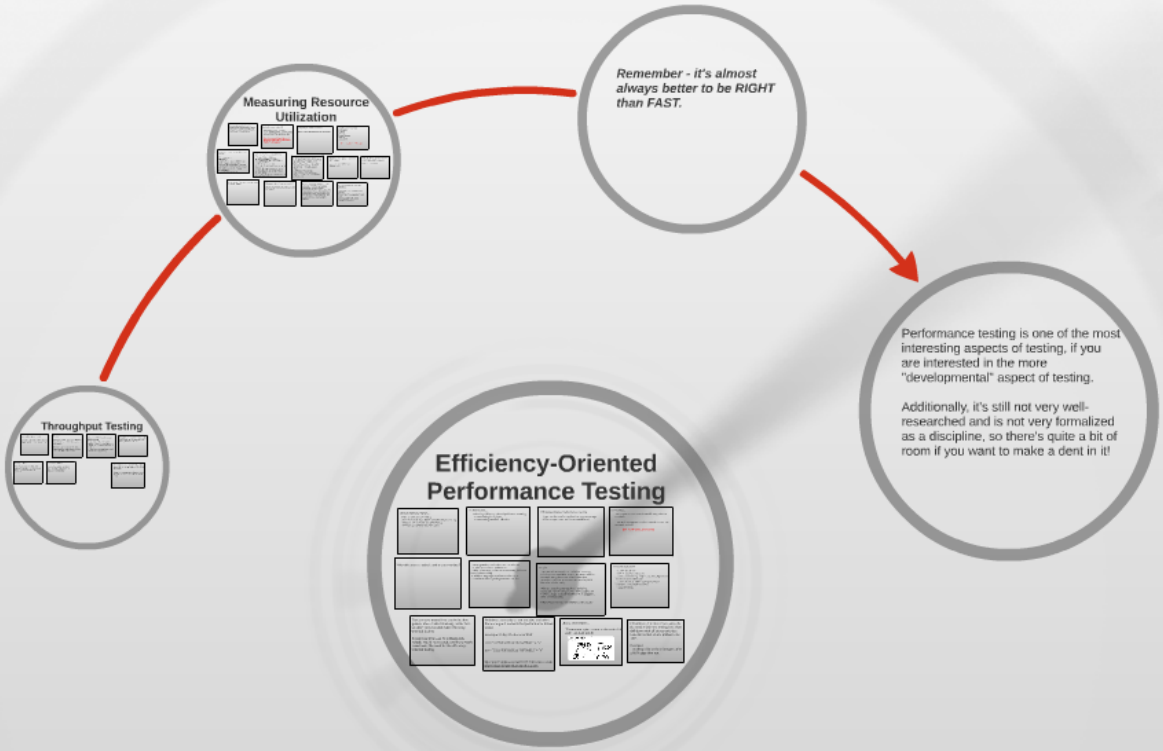
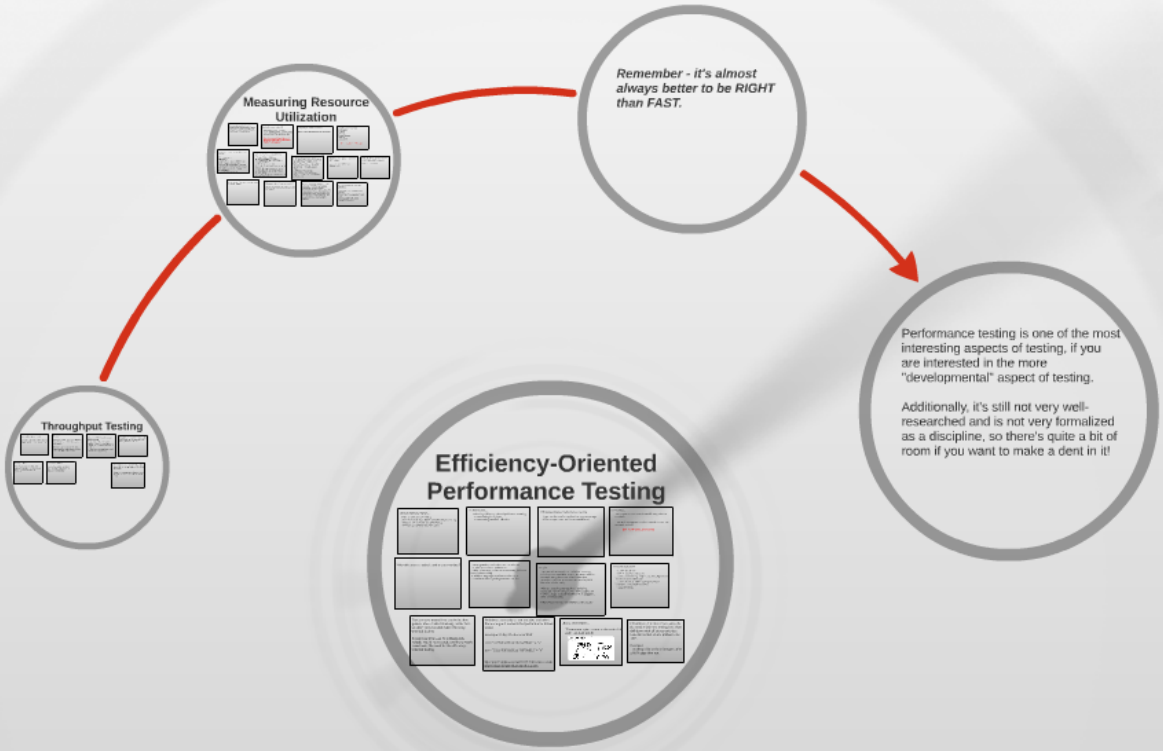

