EECS 4313

Software Engineering Testing

Topic 13:

Load Testing

Zhen Ming (Jack) Jiang

Acknowledgement

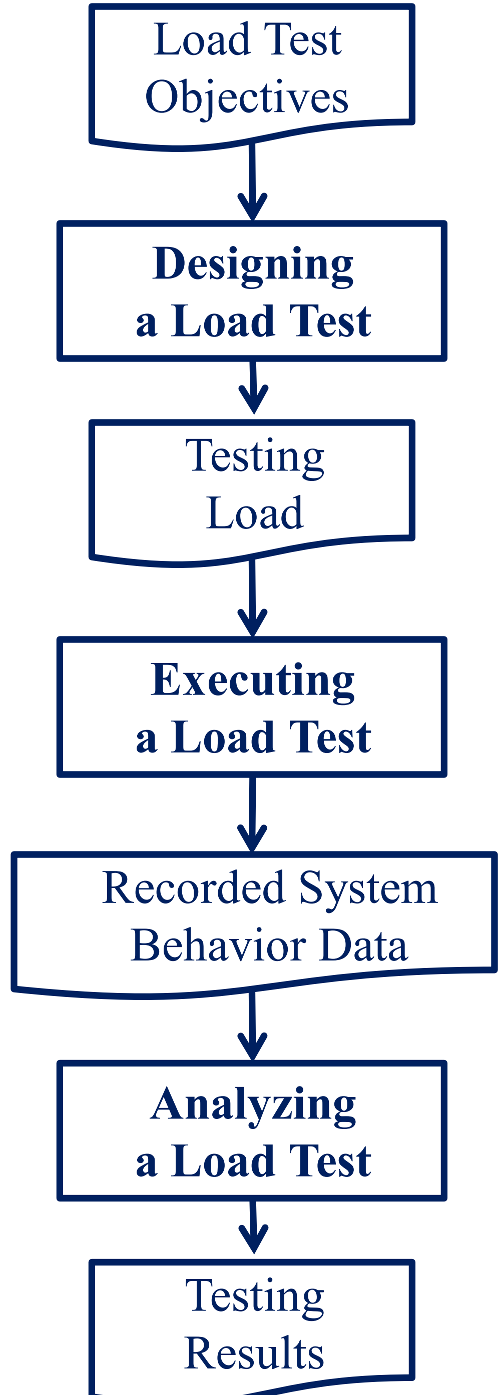
- Ahmed Hassan
- Derek Foo
- Haroon Malik
- Mark Syer
- Parminder Flora

Roadmap

Motivation

Techniques Used in a Load Test

Looking Forward



(Ultra) Large-Scale Software Systems

stackoverflow

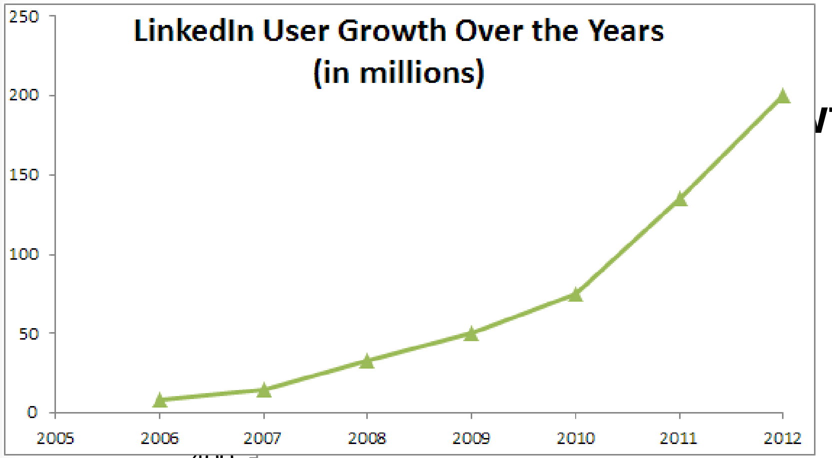
4 million users
2600-3000 req/sec on most weekdays

SAP ERP

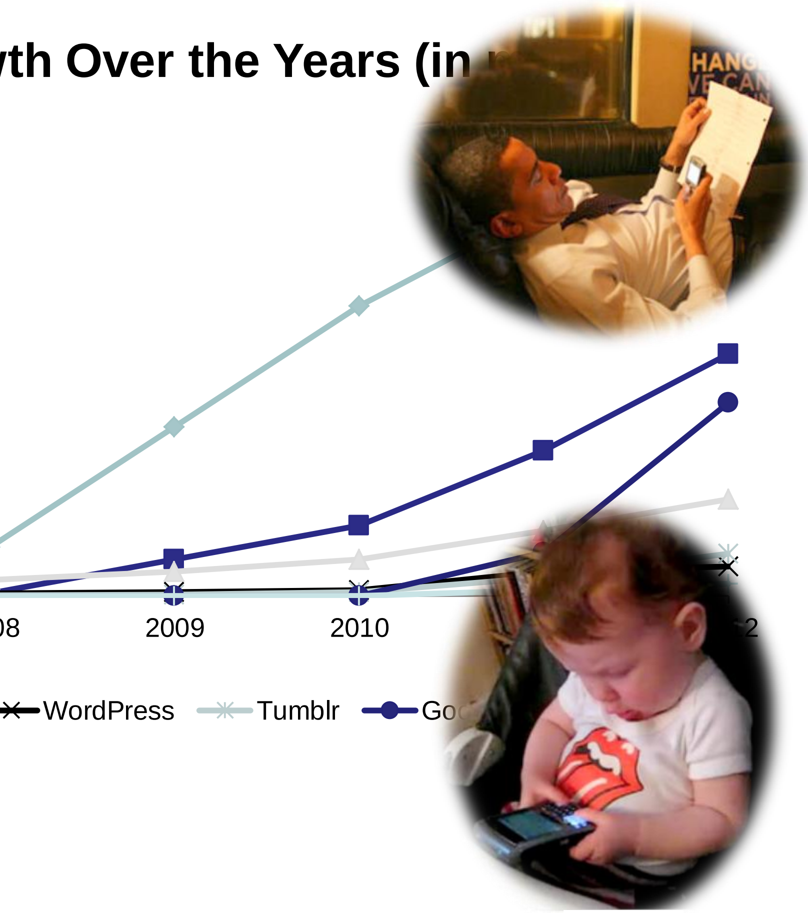
WhatsApp

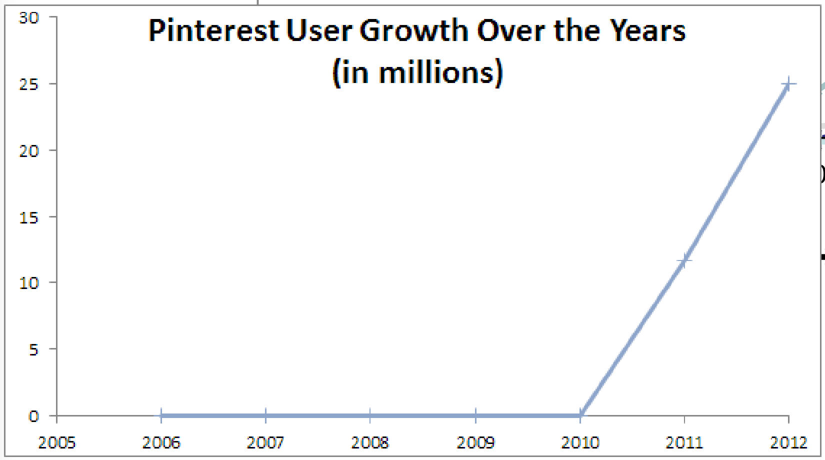
450 million active users
> 50 billion messages every day

Rapid Growth and Varying Usage Patterns



400





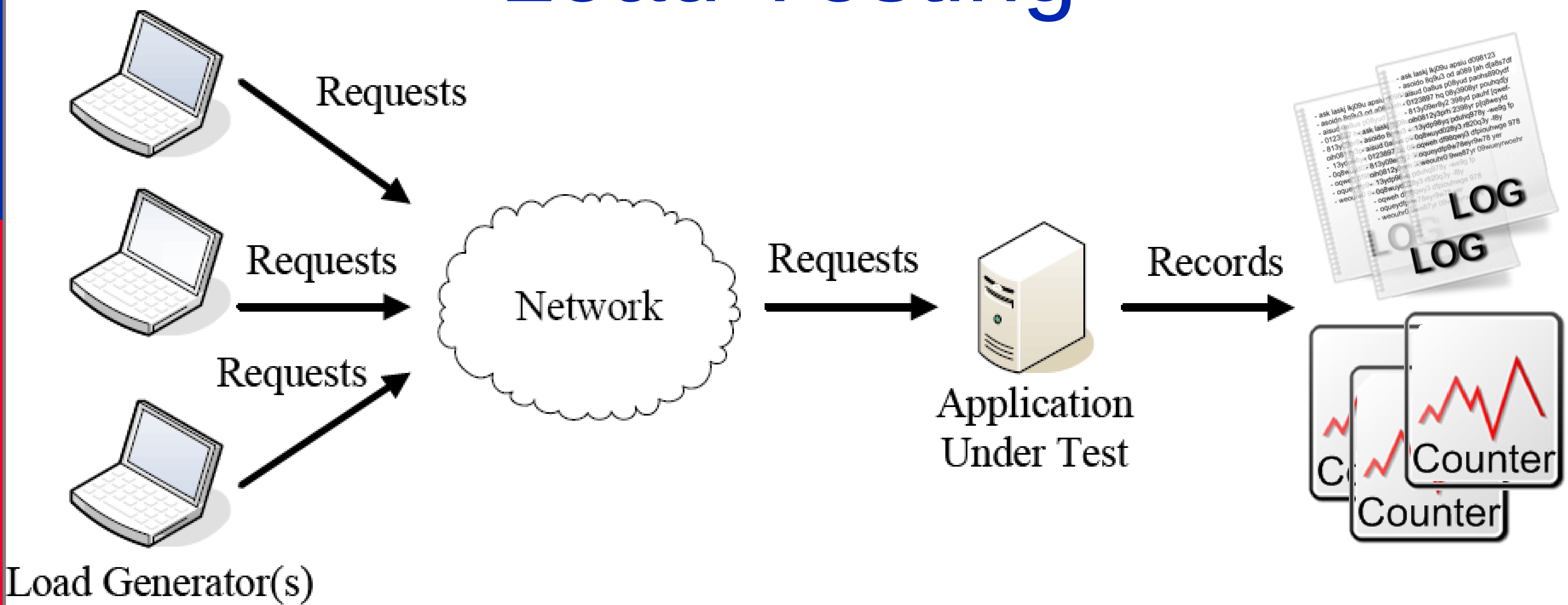
Most field problems for large-scale

lost $7.2 million

89 million users

(05/24/13)

Load Testing



Test Design **Test Execution** **Test Analysis**

Mimics multiple users repeatedly performing the same tasks

Take hours or even days

-

-

Experimental Design

Experimental Design

- Suppose a system has 5 user configuration parameters. Three out of five parameters have 2 possible values and the other two parameters have 3 possible values. Hence, there are $2^3 \times 3^2 = 72$ possible configurations to test.
- Apache webserver has 172 user configuration parameters (158 binary options). This system has 1.8×10^{55} possible configurations to test!

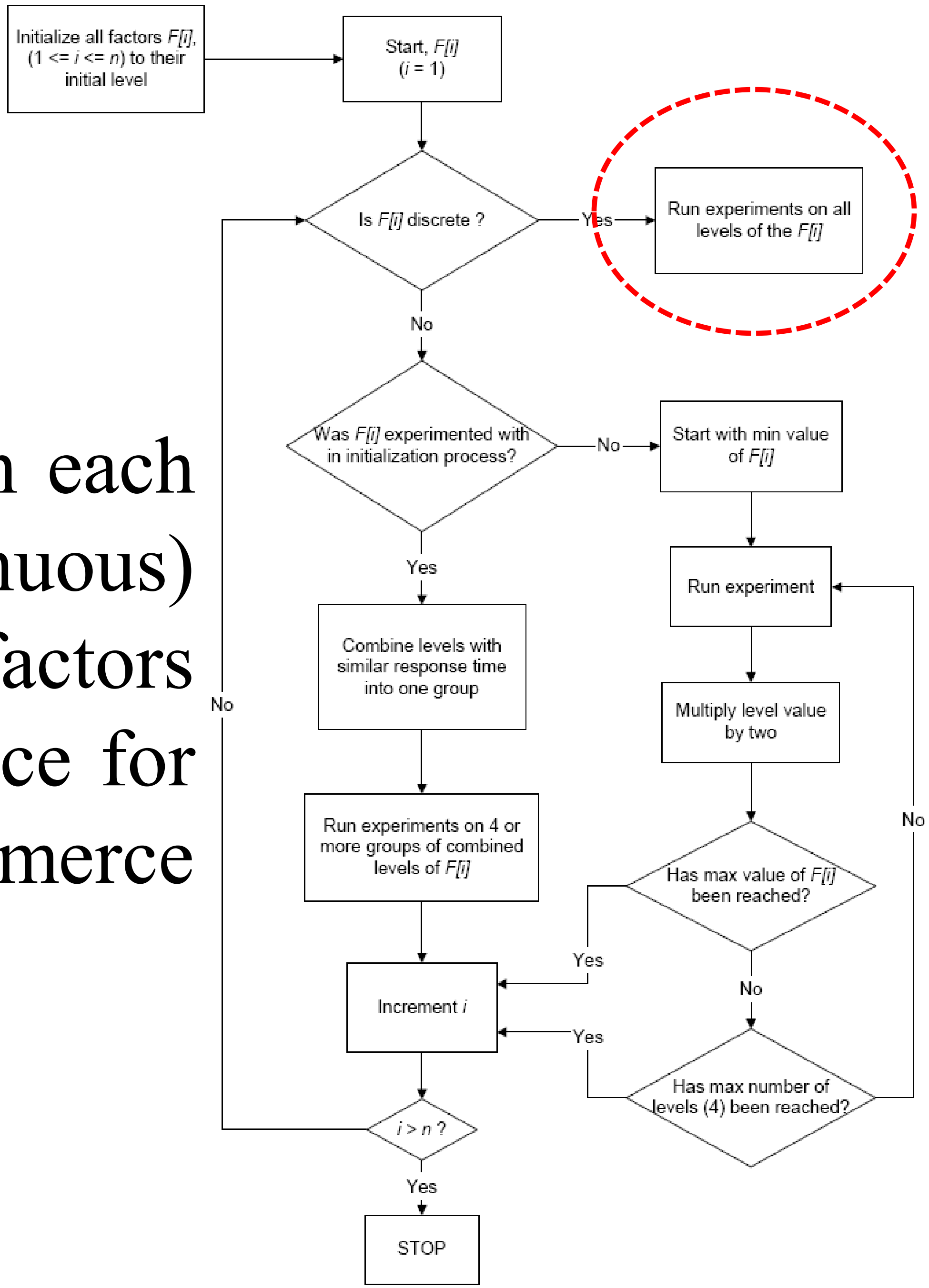
The goal of a proper *experimental design* is to obtain the maximum information with the minimum number of experiments.

Experimental Design Terminologies

- The outcome of an experiment is called the ***response variable***.
  - *E.g.*, throughput and response time for the tasks.
- Each variable that affects the response variable and has several alternatives is called a ***factor***.
  - *E.g.,* to measure the performance of a workstation, there are five factors: CPU type, memory size, number of disk drives and workload.
- The values that a factor can have are called ***levels***.
  - E.g., Memory size has 3 levels: 2 GB, 6 GB and 12 GB
- Repetition of all or some experiments is called ***replication***.
- ***Interaction effects***: Two factors A and B are said to interact if the effect of one depends on the other.

