Kotlin Coroutines

Asynchronous Programming Made Simple

Problem

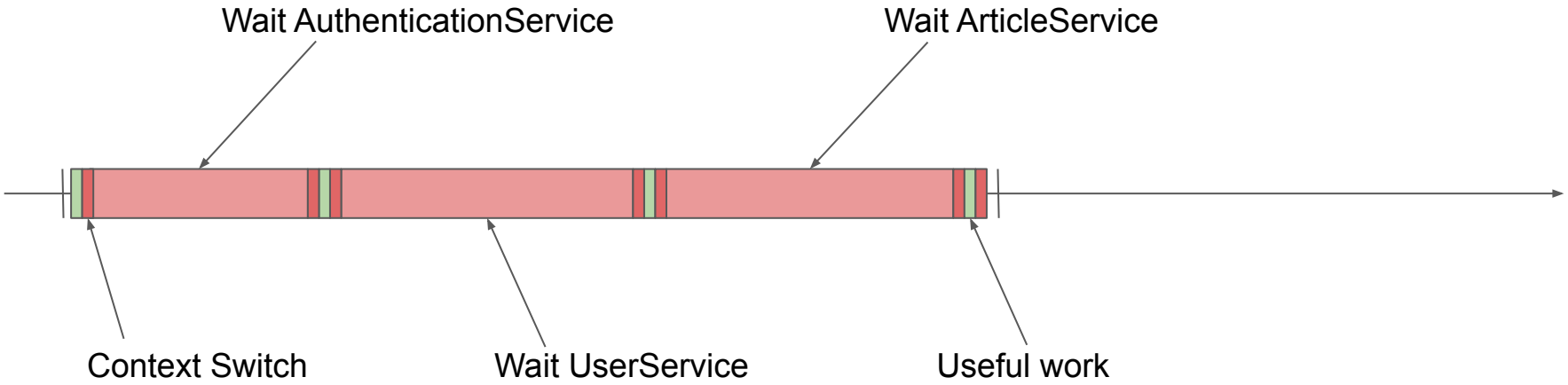
BlogService

```kotlin
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.")
    }
}
```

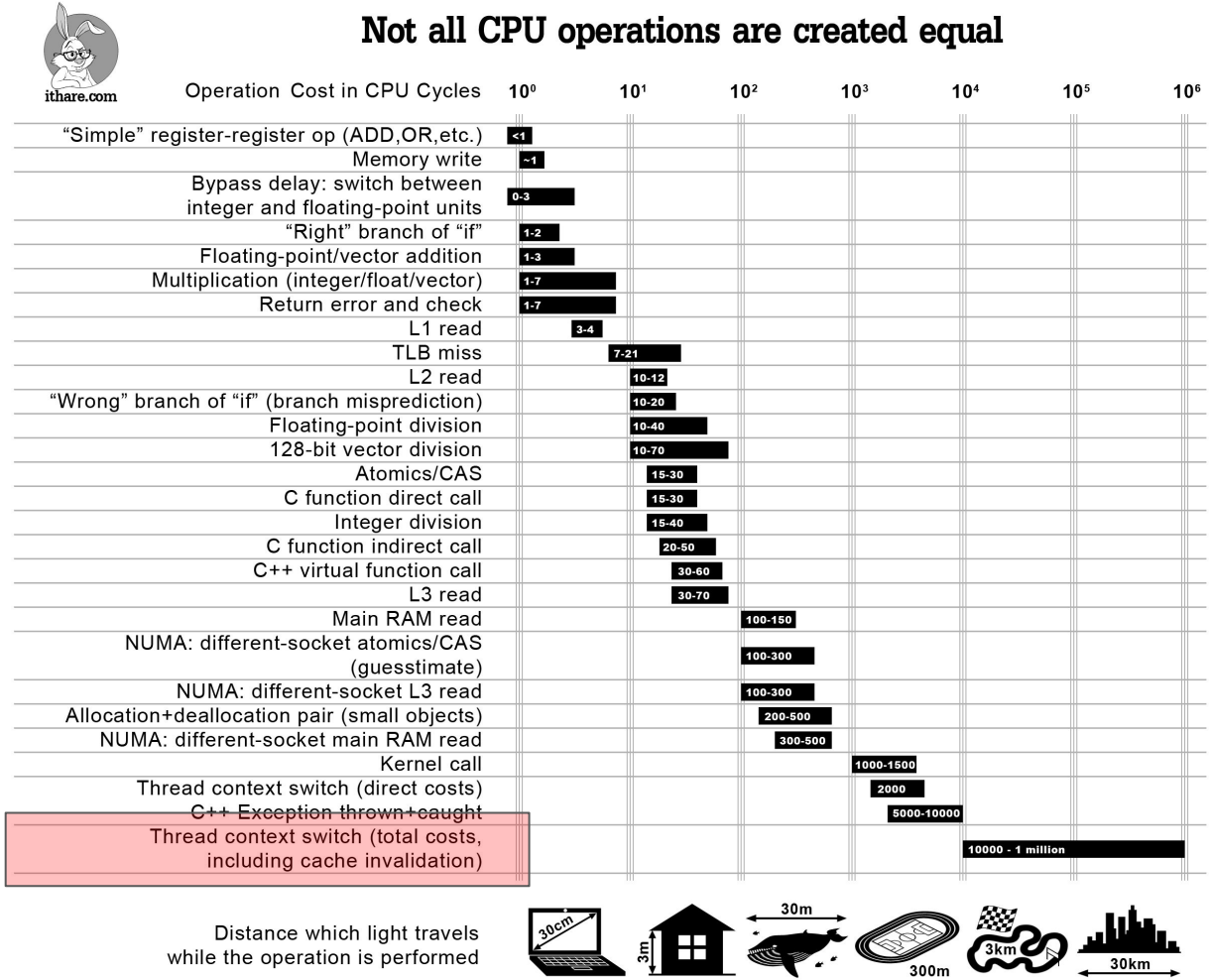
AuthenticationService

```
@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()
    }
}
```

