UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Computer Systems Modelling
Case Studies

Samuel Kounev
Systems Research Group
University of Cambridge – Computer Laboratory

Lent Term 2008

Goals

- Present some practical performance modelling case studies demonstrating:
  - Modelling of real-world distributed systems
  - Workload characterization
  - Model development and validation
  - Model analysis tools/techniques
  - Performance prediction and capacity planning
- Discuss trade-offs in using different modelling formalisms
  - Queueing networks
  - Stochastic Petri nets

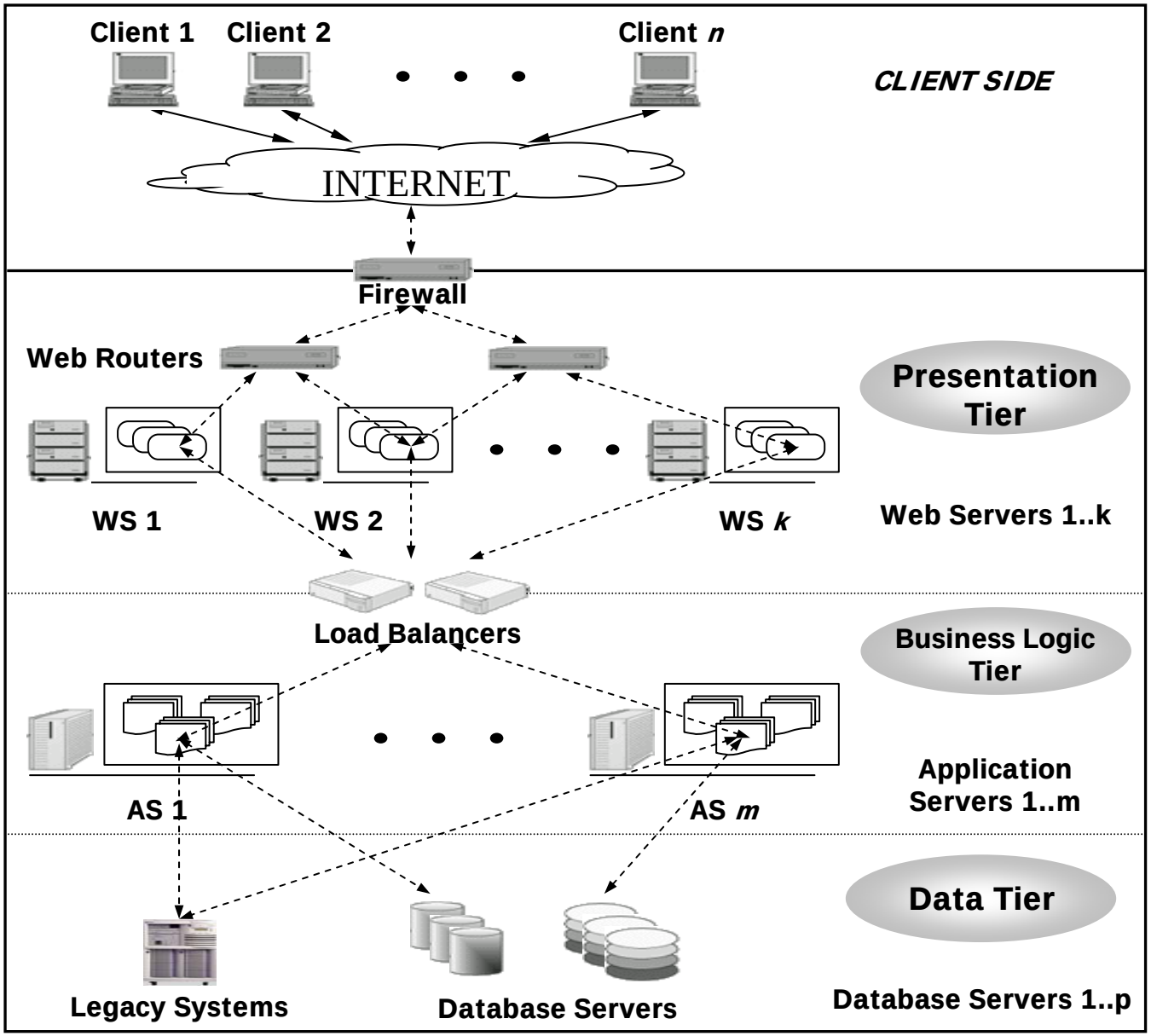
Motivation

- Modern E-Business systems gaining in size and complexity.
- Quality of service requirements of crucial importance!
- Hard to estimate the size and capacity of the deployment environment needed to meet Service Level Agreements (SLAs).
- Deployers faced with questions such as the following:
  - Does the system scale? Are there **potential bottlenecks**?
  - What is the **maximum load**, the system is able to handle?
  - What would be the **avg. response time, throughput and utilization** under the expected workload?

Roadmap

- Case Study 1: Modeling a realistic e-business system by means of queueing networks solved using analytical methods.
- Case Study 2: Modeling a *small* e-business application by means of queueing Petri nets solved using structured analysis methods.
- Case Study 3: Modeling a *large* representative e-business system by means of queueing Petri nets solved using simulation techniques.

Sizing and Capacity Planning



If (n = 1000)

k=? m=? p=?

so that all

SLAs

are fulfilled.

Approaches to Performance Prediction

Educated Guess

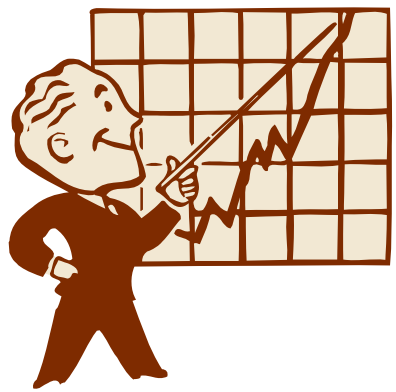
+ Quick, easy and cheap.
- Very inaccurate and risky.

Load Testing (brute force)

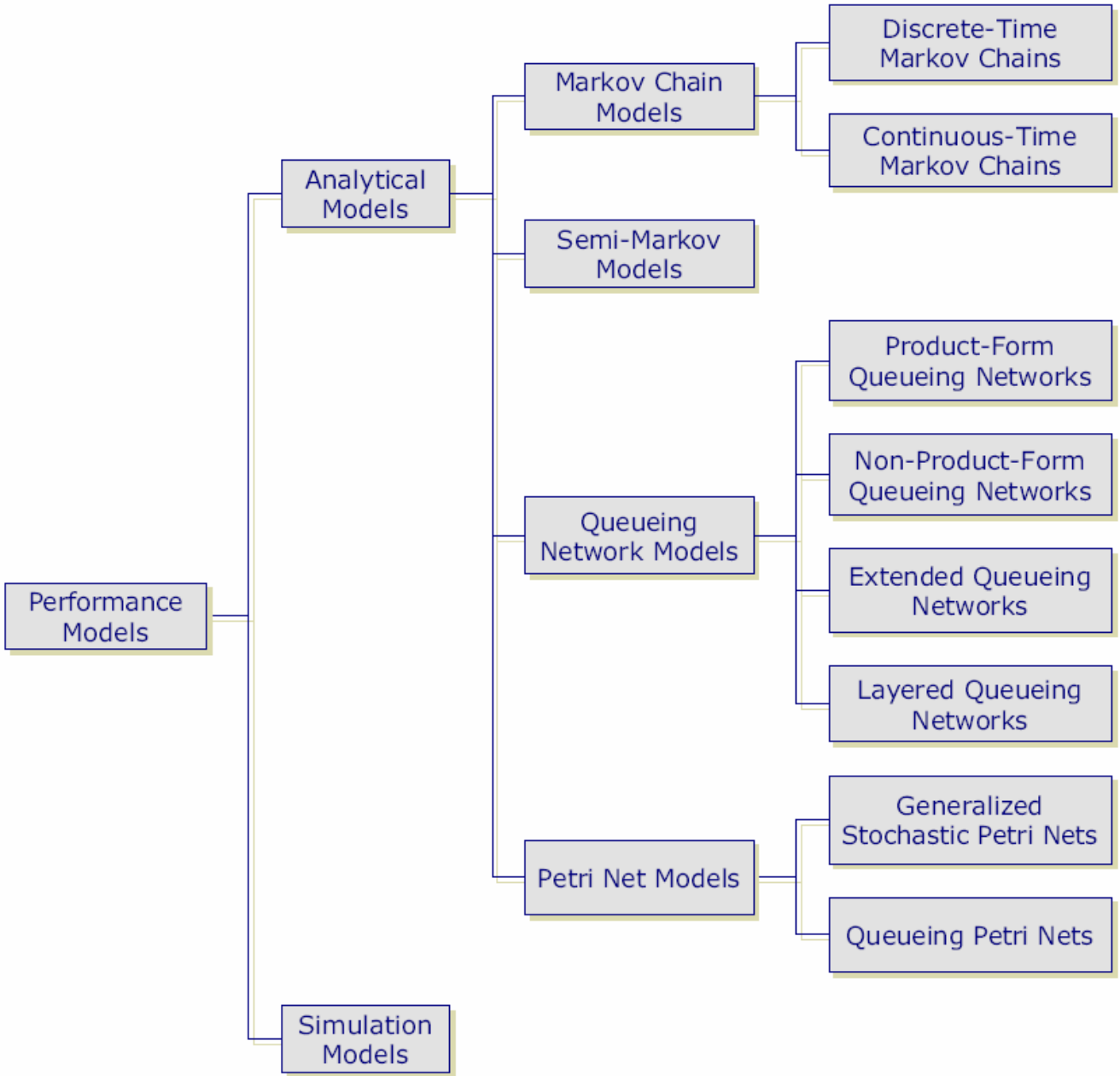
+ Accurate. Helps to identify bottlenecks and fine-tune system prior to production.
- Expensive and time-consuming.
  Assumes system availability for testing.

Performance Modelling

+ Cheaper and quicker than load-testing.
  Could be applied at the design stage.
- Accuracy depends on model representativeness.



Space of Performance Models



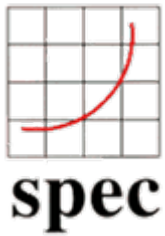
Case Study 1: SPECjAppServer2002 (SjAS02)

- Will study a deployment of SPECjAppServer2002
- Industry standard application server benchmark
- Measures performance and scalability of J2EE app. servers
- Heavy-duty synthetic B2B e-commerce workload
- More info at http://www.spec.org/osg/jAppServer/

- Case study taken from *"Performance Modeling and Evaluation of Large-Scale J2EE Applications"* by S. Kounev and A. Buchmann, Proceedings of the 29th International CMG Conference, 2003.

  http://www.dvs.tu-darmstadt.de/publications/pdf/03-cmg-SPECjAS02_QN.pdf

OSG Java Subcommittee



OSG Java Subcommittee

IBM

intel

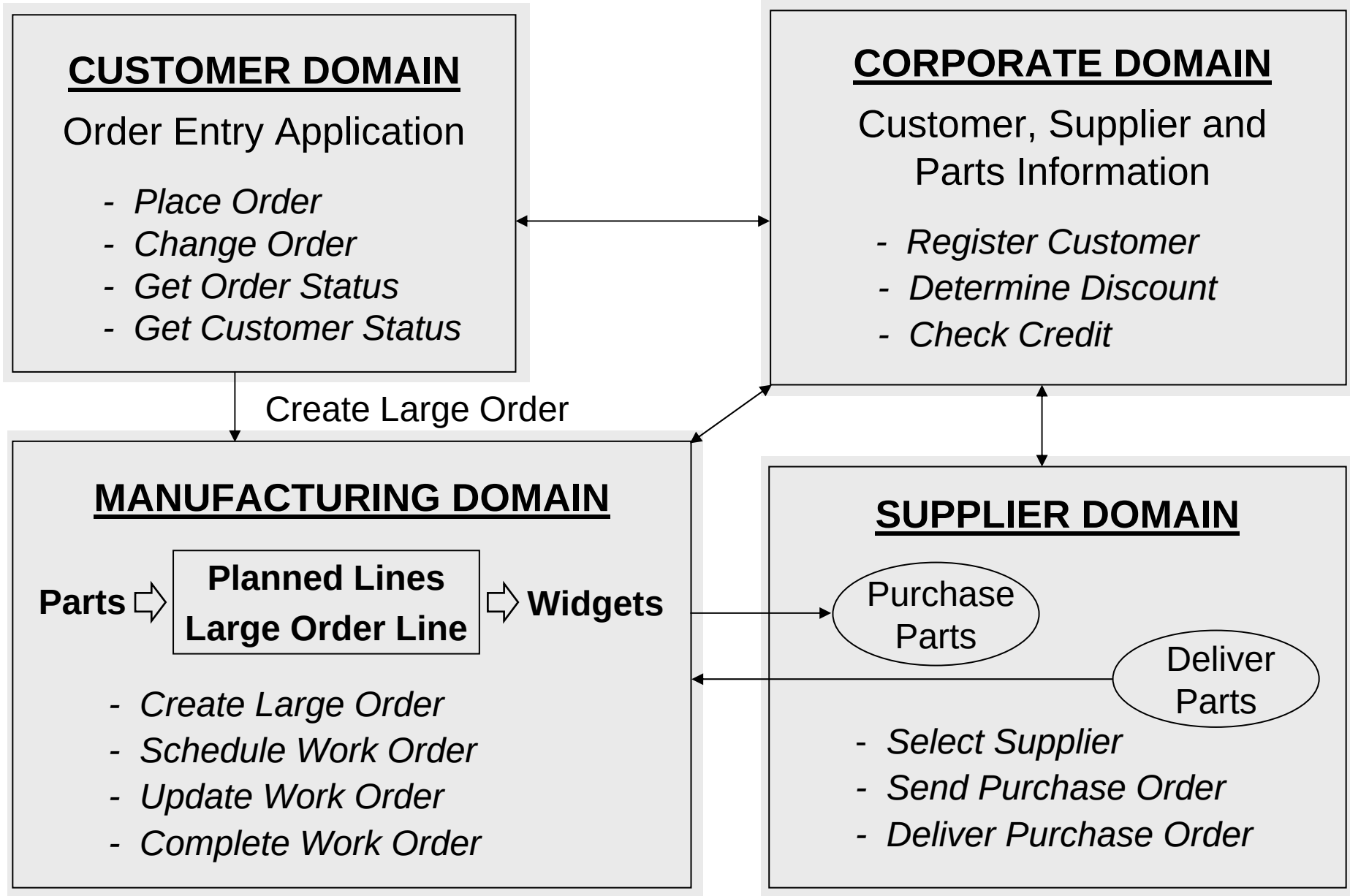
AMD

TECHNISCHE UNIVERSITÄT DARMSTADT

Borland

ORACLE

SPECjAppServer Business Domains



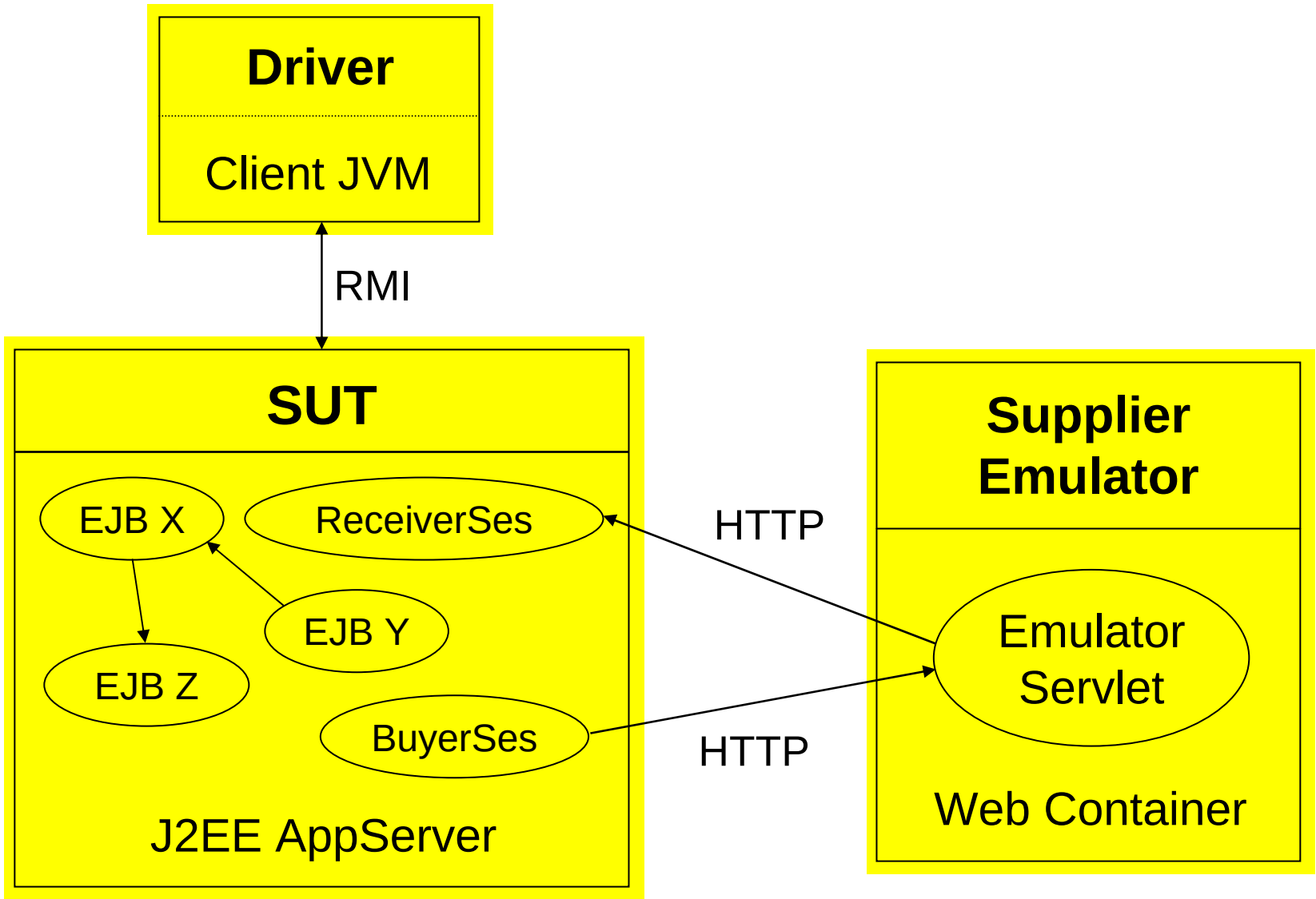
SPECjAppServer Application Design

Benchmark Components:

1. ***EJBs*** – J2EE application deployed on the *System Under Test (SUT*)
2. ***Supplier Emulator*** – web application simulating external suppliers
3. ***Driver*** – Java application simulating clients interacting with the system

- RDBMS used for persistence
- Throughput is function of chosen *Transaction Injection Rate*
- Performance metric is **TOPS = Total Ops Per Second**

