Aprameyah
Technologies Pvt. Ltd.

MAD Adv I

About Me

- Maruthi R Janardhan
  - Been with IBM, ANZ, HCL-HP, my own startup Leviossa..
  - Total 16 years programming/consulting C, C++, Java, Javascript, Ruby, Perl, Python, PHP, etc
  - Two pending Indian Patents

Polyglot Databases

- Understand Consistency Models - ACID/BASE
- Whats driving us to this
- Benefits
- Challenges
- Deciding Criteria

ACID/BASE

- Limits of instantaneous consistency
  - The balance between write performance and availability in instantaneous consistent systems
- What does Eventual Consistency get us
- CAP Theorem

Deciding Criteria

- Write performance
- Read performance
- Availability
- Consistency
- Partition Tolerance
- Query patterns
    - Text search, graph search

Types of DBs

- Mongo
    - Tuneable consistency, Availability, Partition Tolerance.
    - Indexed Document Store
    - Master slave model with writes only to master - Easy scalability for reads
- Neo4j
    - Graph database tuned for graph questions
    - Consistent and Available. Billions of nodes and relationships

Types of DB

- Cassandra
    - Column store with multi master ability. Closest to SQL
    - Available and Partition Tolerant
    - Easy scalability for writes
- Redis in-memory key-value store

Question of Durability

- What happens to backups
- How facebook handles redundancy and durability

Aprameyah
Technologies Pvt. Ltd.

Threading and Locking

- Understand thread racing
- Solving thread racing with locks
- Using atomic variables

Shared Collections

- Using shared collections
  - Write and read operations. Try multiple thread read and write into a synchronized map
  - Compare it with concurrent map implementation
- Fail fast and snapshot iterators
- CopyOnWriteArrayLists

I/O Performance

- Java IO library read operation
- Asynchronous Read using NIO
- Measure performance in multi threaded environment

Async Programming Model

- NodeJs
  - Demo of code
- Create a web project and perform 3 chained AJAX calls
- Performance of Async vs Sync code

Node.js

- Node.js is server side framework for javascript that runs on google's chrome V8 engines.
- Node.js is single threaded event based server framework
- Comes with a lot of functionality "compiled" and bundled into the runtime beyond the basic JS spec

Modules

- All of Node’s functionality is bundled into modules and we have to require the modules to be able to use them.

- “fs” module contains a lot of operations related to the filesystem.

- Using the below functions, write a function that gets a list of files in a directory and another to make directory (http://nodejs.org/api/fs.html)

```
var fs = require(“fs");
fs.existsSync(path);
fs.statSync(path);
fs.readdirSync(path);
fs.mkdirSync(newDir);
```

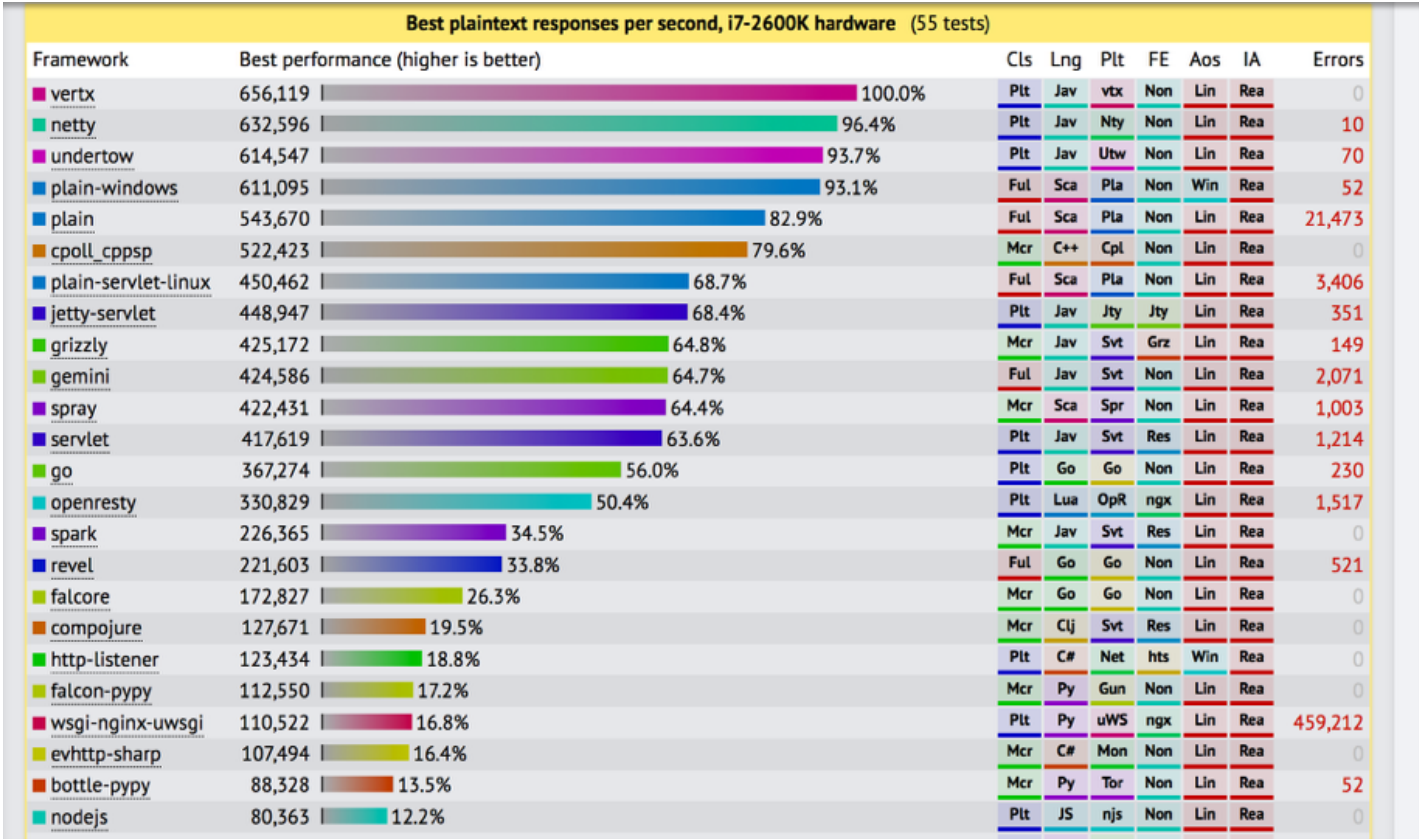
```
function getFilesInDir(path,callback){
    fs.exists(path, function(exists){
        if(exists){
            console.log("Checking if directory...");
            function statFunction(err, stat){
                if(err!==null){
                    console.log("Error");
                    callback(err);
                    return;
                }
                if(!stat.isDirectory()){
                    console.log("Is not a directory");
                    callback("Is not a directory");
                }
                console.log("Reading directory...");
                fs.readdir(path, function(err, filesArr){
                    if(err!==null){
                        console.log("Error");
                        callback(err);
                        return;
                    }
                    console.log(filesArr);
                    callback(filesArr);
                });
            }
            fs.stat(path, statFunction);
        }else{
            console.log("Path does not exist");
            callback("Path does not exist");
        }
    });
}
```

- Create a simple dynamic web project with 3 hardcoded json
- Create sequential AJAX calls using jquery

Vert.x

- We can build reactive, non blocking, event driven apps
- Polyglot - javascript, java, groovy and ruby
- Not an app server, modular
- Good fit for microservices

