CADEC 2015 - REACTIVE

Non-blocking I/O and Reactive frameworks for scalable and resilient services

MAGNUS LARSSON

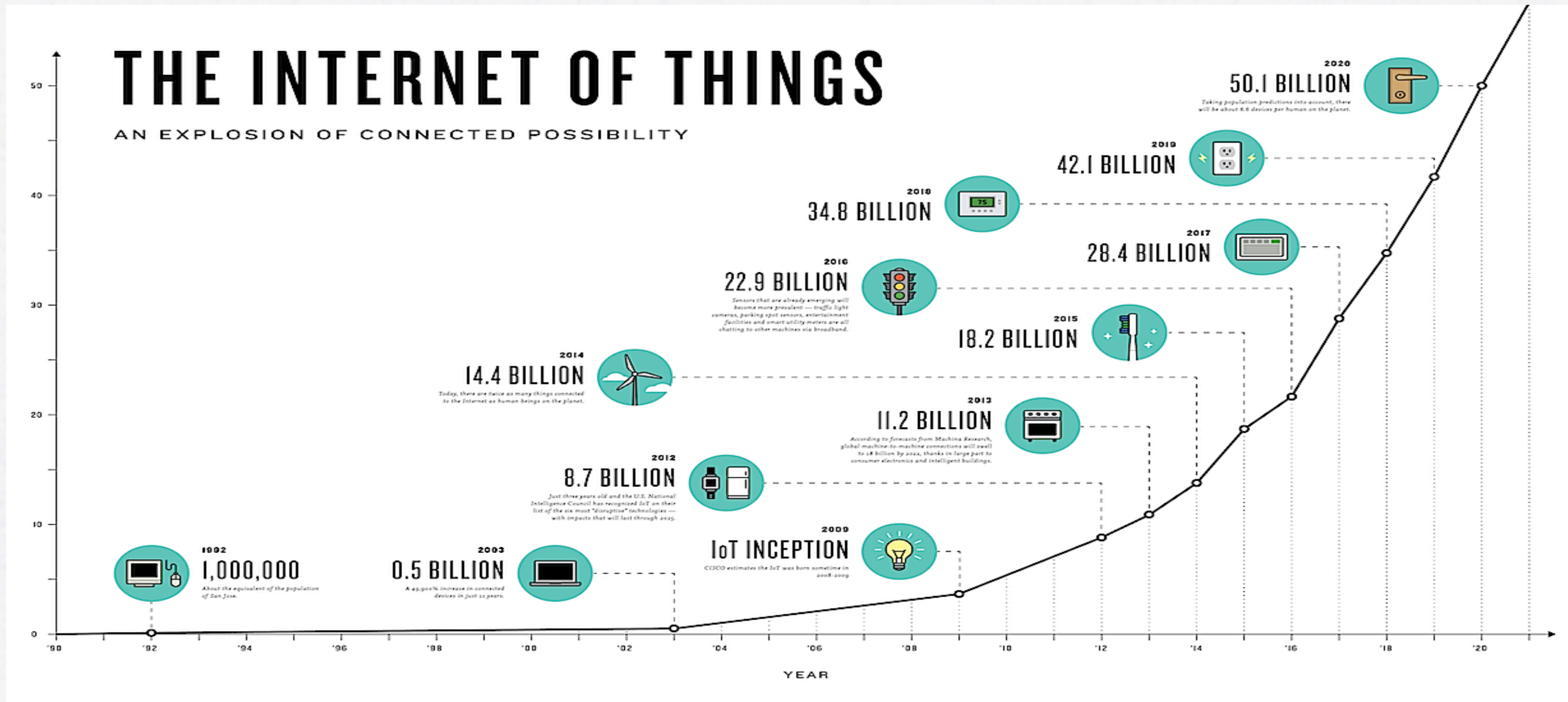
2015-01-28 | CALLISTAENTERPRISE.SE

CALLISTA
— ENTERPRISE —

AGENDA

- Overview
- Demo
- Some source code
- The devil is in the details…
- Summary

THE SCALABILITY CHALLENGE…



Source: http://www.theconnectivist.com/2014/05/infographic-the-growth-of-the-internet-of-things/

CALLISTA
— ENTERPRISE —

...SERVICES FAILS...

Source: http://techblog.netflix.com/2013/01/announcing-ribbon-tying-netflix-mid.html

CALLISTA
— ENTERPRISE —

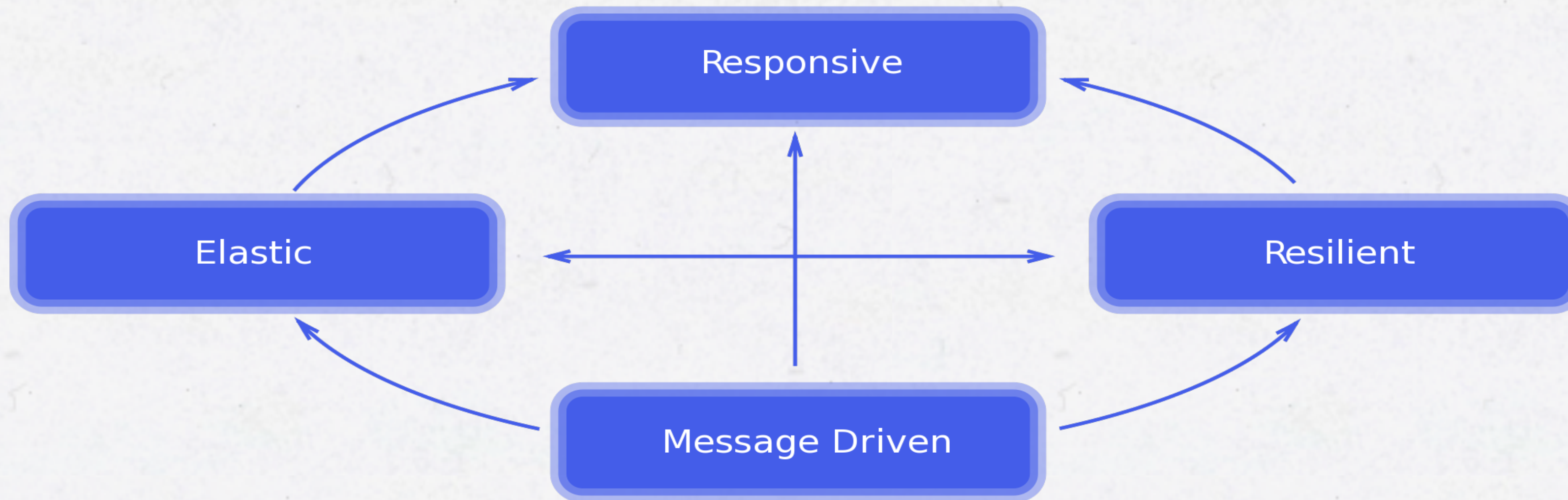
WATCH OUT FOR THE DOMINO EFFECT!

