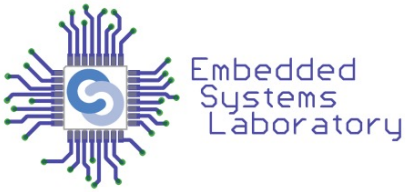
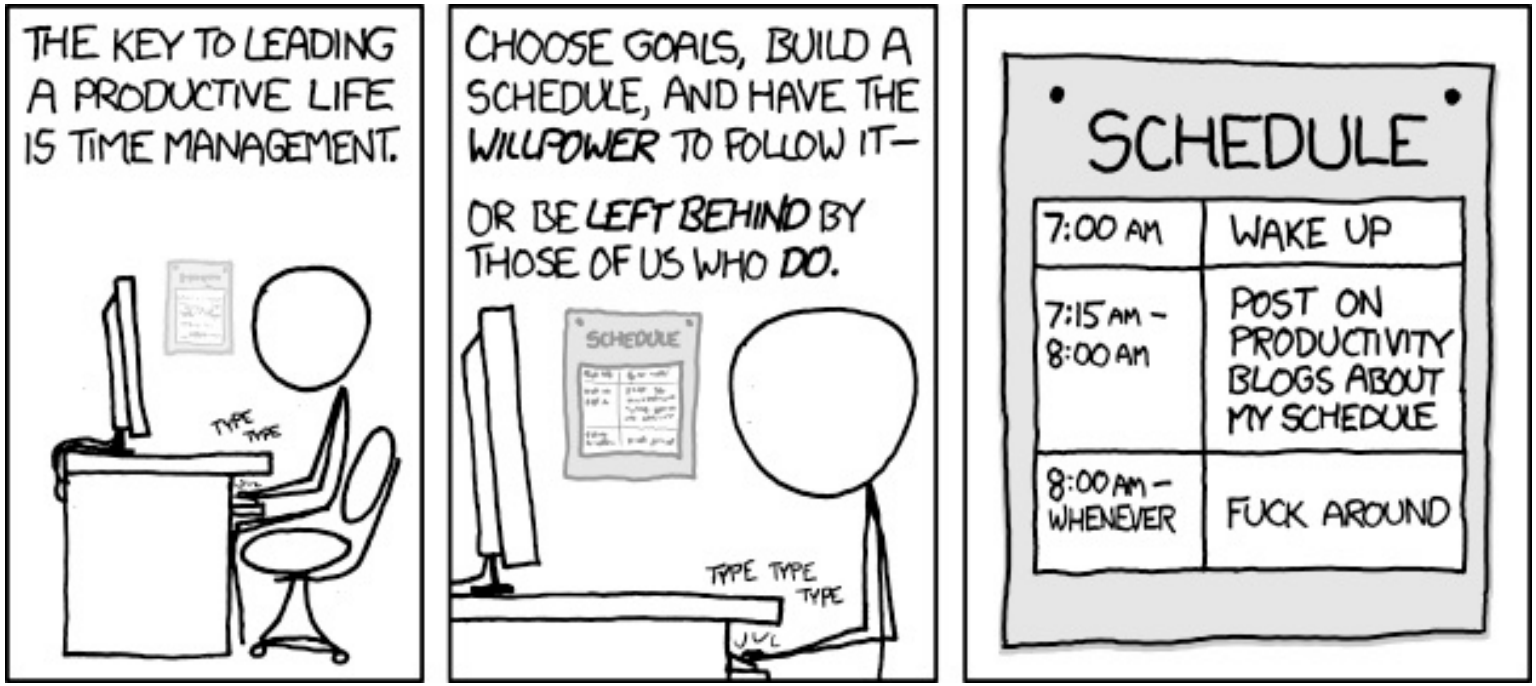


Sisteme Încorporate

Cursul 5

Software Power Management

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București



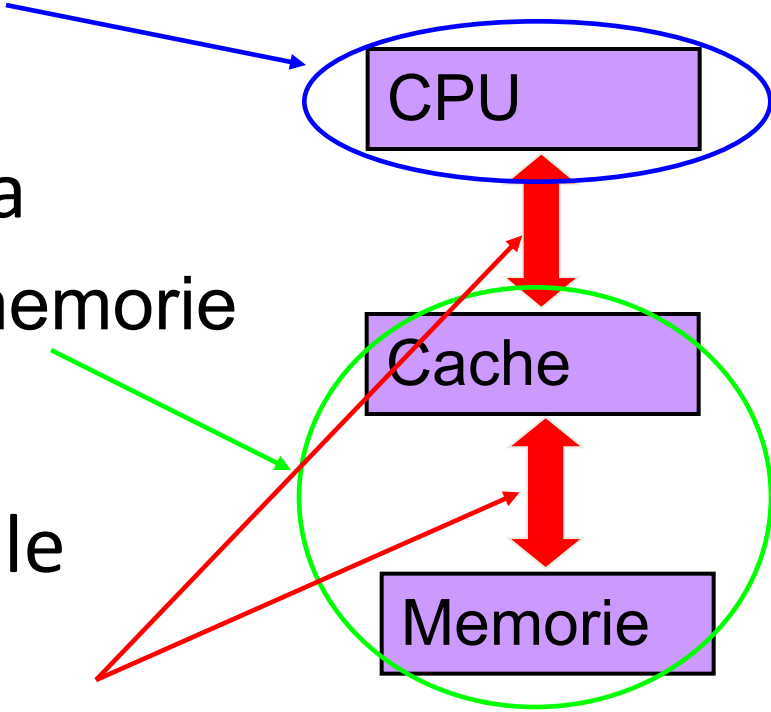
http://xkcd.com/874/

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

Exemplu: strategii low-power

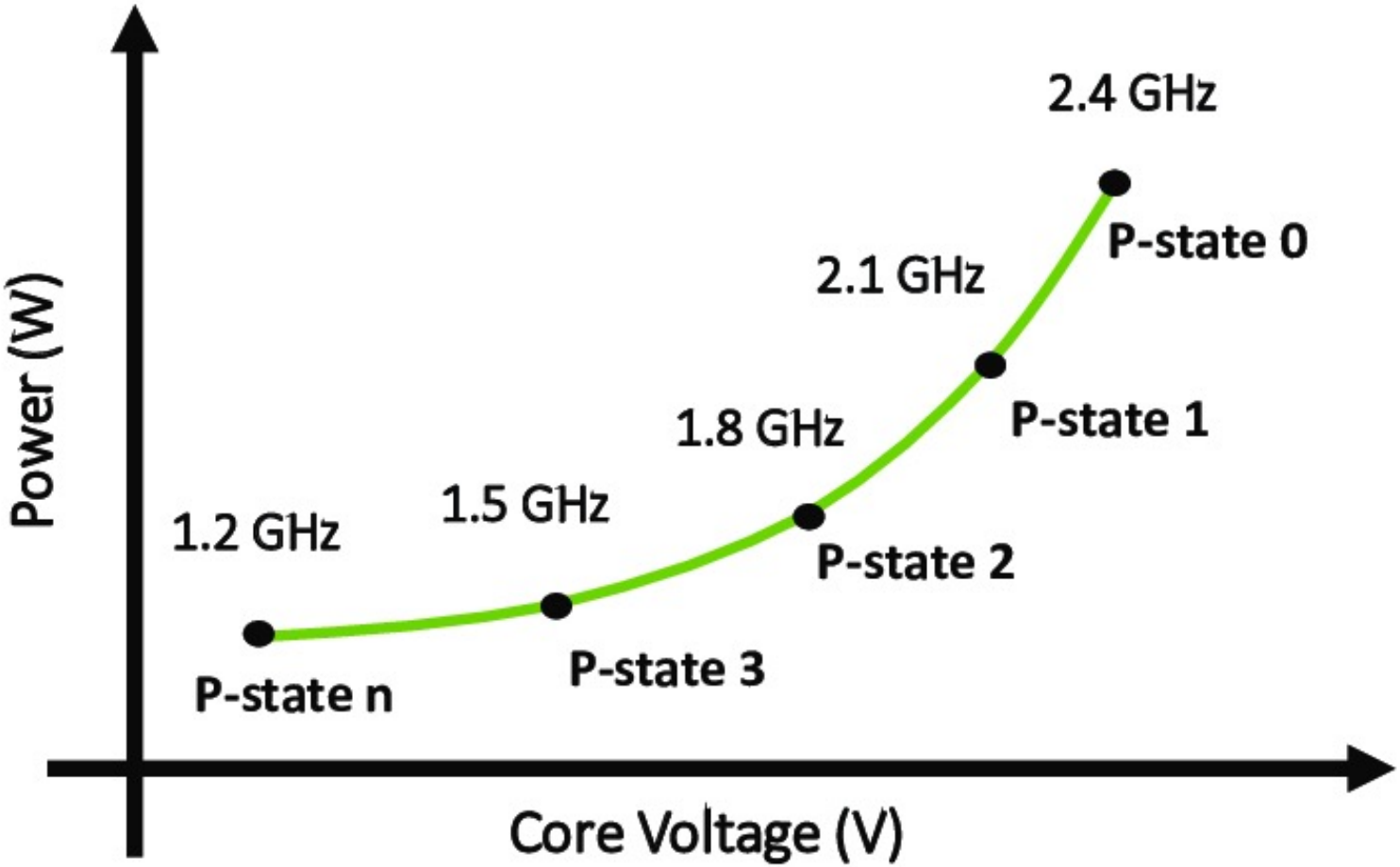
- Cod care rulează pe CPU
  - Optimizări în cod pentru low power
- Cod care accesează memoria
  - Optimizarea accesului la memorie
- Traficul de date pe magistrale
  - Codificare low-power
- Power Management controlat de compilator



