UNIVERSITY OF WASHINGTON

CSE 403

Performance Profiling

Marty Stepp

How can we optimize it?

```
public static String makeString() {
    String str = "";
    for (int n = 0; n < REPS; n++) {
        str += "more";
    }
    return str;
}
```

How can we optimize it?

```
// pre: n >= 1
public static long fib(int n) {
    if (n <= 2) {
        return 1;
    } else {
        return fib(n - 2) + fib(n - 1);
    }
}
```

How can we optimize it?

```
if (x == 1) {
    foo(123);
} else if (x == 2) {
    bar(456);
} else if (x == 3 || x == 4) {
    baz(789);
} else if (x == 5) {
    mumble(123);
} else if (x == 6 || x == 7) {
    qxz(456);
} else {

}
```

How can we optimize it?

```
import java.util.*;

// A set of words in our game.
public class WordDictionary {
    private List<String> words = new ArrayList<String>();

    public void add(String word) {
        words.add(word.toLowerCase());
    }

    public boolean contains(String word) {
        return words.contains(word.toLowerCase());
    }
}
```

How can we optimize it?

- What is "method call overhead"?
- What is "loop unrolling"?
- Which type adds and multiplies faster, int or double?
- Is it faster to use a float rather than a double? Is it faster to use an int rather than a long?
- Is Java's Math class slow?
- Are regular expressions slow?
- Are objects slow? Is inheritance slow?
- Are enums slow? Are <generics> slow?
- Is a linked list slower than an ArrayList?
- Are UDP sockets faster than TCP sockets?
- How long does it take to do a lookup in your SQL database?

The correct answer

- **Who cares?**

App performance

- These days, many apps end up having perfectly acceptable performance without the programmer worrying about it very much.
  - How do we decide what "acceptable" performance is?
- When an app is too slow, often the performance bottleneck is caused a small number of bugs or lines of code.
  - If we can find and fix these later, we can largely ignore the issue of optimization the rest of the time.

Optimization quotes

- "We follow two rules in the matter of optimization:
  - 1. Don't do it.
  - 2. *(for experts only)*
    Don't do it yet."
    -- M.A. Jackson

- "Premature optimization is the root of all evil."
  -- D. Knuth

High-level perf. questions

- If your app *is* too slow, ask yourself some questions:
  - Did I choose the right data structure?
  - Am I using the right sorting algorithm?
  - Is my recursive method TOO recursive?
  - Am I throwing away an expensive computation result that could instead reuse?
  - Am I creating too many objects unnecessarily or otherwise wasting memory?
  - Is my app's performance dependent on a fast network connection?

Perf. questions, continued

- Or, more concisely:
  - "Where the heck is this bottleneck, and how do I get rid of it?"
- "Where the heck is this bottleneck?"
  - profiling and performance testing
- "How do I get rid of it?"
  - optimization
  - user experience improvements
  - bribe the customer to change the performance requirements

