#6: Kotlin Coroutines

juno fitbit SPACE

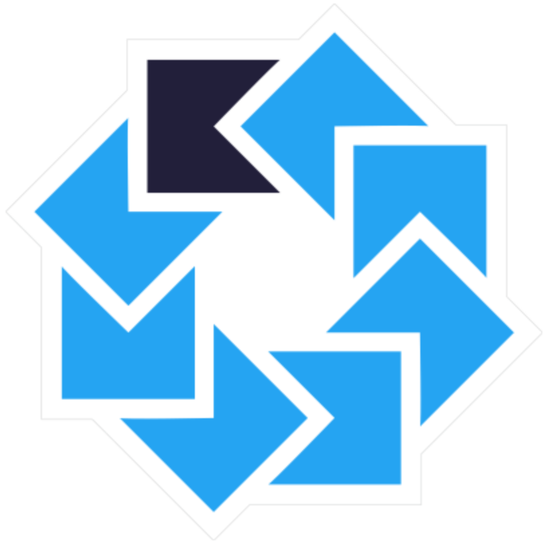
school.kt

Руслан Ибрагимов

Full Stack Developer @ ObjectStyle

@HeapyHop

ruslan@Ibragimov.by



fitbit®

SPACE

juno

Менторы

Чай или кофе?

http://etc.ch/bStx

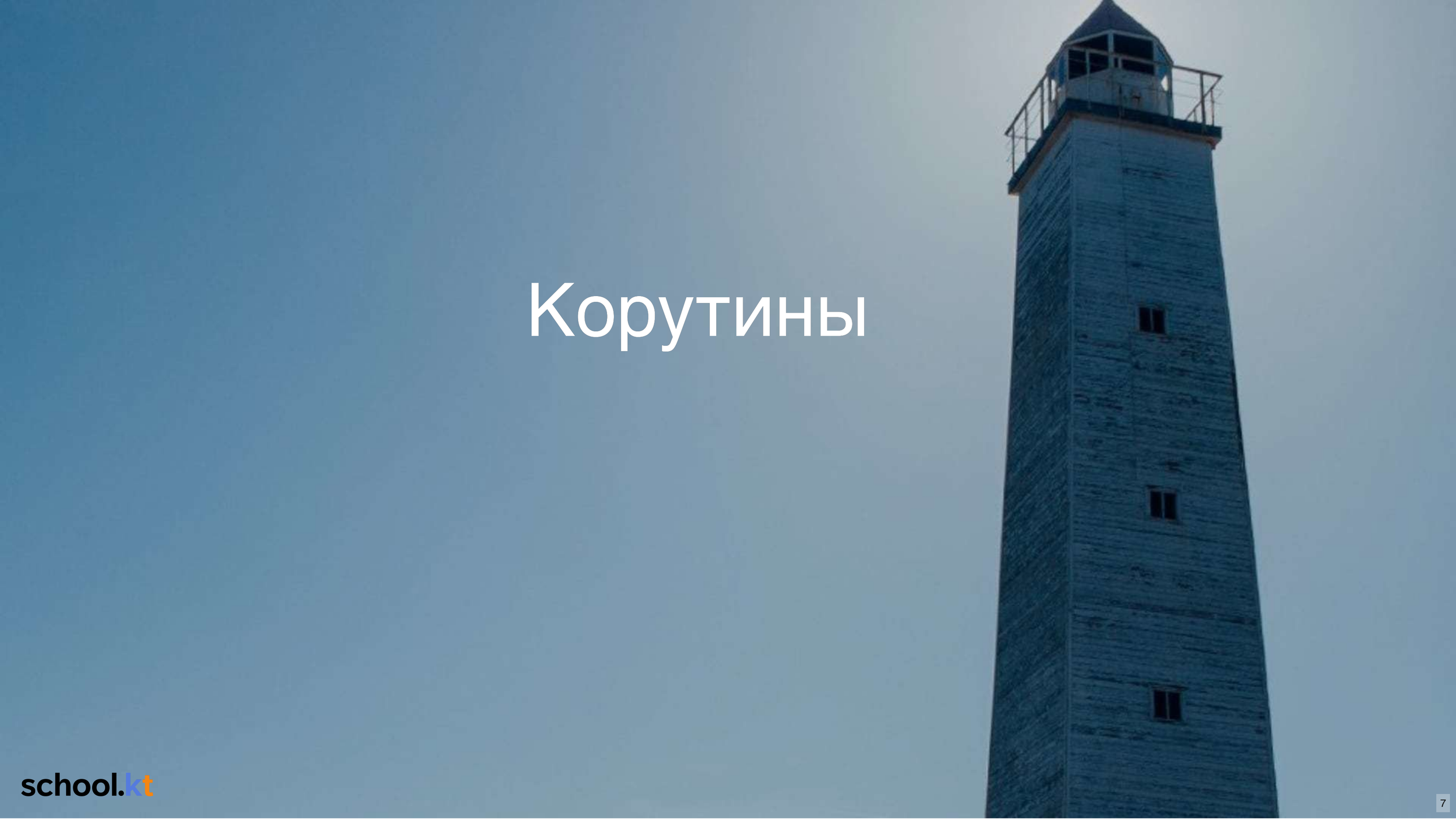
35 votes - 35 participants

Пишете ли вы асинхронный код?

http://etc.ch/bStx

35 votes - 35 participants

Корутины



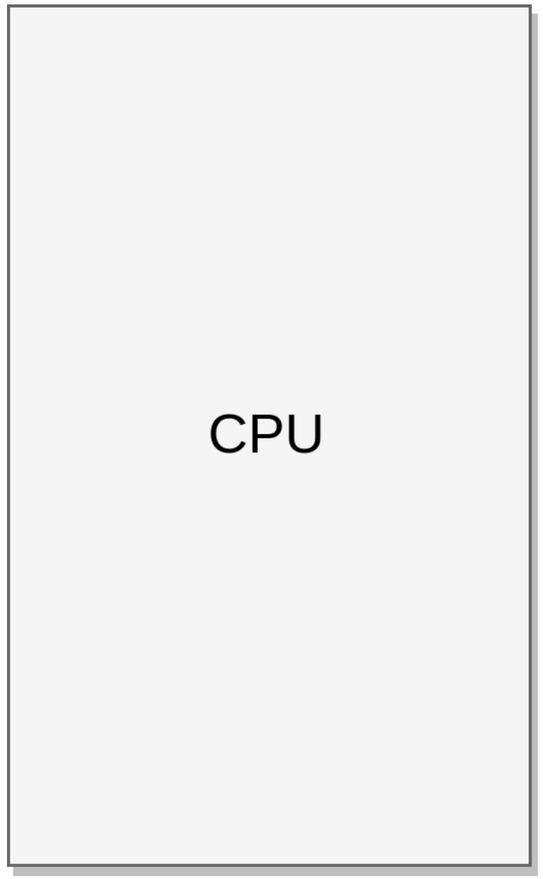
Сегодня

- Часть 1
  - Блокирующие API
  - Неблокирующие API
  - Асинхронность
  - От колбеков к будущему
  - Reactive Manifesto, Reactive Streams
  - Async/Await

Сегодня

- Часть 2
  - Корутины
  - Особенности реализации на JVM
  - Structured Concurrency
  - Обзор kotlinx.coroutines

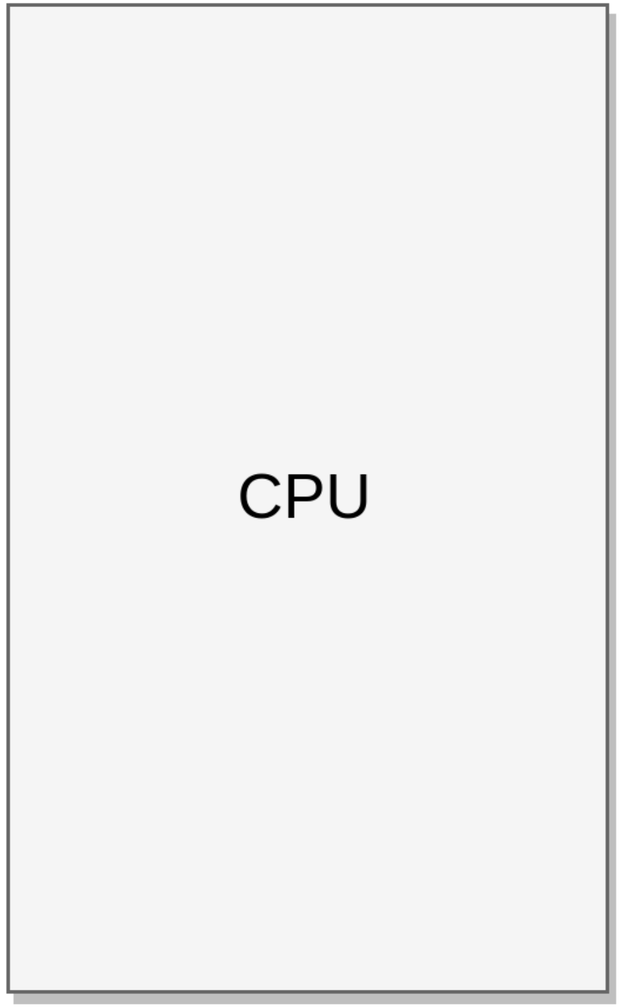
Матчасть



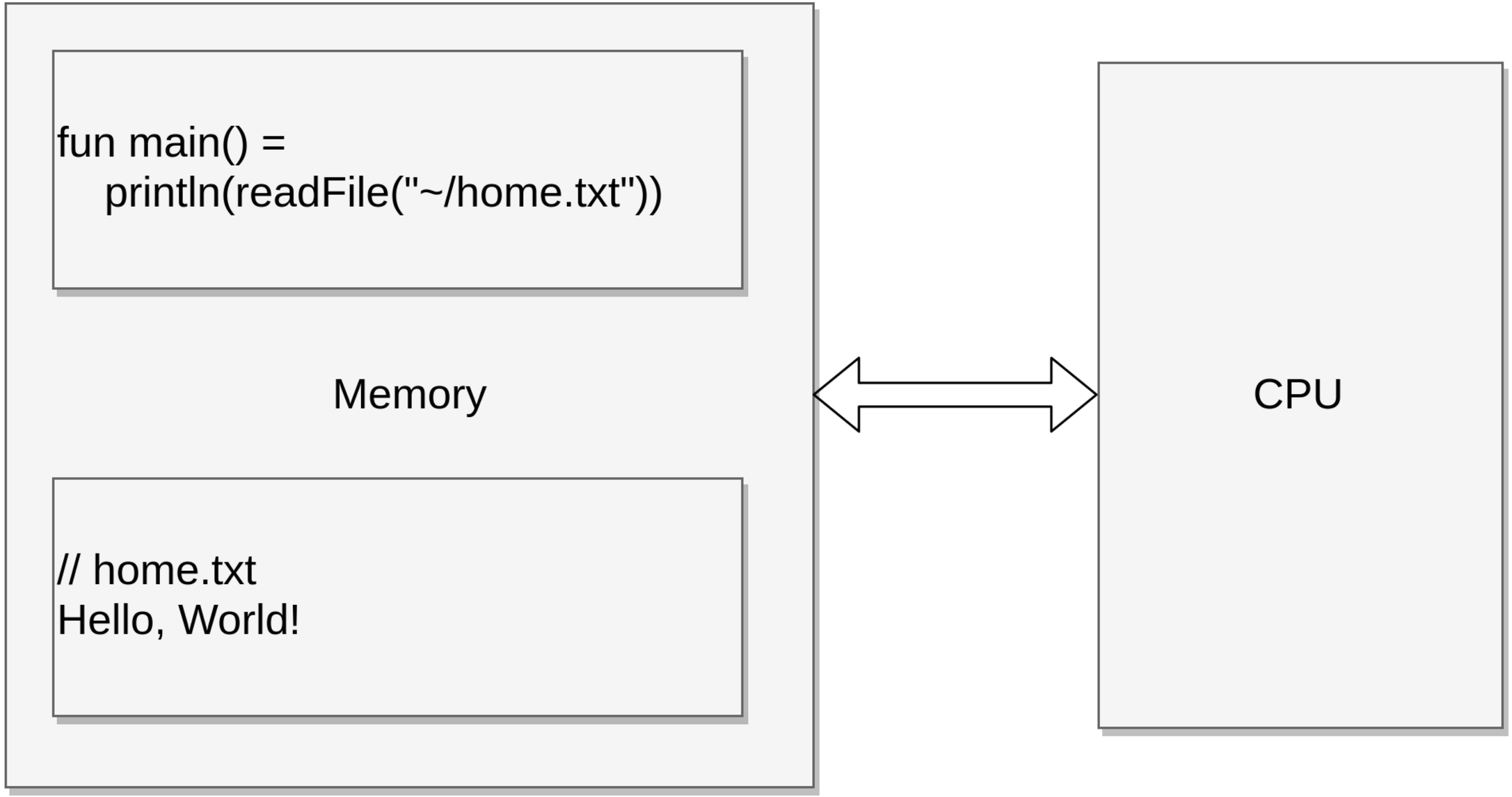
Матчасть

```
fun main() =
    println(readFile("~/home.txt"))
```

```
// home.txt
Hello, World!
```



