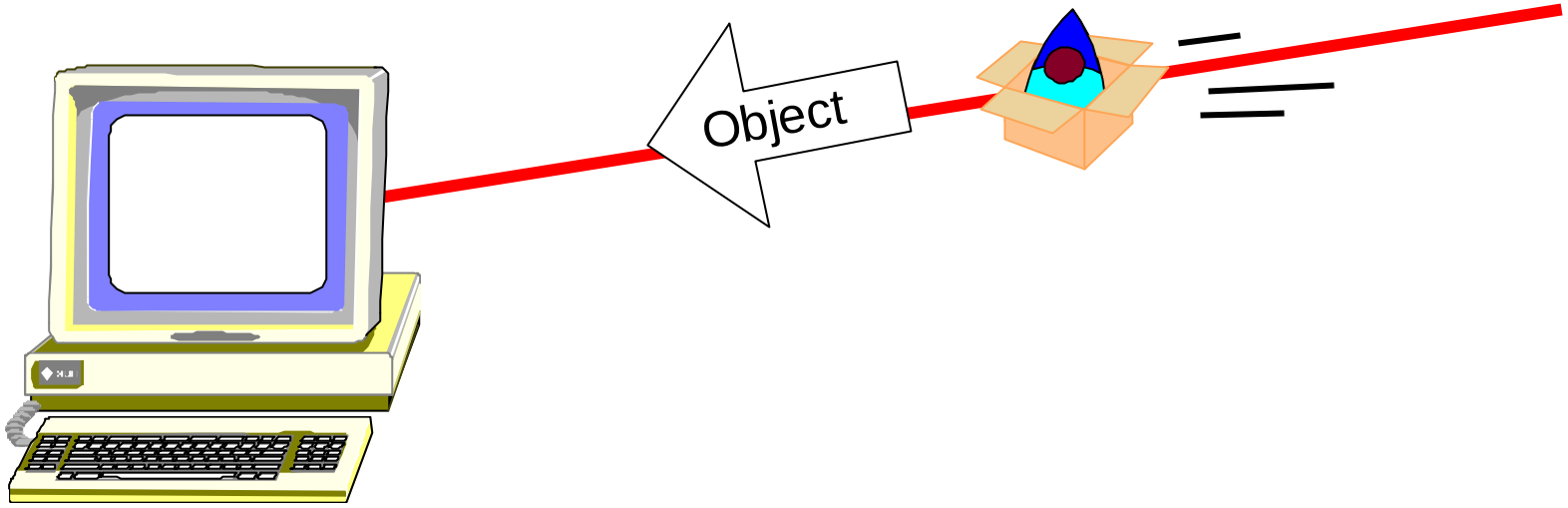
Mobile Java Classes

Update

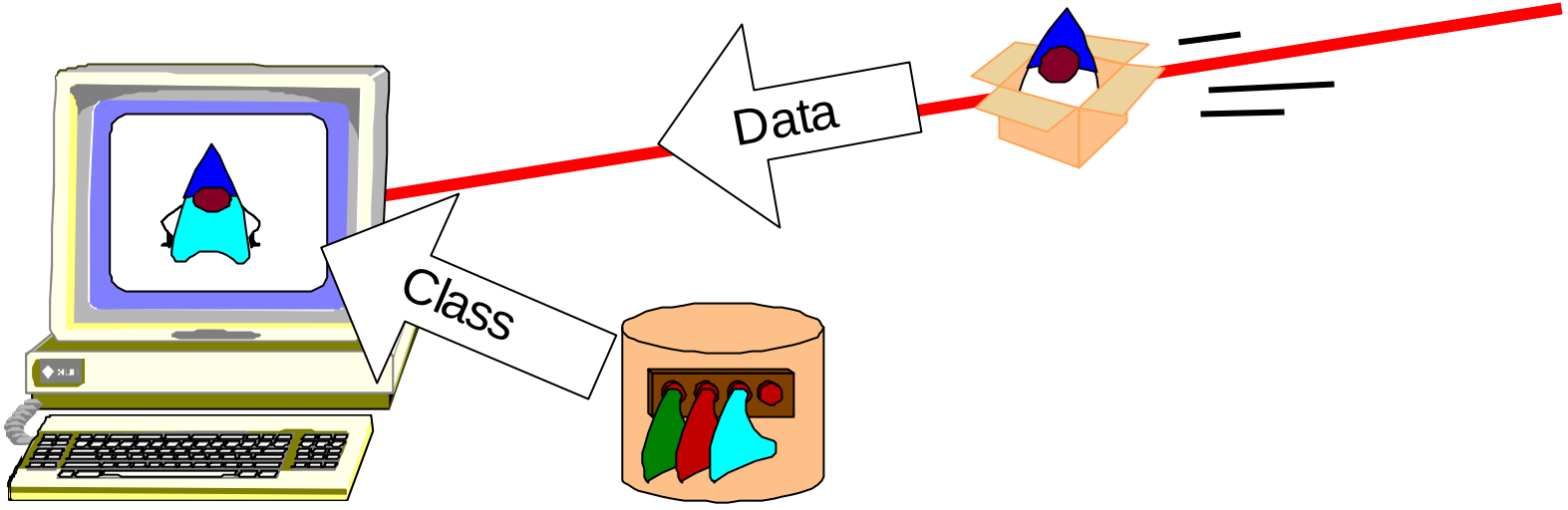
Richard Hayton

Mobile Java Objects