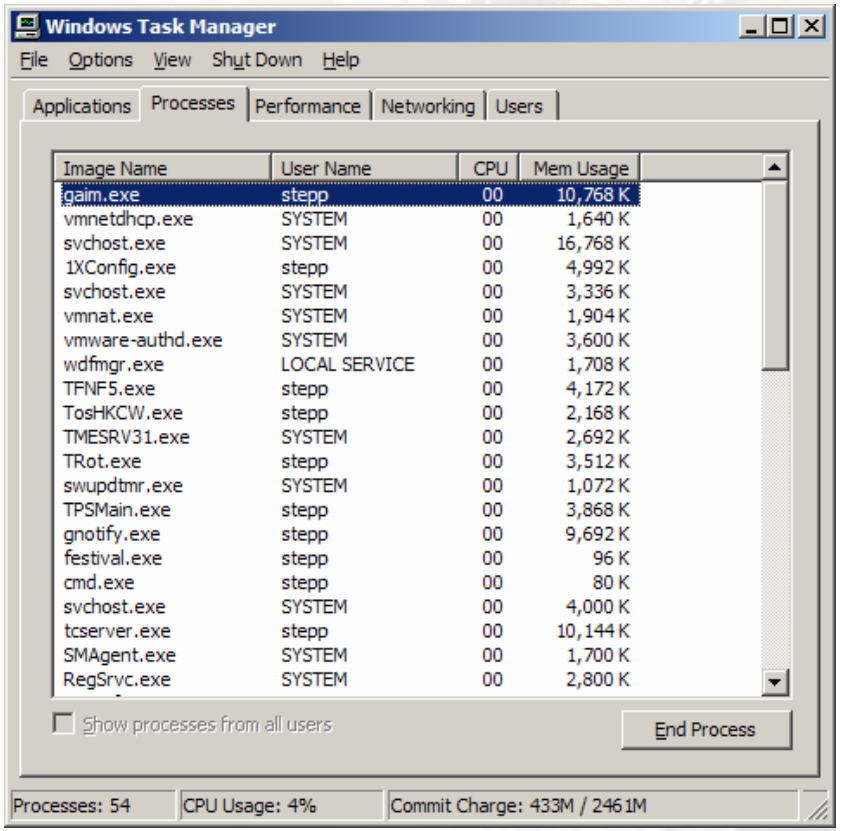
Optimization metrics

- runtime / CPU usage:
  - what lines of code the program is spending the most time in
  - what call/invocation paths were used to get to these lines
    - naturally represented as tree structures

- memory usage:
  - what kinds of objects are sitting on the heap
  - where were they allocated
  - who is pointing to them now
  - "memory leaks" (does Java have these?)

Memory optimization

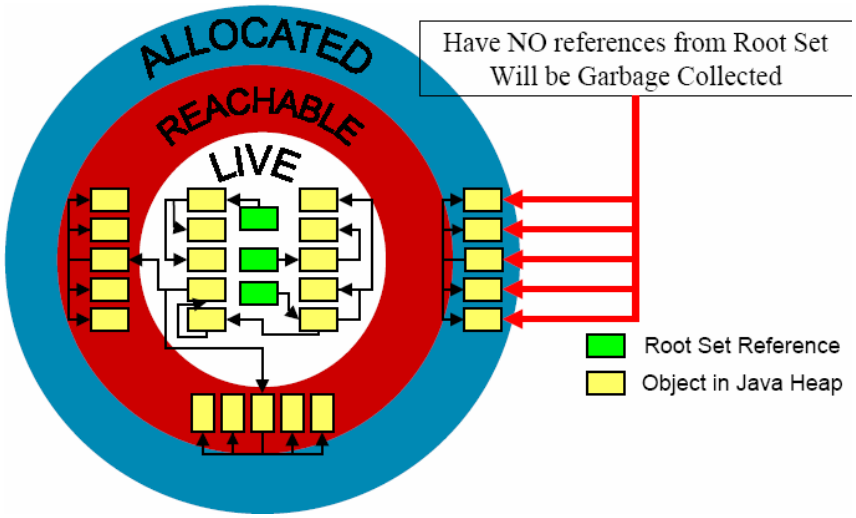
- How much memory is my app using?
  - Windows: Shift-Ctrl-Escape shows Task Manager
  - Linux: run `top`
  - Mac: probably using too much memory (it's a Mac, after all)
- Why is my app using so much memory?

