Automated Application Deployment and Staging using Code Generation

Calton Pu

Galen Swint, Gueyoung Jung, Qinyi Wu, Jason Parekh

CERCS, Georgia Institute of Technology

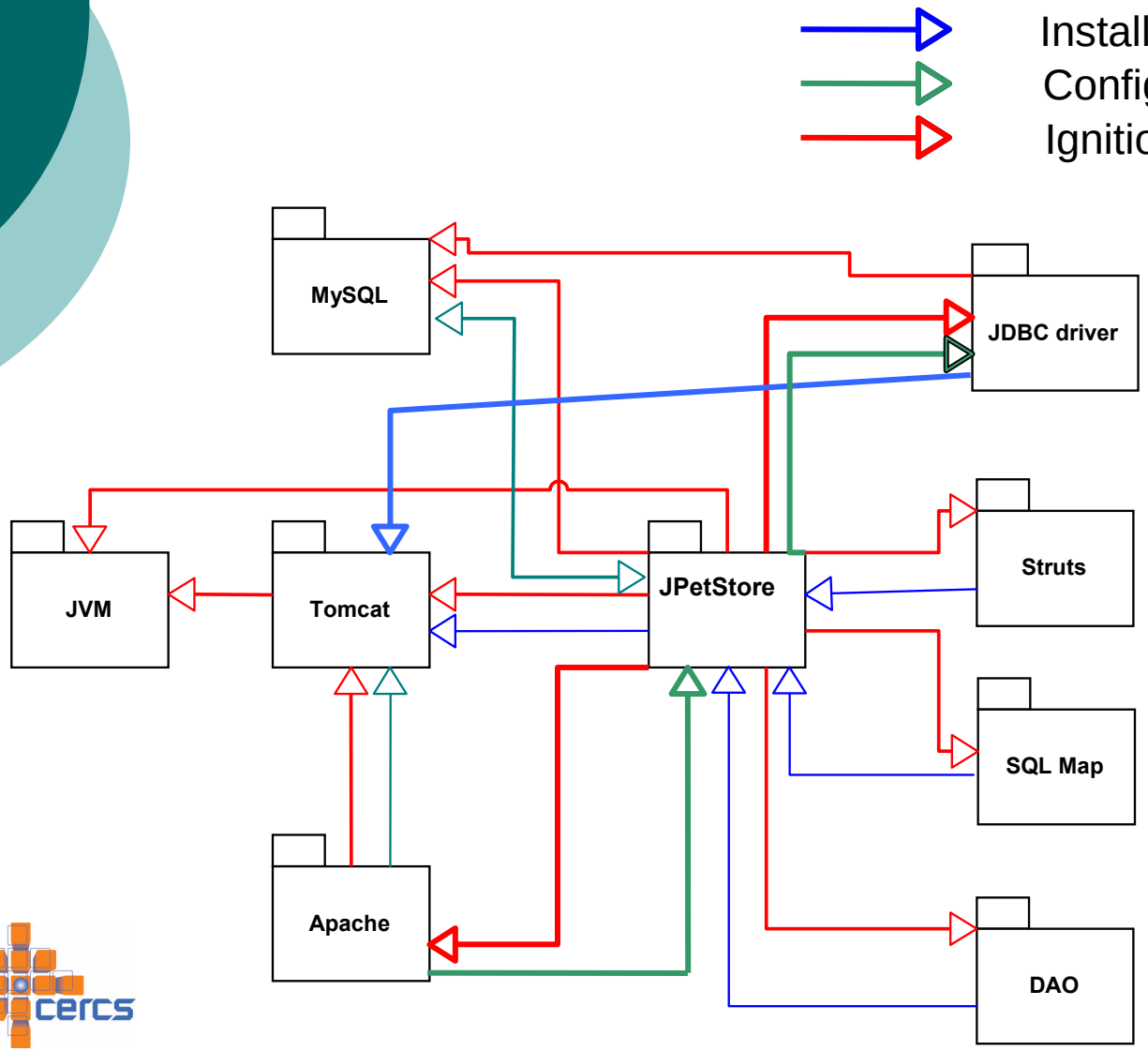
Akhil Sahai

Hewlett Packard Laboratories

Outline of Presentation

- Application management challenge
- Elba Project and Mulini code generator
- Clearwater code generation architecture and tools
- Conclusion and Future Work

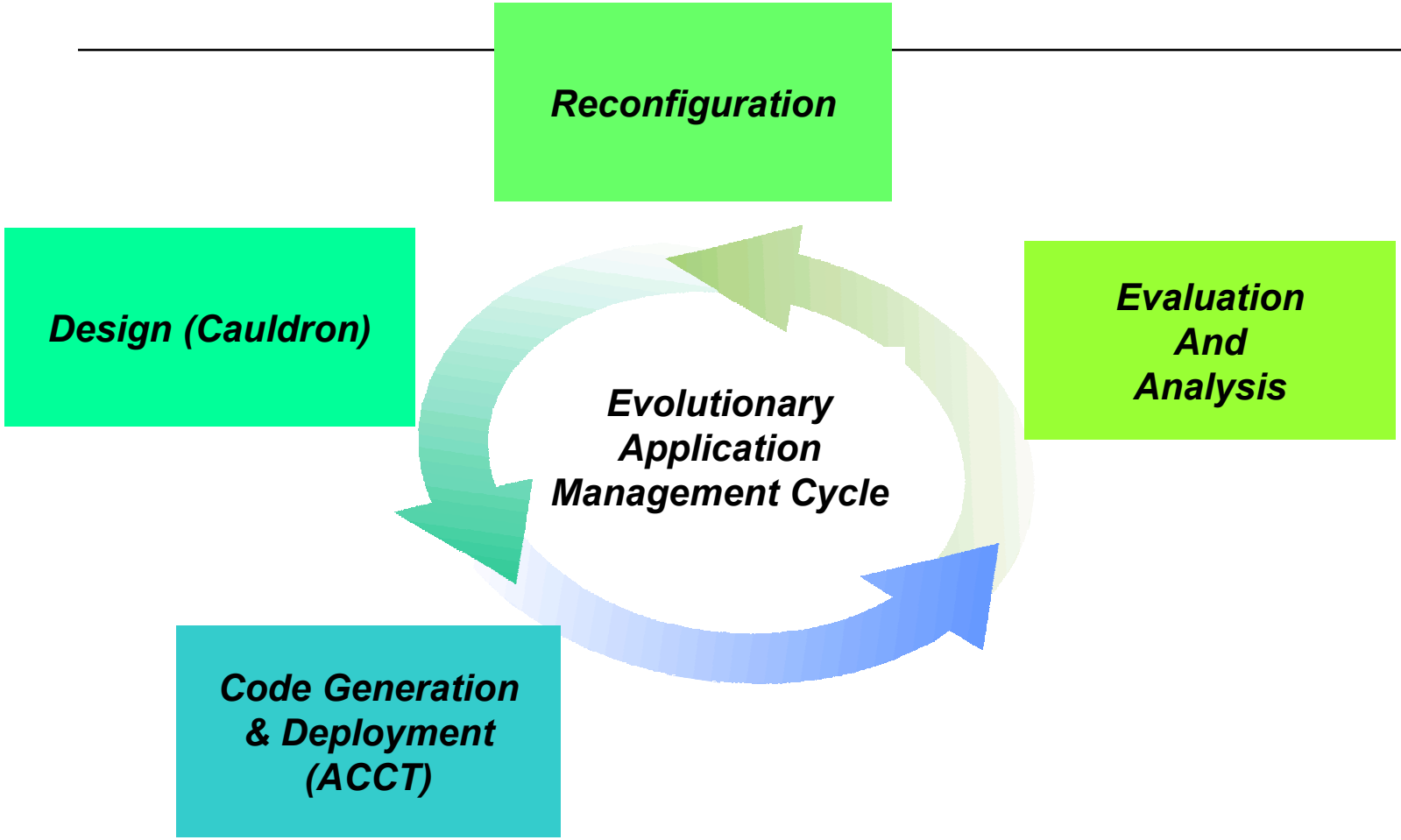
A 3-Tier Application Example



Application Trends

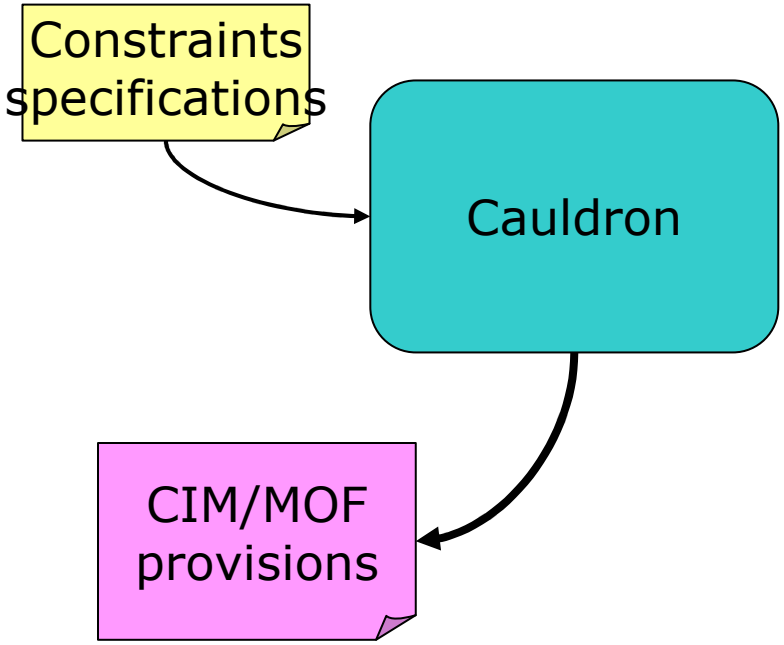
- Enterprise applications more complex
  - Enterprises grow more complex, larger
  - Data centers grow more complex, larger
  - Service composition will save the world
  - Management challenges grows with application complexity and Moore's Law
- Application management cycle
  - Design, Deployment, Evaluation, Reconfiguration (after growth)

Application Management Cycle



Cauldron Application Design Tool

- Input: application requirements, system resources
- Output: CIM/MOF
- Performs provisioning under constraints
- Computes pairwise deployment dependencies



Automated Composable Code Translator (ACCT)

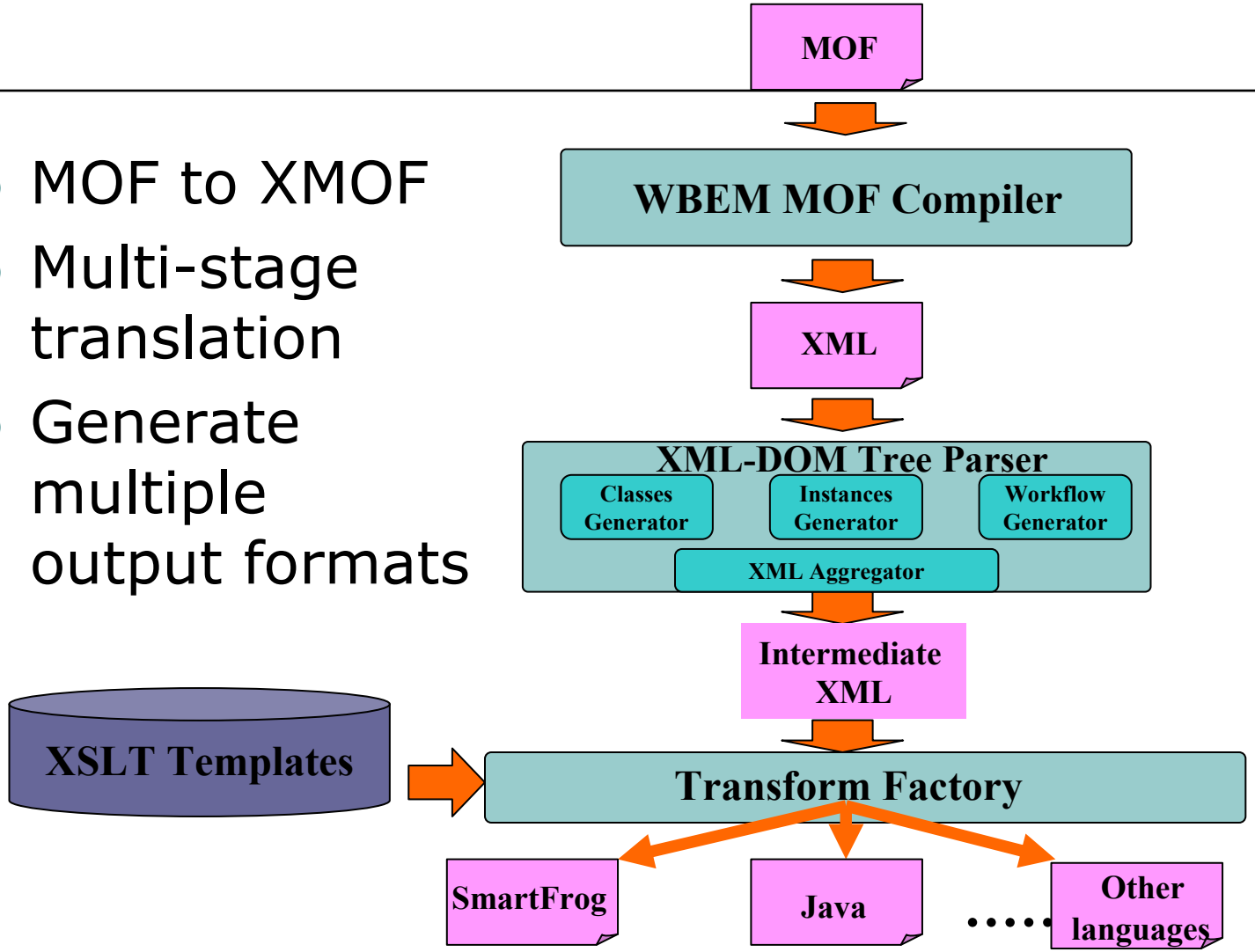
- Mapping Cauldron (high-level) to SmartFrog (deployable)
  - Multiple high level specifications
  - Multiple workflow representations
  - Multiple deployment output (SF, Java)
- Automate translation process
  - Clearwater architecture (extensible)

ACCT Project

- ACCT translator (DSOM'05)
  - Input: CIM/MOF provisions from Cauldron
  - Output: SmartFrog deployment program
- Based on Clearwater architecture
  - Translate MOF into XML-MOF, an evolvable format
  - Translation using XSLT templates for extensible code generation

ACCT Translator Architecture

- MOF to XMOF
- Multi-stage translation
- Generate multiple output formats



Workflow Translation

- Cauldron MOF contains pairwise dependencies – SS, FF, FS, SF
- SmartFrog expects a global workflow specification
- ACCT maps workflow in its "Workflow Generator"
- Generator computes parallel and serial operations from MOF

ACCT Transformation

```
instance of LogicalServer {
  Id = "Tomcat_LS1";
  Caption = "Tomcat Logical Server";
  Description = "Logical Server for Tomcat ";
  IpAddress = "130.207.5.228";
  HostName = "artemis.cc.gatech.edu";
};
instance of LogicalServerInLogicalApplication
{
  LogicalApplication = "Tomcat\";
  LogicalServer =Tomcat_LS1\";
};
instance of LogicalApplication {
  Id = "Tomcat";
  Version = "5.0.19";
  Caption = "Tomcat";
  Description = "Tomcat application Server";
};
instance of LogicalApplication {
  Id = "MySQLDriver";
  Version = "3.0.11";
  Caption = "MySQLDriver";
  Description = "MySQL driver";
};
instance of Activity {
  Id = "Tomcat_Installation";
  ActivityType = "script";
};
instance of Activity {
  Id = "Tomcat_Installation";
  ActivityType = "script";
};
Instance of ActivityPredecessorActivity {
  DependenceType=”Finish-Start”;
  AntecedentActivity=”Tomcat_Installation”;

DependentActivity=”MySQLDriver_installation
”;
};
```

(a) MOF

```
<Instance                                  Name="Tomcat"
Class="LogicalApplication">
  <Variable
Name="Id"Type="string">Tomcat</Variable>
  <Variable Name="Version"Type="string">
    5.0.19</Variable>
  <Variable Name="Entity" Type="string">
    Activity_Tomcat_Installation</Variable>
  <Variable Name="Host" Type="string">
    artemis.cc.gatech.edu</Variable>
</Instance>
  <Workflow>
    <Work Type=”Execution”></Work>
    <Work Type=”EventSend”>
      <To> MySQLDriver_Installation</To></Work>
    <Work Type=”Terminate”>
     Tomcat_Installation </Work>
  </Workflow>
<Instance  Name="MySQLDriver"
           Class="LogicalApplication">
  <Variable Name="Id" Type="string">
    MySQLDriver</Variable>
  <Variable Name="Version" Type="string">
    3.0.11</Variable>
  <Variable Name="Entity" Type="string">
    Activity_MySQLDriver_Installation</Variable>
  <Variable Name="Host" Type="string">
    demeter.cc.gatech.edu</Variable>
</Instance>
 <Workflow>
   <Work Type=”OnEvent”>
     <From> Tomcat_Installation</From> </Work>
   <Work Type=”Execution”></Work>
   <Work Type=”Terminate”>
     MySQLDriver_Installation</Work>
 </Workflow>
```

