Cancellation and Exceptions

Kevin Jones

@kevinrjones www.rocksolidknowledge.com

Cancelling Coroutines

Coroutines can have a parent-child relationship

- Cancelling parent cancels children
- Cancelling children does not cancel siblings

Important to understand these relationships

- See how these are created

Job Hierarchies

Jobs may have parents

Jobs my have siblings

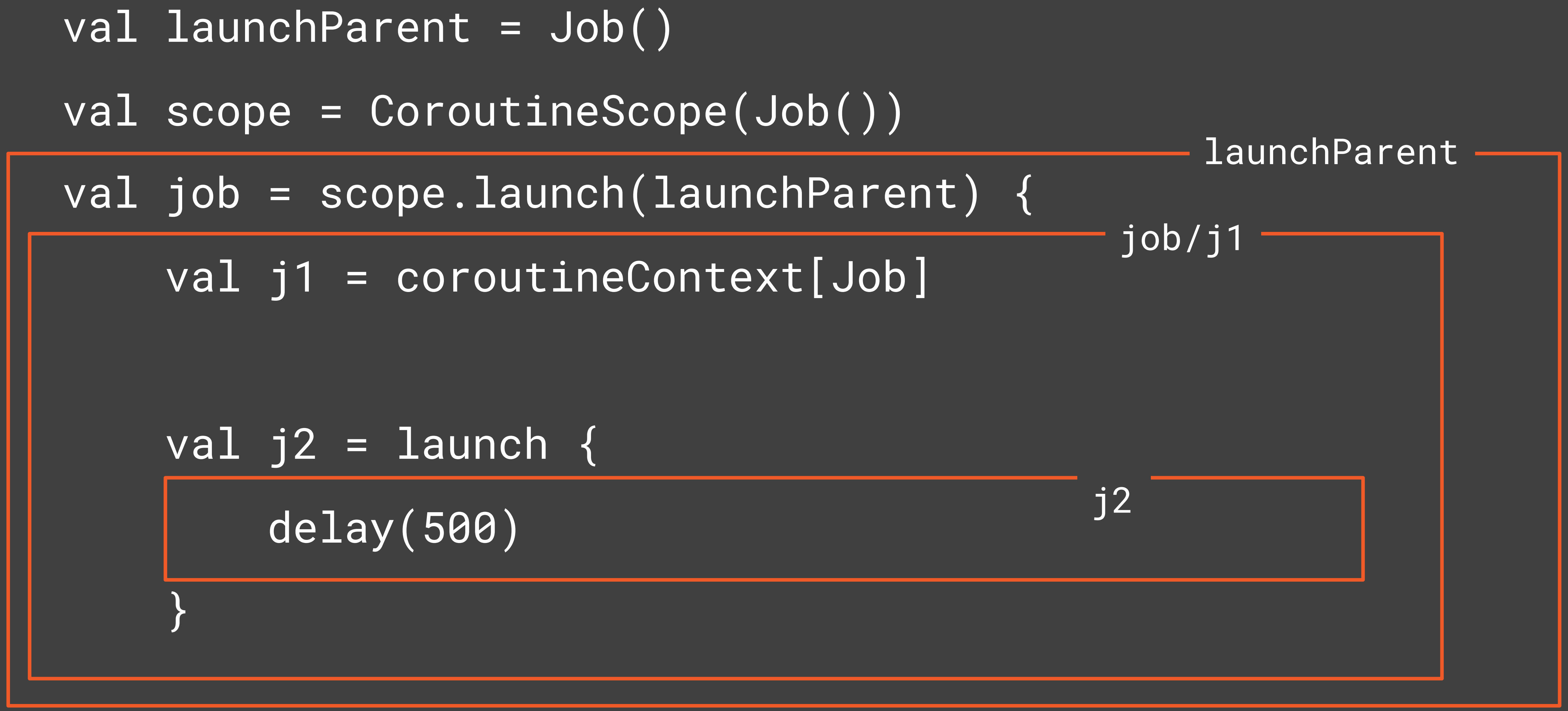
There is no 'parent' property on 'Job'

- There is a 'children' property

Demo

Jobs

Job Hierarchy



```
val launchParent = Job()

val scope = CoroutineScope(Job())

val job = scope.launch(launchParent) {

    val j1 = coroutineContext[Job]


    val j2 = launch {

        delay(500)

    }
```

Cancellation

Cancelling parent cancels children

Cancelling child does not cancel siblings

- Or parent

Cancellation is co-operative

Cancellation throws a specific exception

Be careful with async/await

How Do You Co-operate

To co-operate use 'isActive'

Can also use 'ensureActive()'