BlogService - with Threads



Not all CPU operations are created equal



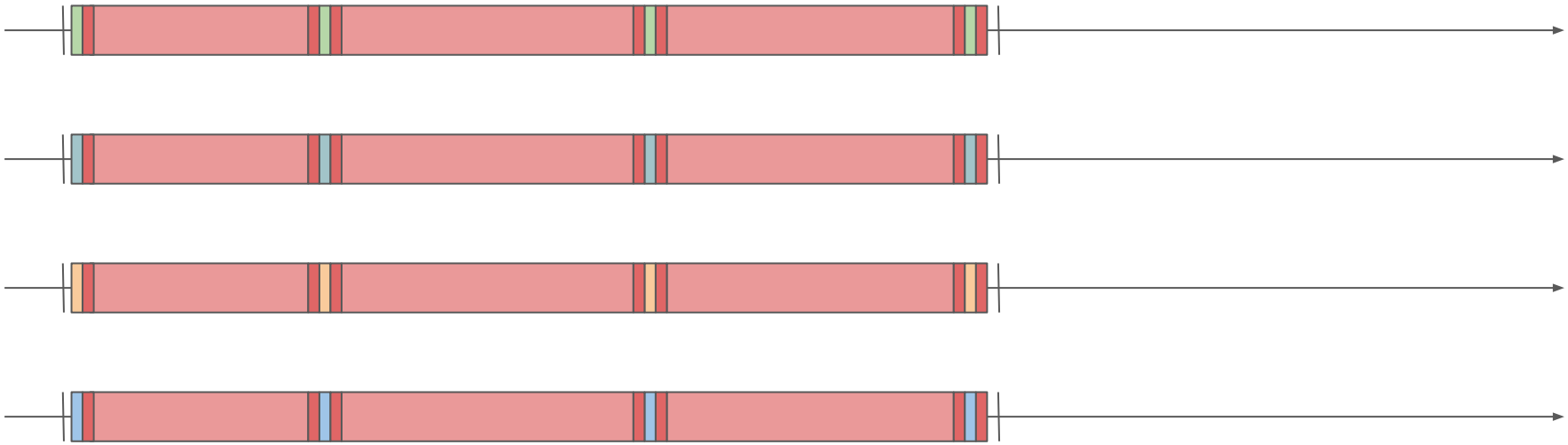
1M threads?

```
(1..1_000_000).forEach {
    thread(start = true) {
        println(it)
        Thread.sleep( millis: 1000L)
    }
}
```