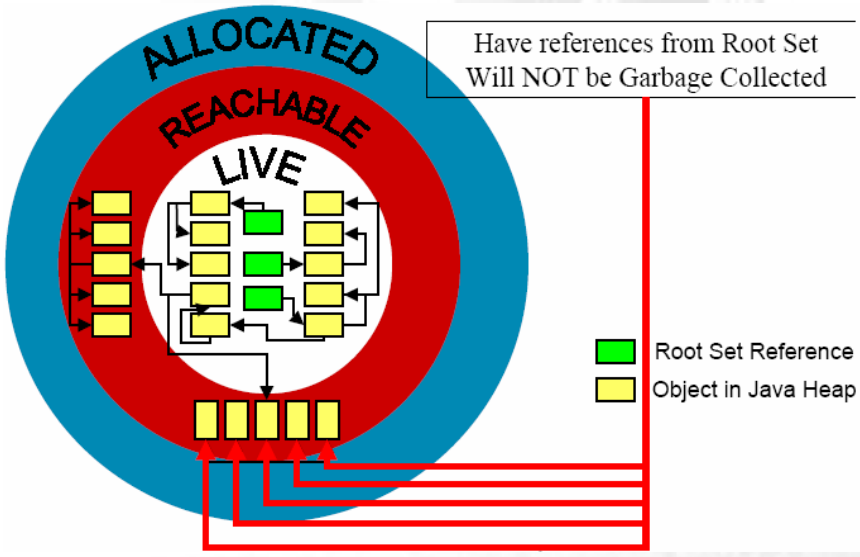


Garbage collection

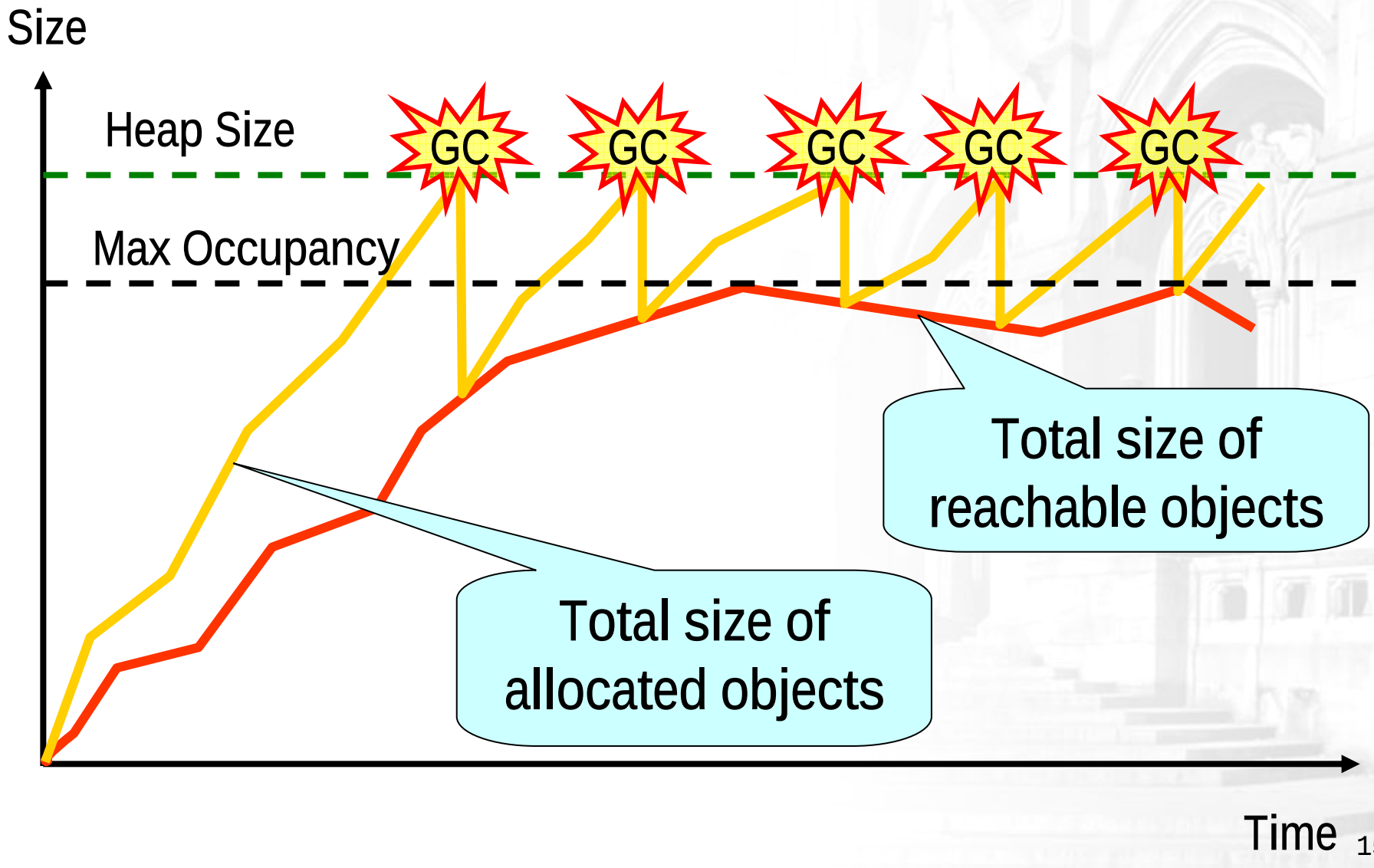
- A memory manager that reclaims objects that are not reachable from a root-set

