



Côté Serveur

De Java 8 à Vert.X

Sommaire

- Java 8
 - lambdas
 - streams
- La JVM, l'environnement polyglotte
- Groovy
- Vert.X



Java - retour en arrière

- 1994 - Création du langage centré sur le Web
- 1998 - Java 1.2 : J2SE et J2EE
- 2000 - Java 1.3 : HotSpot, JNDI
- 2002 - Java 1.4 : Expressions régulières, XML
- 2004 - Java 5 : Génériques, Enums, Autoboxing, Annotations, VarArgs, Java Web Start, ...
- 2006 - Java 6 : Beaucoup de patchs de sécu
- 2006 - Le compilateur et la JVM passent open Source
- 2007 - Java passe open Source et devient openJDK
- 2009 - Oracle rachète Sun
- 2011 - Java 7 : Licence GPL, Switch String, NIO, ...
- 2013 - Java 8 : Closures, lambdas, streams, ...
- 2016 - Java 9 : ?



ORACLE®

Java 8

Les nouveautés

- Lambda expressions
 - lambda
 - références de méthodes
 - interfaces fonctionnelles
 - programmation fonctionnelle
- Stream API
 - pipeline process



Java 8 - Lambda

Peut être vu comme un sucre syntaxique pour les classes anonymes

Simplifient énormément le développement

```
1 () -> System.out.println(this)
2 (String str) -> System.out.println(str)
3 str -> System.out.println(str)
4 (String s1, String s2) -> { return s2.length() - s1.length(); }
5 (s1, s2) -> s2.length() - s1.length()
6 Arrays.sort(strArray, (String s1, String s2) -> s2.length() - s1.length());
7 server.requestHandler(request -> {
    request.response().end("hello world!");
});
```

Java 8 - Lambda - Scope

```
1 import static java.lang.System.out;
2
3 public class Hello {
4     Runnable r1 = () -> out.println(this);
5     Runnable r2 = () -> out.println(toString());
6
7     public String toString() { return "Hello, world!"; }
8
9     public static void main(String... args) {
10         new Hello().r1.run(); //Hello, world!
11         new Hello().r2.run(); //Hello, world!
12     }
13 }
```

r1 et r2 appellent la méthode `toString()` de la classe Hello

Java 8 - Lambda - Method references

```
1 public class FileFilters {  
2     public static boolean fileIsPdf(File file) { /*code*/ }  
3     public static boolean fileIsTxt(File file) { /*code*/ }  
4     public static boolean fileIsRtf(File file) { /*code*/ }  
5 }
```

```
1 Stream<File> pdfs = getFiles().filter(FileFilters::fileIsPdf);  
2 Stream<File> txts = getFiles().filter(FileFilters::fileIsTxt);  
3 Stream<File> rtfs = getFiles().filter(FileFilters::fileIsRtf);
```

On a codé
getFiles()

```
1 Files.lines(Paths.get("Nio.java"))  
2         .map(String::trim)  
3         .forEach(System.out::println);
```

Java 8 - Lamda - functional IFace

Défini une interface avec une méthode abstraite :

- Function<T,R> - prend un T et retourne un R.
- Supplier<T> - retourne juste un T.
- Predicate<T> - prend un T et retourne un booléen.
- Consumer<T> - prend un T et effectue une action dessus.
- BiFunction - comme Function, mais avec 2 paramètres.
- BiConsumer - comme Consumer, mais avec 2 paramètres.

```
1 Function<String, String> atr = (name) -> {return "@ " + name; };
2 Function<String, Integer> leng = (name) -> name.length();
3 Function<String, Integer> leng2 = String::length;

1 for (String s : args) out.println(leng2.apply(s));
```

Java 8 - Lambda - Java 7?

```
1 // Java 7
2 ActionListener al = new ActionListener() {
3     @Override
4     public void actionPerformed(ActionEvent e) {
5         System.out.println(e.getActionCommand());
6     }
7 }

8 // Java 8
9 ActionListener al8 = e -> System.out.println(e.getActionCommand());
```

Création d'un ActionListener

Java 8 - Lambda - Java 7?

```
1 // Java 7
2 for (String s : list) {
3     System.out.println(s);
4 }

5 //Java 8
6 list.forEach(System.out::println);
```

Afficher une liste de Strings

Java 8 - Lambda - Java 7?

```
1 // Java 7
2 Collections.sort(list, new Comparator<String>() {
3     @Override
4     public int compare(String s1, String s2) {
5         return s1.length() - s2.length();
6     }
7 });

8 //Java 8
9 Collections.sort(list, (s1, s2) -> s1.length() - s2.length());
10 // or
11 list.sort(Comparator.comparingInt(String::length));
```

Trier une liste de Strings

Java 8 - Lambda - prog fonctionnelle

```
1 Function<Integer, String> f = Function.<Integer>identity()
2           .andThen(i -> 2*i).andThen(i -> "str" + i);
```

Java 8 - Streams

Streams ???

Un peu comme un Iterator mais qui peut s'exécuter en parallèle.

L'interface Stream supporte le concept de map/filter/reduce

Java 8 - Streams

Streamer une collection

L'interface collection a 2 méthodes :

- **stream()**
 - retourne un stream séquentiel dont la source est la collection.
- **parallelStream()**
 - retourne un possible stream parallèle dont la source est la collection

Java 8 - Streams

Streamer des fichiers

```
1 try (FileReader fr = new FileReader( "file" ));  
2     BufferedReader br = new BufferedReader(fr)) {  
3         br.lines().forEach(System.out::println);  
4     }
```

```
1 try (Stream st = Files.lines(Paths.get( "file" ))) {  
2     st.forEach(System.out::println);  
3 }
```

Le fichier est lu au fil de l'eau (pas chargé en mémoire)