(b) XMOF

```
sfProcessComponentName  "Tomcat_Installation";
LogicalApplication_Tomcat  extends  LogicalApplication {
    Id   "Tomcat";
    Version   "5.0.19";
    Activity LAZY ATTRIB   Activity_Tomcat_Installation;
    sfProcessHost   "artemis.cc.gatech.edu";
}
Activity_Tomcat_Installation  extends  Activity {
    Id   "Tomcat_Installation";
    Entity   LAZY ATTRIB LogicalApplication_Tomcat;
}
-- extends EventSend {
    sendTo eventQueue:queue_Tomcat_Ignition;
    event "Activity_Tomcat_Installation_FS";
}
-- extends Terminator {
    kill eventQueue:queue_Tomcat_Installation;
}
sfProcessComponentName "MySQLDriver_Installation";
-- extends OnEvent {
    registerWith queue_MySQLDriver_Installation ;
    Activity_Tomcat_Installation_FS extends DoNothing
}
LogicalApplication_MySQLDriver extends LogicalApplicatior
    Id   "MySQLDriver";
    Version   "3.0.11";
    ActivityLAZYATTRIBActivity_MySQLDriver_Installation;
    sfProcessHost   "demeter.cc.gatech.edu";
}
Activity_MySQLDriver_Installation extends Activity {
    Id   "MySQLDriver_Installation";
    Entity  LAZY ATTRIB   LogicalApplication_MySQLDriver;
}
-- extends Terminator {
    kill eventQueue:queue_MySQLDriver_Installation;
}
```

(c) SmartFrog

Automation of Application Deployment Cycle

- Automated application deployment
  - Cauldron generates CIM/MOF
  - ACCT generates SmartFrog
  - SmartFrog generates Java
  - Java program deploys application
- How do we debug the application deployment cycle?
  - ***Staging*** before production deployment

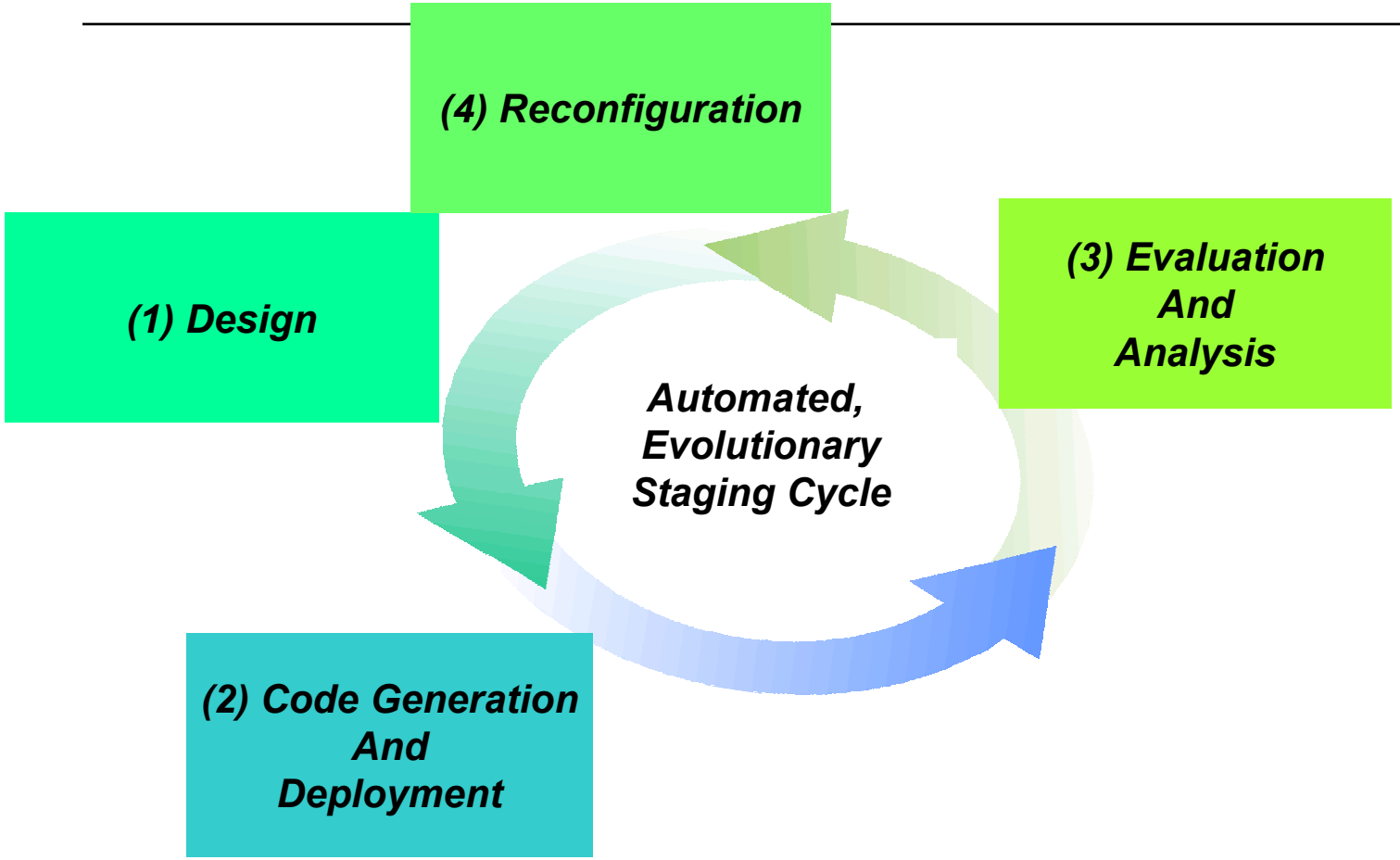
Challenges in Staging

- Staging deploys in a test environment
  - Good: Verify functionality and the satisfaction of SLAs, Uncover over- or under-provisioning
  - Bad: Manual or *ad hoc* approaches to staging are error-prone, time consuming
- Technical challenges of good staging
  - Correctly translating application specifications into staging configuration
  - Properly mapping SLA to staging workload
  - Exhaustively evaluating configurations and workloads

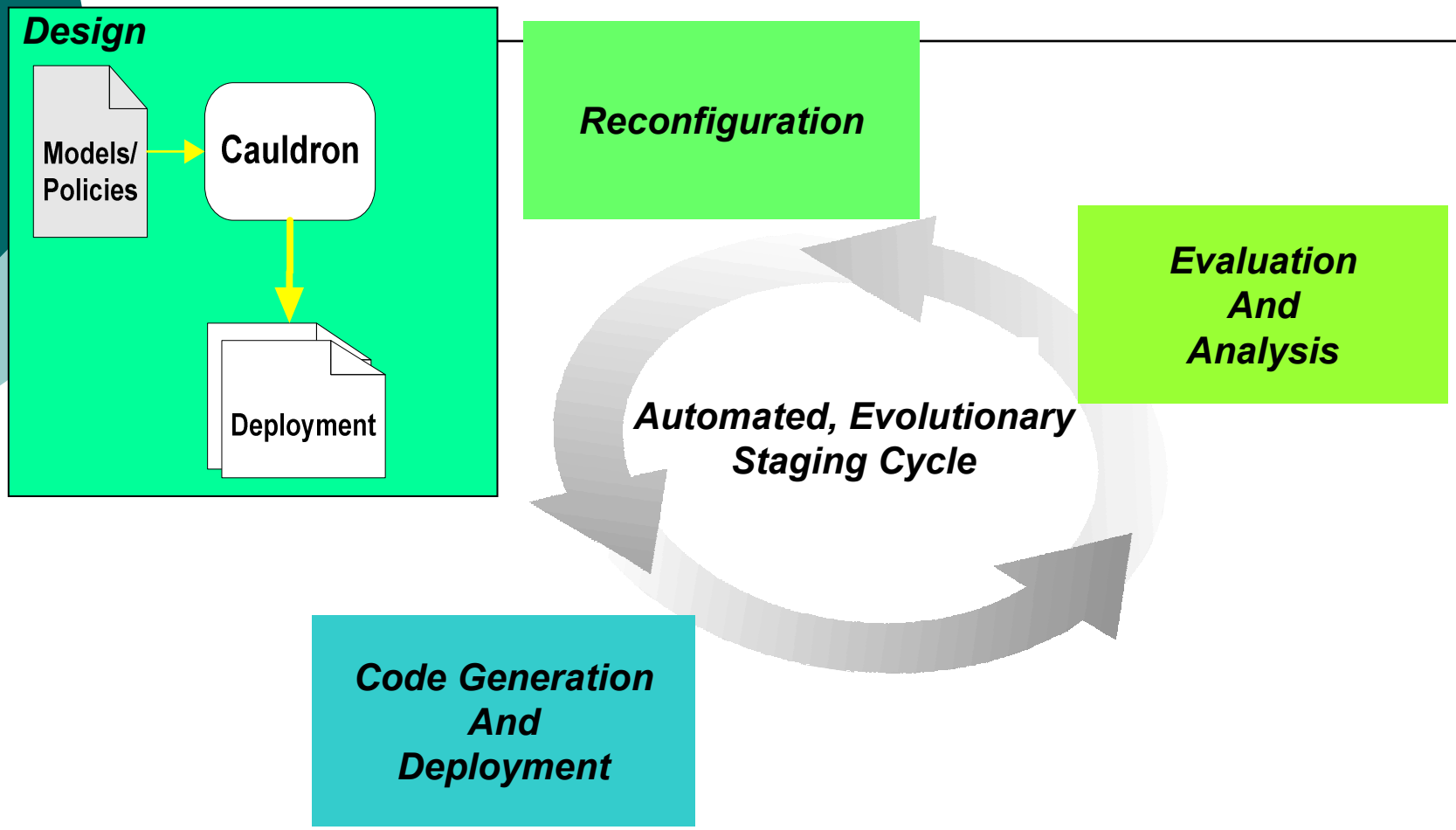
Elba Project

- Goal: Automate staging cycle
  - Systems design, deployment, evaluation, and reconfiguration
- Generate / translate staging plan
  - Reuse system deployment specifications
  - Reuse Policy specifications
  - TBL describes staging parameters

