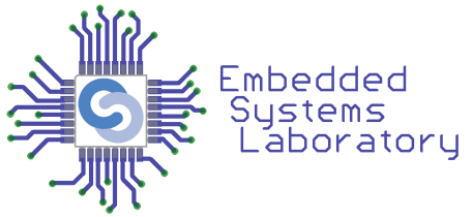
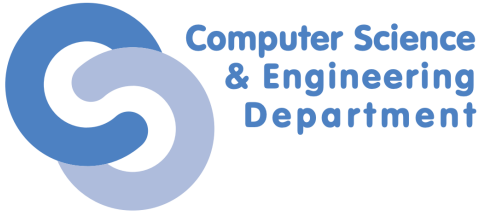




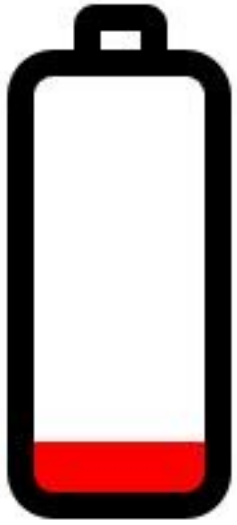
Sisteme Încorporate

Cursul 5

Software Power Management

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București

When I don't really need my phone :)



When I really need my phone :(

Software Low-Power

- Software-ul folosește resursele hardware
  - O parte semnificativă a funcționalității unui sistem o constituie programul executat
  - Impactul unui program asupra consumului de energie al unui sistem este semnificativ
  - Software inteligent proiectat -> reducerea consumului
- Fiecare instrucțiune are un cost în energie

Saving Power Consumption

Algorithms $\Longrightarrow$ Minimize Operating Time

HLL Source Code $\Longrightarrow$ Code Optimization

Compiler $\Longrightarrow$ Energy Miser

Operating System $\Longrightarrow$ Scheduling

Instruction Set Architecture $\Longrightarrow$ Energy Exposed

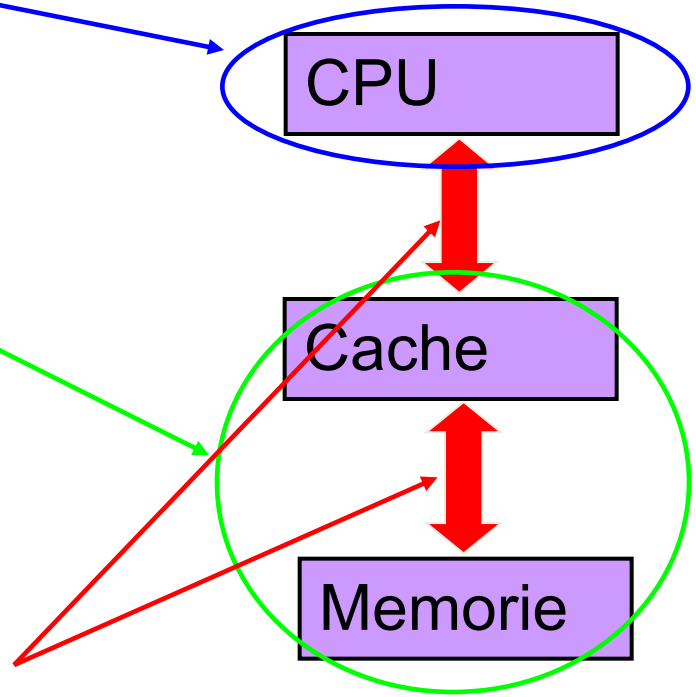
Microarchitecture $\Longrightarrow$ Clock Gating

Circuit Design $\Longrightarrow$ Low Voltage Swings

Manufacturing $\Longrightarrow$ Low-k Dielectric

Strategii low-power

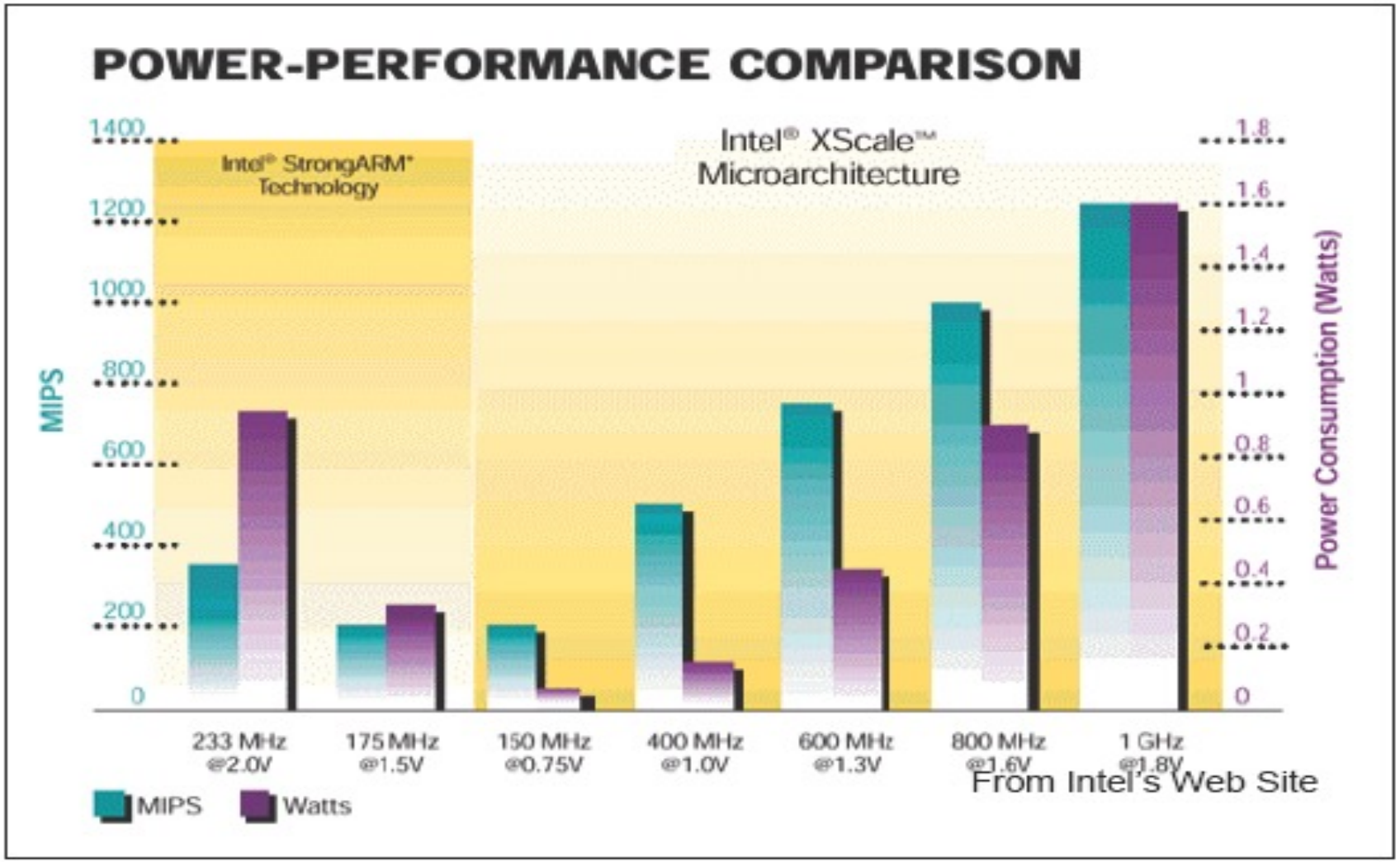
- Cod care rulează pe CPU
  - Optimizări în cod pentru low power
- Cod care accesează memoria
  - Optimizarea accesului la memorie
- Traficul de date pe magistrale
  - Codificare low-power
- Power Management controlat de compilator



Modalități de reglare

- Hibernare
  - Tranziția dintr-o stare de consum ridicat într-una de consum redus (low power)
- Scalarea dinamică a frecvenței și a tensiunii
  - Se stabilesc mai multe valori fixe pentru f și V
  - Instrucțiuni speciale prin care se stabilesc f și V
    - Prezente la toate arhitecturile majore (ARM, RISC-V, x86)
- Procesorul ține cont de temperatura de lucru
  - Thermally aware processor

Performanța = f(V, Hz)



Thermally aware processor

- Diferență mare între puterea nominală și puterea maximă a unui procesor
- Proiectarea unui procesor care să funcționeze la putere maximă este dificilă (maxim 60W disipați)
- Unitatea de detecție monitorizează temperatura și generează o întrerupere atunci când aceasta a depășit un nivel de prag
- Rutina de tratare a întreruperii duce procesorul în modul low-power
- Trade-off: are impact asupra puterii de procesare

Optimizarea codului orientată low-power

- Instrucțiunile high-level pot fi compilate în mai multe secvențe echivalente de instrucțiuni simple (C -> ASM)
    - Instrucțiuni diferite -> costuri de energie diferite
    - Ordinea execuției unui set de instructiuni afectează consumul
- Selectarea instrucțiunilor
  - Trebuie făcut un “amestec” de instrucțiuni care dau cel mai mic consum de energie
- Memorii duale
  - Două bancuri de memorie pe același chip
    - Load dual vs. load simplu
    - Reducerea consumului cu aproape 50%

Exemple de optimizare software

Înlocuiți condițiile compuse cu expresii aritmetice, folosind codarea poziției biților în loc de incrementarea numerelor întregi

- OK:

```
for (i = 0 ; i <=31 ; i++) {
  if ((i ==0) || (i==5) || (i==11) || (i==14) || (i==20)) { …
```

- Better:

```
for (i = 1 ; i != 0 ; i = i << 1) {
  if ((i & (1<<0|1<<5|1<<11|1<<14|1<<20)) != 0) { …
```

Micșorează dimensiunea variabilelor

- OK:

```
unsigned long i ;
for (i = 0 ; i < 100 ; i++) { …
```

- Better:

```
unsigned char i ;
for (i = 0 ; i < 100 ; i++) { …
```

Exemple de optimizare software

Transformați constantele literale utilizate frecvent în constante globale

- OK: `#define ERROR_STRING “Input error – try again”`
  `… display_message(ERROR_STRING) ;`

- Better:
```
const char * Error_string = “Input error – try again” ;
…
display_message(Error_string ) ;
```

Înlocuiți împărțirea cu înmulțire

- OK: `if ((f / g) < h) { …`
- Better: `if (f < (h * g)) { …`

Exemple de optimizare software

Embedded Systems Laboratory

Folosiți expresii reciproce în loc de împărțire

- **OK:** `if ((1/y) < 0.5) { …`
- **Better:** `if (y < 2.0) { …`

Folosiți înmulțirea în locul radicalului

- **OK:** `if (y < sqrt(z)) { …`
- **Better:** `if (y * y) < z) { …`

Pentru timpi de execuție mai buni, sortați expresiile condiționale compuse în ordine

- **Pentru condițiile AND booleene, plasați mai întâi termenii cu cea mai mare probabilitate de a fi evaluați la 0 logic**
- **Pentru condițiile OR booleene, plasați mai întâi termenii cu cea mai mare probabilitate de a fi evaluați la 1 logic.**
- **Pentru termenii care au probabilități aproximativ egale, plasați mai întâi termenii mai simpli.**

Exemple de optimizare software

Pentru accesări de matrice (în special matrice multidimensionale), utilizați pointeri în loc de referirea la membrii matricei

– OK:
```
for (j = 0 ; j < ASIZE ; j++) {
        for (k = 0 ; k < BSIZE ; k++) {
                a[j][k] = 0 ;       …
```

– Better:
```
p = &a[0][0] ;
for (j = 0 ; j < ASIZE * BSIZE ; j++) {
        *p++ = 0.0 ;       …
```

Pentru unele compilatoare, diferența de dimensiune a codului compilat și de viteză de execuție poate fi semnificativă!

