

Inter-Reactive Kotlin Applications



Julien Viet
[@julienviet](#)

Julien Viet

Open source developer for 15+ years

Current **@vertx_project** lead  **redhat.**

Principal software engineer at

Marseille Java User Group Leader

 <https://www.julienviet.com/>

 <http://github.com/vietj>

 @julienviet

Outline

- ✓ Reactive applications
- ✓ Going event driven
- ✓ Going interactive with coroutines
- ✓ Streaming with channels
- ✓ Coroutines vs RxJava

Workbook1

Q Search Sheet

😊

HomeInsertPage LayoutFormulasDataReviewView

Paste

Font

Alignment

Number

Conditional Formatting

Format as Table

Cell Styles

Cells

Editing

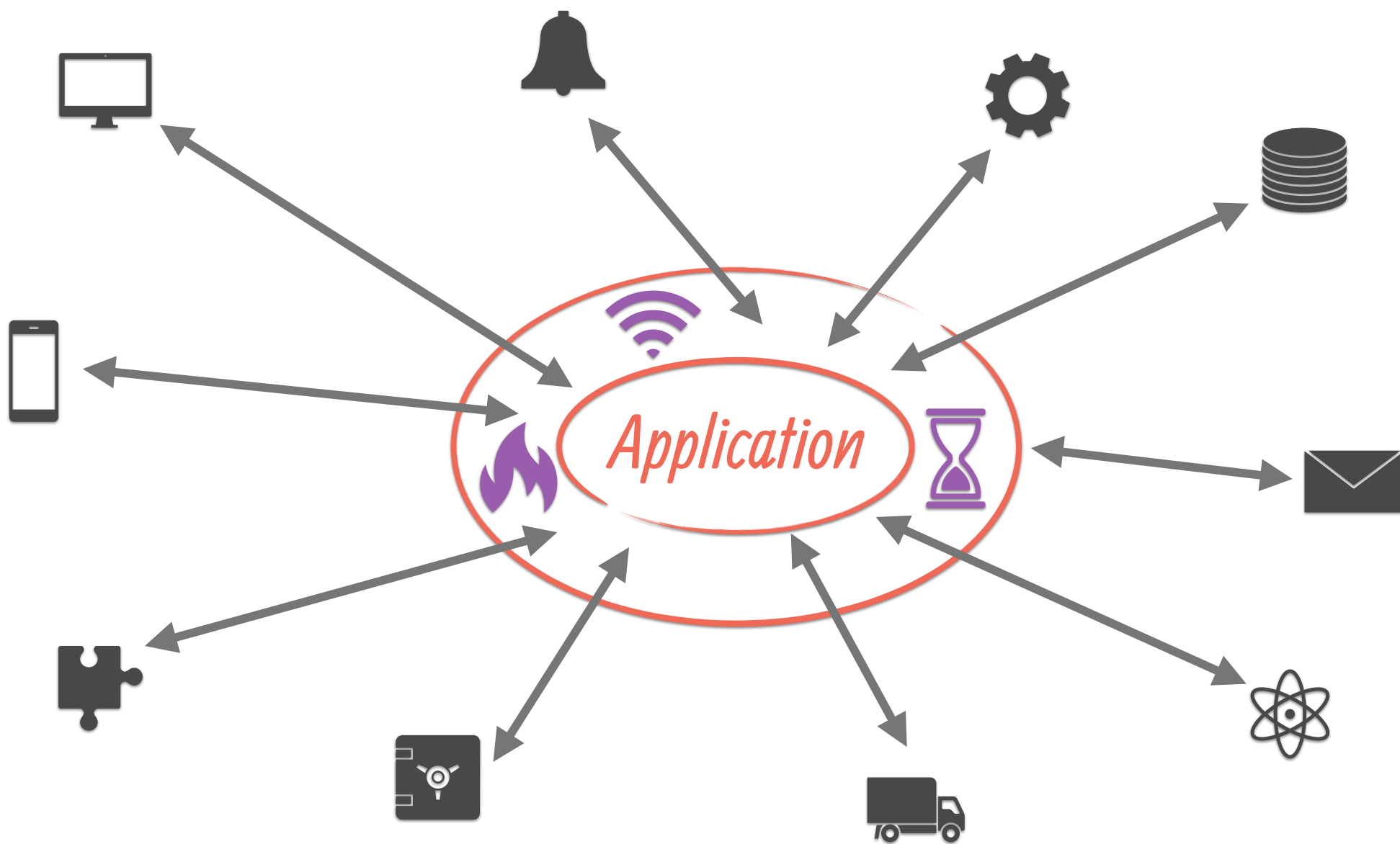
AVERAGE

✖

✔

fx

=A1+A2



Software

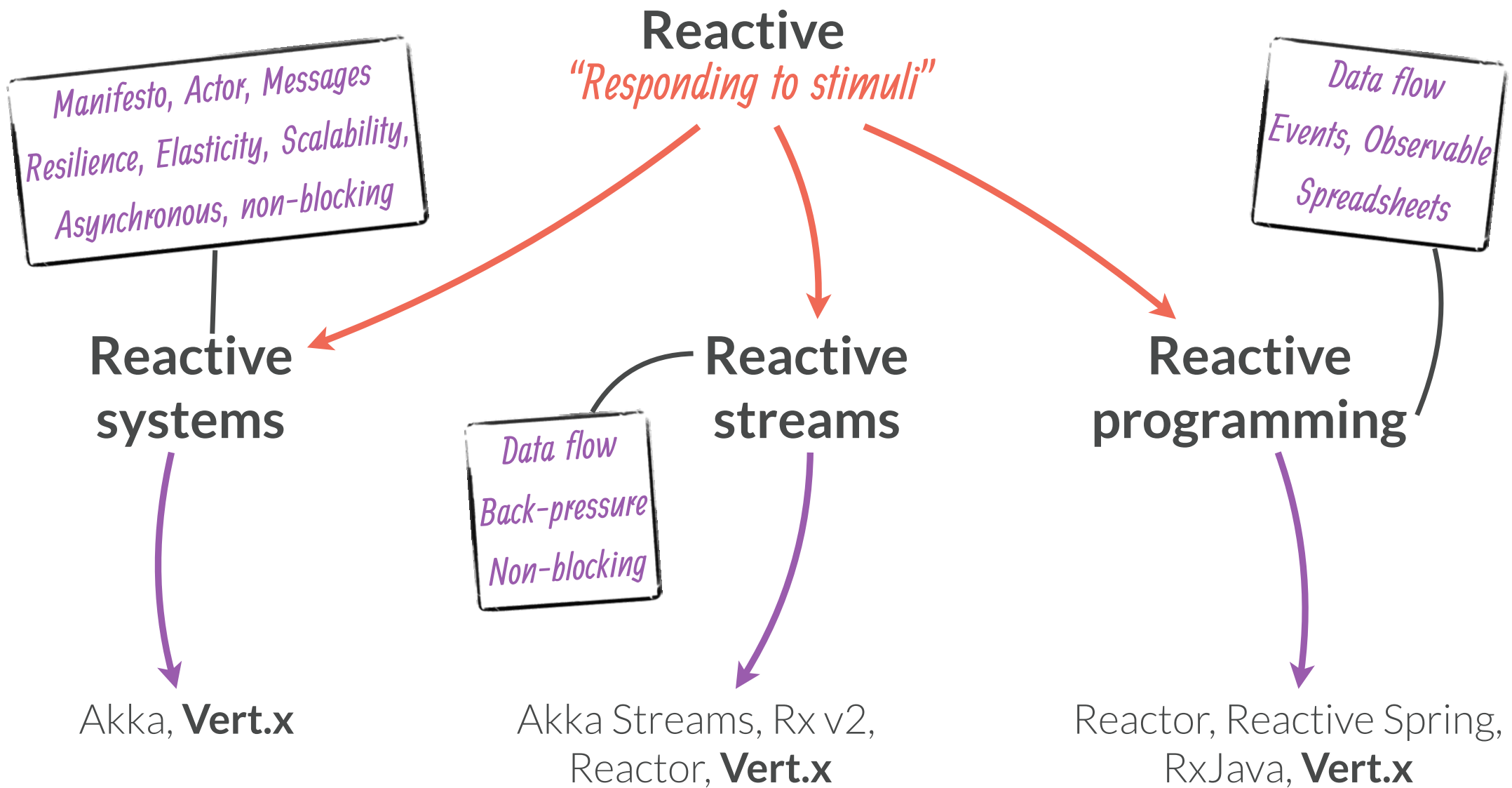


Metrics

Availability

Messages

Requests



Eclipse Vert.x

Open source project started in 2012

Eclipse / Apache licensing

A **toolkit** for building **reactive** applications for the JVM

7K ★ on 

Built on top of  Netty

 <https://vertx.io>

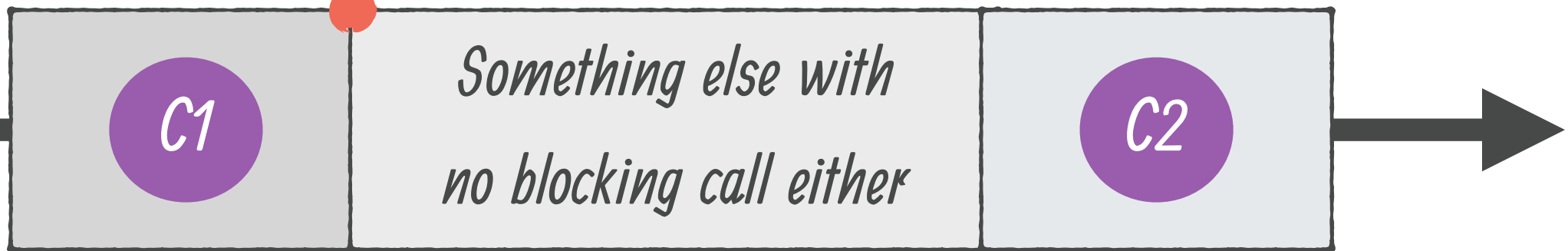
 @vertx_project

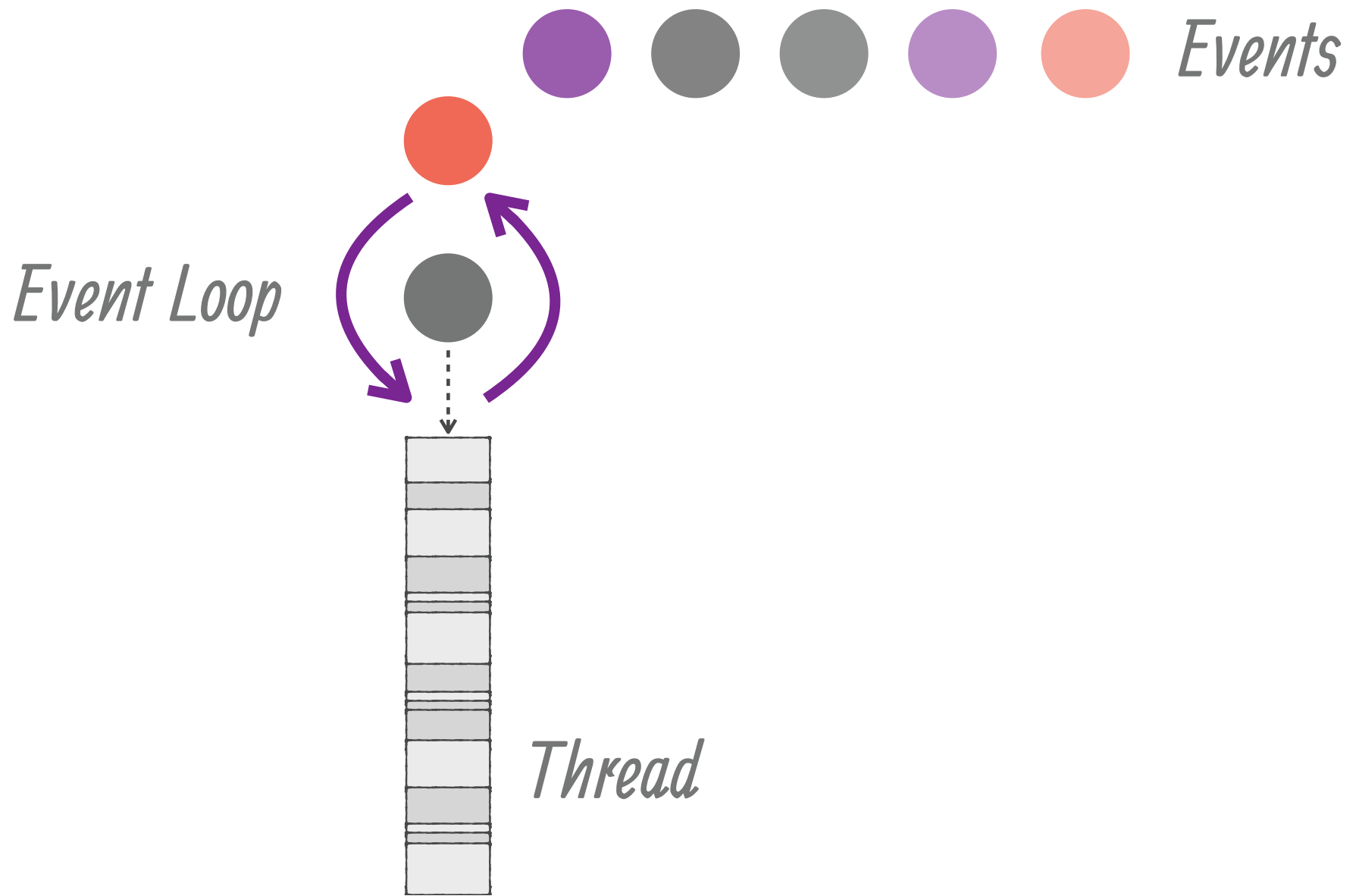
Going event driven

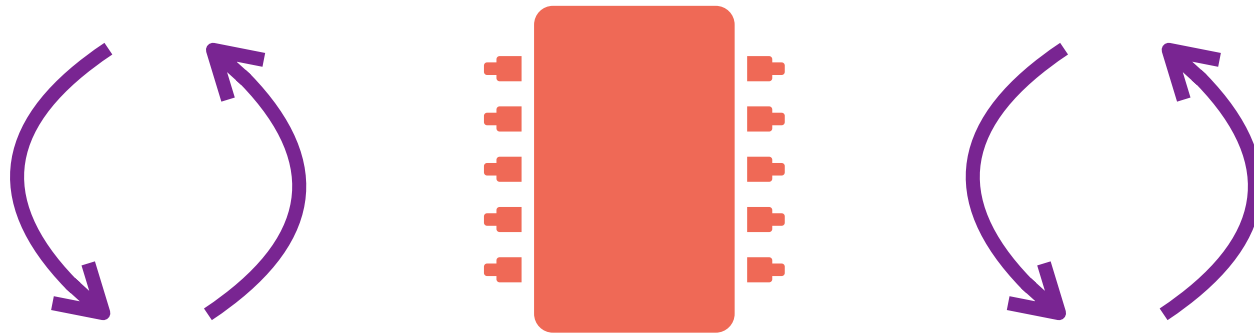


```
while (isRunning) {  
    val line = bufferedReader.readLine()  
    when (line) {  
        "ECHO" → bufferedWriter.write(line)  
        // ...  
        // Other cases ( ... )  
        // ...  
        else → bufferedWriter.write("Unknown command")  
    }  
}
```


"When you have a line of text, call C2"







2 event-loops per CPU core by default

Movie rating application

```
router {  
  get("/movie/:id") {  
    ctx → getMovie(ctx)  
  }  
  post("/rate/:id") {  
    ctx → rateMovie(ctx)  
  }  
  get("/rating/:id") {  
    ctx → getRating(ctx)  
  }  
}
```

Movie rating application

```
router {  
  get("/movie/:id") {  
    ctx → getMovie(ctx)  
  }  
  post("/rate/:id") {  
    ctx → rateMovie(ctx)  
  }  
  get("/rating/:id") {  
    ctx → getRating(ctx)  
  }  
}
```



```
fun getMovie(ctx: RoutingContext) {  
  
    val id = ctx.pathParam("id")  
    val params = json { array(id) }  
    client.queryWithParams("SELECT TITLE FROM MOVIE WHERE ID=?", params) {  
        if (it.succeeded()) {  
            val result = it.result()  
            if (result.rows.size == 1) {  
                ctx.response().end(json {  
                    obj("id" to id, "title" to result.rows[0]["TITLE"]).encode()  
                })  
            } else {  
                ctx.response().setStatusCode(404).end()  
            }  
        } else {  
            ctx.fail(it.cause())  
        }  
    }  
}
```

```
fun getMovie(ctx: RoutingContext) {  
  
    val id = ctx.pathParam("id")  
    val params = json { array(id) }  
    client.queryWithParams("SELECT TITLE FROM MOVIE WHERE ID=?", params) {  