Vert.x Performance

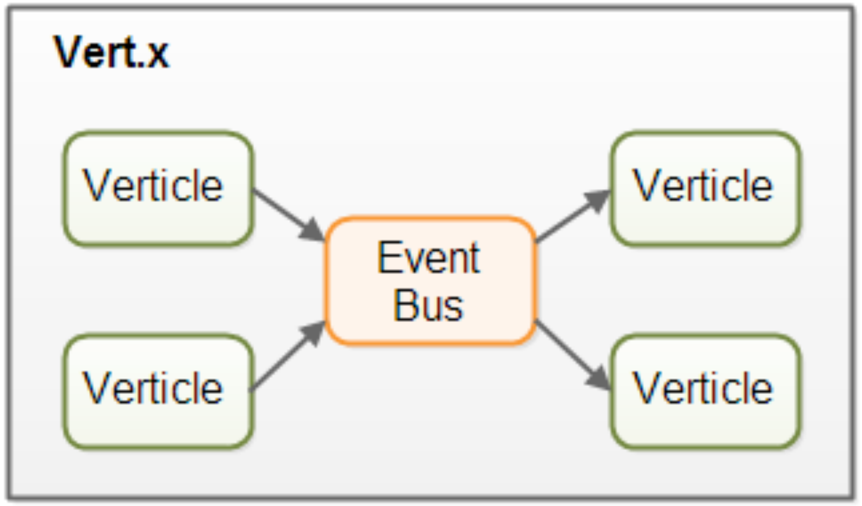


Best plaintext responses per second, i7-2600K hardware (55 tests)

Framework	Best performance (higher is better)		Cls	Lng	Plt	FE	Aos	IA	Errors
vertx	656,119	100.0%	Plt	Jav	vtx	Non	Lin	Rea	0
netty	632,596	96.4%	Plt	Jav	Nty	Non	Lin	Rea	10
undertow	614,547	93.7%	Plt	Jav	Utw	Non	Lin	Rea	70
plain-windows	611,095	93.1%	Ful	Sca	Pla	Non	Win	Rea	52
plain	543,670	82.9%	Ful	Sca	Pla	Non	Lin	Rea	21,473
cpoll_cppsp	522,423	79.6%	Mcr	C++	Cpl	Non	Lin	Rea	0
plain-servlet-linux	450,462	68.7%	Ful	Sca	Pla	Non	Lin	Rea	3,406
jetty-servlet	448,947	68.4%	Plt	Jav	Jty	Jty	Lin	Rea	351
grizzly	425,172	64.8%	Mcr	Jav	Svt	Grz	Lin	Rea	149
gemini	424,586	64.7%	Ful	Jav	Svt	Non	Lin	Rea	2,071
spray	422,431	64.4%	Mcr	Sca	Spr	Non	Lin	Rea	1,003
servlet	417,619	63.6%	Plt	Jav	Svt	Res	Lin	Rea	1,214
go	367,274	56.0%	Plt	Go	Go	Non	Lin	Rea	230
openresty	330,829	50.4%	Plt	Lua	OpR	ngx	Lin	Rea	1,517
spark	226,365	34.5%	Mcr	Jav	Svt	Res	Lin	Rea	0
revel	221,603	33.8%	Ful	Go	Go	Non	Lin	Rea	521
falcore	172,827	26.3%	Mcr	Go	Go	Non	Lin	Rea	0
compojure	127,671	19.5%	Mcr	Clj	Svt	Res	Lin	Rea	0
http-listener	123,434	18.8%	Plt	C#	Net	hts	Win	Rea	0
falcon-pypy	112,550	17.2%	Mcr	Py	Gun	Non	Lin	Rea	0
wsgi-nginx-uwsgi	110,522	16.8%	Plt	Py	uWS	ngx	Lin	Rea	459,212
evhttp-sharp	107,494	16.4%	Mcr	C#	Mon	Non	Lin	Rea	0
bottle-pypy	88,328	13.5%	Mcr	Py	Tor	Non	Lin	Rea	52
nodejs	80,363	12.2%	Plt	JS	njs	Non	Lin	Rea	0

Verticles

- Vert.x can deploy and execute components called Verticles.
- You can think of verticles as being similar to servlets or message driven EJBs driven by an event bus



Verticle Messaging

- Messages can be simple objects (e.g. Java objects), strings, CSV, JSON, binary data or whatever else you need

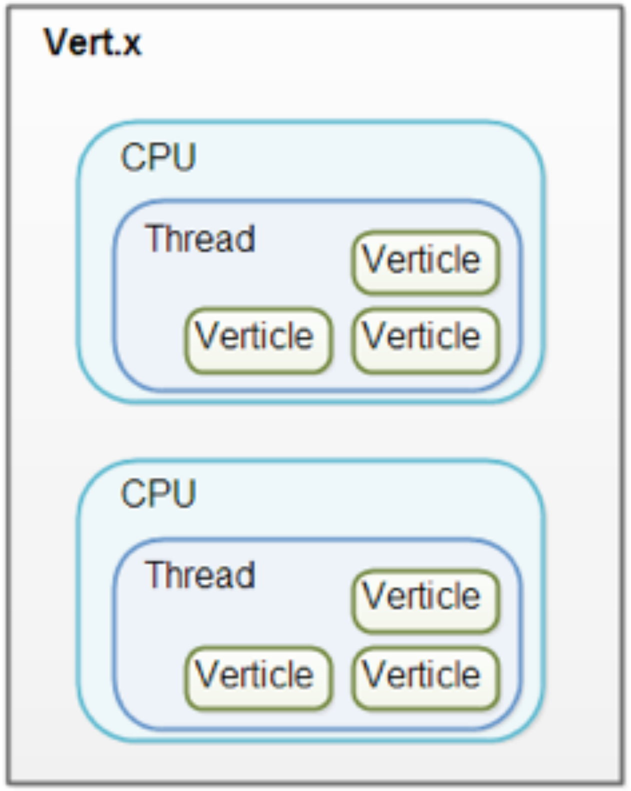
- Verticles can send and listen to addresses. An address is like a named channel.

- When a message is sent to a given address, all verticles that listen on that address receive the message.

- Verticles can subscribe and unsubscribe to addresses without the senders knowing.

Threading Model

- verticle is only ever executed by a single thread, and always by the same thread.
- A single thread can distribute messages to multiple verticles.
- Vert.x creates one thread per CPU
- Vert.x comes with a set of built-in services (functionality). Some of these services are:
-     HTTP server
-     JDBC connector
-     MongoDB connector



Number of Instances

- Deploy multiple verticle instances to run in different threads
  - DeploymentOptions options = new DeploymentOptions().setInstances(16);

Creating Verticles

```
public class MyVerticle extends AbstractVerticle {

    @Override
    public void start(Future<Void> startFuture) {
        System.out.println("MyVerticle started!");
    }

    @Override
    public void stop(Future stopFuture) throws Exception {
        System.out.println("MyVerticle stopped!");
    }

}

public static void main(String[] args) {
      VertxOptions options = new VertxOptions().setWorkerPoolSize(10);
      Vertx vertx = Vertx.vertx(options);
      vertx.deployVerticle("com.mydomain.MyVerticle");
}
```

Verticle Events

- The start() method : start HTTP or TCP server, register event handlers on the event bus, deploy other verticles, or whatever else your verticle needs to do its work.