Exemple de optimizare software

Pentru inițializarea structurilor, declarați o instanță constantă și utilizați memcpy() pentru a inițializa intrările în loc să utilizați asignări individuale. Utilizați memset() pentru a inițializa întreaga structură la 0.

- OK:

```
if ((p = (struct mystruct *) malloc(sizeof(struct mystruct)) != NULL) {
p->field1 = 1 ;
p->field2 = 2 ;        …
```

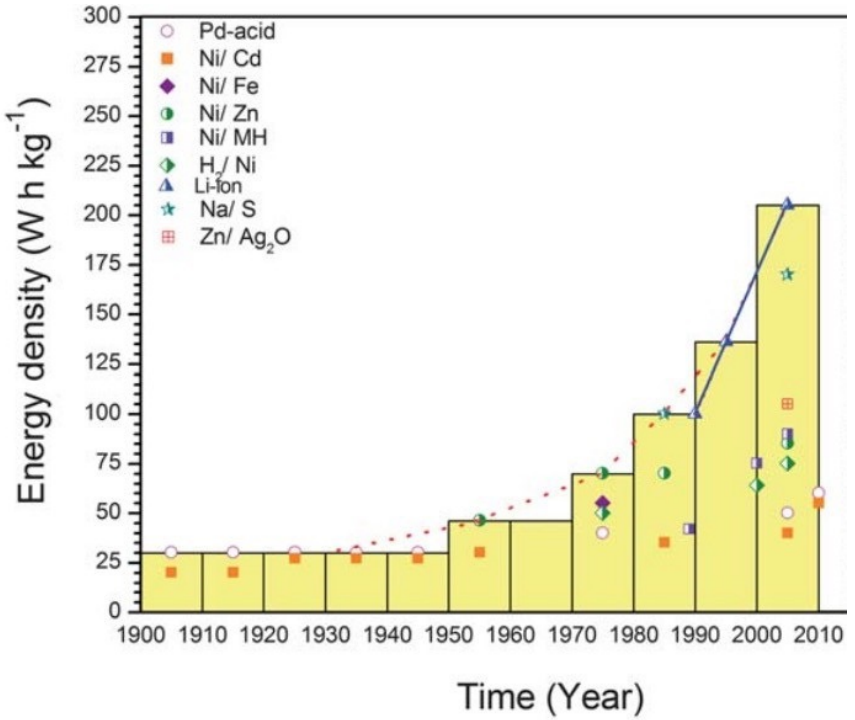
- Better:

```
if ((p = (struct mystruct *) malloc(sizeof(struct mystruct)) != NULL) {
memcpy(p, &const_mystruct, sizeof(struct mystruct)) ;
```

Power Aware Software

Embedded Systems Laboratory

- Cod structurat a.î. să minimizeze consumul de putere
- Cel mai des întâlnit pe dispozitive alimentate din baterii
- Devine din ce în ce mai relevant pentru toate dispozitivele
- În multe sisteme embedded consumul de putere este factorul critic
- Nu există legea lui Moore pentru baterii
  - Putem spune că avem o dublare a capacității la aproximativ 10 ani



Model pentru Power Aware Software

Embedded Systems Laboratory

Informații necesare:

- Puterea (sau curentul) pentru fiecare tip de instrucțiune
- Numărul de cicluri necesare pentru fiecare instrucțiune
- Overhead-ul executării următoarei instrucțiuni (probabilitatea de blocare)

Puterea consumată de tipul de instrucțiune executată nu este deseori specificată în datasheet, dar poate fi măsurată empiric.

- Bloc de instrucțiuni identice configurate pentru execuție din cache.
- Alte intrări, cum ar fi valorile datelor, ar trebui randomizate pe parcursul mai multor încercări.

Un simulator de set de instrucțiuni (ISS) poate fi folosit pentru a prezice puterea.

Unele, dar nu toate, instrumentele de modelare țin cont de efecte potențial importante, dar complexe, cum ar fi comportamentul cache, blocajele pipeline, valorile datelor, acuratețea branch prediction etc.

Optimizări low-level pentru consum

- Dacă sunt disponibile informații despre consumul de energie per instrucțiune, atunci preferați instrucțiunile cu consum redus, acolo unde este posibil.
- Desfășurați buclele mici (loop unrolling) pentru a reduce suprasarcina.
- Utilizați cele mai mici tipuri de date posibile.
- Compactați datele.
- Organizați datele și instrucțiunile pentru a optimiza performanța cache.
  - Evitați să mergeți la memoria externă oriunde este posibil.
  - Poate fi practic să reglați anumite aplicații embedded pentru a obține o performanță cache aproape ideală.
    - De exemplu, algoritmul optim de cache al lui Belady, unde liniile evacuate din cache sunt selectate pe baza cunoștințelor perfecte despre care intrare nu va fi necesară pentru cea mai lungă perioadă de timp în viitor.

Tehnici de programare power-aware la nivel de sistem de calcul

- Valorificați toate controalele disponibile care influențează consumul de energie, dar care îndeplinesc în continuare criteriile de performanță ale sistemului.
- Reduceți tensiunea sau frecvențele de ceas ori de câte ori sarcina de lucru este ușoară.
- Puneți domeniile de putere care nu sunt utilizate în mod curent în modul de sleep.
  - Multe dispozitive utilizate în sistemele încorporate conțin mai multe domenii de putere controlabile separat.
  - Domenii de putere analogice separate de cele digitale la Cypress PSoC.
  - Procesoarele Atmel ATMega au periferice on-chip controlabile separat.
- Puneți întregul dispozitiv în modul de repaus atunci când este inactiv.
  - Dar cum vă veți trezi?
    - Timer cu numărătoare inversă
    - Un eveniment extern declanșează o întrerupere

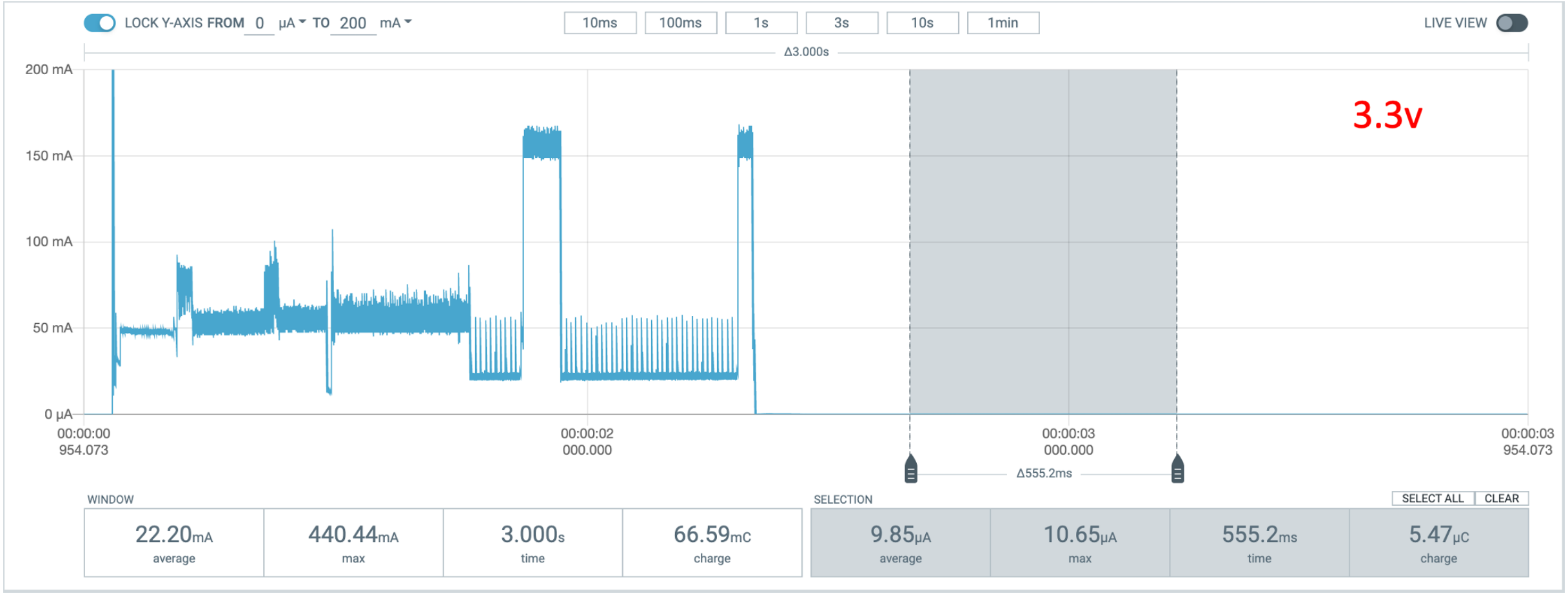
Cum poate software-ul să reducă consumul?

