R2DBC

Поиграем в игру?

- ODBC
  - Open Database Connectivity
- JDBC
  - Java Database Connectivity
- R2DBC
  - Reactive Relation Database Connectivity

Поиграем в игру?

- ODBC
  - 26
- JDBC
  - 21
- R2DBC
  - ~1

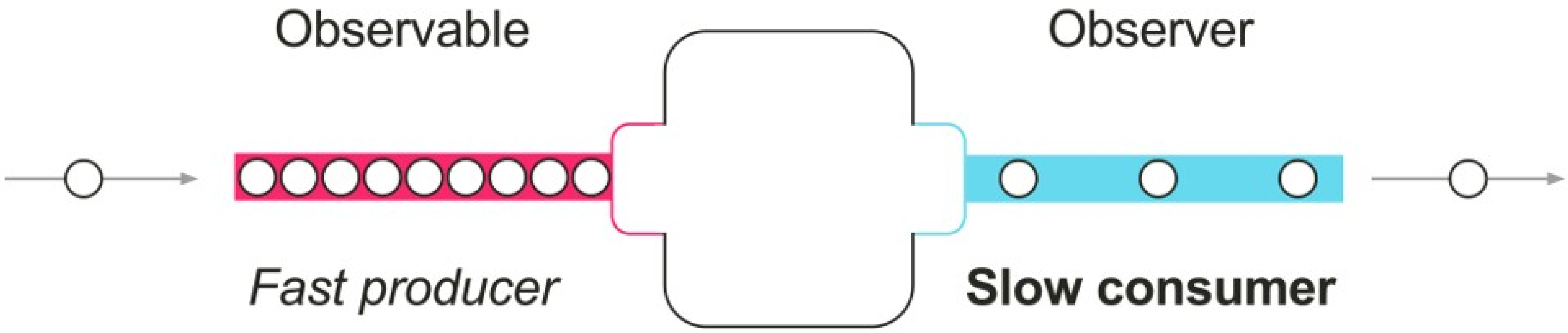
R2DBC

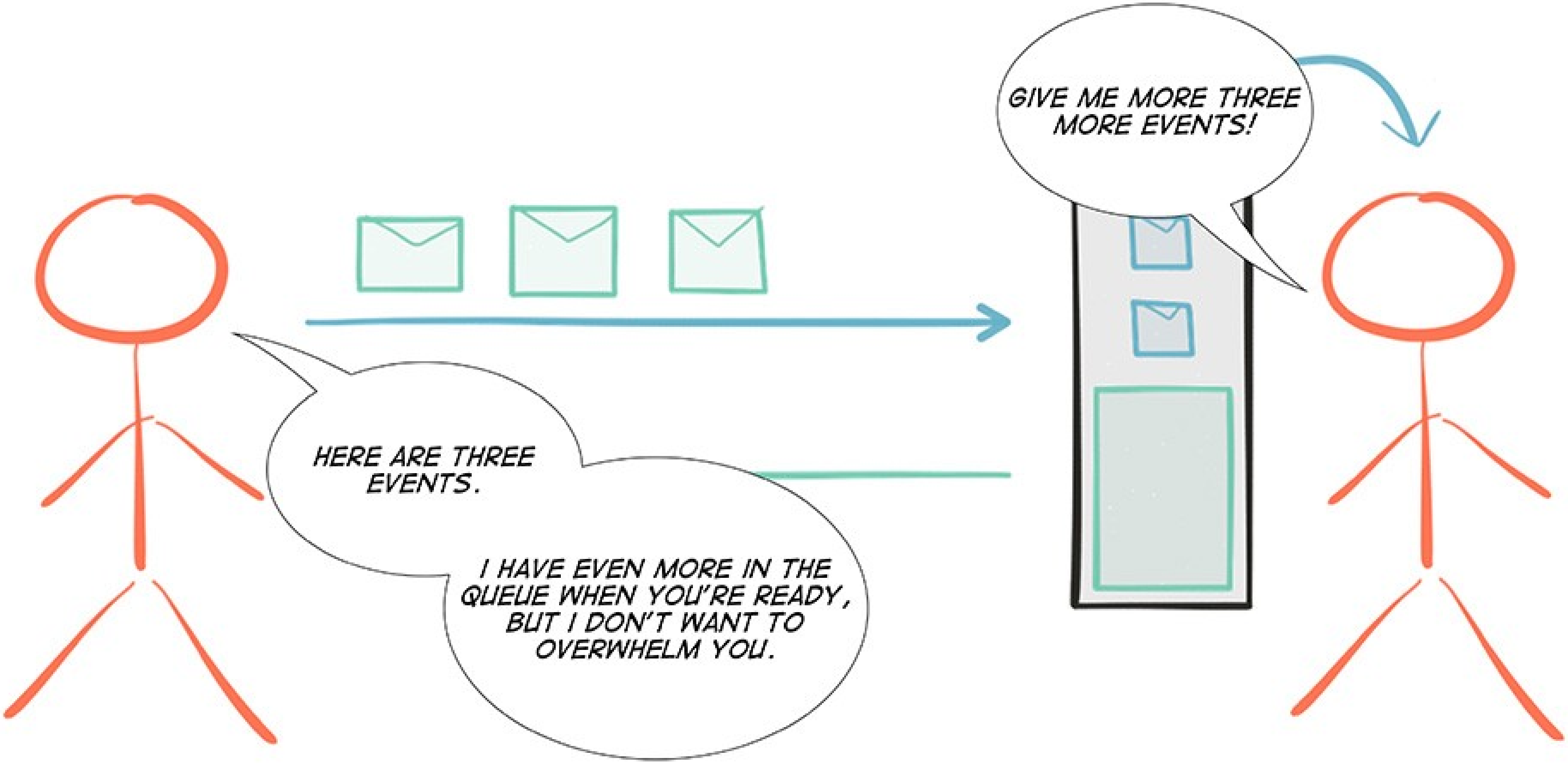
- **Reactive Streams** - R2DBC is founded on Reactive Streams providing an asynchronous, non-blocking API
- **Relational Databases** - R2DBC engages relational databases with a reactive API, something not possible with the blocking nature of JDBC and JPA

