



Methodology for Addressing Performance Problems

Learning objectives

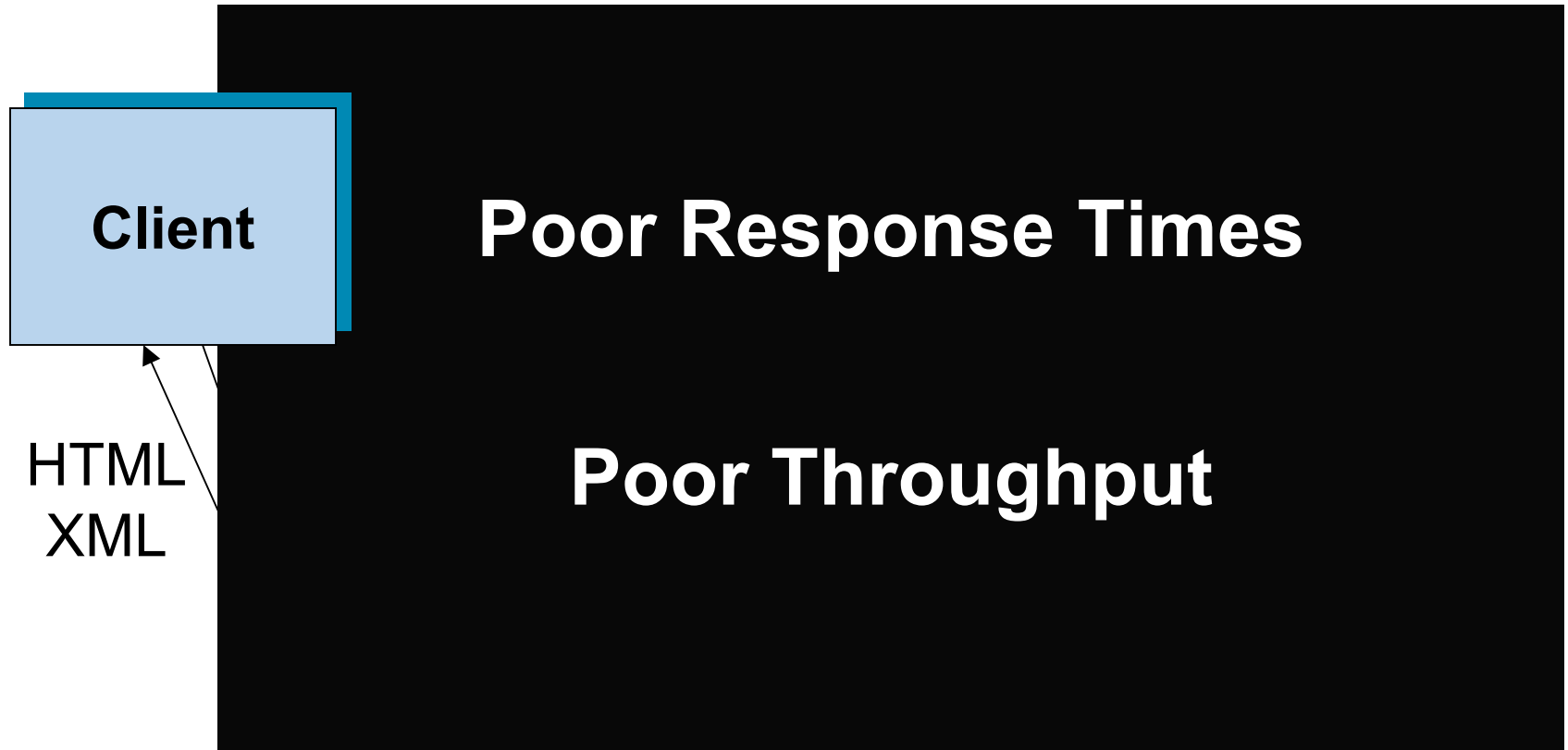
At the end of this section you will be able to:

- Understand how to approach performance problems
 - Identify bottlenecks in resource usage
- Systematically solve performance problems
- See examples of how to perform assessment, measurement and analysis
- Use tools
 - HPjconfig - Assess your system's kernel parameters
 - SIGQUIT – Use for quick snapshot of performance
- Understand the key areas for successful application development and deployment

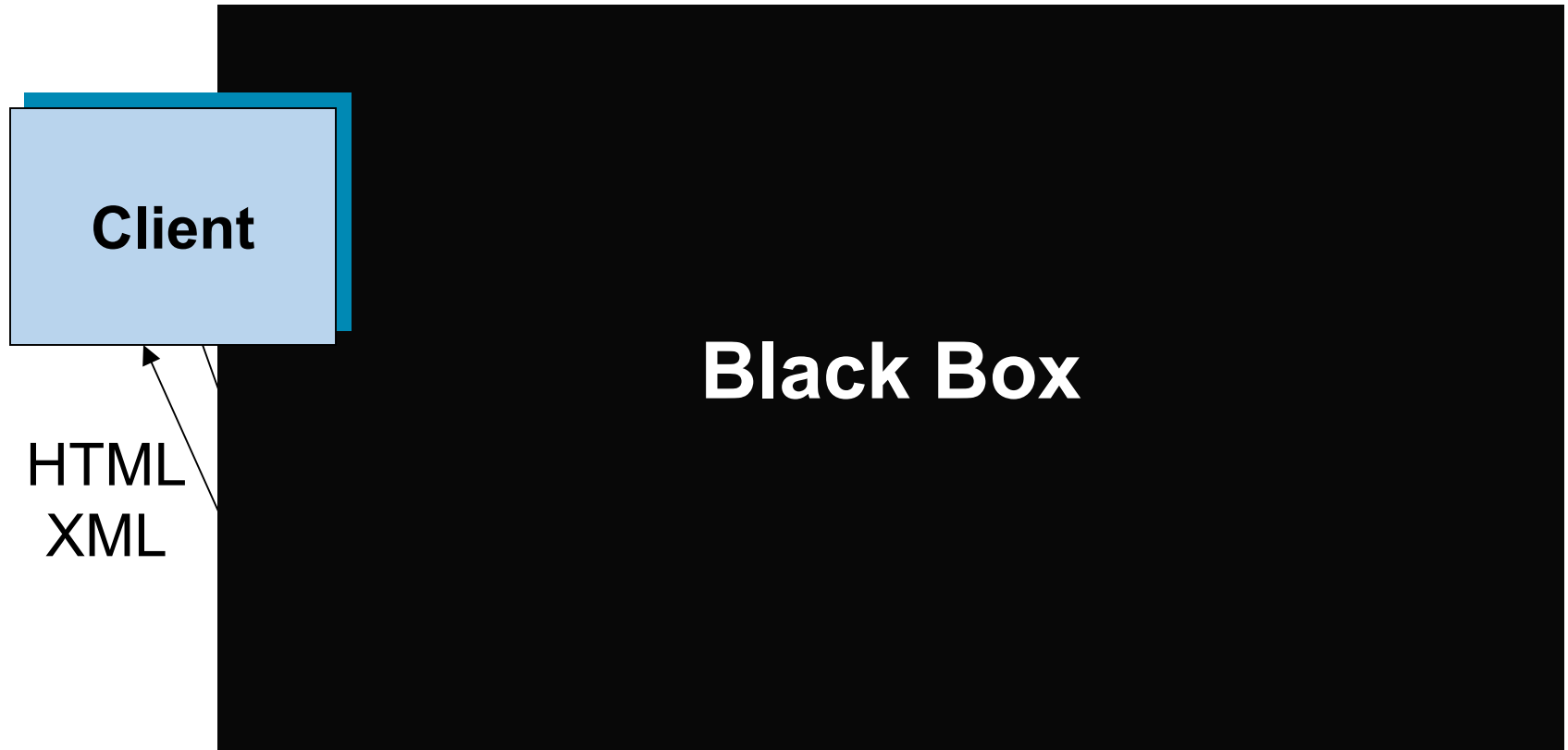
Bottlenecks

- Where are they?
- How do you find them?

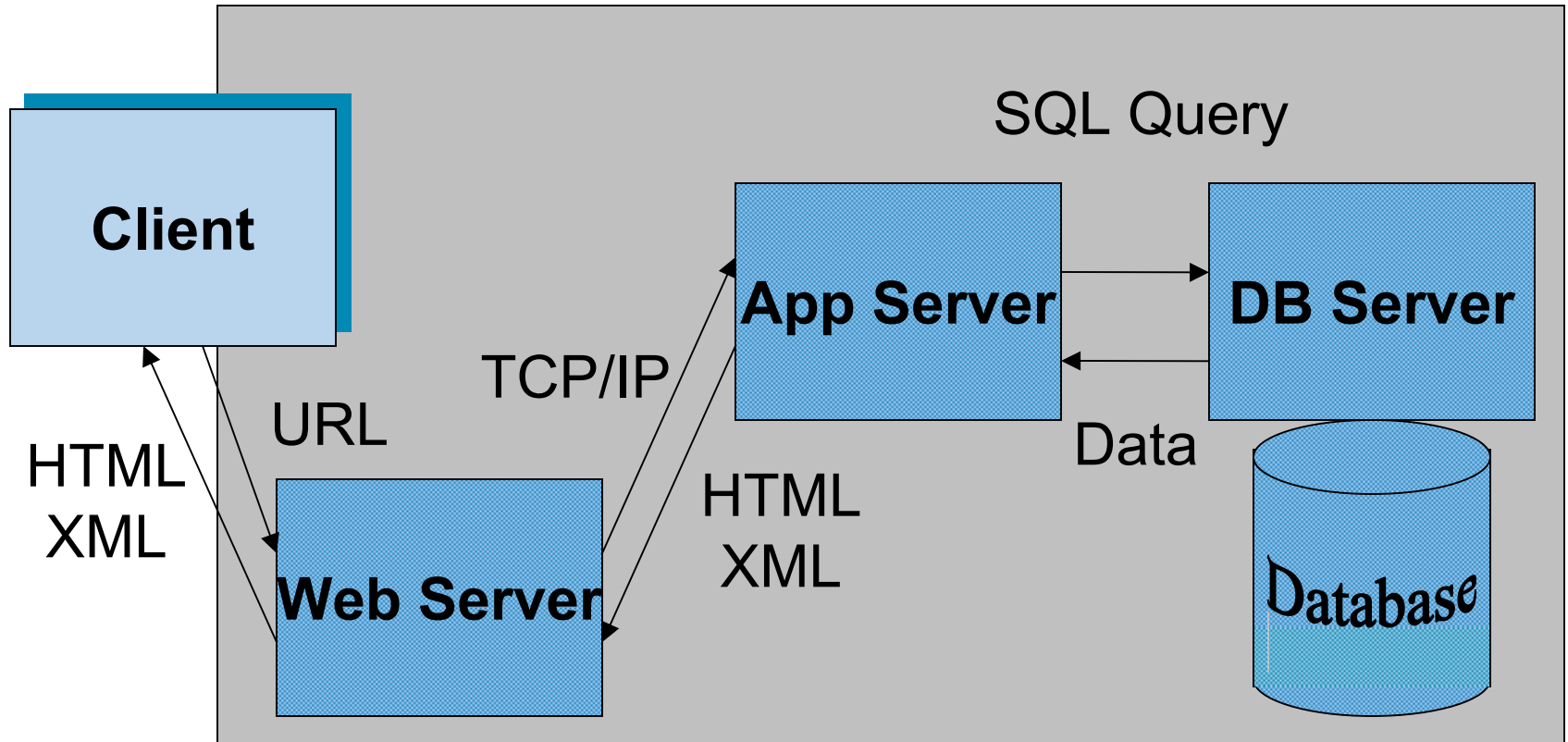
Where's the Performance Problem?



