Performance Tuning Transaction Processing Systems

Dr. Russ Shermer, CSQA, CSTE

Introduction

- Motivation & background
- Comparison of Real-time and Batch
- Terminology & theory
- Discussion of tools and techniques used
- Results summary

Motivation

- Web-based, real-time systems are popular and well-understood
- Backend-transactional systems are still fundamental to most businesses
- The approach to tuning these systems is radically different than for real-time

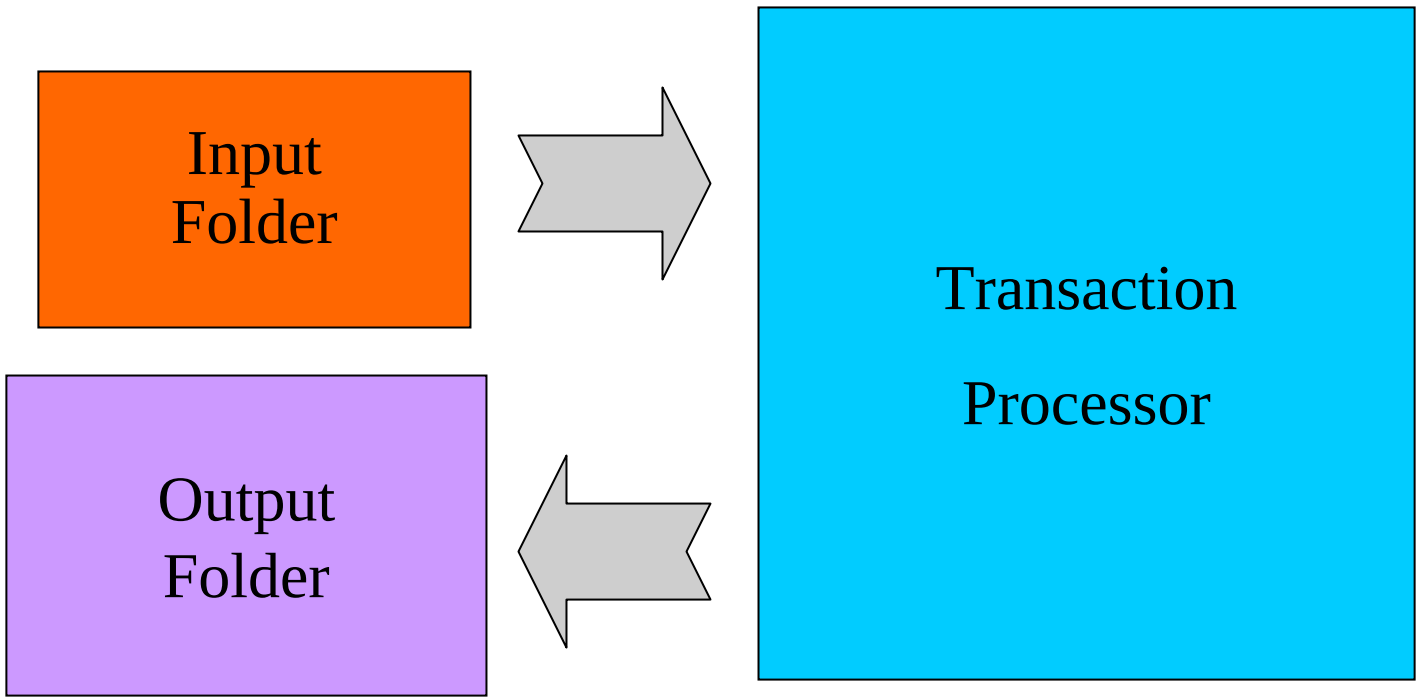
Background

- Large claims processing implementation
- Government-mandated deadline
- 3rd Party Java Application Server platform
- Politics, vendors, customers…

Challenges

- Ill-defined requirements
- Aggressive deadlines
- Standard performance products not effective
- Limited diagnostic tools
- Poor test data
- Off-hours environment access
- Translation of test results to real results