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
    - Intel, ARM, AMD etc.
- Procesorul ține cont de temperatura de lucru
  - Thermally aware processor

Performanța = f(V, Hz)



Intel P-states

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%

Optimizarea codului orientată low-power

- Reordonarea instrucțiunilor în cod pentru a reduce efectul comutării unităților funcționale, magistralelor de memorie,I/O etc.
- Operand swapping
  - Rotirea operanzilor între ei la intrarea în multiplicator
  - Înmultirea dă același rezultat, dar consumul este redus semnificativ!
- Optimizări standard ale compilatorului
  - Software pipelining, dead code elimination, eliminarea redundanțelor
  - Alocarea registrelor
- Utilizarea unui stil de programare în funcție de procesorul folosit
  - e.g. pe ARM tipul de date `int` este cu ~ 20% mai eficient decat `char` sau `short` pentru că acestea folosesc extensia de semn
  - e.g. pe ARM pot fi folosite sufixele condiționale în locul instrucțiunilor de branch

Minimizarea costurilor de acces la memorie

- Reducerea numărului de accese la memorie -> folosește mai mult registrele generale
  - Costul unui acces de registru general << acces adresă de memorie
- Minimizarea numarului operatiilor de citire si scriere

```
for (i=0;i< n;i++)
    b[i] = f(a[i]);
for (i=0;i< n;i++)
    c[i] = g(b[i]);
```

```
for (i=0;i< n;i++) {
    b[i] = f(a[i]);
    c[i] = g(b[i]);
}
```

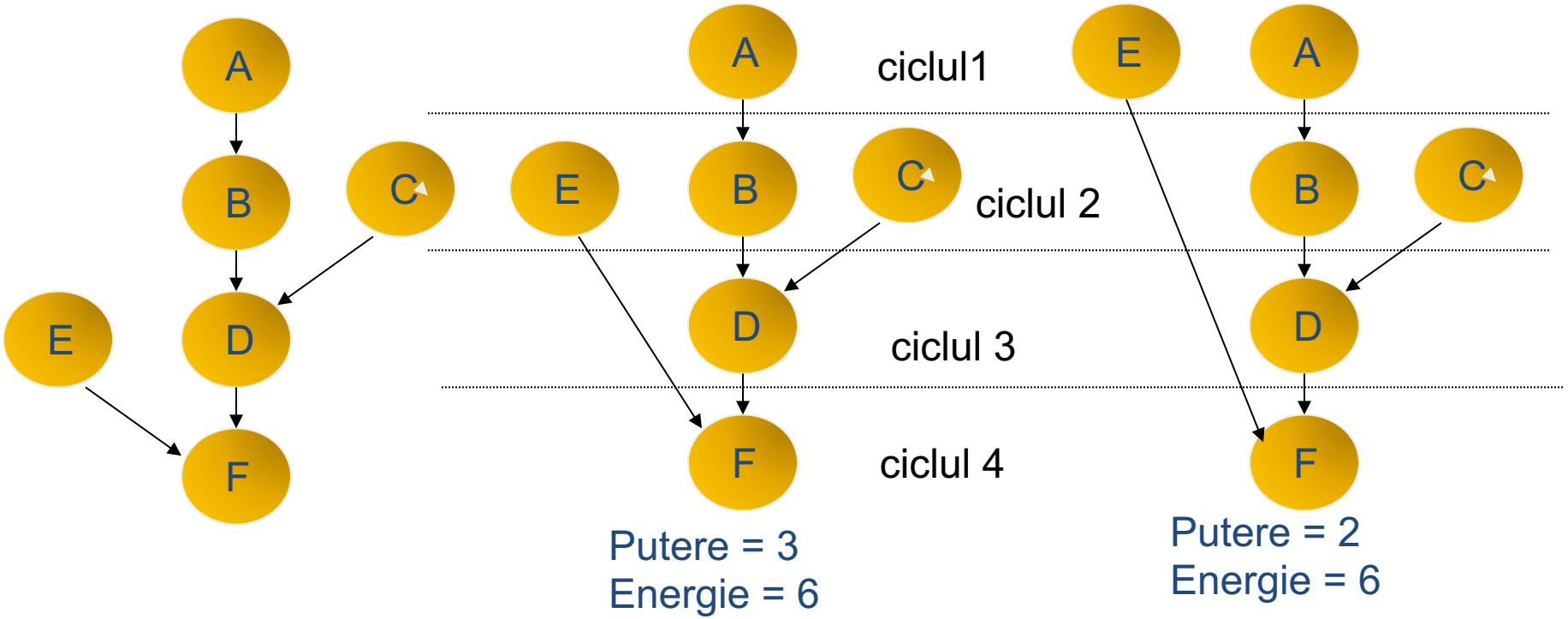
- Optimizări cache
  - Reordonează accesul la memorie a.î. să îmbunătățească numărul de hit-uri din cache

