SmartThings

Make your world smarter.

Ryan Applegate
Adam Lanners

Node for the JVM

(Getting Groovy with Vert.x)

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

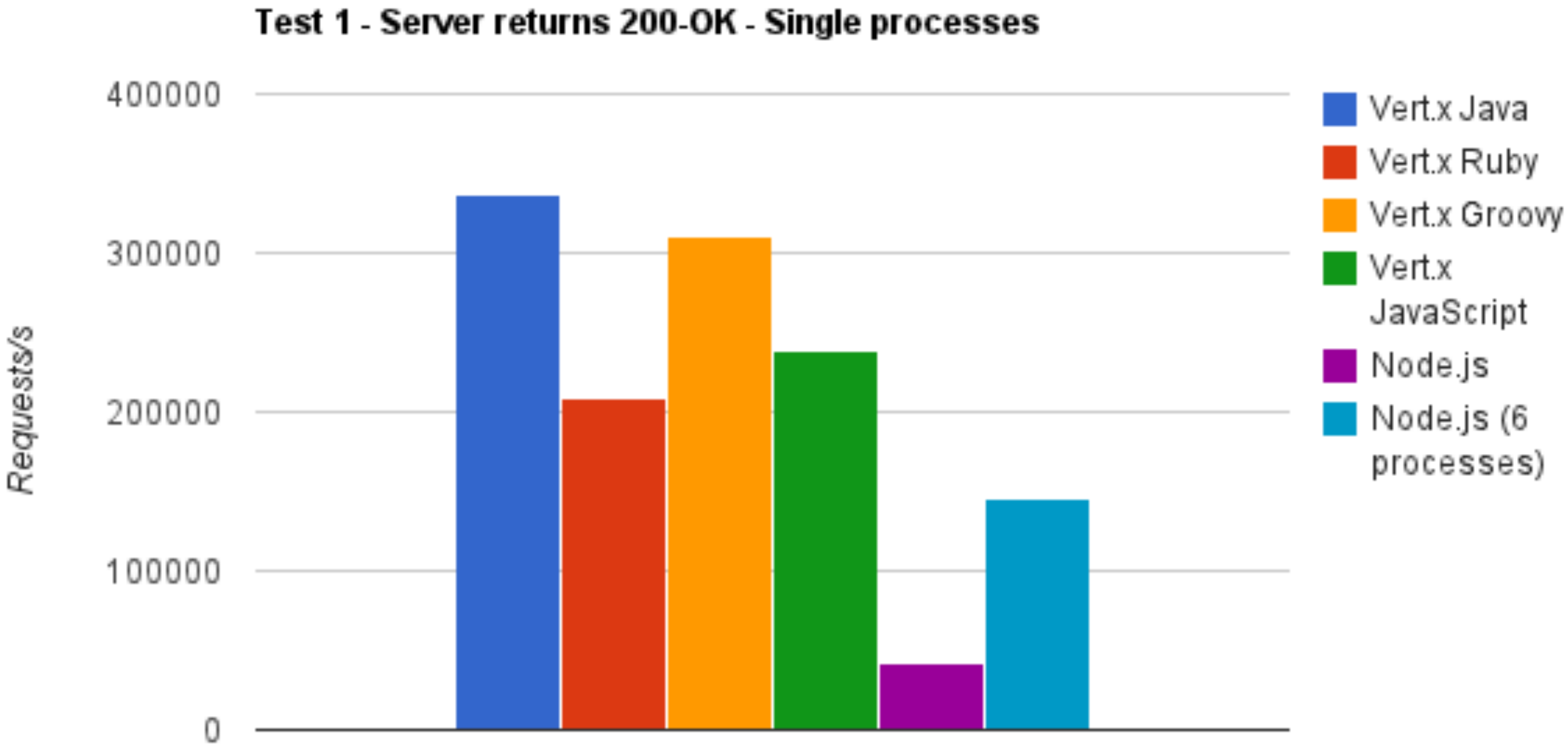
Interprocess Communication via Event Bus

