

The following viewgraphs about
RIDL from:

D: A Framework for
Distributed Programming

Cristina Videira Lopes

Implementation of the functional specs

```
public class Shape {  
    protected double x_= 0.0, y_= 0.0;  
    protected double width_=0.0, height_=0.0;  
  
    double get_x() { return x_(); }  
    void set_x(int x) { x_ = x; }  
    double get_y() { return y_(); }  
    void set_y(int y) { y_ = y; }  
    double get_width(){ return width_(); }  
    void set_width(int w) { width_ = w; }  
    double get_height(){ return height_(); }  
    void set_height(int h) { height_ = h; }  
    void adjustLocation() {  
        x_ = longCalculation1();  
        y_ = longCalculation2();  
    }  
    void adjustDimensions() {  
        width_ = longCalculation3();  
        height_ = longCalculation4();  
    }  
}
```

state

methods

Make it distributed...

Nice functional encapsulation

```

public class Shape
    implements ShapeI {
    protected AdjustableLocation loc;
    protected AdjustableDimension dim;
    public Shape() {
        loc = new AdjustableLocation(0, 0);
        dim = new AdjustableDimension(0, 0);
    }
    double get_x() throws RemoteException {
        return loc.x(); }
    void set_x(int x) throws RemoteException {
        loc.set_x(); }
    double get_y() throws RemoteException {
        return loc.y(); }
    void set_y(int y) throws RemoteException {
        loc.set_y(); }
    double get_width() throws RemoteException {
        return dim.width(); }
    void set_width(int w) throws RemoteException {
        dim.set_w(); }
    double get_height() throws RemoteException {
        return dim.height(); }
    void set_height(int h) throws RemoteException {
        dim.set_h(); }
    void adjustLocation() throws RemoteException {
        loc.adjust();
    }
    void adjustDimensions() throws RemoteException {
        dim.adjust();
    }
}

```

```

interface ShapeI extends Remote {
    double get_x() throws RemoteException ;
    void set_x(int x) throws RemoteException ;
    double get_y() throws RemoteException ;
    void set_y(int y) throws RemoteException ;
    double get_width() throws RemoteException ;
    void set_width(int w) throws RemoteException ;
    double get_height() throws RemoteException ;
    void set_height(int h) throws RemoteException ;
    void adjustLocation() throws RemoteException ;
    void adjustDimensions() throws RemoteException ;
}

```

```

class AdjustableLocation {
    protected double x_, y_;
    public AdjustableLocation(double x, double y) {
        x_ = x; y_ = y;
    }
    synchronized double get_x() { return x_; }
    synchronized void set_x(int x) { x_ = x; }
    synchronized double get_y() { return y_; }
    synchronized void set_y(int y) { y_ = y; }
    synchronized void adjust() {
        x_ = longCalculation1();
        y_ = longCalculation2();
    }
}

```

```

class AdjustableDimension {
    protected double width_=0.0, height_=0.0;
    public AdjustableDimension(double h, double w) {
        height_ = h; width_ = w;
    }
    synchronized double get_width() { return width_; }
    synchronized void set_w(int w) { width_ = w; }
    synchronized double get_height() { return height_; }
    synchronized void set_h(int h) { height_ = h; }
    synchronized void adjust() {
        width_ = longCalculation3();
        height_ = longCalculation4();
    }
}

```

```

public class Shape
    implements ShapeI {
    protected AdjustableLocation loc;
    protected AdjustableDimension dim;
    public Shape() {
        loc = new AdjustableLocation(0, 0);
        dim = new AdjustableDimension(0, 0);
    }
    double get_x() throws RemoteException {
        return loc.x(); }
    void set_x(int x) throws RemoteException {
        loc.set_x(); }
    double get_y() throws RemoteException {
        return loc.y(); }
    void set_y(int y) throws RemoteException {
        loc.set_y(); }
    double get_width() throws RemoteException {
        return dim.width(); }
    void set_width(int w) throws RemoteException {
        dim.set_w(); }
    double get_height() throws RemoteException {
        return dim.height(); }
    void set_height(int h) throws RemoteException {
        dim.set_h(); }
    void adjustLocation() throws RemoteException {
        loc.adjust();
    }
    void adjustDimensions() throws RemoteException {
        dim.adjust();
    }
}

```

```

interface ShapeI extends Remote {
    double get_x() throws RemoteException ;
    void set_x(int x) throws RemoteException ;
    double get_y() throws RemoteException ;
    void set_y(int y) throws RemoteException ;
    double get_width() throws RemoteException ;
    void set_width(int w) throws RemoteException ;
    double get_height() throws RemoteException ;
    void set_height(int h) throws RemoteException ;
    void adjustLocation() throws RemoteException ;
    void adjustDimensions() throws RemoteException ;
}

```

```

class AdjustableLocation {
    protected double x_, y_;
    public AdjustableLocation(double x, double y) {
        x_ = x; y_ = y;
    }
    synchronized double get_x() { return x_; }
    synchronized void set_x(int x) { x_ = x; }
    synchronized double get_y() { return y_; }
    synchronized void set_y(int y) { y_ = y; }
    synchronized void adjust() {
        x_ = longCalculation1();
        y_ = longCalculation2();
    }
}

```

```

class AdjustableDimension {
    protected double width_=0.0, height_=0.0;
    public AdjustableDimension(double h, double w) {
        height_ = h; width_ = w;
    }
    synchronized double get_width() { return width_; }
    synchronized void set_w(int w) { width_ = w; }
    synchronized double get_height() { return height_; }
    synchronized void set_h(int h) { height_ = h; }
    synchronized void adjust() {
        width_ = longCalculation3();
        height_ = longCalculation4();
    }
}

```

thread synchronization

remote interaction

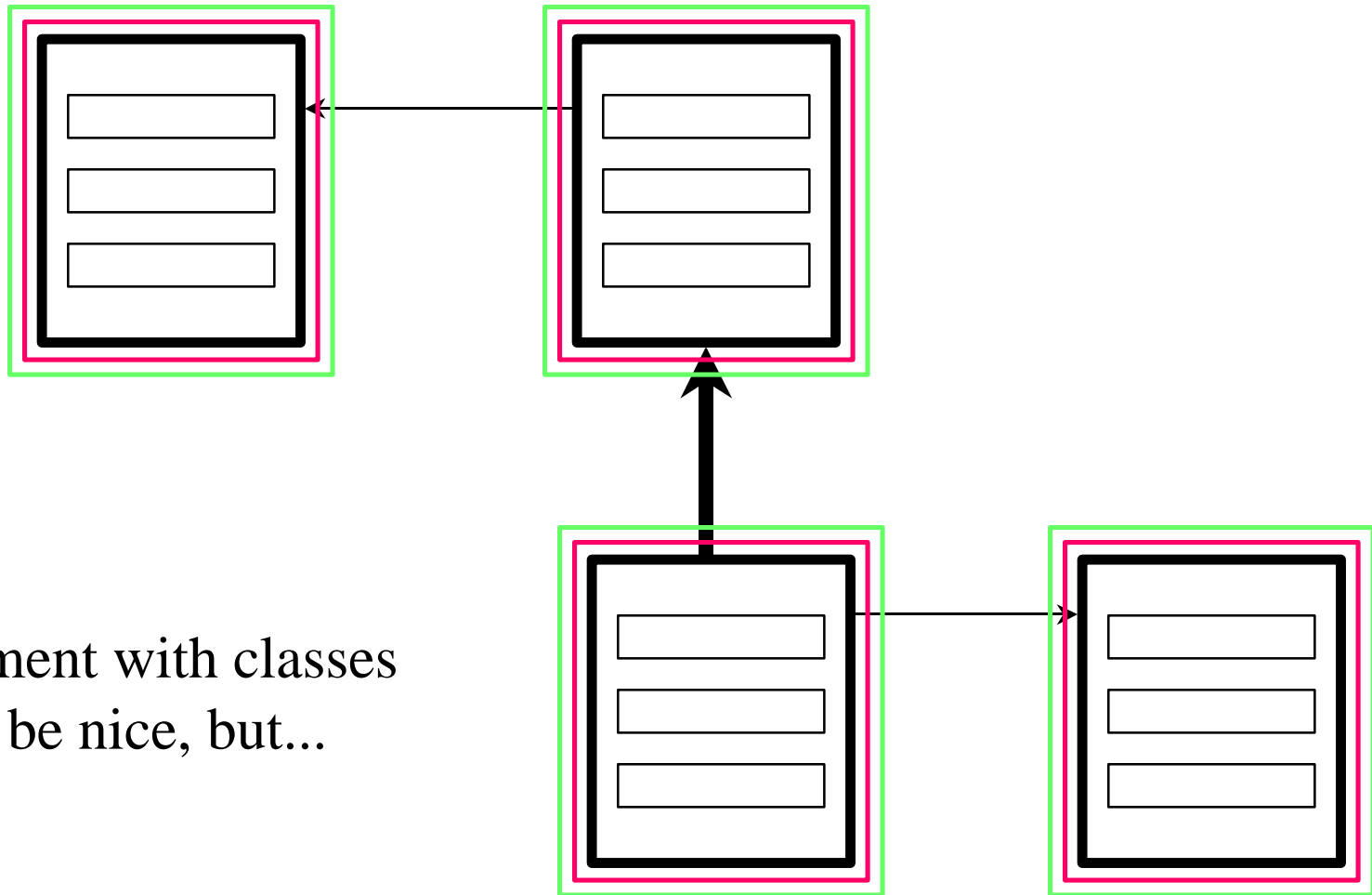
```
public class Shape
    implements ShapeI {
    protected AdjustableLocation loc;
    protected AdjustableDimension dim;
    public Shape() {
        loc = new AdjustableLocation(0, 0);
        dim = new AdjustableDimension(0, 0);
    }
    double get_x() throws RemoteException {
        return loc.x(); }
    void set_x(int x) throws RemoteException {
        loc.set_x(); }
    double get_y() throws RemoteException {
        return loc.y(); }
    void set_y(int y) throws RemoteException {
        loc.set_y(); }
    double get_width() throws RemoteException {
        return dim.width(); }
    void set_width(int w) throws RemoteException {
        dim.set_w(); }
    double get_height() throws RemoteException {
        return dim.height(); }
    void set_height(int h) throws RemoteException {
        dim.set_h(); }
    void adjustLocation() throws RemoteException {
        loc.adjust();
    }
    void adjustDimensions() throws RemoteException {
        dim.adjust();
    }
}
```

```
interface ShapeI extends Remote {
    double get_x() throws RemoteException ;
    void set_x(int x) throws RemoteException ;
    double get_y() throws RemoteException ;
    void set_y(int y) throws RemoteException ;
    double get_width() throws RemoteException ;
    void set_width(int w) throws RemoteException ;
    double get_height() throws RemoteException ;
    void set_height(int h) throws RemoteException ;
    void adjustLocation() throws RemoteException ;
    void adjustDimensions() throws RemoteException ;
}
```

```
class AdjustableLocation {
    protected double x_, y_;
    public AdjustableLocation(double x, double y) {
        x_ = x; y_ = y;
    }
    synchronized double get_x() { return x_; }
    synchronized void set_x(int x) { x_ = x; }
    synchronized double get_y() { return y_; }
    synchronized void set_y(int y) { y_ = y; }
    synchronized void adjust() {
        x_ = longCalculation1();
        y_ = longCalculation2();
    }
}
```

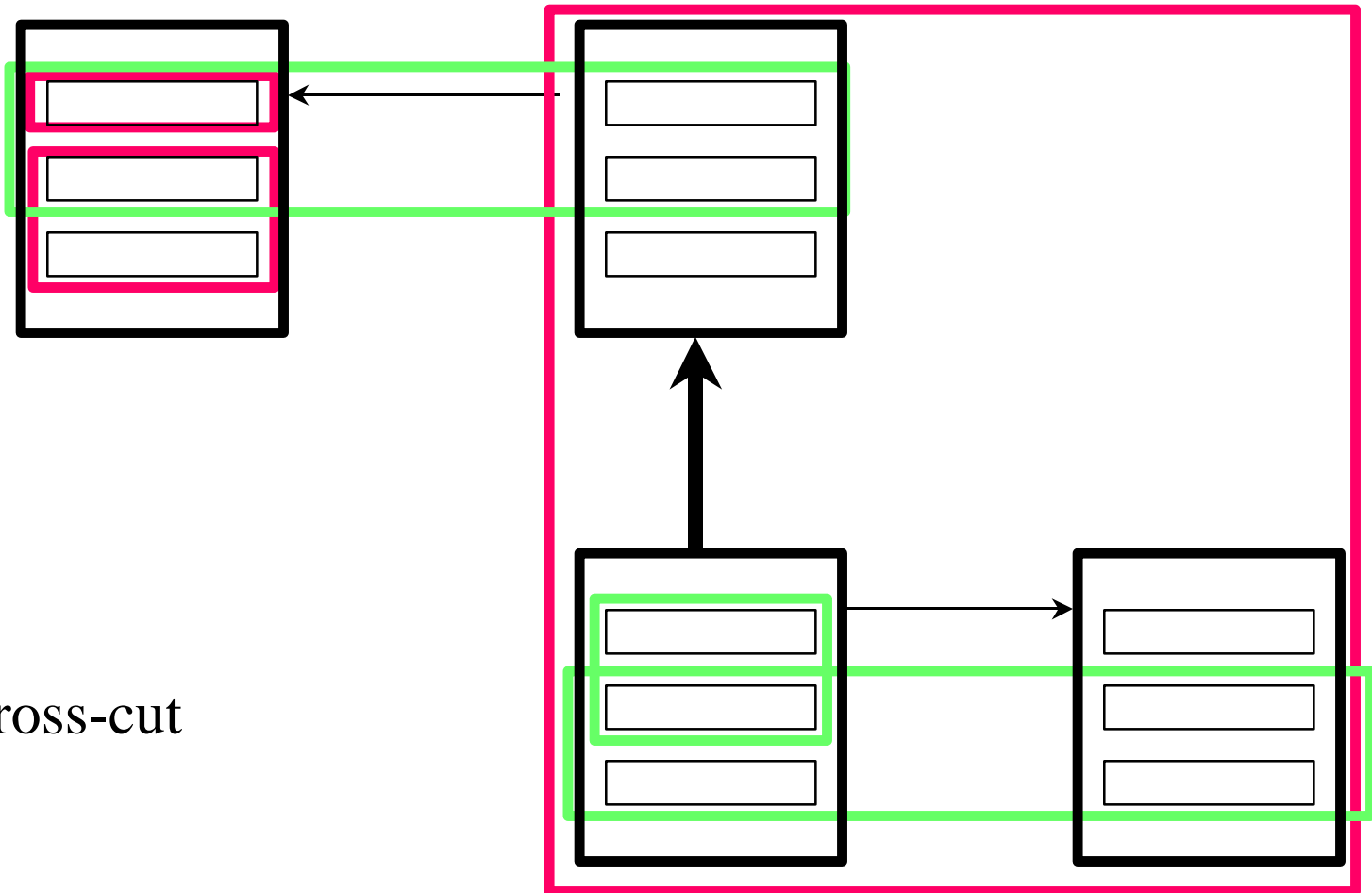
```
class AdjustableDimension {
    protected double width_=0.0, height_=0.0;
    public AdjustableDimension(double h, double w) {
        height_ = h; width_ = w;
    }
    synchronized double get_width() { return width_; }
    synchronized void set_w(int w) { width_ = w; }
    synchronized double get_height() { return height_; }
    synchronized void set_h(int h) { height_ = h; }
    synchronized void adjust() {
        width_ = longCalculation3();
        height_ = longCalculation4();
    }
}
```

The source of tangling



Alignment with classes
would be nice, but...