- Minimizați timpul de execuție prin selectarea unui algoritm rapid.
- Minimizați lățimea biților pentru toate datele.
- Minimizați lucrul cu memoria externă.
- Optimizați utilizarea cache-ului prin structurarea datelor și a codului.
- Pentru dispozitivele cu comutare acționată de curent (TTL), codificați datele pentru a maximiza apariția valorilor logice cu consum redus de energie (logic 1).
- Profitați de modurile de gestionare a energiei hardware accesibile software-ului.
- Puneți elementele hardware inactive în modul de repaus, ori de câte ori este posibil.
- Utilizați instrucțiuni cu consum redus de energie, ori de câte ori este posibil.
- Power aware task scheduling

Planificarea în timp real pentru programare power-aware

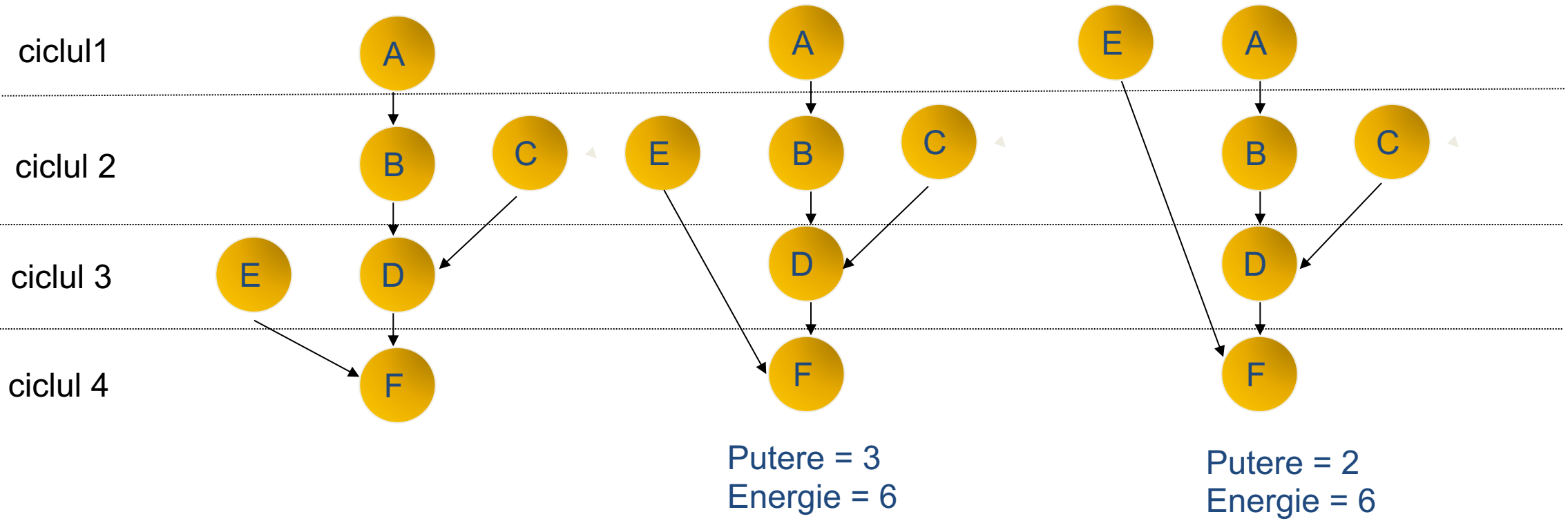
Embedded Systems Laboratory

Obiectivul este de a modula tensiunea, frecvența de ceas și/sau perioadele de repaus pentru a minimiza consumul de energie al sistemului, respectând în același timp criteriile de performanță în timp real.

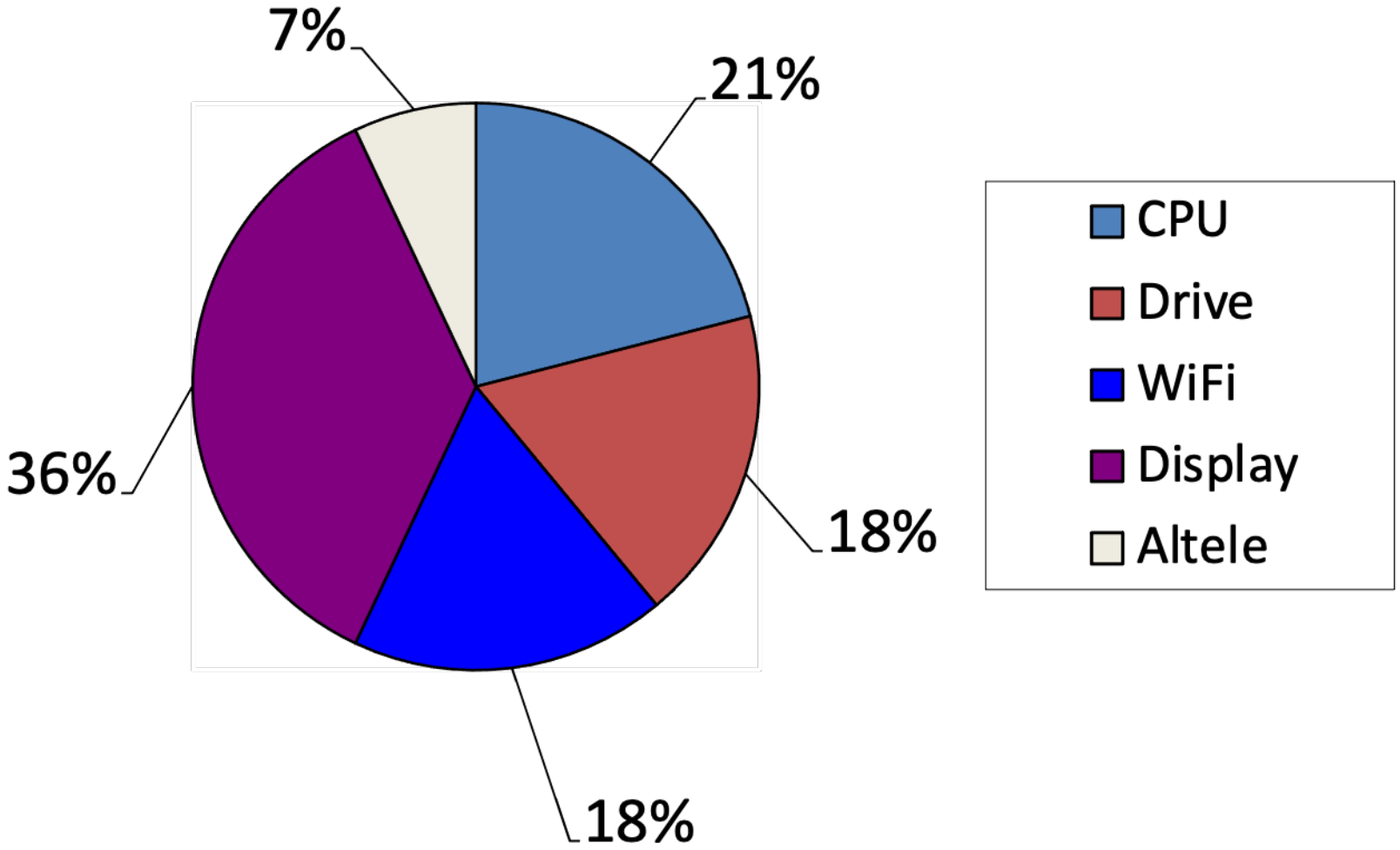


Optimizarea energiei = Optimizarea puterii consumate?

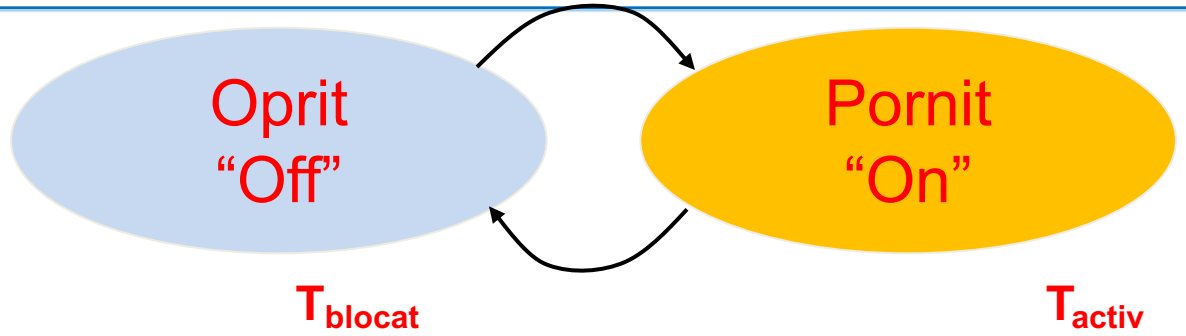
- Planificarea activității poate duce la reducerea puterii dar nu și a energiei



Unde se consumă energia?



Power management (PM)

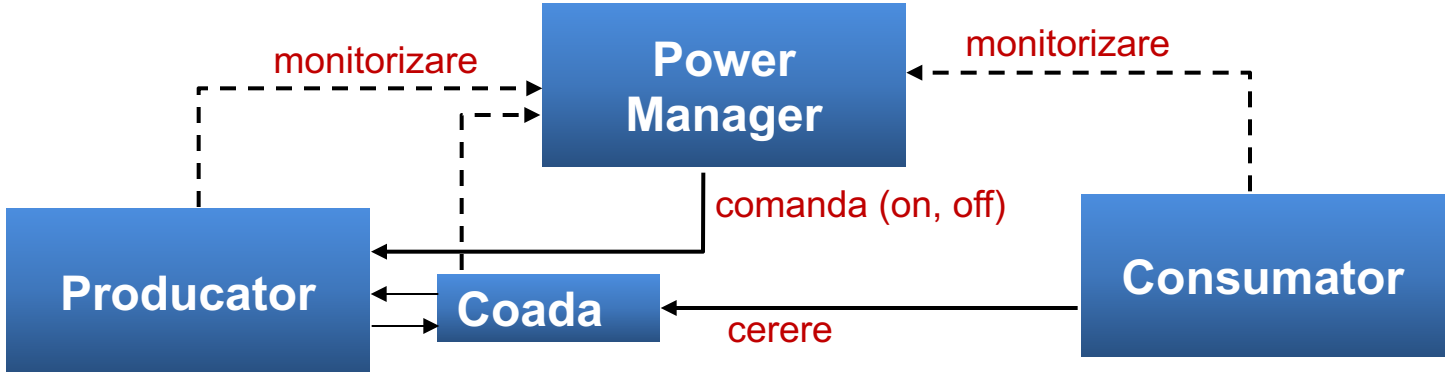


- Subsistemele au deseori cicli de funcționare limitați
  - CPU, hard-disk, interfața wireless sunt foarte des nefolosite
- Este o mare diferență între consumul în starea “on” și cel în starea “off”
  - Pentru Low-Power CPU:

    StrongARM 400mW (pornit)/ 50 mW (idle) / 0.16 mW (sleep)
  - Hard Disk :

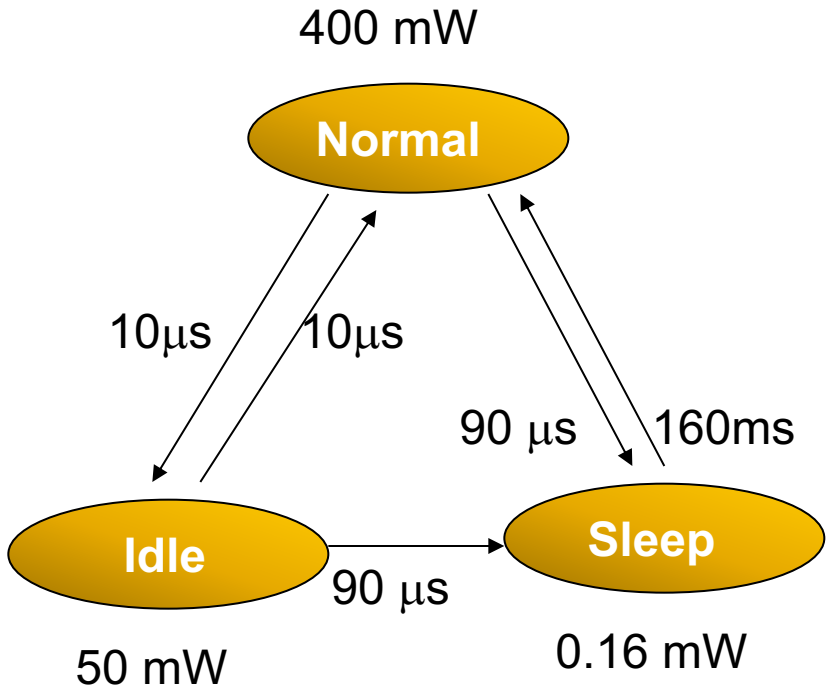
    1.35W (idle spinning) / 0.4W (standby) / 0.2W (sleep) / 4.7W (start-up)

Sistem generic cu power-management



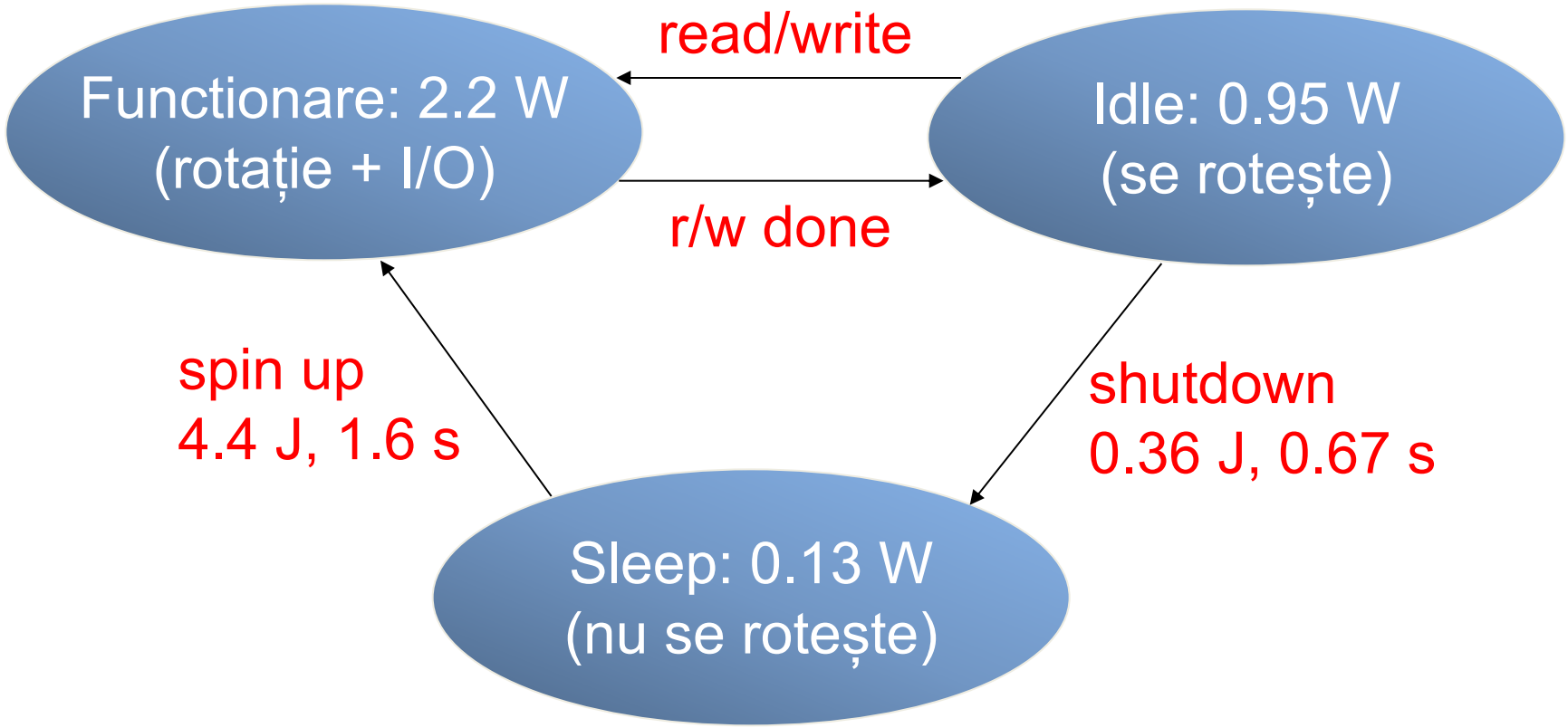
- Trebuie să existe o interfață între componentele care au nevoie de management (procesoare, drivere, periferice) și unitatea de power-management
  - Algoritm de power management
    - Când și cum se poate acționa
  - În esență, PM este un controller de sistem
- Reprezentare abstractă: automat de stări
  - Stări = politica de PM aplicată și consumul de energie
  - Muchii = timpi de tranziție și consum per tranziție

StrongARM SA-1100 – Stări de consum

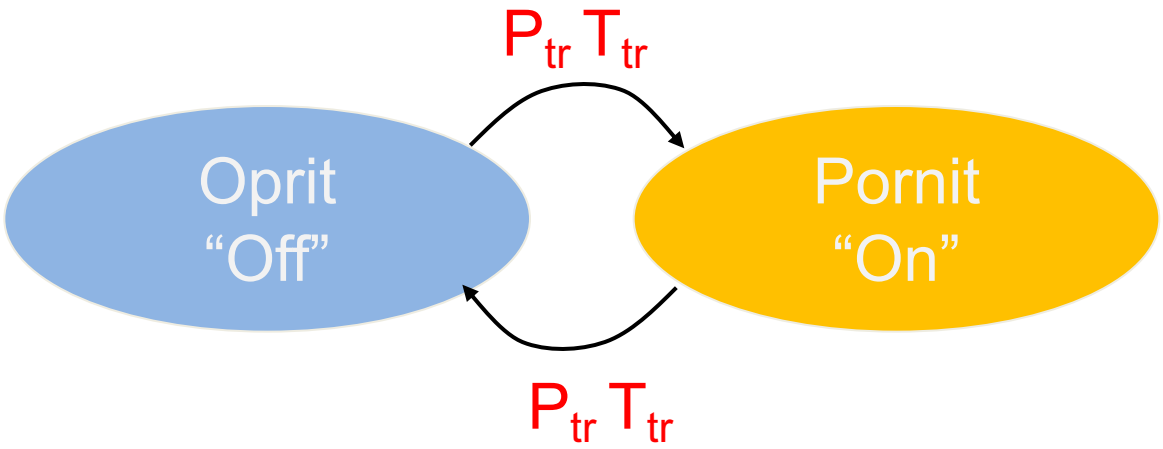


- Normal
  - CPU și perifericele sunt alimentate
  - Toate subansamblele primesc semnal de ceas
- Idle
  - Semnalul de ceas al CPU e oprit
  - Ceasurile către periferice sunt active
  - Se intoarce în modul normal printr-o întrerupere
- Sleep
  - DRAM în modul self-refresh
  - Toate funcțiile sunt dezactivate mai putin ceasul de timp real
  - “Trezire” printr-o întrerupere preprogramată sau de la utilizator

Fujitsu MHF 2043 AT (hard-disk)



Când e util PM?



- Dacă T_{tr}=0, P_{tr}=0 atunci politica de PM e evidentă
  - Oprește o componentă când nu ai nevoie de ea
- Dar, de obicei T_{tr}!= 0, P_{tr} != 0
  - Oprește doar atunci când T_{idle} este suficient de mare
  - Problemă: Cum pot să-mi dau seama cât o sa fie T_{idle} ?

Probleme date de Shutdown

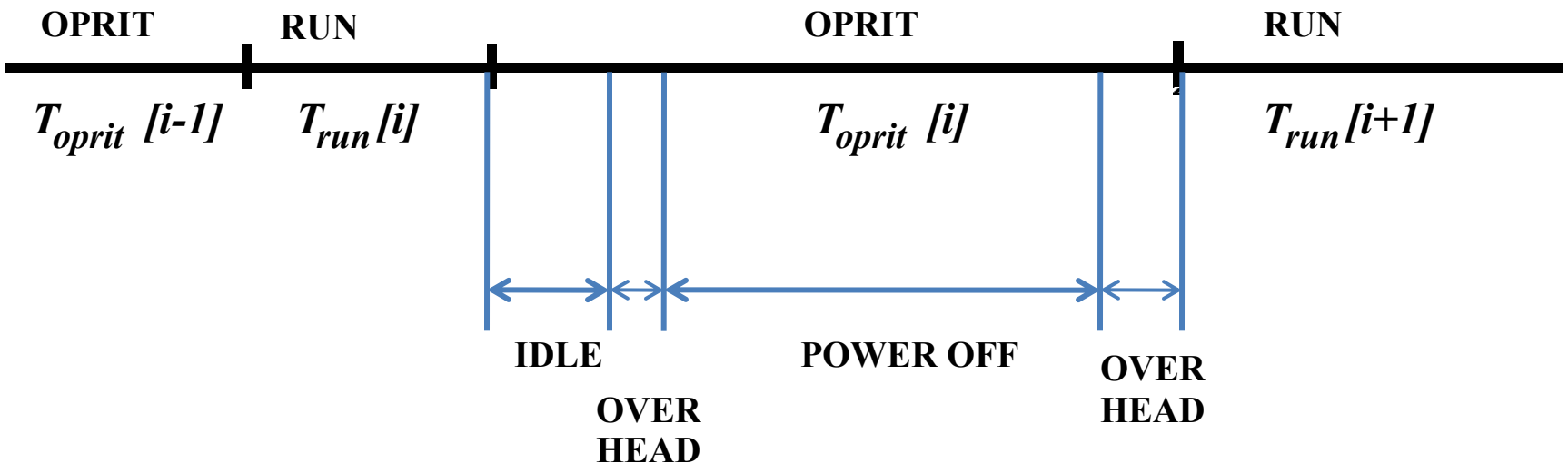
- Costul restartării: compromis între latență și consum
  - mărirea latenței (timpul de răspuns)
    - » Timpul de rotație pentru disc (spin-up)
  - mărirea consumului
    - » Curent mai mare la spin-up
- Când să se faca shutdown

