http://bitser.net/isolate-interest

The Java Isolation API:
Introduction, applications and inspiration

Doug Lea
SUNY Oswego
dl@cs.oswego.edu

Pete Soper
Sun Microsystems
pete.soper@sun.com

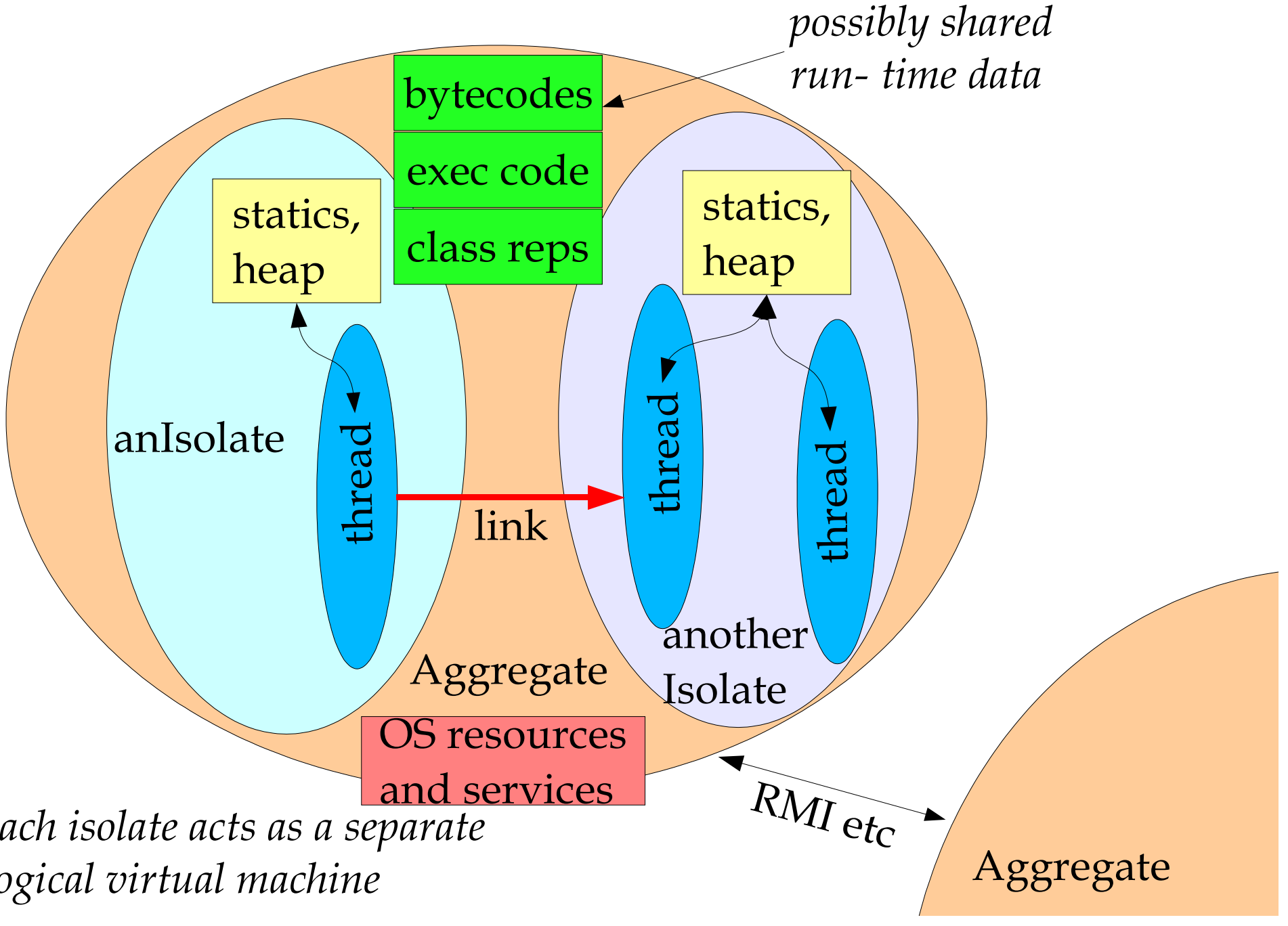
Miles Sabin
miles@milessabin.com

Overview

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
  - Application to mobility
  - Relationship to the π-calculus
- Status
  - Public review draft in JSR-121 was aimed at J2SE 1.5, now planning 2nd public draft aimed at J2SE & J2ME.