Source: http://techblog.netflix.com/2013/01/announcing-ribbon-tying-netflix-mid.html

CALLISTA
— ENTERPRISE —

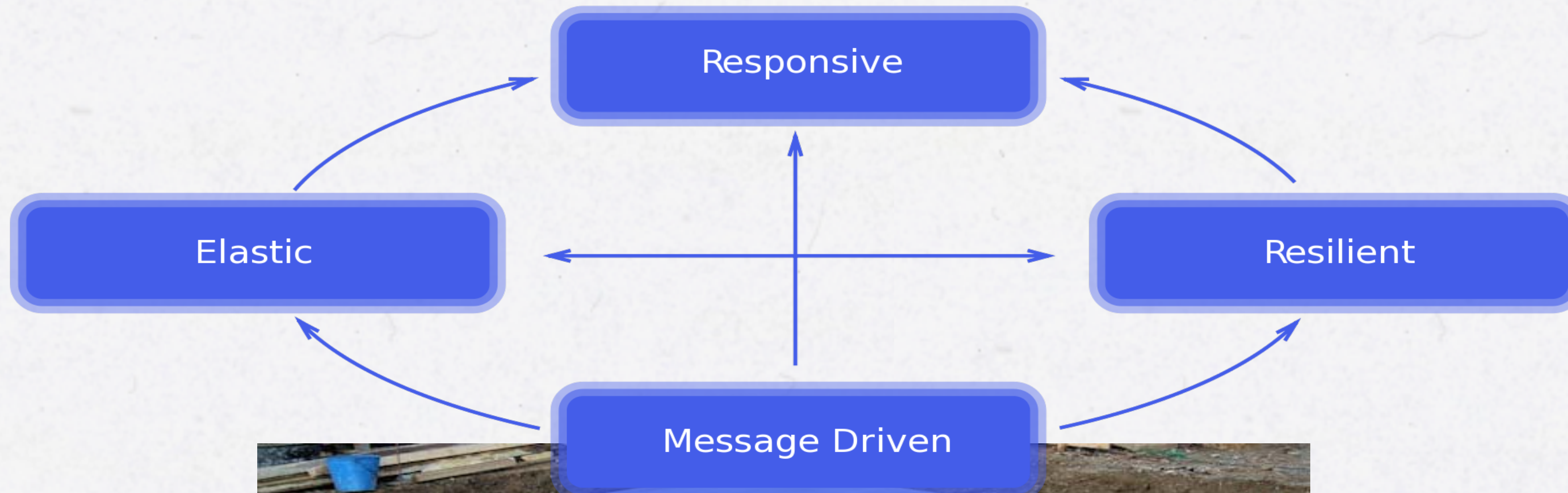
THE REACTIVE MANIFESTO

- http://www.reactivemanifesto.org



CALLISTA
— ENTERPRISE —

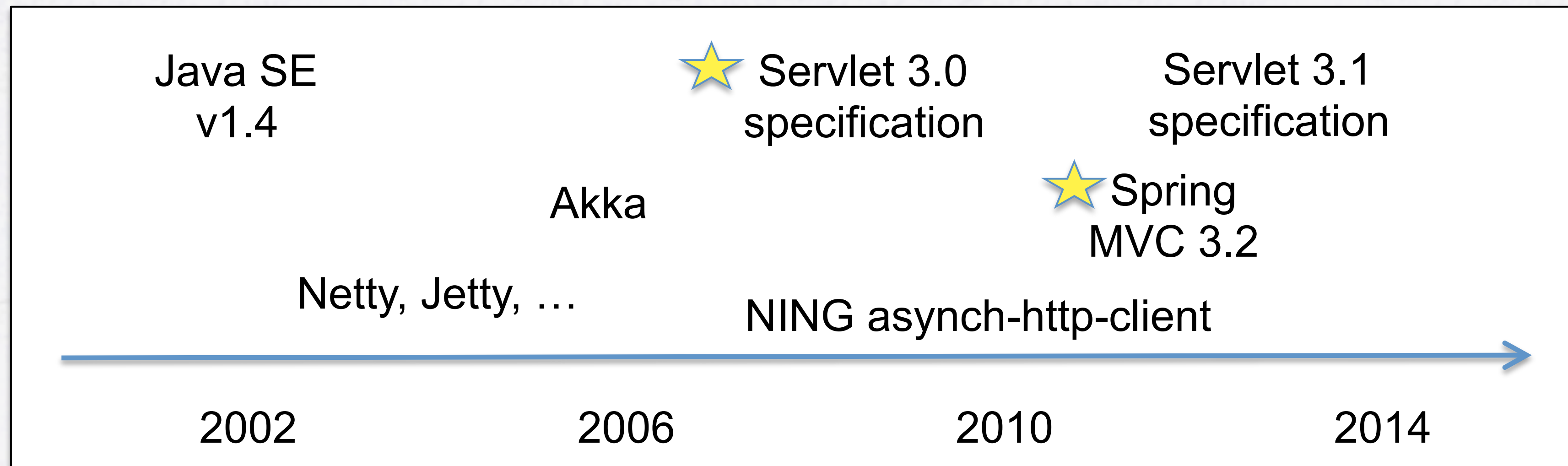
THE REACTIVE MANIFESTO



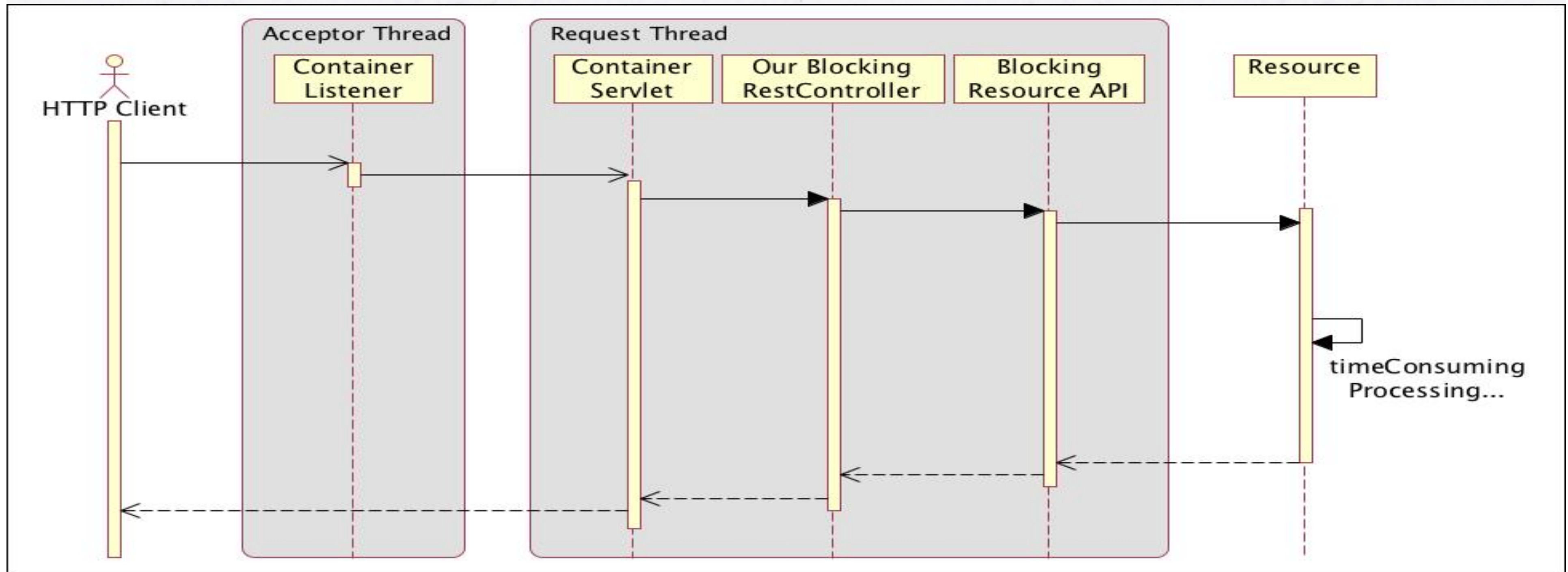
CALLISTA
— ENTERPRISE —

IS NON-BLOCKING I/O NEW?

- No!!!
- A short history lesson…
  - Supported in operating systems "for ever"
  - In Java SE since 2002
  - But it took some time to get mature, e.g. portable and easy to use…