Measuring Resource Utilization
Remember - it's almost always better to be RIGHT than FAST.
Throughput Testing
Efficiency-Oriented Performance Testing
Performance testing is one of the most interesting aspects of testing, if you are interested in the more "developmental" aspect of testing.
Additionally, it's still not very well-researched and is not very formalized as a discipline, so there's quite a bit of room if you want to make a dent in it!


Measuring Resource Utilization
Remember - it's almost always better to be RIGHT than FAST.
Throughput Testing
Efficiency-Oriented Performance Testing
Performance testing is one of the most interesting aspects of testing, if you are interested in the more "developmental" aspect of testing.
Additionally, it's still not very well-researched and is not very formalized as a discipline, so there's quite a bit of room if you want to make a dent in it!

Efficiency-Oriented Performance Testing

Covered last time...

What is performance testing?
Service-Oriented vs Efficiency-Oriented perf testing
Details of service-oriented perf testing
Developing a performance test plan

In this lecture...

Performing efficiency-oriented performance testing
--> Measuring throughput
--> Measuring resource utilization

Efficiency-Oriented Performance Testing

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

Remember...

Developers tend to care about efficiency-oriented numbers

User and management tend to care about service-oriented numbers

(not in all cases, of course)

Why efficiency-oriented performance testing?

1. More granular results than service-oriented
2. Easier to pin down bottlenecks
3. Able to increase hardware as necessary (or know if that is impossible)
4. Talk in a language developers understand
5. Can be 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 lists 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 Solutions

1. Faster hardware
2. Better page construction
3. Check for bad (e.g., O(n^2) for sort) algorithms
4. Lots of malloc/free calls
5. Spread work to other cores/processors
6. Make it a distributed system
7. Cache listings

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.

If response time was 40 milliseconds, nobody would have cared, and there would have been little need for the efficiency-oriented testing.

Problems are rarely so cut-and-dry, and often there are good reasons that performance issues occur.

Example: Ruby 23-character limit

str = "01234567890123456789O1" + "x"

str = "01234567890123456789O12" + "x"

http://patshaughnessy.net/2012/1/4/never-create-ruby-strings-longer-than-23-characters

Also, remember...

"Premature optimization is the root of all evil." -Donald 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.

Example:
Sorting a list of three integers after a kNN algorithm run.

Covered last time...