Aggregates vs Isolates vs Threads



Motivation

- Performance
  - Reduce footprint, start-up overheads for running independent programs
- Security
  - Prevent interference via shared resources or communication
  - Simplify construction of obviously secure systems
- Management
  - Configure, monitor and kill activities without disrupting others
    - Especially in container frameworks
  - Stay within Java; not OS via Runtime.exec

JVMs vs Aggregates

- Not necessarily a "single program" anymore
  - Each Isolate is a logical virtual machine
- A JVM is,
  - A running instance of the JRE
  - Strong associations with a single OS process
- An Aggregate is,
  - A container of Isolates
    - An administrative and management boundary
  - A set of services and service guarantees
    - Bytecode execution and run-time functions
- Aggregate as a less ambiguous term

Isolating State

- Visible Per-Aggregate vs per-Isolate state
  - Case-by-case analysis of statics, startup settings, global JVM state
    - See also Czajkowski et al MVM papers (in bibliography)
  - Spec requires very few global settings
    - All immutable: User identity, command-line settings
- Native methods
  - JSR-121 does not strictly guarantee that bad JNI code will not crash some or all Isolates
    - Implementations can make stronger guarantees, but at likely cost of crossing address spaces for JNI calls

Security

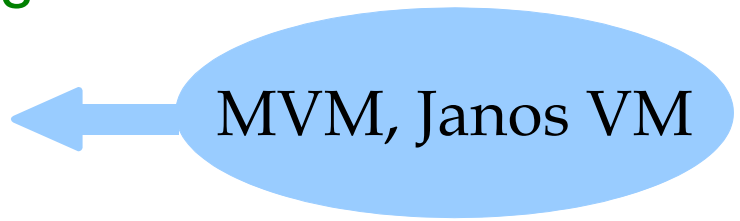
- Per-Isolate Security Managers
  - Can arrange different managers and policies for different Isolates
  - Common default security policy files
- Checks for creating, controlling and communication between isolates
  - IsolatePermission controls access (CDC&J2SE)
- Aggregate runs under single User identity
  - No Unix-style substitute-user capability
- Capability-style communication
  - Must have Link to communicate, and must have Isolate handle to create Link

Resource Management

- Not specified in JSR-121
  - ***NO*** guarantees about scheduling, heap mgt, etc
  - Hints are possible via IsolateParameters
- Current Sun Research
  - Sun technical report TR-2003-124 (in bib.)
  - More papers coming
- Interactions with system-wide resource monitoring, profiling, debugging APIs
  - JMX, JVMPI, JVMTI, etc

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
  - Isolates still get own versions of all statics/globals
    - including AWT thread, shutdown hooks, ...
- Isolates scheduled onto JVMs
- LAN Cluster JVMs
  - Isolates on different machines, one admin domain.



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 (base package)

- Package javax.isolate
  - Isolate
  - IsolateParameters
  - Link
  - DataMessage
  - StatusMessage
  - CompositeMessage
- New Interface
  - Message (just a tag)
- New Exceptions
  - IsolateStartupException
- Changes to existing APIs
  - Documentation clarifications

API Structure (additional pkgs)

- javax.isolate.tbd (CDC+)
  - IsolatePermission
  - ObjectMessage
- javax.isolate.io (J2SE)
  - IOMessage interface
  - file/network I/O classes
- javax.isolate.nio (J2SE)
  - ByteBuffer
  - ChannelMessage
- javax.isolate.util (J2SE)
  - Visitor pattern & support

Open API Design Issues

- Base package deemed too big for CLDC
  - But don't want to abandon strong typing
- Total package set deemed “overkill” even for J2SE

Main Classes

- **`public final class Isolate implements Message`**
  - Create with name of class with a "**`main`**", arguments (simple) or with IsolateParameters (two flavors of additional parms)
  - Methods to start and terminate and query isolate, get its parms and starting links
- **`Message: interface tag for Link msgs`**
- **`public class Link`**
  - A pipe-like data channel to another isolate
    - byte arrays, ByteBuffers, Strings and serializable types
    - SocketChannels, FileChannels and other IO types
    - Isolates, Links

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

Running Independent Programs


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

Configuration

- Inheriting execution contexts
  - Different rules and defaults for IsolateParameters (context, in/out/err bindings and start links)
    - 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