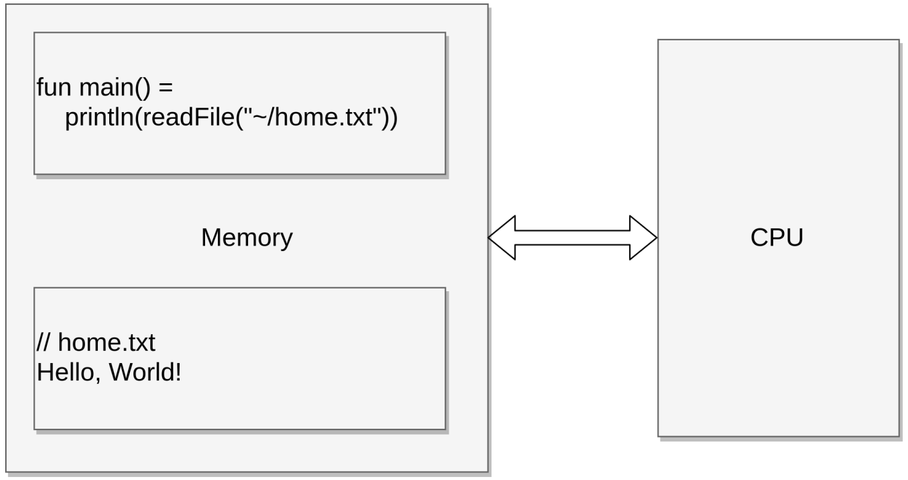
Матчасть

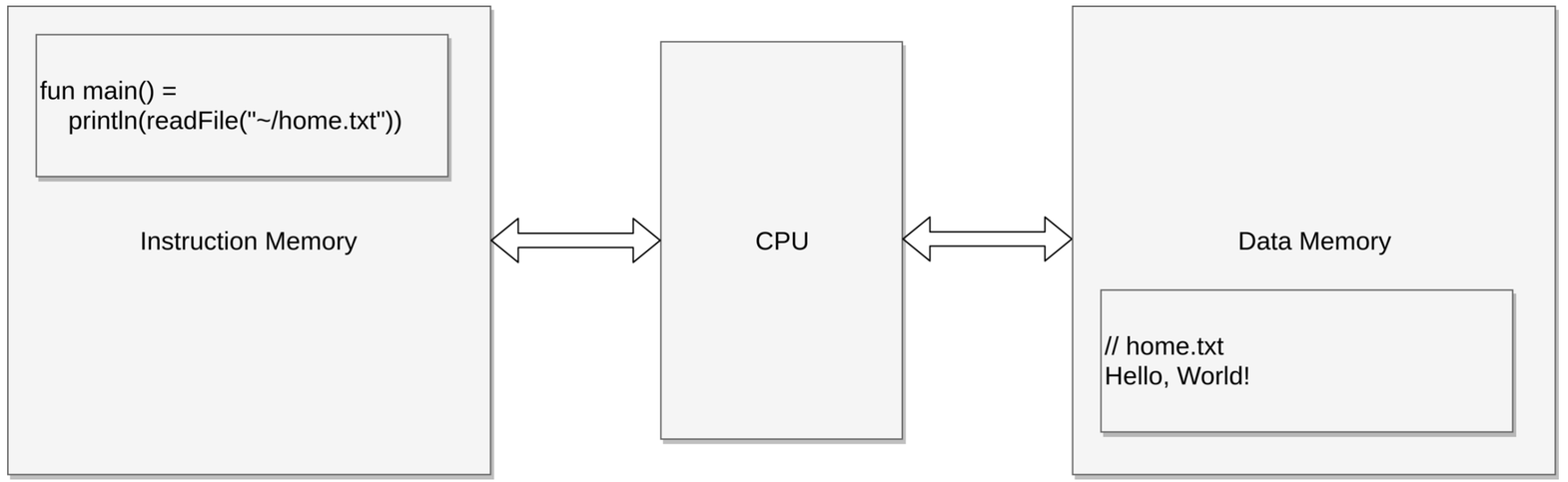

fun main() =
println(readFile("~/home.txt"))
Memory
// home.txt
Hello, World!
CPU

Большинство современных компьютеров работают на

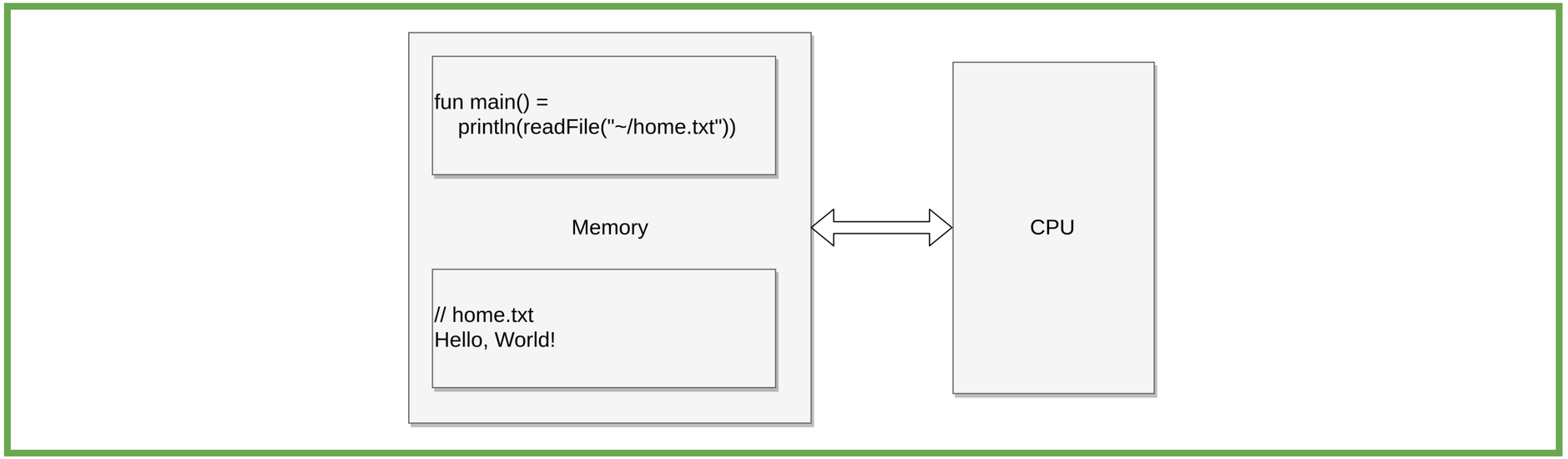
http://etc.ch/bStx

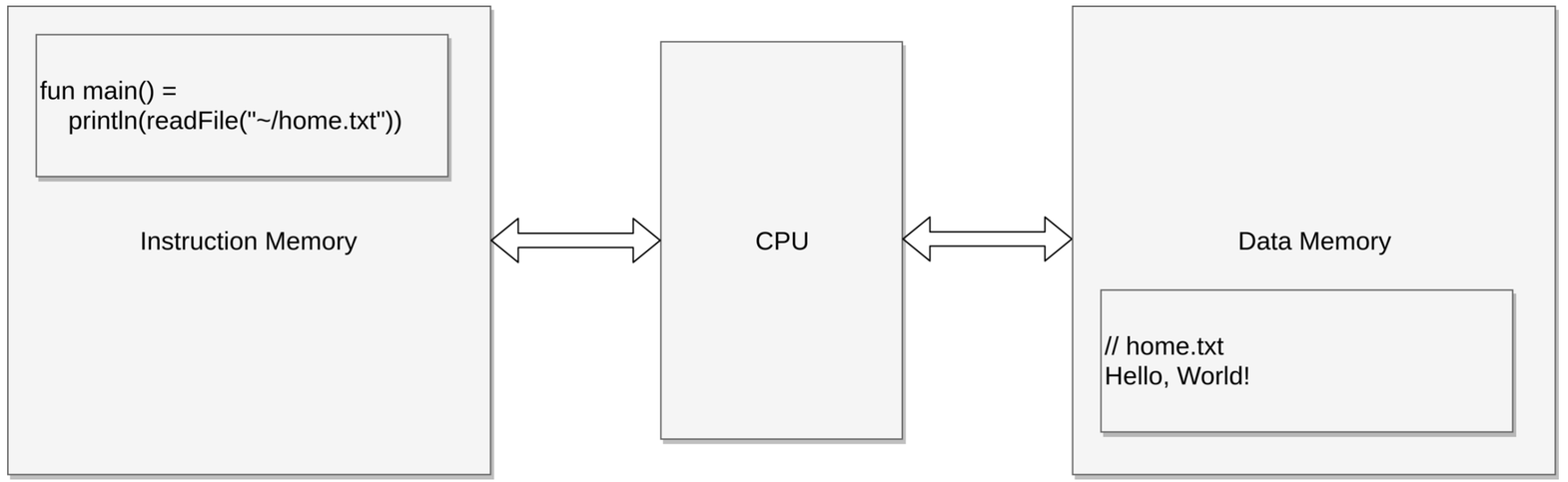
Матчасть


fun main() =
println(readFile("~/home.txt"))
Memory
// home.txt
Hello, World!
CPU

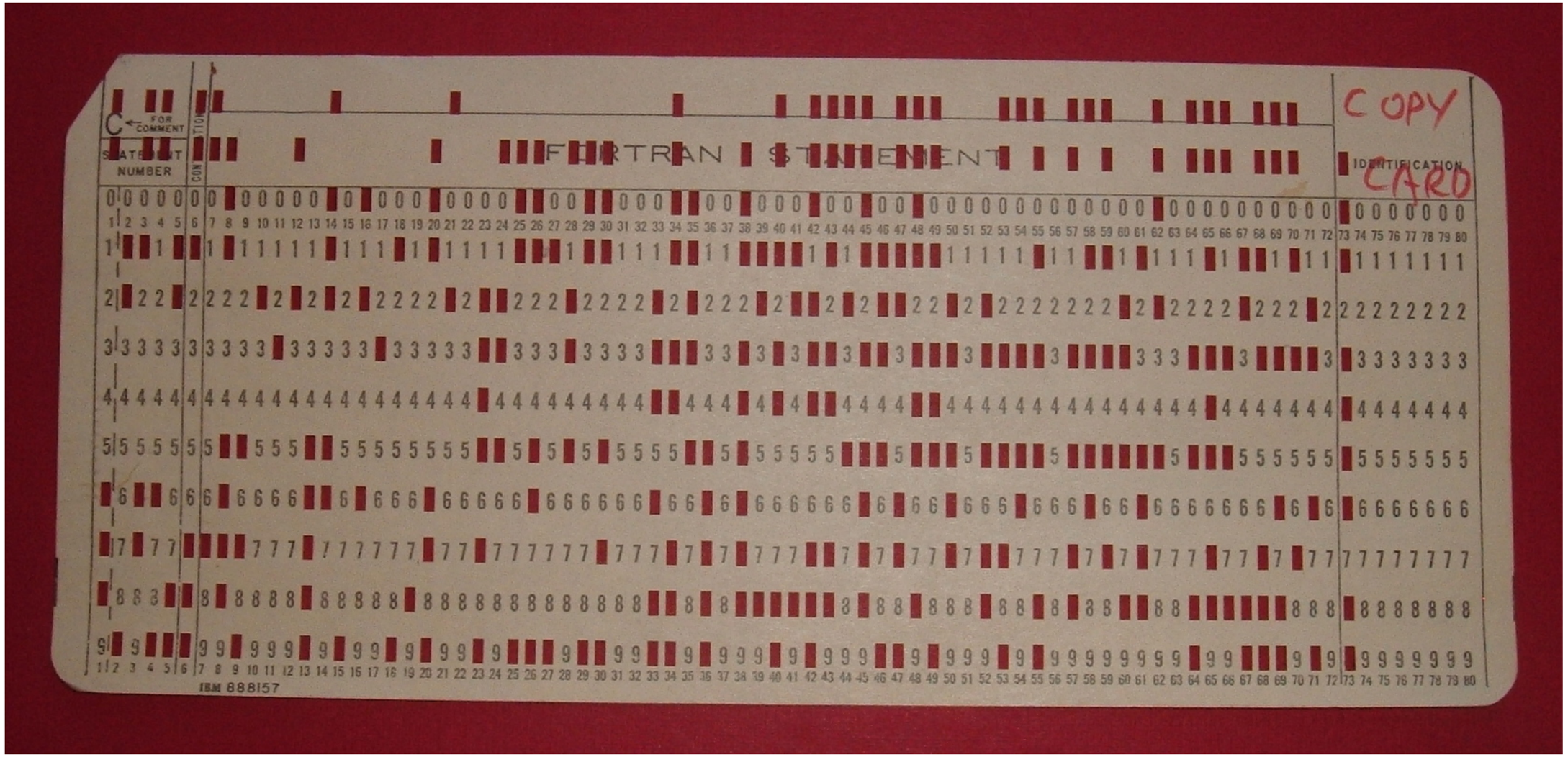

fun main() =
println(readFile("~/home.txt"))
Instruction Memory
CPU
Data Memory
// home.txt
Hello, World!

