

# Foundations: Principles of Measurement

**Performance Engineering: Theory & Practice**

A series of horizontal lines in shades of green and white, extending from the right side of the text area towards the right edge of the slide.

# Seminar Topics

- **What kind of measurements**

- **Response time, latency, service time and queue time**
- **Throughput**
- **Utilization**
- **Queuing**

- **Exploratory Data Analysis Primer**

- **Measurement techniques**

- **Event-oriented measurements**
- **Sampling**

- **What to measure**

- **Hardware utilization (CPU, Memory, Disk, Network)**
- **Processes and other OS entities**
- **Applications: web servers, DBMS**

- **Measurement Tools**

- **System and Application Monitors**
- **Application profilers**
- **Load testing and benchmarking**

# Instrumentation

- **Many types of instrumentation built into hardware & software**
  - **Computer measurement is the original “Big Data” app**
  - **Instrumentation embedded in the hardware**
  - **OS and other system software**
  - **Application-level monitoring**
- **Understanding the measurements**
  - **Which measurements are important**
  - **How measurements are derived**
  - **How measurements can be interpreted**

# Instrumentation

- **Computer measurement is the original “Big Data” app**
  - e.g.,
    - **computer capacity planning using R**
    - **chargeback for pay-for-play cloud services**
- **Measurements are the pre-requisite for management, planning, feedback and control systems**
- **Many **limitations** on every known platform**
  - **key measurements may simply be unavailable or inadequate**
  - **subject to excessive overhead/volume**
  - **can be poorly designed or easily misunderstood**



# Instrumentation

- **e.g., limitations in Windows**
  - **key measurements may simply be unavailable or inadequate**
    - **HTTP Server response times for service level reporting**
    - **IO rate and response time/File**
    - **delays due to .NET Framework Garbage Collection**
  - **subject to excessive overhead/volume**
    - **applies to almost every high volume event stream that is not subject to a robust filtering mechanism**
  - **can be poorly designed or easily misunderstood**
    - **response time/latency distributions; reporting only overall interval Mean or Maximum values instead**

# Instrumentation

- **These problems are not limited to Windows!**
  - **Vendors of closed, proprietary systems are sometimes reluctant to expose measurements they consider too revealing**
    - e.g., VMware does not disclose much about guest machine dispatching delays
    - Customers running Apple iOS do not have easy access to performance monitoring tools that provide basic resource usage metrics
      - CPU
      - Memory/Paging
      - Network traffic and latency
  - **Technology often races ahead, while measurement data to manage it lags**
    - e.g., Containerization, serverless ([link](#))

# Instrumentation

- **Measurement sources**

- **Continuous: (relatively low overhead)**

- **Performance monitors: applications specifically designed for trouble-shooting performance problems**
    - **Resource-based cost accounting (on premises and cloud-based)**
    - **End-to-End response time reporting across multiple tiers**
    - **Measurements often directed at providing feedback on automation & control mechanisms (e.g., CPU priority queuing, concurrency and locking)**

- **On demand probes: (high potential overhead)**

- **Profilers**
    - **Event-oriented tracing tools, mainly designed for problem diagnosis, can often be re-purposed for use in a computer performance context**

# Instrumentation

- **Validity**
  - **Understanding & interpreting measurements correctly**
- **Accuracy and Reliability**
  - **“Capture ratio” problems**
  - **Sampling error**
  - **Representativeness**
  - **Measurement interference**
- **Cost trade-offs**
  - **Overhead vs. accuracy**
- **Awash in measurement data, but also many perceptible gaps**
  - **Tendency to instrument aspects that are interesting mainly to the Developer**

# Why Measure?

- To ensure that performance **management objectives** are being met
  - Customer Satisfaction
  - Service Level objectives
  - Quality of Service (QoS)
- The measurements you gather feed a variety of important IT management processes
  - Diagnose performance problems
  - Management reporting
  - Pro-active capacity planning
  - Chargeback

# What should you measure?

- **Requirements differ for**
  - **Client machines & interactive applications**
    - **Foreground**
      - **Responsiveness of the User interface**
    - **Background**
      - **Long running tasks that can be performed asynchronously**
  - **Multi-User Servers & services (Web, Data, etc.)**
    - **“Mission-critical” production**
    - **Transaction processing workloads have inherent parallelism**
    - **Concurrency, resource-sharing, and Locking**

# What should you measure?

- **Client machines & interactive applications**
  - **Why is this application running slowly?**
  - **Compared to what?**
- **Foreground/Background**
  - **What is blocking the User interface?**
  - **Can we reproduce this behavior in the lab?**
- **e.g., AJAX with JavaScript code running foreground and making asynchronous calls to web services in the background**
- **Because impact is limited to one (possibly generic) customer, running one-off performance diagnostic tool is usually acceptable**

# Top command in Linux & Unix

```

Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
gary@gary-VirtualBox: ~
%Cpu(s): 13.0 us, 0.3 sy, 0.0 ni, 86.0 id, 0.3 wa, 0.0 hi, 0.3 si, 0.0 st
KiB Mem: 4047424 total, 982932 used, 3064492 free, 32196 buffers
KiB Swap: 4192252 total, 0 used, 4192252 free. 340968 cached Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+  COMMAND
 1405 gary       20   0 1196780 190160 66260 S   9.3   4.7   0:15.24 compiz
   931 root       20   0  353012  92488 24416 S   3.3   2.3   0:03.74 Xorg
     3 root       20   0     0     0     0 S   0.3   0.0   0:00.02 ksoftirqd/0
    77 root       20   0     0     0     0 S   0.3   0.0   0:00.36 kworker/u2:4
   700 root       20   0  239924   1720   1416 S   0.3   0.0   0:00.03 VBoxService
  1062 gary       20   0  116104   3088   2720 S   0.3   0.1   0:00.16 VBoxClient
  1184 gary       20   0  551612  30100 24308 S   0.3   0.7   0:00.30 unity-panel-+
  1702 gary       20   0  621624  34424 25296 S   0.3   0.9   0:00.43 gnome-termin+
  1772 gary       20   0   29220   3128   2628 R   0.3   0.1   0:00.12 top
     1 root       20   0  116928   5204   3724 S   0.0   0.1   0:00.96 systemd
     2 root       20   0     0     0     0 S   0.0   0.0   0:00.00 kthreadd
     4 root       20   0     0     0     0 S   0.0   0.0   0:00.00 kworker/0:0
     5 root        0 -20     0     0     0 S   0.0   0.0   0:00.00 kworker/0:0H
     6 root       20   0     0     0     0 S   0.0   0.0   0:00.00 kworker/u2:0
     7 root       20   0     0     0     0 S   0.0   0.0   0:00.09 rcu_sched
     8 root       20   0     0     0     0 S   0.0   0.0   0:00.00 rcu_bh
     9 root       20   0     0     0     0 S   0.0   0.0   0:00.07 rcuos/0
    10 root       20   0     0     0     0 S   0.0   0.0   0:00.00 rcuob/0
    11 root       rt   0     0     0     0 S   0.0   0.0   0:00.00 migration/0
    12 root       rt   0     0     0     0 S   0.0   0.0   0:00.00 watchdog/0
    13 root        0 -20     0     0     0 S   0.0   0.0   0:00.00 khelper
    14 root       20   0     0     0     0 S   0.0   0.0   0:00.00 kdevtmpfs
    15 root        0 -20     0     0     0 S   0.0   0.0   0:00.00 netns
    16 root        0 -20     0     0     0 S   0.0   0.0   0:00.00 perf

[1]+  Stopped                  top
gary@gary-VirtualBox:~$

```



# Taskman in Windows

Task Manager

File Options View

Processes Performance App history Startup Users Details Services

Name	Status	7% CPU	38% Memory	0% Disk	0% Network	1% GPU	GPU Engine
> Microsoft Edge (20)	🟢	1.0%	2,356.0 MB	0 MB/s	0 Mbps	0.5%	GPU 1 - 3D
> SQL Server Windows NT - 64 Bit		0%	1,488.4 MB	0 MB/s	0 Mbps	0%	
> Intel(R) Technology Access - Service		0%	331.8 MB	0 MB/s	0 Mbps	0%	
> Photos (2)	🟢	0%	175.3 MB	0 MB/s	0 Mbps	0%	GPU 1 - 3D
> Antimalware Service Executable		0.2%	152.2 MB	0 MB/s	0 Mbps	0%	
> Microsoft Outlook (32 bit) (4)		0.2%	125.4 MB	0 MB/s	0.1 Mbps	0%	
> Microsoft PowerPoint (32 bit) (3)		0%	115.3 MB	0 MB/s	0 Mbps	0%	
> Microsoft Visual Studio 2017 (32 bit) (11)		0%	114.1 MB	0 MB/s	0 Mbps	0%	GPU 1 - 3D
> Cortana (3)	🟢	0%	101.6 MB	0 MB/s	0 Mbps	0%	GPU 1 - 3D
Desktop Window Manager		0.4%	87.1 MB	0 MB/s	0 Mbps	0%	
> SQLDMVCollectionService.exe		0%	54.3 MB	0 MB/s	0 Mbps	0%	
> Windows Explorer		0.1%	49.5 MB	0 MB/s	0 Mbps	0%	
> Service Host: Diagnostic Policy Service		0%	46.0 MB	0 MB/s	0 Mbps	0%	
> Microsoft Windows Search Indexer		0%	40.8 MB	0 MB/s	0 Mbps	0%	
> Windows Shell Experience Host	🟢	0%	36.2 MB	0 MB/s	0 Mbps	0%	GPU 1 - 3D
> Task Manager		0.7%	34.4 MB	0 MB/s	0 Mbps	0%	
Windows Defender SmartScreen		0%	29.2 MB	0 MB/s	0 Mbps	0%	
Lenovo.Modern.ImController.PluginHost (32 ...)		0%	23.5 MB	0 MB/s	0 Mbps	0%	
> Sql Server Telemetry Client		0%	21.6 MB	0 MB/s	0 Mbps	0%	
> Performance Counter Collection Service		0%	21.2 MB	0 MB/s	0 Mbps	0%	

⬆️ Fewer details

End task

# What should you measure?

- **Multi-User Servers & services (Web, Data, etc.)**
  - **DevOps:**
    - **Connected**
    - **High Availability (7 x 24 x 365)**
  - **Require continuous performance monitoring:**
    - **Management by Objective**
    - **Management by Exception**
    - **Anomaly detection in large server farms that can automatically trigger detailed probes**
      - **Clustered servers**
      - **Interactions across multiple tiers**

# What should you measure?

- **DevOps: continuous performance monitoring of production systems**
  - **Overhead considerations:**
    - **Can we gather enough information on a continuous basis**
      - **to allow enough visibility into the problem**
      - **to determine the next steps in diagnosing it**
      - **without becoming overwhelmed with data that must be reduced and analyzed?**
    - **e.g., Dev requests an instrumented profile at the application level that records every Method entry and exit**

# What should you measure?

- **DevOps: continuous performance monitoring of production systems**
  1. **Real-Time** data collection and reporting
    - Service level reporting
    - Alerts (see, e.g., [An Introduction to DigitalOcean Monitoring](#))
  2. Data from the **Recent Past** available for post-mortems
  3. Repository of **Historical** data for analysis and trending
- **From reactive fire-fighting mode to a pro-active custodial mode**
  - **Quality Control/Change Control gatekeeper procedures**

# What should you measure?

- **DevOps: continuous performance monitoring of production systems**
- **Management by Objective**
  - **Service Level Reporting**
    - **Customer load/throughput**
    - **Response time**
      - **report on distributions where possible, not just averages**
      - **dynamic response time buckets ???**
  - **Compliance with Service Level Agreements (and contracts) which may entail performance guarantees**

# What should you measure?

- **DevOps: continuous performance monitoring of production systems**
  - **Management by Exception**
    - **Real-time alerting: reporting Dashboards**
    - **Alerts that fire too frequently lose their potency**
      - **Trigger/Release and Re-arm dampening mechanisms**
    - **Cascading Alerts are problematic, too!**
      - **They can overwhelm the reporting system**
      - **Quantity of data can also obscure the **root cause****
  - **Historically, rule-based, “Expert” diagnostic systems were very brittle**
    - **Very few reliable Rules of Thumb (ROT) exist!**
    - **see <https://performancebydesign.blogspot.com/2011/01/performance-rules.html>**

# What should you measure?

- **DevOps: continuous performance monitoring of production systems**
  - **Anomaly detection in large scale server farms**
    - **Clustered servers**
    - **Interactions across multiple tiers (synchronized clocks?)**
- **Since there are very few reliable performance Rules, apply Statistical Quality Control techniques instead**
  - **e.g.,**
    - **set a critical Alert threshold at the 99<sup>th</sup> percentile**
    - **set a severe Alert threshold at the 98<sup>th</sup> percentile**
    - **set an informational Alert threshold at the 95<sup>th</sup> percentile**

# What should you measure?

- **Response time**
  - **also known as Latency**
  - **Service time**
    - **Note: load-dependent servers**
    - **Note: cache cold start vs. warm start**
  - **Queue time**
- **Throughput**
- **Utilization**
  - **Note: relationship of queuing to utilization**
- **Queue length**



# Probability and Statistics Primer

- **Performance analysis is one of the original “Big Data” applications**
  - **Because analysts are flooded with job accounting & performance measurement data!**
  - **Large amounts data requires statistical analysis!**
- **But what sort of statistical analysis is required?**
  - **Not a textbook case of hypothesis testing (e.g., t-Tests for statistical significance, etc.)**
- **Exploratory Data Analysis**

# Exploratory Data Analysis

“At the roots of all our judgments there are a certain number of essential ideas which dominate all our intellectual life; they are what philosophers since Aristotle have called the categories of the understanding: ideas of **time**, space, class, **number**, **cause**, substance, personality, etc. They correspond to the most universal properties of things.”

- Emile Durkheim, *The Elementary Forms of the Religious Life*, 1912.
- **Clocks and Time measurements: see Gunther, ch. 1.**
- see also **[“High Resolution Clocks and Timers for Performance Measurement in Windows”](#)**

# Exploratory Data Analysis

- A term introduced by the Princeton mathematician **John Tukey** to distinguish descriptive statistics from the better known forms associated with hypothesis testing (R. A. Fisher, et. al.)
  - Prior to the widespread use of computers to analyze statistical data, Tukey published two textbooks that emphasized **visual explanation**
    - e.g., Tukey invented the Box-and-Whisker plot
    - Also known for saying, “An approximate answer to the right problem is worth a good deal more than an exact answer to an approximate problem.”
- The best known disciple of Tukey today is Edward R. Tufte

# Exploratory Data Analysis

- **Edward R. Tufte**
  - self-publishes 4 remarkable books on visualizing quantitative data:
    - *The Visual Display of Quantitative Information, 1983.*
    - *Visual Explanations, 1997.*
  - Teaches a popular, one-day seminar of the subject
  - Popularized **sparklines**
  - Extremely critical of PowerPoint and its use of visual “chart-junk”

# Descriptive statistics

- **Univariate**
  - **frequency distribution, histogram, probability density function (pdf)**
  - **central tendency: mean, median, mode**
  - **dispersion: range, variance, standard deviation (s.d.)**
- **Bivariate**
  - **x-y scatterplot; regression; correlation**
- **Multivariate**
  - **step-wise multiple regression, principal component analysis, k-means cluster analysis**

**Anomaly detection example:**

**How do measurements taken during a specific interval when there was a performance problem compare to measurements that are “typical” for that machine?**

Data Mining/Analytics
localhost/DemandTechPortal/DataMining.aspx?Command=NavigateFromBoxPlot&Machine=REDHOOK&PivotDate=8/5/2018&ReportHour=14

### Data Mining and Analytics

Machine: REDHOOK

Pivoting on Date: 8/5/2018 beginning at Hour 14


**Select Machine:**

Machine: REDHOOK

Group: PrimeWatch

or

Search:



**Pivot Date:** 8/5/2018  
Sunday

**End Date:** 8/12/2018

**Start Date:** 6/28/2018

**# of Days:** 45

**Pick an Analysis End Date:**

August 2018						
Su	Mo	Tu	We	Th	Fr	Sa
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

**Pick a Start Date:**

June 2018						
Su	Mo	Tu	We	Th	Fr	Sa
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

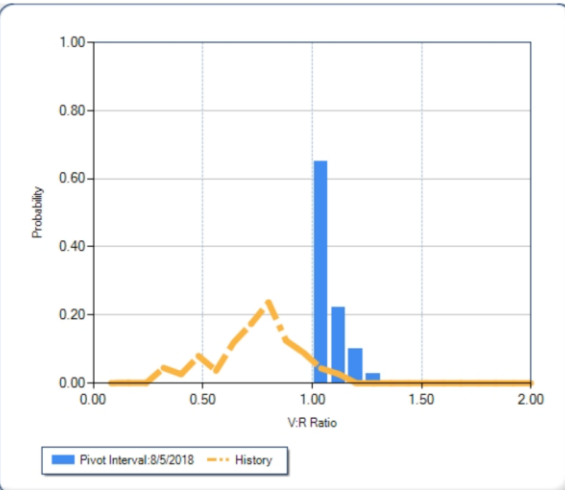
**Select Analysis Time Window:**  
Hour: 14 Duration in Minutes: 180

**Filter Results by Day of the Week:**  
Include all Days of the Week

Export

**Select Counter set from Report template:**  
ServerHealth

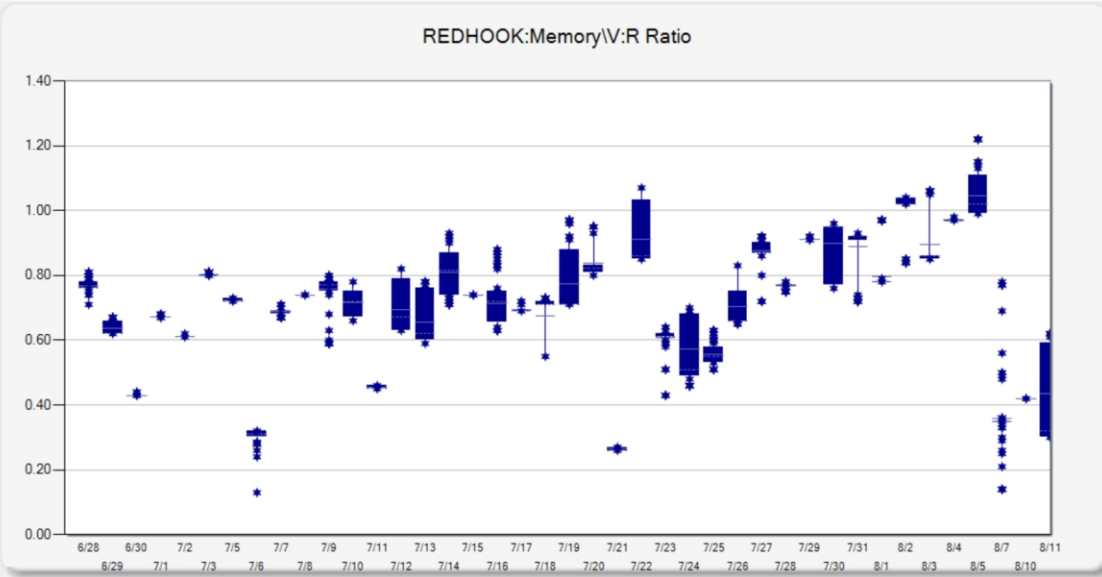
**Select counter from Report definition:**  
Memory\V:R Ratio Build



Summary Statistics			
	Pivot Data	History Data	Delta
N	180	6991	
Mean	1.046	0.700	1.892
s.d.	0.063	0.183	
Minimum	0.990	0.130	0.860
5th percentile	0.990	0.340	0.650
25th percentile	0.990	0.610	0.380
Median	1.020	0.720	0.300
75th percentile	1.110	0.810	0.300
95th percentile	1.140	0.970	0.170
Maximum	1.220	1.070	0.150

**Active Filter:** None

**Calculate difference using:**  
 Normalized Distance (s.d.)  
 Absolute distance

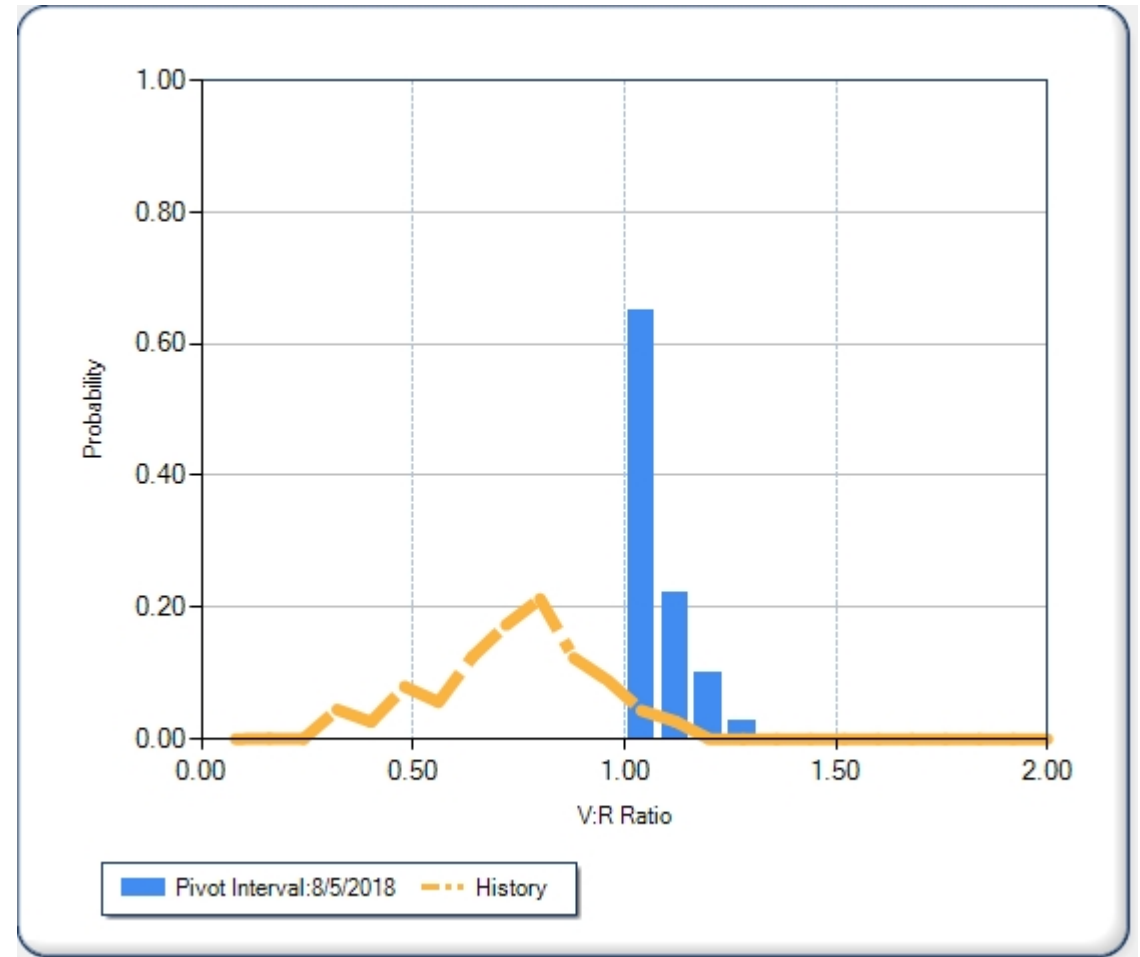


# Univariate statistics

- Frequency distribution
- Histogram
  - summarized using bins

$$\#bins = (\max - \min) / width$$

- Probability density function
  - $\Sigma$  area under the curve = 1

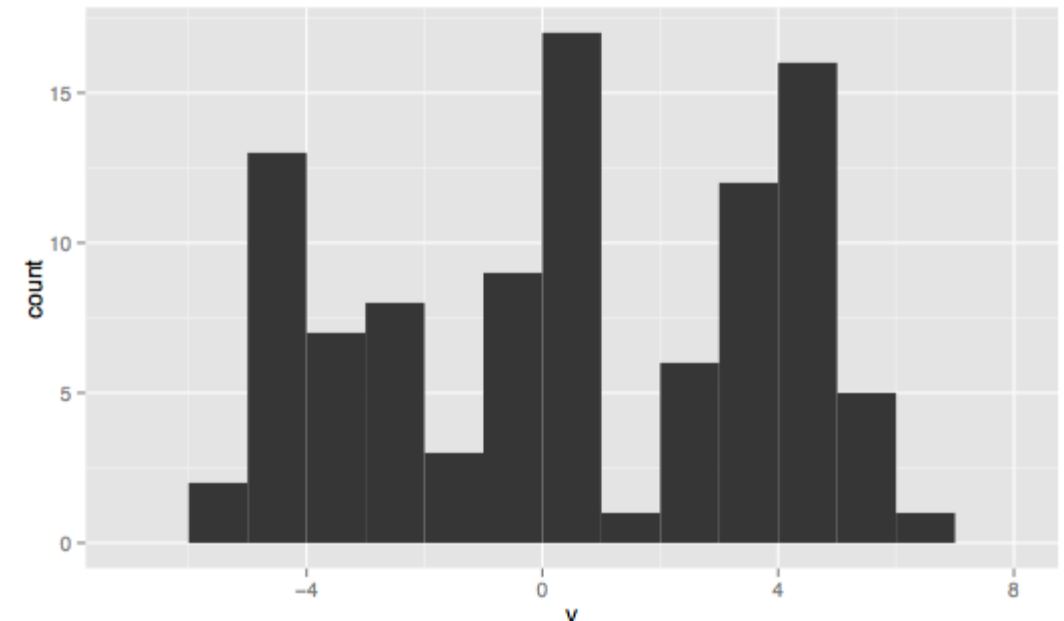


# Univariate statistics

- **central tendency:**
  - **mean**
    - “best” likelihood estimator is not guaranteed to be a good estimator

$$\text{mean} = \frac{\sum x}{n}$$

- can be calculated continuously
  - **median, mode**
- **dispersion:**
    - **min, max, range, quartiles**
    - **variance, standard deviation (s.d.)**



