KotlinConf

Kotlin Coroutines in Practice

Roman Elizarov
elizarov@
@relizarov

JET BRAINS

KotlinConf

Coroutines recap

Coroutines are like light-weight threads

Coroutines are like light-weight threads

```
suspend fun main() {
    val jobs = List(100_000) {
        GlobalScope.launch {
            delay(5000)
            print(".")
        }
    }
    jobs.forEach { it.join() }
}
```

Coroutines are like light-weight threads

```
suspend fun main() {
    val jobs = List(100_000) {
        GlobalScope.launch {
            delay(5000)
            print(".")
        }
    }
    jobs.forEach { it.join() }
}
```

Coroutines are like light-weight threads

```
suspend fun main() {
    val jobs = List(100_000) {
        GlobalScope.launch {
            delay(5000)
            print(".")
        }
    }
    jobs.forEach { it.join() }
}
```

Quantity

数量

结论:数量 != 质量

质量

~~Quantity~~ → Quality

A practical challenge

```
suspend fun downloadContent(location: Location): Content
```

```
fun processReferences(refs: List<Reference>)
```

References

```
fun processReferences(refs: List<Reference>) {
    for (ref in refs) {
        …
    }
}
```

References

```
fun processReferences(refs: List<Reference>) {
    for (ref in refs) {
        val location = ref.resolveLocation()
        …

    }
}
```

```
fun processReferences(refs: List<Reference>) {
    for (ref in refs) {
        val location = ref.resolveLocation()
        val content = downloadContent(location)
        processContent(ref, content)
    }
}
```

```
suspend fun processReferences(refs: List<Reference>) {
    for (ref in refs) {
        val location = ref.resolveLocation()
        val content = downloadContent(location)
        processContent(ref, content)
    }
}
```

Sequential

```
suspend fun processReferences(refs: List<Reference>) {
    for (ref in refs) {
        val location = ref.resolveLocation()
        val content = downloadContent(location)
        processContent(ref, content)
    }
}
```

```
fun processReferences(refs: List<Reference>) {
    for (ref in refs) {
        val location = ref.resolveLocation()
        val content = downloadContent(location)
        processContent(ref, content)
    }
}
```

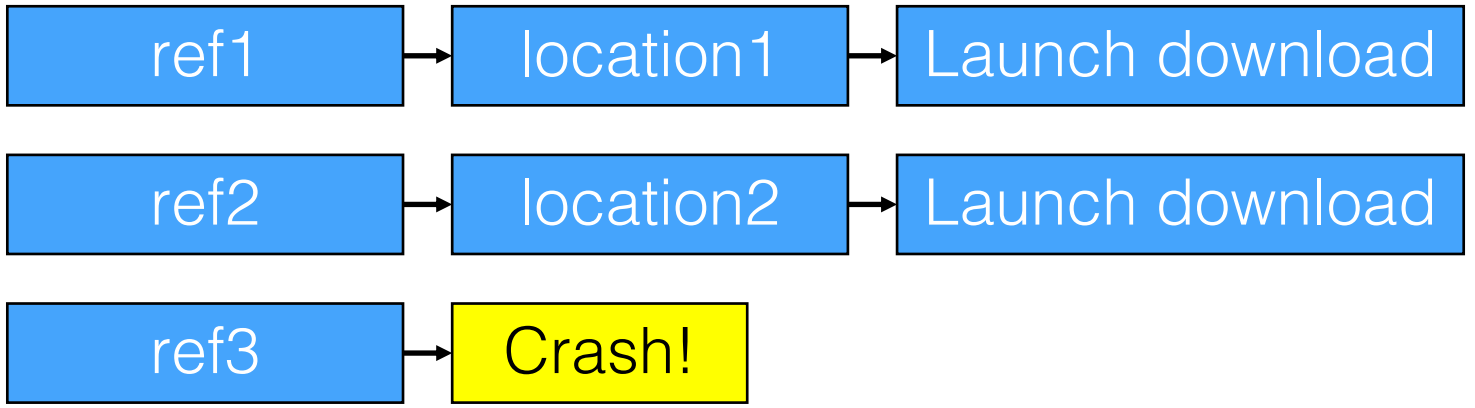
```
fun processReferences(refs: List<Reference>) {
    for (ref in refs) {
        val location = ref.resolveLocation()
        GlobalScope.launch {
            val content = downloadContent(location)
            processContent(ref, content)
        }
    }
}
```

```
fun processReferences(refs: List<Reference>) {
    for (ref in refs) {
        val location = ref.resolveLocation()
        GlobalScope.launch {
            val content = downloadContent(location)
            processContent(ref, content)
        }
    }
}
```

Coroutines are cheap! What could go wrong?

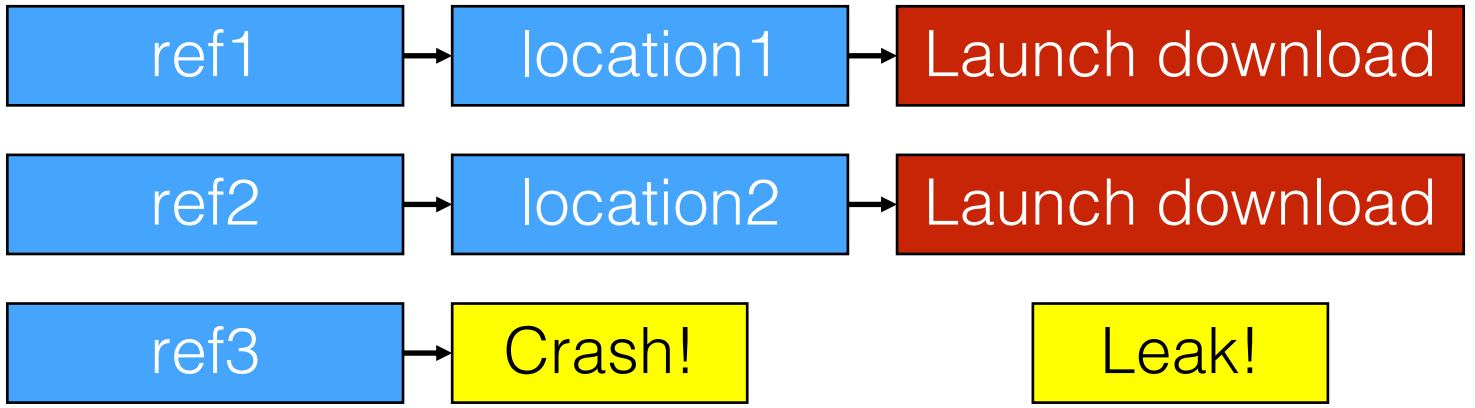
```
fun processReferences(refs: List<Reference>) {
    for (ref in refs) {
        val location = ref.resolveLocation()
        GlobalScope.launch {
            val content = downloadContent(location)
            processContent(ref, content)
        }
    }
}
```

Crash!



```
fun processReferences(refs: List<Reference>) {
    for (ref in refs) {
        val location = ref.resolveLocation()
        GlobalScope.launch {
            val content = downloadContent(location)
            processContent(ref, content)
        }
    }
}
```

Crash!