- Shutdown stuff in stop method

- The verticle will be deployed asynchronously

```
      vertx.deployVerticle("com.mydomain.MyVerticle",new
Handler<AsyncResult<String>>() {
            @Override
            public void handle(AsyncResult<String> stringAsyncResult) {
                System.out.println("Verticle deployment complete");
                }
            });
```

Registering to Events

- When a verticle wants to listen for messages from the event bus, it listens on a certain address. An address is just a name (a String) which you can choose freely.
- An address is thus more like the name of a channel with multiple receivers

```
vertx.eventBus().consumer("Channel1", message -> {
       System.out.println("message.body() = "
                + message.body());
});
vertx.eventBus().publish("Channel1", "message 2");
```

Types Of Verticles

- Standard Verticles
    - These are the most common and useful type - they are always executed using an event loop thread.
- Worker Verticles
    - These run using a thread from the worker pool. An instance is never executed concurrently by more than one thread.

```
DeploymentOptions options = new DeploymentOptions().setWorker(true);

vertx.deployVerticle("com.mycompany.MyOrderProcessorVerticle", options);
```

- Multi-threaded worker verticles
    - These run using a thread from the worker pool. An instance can be executed concurrently by more than one thread.

Vert.x buffers

- Carry Binary Information in buffers. Dynamically resizable

- Can be used as message payloads

```
byte[] initialData = new byte[]{1, 2, 3};

Buffer buffer     = Buffer.buffer(initialData);
buffer.setShort ( 10, (short) 127);
buffer.appendByte  ((byte)  127);
```

Running Blocking Code

- When we HAVE to invoke synchronous APIs, vertx provides a way to do that (Executes using a thread from worker pool):

```
vertx.executeBlocking(future -> {
      // Call some blocking API that takes a significant amount of time
to return
      String result = someAPI.blockingMethod("hello");
      future.complete(result);
   }, res -> {
     System.out.println("The result is: " + res.result());
});
```

Vertx Web

- Vertx is bundled with a router

```
HttpServer server = vertx.createHttpServer();
Router router = Router.router(vertx);
router.get("/services/users/:id").handler(new UserLoader());
server.requestHandler(router::accept).listen(8080);
```

- In the handler

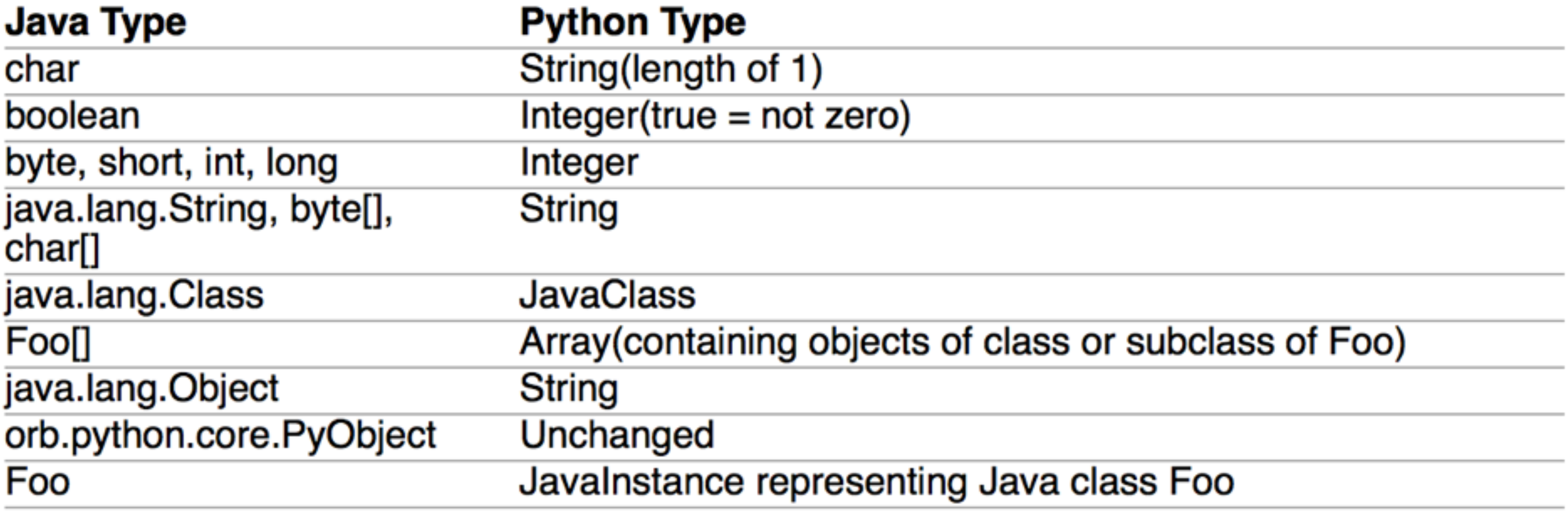
```
String id = routingContext.request().getParam(“id”);
HttpServerResponse response = routingContext.response();
response.putHeader("content-type", “application/json");
response.end(jsonresponse)
```

Scripting Integration

- JSR 223 allows for different scripting languages to be integrated into Java
    - javascript (nashhorn engine)
    - python (jython interpreter)
    - ruby (jruby)

Data Type Mapping

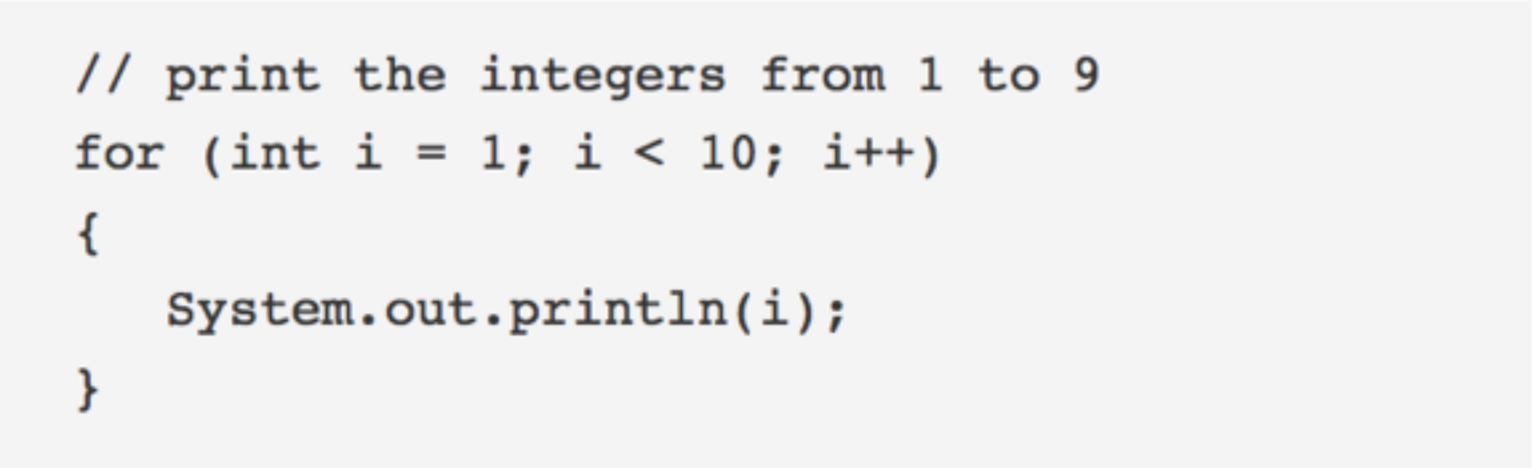
- Usually some data type mapping is defined. Here is a def for Jython