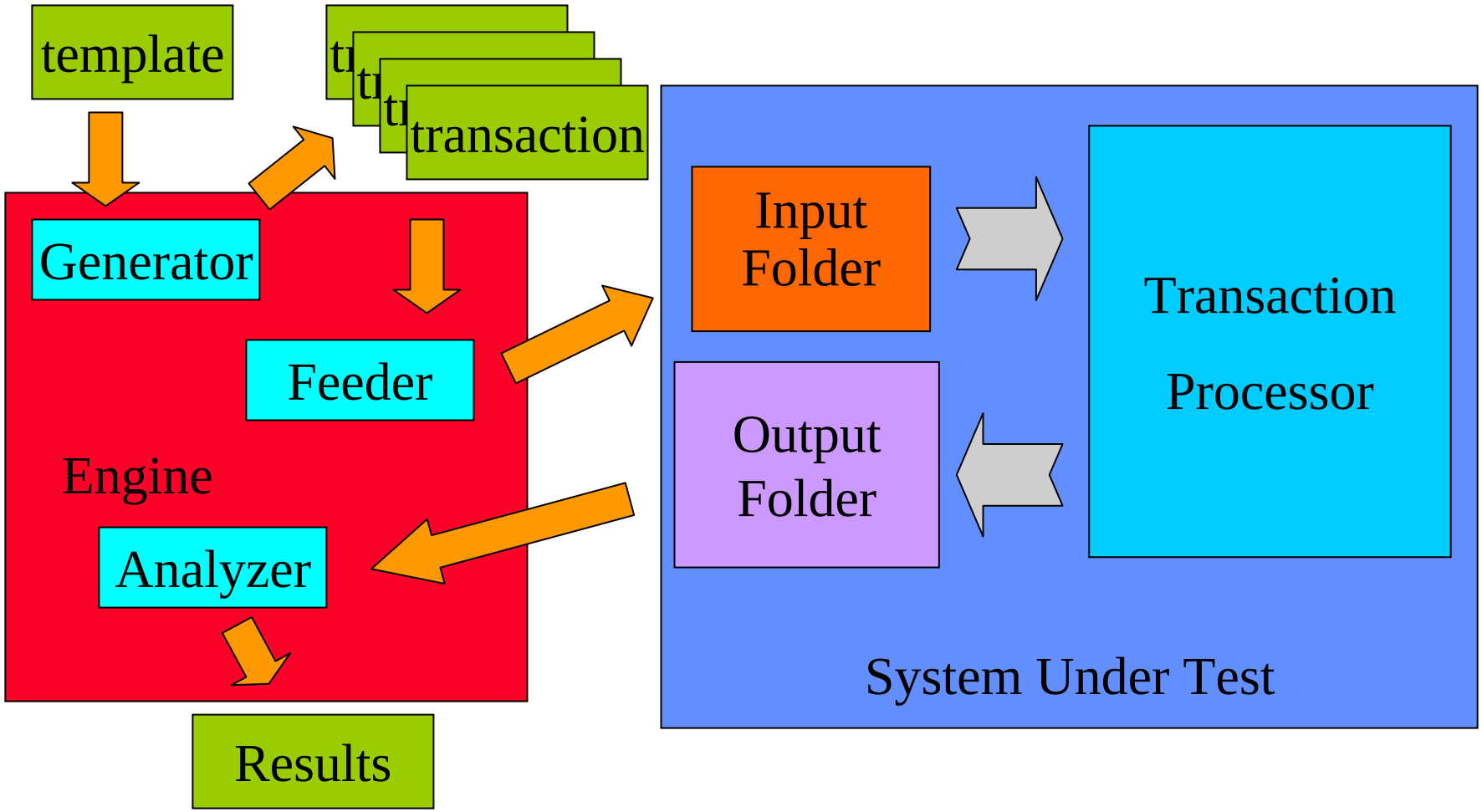
System Under Test



Performance Engine

- Homegrown
- Generates required data prior to the run
- Controlled feed rate
- Calculates throughput after the run via file timestamps

Performance Engine



Batch vs. Realtime

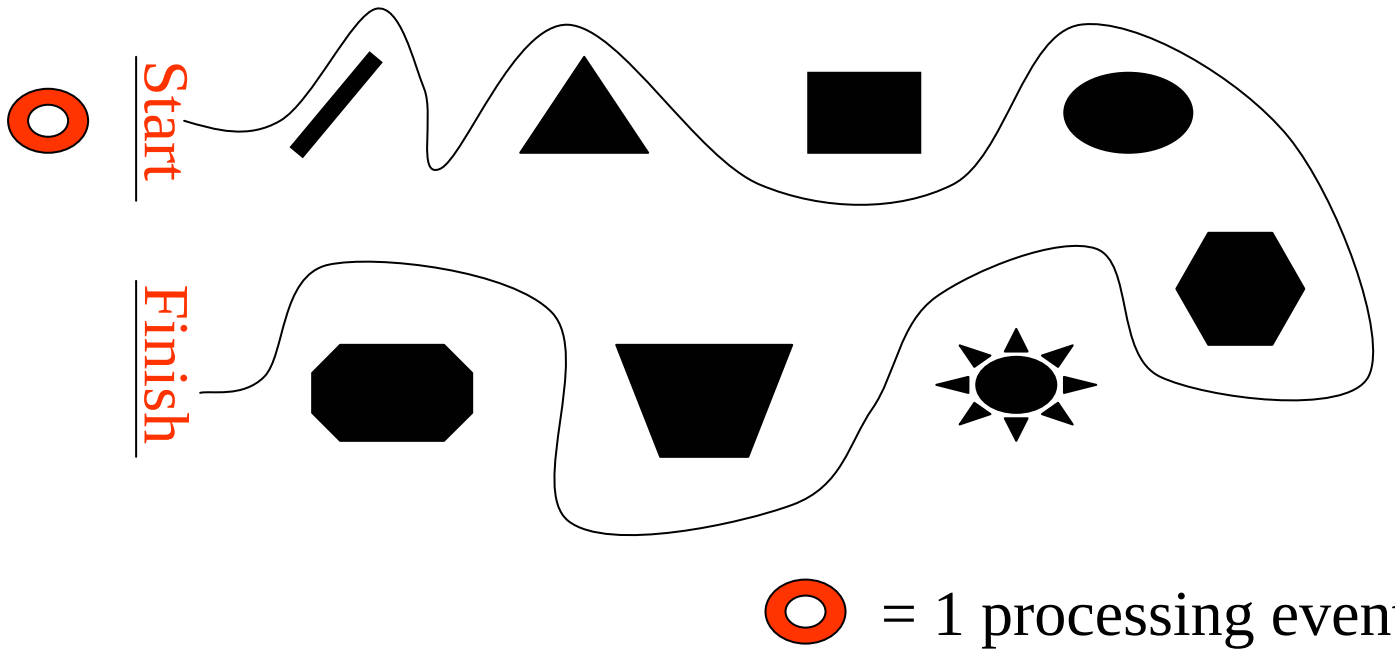
- Realtime: response vs. load is the most important statistic
- Batch: overall throughput is the critical value