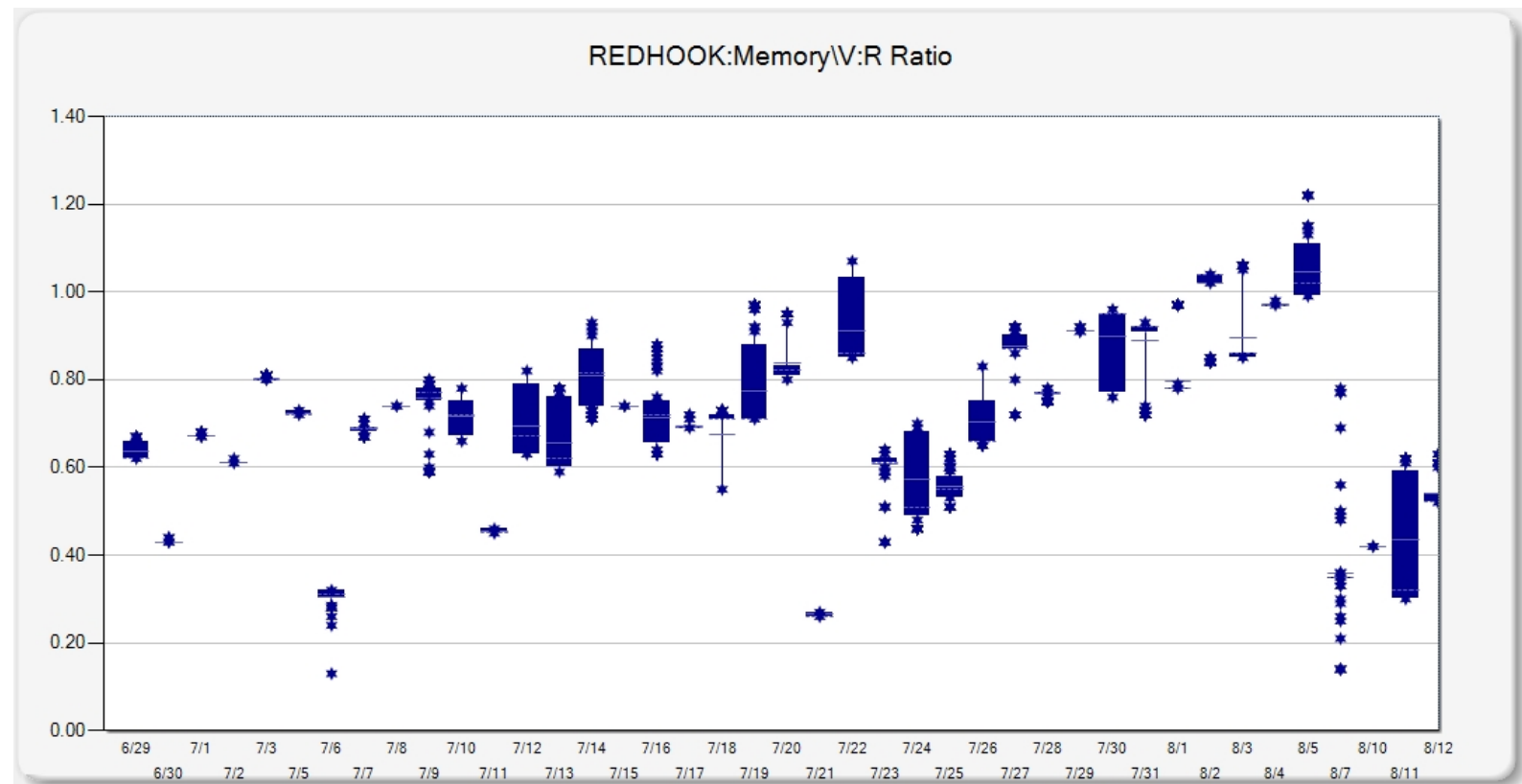


# Univariate statistics

- **dispersion:**
  - **min, max, range, quartiles**
  - **variance, s.d.**
  
- **box plot**

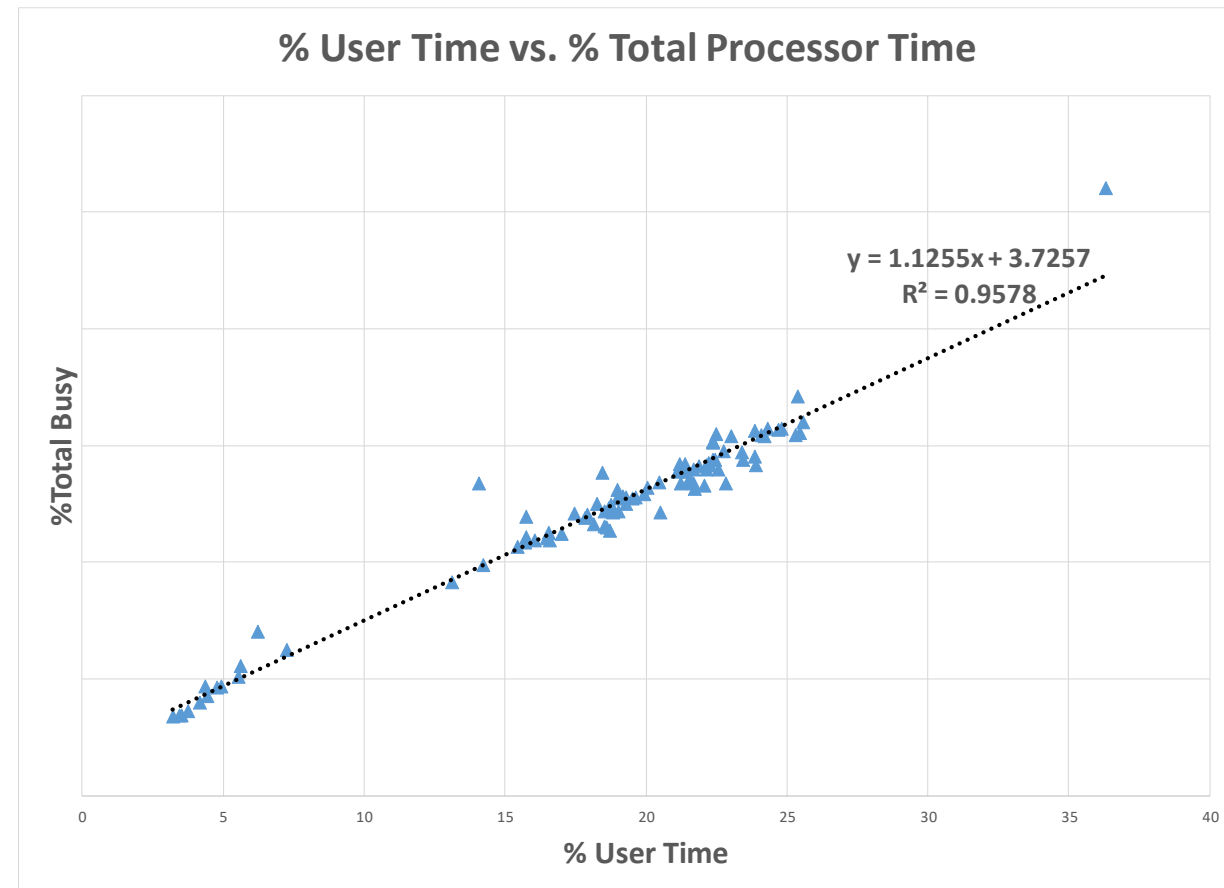
$$s.d. = \sqrt{\left( \frac{(\sum x^2 - \sum x)^2}{N - 1} \right)}$$

**How useful is the s.d.  
when the underlying  
distribution is not  
normal?**

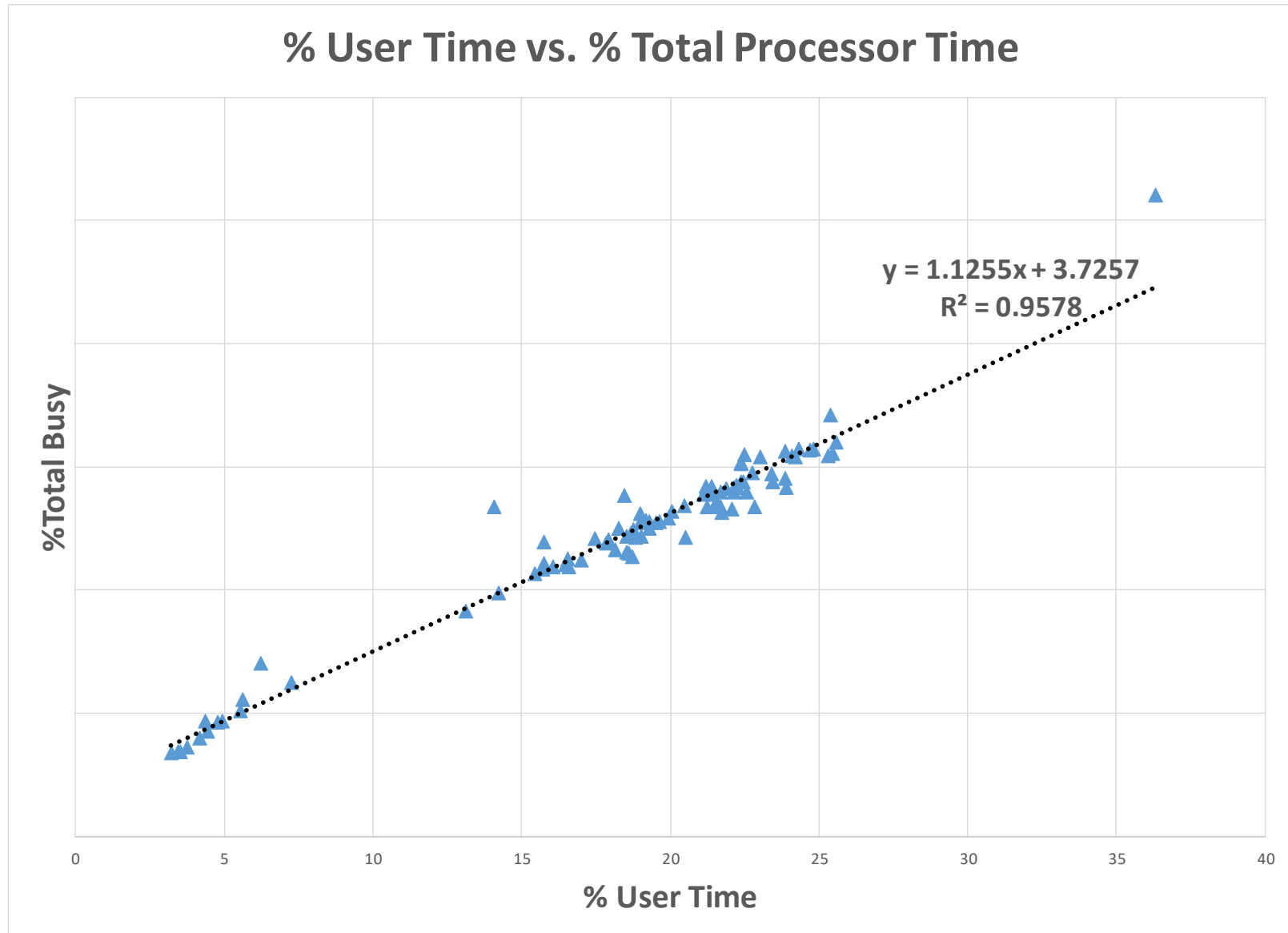


# Bivariate statistics

- **x-y scatterplot**
- **linear regression (least squares)**
  - **r**, correlation coefficient
  - **r<sup>2</sup>** : interpreted as the *proportion* of the variability in **x** explained by the variability of **y**
- **nonlinear regression**
  - **curve-fitting**



# Linear regression (ordinary Least Squares)



**Least Squared method minimizes the vertical distance between each point and the fitted line.**

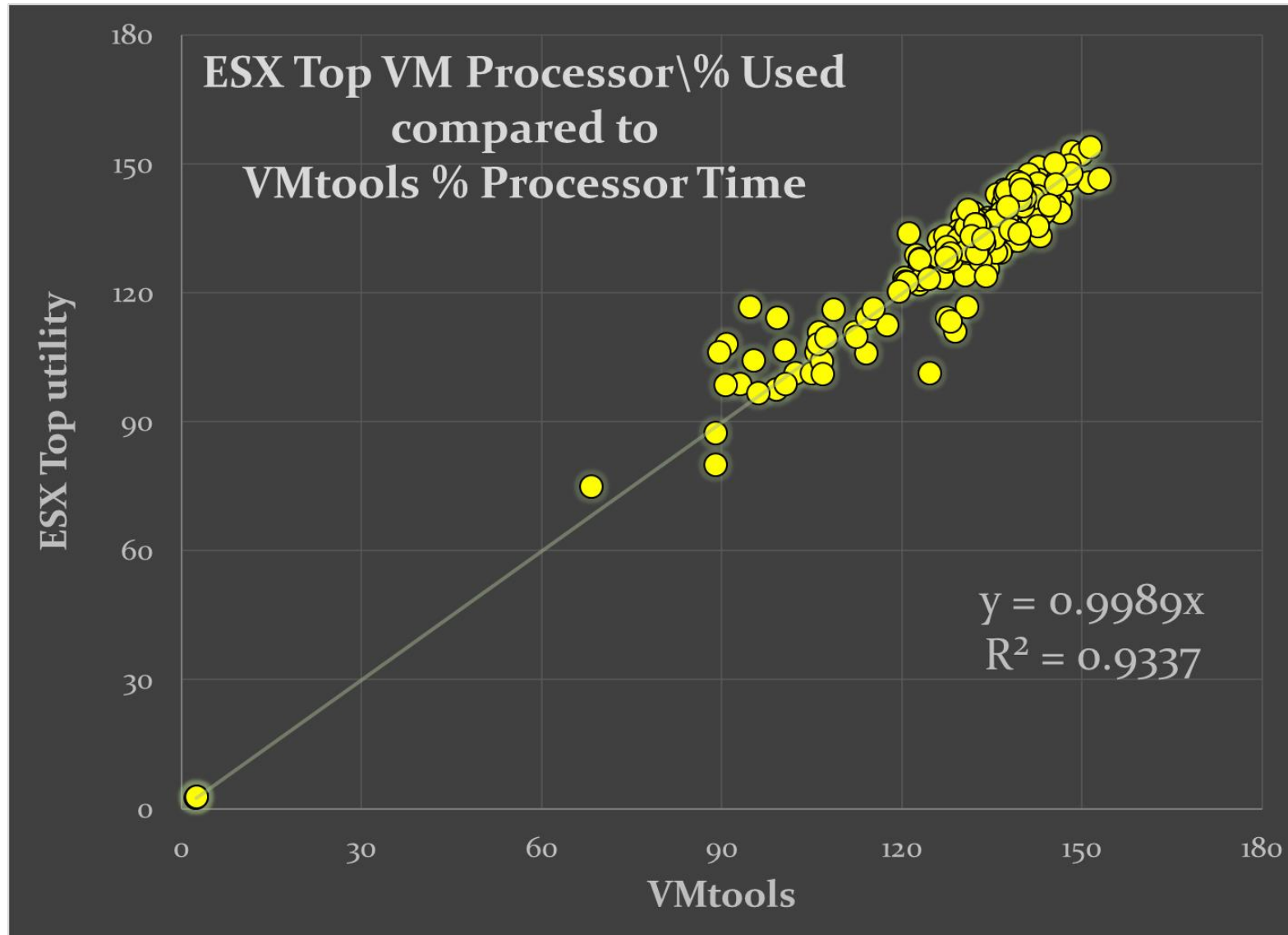
**In Excel:**

**Add a linear Trendline to an x-y scatterplot**

**Optionally, request statistics**

**Remember that correlation is not causation!**

# Linear regression (ordinary Least Squares)



**Least Squared method minimizes the vertical distance between each point and the fitted line.**

**In Excel:**

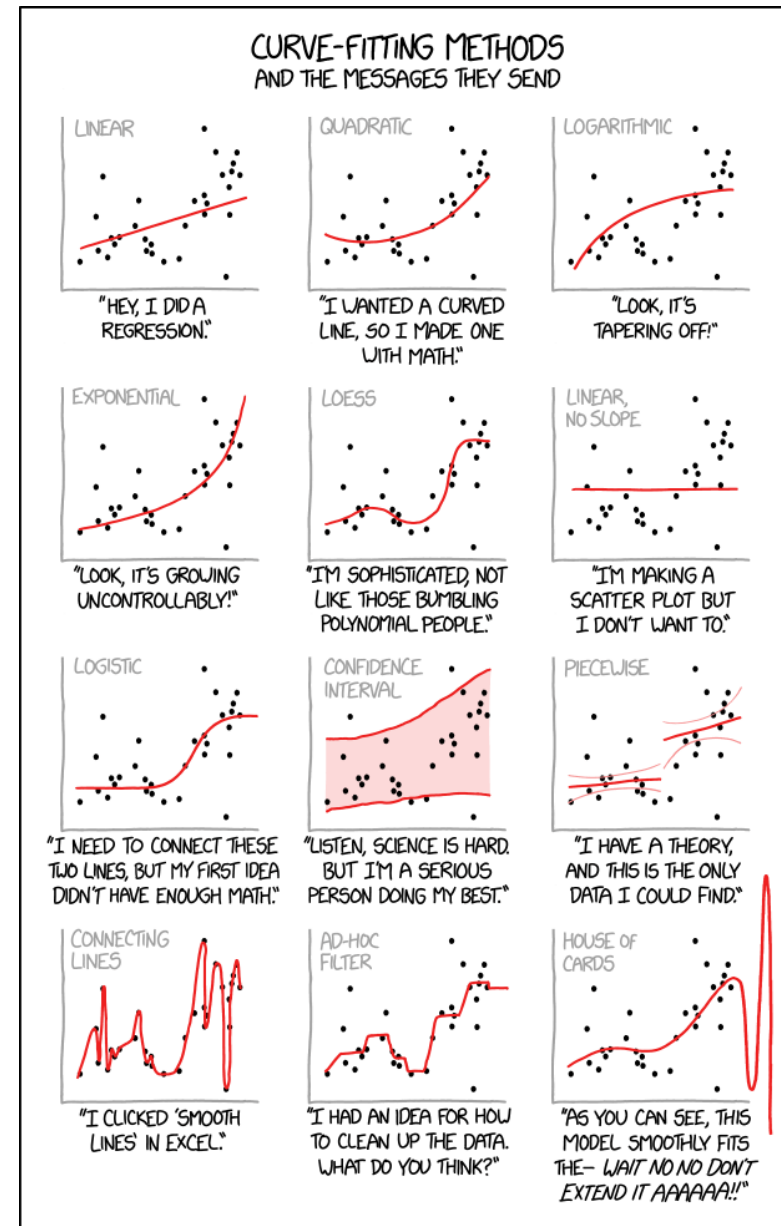
**Add a linear Trendline to an x-y scatterplot**

**Optionally, request statistics**

## • Nonlinear regression and curve-fitting

- Queuing theory predicts exponential relationships between measures of load & response times (or throughput)
- But fitting measurement points to an exponential equation requires a wide distribution of x,y observations
- when the distributions are relatively narrow, the regression line is more apt to be flat (or random-looking)

\* <https://xkcd.com/2048/>

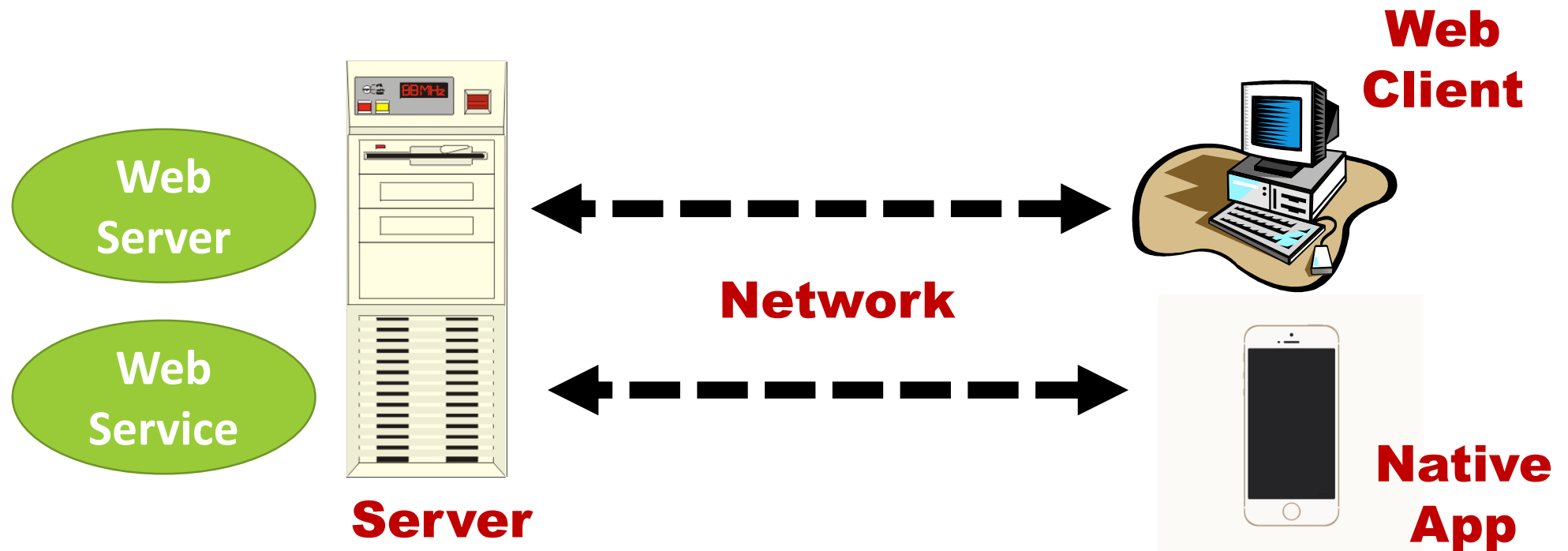


# Multivariate statistics

- **Anomaly detection**
- **Data Mining and ML techniques**
  - **Step-wise multiple regression**
  - **Principal component analysis**
  - **k-means cluster analysis**
    - **partitions  $n$  observations into  $k$  clusters in a multi-dimensional space, minimizing the “distance” between observations in a cluster and maximizing the “distance” between clusters**
- **Caution:**
  - **curve-fitting, instead of modeling**
  - **e.g., step-wise multiple regression with enough dependent variables will yield correlations that pass significance tests (i.e.,  $p < .05$ ) based on probability alone**
  -

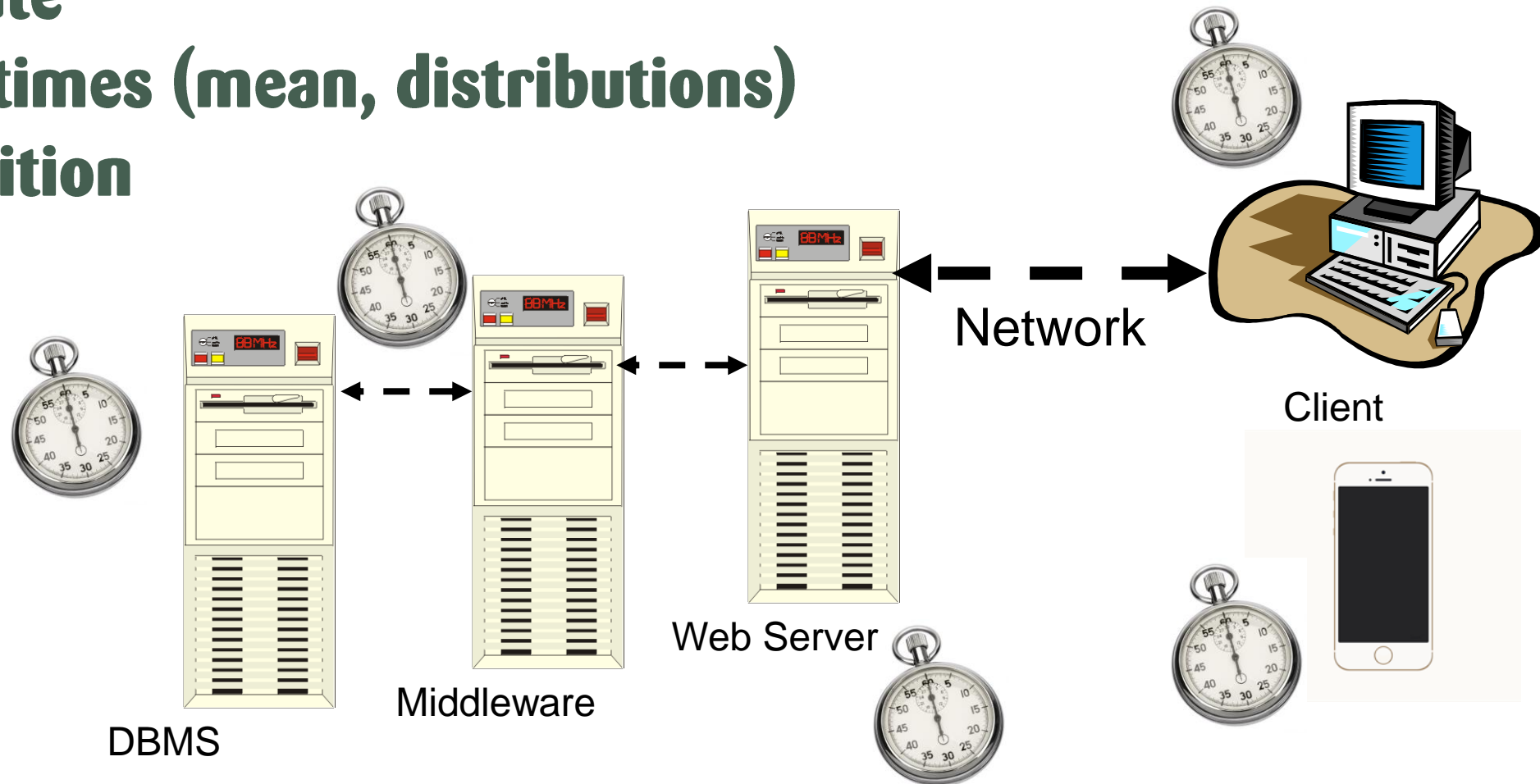
# Measuring Response Time

- **End-to-End (ETE) Response times (Client-Server)**
  - Request rate
  - Response times (mean, distributions)



