Algorithms for CPU and DRAM DVFS Under Inefficiency Constraints

Rizwana Begum
Drexel University

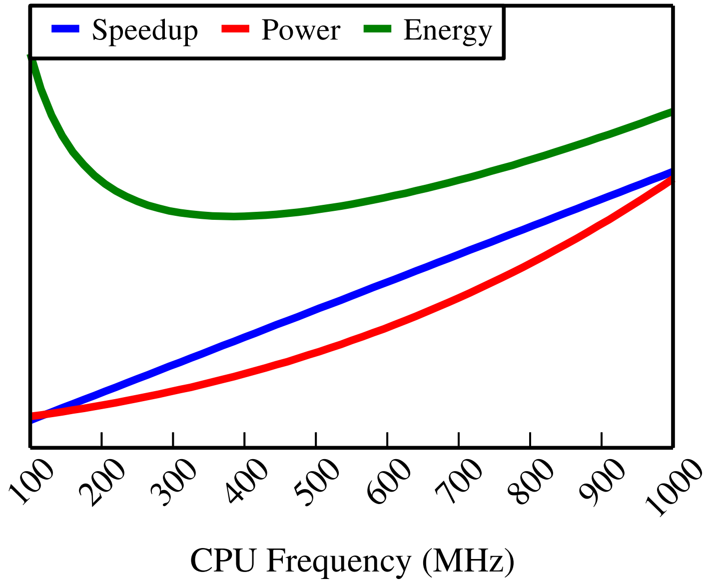
Mark Hempstead
Tufts University

Guru Prasad, Geoffrey Challen
University at Buffalo

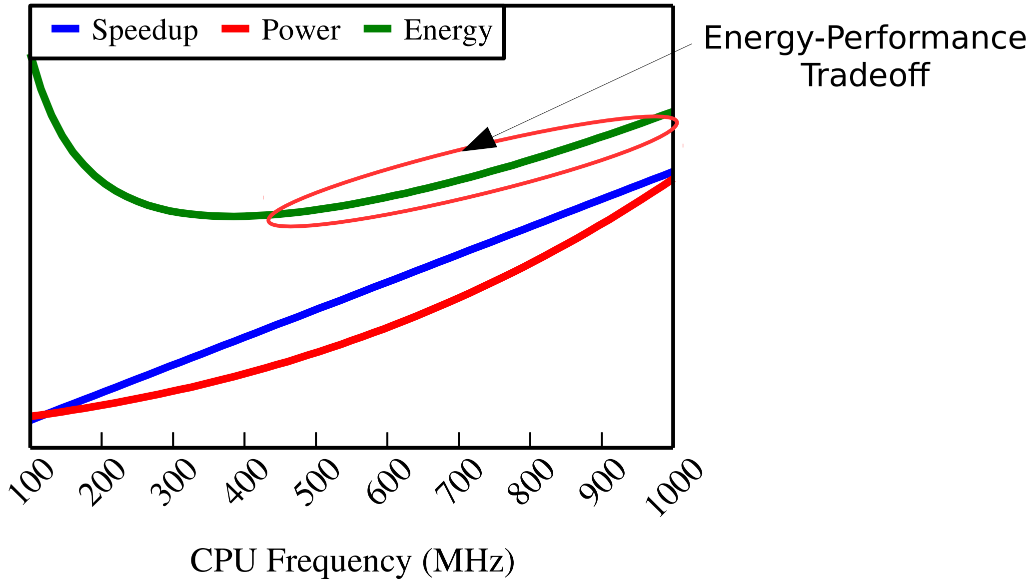
Oct 4, 2016

34th IEEE
International
Conference on
Computer Design
ICCD 2016

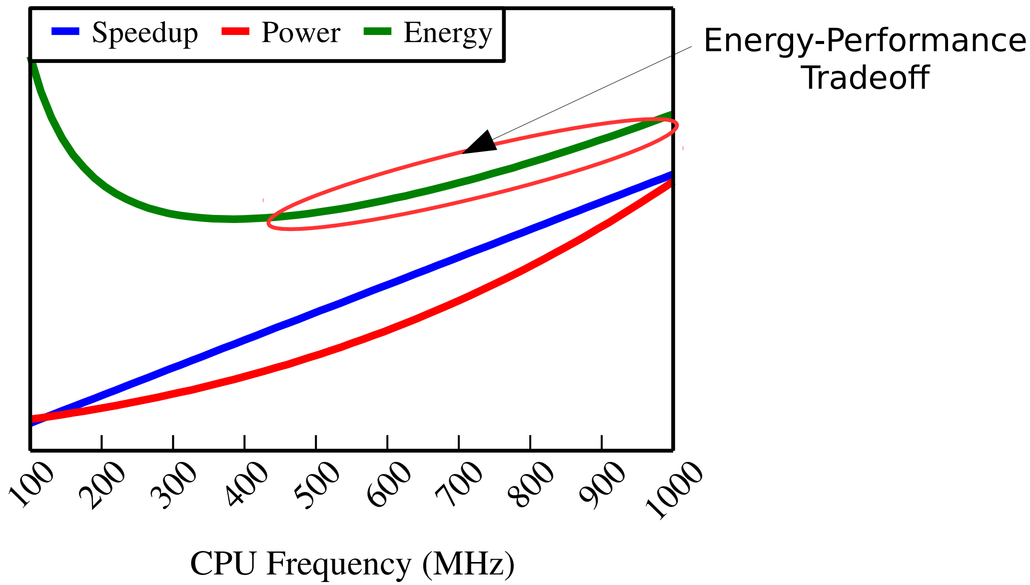
Dynamic Voltage and Frequency Scaling



Dynamic Voltage and Frequency Scaling



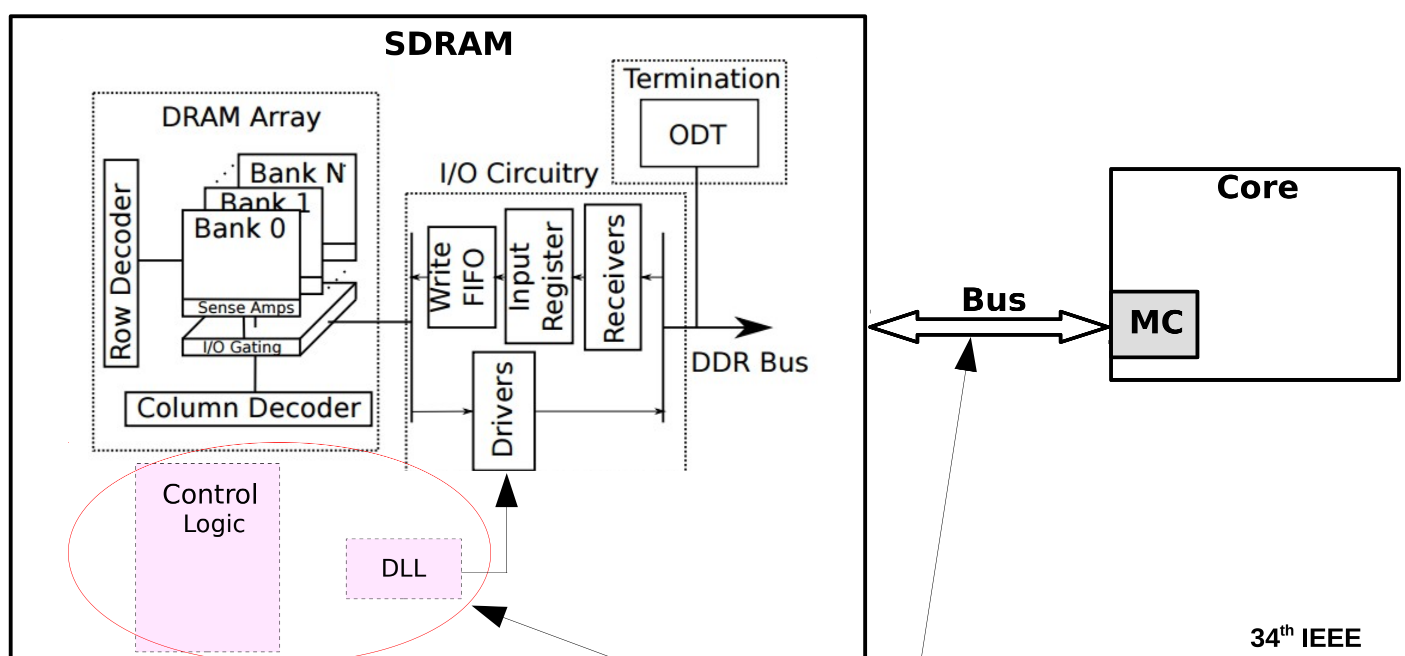
Dynamic Voltage and Frequency Scaling



SDRAM Frequency Scaling

DRAM DFS

- Only Frequency Scaling
- Performance and power are proportional to DRAM frequency
- Increase in energy with DRAM frequency is due to scalable and non-scalable parts of DRAM



David, Howard, et al. international conference on Autonomic computing. ACM, 2011.

CPU DVFS and Memory DFS

- Managing Systems - a challenging task

CPU DVFS and Memory DFS

- Managing Systems - a challenging task

- CPU intensive applications – higher CPU frequency
- Interplay of performance and energy of CPU and memory frequency scaling is complex

Performance vs. Energy Constraints

- Previous efforts explored DVFS under performance constraints

Performance vs. Energy Constraints

- Previous efforts explored DVFS under performance constraints
- Servers --- working under performance constraints is imperative
- Mobile systems –-- operating under energy constraints is fitting

Performance vs. Energy Constraints

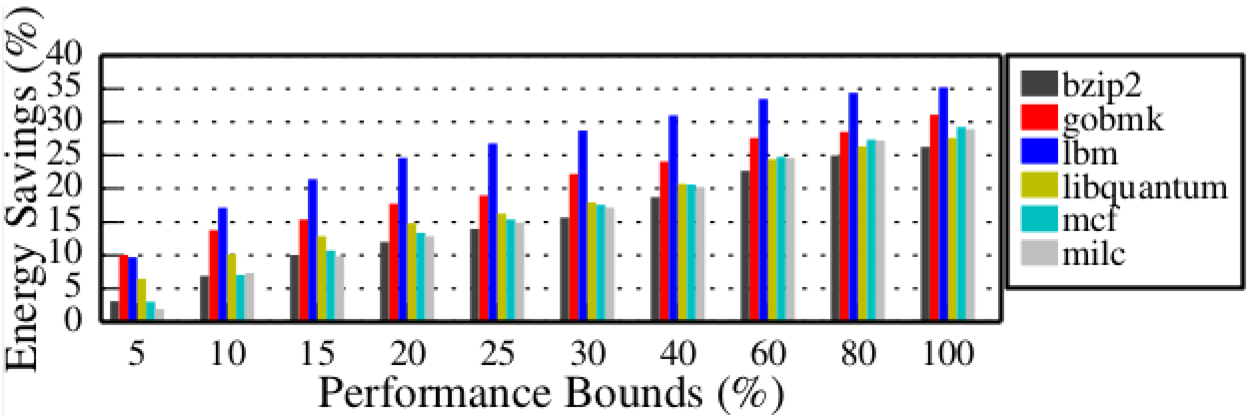
- Previous efforts explored DVFS under performance constraints
- Servers --- working under performance constraints is imperative
- Mobile systems –-- operating under energy constraints is fitting
- Approaches that work under performance constraints are not directly applicable to systems operating under energy constraints

Performance vs. Energy Constraints

- Choosing the right performance bound for desired energy savings is a daunting task --- application and device dependent

Performance vs. Energy Constraints

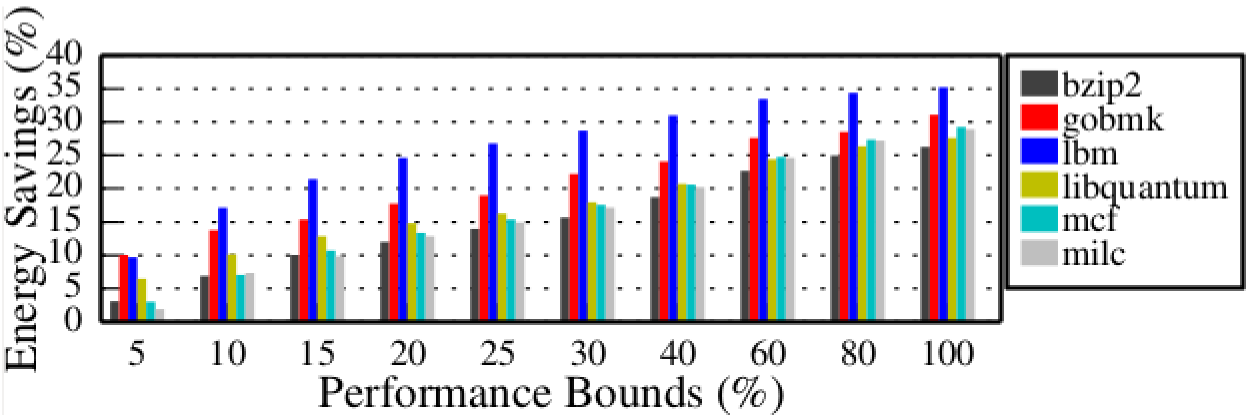
- Choosing the right performance bound for desired energy savings is a daunting task --- application and device dependent



- Energy savings achieved for a specific performance bound vary across applications

Performance vs. Energy Constraints

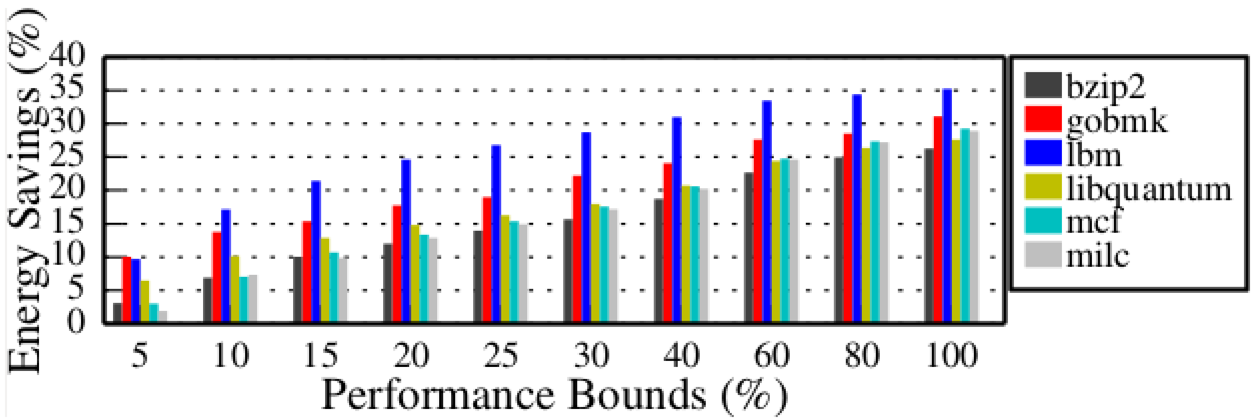
- Choosing the right performance bound for desired energy savings is a daunting task --- application and device dependent



- Energy savings achieved for a specific performance bound vary across applications
- Performance bound of 10% saves 5% - 15% energy
- 20% energy savings require performance bound of 15% - 50% depending upon the application

Performance vs. Energy Constraints

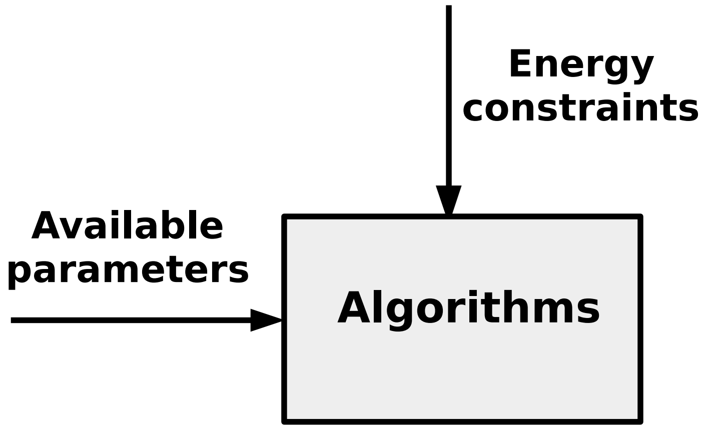
- Choosing the right performance bound for desired energy savings is a daunting task --- application and device dependent



