Distributed Objects

—

The Next Ten Years

Andrew P. Black

OGI School of Science & Engineering
Oregon Health & Science University

black@cse.ogi.edu

Eden Project:1980

Early days for distributed systems

- John White first wrote about ***RPC*** (1976)
- ***RPC*** design alternatives explored by Nelson in his Ph.D. Research (1981)
- RPC implemented efficiently at PARC (1982)
- Bsd Unix does not yet have any IPC or networking system calls.
- “Department network” was a piece of cable in computer room connecting 2 Vaxen

Key Ideas of Object Orientation

Alan Kay (early 1970s)

- Objects localize data structures and code
- Objects are *a recursion on the idea of the computer itself*

  "The basic principle of recursive design is to make the parts have the same power as the whole".

  Rather than dividing the computer into "lesser stuffs", like data structures and procedures, we should divide it into lots of little computers that communicate together.

Goals of RPC

3 Implementing remote procedure calls
Andrew D. Birrell , Bruce Jay Nelson
ACM Transactions on Computer Systems (TOCS) February 1984
Volume 2 Issue 1

The primary purpose of our RPC project was to make distributed computation easy. Previously, it was observed within our research community that the construction of communicating programs was a difficult task, undertaken only by members of a select group of communication experts. Even researchers with substantial systems experience found it difficult to acquire the specialized expertise required to build distributed systems with existing tools. This seemed undesirable. We have available to us a very large, very powerful communication network, numerous powerful computers, and an environment that makes building programs relatively easy. The existing communication mechanisms appeared to be a major factor constraining further development of distributed computing.

1999: Lampson's Evaluation:

History: What Worked?

YES	NO (Not Yet?)
Virtual memory*	Capabilities*
Address spaces*	Fancy type systems*
Packet nets*	Functional programming
Objects / subtypes	Formal methods*
RDB and SQL	Software engineering
Transactions*	RPC (except for Web)*
Bitmaps and GUIs*	Distributed computing*
Web	Persistent objects
Algorithms	Security*

Birrell & Nelson Continue …

3 **Implementing remote procedure calls**
Andrew D. Birrell , Bruce Jay Nelson
ACM Transactions on Computer Systems (TOCS) February 1984
Volume 2 Issue 1

Our hope is that by providing communication with almost as much ease as local procedure calls, people will be encouraged to build and experiment with distributed applications. RPC will, we hope, remove unnecessary difficulties, leaving only the fundamental difficulties of building distributed systems: timing, independent failure of components, and the coexistence of independent execution environments.

The Object Model

- ubiquitous object reference mechanism
- objects export an interface
- objects are instances of a class
- send messages to objects, with objects as arguments
- objects respond by autonomously executing a *method*
- state of an object is (somewhat) encapsulated
- objects sharing is the normal form of data access
- objects are ***not*** explicitly deallocated

Objects and Distributed Systems

- Good match
  - expression 2 + 3 finally makes sense!
  - the target of a message formalizes the idea of a "binding"
  - objects export an interface
  - send messages to objects …
  - objects respond by autonomously executing a *method*
  - Objects are a natural unit of mobility

- Problematic
  - – ubiquitous object reference mechanism
    - - identity of Objects
  - – objects are instances of a class
  - – state of an object is ~~(somewhat)~~ encapsulated
  - – objects sharing is the normal form of data access
  - – … with objects as arguments
  - – objects are ***not*** explicitly deallocated
  - – Objects are not a natural unit of mobility
    - - because there is nothing in an object but object references!

What the Object Model Missed

- Sharing is not enough
  - eventually, you need the bits
- Immutability — our secret weapon!
  - Monotonicity?
- Replication and Caching
  - Object model gives us no help
- Both sides of Information Hiding
  - ***what***
  - from ***whom***

“Fundamental Difficulties” of Distribution

- Heterogeneity
- Openness
- Security
- Scalability
- Failures
- Time and Ordering of Events
- Concurrency

Largely Solved Problems (I claim)

- Heterogeneity
  - Standards/Protocols (wire support)
  - Middleware (API support)
  - JITs and VMs (Mobile code support)
- Openness
  - Structural typing
  - Published Interfaces
  - Open Source movement

Largely Solved Problems (cont.)

