This slide intentionally left blank.

Performance Magic in Java

Previously Known As:

Profiling Hadoop for Fun and Profit

Shevek
shevek@anarres.org

Introduction

Thus spake the master programmer:

> When you have learned to snatch
> the error code from the trap frame,
> it will be time for you to leave.

– The Tao of Programming, Geoffrey James

This whole project felt like a lesson in doing just this!

Instrumentation in Java

- The old way:

```
public void foo(...) throws IOException {
  long start = System.currentTimeMillis();
  …
  long end = System.currentTimeMillis();
  LOG.info(“Method took “ + (end – start) + “ ms.”);
}
```

- The JVMPI way (1.5+):

```
public static void agentmain(String args, Instrumentation instrumentation) {
  ClassFileTransformer transformer = new ...();
  instrumentation.addTransformer(transformer);
  // Hold onto your hats, folks. Here we go...
}
```

Given the choice, we choose to do it the hard way.

Bytecode Instrumentation

- We and modify the bytecode as it loads.
  - Somewhat dangerous.
  - Actually, very dangerous.
- I've written a few compilers.
  - Some of them target the JVM.
  - It's just a rather nice stack machine.
- Use the asm library (sorry Apache).

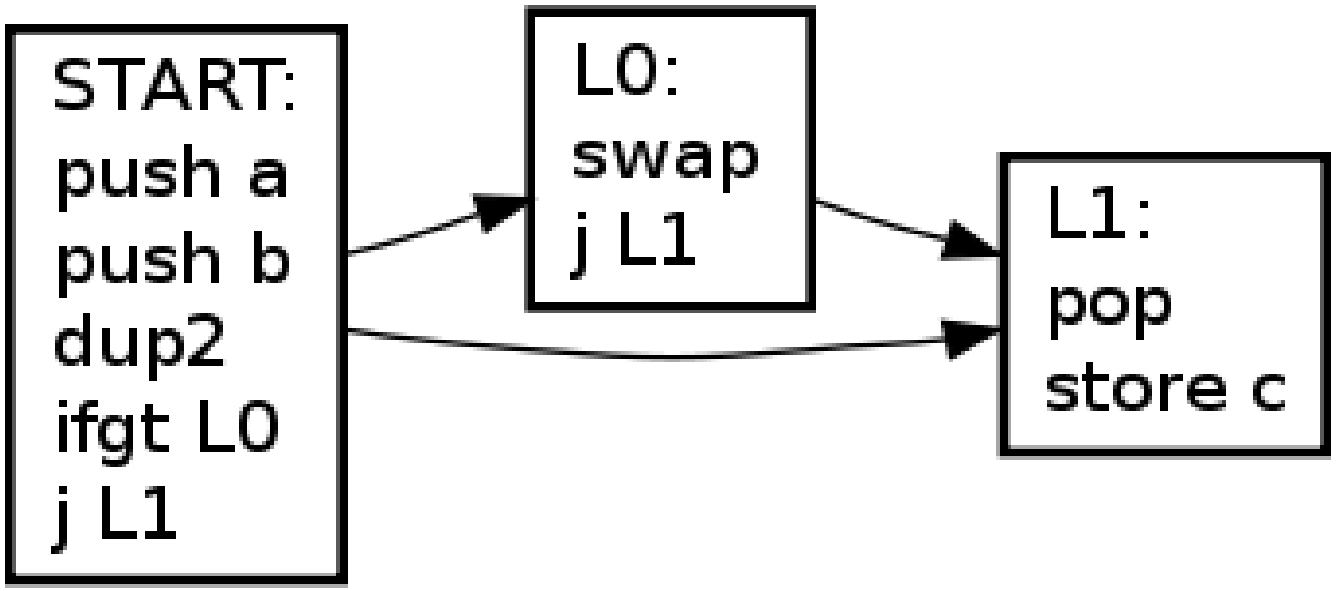
jcf-dump -c is the JVM hacker's lockpick.

What to Instrument?

- Individual bytecodes
  - Unnecessary.
- Basic blocks
  - Maybe.
- Methods
  - OK.
- Classes
  - Not detailed enough.

Basic Block Instrumentation

- Gives per-bytecode statistics without per-bytecode overhead.



If a block is entered, each instruction in the block **must** be executed.

Basic Block Reporting

```
150          * thing we do is strip the signature, just use
151          * the original entry.
152          */
153  20      if (ArchiveUtil.isSignatureFile(entry.getName()))
154          {
155   0              continue;
156          }
157  20      ZipEntry outputEntry = new ZipEntry(entry.getName());
158  20      outputEntry.setComment(entry.getComment());
159  20      outputEntry.setExtra(entry.getExtra());
160  20      outputEntry.setTime(entry.getTime());
161  20      output.putNextEntry(outputEntry);
162
163          // Read current entry
164  20      byte[] entryBytes = IOUtil
165                      .createByteArrayFromInputStream(archive);
166
167          // Instrument embedded archives if a classPattern has been specified
168  20      if ((classPattern.isSpecified()) && ArchiveUtil.isArchive(entryName))
169          {
170   1              Archive archiveObj = new Archive(file, entryBytes);
171   1              addInstrumentationToArchive(archiveObj);
172   1              if (archiveObj.isModified())
173              {
```

A sample cobertura report.

