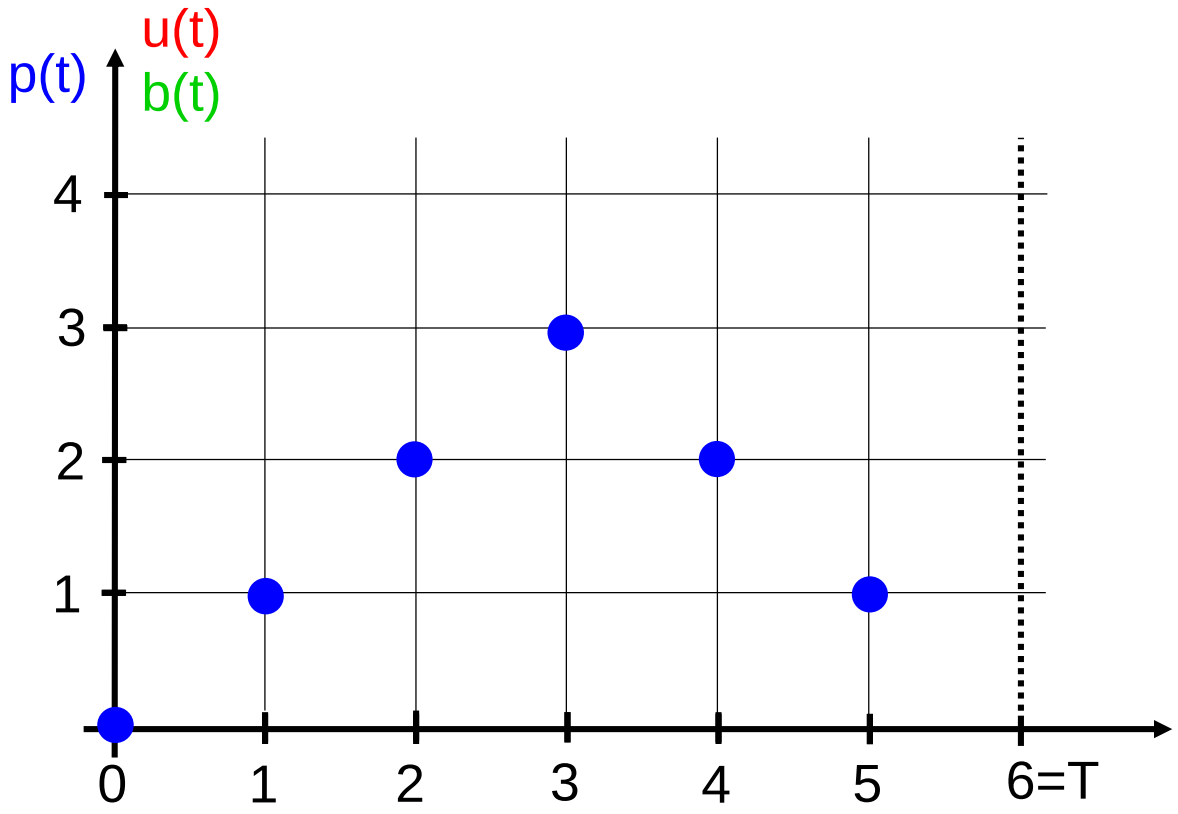
- How can we efficiently compute an optimal use function?
  - There are several options available as we just need to solve a convex optimization problem.
  - A simple but inefficient possibility is to convert the problem into a linear program. At first suppose that the utility is simply

