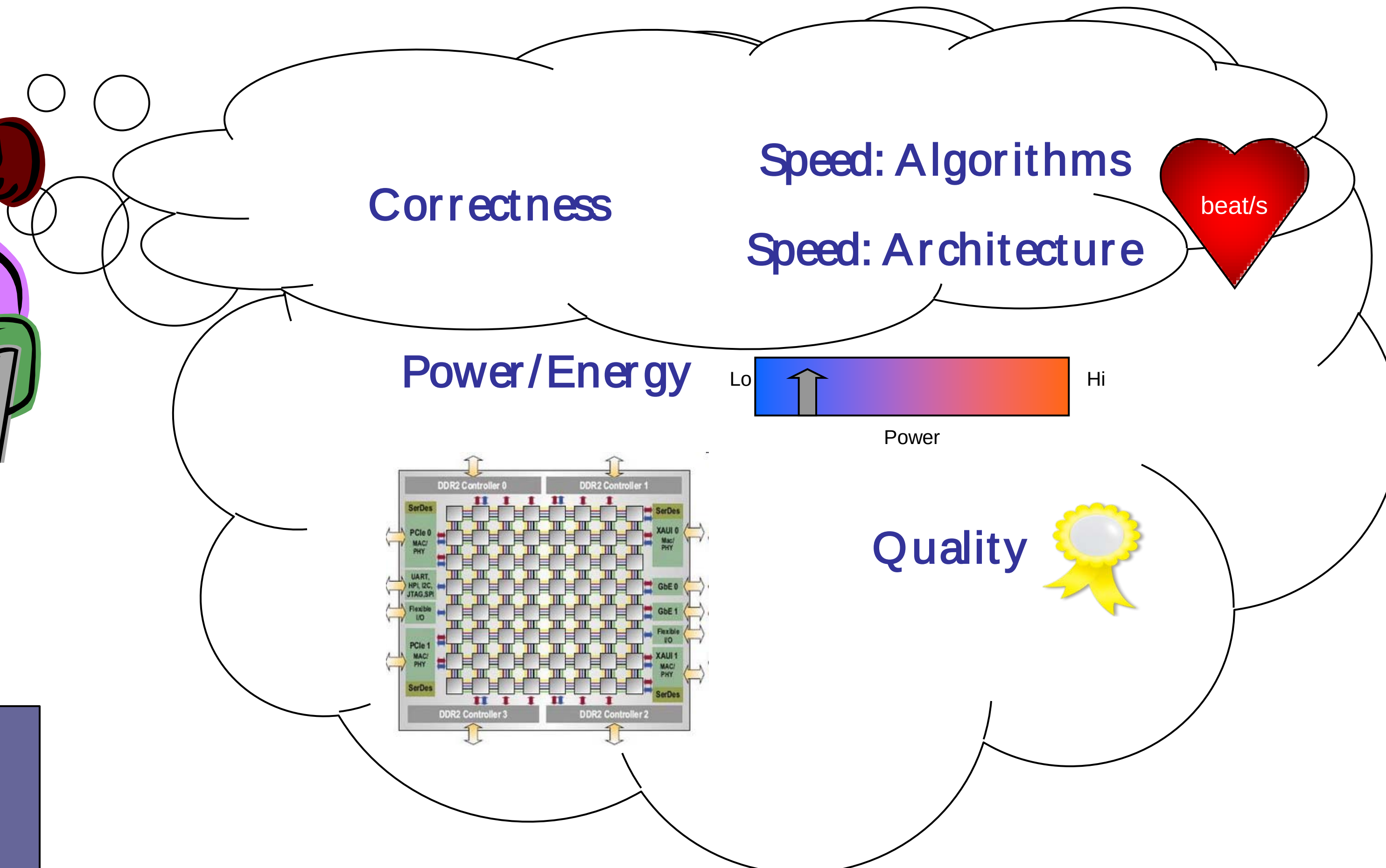
SEEC: A Framework for SElf-awarE Computing

Henry Hoffmann, Martina Maggio

Outline

- Introduction/Motivating Example
- The SEEC Framework
- Experimental Validation
- Conclusions

Multicore Computing Systems Increase Burden on Application Developers

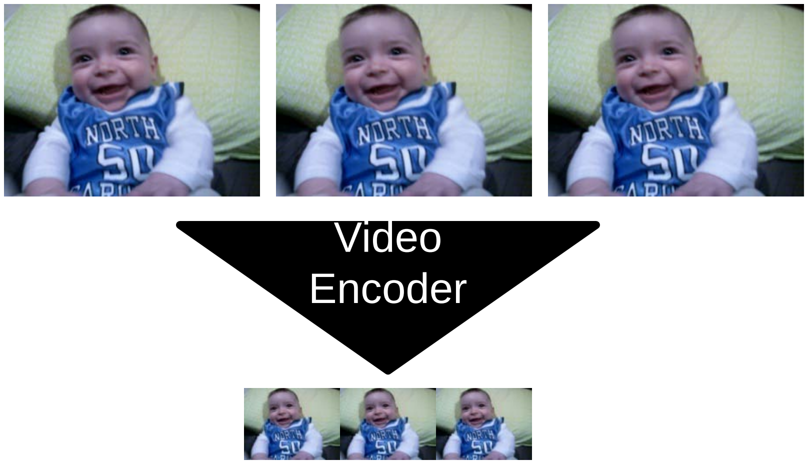


Application

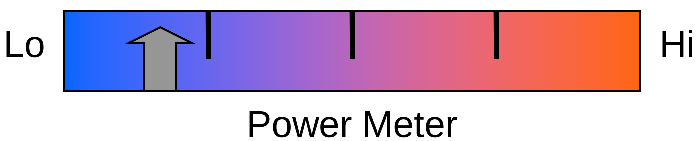
Today, application programmers have to address many, sometimes competing, concerns

Example: Developing a Multicore Video Encoder

Allocate resources for best case

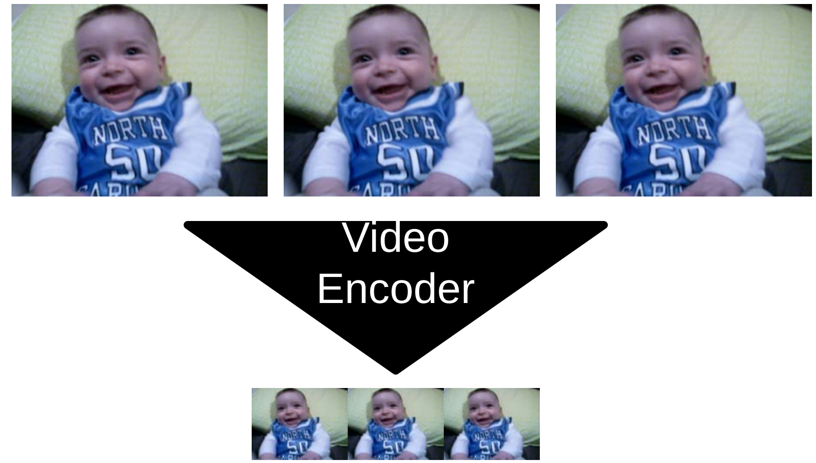


Encoder must drop frames to keep up

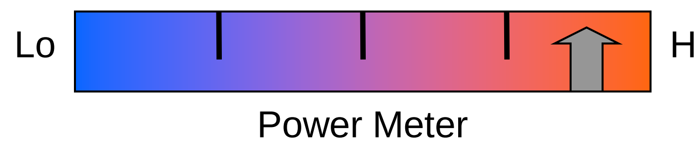


The power is low

Allocate resources for worst case



Encoder Exceeds Goals



The power is high, resources wasted

Application programmers need to balance competing constraints in fluctuating environments

Self-aware Computing Can Address Challenges of Multi-objective Optimization

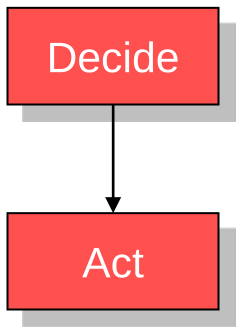
Self-aware (or self-*, adaptive, autonomic, etc.) systems have the flexibility to change behavior online to balance multiple needs in dynamic environments

Self-aware (self-*, adaptive, etc.) computing has become a discipline unto itself:

- Laddaga [DARPA BAA 1997, IEEE Intelligent Systems 1999]
- Kephart and Chess [IEEE Computer 2003]
- Babaoglu et al. [LNCS 2005]

The Self-Aware Computing Idea

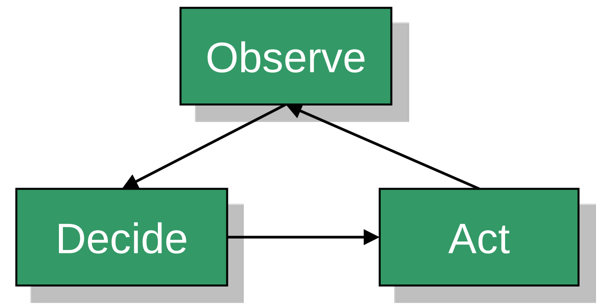
Traditional Systems



- Run in open loop
- Assumptions made at design time
- Based on guesses about future

- Programmer optimizes for system
- No flexibility to adapt to changes

Self-Aware Systems



- Run in closed loop
- Understand user goals
- Monitor the environment

- System optimizes for application
- Flexibly adapt behavior

Prior Work in Self-Aware Systems

- Self-aware/Adaptive/Autonomic systems have been used to solve problems in:
  - Hardware [Bitirgen et al, MICRO 2008, Albonesi et al. IEEE Computer 2003]
  - Software [Salehie & Tahvildari ACM TAAS 2009]
  - Real-time Systems [Block et al. ECRTS 2008]
  - Mobile Computing [Masters MobileHCI 2008]
  - Dynamic Compilation [Sorber et al, SenSys 2007, Baek & Chilimbi PLDI 2010]
  - Numerical Libraries [ATLAS, SPIRAL, FFTW]
  - Many others…