- Security
  - Authentication and Access Control are technically solved
  - Social and regulatory issues dominate
  - Security for mobile code/objects is still problematic
- Scalability
  - “The web won’t scale” — but it has x 10^6…

    …in numbers of computers

The Hard Problems

- Scaling in other dimensions
  - number of instances of an object
  - capacity of individual computers
- Evolution over time
  - versioning object
  - safe and unsafe type changes
  - new interfaces for old objects
  - new objects for old interfaces

Hard Problems for Languages

- Failure
  - always partial
  - mask by replication in space or time
  - propagate to software that can't know what has gone wrong
  - building "firewalls" that contain the consequences of failure

Hard Problems for Languages (cont)

- Timing
  - speed gap between local and remote is growing
  - no global clock or global event ordering
  - how can a language with no concept of time help us to reason about timing?
- Concurrency
  - modular construction of concurrent programs

So What Am I doing about it?

- Timing for distributed object systems
  - Real-Rate media streaming — Infopipes
  - Embedded control applications — Timber
- Transactions and Failures
  - Does “ACID” mean anything?
    - current work with Martin Oderski *et al.* at EPFL
- What about Programming Languages?
  - Programming Languages are dead!

Infopipes

Joint work with Jonathan Walpole & Calton Pu

Infopipes — An Abstraction for Multimedia Streaming

Restricted Domain: Information flow

– applications that transfer and process streams of information

- Examples:
  - distributed multimedia
    - streaming video and/or audio in real-time
  - environmental observation
    - Columbia River data: Forecast/Nowcast

Application Requirements

- Applications need to direct *streams* of information
  - to the right place
  - at the right time
  - containing the right information
  - with the right Quality
    - Quality of Service is a compromise between application specific *desires* and available *resources*

The Solution

- CORBA ORB, DCE RPC, Java RMI…

Solution

- ~~CORBA ORB, DCE RPC, Java RMI…~~

No!

Solution

- ~~CORBA ORB, DCE RPC, Java RMI…~~

No!

- These abstractions hide communication
- We want to reify communication

to reify = "to make the abstract real"

  - create concrete objects that represent communications abstraction
  - messages to these objects let us examine and change the properties of the communication link