KotlinConf

Structured concurrency

```
fun processReferences(refs: List<Reference>)
```

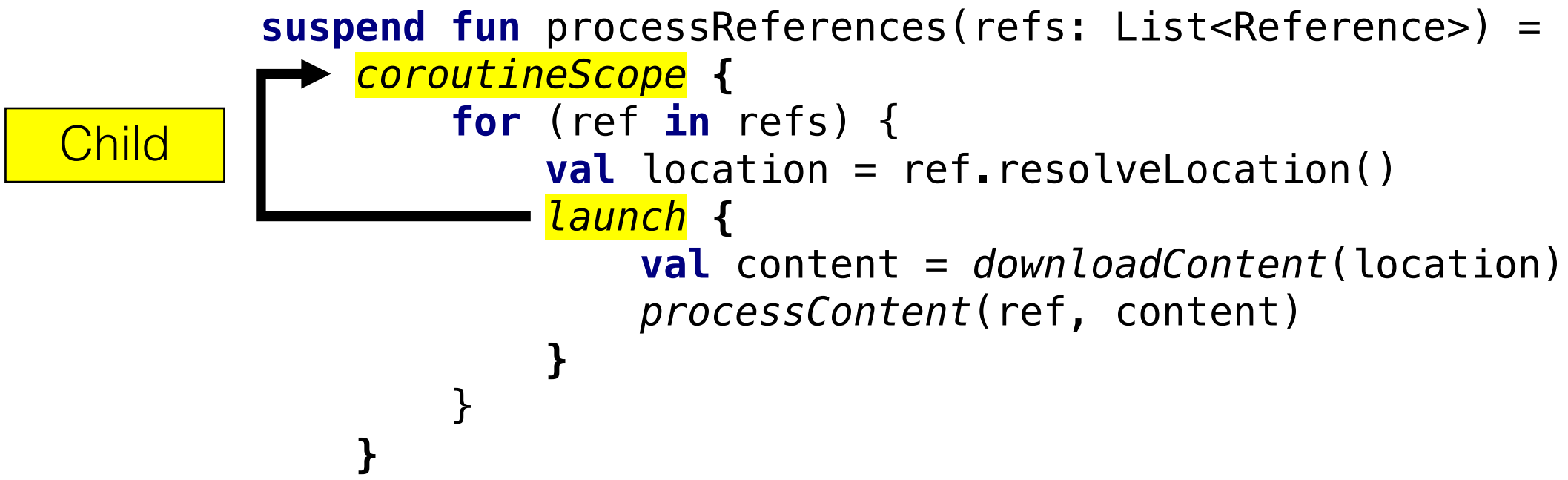
```
suspend fun processReferences(refs: List<Reference>)
```

```
suspend fun processReferences(refs: List<Reference>) =
    coroutineScope {
        …
    }
```

```
suspend fun processReferences(refs: List<Reference>) =
    coroutineScope {
        for (ref in refs) {
            val location = ref.resolveLocation()
            GlobalScope.launch {
                val content = downloadContent(location)
                processContent(ref, content)
            }
        }
    }
```

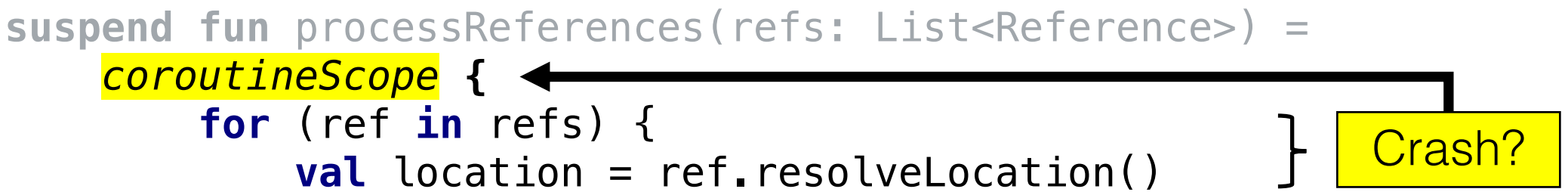
删除全局范围采用结构范围

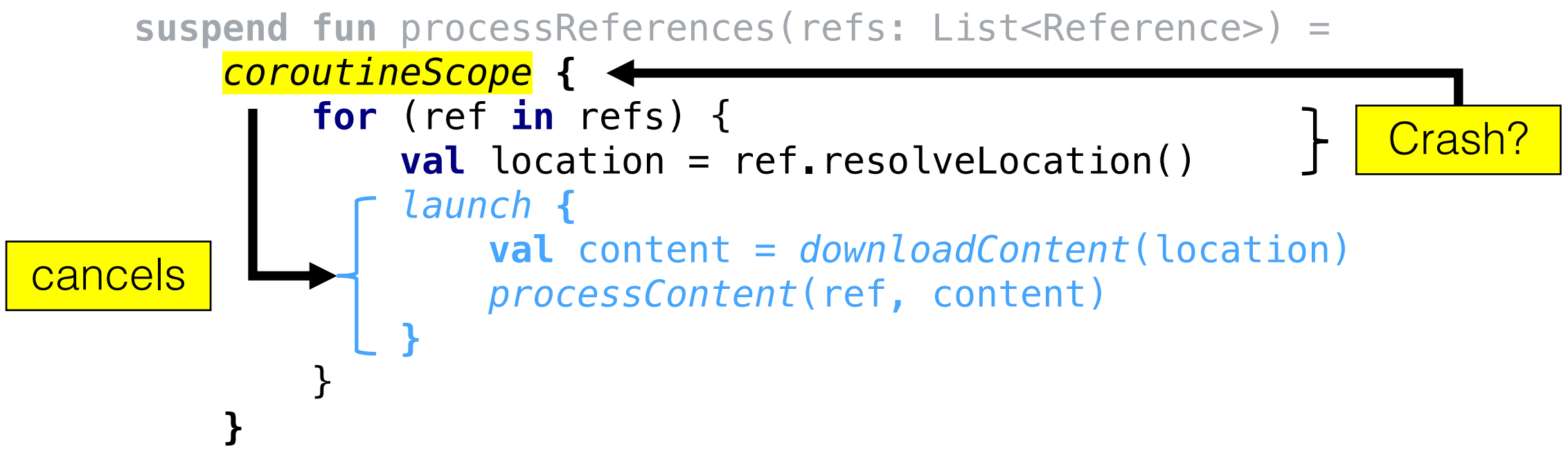
```
suspend fun processReferences(refs: List<Reference>) =
    coroutineScope {
        for (ref in refs) {
            val location = ref.resolveLocation()
            launch {
                val content = downloadContent(location)
                processContent(ref, content)
            }
        }
    }
```

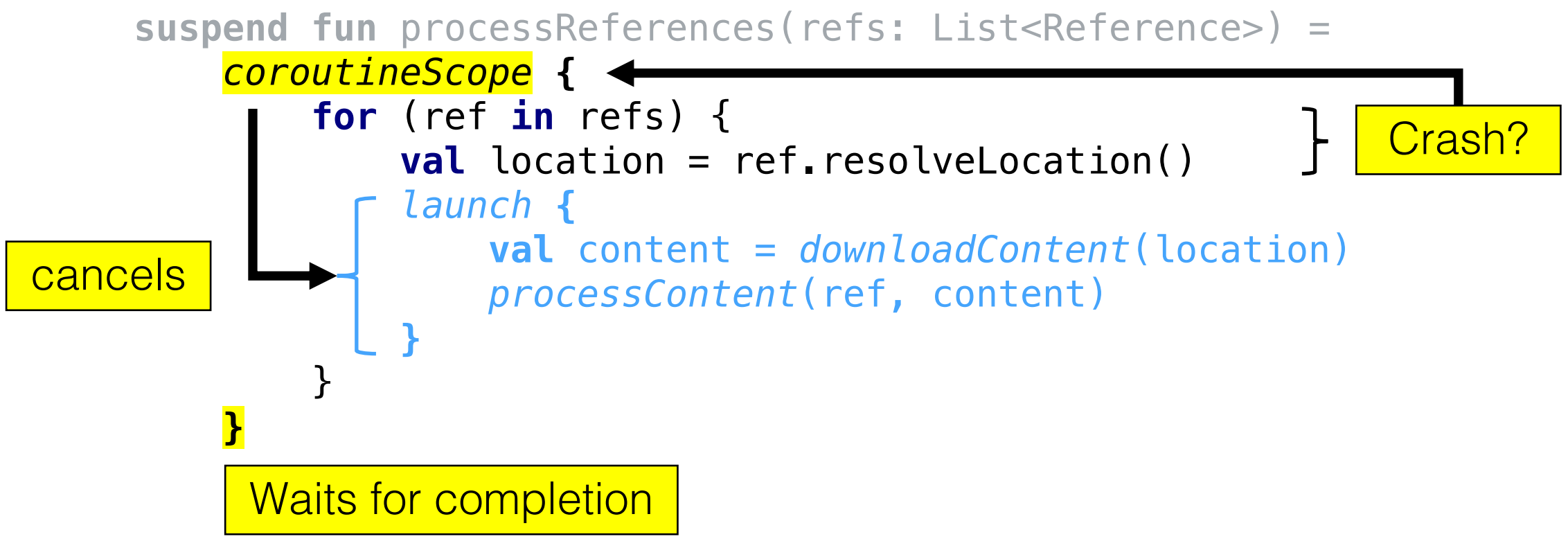

Child
suspend fun processReferences(refs: List<Reference>) =
coroutineScope {
for (ref in refs) {
val location = ref.resolveLocation()
launch {
val content = downloadContent(location)
processContent(ref, content)
}
}
}

```
suspend fun processReferences(refs: List<Reference>) =
    coroutineScope {
        for (ref in refs) {
            val location = ref.resolveLocation()
            launch {
                val content = downloadContent(location)
                processContent(ref, content)
            }
        }
    }
```