  Optim vs. *Idle Time Threshold* vs. *Predictiv*
- Când să se faca wake-up

  Optim vs. *La cerere* vs. *Predictiv*

Abordare Convențională

"Oprește sistemul când utilizatorul nu l-a mai folosit un timp de x minute și restartează la cerere"



Abordarea bazată pe predicție

"Folosește informațiile strânse pentru a determina în ce situație $T_{oprit}[i]$ e suficient de mare

$(T_{oprit}[i] \geq T_{cost})$"

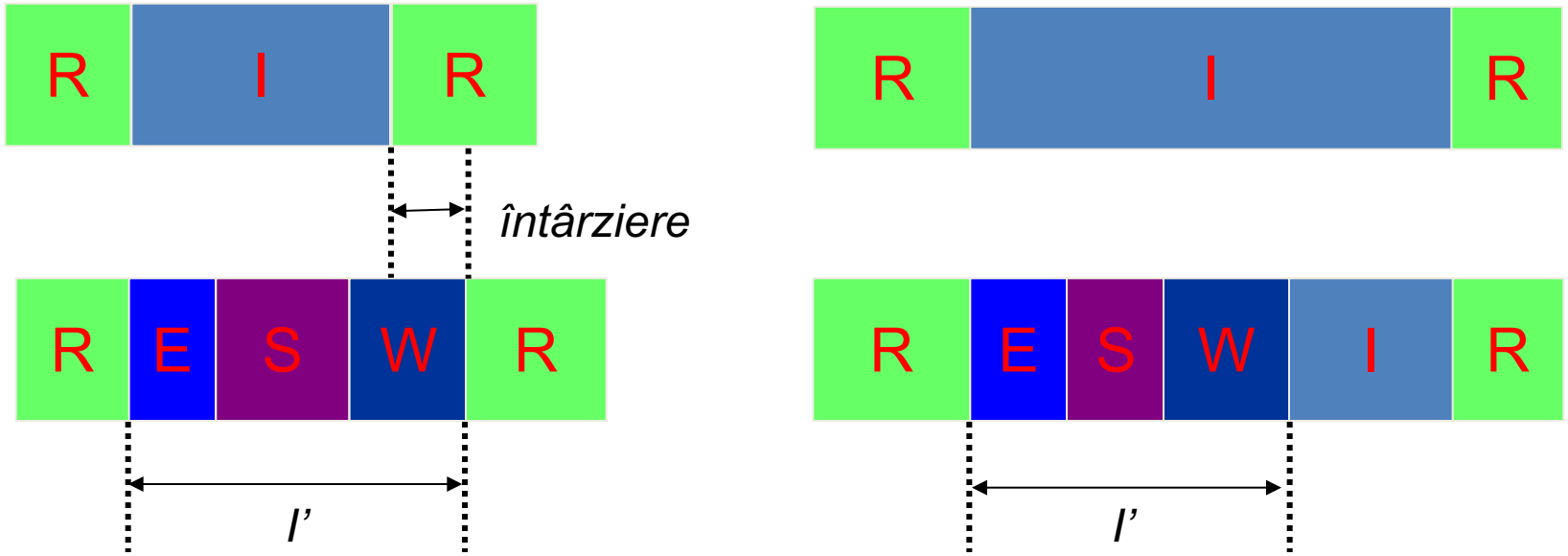
- Exemplu:

$$T_{oprit}[i] \geq T_{cost} \Leftrightarrow T_{run}[i] \leq T_{on\_threshold}$$

  - Poate duce la reducerea seminificativă (~20x) a consumului
  - Elimină timpii de așteptare în care sistemul este nefolosit

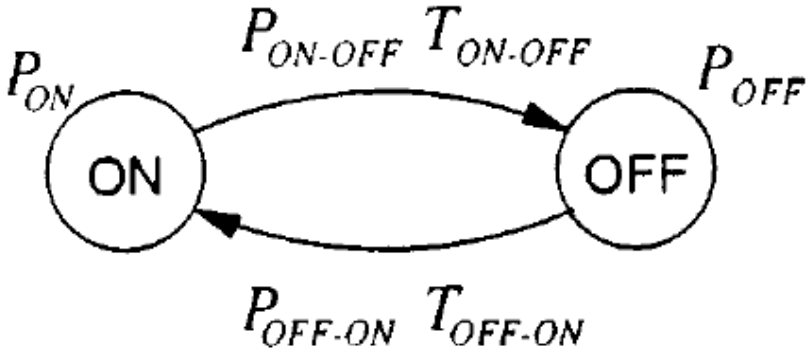
Pre-wakeup

- Inițializarea sistemului durează timp, lucru care afectează performanțele
- Dacă timpii de lucru pot fi prevăzuți, sistemul poate fi pre-inițializat înainte de pornirea efectivă



Breakeven Point

- Breakeven point: timpul minim de idle pentru care e eficient să se facă shutdown
- PM eficient -> T_{BE} < Media T_{idle}



$$T_{TR} = T_{\mathrm{On,\,Off}} + T_{\mathrm{Off,\,On}}$$

$$P_{TR} = \frac{T_{\mathrm{On,\,Off}} P_{\mathrm{On,\,Off}} + T_{\mathrm{Off,\,On}} P_{\mathrm{Off,\,On}}}{T_{TR}}$$

$$T_{BE} = T_{TR} + T_{TR} \frac{P_{TR} - P_{\mathrm{On}}}{P_{\mathrm{On}} - P_{\mathrm{Off}}} \text{ if } P_{TR} > P_{\mathrm{On}}$$

$$T_{BE} = T_{TR} \text{ if } P_{TR} \leq P_{\mathrm{On}}.$$

Implementarea PM

- Clock gating
- Oprirea sursei de alimentare
- Oprirea ecranului
- Oprirea motoarelor (și în general a tuturor efectoarelor mecanice)

Puterea în CMOS

$$P = \frac{1}{2} ACV^2 f + \tau AVI_{short} f + VI_{leak}$$

P = putere totală

V = tensiunea de alimentare

f = frecvența de ceas

C = capacitatea liniilor de ieșire

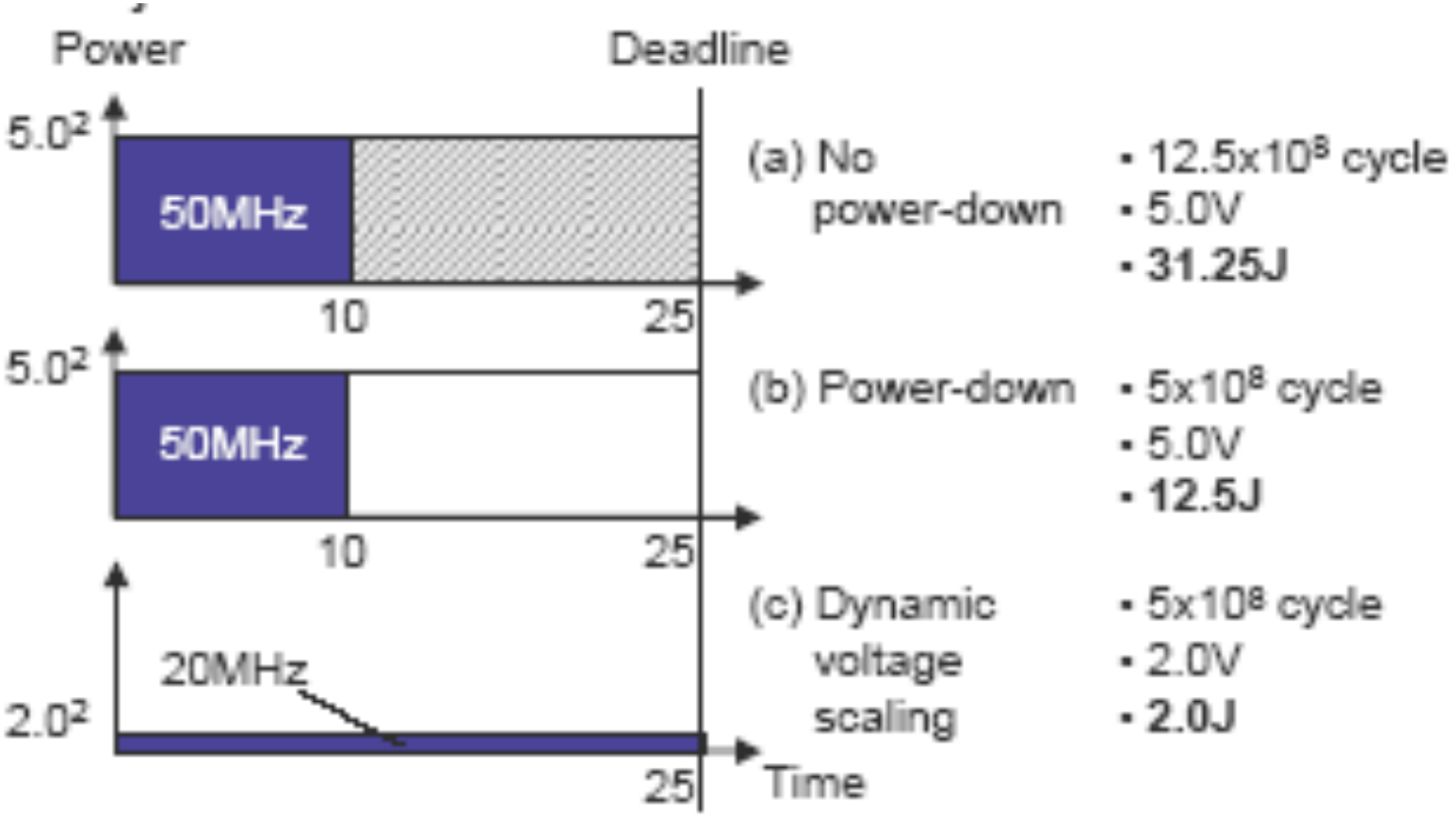
A = activitate (tranziții logice pe ciclu de ceas)