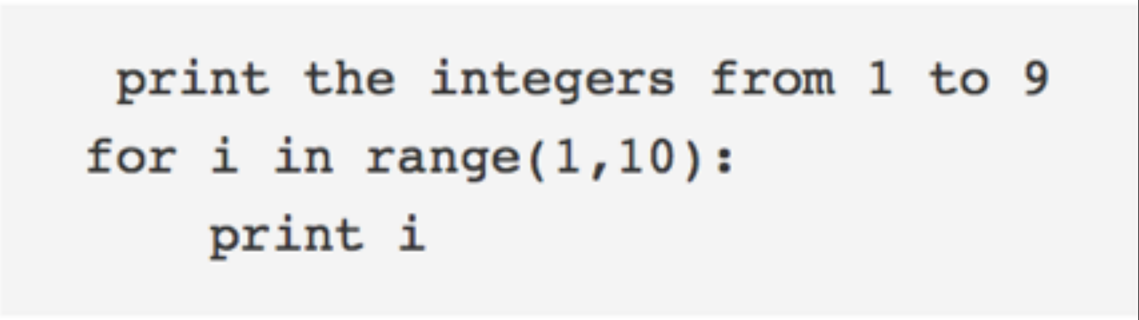
Java Type	Python Type
char	String(length of 1)
boolean	Integer(true = not zero)
byte, short, int, long	Integer
java.lang.String, byte[], char[]	String
java.lang.Class	JavaClass
Foo[]	Array(containing objects of class or subclass of Foo)
java.lang.Object	String
orb.python.core.PyObject	Unchanged
Foo	JavaInstance representing Java class Foo

Aprameyah
Technologies Pvt. Ltd.

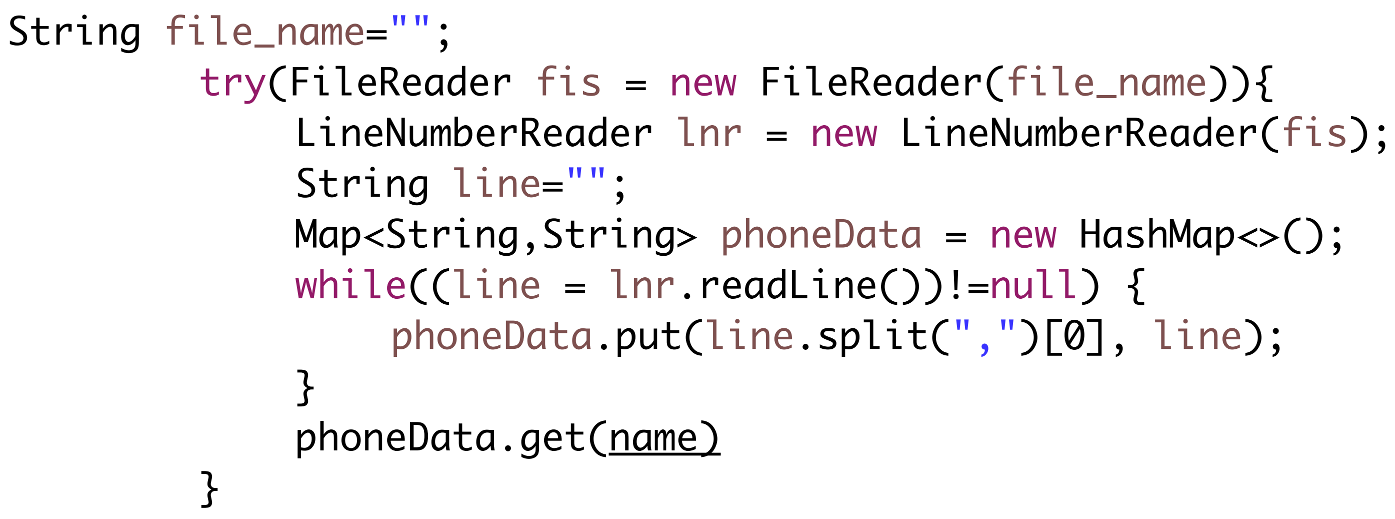
Concise Syntax of Python



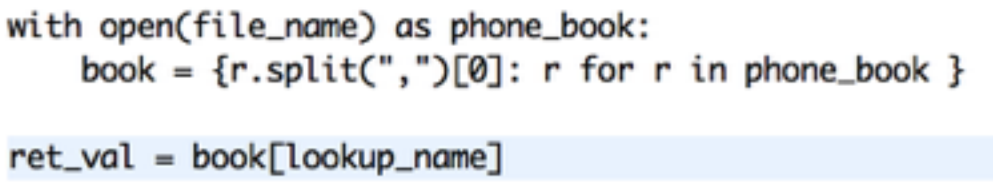
```
// print the integers from 1 to 9
for (int i = 1; i < 10; i++)
{
   System.out.println(i);
}
```



```
 print the integers from 1 to 9
for i in range(1,10):
    print i
```



```
String file_name="";
        try(FileReader fis = new FileReader(file_name)){
            LineNumberReader lnr = new LineNumberReader(fis);
            String line="";
            Map<String,String> phoneData = new HashMap<>();
            while((line = lnr.readLine())!=null) {
                phoneData.put(line.split(",")[0], line);
            }
            phoneData.get(name)
        }
```



```
with open(file_name) as phone_book:
    book = {r.split(",")[0]: r for r in phone_book }

ret_val = book[lookup_name]
```

Vertx on docker

- Create a Dockerfile
- docker build --tag restvertx .
- docker run -p 8080:8080 restvertx &
- Access docker host with IP: http://192.168.99.100:8080/api/user/2
  - docker stop restvertx
  - docker rm $(docker ps -a -q)

Dockerfile

```
# Extend vert.x image
FROM vertx/vertx3

#
ENV VERTICLE_NAME com.mydomain.myapp.RestAppVerticle
ENV VERTICLE_FILE target/VertxREST-0.0.1-SNAPSHOT.jar

# Set the location of the verticles
ENV VERTICLE_HOME /usr/verticles

EXPOSE 8080

# Copy your verticle to the container
COPY $VERTICLE_FILE $VERTICLE_HOME/

# Launch the verticle
WORKDIR $VERTICLE_HOME
ENTRYPOINT ["sh", "-c"]
CMD ["vertx run $VERTICLE_NAME -cp $VERTICLE_HOME/*"]
```

Vertx Web Handling User Data

```
router.post("/services/users").handler(new UserPersister());
```

