A Component-Based Approach to Improving the Modularity of OSCAR

Geoffroy Vallée, Thomas Naughton and Stephen L. Scott

Oak Ridge National Laboratory, Oak Ridge, TN, USA

Jeffrey M. Squyres

Indiana University, Bloomington, IN, USA

Erich Focht

NEC High Performance Computing Europe, Stuttgart, Germany

Introduction

- Overview
- Motivation
- Component Architecture (OCA)
- Implementation
- Future Work
- Conclusion

What does OSCAR do?

- Wizard based cluster software installation
  - Operating system
  - Cluster environment
- Automatically configures cluster components
- Increases consistency among cluster builds
- Reduces time to build / install a cluster
- Reduces need for expertise

Overview

- Cluster installation and management toolkit
  - add/del cluster node
  - add/del cluster software
- Comprised of,
  - graphical wizard
  - libraries & scripts (glue/guts)
  - installation/validation tests
  - oscar packages
    - *core:* SIS, C3, Env-Switcher/Modules, ODA
    - *included:* PVM, Torque, LAM/MPI, ...
- OSCAR "users" get **all** of this! – good/bad?

OSCAR Score Sheet

Good/Bad depends on type of "user"

a) end-user

b) developer or "re-user"

Good

- Many Linux distributions
- Extensible Meta-Packages
  - Decouples applications from core infrastructure
  - Leveraged by "spin-offs"
    - HA-OSCAR, SSS-OSCAR, SSI-OSCAR

Bad

- Maintain/Add distributions
- Initial implementation had RPM bias
  - Meta-packages
  - Libraries/Scripts
- Modification to *core* infrastructure is too difficult

How bad is it?

- End-users
  - Ok, not concerned with *core* anyway
  - Bad, indirectly effected by slow release cycles
- Developers / Re-users
  - Ok, some restructuring for non-RPM
  - Ok, not <u>all</u> concerned with changing core
  - Bad, monolithic design slows innovation
  - Bad, tightly coupled core limits "spin-offs"

<u>Example</u>: Add a new Linux distribution

- Must touch several areas to make changes
- Testing is impacted because changes throughout

New Class of Users

- The "re-users" are major motivation for changes to the *core* infrastructure
  - A need to replace pieces of the system, as is done via extensible meta-package system
  - A need to allow alternate approaches to OSCAR
  - Alternate approaches based on many factors: technical, political or investigative

<u>Example</u>: Replace C3 tool with an alternate approach

- A new more scalable C3 (investigative/technical)
- A custom tool built "in house" (political/technical)
- A tool for a unique environment [no-TCP] (technical)

Prior Improvements

- Previous efforts addressed select areas
  - PackMan/DepMan
    - Native package manager abstraction
  - Database
    - Centralize access and revised data schema
  - Improve “config.xml”
    - Less RPM bias and modify for better XML validation

- Not address more fundamental problems
  - Monolithic design
  - Tightly coupled infrastructure (core)

Component Architecture

- OSCAR Component Architecture (OCA)
  - Effort to improve modularity within OSCAR
  - Based on component technology
  - Influenced by Open MPI component design*

- Modularity
  - Isolate functionality
  - Improve potential for reuse

- Customization
  - Adapt to new needs (static)

** Members of the Open MPI project are involved in OSCAR, so it was inevitable. :-)*

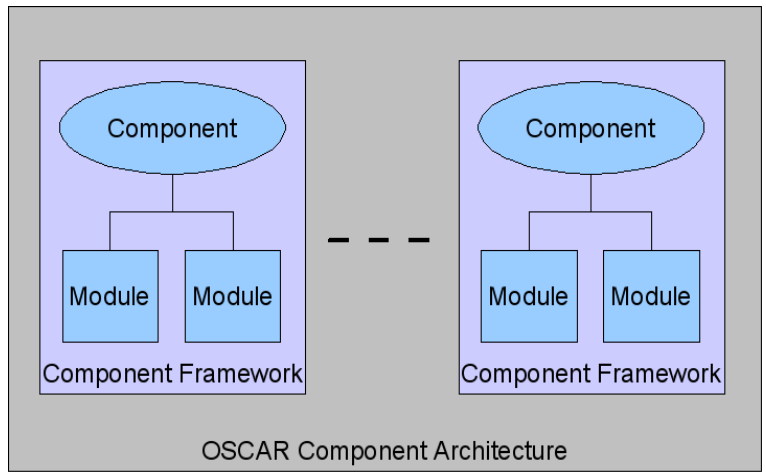
OCA Structure

Architecture

- Manage frameworks
- Provides basic support

Framework

- Manage a functional unit
- Liason for rest of system
- Defines the semantics/contract