# Measuring Response Time

- **End-to-End (ETE) Response times (Client-Server)**
  - Request rate
  - Response times (mean, distributions)
  - Decomposition



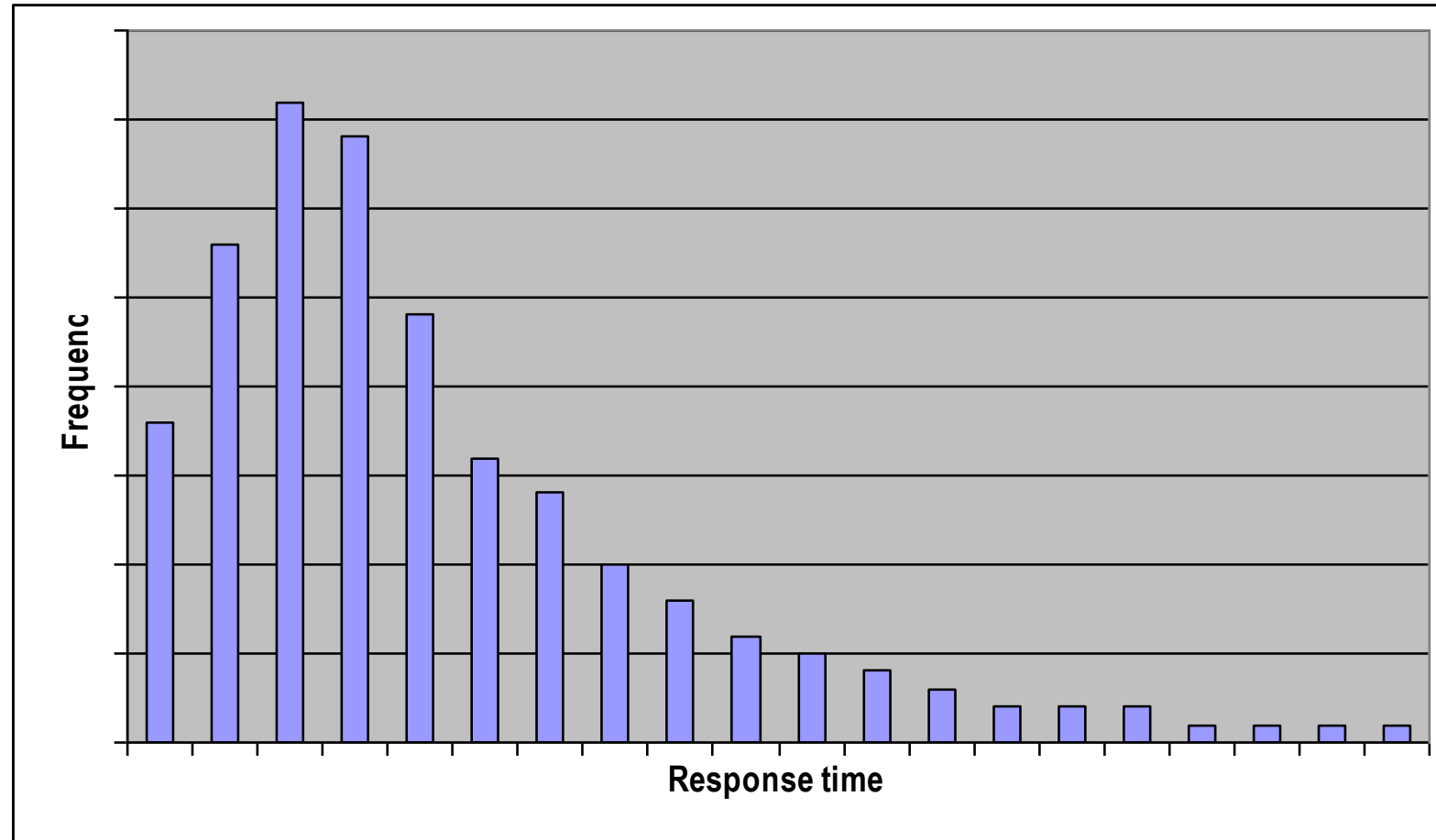


# Why Measure Response Time

- **Best way to encapsulate the “User Experience”**
- **What is the *economic value* of good response time?**
  - **Customer satisfaction also correlated with the *consistency* of response time**
    - **Corollary:** report response time distributions, not just averages
    - **Corollary:** provide feedback to the customer to manage service level expectations
      - Status bars, etc.
  - **Productivity (mainly concerned with internal customers)**
    - **Tasks completed/sec**
    - **Error rates**
    - **Can the system be too fast?**

# Response time distributions

- Response times are usually right-hand exponential
- Poisson (i.e., exponential) distributions

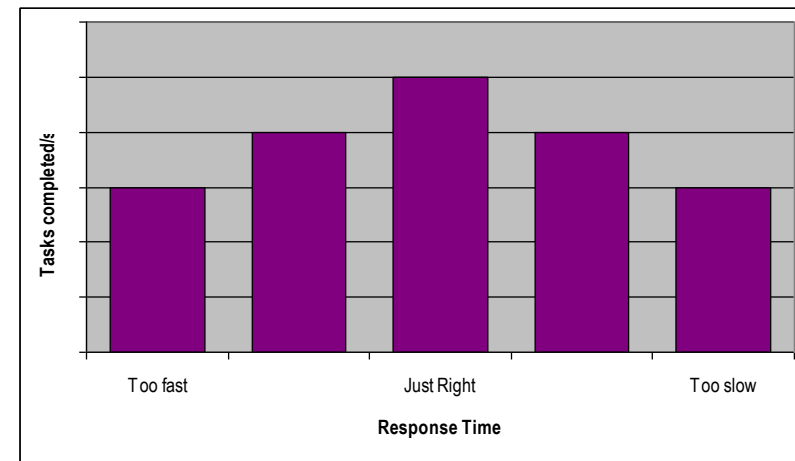


# The *economic value* of sub-second response time

- Much publicized IBM study (Doherty and Thadani, 1981) showed **TSO *trivial transaction*** rates (for a source code browser Editor) increased significantly when the system delivered *sub-second* response time
- “Dumb” Terminal displays were channel attached
  - Productivity improvements were inferred using a psychological model of attention, short-term memory buffer overflow, etc.
  - Better response times led to shorter *Think Times* between transactions
  - In fact, 90% of all **Think Time** events were between 0.2 - 0.5 seconds
  - 0.2 seconds = human *reaction* time

# The *economic value* of subsecond response time

- Thadani's inference that productivity improved is not justified!
  - Users *adapt* their behavior to the system.
  - When the system is slower, they work smarter
    - e.g., more likely to use a SEARCH command instead of Page Down command when the system is slower
  - Nevertheless, customers prefer faster, more responsive systems
  - But, systems can also be *too fast*
  - See Barber & Lucas (CACM, 1983):

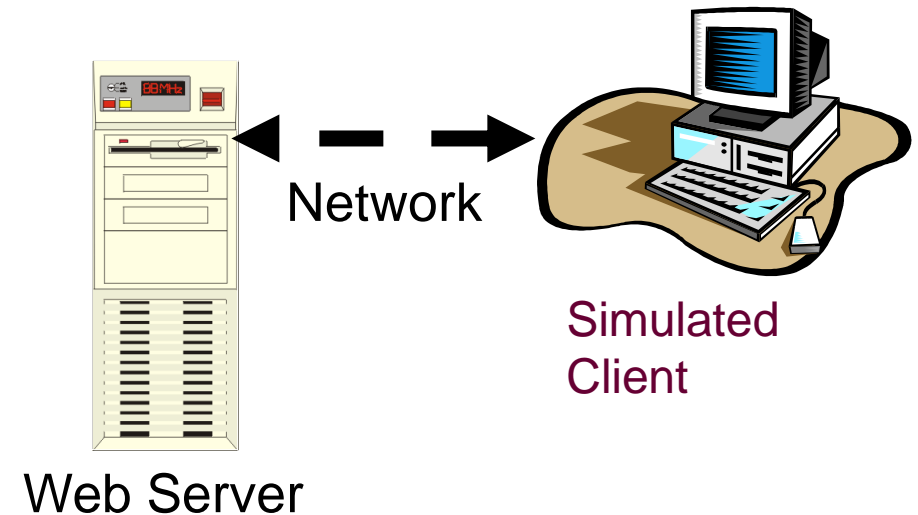


# Why Measure Response Time

- What is the **unit of work**?
  - e.g.,
    - Response messages to HTTP Get Requests that trigger additional Get Requests
    - Two approaches:
      - Measure the time to the first response event
      - Measure the total time to complete the Page build
    - When is the application functional?
  - Instrumented applications
  - Properly instrumented of **Key Volume Indicators** can also feed business element-based forecasting directly

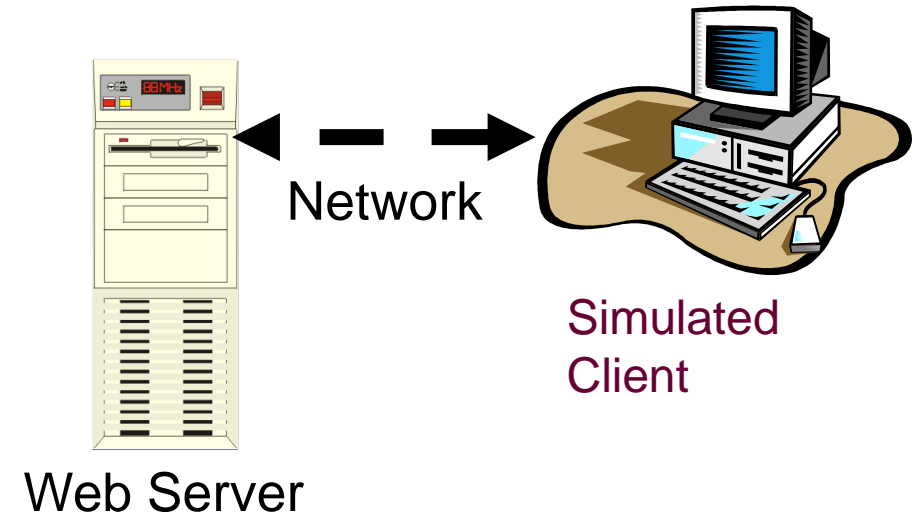
# Measuring Response Time

- Time the requests of **simulated clients**
  - Periodic execution of representative Scripts that execute on the client
    - Often a service bureau does this
    - Geographical distribution matters
    - \$\$\$
  - Requests must be repeatable
  - Periodic sampling
  - be careful about generating too much additional load during peak periods



# Measuring Response Time

- **Time the requests of simulated clients**
  - **How representative is the sample of timed requests vs. overall workload?**
  - **Complex interactions or update transactions may be under-sampled**
  - **Simulated Client's network access vs. customers' network access**
- **Real User Measurements (RUM) are an alternative today**



# Sampling issues

- **Probability and Statistics**

- **Measure a selected subset of a *population* in order to try and understand the underlying population**
  - **Random sampling**
  - **Systematic sampling**
  - **Stratified sampling**
  - **Convenience sampling**
- **Sample size (based on acceptable margin of error)**
- **See: Cochran, W. G. (1977). *Sampling techniques* (3rd ed.). New York: John Wiley & Sons.**



# Sampling

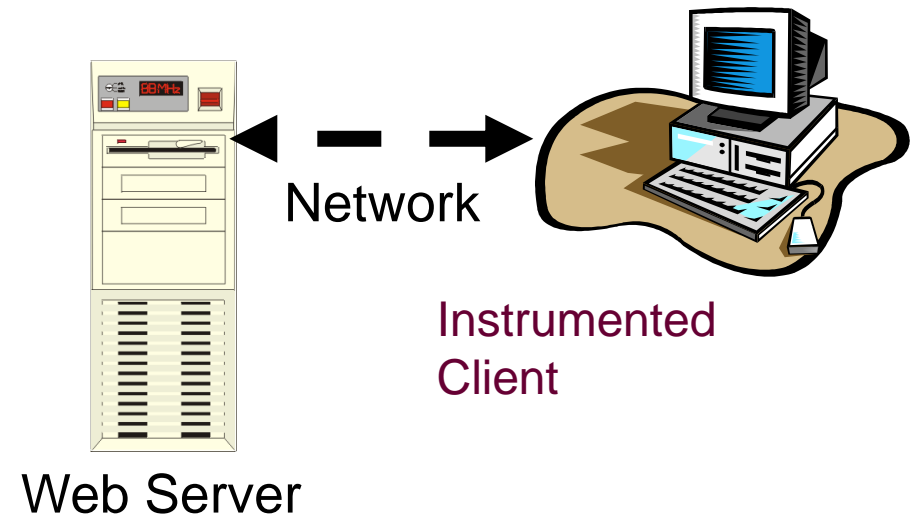
- **Sample size (based on acceptable margin of error)**

$$\# \text{ of observations} = t^2 * s^2 / d^2$$

- ***t*** = student's t value for tail of the distribution
  - e.g., 0.25 = 1.96
- ***s*** = estimated standard deviation of the population
- ***d*** = acceptable margin of error
  - e.g., 3%
- Assuming no systematic sources of sampling error, sample sizes of 300-3000 provide estimates of the population that are accurate within 1-3%.
- **Information Theory:** if the periodic frequency of a wave is ***x***, then you must sample at a minimum rate of ***2x***

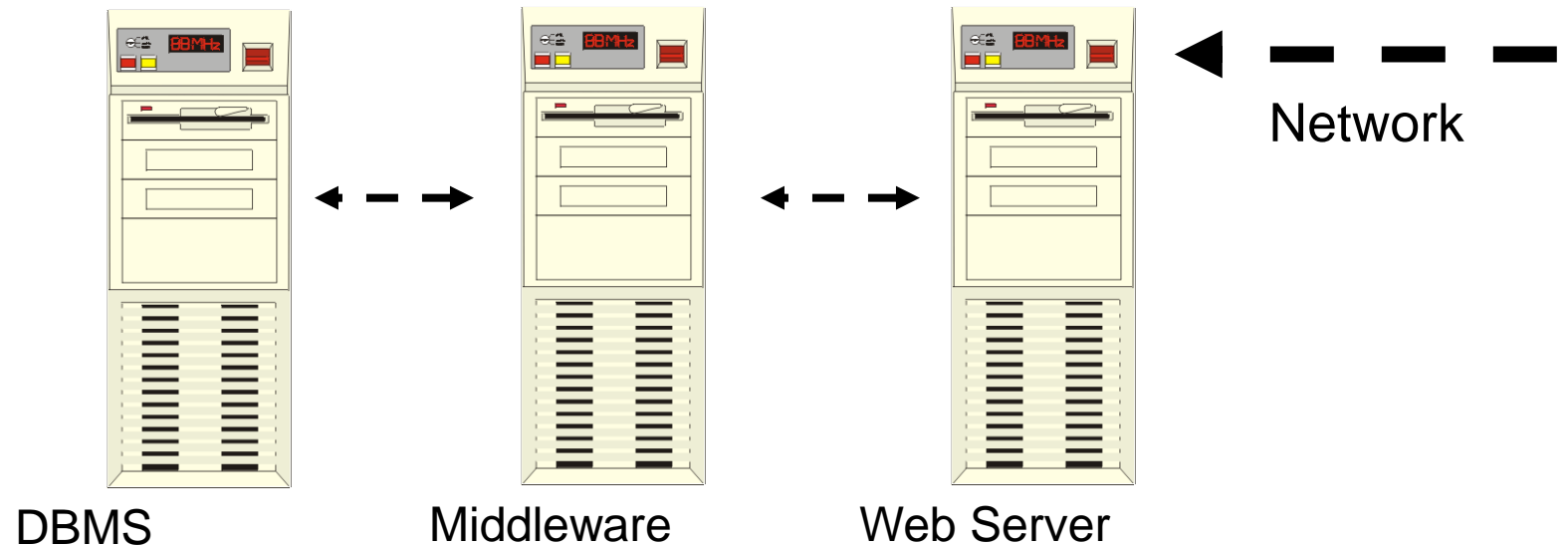
# Measuring Web app Response Times

- Instrument the clients (Real User Measurements)
  - e.g., Inject JavaScript code into the HTML page to access
  - Observe customers' behavior directly:
    - Internal customers
    - External customers can opt in (privacy concerns):
      - e.g., Google Analytics
    - Volume considerations: measurement data is returned using **Web beacons**
    - Representative data?



# Measuring Response Time

- **When are Server-side Response time measurements useful, necessary, adequate, or sufficient?**
- **ETE response time decomposition requires instrumenting each app layer/tier!**



# Measuring Response Time

- **When are Server-side Response time measurements useful, necessary, adequate or sufficient?**
  - **Obvious disadvantage: does not characterize the User Experience**
    - But, suppose network delay is minimal
      - e.g., “dumb” terminals in an IBM mainframe hung from a channel-attached communications controller
    - But, suppose **network delay is outside your span of control**
      - e.g., External customers that must use the public Internet to access your application (e.g., e-commerce applications)
      - Do you want to be graded on something you cannot control?
    - But, suppose **your application uses a web service** to communicate across the public Internet to other computers

# Measuring Response Time

- **When are Server-side Response time measurements useful, necessary, adequate or sufficient?**
- **We will pursue this topic *after* we understand better how web pages are rendered in the browser.**

# Measuring Response Time

- **Desktop Application Response time (Client only)**
  - **Key ingredient in usability studies**
    - **Novice Users vs. Power Users**
  - **Determining the boundaries of units of highly interactive work is often difficult**
    - **e.g., Keyboard events**
      - **Keydown event is transformed into a repeat key after a delay**
    - **Mouse clicks, moves and hovers**
      - **The ergonomics of double-click**
    - **Animation (and interaction with graphics hardware)**
    - **Streaming (throughput-oriented)**



Client

# Throughput

- **Not all workloads of interest are response-oriented**

$$\text{Throughput} = \text{Tasks completed} / \text{Time}$$

- **Bytes/sec**
  - **Packets/sec**
  - **I/Os per second**
  - **Requests/sec**
  - **Pages printed/minute**
  - **Jobs/hour**
- **Requires instrumentation that counts arrivals ( $\lambda$ ) and/or completions (event-oriented)**
    - **Equilibrium assumption in Little's Law: arrivals = completions**
$$\# \text{ in system} = \lambda * \text{ResponseTime}$$
- **Utilization = Throughput / Capacity**

# Utilization

- **Measure utilization of the hardware**
  - **Processors**
  - **Memory**
  - **Disks**
  - **Network Interfaces**
- **Break down utilization by process**
  - **Response time decomposition, if  $\lambda$  is known**
    - **Utilization Law:  $u = \lambda * t_s$**
    - **“Capture ratio” problems occur when**

$$\sum u_{\text{process } n} < u_{\text{total}}$$



# Utilization

- **Measure both utilization and *queue length* for the resource**
  - **queuing models predict an *exponential* relationship, but real world measurements do not always corroborate**
- **Potential Complications**
  - **Queuing discipline at the resource (fair vs. unfair, priority)**
    - priority schemes can lead to starvation
  - **Load dependent servers (e.g., physical disks) where performance levels vary with the load on the resource**
  - **Cache effects (cold start vs. warm start)**
    - bi-modal service time distribution violates most queuing model assumptions
  - **internal or “hidden” queues**
  - **the number of queued requests may be severely limited (e.g., IP routers, a max worker threads setting in an application)**

# Utilization

- **Probability that a device is Busy ( $0 \leq u \leq 1$ )**
- **Measurement techniques**
  - **Sampling**
    - **Utilization = Device busy samples / total samples**
    - **Sampling frequency**
    - **Systematic under-sampling**
  - **Apply the Utilization Law:  $u = \lambda * t_s$** 
    - **Measure  $\lambda$**
    - **Measure Service Time ( $t_s$ )**
      - **Event-driven**
    - **Then, calculate.**

# Measuring CPU Utilization

- **Event-driven measurement technique (z/OS)**
  - **For each Scheduler event**
    - Context switches, Waits, Interrupts
  - **Record the exact time the event occurred**
  - **At the end of each period in which a Task used the processor, accumulate the amount of elapsed time in a counter at the Task level**
    - **CALLDISP** function can be used *from inside a Task* to update the Task level counters
  - **Scheduler puts the CPU in a Wait State if a task completes and there is no work ready to be dispatched**
    - Hardware accumulates elapsed time in the Wait State

$$u_{\text{CPU}} = (\text{Duration} - T_w) / \text{Duration}$$

# Measuring CPU Utilization

- **Event-driven measurement technique (z/OS)**
  - **Extremely accurate (even when compared to a hardware monitor)**
  - **Processing that cannot be attributed to a specific Task is **uncaptured****
    - e.g., system-level Interrupt processing
      - **Can normally be apportioned based on workload I/O rates**
  - **Discretionary work that the Scheduler performs when there are no Ready tasks, but before the machine is placed in the Wait State, may inflate % CPU busy at **low utilizations****

# Measuring CPU Utilization

- **Sample-driven measurement technique (Windows)**
  - **Processor state is sampled once every clock interval**
    - **Virtual Timer ticks occur 64 times per second**
  - **Running thread is charged for all the elapsed time in the previous interval**
  - **If no thread is active, charge the Idle thread**

$$u_{\text{CPU}} = (\text{Duration} - T_{\text{IDLE}}) / \text{Duration}$$

**Assumes Processor HALT instruction is only used to conserve power**

# Measuring CPU Utilization

- **Sample-driven measurement technique (Windows)**
  - **Sampling errors are likely when the number of observations is small (e.g.,  $n < 300$ )**
  - **No mechanism to get accurate CPU timings from inside a thread**
  - **Processing that cannot be attributed to a specific Task is **uncaptured****
    - **e.g., Device Driver processing**
      - **Physical I/O rates are not captured at the Process level**
      - **A system thread performs all file Cache Lazy Writes, not the original process**

# Measuring CPU Utilization

- **Sample-driven measurement technique (Windows)**
  - **Currently augmented by instrumentation in the thread Dispatcher, similar to mainframe z/OS**
  - **QueryThreadCycleTime API to get CPU timings from inside a thread**
  - **Augmented by ETW trace event triggered by CPU **context switches**, beginning in Vista and Longhorn Server**

# Measuring Disk Utilization

- **Sample-driven measurement technique (OS/360 and 370)**

- **Sample UCB busy bit many times per second:**

$$U_{\text{disk}} = \text{Busy samples} / \text{Total Samples}$$

- **UCBBSY is systematically **under-sampled** because higher priority interrupt processing can preempt the sampling function**
- **Overhead** increases as a function of the # of devices
- **augmented by UCB I/O event counter is used to calculate service time from utilization**

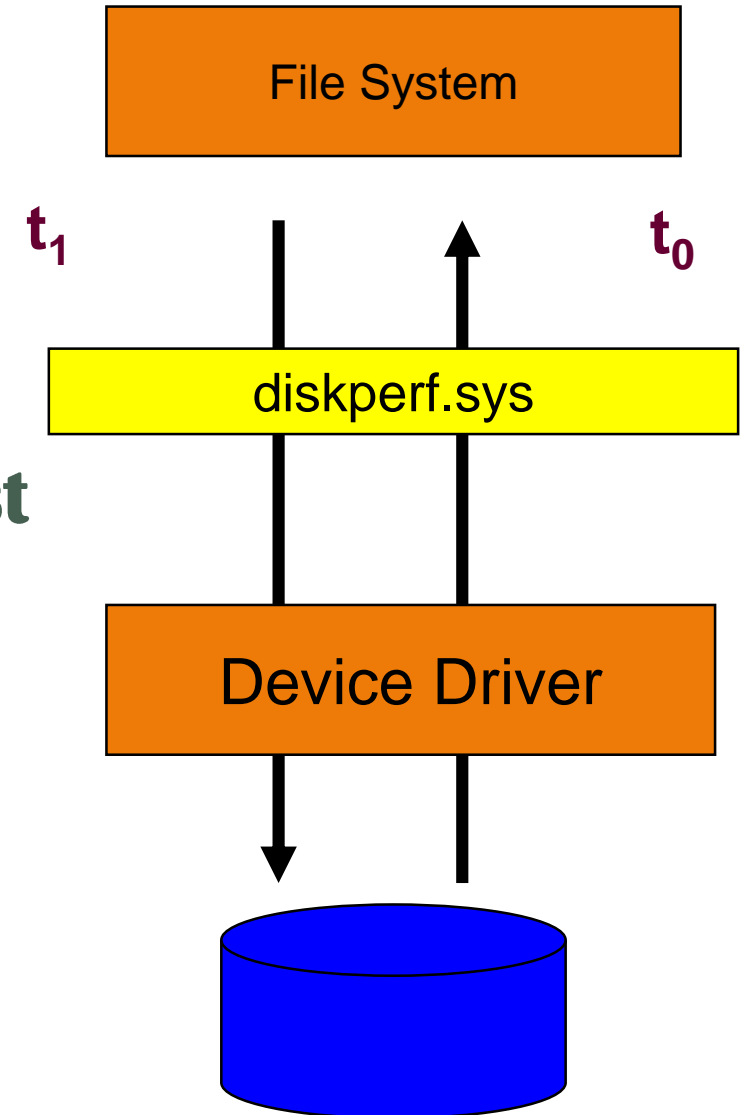


# Measuring Disk Utilization

- **Event-driven measurement technique (MVS/XA)**
    - **Hardware timing mechanism**
      - Reduced overhead
      - Timer ticks based on the processor clock
    - **At the end of each channel program, accumulate**
      - **Connect Time** (permits chargeback by I/O bytes)
      - **Disconnect Time**
- in the Channel Measurement Block (CMB) in HSA**