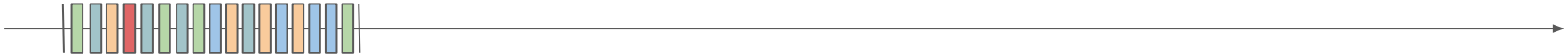
```
Exception in thread "main" java.lang.OutOfMemoryError: unable to create new native thread
    at java.lang.Thread.start0(Native Method)
    at java.lang.Thread.start(Thread.java:717)
    at kotlin.concurrent.ThreadsKt.thread(Thread.kt:30)
    at kotlin.concurrent.ThreadsKt.thread$default(Thread.kt:15)
    at by.heap.komodo.samples.coroutines.SuspendKt.main(Suspend.kt:40)
```

Intel Core i7-6700HQ, 32GB - **10k** Thread

Thread Model

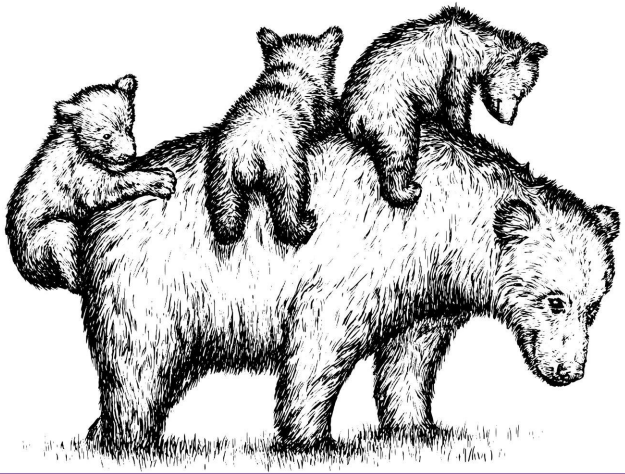


Async Model



Use Cases

Getting the wrong idea from that conference talk you attended



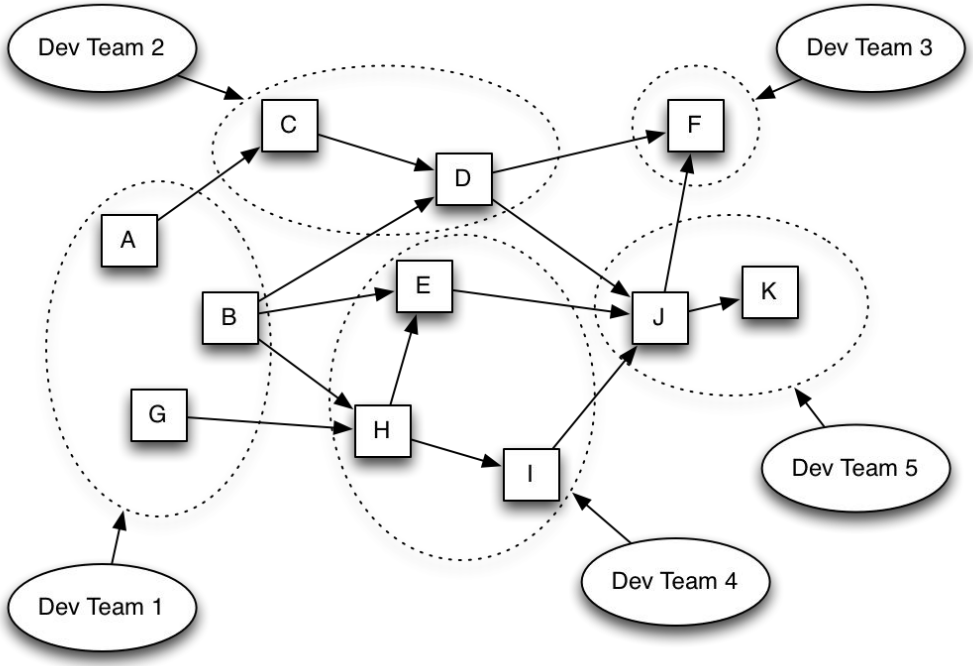
Solving Imaginary Scaling Issues

At Scale

O RLY?