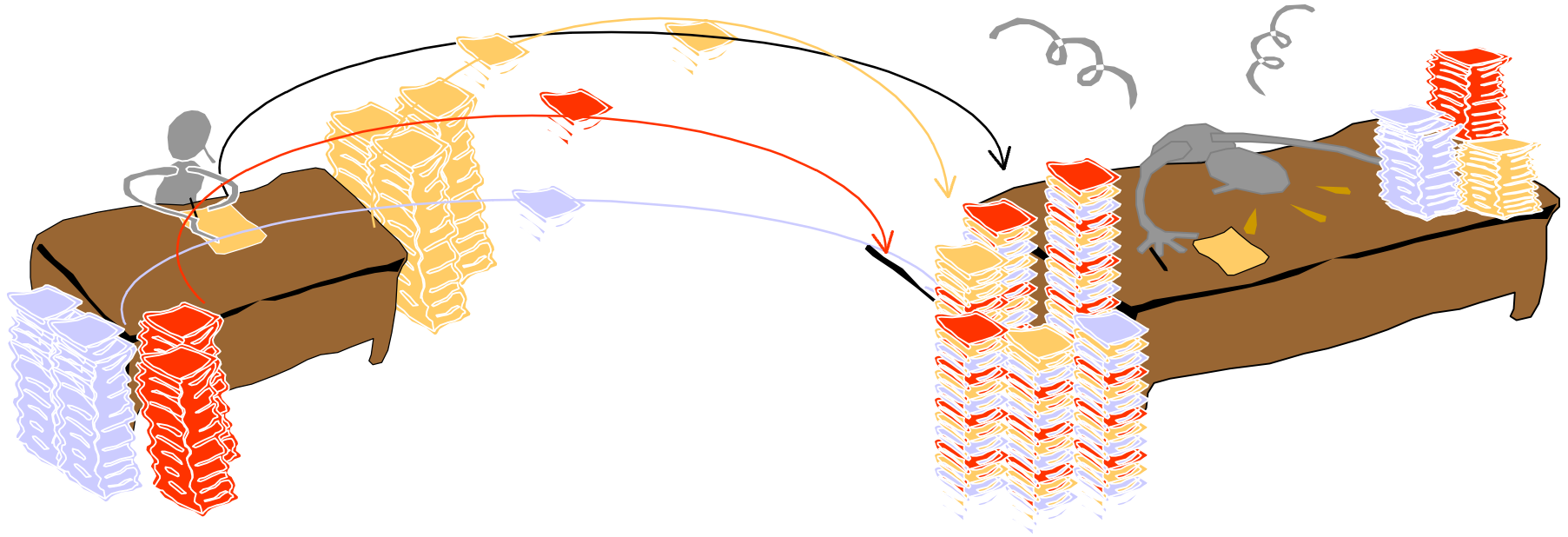
The source of tangling



...issues cross-cut
classes



What is the problem? **Tangling!**



During implementation
separate issues
are mixed together

During maintenance individual
issues need to be
factored out of the tangled code


D

```
interface ShapeI extends Remote {
    double get_x() throws RemoteException ;
    void set_x(int x) throws RemoteException ;
    double get_y() throws RemoteException ;
    void set_y(int y) throws RemoteException ;
    double get_width() throws RemoteException ;
    void set_width(int w) throws RemoteException ;
    double get_height() throws RemoteException ;
    void set_height(int h) throws RemoteException ;
    void adjustLocation() throws RemoteException ;
    void adjustDimensions() throws RemoteException ;
}
public class Shape
    implements ShapeI {
    protected AdjustableLocation loc;
    protected AdjustableDimension dim;
    public Shape() {
        loc = new AdjustableLocation(0, 0);
        dim = new AdjustableDimension(0, 0);
    }
    double get_x() throws RemoteException {
        return loc.x();
    }
    void set_x(int x) throws RemoteException {
        loc.set_x(x);
    }
    double get_y() throws RemoteException {
        return loc.y();
    }
    void set_y(int y) throws RemoteException {
        loc.set_y(y);
    }
    double get_width() throws RemoteException {
        return dim.width();
    }
    void set_width(int w) throws RemoteException {
        dim.set_w(w);
    }
    double get_height() throws RemoteException {
        return dim.height();
    }
    void set_height(int h) throws RemoteException {
        dim.set_h(h);
    }
    void adjustLocation() throws RemoteException {
        loc.adjust();
    }
    void adjustDimensions() throws RemoteException {
        dim.adjust();
    }
}
class AdjustableLocation {
    protected double x_, y_;
    public AdjustableLocation(double x, double y) {
        x_ = x; y_ = y;
    }
    synchronized double get_x() { return x_; }
    synchronized void set_x(int x) {x_ = x;}
    synchronized double get_y() { return y_; }
    synchronized void set_y(int y) {y_ = y;}
    synchronized void adjust() {
        x_ = longCalculation1();
        y_ = longCalculation2();
    }
}
class AdjustableDimension {
    protected double width_=0.0, height_=0.0;
    public AdjustableDimension(double h, double w) {
        height_ = h; width_ = w;
    }
    synchronized double get_width() { return width_; }
    synchronized void set_w(int w) {width_ = w;}
    synchronized double get_height() { return height_; }
    synchronized void set_h(int h) {height_ = h;}
    synchronized void adjust() {
        width_ = longCalculation3();
        height_ = longCalculation4();
    }
}
```

Write
this



Instead of
writing this



```
public class Shape {
    protected double x_= 0.0, y_= 0.0;
    protected double width_=0.0, height_=0.0;

    double get_x() { return x_(); }
    void set_x(int x) { x_ = x; }
    double get_y() { return y_(); }
    void set_y(int y) { y_ = y; }
    double get_width(){ return width_(); }
    void set_width(int w) { width_ = w; }
    double get_height(){ return height_(); }
    void set_height(int h) { height_ = h; }
    void adjustLocation() {
        x_ = longCalculation1();
        y_ = longCalculation2();
    }
    void adjustDimensions() {
        width_ = longCalculation3();
        height_ = longCalculation4();
    }
}
```

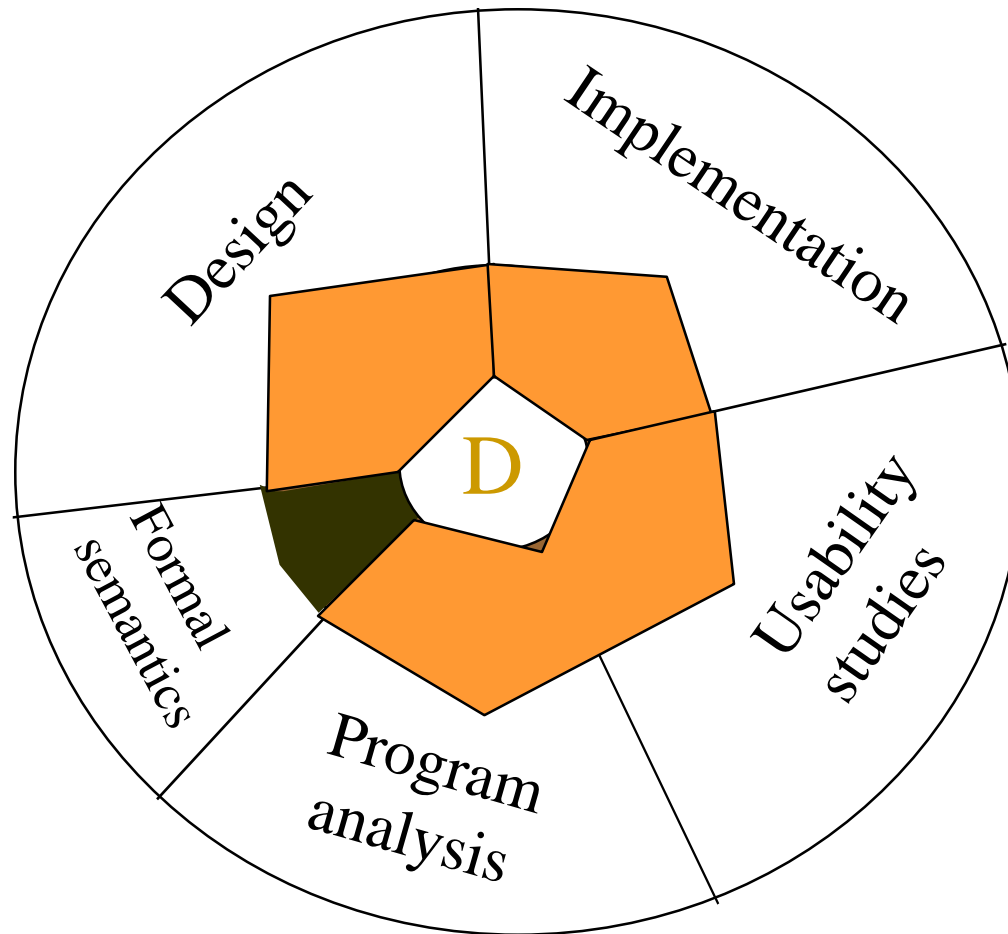
```
coordinator Shape {
    selfex adjustLocation, adjustDimensions;
    mutex {adjustLocation, get_x, set_x,
           get_y, set_y};
    mutex {adjustDimensions, get_width, get_height,
           set_width, set_height};
}
```

```
portal Shape {
    double get_x() {};
    void set_x(int x) {};
    double get_y() {};
    void set_y(int y) {};
    double get_width() {};
    void set_width(int w) {};
    double get_height() {};
    void set_height(int h) {};
    void adjustLocation() {};
    void adjustDimensions() {};
}
```

Thesis

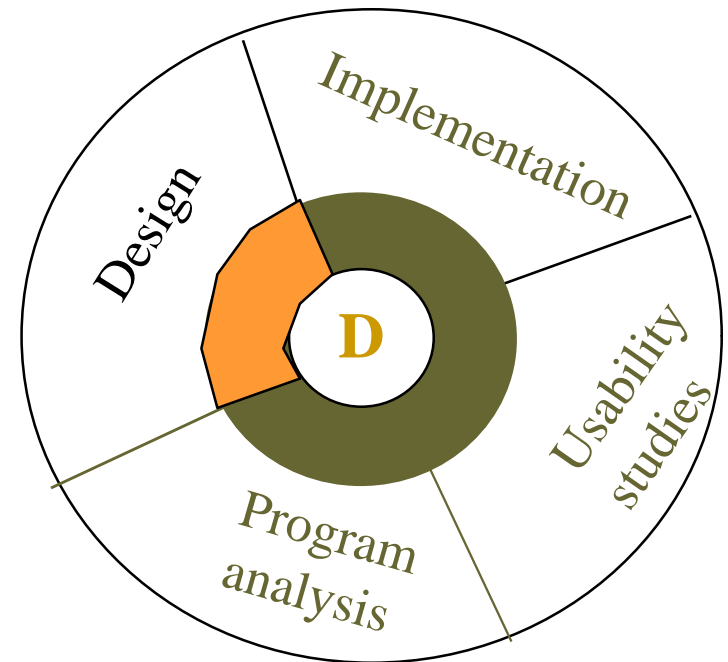
- distribution concerns can be untangled from functionality code by providing new composition mechanisms:
 - given by new and separate languages
 - smoothly integrated with OOPL
 - very low cost
- distributed programs are easier to write and understand

Presentation



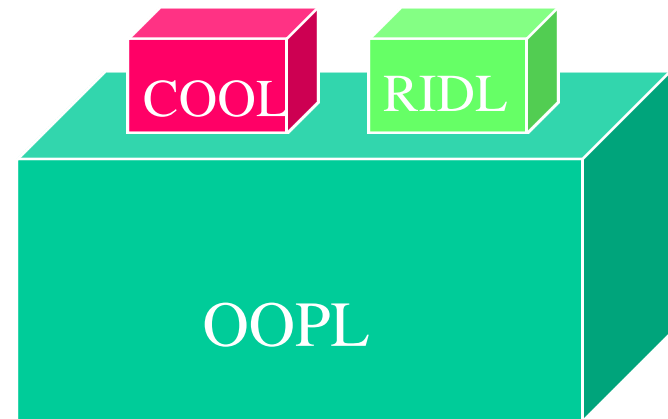
Outline

- Overview
- **D: Design**
- **D: Implementation**
- Validation Results
- Conclusion



What is D

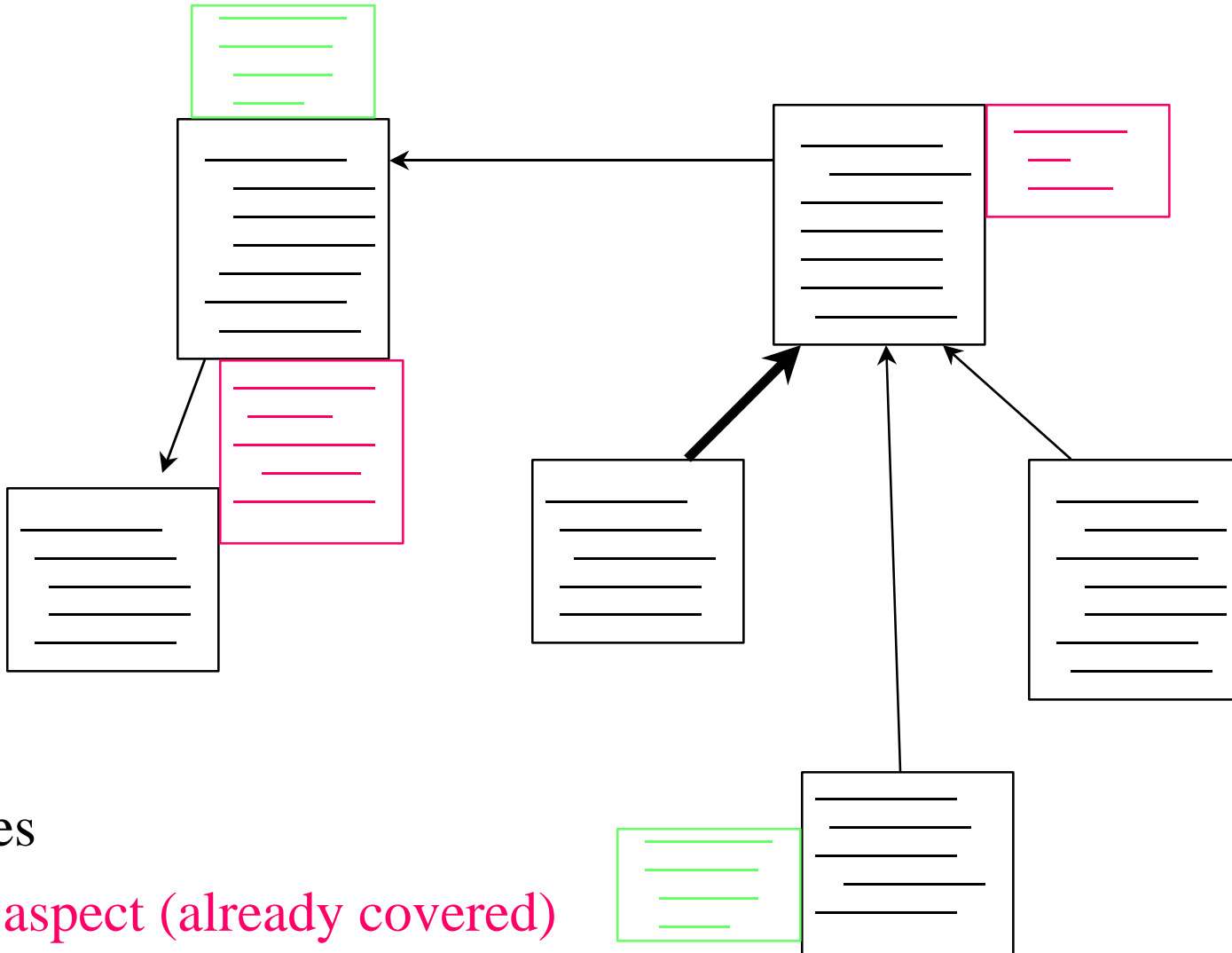
- **COOL**: language for programming thread synchronization
- **RIDL**: language for programming remote interaction and data transfers
- Cooperating with OOPL



Goals of D

- To decrease code tangling by dividing programs both in
 - units of functionality (components)
 - units of control over concurrency and distribution (aspects)
 - (can't do this well with OO...)