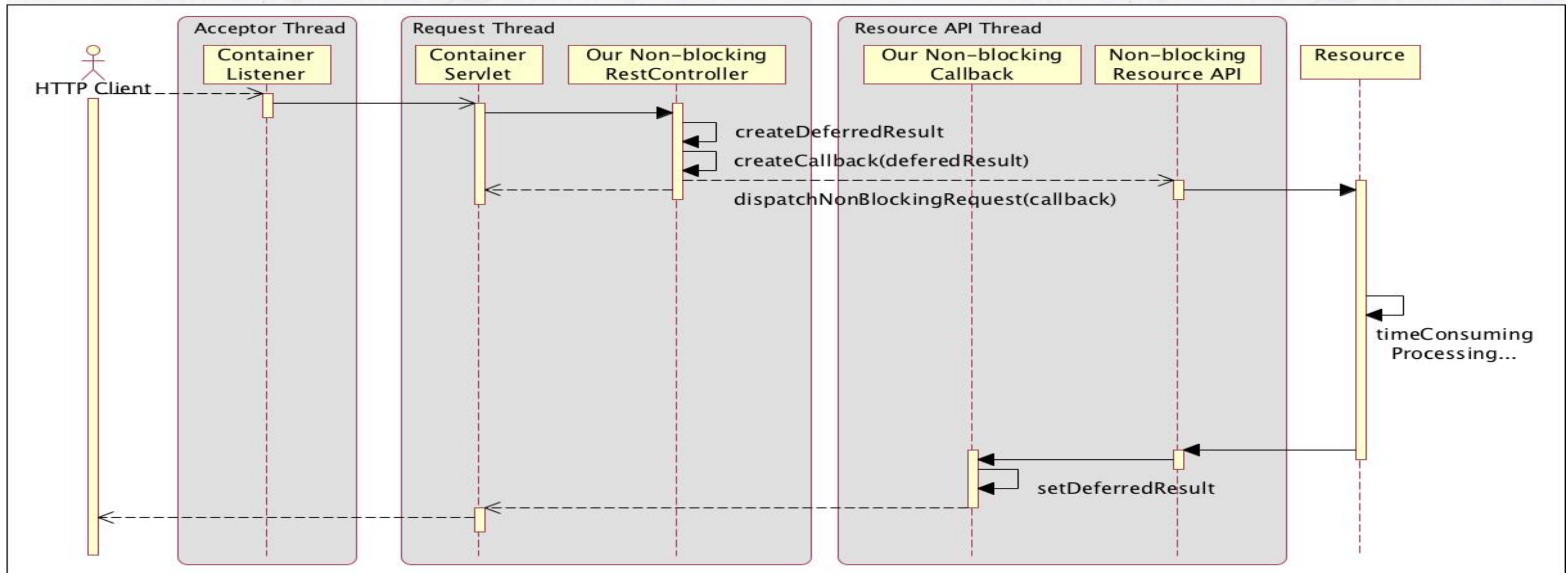


TRADITIONAL BLOCKING I/O



CALLISTA
— ENTERPRISE —

NON-BLOCKING I/O

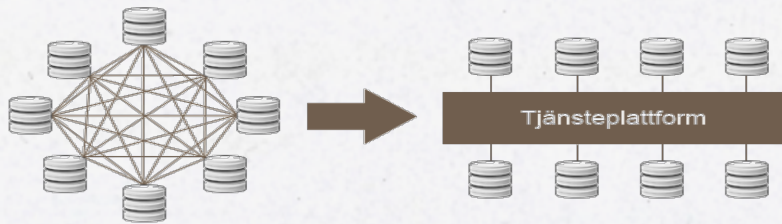


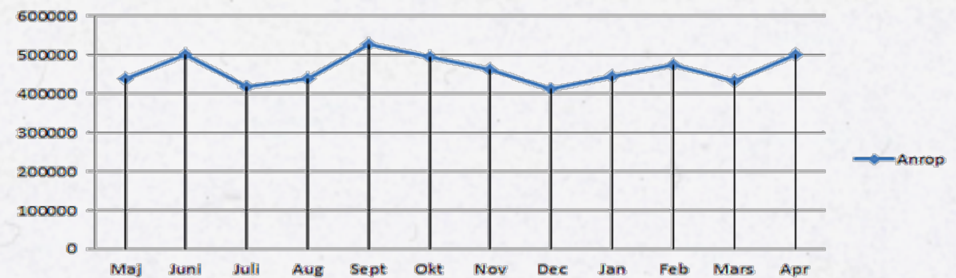
CALLISTA
— ENTERPRISE —

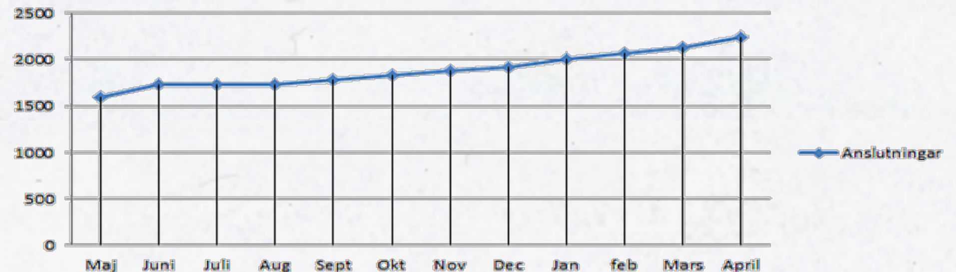
CALLISTA
— ENTERPRISE —

AN EXAMPLE OF POTENTIAL PROBLEMS WITH BLOCKING I/O

National Healthcare Service Platform



