Power, Energy and Performance

Kunal Korgaonkar
Prof. Tajana Simunic Rosing
Department of Computer Science and Engineering
UCSD

Motivation for Power Management

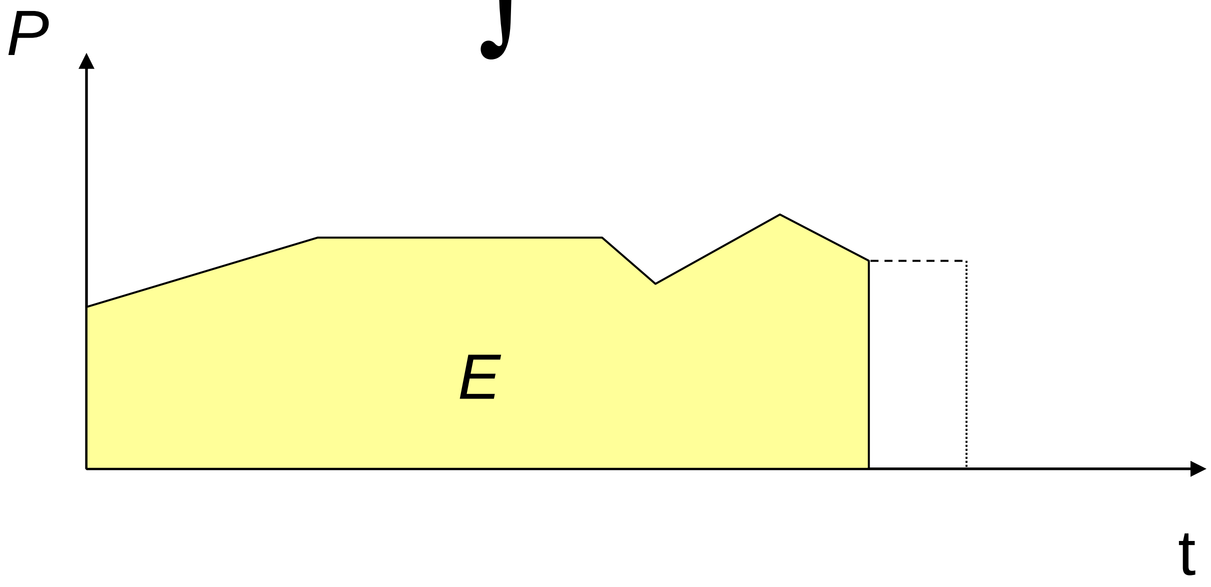
- Power consumption is a critical issue in system design today
  - Mobile systems: maximize battery life
  - High performance systems: minimize operational costs

1.2% of total electrical use in US devoted for powering and cooling data centers

$1 operation => $1 cooling

Power and Energy Relationship

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



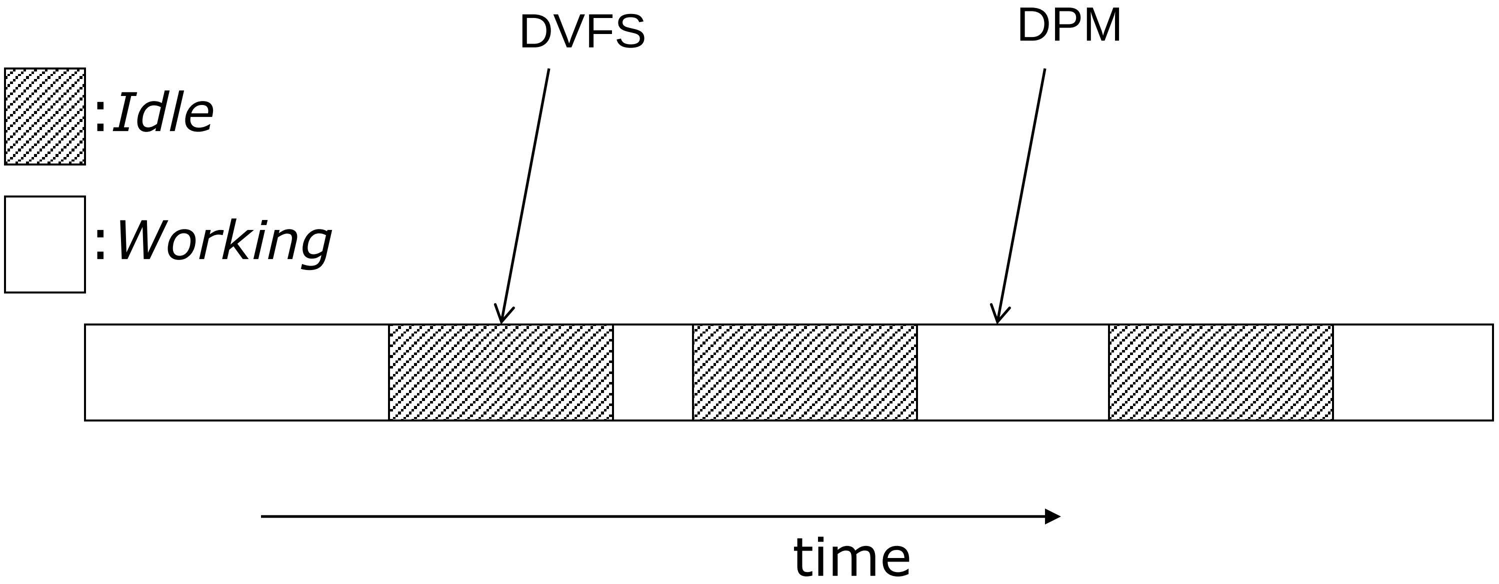
Low Power vs. Low Energy

- Minimizing the **power consumption** is important for
  - the design of the power supply
  - the design of voltage regulators
  - the dimensioning of interconnect
  - short term 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)
  - cooling
    - high costs
    - limited space
  - dependability
  - long lifetimes, low temperatures

Intuition

- System 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
- System Level Power Management Policies
  - Manage devices with different power management capabilities
  - Understand tradeoff between DPM and DVFS

DPM/DVFS



Dynamic Voltage Scaling (DVS)

Power consumption of CMOS circuits (ignoring leakage):

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

α : switching activity

C_L : load capacitance

V_{dd} : supply voltage

f : clock frequency

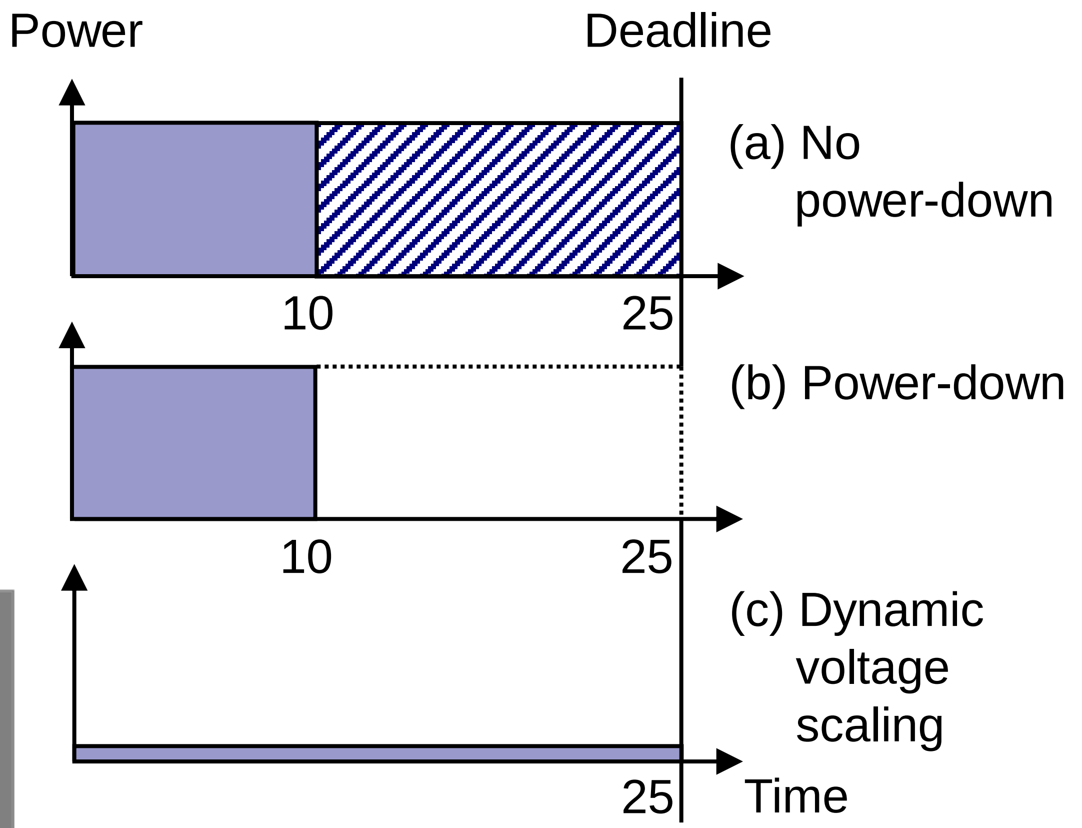
Delay for CMOS circuits:

$$\tau = k \, C_L \frac{V_{dd}}{(V_{dd} - V_t)^2} \text{ with}$$

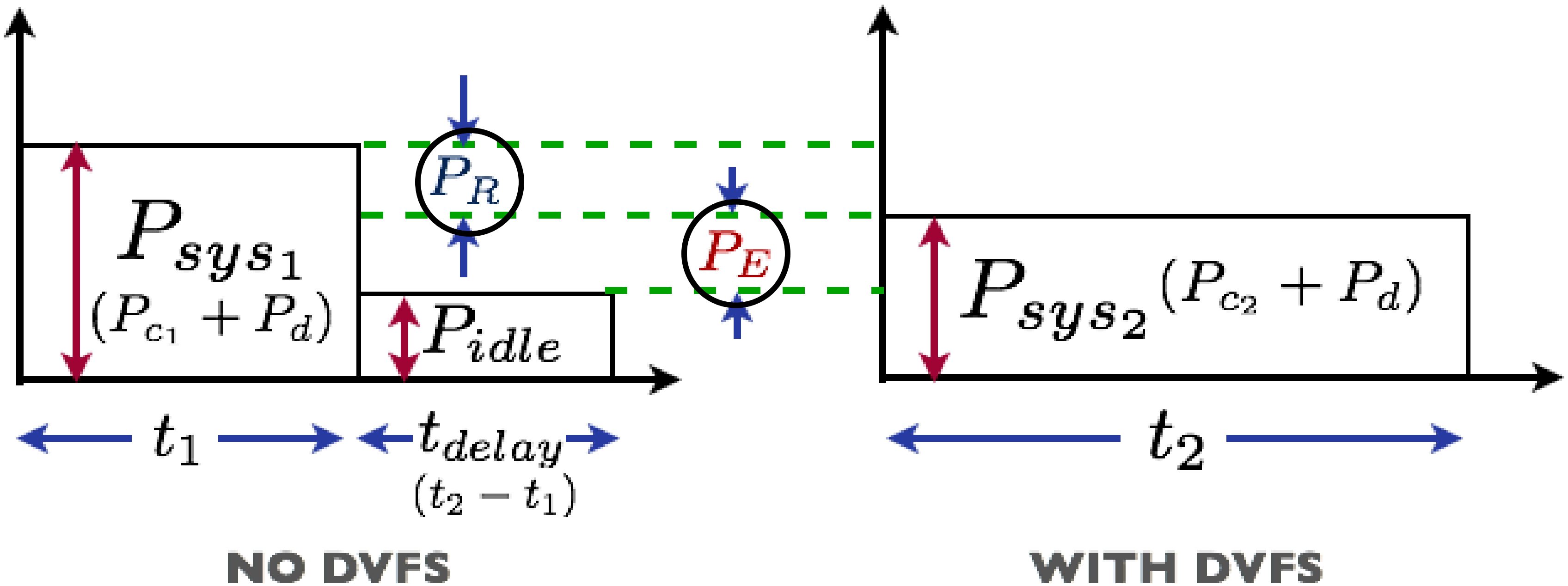
V_t : threshhold voltage

(V_t substancially < than V_{dd})