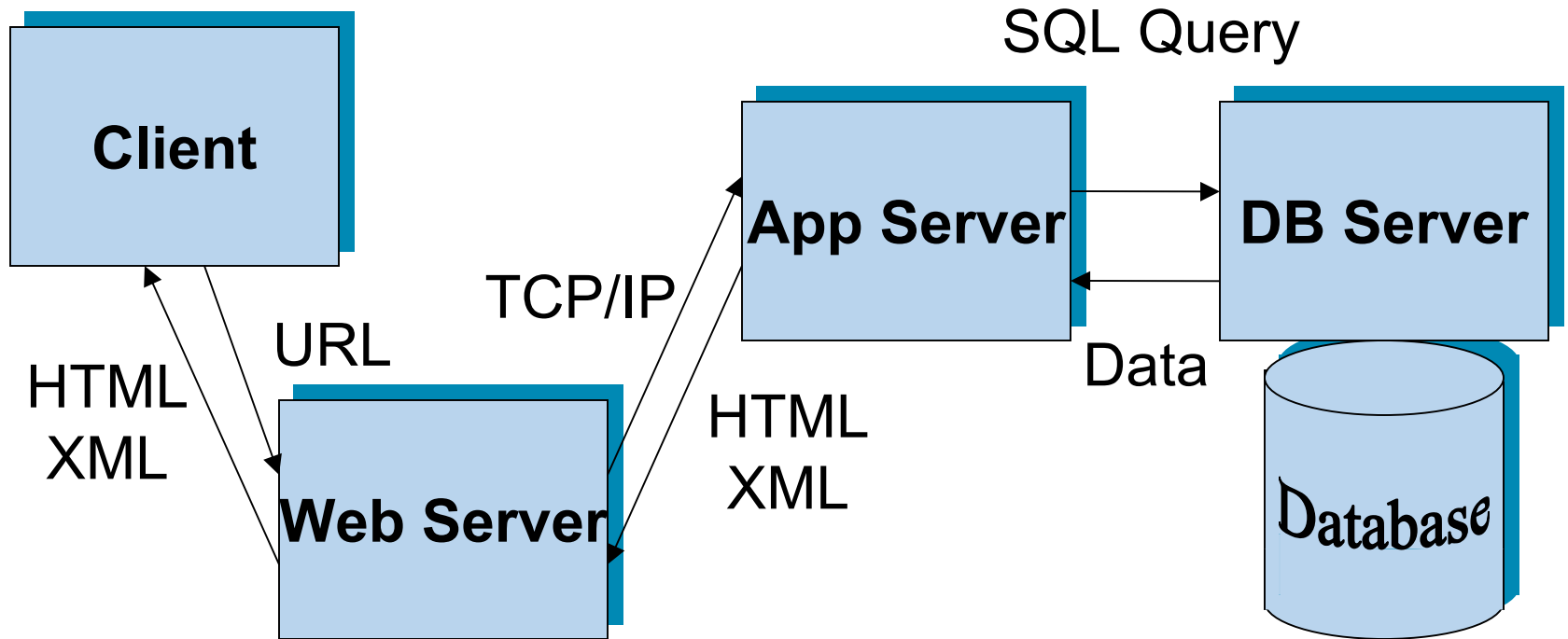
Where's the Performance Problem?



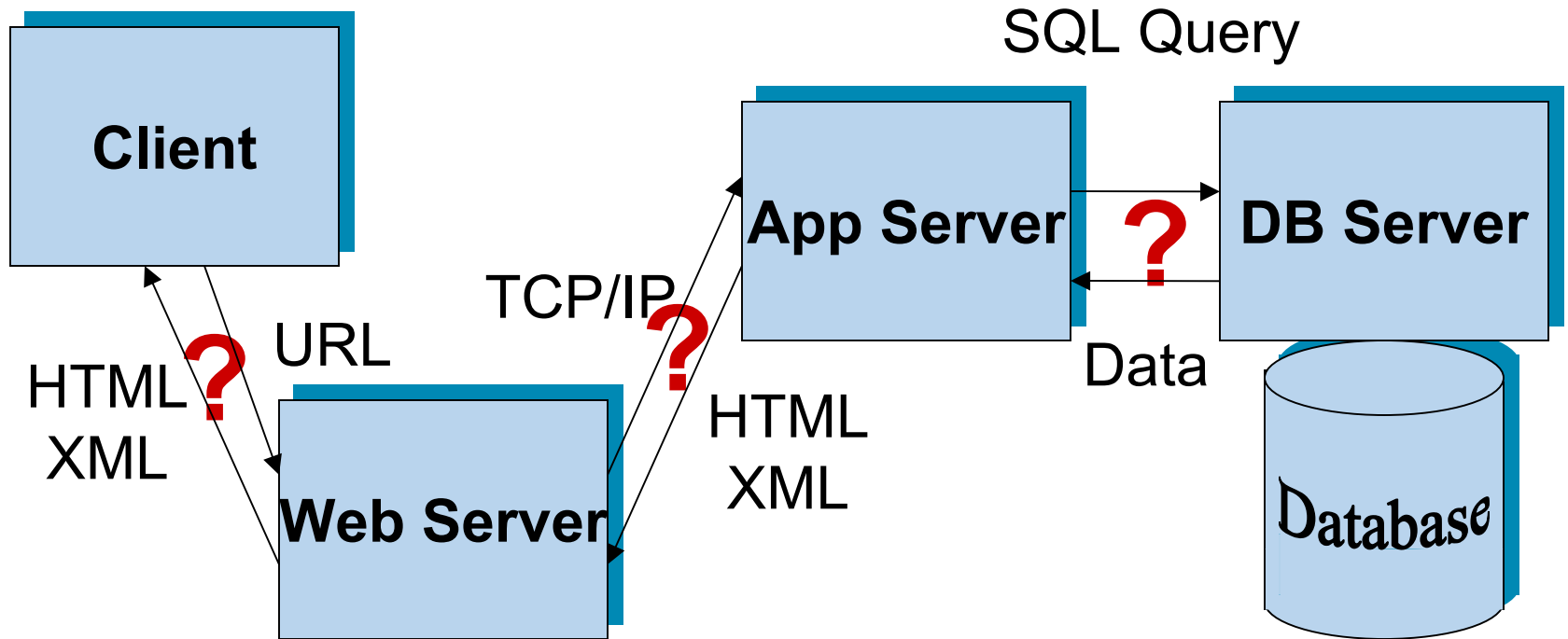
Where's the Performance Problem?



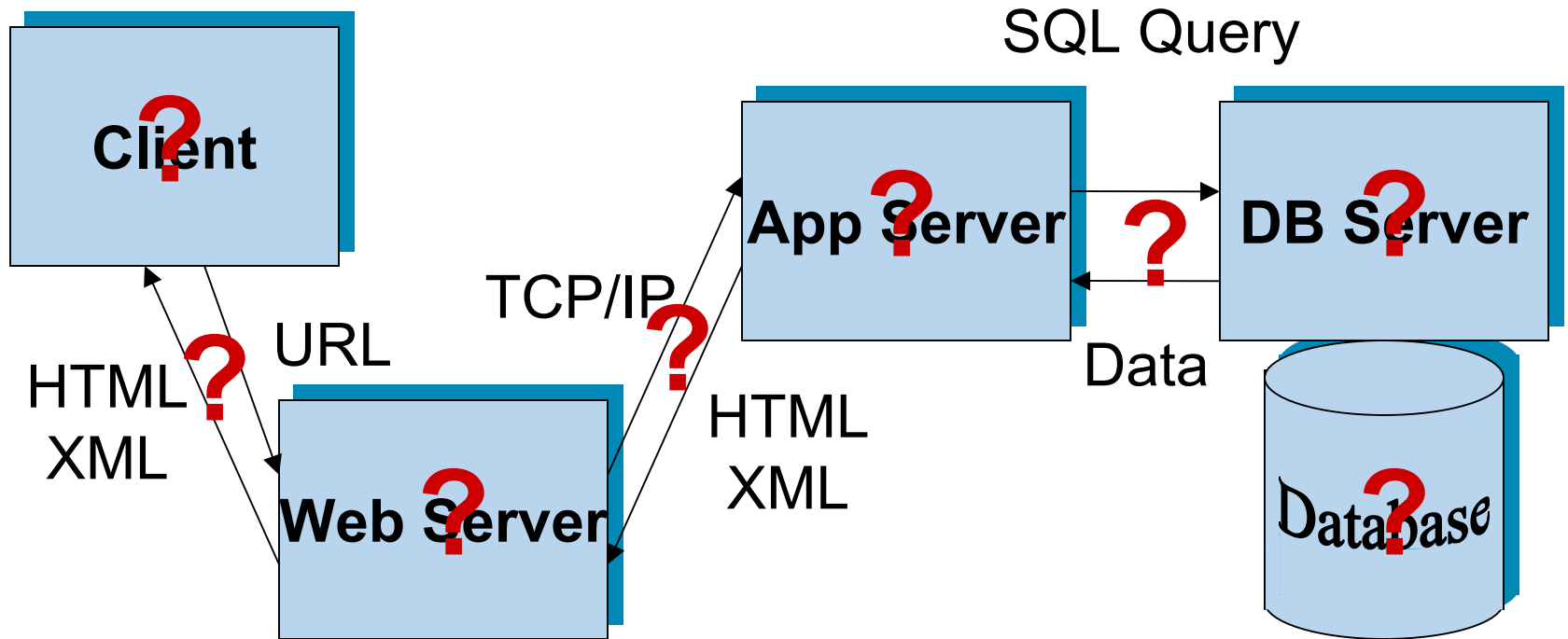
Where's the Performance Problem?



