Role-Based Exploration of Object-Oriented Programs

Brian Demsky, Martin Rinard

MIT

Presented by Xiaoxia Ren

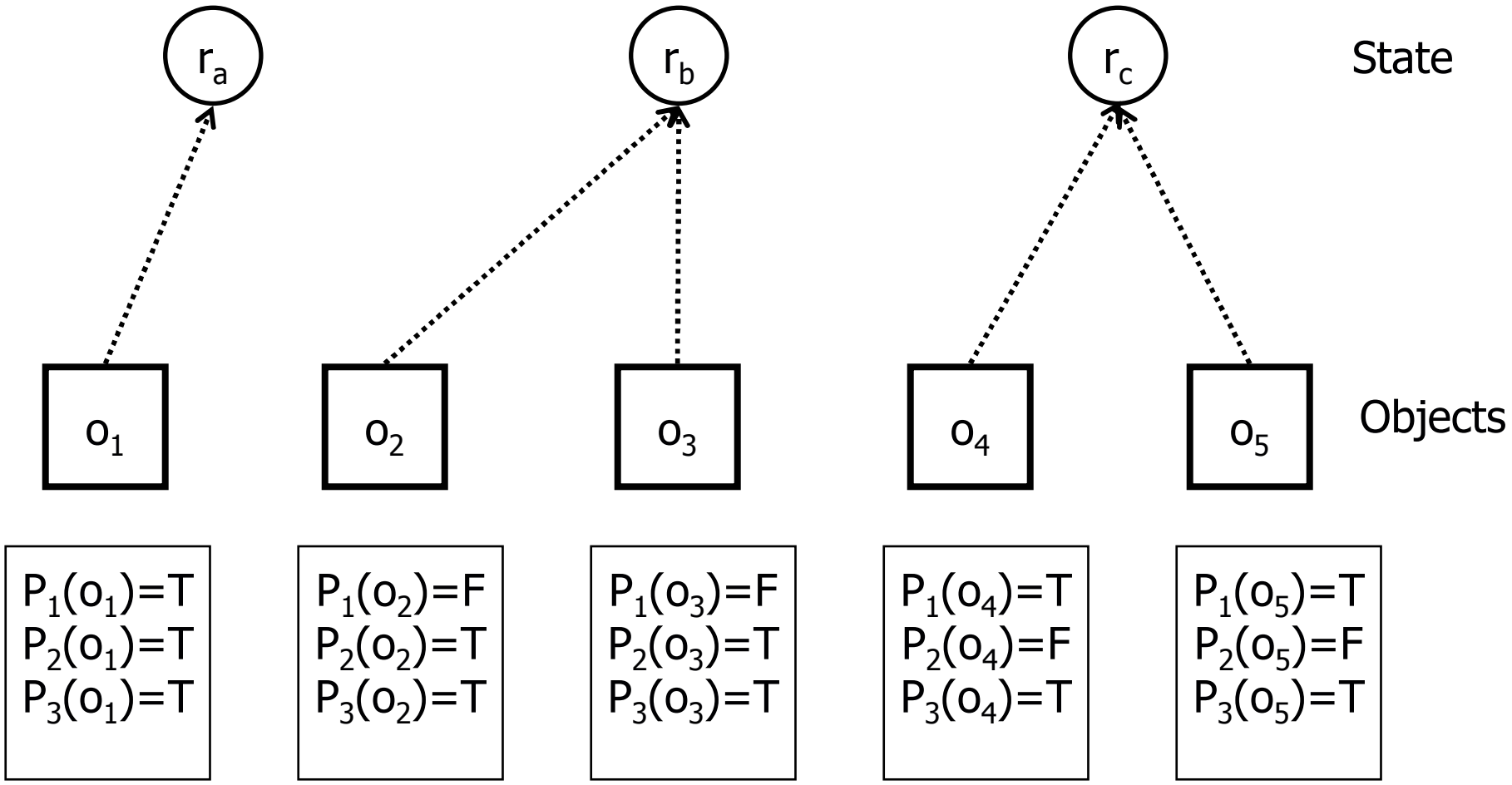
Introduction

- An object's referencing relationships determine important aspects of its purpose in the computation.
- As program runs, each object transits through a sequence of states.
- help developers discover and understand
  - the different states of objects in the computation
  - the referencing relationships between objects in different states, and
  - How states and actions interact

Role Separation Criteria

- How to automatically infer an appropriate set of object states for a given program?
- Define a set of predicates to classify objects into roles
  - Evaluate predicates on each object
  - Objects with the same values for the predicates be considered in the same state
  - Call each state a role

An Example



Copied from Demsky & Rinard's Slides

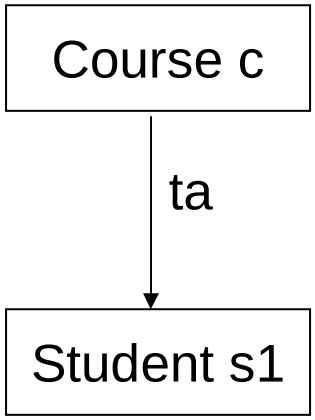
Choosing Predicates

- The key of effectively relating the object states to important properties of a program
- Each predicate should capture some aspect of the object's referencing relationships
- One obvious category of predicates is the predicates that capture the class of the object
  - For each class A, $P_A(o)$ = true if object o is an instance of class A
- How about others?
  - to capture important distinctions between objects of the same class

Heap Alias Predicate

- The functionality of an object often depends on the object that refer to it.
- Separate objects with different kinds of heap aliases into different roles
- Defined for each field of each class
  - $P_{A.f}(o)$ is true if o has a reference from the f field of an instance of class A.