@ThePracticalDev

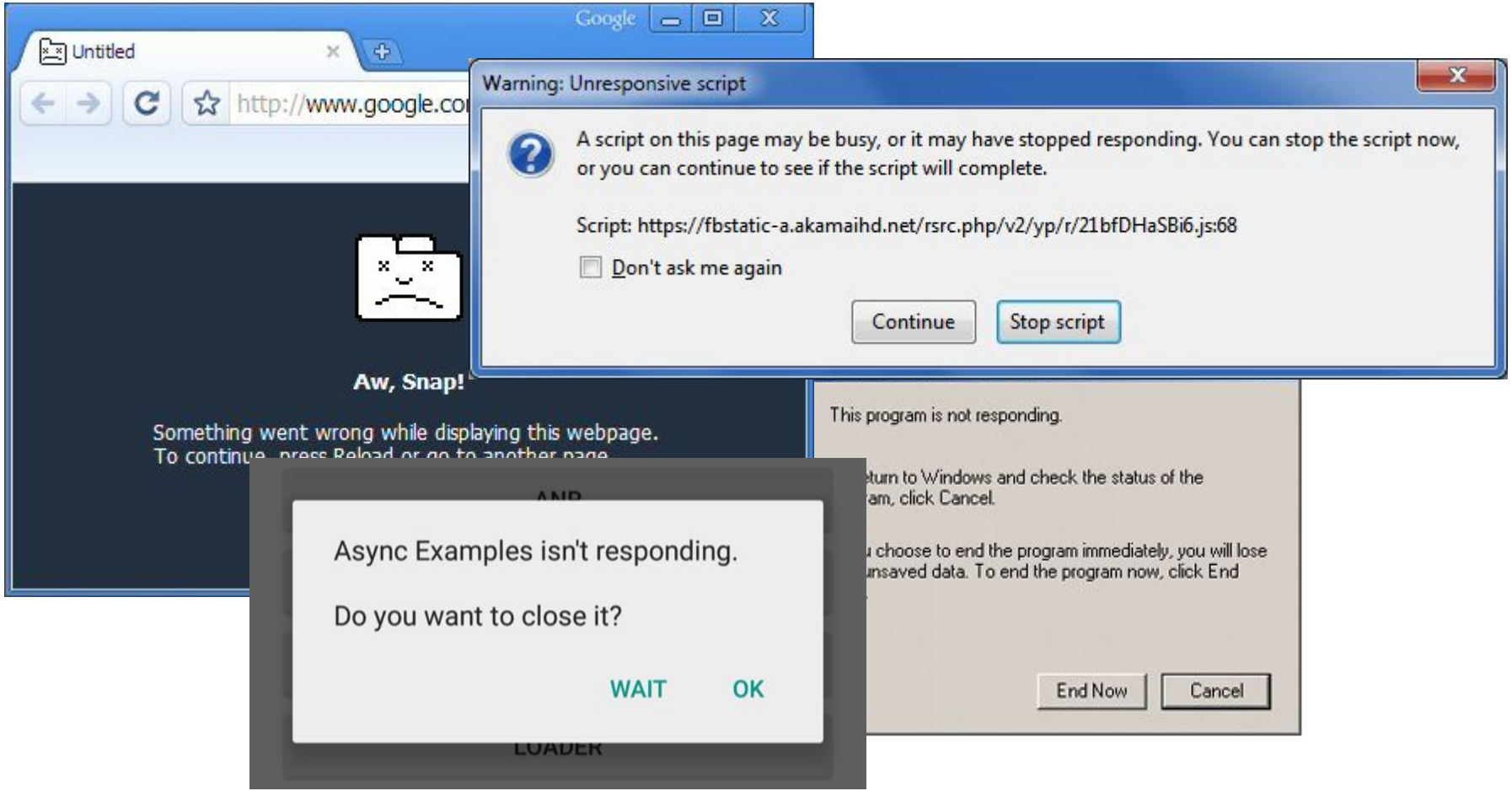
Microservices/SOA



Android (UIThread)

https://stackoverflow.com/questions/3652560/what-is-the-android-uithread-ui-thread#3653478:

The UIThread is the main thread of execution for your application. This is where **most of your application code** is run. All of your application components (Activities, Services, ContentProviders, BroadcastReceivers) are created in this thread, and any system calls to those components are performed in this thread.


