Make your world smarter.

Ryan Applegate

Getting Groovy

with

Who am I

- Ryan Applegate
- Senior Software Engineer @ SmartThings
- @rappleg on Twitter and GitHub

Vert.x Agenda

Background (Inspired by Node.js)

Why Vert.x?

Benchmarks (How does Vert.x stack up?)

Terminology and examples

Demo 1 – Websockets

How does SmartThings use Vert.x?

Demo 2 – SmartThings Web IDE

What's new in Vert.x 3

What is Node?

Server Side Javascript

Event Driven Non-Blocking I/O

Single thread/single event loop

Application registers handlers

Events trigger handlers

Everything runs on the event loop

Reactor Pattern Issues

- MUST not block the event loop
- Some work is naturally blocking
  - Intensive data crunching
  - 3rd-party blocking API's (e.g. JDBC, etc...)
  - Node.js is not good for this type of work

Why Vert.x?

- Same event-driven non-blocking IO programming model as Node
- Polyglot (Groovy, Ruby, Java, Javascript, Python, Scala, and Clojure)
- Mature concurrency framework (JVM)
- Hazelcast for Clustering
- Interprocess Communication via Event Bus
- Built on Netty and NIO2 for Network I/O

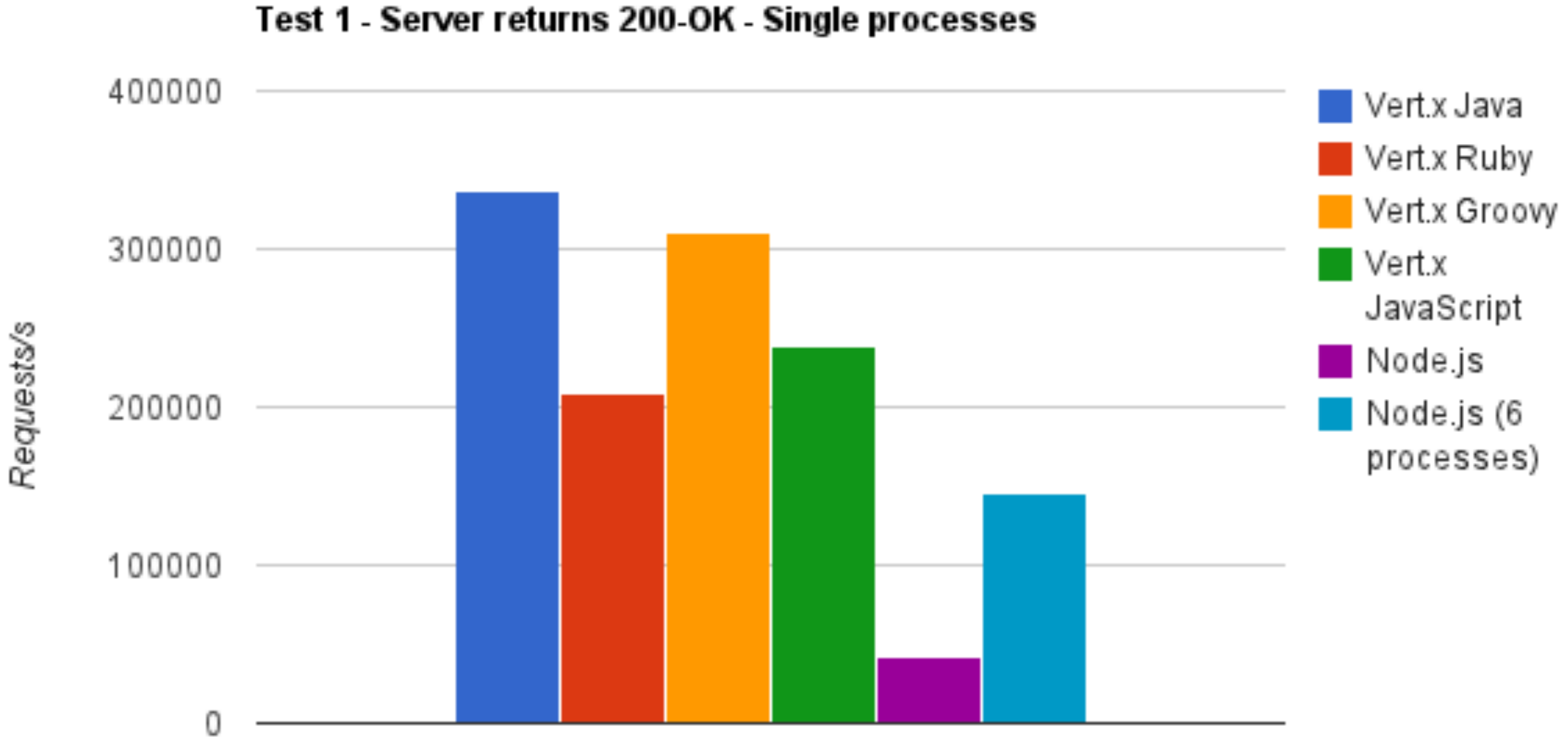
Ideal choice for creating *microservices*.

Lightweight - Vert.x core is around 650kB in size.

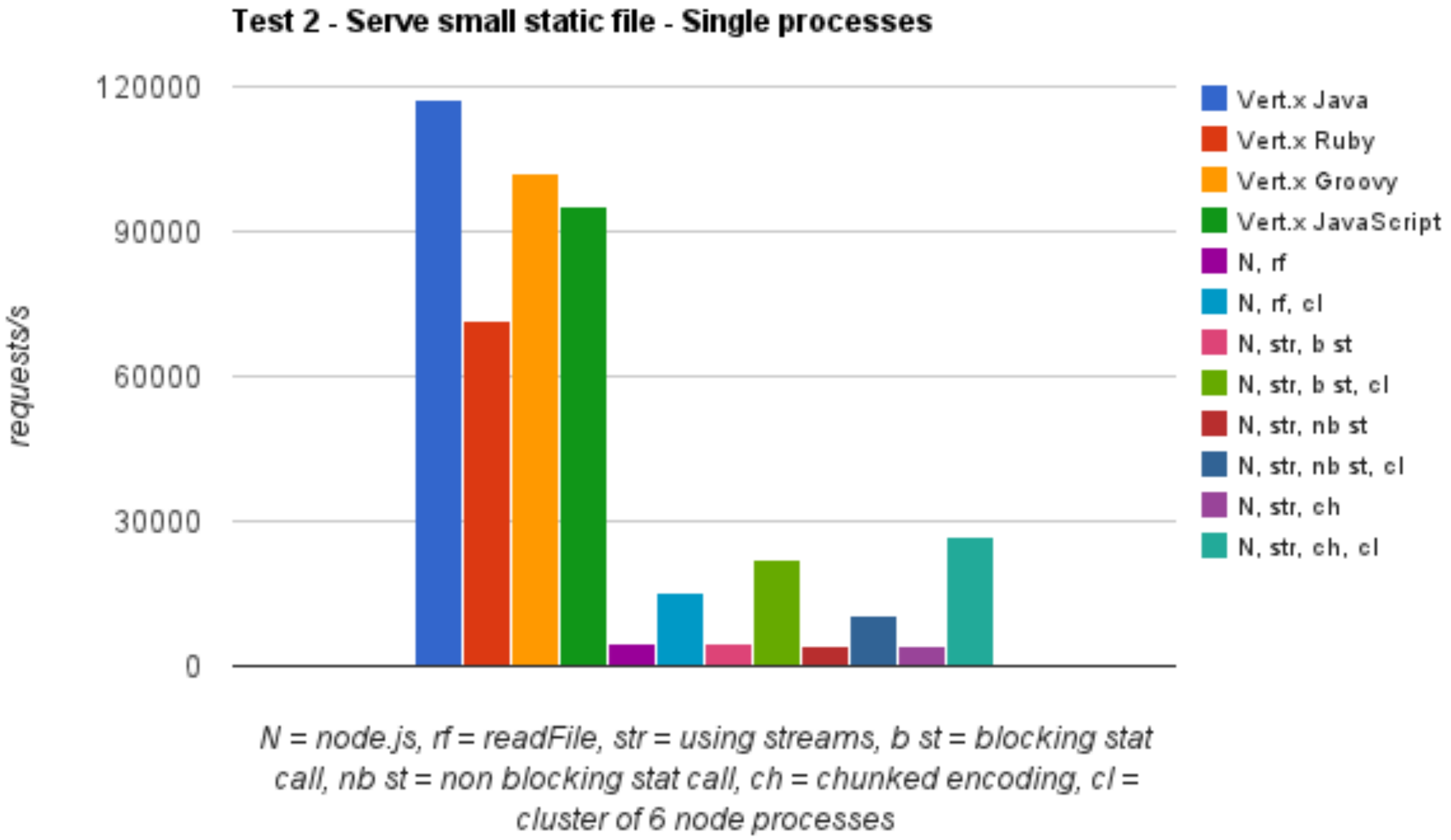
Fast - We'll take a look at some independent benchmarks

It's not an application server - There's no monolithic Vert.x instance into which you deploy applications. You just run your apps wherever you want to.