Asynchronous & Effortlessly Scalable

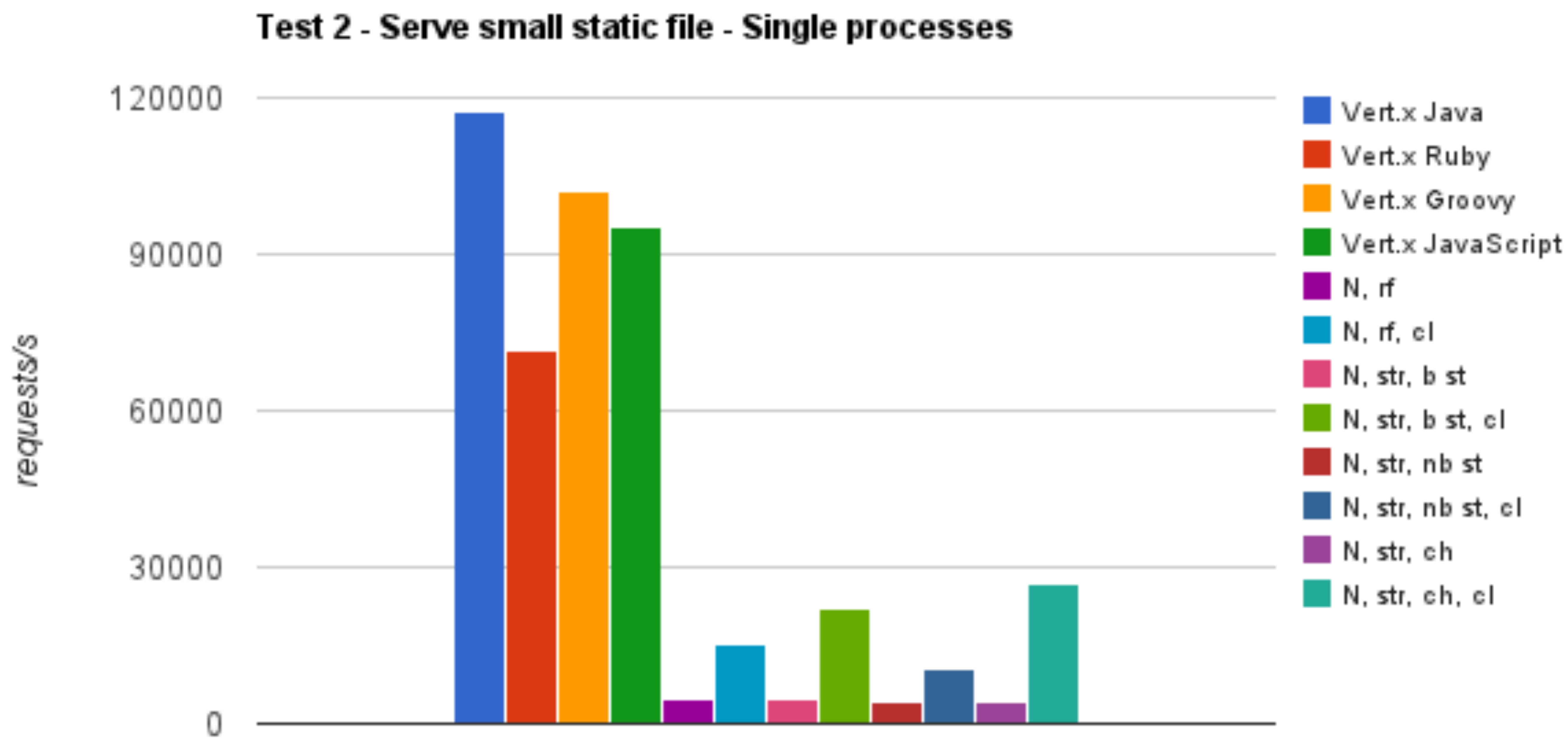
Vert.x Caveat

Built on Netty and NIO2 for Network I/O

MUST be running Java 7

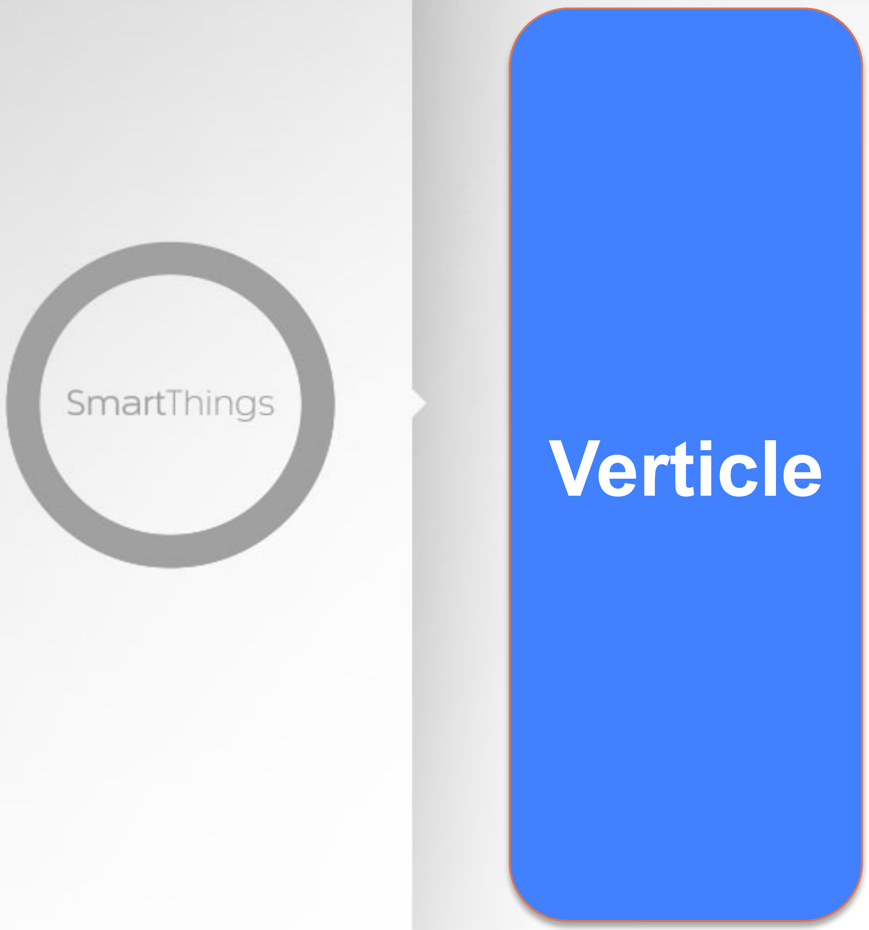

Test 1 - Server returns 200-OK - Single processes
Requests/s
400000
300000
200000
100000
0
Vert.x Java
Vert.x Ruby
Vert.x Groovy
Vert.x JavaScript
Node.js
Node.js (6 processes)