- **root set**: all objects with an immediate reference
  - all reference variables in each frame of every thread's stack
  - all static reference fields in all loaded classes



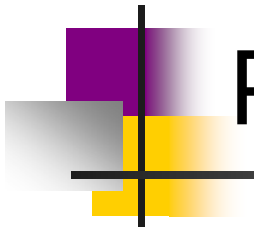


Heap and garbage collection



Profiling, benchmarking, ...

- **profiling**: Measuring relative system statistics.
  - Where is the most time being spent? ("classical" profiling)
    - Which method takes the most time?
    - Which method is called the most?
  - How is memory being used?
    - What kind of objects are being created?
    - This in especially applicable in OO, GCed environments.
  - Profiling is *not* the same as benchmarking or optimizing.
- **benchmarking**: Measuring the absolute performance of your app on a particular platform.
- **optimization**: Applying refactoring and enhancements to speed up code.



Profiling motivation

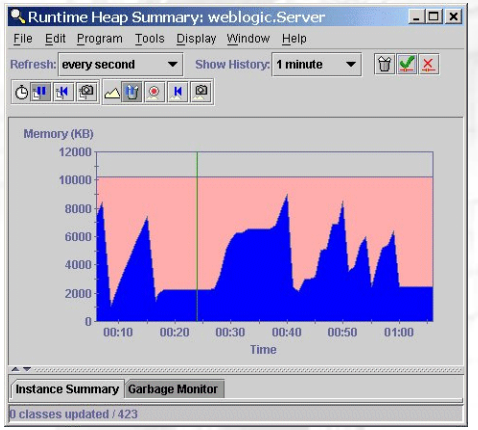
- Why use a profiler?
  - your intuition about what's slow is often wrong
  - performance is a major aspect of program acceptance by users and customers
- Profiler advantages:
  - accuracy
  - completeness
  - solid, statistical information
  - platform- and machine-independence
- When should I profile my code?

What profiling tells you

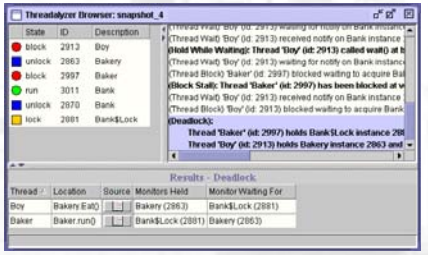
- Basic information:
  - How much time is spent in each method? ("flat" profiling)
  - How many objects of each type are allocated?
- Beyond the basics:
  - Program flow ("hierarchical" profiling)
    - Do calls to method A cause method B to take too much time?
  - Per-line information
    - Which line(s) in a given method are the most expensive?
    - Which methods created which objects?
  - Visualization aspects
    - Is it easy to use the profiler to get to the information you're interested in?