SPECjAppServer2002 Components



Goals of the Case Study

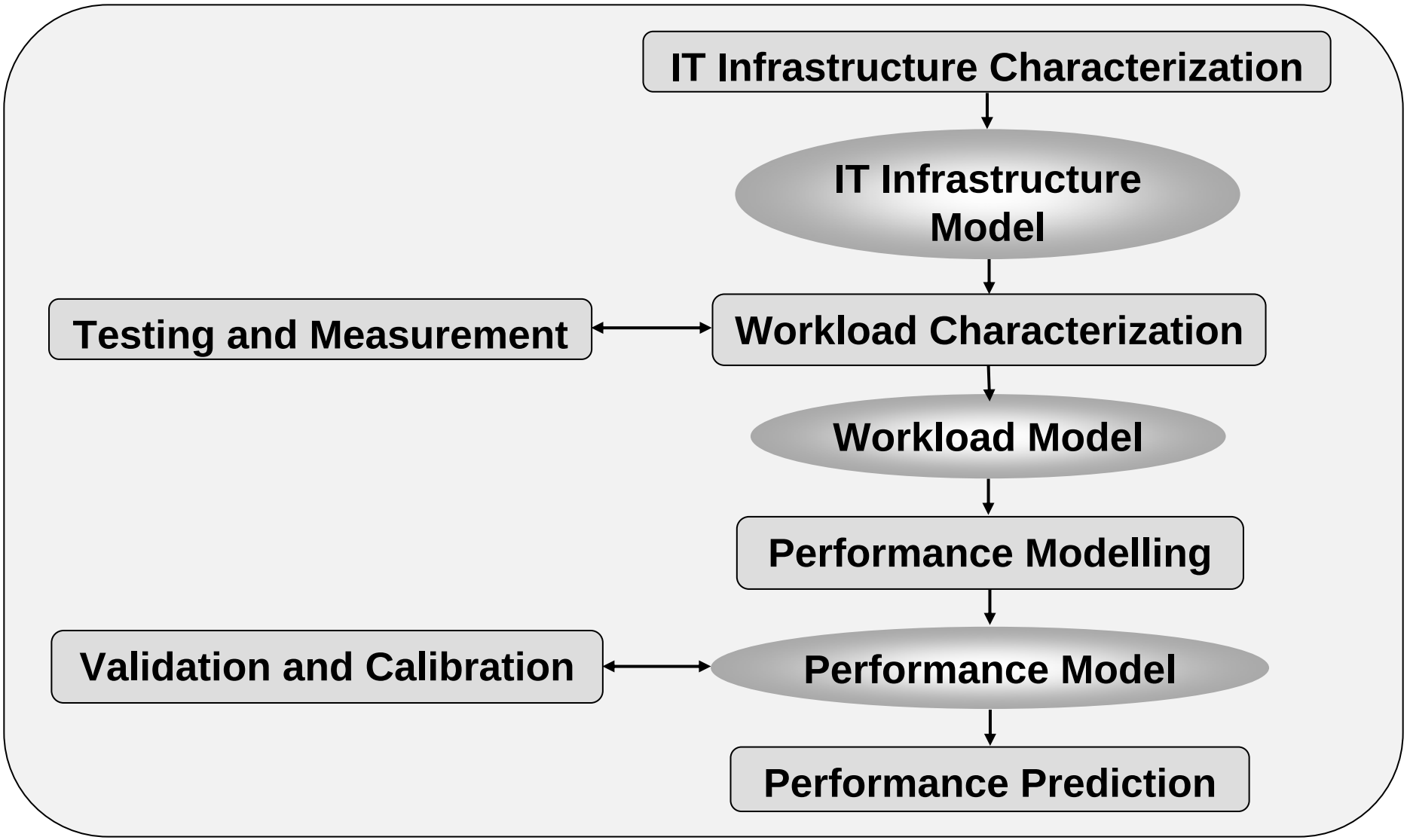
Deployed SPECjAppServer2002 in:

- Cluster of WebLogic Servers (WLS) as a J2EE container
- Using a single Database Server (DBS) for persistence

Interested in knowing:

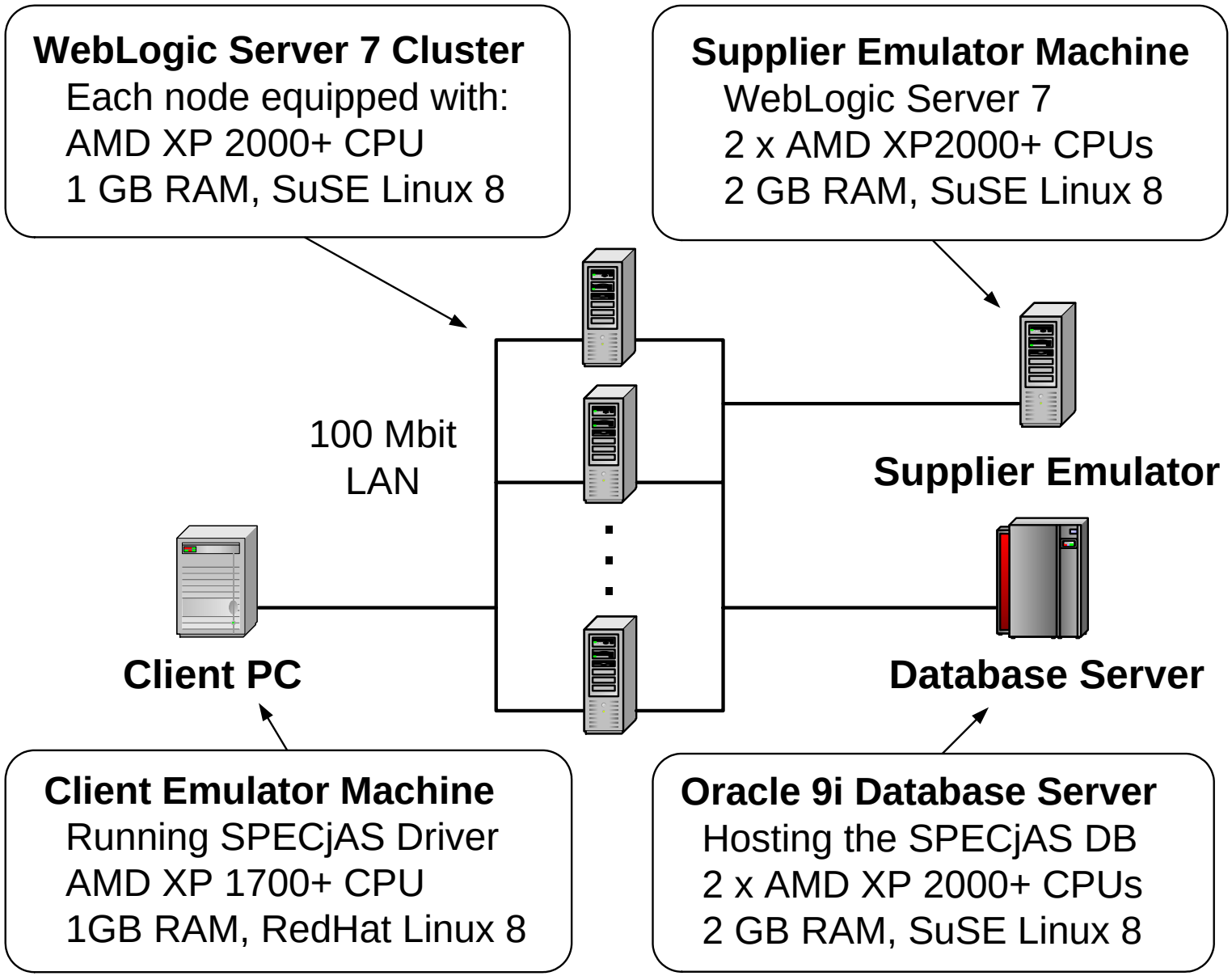
- How many WLSs are needed to guarantee adequate performance under the expected workload?
- For a given number of WLSs, what would be the average trans. response time, throughput and server utilization?
- Will the capacity of the DBS suffice to handle the load?

Capacity Planning Methodology (Menasce et al.)



RESOURCE MODEL

IT Infrastructure Characterization



Workload Characterization

Goal: To describe the system workload in a qualitative and quantitative manner:

1. Describe the types of requests that are processed by the system (called **request classes**).
2. Identify the hardware and software **resources used** by each request class.
3. Measure the total amount of service time (called **service demand**) for each request class at each resource.
4. Specify number of requests of each class that the system will be exposed to (called **workload intensity**).

Workload Characterization – Step 1

We identify the following five **request classes**:

Order Entry Application

1. NewOrder (NO) – places a new order in the system
2. ChangeOrder (CO) – modifies an existing order
3. OrderStatus (OS) – retrieves the status of a given order
4. CustStatus (CS) – lists all orders of a given customer

Manufacturing Application

5. WorkOrder (WO) – the unit of work at the production lines

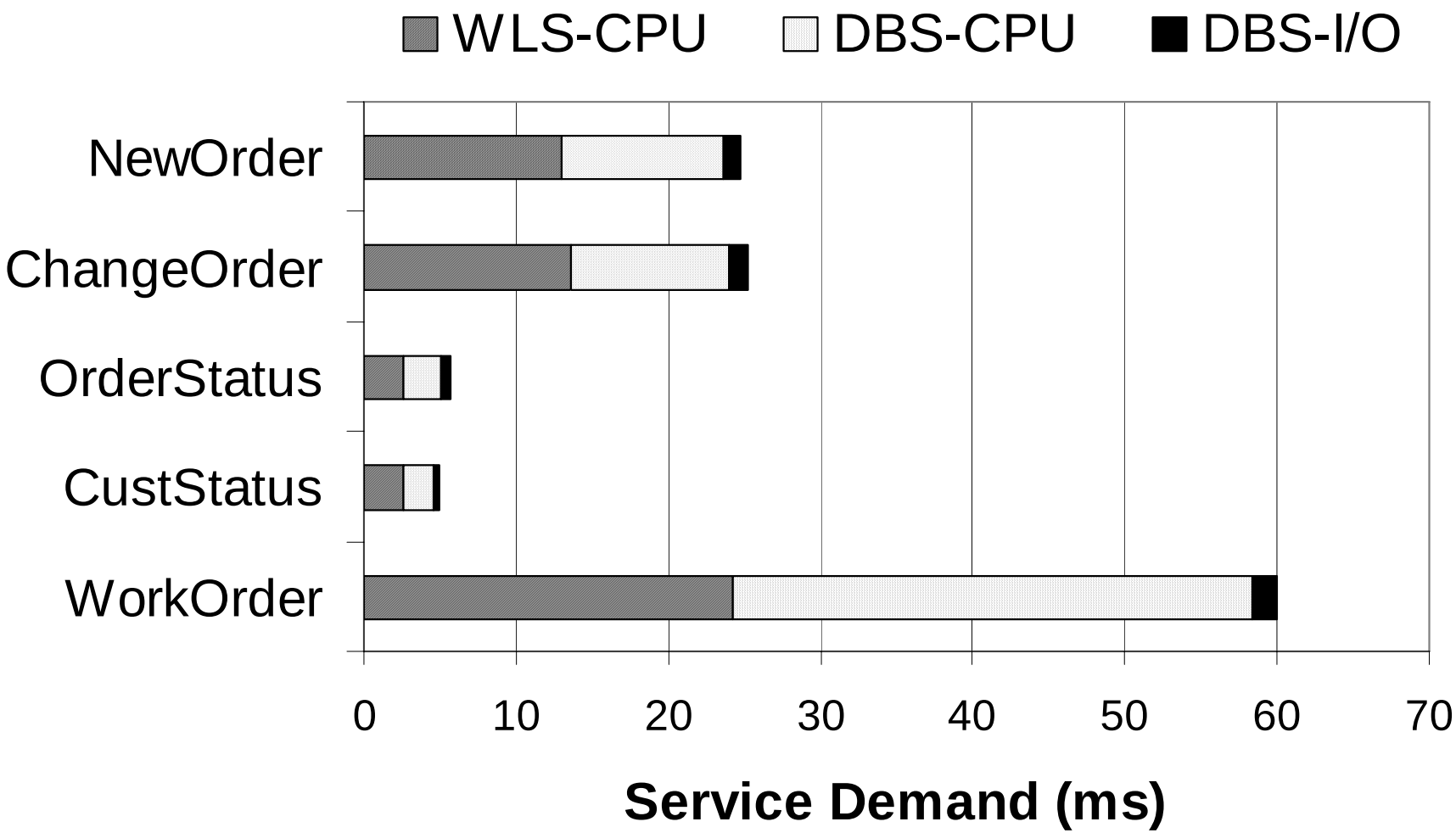
Workload Characterization – Step 2

We identify the following **resources**:

1. The CPU of a WebLogic Server (WLS-CPU)
2. The Local Area Network (LAN)
3. The CPUs of the database server (DBS-CPU)
4. The disk drives of the database server (DBS-I/O)

We ignore network service demands, since over a 100 Mbit LAN all communication times were negligible.

Workload Characterization – Step 3



Workload Characterization – Step 4

Workload intensity is usually specified in one of two ways:

- Average request arrival rates (open QNs)
- Average number of requests in the system (closed QNs)

In our study, we quantify workload intensity by specifying:

- The number of concurrent order entry clients
- The average customer think time – *Customer Think Time*
- Number of planned production lines in the Mfg domain
- Average time production lines wait after processing a WorkOrder before starting a new one – *Mfg Think Time*

Building a Performance Model

We model each processing resource using a queue:

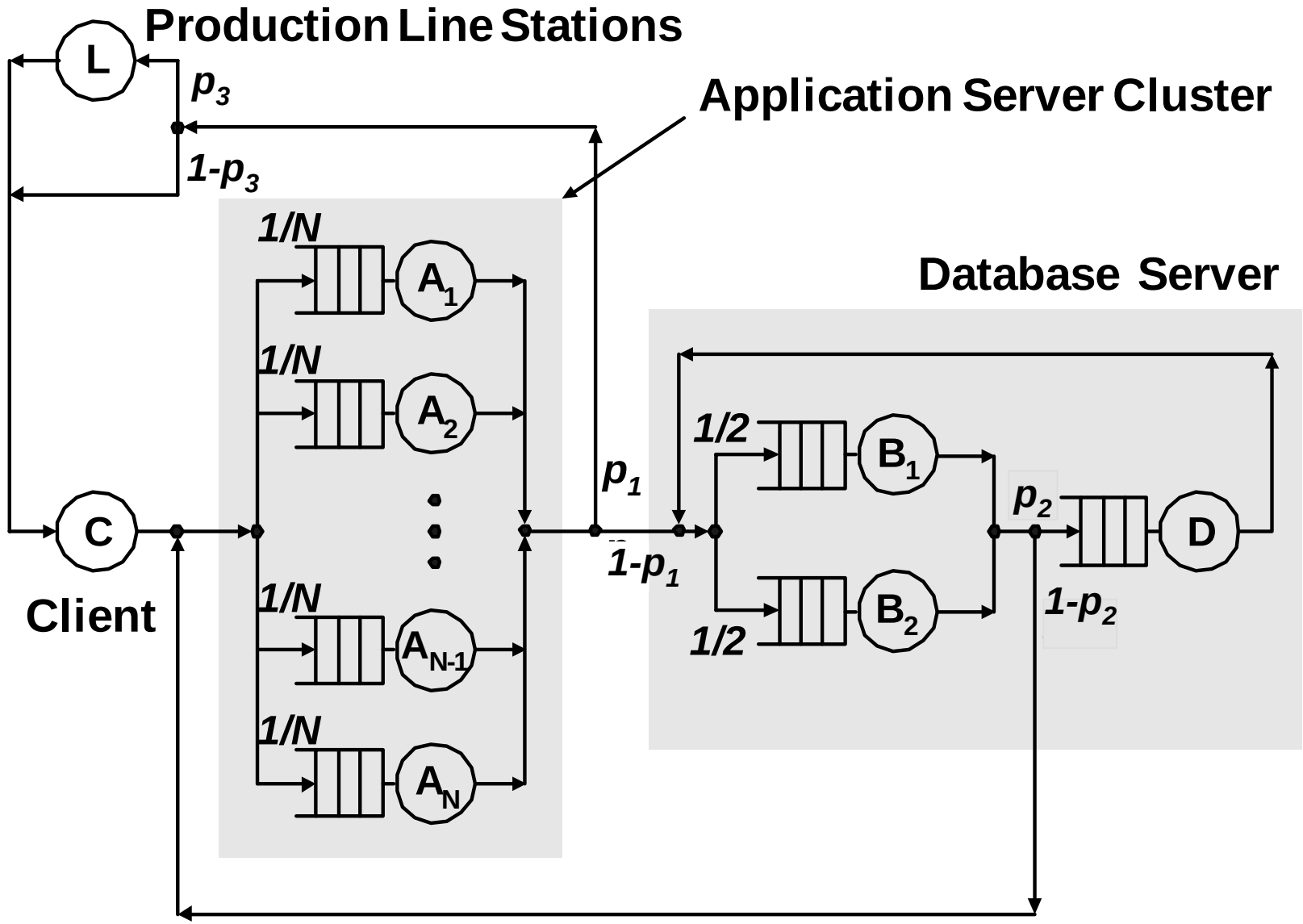
- The CPUs of the *N* WLSs in the cluster – *N* PS queues $\mathbf{A_1}…\mathbf{A_N}$
- The two CPUs of the DBS – 2 PS queues $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$
- The disk drive of the DBS – 1 FCFS queue ***D***

We model the client using a delay resource ***C***

- Delay of order requests at this queue = *Customer Think Time*
- Delay of WorkOrder requests at this queue = *Mfg Think Time*

Each WorkOrder is delayed at the virtual production line stations during processing. To model this we introduce an additional delay resource ***L***.

