Self-Adaptive Systems

David Garlan
Carnegie Mellon University

Portland State University

Portland, Oregon

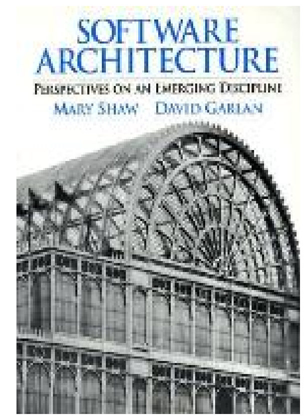
January 2017

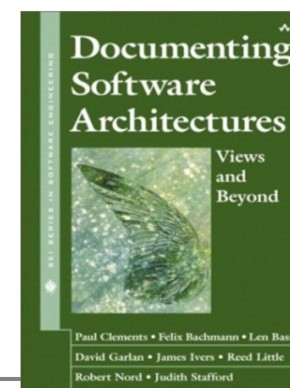
institute for SOFTWARE RESEARCH

Carnegie Mellon

About Me

- 1987 PhD at CMU
- 3 years at Tektronix developing a software product line architecture
- 1990 joined faculty at CMU
  - Began collaboration with Mary Shaw
  - Became involved in the Master in Software Engineering program, now Director
- 1992 taught first course in Architectures for Software Systems
- 1996 published book on Software Architecture with Mary Shaw
- 2003 published book on Documenting Software Architecture. Second edition 2011.





© David Garlan 2017

Talk Outline

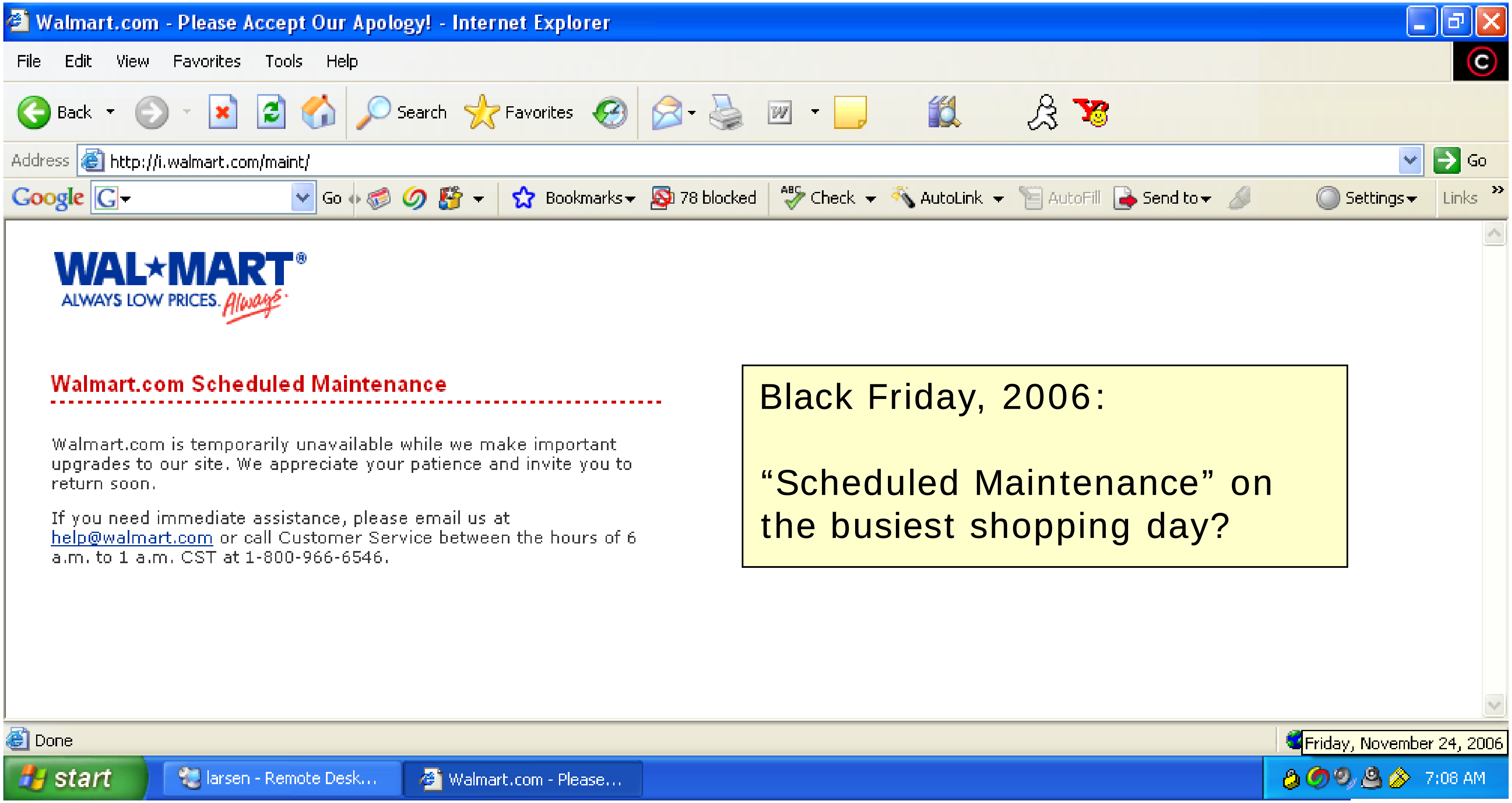
- The vision of Self-Healing Systems
  - The problem and its solution
- Architecture-based self-adaptation
  - Rainbow and Stitch
- Some current research directions
  - Applications to security
  - Run-time diagnosis and fault localization
  - Human-in-the-loop adaptation
  - Other research areas

The Problem

- An important requirement for modern software-based systems

Maintain high-availability and optimal performance even in the presence of
- changes in environment
- system faults
- attacks
- changes in user needs and context

Websites Fail to Adapt



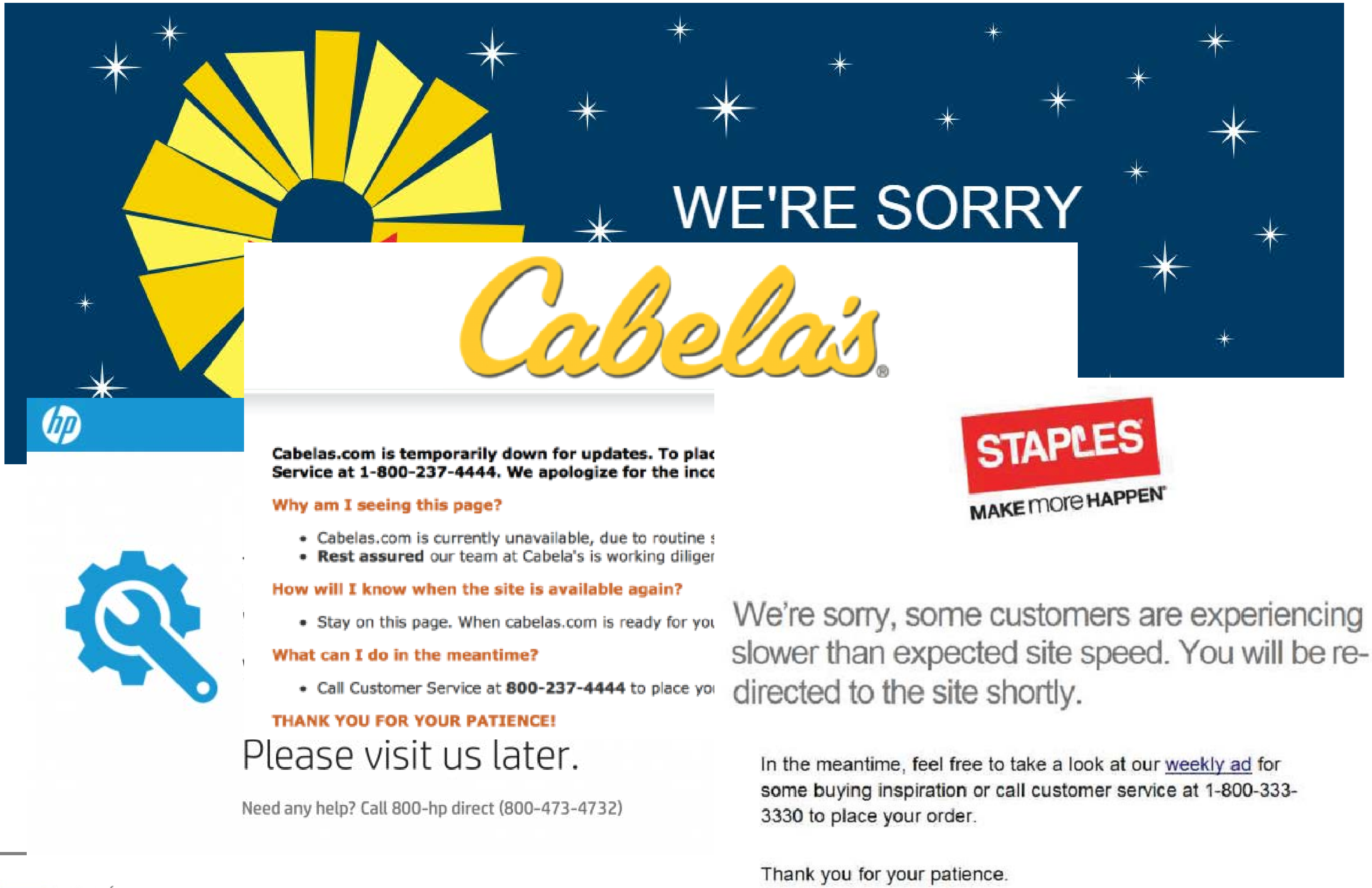
Black Friday, 2006:

“Scheduled Maintenance” on the busiest shopping day?

Amazon.com disrupted due to Xbox 360 the day before

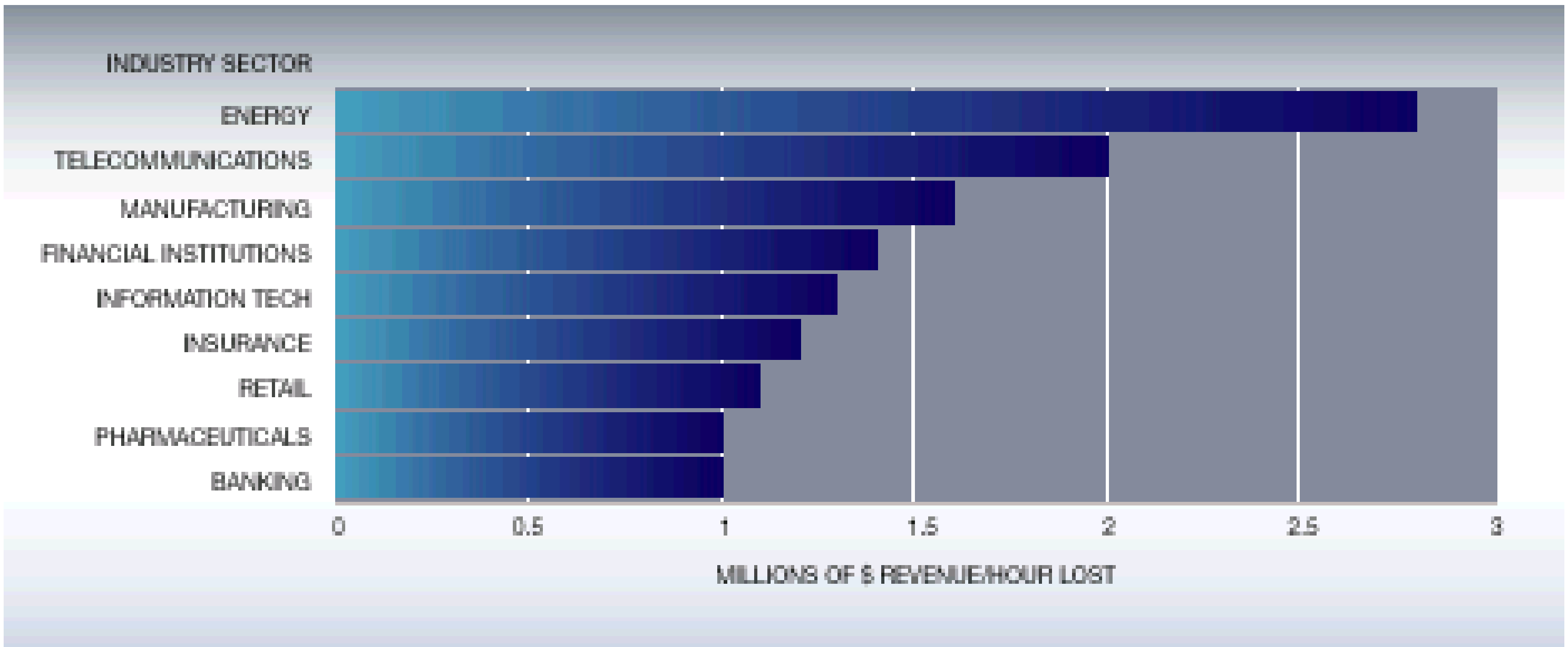
© David Garlan 2017

Black Friday 2014



Cost of Downtime

- Average hourly impact of downtime by industry sector



Data from *IT Performance Engineering and Measurement Strategies: Quantifying Performance Loss*, Meta Group, Stamford, CT (October 2000).