Tools

- Many free Java profiling/optimization tools available:
  - TPTP profiler extension for Eclipse
  - Extensible Java Profiler (EJP) - open source, CPU tracing only
  - Eclipse Profiler plugin
  - Java Memory Profiler (JMP)
  - Mike's Java Profiler (MJP)
  - JProbe Profiler - uses an instrumented VM



- hprof (`java -Xrunhprof`)
  - comes with JDK from Sun, free
  - good enough for anything I've ever needed (if you use it with a visualizer like HPjmeter)



Using hprof

```
usage: java -Xrunhprof:[help]|[<option>=<value>, ...]

Option Name and Value  Description                    Default
---------------------  -----------                    -------
heap=dump|sites|all    heap profiling                 all
cpu=samples|times|old  CPU usage                      off
monitor=y|n            monitor contention             n
format=a|b             text(txt) or binary output     a
file=<file>            write data to file             off
depth=<size>           stack trace depth              4
interval=<ms>          sample interval in ms          10
cutoff=<value>         output cutoff point            0.0001
lineno=y|n             line number in traces?         Y
thread=y|n             thread in traces?              N
doe=y|n                dump on exit?                  Y
msa=y|n                Solaris micro state accounting n
force=y|n              force output to <file>         y
verbose=y|n            print messages about dumps     y
```

Sample hprof usage

- Running hprof:
  - To profile heap **`memory`** / object usage:
    `java -Xrunhprof` ***`mainClassName`***
  - To profile **CPU** cycles:
    `java -Xrunhprof:cpu=old,thread=y,depth=10,cutoff=0,format=a`
    `mainClassName`

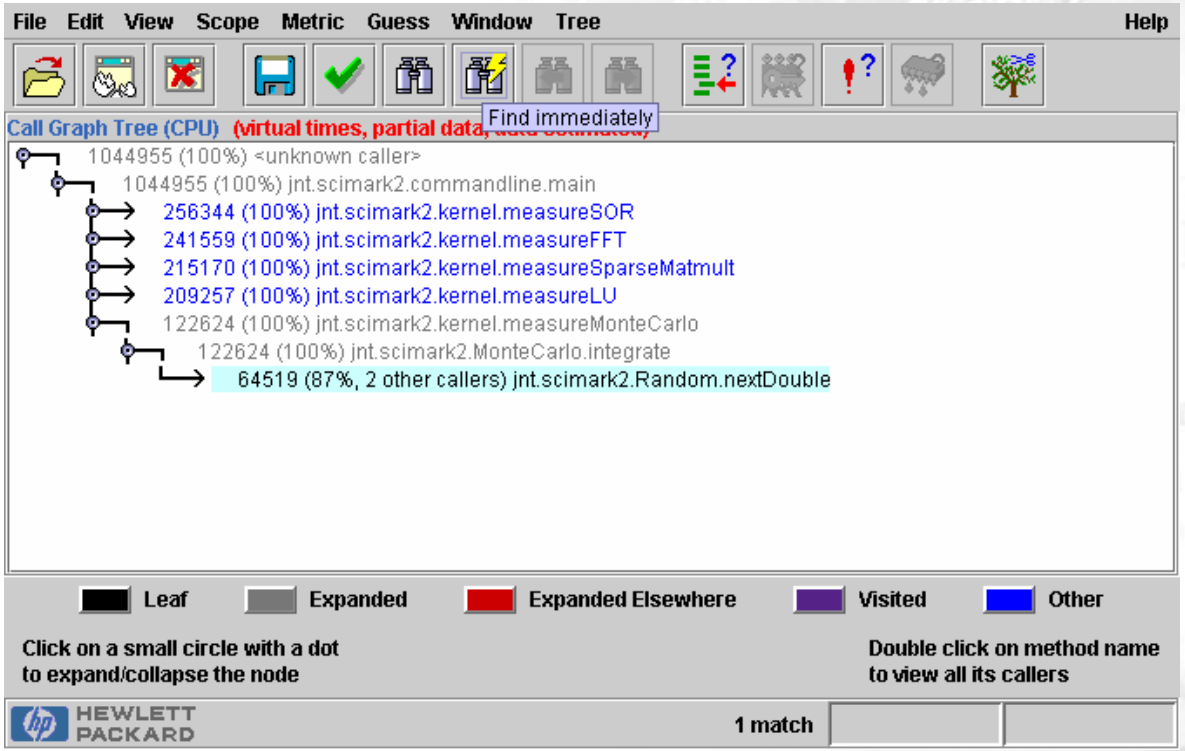
- After execution, open the text file `java.hprof.txt` in the current directory with a text editor or a viewer