Minimizarea costurilor de acces la memorie

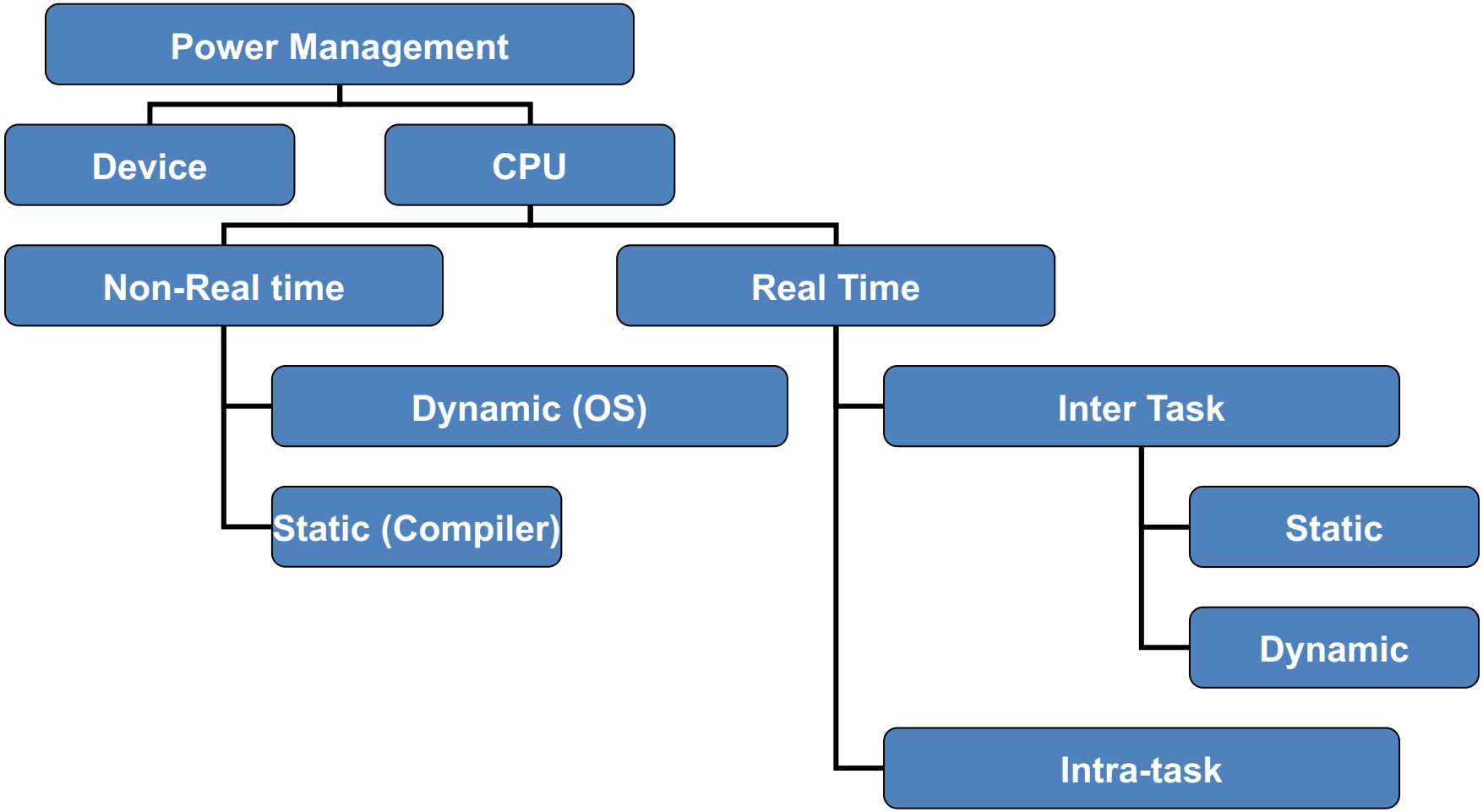
- Optimizarea buclelor de cod (loop unrolling, loop fusion) reduc numărul de accese la memorie
- Și mai eficient: cod care reduce numărul de accese la I/O și exploatează din plin ierarhia memoriilor
  - Alocarea datelor pentru minimizarea tranzițiilor pe busul de I/O
    - Codificarea magistralelor de adrese
  - Exploatarea ierarhiei de memorii
    - Datele pentru procesarea video sau DSP trebuiesc stocate în memoria superioară (registe generale sau cache)

Optimizarea energiei = Optimizarea puterii consumate?

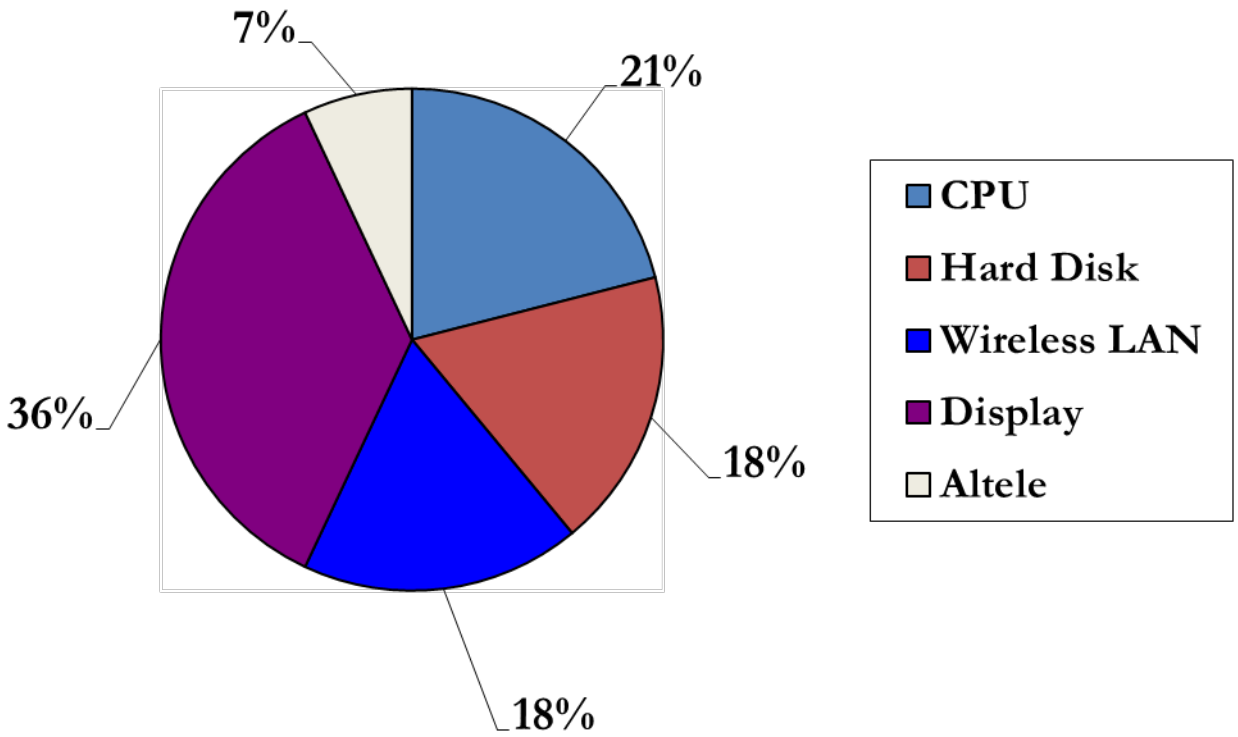
- Planificarea activității poate duce la reducerea puterii dar nu și a energiei



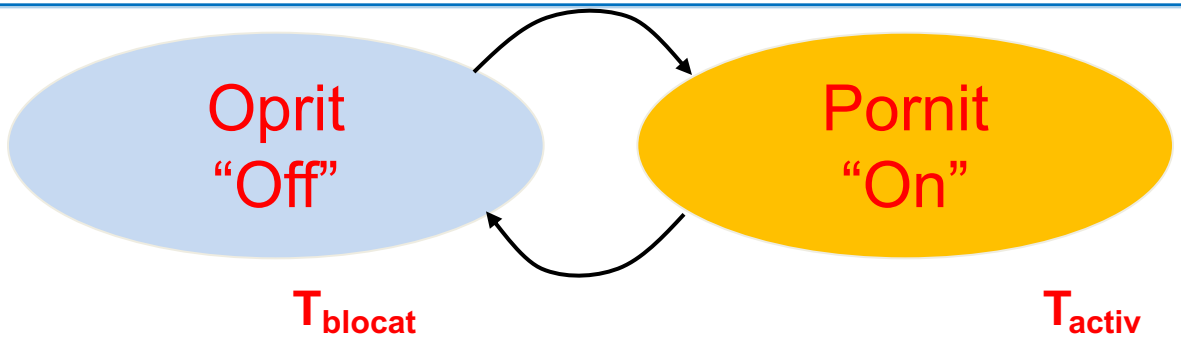
Taxonomia Power Management



Unde se consumă energia?