Infopipes Reify Information Flows

- Infopipes reify communication

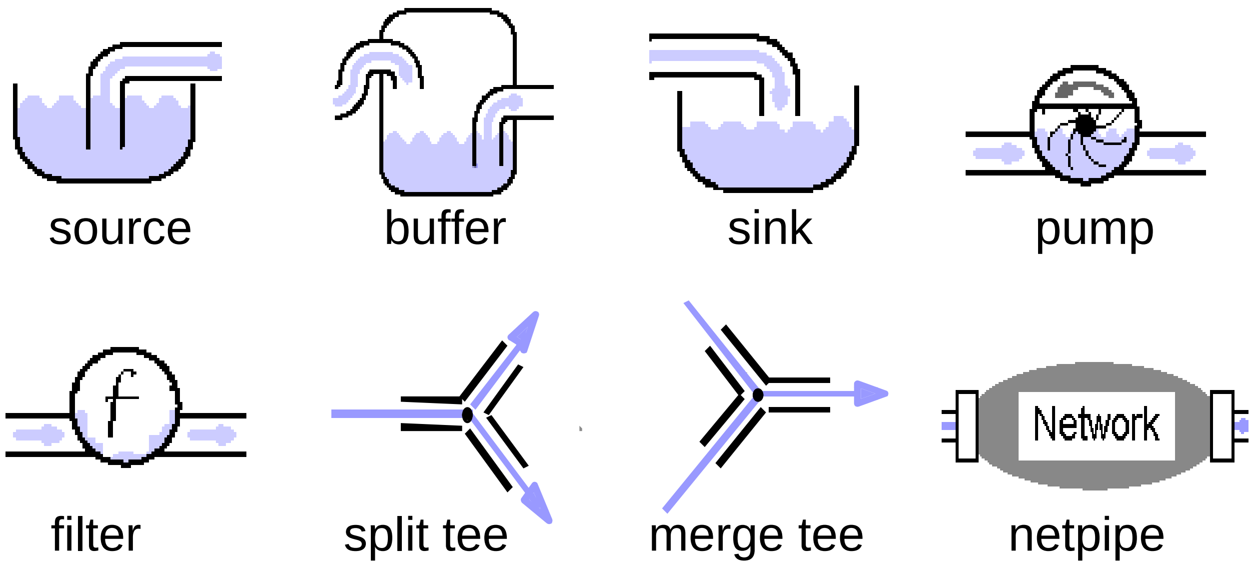
  …but at the *application* level

  not at the implementation level
- Example
  - bandwidth of Infopipe carrying compressed video
  - measured in frames per second, *not* bits per second
- Why?
  - If the application is going to do anything with flow information, it must be in application-level terms

What *are* Infopipes?

- System and distributed system abstraction
- Have well-defined characteristics, specifically, *rate*, *latency* and *jitter.*
- Compositional: the characteristics of a composite Infopipe can be calculated from those of its components
  - Seamless interconnection

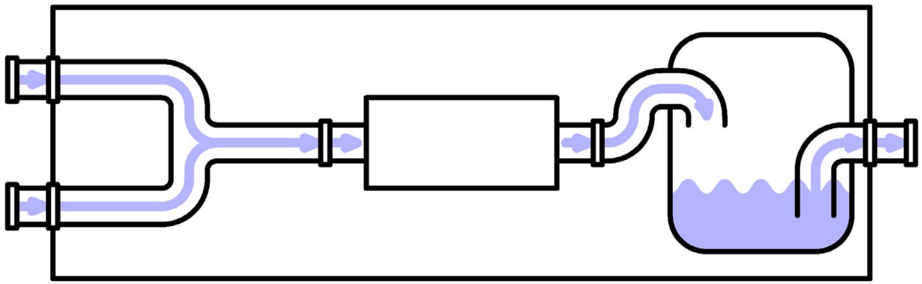
Think “Plumbing”



- An Infopipe has zero or more “Inports” and zero or more “Outports”

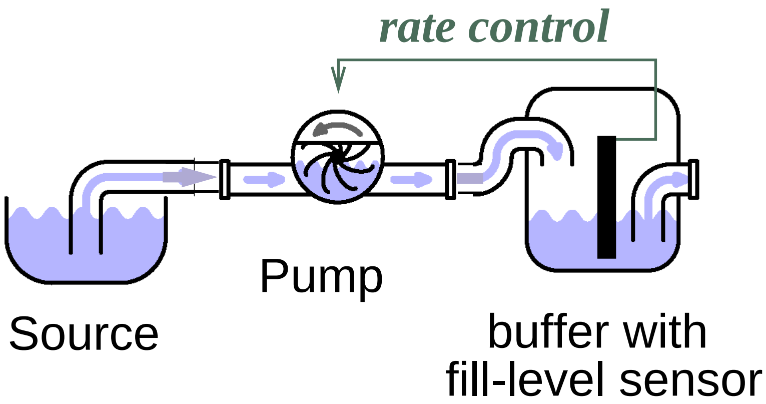
Composite Components

- Complex components can be built by putting a “black box” abstraction boundary around a Pipeline.