Benchmark #1



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

Verticle



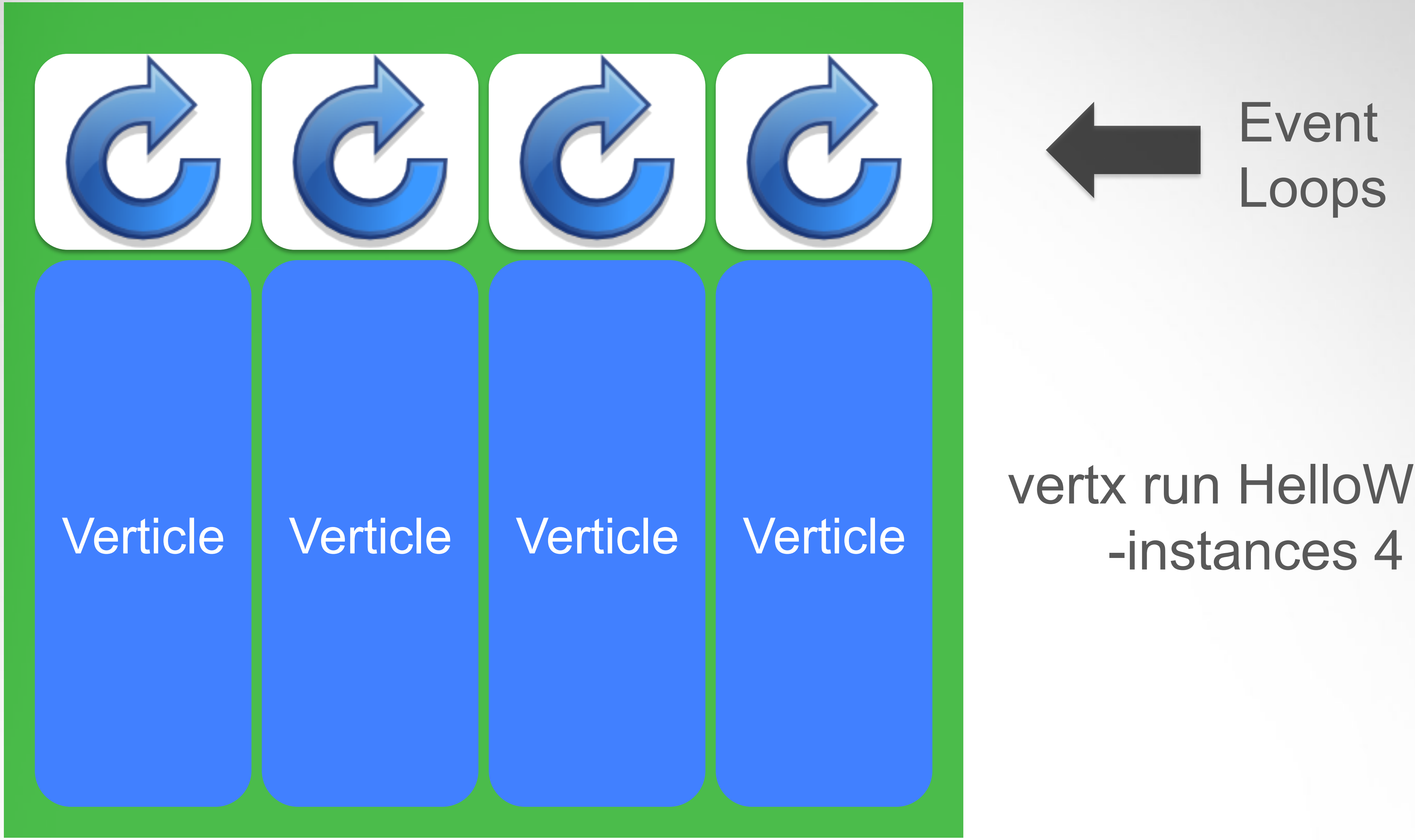
SmartThings

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



```
vertx run HelloWorld
-instances 4
```