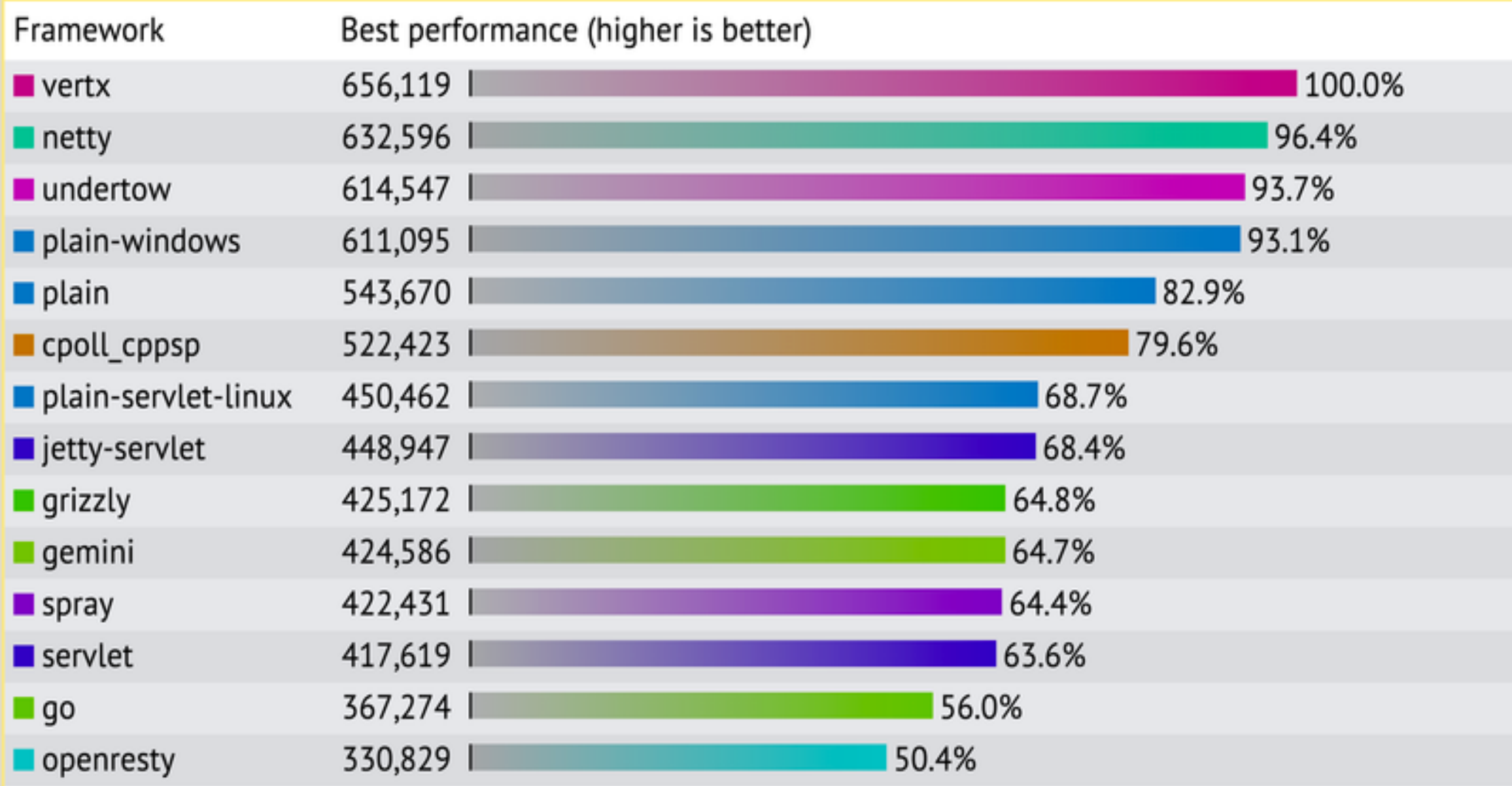
Modular - when you need more bits just add the bits you need and nothing more.


Test 1 - Server returns 200-OK - Single processes
Requests/s
400000
300000
200000
100000
0
Vert.x Java
Vert.x Ruby
Vert.x Groovy
Vert.x JavaScript
Node.js
Node.js (6 processes)

Benchmark #1

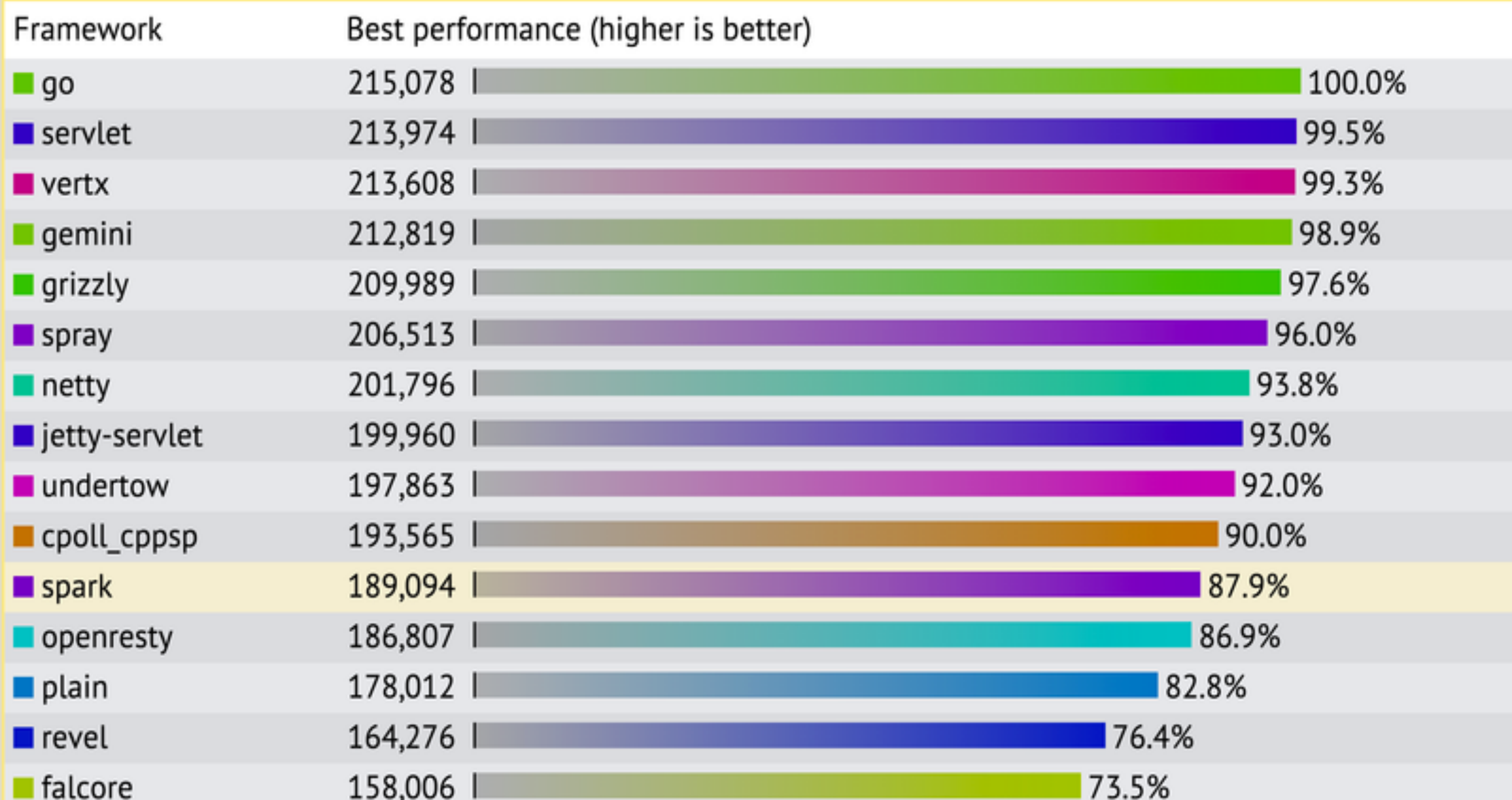

Test 2 - Serve small static file - Single processes
120000
90000
60000
30000
0
requests/s
Vert.x Java
Vert.x Ruby
Vert.x Groovy
Vert.x JavaScript
N, rf
N, rf, cl
N, str, b st
N, str, b st, cl
N, str, nb st
N, str, nb st, cl
N, str, ch
N, str, ch, cl
N = node.js, rf = readFile, str = using streams, b st = blocking stat call, nb st = non blocking stat call, ch = chunked encoding, cl = cluster of 6 node processes