Heap Alias Predicate



Student s2

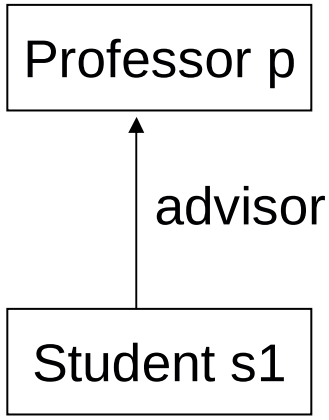
$P_{Course.ta}(s1) = true$

$P_{Course.ta}(s2) = false$

Reference-to Predicate

- The functionality of an object often depends on the objects to which it refers
- Separate objects in different roles if they differ in which fields contain null values
- Defined for each field of a class
  - $P^f(o)$ is true if o has a non-null field of f, false otherwise.

Reference-to Predicate



Student s2

$P^{adviser}(s1) = true$

$P^{adviser}(s2) = false$

Other Role Separation Criteria

- Reachability
  - For key local and global variables v:

    $P_v(o)$ = true if object o is reachable from v
- Identity
  - For each pair of fields f,g:

    $P_{f,g}(o)$ = true if object o has the cyclic path o.f.g=o
- History
  - For key methods m and parameters n:

    $P^{m,n}(o)$= true if object o has been parameter n of method m

Role Subspaces

- Different activities require exploration at varying levels of detail
  - initially need very coarse information then later explore certain aspects in greater detail
  - Find certain details distracting and coarsen aspects of objects orthogonal to the developer's current interest
- Role subspaces provide a way to manage role separation criteria
  - Developers specify a role subspace by specifying a subset of role separation criteria

Role Subspaces -- example

Role Subspace {

Class: Student

Non-null Fields: advisor

}

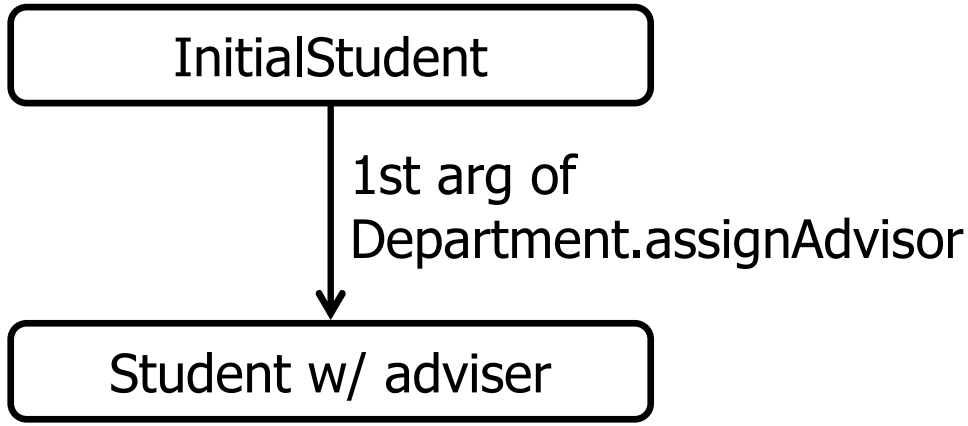
$P_{Student}(o)$

$P^{adviser}(o)$

$P_{Course.ta}(o)$

...

$P^{courses}(o)$



Dynamic Role Inference

- Instrument the program to generate execution traces.
- Uses trace to reconstruct the heap, dynamically compute
  - Roles that each object plays
  - Transitions between roles
  - Roles of methods' parameters
- Present to user for interactive exploration