- Energy savings achieved for a specific performance bound vary across applications
- Performance bound of 10% saves 5% - 15% energy
- 20% energy savings require performance bound of 15% - 50% depending upon the application

Demands a new energy management design that maximizes performance under given energy constraints

Energy Management Systems

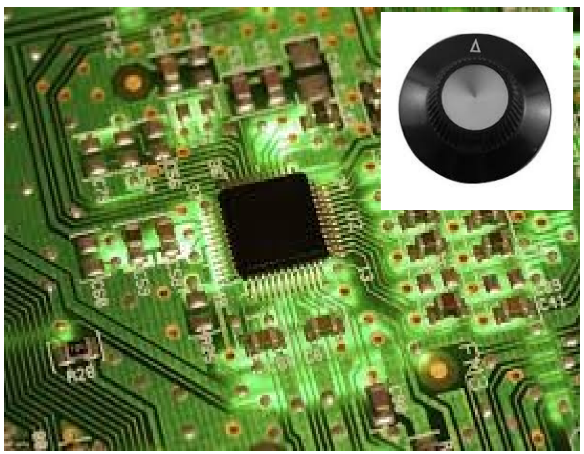


Energy Management Systems

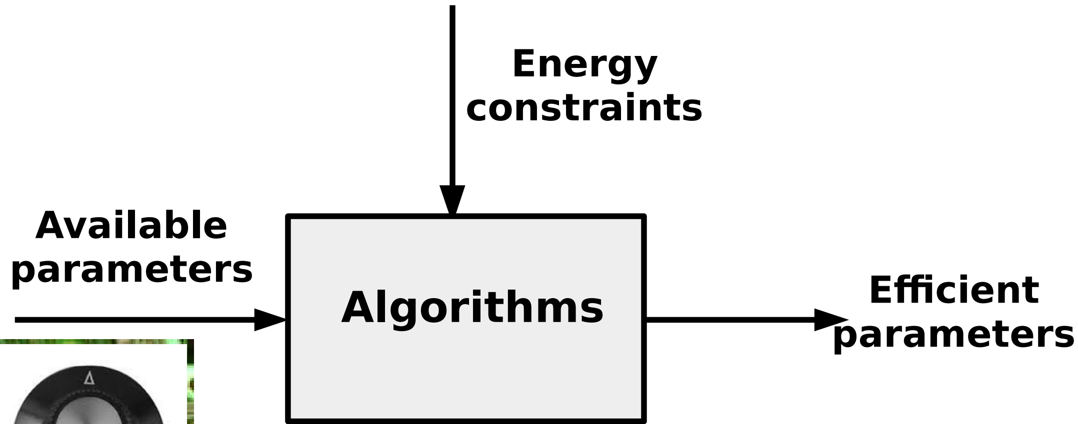
Energy constraints

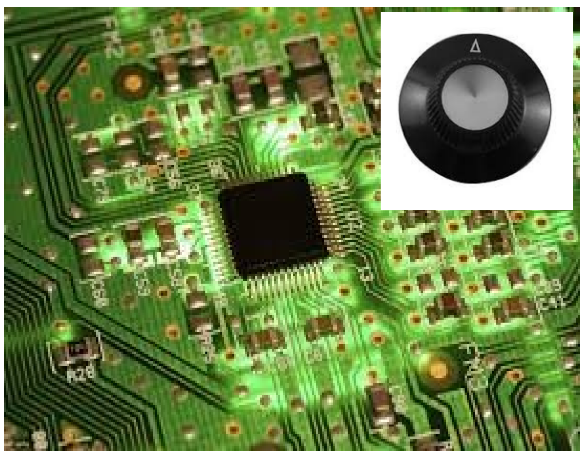
Available parameters

Algorithms

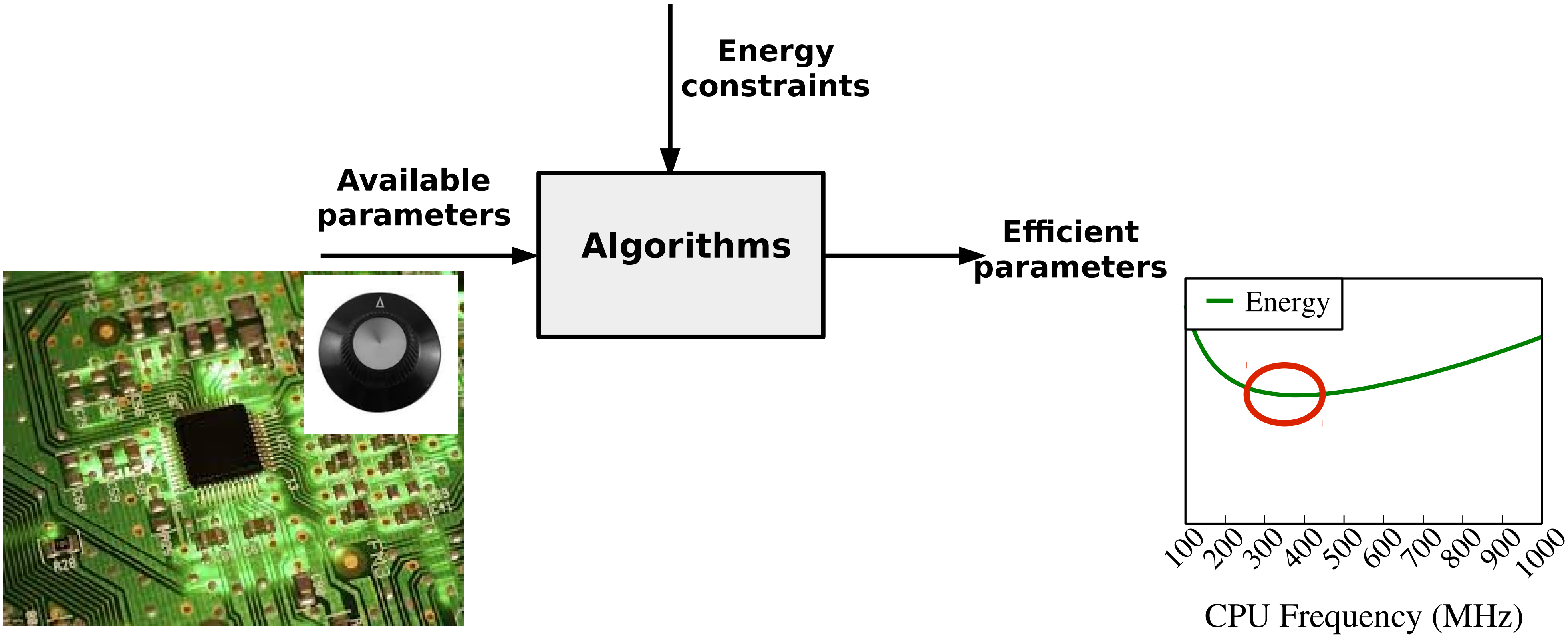


Energy Management Systems





Energy Management Systems



Outline

- Energy constraints
- System Design
- Performance and Energy Models
- Algorithms
- Results
- Conclusions

Energy Constraints

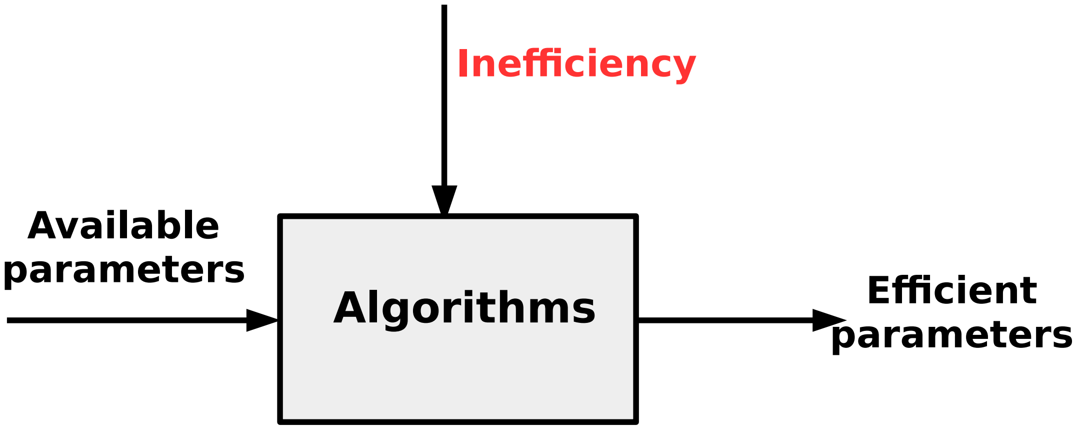
- Absolute energy or rate of energy consumption as energy constraints --- application and device dependent

Energy Constraints

- Absolute energy or rate of energy consumption as energy constraints --- application and device dependent
- Need for a new metric –-- ***Inefficiency***

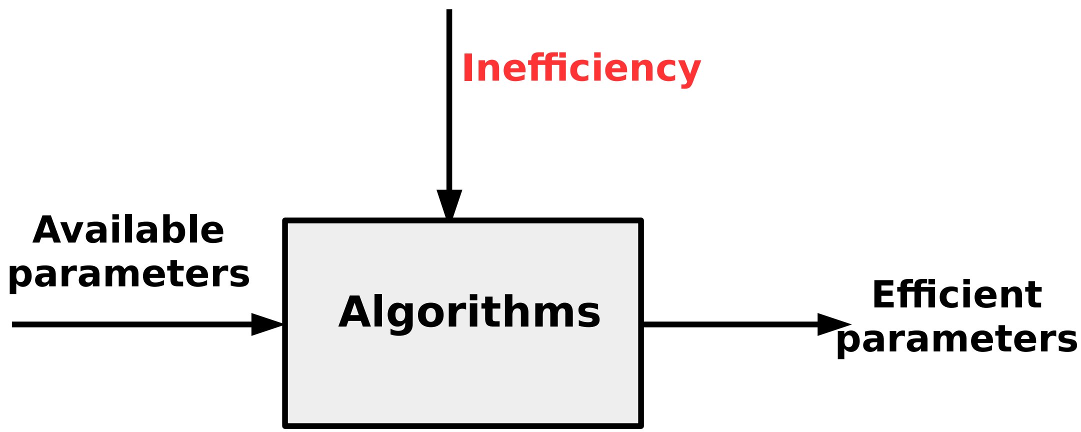
Inefficiency

- Inefficiency: Additional energy that can be used by the *application* to improve performance



Inefficiency

- Inefficiency: Additional energy that can be used by the *application* to improve performance

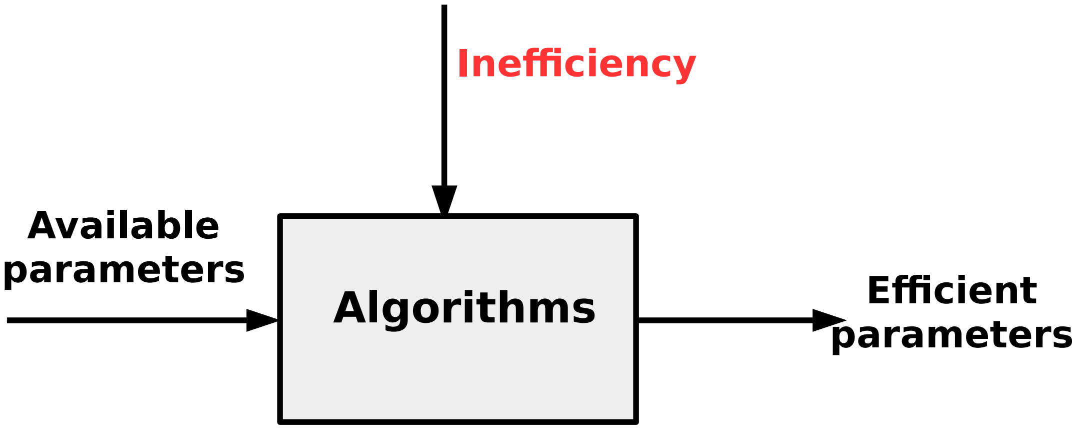


- Inefficiency:

$$\textbf{\textit{Inefficiency}} = \frac{E_{total}}{E_{min}}$$

Inefficiency

- Inefficiency: Additional energy that can be used by the *application* to improve performance



- Inefficiency:

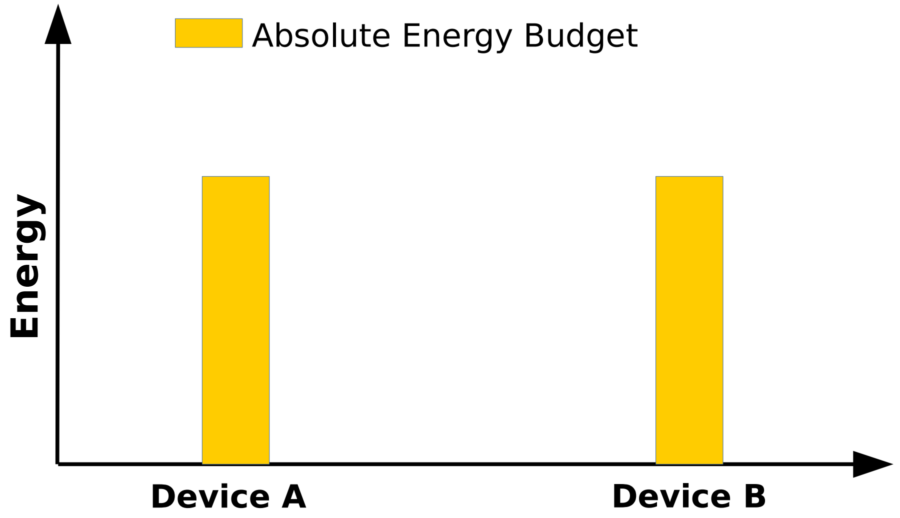
$$\textit{Inefficiency} = \frac{E_{total}}{E_{min}}$$