Google
Untitled
http://www.google.co
Aw, Snap!
Something went wrong while displaying this webpage.
Warning: Unresponsive script
A script on this page may be busy, or it may have stopped responding. You can stop the script now, or you can continue to see if the script will complete.
Script: https://fbstatic-a.akamaihd.net/rsrc.php/v2/yp/r/21bfDHaSBi6.js:68
Don't ask me again
Continue
Stop script
This program is not responding.
End Now
Cancel
Async Examples isn't responding.
Do you want to close it?
WAIT
OK

How?

Callbacks

AuthenticationService

```kotlin
@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."))
            }
        })
    }
}
```

Callback HELL!

```
fun post(token: String, article:                                    t) {
    authenticationService.getUse
        when (authResult) {
            is Success<String> -
                userService.getU
                    when (userRe
                        is Succe
                            arti                                    articleResult →
                            }
                        }
                        is Fail
                    }
                }
            }
            is Fail → callback(
        }
    }
}
```

Futures

Futures

```
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.")
            }
        }
}
```

CompletableFuture

- compose;
- combine;
- handle;
- accept;
- apply;
- supply.

CompletableFuture.java

☐ Show inherited members (Ctrl+F12) ☐ Show Anonymous Classes (Ctrl+I)

☑ Show Lambdas (Ctrl+L)

```
runAsync(Runnable, Executor): CompletableFuture<Void>
screenExecutor(Executor): Executor
supplyAsync(Supplier<U>): CompletableFuture<U>
supplyAsync(Supplier<U>, Executor): CompletableFuture<U>
thenAccept(Consumer<? super T>): CompletableFuture<Void>
thenAcceptAsync(Consumer<? super T>): CompletableFuture<
thenAcceptAsync(Consumer<? super T>, Executor): Completab
thenAcceptBoth(CompletionStage<? extends U>, BiConsumer-
thenAcceptBothAsync(CompletionStage<? extends U>, BiCons
thenAcceptBothAsync(CompletionStage<? extends U>, BiCons
thenApply(Function<? super T, ? extends U>): CompletableFutu
thenApplyAsync(Function<? super T, ? extends U>): Completab
thenApplyAsync(Function<? super T, ? extends U>, Executor): C
thenCombine(CompletionStage<? extends U>, BiFunction<? su
thenCombineAsync(CompletionStage<? extends U>, BiFunctio
thenCombineAsync(CompletionStage<? extends U>, BiFunctio
thenCompose(Function<? super T, ? extends CompletionStage
thenComposeAsync(Function<? super T, ? extends Completion
thenComposeAsync(Function<? super T, ? extends Completion
thenRun(Runnable): CompletableFuture<Void> CompletionSta
thenRunAsync(Runnable): CompletableFuture<Void> Complet
thenRunAsync(Runnable, Executor): CompletableFuture<Void
timedGet(long): Object
toCompletableFuture(): CompletableFuture<T> CompletionSta
toString(): String Object
tryPushStack(Completion): boolean
uniAccept(CompletableFuture<S>, Consumer<? super S>, UniA
uniAcceptStage(Executor, Consumer<? super T>): Completable
uniApply(CompletableFuture<S>, Function<? super S, ? extend
uniApplyStage(Executor, Function<? super T, ? extends V>): Co
uniCompose(CompletableFuture<S>, Function<? super S, ? ext
uniComposeStage(Executor, Function<? super T, ? extends Cor
uniExceptionally(CompletableFuture<T>, Function<? super Thr
```

Coroutines

BlogService

```kotlin
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.")
    }
}
```

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.")
    }
}
```

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.")
    }
}
```

0 callbacks!*

* explicit

Continuation

```
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.")
    }
}
```

Coroutines: Hello, World!

```
compile("org.jetbrains.kotlinx:kotlinx-coroutines-core:0.16")
```

```
fun main(args: Array<String>) {
    delay( time: 1000)
    println("Hello, World!")
}
```