When to Evaluate Roles

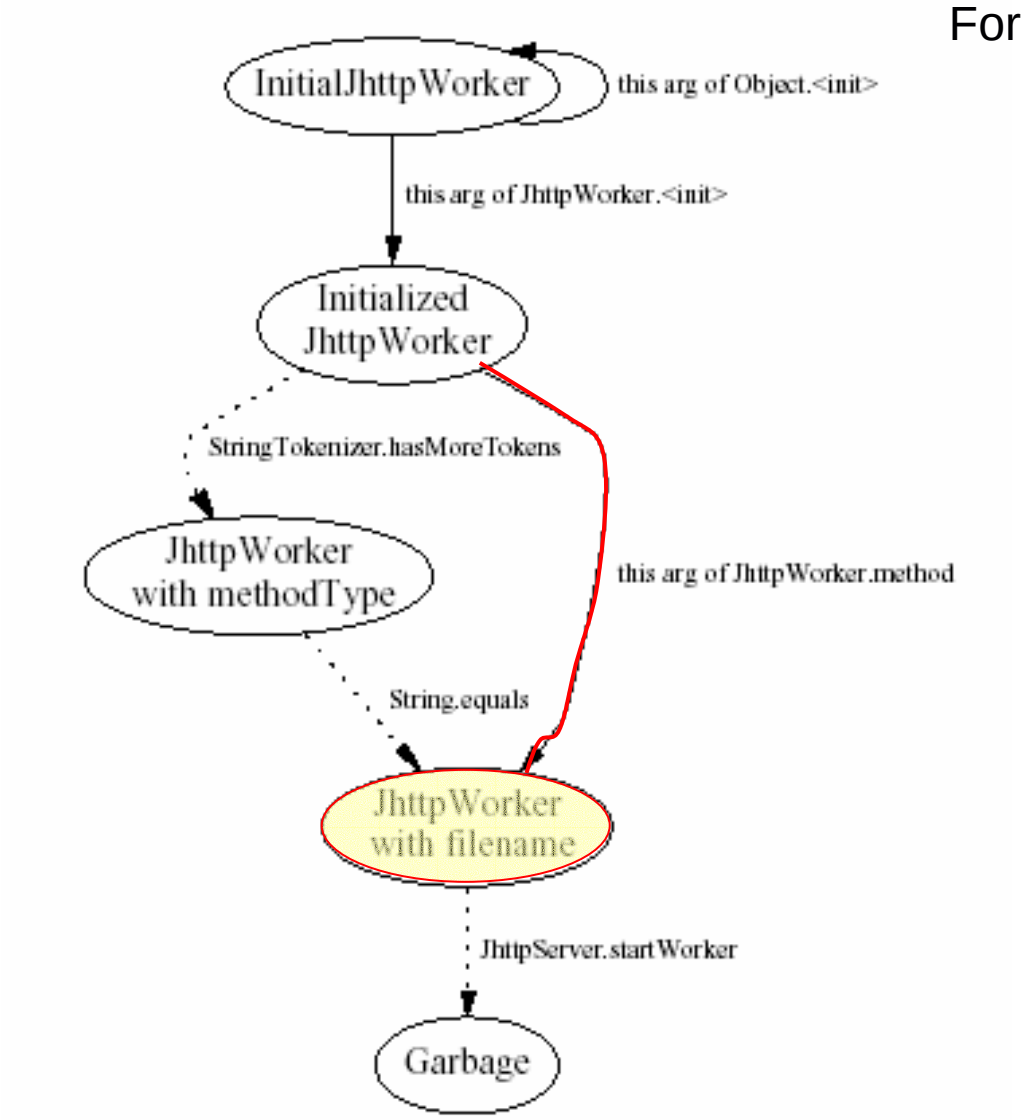
- Evaluates the roles of objects at method boundaries
  - Evaluating the roles of objects after each statement would often observe objects in transient states
  - Objects are likely to have consistent states at method entry and exit points
- The developer can modify this default policy

Presenting the Results

- Uses a graphical web-based interface to support interactive exploration
- The tool presents:
  - Role transition diagrams for each class
  - A role relationship diagram
  - Links from the diagrams to the appropriate
    - role descriptions
    - enhanced method interfaces

Role Transition Diagrams

For class JhttpWorker



Role Definitions

- Role: ***JhttpWorker with filename***
- Class: `JhttpWorker`
- Heap aliases: none
- non-null fields:
  `httpVersion, fileName, methodType, client`
- identity relations: none

Enhanced Method Interfaces

- enhanced method interfaces provide:
  - the roles of the parameters
  - the role changes that the method performs
- this information is useful for understanding
  - assumptions that methods make
  - effects of a method on objects it accesses (read, write or role transition)

Enhanced Method Interfaces

Method: SocketInputStream.<init>(this,plainsocket)

Call Context: {

this: Initial InputStream -> InputStream w/impl,

plainsocket: PlainSocket w/fd ->PlainSocket w/input }

Write Effects:

this.impl=plainsocket

this.temp=NEW

this.fd=plainsocket.fd

Read Effects:

plainsocket

NEW

plainsocket.fd

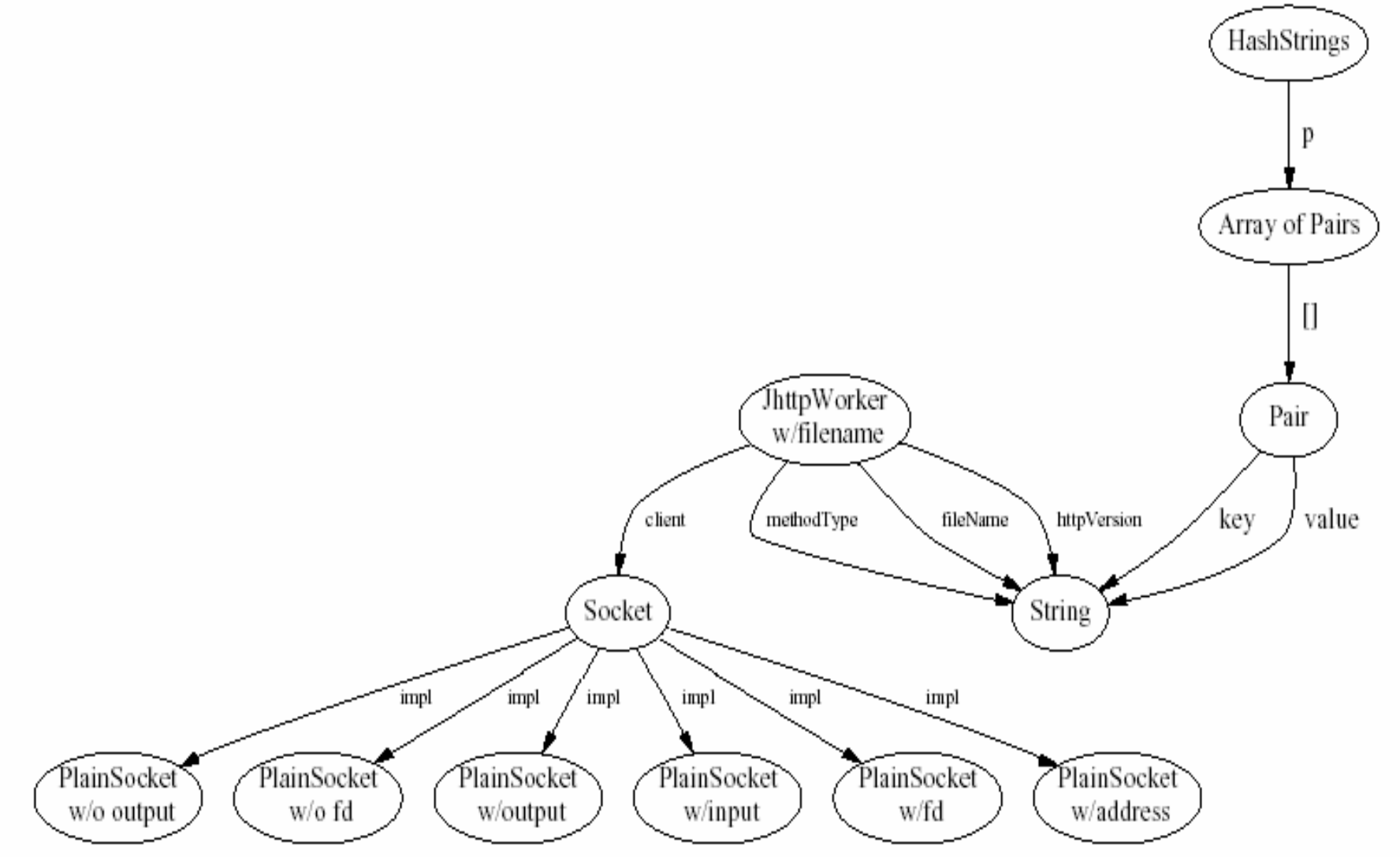
Role Transition Effects:

plainsocket: PlainSocket w/fd -> PlainSocket w/input

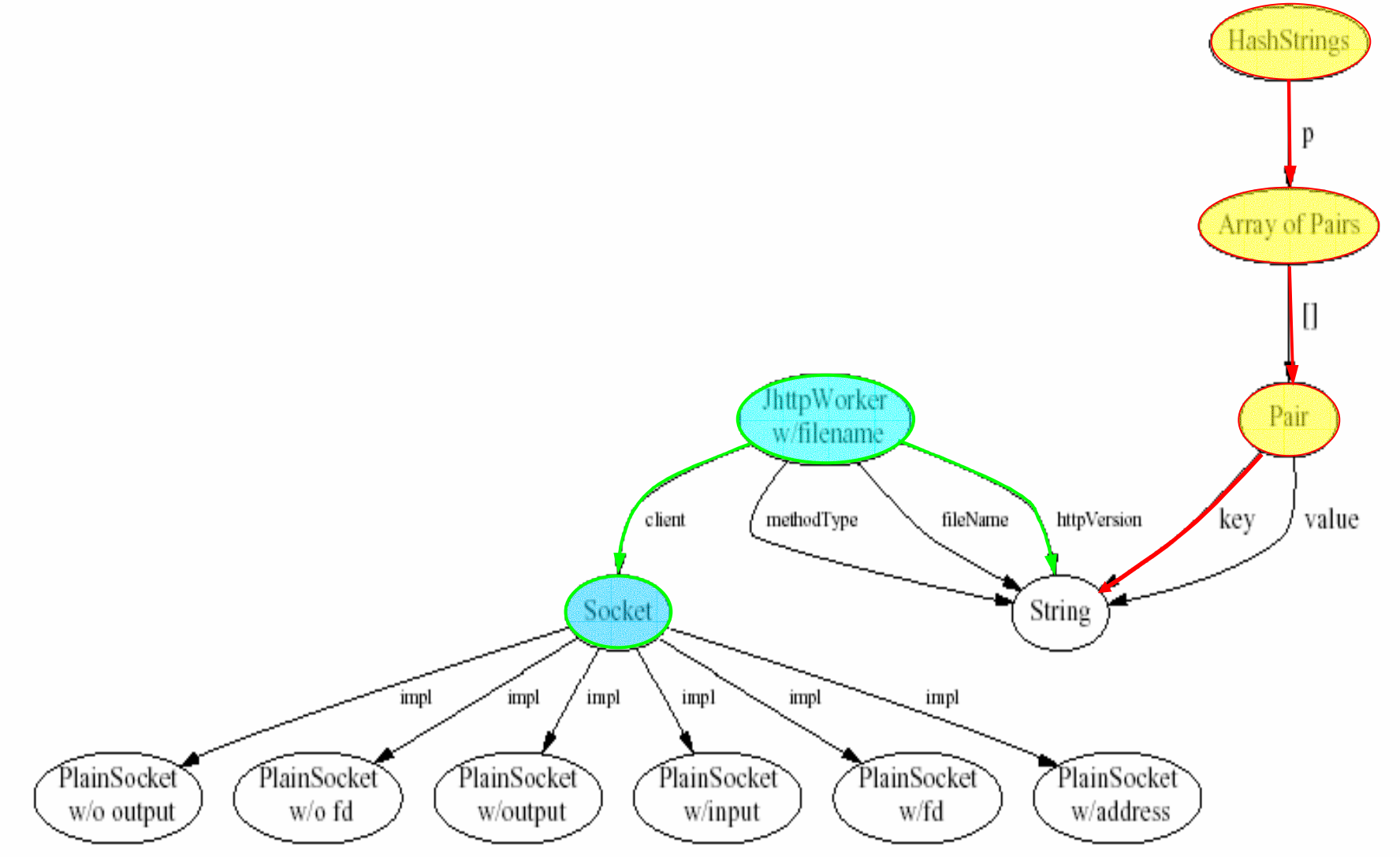
this: Initial InputStream -> InputStream w/fd