E_{min} – Minimum energy application could have consumed on the same device

E_{total} – Additional energy application can use to improve performance

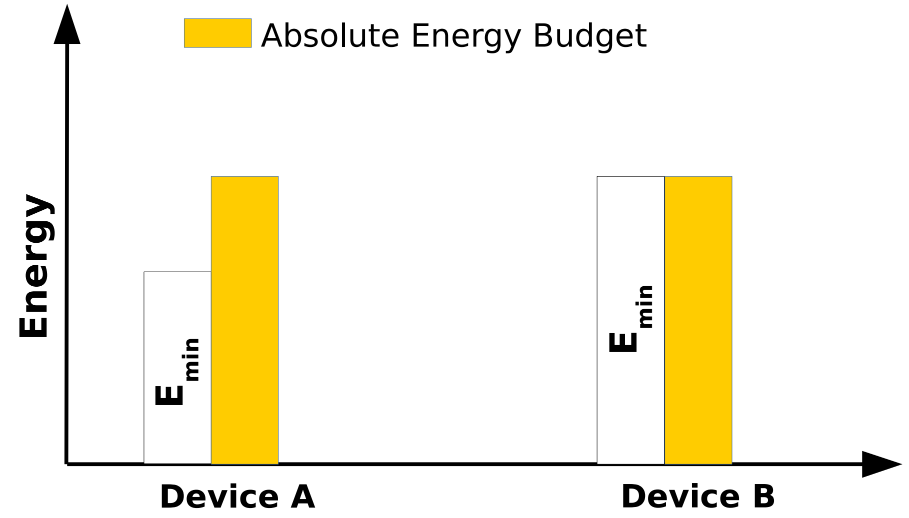
Inefficiency as a System Resource

- Agnostic to Devices



Inefficiency as a System Resource

- Agnostic to Devices



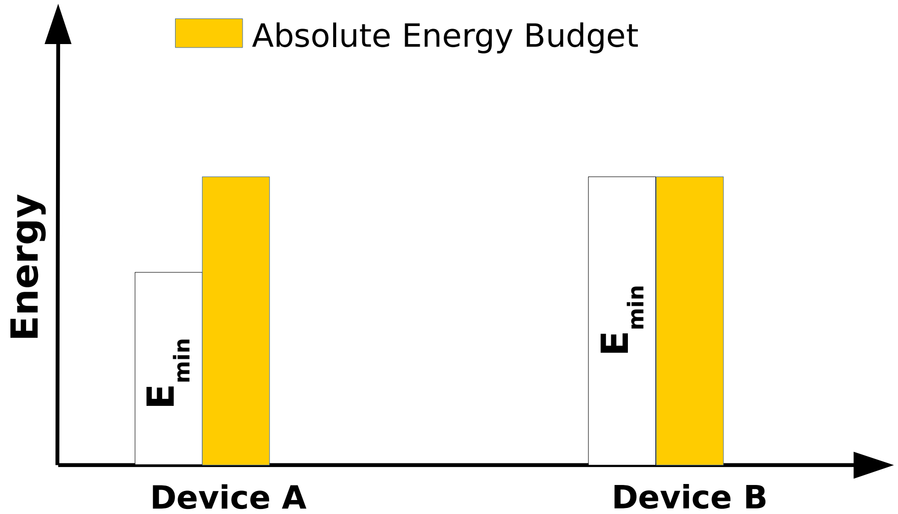
Inefficiency as a System Resource

- Agnostic to Devices

$$Inefficiency = \frac{E_{total}}{E_{min}}$$

⇓

$$E_{total} = Inefficiency \times E_{min}$$



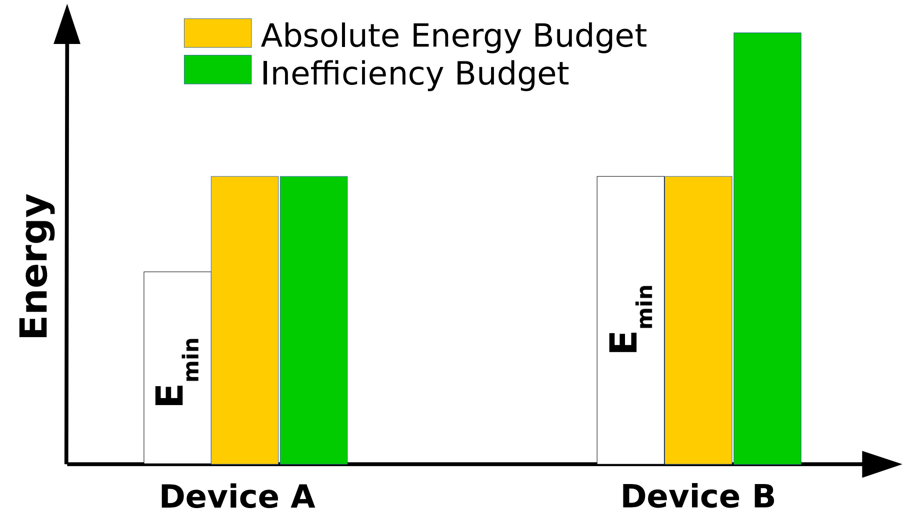
Inefficiency as a System Resource

- Agnostic to Devices

$$\textit{Inefficiency} = \frac{E_{total}}{E_{min}}$$

⇩

$$E_{total} = \textit{Inefficiency} \times E_{min}$$

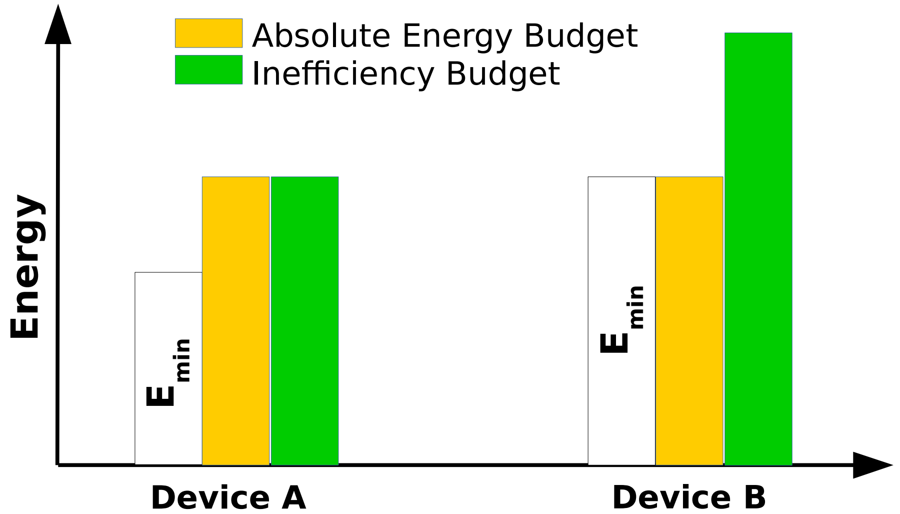


Inefficiency as a System Resource

- Agnostic to Devices

$$\textbf{Inefficiency} = \frac{E_{total}}{E_{min}}$$

$$E_{total} = \textbf{Inefficiency} \times E_{min}$$



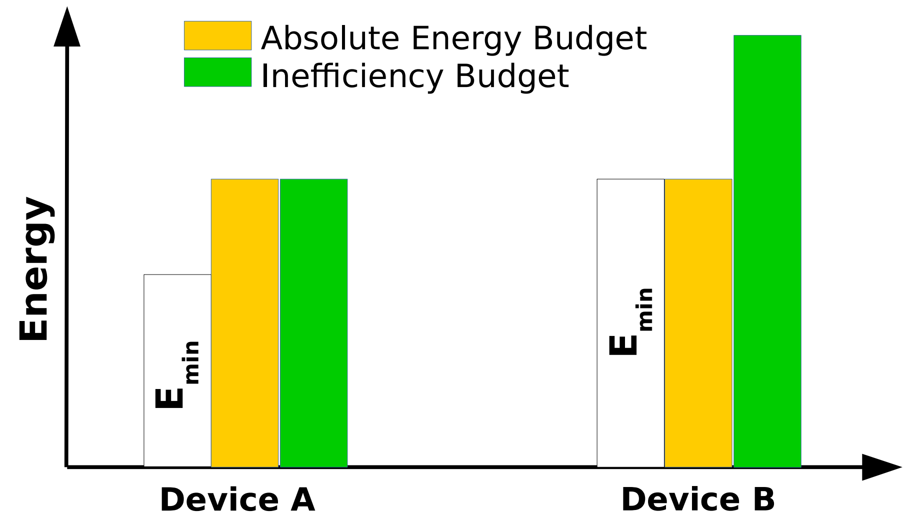
- Relative to inherent energy needs of the application
  - Inefficiency tied to priority of the applications

Inefficiency as a System Resource

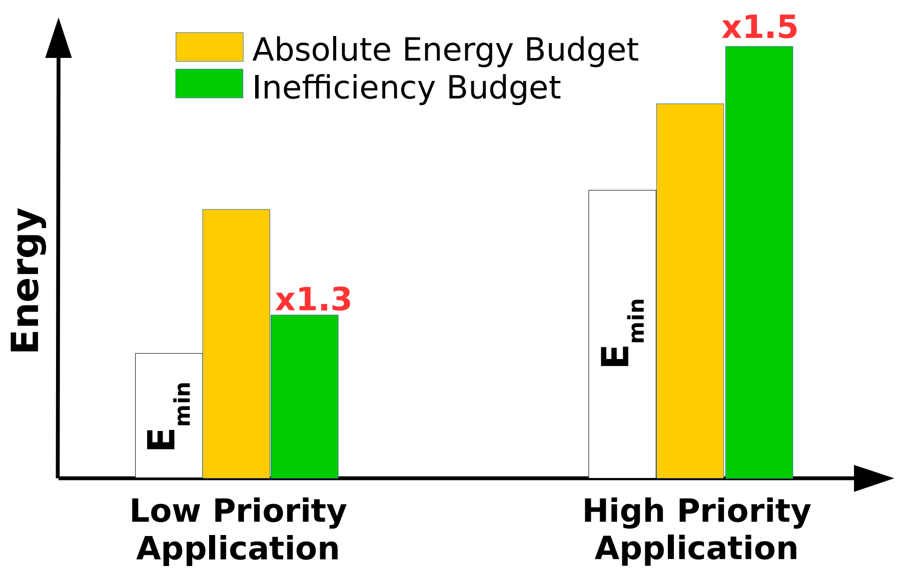
- Agnostic to Devices

$$\textit{Inefficiency} = \frac{E_{total}}{E_{min}}$$

$$E_{total} = \textit{Inefficiency} \times E_{min}$$



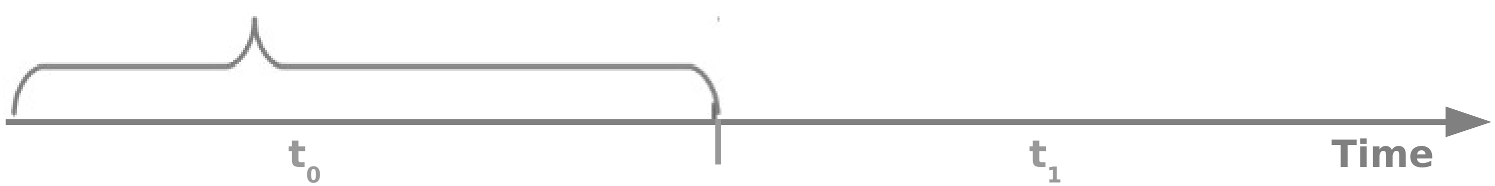
- Relative to inherent energy needs of the application
  - Inefficiency tied to priority of the applications



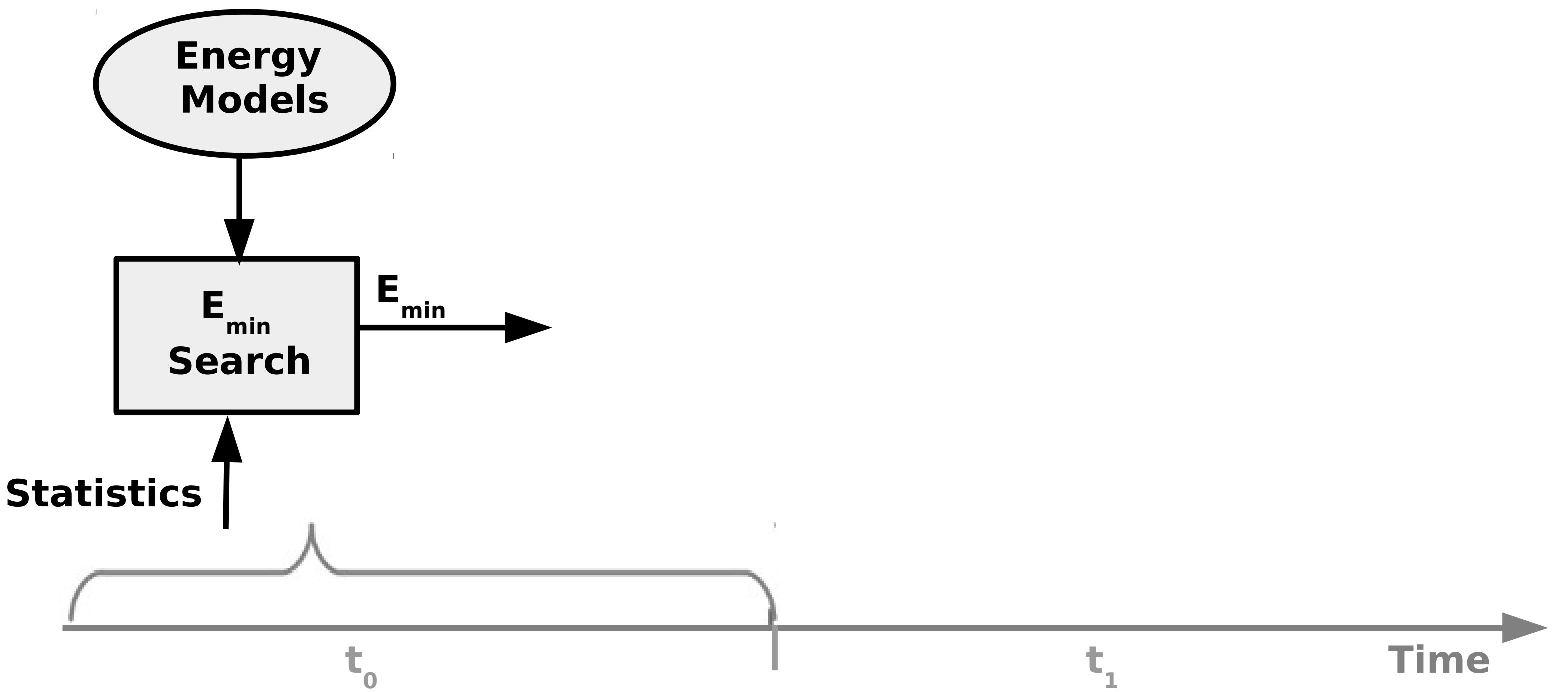
Outline

- Energy constraints
- **System Design**
- Performance and Energy Models
- Algorithms
- Results
- Conclusions