Method Instrumentation

```
public class MyClass {
    public Object doSomething() {
        // do something
    }
}
```

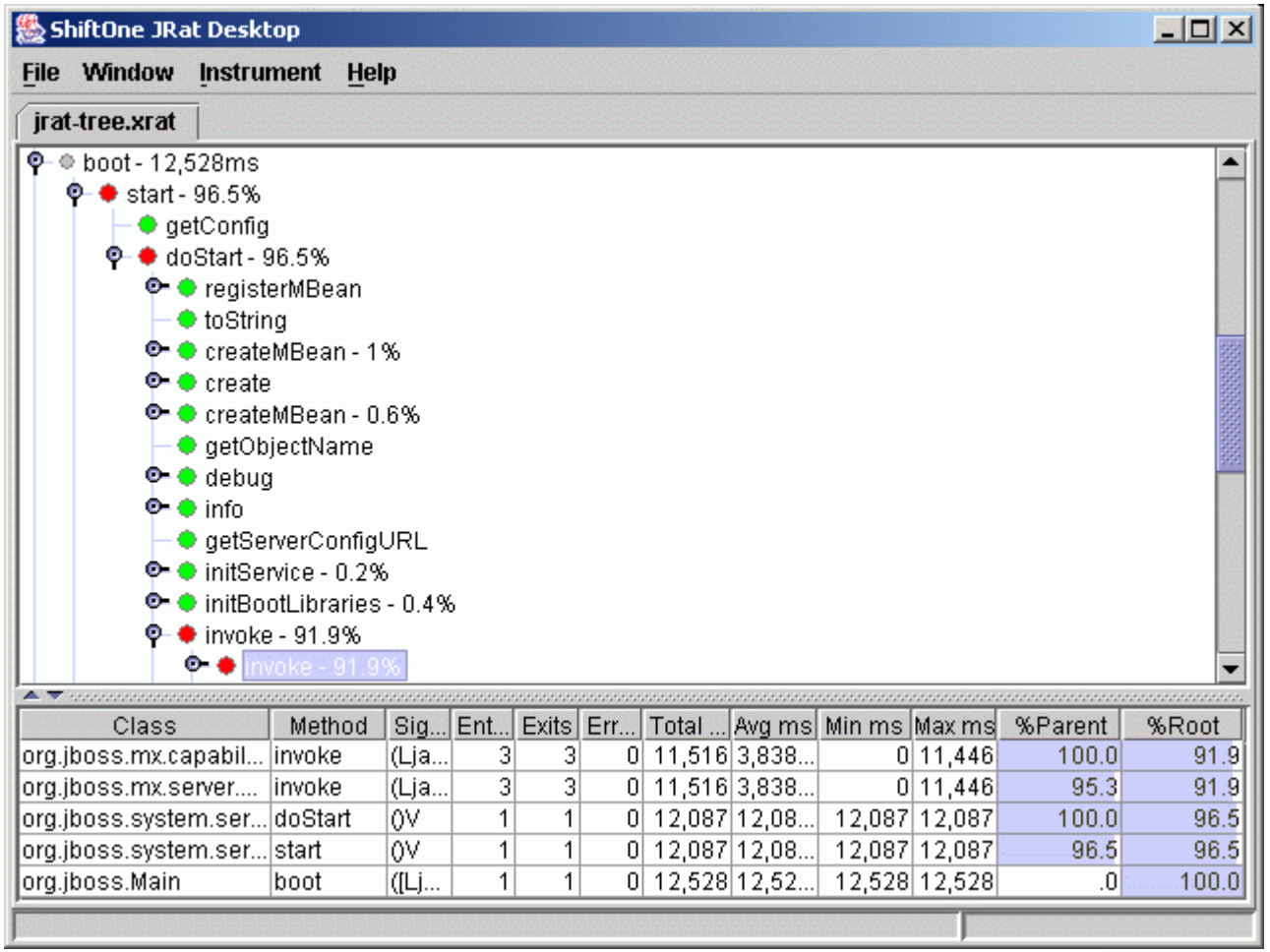
becomes

```
public class MyClass {
    private static final MethodHandler handler = HandlerFactory.getHandler(...);

    public Object doSomething() {
        handler.onMethodStart(this);
        long startTime = Clock.getTime();
        try {
           Object result = real_renamed_doSomething(); // call your method
           handler.onMethodFinish(this, Clock.getTime() - startTime, null);
        } catch(Throwable e) {
           handler.onMethodFinish(this, Clock.getTime() - startTime, e);
           throw e;
        }
    }

    public Object real_renamed_doSomething() {
        // do something
    }
}
```

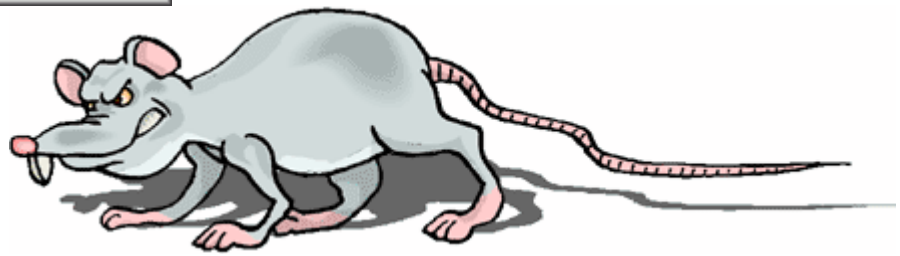
Introducing JRat



- Kosher UI.
- Clean code.