Programming in D



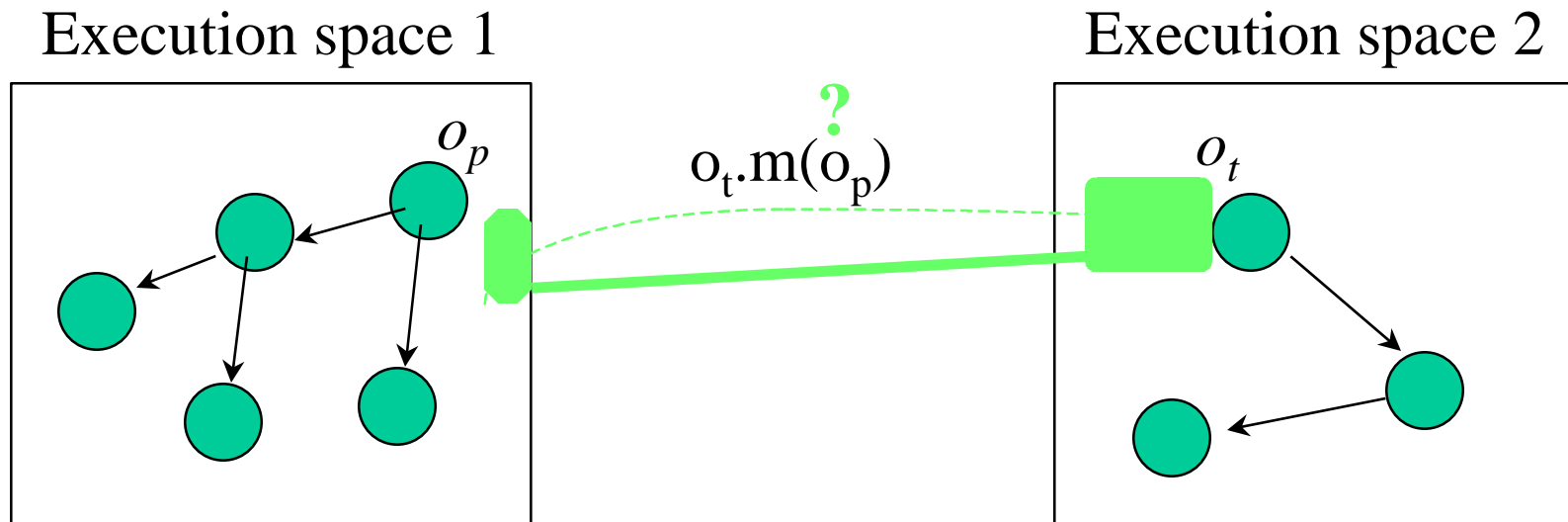
classes

Cool aspect (already covered)

Ridl aspect

RIDL

- provides means for dealing with data transfers between different execution spaces



Portals

RIDL

- Identifies “good” abstractions for controlling remote interactions of OO programs
 - remote method calls
 - different parameter passing semantics
 - selective object copying
 - ...

Sources:

Study of many distributed programs

RIDL Book Locator / Printer

```
class Book {
    protected String title, author;
    protected int isbn;
    protected OCRImage firstpage;
    protected Postscript ps;
}
```

```
class Location {
    private String building;
}
```

```
class BookLocator {
    private Book books[];
    private Location locations[];
    public void register(Book b, Location l){
        // Verify and add book b to database
    }
    public Location locate (String title) {
        Location loc;
        // Locate book and get its location
        return loc;
    }
}
```

```
class Printer {
    public void print(Book b) {
        // Print the book
    }
}
```

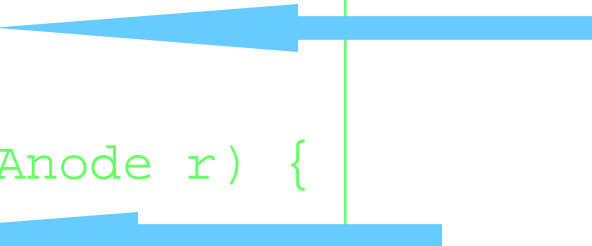
```
portal BookLocator {
    void register (Book book,
                  Location l);
    Location locate (String title)
    default:
        Book: copy{Book only title,
                  author,
                  isbn;}
}
```

```
coordinator BookLocator {
    selfex register;
    mutex {register, locate};
}
```

```
portal Printer {
    void print(Book book) {
        book: copy { Book only title,
                   ps; }
    }
}
```

Parameter Passing Modes

```
portal ANode {  
    ANode get_right() {  
        return: gref;  
    };  
    void set_right(ANode r) {  
        r: copy;  
    };  
}
```

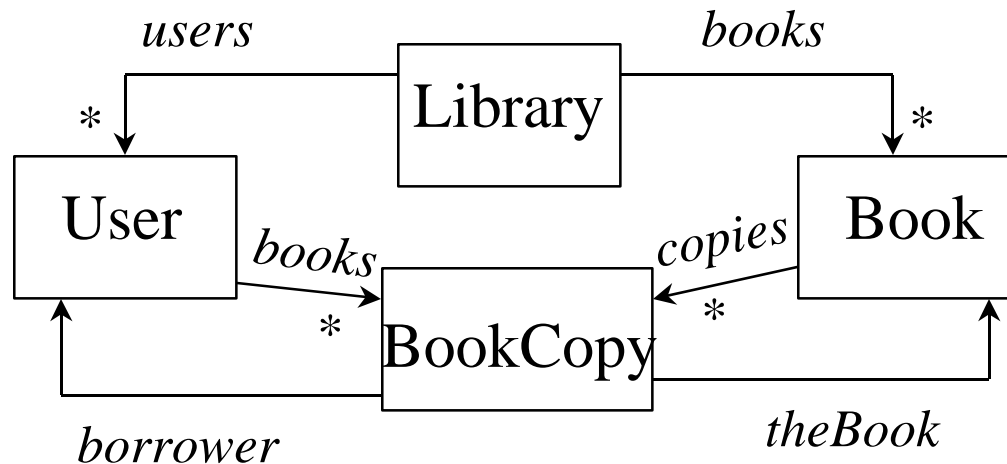
Two blue arrows point from the right side of the slide towards the code. The first arrow points to the line 'return: gref;' in the 'get_right()' function. The second arrow points to the parameter 'r' in the 'set_right(ANode r)' function signature.

Selective Marshaling

```
class Book {  
    protected String title, author;  
    protected int isbn;  
    protected OCRImage firstpage;  
    protected Postscript ps;  
}
```

```
portal Printer {  
    void print(Book book) {  
        book: copy { Book only title, ps; }  
    }  
}
```

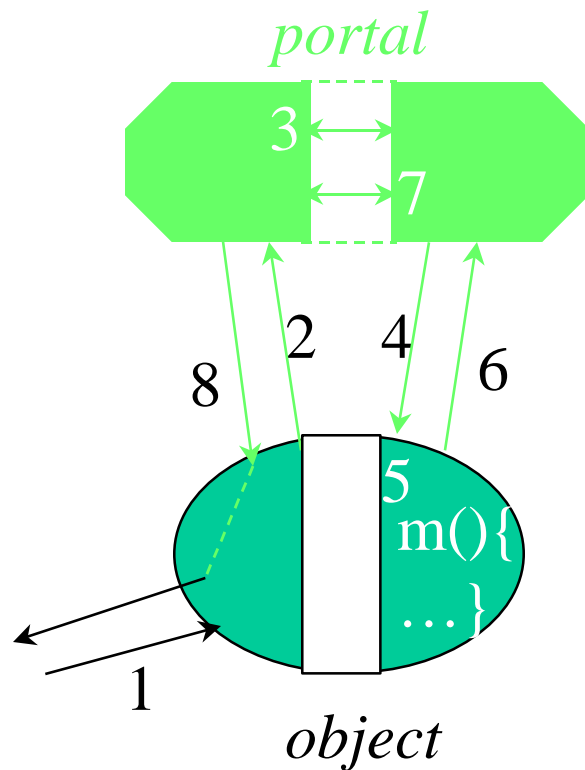
Selective Marshaling



```
portal Library {
  BookCopy getBook(User u, String title) {
    return: copy {BookCopy bypass borrower,
                  Book bypass copies;}
    u: copy {User bypass books;}
  }
  Book findBook(String title) {
    return: copy {Book bypass copies, ps;}
  }
}
```

Programming with RIDL

Protocol object/portal:



- 1: remote method invocation
- 2: request presented to the portal
- 3: parameters extracted according to transfer specifications
- 4: request proceeds to the object
- 5: method execution
- 6: return is presented to portal
- 7: return value processed according to transfer specification
- 8: method returns; return value sent

RIDL View of Classes

- Stronger and more global visibility:
 - portal can access:
 - all methods of its class, independent of access control; all non-private methods of superclasses
 - all variables of classes of parameters and of any objects that they contain
- Limited actions:
 - only read variables, not modify them
 - only define remote methods, not invoke them

D Design Points

COOL

- provider defines synchronization
- smallest unit of synchronization is the method
- coordination contained within one coordinator
- association between coordinator and object is static

RIDL

- provider defines remote interaction
- smallest unit of remote interaction is the method
- remote interaction contained within one portal
- association between portal and object is static

D Design Principles

- Separation of concerns
- Enforcement of the separation
- Add-on integration with existing languages

Demeter/Java with COOL and RIDL

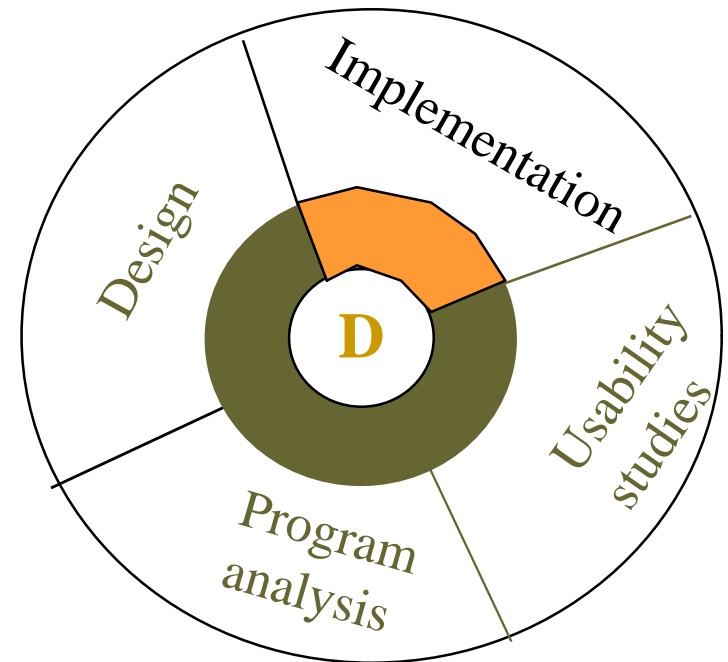
- The two aspect languages of D
- Java as the component language
 - no overloading (constructors ok) ← *control of effort*
 - no synchronized qualifier/statement ← *semantic*
 - no wait/notify methods ← *strengthening*
 - no “Remote” or “Serializable” interfaces

Summary

- D is two languages, add-ons to an OOPL:
 - COOL for matters of thread synchronization
 - RIDL for matters of remote interaction