Ad-hoc Approach

Iteratively going through each (discrete and continuous) factors and identity factors which impact performance for an three-tired e-commerce system.



[Sopitkamol et al., WOSP 2005]

Covering Array

- A *t-way covering array* for a given input space model is a set of configurations in which each valid combination of factor-values for every combination of t factors appears at least once.
- Suppose a system has 5 user configuration parameters. Three out of five parameters have 2 possible values (0, 1) and the other two parameters have 3 possible values (0, 1, 2). There are total $2^3 \times 3^2 = 72$ possible configurations to test.

A 2-way covering array

A	B	C	D	E
0	1	1	2	0
0	0	0	0	0
0	0	0	1	1
1	1	1	0	1
0	1	0	0	2
1	0	1	1	0
1	1	1	1	2
1	0	0	2	1
1	0	0	2	2

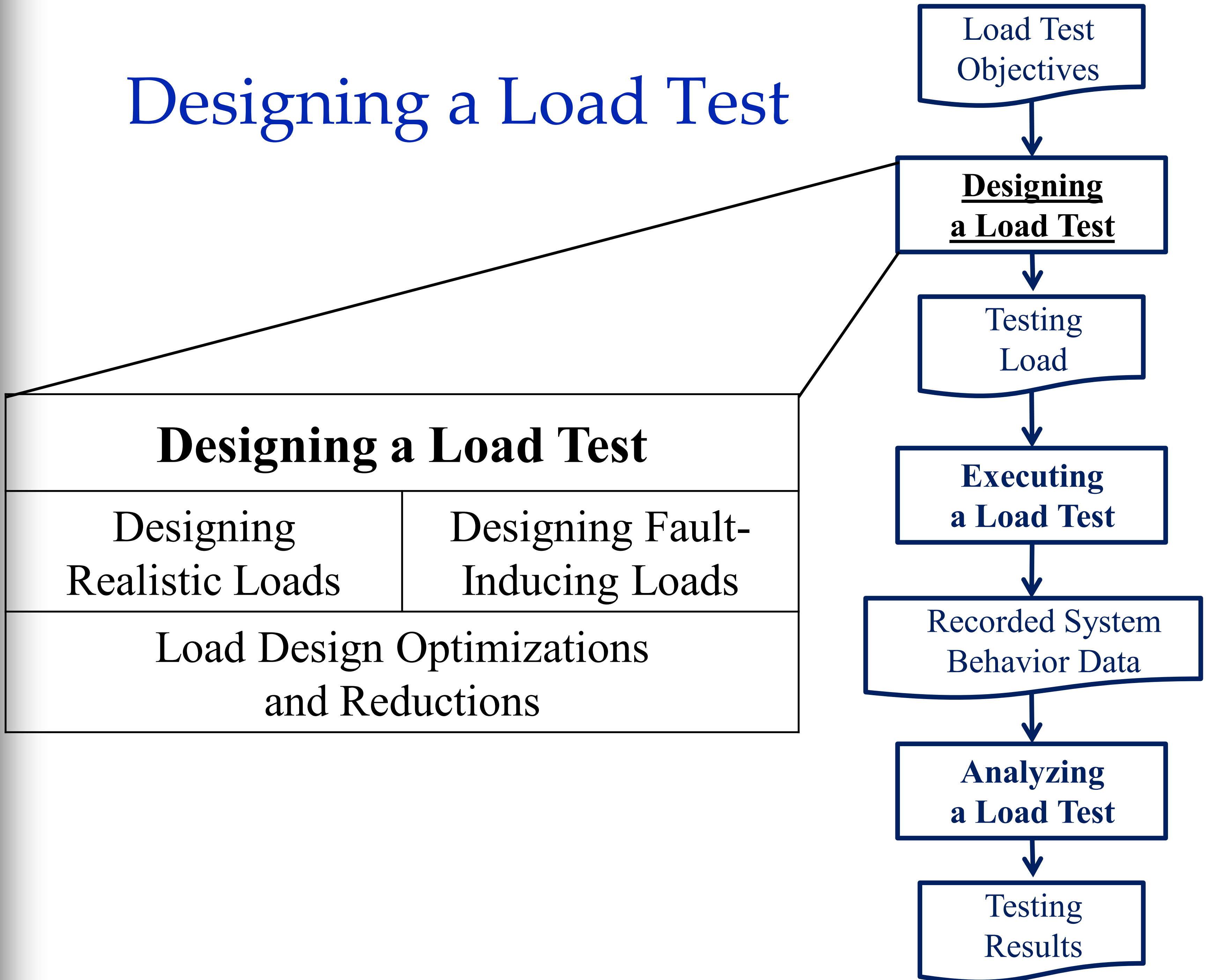
Covering Array and CIT

- There are many other kinds of covering array like: variable-strength covering array, test case-aware covering array, etc.
- *Combinatorial Interaction Testing* (CIT) models a system under test as a set of factors, each of which takes its values from a particular domain. CIT generates a sample that meets the specific coverage criteria (e.g., 3-way coverage).
- Many commercial and free tools: http://pairwise.org/tools.asp

[Yilmaz et al., IEEE Computer 2014]

Designing a Load Test

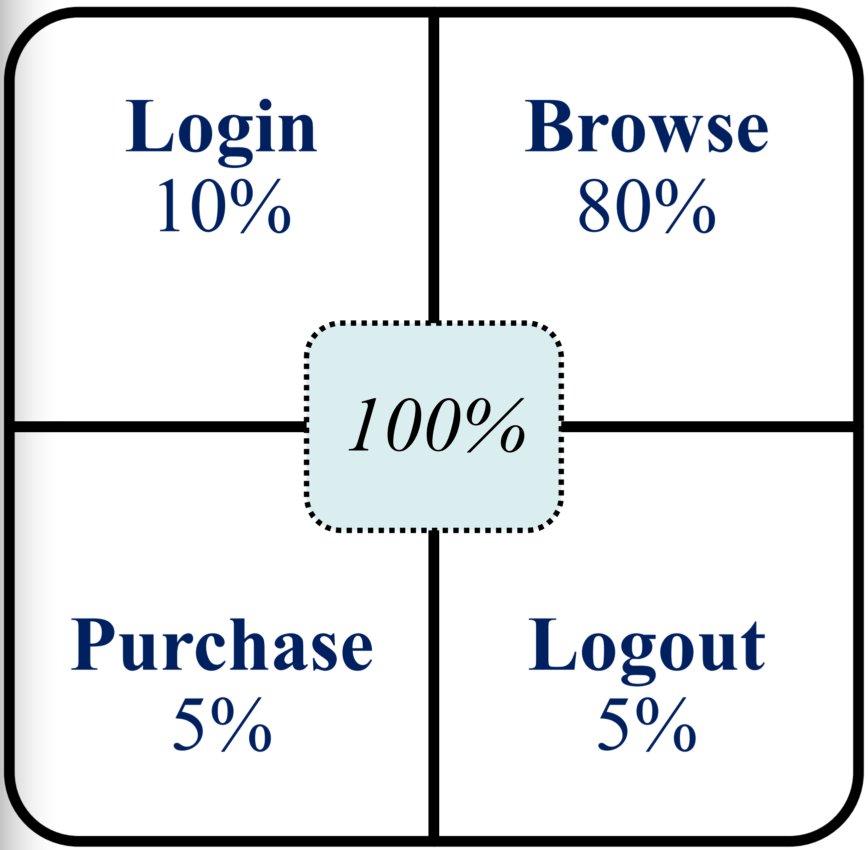
Designing a Load Test



Designing Realistic Loads

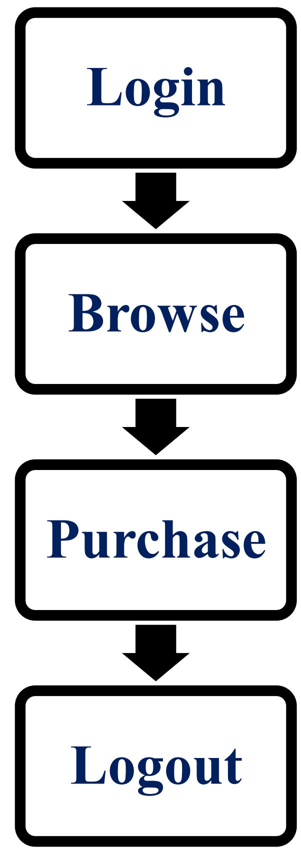
An E-Commerce System

Aggregate Workload



Steady Load, Step-wise load, Extrapolated load

Use-Case



Load Derived from UML, Markov and Stochastic Form-oriented Models

Characterizing an Aggregate Workload

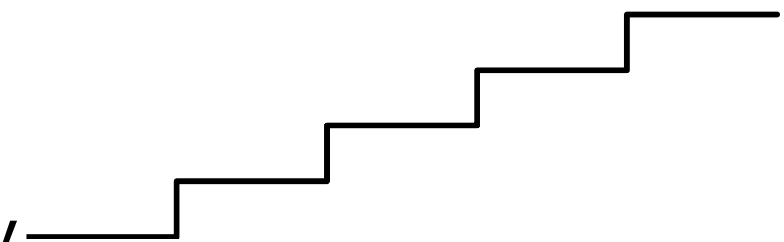
- Workload Mix
  - browsing (30%), purchasing (10%) and searching (60%)
- Workload Intensity
  - Rate of requests (5 requests/sec)

Aggregate Workload (1)

- Steady Load
  - Ease of measurement
  - Memory leaks?

[Bondi, CMG 2007]

- Step-wise Load
  - Same workload mix
  - Different workload intensity



[Hayes, CMG 2000]

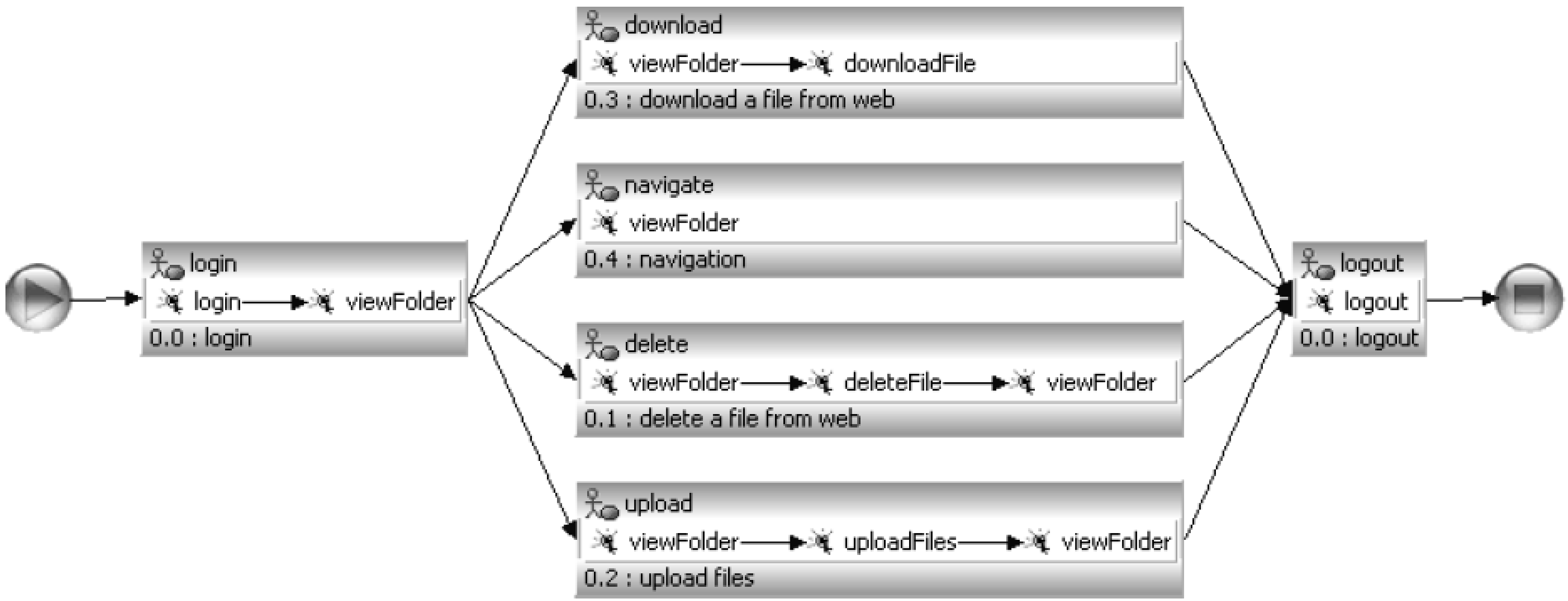
Derived the testing loads from historic data

Aggregate Workload (2)

- In case of missing past usage data, testing loads can be extrapolated from the following sources:
  - Beta-usage data
  - Interviews with domain experts
  - Competitors' data

[Barber, WSE 2004]

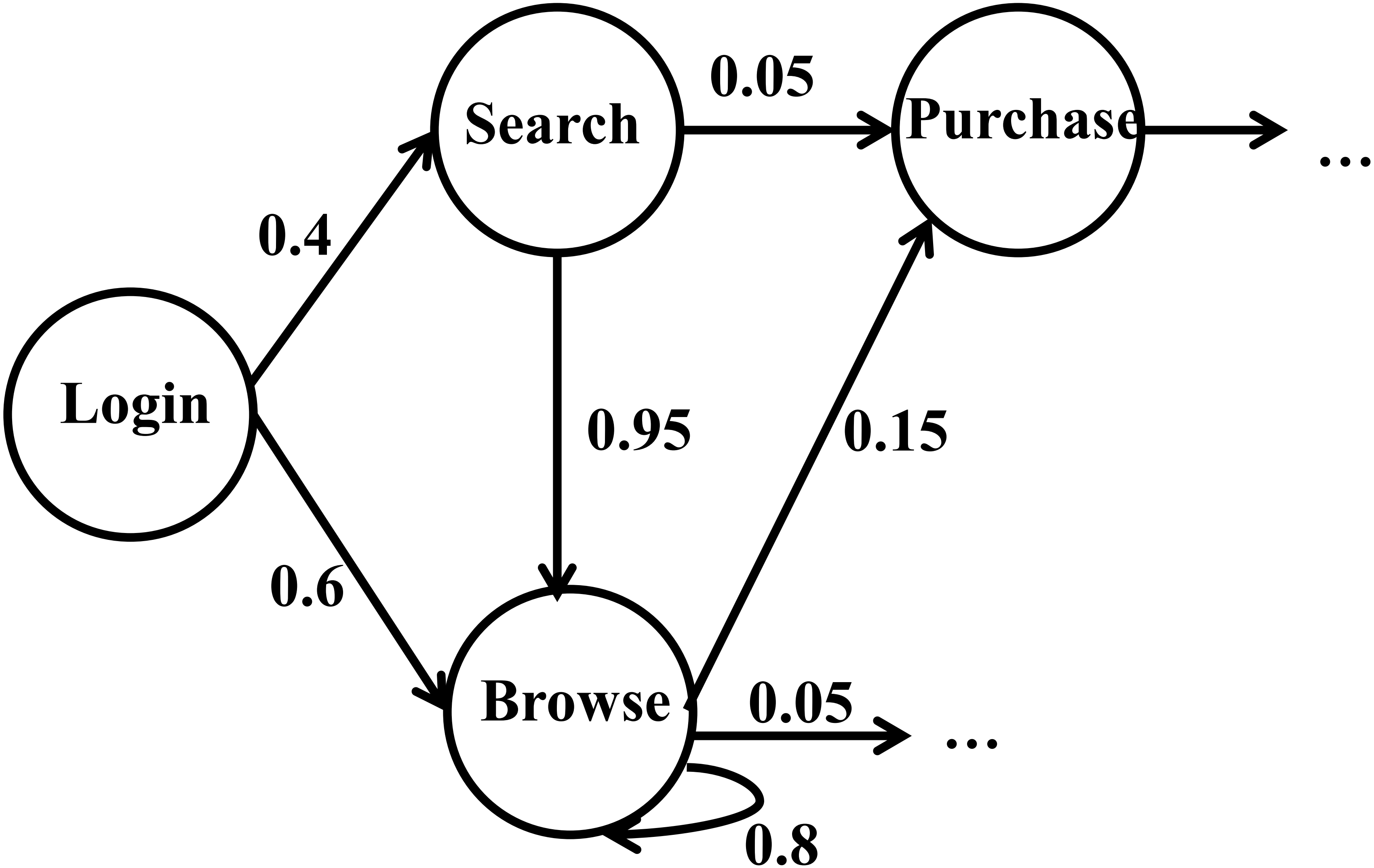
Use-Case (1)
- UML Diagrams



The RUG (Realistic Usage Model)
- derived based on UML use case diagrams

[Wang, ISPA 2004]