Visualization tools

- CPU samples
  - critical to see the traces to modify code
  - hard to read - far from the traces in the file
  - HP's *HPjmeter* analyzes java.hprof.txt visually
    - http://www.hp.com/products1/unix/java/hpjmeter/
  - another good tool called *PerfAnal* builds and navigates the invocation tree
- Heap dump
  - critical to see what objects are there, and who points to them
  - very hard to navigate in a text file!
  - Tools: HPjmeter or *HAT* navigate heap objects
    - https://hat.dev.java.net/

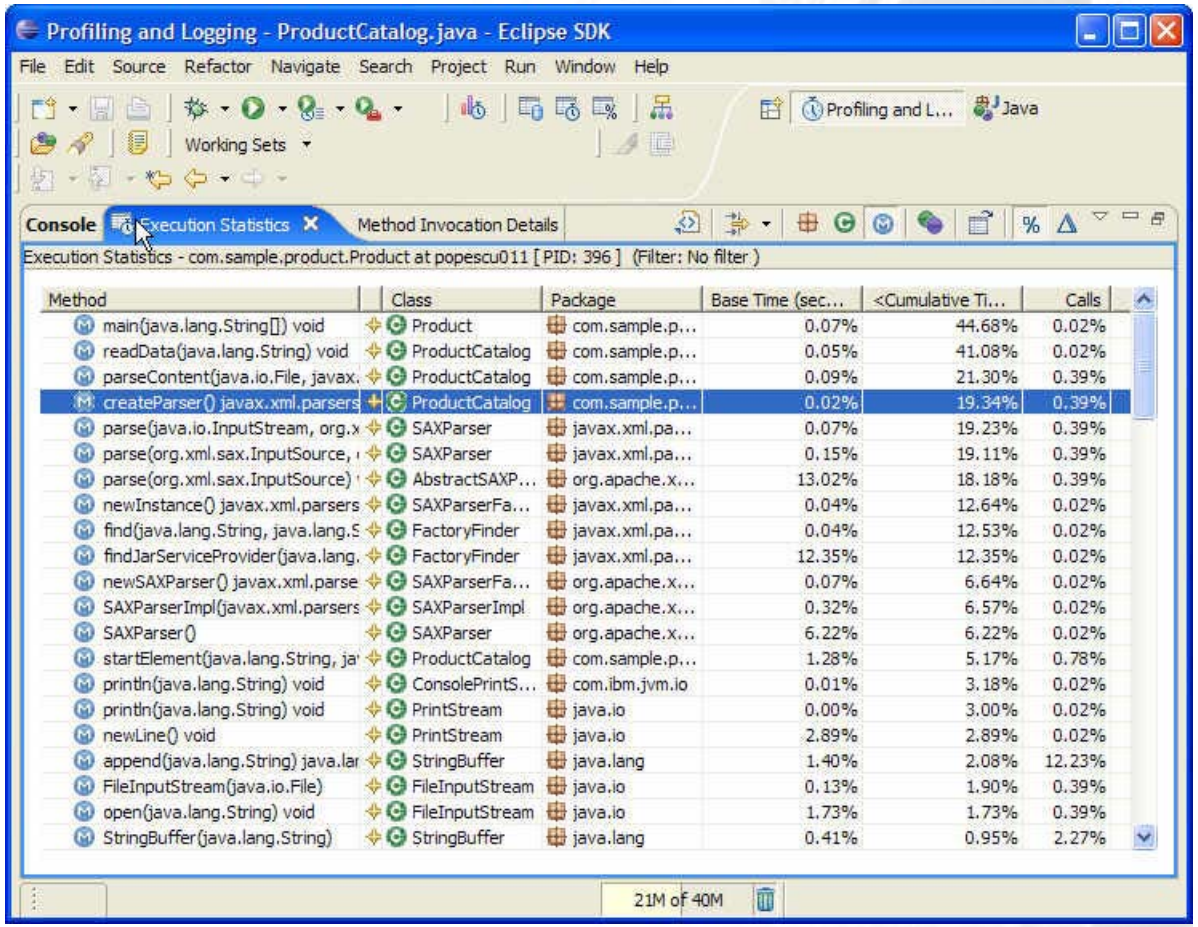
HPjmeter

- Useful tool for analyzing `java.hprof.txt`
- Can display heap (memory/objects) data, CPU samples, call trees



