




# CS1632, LECTURE 11: PERFORMANCE TESTING

Bill Laboon



BEFORE WE TALK PERFORMANCE  
TESTING...

LET'S ASK: WHAT DO WE MEAN BY  
**PERFORMANCE?**



“I shall not today attempt further to define the kinds of material I understand to be embraced within that shorthand description, and perhaps I could never succeed in intelligibly doing so. But ***I know it when I see it.***”

-Potter Stewart, US Supreme Court Justice  
Concurrence in *Jacobellis v. Ohio*



BUT...

- We *can* talk about what kinds of performance we're interested in, and set specific goals for the system under test.
- A video game console will have very different performance requirements than a weather forecasting supercomputer.

# PERFORMANCE INDICATORS

- Quantitative measures of the performance of a system under test
- Examples:
  - How long does it take to respond to a button press?
  - How many users can access the system at one time?
  - How long can the system go without a failure?
  - How much CPU does a standard query on the database takeup?
  - How big is the program in megabytes?
  - How fast the program calculate some function?

# KINDS OF PERFORMANCE INDICATORS

- Service-Oriented
- Efficiency-Oriented



## SERVICE-ORIENTED

Service-oriented indicators measure how well a system is providing a service to the users. They are measured from a specific user's point of view.



# EFFICIENCY-ORIENTED

Efficiency-oriented indicators measure how well the system makes use of the computational resources available to it. This is looking at the performance from a more developmental perspective.





# CATEGORIES OF SERVICE-ORIENTED INDICATORS

- Availability - How available is the system to the user?  
What percentage of the time can they access it?
- Response Time - How quickly does the system  
respond to user input?



## CATEGORIES OF EFFICIENCY-ORIENTED INDICATORS

- Throughput - How many events can occur and be processed in a given amount of time?
- Utilization - What percentage or absolute amount of computing resources are used to perform a task?



# TESTING PERFORMANCE

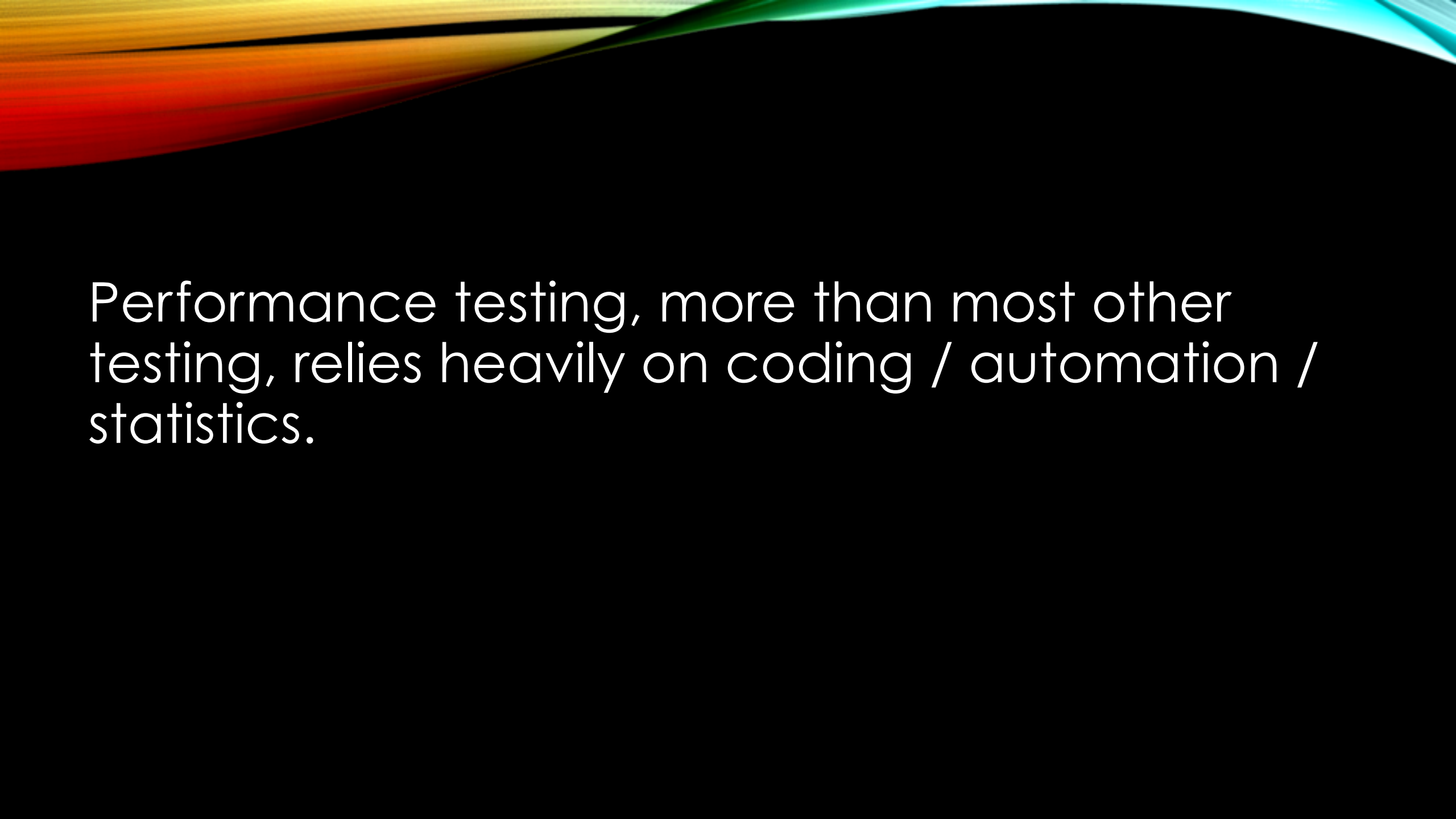
- In order to test performance, you should have **performance targets** - quantitative values that the **performance indicators** should reach.
- Performance targets may be assigned to a subset of the most important performance indicators, **called key performance indicators (KPIs)**.