Матчасть

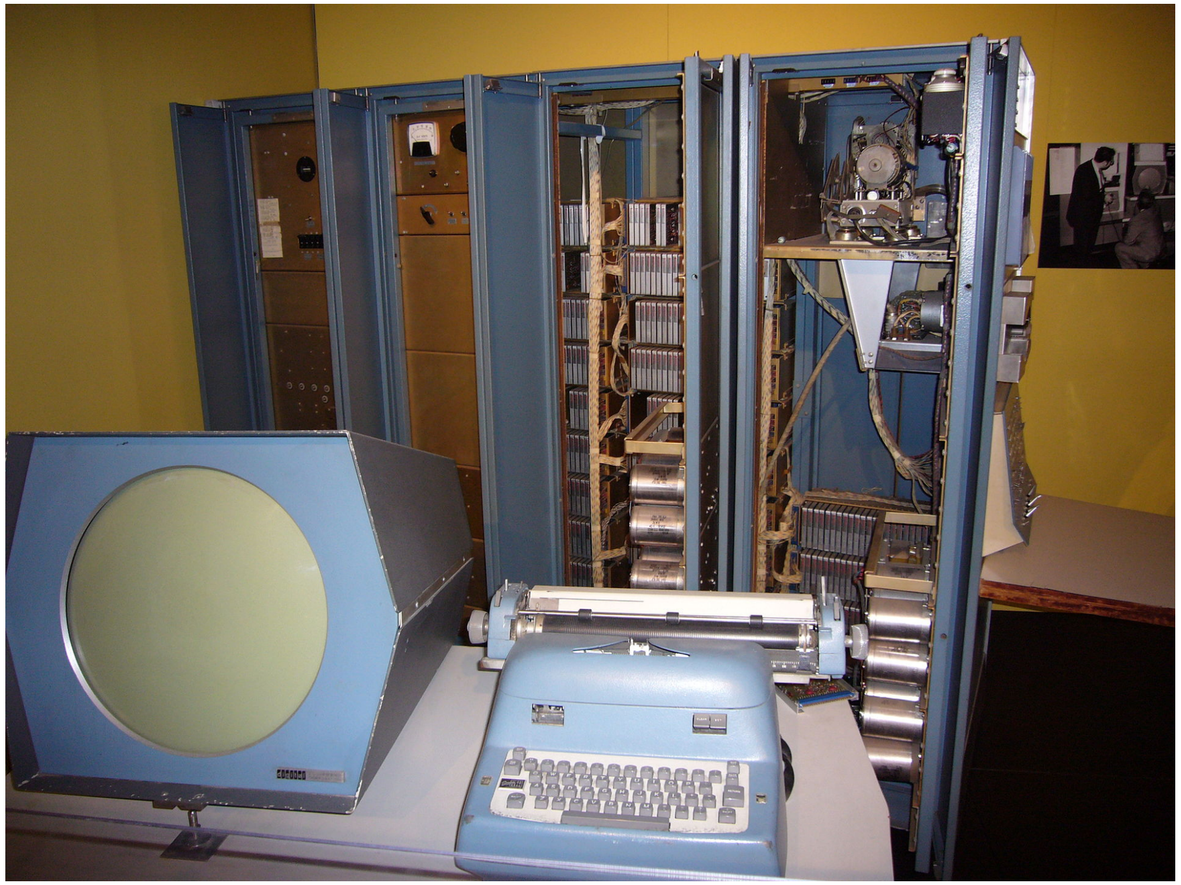

fun main() =
println(readFile("~/home.txt"))
Memory
// home.txt
Hello, World!
CPU

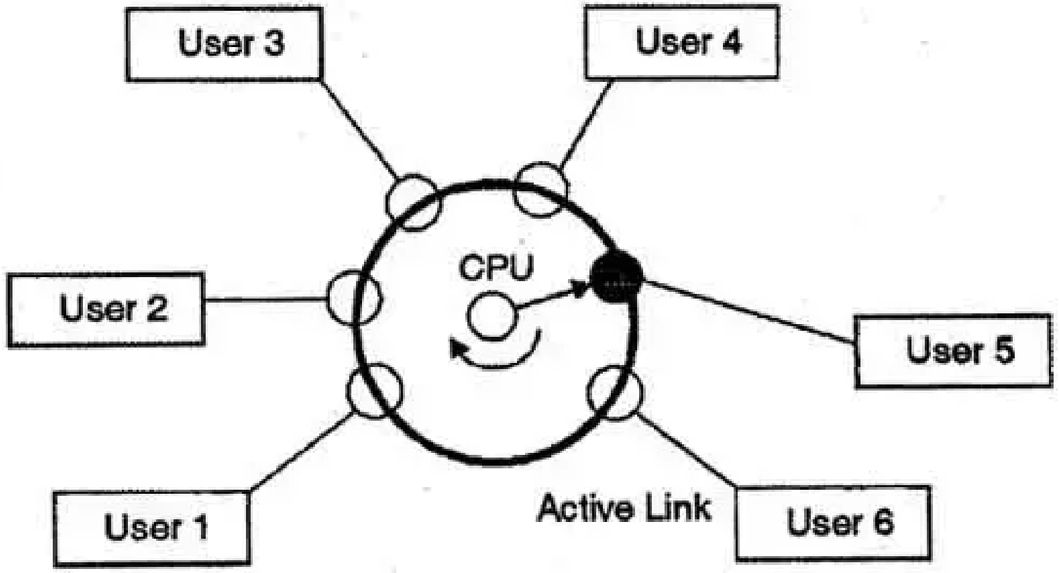

fun main() =
println(readFile("~/home.txt"))
Instruction Memory
CPU
Data Memory
// home.txt
Hello, World!

Матчасть



Матчасть

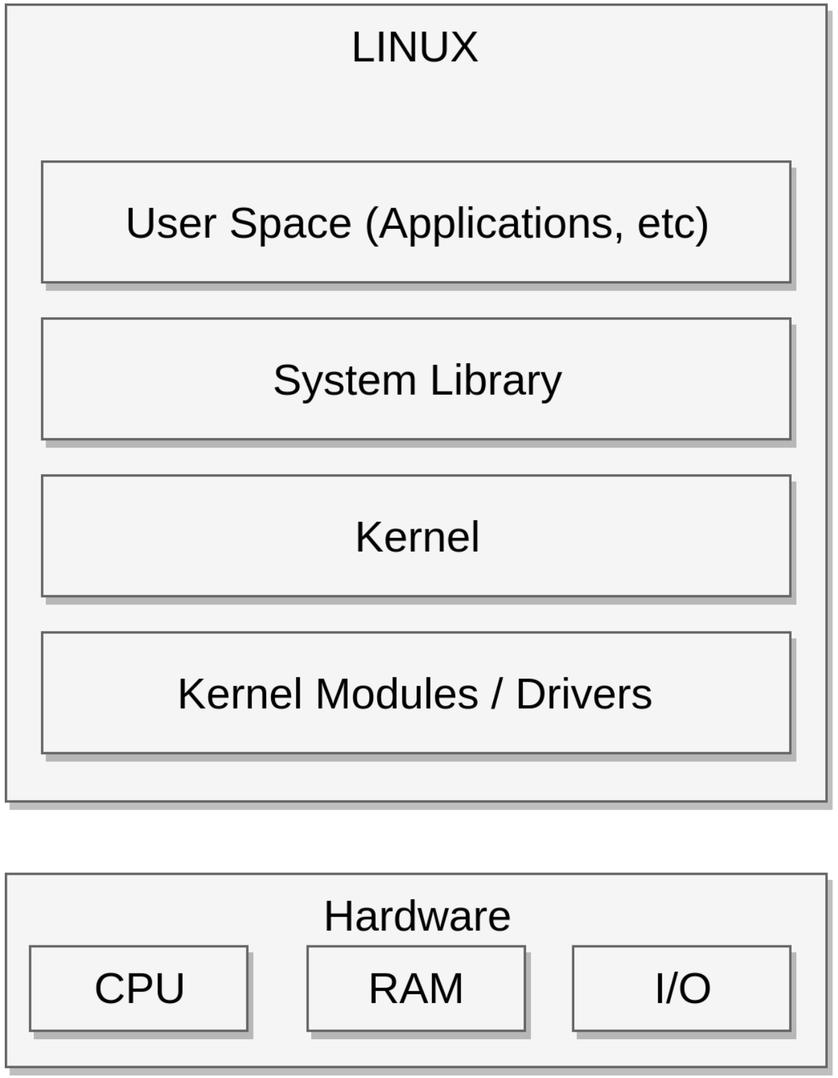




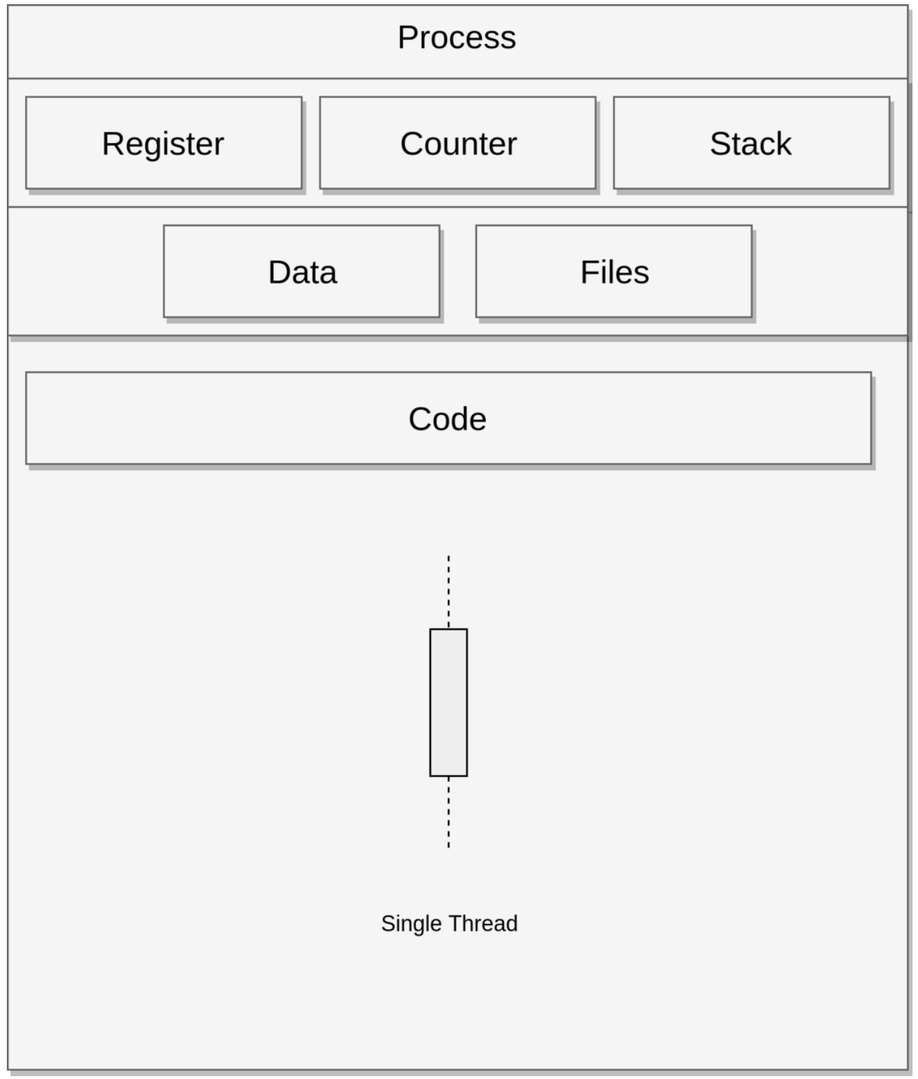
Steven Levy. Hackers: Heroes of the Computer Revolution