Best plaintext responses per second, i7-2600K hardware (55 tests)



Framework	Best performance (higher is better)	
vertx	656,119	100.0%
netty	632,596	96.4%
undertow	614,547	93.7%
plain-windows	611,095	93.1%
plain	543,670	82.9%
cpoll_cppsp	522,423	79.6%
plain-servlet-linux	450,462	68.7%
jetty-servlet	448,947	68.4%
grizzly	425,172	64.8%
gemini	424,586	64.7%
spray	422,431	64.4%
servlet	417,619	63.6%
go	367,274	56.0%
openresty	330,829	50.4%

Best JSON responses per second, i7-2600K hardware (102 tests)



Framework	Best performance (higher is better)	
go	215,078	100.0%
servlet	213,974	99.5%
vertx	213,608	99.3%
gemini	212,819	98.9%
grizzly	209,989	97.6%
spray	206,513	96.0%
netty	201,796	93.8%
jetty-servlet	199,960	93.0%
undertow	197,863	92.0%
cpoll_cppsp	193,565	90.0%
spark	189,094	87.9%
openresty	186,807	86.9%
plain	178,012	82.8%
revel	164,276	76.4%
falcore	158,006	73.5%

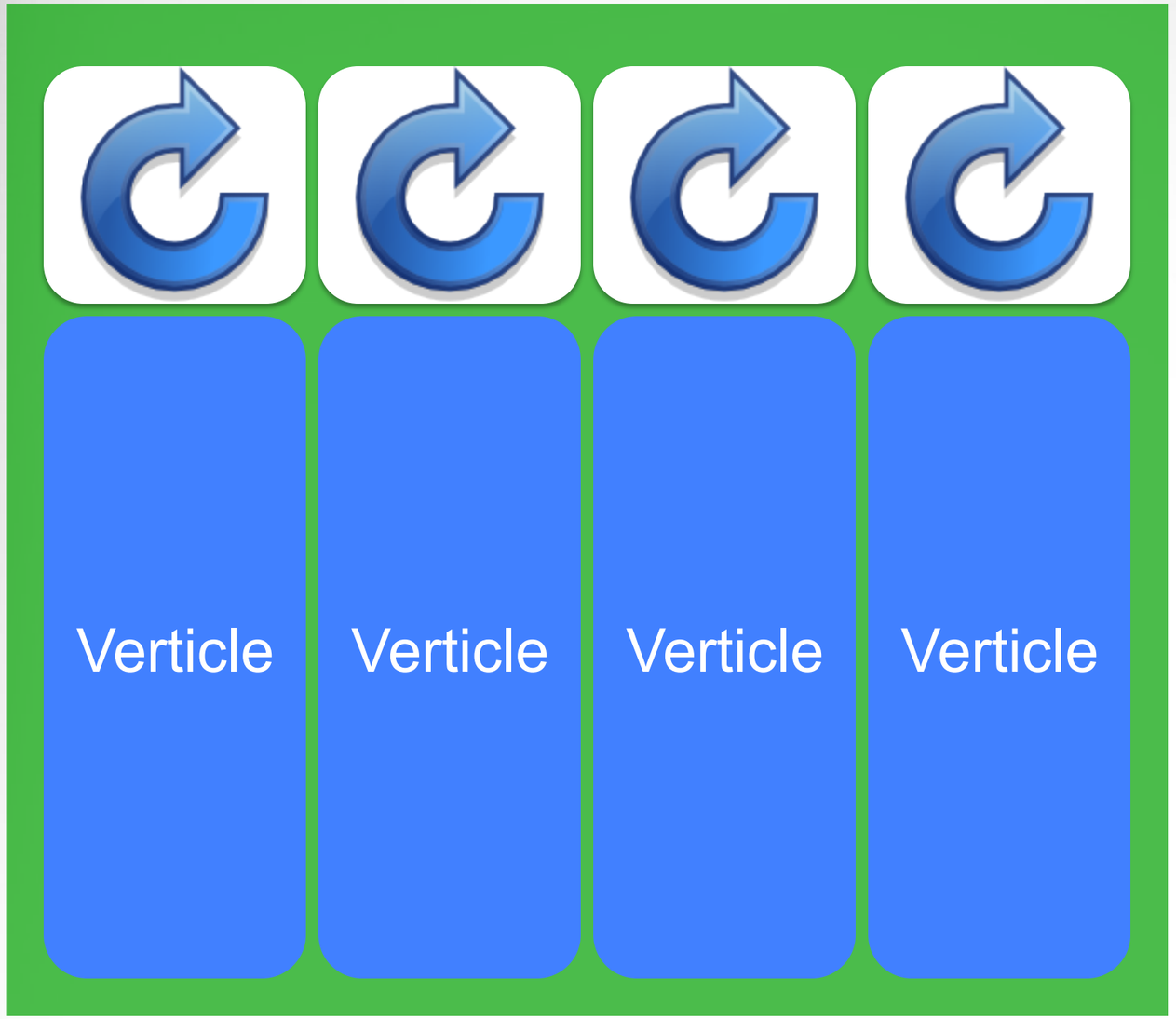
VERT.X

Verticle

The unit of deployment in vert.x is called a verticle (think of a particle, for vert.x). Verticles can currently be written in Java, JavaScript, Ruby, Python, Groovy, Clojure, and Scala.

A verticle is defined by having a main which is just the script (or class in the case of Java) to run to start the verticle.

Vert.x Instance



Event Loops

vertx run HelloWorld -instances 4

SmartThings

Running Vert.x Server

Server.groovy

```
vertx.createHttpServer().requestHandler { req ->
  def file = req.uri == "/" ? "index.html" : req.uri

  req.response.sendFile "webroot/$file"
}.listen(8080)
```

Start the server

```
vertx run Server.groovy
```

Utilize more cores, up your instances...

```
vertx run Server.groovy -instances 32
```

Concurrency

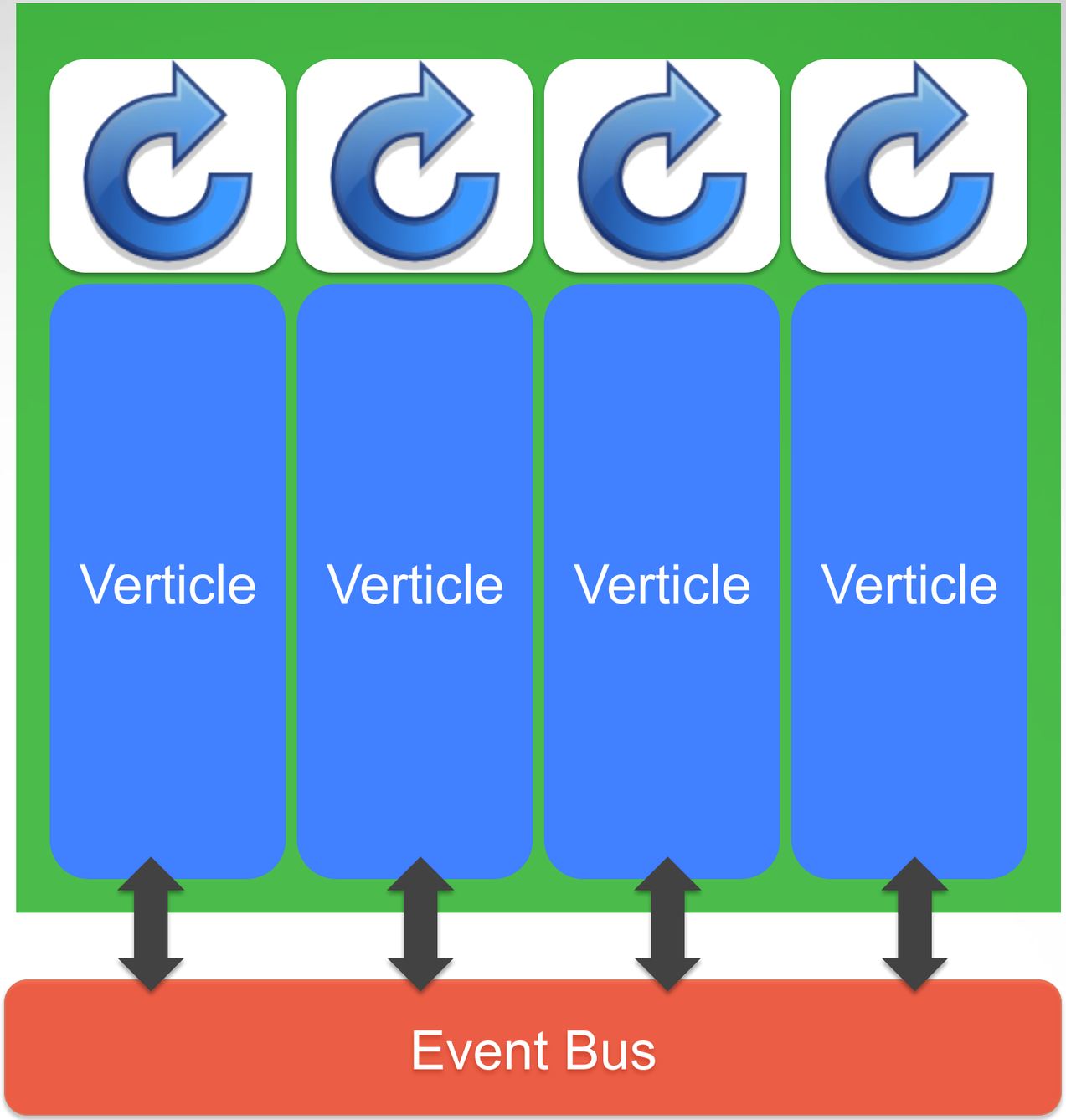
Verticle instance ALWAYS executes on assigned thread/event loop.