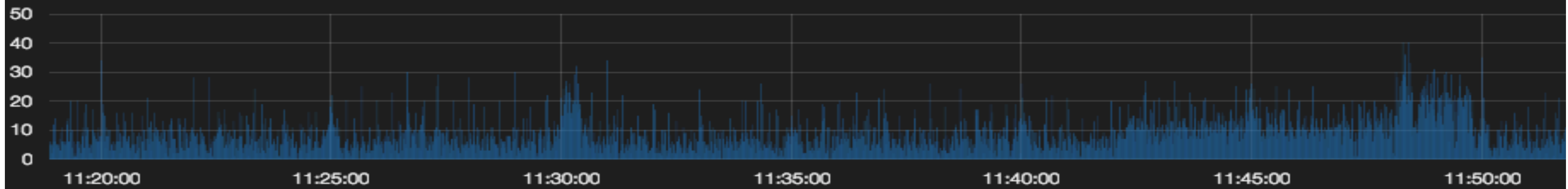


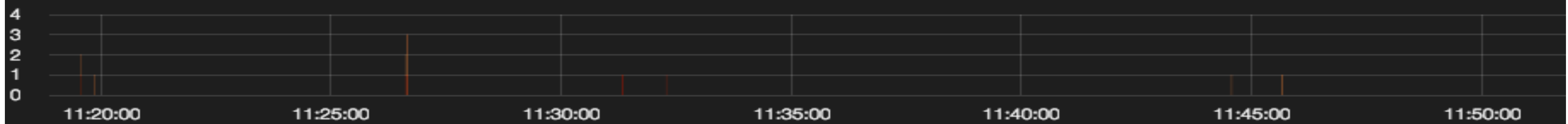


- National reference architecture
- Standardized protocols
- Standardized message formats
- Service catalog for routing
- In operation since 2010
  - > 2000 connected care units
  - > 500 000 messages/day (8h)

CALLISTA
— ENTERPRISE —

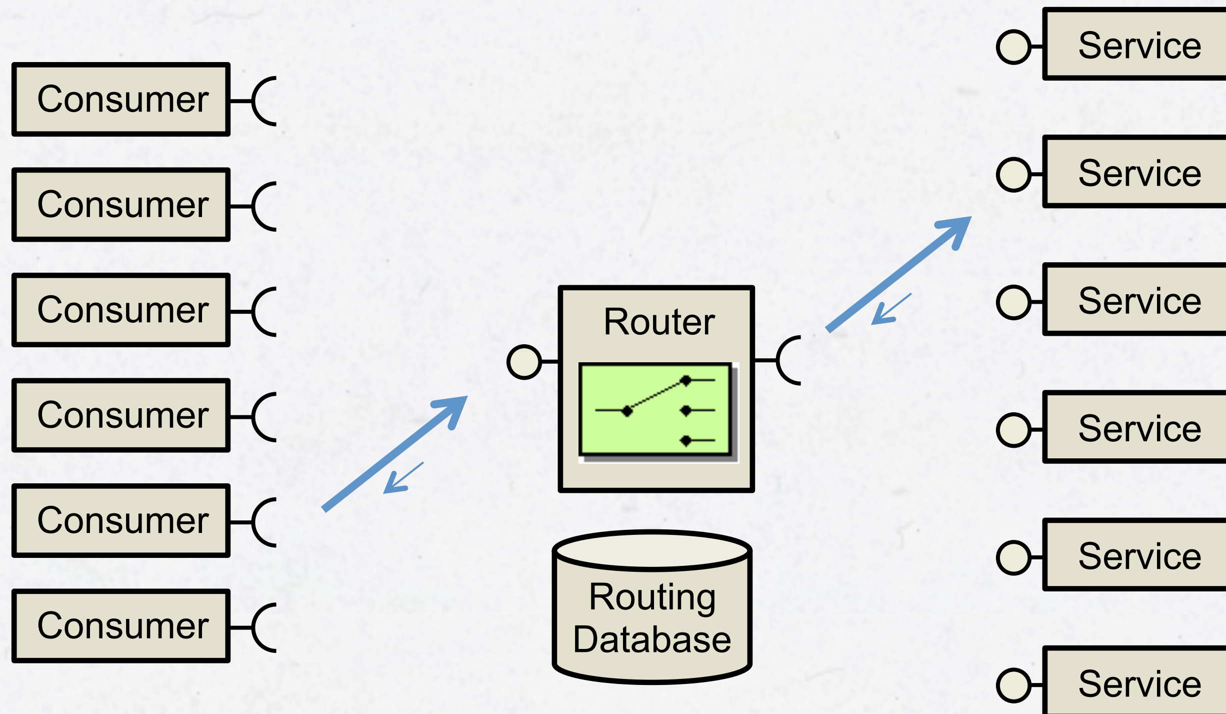
VIEW FROM THE RUNNING SYSTEM IN PRODUCTION