Java 8 - Streams

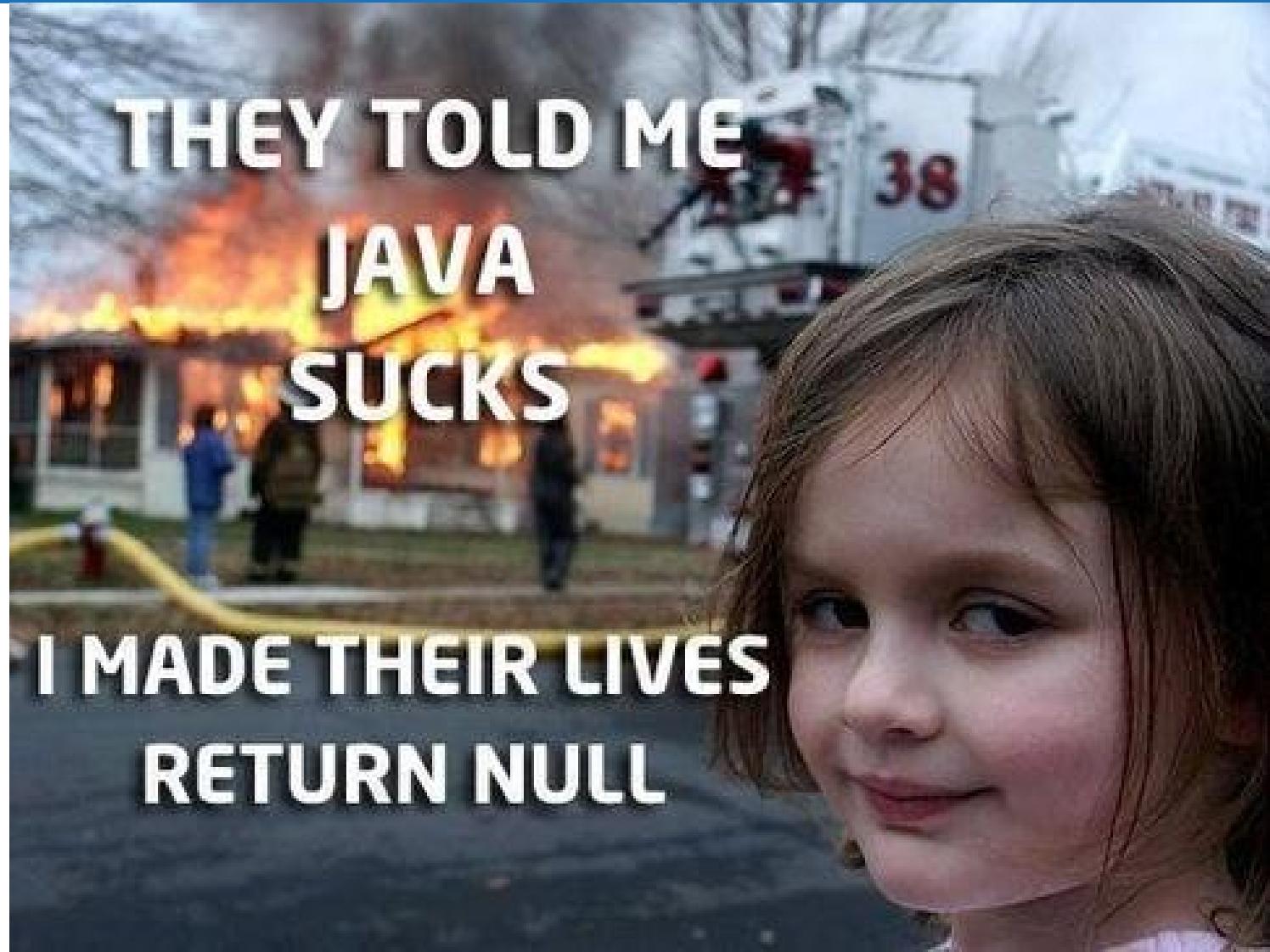
Streamer une arborescence de fichiers

- `list(Path dir)`
 - parcours d'un répertoire
- `walk(Path dir)`
 - parcours récursif
- `walk(Path dir, int maxDepth)`
 - idem mais limité en profondeur

Stream infini

```
1 Stream.generate(() -> Math.random());
```

Java 8 - c'est cool



JVM polyglotte

Dois-je coder en java pour exécuter un programme sur la JVM?

Oui, **mais pas que**, fais-toi plaisir!



JVM polyglotte

On peut programmer en plus de **200** langages

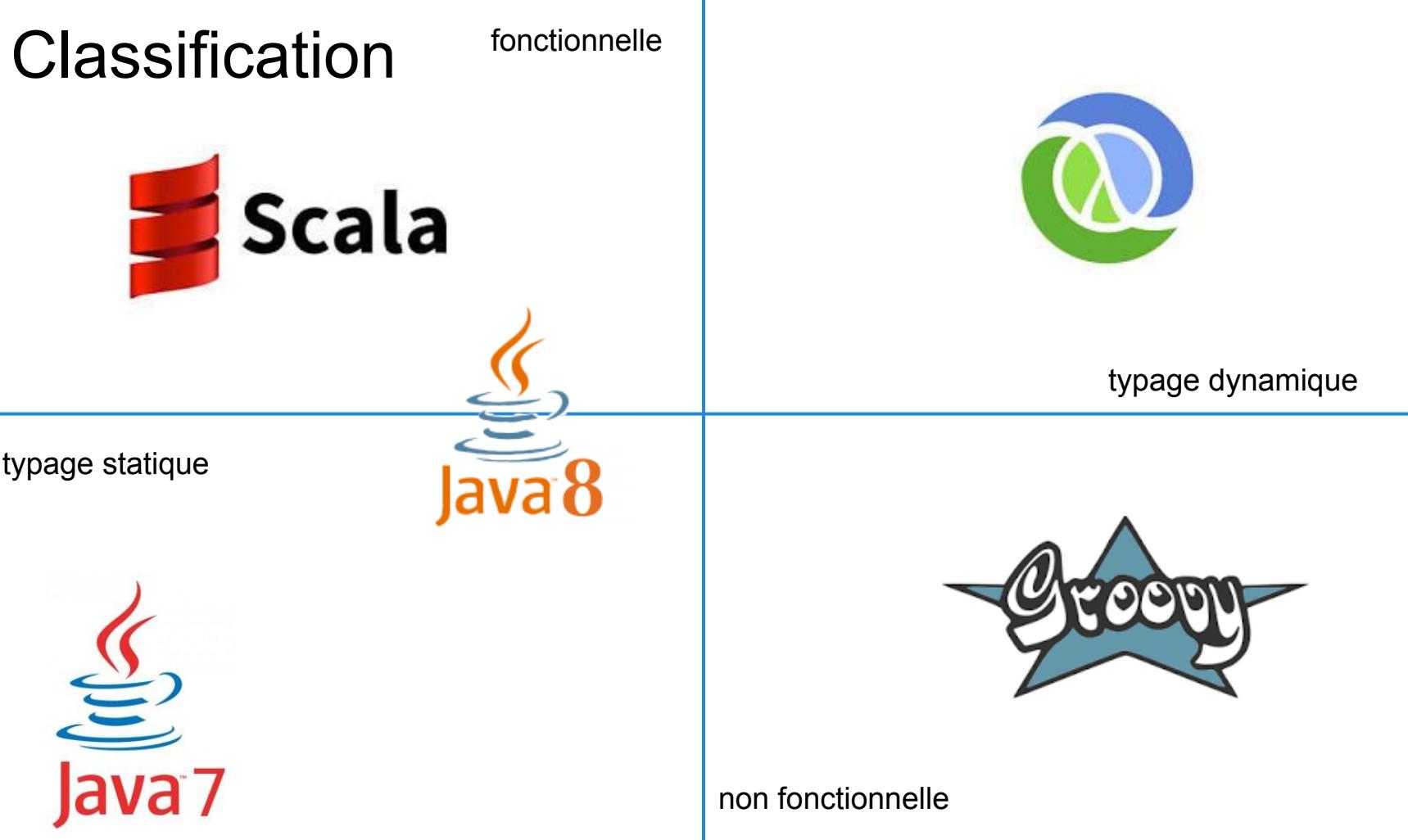
Java, JavaFX, Ceylon, Gosu, **Groovy**, Clojure,
~~Rhino~~ **Nashorn**, Mirah, **Scala**, JRuby, Xtend,
JPython, Fantom, Oxygene, Kotlin, ...