DVFS vs. power down



Energy Savings with DVFS



$P_R = P_{c_1} - P_{c_2}$ **Reduction in CPU power**

$P_E = (P_{c_2} - P_{c_{idle}}) + (P_d - P_{d_{idle}})$ **Extra system power**

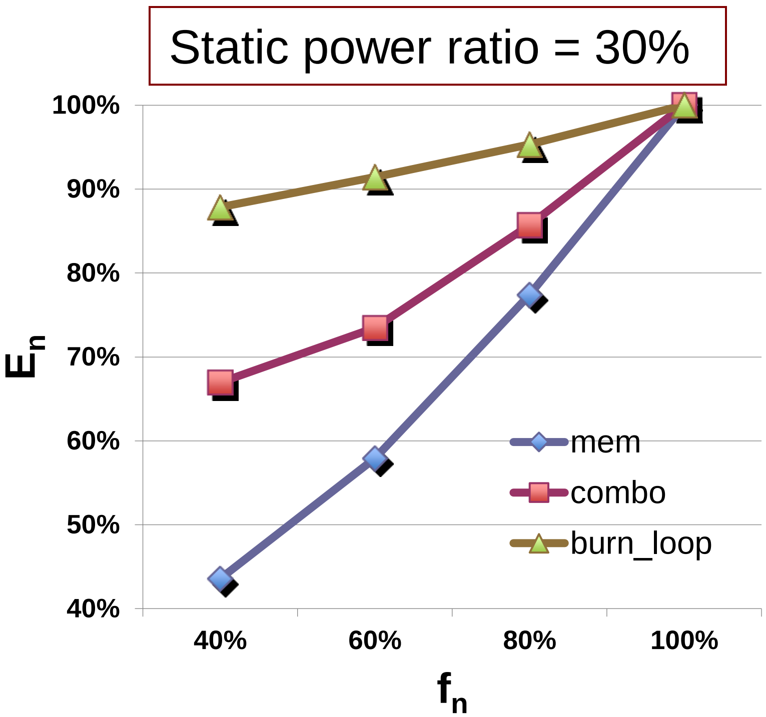
$$\boxed{E_{DVFS} = P_R t_1 - P_E t_{delay}}$$

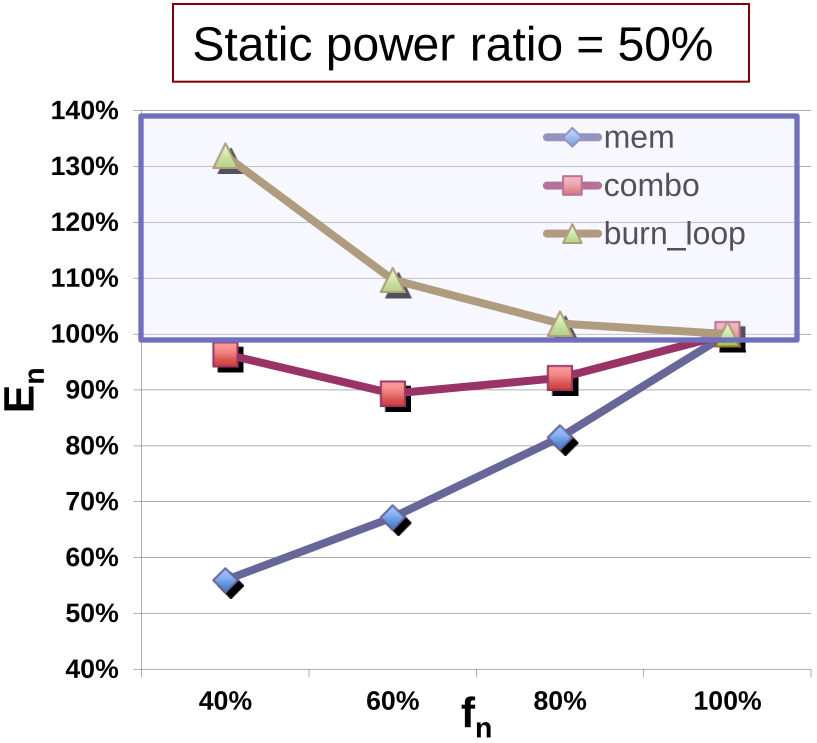
Effectiveness of DVFS

$$\boxed{\begin{aligned} E_{DVFS} &= P_R t_1 - P_E t_{delay} \\ &= E_R - E_E \end{aligned}}$$

- For energy savings
  - $E_R > E_E$
- Factors in modern systems affecting this equation:
  - Performance delay (t_{delay})
  - Idle CPU power consumption
  - Power consumption of other devices (P_E)

DVFS: Mem vs. CPU





Successful Low Power S/W Techniques

1. Understand workload variations
2. Devise efficient ways to detect them
3. Utilize the detected workload variations with available H/W

- Relatively easy for real-time jobs
- Hard for non real-time jobs
  - No timing constraints, no periodic execution
  - Unknown execution time

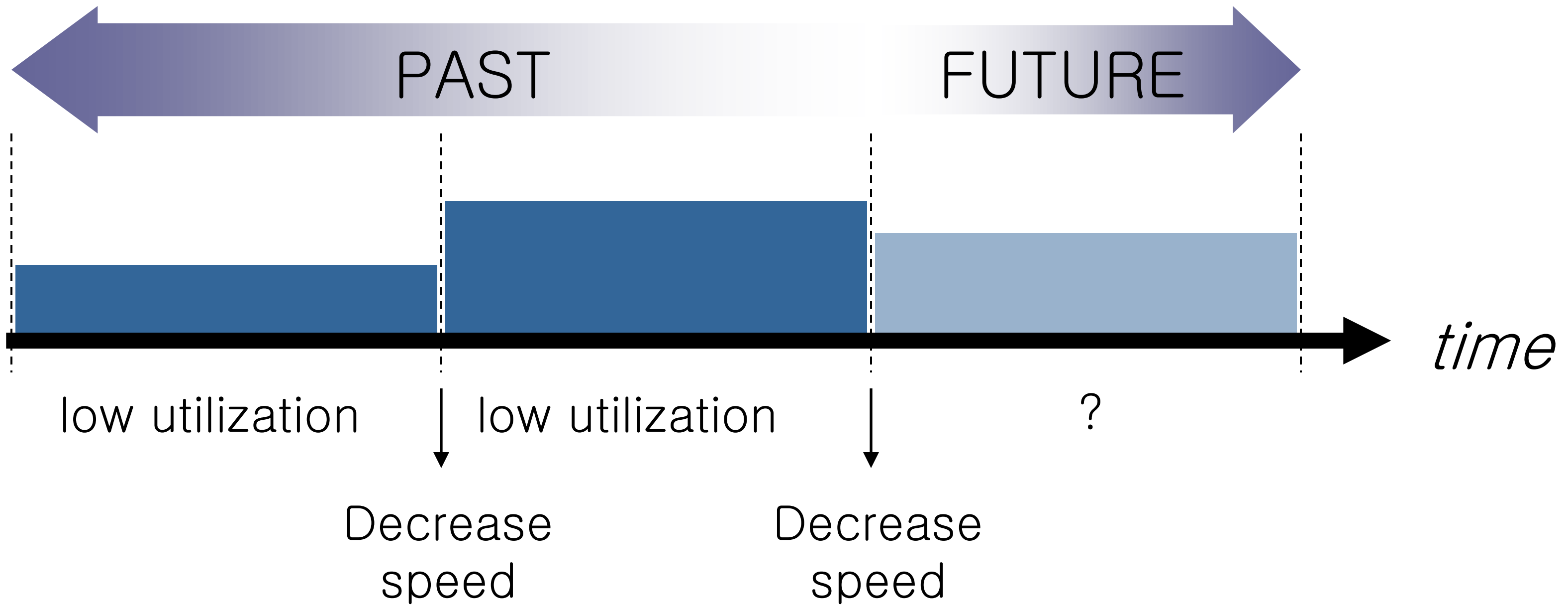
It is hard to predict the future workload!!

