Sisteme Incorporate

Cursul 4

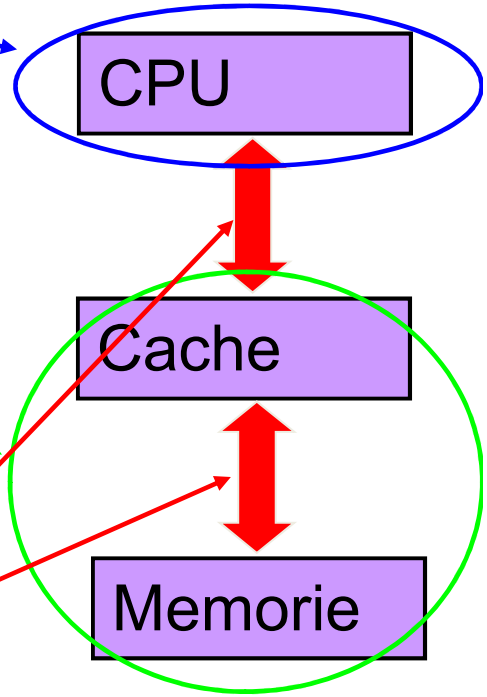
Software Power Management

Software Low-Power

- Software-ul foloseste resursele hardware
  - O parte semnificativa a functionalitatii unui sistem o constituie programul executat
  - Impactul unui program asupra consumului de energie al unui sistem este semnificativ
  - Software inteligent proiectat -> reducerea consumului
- Fiecare instructiune are un cost in energie

Strategii low-power

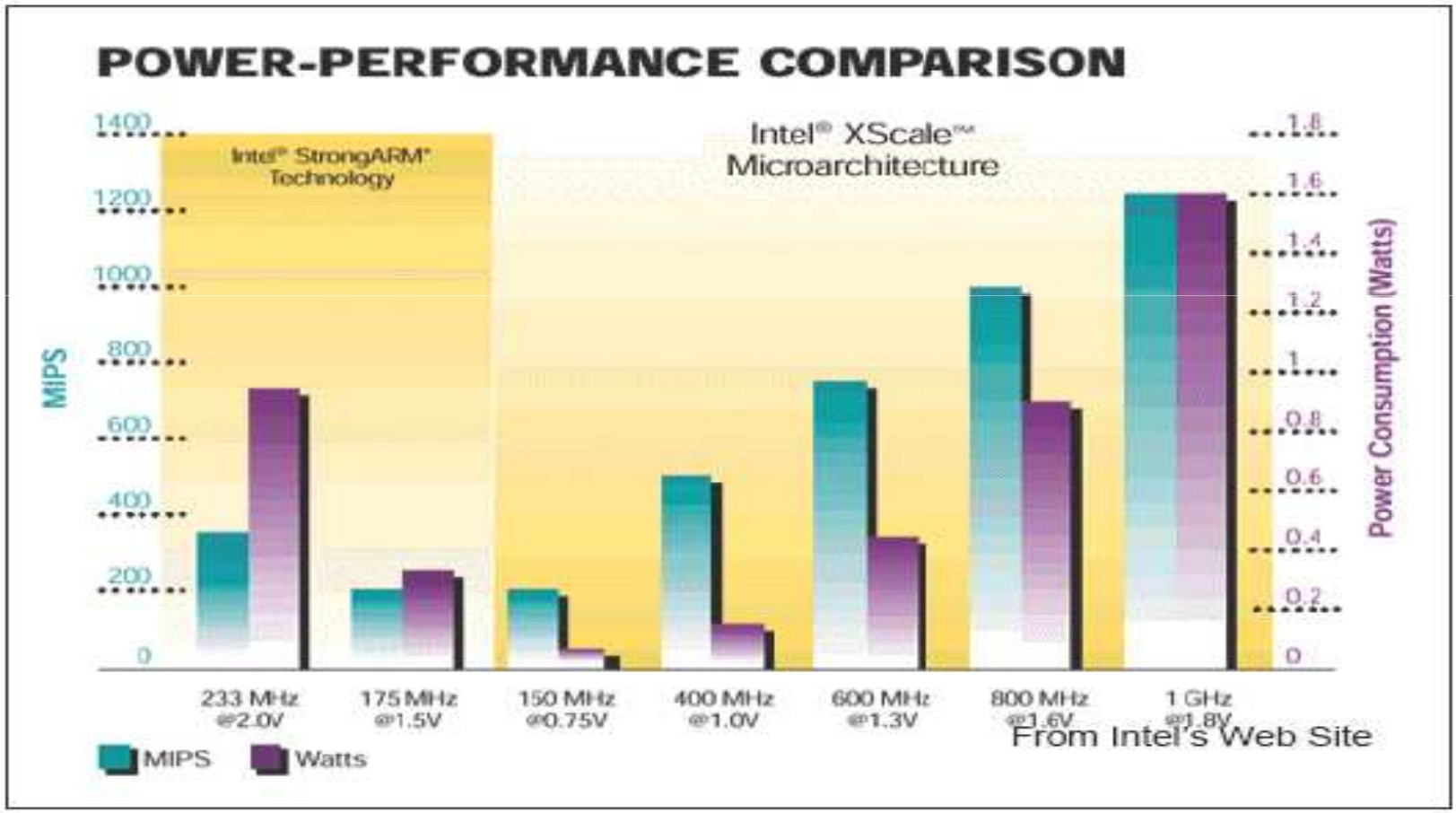
- Cod care ruleaza pe CPU
  - Optimizari in cod pentru low power
- Cod care acceseaza memoria
  - Optimizarea accesului la memorie
- Traficul de date pe magistrale
  - Codificare low-power
- Power Management controlat de compilator



Modalitati de reglare

- Hibernare
  - Tranzitia dintr-o stare de consum ridicat intr-una de consum redus (low power)
- Scalarea dinamica a frecventei si a tensiunii
  - Se stabilesc mai multe valori fixe pentru f si V
  - Instructiuni speciale prin care se stabilesc f si V
    - Intel XScale, StrongARM SA-2, AMD mobile K6 Plus
- Procesorul tine cont de temperatura de lucru
  - Thermally aware processor

Performanta = f(V, Hz)



Thermally aware processor

- Diferenta mare intre puterea nominala si puterea maxima a unui procesor
- Proiectarea unui procesor care sa functioneze la putere maxima este dificila (maxim 60W disipati)
- Unitatea de detectie monitorizeaza temperatura si genereaza o intrerupere atunci cand aceasta a depasit un nivel de prag
- Rutina de tratare a intreruperii duce procesorul in modul low-power
- Trade-off: are impact asupra puterii de procesare

Optimizarea codului orientata low-power

- Instructiunile high-level pot fi compilate in mai multe secvente echivalente de instructiuni simple (C -> ASM)
    - Instructiuni diferite -> costuri de energie diferite
    - Ordinea executiei unui set de instructiuni afecteaza consumul
- Selectarea instructiunilor
  - Trebuie facut un “amestec” de instructiuni care dau cel mai mic consum de energie
- Memorii duale
  - Doua bancuri de memorie pe acelasi chip
    - Load dual vs. load simplu
    - Reducerea consumului cu aproape 50%

Optimizarea codului orientata low-power

- Reordonarea instructiunilor in cod pentru a reduce efectul comutarii unitatilor functionale, magistralelor de memorie,I/O etc.
- Operand swapping
  - Rotirea operanzilor intre ei la intrarea in multiplicator
  - Inmultirea da acelasi rezultat, dar consumul este redus semnificativ!
- Optimizari standard ale compilatorului
  - Software pipelining, dead code elimination, eliminarea redundantelor
  - Alocarea registrelor
- Utilizarea unui stil de programare in functie de procesorul folosit
  - e.g. pe ARM tipul de date `int` este cu ~ 20% mai eficient decat `char` sau `short` pentru ca acestea folosesc extensia de semn
  - e.g. pe ARM pot fi folosite sufixele conditionale in locul instructiunilor de branch

Minimizarea costurilor de acces la memorie

- Reducerea numarului de accese la memorie -> foloseste mai mult registrele generale
  - Costul unui acces de registru general << acces adresa de memorie

```
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]);
}
```

