Building things you can take apart

Kevin Lynagh
Keming Labs

August 2013
Hacker School

Modularity

Systems

resources

machines

languages / runtimes

programs

namespaces / classes

methods / functions

Humans

WEATHERTRON

SIMPLE AS

C

Java

JavaScript

↳iOS

Sounds like you need a...

build tool

What's a build tool?

A build tool:

Runs commands...

in the right order

only when necessary

Lets talk about

Make

Buildin’ stuff since 1977

Lets talk about

Rake

Buildin’ stuff since 2003

Lets talk about

Ant

Buildin’ stuff since 2000

Lets talk about

Maven

Downloading the Internet
and building stuff since 2004

Make's interface

Make's interface

```
$ vi Makefile
```

```
$ make
```

Write a Makefile

A Makefile is a list of targets
- what they depend on
- system commands that build them

```
all: hello

hello: main.o factorial.o hello.o
	g++ main.o factorial.o hello.o -o hello

main.o: main.cpp
	g++ -c main.cpp

factorial.o: factorial.cpp
	g++ -c factorial.cpp

hello.o: hello.cpp
	g++ -c hello.cpp

clean:
	rm -rf *o hello
```

Run make

```
$ make
```

Modularity

Make's interface

```
$ vi Makefile
```

```
$ make
```

Make's assumptions

All dependencies can be known in advance

Everything can be done from the shell

All you want to do is make

All dependencies can be known in advance

```
$ make
```

All dependencies can be known in advance

```
$ ./configure
$ make
```

All dependencies can be known in advance

```
$ autoconf
$ ./configure
$ make
```

Aside

$ wc -l configure

22760 configure

Everything can be done from the shell

Everything can be done from the shell

```
$ time bundle exec ruby -e '1+1'

real    0m1.375s
user    0m1.250s
sys     0m0.084s
```

Everything can be done from the shell

```
$ time java -cp clojure-1.5.1.jar \
       clojure.main -e "(+ 1 1)"
real     0m1.352s
user     0m2.028s
sys      0m0.080s
```

Aside

We have 2^{18} moar transistors since make was first released

All you want to do is make

All you want to do is make

What's the dependency graph?
What depends on X?
What's queued to build?
What are you building now?

Make's assumptions

All dependencies can be known in advance

Everything can be done from the shell

All you want to do is make

Revise assumptions

Dependencies cannot all be known in advance

Not everything can be done from the shell

Do more than just make

Modularity

Making

make

more **modular**

Dependencies cannot all be known in advance.

They must be **discovered.**
(by a program)

```
all: hello

hello: main.o factorial.o hello.o
	g++ main.o factorial.o hello.o -o hello

main.o: main.cpp
	g++ -c main.cpp

factorial.o: factorial.cpp
	g++ -c factorial.cpp

hello.o: hello.cpp
	g++ -c hello.cpp

clean:
	rm -rf *o hello
```

```
[{"target": "all", "deps": ["hello"]},

 {"target": "hello",
  "deps": ["main.o","factorial.o","hello.o"],
  "cmd": "g++ main.o factorial.o hello.o -o hello"},

 {"target": "main.o",
  "deps": ["main.cpp"],
  "cmd": "g++ -c main.cpp"},

{"target": "factorial.o",
 "deps": ["factorial.cpp"],
 "cmd": "g++ -c factorial.cpp"},

{"target": "hello.o",
 "deps": ["hello.cpp"],
 "cmd": "g++ -c hello.cpp"},

{"target": "clean",
 "deps": [],
```

```
<?xml version="1.0" encoding="UTF-8"?>
<targets>
  <target>
    <deps>
      <target>hello</target>
    </deps>
    <name>all</name>
  </target>
  <target>
    <cmd>g++ main.o factorial.o hello.o -o hello</cmd>
    <deps>
      <target>main.o</target>
      <target>factorial.o</target>
      <target>hello.o</target>
    </deps>
    <name>hello</name>
</target>
<target>
  <cmd>g++ -c main.cpp</cmd>
  <deps>
    <target>main.cpp</target>
```

Your programs should interface with programs, not people.

(You can write a second program to interface with people)

Aside

SQL

Do we even need

Makefiles

Do we even need

Rakefiles

Do we even need

build.xml

Do we even need

pom.xml

No Makefile?

How you know targets?