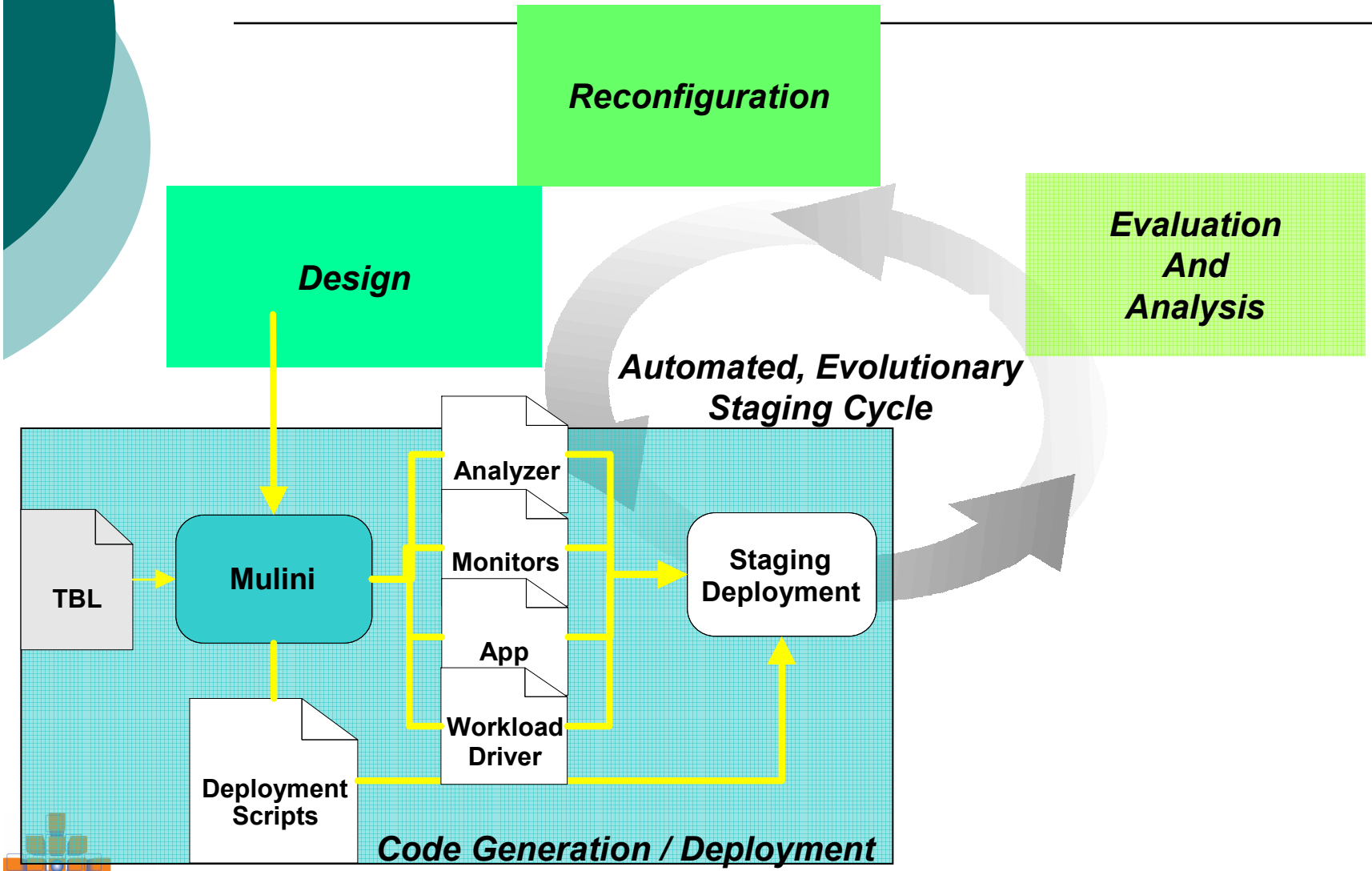
Staging Simulates Deployment



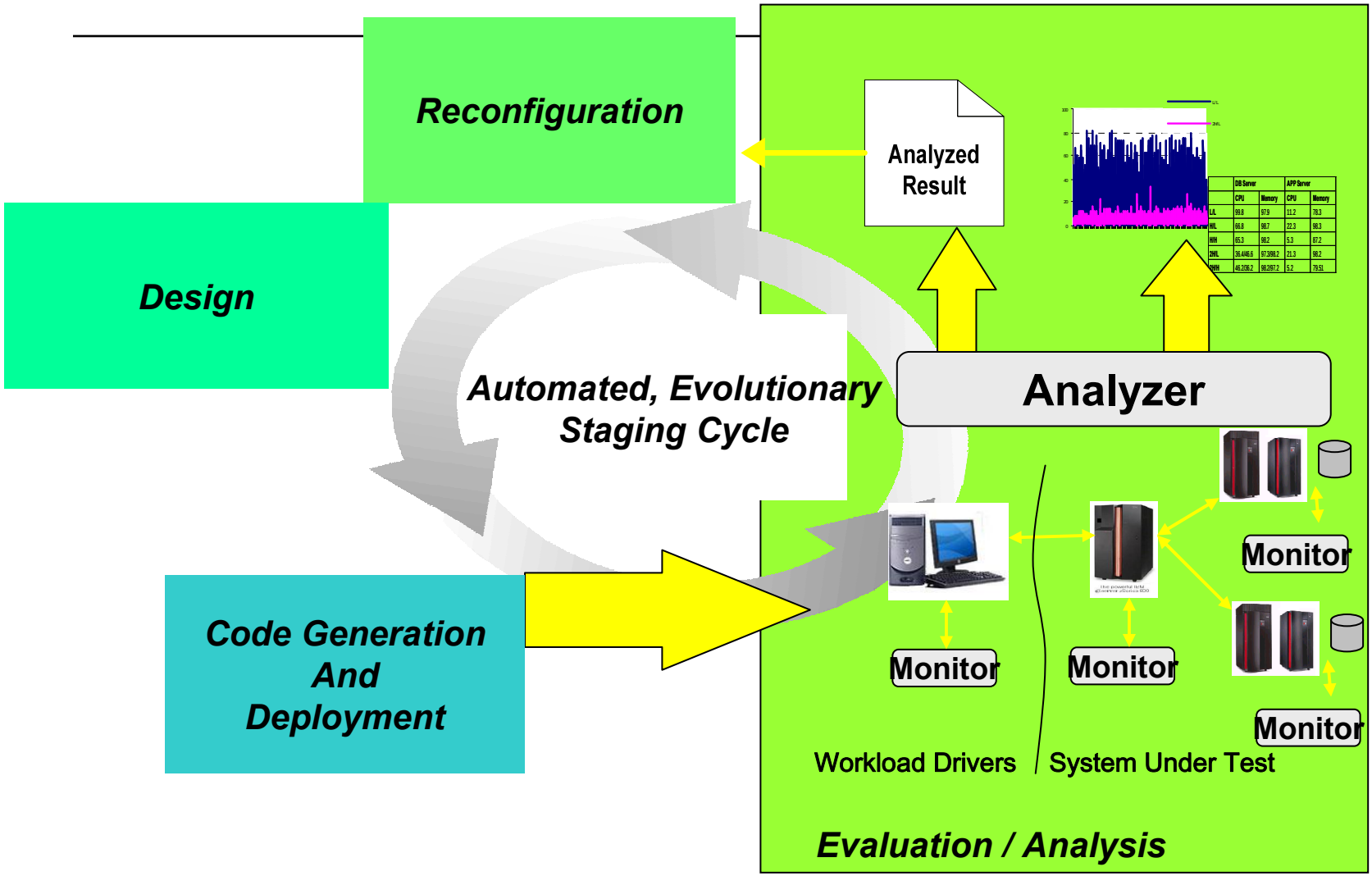
(1) Design: Cauldron Tool



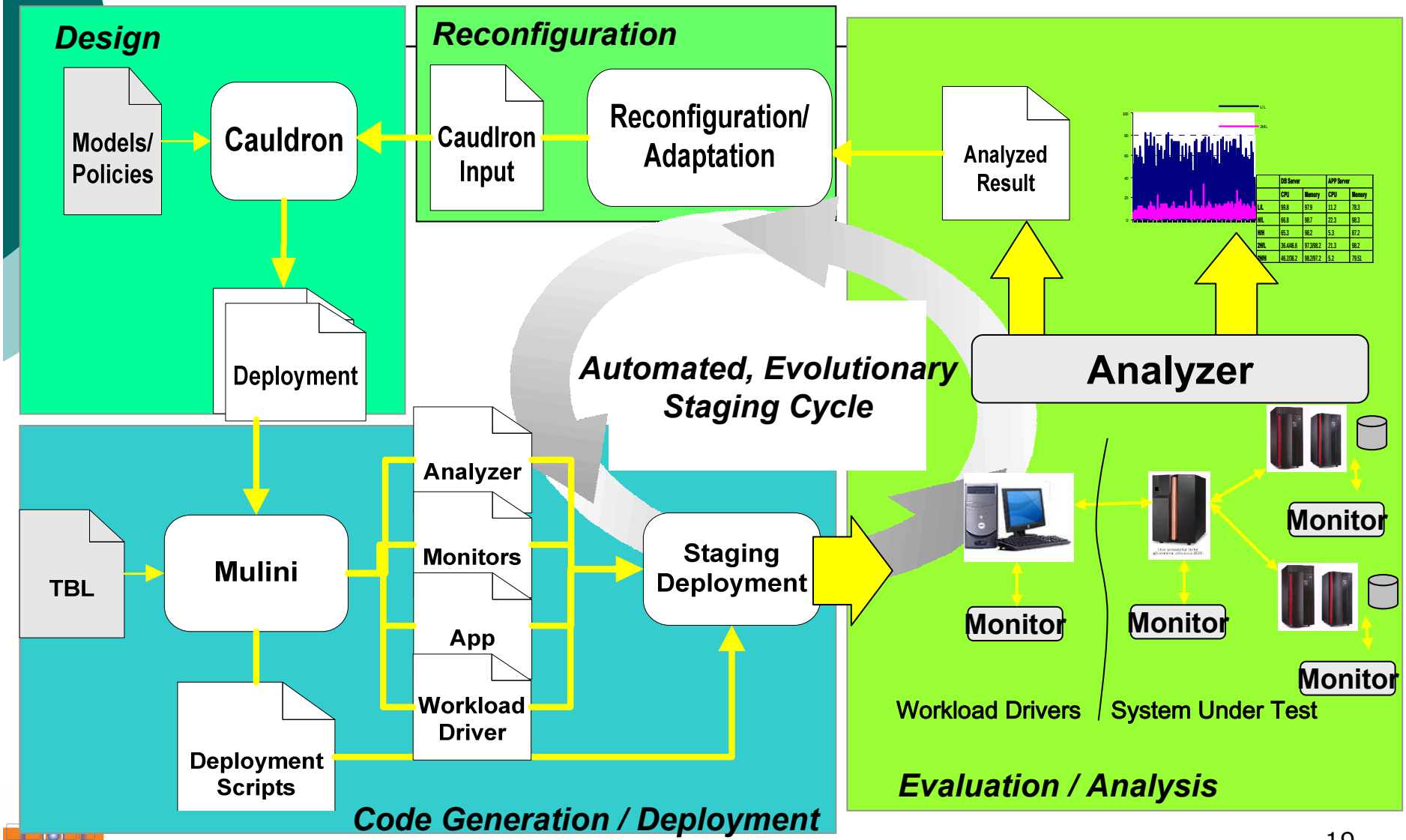
(2) Mulini Code Generator



(3) Automated Analyzer



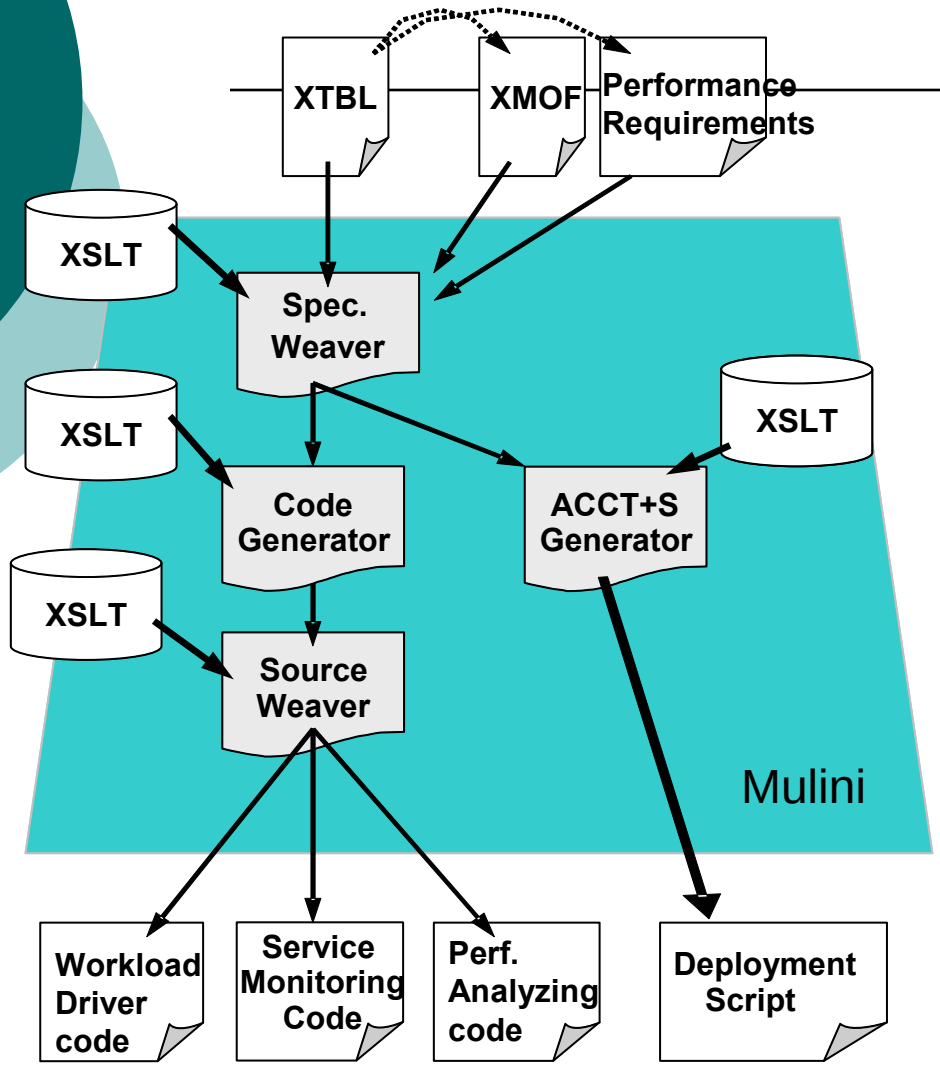
Elba: Overall Picture



Mulini: Basic Ideas

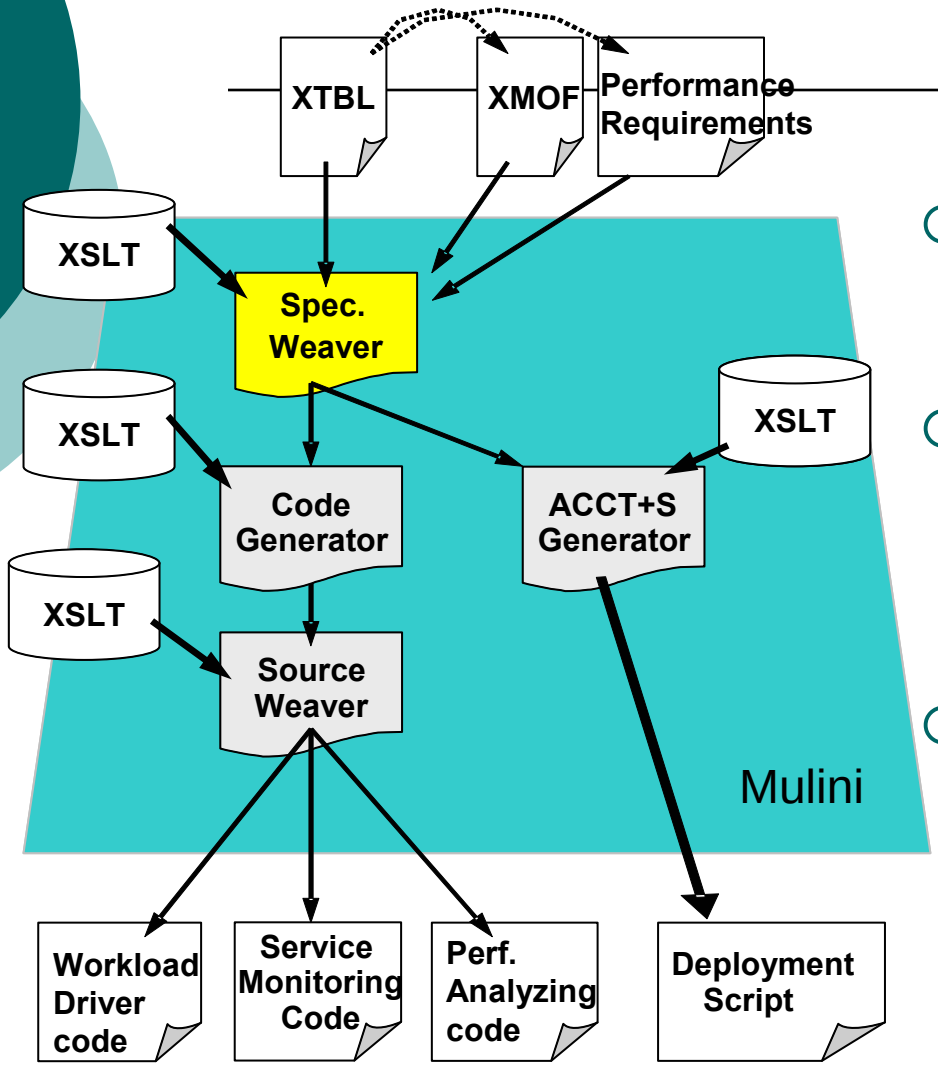
- Based on the *Clearwater* approach (ASE'05)
  - XML-based specifications
  - XSLT templates for generation
  - Extensible, Flexible, Modular
- Weaves monitoring code after generation

Mulini Design Goals



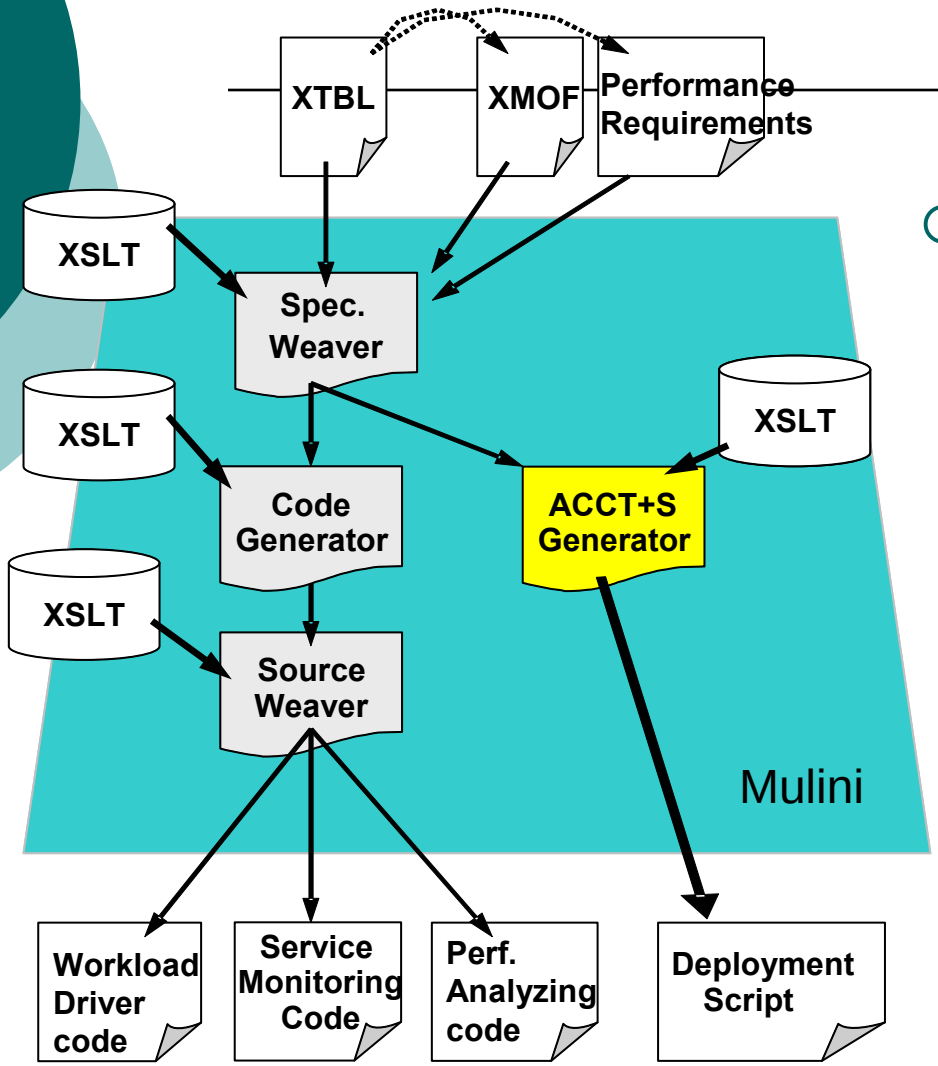
- Maps high-level abstractions to low-level staging code
- Instruments application using aspect weaving technology

Mulini Specification Weaver



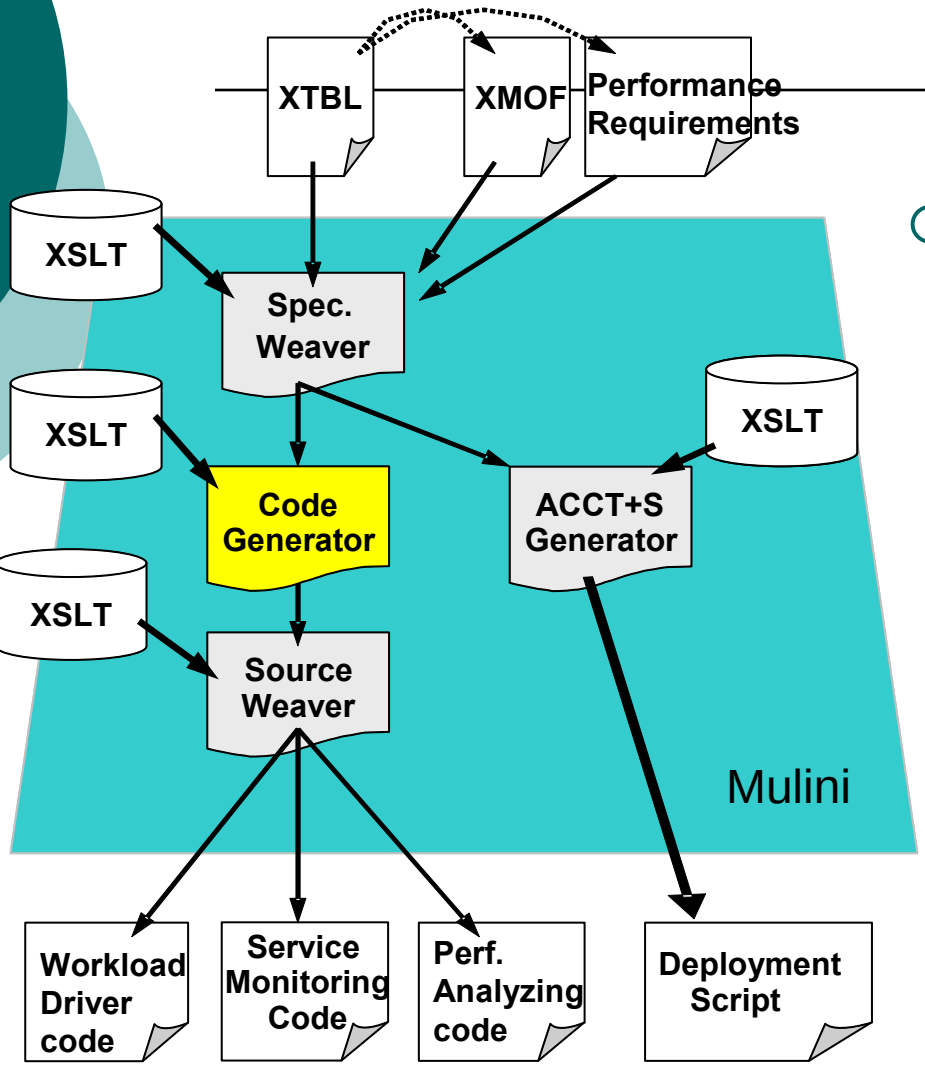
- Load XTBL with XMOF and WSML
- Extract information from XMOF and WSML
- Integrate to XTBL+