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
```

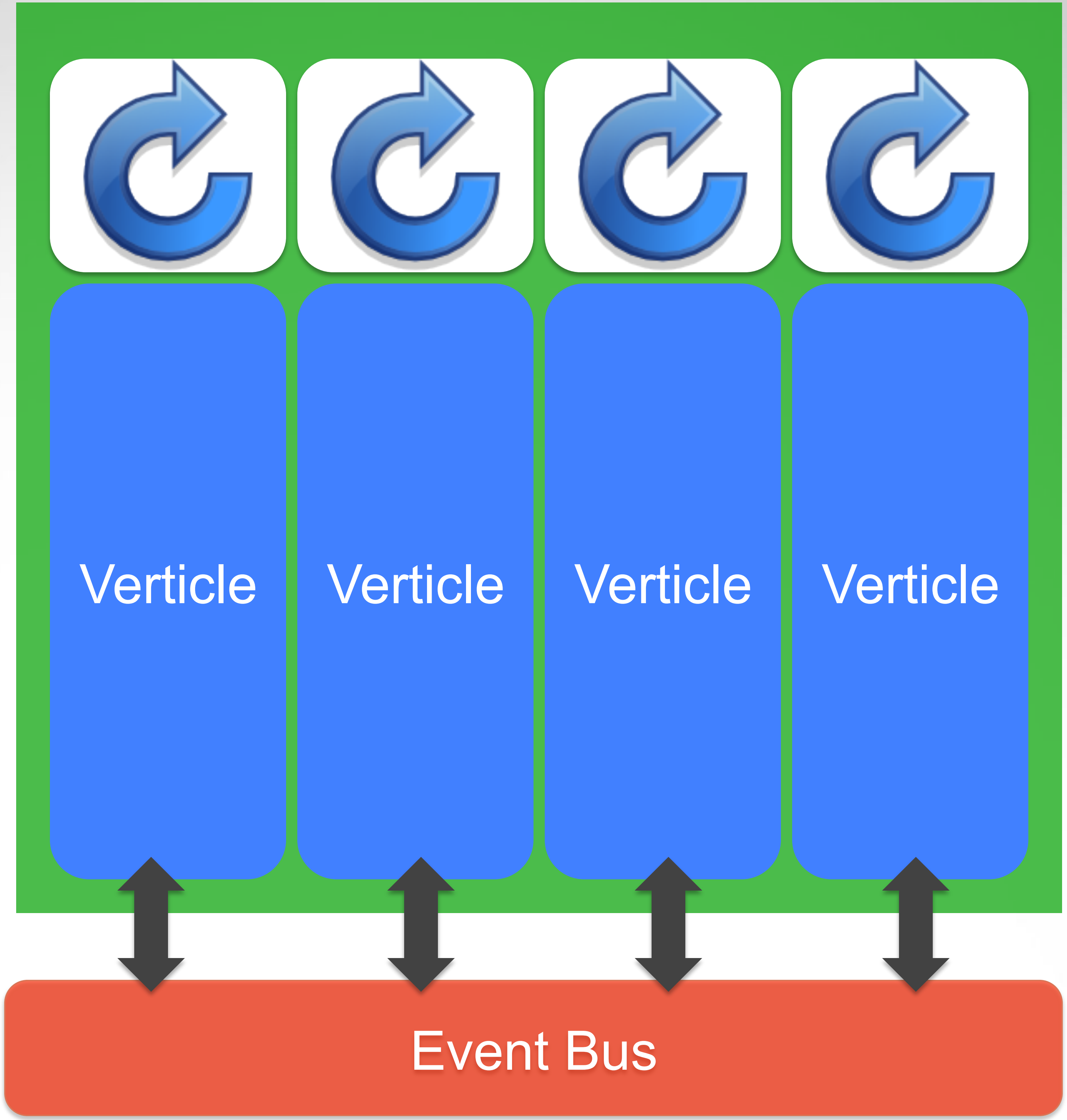
Concurrency

Verticle instance ALWAYS executes on assigned thread/event loop.

Verticles have isolated classloaders and cannot share global state.

Write all your code as single threaded.

No more synchronized and volatile!


Verticle
Verticle
Verticle
Verticle
Event Bus

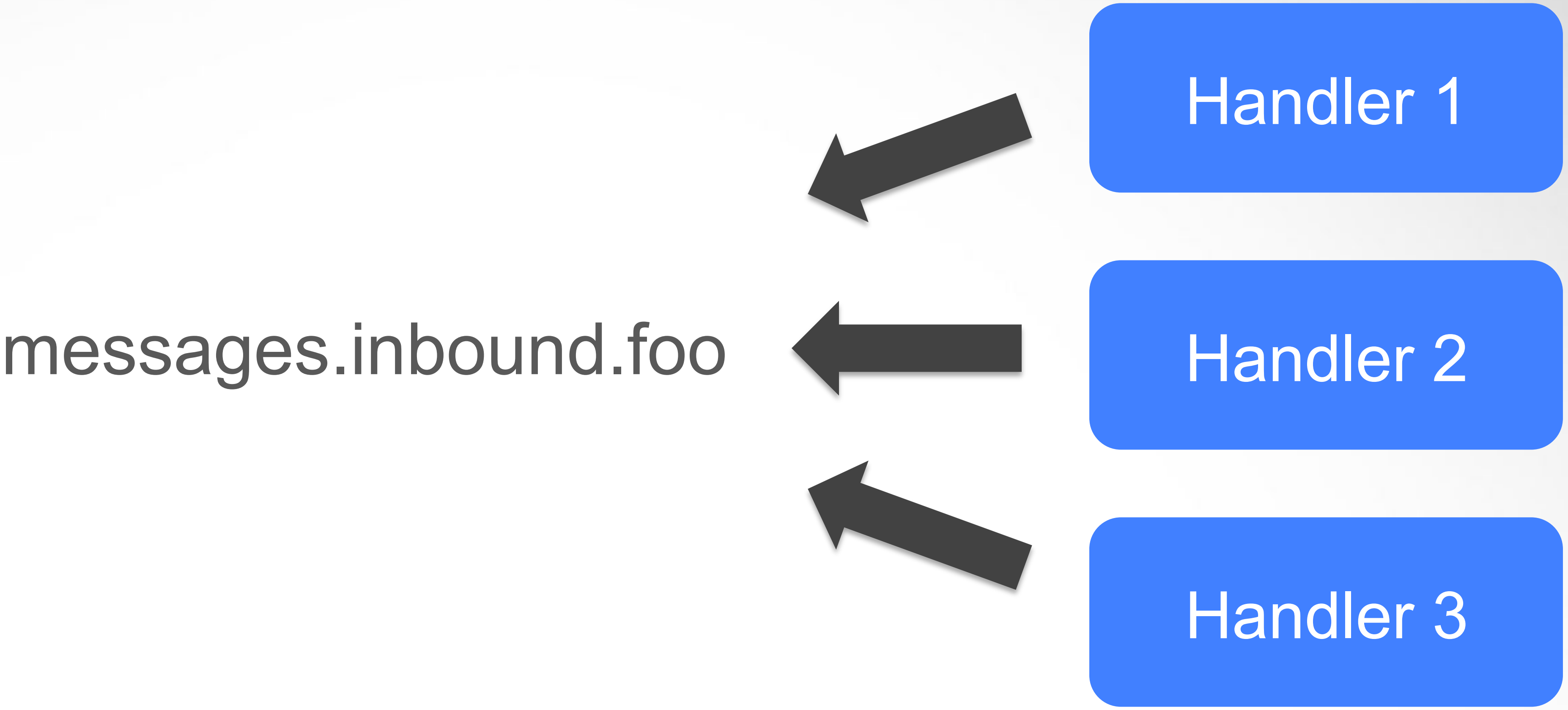
Event Bus Addressing

Address simply a String

Dot-style namespacing recommended

"messages.inbound.foo"

Handler Registration

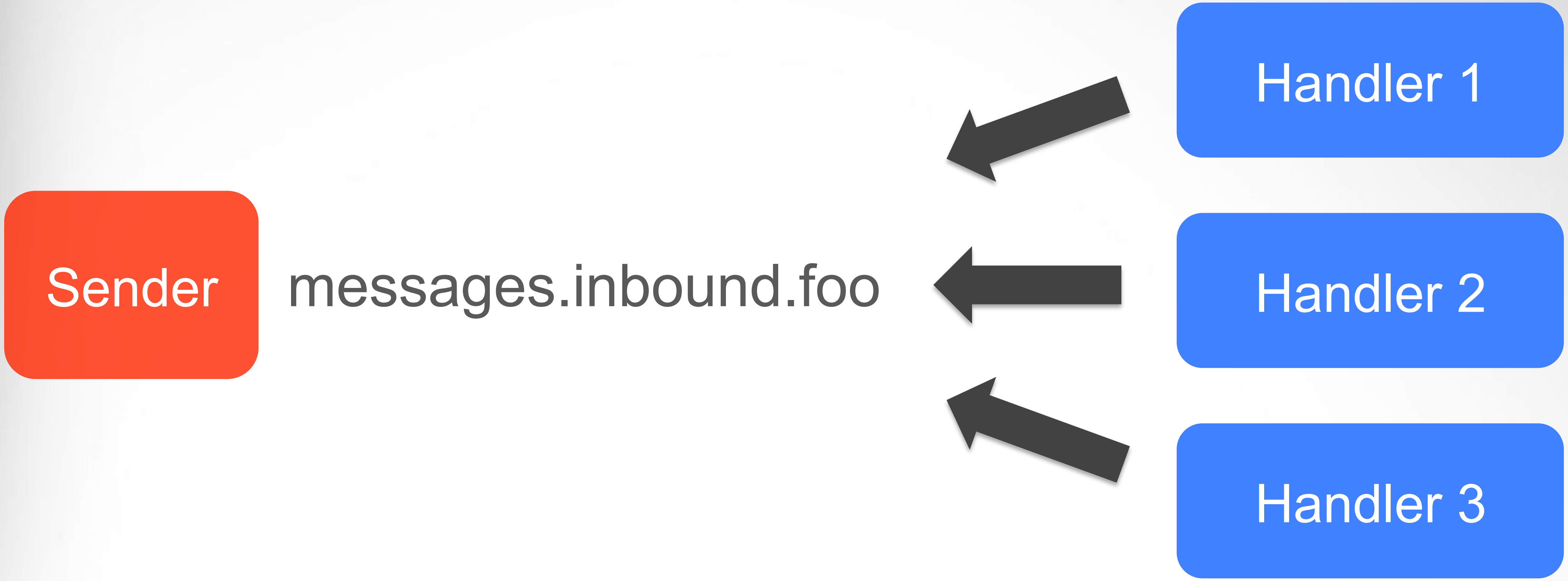


Handler Registration