L'intérêt : à chaque problème sa solution

“Quand on n'a qu'un marteau dans la main, les vis ressemblent à des clous”

JVM polyglotte

Classification



JVM polyglotte - Scala

- 2003
- Orienté objet et langage fonctionnel
- Typage fort
- Conçu pour la JVM
- Concurrence : Actors
- Beaucoup de capacités avancées
- Aussi rapide que Java



JVM polyglotte - Scala



```
object HelloWorld {  
    def main(args: Array[String]) {  
        println("Hello World")  
    }  
}
```

```
class Person(name: String, age: Int)  
val pers = new Person("Xavier", 25)
```

JVM polyglotte - Clojure

- 2007
- Clojure c'est LISP
- Typage dynamique
- Conçu pour la JVM
- Le code est une donnée
- Des macros puissantes
- Bonne interopérabilité avec Java
- Clojure est rapide
- Concurrence : mémoire transactionnelle partagée



JVM polyglotte - Clojure

La programmation fonctionnelle ?

- Parallélisation
- Modularité et composition
- Amélioration de la qualité du code
- Abstraction



```
(println "Hello World")  
(def biggest  
  "trouve le plus grand de 2 nombres"  
  (fn [x y]  
    (if (> x y) x y)))
```

JVM polyglotte - Groovy

- 2003
- Typage dynamique
- Orienté objet
- Conçu pour la JVM
- Inspiré par Python, Ruby et Smalltalk
- Bonne intégration avec Java
 - un jar en plus
 - appeler du groovy depuis java == appeler du java depuis groovy
 - une syntaxe correcte Java est correcte en Groovy
 - renommer un .java et .groovy marche
 - Le second langage ciblant la JVM
 - Java fut le premier



Groovy



Groovy

- Complètement orienté objet
- Typage optionnel, statique ou dynamique
- Duck Typing
- Surcharge d'opérateurs
- Closures
- Multimethods et Metaprogramming

Groovy

Type optionnel

```
def s1 = "Homer"  
assert s1.class == String.class
```

```
s1 = 3  
assert s1.class == Integer.class
```

```
Integer s2 = 3  
s2 = "Simpson" // Oups une exception
```

Groovy

Typage dynamique?

Changer :

- le type des variables
- le comportement des objets
- le comportement des classes

Au run-time ...

Groovy

Exemple :
Surcharger la classe String

```
String.metaClass.isUppercase = {->  
    delegate.toCharArray().every{ Character.isUppercase(it) }  
}  
  
assert "GROOVY".isUpperCase() == true
```

Groovy

L'interpréteur Groovy

```
$> groovy -e "print new Date()"  
$> groovy -e "new File('.').  
eachFileRecurse { println it }"
```

Groovy - script

- pas de classe obligatoire ou de méthode main
- groovy les crée pour nous

```
// Fibonacci.groovy
// une méthode, le type de retour n'est pas obligatoire
def fibonacci(n) {
    n<2 ? 1 : fibonacci(n-1)+fibonacci(n-2)
}
if(args) { // arguments de la ligne de commande
    println fibonacci(args[0] as Integer)
}
```

Groovy - script

Lancement du script

```
$> groovy Fibonacci 8  
34
```

```
$> groovyc Fibonacci.groovy // Compilateur  
$> java -cp $GROOVY_HOME/embeddable/groovy-all.2.4.3.jar::.  
Fibonacci 8  
34
```

Groovy - syntaxe

La majorité de la syntaxe java s'applique à Groovy

- packages
- imports
- structures de contrôle
- gestion des exceptions
- classes et méthodes
- instantiation d'objets et appels de méthodes

Groovy - imports par défaut

Ces imports sont faits implicitement :

- `java.lang.*`
- `java.util.*`
- `java.net.*`
- `groovy.lang.*`
- `groovy.util.*`
- `java.math.BigInteger`
- `java.math.BigDecimal`

Groovy

- Une valeur nulle vaut false
 - Une collection ou une map vide vaut false
 - Une chaîne vide vaut false
 - Zéro vaut false
-
- == compare les valeurs (comme java equals())
 - la méthode “is” pour comparer les références d’objets

Groovy - Strings

```
def multiLine = """Master of puppet
I'm pulling the strings"""

def firstname = 'Kirk'; def name = "Hammet"
// GStrings

def fullname == "$firstname $name"
// Surchage d'opérateur

def s = name * 3

assert s == "HammetHammetHammet"

s = fullname - firstname

assert s.trim() == name
// slashy strings

s = /\nSlade Hudson\r/
assert s.size() == 16
```

Groovy - classes

```
class Person {  
    String firstName  
    String name  
    @Override  
    String toString() { "$firstName $name" }  
}
```

```
$> groovyc Person.groovy
```

-> Une Person.class directement utilisable

Groovy - classes

```
// Depuis groovy
p = new Person(name: "Mustain", firstName: "Dave")
assert p.toString() == "Dave Mustain"

// Depuis Java
public class Test {
    public static void main(String... args) {
        Person p = new Person();
        p.setName("Harris");
        p.setFirstName("Steve");
        Map<String, String> map = new HashMap<>();
        map.put("name", "Murray");
        map.put("firstName", "Dave");
        Person p1 = new Person(map);
    }
}
```

Groovy - opérateurs bonus

x?.method()

x = y ?: "no y"

[“Max”, “Cavalaria”]*.size() // [3, 8]

<=>, =~, ==~, .@, .&

Groovy - closures

```
// un bloc de code assigné à une variable
def num = { int n -> n <=> 0 }
// De quel type est-ce ?
assert num instanceof Closure
// it : l'argument par défaut de la closure
def num2 = { it <=> 0 }
assert num(-3) == num2(-3)

def x = 3
def times = { it * x }
assert times(3) == 9
x = 4
assert times(3) == 12
```