Matthew Hutchinson - https://www.flickr.com/photos/hiddenloop/307119987/

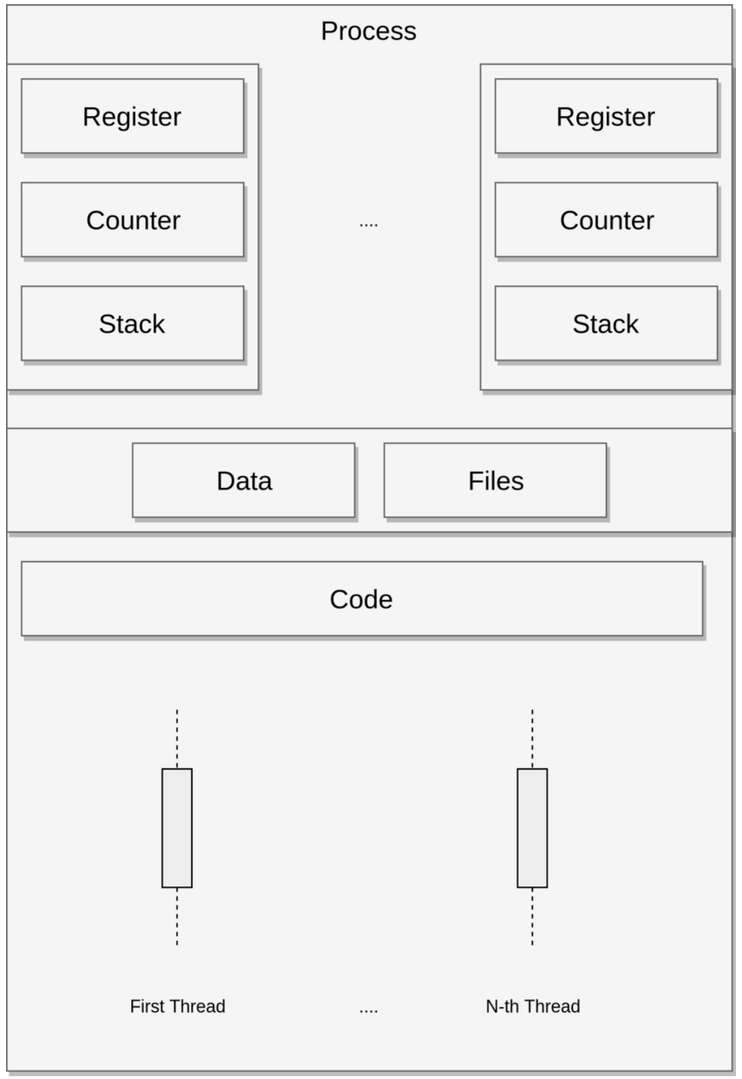
Операционная Система



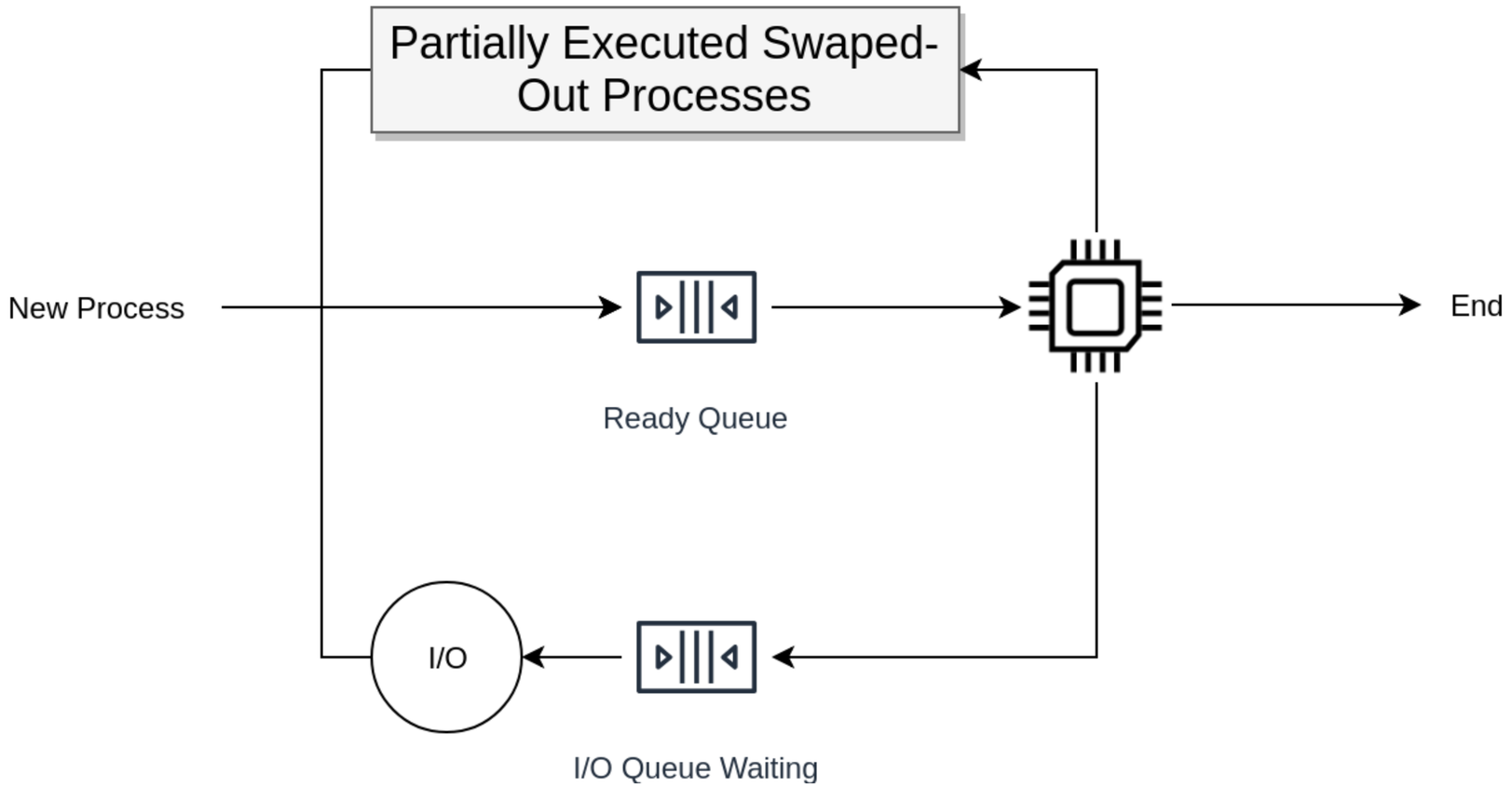
Процесс


Process
Register
Counter
Stack
Data
Files
Code
Single Thread

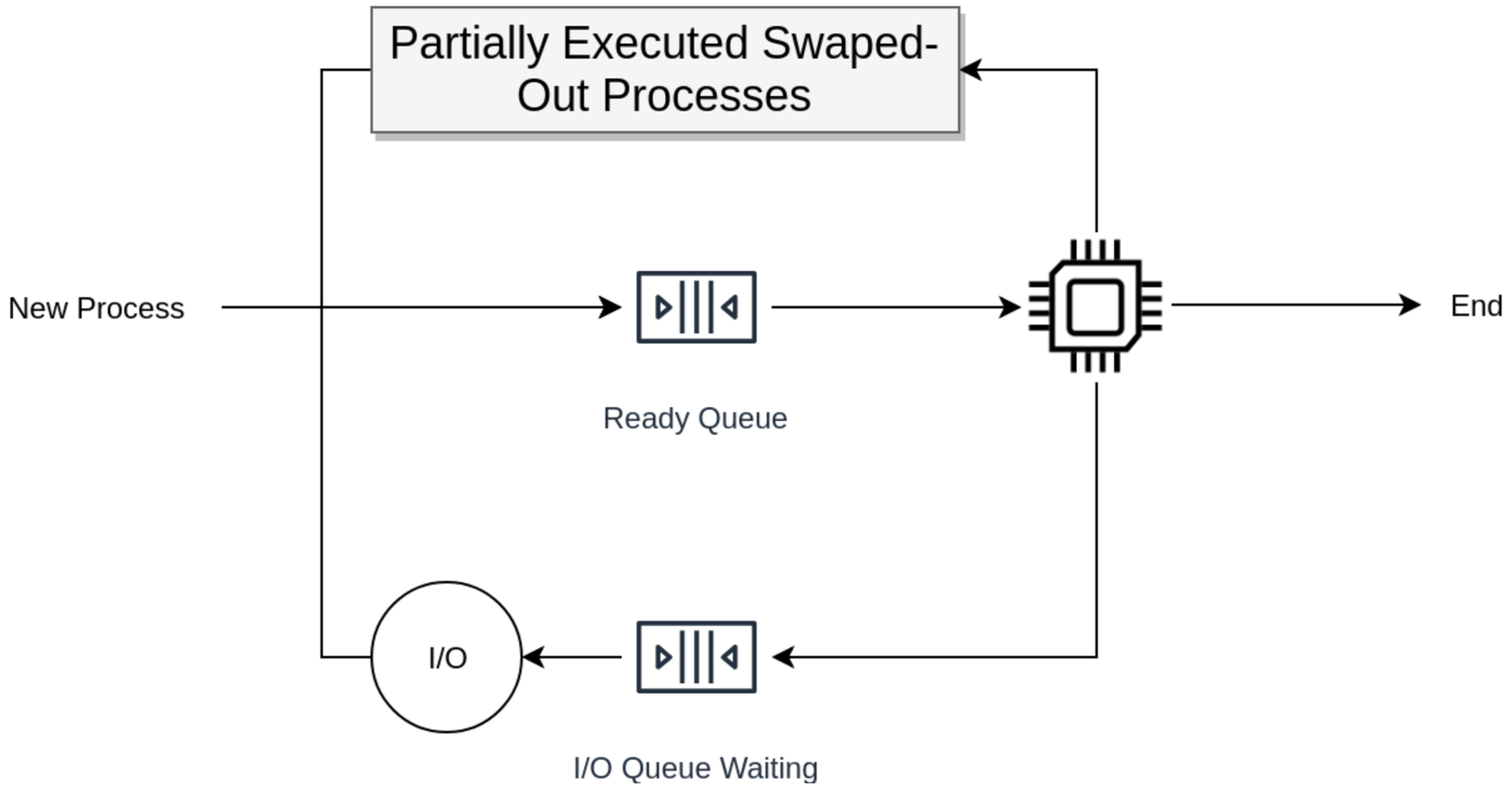
Поток


Process
Register
Counter
Stack
....
Register
Counter
Stack
Data
Files
Code
First Thread
....
N-th Thread