Component

- General abstraction for a unit

Module

- Concrete implementation for a unit

Component/Module analogous to OOP class/object

OCA Advantages

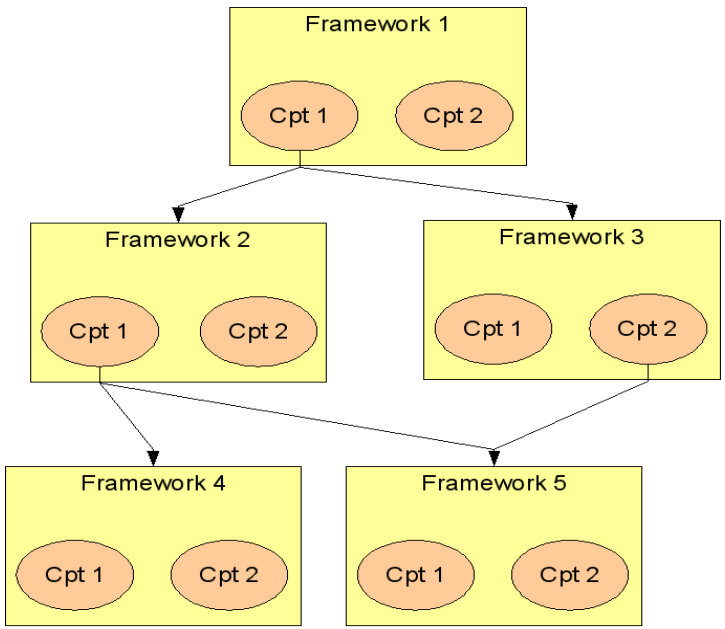
- Modularity
  - Begin breaking monolithic design into pieces, which helps reduce the tight coupling
- Organization / Structure
  - Developers know where to find/change things
- Extensibility
  - Add alternate approaches (components)
  - Add new approaches (frameworks)
- Reuse
  - The “spin-offs” can better customize without having to effectively fork for their needs

Interfaces

- Interfaces
  - The heart of a component-based system
  - Define the contracts, which govern usage
- Framework Interfaces
  - Expose high-level generic access to users
  - Encode the policy
    - selection patterns: *one-of-many* / *many-of-many*
  - Ex: open(), execute()
- Component Interfaces
  - Provide a standard for (internal) framework interactions
  - Hides the appliability logic behind interface
    - component determines if suitable for use by framework

Framework Hierarchies

- Frameworks are associated with a specific task
  - The underlying components actually do the work
- Frameworks can be linked together to perform more complex tasks
- Increased level of abstraction



Implementation

- Written in Perl and organized as as a collection of modules
- Frameworks create a module and directory to house components
- Components determine applicability and if so they successfully “open”

```
$OSCAR_HOME/lib/OSCAR/
                      |
                      |--> Debugger.pm
                      |--> OCA.pm
                      |--> OCA/
                             |--> Framework1.pm
                             |--> Framework1/
                             |      |--> Component1.pm
                             |      |-->     ...
                             |      |--> Component{N}.pm
                             |
                             |-->     ...
                             |
                             |--> Framework{N}.pm
                             |--> Framework{N}/
                                    |--> Component1.pm
                                    |-->     ...
                                    |--> Component{N}.pm
```

Basic Usage

1. User "opens" a Framework.
   Framework open()
   → Component1 open()
   → Component2 open()
   ...
   → Component*N* open()

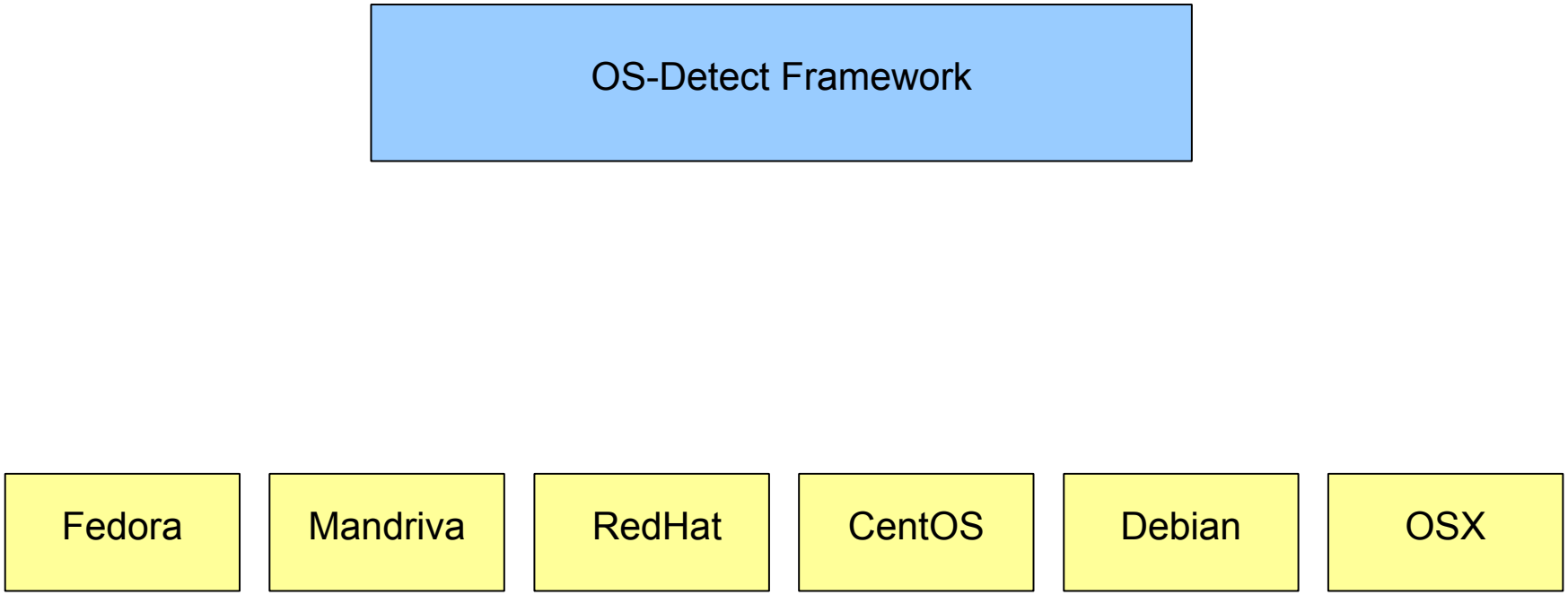
2. Framework selects (policy) from "applicable" components (those that successfully opened).

