IBM Research

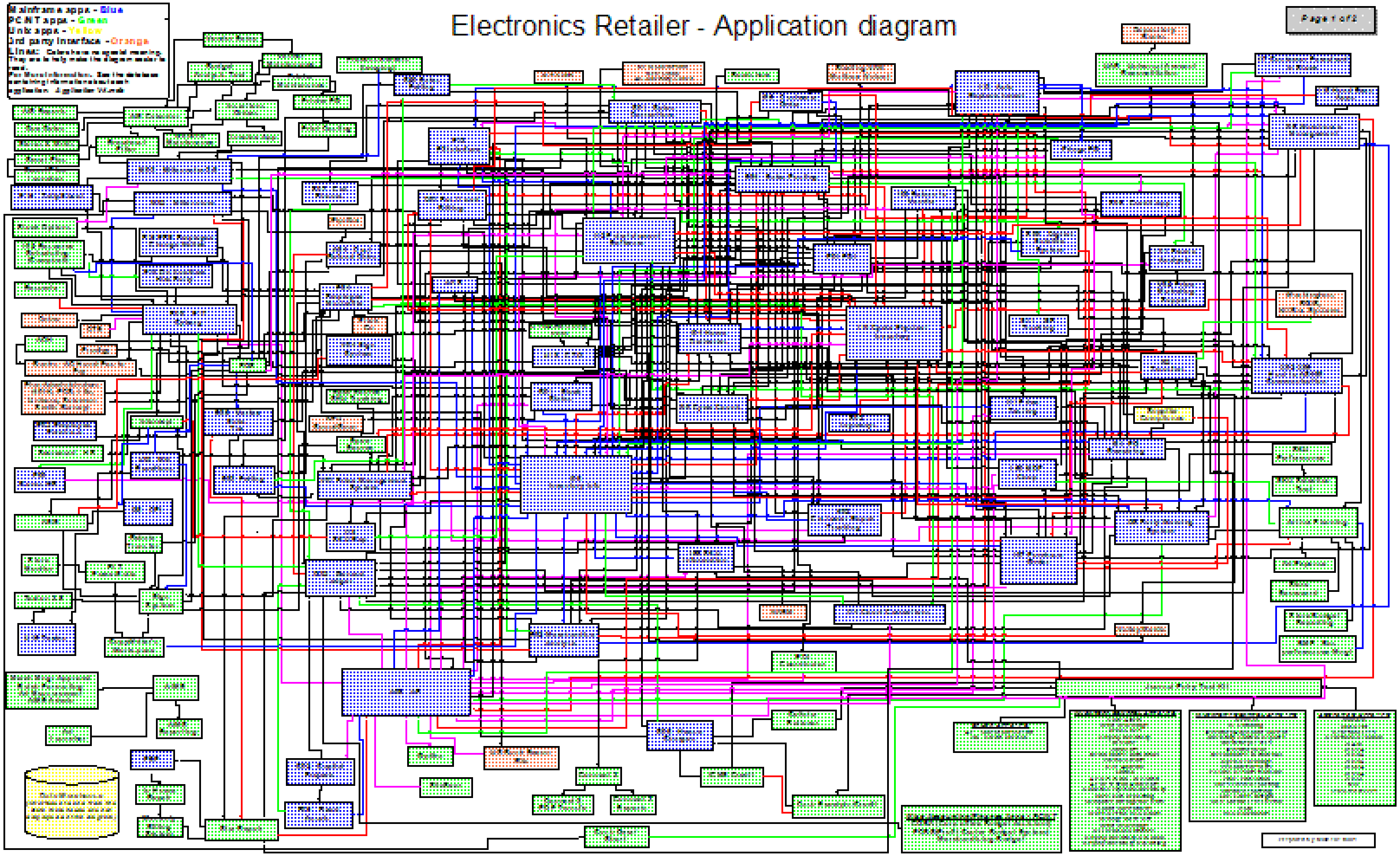
The Utility of Utility: Policies for Autonomic Computing

Jeff Kephart (kephart@us.ibm.com)
IBM Thomas J Watson Research Center
Hawthorne, NY, USA

LCCC 2011

December 6, 2011

IT is Becoming Too Complex!



Autonomic Computing and Agents

- AC definition
  - "Computing systems that manage themselves in accordance with high-level objectives from humans." Kephart & Chess, IEEE Computer 2003
  - Self-configuring, self-healing, self-optimizing, self-protecting
- Agents definition
  - "An encapsulated computer system, situated in some environment, and capable of flexible, autonomous action in that environment in order to meet its design objectives." Jennings, et al, *A Roadmap of Agent Research and Development*, JAAMAS 1998
- Autonomic elements ~ agents
- Autonomic systems ~ multi-agent systems

COVER FEATURE

The Vision of Autonomic Computing

Systems manage themselves according to an administrator's goals. New components integrate as effortlessly as a new cell establishes itself in the human body. These ideas are not science fiction, but elements of the grand challenge to create self-managing computing systems.

Jeffrey O. Kephart
David M. Chess
IBM Thomas J. Watson Research Center

In mid-October 2001, IBM released a manifesto observing that the main obstacle to further progress in the IT industry is a looming software complexity crisis.[1] The company cited applications and environments that weigh in at tens of millions of lines of code and require skilled IT professionals to install, configure, tune, and maintain.

The manifesto pointed out that the difficulty of managing today's computing systems goes well beyond the administration of individual software environments. The need to integrate several heterogeneous environments into corporate-wide computing systems, and to extend that beyond company boundaries into the Internet, introduces new levels of complexity. Computing systems' complexity appears to be approaching the limits of human capability, yet the march toward increased interconnectivity and integration rushes ahead unabated.

This march could turn the dream of pervasive computing—trillions of computing devices connected to the Internet—into a nightmare. Programming language innovations have extended the size and complexity of systems that architects can design, but relying solely on further innovations in programming methods will not get us through the present complexity crisis.

As systems become more interconnected and diverse, architects are less able to anticipate and design interactions among components, leaving such issues to be dealt with at runtime. Soon systems will become too massive and complex for even the most skilled system integrators to install, configure, optimize, maintain, and merge. And there will be no way to make timely, decisive responses to the rapid stream of changing and conflicting demands.

AUTONOMIC OPTION

The only option remaining is *autonomic computing*—computing systems that can manage themselves given high-level objectives from administrators. When IBM's senior vice president of research, Paul Horn, introduced this idea to the National Academy of Engineers at Harvard University in a March 2001 keynote address, he deliberately chose a term with a biological connotation. The autonomic nervous system governs our heart rate and body temperature, thus freeing our conscious brain from the burden of dealing with these and many other low-level, yet vital, functions.

The term autonomic computing is emblematic of a vast and somewhat tangled hierarchy of natural self-governing systems, many of which consist of myriad interacting, self-governing components that in turn comprise large numbers of interacting, autonomous, self-governing components at the next level down. The enormous range in scale, starting with molecular machines within cells and extending to human markets, societies, and the entire world socioeconomy, mirrors that of computing systems, which run from individual devices to the entire Internet. Thus, we believe it will be profitable to seek inspiration in the self-governance of social and economic systems as well as purely biological ones.

Clearly then, autonomic computing is a grand

0018-9162/03/$17.00 © 2003 IEEE Published by the IEEE Computer Society January 2003

Kephart and Chess, *The Vision of Autonomic Computing, IEEE Computing, January 2003.*

The focus of this talk

- I start from two premises:
  - Autonomic systems are "Computing systems that manage themselves in accordance with high-level objectives from humans."
  - Autonomic systems ~ multi-agent systems
- Which leads to…
- How do we get a (decentralized) Multi-Agent System to act in accordance with high-level objectives?
- My claim
  - Objectives should be expressed in terms of ***utility***
  - ***Utility*** is an essential piece of information that must be processed, transformed, and communicated by agents

Outline

- Autonomic Computing and Multi-Agent Systems

- **Utility Functions**
  - **As means for expressing high-level objectives**
  - **As means for managing to high-level objectives**

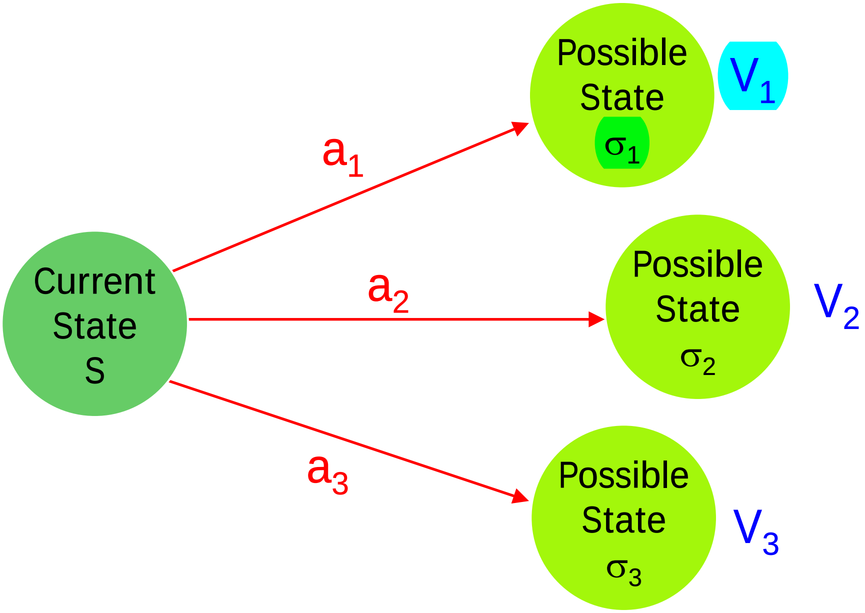
- Examples
  - Unity, and its commercialization
  - Power and performance objectives and tradeoffs
  - Applying utility concepts at the data center level

- Conclusions

How to *represent* high-level policies?

Kephart and Walsh, Policy04

- Utility functions map any possible state of a system to a scalar value
- They can be obtained from
  - Service Level Agreement
  - preference elicitation
  - simple templates
- They are a very useful representation for high-level objectives
  - Value can be transformed and propagated among agents to guide system behavior


Current State S

a_1

a_2

a_3

Possible State σ_1

V_1

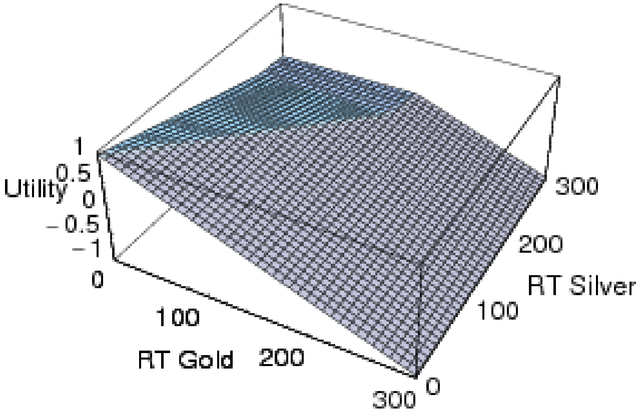
Possible State σ_2

V_2

Possible State σ_3

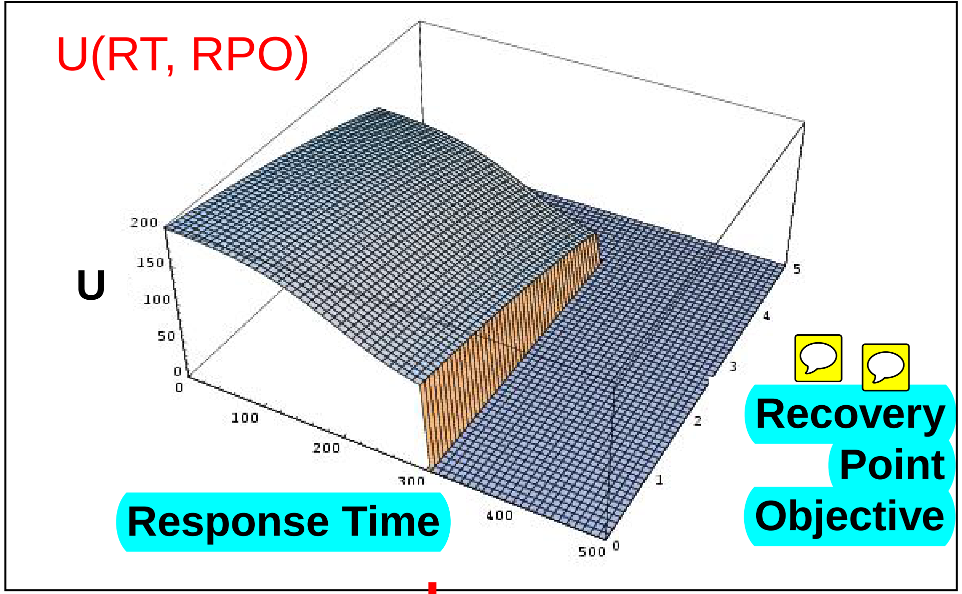
V_3

U(**RT**) =

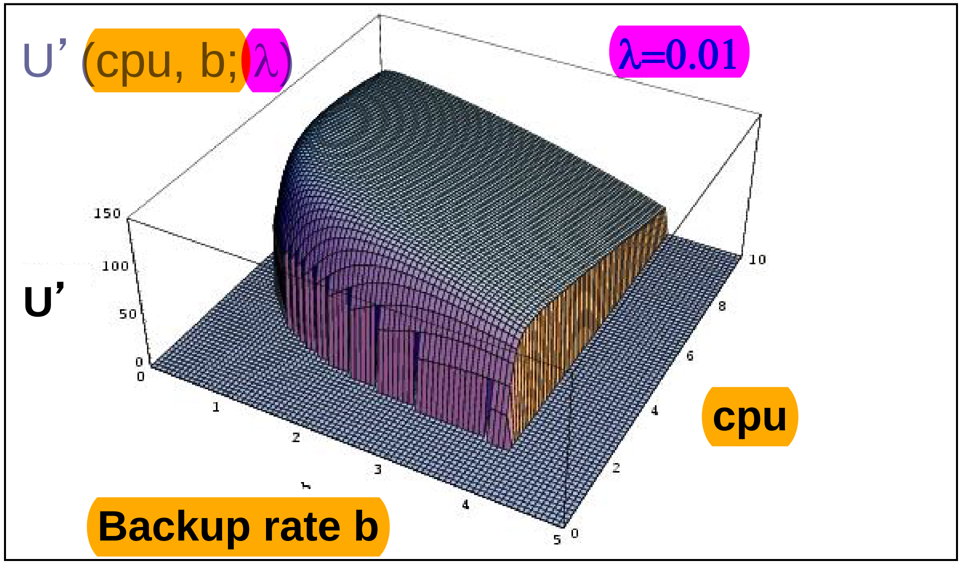


How to *manage* with high-level policies?