Use-Case (2) - Markov Chain



Use-Case (2)
- Markov Chain

```
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=ANTHONY%20
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=11&browse_actor=&brow
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dslogin.jsp?username=user41&password=password HTTP/1.1" 200 2539 16
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=WILLIAM%20
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=15&browse_actor=&brow
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=HILARY%20G
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=6&browse_actor=&brows
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_ti
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=5961&item=646&quan=3&ite
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=41&item=4544&quan=1&item
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dslogin.jsp?username=user3614&password=password HTTP/1.1" 200 728 6
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_ti
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=ELLEN%20GA
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=9&browse_actor=&brows
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=ANGELINA%2
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=JULIA%20TA
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=3614&item=4717&quan=2&it
192.168.0.1 - [22/Apr/2014:00:32:31 -0400] "GET /dslogin.jsp?username=user13337&password=password HTTP/1.1" 200 1960 9
192.168.0.1 - [22/Apr/2014:00:32:31 -0400] "GET /dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_ti
192.168.0.1 - [22/Apr/2014:00:32:31 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=13337&item=322&quan=2&it
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dslogin.jsp?username=user5414&password=password HTTP/1.1" 200 2579 10
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=GRACE%20BR
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=5414&item=198&quan=3&ite
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dsnewcustomer.jsp?firstname=RHVSQS&lastname=EBFMQDBVNM&address1=909823
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_ti
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=20001&item=7868&quan=3&i
192.168.0.1 - [22/Apr/2014:00:32:36 -0400] "GET /dslogin.jsp?username=user13713&password=password HTTP/1.1" 200 729 6
192.168.0.1 - [22/Apr/2014:00:32:36 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=9&browse_actor=&brows
192.168.0.1 - [22/Apr/2014:00:32:36 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=13713&item=493&quan=3&it
192.168.0.1 - [22/Apr/2014:00:32:41 -0400] "GET /dslogin.jsp?username=user9011&password=password HTTP/1.1" 200 728 6
```

web access logs for the past few months

Use-Case (2)
- Markov Chain

192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_title=HOLY%20AUTUMN&limit_num=8&customerid=41 HTTP/1.1" 200 4073 10

192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=5961&item=646&quan=3&item=2551&quan=1&item=45&quan=3&item=9700&quan=2&item=1566&quan=3&item=4509&quan=3&item=5940&quan=2 HTTP/1.1" 200 3049 177

192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=41&item=4544&quan=1&item=6970&quan=3&item=5237&quan=2&item=650&quan=1&item=2449&quan=1 HTTP/1.1" 200 2515 113

Web Access Logs

Use-Case (2)
- Markov Chain

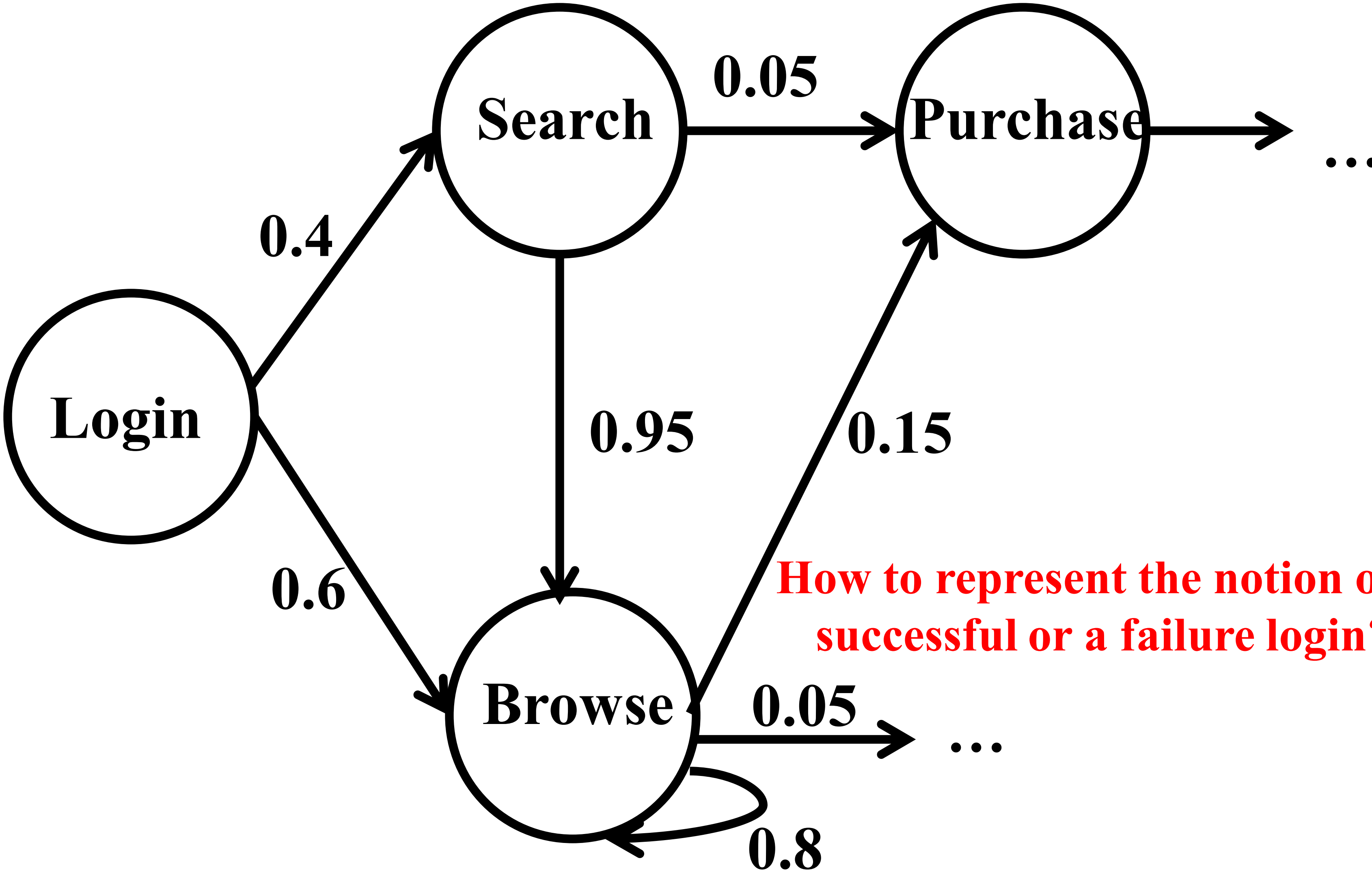
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /**dsbrowse.jsp**?browsetype=title&browse_category=&browse_actor=&browse_title=HOLY%20AUTUMN&limit_num=8&**customerid=41** HTTP/1.1" 200 4073 10

192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /**dspurchase.jsp**?confirmpurchase=yes&**customerid=5961**&item=646&quan=3&item=2551&quan=1&item=45&quan=3&item=9700&quan=2&item=1566&quan=3&item=4509&quan=3&item=5940&quan=2 HTTP/1.1" 200 3049 177

192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /**dspurchase.jsp**?confirmpurchase=yes&**customerid=41**&item=4544&quan=1&item=6970&quan=3&item=5237&quan=2&item=650&quan=1&item=2449&quan=1 HTTP/1.1" 200 2515 113

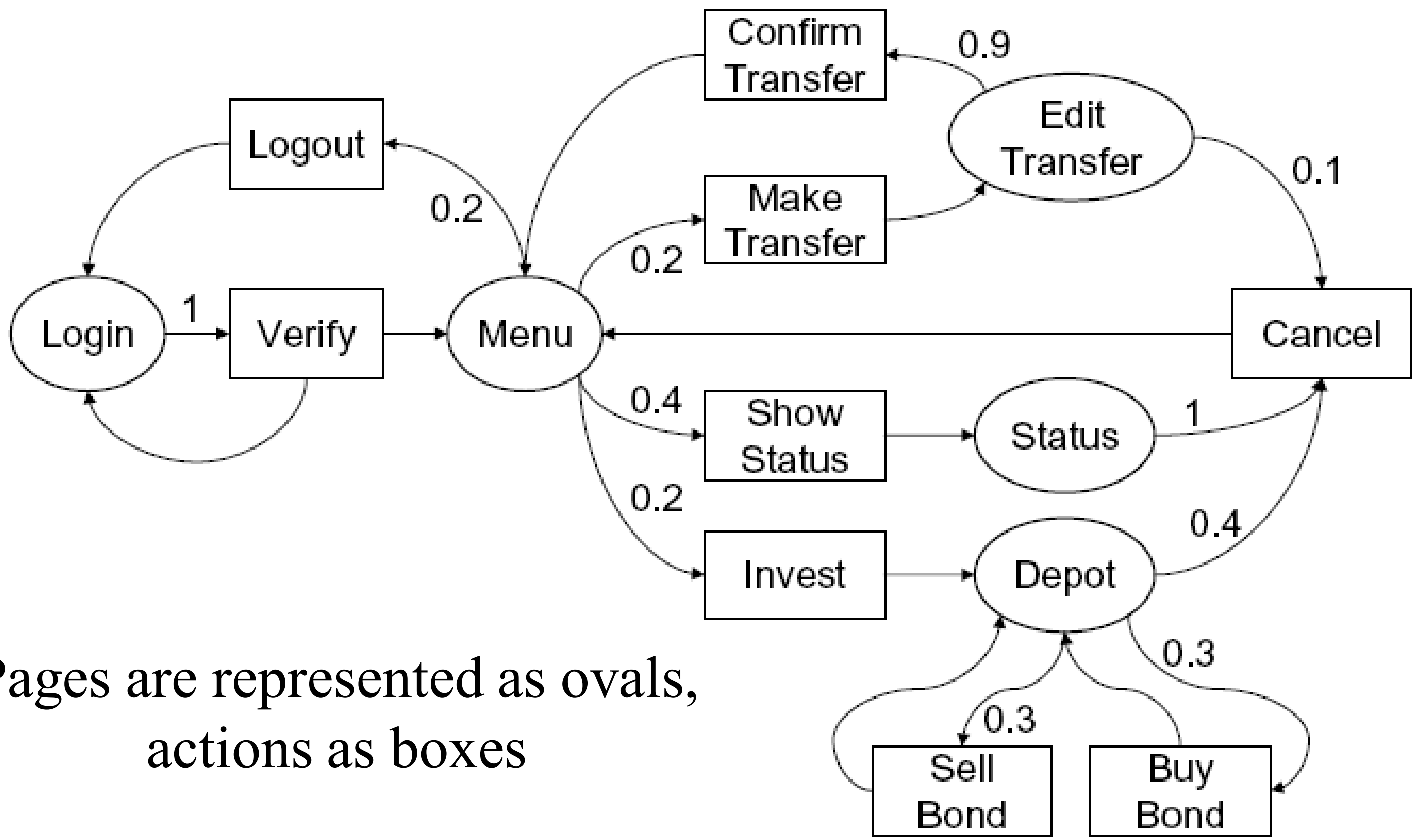
For customer 41: browse -> purchase

Use-Case (2)
- Markov Chain



How to represent the notion of a successful or a failure login?

Use-Case (3)
- Stochastic Form-Oriented Model



Pages are represented as ovals, actions as boxes

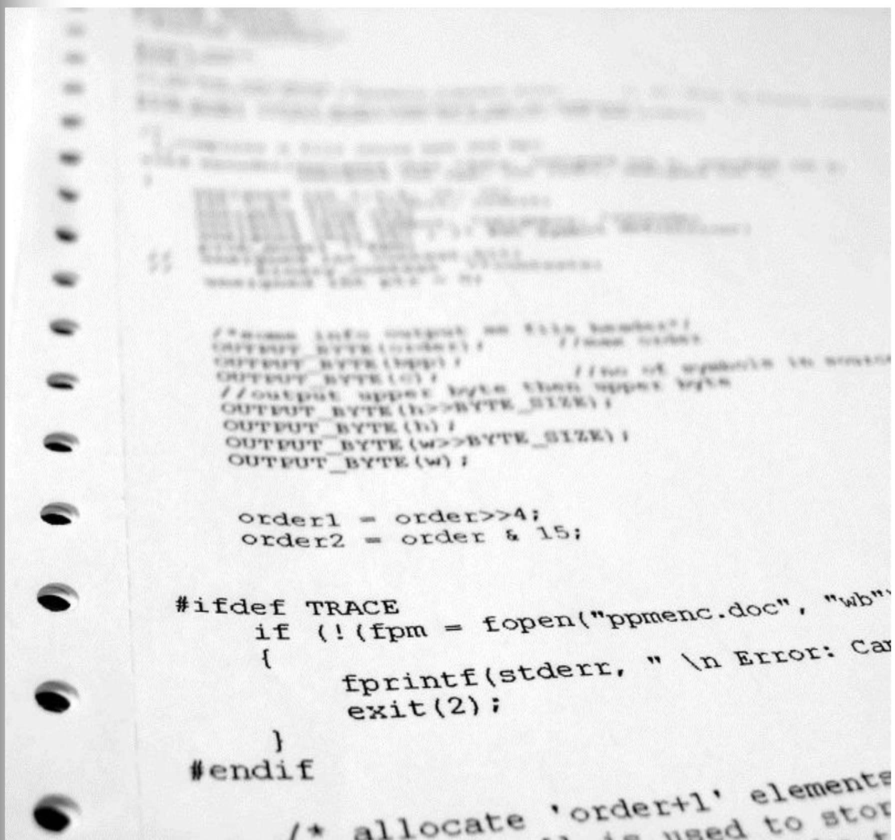
[Cai et al., ASE 2004]

Designing a Load Test	
Designing Realistic Loads	Designing Fault-Inducing Loads
Load Design Optimizations and Reductions	

Designing Fault-Inducing Loads

Source Code

System Models



Code Analysis

(e.g., data flow analysis or symbolic execution)

Model Formulation

(e.g., linear programing and genetic algorithm)

Source Code Analysis (1) - Data Flow Analysis

- Identifies potential load sensitive modules and regions for load sensitive faults (e.g., memory leaks and incorrect dynamic memory allocation
  - Annotating the Control Flow Graph of malloc()/free() calls and their sizes
  - **Load Sensitivity Index (LSI)** indicates the net increase/decrease of heap space used for each iteration: the difference of the heap size before/after each iteration
- Write test cases which exercise the code regions with high LSI values

[Yang el al., ISSTA 1996]

Source Code Analysis (2) - Symbolic Executions

```
y = x;
if (y > 0) then y++;
return y;
```

Two path conditions:

- $x > 0$
- $x \leq 0$

Path Performance Estimation (Response Time)