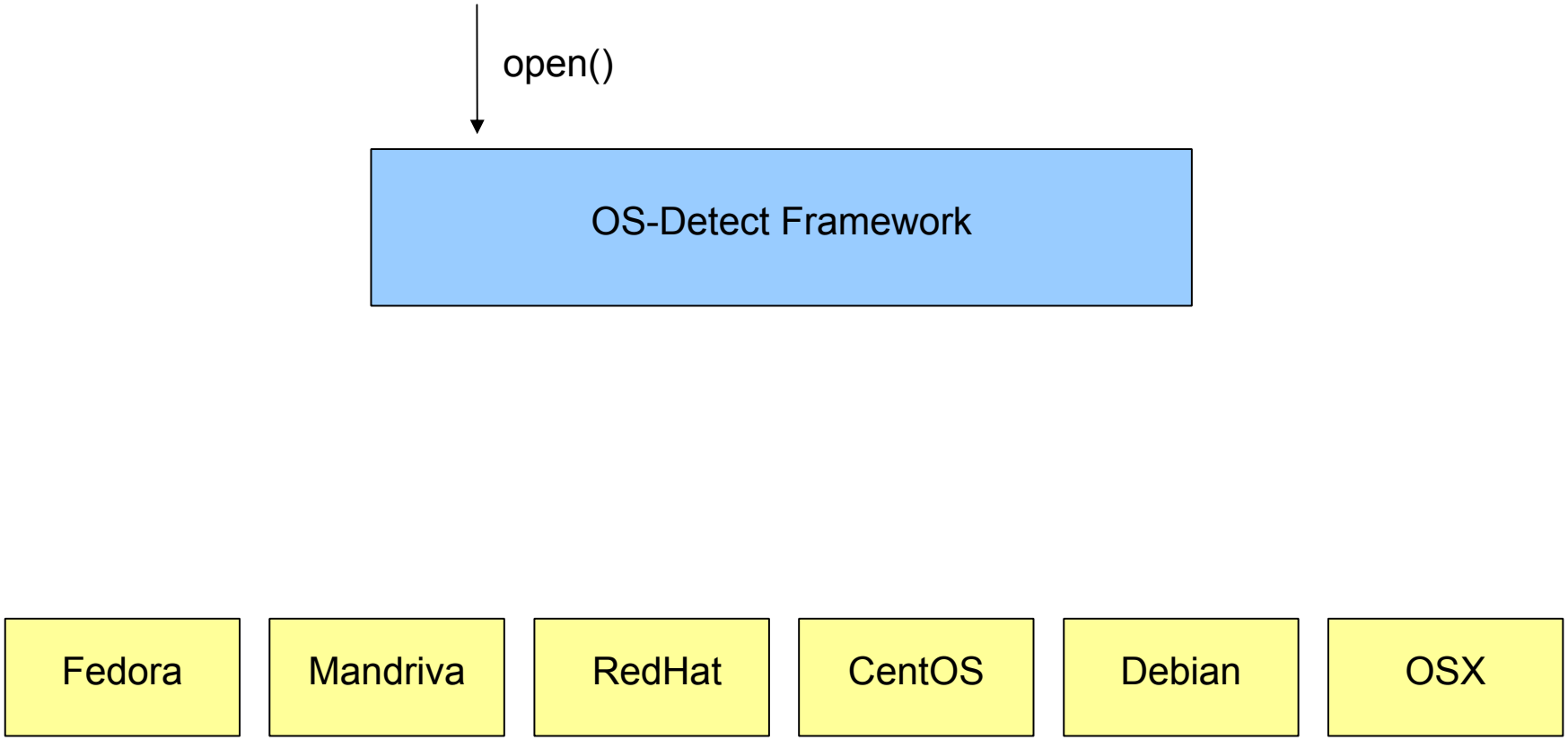
3. Framework returns handle for component to user