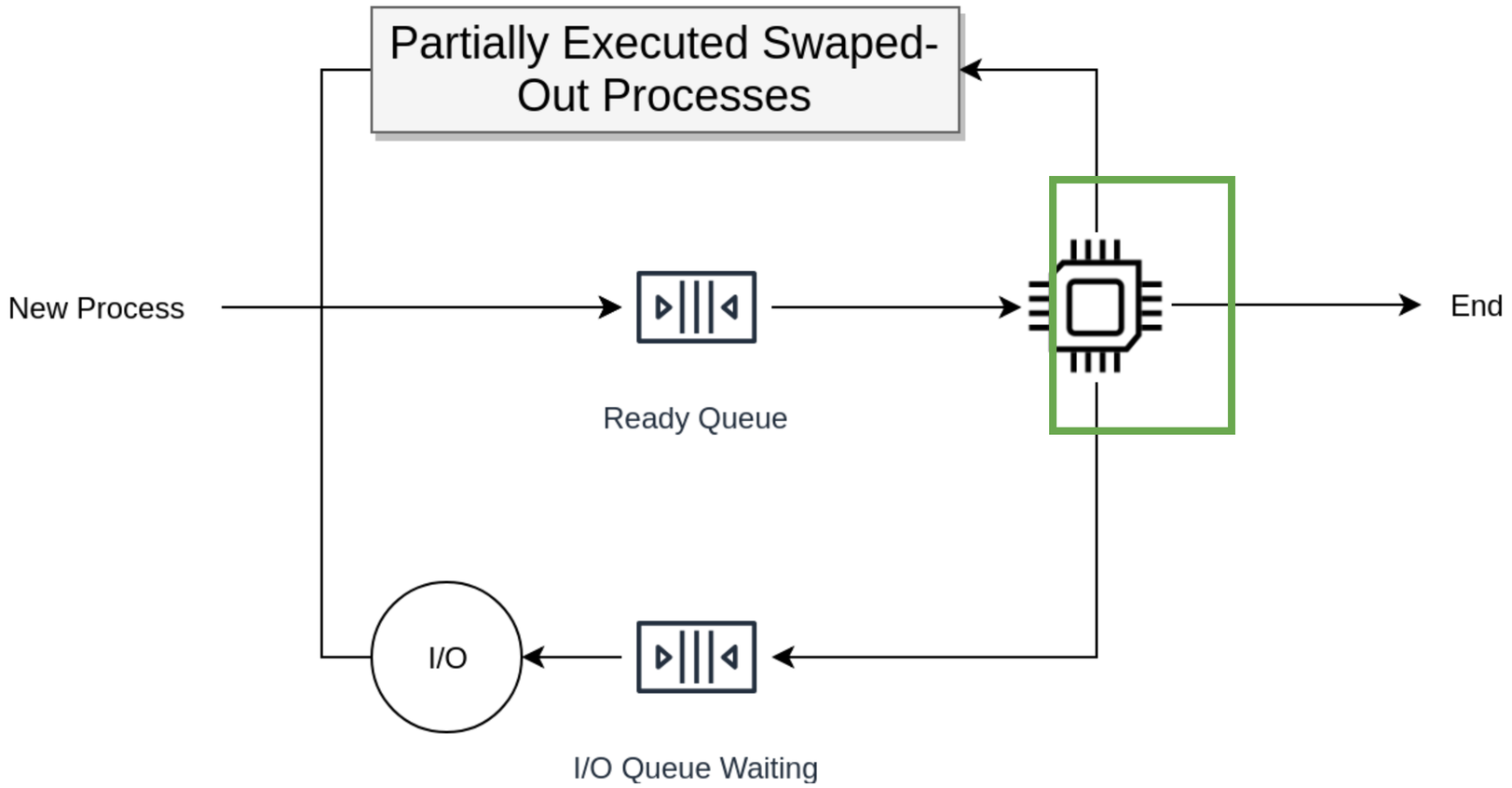
Планировщик



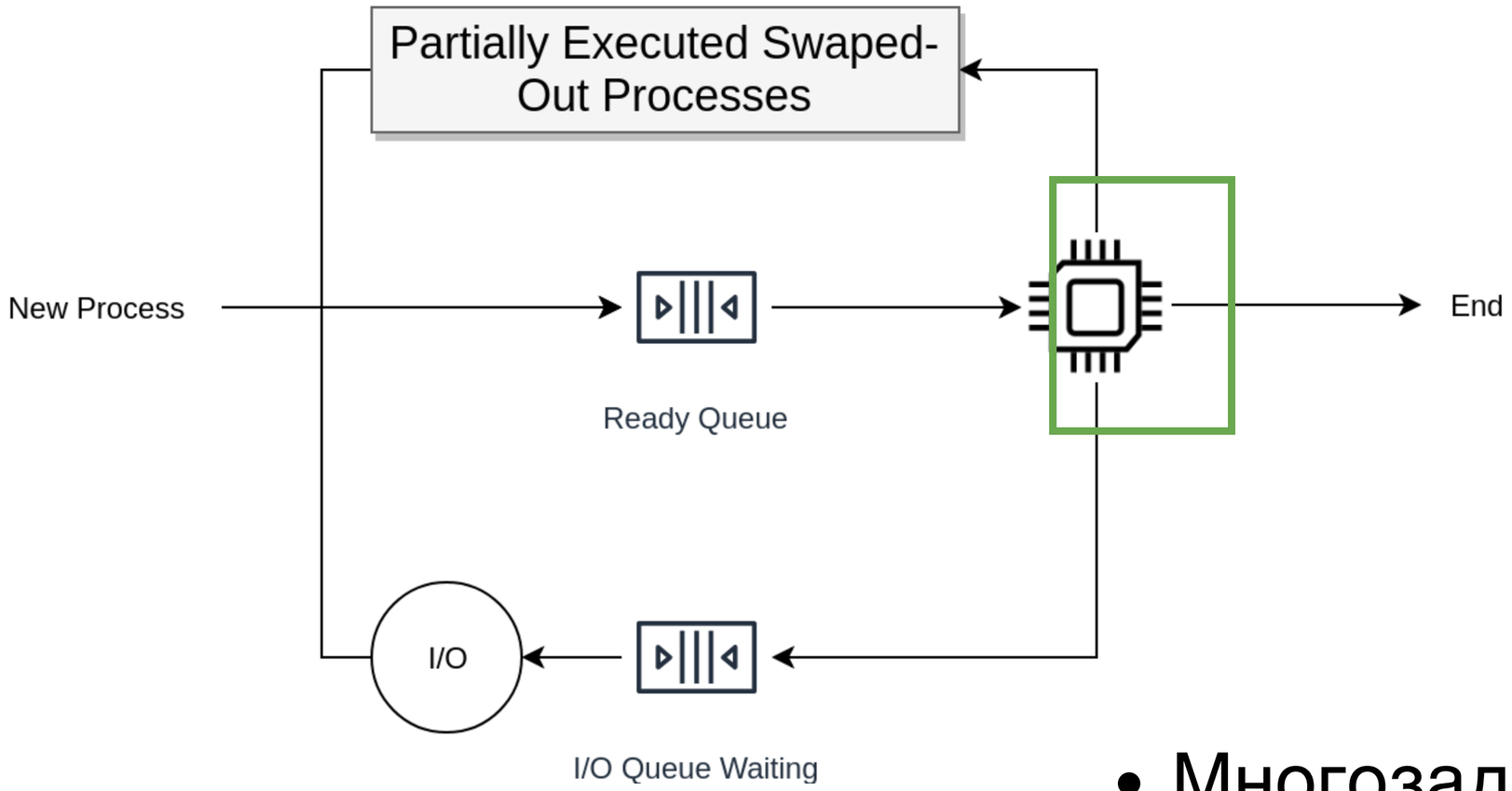
Диспетчер, Переключение Контекста



Диспетчер, Переключение Контекста

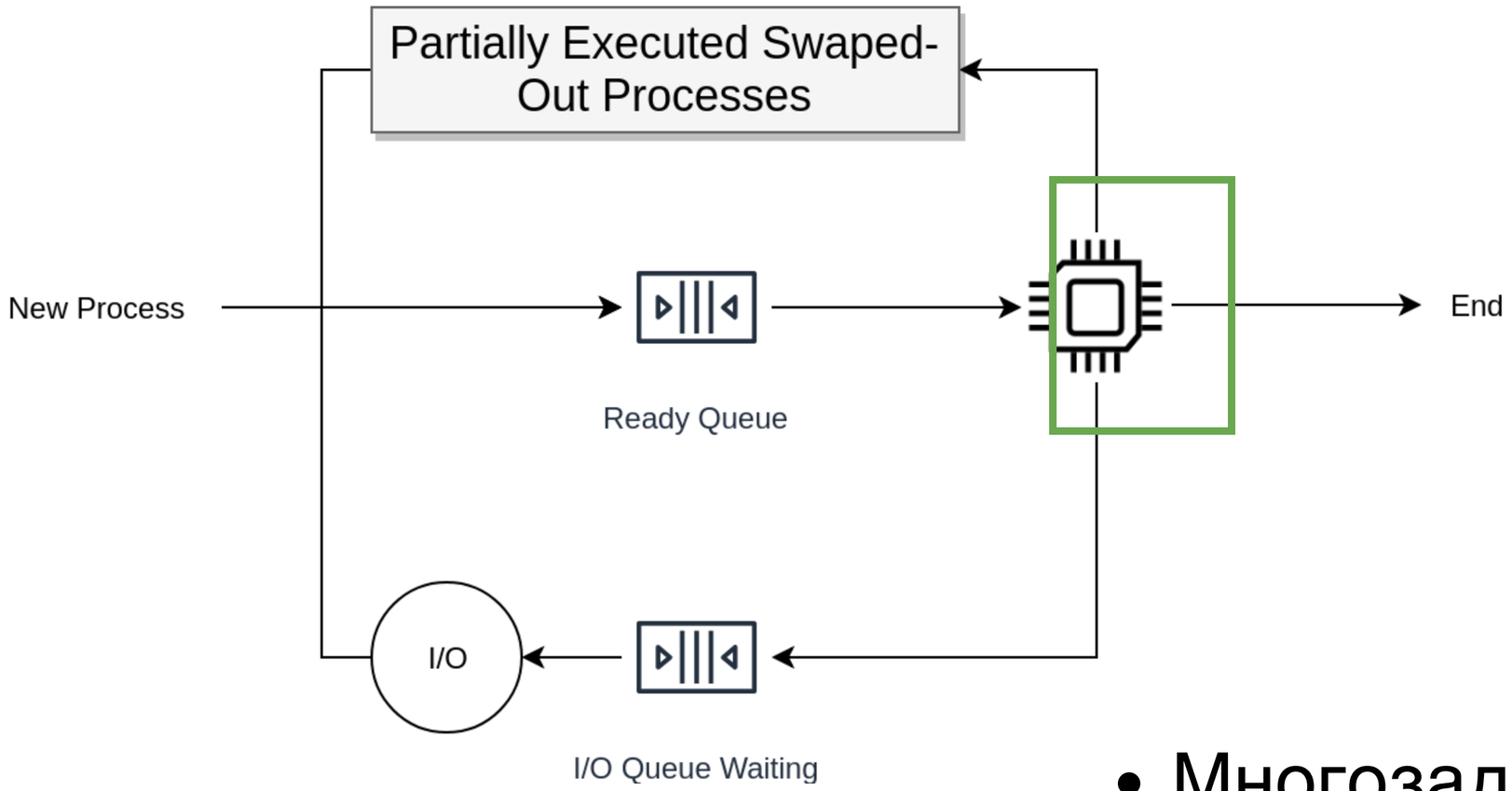


Диспетчер, Переключение Контекста



- Многозадачность

Диспетчер, Переключение Контекста



- Многозадачность
- Ожидание I/O

Многозадачность

- Кооперативная
- Вытесняющая

Многозадачность

- Кооперативная
- Вытесняющая

  Прерывание

  Programmable interval timer (PIT)

  schedule(1ms, handler)

