Isolates:
Quick overview

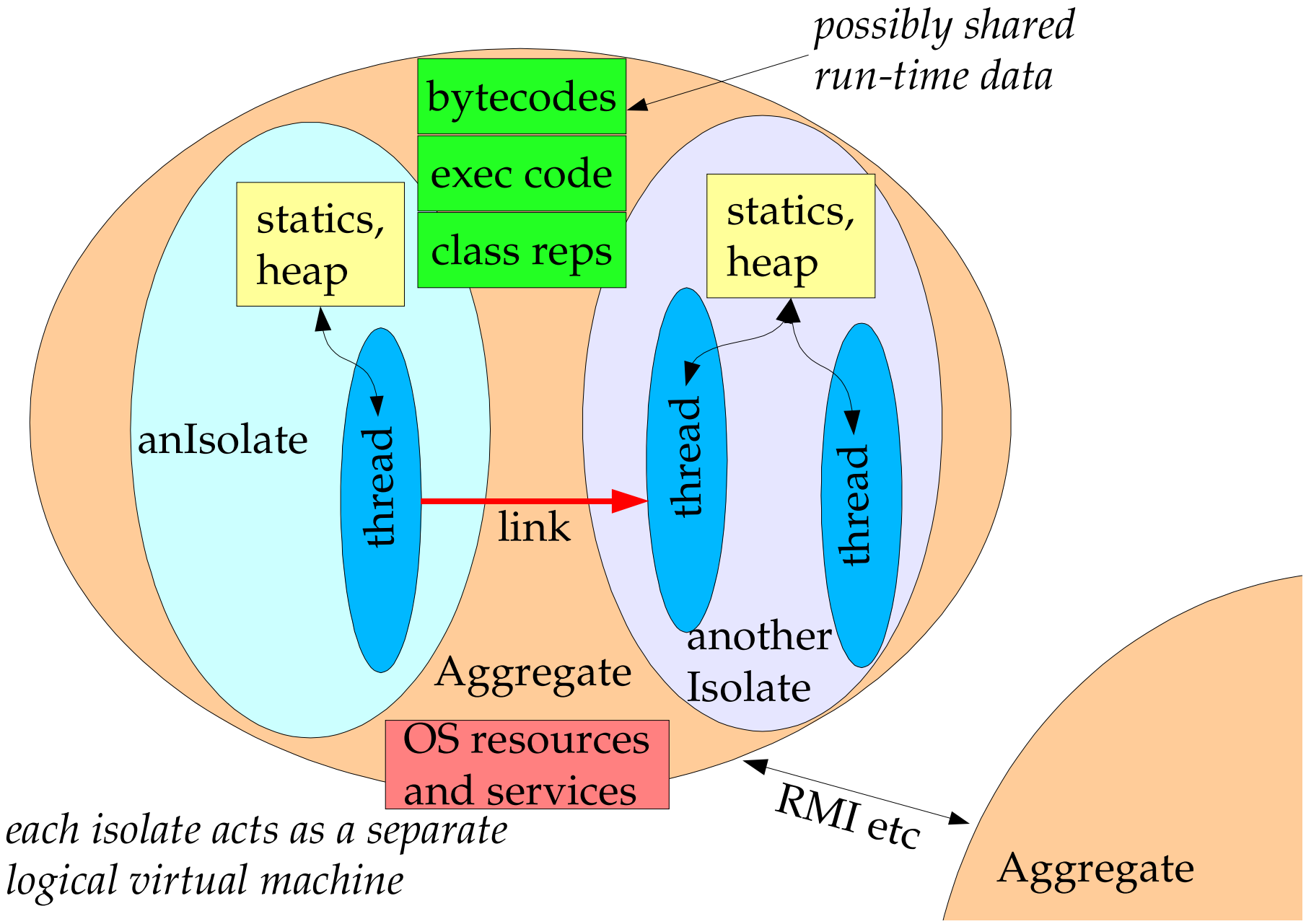
Doug Lea
SUNY Oswego
dl@cs.oswego.edu

JSR-121: Isolates

Isolate *noun*. pronounciation: *isolet*. 1. A thing that has been isolated, as by geographic, ecologic or social barriers - *American Heritage Dictionary*

- **Outline**
  - **Motivation**
  - **Some design and implementation issues**
  - **API overview and code examples**
- **Status**
  - **At public review draft in JSR-121.**

Aggregates vs Isolates vs Threads



Three Implementation Styles

- One Isolate per OS process
  - Internal sharing via OS-level shared memory, comms via IPC
    - class representations, bytecodes, compiled code, immutable statics, other internal data structures
- All Isolates in one OS address space / process managed by aggregate
  - Isolates still get own versions of all statics/globals
    - including AWT thread, shutdown hooks, ...
- LAN Cluster JVMs
  - Isolates on different machines under a common administrative domain. *NOT* a substitute for RMI
    - Little or no internal sharing

Target Usage Patterns

- Partitioning applications
  - Contained applications (*lets)
    - Applets, Servlets, Xlets, etc can run as Isolates
    - Container utility services can run as Isolates
- Service Handler Forks
  - ServerSocket.accept can launch handler for new client as Isolate
  - Pools of "warm" Isolates
- Minimizing startup time and footprint
  - User-level "`java`" program, web-start, etc can start JVM if not already present then fork Isolate
  - OS can start JVM at boot time to run daemons