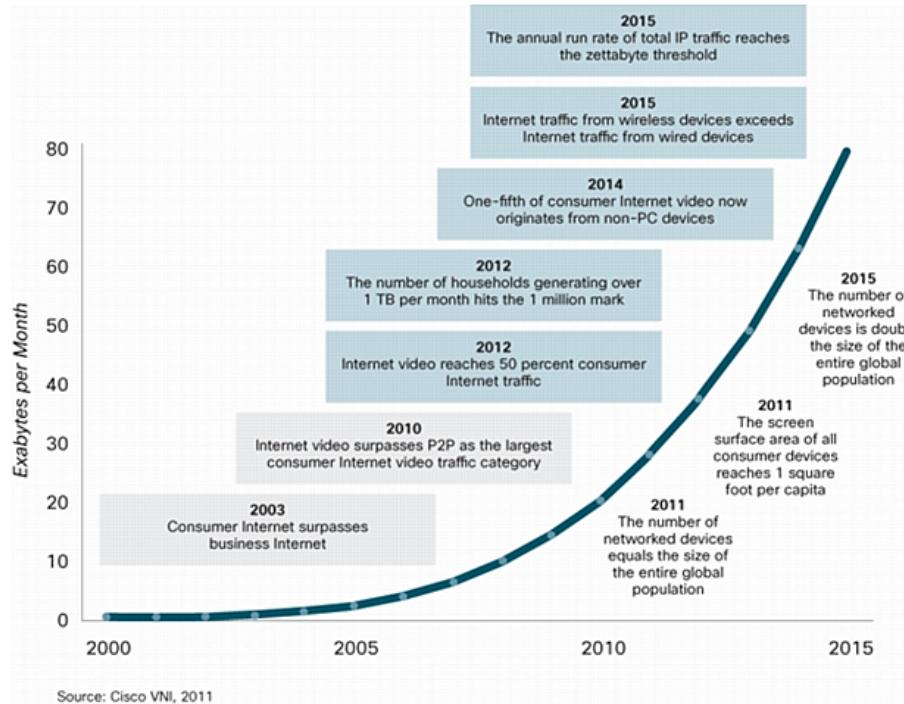
- Polyglotte
- Principes
- Fonctionnalités
- Event bus
- Données partagées
- Verticles
- Modules
- Benchmarks
- Au delà de Vert.X

<http://www.vertx.io>

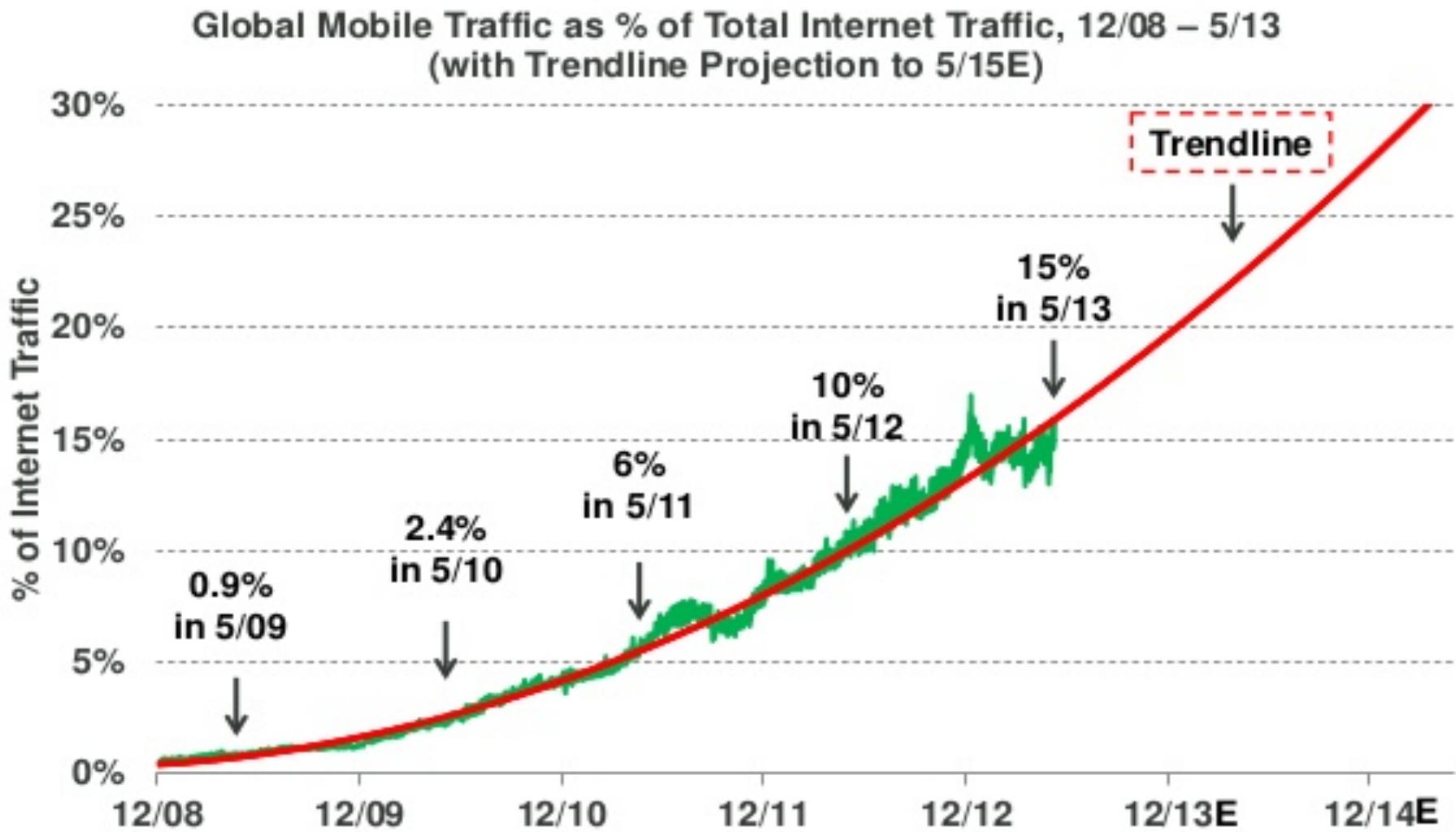
Vert.X - C10k

- Répondre à un grand nombre de clients en même temps
- C10k = 10 000 connexions simultanées