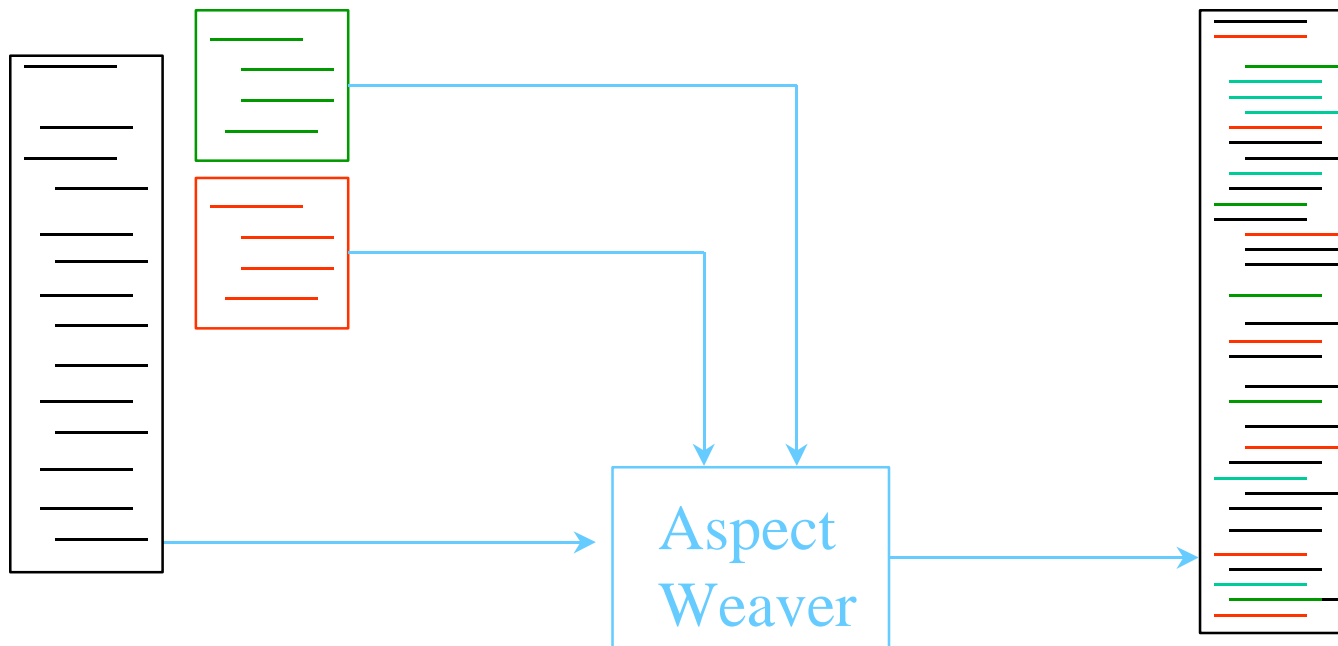
Outline

- Overview
- D: Design
- D: Implementation
- Validation Results
- Conclusion



The Aspect Weaver

Tool that automates program transformations

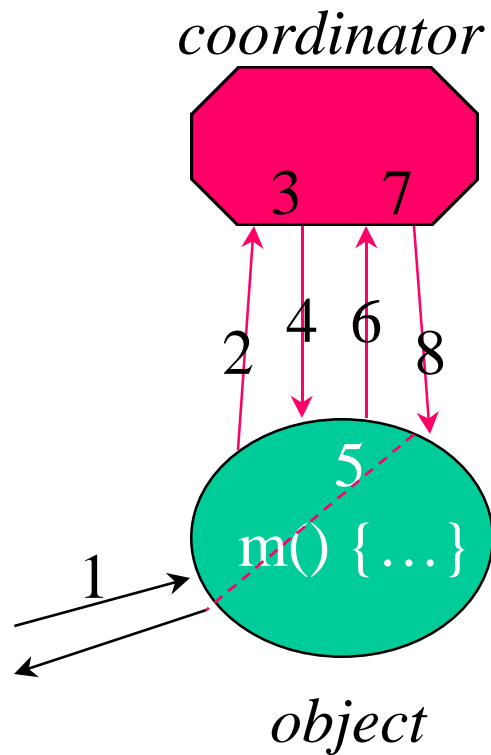


Target Architectures (the output code)

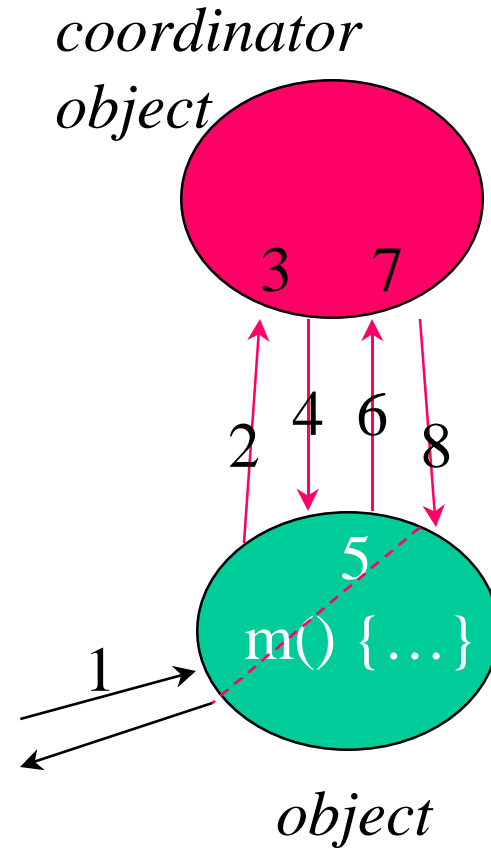
- Translation of aspect modules +
Woven code in the classes +
library
- Simplicity over optimization
- In a real tool: re-design these architectures!
(must optimize)

Programming with COOL

Implementing COOL

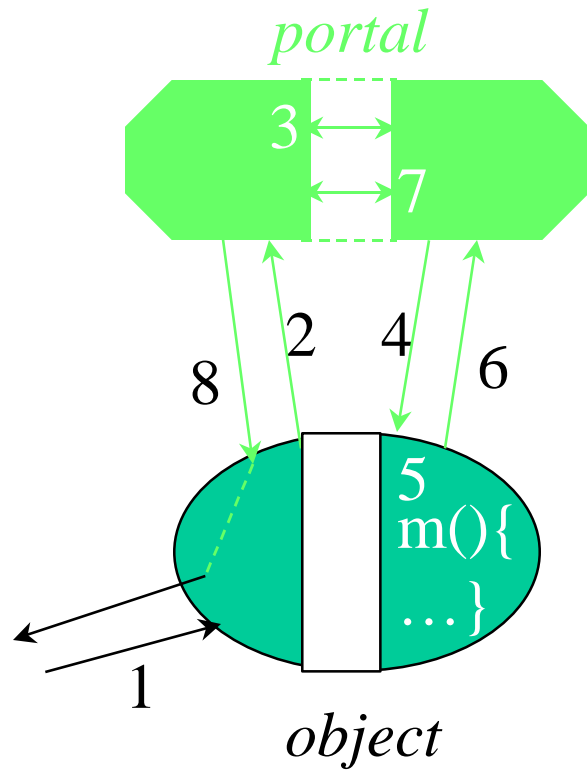


Semantics

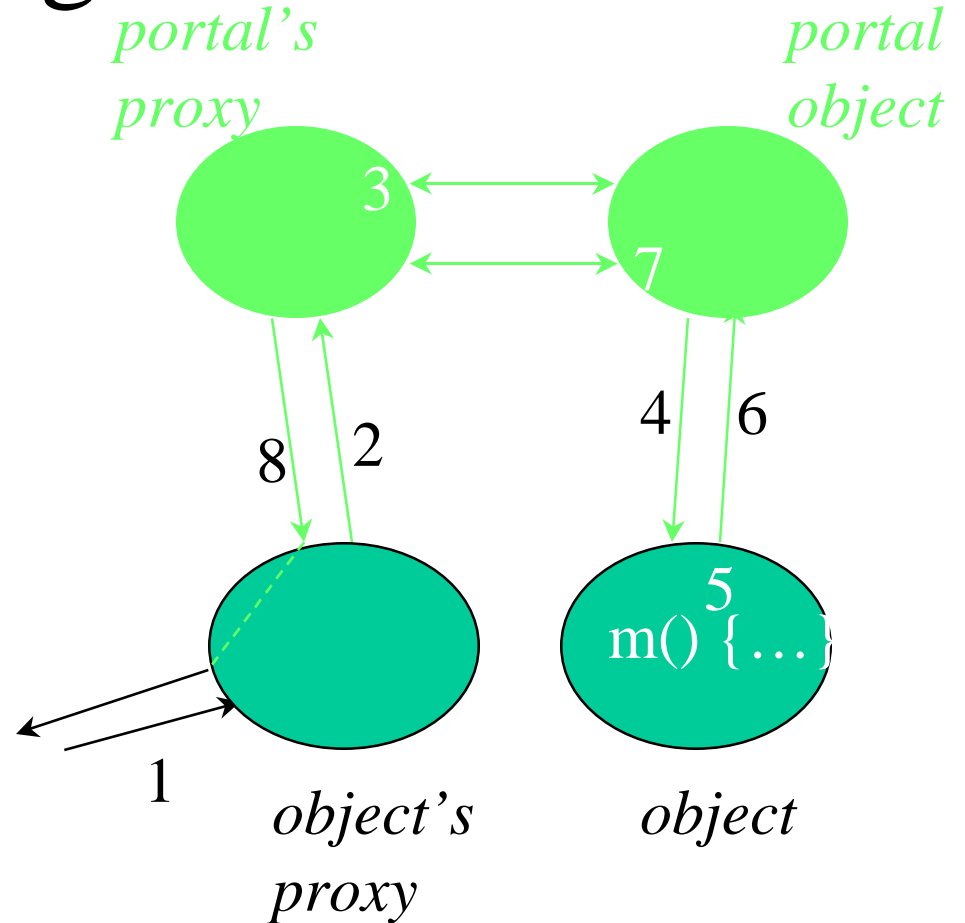


Implementation

Implementing *RIDL* Programming with *RIDL*



Semantics



Implementation

RIDL

APPLICATION
LAYER

Space 1
(client of aObj)

Space 2

real reference

virtual reference

the object's proxy



the "real" object

R
I
D
L

the portal proxy



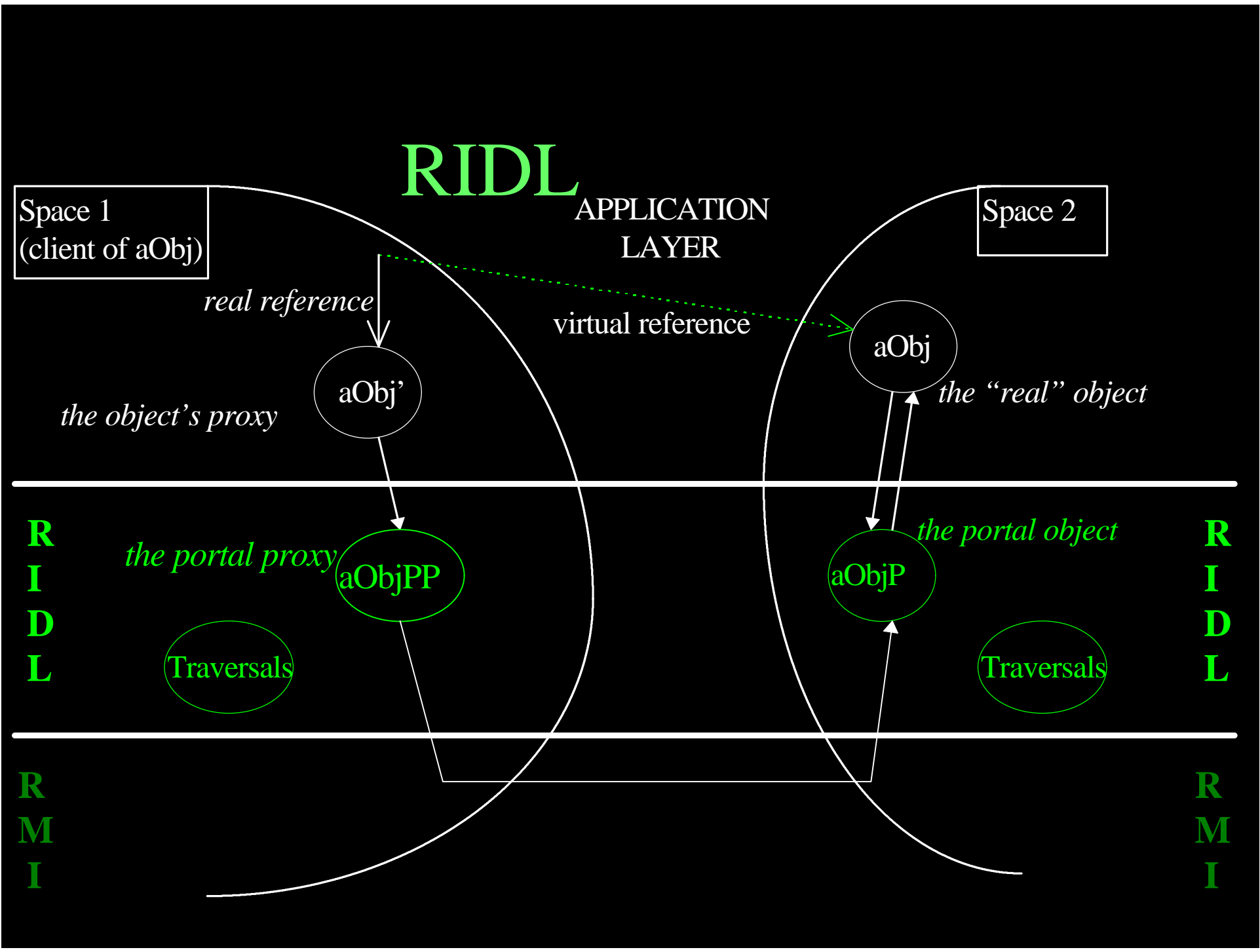
the portal object



R
I
D
L

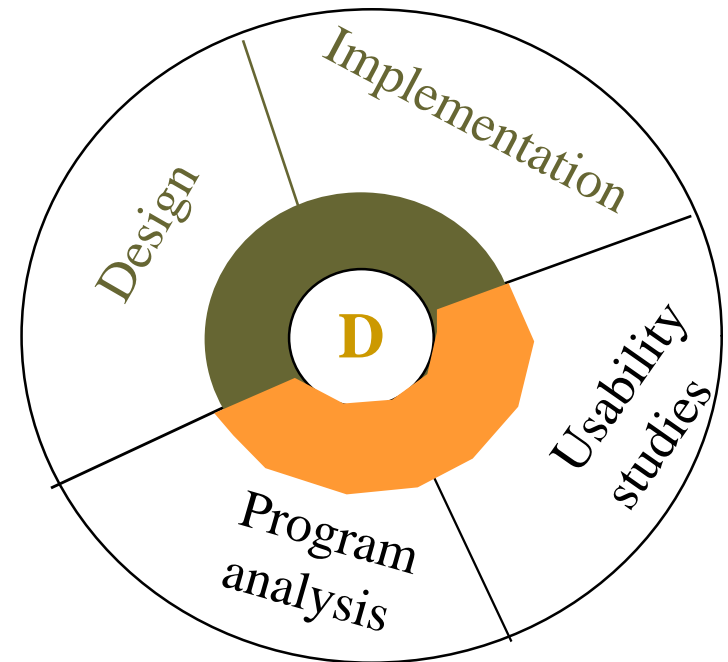
R
M
I

R
M
I



Outline

- Overview
- D: Design
- D: Implementation
- Validation Results
- Conclusion



Thesis

- distribution concerns can be untangled from functionality code by providing new composition mechanisms:
 - given by new and separate languages
 - smoothly integrated with OOPL
 - very low cost
- distributed programs are easier to write and understand

Results for DJ (Crista's implementation of D)

- Case-studies
 - empirical study
 - benefits of the design
- Performance
 - cost of implementation (target architectures)
- Alpha-usage
 - human understanding
 - acceptance

Case-Studies

- 10 small applications
 - two implementations: DJ and plain Java
- analysis: identification of aspect code
 - synchronized qualifier/statement
 - wait/notify
 - variables used for synchronization state
 - Remote interface / RemoteException
 - splitting parts design
 - ...