# Measuring Disk Response time

- Event-driven measurement technique
- Store hardware Timestamp Counter (TSC) value at the end of the last I/O operation in the request queue
  - QueryPerformanceCounter  
QueryPerformanceFrequency
- Store TSC value when the next I/O initiated

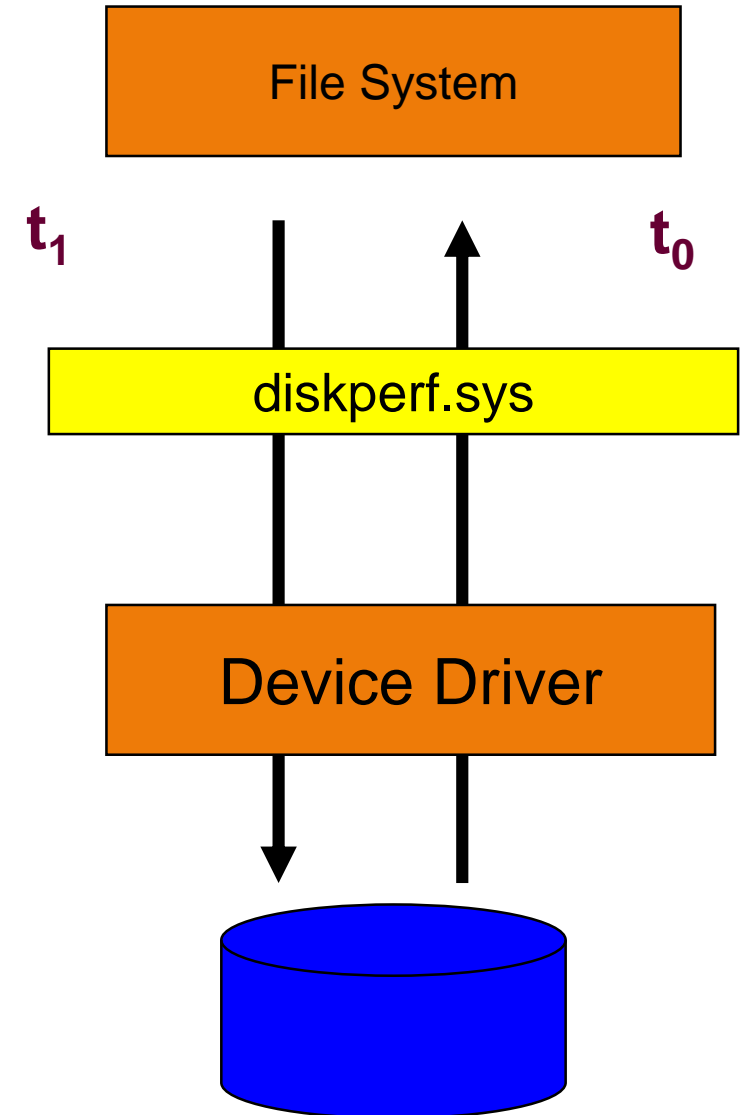


# Measuring Disk Response time

- Accumulate  $t_1 - t_0$  to measure % Idle Time

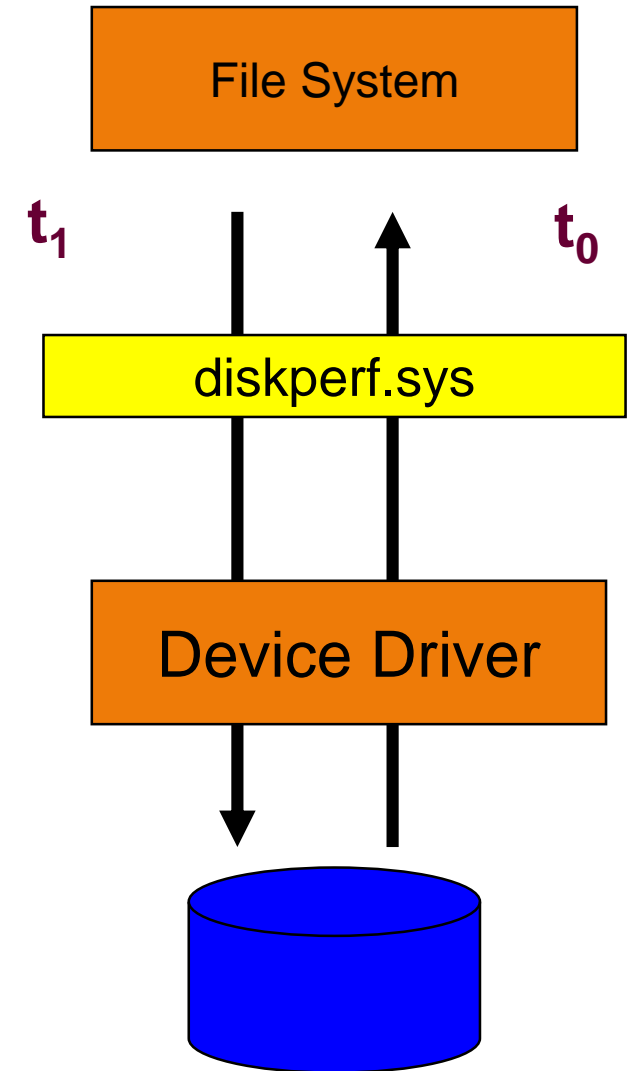
- Calculate:

$$\% \text{ Idle Time} = \Sigma(t_1 - t_0) / \text{Duration}$$



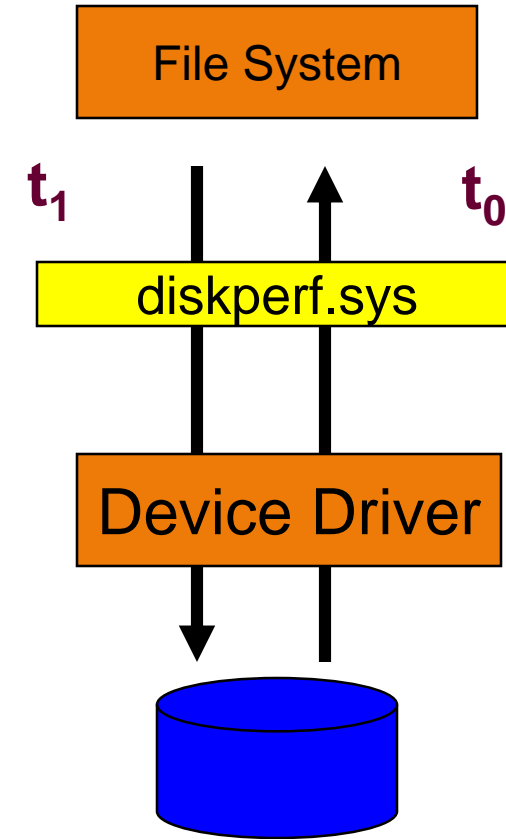
# Measuring Disk Response time

- Values **diskperf** measures:
  - $\lambda$  (Arrival rate): **Disk transfers/sec**
  - **RTT (Response time): Avg Disk secs/transfer**
  - **Throughput: Avg Disk Bytes/transfer**
  - **100%-utilization: % Idle time**
- All other disk I/O Counter values are *derived*.



# Measuring Disk Utilization

- You can calculate :
  - **% Disk busy = 100 - % Idle time**
- If it is an actual disk, calculate:
  - **Disk Service time = % Disk busy / Disk transfers/sec**
  - **Disk Queue time = Avg. Disk secs/transfer - Disk service time**

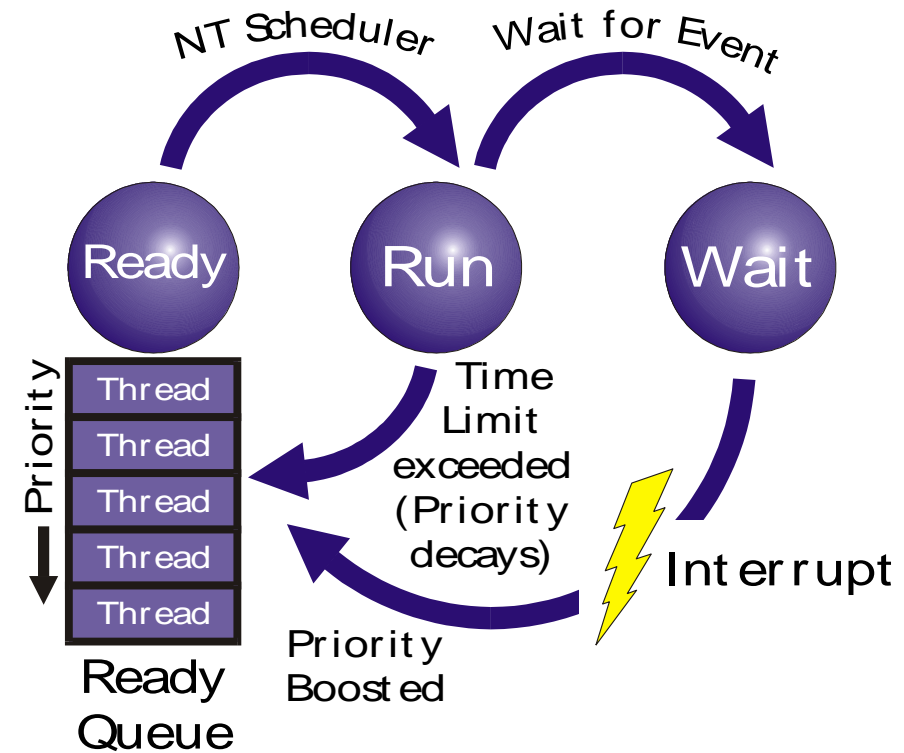


# Measuring Resource Utilization

- **Next, let's review some of the specific measurement issues that arise for specific types of resources:**
  - **CPU**
  - **Memory**
  - **Disk**
  - **Network**
  - **Tasks or Processes**

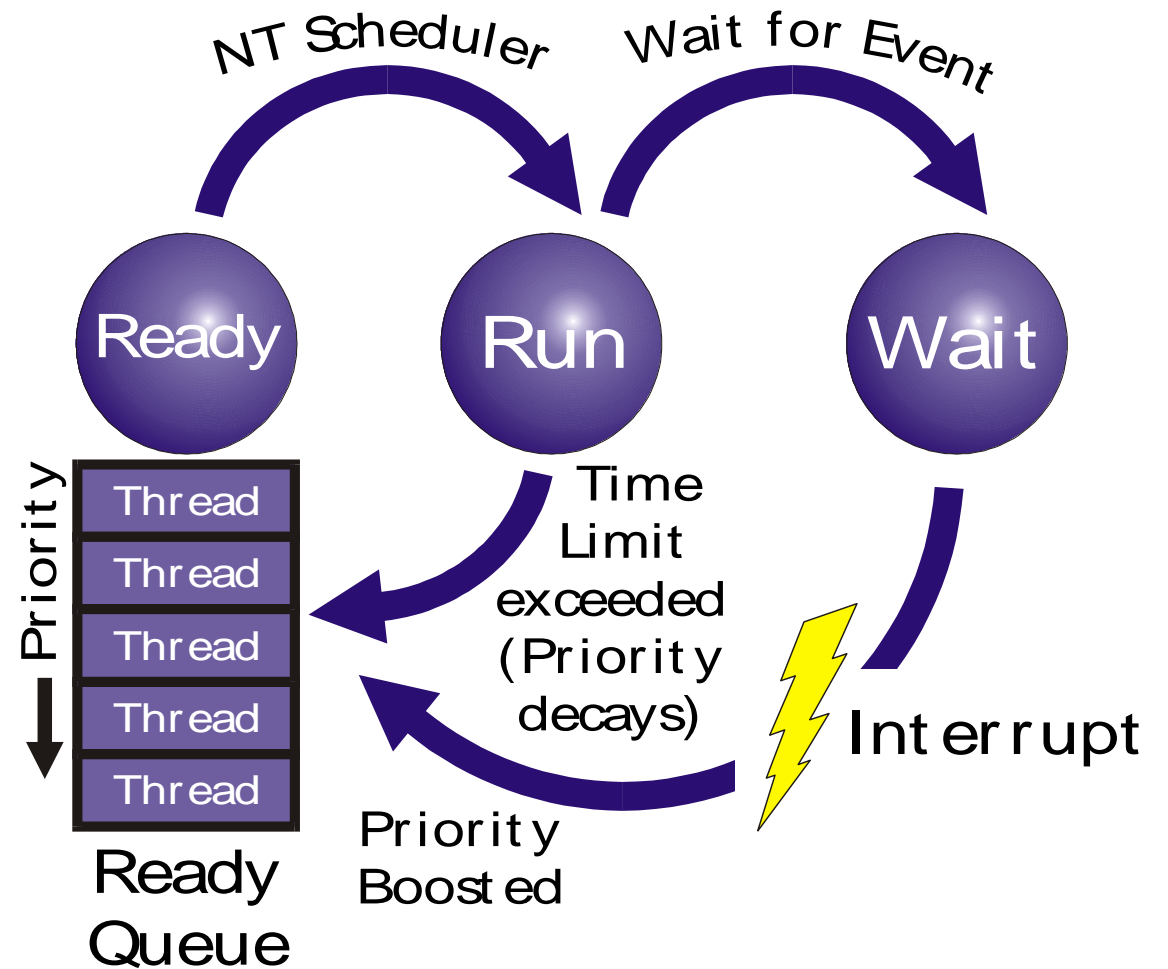
# Measuring CPU Utilization

- Thread is the unit of dispatching
  - Context switch
- Priority queuing with preemptive scheduling
  - Workloads receive favored access to the processor based on their **priority**
    - Fixed
    - Dynamic: **MTTW** used to re-balance the priorities of CPU-bound and I/O bound work automatically
  - Interrupt servicing automatically receives highest available priority
  - Waiting threads in the **Ready Queue**



# Measuring CPU Utilization

- **Context switch**
  - **preemption**
  - **Quantum time slice expiration**
  - **“Idle thread”**
- **Can we look inside the thread Ready Queue?**
- **At the Thread level:**
  - **Thread State**
  - **Wait State Reason**



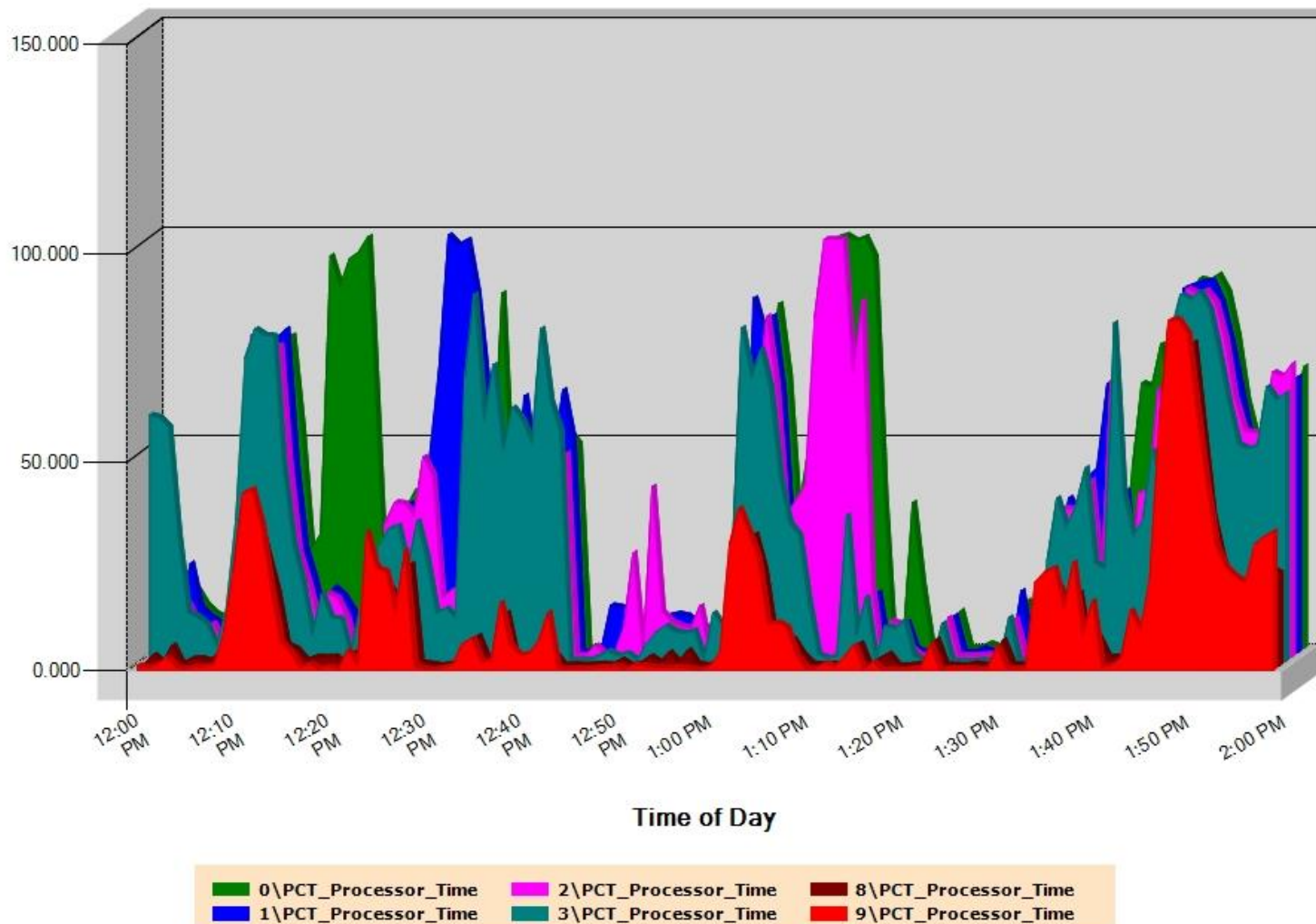


# Measuring CPU Utilization

- **Multiprocessors**
  - **Symmetric Multiprocessing defined**
    - all threads & interrupts equally likely to run on any processor
  - **Asymmetric Multiprocessing**
    - **soft affinity**: attempt to dispatch work on the same processor where it ran last
      - benefit from a **cache warm start**
    - Selective enablement of Interrupts (z/OS)
    - Hardware partitioning strategies (hard affinity)
  - **ccNUMA (node affinity)**

# ccNUMA Multiprocessors

NTSMF Report for MLTPSDB21 Processor Usage on: 4/27/2005 12:00:00 PM



# Measuring CPU Utilization

- **Access processor Hardware counters to learn about:**
  - **Instruction execution rate** (Intel Architecture: Instructions retired/sec)
  - **execution pipeline stall**
  - **Processor Cache misses and other types of instruction execution pipeline delays**
  - **Front-side bus utilization**
- **How are CPU utilization measurements perturbed when virtualization is used?**
  - **virtual clock (rdtsc)**

# Measuring Memory Usage

- **Memory is not utilized in quite the same way as other resources**
  - **Memory is allocated & occupied **continuously**, until it is freed or released**
  - **In other words, Little's Law does not apply!**
  - **Counting memory allocation/deallocation events is subject to high overhead**
    - **Many allocations are very transient**
    - **Some allocations are extremely persistent**
  - **Some Processes use internal memory management schemes that are not visible to the OS memory Manager**
    - **e.g., SQL Server DBMS process, any .NET application or Java applet**

# Measuring Memory Usage

- **Memory is not utilized in quite the same way as other resources**
  - **Shared memory locations are subject to locking and lock contention**
    - **Instrumentation for locking is typically not enabled due to performance considerations because it make contentions significantly worse**
    - **e.g., the File System locks**
  - **As process virtual memory is mapped to actual hardware memory, contention for physical memory can occur (due to **over-commitment**), but this is not queuing either, in any sense that would allow Little's Law to apply**

# Measuring Memory Usage

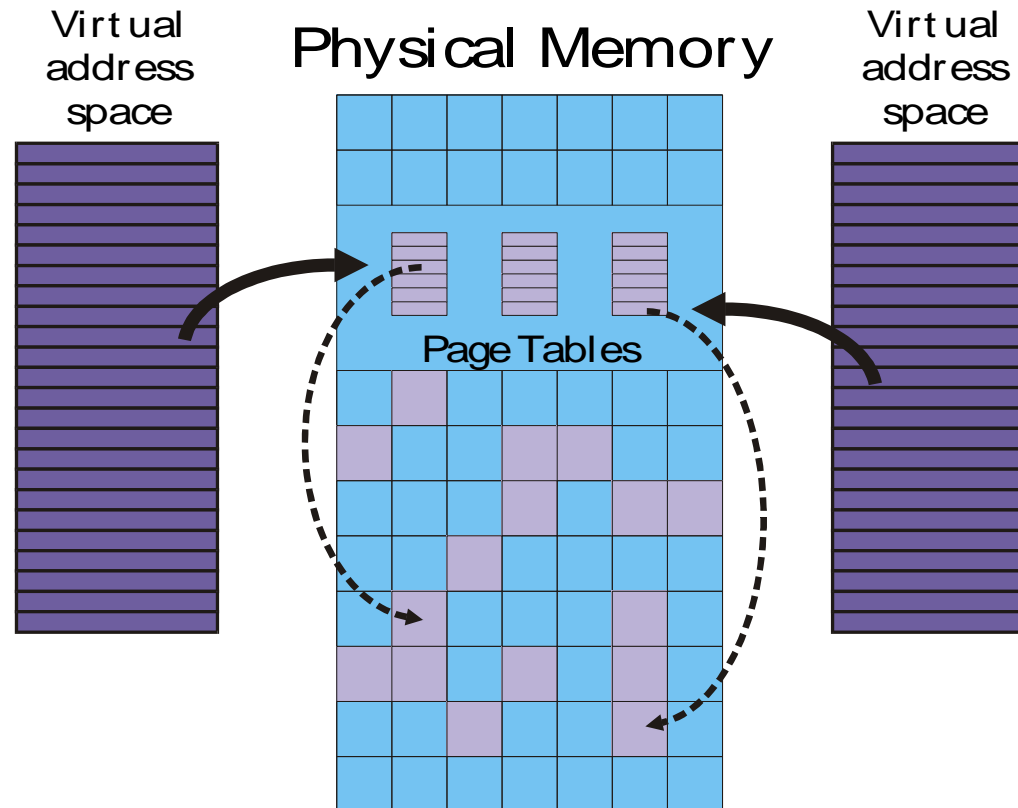
- **Memory is not utilized in quite the same way as other resources**
  - **The unit of virtual memory allocation is the Process**
    - **Why?**
  - **OS can maintain physical and virtual memory allocation high water marks that are useful in page stealing**
  - **Detail at the process level requires use a state sampling approach, periodically taking an inventory of current memory allocations**
    - **Overhead issues as the size of memory increases**

# Measuring Memory Utilization

- **Physical (or Real) memory (RAM)**
  - an LRU-managed cache
  - tends to become fully occupied, except for a small cushion of **Available Frames** : (physical memory Pages)
- **Virtual Memory**
  - Logical view of memory presented to an address space
  - Virtual memory page may or may not be resident in RAM
    - mapped to RAM using **Page Tables** (hardware-specific)
  - Access to a virtual memory location that is not resident in RAM leads to a **page fault**
  - When RAM is full, page stealing occurs to replenish the Available Frame cushion

# Measuring Memory Utilization

- **Physical:Virtual Memory**





# Measuring Memory Utilization

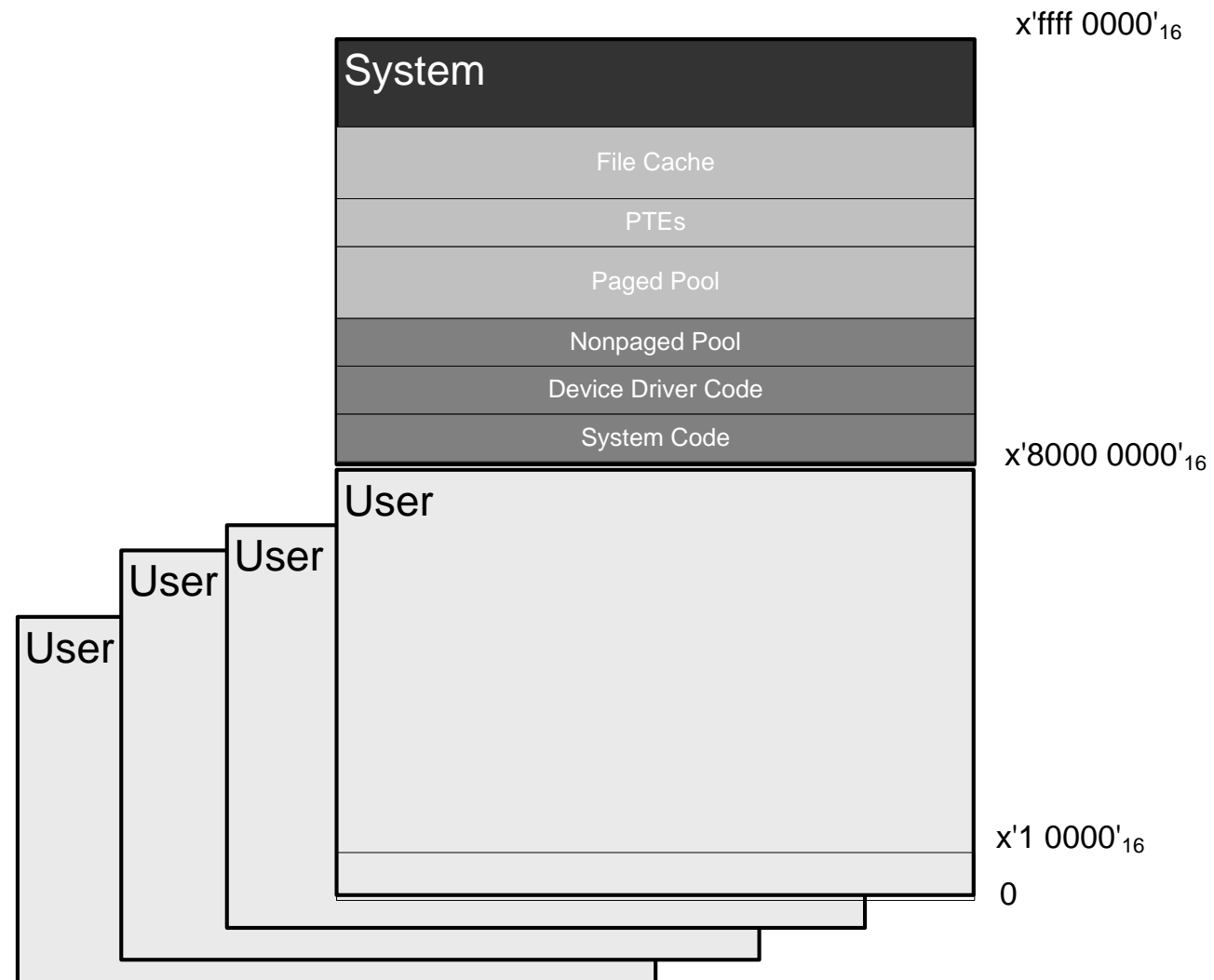
- **Paging activity is counted by the OS**
  - **Page-Ins**
  - **Page-Outs (for changed or “dirty” pages)**
  - **The **delay** a process suffers when it accesses a non-resident page is usually not measured**
    - **page fault resolution time** can often be inferred from disk performance statistics
  - **Memory management overhead instrumentation is very useful when it is available (**steal scans/sec, rate of garbage collections**)**
  - **Age of a page can sometimes be determined per address space**
    - **Unreferenced Interval Count (UIC) from clock algorithm used in z/OS)**
    - **Windows ages pages using a set of four sub-pools of the Available pool**