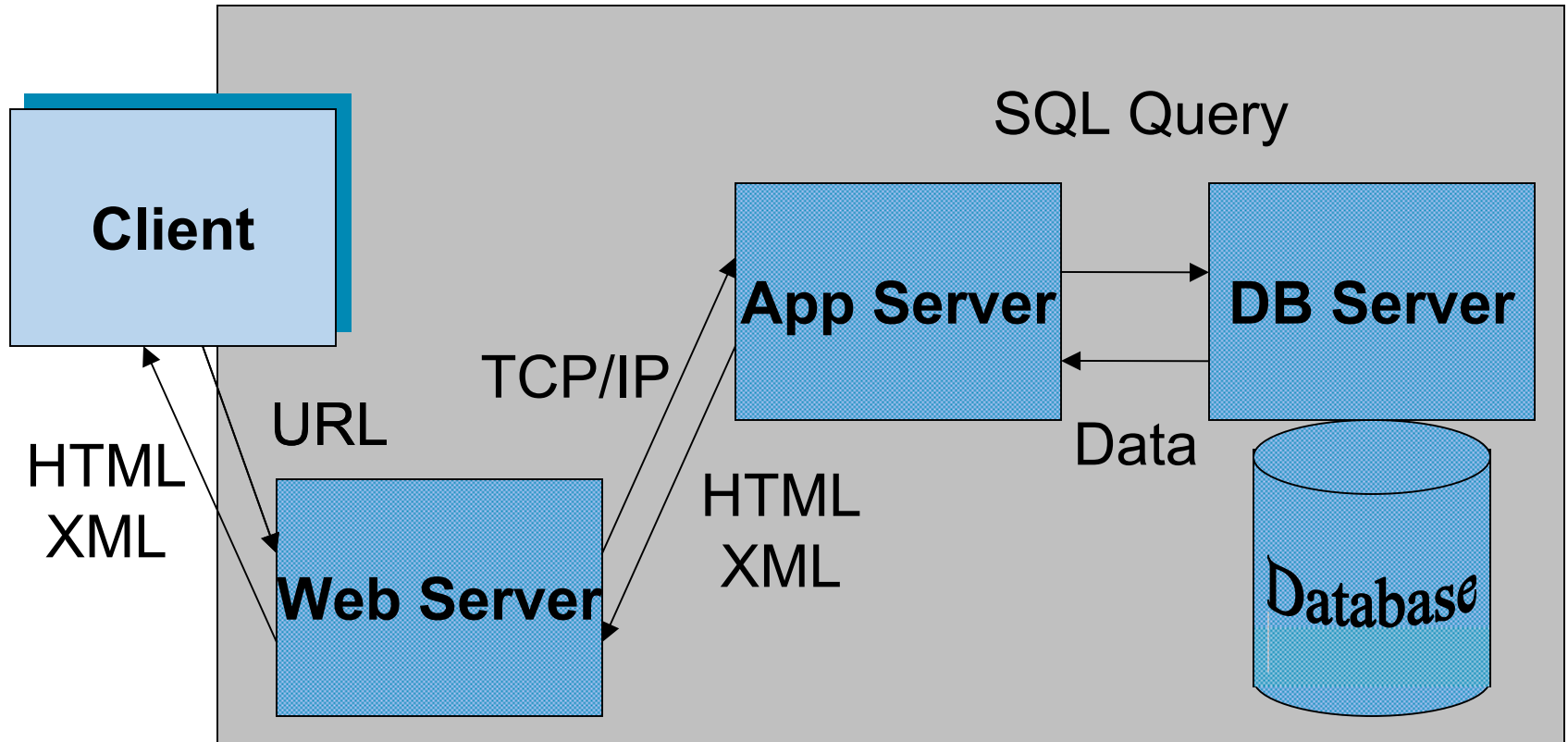
Where's the Performance Problem?



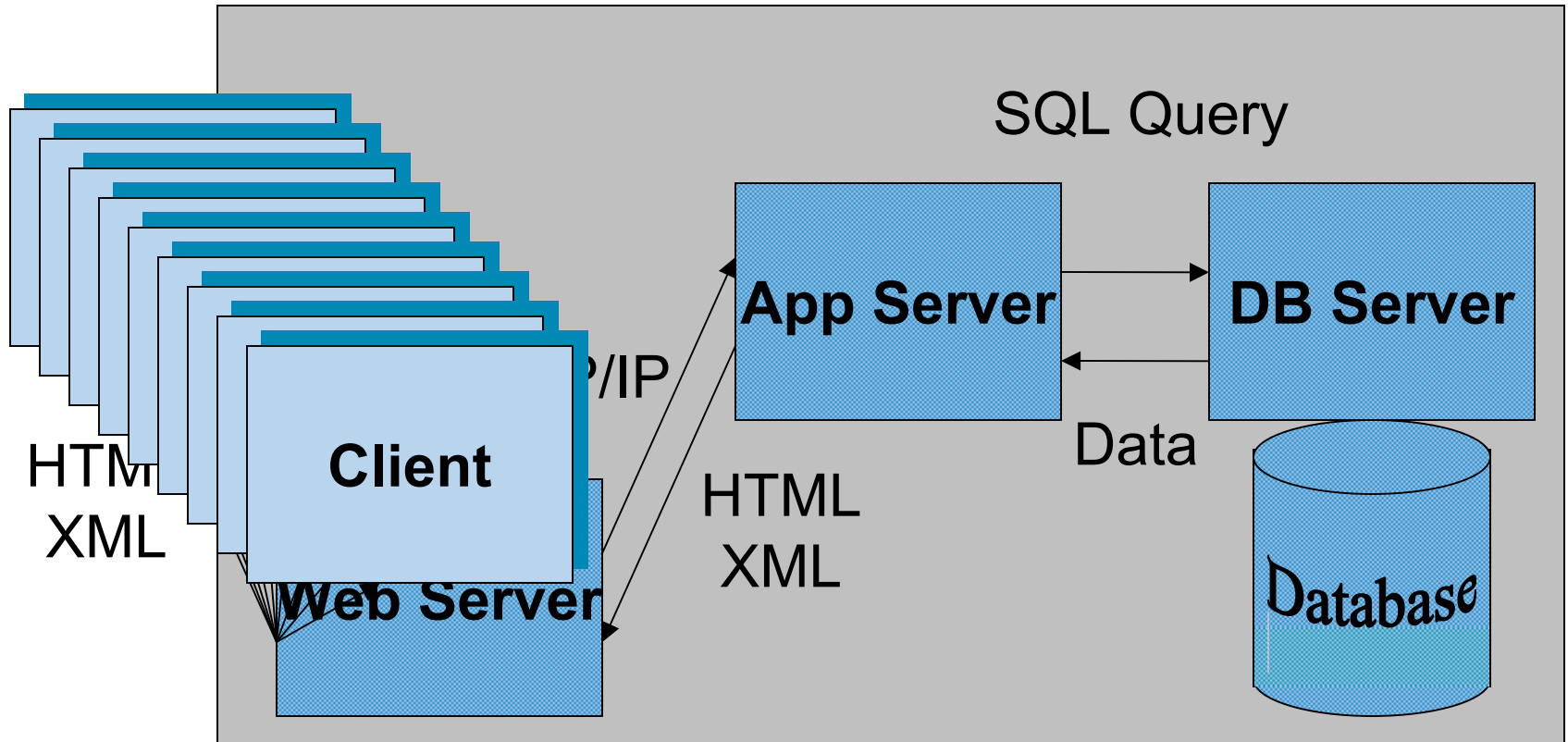
Where's the Performance Problem?



Where's the Performance Problem?



Where's the Performance Problem?



Simulate Many, Many Clients

- Load program required to stress system
- Commercial products available:
 - LoadRunner www.mercuryinteractive.com
 - SilkPerformer www.segure.com
 - e-Load www.empirix.com
 - Benchmark Factory www.benchmarkfactory.com
 - WebLoad www.radview.com
- Less expensive/free:
 - Web Application Stress Tool homer.rte.microsoft.com
 - Jmeter java.apache.org
 - LoadItUp www.broadgun.com
- Scripts must mimic expected user behavior
- Scalable loads with no artificial bottlenecks

Performance Tuning Rules

1. Performance depends on many issues
 - You never know beforehand which ones
 - Many dependencies make prediction difficult
2. Tuning -> Trade-offs
 - Understand all of the requirements
3. Measurement perturbs the system
 - Tools take resources used by the application

Performance Tuning Rules

4. Need multiple tools to measure

- Data must be cross-checked between tools
- Each tool has a strength
 - Different metrics
 - Low intrusion
 - More information

Application Server Problem: Find the Bottleneck

- What is a “bottleneck?”
- **Resource demand: higher than availability**

How to Find a Bottleneck? Develop a Process

- **Assess** the problem
- **Measure** the system
- **Analyze** all data
- **Identify** bottlenecks
- **Tune** the system

HP-UX Tuning and Performance: Concepts, Tools, and Methods

Robert F. Sauers, Peter S. Weygant

Performance Analysis Process

Assess the Problem

- System configuration
 - Patches
 - Kernel and networking values
- Create objective, measurable expectations
 - Response time - consistent
 - Throughput - high
 - Recognize: Throughput increase often causes increase in response time variability
 - Example: Lower overall Garbage Collection (GC) time requires a large heap which increases the time required to do each individual GC