Bounded Buffer

```
public class BoundedBuffer {
    private Object array[];
    private int putPtr = 0, takePtr = 0;
    private int usedSlots=0;

    public void put(Object o) {
        array[putPtr] = o;
        putPtr = (putPtr + 1) % array.length;
        usedSlots++;
    }

    public Object take() {
        Object old = array[takePtr];
        array[takePtr] = null;
        takePtr = (takePtr + 1) % array.length;
        usedSlots--;
        return old;
    }
}
```

```
coordinator BoundedBuffer {
    selfex put, take;
    mutex {put, take};
    cond full = false, empty = true;
    put: requires !full;
    on_exit {
        empty = false;
        if (usedSlots == array.length)
            full = true;
    }
    take: requires !empty;
    on_exit {
        full = false;
        if (usedSlots == 0) empty = true;
    }
}
```

DJ

```
public class BoundedBuffer {
    private Object[] array;
    private int putPtr = 0, takePtr = 0;
    private int usedSlots = 0;

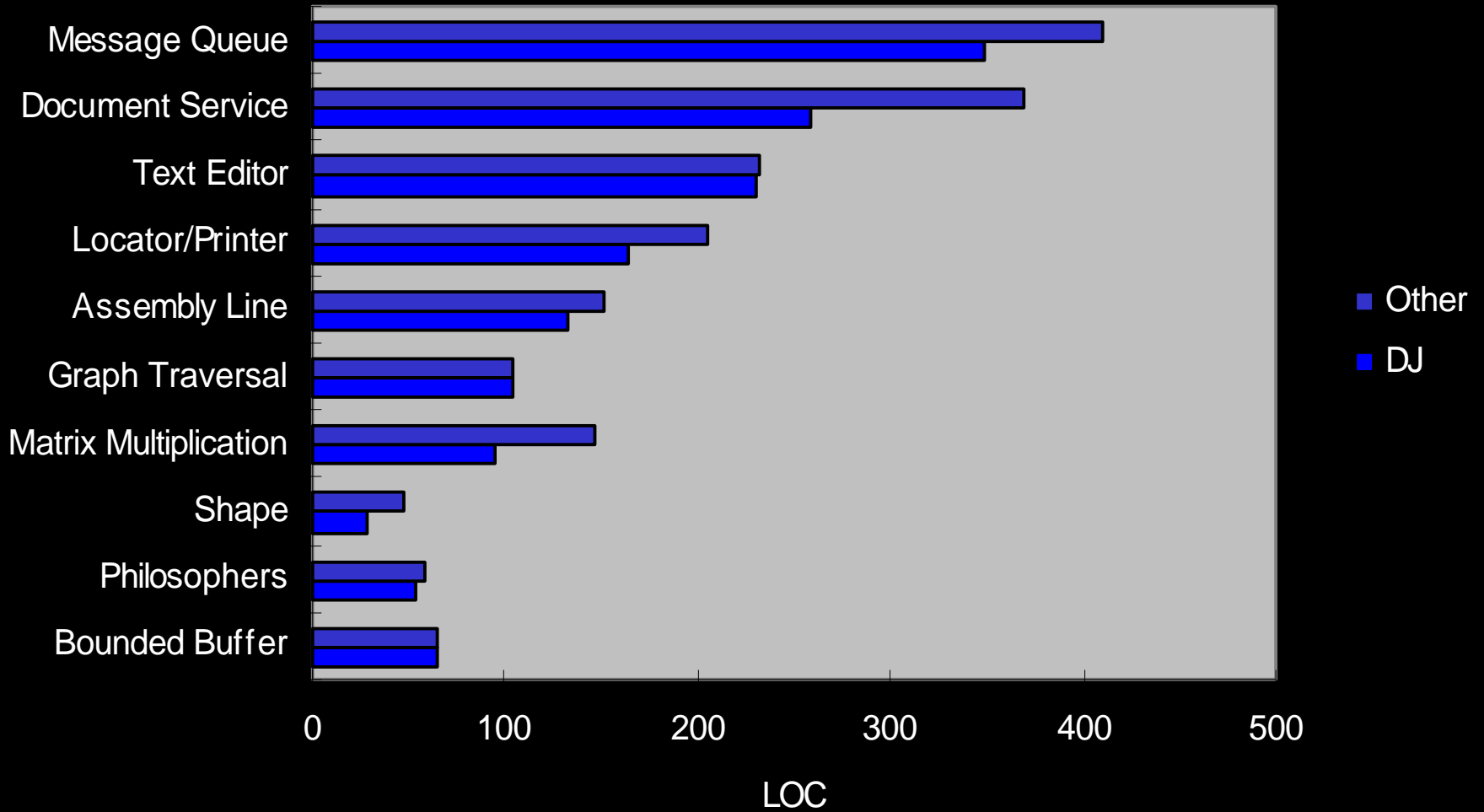
    public synchronized void put(Object o) {
        while (usedSlots == array.length) {
            try {
                wait();
            }
            catch (InterruptedException e) {};
        }
        array[putPtr] = o;
        putPtr = (putPtr + 1) % array.length;
    }
    if (usedSlots++ == 0)
        notifyAll();
}

public synchronized Object take() {
    while (usedSlots == 0) {
        try {
            wait();
        }
        catch (InterruptedException e) {};
    }
    Object old = array[takePtr];
    array[takePtr] = null;
    takePtr = (takePtr+1) % array.length;
}
    if (usedSlots-- == array.length)
        notifyAll();
    return old;
}
```

Java

Case-studies

Lines of Code



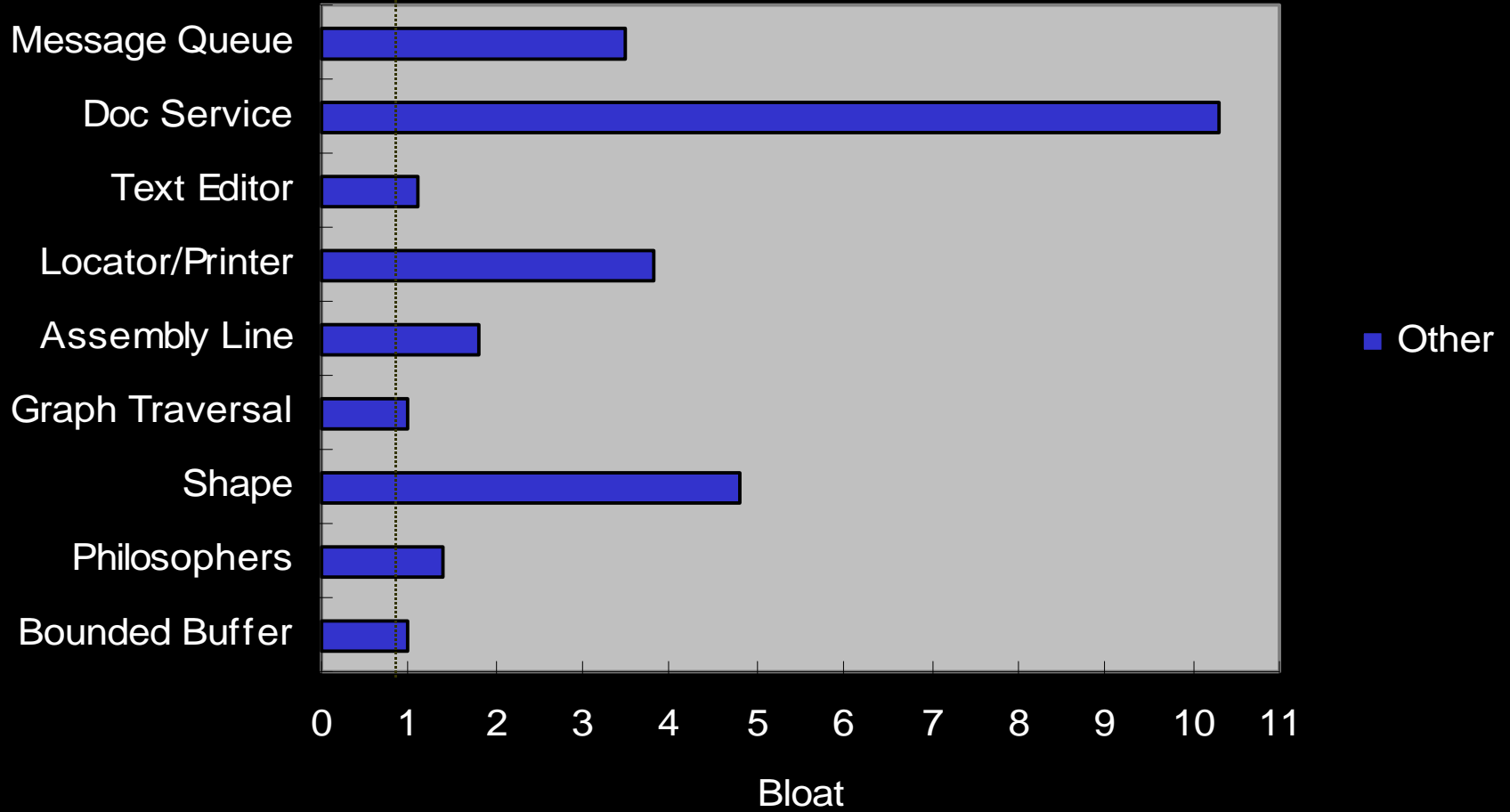
Aspectual Bloat

$$\text{aspectual bloat} = \frac{\text{LOC in Java} - \text{LOC in JCore}}{\text{LOC in Cool+Ridl}}$$

Measures how poorly Java, without D, captures the aspect programs

Case-studies

Aspectual Bloat



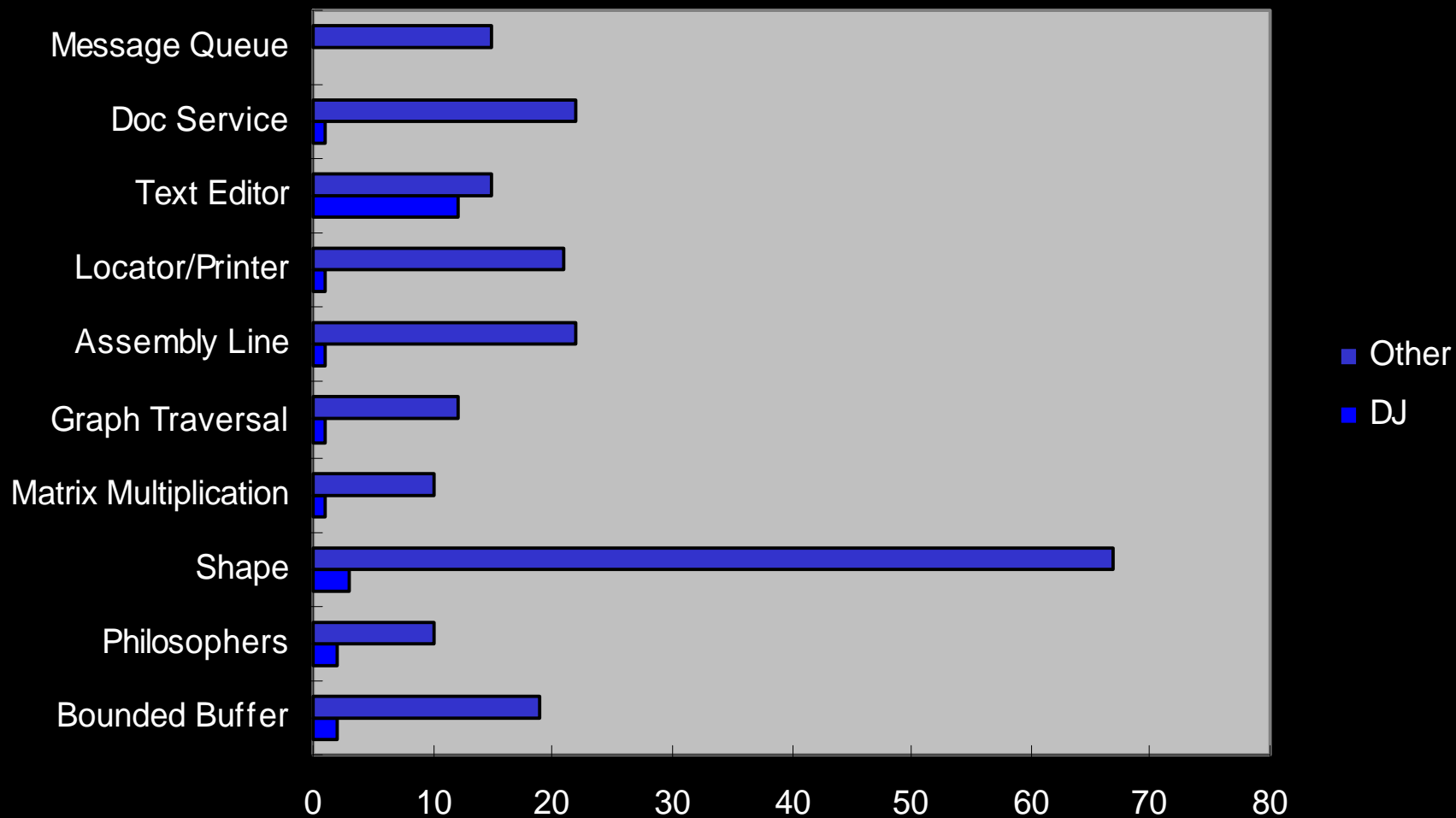
Tangling Ratio

$$\text{tangling} = \frac{\begin{array}{l} \# \text{ of transition points} \\ \text{between aspect code and functionality code} \end{array}}{\text{LOC}}$$

Measures intermingling, dispersion

Case-studies

Tangling (%)



Observations

- D effectively separates aspect code from classes and localizes it in coordinators and portals.
- Aspect programs in D, in many cases, are shorter; never more lengthy.
- DJ versions, in many cases, are smaller; never bigger.