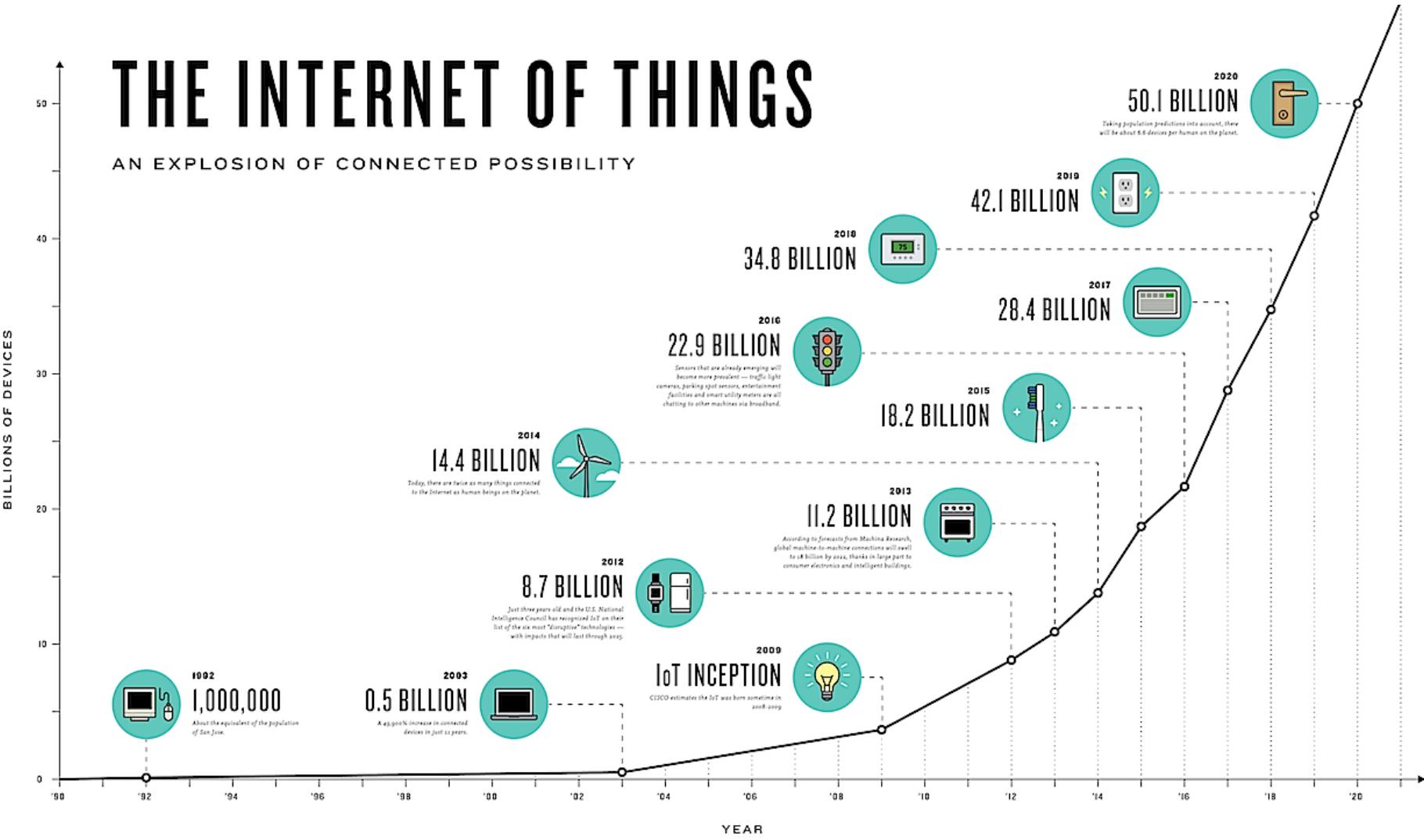
Figure 1. Five Traffic Milestones and Three Traffic Generator Milestones by 2015