I_{leak} = curent de mers în gol

I_{short} = curent de scurt-circuit

τ = durata curentului de scurt-circuit

Tehnici

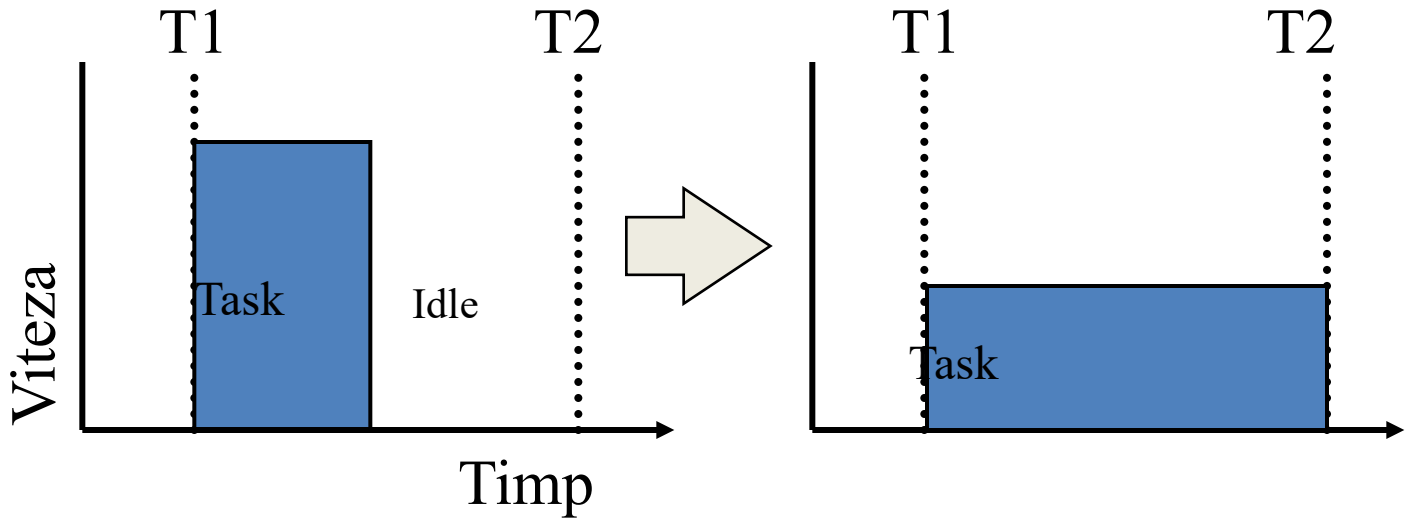

Power
Deadline
5.0²
50MHz
10
25
(a) No power-down
• 12.5x10⁸ cycle
• 5.0V
• 31.25J
5.0²
50MHz
10
25
(b) Power-down
• 5x10⁸ cycle
• 5.0V
• 12.5J
20MHz
2.0²
25
Time
(c) Dynamic voltage scaling
• 5x10⁸ cycle
• 2.0V
• 2.0J

Dynamic voltage/frequency scaling

- Reducerea consumului vs. întârzieri
- Sarcini non-real time
  - Fără constrângeri de timp
  - Nu se știe când și câte date vor fi procesate într-un interval
  - Nu se știe cât timp va dura procesarea datelor
- Predicția încărcării viitoare a sistemului
  - Predictie: Cât de încărcat va fi procesorul în viitor
  - Smoothing: “Nivelarea” timpilor de execuție

Reducerea tensiunii de alimentare

- Exemplu: sarcina care durează 100ms și 50ms cu CPU la viteză maximă
  - Sistem normal: 50ms calcule, 50ms idle time
  - Jumătate din viteza/tensiune: 100ms calcule, 0ms idle
  - Același număr de instrucțiuni DAR reducere la 1/4 a consumului

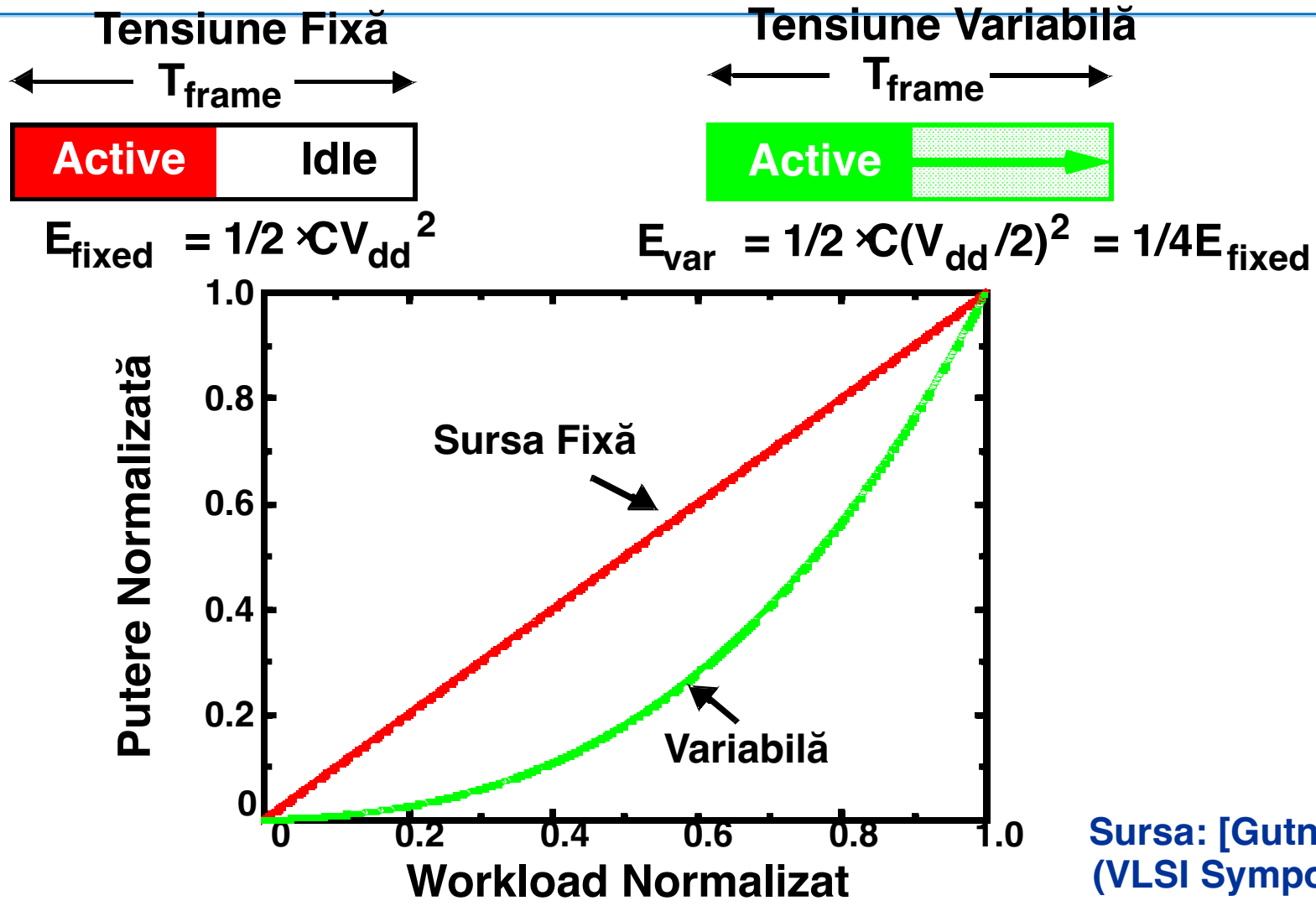


Reducerea tensiunii: probleme

- Abordare statică: Tensiunea minimă de alimentare este dictată de timpul maxim de execuție al unui task
  - Nu e o problemă dacă timpul de execuție nu e important
    - » Productivitatea poate fi îmbunătățită prin paralelizare, pipelining (vezi curs anterior)
  - Sistemele reale au perioade intense de calcul și timpi de mers în gol care nu pot fi prevăzuți

Soluția: variația dinamică a tensiunii.