JRat Desktop showing the TreeHandler.

- LGPL license.



JRat Output

- One output file per handler.
  - jrat.log
    - The jrat boot/execution log file.
  - memory.csv
    - A log of memory usage.
  - tree.jrat
    - A log of call tree timings.
    - The most valuable of the log files.

Integration Guidelines

- Use the compiler where possible.
  - Else lean heavily on the bytecode verifier.
- Avoid reflection where possible.
- Make the code resilient against change.
- Make the code elegant and comprehensible.

Hadoop Already!

- We can add a JVM parameter to the TaskTracker Child.
  - `mapred.task.profile.params`
  - `mapreduce.task.profile.params`
- Where's our JAR file?
  - Write a sleep job, log into the task tracker, and discover...
  - `-javaagent:../../jars/profiler.jar`
- Discovering this was a pain in the arse.

Profiler Startup

- Open `memory.csv` for append.

That's enough of a problem for now...

Hadoop Deletes All Yr Files

- Our cleaner does this too.

```
TaskRunner.java:

  public static void setupWorkDir(JobConf conf, File workDir) throws IOException {
    …

    /** deletes only the contents of workDir leaving the directory empty. We
     * can't delete the workDir as it is the current working directory.
     */
    FileUtil.fullyDeleteContents(workDir);
```

- But we saved our files there at startup!
- Bother. Let's patch Hadoop.

I hate this code already.

Patching Without Patching

- We can't reinstall Hadoop on the cluster.
- The bad code runs in the TaskTracker's Child.
- We control the Child JVM via Instrumentation.
- We can detect and replace bytecode patterns.
- e.g. FileUtil.fullyDeleteContents().
- We can patch Hadoop.

I love this code.

Our First Patch to Hadoop

```
public static void setupWorkDir(JobConf conf, File workDir) throws IOException {
  …

  /** deletes only the contents of workDir leaving the directory empty. We
   * can't delete the workDir as it is the current working directory.
   */
  FileUtil.fullyDeleteContents(workDir);
```

becomes

```
public static void setupWorkDir(JobConf conf, File workDir) throws IOException {
  …

  /** deletes only the contents of workDir leaving the directory empty. We
   * can't delete the workDir as it is the current working directory.
   */
goto L0;
    FileUtil.fullyDeleteContents(workDir);
L0:
    AgentUtil.fullyDeleteContentsExceptProfile(workDir);
```

Actually, I just love the looks on people's faces.

Hooking Bytecode

- We use the asm library.

```
public interface HadoopHook {
  public boolean matchCaller(String caller);
  public boolean matchCaller(String caller, String method);
  public boolean matchCallee(String callee, String method, String descriptor);
  public void preMethod(GeneratorAdapter adapter);
  public void preCall(GeneratorAdapter adapter);
  public void postCall(GeneratorAdapter adapter);
}

public class HadoopCleanupHook implements HadoopHook {
  public boolean matchCaller(String caller) {
    return caller.startsWith("org/apache/hadoop/mapred")
      && caller.endsWith("TaskRunner");
  }
  …
  public void preCall(GeneratorAdapter adapter) {
    jump = new Label();
    adapter.goTo(jump);
  }
  public void postCall(GeneratorAdapter adapter) {
    adapter.mark(jump);
    adapter.invokeStatic(Agent.TYPE, Agent.cleanup);
  }
}
```

Profiler Shutdown

- Runtime.addShutdownHook()
  - Allows us to hook the JVM exit.
- ShutdownListener
  - Called at Child exit-time and saves state to disk.

Hadoop Deletes All Yr Files

- Gaaaaaaaaaaaaaaaaaaaaaaaaaaaaaah!!11one

- The TaskTracker deletes them, so we can't patch it.
- We will save the state before JVM shutdown.

- We have the hang of this now.

Our Second Patch

```
public void done(TaskUmbilicalProtocol umbilical,
                 TaskReporter reporter
                 ) throws IOException, InterruptedException {
  …
  sendLastUpdate(umbilical);
  //signal the tasktracker that we are done
  sendDone(umbilical); // Deletes yr filez.
}
```

becomes

```
public void done(TaskUmbilicalProtocol umbilical,
                 TaskReporter reporter
                 ) throws IOException, InterruptedException {
  …
  Agent.shutdown();     // New code
  sendLastUpdate(umbilical);
  //signal the tasktracker that we are done
  sendDone(umbilical); // Deletes yr filez.
}
```

Now we just lose the return of main(String[]).

Retrieving the Output

- Hadoop only lets us log to five output files.
  - stdout - used
  - stderr - used
  - syslog - used
  - debugout - um
  - profile.out – unused!

- Dear Hadoop, can we please have the entire log dir?
  - No.
- So, profile.out had better be a JAR file.