© David Garlan 2017

How is this addressed today?

- Technique 1: Build resilience directly into application code
  - Use exceptions, timeouts, and other low-level programming mechanisms
- Unfortunately, this approach is not good for
  - Locating the cause of a problem
  - Detecting “softer” system anomalies
  - Anticipating future problems
  - Maintainability: hard to add and modify adaptation policies and mechanisms
  - Handling changing objectives
  - Legacy systems: hard to retrofit later

How is this addressed today?

- Technique 2: Human oversight
  - Operators, system administrators, users
  - Global oversight, intelligent response
- Unfortunately, this approach is
  - Costly
  - Error-prone
  - Slow

© David Garlan 2017

Cost of Human Oversight

- Estimated 1/3-1/2 total IT budget to prevent or recover from crash
- “For every dollar to purchase storage, you spend $9 to have someone manage it”—Nick Tabellion
- Administrative cost: 60-75% overall cost of database ownership
- 40% of root causes of computer system outage is attributable to operator error

- Washington Post, October 17, 2014 article: *"Stop worrying about mastermind hackers. Start worrying about the IT guy."*

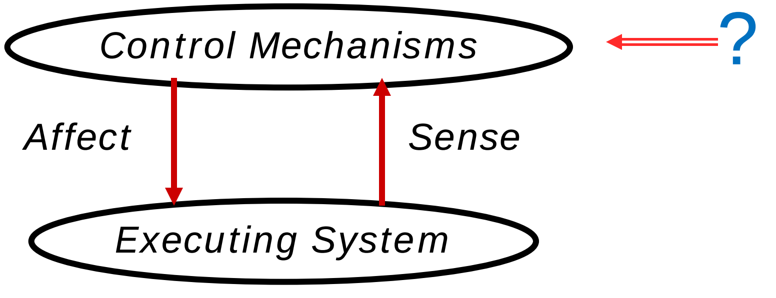
  "the weakest link often involves the inherent fallibility of humans. ... even the most skilled system administrators struggle to keep every computer at large institutions running smoothly, with the proper software updates, security patches and configurations."

A New Approach

- Goal: systems automatically and optimally adapt to handle
  - faults and attacks
  - variable resources and environments
  - changes in user needs

But how?

Answer: Move from open-loop to closed-loop systems



© David Garlan 2017

Example: Google File System

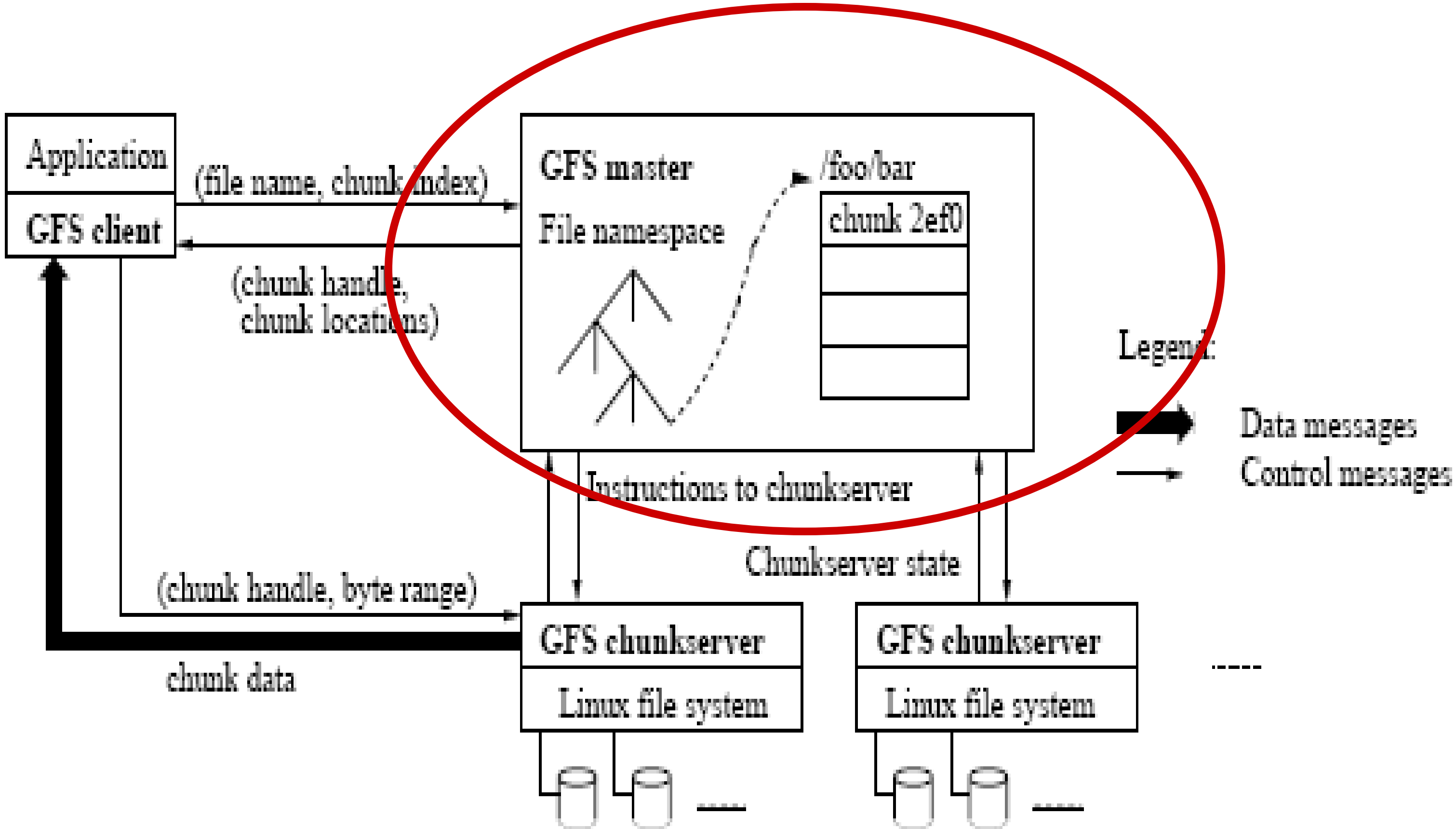


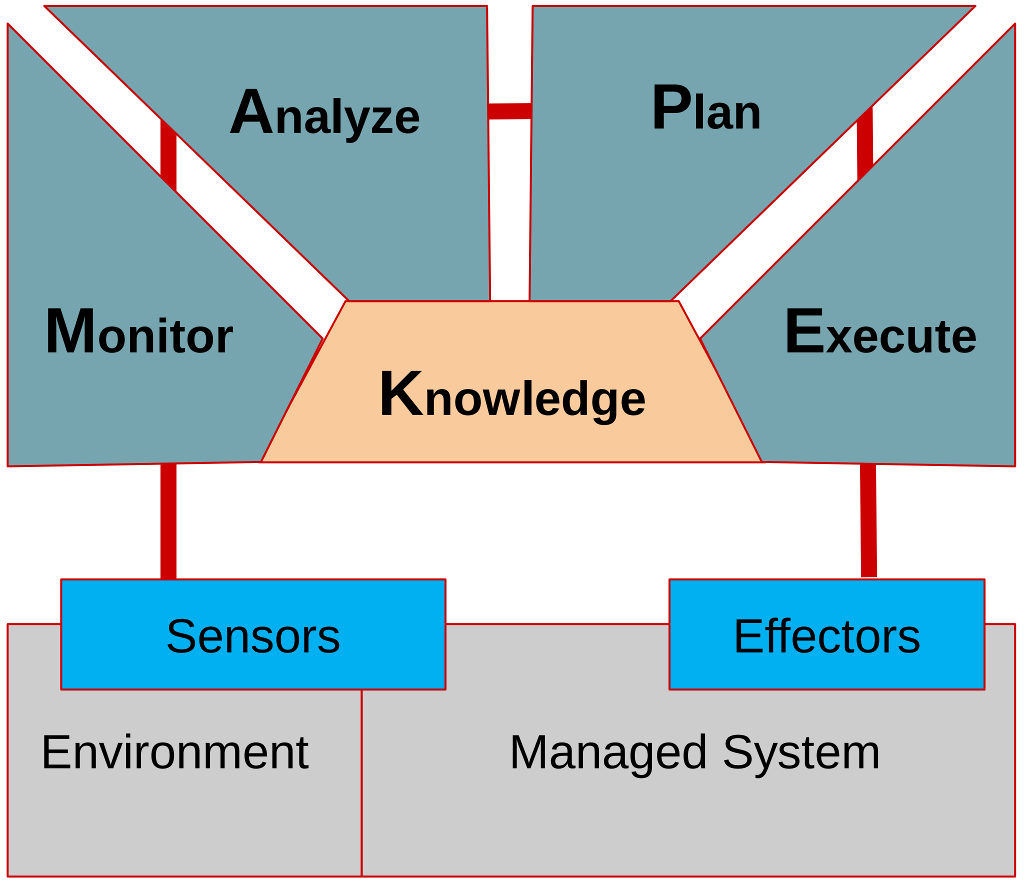
Figure 1: GFS Architecture

Source: "The Google File System" Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. SOSP 2003.

The Challenge

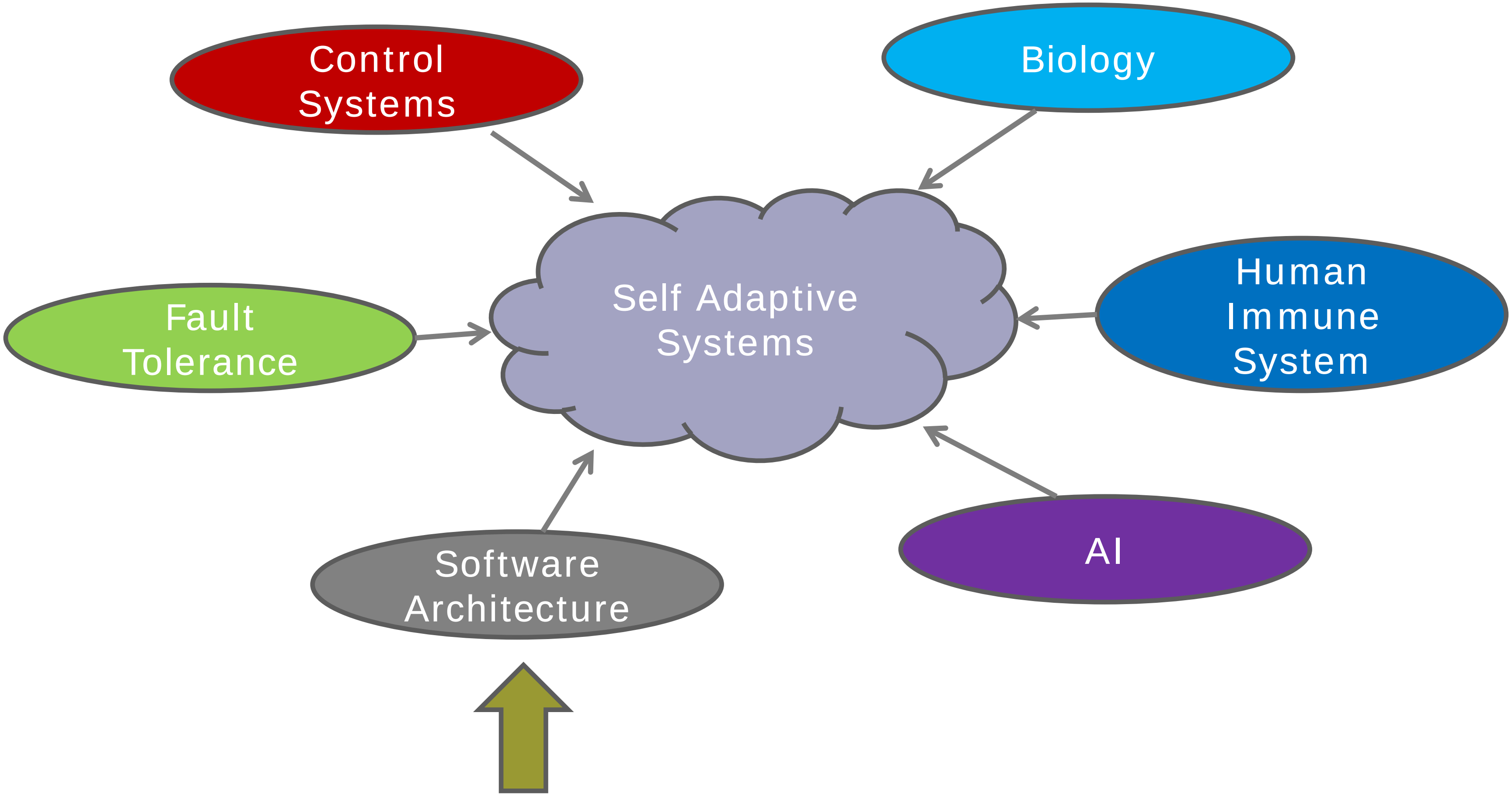
- Provide effective engineering support for making systems self-adaptive
  - Applicable to legacy systems
  - Low development cost
  - Domain-specific adaptations
  - Multiple quality dimensions
  - Easily change/augment adaptation policies and mechanisms
  - Reason about the effects of self-adaptation actions and strategies