Performance Analysis Process

Assess the Problem

- Number of CPU's
- Amount of memory
- Disk space and swap space
- Number of concurrent and max users
- Versions
 - OS version, Java version, Middleware version
- Correct patches for the OS version
- Correct setting of kernel parameter values

Performance Analysis Process

Assess the Problem

- Kernel Parameter Tuning
- HPjconfig
 - Tool to help calculate the best values
 - Suggested starting set based on experience
 - Downloadable from www.hp.com/go/java

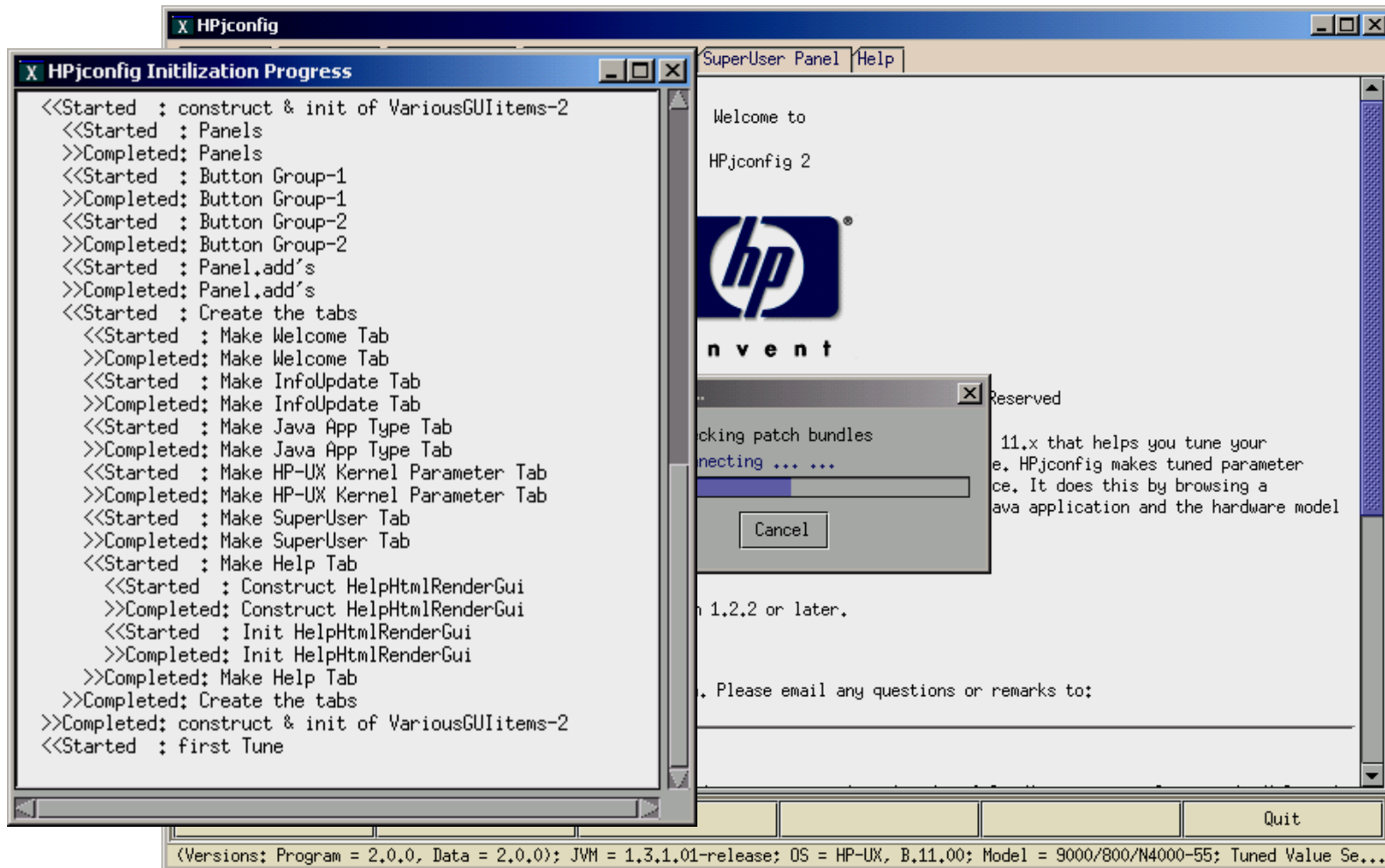
Performance Analysis Process

HPjconfig: Start-up

- Starting HPjconfig:
 - Unzip and untar the file:
 - >gunzip HPjconfig-2.0.tar.gz
 - >tar xvf HPjconfig-2.0.tar
 - Start the program
 - >export DISPLAY=<Display's IP Address>:0.0
 - >export PATH=\$PATH:/usr/sbin
 - >java -cp HPjconfig.jar:HPjconfig_data.jar
HPjconfig