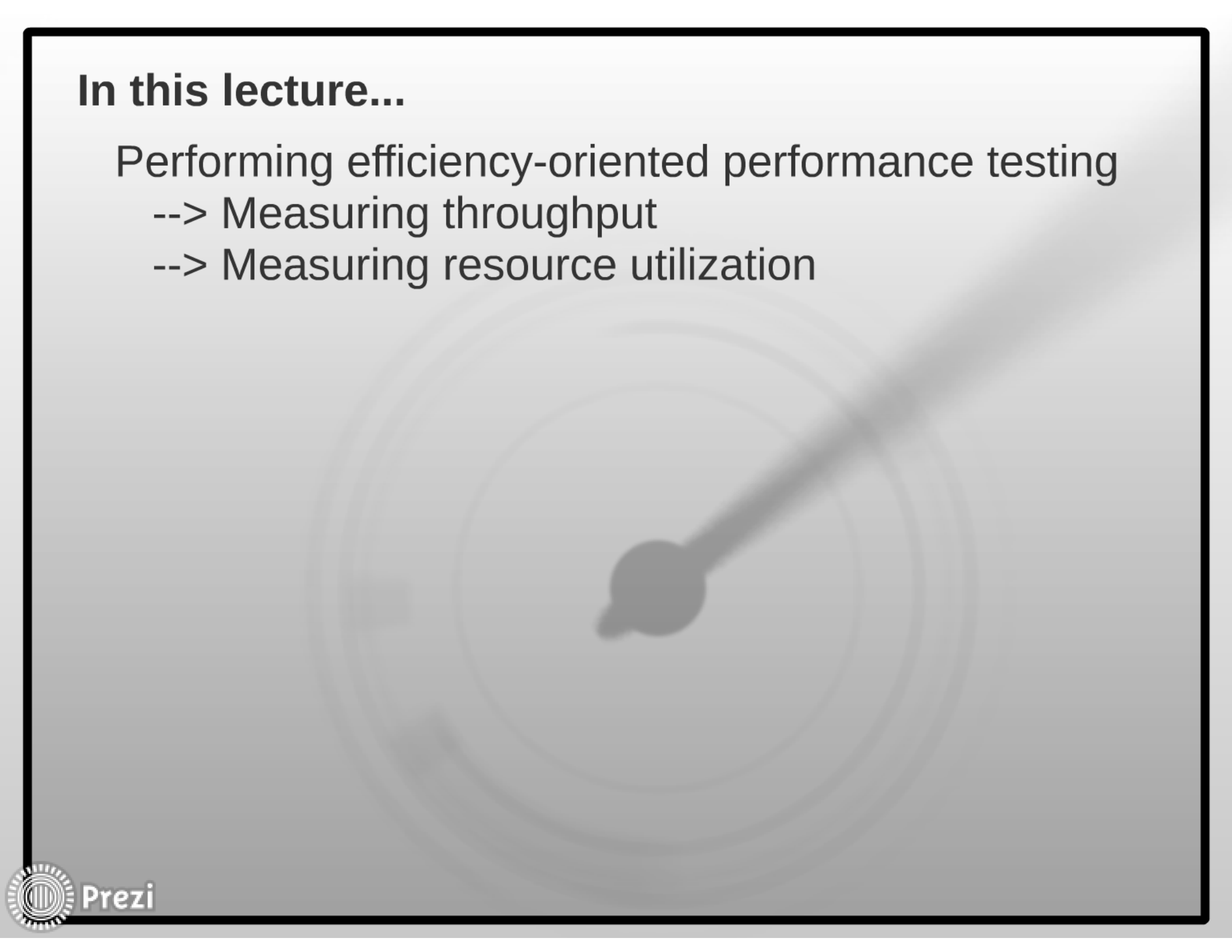
What is performance testing?
Service-Oriented vs Efficiency-Oriented perf testing
Details of service-oriented perf testing
Developing a performance test plan

In this lecture...