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

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



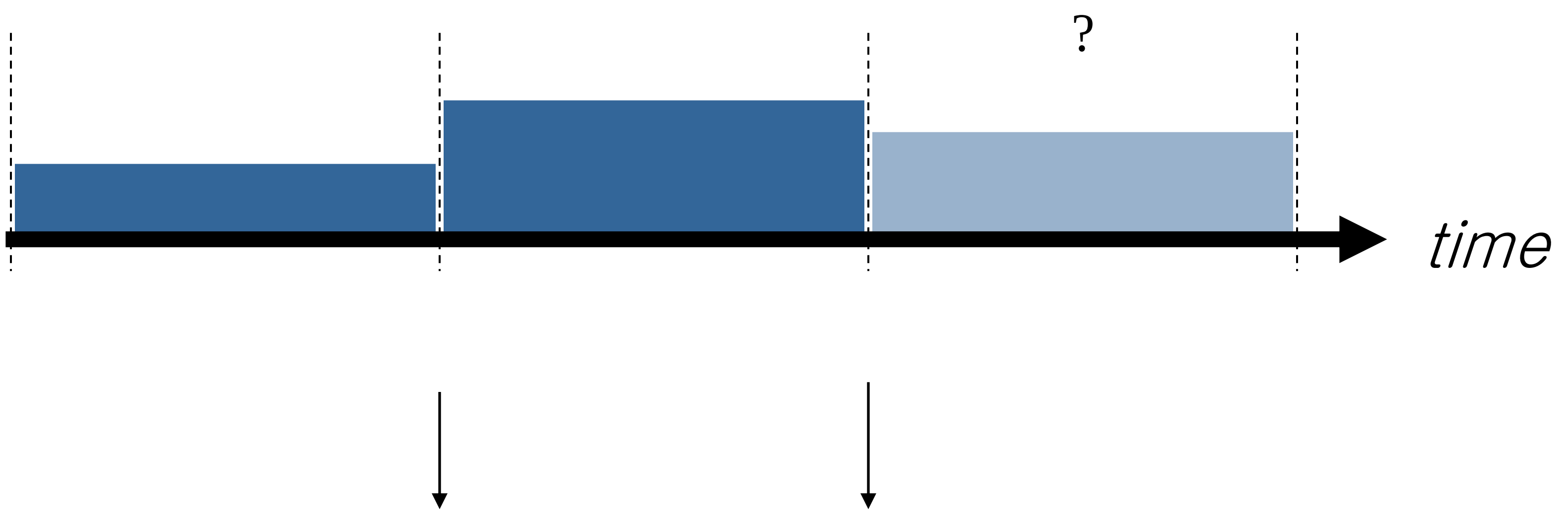
FLAT

- 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

<Const>=0.7



Increase/Decrease the speed
the next utilization to be 0.7

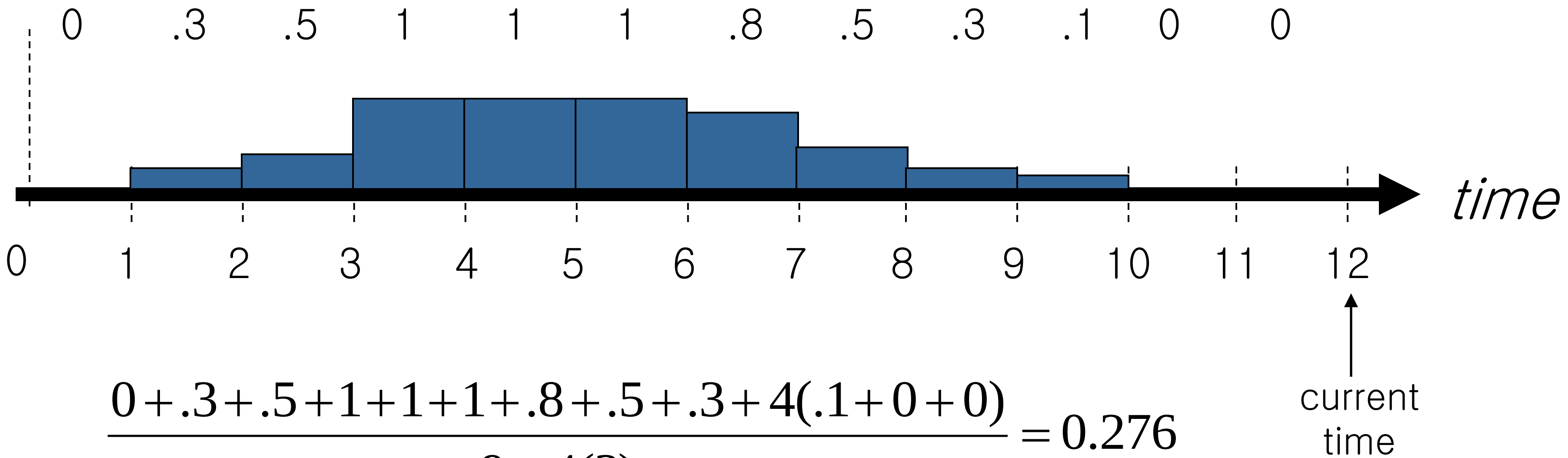
LONG-SHORT

- 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



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

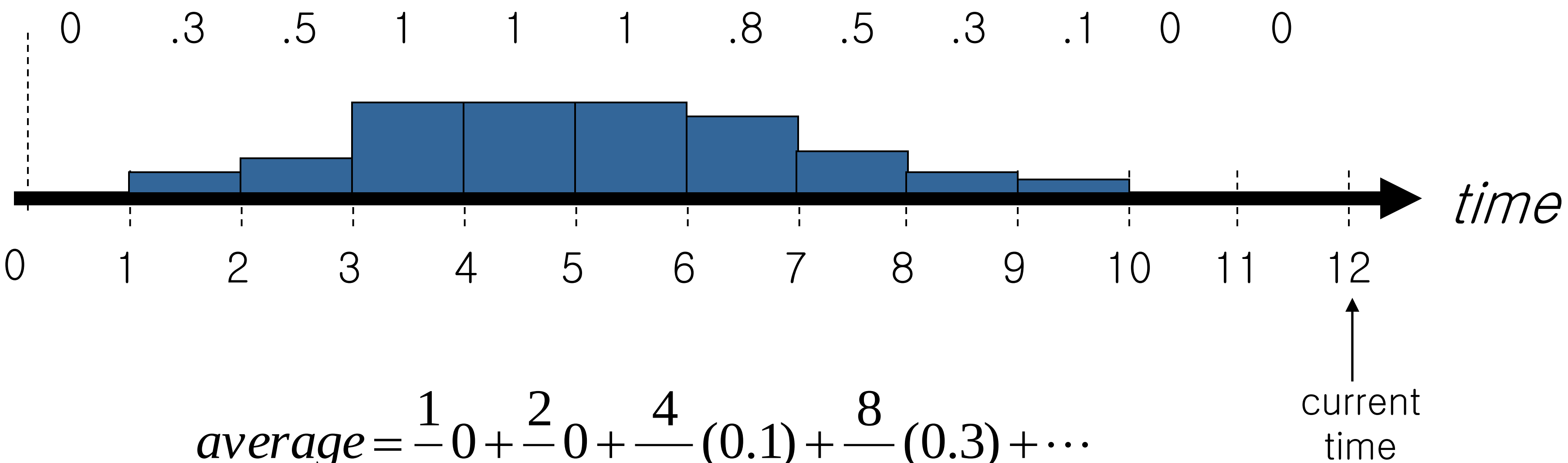
$$f_{clk}=0.276\times f_{\max}$$

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



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

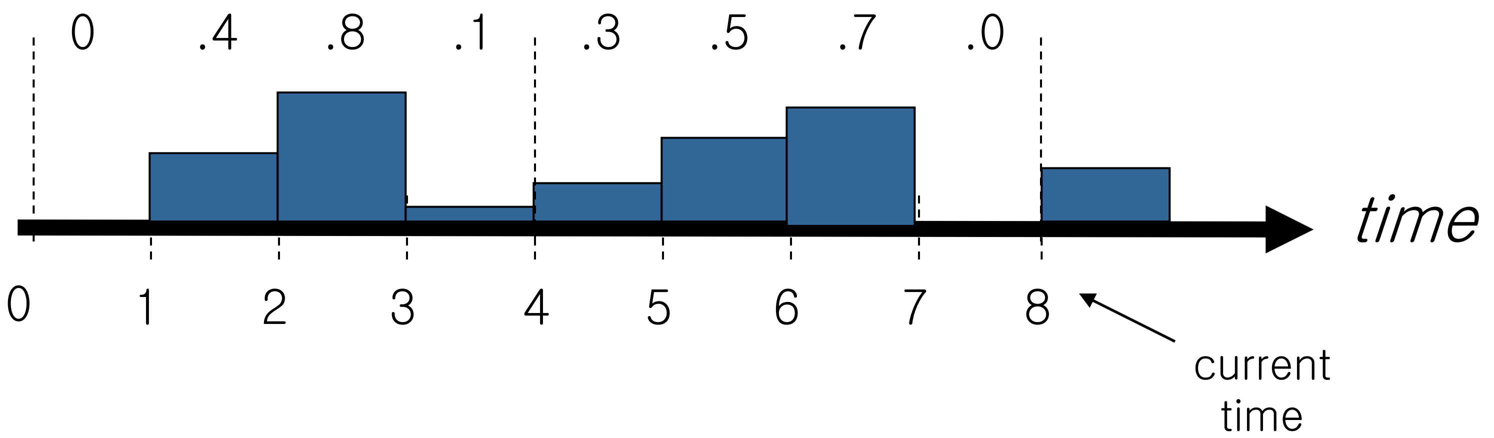
$$f_{clk} = average \times f_{\max}$$

CYCLE

- Workload Prediction
  - Examine the last 16 windows
    - Does there exist a cycle of length X?
    - If so, predict by extending this cycle
    - Otherwise, use the FLAT algorithm

Example: CYCLE

utilization = # cycles of busy interval / window size



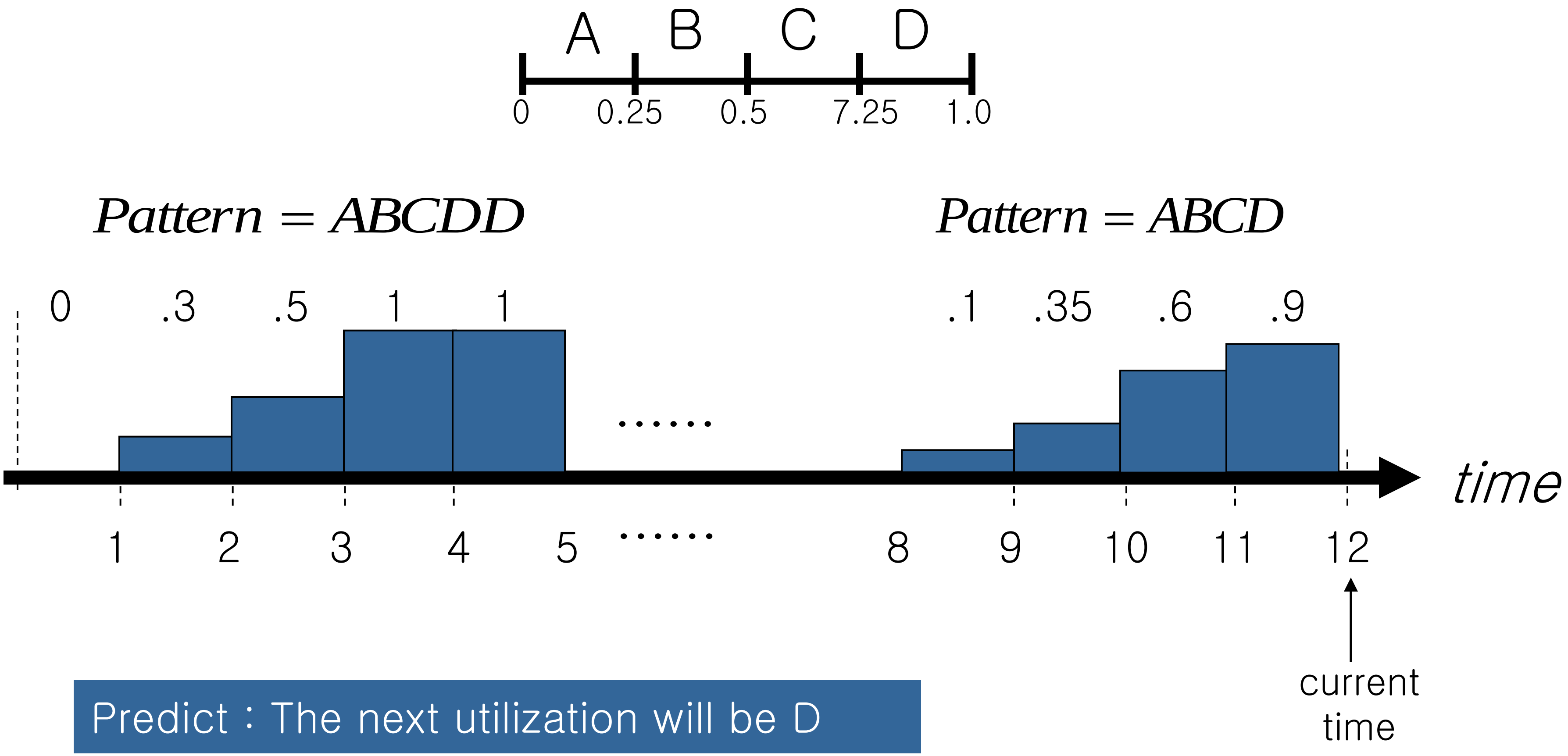
Predict : The next utilization will be .3

PATTERN

- A generalized version of CYCLE

- 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

Example: PATTERN



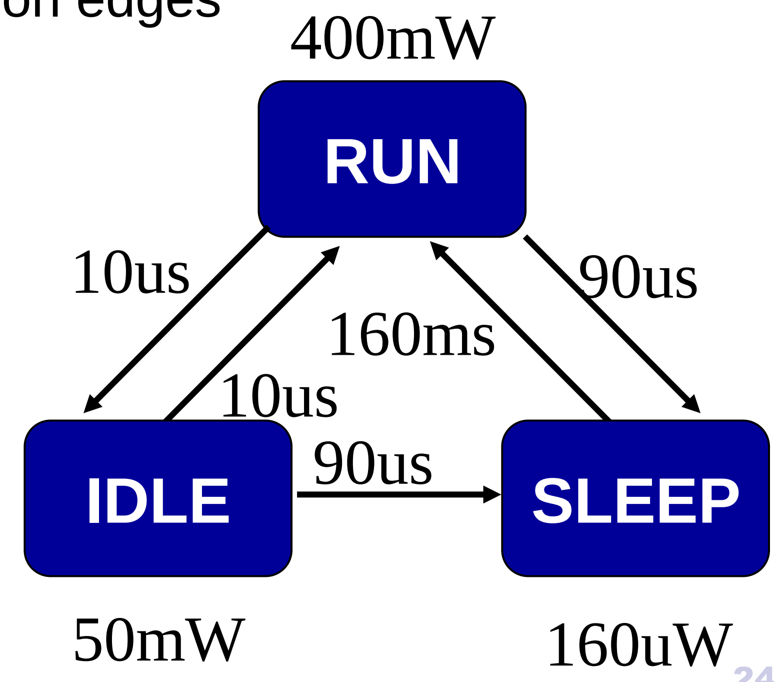
Predict : The next utilization will be D

Dynamic Power Management

- Components with several internal states
  - Corresponding to power and service levels
- Abstracted as **power state machines**
  - State diagram with:
    - Power and service annotation on states
    - Power and delay annotation on edges

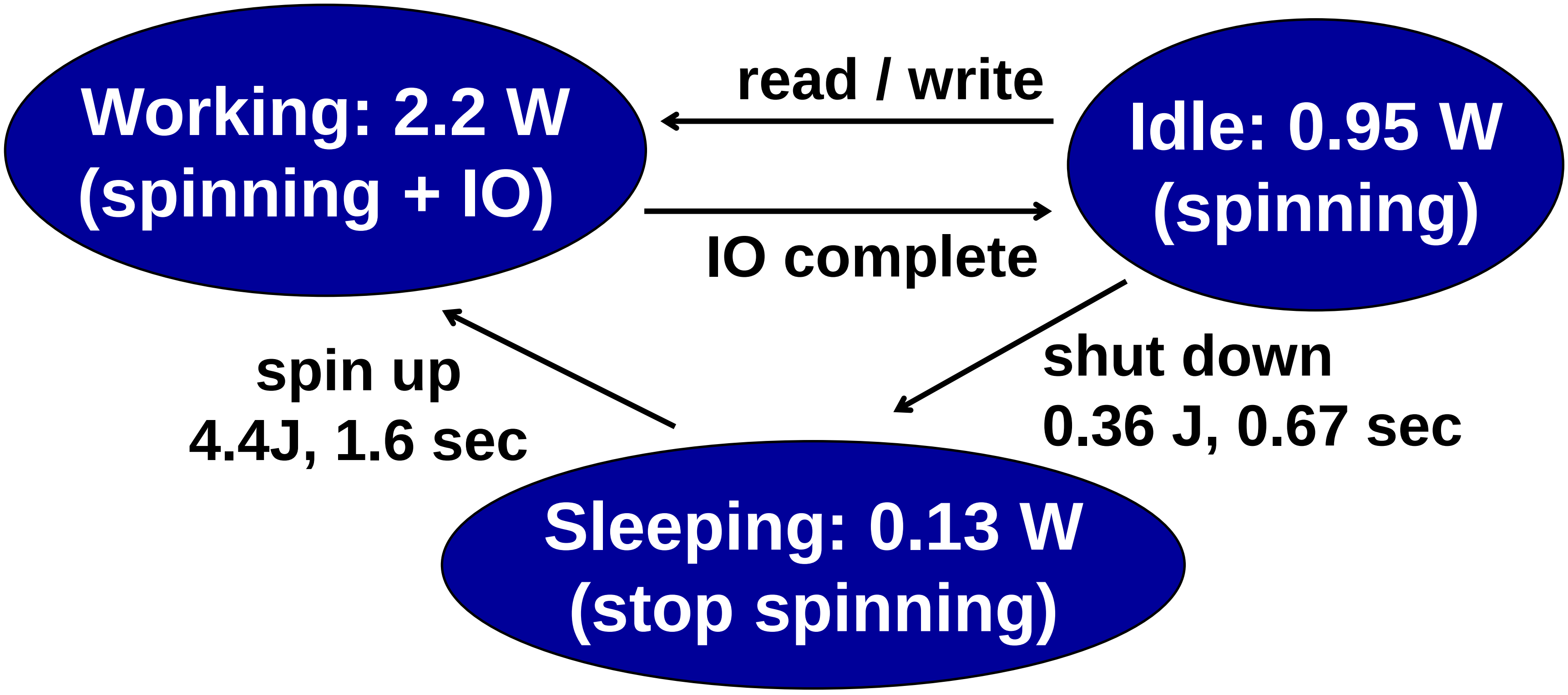
Example: SA-1100

- **RUN:** operational
- **IDLE**: a sw routine may stop the CPU when not in use, while monitoring interrupts
- **SLEEP:** Shutdown of on-chip activity