# PERFORMANCE THRESHOLDS

- **Performance thresholds** are the absolute minimum performance level a system can reach and be considered production-ready.
- These may be used in addition to performance targets as a “bare minimum” to reach, although not the desired target.

# EXAMPLE

- You are developing a web application which is expecting a relatively constant 20 hits per second.
- A key performance indicator (KPI) might be response time, with a performance threshold of three seconds mean time to respond and a performance target of one second.
- Ideally, your stakeholders would like sub-second response time, but they would be satisfied as long as the response time is below three seconds.



Performance testing, more than most other testing, relies heavily on coding / automation / statistics.

# STATISTICS? WHY?

- You shouldn't really trust a single result in performance testing, especially for response times. You should always be trying multiple times and discussing a mean value, maximum value, minimum values, etc.
- There are so many variables in a single test run (other processes taking up CPU, pipelining issues, memory swaps, VM startup times, etc.) that a single test run is almost worthless.



# HUMAN ERROR

- Minor issues with human error can cause massive changes in performance results.



# SHE BLINDED ME WITH SCIENCE

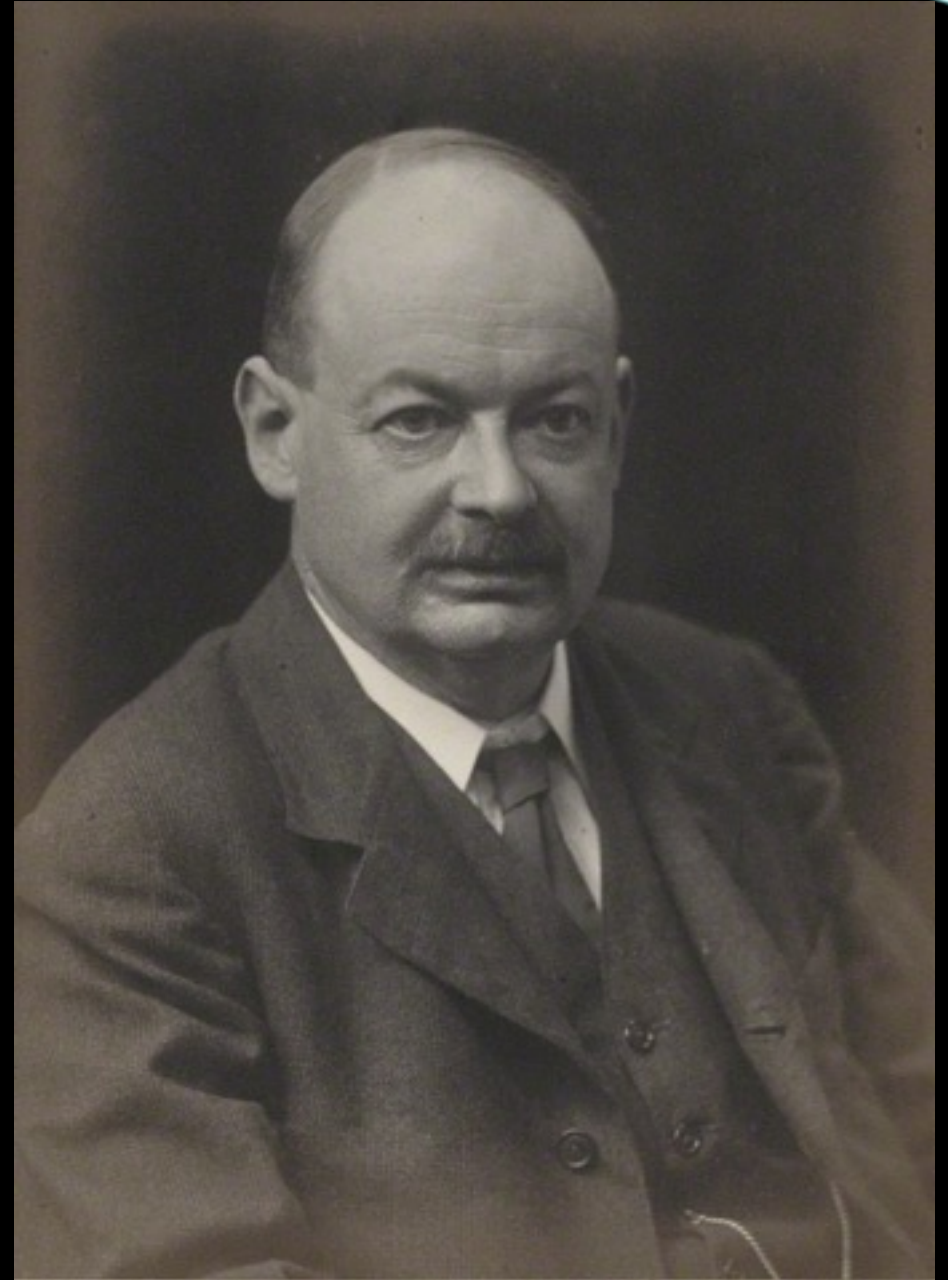
- Performance testing is more like a science than other kinds of testing.
- You want to run multiple experiments and eliminate as many possible variables OTHER THAN THE CODE UNDER TEST as you can.
- Don't run version 1 on a ChromeBook and version 2 on a Cray supercomputer and talk about the massive speed improvement.

# KINDS OF EVENTS TO TEST FOR RESPONSE TIME

- Time for calculation to take place
- Time for character to appear on screen
- Time for image to appear
- Time to download
- Time for server response
- Time for page to load
- Time for code to execute

# WHAT IS TIME?

By Walter Stoneman - <http://www.npg.org.uk/collections/search/portraitLarge/mw124669/John-McTaggart-Ellis-McTaggart>, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=22142173>




# ASSUMING THAT TIME EXISTS...

- user time: Amount of time user code executes
- system time: Amount of time kernel code executes
- total time : user time + system time
- real time: “Actual” amount of time taken (wall clock time)

# EXAMPLE

- time command in Unix
  - time java Foo
  - time curl <http://www.example.com>
  - time ls -l
- Windows PowerShell has something similar
  - Measure-Command { java Foo -wait }

- 
- Users almost always care about real (“wall clock”) time.
  - You usually use total, user, and system time to help developers, not as KPIs in and of themselves.