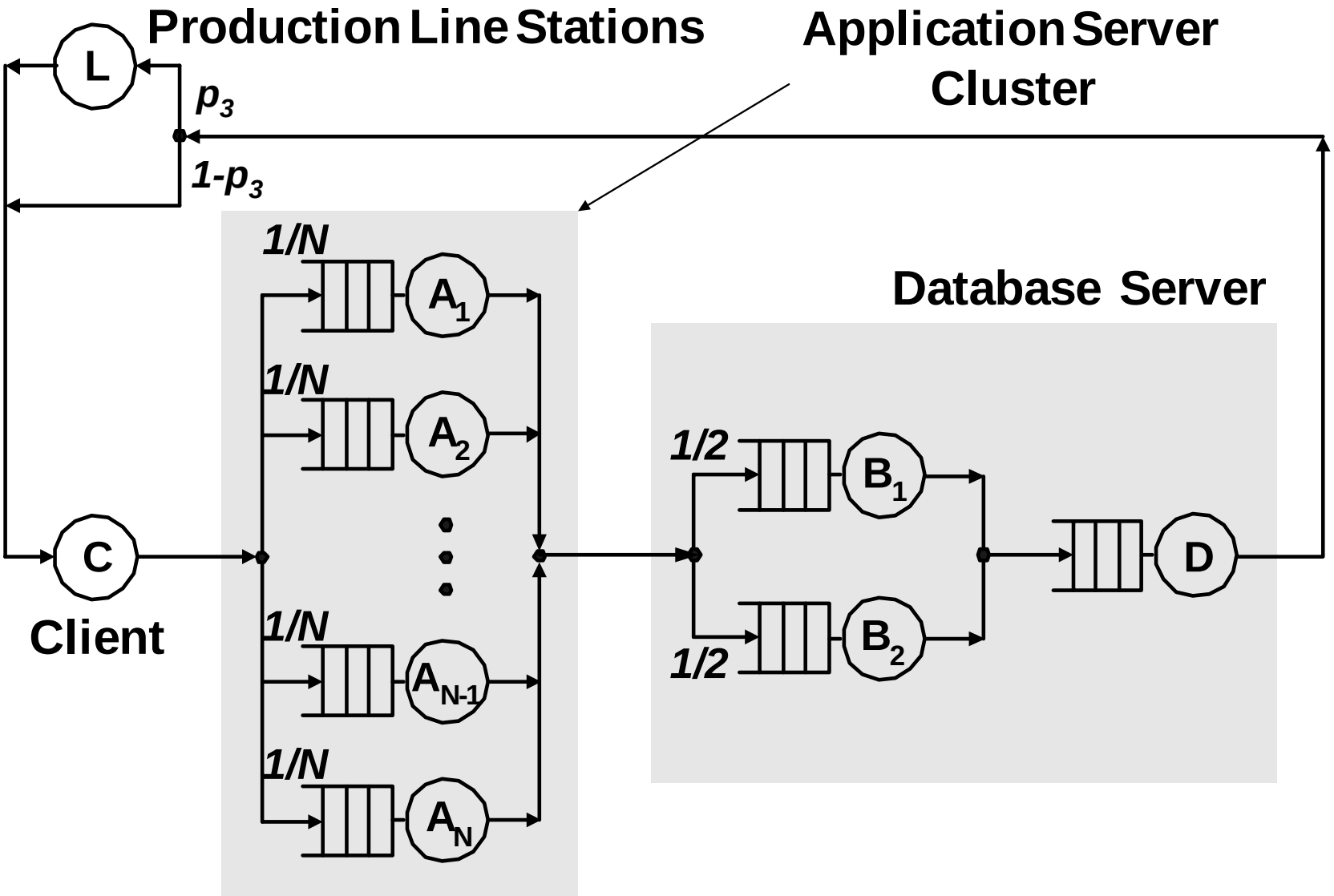
Queueing Network (QN) Model of the System



Formal Queue Definitions (Kendall's Notation)

Queue	Type	Description
$A_1..A_N$	$G/M/1/PS$	WLS CPUs
B_1, B_2	$G/M/1/PS$	DBS CPUs
D	$G/M/1/FCFS$	DBS Disk Subsystem
C	$G/M/\infty/IS$	Client Machine
L	$G/M/\infty/IS$	Prod. Line Stations

Simplified QN Model of the System



Model Input Parameters

1. Number of order entry clients of each type – NewOrder, ChangeOrder, OrderStatus and CustStatus.
2. Average think time of order entry clients – *Customer Think Time.*
3. Number of planned production lines generating WorkOrder requests.
4. Average time production lines wait after processing a WorkOrder before starting a new one – *Mfg Think Time.*
5. Service demands of the 5 request classes at queues A_i, B_k and D (as measured during workload characterization).

Scenarios that we will study

We will analyze several different instances of the model for different workload intensities – **low**, **moderate** and **heavy**:

Parameter	Low	Moderate	Heavy
NewOrder Clients	30	50	100
ChangeOrder Clients	10	40	50
OrderStatus Clients	50	100	150
CustStatus Clients	40	70	50
Planned Lines	50	100	200
Customer Think Time	2 sec	2 sec	3 sec
Mfg Think Time	3 sec	3 sec	5 sec

In each case we will apply the model for different number of application servers – from 1 to 9. We will first consider the case without large order lines in the Mfg domain.

Model Analysis and Validation

We have a **closed non-product-form queueing network** model with five request classes to analyze.

We employed the **PEPSY-QNS tool** from the University of Erlangen-Nuernberg. For more information see:

http://www4.informatik.uni-erlangen.de/Projects/PEPSY/en/pepsy.html

- Available free of charge for non-commercial use
- Supports a wide range of solution methods (over 30)
- Offers both exact and approximate methods

G. Bolch and M. Kirschnick. “The Performance Evaluation and Prediction System for Queueing Networks – PEPSY-QNS”. TR-I4-94-18, University of Erlangen-Nuremberg, Germany, 1994.

Other Queueing Network Analysis Tools

- **Java Modelling Tools – JMT (http://jmt.sourceforge.net/)**
  - JSIMgraph - Queueing network models simulator with graphical user interface
  - JSIMwiz - Queueing network models simulator with wizard-based user interface
  - JMVA - Mean Value Analysis of queueing network models
  - JABA - Asymptotic Analysis of queueing network models

- **SHARPE (http://www.ee.duke.edu/~kst/software_packages.html**

  C. Hirel, R. A. Sahner, X. Zang, and K. S. Trivedi. “Reliability and Performability Modeling Using SHARPE 2000”. In Computer Performance Evaluation / TOOLS 2000, Schaumburg, IL, USA, pages 345–349, 2000. See http://www.ee.duke.edu/~kst/.

- **Daniel Menasce’s MS Excel Workbooks:**
  - http://cs.gmu.edu/~menasce/webbook/index.html
  - http://cs.gmu.edu/~menasce/ebook/index.html
  - http://cs.gmu.edu/~menasce/webservices/index.html
  - http://cs.gmu.edu/~menasce/perfbyd/

Other Queueing Network Analysis Tools

- **OMNeT++ (http://www.omnetpp.org/)**

  Varga. “The OMNeT++ Discrete Event Simulation System”. In Proceedings of the European Simulation Multiconference (ESM'2001). June 6-9, 2001. Prague, Czech Republic. Can be used to simulate queueing networks, see http://www.omnetpp.org/doc/queueing-tutorial.pdf

- **QNAT (http://poisson.ecse.rpi.edu/~hema/qnat/)**

  H. T. Kaur, D. Manjunath, and S. K. Bose. “The Queuing Network Analysis Tool (QNAT)”. In Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, CA, USA, vol. 8, pages 341–347, 2000.

- **RAQS (http://www.okstate.edu/cocim/raqs/)**

  M. Kamath, S. Sivaramakrishnan, and G. Shirhatti. “RAQS: A software package to support instruction and research in queueing systems”. In Proceedings of the 4th Industrial Engineering Research Conference, IIE, Norcross, GA., pages 944–953, 1995.

Scenario 1 (Low Load) with 1 AS

We have **130** concurrent clients and **50** planned production lines.

	1 Application Server		
METRIC	Model	Measured	Error
NewOrder Throughput	14.59	14.37	1.5%
ChangeOrder Throughput	4.85	4.76	1.9%
OrderStatus Throughput	24.84	24.76	0.3%
CustStatus Throughput	19.89	19.85	0.2%
WorkOrder Throughput	12.11	12.19	0.7%
NewOrder Response Time	56ms	68ms	17.6%
ChangeOrder Response Time	58ms	67ms	13.4%
OrderStatus Response Time	12ms	16ms	25.0%
CustStatus Response Time	11ms	17ms	35.2%
WorkOrder Response Time	1127ms	1141ms	1.2%
WebLogic Server CPU Utilization	66%	70%	5.7%
Database Server CPU Utilization	36%	40%	10%

Scenario 1 (Low Load) with 2 AS

	2 Application Servers		
METRIC	Model	Measured	Error
NewOrder Throughput	14.72	14.49	1.6%
ChangeOrder Throughput	4.90	4.82	1.7%
OrderStatus Throughput	24.89	24.88	0.0%
CustStatus Throughput	19.92	19.99	0.4%
WorkOrder Throughput	12.20	12.02	1.5%
NewOrder Response Time	37ms	47ms	21.3%
ChangeOrder Response Time	38ms	46ms	17.4%
OrderStatus Response Time	8ms	10ms	20.0%
CustStatus Response Time	7ms	10ms	30.0%
WorkOrder Response Time	1092ms	1103ms	1.0%
WebLogic Server CPU Utilization	33%	37%	10.8%
Database Server CPU Utilization	36%	38%	5.2%

Scenario 1 (Low Load) Observations

- Response time results much less accurate than throughput and utilization results:
  - Running a **transaction mix** vs. single transaction
  - Additional delays from **software contention**
- The lower the service demand the higher the response time error (e.g. WorkOrder vs. CustStatus)

Scenario 2 (Moderate Load) with 3 and 6 AS

We have **260** concurrent clients and **100** planned production lines.

	3 WebLogic Servers			6 WebLogic Servers		
METRIC	Model	Measured	Error	Model	Measured	Error
X_{NO}	24.21	24.08	0.5%	24.29	24.01	1.2%
X_{CO}	19.36	18.77	3.1%	19.43	19.32	0.6%
X_{OS}	49.63	49.48	0.3%	49.66	49.01	1.3%
X_{CS}	34.77	34.24	1.5%	34.80	34.58	0.6%
X_{WO}	23.95	23.99	0.2%	24.02	24.03	0.0%
R_{NO}	65ms	75ms	13.3%	58ms	68ms	14.7%
R_{CO}	66ms	73ms	9.6%	58ms	70ms	17.1%
R_{OS}	15ms	20ms	25.0%	13ms	18ms	27.8%
R_{CS}	13ms	20ms	35.0%	11ms	17ms	35.3%
R_{WO}	1175ms	1164ms	0.9%	1163ms	1162ms	0.0%
U_{WLS}	46%	49%	6.1%	23%	25%	8.0%
U_{DBS}	74%	76%	2.6%	73%	78%	6.4%

Scenario 3 (Heavy Load) with 4 and 6 AS

We have **350** concurrent clients and **200** production lines.

	4 WebLogic Servers			6 WebLogic Servers		
METRIC	Model	Msrd.	Error	Model	Msrd.	Error
X_{NO}	32.19	32.29	0.3%	32.22	32.66	1.3%
X_{CO}	16.10	15.96	0.9%	16.11	16.19	0.5%
X_{OS}	49.59	48.92	1.4%	49.60	49.21	0.8%
X_{CS}	16.55	16.25	1.8%	16.55	16.24	1.9%
X_{WO}	31.69	31.64	0.2%	31.72	32.08	1.1%
R_{NO}	106ms	98ms	8.2%	103ms	94ms	9.6%
R_{CO}	106ms	102ms	3.9%	102ms	98ms	4.1%
R_{OS}	25ms	30ms	16.7%	24ms	27ms	11.1%
R_{CS}	21ms	31ms	32.3%	20ms	27ms	25.9%
R_{WO}	1310	1260ms	4.0%	1305ms	1251ms	4.3%
U_{WLS}	40%	42%	4.8%	26%	29%	10.3%
U_{DBS}	87%	89%	2.2%	88%	91%	3.3%

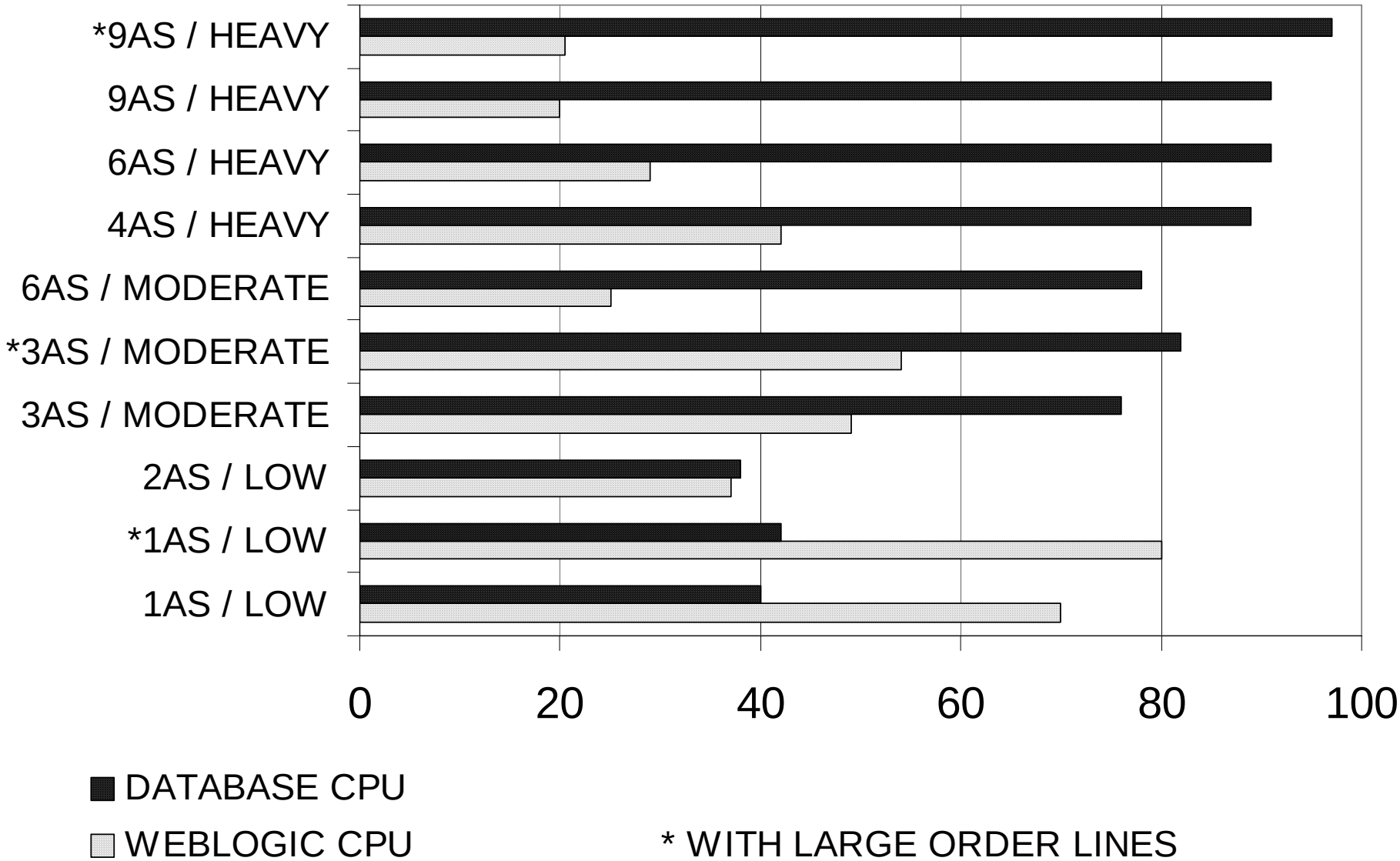
Scenarios with Large Order Lines

- **Large order (LO)** lines activated upon arrival of large orders in the customer domain. Each large order generates a separate **work order**.
- Since LO lines are triggered by NewOrder transactions we can integrate the load they produce into the service demands of NewOrder requests.
- We measure NewOrder's service demand with the LOs enabled. The additional load impacts the service demands of NewOrder requests. The latter no longer have the same semantics.

Scenarios with Large Order Lines (2)

	Low / 1 WLS		Moderate / 3 WLS		Heavy / 9 WLS	
METRIC	Model	Error	Model	Error	Model	Error
X_{CO}	4.79	6.4%	19.09	3.5%	15.31	4.5%
X_{OS}	24.77	2.9%	49.46	2.3%	48.96	3.1%
X_{CS}	19.83	2.4%	34.67	2.1%	16.37	1.9%
X_{WO}	11.96	5.7%	23.43	2.6%	29.19	1.2%
R_{CO}	86ms	60.7%	95ms	34.5%	-	-
R_{OS}	18ms	71.0%	22ms	55.1%	-	-
R_{CS}	16ms	74.6%	19ms	59.6%	-	-
R_{WO}	1179ms	16.1%	1268ms	5.0%	-	-
U_{WLS}	80%	0.0%	53%	1.9%	20%	0.0%
U_{DBS}	43%	2.4%	84%	2.4%	96%	1.0%

Summarized Utilization Results

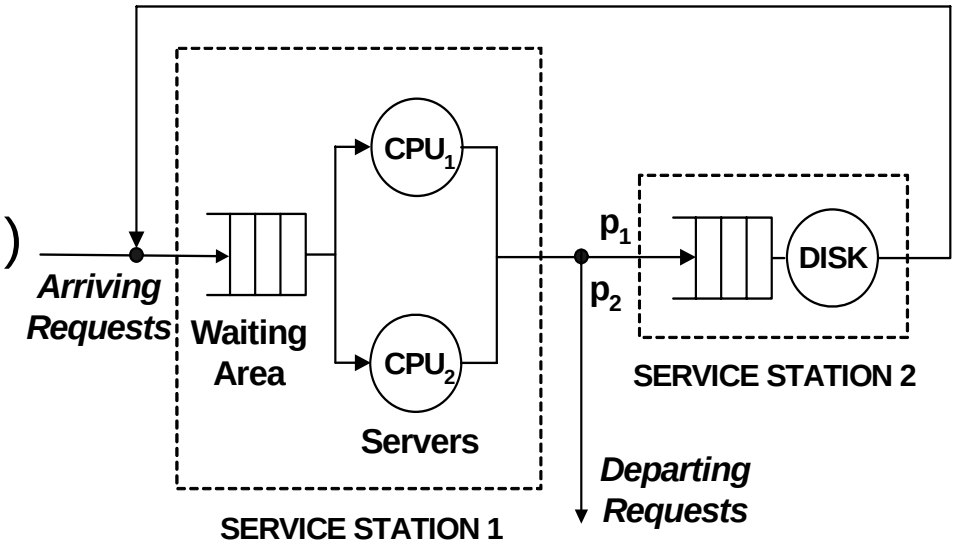


Conclusions From Case Study

- Studied a realistic J2EE application and showed how to build a performance model and use it for capacity planning.
- Model was extremely accurate in predicting transaction throughput and CPU utilization and less accurate for transaction response time.
- Ignoring the scenarios with large orders, the average modelling error for throughput was 2%, for utilization 6% and for response time 18%.
- Two problems encountered:
  - Poor model expressiveness: no way to accurately model asynchronous processing and software contention.
  - Problem solving large *non-product* form QNs analytically.

PROS vs. CONS of Queueing Networks (QNs)

- **QN**: Set of interconnected queues
- **Queue** = waiting area and servers
- Scheduling strategies(FCFS,PS,...)
- Single-class vs. multi-class
- Open, closed or mixed

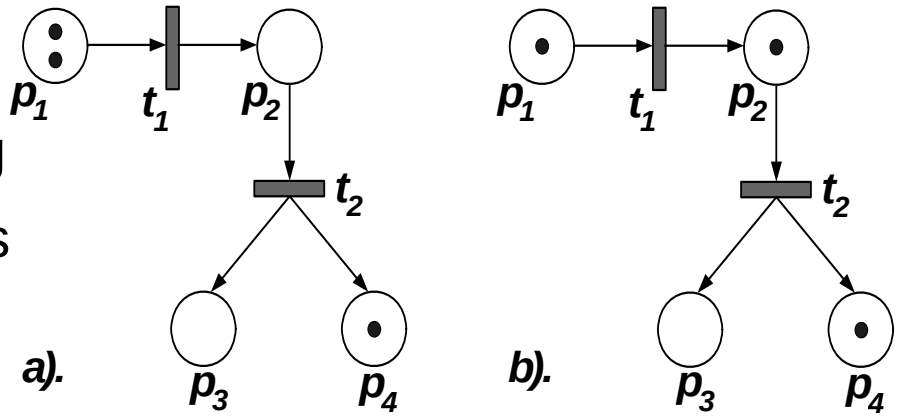


PROS: Very powerful for modelling **hardware contention** and scheduling strategies. Many efficient analysis techniques available.

CONS: Not suitable for modelling blocking, synchronization, simultaneous resource possession and **software contention** in general. Although *Extended QNs* provide some limited support for the above, they are very restrictive and inaccurate.

