Sexy async code without *await*?

A first look into Project Loom in Java

What is this talk about?

- Blocking vs. non-blocking APIs
- Thread per request vs. event loop
- Callbacks and Futures?
- What is async/await?
- What is the blue/red world problem?
- Project Loom = *async/await* in Java?

About

Lukas Steinbrecher

Developer @ Senacor

lukstei.com
github.com/lukstei

lukas.steinbrecher@senacor.com

Chapter 1

Threads and what is a blocking call?

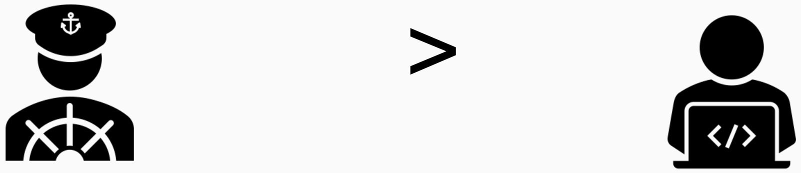
Super Duper Bank

“super duper product“:

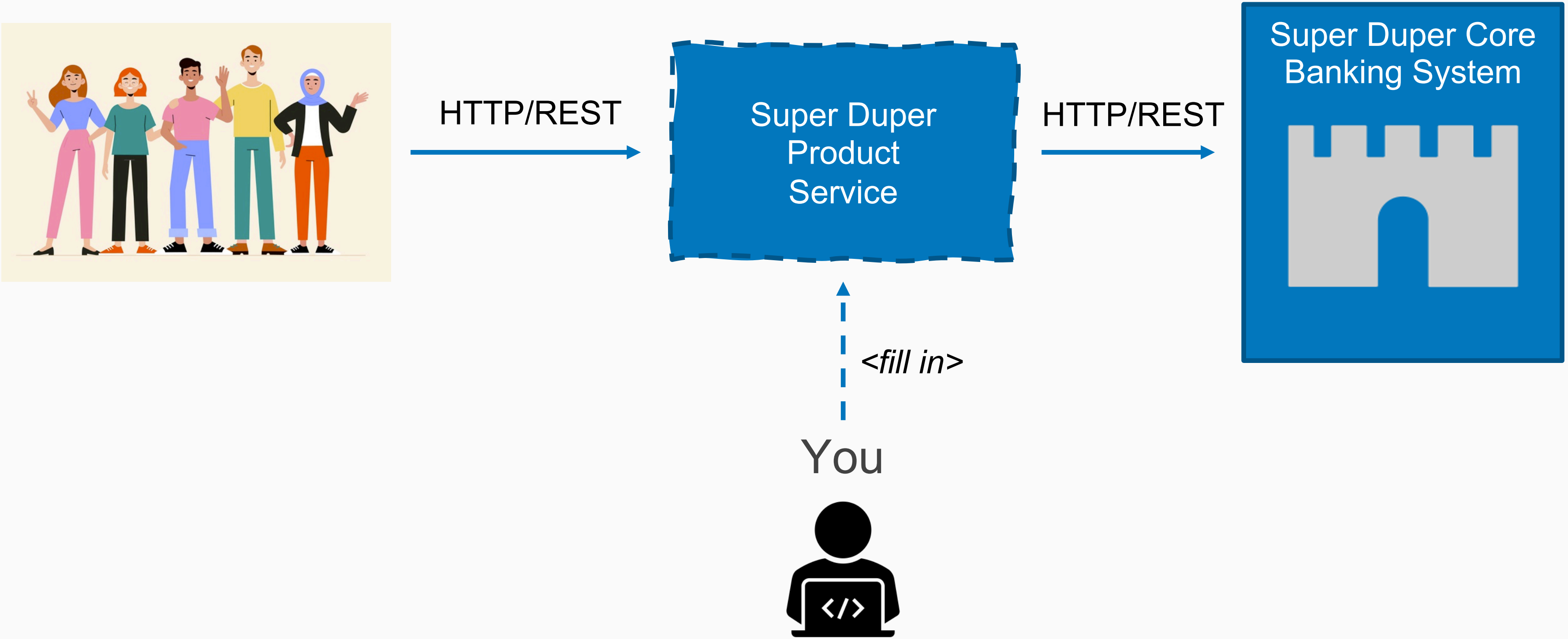
- Giro account
- Savings account

Boss B. “The Boss” Bossy

You



Super Duper Bank



HTTP/REST

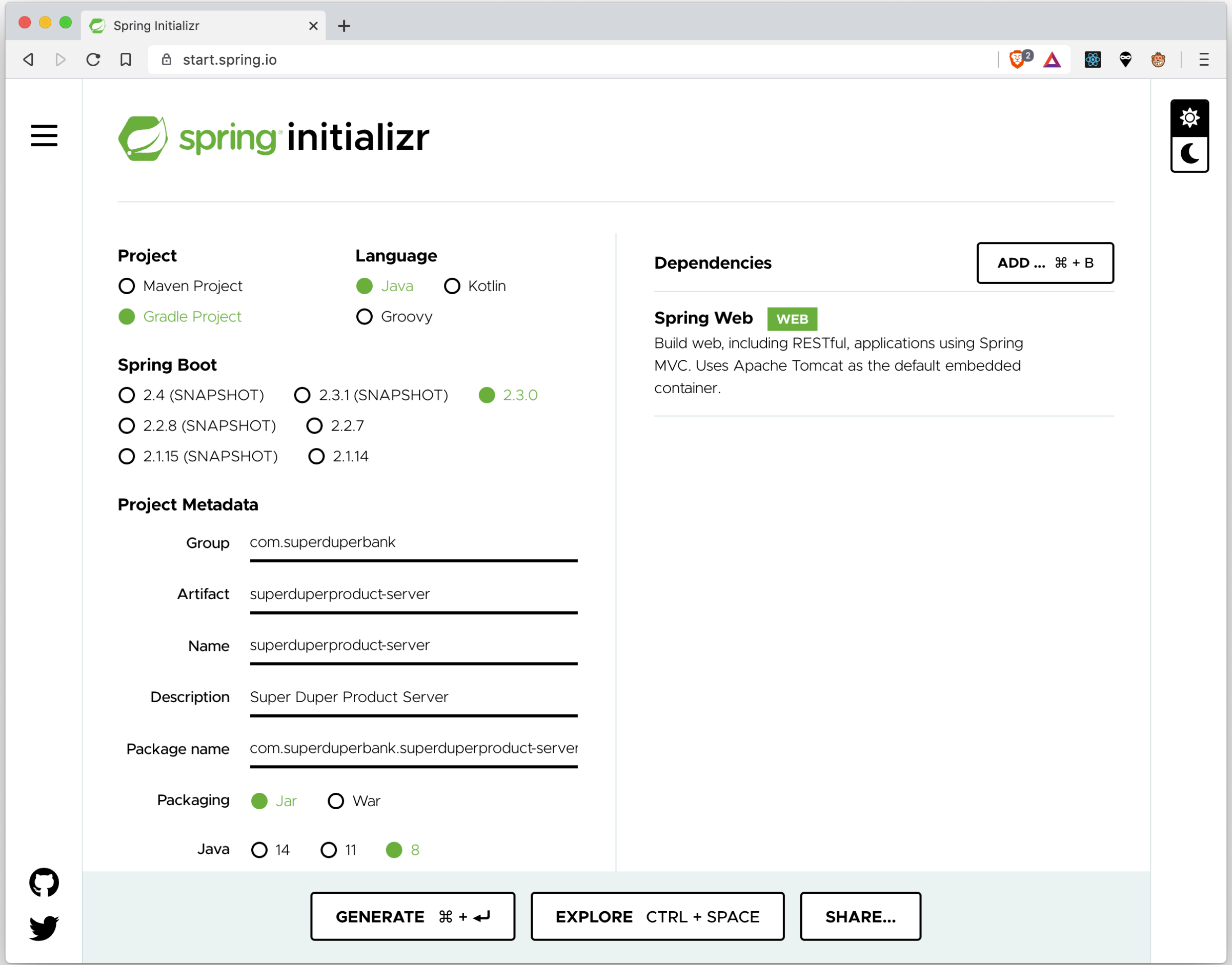
Super Duper Product Service

HTTP/REST

Super Duper Core Banking System

<fill in>

You



Project

○ Maven Project
● Gradle Project

Language

● Java ○ Kotlin
○ Groovy

Spring Boot

○ 2.4 (SNAPSHOT) ○ 2.3.1 (SNAPSHOT) ● 2.3.0
○ 2.2.8 (SNAPSHOT) ○ 2.2.7
○ 2.1.15 (SNAPSHOT) ○ 2.1.14

Project Metadata

Group com.superduperbank

Artifact superduperproduct-server

Name superduperproduct-server

Description Super Duper Product Server

Package name com.superduperbank.superduperproduct-server

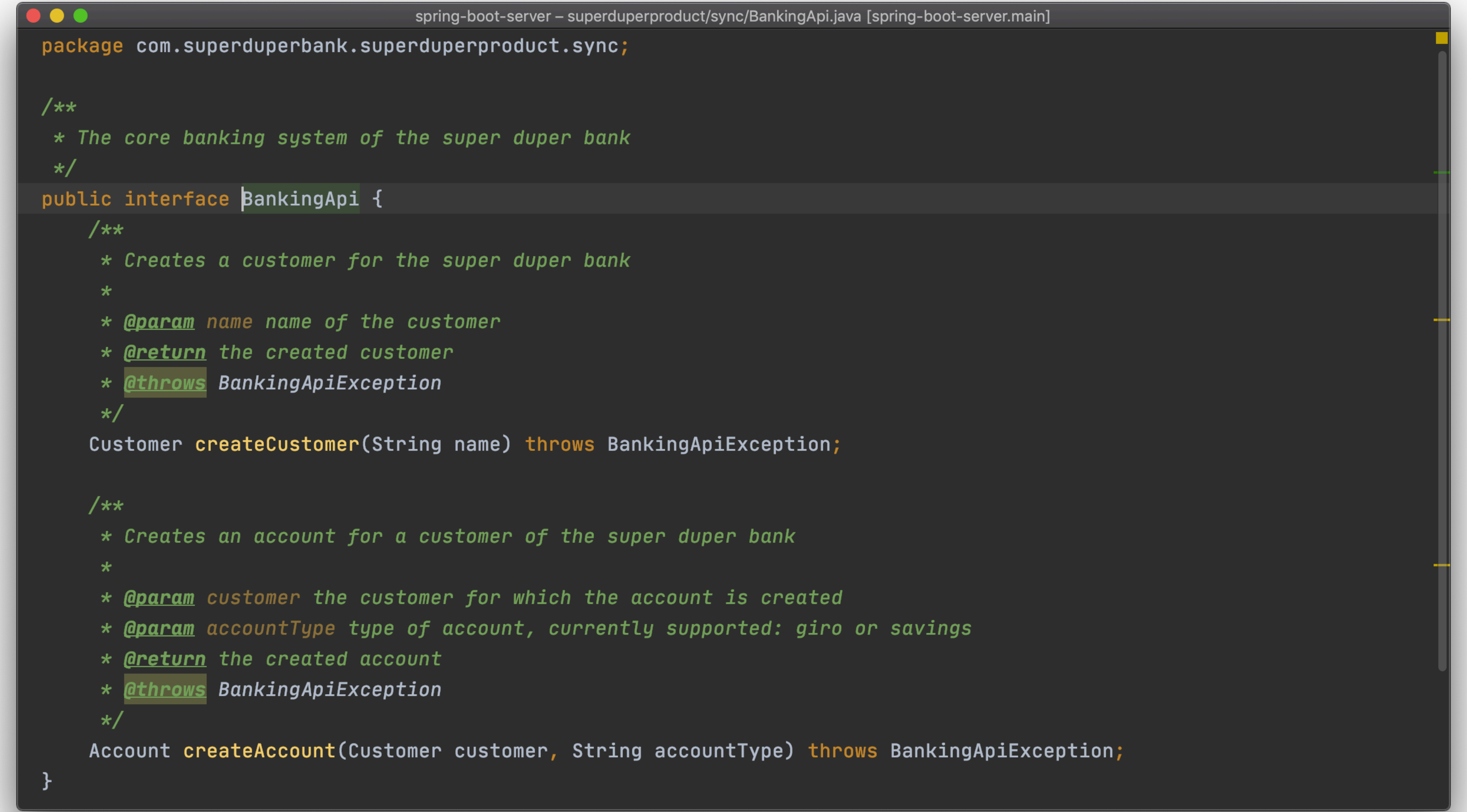
Packaging ● Jar ○ War

Java ○ 14 ○ 11 ● 8

Dependencies

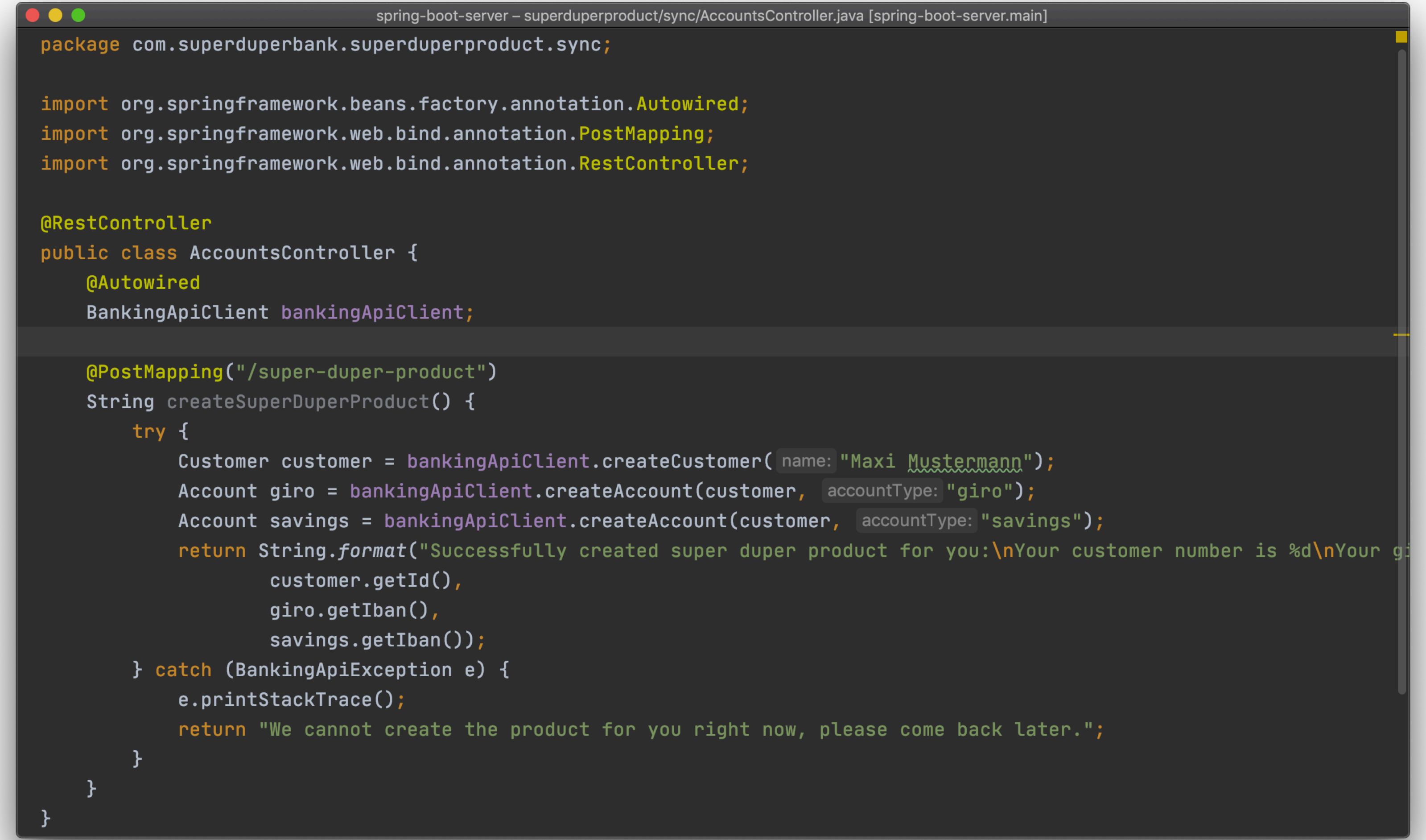
Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

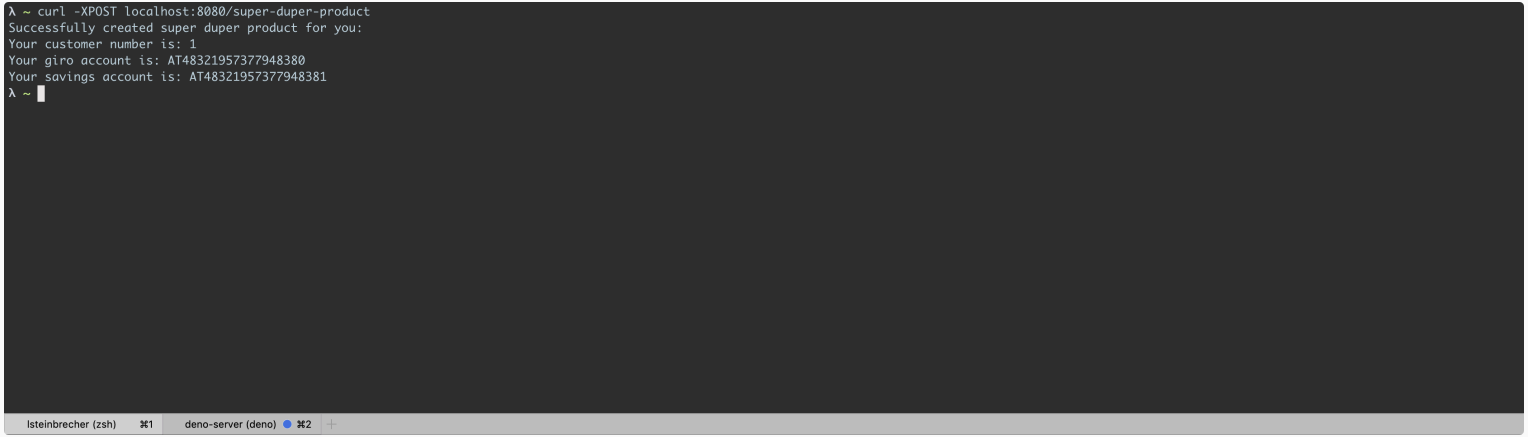


spring-boot-server – superduperproduct/sync/BankingApi.java [spring-boot-server.main]

```java
package com.superduperbank.superduperproduct.sync;

/**
 * The core banking system of the super duper bank
 */
public interface BankingApi {
    /**
     * Creates a customer for the super duper bank
     *
     * @param name name of the customer
     * @return the created customer
     * @throws BankingApiException
     */
    Customer createCustomer(String name) throws BankingApiException;

    /**
     * Creates an account for a customer of the super duper bank
     *
     * @param customer the customer for which the account is created
     * @param accountType type of account, currently supported: giro or savings
     * @return the created account
     * @throws BankingApiException
     */
    Account createAccount(Customer customer, String accountType) throws BankingApiException;
}
```


spring-boot-server – superduperproduct/sync/AccountsController.java [spring-boot-server.main]

```java
package com.superduperbank.superduperproduct.sync;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class AccountsController {
    @Autowired
    BankingApiClient bankingApiClient;

    @PostMapping("/super-duper-product")
    String createSuperDuperProduct() {
        try {
            Customer customer = bankingApiClient.createCustomer( name: "Maxi Mustermann");
            Account giro = bankingApiClient.createAccount(customer,  accountType: "giro");
            Account savings = bankingApiClient.createAccount(customer,  accountType: "savings");
            return String.format("Successfully created super duper product for you:\nYour customer number is %d\nYour gi
                    customer.getId(),
                    giro.getIban(),
                    savings.getIban());
        } catch (BankingApiException e) {
            e.printStackTrace();
            return "We cannot create the product for you right now, please come back later.";
        }
    }
}
```



```
λ ~ curl -XPOST localhost:8080/super-duper-product
Successfully created super duper product for you:
Your customer number is: 1
Your giro account is: AT48321957377948380
Your savings account is: AT48321957377948381
λ ~
```

Super Duper Bank

How many customers can we serve?