Why Reactive

Reactive API vs Coroutines API

	Reactive	Coroutines
Single Value	fun single(): Mono<T>	suspend fun single(): T
Cold Stream	fun coldStream(): Flux<T>	-
Hot Stream	fun hotStream(): Flux<T>	fun hotStream(): ReceiveChannel<T>


Observable
Observer
Fast producer
Slow consumer


GIVE ME MORE THREE MORE EVENTS!
HERE ARE THREE EVENTS.
I HAVE EVEN MORE IN THE QUEUE WHEN YOU'RE READY, BUT I DON'T WANT TO OVERWHELM YOU.

Why not ADBA?

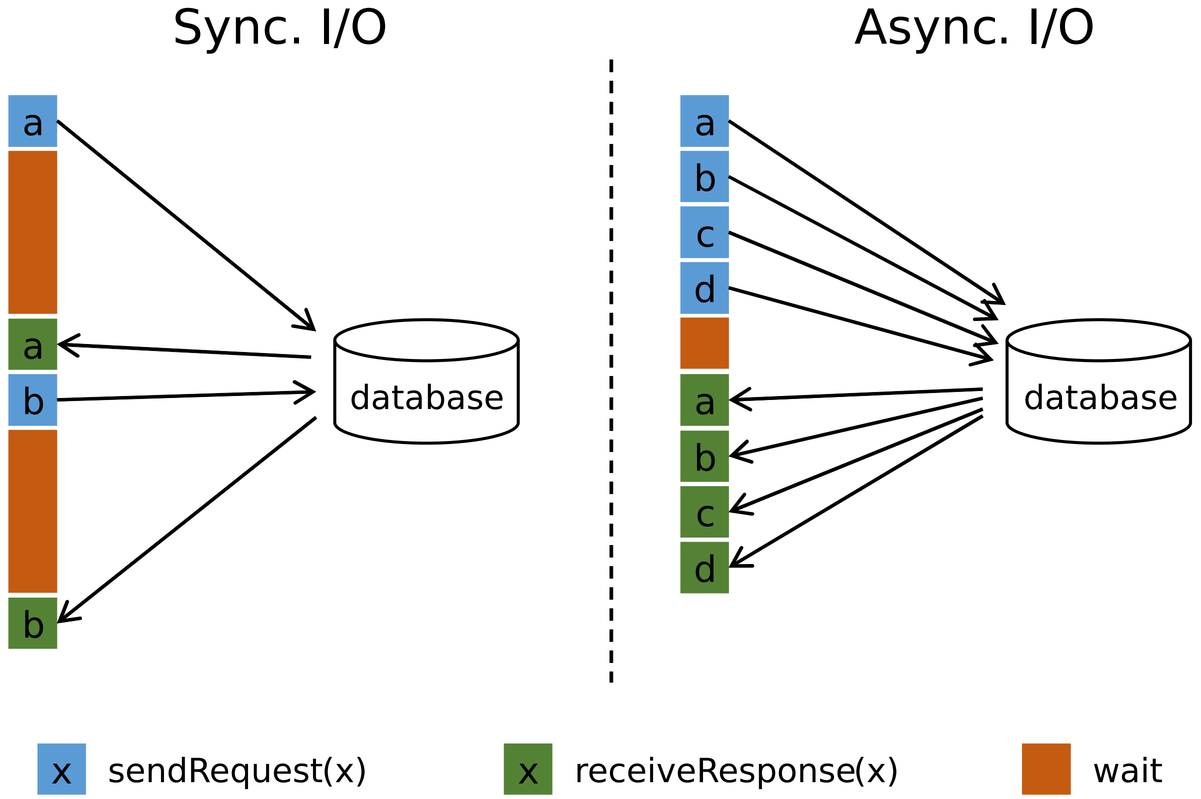