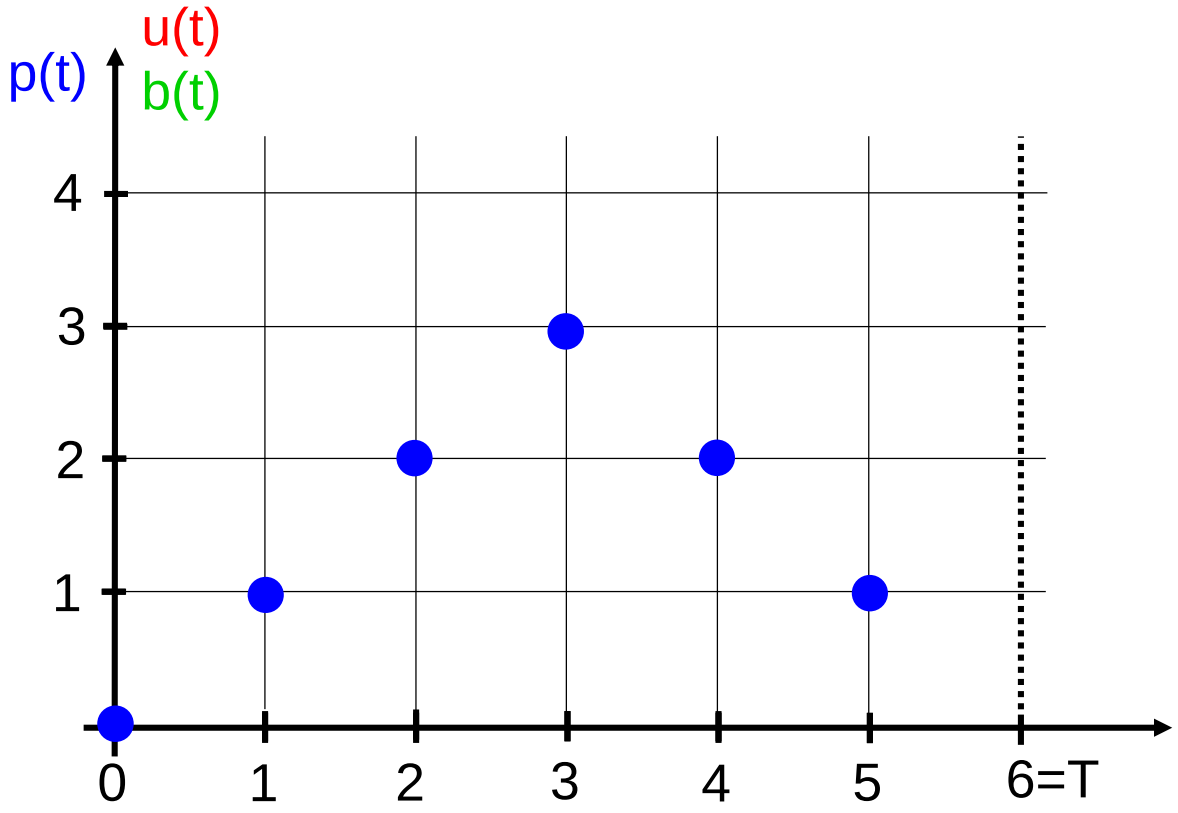
$$U(0,T) = \sum_{0 \le \tau < T} u(\tau)$$