- In Handler

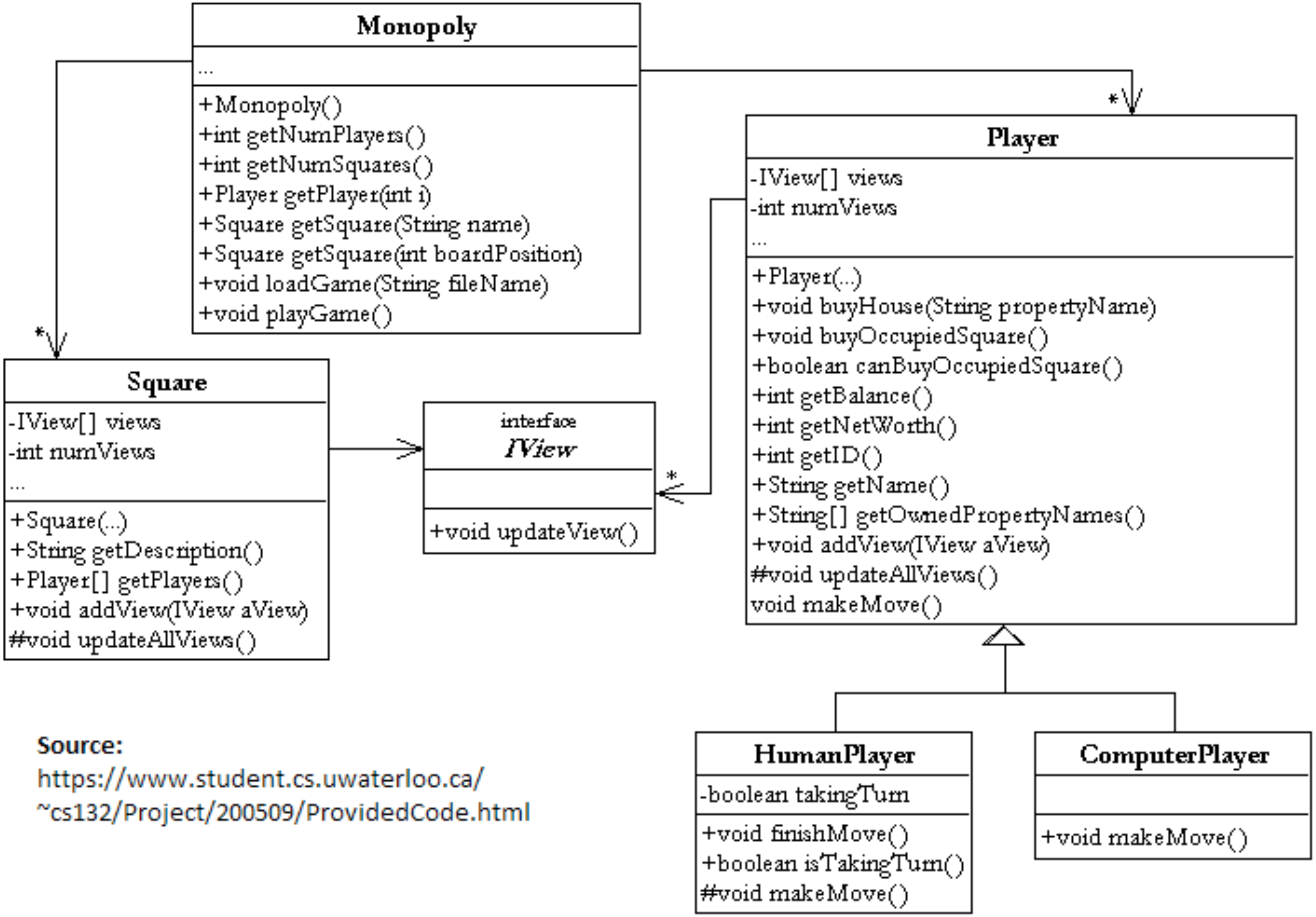
```
routingContext.request().bodyHandler(bodyHandler)
```

- In Body Handler

```
public void handle(Buffer buf) {
String json = buf.toString(“UTF-8");
response.setStatusCode(204).end("Data saved");
```

Apprameyah
Technologies Pvt. Ltd.

Traditional OO Design



Monopoly

...

+Monopoly()
+int getNumPlayers()
+int getNumSquares()
+Player getPlayer(int i)
+Square getSquare(String name)
+Square getSquare(int boardPosition)
+void loadGame(String fileName)
+void playGame()

Player

-IView[] views
-int numViews
...

+Player(...)
+void buyHouse(String propertyName)
+void buyOccupiedSquare()
+boolean canBuyOccupiedSquare()
+int getBalance()
+int getNetWorth()
+int getID()
+String getName()
+String[] getOwnedPropertyNames()
+void addView(IView aView)
#void updateAllViews()
void makeMove()

Square

-IView[] views
-int numViews
...

+Square(...)
+String getDescription()
+Player[] getPlayers()
+void addView(IView aView)
#void updateAllViews()

interface
IView

+void updateView()

HumanPlayer

-boolean takingTurn

+void finishMove()
+boolean isTakingTurn()
#void makeMove()

ComputerPlayer

+void makeMove()

Source:
https://www.student.cs.uwaterloo.ca/~cs132/Project/200509/ProvidedCode.html

Stateless OO Design Classes

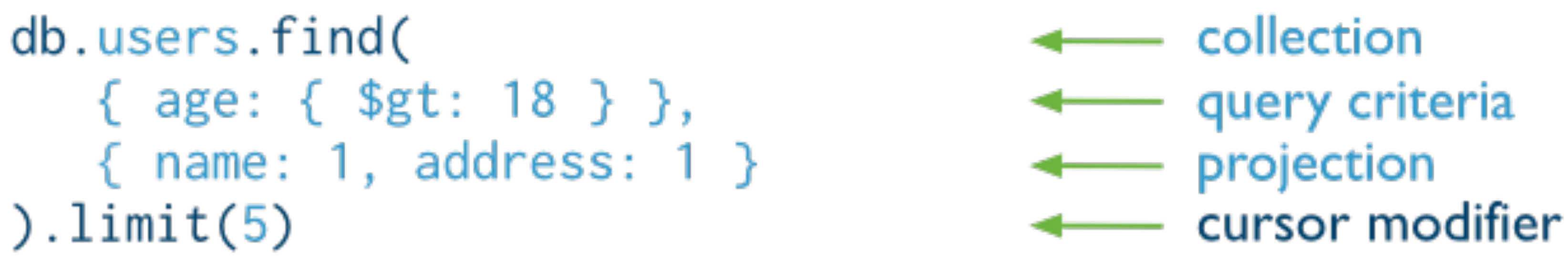
- Model Classes - No patterns, just DTOs
- Business Classes - Design patterns apply here
- Technology Classes
- Utility Classes

Mongodb

- Mongodb is an indexed document store.
- A document is typically something like a json structure
- Can be indexed based on any of the fields
- Can be replicated, shraded, tunable consistency with single master and leader election.

