Power Management

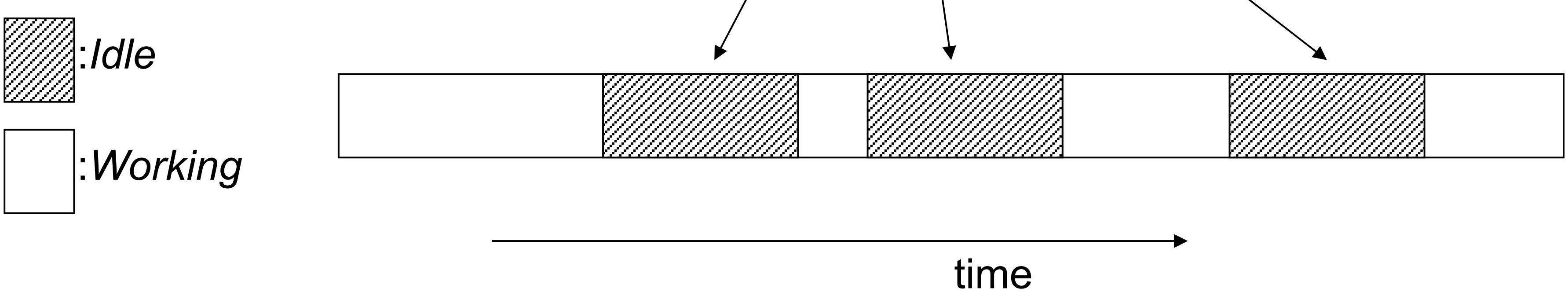
Motivation for Power Management

- Power consumption is a critical issue in system design today
  - Mobile systems face battery life issues
  - High performance systems face heating issues

$7B for powering and cooling data centers in the US

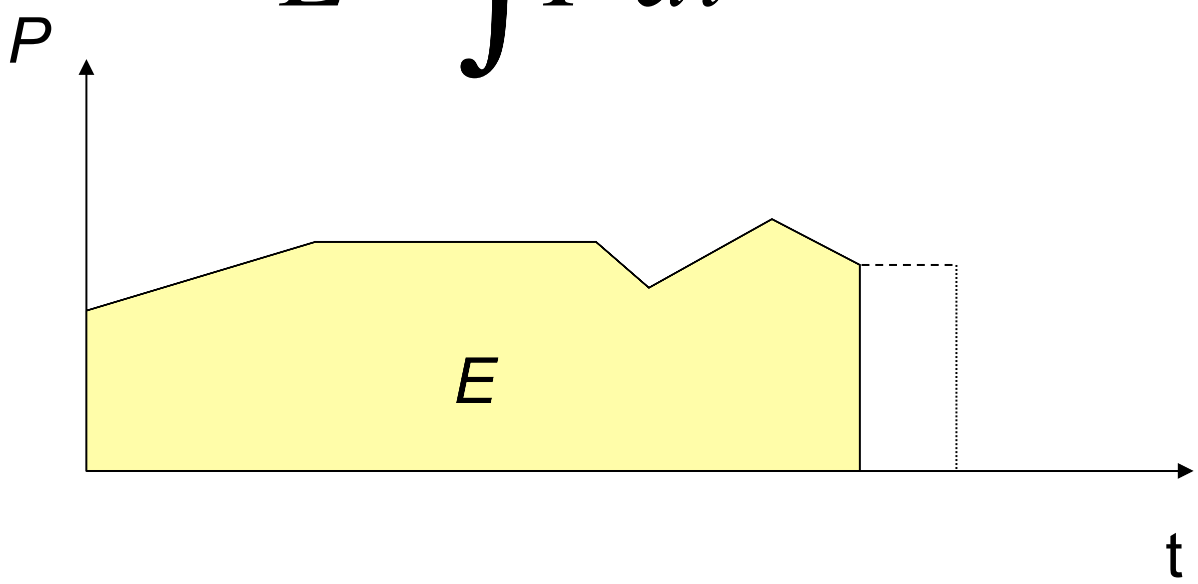
Intuition

- Systems and components are:
  - Designed to deliver *peak performance*, but …
  - Not needing peak performance most of the time
- Dynamic Power Management (DPM)
  - Shut down components during idle times
- Dynamic Voltage Frequency Scaling (DVFS)
  - Reduce voltage and frequency of components



Power and Energy Relationship

$$E = \int P \, dt$$



Low Power vs. Low Energy

- Minimizing the **power consumption** is important for
  - the design of the power supply
  - the design of voltage regulators
  - the dimensioning of interconnect
  - cooling
- 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 (solar panels, in space)
  - dependability
  - long lifetimes, low temperatures

Dynamic Voltage Frequency Scaling (DVFS)

Power consumption of CMOS circuits (ignoring leakage):

$P = \alpha\, C_L\, V_{dd}^2\, f$ with

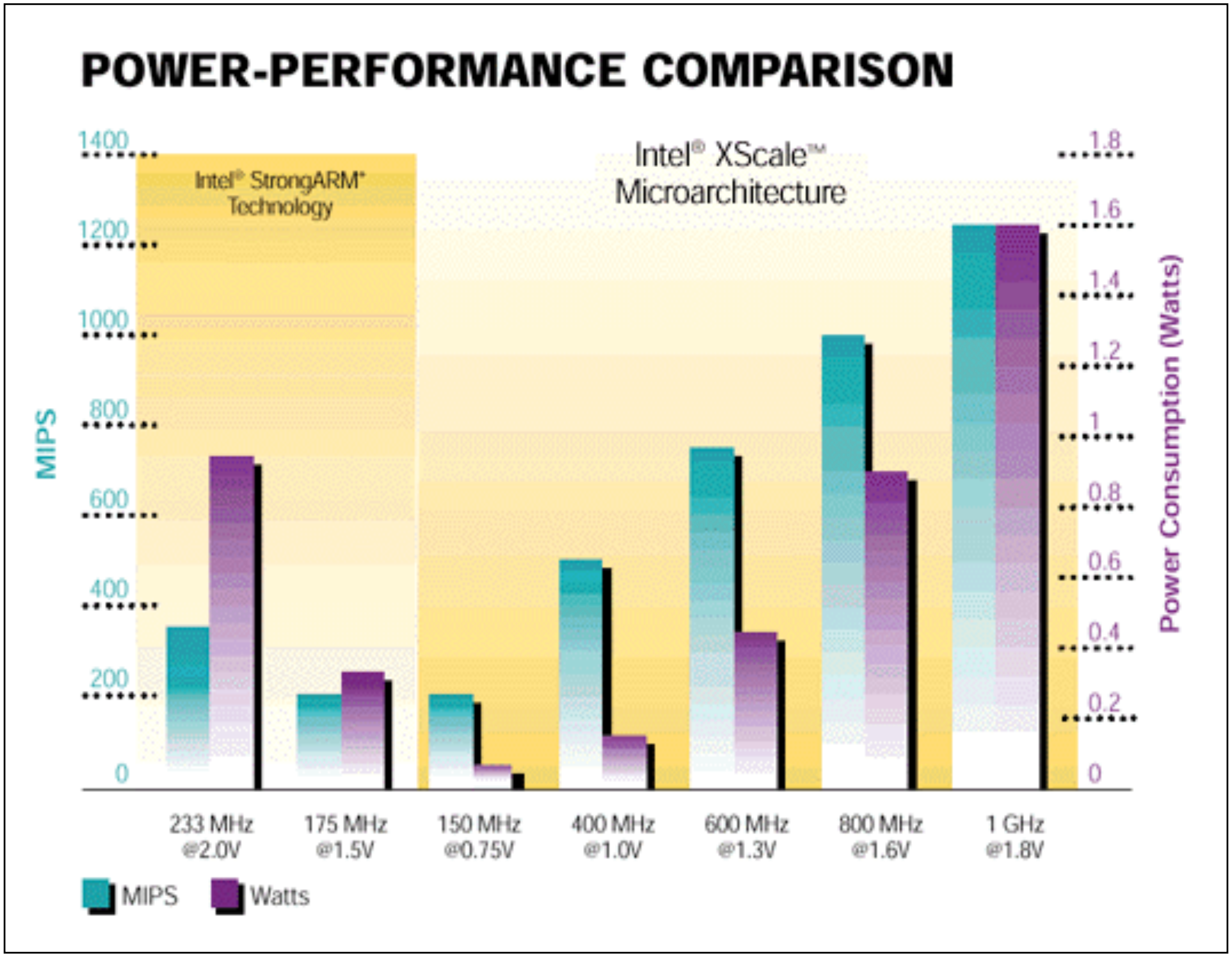
α : switching activity

C_L : load capacitance

V_{dd} : supply voltage

f : clock frequency

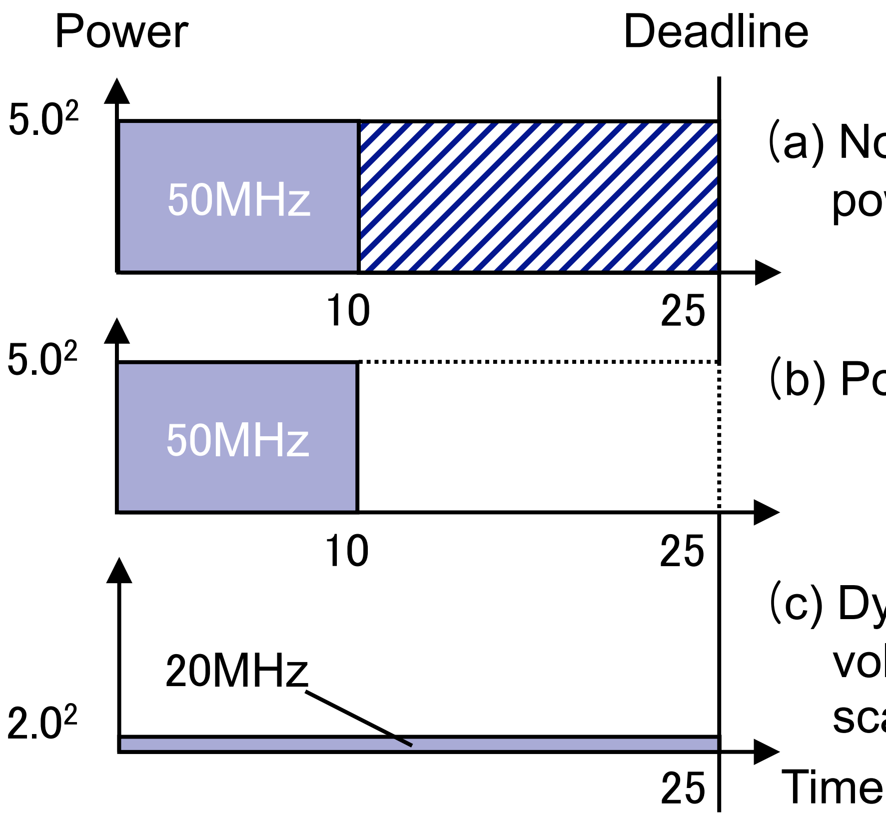
Variable-voltage/frequency



From Intel's Web Site

Basic Idea of DVFS

- $E \propto N_{cycle} \cdot V_{DD}^2$



(a) No power-down
- 12.5×10^8 cycle
- 5.0V
- 31.25J

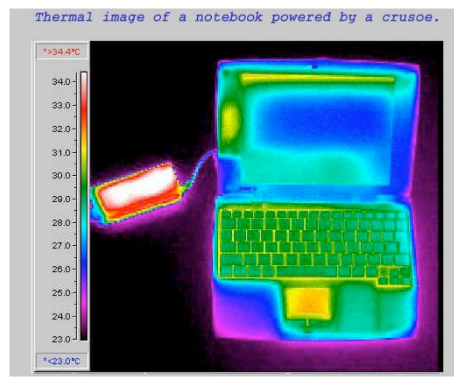
(b) Power-down
- 5×10^8 cycle
- 5.0V
- 12.5J

