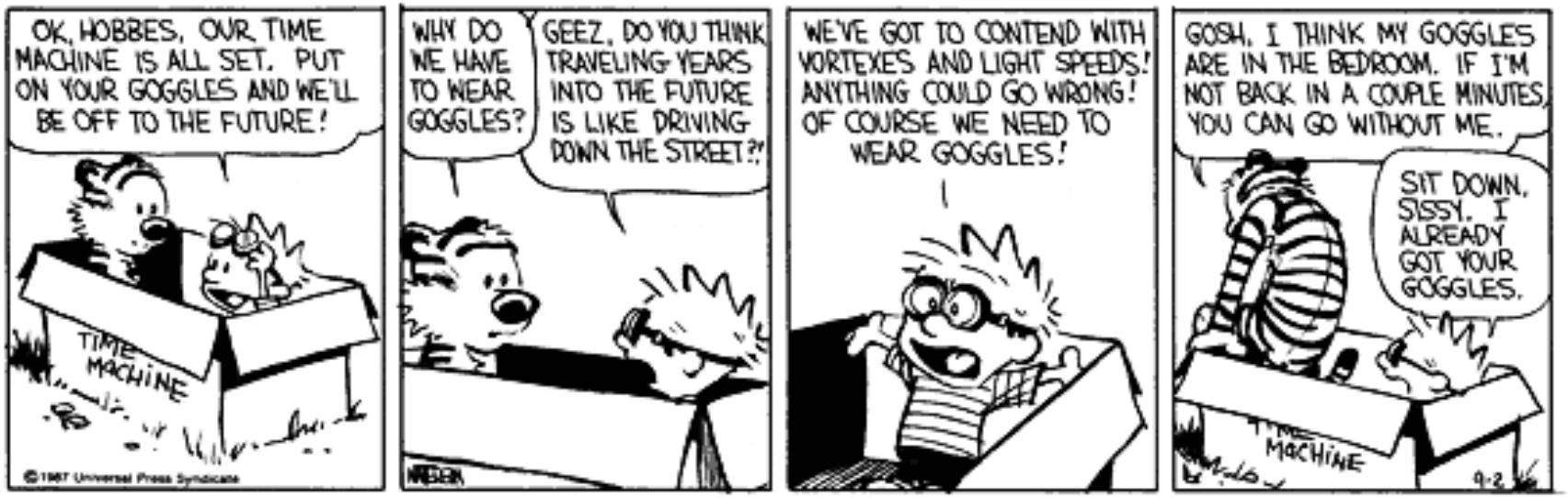
CERN School of Computing

Tools and Techniques

Introduction

Tools you can use individually: Test frameworks, memory checkers

The size of the task: Building software for a collaboration



What do you need to do the job?

I need to calculate the sum of primes numbers in the 1st 100 integers:

```
int sumPrimes() {
    int sum = 0;
    for ( int i=1; i < 100; i++ ) {  // loop over possible primes
        bool prime = true;
        for (int j=1; j < 10; j++) { // loop over possible factors
            if (i % j == 0) prime = false;
        }
        if (prime) sum += i;
    }
    return sum;
}
```

This is quick, throw-away code

- Not well structured, efficient, general or robust
- I understand what I intended, because I wrote it just now

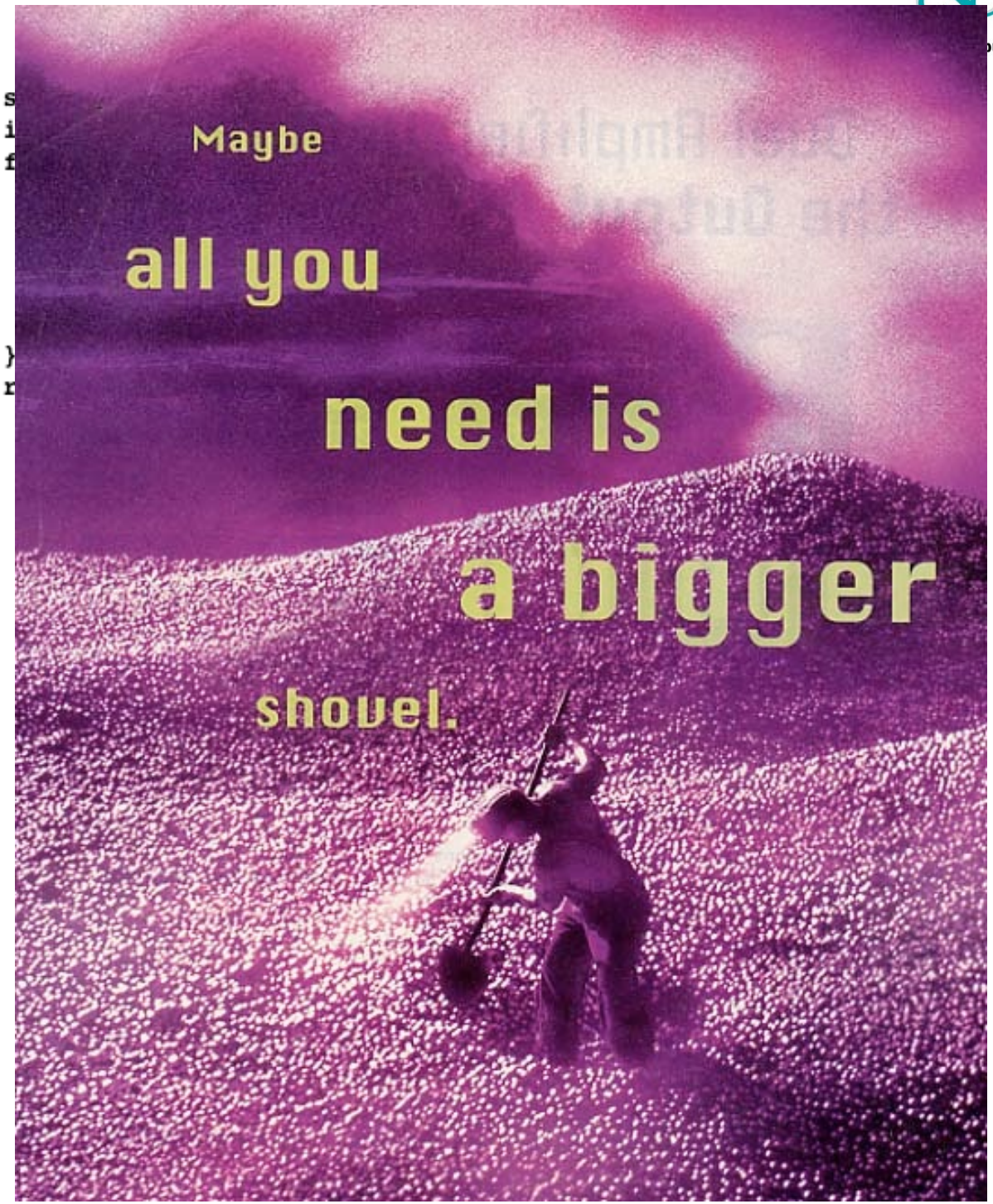
Already, I need an editor, compiler, linker, and probably a debugger

“Don’t worry, I’ll remember what I changed.”

“The answer looks OK, lets move on.”

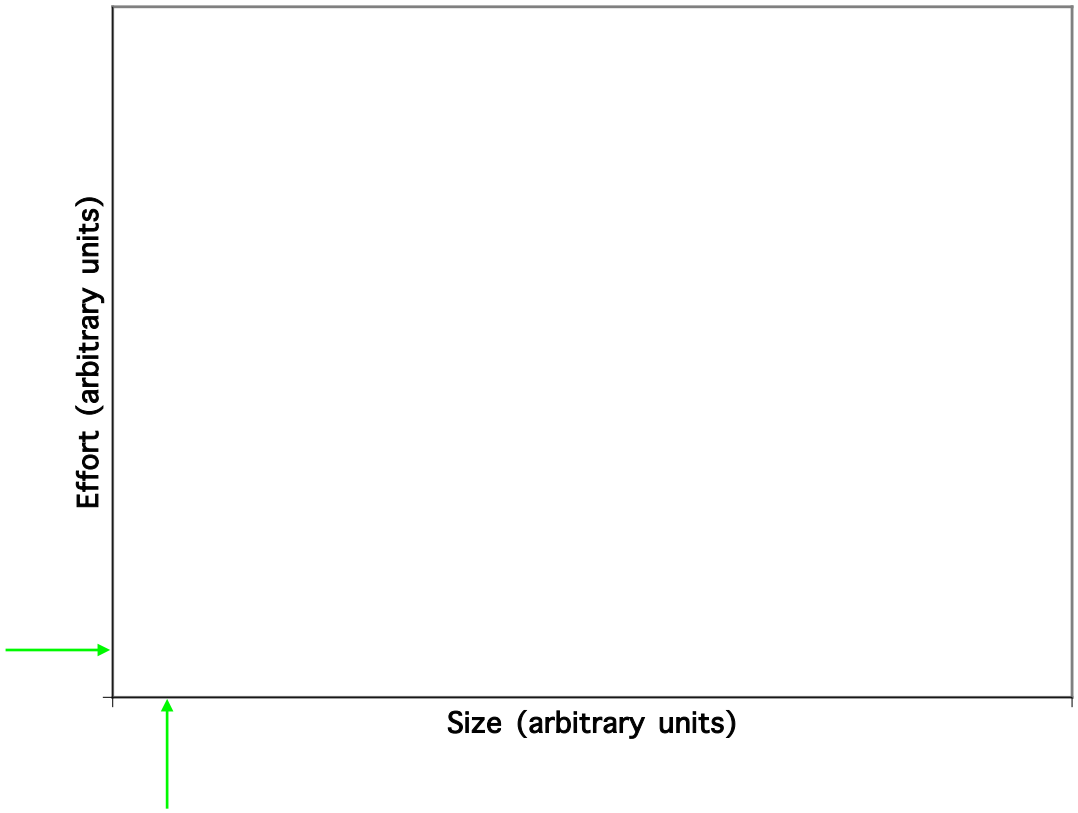
“Does anybody know where this value came from?”

“Your #%@!& code broke again!”



Projects come in different sizes

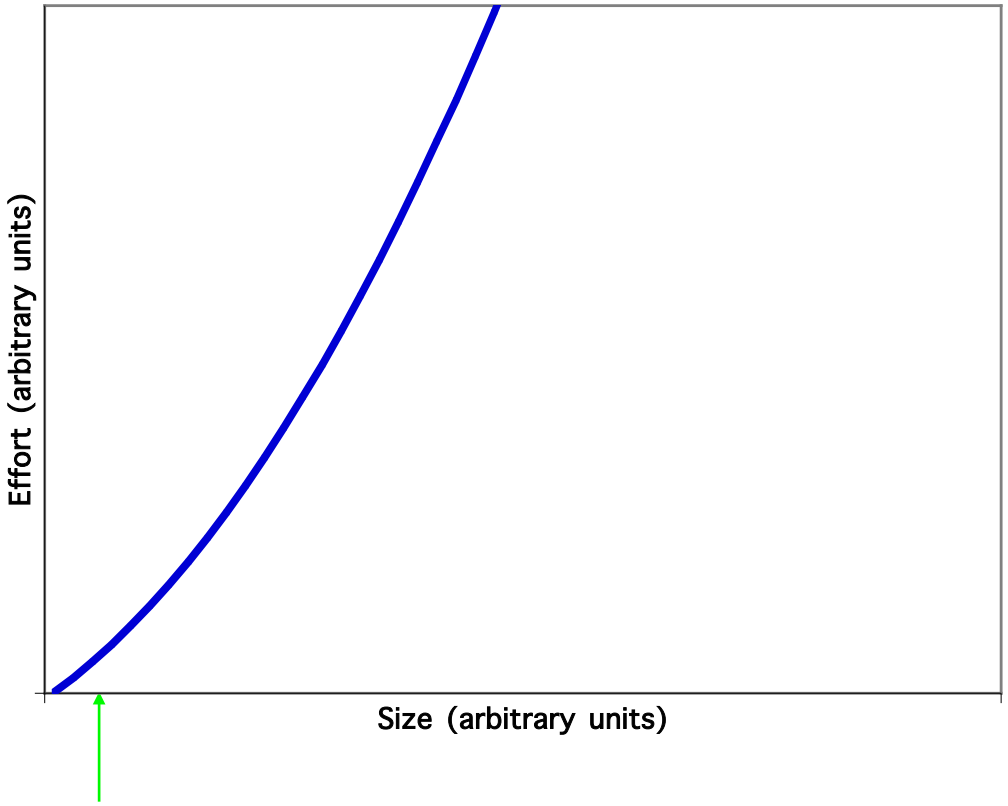
My sample program is a pretty small project!



Projects come in different sizes

My sample program is a pretty small project!

It can be done with a simple technique:



But that won't solve larger problems well

Projects come in different sizes

My sample program is a pretty small project!

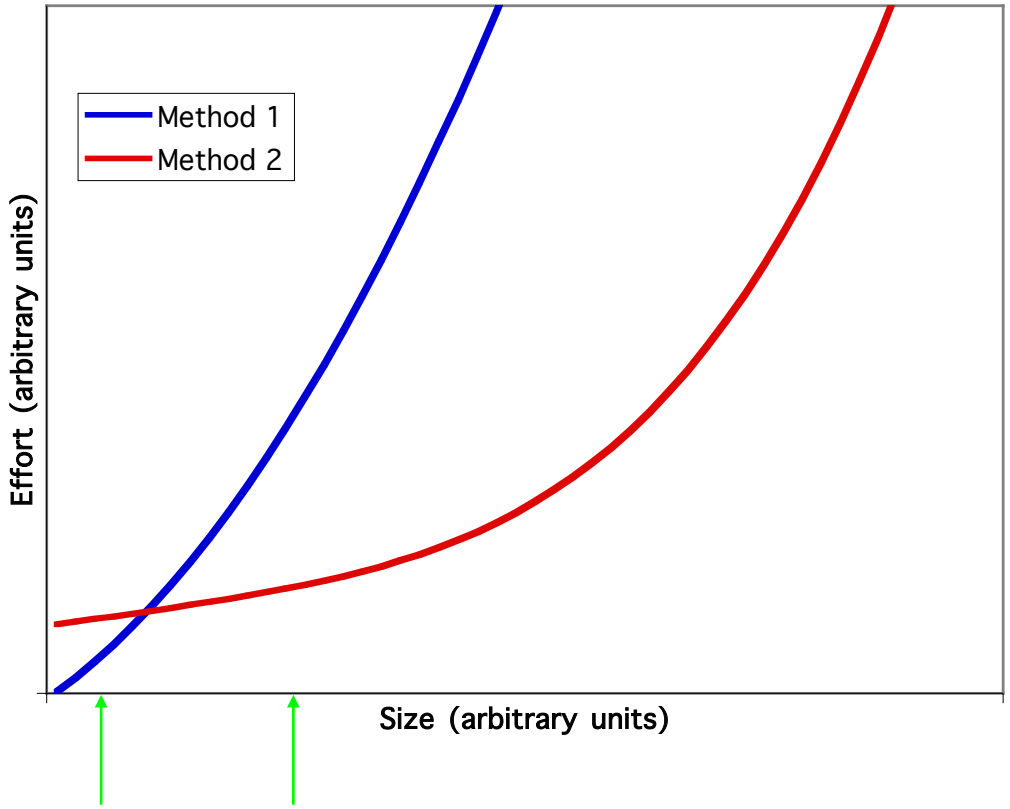
It can be done with a simple technique:

But that won't solve larger problems well

Projects come in different sizes

A larger project may need a different approach

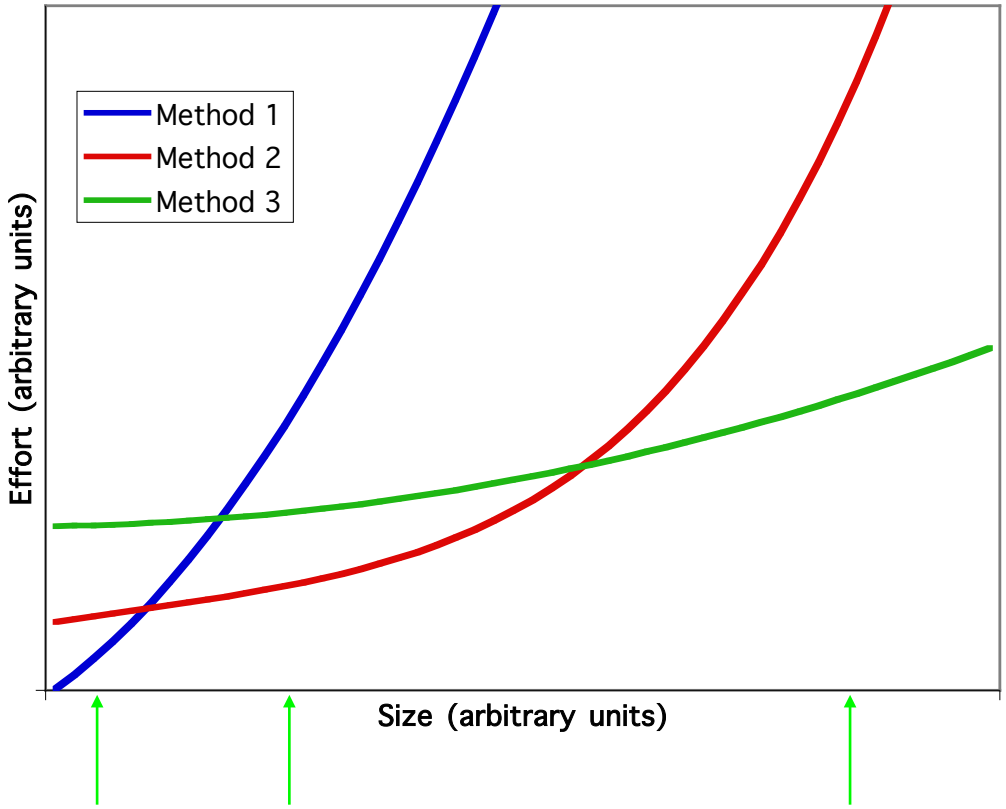
- Those tend to require more effort up front



What do you do when your project grows?