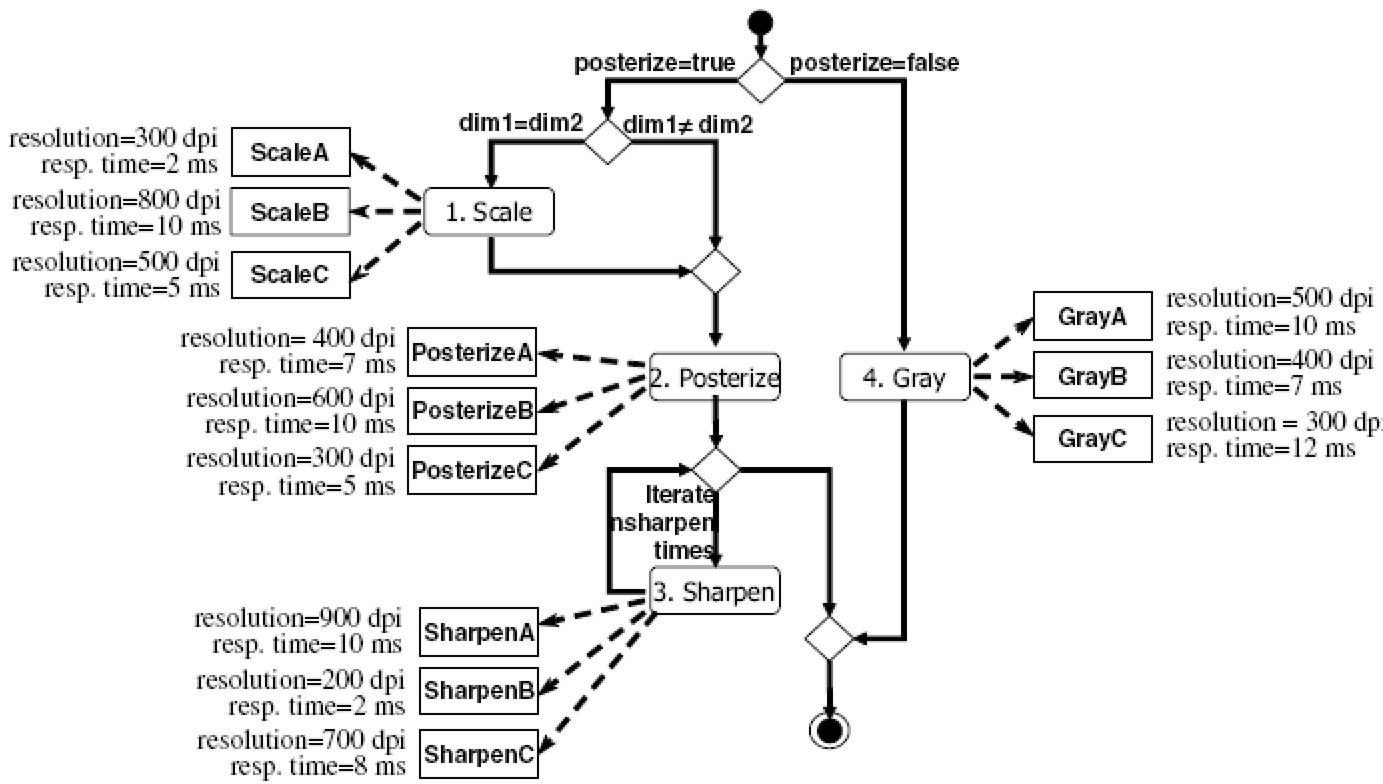
- Weight of 10 for invoking bytecode
- Weight of 1 for all other methods

Memory Analysis

- Uses Java PathFinder's built-in object lifecycle listener mechanism to track the heap size of each path

[Zhang et al., ASE 2011]

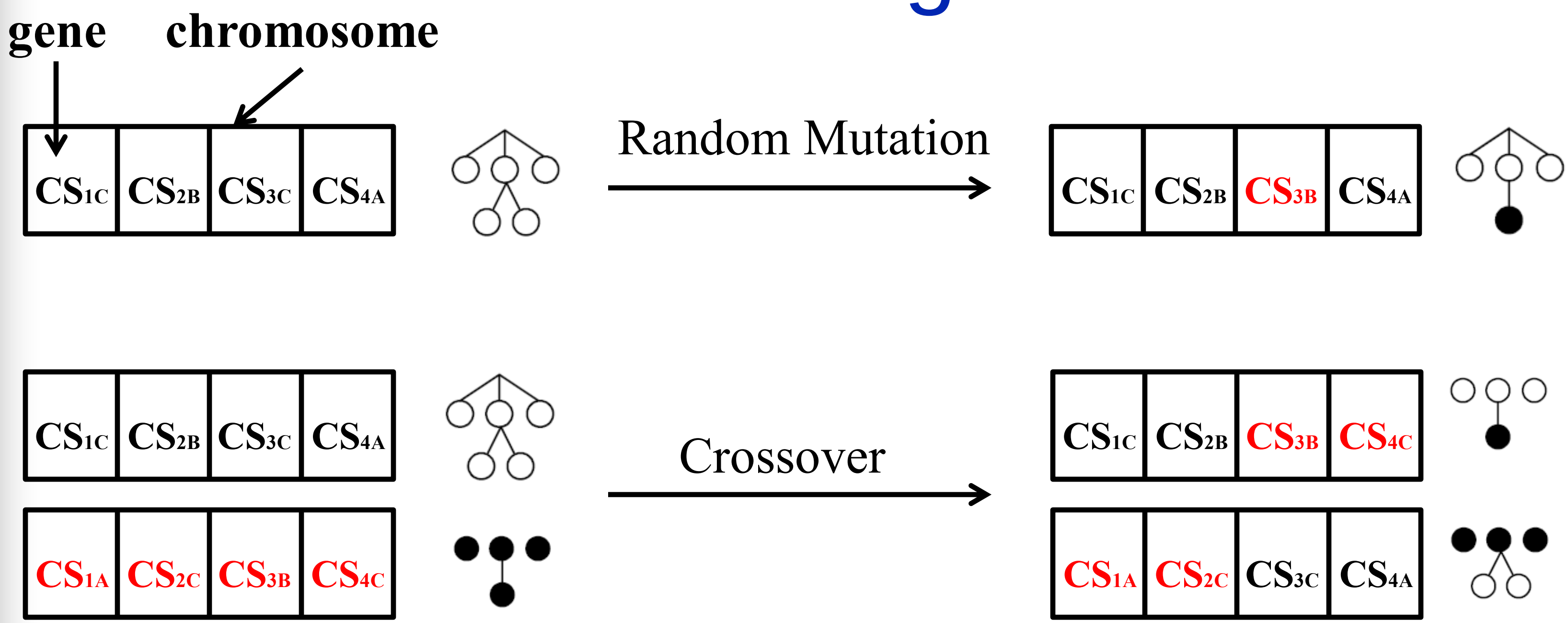
System Models
- Genetic Algorithms



A Service-Oriented Architecture (SOA) Example
- An image processing composite web service

[Penta et al., GECCO 2007]

System Models
- Genetic Algorithms



Fitness function: *How good is the solution?*

<u>Genetic Algorithms applied to SOA</u>

- Each gene: a particular type of web service
- A chromosome: the resulting workflow
- The fitness function: the risky workflow with high response time (SLA violation)

[Penta et al., GECCO 2007]

Load Design Reductions - Extrapolation

- **Question**: Can we reduce the load testing effort and costs, when there is limited time and hardware/software resources?
- Extrapolation for step-wise load testing
  - Only examine a few load levels
  - Extrapolate the system performance for the other load levels

[Menasce et al., CMG 2002]

Load Design Reductions
- Probability Mass Coverage

Sample states for a telecom system:

(2, 3, 0, 1, 5)
- 2 active calls
- 3 leaving voice mail
- 0 updating profile
- 1 checking status
- 5 accessing voice mail

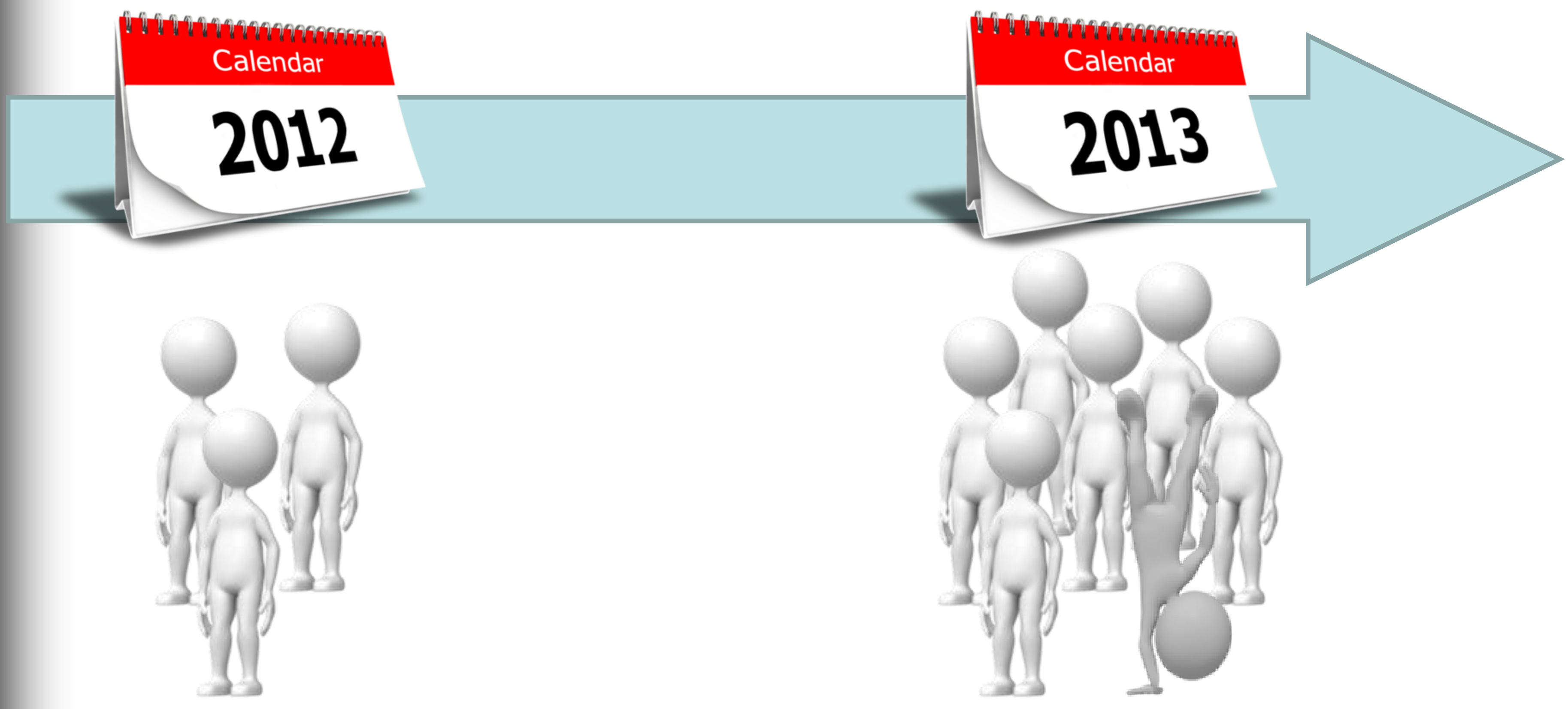
(0, 0, 0, 0, 0)
- Idle state

Probability	# of States
0.3	34
0.4	51
0.5	72
0.6	99
0.7	137
0.8	206
0.9	347
0.99	721
0.999	843
1.0	857

Test Coverage

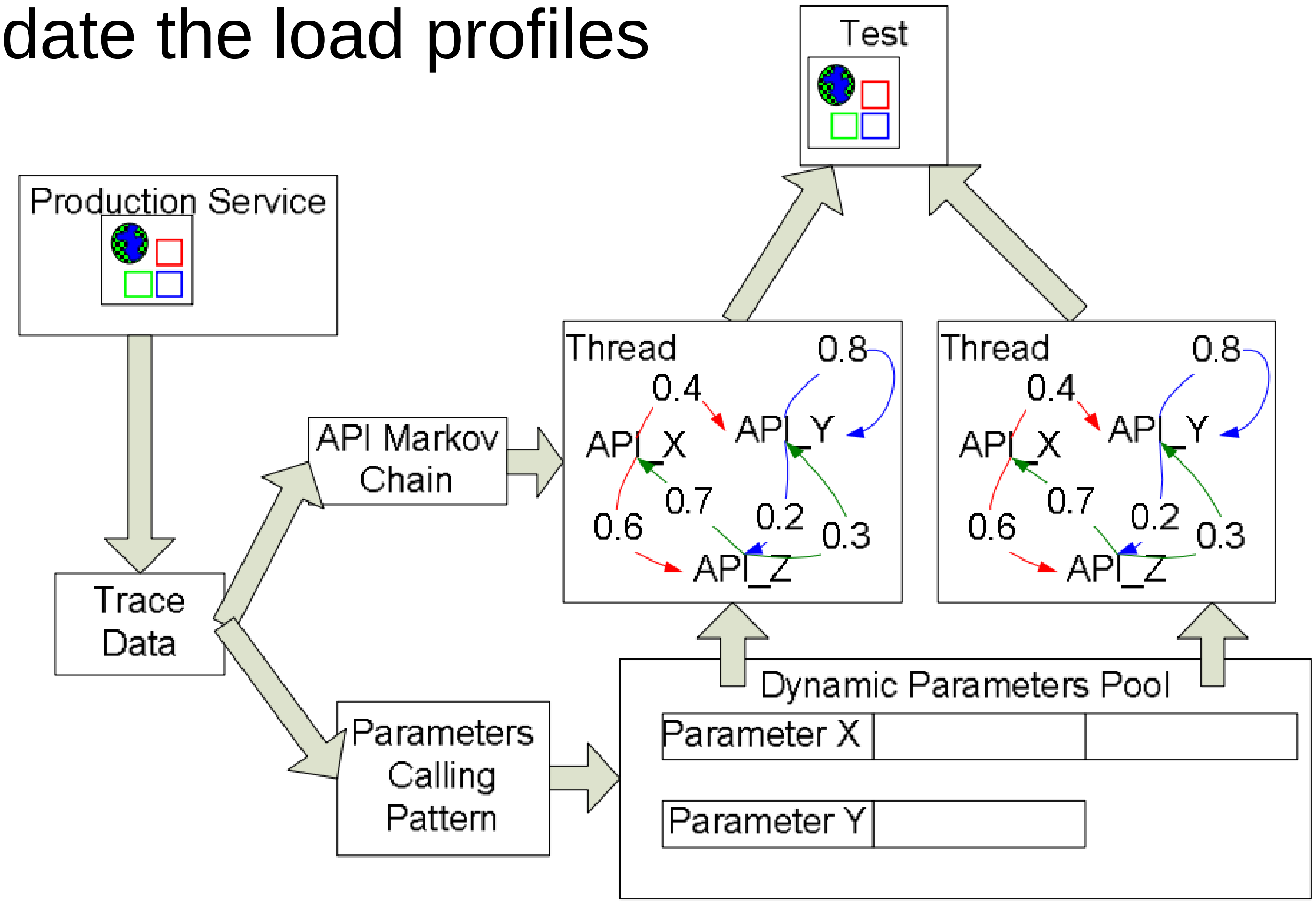
[Avritzer et al., IEEE SW 1996]

Realistic load tests are based on (historical) field workloads, but field workloads change over time