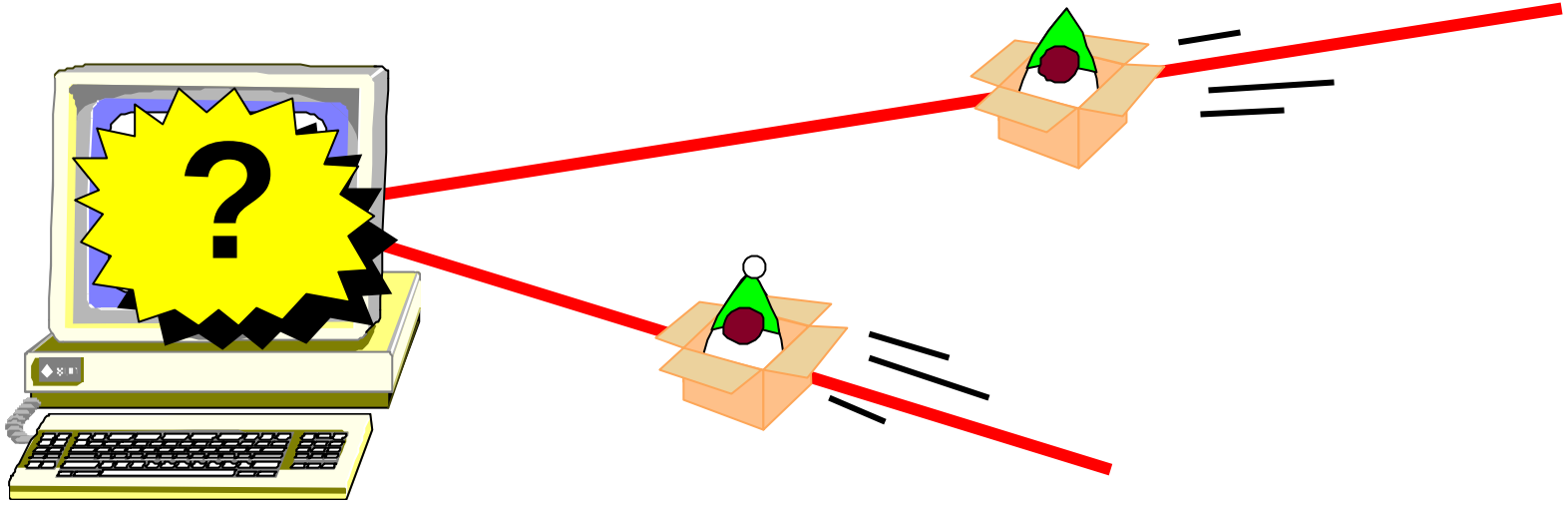
- Downloading objects into hosts:
  - Applets
  - Application Download
  - Agents
  - Remote Method Invocation