(c) Dynamic voltage scaling
- 5×10^8 cycle
- 2.0V
- 2.0J

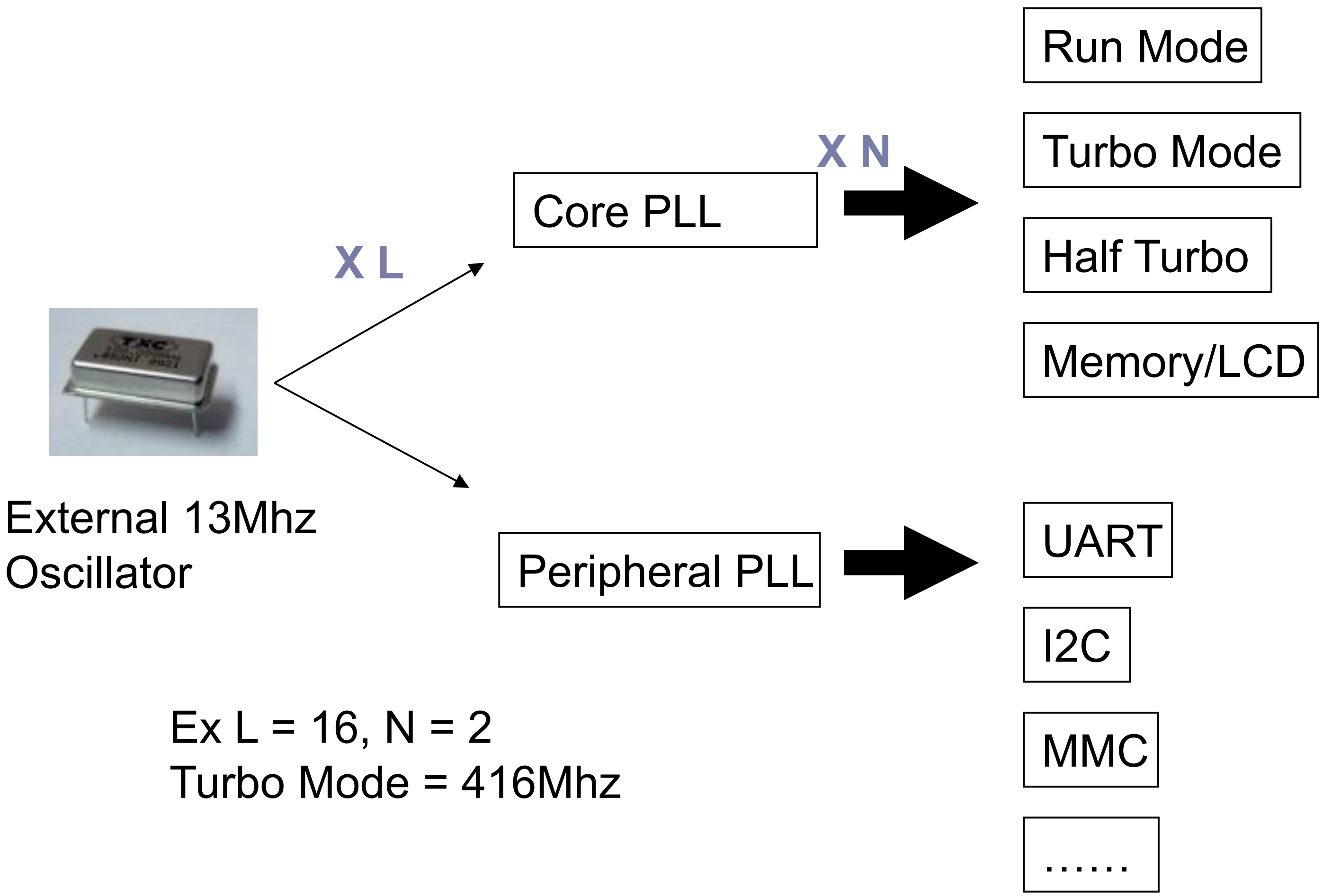
→ Slow and Steady wins the race!

Commercial DVFS Processors

- Transmeta Crusoe
- AMD K2+ (PowerNow Technology)
- Intel SpeedStep
- XScale



DVFS ::
PXA27x Clock Overview



Ex L = 16, N = 2
Turbo Mode = 416Mhz

DVFS :: How to Change?

- Reconfigure PLL
  - Modify a few registers
  - Operate in a completely new configuration
  - Take a lot of time(~ a few ms)
- Normal<->Turbo
  - Modify only CP14 register
  - Change between preset frequencies
  - Take only a little time (under us)

For detailed information, refer to PXA27x Developer’s Manual

DVFS :: PXA27x Operating Modes

Core Run Freq (MHz)	CLKCFG[T]	Core Turbo Freq (MHz)	CLKCFG[T]	CLKCFG[HT]	CCCR[L]	CCCR[2N]	System Bus (MHz)	CLKCFG[B]	CLK_MEM (MHz)	CCCR[A]	SDCLK<2:1> SDRAM Clocks (MHz)	MDREFR[KxDB2]††††	Synchronous Flash (MHz)	MDREFR[K0DB4]	MDREFR[K0DB2]	LCD (MHz)
13†	X	—	X	X	X	X	13	X	13	X	13	X	13	X	X	13 or 26††
91†††	0	—	—	0	7	2	45	0	91	0	45	1	22.5	1	1	91
104	0	104	1	0	8	2	104	1	104	1	104	0	52	0	1	52
156	0	156	1	1	8	6	104	1	104	1	104	0	52	0	1	52
208	0	208	1	0	16	2	104	0	104	0	104	0	52	1	X	104
208	0	208	1	0	16	2	208	1	208	1	104	1	52	1	X	104
208	0	312	1	0	16	3	104	0	104	0	104	0	52	1	X	104
208	0	312	1	0	16	3	208	1	208	1	104	1	52	1	X	104
208	0	416	1	0	16	4	208	1	208	1	104	1	52	1	X	104
208	0	520	1	0	16	5	208	1	208	1	104	1	52	1	X	104
208	0	624†††††	1	0	16	6	208	1	208	1	104	1	52	1	X	104

DVFS :: Usage

- dvfm
  - voltage and frequency scaling utility
- Usage
  - # dvfm l_value 2n_value fast_bus_mode turbo_mode mem_clk_conf
- Examples
  - # dvfm 16 5 1 2 1
    - 520Mhz Turbo Mode
  - # dvfm 16 3 1 2 1
    - 312Mhz Normal Mode

Xscale DVFS Settings

V-f setting (MHz/V)	Active Power (mW)	Idle Power (mW)	dvfm setting
624/1.5	925	260	dvfm 16 6 1 2 1
520/1.5	747	222	dvfm 16 5 1 2 1
416/1.4	570	186	dvfm 16 4 1 2 1
312/1.3	390	154	dvfm 16 3 1 2 1
208/1.2	279	129	dvfm 16 2 1 2 1
104/1.1	116	64	dvfm 8 2 1 2 1

Successful DVFS Technique

1. Understand workload variations of your target
2. Devise efficient ways to detect them
3. Devise efficient ways to utilize the detected workload variations using available H/W supports

Non Real-Time Jobs

- Non Real-Time Jobs
  - No timing constraints
  - No periodic executions
  - Unknown execution time

It is hard to predict the future workload!!♪

DVFS for Non Real-Time Jobs

- Basic Approach:
  - Predict workload based on history information
  - Usually based on *some variations of interval scheduler*
    - PAST, FLAT
    - LONG_SHORT, AGED_AVERAGE
    - CYCLE, PATTERN, PEAK

Key Question

How can we predict the future workload?♪

- Based on long term history:♪
  - ♪Hard to adapt quickly for the changed workload♪

♪

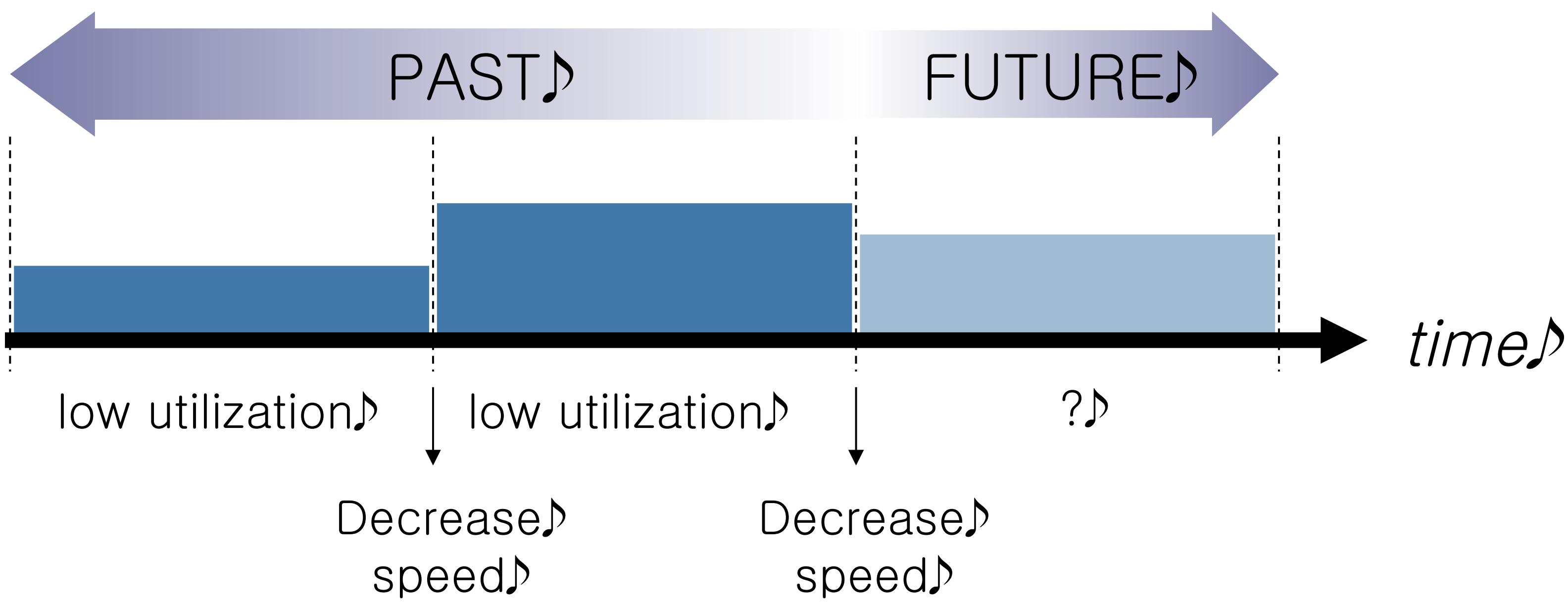
- Based on short term history:♪
  - ♪Too many clock/voltage changes♪