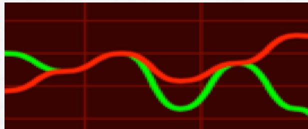
CALLISTA
— ENTERPRISE —

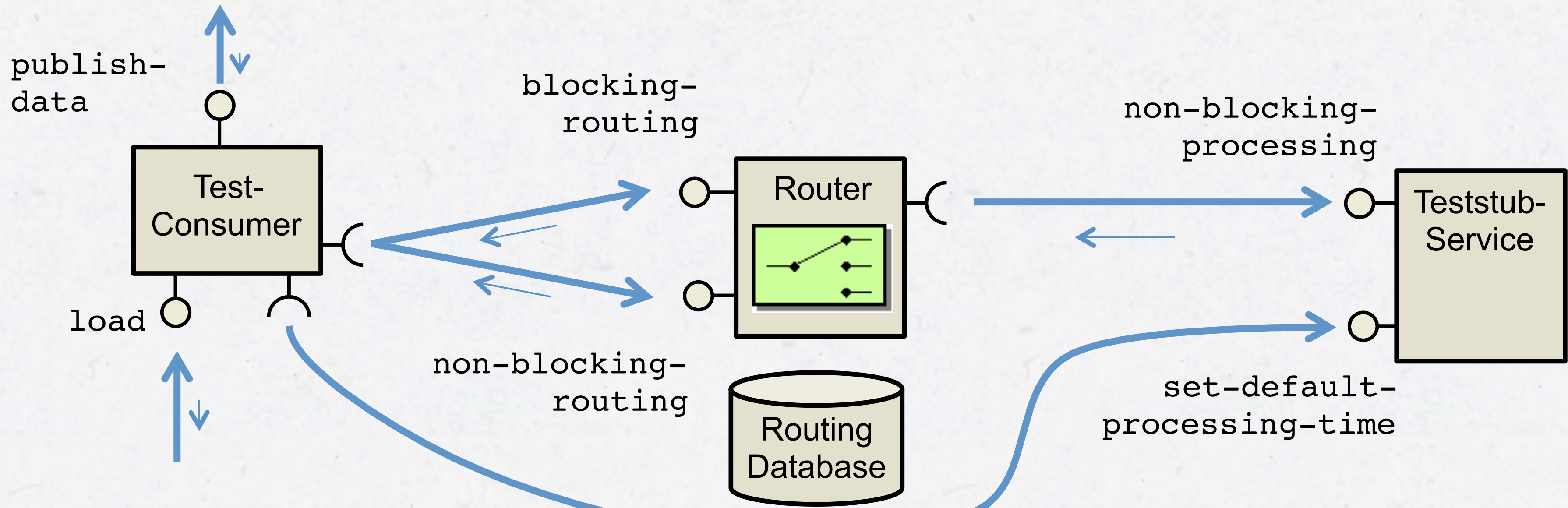
HIGH LEVEL ARCHITECTURE...



CALLISTA
— ENTERPRISE —

SIMULATION OF THE ENVIRONMENT

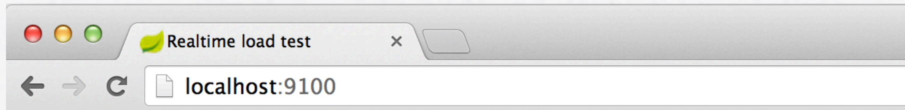




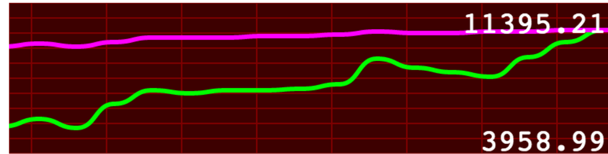

```
curl "http://localhost:9100/
  load?minMs=3000&maxMs=6000&
  test=2&tps=50"
```