this: InputStream w/fd -> InputStream w/impl

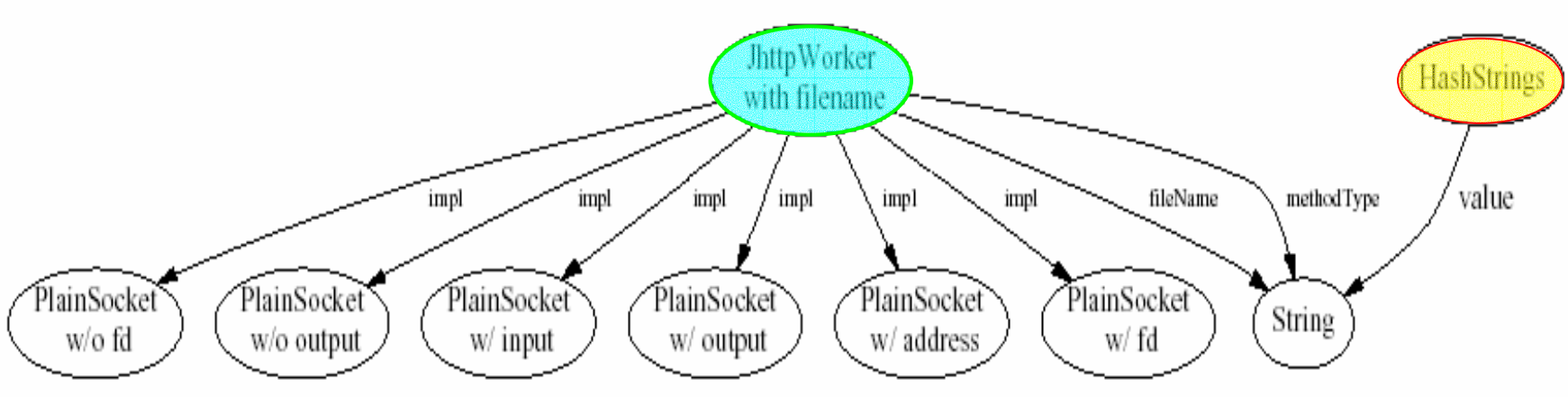
Role Relationship Diagrams



Multiple Object Data Structures



Multiple Object Data Structures



Portion of role relationship diagram for JhttpServer after part object abstraction