PAST

- Looking a fixed window into the past
- Assume the next window will be like the previous one
- If the past window was
  - mostly busy $\Rightarrow$ increase speed
  - mostly idle $\Rightarrow$ decrease speed

Example: PAST

$$\text{Utilization} = \frac{\text{busy time}}{\text{window size}}$$

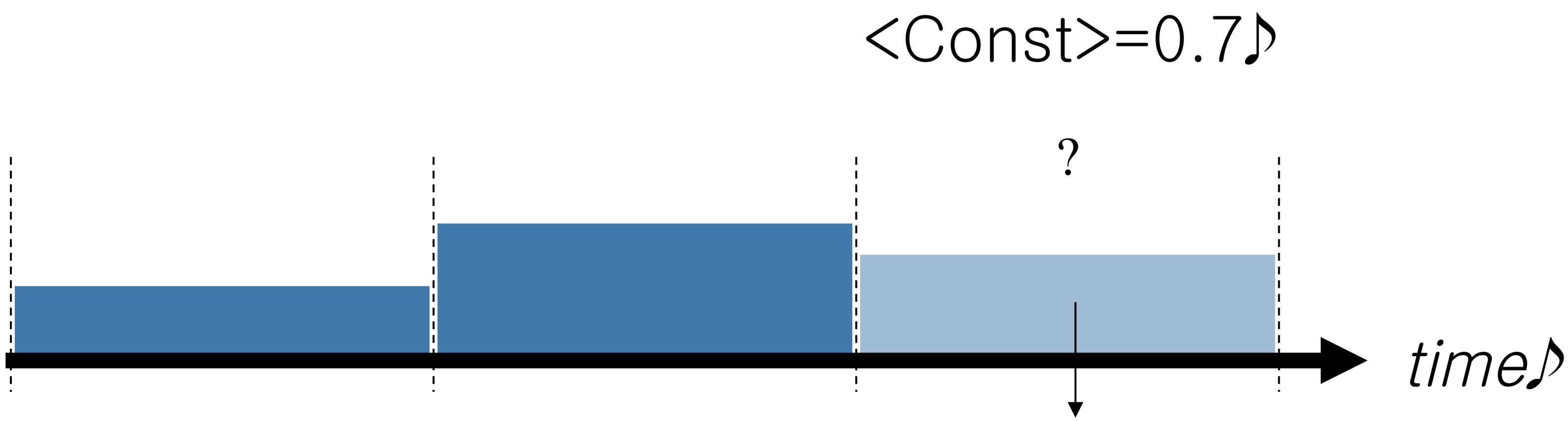


FLAT

- Try to smooth speed to a global average

- Make the utilization of next window=<const>
  - Set speed fast enough to complete the predicted new work being pushed into the coming window

Example: FLAT



Increase/Decrease the speed♪
the next utilization to be 0.7♪

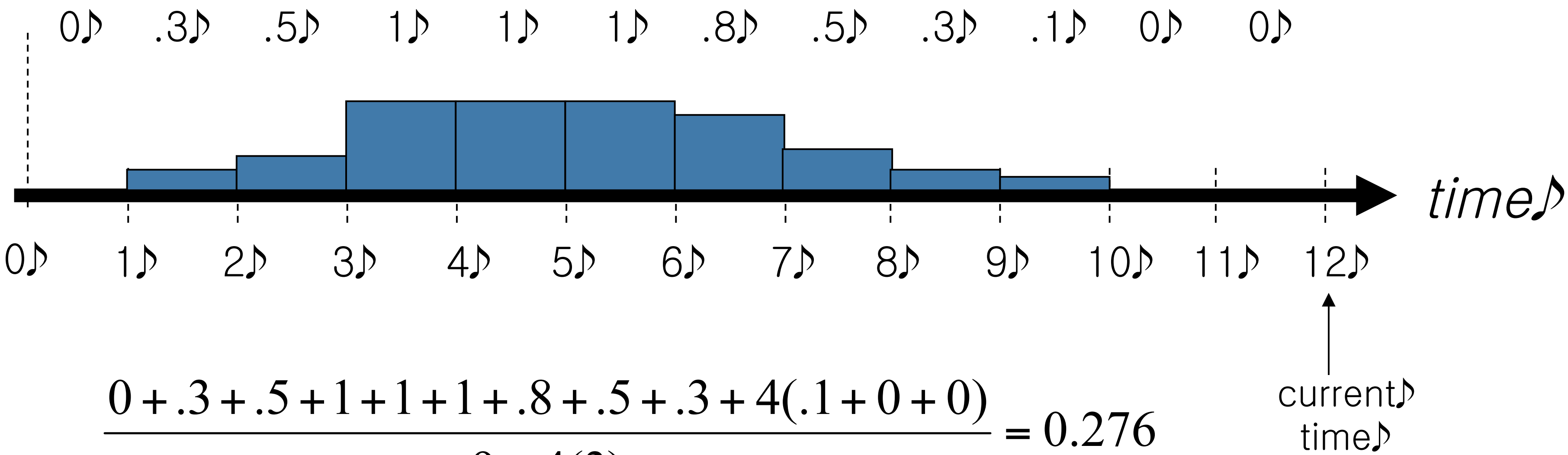
LONG-SHORT

- Look up the last 12 windows
  - Short-term past : 3 most recent windows
  - Long-term past : the remaining windows

- Workload Prediction
  - the utilization of next window will be a weighted average of these 12 windows' utilizations

Example: LONG-SHORT

utilization = # cycles of busy interval / window size♪



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

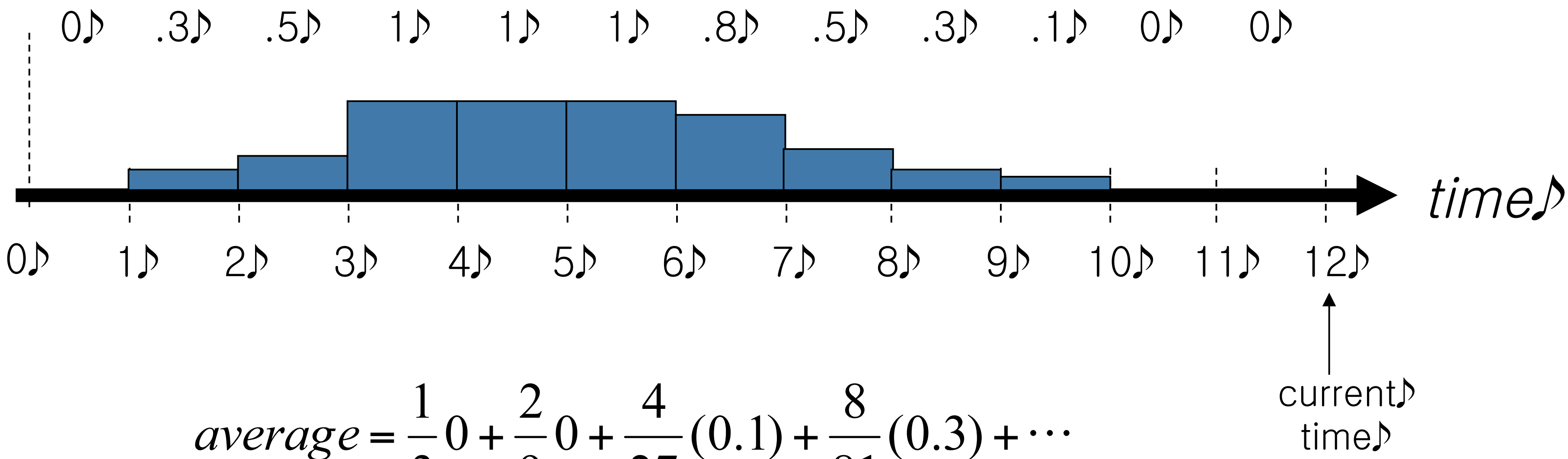
$$f_{clk} = 0.276 \times f_{\max}$$

AGED-AVERAGE

- Employs an exponential-smoothing method
- Workload Prediction
  - The utilization of next window will be a weighted average of all previous windows' utilizations
    - geometrically reduce the weight

Example: AGED_AVERAGE

utilization = # cycles of busy interval / window size♪



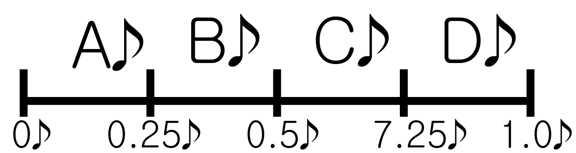
$$average = \frac{1}{3}0 + \frac{2}{9}0 + \frac{4}{27}(0.1) + \frac{8}{81}(0.3) + \cdots$$