Mulini Integrated with ACCT



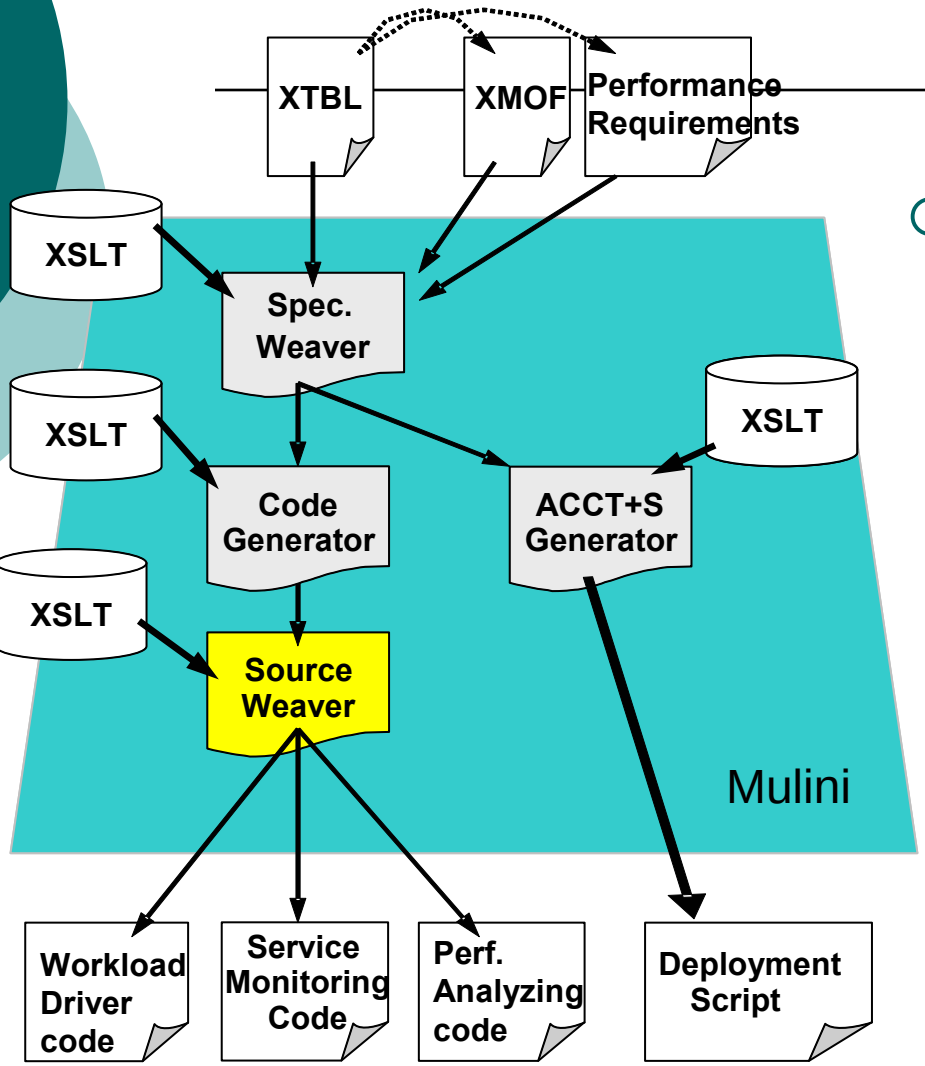
- ACCT+S
  - Extend ACCT (DSOM2005)
  - Generate staging deployment script using XTBL+

Mulini Code Generator



- Generates staging code (e.g., workload driver, monitoring code, Makefiles, execution script)

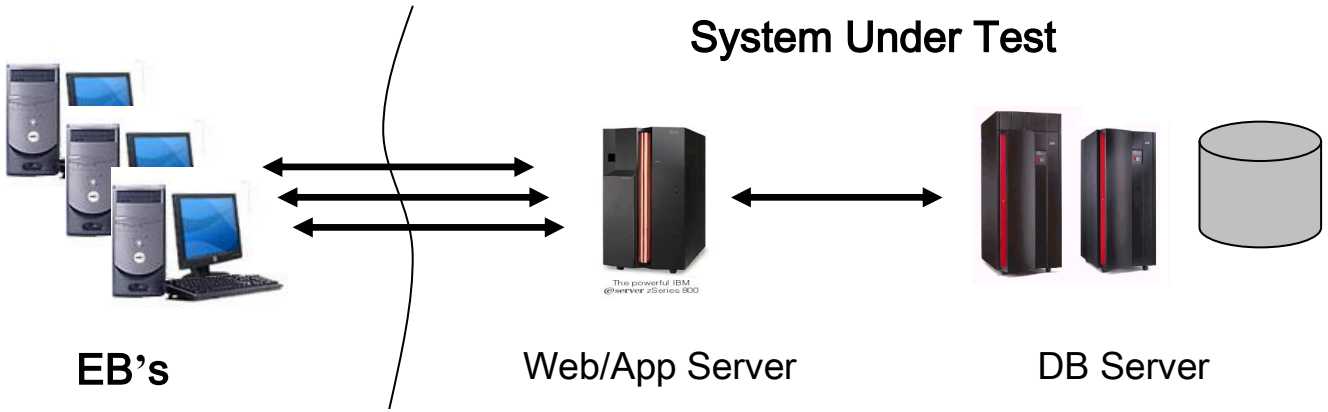
Mulini Code Weaver



- Perform fine grain instrumentation on base code and application source code using XTBL+

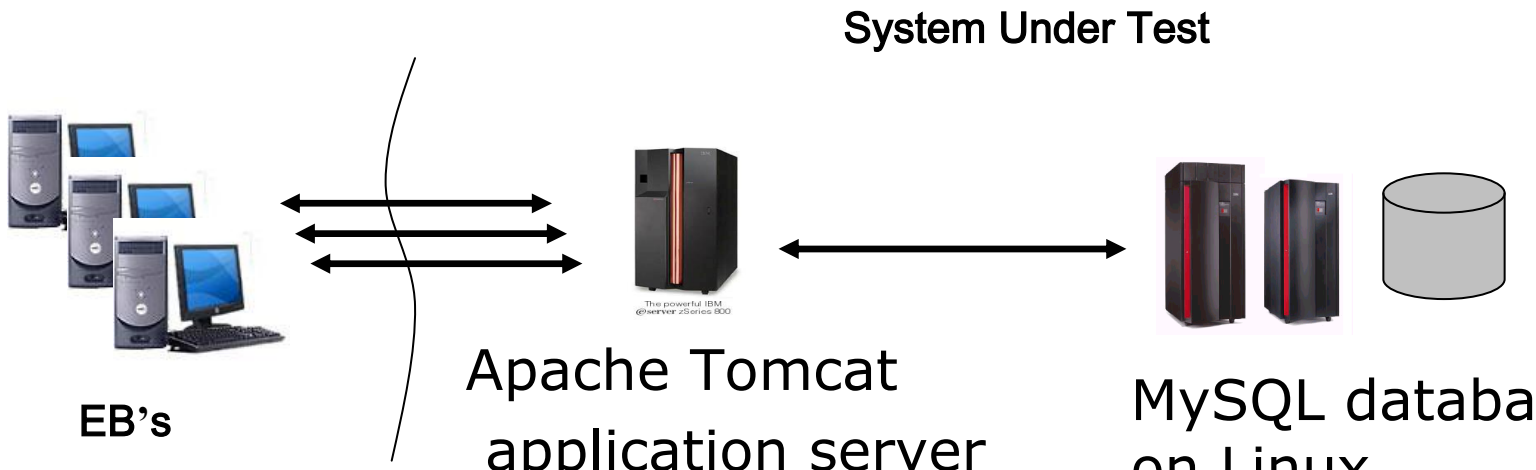
TPC-W On-line Bookstore

- Emulated Browsers (EB's)
- e-Commerce application (Bookstore)
- EB navigates Web pages (e.g., Home, Bestseller, SearchReq) based on transition models
- Measure WIPS, Response Times



Experiments: Software Setup

- TPC-W On-line Bookstore 2-tier servlet version

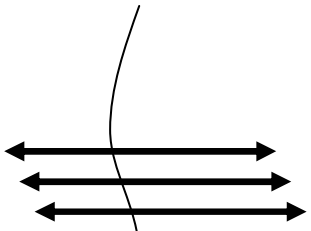


Experiments: Hardware Setup

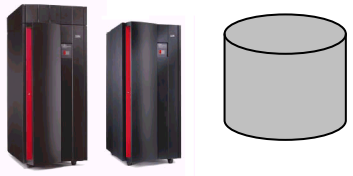
Low-end (L)

High-end (H)

EB's



System Under Test



Low-end (L) :
Dual-processor P-III
800 MHz with 512
MB Memory
$500

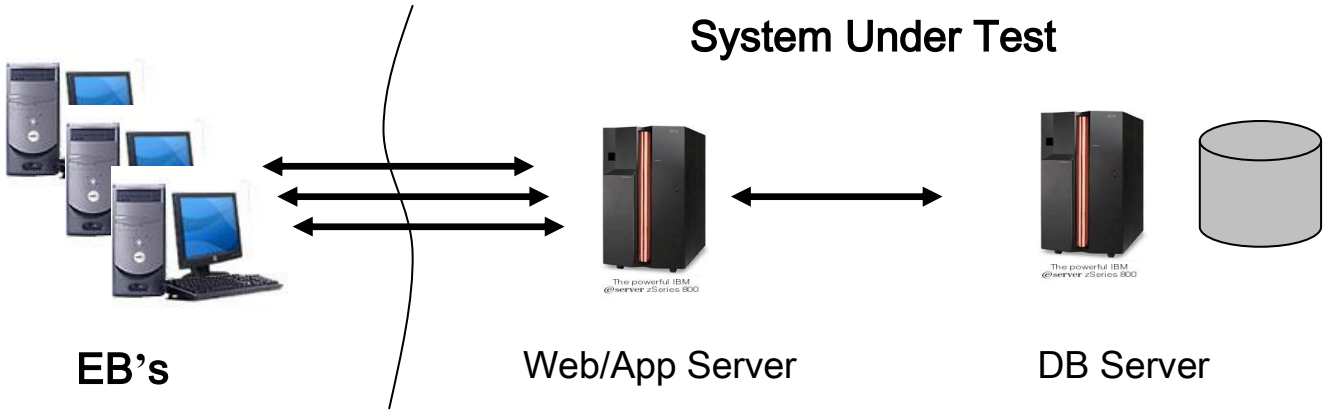
High-end (H) :
Dual-processor
Xeon 2.8 GHz with
4 GB Memory
$3,500

Experiments : Metrics

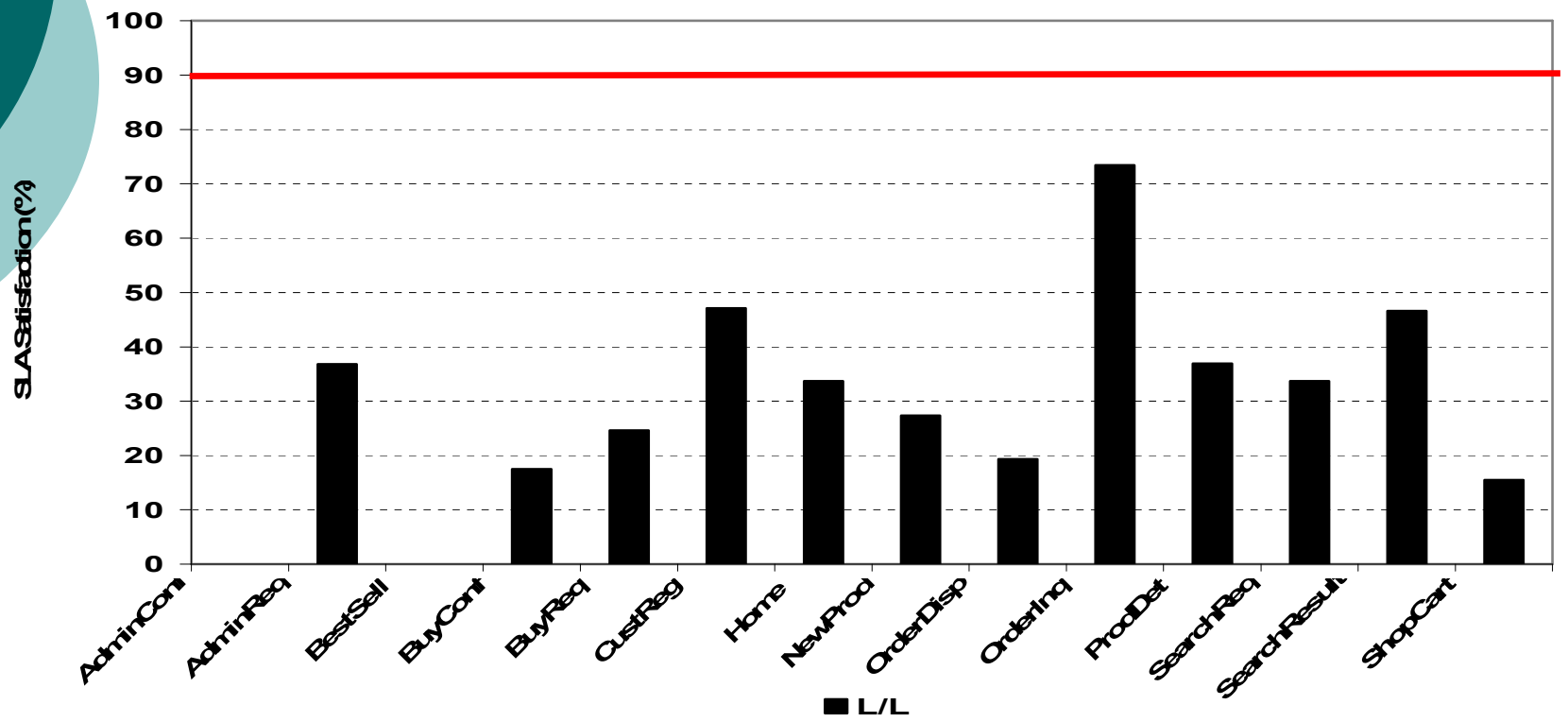
- Performance metrics of TPC-W
  - Average Response Time of each interaction type
  - Average Overall WIPS
- Implementation-level metrics
  - Average CPU and Memory Utilizations
  - Average DB Query Response Time
- Num of current users : 150

L/L Configuration

- L/L ($1,000)



TPC-W Measurements (L/L)



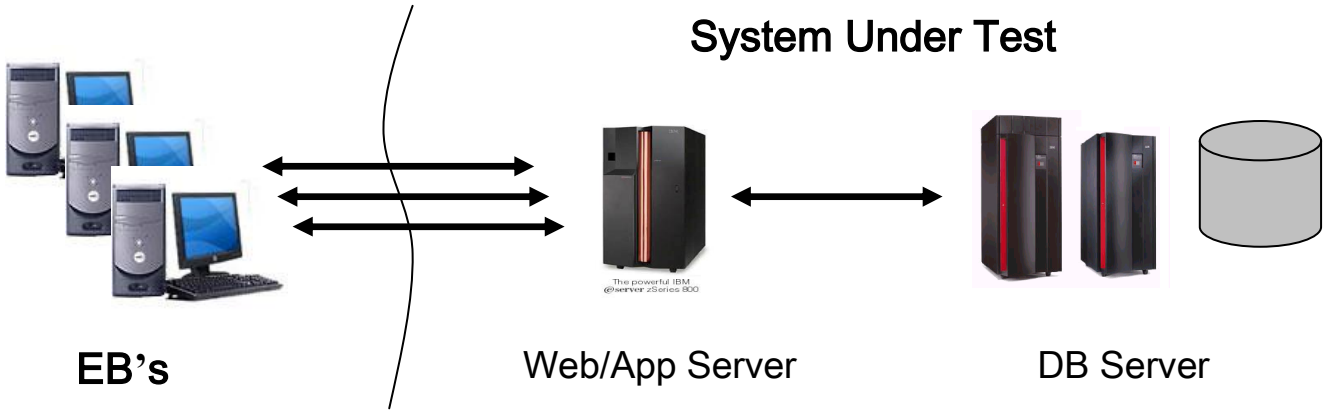
Summary of SLA satisfaction (% of interactions)