CALLISTA
— ENTERPRISE —

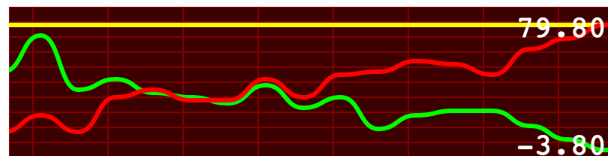
SAMPLE OUTPUT FROM A LOAD TEST



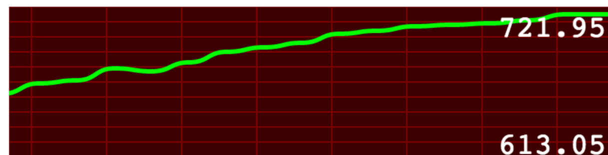
Response time and Processing time ms

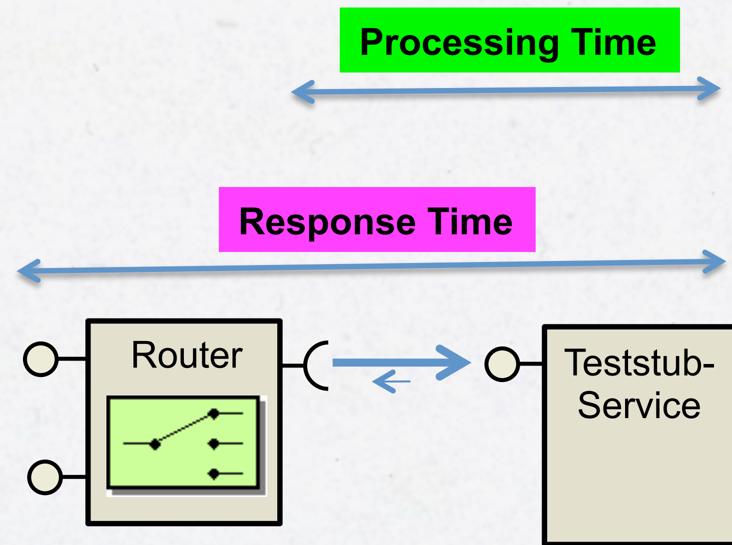


Requested , Actual and Error TPS



Concurrent requests





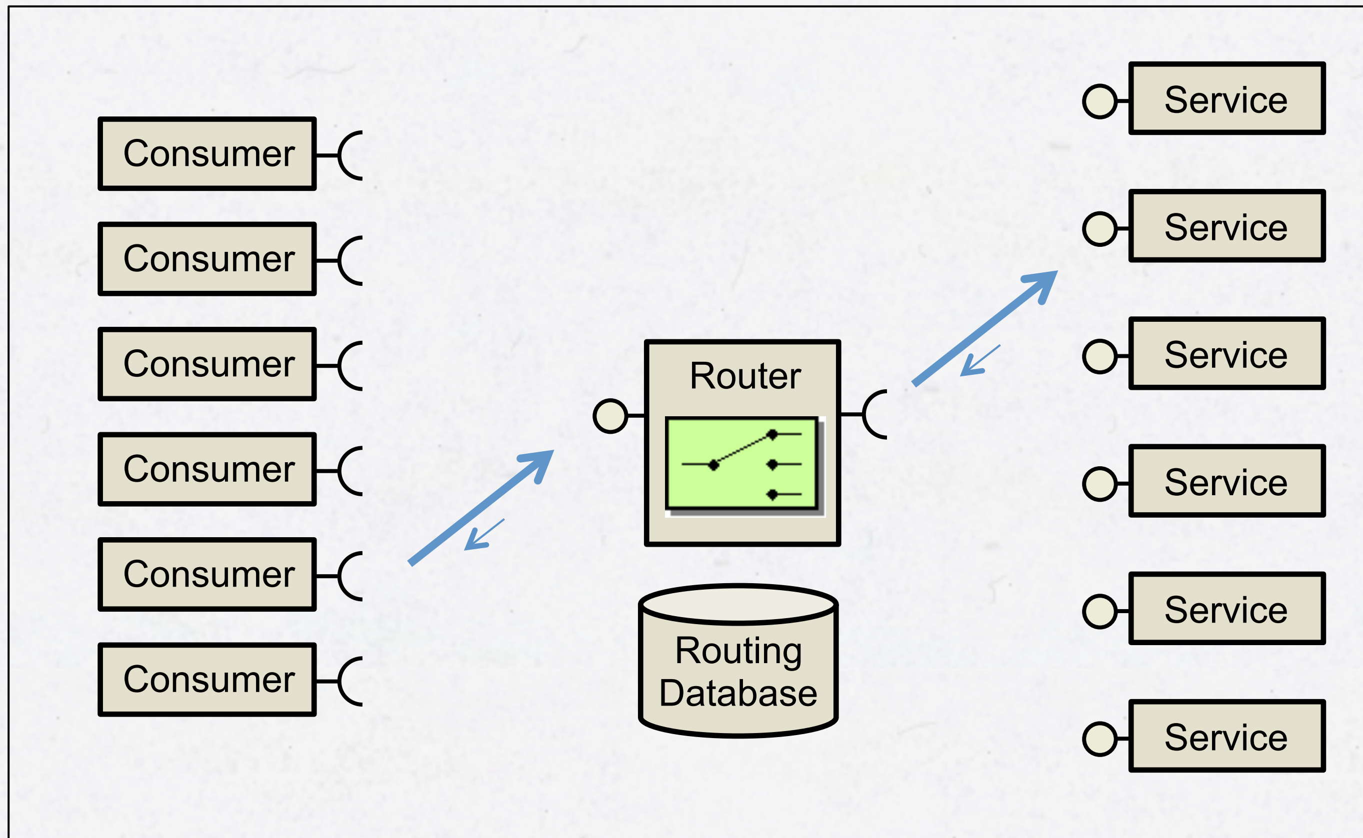
DEMO

- Normal load is
  - 20 – 50 reqs/s
  - Service Provider response times: 3-6 s
  - Default request timeout: 10 s
- Start with 20 reqs/s and step up to 50 reqs/s
- If ok
  - Add a increase of load, 65 reqs/s
  - Add a minor problem, increase response times by 1s
  - What happens? Why?
- Switch to non blocking I/O and **go unleashed**!!!

CALLISTA
— ENTERPRISE —

CALLISTA
— ENTERPRISE —

PATTERNS – A SIMPLE ROUTER



BLOCKING I/O WITH SPRING MVC

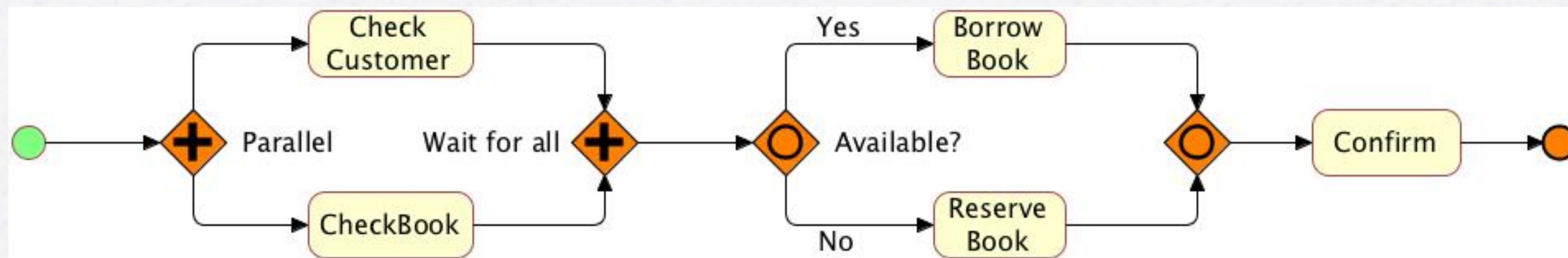
```
@RestController
public class RouterController {

 @RequestMapping("/router")
 public String router(
  @RequestParam String qry) {

  try {
   return restTemplate.getForObject(
    url + "?qry=" + qry, String.class);

  } catch (RuntimeException ex) {
   return util.handleException(ex, url);
  }
 }
}
```

NON-BLOCKING I/O WITH SPRING MVC

```java
@RestController
public class RouterController {

 @RequestMapping("/router")
 public String router(
  @RequestParam String qry) {

   try {
    return restTemplate.getForObject(
     url + "?qry=" + qry, String.class);

   } catch (RuntimeException ex) {
    return util.handleException(ex, url);
   }
 }
}
```

```java
@RequestMapping("/router")
public DeferredResult<String> router(
  @RequestParam String qry) {

  final DeferredResult<String> dr =
   new DeferredResult<>();

  asyncHttpClient.execute(url + "?qry=" + qry,

   throwable -> {
    util.handleException(throwable, url, dr);
   },

   response -> {
    dr.setResult(util.createResponse(response));
   }
  );

  // Return to let go of the precious thread
  // we are holding on to...
  return dr;
}
```

A SLIGHTLY MORE COMPLEX ROUTING SLIP



- Four rest-calls, one conditional, needs to be performed in sequence
- But we want to use Non Blocking I/O to be able to scale!
- …and we want to execute the first two in parallel to minimize latency

- What does that look like in code???