Projects come in different sizes

If you're trying to solve a really large problem:



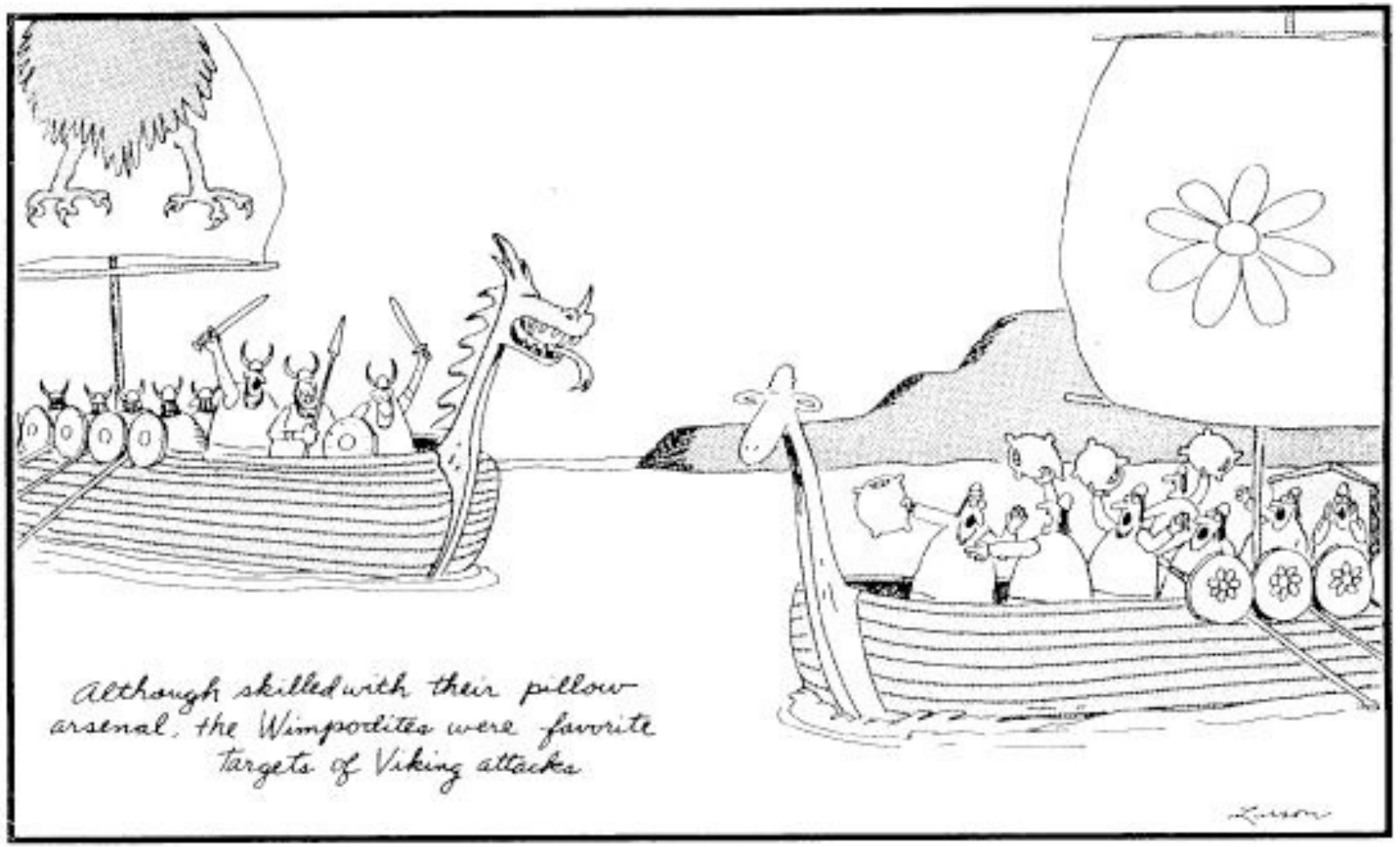
What has all this to do with us?

Our systems tend to be complex systems

- HEP tends to work at the limit of what we know how to do

"If you only have a hammer, wood screws look a lot like nails" - ??

"If you only have a screwdriver, nails are pretty useless" - Don Briggs



Larger projects have standard ways of doing things

To make it possible to communicate, you need a shared vocabulary

- Standards for languages, data storage, etc.

For people to work together, you have to control integrity of source code

- E.g. SVN/Git to provide versioning and control of source code

Just building a large system can be difficult

- Need tools for creating releases, tracking problems, etc.

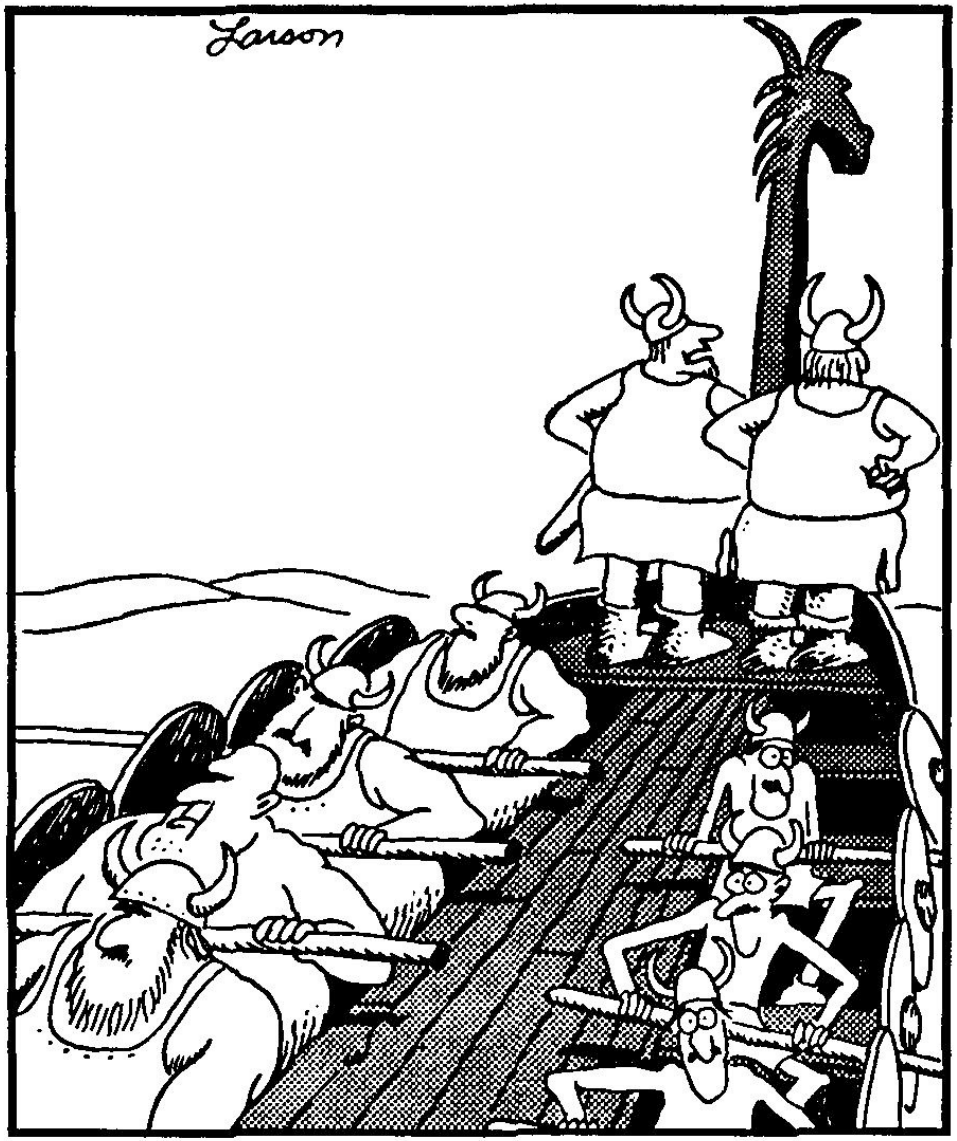
But individual effort is still important!

You can't build a great system from crummy parts

You want your efforts to make a difference

Good tools & technique can help you do a better job

"Whatever you do may seem insignificant, but it is most important that you do it." - Gandhi



"I've got it, too, Omar ... a strange feeling like we've just been going in circles."

CERN School of Computing

Tools you can use

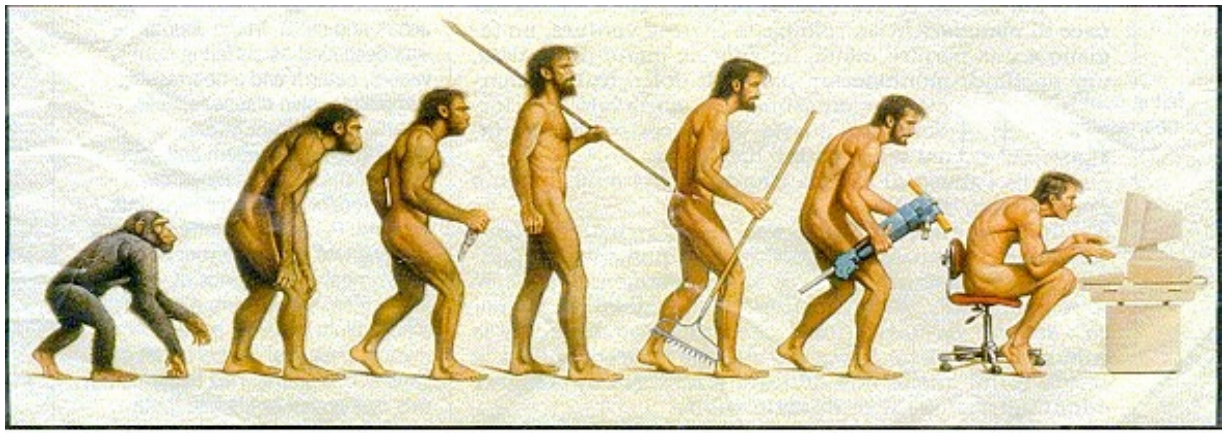
Knowing whether it works - JUnit, CppUnit, etc

Toward an informed way of experimental working

Progress often comes from small, experimental changes

- Allows you to make quick progress on little updates
- Without risk to the big picture

How do you know those steps are progress?



Somewhere, something went terribly wrong

Testing

But don't you see Gerson - if the particle is too small and too short-lived to detect, we can't just take it on faith that you've discovered it."

The role of testing tools

Remember our original example: sum of primes in first 100 integers

- Simple routine, written in a few minutes
- "So simple it must be right"

```
int sumPrimes() {
    int sum = 0;
    for ( int i=1; i < 100; i++ ) {  // loop over possible primes
        bool prime = true;
        for (int j=1; j < 10; j++) { // loop over possible factors
            if (i % j == 0) prime = false;
        }
        if (prime) sum += i;
    }
    return sum;
}
```

The role of testing tools

Remember our original example:

- Simple routine, written in a few minutes
- "So simple it must be right"

```
int sumPrimes(int end) {
    int sum = 0;
    for ( int i=1; i < end; i++ ) {  // loop over possible primes
        bool prime = true;
        for (int j=1; j < 10; j++) { // loop over possible factors
            if (i % j == 0) prime = false;
        }
        if (prime) sum += i;
    }
    return sum;
}
```

- (Assume) valuable enough to reuse and extend

But it's not right...

"Study it forever and you'll still wonder. Fly it once and you'll know."
- Henry Spencer

How to test?

Simplest: Run it and look at the output

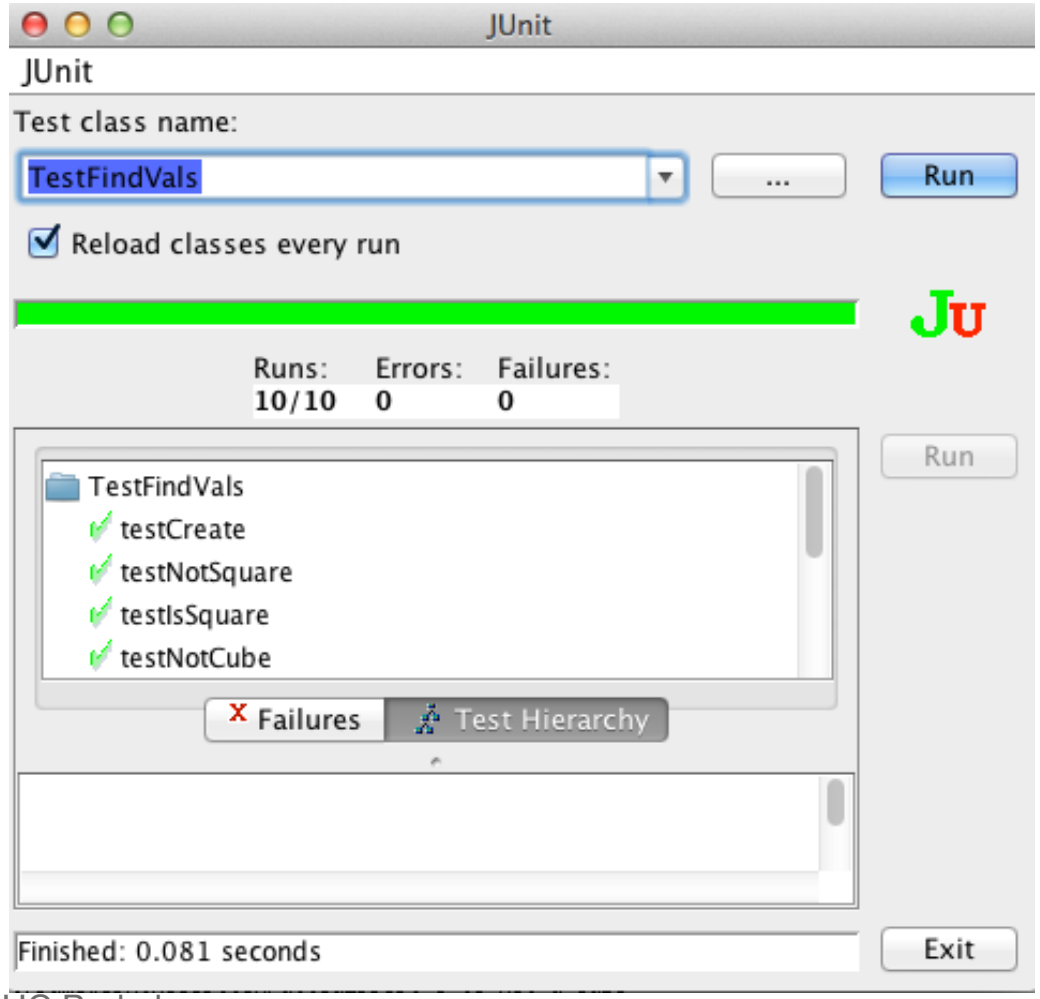
- Gets boring fast!
- How often are you willing to do this? Really carefully?

More realistic: Code test routines to provide inputs, check outputs

- Can become ungainly

Most useful: A test framework

- Great feedback
- Better control over testing



Testing Frameworks: CppUnit, Junit, et al

More

CERN School of Computing

Each time you write a function:

```
public class FindVals {
    // determine whether a number is a squared integer
    boolean isSquare(int val) {
       double root = Math.floor(Math.pow(val, 0.5));
       if (Math.abs(root*root - val) < 1.E-6 ) return true;
       else return false;
    }
}
```

You should write a test:

```
 public void testIsSquare() {
       FindVals s = new FindVals();
       Assert.assertTrue( s.isSquare(4) );
    }
```

Invoke a function

Check the result

Plus tests for other cases…

Embed that in a framework

Gather together all the tests

```
// define test suite
   public static Test suite() {
      // all tests from here down in heirarchy
      TestSuite suite = new TestSuite(TestFindVals.class);
      return suite;
   }
```

Junit uses class name to find tests

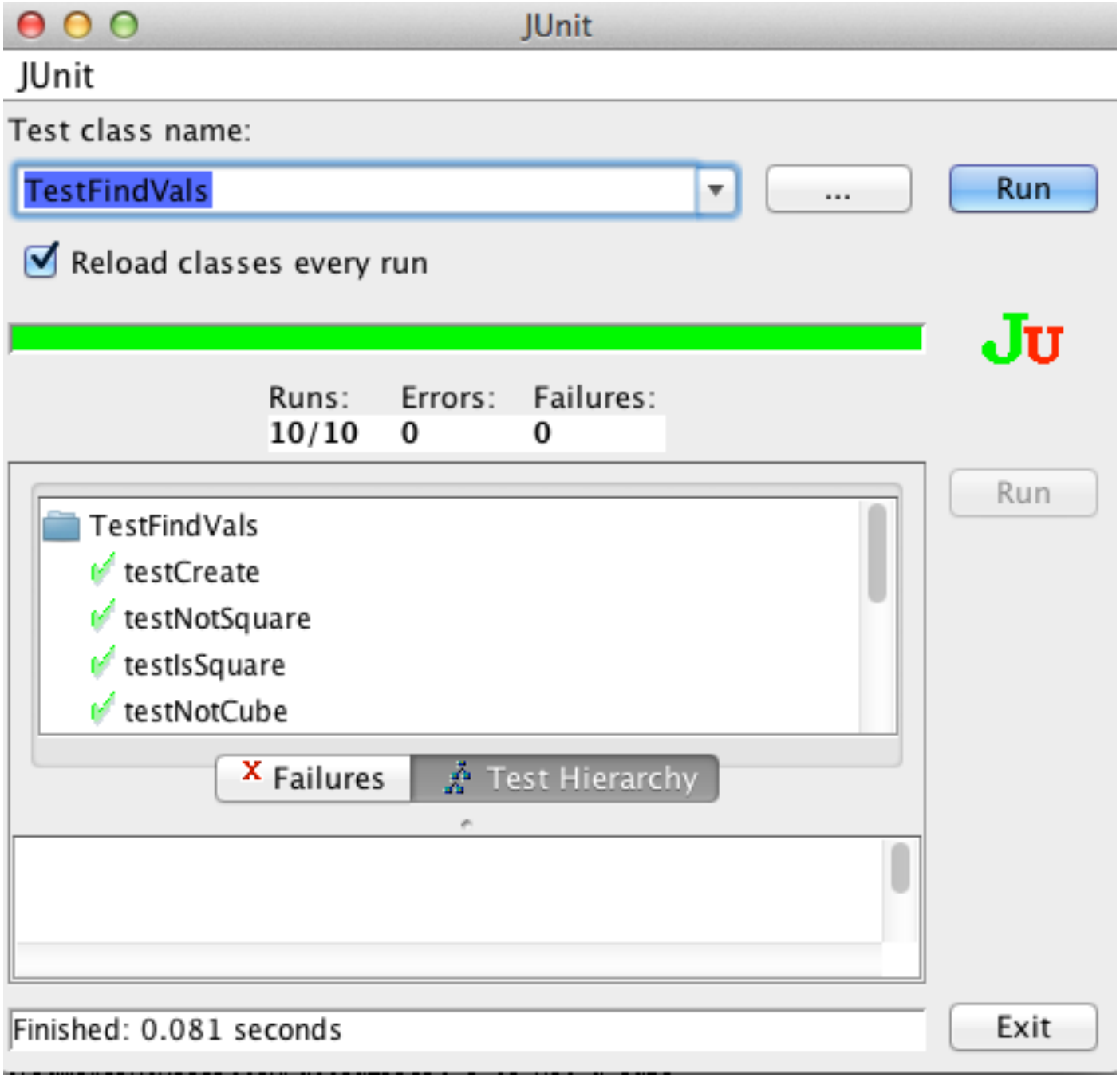
Start the testing

- To just run the tests:
  `junit.textui.TestRunner.main(TestFindVals.class.getName());`
- Via a GUI:
  `junit.swingui.TestRunner.main(TestFindVals.class.getName());`