Crash?


suspend fun processReferences(refs: List<Reference>) =
coroutineScope {
for (ref in refs) {
val location = ref.resolveLocation()
launch {
val content = downloadContent(location)
processContent(ref, content)
}
}
}
Crash?


suspend fun processReferences(refs: List<Reference>) =
coroutineScope {
for (ref in refs) {
val location = ref.resolveLocation()
launch {
val content = downloadContent(location)
processContent(ref, content)
}
}
}
cancels
Crash?


suspend fun processReferences(refs: List<Reference>) =
coroutineScope {
for (ref in refs) {
val location = ref.resolveLocation()
launch {
val content = downloadContent(location)
processContent(ref, content)
}
}
}
Crash?
cancels
Waits for completion

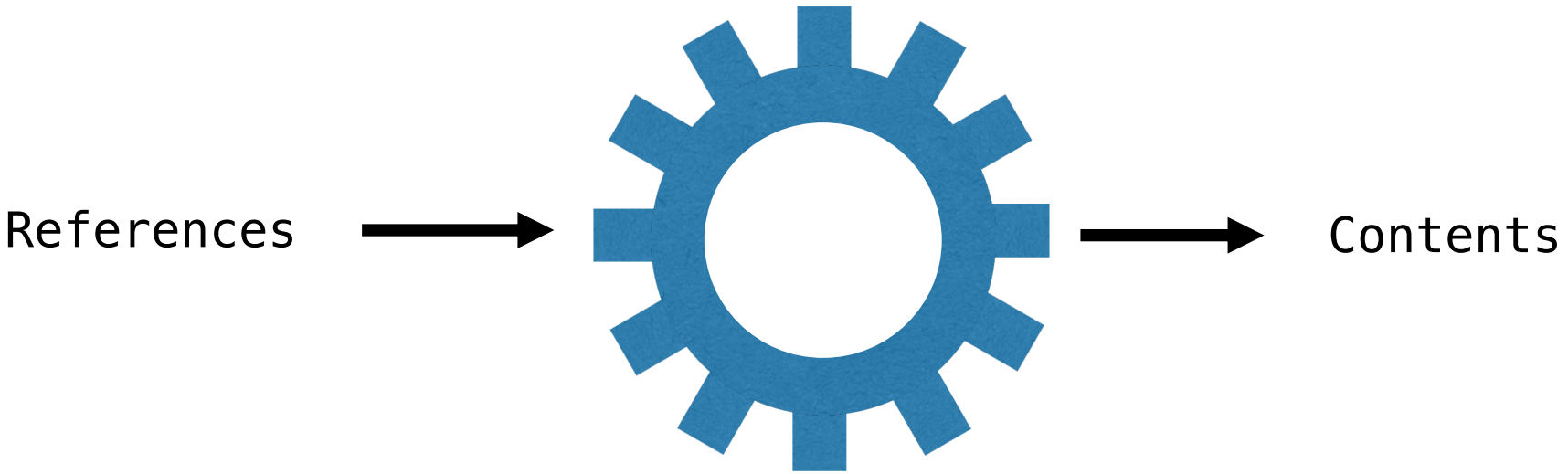
Never leaks jobs

```
suspend fun processReferences(refs: List<Reference>) =
    coroutineScope {
        for (ref in refs) {
            val location = ref.resolveLocation()
            launch {
                val content = downloadContent(location)
                processContent(ref, content)
            }
        }
    }
```

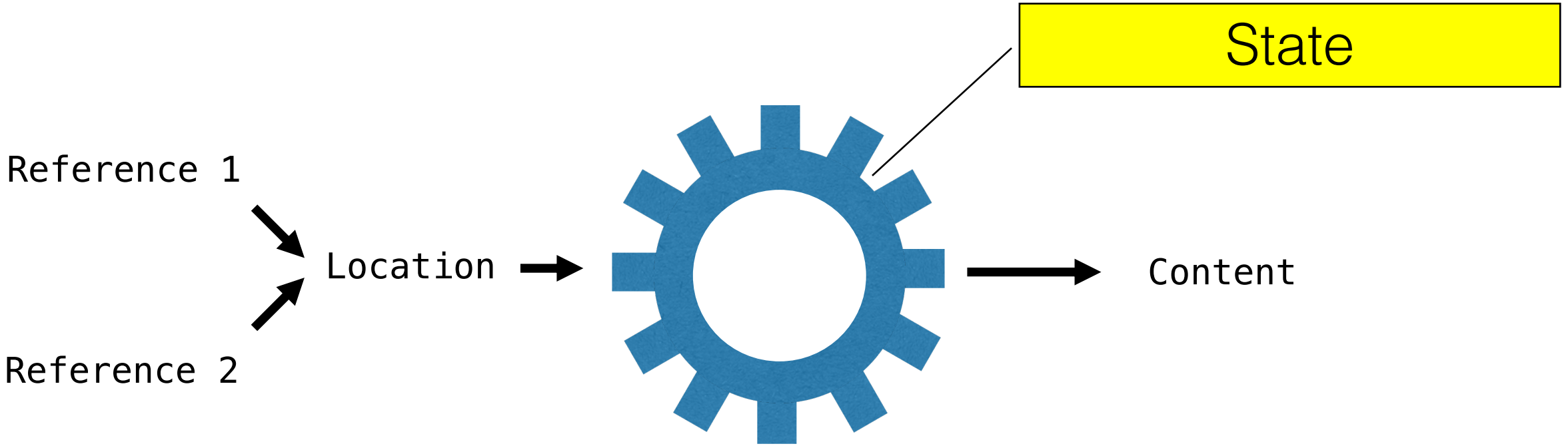
KotlinConf

The state

Download process



Download process



```
class Downloader {

}
```

```
class Downloader {
    private val requested = mutableSetOf<Location>()
}
```

```
class Downloader {
    private val requested = mutableSetOf<Location>()

    fun downloadReference(ref: Reference) {
        val location = ref.resolveLocation()
        …
    }
}
```

```
class Downloader {
    private val requested = mutableSetOf<Location>()

    fun downloadReference(ref: Reference) {
        val location = ref.resolveLocation()
        if (requested.add(location)) {
            // schedule download
        }
    }
}
```

```
class Downloader {
    private val requested = mutableSetOf<Location>()

    fun downloadReference(ref: Reference) {
        val location = ref.resolveLocation()
        if (requested.add(location)) {
            // schedule download
        }
        // ... wait for result ...
        processContent(ref, content)
    }
}
```

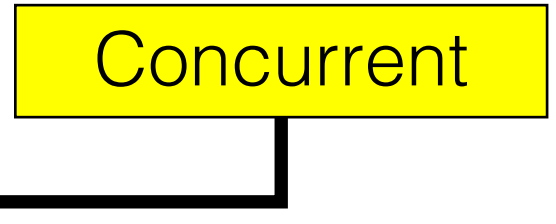
```
class Downloader {
    private val requested = mutableSetOf<Location>()

    fun downloadReference(ref: Reference) {
        val location = ref.resolveLocation()
        if (requested.add(location)) {
            // schedule download
        }
        // ... wait for result ...
        processContent(ref, content)
    }
}
```

Concurrent

```
class Downloader {
    private val requested = mutableSetOf<Location>()

    fun downloadReference(ref: Reference) {
        val location = ref.resolveLocation()
        if (requested.add(location)) {
            // schedule download
        }
        // ... wait for result ...
        processContent(ref, content)
    }
}
```



Shared mutable state

```
class Downloader {
    private val requested = mutableSetOf<Location>()

    fun downloadReference(ref: Reference) {
        val location = ref.resolveLocation()
        if (requested.add(location)) {
            // schedule download
        }
        // ... wait for result ...
        processContent(ref, content)
    }
}
```