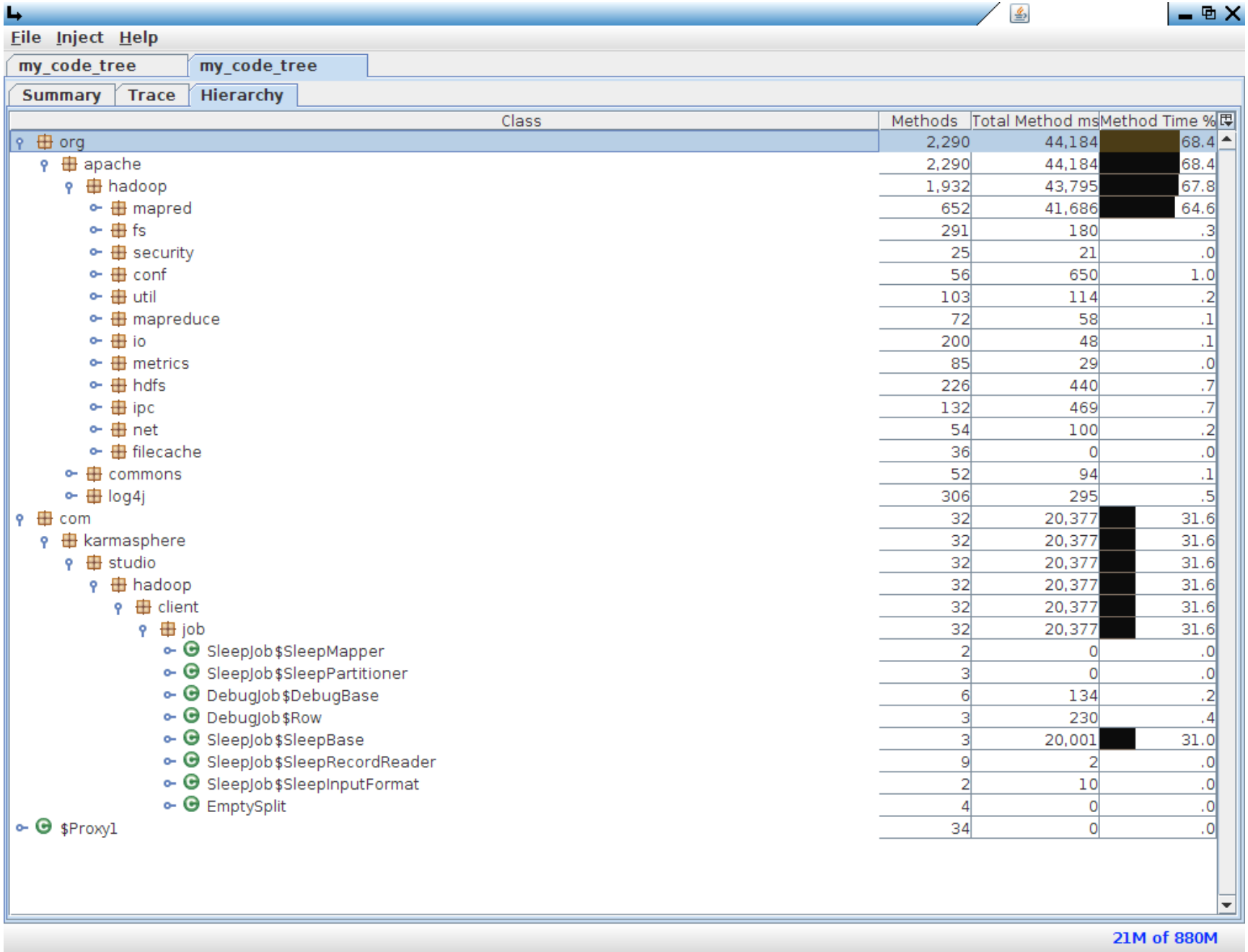
Let's get down and dirty.

Saving to a JAR file

- We don't know the profile.out filename.
  - We can get this by parsing agent parameters.
  - Hadoop replaces %s in `mapred.task.profile.params`
- We don't know the JRat log directory.
  - We can use a terrible heuristic.
  - It basically always gives the right answer.