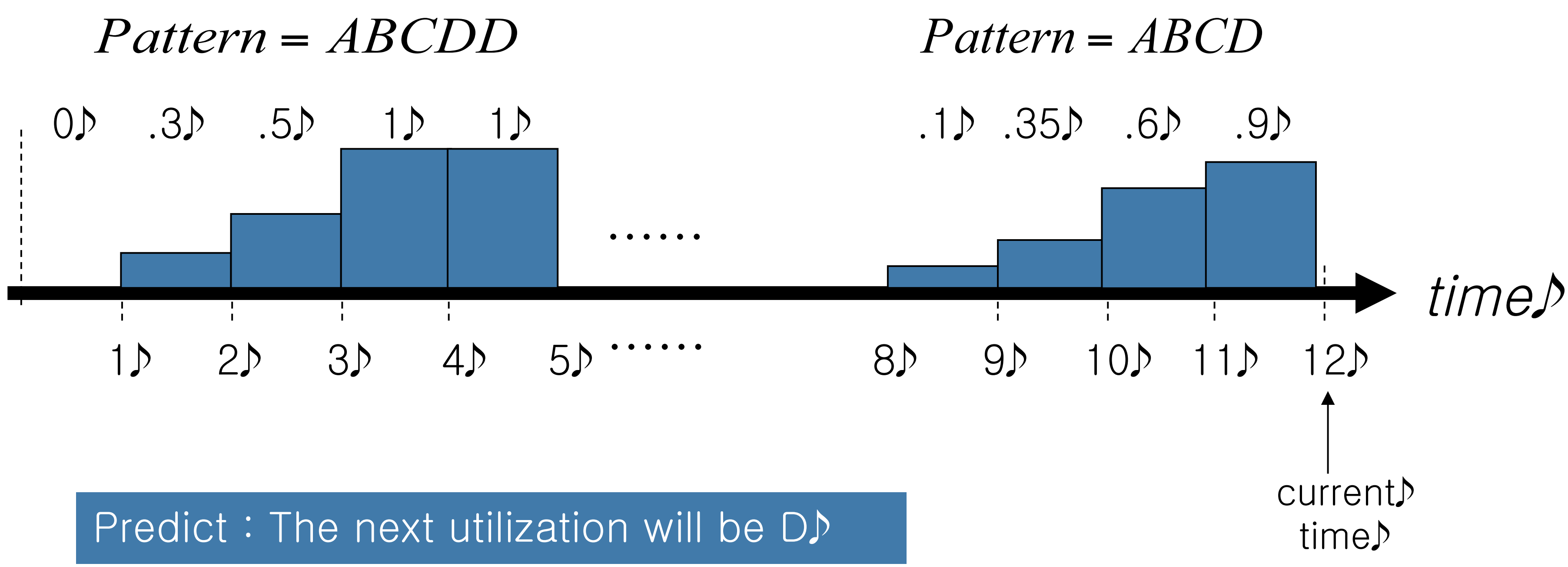
$$f_{clk} = average \times f_{\max}$$

PATTERN

- Workload Prediction
  - Convert the n-most recent windows' utilizations into a pattern in alphabet {A, B, C, D}.
  - Find the same pattern in the past
  - Use the pattern to predict utilization

Example: PATTERN





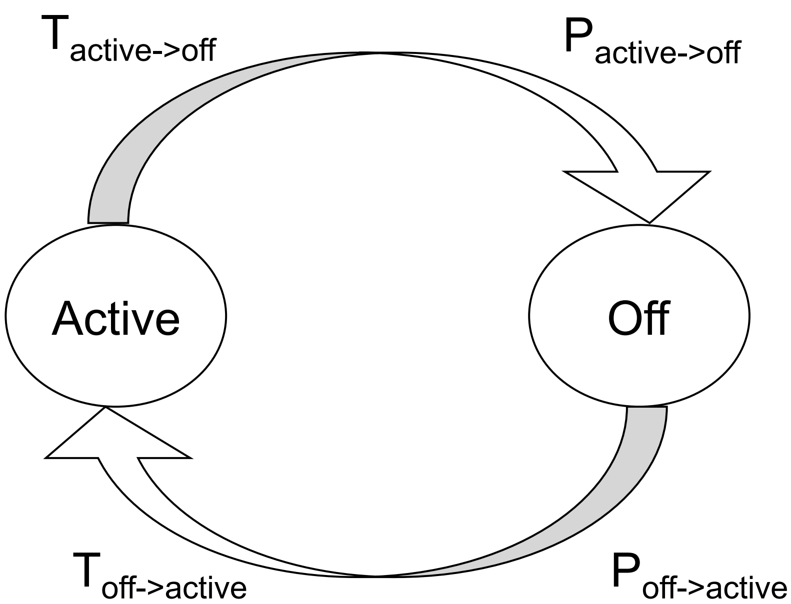
Predict : The next utilization will be D♪

Dynamic Power Management

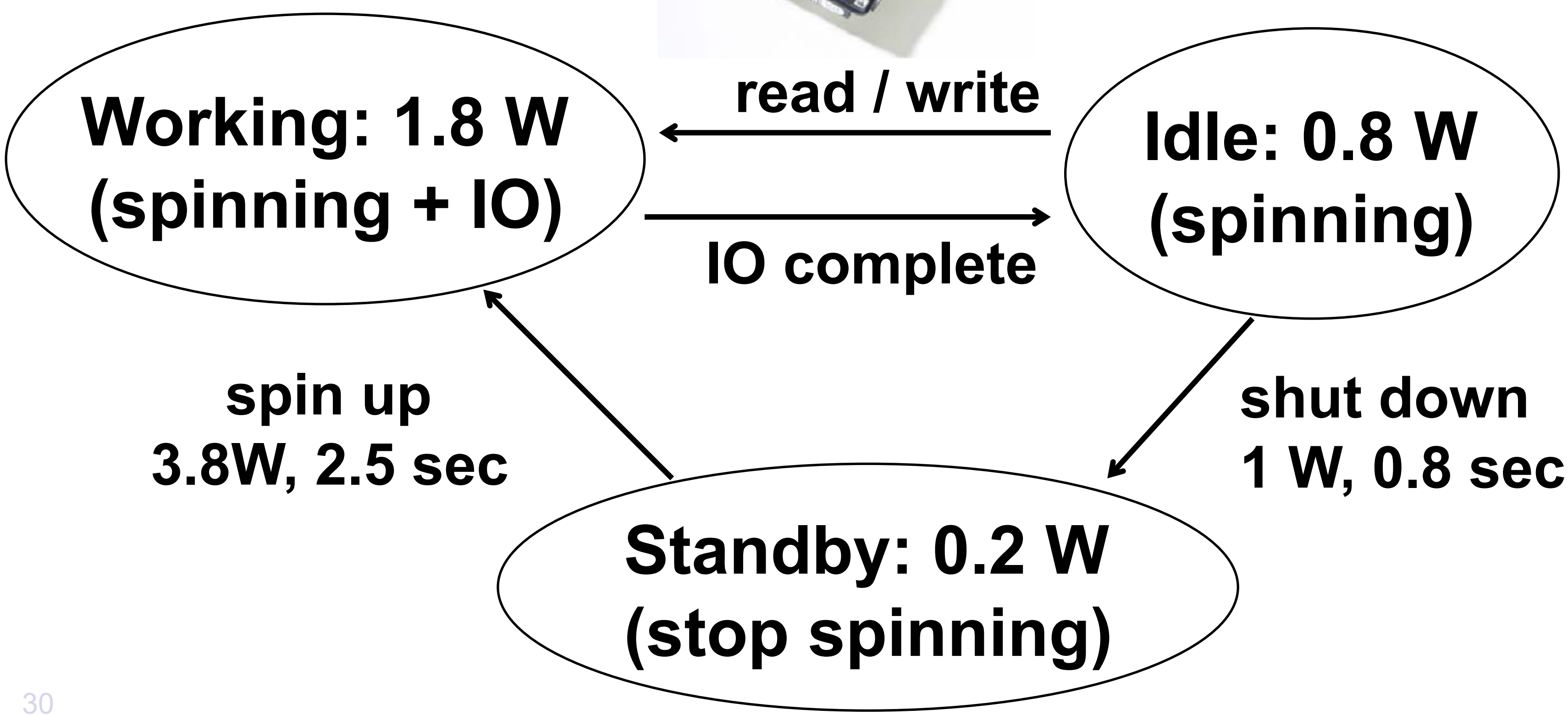
- Power manageable components support multiple power states. Eg:
  - Active
  - Off

$$P_{tr} = P_{off\text{->}active} + P_{active\text{->}off}$$

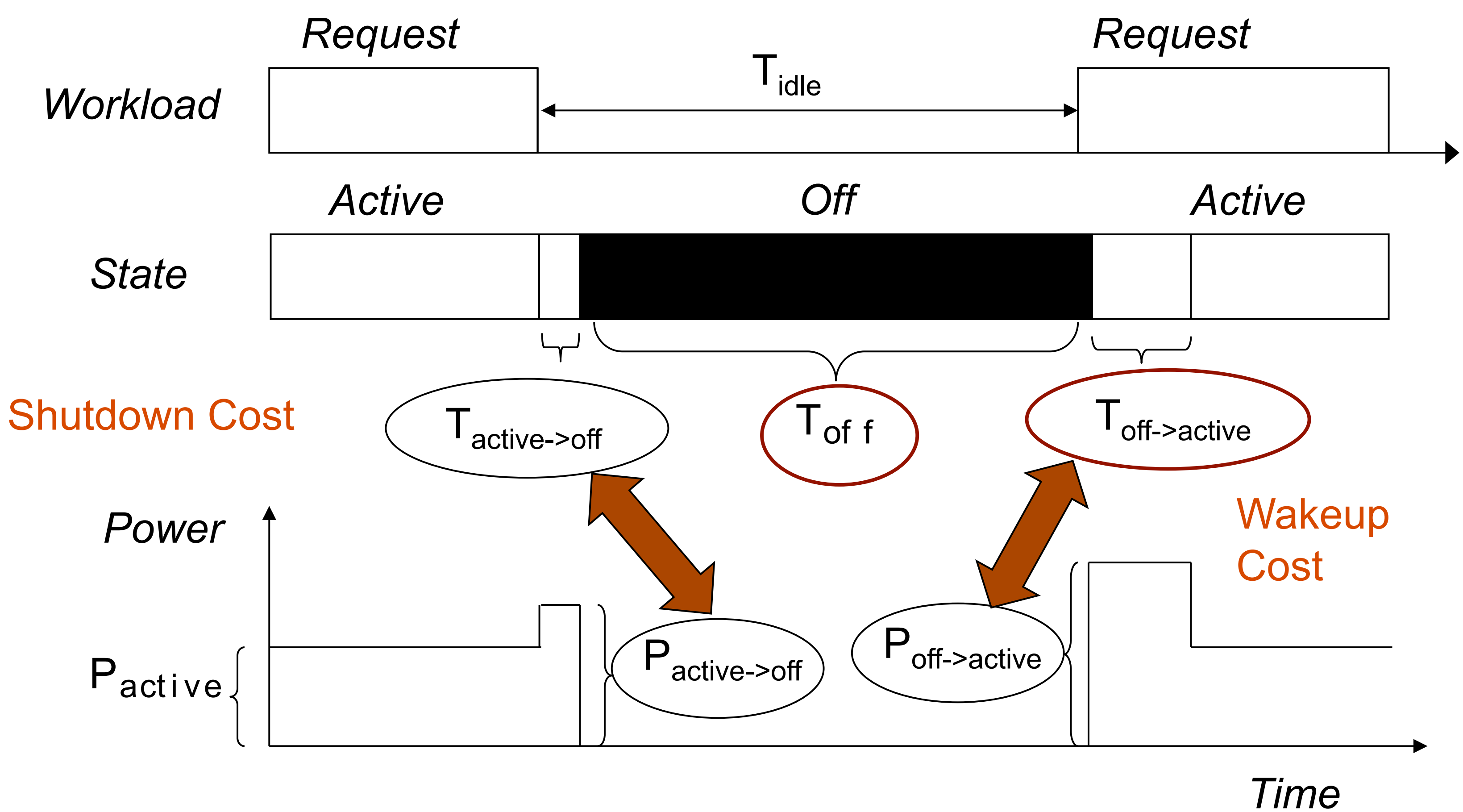
$$T_{tr} = T_{off\text{->}active} + T_{active\text{->}off}$$