# Measuring Memory Utilization

- **Overall paging activity rates are often correlated with the V:R ratio**
  - **Windows Hyper-V dynamic memory manager uses this statistic to balance memory usage among Guest machines**
- **paging activity per Process**
  - **not available in Windows NT because the counter also includes “soft” pages**
- **per Process **working sets** (resident pages)**
  - **Normally, an active page is resident; inactive pages are paged out**
  - **Management controls override global LRU and protect specific process working set pages from being stolen**

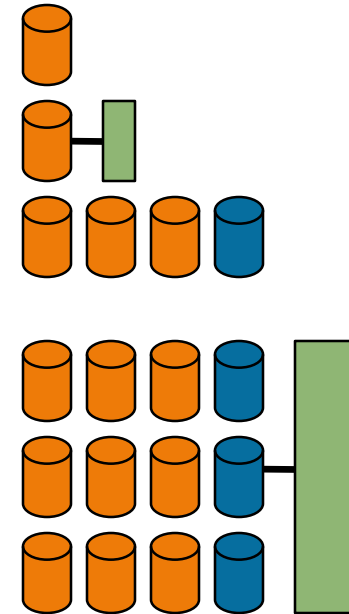
# Measuring Memory Usage

- **System (shared) virtual memory map**
  - **Memory utilization of system areas**
- **per Process virtual memory**
  - **Memory leaks**



# Measuring Disk Utilization

- **What is the physical disk entity?**
  - **A simple disk**
  - **A disk installed behind a caching controller**
  - **An array of disks (RAID?)**
  - **SSDs**
- **A virtual disk exposed by a storage processor**
- **If it is not a simple disk, you will likely need performance data from the storage controller to diagnose a performance problem**



# Measuring Disk Utilization

- **SSDs**
  - **revolution in disk performance**
  - **access patterns do not matter, but Reads vs. Writes does**
- **Rotating mechanical disks**
  - **What is the access pattern of the physical disk?**
    - **Optimize seek time by packing files accessed concurrently close to each other on the disk**
    - **Determine suitability for cache**
  - **Can normally only be determined from file I/O event tracing**

# Measuring Network Utilization

- **Utilization can be measured from byte count fields in packet headers**
  - **Utilization (or bandwidth) is much less important than latency (or RTT, Round Trip Time) for response-oriented workloads, especially over long distance**
    - **You can always add capacity**
    - **You cannot do much about the speed of light**
  - **High interrupt processing rates (due to small-sized packets) cause performance issue at network hubs**
    - **Typical ACK packet = 50 bytes**
    - **Ethernet packet size = 1500 bytes**
    - **TCP Window size usually < 64KB**

# Measuring Utilization by Process (or Task)

- **Diagnosing performance problems at the application level**
- **Response Time decomposition**
  - **Bottleneck analysis: Look for the resource with the heaviest utilization or fastest growing queue**
- **Capacity planning**
  - **Different workloads have different rates of growth**

# Measuring Utilization by Process (or Task)

- **Processor**
  - Utilization by processor state (User vs. kernel mode)
  - Profilers can drill down to report usage by module and instruction
- **Memory**
  - Working sets
  - Virtual memory allocated
  - Paging rate
- **Disk**
  - I/Os per disk ( $m * n$  volume considerations)
  - I/Os per file ( $m * n$  volume considerations)
- **Network**
  - By Protocol
  - By Application (TCP Port)



# Measuring Utilization by Process (or Task)

- **Validity issues (aka “capture ratio” problems)**

`SUM(Process.% Processor Time) < SUM(Processor.%Processor Time)`

`SUM(Process.Working set) + System.Working set > sizeof(RAM)`

# Measuring Utilization by Thread (subtask)

- **Processor utilization**
  - **Virtual memory address space is shared, but you may see local heap allocation statistics**
- **Parallelism and Concurrency**
  - **Increasingly important with multi-core processors**
  - **At the thread level, it is difficult to identify the function that is executing without access to source code**
  - **Amdahl's Law on parallel scalability**
  - **Serialization and Locking**

# Measuring Queues

- **Useful in Bottleneck analysis**
  - **Predictable relationship between**
    - **Arrival/completion rate**
    - **Service time**
    - **utilization**
    - **response time**
  - **Sampling issues**
  - **Queue length is often calculated, instead of being measured directly.**
- **Elements of queuing models**
  - **Solve using simulation**
  - **Solve analytically**

# Measuring Queues

- **Queuing models: an analytic tool for capacity planning**
  - **Capture and characterize the current workload**
  - **Create an accurate mathematical representation of the flow of work through the system**
    - **During **calibration**, the model values output are *validated* against actual response time & queue length measurements**
  - **Forecast the new workload**
  - **Evaluate the model at the new, projected workload level**

# Measuring the Application

- **The application may report unique, domain-specific measurements**
- **The application may have unique, domain-specific measurement sources**
- **Using domain-specific tools effectively may require an understanding of the internal application architecture**
  - **Web servers**
  - **Database servers**
  - **Messaging and Collaboration servers**

# Measuring the Application

- **Web servers (Apache, Websphere, IIS, etc.)**
  - **Web logs contain a record of every Request (URL)**
    - Regularly mined for information about content management
  - **Clustered for reliability and scalability**
  - **Competing approaches to dynamic HTML**
    - client-side JavaScript, J2ee JavaBeans, ASP.NET, etc.
  - **Multiple application tiers**
    - Presentation layer (including client-side scripts)
    - Middleware (business Objects)
    - Permanent back-end data store
  - **Session-oriented management**
    - http protocol was original conceived as session-less (REST)
  - **caching strategies**

# Measuring the Application

- **Database servers (DB2, Oracle, SQL Server)**
  - **I/O scalability**
  - **Data-in-Memory buffering of frequently accessed objects**
  - **Serialization & locking (transaction processing)**
  - **Query EXPLAIN**
- **Messaging and Collaboration servers**
- **Web services**

# Measurement sources

- **Continuous, interval-based monitors**
  - **Calculate & report aggregated and summary statistics**
  - **Many of the statistics reported are derived from event-oriented instrumentation**
    - e.g., I/Os per second, bytes/sec, packets/sec
- **Point-in-time snapshots**
  - e.g., periodically, inventory the file system
- **Event logs and Diagnostic traces**
  - **Time-stamped**
    - Packet sniffers
    - Web logs
  - **log performance statistics upon completion (e.g., Process accounting)**
  - **Many diagnostic facilities produce so much data that you could afford to run them for short periods of time**
    - Sometimes use circular buffers that are overridden as necessary



# Measurement tools

- **Performance monitors**
  - **Umbrella category that may range far afield**
  - **Includes specialized, high impact, application-level profilers**
- **Performance data bases**
  - **Repository where statistics from many data sources and/or machines are aggregated and summarized**
  - **Correlate information in time**
  - **Report generation**
- **Analytic and simulation modeling**
- **Load generation and stress testing**
  - **Generate repeatable benchmarks**

# Example: RMF/SMF

- **Record-oriented**
  - **RMF interval duration**
  - **RMF sampling rate**
  - **Most SMF records are event-oriented, with the exception of the SMF Type 30 interval accounting record**
- **Extensible**
- **Additional performance counters provided by:**
  - **CICS**
  - **DB2**
  - **plus, many more**

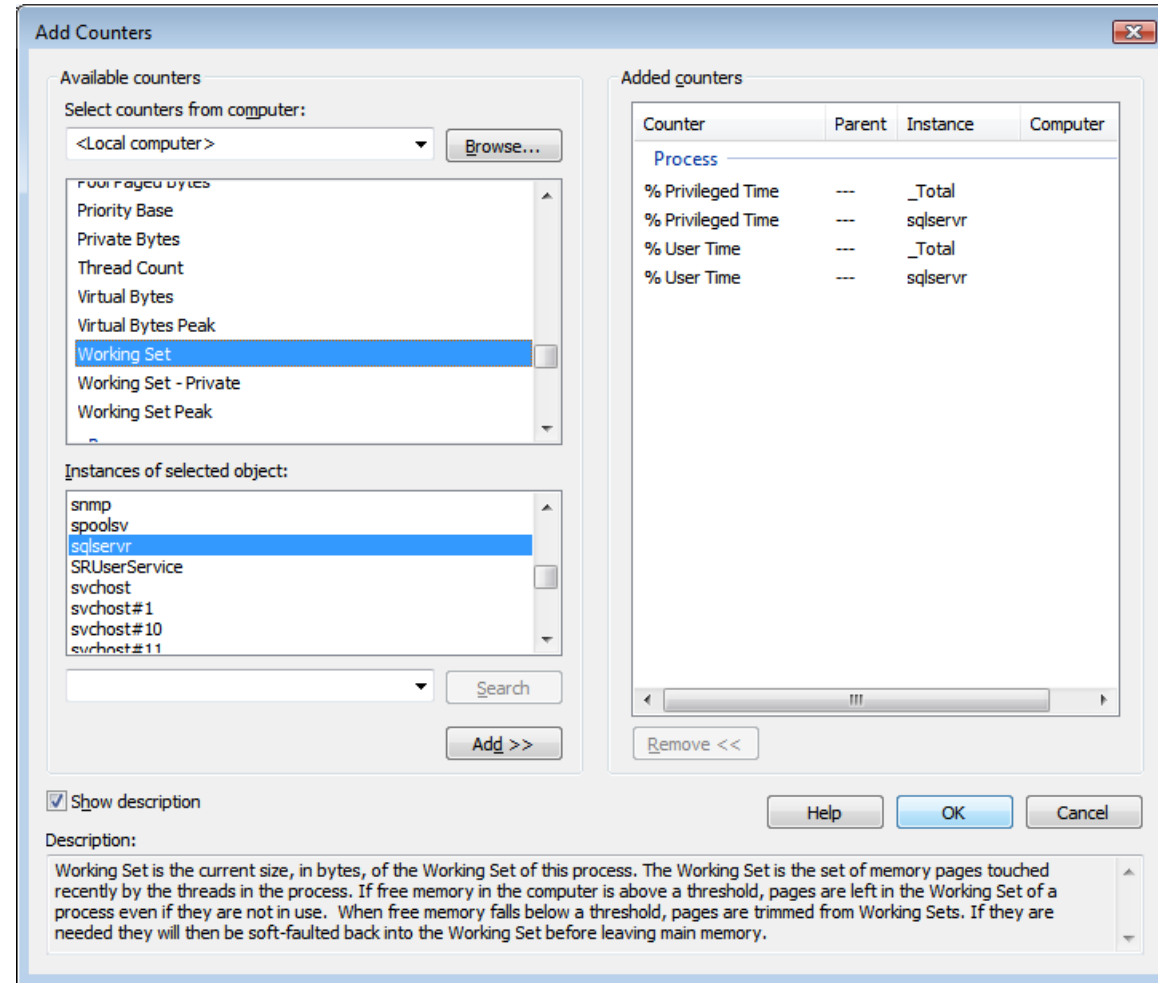
# Example: Windows Performance monitoring API

- **Object-oriented**
- **Extensible**
- **Additional performance counters provided by:**
  - **Exchange**
  - **SQL Server**
  - **Internet Information Server**
  - **Hyper-V**
  - **plus, many more**

# Windows Performance monitor API

## • Objects

- some Objects are instanced
- Some instanced Objects have parent relationships
- **\_Total** instance
- Each Object has a corresponding collection service



# Windows Performance monitor API

- **Standard Counters**

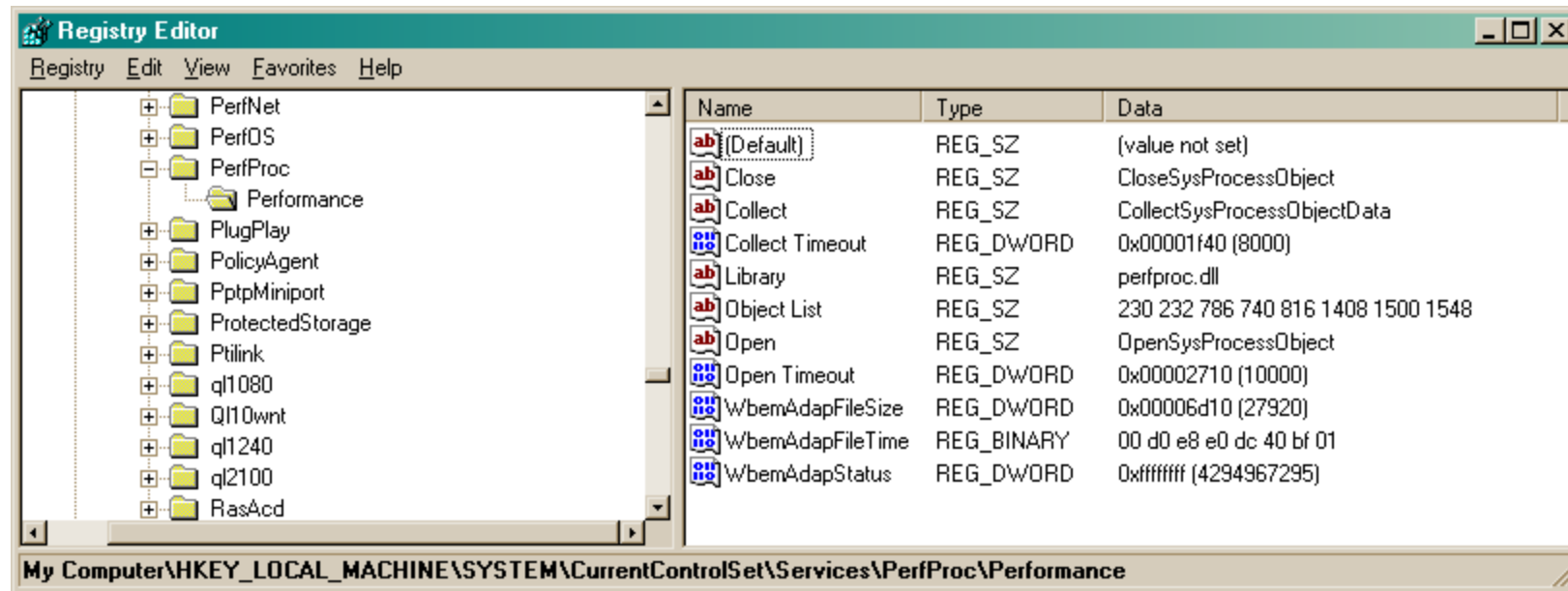
- **System**
- **Processor**
- **Memory; Cache; Paging Files**
- **Physical Disk: Logical Disk**
- **Process: Thread**
- **Network services**
  - **Browser, Server, Redirector**
  - **TCP/IP**
- **Active Directory**

- **Extended Counters**

- ◆ **SQL Server**
- ◆ **MS Exchange**
- ◆ **IIS Web Server**
- ◆ **Message Queue Server**
- ◆ **SNA Server**
- ◆ **SMS**
- ◆ **3<sup>rd</sup> party products**
- ◆ **Home grown application monitoring**

# Windows Performance monitor API

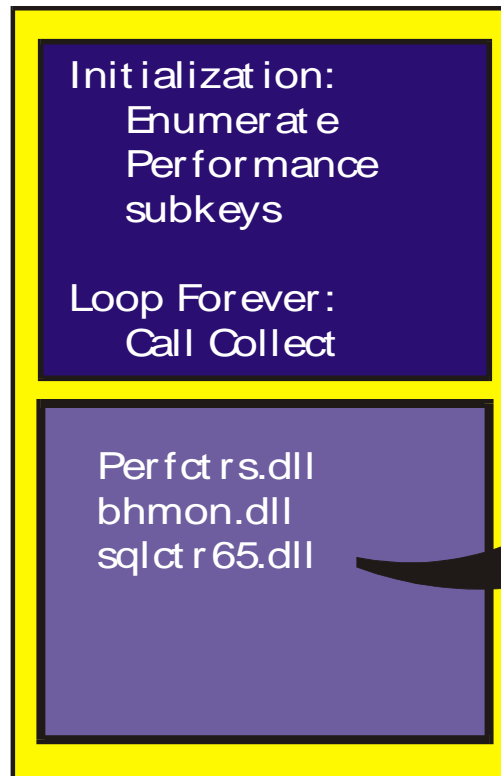
- Locate the extended Perflib DLLs by scanning **HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\service-name**



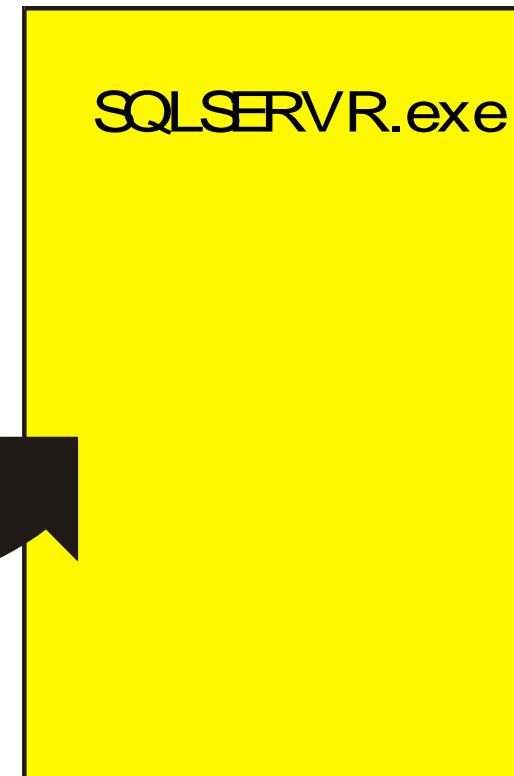
# Windows Performance monitor API

- **Perflib extensions must synchronize with any separate monitored address spaces**

Performance Monitoring application address space



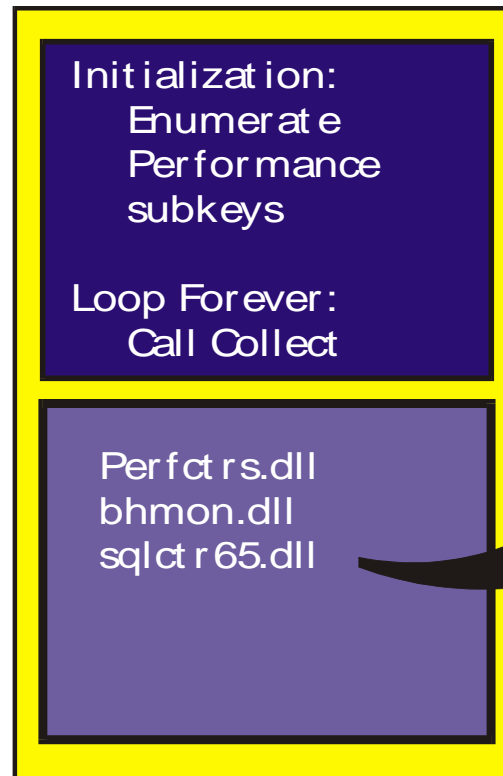
Monitored application address space



# Windows Performance monitor API

- When running *remotely*, Perflib dlls are loaded into the Winlogon.exe (win2k) or Regsvc.exe (win2k23) address space.

Performance Monitoring application address space



Monitored application address space





# Windows Performance monitor API

## □ Counter Types

- ◆ **Encapsulates the formula needed to compute the measurement value**
- ◆ **See [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/countertype\\_qualifier.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/countertype_qualifier.asp)**
  - **Noncomputational Counter Types**
  - **Basic Algorithm Counter Types**
  - **Counter Algorithm Counter Types**
  - **Timer Algorithm Counter Types**
  - **Precision Timer Algorithm Counter Types**
  - **Queue-length Algorithm Counter Types**
  - **Base Counter Types**
  - **Statistical Counter Types**

# Counter Types (from WMI spec)

0	PERF_COUNTER_RAWCOUNT_HEX
256	PERF_COUNTER_LARGE_RAWCOUNT_HEX
2816	PERF_COUNTER_TEXT
65536	PERF_COUNTER_RAWCOUNT
65792	PERF_COUNTER_LARGE_RAWCOUNT
4260864	PERF_SAMPLE_COUNTER
4523008	PERF_COUNTER_QUEUELEN_TYPE
4523264	PERF_COUNTER_LARGE_QUEUELEN_TYPE
5571840	PERF_COUNTER_100NS_QUEUELEN_TYPE
272696320	PERF_COUNTER_COUNTER
272696576	PERF_COUNTER_BULK_COUNT
537003008	PERF_RAW_FRACTION

# Windows Performance monitor API

## □ Counter Types

### ◆ Accumulators or Difference Counters

- used to calculate interval  $\Delta s$
- $\Delta / \text{interval} : \text{rate/second}$
- many variants: CPU time, etc.