Performing efficiency-oriented performance testing

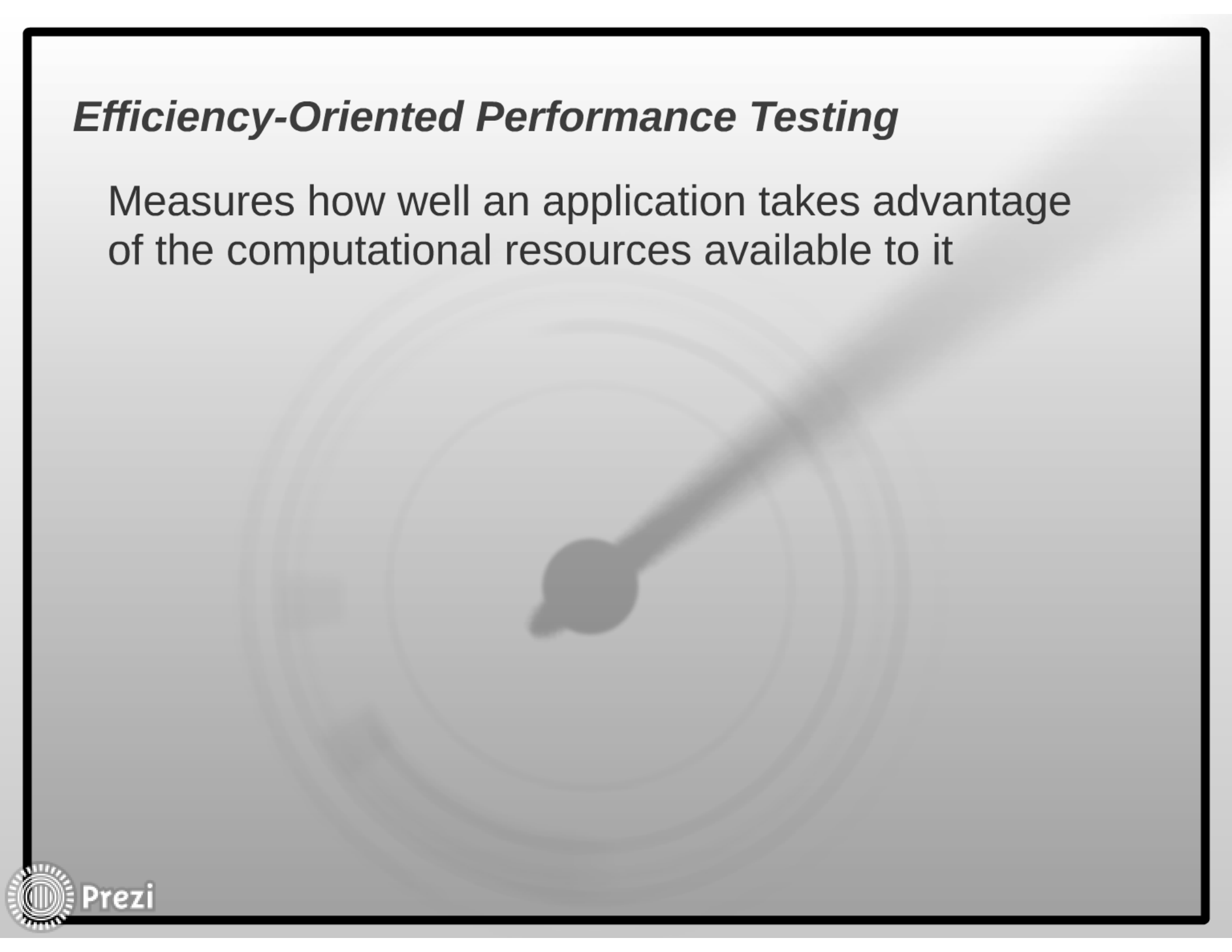
--> Measuring throughput

--> Measuring resource utilization



Efficiency-Oriented Performance Testing

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



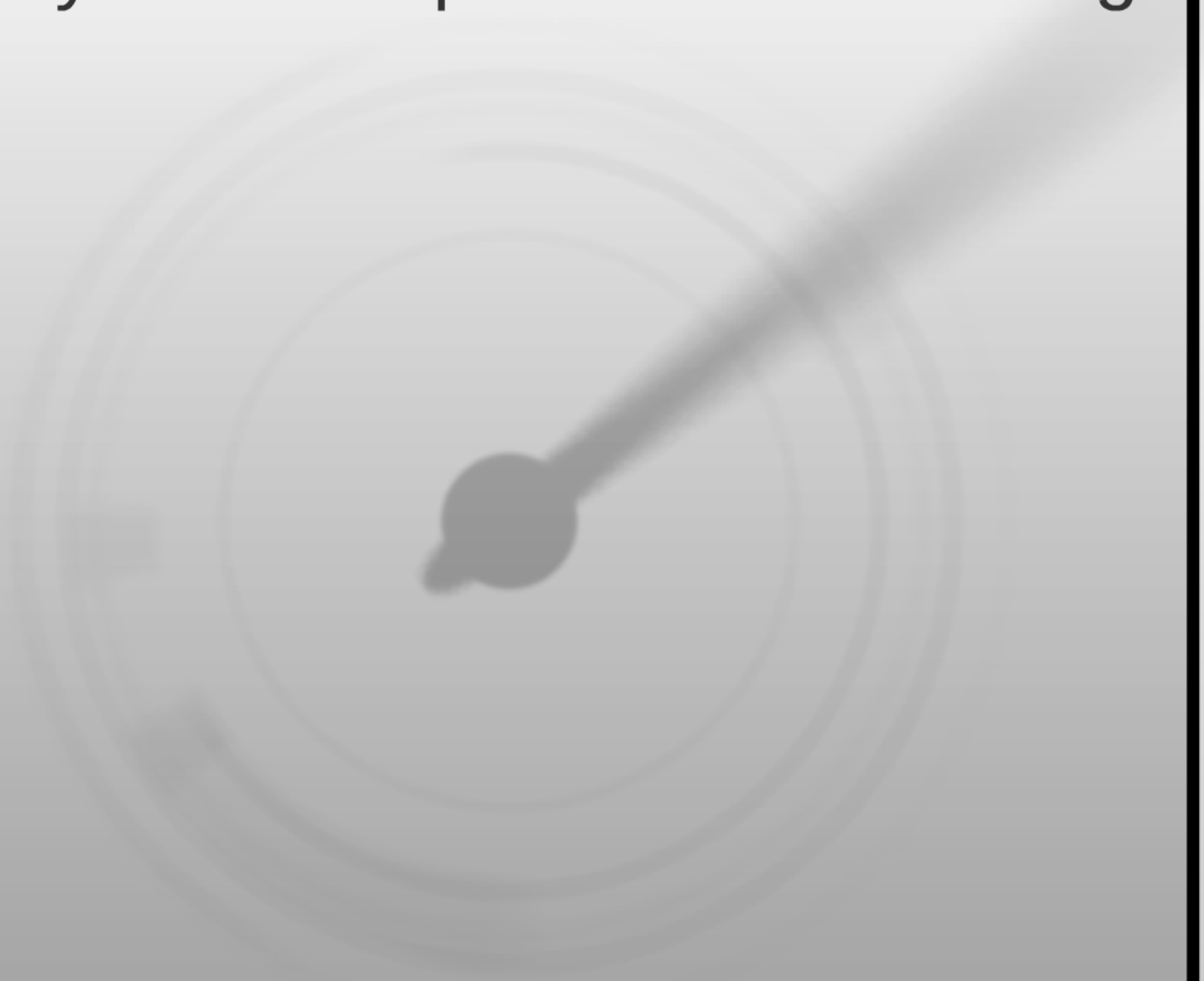
Remember...

Developers tend to care about efficiency-oriented numbers

User and management tend to care about service-oriented numbers

(not in all cases, of course)

Why efficiency-oriented performance testing?



1. More granular results than service-oriented
2. Easier to pin down bottlenecks
3. Able to increase hardware as necessary (or know if that is impossible)
4. Talk in a language developers understand
5. Can be 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 lists 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 Solutions

1. Faster hardware
2. Better page construction
3. Check for bad (e.g., O(n^2) for sort) algorithms
4. Lots of malloc/free calls
5. Spread work to other cores/processors
6. Make it a distributed system
7. Cache listings

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.

If response time was 40 milliseconds, nobody would have cared, and there would have been little need for the efficiency-oriented testing.

Problems are rarely so cut-and-dry, and often there are good reasons that performance issues occur.