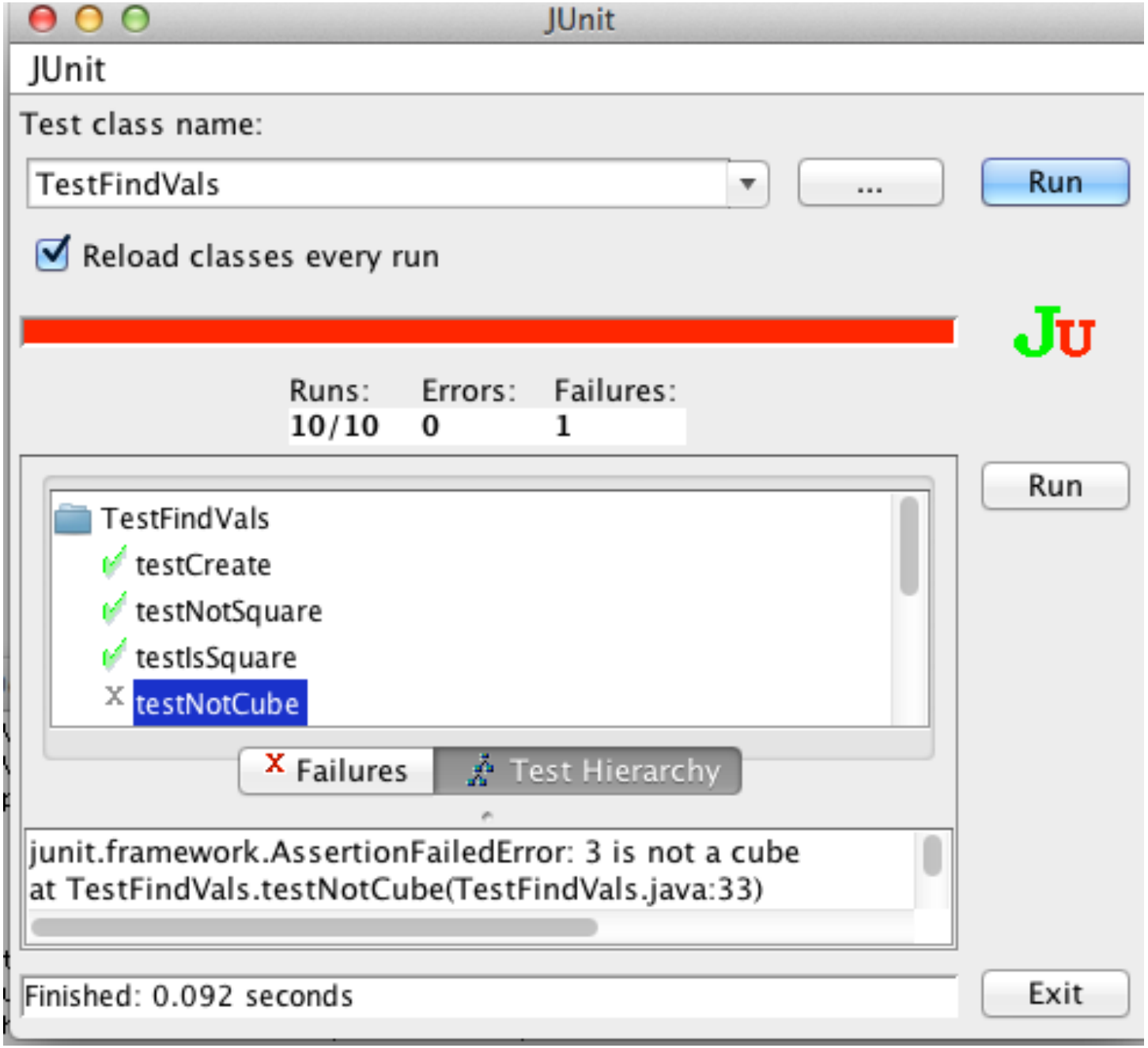
Invoke tests for my class

And that's it!

Running the tests



Running the tests



How JUnit works - one test:

```
public void testOneIsNotPrime() {
        SumPrimes s = new SumPrimes();
        Assert.assertEquals("check sumPrimes(1)", 0, s.sumPrimes(1));
}
```

This defines a "method" (procedure) that runs one test (line 1 and 4)

- JUnit treats as a test procedure any method whose name starts with "test"

Line 2 creates an object "s" to be tested

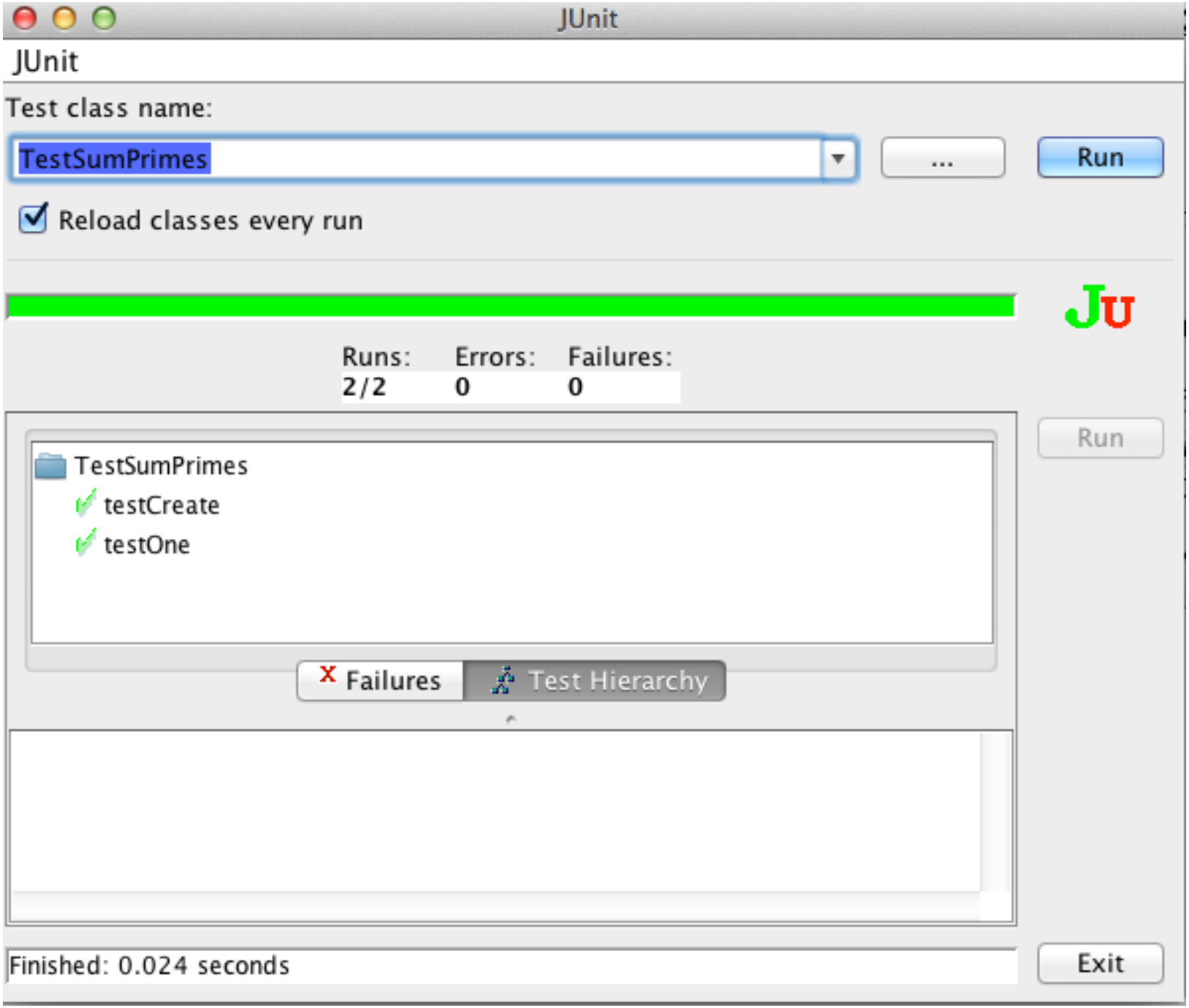
Line 3 checks that sumPrimes(1) returns a 0

Assert is a class that checks conditions

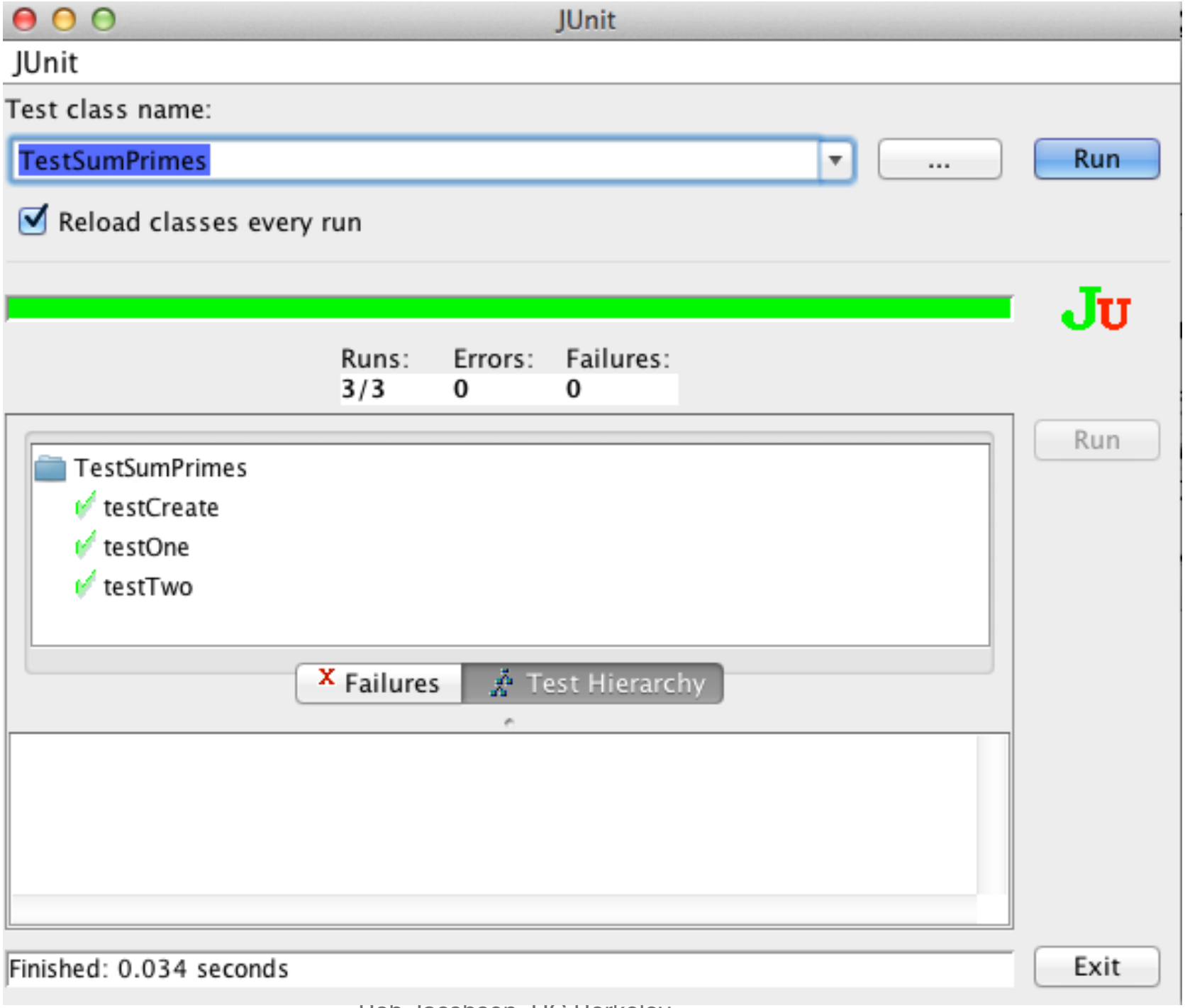
assertEquals("message", valueExpected, valueToTest) does the check