Object = Class + Data



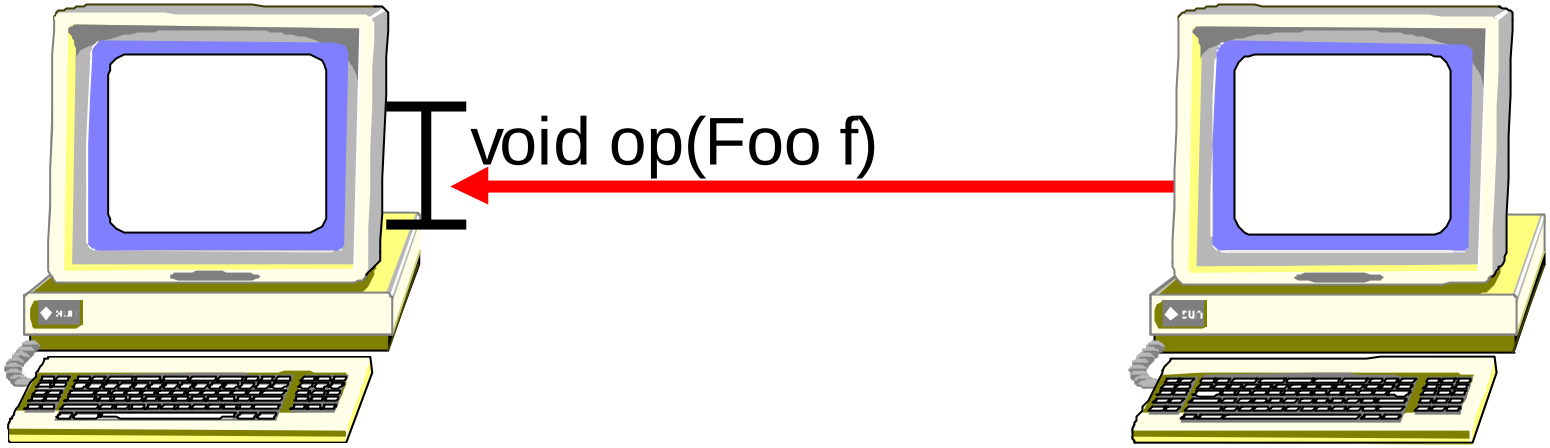
- Data loaded over wire
- Classes are loaded from:
  - local disc
  - web server (applets)
  - elsewhere?