Error:(56, 5) Kotlin: Suspend function **'delay'** should be called only from a **coroutine** or another **suspend function**

```
fun main(args: Array<String>) {
    delay( time: 1000)
    println("Hello, World!")
}
```

Coroutines: Hello, World!

```
fun main(args: Array<String>) {
    val blogService = BlogService(
        AuthenticationService(),
        UserService(),
        ArticleService()
    )
    val result = blogService.post( token: "token", Article())

    // work with result
}
```

Coroutines: Hello, World!

```
fun main(args: Array<String>)  {
    launch(CommonPool) {
        delay(1000)
        println("Hello, World!")
    }
}

// Nothing
```

Coroutines: Hello, World!

```
fun main(args: Array<String>)  {
    launch(CommonPool) {
        delay(1000)
        println("Hello, World!")
    }

    Thread.sleep(2000)
}

// Hello, World!
```

Coroutines: Hello, World!

```
fun main(args: Array<String>)  {
   runBlocking {
       delay(1000)
       print("Hello, ")
   }

   print("World!")
}

// Hello, World!
```

Coroutines: Hello, World!

```kotlin
fun main(args: Array<String>) = runBlocking {
    val blogService = BlogService(
        AuthenticationService(),
        UserService(),
        ArticleService()
    )
    val result = blogService.post( token: "token", Article())

    // work with result
}
```

Coroutines + Spring

```
@RestController
class BlogController(
    val blogService: BlogService
) {

    @PostMapping("/{token}")
    fun createArticle(@PathVariable token: String, article: Article): CompletableFuture<String> {
        return future {
            val result = blogService.post(token, article)

            when (result) {
                is Success → result.data
                is Fail → throw RuntimeException(result.message)
            }
        }
    }
}
```

Suspend functions
Coroutines builders
kotlinx.coroutines

Coroutines & Kotlin

- suspend – language
- low-level core API: coroutine builders, etc – kotlin.coroutines (kotlin-stdlib)
- libraries – example: kotlinx.coroutines (kotlinx-coroutines-core)

Suspend

Suspending Functions

```
suspend fun delay(
    time: Long,
    unit: TimeUnit = TimeUnit.MILLISECONDS
) {
    // ...
}
```

Suspending Functions

```
suspend fun foo() {
    delay(1000)
}
```

```
16
17  suspend fun foo() {
18      delay( time: 1000)
19  }
20
```

Suspending Lambda

```
public fun launch(
    context: CoroutineContext,
    start: CoroutineStart = CoroutineStart.DEFAULT,
    block: suspend CoroutineScope.() -> Unit
): Job {
    …
}
```

Generators

buildSequence

kotlin.coroutines.experimental.

buildSequence

```
public fun <T> buildSequence(
    builderAction: suspend SequenceBuilder<T>.() -> Unit
): Sequence<T> = Sequence { buildIterator(builderAction) }
```

kotlin.coroutines.experimental.

buildSequence

```
val lazySeq: Sequence<Int> = buildSequence {
    for (i in 1..100) {
        yield(i)
    }
}

lazySeq.take(3).forEach { print(it) }
// 123
```

kotlin.coroutines.experimental.

buildSequence

```
val lazySeq: Sequence<Int> = buildSequence {
    for (i in 1..100) {
        delay(1000)
        yield(i)
    }
}
```

Error:(22, 9) Kotlin: Restricted suspending functions can only invoke member or extension suspending functions on their restricted coroutine scope

```
public fun <T> buildSequence(
    builderAction: suspend SequenceBuilder<T>.() -> Unit
): Sequence<T> = Sequence { buildIterator(builderAction) }
```

kotlin.coroutines.experimental.

SequenceBuilder

```
@RestrictsSuspension
public abstract class SequenceBuilder<in T> internal constructor() {
    public abstract suspend fun yield(value: T)
    public abstract suspend fun yieldAll(iterator: Iterator<T>)
    ...
}
```

kotlin.coroutines.experimental.