How you know their dependencies?

redo

https://github.com/apenwarr/redo

(based on design by DJB)

Targets by convention

$ redo public/thingy/style.css

Runs script at:

public/thingy/style.css.do
public/thingy/default.css.do
public/default.css.do
default.css.do

Scripts register dependencies

public/thingy/default.css.do

```
#!/usr/bin/env ruby
require 'rubygems'; require 'sass'

src = "src/sass/" + ARGV[1] + ".sass"
engine = Sass::Engine.for_file(src)
$stdout.puts engine.render
`redo-ifchange #{src}`
```

Scripts register dependencies

src/sass/style.sass

```
@import "colors"

body
  margin: 0
  color: $body_color
```

src/sass/_colors.sass

```
$body_color: darkGray
```

Makefile

```
style.css: style.sass _colors.sass
	sass style.sass > style.css
```

Scripts register dependencies

public/thingy/default.css.do

```
#!/usr/bin/env ruby
require 'rubygems'; require 'sass'

src = "src/sass/" + ARGV[1] + ".sass"
engine = Sass::Engine.for_file(src)
$stdout.puts engine.render
deps = engine.dependencies
   .map{|x| x.options[:filename]}
`redo-ifchange #{src} #{deps.join(' ')}`
```

Less is more.

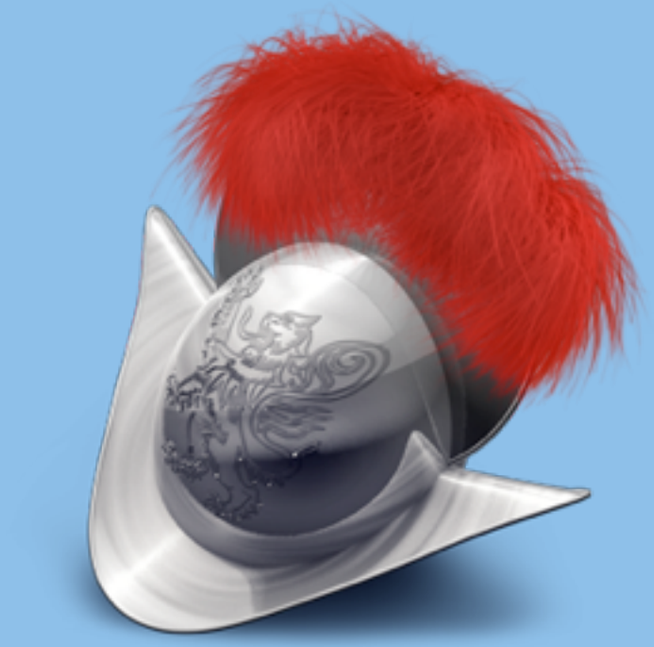
If you think you need to invent a new syntax…maybe you don’t.