- **Elicit** utility function U(**S**) expressed in terms of service attributes **S**
- **Model** how each attribute S_i depends on controls **C** and observables **O**
  - Models expressed as **S**(**C**; **O**)
  - E.g., RT(routing weights, request rate)
  - Models from experiments, learning, theory
- **Transform** from service utility U to resource utility U’ by substitution
  - $U(\mathbf{S}) = U(\mathbf{S}(\mathbf{C}; \mathbf{O})) = U'(\mathbf{C}; \mathbf{O})$
- **Optimize** resource utility. As observable **O** changes, set **C** to values that maximize U’ (**C**; **O**)
  - $\mathbf{C*(O)} = \mathrm{argmax}_C\ U'(\mathbf{C}; \mathbf{O})$
  - $U'*(\mathbf{O}) = U'(\mathbf{C*(O)}; \mathbf{O})$



Transform



Outline

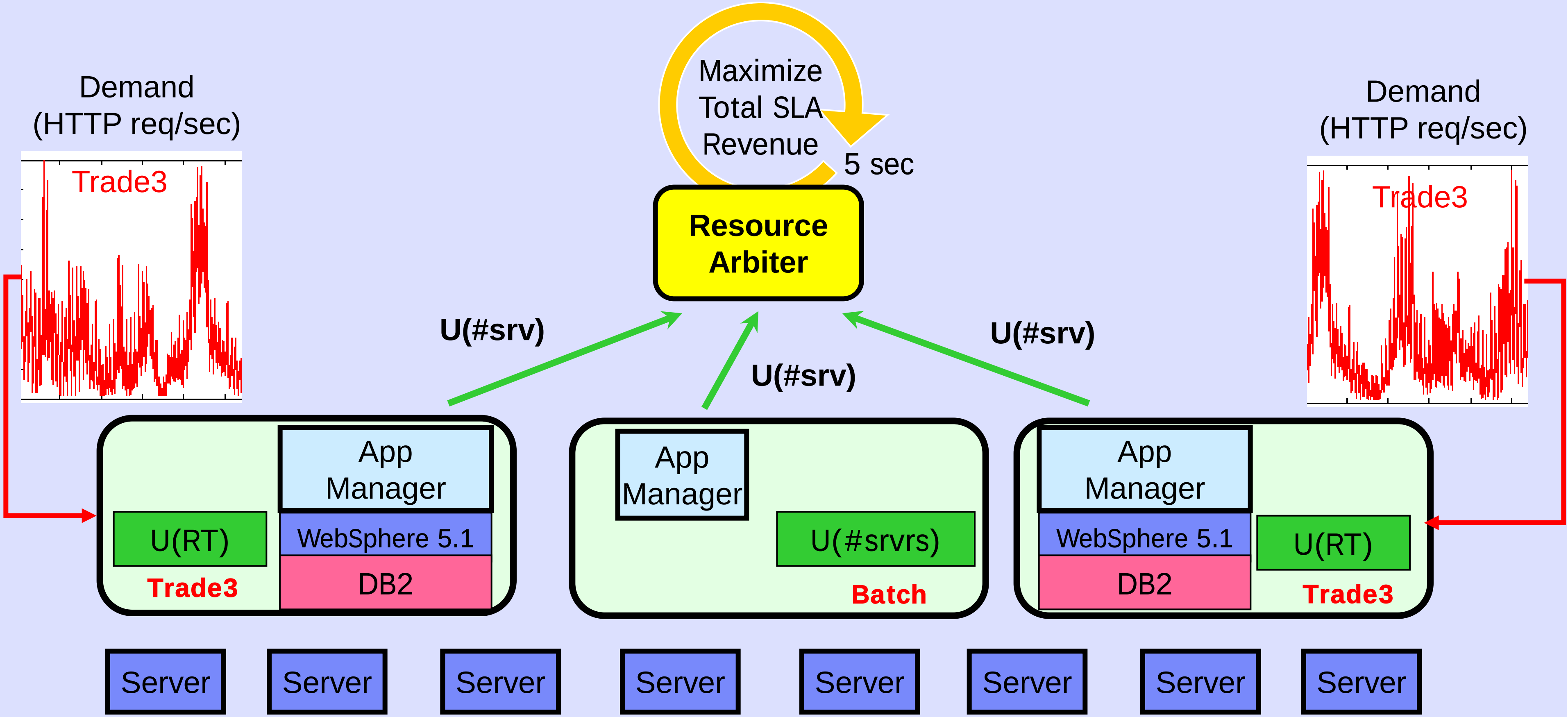
- Autonomic Computing and Multi-Agent Systems

- Utility Functions
  - As means for expressing high-level objectives
  - As means for managing to high-level objectives

- **Examples**
  - **Unity, and its commercialization**
  - Power and performance objectives and tradeoffs
  - Applying utility concepts at the data center level

- Conclusions

Unity Data Center Prototype: Experimental setup

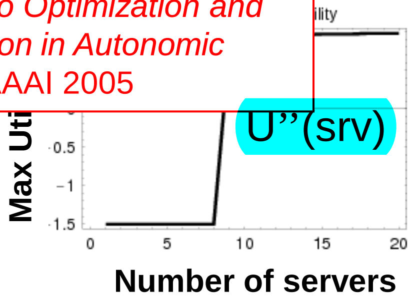


Chess, Segal, Whalley and White, *Unity: Experiences with a Prototype Autonomic Computing System,* ICAC 2004

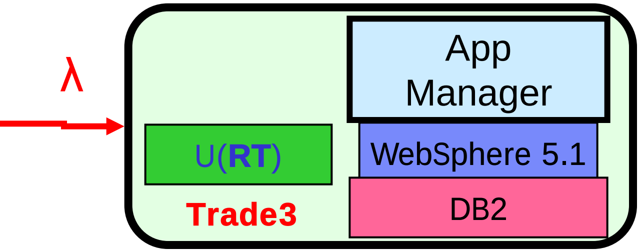
How App Mgr computes its external resource utility

Alternative to generating full curve: utility elicitation

Patrascu, Boutilier et al. *New Approaches to Optimization and Utility Elicitation in Autonomic Computing,* AAAI 2005



Resource Arbiter



Elicit: $U(\mathbf{RT})$ Service-level utility

My controls | Arbiter's controls | Observable

Model: $U(\mathbf{RT}(\mathbf{C};\ \text{srv}, \lambda))$

Transform: $U'(\mathbf{C};\ \text{srv}, \lambda) = U(\mathbf{RT}(\mathbf{C};\ \text{srv}, \lambda))$

Internal resource-level utility

Optimize:

Optimal internal control settings $\mathbf{C^*}(\text{srv}, \lambda) = \text{argmax}_{\mathbf{C}} U'(\mathbf{C};\ \text{srv}, \lambda)$

External resource-level utility $U''(\text{srv}, \lambda) = U'(\mathbf{C^*}(\text{srv}, \lambda);\ \text{srv}, \lambda)$

Chess, Segal, Whalley and White, *Unity: Experiences with a Prototype Autonomic Computing System,* ICAC 2004

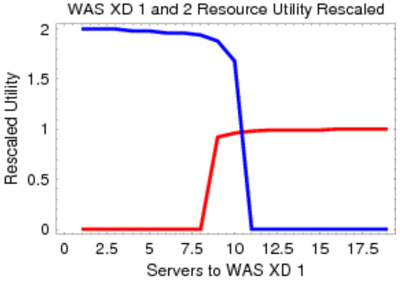
How the Arbiter determines optimal resource allocation

Decision problem:

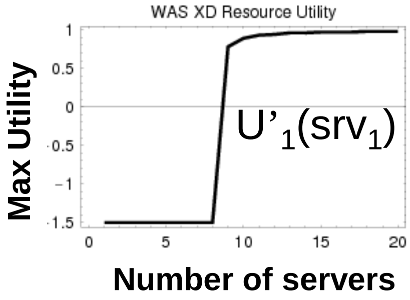
Allocate resources

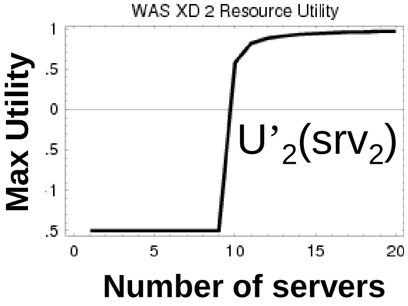
srv* = argmax$_{\mathbf{srv}}\Sigma U''_i(srv_i)$

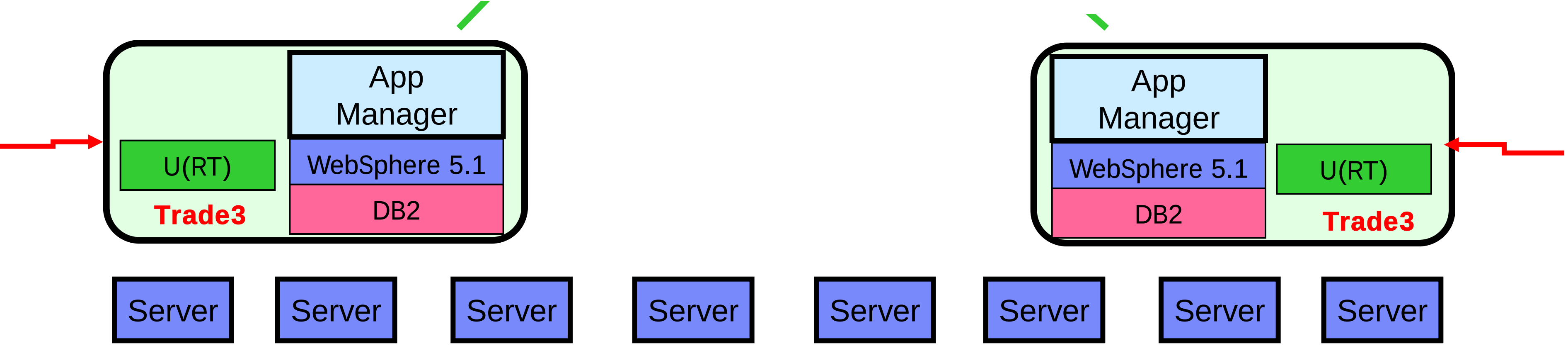
Effectively maximizes $\Sigma U_i(\mathbf{S}_i)$



Resource Arbiter

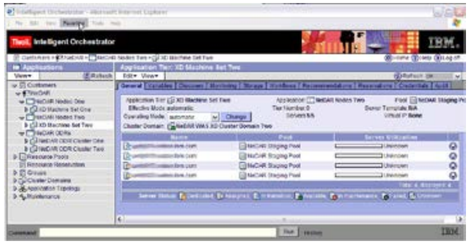




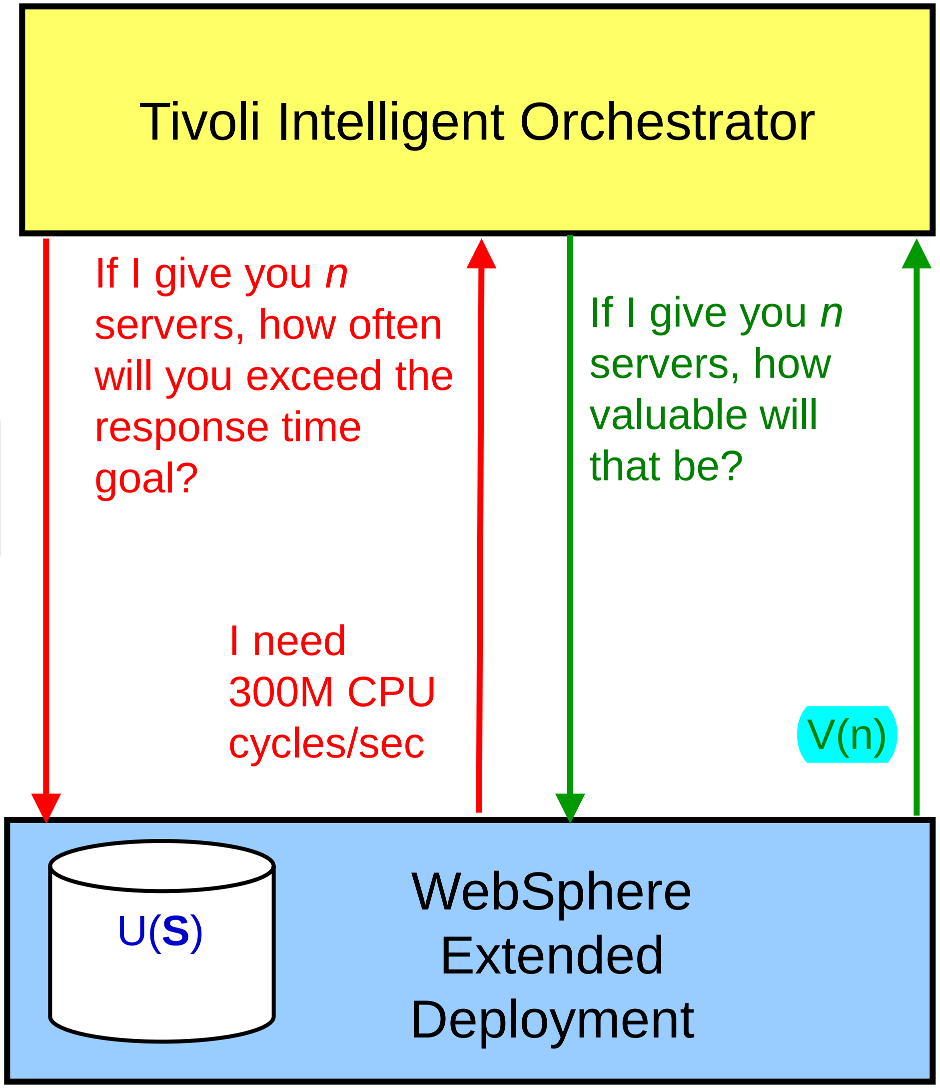


How we commercialized Unity

Das et al., ICAC 2006



The confusing old way



Tivoli Intelligent Orchestrator

If I give you *n* servers, how often will you exceed the response time goal?

I need 300M CPU cycles/sec

If I give you *n* servers, how valuable will that be?