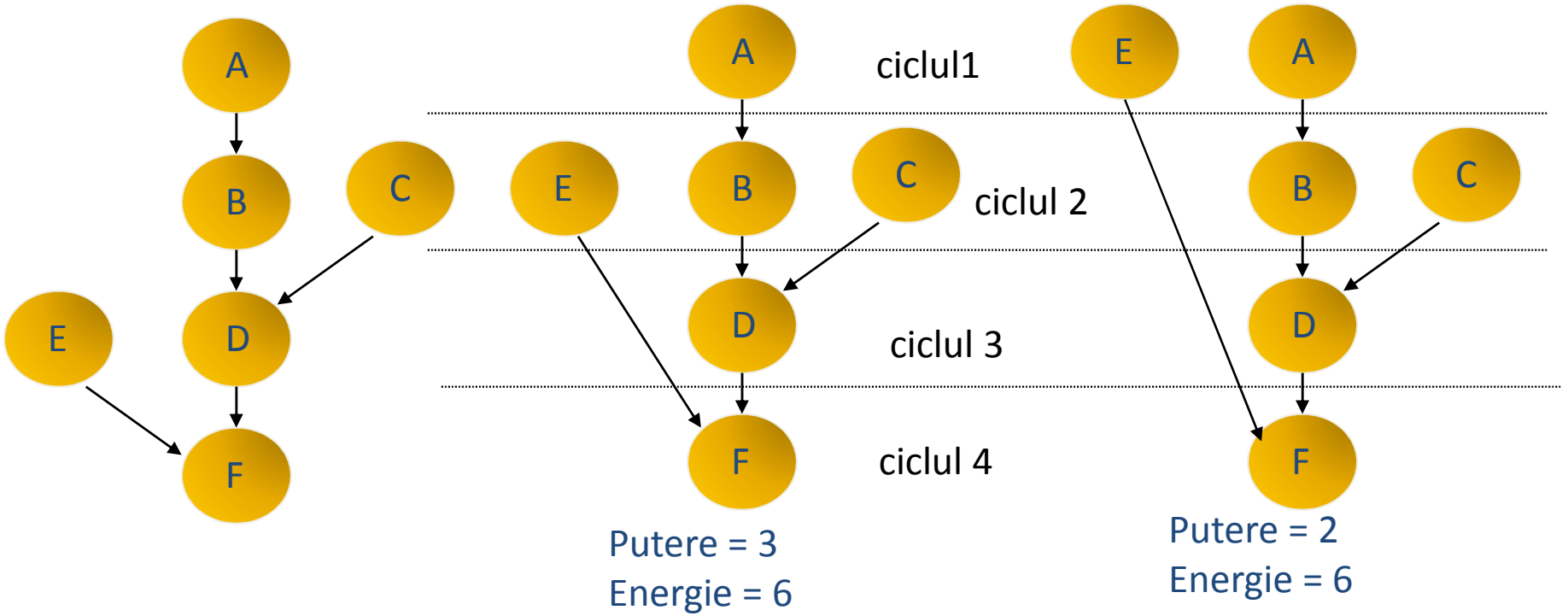
- Optimizari cache
  - Reordoneaza accesul la memorie a.i. sa imbunatateasca numarul de hit-uri din cache

Minimizarea costurilor de acces la memorie

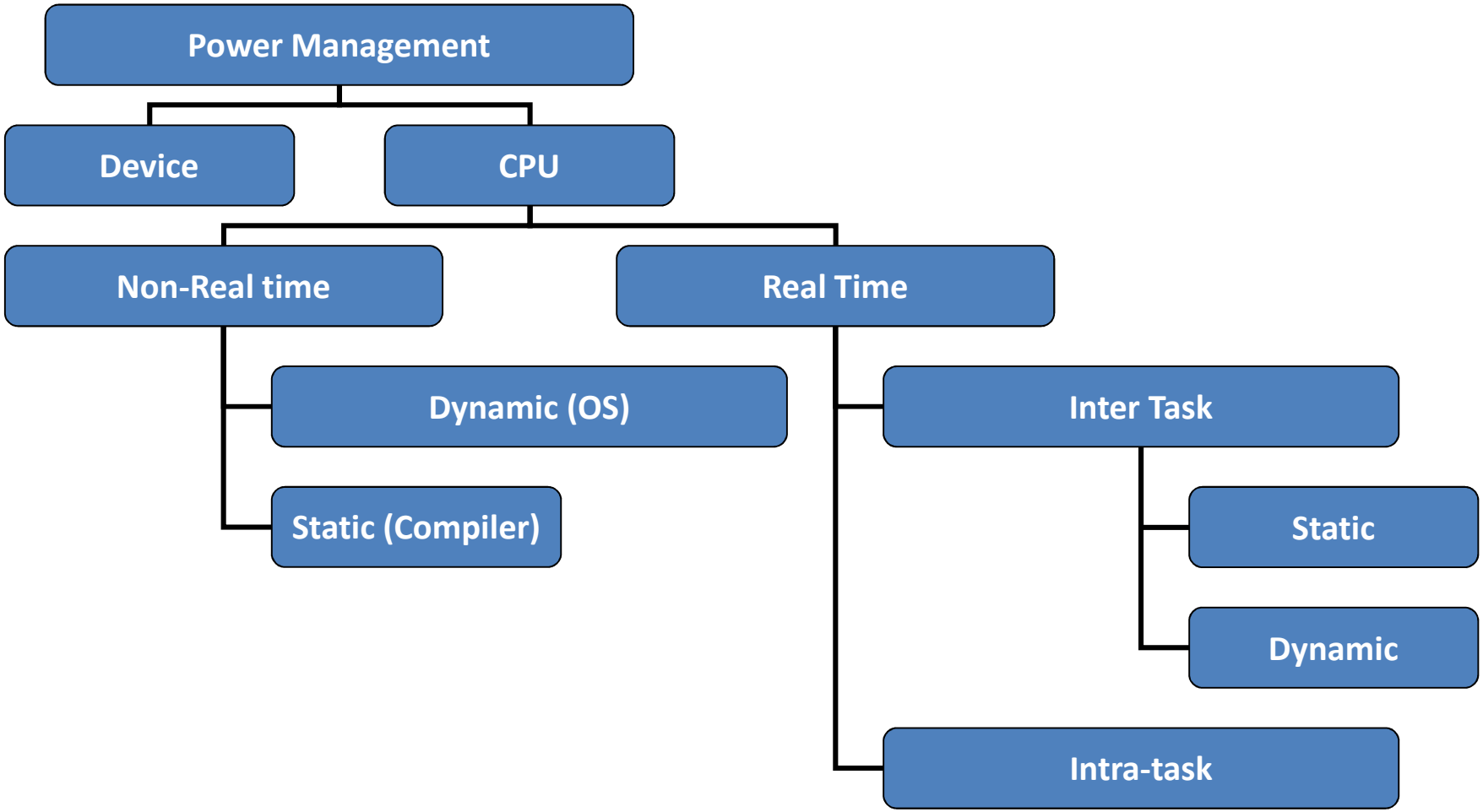
- Optimizarea buclelor de cod (loop unrolling, loop fusion) reduc numarul de accese la memorie
- Si mai eficient: cod care reduce numarul de accese la I/O si exploateaza din plin ierarhia memoriilor
  - Alocarea datelor pentru minimizarea tranzitiilor pe busul de I/O
    - Codificarea magistralelor de adrese
  - Exploatarea ierarhiei de memorii
    - Datele pentru procesarea video sau DSP trebuiesc stocate in in memoria superioara (registe generale sau cache)

Optimizarea energiei = Optimizarea puterii consumate?