Load Profile Evolution and Maintenance

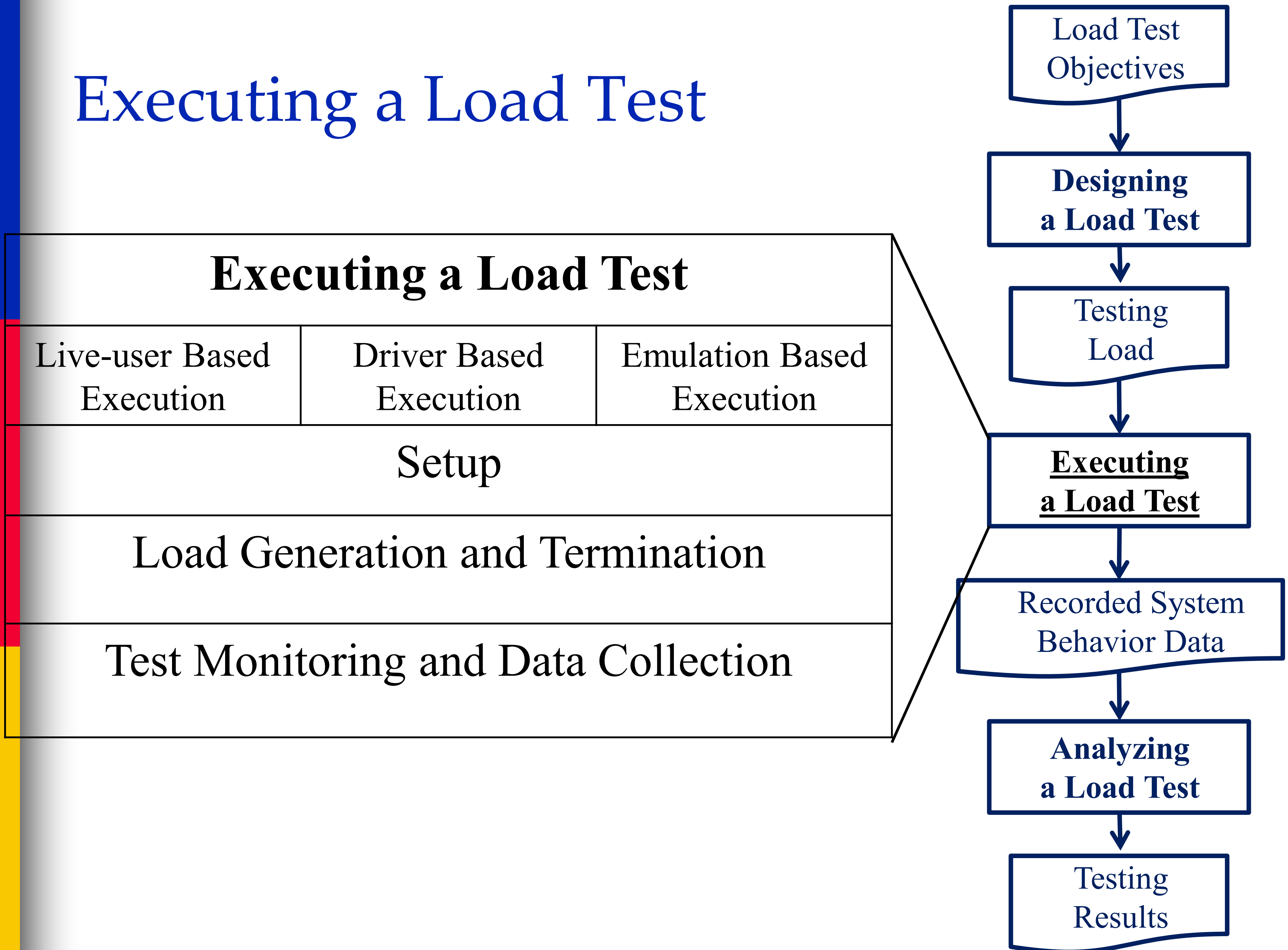
- Field workload can evolve over time. Hence, load testing practitioners need to periodically update the load profiles



[Barros et al., DSN 2007]

Executing a Load Test

Executing a Load Test



Live-user Based Test Execution

- Reflects realistic user behavior
- Obtain real user feedbacks on acceptable performance and functional correctness

- Hard to scale (e.g., limited testing time)
- Limited test complexity due to manual coordination

- Coordinated live-user testing
- Users are selected based on different testing criteria (e.g., locations, browser types, etc.)

Driver-based Test Execution

- 👍 Easy to automate
- 👍 Scale to large number of requests

- 👎 Load driver configurations
- 👎 Hard to track some system behavior (e.g., audio quality or image display)

- Specialized Benchmarking tools (e.g., LoadGen)
- Centralized Load Drivers (e.g, LoadRunner, WebLoad)
  - Easy to control load, but hard to scale (limited to a machine's memory)
- Peer-to-peer Load Drivers (e.g., JMeter, PeerUnit)
  - Easy to scale, but hard to control load

Emulation-based Test Execution

- Special platforms enabling early and continuous verification of system behavior under load

[Hill et al., IEEE SW 2010]

- Special platforms enabling deterministic execution and replay

[Musuvathi et al., OSDI 2008]

Three General Aspects When Executing a Load Test

Test Setup

- System Deployment
- Test Execution Setup

Load Generation and Termination

- Static Configuration
- Dynamic Feedback
- Deterministic

Test Monitoring and Data Collection

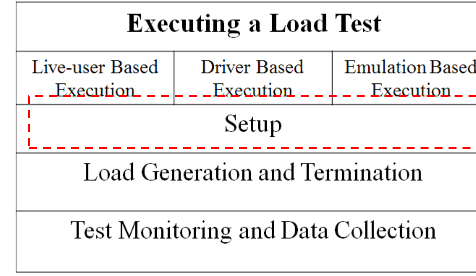
- Metrics and Logs

Executing a Load Test		
Live-user Based Execution	Driver Based Execution	Emulation Based Execution
Setup		
Load Generation and Termination		
Test Monitoring and Data Collection		

System Deployment for Live-user and Driver-based Executions

- Field load testing
  - Costly but realistic
- Selection of hardware
  - Dedicated hardware, or
  - Cloud-based testing
- Creating realistic databases
  - Importing realistic raw data
  - Sanitizing field database
- Mimicking realistic network traffic
  - Network latency
  - Network spoofing
- Do not deploy drivers on the same machines with the SUT

System Deployment for Emulation-based Executions



- For continuous performance evaluation:
  - Automated Code Generations for Incomplete System Components via a Model Interpreter

[Hill et al., ECBS 2008]

- For deterministic executions:
  - Deploy on the CHESS platform

[Musuvathi et al., OSDI 2008]

Executing a Load Test		
Live-user Based Execution	Driver Based Execution	Emulation Based Execution
Setup		
Load Generation and Termination		
Test Monitoring and Data Collection		

Test Execution Setup

- Live-user-based executions
  – Tester recruitment, setup and training
- Driver-based executions
  – Programming
  – Store-and-replay configuration
  – Model configurations
- Emulation-based executions
  – Write your own load driver

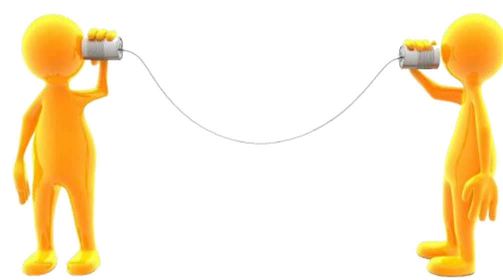
Executing a Load Test		
Live-user Based Execution	Driver Based Execution	Emulation Based Execution
Setup		
Load Generation and Termination		
Test Monitoring and Data Collection		

Load Generation and Termination

Static Configuration

- Timer-Driven
- Counter-Driven
- Statistic-Driven

Dynamic Feedback



- Dynamically steer the testing loads based on system feedback

Deterministic

- Systematically execute all the possible inter-leavings

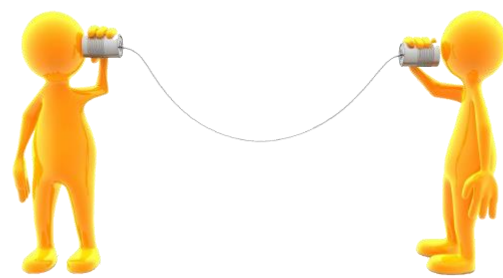
Executing a Load Test

Live-user Based Execution	Driver Based Execution	Emulation Based Execution
Setup		
Load Generation and Termination		
Test Monitoring and Data Collection		

Load Generation and Termination

Static Configuration

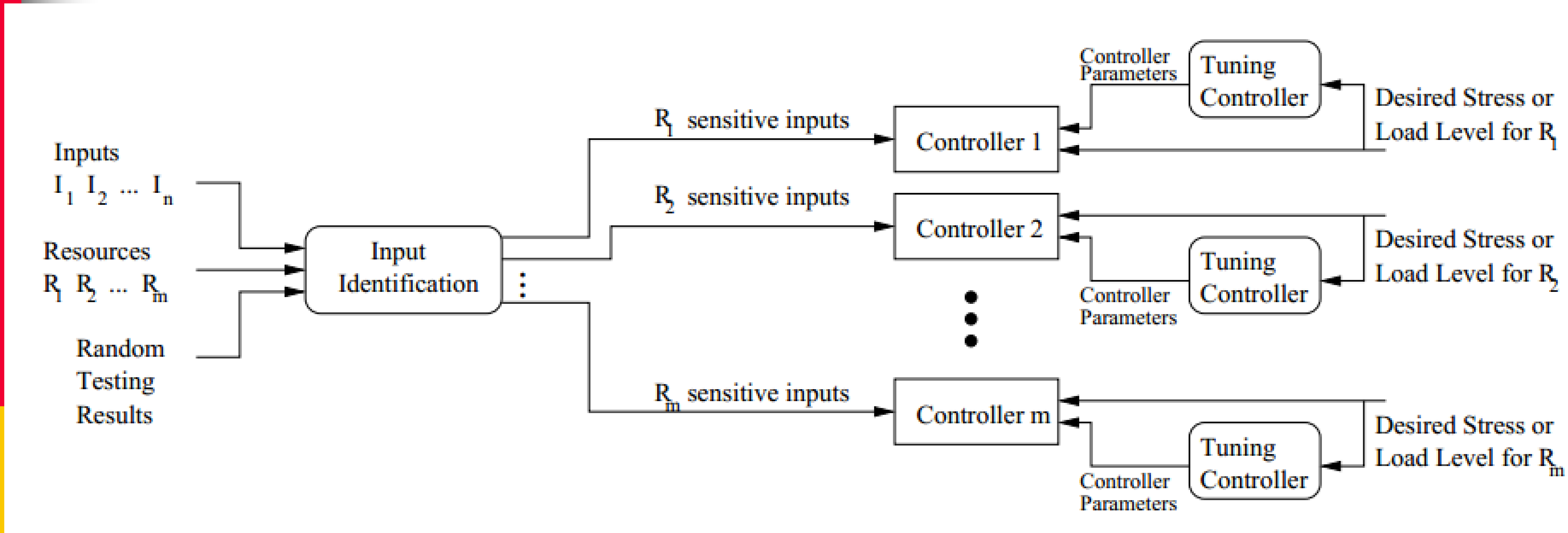
Dynamic Feedback



Deterministic

	Live-user Based	Driver Based	Emulation Based
Static	✓	✓	✓
Dynamic	✗	✓	✗
Deterministic	✗	✗	✓

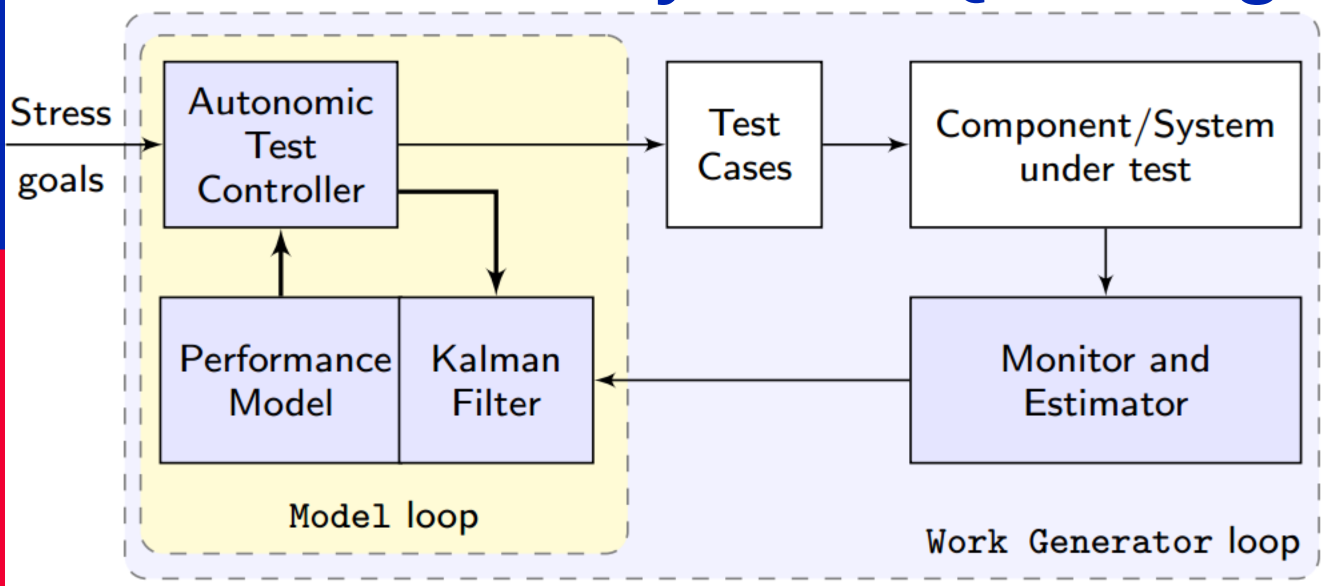
Dynamic Feedback (1)
- System Identification Techniques



Start with random testing to identify performance sensitive input

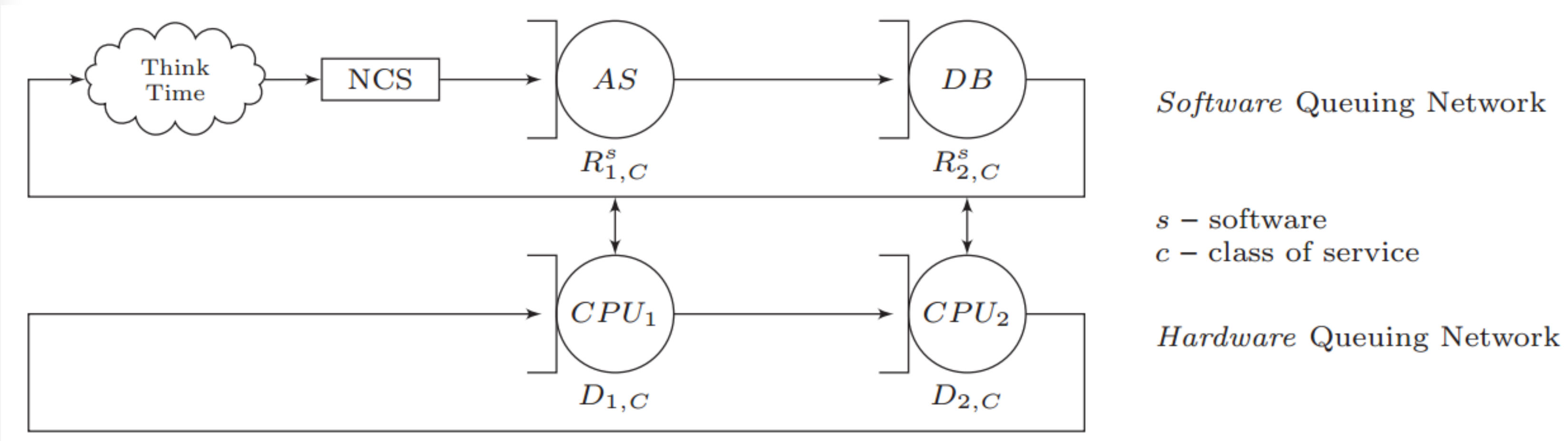
[Bayan et al., SAC 2008]