Verticles can have isolated classloaders and therefore not share global state.

Write all your code as single threaded.

No more synchronized and volatile!

SmartThings



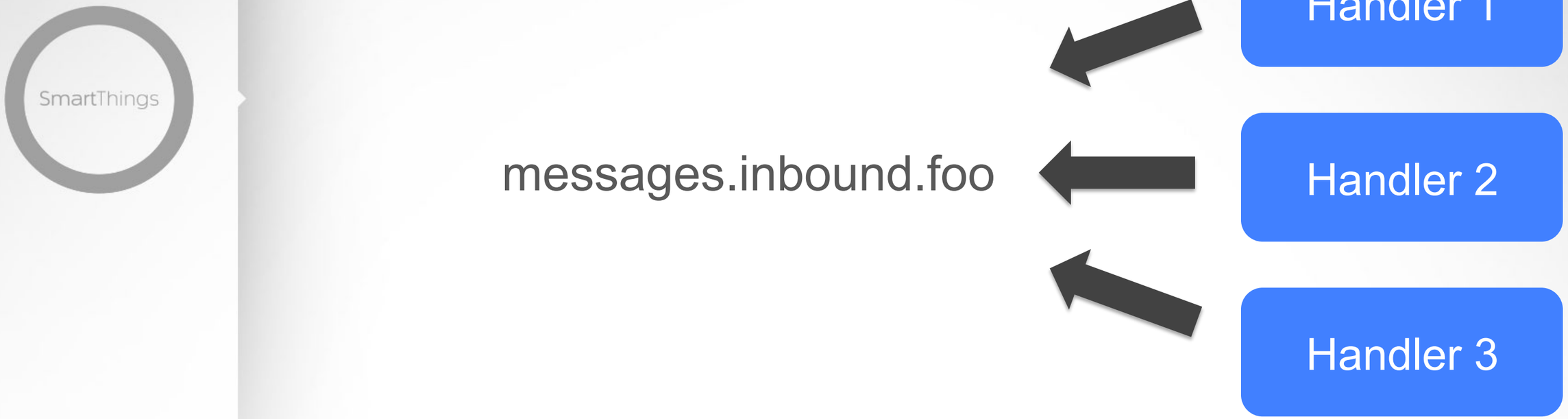
Event Bus Addressing

Address simply a String

Dot-style namespacing recommended

"messages.inbound.foo"

Handler Registration

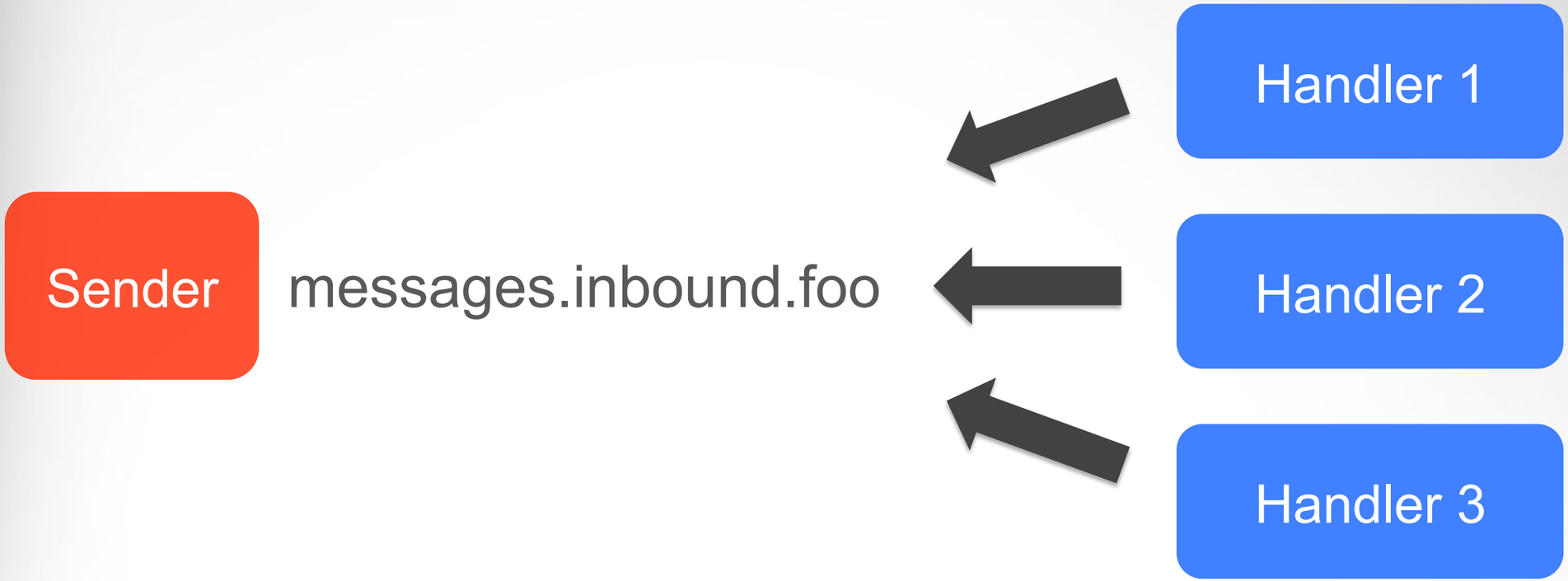


Handler Registration


```
def eb = vertx.eventBus()

eb.registerHandler("test.address") { message ->
  println "I received a message ${message.body}"
}
```

Pub/Sub

Deliver single message to all handlers registered at an address



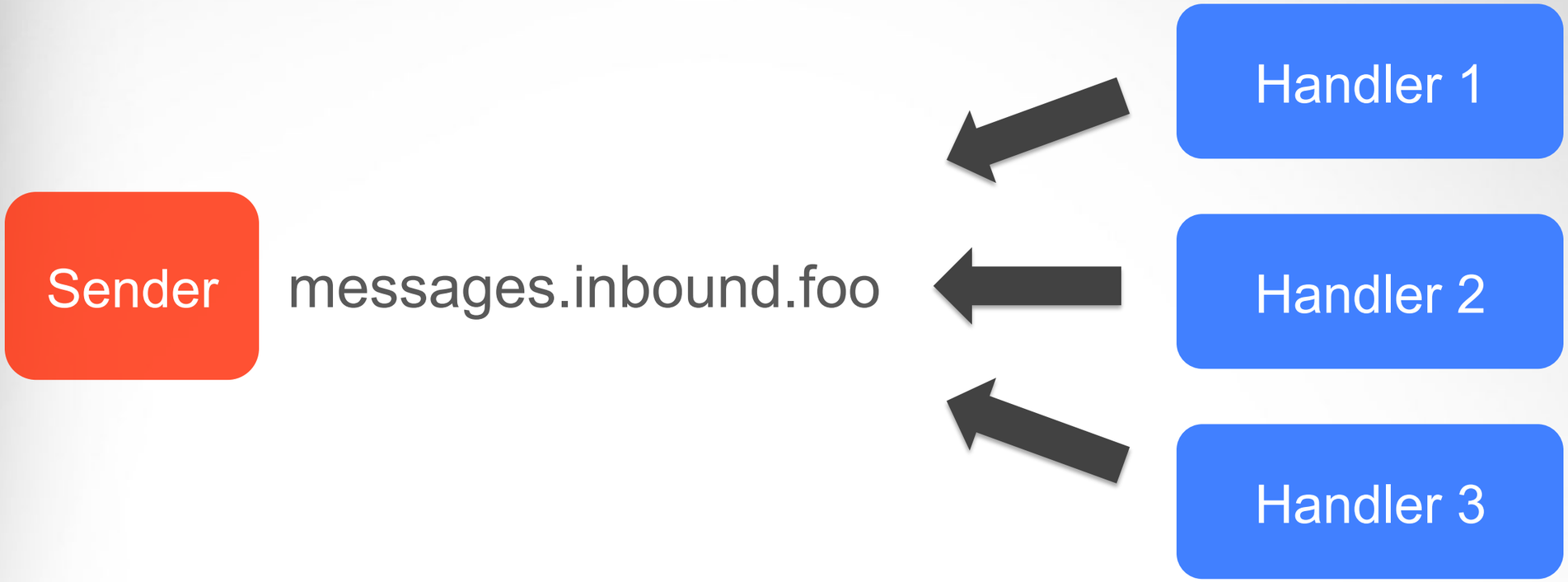
Pub/Sub

Deliver single message to all handlers registered at an address


```
eb.publish("test.address", ”hello world")
```