We build on previous work to create a programming model for self-aware systems

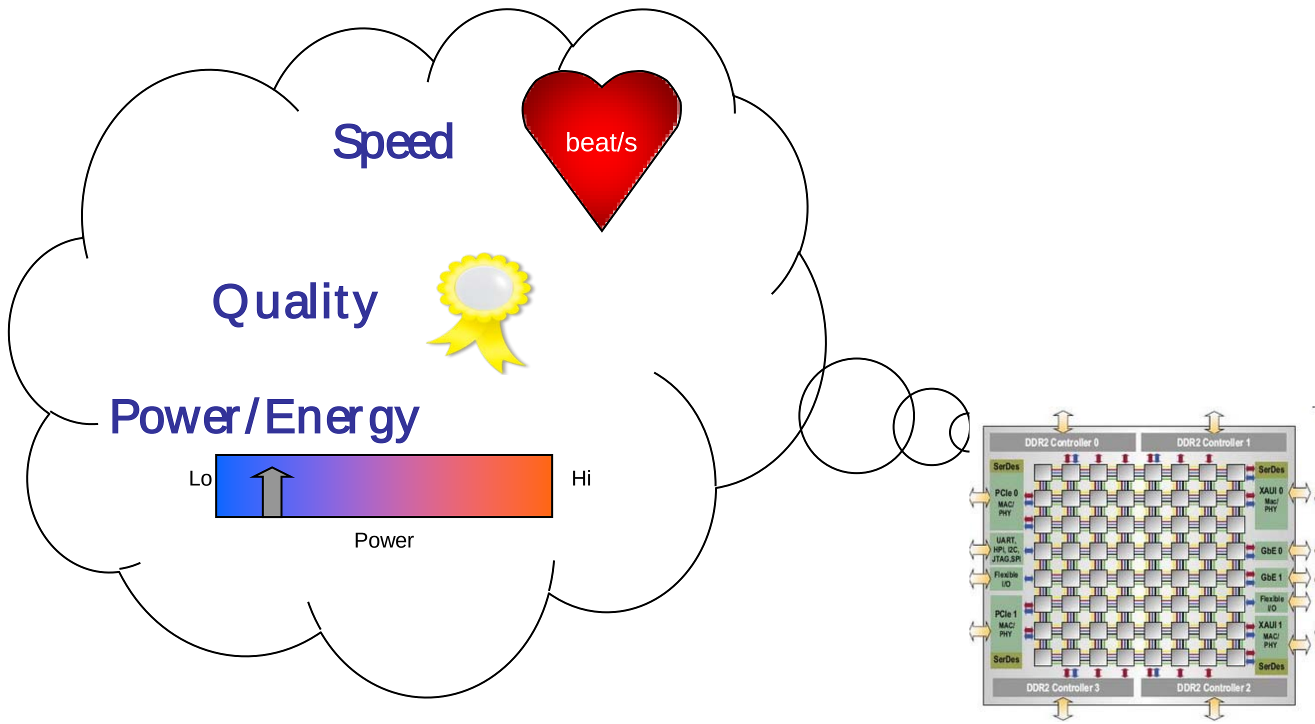
Outline

- Introduction/Motivating Example
- The SEEC Framework
- Experimental Validation
- Conclusions

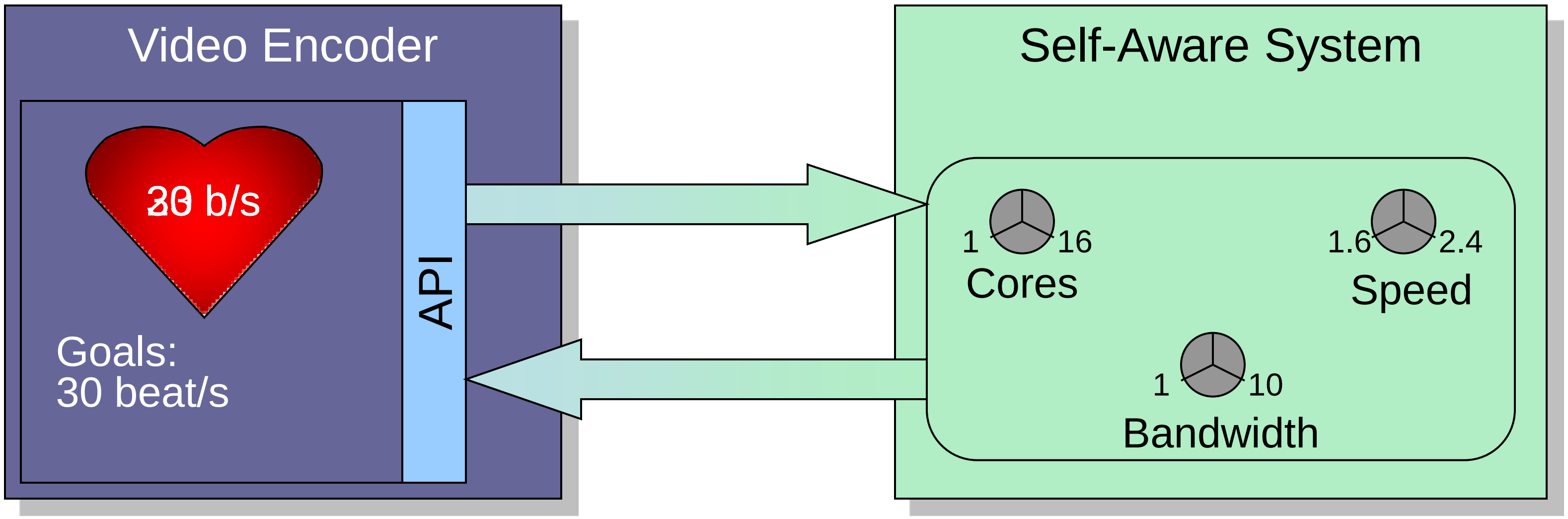
SElf-awarE Computing (SEEC) Framework

- Goal:
  Reduce programmer burden with self-aware programming model
- Key Features:
  1. Applications *explicitly* state goals, system meets goals optimally
  2. One unified decision engine adapts algorithms, software and hardware

Application

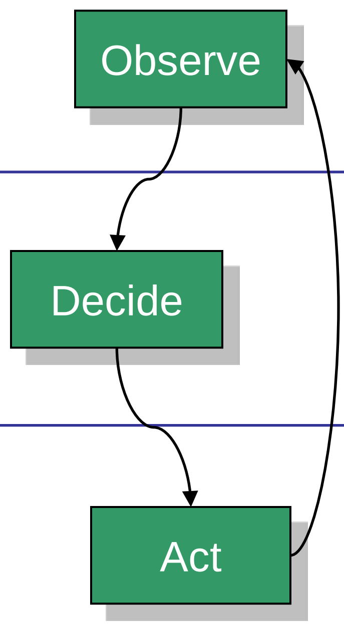


Example Self-Aware System Built from SEEC



- At key intervals, applications issue a heartbeat (e.g. once per frame)
- Apps also register desired performance (e.g. 30 beats (frames) per second)
- The performance (heart rate) and goals can be read by system software
- If performance is low the system adapts to increase performance
- If performance exceeds goals, the system frees resources

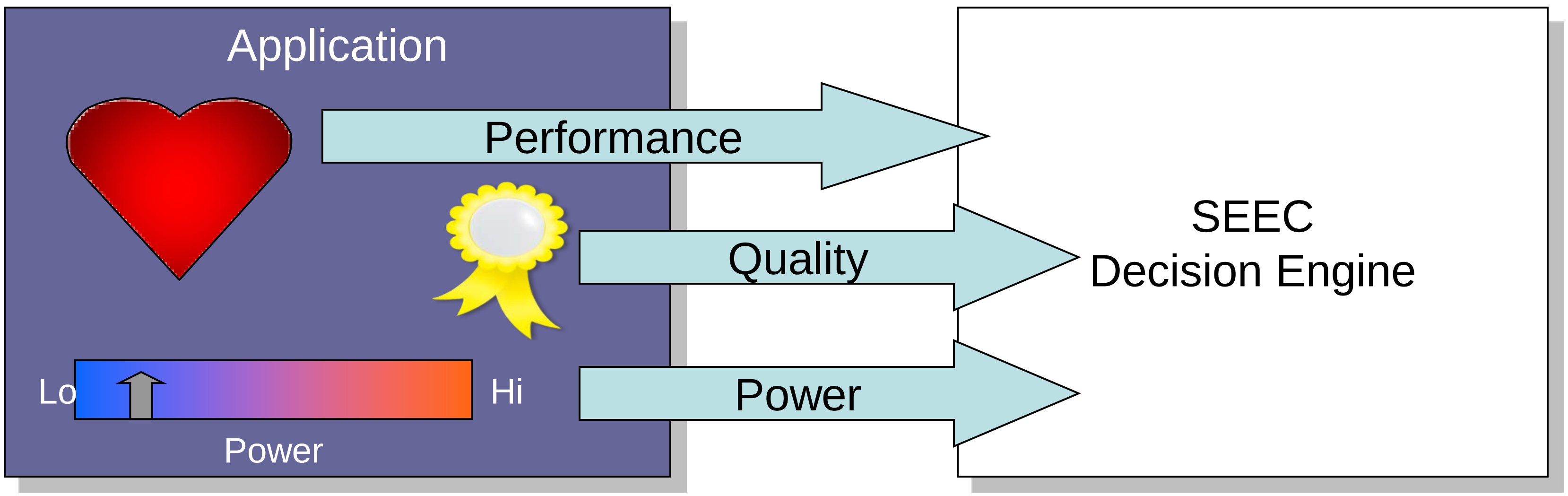
Roles in the SEEC Framework



	Application Developer	Systems Developer	SEEC System Infrastructure
Observe	Express application goals and progress (e.g. frames/ second)		Read goals and performance
Decide			Determine how to adapt (e.g. How much to speed up the application)
Act		Provide a set of actions and a callback function (e.g. allocation of cores to process)	Initiate actions based on results of decision phase