Not everything can be done from the shell

Hipster language virtual machine startup time

Distributed builds

Not everything can be done from the shell

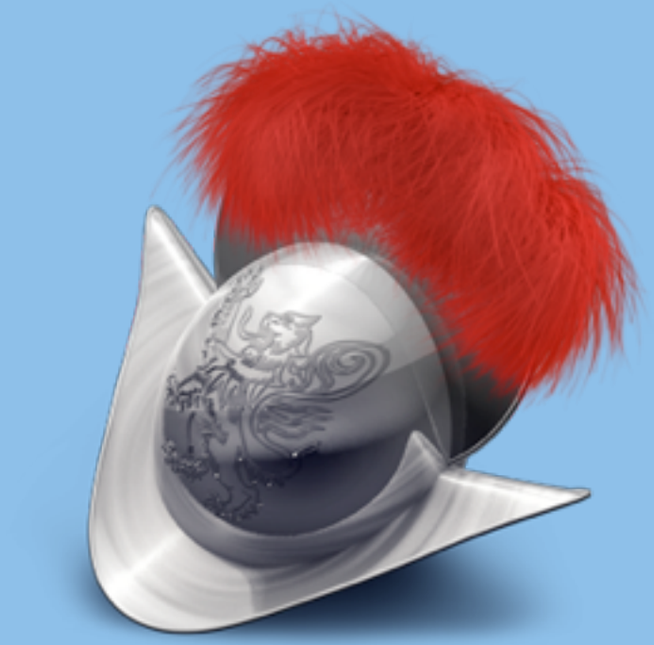


Guard

Grunt

Leiningen

Not everything can be done from the shell



Ruby

JavaScript

Clojure

Not everything can be done from the shell

a single computer

language of the month's VM

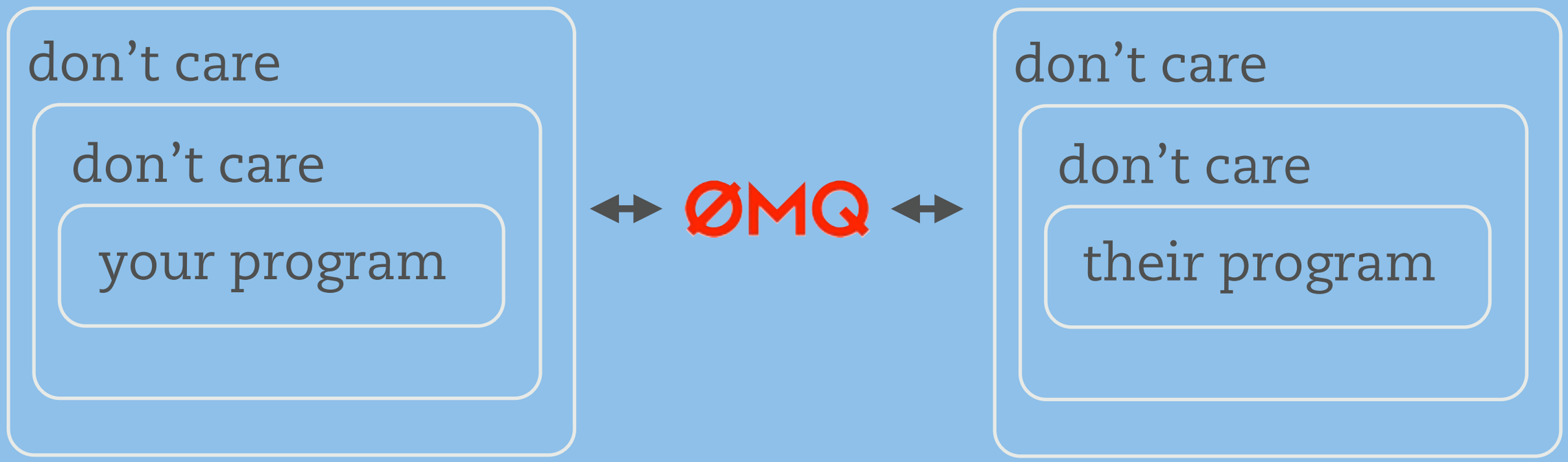
your program ⟷ their program

Not everything can be done
from the shell

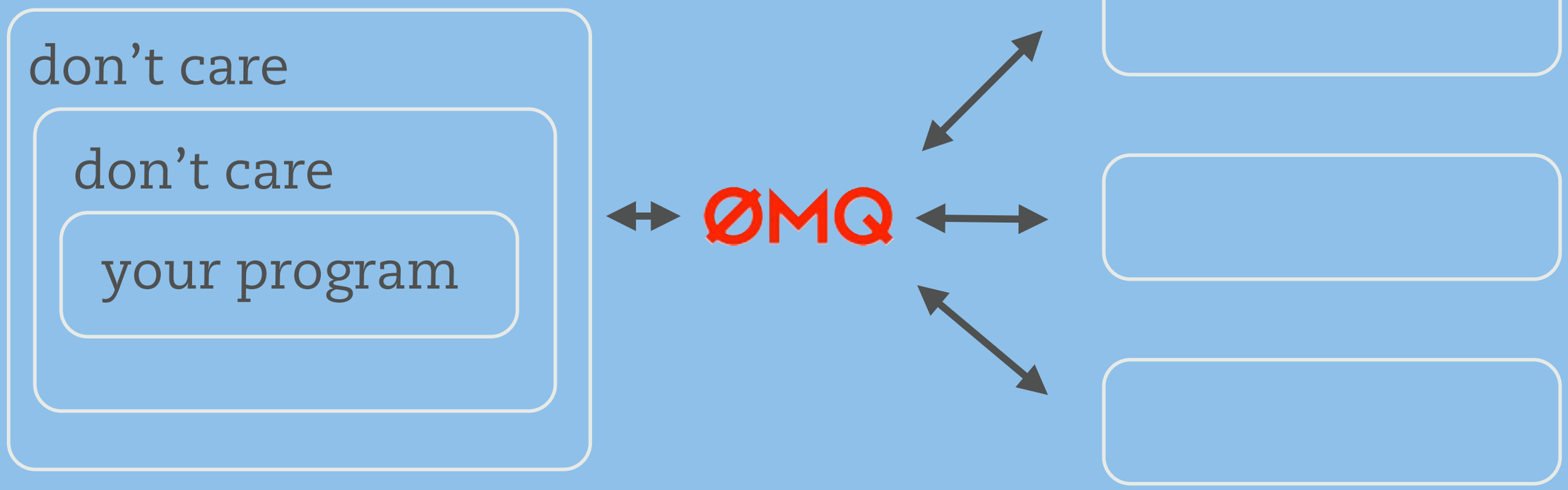
HTTP

redis

Not everything can be done from the shell



Not everything can be done from the shell



Think queues.

Think data.

Do more than just make

What's the dependency graph?
What depends on X?
What's queued to build?
What are you building now?

Do more than just make

Your programs should interface with programs, not people.

Think queues/data

Do more than just make

What's the dependency graph?
What depends on X?

```
$ curl https://localhost:8080/debug/buildgraph
```

↳ JSON

Modularity

Systems

resources

machines

languages / runtimes

programs

namespaces / classes

methods / functions

Humans

Systems

resources

machines

languages / runtimes

programs

namespaces / classes

methods / functions

Humans

Your programs should interface with programs, not people.

Less is more.

Think queues.

Think data.

Building things you can take apart

Kevin Lynagh
Keming Labs

August 2013
Hacker School