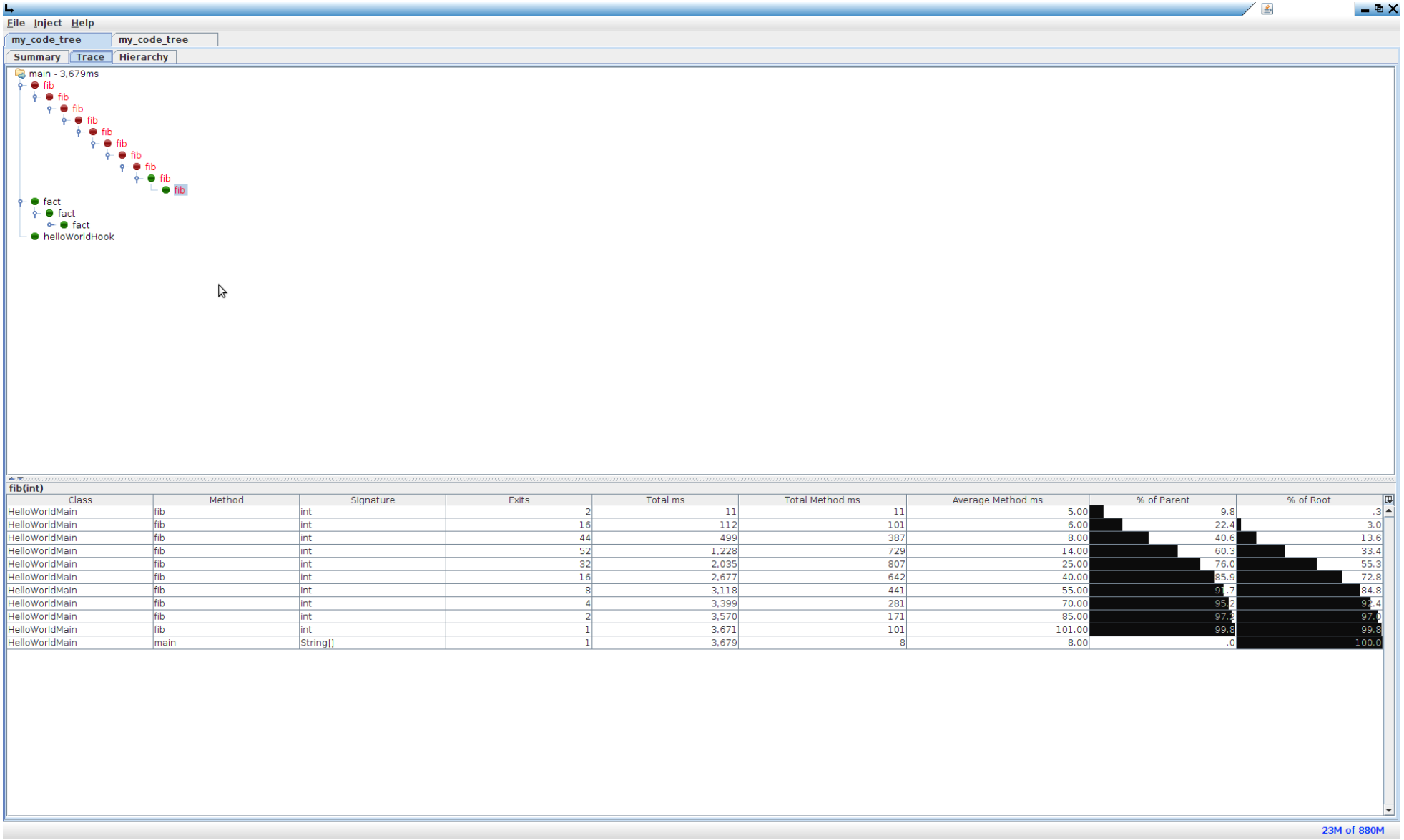
- At last, we can run a profiler under a job.

It Works!



Class	Methods	Total Method ms	Method Time %
org	2,290	44,184	68.4
apache	2,290	44,184	68.4
hadoop	1,932	43,795	67.8
mapred	652	41,686	64.6
fs	291	180	.3
security	25	21	.0
conf	56	650	1.0
util	103	114	.2
mapreduce	72	58	.1
io	200	48	.1
metrics	85	29	.0
hdfs	226	440	.7
ipc	132	469	.7
net	54	100	.2
filecache	36	0	.0
commons	52	94	.1
log4j	306	295	.5
com	32	20,377	31.6
karmasphere	32	20,377	31.6
studio	32	20,377	31.6
hadoop	32	20,377	31.6
client	32	20,377	31.6
job	32	20,377	31.6
SleepJob$SleepMapper	2	0	.0
SleepJob$SleepPartitioner	3	0	.0
DebugJob$DebugBase	6	134	.2
DebugJob$Row	3	230	.4
SleepJob$SleepBase	3	20,001	31.0
SleepJob$SleepRecordReader	9	2	.0
SleepJob$SleepInputFormat	2	10	.0
EmptySplit	4	0	.0
$Proxy1	34	0	.0

Call Traces


File Inject Help
my_code_tree
my_code_tree
Summary
Trace
Hierarchy
main - 3,679ms
fib
fib
fib
fib
fib
fib
fib
fib
fib
fib
fact
fact
fact
helloWorldHook
fib(int)
Class | Method | Signature | Exits | Total ms | Total Method ms | Average Method ms | % of Parent | % of Root
HelloWorldMain | fib | int | 2 | 11 | 11 | 5.00 | 9.8 | .3
HelloWorldMain | fib | int | 16 | 112 | 101 | 6.00 | 22.4 | 3.0
HelloWorldMain | fib | int | 44 | 499 | 387 | 8.00 | 40.6 | 13.6
HelloWorldMain | fib | int | 52 | 1,228 | 729 | 14.00 | 60.3 | 33.4
HelloWorldMain | fib | int | 32 | 2,035 | 807 | 25.00 | 76.0 | 55.3
HelloWorldMain | fib | int | 16 | 2,677 | 642 | 40.00 | 85.9 | 72.8
HelloWorldMain | fib | int | 8 | 3,118 | 441 | 55.00 | 93.7 | 84.8
HelloWorldMain | fib | int | 4 | 3,399 | 281 | 70.00 | 95.2 | 92.4
HelloWorldMain | fib | int | 2 | 3,570 | 171 | 85.00 | 97.2 | 97.0
HelloWorldMain | fib | int | 1 | 3,671 | 101 | 101.00 | 99.8 | 99.8
HelloWorldMain | main | String[] | 1 | 3,679 | 8 | 8.00 | .0 | 100.0
23M of 880M

Performance Sucks!