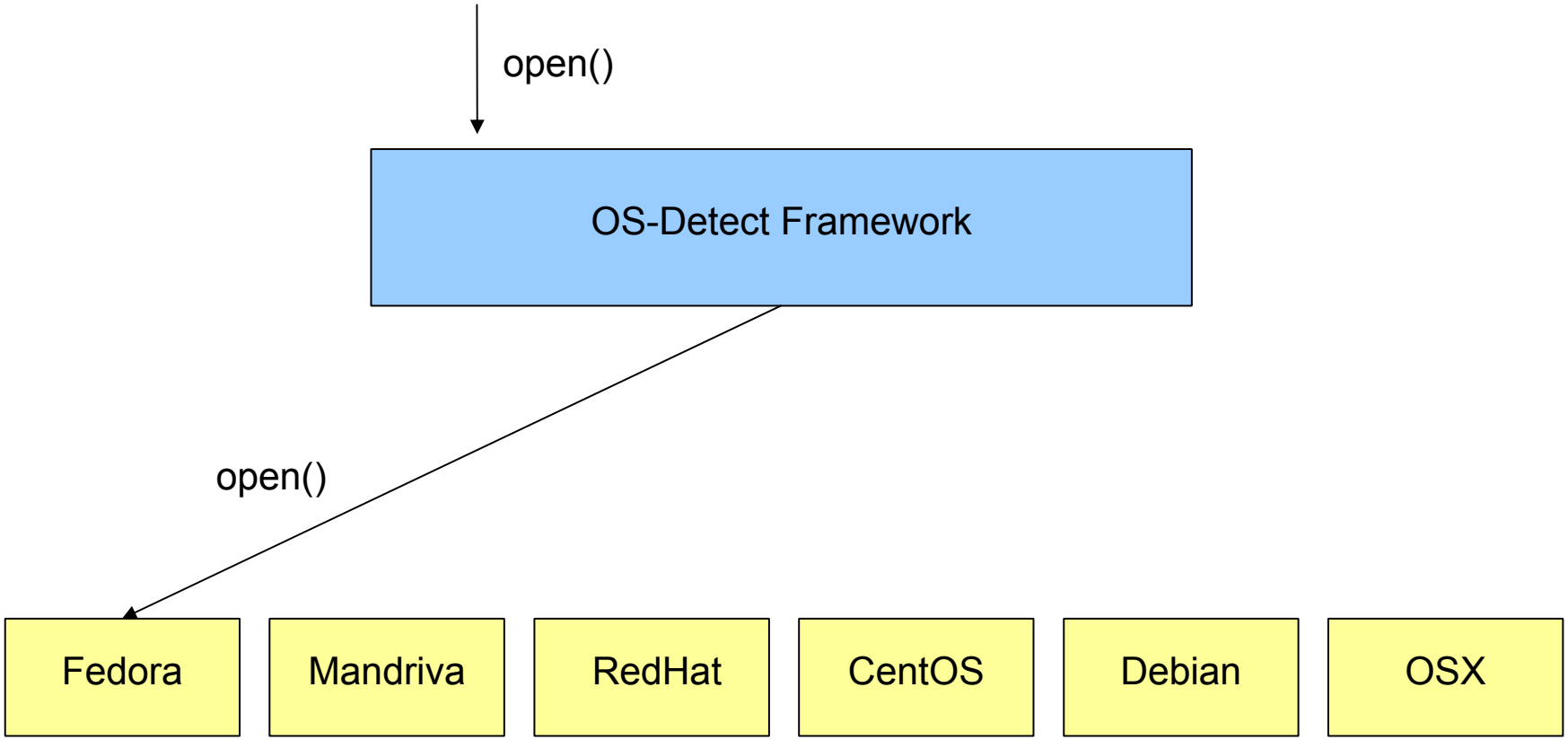
Framework: OS_Detect

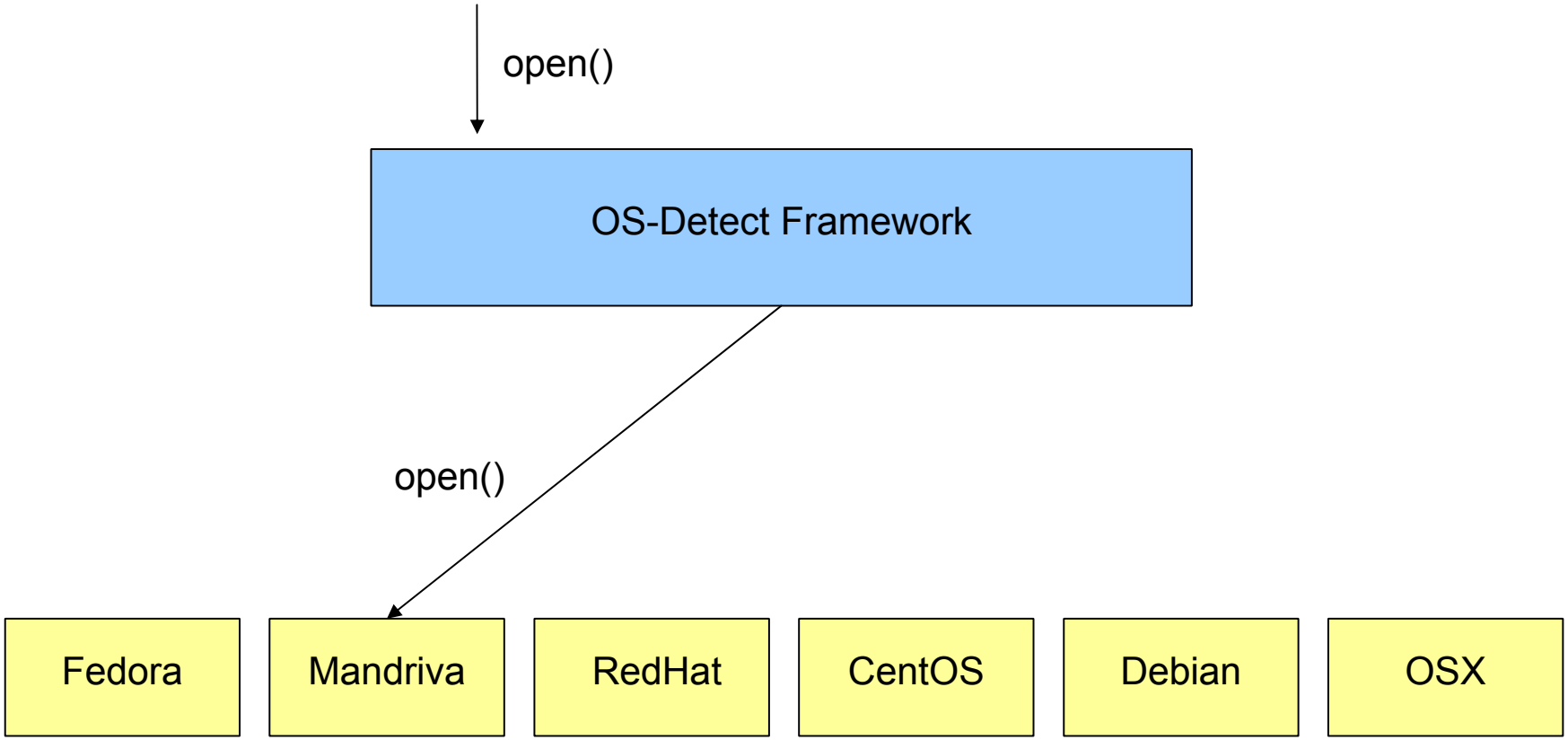
- First framework – very simple
- Mechanism for platform identification
  - Simplify support for new distros & CPU types
- Components are the “supported” platforms
- Selection pattern is one-of-many
  - Only one platform (component) should be applicable*
- Provides the platform data via Perl hash
  - Ex. os=>'linux', arch=>'i386', distro=>'centos', etc.