SequenceBuilder

```
suspend fun SequenceBuilder<Int>.answer() {
    this.yield(42)
}

val ultimateAnswerSeq: Sequence<Int> = buildSequence {
    while (true) {
        answer()
    }
}
```

buildIterator

Iterator<T>

buildSequence
buildIterator
@RestrictsSuspension

kotlinx.coroutines

kotlin.coroutines.experimental.

launch

```
public fun launch(
    context: CoroutineContext,
    start: CoroutineStart = CoroutineStart.DEFAULT,
    block: suspend CoroutineScope.() → Unit
): Job {
```

CoroutineContext

- Unconfined
- CommonPool
- newSingleThreadContext, newFixedThreadPoolContext
- Executor.asCoroutineDispatcher

CoroutineStart

```
CoroutineStart.DEFAULT → block.startCoroutineCancellable(completion)
CoroutineStart.ATOMIC → block.startCoroutine(completion)
CoroutineStart.UNDISPATCHED → block.startCoroutineUndispatched(completion)
CoroutineStart.LAZY → Unit // will start lazily
```

CoroutineScope

```
public interface CoroutineScope {
    public val isActive: Boolean
    public val context: CoroutineContext
}
```

kotlin.coroutines.experimental.

launch

```
fun main(args: Array<String>) = runBlocking<Unit> {
    val jobs = List( size: 100_000) {
        launch(CommonPool) {
            delay( time: 1000L)
            print(".")
        }
    }
    jobs.forEach { it.join() }
}
```

kotlinx.coroutines.experimental.**launch**

```
val job = launch(CommonPool) {
    while (isActive) {
        delay(100)
        println(42)
    }
}
job.cancel()
```

kotlinx.coroutines.experimental.NonCancellable

```
val job = launch(CommonPool) {
    try {
        // ...
    } finally {
        run(NonCancellable) {
            // this code isn't cancelled
        }
    }
}
job.cancel()
```

kotlin.coroutines.experimental.**async**

```
public fun <T> async(
    context: CoroutineContext,
    start: CoroutineStart = CoroutineStart.DEFAULT,
    block: suspend CoroutineScope.() → T
): Deferred<T> {
```

kotlin.coroutines.experimental.**async**

```kotlin
fun main(args: Array<String>) {
    val blogService = BlogService(
        AuthenticationService(),
        UserService(),
        ArticleService()
    )

    runBlocking {
        val result1 = async(Unconfined) { blogService.post( token: "token1", Article()) }
        val result2 = async(Unconfined) { blogService.post( token: "token2", Article()) }

        println("Result: ${result1.await()}, ${result2.await()}")
    }
}
```

kotlin.coroutines.experimental.**yield**

```
suspend fun foo() {
    list.forEach {
        /*
         ...
        compute relatively heavy task
         */
        yield()
    }
}
```

kotlinx.coroutines.experimental.(withTimeout/withTimeoutOrNull)

```
withTimeout(100) {
    request.await()
}

withTimeoutOrNull(100) {
    request.await()
}
```

Recursive Coroutines

```
suspend fun test() {
    println(Instant.now())
    test()
}

tailrec suspend fun test() {
    println(Instant.now())
    test()
}
```

Debug

-Dkotlinx.coroutines.debug

Thread.currentThread().name

[main @coroutine#2]
[main @coroutine#3]
[main @coroutine#1]

```
public fun newCoroutineContext(context: CoroutineContext):
CoroutineContext = if (DEBUG) context +
CoroutineId(COROUTINE_ID.incrementAndGet()) else context
```

Call Coroutines from Java

```
suspend fun foo(): Int {
    //...
}

fun fooJava(): CompletableFuture<Int> =
    future { foo() }
```

Not Covered

- Channels
- Select
- ...

Core API

createCoroutine
startCoroutine
suspendCoroutine
suspendCancellableCoroutine

Learn Kotlin Coroutines

- Guide to kotlinx.coroutines by example
- Coroutines for Kotlin
- #coroutines Kotlin Slack
- Андрей Бреслав — Асинхронно, но понятно. Сопрограммы в Kotlin
- Andrey Breslav — Kotlin Coroutines (JVMLS 2016, old coroutines!)
- Корутины в Kotlin - Роман Елизаров, JetBrains

Kotlin Coroutines

Kotlin 1.1: Experimental status.