- We created about a 10x overhead.
  - This shouldn't be the hugest surprise in the world.

Things That Are Slow

- Reading the clock is slow.
  - A small surprise, but not a big one.

```
public class ClockThread extends Thread {
    private volatile long millis;

    public long getClock() {
        return millis;
    }

    @Override
    public void run() {
            for (;;) {
                millis = System.currentTimeMillis();
                Thread.sleep(1);
            }
    }
}
```

Now we rewrite JRat to call our Clock instead of System.currentTimeMillis()

Function Call is Slow

- We are adding five function calls per call.
- We reduce this by sampling.

```
public void run() {
        while (read(input)) {
                mapper.map(input);
        }
}
```

becomes

```
public void run() {
        int counter = 0;
        while (read(input)) {
                counter++
                ThreadState.setEnabled(counter & 0x1f == 0);
                mapper.map(input);
        }
}
```

… which does not synchronize threads.

Comparator is Slow

```
protected class SpillThread extends Thread {

    @Override
    public void run() {
      spillLock.lock()
```

becomes

```
protected class SpillThread extends Thread {

    @Override
    public void run() {
      ThreadState.getInstance().setEnabled(false);   // Disable for this thread
      spillLock.lock()
```

Keep hunting – lucky we have a profiler.

Compare and Exchange Is Slow

- AtomicInteger becomes cmpxchg

Adding Functionality

```
while (input.next(key, value)) {
  mapper.map(key, value, output, reporter);
}
```

becomes

```
while (input.next(key, value)) {
  try {
    Object _k = key;
    Object _v = value;
    mapper.map(key, value, output, reporter);
  } catch (Throwable t) {
    LOG.ohs**t(..., _k, _v, ..., t);
  }
}

// Actually:
int[] l_params = new int[params.length];
for (int i = l_params.length - 1; i >= 0; i--) {
  l_params[i] = adapter.newLocal(T_OBJECT);
  adapter.storeLocal(l_params[i]);
}

for (int i = 0; i < l_params.length; i++)
  adapter.loadLocal(l_params[i]);
…
```

We have the hang of this now!

Other Instrumentation

- Don't instrument the spill thread.
- Catch exceptions from Mapper.map()
- Add one to x.

