Java => Language

- Usually the trickier direction
  - Language dispatch
  - Type system mismatches
- Multiple alternatives

Use an API

- JSR223 (Java scripting API)
- Native API

JSR 223 (Java Scripting)

```
import javax.script.*;

public class CalcMain {
    public static void main(String[] args) throws Exception {
        ScriptEngineManager factory = new ScriptEngineManager();
        ScriptEngine engine = factory.getEngineByName("groovy");

        System.out.println(engine.eval("(1..10).sum()"));
    }
}
```

JSR 223 ...

```
ScriptEngineManager factory = new ScriptEngineManager();
ScriptEngine engine = factory.getEngineByName("groovy");

engine.eval("def fact(n) { n == 1 ? 1 : n * fact(n - 1) }");

Invocable inv = (Invocable) engine;
Object result = inv.invokeFunction("fact", 5);
System.out.println(result);
```

Also invokeMethod(obj, name, ...args)

JSR223 Benefits

- Can allow multi-language pluggability
- Learn a single API
- Polyglot boundary obvious

JSR223 Detriments

- Not very transparent
- Lowest Common Denominator API

Native Embedding API

- Benefits
  - Tighter integration with language
  - Fewer setup issues
- Detriments
  - Language-specific
  - Deeper knowledge of lang runtime req'd

Generate Class Files

- Java obviously likes .class format
- Most languages can generate them
  - Some always generate them (Groovy, Scala)
  - But there may still be mismatches

Clojure's gen-class

```
(ns some.Example
  (:gen-class))

(defn -toString
  [this]
  "Hello, World!")
```

```
(ns some.Example
  (:gen-class
     :implements
[clojure.lang.IDeref]))

(defn -deref
  [this]
  "Hello, World!")
```

Clojure's gen-class

```
(ns clojure.examples.instance
    (:gen-class
     :implements [java.util.Iterator]
     :init init
     :constructors {[String] []}
     :state state))

(defn -init [s]
  [[] (ref {:s s :index 0})])

(defn -hasNext [this]
  (let [{:keys [s index]} @(.state this)]
    (< index (count s))))

(defn -next [this]
  (let [{:keys [s index]} @(.state this)
        ch (.charAt s index)]
    (dosync (alter (.state this) assoc :index (inc index)))
    ch))
```

JRuby's jrubyc --java

```
class MyRunnable
  java_implements "java.lang.Runnable"

  java_signature "void run()"
  def run
    puts 'here'
  end

  java_signature "void main(String[])"
  def self.main(args)
    t = java.lang.Thread.new(MyRunnable.new)
    t.start
  end
end
```

Runtime Interface Impl

- Sometimes an easier fit
  - JRuby, Clojure don't need precompile
    - No signatures needed in JRuby
- More "poly" friendly
  - Interface is a clear contract
  - Hide details of lang, runtime, classloading