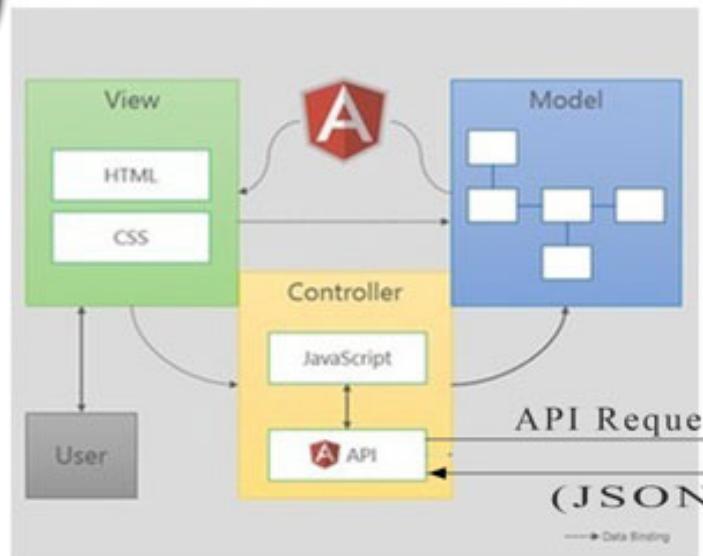
Vert.X - C10k



Vert.X - C10k

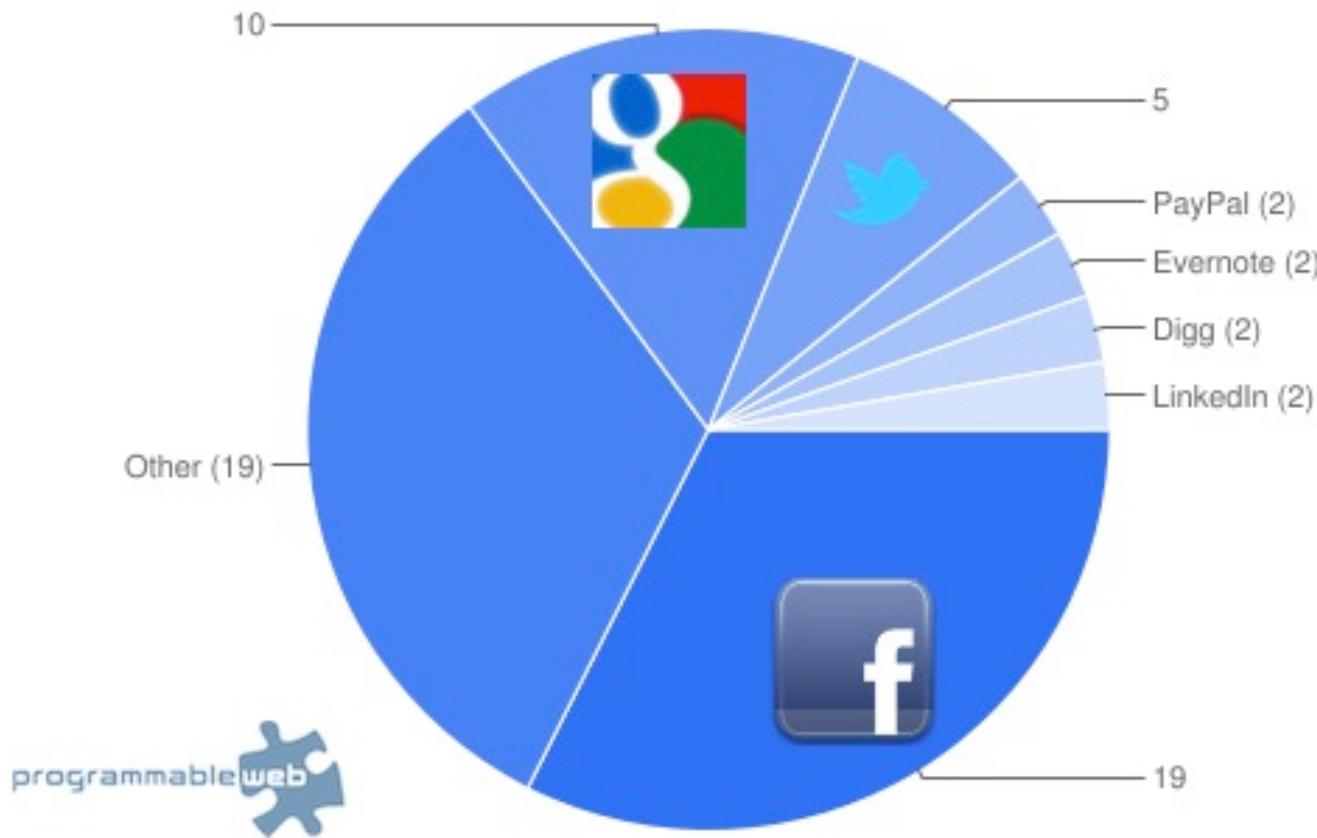


Vert.X - C10k



Vert.X - C10k

API Headaches

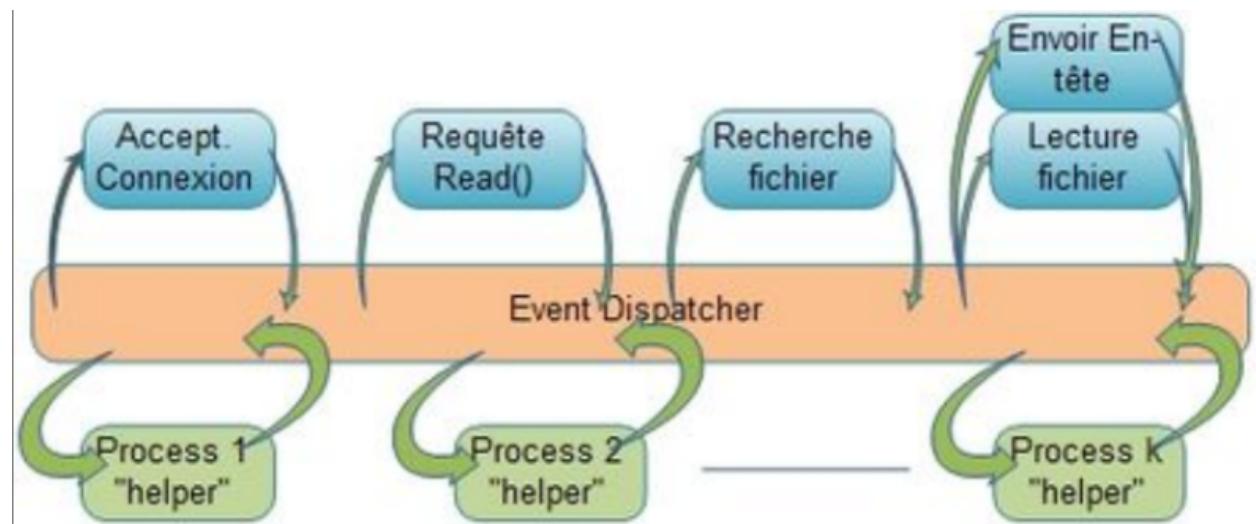


Vert.X - C10k



Vert.X - C10k

- scalable verticalement et horizontalement
- distribué
- événementiel et asynchrone
- I/O non bloquantes
- tolérant à la panne
- extensible



Vert.X - polyglotte

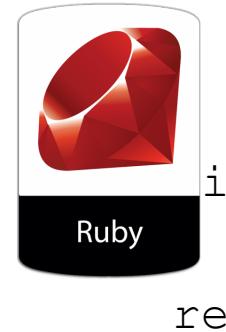


```
load('vertx.js');
vertx.createHttpServer().requestHandler(function(req) {
  'use strict';
  var file = req.path === '/' ? 'index.html' : req.path;
  req.response.sendFile('webroot/' + file);
}).listen(8080);
```

```
$> vertx run server.js -instances 32
```

Vert.X - polyglotte

```
require "vertx"
Vertx::HttpServer.new.request_handler do |req|
  file = req.uri == "/" ? "index.html" : req.uri
  req.response.send_file "webroot/#{file}"
end.listen(8080);
```



```
$> vertx run server.rb -instances 32
```

Vert.X - polyglotte

```
import vertx
server = vertx.create_http_server()
@server.request_handler
def request_handler(req):
    file = "index.html" if req.uri == "/" else req.uri
    req.response.send_file("webroot/%s"%file)
server.listen(8080)
```



python™

```
$> vertx run server.py -instances 32
```

Vert.X - polyglotte



```
vertx.createHttpServer().requestHandler { req ->
    def file = req.uri == "/" ? "index.html" : req.uri
    req.response.sendFile "webroot/$file"
}.listen(8080)
```