Class Clash



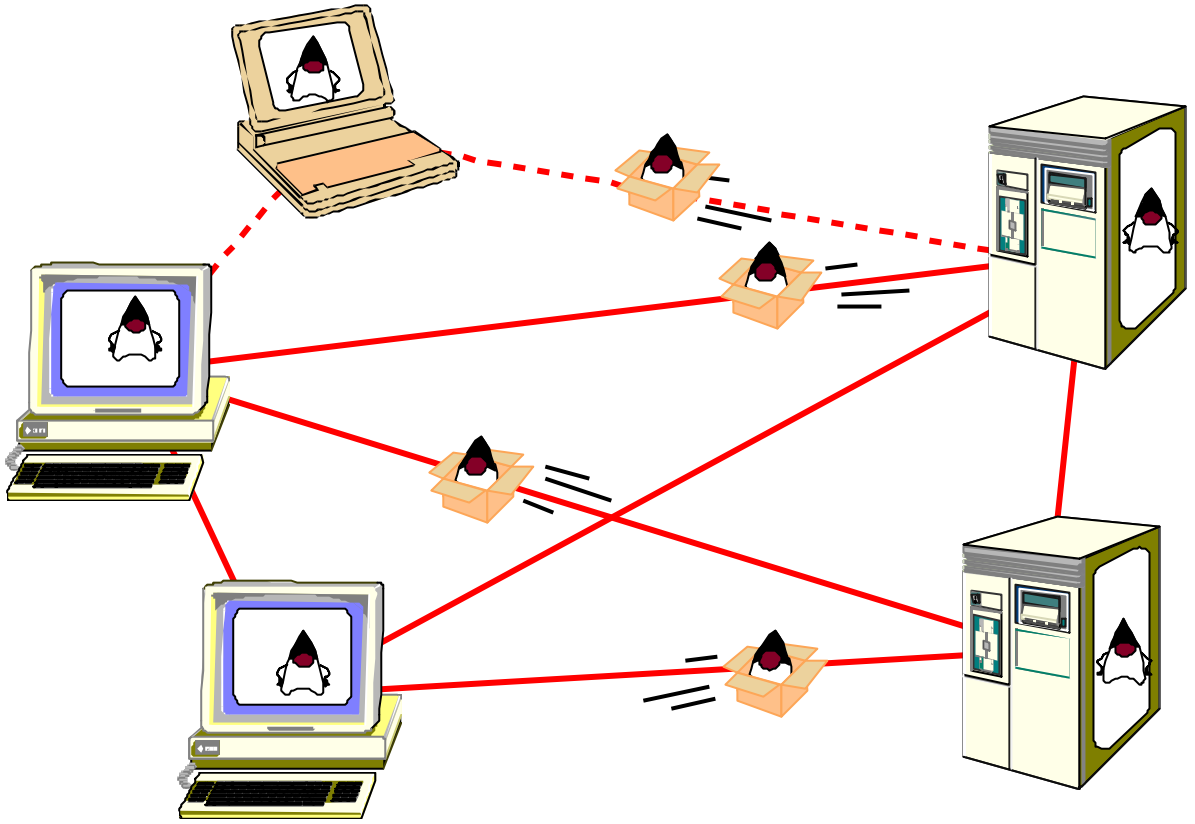
- What happens if we try and load two objects with different classes, but the same class name?
  - Sun's RMI Breaks
- *Is this a sensible thing to want to do?*

What do interfaces really define?



- ✓ Operation Name
- ✓ Types of arguments & results
- ? Names of subclasses of arguments/results
  - ? Names of classes required by subclassed args
  - ? Names of subclasses used by other clients

RMI Þ a single class namespace



- Cannot stagger rollout of new class versions
- Cannot have global ‘generic’ services
  - Traders, Persistent Object Stores, Agent Places

An old problem revisited….

- There is an analogous problem in functional languages.
- Consider the Lambda Calculus expression:

$$\lambda x.(\lambda y.xy)\ (z) \ = \ \lambda y.zy$$

- However, consider

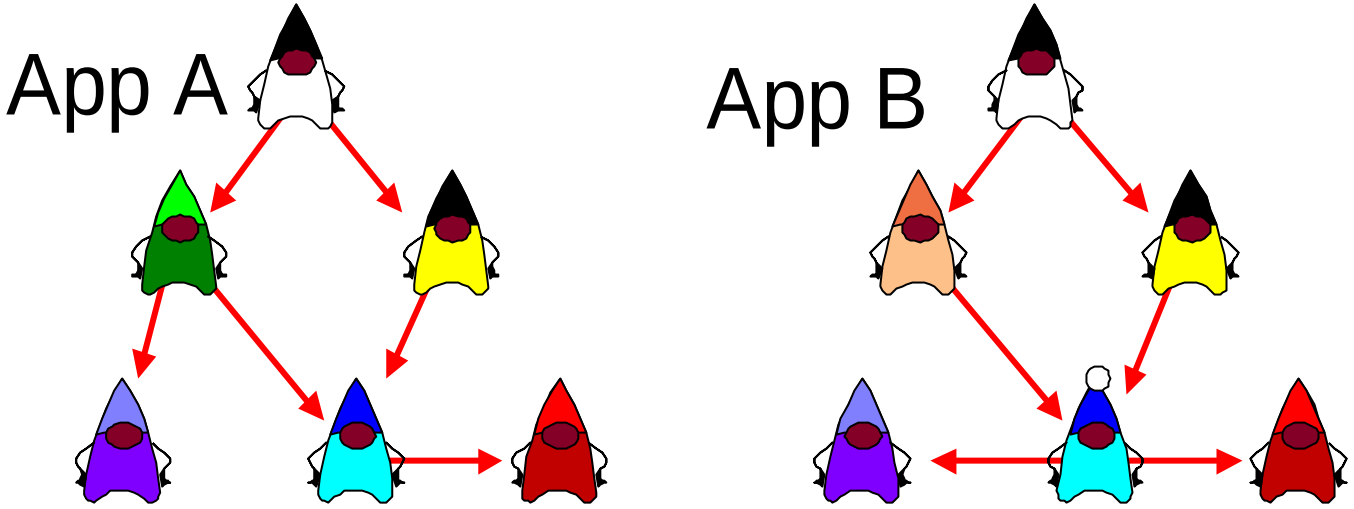
$$\lambda x.(\lambda y.xy)\ (y) \ = \lambda y.yy \ ???$$