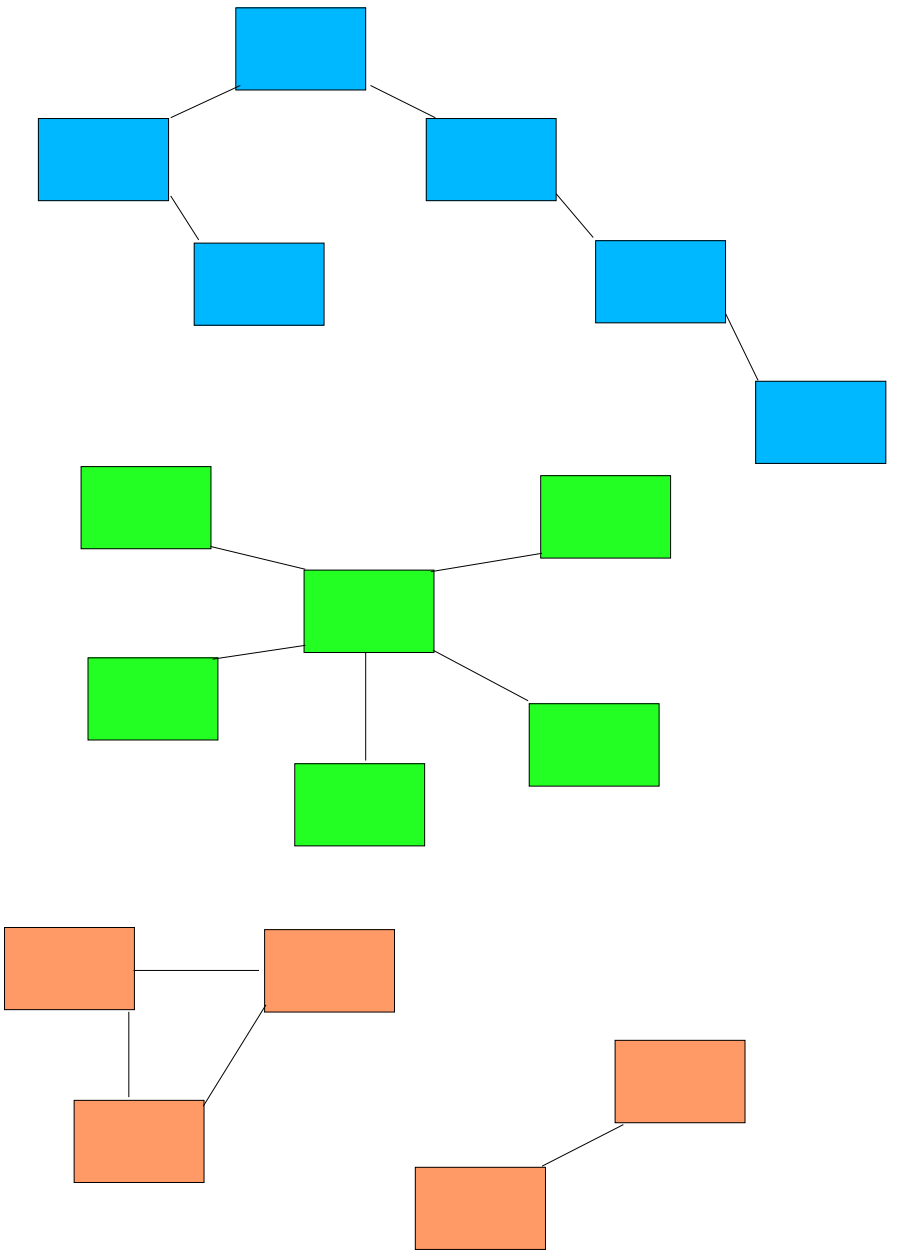
- Preserves distinction between exit and halt
  - **`exit`** causes Isolate to run shutdown hooks etc
    - Does NOT guarantee eventual termination
  - **`halt`** causes sure, abrupt termination
  - Isolates may also terminate for the usual reasons
  - Aggregate shuts down when ALL Isolates do
- Monitoring 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 {
  Link data;
  Isolate child;
  CompositeMessage getMessage() { return data.receive(); }
  StatusMessage runStarlet(String mCls, String[] mArgs,
                           String[] sec /*,...*/) {
    IsolateParameters context = new
                              IsolateParameters(mCls, mArgs);
    context.setContext(
      “jsr121.exp.java.properties.java.security.manager”,
          sec);
    child = new Isolate(context);
    data = Link.newLink(child, Isolate.currentIsolate());
    StatusLink s = child.newStatusLink();
    child.start(new Link[] { data } );
    return s.receive();
  }
}
```

Communication and Control

- App frameworks can impose policies:
  - Hierarchical
    - Parent/child trees
  - Centralized
  - Ad-hoc
  - Can add monitoring for application-specific events and/or tie to external monitoring



Communicating (old API)