IBM MAPE-K



J.O. Kephart, and D.M. Chess. "The vision of autonomic computing." *Computer* 36, no. 1, 2003

Related Disciplines



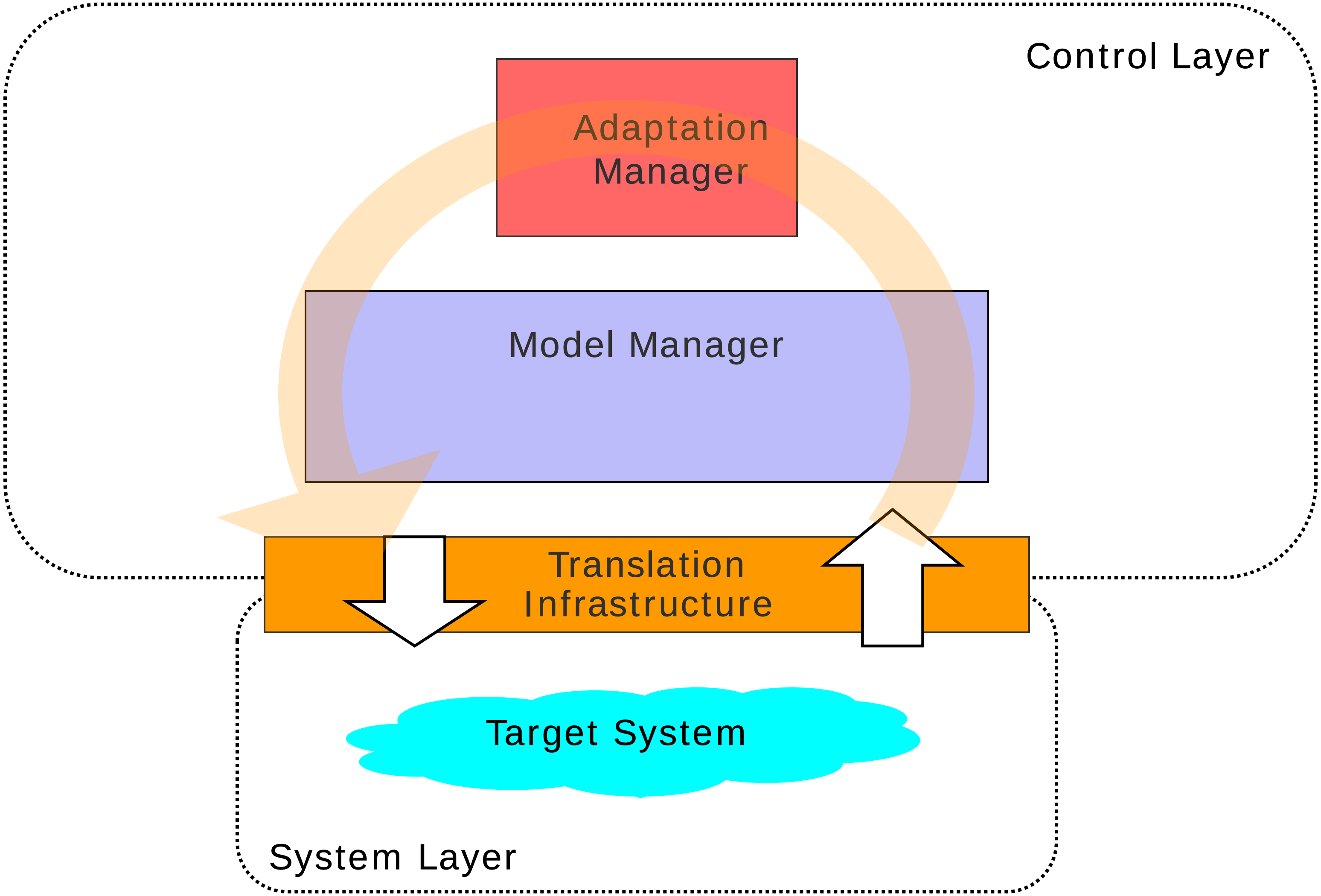
© David Garlan 2017

Rainbow Approach

- A framework that
  - Allows one to add a control layer to existing systems
  - Uses architecture models to detect problems and reason about repair
  - Can be tailored to specific domains
  - Separates concerns through multiple extension points: probes, actuators, models, fault detection, repair
- The framework is instantiated for specific domains, systems, mechanisms, and policies

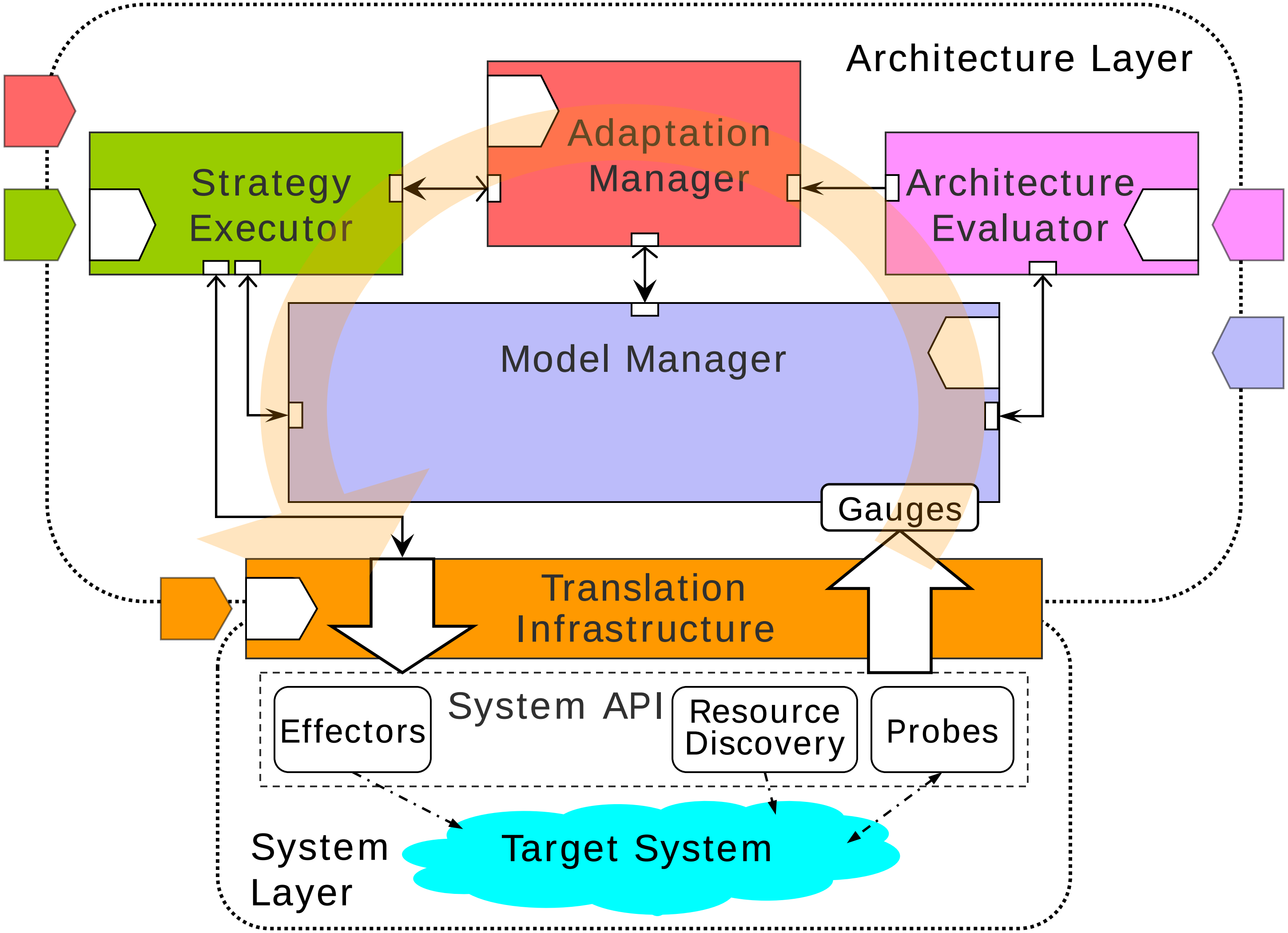
© David Garlan 2017

Rainbow



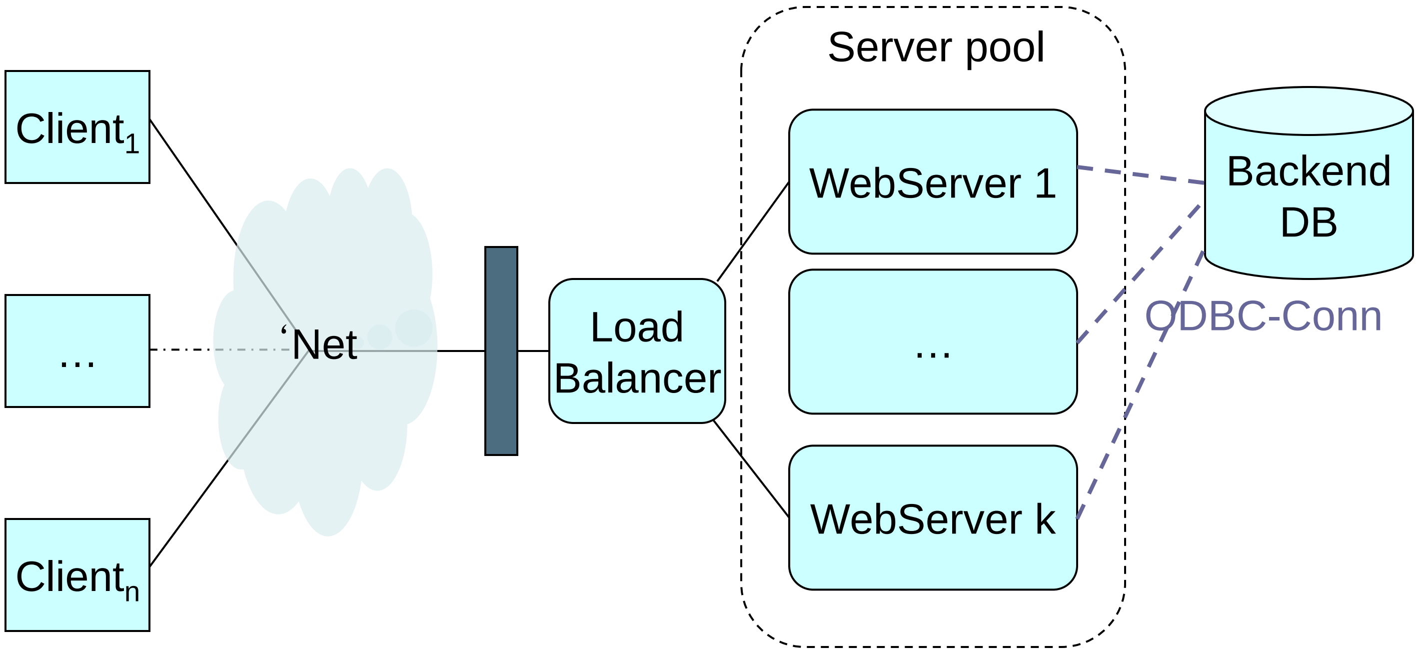
Control Layer
Adaptation Manager
Model Manager
Translation Infrastructure
Target System
System Layer