PROS vs. CONS of Petri Nets (PNs)

- **PN**: places, tokens and transitions.
  marking, transition enabling/firing
- **CPNs**: allow tokens of different colors
  and transition modes
- **GSPNs**: allow timed transitions
- **CGSPNs**: CPNs + GSPNs

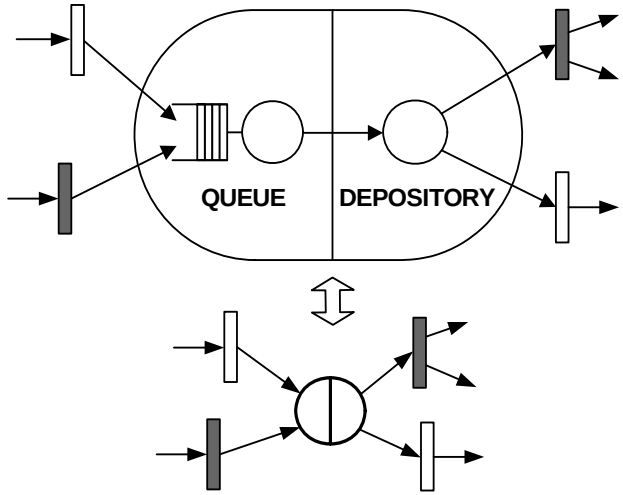


PROS: Suitable both for qualitative and quantitative analysis.
Lend themselves very well to modelling blocking, synchronization, simultaneous resource possession and software contention.

CONS: No direct means for modelling scheduling strategies. Not as many algorithms/tools for efficient quantitative analysis are available as for Queueing Networks.

Queueing Petri Nets (QPNs = QNs + PNs)

- Introduced by **Falko Bause** in 1993.
- Combine Queueing Networks and Petri Nets
- Allow integration of queues into places of PNs
- Ordinary vs. Queueing Places
- **Queueing Place** = Queue + Depository



PROS:

- Combine the modelling power and expressiveness of QNs and PNs.
- Easy to model synchronization, simultaneous resource possession, asynchronous processing and software contention.
- Allow the integration of hardware and software aspects.

CONS:

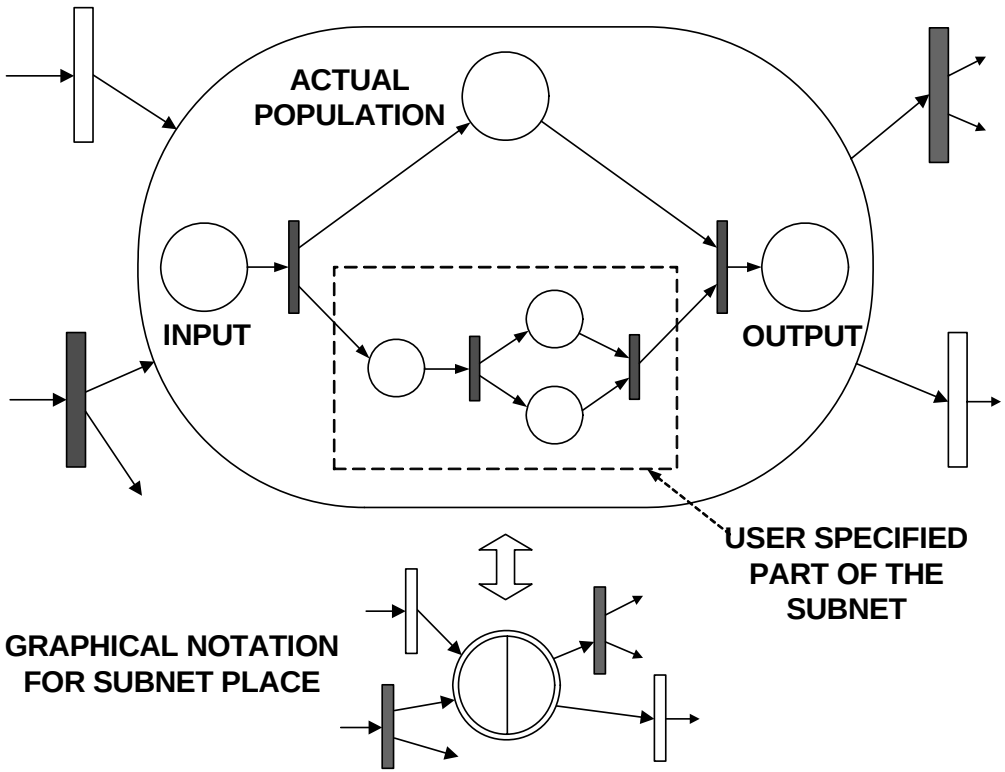
- Analysis suffers the **state space explosion** problem.

Hierarchical Queueing Petri Nets (HQPNs)

- Allow hierarchical model specification
- **Subnet Place**: contains a nested QPN
- Structured analysis methods alleviate the state space explosion problem and enable larger models to be analyzed.

Analysis Tools for HQPNs

The **HQPN-Tool** from the University of Dortmund.
Supports a number of structured analysis methods.
Available free of charge for non-commercial use.



Stochastic Petri Net Analysis Tools

- **QPME (http://www.dvs.tu-darmstadt.de/staff/skounev/QPME)**

  S. Kounev, C. Dutz and A. Buchmann“. QPME - Queueing Petri Net Modeling Environment”. Proceedings of the 3rd International Conference on Quantitative Evaluation of SysTems (QEST-2006), Riverside, USA, September 11-14, September 2006**.**

- **TimeNET (http://pdv.cs.tu-berlin.de/~timenet/)**

  Zimmermann, J. Freiheit, R. German, and G. Hommel. “Petri Net Modelling and Performability Evaluation with TimeNET 3.0”. In Proceedings of the 11th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS’2000), Schaumburg, Illinois, USA, LNCS 1786, pages 188–202, March 2000.

- **Möbius (http://www.mobius.uiuc.edu/)**

  T. Courtney, D. Daly, S. Derisavi, S. Gaonkar, M. Griffith, V. Lam, and W. Sanders. “The Möbius Modeling Environment: Recent Developments”. In Proceedings of the 1st International Conference on Quantitative Evaluation of Systems (QEST 2004), Enschede, The Netherlands, pages 328–329, Sept. 2004.

Stochastic Petri Net Analysis Tools (2)

- **SPNP (http://www.ee.duke.edu/~kst/software_packages.html)**

  C. Hirel, B. Tuffin, and K. Trivedi. “SPNP: Stochastic Petri Nets”. Version 6.0. In B. Haverkort, H. Bohnenkamp, and C. Smith, editors, Computer performance evaluation: Modelling tools and techniques; 11th International Conference; TOOLS 2000, Schaumburg, Illinois, USA, LNCS 1786. Springer Verlag, 2000.

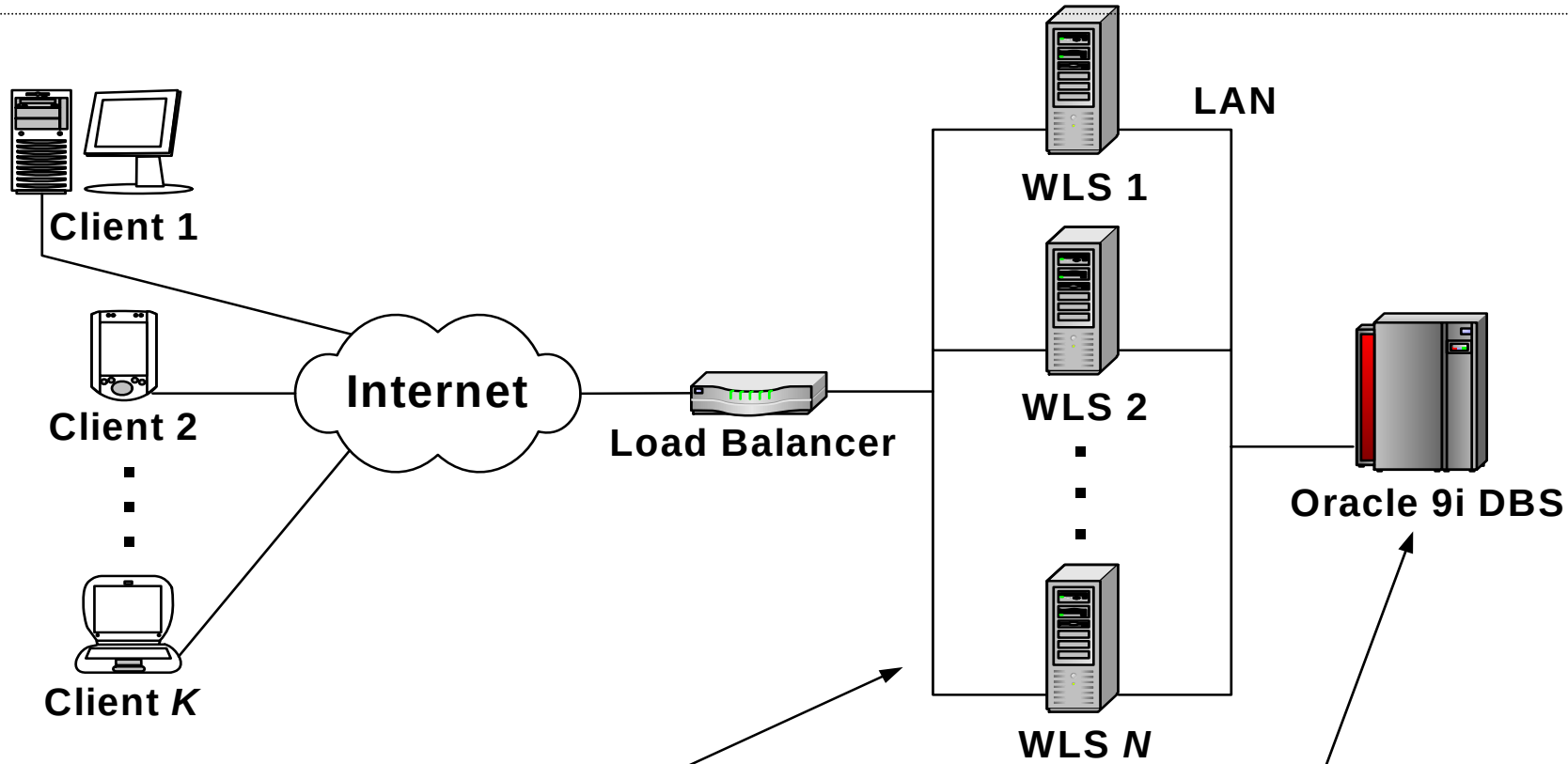
- **GreatSPN (http://www.di.unito.it/greatspn/index.html)**

  G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets. Performance Evaluation, 24(1-2):47–68, Nov. 1995.

- **Petri Nets Tool Database**

  http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html

Case Study 2: Cust. App. of SPECjAppServer



WebLogic Server 7.0 Cluster
Each node equipped with:
AMD XP 2000+ CPU, 1 GB RAM
Running on SuSE Linux 8.0

Oracle 9i (9.0.1) Database Server
Hosting the SPECjAppServer DB
1,7 GHz AMD XP CPU, 1 GB RAM
Running on Red Hat Linux 7.2

Case study taken from *"Performance Modeling of Distributed E-Business Applications using Queueing Petri Nets*". S. Kounev and A. Buchmann. Proc. of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). http://www.dvs1.informatik.tu-darmstadt.de/publications/pdf/03-ispass-QPNs.pdf

Capacity Planning Issues

We are interested in finding answers to the following questions:

- What level of performance does the system provide under load?
- Average response time, throughput and utilization = ?
- Are there potential system bottlenecks?
- How many application servers would be needed to guarantee adequate performance?

Need also optimal values for the following configuration parameters:

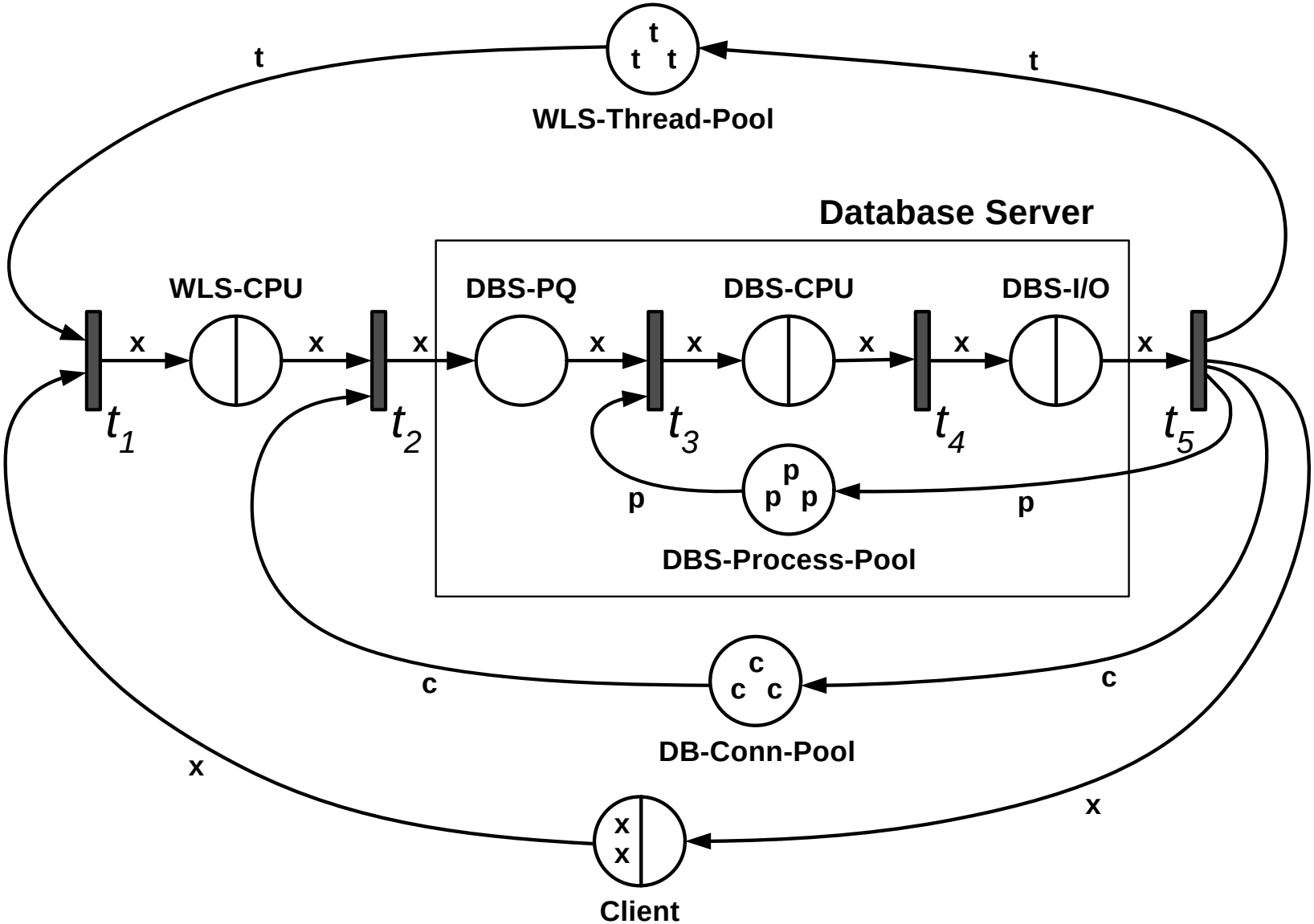
- Number of **threads** in WebLogic (WLS) thread pools
- Number of **connections** in WLS database connection pools
- Number of **processes** of the Oracle server instance

Workload Characterization

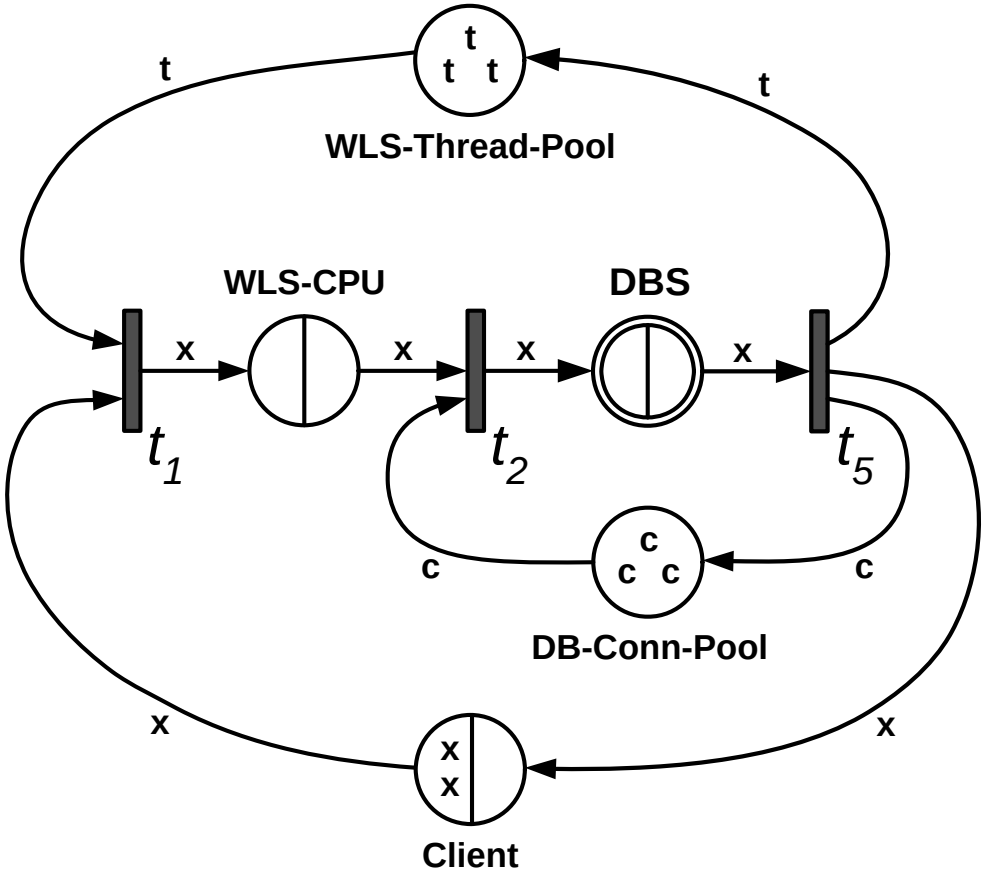
1. Describe the types of requests (**request classes**) that arrive at the system: NewOrder, ChangeOrder, OrderStatus, CustStatus.
2. Identify the hardware and software **resources used** by each request class: **HW:** WLS-CPU, Network, DBS-CPU, DBS-Disk, **SW:** WLS Thread, DB Connection, DBS Process.
3. Measure the total service time (**service demand**) of each request class at each processing resource:

TX-Type	WLS-CPU	DBS-CPU	DBS-I/O
NewOrder	70ms	53ms	12ms
ChangeOrder	26ms	16ms	6ms
OrderStatus	7ms	4ms	0ms
CustomerStatus	10ms	5ms	0ms