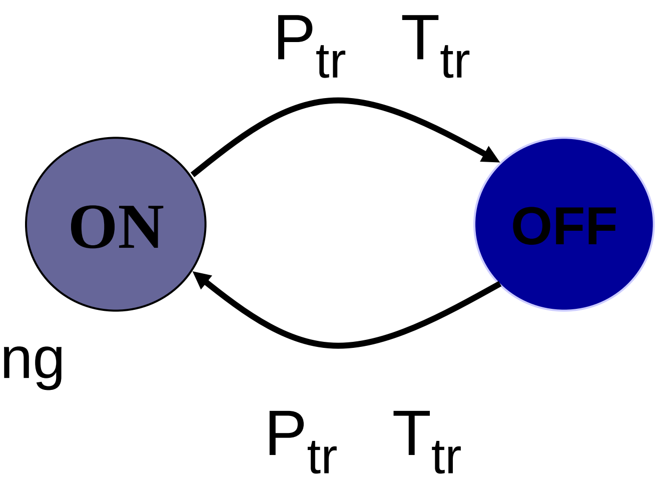
Example: Hard Disk Drive

Fujitsu MHF 2043 AT

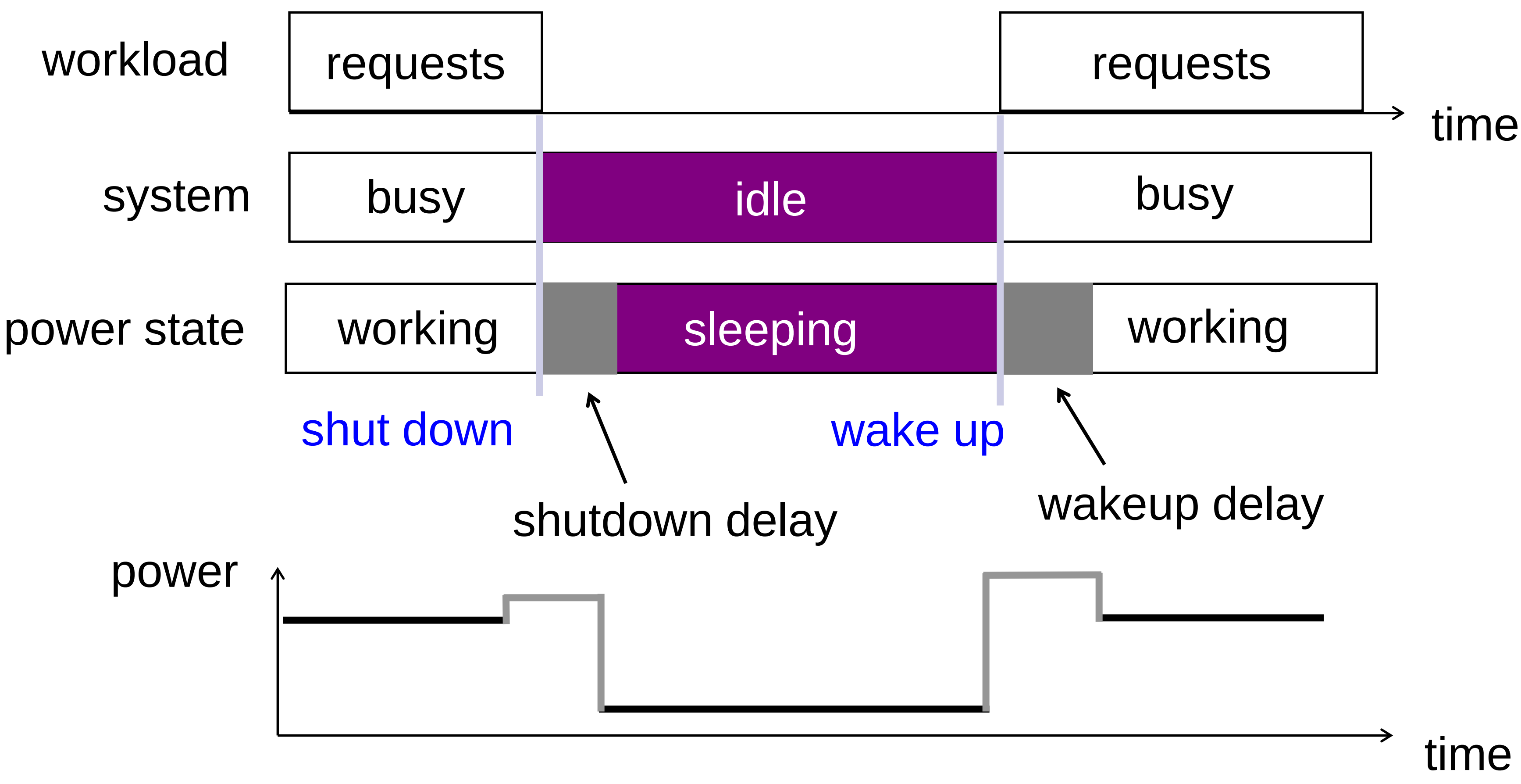


The Applicability of DPM

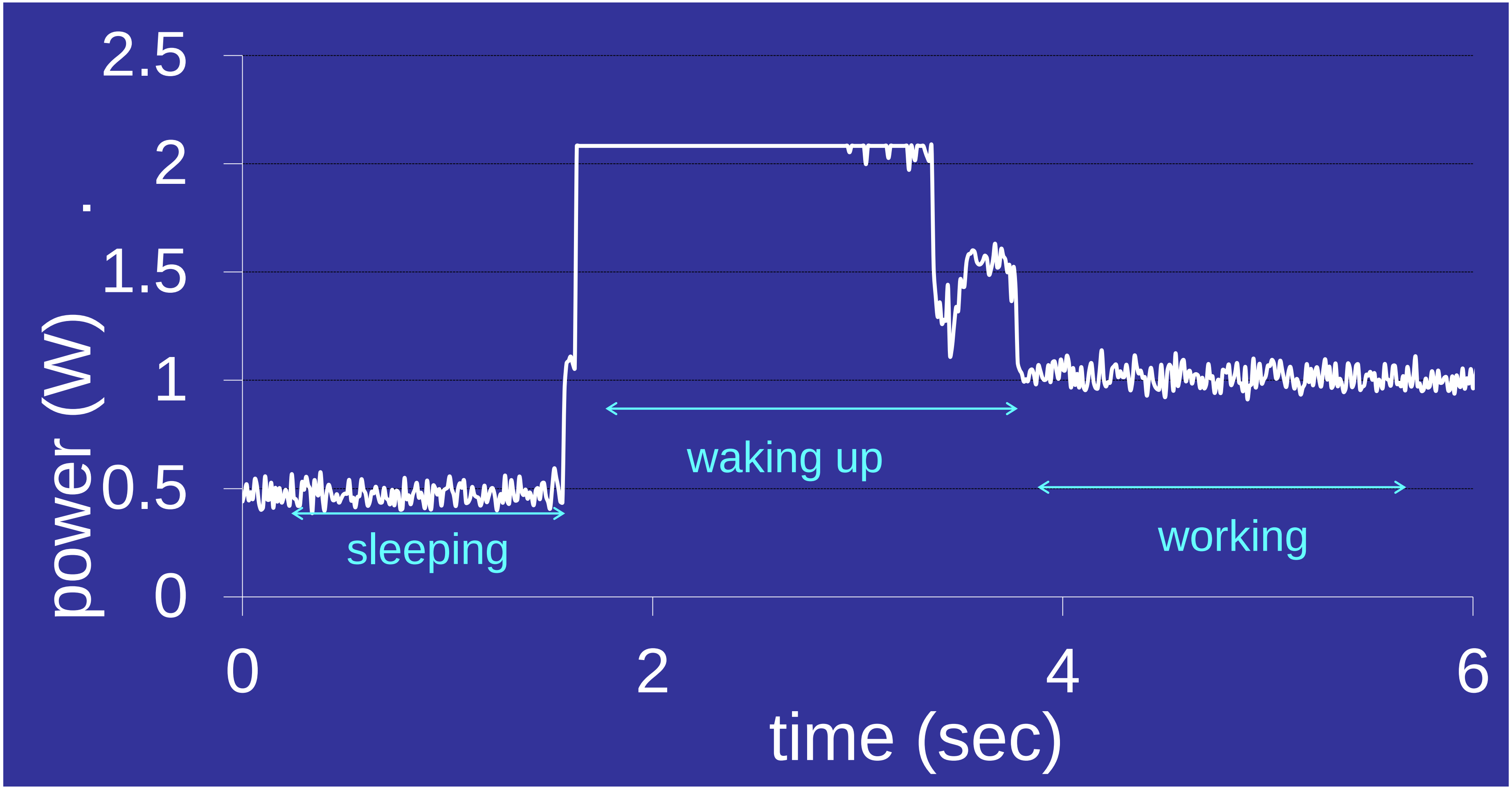
- State transition power (P_{tr}) and delay (T_{tr})
- If $T_{tr} = 0$, $P_{tr} = 0$ the policy is trivial
  - Stop a component when it is not needed
- If T_{tr} != 0 or P_{tr} != 0 (always…)
  - Shutdown only when idleness is long enough to amortize the cost
  - What if T and P fluctuate?



Workload and System Representation



Waking Up Hard Disk



Measurements done on Fujitsu MHF 2043 AT hard disk

Break Even Time

- Minimum idle time for amortizing the cost of component shutdown

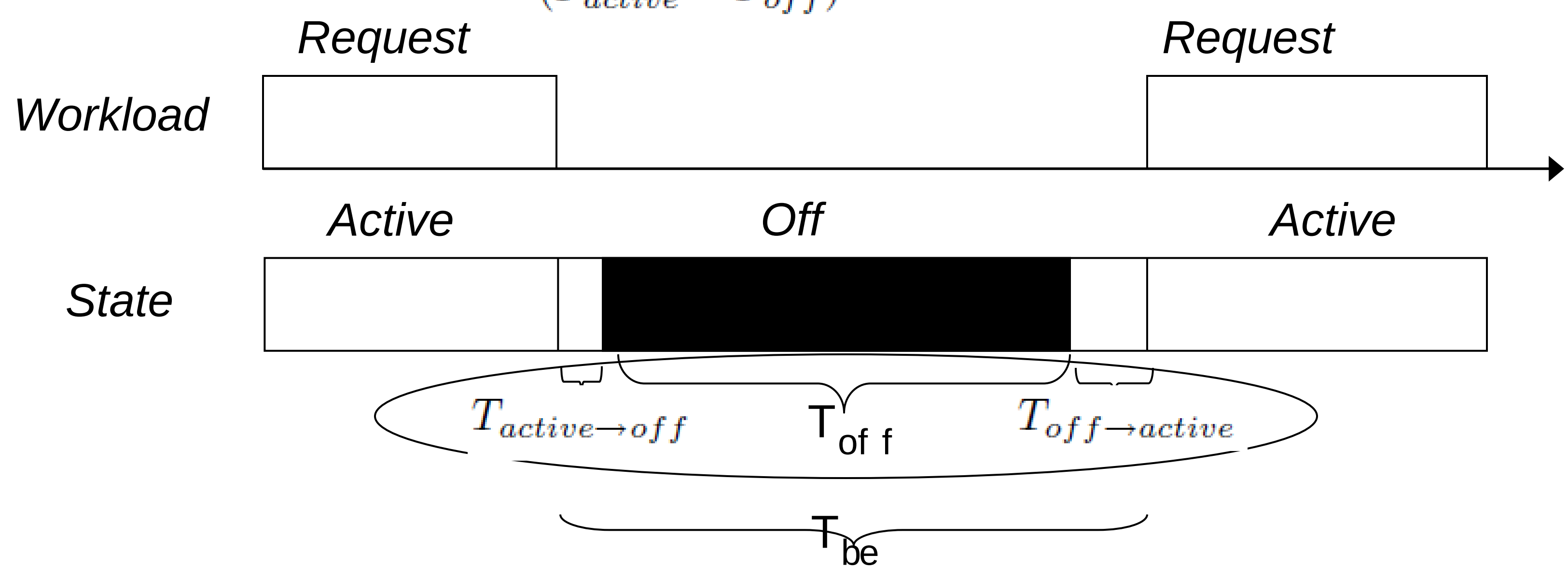
$$Energy_{saved} = Energy_{active} - (Energy_{tr} + Energy_{off})$$