        if (it.succeeded()) {  
            val result = it.result()  
            if (result.rows.size == 1) {  
                ctx.response().end(json {  
                    obj("id" to id, "title" to result.rows[0]["TITLE"]).encode()  
                })  
            } else {  
                ctx.response().setStatusCode(404).end()  
            }  
        } else {  
            ctx.fail(it.cause())  
        }  
    }  
}
```

```
fun getMovie(ctx: RoutingContext) {  
  
    val id = ctx.pathParam("id")  
    val params = json { array(id) }  
    client.queryWithParams("SELECT TITLE FROM MOVIE WHERE ID=?", params) {  
        if (it.succeeded()) {  
            val result = it.result()  
            if (result.rows.size == 1) {  
                ctx.response().end(json {  
                    obj("id" to id, "title" to result.rows[0]["TITLE"]).encode()  
                })  
            } else {  
                ctx.response().setStatusCode(404).end()  
            }  
        } else {  
            ctx.fail(it.cause())  
        }  
    }  
}
```

```
fun getMovie(ctx: RoutingContext) {  
  
    val id = ctx.pathParam("id")  
    val params = json { array(id) }  
    client.queryWithParams("SELECT TITLE FROM MOVIE WHERE ID=?", params) {  
        if (it.succeeded()) {  
            val result = it.result()  
            if (result.rows.size == 1) {  
                ctx.response().end(json {  
                    obj("id" to id, "title" to result.rows[0]["TITLE"]).encode()  
                })  
            } else {  
                ctx.response().setStatusCode(404).end()  
            }  
        } else {  
            ctx.fail(it.cause())  
        }  
    }  
}
```

Movie rating application

```
router {  
  get("/movie/:id") {  
    ctx → getMovie(ctx)  
  }  
  post("/rate/:id") {  
    ctx → rateMovie(ctx)  
  }  
  get("/rating/:id") {  
    ctx → getRating(ctx)  
  }  
}
```

```

val movie = ctx.pathParam("id")
val rating = Integer.parseInt(ctx.queryParam("getRating")[0])
client.getConnection {
    if (it.succeeded()) {
        val connection = it.result()
        val queryParams = json { array(movie) }
        connection.queryWithParams("SELECT TITLE FROM MOVIE WHERE ID=?", queryParams) {
            if (it.succeeded()) {
                val result = it.result()
                if (result.rows.size == 1) {
                    val updateParams = json { array(rating, movie) }
                    connection.updateWithParams("INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?", updateParams) {
                        if (it.succeeded()) {
                            ctx.response().setStatusCode(201).end()
                        } else {
                            connection.close()