Demo

Cancelling through co-operation

Cancellation Throws Exceptions

Suspending functions throw exception when cancelled

- CancellationException

Need to close resources in our code

May need to run suspending function in finally

- Will throw CancellationException
- Needs to execute in a special context

Can Specify the Exception

Can be used to specify the reason

- job.cancel(CancellationException("why"))

Can specify any exception

- Must derive from CancellationException
- job.cancel(SomeExceptionType())

More on exceptions later

Demo

Handling cancellation exceptions

Be Careful With 'await'

'Job' will cancel successfully or complete

'Deferred' could throw an exception in 'await'

- If deferred has already been cancelled

Demo

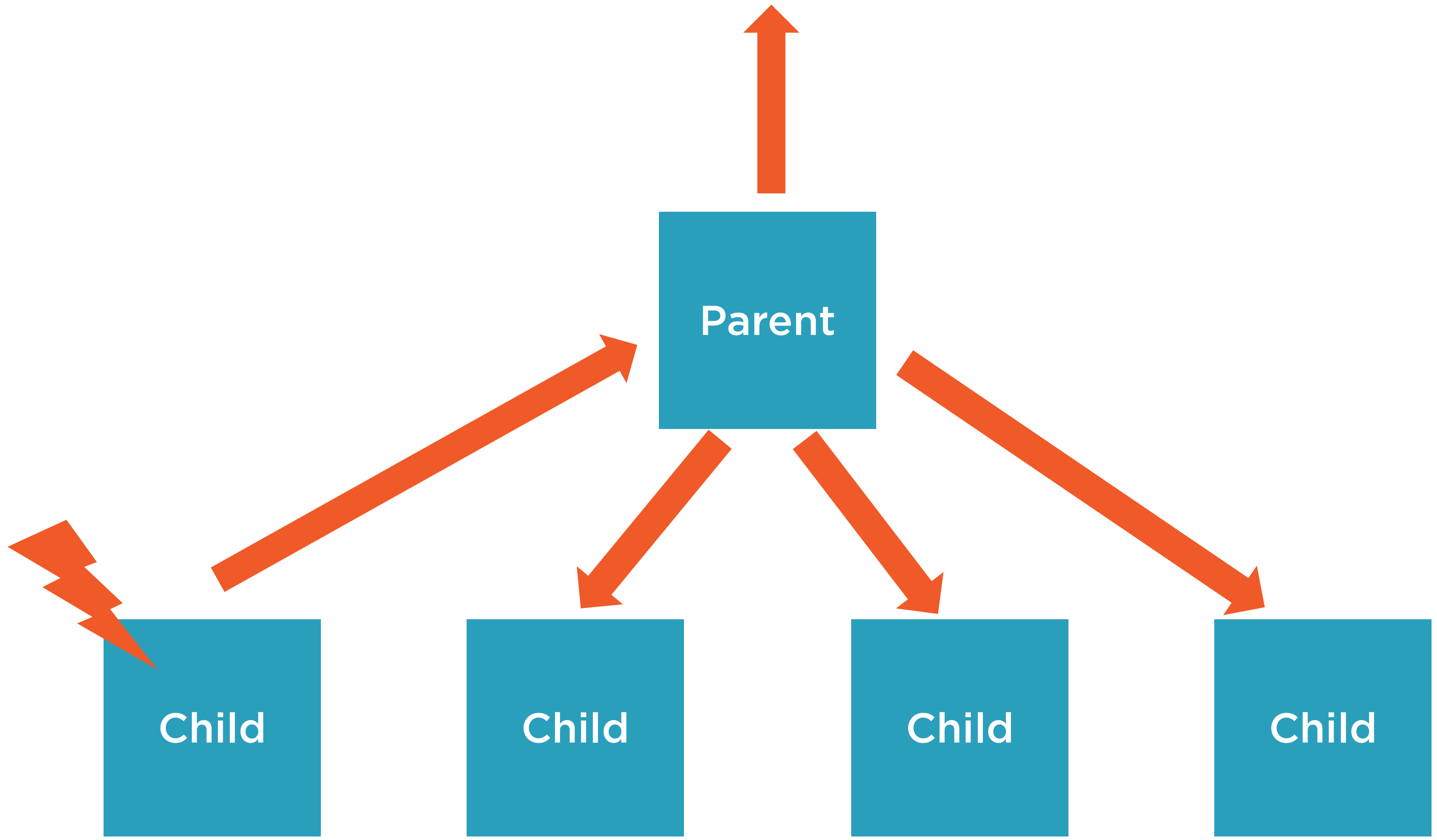
Cancelling 'Deferred'

Exceptions

What happens when an exception is thrown?

How do we manage them?