V(n)

U(**S**)

WebSphere Extended Deployment

The clean new way

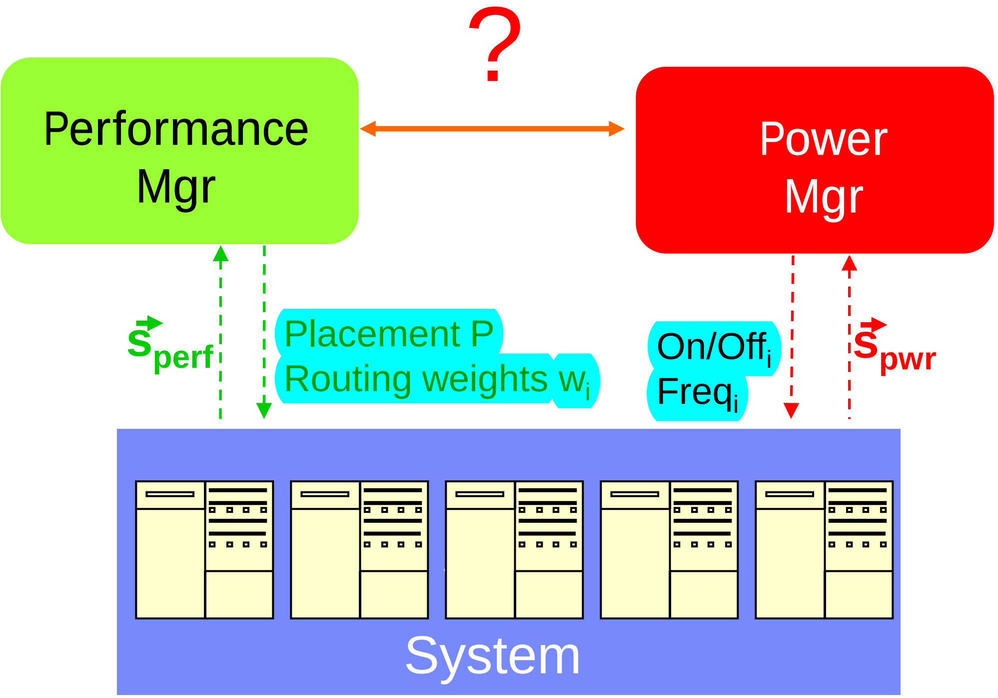
This was not actually that simple – product release cycles didn't mesh, so we needed an evolutionary approach.

Outline

- Autonomic Computing and Multi-Agent Systems
- Utility Functions
  - As means for expressing high-level objectives
  - As means for managing to high-level objectives
- **Examples**
  - Unity, and its commercialization
  - **Power and performance objectives and tradeoffs**
  - Applying utility concepts at the data center level
- Conclusions

Utility functions for interacting power-performance agents

- How to trade off power vs performance?
  - In an individual machine
  - In a rack of machines
  - In an entire data center
- Formulate a joint power-performance utility function **U**(performance, power)
  - Maximize U($\mathbf{s}_{\mathbf{perf}}$, $\mathbf{s}_{\mathbf{pwr}}$)
  - Often just U($\mathbf{s}_{\mathbf{perf}}$) – ε **pwr**
- How to optimize **U**?
- How can semi-autonomous power and performance agents cooperatively optimize **U?**
  - Mediated through coordinator?
  - Direct bilateral interactions?



Kephart, Chan, Das, Levine, Tesauro, Rawson, Lefurgy. *Coordinating Multiple Autonomic Managers to Achieve Specified Power-Performance Tradeoffs*. **ICAC 2007.** (Emergent phenomena can occur when autonomic managers don't communicate effectively.)

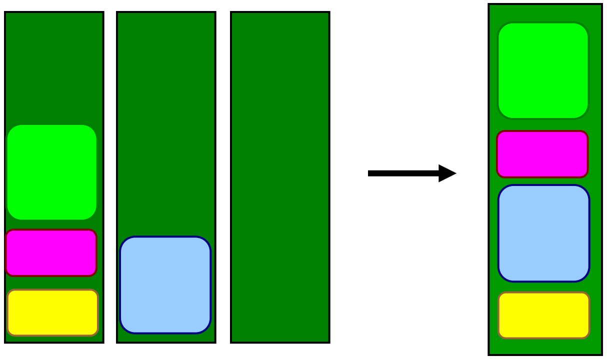
Hanson et al. *Autonomic Manager for Power*, **NOMS 2010**

Steinder, Whalley, Hanson, Kephart, *Coordinated Management of Power Usage and Runtime Performance*, **NOMS 2008**

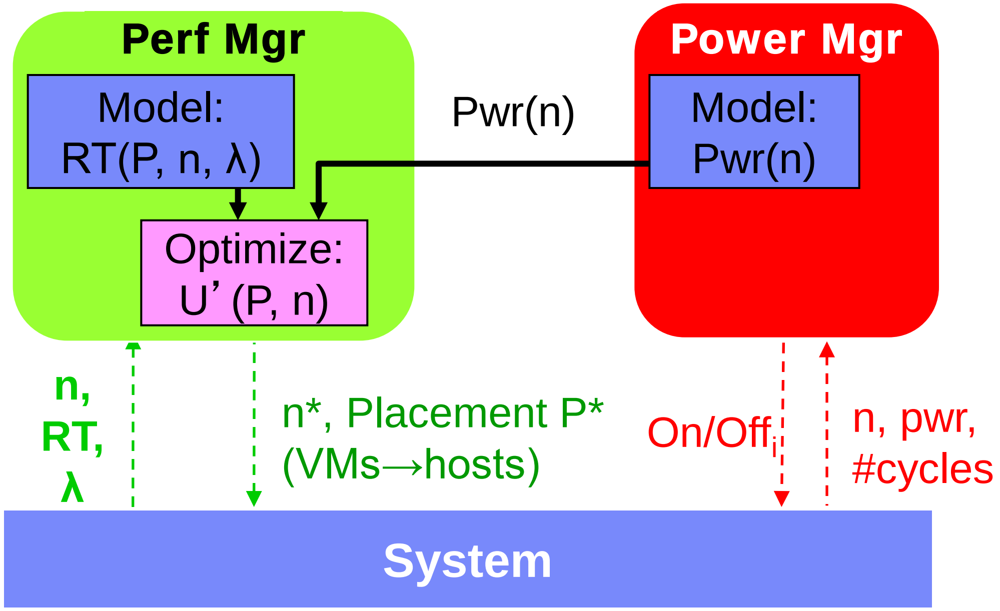
Das, Kephart, Lefurgy, Tesauro, Levine, Chan *Autonomic Multi-agent Management of Power and Performance in Data Centers*, **AAMAS 2008**

Power-aware dynamic server consolidation

Goal: Save power by dynamically migrating VMs so as to occupy fewer servers without sacrificing performance too much. Turn unused servers off.



Maximize U(RT, pwr)



Perf Mgr

Model: RT(P, n, λ)

Optimize: U' (P, n)

Pwr(n)

Power Mgr

Model: Pwr(n)

n, RT, λ

n*, Placement P* (VMs→hosts)

On/Off_i

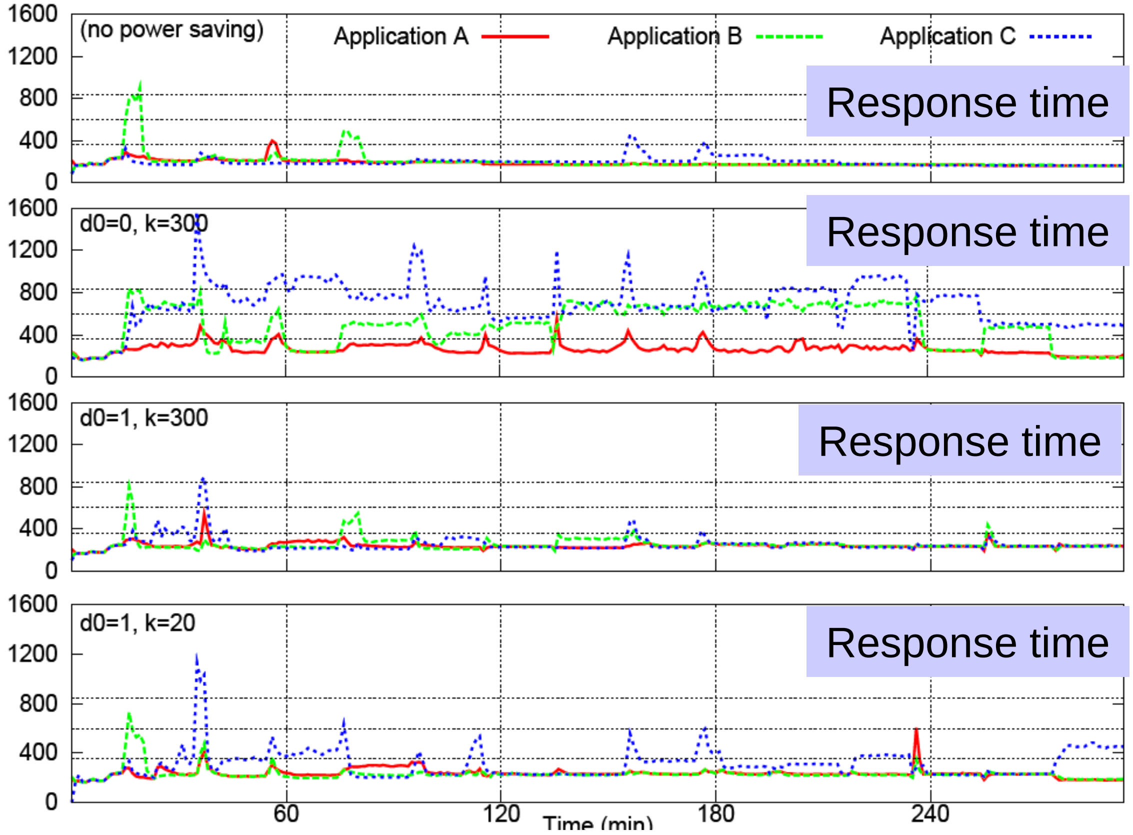
n, pwr, #cycles

System

Steinder, Whalley, Hanson, Kephart, *Coordinated Management of Power Usage and Runtime Performance*, **NOMS 2008**

Experimental results

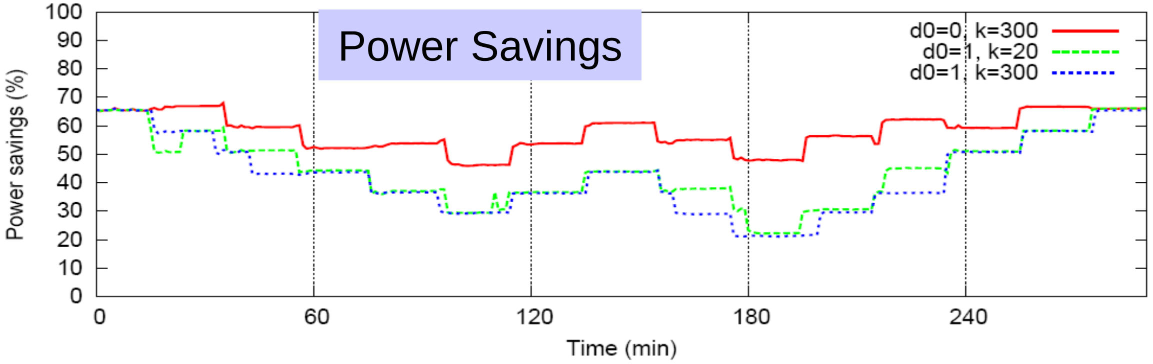
(3 different utility functions)



1. Always meet SLAs

2. Always maximize performance

3. Permit 10% performance degradation for 10% power savings



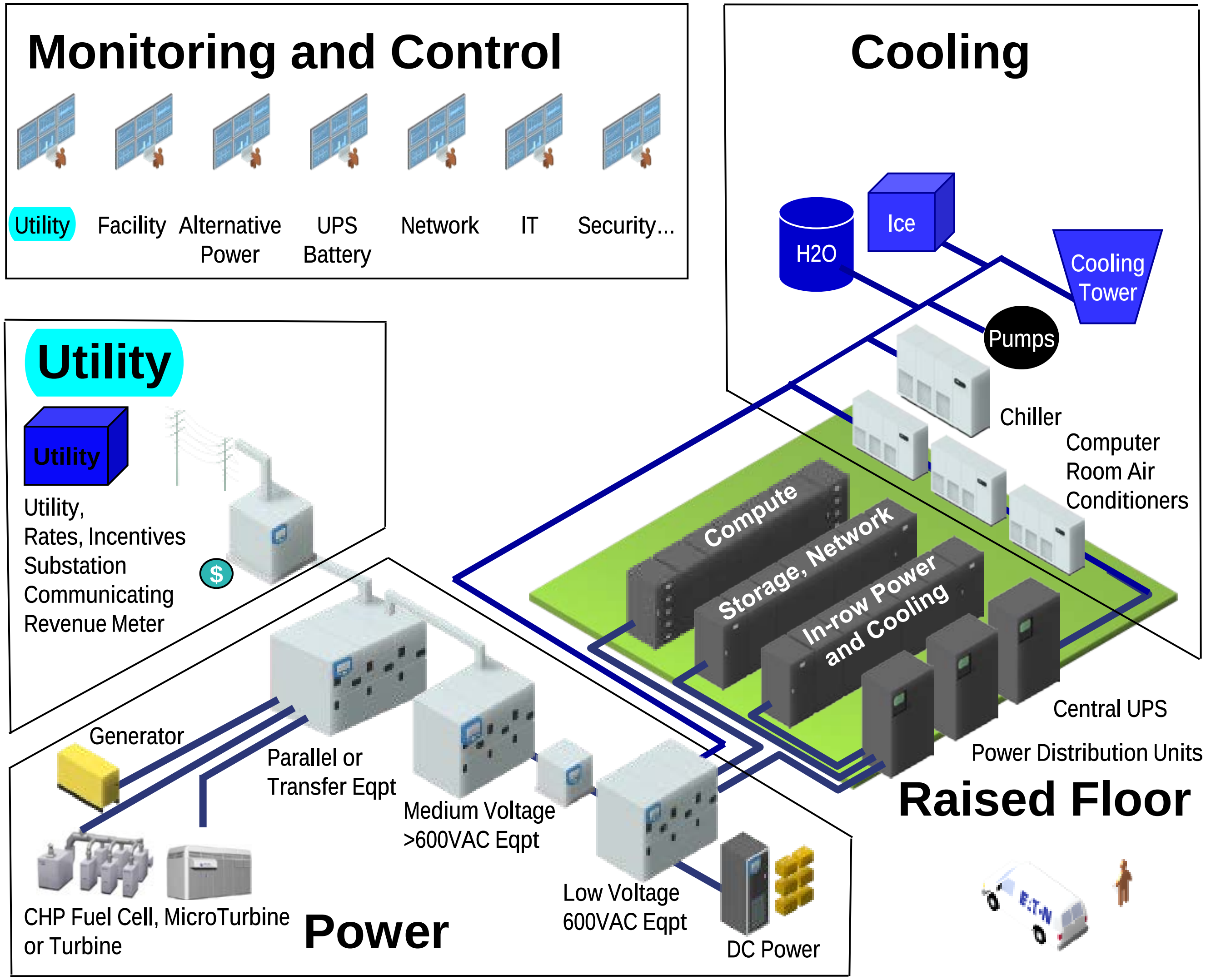
Conclusions. Substantial power savings (up to 65%) can be attained without violating SLA.