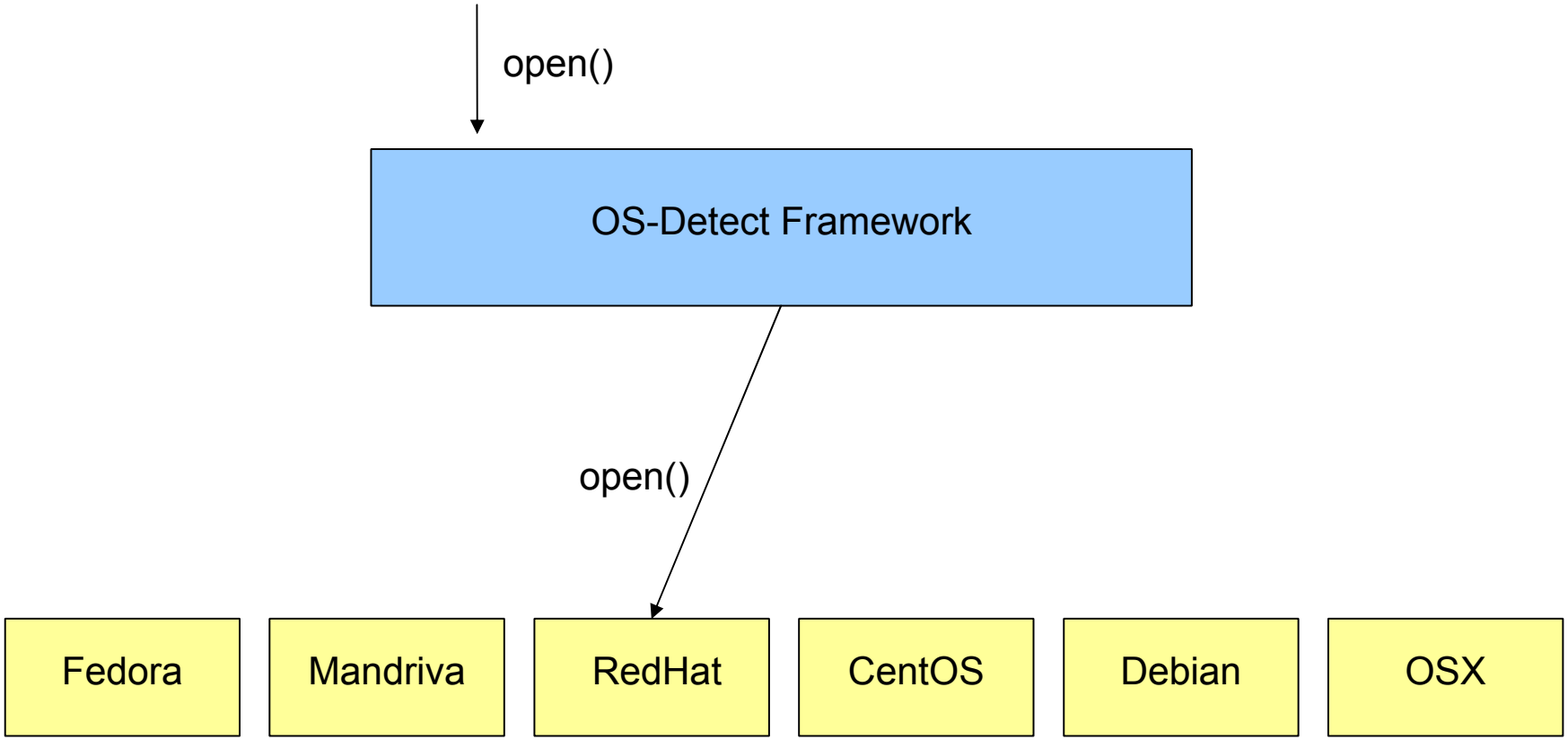
* ignoring virtualization

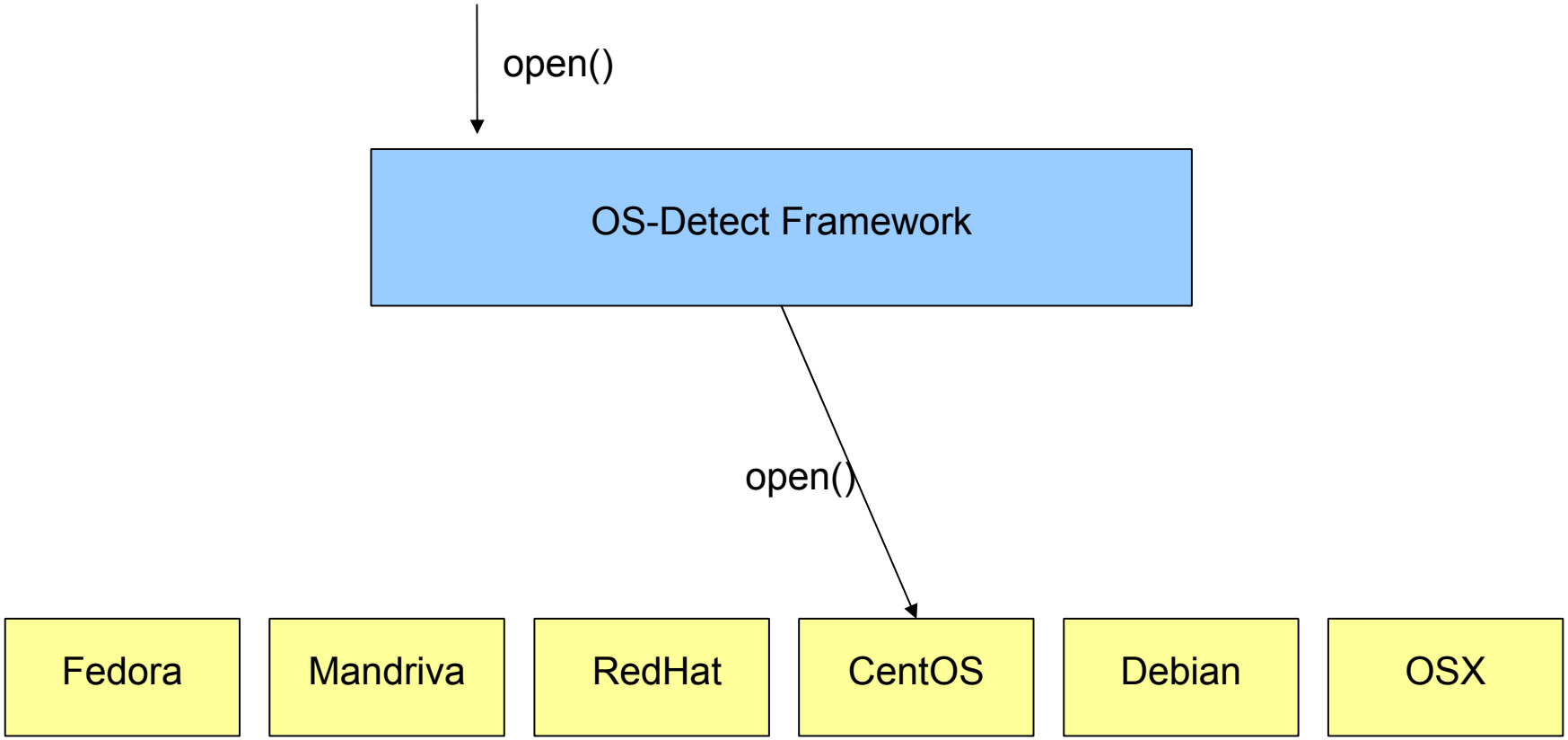

OS-Detect Framework
Fedora
Mandriva
RedHat
CentOS
Debian
OSX

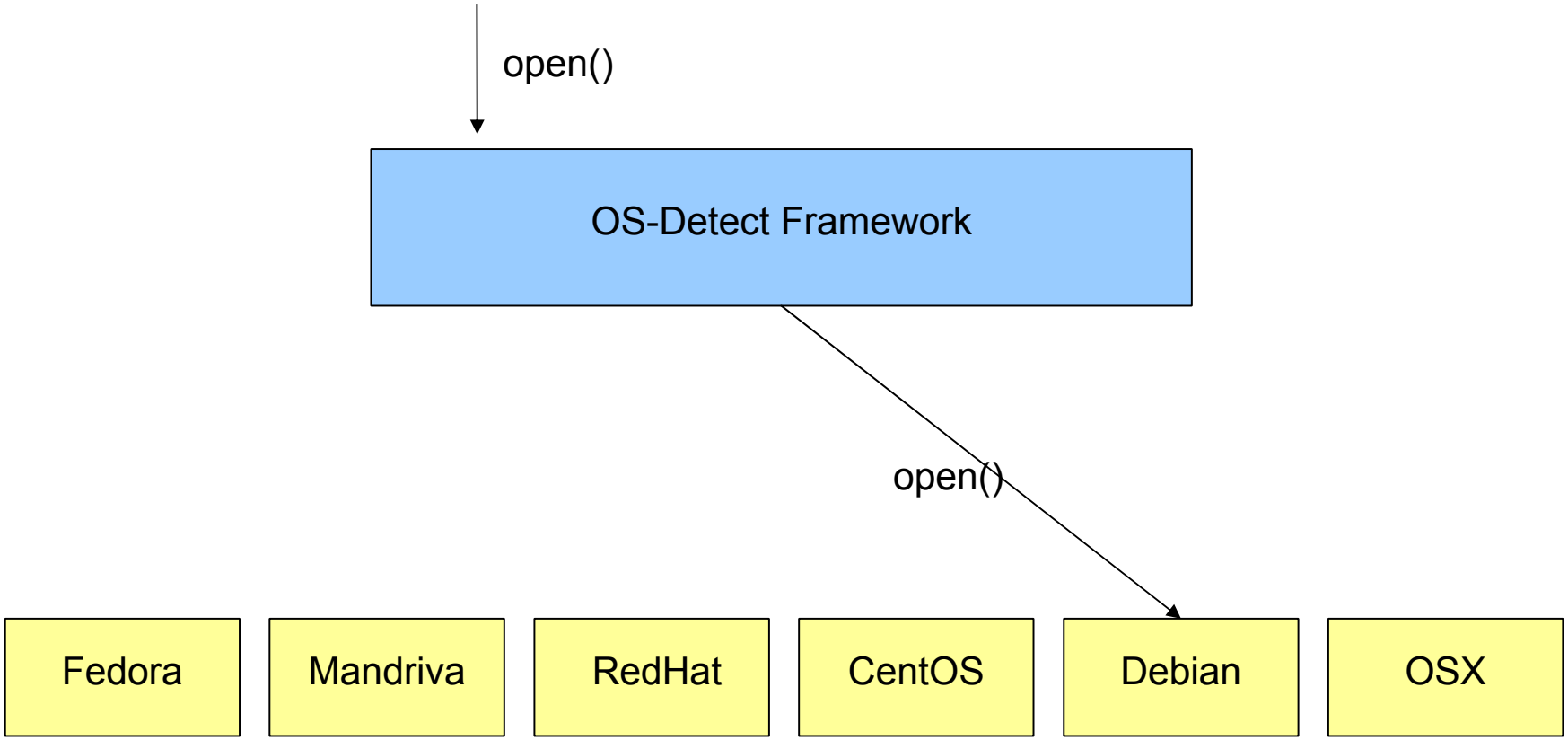

open()
OS-Detect Framework
Fedora
Mandriva
RedHat
CentOS
Debian
OSX

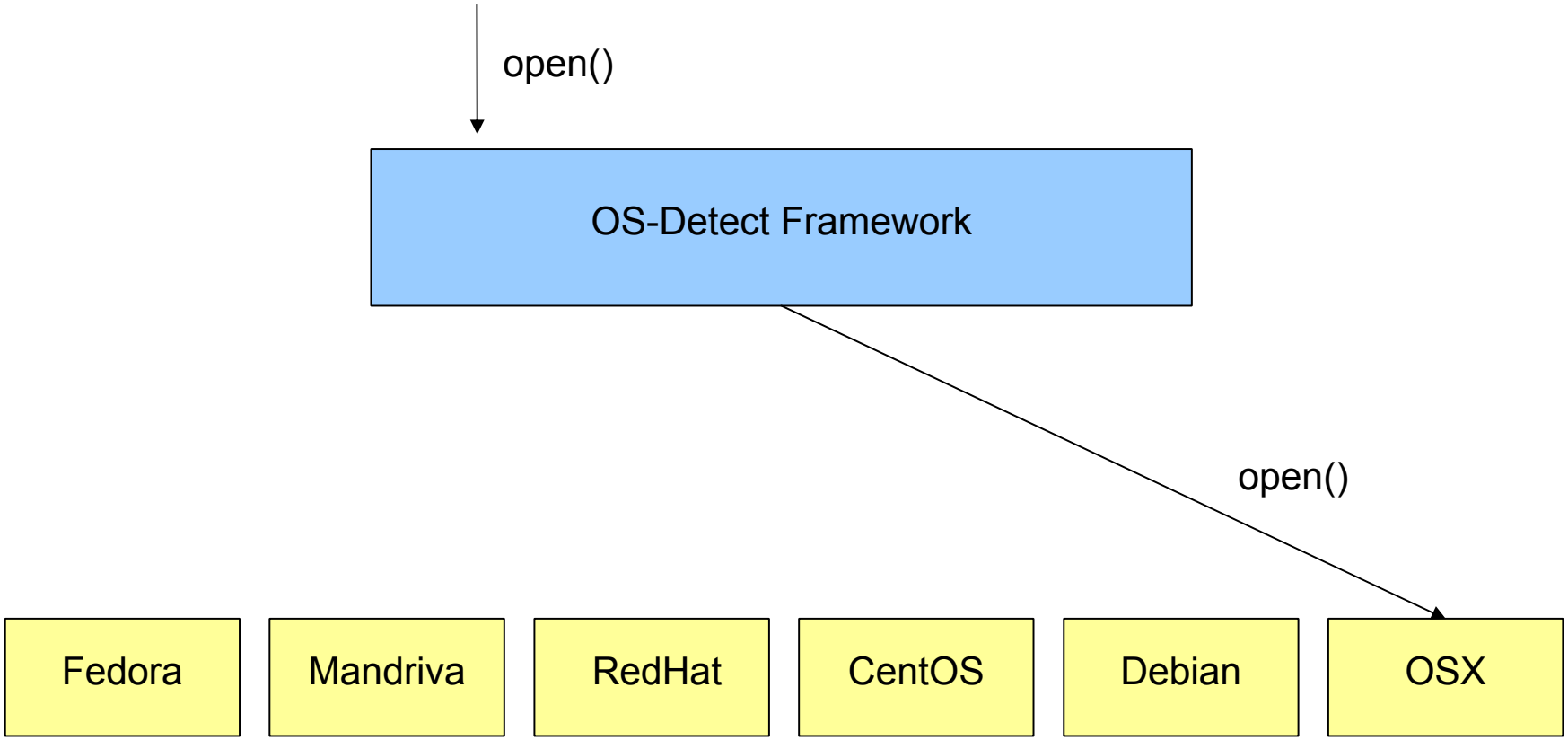

open()
OS-Detect Framework
open()
Fedora
Mandriva
RedHat
CentOS
Debian
OSX


open()