# MEASURING AVAILABILITY PERFORMANCE INDICATORS

- Availability - often referred to as uptime - how often can a user access the system when they expect to be able to do so?
- Often referred to as a SLA (service-level agreement).
  - “I am a web host. I guarantee you that you and your users will be able to access your server 99% of the time in a given month.”

# NINES

- Uptime is often expressed in an abbreviated form as 9's (e.g. 3 nines, 5 nines etc)
- Refers to how many 9's start out the percentage of time available
  - 1 nine: 90% available (36.5 days of downtime per year)
  - 2 nines: 99% available (3.65 days of downtime per year)
  - 3 nines: 99.9% available (8.76 hours of downtime per year)
  - 4 nines: 99.99% available (52.56 minutes of downtime per year)
  - 5 nines: 99.999% available (5.26 minutes of downtime per year)
  - 6 nines: 99.9999% available (31.5 seconds of downtime per year)
  - 9 nines: 99.99999999% available (31.5 ms of downtime per year)





# HOW TO TEST?

- Often difficult – most managers won't let you run a few “test years” before deploying it for real
- Modeling the system and estimating uptime is the best (feasible) approach

# DETERMINE VALUES FOR MODEL WITH LOAD TESTING

- **Load testing** - how many concurrent users and/or work can the system handle?
- Kinds of load testing:
  - **Baseline Test** - A bare minimum amount of use, to provide a baseline
  - **Soak / Stability Test** - Leave it running for an extended period of time, usually at low levels of usage
  - **Stress Test** - High levels of activity



# REALITY

- For true availability numbers, also need to determine:
  - Likelihood of hardware failure
  - Likelihood of program-ending bugs
  - Planned maintenance
  - etc.

# SIC TRANSIT GLORIA MUNDI

- Even with all this work, things go wrong
- Many major service providers, such as Microsoft Azure and Amazon Web Services, “breach” their SLAs in a given month (e.g. their servers are guaranteed to be available 99.9% of the time, but are down for two days due to a power surge, meaning they were only available 93.6% of that month)
- Usually, money is refunded automatically

# DEVELOPING THE SERVICE-ORIENTED TEST PLAN

- Think from a user's perspective!
  - How fast do I expect this to be?
  - What things matter to me, speedwise?
  - How often do I expect this to be available?
  - Are large variances in response time allowed?

# DETERMINE KPIS, TARGETS, AND THRESHOLDS

- Example:
  - Average page load time – Target: less than two seconds, Threshold: less than five seconds
  - Max page load time – Target: less than five seconds, Threshold: less than ten seconds
  - Availability of system: Target: greater than 99.9%, threshold: greater than 99%

# CATEGORIES OF PERFORMANCE TESTING

- Service-Oriented
  - Availability
  - Response Time
- Efficiency-Oriented
  - Throughput
  - Resource utilization



# EFFICIENCY-ORIENTED PERFORMANCE TESTING

Measures how well an application takes advantage of the computational resources available.



# WHY DO EFFICIENCY-ORIENTED PERFORMANCE TESTING?

1. More granular than service-oriented testing
2. Easier to pin down bottlenecks
3. Possible to determine if problem can be solved by hardware modification / scaling / upgrading / etc.
4. Talk in a language developers can understand
5. Easier to get large amounts of data

# EXAMPLE

Rent-A-Cat has added a RESTful API showing which cats are available to rent. However, service-oriented testing has shown that it takes five seconds (minimum) to respond to `/cats/list` (which shows a sorted list of all available cats).