Results are significantly affected by utility function choice.

Outline

- Autonomic Computing and Multi-Agent Systems
- Utility Functions
  - As means for expressing high-level objectives
  - As means for managing to high-level objectives
- **Examples**
  - Unity, and its commercialization
  - Power and performance objectives and tradeoffs
  - **Applying utility concepts at the data center level**
- Conclusions

The Physical Infrastructure that Supports IT is Complex!



IT and Networks

Compute
- Main Frames
- Volume Servers
- Blade Servers

Storage
- SATA Disk
- Tape
- Blended

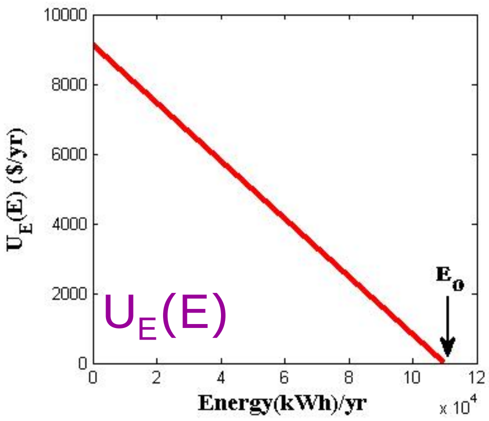
Network
- Corporate Networks
- VoIP
- Integrated Blade/Switch

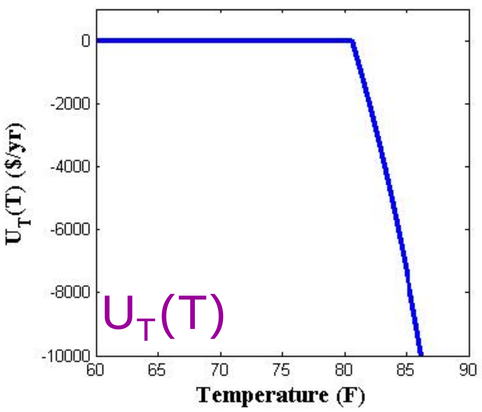
In-Row Power
- Modular UPS
- Rack Mount PDUs

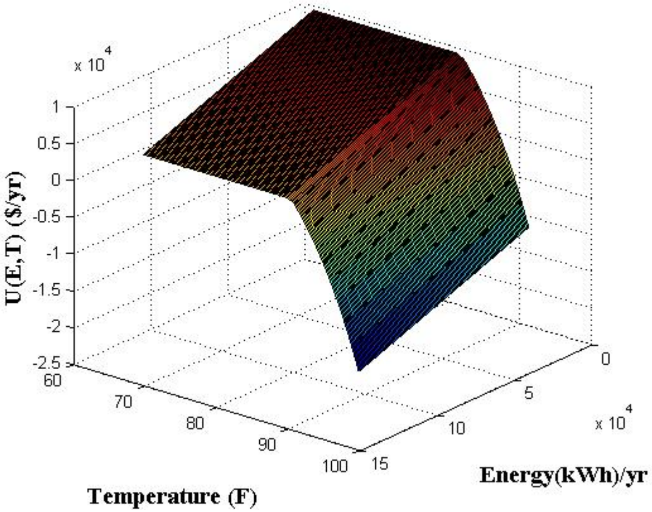
In Row Cooling
- Rear Door Heat Exchanger
- Liquid Cooling Racks
- Overhead Cooling

Trading off energy vs. temperature in a data center

- Cooling costs can account for ~50% of a data center's energy consumption, due to zealous overcooling
- Let's try using a utility function U(E, T) to manage the energy-temperature tradeoff
  - Elicitation not trivial – we tried several forms, both multiplicative and additive







$$U(E,T) = U_E(E) + U_T(T)$$

From utility to optimization

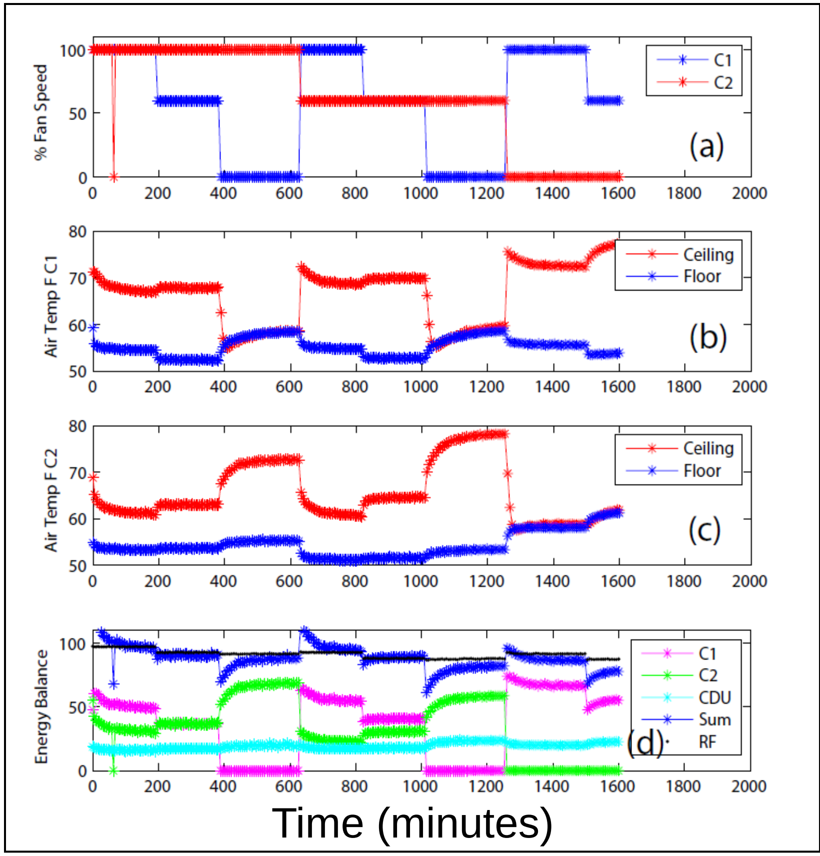
CRAC fan speeds

- **Elicit** U(E, {T})
- **Model** E(θ_1, θ_2) and T(θ_1, θ_2)
  - Via experiments varying fan speeds
  - Could also run CFD calculations
- **Transform** utility to U’(θ_1, θ_2)
- **Optimize** U’(θ_1, θ_2)
  - Set fan speeds to (θ_1, θ_2)*

(θ_1, θ_2)

T(θ_1, θ_2)

E(θ_1, θ_2)



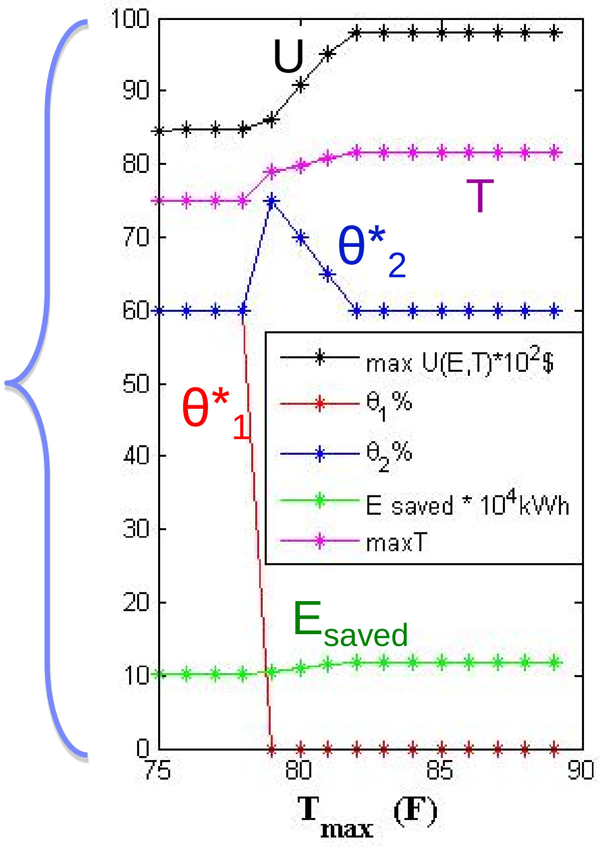
Experimental measurements

How Optimal Fan Speeds Depend upon T_{max}

If (T_{max} < 76F)
$(\theta_1, \theta_2)^* = (60\%, 60\%)$

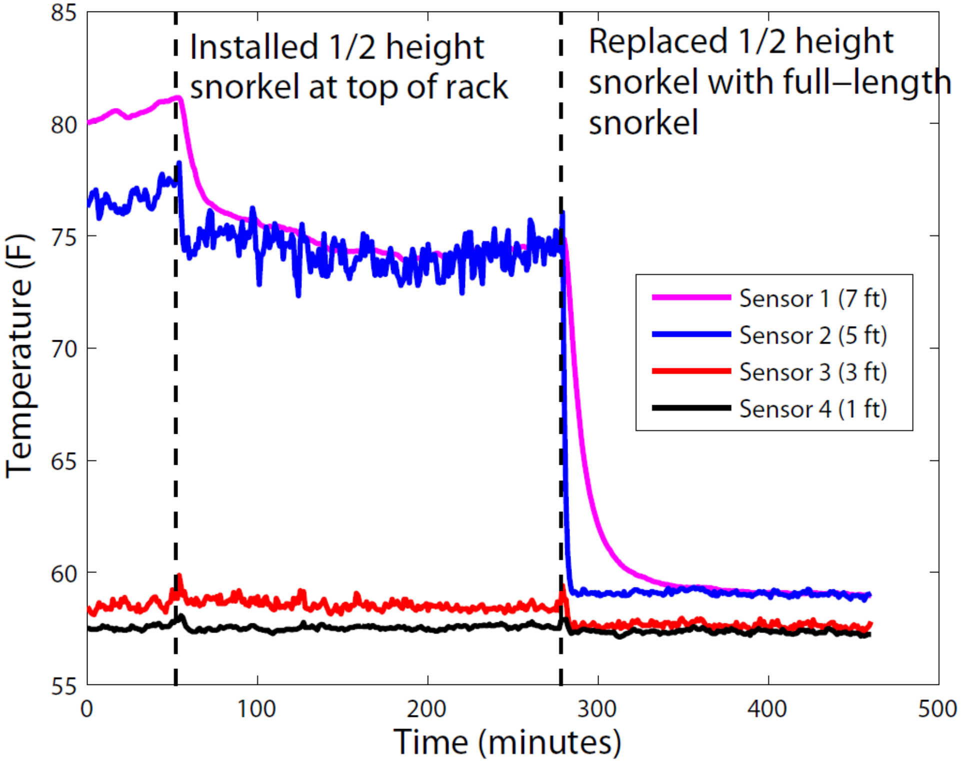
If (76 < T_{max} < 84F)
$(\theta_1, \theta_2)^* = (0\%, >60\%)$

If (T_{max} > 84F)
$(\theta_1, \theta_2)^* = (0\%, 60\%)$



Energy savings = 10 to 12%

Snorkels



Snorkels change the model $T(\theta_1, \theta_2)$; so the transformation to $U'(\theta_1, \theta_2)$ changes.

$(\theta_1, \theta_2)^*$ shifts from (60%, 60%) to (0%, 60%) for extra savings (12% →14%)

Conclusions

- Utility functions help achieve the central goal of autonomic computing
  - "Computing systems that manage themselves in accordance with high-level objectives from humans."
  - Theoretically well-grounded
  - Proven to work in practice in many domains

- Humans express objectives in terms of *value*

- *Value* is propagated, processed, and transformed by agents
  - Guides agent's internal decisions
  - Guides agent's communication with others

- Key technologies needed include
  - Utility function elicitation
  - Learning
  - Modeling / what-if modeling
  - Optimization
  - Agent communication, mediation

The next frontier? An autonomic data center economy

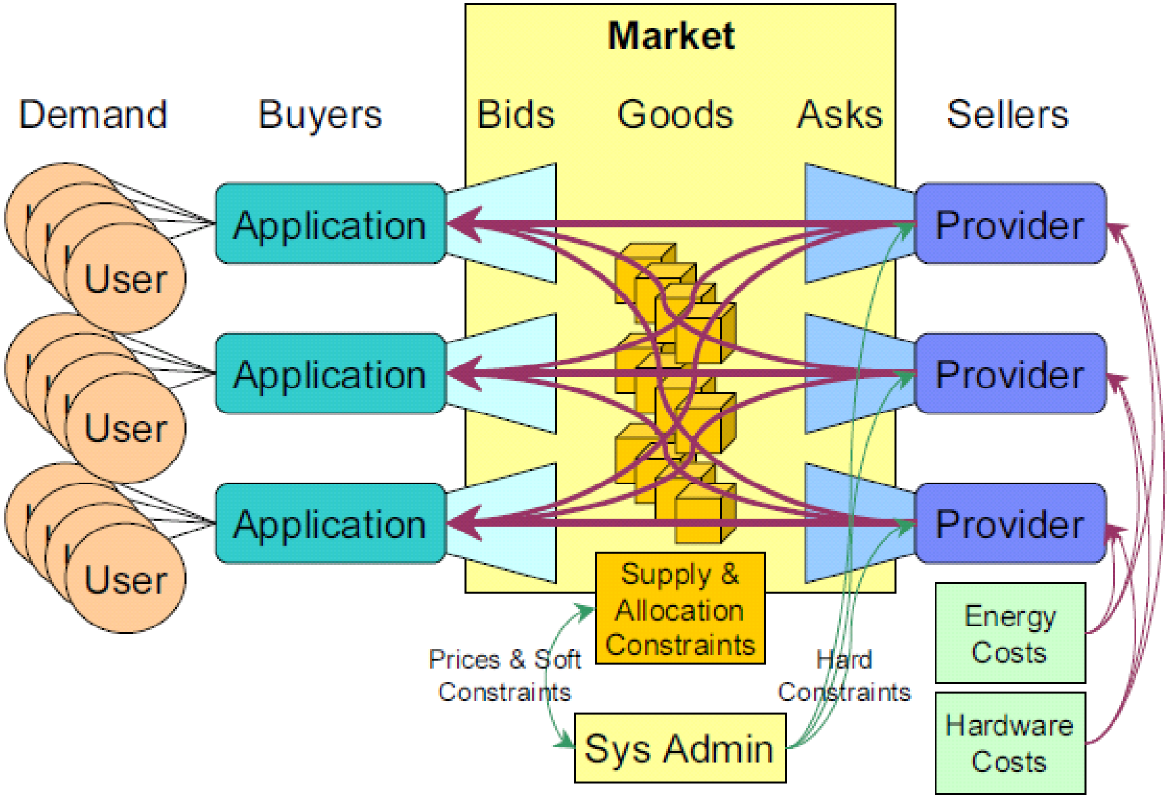


Figure 1: The Data Center Market Model

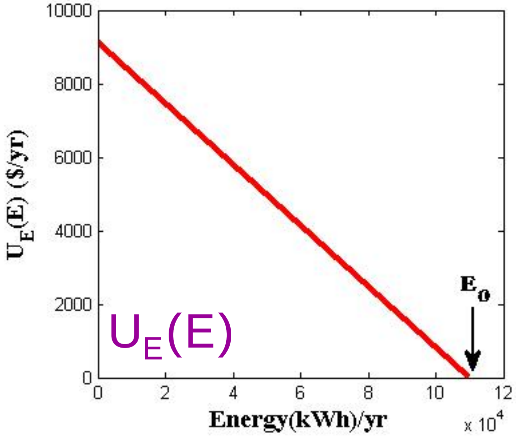
Lubin, Kephart, Das and Parkes. *Expressive Power-Based Resource Allocation for Data Centers*. **IJCAI 2009.**
(Exploring market-based resource allocation for data centers.)