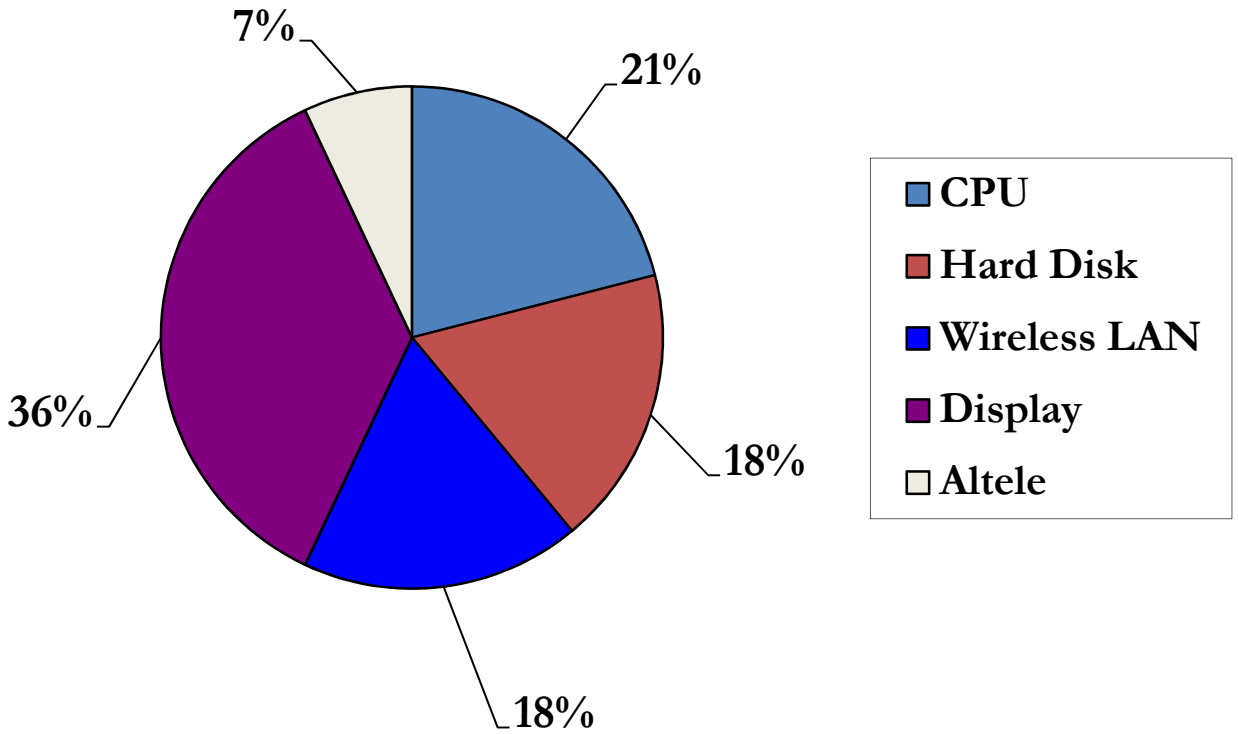
- Planificarea activitatii poate duce la reducerea puterii dar nu si a energiei



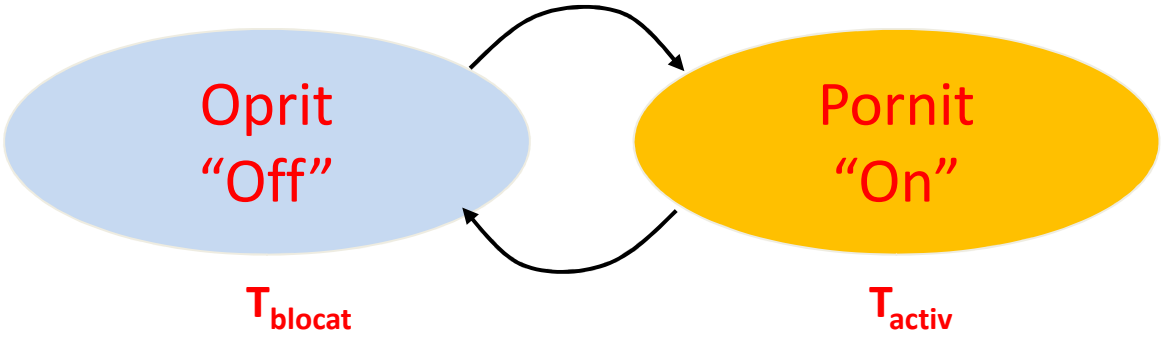
Taxonomia Power Management



Unde se consuma energia?



Power management (PM)

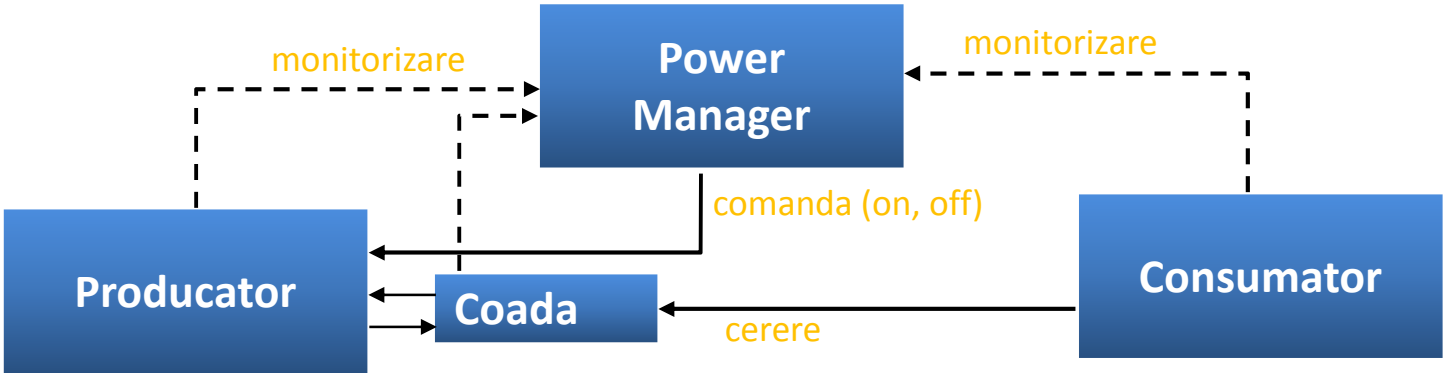


- Subsistemele au deseori cicli de functionare limitati
  - CPU, hard-disk, interfata wireless sunt foarte des nefolosite
- Este o mare diferenta intre consumul in starea “on” si cel in starea “off”
  - Pentru Low-Power CPU:

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

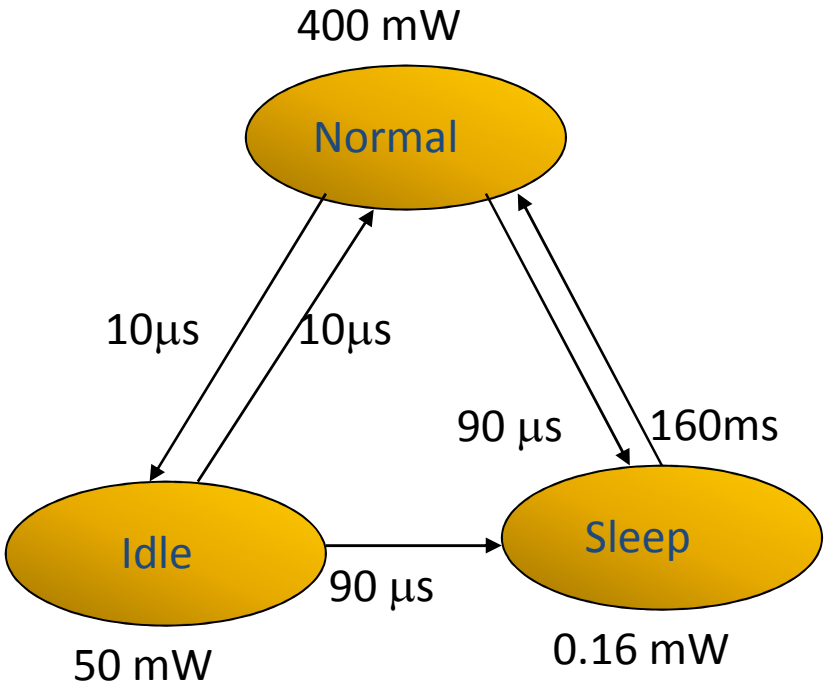
    1.35W (idle spinning) / 0.4W (standby) / 0.2W (sleep) / 4.7W (start-up)