After some testing, you see that after being accessed, network usage is 1%, disk usage is 3%, memory usage is steady, but the CPU is pegged at 99% for five seconds.

Where would you look for solutions to this issue?

# POSSIBLE ISSUES / AMELIORATIONS

1. Just need faster hardware – time to migrate away from the Commodore 64
2. Bad / lengthy HTML – generate pages better
3. Cats sorted with Bogosort – use better sorting algorithm
4. Lots of malloc/free calls – tune garbage collector, modify code to create fewer objects
5. Everything running on single machine - Spread work to other cores/processors
6. Just too popular - Cache listings

# BENEFITS OF EFFICIENCY-ORIENTED TESTS

- The service-oriented test can tell us the general idea of what is wrong, but to fix it, we often need to undertake efficiency-oriented testing, to determine the actual source of the problem.
- If response time was 40 milliseconds, nobody would have cared, and there would have been little need for the efficiency-oriented testing.



“PREMATURE OPTIMIZATION IS THE  
ROOT OF ALL EVIL” -KNUTH

Oftentimes, it makes more sense to do service-oriented testing first, then drill down with efficiency-oriented tests to find out where problems lie later.



# THROUGHPUT TESTING

- What is throughput testing?
- Measuring the maximum number of events possible in a given timeframe.



## EXAMPLES

- You have a router, and you would like to know how many packets it can handle in one second.
- You have a web server, you'd like to know how many static pages of a given size it can serve in one minute.
- You are running a video game server, you'd like to know how many users can play simultaneously.

# HOW IS THAT DIFFERENT FROM SERVICE-ORIENTED TESTING?

1. A given user doesn't care about the number of users who can access a system, just about what it means for them
2. Often more granular (users don't care about, e.g., packets)



# LOAD TESTING

- Variations on load testing (see slides from last lecture) can be used to test throughput
  - Increase number of events until system crashes
  - Increase number of events until response time falls below threshold
  - Etc.
- Perspective is of system, not the user

# LOAD TESTING

- Load testing can also be used for subsystems
  - How many entries can be made in database before events start being queued?
  - How many calculations can occur in one minute?
  - How many threads can be started in a given timeframe?
  - How many images can be written to disk?
  - How many videos can be compressed?
  - Etc.



# MEASURING RESOURCE UTILIZATION

- *You need tools for this*
  - Unless you can tell by the sound of your fan exactly how many operations your program is running on the CPU

# TOOLS

- General purpose
  - Windows Systems – Task Manager, perfmon
  - OS X - Activity Monitor or Instruments, top
  - Unix systems - top, iostat, sar
- Program-Specific Profilers
  - JProfiler
  - VisualVM
  - gprof
  - Many, many more



# A VERY SIMPLE EFFICIENCY TEST

Watch CPU usage while you do something



# KEY RESOURCES TO WATCH

- CPU Usage
- Threads
- Memory
- Virtual Memory
- Disk I/O
- Network I/O

# YOU CAN GET MORE SPECIFIC

- Disk cache misses
- File flushes
- # Destination Unreachable message
- IPv6 Fragments Received/Sec
- Outbound Network Packets discarded
- # Print Queue "Out of Paper" messages
- ACK msgs received by Distributed Routing Table


# RESOURCE UTILIZATION MONITORING OF THIS KIND IS VERY BROAD

- Lots of memory being taken up...
  - ...but by what objects / classes / data?
- Lots of CPU being taken up...
  - ...but by what methods / functions?
- Lots of packets sent...
  - but why? And what's in them?



# MORE SPECIALIZED TOOLS

- Protocol analyzers
  - e.g., Wireshark or tcpdump
  - See exactly what packets are being sent/received
- Profilers
  - See exactly what is in memory
  - What methods are being called and how often
  - What objects/classes have been loaded

- 
- There's a big jump between "is our app slow?" and "we are leaking memory by never removing ConnectionCounter objects, causing more swaps and GC as a percentage of CPU time, thus causing response time to increase monotonically and exponentially in relationship to uptime."