                            ctx.fail(it.cause())
                        }
                    }
                }
            } else {
                connection.close()
                ctx.response().setStatusCode(404).end()
            }
        } else {
            connection.close()
            ctx.fail(it.cause())
        }
    }
} else {
    ctx.fail(it.cause())
}
}

```

```

val movie = ctx.pathParam("id")
val rating = Integer.parseInt(ctx.queryParam("getRating")[0])
client.getConnection {
    if (it.succeeded()) {
        val connection = it.result()
        val queryParams = json { array(movie) }
        connection.queryWithParams("SELECT TITLE FROM MOVIE WHERE ID=?", queryParams) {
            if (it.succeeded()) {
                val result = it.result()
                if (result.rows.size == 1) {
                    val updateParams = json { array(rating, movie) }
                    connection.updateWithParams("INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?", updateParams) {
                        if (it.succeeded()) {
                            ctx.response().setStatusCode(201).end()
                        } else {
                            connection.close()
                            ctx.fail(it.cause())
                        }
                    }
                }
            } else {
                connection.close()
                ctx.response().setStatusCode(404).end()
            }
        } else {
            connection.close()
            ctx.fail(it.cause())
        }
    }
} else {
    ctx.fail(it.cause())
}
}

```

```

val movie = ctx.pathParam("id")
val rating = Integer.parseInt(ctx.queryParam("getRating")[0])
client.getConnection {
    if (it.succeeded()) {
        val connection = it.result()
        val queryParams = json { array(movie) }
        connection.queryWithParams("SELECT TITLE FROM MOVIE WHERE ID=?", queryParams) {
            if (it.succeeded()) {
                val result = it.result()
                if (result.rows.size == 1) {
                    val updateParams = json { array(rating, movie) }
                    connection.updateWithParams("INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?", updateParams) {
                        if (it.succeeded()) {
                            ctx.response().setStatusCode(201).end()
                        } else {
                            connection.close()
                            ctx.fail(it.cause())
                        }
                    }
                }
            } else {
                connection.close()
                ctx.response().setStatusCode(404).end()
            }
        } else {
            connection.close()
            ctx.fail(it.cause())
        }
    }
} else {
    ctx.fail(it.cause())
}
}