Power Manageable Components



Energy/Performance Tradeoff for DPM

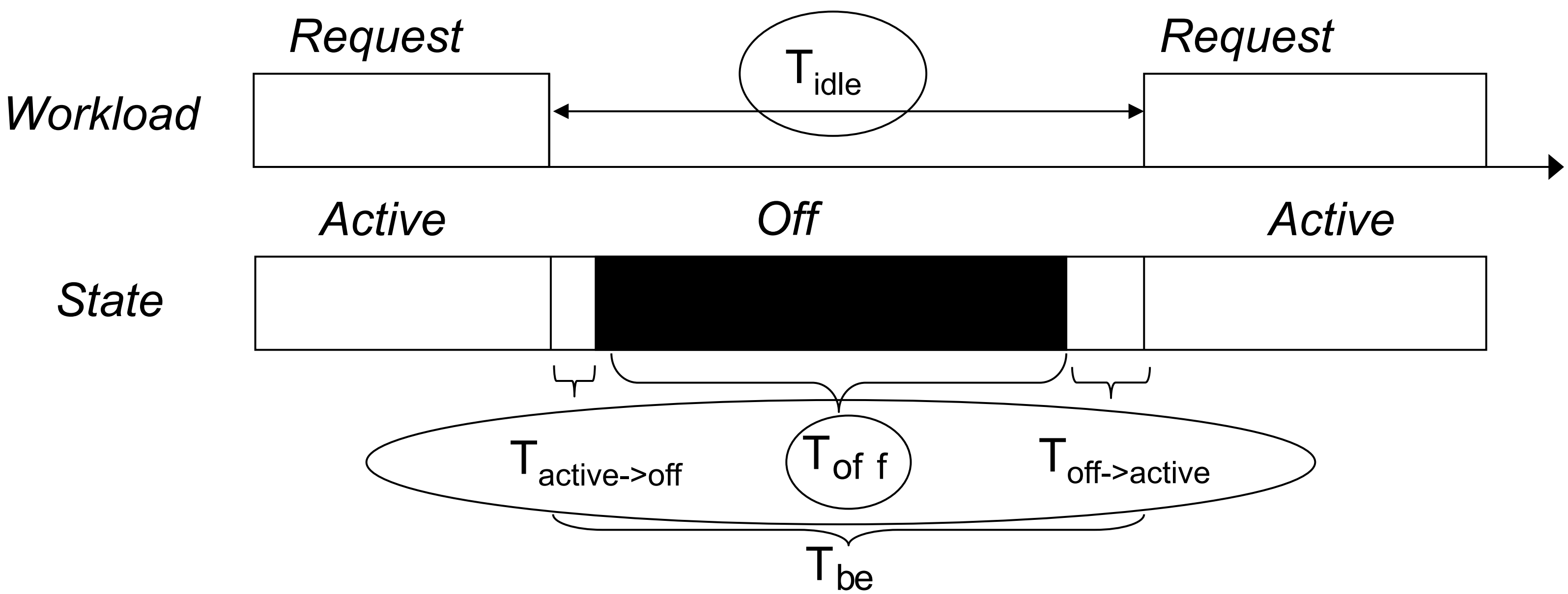


Break Even Time

- Minimum idle time for amortizing the cost of component shutdown

$$T_{off}P_{off} + T_{tr}P_{tr} = T_{idle}P_{active}$$

$$T_{be} = T_{idle} = T_{tr} + T_{off}$$



DPM Applicability

- If idle period length < T_{be}
  - Not possible to save energy by turning off
- Need accurate estimation of upcoming idle periods:
  - *Underestimation*: potential energy savings lost
  - *Overestimation*: performance delay + energy loss
- *Challenge*: Manage energy/performance tradeoff

DPM Policies

- Control procedures that take DPM decisions
- Can be implemented in hardware or software
  - Software offers higher degree of configurability for complex systems

Goal: Maximize energy savings while minimizing performance delay

Classification of DPM Policies

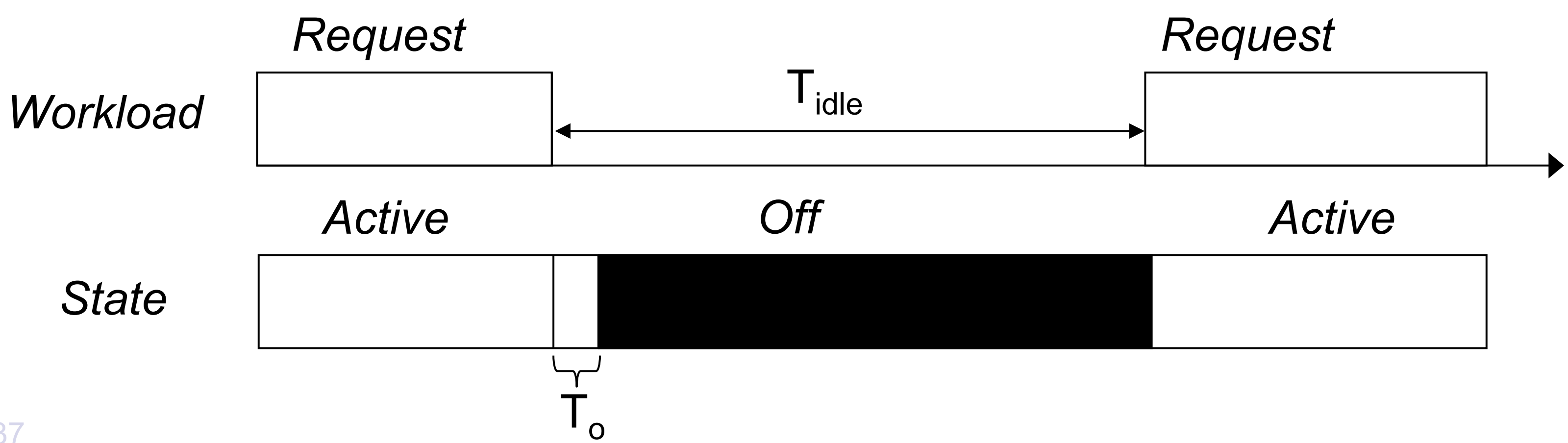
- Timeout Policies
- Predictive Policies
- Stochastic Policies
- Hybrid Policies

Classification of DPM Policies

- Timeout Policies
- Predictive Policies
- Stochastic Policies
- Hybrid Policies

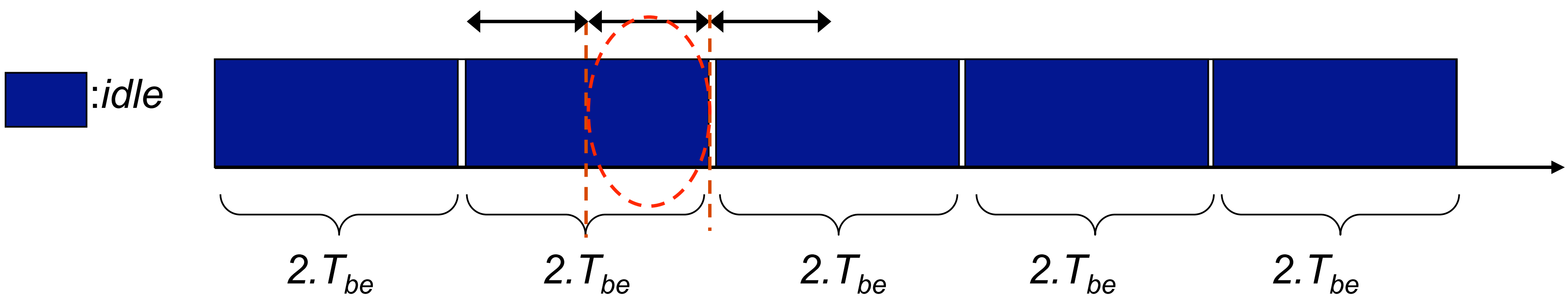
Timeout Policies

- Use elapsed idle time to predict the total idle period duration
  - Assumption:
    - $P(T_{idle} > (T_o + T_{be}) \mid T_{idle} > T_o) \approx 1$
  - T_o is referred to as the timeout
    - Can be fixed or adaptive



Fixed Timeout Policies

- $T_o = T_{be}$ is 2-competitive (Karlin et al, SODA'90)
  - Energy consumption in worst case is twice that of *oracle policy*



In this use case energy consumption of Karlin's algorithm would be twice that of oracle policy

Energy consumption (oracle) = $P_{tr}.T_{tr}$

If $T_{be} = T_{tr}$, oracle policy would sleep in first half and transition to active in second half T_{tr} and $P_{tr} > P_{active}$

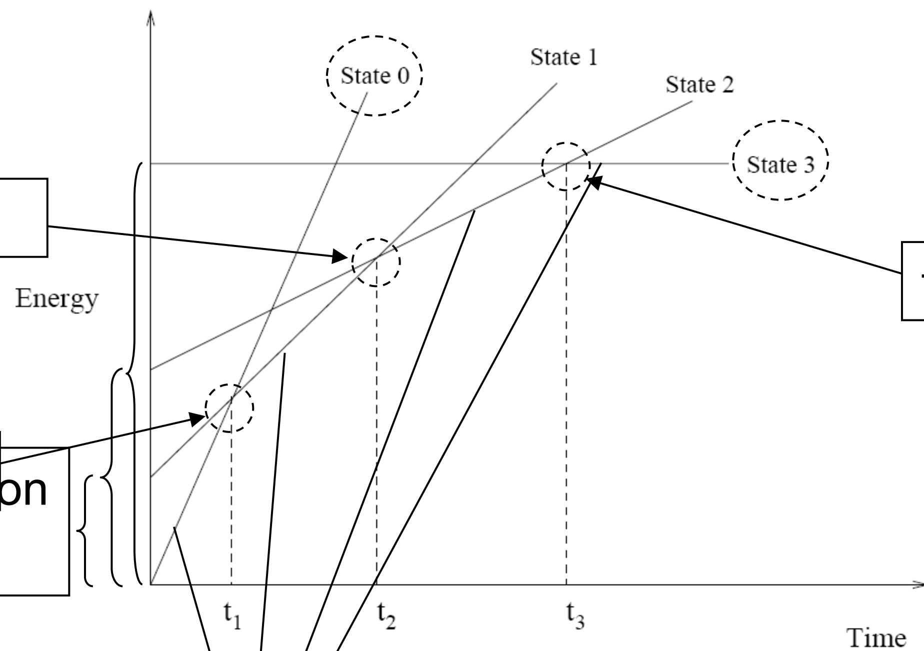
Holds true for, $T_{be} > T_{tr}$

Energy consumption (Karlin) = $P_{active}.T_{tr} + P_{tr}.T_{tr} = 2P_{tr}.T_{tr}$