- We avoid this by “bound variable renaming”
  - We arrange that all variable names inside the function are local to it
- $\lambda x.(\lambda \underline{y}.x\underline{y})\ (y) \ = \lambda \underline{y}.y\underline{y}$

Multiple Class Name Spaces

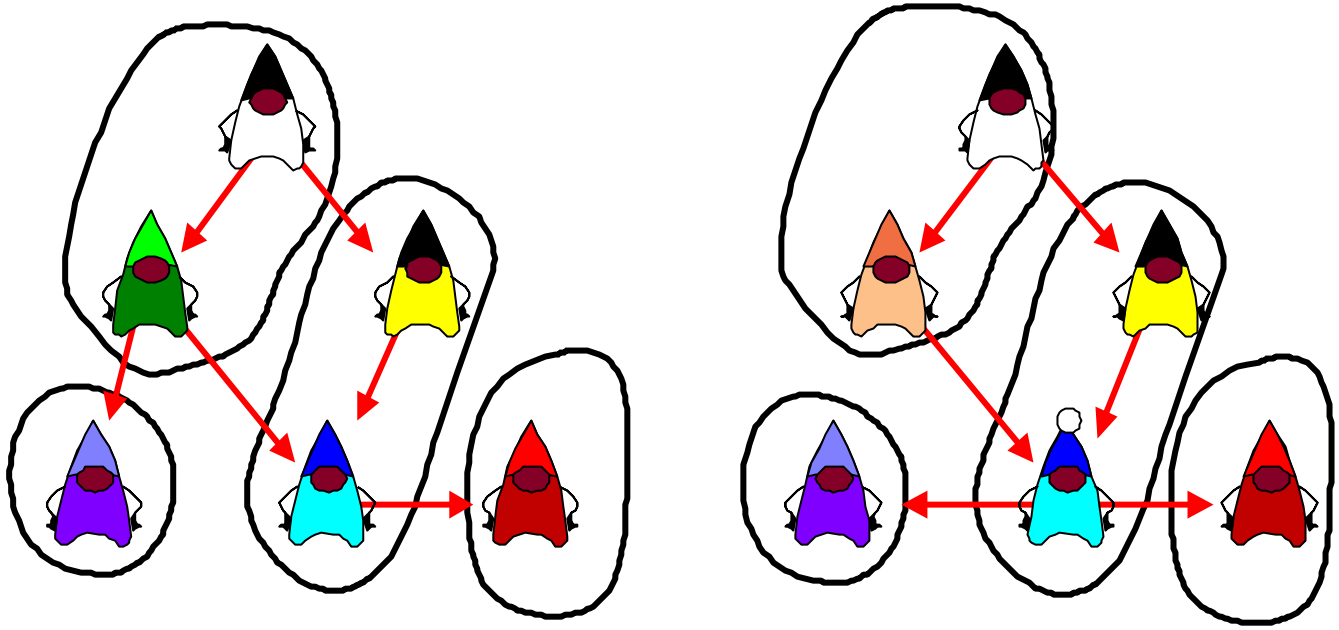
- We can apply an analogous solution to bound variable renaming, by using multiple class loaders to keep class namespaces separate.
- The namespaces may inherit from other namespaces, allowing a controlled degree of interaction
- There are a only a few restrictions
  - No circular dependencies allowed
  - Packages must be internal to namespaces
  - Inherited names cannot be overridden