Latency vs. Throughput

- Real-time systems need to be low-latency
- Batch systems need high throughput
- Some performance tunings (efficiency types) improve both
- Others improve one at the cost of the other

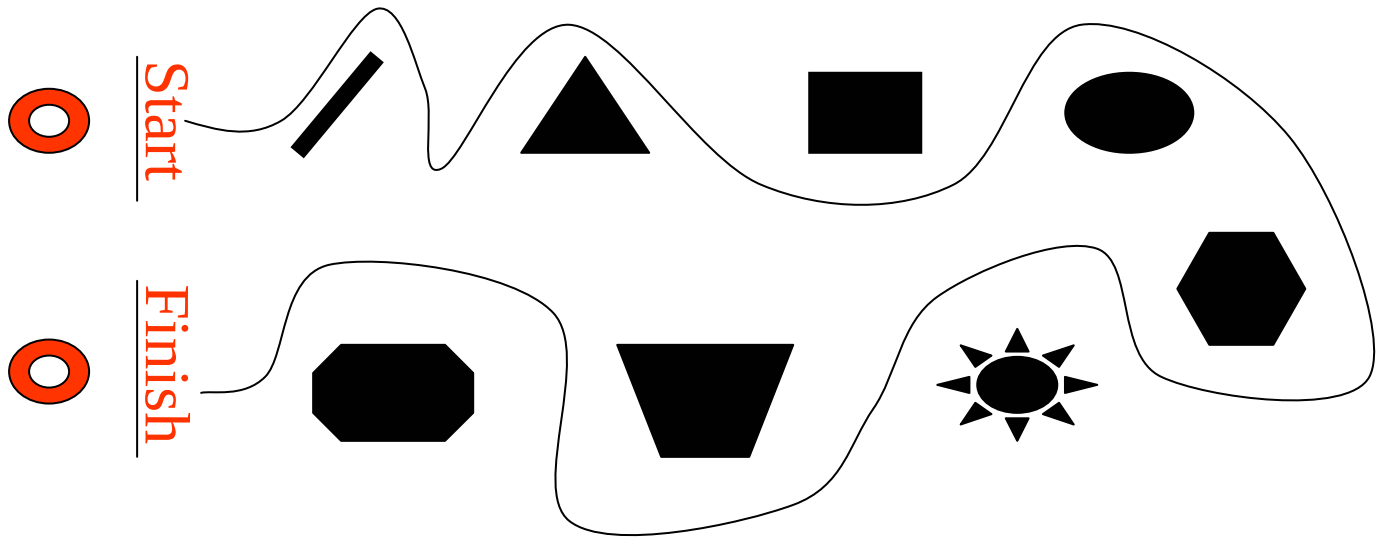
Processing Metaphor

The obstacle course:



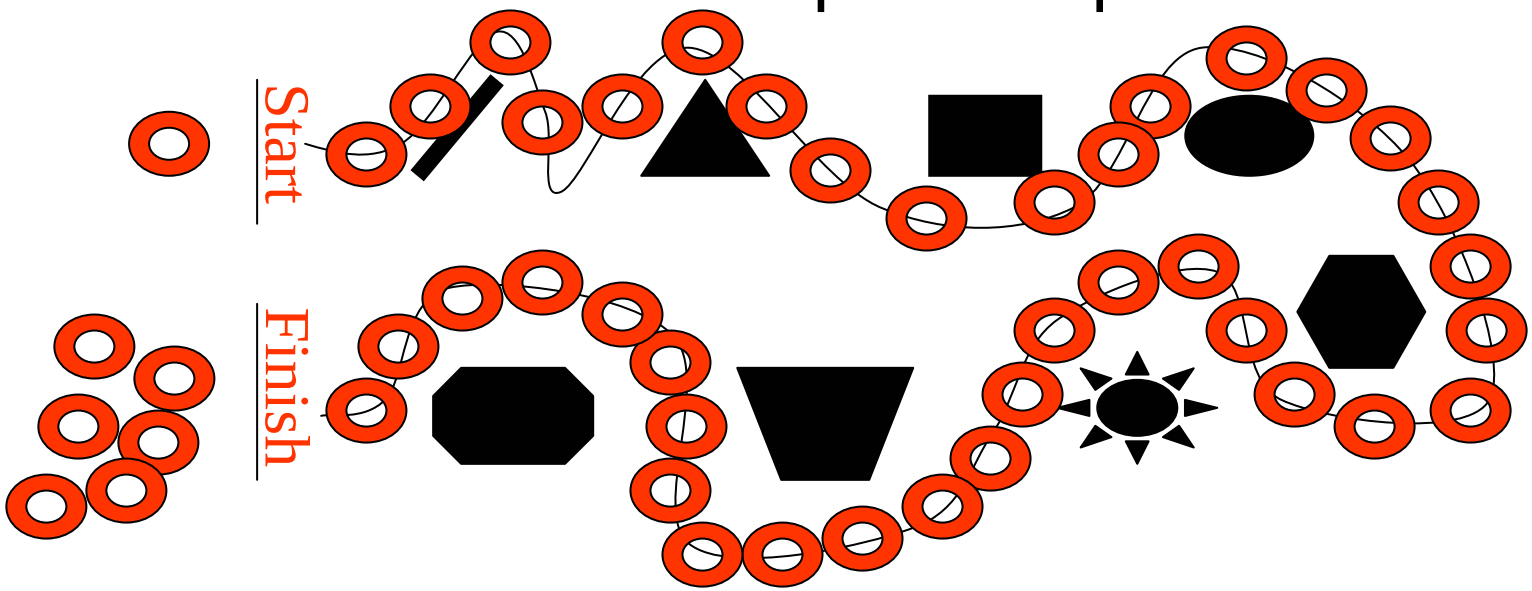
Latency

- Time for one event to complete



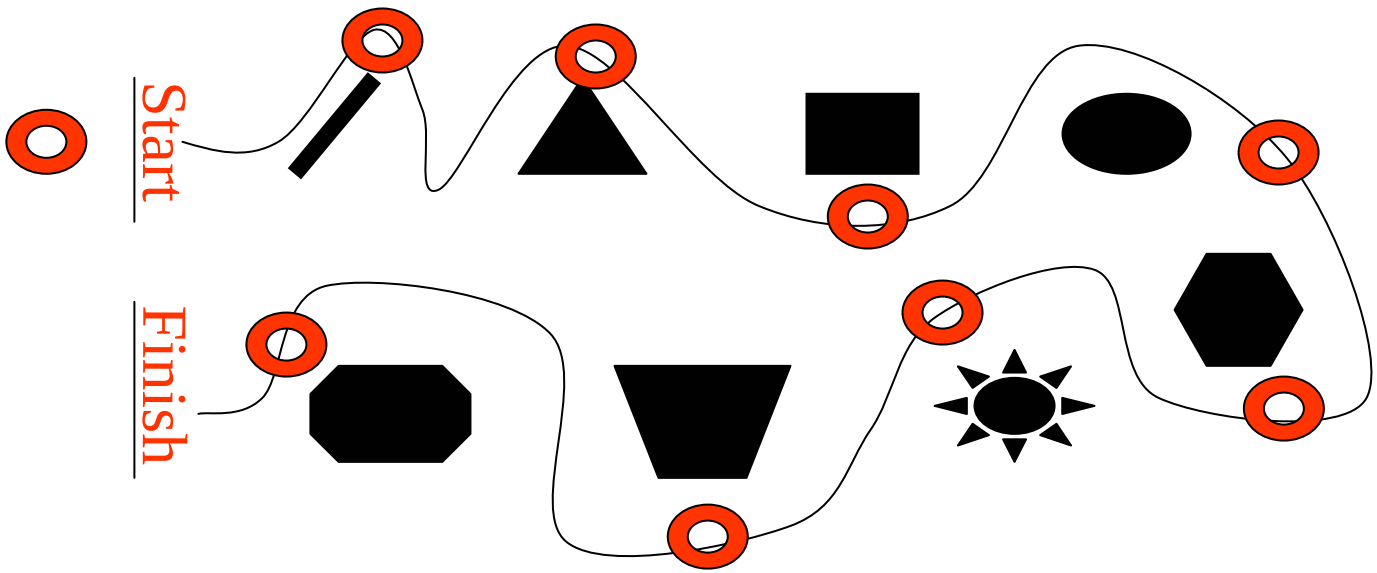
Throughput

- Number of event completions per second



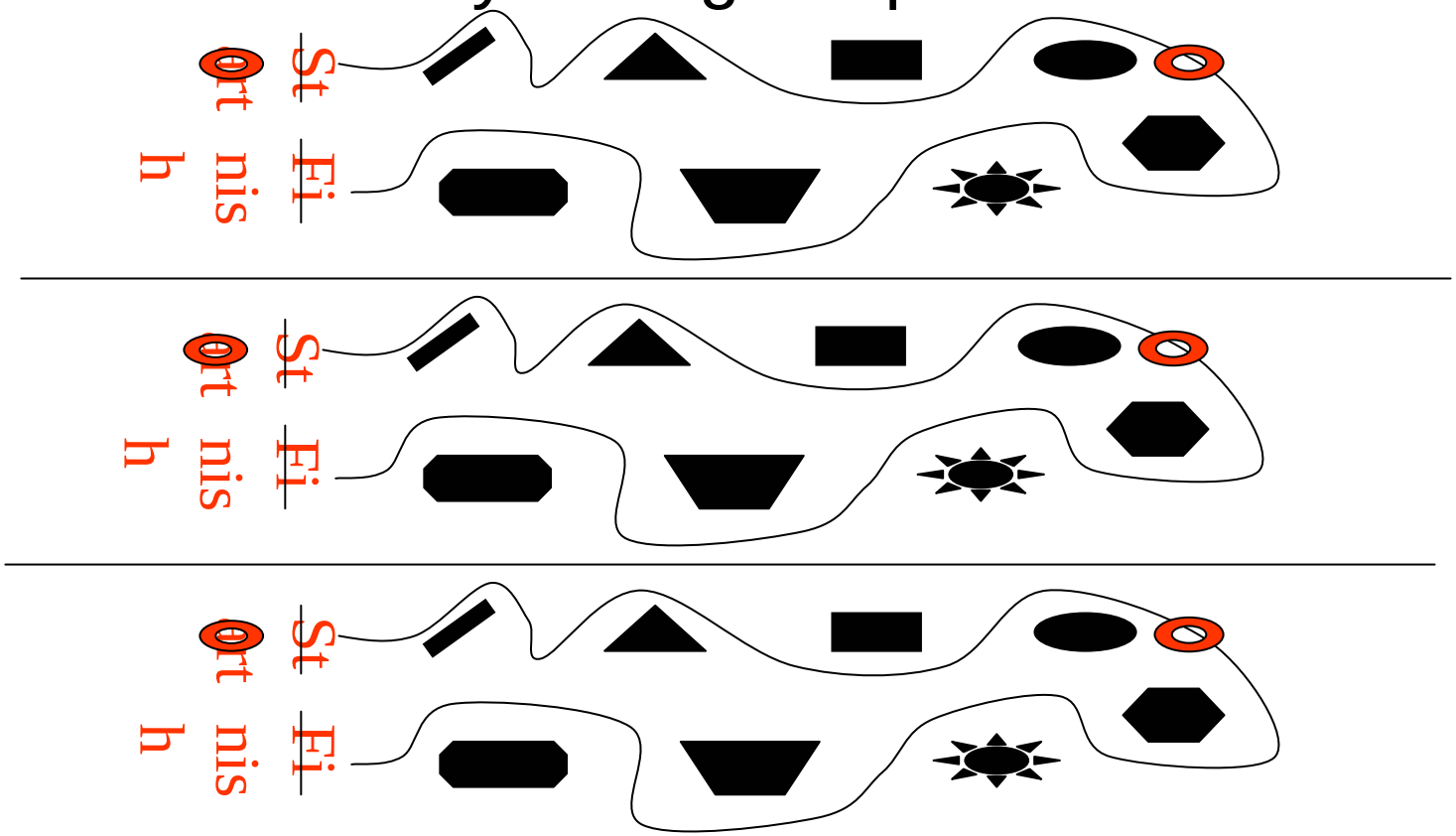
Pipelining

- Concurrent processing through serialization



Parallelism

Concurrency through duplication:



Tuning Techniques

- Test system was nearly identical to production except for number of CPUs in the application server
- Maximize throughput then:
- Tune to make system “CPU bound”

Tuning Techniques

- Increased CPU Utilization
  - Focused on opening concurrency where needed
  - Easiest to implement
  - Had to watch memory consumption

Tuning Techniques

- Database indexing
- Oracle Statspack for analysis
- Network traffic reduction
- Ntop for analysis

Tuning Issues

- Portions of application were batch, others Real-time
- Memory consumption – hit machine and Java limits
- Occasionally exposed software defects when increasing concurrency

Test Execution

- Fully scripted
  - Shutdown application
  - Reset application data/database
  - Apply new tuning configuration
  - Restart application
  - Start monitoring
  - Run transactions
  - End test
  - Capture results