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

Vert.X - polyglotte

```
import org.vertx.java.core.Handler;
import org.vertx.java.core.http.HttpServerRequest;
import org.vertx.java.deploy.Verticle;

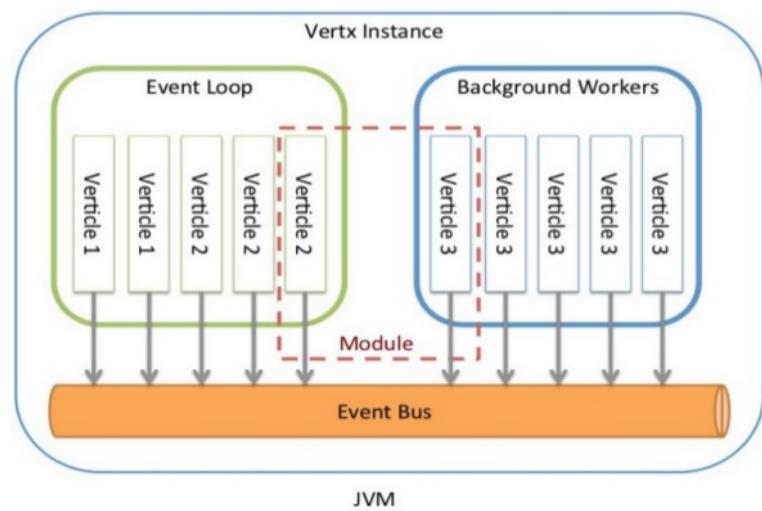
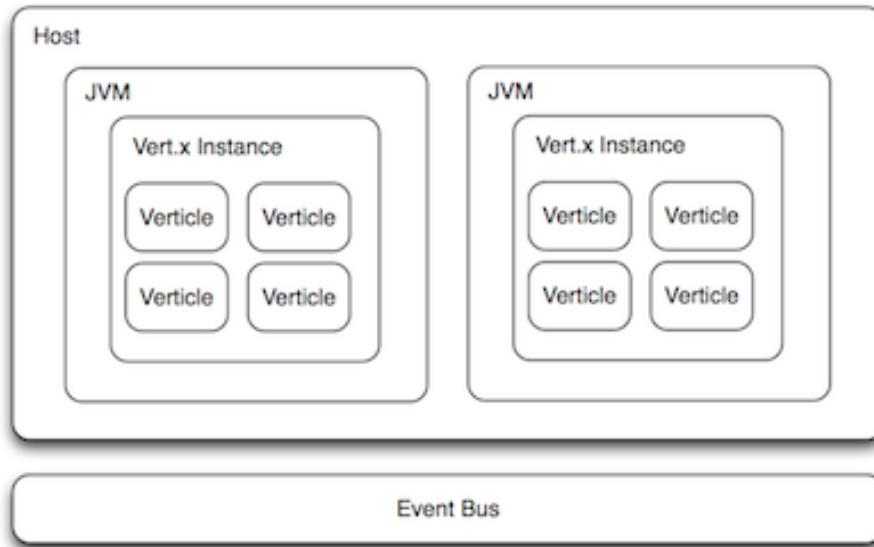
public class Server extends Verticle {
    public void start() {
        vertx.createHttpServer()
            .requestHandler(new Handler<HttpServerRequest>() {
                public void handle(HttpServerRequest req) {
                    String file = req.path.equals("/")
                        ? "index.html" : req.path;
                    req.response.sendFile("webroot/" + file);
                }
            }).listen(8080);
    }
}
```

```
$> vertx run Server.java -instances 32
```



Vert.X - Principes

Les **Verticles** packagés sous forme de **modules** s'exécutent dans une **instance** Vert.X.
Une **instance** de Vert.X s'exécute au sein de sa propre instance de **JVM**



Vert.X - Fonctionnalités

- Clients / Serveurs TCP/SSL
- Clients / Serveurs HTTP/HTTPS
- REST et Multipart
- Websockets et Sock.js
- Accès distribué de l'Event Bus
- Maps et Sets partagés
- Logging

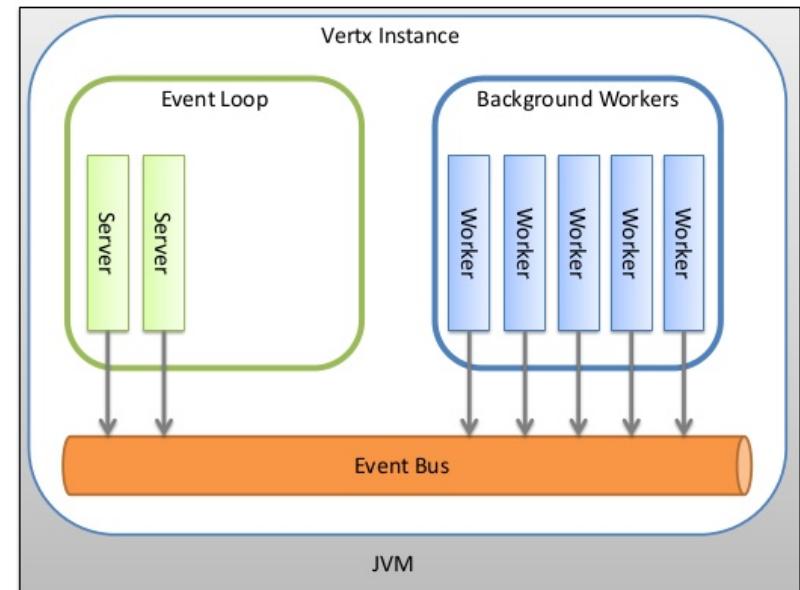
Vert.X - Haute disponibilité

- Gestion automatique du fail-over
- Quand un module tombe sur un noeud, il est automatiquement redémarré sur un autre noeud du cluster

```
$> vertx --ha
```

Vert.X - Event bus

- Permet aux Verticles de communiquer
- Agnostique du langage utilisé
- Fonctionne au travers du cluster
- S'étend jusqu'à la partie client
- Types de messages
 - number
 - string
 - boolean
 - JSON object
 - Vert.x Buffer



Vert.X - Event bus

- Enregistrer un handler
- Dé-référencer un handler
- Adresses
- Messages
 - Publish / subscribe
 - Point à point

Vert.X - Event bus

Enregistrer un handler

```
var eb = vertx.eventBus;  
var myHandler = function(message) {  
    log.info('j\'ai reçu un message ' + message);  
}  
  
eb.registerHandler('test.address', myHandler);  
eb.unregisterHandler('test.address', myHandler);
```

Vert.X - Event bus

Publish Subscribe :

```
eb.publish('test.address', 'hello world');
```

Point à point :

```
eb.send('test.address', 'hello world');
```

Vert.X - Event bus

RPC Server :

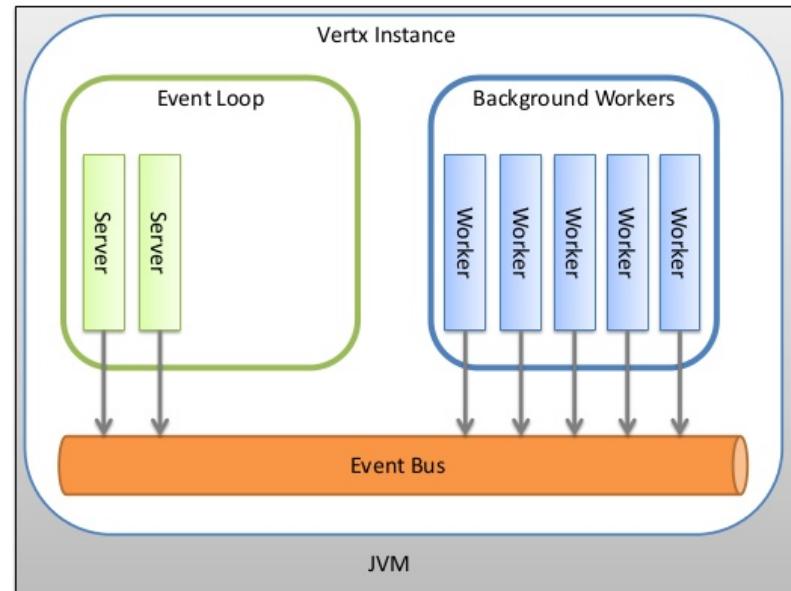
```
var myHandler = function(message, replier) {  
    log.info('J\'ai reçu un message : ' + message);  
    // Faire des trucs sympa  
    // Et répondre  
    replier('Ma réponse');  
}  
eb.registerHandler('test.address', myHandler);
```