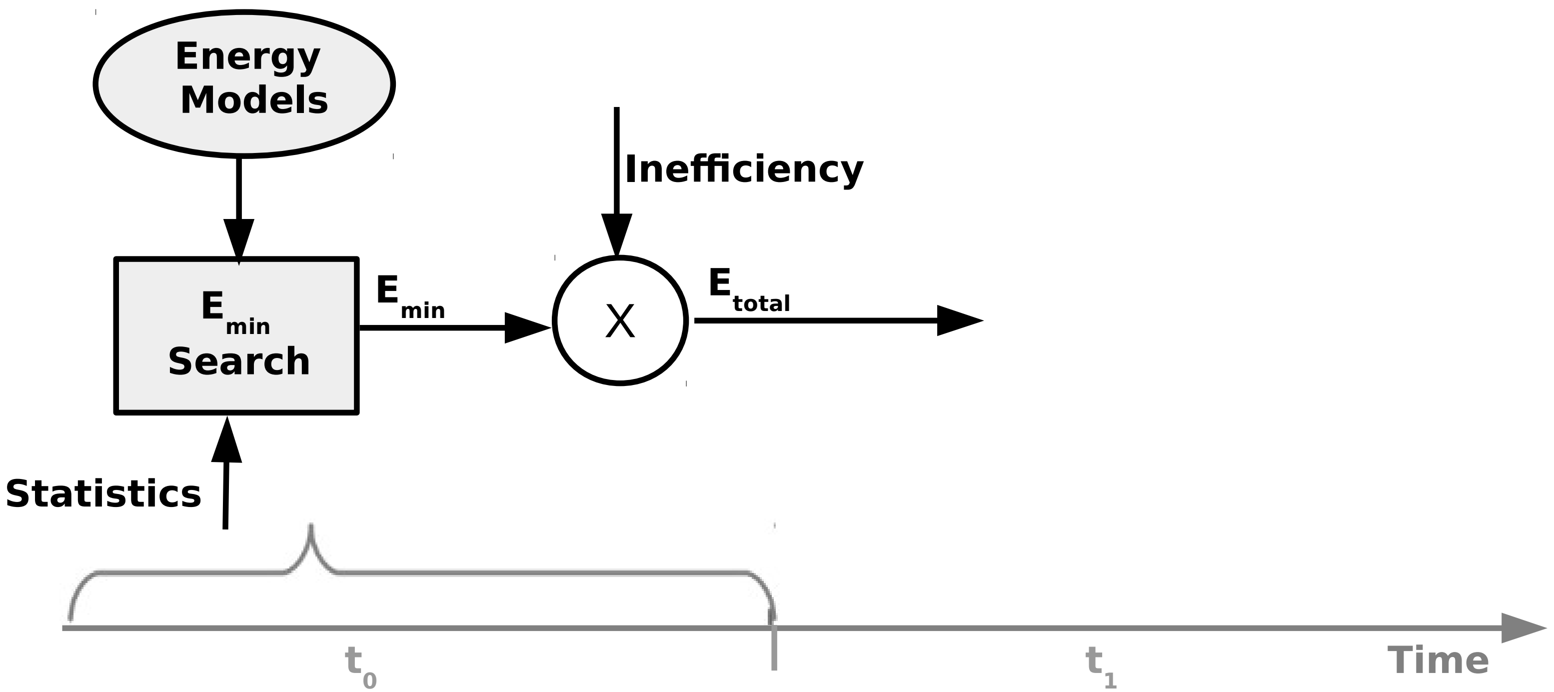
System Design



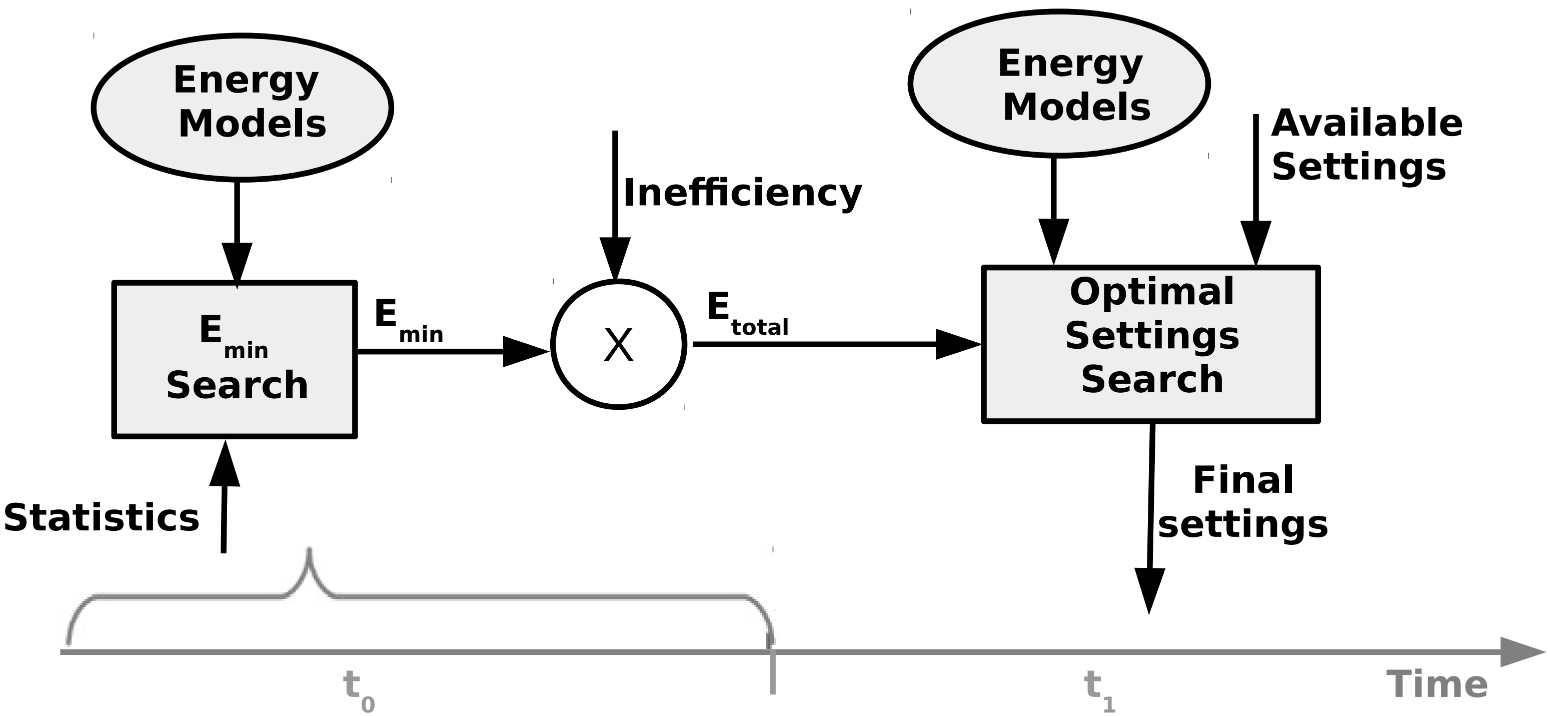
System Design



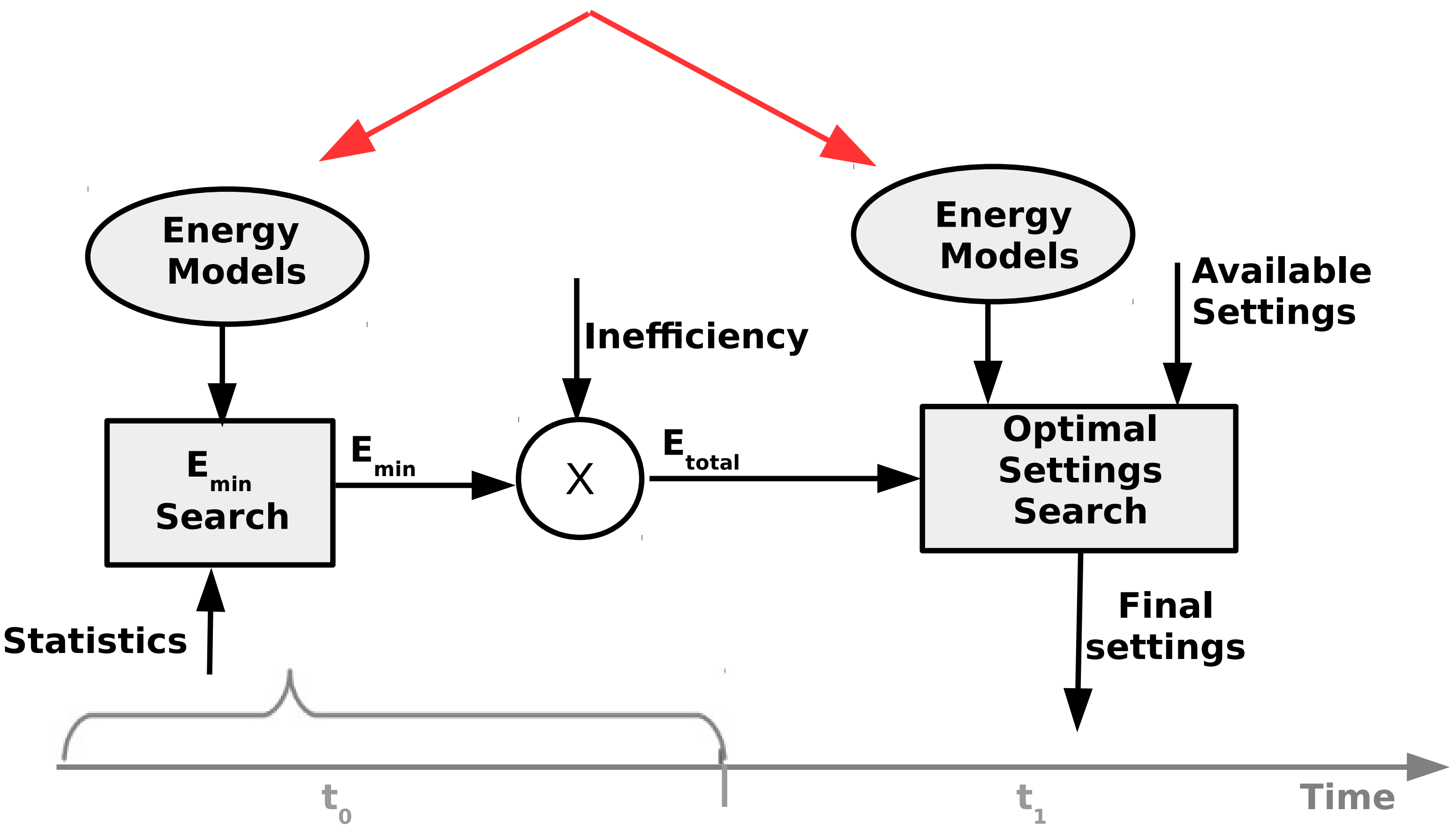
System Design



System Design



System Design



Outline

- Energy constraints
- System Design
- **Performance and Energy Models**
- Algorithms
- Results
- Conclusions

Performance and Energy Models

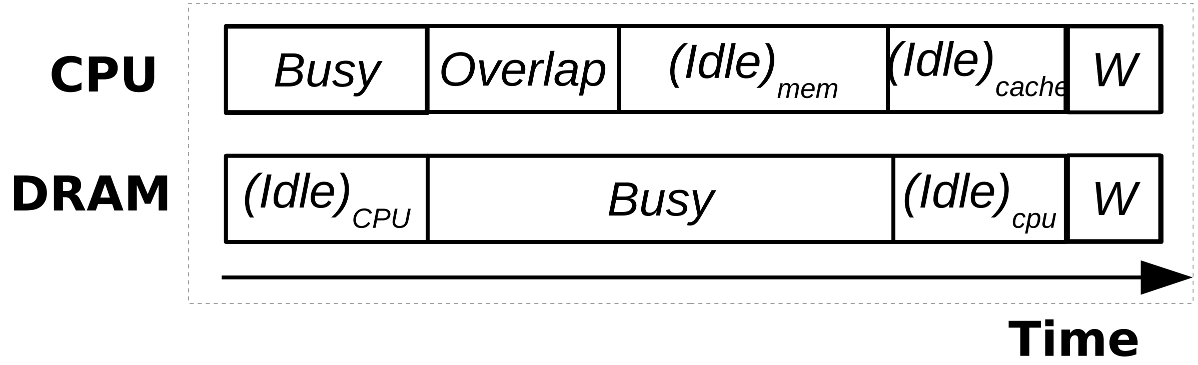
- Developed cross-component energy models that consider the impact of scaling frequency of one component on the performance and energy of the other component

Performance and Energy Models

- Developed cross-component energy models that consider the impact of scaling frequency of one component on the performance and energy of the other component
- CPU and DRAM execution time is divided into 3 states: *Busy, Idle* and *Waiting*

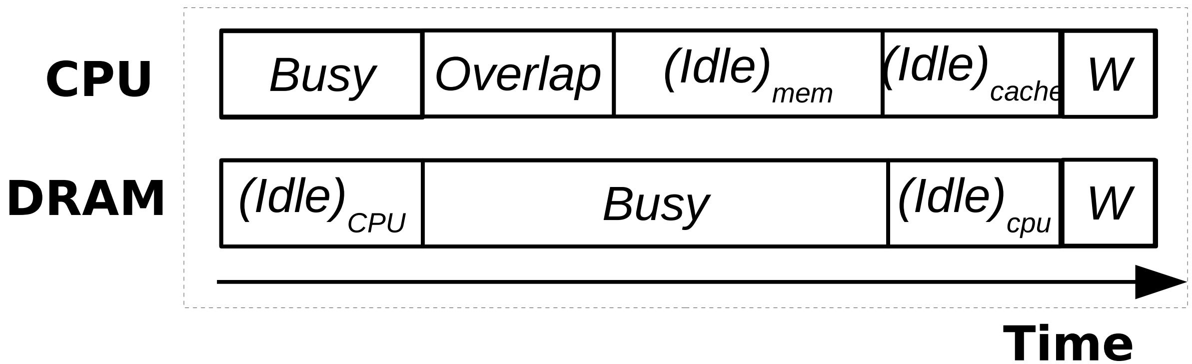
Performance and Energy Models

- Developed cross-component energy models that consider the impact of scaling frequency of one component on the performance and energy of the other component
- CPU and DRAM execution time is divided into 3 states: *Busy, Idle* and *Waiting*



Performance and Energy Models

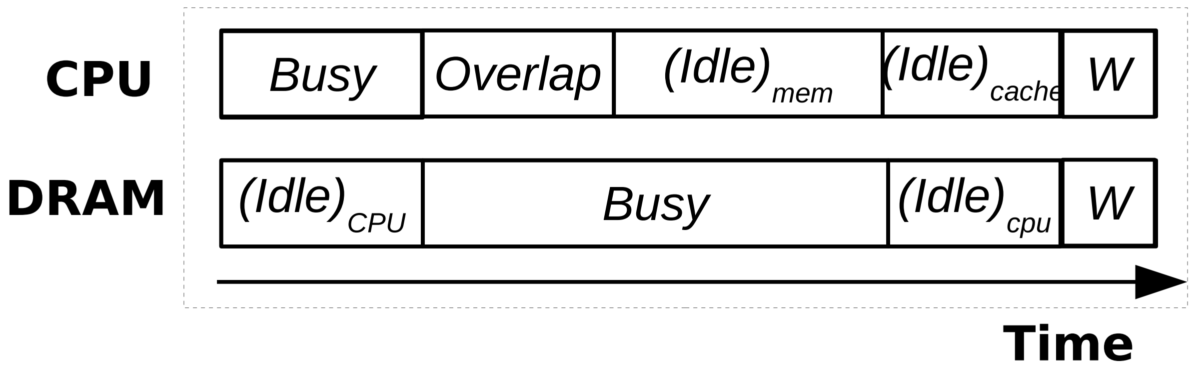
- Developed cross-component energy models that consider the impact of scaling frequency of one component on the performance and energy of the other component
- CPU and DRAM execution time is divided into 3 states: *Busy, Idle* and *Waiting*



- *Busy* scales with component's own frequency, *idle* is a function of other component, and *Overlap* is limited by the speed of the slowest component

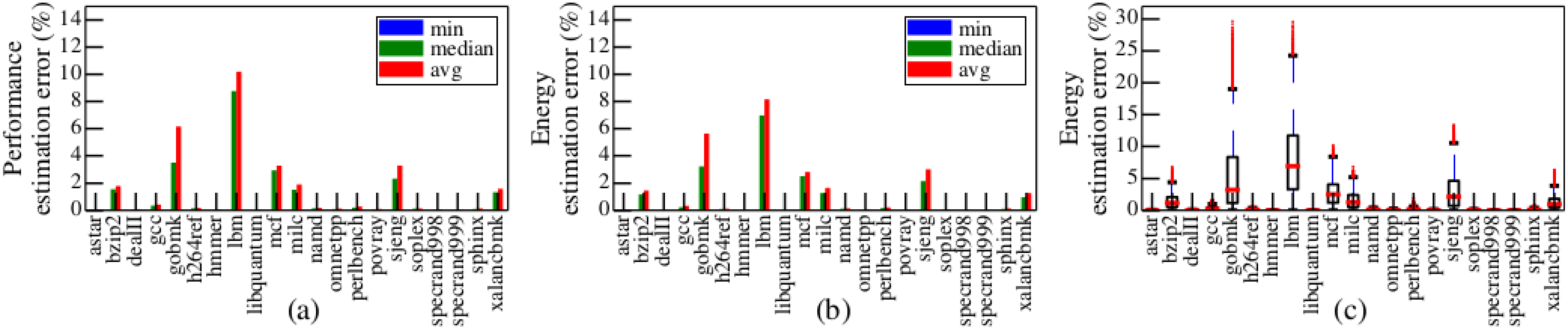
Performance and Energy Models

- Developed cross-component energy models that consider the impact of scaling frequency of one component on the performance and energy of the other component
- CPU and DRAM execution time is divided into 3 states: *Busy, Idle* and *Waiting*