Registering Application Goals

Observe

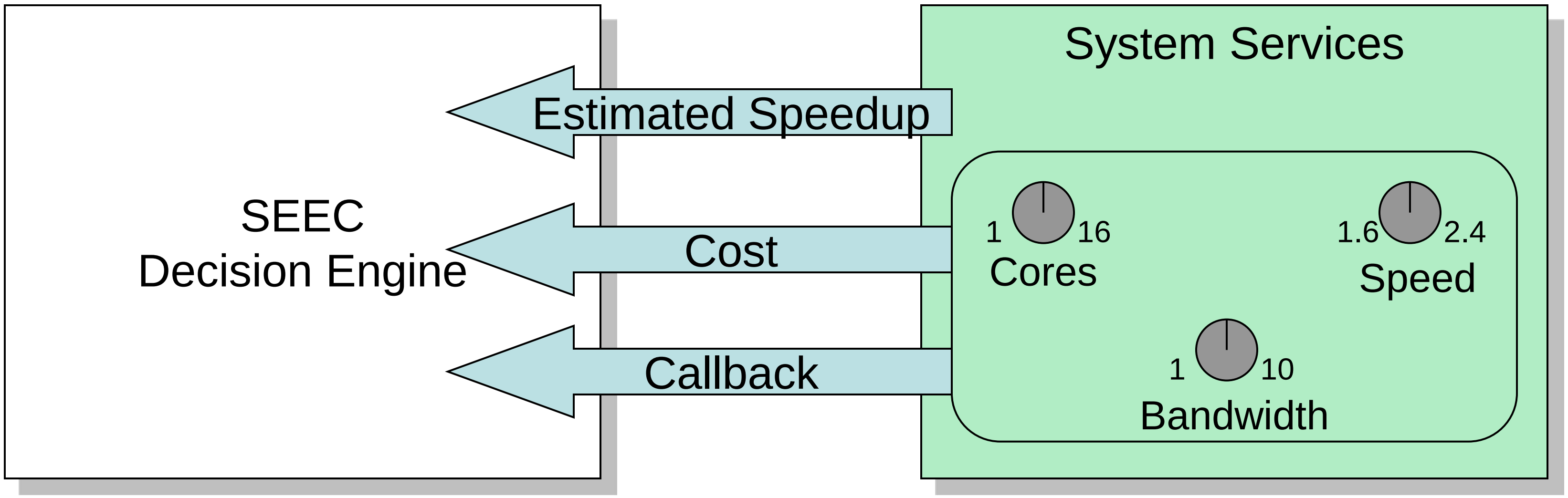


- Performance
  - Goals: target heart rate and/or latency between tagged heartbeats
  - Progress: issue heartbeats at important intervals
- Quality
  - Goals: distortion (distance from “best” value)
  - Progress: distortion over last heartbeat
- Power
  - Goals: target heart rate / Watt and/or target energy between tagged heartbeats
  - Progress: Power/energy over last heartbeat interval

Research to date focuses on meeting performance while minimizing power/maximizing quality

Registering System Actions

Act

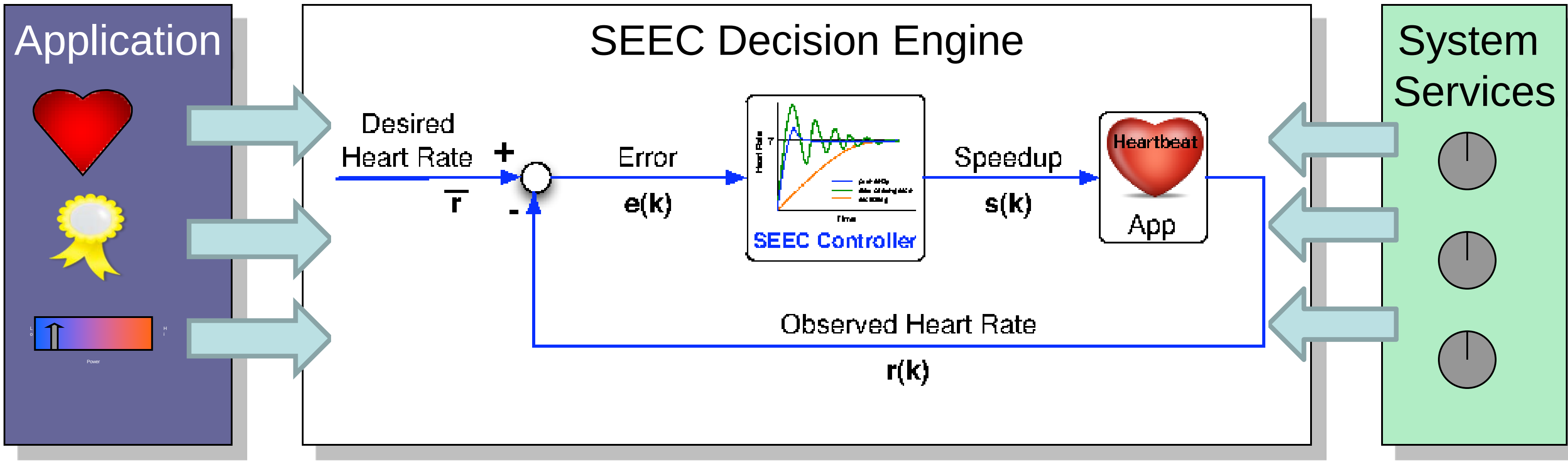


Each action has the following attributes:

- Estimated Speedup
  - Predicted benefit of taking an action
- Cost
  - Predicted downside of taking an action (increased power, lowered quality)
- Callback
  - A function that takes an id an implements the associated action

The SEEC Decision Engine

Decide



Decisions are made to select actions given observations:

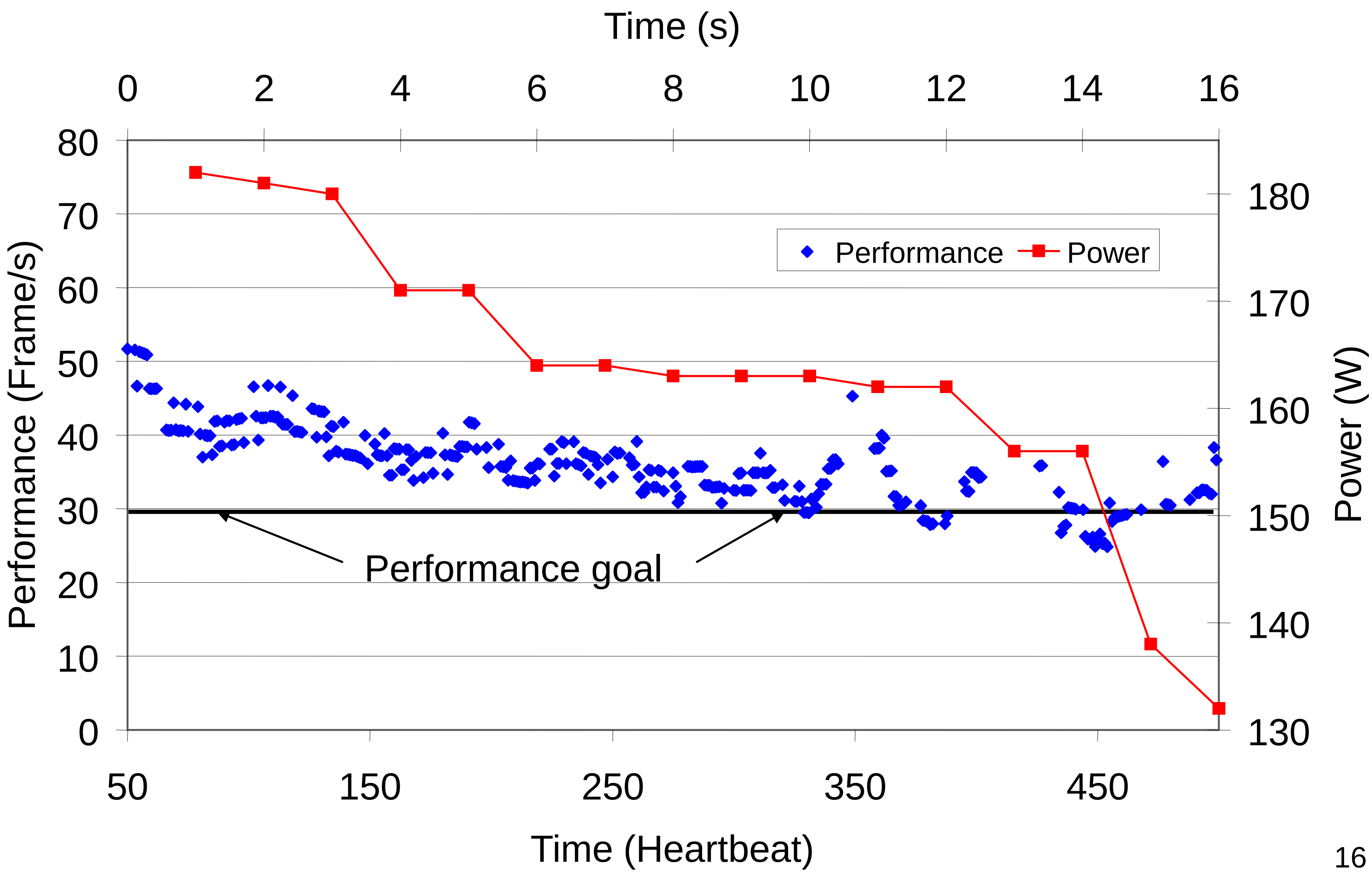
- Read application goals and heartbeats
- Determine speedup with *adaptive* 2^{nd} order control system
- Translate speedup into one or more actions