First Cut System Model

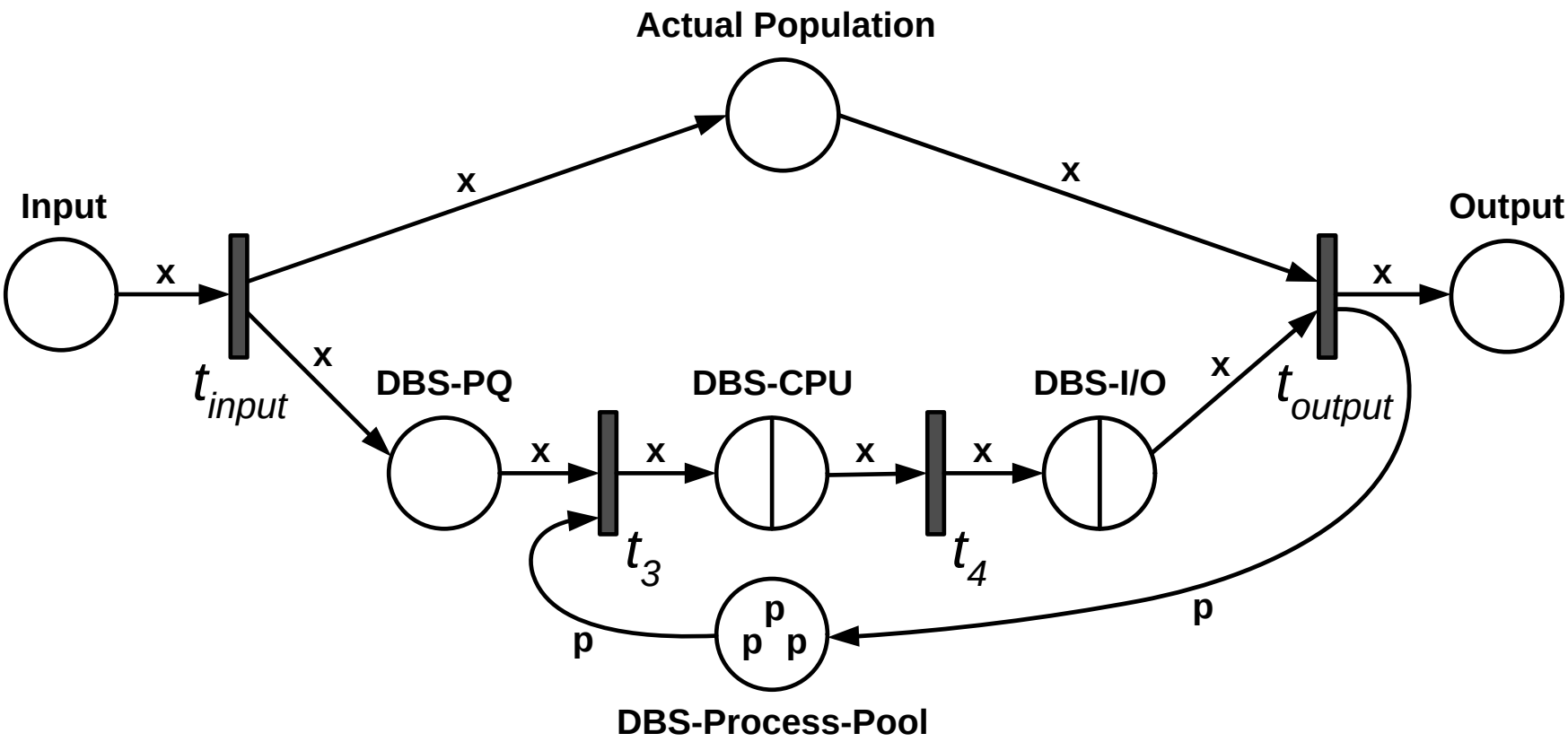


Hierarchical System Model: High-Level QPN



- We isolate the database server and model it using a separate QPN, represented by subnet place „DBS“ above.
- The above QPN is called **High-Level QPN (HLQPN)** of our hierarchical model.

Hierarchical System Model: Low-Level QPN



- The nested DBS subnet of our HQPN - called **Low-Level QPN (LLQPN)**.
- Places Input, Output and Actual Population are standard for each subnet.

Model Analysis

Through analysis of the underlying Markov Chain, we can obtain for each place, queue and depository its ***average token population*** $\mathcal{N}$ and ***utilization*** $\mathcal{U}$ in steady state.

From the **„Flow-In=Flow-Out“ Principle ➔** Places Client, WLS-CPU, DBS-PQ, DBS-CPU and DBS-I/O have the same request ***throughput*** $\mathcal{X}$.

Applying Little‘s Law to place Client, we get: $\mathcal{X} = \frac{\mathcal{N}}{\mathcal{R}}$

For the rest of places, queues and depositories, we get: $\mathcal{R} = \frac{\mathcal{N}}{\mathcal{X}}$

The total end-to-end request ***response time*** is then:

$$\begin{aligned}\mathcal{R}_{Total} = \mathcal{R}_{Client_D} + \mathcal{R}_{WLS-CPU_Q} + \\ + \mathcal{R}_{WLS-CPU_D} + \mathcal{R}_{DBS-PQ} + \\ + \mathcal{R}_{DBS-CPU_Q} + \mathcal{R}_{DBS-I/O_Q}\end{aligned}$$

Scenario 1: Single Request Class

- Single request class – the NewOrder TX
- 80 concurrent clients with avg. client think time of 200ms
- 60 WLS Threads, 40 JDBC Connections, 30 Oracle processes

PLACE	$\mathcal{N}$	$\mathcal{U}$	$\mathcal{X}$	$\mathcal{R}$ $[ms]$
Client_Q	2.85	0.94	14.28	200
Client_D	17.14	1.00	-//-	1200
WLS-CPU_Q	56.67	1.00	-//-	3967
WLS-CPU_D	0.00	0.00	-//-	0
DBS-PQ	0.00	0.00	-//-	0
DBS-CPU_Q	3.11	0.75	-//-	218
DBS-I/O_Q	0.20	0.17	-//-	14
WLS-Thread-Pool	0.00	0.00		
DB-Conn-Pool	36.67	1.00		
DBS-Process-Pool	26.67	1.00		

⇐ Analysis Results

Modelling Error ⇒

METRIC	Model	Measured	Error
WLS-CPU Utilization	100%	100%	0%
DBS-CPU Utilization	75%	65%	15%
NewOrder Throughput	14.28	13.43	6.3%
NewOrder Resp.Time	5399ms	5738ms	5.9%
Thread Queue Length	17.14	18	4.7%

Scenario 1a: Same, but only with 40 Threads

PLACE	$\mathcal{N}$	$\mathcal{U}$	$\mathcal{X}$	$\mathcal{R}$ $[ms]$
Client_Q	2.85	0.94	14.28	200
Client_D	37.14	1.00	-//-	2601
WLS-CPU_Q	36.67	1.00	-//-	2568
WLS-CPU_D	0.00	0.00	-//-	0
DBS-PQ	0.00	0.00	-//-	0
DBS-CPU_Q	3.11	0.75	-//-	218
DBS-I/O_Q	0.20	0.17	-//-	14
WLS-Thread-Pool	0.00	0.00		
DB-Conn-Pool	36.67	1.00		
DBS-Process-Pool	26.67	1.00		

⇐ Analysis Results

Modelling Error ⇒

METRIC	Model	Measured	Error
WLS-CPU Utilization	100%	100%	0%
DBS-CPU Utilization	75%	65%	15%
NewOrder Throughput	14.28	13.41	6.4%
NewOrder Resp.Time	5401ms	5742ms	5.9%
Thread Queue Length	37.14	40	7.1%

- More contention for threads, but less contention for CPU time.
- In both cases, we can reduce the number of DB connections and DBS processes, since they are not effectively utilized.

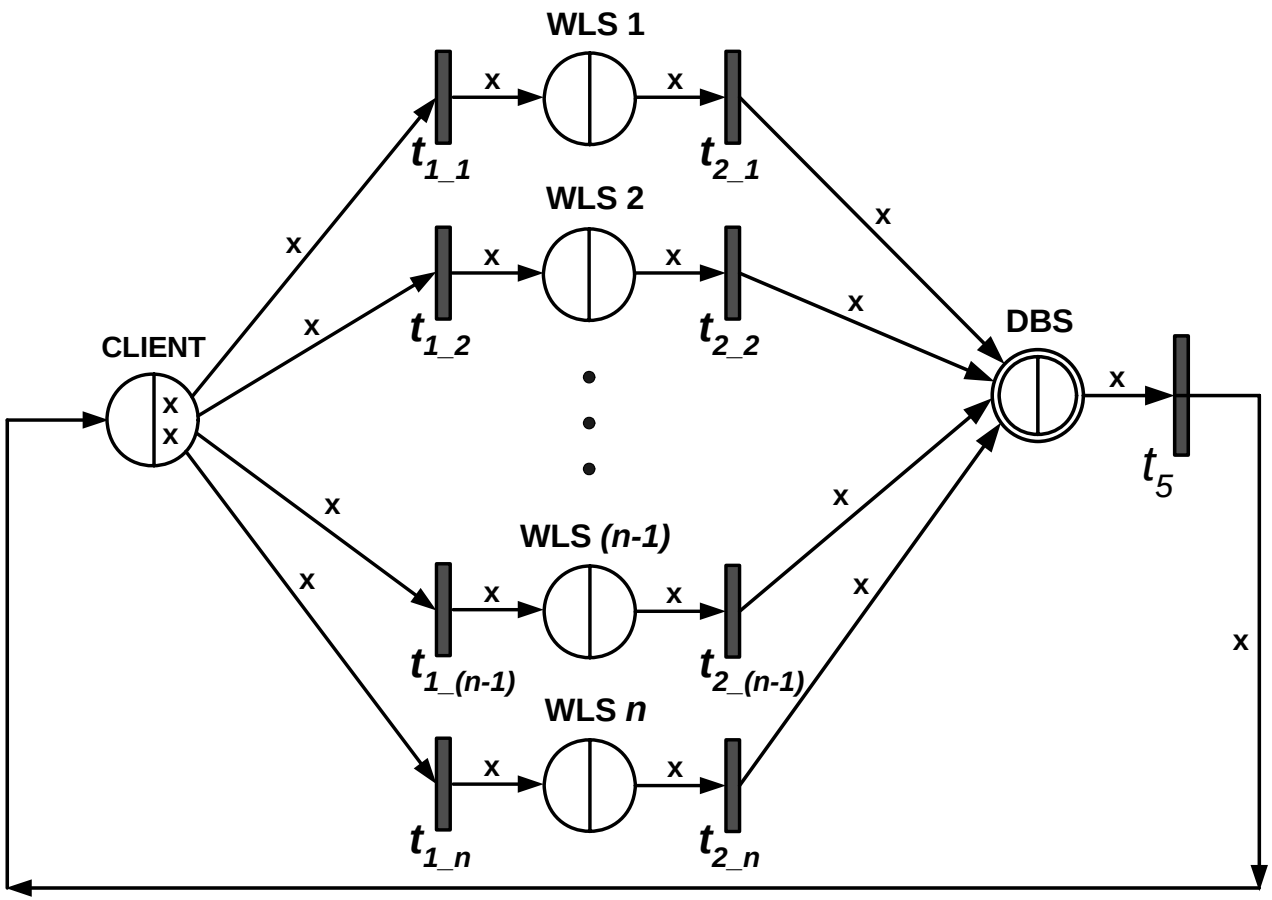
Scenario 2: Multiple Request Classes

- Two request classes – *NewOrder* and *ChangeOrder*
- Some simplifications needed to avoid explosion of the Markov Chain
- Assume that there are plenty of JDBC connections and DBS processes
- Drop places DB-Conn-Pool and DBS-Process-Pool
- 20 clients: 10 NewOrder and 10 ChangeOrder, Avg. think time = 1 sec
- Only 10 WLS Threads

METRIC	Model	Measured	Error
WLS-CPU Utilization	76%	77%	1.2%
DBS-CPU Utilization	54%	64%	15.6%
Avg.free WLS-Threads	6.68	7	4.5%
NewOrder Throughput	7.45	7.47	0.2%
NewOrder Resp. Time	341ms	318ms	7.2%
ChgOrder Throughput	9.22	9.15	0.7%
ChgOrder Resp. Time	84ms	104ms	19.2%

Scenario 3: Multiple Application Servers

- We modify the HLQPN to include multiple WLS places
- 30 NewOrder clients with avg. think time of 1 sec
- No contention for JDBC connections, DBS processes and WLS threads



Scenario 3: Modelling Error

METRIC	Model	Measured	Error
For 2 Application Servers			
WLS-CPU Utilization	64%	68%	6%
DBS-CPU Utilization	96%	91%	5%
NewOrder Throughput	18.28	17.56	4%
NewOrder Resp. Time	640ms	693ms	8%
For 3 Application Servers			
WLS-CPU Utilization	43%	44%	2%
DBS-CPU Utilization	98%	97%	1%
NewOrder Throughput	18.42	17.61	5%
NewOrder Resp. Time	623ms	673ms	7%

Conclusions from Case Study

- QPN models enable us to **integrate both hardware and software aspects of system behavior** in the same model.
- Combining the expressiveness of Queueing Networks and Petri Nets, QPNs are not just powerful as a specification mechanism, but are also **very powerful as a performance analysis and prediction tool**.
- Improved solution methods and software tools for QPNs needed to enable larger models to be analyzed.

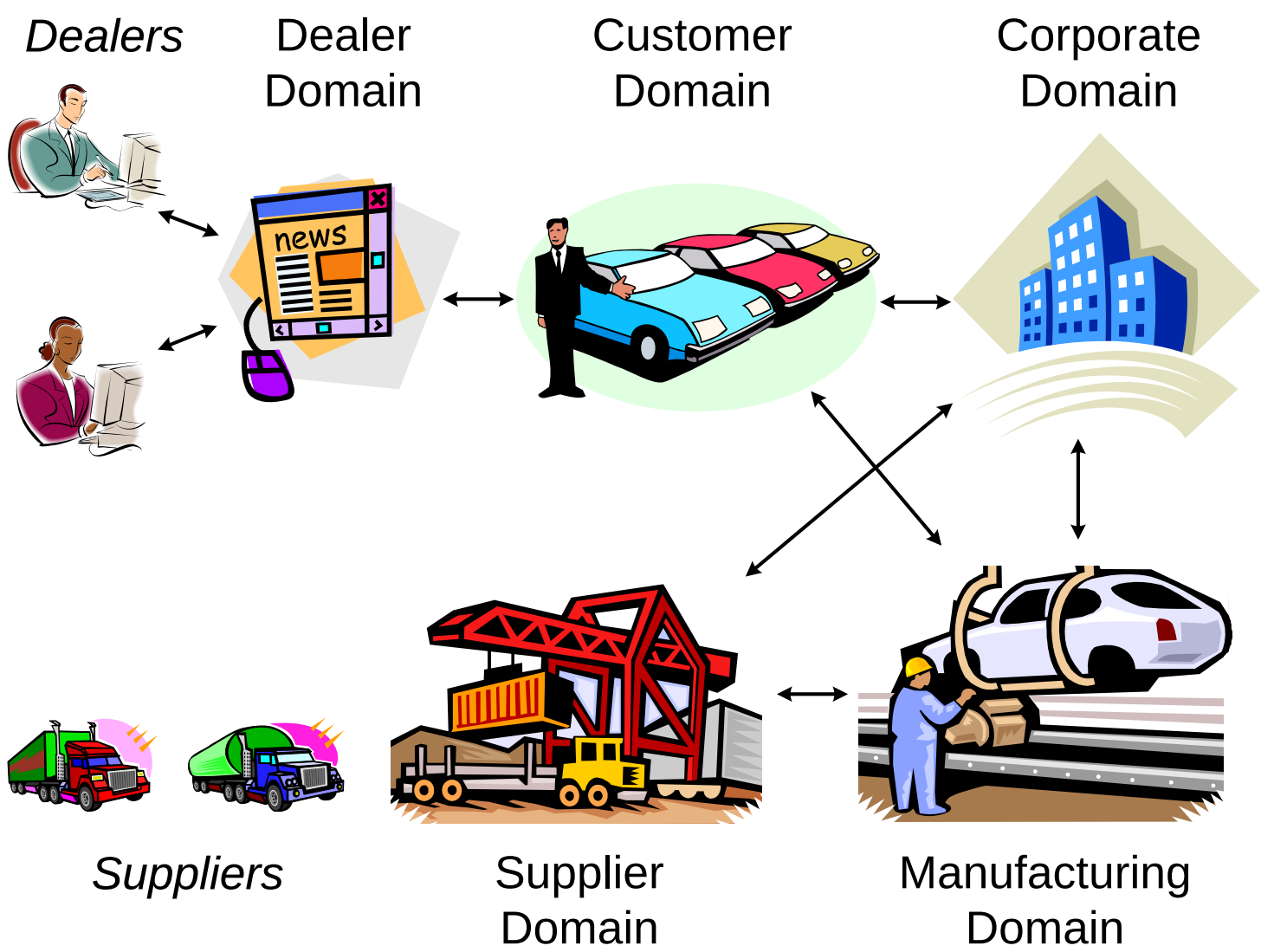
SimQPN – Simulator for QPNs

- Tool and methodology for analyzing QPNs using simulation.
- Provides a scalable simulation engine optimized for QPNs.
- Circumvents the state-space explosion problem.
- Can be used to analyze models of realistic size and complexity.
- Extremely light-weight and fast.
- Portable across platforms.
- Validated in a number of realistic scenarios.