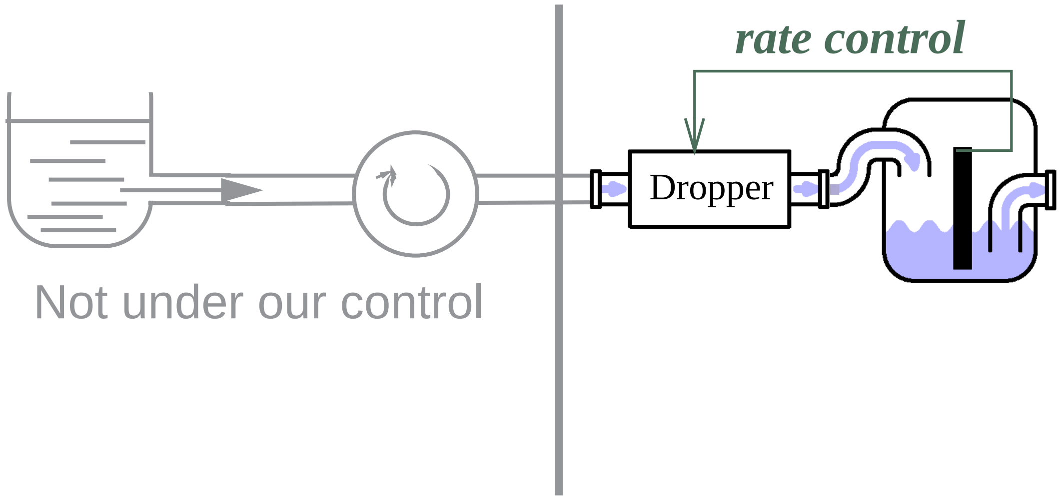
  – enables modularity and reuse

Feedback



- Rate of the pump is adjusted to keep buffer fill level within bounds.

Feedback



- Feedback control drops packets selectively
  - avoid random dropping!
  - *e.g.*, video trans-coder labels packets with high-resolution imagery as “low priority”.

Status

- “Polarity Checking” for ensuring well-formed pipelines
- “Activity” abstraction at a higher level than threads
- Library of reusable Infopipe components
- Look for a Technical Report “real soon now”

Timber

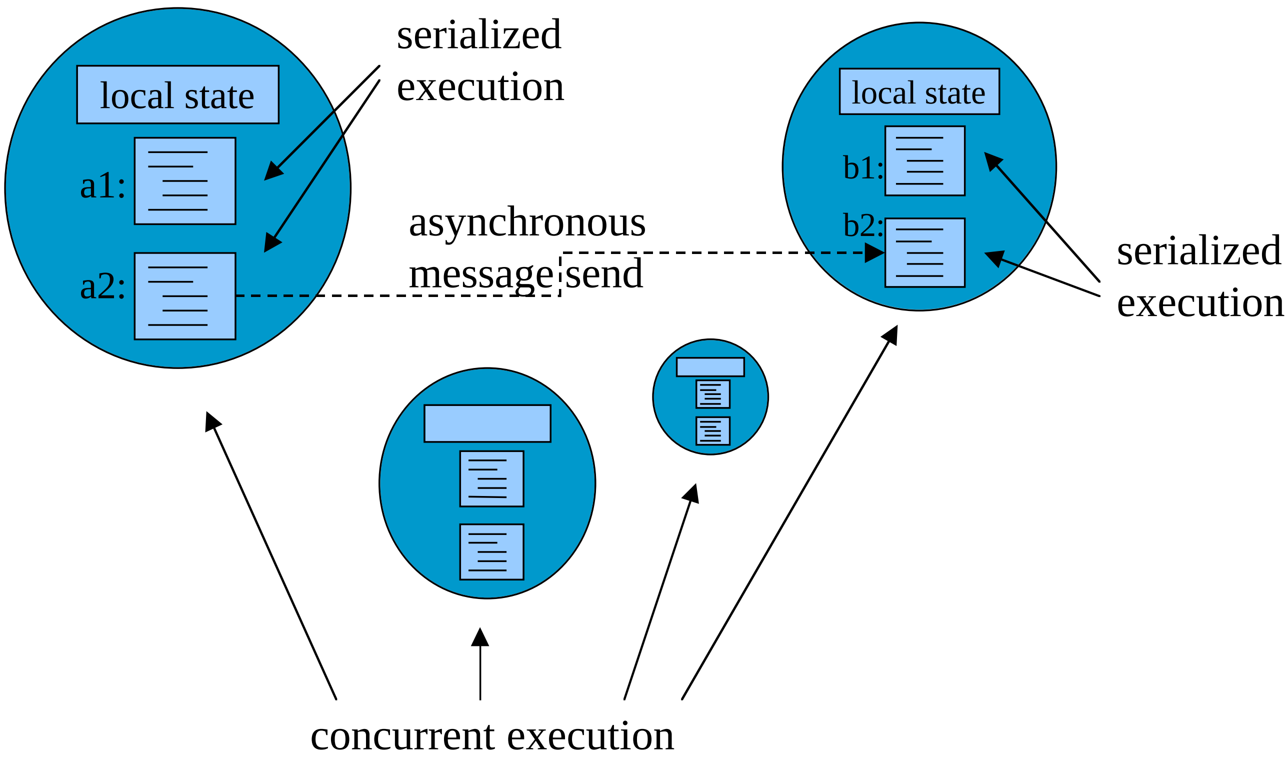
Joint work with Magnus Carlsson, Mark Jones, Dick Kieburtz, Johan Nordlander