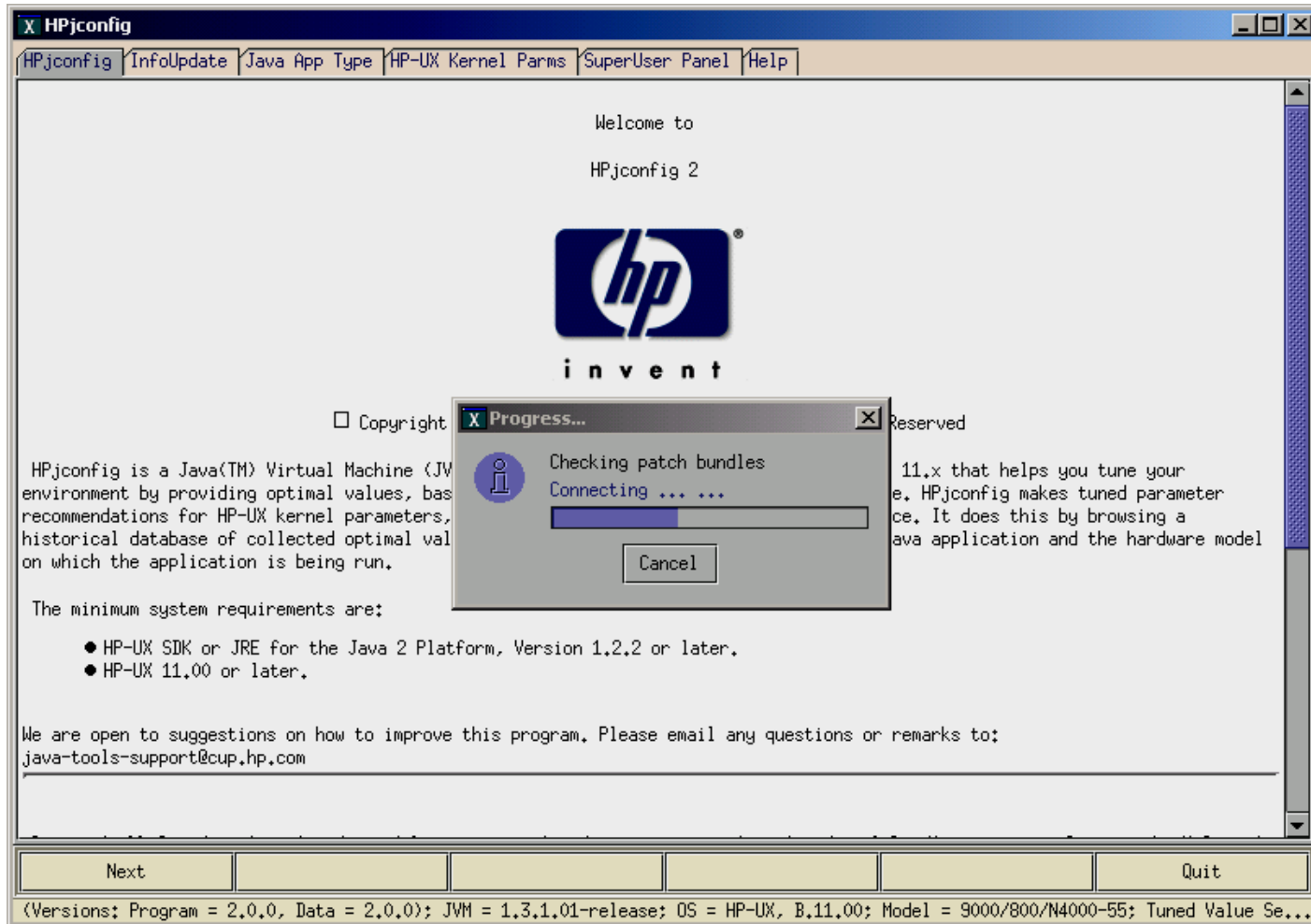
Performance Analysis Process

HPjconfig: Start-up Screen



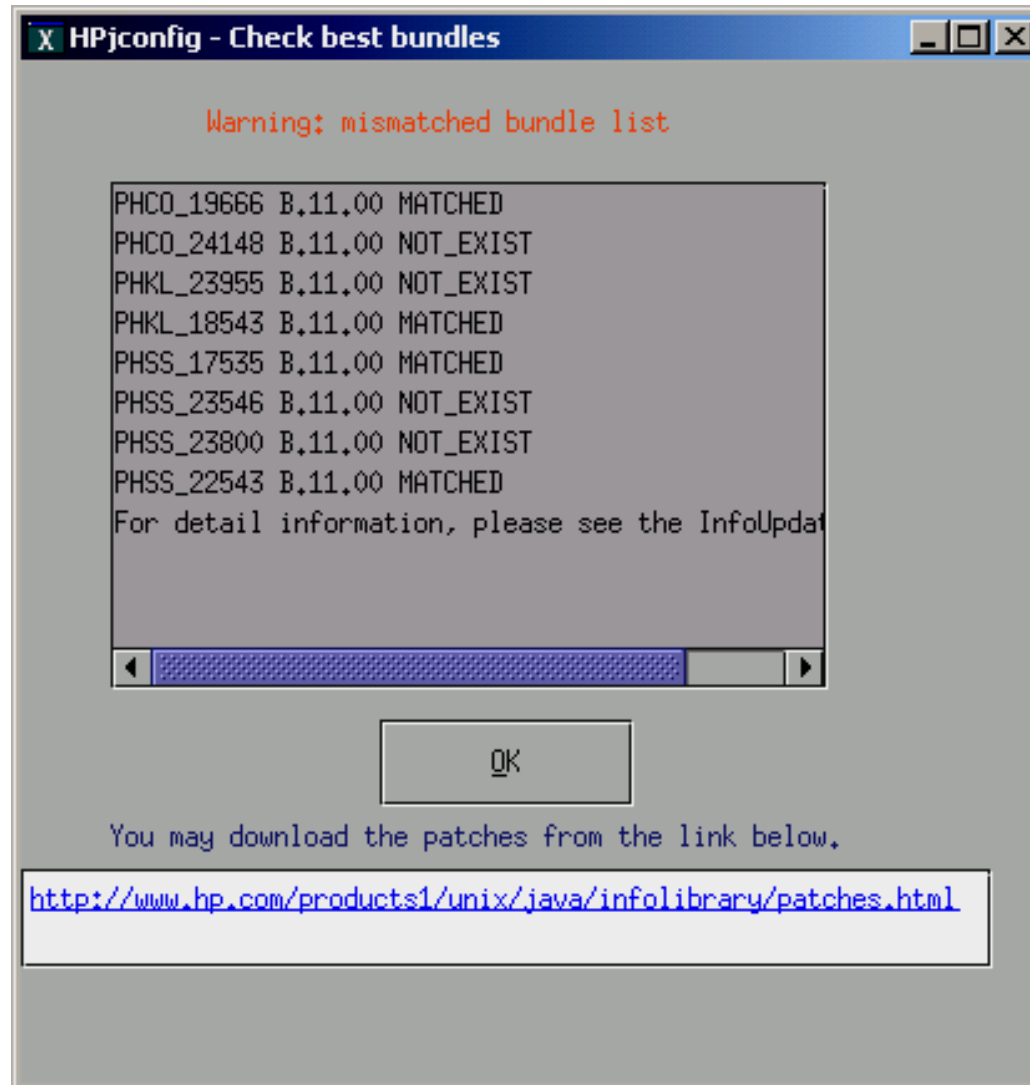
Performance Analysis Process

HPjconfig: Checking Patches



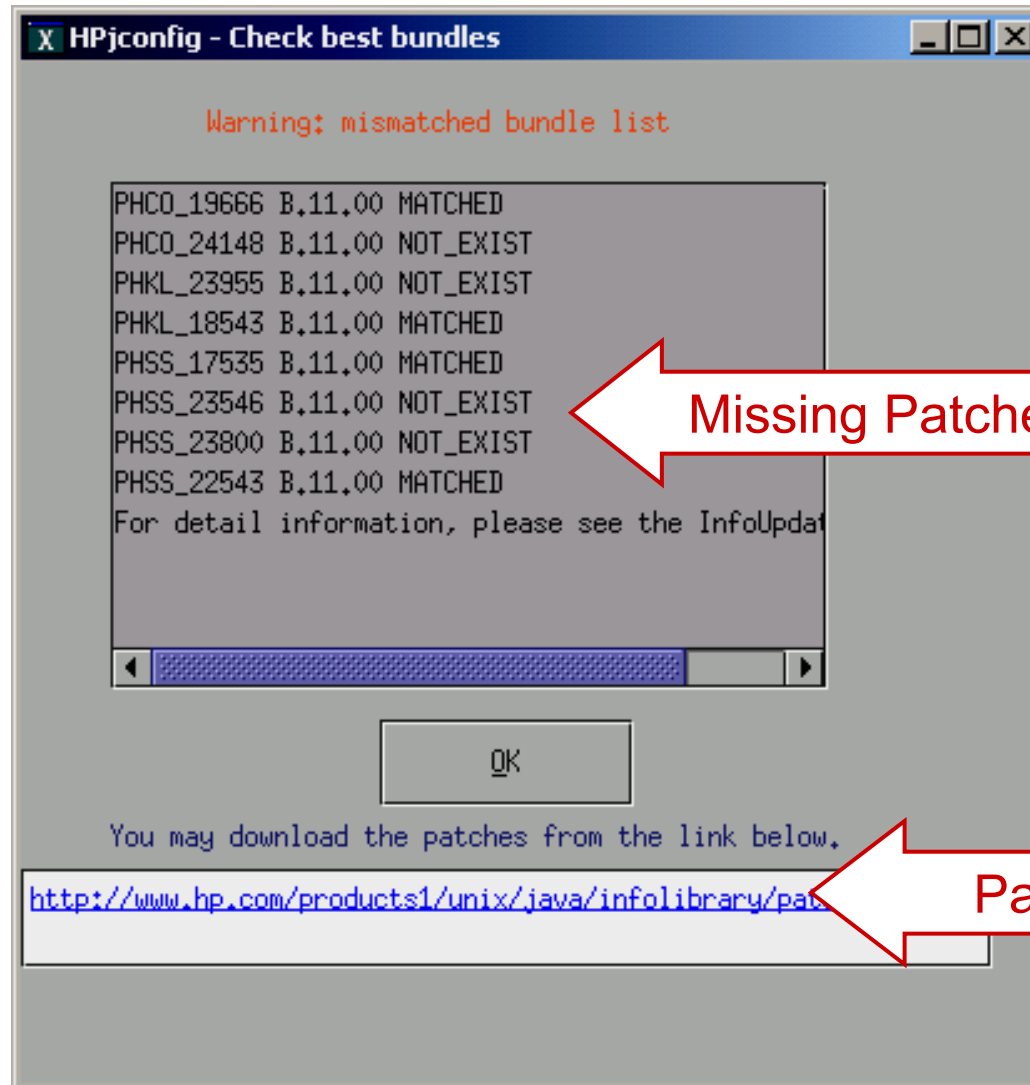
Performance Analysis Process

HPjconfig: Patch Analysis



Performance Analysis Process

HPjconfig: Patch Analysis



Missing Patches

Patch Bundle

Performance Analysis Process

HPjconfig: Information Update

HPJconfig
Information Update

You can download the current version from here.

You may download the required patches at <http://www.hp.com/products1/unix/java/infolib/patches.html> page .

Best tested patch bundles

The best tested product bundle information is from a local cached copy.

PHCO_19666	B.11.00	MATCHED
PHCO_24148	B.11.00	NOT_EXIST
PHKL_23955	B.11.00	NOT_EXIST
PHKL_18543	B.11.00	MATCHED
PHSS_17535	B.11.00	MATCHED
PHSS_23546	B.11.00	NOT_EXIST
PHSS_23800	B.11.00	NOT_EXIST
PHSS_22543	B.11.00	MATCHED

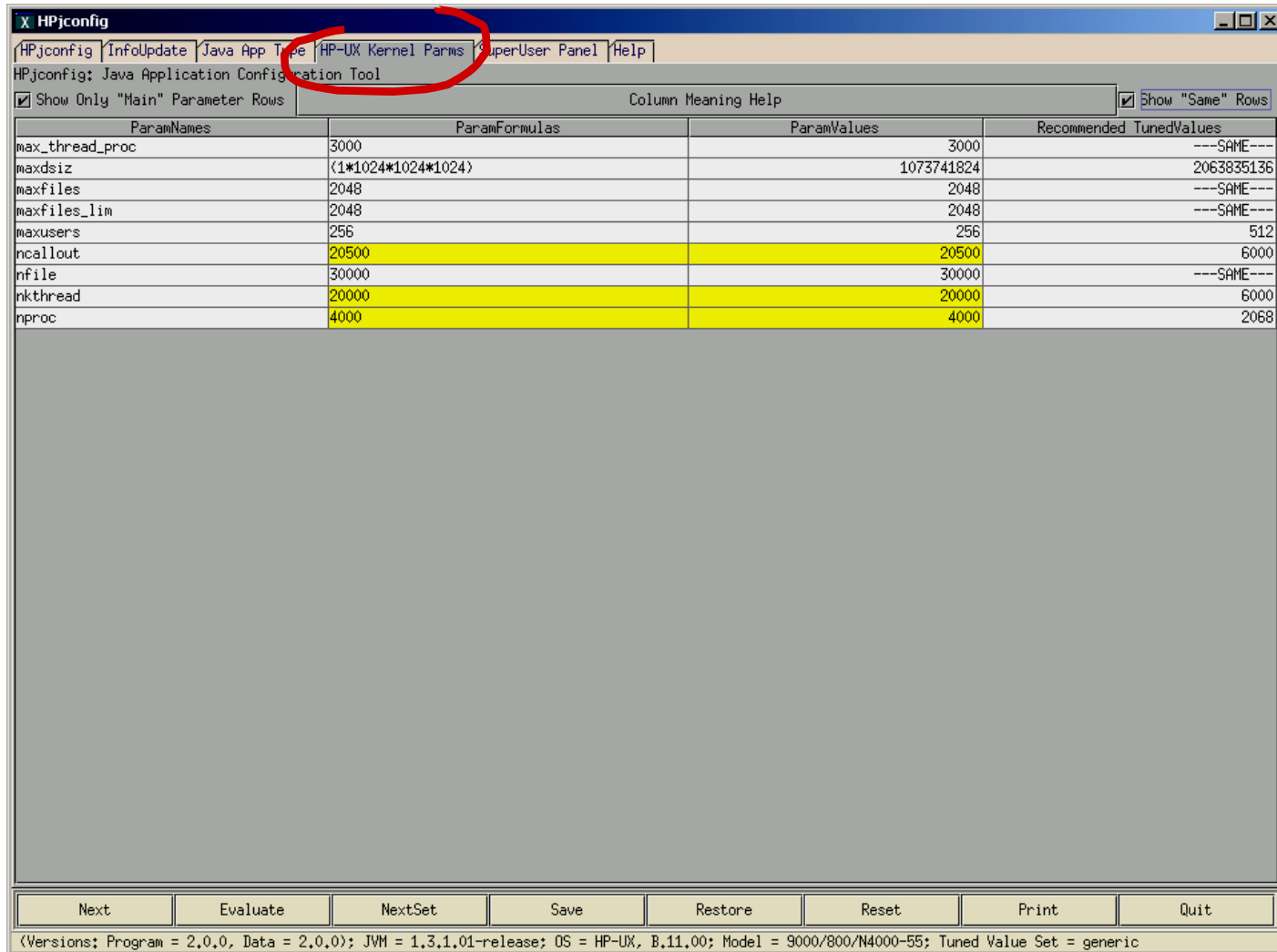
Current patch bundles in the system

Next Quit

(Versions; Program = 2,0,0, Data = 2,0,0); JVM = 1,3,1,01-release; OS = HP-UX, B.11.00; Model = 9000/800/N4000-55; Tuned Value Set = generic