Dynamic Feedback (2)
- Two-Layered Queuing Models



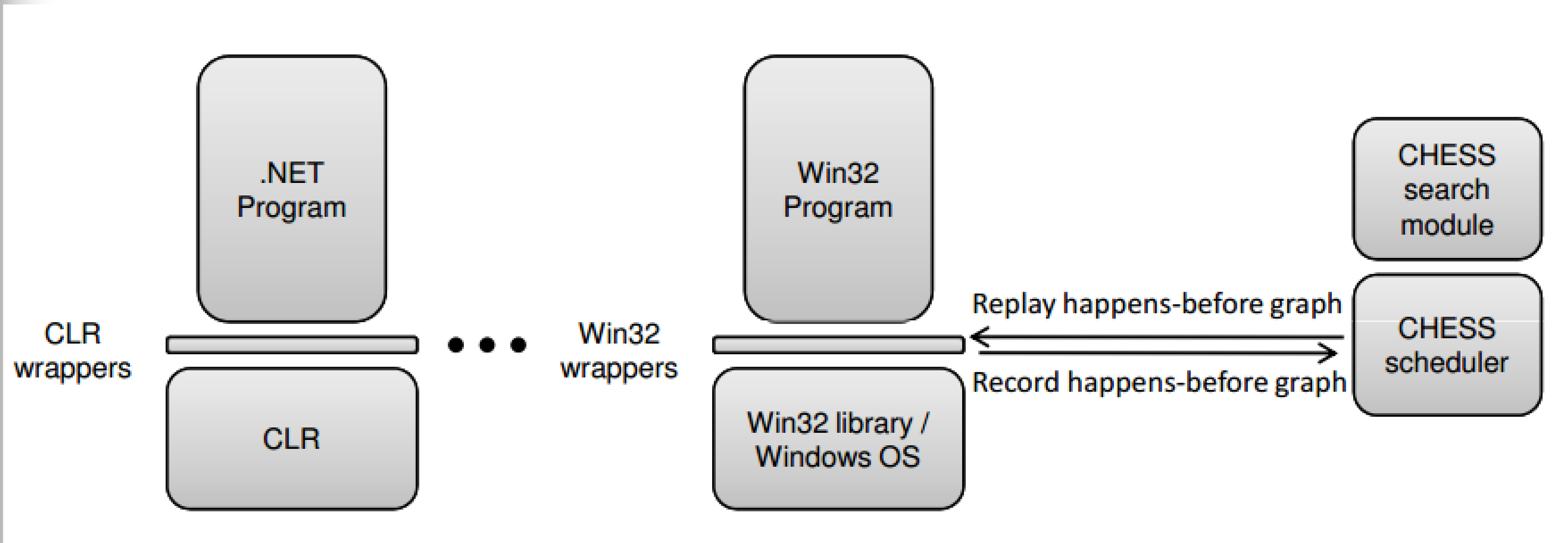
A <u>stress goal</u> is target perf. metric threshold:
- a sw/hw utilization,
- a target response time or
- throughput

for a class of request



[Barna et al., ICAC 2011]

Deterministic Load Execution

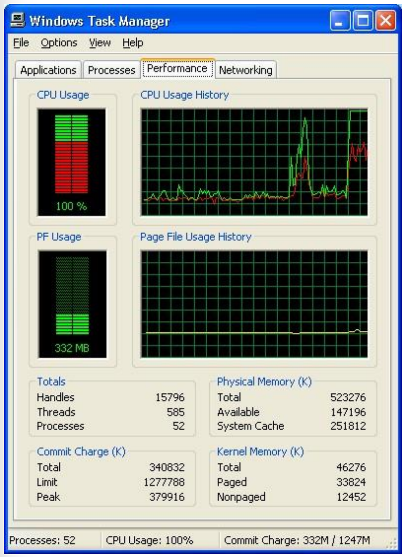


Implemented a wrapper layer via binary instrumentation, between the program & the concurrency API

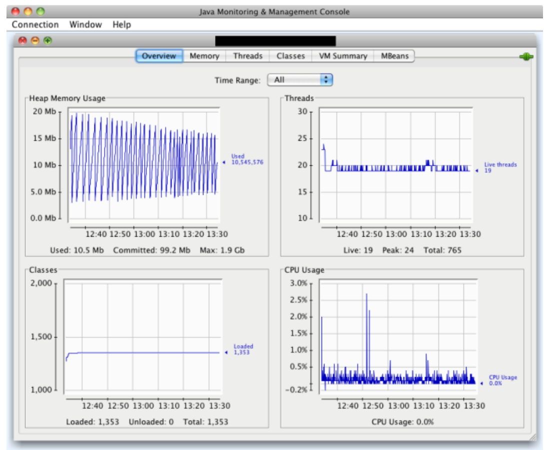
[Musuvathi et al., OSDI 2008]

Executing a Load Test		
Live-user Based Execution	Driver Based Execution	Emulation Based Execution
Setup		
Load Generation and Termination		
Test Monitoring and Data Collection		

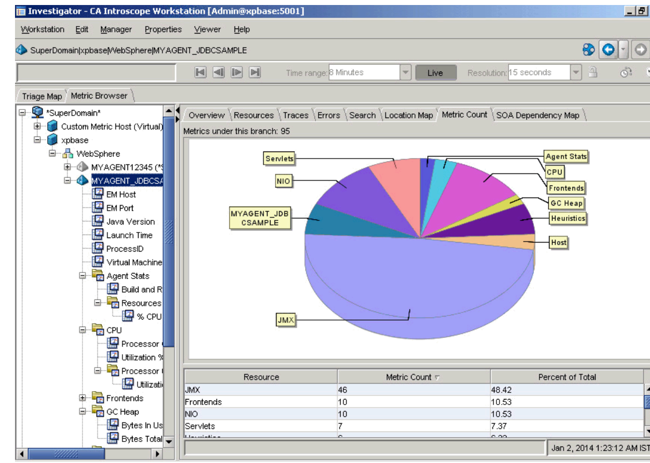
Test Monitoring Tools



Task Manager

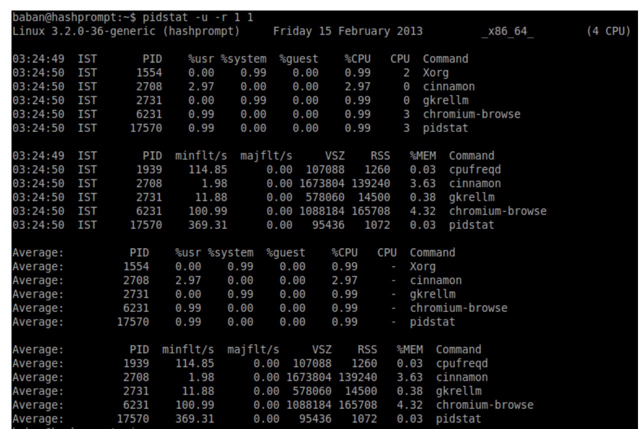


JConsole



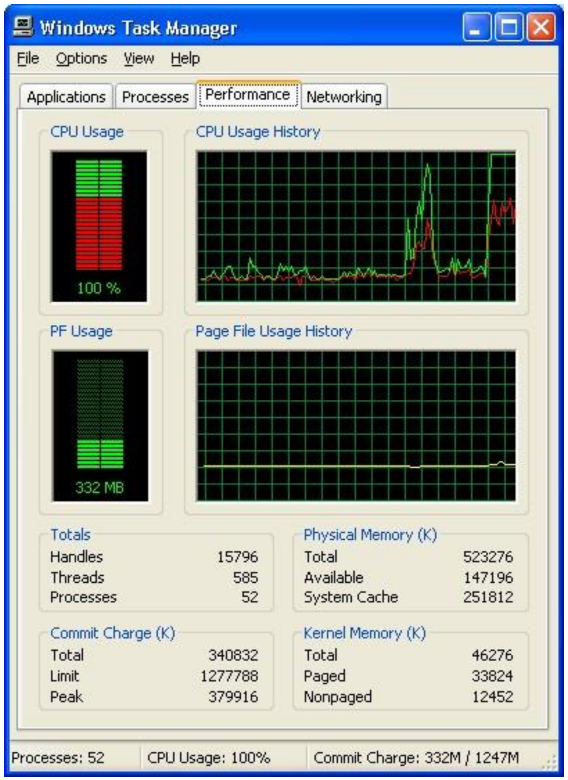
CA Willy

App Dynamics

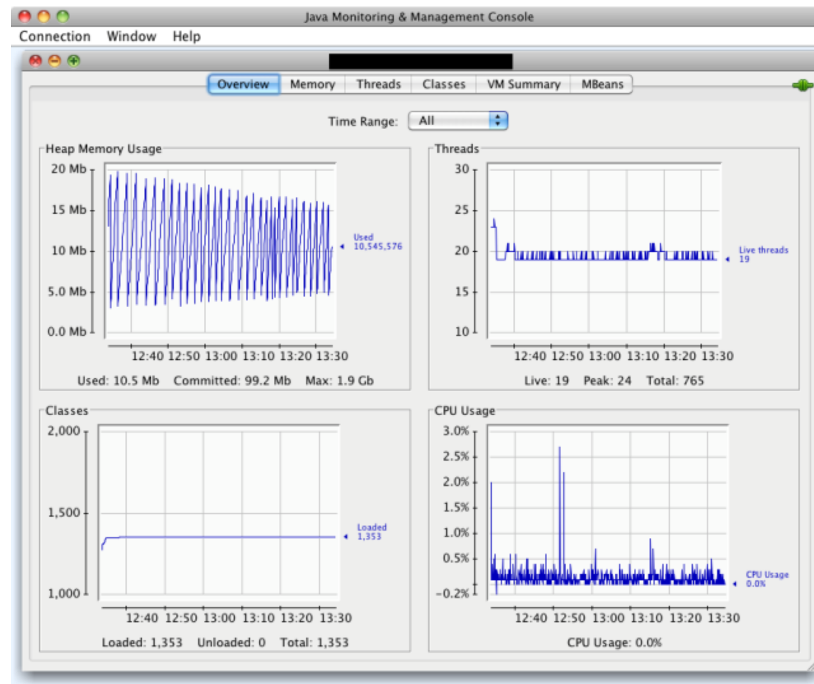


pidstat

Agent-less Monitoring Examples



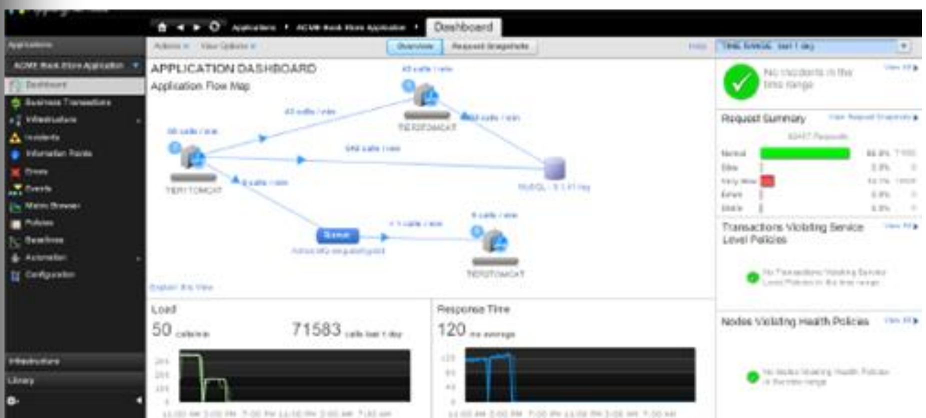
Task Manager

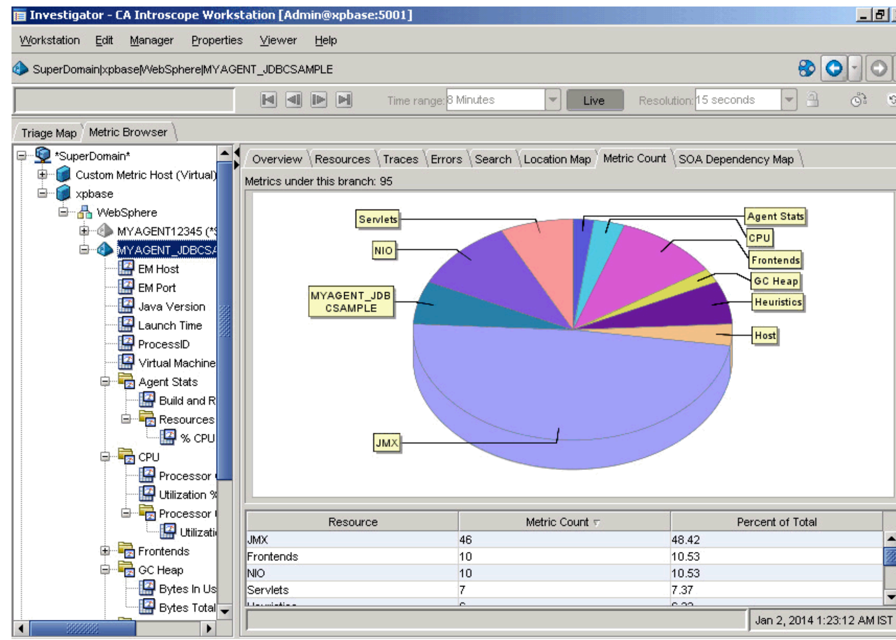


JConsole

PerfMon (Windows), sysstat (Linux), top

Agent-based Monitoring Examples





App Dynamics **CA Willy**

Dell FogLight, New Relic

Instrumentation

- Source code level instrumentation
  - Ad-hoc manual instrumentation,
  - Automated instrumentation (e.g., AspectJ), and
  - Performance instrumentation framework (e.g., the Application Response Time API)
- Binary instrumentation framework
  - DynInst (http://www.dyninst.org/),
  - PIN (https://software.intel.com/en-us/articles/pin-a-binary-instrumentation-tool-downloads/), and
  - Valgrind (http://valgrind.org/)
- Java Bytecode instrumentation framework
  - Ernst's ASE 05 tutorial on "Learning from executions: Dynamic analysis for software engineering and program understanding" (http://pag.csail.mit.edu/~mernst/pubs/dynamic-tutorial-ase2005-abstract.html)

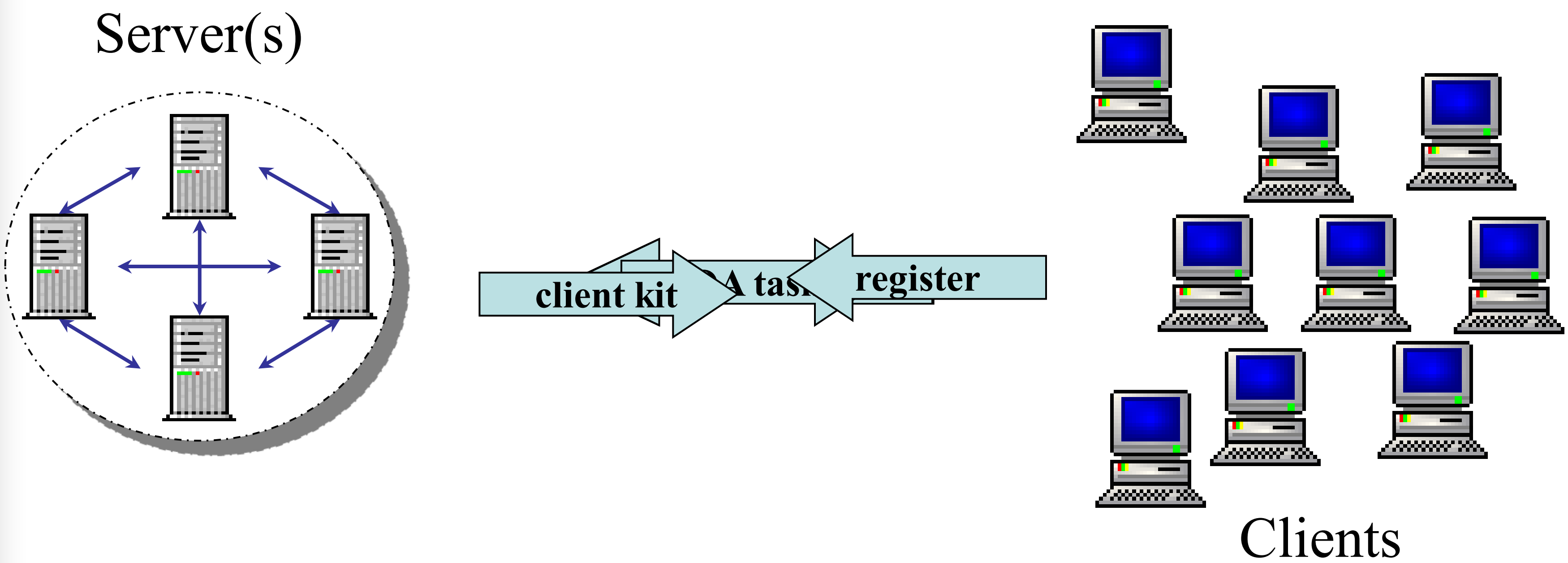
Measurement Bias

- Measurement bias is hard to avoid and unpredictable.
- **<u>Example 1</u>**: How come the same application today runs faster compared with yesterday?
- **<u>Example 2</u>**: Why the response time is very different when running the same binary under different user accounts?
- **<u>Example 3</u>**: Why the code optimization only works on my computer?