Блокирующие API

- I/O
- Synchronization

Блокирующие API

- I/O
- Synchronization

```
InputStream.read
ServerSocket.accept
```

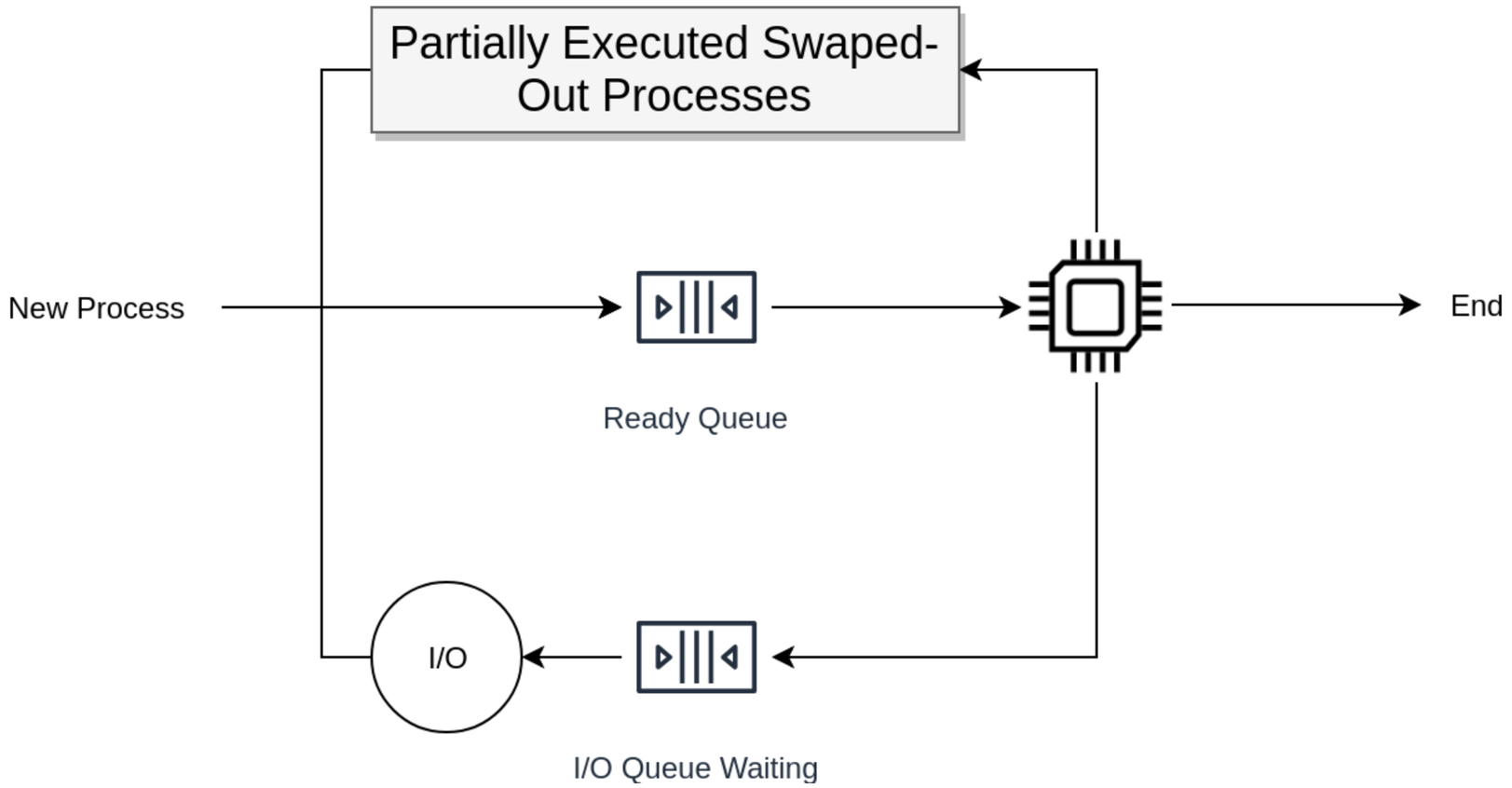
Блокирующие API

- I/O
- Synchronization

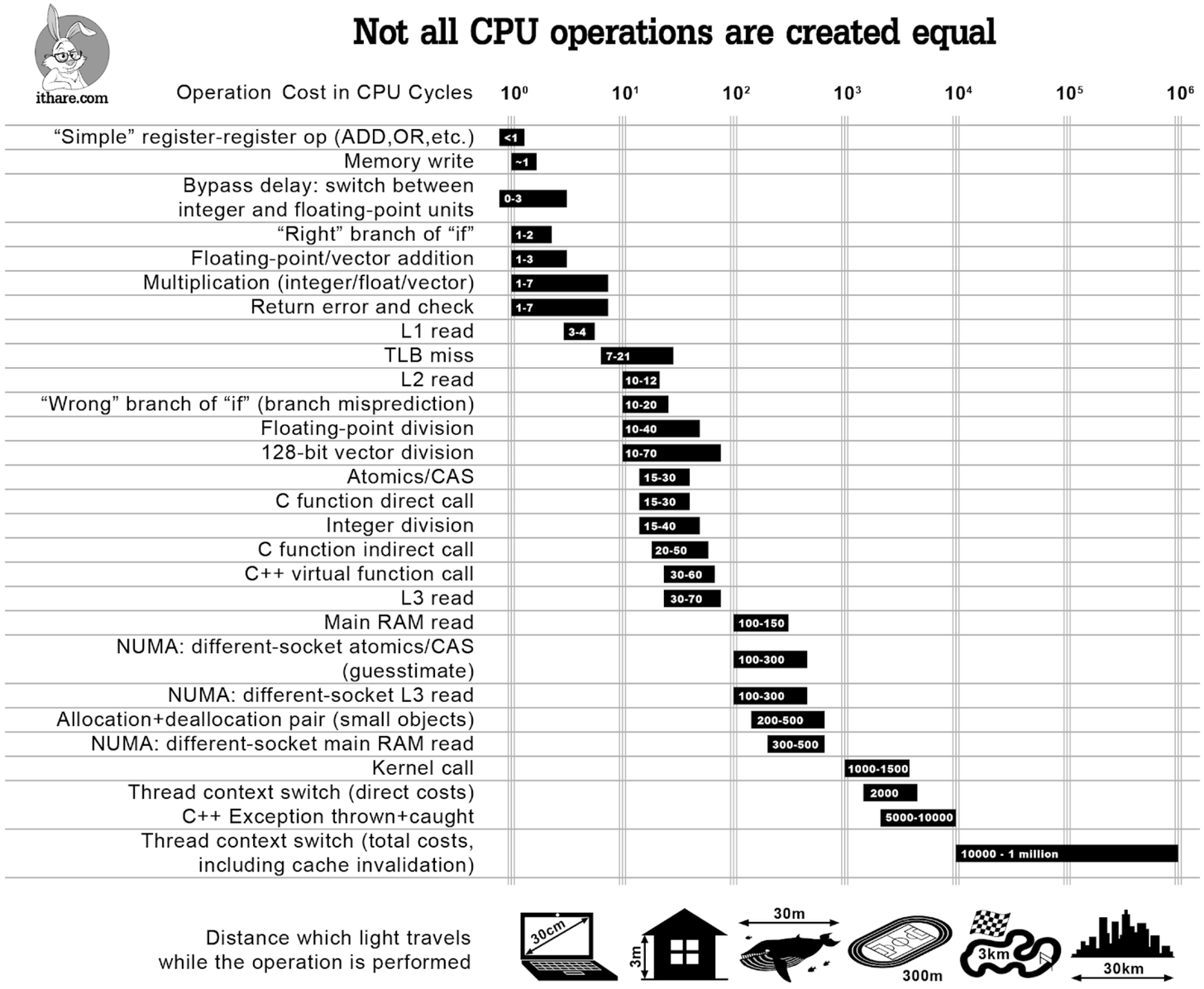
```
InputStream.read
ServerSocket.accept

@Synchronized
fun waitForOtherThread() {}
```

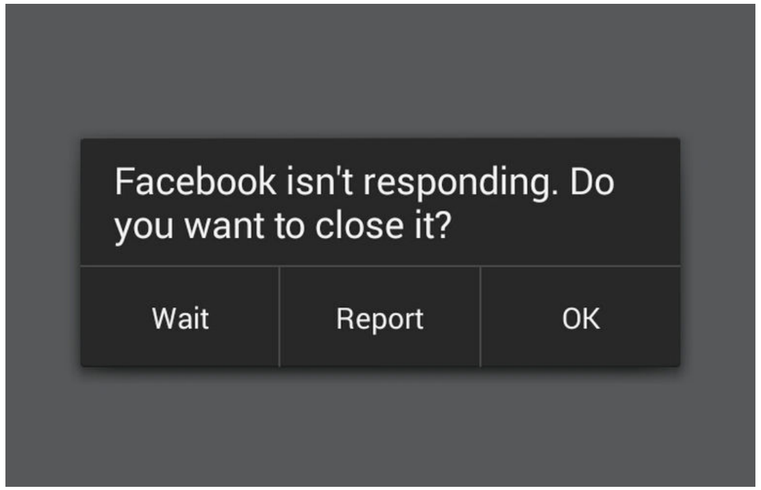
Блокирующие API

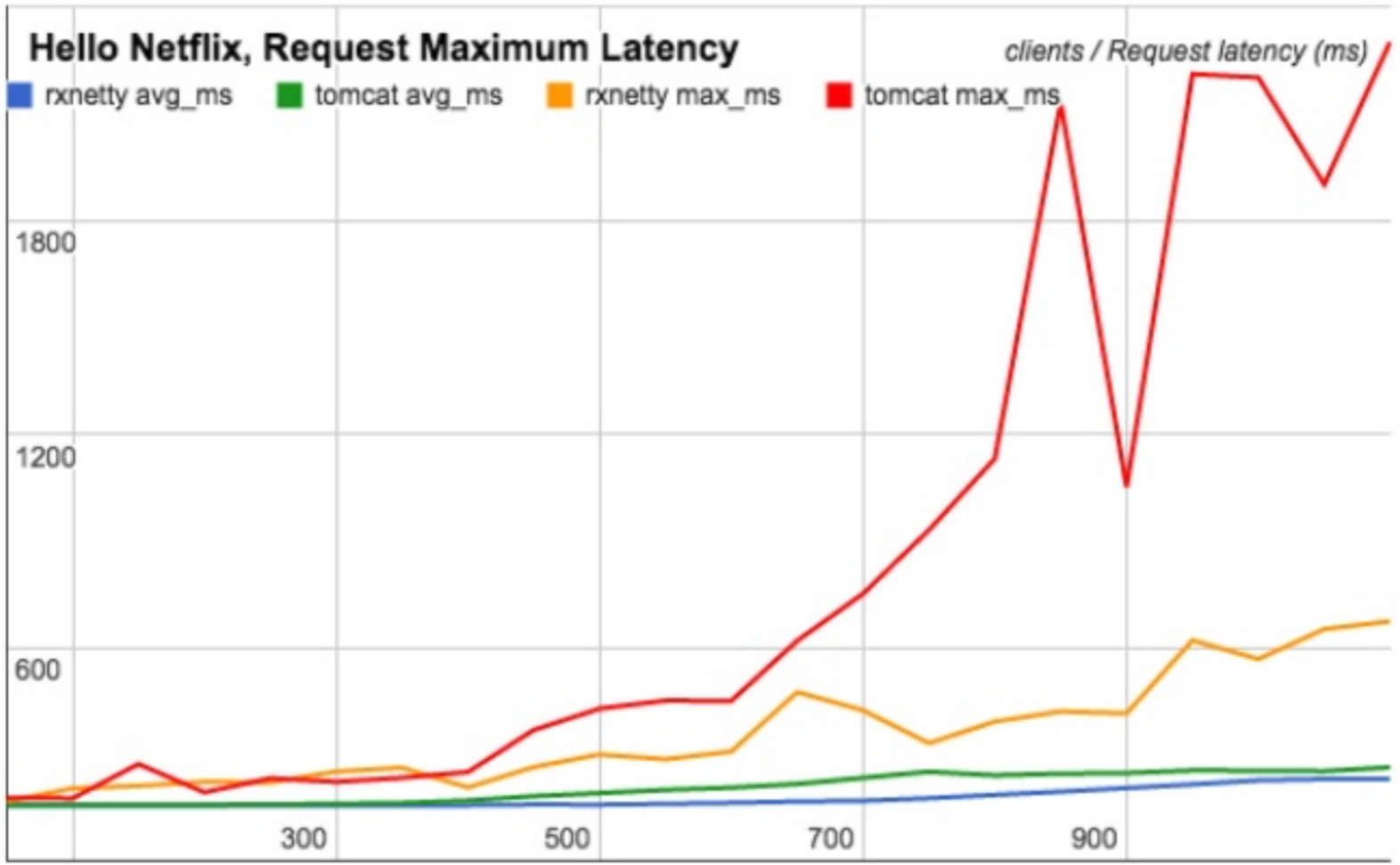


Блокирующие API



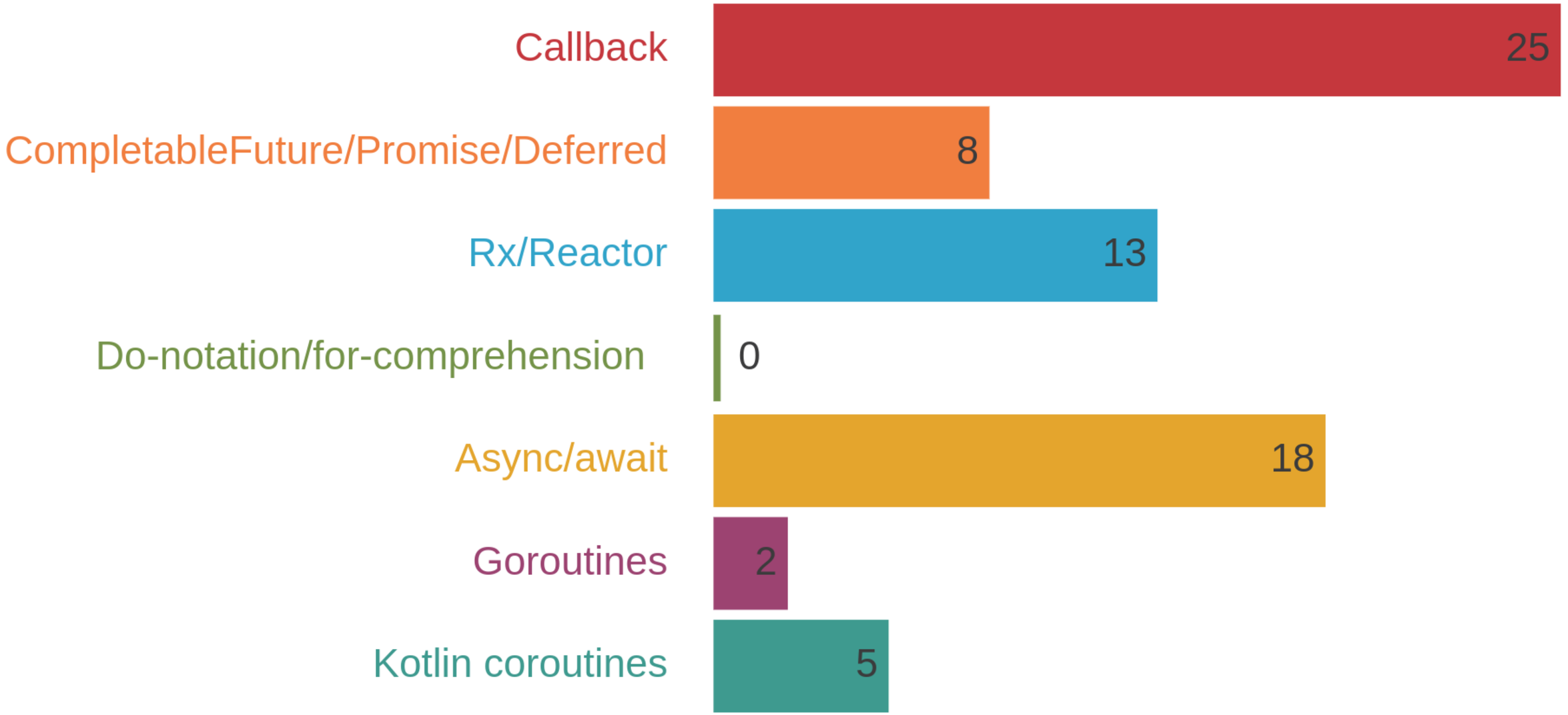
Блокирующие API





Какие API/конструкции языка вы использовали:

http://etc.ch/bStx



71 votes - 34 participants

Не-блокирующие и асинхронные API

Не-блокирующие и асинхронные API

```
interface NonBlocking {
    fun isDataEnough(): Boolean
    fun read(): ByteArray
}
```

Не-блокирующие и асинхронные API

```
interface NonBlocking {
    fun isDataEnough(): Boolean
    fun read(): ByteArray
}


interface Asynchronious {
    fun read(callback: (ByteArray) -> Unit)
}
```

Асинхронные API

```
fun read(callback: (ByteArray) -> Unit)

read { arr ->
    println(arr)
}
```

Асинхронные API

```
fun read(callback: (ByteArray) -> Unit)

read { arr ->
    println(arr)
}
```

Асинхронные API

```
fun read(callback: (ByteArray) -> Unit)
fun call(data: ByteArray, callback: (ByteArray) -> Unit)
fun save(data: ByteArray, callback: (ByteArray) -> Unit)

read { arr ->
    call(arr) { response ->
        save(response) {
            println("Done!")
        }
    }
}
```

Асинхронные API

```
fun read(callback: (ByteArray) -> Unit)
fun call(data: ByteArray, callback: (ByteArray) -> Unit)
fun save(data: ByteArray, callback: (ByteArray) -> Unit)
```

```
read { arr ->
    call(arr) { response ->
        save(response) {
            println("Done!")
        }
    }
}
```

Асинхронные API

```
typealias Callback<T> = (T, Exception) -> Unit
fun read(callback: Callback<ByteArray>)
fun call(data: ByteArray, callback: Callback<ByteArray>)
fun save(data: ByteArray, callback: Callback<ByteArray>)

read { arr, err ->
    if (err) {
        println(err)
    } else {
        call(...)
    }
}
```

Асинхронные API

```
typealias Callback<T> = (T, Exception) -> Unit
fun read(callback: Callback<ByteArray>)
fun call(data: ByteArray, callback: Callback<ByteArray>)
fun save(data: ByteArray, callback: Callback<ByteArray>)

read { arr, err ->
    if (err) {
        println(err)
    } else {
        call(...)
    }
}
```