### ◆ instantaneous Counters

- Integer Counters representing values *right now*
- e.g., memory occupancy; current queue length

### ◆ compound Counters

- utilization: used / capacity
- cache hit ratios

# Windows Performance monitor API

- **API to add performance Objects**
  - **over twenty different Counter types supported**
- **API to retrieve performance Objects**
  - **e.g., sample code to enumerate active Processes**
- **Documented interfaces:**
  - **How to build a Perflib examples**
  - **How to access and display counters (PDH Library)**

# Windows Performance monitor API

## □ Difference Counters

### ◆ **PERF\_COUNTER\_COUNTER, PERF\_COUNTER\_BULK\_COUNT**

- e.g., Total Interrupts/second, File Read Bytes/second

### ◆ **$(Counter_{+1} - Counter_0) / Duration$**

### ◆ **summarization rule:**

- **$Counter_{+1}$**

- **$Counter_{+2}$**

- **$Counter_{+3}$**

- **$Counter_{+4}$**

- **$Counter_{+5}$**



$$(Counter_{+5} - Counter_{+1}) / Duration$$

# Windows Performance monitor API

## □ Time difference Counters

### ◆ **PERF\_100NSEC\_TIMER**

□ e.g., % Processor Time, % Interrupt Time

◆  $(Timer_{+1} - Timer_0) / Duration$

◆ summarization rule:

□  $Timer_{+1}$

□  $Timer_{+2}$

□  $Timer_{+3}$

□  $Timer_{+4}$

□  $Timer_{+5}$



$(Timer_{+5} - Timer_{+1}) / Duration$

# Windows Performance monitor API

## □ Time difference Counters

### ◆ **PERF\_100NSEC\_TIMER**

- e.g., Processor, Process, and Thread % Processor Time
- ◆ **Timer Tick Counts maintained at the Process/Thread level by the NT Scheduler**
- ◆ **Counter design constrained 0 - 100% range**
  - **Multithreaded Process % Processor Time in an SMP is a problem**
- ◆ **Continuously maintained, but sensitive to the collection interval:**
  - **residual data is lost when the Process/Thread terminates**

# Windows Performance monitor API

## □ Time difference Counters

### ◆ **PERF\_100NSEC\_TIMER\_INV**

□ e.g., **System % Total Processor Time**

### ◆ **Idle Thread dispatched when there is no other work to do**

□ **one Idle Thread per processor**

□ **100 - Idle Thread % Processor Time**

□ **System % Total Processor Time is calculated as the average over all Processors**

$\Sigma$  Processor % Processor Time >  $\Sigma$  Process % Processor Time



# Windows Performance monitor API

## □ Time difference Counters

### ◆ **PERF\_COUNTER\_TIMER**

□ e.g., % Disk Read Time, % Disk Time

◆ **Timer Values maintained by the Diskperf Driver**

◆  **$(Timer_{+1} - Timer_0) / Duration$**

◆ **summarization rule:**

□  **$Timer_{+1}$**

□  **$Timer_{+2}$**

□  **$Timer_{+3}$**

□  **$Timer_{+4}$**



**$(Timer_{+4} - Timer_{+1}) / Duration$**

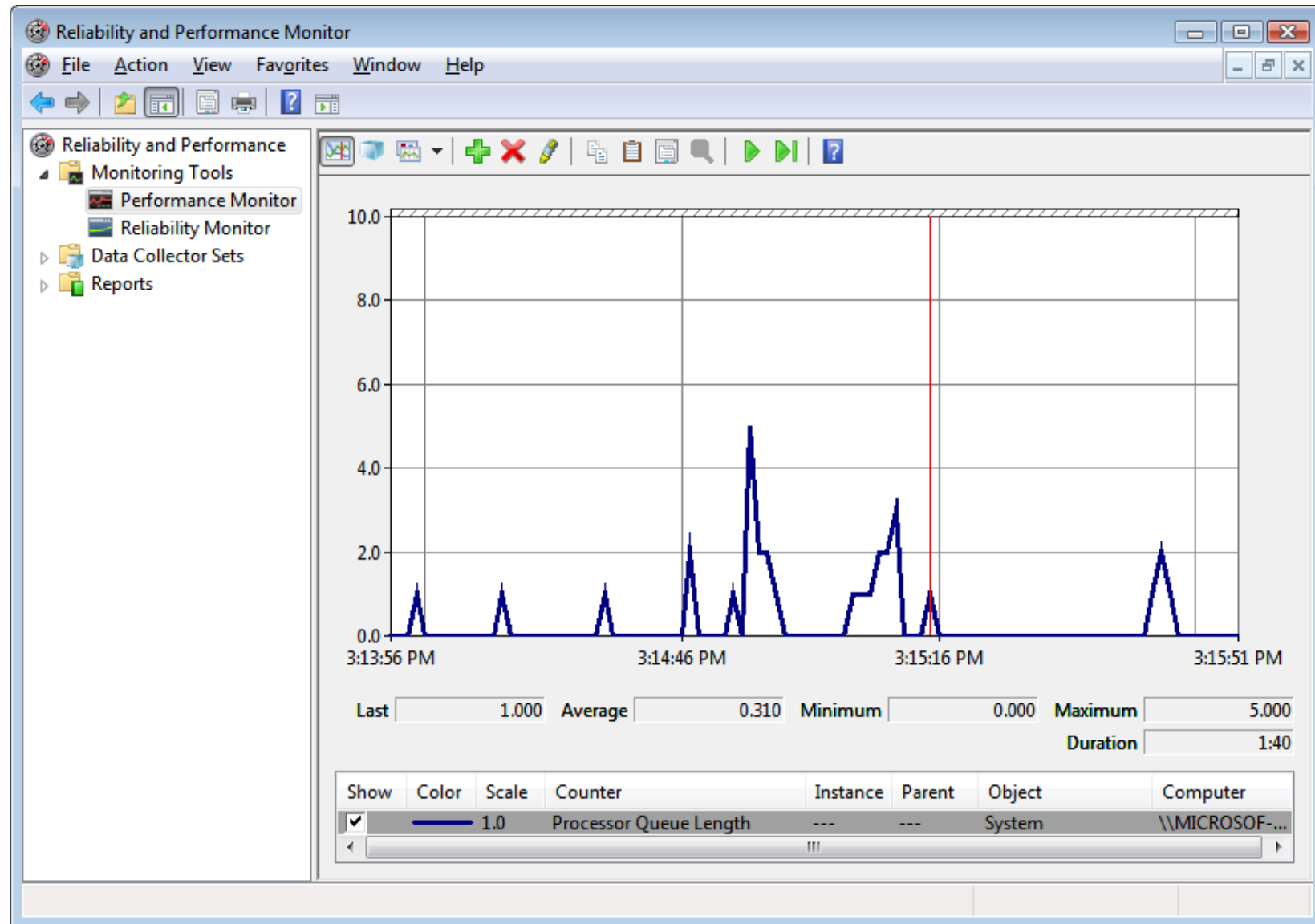
# Windows Performance monitor API

## □ Instantaneous Counters

- ◆ **PERF\_COUNTER\_RAWCOUNT**
  - e.g., Processor Queue Length, Available Bytes
- ◆ Integer value
- ◆ No summarization rule.
  - One instance of a sample

# Windows performance metrics

- e.g., Processor Queue Length
  - Instantaneous Counters always report integer values



# Windows Performance monitor API

## □ Other Counters

### ◆ **PERF\_ELAPSED\_TIME**

- **e.g., System Up Time, Process Elapsed Time**
- **Process start or stop time is not available**

### ◆ **PERF\_RAW\_FRACTION**

- **e.g., Logical Disk % Free Space**
- **Instantaneous measurement: numerator & denominator must be supplied**
- **No summarization rule.**
- **One instance of a sample**

# Performance monitor *Overhead*

## □ **How much does performance monitoring cost?**

- ◆ **The overhead involved in *taking* the measurements**
- ◆ **The overhead involved in *gathering* the measurement data**
- ◆ **The overhead involved in *processing* the measurement data**
  
- ◆ **Measurement overhead should not contribute to performance problems, yet it must be detailed enough to resolve serious performance problems**
  - ◆ **The need to diagnose the cause of critical outages trumps overhead considerations every time.**

# Performance monitor *Overhead*

- **The overhead involved in taking the measurements**
  - ◆ **Counter maintenance is often a *mandatory* part of system overhead**
  - ◆ ***Counter* design promotes efficiency**
  - ◆ **Sampling rate vs. reporting rate**
  - ◆ **High overhead data Objects can be optional**

# Performance monitor *Overhead*

- **The overhead involved in gathering the measurement data**
  - ◆ **What is the data collection interval? 15 minutes?**
  - ◆ **Is there a retrieving data across the network**
  - ◆ **Data categories are often not very modular**
    - **e.g., in z/OS, instances of idle disks are collected across each machine**
    - **e.g., in z/OS, turning on SMF interval accounting can overwhelm existing procedures to capture & process SMF data**
    - **e.g., in Windows, all instances of an Object are collected each collection interval**
      - **Process/Thread Objects are tightly coupled due to overhead issues**
      - **# of Processes and Threads can be large**

# Performance monitor *Overhead*

- **The overhead involved in gathering the measurement data**
  - ◆ **Sampling frequency in Windows**
    - **No process accounting file, so if you do not want to miss too much data from a Process start or end...**
    - **Statistical significance of Instantaneous Counter samples**



# Performance monitor *Overhead*

- **The overhead involved in processing the measurement data**
  - ◆ Real Time problem alerts
  - ◆ Continuous data collection at sufficient granularity to support post hoc problem analysis, as well as capacity planning
    - ◆ e.g., see the [\*\*RRDtool\*\*](#)
  - ◆ Data is often transferred to another platform for post-processing.
  - ◆ Designing efficient data collection procedures for large scale installations is challenging.

# Performance investigations: Decomposition

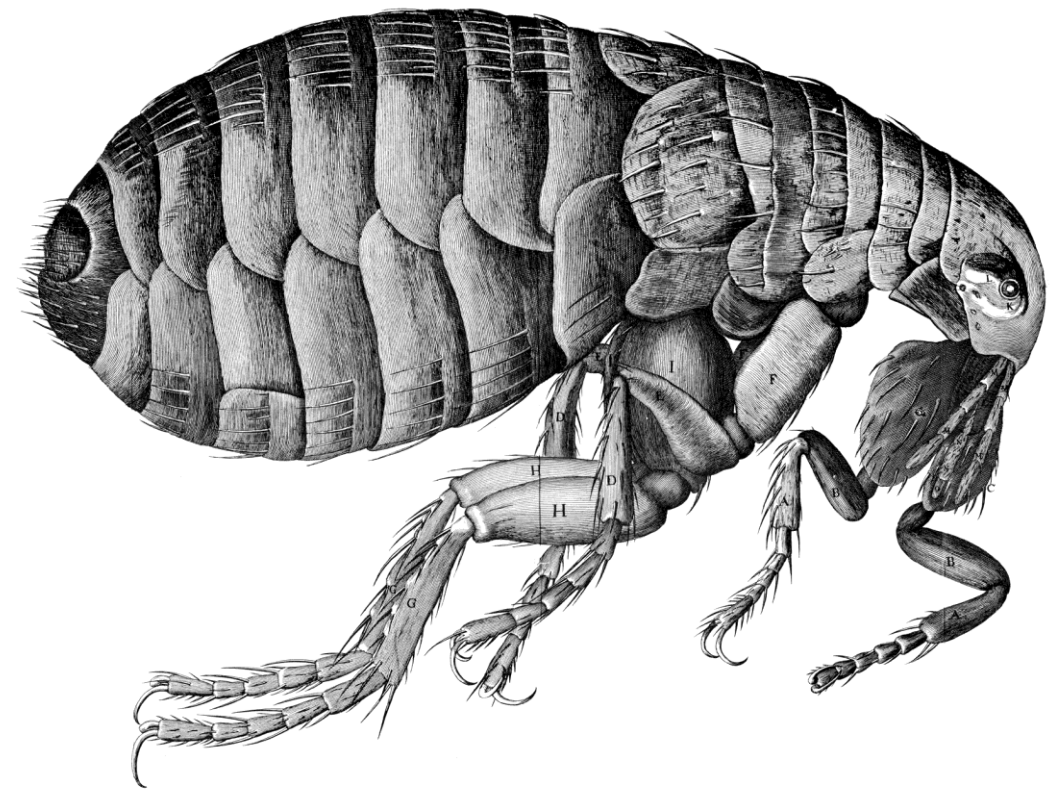
- **e.g., Why is this web application running slow?**
- **Measure, measure, measure!**
- **Measure the subcomponents of response time**
- **Bottleneck detection:**
  - **the subcomponent where the processing time is the longest**
  - **the subcomponent with the fastest growing queue**
- **Drilling deeper into the bottleneck using the appropriate probe:**
  - **application profilers**
  - **diagnostic tracing tools**
- **How deep do you need to go?**
  - **parallelism: concurrency & locking**
  - **frameworks** ⇒ **code generation** ⇒ **hardware instruction execution**
  - **networking stack:**
    - **HTTP** ⇒ **TCP** ⇒ **IP** ⇒ **Ethernet** ⇒

# Decomposition in performance investigations

- **How deep do you need to probe?**
  - **What instrumentation data do you have access to and can you comprehend it?**
  - **other constraints: staffing, time, money, etc.**

So, naturalists observe, a flea  
Has smaller fleas that on him prey;  
And these have smaller still to bite 'em,  
And so proceed *ad infinitum*.

-- Jonathan Swift, *On Poetry*



Engraving of a flea under magnification,  
recorded by Hooke, *Micrographia*, 1665.

# Event logs and Diagnostic Traces

- **Many types of low level diagnostic traces can also be used to resolve performance problems**
  - e.g., Packet sniffers, web logs, protocol analyzers
  - e.g., GTF, D-trace, Instruments, ETW, xEvent, collectd
- **Event-driven trace data have inherent **validity****
  - **But, trace capture tools often have major problems due to:**
    - the volume of data they can generate
    - the lack of data reduction tools
    - the lack of data analysis tools
    - inconsistent/incomplete data model

# Network protocol sniffers

- **Can report on *every* packet that passes through a network node (e.g., a router)**
  - originally, the function was performed using hardware
  - where you tap into the network determines what packets your sniffer sees
- **Typically, they can understand and decode packet headers**
  - ethernet, IP, TCP, HTTP, Oracle RPCs, etc.



Apply a display filter ... &lt;Ctrl-/&gt;

Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
612	17.258232	192.168.1.1	192.168.1.143	DNS	156	Standard query response 0xd6da A adclick.g.doubleclick.net CNAME page
613	17.260562	74.125.225.153	192.168.1.143	HTTP	443	HTTP/1.1 200 OK
614	17.261568	173.194.46.102	192.168.1.143	TCP	54	443 → 61550 [ACK] Seq=5601 Ack=7457 Win=670 Len=0
615	17.269121	74.125.225.26	192.168.1.143	TCP	66	80 → 61617 [SYN, ACK] Seq=0 Ack=1 Win=42900 Len=0 MSS=1430 SACK_PERM=
616	17.269171	192.168.1.143	74.125.225.26	TCP	54	61617 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
617	17.269385	192.168.1.143	74.125.225.26	HTTP	529	GET /pagead/imgad?id=CICAgKDj1Z68TRABGAEyCLi5xpg77JJ HTTP/1.1
618	17.304635	74.125.225.26	192.168.1.143	TCP	54	80 → 61617 [ACK] Seq=1 Ack=476 Win=42880 Len=0
619	17.308120	74.125.225.26	192.168.1.143	TCP	1484	80 → 61617 [ACK] Seq=1 Ack=476 Win=42880 Len=1430 [TCP segment of a r
620	17.310132	192.168.1.143	74.125.225.153	TCP	54	61593 → 80 [ACK] Seq=833 Ack=390 Win=65280 Len=0
621	17.310971	74.125.225.26	192.168.1.143	TCP	1484	80 → 61617 [ACK] Seq=1431 Ack=476 Win=42880 Len=1430 [TCP segment of
622	17.311017	192.168.1.143	74.125.225.26	TCP	54	61617 → 80 [ACK] Seq=476 Ack=2861 Win=65536 Len=0
623	17.313639	74.125.225.26	192.168.1.143	TCP	1484	80 → 61617 [ACK] Seq=2861 Ack=476 Win=42880 Len=1430 [TCP segment of
624	17.316273	74.125.225.26	192.168.1.143	TCP	1484	80 → 61617 [ACK] Seq=4291 Ack=476 Win=42880 Len=1430 [TCP segment of
625	17.316310	192.168.1.143	74.125.225.26	TCP	54	61617 → 80 [ACK] Seq=476 Ack=5721 Win=65536 Len=0
626	17.321736	74.125.225.26	192.168.1.143	TCP	1484	80 → 61617 [ACK] Seq=5721 Ack=476 Win=42880 Len=1430 [TCP segment of
627	17.324366	74.125.225.26	192.168.1.143	TCP	1484	80 → 61617 [ACK] Seq=7151 Ack=476 Win=42880 Len=1430 [TCP segment of
628	17.324390	192.168.1.143	74.125.225.26	TCP	54	61617 → 80 [ACK] Seq=476 Ack=8581 Win=65536 Len=0
629	17.327099	74.125.225.26	192.168.1.143	TCP	1484	80 → 61617 [ACK] Seq=8581 Ack=476 Win=42880 Len=1430 [TCP segment of

> Frame 615: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0  
 > Ethernet II, Src: Cisco-Li\_cf:b7:29 (c8:d7:19:cf:b7:29), Dst: HonHaiPr\_ab:fd:31 (48:5a:b6:ab:fd:31)  
 > Internet Protocol Version 4, Src: 74.125.225.26, Dst: 192.168.1.143  
 ▾ Transmission Control Protocol, Src Port: 80, Dst Port: 61617, Seq: 0, Ack: 1, Len: 0

Source Port: 80  
 Destination Port: 61617  
 [Stream index: 61]  
 [TCP Segment Len: 0]  
 Sequence number: 0 (relative sequence number)  
 [Next sequence number: 0 (relative sequence number)]  
 Acknowledgment number: 1 (relative ack number)  
 1000 .... = Header Length: 32 bytes (8)

> Flags: 0x012 (SYN, ACK)

Window size value: 42900  
 [Calculated window size: 42900]  
 Checksum: 0xbf3f [unverified]  
 [Checksum Status: Unverified]  
 Urgent pointer: 0

> Options: (12 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted, No-Operation (NOP), Window scale  
 > [SEQ/ACK analysis]  
 > [Timestamps]

# Web logs

- **Web servers that log http Requests**
  - **URL (and parameters)**
  - **IP address of the requestor**
  - **cookie data (Identifies the User)**
  - **TOD**
  - **Bytes transferred**
  - **Time Taken**
- **e-Commerce sites are usually quite interested in statistical analysis of web site access patterns (i.e., page Hits)**
  - **Access pattern data can be used to optimize caching policies**



# Web logs

date	time	c-ip	cs-meth	cs-uri-stem	sc-bytes	cs-bytes	time-taken	cs(Referer)
5/10/2002	0:43:27	12.225.174.190	GET	/FAQsCPU.htm	32364	354	1640	http://www.google.com/search?q=cpu+100%25+i
5/10/2002	0:43:27	12.225.174.190	GET	/_themes/expeditn/exptextb.jpg	12746	330	890	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:30	12.225.174.190	GET	/Transparent+SOLDIERONDISK.gif	76014	332	2937	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:32	12.225.174.190	GET	/FAQsCPU_files/image001.jpg	98874	255	3797	-
5/10/2002	0:43:32	12.225.174.190	GET	/_derived/home_cmp_expeditn110_vbtn.gif	3130	339	765	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:33	12.225.174.190	GET	/FAQsCPU_files/image003.jpg	89260	255	3375	-
5/10/2002	0:43:33	12.225.174.190	GET	/_derived/up_cmp_expeditn110_vbtn.gif	3135	337	281	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:33	12.225.174.190	GET	/_derived/up_cmp_expeditn110_vbtn_a.gif	2638	339	235	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:33	12.225.174.190	GET	/_derived/whats_new.htm_cmp_expeditn110_vbtn.gif	3131	348	281	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:34	12.225.174.190	GET	/_derived/home_cmp_expeditn110_vbtn_a.gif	2646	341	203	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:34	12.225.174.190	GET	/_derived/whats_new.htm_cmp_expeditn110_vbtn_a.gif	2664	350	250	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:34	12.225.174.190	GET	/_derived/news.htm_cmp_expeditn110_vbtn_a.gif	3141	343	234	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:34	12.225.174.190	GET	/_derived/news.htm_cmp_expeditn110_vbtn_a.gif	2657	345	344	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:34	12.225.174.190	GET	/_derived/prod01.htm_cmp_expeditn110_vbtn.gif	3143	345	266	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:34	12.225.174.190	GET	/_derived/prod01.htm_cmp_expeditn110_vbtn_a.gif	2662	347	203	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:34	12.225.174.190	GET	/_derived/customer_support_home.htm_cmp_expeditn110_vbtn.gif	3126	360	281	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:35	12.225.174.190	GET	/_derived/customer_support_home.htm_cmp_expeditn110_vbtn_a.gif	2652	362	265	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:35	12.225.174.190	GET	/_derived/partners+overview.htm_cmp_expeditn110_vbtn.gif	3129	358	250	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:35	12.225.174.190	GET	/_derived/partners+overview.htm_cmp_expeditn110_vbtn_a.gif	2645	360	203	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:35	12.225.174.190	GET	/_derived/Contact_us_world.htm_cmp_expeditn110_vbtn.gif	3127	355	281	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:35	12.225.174.190	GET	/_derived/Contact_us_world.htm_cmp_expeditn110_vbtn_a.gif	2652	357	187	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:35	12.225.174.190	GET	/_derived/search.htm_cmp_expeditn110_vbtn.gif	3128	345	360	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:35	12.225.174.190	GET	/_derived/search.htm_cmp_expeditn110_vbtn_a.gif	2641	347	219	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:36	12.225.174.190	GET	/_derived/site_map.htm_cmp_expeditn110_vbtn_a.gif	2655	349	203	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:36	12.225.174.190	GET	/_derived/home_cmp_expeditn110_vbtn.gif	3130	339	344	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:36	12.225.174.190	GET	/_derived/site_map.htm_cmp_expeditn110_vbtn.gif	3123	347	921	http://www.demandtech.com/FAQsCPU.htm
5/10/2002	0:43:38	12.225.174.190	GET	/FAQsCP1.gif	12454	312	906	http://www.demandtech.com/FAQsCPU.htm



# Web logs

- **Aggregate to calculate arrival rate of requests**
- **Can we calculate Response time?**
  - **For the first Get Request to access a page**
  - **For subsequent Get Requests for the same page**
- **Can you correlate with other sources of Response time data?**
  - **e.g., network packet sniffers**

# Web log example

<b>date</b>	<b>time</b>	<b>c-ip</b>	<b>cs-method</b>	<b>cs-uri-stem</b>	<b>sc-status</b>	<b>sc-bytes</b>	<b>cs-bytes</b>	<b>time-taken</b>
5/16/2002	5:11:28	216.23.50.222	GET	/iccmFORUM/public/img/ICCMLink.gif	200	4150	414	516
5/16/2002	5:11:28	216.23.50.222	GET	/iccm.asp	200	42104	517	5015
5/16/2002	5:11:28	216.23.50.222	GET	/iccmFORUM/public/img/RedArrow.gif	200	1076	414	218
5/16/2002	5:11:28	216.23.50.222	GET	/iccmFORUM/public/img/Clear.gif	200	267	411	235
5/16/2002	5:11:29	216.23.50.222	GET	/iccmforum/public/Capacity/CPUResource/Wicks0101.gif	200	17854	432	875
5/16/2002	5:11:29	216.23.50.222	GET	/iccmforum/public/Capacity/CPUResource/Wicks0102.gif	200	17309	432	625
5/16/2002	5:11:29	216.23.50.222	GET	/iccmforum/public/Capacity/CPUResource/Wicks0103.gif	200	9680	432	375
5/16/2002	5:11:29	216.23.50.222	GET	/iccmforum/public/Capacity/CPUResource/Wicks0104.gif	200	17717	432	625
5/16/2002	5:11:30	216.23.50.222	GET	/iccmforum/public/Capacity/CPUResource/Wicks0106.gif	200	16351	432	500
5/16/2002	5:11:30	216.23.50.222	GET	/iccmforum/public/Capacity/CPUResource/Wicks0105.gif	200	16923	432	828
5/16/2002	5:11:30	216.23.50.222	GET	/iccmforum/public/Capacity/CPUResource/Wicks0107.gif	200	7820	432	313
5/16/2002	5:11:30	216.23.50.222	GET	/iccmFORUM/public/img/innovation.gif	200	3531	416	344
5/16/2002	5:11:30	216.23.50.222	GET	/iccmFORUM/public/img/Eval3x1.gif	200	2772	413	219
5/16/2002	5:11:30	216.23.50.222	GET	/iccmFORUM/public/img/BlueArrow.gif	200	297	415	328
5/16/2002	5:11:30	216.23.50.222	GET	/iccmFORUM/public/img/BlueArrowLeft.gif	200	1077	419	343

# GTF (z/OS diagnostic facility)

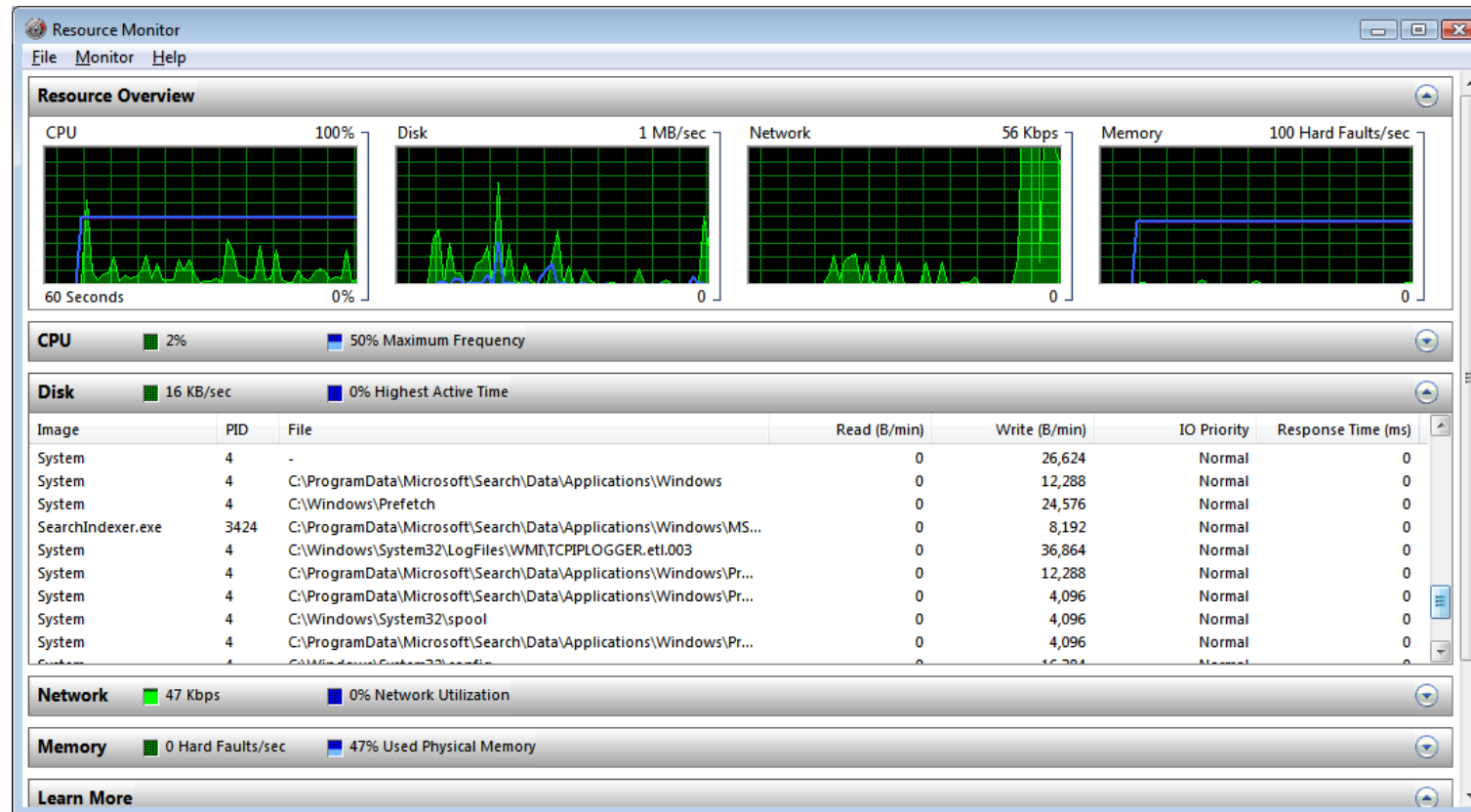
- **Exploits Monitor Call hardware instruction, which are embedded in various system functions**
  - **When Tracing is enabled, the MC instruction branches to the tracing application**
  - **When tracing is disabled, the MC instruction is a NOP**
- **e.g., CCWTrace (which may or may not include data)**
  - **applications include optimizing dataset placement on a volume to minimize head movement or understanding volume access patterns to model the impact of caching**
- **e.g., program load, catalog requests, etc.**
- **Trace-driven simulations**