```
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



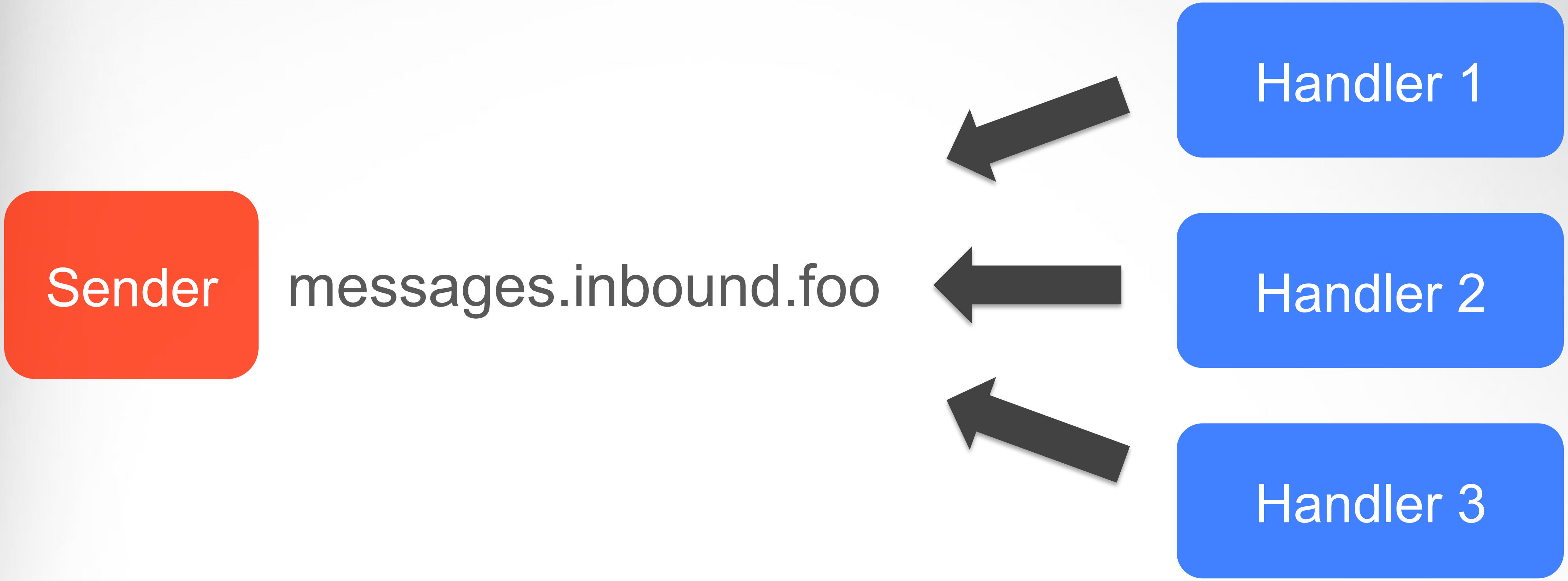
Pub/Sub

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



Sender

messages.inbound.foo

Handler 1

Handler 2

Handler 3

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

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"
```

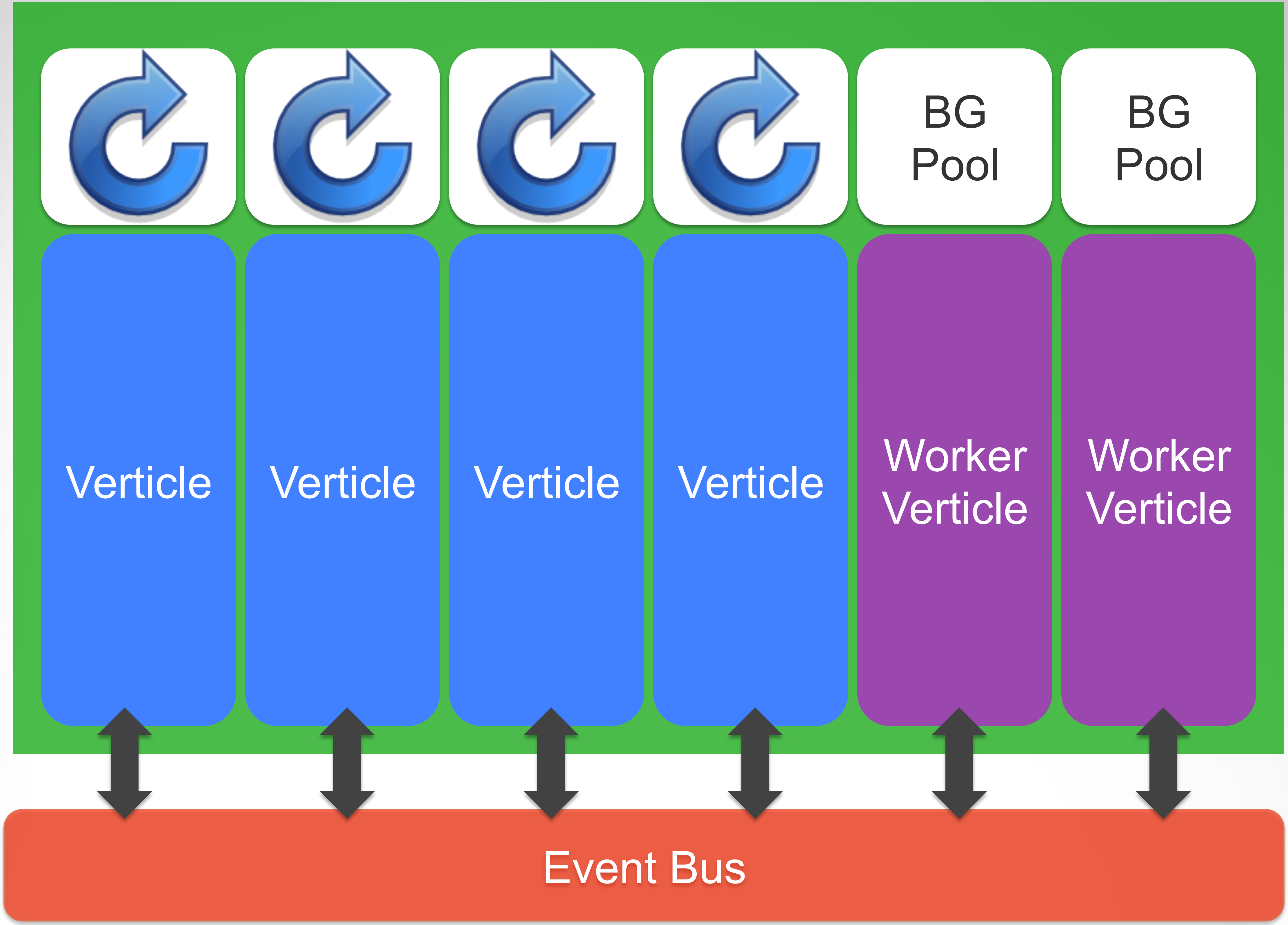
Shared Set

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")
```


