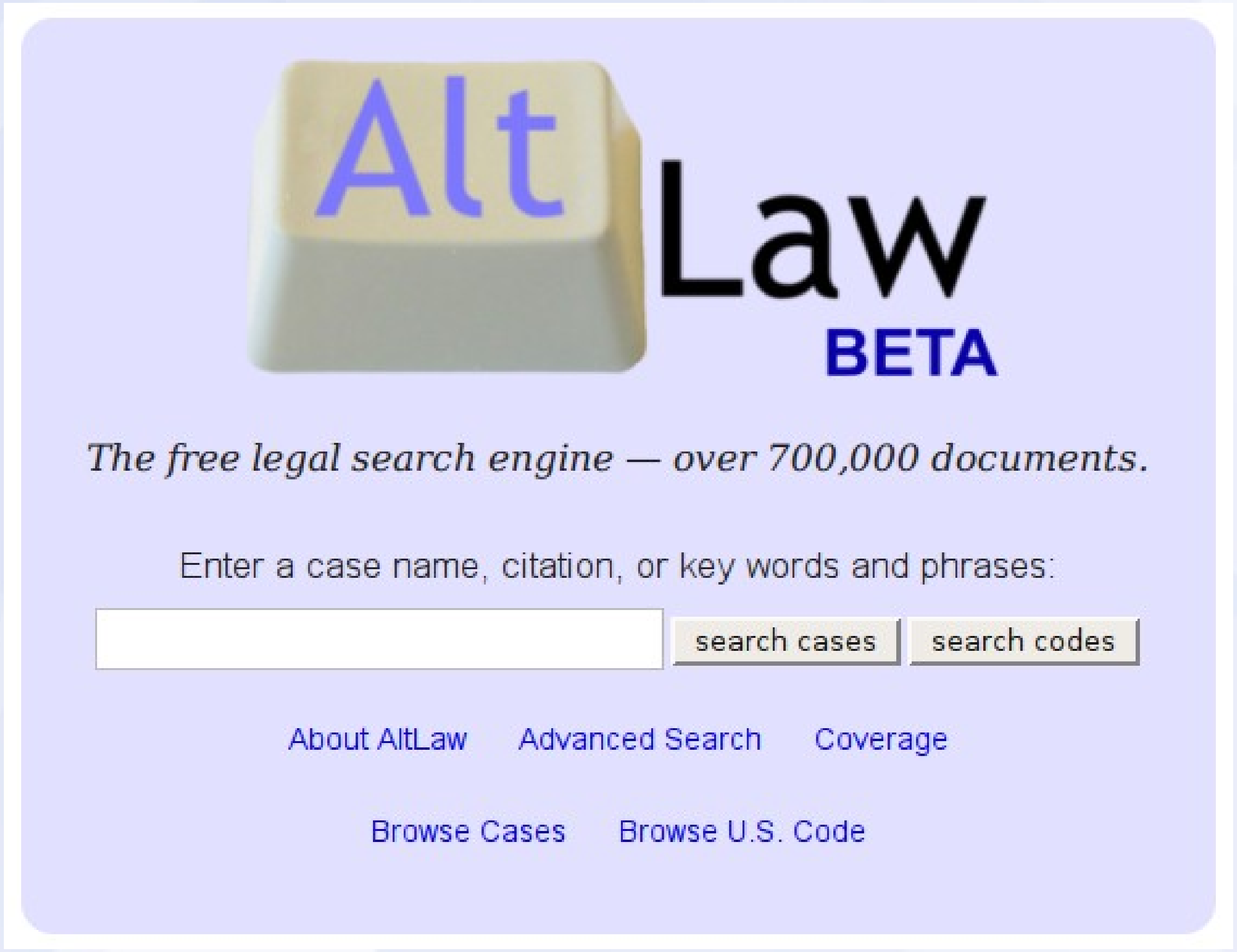
Clojure and Modularity

Philly Lambda
Tuesday, July 21, 2009

Stuart Sierra
http://stuartsierra.com/

www.altlaw.org



The free legal search engine — over 700,000 documents.