Rainbow

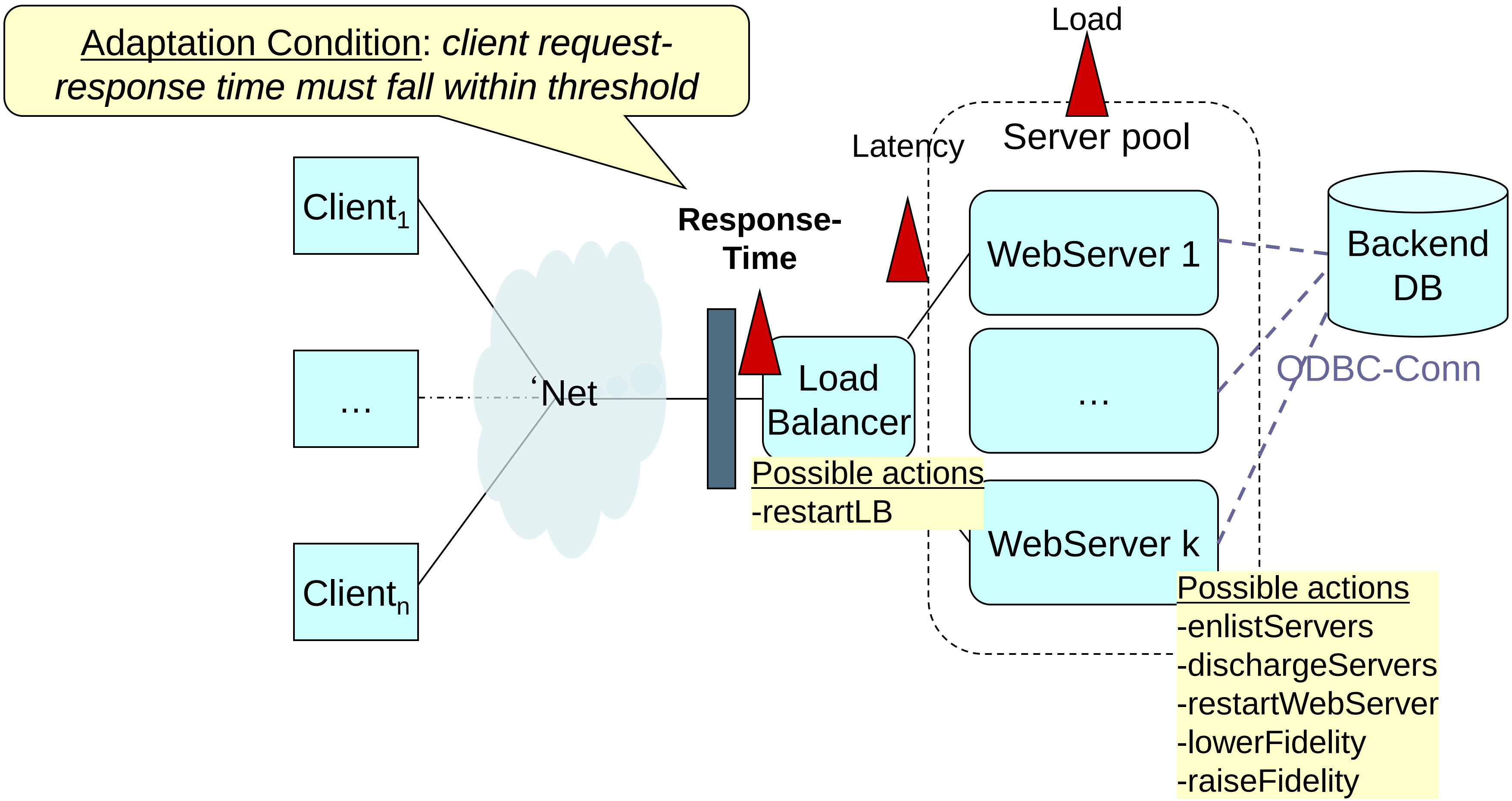


Architecture Layer
Adaptation Manager
Strategy Executor
Architecture Evaluator
Model Manager
Gauges
Translation Infrastructure
System API
Effectors
Resource Discovery
Probes
System Layer
Target System

Self-Adaptation Example: *Znn.com*

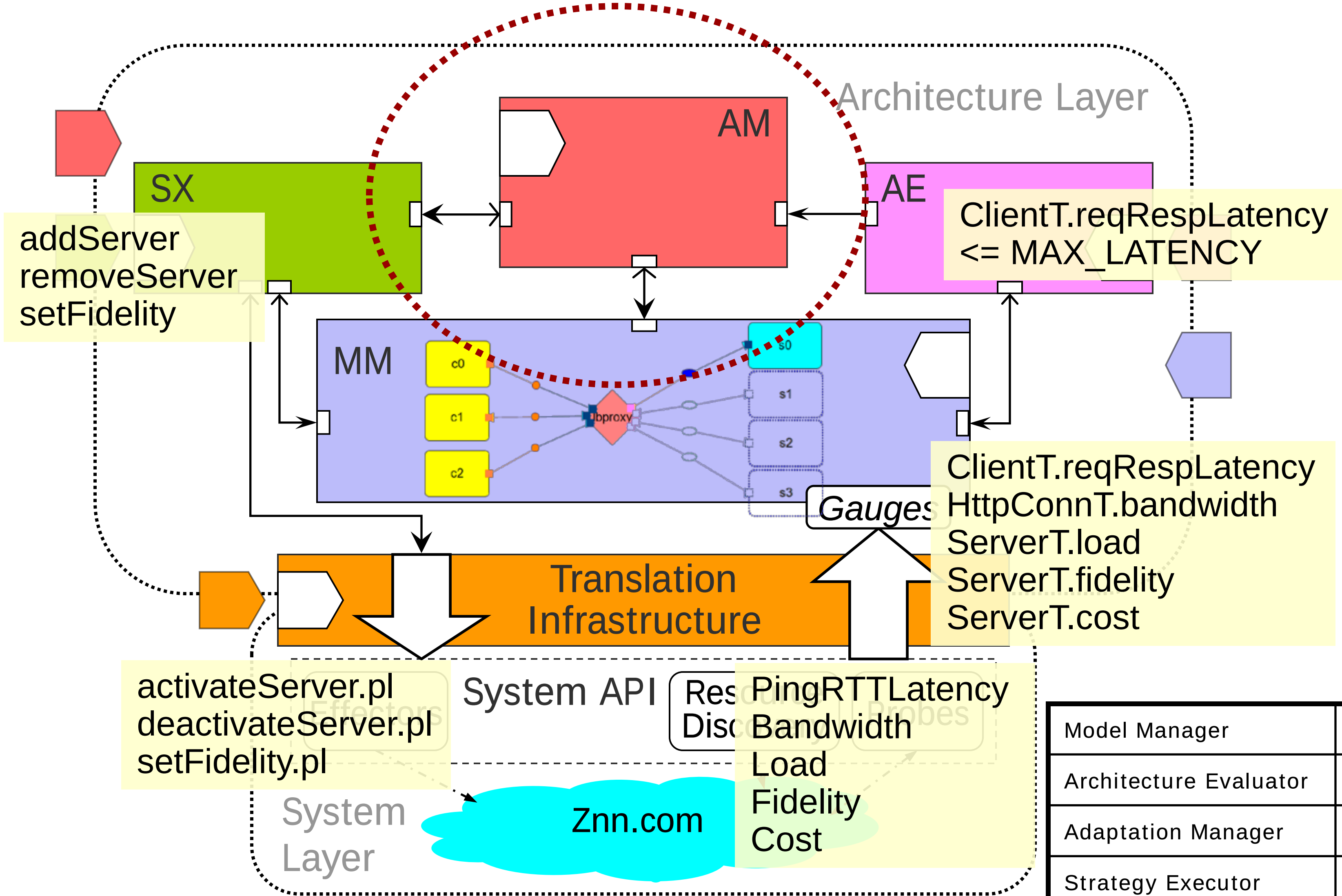


Self-Adaptation Example: *Znn.com*



© David Garlan 2017

Znn.com: Rainbow Customizations



Model Manager	MM
Architecture Evaluator	AE
Adaptation Manager	AM
Strategy Executor	SX

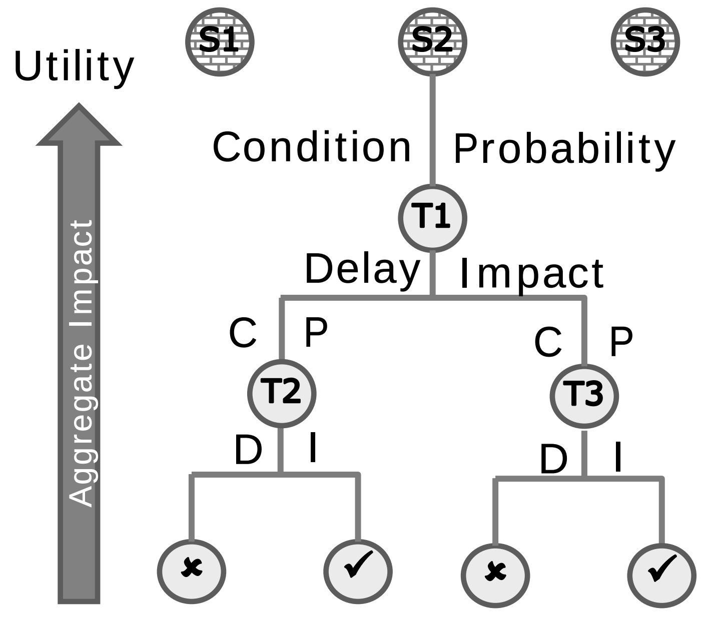
Rainbow Adaptation Decision Overview

- Selection from a set of adaptation strategies
  - Multiple strategies may be applicable in a particular system context
- Language for expressing strategies as a decision tree
  - Conditions: determine which branches are applicable
  - Actions: tactics that modify the system
- Tree is annotated with properties that
  - Permit selection of strategy with highest utility
  - Support formal reasoning about time, uncertainty cost and benefit

© David Garlan 2017

Stitch: A Language for Specifying Self-Adaptation Strategies

- Control-system model: Selection of next action in a strategy depends on observed effects of previous action
- Uncertainty: Probability of taking branch captures non-determinism in choice of action
- Asynchrony: Explicit timing delays to see impact
- Value system: Utility-based selection of best strategy allows context-sensitive adaptation

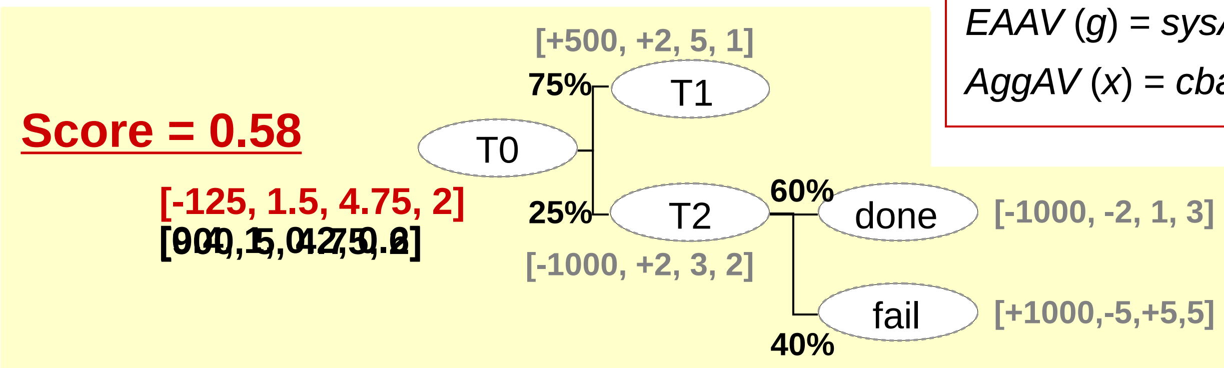


© David Garlan 2017

Strategy Selection

- Given:
  - Quality dimensions and weights (e.g., 4)
  - A strategy with
    - N nodes
    - Branch probabilities as shown
    - Tactic cost-benefit attributes

$u_{latency}(),\ u_{quality}(),\ u_{cost}(),\ u_{disruption}()$
$(w_{latency},\ w_{quality},\ w_{cost},\ w_{disruption})$
$= (0.5, 0.3, 0.1, 0.1)$ **[**= 1**]**



Algorithm

Given tree *g* with node *x* and its children *c*:

$EAAV\ (g) = sysAV + AggAV\ (root\ (g))$

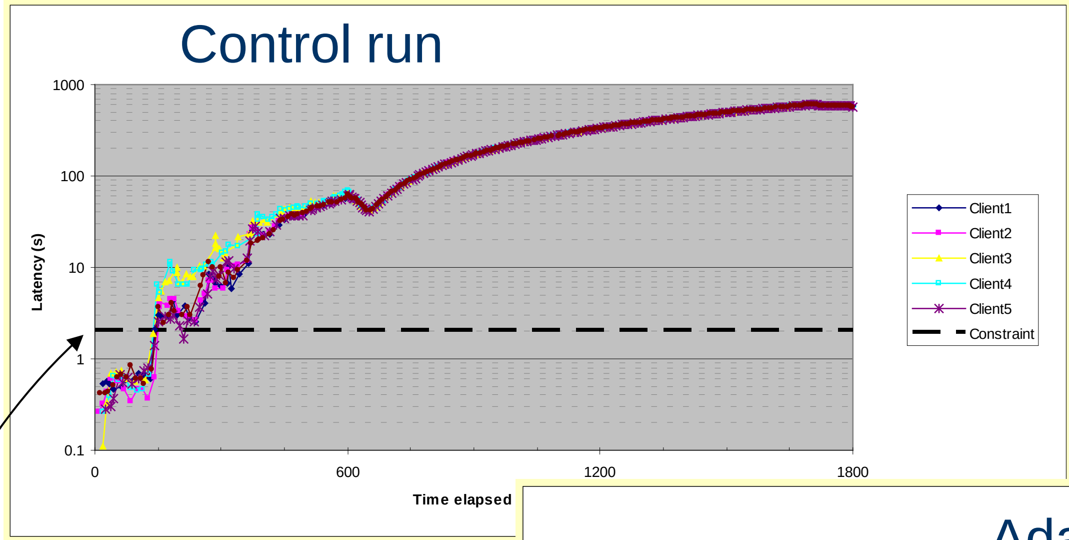
$AggAV\ (x) = cbav(x) + \Sigma_c\ prob(x,c)\ AggAV(c)$

- Propagate cost-benefit vectors up the tree, reduced by branch probabilities
- Merge expected vector with current conditions (assume: [1025, 3.5, 0, 0])
- Evaluate quality attributes against utility functions
- Compute weighted sum to get utility **score**