```
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);
```

Target Usage Patterns

- 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
- Partitioning applications
  - Contained applications (*lets)
    - Applets, Servlets, Xlets, Midlet groups, 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

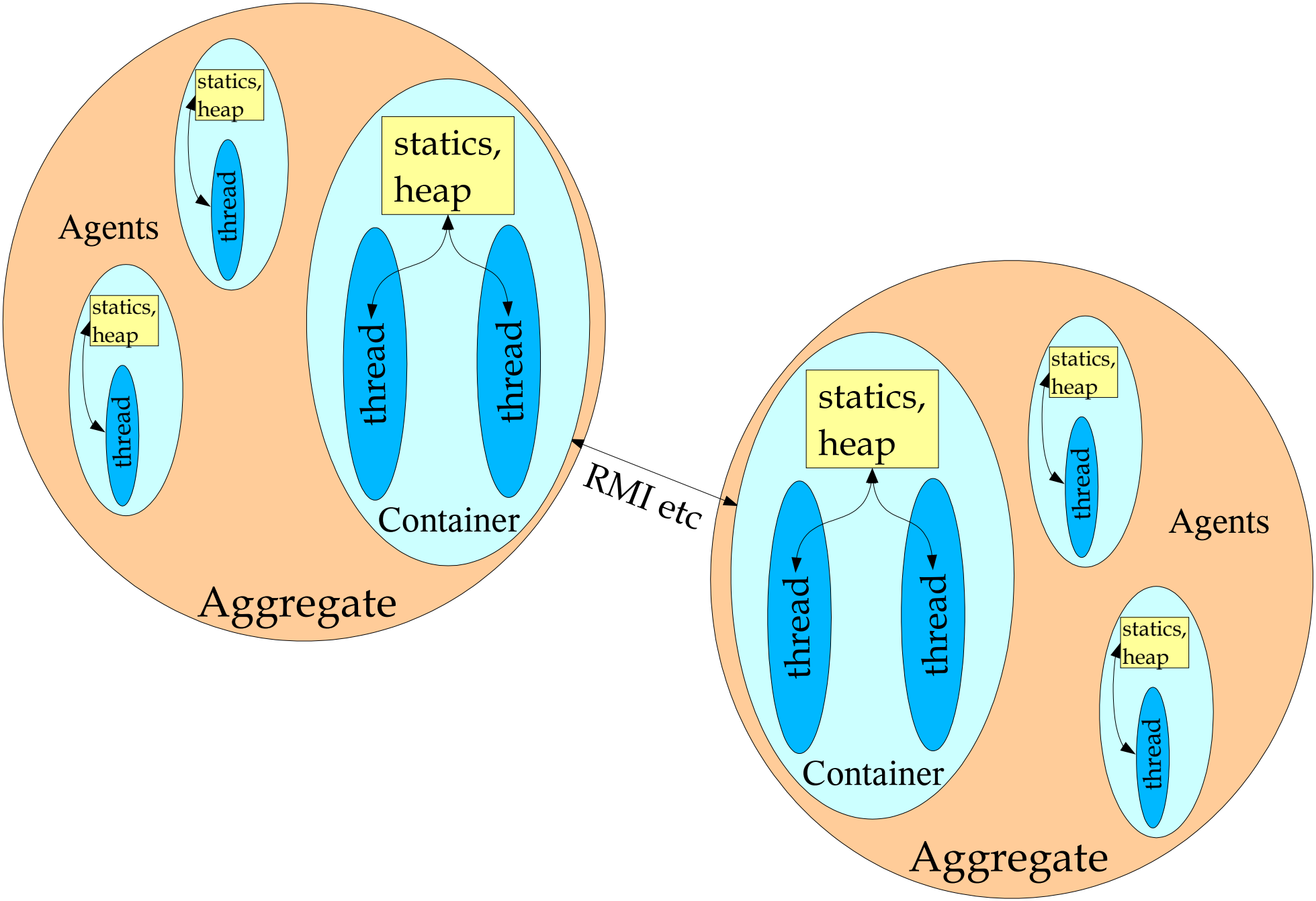
More Usage Patterns

- 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

Isolates and Mobile Code

- Issues for current agent platforms
  - Trust and reliability
    - Resource exhaustion
    - Excessive thread creation
    - ClassLoader-based containment is difficult and imperfect
  - Scalability
    - Stronger safety and robustness guarantees with a separate JVM per-agent, but resource intensive
- Isolation can help but not (yet) a panacea
  - Agent platforms structurally similar to container and *let model