TPC-W Resource Utilization

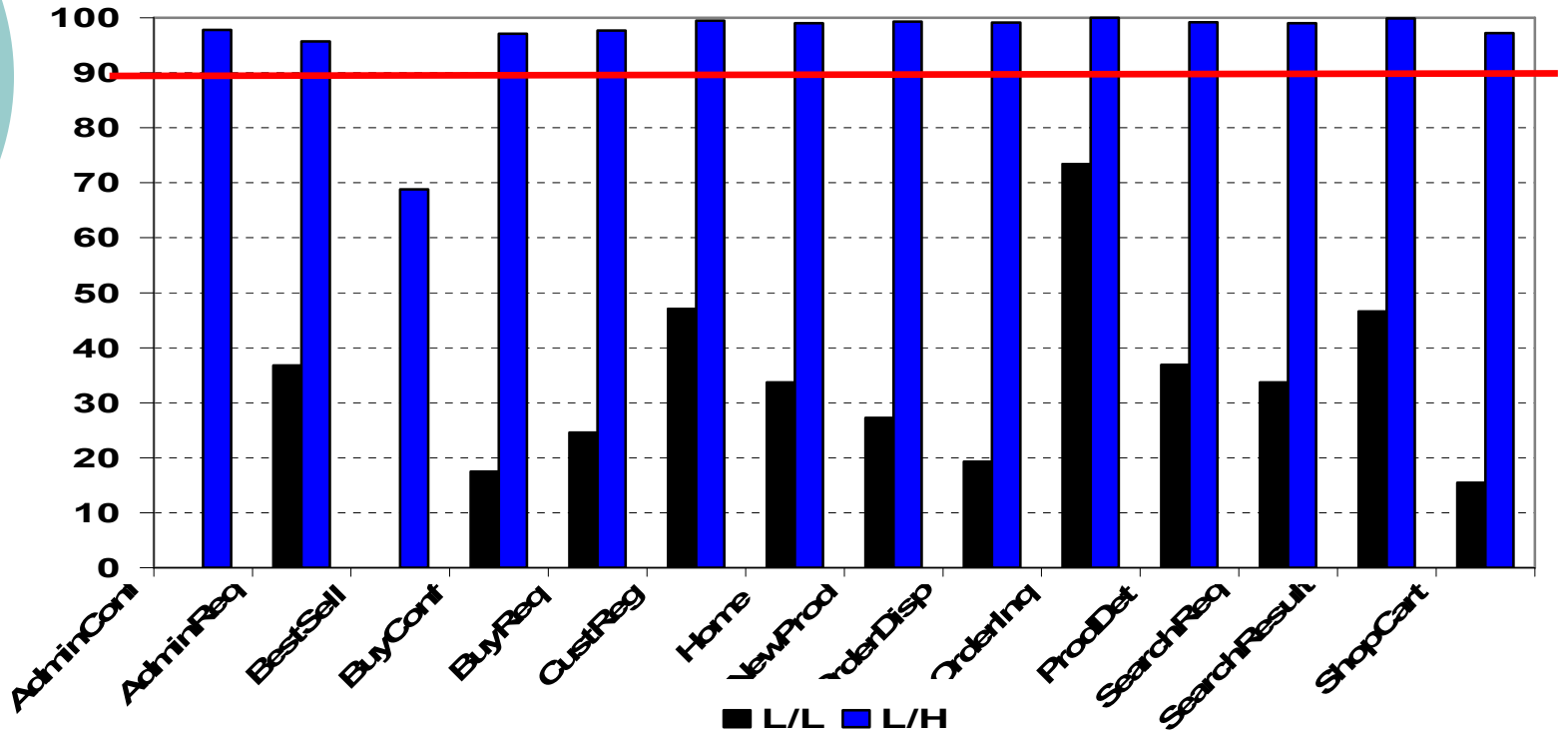
	DB Server		App. Server	
	CPU	Memory	CPU	Memory
L/L	99.8	98.4	21.9	79.7

L/H Configuration

- L/H ($4,000)



TPC-W: Measurements (L/H)



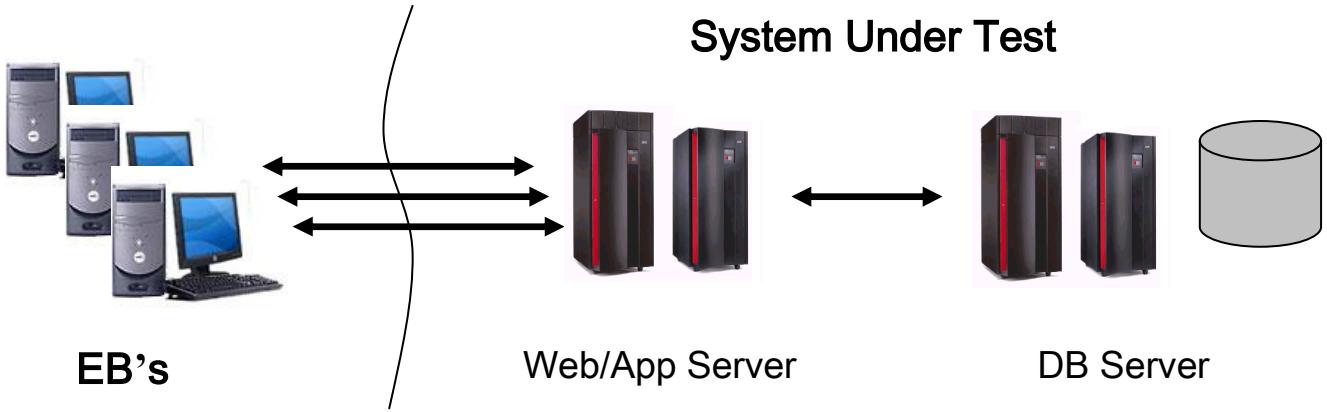
Summary of SLA satisfaction (% of interactions)

TPC-W Resource Utilization

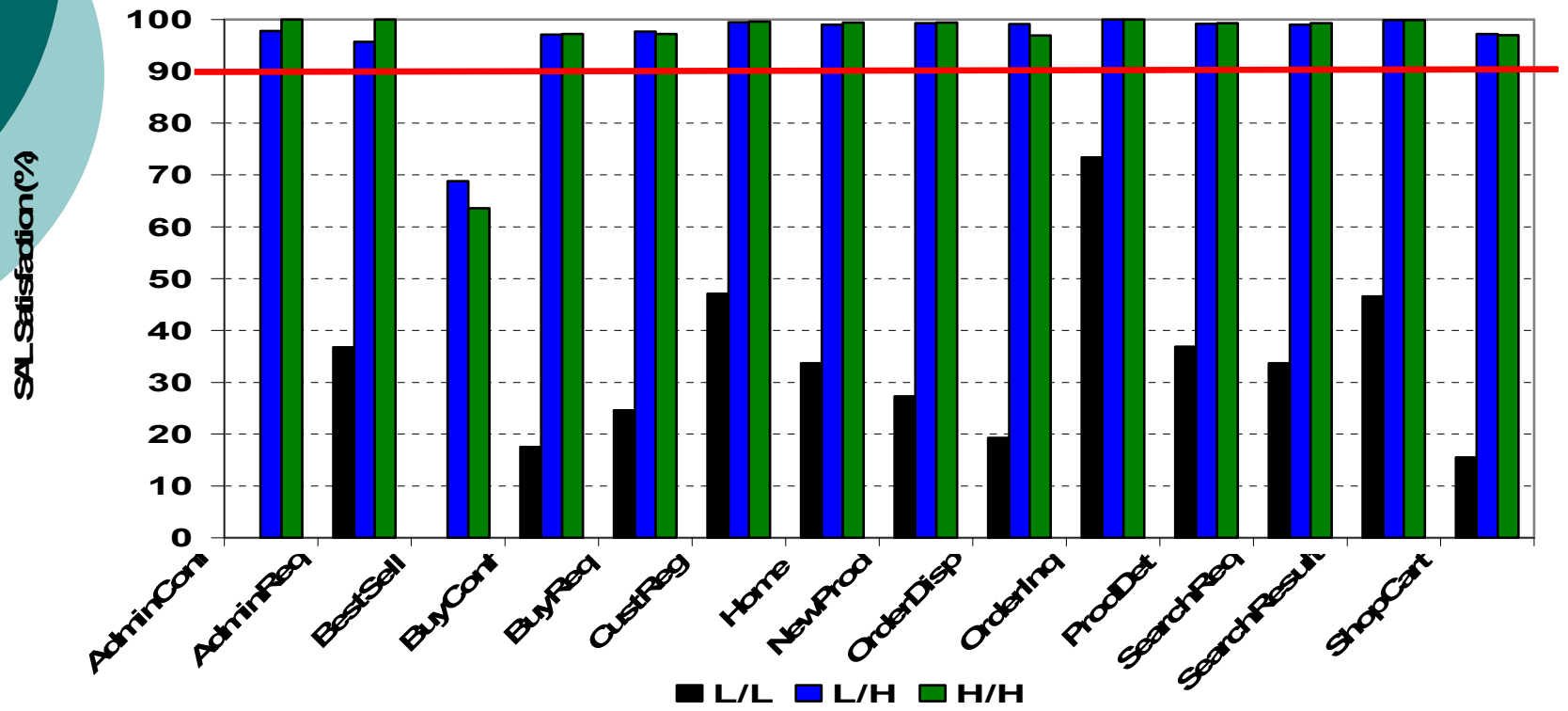
	DB Server		App. Server	
	CPU	Memory	CPU	Memory
L/L	99.8	98.4	21.9	79.7
L/H	65.9	99.2	23.9	100

H/H Configuration

- H/H ($7,000)



TPC-W: Measurements (H/H)



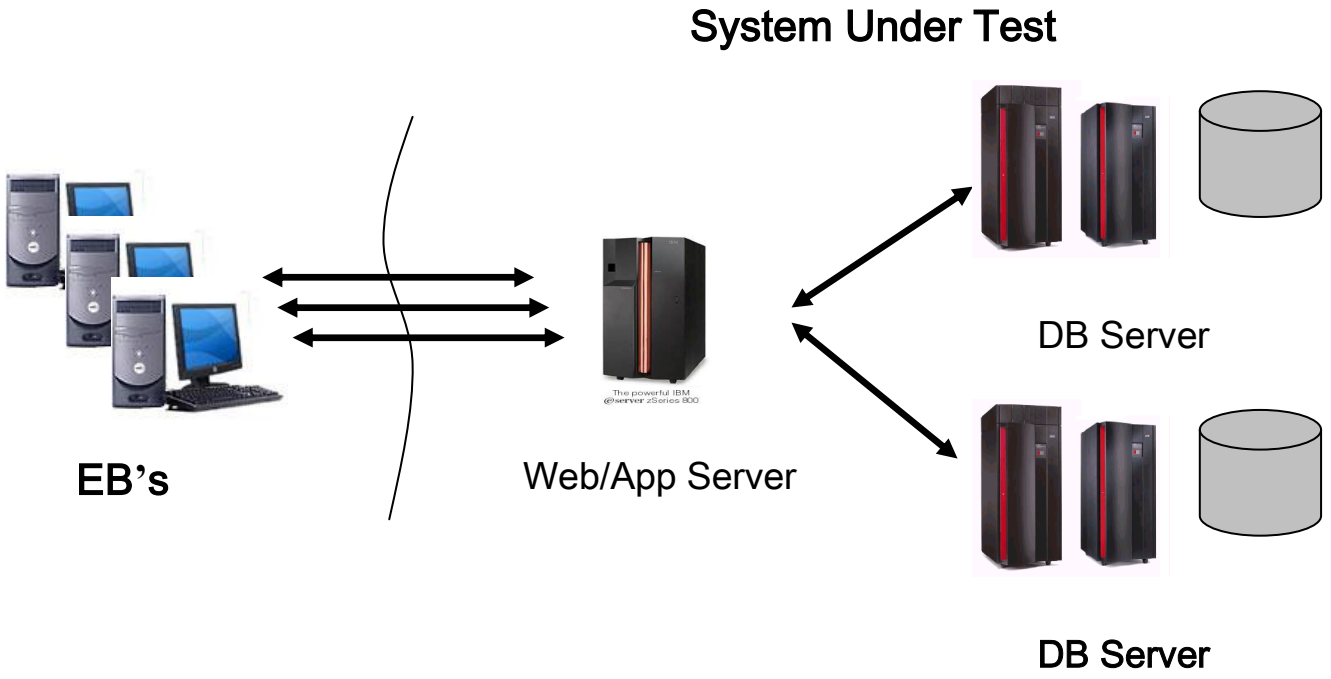
Summary of SLA satisfaction (% of interactions)

TPC-W Resource Utilization

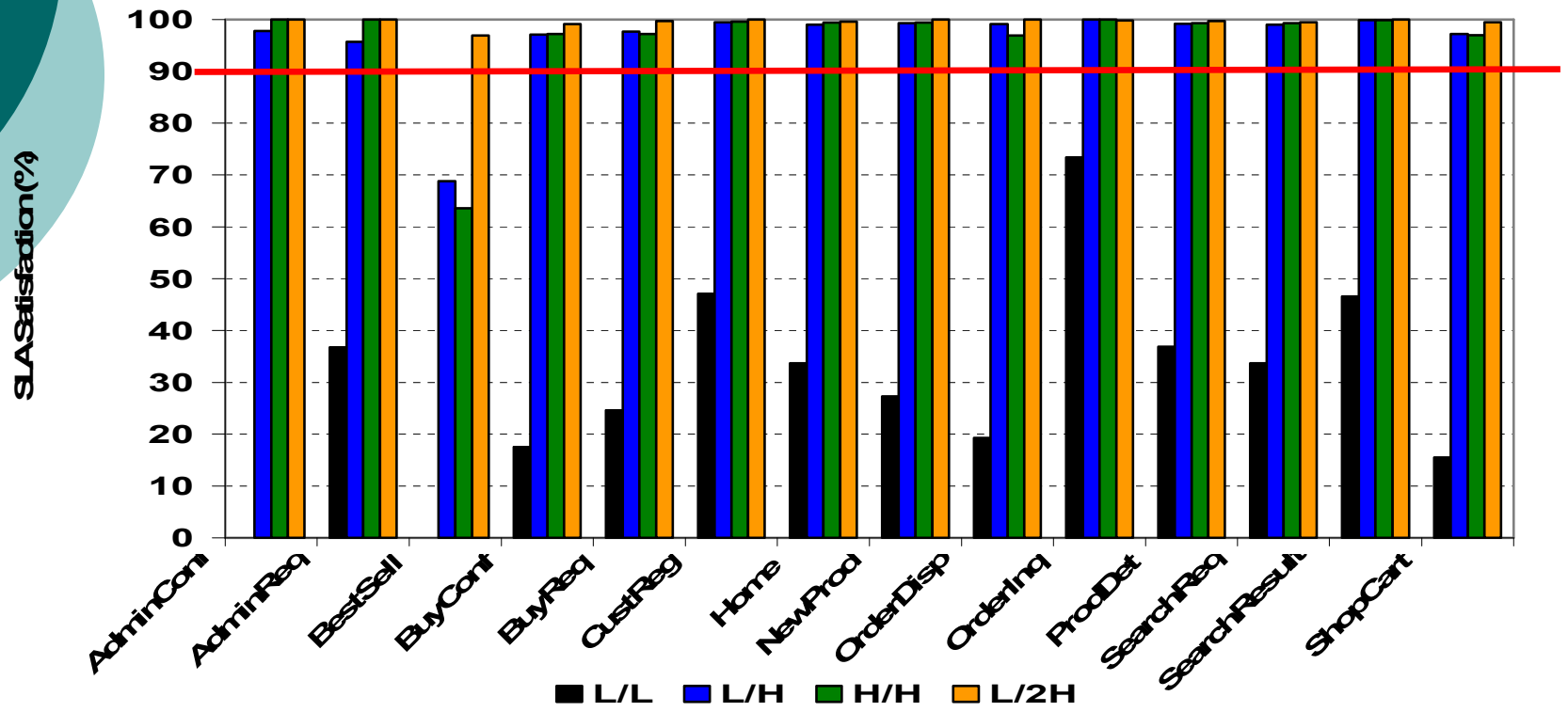
	DB Server		App. Server	
	CPU	Memory	CPU	Memory
L/L	99.8	98.4	21.9	79.7
L/H	65.9	99.2	23.9	100
H/H	62.8	98.4	7.2	79.9

L/2H Configuration

- L/2H ($7,500)



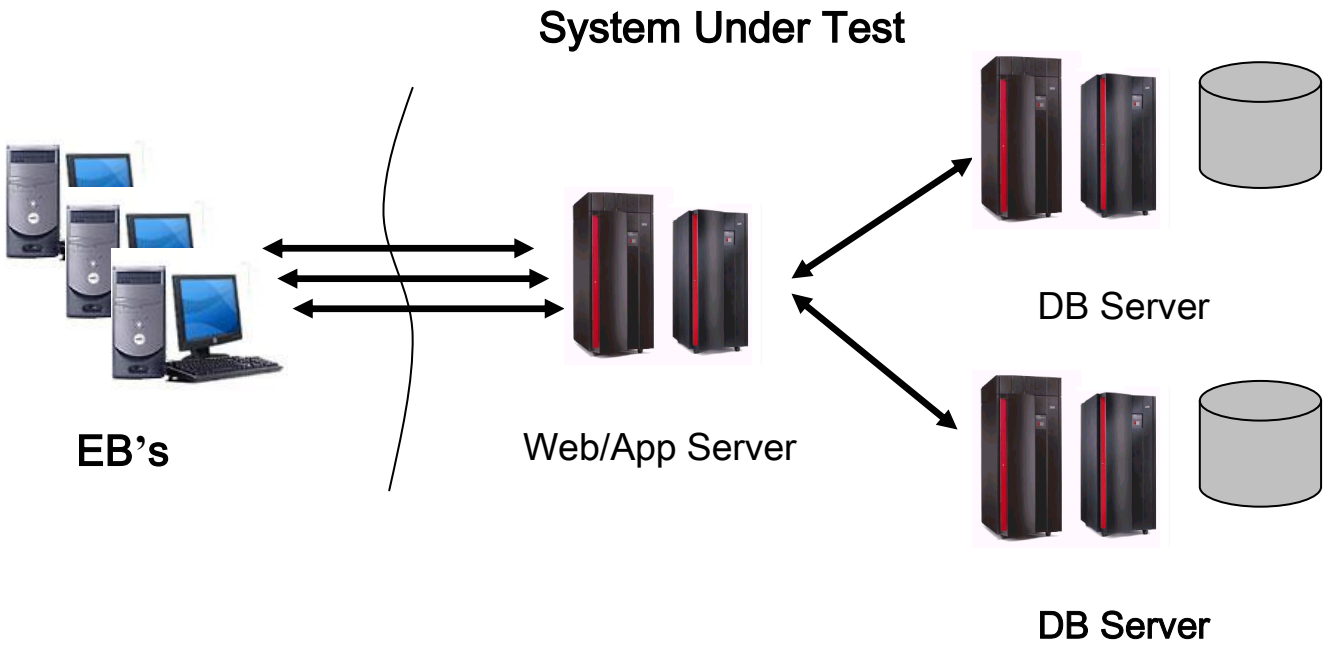
TPC-W Measurements (L/2H)