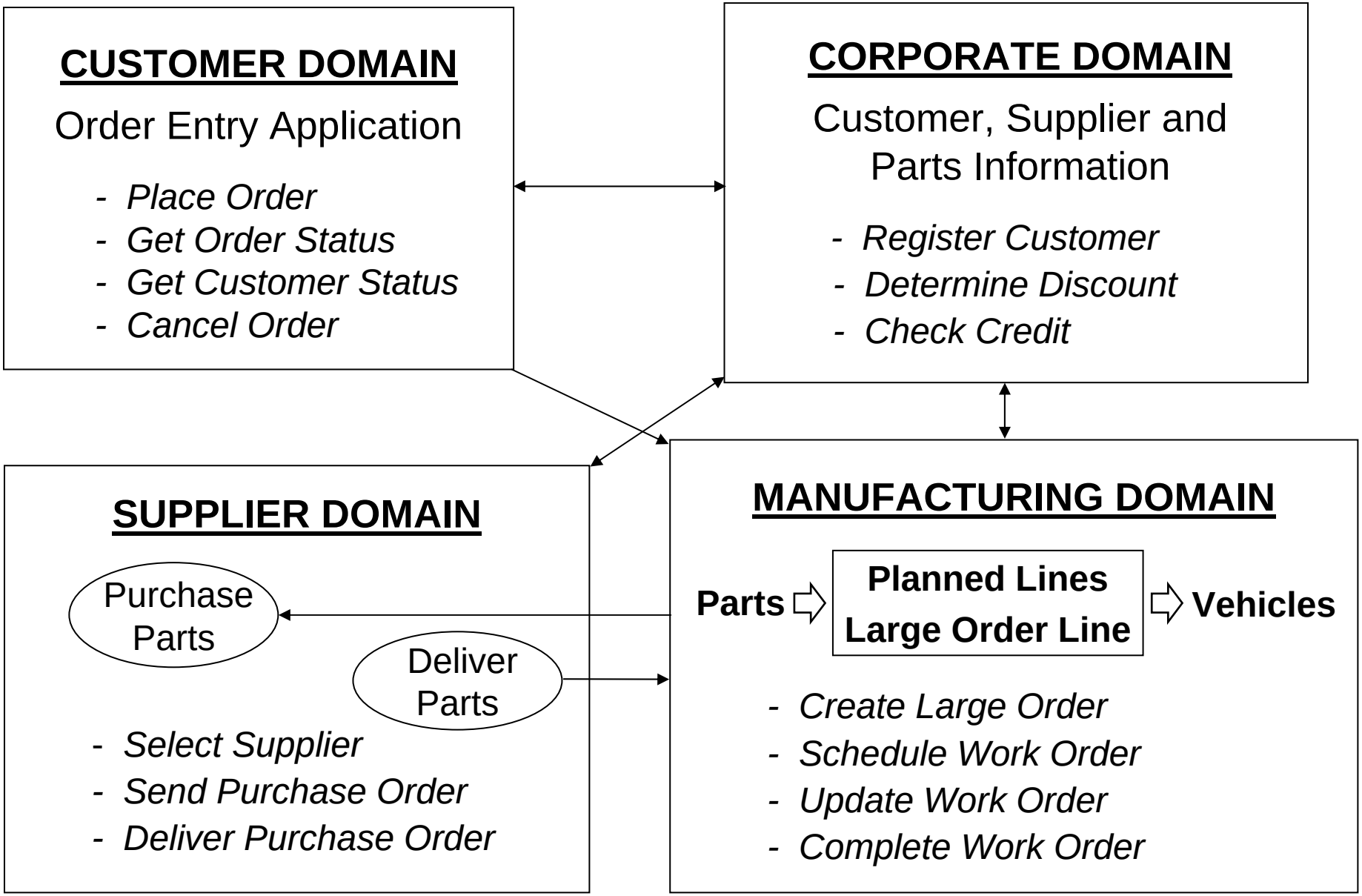
"SimQPN - a tool and methodology for analyzing queueing Petri net models by means of simulation",
Performance Evaluation, Vol. 63, No. 4-5, pp. 364-394, May 2006.

SimQPN

Case Study 3: SPECjAppServer2004



SPECjAppServer2004 Business Domains



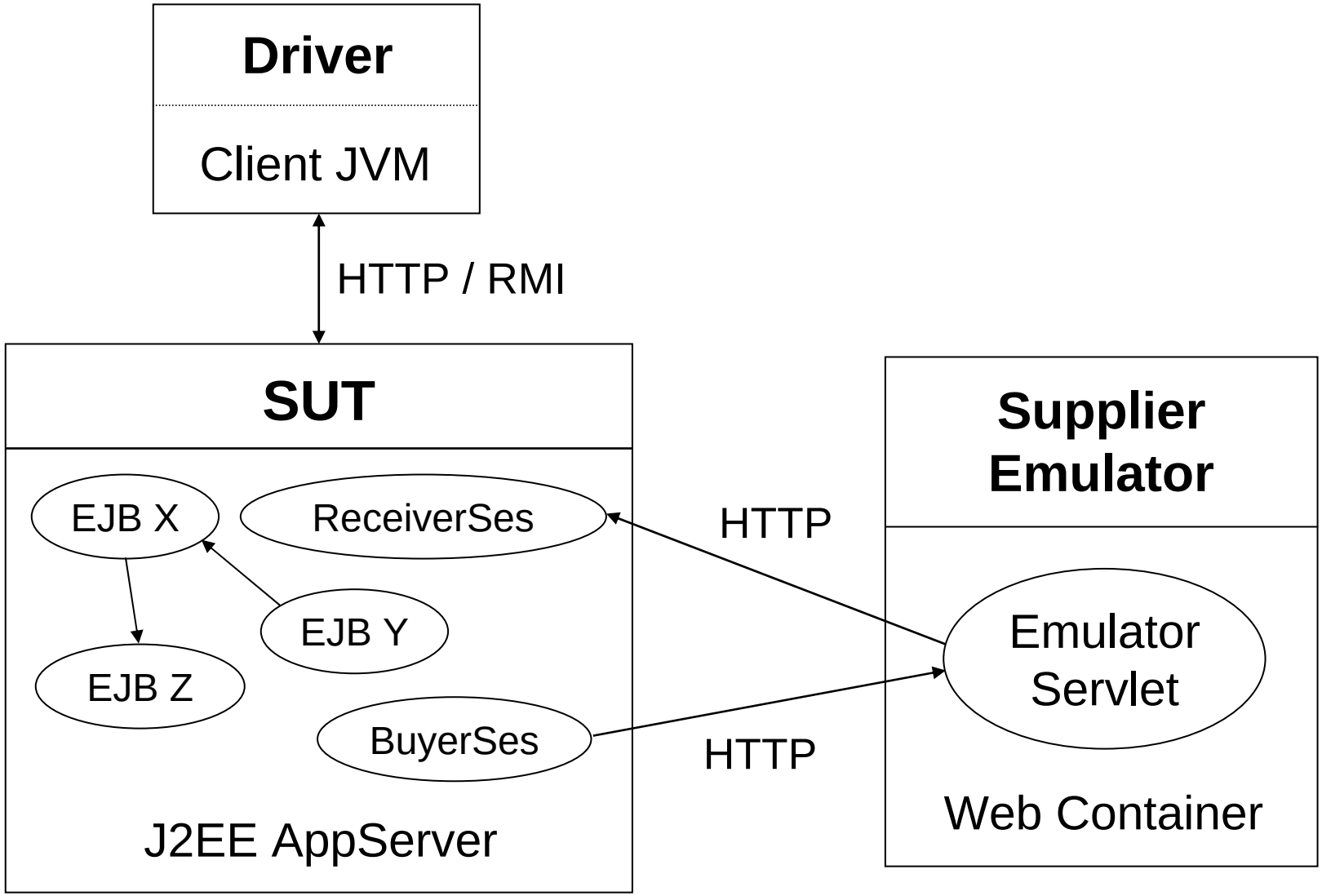
SPECjAppServer2004 Application Design

Benchmark Components:

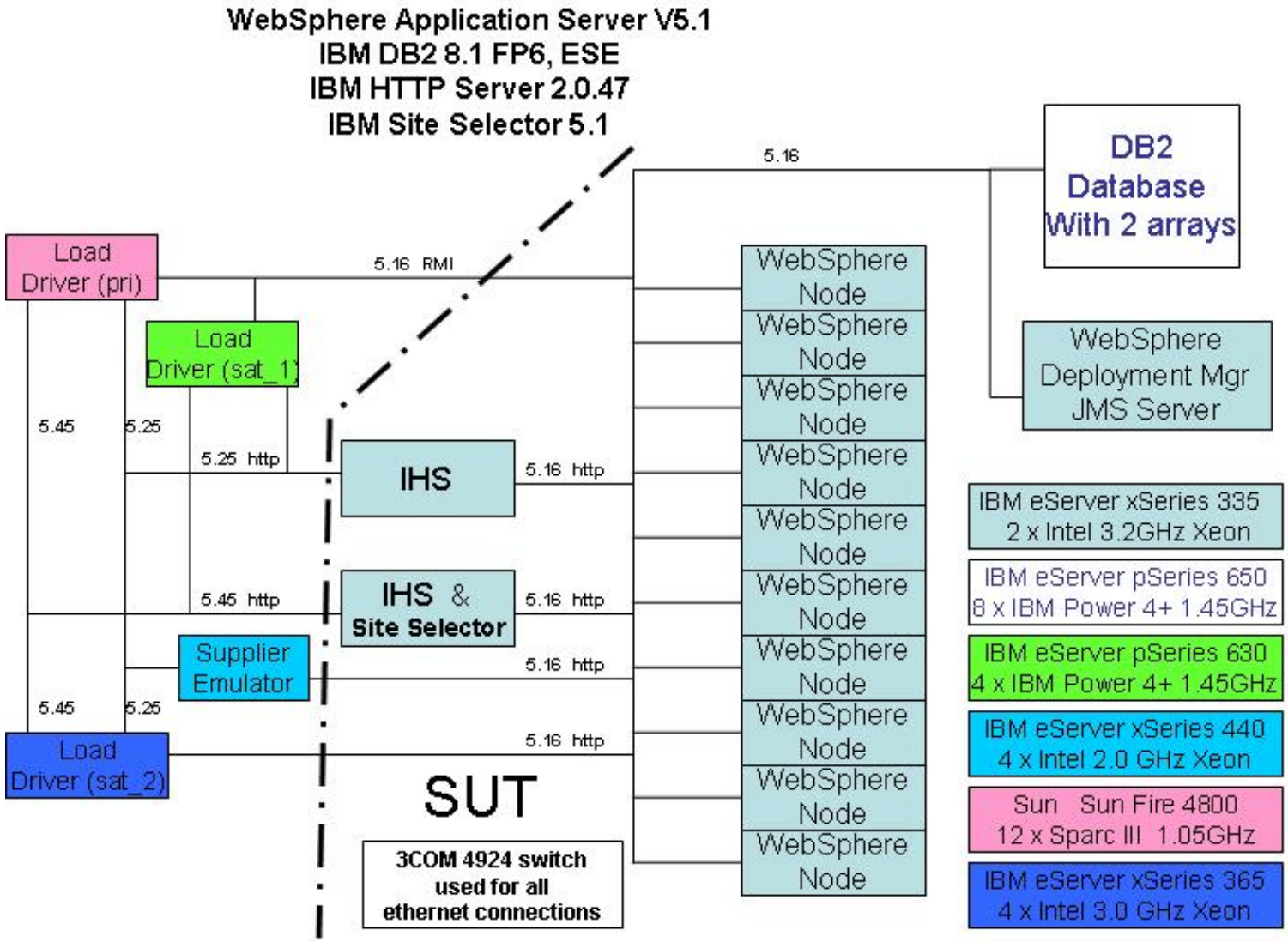
1. ***EJBs*** – J2EE application deployed on the *System Under Test (SUT*)
2. ***Supplier Emulator*** – web application emulating external suppliers
3. ***Driver*** – Java application emulating clients interacting with the system and driving production lines

- RDBMS used for persistence
- Asynchronous-messaging used for inter-domain communication
- Throughput is function of chosen *Transaction Injection Rate*
- Performance metric is **JOPS = JAppServerOpsPerSecond**

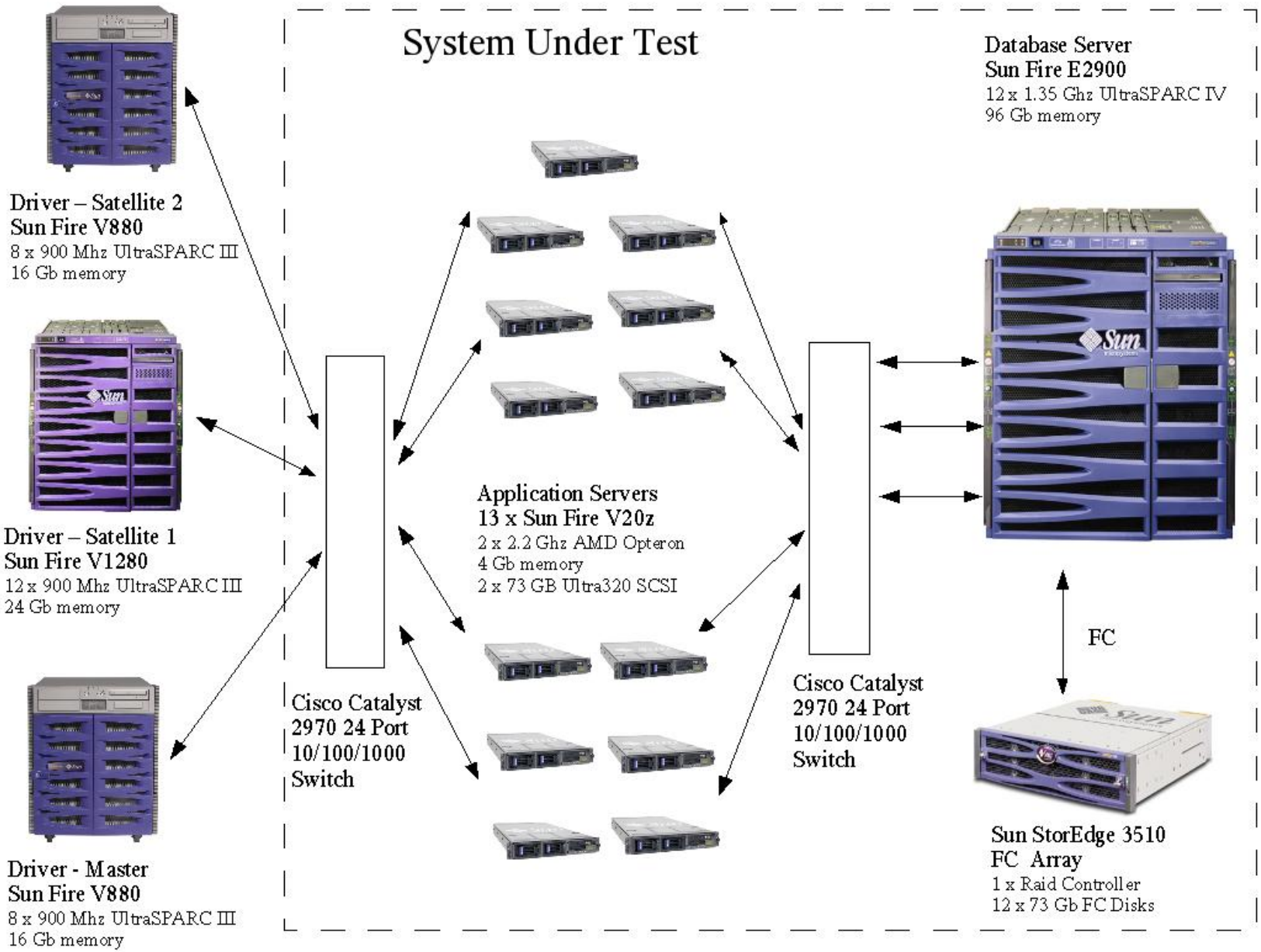
SPECjAppServer2004 Application Design (2)



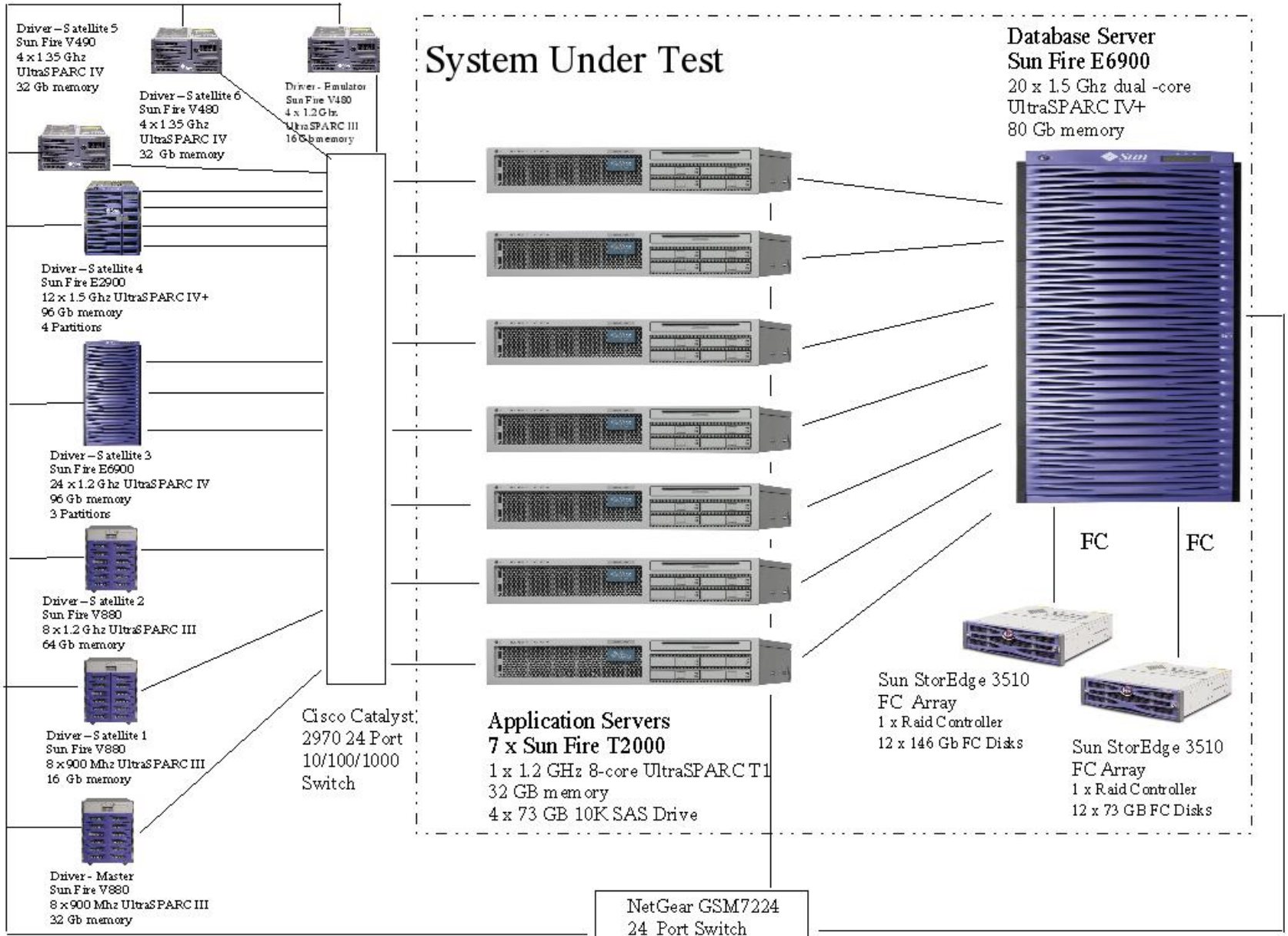
Sample Deployment Environment (IBM)



Sample Deployment Environment (Sun)



Sample Deployment Environment (Sun)

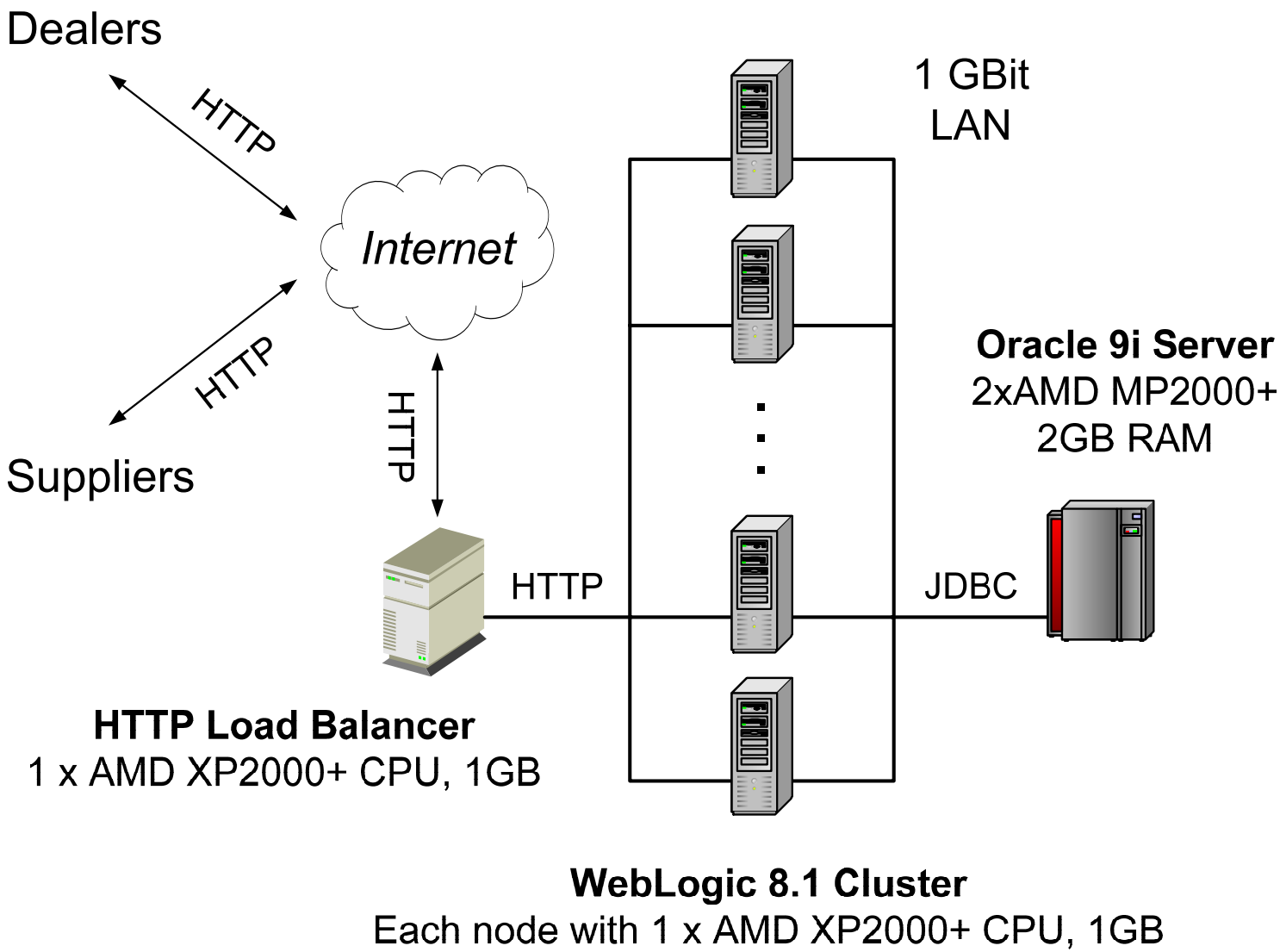


Performance Modelling Methodology

1. Establish performance modelling objectives.
2. Characterize the system in its current state.
3. Characterize the workload.
4. Develop a performance model.
5. Validate, refine and/or calibrate the model.
6. Use model to predict system performance.
7. Analyze results and address modelling objectives.