The control system provides predictable and analyzable behavior

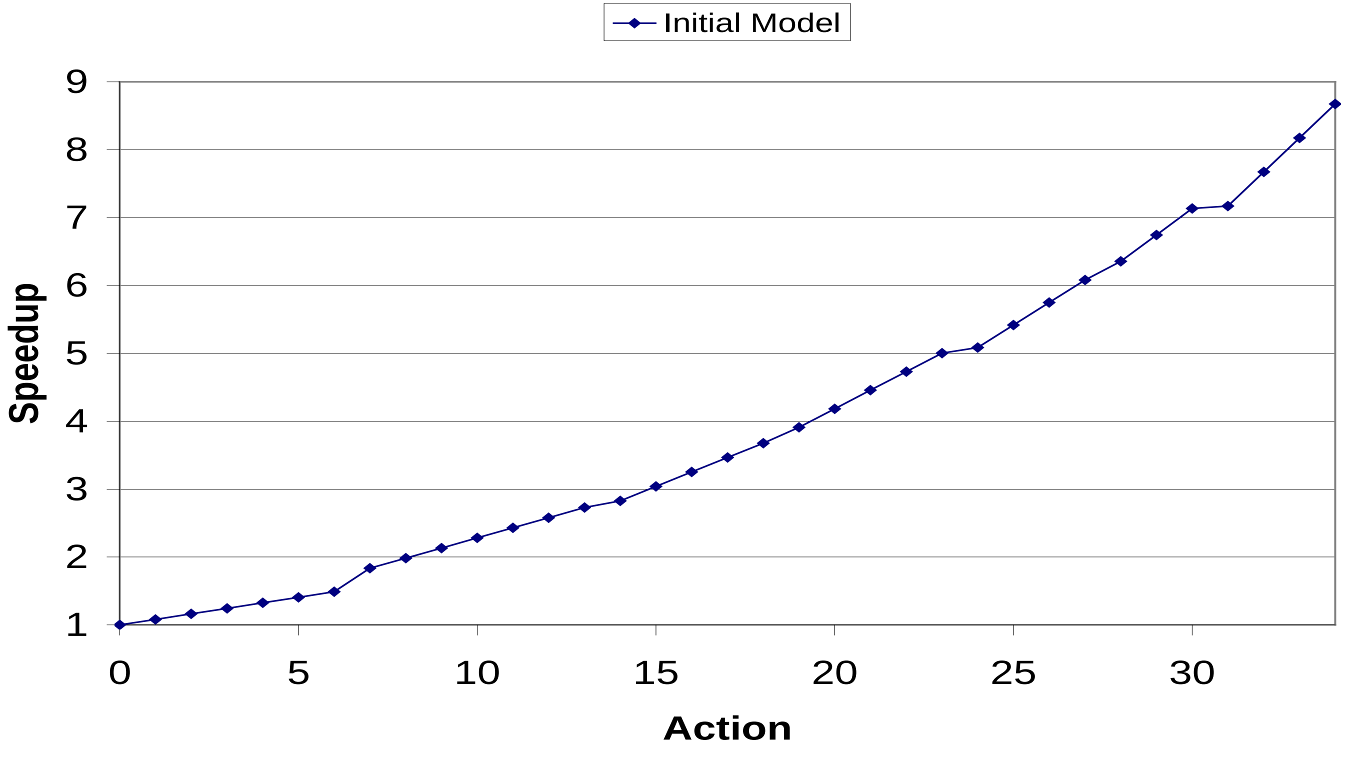
Optimizing Resource Allocation with SEEC

- SEEC can observe, decide and act
- How does this enable optimal resource allocation?
- Let’s implement the video encoder example from the introduction

Performance/Watt Adaptation in Video Encoding



System Models in SEEC

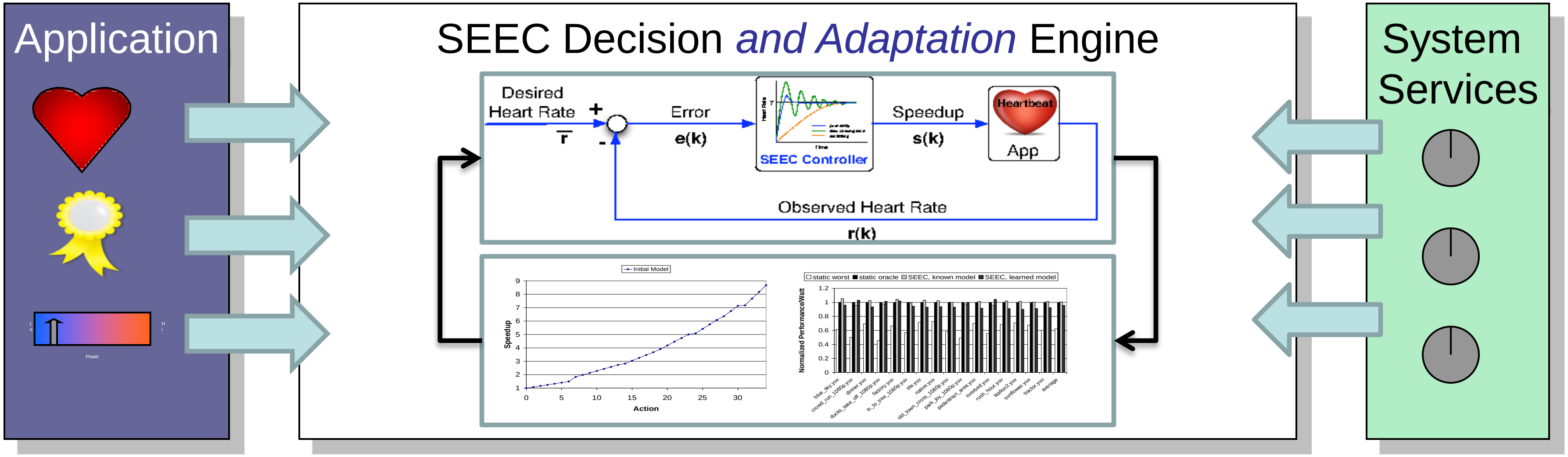


SEEC's control system takes actions based on models (of speedup and cost per action) associated with actions

What if the models are inaccurate?

Updating Models in SEEC

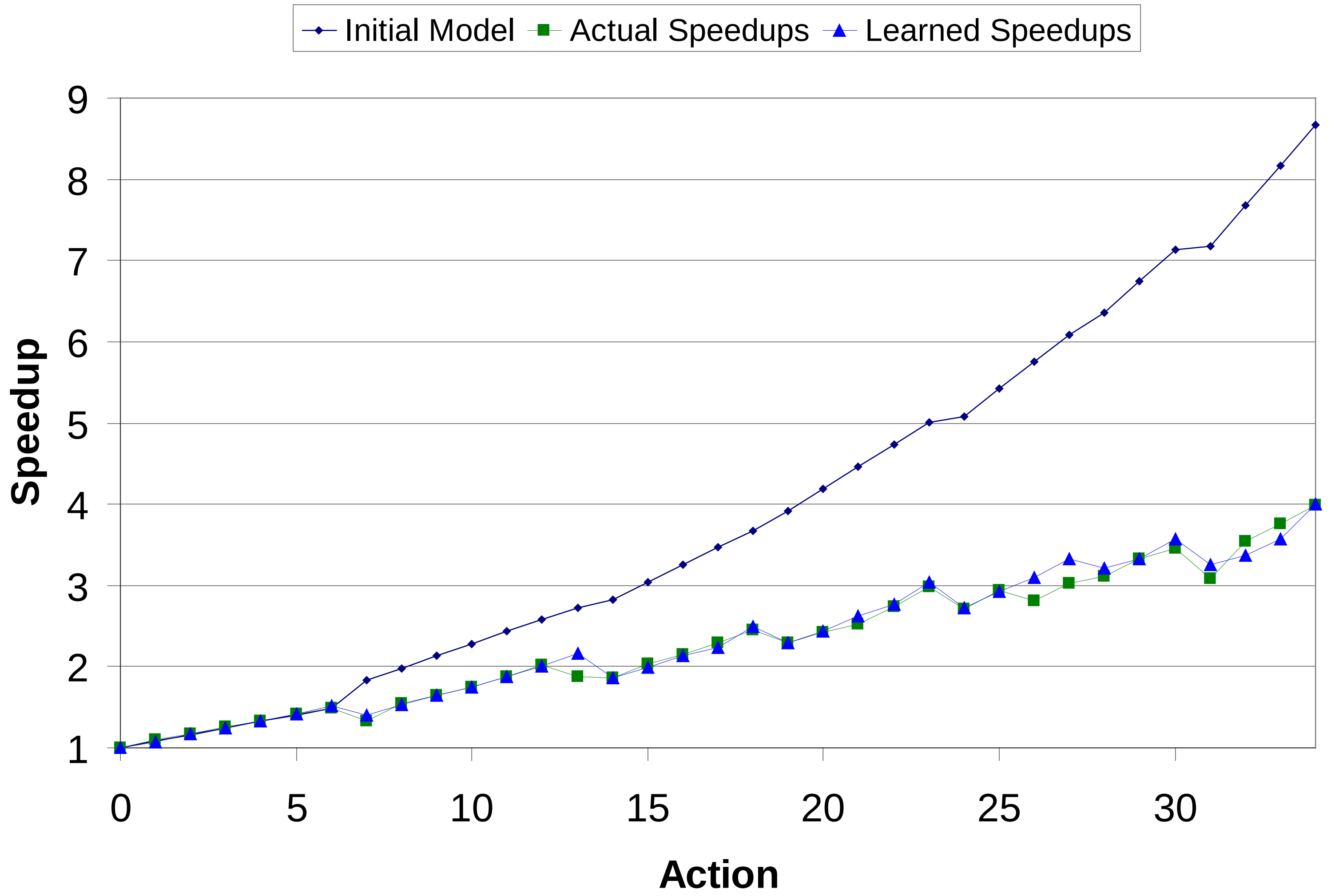
Decide



- After every action, SEEC updates system models
- Kalman filters used to estimate true speedups