Example: Ruby 23-character limit

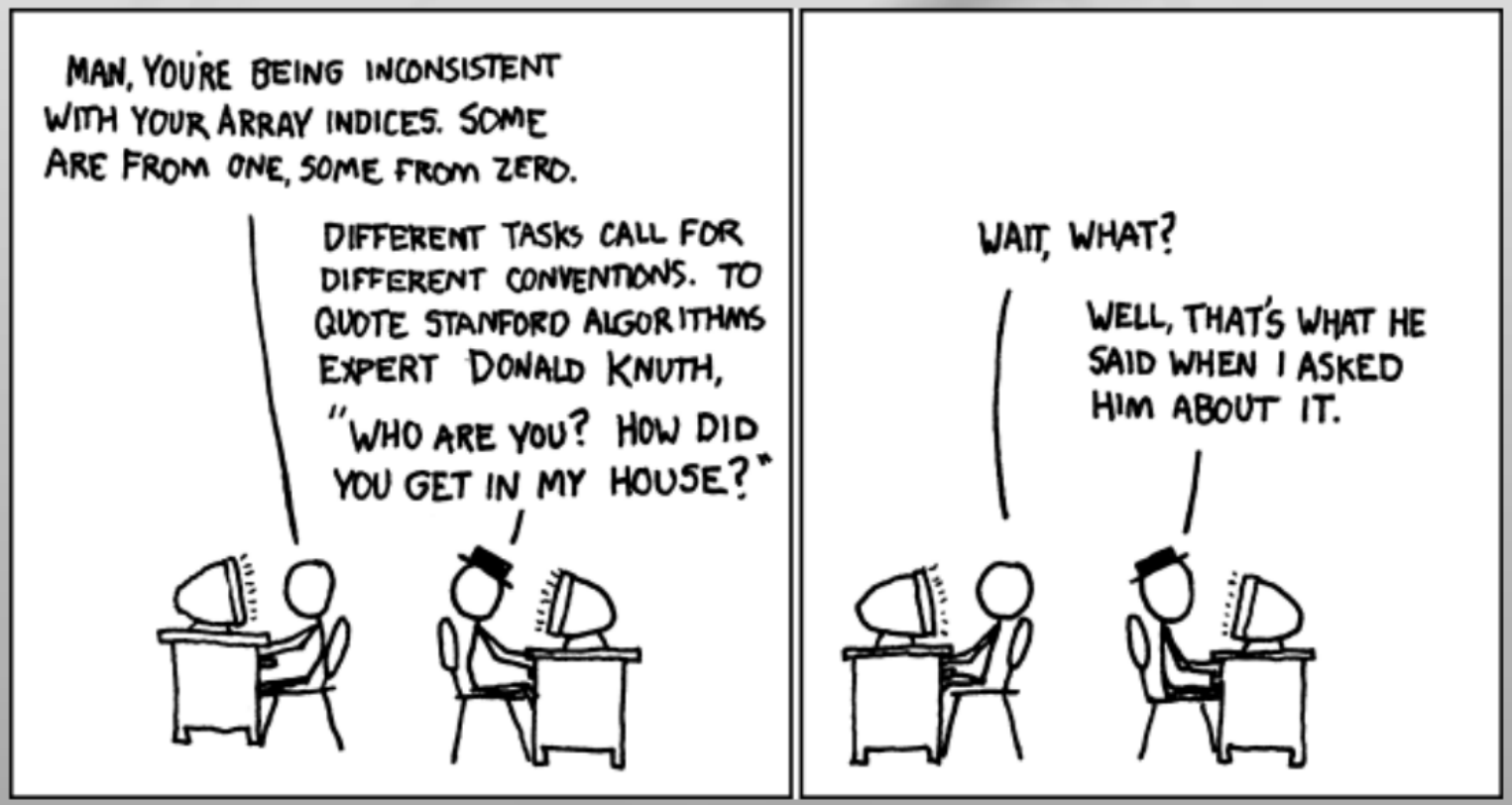
str = "01234567890123456789012" + "x"

str = "012345678901234567890123" + "x"

http://patshaughnessy.net/2012/1/4/never-create-ruby-strings-longer-than-23-characters

Also, remember...