Scripting Advantages

- Off-hours performance runs
- Required for long test runs
- Allowed / required codification of tuning configuration
- Complete test run history archived

Captured Monitoring

- %CPU by process over time (homegrown)
- Ntop – network utilization (automated via wget)
- Vmstat – virtual memory statistics
- Mpstat – multiprocessor statistics
- Iostat (disk I/O statistics)
- Statspack – Oracle database statistics

Top

```
last pid: 26214;
load averages:  2.64, 2.68, 2.68  13:26:10

128 processes:
124 sleeping, 1 running, 1 zombie, 2 on cpu

CPU states:
45.7% idle, 30.9% user, 21.9% kernel, 1.5% iowait,
   0.0% swap

Memory: 16G real, 8417M free, 11G swap in use, 9396M
   swap free
```

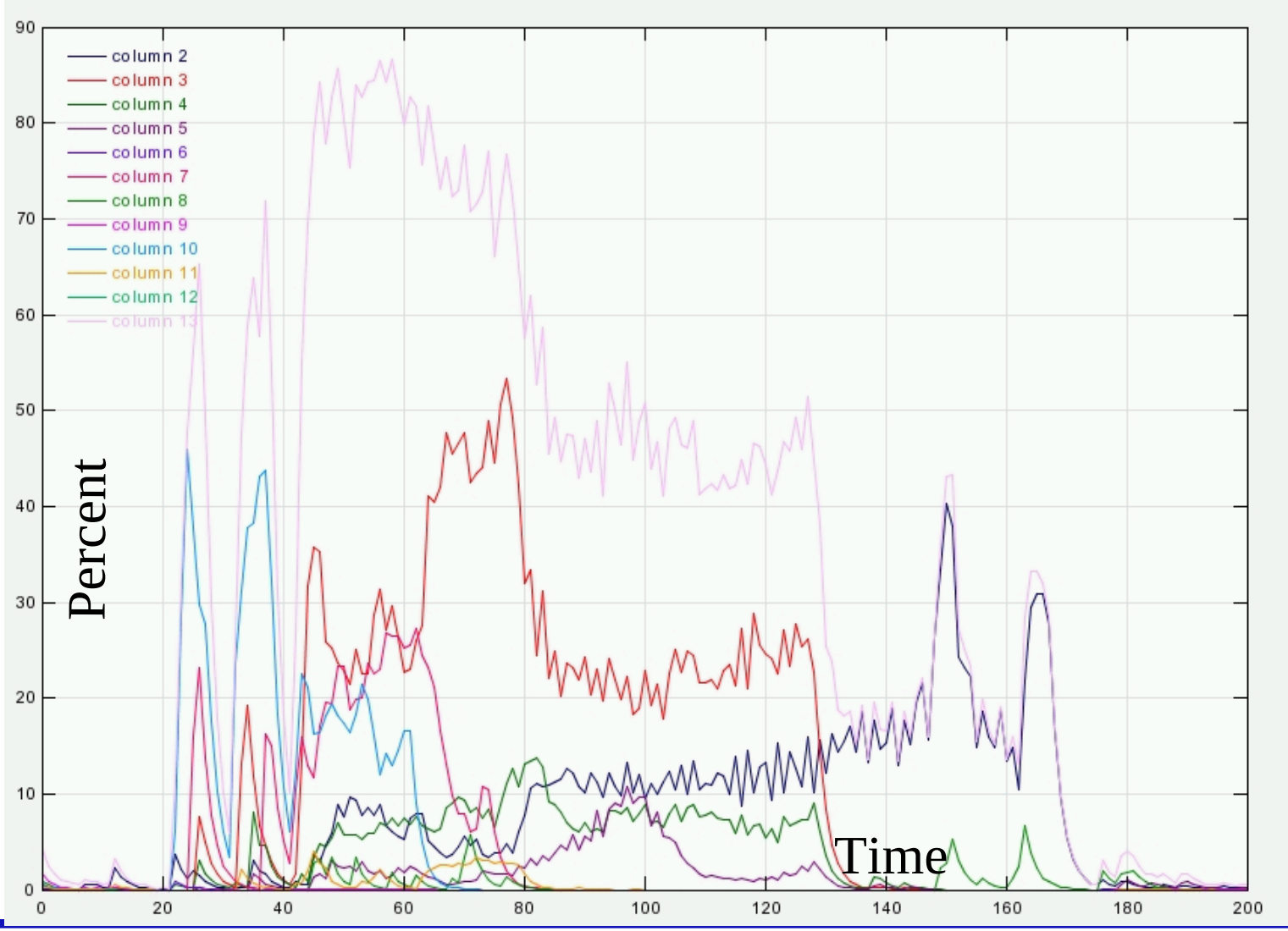
Top

PID	USERNAME	THR	PRI	NICE	SIZE	RES	STATE	TIME	CPU	COMMAND
12922	vitria01	14	0	0	632M	207M	cpu/0	171.6H	24.73%	java
12935	vitria01	13	0	0	159M	115M	run	147.6H	14.65%	java
15594	vitria01	13	59	0	148M	103M	sleep	57.5H	13.71%	java
6210	vitria01	52	59	0	379M	337M	sleep	115:21	1.29%	java
26143	a0cs73	1	2	0	2816K	1704K	cpu/1	0:00	0.13%	top
27629	vtstbot1	1	0	0	1520K	1208K	sleep	16:12	0.08%	HTVQServ
25990	root	1	3	0	3112K	2080K	sleep	0:00	0.08%	sshd
25360	vitria05	12	59	0	151M	67M	sleep	0:38	0.06%	java
7484	vitria03	27	59	0	1043M	603M	sleep	8:49	0.02%	chanserv
17278	vitria05	27	59	0	213M	78M	sleep	1:06	0.02%	java
1094	root	1	0	0	1104K	824K	sleep	2:42	0.01%	hardmond
24759	vitria05	11	59	0	218M	79M	sleep	0:12	0.01%	java
26142	a0cs73	1	0	0	2736K	2224K	sleep	0:00	0.01%	tcsh
26080	a0cs73	1	0	0	1864K	1376K	sleep	0:00	0.01%	ksh
12099	root	10	59	0	32M	9144K	sleep	569:39	0.01%	ntop

CPU Utilization vs. Time

- Perl script to capture CPU utilization by process
- Requires that each process is “tagged”
- Provides a visual CPU workload by component for a given test run
- Another homegrown tool