Performance

COOL

<i>1000 method invocations per thread</i>	
DJ	Java
<i>Single thread calling a selfex method:</i> 28ms	<i>Single thread calling a synchronized method:</i> 7ms
<i>Two threads calling the same selfex method:</i> 90ms	<i>Two threads calling the same synchronized method:</i> 30ms
<i>Two threads calling 2 methods with requires, on_exit:</i> 13s	<i>Two threads, calling 2 methods with wait, notification:</i> 12s

Performance

RIDL

<i>1000 method invocations</i>	
DJ	Java
<i>no parameters:</i> 10s	<i>no parameters:</i> 10s
<i>one gref parameter:</i> 24s	<i>one parameter of type Remote:</i> 24s
<i>one copy parameter (object with 4 Integer fields):</i> 26s	<i>one parameter Serializable (object with 4 Integer fields):</i> 30s
<i>copying directive that selects 3 out of 4 Integer fields of a parameter:</i> 35s	<i>parameter with 3 Integer fields that is partially copied from an object of another class:</i> 28s

Observations

- DJ's performance is within Java's performance

Alpha-Usage

- Four programmers wrote two medium-size applications: space war, distributed library
- Learning, designing, programming:
2 months
 - A different AW implemented at PARC by Mendhekar, Loingtier, Lamping, Kiczales
 - Experiment conducted by Murphy

Applications

- Distributed Space War
 - 1500 LOC, 19 classes, 2 coordinators, 4 portals
- Distributed Library
 - 1200 LOC, 13 classes, 3 coordinators, 4 portals

Observations

- Users found COOL and RIDL easy to use
- No difficulty in understanding effect of aspect code on components
- Aspect languages eased burden of programming some distribution issues (E.g. using RMI)
- Cannot expect aspect modules to capture intent

Outline

- Overview
- D: Design
- D: Implementation
- Validation Results
- Conclusion

Contributions

- Support for programming thread synchronization and remote data transfers separately from the implementation of the components
- Enforcement of separation
- Systematic and simple division of labor
- Basis for better documentation

- Implementation: DJ

Directions in Language Design

- Methodological study of code tangling
- New kinds of interfaces between modules not of the type client/provider, but useful for structuring programs
- Add-on aspect languages; no modifications or extensions to component language

Future Work

- Improve/extend existing languages
 - replication
 - timeouts
 - relation between aspect modules
 - add more imperative features?
 - error handling
- New aspects, new aspect languages

EOP

Outline

- Overview
- Code Tangling / Aspect Identification
- D: Design
- D: Implementation
- Validation Results
- Conclusion
- Demo

```

interface ShapeI extends Remote {
    double get_x() throws RemoteException ;
    void set_x(int x) throws RemoteException ;
    double get_y() throws RemoteException ;
    void set_y(int y) throws RemoteException ;
    double get_width() throws RemoteException ;
    void set_width(int w) throws RemoteException ;
    double get_height() throws RemoteException ;
    void set_height(int h) throws RemoteException ;
    void adjustLocation() throws RemoteException ;
    void adjustDimensions() throws RemoteException ;
}
public class Shape
    implements ShapeI {
    protected AdjustableLocation loc;
    protected AdjustableDimension dim;
    public Shape() {
        loc = new AdjustableLocation(0, 0);
        dim = new AdjustableDimension(0, 0);
    }
    double get_x() throws RemoteException {
        return loc.x(); }
    void set_x(int x) throws RemoteException {
        loc.set_x(x); }
    double get_y() throws RemoteException {
        return loc.y(); }
    void set_y(int y) throws RemoteException {
        loc.set_y(y); }
    double get_width() throws RemoteException {
        return dim.width(); }
    void set_width(int w) throws RemoteException {
        dim.set_w(w); }
    double get_height() throws RemoteException {
        return dim.height(); }
    void set_height(int h) throws RemoteException {
        dim.set_h(h); }
    void adjustLocation() throws RemoteException {
        loc.adjust();
    }
    void adjustDimensions() throws RemoteException {
        dim.adjust();
    }
}
class AdjustableLocation {
    protected double x_, y_;
    public AdjustableLocation(double x, double y) {
        x_ = x; y_ = y;
    }
    synchronized double get_x() { return x_; }
    synchronized void set_x(int x) {x_ = x;}
    synchronized double get_y() { return y_; }
    synchronized void set_y(int y) {y_ = y;}
    synchronized void adjust() {
        x_ = longCalculation1();
        y_ = longCalculation2();
    }
}
class AdjustableDimension {
    protected double width_=0.0, height_=0.0;
    public AdjustableDimension(double h, double w) {
        height_ = h; width_ = w;
    }
    synchronized double get_width() { return width_; }
    synchronized void set_w(int w) {width_ = w;}
    synchronized double get_height() { return height_; }
    synchronized void set_h(int h) {height_ = h;}
    synchronized void adjust() {
        width_ = longCalculation3();
        height_ = longCalculation4();
    }
}

```

Questions:

What exactly are these?

Why are they here?

How can we remove them?

Make them more localized?

How programs become tangled: an example

Book locator service specifications:

- **register** book b in location l
- **unregister** book b
- **locate** book given title
- concurrent accesses
- network service

*How programs become tangled:
an example*

Implementing the functionality:

```
public class BookLocator {
    private Book    books[];
    private Location locations[];
    private int     nbooks = 0;
    // the constructor
    public BookLocator (int dbsize) {
        books = new Book[dbsize];
        locations = new Location[dbsize];
    }
    public void register (Book b, Location l)
    throws LocatorFull {
        if (nbooks > books.length) throw new LocatorFull();
        else { // Just put it at the end
            books[nbooks] = b;
            locations[nbooks++] = l;
        }
    }
    public void unregister (Book b) {
        // find the book and take it out of books[]
        --nbooks;
    }
    public Location locate (String title)
    throws BookNotFound {
        Book abook = books[0];
        int i = 0; boolean found = false;
        while (i < nbooks && found == false) {
            if (abook.get_title().compareTo(str) == 0 )
                found = true;
            else abook = books[++i];
        }
        if (found == false) throw new BookNotFound (str);
        return locations[i];
    }
}
```

```
public class Book {
    public String title, author;
    public int isbn;
    Project owner;
    Postscript ps;

    public Book (String t, String a,
                 int n) {
        title = t; author = a; isbn = n;
    }
    // other methods...
}
```

```
public class Location {
    public int building, room;
    public Location (int bn, int rn) {
        building = bn;
        room = rn;
    }
    // other methods...
}
```

*How programs become tangled:
an example*

Book locator service implementation:

- ✓ **register** book b in location l
- ✓ **unregister** book b
- ✓ **locate** book given title
 - concurrent accesses
 - network service

*How programs become tangled:
an example*

Synchronizing concurrent accesses:

register, unregister (writers)
disable all

locate (reader):
disable register, unregister

```
public class BookLocator {
    private Book    books[];
    private Location locations[];
    private int     nbooks = 0;
    // the constructor
    public BookLocator (int dbsize) {
        books = new Book[dbsize];
        locations = new Location[dbsize];
    }
    public void register (Book b, Location l)
    throws LocatorFull {
        if (nbooks > books.length) throw new LocatorFull();
        else { // Just put it at the end
            books[nbooks] = b;
            locations[nbooks++] = l;
        }
    }
    public void unregister (Book b) {
        // find the book and take it out of books[]
        --nbooks;
    }
    public Location locate (String title)
    throws BookNotFound {
        Book abook = books[0];
        int i = 0; boolean found = false;
        while (i < nbooks && found == false) {
            if (abook.get_title().compareTo(str) == 0 )
                found = true;
            else abook = books[++i];
        }
        if (found == false) throw new BookNotFound (str);
        return locations[i];
    }
}
```

```

public class BookLocator {
    private Book    books[];
    private Location locations[];
    private int     nbooks = 0;
    protected int  activeReaders = 0, activeWriters = 0;
    // the constructor
    public BookLocator (int dbsize) {
        books = new Book[dbsize]; locations = new Location[dbsize];
    }
    public void register (Book b, Location l) throws LocatorFull {
        synchronized (this) {
            while (activeReaders > 0 || activeWriters > 0)
                try { wait(); } catch (InterruptedException e) {}
            ++activeWriters;
        }
        if (nbooks > books.length) throw new LocatorFull();
        else { // Just put it at the end
            books[nbooks] = b; locations[nbooks++] = l;
        }
        synchronized (this) {--activeWriters; notifyAll();}
    }
    // similar for unregister

    public Location locate (String title) throws BookNotFound {
        Location l;
        synchronized (this) {
            while (activeWriters > 0)
                try { wait(); } catch (InterruptedException e) {}
            ++activeReaders;
        }
        Book abook = books[0]; int i = 0; boolean found = false;
        while (i < nbooks && found == false) {
            if (abook.get_title().compareTo(str) == 0 ) found = true;
            else abook = books[++i];
        }
        if (found == false) {
            synchronized (this) {--activeReaders; notifyAll();}
            throw new BookNotFound (str);}
        l = locations[i];
        synchronized (this) {--activeReaders; notifyAll();}
        return l;
    }
}

```

*How programs become tangled:
an example*

**Synchronizing
concurrent accesses:**

*How programs become tangled:
an example*

Book locator service implementation:

- ✓ **register** book b in location l
- ✓ **unregister** book b
- ✓ **locate** book given title
- ✓ **concurrent accesses**
 - network service

```

public interface Locator extends Remote {
    void register(String title, int isbn, Location l)
        throws RemoteException;
    void unregister(String t) throws RemoteException;
    Location locate(String title) throws RemoteException;
}

public class BookLocator implements Locator
    extends UnicastRemoteObject{

    private Book    books[];
    private Location locations[];
    private int     nbooks = 0;
    protected int  activeReaders = 0, activeWriters = 0;
    public BookLocator (int dbsize) {
        books = new Book[dbsize]; locations = new Location[dbsize];
    }
    public void register(String title, int isbn,
        Location l)
        throws LocatorFull, RemoteException {
        synchronized (this) {
            while (activeReaders > 0 || activeWriters > 0)
                try { wait(); } catch (InterruptedException e) {}
            ++activeWriters;
        }
        if (nbooks > books.length) throw new LocatorFull();
        else { // Just put it at the end
            books[nbooks] = b; locations[nbooks++] = l;
        }
        synchronized (this) {--activeWriters; notifyAll();}
    }
    public void unregister(String title)
        throws RemoteException {
        /* ... */
    }
    public Location locate (String title)
        throws BookNotFound, RemoteException {
        /* ... */
    }
}

```

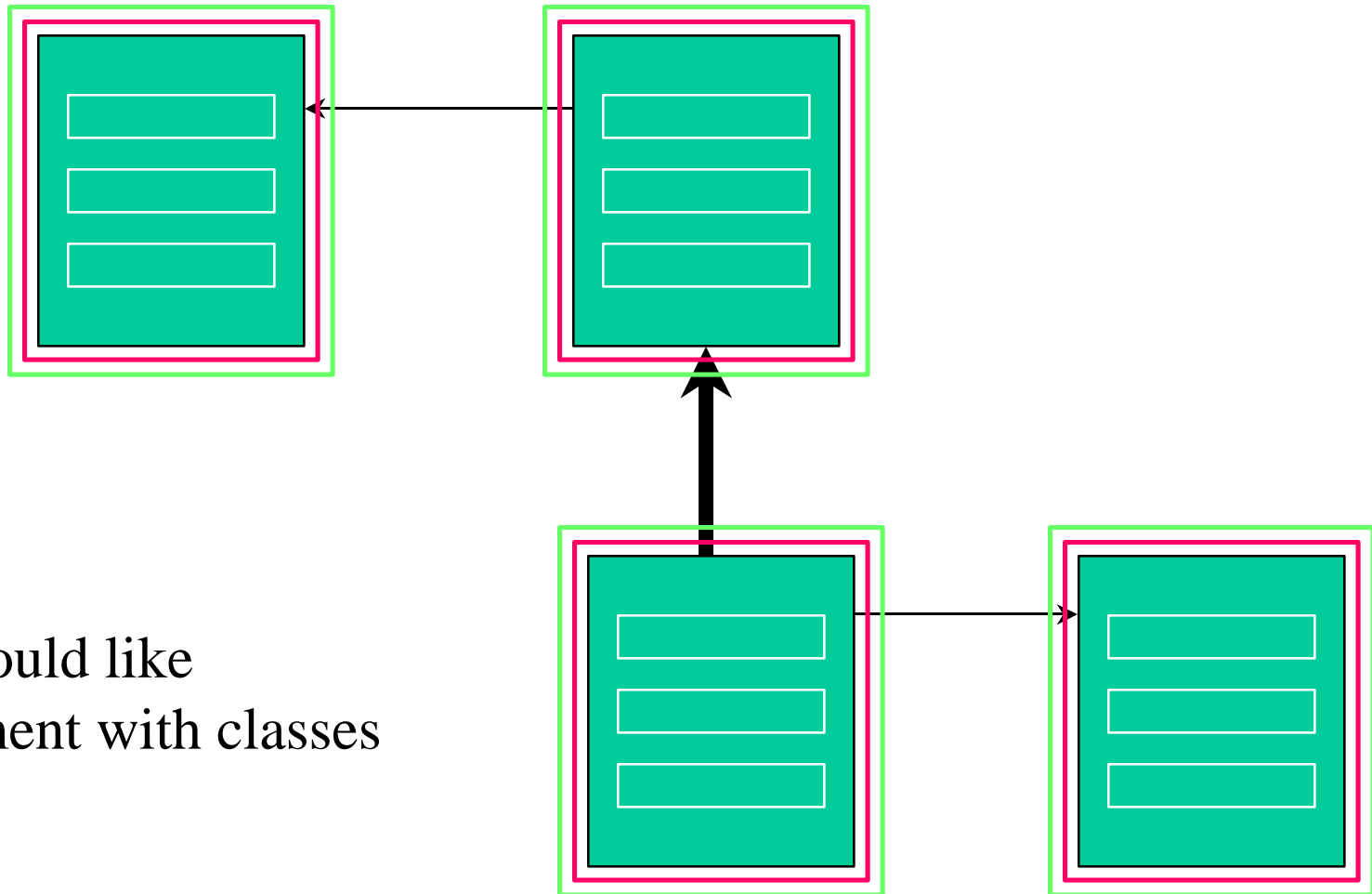
*How programs become tangled:
an example*

Providing for
network access and
remote data transfers:

Two Issues

- Synchronization of threads
- Remote access and data transfers

The source of tangling



We would like
alignment with classes

Code tangling is bad

- Harms program structure
- Distracts from main functionality

- Hard to program, error-prone
- Code difficult to understand, maintain

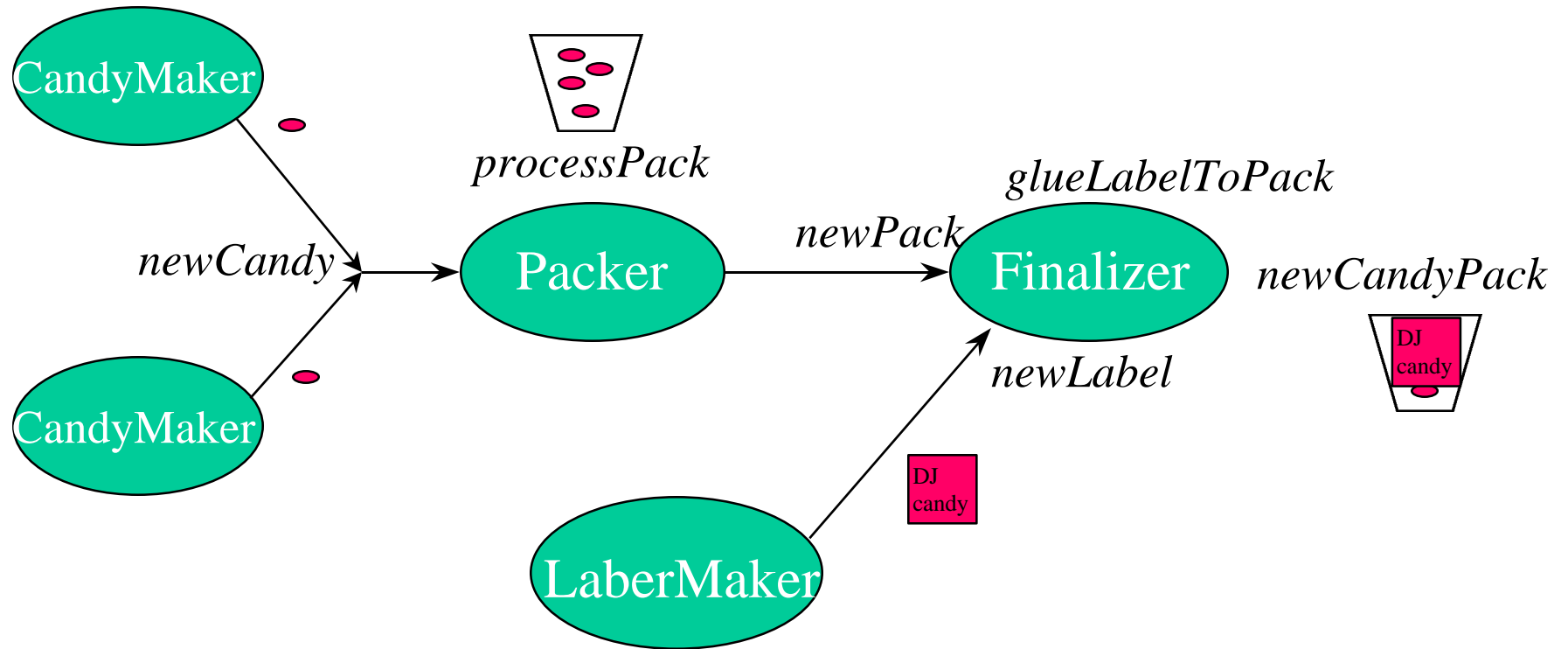
Ways to decrease the tangling

- Style guidelines
- Coding rules
- Design patterns
- Better programming languages