P2P

Deliver message to only one handler registered at an address



P2P

Deliver message to only one handler registered at an address


```
eb.send("test.address", ”hello world")
```

P2P Messaging Options

Send (Fire and Forget)

Request/Reply Model

Implement replyHandler for messages

Sender

```
eb.send("test.address”, ”Some msg") { message ->
  println "I received a reply ${message.body}"
}
```

Receiver

```
eb.registerHandler("test.address") { message ->
  println "I received a message ${message.body}”

  // Do some work here

  message.reply("test.address”)
}
```

Vert.x in the Browser

Clustered along with Vert.x instances using HazelCast (In-memory data grid)

SockJS - Older browsers/Corp Proxy

Talk to event bus through SockJS Bridge

WebSockets - HTML 5 feature that allows a full duplex between HTTP servers

WebSockets on the Server

```
def server = vertx.createHttpServer()

server.websocketHandler{ ws ->
  println "A websocket has connected!”

}.listen(8080, "localhost")
```

Demo – WebSockets in the Browser

BroChat – Connect and join the gr8conf room to send messages back and forth

Simple chat server example to start up HTTP Server on 8080 and allow messages to be sent back and forth using the event bus and websockets

Vert.x Shared State

Shared Data Object (vertx.sharedData())

ConcurrentMap or Set

Elements MUST be immutable values

Currently only available within a Vertx instance, not across the cluster

Allowed Values

- Strings
- Boxed Primitives
- byte[]
- org.vertx.java.core.buffer.Buffer
- org.vertx.java.core.shareddata.Shareable

Shared Map

Verticle 1

```
def map = vertx.sharedData.getMap('demo.mymap')
map["some-key"] = 123
```

Verticle 2

```
def map = vertx.sharedData.getMap('demo.mymap')
// Retrieve value 123 from the map
def value = map."some-key"
```

Shared Set

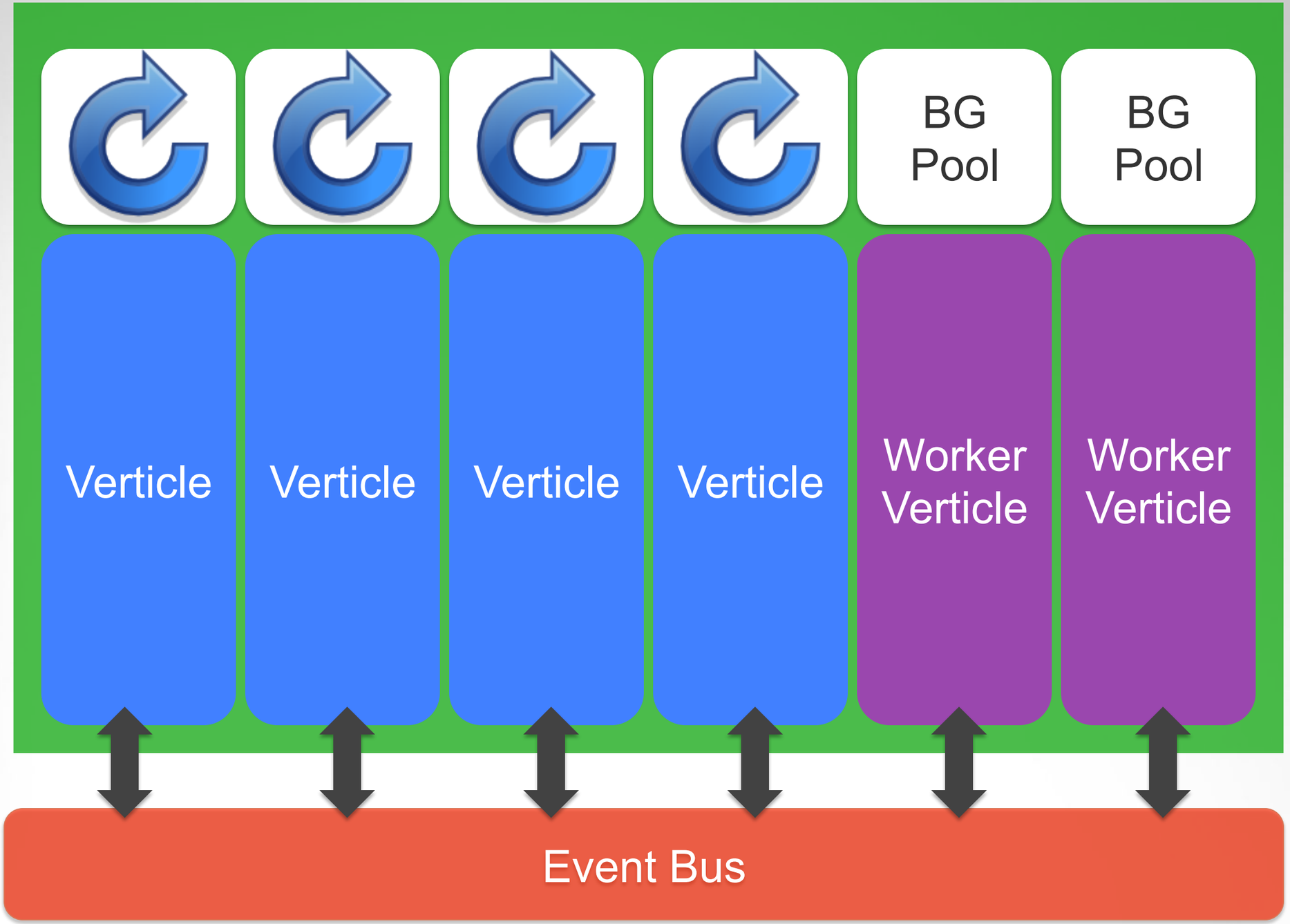
Verticle 1

```
def set = vertx.sharedData.getSet('demo.myset')
set << "some-value"
```

Verticle 2

```
def set = vertx.sharedData.getSet('demo.myset')
// Set will now contain some-value
set.contains("some-value")
```

SmartThings



Worker Verticle Example

```
public class FibWorker extends Verticle {
  @Override
  public void start() {
    def eb = vertx.eventBus()
    eb.registerHandler("fib.request") { message ->
      def result = fib(message.body.intValue())
      def resultMessage = { nbr: message.body,
                            result: result }
      eb.send("fib.response", resultMessage)
    }
  }
  def fib(n) { n < 2 ? 1 : fib(n-1) + fib(n-2) }
}
```

Verticle (Running on Event Loop)


```
public class WorkerExample extends Verticle {
  @Override
  public void start() {
    def eb = vertx.eventBus()
    eb.registerHandler("fib.response") { msg ->
      println "Fib:${msg.body.nbr}=${msg.body.result}"
    }
    container.deployWorkerVerticle("worker.FibWorker")
    { msg ->
      eb.send("fib.request", 20)
    }
  }
}
```