User Interface

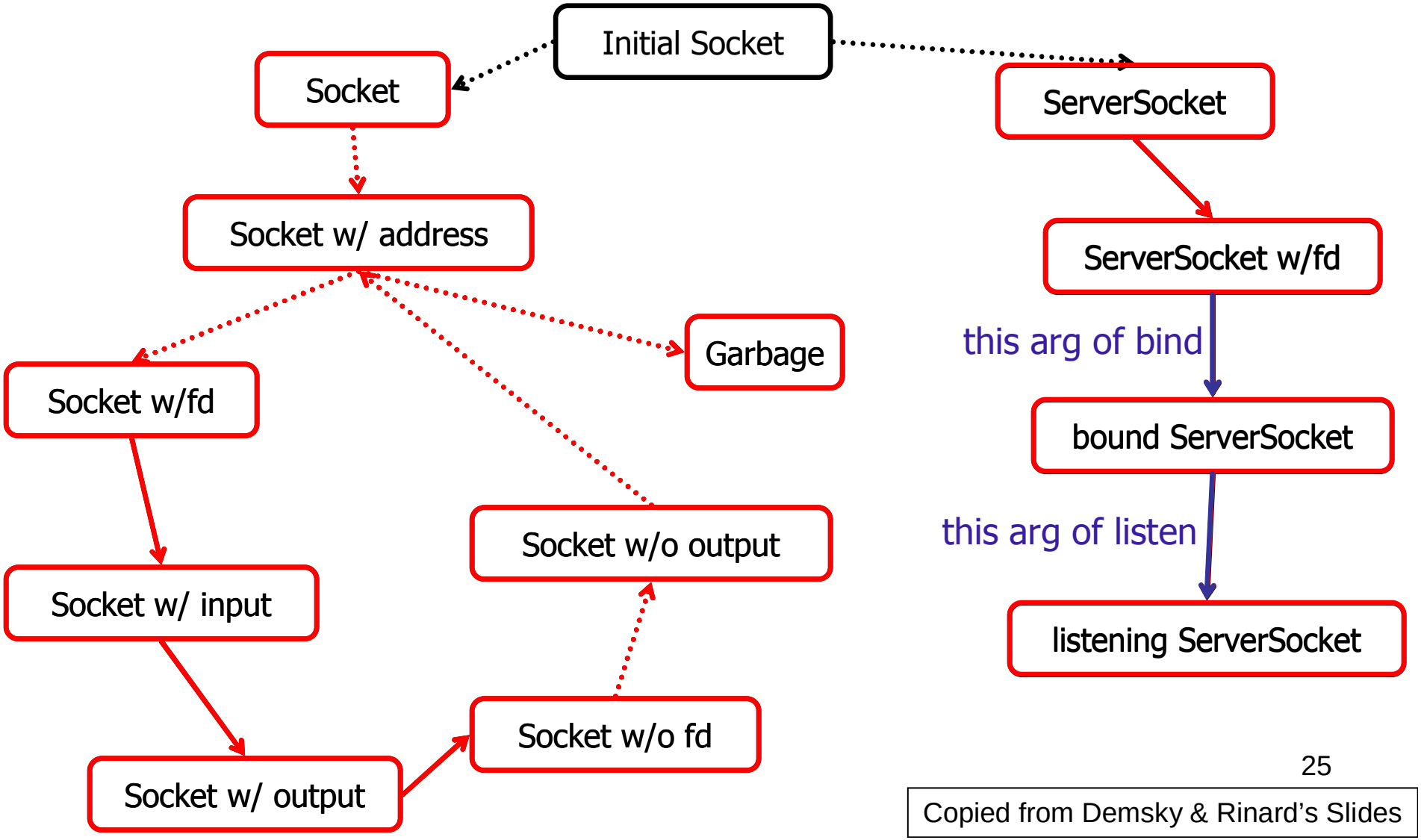
- The developer can:
  - Define multiple role subspaces
  - View projections of role transition diagrams and role relationship diagrams onto the defined role subspaces
  - Declare methods atomic to hide internal role changes or utilizing the multiple object abstraction feature

Exploration Strategy

- Begin with role transition diagrams of each class
- Find opportunities to simplify the role transition diagrams
- Browse enhanced method interfaces to discover important constraints on the parameters
- Observe the role relationship diagram