Performance Analysis Process

HPjconfig: Kernel Parameters



ParamNames	ParamFormulas	ParamValues	Recommended TunedValues
max_thread_proc	3000	3000	---SAME---
maxdsiz	(1*1024*1024*1024)	1073741824	2063835136
maxfiles	2048	2048	---SAME---
maxfiles_lim	2048	2048	---SAME---
maxusers	256	256	512
ncallout	20500	20500	6000
nfile	30000	30000	---SAME---
nkthread	20000	20000	6000
nproc	4000	4000	2068

Show Only "Main" Parameter Rows Column Meaning Help Show "Same" Rows

Next Evaluate NextSet Save Restore Reset Print Quit

(Versions; Program = 2,0,0, Data = 2,0,0); JVM = 1,3,1,01-release; OS = HP-UX, B,11,0,0; Model = 9000/800/N4000-55; Tuned Value Set = generic

Performance Analysis Process

HPjconfig: Kernel Parameters

HPjconfig: Java Application Configuration Tool

Show Only "Main" Parameter Rows Column Meaning Help Show "Same" Rows

ParamNames	ParamFormulas	ParamValues	Recommended TunedValues
max_thread_proc	3000	3000	---SAME---
maxdsiz	(1*1024*1024*1024)	1073741824	2063835136
maxfiles	2048	2048	---SAME---
maxfiles_lim	2048	2048	---SAME---
maxusers	256	256	512
ncallout	20500	20500	6000
nfile	30000	30000	---SAME---
nkthread	20000	20000	6000
nproc	4000	4000	2068

Update to Recommended Values

Next Evaluate NextSet Save Restore Reset Print Quit

(Versions; Program = 2,0,0, Data = 2,0,0); JVM = 1,3,1,01-release; OS = HP-UX, B,11,0,0; Model = 9000/800/N4000-55; Tuned Value Set = generic

Performance Analysis Process

HPjconfig: Kernel Parameters

HPjconfig: Java Application Configuration Tool

Show Only "Main" Parameter Rows Column Meaning Help Show "Same" Rows

ParamNames	ParamFormulas	ParamValues	Recommended TunedValues
max_thread_proc	3000	3000	---SAME---
maxdsiz	(1*1024*1024*1024)	1073741824	2063835136
maxfiles	2048	2048	---SAME---
maxfiles_lim	2048	2048	---SAME---
maxusers	256	256	512
ncallout	20500	20500	6000
nfile	30000	30000	---SAME---
nkthread	20000	20000	6000
nproc	4000	4000	2068

Next Evaluate NextSet **Save** Restore Reset Print Quit

(Versions; Program = 2,0,0, Data = 2,0,0); JVM = 1,3,1,01-release; OS = HP-UX, B,11,0,0; Model = 9000/800/N4000-55; Tuned Value Set = generic

Performance Analysis Process

HPjconfig: Kernel Parameters Save

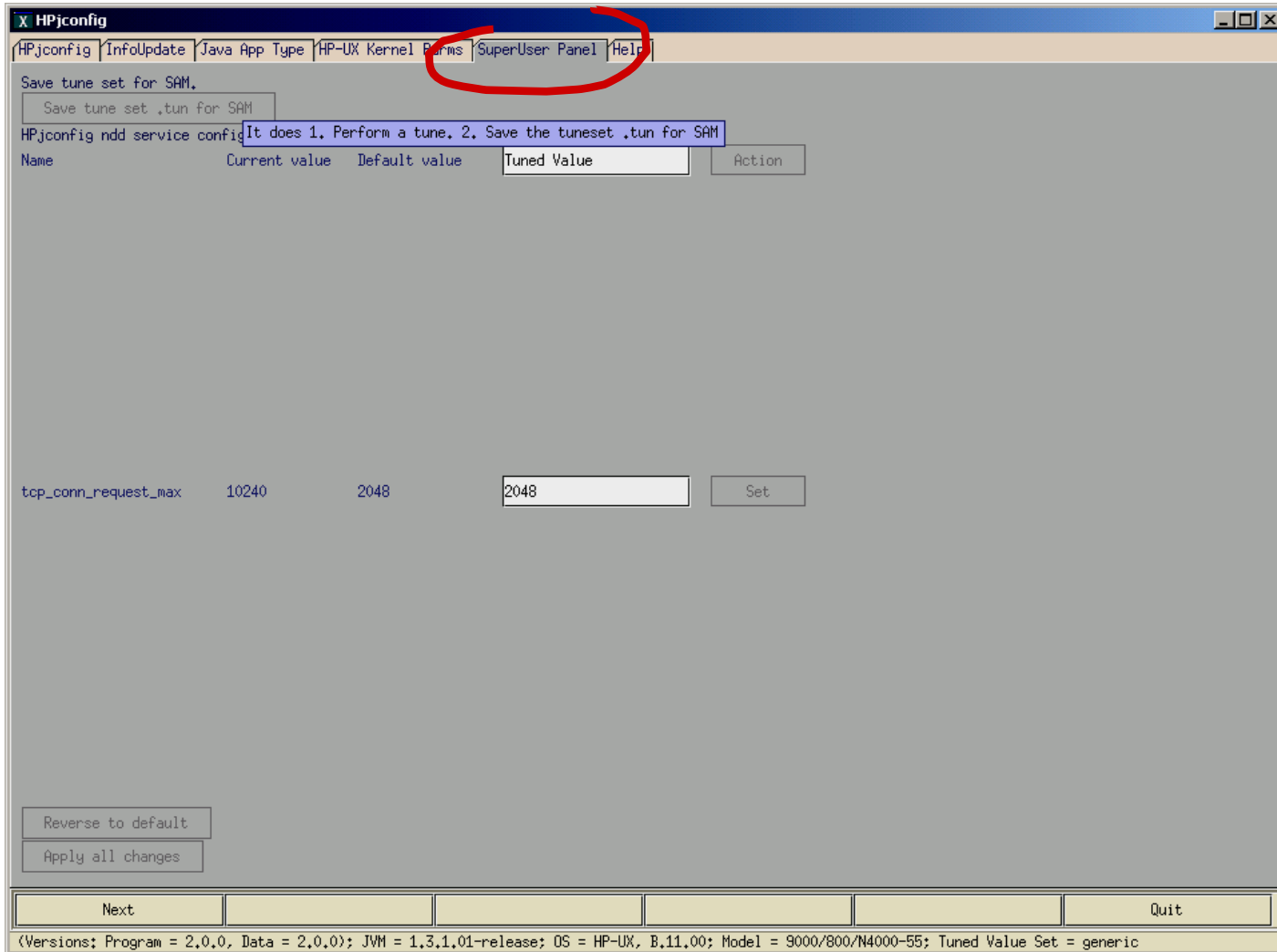
The screenshot shows the HPjconfig application window with a file selection dialog box open. The dialog box prompts the user to enter a path or folder name, which is set to `/extra/coha/HPJCONFIG/`. It also shows a list of files and folders in the selected directory, including `HPjconfig-2.0.tar`, `HPjconfig.jar`, `HPjconfig.release.notes`, and `HPjconfig_data.jar`. The file name `zubat` is entered in the dialog box. The background application window displays a table of kernel parameters.

ParamFormulas	ParamValues	Recommended TunedValues
	3000	---SAME---
	1073741824	2063835136
	2048	---SAME---
	2048	---SAME---
	256	512
	20500	6000
	30000	---SAME---
	20000	6000
	4000	2068

At the bottom of the application window, there is a status bar with the following text: `(Versions; Program = 2.0.0, Data = 2.0.0); JVM = 1.3.1.01-release; OS = HP-UX, B.11.00; Model = 9000/800/N4000-55; Tuned Value Set = generic`