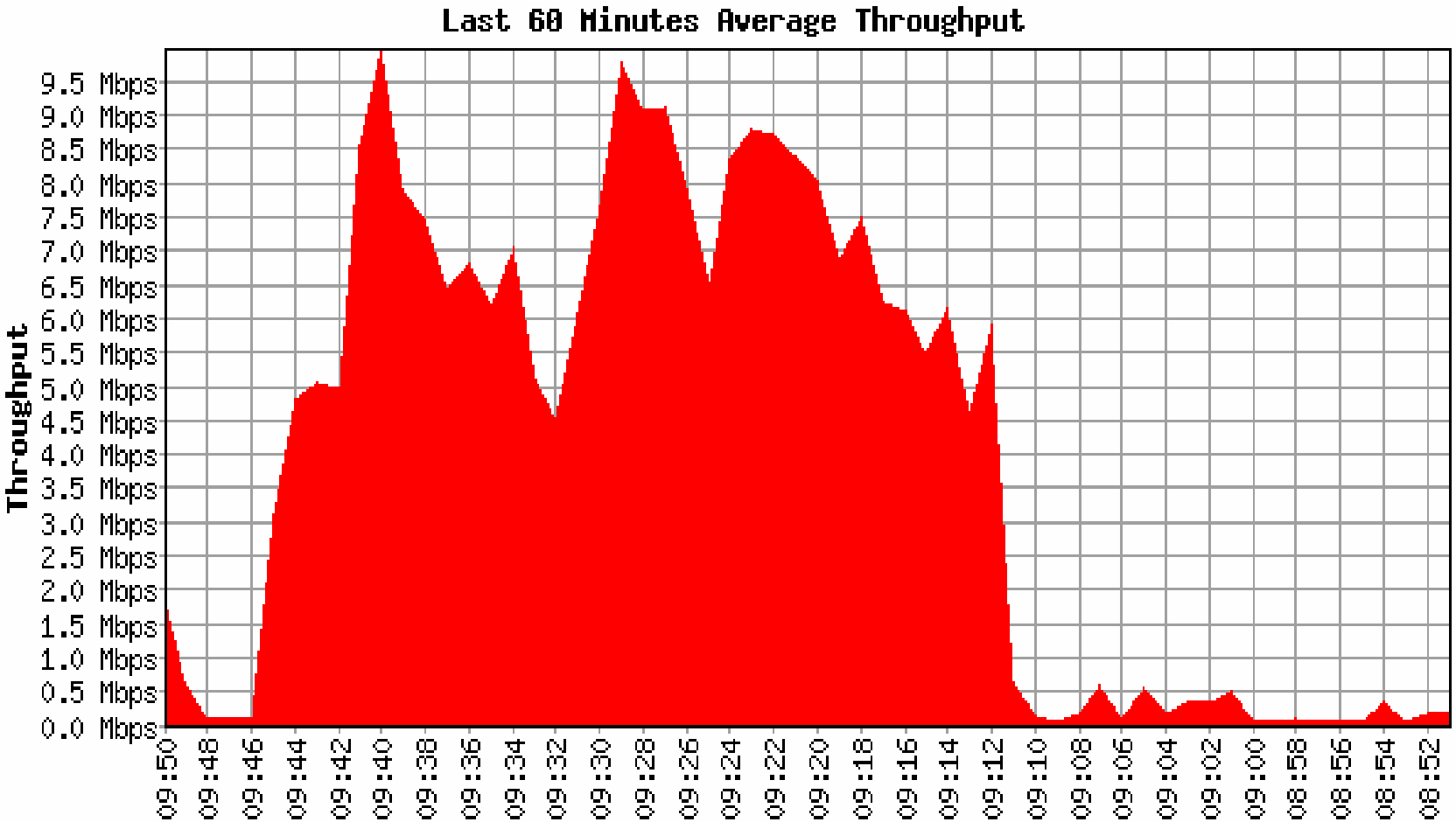
CPU Utilization



Ntop

- “Network Top”
- Freeware network analyzer
- Works by reading IP stack on NIC
- Does not perform individual packet analysis
- Has a web-based reporting interface
- Did I mention it was free?

Ntop


Last 60 Minutes Average Throughput
Throughput
9.5 Mbps
9.0 Mbps
8.5 Mbps
8.0 Mbps
7.5 Mbps
7.0 Mbps
6.5 Mbps
6.0 Mbps
5.5 Mbps
5.0 Mbps
4.5 Mbps
4.0 Mbps
3.5 Mbps
3.0 Mbps
2.5 Mbps
2.0 Mbps
1.5 Mbps
1.0 Mbps
0.5 Mbps
0.0 Mbps
09:50
09:48
09:46
09:44
09:42
09:40
09:38
09:36
09:34
09:32
09:30
09:28
09:26
09:24
09:22
09:20
09:18
09:16
09:14
09:12
09:10
09:08
09:06
09:04
09:02
09:00
08:58
08:56
08:54
08:52

Oracle Statspack

- Allows detailed response analysis
- Rates queries and procs by time and usage
- Identifies important bottlenecks
- Can be run remotely through sqlplus

Oracle Statspack

- Capture a “before” snapshot
- Run performance test
- Capture an “after” snapshot
- Run the statspack analyzer on the two snapshots
- Produces a differential report

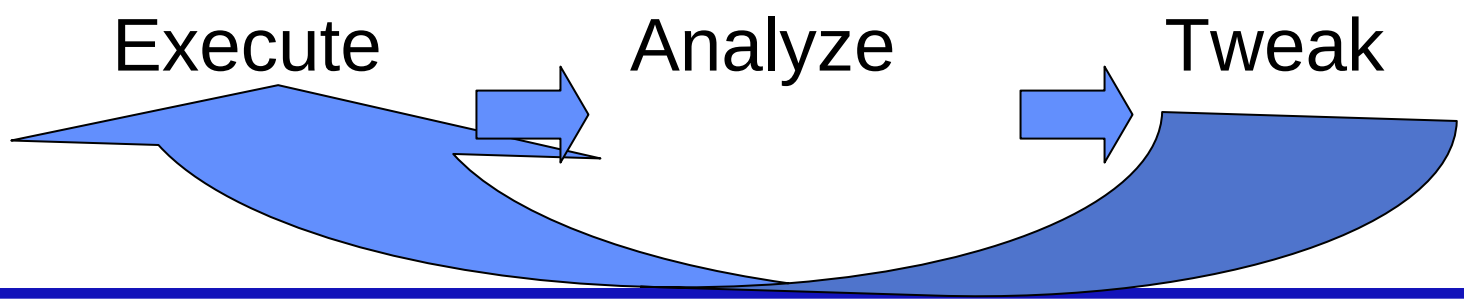
Oracle Statspack

```
Buffer Gets      Executions  Gets per Exec
-------------- ------------ --------------
301,995,711             781      386,678.2
SELECT RTRIM("A2"."CON_PRV_ID"),"A2"."PAY_…

 10,400,496           1,058        9,830.3
SELECT RTRIM("A3"."CON_PRV_ID"),"A2"."IND_…

    494,289               1      494,289.0
 DECLARE   fsdTable      VARCHAR2(30) := …
```

Analysis & Tuning

- Most of the analysis was performed by the performance team
- For one application, the development team was involved in the tuning (great benefit)

- Cycle



Results

- Routing Application
  - Created three performance configurations
    1. Tiny realtime transactions (low latency)
    2. Medium high volume transactions (high throughput)
    3. Low volume, very large transactions (stability)

Results

- Routing Application
  - In addition to the configuration changes to improve performance,
  - Significant database tuning was identified
  - Code efficiencies were analyzed
  - Network bottlenecks drove format changes

Results

- Processing application
  - Concurrency exposed software defects
  - Database tuning made drastic improvements
  - Did not achieve the full results desired

Some Lessons Learned

- Automating the execution process improved productivity, reduced timelines and allowed testing on shared hardware
- Codification of results was critical
- Cooperation of the development team is key
- Easily accessible, dedicated hardware would have significantly decreased tuning time