Enter a case name, citation, or key words and phrases:

Clojure's Four Elements

List	`(print :hello "Philly")`
Vector	`[:eat "Pie" 3.14159]`
Map	`{:lisp 1  "The Rest" 0}`
Set	`#{2 1 3 5 "Eureka"}`

defn

```
(defn greet [name]
  (println "Hello," name))

(defn average [& args]
  (/ (reduce + args) (count args)))
```

Data are Functions

```
({:f "foo" :b "bar"} :f)
"foo"

(:key {:key "value", :x "y"})
"value"

([:a :b :c] 2)
:c

(#{1 5 3} 3)
true
```

defmacro

```
(defmacro when [test & body]
  (list 'if test (cons 'do body)))
```

Java

```
(import '(com.example.package
             MyClass YourClass))

(.method object argument)

(MyClass/staticMethod argument)

(MyClass. argument)
```

	synchronous	asynchronous
coordinated	`ref`	✗
independent	`atom`	`agent`
unshared	`var`	✗

Vars

```
(def life 42)

(defn meaning [] (println life))

(meaning)
42

(binding [life 37]
  (meaning))
37
```

```
(let [life 37]
  (println life)
  (meaning))
37
42
```

Refs

```
(def c (ref 100))

(deref c)
100

(dosync (alter c inc))

(deref c)
101
```

Agents

```
(def fib (agent [1 1 2]))

(deref fib)
[1 1 2]

(send fib conj 3 5)
```

returns immediately!

later on...

```
(deref fib)
[1 1 2 3 5]
```

clojure.contrib.http.agent

```
(http-agent
    "http://www.altlaw.org/"
    :on-success
        (fn [a]
            (println
                (response-body-str a))))
```

Multimethods

```
(defmulti copy
   (fn [in out]
     [(class in) (class out)]))
```

Multimethods

```
(defmethod copy [InputStream OutputStream] ..
(defmethod copy [InputStream Writer] ...
(defmethod copy [InputStream File] ...
(defmethod copy [Reader OutputStream] ...
(defmethod copy [Reader Writer] ...
(defmethod copy [Reader File] ...
(defmethod copy [File OutputStream] ...
(defmethod copy [File Writer] ...
(defmethod copy [File File] ...
```