More Usage Patterns

- Parallel execution on cluster JVMs
  - Java analogs of Beowulf clusters
    - Can use MPI over Links
  - Need partitioning and load-balancing frameworks
- Fault-tolerance
  - Fault detection and re-activation frameworks
  - Redundancy via multiple Isolates
- CSP style programming
  - Always use Isolates instead of Threads
  - Practically suitable only for coarse-grained designs

API Design Goals

- Minimality
  - **The smallest API that fills need**
- Mechanism, not policy
  - **Enable layered frameworks**
- Simple, clean semantics
  - **For termination, communication, etc**
- Compatibility
  - **No changes required in pre-JSR-121 code**
- Generality
  - **Allow multiple mapping strategies to platforms**

API Structure

- New Classes
  - **Isolate**
  - **Link**
  - **IsolateMessage**
  - **IsolateMessageVisitor**
  - **IsolateEvent**
  - **IsolatePermission**
  - **IsolateMessage Dispatcher**
- New Interface
  - **IsolateMessage Dispatcher.Listener**
- New Exceptions
  - **IsolateStartupException**
  - **IsolateResourceError**
  - **ClosedLinkException**
  - **LinkSerialization Exception**
- Potential changes to existing APIs
  - **java.nio: LinkChannel**
  - **java.util.prefs: TransientPreferences**
  - **Documentation clarifications**

Main Classes

- `public final class Isolate`
  - Create with name of class with a "`main`", arguments to main, plus optional standard IO bindings, classpath, security, system property and other context settings.
  - Methods to start, stop, and terminate created isolate
  - Event-based monitoring of life cycle events
- `public abstract class Link`
  - A pipe-like data channel to another isolate
    - byte arrays, ByteBuffers, Strings and serializable types
    - SocketChannels, FileChannels and other IO types (Descriptor Bearing Doobers, aka DBDs)
    - Isolates, Links

Running Independent Programs


```
void runProgram(String classname,
                String[] args) {
  try {
    new Isolate(classname, args).start();
  }
  catch (SecurityException se) { ... }
  catch (Exception other) { ... }
}
```

Starting Isolates

- Isolate creation establishes existence
  - Isolates may (but need not) perform resource allocation and internal initialization upon creation
- Static initializers then main run at `start`
  - Isolates may continue initialization before running
  - All classes are loaded in new Isolate's context
- Failures detected before running user code result in exceptions at creation or start time
  - Cannot be sure whether the same exceptions will be thrown at the same points in all Implementations
- Other failures merely terminate the Isolate

Configuration

- Inheriting execution contexts
  - Different rules and defaults for TransientPreferences context, in/out/err bindings and start messages
    - Impossible to unify all of the ways to provide initial settings while maintaining compatibility
- Other Mechanisms
  - Contained Isolates may obtain additional configuration parameters via JNDI or other means
  - Frameworks can supply a common main that establishes context and then loads application

Stopping Isolates

- Preserves distinction between exit and halt
  - `exit` causes Isolate to run shutdown hooks etc
    - Does NOT guarantee eventual termination
  - `halt` causes sure, abrupt termination
  - Isolates may also terminate for the usual reasons
  - Aggregate shuts down when ALL Isolates do
- Monitoring lifecycles
  - Receiving start, exit, terminated events
- Not hierarchical
  - Parents may terminate independently of children
    - Can layer on methods to await termination

Initializing and Monitoring


```
class Runner {
   LinkMessageDispatcher d = new LinkMessageDispatcher();

   LinkMessageDispatcher.Listener l =
     new LinkMessageDispatcher.Listener() {
       public void messageReceived
         (IsolateMessageDispatcher d, Link l,LinkMessage m){
         IsolateEvent e = m.getEvent();
         System.out.println("State change"+ e.getType());
       }};

   void runStarlet(...) throws ... {
     TransientPreferences ctx = new TransientPreferences();
     ctx.node("java.properties").put("java.class.path",...);

     IsolateMessage stdIn =
       IsolateMessage.
         newFileInputStreamMessage(new FileInputStream(...));

     Isolate p = new Isolate(..., ctx, stdIn, ...);

     d.add(p.newEventLink(Isolate.currentIsolate()), l);

     p.start();
   }}
```

Communicating

```
void appRunner() throws ... {
   Isolate child = new Isolate("Child", ...);
   Link toChild = Link.newLink(Isolate.currentIsolate(), child);
   Link fromChild = Link.newLink(child, Isolate.currentIsolate());
   app.start(new IsolateMessage[] {
     IsolateMessage.newLinkMessage(toChild),
     IsolateMessage.newLinkMessage(fromChild) } );
   toChild.send(IsolateMessage.newStringMessage("hi"));
   String reply = fromChild.receive().getString();
   System.out.println(reply);
   child.exit(0);
   Thread.sleep(10 * 1000);
   if (!app.isTerminated()) app.halt(1);
}

class Child { ...
   public static void main(...) {
     Link fromParent = Isolate.currentIsolateStartMessages()[0];
     Link toParent = Isolate.currentIsolateStartMessages()[1];
     String hi = fromParent.receive().getString();
     toParent.send(IsolateMessage.newStringMessage("bye"));
     System.exit(0);
   } }
```