Boss B. “The Boss” Bossy

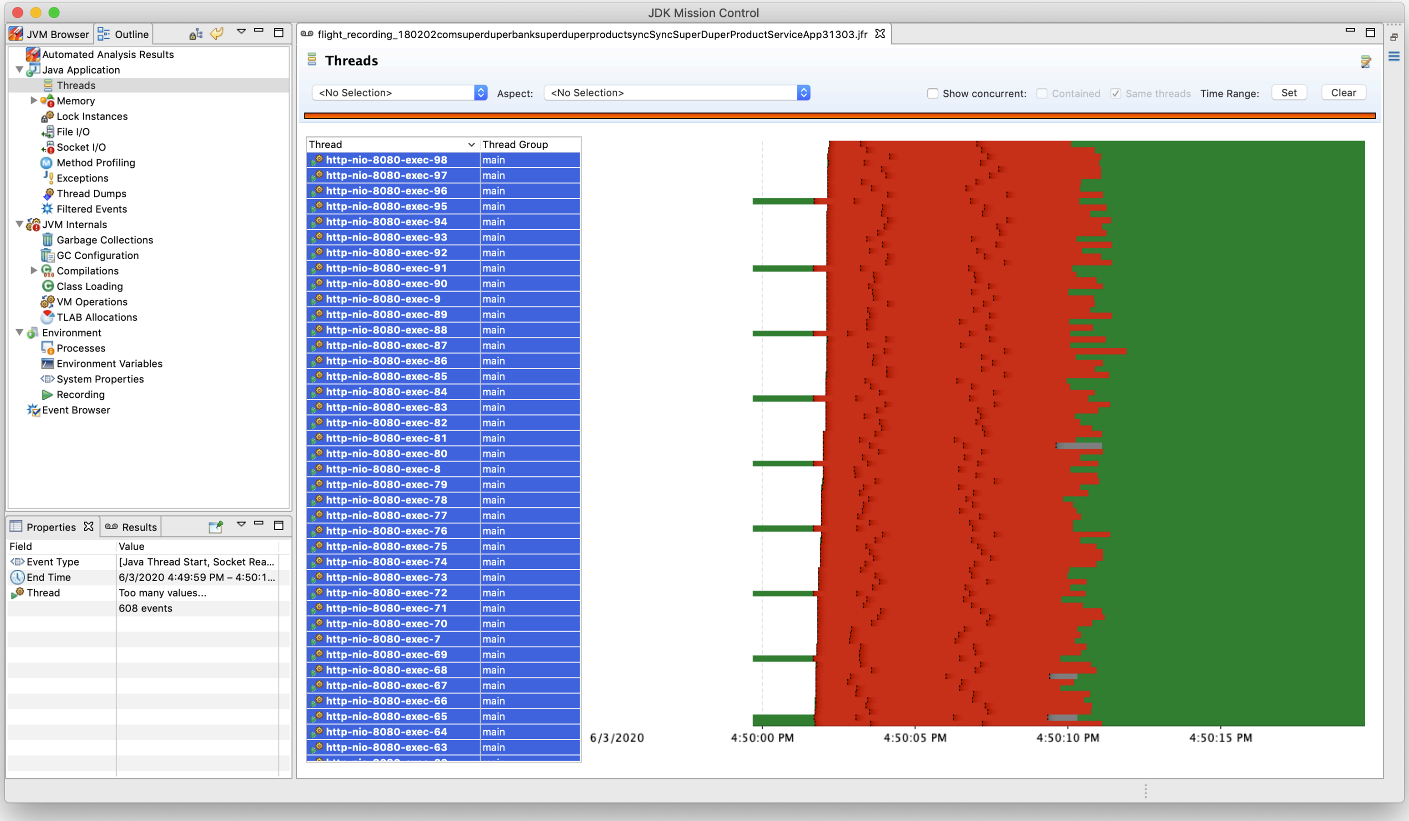
>

You

I won’t fall!

Super Duper Core Banking System

Simulation of 100 requests in Spring Web

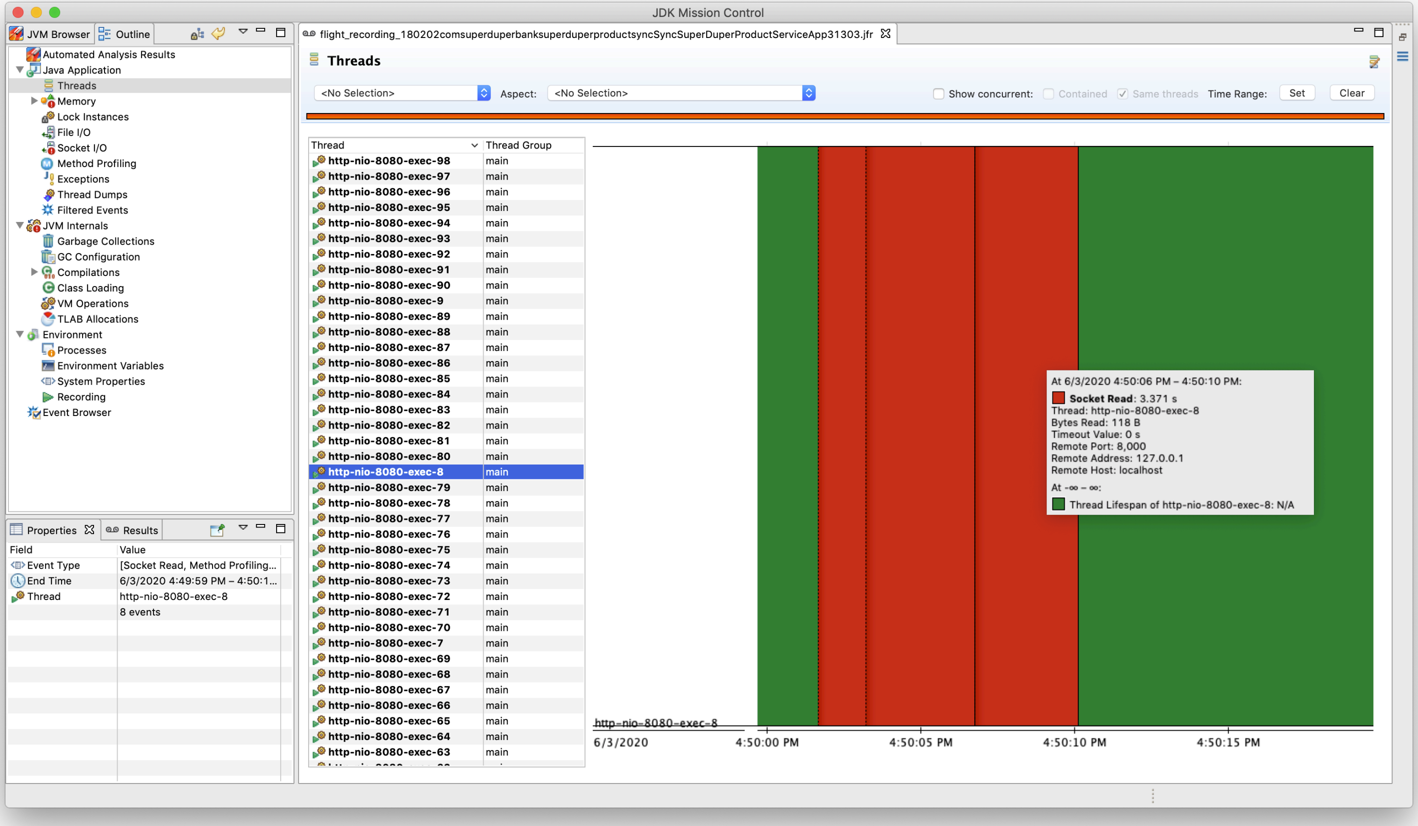


JDK Mission Control

Field	Value
Event Type	[Java Thread Start, Socket Rea...
End Time	6/3/2020 4:49:59 PM – 4:50:1...
Thread	Too many values...
	608 events

Thread	Thread Group
http-nio-8080-exec-98	main
http-nio-8080-exec-97	main
http-nio-8080-exec-96	main
http-nio-8080-exec-95	main
http-nio-8080-exec-94	main
http-nio-8080-exec-93	main
http-nio-8080-exec-92	main
http-nio-8080-exec-91	main
http-nio-8080-exec-90	main
http-nio-8080-exec-9	main
http-nio-8080-exec-89	main
http-nio-8080-exec-88	main
http-nio-8080-exec-87	main
http-nio-8080-exec-86	main
http-nio-8080-exec-85	main
http-nio-8080-exec-84	main
http-nio-8080-exec-83	main
http-nio-8080-exec-82	main
http-nio-8080-exec-81	main
http-nio-8080-exec-80	main
http-nio-8080-exec-8	main
http-nio-8080-exec-79	main
http-nio-8080-exec-78	main
http-nio-8080-exec-77	main
http-nio-8080-exec-76	main
http-nio-8080-exec-75	main
http-nio-8080-exec-74	main
http-nio-8080-exec-73	main
http-nio-8080-exec-72	main
http-nio-8080-exec-71	main
http-nio-8080-exec-70	main
http-nio-8080-exec-7	main
http-nio-8080-exec-69	main
http-nio-8080-exec-68	main
http-nio-8080-exec-67	main
http-nio-8080-exec-66	main
http-nio-8080-exec-65	main
http-nio-8080-exec-64	main
http-nio-8080-exec-63	main

Simulation of 100 requests in Spring Web



Thread	Thread Group
http-nio-8080-exec-98	main
http-nio-8080-exec-97	main
http-nio-8080-exec-96	main
http-nio-8080-exec-95	main
http-nio-8080-exec-94	main
http-nio-8080-exec-93	main
http-nio-8080-exec-92	main
http-nio-8080-exec-91	main
http-nio-8080-exec-90	main
http-nio-8080-exec-9	main
http-nio-8080-exec-89	main
http-nio-8080-exec-88	main
http-nio-8080-exec-87	main
http-nio-8080-exec-86	main
http-nio-8080-exec-85	main
http-nio-8080-exec-84	main
http-nio-8080-exec-83	main
http-nio-8080-exec-82	main
http-nio-8080-exec-81	main
http-nio-8080-exec-80	main
http-nio-8080-exec-8	main
http-nio-8080-exec-79	main
http-nio-8080-exec-78	main
http-nio-8080-exec-77	main
http-nio-8080-exec-76	main
http-nio-8080-exec-75	main
http-nio-8080-exec-74	main
http-nio-8080-exec-73	main
http-nio-8080-exec-72	main
http-nio-8080-exec-71	main
http-nio-8080-exec-70	main
http-nio-8080-exec-7	main
http-nio-8080-exec-69	main
http-nio-8080-exec-68	main
http-nio-8080-exec-67	main
http-nio-8080-exec-66	main
http-nio-8080-exec-65	main
http-nio-8080-exec-64	main
http-nio-8080-exec-63	main

At 6/3/2020 4:50:06 PM – 4:50:10 PM:

🟥 **Socket Read**: 3.371 s
Thread: http-nio-8080-exec-8
Bytes Read: 118 B
Timeout Value: 0 s
Remote Port: 8,000
Remote Address: 127.0.0.1
Remote Host: localhost

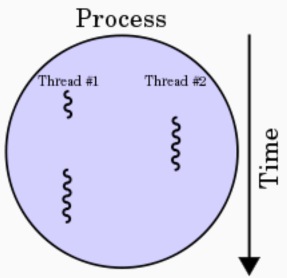
At -∞ – ∞:

🟩 Thread Lifespan of http-nio-8080-exec-8: N/A

Field	Value
Event Type	[Socket Read, Method Profiling...
End Time	6/3/2020 4:49:59 PM – 4:50:1...
Thread	http-nio-8080-exec-8
	8 events

The thread

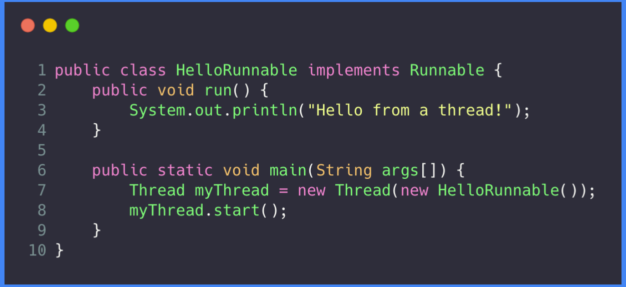
- Mechanism to provide multitasking in one process
- OS[1] threads must support all use cases and programming languages → not very optimized
- Context-switching slow
- Relatively heavy (> 2kb metadata, > 1mb stack size)



→ OS Threads are a limited resource, ~up to a few 1000 threads on a normal computer

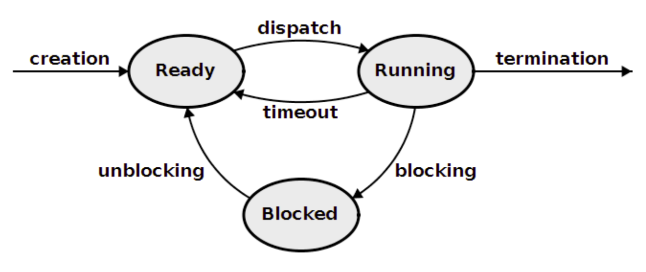
[1] OS: Operating System

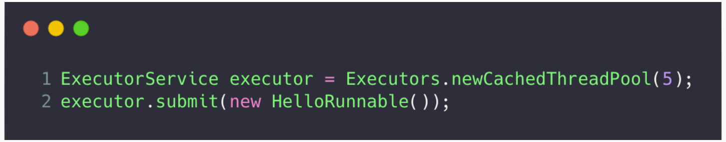
Threads in Java



```java
public class HelloRunnable implements Runnable {
    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        Thread myThread = new Thread(new HelloRunnable());
        myThread.start();
    }
}
```

- java.lang.Thread wraps native OS threads
- To create a thread, create an instance of the Thread class and call the *start()* method
- *java.util.concurrent.Executors* for a higher level API (thread pools, etc.)

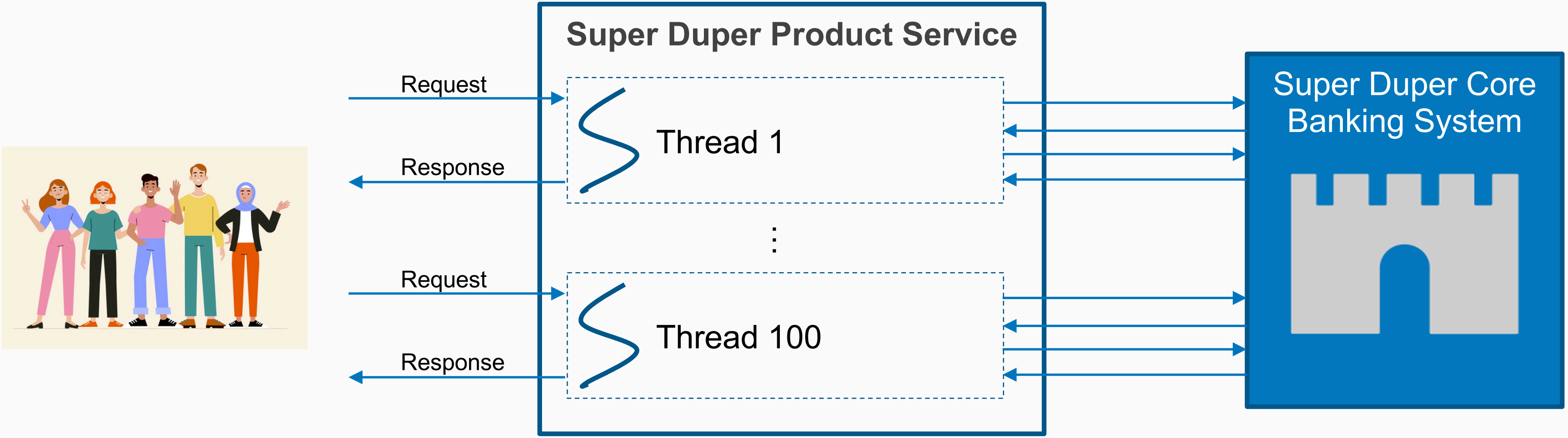




```java
ExecutorService executor = Executors.newCachedThreadPool(5);
executor.submit(new HelloRunnable());
```

Source: https://www.d.umn.edu/~gshute/os/processes-and-threads.xhtml

Thread per request model



Super Duper Product Service

Request

Response

Thread 1

⋮

Request

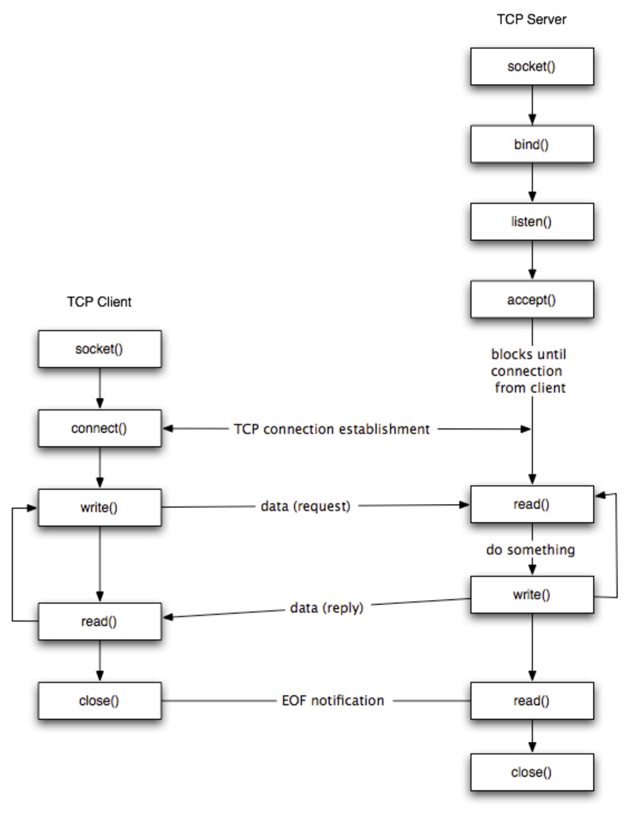
Response

Thread 100

Super Duper Core Banking System

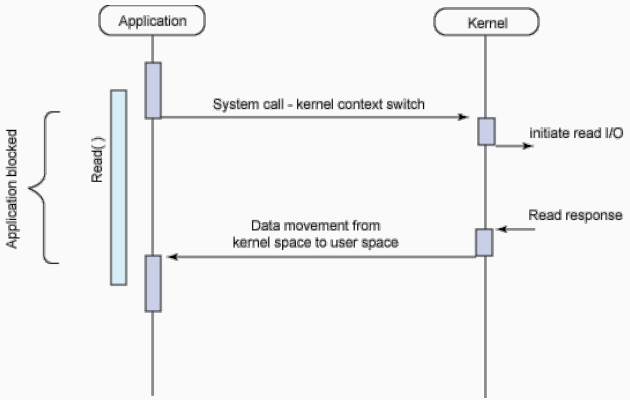
Networking 101

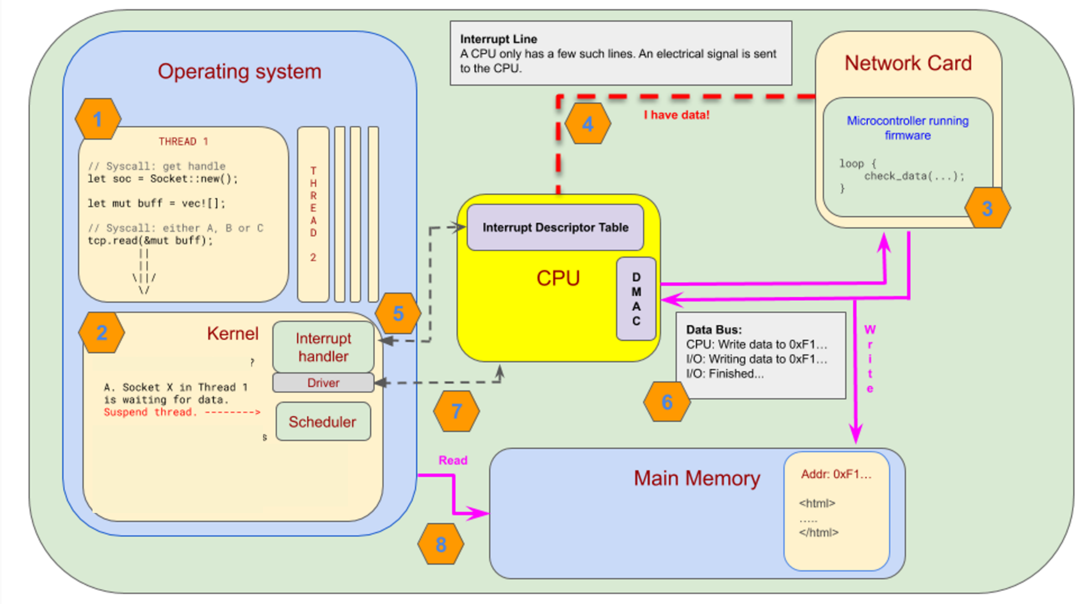
- OS responsible for coordinating access to external devices, e.g. network card
- OS provides primitives and functions (syscalls) to access those resources
- → Sockets as the primitive to access the network
- Every programming language uses this primitives under the hood



Source: https://www.cs.dartmouth.edu/~campbell/cs60/socketprogramming.html

What happens inside a blocking syscall?