Namespaces

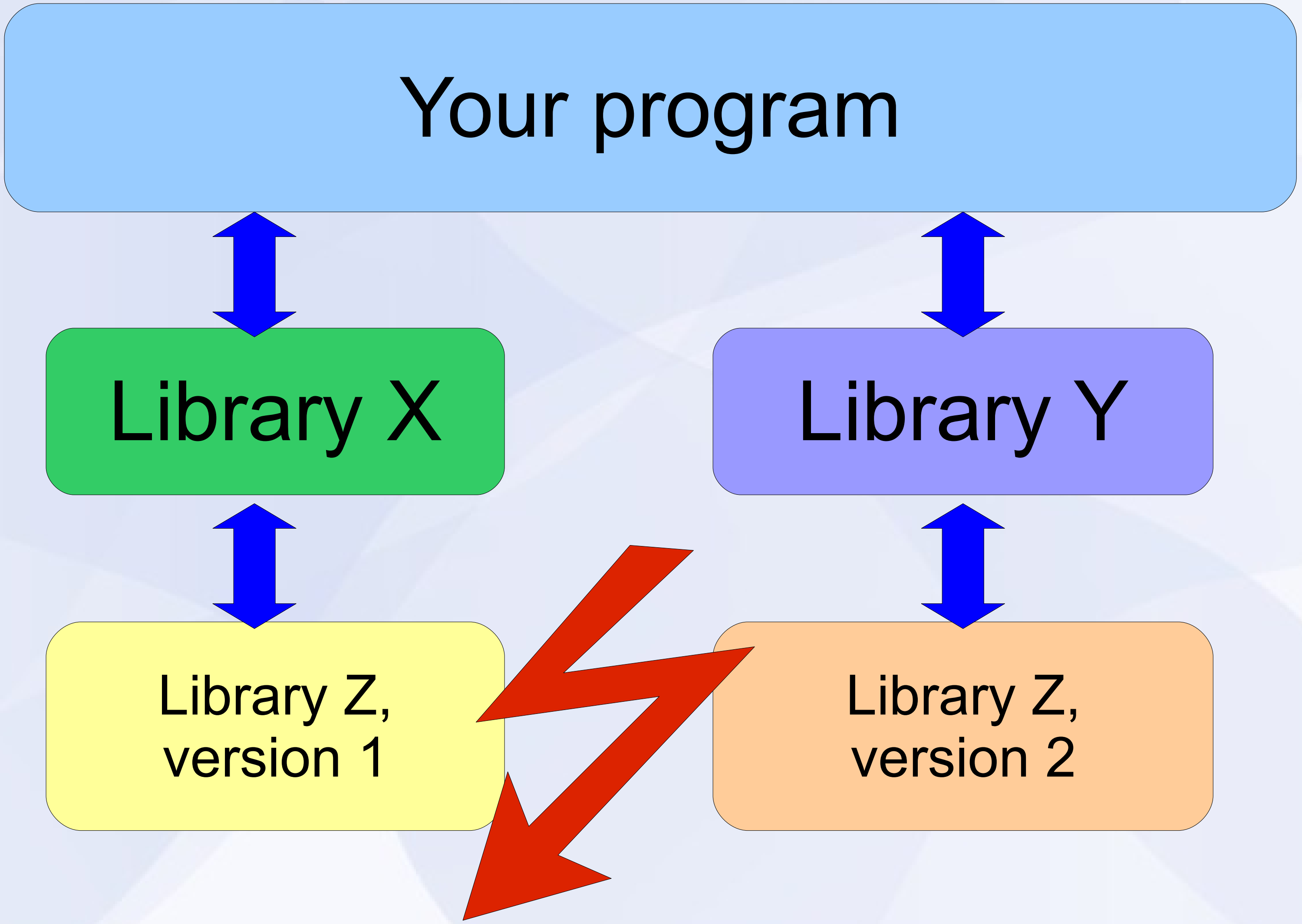
```
(ns my.cool.project
  (:require [clojure.contrib.math :as math])
  (:import (java.math BigDecimal)))

(defn lower-median [x y]
  (math/floor (/ x y))
```