```



```

val movie = ctx.pathParam("id")
val rating = Integer.parseInt(ctx.queryParam("getRating")[0])
client.getConnection {
    if (it.succeeded()) {
        val connection = it.result()
        val queryParams = json { array(movie) }
        connection.queryWithParams("SELECT TITLE FROM MOVIE WHERE ID=?", queryParams) {
            if (it.succeeded()) {
                val result = it.result()
                if (result.rows.size == 1) {
                    val updateParams = json { array(rating, movie) }
                    connection.updateWithParams("INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?", updateParams) {
                        if (it.succeeded()) {
                            ctx.response().setStatusCode(201).end()
                        } else {
                            connection.close()
                            ctx.fail(it.cause())
                        }
                    }
                }
            } else {
                connection.close()
                ctx.response().setStatusCode(404).end()
            }
        } else {
            connection.close()
            ctx.fail(it.cause())
        }
    }
} else {
    ctx.fail(it.cause())
}
}

```

```

val movie = ctx.pathParam("id")
val rating = Integer.parseInt(ctx.queryParam("getRating")[0])
client.getConnection {
    if (it.succeeded()) {
        val connection = it.result()
        val queryParams = json { array(movie) }
        connection.queryWithParams("SELECT TITLE FROM MOVIE WHERE ID=?", queryParams) {
            if (it.succeeded()) {
                val result = it.result()
                if (result.rows.size == 1) {
                    val updateParams = json { array(rating, movie) }
                    connection.updateWithParams("INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?", updateParams) {
                        if (it.succeeded()) {
                            ctx.response().setStatusCode(201).end()
                        } else {
                            connection.close()
                            ctx.fail(it.cause())
                        }
                    }
                } else {
                    connection.close()
                    ctx.response().setStatusCode(404).end()
                }
            } else {
                connection.close()
                ctx.fail(it.cause())
            }
        }
    }
} else {
    ctx.fail(it.cause())
}
}

```

```

val movie = ctx.pathParam("id")
val rating = Integer.parseInt(ctx.queryParam("getRating")[0])
client.getConnection {
    if (it.succeeded()) {
        val connection = it.result()
        val queryParams = json { array(movie) }
        connection.queryWithParams("SELECT TITLE FROM MOVIE WHERE ID=?", queryParams) {
            if (it.succeeded()) {
                val result = it.result()
                if (result.rows.size == 1) {
                    val updateParams = json { array(rating, movie) }
                    connection.updateWithParams("INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?", updateParams) {
                        if (it.succeeded()) {
                            ctx.response().setStatusCode(201).end()
                        } else {
                            connection.close()
                            ctx.fail(it.cause())
                        }
                    }
                }
            } else {
                connection.close()
                ctx.response().setStatusCode(404).end()
            }
        }
    } else {
        connection.close()
        ctx.fail(it.cause())
    }
}
} else {
    ctx.fail(it.cause())
}
}

```

```

val movie = ctx.pathParam("id")
val rating = Integer.parseInt(ctx.queryParam("getRating")[0])
client.getConnection {
    if (it.succeeded()) {
        val connection = it.result()
        val queryParams = json { array(movie) }
        connection.queryWithParams("SELECT TITLE FROM MOVIE WHERE ID=?", queryParams) {
            if (it.succeeded()) {
                val result = it.result()
                if (result.rows.size == 1) {
                    val updateParams = json { array(rating, movie) }
                    connection.updateWithParams("INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?", updateParams) {
                        if (it.succeeded()) {
                            ctx.response().setStatusCode(201).end()
                        } else {
                            connection.close()
                            ctx.fail(it.cause())
                        }
                    }
                }
            } else {
                connection.close()
                ctx.response().setStatusCode(404).end()
            }
        } else {
            connection.close()
            ctx.fail(it.cause())
        }
    }
} else {
    ctx.fail(it.cause())
}
}

```

```

val movie = ctx.pathParam("id")
val rating = Integer.parseInt(ctx.queryParam("getRating")[0])
client.getConnection {
    if (it.succeeded()) {
        val connection = it.result()
        val queryParams = json { array(movie) }
        connection.queryWithParams("SELECT TITLE FROM MOVIE WHERE ID=?", queryParams) {
            if (it.succeeded()) {
                val result = it.result()
                if (result.rows.size == 1) {
                    val updateParams = json { array(rating, movie) }
                    connection.updateWithParams("INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?", updateParams) {
                        if (it.succeeded()) {
                            ctx.response().setStatusCode(201).end()
                        } else {
                            connection.close()
                            ctx.fail(it.cause())
                        }
                    }
                }
            } else {
                connection.close()
                ctx.response().setStatusCode(404).end()
            }
        } else {
            connection.close()
            ctx.fail(it.cause())
        }
    }
} else {
    ctx.fail(it.cause())
}
}

```

```

val movie = ctx.pathParam("id")
val rating = Integer.parseInt(ctx.queryParam("getRating")[0])
client.getConnection {
    if (it.succeeded()) {
        val connection = it.result()
        val queryParams = json { array(movie) }
        connection.queryWithParams("SELECT TITLE FROM MOVIE WHERE ID=?", queryParams) {
            if (it.succeeded()) {
                val result = it.result()
                if (result.rows.size == 1) {
                    val updateParams = json { array(rating, movie) }
                    connection.updateWithParams("INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?", updateParams) {
                        if (it.succeeded()) {
                            ctx.response().setStatusCode(201).end()
                        } else {
                            connection.close()
                            ctx.fail(it.cause())
                        }
                    }
                } else {
                    connection.close()
                    ctx.response().setStatusCode(404).end()
                }
            } else {
                connection.close()
                ctx.fail(it.cause())
            }
        }
    } else {
        ctx.fail(it.cause())
    }
}

```



```

class RateMovie(val ctx: class RateMovie(
    val ctx: RoutingContext,
    val client: SQLClient,
    val movie: String,
    val rating: Int) {

    fun rate() {
        client.getConnection {
            if (it.succeeded()) {
                query(it.result())
            } else {
                ctx.fail(it.cause())
            }
        }
    }
}

...
}

```

Divide and conquer strategy

```
fun query(connection: SqlConnection) {  
    val params = json { array(movie) }  
    connection.queryWithParams("SELECT TITLE FROM MOVIE WHERE ID=?", params) {  
        if (it.succeeded()) {  
            val result = it.result()  
            if (result.rows.size == 1) {  
                update(connection)  
            } else {  
                connection.close()  
                ctx.response().setStatusCode(404).end()  
            }  
        } else {  
            connection.close()  
            ctx.fail(it.cause())  
        }  
    }  
}
```



```
fun update(connection: SQLConnection) {  
    val params = json { array(rating, movie) }  
    connection.updateWithParams("INSERT INTO RATING (VALUE, MOVIE_ID)  
VALUES ?, ?", params) {  
        connection.close()  
        if (it.succeeded()) {  
            ctx.response().setStatusCode(201).end()  
        } else {  
            ctx.fail(it.cause())  
        }  
    }  
}
```