- Not Reactive
- Java 12 – no way (Java 17 maybe, 2021)

```
var result: CompletableFuture<List<Map<String, Any>>>


ds.getSession().use({ session ->

    val submit = session
        .rowOperation("SELECT id, name, manual FROM legoset")
        .collect(collectToMap())
        .submit()

    result = submit.getCompletionStage().toCompletableFuture()
})
```

Relational

Non-Blocking

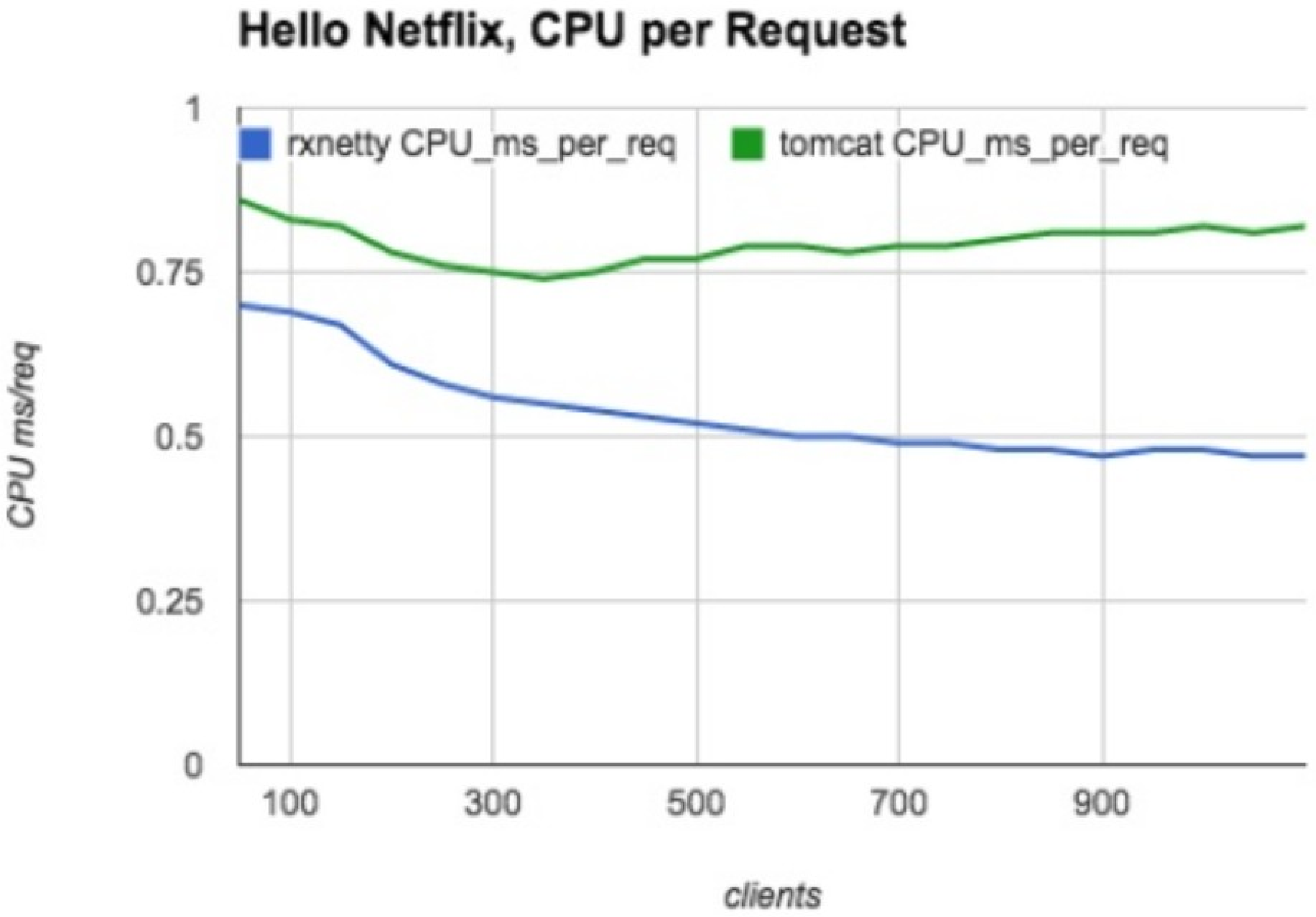
- CPU/RAM consumption per request
- Latency under load