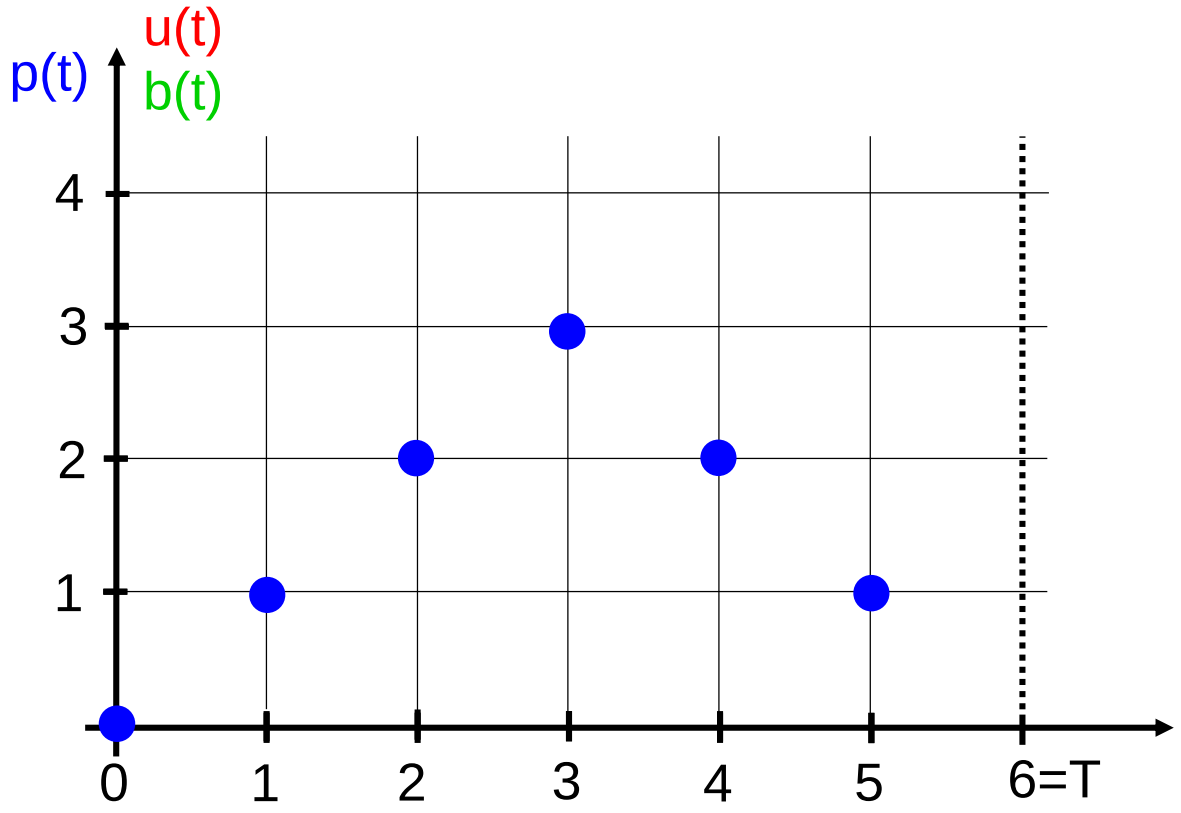
Then the linear program has the form:

[Concave functions μ could be piecewise linearly approximated. This is not shown here.]

$$\begin{aligned}
&\text{maximize} \quad \sum_{0 \le \tau < T} u(\tau) \\
&\forall \tau \in [0,T) \ : \ b(\tau+1) = b(\tau) - u(\tau) + p(\tau) \\
&\forall \tau \in [0,T) \ : \ 0 \le b(\tau+1) \le B \\
&\forall \tau \in [0,T) \ : \ u(\tau) \ge 0 \\
&b(T) = b(0) = b_0
\end{aligned}$$