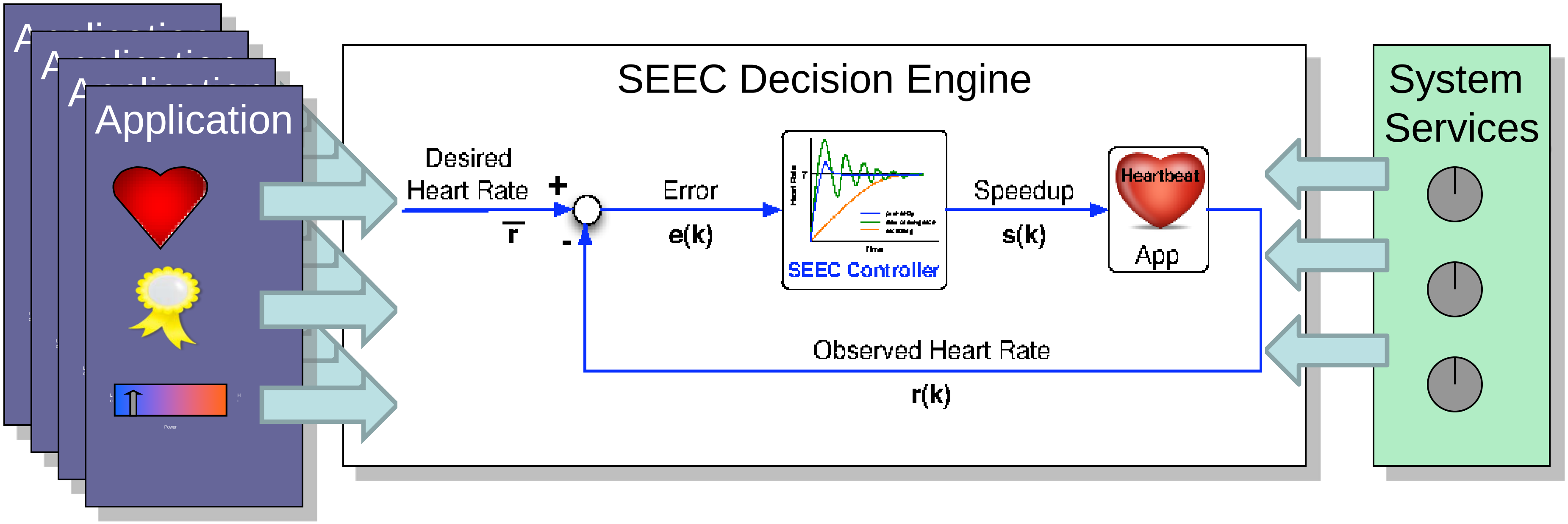
SEEC combines predictability of control systems with adaptability of learning systems

SEEC Online Learning of Speedup Model for Application with Local Minima



Handling Multiple Applications

Decide



- Control actions computed separately for each application
- For finite resources, several alternatives:
  - Priorities determine which apps meet resource needs
  - Weights determine proportional assignment

Outline

- Introduction/Motivating Example
- The SEEC Framework
- ➡ Experimental Validation
- Conclusions

Systems Built with SEEC

System	Actions	Tradeoff	Benchmarks
Dynamic Loop Perforation	Skip some loop iterations	Performance vs. Quality	7/13 PARSECs
Dynamic Knobs	Make static parameters dynamic	Performance vs. Quality	bodytrack, swaptions, x264, SWISH++
Core Scheduler	Assign N cores to application	Compute vs. Power	11/13 PARSECs
Clock Scaler	Change processor speed	Compute vs. Power	11/13 PARSECs
Bandwidth Allocator	Assign memory controllers to application	Memory vs. Power	STREAM (doesn't make a difference for PARSEC)
Power Manager	Combination of the three above	Performance vs. Power	PARSEC, STREAM, simple test apps (mergesort, binary search)
Learned Models	Power Manager with speedup and cost learned online	Performance vs. Power	PARSECs
Multi-App Control	Power Manager with multiple applications	Performance vs. Power for multiple applications	Combinations of PARSECs

Systems Built with SEEC

System	Actions	Tradeoff	Benchmarks
Dynamic Loop Perforation	Skip some loop iterations	Performance vs. Quality	7/13 PARSECs
Dynamic Knobs	**Make static parameters dynamic**	**Performance vs. Quality**	**bodytrack, swaptions, x264, SWISH++**
Core Scheduler	Assign N cores to application	Compute vs. Power	11/13 PARSECs
Clock Scaler	Change processor speed	Compute vs. Power	11/13 PARSECs
Bandwidth Allocator	Assign memory controllers to application	Memory vs. Power	STREAM (doesn't make a difference for PARSEC)
Power Manager	**Combination of the three above**	**Performance vs. Power**	**PARSEC, STREAM, extra test apps (mergesort, binary search)**
Learned Models	**Power Manager with speedup and cost learned online**	**Performance vs. Power**	**PARSECs**
Multi-App Control	**Power Manager with multiple applications**	**Performance vs. Power for multiple applications**	**Combinations of PARSECs**

Dynamic Knobs: Creating Adaptive Applications

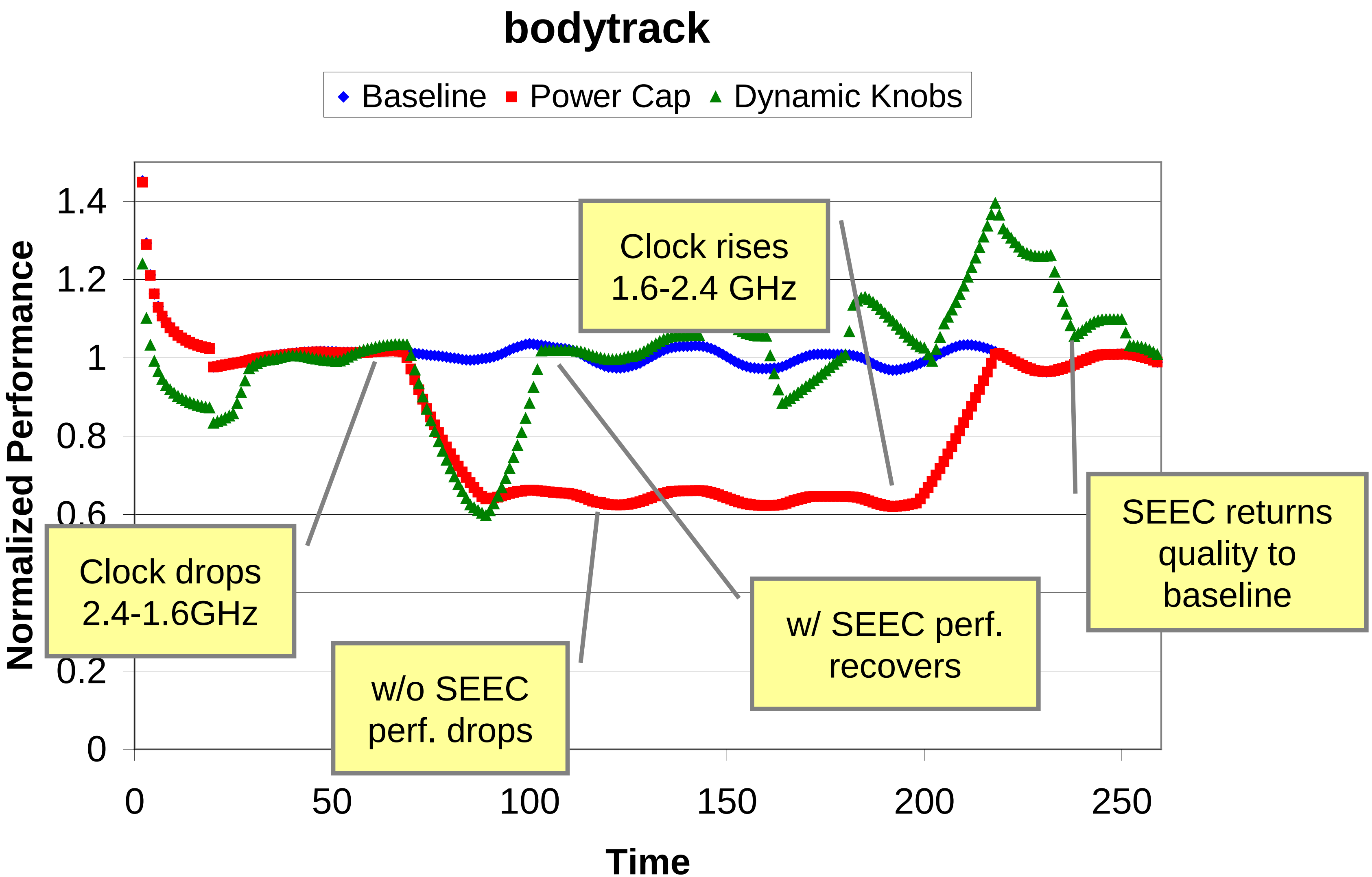
Turn static command line parameters into dynamic structure

Application Goals	Maintain performance and minimize quality loss
System Actions	Adjust memory locations to change application settings
Experiment	Benchmarks: bodytrack, swaptions, SWISH++, x264 Maintain performance when clock speed changes