Application
Kernel
System call - kernel context switch
initiate read I/O
Read response
Data movement from kernel space to user space
Application blocked
Read( )



```
// Syscall: get handle
let soc = Socket::new();

let mut buff = vec![];

// Syscall: either A, B or C
tcp.read(&mut buff);
        ||
        ||
       \||/
        \/
```

```
A. Socket X in Thread 1
is waiting for data.
Suspend thread. --------->
```

```
loop {
    check_data(...);
}
```

```
<html>
.....
</html>
```

Interrupt Line
A CPU only has a few such lines. An electrical signal is sent to the CPU.

Data Bus:
CPU: Write data to 0xF1…
I/O: Writing data to 0xF1…
I/O: Finished...

Source:
https://cfsamson.github.io/book-exploring-async-basics/4_interrupts_firmware_io.html
https://medium.com/martinomburajr/rxjava2-schedulers-2-breaking-down-the-i-o-scheduler-7e83160df2ed

SUNNY
HIGH 84, LOW 62–PAGE D8

The Washington Times

THURSDAY, JUNE 29, 1989 ★ WASHINGTON, D.C. PHONE: (202) 636-3000
SUBSCRIBER SERVICE: (202) 636-3333 25 cents

Major operating systems have been faking synchronous I/O for years

linux, windows, os x all implicated in kernel scandal of the century

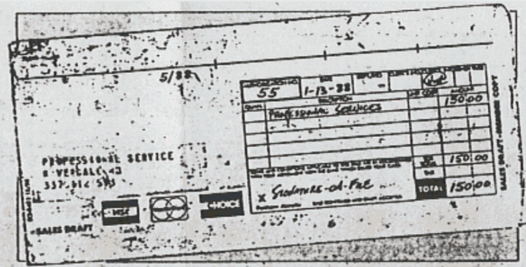
By Paul M. Rodriguez
and George Archibald

A homosexual prostitution ring is under investigation by federal and District authorities and includes among its clients key officials of the Reagan and Bush administrations, military officers, congressional aides and U.S. and foreign businessmen with close social ties to Washington's political elite, documents obtained by The Washington Times reveal.

One of the ring's high-profile clients was so well-connected, in fact, that he could arrange a middle-of-the-night tour of the White House for his friends on Sunday, July 3, of last year. Among the six persons on the extraordinary 1 a.m. tour were two male prostitutes.

Federal authorities, including the Secret Service, are investigating criminal aspects of the ring and have told male prostitutes and their homosexual clients that a grand jury will deliberate over the evidence throughout the summer, The Times learned.

Reporters for this newspaper examined hundreds of credit-card vouchers, drawn on both corporate and personal cards and made payable to the escort service operated by the homosexual ring. Many of the vouchers were run through a so-called "sub-merchant" account of the Chambers Funeral Home by a son of the owner, without the company's knowledge.



Among the client names contained in the vouchers — and identified by prostitutes and escort operators — are government officials, locally based U.S. military officers, businessmen, lawyers, bankers, congressional aides and other professionals.

Editors of The Times said the newspaper would print only the names of those found to be in sensitive government posts or positions of influence. "There is no intention of publishing names or facts about the operation merely for titillation," said Wesley Pruden, managing editor of The Times.

The office of U.S. Attorney Jay B. Stephens, former deputy White House counsel to President Reagan, is coordinating federal aspects of the inquiry but refused to discuss the investigation or grand jury action.

Several former White House colleagues of Mr. Stephens are listed among clients of the homosexual prostitution ring, according to the credit-card records, and those persons have confirmed that the charges were theirs.

Mr. Stephens' office, after first saying it would cooperate with The Times' inquiry, withdrew the offer late yesterday and also declined to say whether Mr. Stephens would recuse himself from the case because of possible conflict of interest.

At least one highly placed Bush administration official and a wealthy businessman who procured homosexual prostitutes from the escort services operated by the ring are cooperating with the investigation, several sources said.

Among clients who charged homosexual prostitute services on major credit cards over the past 18 months are Charles K. Dutcher, former associate director of presidential personnel in the Reagan administration, and Paul R. Balach, Labor Secretary Elizabeth Dole's political personnel liaison to the White House.

In the 1970s, Mr. Dutcher was a congressional aide to former Rep. Robert Bauman, Maryland Republican, who resigned from the House after he admitted having engaged in sexual liaisons with teen-age male

see PROBE, *page A7*

Source: https://twitter.com/piscisaureus/status/776709749584826369

Blocking syscalls – what’s the problem?

- OS suspends thread until result of operation is available
- To do *n* blocking calls at the same time you need *n* threads
- The longer a call blocks the more threads you need to serve more requests → Network calls are slow

…but synchronous calls are natural and easy! :-/

Example – Little’s law

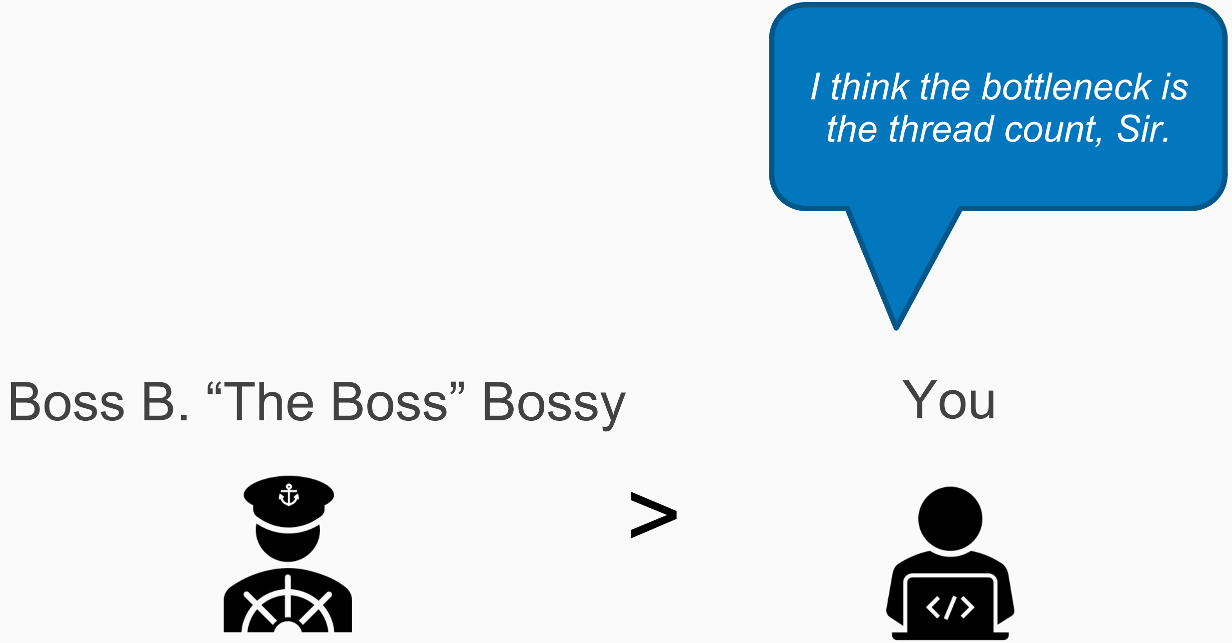
$$L = \lambda W.$$

Avg. # of customers in system = arrival rate * average time in system

For our case:
Avg. # threads needed = requests rate * response time of external system

e.g.:
100 requests/s, 10s response time from CBS
⇒ 1000 threads on avg. needed
⇒ 1000 threads * ~1MB = 1000 MB memory

Super Duper Bank



Boss B. “The Boss” Bossy

>

You

Super Duper Bank

We cannot accept this!
How can we fully utilize our machine?

Boss B. “The Boss” Bossy

>

You

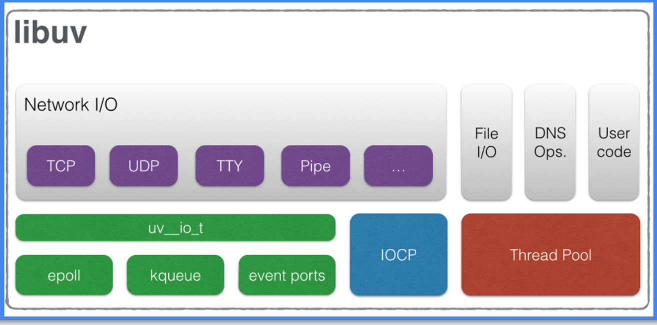
Chapter 2

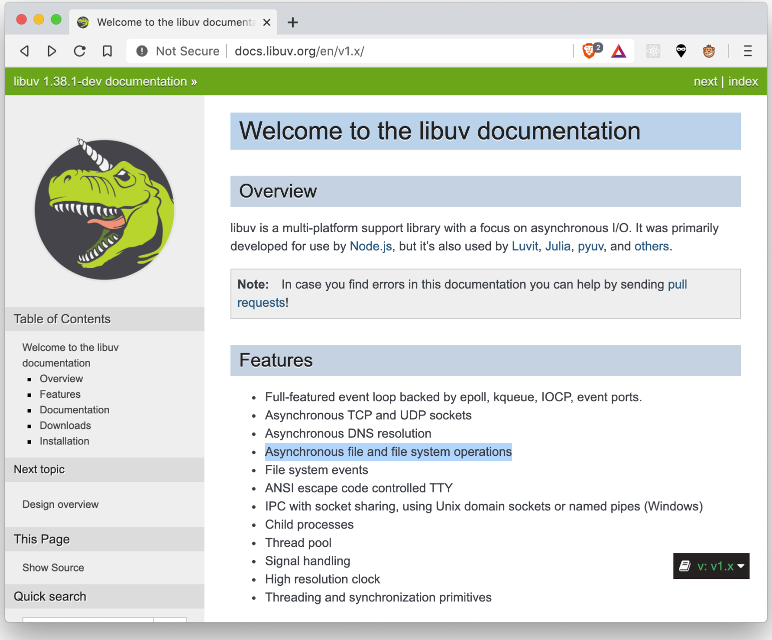
Let's fully utilize our machine

Non-blocking syscalls as a solution for the thread bottleneck problem

- Non-blocking syscalls do not suspend your thread → handle more than one primitive per thread
- Different styles for non-blocking IO
    - Polling, Multiplexed Block, …
- epoll (Linux), kqueue (Mac), IOCP (Windows) popular APIs for non-blocking networking – but all with different semantics
- libuv (Node.JS), mio (Tokio, Rust), Java NIO/Netty for Java: provide OS independent abstractions for non-blocking IO

Asynchronous != non-blocking





libuv 1.38.1-dev documentation » next | index

Welcome to the libuv documentation

Overview

libuv is a multi-platform support library with a focus on asynchronous I/O. It was primarily developed for use by Node.js, but it's also used by Luvit, Julia, pyuv, and others.

Note: In case you find errors in this documentation you can help by sending pull requests!

Features

- Full-featured event loop backed by epoll, kqueue, IOCP, event ports.
- Asynchronous TCP and UDP sockets
- Asynchronous DNS resolution
- Asynchronous file and file system operations
- File system events
- ANSI escape code controlled TTY
- IPC with socket sharing, using Unix domain sockets or named pipes (Windows)
- Child processes
- Thread pool
- Signal handling
- High resolution clock
- Threading and synchronization primitives

Table of Contents

Welcome to the libuv documentation
- Overview
- Features
- Documentation
- Downloads
- Installation

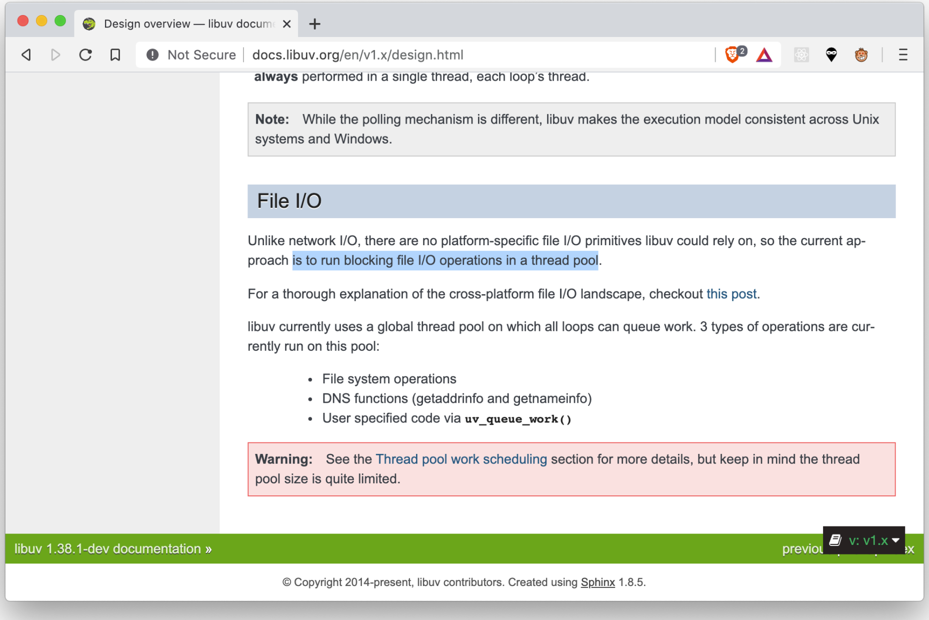
Next topic

Design overview

This Page

Show Source

Quick search


Design overview — libuv docum
Not Secure
docs.libuv.org/en/v1.x/design.html
v: v1.x

always performed in a single thread, each loop's thread.

Note: While the polling mechanism is different, libuv makes the execution model consistent across Unix systems and Windows.

File I/O

Unlike network I/O, there are no platform-specific file I/O primitives libuv could rely on, so the current approach is to run blocking file I/O operations in a thread pool.

For a thorough explanation of the cross-platform file I/O landscape, checkout this post.

libuv currently uses a global thread pool on which all loops can queue work. 3 types of operations are currently run on this pool:

- File system operations
- DNS functions (getaddrinfo and getnameinfo)
- User specified code via `uv_queue_work()`

Warning: See the Thread pool work scheduling section for more details, but keep in mind the thread pool size is quite limited.

libuv 1.38.1-dev documentation » previou

© Copyright 2014-present, libuv contributors. Created using Sphinx 1.8.5.

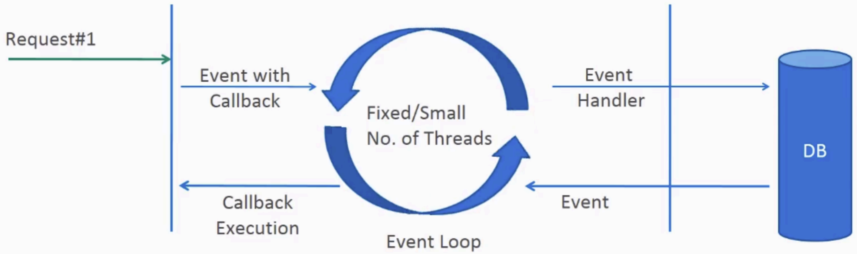
Event based execution model

- Relies on async base – “Don’t block the event loop”

e.g. Node.JS, Eclipse Vert.x, Project Reactor/Spring WebFlux

```
def eventloop_main():
  forever:
    e = wait for next event
    if there is a callback associated with e in our list:
      call the callback
```

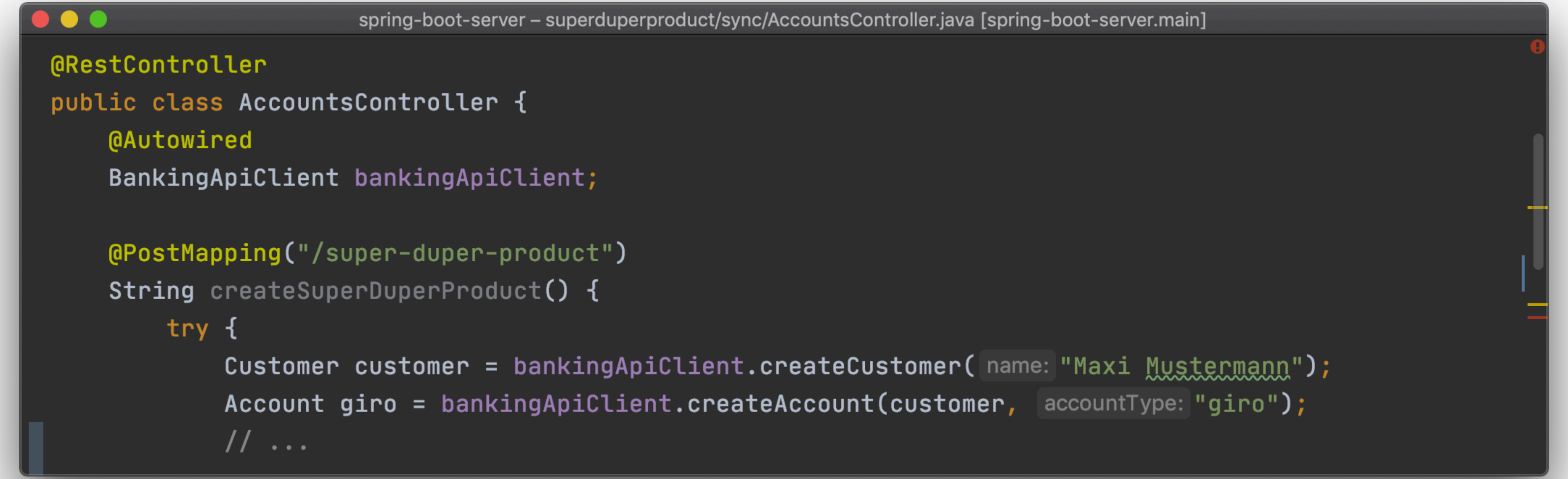
```
def read_from_socket_async(socket s, callback):
  tell OS we are interested in events from socket s
  save callback in our list
```



Source: https://howtodoinjava.com/spring-webflux/spring-webflux-tutorial/

How do we handle the asynchronous operations?