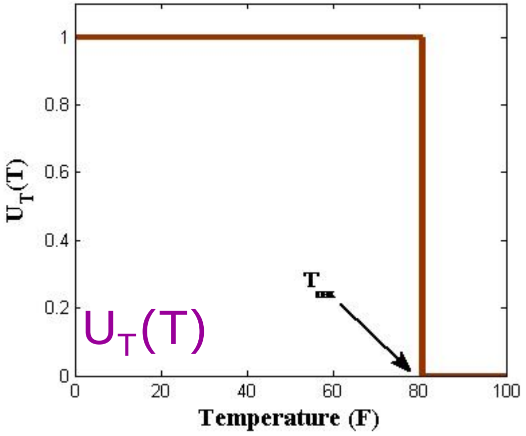
Backup

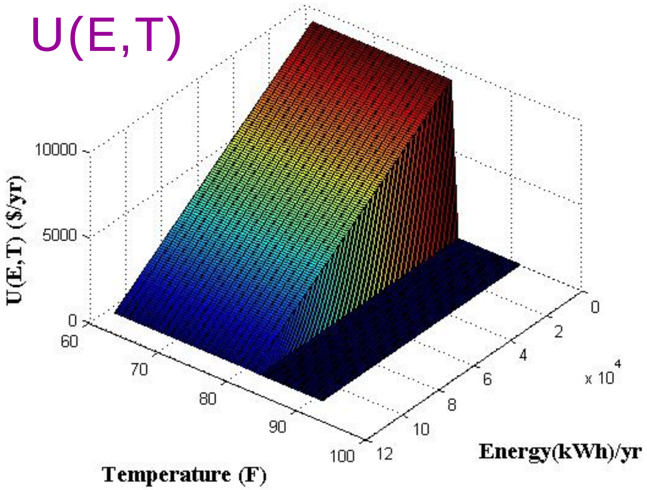
Multiplicative Utility Functions

- Administrator wishes to minimize overall energy consumption subject to a constraint on temperature
  - E.g. $T(\mathbf{x}) < T_{max}$ at all positions **x**.

- Consider the *multiplicative* form: $U(E,T) = U_E(E)\ U_T(T(\mathbf{x}))$
  - Energy utility $U_E(E) = \pi\ (E - E_0)$, where π is \$/(kW-year)
  - Temperature utility $U_T(T(\mathbf{x}))$ is a dimensionless step function, with the entire temperature distribution $T(\mathbf{x})$ as its argument







Dimensionless Temperature Utility Function

Practical Considerations

- Let’s think about $U_T(T(\mathbf{x}))$ a little more
  - $U = 1$ if $T(\mathbf{x}) < T_{max}$ for all $\mathbf{x}$
  - $U = 0$ otherwise
- But we *can’t* measure $T(\mathbf{x})$ for *all* **x**
- One solution: just consider a finite set of measurements $\{T(x_i)\}$
  - Set could be readings from *all* temperature sensors
  - Or just the reading from a single representative temperature sensor i
  - Or just the maximum temperature in a region of interest (maybe entire DC)
- Example if we use *many* or *all* temperature sensors:
  - We can represent $U_T(T(\mathbf{x}))$ as the product of scalar step functions
  - $U_T(T(\mathbf{x})) = \prod_i U_{T,i}(T_i)$
  - $U_{T,i}(T_i) = 1$ if $T_i < T_{max}$; 0 otherwise

Sanity check

- Case A: $T_i > T_{max}$ for all 10 sensors
  - $U_T(T(\mathbf{x})) = 0 * 0 * 0 * 0 * 0 * 0 * 0 * 0 * 0 * 0 = 0$
- Case B: $T_i > T_{max}$ for just sensor #10
  - $U_T(T(\mathbf{x})) = 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 1 * 0 = 0$

- Since $U_T(T(\mathbf{x}))$ is 0 in Case A and B, utility U(E, T) = 0 in both

- Yet most admins would prefer Case B to Case A!

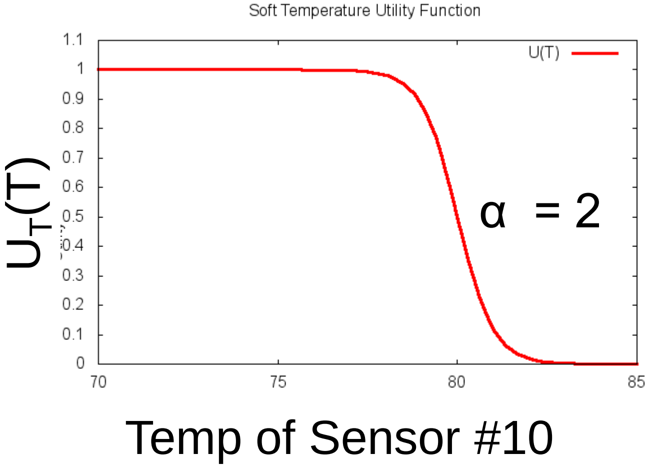
- How could we modify the utility function to prefer B over A?

- One solution: *soften* the temperature constraint …

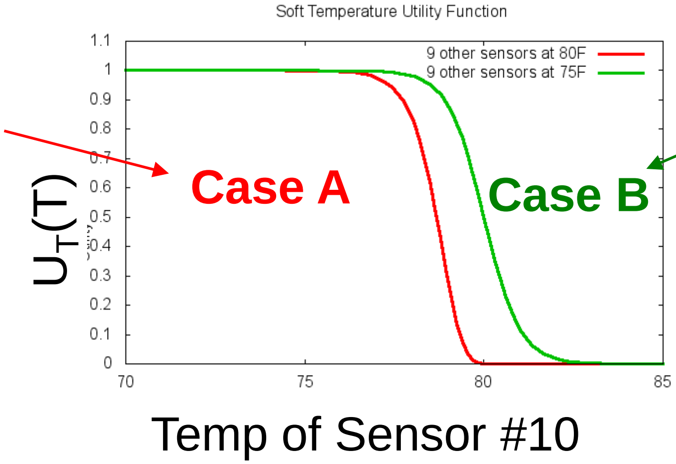
Modifying $U_T(T)$ to express a soft constraint

Soften the scalar step function $U_{T,i}(T_i)$

$$U_{T,i}(T_i) = 1/(1 + e^{-(\alpha(T_{max} - T_i))})$$



All 10 sensors at same temperature



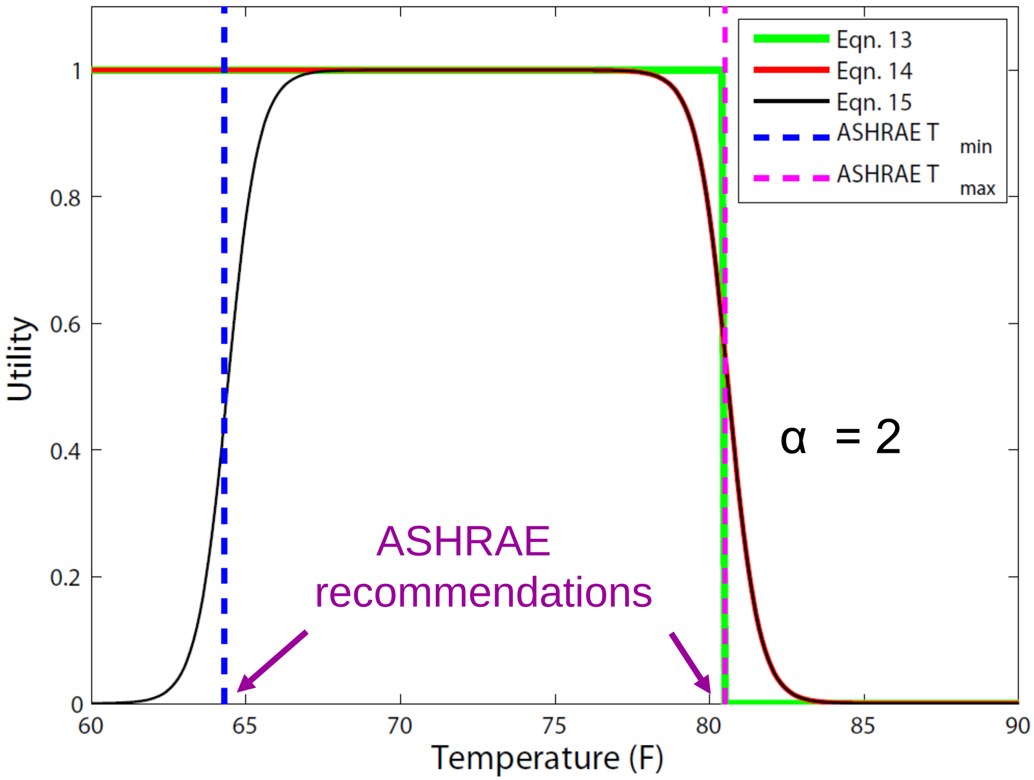
9 sensors at 75F; vary temp of sensor #10

***Soft* step function favors Case B over Case A**

Further variations on $U_T(T)$

ASHRAE specifies both a minimum and a maximum temperature

We can represent the scalar temperature utility as a two-sided soft step function



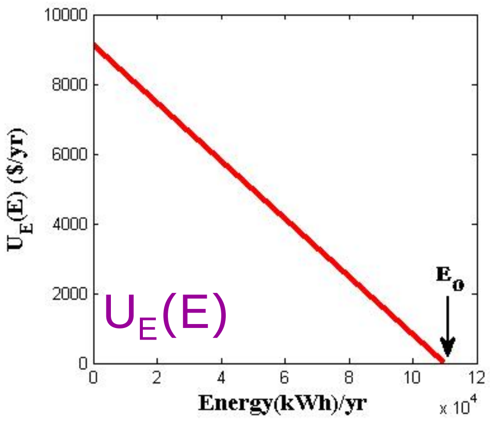
α = 2

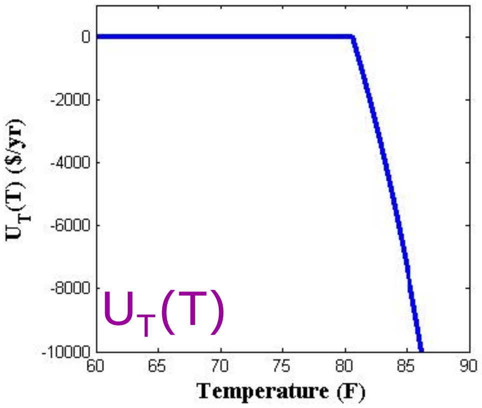
ASHRAE recommendations

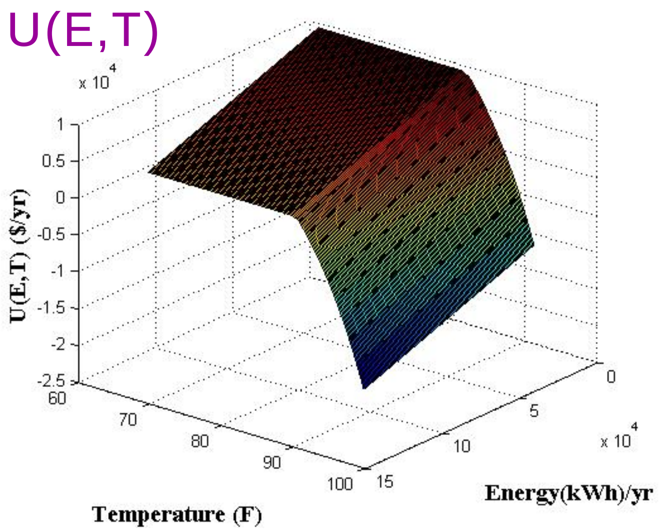
Additive Utility Functions

- Administrator explicitly considers economic costs of energy consumption and temperature-induced equipment lifetime reduction

- This suggests an alternative *additive* form: $U(E,T) = U_E(E) + U_T(T(\mathbf{x}))$
  - Energy utility $U_E(E) = \pi\ (E - E_0)$
  - Temperature utility $U_T(T(\mathbf{x}))$ must now have same dimension: cost/yr