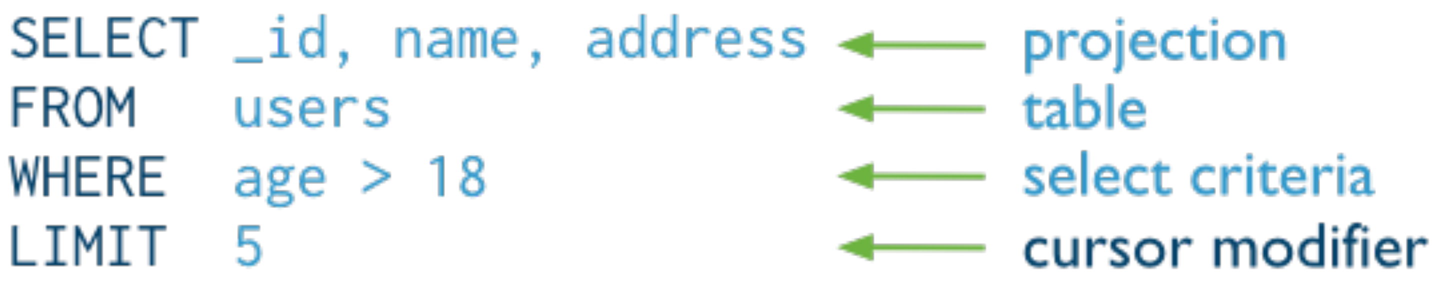
- db.users.insert({name:'Hari',email:'hari@abc.com'})
- show collections
- db.users.update({age: { $gt: 18}},{$set: {status:’A’}}, {multi: true})
- db.users.remove({status: ‘D’})

Mongodb Queries



```
db.users.find(
   { age: { $gt: 18 } },
   { name: 1, address: 1 }
).limit(5)
```

collection
query criteria
projection
cursor modifier



```
SELECT _id, name, address
FROM   users
WHERE  age > 18
LIMIT  5
```

projection
table
select criteria
cursor modifier

Creating Index

- db.users.createIndex({ email: 1 })
- db.users.createIndex({ email: 1, name: 1 })
- Fully covered queries:
    - db.users.find { email: /.*yahoo.com/},{ name: 1, _id: 0 })

Integrating With Java

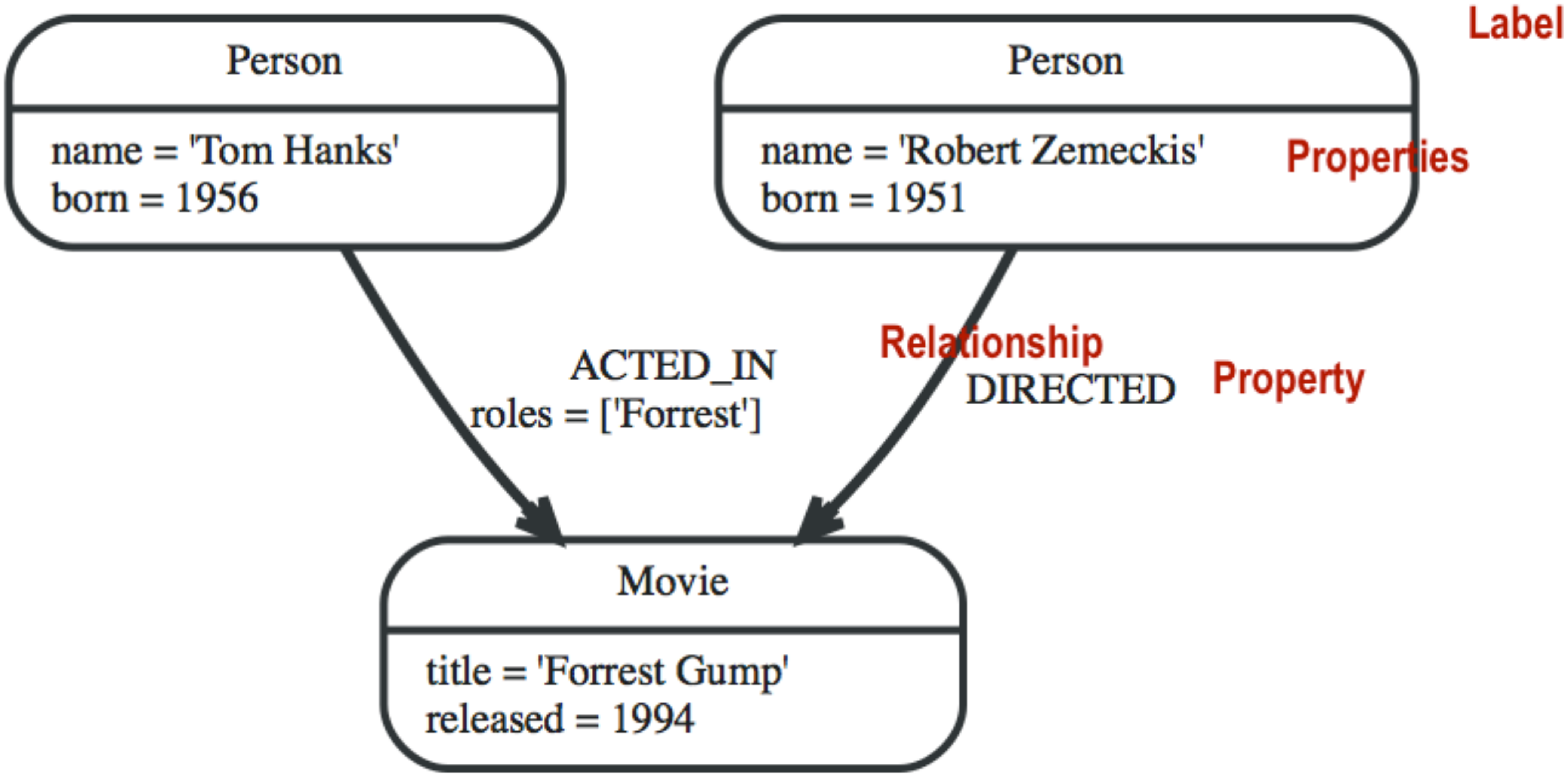
- POM dependencies

Neo4j

- ACID transactions,
- High availability,
- Scales to billions of nodes and relationships,
- High speed querying through traversals,
- Declarative graph query language.

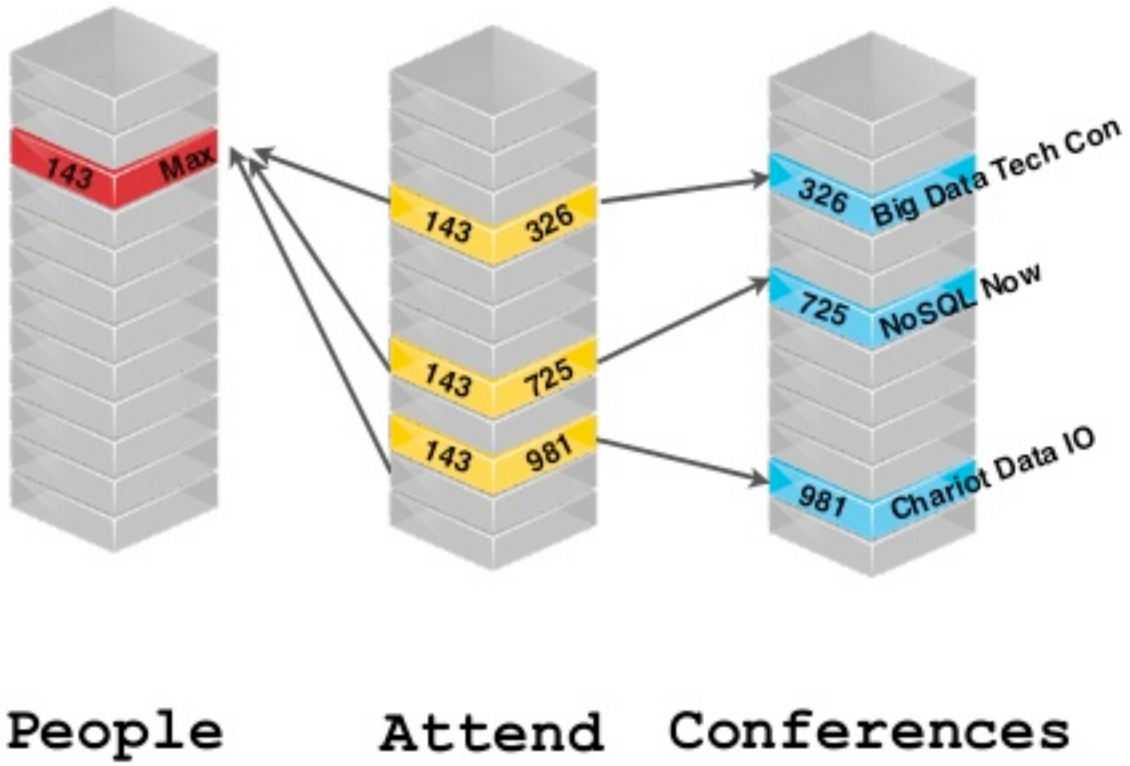
Aprameyah
Technologies Pvt. Ltd.