Shared + Mutable =

Shared
Mutable State

Share by
Communicating

Synchronization Primitives

Communication Primitives

classes  coroutines

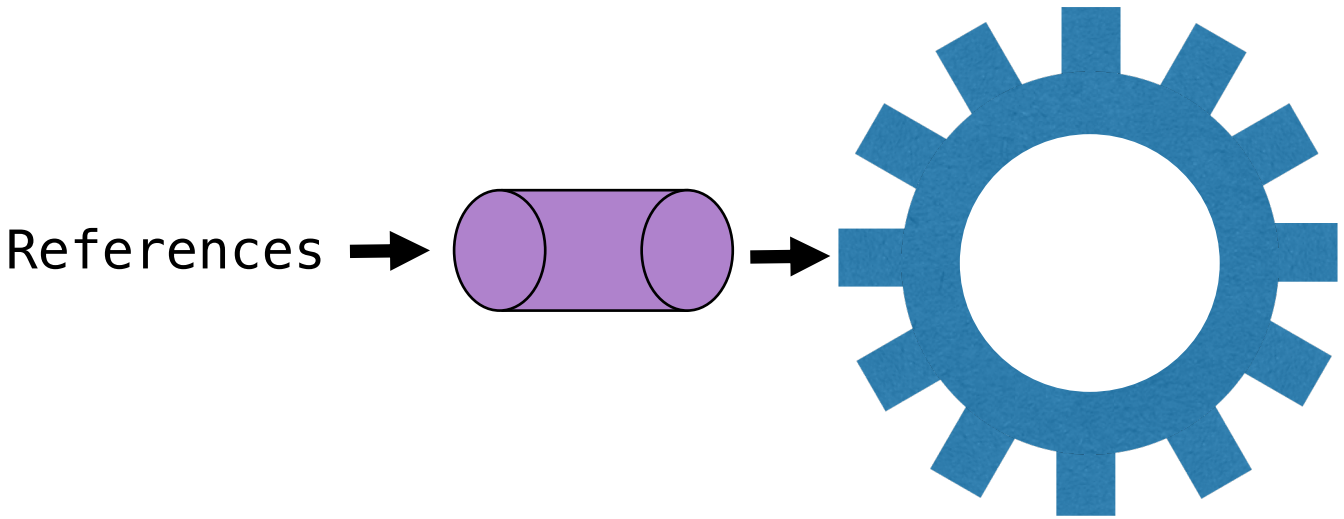
Does not share mutable state

```
launch {
    val requested = mutableSetOf<Location>()
    …
}
```

```
launch {
    val requested = mutableSetOf<Location>()
    for (ref in references) {
        val location = ref.resolveLocation()
        if (requested.add(location)) {
            // schedule download
        }
        // ... wait for result ...
        processContent(ref, content)
    }
}
```

```
launch {
    val requested = mutableSetOf<Location>()
    for (ref in references) {
        val location = ref.resolveLocation()
        if (requested.add(location)) {
            // schedule download
        }
        // ... wait for result ...
        processContent(ref, content)
    }
}
```

Channel



```
fun CoroutineScope.downloader(
    references: ReceiveChannel<Reference>,
) =
    launch {
        …
    }
```

```
fun CoroutineScope.downloader(
    references: ReceiveChannel<Reference>,
) =
    launch {
        …
    }
```

```
fun CoroutineScope.downloader(
    references: ReceiveChannel<Reference>,
) =
    launch {
        …
    }
```


```
fun CoroutineScope.downloader(
    references: ReceiveChannel<Reference>,
) =
    launch {
        …
    }
```

```
fun CoroutineScope.downloader(
    references: ReceiveChannel<Reference>,
) =
    launch {
        val requested = mutableSetOf<Location>()
        for (ref in references) {
            val location = ref.resolveLocation()
            if (requested.add(location)) {
                // schedule download
            }
            // ... wait for result ...
            processContent(ref, content)
        }
    }
```

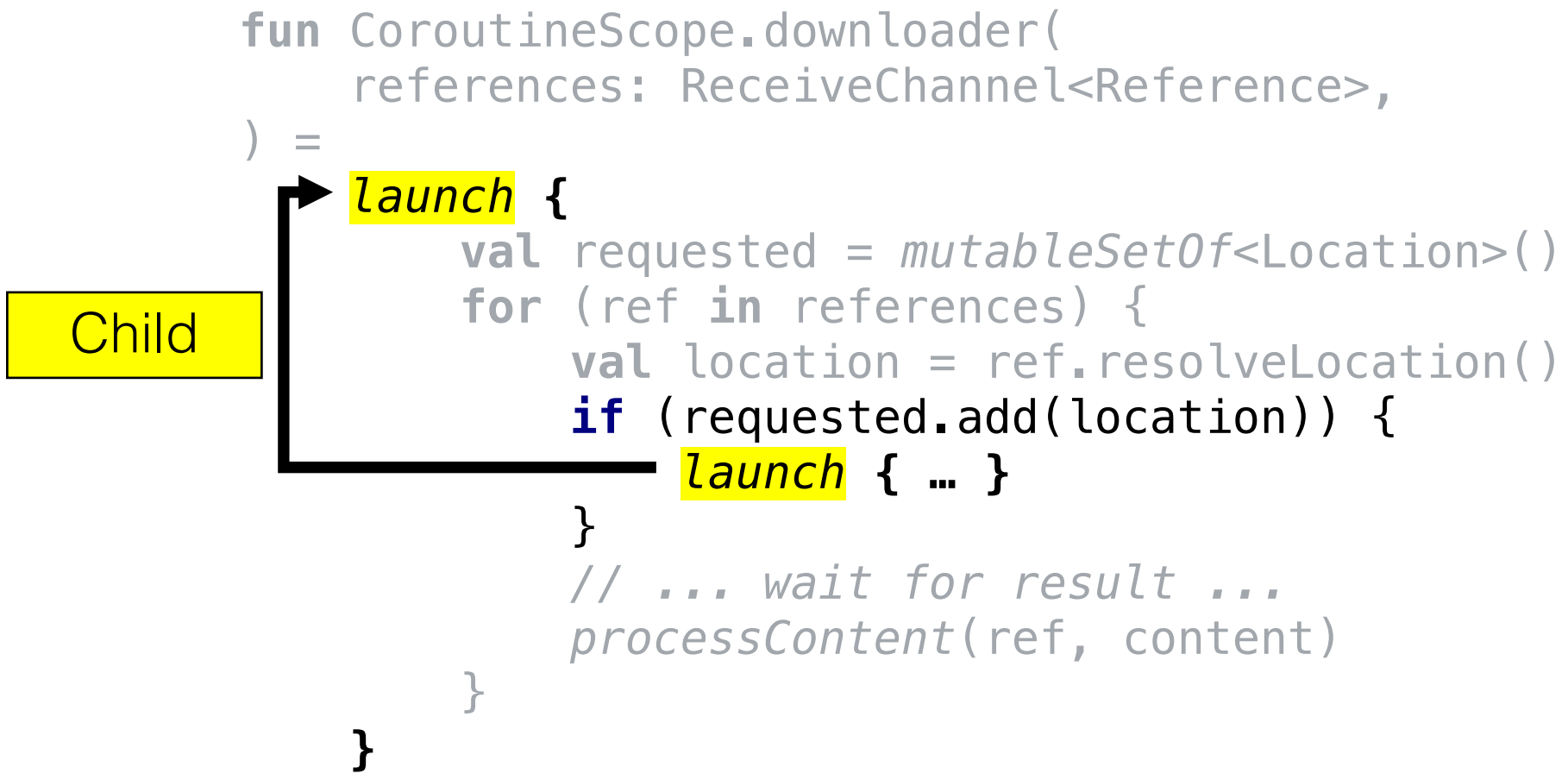
```
fun CoroutineScope.downloader(
    references: ReceiveChannel<Reference>,
) =
    launch {
        val requested = mutableSetOf<Location>()
        for (ref in references) {
            val location = ref.resolveLocation()
            if (requested.add(location)) {
                // schedule download
            }
            // ... wait for result ...
            processContent(ref, content)
        }
    }
```

```
fun CoroutineScope.downloader(
    references: ReceiveChannel<Reference>,
) =
    launch {
        val requested = mutableSetOf<Location>()
        for (ref in references) {
            val location = ref.resolveLocation()
            if (requested.add(location)) {
                // schedule download
            }
            // ... wait for result ...
            processContent(ref, content)
        }
    }
```

```
fun CoroutineScope.downloader(
    references: ReceiveChannel<Reference>,
) =
    launch {
        val requested = mutableSetOf<Location>()
        for (ref in references) {
            val location = ref.resolveLocation()
            if (requested.add(location)) {
                // schedule download
            }
            // ... wait for result ...
            processContent(ref, content)
        }
    }
```

```
fun CoroutineScope.downloader(
    references: ReceiveChannel<Reference>,
) =
    launch {
        val requested = mutableSetOf<Location>()
        for (ref in references) {
            val location = ref.resolveLocation()
            if (requested.add(location)) {
                launch { … }
            }
            // ... wait for result ...
            processContent(ref, content)
        }
    }
```