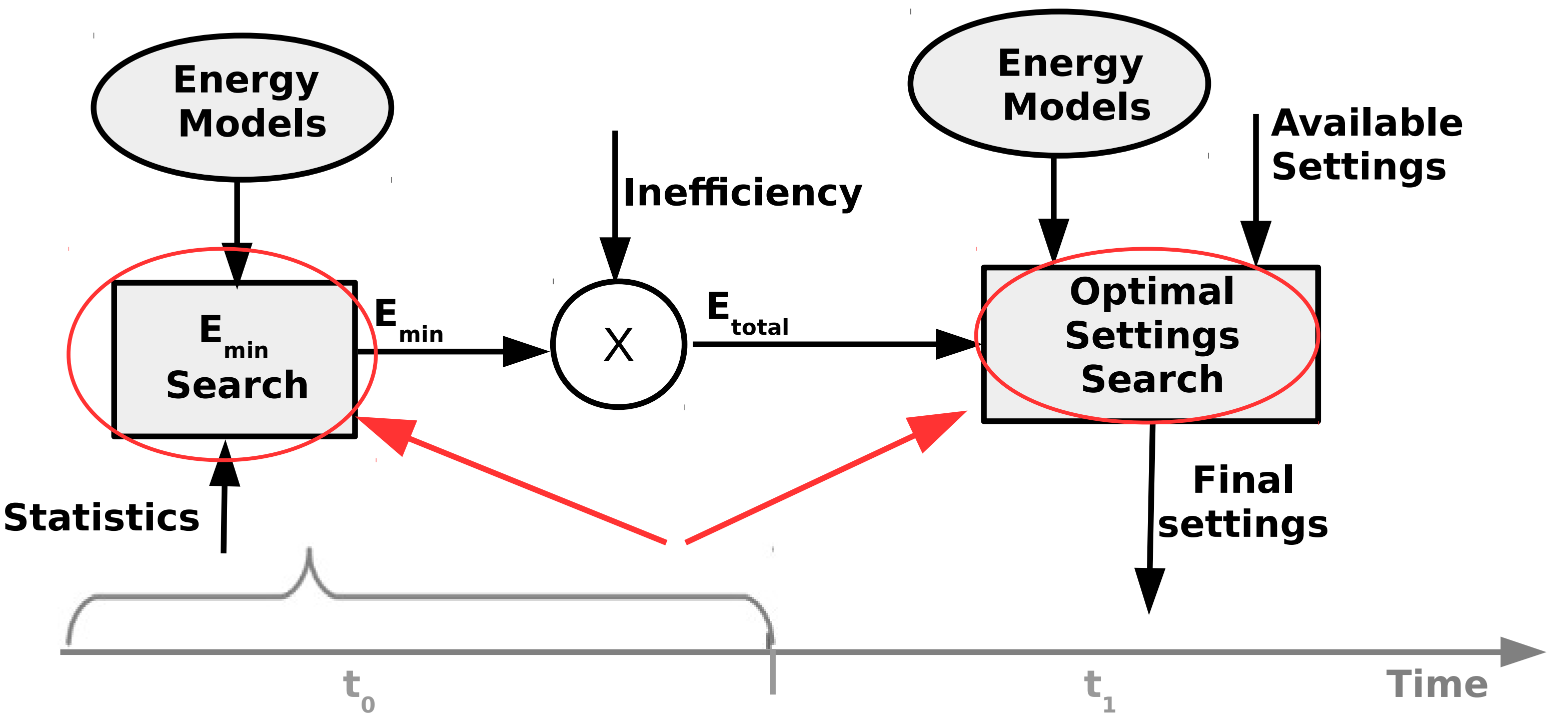
- *Busy* scales with component's own frequency, *idle* is a function of other component, and *Overlap* is limited by the speed of the slowest component
- A total of 14 performance counters are used to track the time spent in each state
- 7 performance counters are newly added, 7 are already included in modern systems
- More details in the paper

Performance and Energy Models



- 12 integer and 9 floating point SPEC CPU2006 benchmarks --- using Gem5 simulator
- Average error is less than 4% except for *Gobmk (6%)* and *Lbm (10%)*
- Model is more than 97% accurate for more than 50% predictions
- Maximum error is less than 10% except for *Gobmk(18%)* and *Lbm(24%)*

System Design



Outline

- Energy constraints
- System Design
- Performance and Energy Models
- **Algorithms**
- Results
- Conclusions

Algorithms

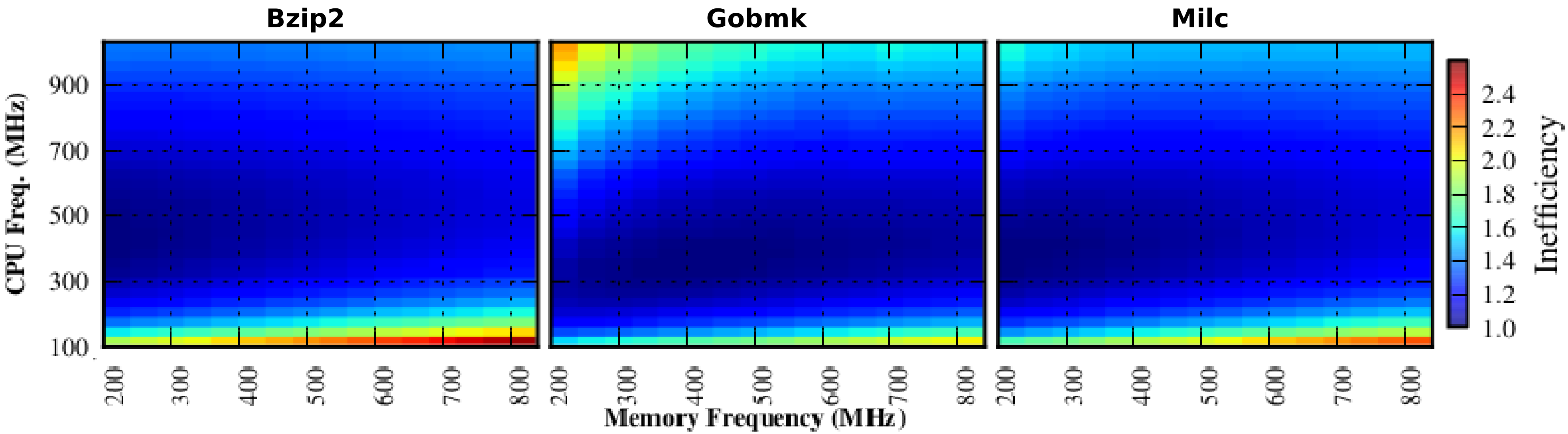
- Energy management algorithms perform two search operations
  - Search for E_{min}
  - Search for Optimal Settings

Algorithms

- Energy management algorithms perform two search operations
  - Search for E_{min}
  - Search for Optimal Settings
- Higher overhead compared to algorithms optimizing under performance constraints as only the second search --- for optimal settings --- is performed

Algorithms

- Energy management algorithms perform two search operations
  - Search for E_{min}
  - Search for Optimal Settings
- Higher overhead compared to algorithms optimizing under performance constraints as only the second search --- for optimal settings --- is performed
- E_{min} moves every tuning interval making the search complex and expensive



Algorithms

➢ We classify our algorithms into two categories

Algorithms

- We classify our algorithms into two categories
  - Search --- search for E_{min} and *cluster* of settings that fall under given inefficiency budget
  - Select --- selects optimal frequency settings from the *cluster*

Algorithms

- We classify our algorithms into two categories
  - Search --- search for E_{min} and *cluster* of settings that fall under given inefficiency budget
  - Select --- selects optimal frequency settings from the *cluster*
- We introduce relative and adaptive algorithms to reduce tuning overhead

Algorithms

- We classify our algorithms into two categories
  - Search --- search for E_{min} and *cluster* of settings that fall under given inefficiency budget
  - Select --- selects optimal frequency settings from the *cluster*
- We introduce relative and adaptive algorithms to reduce tuning overhead
  - Search ---- 1) Exhaustive Search 2) Relative Search
  - Select ---- 1) Best Performing Selection 2) Adaptive Selection

Search Algorithms

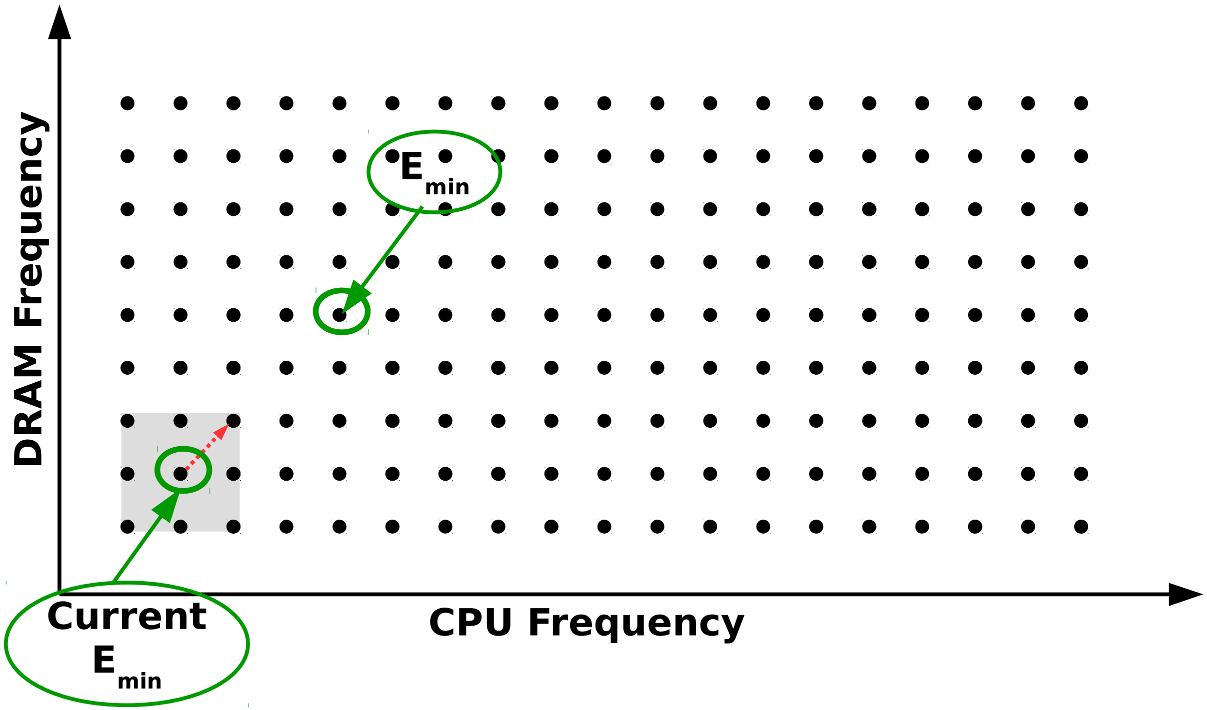
- Exhaustive Search
  - Searches entire space of core and DRAM frequency settings

Search Algorithms

- Exhaustive Search
  - Searches entire space of core and DRAM frequency settings
- Relative Search
  - Reduces tuning cost by starting from previous settings

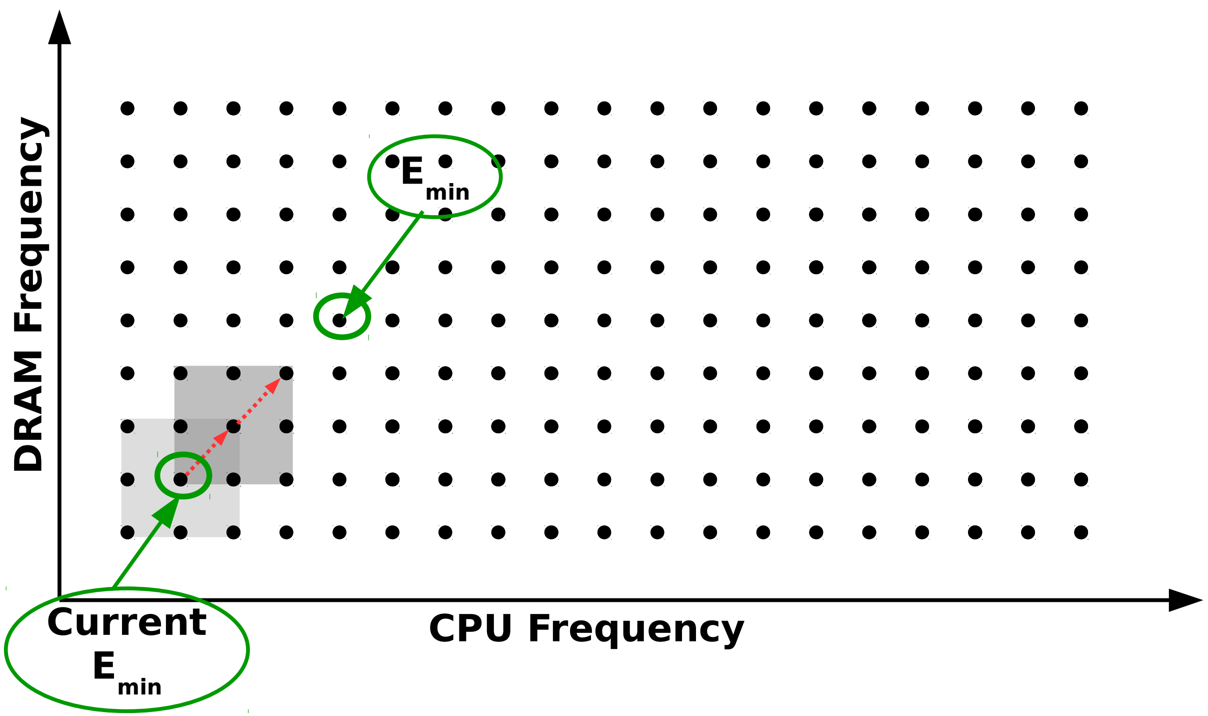
Search Algorithms

- Exhaustive Search
  - Searches entire space of core and DRAM frequency settings
- Relative Search
  - Reduces tuning cost by starting from previous settings



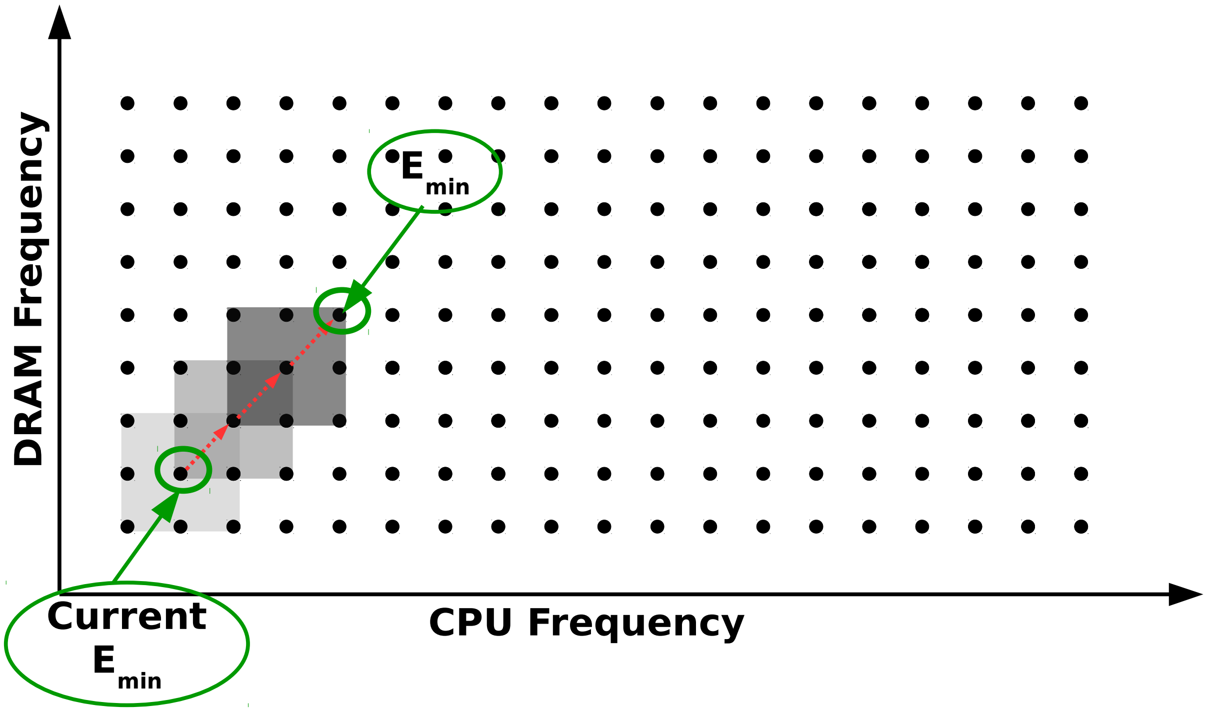
Search Algorithms

- Exhaustive Search
  - Searches entire space of core and DRAM frequency settings
- Relative Search
  - Reduces tuning cost by starting from previous settings