Timber objectives:

- To
  - design a language with explicit time behavior
  - explore reactivity as the basic programming model
  - build upon the full power of a functional language
  - include OO concepts
  - support static timing analysis as well as dynamic adaptivity
- Starting point: O'Haskell
  - Johan Nordlander's thesis work

Concurrency & Objects

Object: State-carrying identity = autonomous thread of control



Reactivity

- Event = method invocation = message send
  - Output event: sending a message
  - Input event: being invoked = receiving a message
  - No active input
- Method = non-blocking code sequence
- Objects alternate between transient activity and indefinite periods of rest
  - Update local state / create new objects / send messages

Controlling Timing

- Each event is associated with a *baseline* ...and a... *deadline*

- Default values same as those of sender
- Can also be set explicitly:

```
after (10*seconds) m
before (25*milliseconds) m
```

Main theme: code can be *time-dependent*, yet *platform-independent*. Static analysis determines feasibility.

Accessing the Timeline

- Built-in constants `baseline` and `deadline`
  - defined only within methods
  - provide access to the baseline and the deadline for the current method execution
- For methods initiated by the environment, timeline must be defined appropriately
  - *e.g.*, for an interrupt, baseline might be time at which the hardware event occurred
  - deadline might be time within which registers must be read

Example: Ping Program

```
-> hosts = ["dogbert", "ratbert", "ratberg”, "theboss"]

-> ping hosts (Port 515)

dogbert: lookup & connect after 20.018 ms
ratbert: lookup & connect after 41.432 ms
ratberg: NetError "Host name lookup failure" after 70.282 ms
theboss: no response within 2 s
```

```
ping hosts port env =
    object
       outstanding := hosts
    in let
       client host start peer =
          record
             connect     = action
                env.putStrLn(host  ++ ": lookup & connect after "
                                   ++ show (baseline-start))
                outstanding := remove host outstanding
                peer.close
             deliver   _ = action done
             neterror e = action
                env.putStrLn(host  ++ ": " ++ show e ++ " after "
                                   ++ show (baseline-start))
                outstanding := remove host outstanding
             close       = action done
       cleanup = action
          forall h <- outstanding do
             env.putStrLn(h ++ ": no response within ″ ++ show timeout)
          env.quit
       timeout = 2*seconds
    in record
       main = action
          forall h <- hosts do
             env.inet.tcp.open h port (client h baseline)
          after timeout cleanup
```