BG Pool
BG Pool
Verticle
Verticle
Verticle
Verticle
Worker Verticle
Worker Verticle
Event Bus

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) }
}
```

SmartThings

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

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

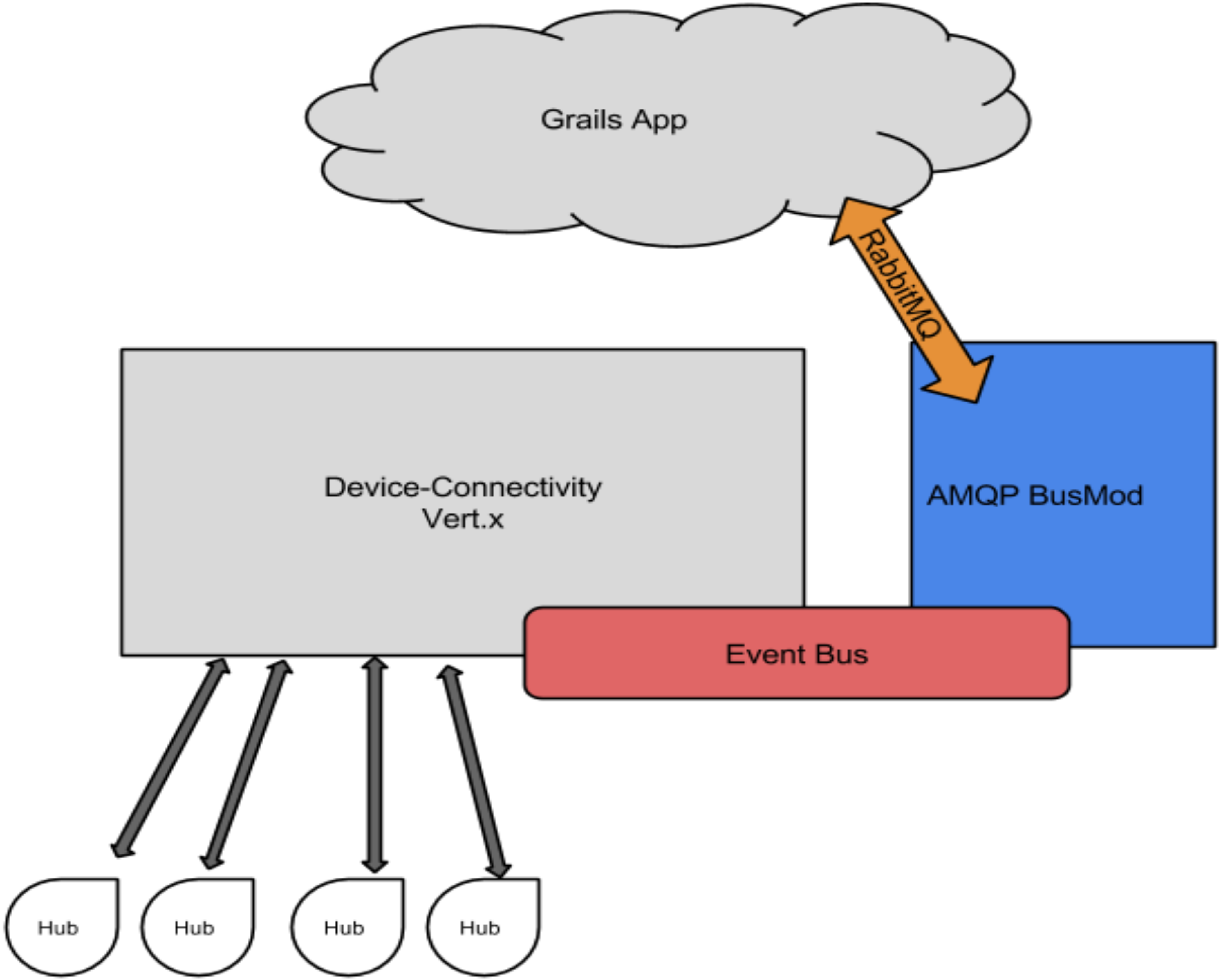
500 events/second ~ 43.2 million events/day from hubs to Vert.x in our production environment

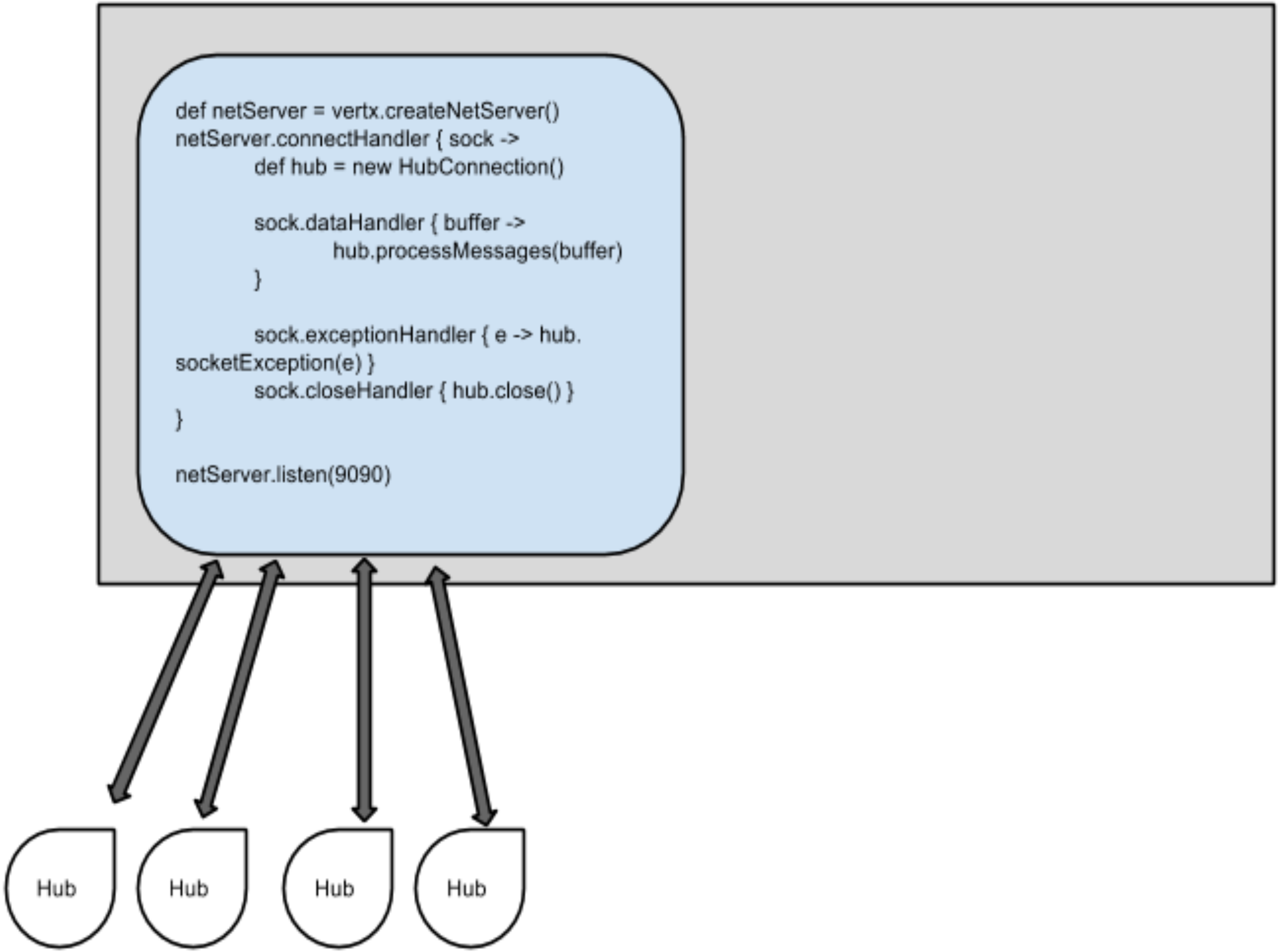