Асинхронные API

```
typealias Callback<T> = (T, Exception) -> Unit
fun read(callback: Callback<ByteArray>)
fun call(data: ByteArray, callback: Callback<ByteArray>)
fun save(data: ByteArray, callback: Callback<ByteArray>)

read { arr, err ->
    if (err) {
        println(err)
    } else {
        call(...)
    }
}
```

Асинхронные API

+ Масшатбируемость

- Сложно читать и писать
- Отладка
- Вложенность
- Обработка ошибок

От колбеков к будущему

```
interface Future<V> {
    val isCancelled: Boolean
    val isDone: Boolean
    fun cancel(mayInterruptIfRunning: Boolean): Boolean
    fun get(): V
}
```

От колбеков к будущему

```
interface Future<V> {
    val isCancelled: Boolean
    val isDone: Boolean
    fun cancel(mayInterruptIfRunning: Boolean): Boolean
    fun get(): V
}
```

От колбеков к будущему

```
interface CompletionStage<T> {
    fun <U> thenApply(fn: Function<in T, out U>): CompletionStage<U>
    fun thenAccept(action: Consumer<in T>): CompletionStage<Void>
    fun <U> thenCompose(fn: Function<in T, out CompletionStage<U>>): CompletionStage<U>
    fun whenComplete(action: BiConsumer<in T, in Throwable>): CompletionStage<T>
}
```

От колбеков к будущему

```
class PerfectFuture<T>(
    resolve: ((T) -> Unit, (Throwable) -> Unit) -> Unit
)
```

```
(T) -> Unit
```

```
(Throwable) -> Unit
```

От колбеков к будущему

```
resolve: ((T) -> Unit, (Throwable) -> Unit) -> Unit

val perfect = PerfectFuture<ByteArray> { resolve, reject ->
    read { bytes, exception ->
        if (exception != null) {
            reject(exception)
        } else {
            resolve(bytes)
        }
    }
}
```

От колбеков к будущему

```
resolve: ((T) -> Unit, (Throwable) -> Unit) -> Unit
```

```
val perfect = PerfectFuture<ByteArray> { resolve, reject ->
    read { bytes, exception ->
        if (exception != null) {
            reject(exception)
        } else {
            resolve(bytes)
        }
    }
}
```

От колбеков к будущему

```
resolve: ((T) -> Unit, (Throwable) -> Unit) -> Unit

val perfect = PerfectFuture<ByteArray> { resolve, reject ->
    read { bytes, exception ->
        if (exception != null) {
            reject(exception)
        } else {
            resolve(bytes)
        }
    }
}
```

От колбеков к будущему

```
resolve: ((T) -> Unit, (Throwable) -> Unit) -> Unit

val perfect = PerfectFuture<ByteArray> { resolve, reject ->
    read { bytes, exception ->
        if (exception != null) {
            reject(exception)
        } else {
            resolve(bytes)
        }
    }
}
```

От колбеков к будущему

```
class PerfectFuture<T>(
    resolve: ((T) -> Unit, (Throwable) -> Unit) -> Unit
) {
    private var exception: Throwable? = null
    private var result: T? = null
    private val doneHandlers = mutableListOf<(T) -> Unit>()
    private val errorHandlers = mutableListOf<(Throwable) -> Unit>()

    init {
        resolve({ r ->
            result = r
            doneHandlers.onEach { it(r) }
        }, { e ->
            exception = e
            errorHandlers.onEach { it(e) }
        })
    }
}
```

От колбеков к будущему

```
class PerfectFuture<T>(
    resolve: ((T) -> Unit, (Throwable) -> Unit) -> Unit
) {
    private var exception: Throwable? = null
    private var result: T? = null
    private val doneHandlers = mutableListOf<(T) -> Unit>()
    private val errorHandlers = mutableListOf<(Throwable) -> Unit>()

    init {
        resolve({ r ->
            result = r
            doneHandlers.onEach { it(r) }
        }, { e ->
            exception = e
            errorHandlers.onEach { it(e) }
        })
    }
}
```

От колбеков к будущему

```
class PerfectFuture<T>(
    resolve: ((T) -> Unit, (Throwable) -> Unit) -> Unit
) {
    private var exception: Throwable? = null
    private var result: T? = null
    private val doneHandlers = mutableListOf<(T) -> Unit>()
    private val errorHandlers = mutableListOf<(Throwable) -> Unit>()

    fun onDone(handle: (T) -> Unit): PerfectFuture<T> {
        doneHandlers.add(handle)
        return this
    }

    fun onError(handle: (Throwable) -> Unit): PerfectFuture<T> {
        errorHandlers.add(handle)
        return this
    }
}
```

От колбеков к будущему

```
class PerfectFuture<T>(
    resolve: ((T) -> Unit, (Throwable) -> Unit) -> Unit
) {
    private var exception: Throwable? = null
    private var result: T? = null
    private val doneHandlers = mutableListOf<(T) -> Unit>()
    private val errorHandlers = mutableListOf<(Throwable) -> Unit>()

    fun onDone(handle: (T) -> Unit): PerfectFuture<T> {
        doneHandlers.add(handle)
        return this
    }

    fun onError(handle: (Throwable) -> Unit): PerfectFuture<T> {
        errorHandlers.add(handle)
        return this
    }
}
```

От колбеков к будущему

```
val perfect = PerfectFuture<ByteArray> { resolve, reject ->
    read { bytes, exception ->
        if (exception != null) {
            reject(exception)
        } else {
            resolve(bytes)
        }
    }
}

perfect.onDone { println("Done $it") }
    .onError { println("Error $it") }
```

От колбеков к будущему

```
val perfect = PerfectFuture<ByteArray> { resolve, reject ->
    read { bytes, exception ->
        if (exception != null) {
            reject(exception)
        } else {
            resolve(bytes)
        }
    }
}
```

```
perfect.onDone { println("Done $it") }
    .onError { println("Error $it") }
```

От колбеков к будущему

+ Нету огромной вложенности

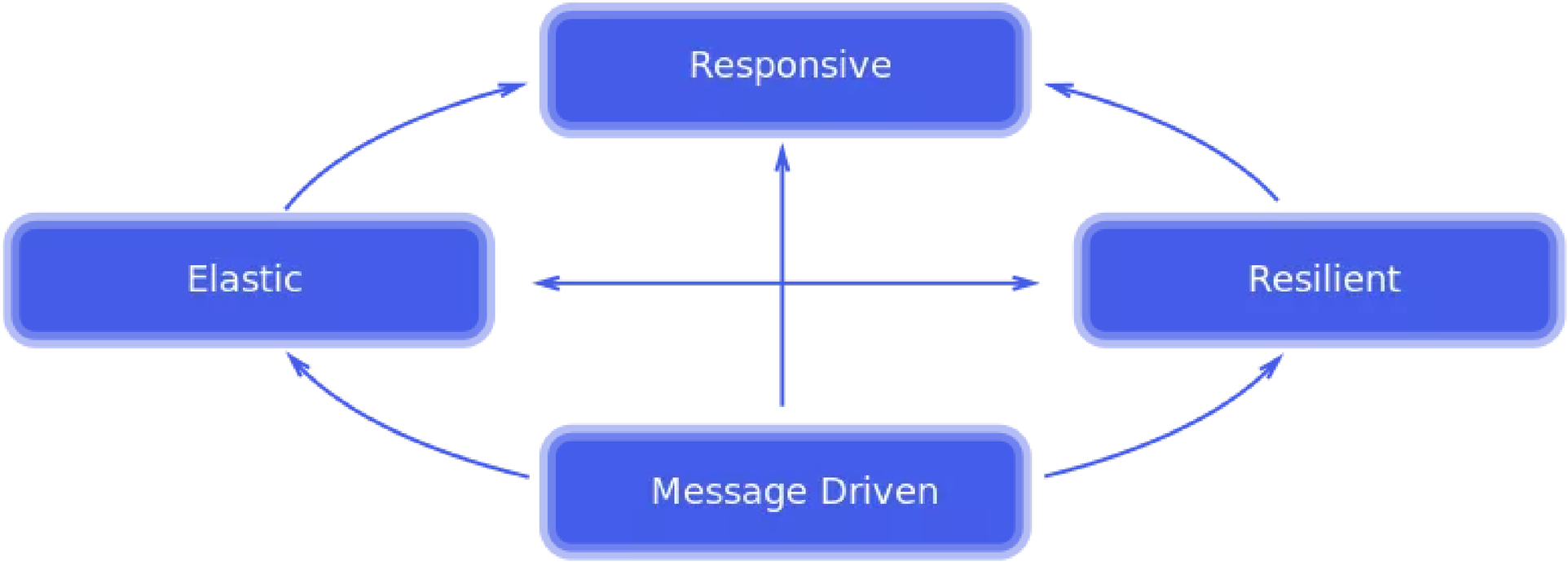
+ Легче обрабатывать ошибки

- Сложно читать и писать

- Отладка

- Везде типы Promise<T>

The Reactive Manifesto



Reactive Streams

```
val flux = Flux
    .range(1, 2)
    .map { 10 + it }
    .subscribeOn(Schedulers.parallel())
    .map { "value $it" }
    .subscribe { println(it) }
```

https://speakerdeck.com/simonbasle/projectreactor-dot-io-reactor3-intro

Reactive Streams

+ Работа с потоками данных

+ Бэкпреше

- Сложно читать и писать

- Отладка

- Везде типы Flux<T>

Async/Await

```
function doubleAfter2Seconds(x) {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve(x * 2);
    }, 2000);
  });
}
```

Async/Await

```
function addPromise(x) {
  return new Promise(resolve => {
    doubleAfter2Seconds(10).then((a) => {
      doubleAfter2Seconds(20).then((b) => {
        doubleAfter2Seconds(30).then((c) => {
          resolve(x + a + b + c);
        })
      })
    })
  });
}
```

Async/Await

```
function addPromise(x) {
  return new Promise(resolve => {
    doubleAfter2Seconds(10).then((a) => {
      doubleAfter2Seconds(20).then((b) => {
        doubleAfter2Seconds(30).then((c) => {
          resolve(x + a + b + c);
        })
      })
    })
  });
}
```

Async/Await

```
function addPromise(x) {
  return new Promise(resolve => {
    doubleAfter2Seconds(10).then((a) => {
      doubleAfter2Seconds(20).then((b) => {
        doubleAfter2Seconds(30).then((c) => {
          resolve(x + a + b + c);
        })
      })
    })
  });
}
```

Async/Await

```
function addPromise(x) {
  return new Promise(resolve => {
    doubleAfter2Seconds(10).then((a) => {
      doubleAfter2Seconds(20).then((b) => {
        doubleAfter2Seconds(30).then((c) => {
          resolve(x + a + b + c);
        })
      })
    })
  });
}
```

Async/Await

```
async function addAsync(x) {
  const a = await doubleAfter2Seconds(10);
  const b = await doubleAfter2Seconds(20);
  const c = await doubleAfter2Seconds(30);
  return x + a + b + c;
}
```

Async/Await

```
async function addAsync(x) {
  const a = await doubleAfter2Seconds(10);
  const b = await doubleAfter2Seconds(20);
  const c = await doubleAfter2Seconds(30);
  return x + a + b + c;
}
```

Async/Await

+ Читаемость

+ Обработка ошибок

- Легко забыть await - async по дефолту

- Построена вокруг определенных типов

BlogService

```
fun post(token: String, article: Article): Result {
    return try {
        val userId = authenticationService.getUserId(token)
        val user = userService.getUser(userId)
        articleService.add(user, article)
        Success( data: "New article created.")
    } catch (e: Exception) {
        LOGGER.log(Level.SEVERE, msg: "Can't create article.", e)
        Fail(e.message ?: "Can't create article.")
    }
}
```

AuthenticationService

```
@Service
class AuthenticationService(
    private val httpClient: HttpClient
) {
    fun getUserId(token: String): String {
        val request = HttpGet(URI("http://authService:8080/"))
        return httpClient.execute(request).entity.content.reader().readText()
    }
}
```

AuthenticationServi


```
@Service
class AuthenticationService(
    private val httpClient: HttpClient
) {
    fun getUserId(token: String): String {
        val request = HttpGet(URI("http://authService:8080/"))
        return httpClient.execute(request).entity.content.reader().readText()
    }
}
```

AuthenticationServi


```
@Service
class AuthenticationService(
    private val httpClient: HttpClient
) {
    fun getUserId(token: String): String {
        val request = HttpGet(URI("http://authService:8080/"))
        return httpClient.execute(request).entity.content.reader().readText()
    }
}
```

AuthenticationService (Callback)

```
@Service
class AuthenticationService(
    private val httpClient: HttpAsyncClient
) {
    fun getUserId(token: String, callback: (Result) -> Unit) {
        val request = HttpGet(URI("https://auth:8080/"))
        httpClient.execute(request, object : FutureCallback<HttpResponse> {
            override fun completed(result: HttpResponse) {
                callback(Success(result.entity.content.reader().readText()))
            }

            override fun failed(ex: Exception) {
                callback(Fail(ex.message ?: "Error fetching auth data."))
            }

            override fun cancelled() {
                callback(Fail("Request canceled."))
            }
        })
    }
}
```