Recap: Synchronous style:

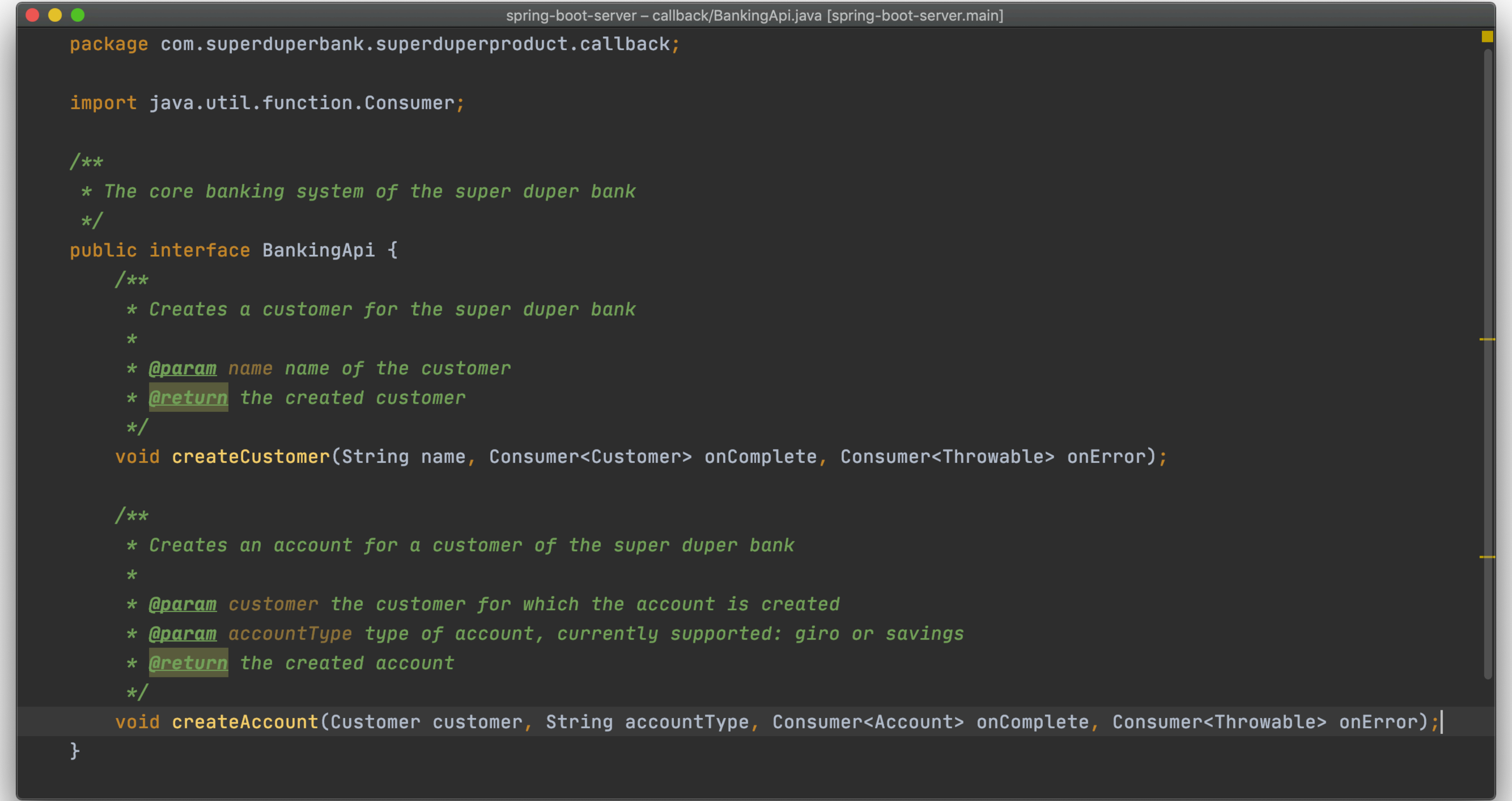


spring-boot-server – superduperproduct/sync/AccountsController.java [spring-boot-server.main]

```java
@RestController
public class AccountsController {
    @Autowired
    BankingApiClient bankingApiClient;

    @PostMapping("/super-duper-product")
    String createSuperDuperProduct() {
        try {
            Customer customer = bankingApiClient.createCustomer( name: "Maxi Mustermann");
            Account giro = bankingApiClient.createAccount(customer,  accountType: "giro");
            // ...
```

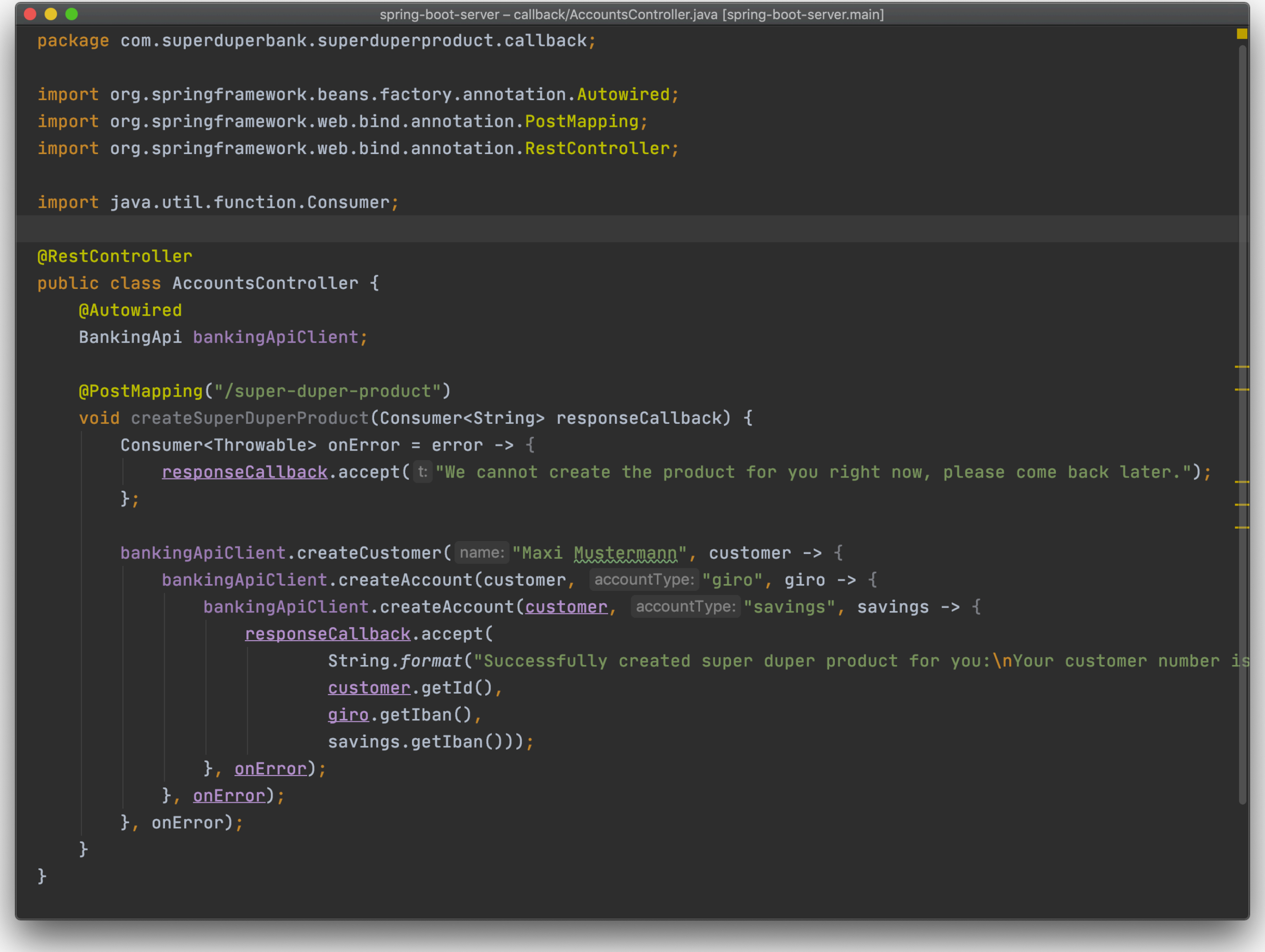
The callback

- Idea: For every asynchronous operation, pass a function which is called when the operation is complete
- Functions as “first class object”, in Java: Function object
- Hollywood principle: ”Don’t call us, we’ll call you”
- Hard to compose → callback hell



spring-boot-server – callback/BankingApi.java [spring-boot-server.main]

```java
package com.superduperbank.superduperproduct.callback;

import java.util.function.Consumer;

/**
 * The core banking system of the super duper bank
 */
public interface BankingApi {
    /**
     * Creates a customer for the super duper bank
     *
     * @param name name of the customer
     * @return the created customer
     */
    void createCustomer(String name, Consumer<Customer> onComplete, Consumer<Throwable> onError);

    /**
     * Creates an account for a customer of the super duper bank
     *
     * @param customer the customer for which the account is created
     * @param accountType type of account, currently supported: giro or savings
     * @return the created account
     */
    void createAccount(Customer customer, String accountType, Consumer<Account> onComplete, Consumer<Throwable> onError);
}
```



spring-boot-server – callback/AccountsController.java [spring-boot-server.main]

```java
package com.superduperbank.superduperproduct.callback;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.function.Consumer;


@RestController
public class AccountsController {
    @Autowired
    BankingApi bankingApiClient;

    @PostMapping("/super-duper-product")
    void createSuperDuperProduct(Consumer<String> responseCallback) {
        Consumer<Throwable> onError = error -> {
            responseCallback.accept( t: "We cannot create the product for you right now, please come back later.");
        };

        bankingApiClient.createCustomer( name: "Maxi Mustermann", customer -> {
            bankingApiClient.createAccount(customer,  accountType: "giro", giro -> {
                bankingApiClient.createAccount(customer,  accountType: "savings", savings -> {
                    responseCallback.accept(
                            String.format("Successfully created super duper product for you:\nYour customer number is
                            customer.getId(),
                            giro.getIban(),
                            savings.getIban()));
                }, onError);
            }, onError);
        }, onError);
    }
}
```

The Future[1] abstraction

- Explicit abstraction for an asynchronous operation
- Future represents the result of an asynchronous computation (which may not yet be completed) and can have three states: *Pending, Error, Done*
- Better composability than callbacks
- Semantic superset of Future: Reactive extensions

[1] called *Promise* in JavaScript

Super Duper Bank

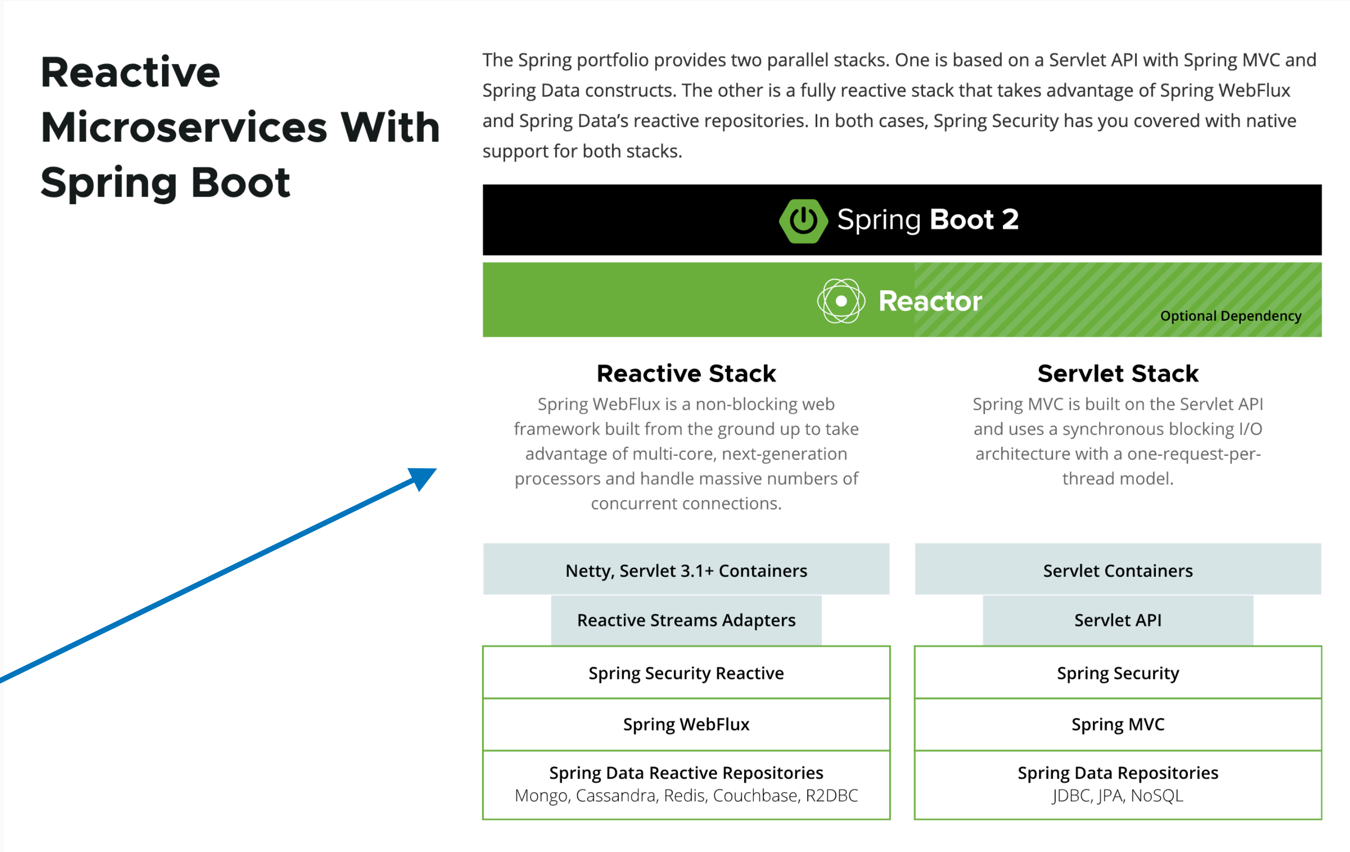
Make my service more scalable!!!!!!!
(until tomorrow)

B. Boss Bossy

>

Let’s use non-blocking I/O, Sir!

You



Reactive Microservices With Spring Boot

The Spring portfolio provides two parallel stacks. One is based on a Servlet API with Spring MVC and Spring Data constructs. The other is a fully reactive stack that takes advantage of Spring WebFlux and Spring Data's reactive repositories. In both cases, Spring Security has you covered with native support for both stacks.

Spring **Boot 2**

Reactor

Optional Dependency

Reactive Stack

Spring WebFlux is a non-blocking web framework built from the ground up to take advantage of multi-core, next-generation processors and handle massive numbers of concurrent connections.

Netty, Servlet 3.1+ Containers

Reactive Streams Adapters

Spring Security Reactive

Spring WebFlux

Spring Data Reactive Repositories
Mongo, Cassandra, Redis, Couchbase, R2DBC

Servlet Stack

Spring MVC is built on the Servlet API and uses a synchronous blocking I/O architecture with a one-request-per-thread model.

Servlet Containers

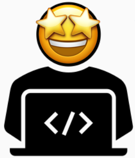
Servlet API

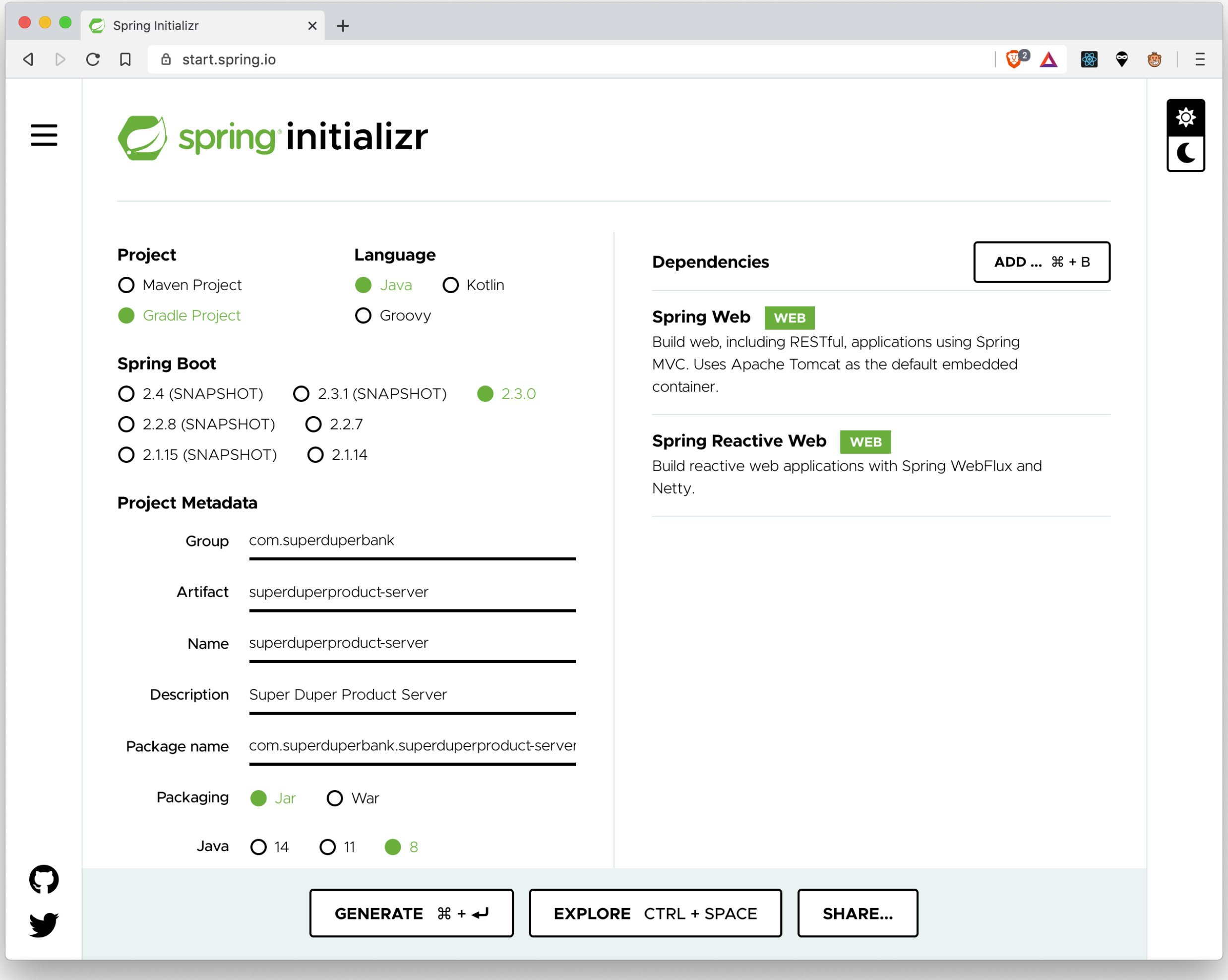
Spring Security

Spring MVC

Spring Data Repositories
JDBC, JPA, NoSQL

You





Project

○ Maven Project
● Gradle Project

Language

● Java ○ Kotlin
○ Groovy

Spring Boot

○ 2.4 (SNAPSHOT) ○ 2.3.1 (SNAPSHOT) ● 2.3.0
○ 2.2.8 (SNAPSHOT) ○ 2.2.7
○ 2.1.15 (SNAPSHOT) ○ 2.1.14

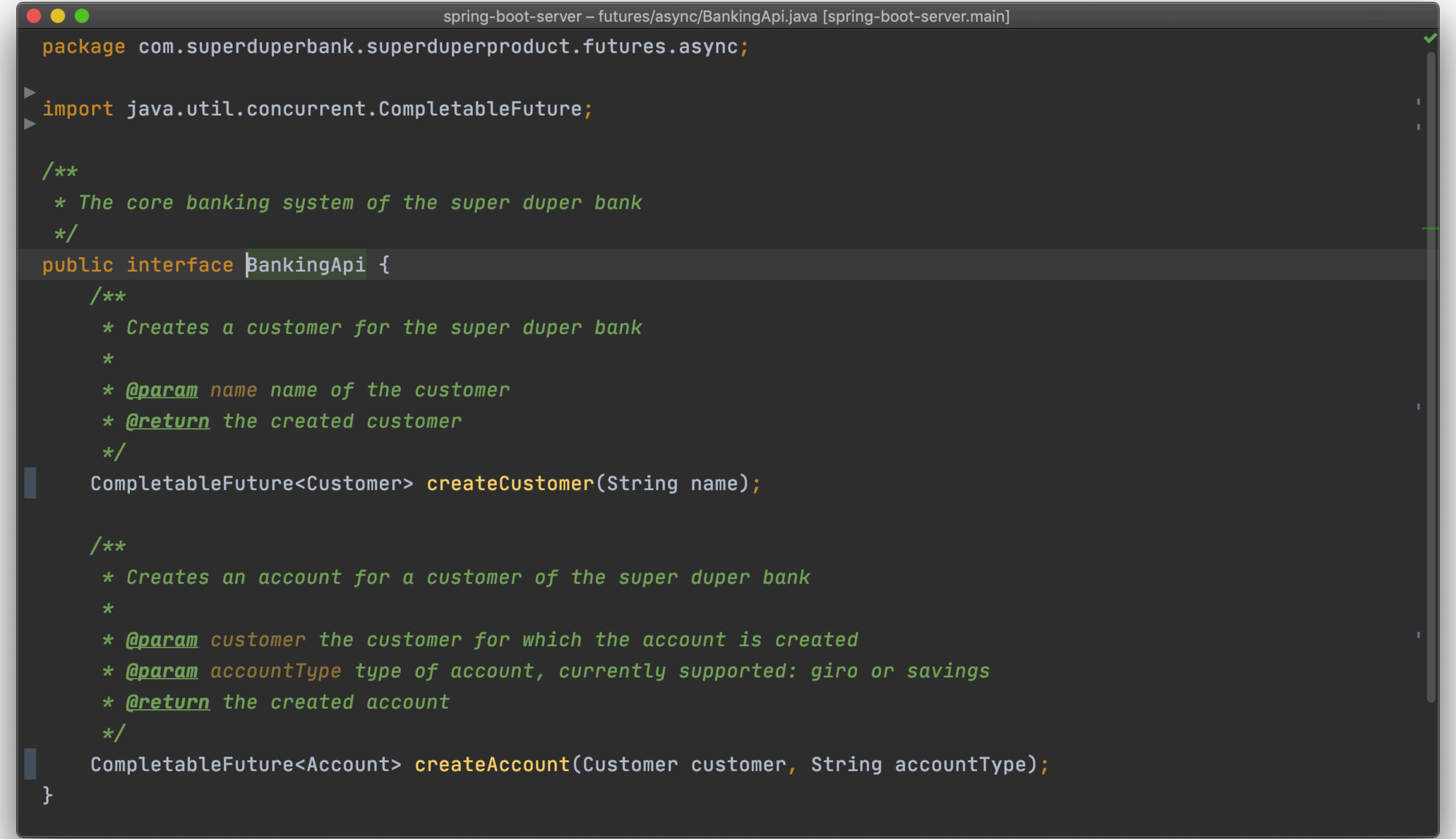
Project Metadata

Group	com.superduperbank
Artifact	superduperproduct-server
Name	superduperproduct-server
Description	Super Duper Product Server
Package name	com.superduperbank.superduperproduct-server
Packaging	● Jar ○ War
Java	○ 14 ○ 11 ● 8