Java Version

http://java.sun.com/j2se/1.4/docs/guide/nio/example/Ping.java

```
Boulder:Users:black:Andrew  Black:talks
Wednesday, 5 December 2001

import java.io.*;
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
import java.nio.charset.*;
import java.util.*;
import java.util.regex.*;


public class Ping {

    static int DAYTIME_PORT = 13;

    static int port = DAYTIME_PORT;


    static class Target {

    InetSocketAddress address;
    SocketChannel channel;
    Exception failure;
    long connectStart;
    long connectFinish = 0;
    boolean shown = false;

    Target(String host) {
        try {
        address = new InetSocketAddress(Inet
                port);
        } catch (IOException x) {
        failure = x;
        }
    }

    void show() {
        String result;
        if (connectFinish != 0)
        result = Long.toString(connectFinish
        else if (failure != null)
        result = failure.toString();
        else
        result = "Timed out";
        System.out.println(address + " : " +
        shown = true;
    }

    }


    static class Printer
    extends Thread
    {
    LinkedList pending = new LinkedList();

    Printer() {
        setName("Printer");
        setDaemon(true);
    }

    void add(Target t) {
        synchronized (pending) {
```

```
Boulder:Users:black:Andrew  Black:talks:Timber:Ping.java           Page 2 of 4
Wednesday, 5 December 2001                                  Printed:  17:16:00

        pending.add(t);
        pending.notify();
        }
    }

    public void run() {
        for (;;) {
        Target t = null;
        synchronized (pending) {
            try {
            pending.wait();
            } catch (InterruptedException x) {
            return;
            }
            while (pending.size() > 0) {
            t = (Target)pending.removeFirst();
            }
        }
        t.show();
        }
    }
    }


    static class Connector
    extends Thread
    {
    Selector sel;
    Printer printer;

    LinkedList pending = new LinkedList();

    Connector(Printer pr) throws IOException {
        printer = pr;
        sel = Selector.open();
        setName("Connector");
    }

    void add(Target t) {
        SocketChannel sc = null;
        try {

        sc = SocketChannel.open();
        sc.configureBlocking(false);
        sc.connect(t.address);

        t.channel = sc;
        t.connectStart = System.currentTimeMillis()

        synchronized (pending) {
            pending.add(t);
        }

        sel.wakeup();

        } catch (IOException x) {
        if (sc != null) {
            try {
            sc.close();
            } catch (IOException xx) { }
        }
        t.failure = x;
```

```
Boulder:Users:black:Andrew  Black:talks:Timber:Ping.java
Wednesday, 5 December 2001

        printer.add(t);
        }
    }

    void processPendingTargets() throws IOException {
        synchronized (pending) {
        while (pending.size() > 0) {
            Target t = (Target)pending.removeFirst();

            SelectionKey sk;
            try {
            sk = t.channel.register(sel,
                        SelectionKey.OP_CONNECT);
            } catch (IOException x) {
            t.channel.close();
            t.failure = x;
            printer.add(t);
            continue;
            }

            sk.attach(t);

        }
        }
    }

    void processSelectedKeys() throws IOException {
        for (Iterator i = sel.selectedKeys().iterator(); i.hasNext();

        SelectionKey sk = (SelectionKey)i.next();
        i.remove();

        Target t = (Target)sk.attachment();
        SocketChannel sc = (SocketChannel)sk.channel();

        try {
            if (sc.finishConnect()) {
            sk.cancel();
            t.connectFinish = System.currentTimeMillis();
            sc.close();
            printer.add(t);
            }
        } catch (IOException x) {
            sc.close();
            t.failure = x;
            printer.add(t);
        }
        }
    }

    volatile boolean shutdown = false;

    void shutdown() {
        shutdown = true;
        sel.wakeup();
    }

    public void run() {
        for (;;) {
        try {
            int n = sel.select();
            if (n > 0)
```

```
Boulder:Users:black:Andrew  Black:talks:Timber:Ping.java
Wednesday, 5 December 2001                                             Pri

            processSelectedKeys();
            processPendingTargets();
            if (shutdown) {
            sel.close();
            return;
            }
        } catch (IOException x) {
            x.printStackTrace();
        }
        }
    }

    }


    public static void main(String[] args)
    throws InterruptedException, IOException
    {
    if (args.length < 1) {
        System.err.println("Usage: java Ping [port] host...");
        return;
    }
    int firstArg = 0;

    if (Pattern.matches("[0-9]+", args[0])) {
        port = Integer.parseInt(args[0]);
        firstArg = 1;
    }

    Printer printer = new Printer();
    printer.start();
    Connector connector = new Connector(printer);
    connector.start();

    LinkedList targets = new LinkedList();
    for (int i = firstArg; i < args.length; i++) {
        Target t = new Target(args[i]);
        targets.add(t);
        connector.add(t);
    }

    Thread.sleep(2000);
    connector.shutdown();
    connector.join();

    for (Iterator i = targets.iterator(); i.hasNext();) {
        Target t = (Target)i.next();
        if (!t.shown)
        t.show();
    }

    }

}
```