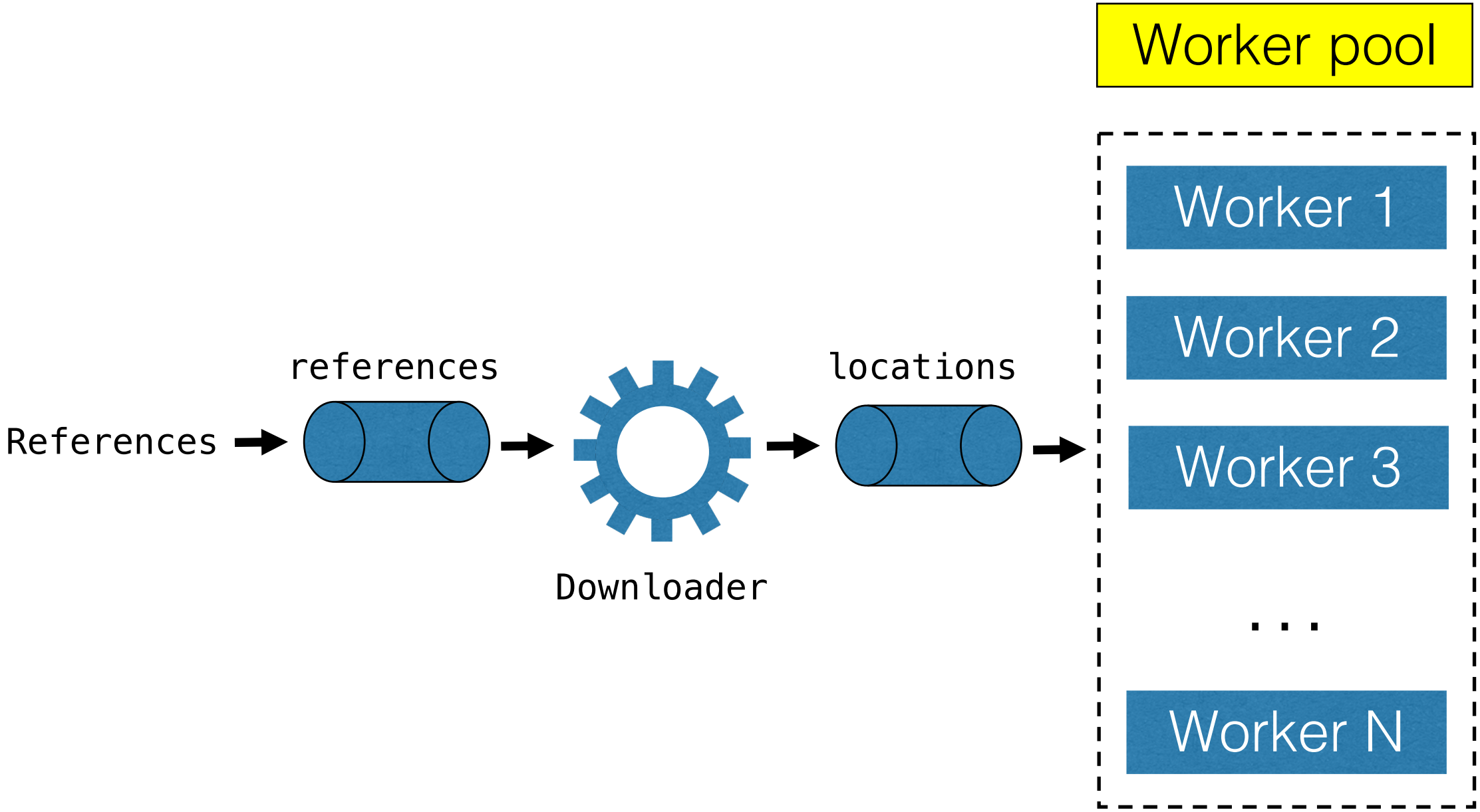
Coroutines are cheap! What could go wrong?


Child
fun CoroutineScope.downloader(
references: ReceiveChannel<Reference>,
) =
launch {
val requested = mutableSetOf<Location>()
for (ref in references) {
val location = ref.resolveLocation()
if (requested.add(location)) {
launch { … }
}
// ... wait for result ...
processContent(ref, content)
}
}

```
fun CoroutineScope.downloader(
    references: ReceiveChannel<Reference>,
) =
    launch {
        val requested = mutableSetOf<Location>()
        for (ref in references) {
            val location = ref.resolveLocation()
            if (requested.add(location)) {
                launch { … }
            }
            // ... wait for result ...
            processContent(ref, content)
        }
    }
```

Coroutines are cheap! But the work they do…

Limiting concurrency


Worker pool
Worker 1
Worker 2
Worker 3
...
Worker N
References
references
Downloader
locations

```
fun CoroutineScope.downloader(
    references: ReceiveChannel<Reference>,
    locations: SendChannel<Location>
)
```

```
fun CoroutineScope.downloader(
    references: ReceiveChannel<Reference>,
    locations: SendChannel<Location>
) =
    launch {
        val requested = mutableSetOf<Location>()
        for (ref in references) {
            val location = ref.resolveLocation()
            if (requested.add(location)) {
                locations.send(location)
            }
        }
    }
```

```
fun CoroutineScope.worker(
    locations: ReceiveChannel<Location>
)
```

```
fun CoroutineScope.worker(
    locations: ReceiveChannel<Location>
)
```

```
fun CoroutineScope.worker(
    locations: ReceiveChannel<Location>
) =
    launch {
        for (loc in locations) {
            val content = downloadContent(loc)
            processContent(ref, content)
        }
    }
```

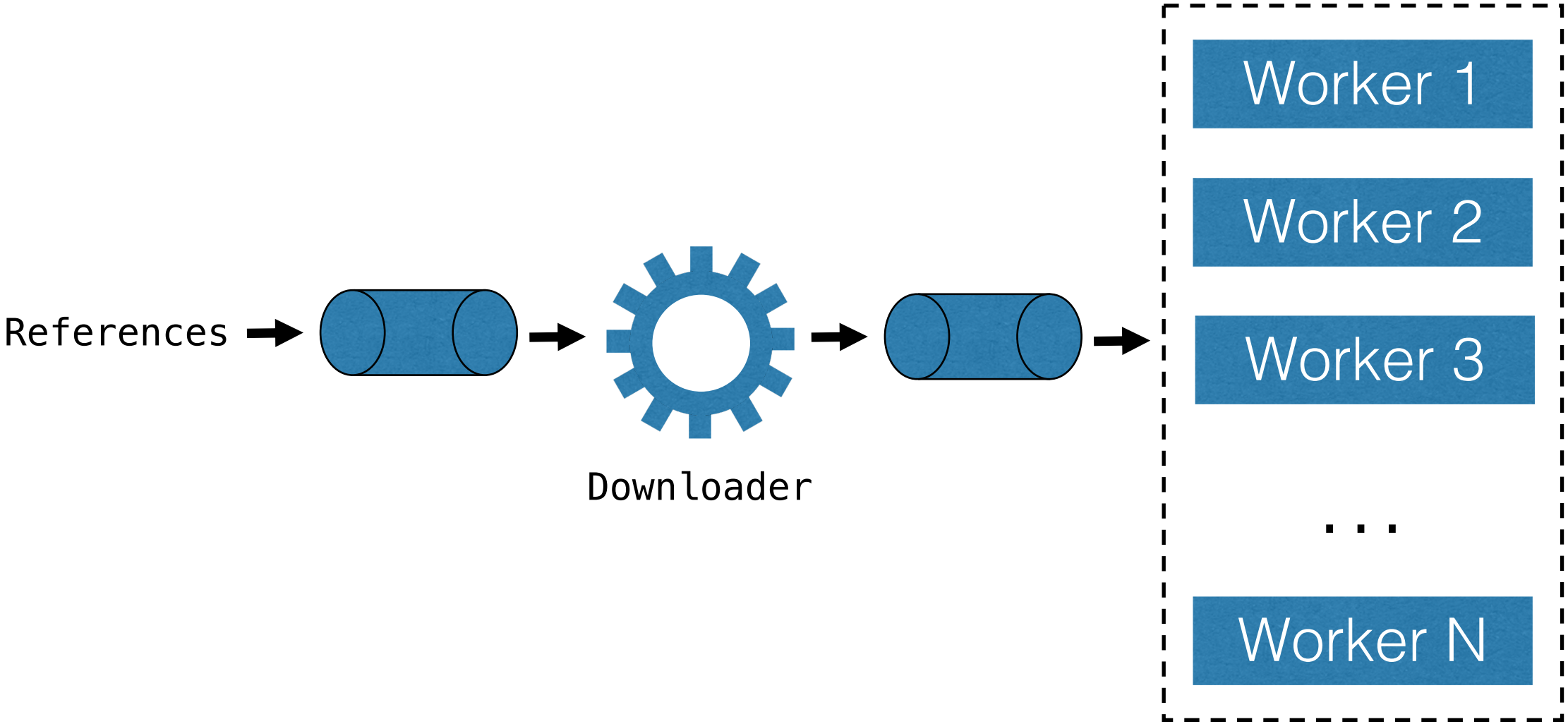
```
fun CoroutineScope.worker(
    locations: ReceiveChannel<Location>
) =
    launch {
        for (loc in locations) {
            val content = downloadContent(loc)
            processContent(ref, content)
        }
    }
```