S. Kounev. *"Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets".* IEEE Transactions on Software Engineering, Vol. 32, No. 7, pp. 486-502, 2006.

Deployment Environment



1. Establish Modelling Objectives

Normal Conditions: 72 concurrent dealer clients (40 Browse, 16 Purchase, 16 Manage) and 50 planned production lines in the mfg domain.

Peak Conditions: 152 concurrent dealer clients (100 Browse, 26 Purchase, 26 Manage) and 100 planned production lines in the mfg domain.

Goals:

- Predict system performance under normal operating conditions with 4 and 6 application servers.
- Predict how much system performance would improve if the load balancer is upgraded with a slightly faster CPU.
- Study the scalability of the system as the workload increases and additional application server nodes are added.
- Determine which servers would be most utilized under heavy load and investigate if they are potential bottlenecks.

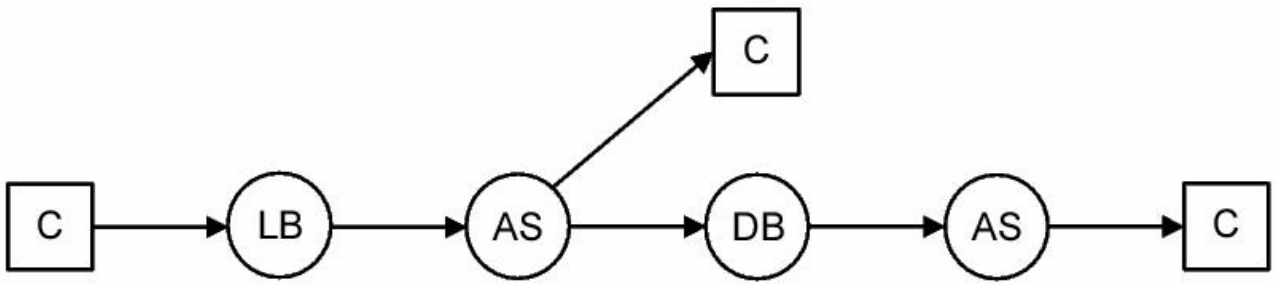
2. Characterize the System

SYSTEM COMPONENT DETAILS

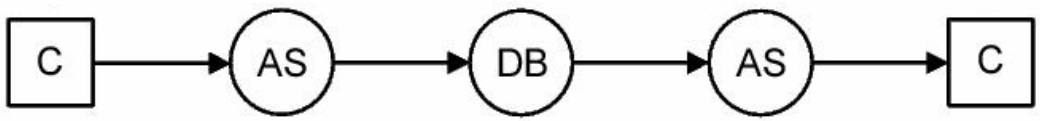
Component	Description
Load Balancer	WebLogic 8.1 Server (HttpClusterServlet) 1 x AMD Athlon XP2000+ CPU 1 GB RAM, SuSE Linux 8
App. Server Cluster Nodes	WebLogic 8.1 Server 1 x AMD Athlon XP2000+ CPU 1 GB RAM, SuSE Linux 8
Database Server	Oracle 9i Server 2 x AMD Athlon MP2000+ CPU 2 GB RAM, SuSE Linux 8
Local Area Network	1 GBit Switched Ethernet

3. Characterize the Workload

1. **Basic Components:** Dealer Transactions and Work Orders.
2. **Workload Classes:** Browse, Purchase, Manage, WorkOrder and LgrOrder.
3. **Inter-Component Interactions:**



(A). Subtransactions of Browse, Purchase and Manage



(B). Subtransactions of WorkOrder and LargeOrder

C = Client

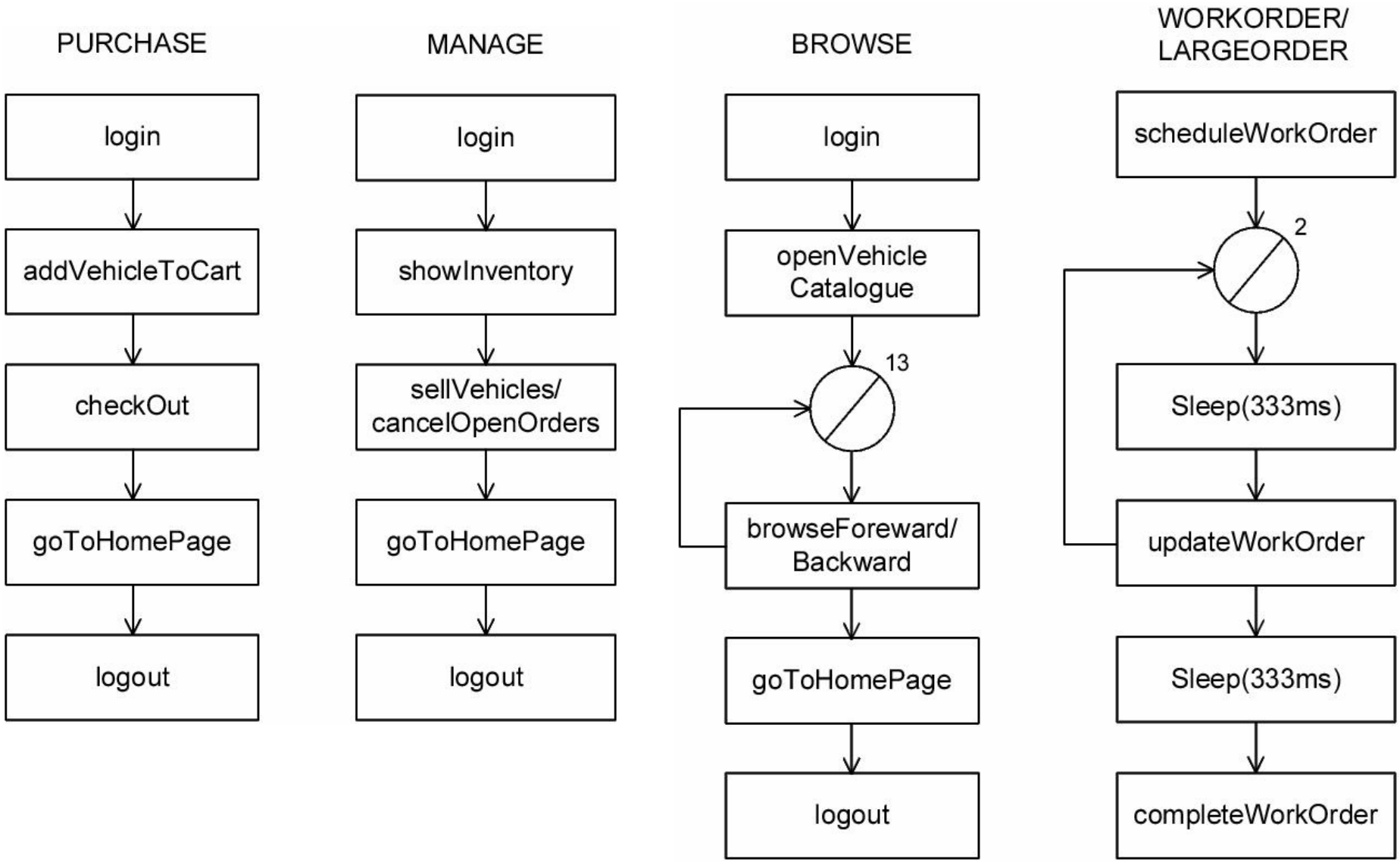
LB = Load Balancer

DB = Database

AS = App. Server

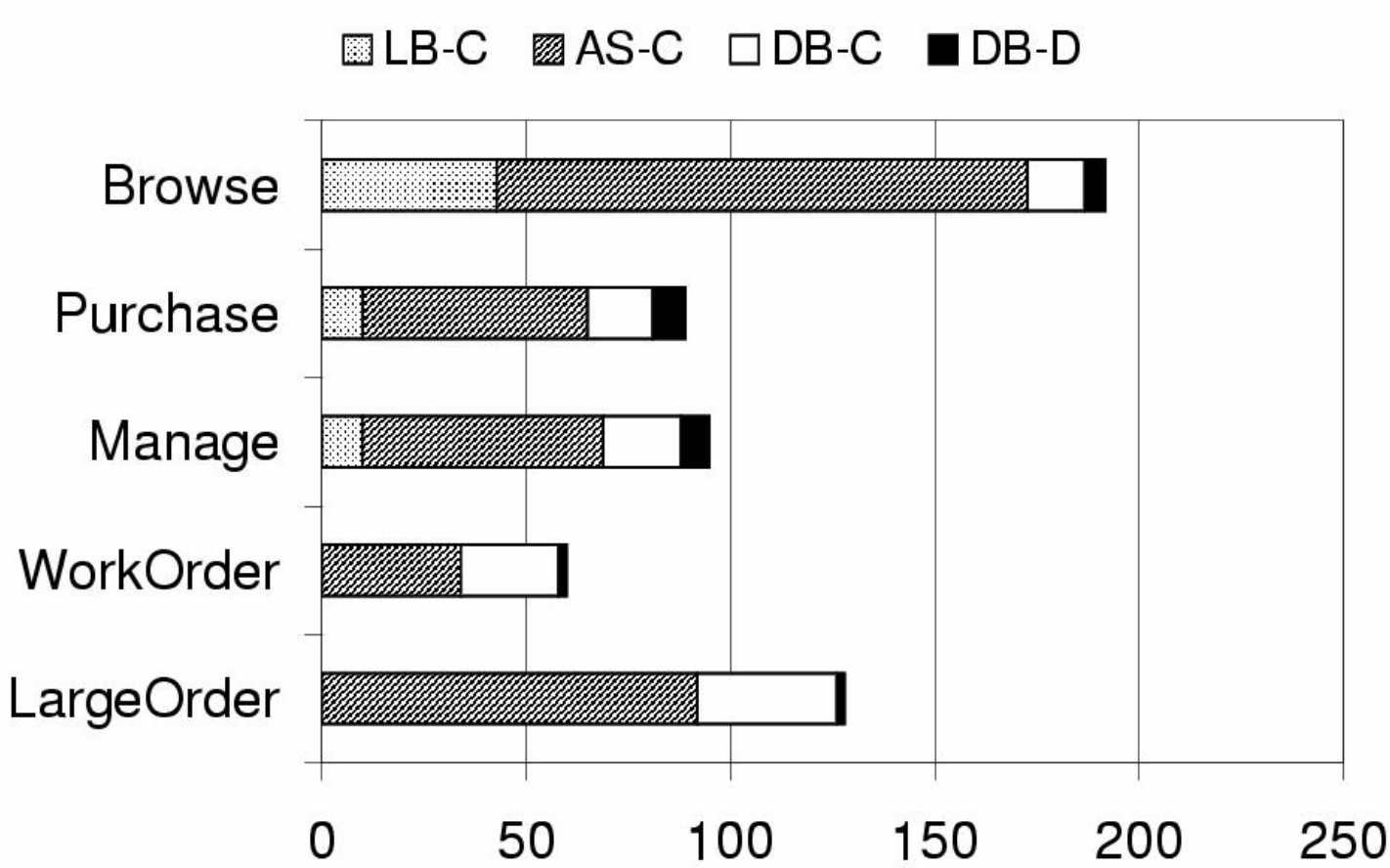
3. Characterize the Workload (2)

Describe the processing steps (subtransactions).



3. Characterize the Workload (3)

Workload Service Demand Parameters (ms)

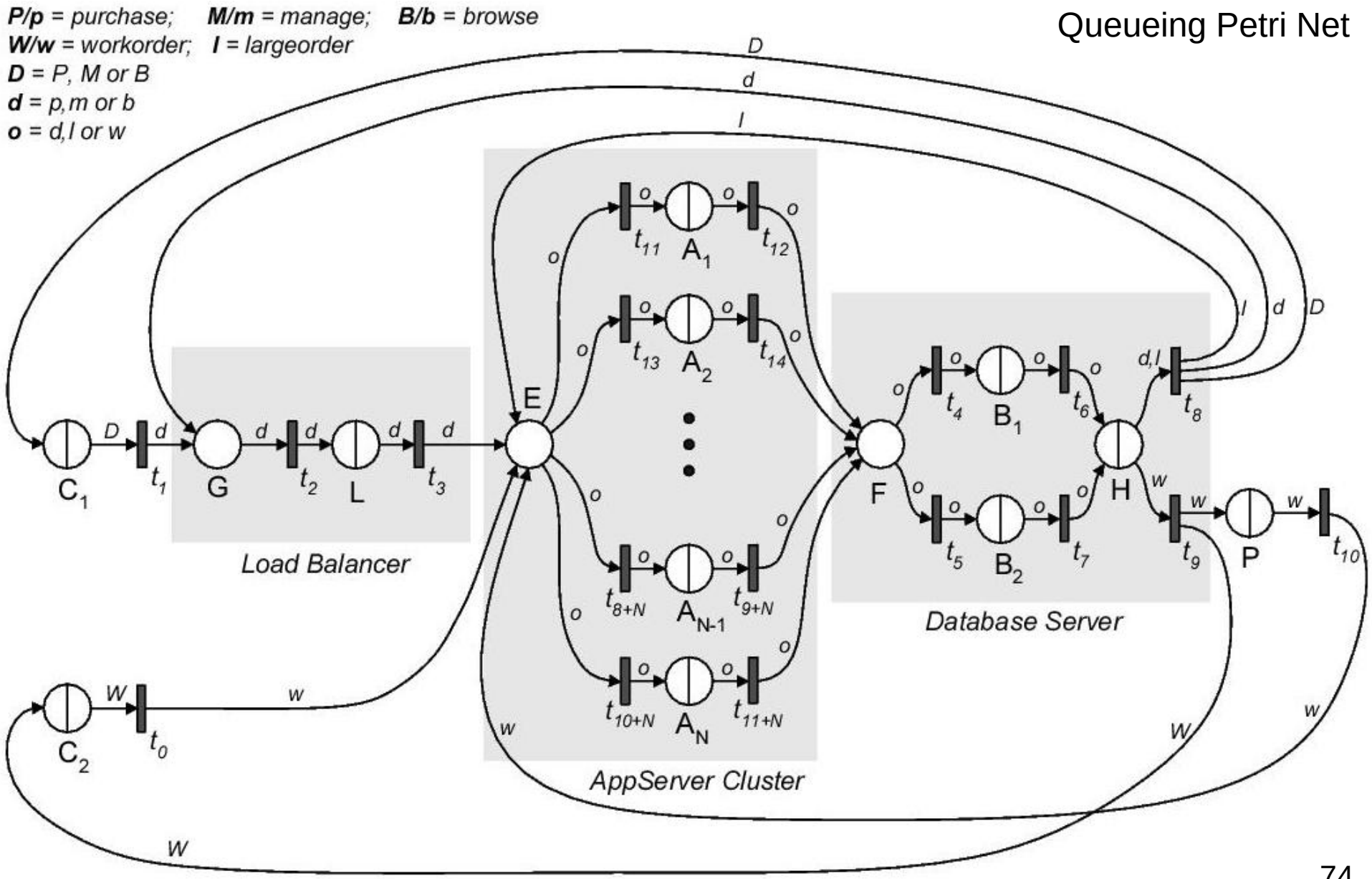


3. Characterize the Workload (4)

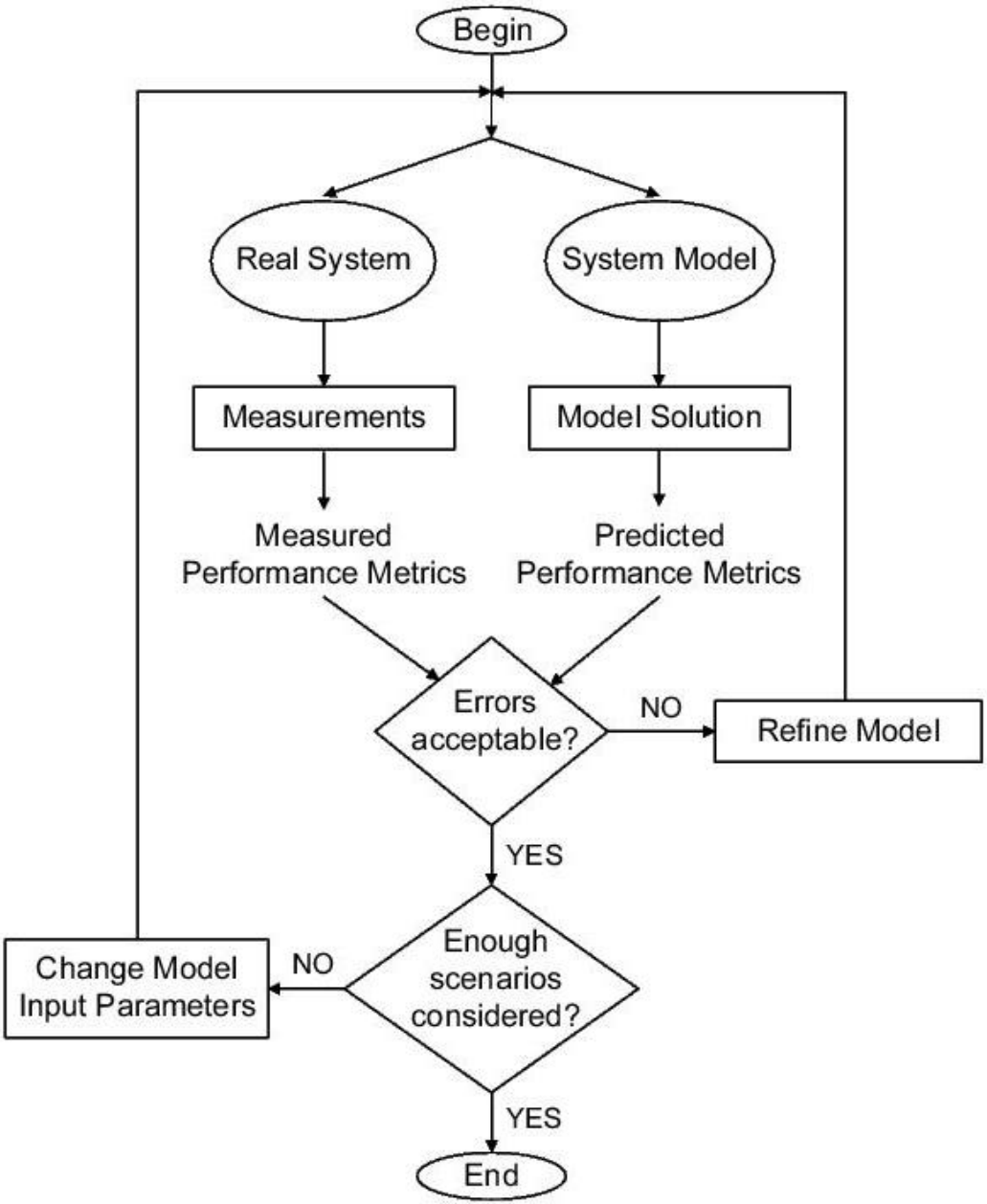
WORKLOAD INTENSITY PARAMETERS

Parameter	Normal Conditions	Peak Conditions
Browse Clients	40	100
Purchase Clients	16	26
Manage Clients	16	26
Planned Lines	50	100
Dealer Think Time	5 sec	5 sec
Mfg Think Time	10 sec	10 sec

4. Develop a Performance Model



5. Validate [Refine, Calibrate]



Assume 2 AS nodes available.