If the check fails, the message and observed values are displayed

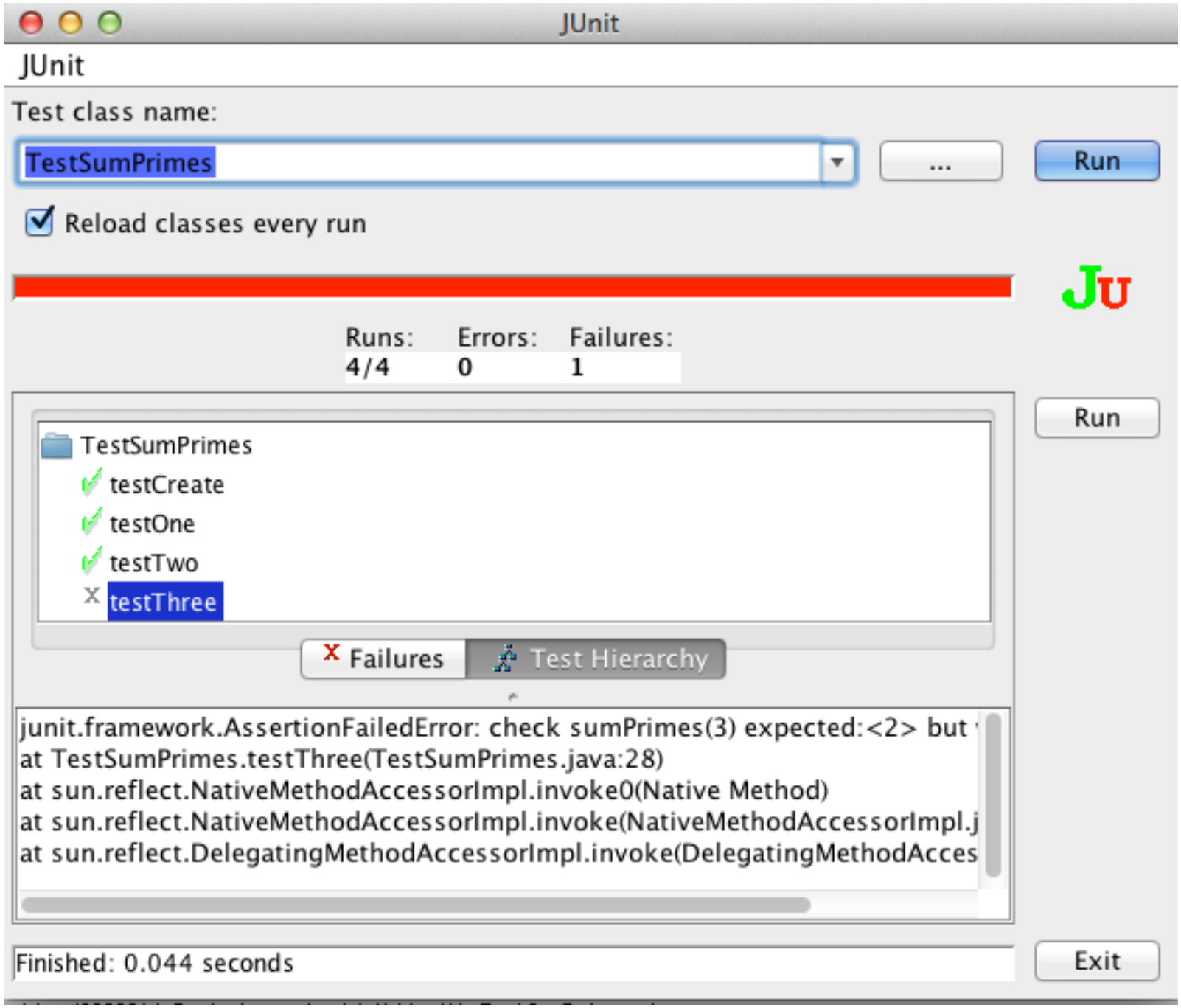
Some checks pass: sumPrimes(1) == 0



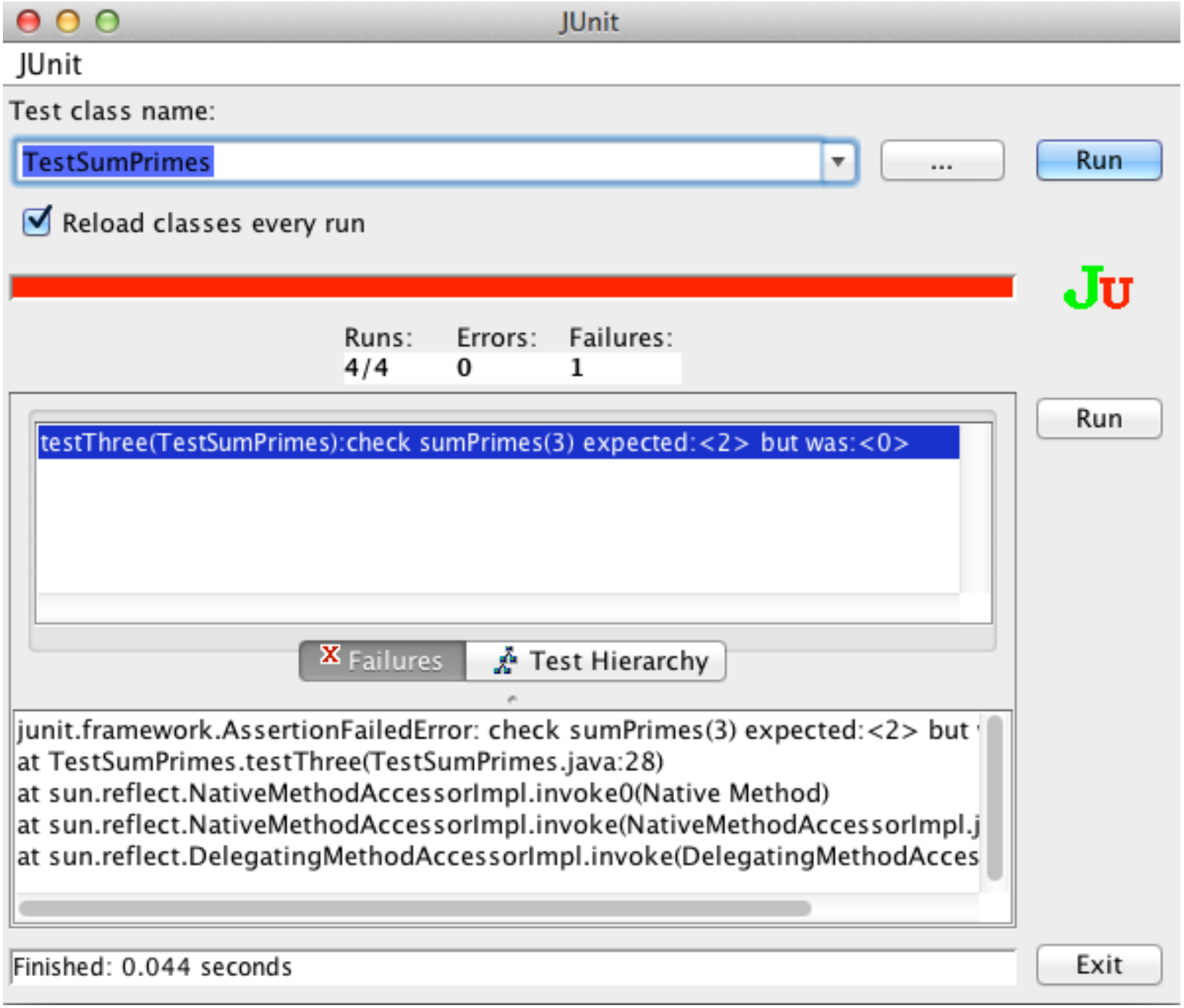
Add another: sumPrimes(2) == 0 (Why?)



And another: sumPrimes(3) == 2



Alternate view with more info:



Results of testing "SumPrimes"

Should "max" be included or not?

Its OK for a prime number to be divisible by one

```
int sumPrimes(int max) {
    int sum = 0;
    for ( int i=1; i < max; i++ ) {  // loop over possible primes
        bool prime = true;
        for (int j=1; j < 10; j++) { // loop over possible factors
            if (i % j == 0) prime = false;
        }
        if (prime) sum += i;
    }
    return sum;
}
```

If you divide a number by itself, the remainder is zero

Lesson 1: Its not easy to understand somebody else's code

- Assumptions, reasons are hard to see

  "Is one a prime number?"

  Test defines the behavior!! assertTrue(sumPrimes(2)==0); is that wanted?

Lesson 2: Better structure would have helped

- Separate "isPrime" from counting loop to allow separate understanding
- Make the algorithm for checking prime even clearer, easier to test separately

Why?

One test isn't worth very much

- Maybe saves you a couple seconds once or twice

But consistently building the tests as you build the code does have value

- Have you ever broken something while fixing a bug? Adding a feature?

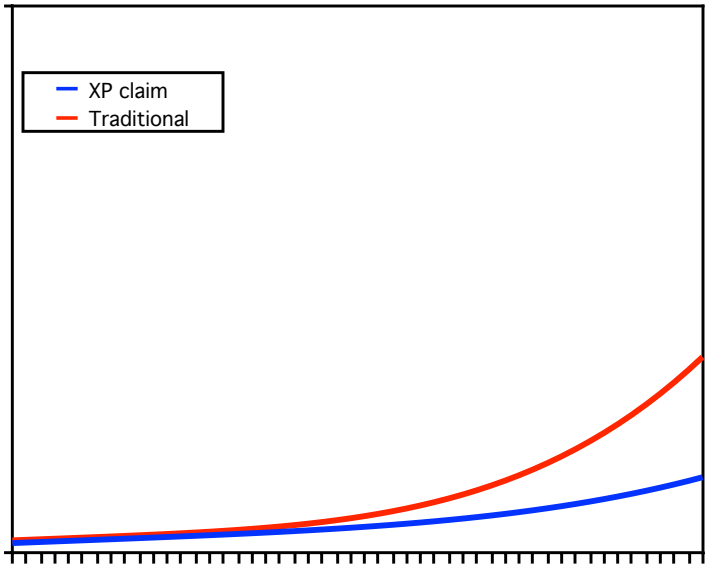
  Tests remember what the program is supposed to do
- A set of tests is definitive documentation for what the code does
- Alternating between writing tests and code keeps the work incremental

  Keeping the tests running prevents ugly surprises
- And it's very satisfying!

"Extreme Programming" advocates writing the tests before the code

More

- Not clear for large projects
- But individuals report excellent results



The art of testing

What makes a good test?

- Not worth testing something that's too simple to fail
- Some functionality is too complex to test reliably
- Best to test functionality that you understand, but can imagine failing

  If you're not sure, write a test

  If you have to debug, write a test

  If somebody asks what it does, write a test

How big should a test be?

- A *Unit test is a unit of failure

  When a test fails, it stops and moves to the next test

  The pattern of failures can tell you what you broke

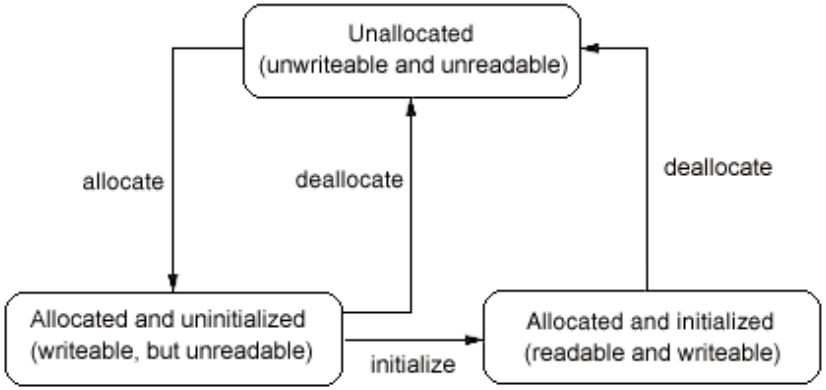
- Make lots of small tests so you know what still works

What about existing code?

- Probably not practical to sit down and write a complete set of tests
- But you can write tests for new code, modifications, when you have a question about what it does, when you have to debug it, etc

Avoiding memory problems - memprof et al

Memory-related problems



Read/write incorrectly

- Read from uninitialized memory
- Read/write via uninitialized pointer/ref
- Read/write past the valid range
- Read/write via a stale pointer/reference

  E.g. after deallocating memory

Memory management mistakes

- Deallocation of (currently) unowned memory

  Freeing something twice results in later overwrites
- Memory leaks

  Forgetting to free something results in unusable memory

More

Often cause "really hard to find" bugs

- Crashes, incorrect results - traceback, dump don't show cause
- Occur far from the real cause - breakpoints don't help
- Often intermittent

Note: Language choice reduces these, but doesn't make them go away!

A better allocator (malloc) can find some of these

Standard GNU malloc has a run-time checking option:

```
$ a.out
Segmentation fault (core dumped)
$ setenv MALLOC_CHECK_ 1
$ a.out
malloc: using debugging hooks
free(): invalid pointer 0x8049840!
```

Why not always leave it set?

- Checking slows program significantly
- Too many errors?

3rd party tools exist to do an even better job

```
Finished  a.out                ( 583 errors,  182 leaked bytes)
  Purify/PureCoverage instrumented a.out (pid 9499 at Mon Sep 23 14:07:32
  UMR: Uninitialized memory read (13 times)
    This is occurring while in:
      putHash        [hash.c:134]
          char* new_key;
          void* old_value;
          int index = hashIndex(key);
          for (last_entry = NULL, entry = ht[index];
               entry && strcmp(entry->key, key);
               last_entry = entry, entry = entry->next) {
          }
      testPutHash    [testHash.c:84]
```

Specialized tools - leak checking

Automated, unambiguous identification of leaks is difficult

- "forgot to free" vs "haven't freed yet" vs "program's ending, don't bother"
- "can no longer reference any part" vs "no references to the beginning"

But reading the code is not a reliable method either

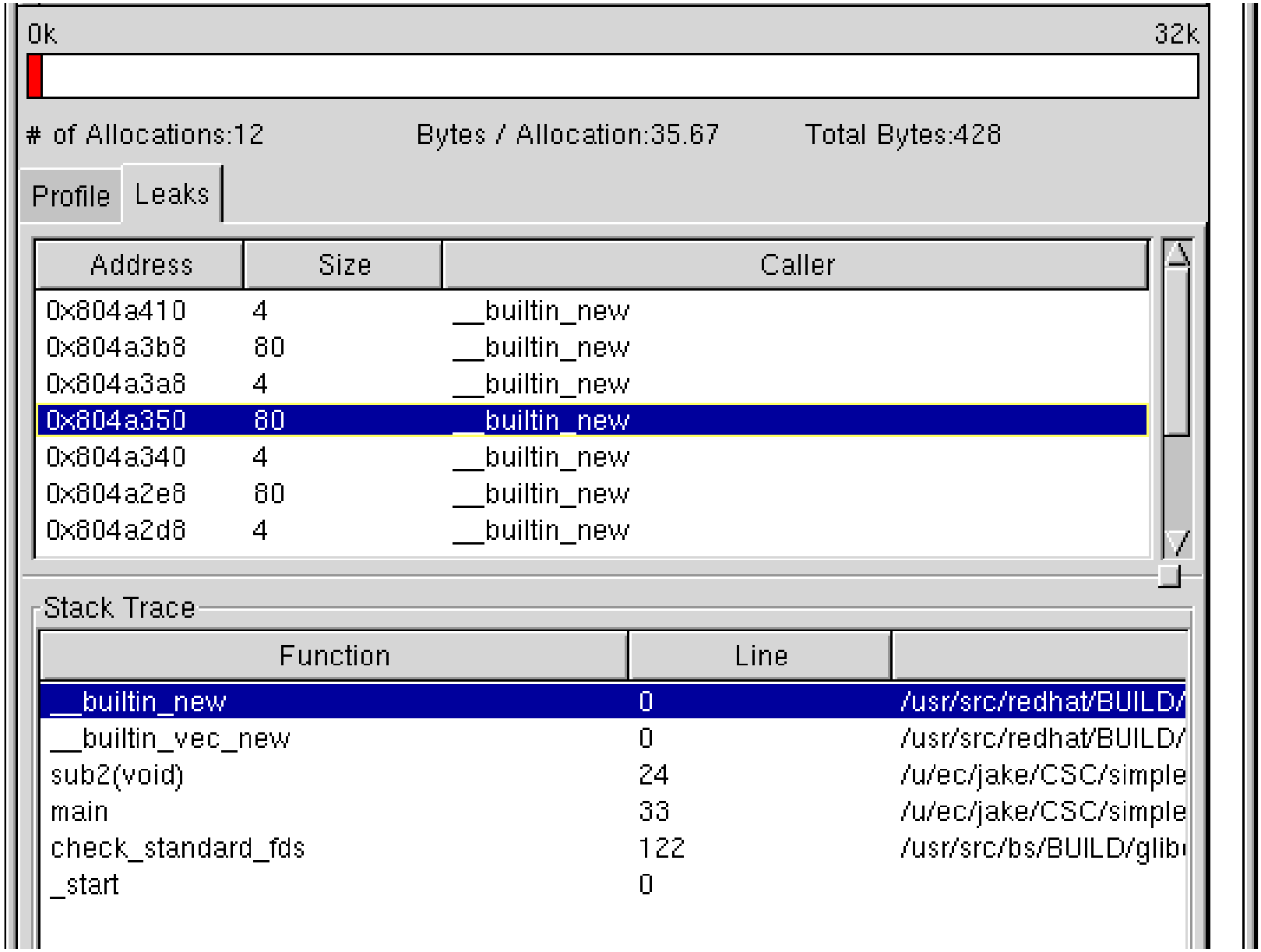
- A leak is a mistake of omission, not commission
- Often requires cooperation to leak memory:
  - Creator of allocated item may have no idea where it goes
  - Consumer may not realize responsible for deallocation
    - Doesn't need to be deallocated
    - Expects some third party to deallocate

Several approaches:

- "Print it all, and let the human sort it out"
- Provide a browser, let human reason about status of remaining memory
- Provide a suite of heuristics that can be tuned to the code's structure

Example: memprof

memprof replaces the allocation library at runtime, provides simple GUI



How do these actually work?

Replacement libraries

- E.g. a more careful malloc, perhaps automatically linked
- Can't check individual load/store instructions

Source code manipulation

- Preprocessor inserts instrumentation before compilation
  - Can know about scope, variable accesses, control flow
  - But requires source code, is language specific

Object code insertion

- Process object code to recognize & instrument load/store instructions
  - Can efficiently check every use of memory
  - Specific to both architecture and compiler, hard to port

Yes, you can write your own code to do some of this

But do you really want to spend the time to do it well?

A small catalog of available memory tools

More

CERN School of Computing

Free validity tests

- GNU C library - enable checking via MALLOC_CHECK_
- DMalloc - replacement library with instrumentation
- ElectricFence - checks for write outside proper boundaries
- AddressSanitizer - integrated with clang & gcc compiler to check operations
- valgrind - instruction-by-instruction checking

Free leak checkers

- Windows Leak Detector - runtime attach
- LeakTracer - compilation based
- Memprof
- MemCheck - part of Valgrind
- ccmalloc

Commercial code-check suites

- Purify (Rational Software)
- Insure (Parasoft)

How do you use these?

Big-bang approach is incredibly depressing

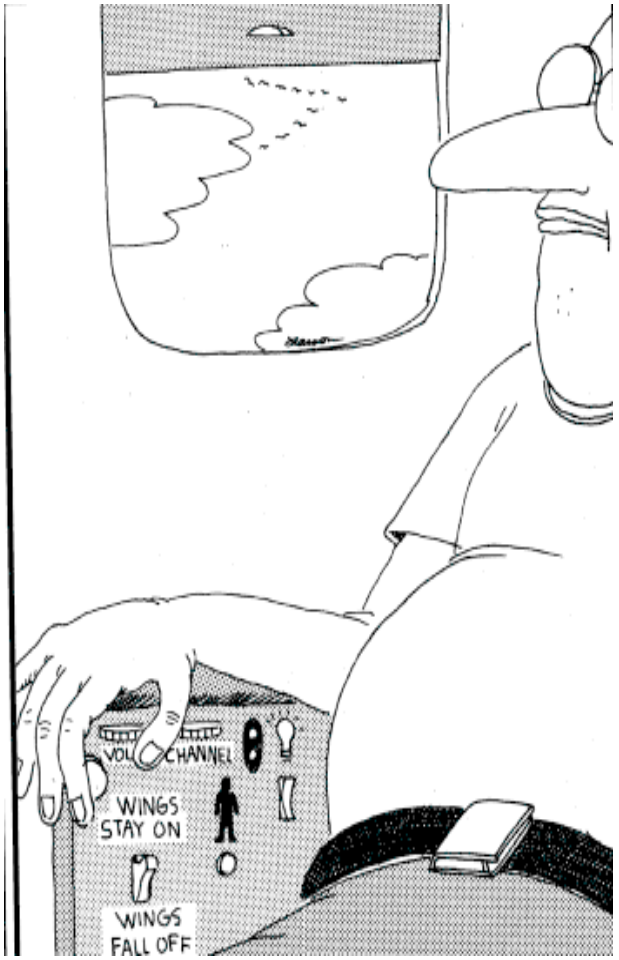
- Familiar products have lots of memory "errors"
- These swamp your own tiny efforts

Better: isolate your own code for initial checks

- Ties in with a test framework:
  "Does it work as expected?"
- Check often, fix incrementally

You still have to test "in the wild"

- Many errors are due to poor interfaces
- Learn from these and fix them!



When Data Arrives

Performance

More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason - including blind stupidity - W.A. Wulf

Perceived performance is what really matters

- Is the system getting the job done or not?
- Function of resources, efficiency, scope, etc.

Most people can only effect efficiency

- That's why people like to tune their programs to make them more efficient
- But it might not be the best way to get improvement

  People are expensive, often overloaded

But if you're going to tune a program, you might as well do a good job

Reminder: Performance assumes correctness!

- You have to make sure the program still works after you tune it

Start by understanding the problem

"Show me what part is taking all the time!"