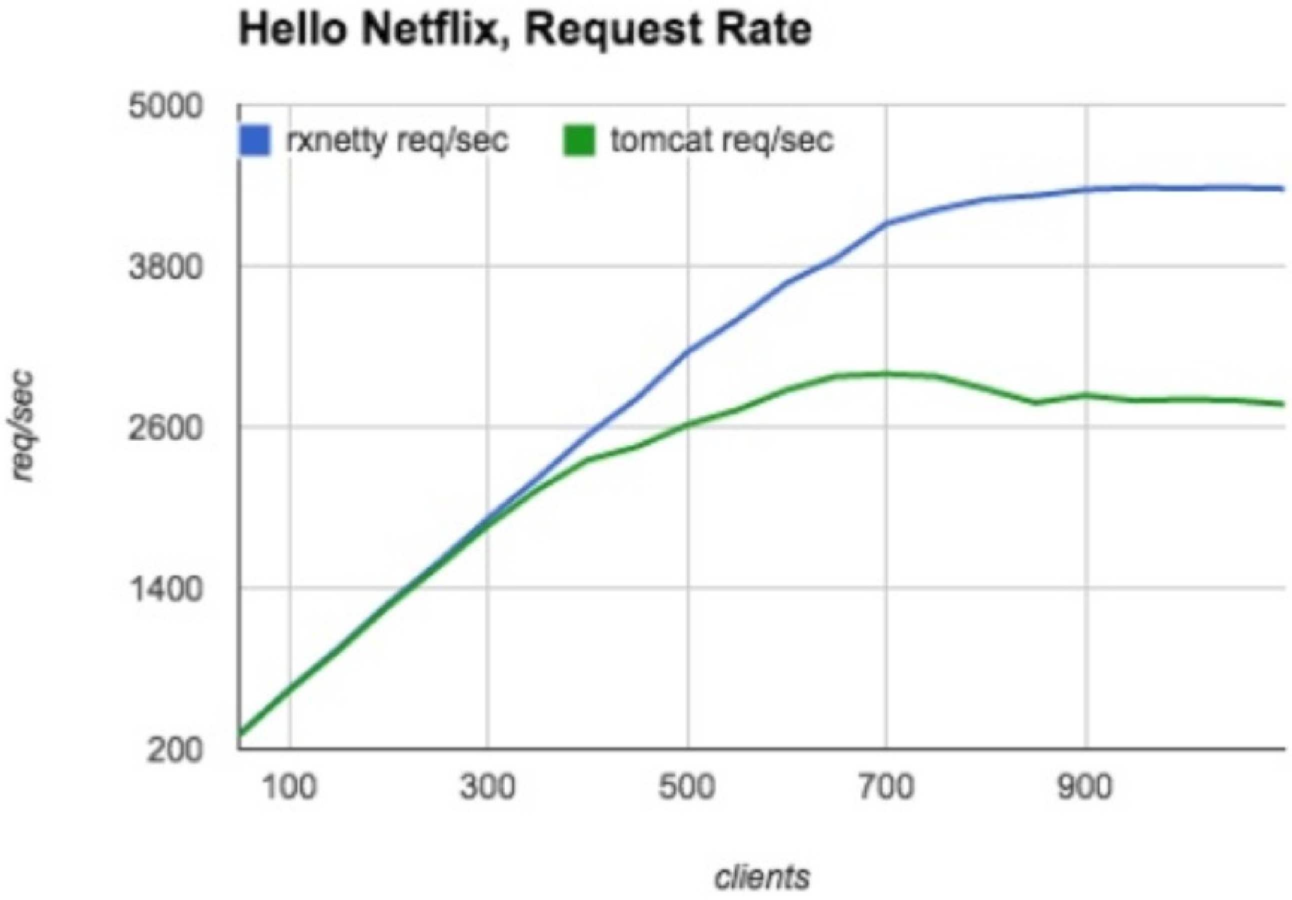
RxNetty vs Tomcat

https://www.slideshare.net/brendangregg/rxnetty-vs-tomcat-performance-results

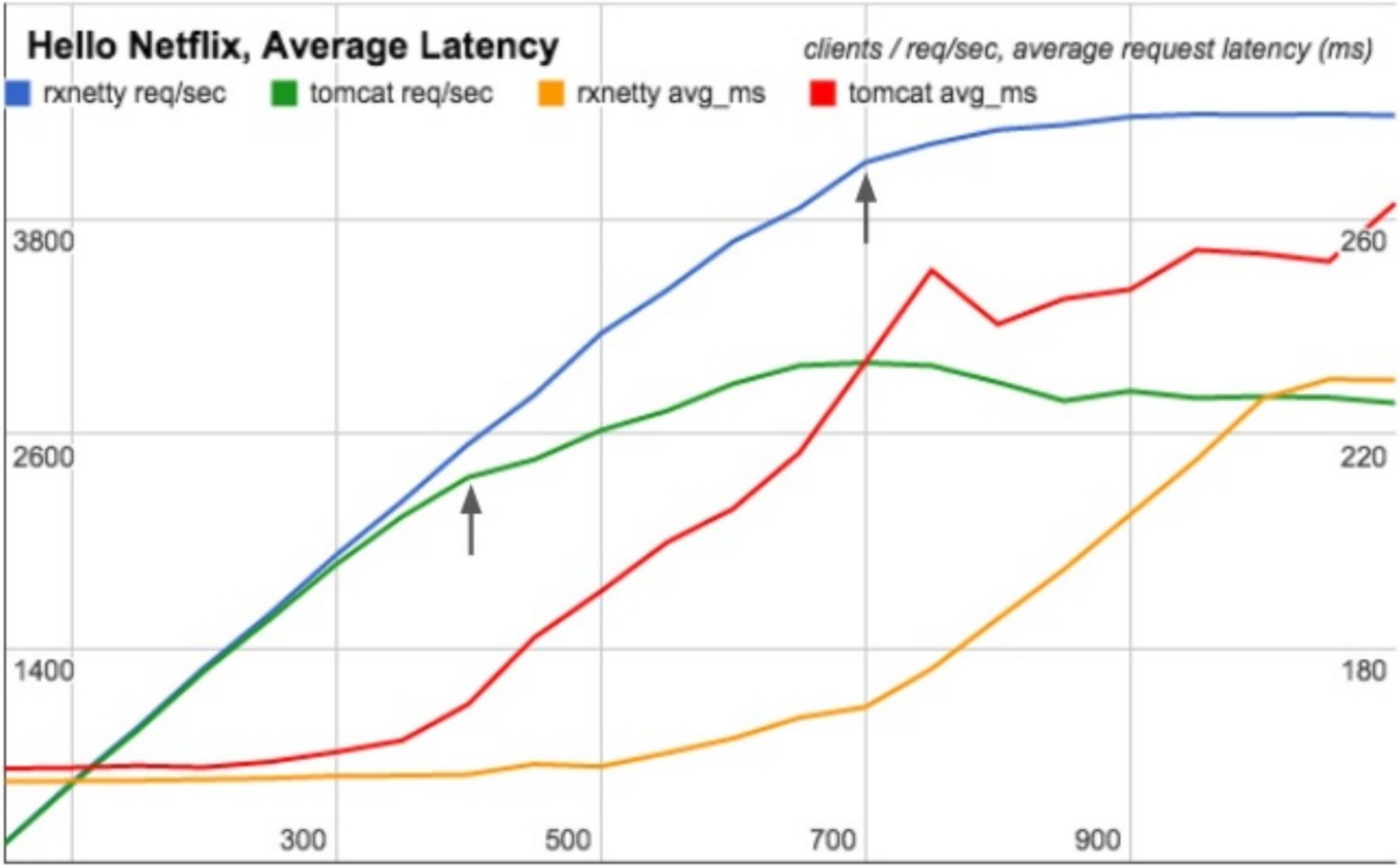
CPU/Request


Hello Netflix, CPU per Request
rxnetty CPU_ms_per_req
tomcat CPU_ms_per_req
1
0.75
0.5
0.25
0
CPU ms/req
100
300
500
700
900
clients