Dependencies

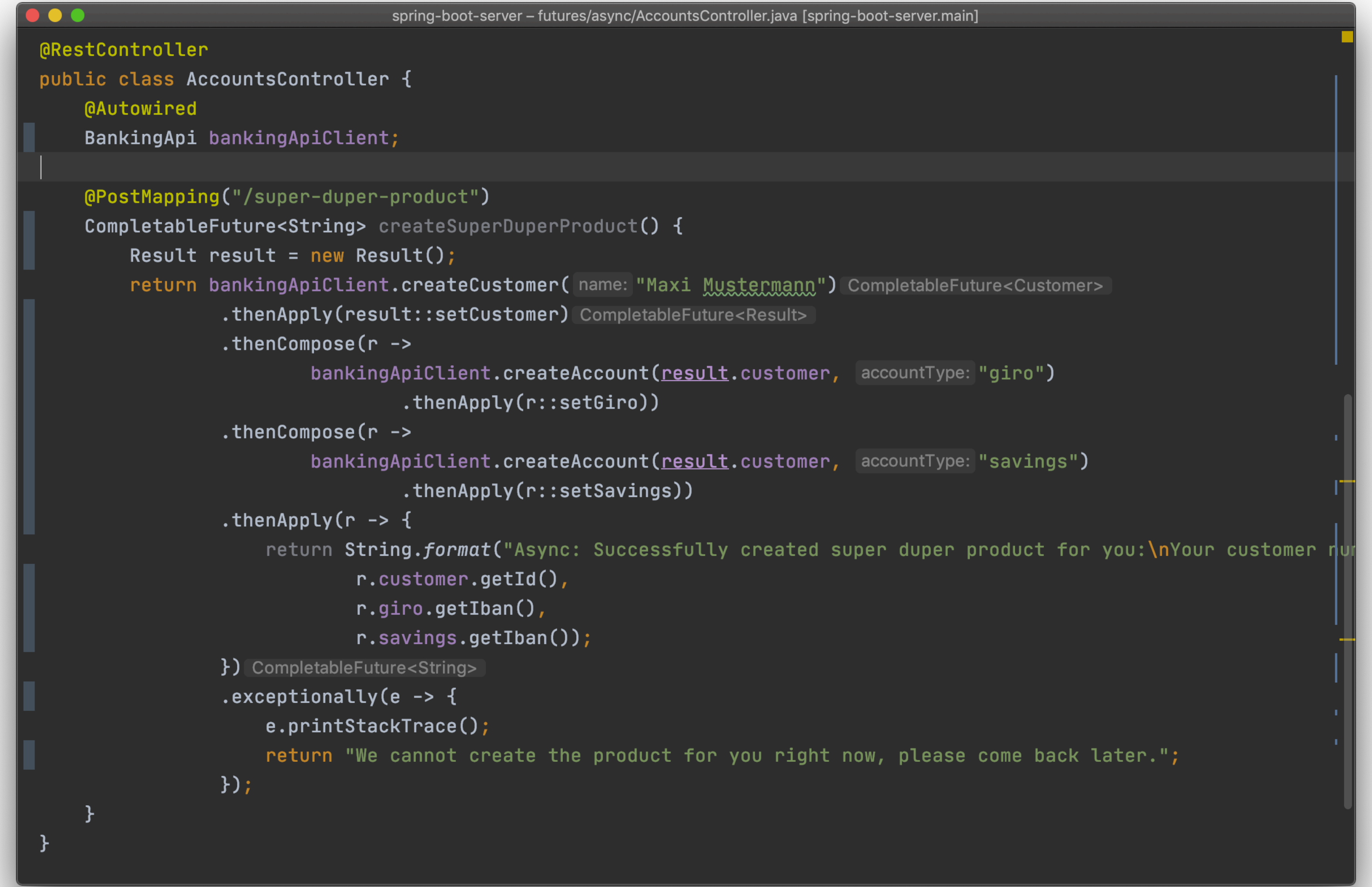
Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Reactive Web WEB
Build reactive web applications with Spring WebFlux and Netty.



spring-boot-server – futures/async/BankingApi.java [spring-boot-server.main]

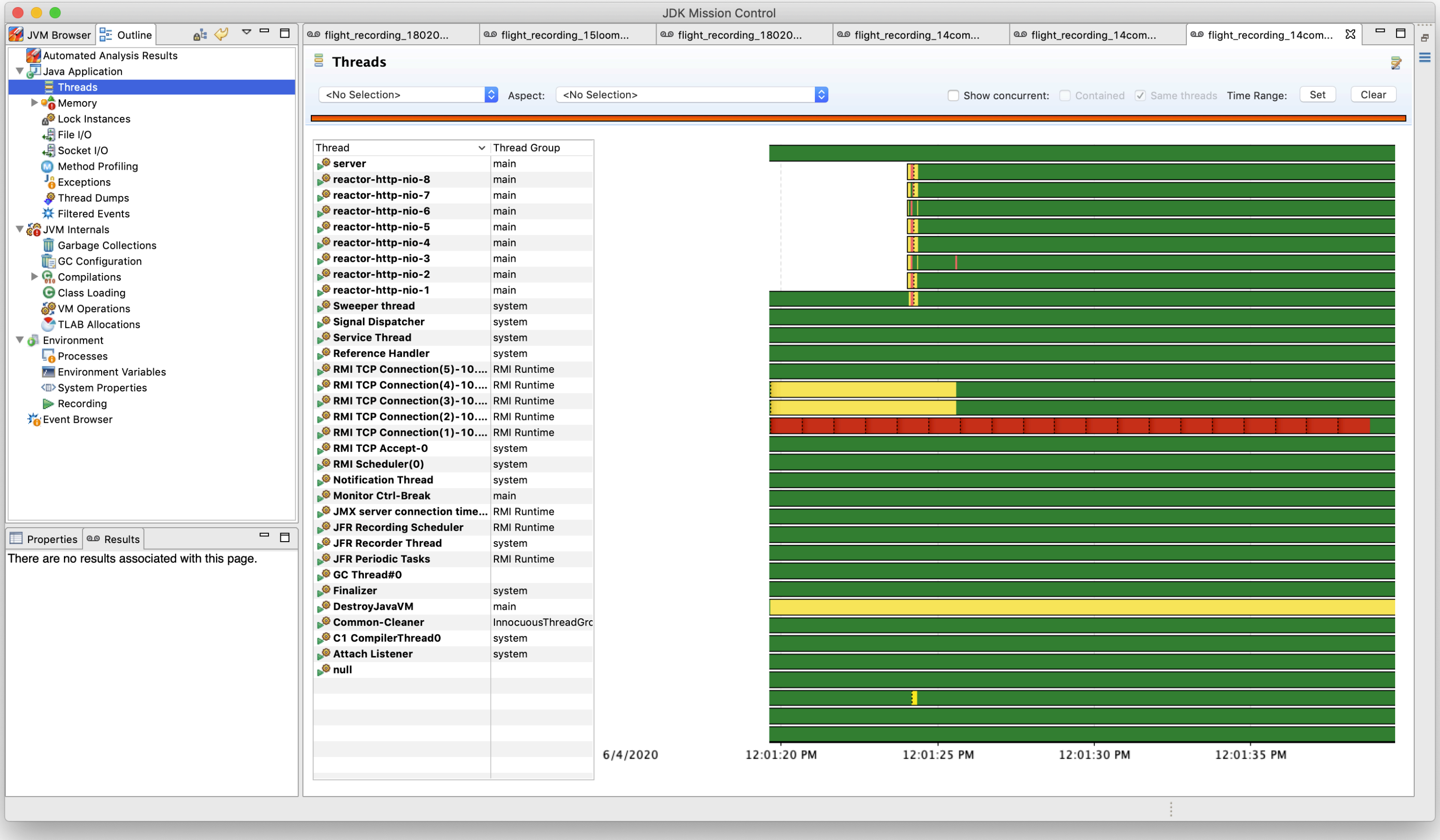
```java
package com.superduperbank.superduperproduct.futures.async;

import java.util.concurrent.CompletableFuture;

/**
 * The core banking system of the super duper bank
 */
public interface BankingApi {
    /**
     * Creates a customer for the super duper bank
     *
     * @param name name of the customer
     * @return the created customer
     */
    CompletableFuture<Customer> createCustomer(String name);

    /**
     * Creates an account for a customer of the super duper bank
     *
     * @param customer the customer for which the account is created
     * @param accountType type of account, currently supported: giro or savings
     * @return the created account
     */
    CompletableFuture<Account> createAccount(Customer customer, String accountType);
}
```



spring-boot-server – futures/async/AccountsController.java [spring-boot-server.main]

```java
@RestController
public class AccountsController {
    @Autowired
    BankingApi bankingApiClient;

    @PostMapping("/super-duper-product")
    CompletableFuture<String> createSuperDuperProduct() {
        Result result = new Result();
        return bankingApiClient.createCustomer( name: "Maxi Mustermann") CompletableFuture<Customer>
                .thenApply(result::setCustomer) CompletableFuture<Result>
                .thenCompose(r ->
                        bankingApiClient.createAccount(result.customer,  accountType: "giro")
                                .thenApply(r::setGiro))
                .thenCompose(r ->
                        bankingApiClient.createAccount(result.customer,  accountType: "savings")
                                .thenApply(r::setSavings))
                .thenApply(r -> {
                    return String.format("Async: Successfully created super duper product for you:\nYour customer nu
                            r.customer.getId(),
                            r.giro.getIban(),
                            r.savings.getIban());
                }) CompletableFuture<String>
                .exceptionally(e -> {
                    e.printStackTrace();
                    return "We cannot create the product for you right now, please come back later.";
                });
    }
}
```

Simulation of 100 requests in Spring WebFlux



Threads

<No Selection> Aspect: <No Selection> □ Show concurrent: □ Contained ☑ Same threads Time Range:

Thread	Thread Group
server	main
reactor-http-nio-8	main
reactor-http-nio-7	main
reactor-http-nio-6	main
reactor-http-nio-5	main
reactor-http-nio-4	main
reactor-http-nio-3	main
reactor-http-nio-2	main
reactor-http-nio-1	main
Sweeper thread	system
Signal Dispatcher	system
Service Thread	system
Reference Handler	system
RMI TCP Connection(5)-10....	RMI Runtime
RMI TCP Connection(4)-10....	RMI Runtime
RMI TCP Connection(3)-10....	RMI Runtime
RMI TCP Connection(2)-10....	RMI Runtime
RMI TCP Connection(1)-10....	RMI Runtime
RMI TCP Accept-0	system
RMI Scheduler(0)	system
Notification Thread	system
Monitor Ctrl-Break	main
JMX server connection time...	RMI Runtime
JFR Recording Scheduler	RMI Runtime
JFR Recorder Thread	system
JFR Periodic Tasks	RMI Runtime
GC Thread#0	
Finalizer	system
DestroyJavaVM	main
Common-Cleaner	InnocuousThreadGrc
C1 CompilerThread0	system
Attach Listener	system
null	

There are no results associated with this page.

Super Duper Bank

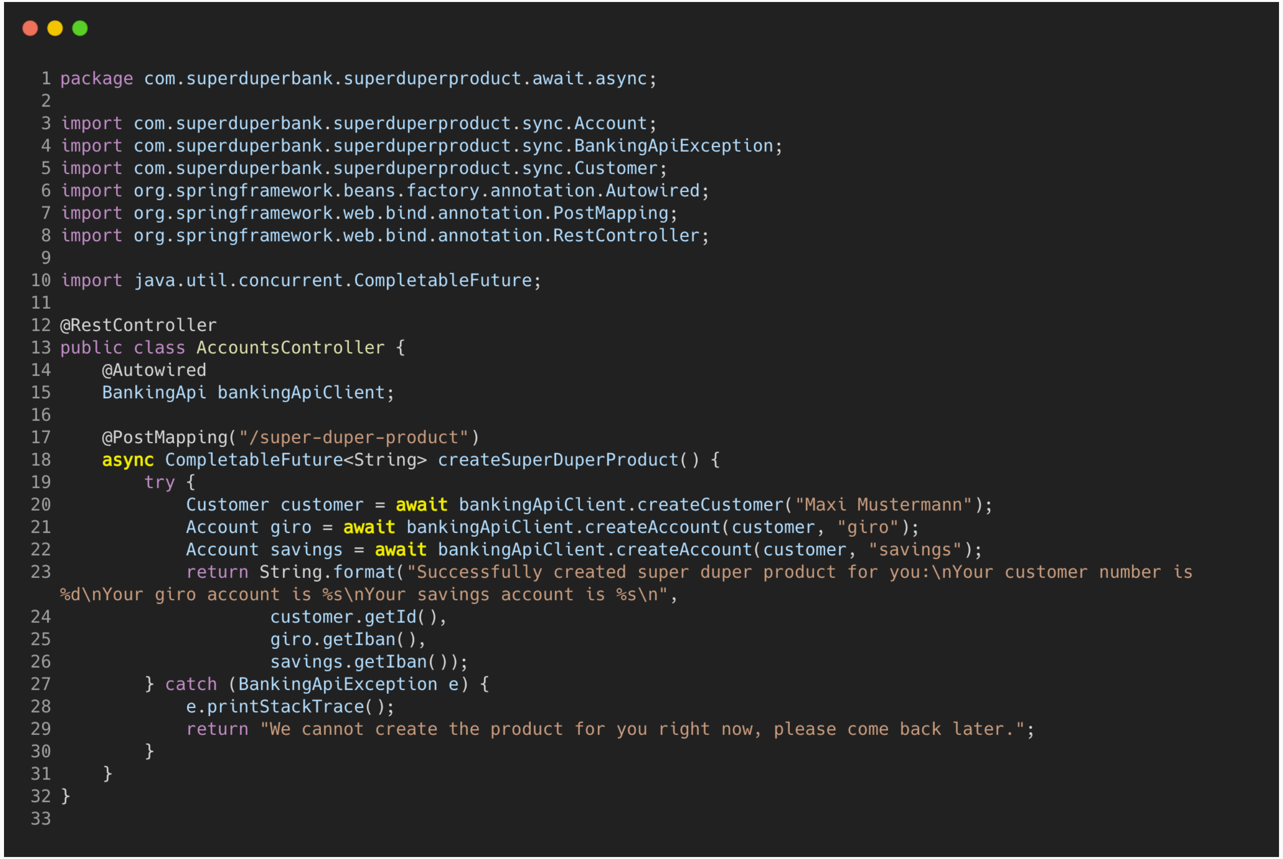
Not that easy…

You

async/await

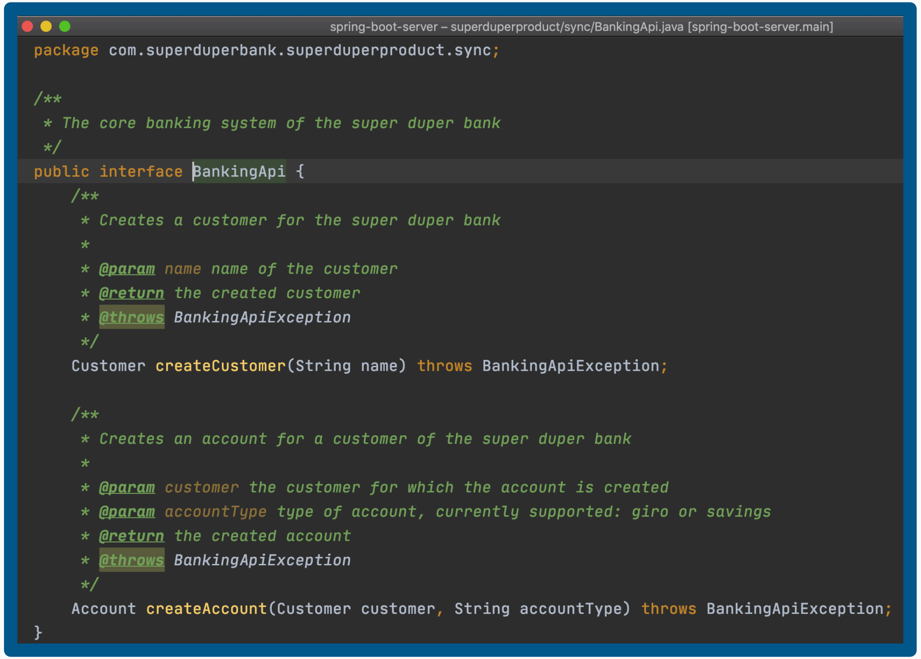
- “Syntactic sugar” for writing asynchronous functions that look like synchronous code
- Under the hood async/await syntax is converted to *Future/Promise* chains
- Still implicitly (or explicitly) return an asynchronous result
- Recently arrived in C#, Rust, JavaScript, Python, …

How async/await could look like in Java (hypothetical)

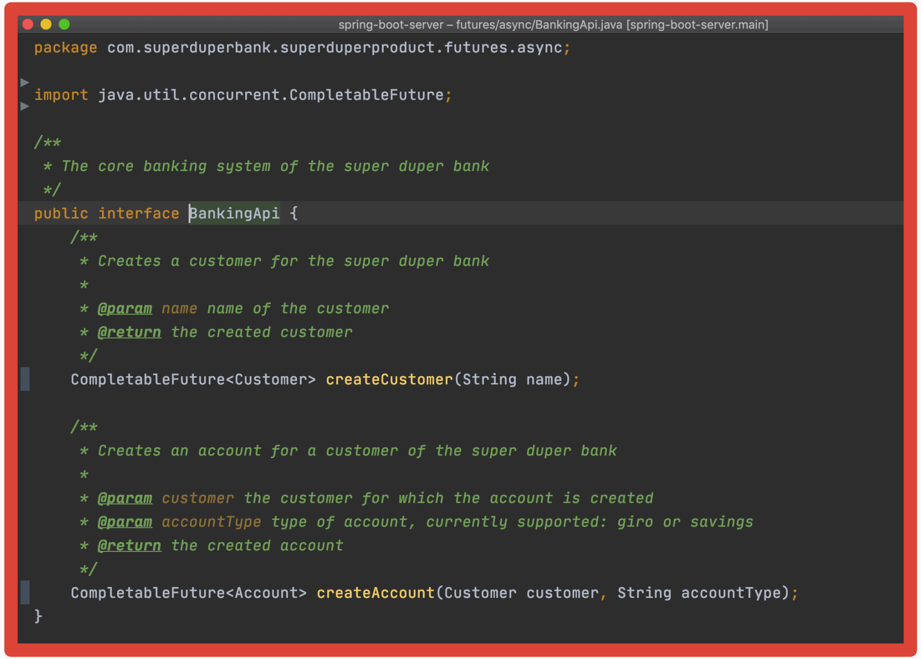


```java
package com.superduperbank.superduperproduct.await.async;

import com.superduperbank.superduperproduct.sync.Account;
import com.superduperbank.superduperproduct.sync.BankingApiException;
import com.superduperbank.superduperproduct.sync.Customer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.concurrent.CompletableFuture;

@RestController
public class AccountsController {
    @Autowired
    BankingApi bankingApiClient;

    @PostMapping("/super-duper-product")
    async CompletableFuture<String> createSuperDuperProduct() {
        try {
            Customer customer = await bankingApiClient.createCustomer("Maxi Mustermann");
            Account giro = await bankingApiClient.createAccount(customer, "giro");
            Account savings = await bankingApiClient.createAccount(customer, "savings");
            return String.format("Successfully created super duper product for you:\nYour customer number is %d\nYour giro account is %s\nYour savings account is %s\n",
                    customer.getId(),
                    giro.getIban(),
                    savings.getIban());
        } catch (BankingApiException e) {
            e.printStackTrace();
            return "We cannot create the product for you right now, please come back later.";
        }
    }
}

```

About blue and red worlds


spring-boot-server – superduperproduct/sync/BankingApi.java [spring-boot-server.main]

```java
package com.superduperbank.superduperproduct.sync;

/**
 * The core banking system of the super duper bank
 */
public interface BankingApi {
    /**
     * Creates a customer for the super duper bank
     *
     * @param name name of the customer
     * @return the created customer
     * @throws BankingApiException
     */
    Customer createCustomer(String name) throws BankingApiException;

    /**
     * Creates an account for a customer of the super duper bank
     *
     * @param customer the customer for which the account is created
     * @param accountType type of account, currently supported: giro or savings
     * @return the created account
     * @throws BankingApiException
     */
    Account createAccount(Customer customer, String accountType) throws BankingApiException;
}
```