Detail in Hoffmann et al. “Dynamic Knobs for Power Aware Computing” ASPLOS 2011

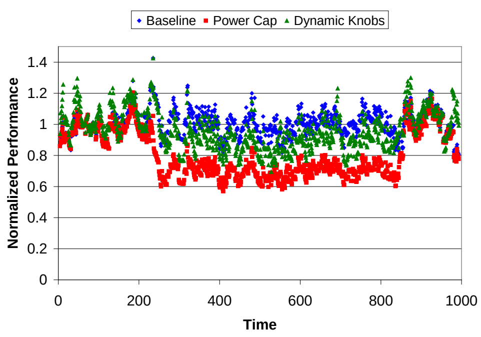
Results

Enabling Dynamic Applications

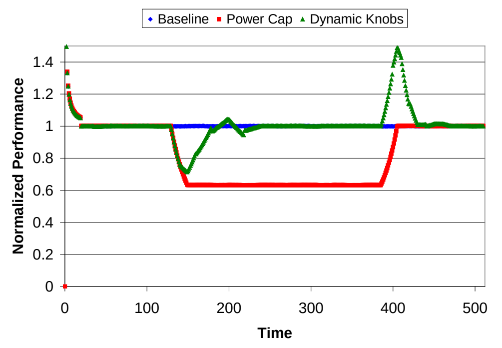


Results
Enabling Dynamic Applications

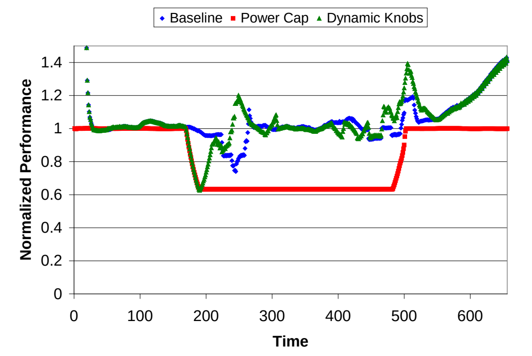
SWISH++



swaptions



x264



Maintains performance despite noise

Perfect behavior

Maintains baseline performance

Dynamic knobs automatically enable dynamic response for a range of applications using a single mechanism

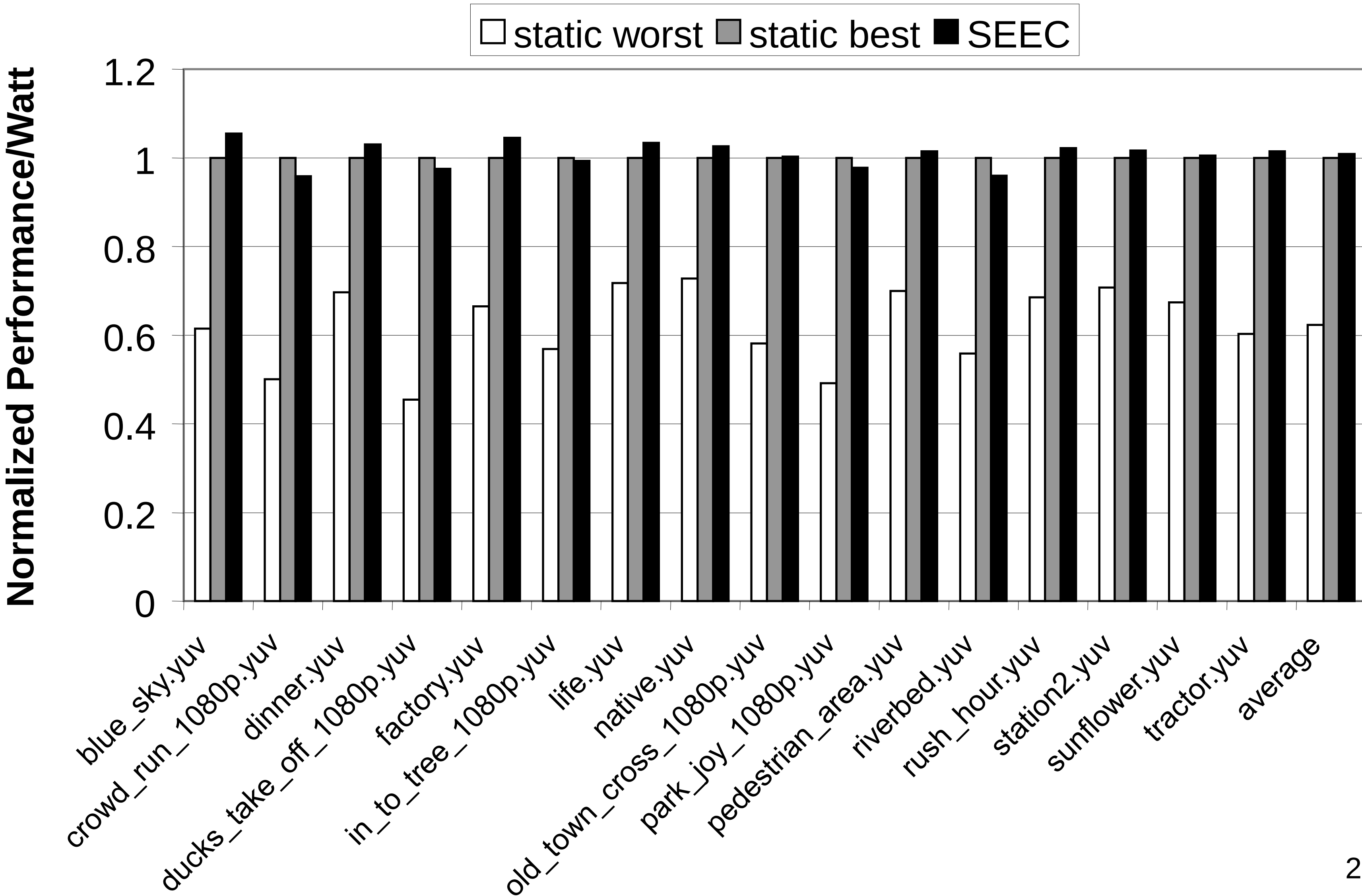
Optimizing Performance per Watt for Video Encoding

Adapt system behavior to needs of individual inputs

Application Goals	Maintain 30 frame/s while minimizing power
System Actions	Change cores, clock speed, and memory bandwidth
Experiment	Benchmark: x264 w/ 16 different 1080p inputs Compare performance/Watt w/ SEEC to best static allocation of resources

Results

Optimizing Performance/Watt for Video Encoder



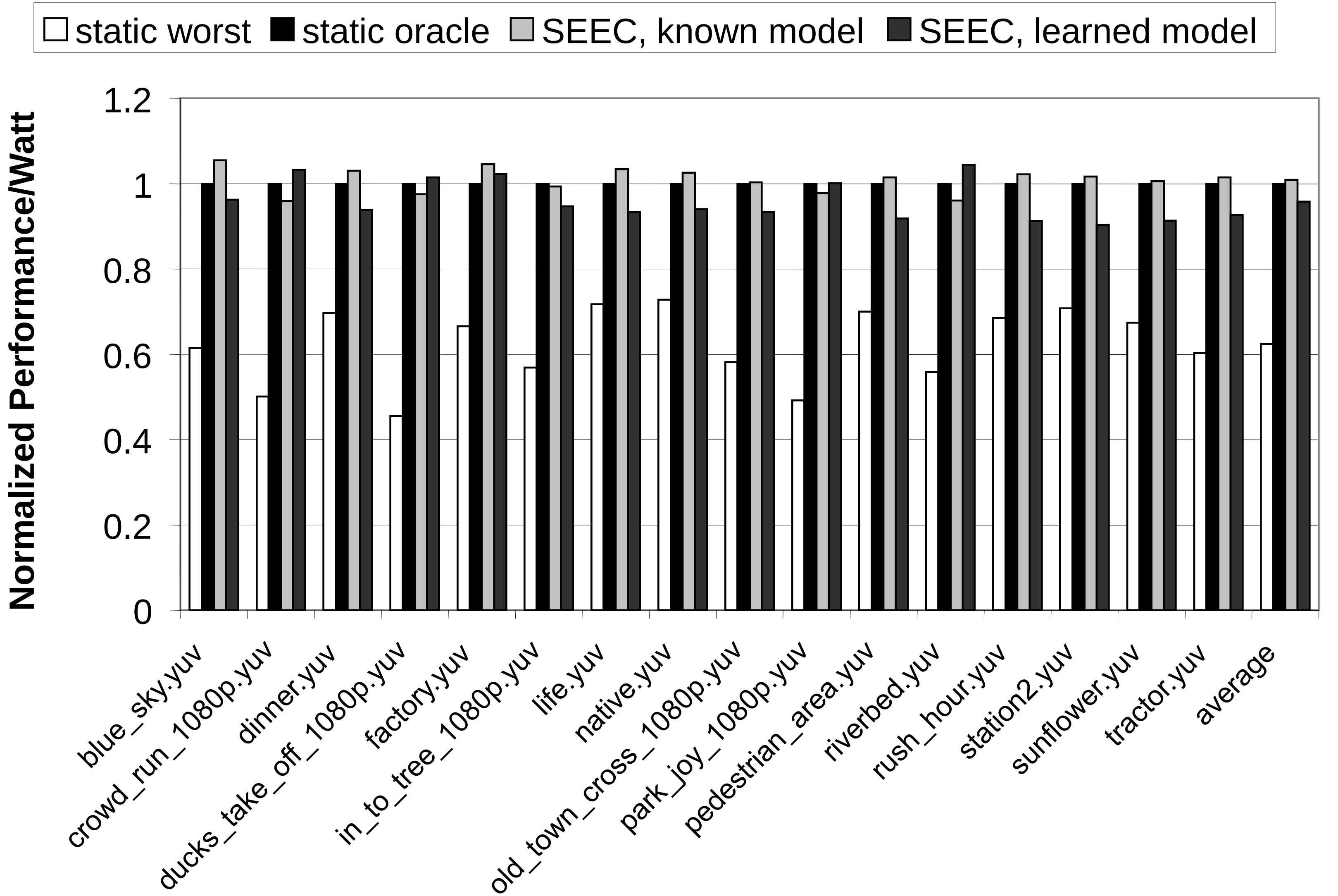
Learning Models Online

Adapt system behavior to needs of individual inputs

Application Goals	Maintain 30 frame/s while minimizing power
System Actions	Change cores, clock speed, and memory bandwidth Tailor models to individual applications while running
Experiment	Benchmark: x264 w/ 16 different 1080p inputs Compare performance/Watt w/ learned model to previous measurements