$$Energy_{active} = Energy_{tr} + Energy_{off}$$

$$P_{active} \cdot (T_{tr} + T_{ms}) = P_{tr} \cdot T_{tr} + P_{off} \cdot T_{ms}$$

$$T_{be} = T_{ms} + T_{tr}$$

$$T_{ms} = \frac{T_{tr} \cdot (P_{tr} - P_{active})}{(P_{active} - P_{off})}$$



DPM Applicability

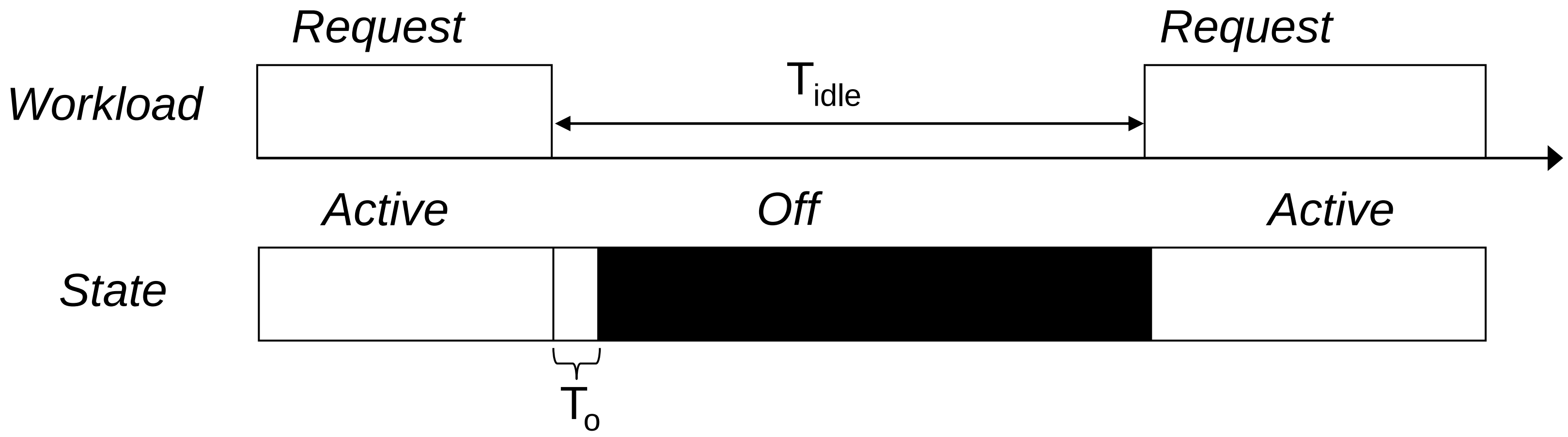
- If idle period length < T_{be}
  - Not possible to save energy by turning off
- Need accurate estimation of upcoming idle periods:
  - *Underestimation*: potential energy savings lost
  - *Overestimation*: perf delay + possible –ve energy savings
- *Challenge*: Manage energy/performance tradeoff
- *DPM Policy Goal*: Maximize energy savings by utilizing sleep states while minimizing performance delay

Classification of DPM Policies

- Timeout Policies
- Predictive Policies
- Stochastic Policies
- Hybrid Policies
  - Online learning DPM

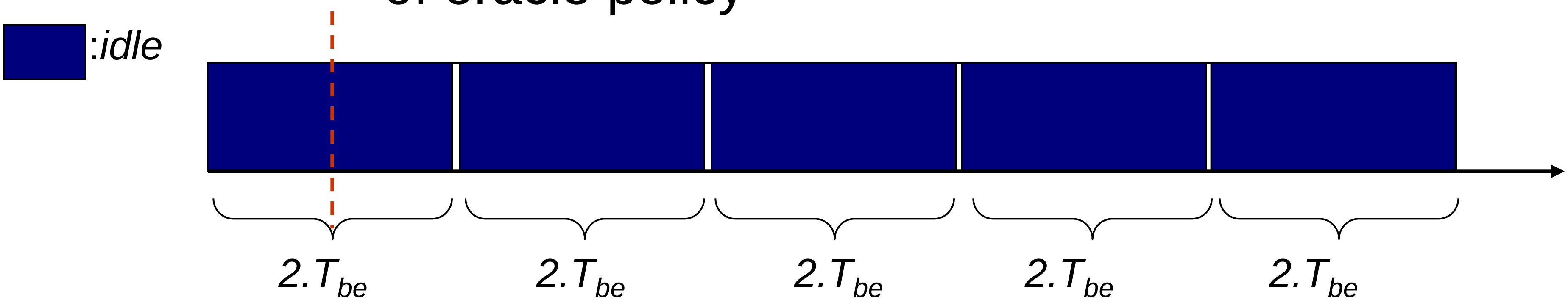
Timeout Policies

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



Fixed Timeout Policies

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



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

Fixed Timeout Policies contd.

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

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

Holds true for, $T_{be} > T_{tr}$ and $P_{tr} > P_{active}$

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

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 perf 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
  - Online learning DPM

Predictive Policies

- Srivastava et al, TVLSI'96 propose 2 predictive algorithms:

Algorithm 1:

- Use a non linear regression equation to perform prediction:

$$T_{pred} = \phi(T^{n}_{active}, T^{n-1}_{idle}, \cdots, T^{n-k}_{active}, T^{n-k-1}_{idle})$$

  - Perform extensive offline trace data analysis to derive regression equation and coefficients
  - Perform shutdown as soon as idle if $T_{pred} > T_{be}$

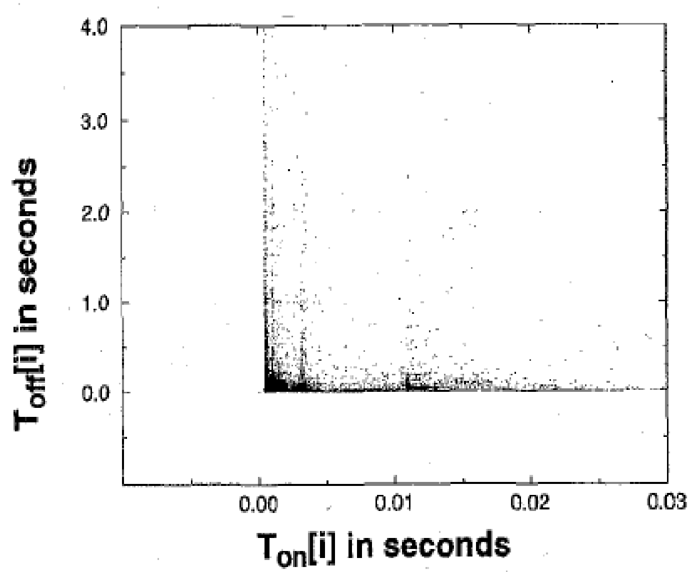
Predictive Policies

Algorithm 2:

- Uses following filtering rule as the prediction heuristic:

$$T^{n}_{idle} > T_{be} \;(\;)\; T^{n}_{active} < T_{threshold}$$

- Valid for class of systems where the scatter plot of idle and active times is L-shaped



Predictive Policies

- Hwang et al, ICCAD'97 propose predictive algorithm capable of online adaptation
  - Uses exponential average of previous idle period lengths to perform prediction:

$$T_{pred}^{n} = \alpha T_{idle}^{n-1} + (1-\alpha) T_{pred}^{n-1}$$

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

Predictive Policies

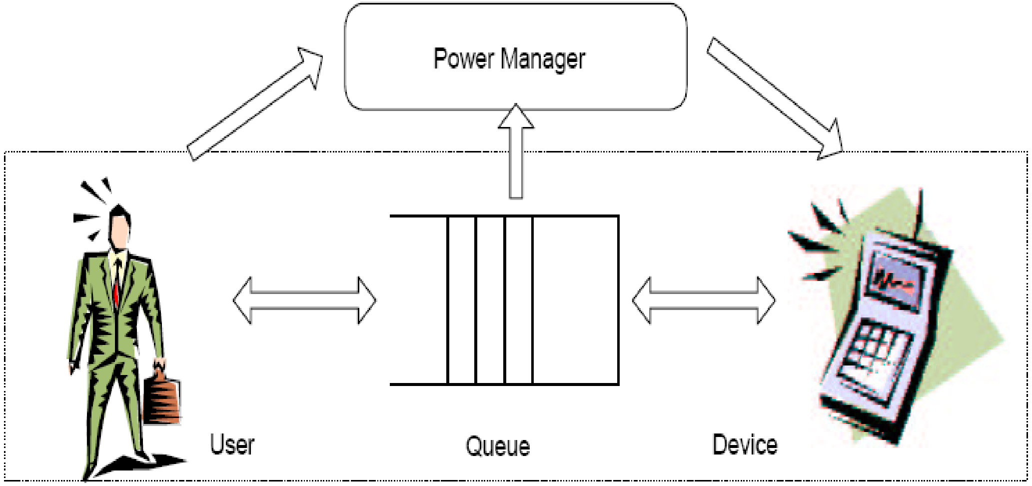
- Advantages
  - More energy efficient than timeout policies
- Disadvantages
  - Depend a lot on correlation between past and future events
  - Tend to be aggressive in shutdown and hence higher performance latency
  - Heuristic with no performance guarantees

Classification of DPM Policies

- Timeout Policies
- Predictive Policies
- Stochastic Policies
- Hybrid Policies
  - Online learning DPM

Time-indexed Semi-Markov Model (Simunic et al, TCAD'01)

- Globally optimal solution with performance constraints met
  - Valid for a stationary class of workloads
  - Implemented and measured large savings in mobile systems



Service Requester (SR)

Service Queue (SQ)

Service Provider (SP)

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

DPM Policies

Software entities that take DPM decisions

- ***Timeout***: Karlin, Irani TECS'03
- ***Predictive***: Srivastava, Hwang ICCAD'99

  ⇨ *Heuristic: No performance guarantees*
- ***Stochastic***: DTMDP (Benini TCAD'00), TISMDP (Simunic TCAD'01)

  ⇨ *Optimality for stationary workloads*
- Observation:

  ⇨ *Low flexibility in terms of user perceived energy savings performance delay tradeoff*

  ⇨ *Policies out perform each other under different workloads*

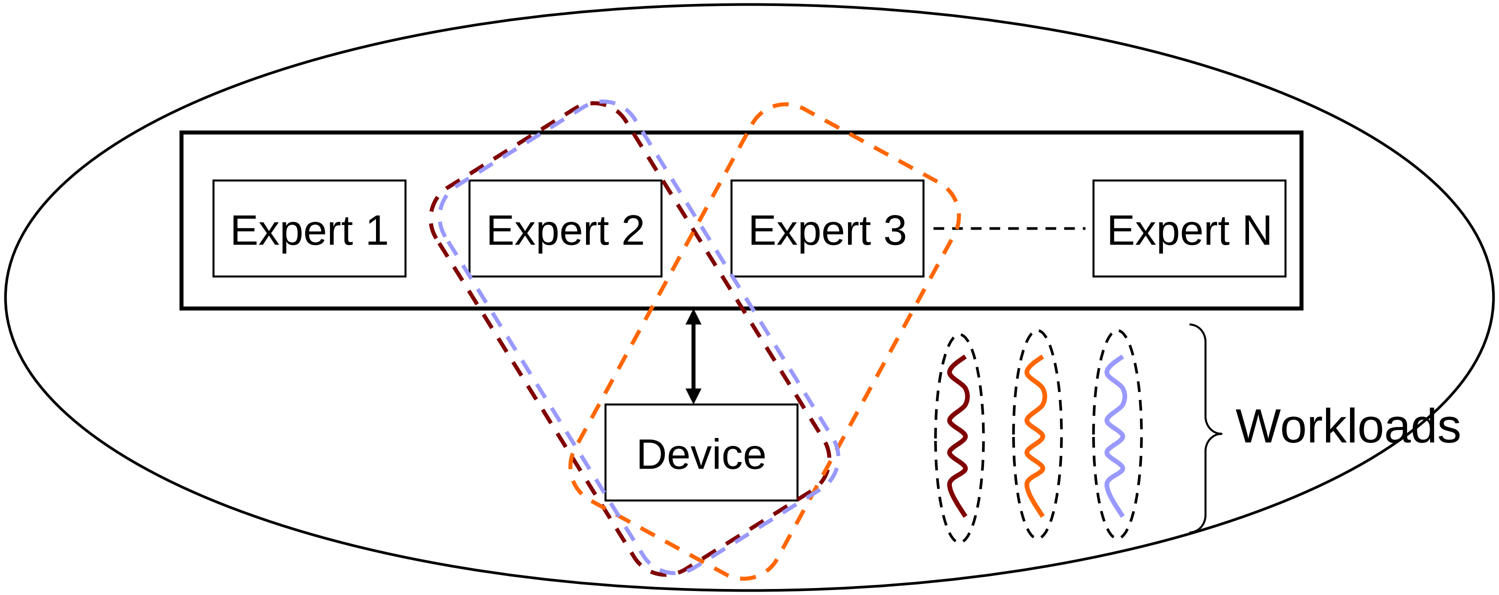
Classification of DPM Policies

- Timeout Policies
- Predictive Policies
- Stochastic Policies
- Hybrid Policies
  - Online learning DPM