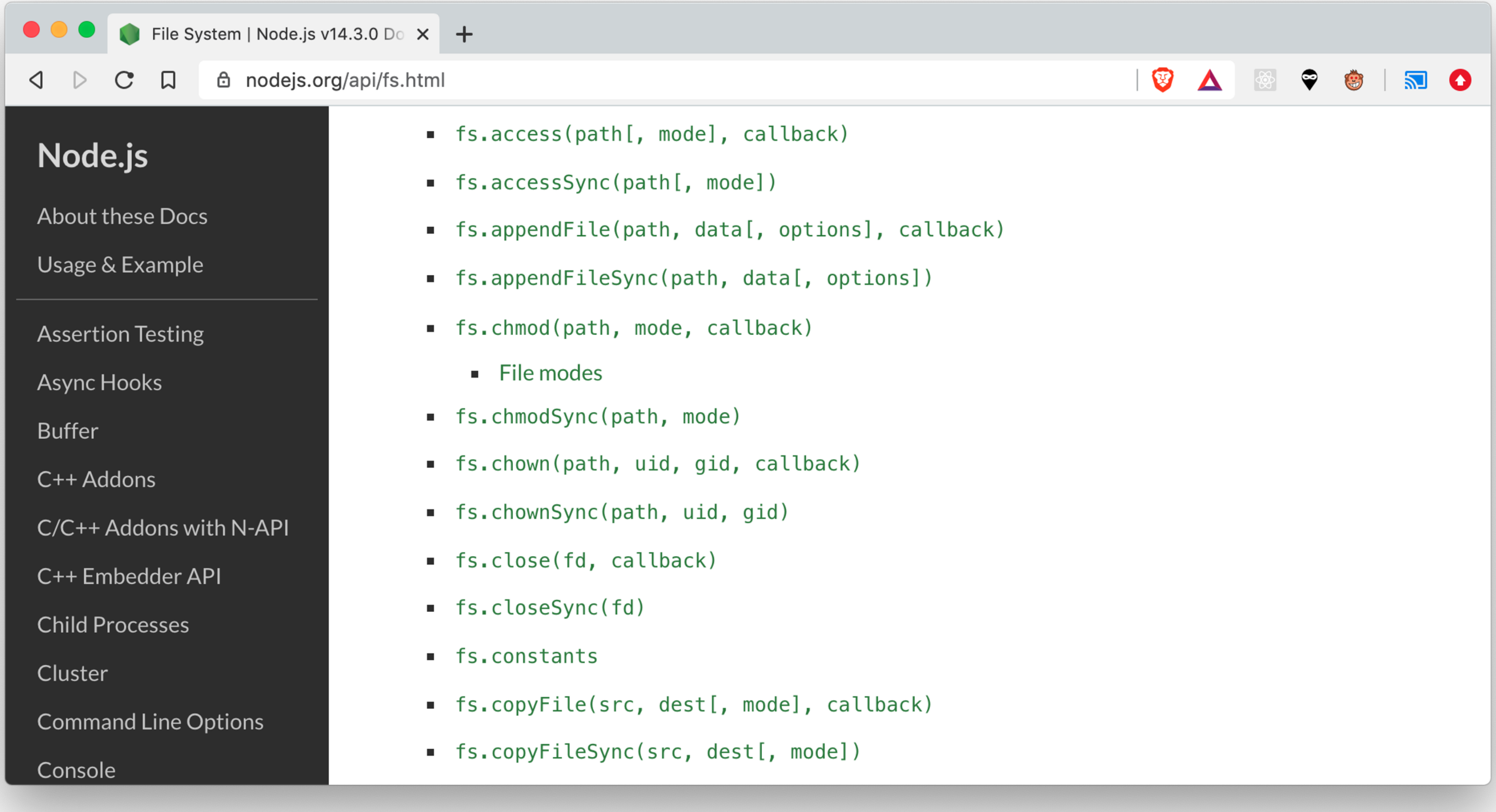

spring-boot-server – futures/async/BankingApi.java [spring-boot-server.main]

```java
package com.superduperbank.superduperproduct.futures.async;

import java.util.concurrent.CompletableFuture;

/**
 * The core banking system of the super duper bank
 */
public interface BankingApi {
    /**
     * Creates a customer for the super duper bank
     *
     * @param name name of the customer
     * @return the created customer
     */
    CompletableFuture<Customer> createCustomer(String name);

    /**
     * Creates an account for a customer of the super duper bank
     *
     * @param customer the customer for which the account is created
     * @param accountType type of account, currently supported: giro or savings
     * @return the created account
     */
    CompletableFuture<Account> createAccount(Customer customer, String accountType);
}
```

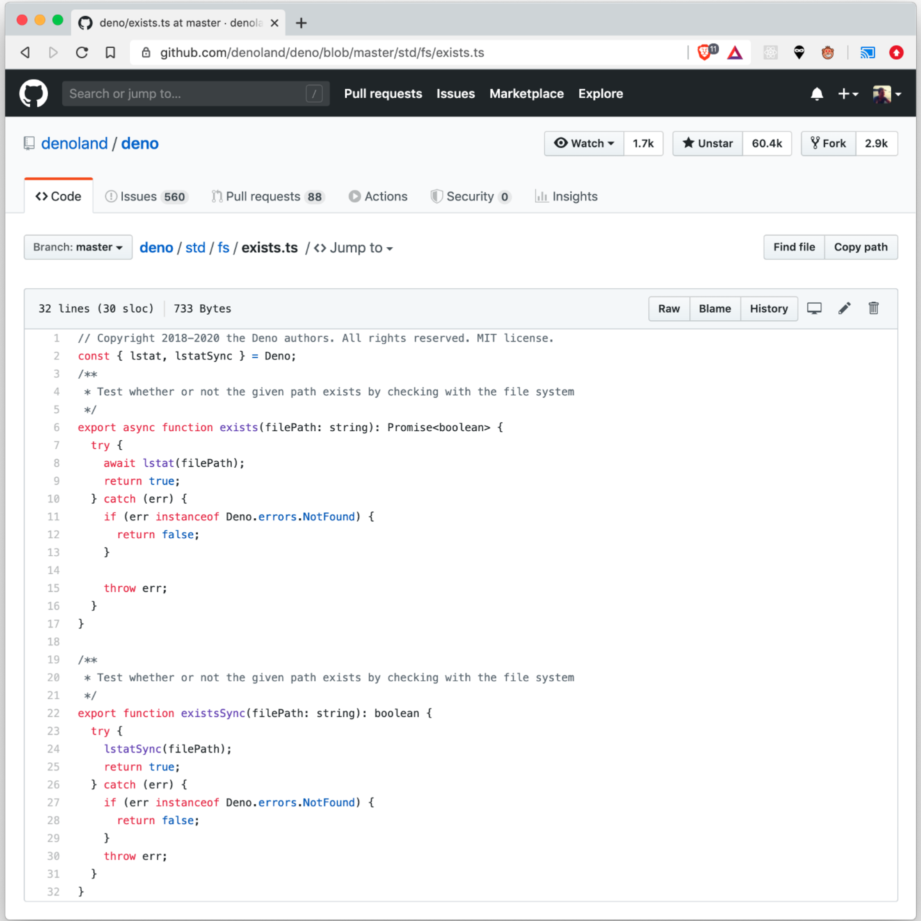
About blue and red worlds

- Going into asynchronous world break your old interfaces and you have to decide beforehand which world you want
- Hard to go from synchronous world to asynchronous world
- Often, we anyway just want a synchronous programming model but are forced to use asynchronous abstractions because of the underlying execution model
- *async/await* can make it look like synchronous, but we are still in the asynchronous world

Source: http://journal.stuffwithstuff.com/2015/02/01/what-color-is-your-function/



- `fs.access(path[, mode], callback)`
- `fs.accessSync(path[, mode])`
- `fs.appendFile(path, data[, options], callback)`
- `fs.appendFileSync(path, data[, options])`
- `fs.chmod(path, mode, callback)`
  - File modes
- `fs.chmodSync(path, mode)`
- `fs.chown(path, uid, gid, callback)`
- `fs.chownSync(path, uid, gid)`
- `fs.close(fd, callback)`
- `fs.closeSync(fd)`
- `fs.constants`
- `fs.copyFile(src, dest[, mode], callback)`
- `fs.copyFileSync(src, dest[, mode])`

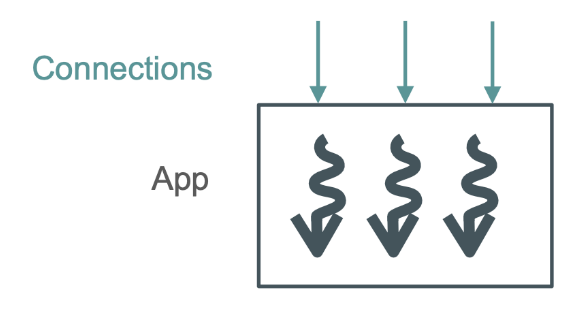


```
// Copyright 2018-2020 the Deno authors. All rights reserved. MIT license.
const { lstat, lstatSync } = Deno;
/**
 * Test whether or not the given path exists by checking with the file system
 */
export async function exists(filePath: string): Promise<boolean> {
  try {
    await lstat(filePath);
    return true;
  } catch (err) {
    if (err instanceof Deno.errors.NotFound) {
      return false;
    }

    throw err;
  }
}

/**
 * Test whether or not the given path exists by checking with the file system
 */
export function existsSync(filePath: string): boolean {
  try {
    lstatSync(filePath);
    return true;
  } catch (err) {
    if (err instanceof Deno.errors.NotFound) {
      return false;
    }
    throw err;
  }
}
```

Chapter 3

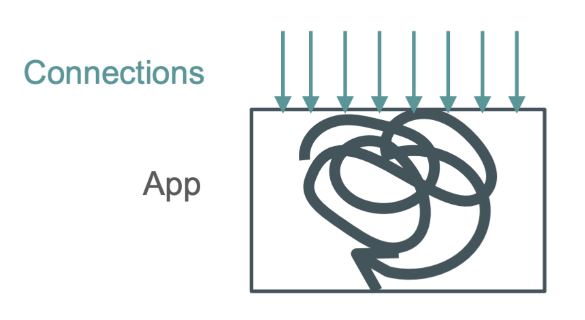
I want my blocking code back :-/

Choose between:



Synchronous style

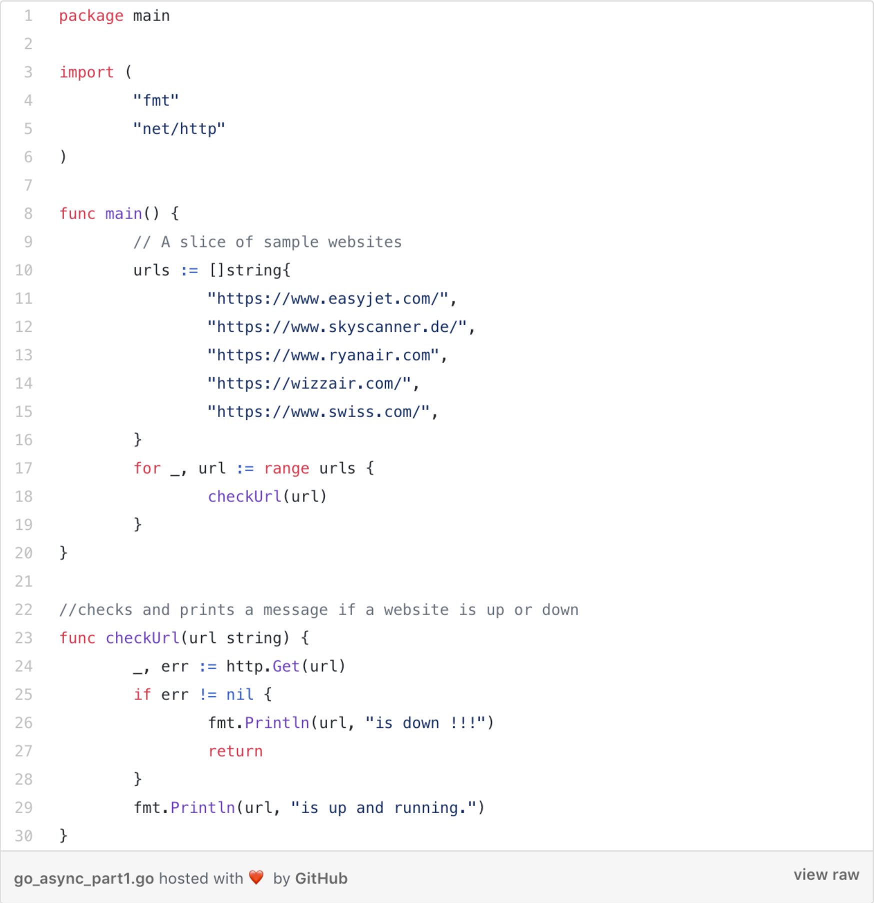
- 😃 Simple
- 🙂 Language integration (Exceptions, control flow)
- 😏 Not very efficient (OS Thread per request -> limited resource)
- 😣 Advanced stuff is more complex (e.g. do two things in parallel)



Asynchronous style

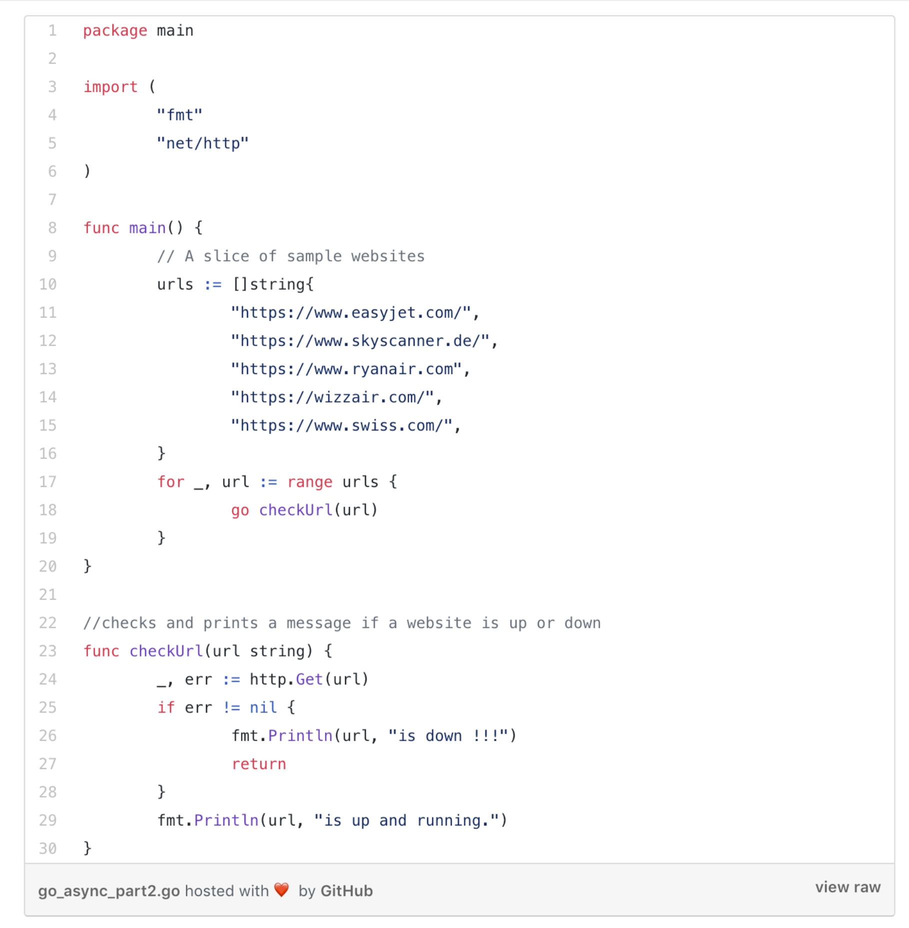
- 😣 Hard to read (without *async/await*), complex, hard to debug
- 😣 Blue and red worlds, virality
- 😣 Rewrite your Application
- 😃 Efficient

Is my website up in go? – Synchronous



```go
package main

import (
	"fmt"
	"net/http"
)

func main() {
	// A slice of sample websites
	urls := []string{
		"https://www.easyjet.com/",
		"https://www.skyscanner.de/",
		"https://www.ryanair.com",
		"https://wizzair.com/",
		"https://www.swiss.com/",
	}
	for _, url := range urls {
		checkUrl(url)
	}
}

//checks and prints a message if a website is up or down
func checkUrl(url string) {
	_, err := http.Get(url)
	if err != nil {
		fmt.Println(url, "is down !!!")
		return
	}
	fmt.Println(url, "is up and running.")
}
```

Is my website up in go? – Asynchronous

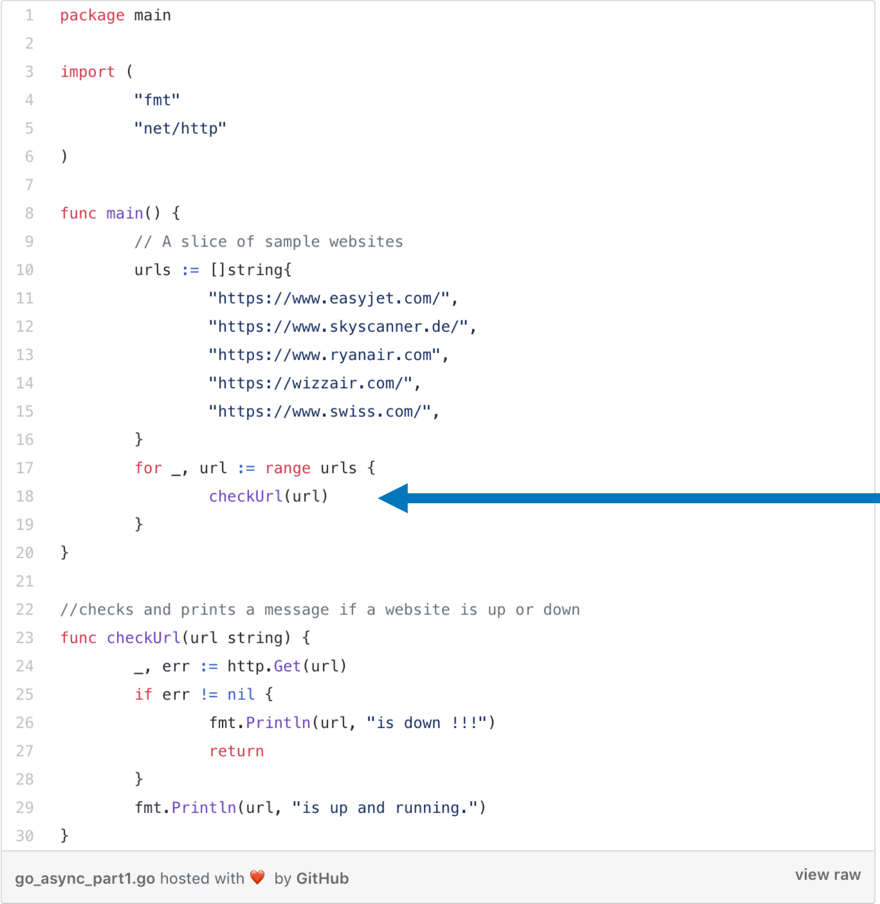


```go
package main

import (
	"fmt"
	"net/http"
)

func main() {
	// A slice of sample websites
	urls := []string{
		"https://www.easyjet.com/",
		"https://www.skyscanner.de/",
		"https://www.ryanair.com",
		"https://wizzair.com/",
		"https://www.swiss.com/",
	}
	for _, url := range urls {
		go checkUrl(url)
	}
}

//checks and prints a message if a website is up or down
func checkUrl(url string) {
	_, err := http.Get(url)
	if err != nil {
		fmt.Println(url, "is down !!!")
		return
	}
	fmt.Println(url, "is up and running.")
}
```

go_async_part2.go hosted with ❤ by GitHub

view raw

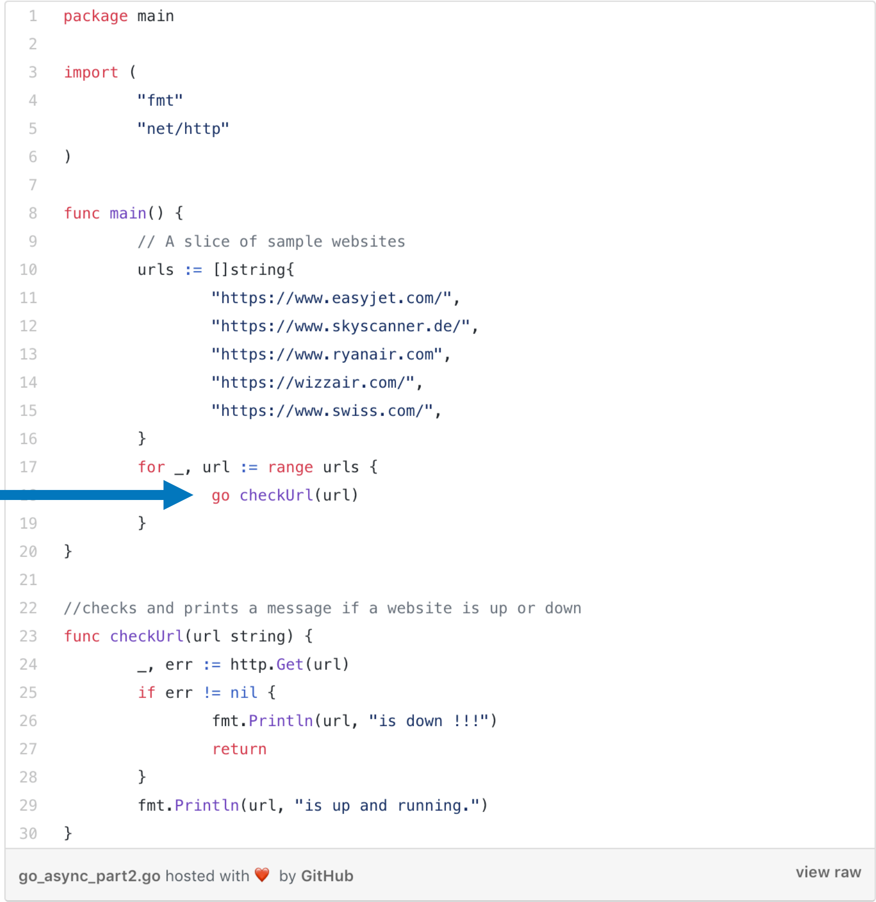
Is my website up in go?



```go
package main

import (
	"fmt"
	"net/http"
)

func main() {
	// A slice of sample websites
	urls := []string{
		"https://www.easyjet.com/",
		"https://www.skyscanner.de/",
		"https://www.ryanair.com",
		"https://wizzair.com/",
		"https://www.swiss.com/",
	}
	for _, url := range urls {
		checkUrl(url)
	}
}

//checks and prints a message if a website is up or down
func checkUrl(url string) {
	_, err := http.Get(url)
	if err != nil {
		fmt.Println(url, "is down !!!")
		return
	}
	fmt.Println(url, "is up and running.")
}
```