Power management (PM)

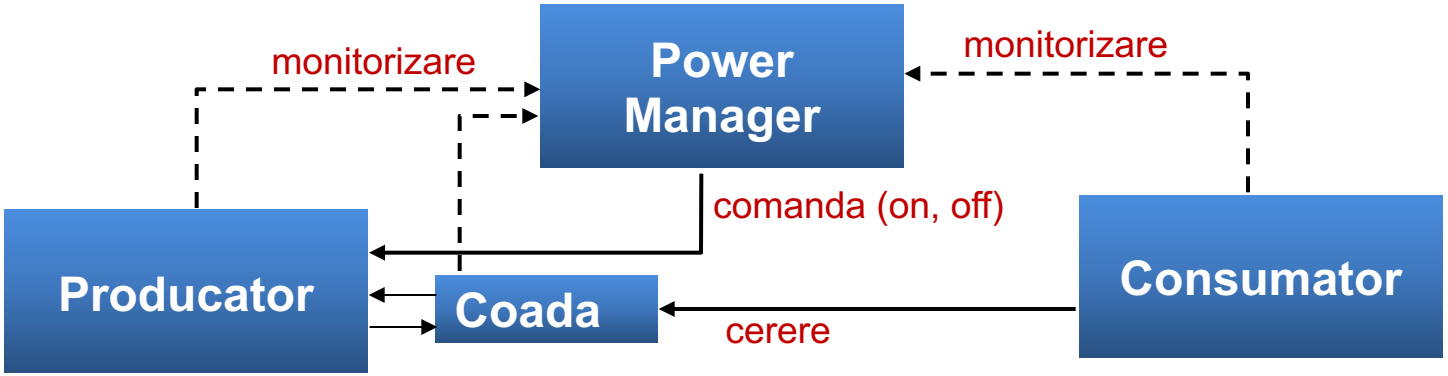


- Subsistemele au deseori cicli de funcționare limitați
  - CPU, hard-disk, interfața wireless sunt foarte des nefolosite
- Este o mare diferență între consumul în starea "on" și cel în starea "off"
  - Pentru Low-Power CPU:

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

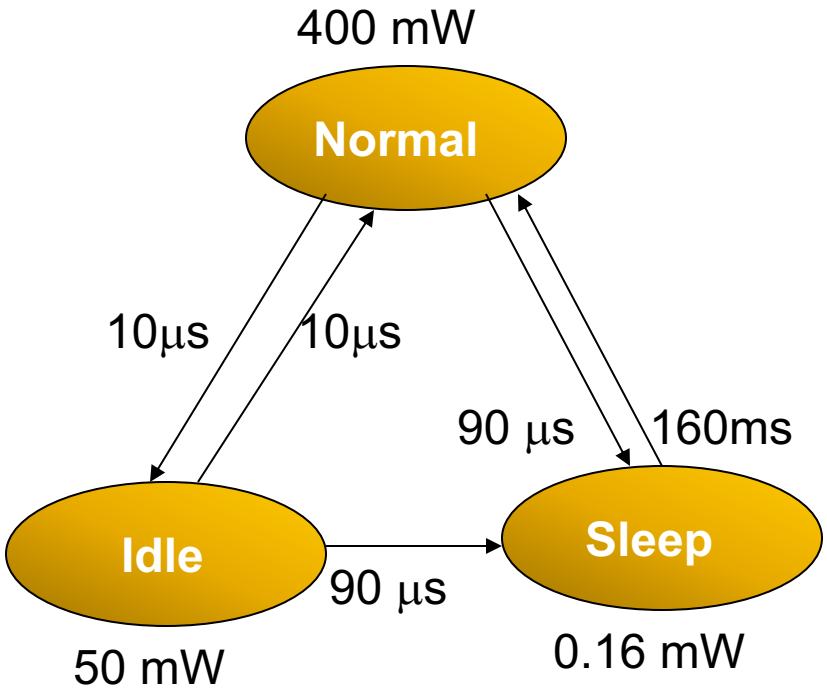
    1.35W (idle spinning) / 0.4W (standby) / 0.2W (sleep) / 4.7W (start-up)

Sistem generic cu power-management



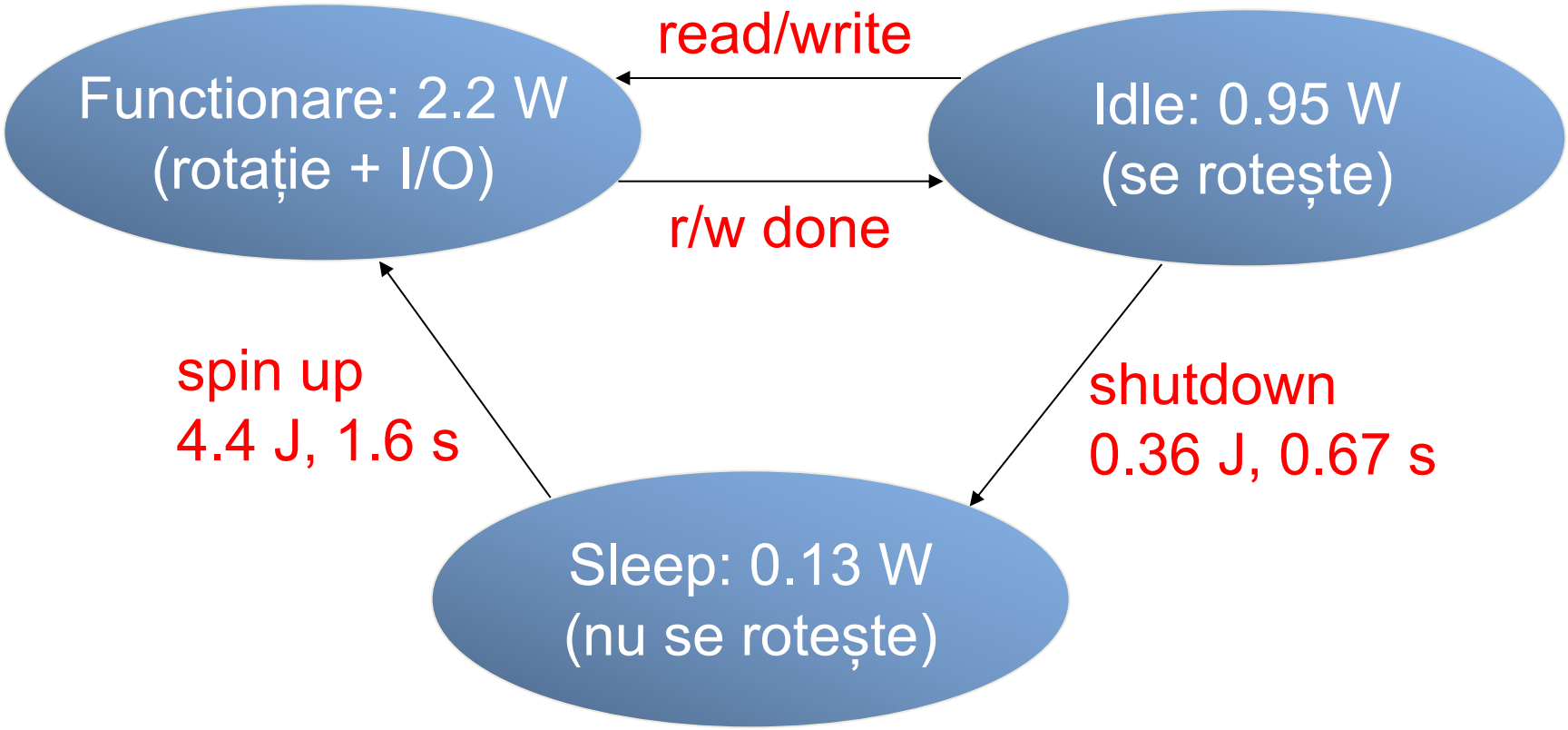
- Trebuie să existe o interfață între componentele care au nevoie de management (procesoare, drivere, periferice) și unitatea de power-management
  - Algoritm de power management
    - Când și cum se poate acționa
  - În esență, PM este un controller de sistem
- Reprezentare abstractă: automat de stări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

StrongARM SA-1100 – Stări de consum

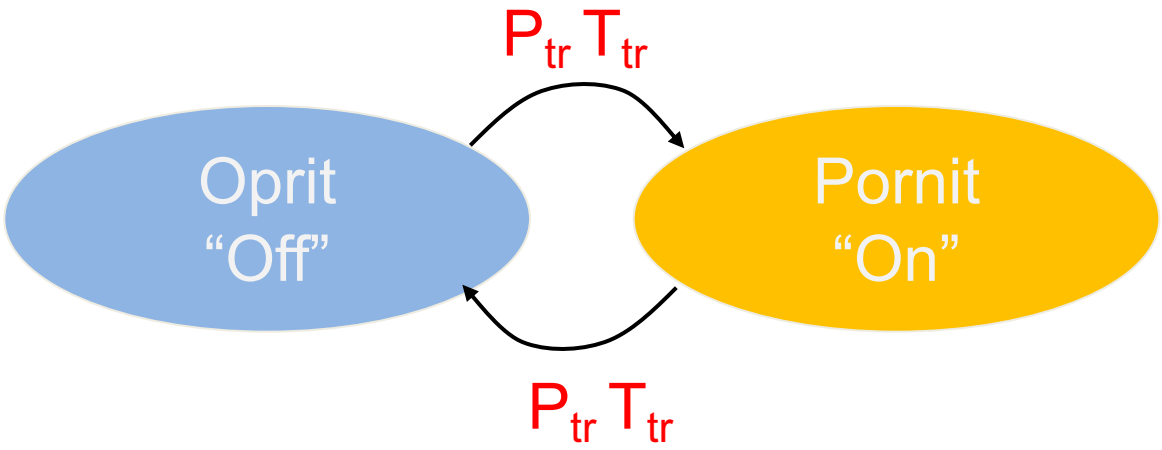


- Normal
  - CPU și perifericele sunt alimentate
  - Toate subansamblele primesc semnal de ceas
- Idle
  - Semnalul de ceas al CPU e oprit
  - Ceasurile către periferice sunt active
  - Se intoarce în modul normal printr-o întrerupere
- Sleep
  - DRAM în modul self-refresh
  - Toate funcțiile sunt dezactivate 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