Ways to decrease the tangling: better programming languages

orthogonal approach seems to be promising
some languages have tried this approach before
D builds on top of all that previous work

COOL Assembly Line



COOL Assembly Line

```
coordinator Packer, Finalizer {
  selfex Packer.newCandy:
  cond packDone = false, packFull = false;
  cond gotPack = false, gotLabel = false;

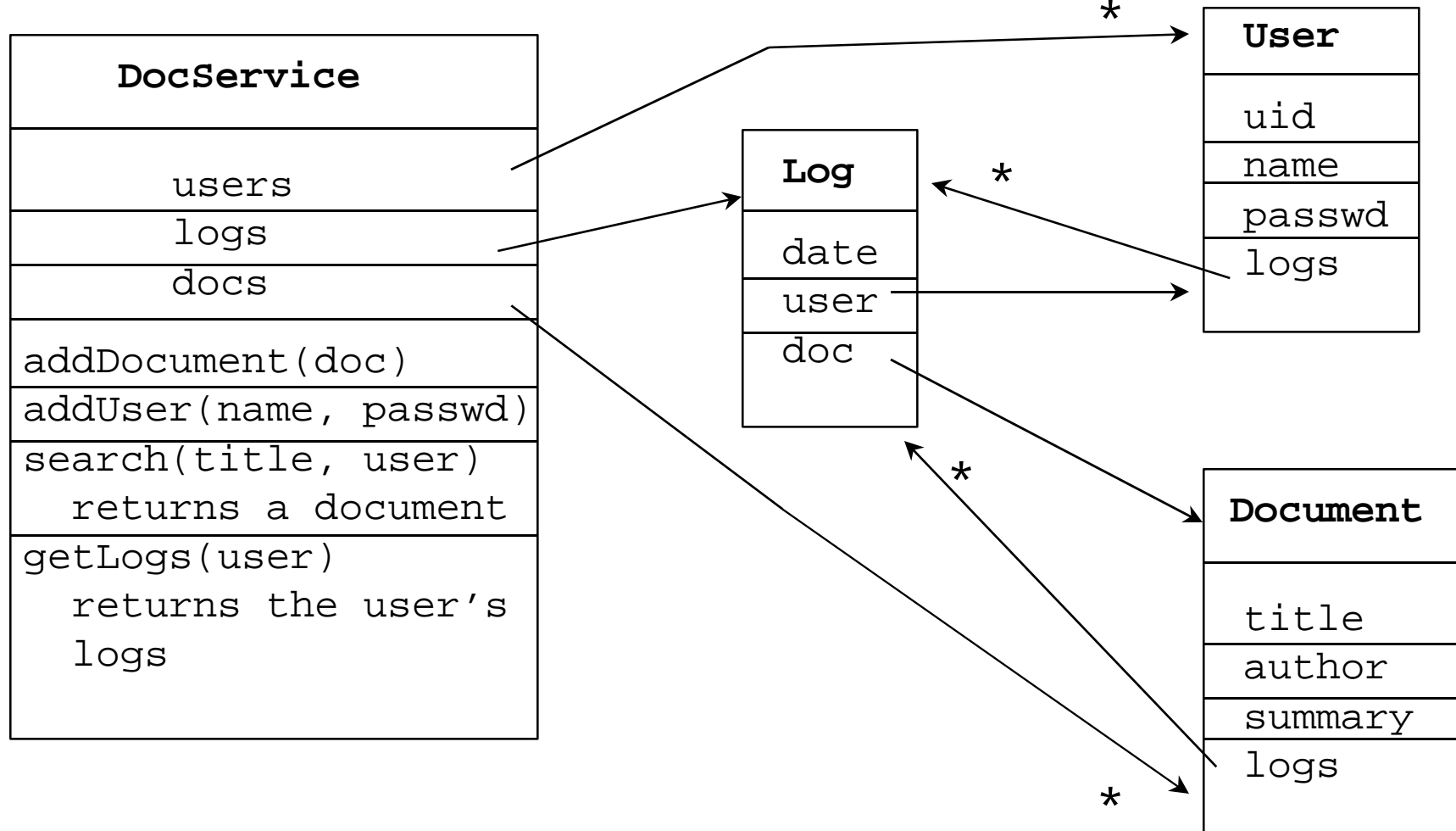
  Packer.newPack: on_exit{packDone = true;}
  Packer.newCandy: requires !packFull &&
                    packDone;

    on_exit {
      if (nCandy == nCandyPerPack)
        packFull = true;
    }
  Packer.processPack: requires packFull;
  Finalizer.newPack: requires !gotPack;
    on_exit {
      gotPack = true;
      packFull = false; packDone = false;
    }
  Finalizer.newLabel: requires !gotLabel;
    on_exit { gotLabel = true; }
  Finalizer.glueLabelToPack:
    requires gotPack && gotLabel;
  Finalizer.newCandyPack:
    on_exit {
      gotPack = false; gotLabel = false;
    }
}
```

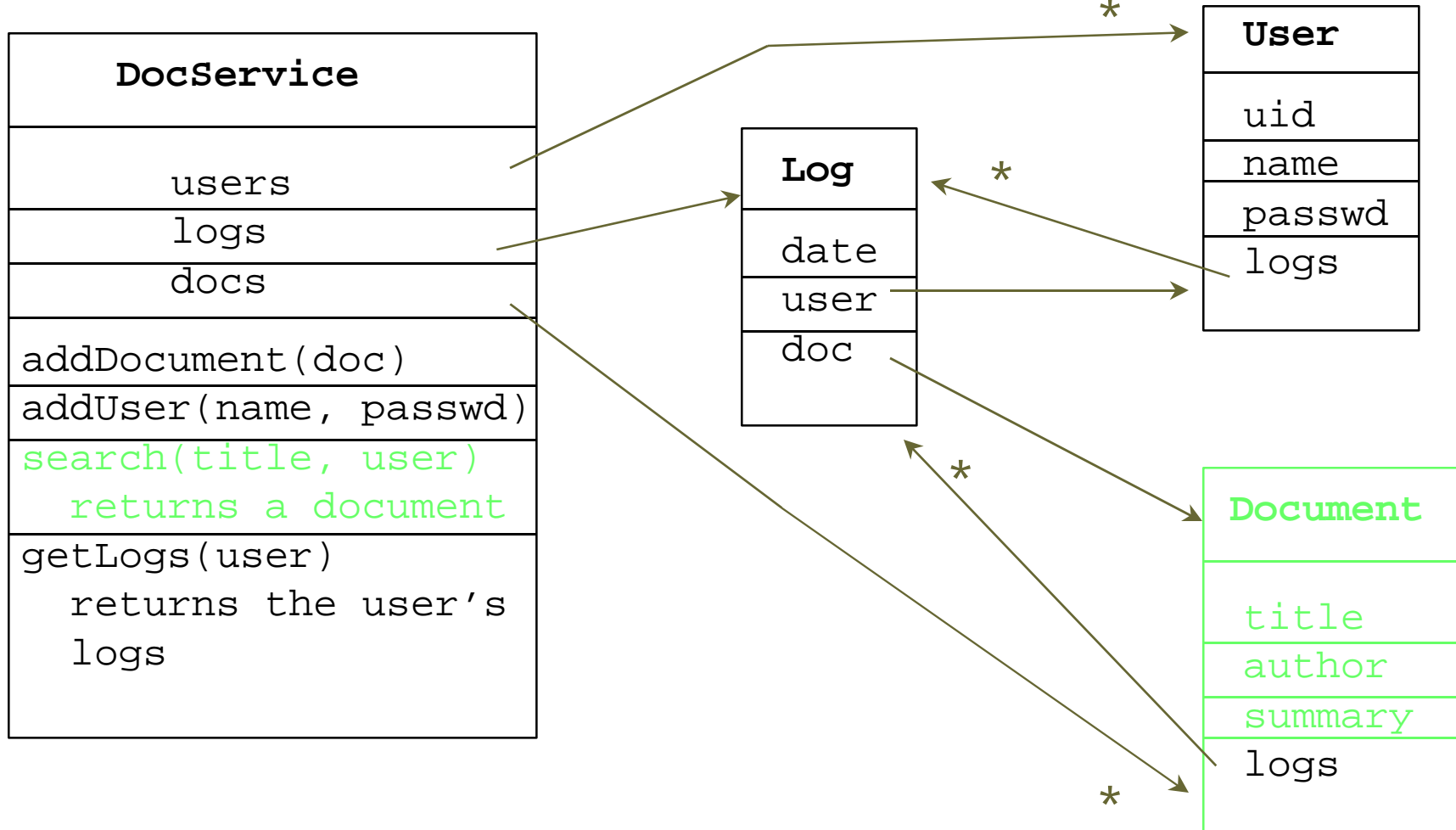
COOL Syntax

```
cooldef : [ perclass | perobject ] coord class_list {  
    autoex method_list;  
    mutex{method_list}; *  
    type var [ = value]; *  
    cond [ perobject | perclass ] condvar = true | false ; *  
    method_list:  
        requires ( boolean_expr ) [orwait t]  
        on_entry { cool_stmt * } *  
        on_exit { cool_stmt * }  
};  
cool_stmt : condvar = true | false ; |  
    var = value; |  
    if (boolean_expr) cool_stmt  
    [else cool_stmt ]
```

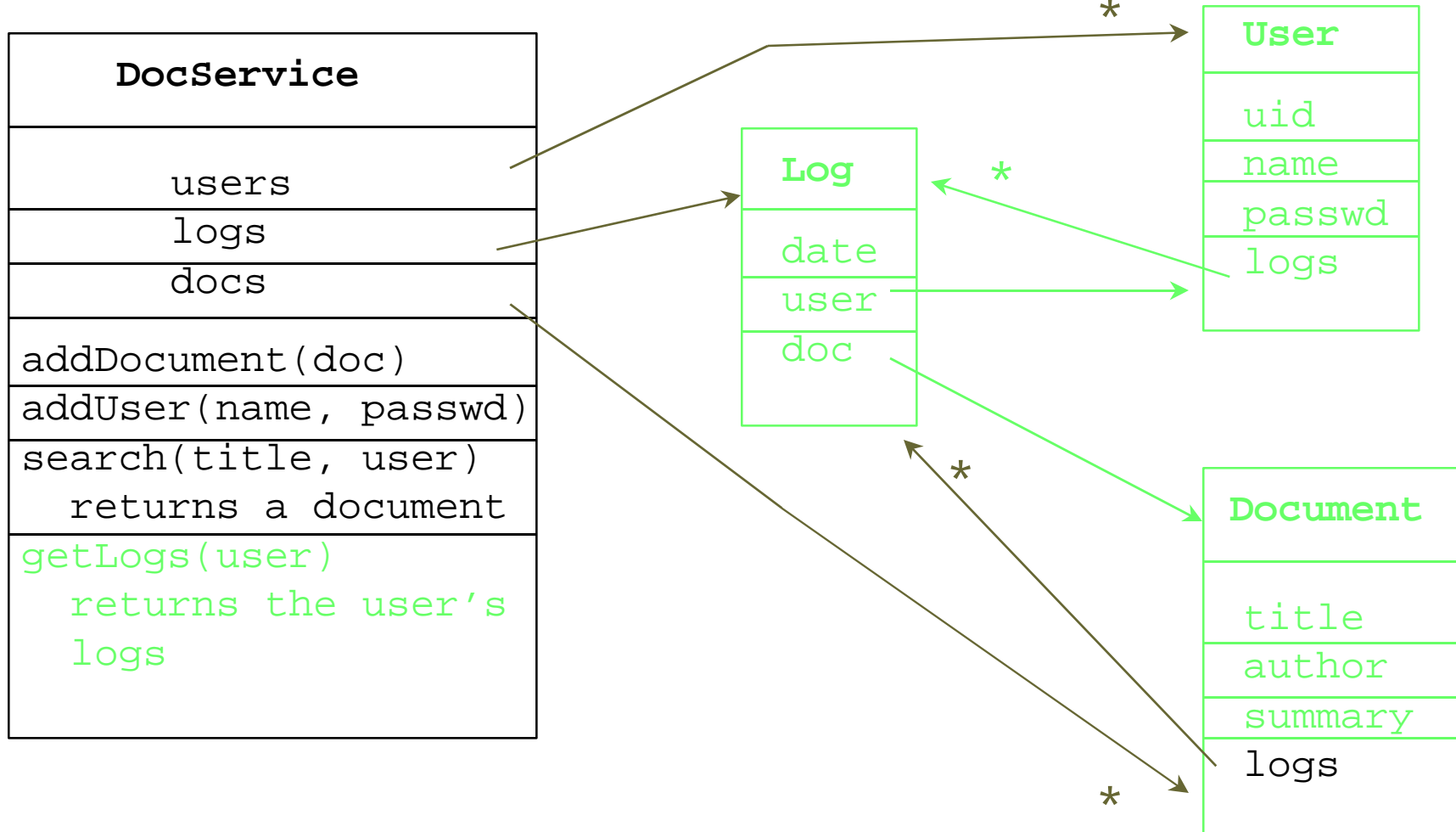
RIDL Document Service



RIDL Document Service



RIDL Document Service



RIDL Document Service

```
portal DocService {
  boolean addDocument(Document doc);
  Integer addUser(String name
                  Integer passwd);
  Document search(String title,
                  Integer uid,
                  Integer passwd){
    return: copy {
      Document bypass logs;
    }
  };
  DVector getUserLogs(Integer uid,
                      Integer passwd){
    return: copy {
      Document bypass logs;
      User bypass logs, passwd;
    }
  };
}
```