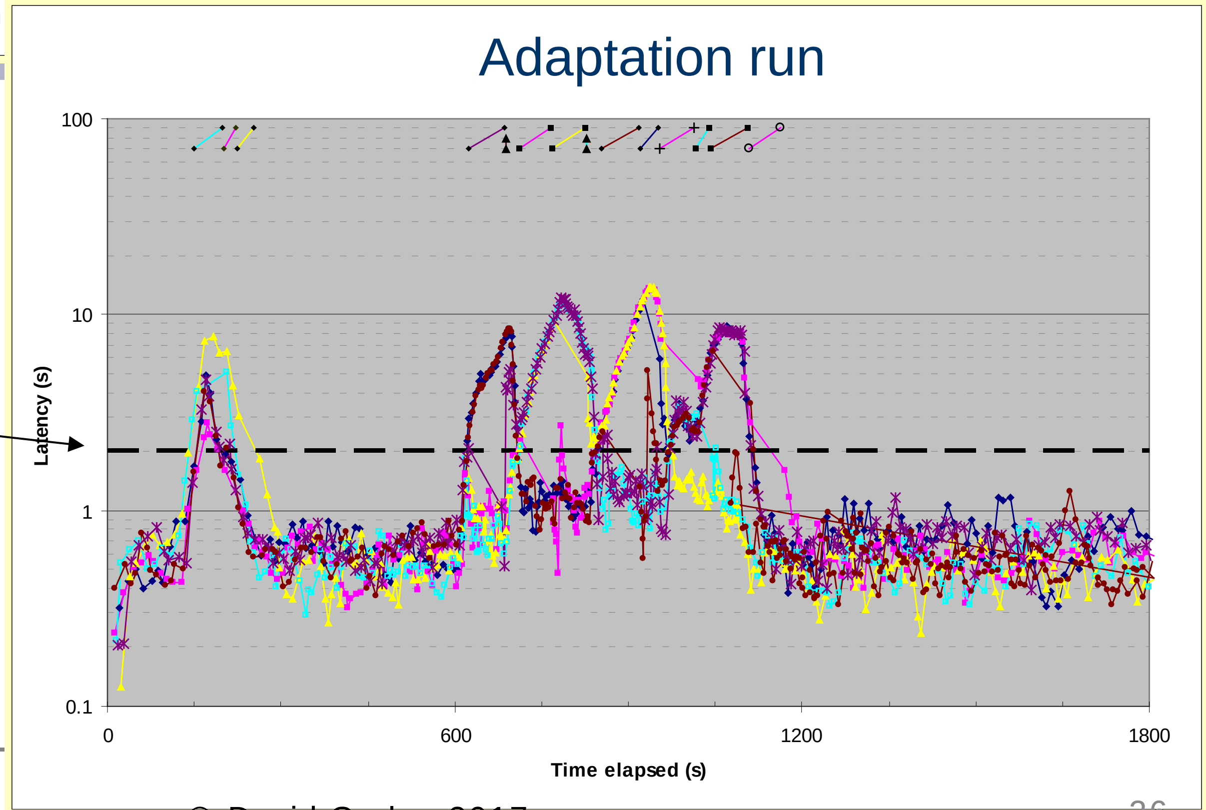
© David Garlan 2017

System Adapts

Data shows that our adaptation approach improves overall system performance



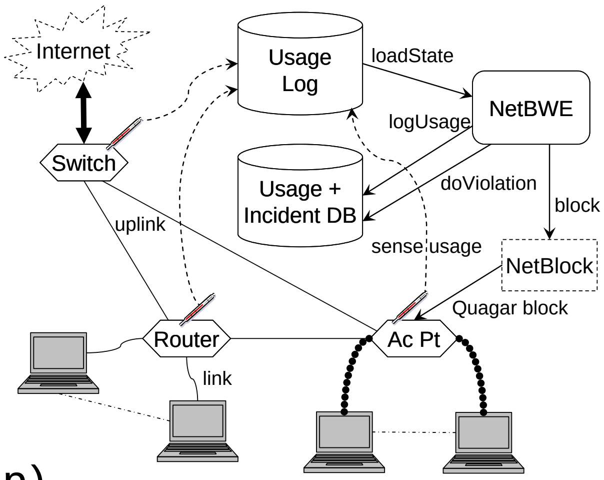
Latency = 2 secs



© David Garlan 2017

System Administration Evaluation

- Sys-admin interviews
  - **Results**: Stitch concepts seem **natural** fit for sys-admin routines
  - Methodology: priming, interview, compose Stitch script from scenarios
  - White problem scenarios: scripts represented in Stitch
  - Actual problem scenarios: structure matches Stitch strategies
- Analysis using CMU sys-admin example: *Netbwe*
  - **Results**:
    - Rainbow captured adaptation concerns
    - Stitch **hoisted** policies buried in Perl code
  - Distinguishable adaptation tasks
    - Core commands as operators
    - Coarser-grained sequence of commands (step) with conditions of applicability and intended effects
    - Adaptations with intermediate condition-actions and observations



Self-adaptive Systems Challenges

1. Self-securing systems
2. Fault diagnosis and localization
3. Human-in-the-loop adaptation
4. Combining Reactive and Deliberative Adaptation
5. Proactive and latency-aware adaptation
6. Architecting for Adaptability
7. Systems of systems

© David Garlan 2017

Application to Security

- Application-layer Denial of Service Attacks
  - Assume an N-tiered model similar to Znn.com
- Quality objectives

Quality	Description
Performance	Request-response time for legitimate users
Cost	Number of active servers
Maliciousness	Percentage of malicious clients
Annoyance	Disruptive side-effects of tactics

© David Garlan 2017

Tactics

Tactic	Description
Add capacity	Activate additional servers to distribute the workload
Blackhole	Blacklist clients; requests are dropped
Reduce service	Reduce content fidelity level (e.g., text vs. images)
Throttle	Limit the rate of requests Accepted by the system
Captcha	Forward requests to Captcha processor to verify that the requester is human
Reauthenticate	Force clients to reauthenticate

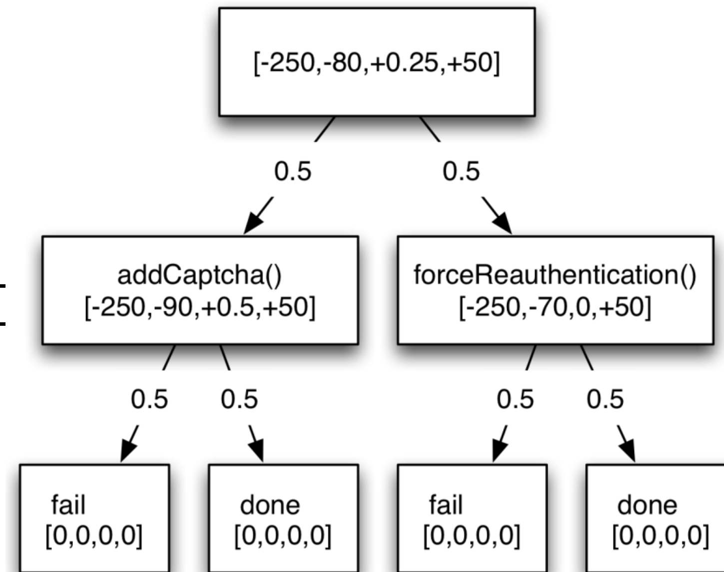
Strategies

Strategy	Description
Outgun/Absorb	Combines Add capacity and Reduce service
Eliminate	Combines Blackholing and Throttling
Challenge	Combines Captcha and Reauthenticate

Tactics and Strategies

```
tactic addCaptcha () {
  condition {exists lb:D.ZNewsLBT in M.components | !lb.captchaEnabled;}
  action {
    set lbs = {select l : D.ZNewsLBT in M.components | !l.captchaEnable
    for (D.ZNewsLBT l : lbs) {
        M.setCaptchaEnabled (l, true);
    }
  }
  effect {forall lb:D.ZNewsLBT in M.components | lb.captchaEnabled;}
}
```

```
strategy Challenge [unhandledMalicious || unhandledSuspicious] {
  t0: (cNotChallenging) -> addCaptcha () @[5000] {
      t0a: (success) -> done;
      t0b: (default) -> fail;
  }
  t1: (!cNotChallenging) -> forceReauthentication () @[5000] {
      t1a: (success) -> done;
      t1b: (default) -> fail;
  }
}
```



© David Garlan 2017

Formal Model – Utility Profile

- Utility profile encodes functions and preferences as reward structures

Utility functions for DoS

U_R	U_M	U_C	U_A
0 : 1.00	0 : 1.00	0 : 1.00	0 : 1.00
100 : 1.00	5 : 1.00	1 : 0.90	100 : 0.00
200 : 0.99	20 : 0.80	2 : 0.30	
500 : 0.90	50 : 0.40	3 : 0.10	
1000 : 0.75	70 : 0.00		
1500 : 0.50			
2000 : 0.25			
4000 : 0.00			

Utility preferences for DoS

Scenario	Priority	w_{U_R}	w_{U_M}	w_{U_C}	w_{U_A}
1	Minimizing number of malicious clients.	0.15	0.6	0.1	0.15
2	Optimizing good client experience.	0.3	0.3	0.1	0.3
3	Keeping cost within budget.	0.2	0.2	0.4	0.2

DoS utility profile encoding

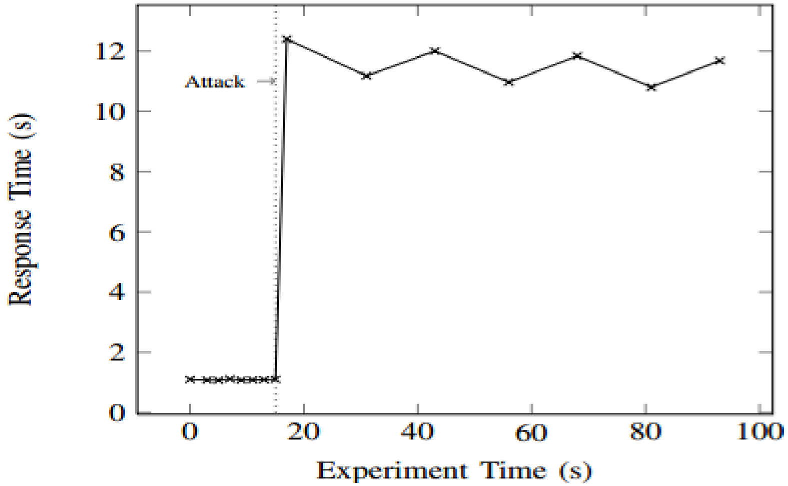
```
formula uM = (mc>=0 & mc <=5? 1:0)
  +(mc>5   & mc <=20? 1+(0.80-1)*((mc-5)/(20-5)):0)
  +(mc>20  & mc <=50? 0.80+(0.40-0.80)*((mc-20)/(50-20)):0)
  +(mc>50  & mc <=70? 0.40+(0.00-0.40)*((mc-50)/(70-50)):0)
  +(mc>70  ? 0:0);
...
rewards "rGU" // Global Utility
  leaf & scenario=1 : 0.15*uR +0.6*uM +0.1*uC +0.15*uA;
...
endrewards
```

Results

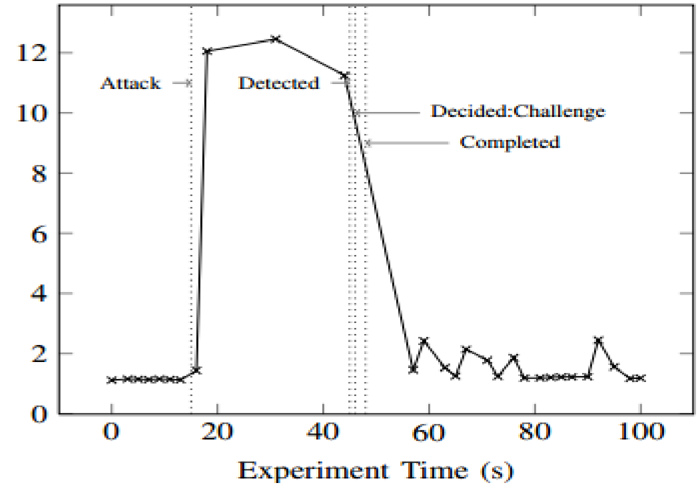
- Different security strategies are picked in different contexts
  - Not hardwired into the system
- Allows combinations of security repair tactics
  - Can create many strategies from the same tactics
- Supports formal reasoning and model checking
  - We use the PRISM probabilistic model checker to analyze strategies
- Allows future addition of security strategies as new tactics become available