u(t)
p(t)
b(t)
4
3
2
1
0
1
2
3
4
5
6=T


u(t)
b(t)
p(t)
4
3
2
1
0
1
2
3
4
5
6=T

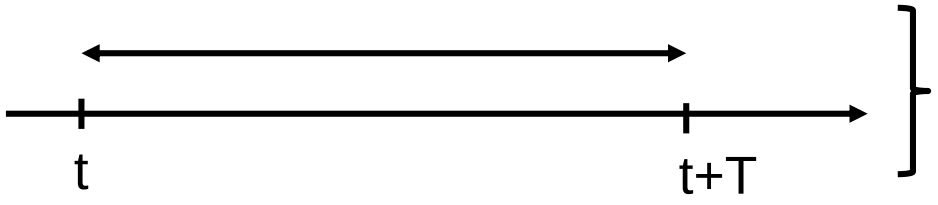

u(t)
b(t)
p(t)
4
3
2
1
0
1
2
3
4
5
6=T

Application Control

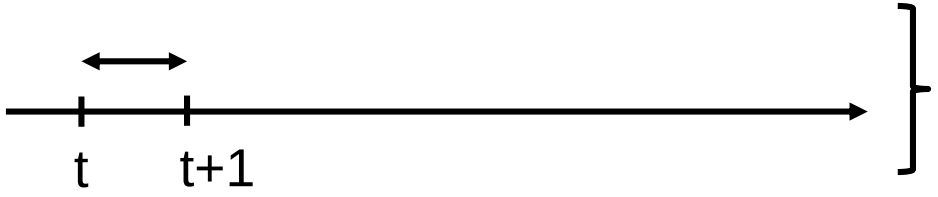
- But what happens if the estimation of the future incoming energy is not correct?
  - If it would be correct, then we would just compute the whole future application control now and would not change anything anymore.
  - This will not work as errors will accumulate and we will end up with many infeasible situations, i.e., the battery is completely empty and we are forced to stop the application.
  - **Possibility**: Finite horizon control
    - At time t, we compute the optimal control (see previous slides) using the currently available battery state b(t) with predictions $\tilde{p}(\tau)$ for all $t \leq \tau < t+T$ and $b(t+T) = b(t)$.
    - From the computed optimal use function $u(\tau)$ for all $t \leq \tau < t+T$ we just take the first use value *u(t)* in order to control the application.
    - At the next time step, we take as initial battery state the actual state; therefore, we take mispredictions into account. For the estimated future energy, we also take the new estimations.

Application Control

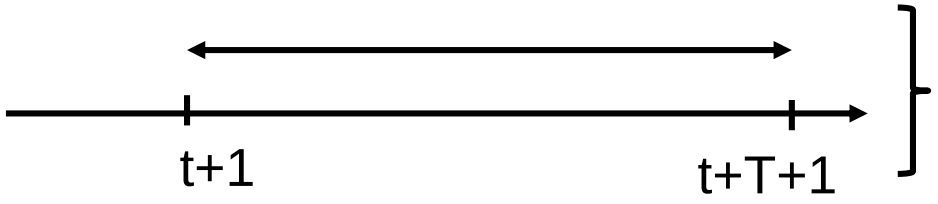
- **Finite horizon control**:



compute the optimal use function in [t, t+T)
using the actual battery state at time t