"Premature optimization is the root of all evil." -Donald 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.

Example:

Sorting a list of three integers after a kNN algorithm run.

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.

How is that different from service-oriented testing?

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

In modern testing, load testing is often used (remember we also used load testing to test availability?)

Example: JMeter

An open-source Java tool from the Apache Foundation, which simulates load on a network, server, or program.

Remember Selenium?

Imagine *n* Seleniums running in parallel in order to determine the impact on the web server.

Load testing is just one kind of throughput testing, but at the application level is used most often.

Lower-level throughput testing has its own tools, or you can roll your own with shell scripts.

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.

How is that different from service-oriented testing?

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

In modern testing, load testing is often used (remember we also used load testing to test availability?)

Example: JMeter

An open-source Java tool from the Apache Foundation, which simulates load on a network, server, or program.

Remember Selenium?

Imagine *n* Seleniums running in parallel in order to determine the impact on the web server.

Load testing is just one kind of throughput testing, but at the application level is used most often.

Lower-level throughput testing has its own tools, or you can roll your own with shell scripts.

Measuring Resource Utilization

You really need tools for this (unless you can tell by the sound of your fan how many arithmetic operations are occurring)

Tools will vary based on OS...

Windows Systems - perfmon
OS X - Activity Monitor or Instruments
Unix systems - top, iostat, sar, etc.

Say what you will about Microsoft products, their built-in performance monitor is really good.

A Very Simple Performance Test -

Watch CPU usage while you do something.

Key Resources to Watch:
CPU Usage
Threads
Memory
Virtual Memory
Disk I/O
Network I/O

These are often called "counters".

Sometimes even more specialized data is needed -

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
2,194 counters by default!

Some notes on measuring CPU usage:

1. Modern CPUs are COMPLEX. Understanding utilization is COMPLEX.
2. You really can't compare CPU usage on different processors.
3. Task Manager will lie to you.
4. Resource sharing between cores will impact your results.
5. Virtualization will mess you up, big time.
6. "Power-efficient" chips can give you bad utilization results.
7. CPU utilization only makes sense to measure over long periods of time.

Tips for measuring memory usage
1. Understand difference between private bytes, virtual bytes, working set, etc.
   Private bytes = What app has asked for
   Working set = In physical memory*
   Virtual bytes = Total virtual space allocated
2. Caching can mess you up
3. Really only good for trends (e.g., whether or not you have a memory leak, not where it is)

Performance monitoring of this kind is VERY broad.

What's taking up CPU? Memory? Packets? etc.

Figure out what's going on over the network with a packet analyzer.

Such as... Wireshark!

Figure out what's going on in your program with a profiler, like...

Oftentimes, these things are overkill.

But knowing they're available can save your bacon.

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."

The key to fixing performance problems:

Determine if it is a performance problem
Track down from top-level to low-level
Keep track of performance throughout versions

You really need tools for this (unless you can tell by the sound of your fan how many arithmetic operations are occurring)

Tools will vary based on OS...

Windows Systems - perfmon
OS X - Activity Monitor or Instruments
Unix systems - top, iostat, sar, etc.

Say what you will about Microsoft products, their built-in performance monitor is really good.

A Very Simple Performance Test -

Watch CPU usage while you do something.

Key Resources to Watch:

CPU Usage

Threads

Memory

Virtual Memory

Disk I/O

Network I/O

These are often called "counters".

Sometimes even more specialized data is needed -

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
2,194 counters by default!

Some notes on measuring CPU usage:

1. Modern CPUs are COMPLEX.
Understanding utilization is COMPLEX.
2. You really can't compare CPU usage on different processors.
3. Task Manager will lie to you.
4. Resource sharing between cores will impact your results.
5. Virtualization will mess you up, big time.
6. "Power-efficient" chips can give you bad utilization results.
7. CPU utilization only makes sense to measure over long periods of time.