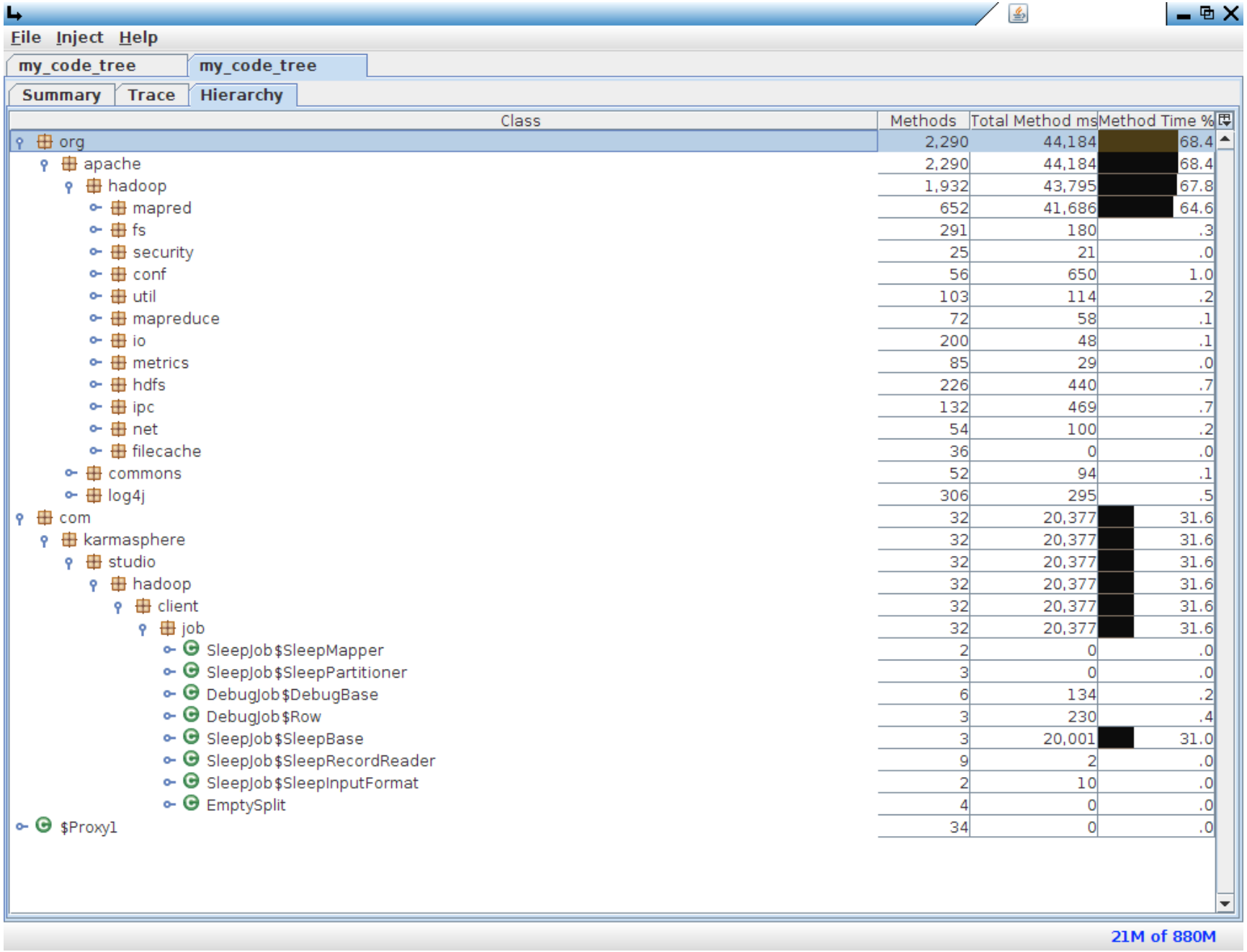
Summary of the Profiler

- It gives about 2x overhead.
- The information it gives is REALLY useful.
- It runs on any stock Hadoop installation.
- It can be controlled per-task.

- It's really worth it.

And now for something completely different!

That Screenshot Again



Class	Methods	Total Method ms	Method Time %
org	2,290	44,184	68.4
apache	2,290	44,184	68.4
hadoop	1,932	43,795	67.8
mapred	652	41,686	64.6
fs	291	180	.3
security	25	21	.0
conf	56	650	1.0
util	103	114	.2
mapreduce	72	58	.1
io	200	48	.1
metrics	85	29	.0
hdfs	226	440	.7
ipc	132	469	.7
net	54	100	.2
filecache	36	0	.0
commons	52	94	.1
log4j	306	295	.5
com	32	20,377	31.6
karmasphere	32	20,377	31.6
studio	32	20,377	31.6
hadoop	32	20,377	31.6
client	32	20,377	31.6
job	32	20,377	31.6
SleepJob$SleepMapper	2	0	.0
SleepJob$SleepPartitioner	3	0	.0
DebugJob$DebugBase	6	134	.2
DebugJob$Row	3	230	.4
SleepJob$SleepBase	3	20,001	31.0
SleepJob$SleepRecordReader	9	2	.0
SleepJob$SleepInputFormat	2	10	.0
EmptySplit	4	0	.0
$Proxy1	34	0	.0

Other Things I Have Done

- LZO compression in pure Java.
- Read only memory in Java.
- More versatile lexers.
- The C Preprocessor.
- Continuation passing form.

And now for something completely different!

LZO Compression

- JNI bites. JNA bites. It all bites.

```
There is no version of LZO in pure Java. The obvious solution is to
take the C source code, and feed it to the Java compiler, modifying
the Java compiler as necessary to make it compile.

This package is an implementation of that obvious solution, for which
I can only apologise to the world.

It turns out, however, that the compression performance on a single
2.4GHz laptop CPU is in excess of 500Mb/sec, and decompression runs
at 815Mb/sec, which seems to be more than adequate. Run
PerformanceTest on an appropriate file to reproduce these figures.

                                                    — lzo-java README
```

- https://github.com/shevek/lzo-java
  - Please note changed upstream URL.

And now for something completely different!

Compiling C with Javac

- Add a preprocessor.
- Make pointer-safe.
- Define a few types.

```
<cpp todir="${build.generated.sources.dir}/jcpp">
        <fileset dir="jcpp" includes="**/*.jcpp" />
        <globmapper from="*.jcpp" to="*.java"/>
        <systemincludepath>
        </systemincludepath>
        <localincludepath>
        <pathelement path="jcpp/src" />
        <pathelement path="jcpp/include" />
        </localincludepath>
</cpp>
```

```
<exec executable="perl" osfamily="unix">
        <arg value="-n" />
        <arg value="-i" />
        <arg value="-e" />
        <arg value="/^\s+$/ or print;" />
        <arg line="${rmwhite.files}" />
</exec>
```

The C Preprocessor

- Enabled the LZO-Java project, but predated it.
- Well tested and widely deployed.
- http://www.anarres.org/projects/jcpp/
- Successfully preprocesses glibc.

```
…
  case '+':
      d = read();
      if (d == '+')
          tok = new Token(INC);
      else if (d == '=')
          tok = new Token(PLUS_EQ);
      else
          unread(d);
      break;
…
```

IBM used it in WebSphere. Does anybody know where?

More Accomodating Lexers

- “Be strict in what you send, but generous in what you receive.”
  - Jon Postel (kind of)
- Be generous in what you accept.
- Be informative in what you provide.
- Be a nice citizen.