Sistem generic cu power-management



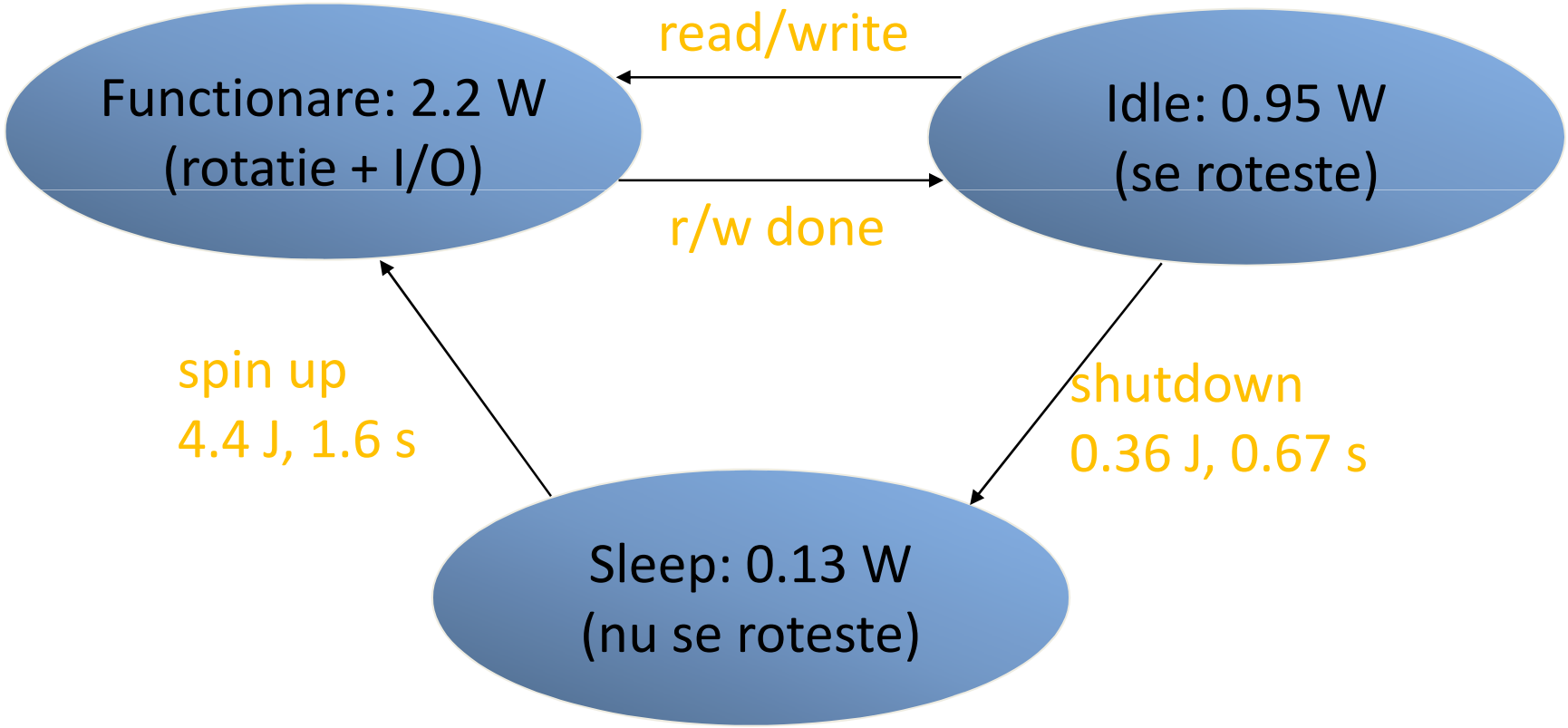
- Trebuie sa existe o interfata intre componentele care au nevoie de management (procesoare, drivere, periferice) si unitatea de power-management
    - Algoritm de power management
        - Cand si cum se poate actiona
    - In esenta, PM este un controller de sistem
- Reprezentare abstracta: automat de stari
    - Stari = politica de PM aplicata si consumul de energie
    - Muchii = timpi de tranzitie si consum per tranzitie

StrongARM SA-1100 – Stari de consum

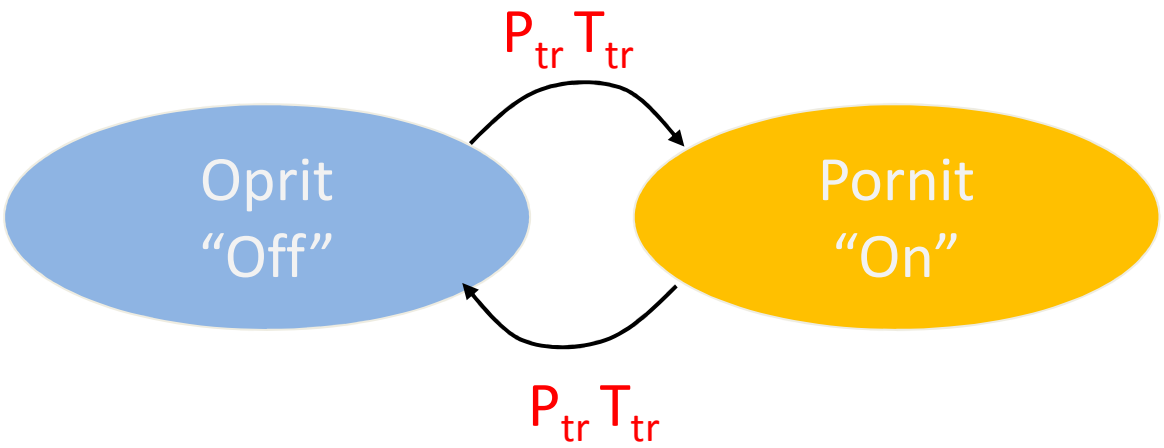


- Normal
  - CPU si perifericele sunt alimentate
  - Toate subansamblele primesc semnal de ceas
- Idle
  - Semnalul de ceas al CPU e oprit
  - Ceasurile catre periferice sunt active
  - Se intoarce in modul normal printr-o intrerupere
- Sleep
  - DRAM in modul self-refresh
  - Toate functiile sunt dezactivate mai putin ceasul de timp real
  - “Trezire” printr-o intrerupere preprogramata sau de la utilizator

Fujitsu MHF 2043 AT (hard-disk)



Cand e util PM?



- Daca T_{tr}=0, P_{tr}=0 atunci politica de PM e evidenta
  - Opreste o componenta cand nu ai nevoie de ea
- Dar, de obicei T_{tr}!= 0, P_{tr} != 0
  - Opreste doar atunci cand T_{idle} este suficient de mare
  - Problema: Cum pot sa-mi dau seama cat o sa fie T_{idle} ?

Probleme date de Shutdown

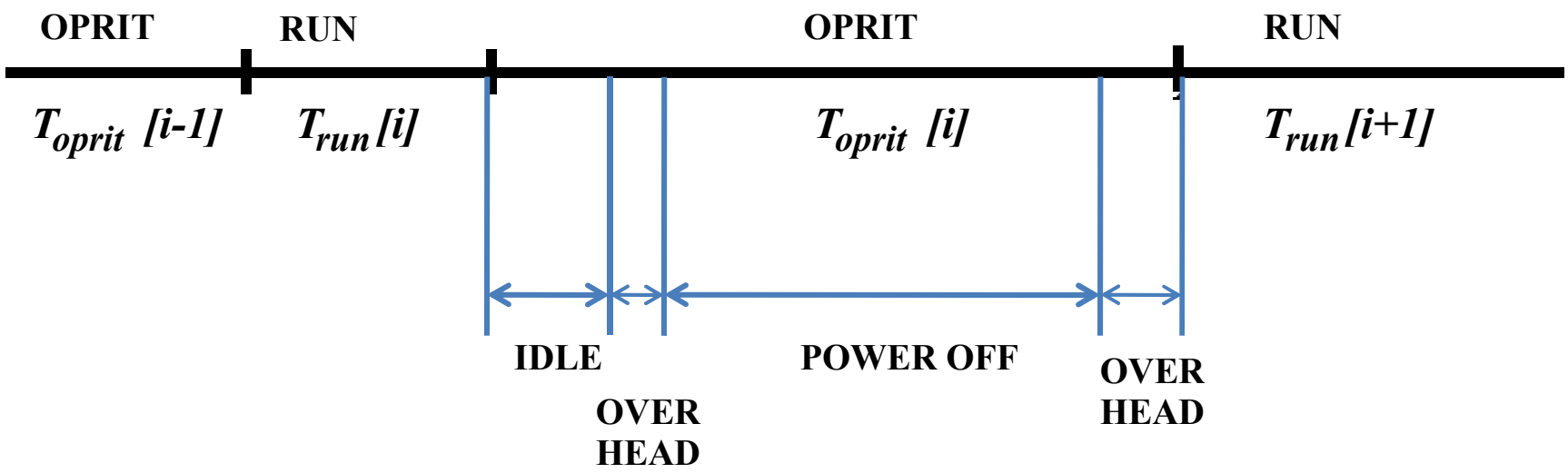
- Costul restartarii: compromis intre latenta si consum
  - marirea latentei (timpul de raspuns)
    - » Timpul de rotatie pentru disc (spin-up)
  - Marirea consumului
    - » Curent mai mare la spin-up
- Cand sa se faca shutdown

  Optim vs. *Idle Time Threshold* vs. *Predictiv*
- Cand sa se faca wake-up

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

Abordare Conventionala

"Opreste sistemul cand utilizatorul nu l-a mai folosit un timp de x minute si restarteaza la cerere"



Abordarea bazata pe predictie

"Foloseste informatiile stranse pentru a determina in ce situatie $T_{oprit}[i]$ e suficient de mare