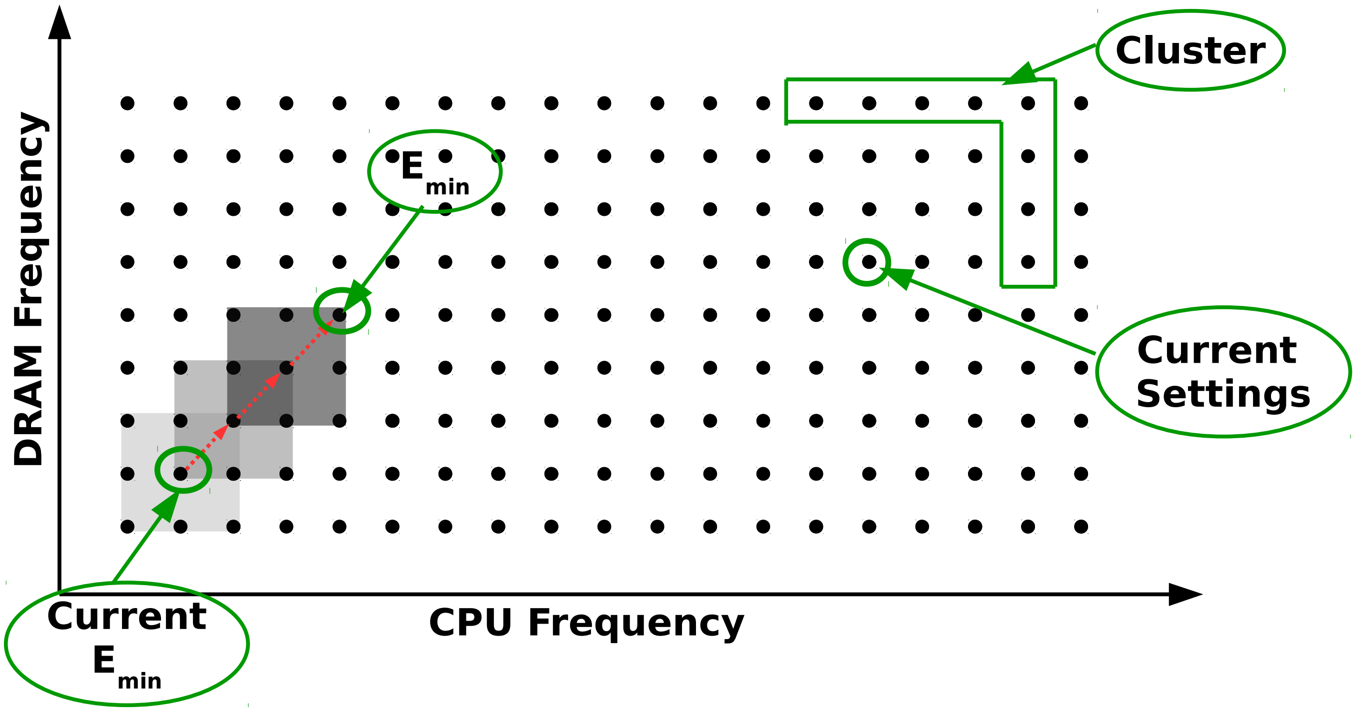
Search Algorithms

- Exhaustive Search
  - Searches entire space of core and DRAM frequency settings
- Relative Search
  - Reduces tuning cost by starting from previous settings



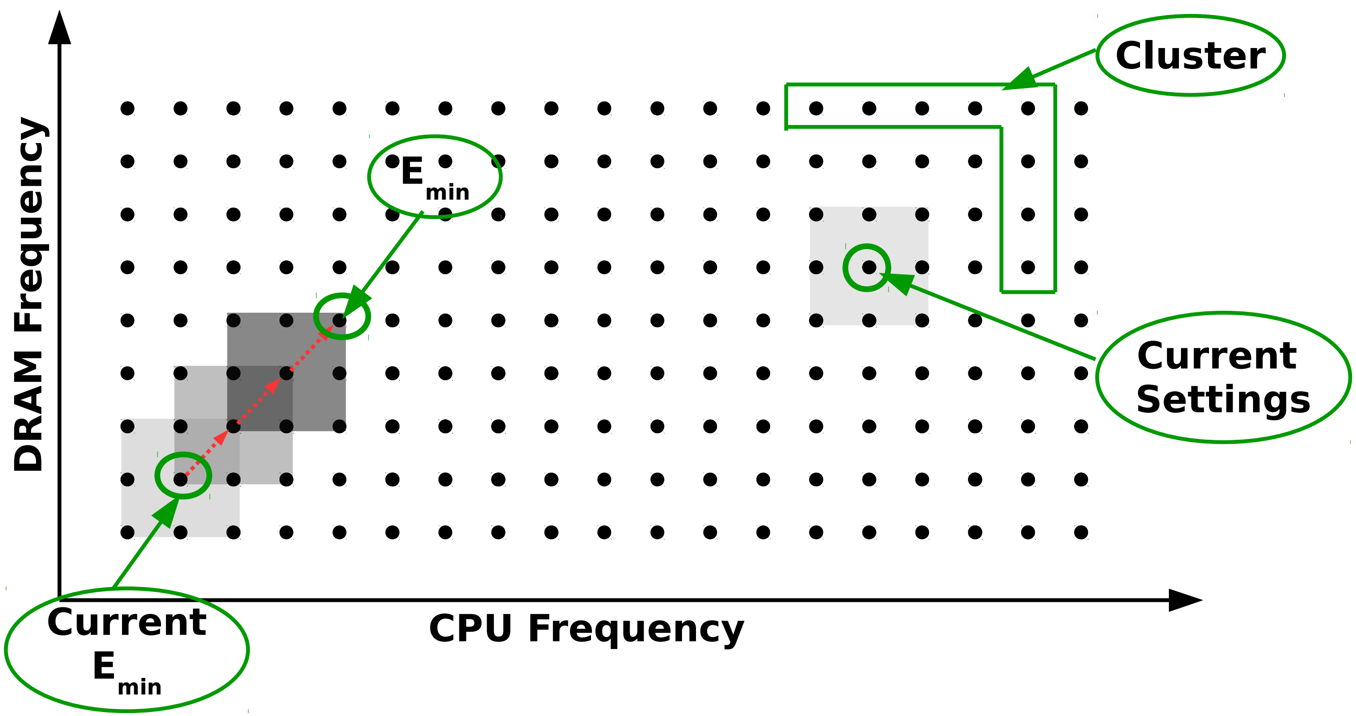
Search Algorithms

- Exhaustive Search
  - Searches entire space of core and DRAM frequency settings
- Relative Search
  - Reduces tuning cost by starting from previous settings



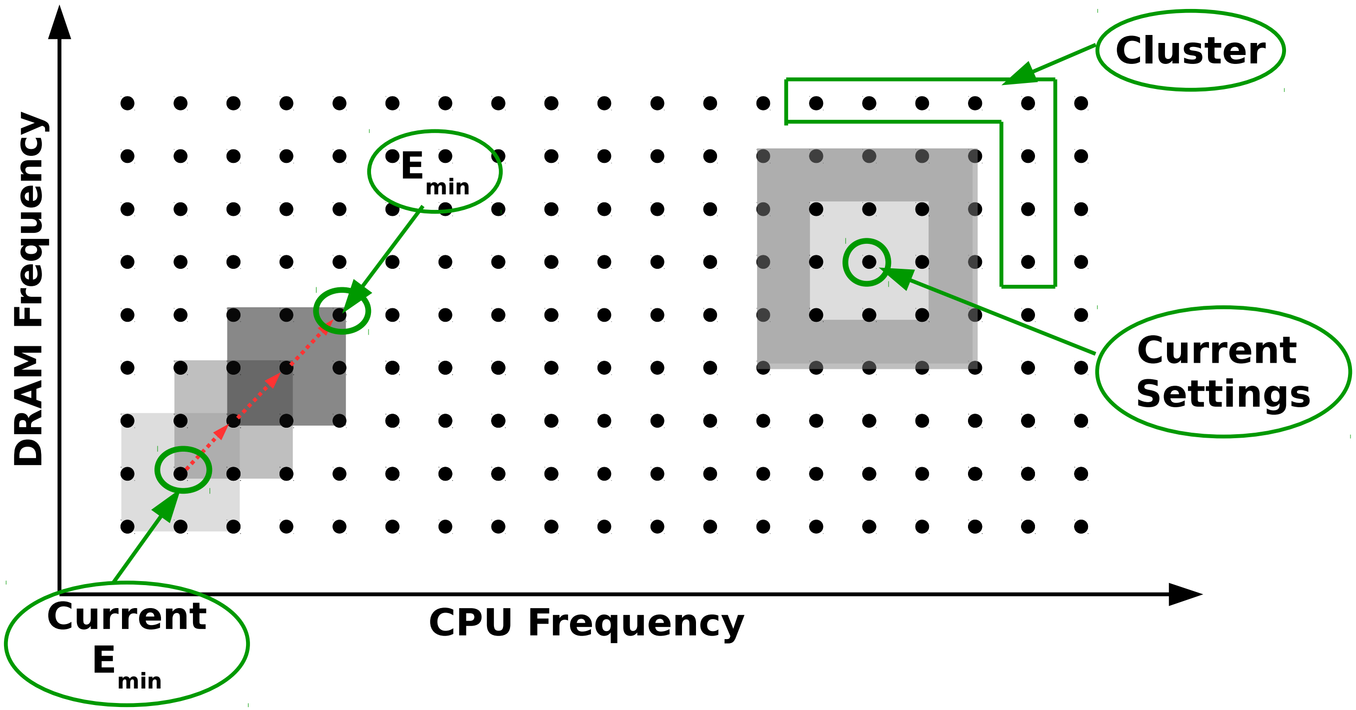
Search Algorithms

- Exhaustive Search
  - Searches entire space of core and DRAM frequency settings
- Relative Search
  - Reduces tuning cost by starting from previous settings



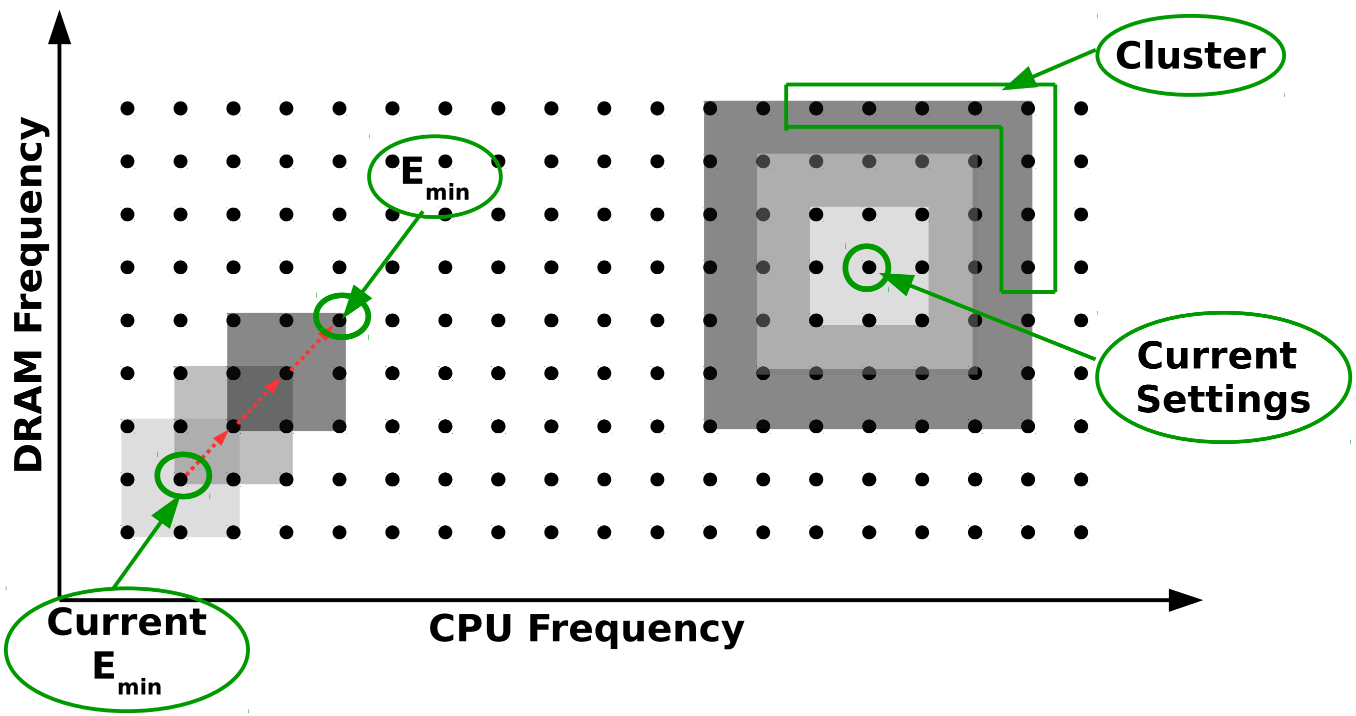
Search Algorithms

- Exhaustive Search
  - Searches entire space of core and DRAM frequency settings
- Relative Search
  - Reduces tuning cost by starting from previous settings



Search Algorithms

- Exhaustive Search
  - Searches entire space of core and DRAM frequency settings
- Relative Search
  - Reduces tuning cost by starting from previous settings



Select Algorithms

- Best Performing Selection
  - Selects settings that deliver best performance --- High overhead as it requires frequent tuning in order to adapt to application phases

Select Algorithms

- Best Performing Selection
  - Selects settings that deliver best performance --- High overhead as it requires frequent tuning in order to adapt to application phases
- Adaptive Selection

```
Initialize best_settings with the first setting in the cluster
current_settings = settings selected during the previous tuning
for settings in cluster
    if settings.performance > best_settings.performance
         best_settings = settings
    if settings == current_settings
         current_settings_found = True
if current_settings_found is True
    performance_loss = difference(best_settings,current_settings)
    if performance_loss is within threshold
         final_settings = current_settings
         skip_tuning_intervals++
        relative_search_steps++
    else
         final_settings = best_settings
         skip_tuning_intervals = 0
 else
      final_settings = best_settings
      skip_tuning_intervals = 0
```

Select Algorithms

- Best Performing Selection
  - Selects settings that deliver best performance --- High overhead as it requires frequent tuning in order to adapt to application phases
- Adaptive Selection

```
Initialize best_settings with the first setting in the cluster
current_settings = settings selected during the previous tuning
for settings in cluster
    if settings.performance > best_settings.performance
        best_settings = settings
    if settings == current_settings
        current_settings_found = True
if current_settings_found is True
    performance_loss = difference(best_settings,current_settings)
    if performance_loss is within threshold
        final_settings = current_settings
        skip_tuning_intervals++
        relative_search_steps++
    else
        final_settings = best_settings
        skip_tuning_intervals = 0
 else
    final_settings = best_settings
    skip_tuning_intervals = 0
```

1. **Detects application phases and skips tuning during long stable phases**

Select Algorithms

- Best Performing Selection
  - Selects settings that deliver best performance --- High overhead as it requires frequent tuning in order to adapt to application phases
- Adaptive Selection