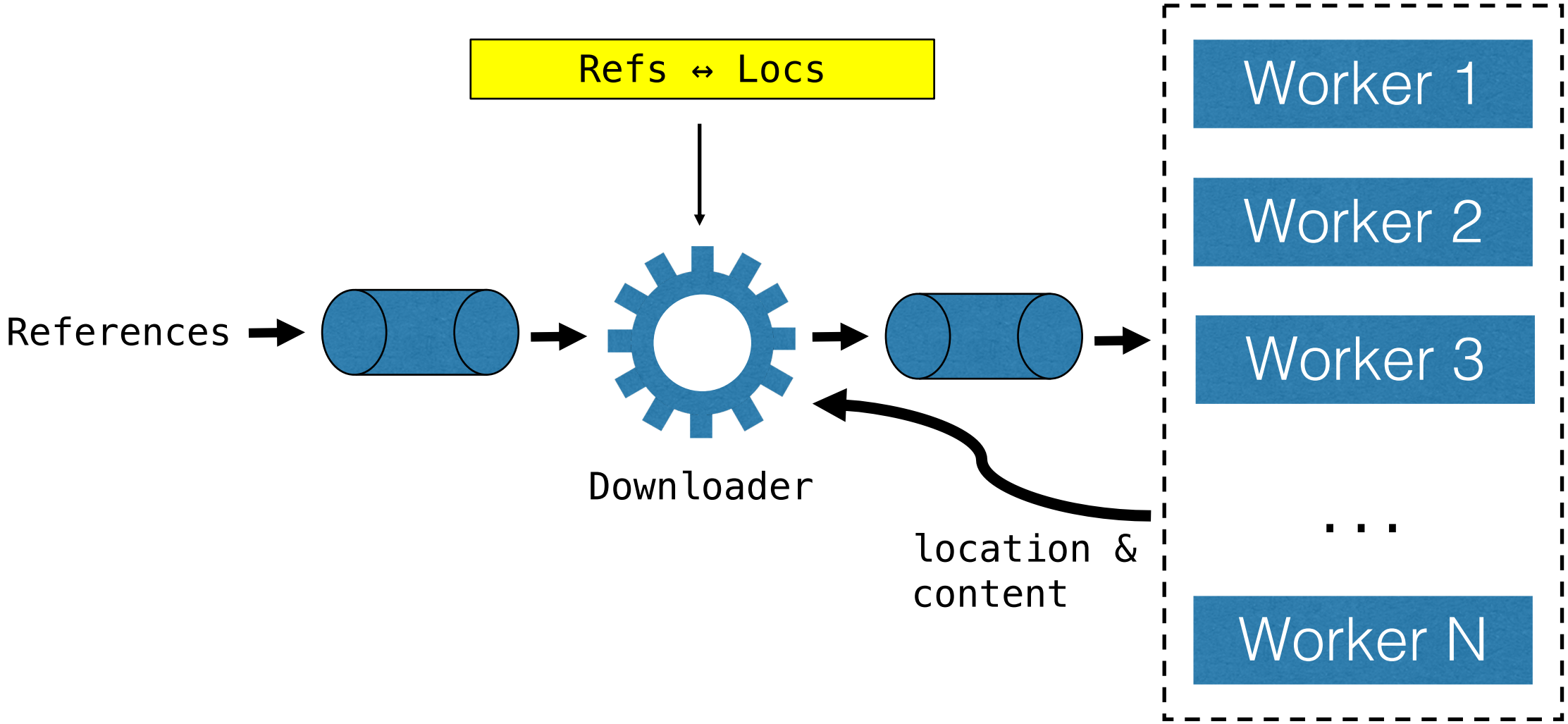
Fan-out

```
fun CoroutineScope.worker(
    locations: ReceiveChannel<Location>
) =
    launch {
        for (loc in locations) {
            val content = downloadContent(location)
            processContent(ref, content)
        }
    }
```

```
fun CoroutineScope.worker(
    locations: ReceiveChannel<Location>
) =
    launch {
        for (loc in locations) {
            val content = downloadContent(loc)
            processContent(ref, content)
        }
    }
```

```
fun CoroutineScope.worker(
    locations: ReceiveChannel<Location>
) =
    launch {
        for (loc in locations) {
            val content = downloadContent(loc)
            processContent(ref, content)
        }
    }
```


References
Downloader
Worker 1
Worker 2
Worker 3
...
Worker N


Refs ↔ Locs
References
Downloader
location &
content
Worker 1
Worker 2
Worker 3
...
Worker N

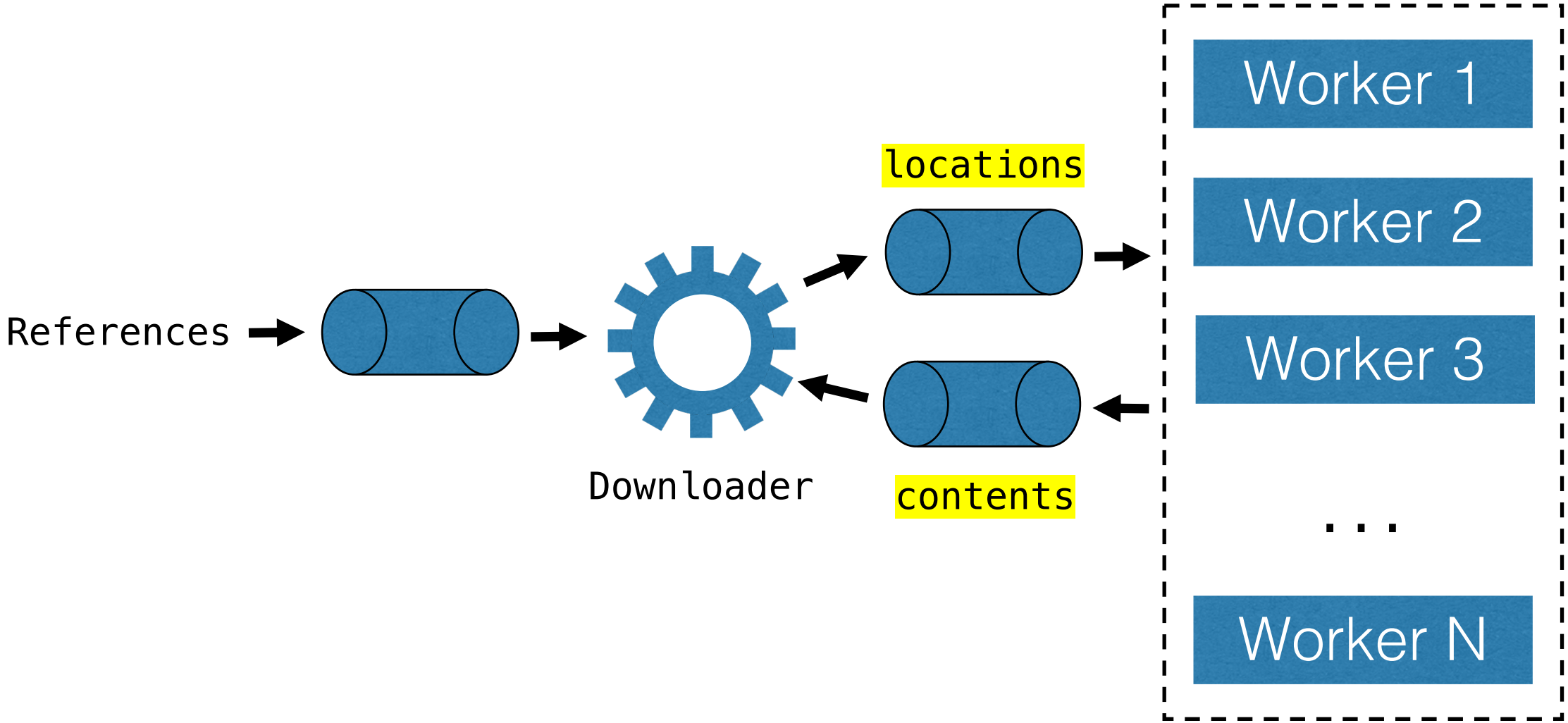
```
data class LocContent(val loc: Location, val content: Content)
```

```
data class LocContent(val loc: Location, val content: Content)

fun CoroutineScope.worker(
    locations: ReceiveChannel<Location>,
    contents: SendChannel<LocContent>
)
```

```
data class LocContent(val loc: Location, val content: Content)

fun CoroutineScope.worker(
    locations: ReceiveChannel<Location>,
    contents: SendChannel<LocContent>
) =
    launch {
        for (loc in locations) {
            val content = downloadContent(loc)
            contents.send(LocContent(loc, content))
        }
    }
```