More stuff with Vert.x Core APIs

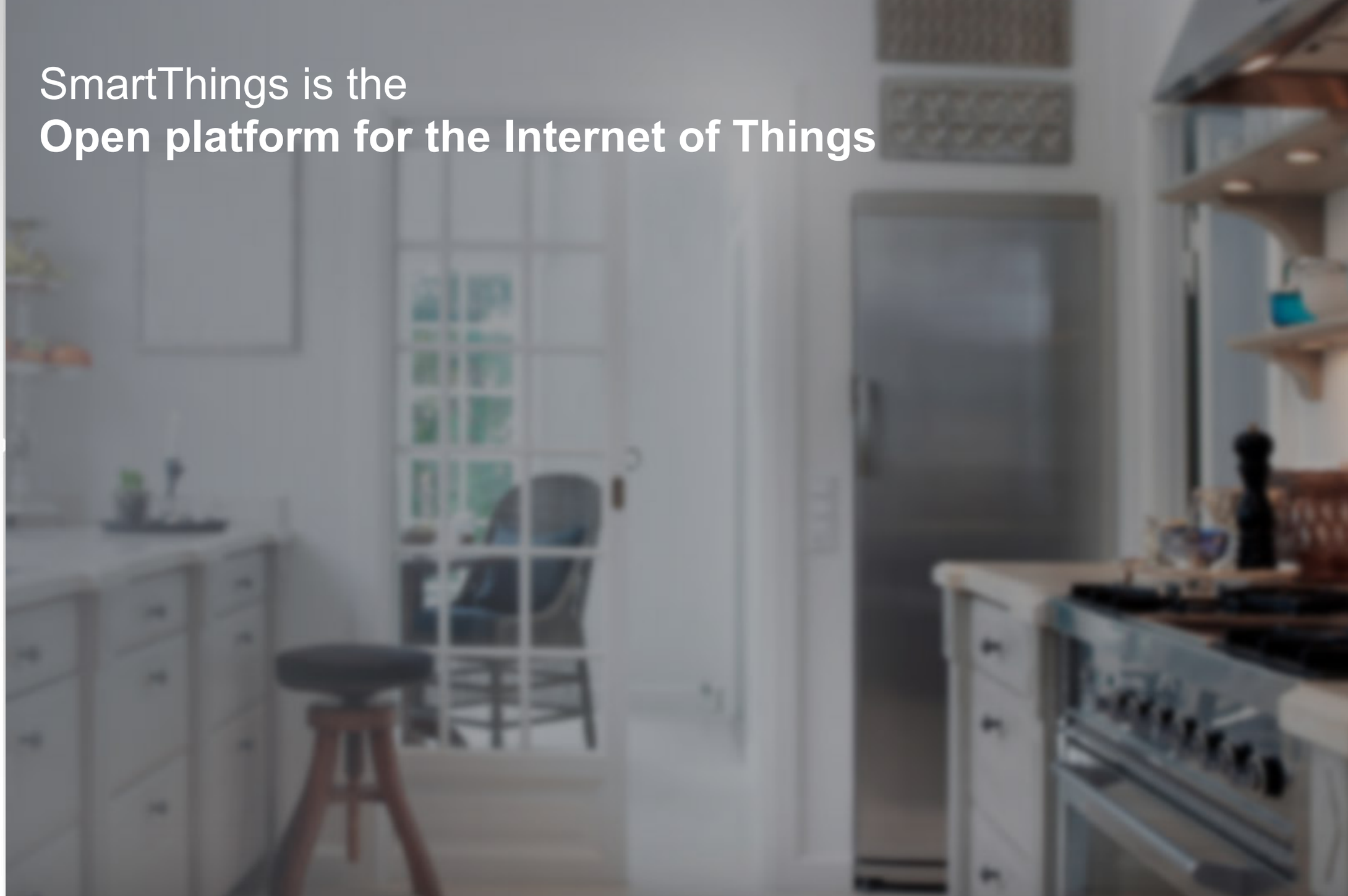
- TCP/SSL servers and clients
- HTTP/HTTPS servers and clients
- WebSockets servers and clients
- Accessing the distributed event bus
- Periodic and one-off timers
- Buffers
- Flow control
- Accessing files on the file system
- Shared map and sets
- Logging
- Accessing configuration
- Writing SockJS servers
- Deploying and undeploying verticles

VERT.X

SmartThings is

Your home in the palm of your hand

SmartThings is the
Open platform for the Internet of Things

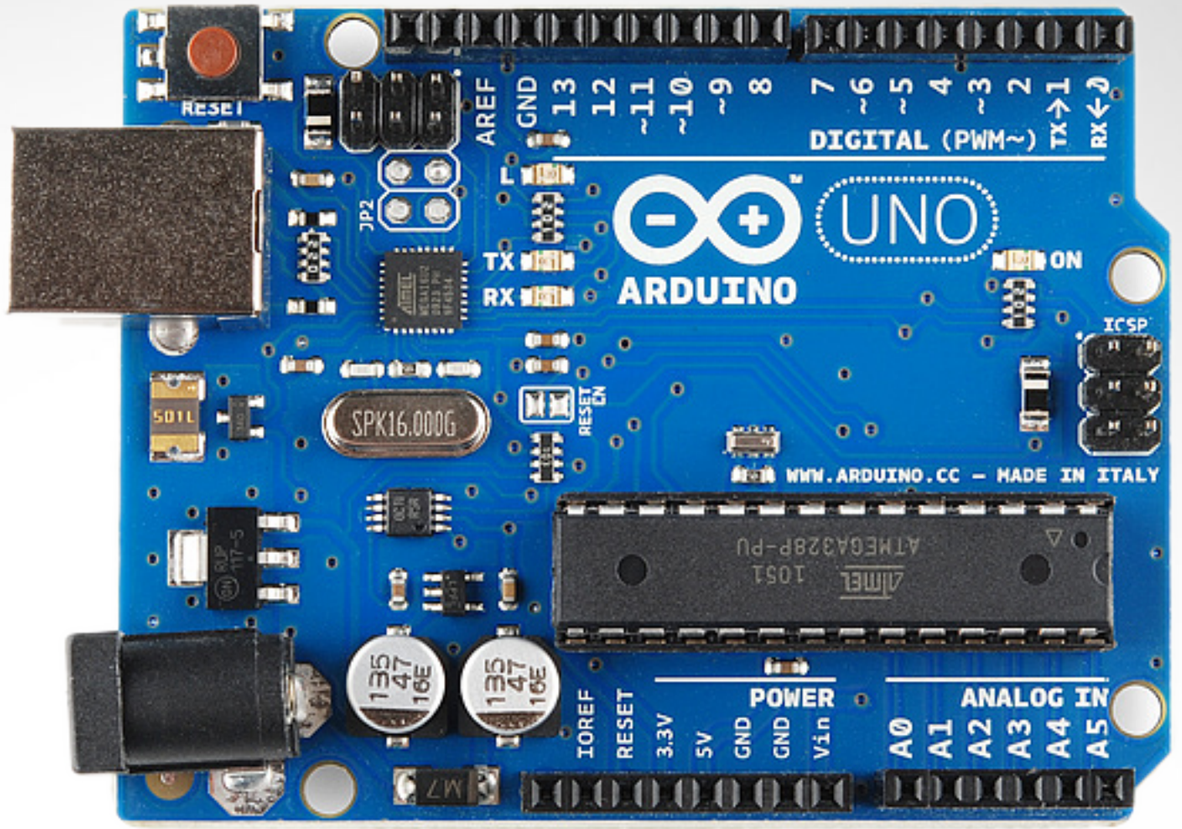


SmartThings

SmartThings

SmartThings

SmartThings



SmartThings

http://www.businessweek.com/articles/2013-02-14/smartthings-aims-to-deliver-the-internet-connected-home

How does SmartThings use Vert.x?

Hubs/Clients need to maintain
always open socket

amqp bus mode to push/pull events
to/from Rabbit MQ

Event Bus to get messages to the
right socket

SmartThings Vert.x Throughput

> 1k events/second ~ 100 million events/day from hubs to Vert.x in our production environment