The Process of Lexing

- A left-to-right process.
  - int foo(int x) { }
    - T_INT, T_IDENTIFIER, T_LPAR, T_INT, T_IDENTIFIER, T_LBRACE, T_RBRACE
  - foo + * &
    - T_IDENTIFIER, T_PLUS, T_STAR, T_AMP
  - foo $ bar @ baz
    - Bad – now what?
    - We get as far as the $ and throw an exception. We learn nothing about “bar”, which is valid.
    - A single typo kills the lexer.

The Development Environment

- If you're editing it, it's probably broken.
- It's no good having a development environment that only works with valid code.
- For example, Eclipse can't reformat invalid code. NetBeans can.
- How many useful errors can we discover before we barf?

Invalid tokens

- foo @ bar “asdf
  - T_IDENTIFIER (“foo”)
  - T_BAD_TOKEN (“@”)
  - T_IDENTIFIER (“bar”)
  - T_UNTERMINATED_STRING_LITERAL (“\”asdf”)

- Use a recovery heuristic to resynchronize.
  - jcpp uses a newline.

Lexer Implementation

- Add extra lexer rules to return invalid tokens.

```
bad_octal_constant = octal_constant ['8'..'9'] digit*;
bad_constant       = bad_octal_constant;
bad_string_literal = 'L'? '"' s_char_seq?;
bad_char_constant  = 'L'? ''' c_char_seq?;
bad_identifier     = digit identifier_nondigit+;
bad_token          = all;
```

- Lexers are usually greedy so this works.
- The parser will still barf, but that's OK.

Examples

- “Be strict in what you send, but generous in what you receive.”

- JCPP (Java C Preprocessor)
  - A hand written lexer for C with resynchronization.
- SableCC (and various examples)
  - As per the examples above.

Read Only Memory in Java

- Only the owner of a data structure should mutate it.
- Return-by-reference is dangerous.
  - But copying data structures is expensive.
- Can the compiler help?

Java 1.5

- Parameterized types.
  - List<X>

```
public interface Foo<X> {
        public void add(X value);
        public X get(int index);
}
```

- Now the compiler can check our code.

```
Foo<String> x = ...;

x.add("bar");  // OK

x.add(5);       // Not OK

String value = x.get(4);       // Note, no cast.
```

Aside on Type Parameters

- We can be generous with type parameters.

```
public class Foo<X> { // Here, we all know.

    @Override
    public <T> T add(Foo<T> remote, T value) { // Also, here!
        ...
    }
}
```

- Now we can say “These two things are of the same type.” without knowing the type!

```
public interface InstanceMap {
    public void set(@Nonnull Class<T> type, T value);

    @CheckForNull
    public T get(@Nonnull Class<T> type);
}
```

Java 1.5 Bytecode

- What happens underneath?

```
public interface Foo<X> {
        public void add(X value);    // It's an Object.
}

public class MyFoo implements Foo<String> {
        @Override
        public void add(String value) {  // This can't override (Object)
                ...
        }
}

public class MyFoo implements Foo<String> {

        public void add(String value) {
                ...
        }

        @Override
        public synthetic void add(Object value) {// So this does.
                add((String)value);
        }
}
```

Bounded Parameters

- We can give required properties of the parameter X.

```
public interface Foo<X extends Bar> {
        public void add(X value) {
                // Now we can use the properties of Bar, but not X.
        }
}

public class MyBar extends Bar { }
public class YourBar extends Bar { }

Foo<MyBar>        // Valid
Foo<YourBar>      // Valid
Foo<String>       // Invalid
```

More Power to Type Bounds

- Help us write correct code.

```
public interface MyContainer<X> {
      public List<X> void getValues();
}

MyContainer<String> x = …;
List<String> l = x.getValues();
x.add(“foo”);
```

- Did we just modify an internal data structure?
- Can the compiler help us find out?

```
public interface MyContainer<X> {
      public List<? extends X> void getValues();
}

MyContainer<String> x = …;
List<? extends String> l = x.getValues();
x.add(“foo”);  // Illegal – can't create a value of type unknown.
```

Even More Power to Type Bounds

- We did read-only. Can we do write-only?

```
public interface MyContainer<X> {
      public List<? super X> void getTarget();
}

MyContainer<String> x = …;
List<? super String> l = x.getTarget();
x.add(“foo”);  // We're allowed to add Strings, or anything below.
x.get(...);       // Illegal, since we don't know the return type.
```

What Does a Bound Tell Us?

- It doesn't tell us the type, just the properties.
- We can have multiple bounds!

```
public interface MyContainer {
        public <T extends JComponent & MyPanel> void add(T panel) {
                // Now we can use the properties of JComponent
                // and MyPanel.
        }
}
```

- Now, we specified multiple behaviours in a language with only single inheritance!
- I forget what bytecode it compiles here.

Types Are Powerful

- Types are the primary tool for the compiler to prove correctness of code.
- If you used a cast, you did something wrong.
- Say what you mean, and the rest will follow.

More Epistemology?

- 12 dirty children.

Summary and Questions

- Ask questions.
  - For example, “How do you profile Hadoop?”
- Heckle.
  - For example, “You got it wrong!”
- Buy me drinks.
  - For example, any good single malt.
- Offer a challenge.
  - For example, “Fix the Eurozone deficit!”
- Use your imagination.

Thank you