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

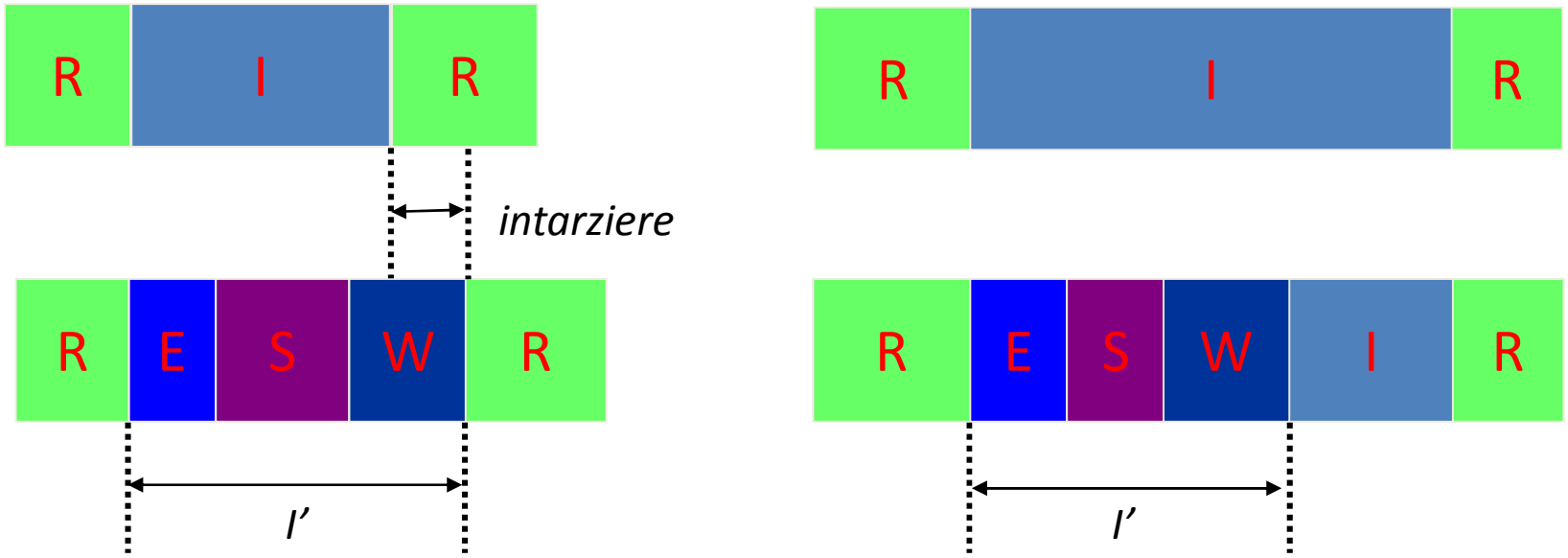
- Exemplu:

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

  - Poate duce la reducerea de pana la 20x a consumului
  - Elimina timpii de asteptare in care sistemul este nefolosit

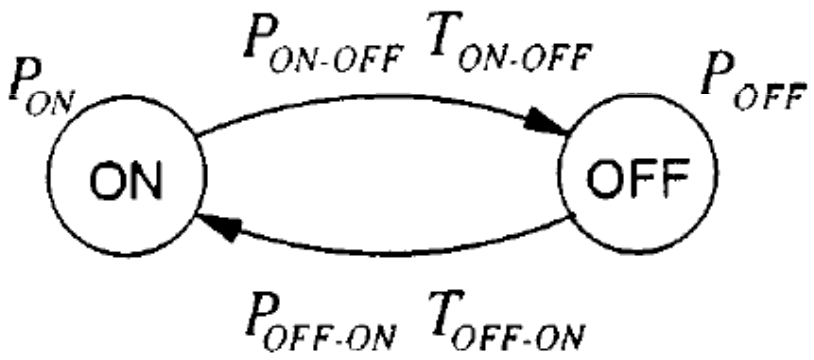
Pre-wakeup

- Initializarea sistemului dureaza timp, lucru care afecteaza performantele
- Daca timpii de lucru pot fi prevazuti, sistemul poate fi pre-initializat inainte de pornirea efectiva



Breakeven Point

- Breakeven point: timpul minim de idle pentru care e eficient sa se faca 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 ecranuluiului
- Oprirea motoarelor (si in general a tuturor efectoarelor mecanice)

Puterea in CMOS

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

P = putere totala

V = tensiunea de alimentare

f = frecventa de ceas

C = capacitatea liniilor de iesire

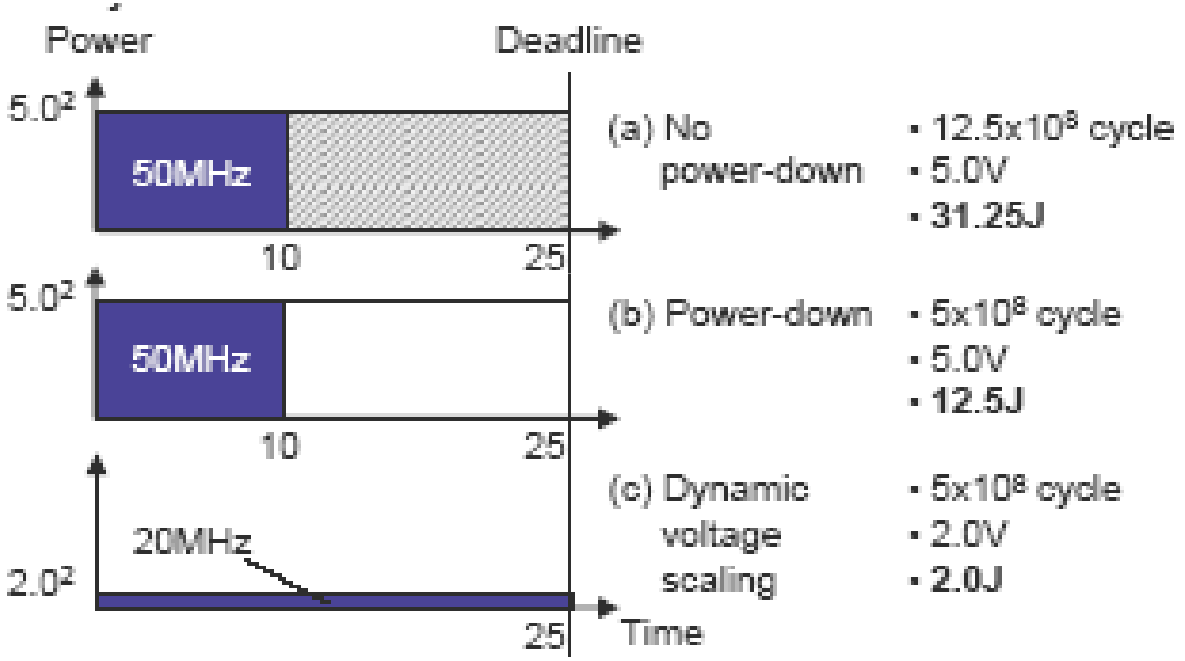
A = activitate (tranzitii logice pe ciclu de ceas)

I_{leak} = curent de mers in gol

I_{short} = curent de scurt-circuit

τ = durata curentului de scurt-circuit

Tehnici

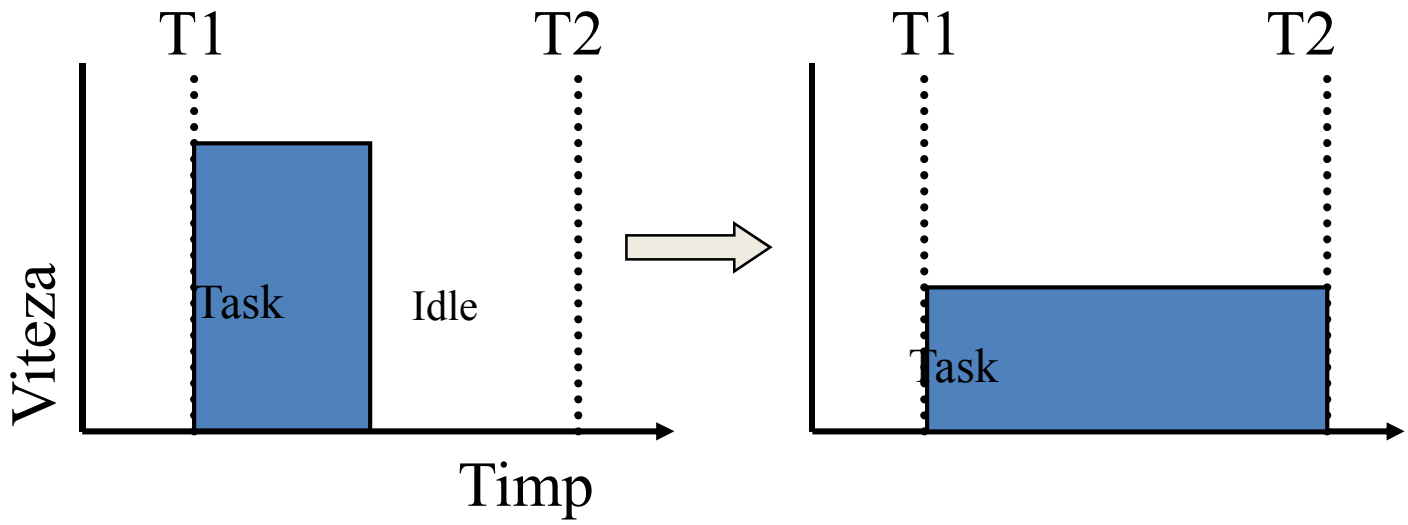

Power
Deadline
5.0²
50MHz
10
25
(a) No power-down
• 12.5x10⁸ cycle
• 5.0V
• 31.25J
5.0²
50MHz
10
25
(b) Power-down
• 5x10⁸ cycle
• 5.0V
• 12.5J
20MHz
2.0²
25
Time
(c) Dynamic voltage scaling
• 5x10⁸ cycle
• 2.0V
• 2.0J