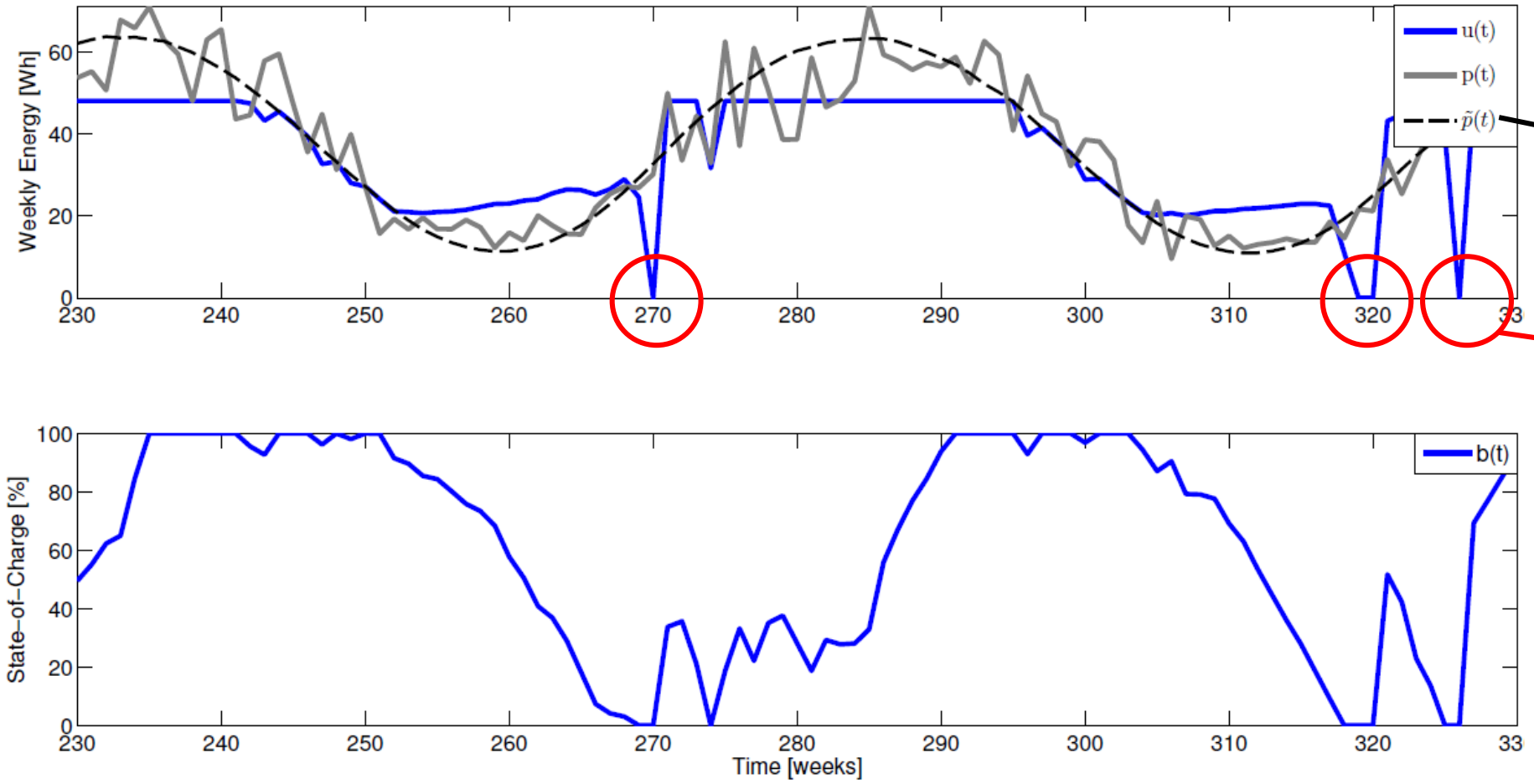


apply this use function in the interval [t, t+1).



compute the optimal use function in [t+1, t+T+1)
using the actual batter state at time t+1

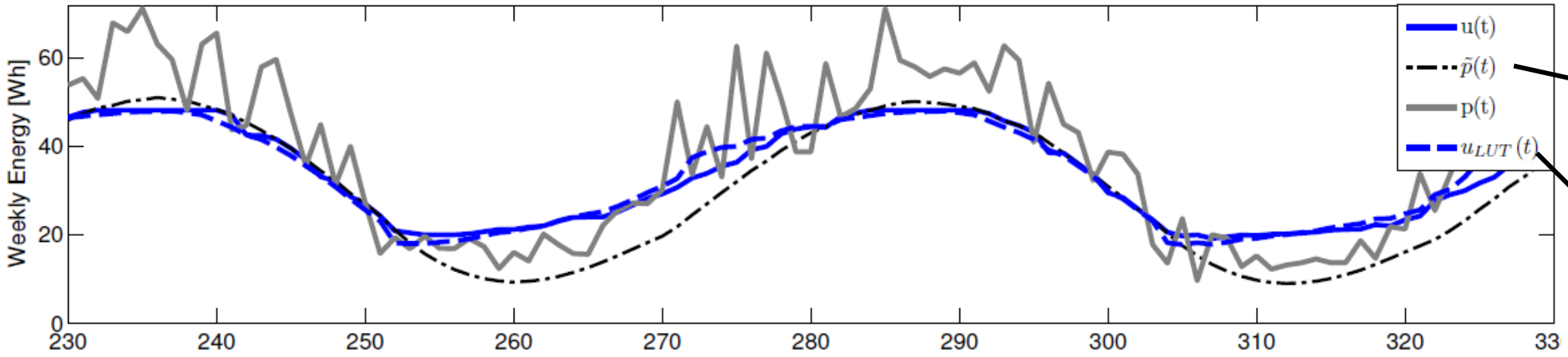
Application Control using Finite Horizon



estimated input energy

still energy breakdown due to misprediction

Application Control using Finite Horizon



more pessimistic prediction

simplified optimization using a look-up-table

[not covered]