go_async_part1.go hosted with ❤ by GitHub

view raw

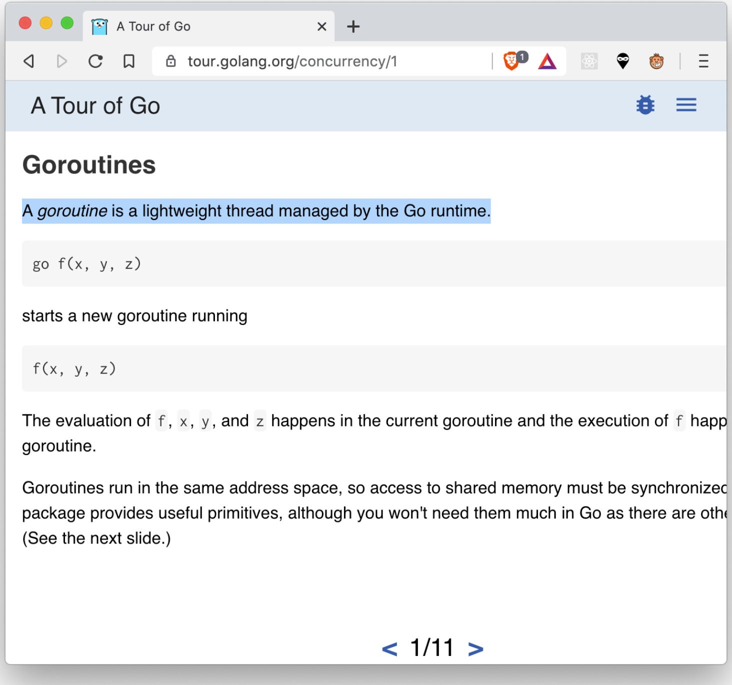


```go
package main

import (
	"fmt"
	"net/http"
)

func main() {
	// A slice of sample websites
	urls := []string{
		"https://www.easyjet.com/",
		"https://www.skyscanner.de/",
		"https://www.ryanair.com",
		"https://wizzair.com/",
		"https://www.swiss.com/",
	}
	for _, url := range urls {
		go checkUrl(url)
	}
}

//checks and prints a message if a website is up or down
func checkUrl(url string) {
	_, err := http.Get(url)
	if err != nil {
		fmt.Println(url, "is down !!!")
		return
	}
	fmt.Println(url, "is up and running.")
}
```

go_async_part2.go hosted with ❤ by GitHub

view raw

Is my website up in go?



Goroutines

A *goroutine* is a lightweight thread managed by the Go runtime.

```
go f(x, y, z)
```

starts a new goroutine running

```
f(x, y, z)
```

The evaluation of `f`, `x`, `y`, and `z` happens in the current goroutine and the execution of `f` happ
goroutine.

Goroutines run in the same address space, so access to shared memory must be synchronized
package provides useful primitives, although you won't need them much in Go as there are othe
(See the next slide.)

< 1/11 >

Virtual thread

aka lightweight thread

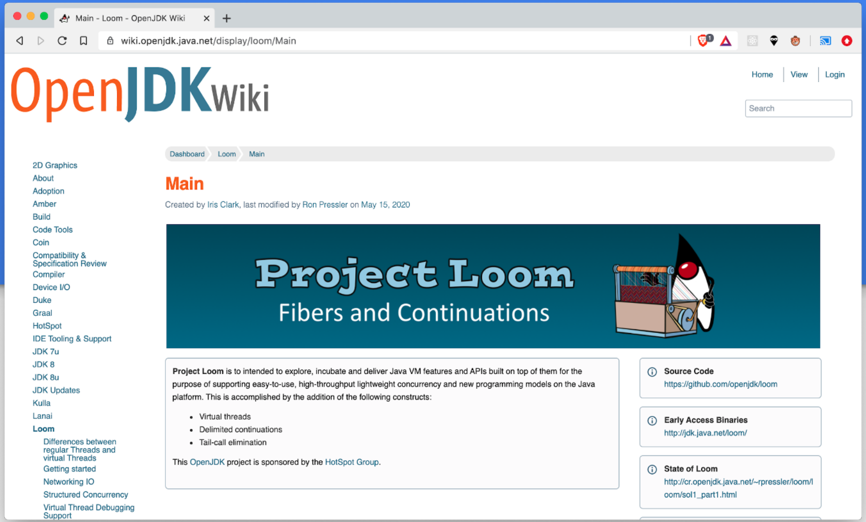
aka fiber

aka green thread

aka user-mode thread

Project Loom

- Official OpenJDK Project to implement virtual threads on the Java platform (JVM)
- Currently in development
- Can be tried out by using a preview build JDK



Main

Created by Iris Clark, last modified by Ron Pressler on May 15, 2020

Project Loom is to intended to explore, incubate and deliver Java VM features and APIs built on top of them for the purpose of supporting easy-to-use, high-throughput lightweight concurrency and new programming models on the Java platform. This is accomplished by the addition of the following constructs:

- Virtual threads
- Delimited continuations
- Tail-call elimination

This OpenJDK project is sponsored by the HotSpot Group.

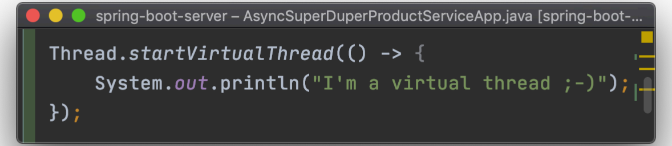
Source Code
https://github.com/openjdk/loom

Early Access Binaries
http://jdk.java.net/loom/

State of Loom
http://cr.openjdk.java.net/~rpressler/loom/loom/sol1_part1.html

What is a virtual thread in Java?

- Like OS threads but
  - Lightweight – have as many as you want
  - Fast – context switches are cheap
- Managed by the Java Runtime
- Use existing APIs (*Thread, Executors, …*)
- No timeslice-based preemption (by default)



spring-boot-server – AsyncSuperDuperProductServiceApp.java [spring-boot-...

```
Thread.startVirtualThread(() -> {
    System.out.println("I'm a virtual thread ;-)");
});
```

virtual thread =

Representation of the state of a computation

+

Something which can control the execution of the computation virtual thread =

Continuation

+

Scheduler

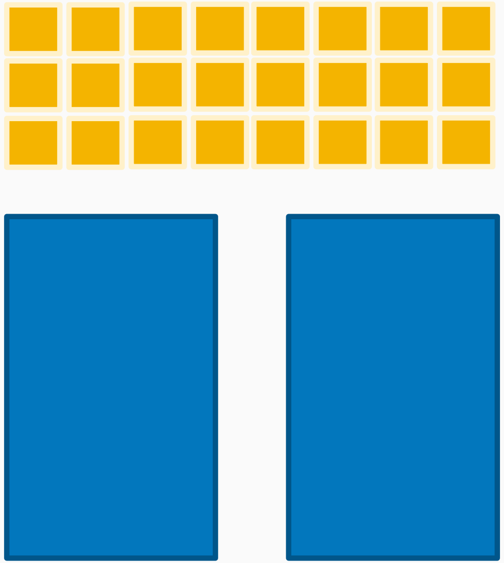
Continuation (*coroutine*)

- Piece of sequential code that can suspend itself and may be continued at a later point
- Low level API, not to be used directly

```
package java.lang;
public class Continuation implements Runnable {
    public Continuation(ContinuationScope scope, Runnable target);
    public final void run() ;
    public static void yield(ContinuationScope scope) ;
    public boolean isDone();
}
```

Scheduler

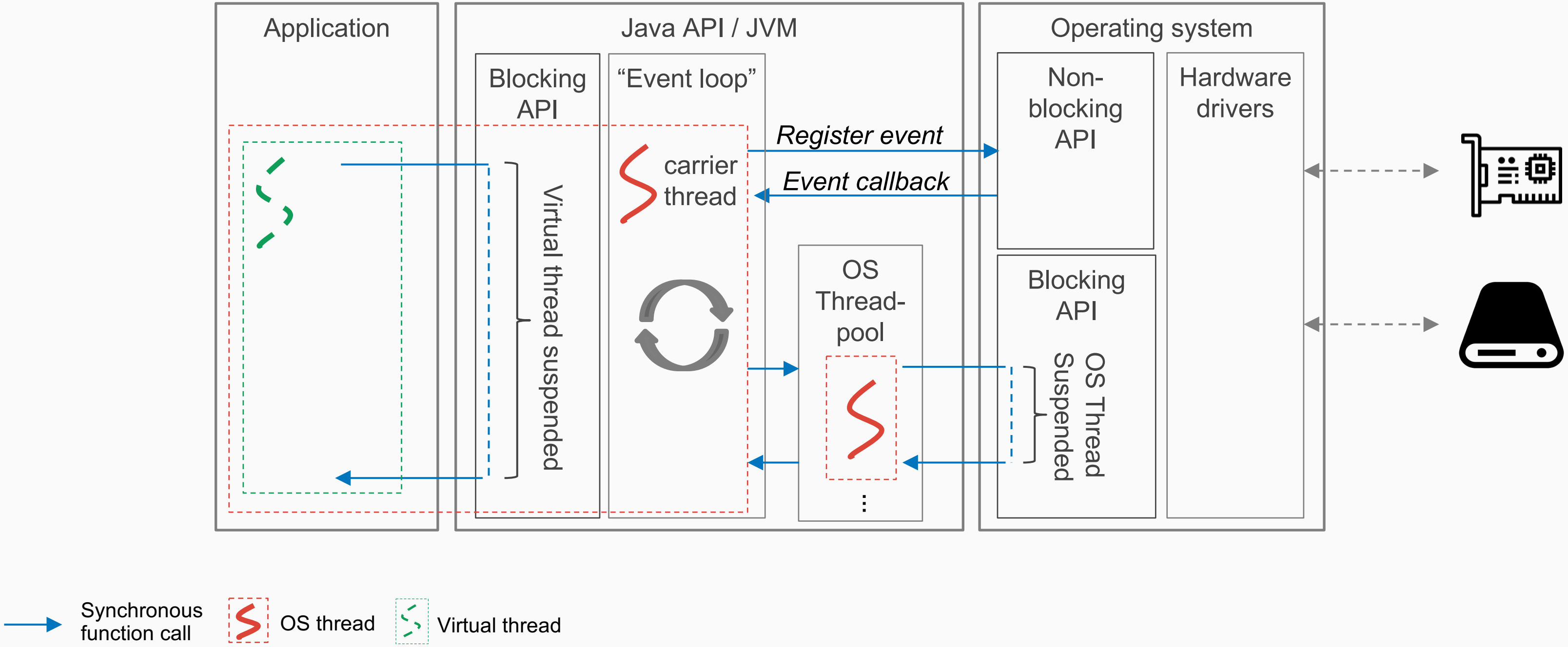
- Scheduler schedules the continuations onto real worker OS threads (carrier thread)
- By default *ForkJoinPool* Scheduler is used which distributes work among all CPU cores
- Possible to change scheduler (e.g. choose to have only one carrier thread -> Node.JS like)



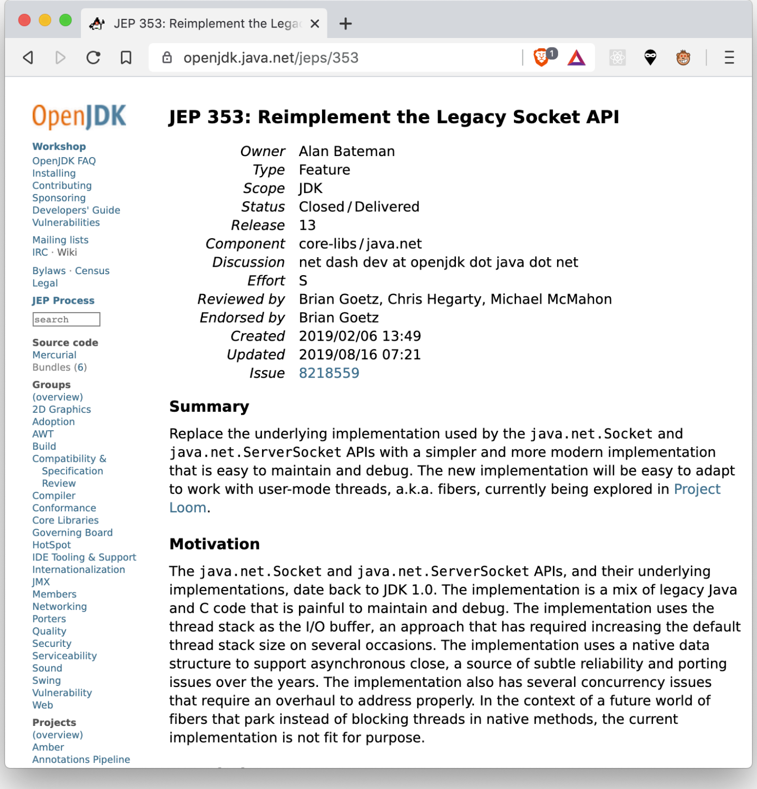
Why virtual threads instead of asynchronous abstractions?

- Enables non-blocking code to be (virtual-thread)-synchronous
    - Normal language constructs for conditional logic, error handling, …
    - Easy debugging
- No need to break your interfaces, no forced blue world for non-blocking IO
    - Libraries that use the JDK primitives will also automatically play well with virtual threads (e.g. Spring Web, JDBC, …)
    - Works with legacy code without changes (in the best case)
- For advanced stuff, e.g. do two things in parallel
    - → use asynchronous abstractions (Future, Reactive) or structured concurrency on the consumer side

Virtual threads allow to translate asynchronous to synchronous APIs



Example: New Socket API implementation ready for virtual threads



OpenJDK

Workshop
OpenJDK FAQ
Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities

Mailing lists
IRC · Wiki

Bylaws · Census
Legal

JEP Process

search

Source code
Mercurial
Bundles (6)

Groups
(overview)
2D Graphics
Adoption
AWT
Build
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support
Internationalization
JMX
Members
Networking
Porters
Quality
Security
Serviceability
Sound
Swing
Vulnerability
Web

Projects
(overview)
Amber
Annotations Pipeline

JEP 353: Reimplement the Legacy Socket API

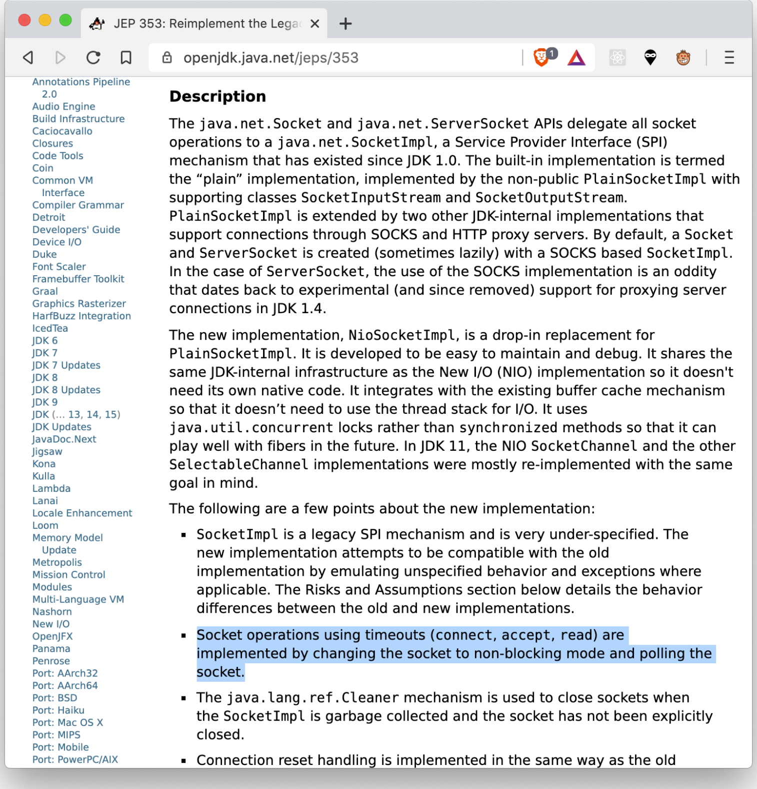
Owner	Alan Bateman
Type	Feature
Scope	JDK
Status	Closed / Delivered
Release	13
Component	core-libs / java.net
Discussion	net dash dev at openjdk dot java dot net
Effort	S
Reviewed by	Brian Goetz, Chris Hegarty, Michael McMahon
Endorsed by	Brian Goetz
Created	2019/02/06 13:49
Updated	2019/08/16 07:21
Issue	8218559

Summary

Replace the underlying implementation used by the `java.net.Socket` and `java.net.ServerSocket` APIs with a simpler and more modern implementation that is easy to maintain and debug. The new implementation will be easy to adapt to work with user-mode threads, a.k.a. fibers, currently being explored in Project Loom.

Motivation

The `java.net.Socket` and `java.net.ServerSocket` APIs, and their underlying implementations, date back to JDK 1.0. The implementation is a mix of legacy Java and C code that is painful to maintain and debug. The implementation uses the thread stack as the I/O buffer, an approach that has required increasing the default thread stack size on several occasions. The implementation uses a native data structure to support asynchronous close, a source of subtle reliability and porting issues over the years. The implementation also has several concurrency issues that require an overhaul to address properly. In the context of a future world of fibers that park instead of blocking threads in native methods, the current implementation is not fit for purpose.