Need tools to get reliable performance info

Several ways to acquire data

More

- Your OS probably has high-level tools for checking machine status
  - top, lsof, vmstat
  - Tools available vary with OS type
    - Linux tools: free, memalloc
    - MacOS X: vm_stat, lsof
- C/C++ have tools like gprof for internal program performance
- Java virtual machines can capture data at runtime

Several approaches:

- Periodic samples
  - Use the procedure stack in each sample to figure out what's being done
  - Use statistical arguments to provide profiles
- Tracking call/return control flow
  - Captures entire behavior, even for fast programs
  - Requires instrumenting the code
- Processor-based instrumentation

Sampling data looks like this:

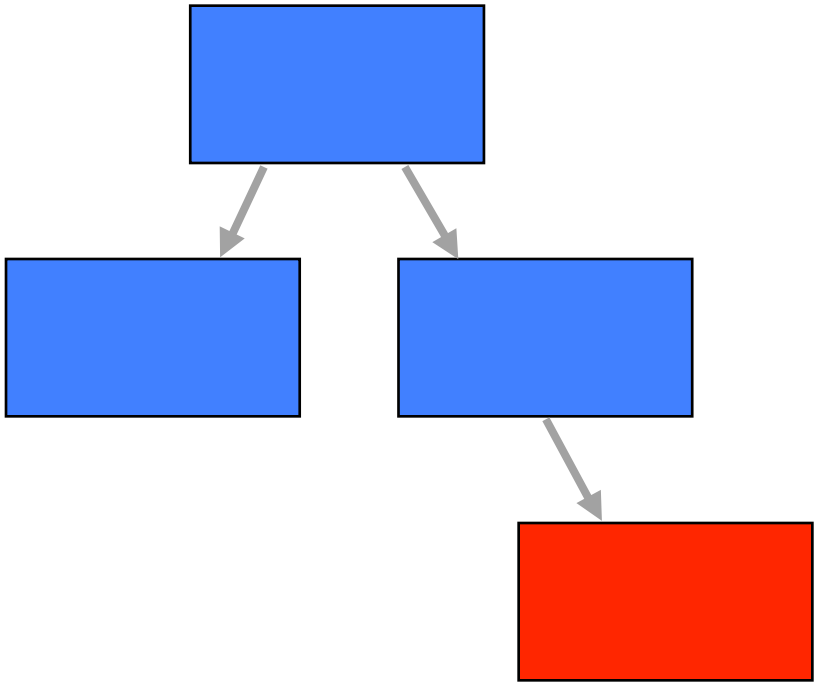
```
CPU SAMPLES BEGIN (total = 909) Sat Feb 12 13:45:46 2000
rank   self  accum   count trace method
   1 28.60% 28.60%     260    31 java/lang/StringBuffer.<init>
   2 26.51% 55.12%     241    18 java/lang/StringBuffer.<init>
   3 24.42% 79.54%     222    48 java/lang/StringBuffer.<init>
   4  4.62% 84.16%      42    21 java/lang/System.arraycopy
   5  3.96% 88.12%      36    49 java/lang/System.arraycopy
   6  3.85% 91.97%      35    36 java/lang/System.arraycopy
   7  0.66% 92.63%       6    33 com/develop/demos/TestHprof.makeStringInline
   8  0.44% 93.07%       4    47 java/lang/String.getChars
   9  0.33% 93.40%       3    23 java/lang/StringBuffer.toString
  10  0.22% 93.62%       2    25 java/lang/StringBuffer.append
  11  0.22% 93.84%       2    59 com/develop/demos/
TestHprof.makeStringWithBuffer
  12  0.22% 94.06%       2    50 com/develop/demos/TestHprof.makeStringWithLocal
  13  0.22% 94.28%       2    40 java/lang/StringBuffer.toString
  14  0.22% 94.50%       2    17 com/develop/demos/TestHprof.addToCat
  15  0.22% 94.72%       2    41 java/lang/String.<init>
  16  0.22% 94.94%       2    30 java/lang/StringBuffer.append
  17  0.22% 95.16%       2     7 sun/misc/URLClassPath$2.run
```

Now what?

Now what?

What you have: How often some function was running

What you want: "Improve this place first"



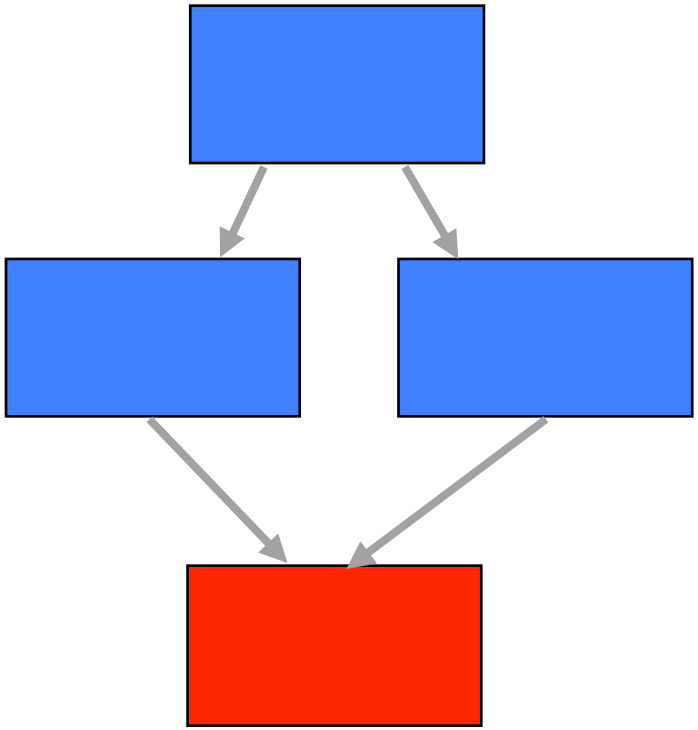
Is this asking for too much work?

Is this a poor algorithm?

Now what?

What you have: How often some function was running

What you want: "Improve this place first"



Who's responsible for all this work?

Tools to help understand performance info

Commercial performance tools tend to have powerful analysis features

- This is why people are willing to pay so much for them...

PerfAnal as an low-end example for exercises

Good for teaching, but better tools exist for real use

More

Gives four views of the program behavior

- Top down look

  How is each routine spending its time

- Bottom up look

  Who is asking this routine to spend time?

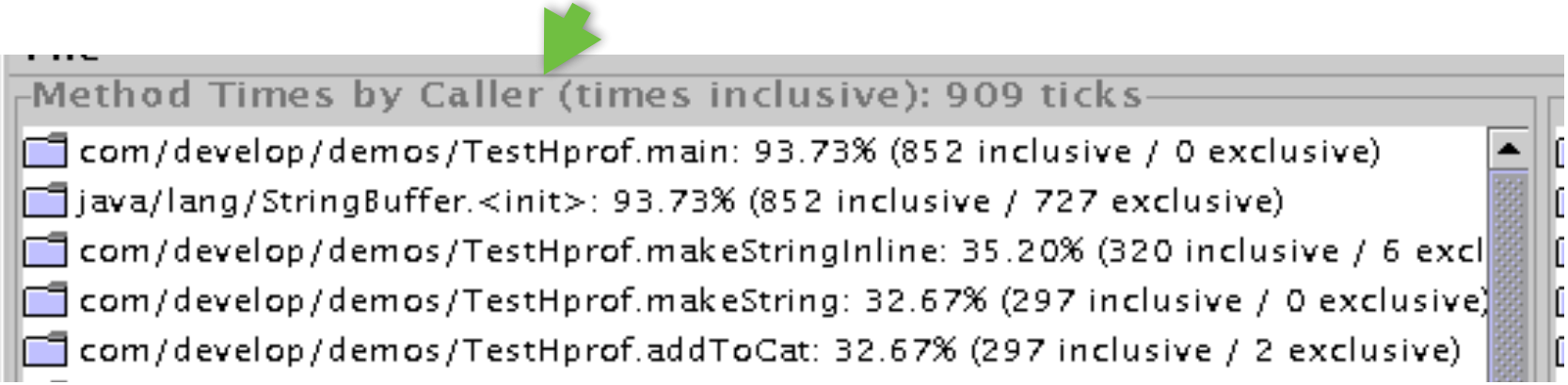
- Detail within each function by line number

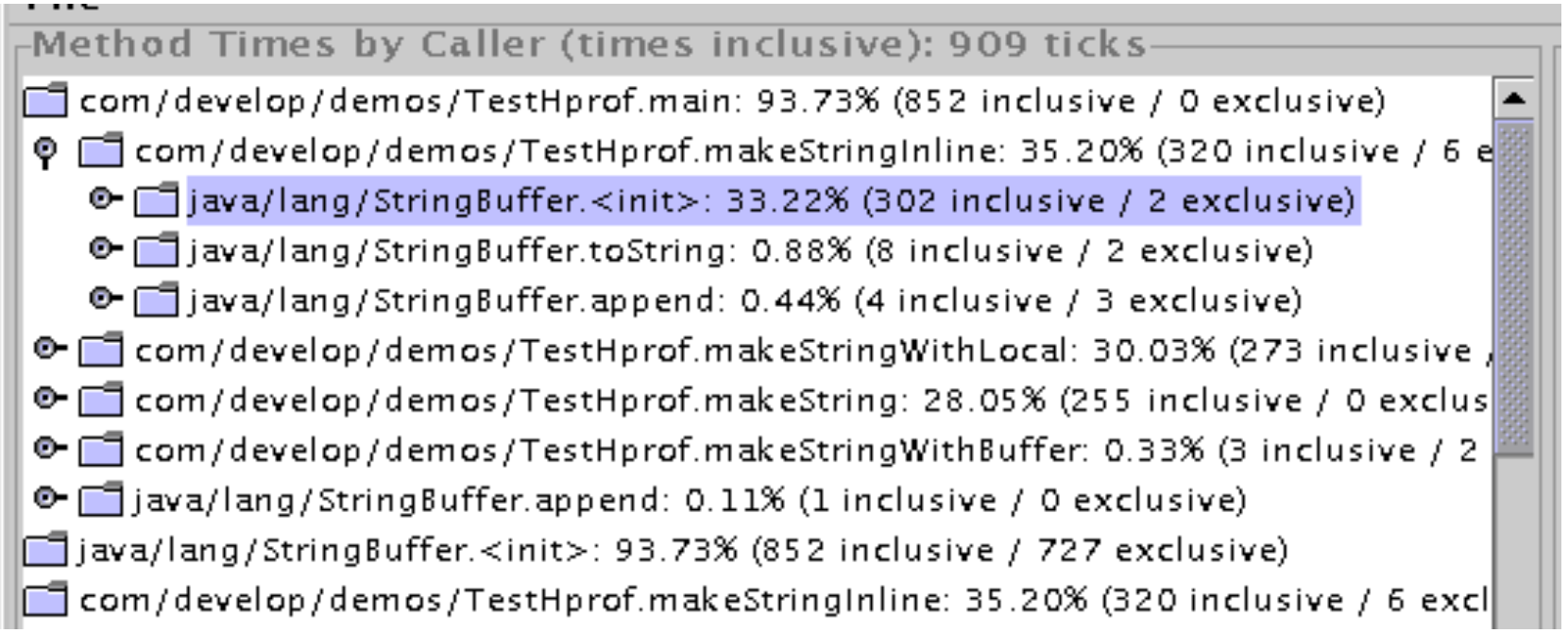
  How is time spent in each function, with/without calls to others?

  Is there just some bad code in there?

Top-down view of the program

How is the routine spending its time?



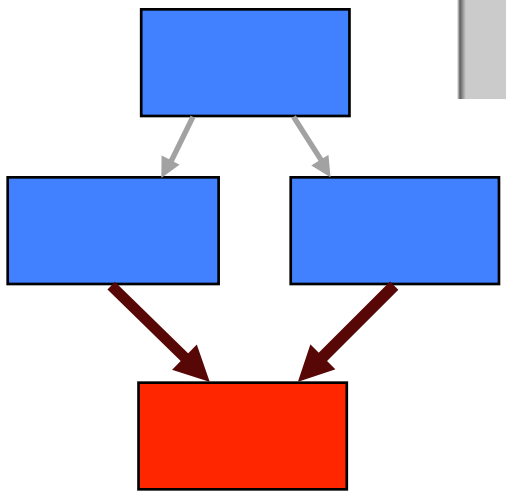


Bottom-up view

Who is asking this routine to spend time?

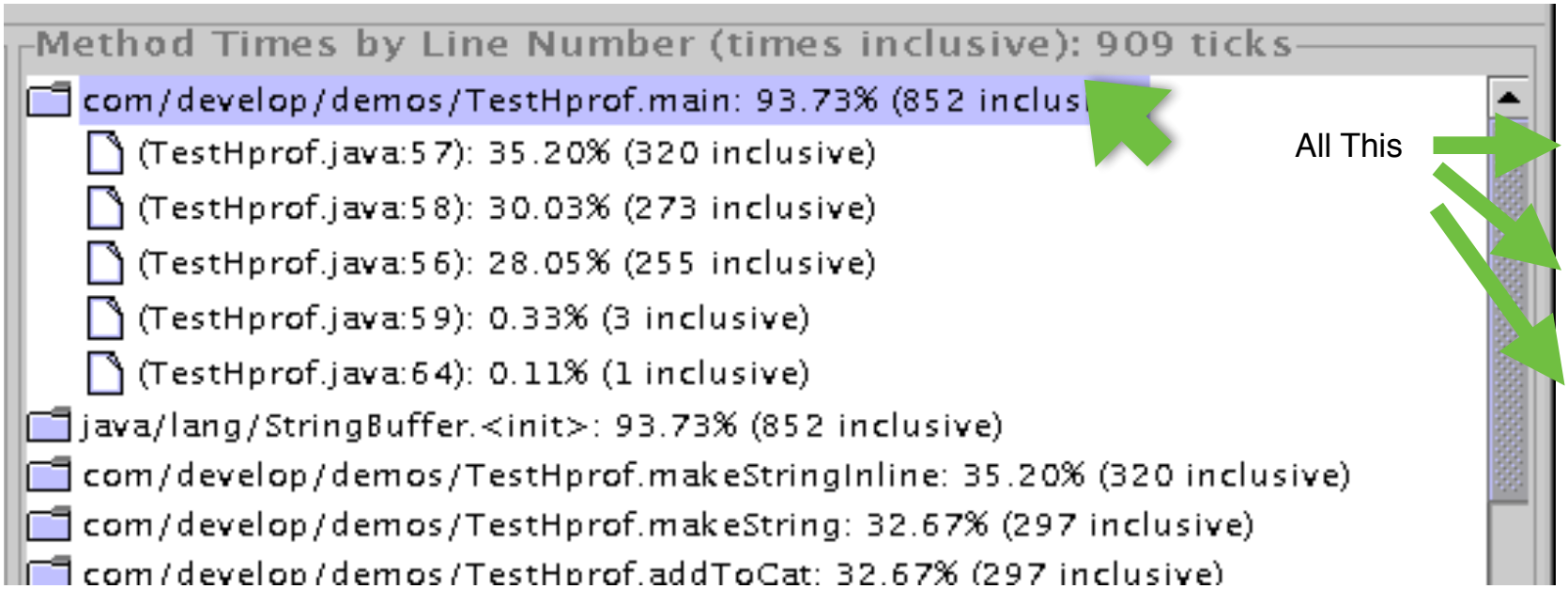
Method Times by Callee (times inclusive): 909 ticks

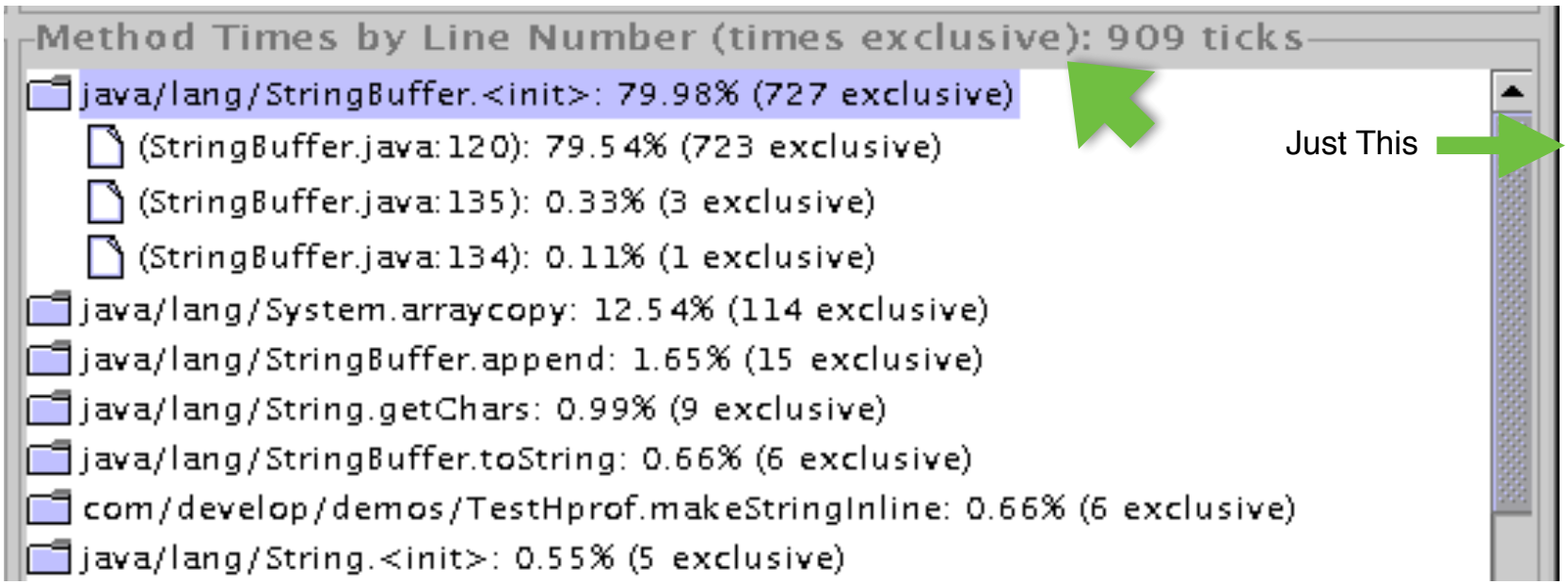
- com/develop/demos/TestHprof.main: 93.73% (852 inclusive)
- java/lang/StringBuffer.<init>: 93.73% (852 inclusive)
  - com/develop/demos/TestHprof.makeStringInline: 33.22% (302 inclusive)
  - com/develop/demos/TestHprof.addToCat: 31.79% (289 inclusive)
    - com/develop/demos/TestHprof.makeString: 31.79% (289 inclusive)
  - com/develop/demos/TestHprof.makeStringWithLocal: 28.71% (261 inclusive)
- com/develop/demos/TestHprof.makeStringInline: 35.20% (320 inclusive)
- com/develop/demos/TestHprof.makeString: 32.67% (297 inclusive)
- com/develop/demos/TestHprof.addToCat: 32.67% (297 inclusive)
- com/develop/demos/TestHprof.makeStringWithLocal: 30.03% (273 inclusive)
- java/lang/StringBuffer.append: 15.73% (143 inclusive)
- java/lang/String.getChars: 13.53% (123 inclusive)
- java/lang/System.arraycopy: 12.54% (114 inclusive)
- java/lang/StringBuffer.toString: 1.43% (13 inclusive)



Even more detail…

Within a member function





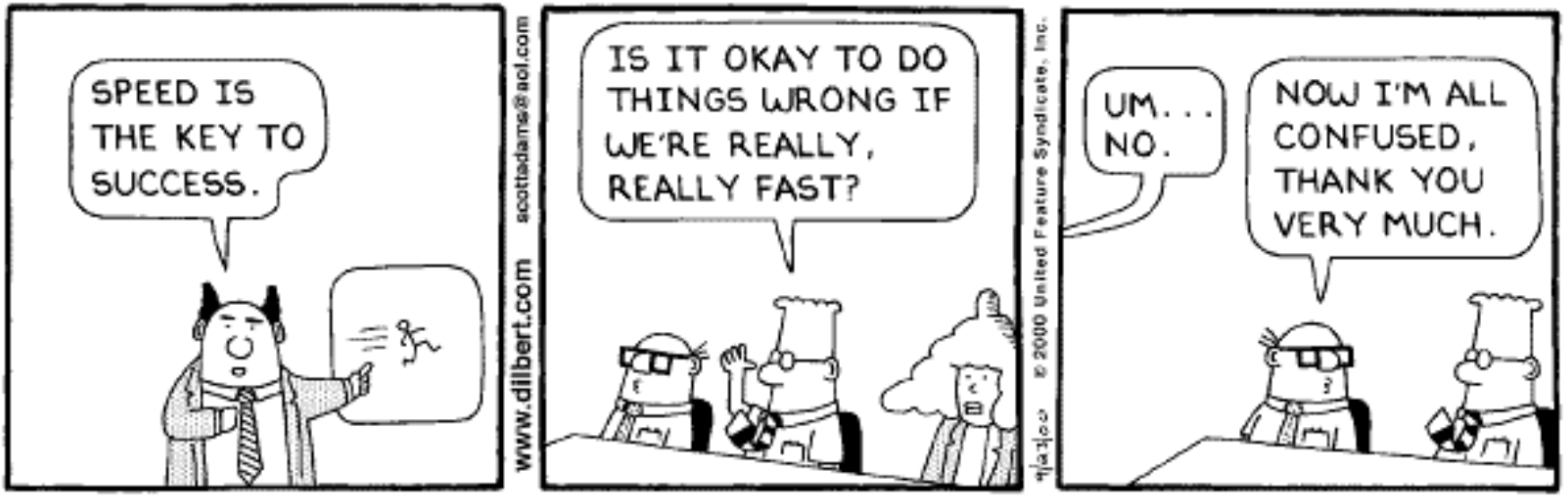
How do you use this?