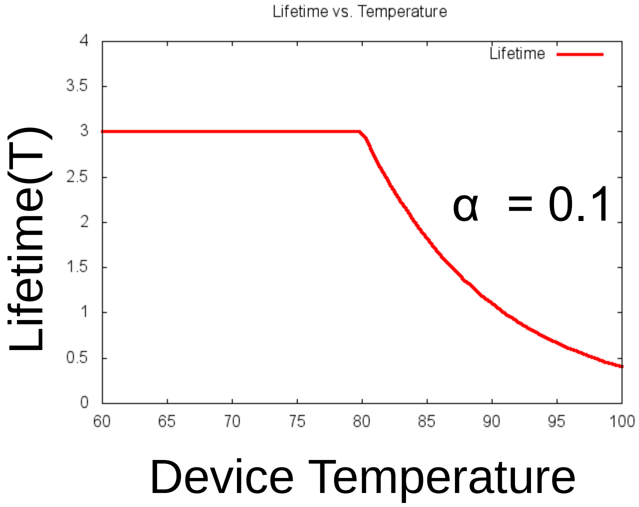


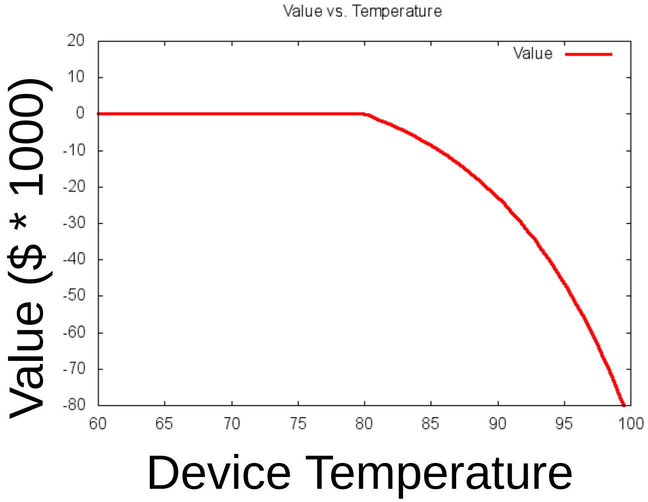
Cost-based Temperature Utility Function

Practical Considerations

- Somehow $U_T(T(\mathbf{x}))$ must capture the cost of running equipment at **x** at temperature $T(\mathbf{x})$

- Cost of device i per year is $C_i/L(T)$
  - C_i = purchase cost
  - $L(T)$ = lifetime if run consistently at temperature T
  - Inverse width α hard to ascertain from published data – widely different reports
    - Seagate drive lifetime reduced 4x for 35C increase in T
    - Google reports little degradation until 40C

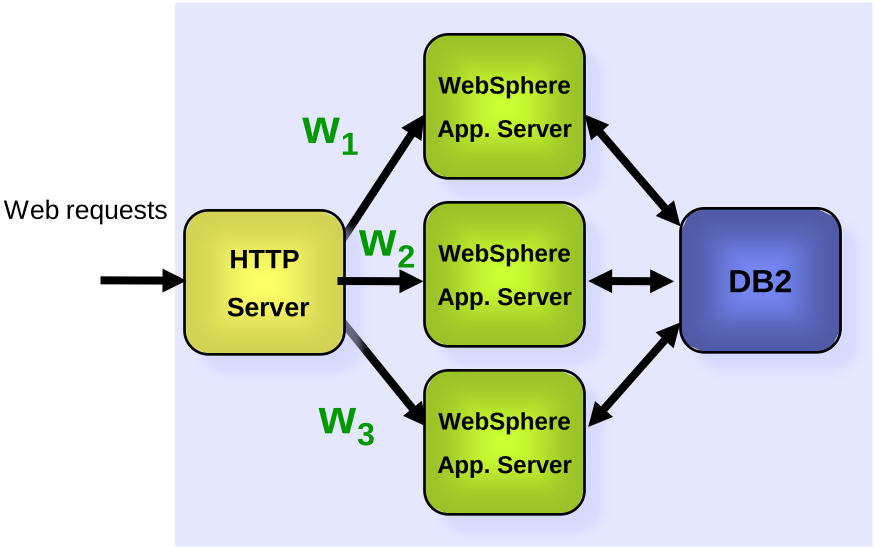




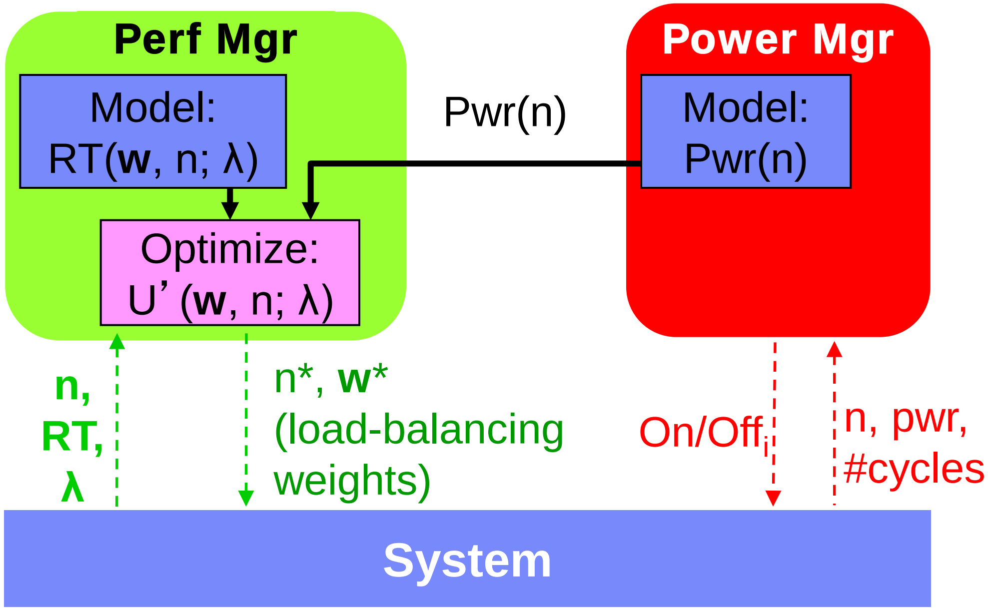
- $U_T(\{T_i\}) = \sum_i C_i \,(1/L_0 - 1/L(T_i))$

Power-aware load balancing

Goal: Save power by routing web traffic to minimal number of app servers w/o sacrificing performance too much.



Maximize U(**RT**, pwr)

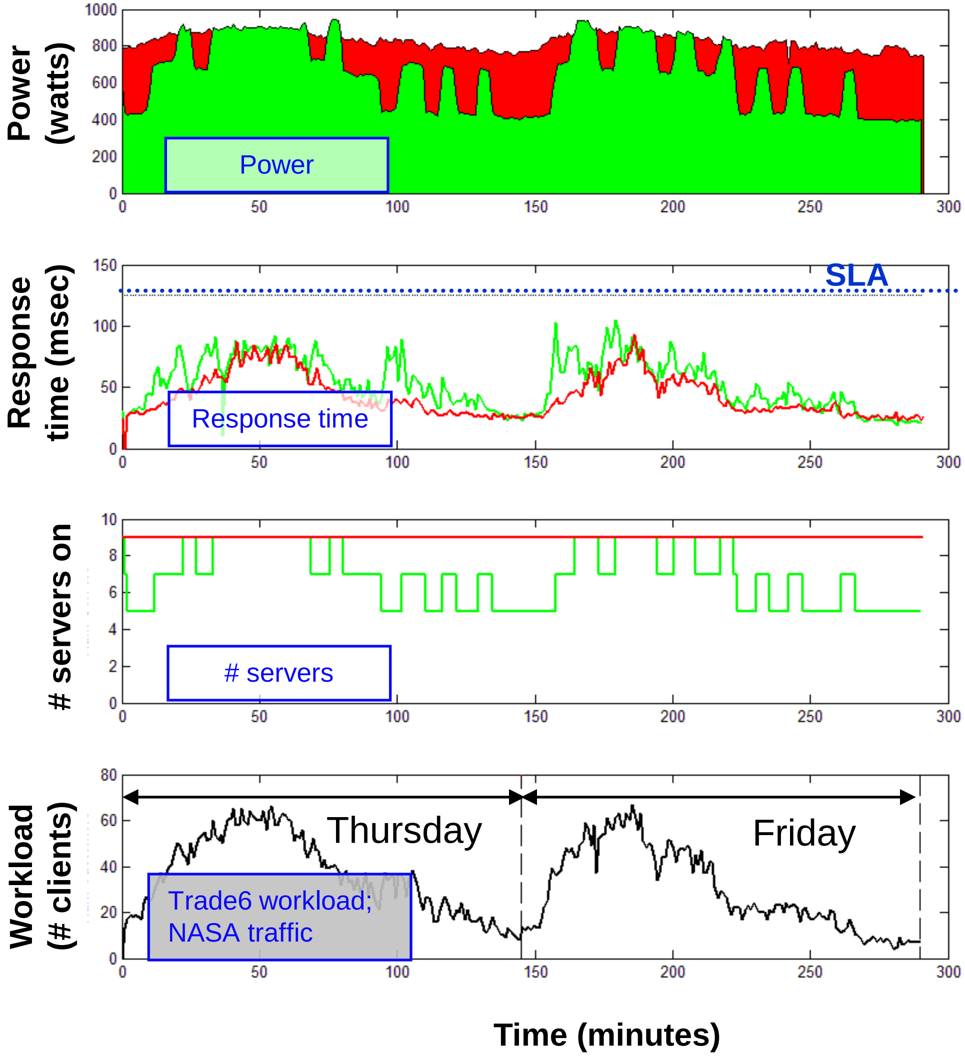


Das, Kephart, Lefurgy, Tesauro, Levine, Chan *Autonomic Multi-agent Management of Power and Performance in Data Centers*, **AAMAS 2008**

Experimental Results

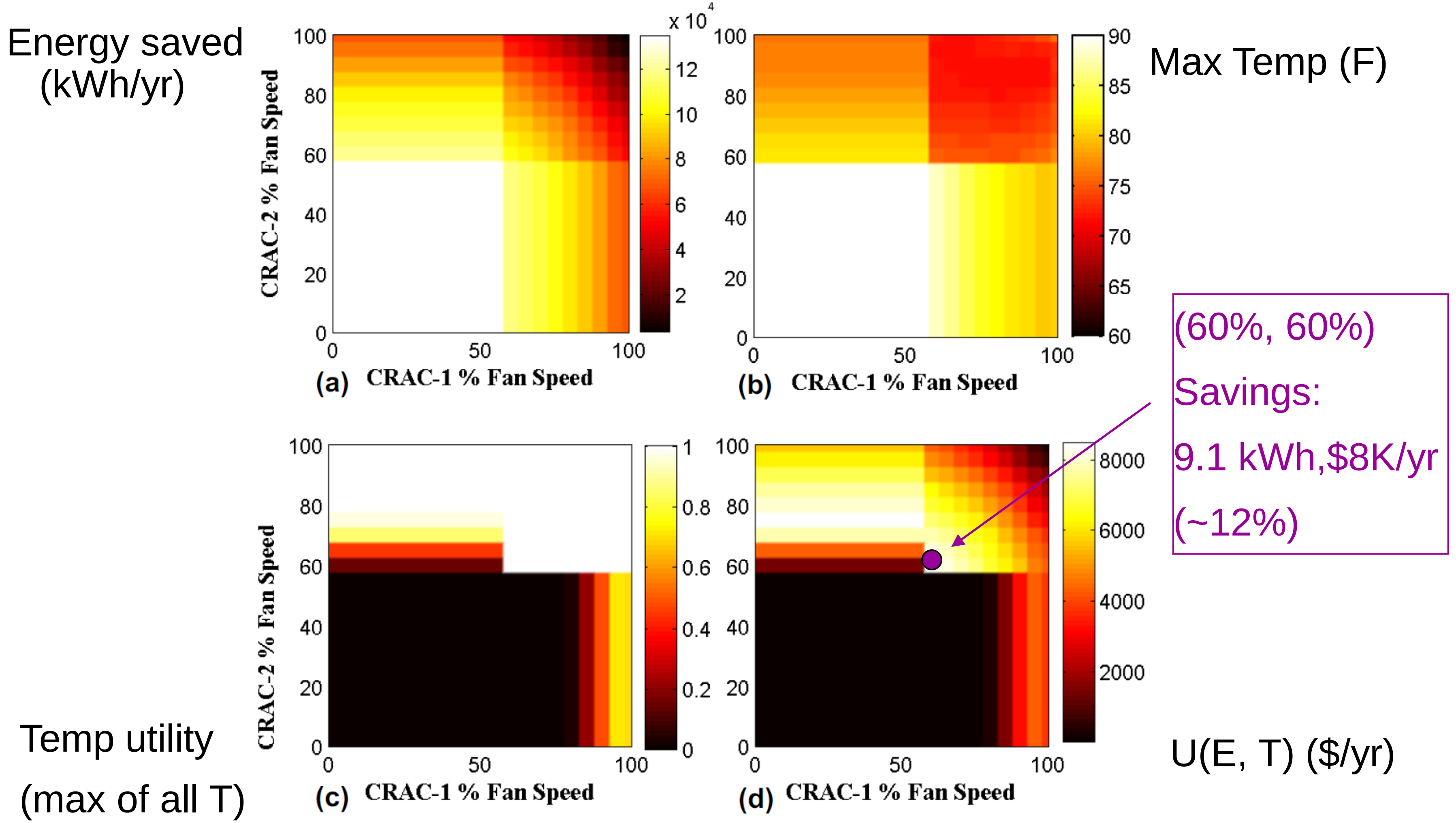
Avg Power Savings = 27%
No SLA violation

- **Elicit** utility function
  - U(RT) = 1/0 if SLA met/unmet
  - $U(RT, Pwr) = U(RT) – \varepsilon\ Pwr$
- **Model** (offline experiments)
  - $RT(n; \lambda)$, $Pwr(n; \lambda)$
- **Transform**
  - $U'(n; \lambda) = U(RT(n; \lambda), Pwr(n; \lambda))$
- **Optimize** (pre-computed policy)
  - $n^*(\lambda) = \text{argmax}_n\ U'(n; \lambda)$
- A few extra tweaks
  - Use forecasted λ to compute $n^*(\lambda)$
  - Add extra to n* a bit to account for latencies (several minutes)
  - Heuristics to ensure that we don't turn servers on and off too often



Multiplicative Utility-Function-Driven Cooling

T_{max} = 80.6F, α = 2.0



(60%, 60%)