TPTP

- a free extension to Eclipse for profiling
  - difficult to install, but very powerful



Warnings

- CPU profiling really slows down your code!
  - Design your profiling tests to be very short
  - CPU Samples is better than CPU Time
- CPU samples don't measure everything
  - Doesn't record object creation and garbage collection time, which can be significant!
- Output files are very large, especially if there is a heap dump

Profiling Web languages

- JavaScript
  - Firebug: http://www.getfirebug.com/
  - Venkman: http://www.mozilla.org/projects/venkman/
- PHP
  - APD: http://www.pecl.php.net/package/apd
  - Benchmark: http://pear.php.net/benchmark
  - DBG: http://dd.cron.ru/dbg
  - Xdebug: http://xdebug.derickrethans.nl/
- Ruby on Rails
  - ruby-prof: http://ruby-prof.rubyforge.org/
- JSP
  - x.Link: http://sourceforge.net/projects/xlink/

Profiling web apps

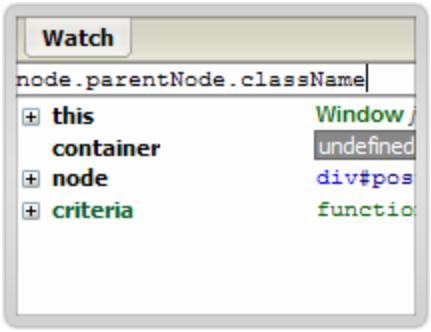
Some new questions arise:

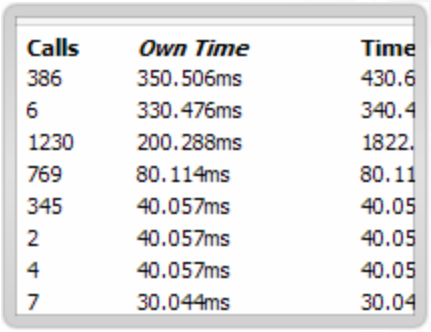
- In what tier of my app does the bottleneck occur?
  - Database?
  - Server-side script?
  - Network delays?
  - Client-side?
  - …
- Can we throw hardware at this problem?
  - A better server?
  - More servers?
  - Indexes on our database?
  - Require the client to have a better computer??

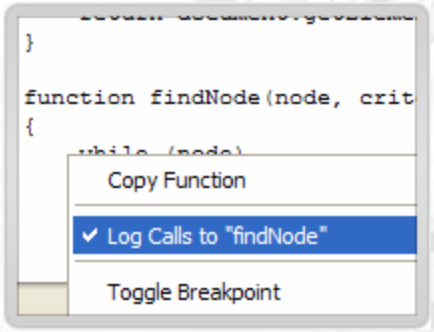
Firebug

- An excellent web debugger/profiler add-on for Firefox

- Profiling in Firebug:
  - open Firebug
  - click Console tab
  - click Profile button
  - use your web app for a while
  - click Profile button again
  - look at data!







What to do with profiler results

- observe which methods are being called the most
  - these may not necessarily be the "slowest" methods!
- observe which methods are taking the most time relative to the others
  - common problem: inefficient unbuffered I/O
  - common problem: poor choice of data structure
  - common problem: recursion call overhead
  - common problem: unnecessary re-computation of expensive information, or unnecessary multiple I/O of same data

What to do after optimization

- Sometimes a performance bottleneck is unavoidable. What to do?
  - It's all about the user experience.
    - Add some UI that informs the user that work is actually happening (progress bars, hourglass).
    - Run the slow task in a thread so that the user can do other work.
- Sometimes performance bottlenecks exist in library code (e.g. Java APIs), not in your code. What to do?
  - Can you use a different call, another algorithm or component?
  - Can you reduce the frequency of calls to the slow method(s)?
  - Are you using the framework / API effectively?