  - **Repeated measurement**
  - **Randomize experiment setup**

[Mytkowicz et al., ASPLOS 2010]

[Example] Skoll – A Distributed Continuous Quality Assurance (DCQA) Infrastructure

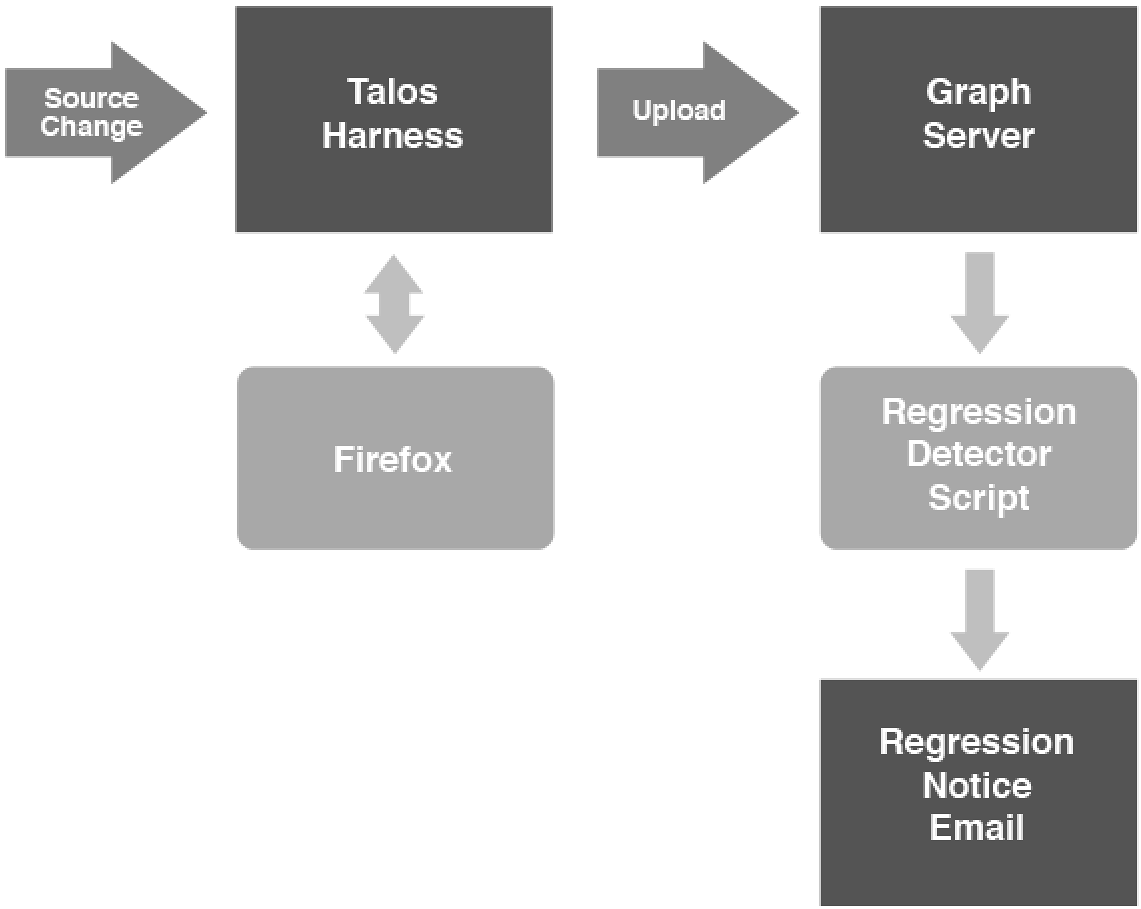


- Server selects best task that matches client characteristics

Performance Regression Testing under different configurations

[Memon et al., ICSE 2004]

[Example] Talos - Mozilla Performance Regression Testing Framework

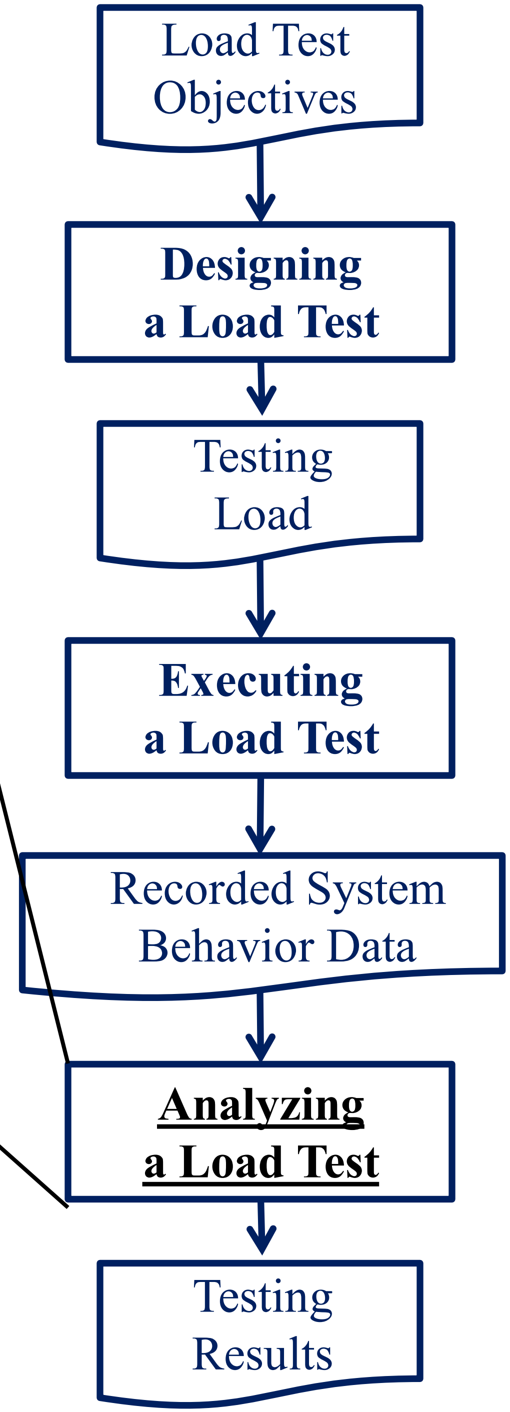


[Talbert et al., http://aosabook.org/en/posa/talos.html]

Analyzing a Load Test

Analyzing a Load Test

Analyzing a Load Test		
Verifying Against Threshold Values	Detecting Known Problems	Automatically Detecting Anomalous Behavior



Sample Counters

	A	B	C	D	E
1	Time	Disk Reads/sec	Disk Writes/sec	Page Faults/sec	Memory
2	2/29/08 16:58	0.049986394	0.000723659	0.003876542	3534848
3	2/29/08 17:01	0	0	0	3534848
4	2/29/08 17:04	0.060612225	0.027551011	0.016530607	3534848
5	2/29/08 17:07	0	0	0	3534848
6	2/29/08 17:10	0	0	0	3534848
7	2/29/08 17:13	0.060733302	0.027606046	0.016563628	3534848
8	2/29/08 17:16	0	0	0	3534848
9	2/29/08 17:19	0.060727442	0.027603383	0.01656203	3534848
10	2/29/08 17:22	0	0	0	3534848
11	2/29/08 17:25	0	0	0	3534848
12	2/29/08 17:28	0	0	0	3534848
13	2/29/08 17:31	0	0	0	3534848
14	2/29/08 17:34	0.121368621	0.055167555	0.038617289	3534848
15	2/29/08 17:37	0	0	0	3534848
16	2/29/08 17:40	0	0	0	3534848
17	2/29/08 17:43	0	0	0	3534848
18	2/29/08 17:46	0	0	0	3534848
19	2/29/08 17:49	0	0	0	3534848
20	2/29/08 17:52	0	0	0	3534848
21	2/29/08 17:55	0.121392912	0.055178596	0.033107158	3534848
22	2/29/08 17:58	0.060592703	0.027542138	0.02203371	3534848

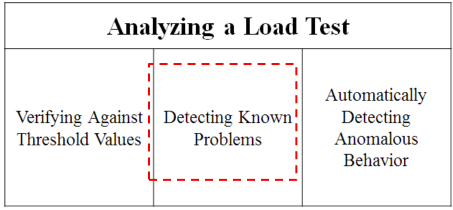
Sample Execution Logs

#	Log Lines
1	time=1, thread=1, session=1, receiving new user registration request
2	time=1, thread=1, session=1, inserting user information to the database
3	time=1, thread=2, session=2, user=Jack, browse catalog=novels
4	time=1, thread=2, session=2, user=Jack, sending search queries to the database
5	time=3, thread=1, session=1, user=Tom, registration completed, sending confirmation email to the user
6	time=3, thread=2, session=2, database connection error: session timeout
7	time=4, thread=1, session=1, fail to send the confirmation email, number of retry = 1
8	time=6, thread=2, session=2, user=Jack, successfully retrieved data from the database
9	time=7, thread=2, system health check
10	time=8, thread=1, session=1, registration email sent successfully to user=Tom
11	time=9, thread=2, session=3, user=Tom, browse catalog=travel
12	time=10, thread=2, session=3, user=Tom, sending search queries to the database
13	time=10, thread=3, session=4, user=Jim, updating user profile
14	time=11, thread=3, session=4, user=Jim, database error: deadlock

Analyzing a Load Test		
Verifying Against Threshold Values	Detecting Known Problems	Automatically Detecting Anomalous Behavior

Verifying Against Threshold Values

- Straight-forward comparison
  - E.g., do the throughput values match with the target?
- Comparison against processed data
  - Maximum
  - Medium or average
  - 90-percentile value
- Comparison against derived data
  - Deriving thresholds
    - What is the response time for previous versions?
  - Deriving target data
    - What will the estimated reliability be?



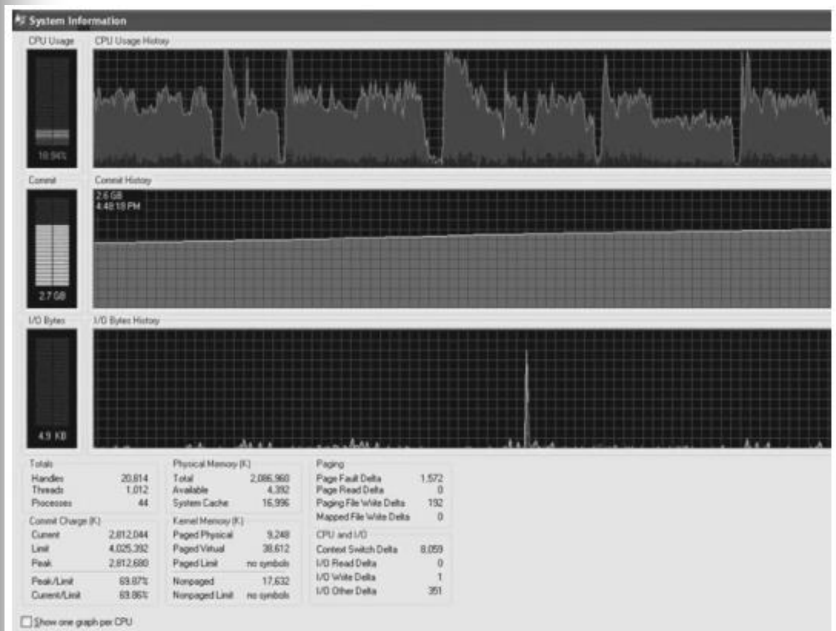
Detecting Known Problems Using Patterns

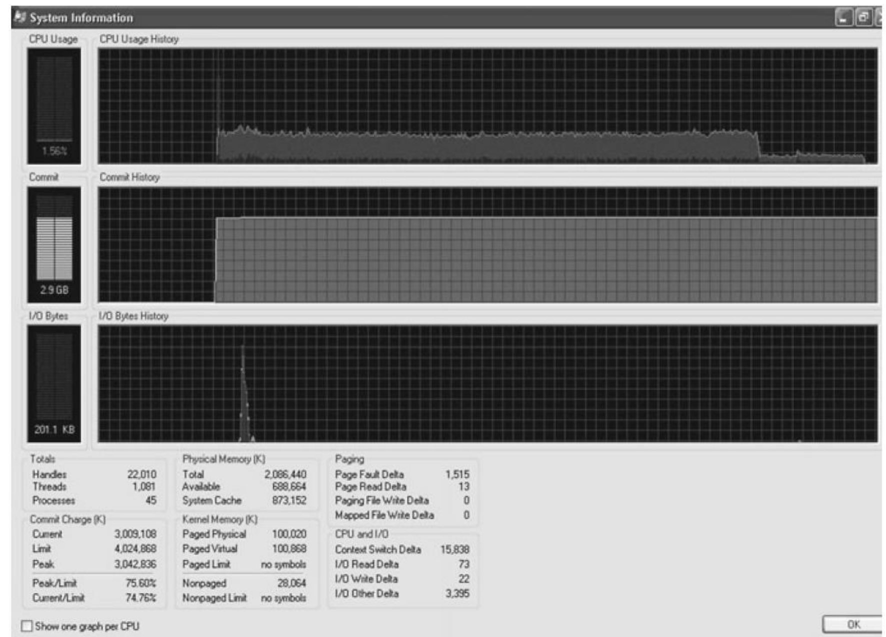
- Patterns in the memory utilizations
  - Memory leak detection
- Patterns in the logs
  - Error keywords

Memory Leaks

Need to wait till system is warmed up (a.k.a., cache filled up)

Deadlocks





Before fix **After fix**

[Avritzer et al., 2012]

Error Keywords

- A large-scale enterprise system can generate 1.6 million log lines in an 8-hour load test
  - 23,000 lines contain “fail” or “failure”
  - How many types of failures are there in this test?