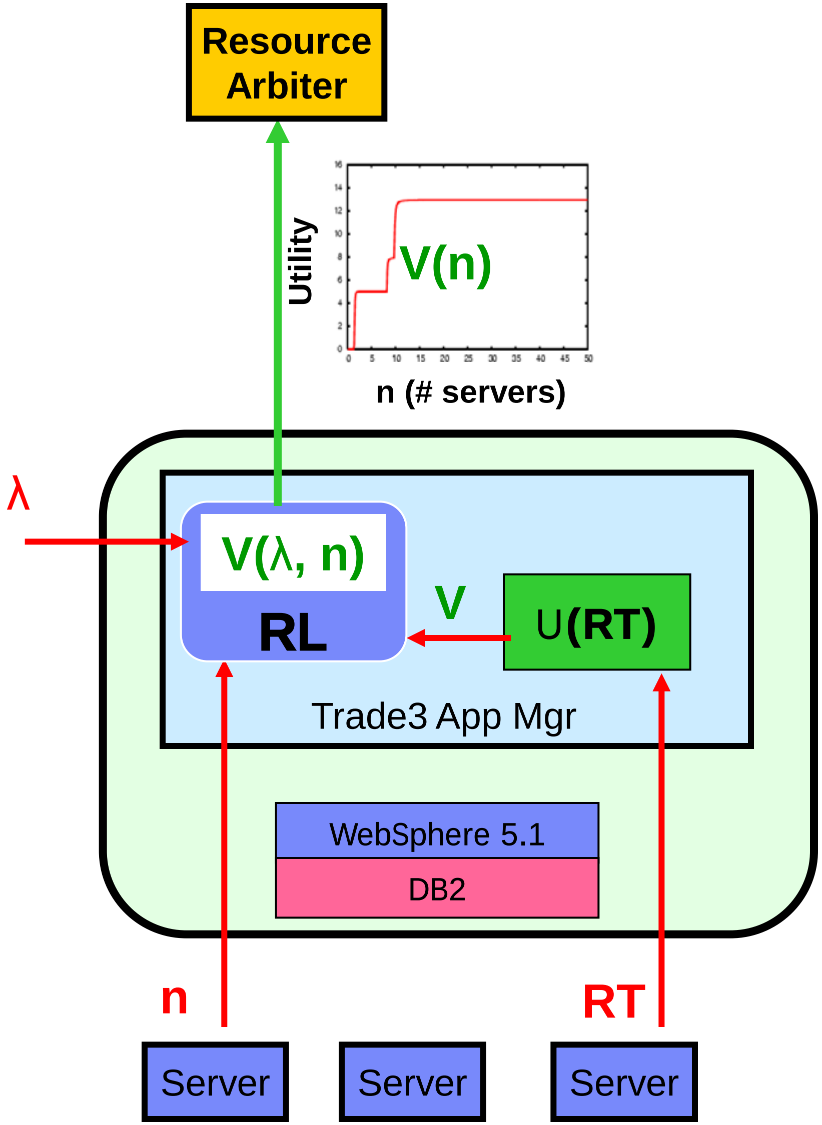
Savings:

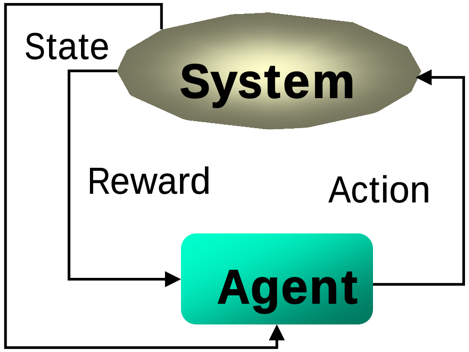
9.1 kWh,$8K/yr

(~12%)

Alternative Approach: Machine Learning

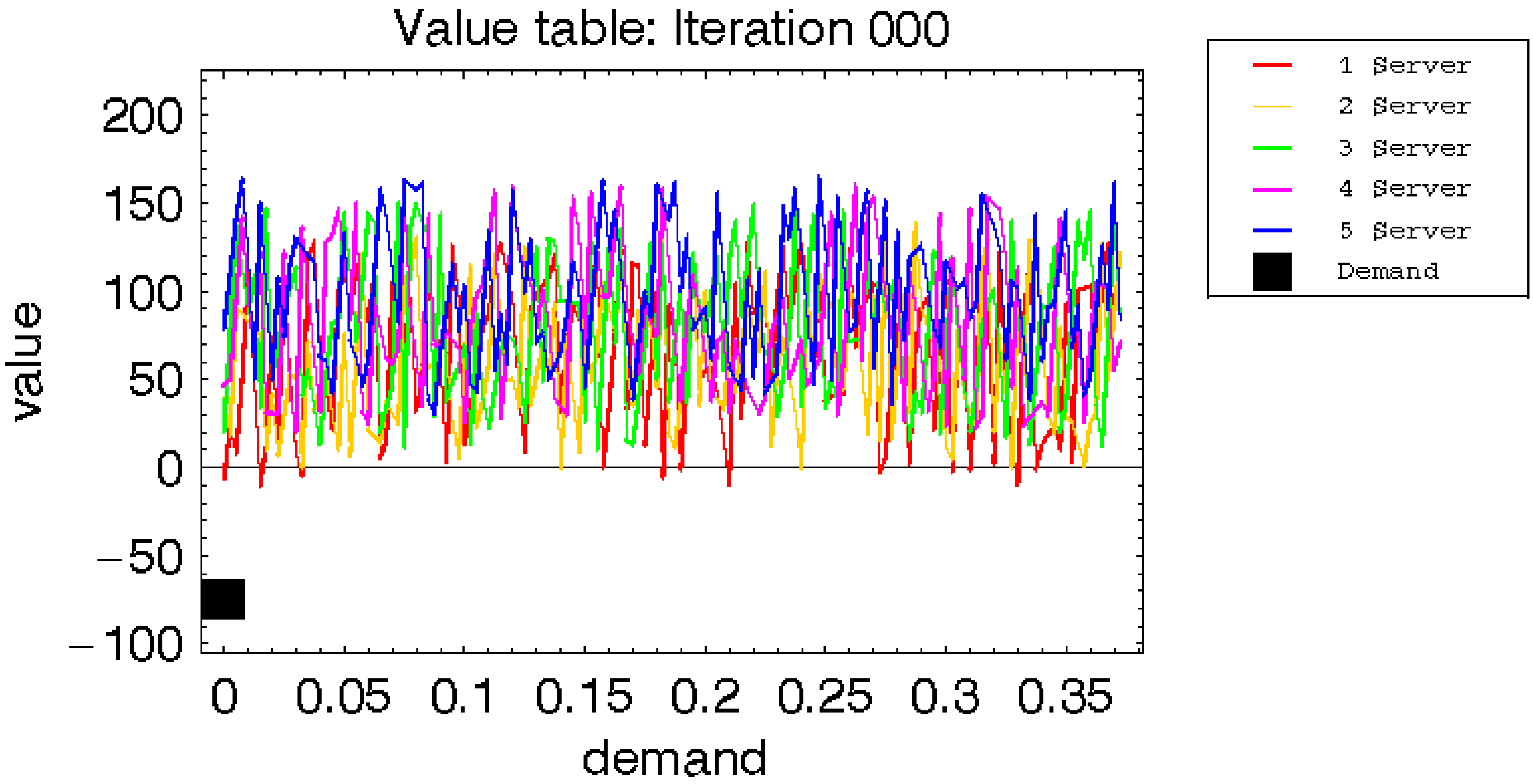
Tesauro et al., AAAI 2005





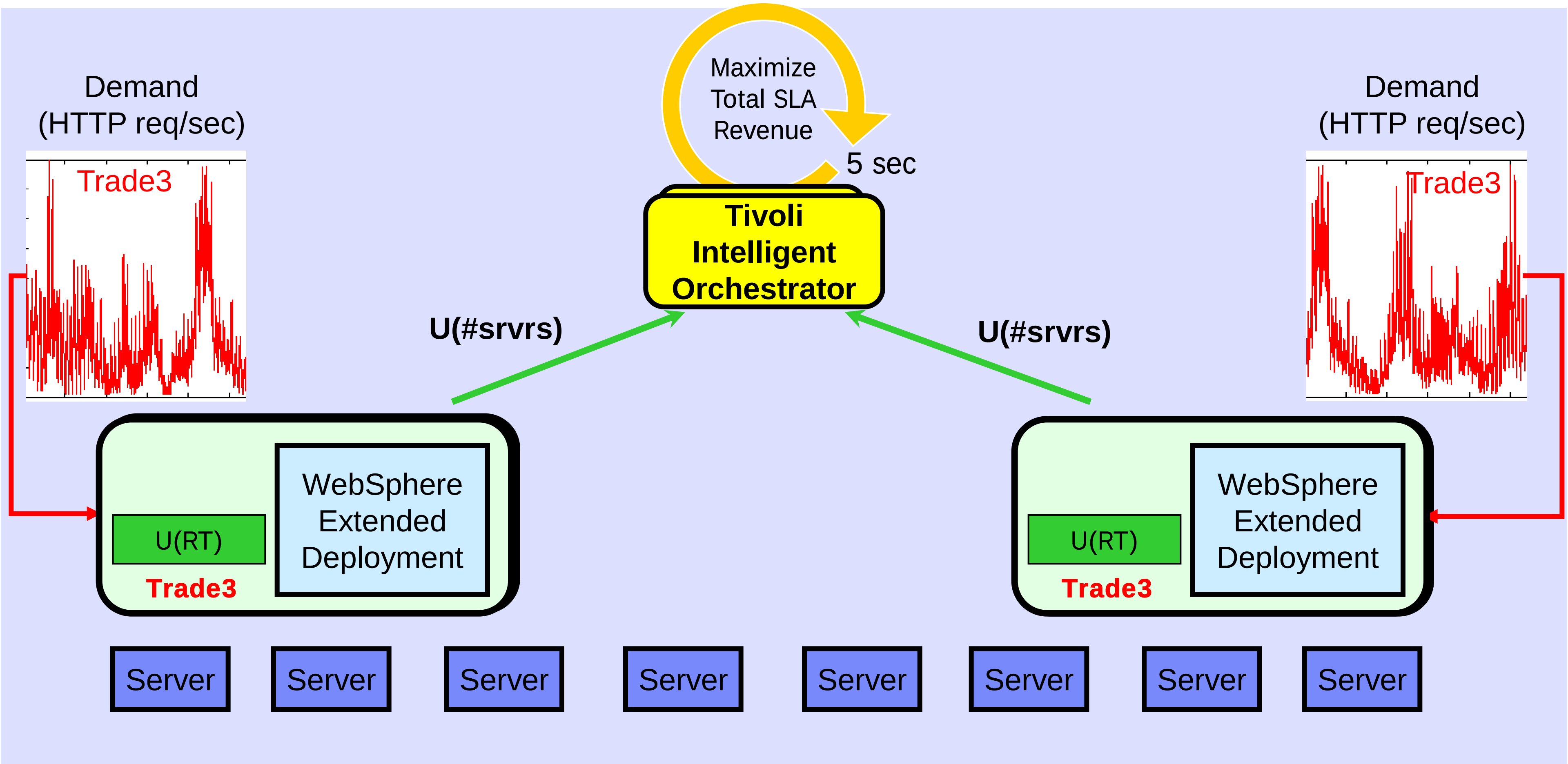
- App Mgr can use reinforcement learning (**RL**) to compute external resource utility
  - State = λ *demand*
  - Action = n *# servers*
  - Reward = V(RT) *SLA payment*
- It learns *long-range* value function V(state, action) = V(λ, n)
- It reports V(n) for current or predicted value of λ

Does Reinforcement Learning work?

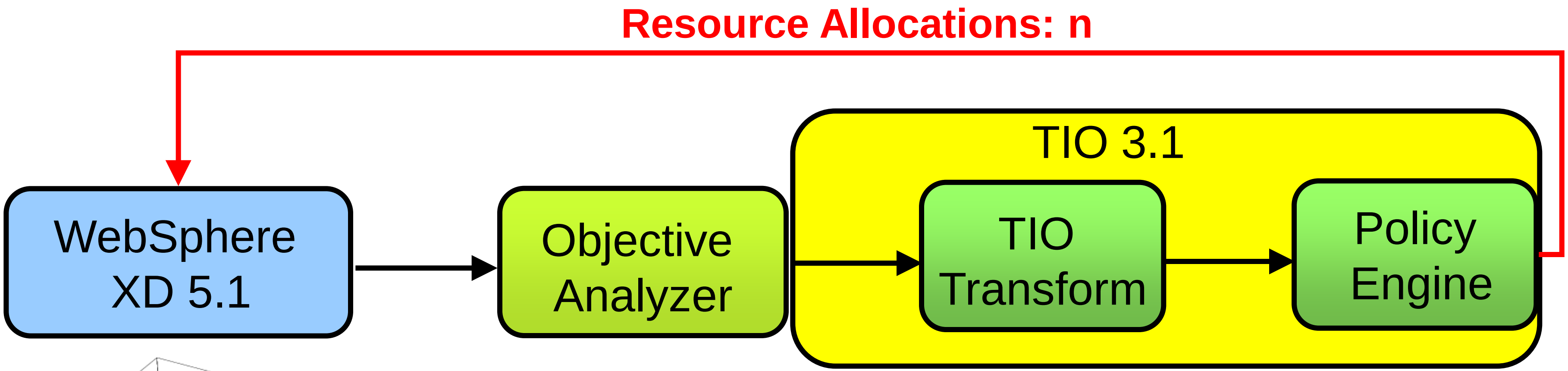


Animation

Commercializing Unity



Utility-based Interactions between WXD and TIO: Step 1



Resource Allocations: n

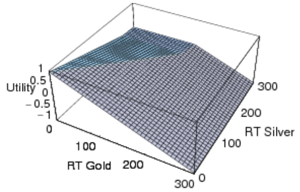
TIO 3.1

WebSphere XD 5.1

Objective Analyzer

TIO Transform

Policy Engine

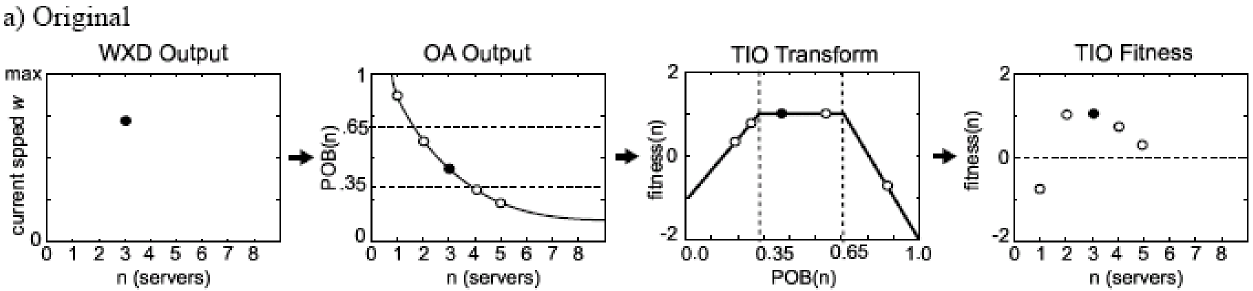


Resource Needs(speed)