OS-Detect Framework
open()
Fedora
Mandriva
RedHat
CentOS
Debian
OSX

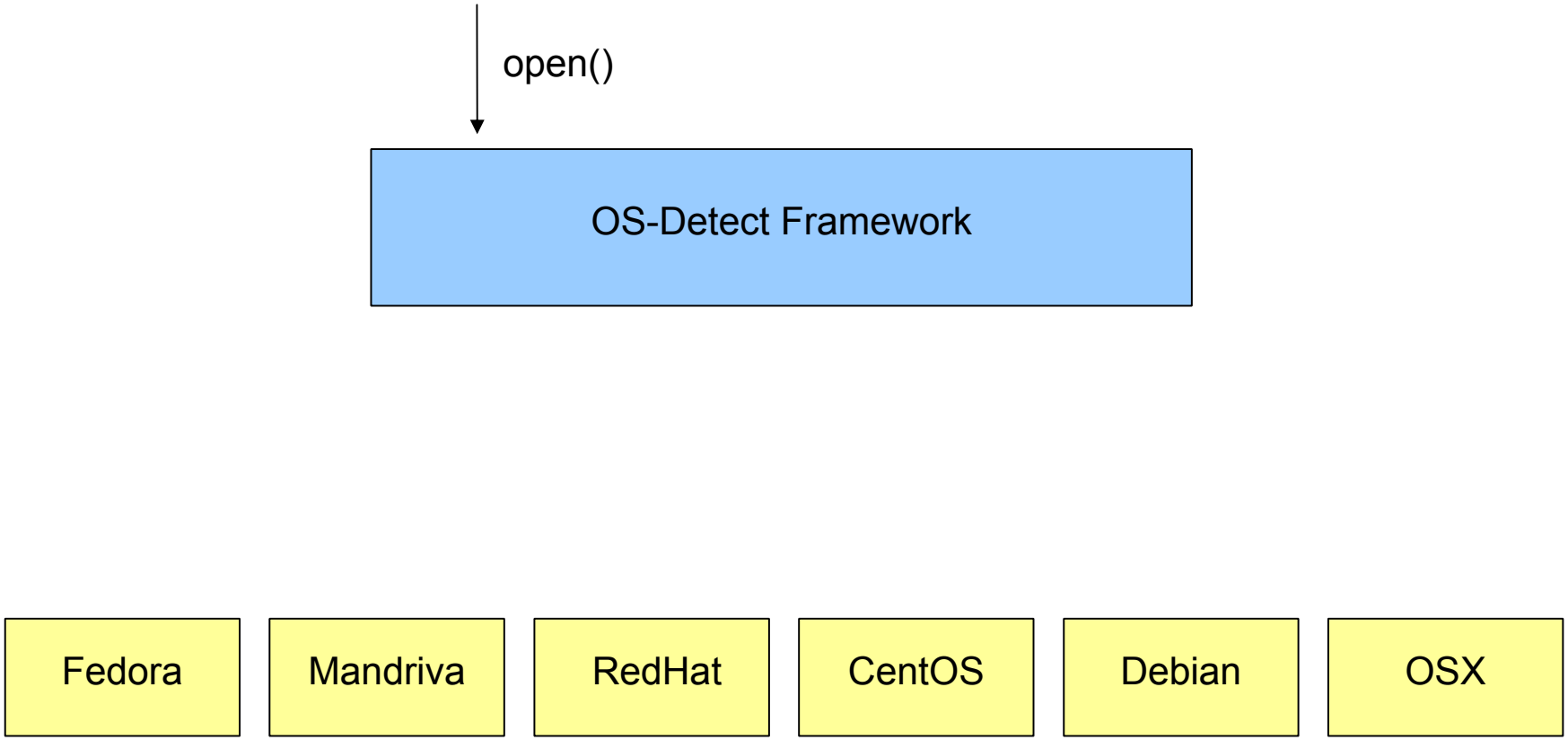

open()
OS-Detect Framework
open()
Fedora
Mandriva
RedHat
CentOS
Debian
OSX


open()
OS-Detect Framework
open()
Fedora
Mandriva
RedHat
CentOS
Debian
OSX

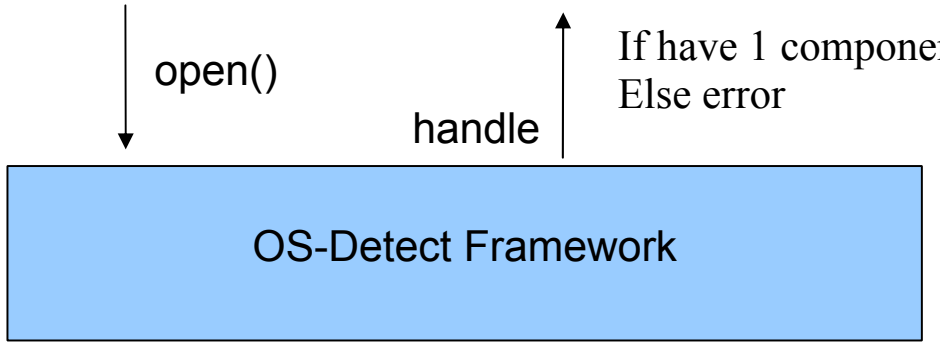

open()
OS-Detect Framework
open()
Fedora
Mandriva
RedHat
CentOS
Debian
OSX


open()
OS-Detect Framework
open()