Network Class Architecture



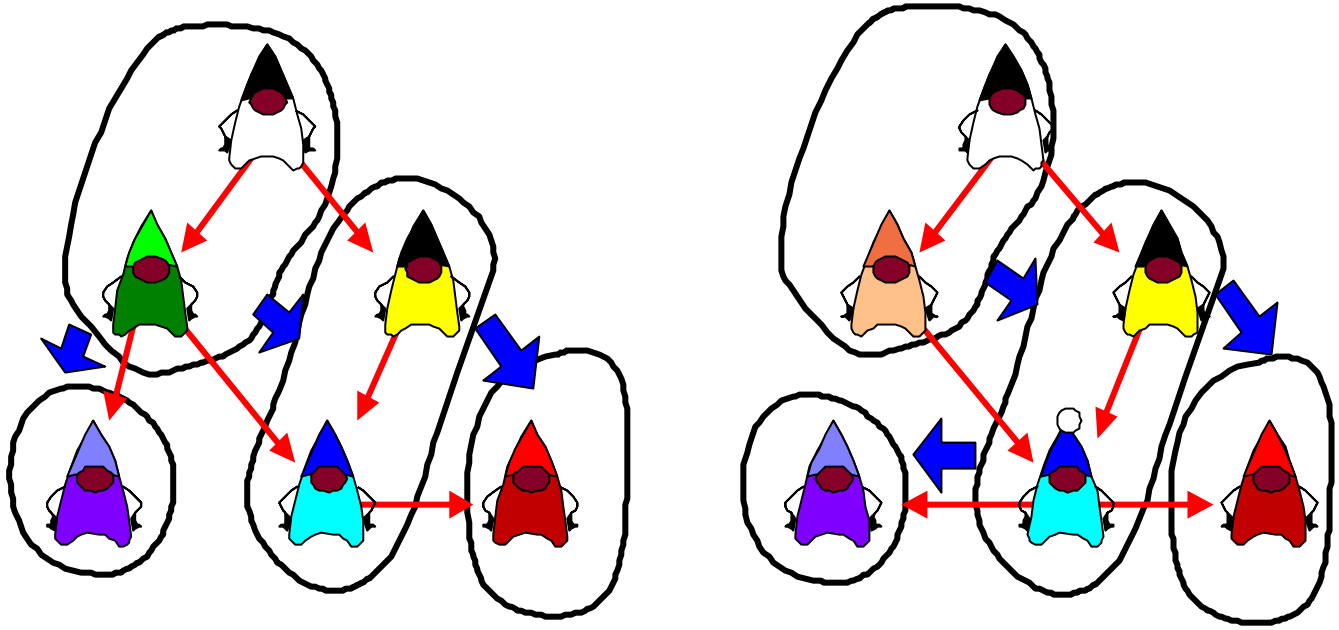
- The programmer writes applications as before