# DTrace

- **Kernel and User trace facility originally developed for Sun Solaris**
  - **which became OpenSolaris**
- **ported to Mac OS X**
  - **wrapped with the Apple Instruments GUI**
  - **see Apple [Instruments User Guide](#)**
- **supports predicates for conditional control of trace data output**
  - **see “DTrace by Example” [tutorial](#)**

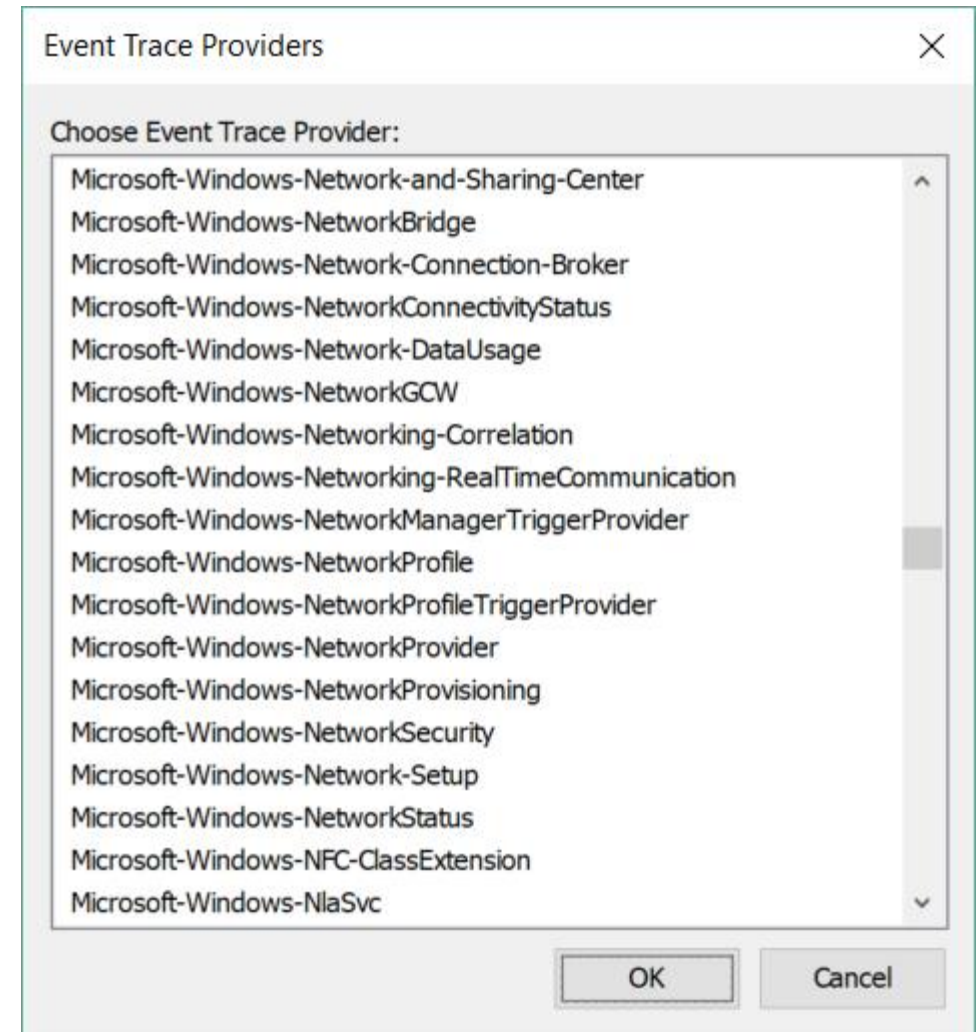
# Event Tracing for Windows (ETW)

- Windows Kernel trace: low level diagnostic trace



# Windows kernel trace facility

- **Secure kernel trace facility**
- **Architected for high volume**
- **Provider : Listener model**
- **Many, many Providers**
  - **IIS, Active Directory, etc.**
- **Very little documentation on what Events they Provide**
- **CallStacks available for many Events**



# Windows ETW trace facility

- **Provider : Listener model**
- **Instruments the Windows OS kernel**
- **e.g., the CSwitch event**

(see [CSwitch class](#) documentation)

```
[EventType{36}, EventTypeName{"CSwitch"}]  
class CSwitch : Thread_V2  
{  
    uint32 NewThreadId;  
    uint32 OldThreadId;  
    sint8  NewThreadPriority;  
    sint8  OldThreadPriority;  
    uint8  PreviousCState;  
    sint8  SpareByte;  
    sint8  OldThreadWaitReason;  
    sint8  OldThreadWaitMode;  
    sint8  OldThreadState;  
    sint8  OldThreadWaitIdealProcessor;  
    uint32 NewThreadWaitTime;  
    uint32 Reserved;  
};
```

# Windows ETW trace facility

- **Windows is awash in raw ETW data**
  - **Filtering predicates not supported**
  - **Inconsistent data models; not RESTful: many Events require links to other Events for full context**
  - **Generic visualizations**
  - **Proposed applying basic E:R **semantics** to this unstructured stream of time-series data:**
    - **Sequencing (Begin, Step, End)**
    - **Parent-child relationships**
    - **Asynch: fork; block; join**
    - **resource or state Switch (*prev*, *current*)**



# SQL Server xEvent trace facility

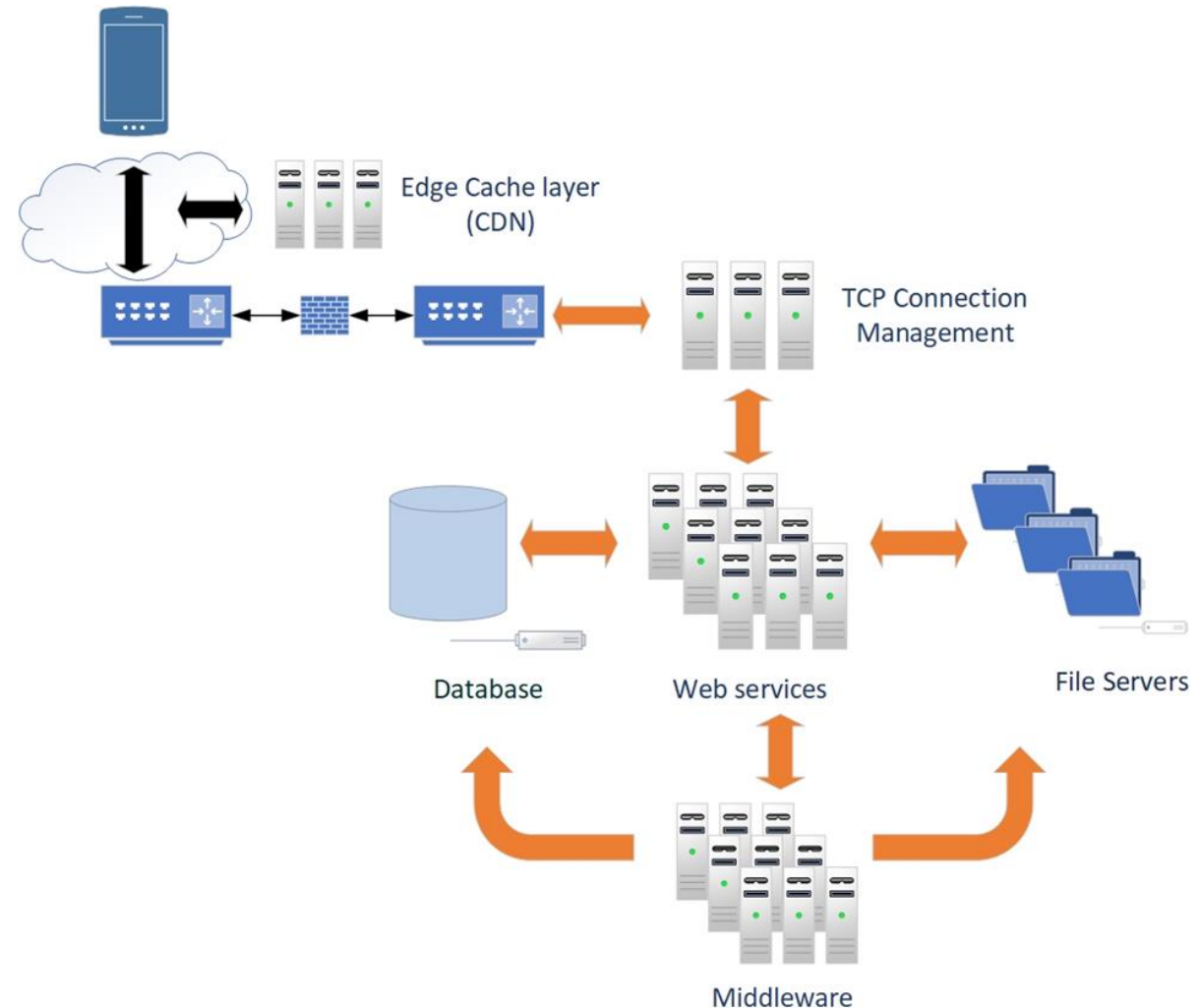
- Filtering predicates *are* supported
  - replaced an earlier diagnostic SQL Trace facility that did not support Filter predicates
- Data payloads up to 1 MB per trace record
  - which meant ETW infrastructure could not be used
- Sample scenarios:
  1. Find the Objects That Have the Most Locks Taken on Them
  2. Which Queries Are Holding Locks

# Using your measurements

- **Understanding and interpreting what is being measured & how it is being measured**
  - **Queuing Theory** suggests how key measurements are likely to be related
  - **Analytic and simulation modeling**
- **Management Reporting**
  - **Service levels (i.e., application response time)**
  - **Exceptions/Anomalies (based on ROTs or Statistical Quality Control techniques)**
- **Forecasting (proactive capacity planning)**
- **Performance Engineering**

# Application Performance Monitoring

- **End-to-end tracing**
  - **Correlate measurements across multiples layers in the application**
    - including visibility into the underlying Framework
    - Data layer  $\Rightarrow$  back-end DBMS
  - **Containers (i.e., Docker)**
  - **Multiple machines**
  - **Multiple platforms**



e.g., Dynatrace

Search your environment...

Home > Services > .NET easyTravel

## IIS .NET .NET\_easyTravel\_x64:9000

Details ...

Properties

0 Applications  
1 Service

20.5/min Dynamic requests  
225/min Resource requests

15.3 ms Response time  
2.46 ms Resource time  
0 % Failure rate

0 Databases  
8 Services

2 IIS

Response time  
Resource time  
Failure rate  
CPU consumption

last 2 hours Show slowest 10 % in chart

Analysis range: Today, 10:46 - 11:16  
Click chart to select analysis timeframe.

View response time hotspots

3 Problems in last 72 hours watched

### Most time consuming requests

Request	Response time	Requests
Images	2.31ms	334/min

### Slowest requests

Request	Response time	Requests
/Booking	136ms	4.43/min
/Journey	123ms	4.93/min
/Report	110ms	5.67/min

Show all requests

1 Event in last 72 hours

Search your environment...

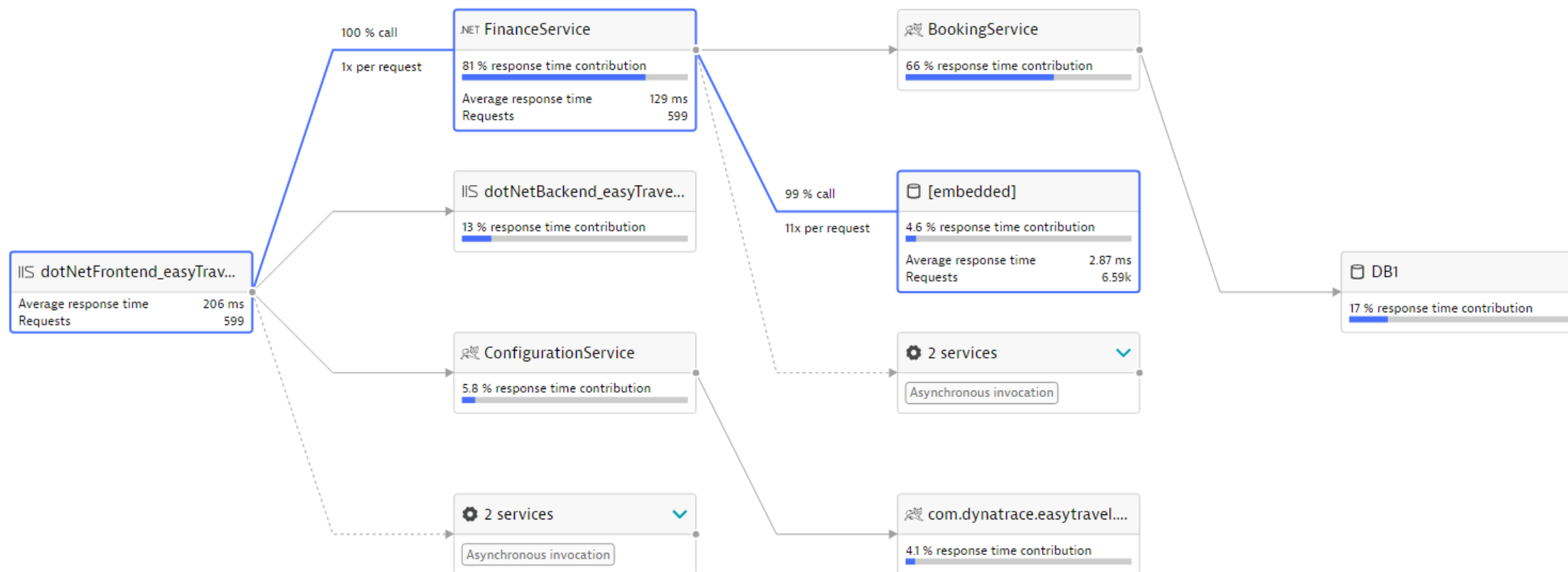
Home > Services > dotNetFrontend\_easyTravel\_x64:9000 > Details > Web request details > Service flow

Showing service flow of /Booking requests to dotNetFrontend\_easyTravel\_x64:9000

today, 12:09 - 14:09

99 % of requests to FinanceService call [embedded] (averaging 11 calls per request)

e.g., Dynatrace



# Interpreting measurement data

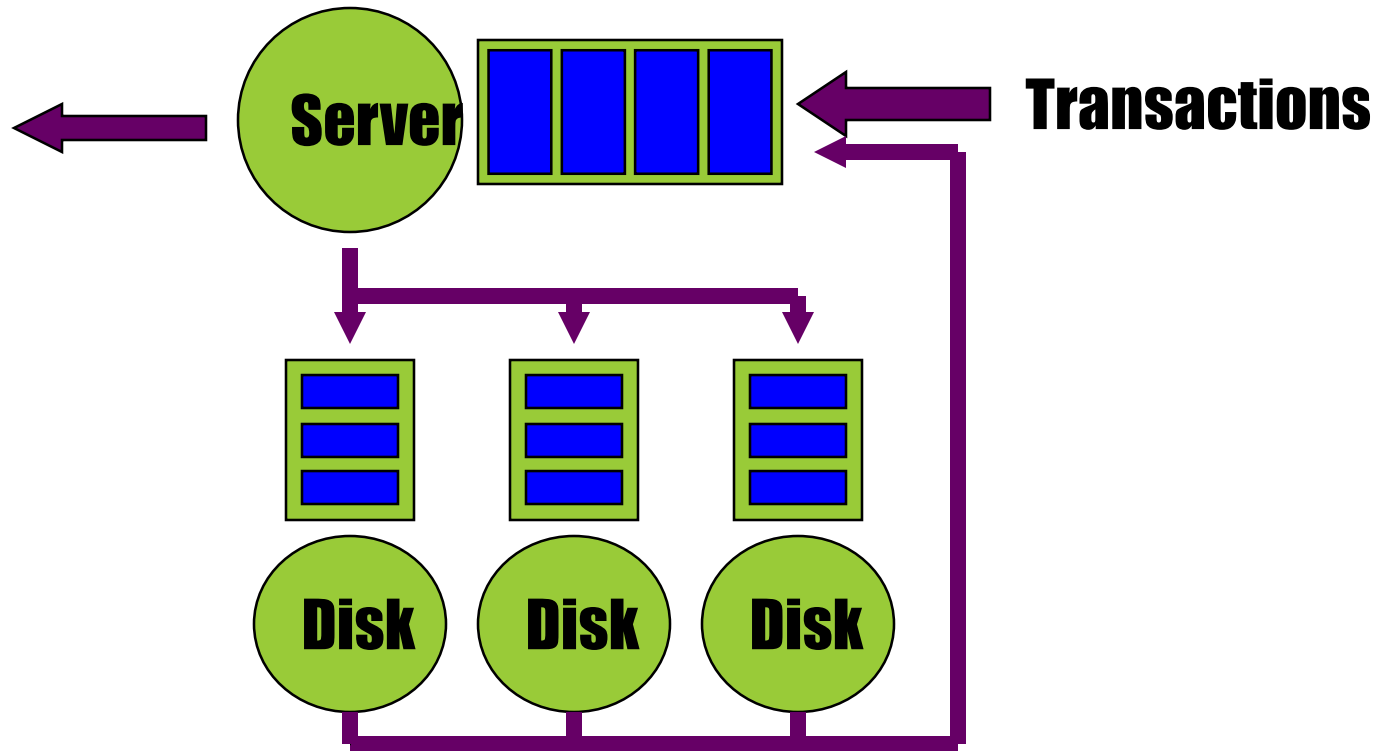
- **Need to understand what is being measured & how it is being measured**
  - **Workload arrival rate**
  - **Resource Utilization**
  - **Queuing**
  - **Service Time**
  - **Response Time**
- **Data Analysis!**

# Interpreting measurement data

- Find meaningful correlations *between* measurements
  - e.g., physical memory utilization vs. paging
    - V:R Ratio
  - **Careful: many performance measures are auto-correlated!**
    - e.g., % Processor Time, % User Time
  - **Association is not Causation : see causal inference**
- **Queueing Theory** suggests how these measurements are likely to be related
  - Statistical significance is not sufficient evidence
  - Instead, formulate a model/hypothesis and then test its plausibility

# Elements of queuing theory

- **Classic network queuing model:**





# Elements of queuing theory

- **Fundamental relationships:**

- **Response time,  $W = W_s + W_q$**
- **utilization =  $\lambda * W_s$**

- **Little's Law:**

$$L = \lambda * W$$

**# in system = arrival rate \* Response time**

- **Little's equivalence relationship is used frequently to minimize measurement overhead:**
  - **measure two elements, calculate the third**

# Elements of queuing theory

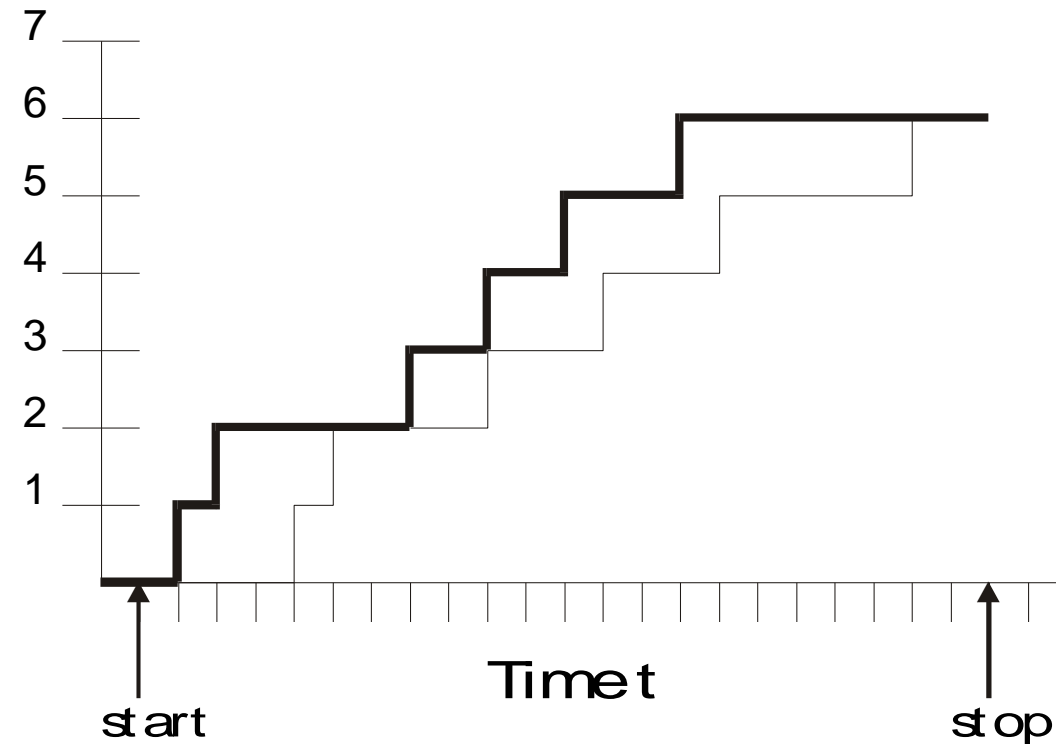
- Little's Law:**

$$L = \lambda * W$$

# in system =  $\lambda$  \* Response time

**Equilibrium Assumption:**

**Arrivals = Completions**



# Elements of queuing theory

## • Little's Law:

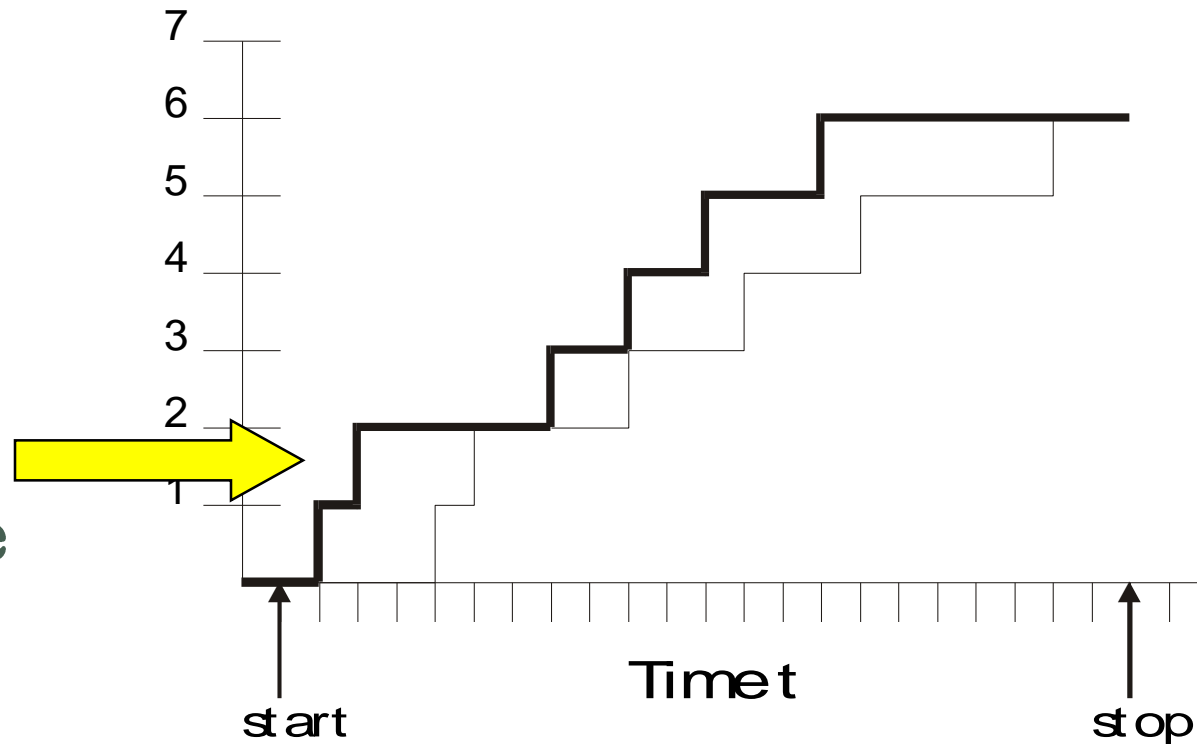
$$L = \lambda * W$$

# in system =  $\lambda$  \* Response time

Assume Arrivals = Completions

Calculate  $P$ , accumulated time in the system, then

$$L = P/T \text{ and } R = P/C$$



# Elements of queuing theory

## • Little's Law:

$$L = \lambda * W$$

# in system =  $\lambda$  \* Response time

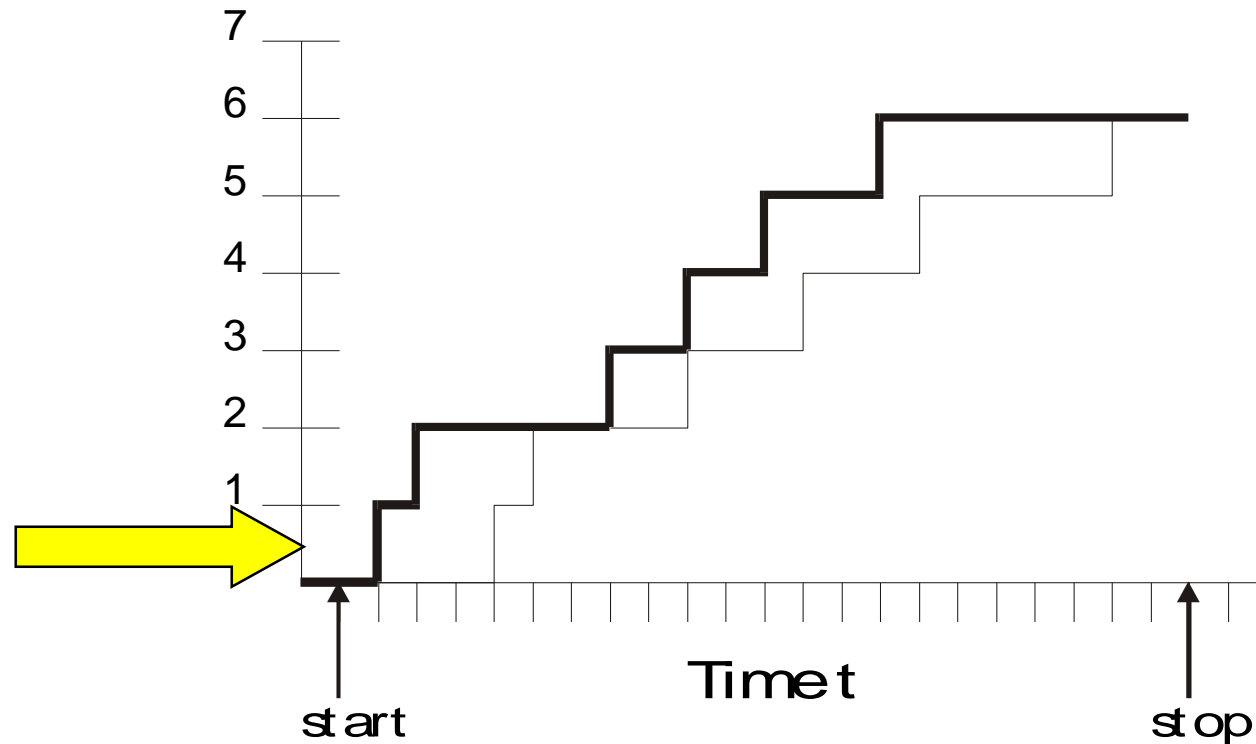
Assume Arrivals = Completions

Calculate **P**, accumulated time in the system, then

$$L = P/T \text{ and } R = P/C$$

And by substitution:

$$L = \frac{P}{T} = \frac{C}{T} * \frac{P}{C} = \lambda * R$$



# Elements of queuing theory

- **Simple analytic queuing models have simple equations that can be easily solved**
- **Realistic models have complex equations that are *impossible* to solve**
  - **Heavy traffic approximation**
  - **Approximate methods**
- **Complex models can be solved using Monte Carlo simulation techniques**