Going interactive with coroutines

Kotlin Coroutines

Toolkit approach

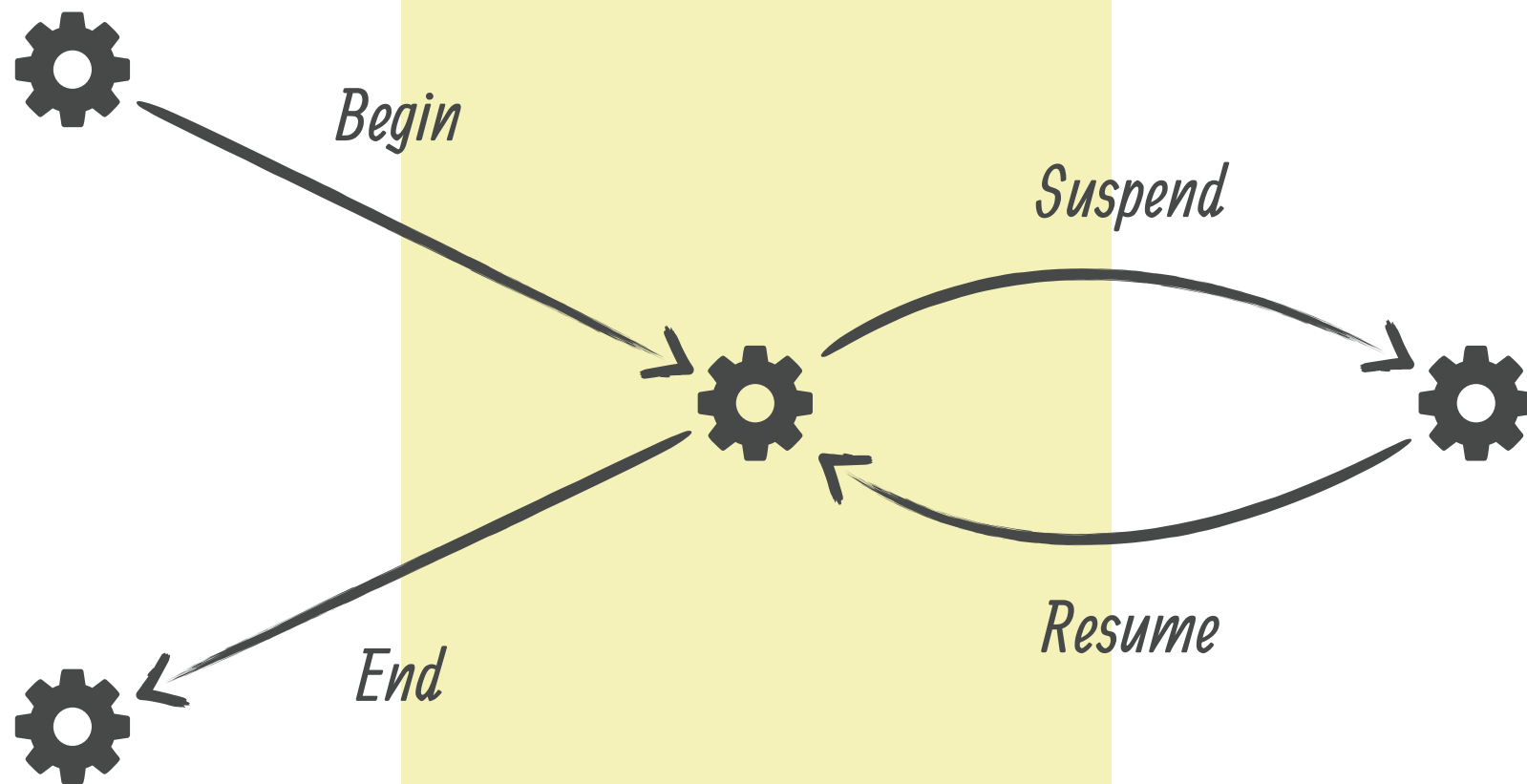
Suspending lambdas and functions

Sequential flow

Coroutines can be composed

Language control flow

Coroutine life cycle



Coroutines for Vert.x

Coroutines are confined on Vert.x event loop thread

`awaitResult<T>` for asynchronous methods

channel support

integrates with coroutine ecosystem

```
launch(vertx.dispatcher()) {  
  
    try {  
  
        val result1 = awaitResult<String> { handler →  
            handler.handle(Future.succeededFuture("OK"))  
        }  
        println("Result 1 $result1")  
  
        val result2 = awaitResult<String> { handler →  
            handler.handle(Future.failedFuture("Ouch"))  
        }  
        println("Result 2 $result1")  
  
    } catch (e: Exception) {  
        println("Ouch ${e.message}")  
    }  
}
```

```
launch(vertx.dispatcher()) {
```

```
  try {
```

```
    val result1 = awaitResult<String> { handler →  
      handler.handle(Future.succeededFuture("OK"))  
    }
```

```
    println("Result 1 $result1")
```

```
    val result2 = awaitResult<String> { handler →  
      handler.handle(Future.failedFuture("Ouch"))  
    }
```

```
    println("Result 2 $result1")
```

```
  } catch (e: Exception) {  
    println("Ouch ${e.message}")  
  }
```

```
}
```

```
launch(vertx.dispatcher()) {  
  
    try {  
  
        val result1 = awaitResult<String> { handler →  
            handler.handle(Future.succeededFuture("OK"))  
        }  
        println("Result 1 $result1")  
  
        val result2 = awaitResult<String> { handler →  
            handler.handle(Future.failedFuture("Ouch"))  
        }  
        println("Result 2 $result1")  
  
    } catch (e: Exception) {  
        println("Ouch ${e.message}")  
    }  
}
```



```
launch(vertx.dispatcher()) {  
  
    try {  
  
        val result1 = awaitResult<String> { handler →  
            handler.handle(Future.succeededFuture("OK"))  
        }  
        println("Result 1 $result1")  
  
        val result2 = awaitResult<String> { handler →  
            handler.handle(Future.failedFuture("Ouch"))  
        }  
        println("Result 2 $result1")  
  
    } catch (e: Exception) {  
        println("Ouch ${e.message}")  
    }  
}
```

```
launch(vertx.dispatcher()) {  
  
    try {  
  
        val result1 = awaitResult<String> { handler →  
            handler.handle(Future.succeededFuture("OK"))  
        }  
        println("Result 1 $result1")  
  
        val result2 = awaitResult<String> { handler →  
            handler.handle(Future.failedFuture("Ouch"))  
        }  
        println("Result 2 $result1")  
  
    } catch (e: Exception) {  
        println("Ouch ${e.message}")  
    }  
}
```

```
launch(vertx.dispatcher()) {  
  
    try {  
  
        val result1 = awaitResult<String> { handler →  
            handler.handle(Future.succeededFuture("OK"))  
        }  
        println("Result 1 $result1")  
  
        val result2 = awaitResult<String> { handler →  
            handler.handle(Future.failedFuture("Ouch"))  
        }  
        println("Result 2 $result1")  
  
    } catch (e: Exception) {  
        println("Ouch ${e.message}")  
    }  
}
```