Problem Formulation

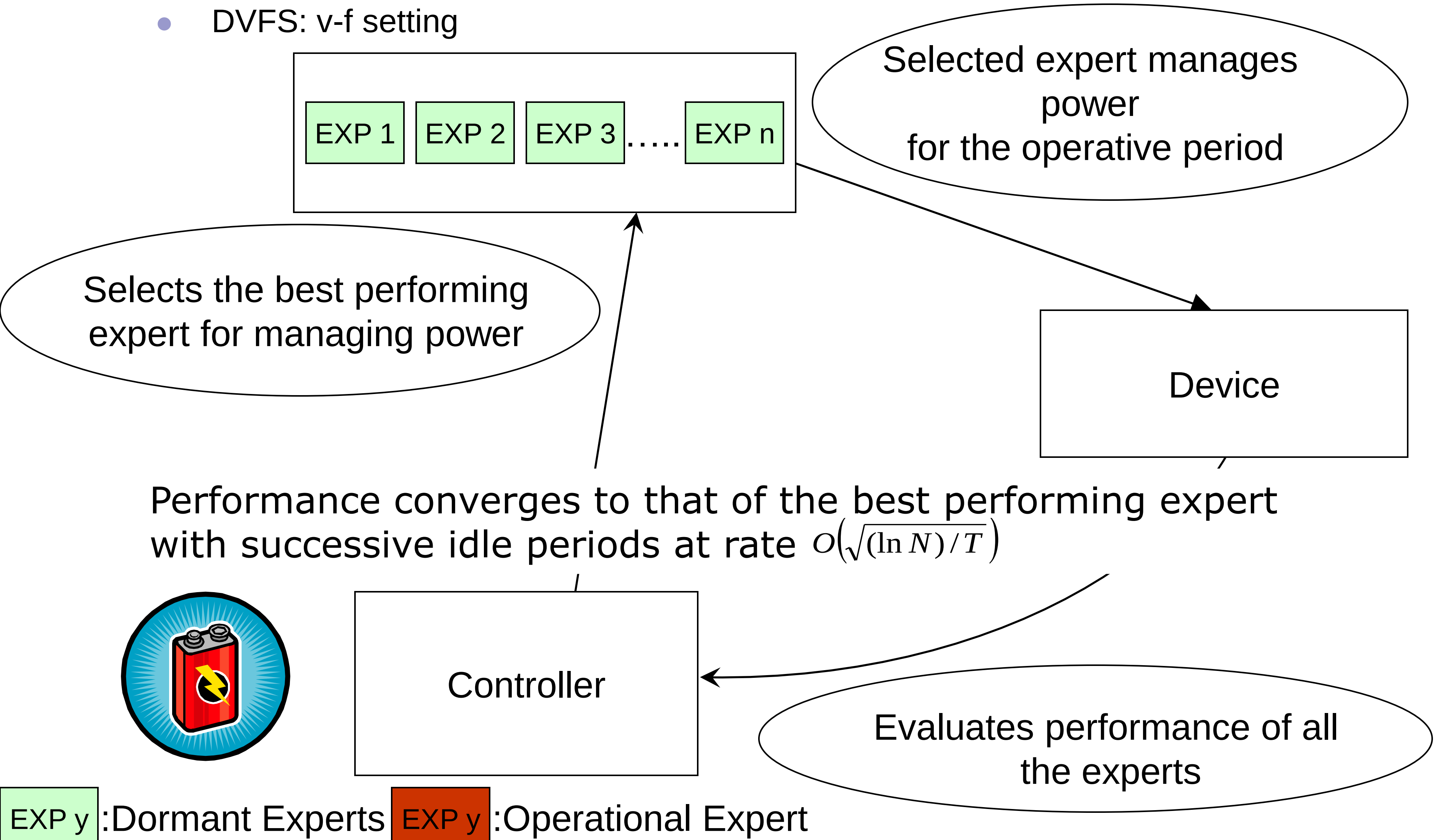
- Take a set of existing DPM/DVFS policies/experts
  - Perform dynamic selection at run time

DPM becomes a problem of:
characterization and *expert selection*



Online Learning for Power Management

- Get up to 70% energy savings per device
  - DPM: A state of the art DPM policy
  - DVFS: v-f setting



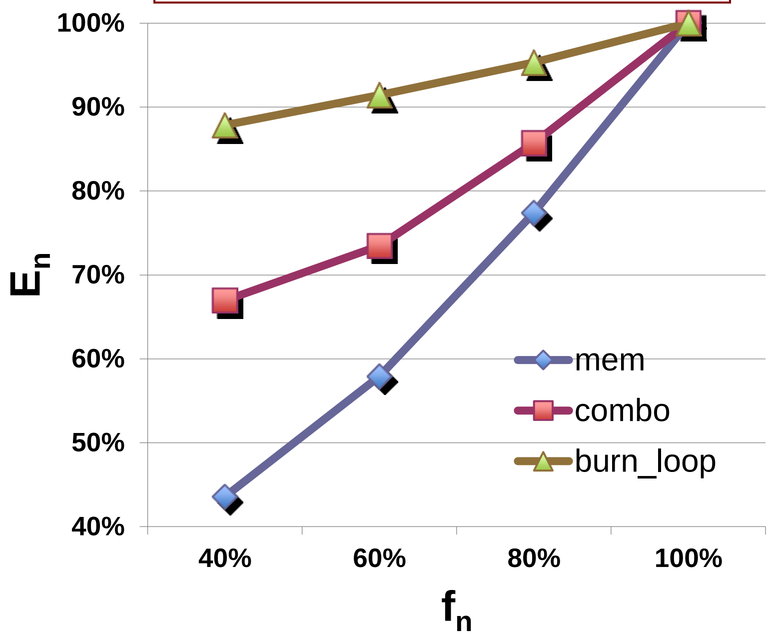
Expert Evaluation (DVFS)

- Operative period == scheduler quantum
  - Suitability of v-f setting expert depends on:
    - Task characteristics (CPU/mem intensiveness)
    - Device leakage characteristics
- Quantify task characteristics
  - **CPI Stack**

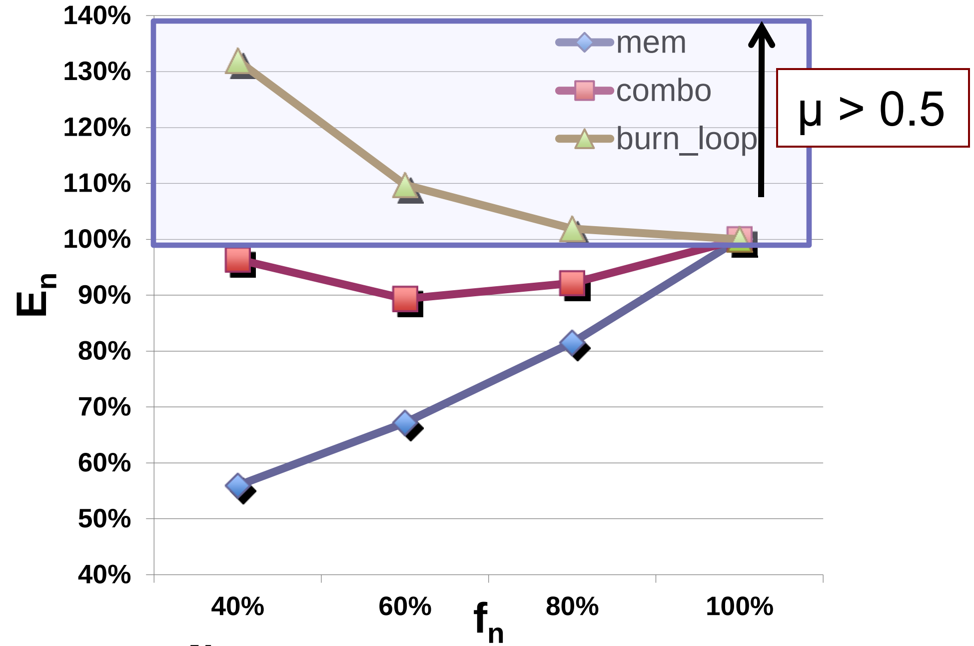
    $CPI_{avg}=CPI_{base}+CPI_{cache}+CPI_{tlb}+CPI_{branch}+CPI_{stall}$
  - Estimate $\mu = CPI_{base}/CPI_{avg}$
  - High μ => CPU-intensiveness