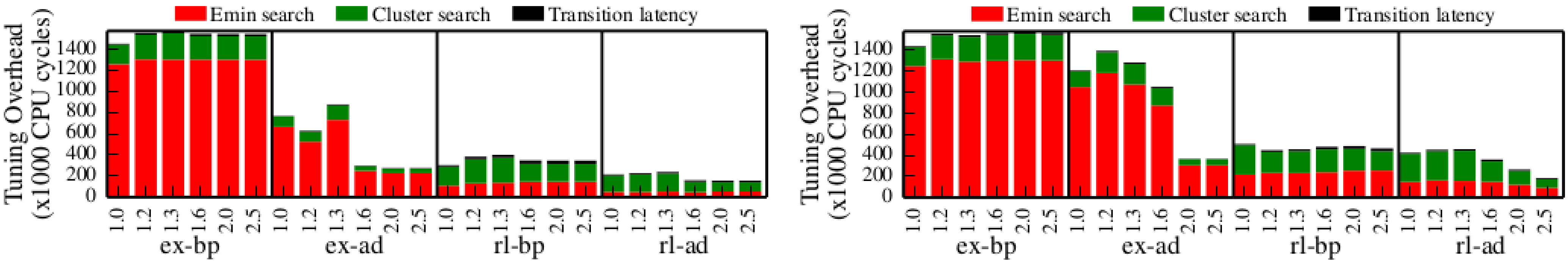
```
Initialize best_settings with the first setting in the cluster
current_settings = settings selected during the previous tuning
for settings in cluster
    if settings.performance > best_settings.performance
         best_settings = settings
    if settings == current_settings
         current_settings_found = True
if current_settings_found is True
    performance_loss = difference(best_settings,current_settings)
    if performance_loss is within threshold
         final_settings = current_settings
         skip_tuning_intervals++
        relative_search_steps++
    else
         final_settings = best_settings
         skip_tuning_intervals = 0
 else
      final_settings = best_settings
      skip_tuning_intervals = 0
```

1. **Detects application phases and skips tuning during long stable phases**
2. **Detects possible "stuck in local minima and gives feedback to relative search to exit the local minima region**

Outline

- Energy constraints
- System Design
- Performance and Energy Models
- Algorithms
- **Results**
- Conclusions

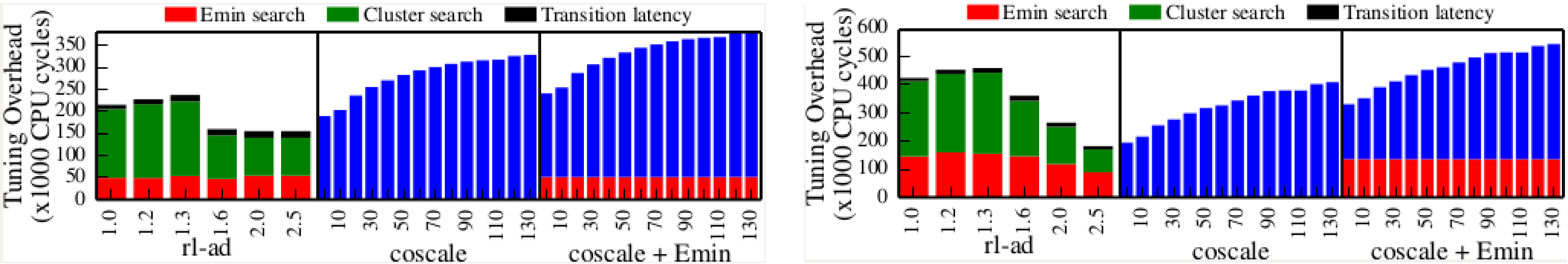
Results



a) Bzip2 b) Gobmk

- 12 integer and 9 floating point SPEC CPU2006 benchmarks --- using Gem5 simulator
- All combinations of search and select algorithms are simulated
- *rl-ad* significantly improves tuning overhead compared to *ex-bp* with less than 3% performance loss --- by selectively tuning and reducing the search space
- Next we compare our algorithms with CoScale's algorithms designed to operate under performance constraints

Results



a) Bzip2

b) Gobmk

- Tuning overhead of *rl-ad* is 32% lower compared to CoScale for *Bzip2,* and 10% higher for *Gobmk*
- Gobmk doesn't provide enough opportunities to skip tuning due to rapidly changing phases
- Comparing tuning overhead to CoScale + E_{min} is fair
- *rl-ad* has 24% lower tuning overhead compared to CoScale
- Algorithms save up to 5% energy with average performance loss of less than 3%

Outline

- Energy constraints
- System Design
- Performance and Energy Models
- Algorithms
- Results
- **Conclusions**

Conclusions

- core DVFS combined with DRAM frequency scaling provides great opportunities to make energy-performance trade-offs

Conclusions

- core DVFS combined with DRAM frequency scaling provides great opportunities to make energy-performance trade-offs
- Coordinated management of core and DRAM frequency scaling is necessary in order to conserve energy effectively

Conclusions

- core DVFS combined with DRAM frequency scaling provides great opportunities to make energy-performance trade-offs
- Coordinated management of core and DRAM frequency scaling is necessary in order to conserve energy effectively
- Energy management approaches that work under performance constraints can not be directly applied to systems operating under energy constraints --- computing performance bound for desired energy savings is a daunting task and requires oracle knowledge of applications and devices

Conclusions

- core DVFS combined with DRAM frequency scaling provides great opportunities to make energy-performance trade-offs
- Coordinated management of core and DRAM frequency scaling is necessary in order to conserve energy effectively
- Energy management approaches that work under performance constraints can not be directly applied to systems operating under energy constraints --- computing performance bound for desired energy savings is a daunting task and requires oracle knowledge of applications and devices
- Introduced *Inefficiency*--- a relative energy constraint that specifies amount of additional energy that can be used to improve performance

Conclusions

- core DVFS combined with DRAM frequency scaling provides great opportunities to make energy-performance trade-offs
- Coordinated management of core and DRAM frequency scaling is necessary in order to conserve energy effectively
- Energy management approaches that work under performance constraints can not be directly applied to systems operating under energy constraints --- computing performance bound for desired energy savings is a daunting task and requires oracle knowledge of applications and devices
- Introduced *Inefficiency*--- a relative energy constraint that specifies amount of additional energy that can be used to improve performance
- Introduced new selective and adaptive search algorithms to reduce tuning overhead of E_{min} and cluster search

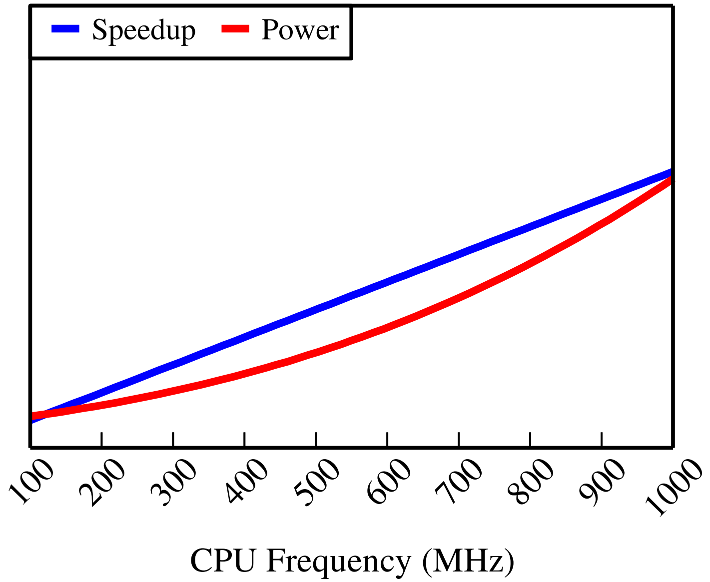
Conclusions

- core DVFS combined with DRAM frequency scaling provides great opportunities to make energy-performance trade-offs
- Coordinated management of core and DRAM frequency scaling is necessary in order to conserve energy effectively
- Energy management approaches that work under performance constraints can not be directly applied to systems operating under energy constraints --- computing performance bound for desired energy savings is a daunting task and requires oracle knowledge of applications and devices
- Introduced *Inefficiency*--- a relative energy constraint that specifies amount of additional energy that can be used to improve performance
- Introduced new selective and adaptive search algorithms to reduce tuning overhead of E_{min} and cluster search
- Presented a holistic approach that works under inefficiency constraint and using the cross-component performance and energy models that we developed, selects optimal frequency settings that deliver best performance staying under given inefficiency budget

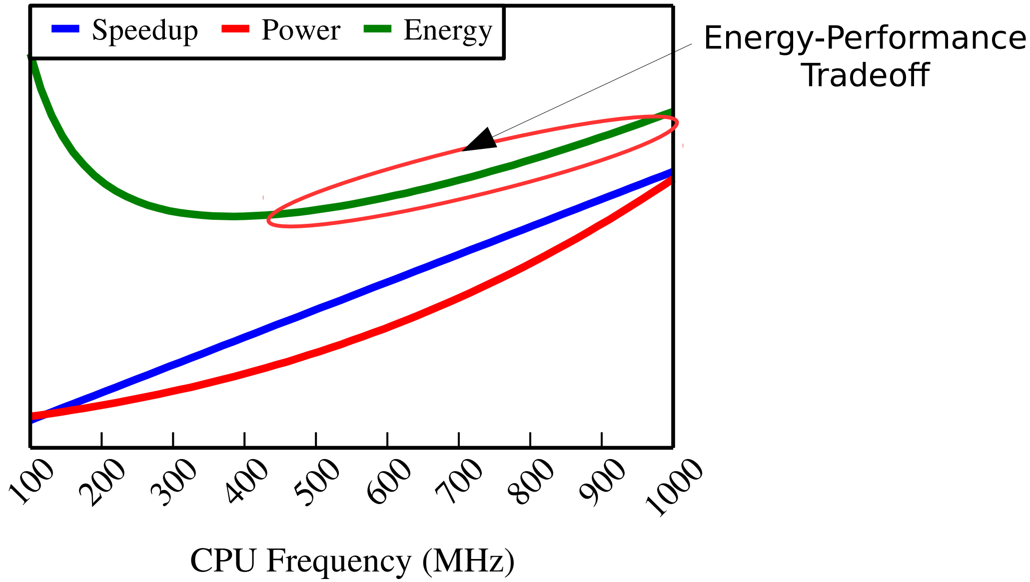
Blue.

Questions ?