  - Enables but doesn't provide resource control

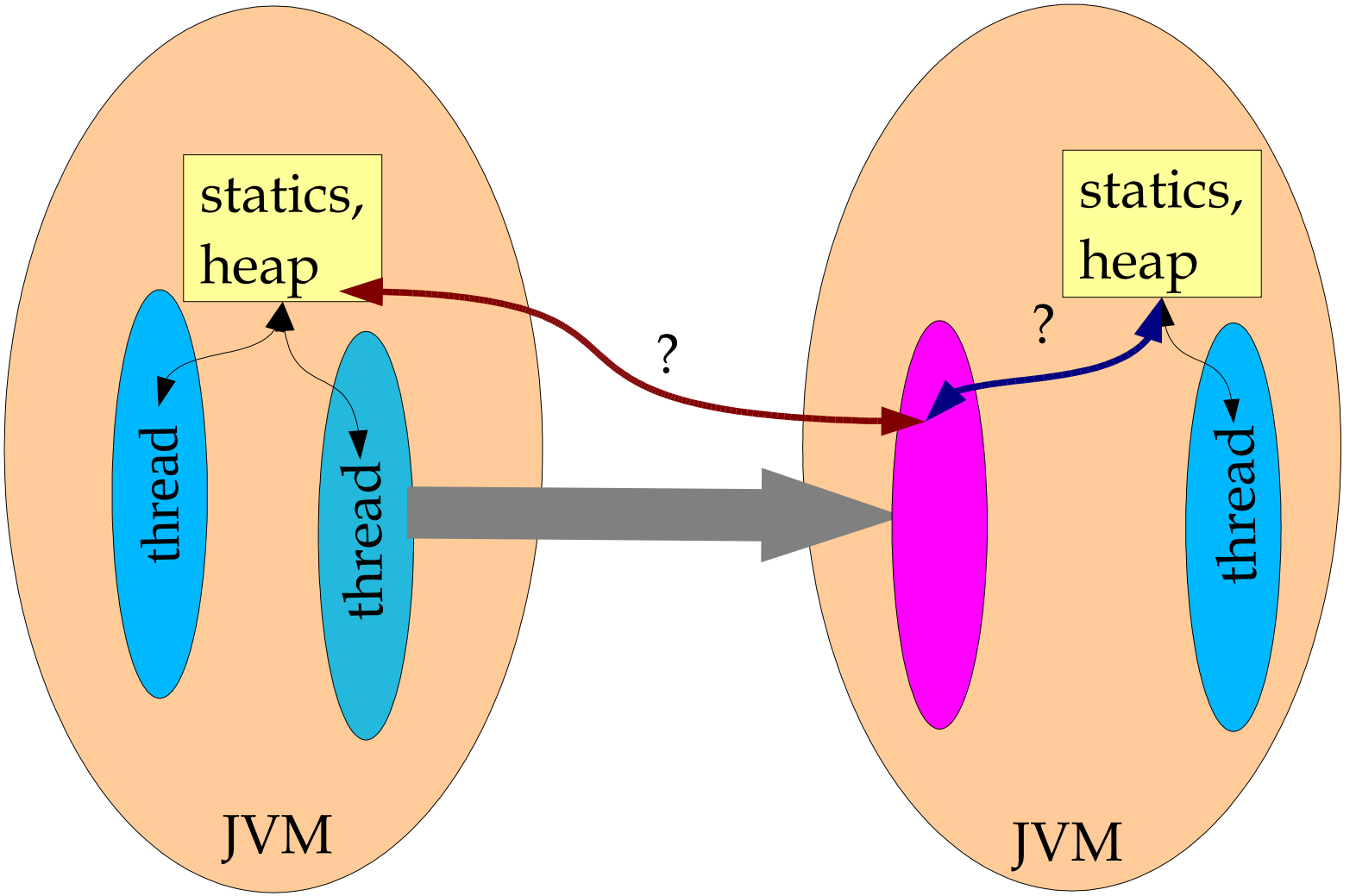
Code Migration



Mobile Processes?

- Java Threads explicitly share resources
  - So, for example, cannot always kill threads safely
- Thread migration frameworks cannot deal with
  - Objects participating in multiple threads
  - Statics, AWT, shutdown hooks, Etc
- At best, existing frameworks work when
  - You obey many unstated programming restrictions
  - You can live with a very loose definition of "work"
    - These aren't so much bugs as model mismatches
- Hasn't led to mainstream acceptance

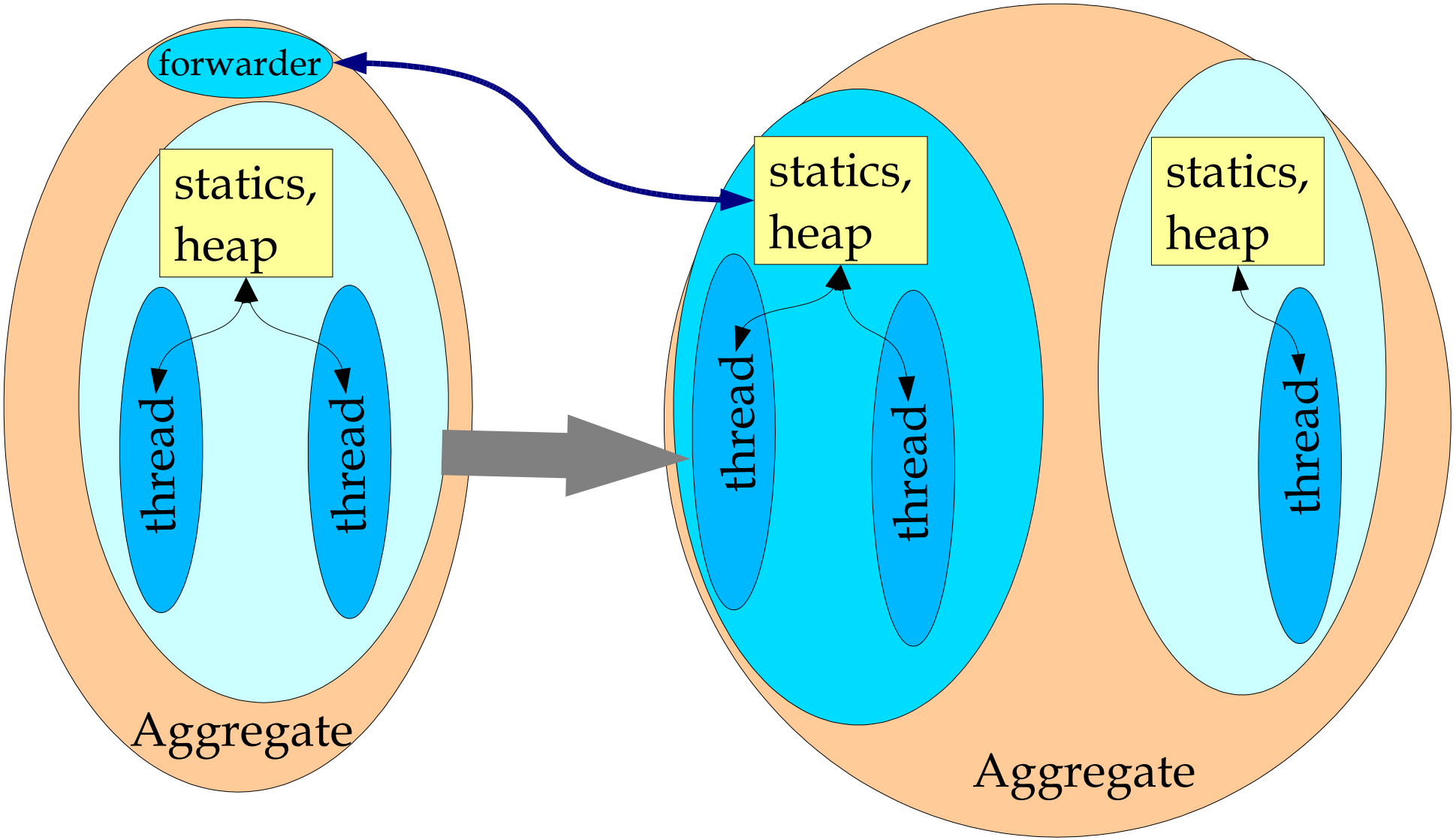
Thread Migration



Isolates and Mobile Processes?

- Isolates form a natural unit of migration
  - Enforced lack of sharing removes many obstacles
  - Practical systems seem possible
- Continued technical challenges
  - Multithreaded code
    - Use of global safe-points