Dynamic voltage/frequency scaling

- Reducerea consumului vs. intarzieri
- Sarcini non-real time
  - Fara constrangeri de timp
  - Nu se stie cand si cate date vor fi procesate intr-un interval
  - Nu se stie cat timp va dura procesarea datelor
- Predictia incarcarii viitoare a sistemului
  - Predictie: Cat de incarcat va fi procesorul in viitor
  - Smoothing: “Nivelarea” timpilor de executie

Reducerea tensiunii de alimentare

- Exemplu: sarcina care dureaza 100ms si 50ms cu CPU la viteza maxima
  - Sistem normal: 50ms calcule, 50ms idle time
  - Jumatate din viteza/tensiune: 100ms calcule, 0ms idle
  - Acelasi numar de instructiuni DAR reducere cu 1/4 a consumului

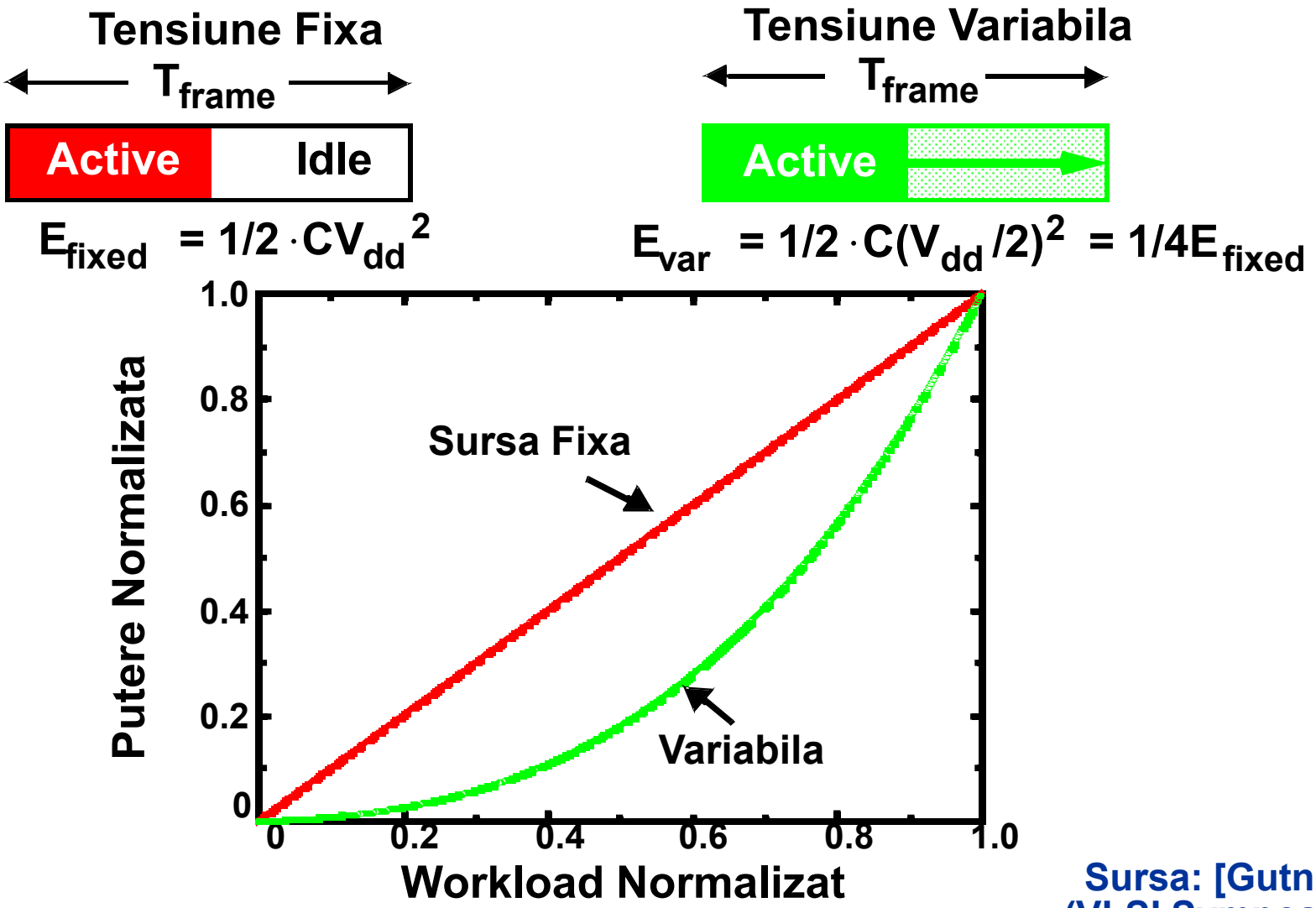


Reducerea tensiunii: probleme

- Abordare statica: Tensiunea minima de alimentare este dictata de timpul maxim de executie al unui task
  - Nu e o problema daca timpul de executie nu e important
    - » Productivitatea poate fi imbunatatita prin paralelizare, pipelining (vezi curs anterior)
  - Sistemele reale au perioade intense de calcul si timpi de mers in gol care nu pot fi prevazuti

Solutia: variatia dinamica a tensiunii.