RPC Client

```
eb.send('test.address', 'Ceci n\'est pas un message',  
    function(reply) {  
        log.info('J\'ai une réponse : ' + reply);  
    }) ;
```

Vert.X - Verticles

- **Verticle**
 - s'exécute dans le thread principal
 - pas d'opération blocante
- **WorkerVerticle**
 - s'exécute dans son propre ClassLoader
 - pool de threads paramétrable



Vert.X - Verticles

```
JsonObject appConfig = container.config();
JsonObject verticle1Config = appConfig.getObject( "verticle1_conf" );
JsonObject verticle2Config = appConfig.getObject( "verticle2_conf" );
JsonObject verticle3Config = appConfig.getObject( "verticle3_conf" );
JsonObject verticle4Config = appConfig.getObject( "verticle4_conf" );
JsonObject verticle5Config = appConfig.getObject( "verticle5_conf" );

// Start the verticles that make up the app
container.deployVerticle( "verticle1.js" , verticle1Config);
container.deployVerticle( "verticle2.rb" , verticle2Config);
container.deployVerticle( "foo.Verticle3" , verticle3Config);
container.deployWorkerVerticle( "foo.Verticle4" , verticle4Config);
container.deployWorkerVerticle( "verticle5.js" , verticle5Config, 10);
```

Vert.X - Verticles

```
container.deployVerticle( "foo.ChildVerticle" , new
AsyncResultHandler<String>() {
    public void handle(AsyncResult<String> asyncResult) {
        if (asyncResult.succeeded()) {
            System.out.println( "Le verticle a été déployé, son id : "
                + asyncResult
        } else {
            asyncResult.cause().printStackTrace();
        }
    }
}) ;
```

Truc et astuces : il existe le **vertx-when**

Vert.X - modules

Distribution d'un ensemble de verticles

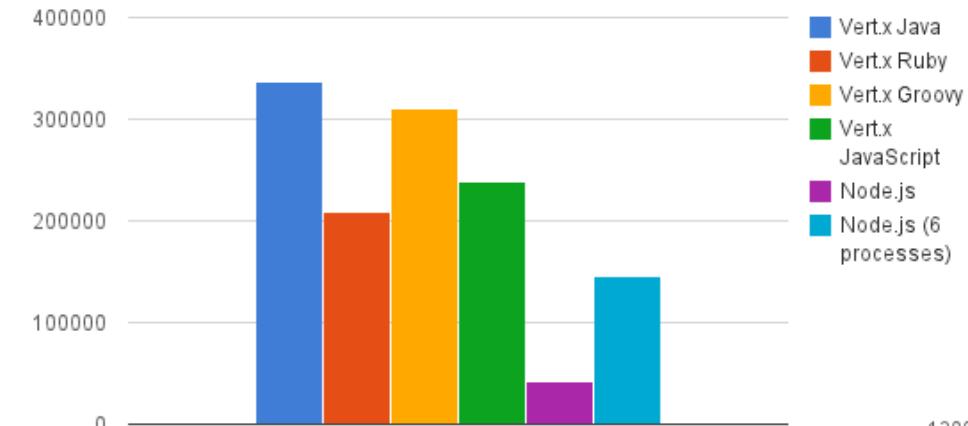
- Fichier Zip
- Configuration JSON externalisée
- Registry public : <http://modulereg.vertx.io>

```
// En ligne de commande
$> vertx runMod monModule.zip --conf=config.json
// Depuis un verticle
container.deployModule("io.vertx~mod-mailer~2.0.0-beta1", config);
```

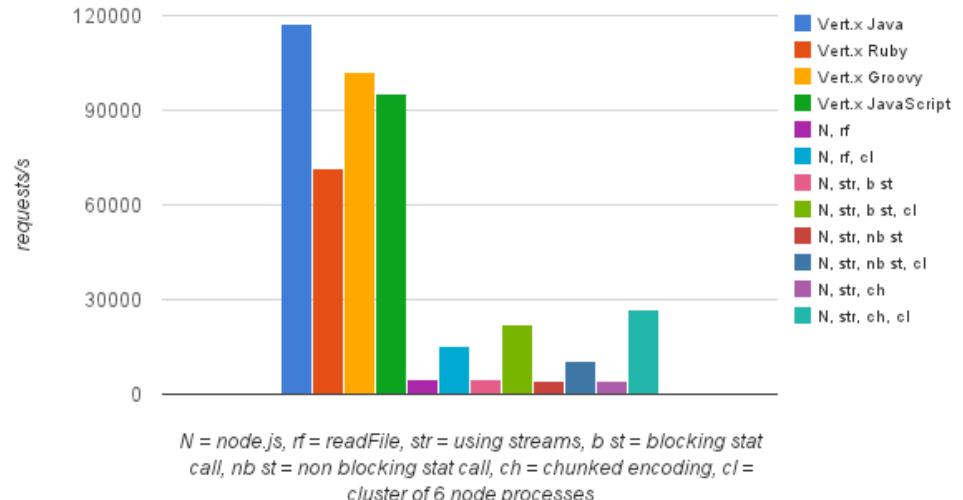
Truc et astuces : il existe le **vertx-when**

Vert.X - benchmarks

Test 1 - Server returns 200-OK - Single processes



Test 2 - Serve small static file - Single processes



Source : <http://www.cubrid.org>

Vert.X - les bonus

Vous aimez Node.js et Express?

<http://pmlopes.github.io/yoke>

APEX arrive avec Vert.X 3

```
Yoke yoke = new Yoke(vertx)
    .use(new Favicon())
    .use(new Static("webroot"))
    .use(new Router())
    .all("/hello", new Handler<HttpServerRequest>() {
        @Override
        public void handle(HttpServerRequest request)
            request.response().end("Hello World!");
    })
)).listen(8080);
```

The end ...



« Si vous avez compris ce que je viens de vous dire, c'est que je me suis probablement mal exprimé »

A. Greenspan