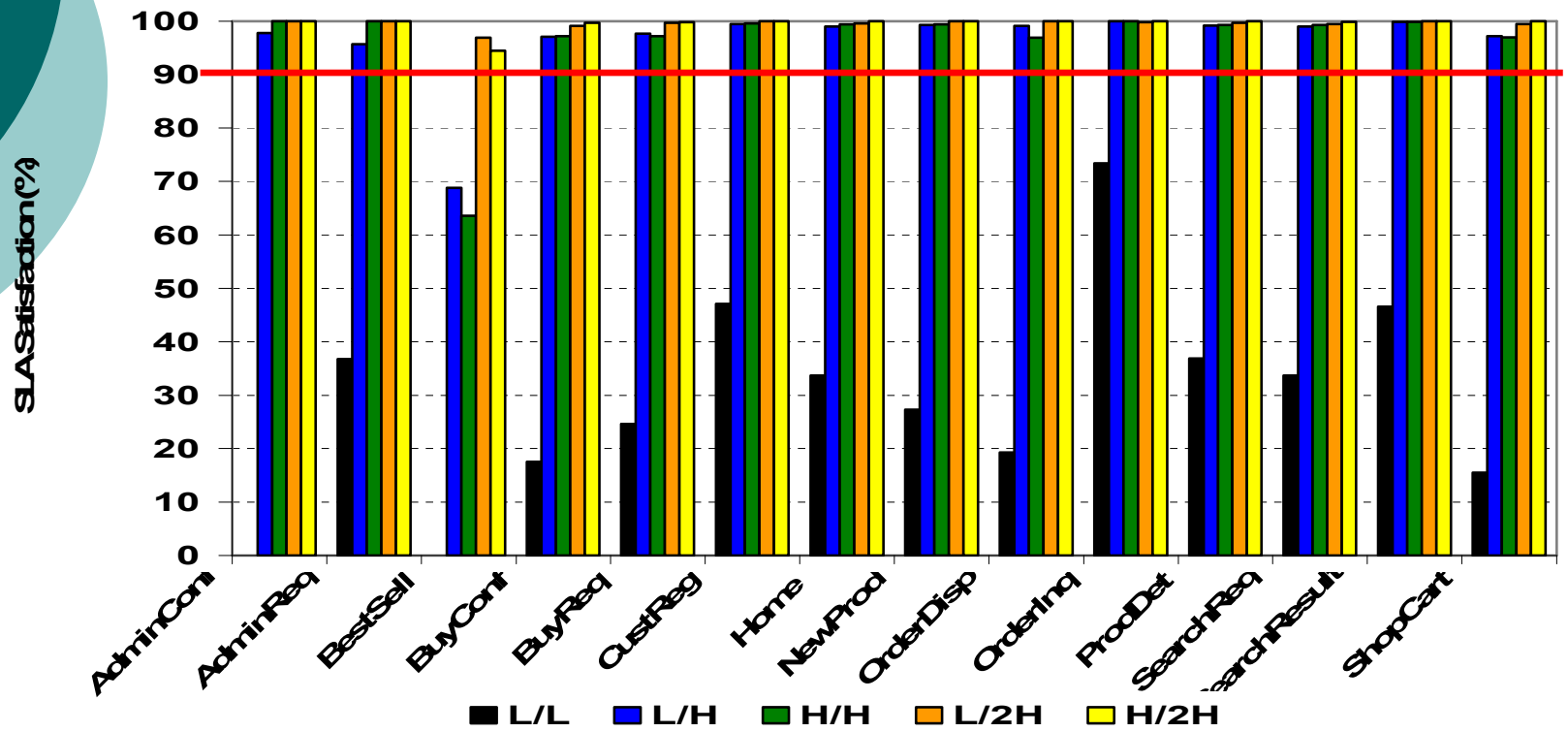
Summary of SLA satisfaction (% of interactions)

H/2H Configuration

- H/2H ($10,500)

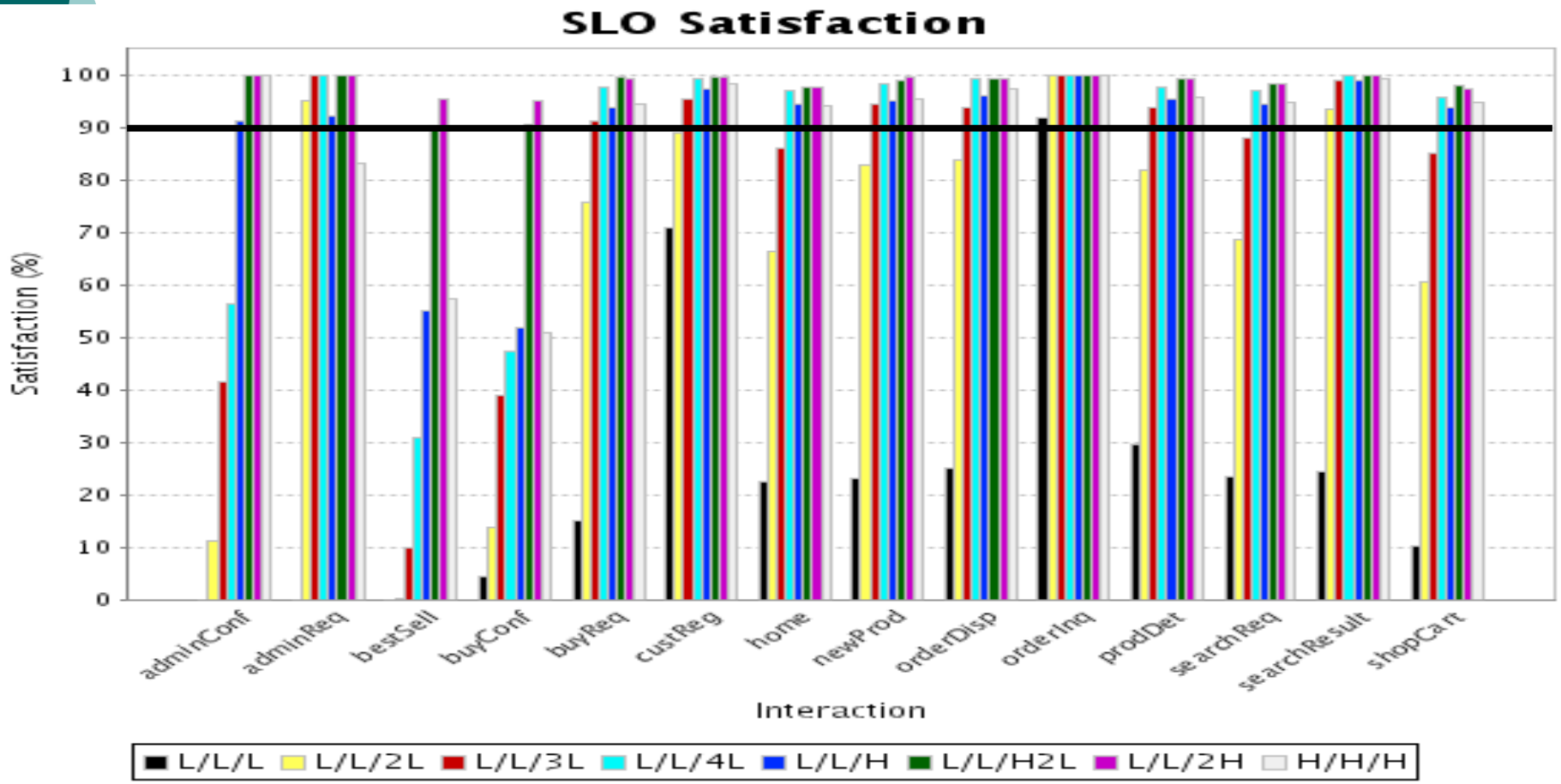


TPC-W Measurements (H/2H)

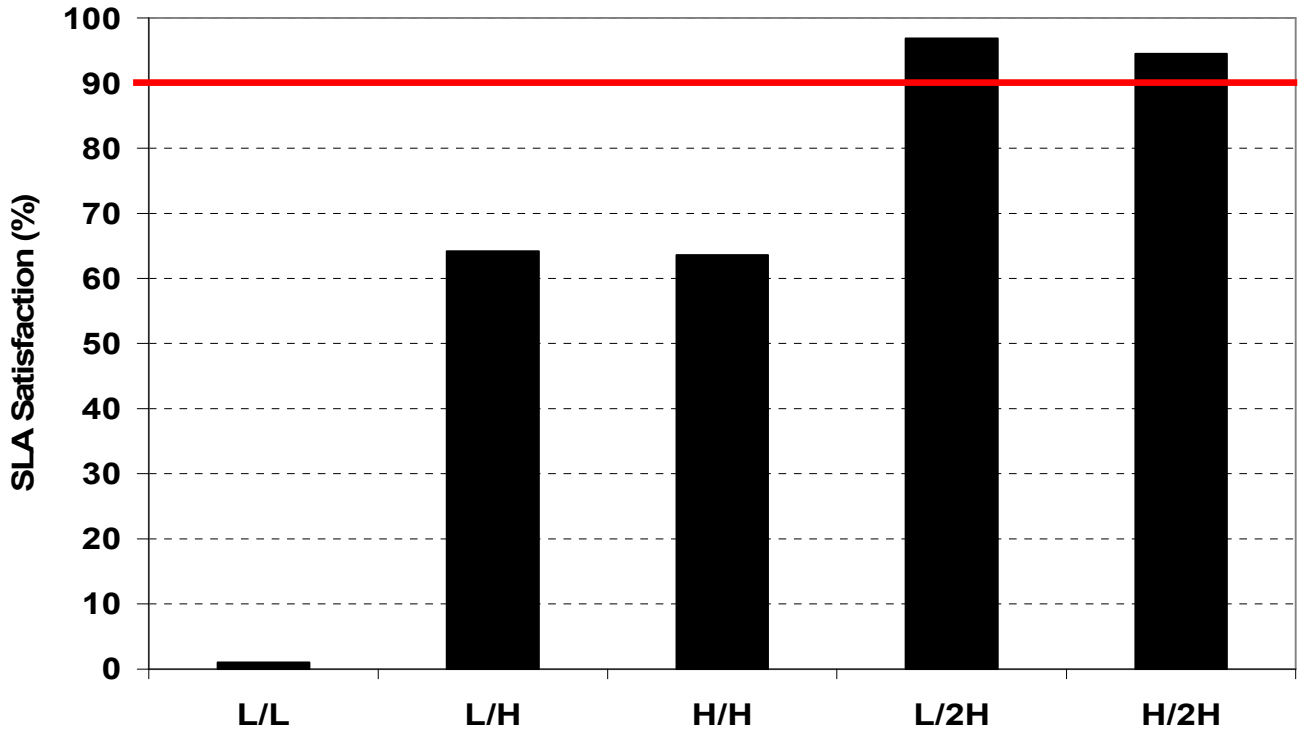


Summary of SLA satisfaction (% of interactions)

TPC-W: 3-Tier Results

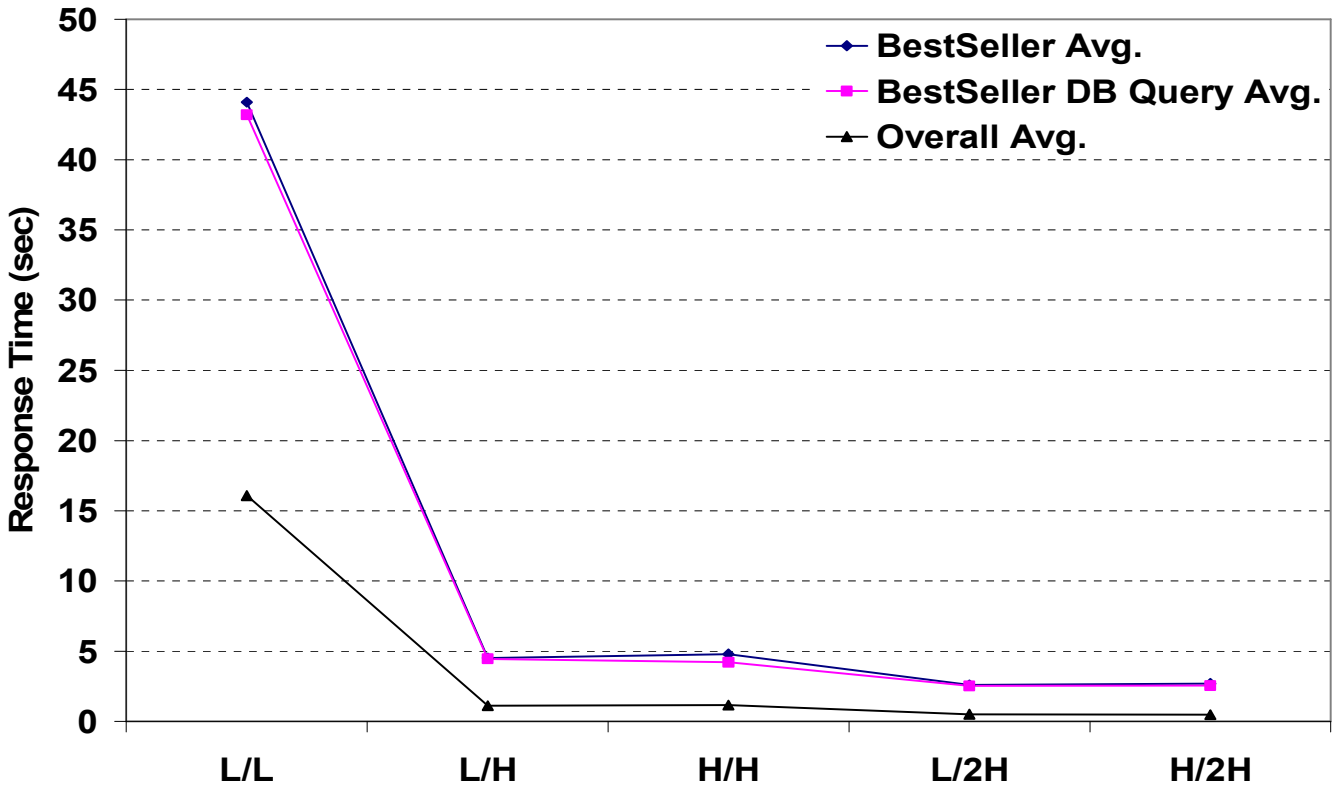


TPC-W: SLA Satisfaction



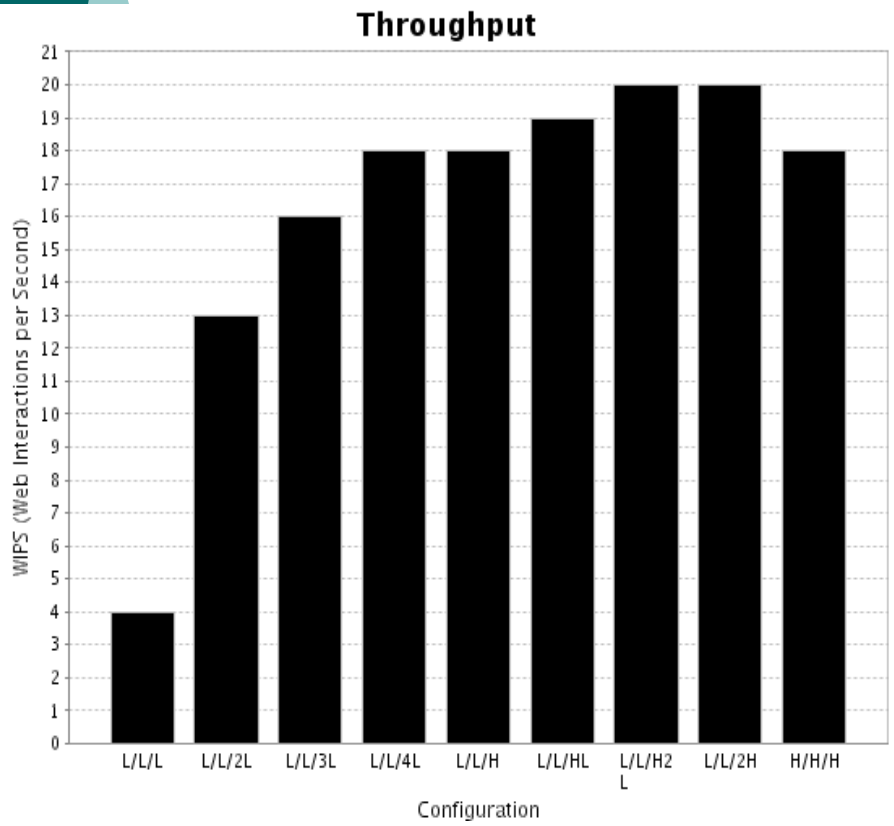
BestSeller SLA satisfaction (% of interactions)