Fixed Timeout Policy

- However, the approach limited to components with 2 power states
- *Irani et al, TECS'03* extend this work for components with multiple power states
  - Define a sets of timeouts: one for each state
  - Enter the low power state as its respective timeout expires

Fixed Timeout Policy



t2: S1 -> S2

t3: S2 -> S3

t1: S0 -> S1

Intercept: transition cost

Slope: power consumption rate

This algorithm is 2-competitive

(Irani et al, TECS’03)

Adaptive Timeout Policy

- Dynamically adjust T_o based on their observation of the workload
- *Douglis et al, USENIX'95* propose several heuristics to adapt timeout. Eg:
  - Initialize T^1_o to T_{be}
  - Observe length of idle period T_{idle}
  - If $T^n_{idle} > (T^n_o + T_{be})$, then $T^{n+1}_o = T^n_o - x$
  - Else, $T^{n+1}_o = T^n_o + x$
- Set floor and ceiling values to avoid getting too aggressive or conservative

Timeout Policies

Advantages

- Extremely simple to implement
- Safety (in terms of performance delay) can be easily improved by increasing T_o

Disadvantages

- Waste energy while waiting for T_o to expire
- Heuristic in nature: no guarantee on energy savings or performance delay
- Always incur performance penalty on wakeup: no mechanism to wake before request arrival

Classification of DPM Policies

- Timeout Policies
- Predictive Policies
- Stochastic Policies
- Hybrid Policies

Predictive Policies

- Take the DPM decision as soon as the idle period begins by predicting the length of upcoming idle period
  - If $T_{pred} > T_{be}$, perform shutdown
  - Else, stay awake
- Addresses the *energy wastage* issue of timeout policies while waiting for timeout to expire

Predictive Policies

- Hwang et al propose an online predictive policy

  - Uses exponential average of previous idle period lengths to perform prediction:

    $$T^{n}_{pred} = \alpha T^{n-1}_{idle} + (1-\alpha)\, T^{n-1}_{pred}$$

  - Value of α controls the tradeoff between recent and past history

Predictive Policies

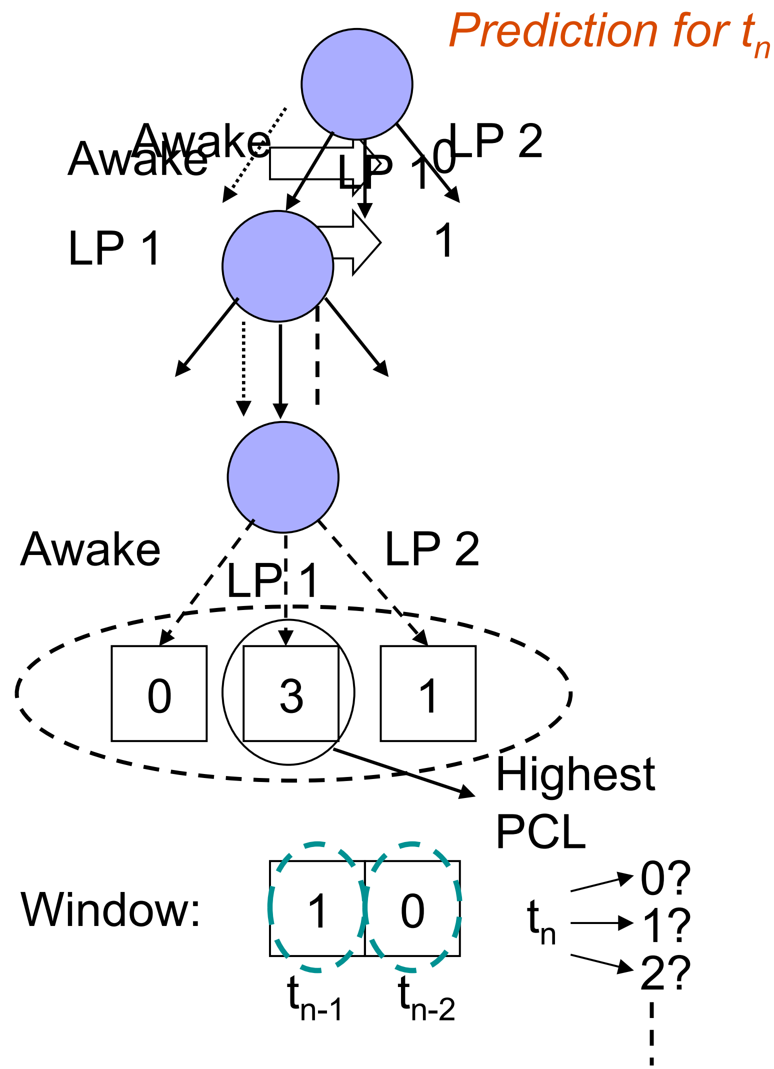
- Chung et al use *adaptive learning trees*
- Capable of managing multiple power states

○ Transforms sequence of optimal decisions in previous idle periods into discrete events

○ Maintains a window of previous events

○ Real time learning and updates PCL and tree structure

Adaptive Learning Tree



Predictive Policies

- Advantages
  - More aggressive than timeout policies
- Disadvantages
  - Depend a lot on correlation between past and future events
  - Tend to be aggressive in shutdown and hence higher performance latency
  - Heuristic with no performance guarantees

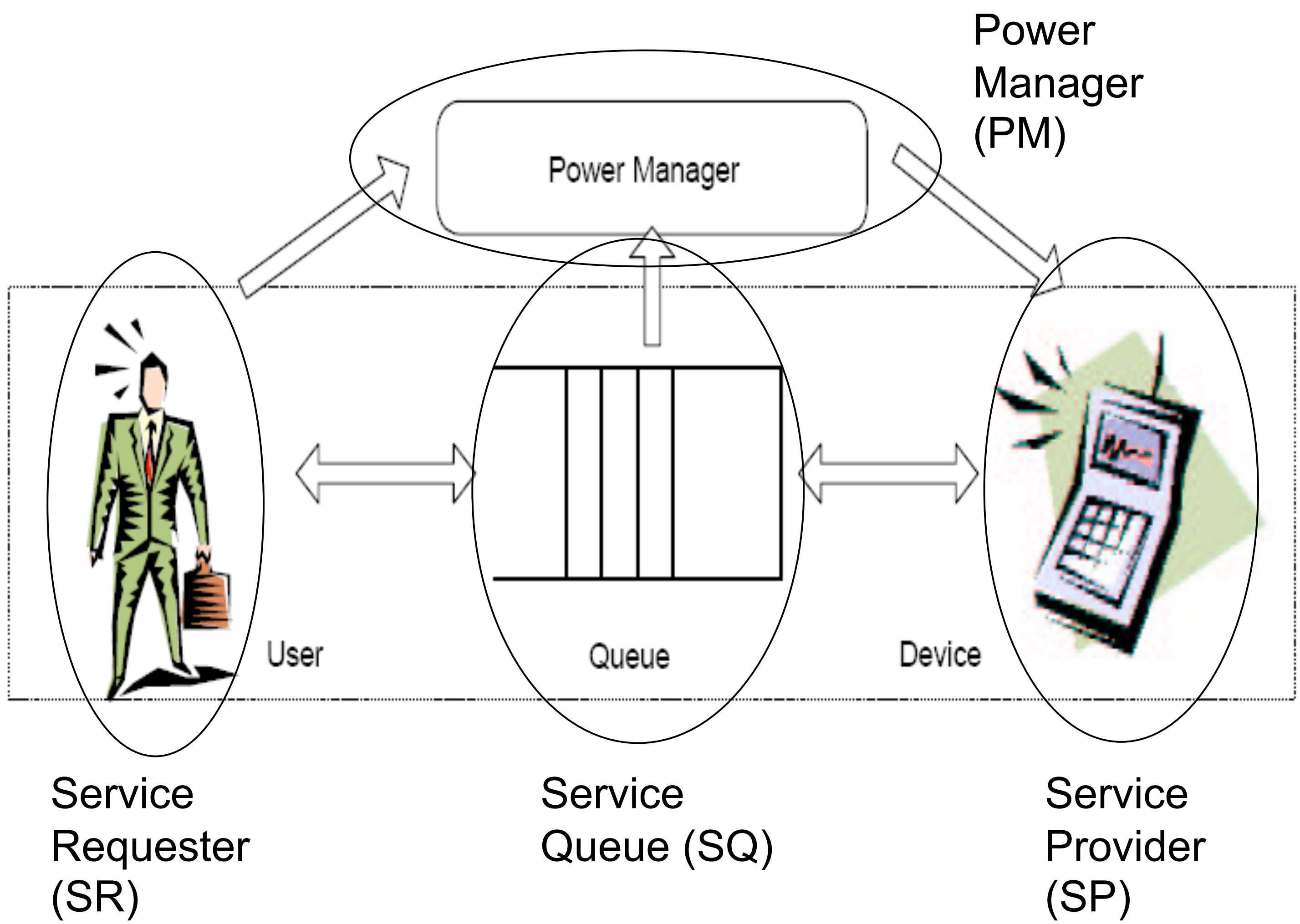
Classification of DPM Policies

- Timeout Policies
- Predictive Policies
- Stochastic Policies
- Hybrid Policies

Stochastic Policies

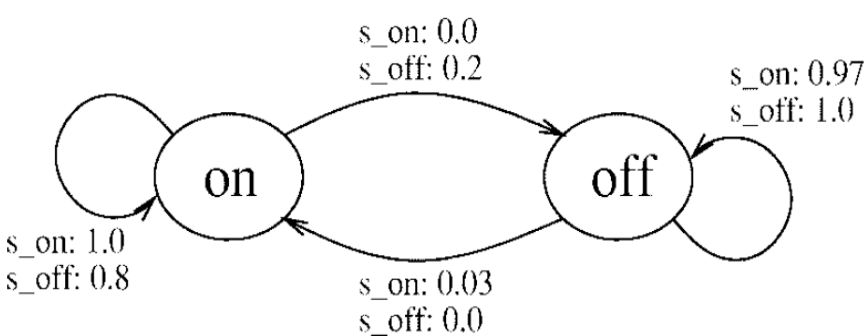
- Try to derive optimal policies for the given power and performance constraints
  - Referred to as *policy optimization*
- Model the system and workload as stochastic processes
- Policy optimization reduces to a stochastic optimization problem