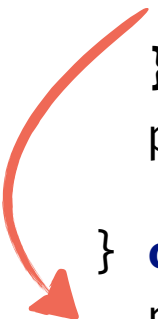


```
launch(vertx.dispatcher()) {  
  
    try {  
  
        val result1 = awaitResult<String> { handler →  
            handler.handle(Future.succeededFuture("OK"))  
        }  
        println("Result 1 $result1")  
  
        val result2 = awaitResult<String> { handler →  
            handler.handle(Future.failedFuture("Ouch"))  
        }  
        println("Result 2 $result1")  
  
    } catch (e: Exception) {  
        println("Ouch ${e.message}")  
    }  
}
```

```
launch(vertx.dispatcher()) {  
  
    try {  
  
        val result1 = awaitResult<String> { handler →  
            handler.handle(Future.succeededFuture("OK"))  
        }  
        println("Result 1 $result1")  
  
        val result2 = awaitResult<String> { handler →  
            handler.handle(Future.failedFuture("Ouch"))  
        }  
        println("Result 2 $result1")  
  
    } catch (e: Exception) {  
        println("Ouch ${e.message}")  
    }  
}
```

```
launch(vertx.dispatcher()) {  
  
    try {  
  
        val result1 = awaitResult<String> { handler →  
            handler.handle(Future.succeededFuture("OK"))  
        }  
        println("Result 1 $result1")  
  
        val result2 = awaitResult<String> { handler →  
            handler.handle(Future.failedFuture("Ouch"))  
        }  
        println("Result 2 $result1")  
  
    } catch (e: Exception) {  
        println("Ouch ${e.message}")  
    }  
}
```

```
launch(vertx.dispatcher()) {  
  
    try {  
  
        val result1 = awaitResult<String> { handler →  
            handler.handle(Future.succeededFuture("OK"))  
        }  
        println("Result 1 $result1")  
  
        val result2 = awaitResult<String> { handler →  
            handler.handle(Future.failedFuture("Ouch"))  
        }  
        println("Result 2 $result1")  
  
    } catch (e: Exception) {  
        println("Ouch ${e.message}")  
    }  
}
```



```
launch(vertx.dispatcher()) {  
  
    try {  
  
        val result1 = awaitResult<String> { handler →  
            handler.handle(Future.succeededFuture("OK"))  
        }  
        println("Result 1 $result1")  
  
        val result2 = awaitResult<String> { handler →  
            handler.handle(Future.failedFuture("Ouch"))  
        }  
        println("Result 2 $result1")  
  
    } catch (e: Exception) {  
        println("Ouch ${e.message}")  
    }  
}
```


(demo)

"Going interactive with coroutines"

Streaming with channels

Streaming with channels

Kotlin provides channels

ReceiveChannel

SendChannel

Vert.x provides streams

ReadStream

WriteStream

send(●)



receive()



```
vertx.createHttpServer().requestHandler { request →
```

```
    val readStream: ReadStream<Buffer> = request
```

```
    readStream.handler { buffer →
```

```
        // Handle each buffer
```

```
    }
```

```
    readStream.exceptionHandler { err →
```

```
        request.response().setStatusCode(500).end("${err.message}")
```

```
    }
```

```
    readStream.endHandler {
```

```
        request.response().end("OK")
```

```
    }
```

```
}.listen(8080)
```

ReceiveChannel

vertx.createHttpServer().requestHandler { request →

```
val readStream: ReadStream<Buffer> = request
```

```
val receiveChannel: ReceiveChannel<Buffer> = readStream.toChannel(vertx)
```

```
launch(vertx.dispatcher()) {
```

```
  try {
```

```
    for (buffer in receiveChannel) {
```

```
      // Handle each buffer
```

```
    }
```

```
    request.response().end("OK")
```

```
  } catch (e: Exception) {
```

```
    request.response().setStatusCode(500).end("${e.message}")
```

```
  }
```

```
}
```

```
}.listen(8080)
```



```
val writeStream: WriteStream<Buffer> = request.response()
```

```
val item = Buffer.buffer("the-item")
```

```
fun sendItemAndClose() {  
    writeStream.write(item)  
    request.response().end()  
}
```

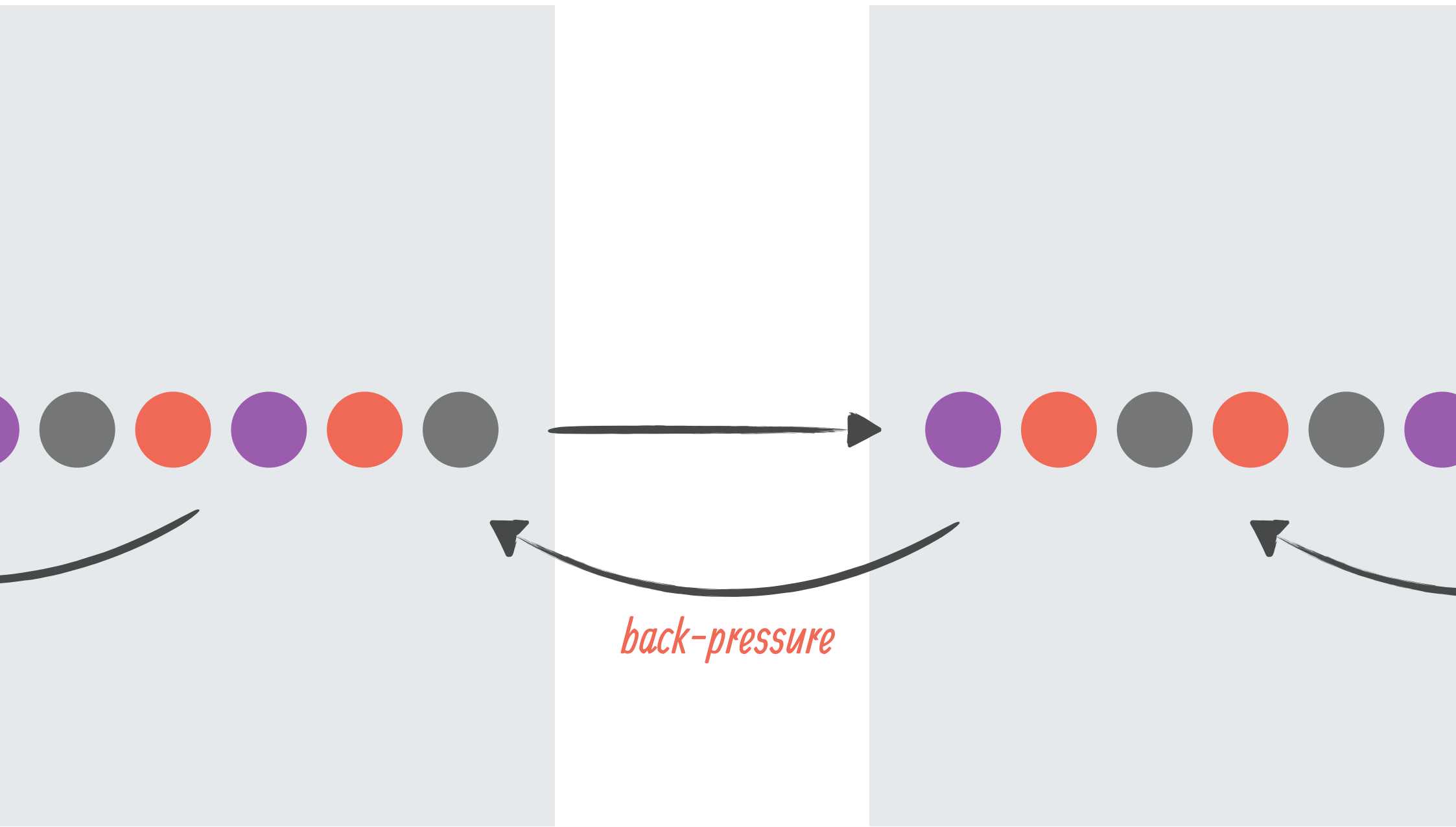
```
if (!writeStream.writeQueueFull()) {  
    sendItemAndClose()  
} else {  
    writeStream.drainHandler {  
        sendItemAndClose()  
    }  
}
```