Variația Tensiunii de Alimentare

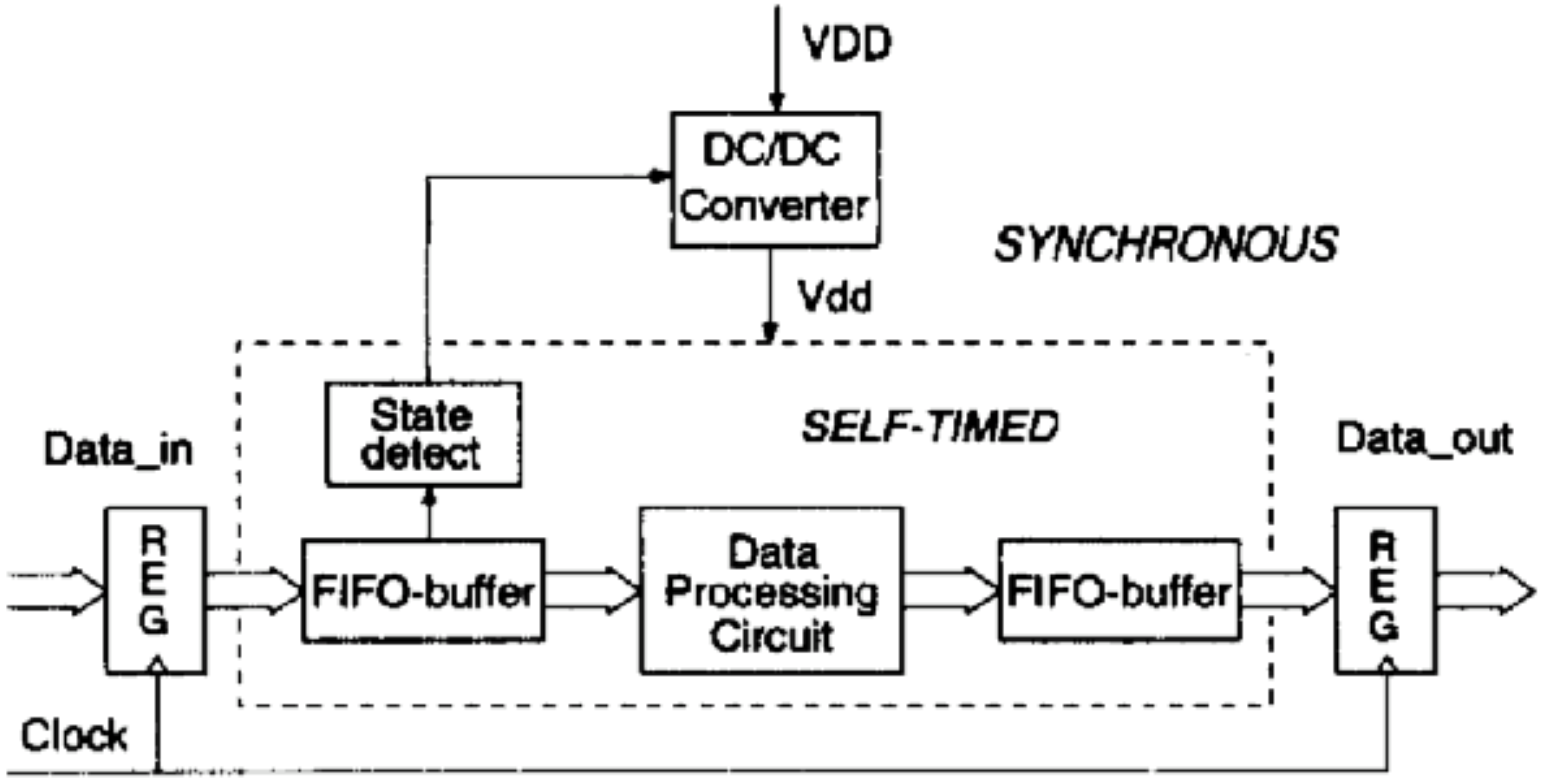


Sursa: [Gutnik96]
(VLSI Symposium)

Variația Dinamică a Tensiunii de Alimentare (DVS)

- Puterea și frecvența de ceas depind de tensiunea de alimentare
- Tehnologia curentă
  - Surse DC-DC eficiente (> 80%)
  - Majoritatea chipurilor CMOS funcționează pe o plajă largă de tensiuni (0.8 – 5V)
- Problema ține de algoritmi și programare:
  - Cum știu când trebuie să variez tensiunea?
    - » La compilare, în hardware sau las SO să facă treaba?

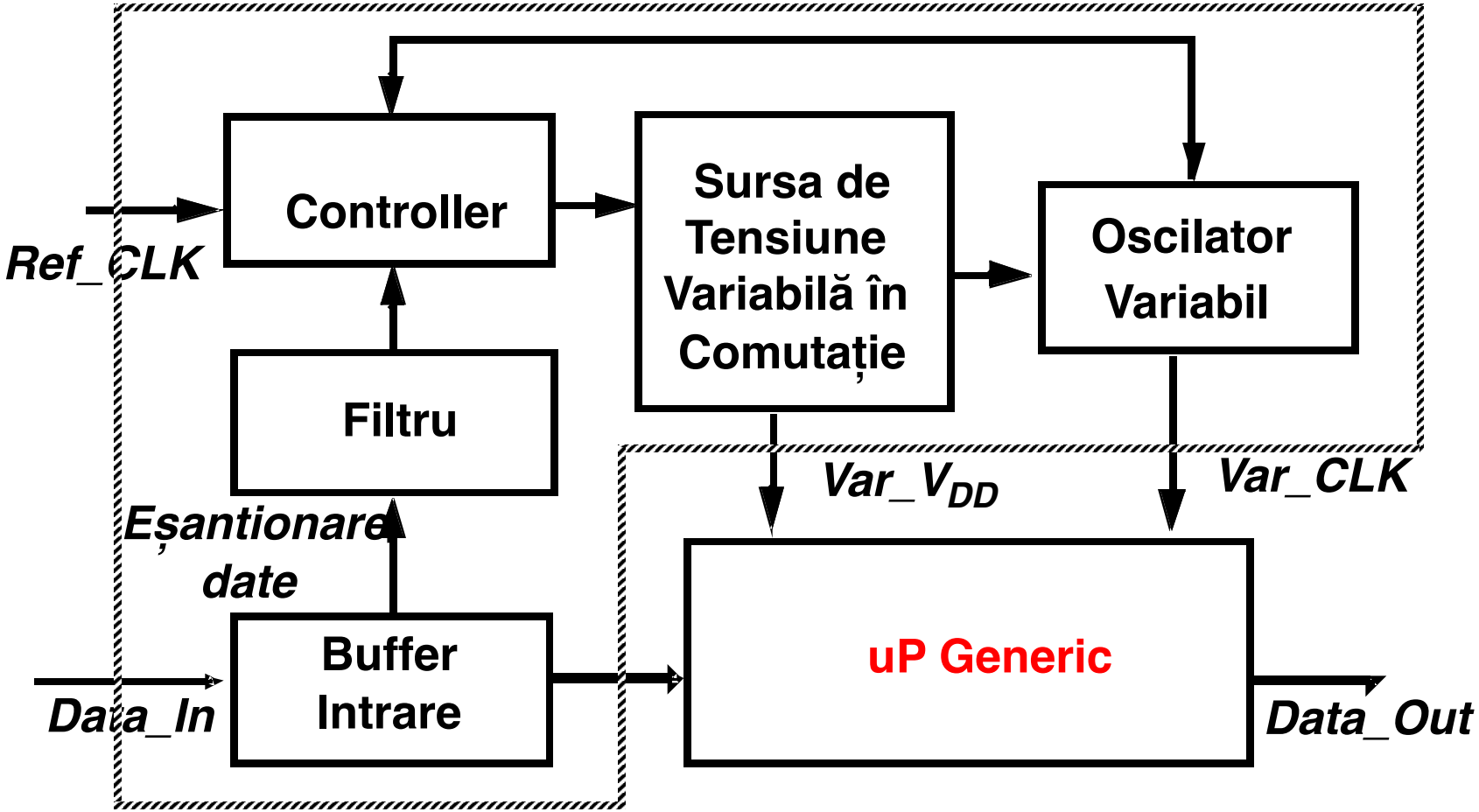
Scalare dinamică pentru sisteme asincrone



- Măsoară fluxul de date la intrare pentru a estima cantitatea de procesare

[Nielsen94]

Scalare dinamică pentru sisteme sincrone



Gutnik & Chandrakasan (1996 VLSI Circuits Symposium)

DVS într-un SO

- Abordarea 1
  - Timpul împărțit în cuante de 10-50 ms
  - f și V sunt modificate într-un interval în funcție de activitatea procesorului în intervalul anterior
    - » Reducere de 50% pentru un procesor la 3.3V-5V
    - » Reducere de 70% pentru un procesor la 2.2V-5V
- Abordarea 2
  - Predicția activității procesorului într-un interval următor
  - Setează f și V în consecință
  - Care strategie de predicție e mai bună?

PAST

- Măsoară timpii de execuție pentru intervalul de timp anterior
- Presupune că intervalul viitor va fi asemănător
- Dacă intervalul anterior a fost
  - Mai mult busy → mărește viteza (f,V)
  - Mai mult idle → micșorează viteza
- Algoritm simplu. Rezultate foarte bune (KISS)

Exemplu PAST

Interval	Run-Percent	Total Work	Predicted Run-percent	Excess Cycles
1	0.2	0.2	N.A.	0
2	0.4	0.4	0.2	0.2
3	0.4	0.6	0.4	0.2

PAST: Deficiențe

- În cazul unor procesări în rafală, cuantele adiacente diferă foarte mult din punctul de vedere al cantității de calcule
- Consideră doar intervalul anterior
- “Netezire” deficitară din cauza istoriei pe termen scurt

FLAT

- Încearcă să netezească viteza la o valoare medie globală
- Prezice că următorul procent de utilizare va fi f(const)

+ “Netezire” foarte bună

+ Numar mic de cicli în exces

- Predicție slabă

Exemplu FLAT [const = 0.3]

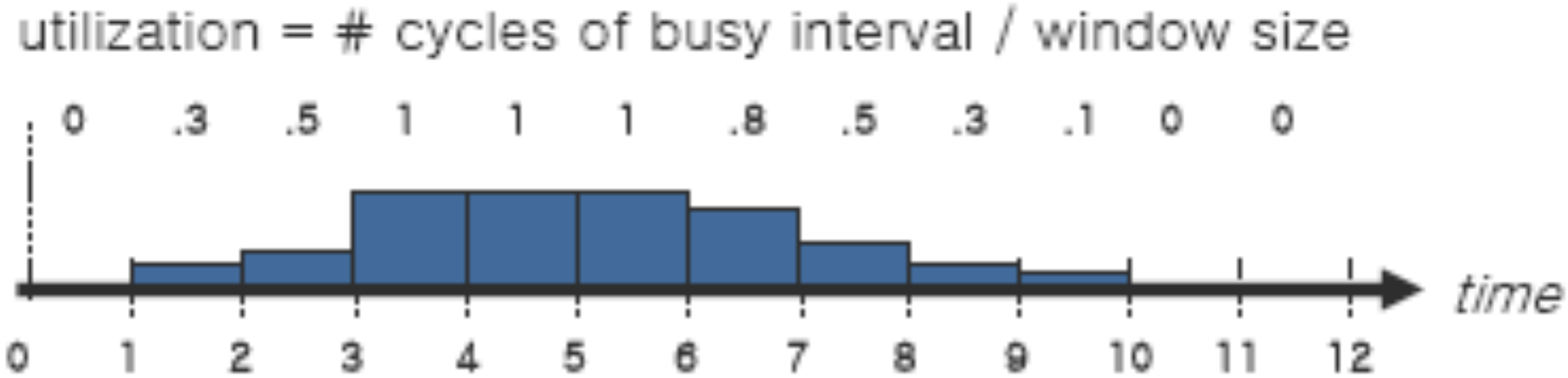
Embedded Systems Laboratory

Interval	Actual Run-percent	Total Work	Predicted Run-percent	Excess Cycles
1	0.2	0.2	0.3	0
2	0.4	0.4	0.3	0.1
3	0.4	0.5	0.3+0.1	0.1