Graph

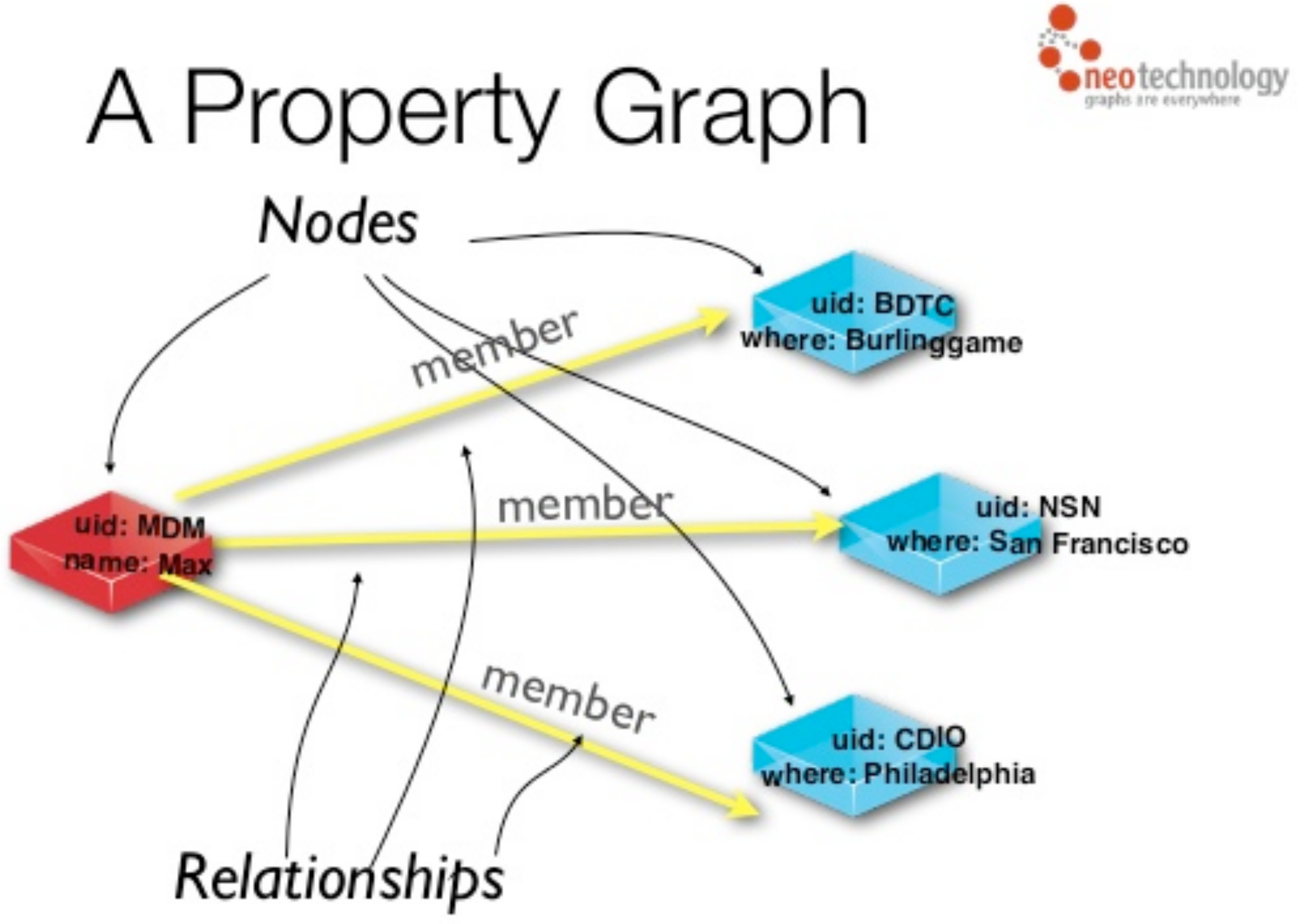


RDBMS

neotechnology
graphs are everywhere

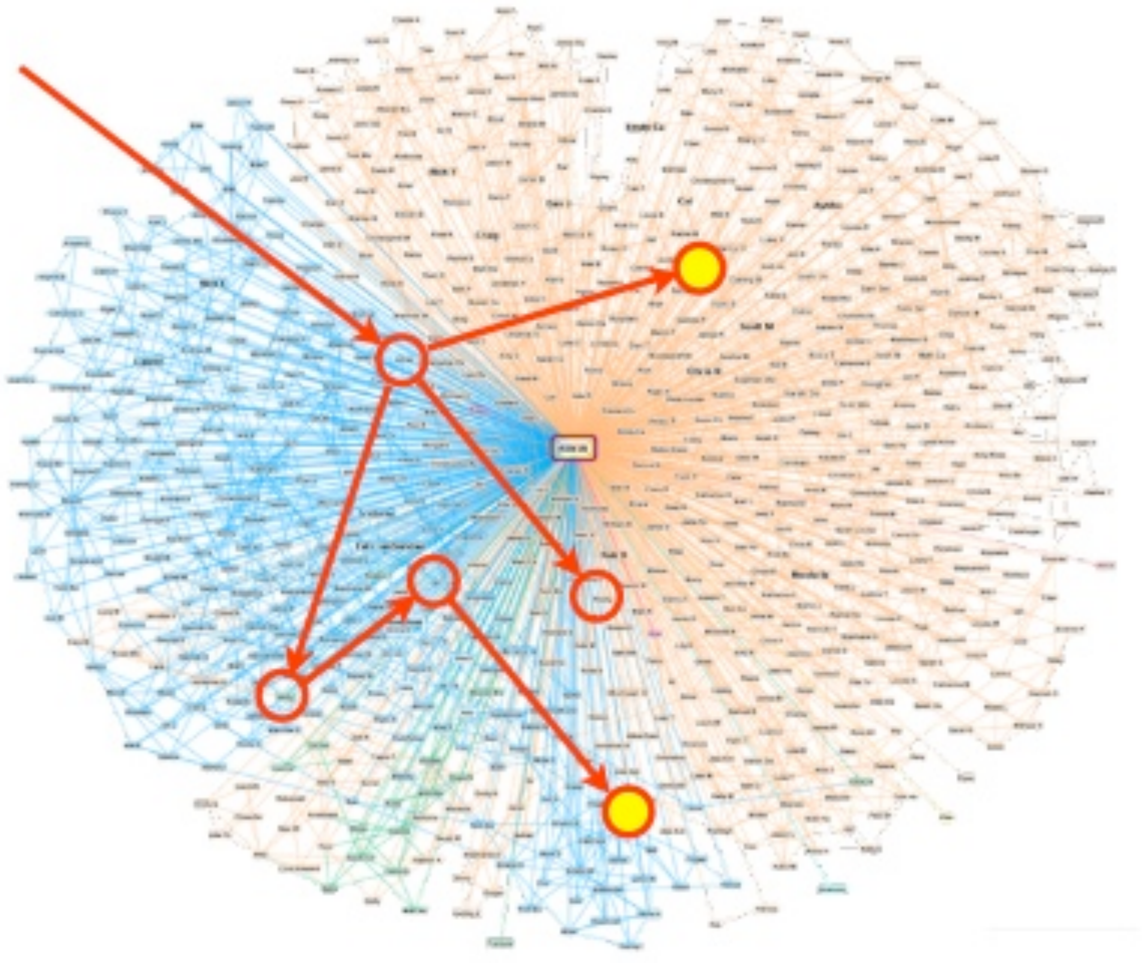


Graph

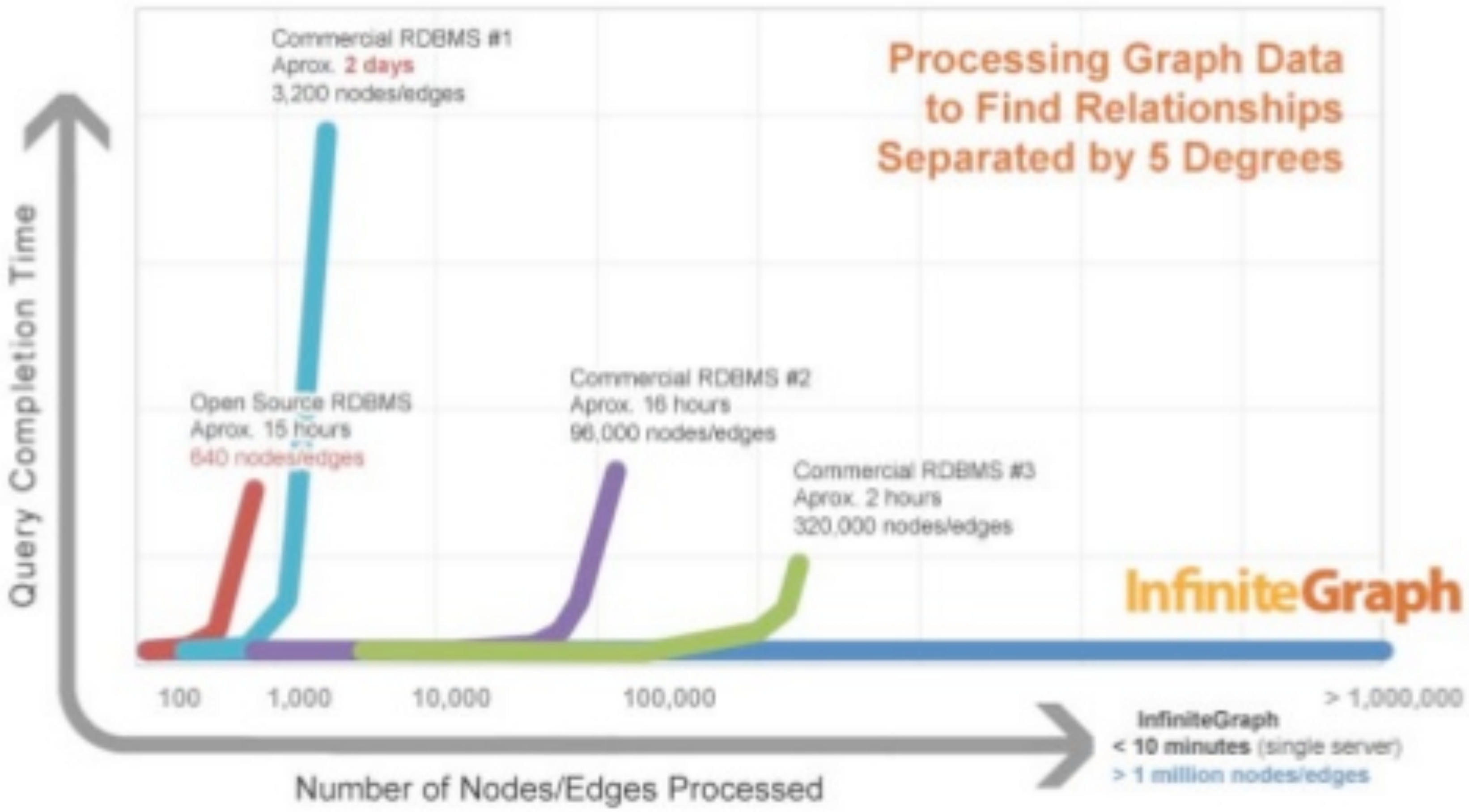

A Property Graph
neotechnology
graphs are everywhere
Nodes
uid: BDTC
where: Burlinggame
uid: MDM
name: Max
uid: NSN
where: San Francisco
uid: CDIO
where: Philadelphia
member
member
member
Relationships

Of course.. a graph is a graph is a graph

What drugs will bind to protein X and not interact with drug Y?



Why use a Graph Database?



Objectivity

InfiniteGraph
Objectivity/DB

Features

- SQL Like easy query language Neo4j CQL
- It supports Indexes by using Apache Lucence
- It supports UNIQUE constraints
- It contains a UI to execute CQL Commands : Neo4j Data Browser
- It supports full ACID
- It uses Native graph storage with Native GPE(Graph Processing Engine)
- It supports exporting of query data to JSON and XLS format

Features

- It provides REST API for the data
- It supports two kinds of Java API: Cypher API and Native Java API to develop Java applications.

CQL

- create (blog:Blog) - node name blog, label Blog
- create (:Blog{title:'Some blog',desc:'This is some blog’}) - Create with properties