  - Reconstructing execution context
    - Reconnecting to and forwarding resources
  - Between-Aggregate limitations
    - Non-transferable context: User identity, etc
    - Resource mismatches

Isolate Migration



Constraints on Mobility

- Distinguish communication/execution:
  - Within administrative domains
  - Between administrative domains
- Isolates operate ***within*** domains that offer
  - Common security and management policies
  - Reliable communication and execution
    - "*Reliable*" means: failed action implies full Aggregate failure
  - Homogenous platform

Relationship to π -calculus

- Never a goal, more a happy accident
  - Supports arbitrary communication topologies with a minimum of API
  - Eliminates the need for yet another naming and lookup scheme
    - Isolate and Link objects act as their own opaque identifiers
  - Enhanced security
    - More finely grained and dynamic access control than static type-based permissions
    - Creation of a link and control of an isolate requires that a legitimate holder of references has given out those references
  - Holds out prospect for formal proofs of security and correctness fo critical systems
  - But not complete
    - Currently links are unicast and point to point
    - Scope for extension to multicast, anycast and choice

Mappings

- Process
- Channel
- Parallel composition
- new x e
- $x!y$
- $x?y$ = e
- Choice

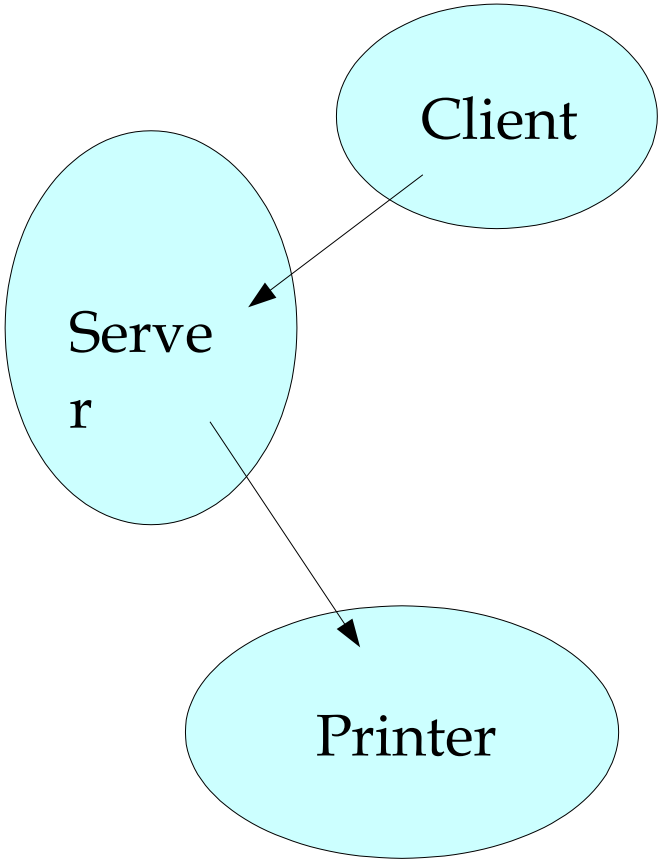
- Isolate
- Link, LinkChannel, Isolate for control and state messages
- Creation of sibling isolates
- x = Link.newLink($from$, to); $e(x)$
- x.send(y)
- y = x.receive(); $e(y)$
- IsolateMessageDispatcher, LinkChannel with NIO Selectors

A Trivial Example (old API)

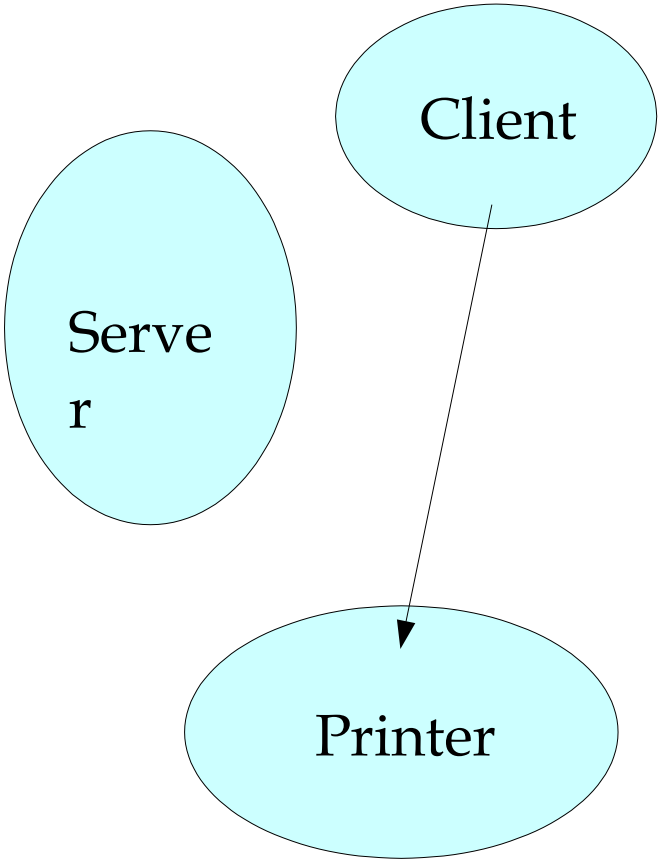
```
(new c (c!a | c?y f(y)))
```