System Model

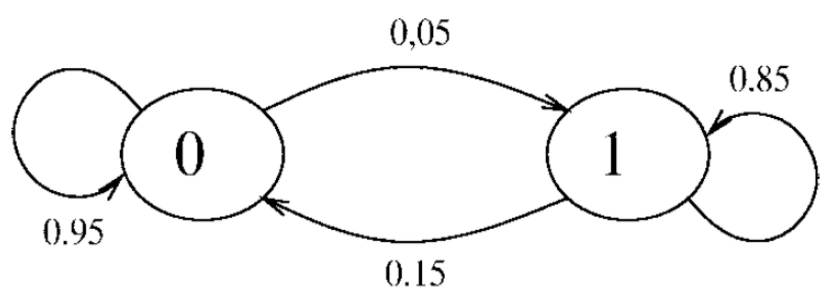


Discrete Time Markov Model (Paleologo et al, DAC'98)

- Models the system using markov decision processes
- Discrete time setting (time slots)
  - System composition of SP, SR, SQ models
  - Defines cost metrics for each state and command pair



SP
(Paleologo et al, DAC'98)



SR

Power Manager

- PM observes the system state (S X R X Q) every time slot and issues command ‘a’
- Markovian stationary policies optimal for this system model
- Policy derived through policy optimization under given energy/ performance constraints

$$
\mathbf{M}_\pi = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{array}
\begin{array}{c} \begin{array}{cc} s\_on & s\_off \end{array} \\
\begin{pmatrix} 0.4 & 0.6 \\ 0.2 & 0.8 \\ 0.5 & 0.5 \\ 1.0 & 0.0 \\ 0.4 & 0.6 \\ 0.8 & 0.2 \\ 0.8 & 0.2 \\ 1.0 & 0.0 \end{pmatrix} \end{array}
$$

(Benini et al, TCAD’99)

Discrete Time Model

- Advantages:
  - Optimal policy => no longer heuristic
- Disadvantages
  - Discrete time => high overhead
  - Optimal only if the modeling assumptions hold
    - Geometric state transition times
    - Stationary workload

Continuous Time Markov Model (Qui et al, TCAD'01)

- Model the system in continuous time space using CTMDP
- Cost associated with commands and states
- Policy optimization done to derive optimal policy
- Event driven policy, i.e. when system state changes
  - State transition times exponential

Continuous Time Model

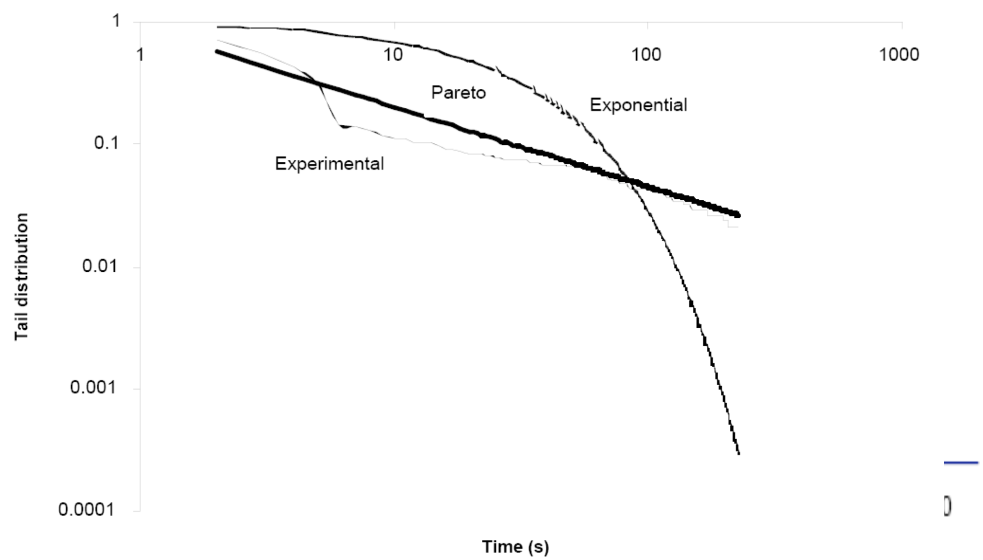
- Advantages:
  - Optimal policy
  - Event driven => lesser overhead
- Disadvantages
  - Optimal only if the modeling assumptions hold
    - Exponential state transition times
    - Stationary workload

Workload and Device Characteristics Simunic et al, TCAD’01

- Request inter-arrival times -> depends on state of SP:
  - If active: exponential
  - If idle: pareto

Important from DPM perspective

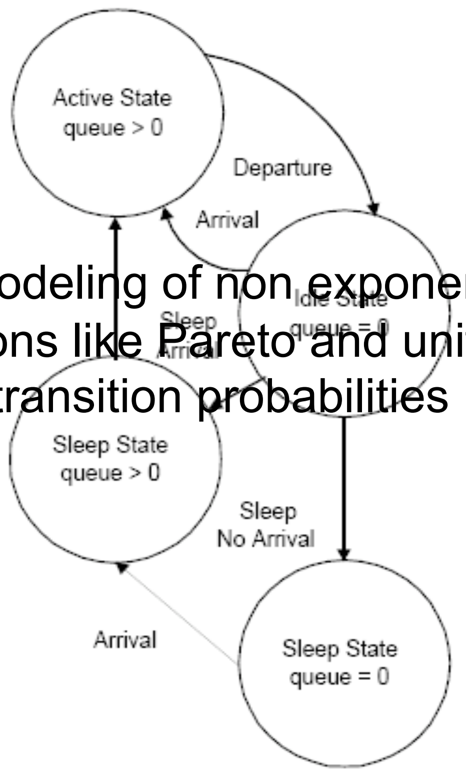
- SP request service times:
  - Follows exponential distribution
- State transition times of SP:
  - Follows uniform distribution



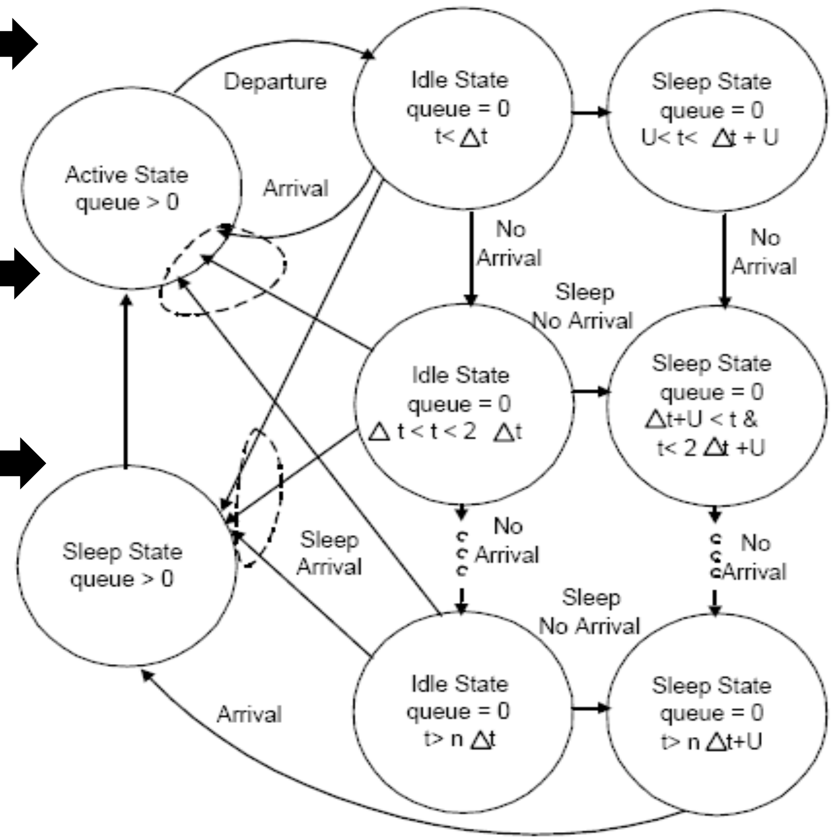
(Simunic et al, TCAD’01)

TISMDP Model

Allows modeling of non exponential distributions like Pareto and uniform for state transition probabilities



SMDP Model



Time Indexed SMDP Model

(Simunic et al, TCAD'01)

TISMDP Model

- Advantages:
  - Optimal and event driven policy
  - System model based on real world device and workload characteristics
- Disadvantages
  - Sub-optimal for non stationary workloads
    - Stationary workload assumption

Classification of DPM Policies

- Timeout Policies
- Predictive Policies
- Stochastic Policies
- Hybrid Policies

Hybrid Policies

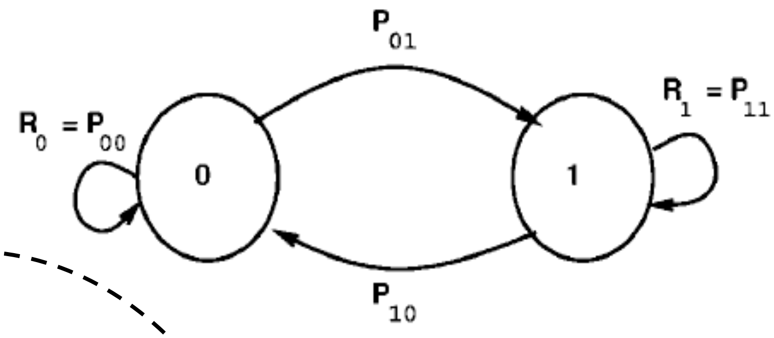
- Extension of policies from previous three classes
- Ideas:
  - Stochastic policies for non stationary workloads
  - Perform selection among multiple DPM policies