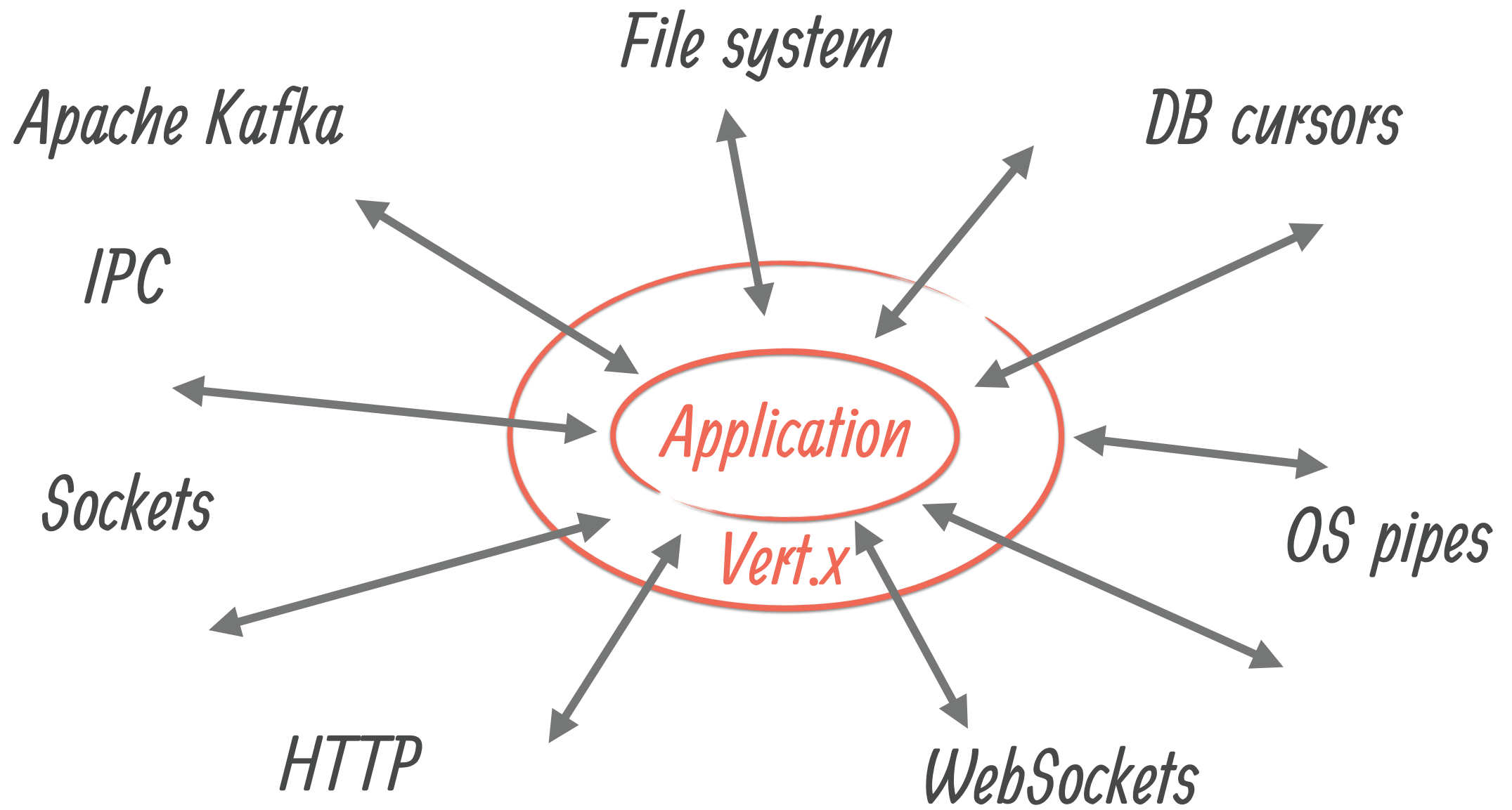


SendChannel

```
val writeStream: WriteStream<Buffer> = request.response()  
val sendChannel = writeStream.toChannel(vertex)  
  
launch(vertex.dispatcher()) {  
    sendChannel.send(Buffer.buffer("the-item"))  
    request.response().end()  
}
```

⏸ *when full*
▶ *when drained*





Preemptive back-pressure

```
try {  
    while (true) {  
        val amount = input.read(buffer)  
        if (amount == -1) {  
            break  
        }  
        output.write(buffer, 0, amount)  
    }  
} finally {  
    output.close()  
}
```

*when buffer is full
block the thread*

Cooperative back-pressure

```
launch(vertx.dispatcher()) {  
    try {  
        while (true) {  
            val buffer = input.receiveOrNull()  
            if (buffer == null) {  
                break;  
            }  
            output.send(buffer);  
        }  
    } finally {  
        output.close()  
    }  
}
```