Experience -- JhttpServer

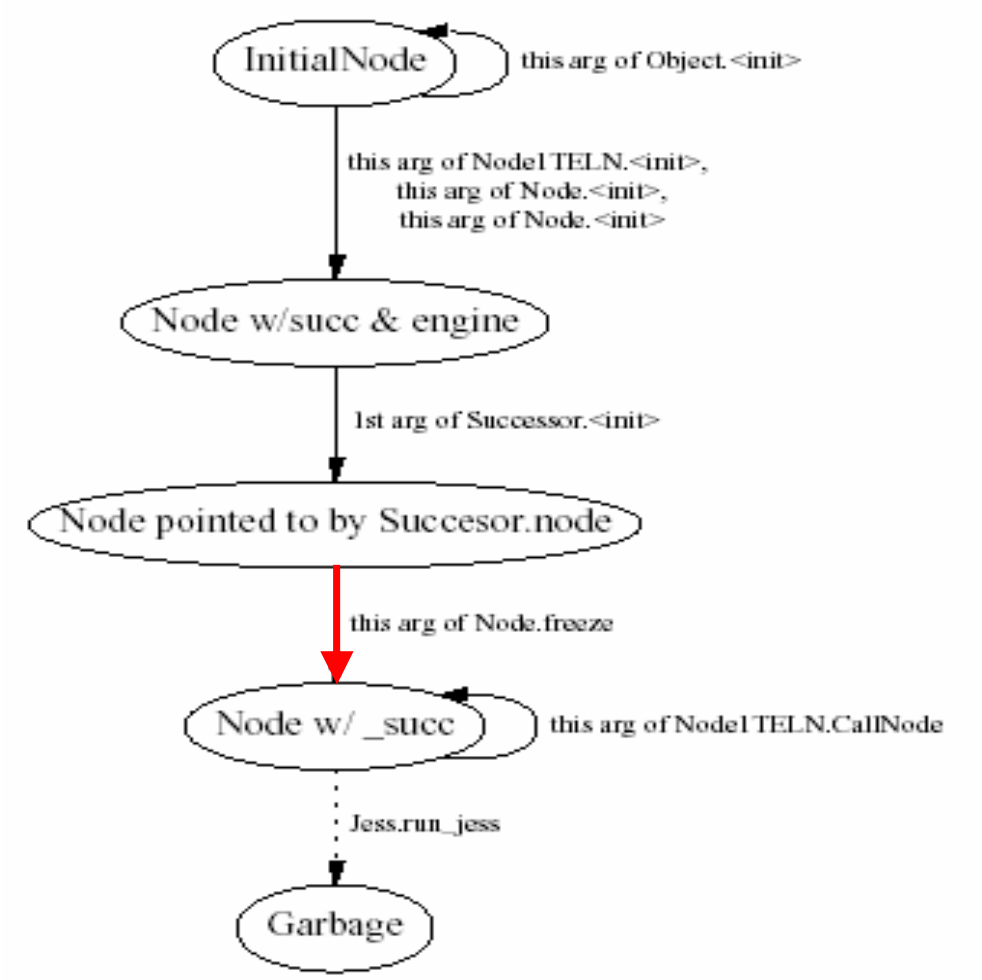
Role Transition Diagram for Socket



Copied from Demsky & Rinard's Slides

Experience -- Jess

Role Transition Diagram for Node1TELN



Role Description

Role **Node pointed to by Succesor.node**

Class: Node1TELN

Heap Aliases: Successor.node

Non-null Fields: engine, succ

Role Node w/ _succ

Class: Node1TELN

Heap Aliases: Successor.node

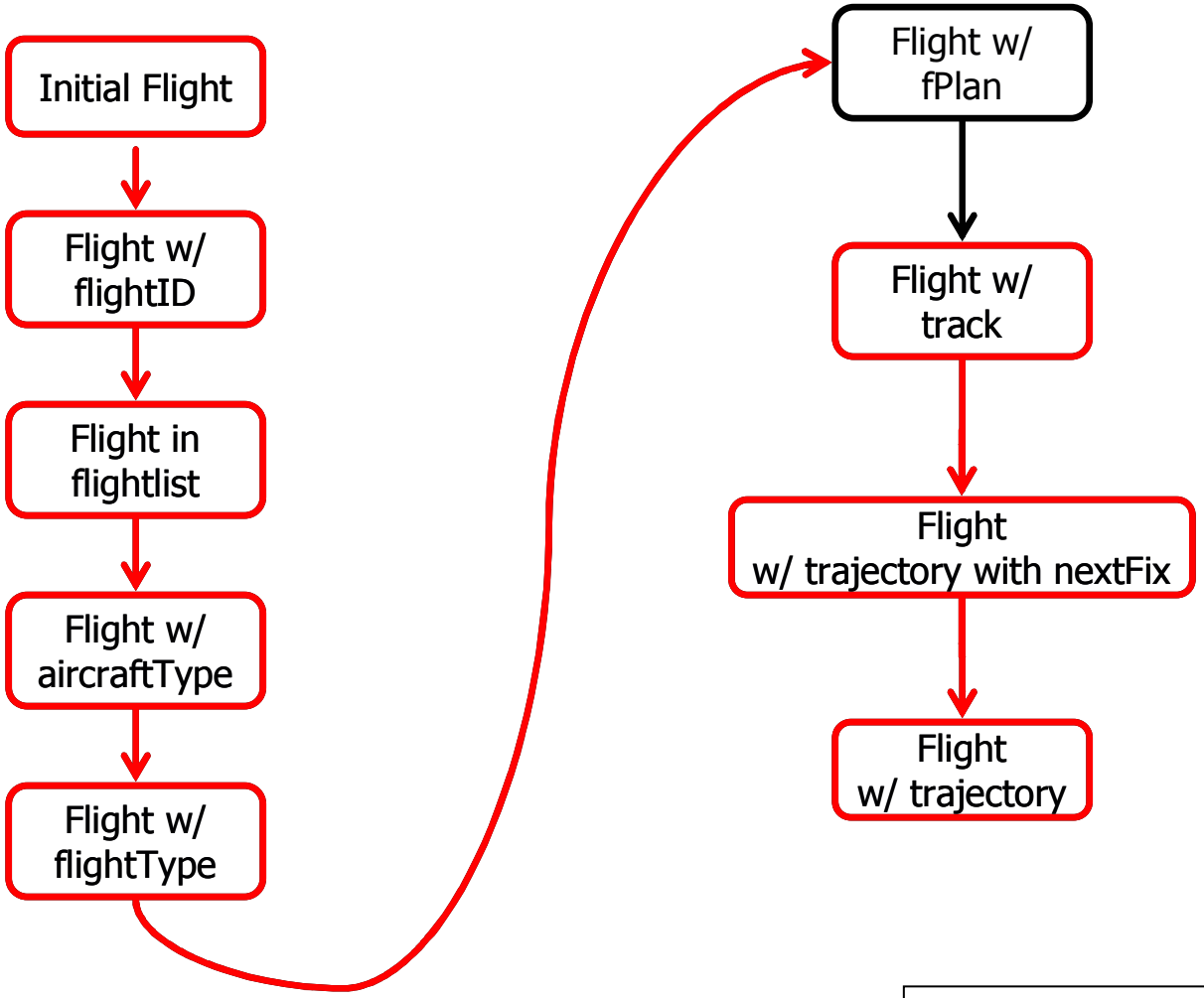
Non-null Fields: engine, _succ

Experience -- Jess

- Most nodes have exactly one Successor object referring to them
- The Node2 class has exactly two Successor objects referring to it
- No other kinds of nodes