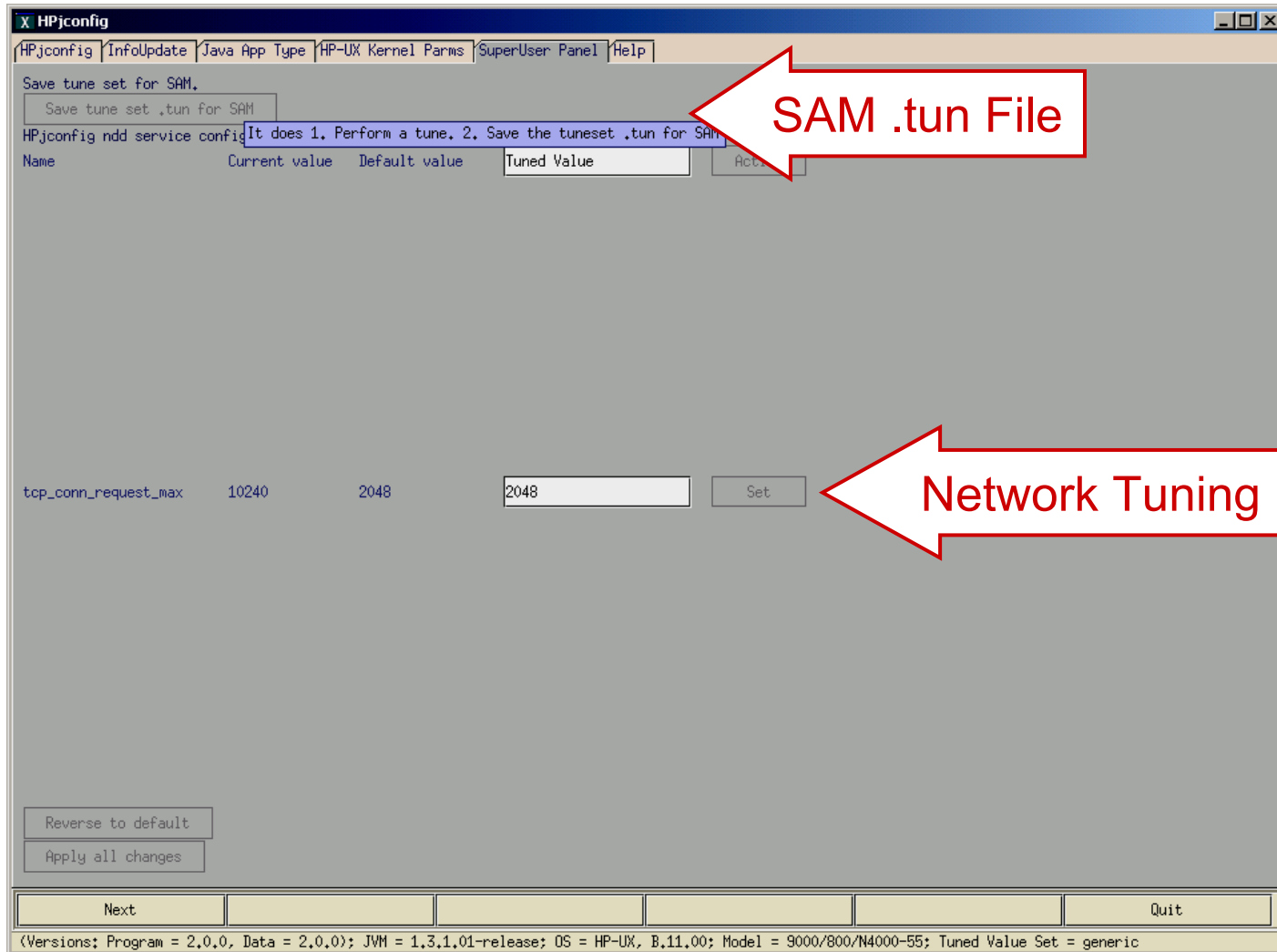
Performance Analysis Process

HPjconfig: SuperUser Panel



Performance Analysis Process

HPjconfig: SuperUser Panel



Performance Analysis Process

Measure the System

- Measure to determine what to tune
- Identify tools available
 - Goal: Low intrusion
- Describe purpose of measurement
- Determine metrics to collect
- Decide required collection time
 - Shorter for easily reproducible
- Identify resources already near saturation
 - Tool intrusion can cause different system behavior

Performance Analysis Process

Measure the System

- Glance/gpm Data Collection
 - CPU usage (across all CPU's)
 - Memory usage
 - Disk usage
 - Network usage

Performance Analysis Process

Measure the System

- Java profiling
 - Methods where most resources consumed
 - Most frequently executed
 - Wall clock time
 - CPU time
 - Objects created
 - High levels of contention
 - Java application profiling
 - java -Xeprof:file=<filename> <ApplicationName>

Performance Analysis Process

Measure the System

- Java Heap and Garbage Collector
 - Frequency of garbage collection (GC)
 - Length of time for each GC
 - Space re-claimed during each GC
 - Patterns of usage of the Java Heap spaces
 - Impact of GC on the allocated heap sizes

- Java Heap profiling

```
java -Xverbosegc:file=<filename> <ApplicationName>
```

Performance Analysis Process

Measure the System

- Thread dump of each thread in the process
 - Determine whether threads are blocked waiting for a monitor
 - Determine whether a deadlock has occurred
 - See the number of concurrent threads

```
kill -s SIGQUIT <pid>
```

pid = Java process identifier

3 times separated by 5 second intervals

Cheap sampling-based performance analysis

Performance Analysis Process

Analyze All Data

- Check all data collected
 - Early pursuit of “obvious” problem -> often misleading
- High resource usage not bad
 - May be the characteristic of the application and the application may require more of the resource
 - Example: Optimized, small processing loop which is CPU intensive – faster CPU will help most

Performance Analysis Process

Identify Bottlenecks

Bottleneck: Limited resource

- Indicators:
 - Resource saturation
 - Growing queue
 - Resource starvation
 - Poor response time

Performance Analysis Process

Tune the System

Increase resource and/or decrease demand

- Devise a test for measuring results
- Organize the tuning effort
 - Modification & test
- Change **one** variable per run
- Prioritize - Look for largest gains
- Know when to stop

The Process

- **Assess** the problem
- **Measure** the system
- **Analyze** all data
- **Identify** bottlenecks
- **Tune** the system

Application Server Problem: Find the Bottleneck

- Running with MAXIMUM load
- System performance:
 - At maximum throughput uses: 25% CPU
 - Expectation is CPU usage should be higher
- How do you find the bottleneck?

Where's the Program Spending Time?

- **Assess** the problem
 - Check system configuration
 - Increase in users should increase throughput
- **Measure**
 - Use system performance tools (Glance)
 - Java tool needed - program state?

Java Tool for Program State

- Check the application while running:
 - **kill -s SIGQUIT <pid>**
 - Interrupts the process
 - Dumps a stack trace of each thread
 - Process continues execution
 - Examine information dumped
 - Many threads running in process
 - 3 are distinct

JAppServ readSockets() for JDK 1.2 and 1.3

"ExecuteThread-0" prio=8 tid=0x1101e8 nid=14 lwp_id=19390

waiting on monitor [0x56838000..0x56838438]

JAppServ.socket.SocketMuxer.readSockets(Compiled Code)

- **waiting on <0x57d4dca0>** (java.lang.Object)

JAppServ.socket.SocketReader.execute(Compiled Code)

JAppServ.myServer.ExecuteThread.run(Compiled Code)

"ExecuteThread-8" prio=8 tid=0x1100e8 nid=12 lwp_id=19385

waiting on monitor [0x56837000..0x56837438]

JAppServ.socket.SocketMuxer.readSockets(Compiled Code)

- **waiting on <0x57d4dca0>** (java.lang.Object)

JAppServ.socket.SocketReader.execute(Compiled Code)

JAppServ.myServer.ExecuteThread.run(Compiled Code)

"ExecuteThread-21" prio=9 tid=0x2a6468 nid=8 lwp_id=19384

runnable [0x56901000..0x56901438]

JAppServ.socket.SocketMuxer.readSockets(Compiled Code)

- **locked <0x57d4dca0>** (java.lang.Object)

JAppServ.socket.SocketReader.execute(Compiled Code)

JAppServ.myServer.ExecuteThread.run(Compiled Code)

Bottleneck: Conclusion

The JAppServ application implemented a polling method that results in large amounts of contention amongst 3 threads for the `java.lang.Object` `<0x57d4dca0>` monitor in `JAppServ.socket.SocketMuxer.readSockets()`

Performance Problems

- Benchmarking
- Scalability

Performance Problem

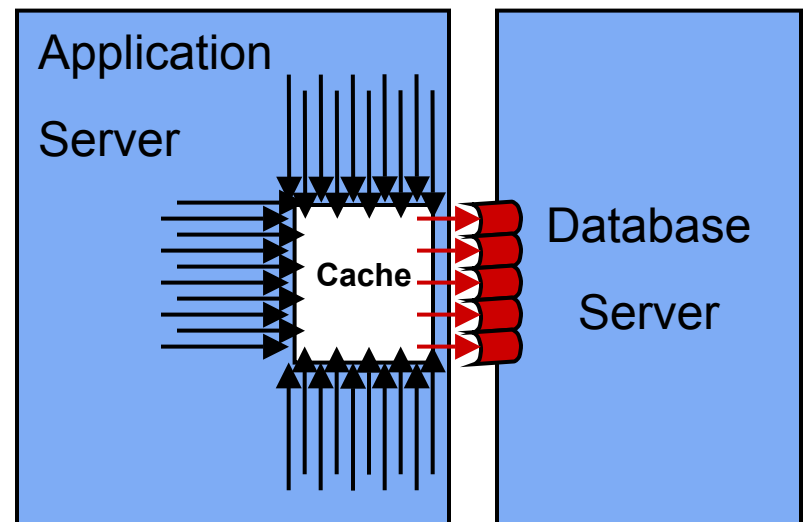
Date: Mon, 06 Mar 2000

From: John Willits johnw@sprint.com

I have developed a “workflow server” that does extensive multi-threading and database operations. It is with this server that I demonstrated a 2X performance degradation when moving from Classic to HotSpot.

First Measurement at HP

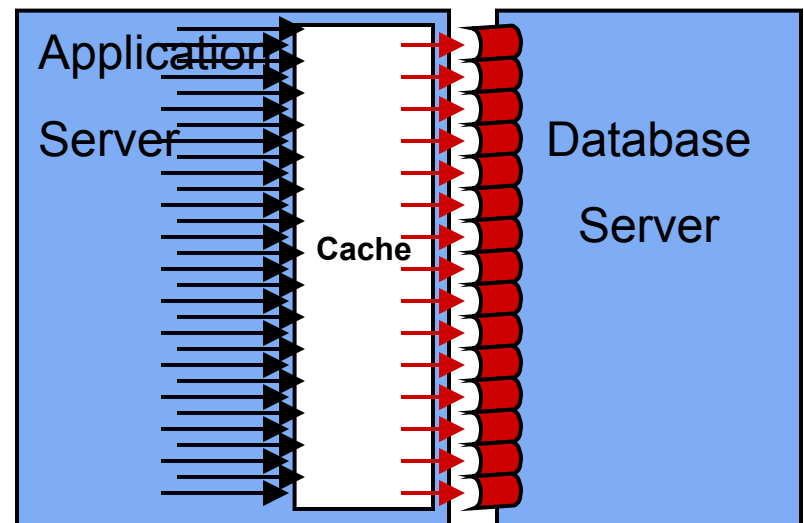
- JDK 1.2.2.03 HotSpot
 - 10% Faster than Classic
 - 115-142 second runs – **Too short!!**
- Parameters to workflow server
 - 32 threads \longrightarrow contending for 5 connections \longrightarrow
 - `server.threads = 32`
 - `database.cache.size = 5`
- Database bound?