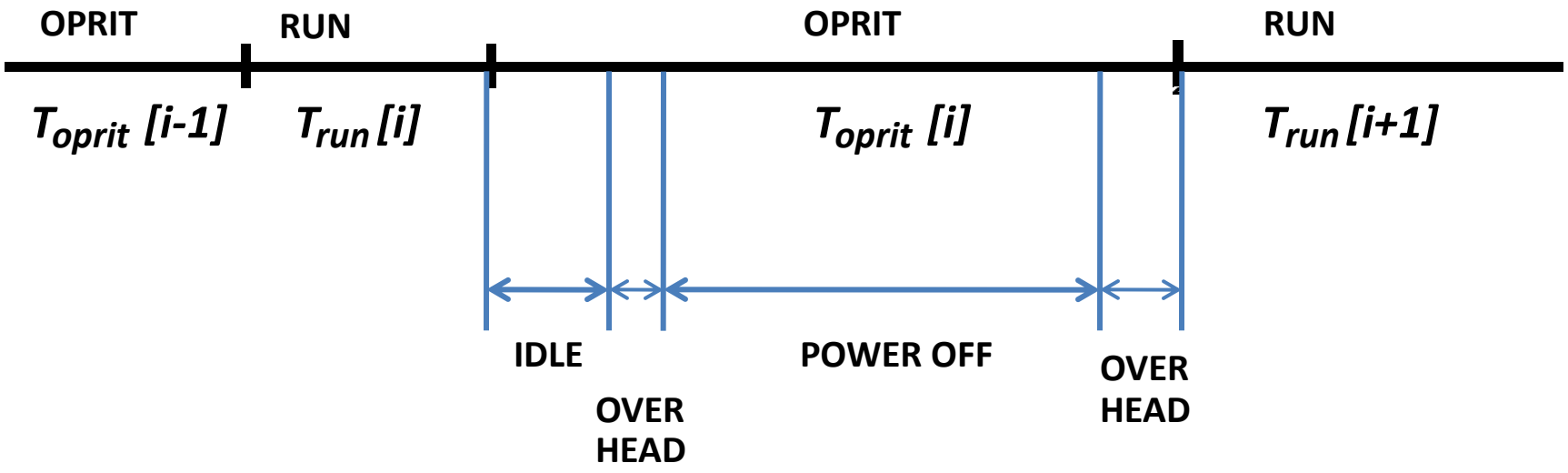
- Costul restartării: compromis între latență și consum
  - mărirea latenței (timpul de răspuns)
    - » Timpul de rotație pentru disc (spin-up)
  - mărirea consumului
    - » Curent mai mare la spin-up
- Când să se facă shutdown

  Optim vs. *Idle Time Threshold* vs. *Predictiv*
- Când să se facă 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

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

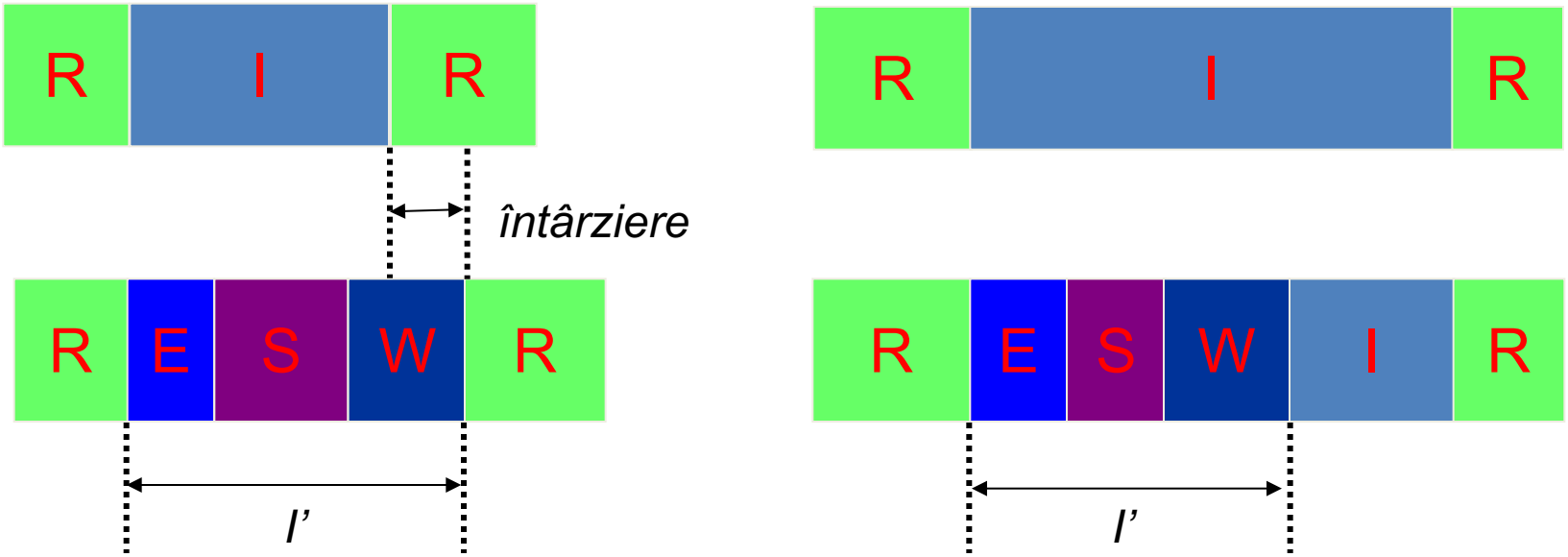
- Exemplu:

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

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

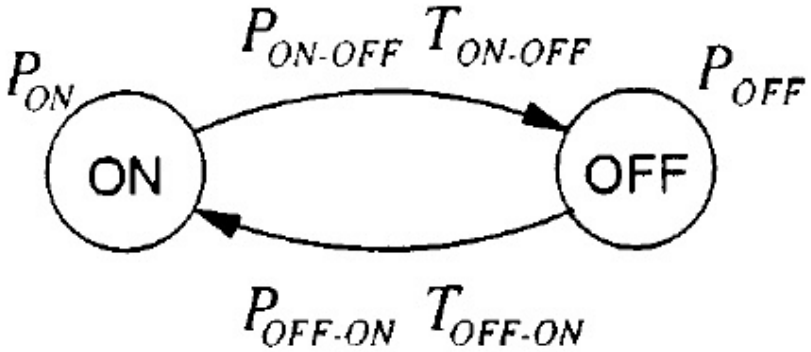
Pre-wakeup

- 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

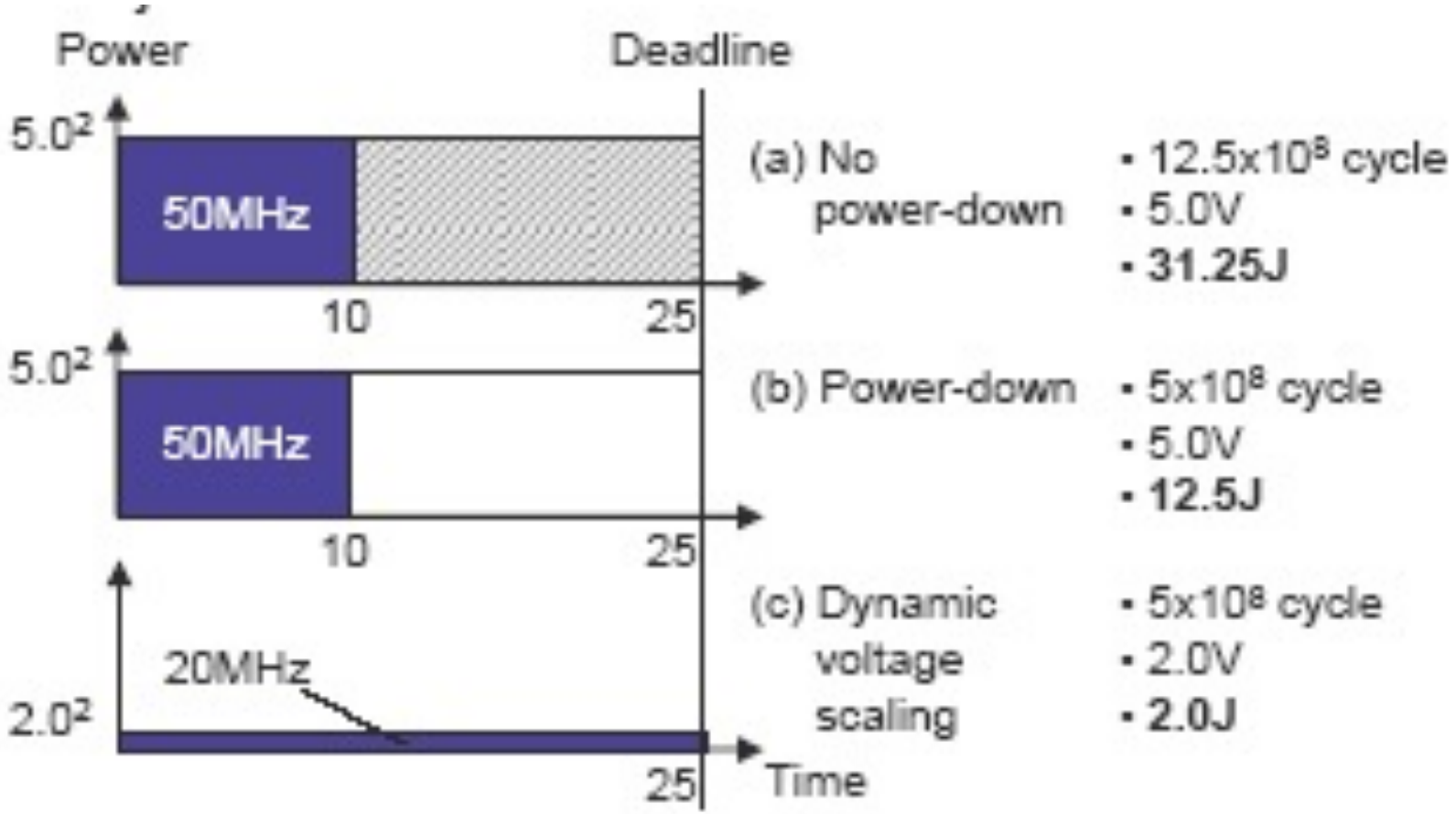
A = activitate (tranziții logice pe ciclu de ceas)

I_{leak} = curent de mers în gol

I_{short} = curent de scurt-circuit

τ = durata curentului de scurt-circuit

Tehnici

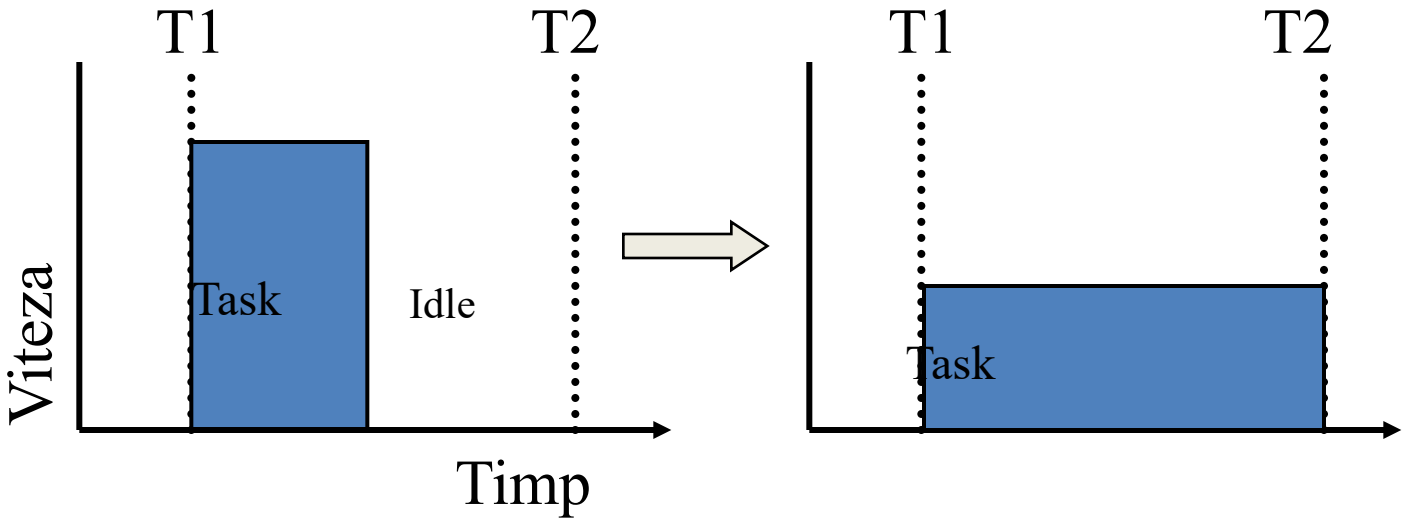

Power
Deadline
5.0²
50MHz
10
25
(a) No power-down
• 12.5x10⁸ cycle
• 5.0V
• 31.25J
5.0²
50MHz
10
25
(b) Power-down
• 5x10⁸ cycle
• 5.0V
• 12.5J
20MHz
2.0²
25
Time
(c) Dynamic voltage scaling
• 5x10⁸ cycle
• 2.0V
• 2.0J

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 cu 1/4 a consumului

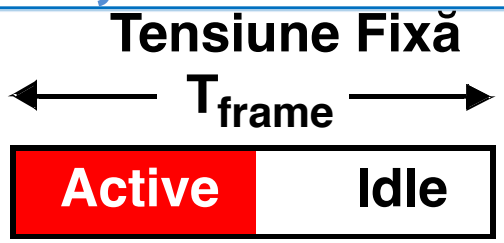


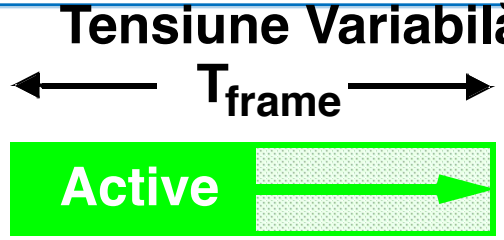
Reducerea tensiunii: probleme

- 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.