Evaluation

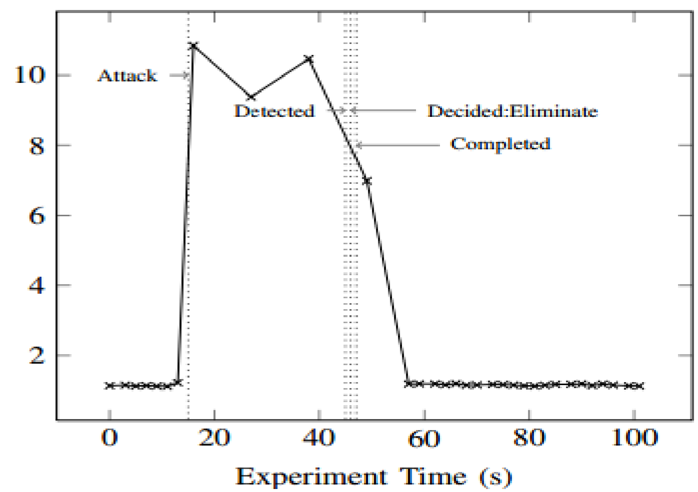
No Adaptation



Minimize Malicious Clients

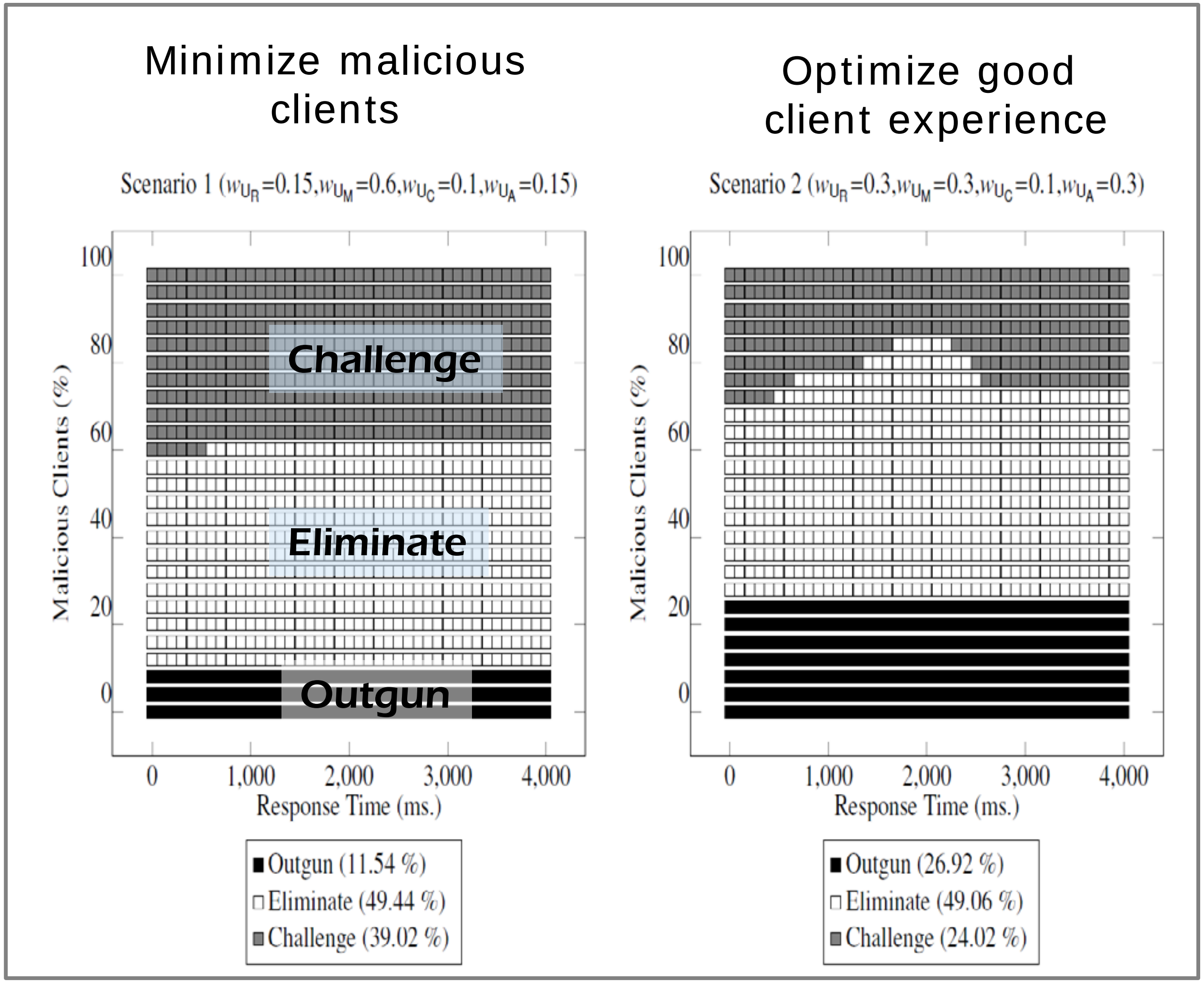


Prioritize Performance



Strategy Selection Analysis

- **Based on quantifying expected utility after strategy execution**
- **Different preferences result in different strategy selections**
- **Choices are consistent**



© David Garlan 2017

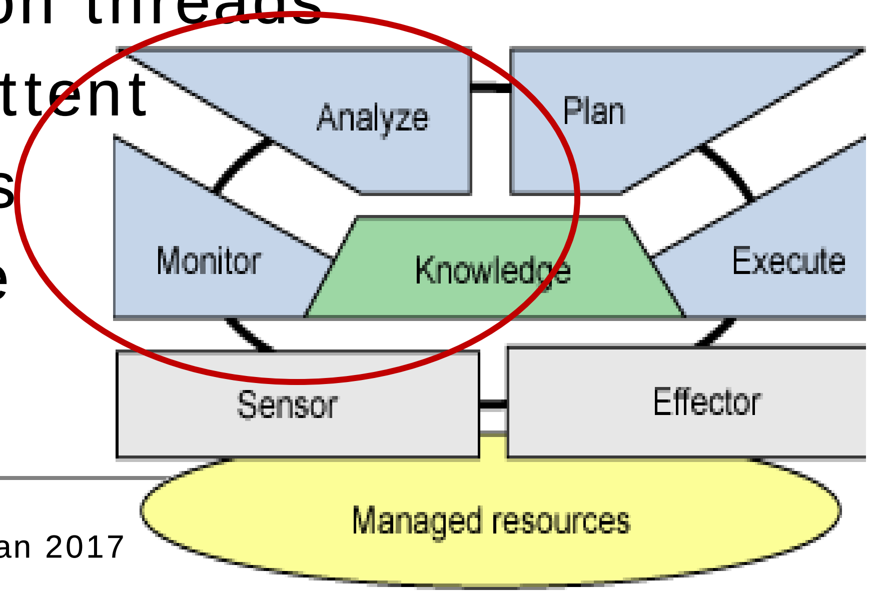
Self-adaptive System Technical Challenges

1. Self-securing systems
2. Fault diagnosis and localization
3. Human-in-the-loop adaptation
4. Combining Reactive and Deliberative Adaptation
5. Proactive and latency-aware adaptation
6. Architecting for Adaptability
7. Systems of systems

© David Garlan 2017

Fault Diagnosis and Localization

- Successful adaptation requires detecting when there is a problem and locating the source of it
- This is a hard problem because:
  - Many possible causes for an observed problem
  - We have incomplete knowledge of the system
  - Many concurrent execution threads
  - Problems may be intermittent
  - May involve combinations
  - Must be done in real time

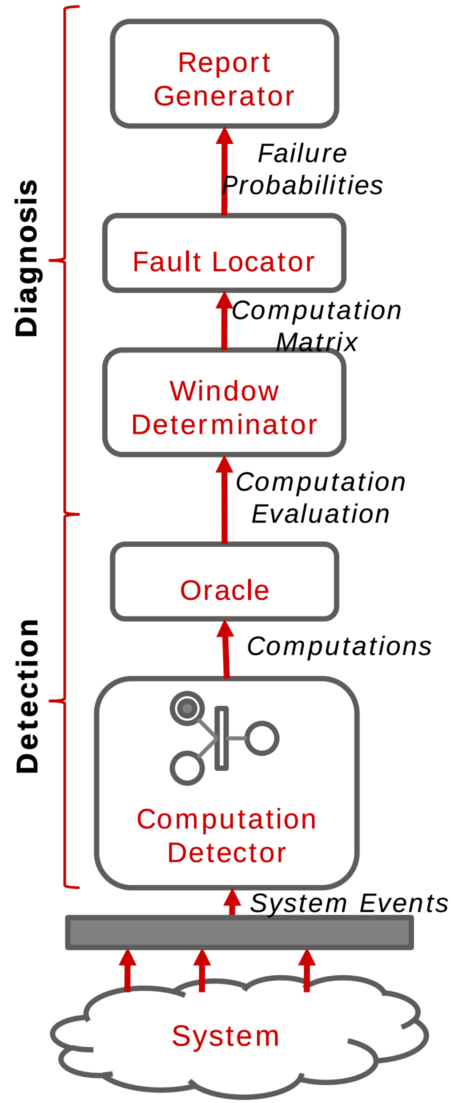


© David Garlan 2017

Approach

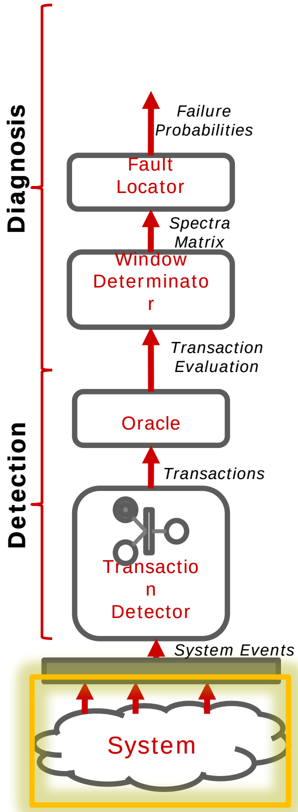
Five step pipeline:

1. Detect *transactions* that map interleaved, concurrent system events to distinct paths in the system
2. Determine whether transactions are successful
3. Create a set of transactions that can be used for analysis
4. Use spectrum-based multiple fault localization do diagnose problems
5. Pass this information to consumers for further action

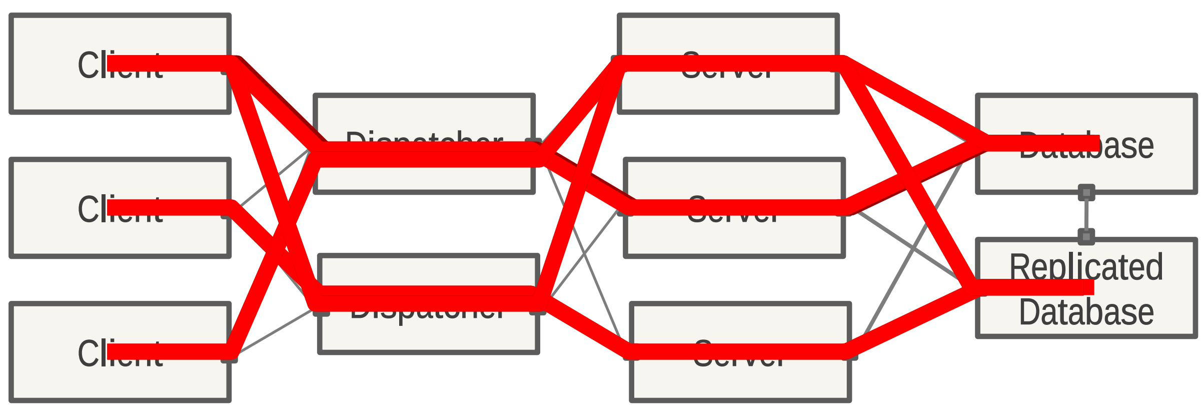


© David Garlan 2017

Example

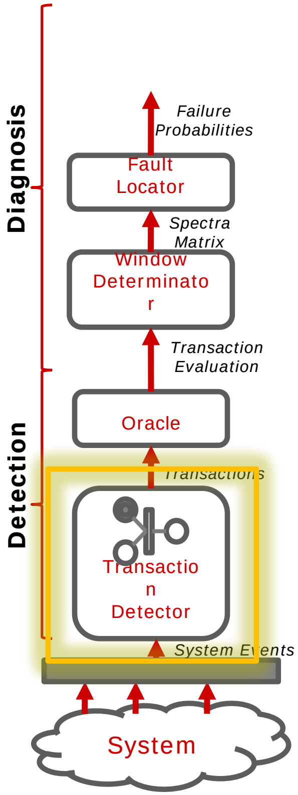


- Web-based system using multiple servers and dispatchers to serve clients
- Multiple concurrent communication threads

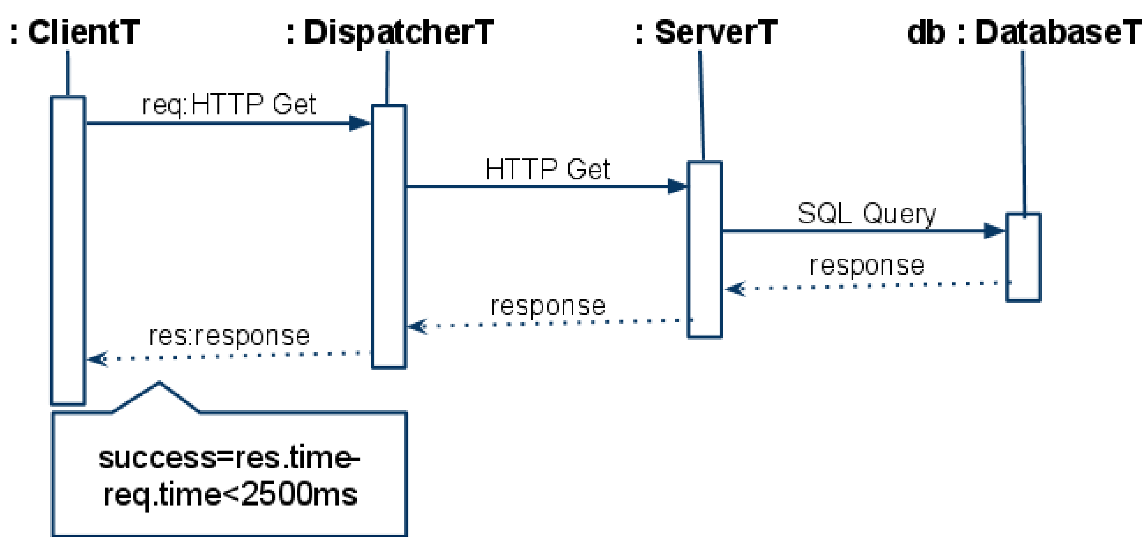


© David Garlan 2017

Transaction Families

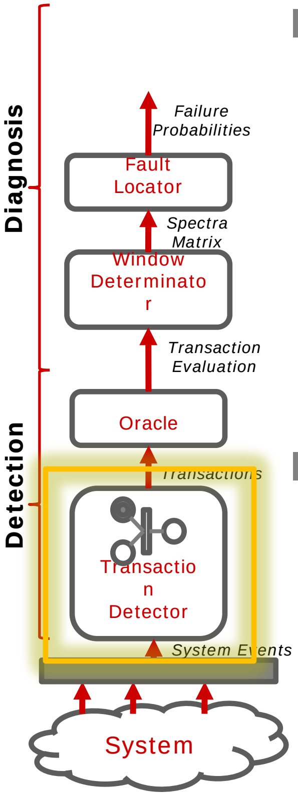


- Transaction families define a parameterized pattern of behaviors
  - Light-weight specification of behavior
  - Define the finite executions
  - Criteria for success/failure