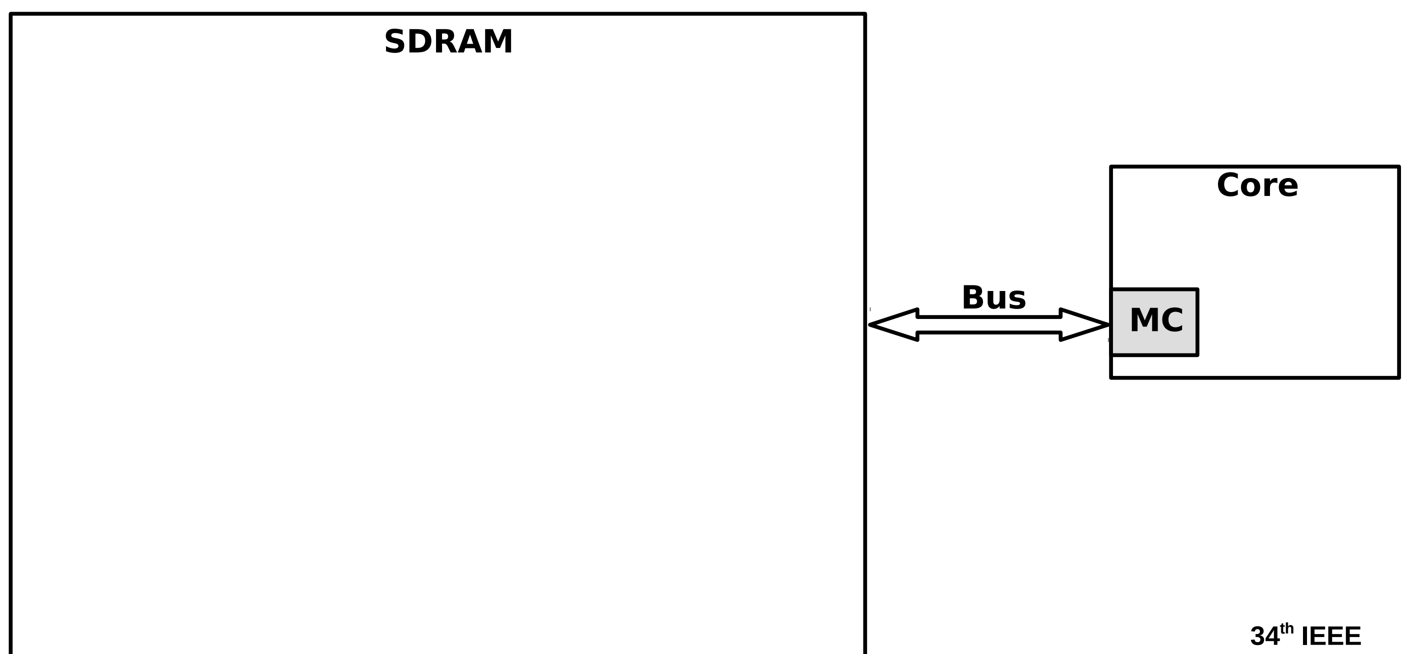
Dynamic Voltage and Frequency Scaling



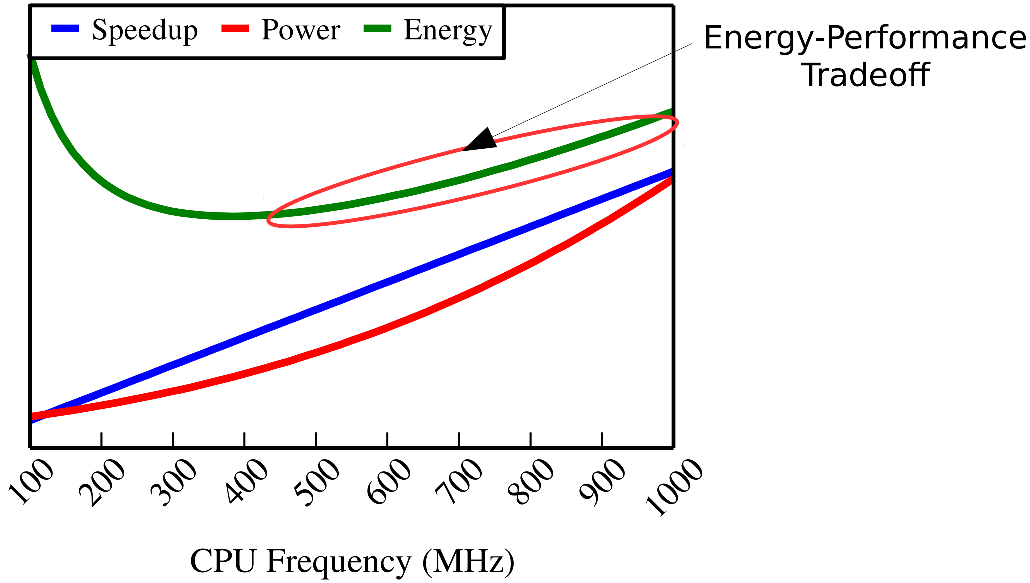
Dynamic Voltage and Frequency Scaling



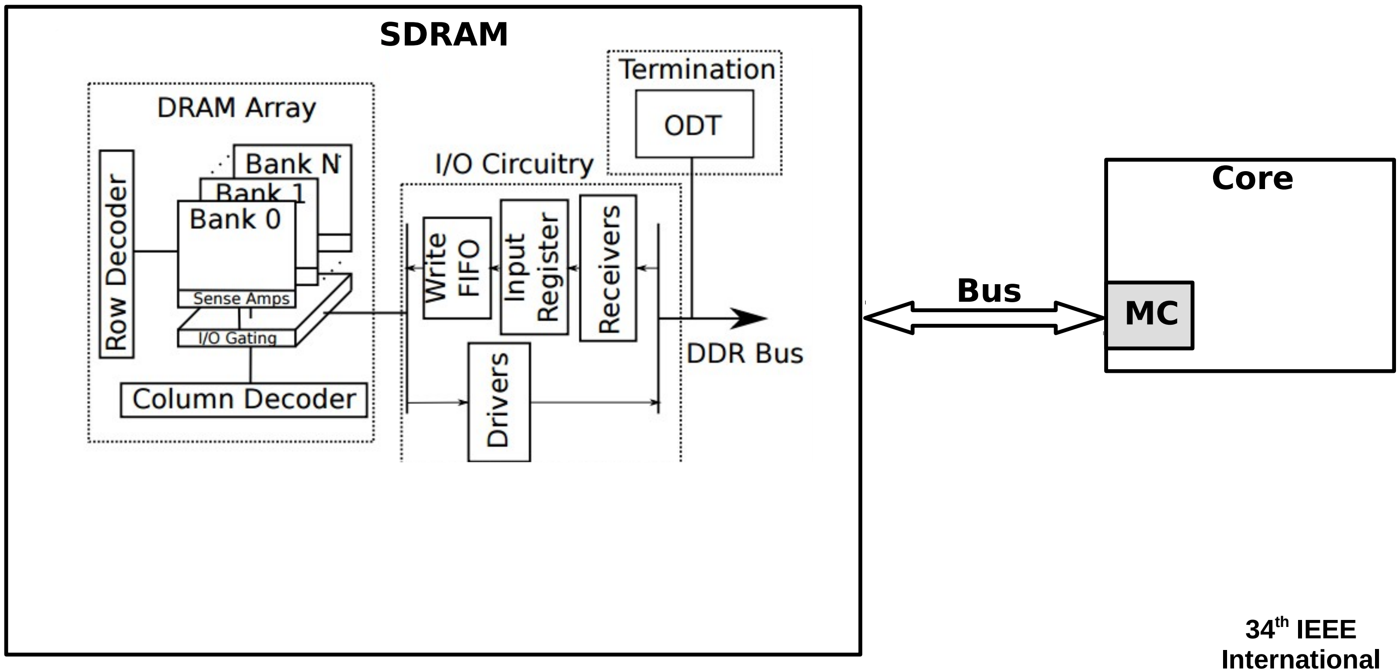
SDRAM Frequency Scaling



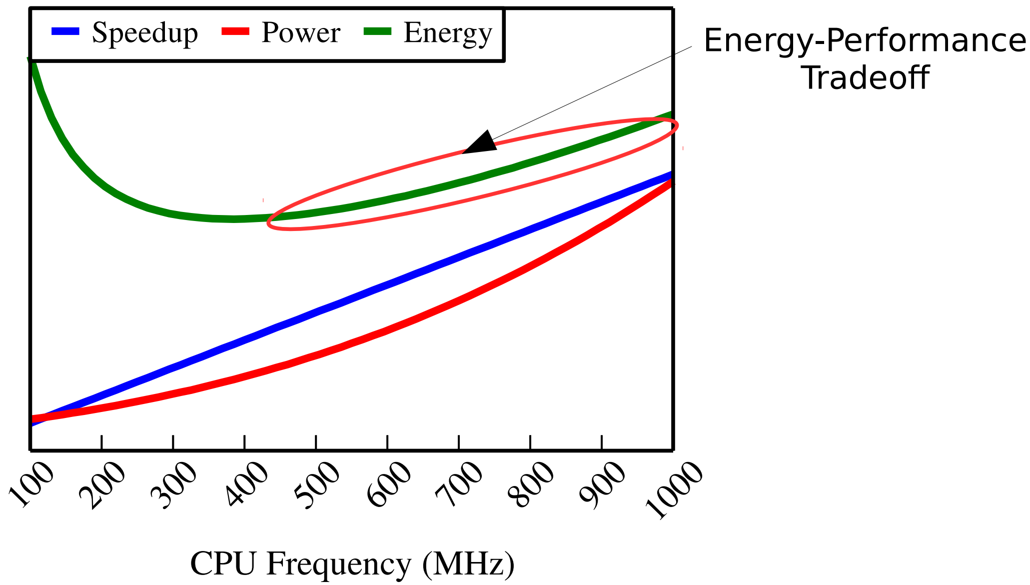
Dynamic Voltage and Frequency Scaling



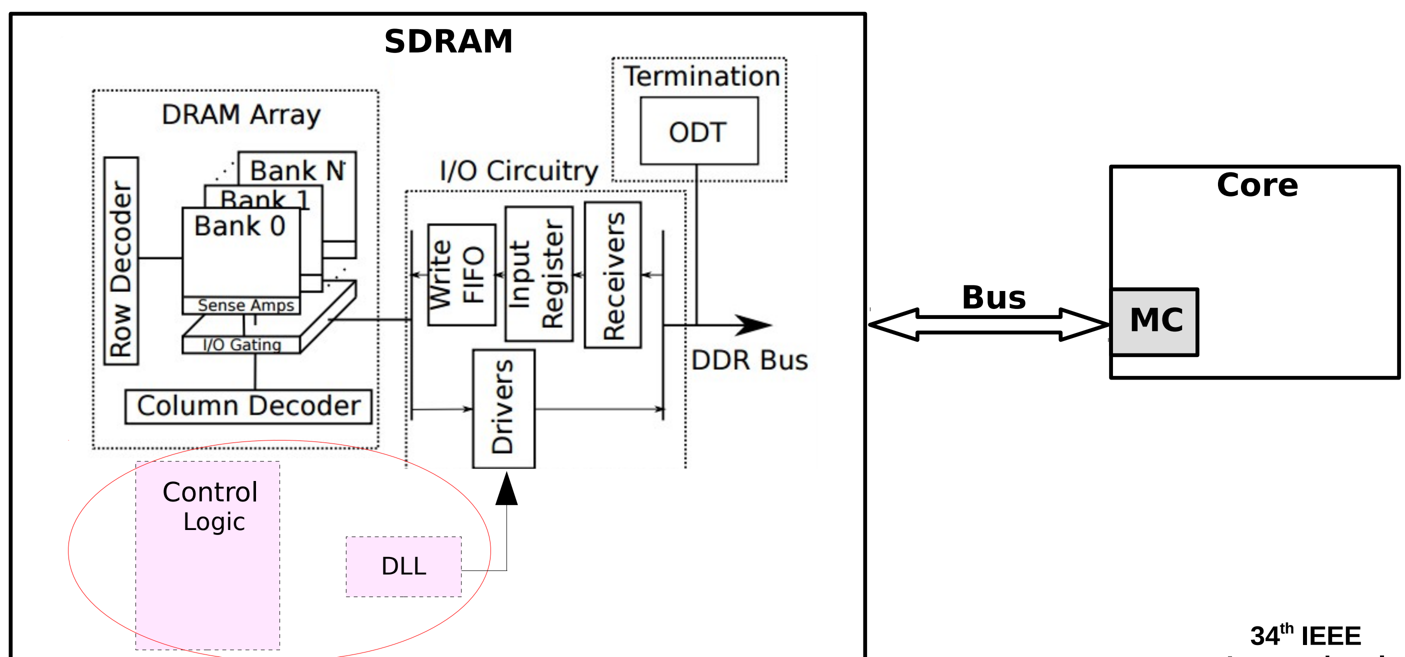
SDRAM Frequency Scaling



Dynamic Voltage and Frequency Scaling



SDRAM Frequency Scaling



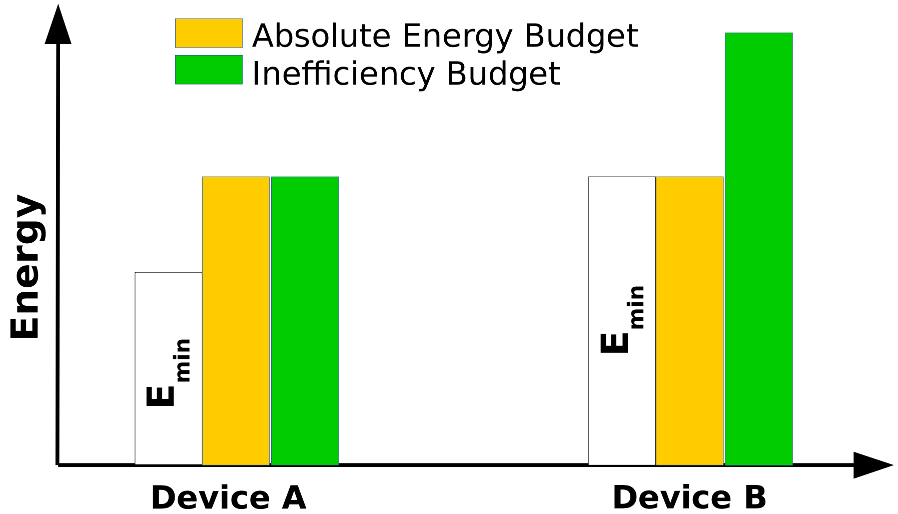
Inefficiency as a System Resource

- Agnostic to Devices

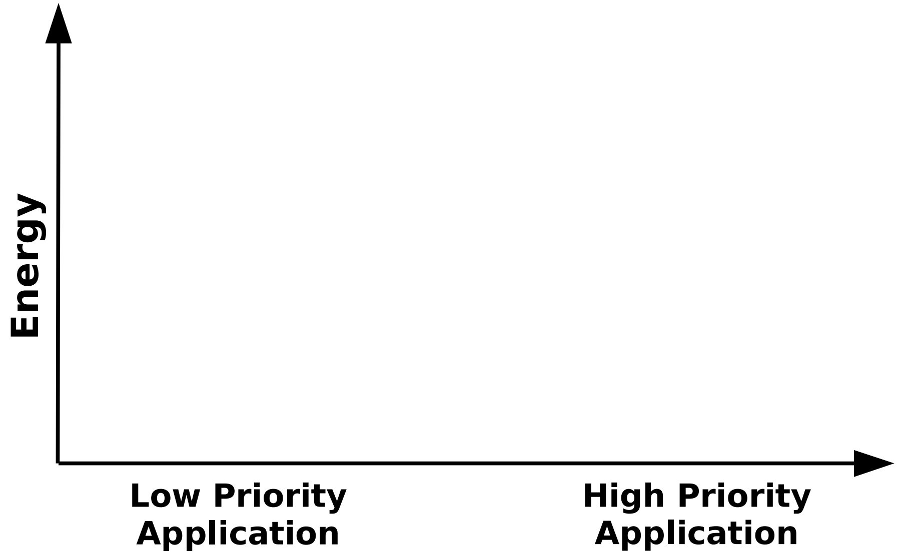
$$Inefficiency = \frac{E_{total}}{E_{min}}$$

$$\Downarrow$$

$$E_{total} = Inefficiency \times E_{min}$$



- Relative to inherent energy needs of the application
  - Inefficiency tied to priority of the applications

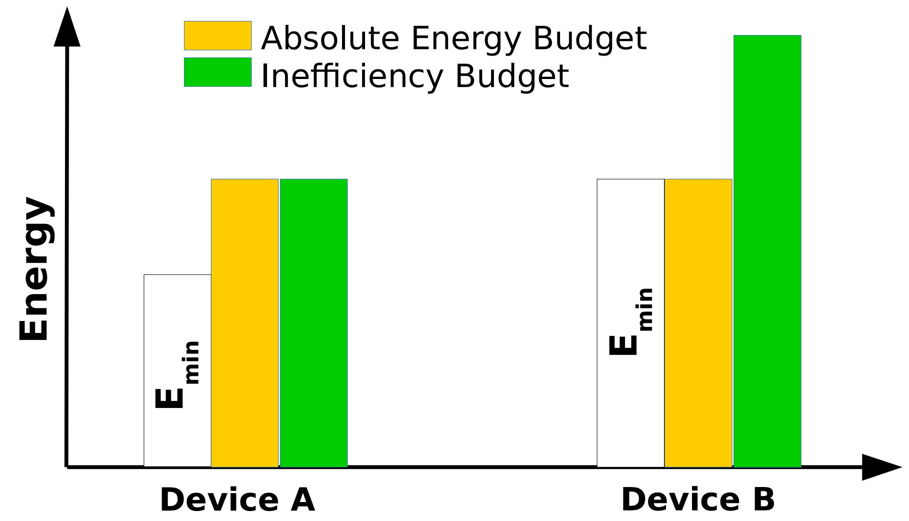


Inefficiency as a System Resource

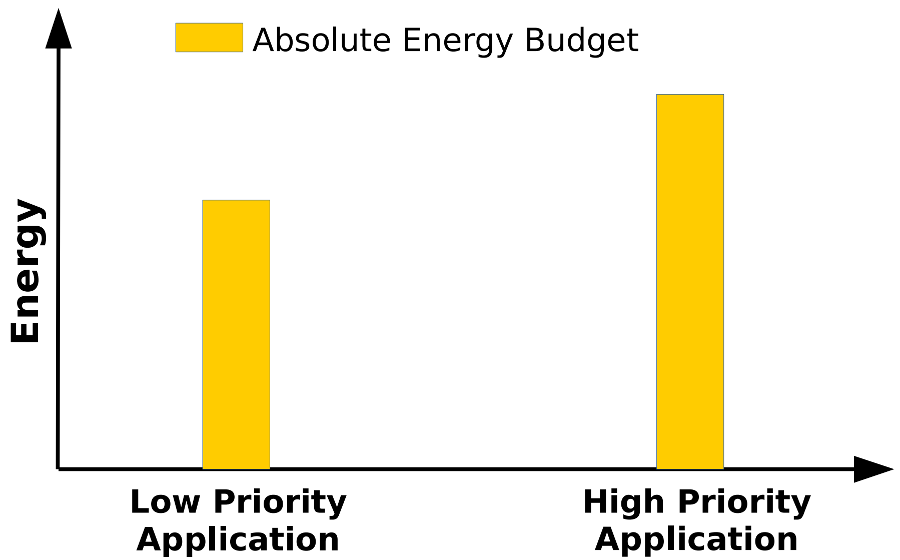
- Agnostic to Devices

$$Inefficiency = \frac{E_{total}}{E_{min}}$$

$$E_{total} = Inefficiency \times E_{min}$$



- Relative to inherent energy needs of the application
  - Inefficiency tied to priority of the applications

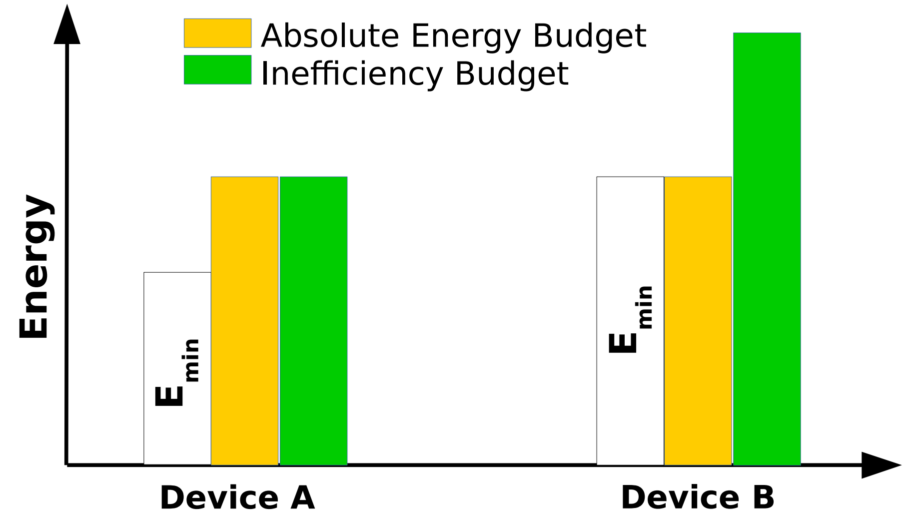


Inefficiency as a System Resource

- Agnostic to Devices

$$Inefficiency = \frac{E_{total}}{E_{min}}$$

$$E_{total} = Inefficiency \times E_{min}$$



- Relative to inherent energy needs of the application
  - Inefficiency tied to priority of the applications