References
Downloader
locations
contents
Worker 1
Worker 2
Worker 3
...
Worker N

```
fun CoroutineScope.downloader(
    references: ReceiveChannel<Reference>,
    locations: SendChannel<Location>,
    contents: ReceiveChannel<LocContent>
)
```

```
fun CoroutineScope.downloader(
    references: ReceiveChannel<Reference>,
    locations: SendChannel<Location>,
    contents: ReceiveChannel<LocContent>
) =
    launch {
        val requested = mutableSetOf<Location>()
        for (ref in references) {
            val location = ref.resolveLocation()
            if (requested.add(location)) {
                locations.send(location)
            }
        }
    }
```

Hmm....

Select

```
select {
    references.onReceive { ref ->
        …
    }
    contents.onReceive { (loc, content) ->
        …
    }
}
```

```
select {
    references.onReceive { ref ->
        …
    }
    contents.onReceive { (loc, content) ->
        …
    }
}
```

```
select<Unit> {
    references.onReceive { ref ->
        …
    }
    contents.onReceive { (loc, content) ->
        …
    }
}
```

```
launch {
    val requested = mutableMapOf<Location, MutableList<Reference>>()
    …
}
```

```
launch {
    val requested = mutableMapOf<Location, MutableList<Reference>>()
    while (true) {
        select<Unit> {
            references.onReceive { ref -> … }
            contents.onReceive { (loc, content) -> … }
        }
    }
}
```

```
launch {
    val requested = mutableMapOf<Location, MutableList<Reference>>()
    while (true) {
        select<Unit> {
            references.onReceive { ref -> … }
            contents.onReceive { (loc, content) -> … }
        }
    }
}
```

```
launch {
    val requested = mutableMapOf<Location, MutableList<Reference>>()
    while (true) {
        select<Unit> {
            references.onReceive { ref ->
                val loc = ref.resolveLocation()
                …
            }
            contents.onReceive { (loc, content) -> … }
        }
    }
}
```

```
launch {
    val requested = mutableMapOf<Location, MutableList<Reference>>()
    while (true) {
        select<Unit> {
            references.onReceive { ref ->
                val loc = ref.resolveLocation()
                val refs = requested[loc]
                …
            }
            contents.onReceive { (loc, content) -> … }
        }
    }
}
```

```
launch {
    val requested = mutableMapOf<Location, MutableList<Reference>>()
    while (true) {
        select<Unit> {
            references.onReceive { ref ->
                val loc = ref.resolveLocation()
                val refs = requested[loc]
                if (refs == null) {
                    requested[loc] = mutableListOf(ref)
                    locations.send(loc)
                }
            }
            contents.onReceive { (loc, content) -> … }
        }
    }
}
```

```
launch {
    val requested = mutableMapOf<Location, MutableList<Reference>>()
    while (true) {
        select<Unit> {
            references.onReceive { ref ->
                val loc = ref.resolveLocation()
                val refs = requested[loc]
                if (refs == null) {
                    requested[loc] = mutableListOf(ref)
                    locations.send(loc)
                } else {
                    refs.add(ref)
                }
            }
            contents.onReceive { (loc, content) -> … }
        }
    }
}
```