Two approaches:

- Make often-used routines faster
- Call slow routines less often

But it has to stay correct!

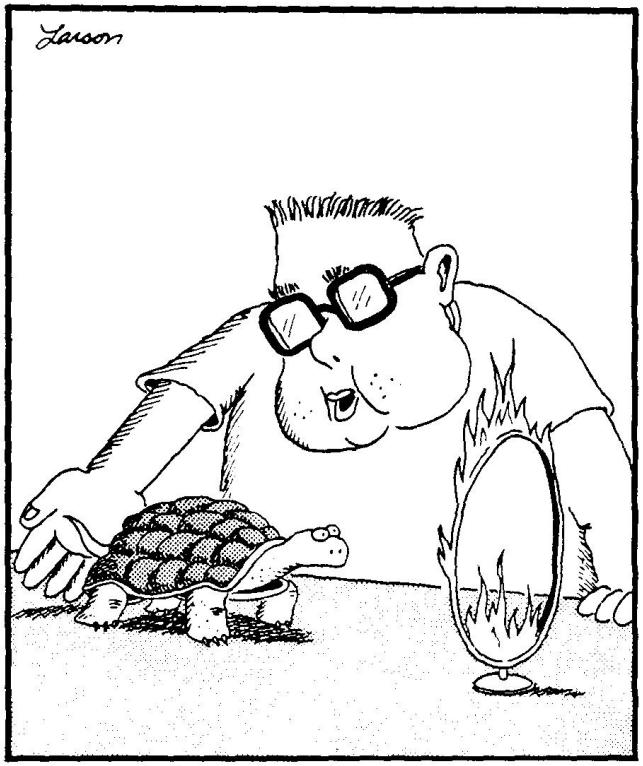
- Start by working in small steps



Sometimes you need to think big thoughts

Not all problems will be solved with an incremental approach

- "Do we have to do this?"
- "Is there a better way to do this?"



"Through the hoop, Bob! Through the hoop!"

Traditional example: Sorting a new deck of cards

More

Method 1: Pattern recognition

- There are a finite number of possible arrangements
- Find which one you have, and then reorder
- 52! = $4x10^{66}$ so will need about $52/2*4x10^{66}/2$ comparisons

Method 2: Bubble sort

- Scan through, finding the smallest number
- Then repeat, scanning through the N-1 that's left
- Cost is $O(N^2)$ "sum of numbers from 1 to N" = $52*(52+1)/2 = 1.4x10^3$

Method 3: Better sorts - Shell sort, syncsort, split sort, ...

- Even for arbitrary data, better sort algorithms exist
- O(N logN) = k * 52 * 5.7 = k * 300, where "k" is time per operation
- For N large, important gain regardless of k
- As ideas improve, k has come down from 5 to about 1.2 = 360

Method 4: Bin sort ("Solitaire sort")

- Use knowledge that there are 52 specific items
- Throw each card into the right bin with 52 calculations

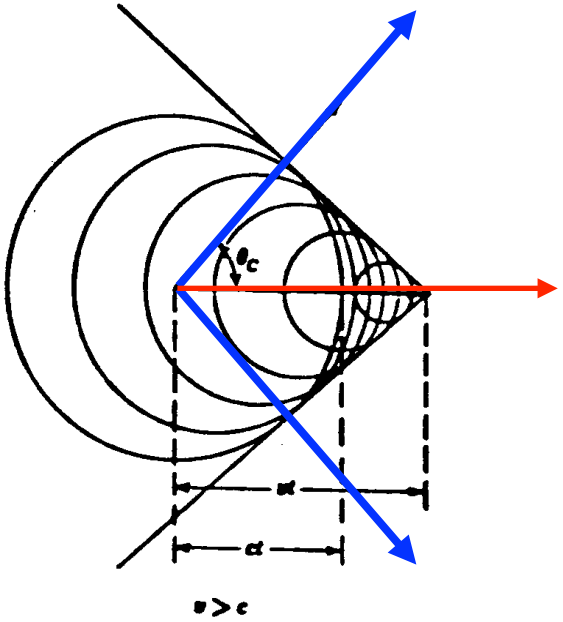
Method 5: New decks are already sorted (No operations!)

Telling pions from kaons via Cherenkov light

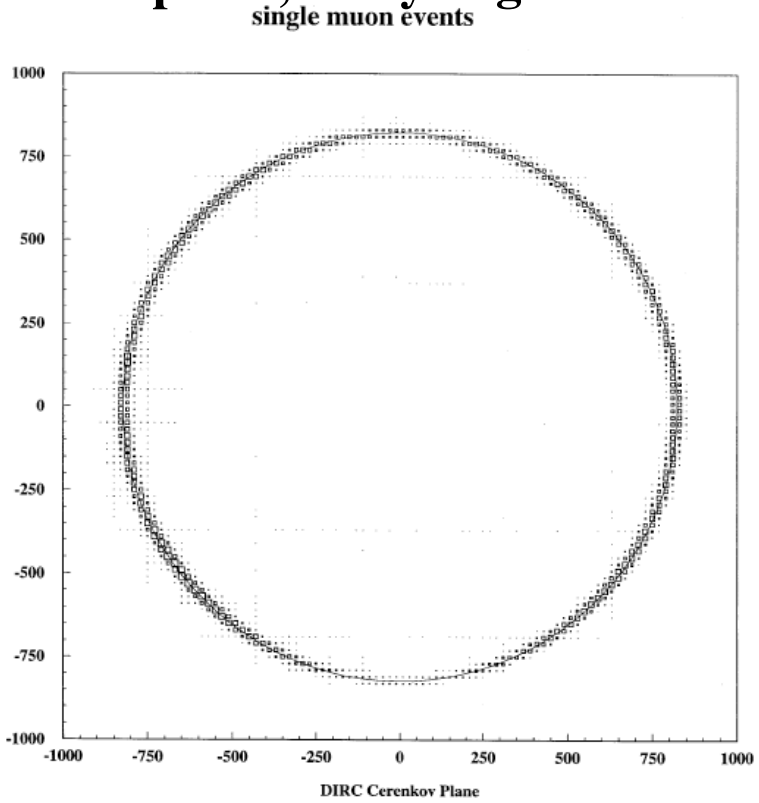
Pions & Kaons have similar interactions in matter, differ in mass

Particles moving faster than light in a medium (glass, water) emit light

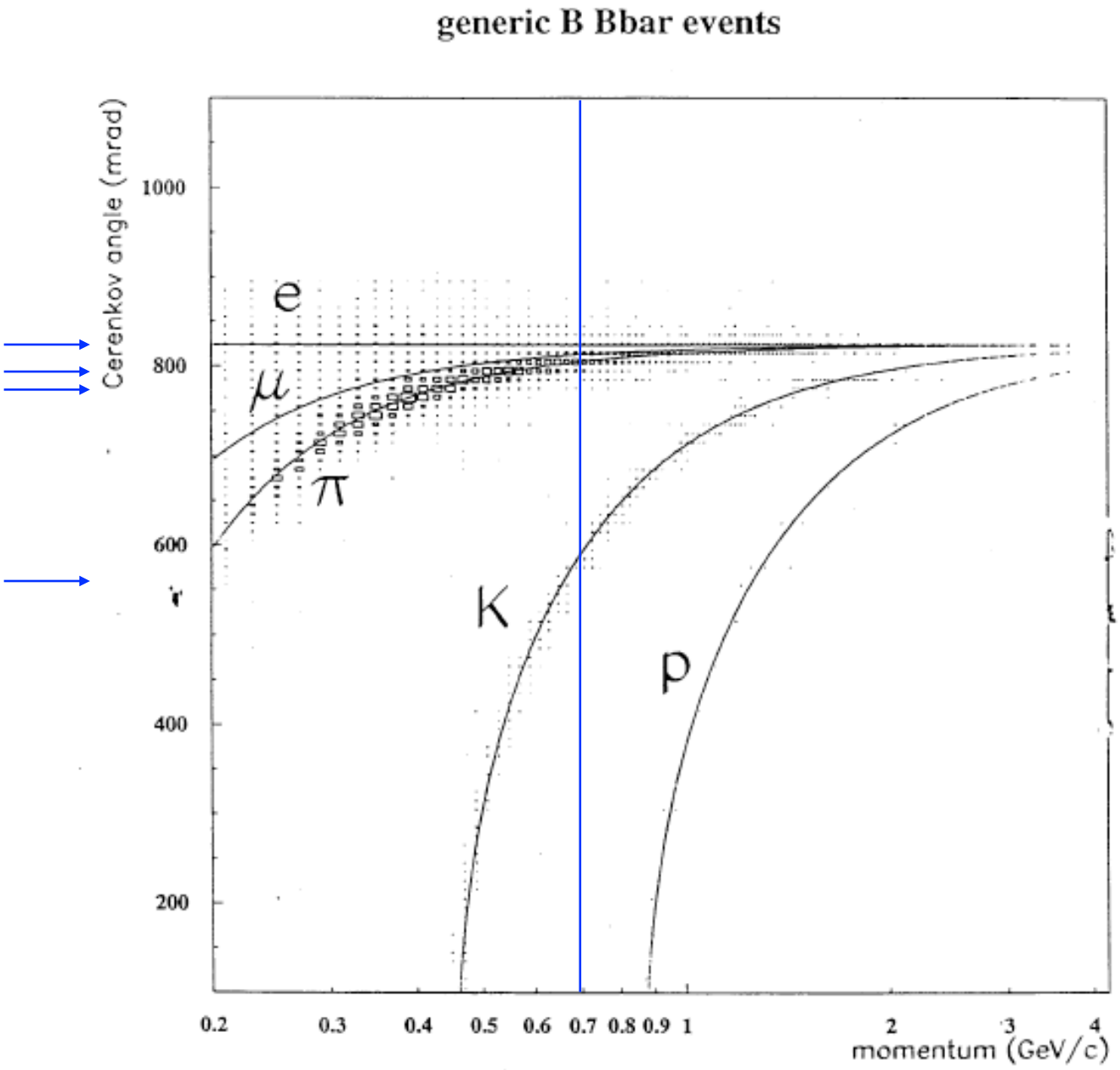
- Angle is related to velocity
- Light forms a cone



Focus it onto a plane, and you get a circle:



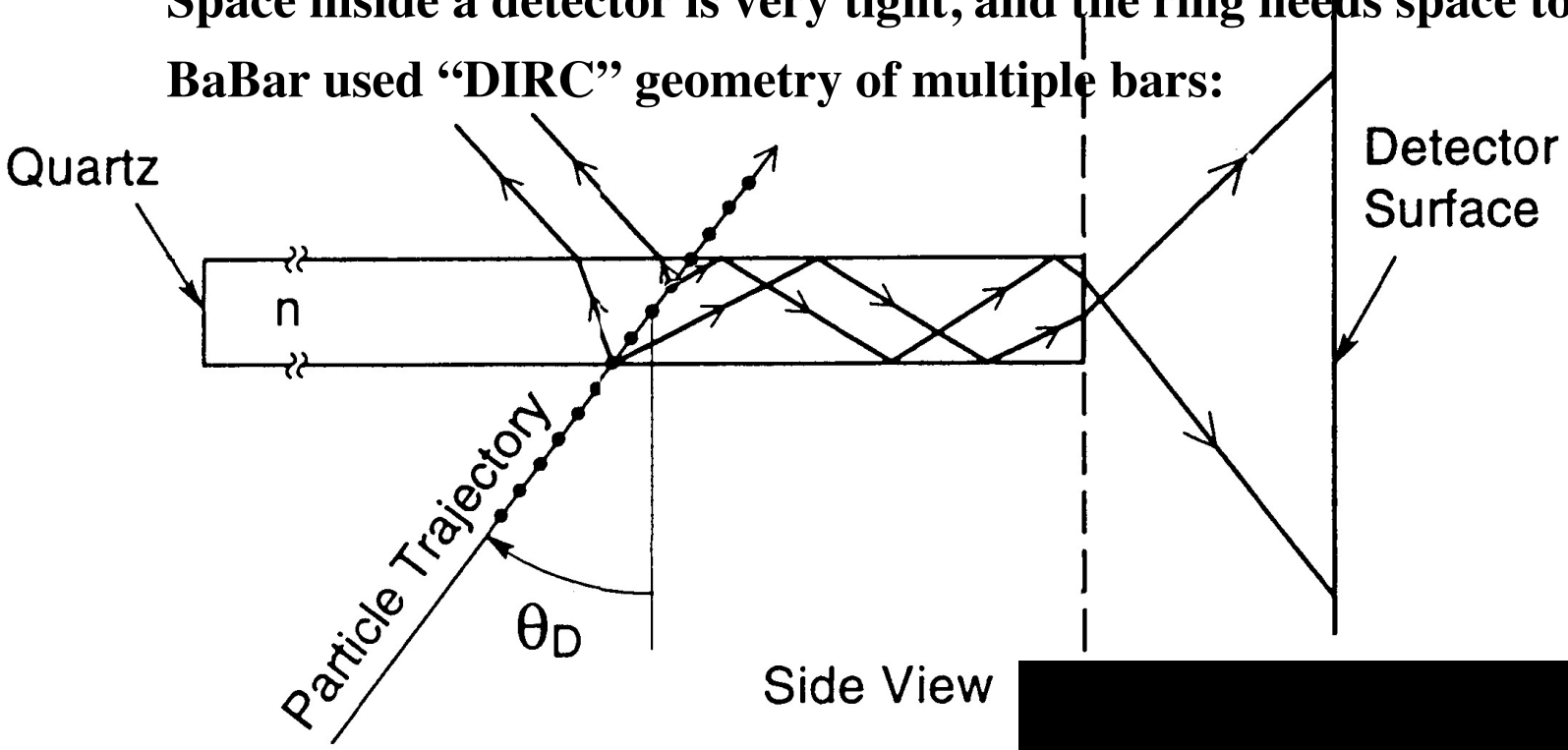
Radius of the reconstructed circle give particle type:

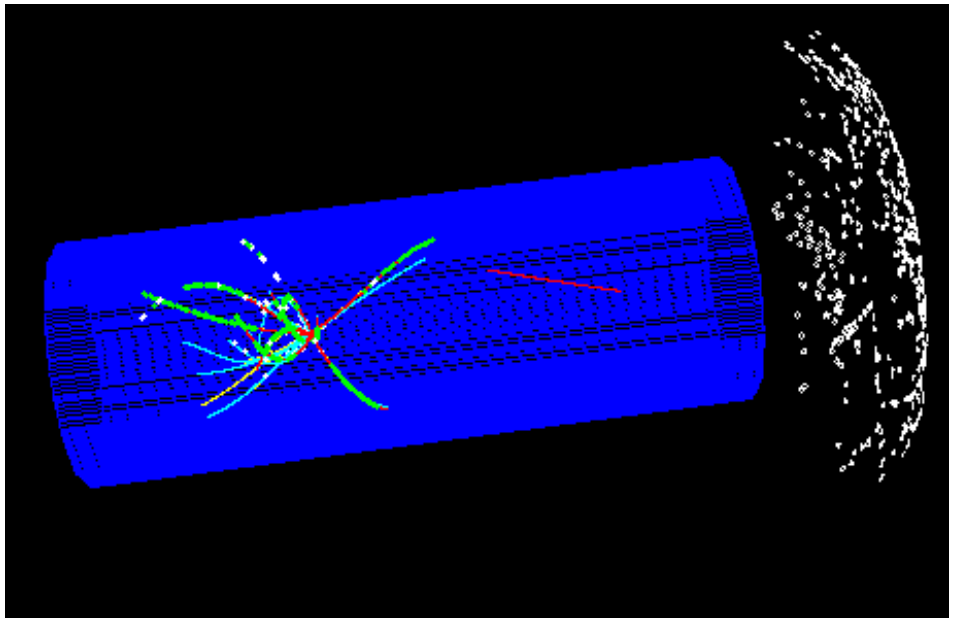


How to make this fit?