Annotations Pipeline 2.0
Audio Engine
Build Infrastructure
Caciocavallo
Closures
Code Tools
Coin
Common VM Interface
Compiler Grammar
Detroit
Developers' Guide
Device I/O
Duke
Font Scaler
Framebuffer Toolkit
Graal
Graphics Rasterizer
HarfBuzz Integration
IcedTea
JDK 6
JDK 7
JDK 7 Updates
JDK 8
JDK 8 Updates
JDK 9
JDK (… 13, 14, 15)
JDK Updates
JavaDoc.Next
Jigsaw
Kona
Kulla
Lambda
Lanai
Locale Enhancement
Loom
Memory Model Update
Metropolis
Mission Control
Modules
Multi-Language VM
Nashorn
New I/O
OpenJFX
Panama
Penrose
Port: AArch32
Port: AArch64
Port: BSD
Port: Haiku
Port: Mac OS X
Port: MIPS
Port: Mobile
Port: PowerPC/AIX

Description

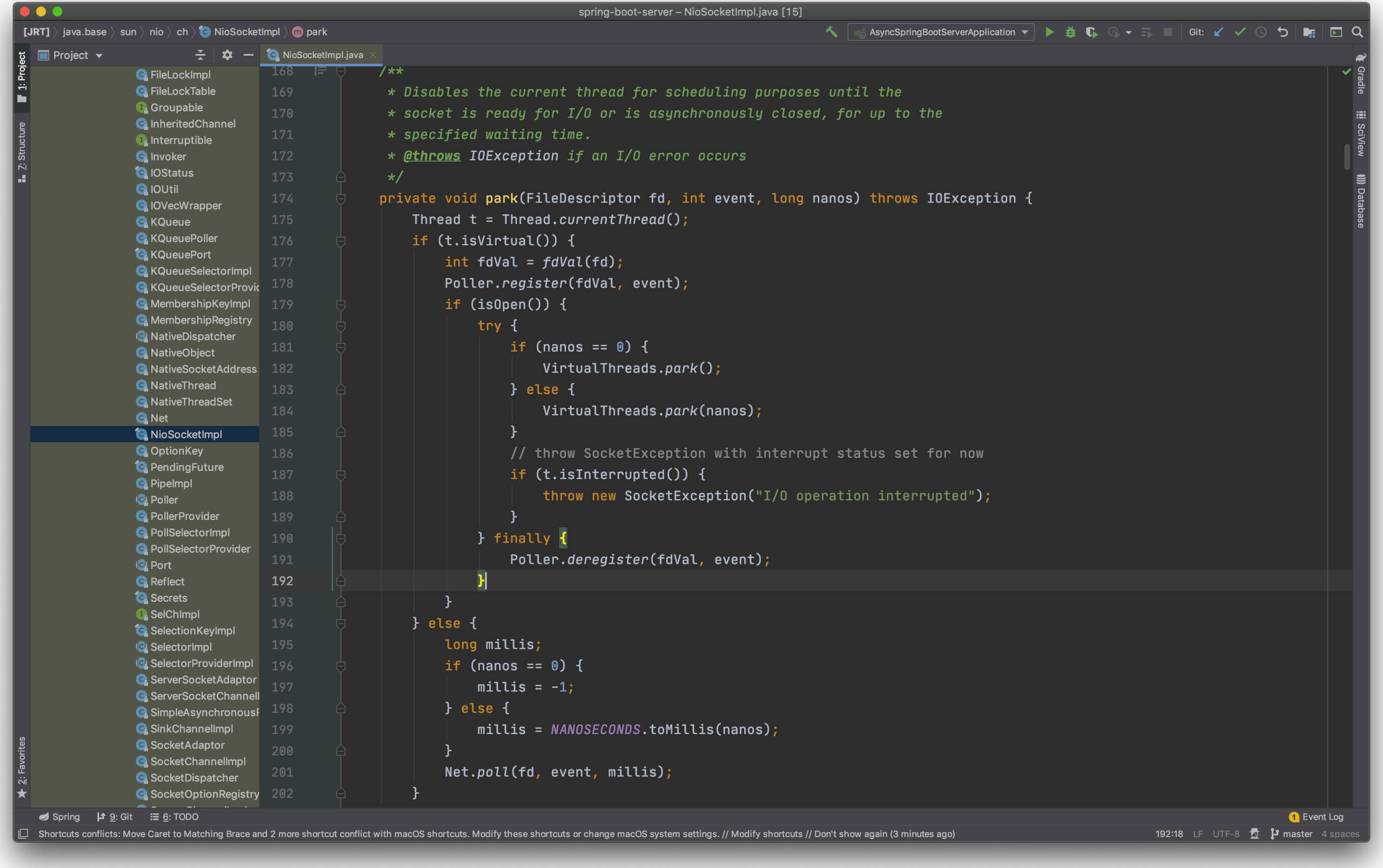
The `java.net.Socket` and `java.net.ServerSocket` APIs delegate all socket operations to a `java.net.SocketImpl`, a Service Provider Interface (SPI) mechanism that has existed since JDK 1.0. The built-in implementation is termed the “plain” implementation, implemented by the non-public `PlainSocketImpl` with supporting classes `SocketInputStream` and `SocketOutputStream`. `PlainSocketImpl` is extended by two other JDK-internal implementations that support connections through SOCKS and HTTP proxy servers. By default, a `Socket` and `ServerSocket` is created (sometimes lazily) with a SOCKS based `SocketImpl`. In the case of `ServerSocket`, the use of the SOCKS implementation is an oddity that dates back to experimental (and since removed) support for proxying server connections in JDK 1.4.

The new implementation, `NioSocketImpl`, is a drop-in replacement for `PlainSocketImpl`. It is developed to be easy to maintain and debug. It shares the same JDK-internal infrastructure as the New I/O (NIO) implementation so it doesn't need its own native code. It integrates with the existing buffer cache mechanism so that it doesn’t need to use the thread stack for I/O. It uses `java.util.concurrent` locks rather than `synchronized` methods so that it can play well with fibers in the future. In JDK 11, the NIO `SocketChannel` and the other `SelectableChannel` implementations were mostly re-implemented with the same goal in mind.

The following are a few points about the new implementation:

- `SocketImpl` is a legacy SPI mechanism and is very under-specified. The new implementation attempts to be compatible with the old implementation by emulating unspecified behavior and exceptions where applicable. The Risks and Assumptions section below details the behavior differences between the old and new implementations.
- Socket operations using timeouts (`connect`, `accept`, `read`) are implemented by changing the socket to non-blocking mode and polling the socket.
- The `java.lang.ref.Cleaner` mechanism is used to close sockets when the `SocketImpl` is garbage collected and the socket has not been explicitly closed.
- Connection reset handling is implemented in the same way as the old

Example: New Socket API implementation ready for virtual threads



```java
    /**
     * Disables the current thread for scheduling purposes until the
     * socket is ready for I/O or is asynchronously closed, for up to the
     * specified waiting time.
     * @throws IOException if an I/O error occurs
     */
    private void park(FileDescriptor fd, int event, long nanos) throws IOException {
        Thread t = Thread.currentThread();
        if (t.isVirtual()) {
            int fdVal = fdVal(fd);
            Poller.register(fdVal, event);
            if (isOpen()) {
                try {
                    if (nanos == 0) {
                        VirtualThreads.park();
                    } else {
                        VirtualThreads.park(nanos);
                    }
                    // throw SocketException with interrupt status set for now
                    if (t.isInterrupted()) {
                        throw new SocketException("I/O operation interrupted");
                    }
                } finally {
                    Poller.deregister(fdVal, event);
                }
            }
        } else {
            long millis;
            if (nanos == 0) {
                millis = -1;
            } else {
                millis = NANOSECONDS.toMillis(nanos);
            }
            Net.poll(fd, event, millis);
        }
```

Shortcuts conflicts: Move Caret to Matching Brace and 2 more shortcut conflict with macOS shortcuts. Modify these shortcuts or change macOS system settings. // Modify shortcuts // Don't show again (3 minutes ago)

Limitations

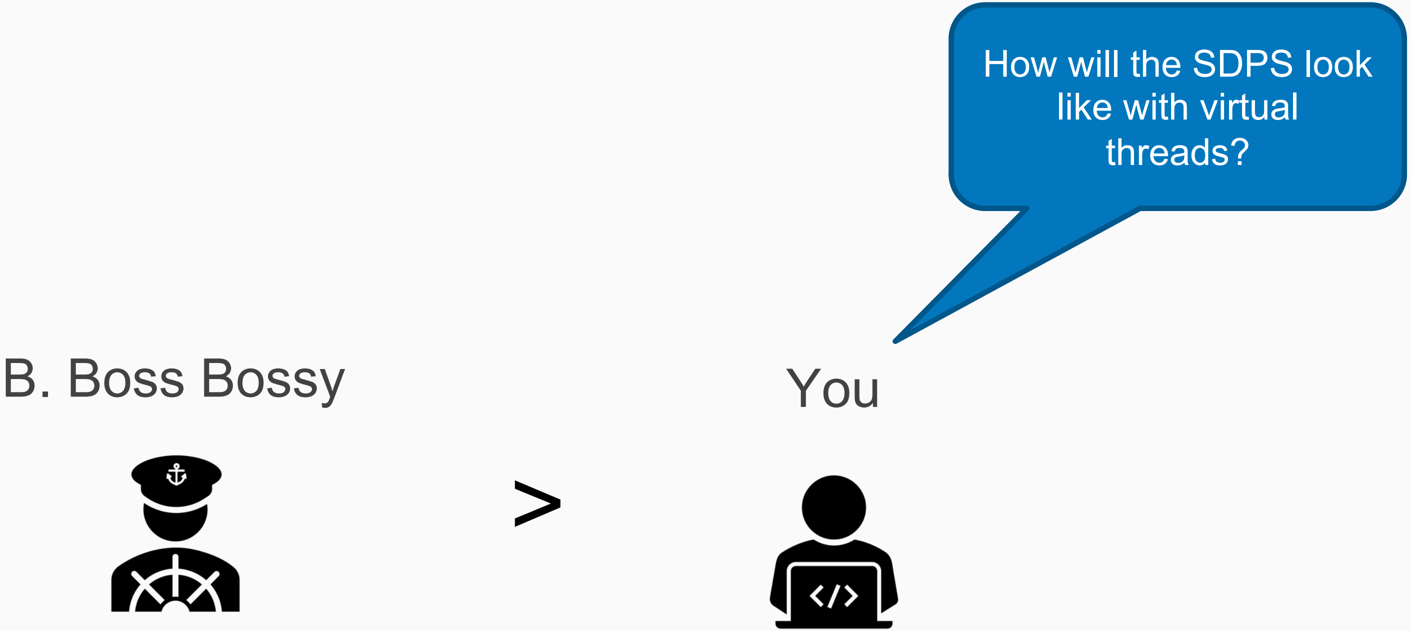
Temporary

- Limited debugging support
    - Dealing with a large number of virtual threads, Setting local variables, Suspending or resuming a virtual thread, Stack traces for fibers will include scheduler related frames
- Not all Java APIs virtual thread ready as of now

Permanent

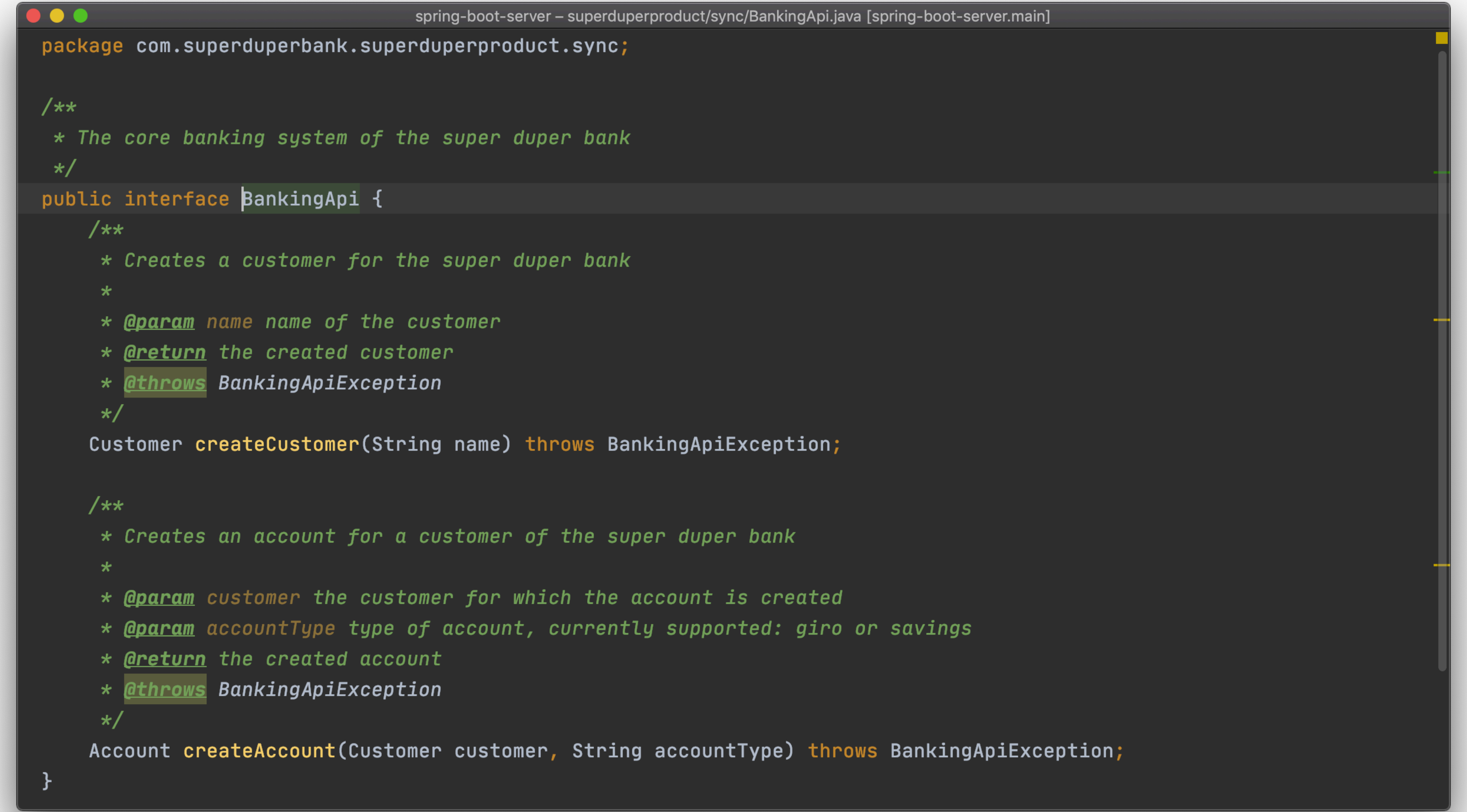
- Semantic differences to threads → not all legacy code will work without changes
- Native frames not supported

Super Duper Bank

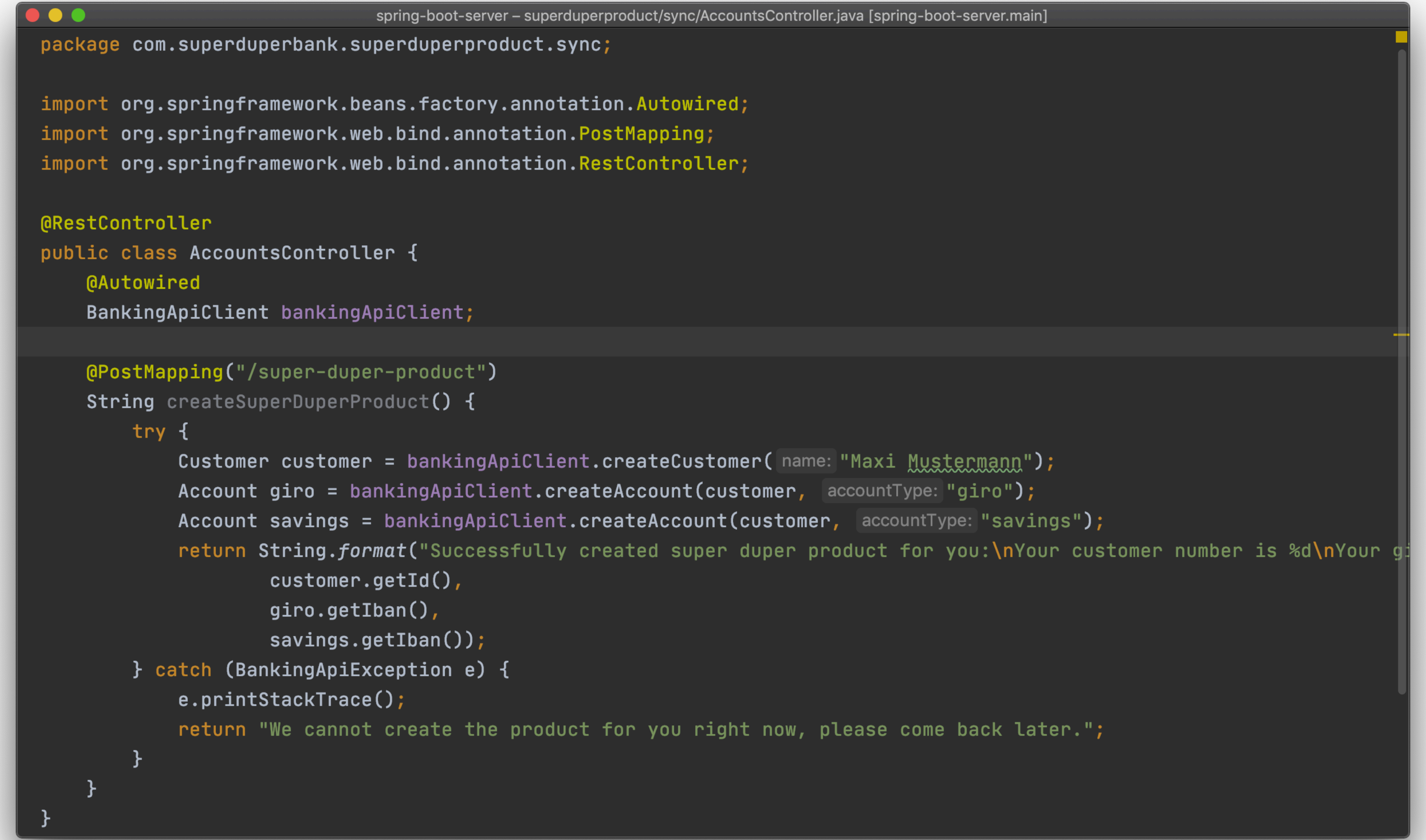


How will the SDPS look like with virtual threads?

B. Boss Bossy > You


spring-boot-server – superduperproduct/sync/BankingApi.java [spring-boot-server.main]

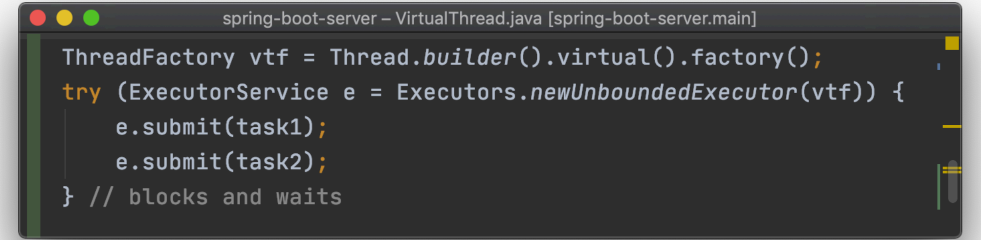
```java
package com.superduperbank.superduperproduct.sync;

/**
 * The core banking system of the super duper bank
 */
public interface BankingApi {
    /**
     * Creates a customer for the super duper bank
     *
     * @param name name of the customer
     * @return the created customer
     * @throws BankingApiException
     */
    Customer createCustomer(String name) throws BankingApiException;

    /**
     * Creates an account for a customer of the super duper bank
     *
     * @param customer the customer for which the account is created
     * @param accountType type of account, currently supported: giro or savings
     * @return the created account
     * @throws BankingApiException
     */
    Account createAccount(Customer customer, String accountType) throws BankingApiException;
}
```


spring-boot-server – superduperproduct/sync/AccountsController.java [spring-boot-server.main]

```java
package com.superduperbank.superduperproduct.sync;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class AccountsController {
    @Autowired
    BankingApiClient bankingApiClient;

    @PostMapping("/super-duper-product")
    String createSuperDuperProduct() {
        try {
            Customer customer = bankingApiClient.createCustomer( name: "Maxi Mustermann");
            Account giro = bankingApiClient.createAccount(customer,  accountType: "giro");
            Account savings = bankingApiClient.createAccount(customer,  accountType: "savings");
            return String.format("Successfully created super duper product for you:\nYour customer number is %d\nYour gi
                    customer.getId(),
                    giro.getIban(),
                    savings.getIban());
        } catch (BankingApiException e) {
            e.printStackTrace();
            return "We cannot create the product for you right now, please come back later.";
        }
    }
}
```

Sneak peek: Structured concurrency

- Threads normally “float around” in application
- Idea: Bind thread lifetimes to code blocks
- Currently implemented with try-with-resources syntax
- Final design still in discussion



spring-boot-server – VirtualThread.java [spring-boot-server.main]

```java
ThreadFactory vtf = Thread.builder().virtual().factory();
try (ExecutorService e = Executors.newUnboundedExecutor(vtf)) {
    e.submit(task1);
    e.submit(task2);
} // blocks and waits
```

Cool! How can I try it out?

https://wiki.openjdk.java.net/display/loom

- Download preview build https://jdk.java.net/loom/
- Configure new JDK in Intellij (or Eclipse 🙈)
- Spawn 100k Virtual Threads
- Wait for release in Java 1X

Key takeaways

- Blocking OS calls forces you to have one thread per ”program” (e.g. request)
- Non-Blocking I/O calls are complex
- Event-based libraries (libuv, Netty) wrap non-blocking OS calls and provide asynchronous abstractions
  - Callback: simple, not composable, Futures: composable but “unnatural” usage
  - Async/await: Syntax to make working with Futures more natural
- Project Loom implements lightweight virtual threads in the Java platform
  - No blue/red world problem, just write synchronous code as usual, use your favorite (synchronous-style) libraries and enjoy more efficiency (e.g. Spring Web, JDBC)
  - Virtual threads are cheap – have millions of them
  - Still uses non blocking IO under the hood – but wraps them in existing synchronous APIs

→ Final question: Is the virtual thread approach superior to the event-loop model?

I want to learn more!

In depth article about Project Loom and it’s current state (May 2020)	https://cr.openjdk.java.net/~rpressler/loom/loom/sol1_part1.html
Blue/red world problem	http://journal.stuffwithstuff.com/2015/02/01/what-color-is-your-function/
Build an event loop in Rust	https://cfsamson.github.io/book-exploring-async-basics/1_concurrent_vs_parallel.html
Implement green threads in Rust in 200 lines	https://cfsamson.gitbook.io/green-threads-explained-in-200-lines-of-rust/

Thank you! Questions?

Lukas Steinbrecher

Developer @ Senacor

lukstei.com
github.com/lukstei

lukas.steinbrecher@senacor.com