Thrown Exceptions



Demo

Exception cancelling jobs

What if You Don't Want This Behavior?

Use SupervisorJob or SupervisorScope

- Some subtleties around this

SupervisorJob has to be the direct parent

Demo

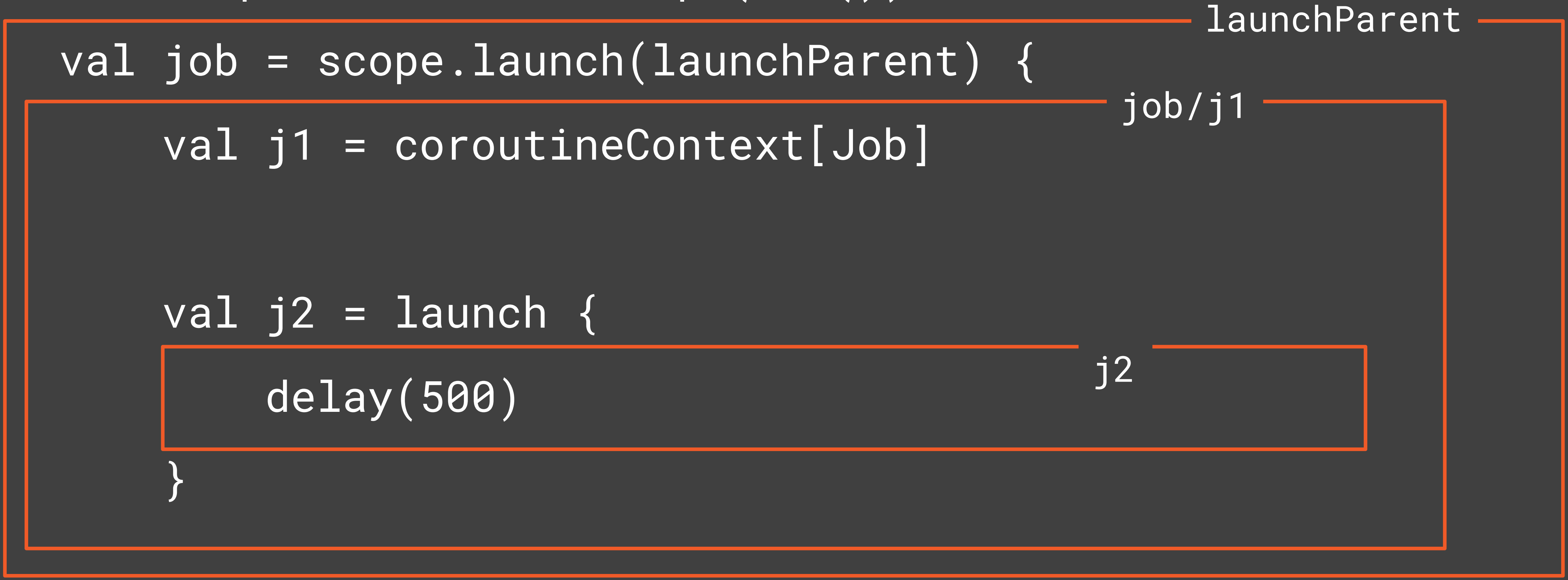
SupervisorJob and supervisorScope

Why Does That Code Not Work?

Remember this Picture?

```
val launchParent = SupervisorJob()

val scope = CoroutineScope(Job())
```



```
val job = scope.launch(launchParent) {
    val j1 = coroutineContext[Job]


    val j2 = launch {
        delay(500)
    }
```

Demo

Using SupervisorJob

Demo

Using SupervisorScope

To Use 'supervisor' or Not?

Job/coroutineScope

- Child is cancelled
- Parent is cancelled
- Siblings are cancelled

SuperviorJob/supervisorScope

- Child is cancelled
- Parent is not cancelled
- Siblings are not cancelled

Cancelling Scope

When a scope is cancelled

- Cannot start further job in that scope
- UI might freeze

Unhandled Exceptions

Notice that in the last example the exception is still reported

- Exceptions are always propogated up
- There is a default 'CoroutineExceptionHandler' in the context
- On the JVM reports the exception
- In Android it kills the process
- Can replace it

Demo

CoroutineExceptionHandler

async Coroutine Builder

'async' behaves differently

If async is root coroutine

- Exception is thrown on 'await'

If async is a child coroutine

- Will propagate the exception immediately

Demo

Exceptions in async

Summary

Jobs exist in a hierarchy

- Cancelling parent cancels children
- But not vice versa

Cancellation

- Is co-operative
- Throws CancellationException

Exceptions are passed from child to parent

- Depending on the scope may cancel siblings
- Are always propagated up
- Be careful of 'async' behaviour

What’s Next