WHERE TO INITIATE THE NEXT PROCESSING STEP?

```
@RequestMapping("/router")
public DeferredResult<ResponseEntity<String>> router(@RequestParam String qry) {

    final DeferredResult<ResponseEntity<String>> dr = new DeferredResult<>();

    asyncHttpClient.execute(url + "/service?qry=" + qry,

        throwable -> {
                util.handleException(throwable, url, dr);
        },

        response -> {
            ...
        }
    );

    // Return to let go of the precious thread we are holding on to...
    return dr;
}
```


CALLISTA
— ENTERPRISE —

IT HAS TO GO INTO THE CALLBACK METHOD!

```
@RequestMapping("/router")
public DeferredResult<ResponseEntity<String>> router(@RequestParam String qry) {

    final DeferredResult<ResponseEntity<String>> dr = new DeferredResult<>();

    asyncHttpClient.execute(url + "/service?qry=" + qry,

        throwable -> {
                util.handleException(throwable, url, dr);
        },

        response -> {
            ...
        }
    );

    // Return to let go of the precious thread we are holding on to...
    return dr;
}
```

Here we can initiate the next processing step

CALLISTA
— ENTERPRISE —

ROUTING SLIP USING NON BLOCKING I/O

```
@RequestMapping("/bookLoan-callback")
public DeferredResult<ResponseEntity<String>> bookLoan(String bookId, String custId) {

  DeferredResult<ResponseEntity<String>> dr = new DeferredResult<>();

  util.execute(dr, "#1, check customer", "...?custId=" + custId,
    (Response r1) -> {

      util.execute(dr, "#2, check book", "...?bookId=" + bookId,
        (Response r2) -> {

          boolean isAvailable = util..getActionResult(r2).equals(RESULT_AVAILABLE);
          String requestName =  isAvailable ? "#3.1, borrow book" : "#3.2, reserve book";
          String action      =  isAvailable ? RESULT_BORROWED     : RESULT_RESERVED;
          String url3        =  "..." + "?bookId=" + bookId + "&custId=" + custId;

          util.execute(dr, requestName, url3,
            (Response r3) -> {

              util.execute(dr, "#4, confirm", "...?bookId=" + bookId + "&custId=" + custId,
                (Response r4) -> {

                  dr.setResult(util.getResult(action));
                  ...
```

ROUTING SLIP USING NON BLOCKING I/O

```
@RequestMapping("/bookLoan-callback")
public DeferredResult<ResponseEntity<String>> bookLoan(String bookId, String custId) {

  DeferredResult<ResponseEntity<String>> dr = new DeferredResult<>();

  util.execute(dr, "#1, check customer", "...?custId=" + custId,
    (Response r1) -> {

      util.execute(d
        (Response r2

          boolean is                                                          SULT_AVAILABLE);
          String req                                                          "#3.2, reserve book";
          String action       =  isAvailable ? RESULT_BORROWED      : RESULT_RESERVED;
          String url3         =  "..." + "?bookId=" + bookId + "&custId=" + custId;

          util.execute(dr, requestName, url3,
            (Response r3) -> {

              util.execute(dr, "#4, confirm", "...?bookId=" + bookId + "&custId=" + custId,
                (Response r4) -> {

                  dr.setResult(util.getResult(action));
                  ...
```

A.k.a "Callback Hell"

ROUTING SLIP USING NON BLOCKING I/O

```
                  ...
                  dr.setResult(util.getResult(action));
                }
              );
            }
          );
        }
      );
    }
  );

  // Return to let go of the precious thread we are holding on to...
  return deferredResult;
}
```