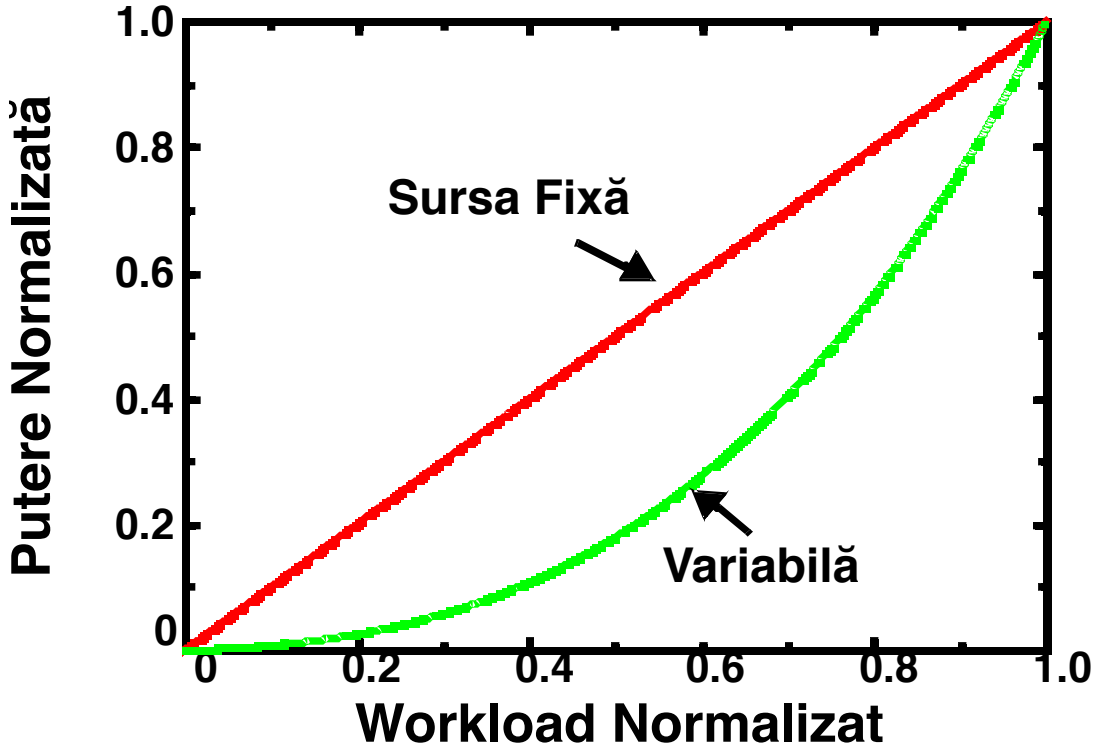
Variația Tensiunii de Alimentare





$E_{fixed} = 1/2 \times C V_{dd}^2$

$E_{var} = 1/2 \times C (V_{dd}/2)^2 = 1/4 E_{fixed}$

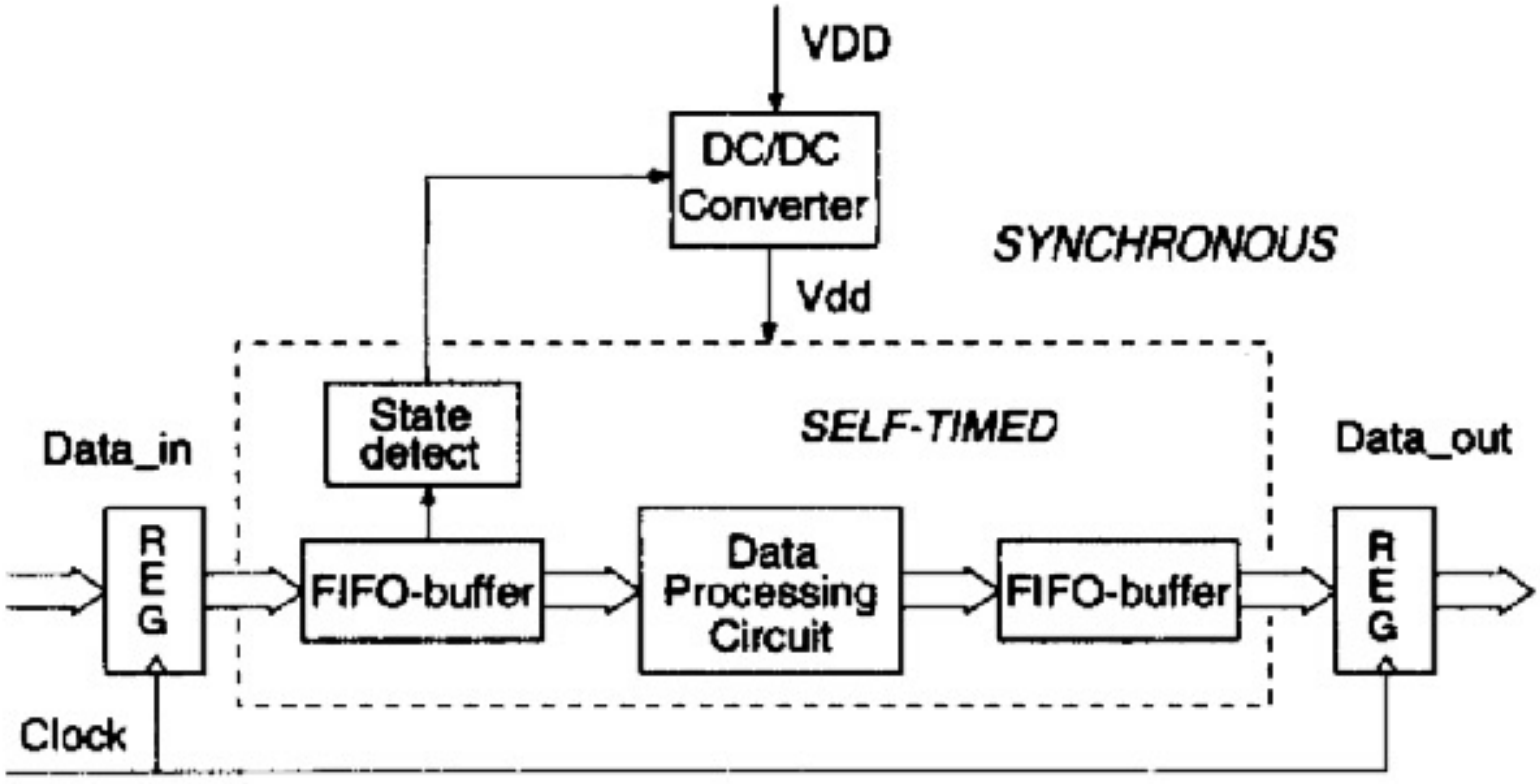


Sursa: [Gutnik96] (VLSI Symposium)

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?

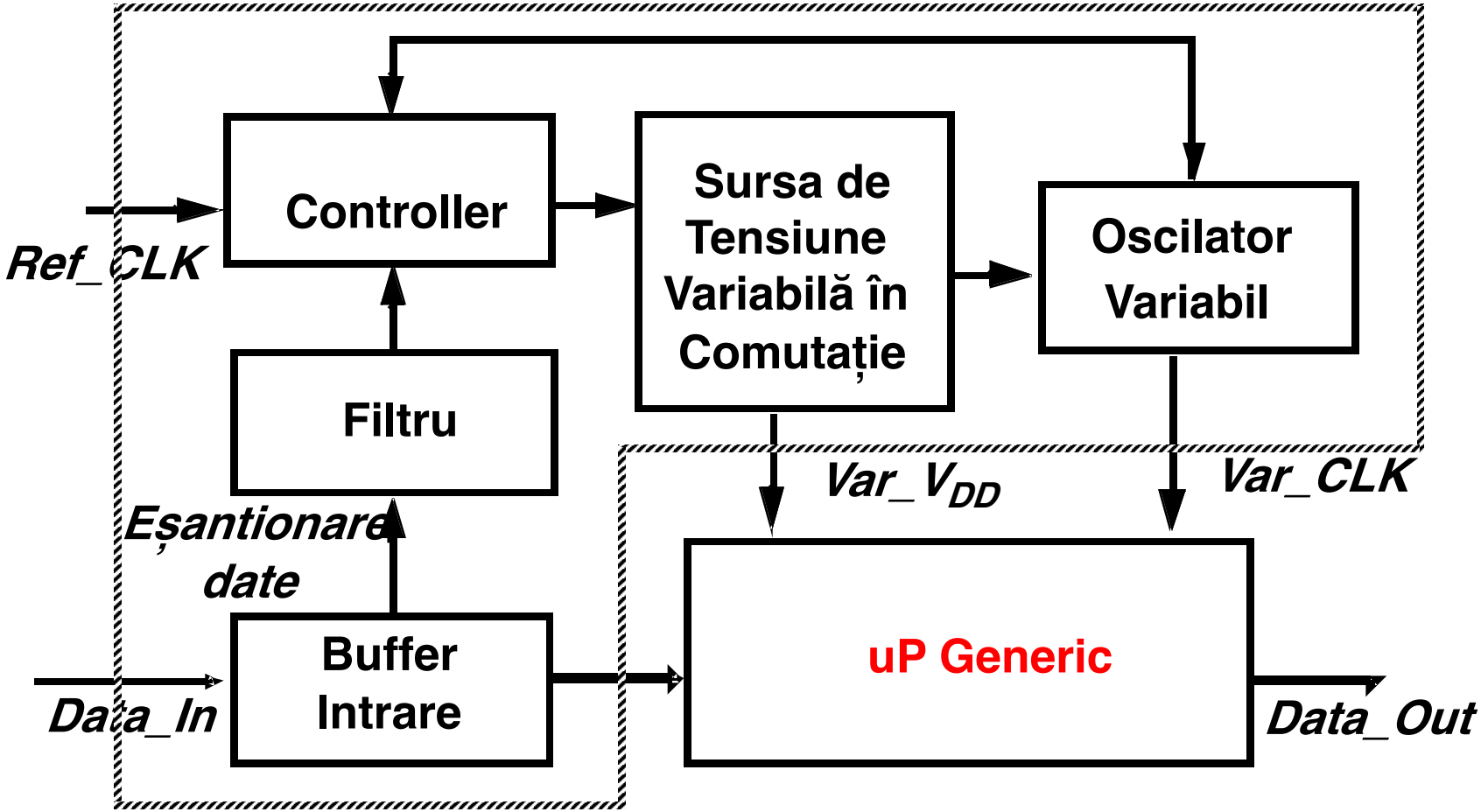
Scalare dinamică pentru sisteme asincrone