Network Class Architecture



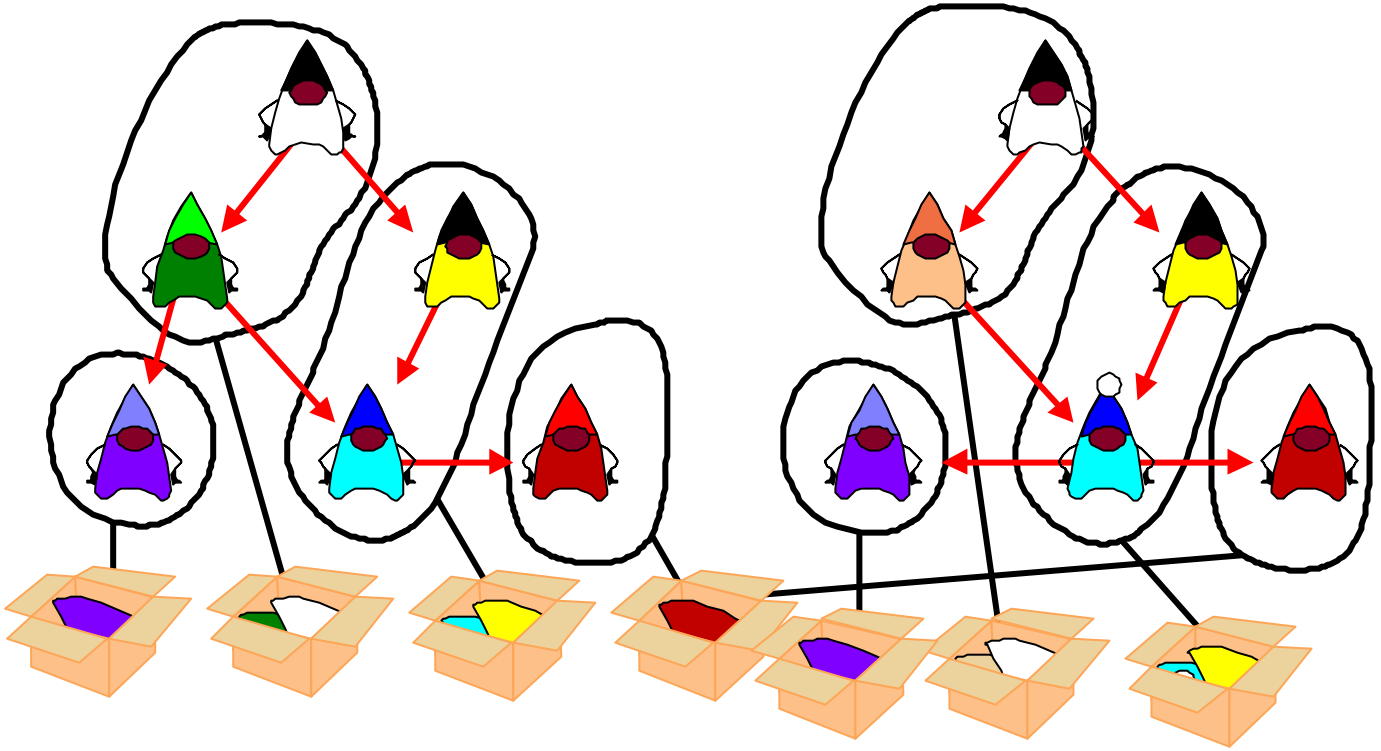
- The programmer writes applications as before
- Classes are organized into bundles (components ≡ Jar)

Network Class Architecture



- The programmer writes applications as before
- Classes are organized into bundles (components ≡ Jar)
- Bundles import other bundles and give explicit versioning information.

Network Class Architecture



- Each bundle is loaded by a classloader
  - allows late binding of imported bundles
  - bundle classloaders may be sharable or private
  - transparent to the application programmer

Network Class Repository

- Class Repositories serve classes to *local* apps.
  - They act as ‘code caches’
  - They allow managerial control over class use
- Bundles are published on Web Servers
  - As standard JARS (+ extra manifest info)
  - Only accessed by class repositories
- A JVM loads individual classes from its class repository on demand
  - including multiple versions of classes

Status - code written so far

- Class Repository
- Federation between class repositories
  - global scaling of class dissemination
- Per-JVM Class Loading Architecture
  - uses Class Repository and local classes
- Integration with FlexiNet serialization
  - Classes can be passed between JVMs regardless of name

Future Work

- Flexible bundle ‘import’ statements
  - Currently bundles import explicit versions
  - Could be extended to more flexible information (e.g. Bundle X, version 2,3 or 4)
- Class Publishers
  - Bundles of classes are published on Web servers.
  - The server does not know when (if) it can ever garbage collect the classes
- Policy Framework
  - Which classes should I load?
  - Integration with JDK 1.2, signed JARs etc.