AuthenticationService (Callback)

```
@Service
class AuthenticationService(
    private val httpClient: HttpAsyncClient
) {
    fun getUserId(token: String, callback: (Result) → Unit) {
        val request = HttpGet(URI("https://auth:8080/"))
        httpClient.execute(request, object : FutureCallback<HttpResponse> {
            override fun completed(result: HttpResponse) {
                callback(Success(result.entity.content.reader().readText()))
            }

            override fun failed(ex: Exception) {
                callback(Fail(ex.message ?: "Error fetching auth data."))
            }

            override fun cancelled() {
                callback(Fail("Request canceled."))
            }
        })
    }
}
```

BlogService

```
fun post(token: String, article: Article, callback: (Result<String>) → Unit) {
    authenticationService.getUserId(token) { authResult →
        when (authResult) {
            is Success<String> → {
                userService.getUser(authResult.data) { userResult →
                    when (userResult) {
                        is Success<User> → {
                            articleService.add(userResult.data, article) { articleResult →
                                callback(articleResult)
                            }
                        }
                        is Fail → callback(authResult)
                    }
                }
            }
            is Fail → callback(authResult)
        }
    }
}
```

AuthenticationService (Future)

```
class AuthenticationService(
    private val httpClient: HttpAsyncClient
) {
    fun getUserId(token: String): CompletableFuture<String> {
        val request = HttpGet( uri: "https://auth:8080/")
        val future = CompletableFuture<HttpResponse>()

        httpClient.execute(request, object : FutureCallback<HttpResponse> {
            override fun completed(result: HttpResponse) {
                future.complete(result)
            }

            override fun cancelled() {
                future.cancel( mayInterruptIfRunning: false)
            }

            override fun failed(ex: Exception) {
                future.completeExceptionally(ex)
            }
        })

        return future.thenApply { it.entity.content.reader().readText() }
    }
}
```

BlogService (Future)

```
fun post(token: String, article: Article): CompletableFuture<Result> {
    return authenticationService.getUserId(token)
        .thenCompose(userService::getUser)
        .thenCompose { user ->
            articleService.add(user, article)
        }
        .handle { u, e ->
            if (e != null) {
                Fail(e.message ?: "Can't create article.")
            } else {
                Success( data: "New article created.")
            }
        }
}
```

BlogService (Coroutines)

```
fun post(token: String, article: Article): Result {
    return try {
        val userId = authenticationService.getUserId(token)
        val user = userService.getUser(userId)
        articleService.add(user, article)
        Success( data: "New article created.")
    } catch (e: Exception) {
        LOGGER.log(Level.SEVERE, msg: "Can't create article.", e)
        Fail(e.message ?: "Can't create article.")
    }
}
```

BlogService (Coroutines)

```
suspend fun post(token: String, article: Article): Result {
    return try {
        val userId = authenticationService.getUserId(token)
        val user = userService.getUser(userId)
        articleService.add(user, article)
        Success( data: "New article created.")
    } catch (e: Exception) {
        LOGGER.log(Level.SEVERE, msg: "Can't create article.", e)
        Fail(e.message ?: "Can't create article.")
    }
}
```

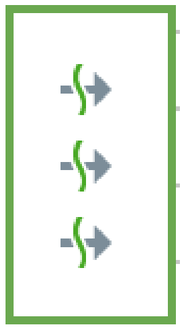
BlogService (Coroutines)

```
suspend fun post(token: String, article: Article): Result {
    return try {
        val userId = authenticationService.getUserId(token)
        val user = userService.getUser(userId)
        articleService.add(user, article)
        Success( data: "New article created.")
    } catch (e: Exception) {
        LOGGER.log(Level.SEVERE, msg: "Can't create article.", e)
        Fail(e.message ?: "Can't create article.")
    }
}
```

BlogService (Coroutines)

```
    suspend fun post(token: String, article: Article): Result {
        return try {
            val userId = authenticationService.getUserId(token)
            val user = userService.getUser(userId)
            articleService.add(user, article)
            Success( data: "New article created.")
        } catch (e: Exception) {
            LOGGER.log(Level.SEVERE, msg: "Can't create article.", e)
            Fail(e.message ?: "Can't create article.")
        }
    }
```

BlogService (Coroutines)



```
    suspend fun post(token: String, article: Article): Result {
        return try {
            val userId = authenticationService.getUserId(token)
            val user = userService.getUser(userId)
            articleService.add(user, article)
            Success( data: "New article created.")
        } catch (e: Exception) {
            LOGGER.log(Level.SEVERE, msg: "Can't create article.", e)
            Fail(e.message ?: "Can't create article.")
        }
    }
```

AuthenticationService (Coroutines)

```
class AuthenticationService(
    private val httpClient: HttpAsyncClient
) {
    suspend fun getUserId(token: String): String {
        val request = HttpGet( uri: "https://auth:8080/")

        return httpClient.execute(request).entity.content.reader().readText()
    }
}
```

AuthenticationService (Coroutines)

```
class AuthenticationService(
    private val httpClient: HttpAsyncClient
) {
    suspend fun getUserId(token: String): String {
        val request = HttpGet( uri: "https://auth:8080/")

        return httpClient.execute(request).entity.content.reader().readText()
    }
}
```

Wrapper for HttpAsyncClient

```
suspend fun HttpAsyncClient.execute(request: HttpUriRequest): HttpResponse {
    return suspendCancellableCoroutine { cont: CancellableContinuation<HttpResponse> →
        val future :Future<HttpResponse!>! = this.execute(request, object : FutureCallback<HttpResponse> {
            override fun completed(result: HttpResponse) {
                cont.resume(result)
            }

            override fun cancelled() {
                // Nothing
            }

            override fun failed(ex: Exception) {
                cont.resumeWithException(ex)
            }
        })

        cont.cancelFutureOnCompletion(future);
        Unit
    }
}
```

Wrapper for HttpAsyncClient

```
suspend fun HttpAsyncClient.execute(request: HttpUriRequest): HttpResponse {
    return suspendCancellableCoroutine { cont: CancellableContinuation<HttpResponse> →
        val future :Future<HttpResponse!>! = this.execute(request, object : FutureCallback<HttpResponse> {
            override fun completed(result: HttpResponse) {
                cont.resume(result)
            }

            override fun cancelled() {
                // Nothing
            }

            override fun failed(ex: Exception) {
                cont.resumeWithException(ex)
            }
        })

        cont.cancelFutureOnCompletion(future);
        Unit
    }
}
```

Wrapper for HttpAsyncClient

```
suspend fun HttpAsyncClient.execute(request: HttpUriRequest): HttpResponse {
    return suspendCancellableCoroutine { cont: CancellableContinuation<HttpResponse> →
        val future :Future<HttpResponse!>! = this.execute(request, object : FutureCallback<HttpResponse> {
            override fun completed(result: HttpResponse) {
                cont.resume(result)
            }

            override fun cancelled() {
                // Nothing
            }

            override fun failed(ex: Exception) {
                cont.resumeWithException(ex)
            }
        })

        cont.cancelFutureOnCompletion(future);
        Unit
    }
}
```

Wrapper for HttpAsyncClient

```
suspend fun HttpAsyncClient.execute(request: HttpUriRequest): HttpResponse {
    return suspendCancellableCoroutine { cont: CancellableContinuation<HttpResponse> →
        val future :Future<HttpResponse!>! = this.execute(request, object : FutureCallback<HttpResponse> {
            override fun completed(result: HttpResponse) {
                cont.resume(result)
            }

            override fun cancelled() {
                // Nothing
            }

            override fun failed(ex: Exception) {
                cont.resumeWithException(ex)
            }
        })

        cont.cancelFutureOnCompletion(future);
        Unit
    }
}
```

Wrapper for CompletableFuture

```
suspend fun <T> CompletableFuture<T>.await(): T {
    return suspendCancellableCoroutine { cont: CancellableContinuation<T> →
        this.whenComplete { t, u →
            if (u == null) {
                cont.resume(t)
            } else {
                cont.resumeWithException(u)
            }
        }
    }
}
```

Wrapper for Coroutines

```
suspend fun doWork(): String {
    delay( timeMillis: 1000)
    return "Hello, World!"
}

fun forJava(): CompletableFuture<String> {
    return GlobalScope.future { doWork() }
}
```

Coroutines

+ Читаемость
+ Обработка ошибок
+ Работает с любыми типами в обе стороны

- Без IDE непонятно, какой вызов
- Магически не фиксят блокирующий API

kotlinx.coroutines

Стандартная библиотека	**kotlinx.coroutines**
Встроена	Нужно ставить из maven
suspend, suspendCoroutine, startCoroutine, Continuation	launch, Dispatchers, runBlocking
Можно делать что угодно	Определенный стиль

launch

```
val job : Job = GlobalScope.launch {
    println("a")
    delay( timeMillis: 1000)
    println("b")
    delay( timeMillis: 1000)
    println("c")
}
```

Dispatchers

```
val job :Job = GlobalScope.launch(Dispatchers.Default) {
    println("a")
    delay( timeMillis: 1000)
    println("b")
    delay( timeMillis: 1000)
    println("c")
}
```

Dispatchers

```
val job: Job = GlobalScope.launch(Dispatchers.Default) {
    println("a")
    delay( timeMillis: 1000)
    println("b")
    delay( timeMillis: 1000)
    println("c")
}
```

withContext

```
val job: Job = GlobalScope.launch(Dispatchers.Default) {
    println("a")
    delay( timeMillis: 1000)
    withContext(Dispatchers.Main) { this: CoroutineScope
        println("b")
    }
    delay( timeMillis: 1000)
    println("c")
}
```

withContext

```
val job : Job = GlobalScope.launch(Dispatchers.Default) {
    println("a")
    delay( timeMillis: 1000)
    withContext(Dispatchers.Main) { this: CoroutineScope
        println("b")
    }
    delay( timeMillis: 1000)
    println("c")
}
```

CoroutineContext

- CoroutineName
- Job
- CoroutineExceptionHandler

Structured Concurrency

```
suspend fun loadAndCombine(name1: String, name2: String): Image {
    val deferred1 = async { loadImage(name1) }
    val deferred2 = async { loadImage(name2) }
    return combineImages(deferred1.await(), deferred2.await())
}
```

Structured Concurrency

```
suspend fun loadAndCombine(name1: String, name2: String): Image {
    val deferred1 = async { loadImage(name1) }
    val deferred2 = async { loadImage(name2) }
    return combineImages(deferred1.await(), deferred2.await())
}
```

Structured Concurrency

CoroutineScope

```
suspend fun loadAndCombine(name1: String, name2: String): Image =
    coroutineScope {
        val deferred1 = async { loadImage(name1) }
        val deferred2 = async { loadImage(name2) }
        combineImages(deferred1.await(), deferred2.await())
    }
```

Structured Concurrency

CoroutineScope

```
suspend fun loadAndCombine(name1: String, name2: String): Image =
    coroutineScope {
        val deferred1 = async { loadImage(name1) }
        val deferred2 = async { loadImage(name2) }
        combineImages(deferred1.await(), deferred2.await())
    }
```

Под капотом

```
// file.kt
package school

import kotlinx.coroutines.delay

suspend fun fetch() {
    delay( timeMillis: 1000)
}
```

Под капотом

```
FileKt.class
FileKt$fetch$1.class
```

Под капотом

```
public final class FileKt {
    public static final Object fetch(
            Continuation<? super Unit>
    );
}
```

Под капотом

```
public final class FileKt {
    public static final Object fetch(
        Continuation<? super Unit>
    );
}
```

Под капотом

```
public final class FileKt {
    public static final Object fetch(final Continuation<? super Unit> $continuation) {
        return new FileKt$fetch.FileKt$fetch$1((Continuation)$continuation)
        .doResume((Object)Unit.INSTANCE, (Throwable)null);
    }
}
```

Под капотом

```
public final class FileKt {
    public static final Object fetch(final Continuation<? super Unit> $continuation) {
        return new FileKt$fetch.FileKt$fetch$1((Continuation)$continuation)
        .doResume((Object)Unit.INSTANCE, (Throwable)null);
    }
}
```

Под капотом

```
final class FileKt$fetch$1 extends CoroutineImpl {
  public final Object doResume(Object, Throwable);
  FileKt$fetch$1(Continuation);
}
```

Под капотом

```
final class FileKt$fetch$1 extends CoroutineImpl {
  public final Object doResume(Object, Throwable);
  FileKt$fetch$1(Continuation);
}
```

Под капотом

```
static final class FileKt$fetch$1 extends CoroutineImpl {
    public final Object doResume(final Object data, @Nullable final Throwable throwable) {
        final Object coroutine_SUSPENDED = IntrinsicsKt.getCOROUTINE_SUSPENDED();
        switch (super.label) {
            case 0: {
        ...
            break;
            }
            case 1: {
                ...
                break;
            }
            default: {
                throw new IllegalStateException("call to 'resume' before 'invoke' with coroutine");
            }
        }
        return Unit.INSTANCE;
    }
}
```

Под капотом

Kotlin Coroutines – Deep Dive – Ильмир Усманов

https://bkug.by/2019/01/31/otchet-o-bkug-12/