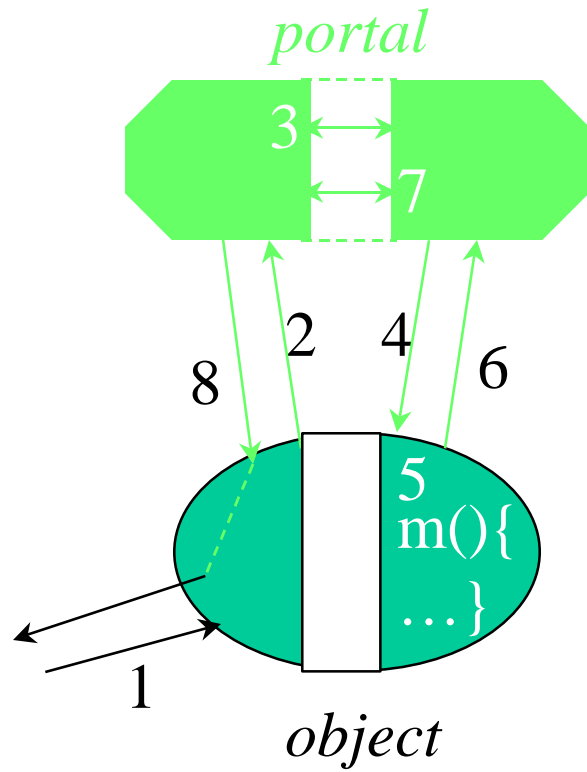
COOL Design

- provider (i.e. the class) defines the synchronization (monitor approach)
- smallest unit of synchronization is the method
- no middle ground between one instance and all instances of classes
- coordination is contained within one coordinator
- association between an object and its coordinator is static

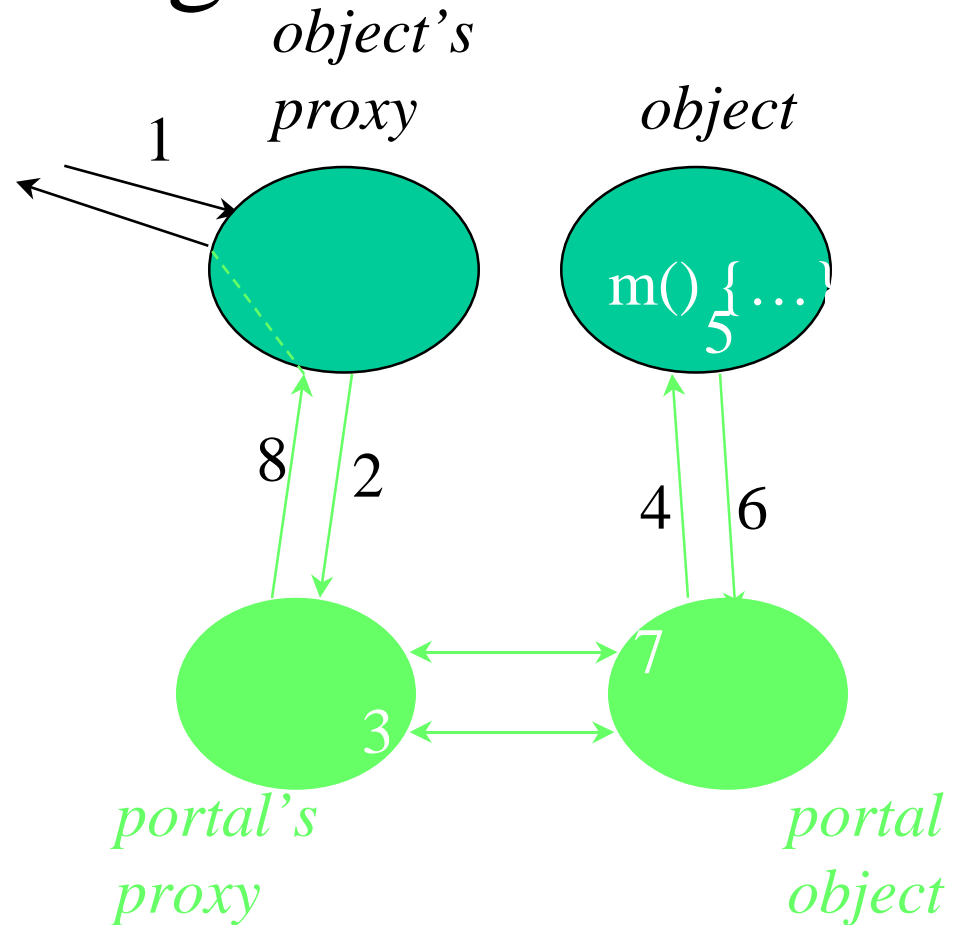
RIDL Design

- provider (i.e. the class) defines the remote interaction
- smallest unit for remote interaction is the method
- parameter passing semantics ...
- remote interaction is contained within one portal
- association between an object and its portal is static
- no multi-class portals

Implementing RIDL



Semantics



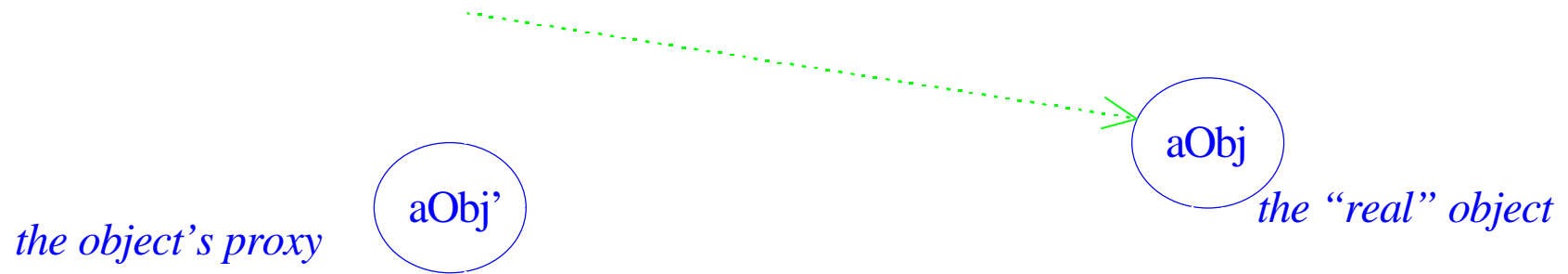
Implementation

RIDL

```
class BookLocator {  
    private Book    books[];  
    private Location locations[];  
    public void register(Book b, Location l){  
        // Verify and add book b to database  
    }  
    public Location locate (String title) {  
        Location loc;  
        // Locate book and get its location  
        return loc;  
    }  
}
```

```
portal BookLocator {  
    void register (Book book,  
                  Location l);  
    Location locate (String title)  
    default:  
        Book: copy{Book only title,  
                  author,  
                  isbn;}  
}
```

RIDL Protocol



D's Remote Objects

```
class BookLocator {
    BookLocatorP _p;           // portal object
    BookLocatorPP _pp = null; // portal proxy
    BookLocator(BookLocatorPP proxy) { _pp = proxy; }
    BookLocator(...) { _p = new BookLocatorP(this); }

    protected void _d_register(Book b, Location l) {
        original implementation of f
    }

    void register(BookLocator b, Location l) {
        if (_pp != null) // this is a proxy
            _pp.register(b, l);
        else // this is a real object
            _d_register(b, l);
    }
    // similar for locate
}
```

D's Remote Objects

```
class BookLocator {
    BookLocatorP _p;           // portal object
    BookLocatorPP _pp = null; // portal proxy
    BookLocator(BookLocatorPP proxy) { _pp = proxy; }
    BookLocator(...) { _p = new BookLocatorP(this); }

    protected void _d_register(Book b, Location l) {
        original implementation of f
    }


    void register(BookLocator b, Location l) {
        if (_pp != null) // this is a proxy
            _pp.register(b, l);
        else // this is a real object
            _d_register(b, l);
    }
    // similar for locate
}
```

D's Remote Objects

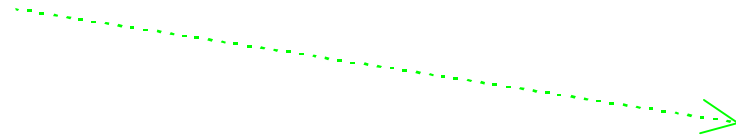
```
class BookLocator {
    BookLocatorP _p;           // portal object
    BookLocatorPP _pp = null; // portal proxy
    BookLocator(BookLocatorPP proxy) { _pp = proxy; }
    BookLocator(...) { _p = new BookLocatorP(this); }

    protected void _d_register(Book b, Location l) {
        original implementation of f
    }

    void register(BookLocator b, Location l) {
        if (_pp != null) // this is a proxy
            _pp.register(b, l);
        else // this is a real object
            _d_register(b, l);
    }
    // similar for locate
}
```



RIDL Protocol



Portal objects

```
class BookLocatorPP {
    BookLocatorPRI rself;
    BookLocatorPP(BookLocatorPRI o){ rself = o; }
    void register(Book b, Location l) {
        Dargument a1, a2;
        a1 = new Dargument(b, BookLocatorTraversals.t1);
        a2 = new Dargument(l, null);
        rself.register(a1, a2); // redirect
    }
    // similar for locate
}
```

```
class BookLocatorP implements BookLocatorPRI {
    BookLocator myself;
    BookLocatorP(BookLocator o) { myself = o; }
    void register(Dargument a1, Dargument a2) {
        myself.register(a1.obj, a2.obj);
    }
}
```

Portal objects

```
class BookLocatorPP {  
    BookLocatorPRI rself;  
    BookLocatorPP(BookLocatorPRI o){ rself = o; }  
    void register(Book b, Location l) {  
        Dargument a1, a2;  
        a1 = new Dargument(b, BookLocatorTraversals.t1);  
        a2 = new Dargument(l, null);  
        rself.register(a1, a2); // redirect  
    }  
    // similar for locate  
}
```

```
class BookLocatorP implements BookLocatorPRI {  
    BookLocator myself;  
    BookLocatorP(BookLocator o) { myself = o; }  
    void register(Dargument a1, Dargument a2) {  
        myself.register(a1.obj, a2.obj);  
    }  
}
```

Portal objects

```
class BookLocatorPP {
    BookLocatorPRI rself;
    BookLocatorPP(BookLocatorPRI o){ rself = o; }
    void register(Book b, Location l) {
        Dargument a1, a2;
        a1 = new Dargument(b, BookLocatorTraversals.t1);
        a2 = new Dargument(l, null);
        rself.register(a1, a2); // redirect
    }
    // similar for locate
}
```

```
class BookLocatorP implements BookLocatorPRI {
    BookLocator myself;
    BookLocatorP(BookLocator o) { myself = o; }
    void register(Dargument a1, Dargument a2) {
        myself.register(a1.obj, a2.obj);
    }
}
```

RIDL

APPLICATION
LAYER

Space 1
(client of aObj)

Space 2

real reference

virtual reference

the object's proxy



the "real" object

R
I
D
L

the portal proxy



the portal object

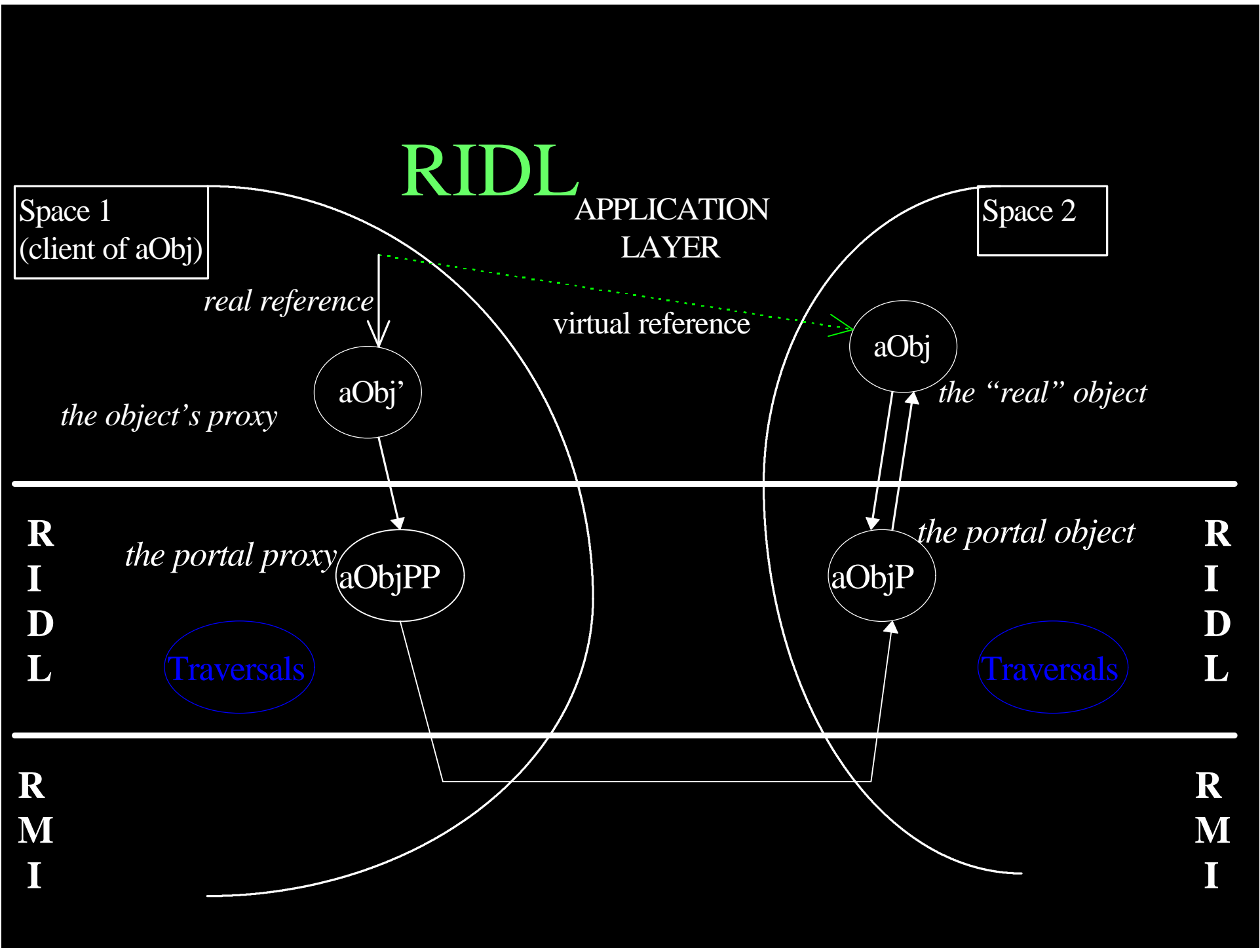


R
I
D
L



R
M
I

R
M
I



Traversal objects

```
class BookLocatorTraversals {
    public static Traversal t1;
    static boolean once = false;
    public static synchronized void init() {
        IncompleteClass c;
        if (once) return;

        t1 = new Traversal("t1", "BookLocatorTraversals");
        c = new IncompleteClass("Book");
        c.bypass("firstPage");
        c.bypass("ps");
        t1.incompleteClass(c);
    }
}
```

```
portal BookLocator {
    void register (Book book,
                  Location l);
    Location locate (String title)
    default:
        Book: copy{Book only title,
                  author,
                  isbn;}
}
```