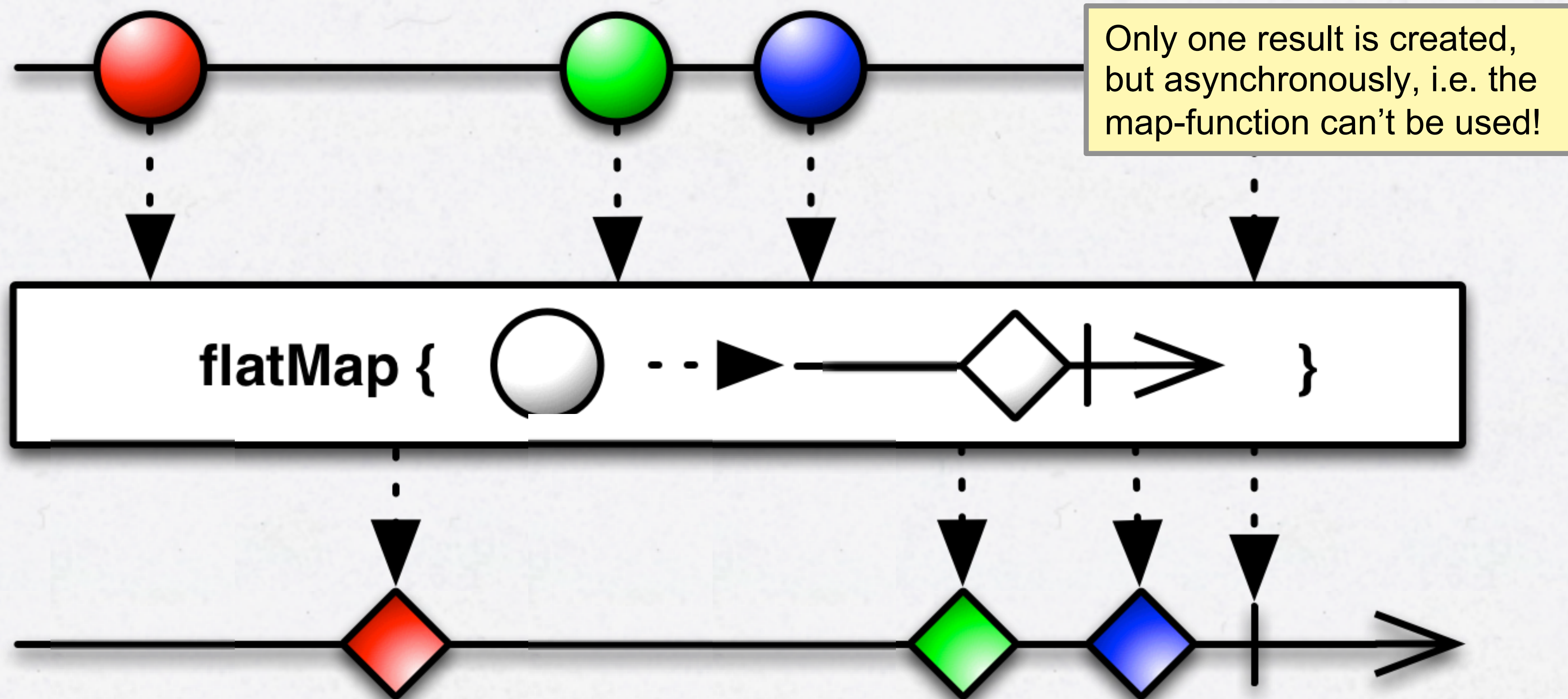

CALLISTA
— ENTERPRISE —

INTRODUCING RXJAVA

- A library for composing asynchronous and event-based programs
- Originated from Microsoft, https://rx.codeplex.com
- Netflix made the port to Java
- Several languages supported, http://reactivex.io
- Works on Java 7 (bye bye Lambdas…)
- Very clean API based on Observer/Observable
  - *"The Observer pattern done right!"*

CALLISTA
— ENTERPRISE —

FLATMAP FUNCTION WITH ASYNCH HTTP CLIENT



RXJAVA FLATMAP FUNCTION WITH ASYNCH-HTTP-CLIENT

```
Observable<State> observable = Observable.just(new State(query))
  .flatMap(state -> doAsyncCall(state, 1))
  .flatMap(state -> doAsyncCall(state, 2))
  .flatMap(state -> doAsyncCall(state, 3))
  .flatMap(state -> doAsyncCall(state, 4));

Subscription subscription = observable.subscribe(
  state     -> deferredResult.setResult(state.getTotalResult()),
  throwable -> deferredResult.setErrorResult(handleException(throwable))
);
```


CALLISTA
— ENTERPRISE —

A ROUTING SLIP PATTERN THE RXJAVA WAY

```
@RequestMapping("/bookLoan")
public DeferredResult<ResponseEntity<String>> routingSlip(String bookId, String custId) {

  final DeferredResult<ResponseEntity<String>> deferredResult = new DeferredResult<>();

  Observable<State> observable = Observable.from(Arrays.asList(
      new Request("#1, check customer", ”...?custId=" + custId),
      new Request("#2, check book",     ”...?bookId=" + bookId)))
    .flatMap(request -> util.execute(request))
    .buffer(2)

    .flatMap(results -> {
      State state = new State();
      if (util.isBookStatus(results, RESULT_AVAILABLE)) {
        state.setAction(RESULT_BORROWED);
        return util.execute(state, "#3.1, borrow book", ”...?bookId=" + bookId + "&custId=" + custId);
      } else {
        state.setAction(RESULT_RESERVED);
        return util.execute(state, "#3.2, reserve book", ”...?bookId=" + bookId + "&custId=" + custId);
      }
    })

    .flatMap(state -> util.execute(state, "#4, confirm", ”...?bookId=" + bookId + "&custId=" + custId));
```

A ROUTING SLIP PATTERN THE RXJAVA WAY

```
  ...

  // Subscribe to the observable, i.e. start the processing
  Subscription subscription = observable.subscribe(
    state -> {
      // We are done, create a response and send it back to the caller
      long processingTimeMs = System.currentTimeMillis() - timestamp;
      deferredResult.setResult(util.createResponse(state.getLastResponse()));
    },
    throwable -> {
      CommunicationException commEx = (CommunicationException) throwable;
      deferredResult.setErrorResult(commEx.getErrorResponse());
    }
  );

  // Unsubscribe, i.e. tear down any resources setup during the processing
  deferredResult.onCompletion(() -> subscription.unsubscribe());

  // Return to let go of the precious thread we are holding on to...
  return deferredResult;
}
```

CALLISTA
— ENTERPRISE —

THE DEVIL IS IN THE DETAILS…

- Logging
  - Use LogBack instead of Log4J to avoid scalability bottlenecks in Log4J
- Error handling
  - Communication, Service and Timeout errors
- Filters
  - I/O operations
  - Outbound filters
- ThreadLocal
  - E.g. logging MDC (correlation id), Request and/or Security Contexts
  - RxJava provides a callback for thread creation
- Tests
  - Spring Test MVC will get a DeferredResult-object as a response…

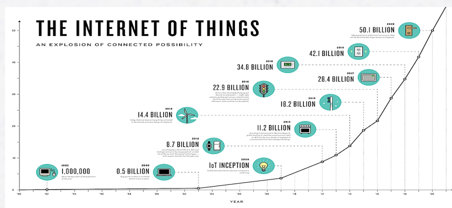
RECOMMMENDED READING

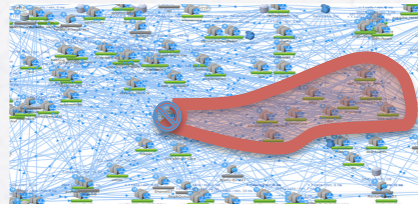
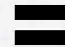
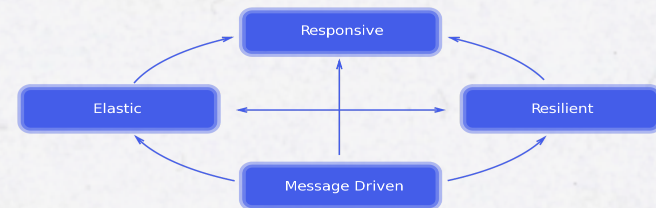
- Callista blogs
  - Trying out functional programming in Java 8
  - Testing nonblocking rest services with spring-mvc and spring-boot
  - Developing non-blocking REST services with Spring MVC
  - A first look at Spring Boot, is it time to leave XML based configuration behind?
- Callista presentations
  - Don't block your mobiles and Internet of things
  - Avoid callback hell when using non-blocking I/O
- New blogs and presentations are published on
  - Twitter
  - RSS

SUMMARY

- How log can you stay with Blocking I/O without risking your business?

- The solution



  - Asynchronous message passing and non blocking I/O
  - Avoid callback hell using
    - » Reactive frameworks
    - » Functional programming

We Are Reactive

QUESTIONS?

CALLISTA
— ENTERPRISE —