- 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

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

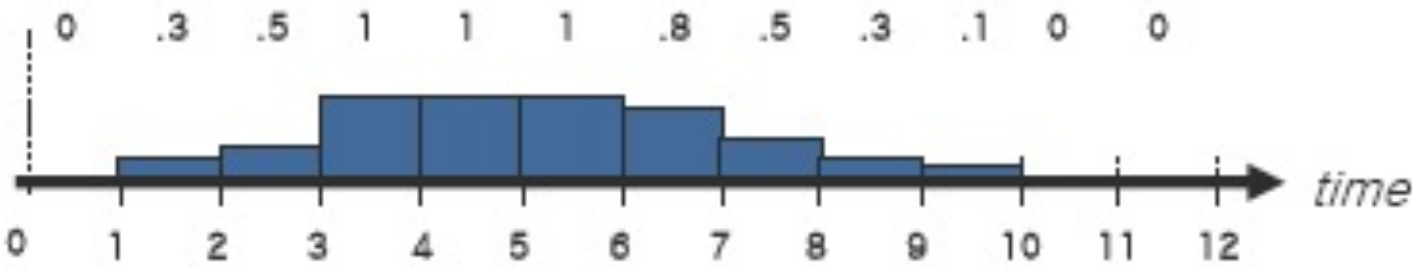
$$f_{clk} = 0.276 \times f_{max}$$

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

utilization = # cycles of busy interval / window size



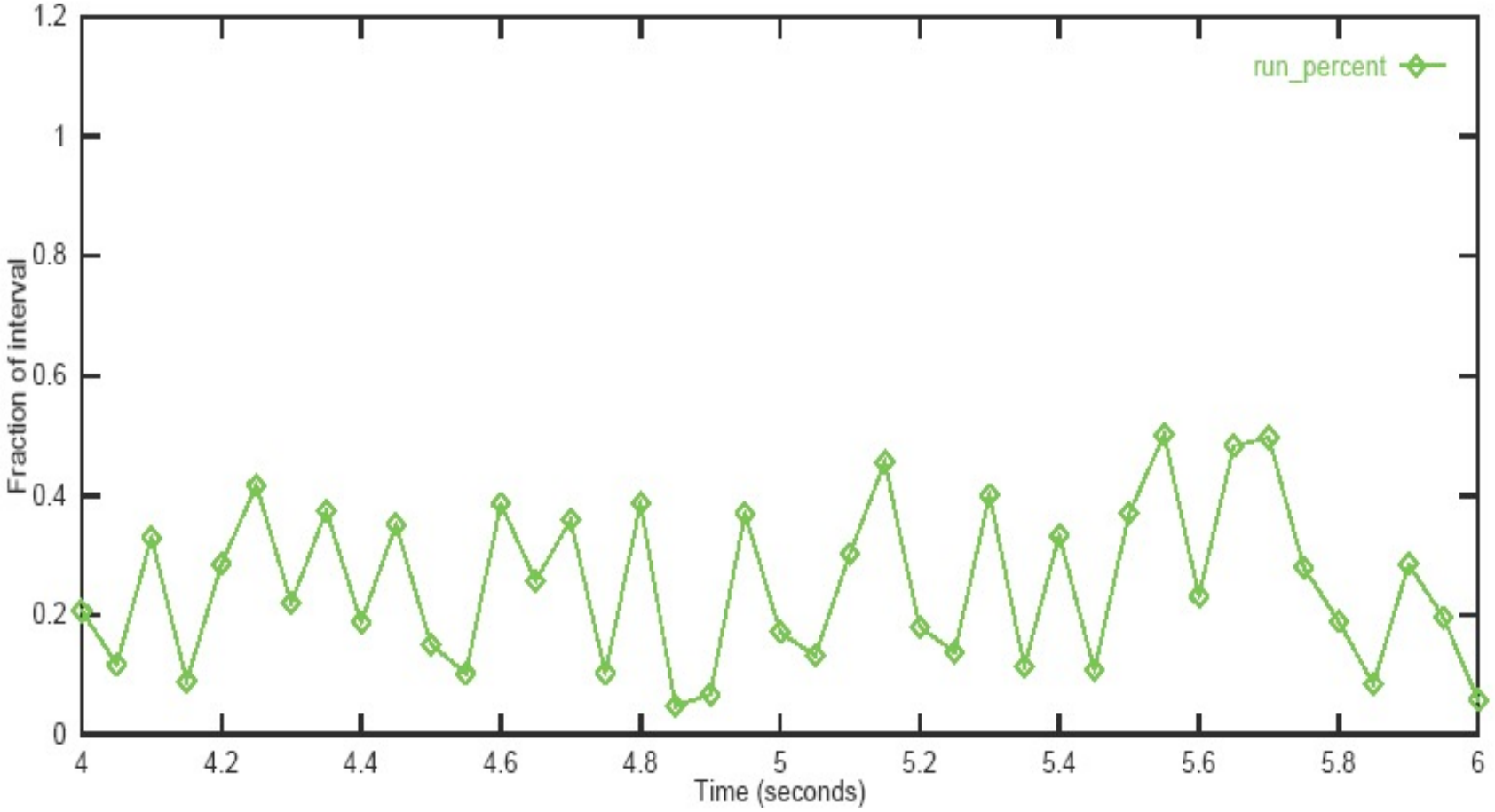
$$average = \frac{1}{3}0 + \frac{2}{9}0 + \frac{4}{27}(0.1) + \frac{8}{81}(0.3) + \cdots$$

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

CYCLE

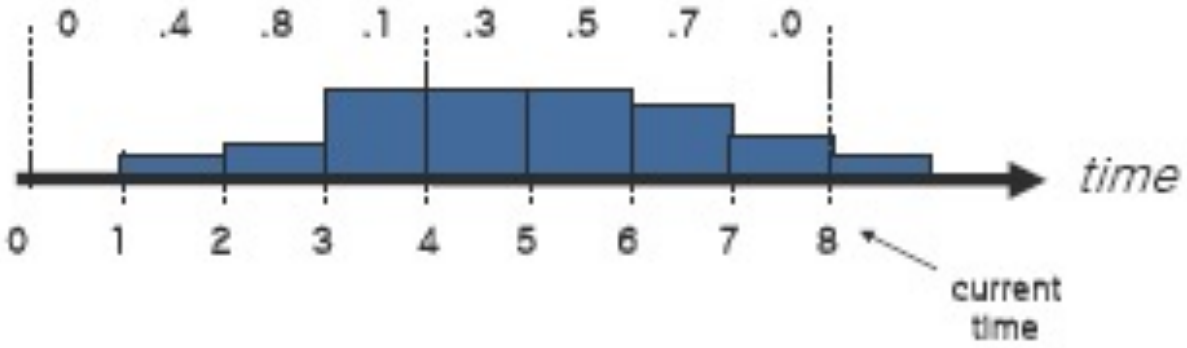
- Examinează ultimele 16 ferestre
- Există un ciclu de lungime X? (eroare < prag)
  - Dacă da, prezice extinderea ciclului
  - Dacă nu, folosește FLAT
- Un FLAT care “ghicește” mai bine

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 obligatorii

- 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