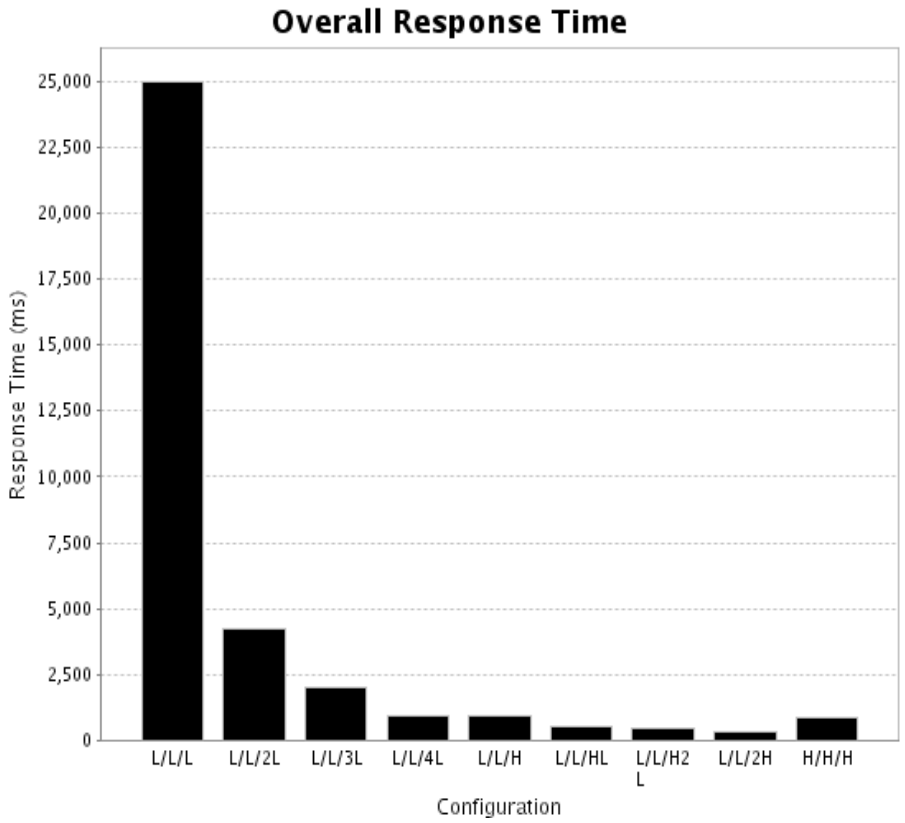
TPC-W: Response Time



BestSeller Average Response Time

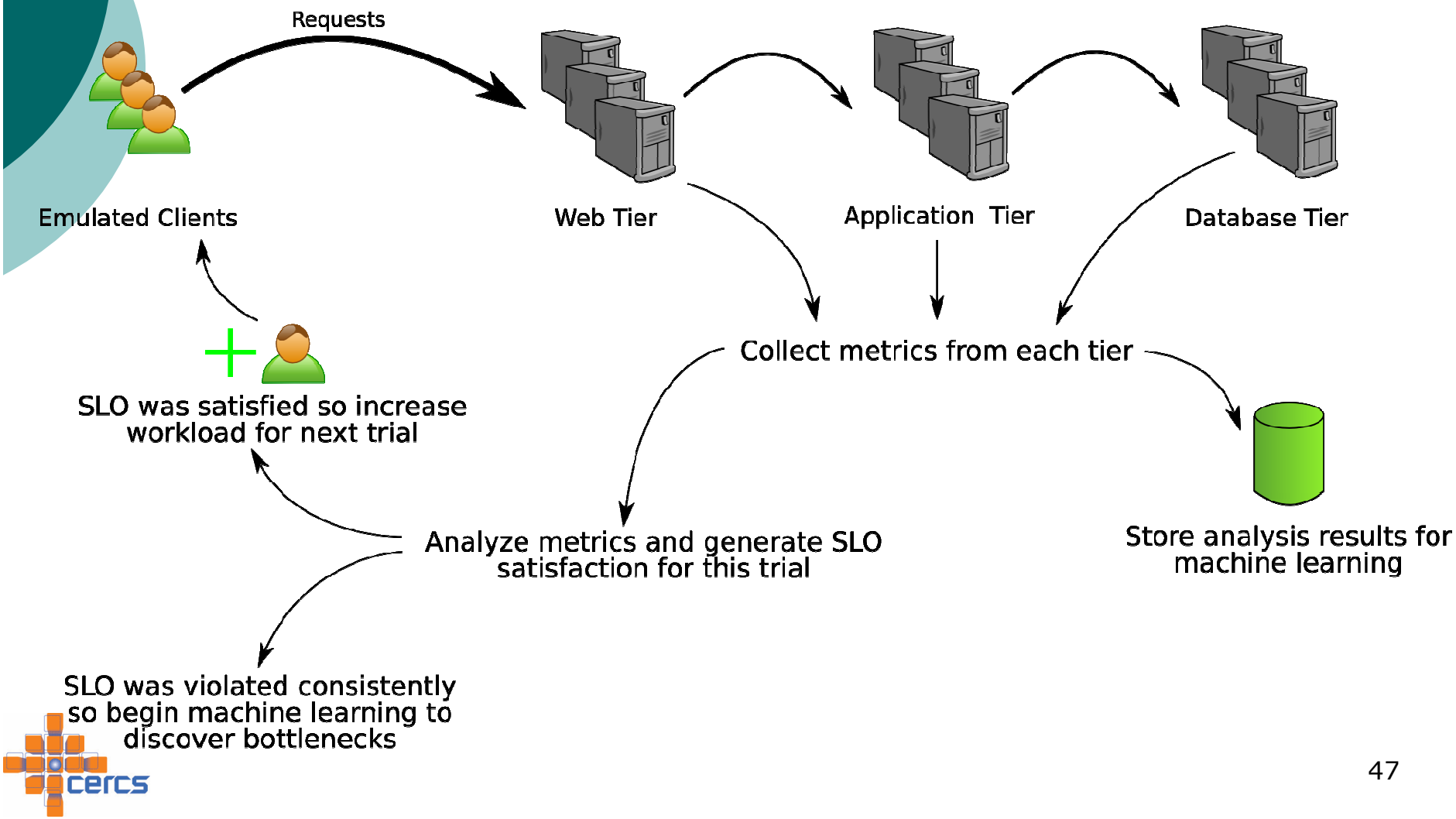
TPC-W 3-Tier Results





Iterative staging results: WIPS and overall average response time

Experiments: Scale Up & Out



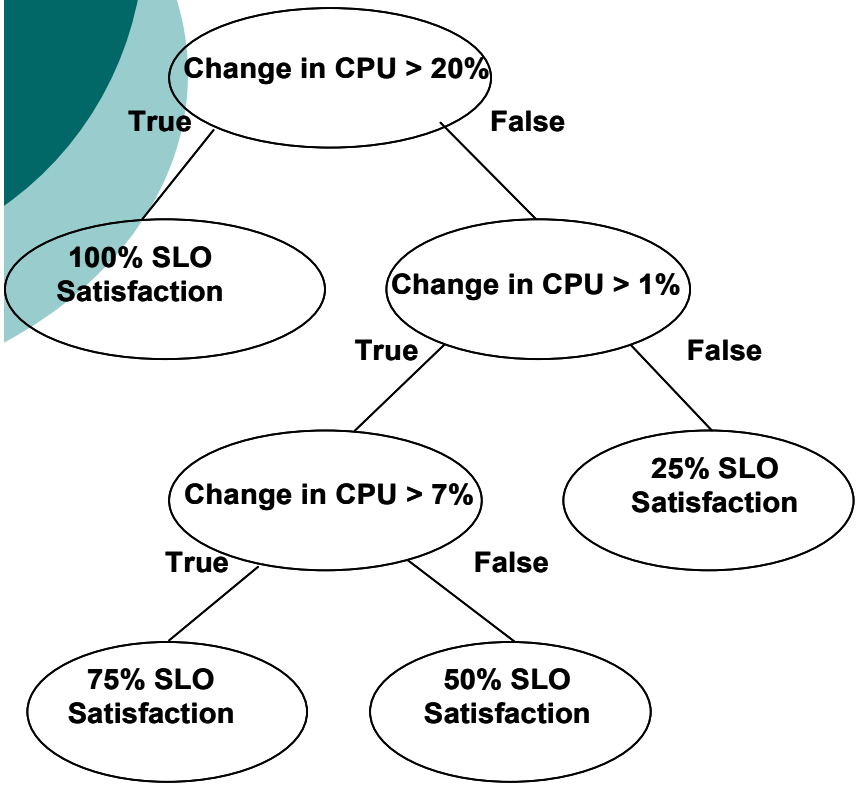
Automation of Analysis

- Training the machine learning classifiers with collected metric data and the SLO satisfaction
  - Each trial becomes a training instance of which each collected metric (e.g., CPU and memory) is an attribute
  - Use a ‘delta’ approach to find derivatives that indicate changes in measured data

Finding Bottlenecks

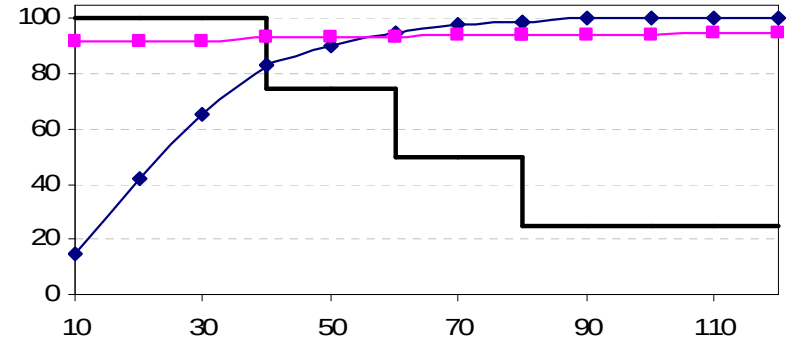
- Query the generated models to find candidate bottleneck metrics
  - Multiple classifiers tested
  - Decision tree chosen
- Each metric is tested w.r.t. SLO satisfaction
  - A metric able to skew the predicted SLO satisfaction from the actual SLO satisfaction is a potential bottleneck

Learning Classifier Detection

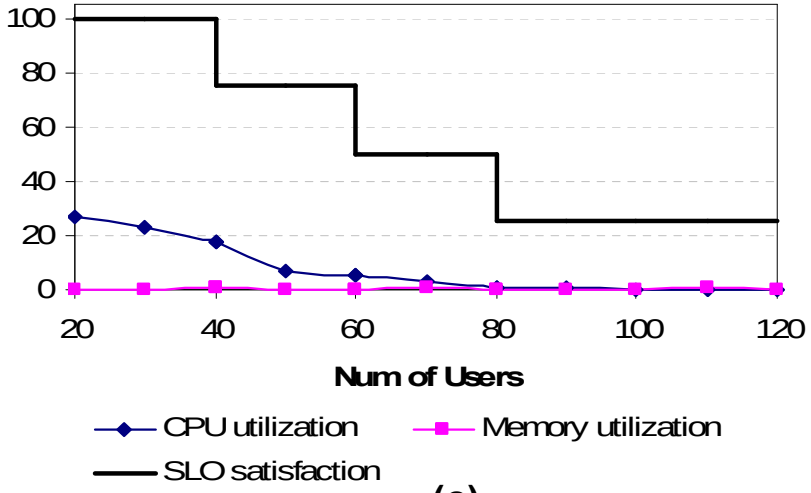


(a)

Decision Tree Classifier



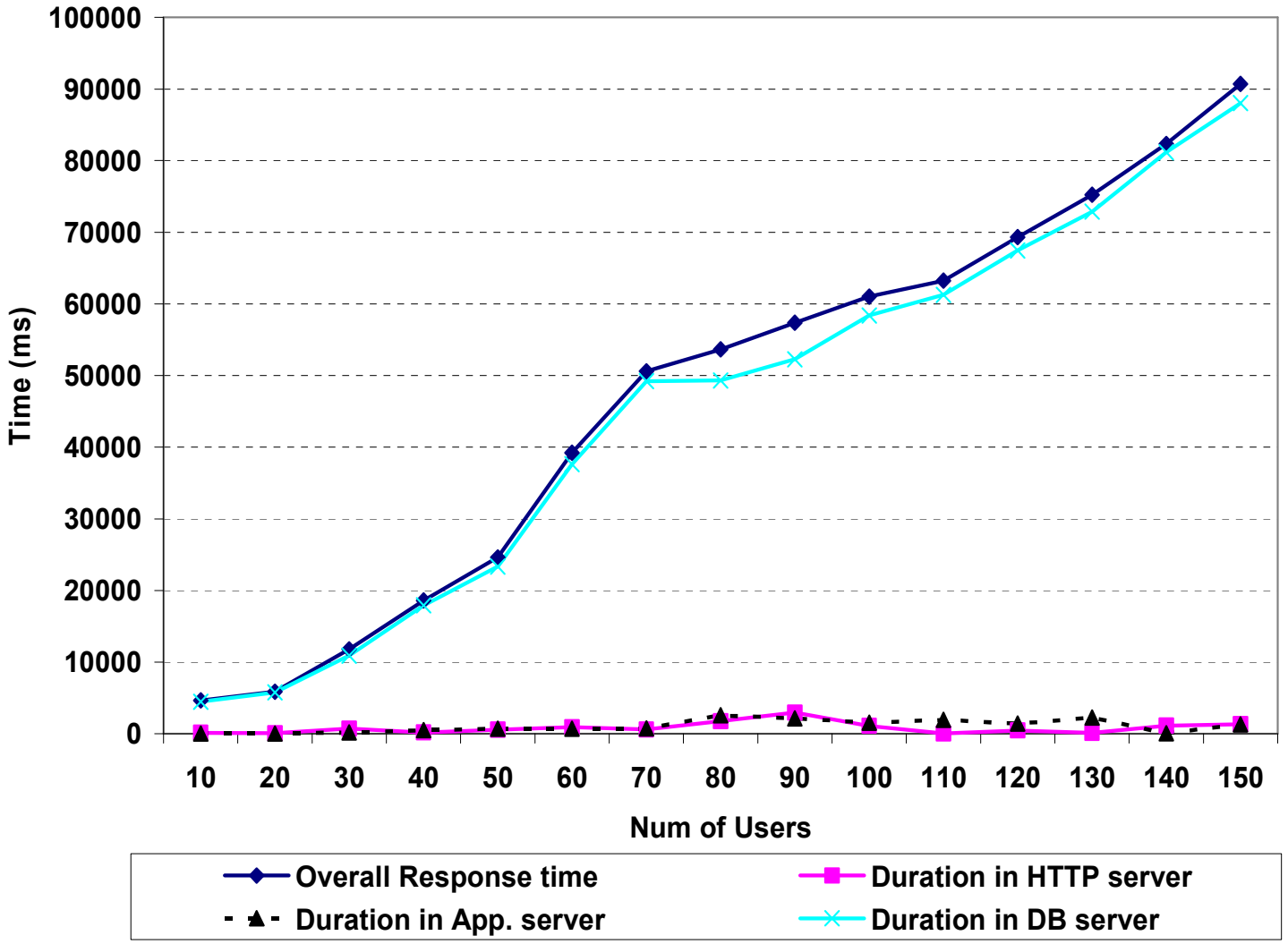
(b)



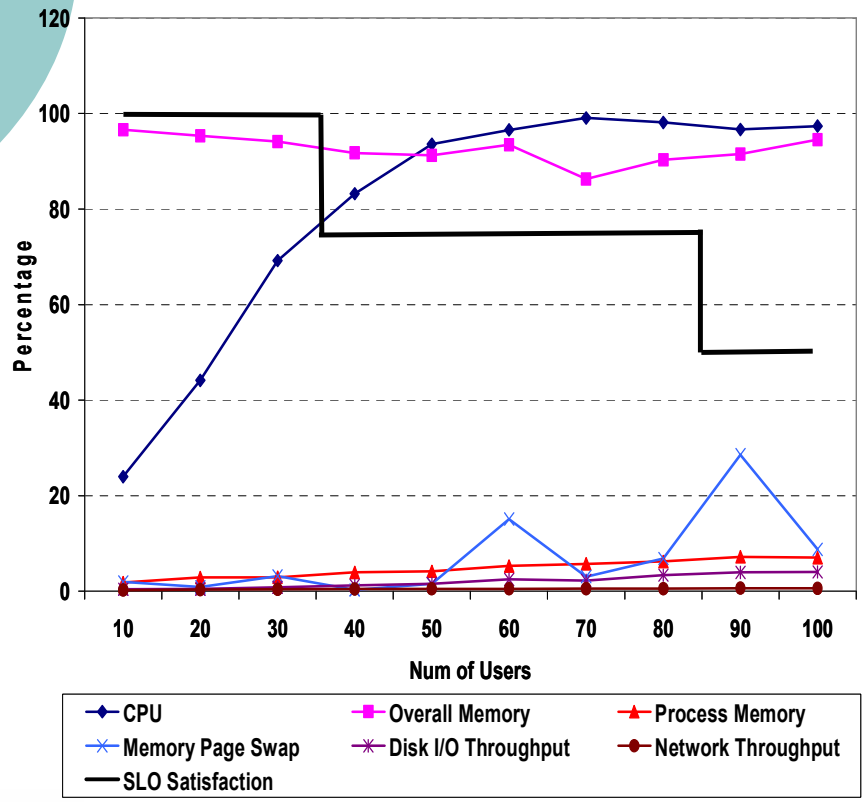
(c)