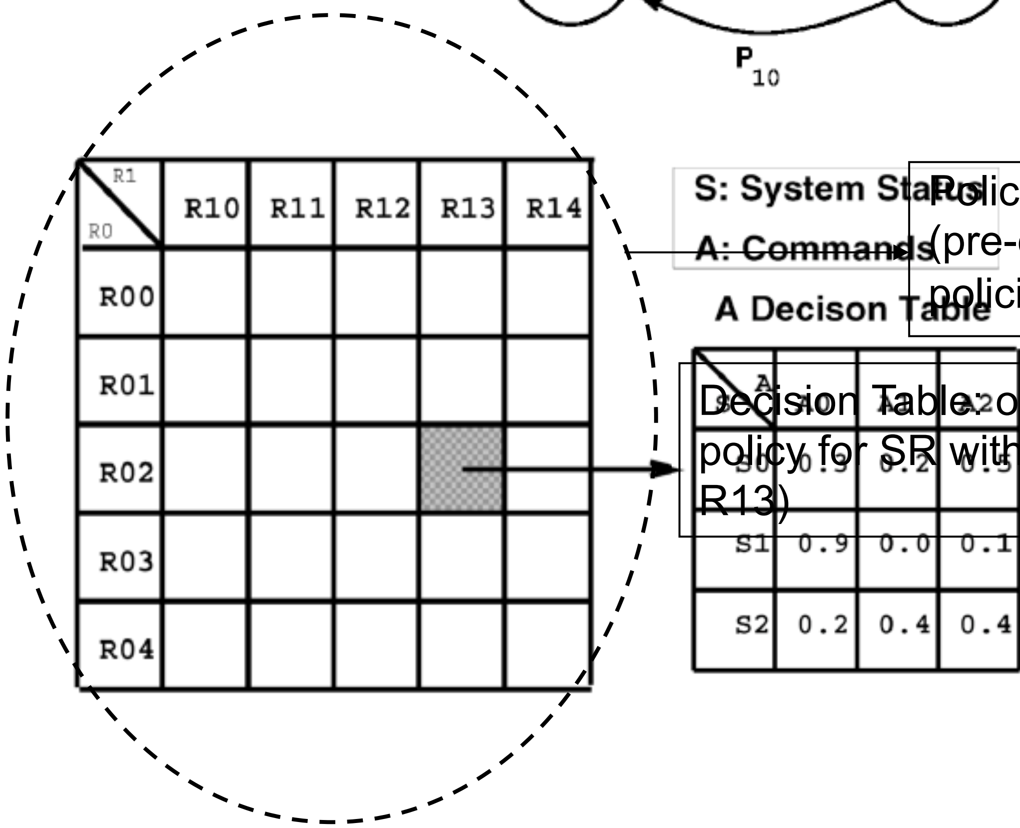
Stochastic Policies for non stationary workloads

- Chung et al, TC'02 extend the DTMDP model (TCAD'99) for handling non stationary workloads
- Key ideas:
  - Policy pre-characterization
  - Parameter learning
  - Policy interpolation

Policy Pre-characterization

(Chung et al, TC'02)





Policy Table (pre-characterized policies for different SR)

Decision Table: optimal policy for SR with (R02, R13)

Parameter Learning

At time T_n

SR

W(0)	W(1)	W(2)	W(3)		W(k-1)	W(k)
0	1	1	1	....	0	1

At time T_{n+1}

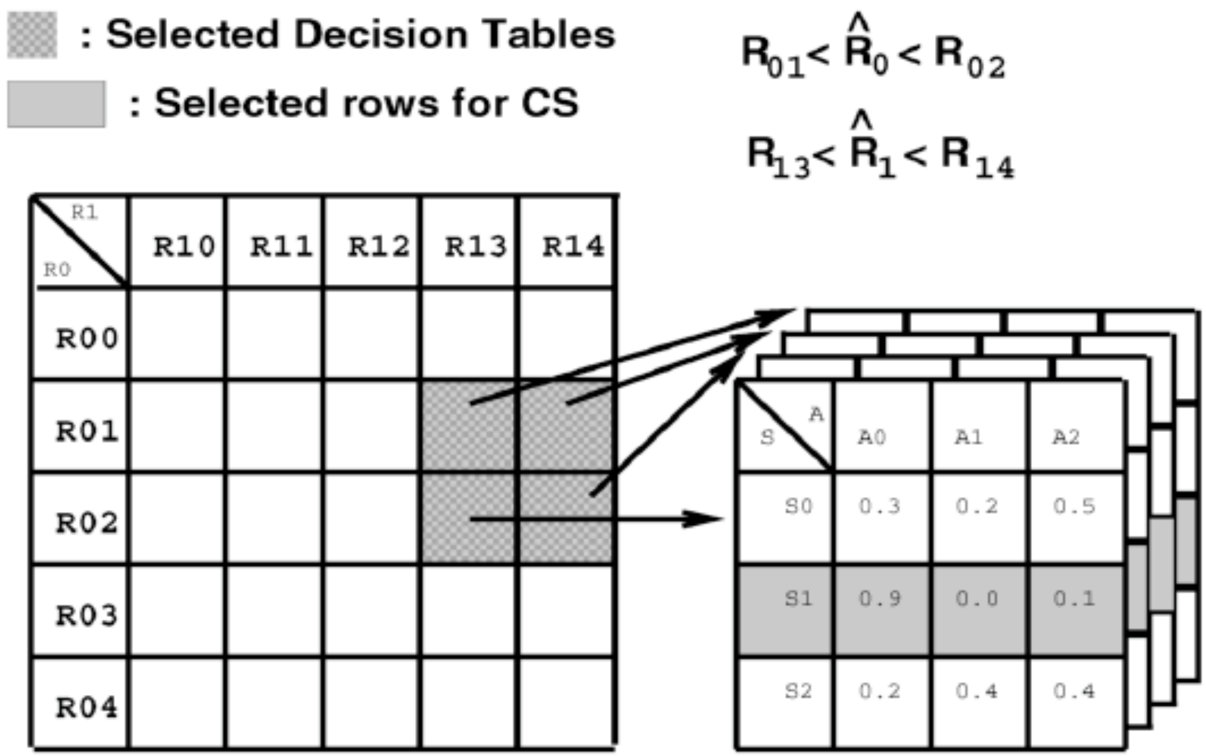
SR

W(0)	W(1)	W(2)	W(3)		W(k-1)	W(k)
0	0	1	1	....	0	0

Dynamically estimates $\hat{R}_0$ and $\hat{R}_1$

Uses Maximum Likelihood Estimation

Policy Interpolation



(Chung et al, TC'02)

Performs linear interpolation on selected decision tables for probability distribution of actions

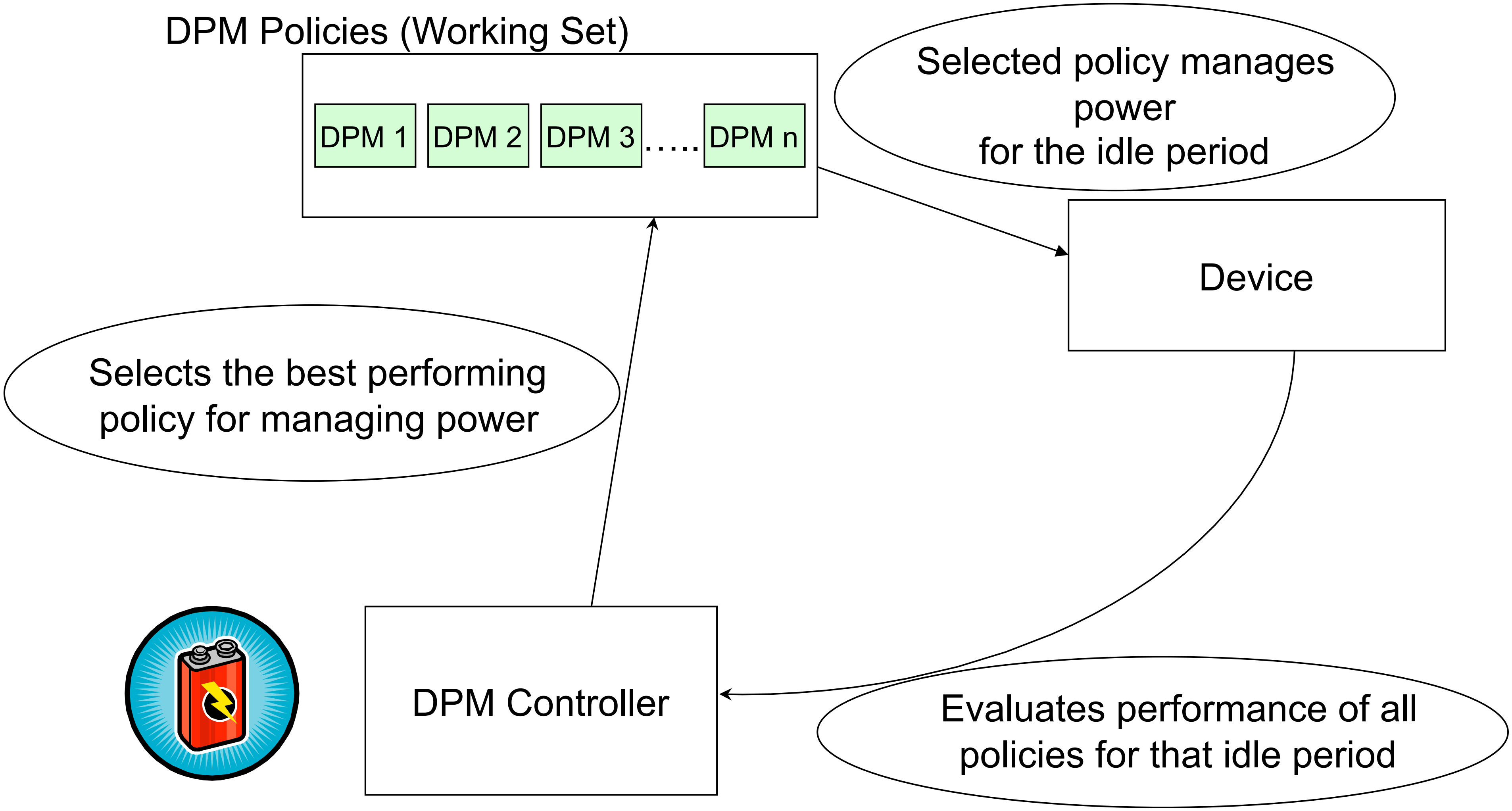
Policy interpolation model

- Advantages:
  - Takes non stationary workloads into account
    - Adapts with changing workloads
- Disadvantages
  - Not globally optimal
  - Discrete time model => overhead
  - Markovian workload assumption

Selection among multiple policies

- Dhiman et al, ICCAD'06 propose a framework for selection among multiple policies
- Based on the observation that different policies outperform each other under different workloads
- Employ an online learning algorithm to perform policy selection
- Learning algorithm provides performance bounds on convergence to the best suited policy

Online Learning for DPM



Online learning based model

- Advantages:
  - Adapts with changing workloads
    - Controller converges to the best suited policy
- Disadvantages
  - Performance as good as that of the best policy in the set

Quantitative Comparison (Lu et al)

Algorithm	P	N_{sd}	N_{wd}	T_{ss}	T_{bs}	
	desktop					
off-line	1.64	164	0	166	0	Oracle
SM	1.92	156	25	147	18.2	TISMDP
CA	1.94	160	15	142	17.6	Karlin
SW	1.97	168	26	134	18.7	Policy Interpolation
$\tau = 30$	2.05	147	18	142	30.0	
LT	2.07	379	232	62	5.7	Learning Tree
ATO3	2.09	147	26	138	29.9	
ATO1	2.19	141	37	135	27.6	
ATO2	2.22	595	430	41	4.1	
$\tau = 120$	2.52	55	3	238	120.0	
DM	2.60	105	39	130	48.9	DTMDP
EA	2.99	595	503	30	7.6	Exponential Average
always-on	3.48	-	-	-	-	