Variatia Tensiunii de Alimentare

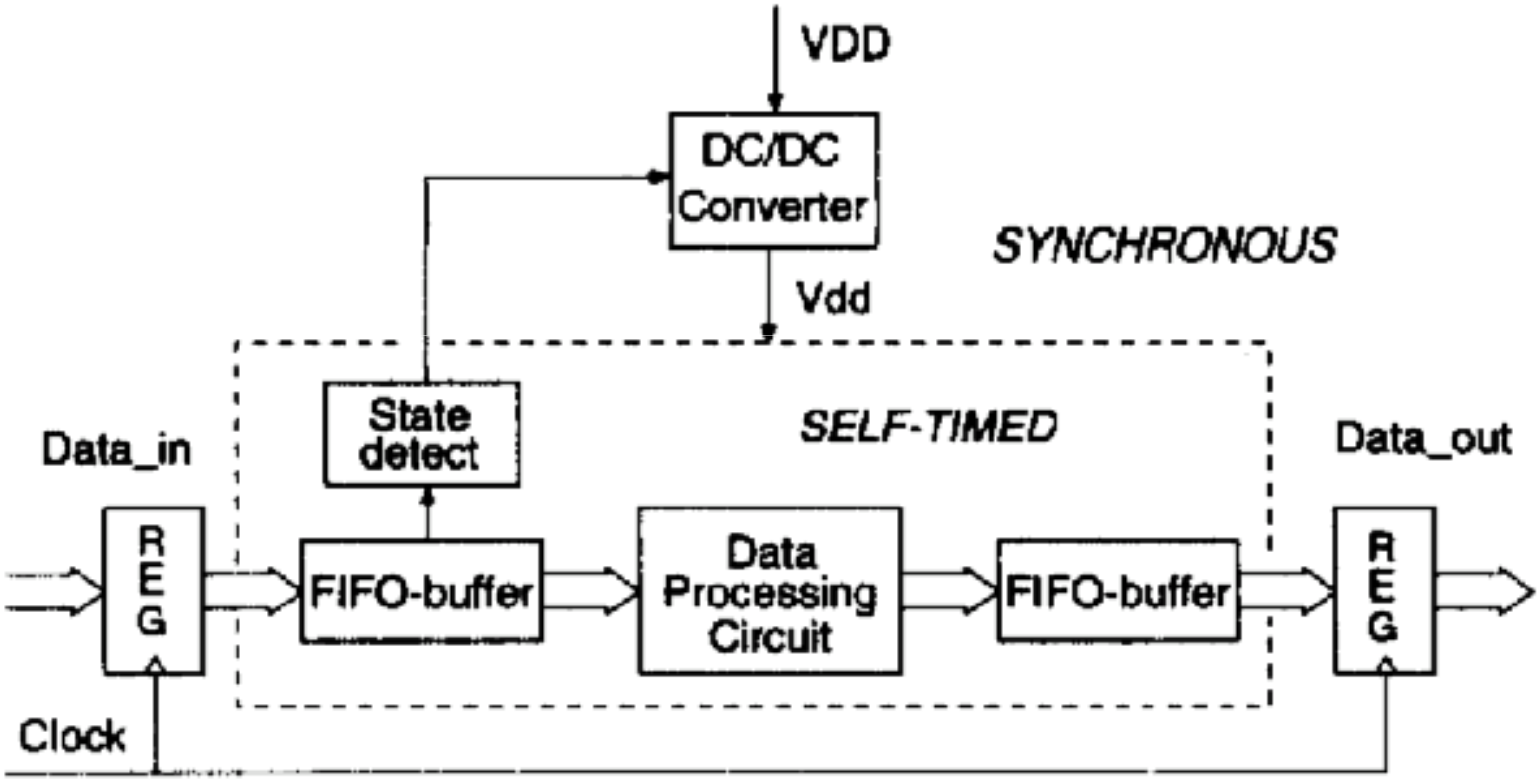


Sursa: [Gutnik96]
(VLSI Symposium)

Variatia Dinamica a Tensiunii de Alimentare (DVS)

- Puterea si frecventa de ceas depind de tensiunea de alimentare
- Tehnologia curenta
  - Surse DC-DC eficiente (> 80%)
  - Majoritatea chipurilor CMOS functioneaza pe o plaja larga de tensiuni (0.8 – 5V)
- Problema tine de algoritmi si programare:
  - Cum stiu cand trebuie sa variez tensiunea?
    - » La compilare, in hardware sau las SO sa faca treaba?

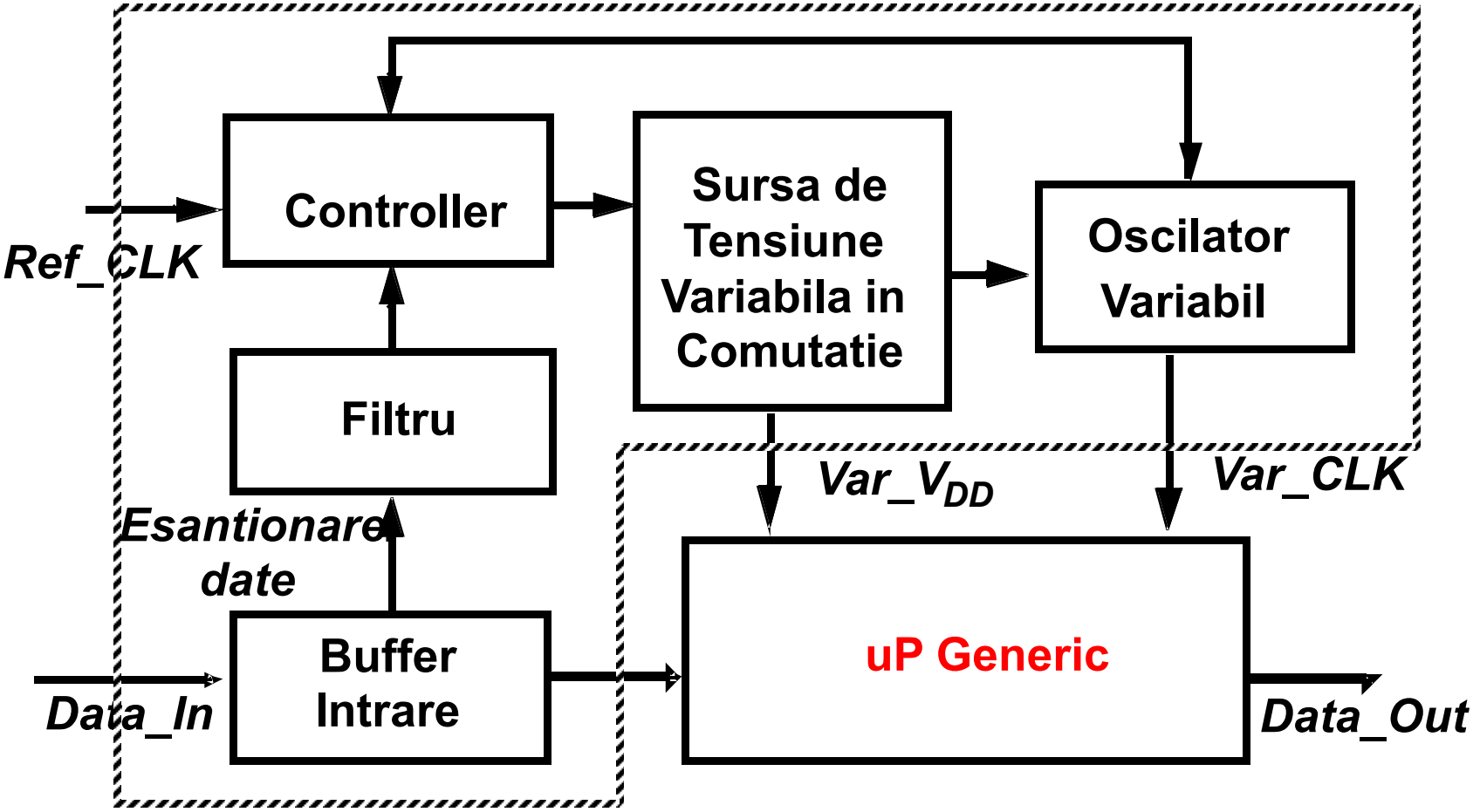
Scalare dinamica pentru sisteme asincrone



[Nielsen94]

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

Scalare dinamica pentru sisteme sincrone



Gutnik & Chandrakasan (1996 VLSI Circuits Symposium)

DVS intr-un SO

- Abordarea 1
  - Timpul impartit in cuante de 10-50 ms
  - f si V sunt modificate intr-un interval in functie de activitatea procesorului in intervalul anterior
    - » Reducere de 50% pentru un procesor la 3.3V-5V
    - » Reducere de 70% pentru un procesor la 2.2V-5V
- Abordarea 2
  - Predictia activitatii procesorului intr-un interval urmator
  - Seteaza f si V in consecinta
  - Care strategie de predictie e mai buna?

PAST

- Masoara timpii de executie pentru intervalul de timp anterior
- Presupune ca intervalul viitor va fi asemanator
- Daca intervalul anterior a fost
  - Mai mult busy → mareste viteza (f,V)
  - Mai mult idle → micsoreaza 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: Deficiente

- In cazul unor procesari in rafala, cuantele adiacente difera foarte mult la cantitatea de calcule
- Considera doar intervalul anterior
- “Netezire” deficitara din cauza istoriei pe termen scurt

FLAT

- Incearca sa netezeasca viteza la o valoare medie globala
- Prezice ca urmatorul procent de utilizare va fi f(const)

+ “Netezire” foarte buna
+ Numar mic de cicli in exces
- Predictie slaba

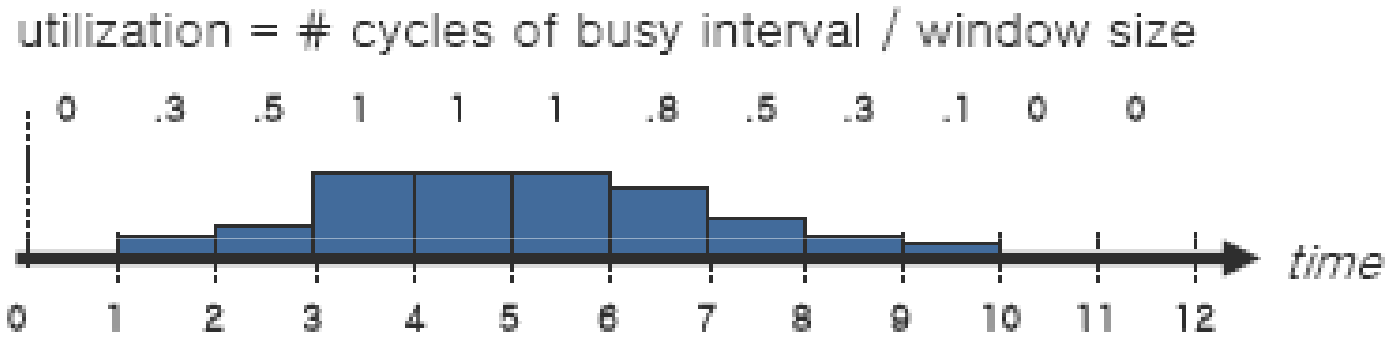
Exemplu FLAT [const = 0.3]