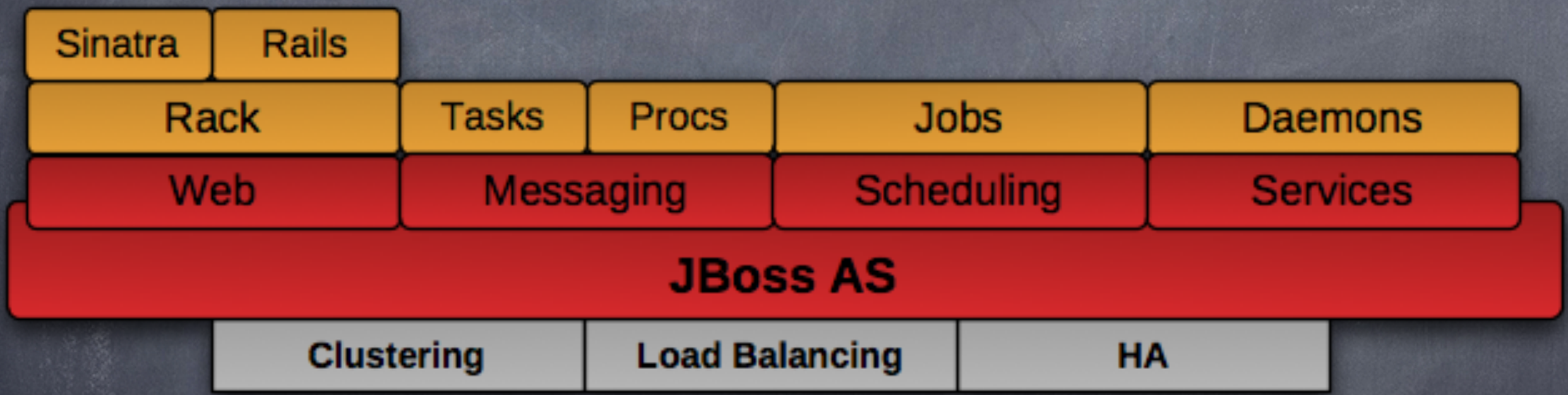
```
java_import java.util.Comparator
java_import java.util.Collections

class CompareStrings
  include Comparator

  def compare(o1, o2)
    o1 <=> o2
  end
end

Collections.sort(some_list, CompareStrings.new)
Collections.sort(some_list) {|o1, o2| o1 <=> o2}
```

Platform Native

- Some servers, etc support langs directly
  - Torquebox = JBoss + Ruby/Rails
  - Immutant = JBoss + Clojure
  - Grails = Spring + Groovy
- Integration issues "solved" for you

TorqueBox

- Enterprise services for Ruby
  - JBoss backend
  - Ruby DSL/frontend
- Fits Ruby sensibilities
- Provides services Rubyists use


Sinatra
Rails
Rack
Tasks
Procs
Jobs
Daemons
Web
Messaging
Scheduling
Services
JBoss AS
Clustering
Load Balancing
HA

Domain-friendly Config

```
TorqueBox.configure do
  ruby do
    version "1.9"
    compile_mode "off"
    debug false
    interactive true
    profile_api true
  end
end
```

```
ruby:
  version: 1.9
  compile_mode: off
  debug: false
  interactive: true
  profile_api: true
```

Messaging

```
TorqueBox.configure do
  ...
  queue '/queues/my_app_queue'
  topic '/queues/my_app_topic'
end
```

```
application:
  ..
queues:
  /queues/my_app_queue:

topics:
  /queues/my_app_topic:
```

Messaging

```
queue = fetch('/queues/foo')
queue.publish "A text message"

topic = fetch('/topics/foo')
topic.publish "A text message"
```

```
queue = TorqueBox::Messaging::Queue.new('/queues/foo')
message = queue.receive

topic = TorqueBox::Messaging::Topic.new('/topics/foo')
message = topic.receive
```

immutant

- Same thing but for Clojure
- Clojure sensibilities

Code is Config!

```
(defproject my-app "1.2.3"
            :dependencies [[org.clojure/clojure "1.3.0"]
                           [noir "1.2.0"]]
            :immutant {:init my-app.core/initialize
                       :resolve-dependencies true
                       :lein-profiles [:dev :clj15]
                       :context-path "/"
                       :virtual-host "foo.host"
                       :swank-port 4111
                       :nrepl-port 4112})
```

```clojure
(ns my-app.init
  (:require [immutant.daemons    :as daemons]
            [immutant.jobs       :as jobs]
            [immutant.messaging  :as messaging]
            [immutant.web        :as web]
            [immutant.repl       :as repl]
            [immutant.utilities  :as util]
            [noir.server         :as server]
            [my-app.core         :as core]))

;; point noir to the right place for views
(server/load-views (util/app-relative "src/my_app/views"))

;; start a web endpoint
(web/start "/" (server/gen-handler {:mode :dev :ns 'my-app}))

;; spin up a repl
(repl/start-swank 4321)

;; schedule a job
(jobs/schedule "my-job" "*/5 * * * * ?" core/process-tps-reports)

;; start a daemon
(deamons/start "my-daemon" core/daemon-start core/daemon-stop)

;; create a queue
(messaging/start "/queue/foo")
```