```
class Parent {
  public static void main(String[] args) {
    Isolate firstChild = new Isolate("FirstChild", null);
    Isolate secondChild = new Isolate("SecondChild", null);
    Link l = Link.newLink(firstChild, secondChild);
    IsolateMessage[] startMessages =
      new IsolateMessage[] { IsolateMessage.newLinkMessage(l) };
    firstChild.start(startMessages);
    secondChild.start(startMessages);
  }
}

class FirstChild {
  public static void main(String[] args) {
    Link toSecond = Isolate.
      currentIsolateStartMessages()[0].getLink();
    Object a = ...;
    toSecond.send(IsolateMessage.newSerializableMessage(a));
}

class SecondChild {
  public static void main(String[] args) {
    Link fromFirst = Isolate.
      currentIsolateStartMessages()[0].getLink();
    f(fromFirst.receive().getSerializable());
}
```

Another Example



Before interaction



After interaction

Example Continued (old API)

```
class Client {
  void doPrint() {
    Link replyLink = Link.newLink(server, self);
    clientToServer.send(
      IsolateMessage.newCompositeMessage(
        new IsolateMessage[] {
          IsolateMessage.newIsolateMessage(self),
          IsolateMessage.newLinkMessage(replyLink)
    } ) );
    Link clientToPrinter = replyLink.receive().getLink();
    usePrinter(clientToPrinter);
} }

class Server {
  void handleClient() {
    IsolateMessage[] m = fromClient.receive().getComposite();
    Isolate client = m[0].getIsolate();
    Link replyLink = m[1].getLink();
    Link clientToPrinter = Link.newLink(client, printer);
    IsolateMessage lm =
      IsolateMessage.newLinkMessage(clientToPrinter)
    replyLink.send(lm);
    serverToPrinter.send(lm);
} }
```

Status

- JSR 121 page at the JCP
  - http://jcp.org/jsr/detail/121.jsp
- isolate-interest mailing list
  - http://bitser.net/isolate-interest/
- Bibliography of related work
  - http://www.bitser.net/isolate-interest/bib.html
- First public review implementations
  - http://www.cs.utah.edu/flux/janos/
    - Partial, no NIO
    - Derived from Kaffe, pre-Java2, strictly speaking not Java™
    - “many isolates to one JVM style”
  - Feature complete on two platforms, not included in J2SE 1.5
- APIs refactored and moved to javax.

Next Steps

- Upgrade JSR-121 to JCP rev 2.6
  - Unanimous EG consent
  - One EG member JSPA upgrade
- Involve Community
  - Expand EG
- Create finished spec(s), RI(s) and TCK(s)
- Rendezvous with umbrella specifications like J2SE rev X.Y, JSR-185 (wireless)
- Back to java.lang?

Credits

- Sun Task API group
  - Greg Czajkowski
  - Bill Foote
  - Hideya Kawahara
  - Tim Lindholm
  - Glenn Skinner
  - Pete Soper
- Past JSR-121 EG Members
  - Beth Hutchison, IBM
  - Jens Jensen, Ericsson
  - Peter Donald, Apache
  - Kumanan Yogaratnam, Espial
- Current EG Members
  - Dat Doan, Espial
  - Richard Houldsworth, Philips
- (Current EG cont'd)
  - Norbert Kuck, SAP
  - Doug Lea, SUNY Oswego
  - Michey Mehta, HPQ
  - Miles Sabin
  - Pete Soper, Sun (lead)
  - Patrick Tullmann, U of Utah
  - David Unietis, Oracle
  - Matthew Webster, IBM