- match(b:Blog) return b - select * from blog b
- MATCH (b:Blog),(u:User) CREATE (b)-[r:AUTHORED_BY]->(u)
    - Form relationship between EVERY blog and EVERY User
- MATCH (u:User{name:'Faizal'}) CREATE (blog:Blog{title:'related blog',desc:'This is some related blog'})-[r:AUTHORED_BY{when:'5th Sept'}]->(u)
    - Create an object with a relationship
- match(b:Blog)-[a:AUTHORED_BY]-(u:User{name:'Faizal'}) return a.when
    - Find all dates when Faizal authored blogs
- MATCH (b:Blog) WHERE b.title =~ ‘.*Some.*' RETURN b

CQL

- MATCH (u:User{name:'Faizal'}) CREATE (blog:Blog{title:'related blog',desc:'This is some related blog'})-[r:AUTHORED_BY{when:'5th Sept'}]->(u)

  - Create an object with a relationship

- match(b:Blog)-[a:AUTHORED_BY]-(u:User{name:'Faizal'}) return a.when

  - Find all dates when Faizal authored blogs

- MATCH (b:Blog) WHERE b.title =~ ‘.*Some.*' RETURN b

CQL

- MATCH (b:Blog)-[rel]-(u:User) DELETE rel
    - Delete only the relationships
- MATCH (u:User{name:’Faizal'}) SET u.sex = ‘Male' RETURN u
    - Set a field and return the node

Java API support

- Neo4J supports two types of Java APIs
  - Native Java API
  - CQL API