Experience – Direct-To

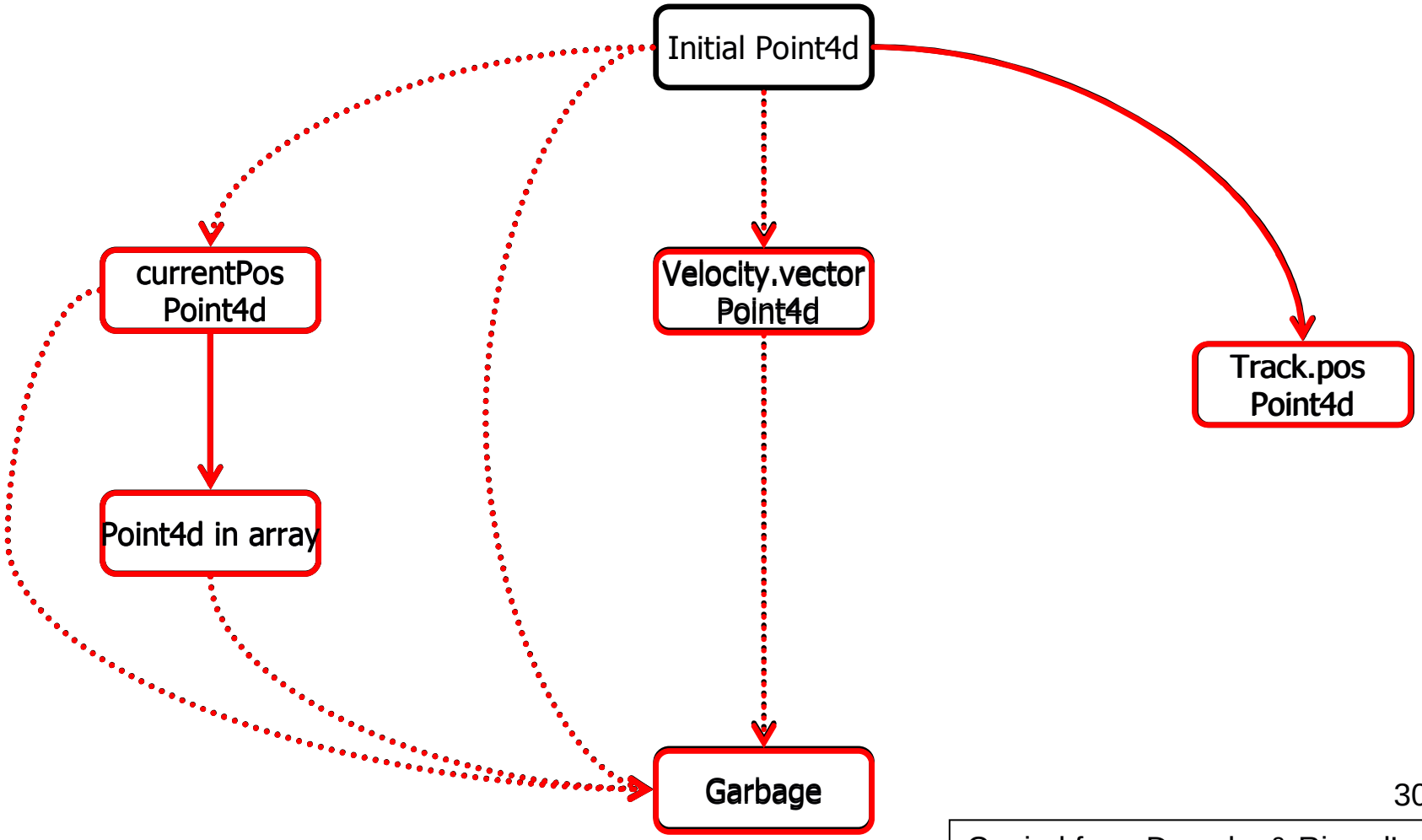
Role Transition Diagram for Flight



Copied from Demsky & Rinard's Slides

Experience – Direct-To

Role Transition Diagram for Point4d



Copied from Demsky & Rinard's Slides

Applications

- Program Understanding - help discover
  - different conceptual roles
  - Important referencing relationships between objects playing different roles
  - Constraints between roles and actions of program
- Maintenance
- Verifying expected Behavior
- Documentation
- Design

Related Works

- Design formalisms
  - The concept of abstract object states
- Program understanding tools
  - Properties of the objects that programs manipulate
- Static analyses
  - Automatically discovering or verifying properties of linked data structures

Conclusions

- Focus on changing object states – roles
- Role separation criteria
- Role subspaces
- Graphical role exploration

Questions?

- All examples in “Experience” are small examples
- Dynamic analysis only based on some specific executions
- Cost
- Scalability (No. of roles nodes, the relationship diagrams, …)
- ……