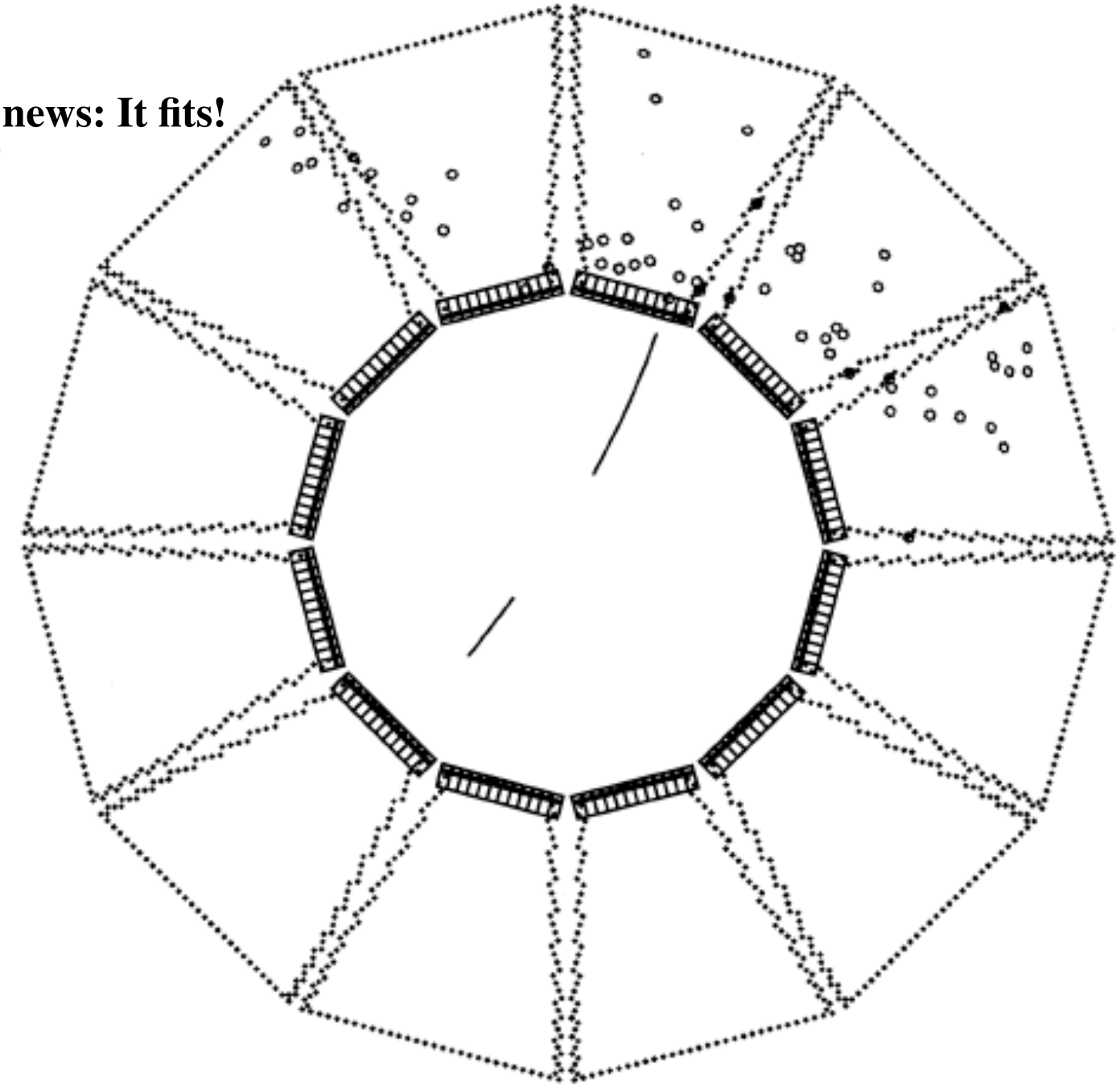
Space inside a detector is very tight, and the ring needs space to form

BaBar used "DIRC" geometry of multiple bars:





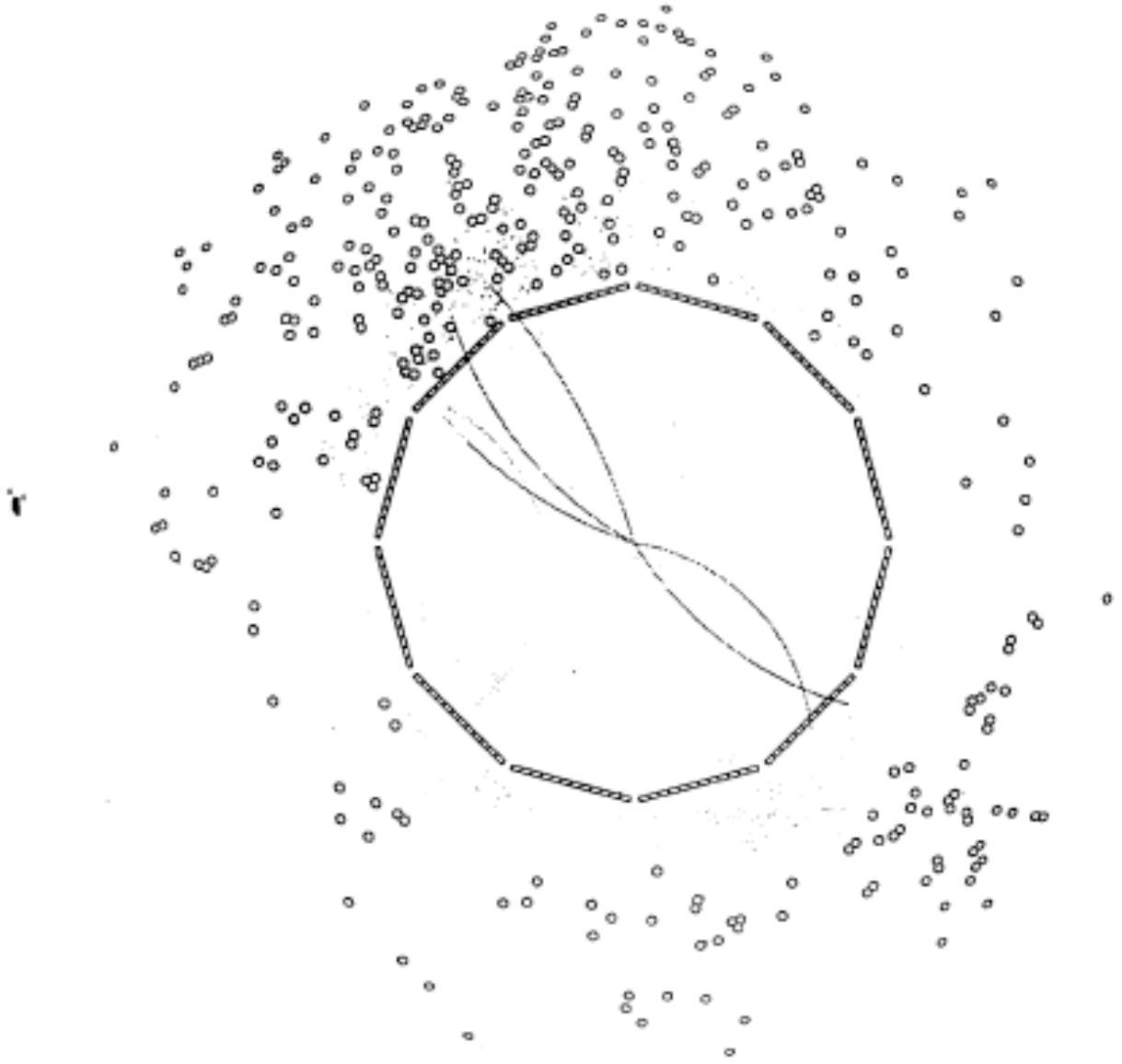
Good news: It fits!



Bad news: Rings get messy due to ambiguities in bouncing

Simple event with five charged particles:

“I don’t want to do this”

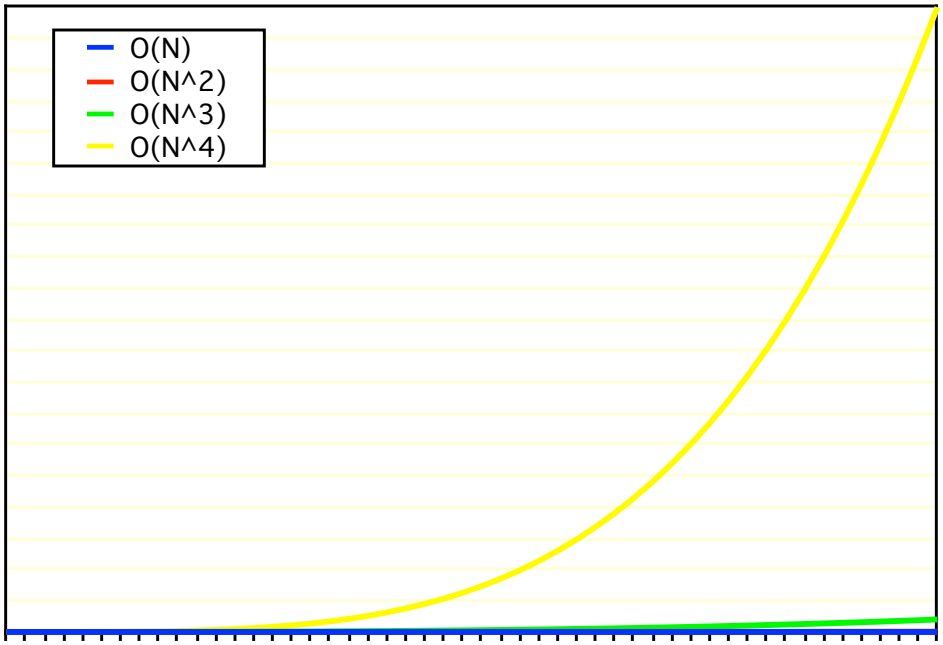


Why is this hard?

Brute-force circle-finding is an O(N⁴) problem

- Basic algorithm: Are these four points consistent with a 'circle'?

We catalog algorithms by how their cost grows with input size: O(N)



Realistic solution for DIRC? (Avoiding O(N⁴))

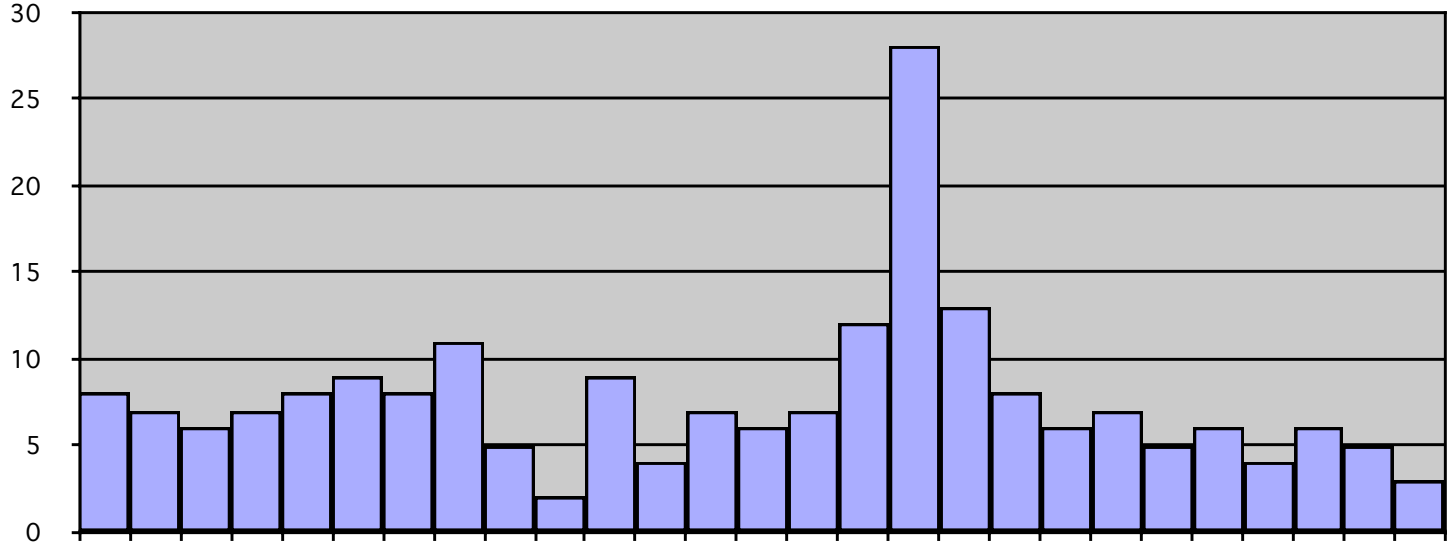
Use what you know:

- Have track trajectories, know position and angle in DIRC bars
- All photons from a single track will have the same angle w.r.t. track

  No reason to expect that for photons from other tracks

For each track, plot angle between track and every photon - O(N)

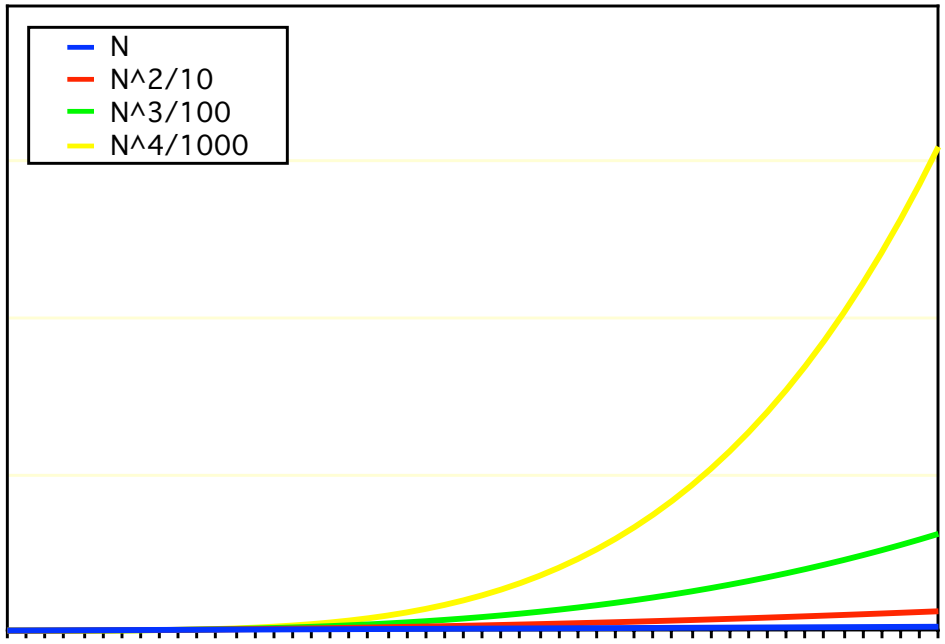
- Don't do pattern recognition with individual photons
- Instead, look for overall pattern you already know is present



Not perfect, but optimal?

"But each operation is so much slower…"

How do I compare a "fast" O(N⁴) algorithm with a slow O(N)?

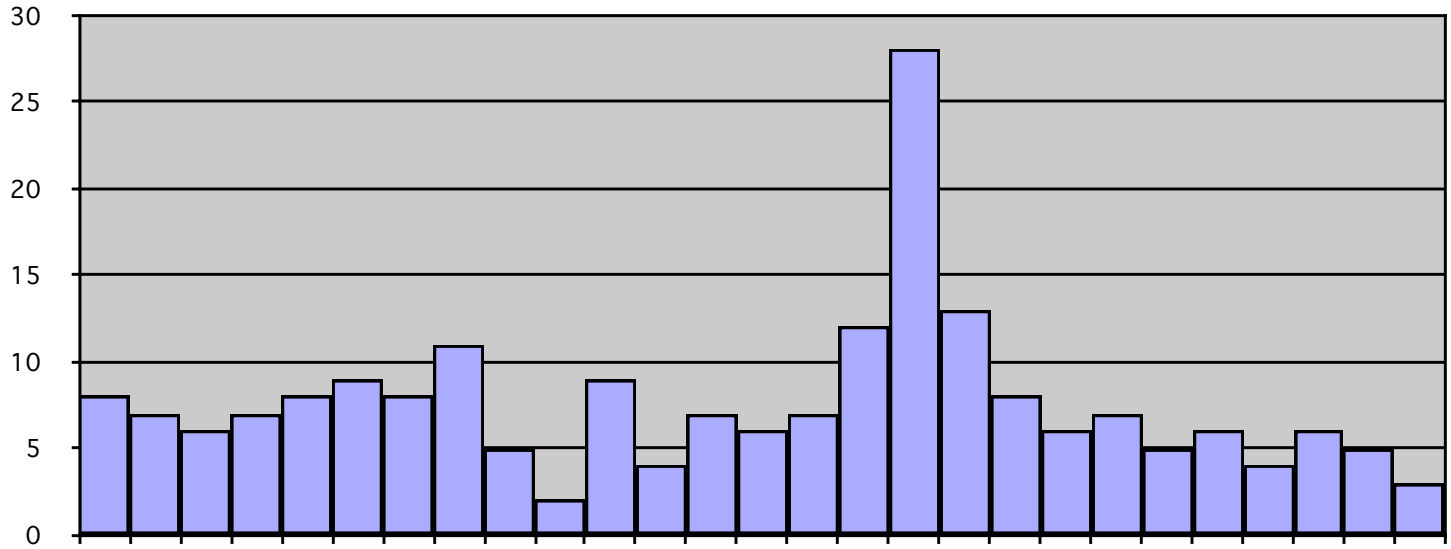


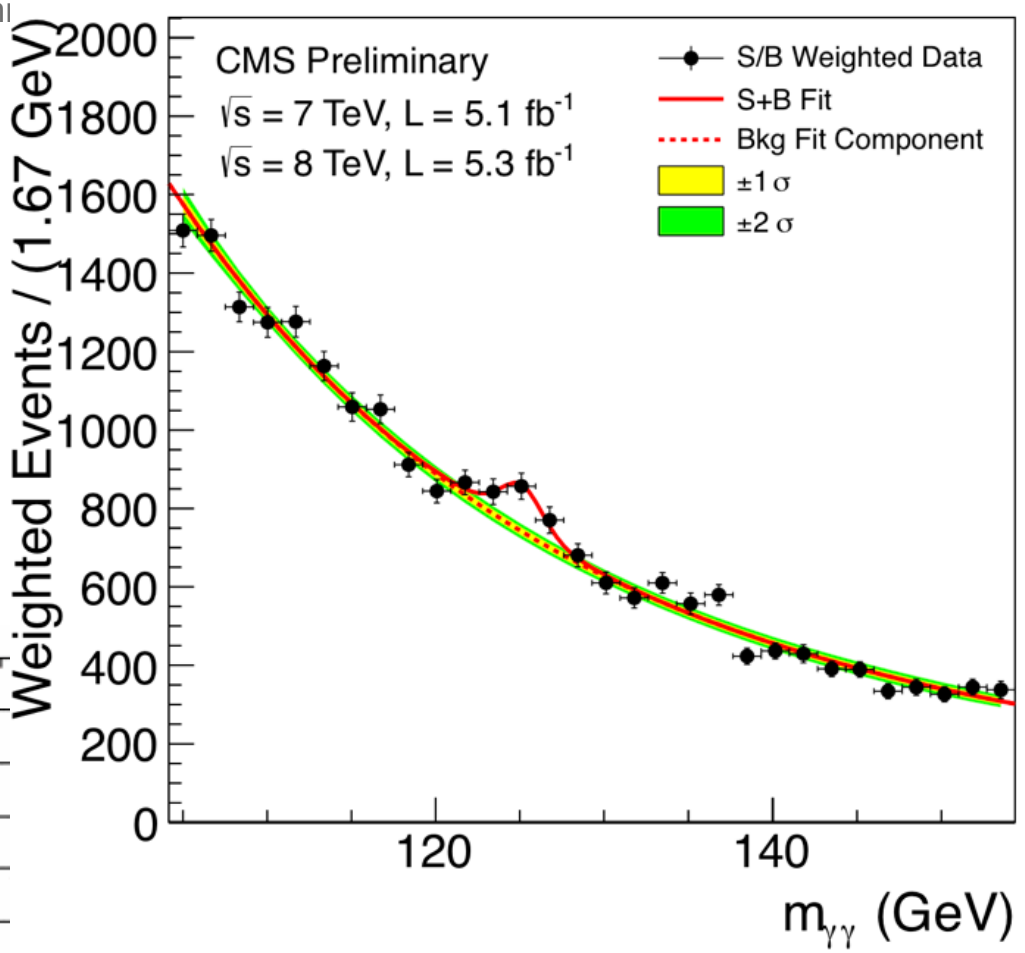
Many realistic problems deal with lots of data items