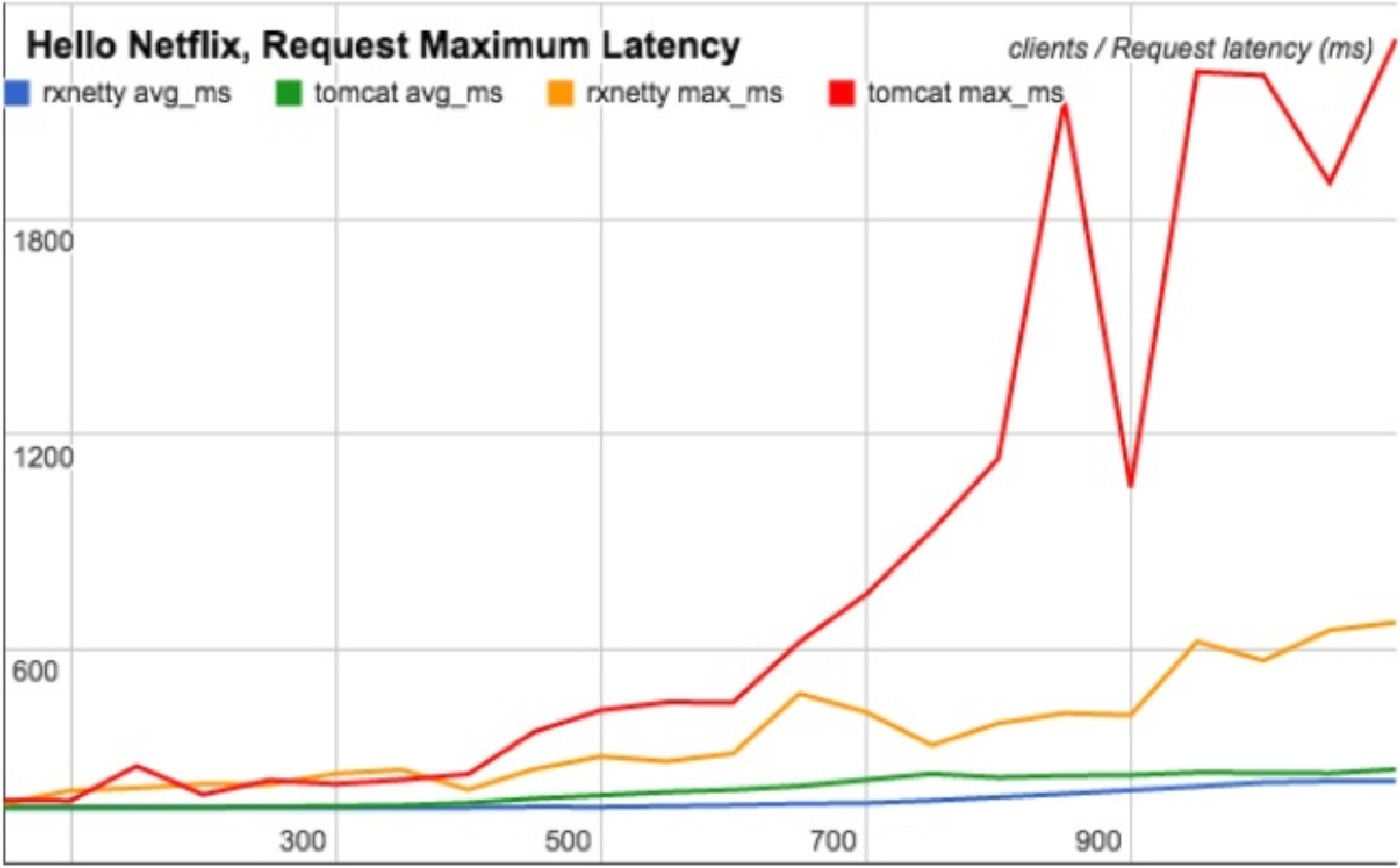
Request rate



Reqest Latency (avg)



Reqest Latency (max)



R2DBC

- r2dbc-spi
- r2dbc-client
- r2dbc-postgresql
- r2dbc-over-adba

Supported DBs

- PostgreSQL
- H2
- Microsoft SQL Server

Connection

```
val connectionFactory = PostgresqlConnectionFactory(
    PostgresqlConnectionConfiguration.builder()
        .host("localhost")
        .database("bkug")
        .username("bkug")
        .password("password").build()
)
```

Select SPI

```
connectionFactory.create()
    .flatMapMany { conn ->
        conn.createStatement("SELECT value FROM bkug")
            .execute()
            .flatMap { result ->
                result.map { row, meta -> row.get("value") }
            }
    }
```

Insert SPI

```
connectionFactory.create()
    .flatMapMany { conn ->
        conn.createStatement(
            "INSERT INTO table (id, name, manual) VALUES($1, $2, $3)")
            .bind("$1", 42055)
            .bind("$2", "Description")
            .bindNull("$3", Integer::class.java)
            .execute()
    }
```

Insert Transactional SPI

```
connectionFactory.create()
    .flatMapMany { conn ->
        conn.beginTransaction()
            .thenMany(conn.createStatement(
                "INSERT INTO table (id, name, manual) VALUES($1, $2, $3)")
                .bind("$1", 42055)
                .bind("$2", "Description")
                .bindNull("$3", Integer::class.java)
                .execute())
            .delayUntil { conn.commitTransaction() }