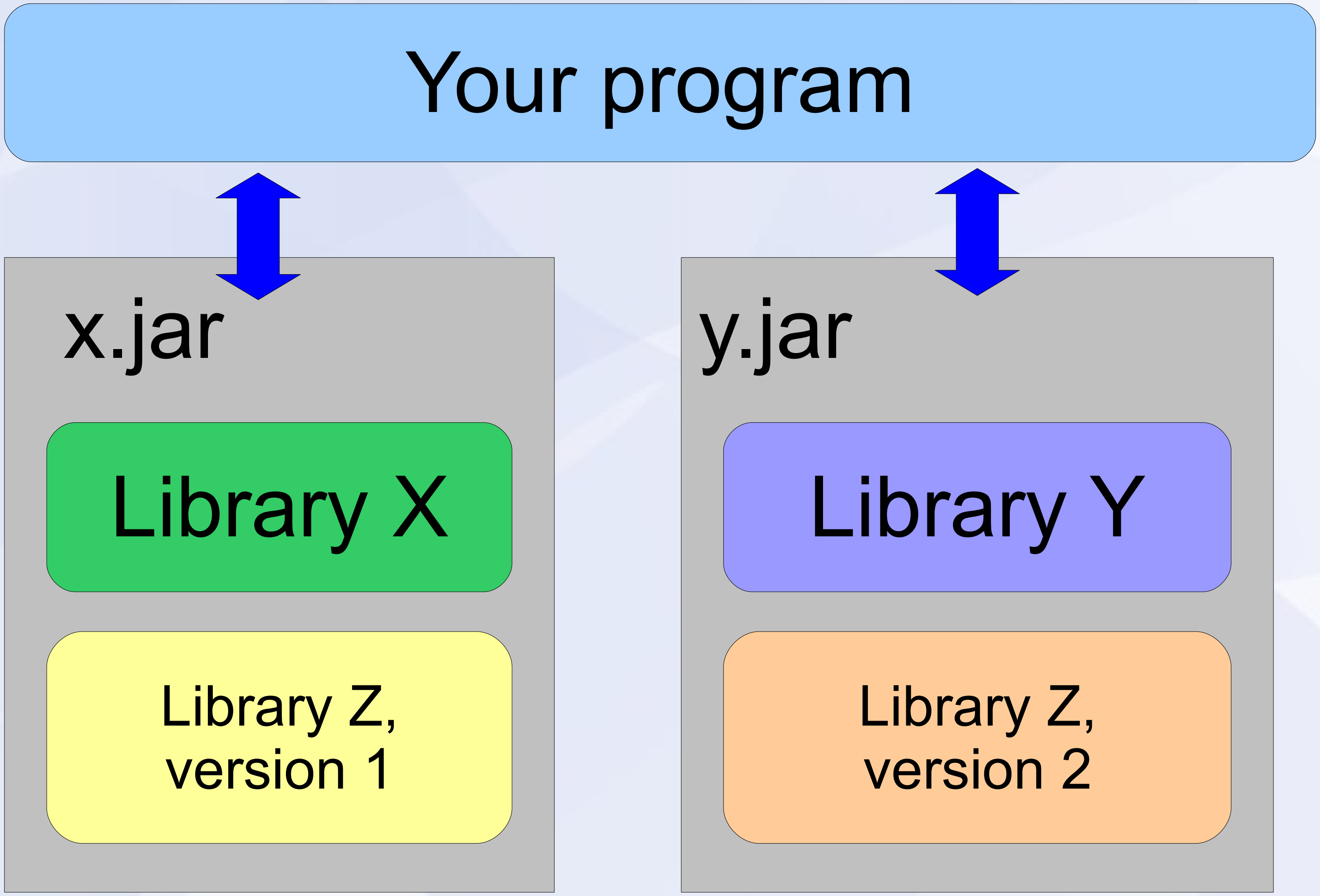
Modularity and Dependency Management

- CPAN
- Python Eggs
- Rubygems
- PEAR (PHP)
- OSGi
- Maven
- Ivy

Modularity


Your program
Library X
Library Y
Library Z, version 1
Library Z, version 2

"JAR hell"


Your program
x.jar
Library X
Library Z, version 1
y.jar
Library Y
Library Z, version 2

CPAN

- System-wide or per-user library installation
- User manages libraries
- One repository, many mirrors
- Multiple versions of a lib may be installed; each process may only use one version
- Integrated docs, tests, & bug tracker

Rubygems

- System-wide or per-user library installation
- User manages libraries
- Multiple repositories, names may conflict
- Multiple versions of a lib may be installed; each process may only use one version
- Docs, tests, and bug tracking not integrated

ASDF

- System-wide or per-user library installation
- User manages libraries
- Wiki page acts as the repository!
- No integrated docs/tests/bug-tracking
- Does not support multiple versions of the same lib

Maven / Ivy

- Per-project library installation
- Build system manages libraries; user manages private repository
- Multiple public repositories
- Optional integration with docs/tests
- Permits multiple versions of a same lib, must be handled by a framework

OSGi

- Java EE, Glassfish, Eclipse
- Bundle: JAR file + extra manifest headers
- Each Bundle gets its own ClassLoader
- Multiple, nested ClassLoader contexts within a single JVM

More

- http://clojure.org/
- Google Groups: Clojure
- #clojure on irc.freenode.net
- http://github.com/richhickey/clojure-contrib
- http://stuartsierra.com/
- http://github.com/stuartsierra
- http://www.altlaw.org/