Comparison

- Timber version
  - all actions are defined inside `ping` object
    - can safely manipulate `outstanding`
  - solution is straightforward:
    - one object, one instance variable
- Java version
  - 10 class variables
  - 3 threads
    - timeout, printing, de-multiplex of connection events
  - Less concurrency (gethostbyname bug!)

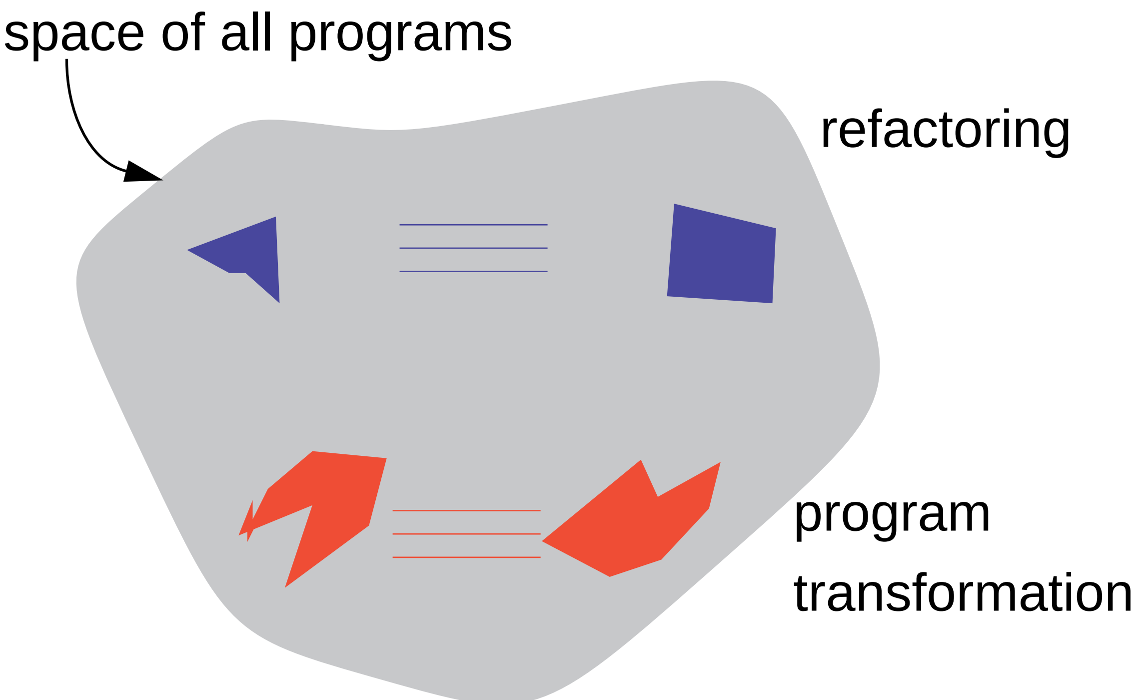
Perspectives

Joint work with Mark Jones

What comes after Text?

- Program transformations are common currency at PoPL
- Refactoring is common practice in industry
- Both are semantics preserving …

Equivalences



Use equivalence classes, not witnesses?