            .onErrorResume { e ->
                conn.rollbackTransaction().then(Mono.error(e))
            }

    }
```

Select Client

```
val r2dbc = R2dbc(connectionFactory)

r2dbc.withHandle { handle ->
    handle.select("SELECT value FROM bkug")
        .mapRow { row -> row.get("value") }
}
```

Insert Client

```
r2dbc.withHandle { handle →
    handle.createUpdate("INSERT INTO bkug VALUES($1, $2)")
        .bind("$1", 100)
        .bind("$2", 200)
        .execute()
}
```

Insert Transactional Client

```
r2dbc.inTransaction { handle ->
    handle.createUpdate("INSERT INTO bkug VALUES($1, $2)")
        .bind("$1", 100)
        .bind("$2", 200)
        .execute()
}
```

Insert Transactional Client

```
r2dbc.inTransaction { handle →
    handle.createUpdate("INSERT INTO bkug VALUES($1, $2)")
        .bind("$1", 100)
        .bind("$2", 200)
        .execute()
}
```

Spring Data R2DBC

```
interface CustomerRepository extends ReactiveCrudRepository<Customer, Long> {

    @Query("select id, firstname, lastname from customer c where c.lastname = $1")
    Flux<Customer> findByLastnameLike(String lastname);
}
```

Spring Data R2DBC
Spring Fu

```
suspend fun count() = client.execute()
    .sql("SELECT COUNT(*) FROM users").fetch().one()

suspend fun findAll() = client.select()
    .from("users").asType(User::class).fetch().all()

suspend fun findOne(id: String) = client.execute()
    .sql("SELECT * FROM users WHERE login = \$1")
    .bind(1, id).fetch().one()
```

Benchmarks

Fin!