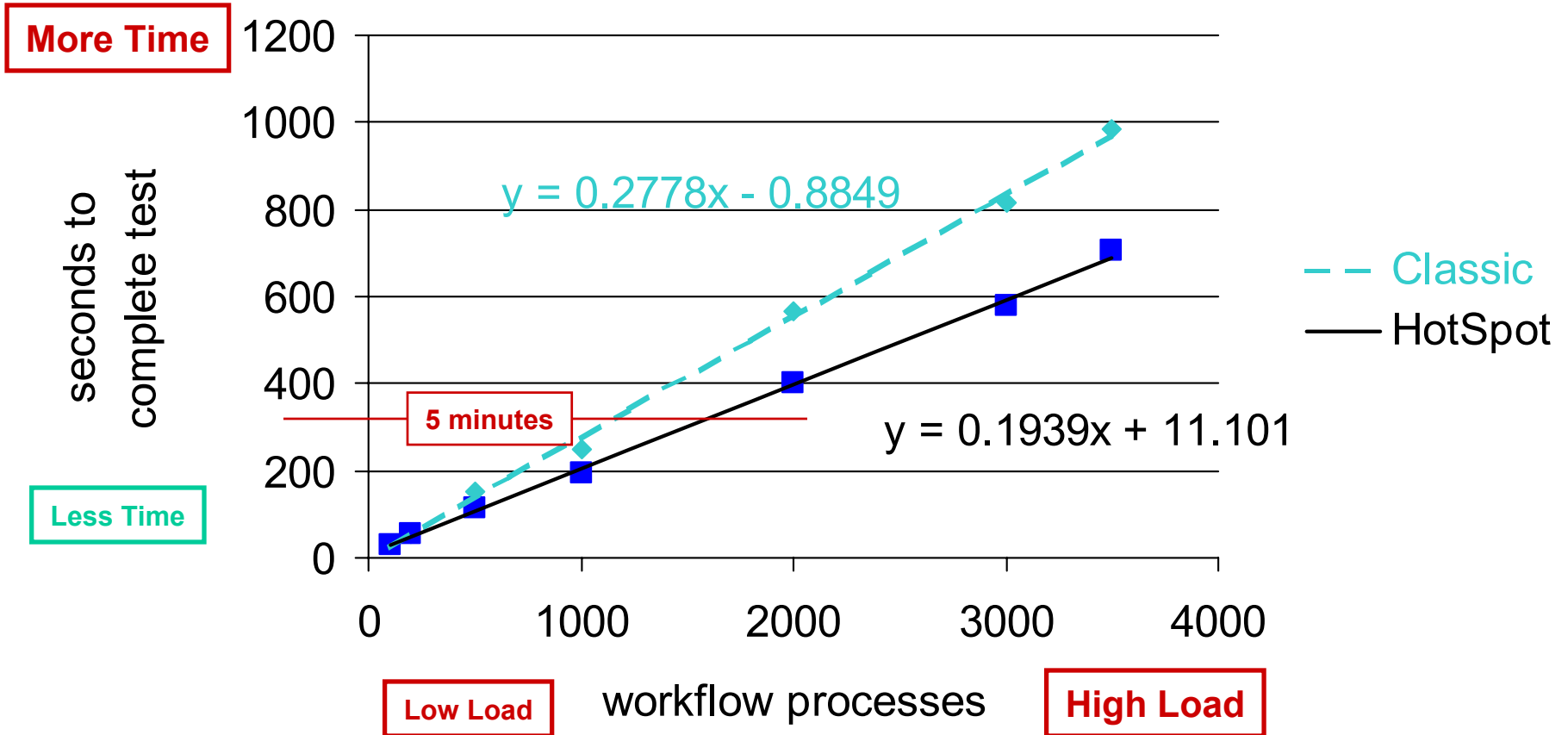
Second Measurement at HP

- Reset parameters to make CPU bound
 - 32 threads contending for 16 connections
 - `server.threads = 32` →
 - `database.cachesize = 16` →
 - `server.process.cache.size = 500` (prefetch)

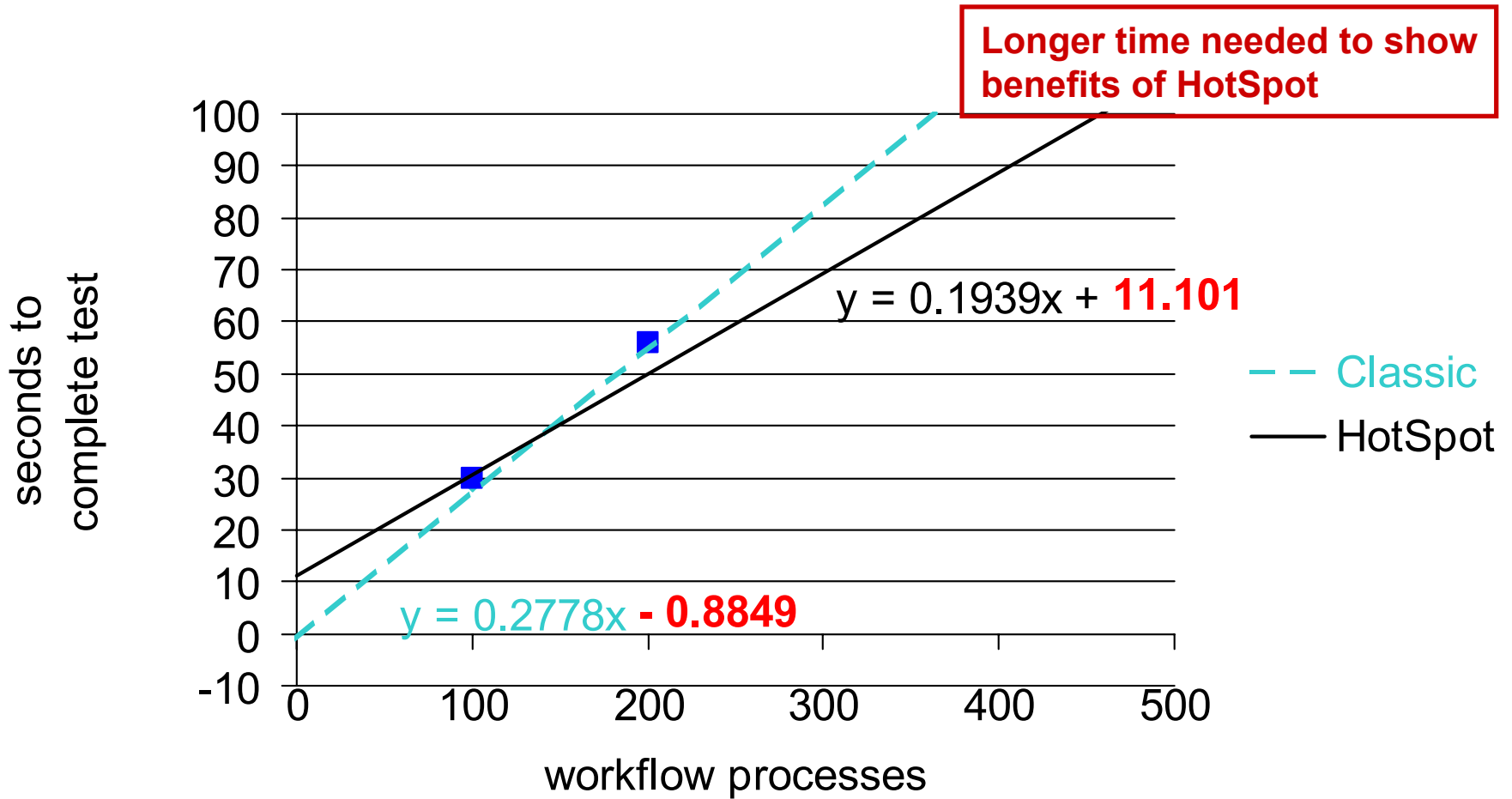
- CPU bound



Conclusion: Classic 43% Slower than HotSpot



Conclusion: Time Needed to Show Benefits



Lessons Learned

Application

System

Scalability

Simulation

Instrument

Use Resources Available



Measure, Measure, Measure

- ***Before*** modification:
 - Set Performance Goals
 - Measure performance
 - Analyze **ALL** performance data

Java Lessons Learned

Application 1

- Design performance into the application
- Test performance **early**
 - Do **NOT** wait until just prior to deployment
 - **Do** make performance measurements:
 - While developing
 - Iterative feedback to design and coding

Java Lessons Learned

Application 2

- Test different configurations:
 - Number of users
 - Amount of data per user
 - Number of processes
 - Number of threads per process
 - Number of threads in each thread pool
- Identifies algorithmic limitations

Java Lessons Learned

Application Problem Areas 1

- Number of threads
 - Tune for maximum throughput
- Number of short-lived objects
 - Minimize
- Java synchronization
 - Minimize time lock held: just critical section

Java Lessons Learned

Application Problem Areas 2

- Minimize communication between processes:
 - On same machine and on different machines
 - Locality is important for fast execution and low latency
- Run processes on different machines:
 - Web Server
 - Application Server
 - Database Server
 - Different demands for each type of application
 - Resource usage
 - Caching at each layer can reduce network traffic to next level

Java Lessons Learned

System Level Analysis

- Install recommended patches
- Set kernel parameters appropriately
 - HPjconfig
- Measure system performance
 - gpm (Glance)

Java Lessons Learned

Java System Level Analysis

- Measure Java application performance
 - Extended profiling (-Xeprof)
 - Java Heap Analysis: Always use -Xverbosegc
 - Use JVMPI: -Xrunhprof
- Analyze collected data
 - Use HPjmeter


Java Lessons Learned

Scalability

- Design Patterns:
 - Best: Locality
 - Make subset of resources thread local
 - Eliminate contention
 - Second: Split single set of shared resources into subsets
 - Reduce contention for single monitor
 - Determine access strategy

Java Lessons Learned

Model Deployment Environment

- Meaningful
 - Model deployment environment using load generation software
 - Number of users
 - Throughput
 - Response times
- Scalable
 - Increase load  create artificial bottleneck
- Consistent, reproducible measure

Java Lessons Learned

Instrument

- Visualize the application's control points
- Insert probes in your code for:
 - Transaction start/end
 - Throughput
 - Buffer sizes
 - Number of threads in pool
 - Active and total
- Use the Java Management Extensions (JMX)
 - Tie-in with your current management framework

Java Lessons Learned

Use Resources Available

- Access to all information:
 - www.hp.com/java
- Developer information:
 - www.hp.com/java - “developer’s connection”
 - Performance Tuning Checklist
 - How to tune your application
- Java application profiling:
 - www.hpjmeter.com

Summary

- Process for solving problems:
 - **Assess** the problem
 - **Measure** the system
 - **Analyze** all data
 - **Identify** bottlenecks
 - **Tune** the system