Results

Performance/Watt with Online Learning



Managing Application and System Resources Concurrently

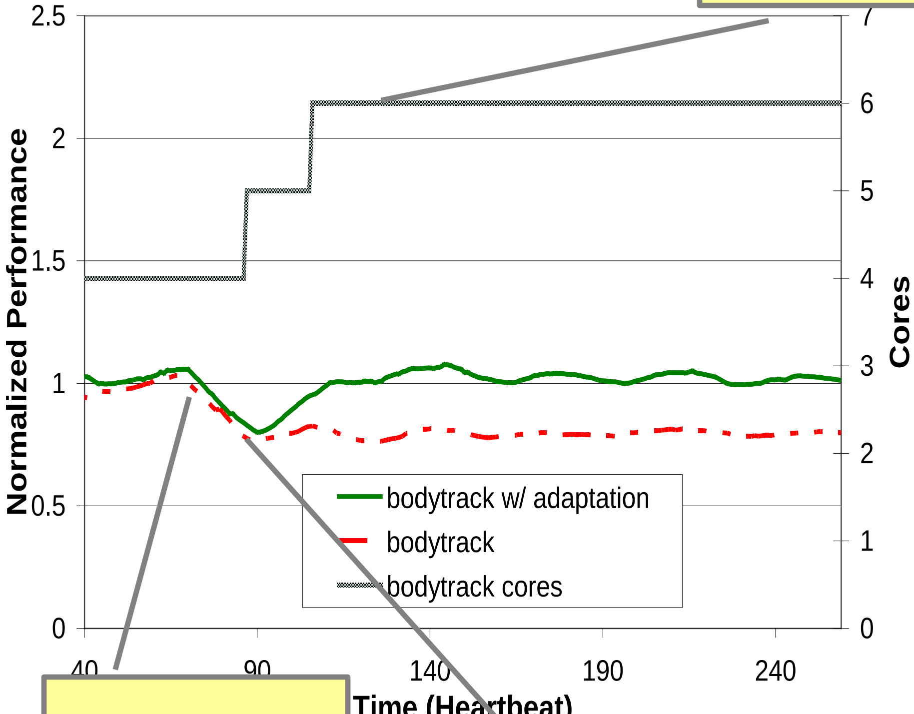
Manage multiple applications when clock frequency changes

Application Goals	**bodytrack:** maintain performance, minimize power **x264:** maintain performance, minimize quality loss
System Actions	Change core allocation to both applications Change x264's algorithms
Experiment	Maintain performance of both application when clock frequency changes

Results

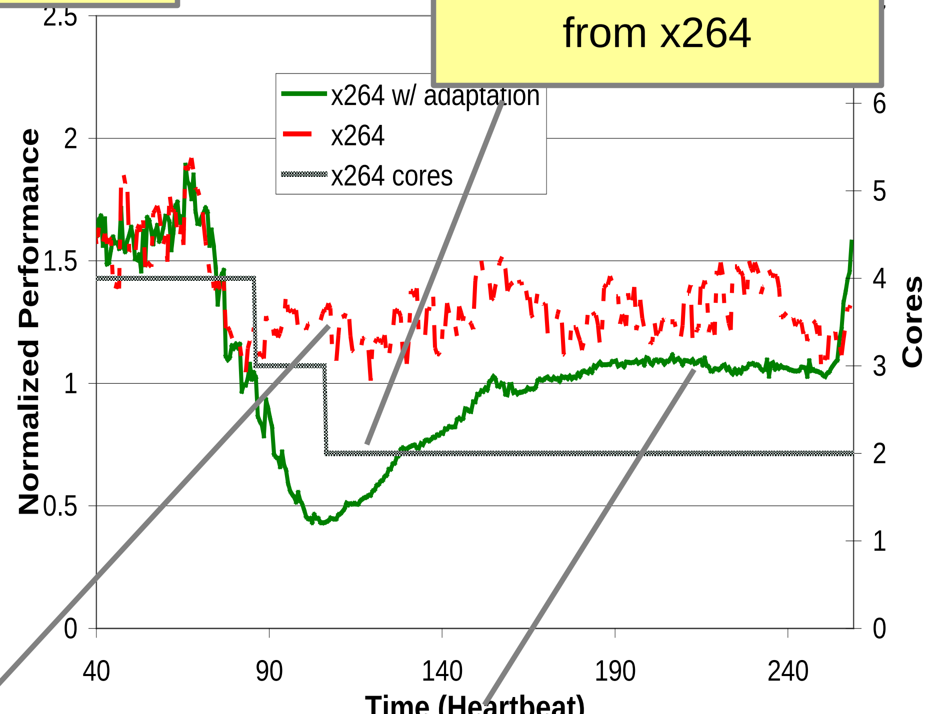
SEEC Management of Multiple Applications

bodytrack



SEEC allocates cores to bodytrack

x264



SEEC removes cores from x264

Clock drops 2.4-1.6GHz

w/o SEEC app **misses** goals

w/o SEEC app **exceeds** goals

SEEC adjusts algorithm to meet goals

Summary of Experiments

Experiment	Demonstrated Benefit of SEEC
Dynamic Knobs	Maintains application performance in the face of loss of compute resources
Performance/Watt	Out-performs oracle for static allocation of resources by adapting to fluctuations in input data
Performance/Watt with learning	Learns models online and still achieve 95% of static oracle
Multi-App control	Maintains performance of multiple apps by managing algorithm and system resources to adapt to loss of compute resources

Outline

- Introduction/Motivating Example
- The SEEC Framework
- Experimental Validation
- Conclusions