Tips for measuring memory usage

1. Understand difference between private bytes, virtual bytes, working set, etc.
   Private bytes = What app has asked for
   Working set = In physical memory*
   Virtual bytes = Total virtual space allocated
2. Caching can mess you up
3. Really only good for trends (e.g., whether or not you have a memory leak, not where it is)

Performance monitoring of this kind is VERY broad.

What's taking up CPU? Memory? Packets? etc.

Figure out what's going on over the network with a packet analyzer.

Such as... Wireshark!

Figure out what's going on in your program with a profiler, like...

Oftentimes, these things are overkill.

But knowing they're available can save your bacon.

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."

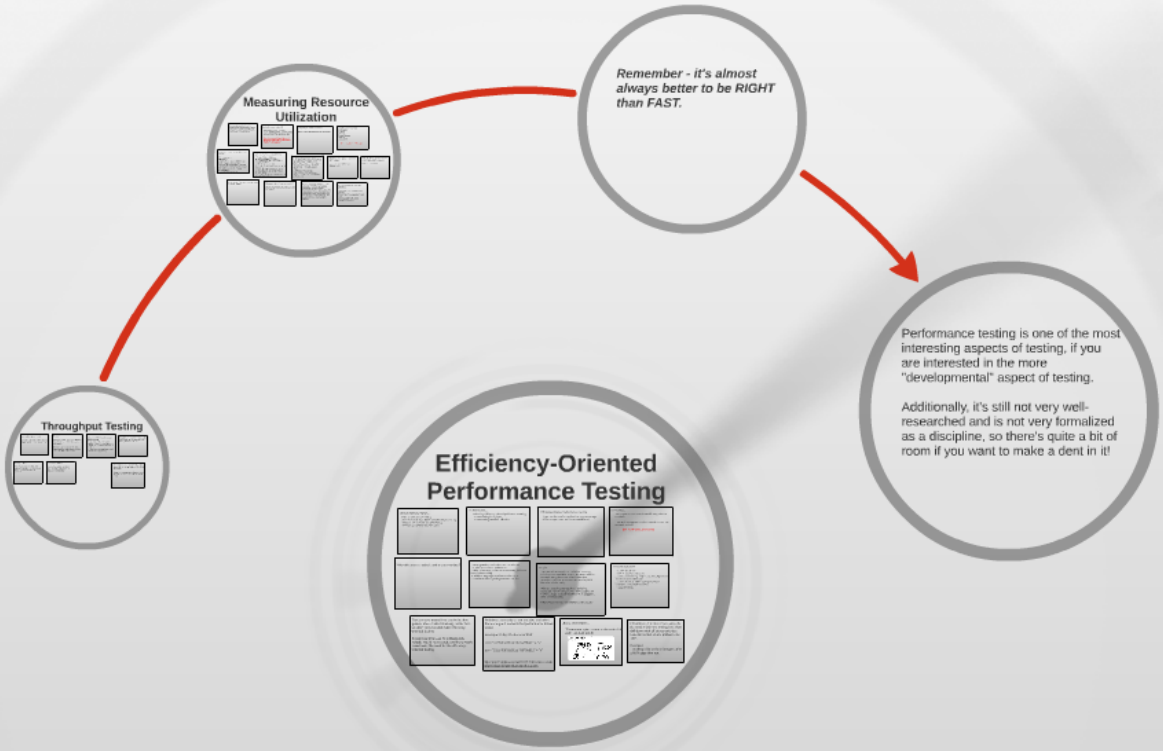
The key to fixing performance problems:

Determine if it is a performance problem
Track down from top-level to low-level
Keep track of performance throughout versions

Remember - it's almost always better to be RIGHT than FAST.

Performance testing is one of the most interesting aspects of testing, if you are interested in the more "developmental" aspect of testing.

Additionally, it's still not very well-researched and is not very formalized as a discipline, so there's quite a bit of room if you want to make a dent in it!


Measuring Resource Utilization
Remember - it's almost always better to be RIGHT than FAST.
Throughput Testing
Efficiency-Oriented Performance Testing
Performance testing is one of the most interesting aspects of testing, if you are interested in the more "developmental" aspect of testing.
Additionally, it's still not very well-researched and is not very formalized as a discipline, so there's quite a bit of room if you want to make a dent in it!