In our load testing environment we’ve easily achieved 10x our production numbers and still plenty of room to go

Cluster of 5 Vert.x instances

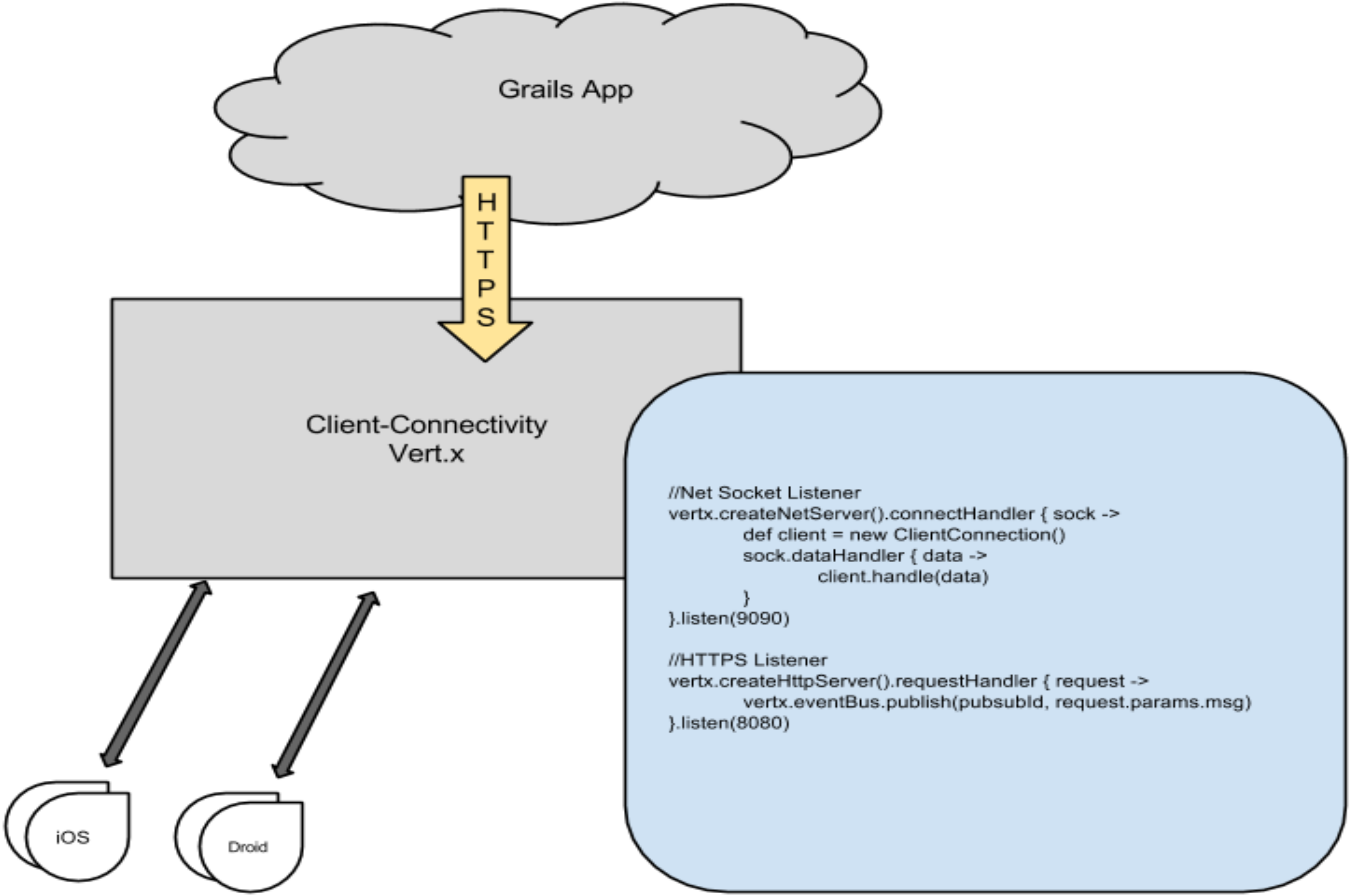
Primary reason is stability, not throughput

Mirrored on ios and android clients

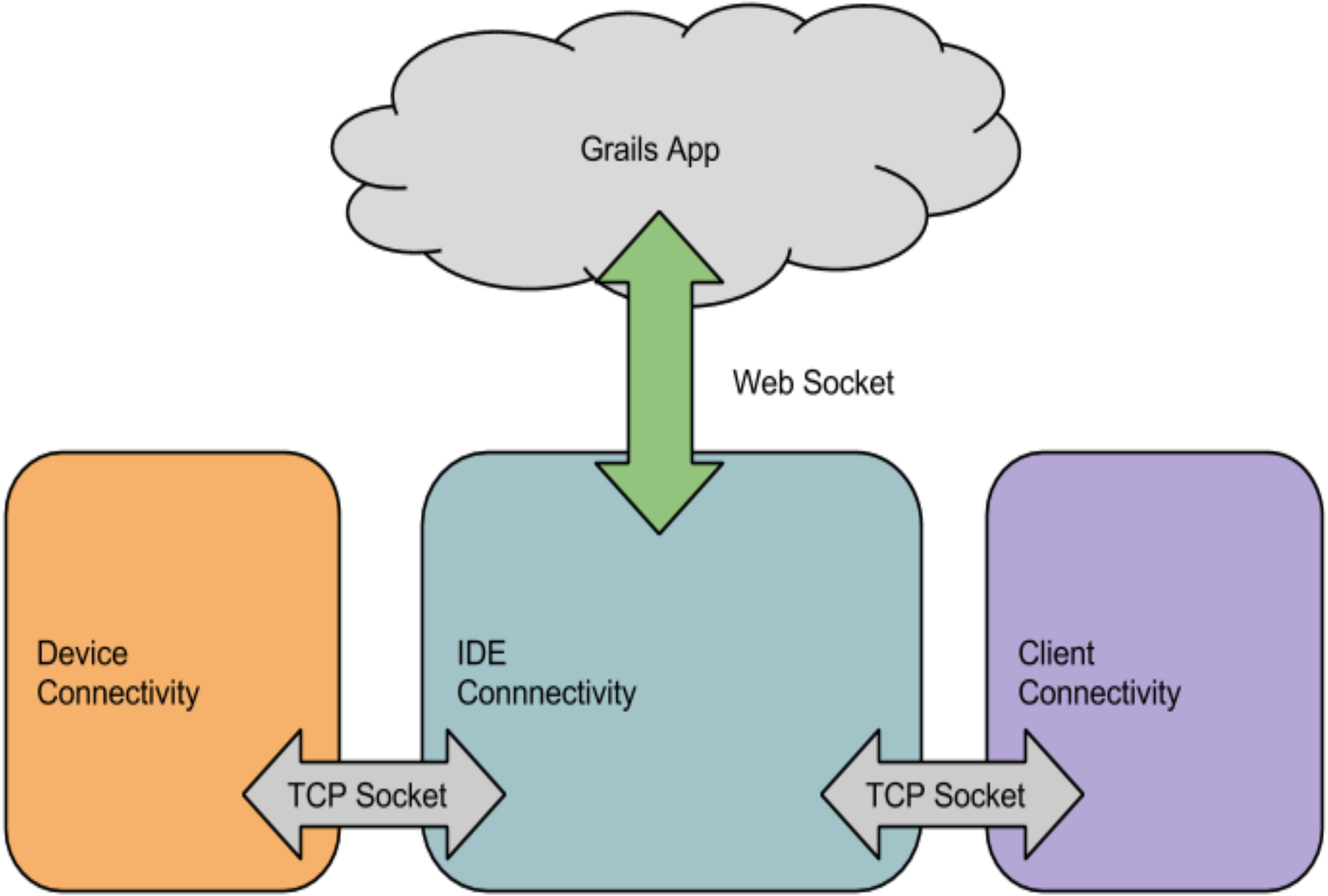

Grails App
RabbitMQ
Device-Connectivity
Vert.x
AMQP BusMod
Event Bus
Hub
Hub
Hub
Hub



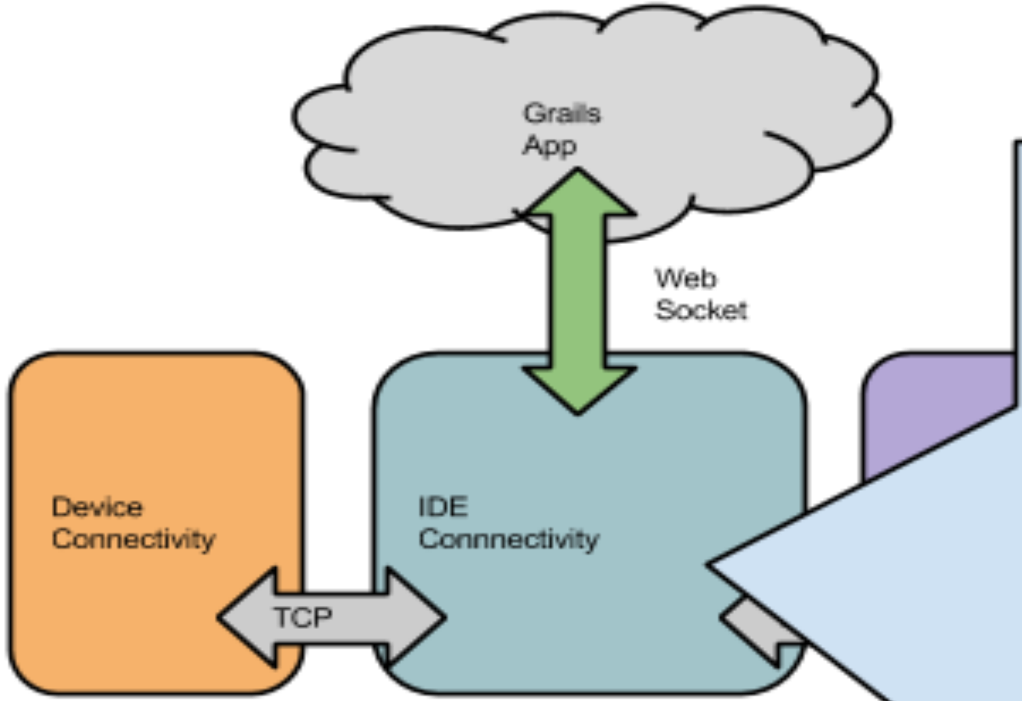
```
def netServer = vertx.createNetServer()
netServer.connectHandler { sock ->
        def hub = new HubConnection()

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

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

netServer.listen(9090)
```



```
//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)
```



Grails App
Web Socket
Device
Connectivity
IDE
Connnectivity
Client
Connectivity
TCP Socket
TCP Socket



```
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)
```

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/

Questions?

SmartThings

Make your world smarter.

Ryan Applegate
Adam Lanners