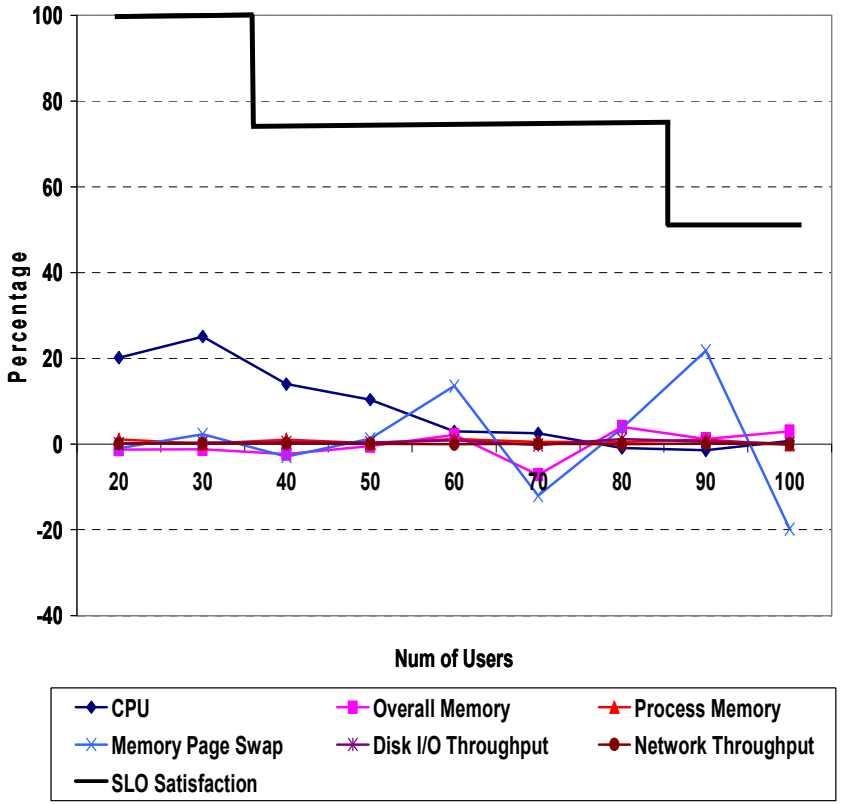
Bottleneck Detection

BestSeller Bottleneck (DB)

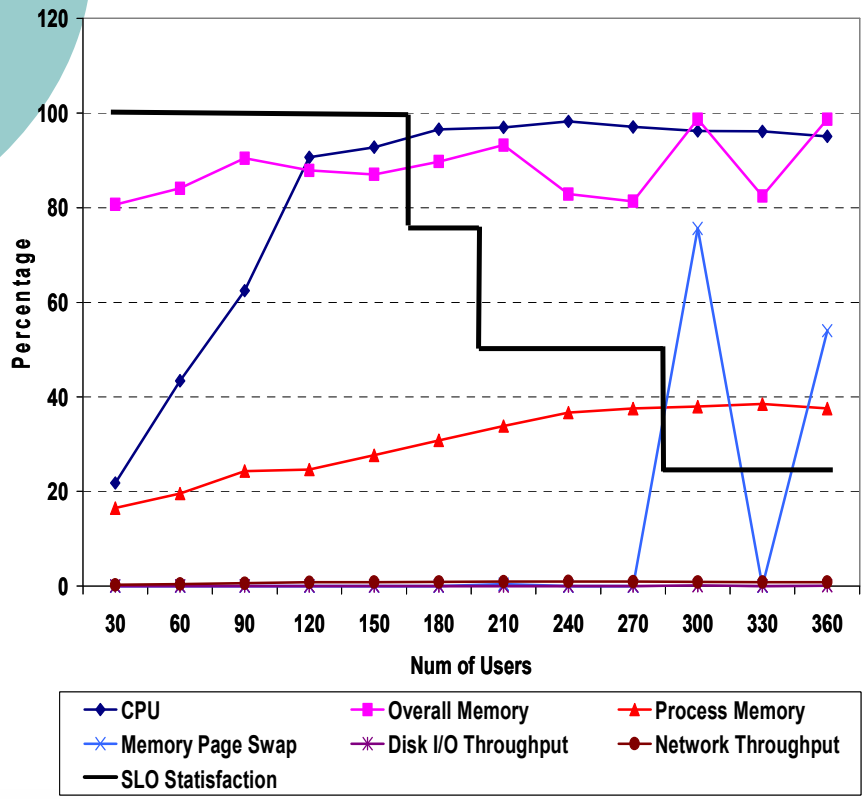


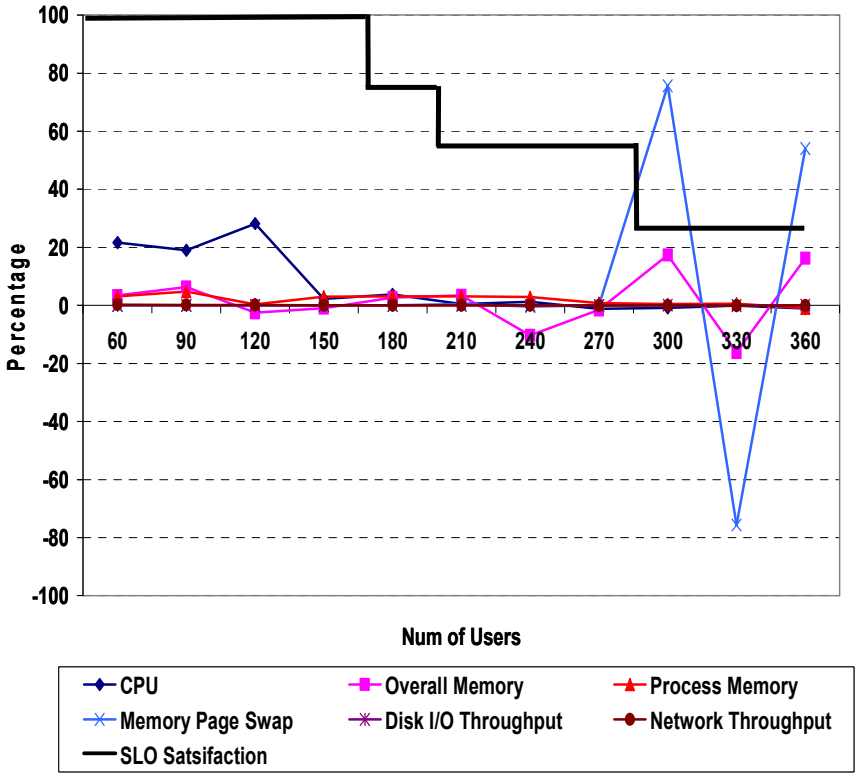
TPC-W Observed Metrics



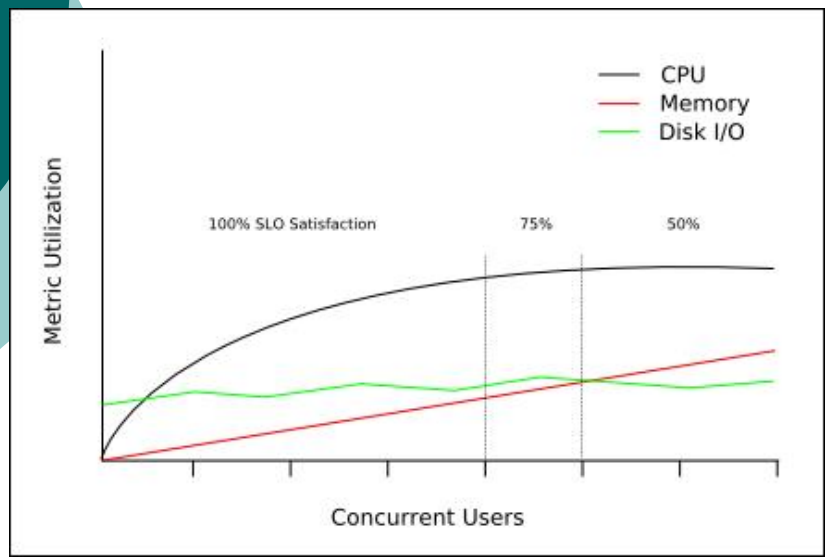


RUBiS Observed Metrics





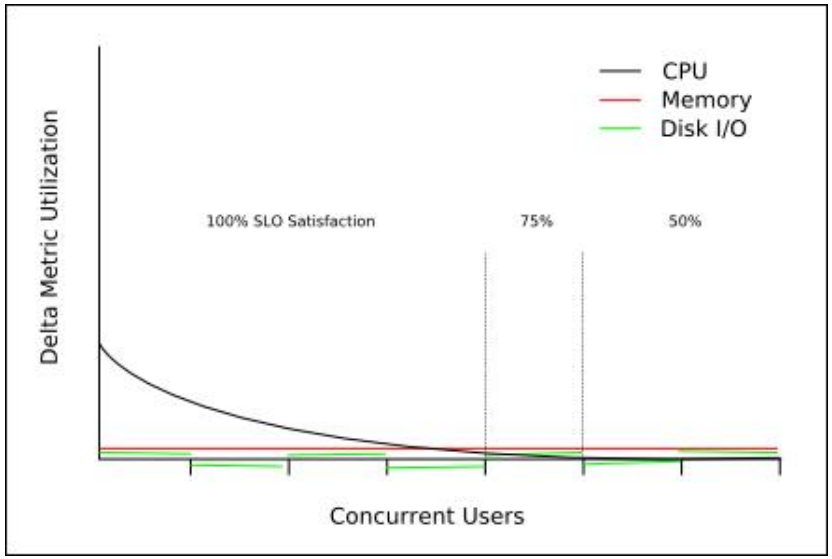
The Delta Training



The raw metric values over the workload

The bottleneck detection process converts metric values into delta metric values for training which reveals the trends of metrics rather than their raw values

The delta (derivative) metric values



Using the delta metric values, the machine learning is able to correlate metrics that either slowly peak to maximum utilization and ignore those that have room to grow

Querying the Generated Model

Sample scenario with only three metrics
Metric values

Workload	CPU	Memory	Network	**SLO Satisfaction**
50	35%	52%	10%	**100%**
100	60%	56%	10%	**100%**
150	80%	60%	11%	**100%**
200	90%	63%	11%	**100%**
250	96%	67%	12%	**95%**
300	99%	72%	12%	**85%**
350	99%	75%	12%	**70%**
400	99%	78%	12%	**40%**

* *Only displayed for comprehension*
* **Prediction attribute**

Training set
Delta metric values

Workload	CPU	Memory	Network	**SLO Satisfaction**
100-50	25%	4%	0%	**100%**
150-100	20%	4%	1%	**100%**
200-150	10%	3%	0%	**100%**
250-200	4%	4%	1%	**100%**
300-250	3%	5%	0%	**95%**
350-300	0%	3%	0%	**85%**
400-350	0%	3%	0%	**70%**

Base test instance

25%	4%	0%	**100%**

Test set

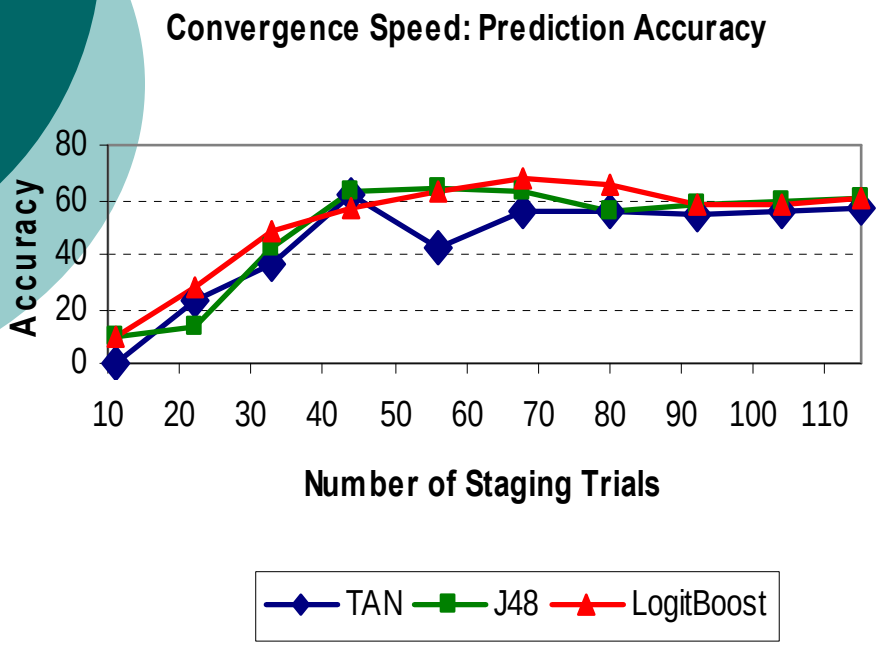
Metric Being Tested	CPU	Memory	Network	**Predicted SLO Satisfaction**
CPU	0%	4%	0%	**70%**
Memory	25%	3%	0%	**100%**
Network	25%	4%	0%	**100%**

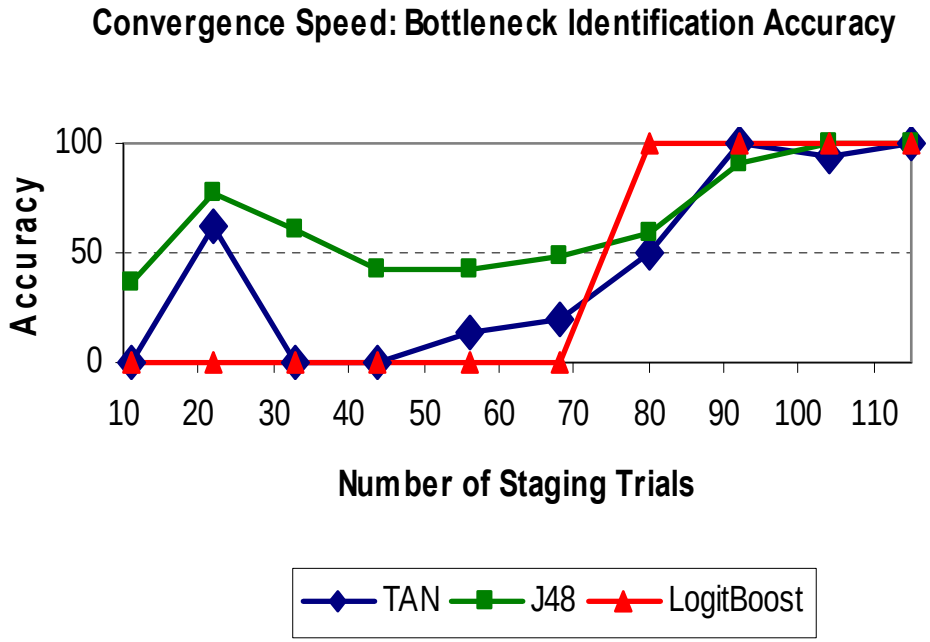
Since the test instance corresponding to CPU has **predicted SLO** different from the base test instance's **SLO satisfaction**, it is considered a candidate bottleneck. The other test instances' **predicted SLO** is the same as the base test instance's **SLO** showing these corresponding metrics do not have a correlation to SLO satisfaction

Evaluation Environment

- RUBiS (Rice University Bidding System)
  - Online auctions (eBay, Yahoo Auctions, etc.)
- Servers
  - Intel Xeon 2.8Ghz, 1GB RAM, 100Mb LAN
- Software
  - Web tier: Apache 2.0.54
  - Application server tier: JOnAS 4.4.6 / Tomcat 5.5.12
  - Database tier: MySQL 3.23.58
- Machine Learning Classifiers (via WEKA Toolkit)
  - Tree-augmented Naïve Bayesian Network
  - LogitBoost
  - C4.5 Decision Tree (J48 implementation)

Evaluation Results





- Convergence speed reveals how many trials are necessary to provide strong results in bottleneck identification
- All three classifiers required at least 44 trials to reach the stabilized state; however, it was not until 90 trials that each classifier begins to discover the real bottlenecks of the system
- Out of the three classifiers, the most stable seems to be the J48 decision tree as it usually has higher bottleneck identification accuracy

Summary of Elba Project

- Application management growing complex
  - Design, deploy, evaluate, reconfigure
  - Staging followed by production deployment
  - Need to automate the entire process
- Automated deployment code generation
  - Translate CIM/MOF into SmartFrog [DSOM'05]
  - Translate CIM/MOF and SLA specs into application settings for staging [NOMS'06]
  - Monitor and evaluate (DSOM'06)

Current and Future Work

- Current work
  - Benchmarks (RUBiS, TPC-W, RUBBoS, DSOM'05, NOMS'06, DSOM'06)
  - Workflow synchronization and dependency management (CEC'06, ICWS'06, ICDE'07)
- Future work
  - Reconfigure and Redesign (step 4)
  - More real applications (e.g., network and data center management)