© David Garlan 2017

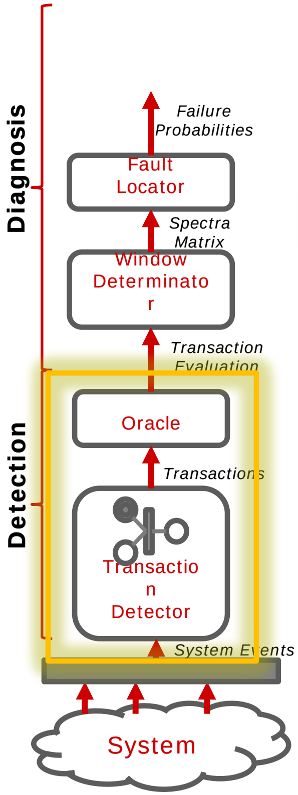
Detecting Transactions

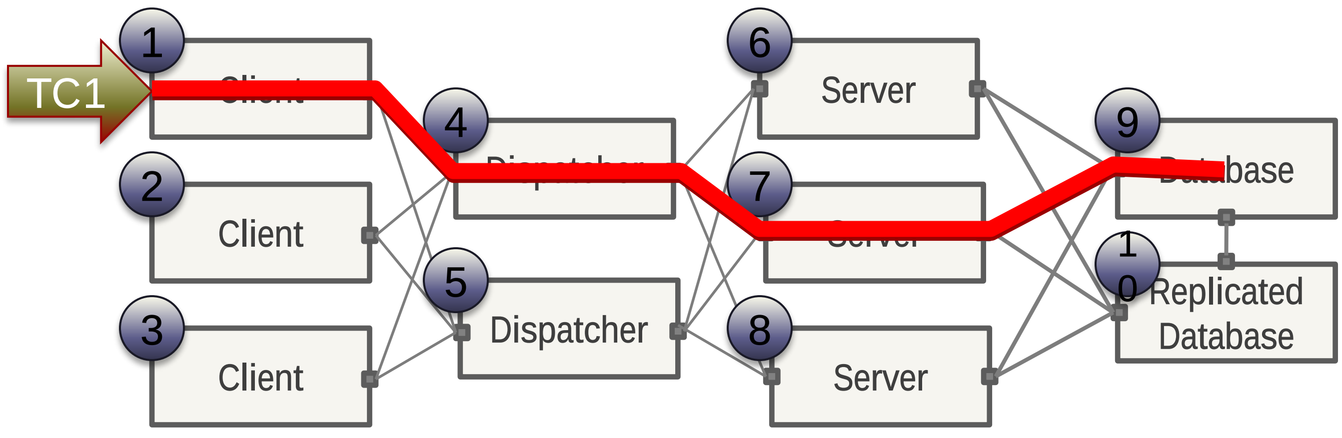


- Map system events to architecture observations
  - Adapt work from dynamic architecture reconstruction* to map events and monitor transaction family instances
- Determine whether the transaction passes or fails
  - Success criteria defined with transaction family

*Schmerl, et al. *Discovering Architectures from Running Systems.* IEEE TOSE 32(7), 2006.

Evaluating Transactions

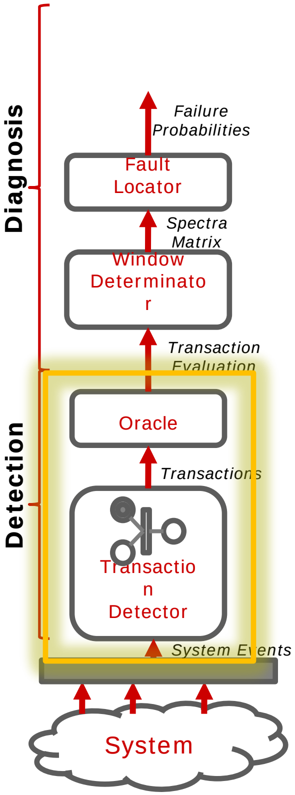


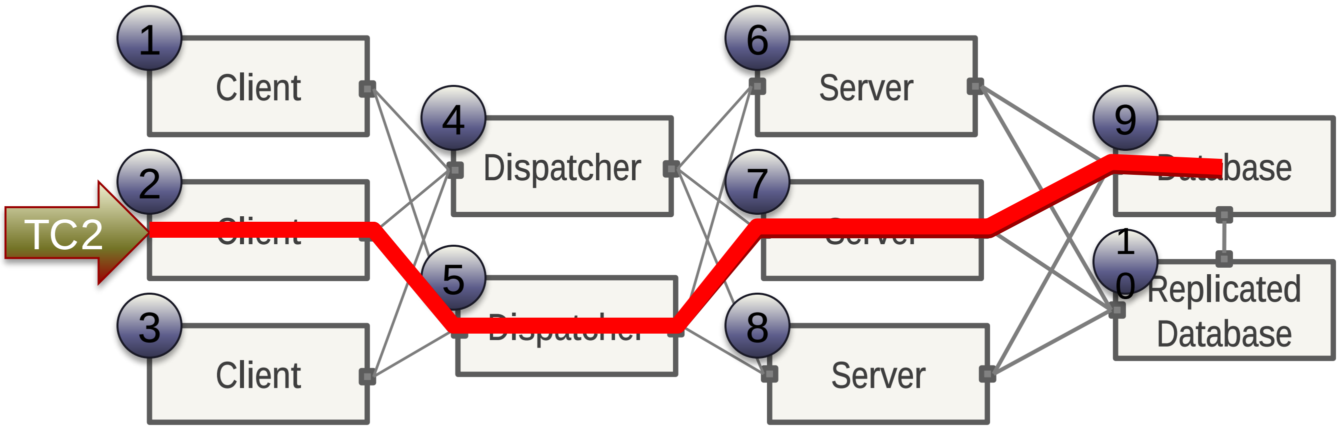


Txn	1	2	3	4	5	6	7	8	9	10	Pass
T1	■			■			■		■		✓

© David Garlan 2017

Evaluating Transactions



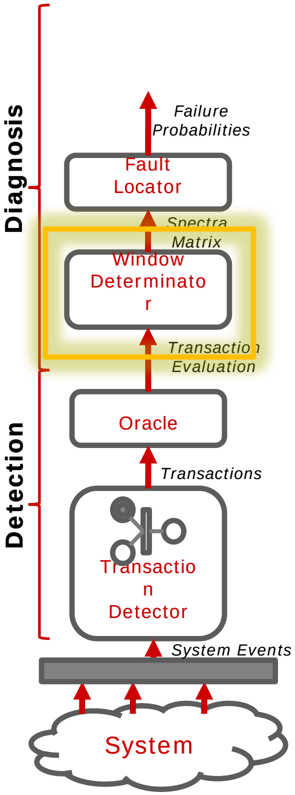


Txn	1	2	3	4	5	6	7	8	9	10	Pass
T1	■			■			■		■		✓
T2		■			■		■		■		✗

...and so on

© David Garlan 2017

Localization



- Use a technique called "Spectrum-based Fault Localization"

Txn	1	2	3	4	5	6	7	8	9	10	Pass
T1											✓
T2											✗
T3											✓
T4											✓
T5											✗
…	…	…	…	…	…	…	…	…	…	…	…
Tn											✗

Samsung Case Study

- Sponsored by Samsung
- Diagnosis for Manufacturing Control Systems
  - Stringent requirements for up-time
- Key challenge is scalability and performance
  - High volume of monitored events
  - Many components
  - Must diagnose problem quickly
- Successful demonstration
  - Simulated system
  - Can handle thousands of events and find real failures

Target System

- Large scale industrial system for manufacturing of semiconductors.
  - System controls wafer manufacture, deciding which systems are used to process what.
  - System is divided into multiple components exchanging messages over an event bus.
- Typical failures
  - Messages are lost (or not sent at all)
  - Messages are sent too late
  - Unexpected messages are sent
  - Database performance slowdowns, affecting overall system performance

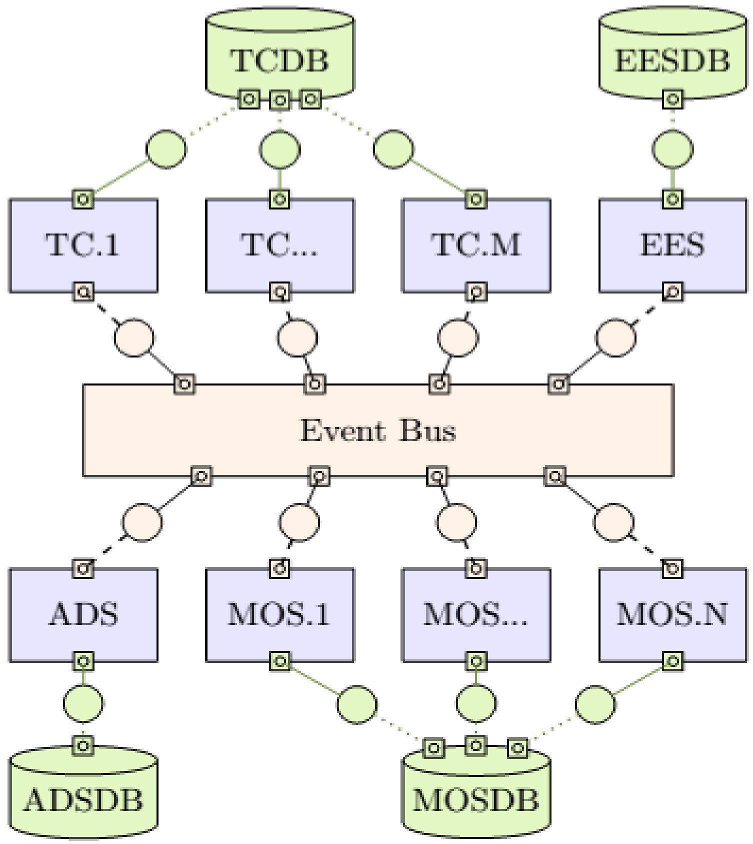
© David Garlan 2017

Samsung Challenges

- Why is it difficult to diagnose failures in this system?
  - Protocols work correctly most of the time.
  - Problems are serious, but rare: a lot needs to be monitored to see a failure happening.
  - Given the volume of data (~2000 messages / second) it is not possible for human operators to identify problems quickly.
  - The complexity of the system makes it difficult for developers to figure out where the true source of a problem is.