*when buffer is full
suspends the coroutine*

Example - JSON parser

Most parsers requires full buffering

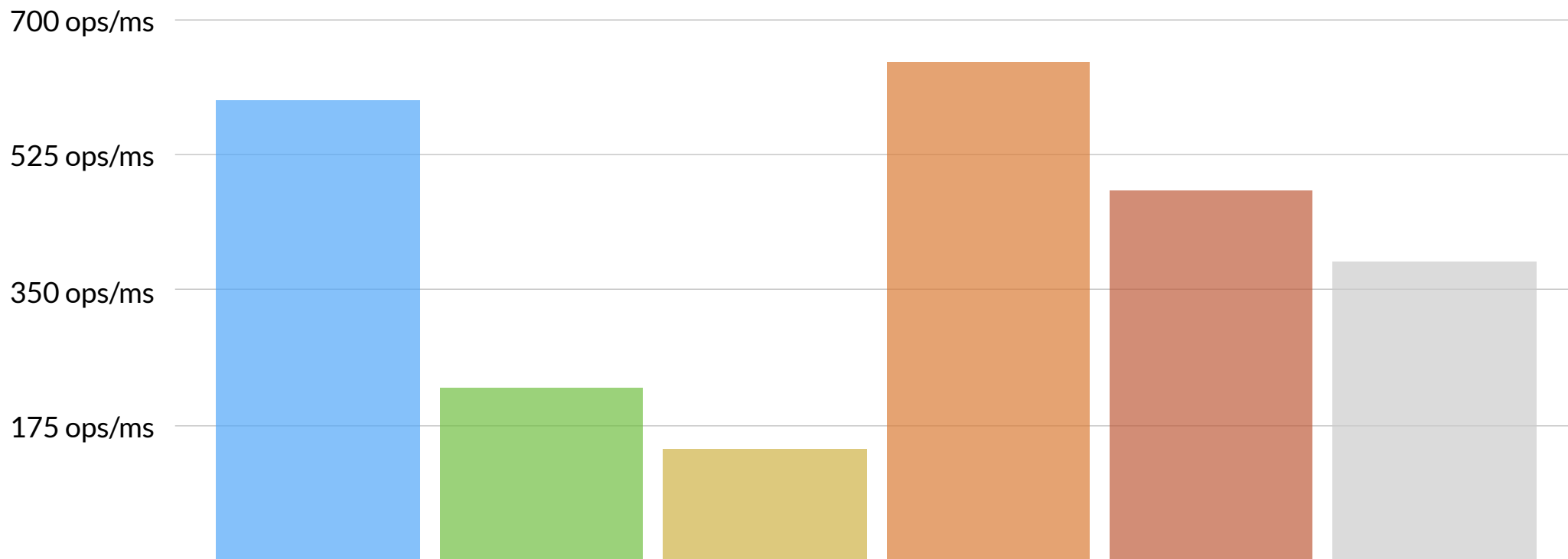
- Process JSON as soon as possible

- Reduce the footprint

Handle large JSON documents

JSON streaming

Synchronous Coroutine 1 Coroutine (10 chunks)
Vert.x parser Vert.x reactive parser Vert.x reactive parser (10 chunks)



Coroutines vs RxJava

Rxified application

```
router {  
  get("/movie/:id") {  
    ctx → getMovie(ctx)  
  }  
  post("/rate/:id") {  
    ctx → rateMovie(ctx)  
  }  
  get("/rating/:id") {  
    ctx → getRating(ctx)  
  }  
}
```



```
val movie = ctx.pathParam("id")
val rating = ctx.queryParam("getRating")[0]
val query = "SELECT TITLE FROM MOVIE WHERE ID=?"
val queryParams = json { array(movie) }
val update = "INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?"
val updateParams = json { array(rating, movie) }

val single = client.rxGetConnection().flatMap {
    connection →
    connection
        .rxQueryWithParams(query, queryParams)
        .flatMap {
            result →
            if (result.results.size == 1) {
                connection.rxUpdateWithParams(update, updateParams)
            } else {
                Single.error<UpdateResult>(NotFoundException())
            }
        }
        .doAfterTerminate { connection.close() }
}
```

```
val movie = ctx.pathParam("id")
val rating = ctx.queryParam("getRating")[0]
val query = "SELECT TITLE FROM MOVIE WHERE ID=?"
val queryParams = json { array(movie) }
val update = "INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?"
val updateParams = json { array(rating, movie) }

val single = client.rxGetConnection().flatMap {
    connection →
    connection
        .rxQueryWithParams(query, queryParams)
        .flatMap {
            result →
            if (result.results.size == 1) {
                connection.rxUpdateWithParams(update, updateParams)
            } else {
                Single.error<UpdateResult>(NotFoundException())
            }
        }
        .doAfterTerminate { connection.close() }
}
```

```
val movie = ctx.pathParam("id")
val rating = ctx.queryParam("getRating")[0]
val query = "SELECT TITLE FROM MOVIE WHERE ID=?"
val queryParams = json { array(movie) }
val update = "INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?"
val updateParams = json { array(rating, movie) }

val single = client.rxGetConnection().flatMap {
    connection →
    connection
        .rxQueryWithParams(query, queryParams)
        .flatMap {
            result →
            if (result.results.size == 1) {
                connection.rxUpdateWithParams(update, updateParams)
            } else {
                Single.error<UpdateResult>(NotFoundException())
            }
        }
        .doAfterTerminate { connection.close() }
}
```

```
val movie = ctx.pathParam("id")
val rating = ctx.queryParam("getRating")[0]
val query = "SELECT TITLE FROM MOVIE WHERE ID=?"
val queryParams = json { array(movie) }
val update = "INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?"
val updateParams = json { array(rating, movie) }

val single = client.rxGetConnection().flatMap {
    connection →
    connection
        .rxQueryWithParams(query, queryParams)
        .flatMap {
            result →
            if (result.results.size == 1) {
                connection.rxUpdateWithParams(update, updateParams)
            } else {
                Single.error<UpdateResult>(NotFoundException())
            }
        }
        .doAfterTerminate { connection.close() }
}
```

```
val movie = ctx.pathParam("id")
val rating = ctx.queryParam("getRating")[0]
val query = "SELECT TITLE FROM MOVIE WHERE ID=?"
val queryParams = json { array(movie) }
val update = "INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?"
val updateParams = json { array(rating, movie) }

val single = client.rxGetConnection().flatMap {
    connection →
    connection
        .rxQueryWithParams(query, queryParams)
        .flatMap {
            result →
            if (result.results.size == 1) {
                connection.rxUpdateWithParams(update, updateParams)
            } else {
                Single.error<UpdateResult>(NotFoundException())
            }
        }
        .doAfterTerminate { connection.close() }
}
```

```
val movie = ctx.pathParam("id")
val rating = ctx.queryParam("getRating")[0]
val query = "SELECT TITLE FROM MOVIE WHERE ID=?"
val queryParams = json { array(movie) }
val update = "INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?"
val updateParams = json { array(rating, movie) }

val single = client.rxGetConnection().flatMap {
    connection →
    connection
        .rxQueryWithParams(query, queryParams)
        .flatMap {
            result →
            if (result.results.size == 1) {
                connection.rxUpdateWithParams(update, updateParams)
            } else {
                Single.error<UpdateResult>(NotFoundException())
            }
        }
        .doAfterTerminate { connection.close() }
}
```

```
val movie = ctx.pathParam("id")
val rating = ctx.queryParam("getRating")[0]
val query = "SELECT TITLE FROM MOVIE WHERE ID=?"
val queryParams = json { array(movie) }
val update = "INSERT INTO RATING (VALUE, MOVIE_ID) VALUES ?, ?"
val updateParams = json { array(rating, movie) }

val single = client.rxGetConnection().flatMap {
    connection →
    connection
        .rxQueryWithParams(query, queryParams)
        .flatMap {
            result →
            if (result.results.size == 1) {
                connection.rxUpdateWithParams(update, updateParams)
            } else {
                Single.error<UpdateResult>(NotFoundException())
            }
        }
    .doAfterTerminate { connection.close() }
}
```

```
val consumer = createKafkaConsumer(vertx, map, String::class, JsonObject::class)
val stream = consumer.toObservable()

stream
    .map({ record → record.value().getInteger("temperature") })
    .buffer(1, TimeUnit.SECONDS)
    .map({ list → list.sum() })
    .subscribe({ temperature → println("Current temperature is " + temperature) })
```


Coroutines and RxJava

Both are complementary

Combine them with

`kotlinx-coroutines-rx1`

`kotlinx-coroutines-rx2`

TL;DR

TL;DR

- ✓ Coroutines are great for workflows and correlated events
- ✓ Vert.x provides an unified end-to-end reactive model + ecosystem
- ✓ Make your Reactive applications Interactive in Kotlin

 *vertx.io*

 *Kotlin Slack #vertx*

 *Guide to async programming with Vert.x*

<https://goo.gl/AcWW3A>

 *Building Reactive Microservices in Java*

<https://goo.gl/ep6yB9>