LONG_SHORT

- Compromis între comportamentul pe timp lung și cel recent
- Istorie de 12 cuante de timp anterioare
  - Short-term: cele mai recente 3 cuante
  - Long-term: restul de 9 intervale
- Procesarea pentru intervalul curent este media ponderată a celor 12 intervale anterioare
- “Netezește” la o medie globală dar ia în considerare și maximele locale

Exemplu LONG_SHORT



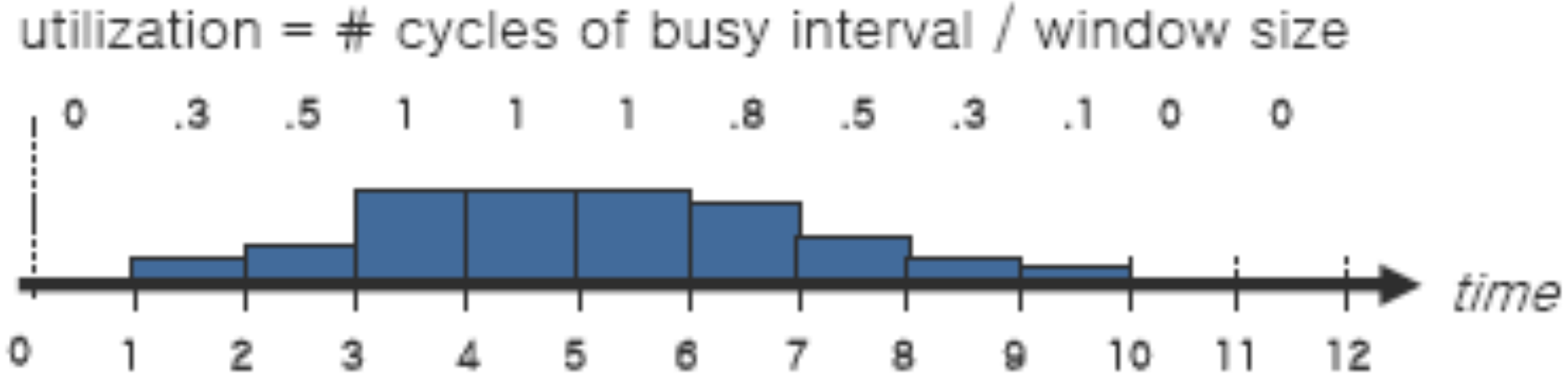
$$\frac{0+.3+.5+1+1+1+.8+.5+.3+4(.1+0+0)}{9+4(3)} = 0.276$$

$$f_{clk} = 0.276 \times f_{max}$$

AGED_AVERAGE

- Algoritm de netezire exponențială
- Procesarea pentru intervalul curent este media ponderată a tuturor intervalelor anterioare
  - Ponderile sunt reduse în progresie geometrică

Exemplu AGED_AVERAGE



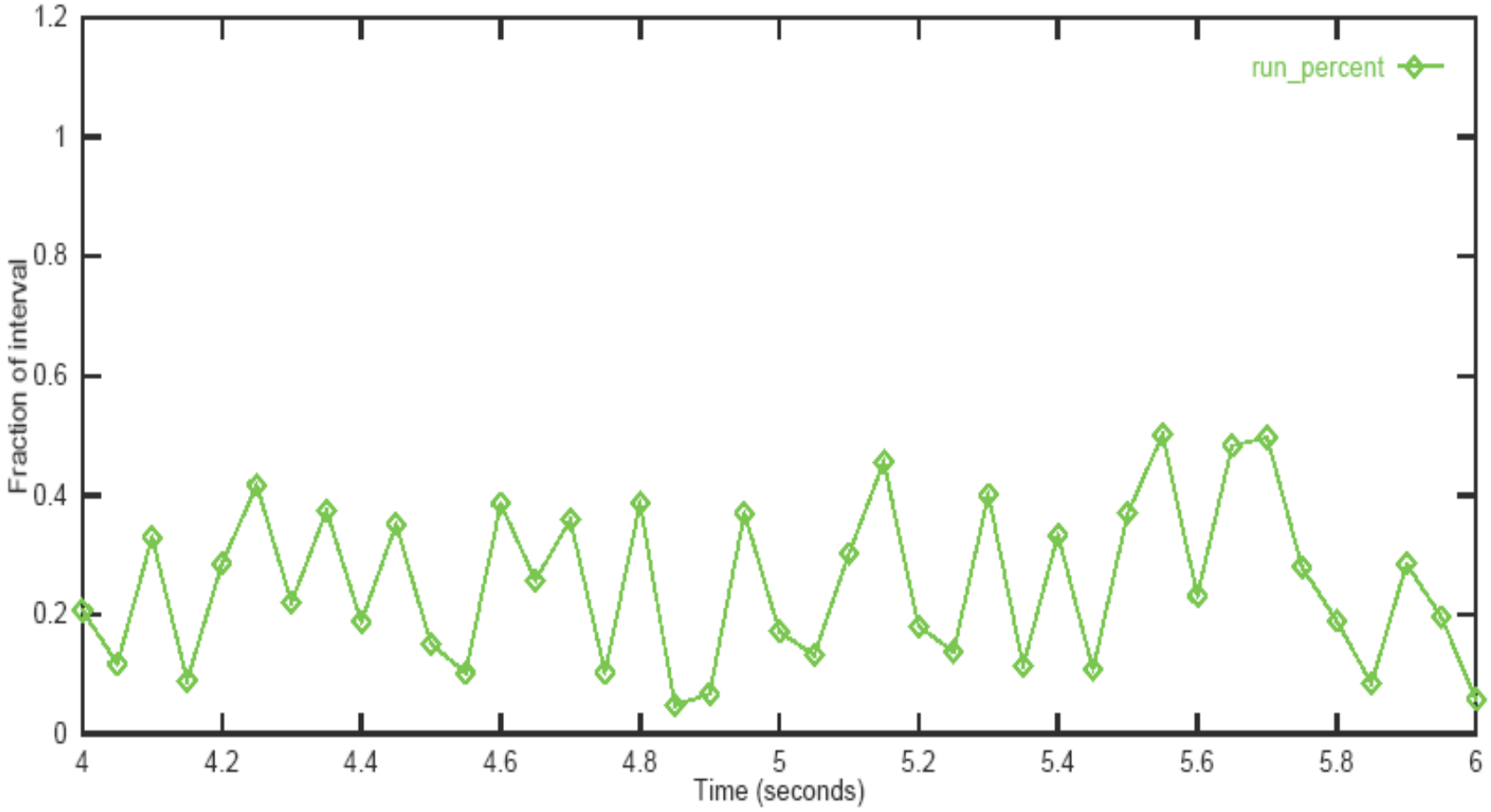
$$average = \frac{1}{3}0 + \frac{2}{9}0 + \frac{4}{27}(0.1) + \frac{8}{81}(0.3) + \cdots$$

$$f_{clk} = average \times f_{max}$$

CYCLE

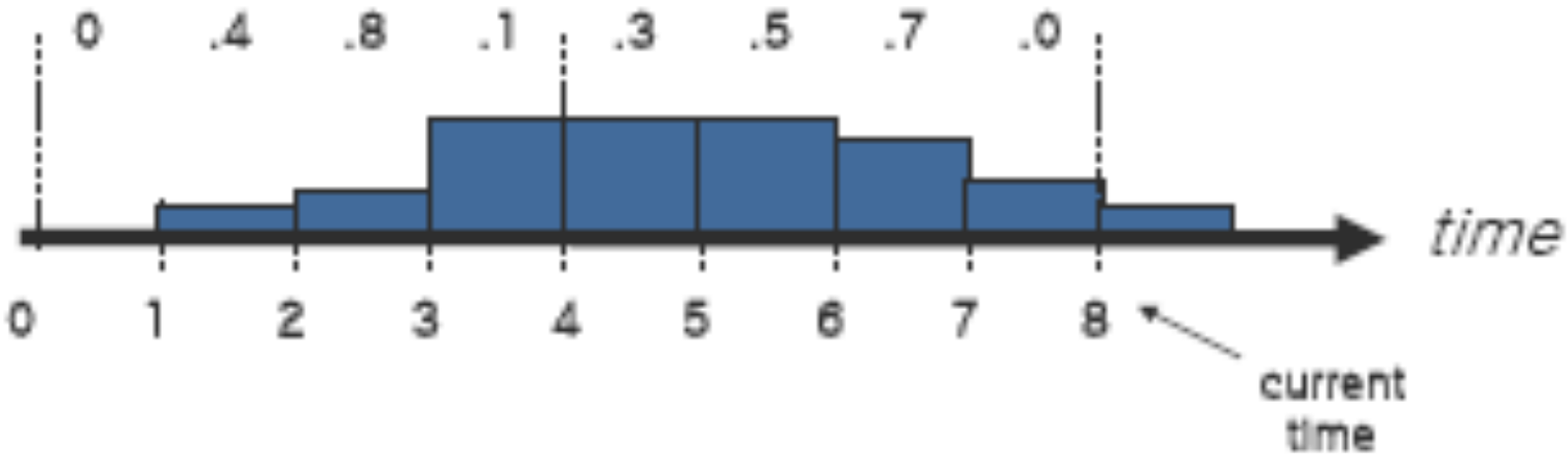
- Examinează ultimele 16 ferestre
- Există un ciclu de lungime X? (eroare < prag)
  - Dacă da, prezice extinderea ciclului
  - Dacă nu, folosește FLAT
- Un FLAT care “ghicește” mai bine

Motivația din spatele CYCLE



Exemplu CYCLE

utilization = # cycles of busy interval / window size



$$\text{error measur}: \frac{|0-.3|+|.4-.5|+|.8-.7|+|.1-0|}{4} = 0.15$$

Predict : The next utilization will be .3

PEAK

- Euristică
  - Procentele de creștere sunt de obicei simetrice cu cele în scădere
  - Procentele în scădere continuă să scadă (-> 0)
  - Perioadă susținută de run-percent = 1 va scade eventual
  - Perioadă susținută de run-percent = 0 se va menține stabilă și pe viitor

Rezultate Experimentale

- FLAT are performanțe mai bune decât PAST
- LONG_SHORT: low-energy, latență mare
- AGED_AVERAGES se comportă mai bine când este echivalent cu FLAT
- CYCLE are cele mai proaste performanțe când încearcă să prezică cât mai “inteligent”
- PEAK este unul dintre cei mai buni algoritmi euristici

Lecturi recomandate

- Comparing System-level Power Management Policies, Y H Lu and G De Micheli, IEEE Design and Test of Computers, March-April 2001
- Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU, K Govil et al., International Conference on Mobile Computing and Networking 1995