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 intre comportamentul pe timp lung si 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 ponderata a celor 12 intervale anterioare
- “Netezeste” la o medie globala dar ia in considerare si maximele locale

Exemplu LONG_SHORT



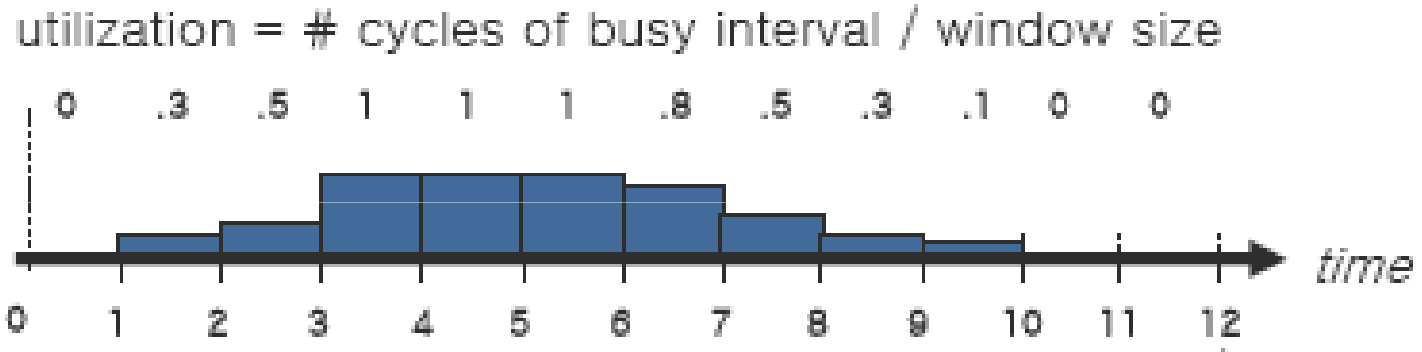
$$\frac{0+.3+.5+1+1+1+.8+.5+.3+4(.1+0+0)}{9+4(3)} = 0.276$$

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

AGED_AVERAGE

- Algoritm de netezire exponentiala
- Procesarea pentru intervalul curent este media ponderata a tuturor intervalelor anterioare
  - Ponderile sunt reduse in progresie geometrica

Exemplu AGED_AVERAGE



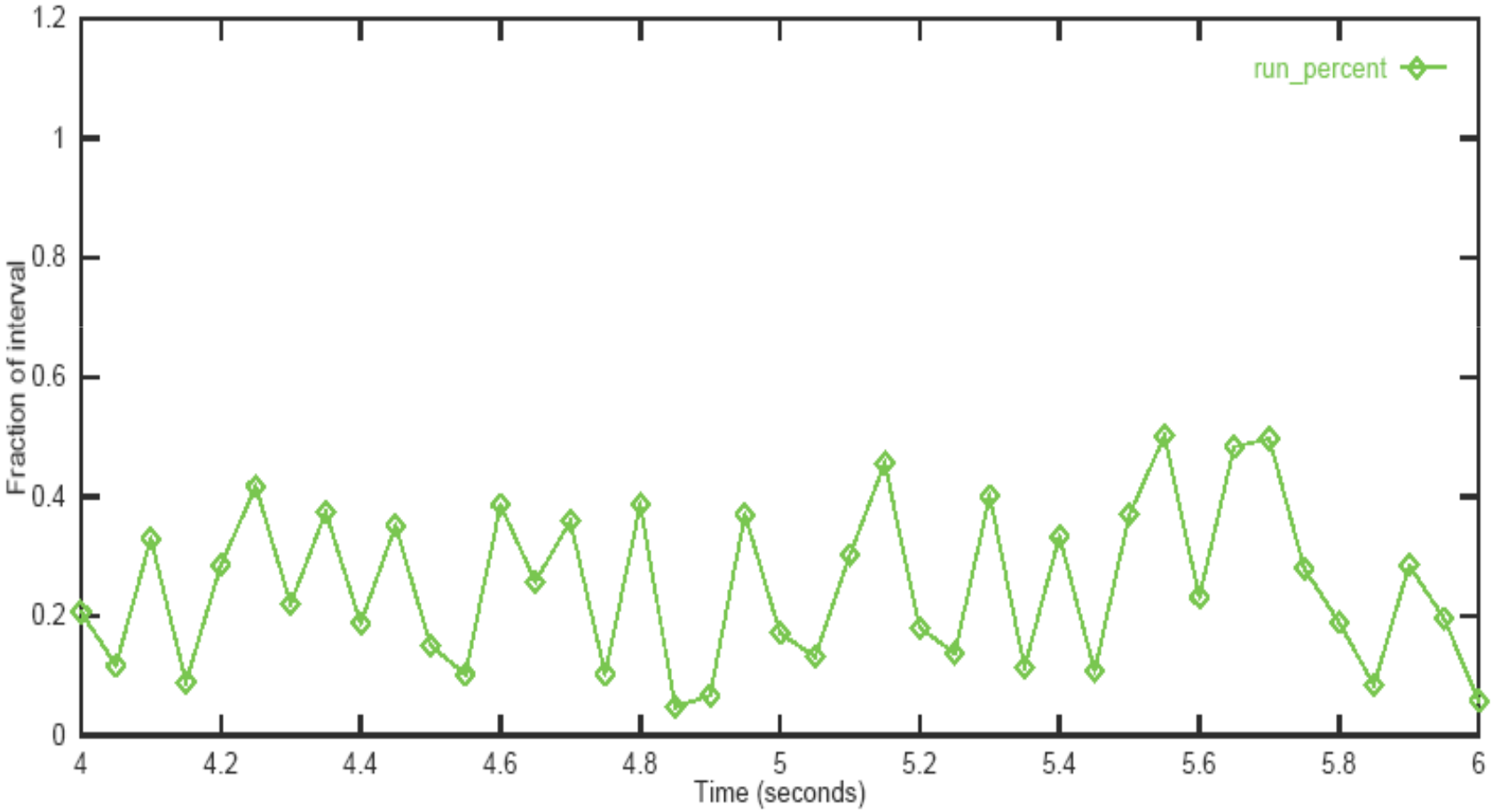
$$average = \frac{1}{3}0 + \frac{2}{9}0 + \frac{4}{27}(0.1) + \frac{8}{81}(0.3) + \cdots$$

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

CYCLE

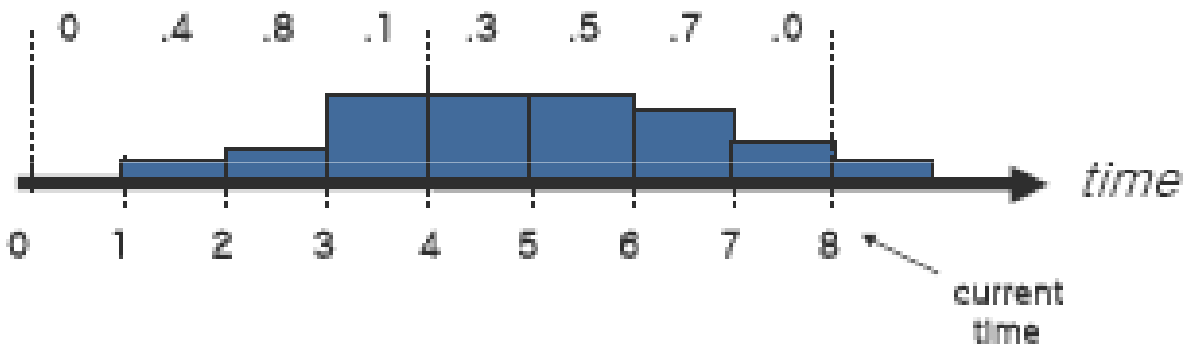
- Examineaza ultimele 16 ferestre
- Exista un ciclu de lungime X? (eroare < prag)
  - Daca da, prezice extinderea ciclului
  - Daca nu, foloseste FLAT
- Un FLAT care “ghiceste” mai bine

Motivatia 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

- Euristica
  - Procentele de crestere sunt de obicei simetrice cu cele in scadere
  - Procentele in scadere continua sa scada (-> 0)
  - Perioada sustinuta de run-percent = 1 va scadea eventual
  - Perioada sustinuta de run-percent = 0 se va mentine stabila si pe viitor

Rezultate Experimentale

- FLAT are performante mai bune decat PAST
- LONG_SHORT: low-energy, latenta mare
- AGED_AVERAGES se comporta mai bine cand este echivalent cu FLAT
- CYCLE are cele mai proaste performante cand incearca sa prezica cat 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