Resource Needs in **PoB(n)**

Resource Needs in **Fitness(n)**

Utility(current n)



a) Original

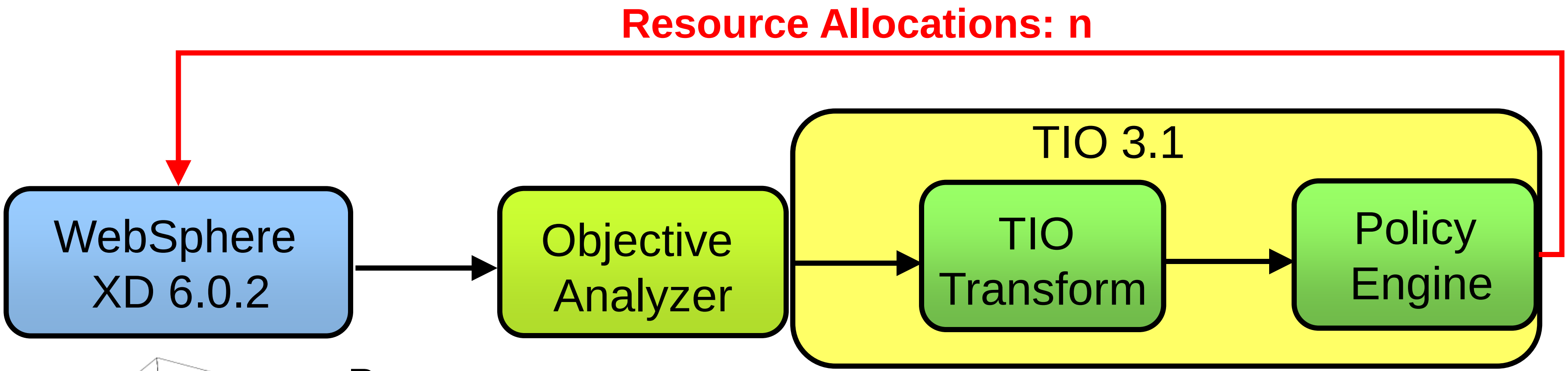
Original

WXD 5.1

TIO 3.1

- TIO cannot make well-founded resource allocation decisions
  - WS XD can’t articulate its needs to TIO
  - PoB not commensurate with utility

Utility-based Interactions between WXD and TIO: Step 2



Resource Allocations: n

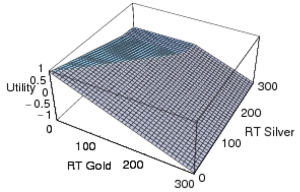
WebSphere XD 6.0.2

Objective Analyzer

TIO 3.1

TIO Transform

Policy Engine



Resource Utility(n)

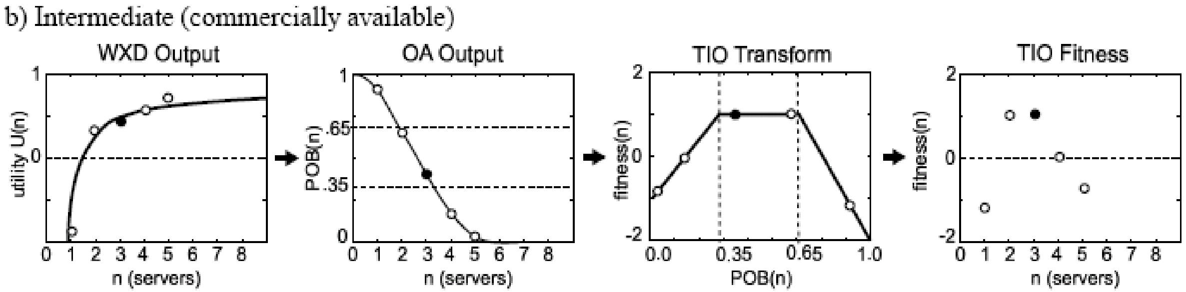
PoB(n)

Fitness(n)

Utility(current n)

Intermediate

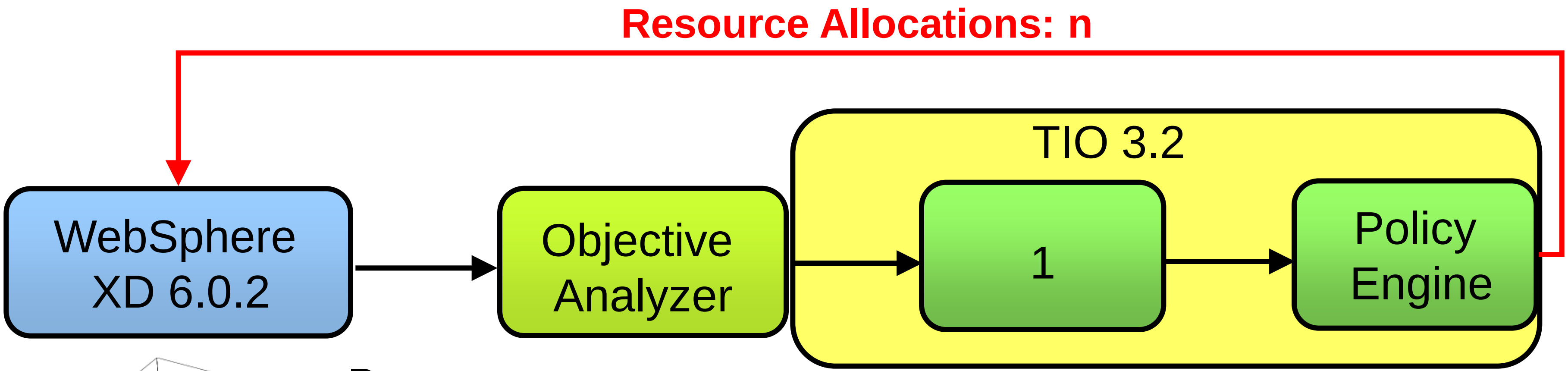
WXD 6.0.2



TIO 3.1

- WS XD research team added ResourceUtil interface of WXD
- We developed a good heuristic for converting ResourceUtil to PoB in Objective Analyzer
  - Interpolate discrete set of ResourceUtil points and map to PoB
  - This PoB better reflects WS XD’s needs

Utility-based Interactions between WXD and TIO: Step 3



Resource Allocations: n

WebSphere XD 6.0.2

Objective Analyzer

TIO 3.2

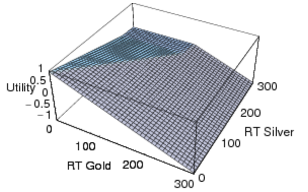
1

Policy Engine

Resource Utility(n)

Resource Utility(n)

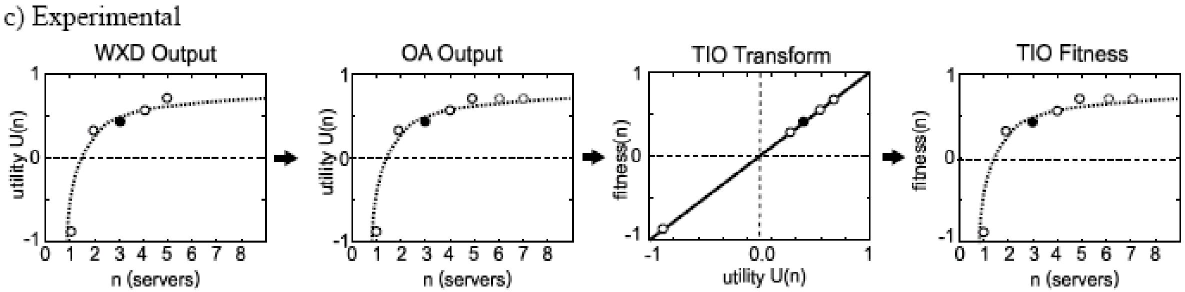
Resource Utility(n)



Utility(current n)

New

Modified WXD 6.0.2



Modified TIO 3.1

- We modified TIO to use ResourceUtil(n) directly instead of PoB(n)
- Most mathematically principled basis for TIO allocation decisions
- It enables TIO to be in perfect synch with the goals defined by WS XD
- Basic scheme can work, not just for XD, but for any other entity that may be requesting resource, provided that it can estimate its own utilities

Das et al., ICAC 2006

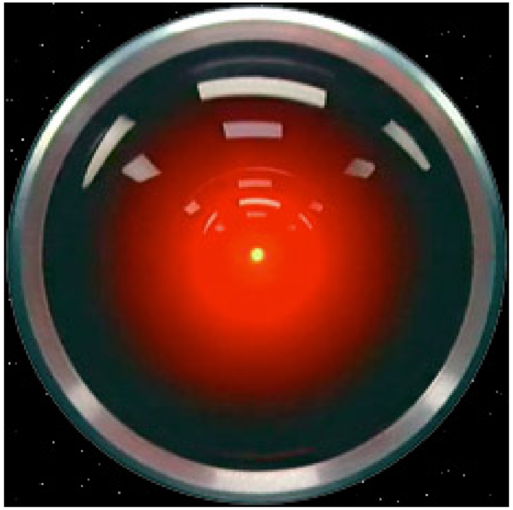
Commercializing *Unity*

- Barriers are not just technical in nature
- Strong product line legacies must be respected; otherwise
  - Difficult for the vendor
  - Risk alienating existing customer base
- **Solution**: Infuse agency/autonomicity gradually into existing products
  - Demonstrate value incrementally at each step
- We worked with colleagues at IBM Research and IBM Software Group to implement the *Unity* ideas in two commercial products:
  - Application Manager: **IBM WebSphere Extended Deployment (WXD)**
  - Resource Arbiter: **IBM Tivoli Intelligent Orchestrator (TIO)**

Visions of Autonomic Computing

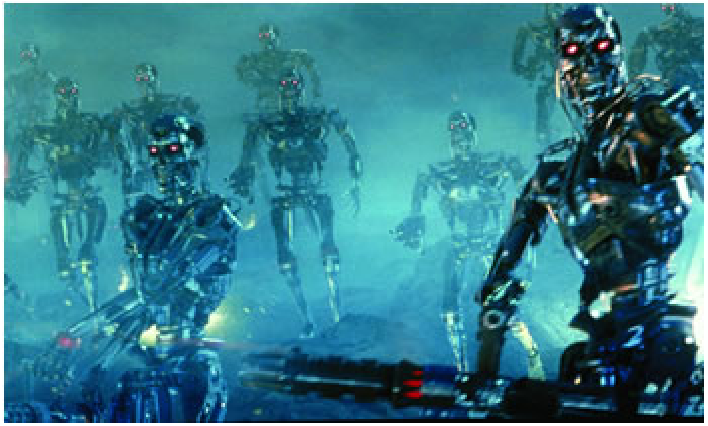
Star Trek: The Next Generation

Hal 9000, *2001*



Machines will take over all management tasks, rendering humans superfluous

Terminator



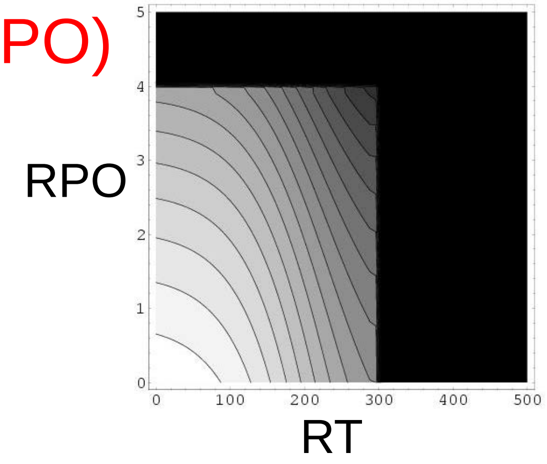
Machines will free people to manage systems at a higher level

Wrong!

Right!

Finding the optimal control parameters

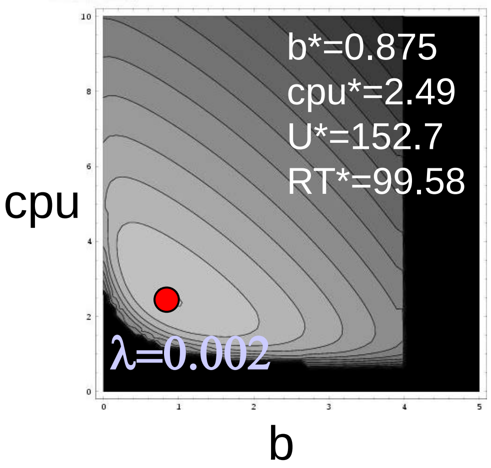
U(RT, RPO)



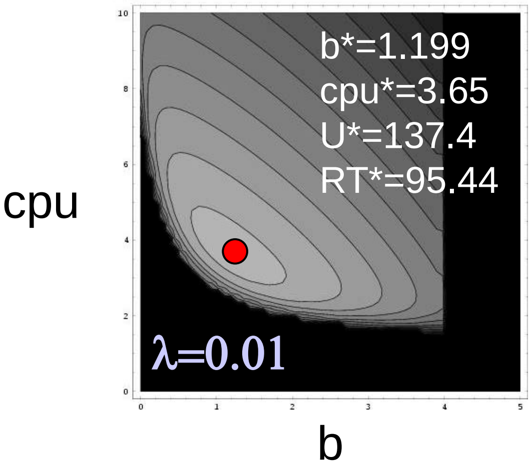
Even if service-level utility remains fixed, resource-level utility depends upon environment.

Thus system responds to environmental changes.

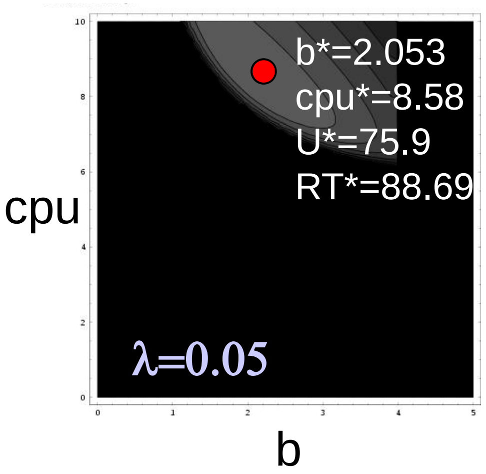
U’ (cpu, b; λ)



b*=0.875
cpu*=2.49
U*=152.7
RT*=99.58



b*=1.199
cpu*=3.65
U*=137.4
RT*=95.44



b*=2.053
cpu*=8.58
U*=75.9
RT*=88.69

Multi-agent management of performance and power

- We have explored using utility functions to manage performance and power objectives and tradeoffs in multiple scenarios
- Two separate agents: Performance and Power
- Various control parameters, various coordination and communication mechanisms
  - Power controls: clock frequency & voltage, sleep modes, …
  - Performance controls: routing weights, # servers, VM placement …
  - Coordination: unilateral, bilateral, mediated, …
- Examples
  - Energy-aware load balancing
  - Energy-aware server consolidation
  - Optimal power capping