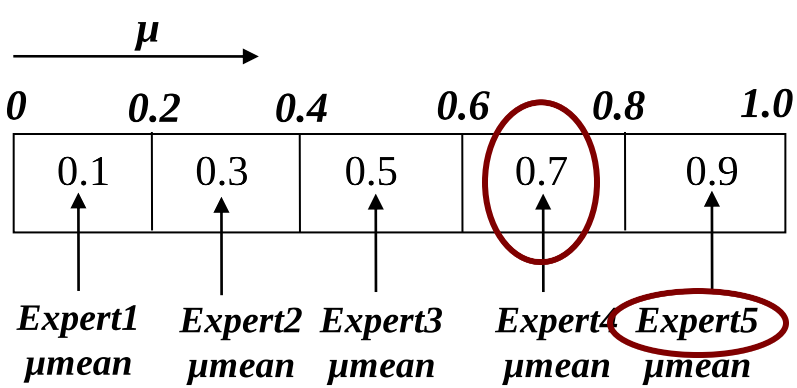
DVFS: Mem vs. CPU

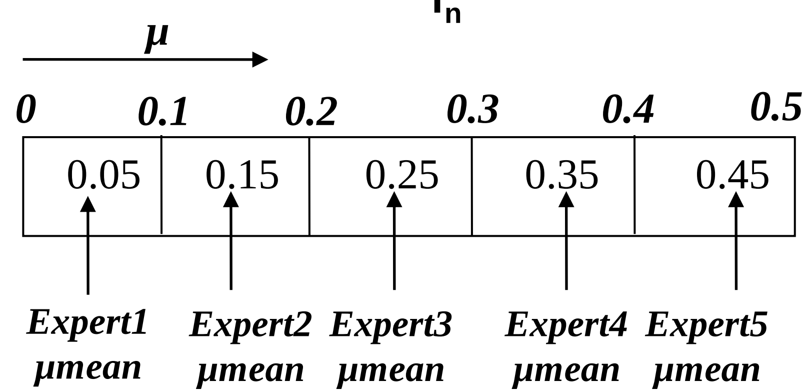
Static power ratio = 30%



Static power ratio = 50%



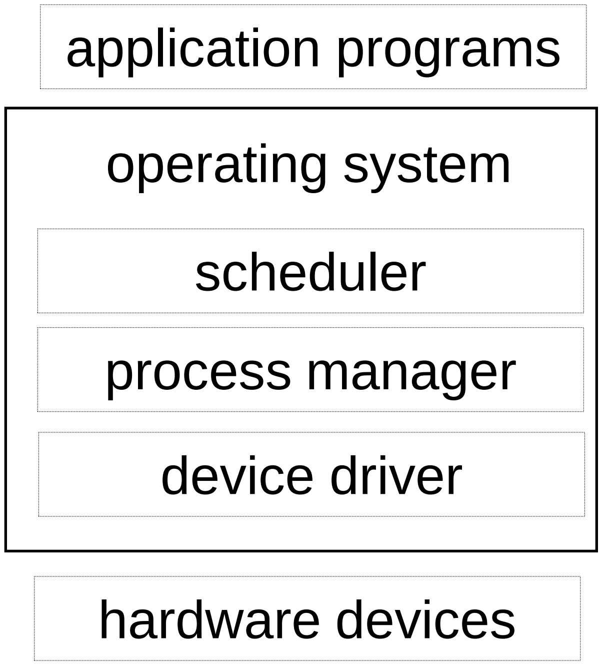




Energy Loss = (0.9 – 0.7) = 0.2
Performance Loss = 0

DPM and Operating Systems

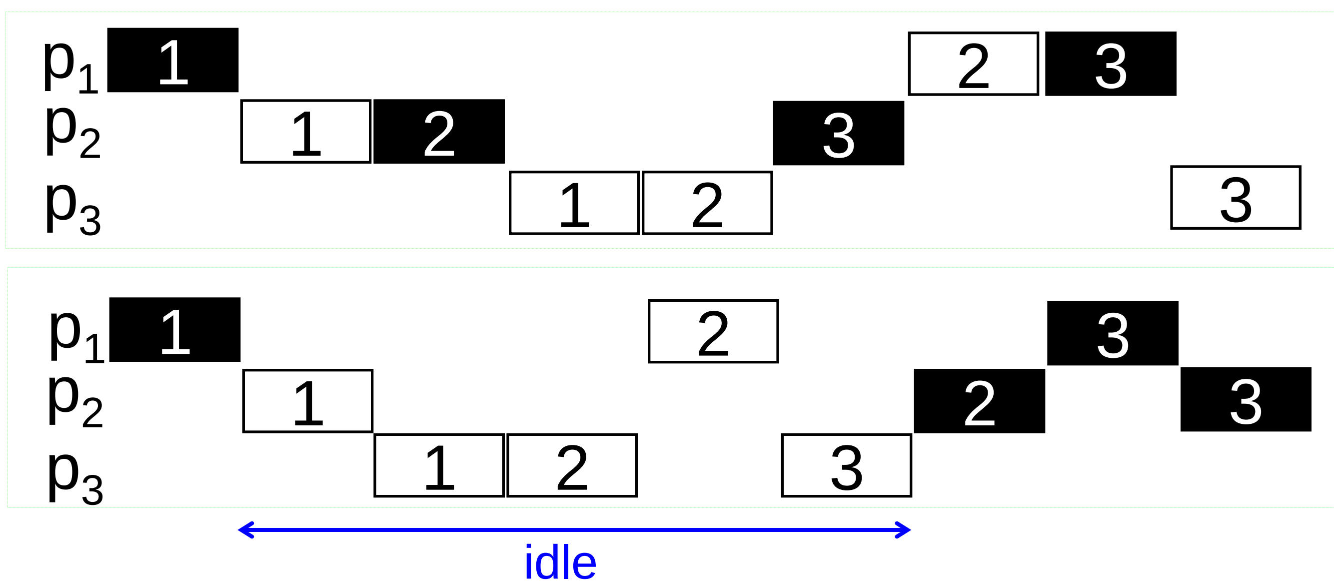
- Application
  - should not directly control hardware power
  - no power management in legacy programs
- Scheduler
  - selects processes and affects idle periods
- Process manager
  - knows multiple requesters
  - can estimate idle periods more accurately
- Driver
  - detects busy and idle periods
- Device
  - consumes power
  - should provide mechanism, not policy



Low-Power Device Scheduling

tasks specify
- device requirements
- timing requirements

operating system
1. groups tasks with same device requirements
2. execute tasks in groups
3. wake up devices in advance to meet timing constraints



Summary

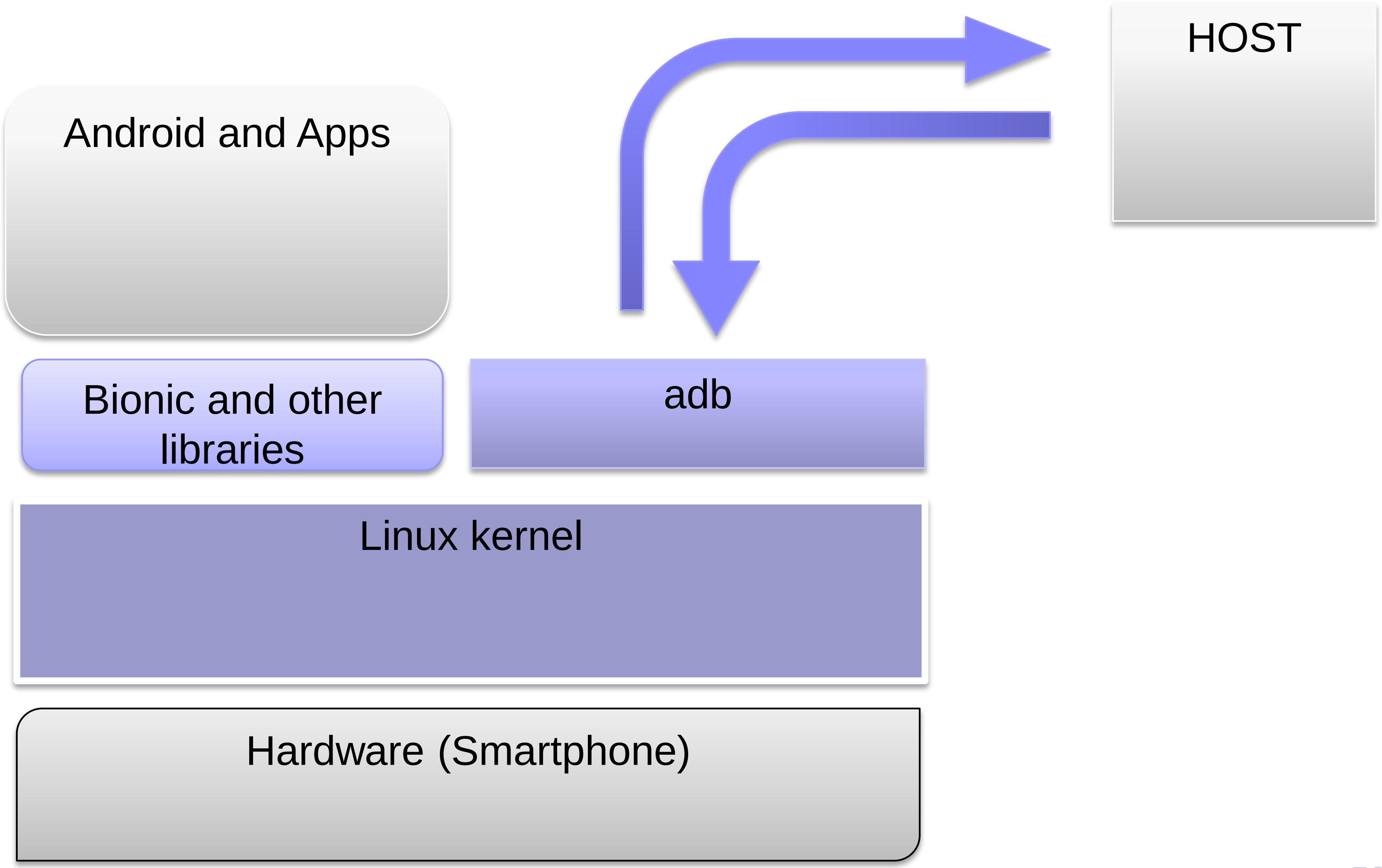
- Simple power management policies provide great energy performance tradeoffs
- Lower v-f setting offer worse e/p tradeoffs due to high performance delay
- Operating System can help create longer idle periods
- Research topics:
  - Multithreaded workload scheduling and power management
  - Speed up a core instead of slow down
  - Fast sleep times
  - Interaction with the rest of the system

Sources and References

- Jihong Kim, ES Class, SNU
- Peter Marwedel, “Embedded Systems Design,” 2004.
- Frank Vahid, Tony Givargis, “Embedded System Design,” Wiley, 2002.
- Wayne Wolf, “Computers as Components,” Morgan Kaufmann, 2001.
- Nikil Dutt @ UCI

Android/Linux Power Management and Class Projects

Experimental Setup

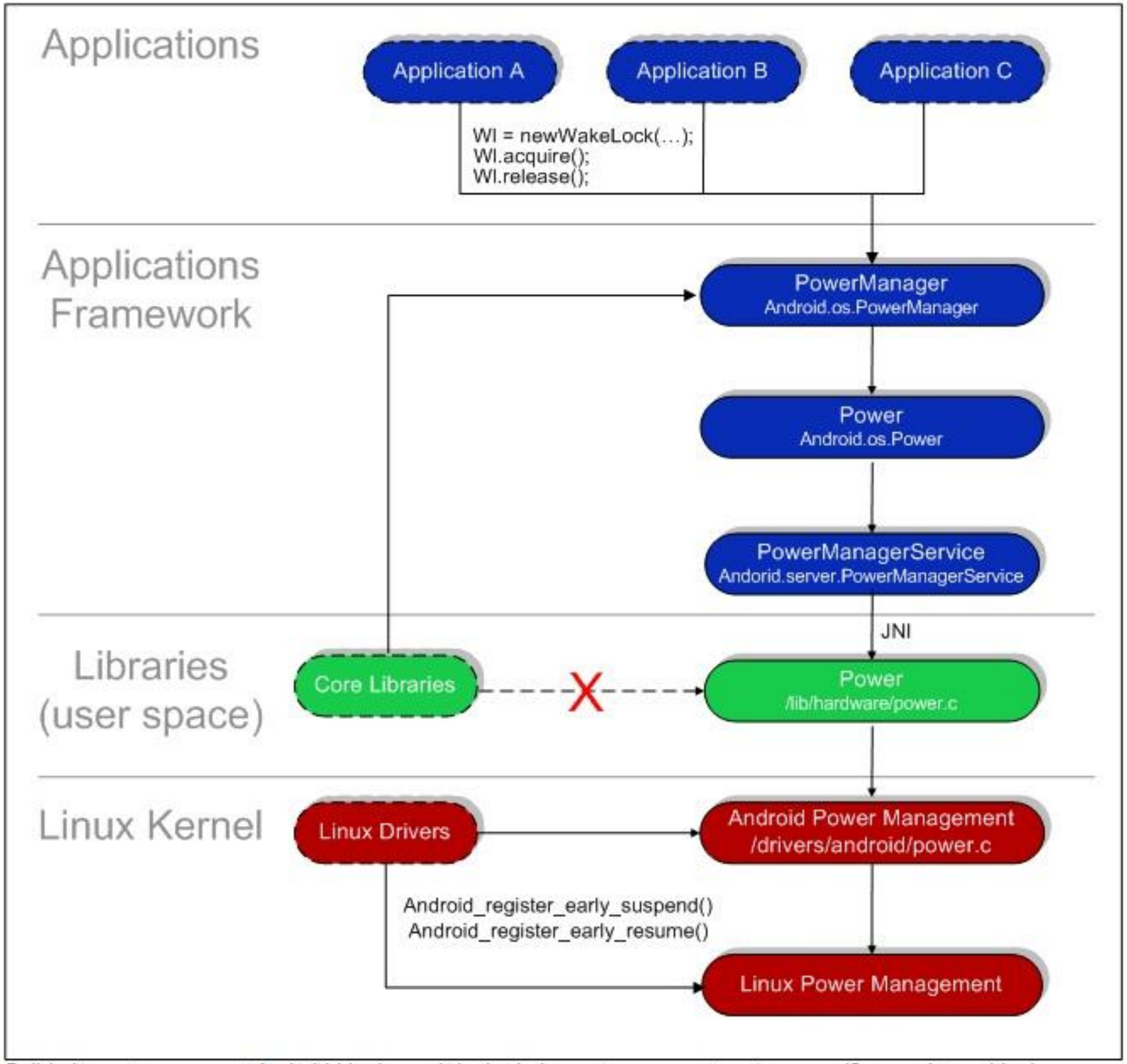


Android power management

- Android requires that applications and services request CPU (and other hardware resources) with "wake locks" through the Android application framework and native Linux libraries
- Examples
  - WAKE_LOCK_SUSPEND: prevents a full system suspend
  - WAKE_LOCK_IDLE: low-power states, which often cause large interrupt latencies or that disable a set of interrupts, will not be entered from idle until the wake locks are released
  - SCREEN_BRIGHT_WAKE_LOCK: Wake lock that ensures that the screen is on at full brightness; the keyboard backlight will be allowed to go off

http://www.netmite.com/android/mydroid/development/pdk/docs/power_management.html,
http://processors.wiki.ti.com/index.php/TI-Android-GingerBread-2.3.4-DevKit-2.1_PortingGuides#Power_Management