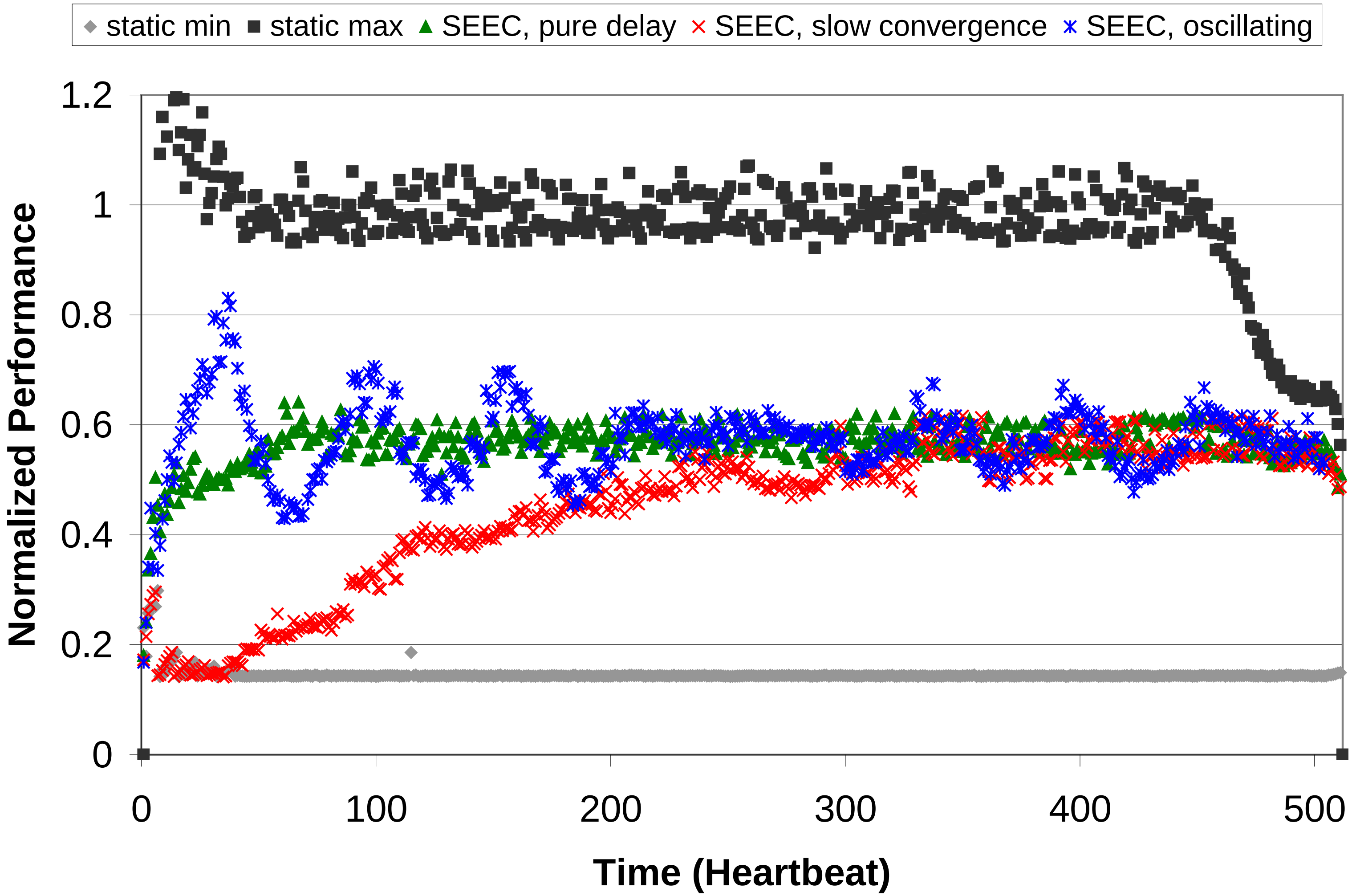
SEEC References

- Application Heartbeats framework:
  - Hoffmann, Eastep, Santambrogio, Miller, Agarwal. Application Heartbeats: A Generic Interface for Specifying Program Performance and Goals in Autonomous Computing Environments. ICAC 2010
- Control Systems:
  - Maggio, Hoffmann, Santambrogio, Agarwal, Leva. Controlling software applications within the Heartbeats framework. CDC 2010
  - Maggio, Hoffmann, Santambrogio, Agarwal, Leva. Power-Aware Design for Embedded Systems. Under Review.
- Adaptive Applications:
  - Hoffmann, Misailovic, Sidiroglou, Agarwal, Rinard. Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures. MIT-CSAIL-TR-2209-042. 2009
  - Hoffmann, Sidiroglou, Carbin, Misailovic, Agarwal, Rinard. Dynamic Knobs for Power-Aware Computing. ASPLOS 2011.
- The SEEC Framework:
  - Hoffmann, Maggio, Santambrogio, Leva, Agarwal. SEEC: A Framework for Self-aware Computing. MIT-CSAIL-TR-2010-049 2010.

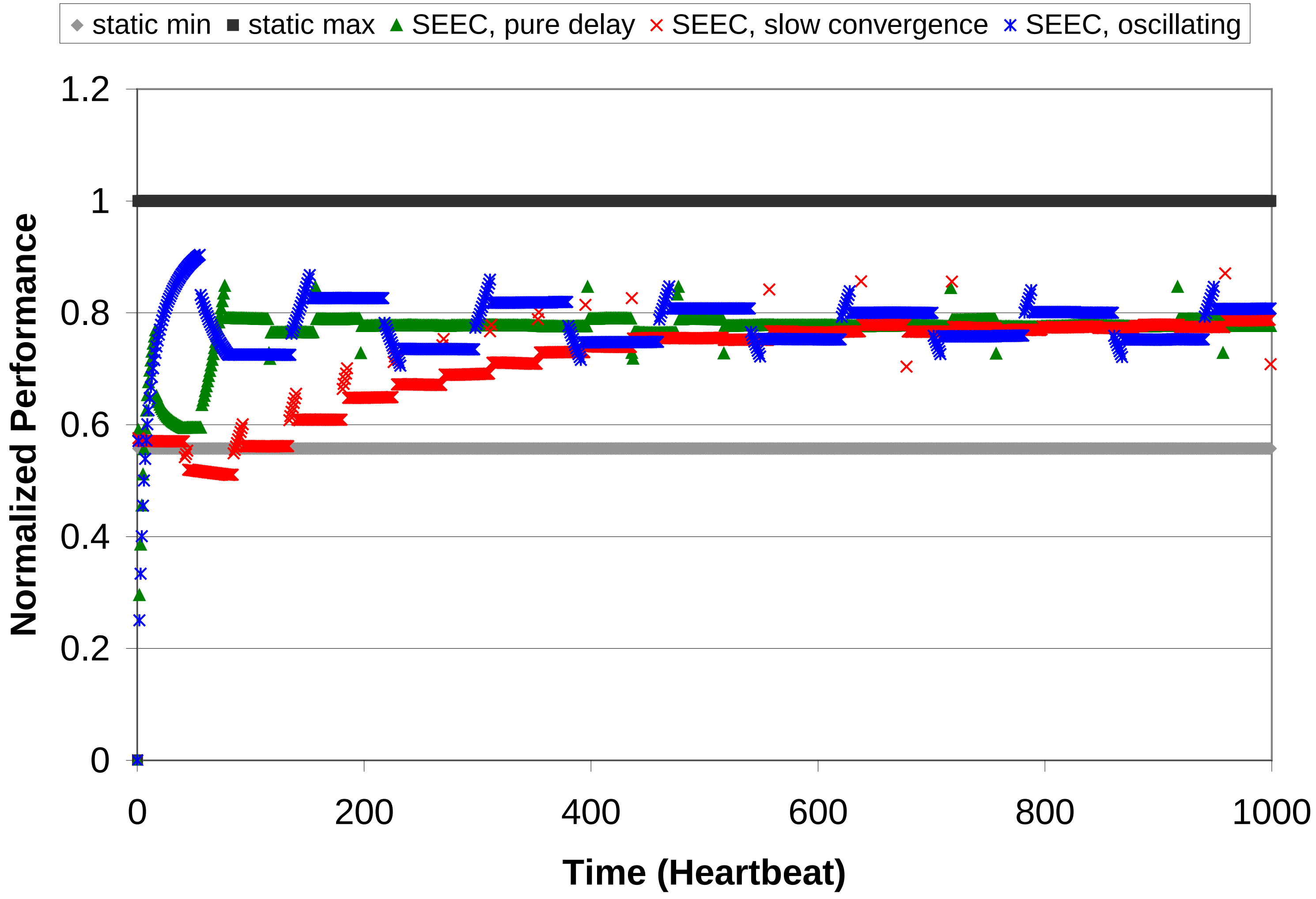
Conclusions

- SEEC is designed to help ease programmer burden
  - Solves resource allocation problems
  - Adapts to fluctuations in environment
- SEEC has two distinguishing features
  - Incorporates goals and feedback directly from the application
  - Abstracts sensors, controller, and actuator to create a generic feedback control system capable of managing algorithm, software, and hardware adaptation
- Demonstrated the benefits of SEEC in several experiments
  - SEEC can optimize performance per Watt for video encoding
  - SEEC can adapt algorithms and resource allocation to meet goals in the face of power caps or other changes in environment

Backup Slides



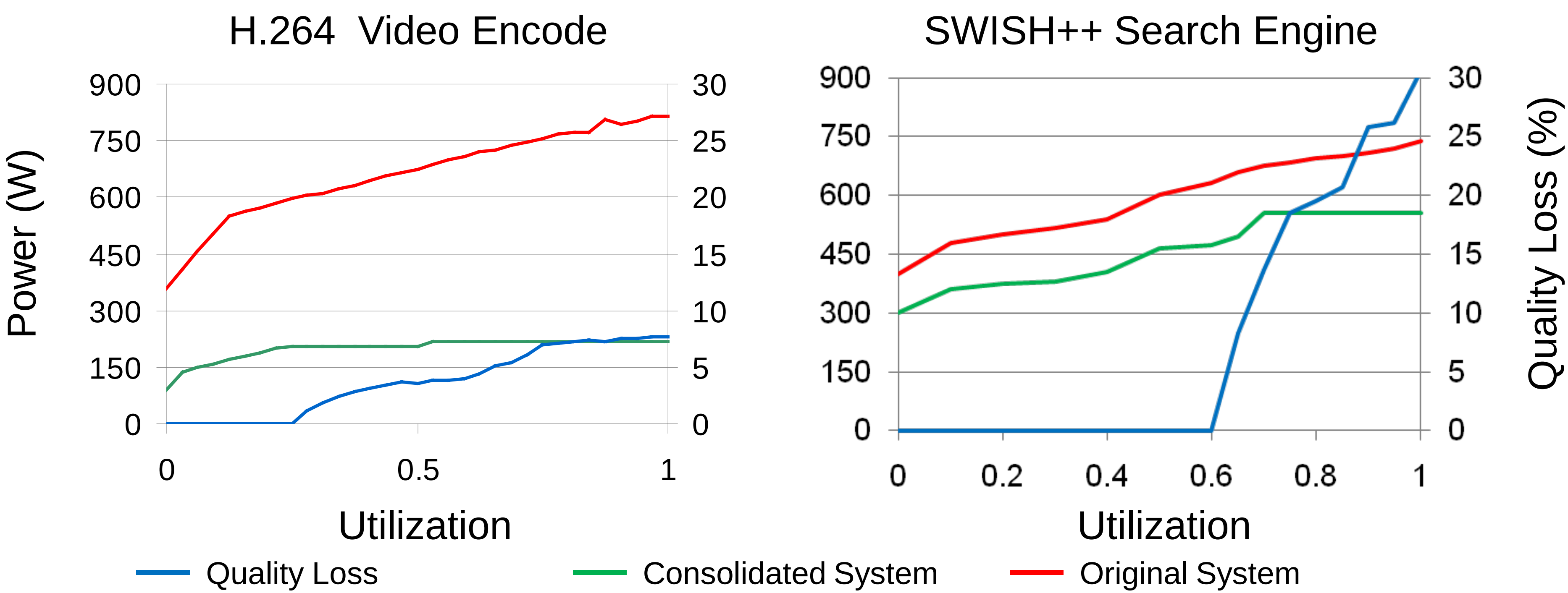
Controlling Memory Bandwidth for STREAM



Using Code Perforation to Save Power in Server Farms

- Currently peak load met by provisioning extra hardware
- Instead, we reduce hardware
- At low loads, no perforation necessary
- At high loads, perforation increases capacity
  - Runtime detects performance degradation from load
  - Runtime adjusts perforation in running apps to respond to load
  - Same peak load met with fewer machines
- Tested by consolidating mini-server farm

Power Saving With Code Perforation



- Power:
  - Up to 3/4 reduction in machines and power for video
  - Up to 1/3 reduction in machines and power for search
- Quality:
  - Max 8% loss for video
  - Max 30% loss for search (Fewer total results – precision of top 10 unchanged)