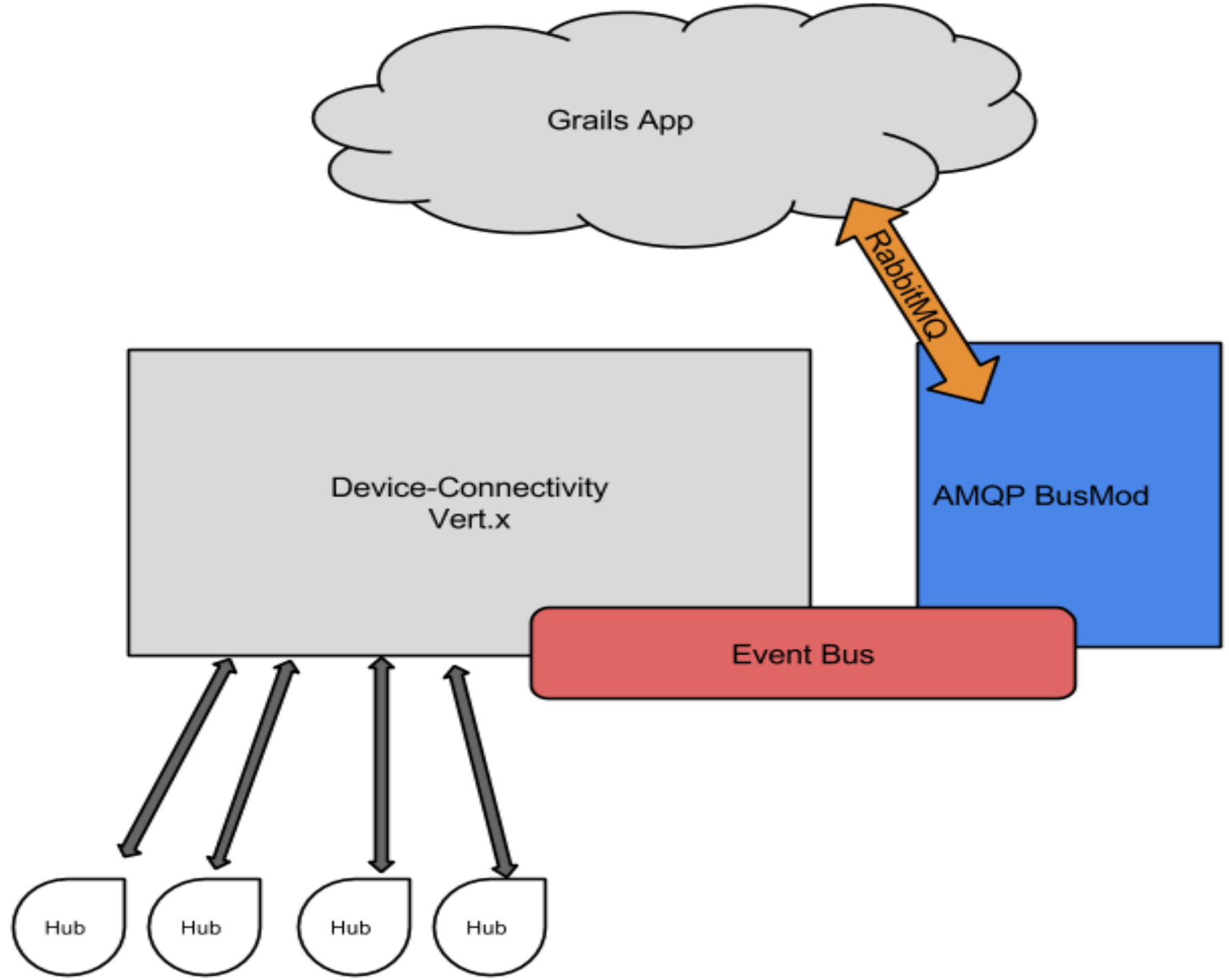
In our load testing environment we've easily achieved 5x our production numbers and still plenty of room to go. That's almost ½ billion events/day!

Running 8 Vert.x instances in prod

Primary reason is stability, not throughput

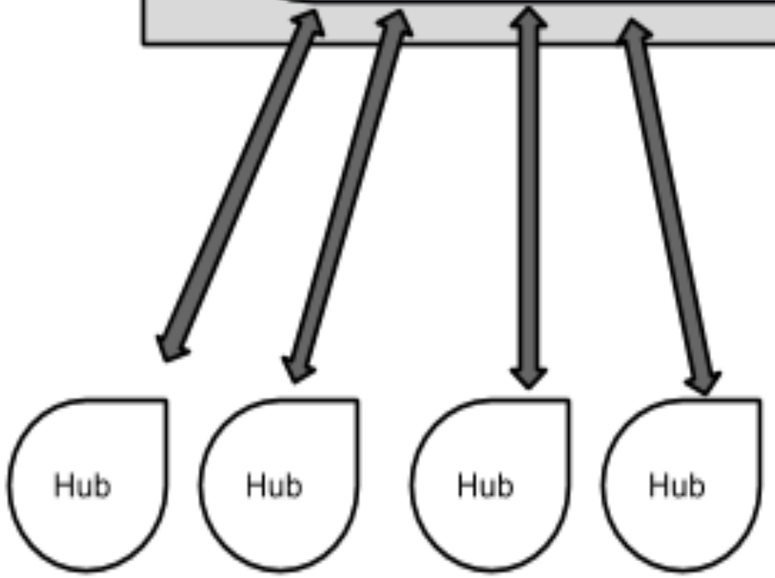
Mirrored on ios, android, and windows clients

SmartThings

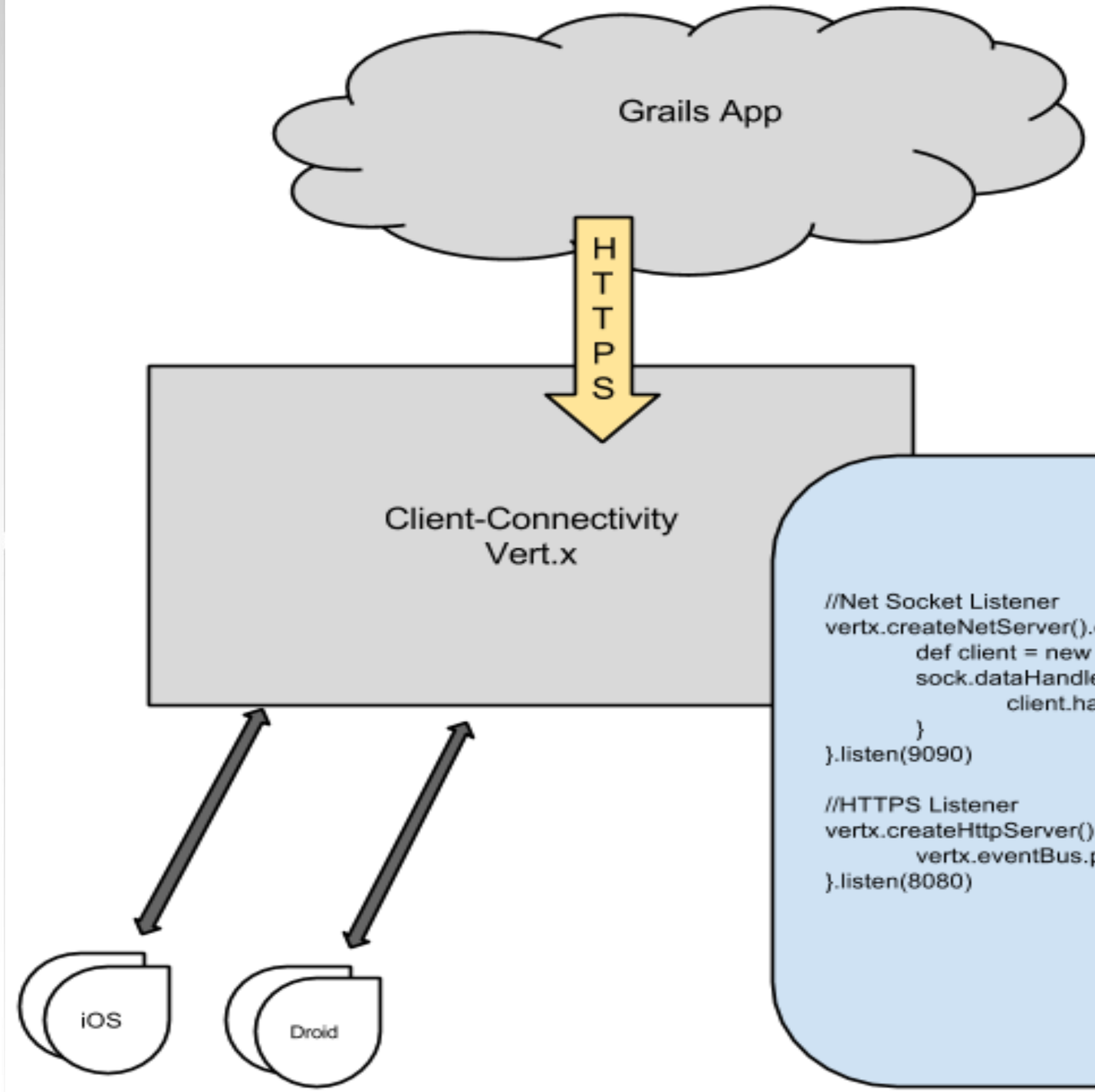


SmartThings

```
def netServer = vertx.createNetServer()
netServer.connectHandler { sock ->
        def hub = new HubConnection()

        sock.dataHandler { buffer ->
                hub.processMessages(buffer)
        }

        sock.exceptionHandler { e -> hub.
socketException(e) }
        sock.closeHandler { hub.close() }
}

netServer.listen(9090)
```