Events	Frequency
Error occurred during purchasing, item=*$v*	500
Error! Cannot retrieve catalogs for user=*$v*	300
Authentication error for user=*$v*	100

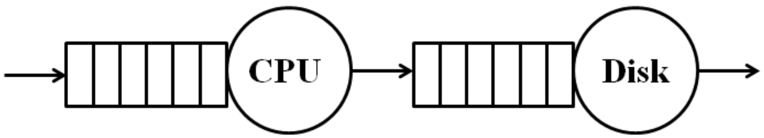
[Jiang et al., JSME 2008]

Analyzing a Load Test		
Verifying Against Threshold Values	Detecting Known Problems	Automatically Detecting Anomalous Behavior

Automated Detection of Anomalous Behavior

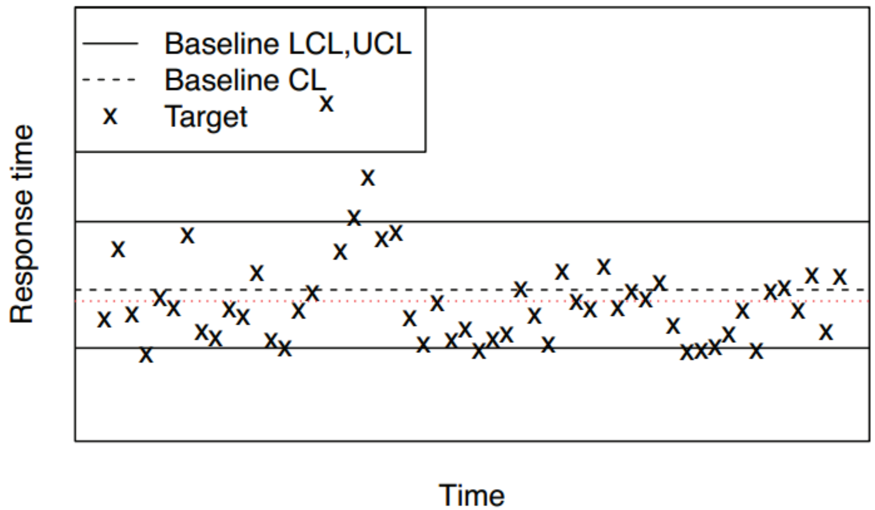
Automatically derive "expect/normal" behavior and flag anomalous behavior

Data Mining

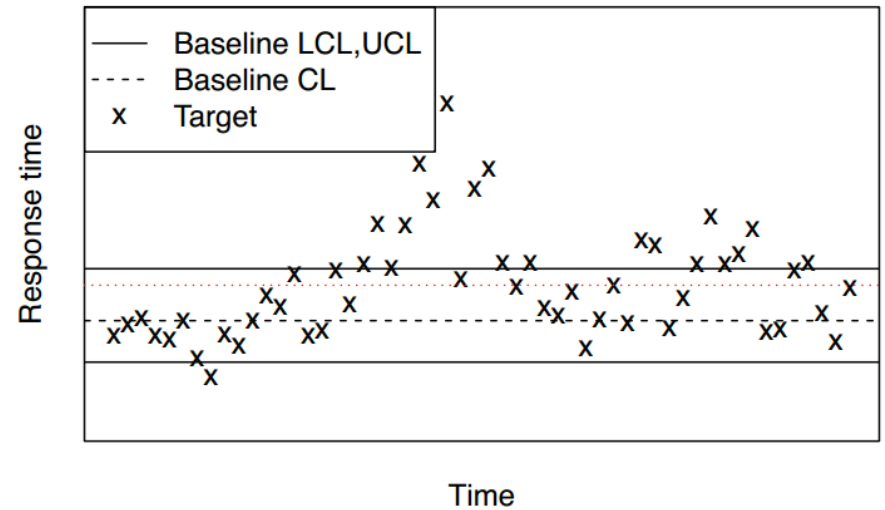


Queuing Theory

Deriving Performance Ranges Using Control Charts



Normal Operations



Suspicious Test

- **Derive control charts from the past good tests**
- **Flag new tests as anomalous if there are many violations in the control charts**

[Nguyen et al., APSEC 2011]

Automated Derivation of Performance Rules

- From past tests, we can extract rules such as:

DB read/sec **Medium** + Throughput **Medium** → Request Queue size **Low**

- **Flags tests where the rule does not hold**

DB read/sec **Medium** + Throughput **Medium** → Request Queue size **High**

[Foo et al., QSIC 2011]

Severity	Performance Regression	Symptoms
0.66	Application Server CPU Utilization	Show Rules
0.60	Application Server Memory Utilization	Show Rules
0.58	Database Disk Read Bytes/sec	Show Rules

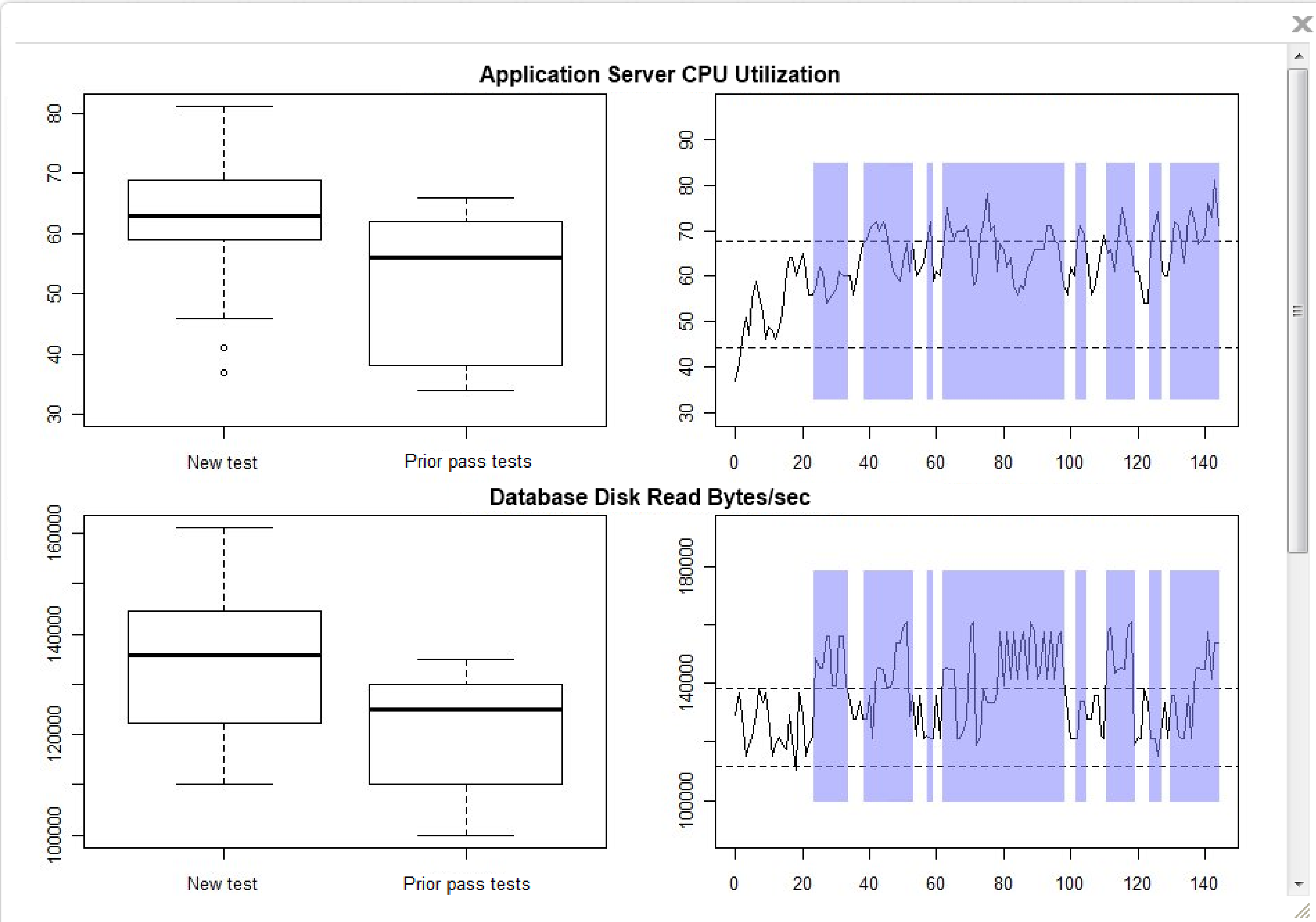
Counter Analysis Report

<table>
<tr><th>Severity</th><th>Performance Regression</th><th>Symptoms</th></tr>
<tr><td>0.66</td><td>Application Server CPU Utilization</td><td>Hide Rules
<table>
<tr><th>Graph</th><th>Conf. Change</th><th>Expected Correlation</th></tr>
<tr><td></td><td>0.79</td><td>Database Logical Disk Reads/sec=Mid
Database Memory Page Writes/sec=Mid
Database CPU Utilization=Mid
Database Memory Page Reads/sec=Mid
Application Server CPU Utilization=Mid(High)
Application Server Memory Utilization=Mid(High)</td></tr>
<tr><td></td><td>0.65</td><td>Database Logical Disk Reads/sec=Mid
Database Memory Page Reads/sec=Mid
Application Server CPU Utilization=Mid(High)
Application Server Memory Utilization=Mid(High)
Database Disk Read Bytes/sec=Mid(High)</td></tr>
</table></td></tr>
<tr><td>0.60</td><td>Application Server Memory Utilization</td><td>Show Rules</td></tr>
<tr><td>0.58</td><td>Database Disk Read Bytes/sec</td><td>Show Rules</td></tr>
</table>

[Foo et al., QSIC 2011]

Severity	Performance Regression	Symptoms
0.66	Application Server CPU Utilization	Show Rules
0.60	Application Server Memory Utilization	Show Rules
0.58	Database Disk Read Bytes/sec	Show Rules

Counter Analysis Report



Severity		
0.66	App	High) Mid(High) High) Mid(High))
0.60	App	
0.58	Data	

[Foo et al., QSIC 2011]

Deriving Performance Signatures Using Statistical Techniques

Performance Counters

	A	B	C	D	E
1	Time	Disk Reads/sec	Disk Writes/sec	Page Faults/sec	Memory
2	2/29/08 16:58	0.049986394	0.000723659	0.003876542	3534848
3	2/29/08 17:01	0	0	0	3534848
4	2/29/08 17:04	0.060612225	0.027551011	0.016530607	3534848
5	2/29/08 17:07	0	0	0	3534848
6	2/29/08 17:10	0	0	0	3534848
7	2/29/08 17:13	0.060733302	0.027606046	0.016563628	3534848
8	2/29/08 17:16	0	0	0	3534848
9	2/29/08 17:19	0.060727442	0.027603383	0.01656203	3534848
10	2/29/08 17:22	0	0	0	3534848
11	2/29/08 17:25	0	0	0	3534848
12	2/29/08 17:28	0	0	0	3534848
13	2/29/08 17:31	0	0	0	3534848
14	2/29/08 17:34	0.121368621	0.055167555	0.038617289	3534848
15	2/29/08 17:37	0	0	0	3534848
16	2/29/08 17:40	0	0	0	3534848
17	2/29/08 17:43	0	0	0	3534848
18	2/29/08 17:46	0	0	0	3534848
19	2/29/08 17:49	0	0	0	3534848
20	2/29/08 17:52	0	0	0	3534848
21	2/29/08 17:55	0.121392912	0.055178596	0.033107158	3534848
22	2/29/08 17:58	0.060592703	0.027542138	0.02203371	3534848

Principle Component Analysis (PCA)

PC-1

PC-2

PC-3

Performance Signature

[Malik et al., CSMR 2011]

Deriving Performance Signatures Using Statistical Techniques

PC	*Eigen-Value*	*Variability (%)*	*Cumulative Variability %)*
PC1	11.43	63.506	63.506
PC2	2.47	15.260	78.765
PC3	1.720	9.554	88.319
PC4	0.926	5.143	93.463
⋮	⋮	⋮	⋮
PC12	0.001	0.003	100.00

Dimensionality Reduction using PCA (e.g., select top PCs with Cumulative Variability > 90%)

Rank	PC	Var	Weight	% Imp
1	PC1	N	0.974	5.636
2	PC1	M	0.972	5.613
3	PC1	R	0.966	5.544
4	PC1	Q	0.946	5.317
5	PC1	P	0.944	5.294
6	PC1	E	0.933	5.172
7	PC2	I	0.912	4.942

Selecting Top K Performance Counters (e.g., 7 out of 18 counters ~ 61% data reduction)

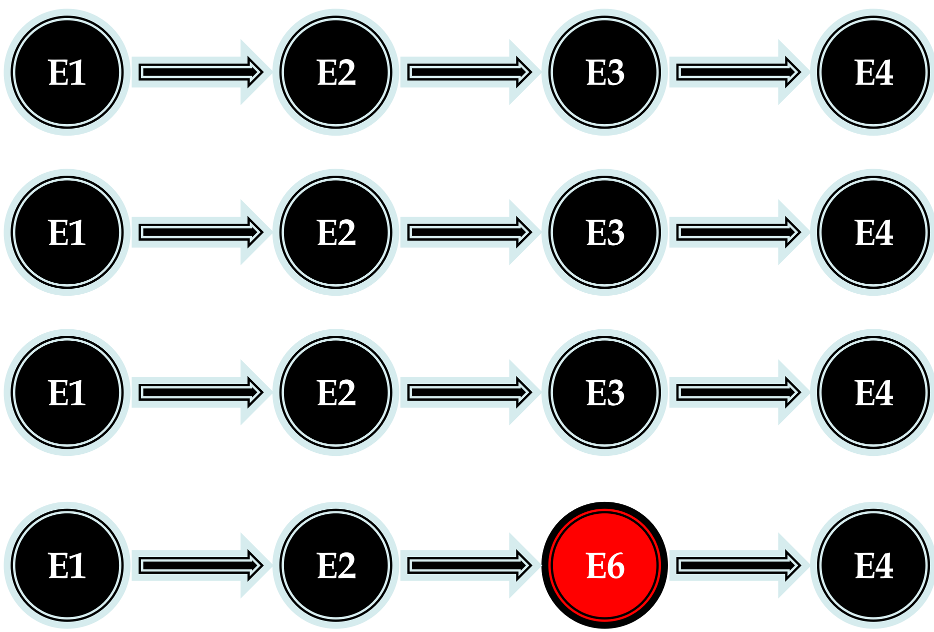
	1X Load	*2X Load*	*4X Load*	*8X Load*
1X Load	1			
2X Load	0.703	1		
4X Load	0.570	0.957	1	
8X Load	0.219	0.462	0.513	1

Spearman's rank correlation

[Malik et al., CSMR 2011]

Automated Functional Analysis

- (E2, E3) are always together:
  - (acquire_lock, release_lock)
  - (open_inbox, close_inbox)
- If we see (E2, E6), this might be a problem



Deriving Anomalous Functional Behavior

#	Z-Stat	Kinds	Min	Max	Total	Event
						SessionID=19420, Entering purchase for simple quantity queries
E_{19}	34.73	2	317	16,273	16,590	SessionID=14128, End of purchase process
E_{22}	20.65	2	1	3,857	3,858	SessionID=12067, Purchase complete

Freq	Sample	Details (Sort by Freq)
87,528 (98%)	ds2logs.txt 688 ds2logs.txt 689	E_{13} --> SessionID=19420, Entering purchase for simple quantity queries E_{14} --> SessionID=19420, Initial purchase, update cart
1,436 (<1%)	ds2logs.txt 2,484 ds2logs.txt 2,488	E_{13} --> SessionID=16242, Entering purchase for simple quantity queries E_{13} --> SessionID=16242, Entering purchase for simple quantity queries
358 (<1%)	ds2logs.txt 10,020 ds2logs.txt 10,021	E_{13} --> SessionID=13496, Entering purchase for simple quantity queries E_{15} --> SessionID=13496, Finish purchase before commit

[Jiang et al., ICSM 2008]

Load Testing Demo