Two Specific Validation Scenarios:

1: 20 B, 10 P, 10 M, 30 PL

2: 40 B, 20 P, 30 M, 50 PL

Max. Modelling Error:

- For Throughput: 8.1%
- For Utilization: 10.2%
- For Resp. Times: 12.9%

Note: Validation process iterative!

6. Predict System Performance

ANALYSIS RESULTS FOR SCENARIOS UNDER NORMAL CONDITIONS WITH 4 AND 6 AS NODES

	4 App. Server Nodes			6 App. Server Nodes		
METRIC	Model	Measured	Error	Model	Measured	Error
X_B	7.549	7.438	+1.5%	7.589	7.415	+2.3%
X_P	3.119	3.105	+0.5%	3.141	3.038	+3.4%
X_M	3.111	3.068	+1.4%	3.117	2.993	+4.1%
X_W	4.517	4.550	-0.7%	4.517	4.320	+4.6%
X_L	0.313	0.318	-1.6%	0.311	0.307	+1.3%
R_B	299ms	282ms	+6.0%	266ms	267ms	-0.4%
R_P	131ms	119ms	+10.1%	116ms	110ms	+5.5%
R_M	140ms	131ms	+6.9%	125ms	127ms	-1.6%
R_W	1086ms	1109ms	-2.1%	1077ms	1100ms	-2.1%
U_{LB}	38.5%	38.0%	+1.3%	38.7%	38.5%	+0.1%
U_{AS}	38.0%	35.8%	+6.1%	25.4%	23.7%	+0.7%
U_{DB}	16.7%	18.5%	-9.7%	16.7%	15.5%	+0.8%

6. Predict System Performance (2)

ANALYSIS RESULTS FOR SCENARIOS UNDER PEAK CONDITIONS WITH 6 APP. SERVER NODES

	Original Load Balancer			Upgraded Load Balancer		
METRIC	Model	Measured	Error	Model	Measured	Error
X_B	17.960	17.742	+1.2%	18.471	18.347	+0.7%
X_P	4.981	4.913	+1.4%	5.027	5.072	-0.8%
X_M	4.981	4.995	-0.3%	5.013	5.032	-0.4%
X_W	8.984	8.880	+1.2%	9.014	8.850	+1.8%
X_L	0.497	0.490	+1.4%	0.501	0.515	-2.7%
R_B	567ms	534ms	+6.2%	413ms	440ms	-6.5%
R_P	214ms	198ms	+8.1%	182ms	165ms	+10.3%
R_M	224ms	214ms	+4.7%	193ms	187ms	+3.2%
R_W	1113ms	1135ms	-1.9%	1115ms	1123ms	-0.7%
U_{LB}	86.6%	88.0%	-1.6%	68.2%	70.0%	-2.6%
U_{AS}	54.3%	53.8%	+0.9%	55.4%	55.3%	+0.2%
U_{DB}	32.9%	34.5%	-4.6%	33.3%	35.0%	-4.9%

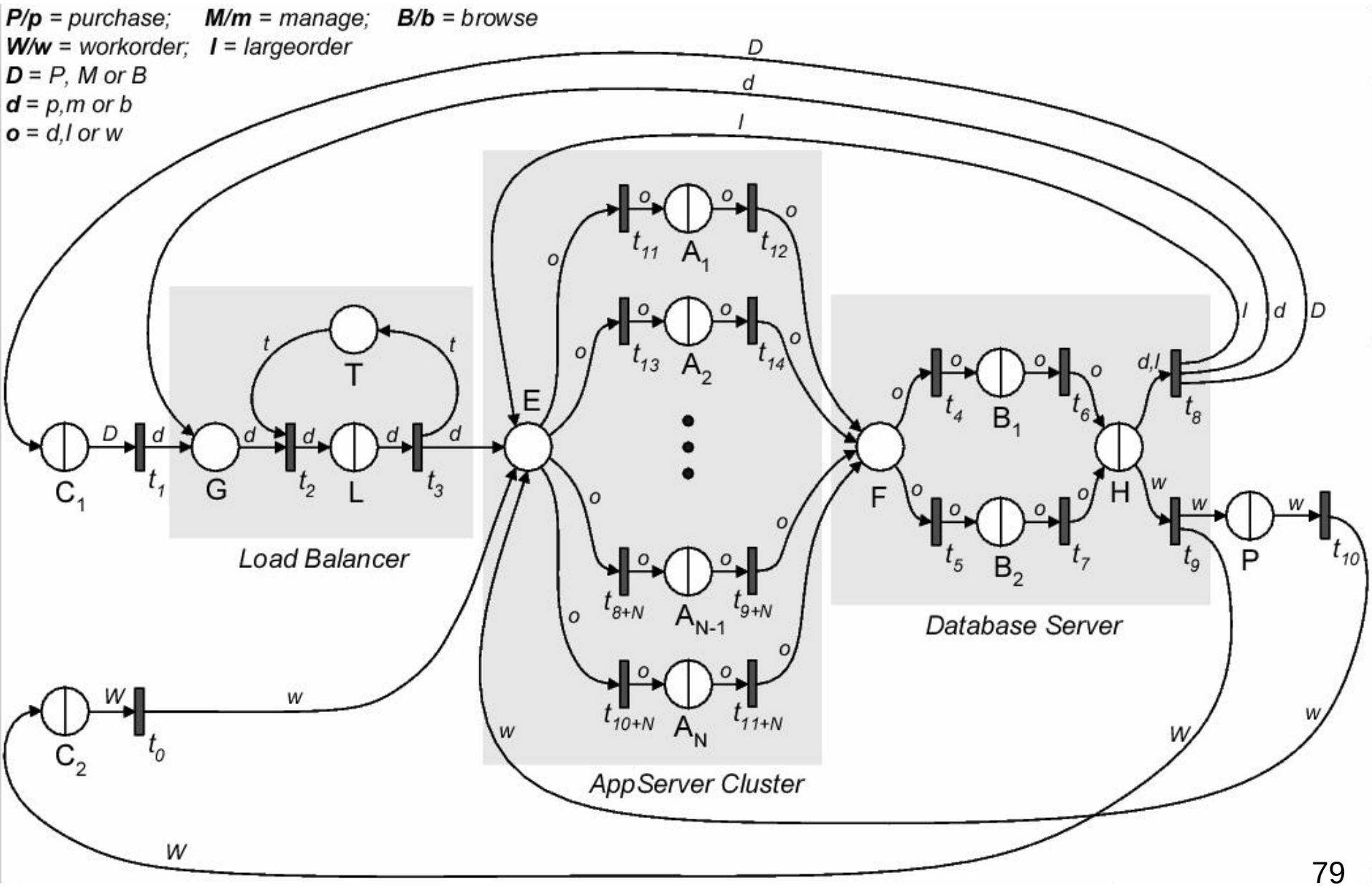
6. Predict System Performance (3)

ANALYSIS RESULTS FOR SCENARIOS UNDER HEAVY LOAD WITH 8 APP. SERVER NODES

	Heavy Load Scenario 1			Heavy Load Scenario 2		
METRIC	Model	Measured	Error	Model	Measured	Error
X_B	26.505	25.905	+2.3%	28.537	26.987	+5.7%
X_P	4.948	4.817	+2.7%	4.619	4.333	+6.6%
X_M	4.944	4.825	+2.5%	4.604	4.528	+1.6%
X_W	8.984	8.820	+1.8%	9.003	8.970	+0.4%
X_L	0.497	0.488	+1.8%	0.460	0.417	+10.4%
R_B	664ms	714ms	-7.0%	2012ms	2288ms	-12.1%
R_P	253ms	257ms	-1.6%	632ms	802ms	-21.2%
R_M	263ms	276ms	-4.7%	630ms	745ms	-15.4%
R_W	1116ms	1128ms	-1.1%	1123ms	1132ms	-0.8%
U_{LB}	94.1%	95.0%	-0.9%	99.9%	100.0%	-0.1%
U_{AS}	54.5%	54.1%	+0.7%	57.3%	55.7%	+2.9%
U_{DB}	38.8%	42.0%	-7.6%	39.6%	42.0%	-5.7%

150 Browse Clients *200 Browse Clients*

6. Predict System Performance (4)



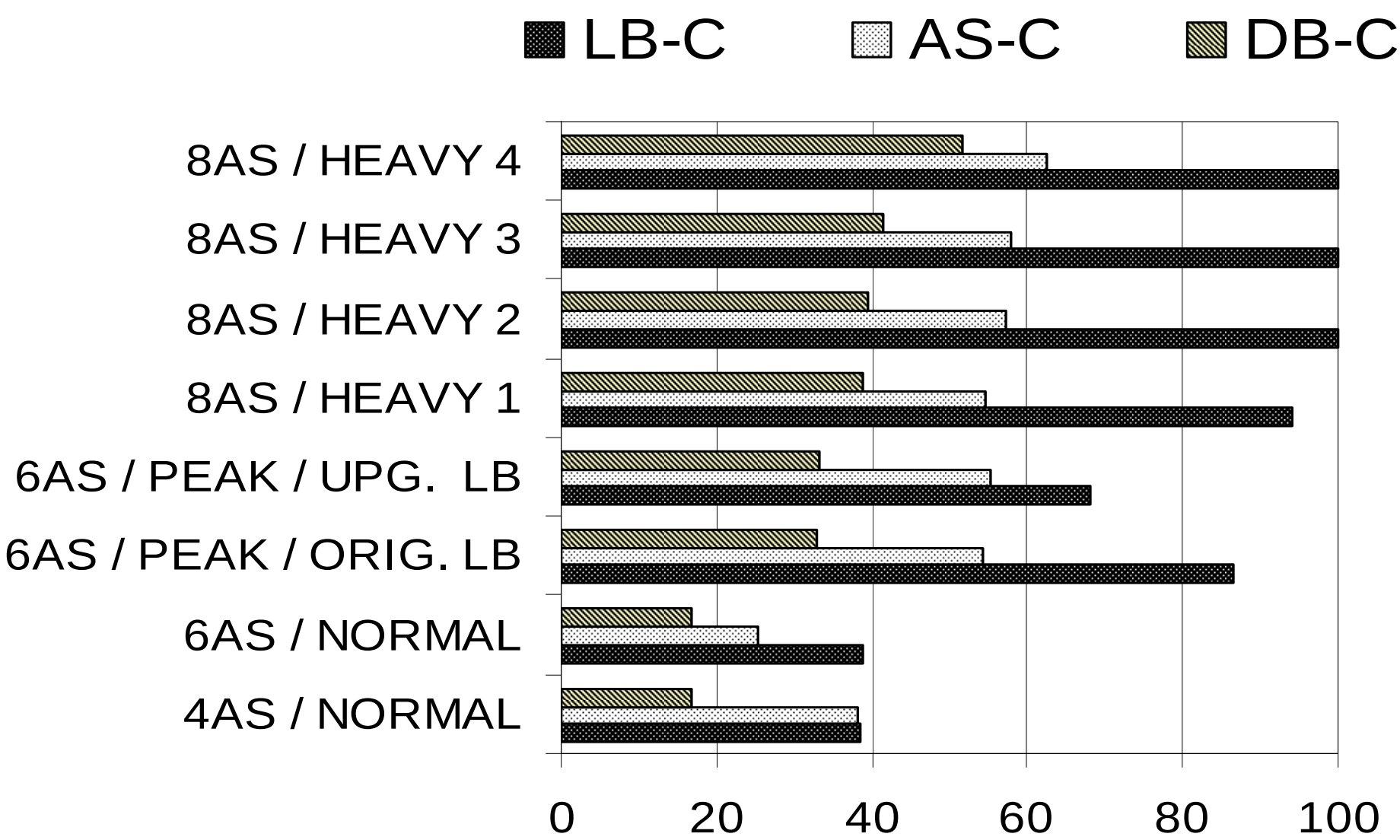
6. Predict System Performance (4)

	Heavy Load Sc. 3 with 15 Threads			Heavy Load Sc. 3 with 30 Threads		
METRIC	Model	Measured	Error	Model	Measured	Error
X_B	28.607	27.323	+4.7%	28.590	27.205	+5.1%
X_P	4.501	4.220	+6.7%	4.499	4.213	+6.8%
X_M	4.489	4.387	+2.3%	4.494	4.485	+0.2%
X_W	10.784	10.660	+1.2%	10.793	10.800	-0.1%
X_L	0.447	0.410	+9.0%	0.450	0.446	+0.1%
R_B	5495ms	5740ms	-4.2%	5495ms	5805ms	-5.3%
R_P	1674ms	1977ms	-15.3%	1665ms	2001ms	-16.8%
R_M	1685ms	1779ms	-5.3%	1670ms	1801ms	-7.3%
R_W	1125ms	1158ms	-2.8%	1125ms	1143ms	-1.6%
U_{LB}	100.0%	93.0%	+7.5%	99.9%	100.0%	-0.1%
U_{AS}	57.9%	57.8%	+0.2%	57.9%	58.0%	-0.2%
U_{DB}	41.6%	44.0%	-5.5%	41.6%	44.0%	-5.5%
N_{LBQ}	146	161	-9.3%	131	146	-10.3%

Sc.3: 300 B, 30 P, 30 M, 120 PL → Max Error **16.8%**

Sc.4: 270 B, 90 P, 60 M, 120 PL → Max Error **15.2%**

7. Analyze Results & Address Objectives



Benefits of using QPNs

1. QPN models allow the integration of hardware and software aspects of system behavior.
2. Using QPNs, DCS can be modeled *accurately*.
3. The knowledge of the structure and behavior of QPNs can be exploited for efficient simulation using SimQPN.
4. QPNs can be used to combine qualitative and quantitative system analysis.
5. QPN models have an intuitive graphical representation facilitating model development.

Summary & Conclusions from Case Study

- Presented a systematic approach for performance prediction.
- Studied a **representative** application and predicted its performance under **realistic** load conditions.
- Model predictions were validated against measurements on the real system. The modelling error did not exceed 21.2%!

- QPN models can be exploited for **accurate** performance prediction in realistic scenarios.
- Proposed methodology provides a powerful tool for sizing and capacity planning.

References

- *"Performance Modeling of Distributed E-Business Applications using Queueing Petri Nets"*, S. Kounev and A. Buchmann. Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS).

- *"Performance Modeling and Evaluation of Large-Scale J2EE Applications"*, S. Kounev and A. Buchmann, Proceedings of the 29th International CMG Conference on Resource Management and Performance Evaluation of Enterprise Computing Systems, 2003.

- *"Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets"*, S. Kounev. IEEE Transactions on Software Engineering, Vol. 32, No. 7, pp. 486-502, July 2006.

- *"J2EE Performance and Scalability - From Measuring to Predicting"*, S. Kounev. In Proceedings of the 2006 SPEC Benchmark Workshop, Austin, Texas, January 23, 2006.

Further Reading

- *"Performance by Design: Computer Capacity Planning by Example"* by D. Menascé, V.A.F. Almeida and L. W. Dowdy. Prentice Hall, ISBN 0-13-090673-5, 2004

- *"Performance Engineering of Distributed Component-Based Systems - Benchmarking, Modeling and Performance Prediction"* by S. Kounev, Shaker Verlag, ISBN: 3832247130, 2005

- *"Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning"* by D. Menascé and V.A.F. Almeida, Prentice Hall, ISBN 0-13-086328-9, 2000

- *"Capacity Planning for Web Performance: Metrics, Models and Methods"* by D. Menascé and V.A.F. Almeida, Prentice Hall, Upper Saddle River, NJ, 1998.

Further Reading (2)

- *"Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications"* by G. Bolch, S. Greiner, H. de Meer, K. Trivedi; Wiley-Interscience, 2 Edition, ISBN: 0471565253, 2006

- *"Probability and Statistics with Reliability, Queuing and Computer Science Applications"* by K. S. Trivedi, John Wiley & Sons, Inc., Second edition, 2002.

Links to Further Related Tools

- OMNeT++
  - See http://www.omnetpp.org/
  - Public-source, component-based, modular and open-architecture simulation environment with strong GUI support and an embeddable simulation kernel.
- DesmoJ
  - http://asi-www.informatik.uni-hamburg.de/themen/sim/forschung/Simulation/Desmo-J/index.html
  - A framework for discrete-event modelling and simulation
- A Collection of Modelling and Simulation Resources
  - See http://www.idsia.ch/~andrea/simtools.html
- JMeter
  - See http://jakarta.apache.org/jmeter/
  - Java-based tool for load testing client-server applications.
- Faban benchmark harness and driver framework
  - See http://faban.sunsource.net/
- List of open-source testing tools
  - See http://www.opensourcetesting.org