# Elements of queuing theory

- **Notation:**
  - **arrival rate distribution**
  - **service time distribution**
  - **#servers**
- ***M/M/1***
  - **exponential arrival rate distribution**
  - **exponential service time distribution**
  - **1 server**
  - **memoryless**
  - **arrivals selected from an infinite population**

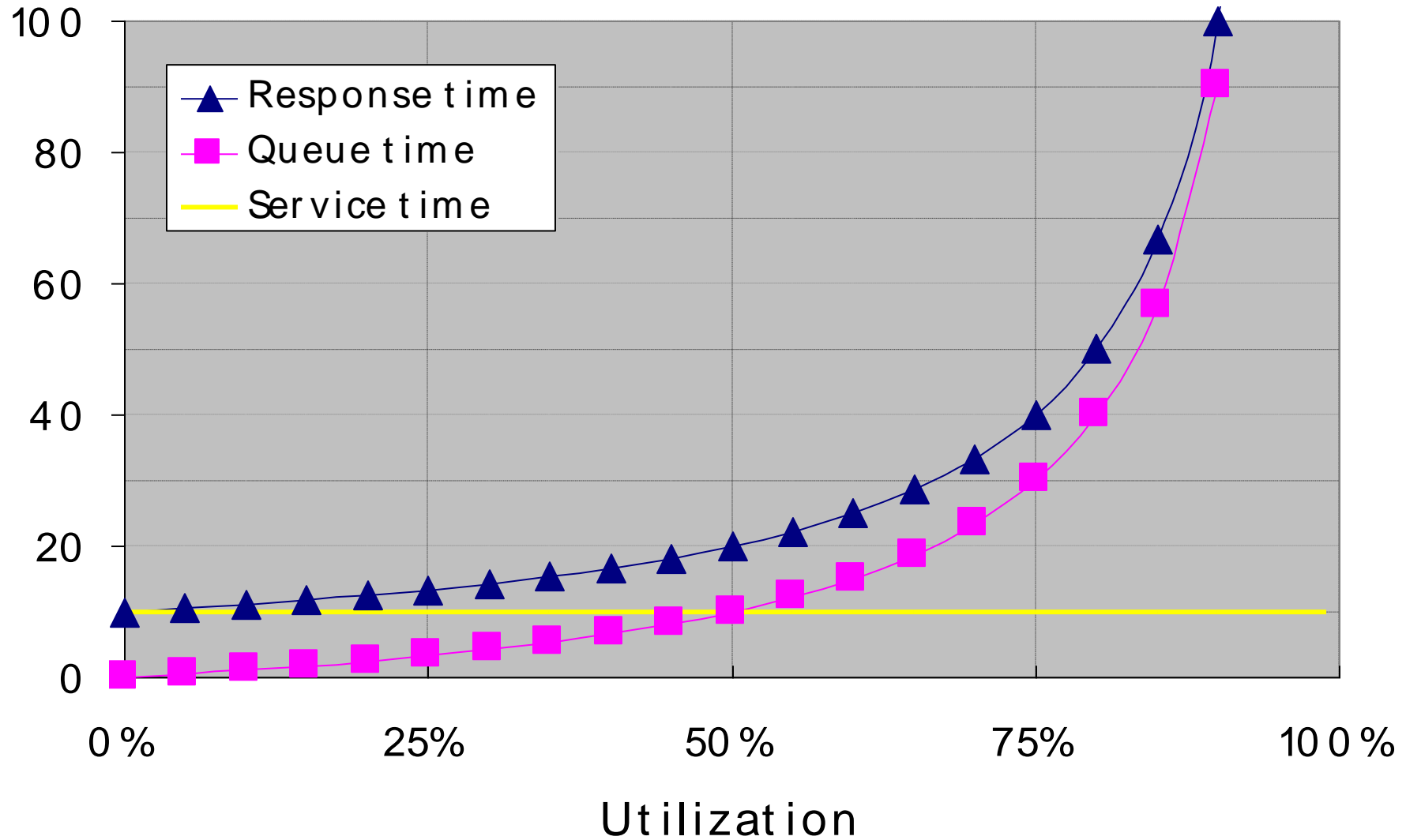
# Elements of queuing theory

- ***M/M/1***
  - **Calculate queue time based on service time and utilization:**

$$W_q = (W_s * u) / (1 - u)$$

# M/M/1

(assumes 10 ms. Service time)



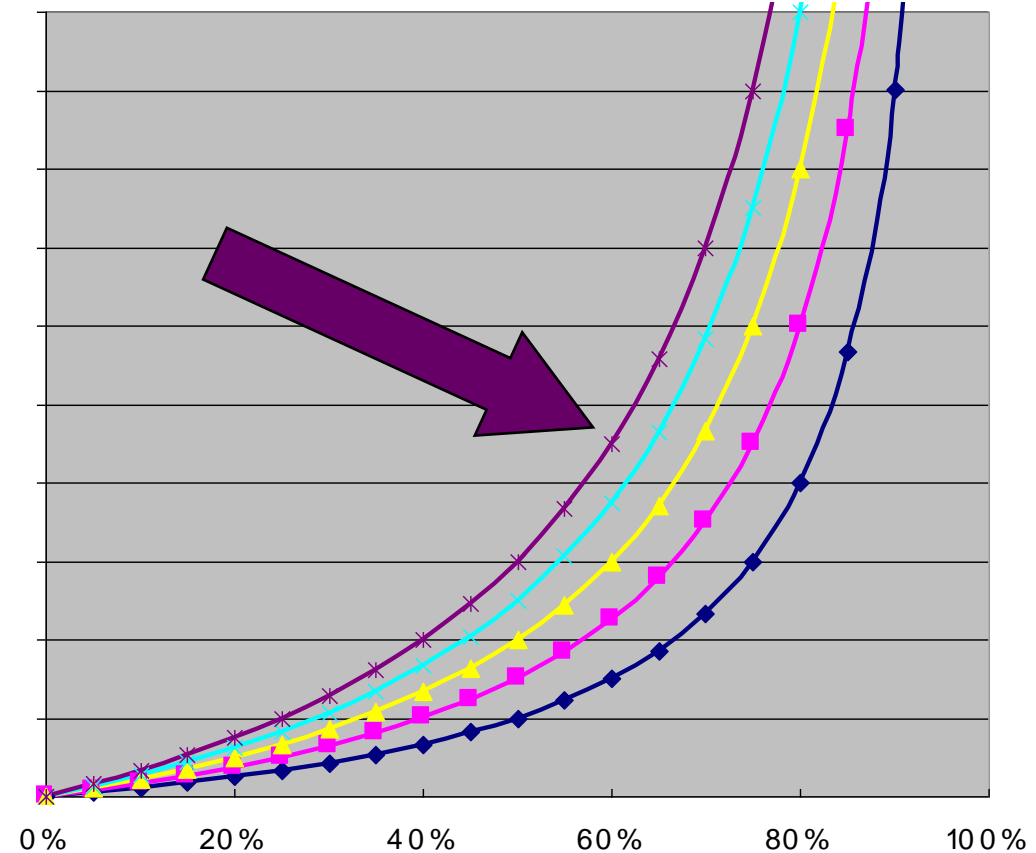


# Analytic and simulation modeling

- **Queuing systems mimic the actual operation of real systems and networks**
  - They rarely degrade gradually!
  - Gradual *quantitative* change can become *qualitative* change -- which may defeat change control!
- **Performance tuning methodology:**
  - Systematically identify and eliminate *bottlenecks*
  - Operational definition of a bottleneck: the resource with the *fastest* growing queue

# Bottleneck analysis

- Don't expect complex systems to scale *linearly*
- Successively identify and eliminate *bottlenecks*

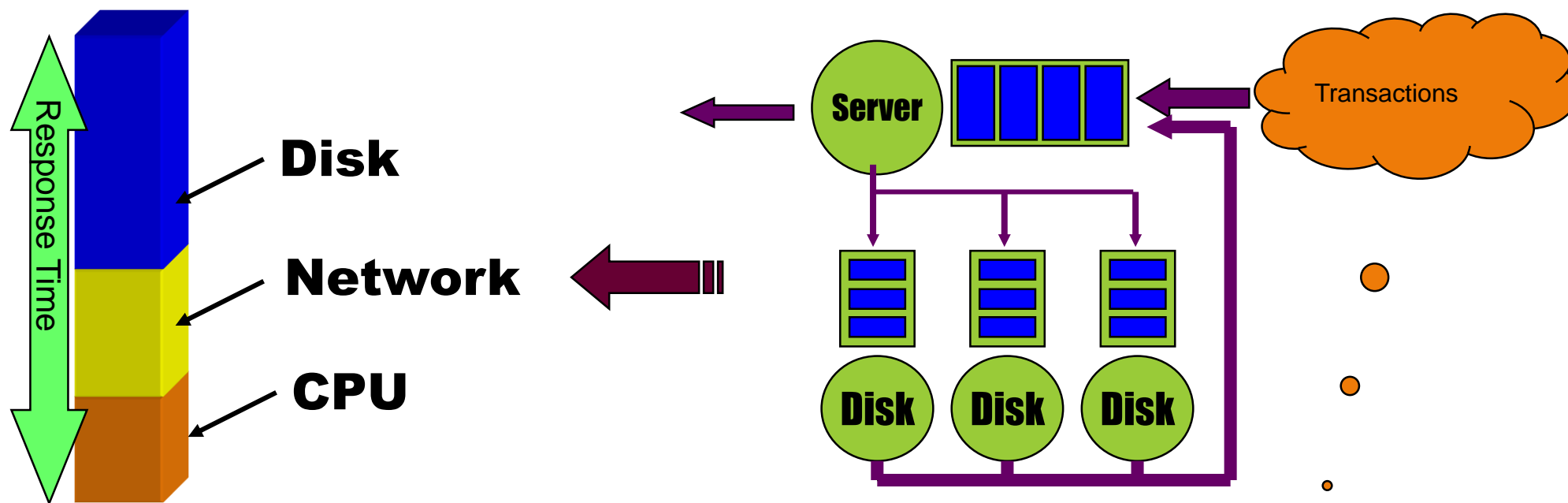


# Bottleneck analysis

- **Complications:**
  - **When the queuing discipline is *unfair!***
    - e.g., priority-based queuing
  - **Load-dependent servers**
    - **When throughput or service time at a server is *not* constant with respect to utilization**
    - e.g.,
      - **Ethernet collisions**
      - **SCSI command queuing**

# Decomposition

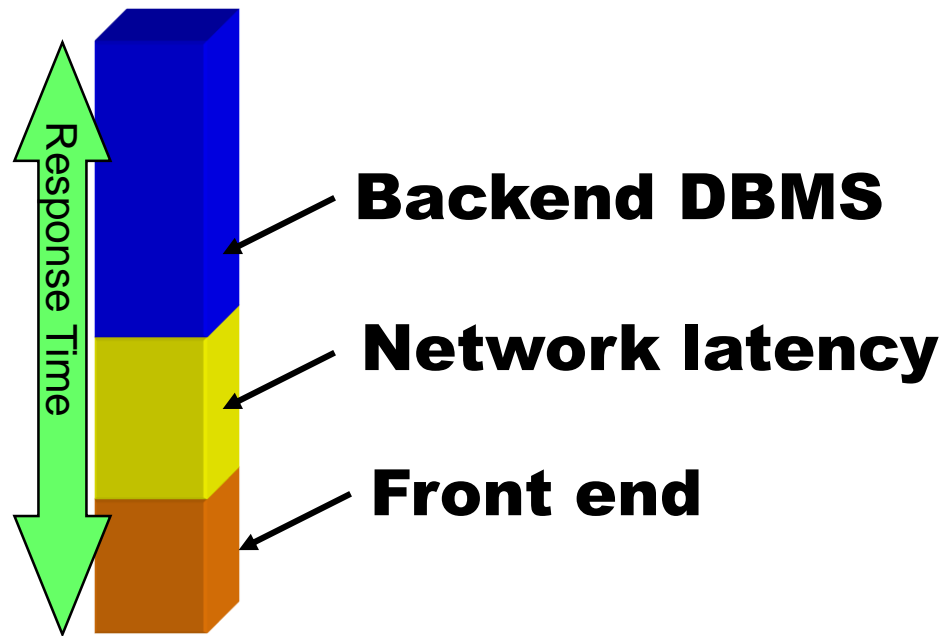
- Reduce application response time into its component parts by *resource*



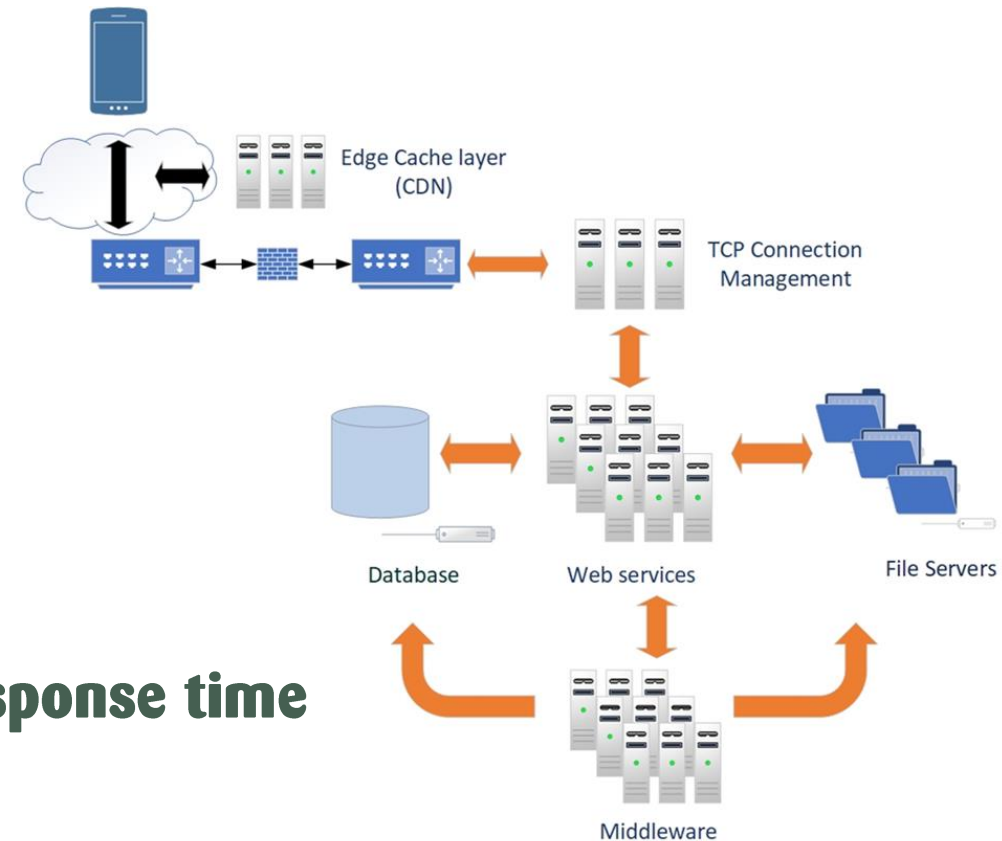
- Focus on the resource that plays the *largest* part in response time

# Decomposition

- Reduce application response time into its component parts by *tier/layer*



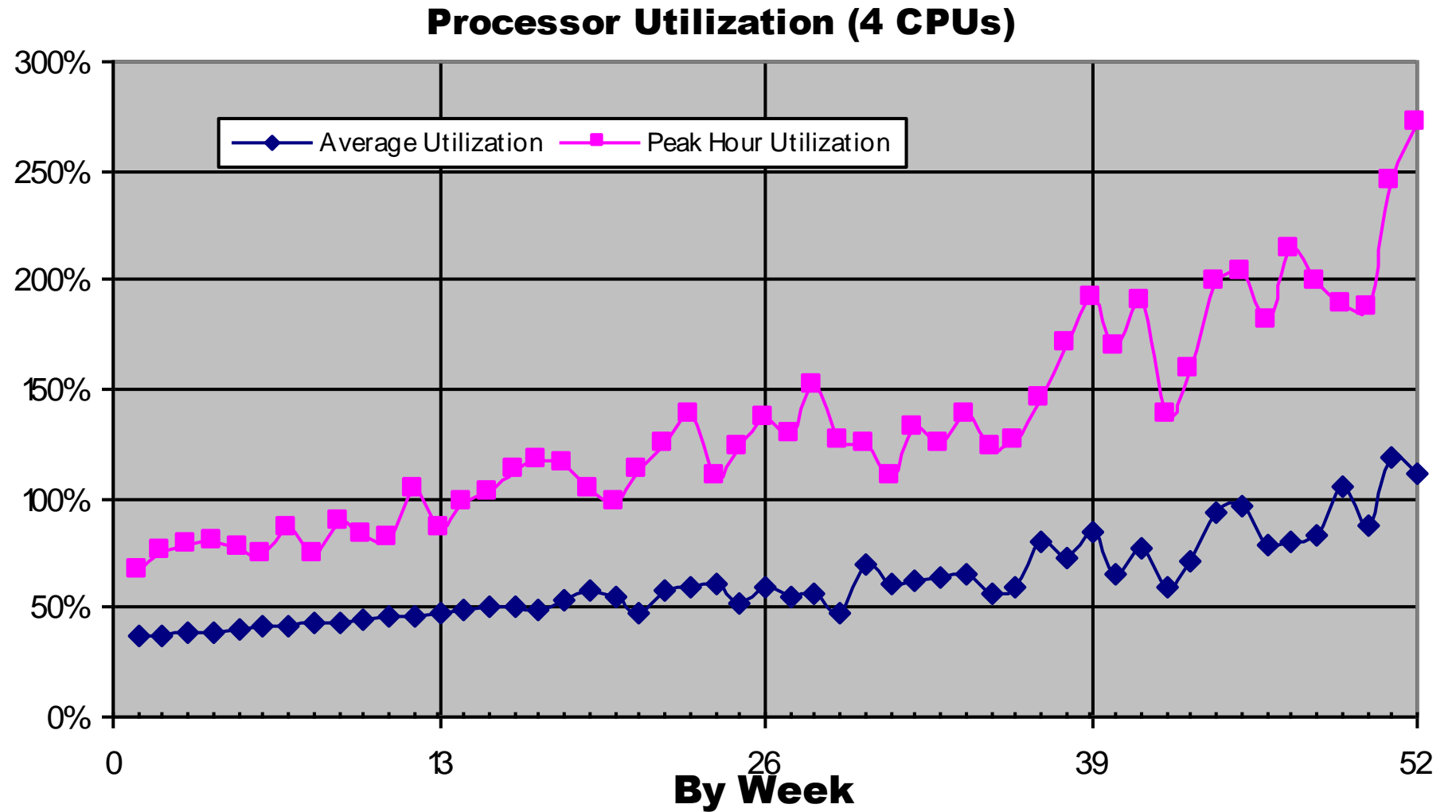
- Focus on the layer that plays the *largest* part in response time



# Forecasting

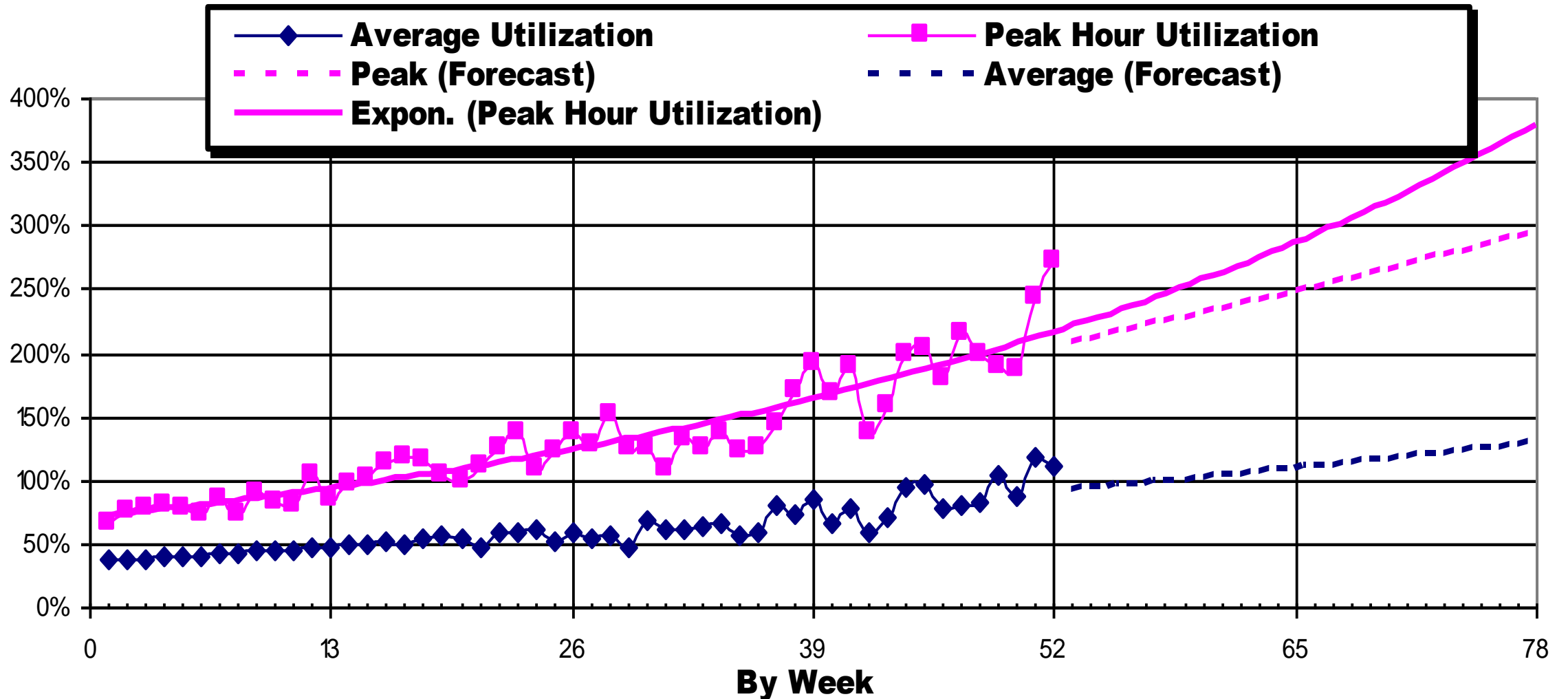
- **Calculate resource utilization per Request**
  - **e.g., CPU busy/Request, i.e.,  $s = U / \lambda$  where  $\lambda =$** 
    - **HTTP Requests/sec**
    - **Web Service Get Requests/sec**
    - **TCP Connections Active**
  - **Projection using statistical techniques**
    - **linear regression**
    - **non-linear regression**
  - **Peak/average ratio**

# Forecasting



# Forecasting

## Processor Utilization (4 CPUs)





# Performance Engineering

- **How to add performance considerations to the design and development process?**
  - **Build in performance considerations every step of the application development lifecycle**
    - **Design, Develop, Test, Deploy, Revise, etc.**
  - **Requires an analytic approach (e.g., queuing theory)**
    - **Practitioners too often conceive of performance engineering as a set of tips (best practices) for building better performing applications**

# References

- **Bryan Cantrill, “Hidden in Plain Sight”, *ACM Queue*, Feb. 2006.**
- **Neil Gunther, *The Practical Performance Analyst*, 1998.**
- **Daniel Menascé, *et. al.*, *Performance By Design*, 2004.**
- **Friedman, *Microsoft Windows Server 2003 Performance Guide*, 2005.**