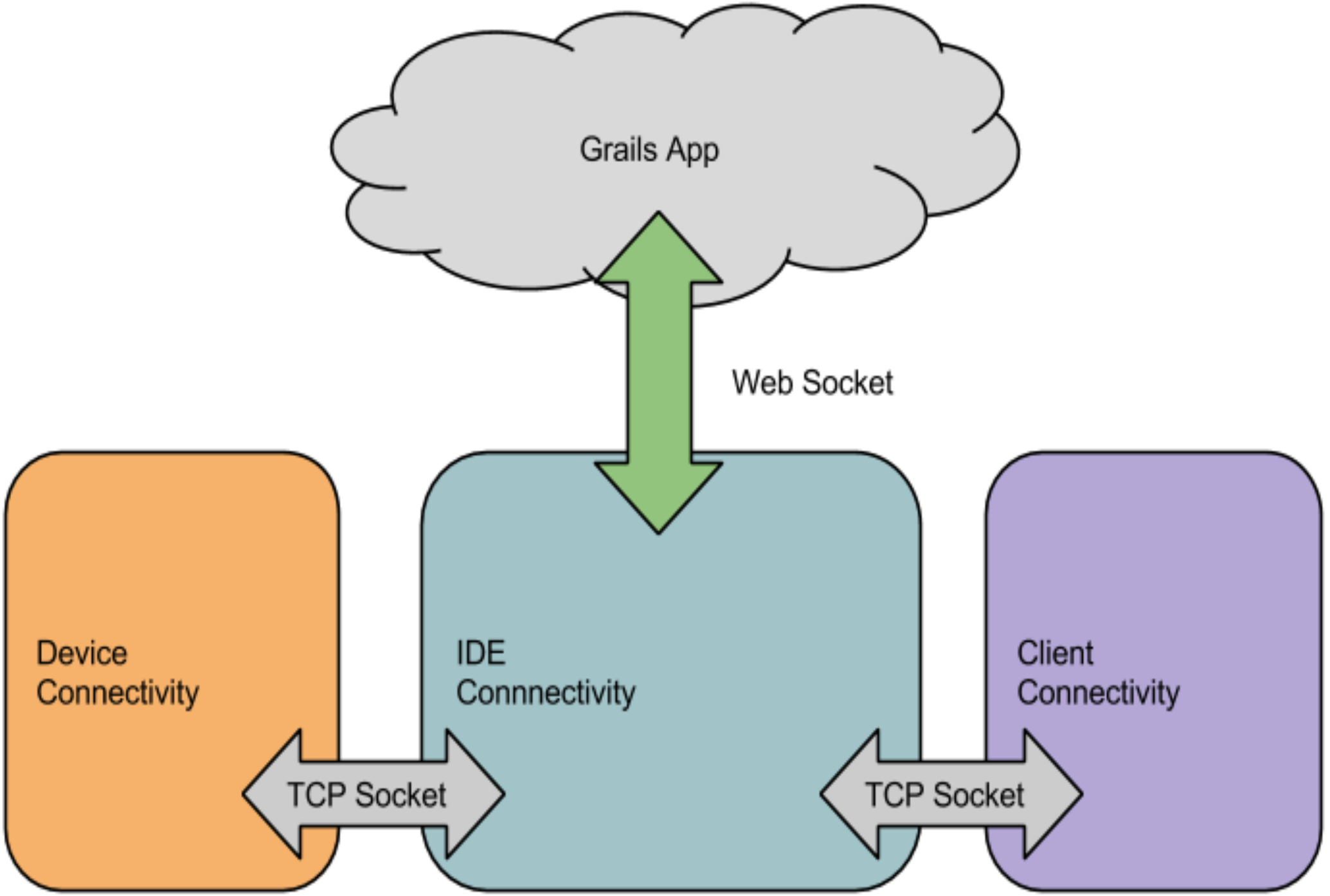


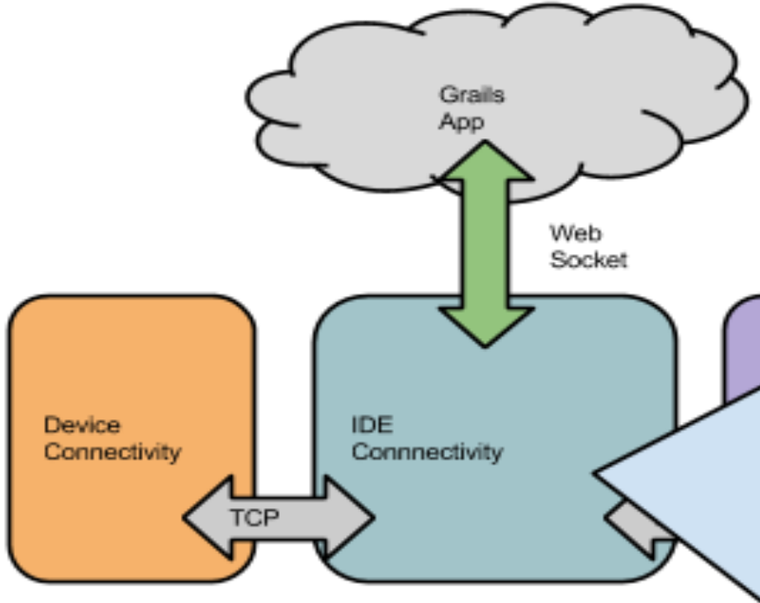
SmartThings




```
//Net Socket Listener
vertx.createNetServer().connectHandler { sock ->
        def client = new ClientConnection()
        sock.dataHandler { data ->
                client.handle(data)
        }
}.listen(9090)

//HTTPS Listener
vertx.createHttpServer().requestHandler { request ->
        vertx.eventBus.publish(pubsubId, request.params.msg)
}.listen(8080)
```

SmartThings






```
vertx.createHttpServer().websocketHandler { ws ->

        switch(type) {
                case 'device':
                        //device conn configs
                break
                case 'client':
                        //client conn configs
                break
                default:
                        ws.reject()
        }

        vertx.createNetClient().connect(configPort, configHost) { socket ->

                //Write *-conn socket data to Web Socket
                socket.dataHandler { data ->
                        ws.writeTextFrame(buffer.toString().trim())
                }
                //Send web socket commands down TCP Socket
                ws.dataHandler { data ->
                        socket << "${data}\n"
                }
                ws.closedHandler {
                        socket.close()
                }
        }
}.listen(9090)
```

SmartThings

Demo – WebSockets in the IDE

- Log into SmartThings IDE and show simulator
- Show how ide, device, and client conns work together to send device messages back and forth using the event bus and websockets to update the simulator

What's new in Vert.x 3

Truly embeddable, pluggable library (no longer framework)

Simple flat model (no extra classloaders)

Build in RxJava support (Rx-ified versions of all Vert.x APIs)

Vert.x-Web - this is a toolkit for writing modern web applications with Vert.x

Experimenting with synchronous style code without the need for callback hell of programming against asynchronous APIs

Vert.x 3 - Continued

Vertx 3 is even more targeted at the reactive microservice space

Support for pluggable messaging

Support for more than one cluster manager

Async support for MySQL, Redis, PostgreSQL, MongoDB, etc…

Out of the box metrics support with DropWizard metrics

Vert.x 3 - Core

Vert.x core contains fairly low level functionality including support for HTTP, TCP, file system access, and various other features. You can use this directly in your own applications, and it's used by many of the other components of Vert.x

https://github.com/vert-x3/vertx-examples/tree/master/core-examples

Vert.x 3 – Core Examples

Embed Vert.x core in any Java class and run it that way

Vert.x Net servers and clients (TCP/SSL)

HTTP/HTTPS servers

Websockets

Pub/Sub

Vert.x 3 - Web

Vert.x-Web is a toolkit for writing sophisticated modern web applications and HTTP microservices.

https://github.com/vert-x3/vertx-examples/tree/master/web-examples

Vert.x 3 – Web Examples

HTTP/REST microservices

Static sites with templating

Sessions

Auth

Cookies

HTML Forms

Upgrading from Vert.x 2 – Dependency Changes

Remove vert.x-platform from pom

Change all imports for Vertx from org.vertx to io.vertx

If using a language other than Java, change the dependency to vertx-lang-<<language>>

Remove any modules references that are using Vert.x 2.x

Use Vertx-unit and remove old teststools dependency

Upgrading from Vert.x 2 – Build Changes

Remove all vertx maven plugin code to generate modules and create fat jars instead

If you were running your application with runMod or something like that then you need to create a *fat* jar, changing the build file as in

Upgrading from Vert.x 2 – Code Changes

Verticle is now an interface and not a class to extend, so using Groovy as an example you now extend GroovyVerticle. In Java extendAbstractVerticle instead.

JsonObject.toMap() changed to JsonObject.getMap()

There isn’t a container variable in Verticles anymore for deploying verticles and also a config file. You need to usevertx.getOrCreateContext().config() to get to it

Resources

http://vertx.io/

http://vertx.io/core_manual_groovy.html

http://vertxproject.wordpress.com/2012/05/09/vert-x-vs-node-js-simple-http-benchmarks/

http://techempower.com/benchmarks/

Questions?

Make your world smarter.

Ryan Applegate