Fedora
Mandriva
RedHat
CentOS
Debian
OSX

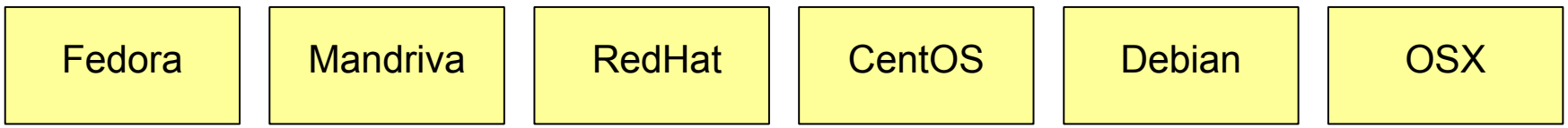
Selection pattern: *one-of-many*



Selection pattern: *one-of-many*



If have 1 component, return handle
Else error



Future Work

- Begin breaking core facilities into frameworks and components
- Adjust current implementation to a more object-oriented approach
- Improve testing and diagnostics of OCA

Summary

- OSCAR “users” (re-users) have increased
- The core infrastructure is tightly coupled due to a monolithic design
- Use a component-based approach to improve modularity/customization – OCA
- First framework OS_Detect provides canonical source for supported platform data

Questions?

OSCAR Homepage

http://oscar.openclustergroup.org/

ORNL's work was supported by the U.S. Department of Energy,
under Contract DE-AC05-00OR22725.