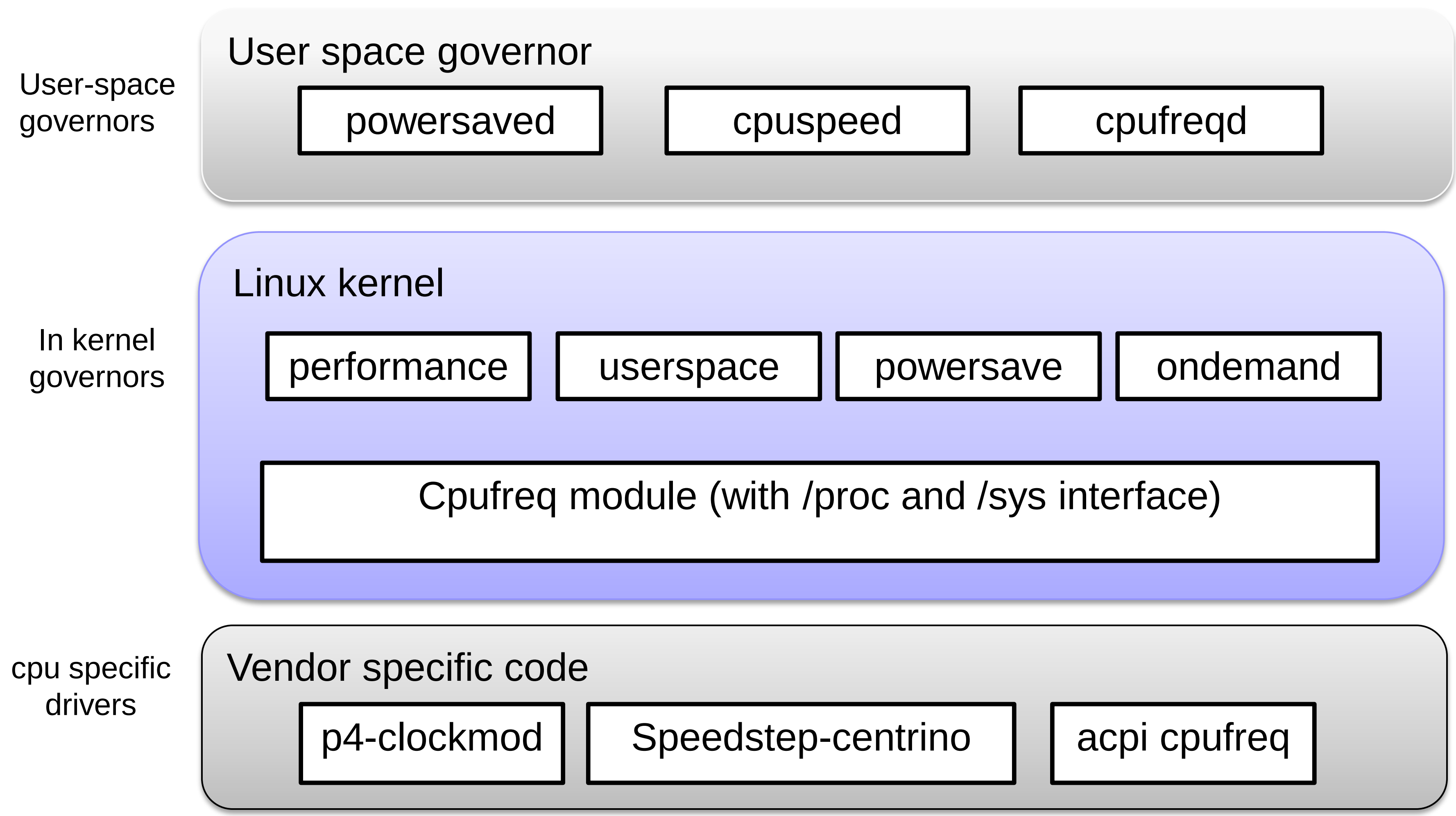
Android Power Management



How it works

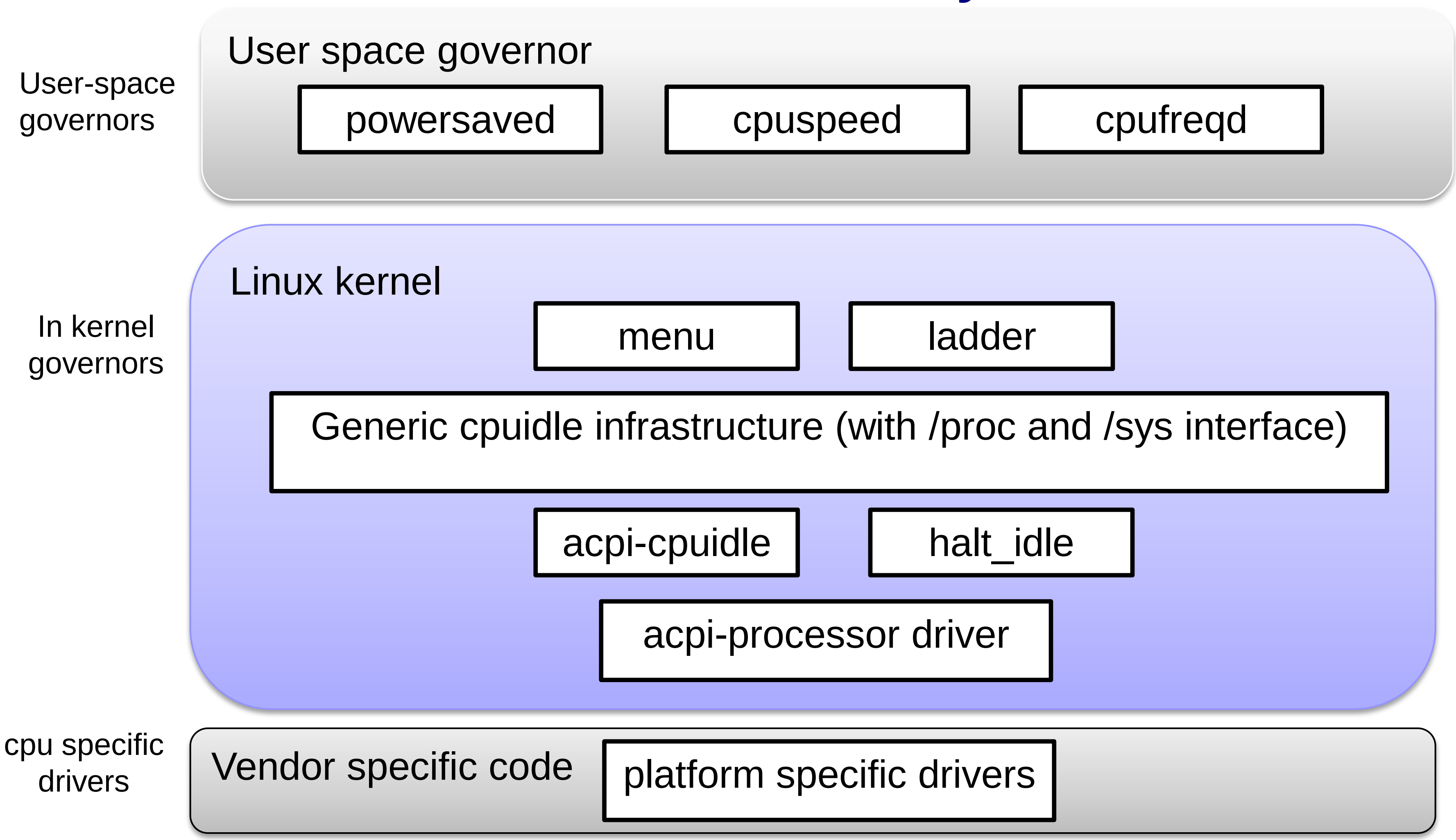
- **PowerManager class:**
  - The Android Framework exposes power management to services and applications through the PowerManager class
- **Registering Drivers with the PM Driver**
  - You can register Kernel-level drivers with the Android Power Manager driver so that they're notified immediately before power down or after power up
  - E.g. Display driver can be registered to completely power down when a request comes in to power down from the user space

Linux CPU Frequency Subsystem



http://software.intel.com/en-us/articles/enhanced-intel-speedstepr-technology-and-demand-based-switching-on-linux/

Linux Idle Subsystem



http://software.intel.com/en-us/articles/enhanced-intel-speedstepr-technology-and-demand-based-switching-on-linux/

Workloads

- SPEC benchmarks
  - Standardized applications
  - But not mobile specific
- Apps
  - Written with real market needs in mind
  - But not standardized
- Micro-benchmarks
  - Can be used to understand/study a specific mobile SOC subsystem
  - Easy to debug

Project Activities (all in parallel)

Data collection
Data Analysis
Understanding and coming up with power management policies

Project Activity: Data collection

- Session based
  - 2 to 5 mins
  - Multiple runs needed
  - For all three workloads
- Full ‘working day’ based
  - Full day logs
  - You will be the user

Part Activity: Data Analysis

- Use logs of real apps whenever possible
- As instructed use
  - SPEC benchmarks and
  - Micro-benchmarks

Project Activity: Understanding and coming up with your own power management policies

- Coming up with your own frequency governor
- TBA

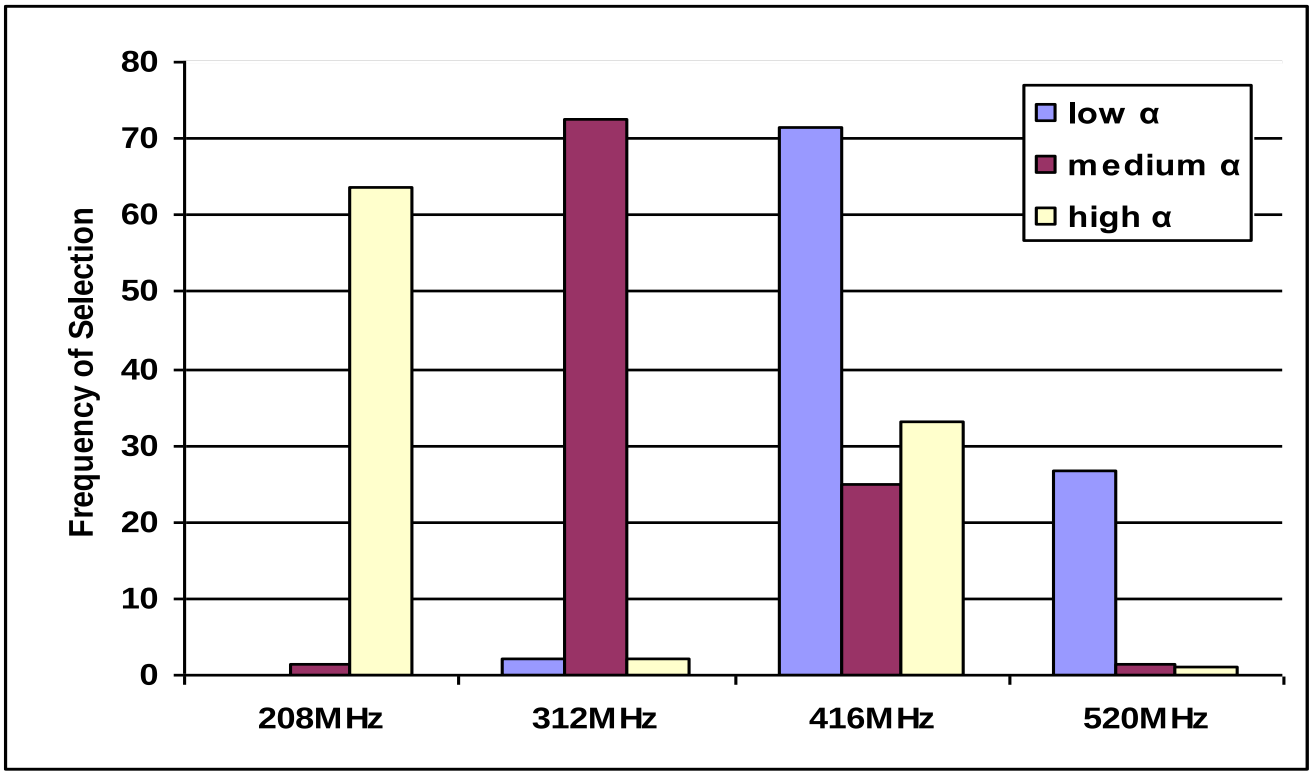
What you will learn through these projects

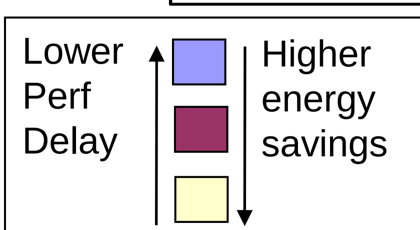
- You will get to learn about
  - Android and Linux kernel
  - Mobile SOC hardware capability
  - Linux CPU frequency subsystem
  - How to use cpu frequency subsystem for power management
  - How benchmarks and real apps behave
  - Understanding performance, power and ‘response time’ trade-offs

BACKUP

CPU: Frequency of Selection

For qsort





Identifies both CPU-intensive and mem intensive phases correctly