Kotlin 1.2: ?

```
// build.gradle
kotlin {
    experimental {
        coroutines 'enable'
    }
}
```

kotlin.coroutines.experimental -> kotlin.coroutines

Q&A

Ruslan Ibragimov @HeapyHop

Belarus Kotlin User Group: https://bkug.by/

Java Professionals BY: http://jprof.by/

Awesome Kotlin: https://kotlin.link/

Slides: https://goo.gl/5sJXeH

Shared mutable state and concurrency

- Thread-safe data structures (Atomics)
- Thread confinement fine-grained
- Thread confinement coarse-grained
- Mutual exclusion (suspending)
- Actors
- Read more

Core API

createCoroutine

kotlin.coroutines.experimental.**createCoroutine**

```
public fun <R, T> (suspend R.() -> T).createCoroutine(
        receiver: R,
        completion: Continuation<T>
): Continuation<Unit> = SafeContinuation(
        createCoroutineUnchecked(receiver, completion),
        COROUTINE_SUSPENDED
)
```

```
block.createCoroutine(receiver, completion)
```

```
launch(CommonPool) {
    delay(1000)
    println("Hello, World!")
}
```

Bytecode

```
package by.heap.komodo.samples.coroutines.bytecode

import kotlinx.coroutines.experimental.delay

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

Bytecode

```
-rw-r--r-- 1 yoda yoda 1342 Jun  1 08:03 ExampleKt.class
-rw-r--r-- 1 yoda yoda 1833 Jun  1 08:03 ExampleKt$fetch$1.class
```

Bytecode

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

Bytecode

```
public final class ExampleKt {
    @Nullable
    public static final Object fetch(@NotNull final Continuation<? super
Unit> $continuation) {
    Intrinsics.checkParameterIsNotNull((Object)$continuation,
"$continuation");
    return new
ExampleKt$fetch.ExampleKt$fetch$1((Continuation)$continuation).doResume((Ob
ject)Unit.INSTANCE, (Throwable)null);
    }
}
```

Bytecode

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

Bytecode

```
static final class ExampleKt$fetch$1 extends CoroutineImpl {
      @Nullable
      public final Object doResume(@Nullable 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;
      }
}
```

startCoroutine

kotlin.coroutines.experimental.

startCoroutine

```
public fun <R, T> (suspend R.() -> T).startCoroutine(
        receiver: R,
        completion: Continuation<T>
) {
    createCoroutineUnchecked(receiver, completion).resume(Unit)
}
```

```
block.startCoroutine(receiver, completion)
```

suspendCoroutine

kotlin.coroutines.experimental.**suspendCoroutine**

```
public inline suspend fun <T> suspendCoroutine(
    crossinline block: (Continuation<T>) -> Unit
): T = suspendCoroutineOrReturn { c: Continuation<T> ->
    val safe = SafeContinuation(c)
    block(safe)
    safe.getResult()
}
```

kotlin.coroutines.experimental.**suspendCoroutine**

```
suspend fun <T> CompletableFuture<T>.await(): T =
    suspendCoroutine { cont: Continuation<T> ->
        whenComplete { result, exception ->
            if (exception == null)
                cont.resume(result)
            else
                cont.resumeWithException(exception)
        }
    }
```

suspendCancellableCoroutine

kotlin.coroutines.experimental.suspendCancellableCoroutine

```
public inline suspend fun <T> suspendCancellableCoroutine(
    holdCancellability: Boolean = false,
    crossinline block: (CancellableContinuation<T>) -> Unit
): T = suspendCoroutineOrReturn { cont ->
    val cancellable = CancellableContinuationImpl(cont, active = true)
    if (!holdCancellability) cancellable.initCancellability()
    block(cancellable)
    cancellable.getResult()
}
```

kotlin.coroutines.experimental.

suspendCancellableCoroutine

```
suspend fun <T> CompletableFuture<T>.await(): T =
    suspendCancellableCoroutine { cont: CancellableContinuation<T> ->
        whenComplete { result, exception ->
            if (exception == null)
                cont.resume(result)
            else
                cont.resumeWithException(exception)
        }
        cont.invokeOnCompletion { this.cancel(false) }
    }
```

suspend
createCoroutine
startCoroutine
suspendCoroutine
suspendCancellableCoroutine