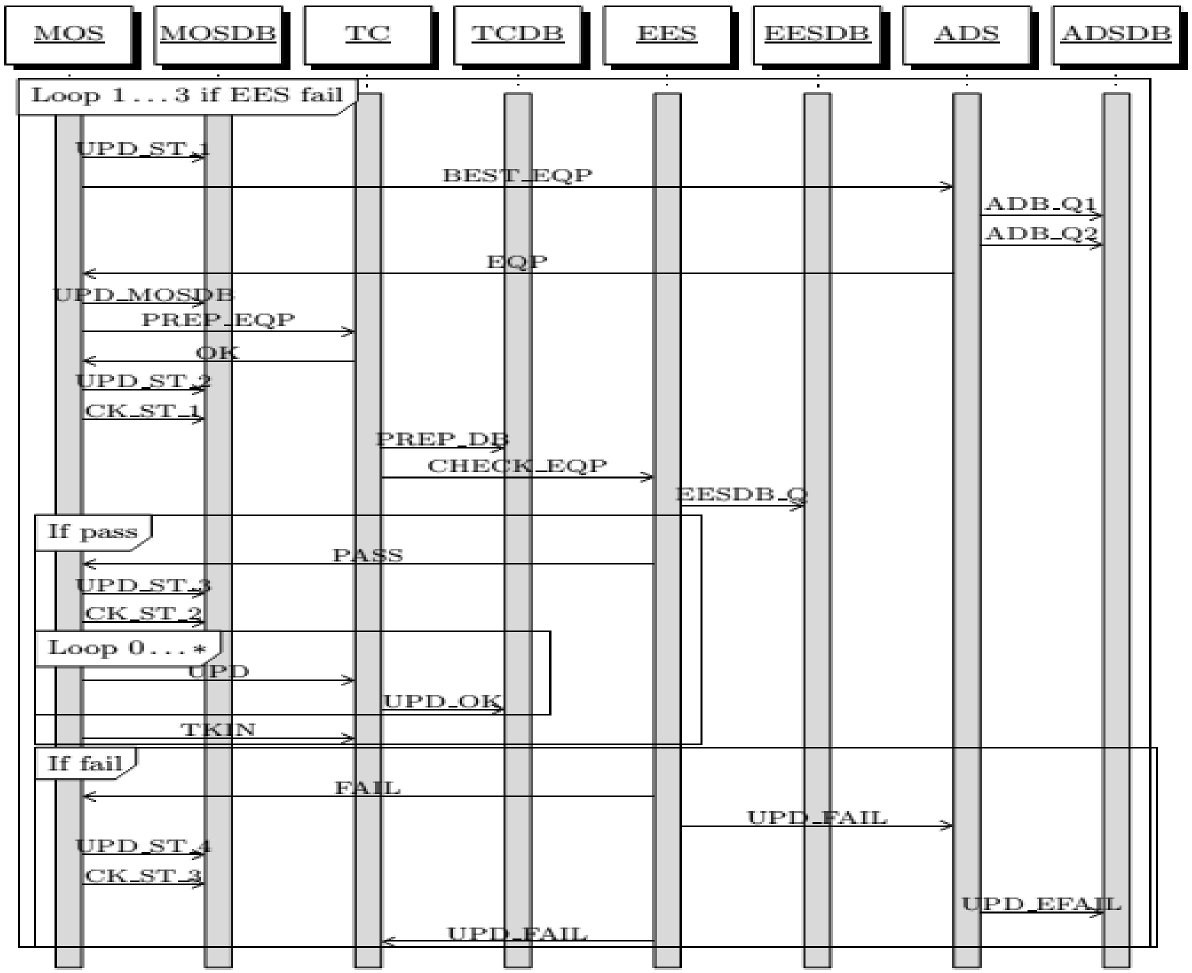
© David Garlan 2017

Simulated System



© David Garlan 2017

The TKIN Protocol



Results

- Can handle high volume in real time
  - Thousands of events
- Diagnosis time is low
  - Under 20 seconds for all classes of failure modeled
- Accuracy is high
  - Rankings are consistent with actual faults

© David Garlan 2017

Additional challenges

- Diagnose faults even when we can't monitor all components
  - Monitoring is expensive
  - Examples: (a) Samsung doesn't directly monitor it's databases, which must have maximum performance; (b) Power for sensor nets.
- Determine an optimal placement of probes
  - Depends on many factors: cost of probe, ability to repair, likelihood of failure
- Adapt probes at run-time
  - Example: increase visibility when you think there's a problem

© David Garlan 2017

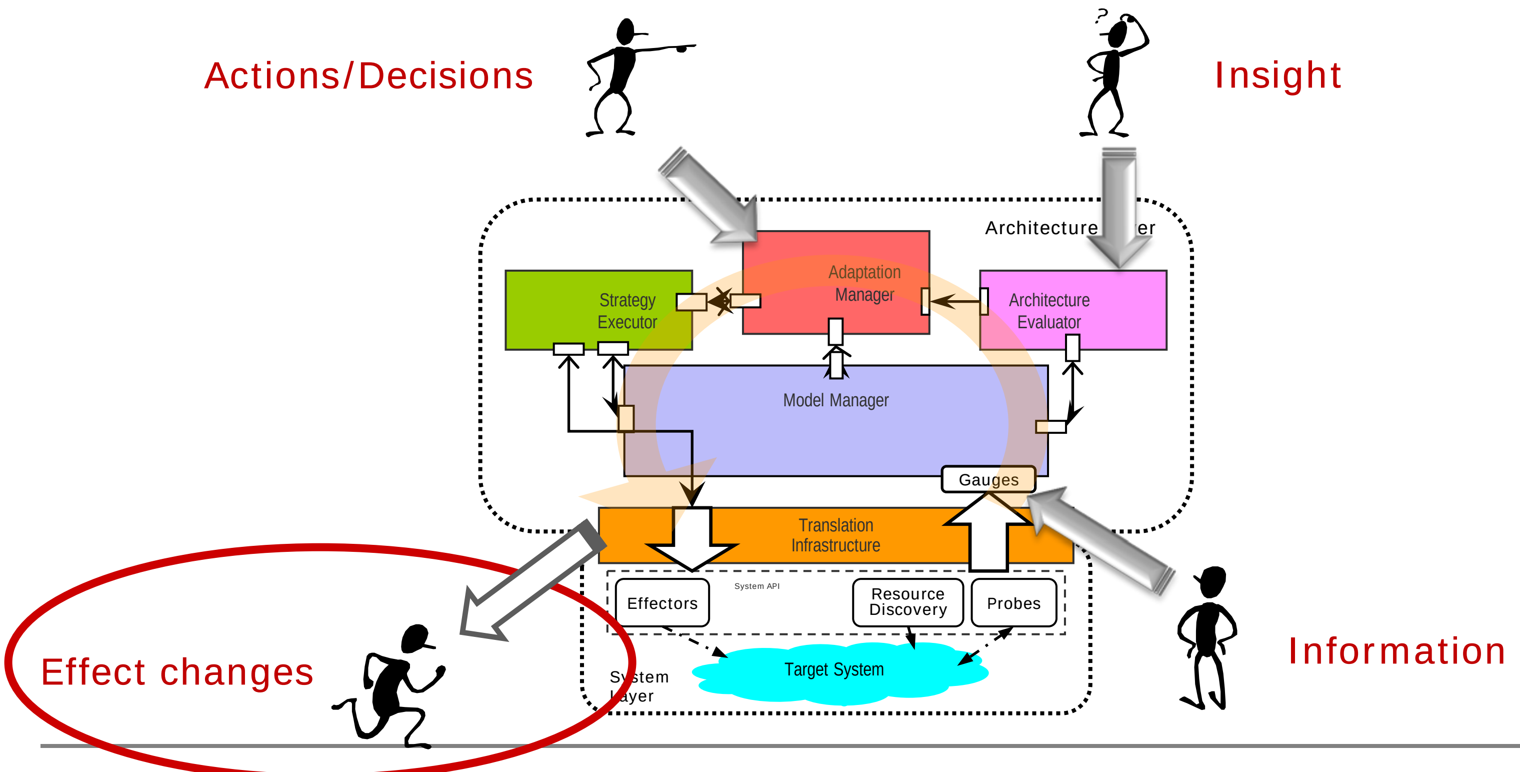
Self-adaptive System Technical Challenges

1. Self-securing systems
2. Fault diagnosis and localization
3. Human-in-the-loop adaptation
4. Combining Reactive and Deliberative Adaptation
5. Proactive and latency-aware adaptation
6. Architecting for Adaptability
7. Systems of systems

© David Garlan 2017

Human-in-the-loop Adaptation

- Real systems often require humans and automated systems to collaborate
- Humans may be involved in different ways



Challenges for Human "Actuation"

- Different humans have different capabilities, permissions, roles, and mental states
  - Varying human attention and readiness to be involved
- The same effect may be accomplished with an automatic mechanism
  - Time-scale differences
  - Effectiveness differences
- Implies the need for a way to determine when to involve the user

© David Garlan 2017

Model for Human Involvement

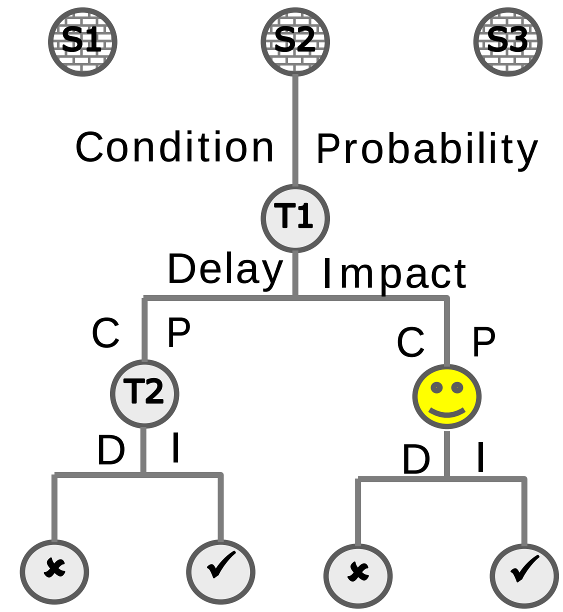
- Opportunity-Willingness-Capability Model (OWC)*
  - Inspiration from human-cyber design
- Opportunity:
  - Is the human in a position to carry out an action
  - E.g., Physically located on site? Access to the room? Has permissions?
- Capability:
  - How likely the human is to succeed at the task
  - E.g., level of training, seniority, experience.
- Willingness:
  - How likely the human is to do the task if asked
  - E.g., level of attention, stress, incentives

*Eskins, Sanders: The Multiple-Asymmetric-Utility System Model: A Framework for Modeling Cyber-Human Systems.

© David Garlan 2017

Integration with Rainbow

- Some tactics are enacted by humans
- Opportunity is captured in strategy conditions
- Willingness and Capability affect probabilities
- Timing captured by delay -- human tactics usually have longer delays than automated execution
- Normal strategy evaluation and execution can then be used



Some Additional Self-adaptive System Technical Challenges

1. Self-securing systems
2. Fault diagnosis and localization
3. Human-in-the-loop adaptation

- Combining Reactive and Deliberative Adaptation
- Proactive and latency-aware adaptation
- Architecting for Adaptability
- Systems of systems

Conclusion

- Today's systems must adapt to meet dynamically changing environments, failures, attacks, requirements
- Architecture models and an adaptation language can be combined for effective self-adaption
- Rainbow: a framework for MAPE-based adaptation
  - Uses architecture models and a provides a language for self-adaptation
  - Supports the ability to evolve and reason about self-adaptation capabilities
- Self-adaptation is an active area of research with many challenges, but huge potential to impact the design and implementation of systems.

© David Garlan 2017

To get involved ...

- SEAMS: International Symposium on Software Engineering for Adaptive and Self-managing Systems
  - Co-located with ICSE
  - Conference: May 22-23 in Buenos Aires
- Other conferences
  - ICAC: International Conference on Autonomic Computing, mid-July
  - SASO: Foundations and Applications of Self* Systems, September

© David Garlan 2017

References

- Rainbow
  - “Software Architecture-Based Self-Adaptation,” David Garlan, Bradley Schmerl and Shang-Wen Cheng. *Autonomic Computing and Networking*, ISBN 978-0-387-89827-8, Springer, 2009.
  - “Increasing System Dependability through Architecture-based Self-repair.” Garlan, Cheng & Schmerl. *Architecting Dependable Systems*, Springer-Verlag, 2003.
  - “Rainbow: Architecture-Based Self Adaptation with Reusable Infrastructure.” D. Garlan, et al. IEEE Computer, Vol. 37(10), October 2004.
  - “Stitch: A Language for Architecture-Based Self-Adaptation.” S.W. Cheng and D. Garlan. *Journal of Systems and Software,* Vol. 85(12), December 2012.

© David Garlan 2017

- Diagnosis
  - "Architecture-based Run-time Fault Diagnosis," Paulo Casanova, Bradley Schmerl, David Garlan and Rui Abreu. *Proc. of the 5th European Conference on Software Architecture, Sept 2011.*
  - "Diagnosing Unobserved Components in Self-Adaptive Systems." Paulo Casanova, David Garlan, Bradley Schmerl and Rui Abreu. 9th International Symp. on Software Engineering for Adaptive and Self-Managing Systems, Hyderabad, India, June 2014.
- Proactivity
  - "Stochastic Game Analysis and Latency Awareness for Proactive Self-Adaptation," J. Cámara, G. A. Moreno & David Garlan. In *Proc. of the 9th Intl Conf. on Software Engineering for Adaptive and Self-Managing Systems (SEAMS),* June 2014.
  - "Analyzing Latency-aware Self-adaptation using Stochastic Games and Simulations." Javier Cámara, Gabriel A. Moreno, David Garlan and Bradley Schmerl. In *ACM Transactions on Autonomous and Adaptive Systems*, 2015.

© David Garlan 2017

- Planning
  - “Optimal Planning for Architecture-Based Self-Adaptation via Model Checking of Stochastic Games,” Javier Cámara, David Garlan, Bradley Schmerl and Ashutosh Pandey. *Proc. of the 10th DADS Track of the 30th ACM Symposium on Applied Computing*, Salamanca, Spain, 13-17 April 2015.
- Human-in-the-loop
  - Javier Cámara, Gabriel A. Moreno and David Garlan. Reasoning about Human Participation in Self-Adaptive Systems. *Proc. of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)*, Florence, Italy, 18-19 May 2015.
- Security
  - “Architecture-Based Self-Protection: Composing and Reasoning about Denial-of-Service Mitigations.” Schmerl, et al. In *Proc. of HotSoS 2014: 2014 Symposium and Bootcamp on the Science of Security,* April 2014.