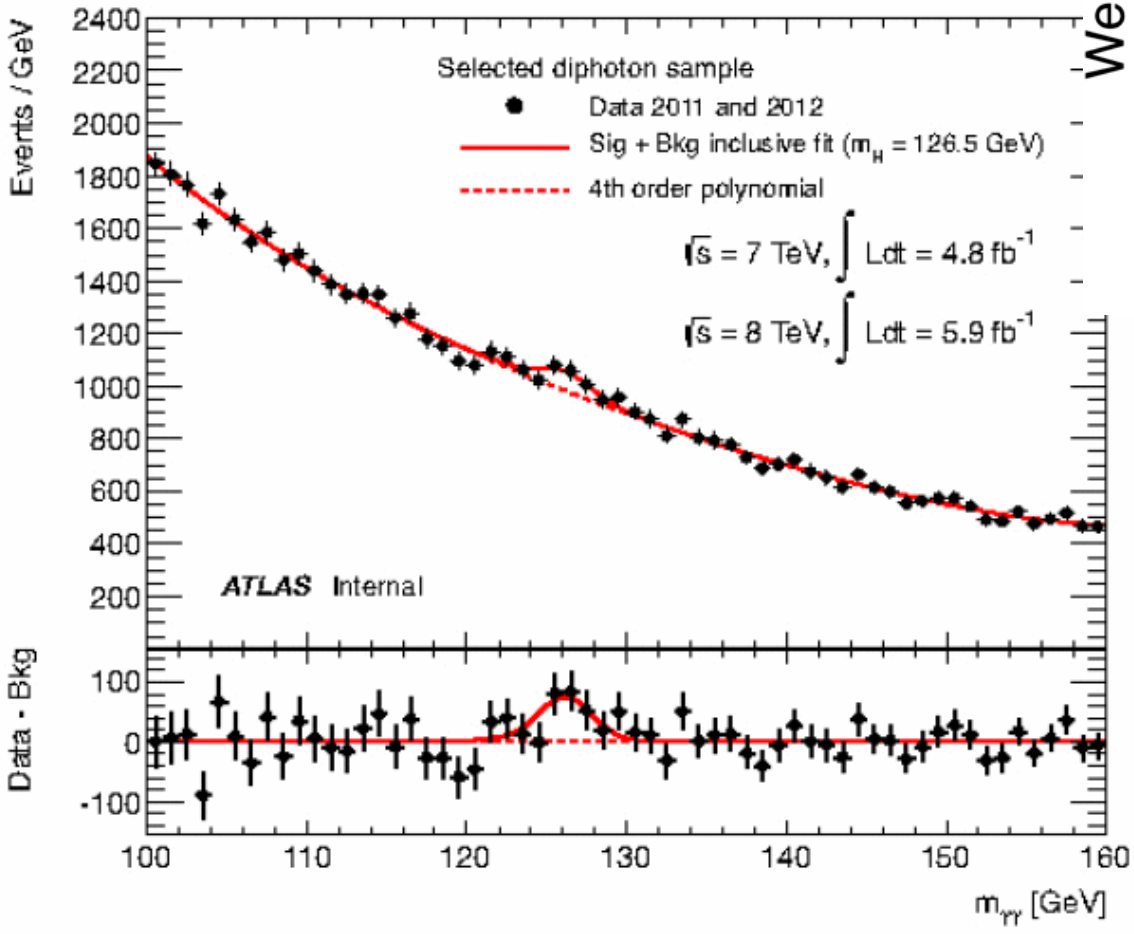
- Sharp coding is unlikely to save you a factor of 50^2 per calculation

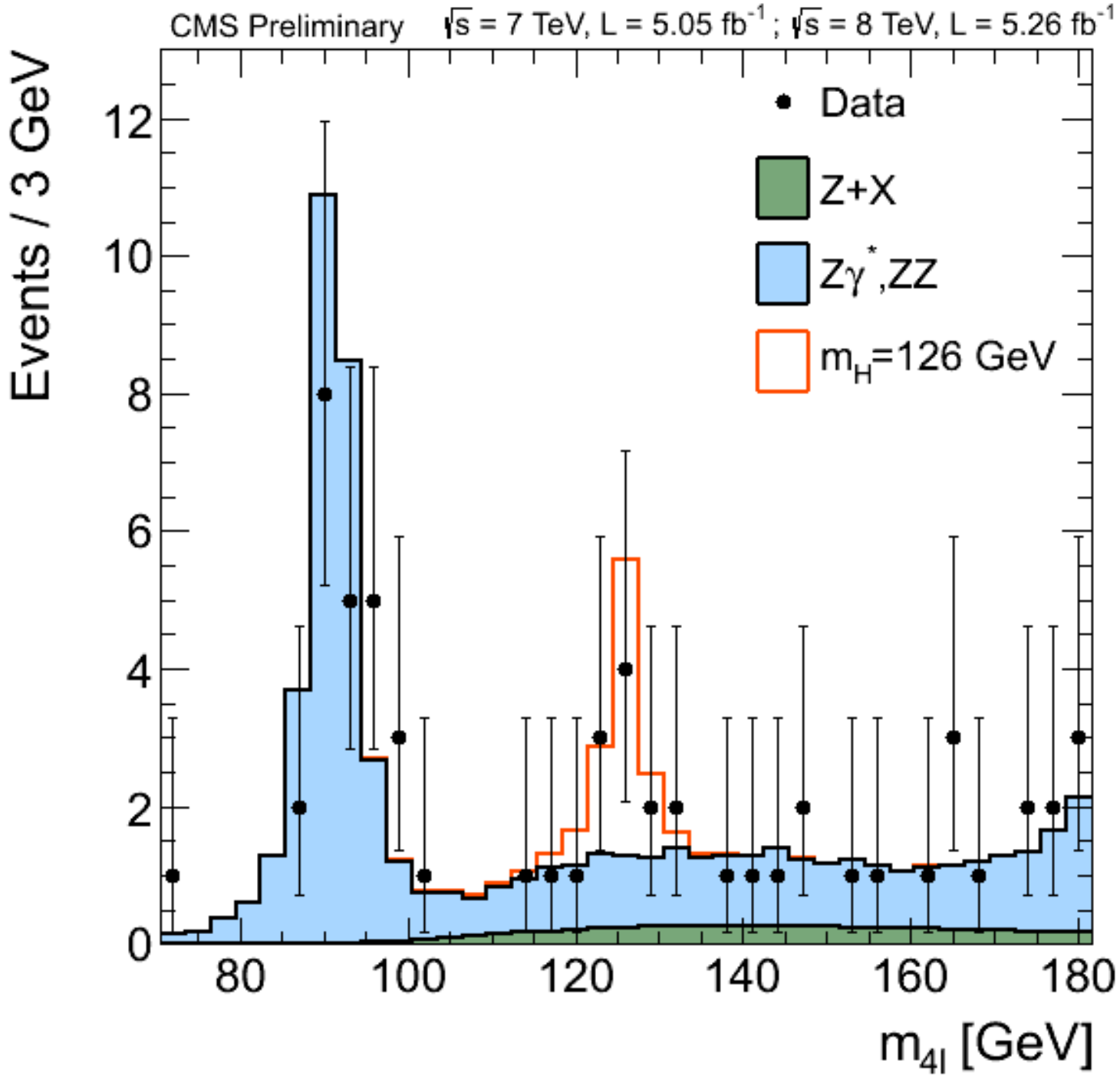
Where else do we see this pattern?

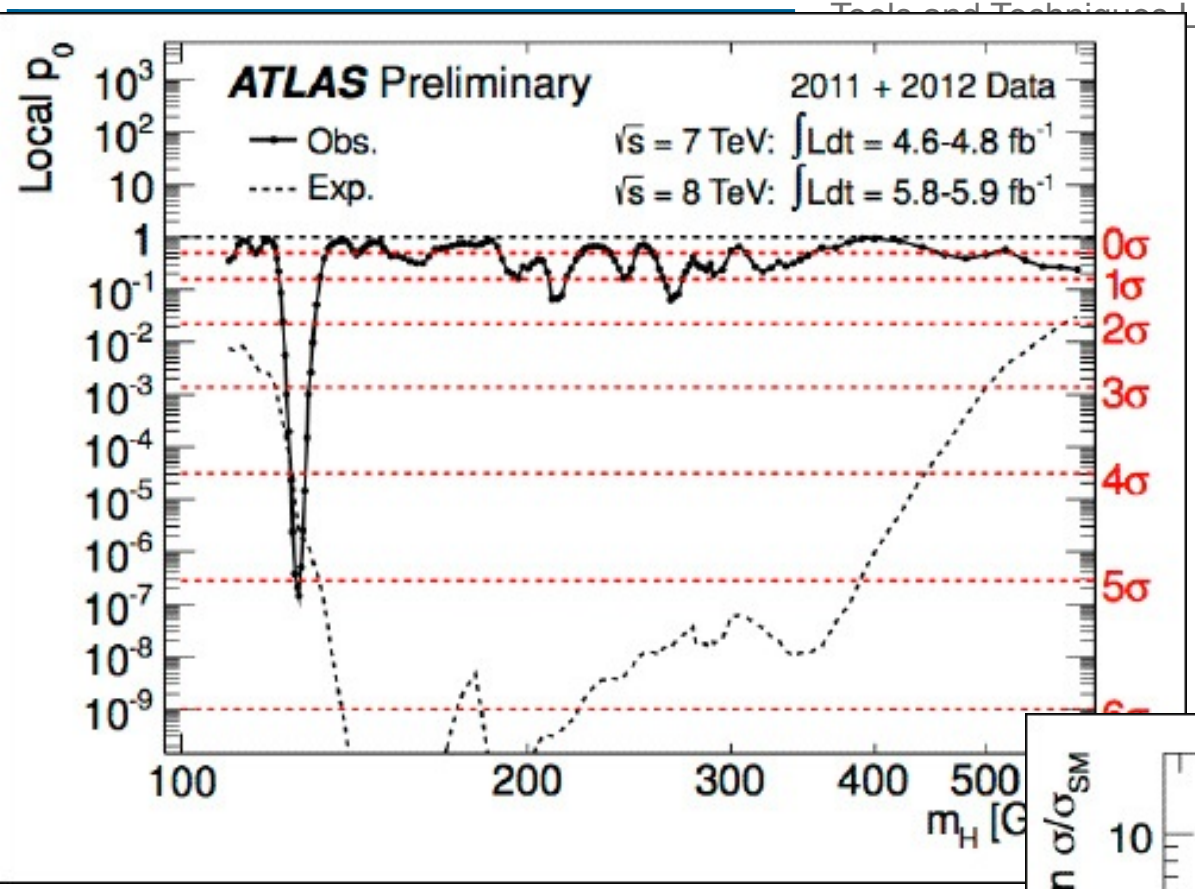
What do we do when we can't figure out the exact answer?

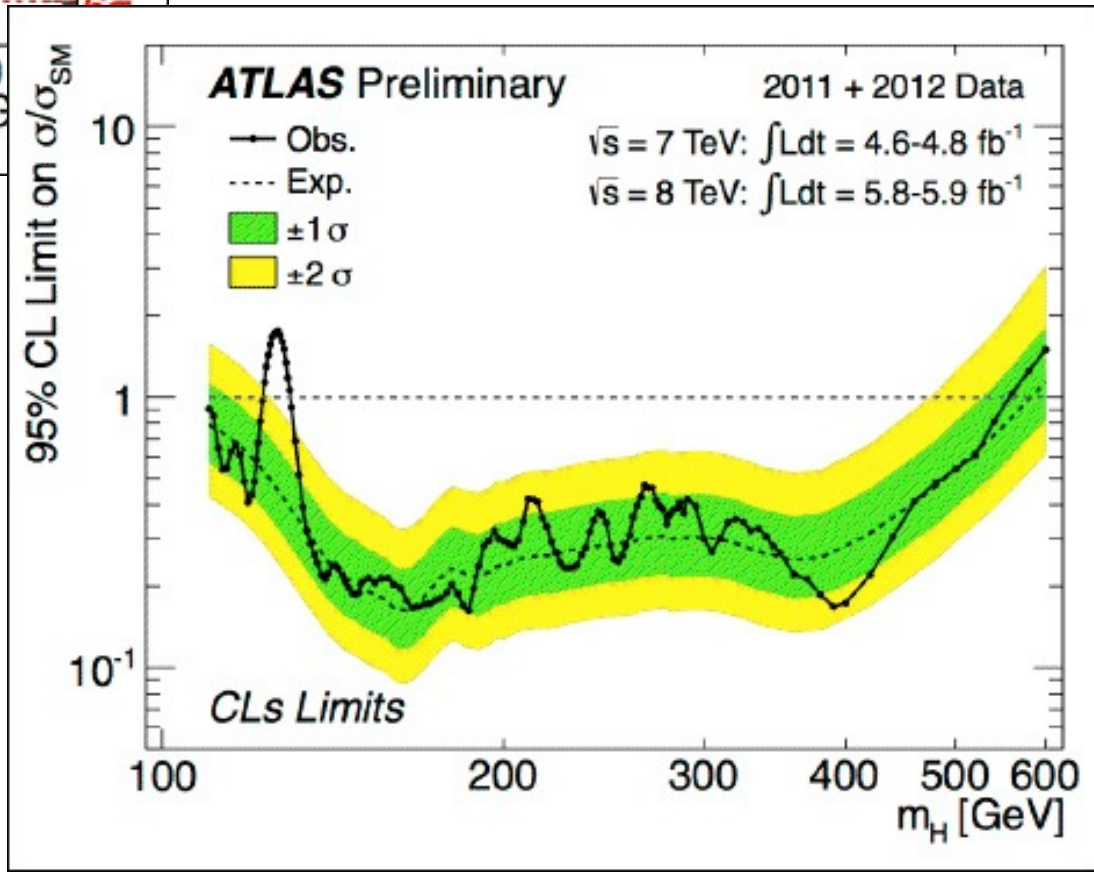



CMS Preliminary
√s = 7 TeV, L = 5.1 fb-1
√s = 8 TeV, L = 5.3 fb-1
S/B Weighted Data
S+B Fit
Bkg Fit Component
±1 σ
±2 σ
Weighted Events / (1.67 GeV)
mγγ (GeV)


Selected diphoton sample
Data 2011 and 2012
Sig + Bkg inclusive fit (mH = 126.5 GeV)
4th order polynomial
√s = 7 TeV, ∫ Ldt = 4.8 fb-1
√s = 8 TeV, ∫ Ldt = 5.9 fb-1
ATLAS Internal
Events / GeV
Data - Bkg
mγγ [GeV]


CMS Preliminary
√s = 7 TeV, L = 5.05 fb-1 ; √s = 8 TeV, L = 5.26 fb-1
Data
Z+X
Zγ*,ZZ
mH=126 GeV
Events / 3 GeV
0
2
4
6
8
10
12
80
100
120
140
160
180
m4l [GeV]


ATLAS Preliminary
2011 + 2012 Data
Obs.
Exp.
√s = 7 TeV: ∫Ldt = 4.6-4.8 fb⁻¹
√s = 8 TeV: ∫Ldt = 5.8-5.9 fb⁻¹
Local p0
0σ
1σ
2σ
3σ
4σ
5σ
100
200
300
400
500
mH [G


ATLAS Preliminary
2011 + 2012 Data
Obs.
Exp.
±1σ
±2σ
√s = 7 TeV: ∫Ldt = 4.6-4.8 fb⁻¹
√s = 8 TeV: ∫Ldt = 5.8-5.9 fb⁻¹
95% CL Limit on σ/σSM
CLs Limits
100
200
300
400
500
600
mH [GeV]

Goal: “An informed way of experimental working”

Find a way of doing good work

Use tools wisely

Think about what you’re doing