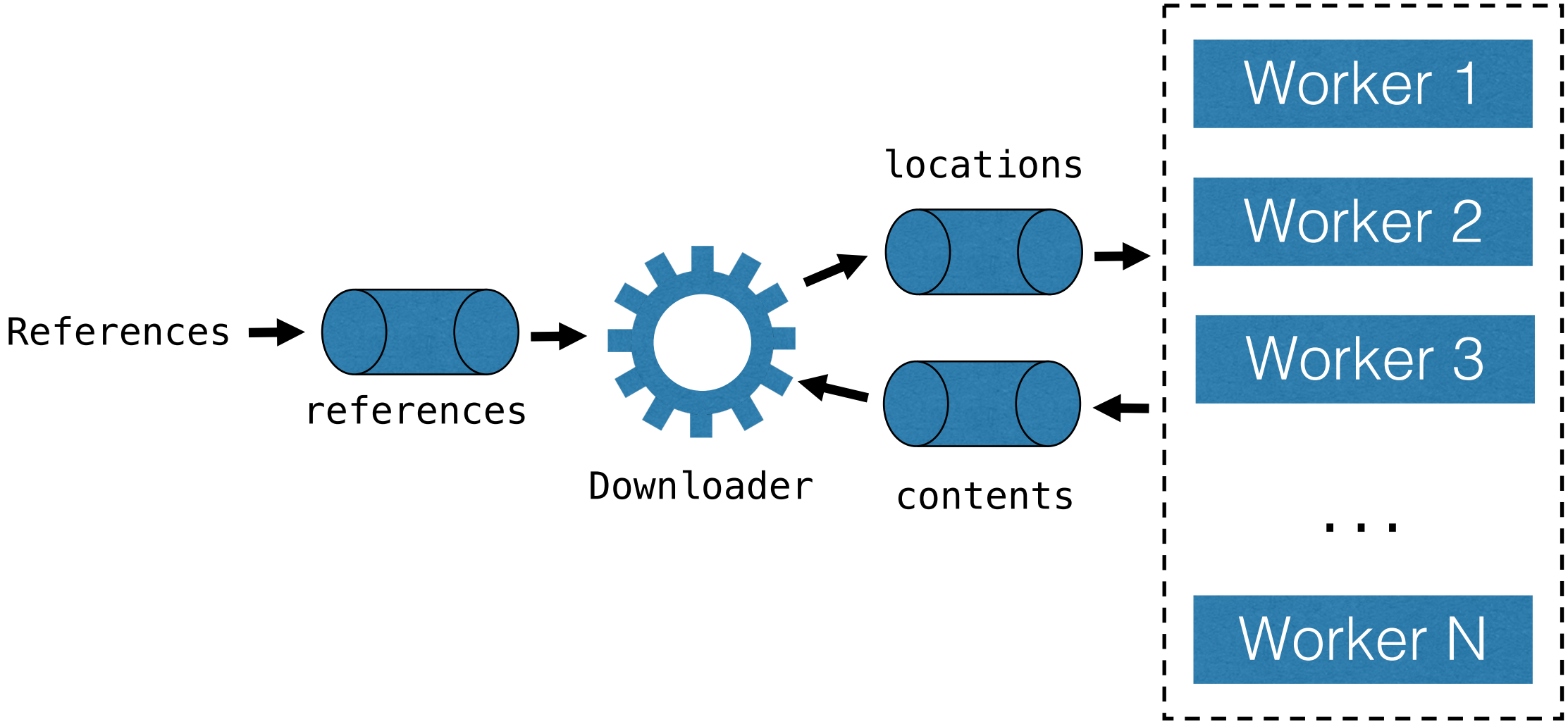
No concurrency

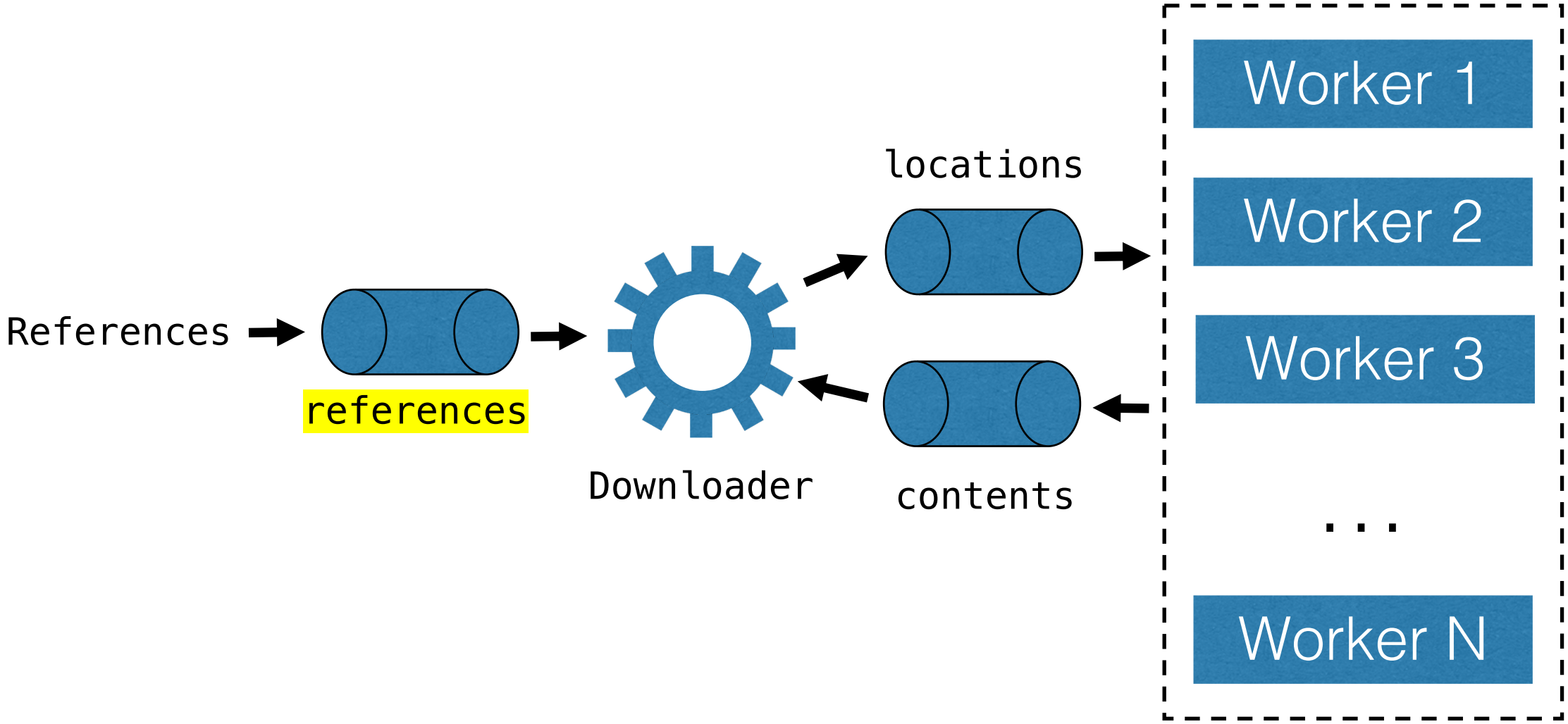
No synchronization

```
launch {
    val requested = mutableMapOf<Location, MutableList<Reference>>()
    while (true) {
        select<Unit> {
            references.onReceive { ref ->
                val loc = ref.resolveLocation()
                val refs = requested[loc]
                if (refs == null) {
                    requested[loc] = mutableListOf(ref)
                    locations.send(loc)
                } else {
                    refs.add(ref)
                }
            }
            contents.onReceive { (loc, content) -> … }
        }
    }
}
```

```
launch {
    val requested = mutableMapOf<Location, MutableList<Reference>>()
    while (true) {
        select<Unit> {
            references.onReceive { ref -> … }
            contents.onReceive { (loc, content) ->
                val refs = requested.remove(loc)!!
                for (ref in refs) {
                    processContent(ref, content)
                }
            }
        }
    }
}
```

Putting it all together


References
references
Downloader
locations
contents
Worker 1
Worker 2
Worker 3
...
Worker N

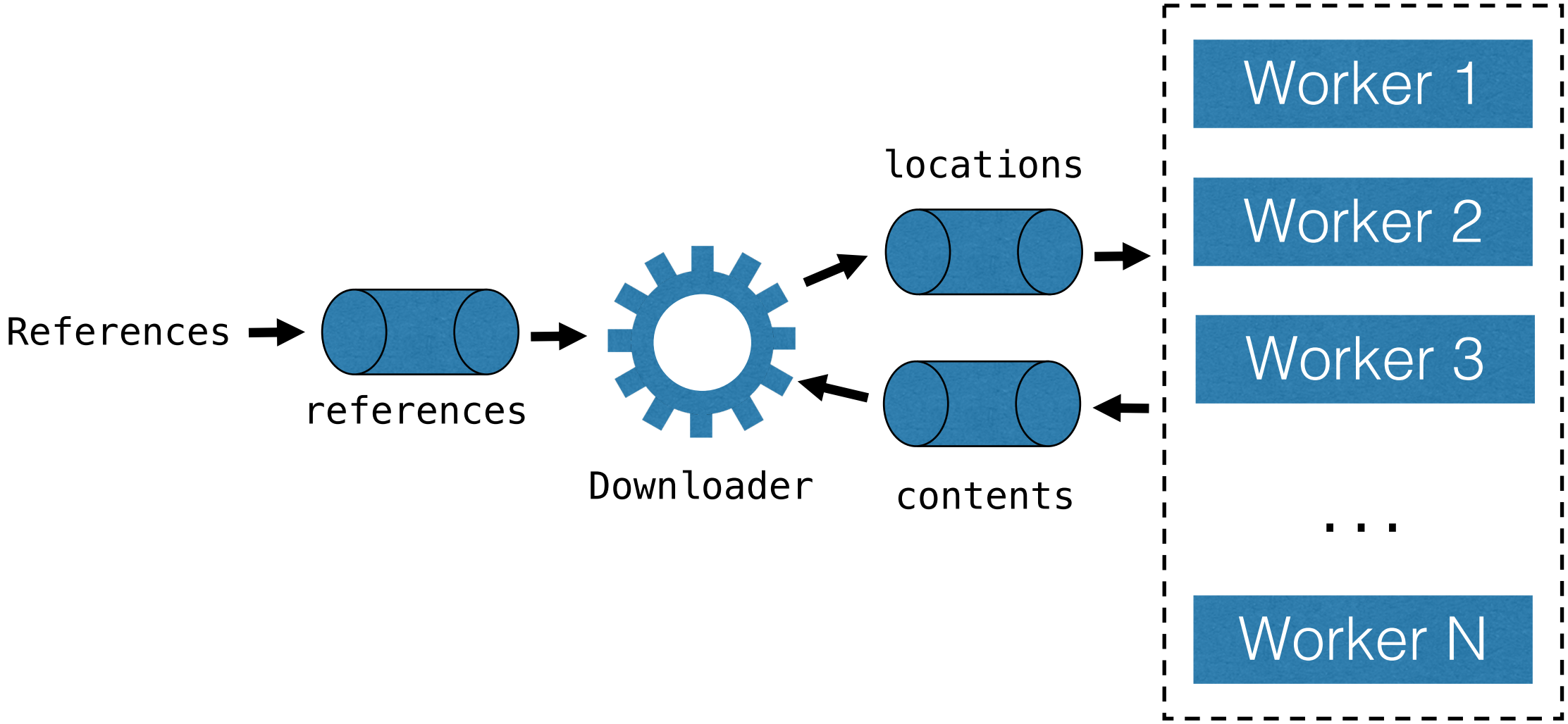

References
references
Downloader
locations
contents
Worker 1
Worker 2
Worker 3
...
Worker N

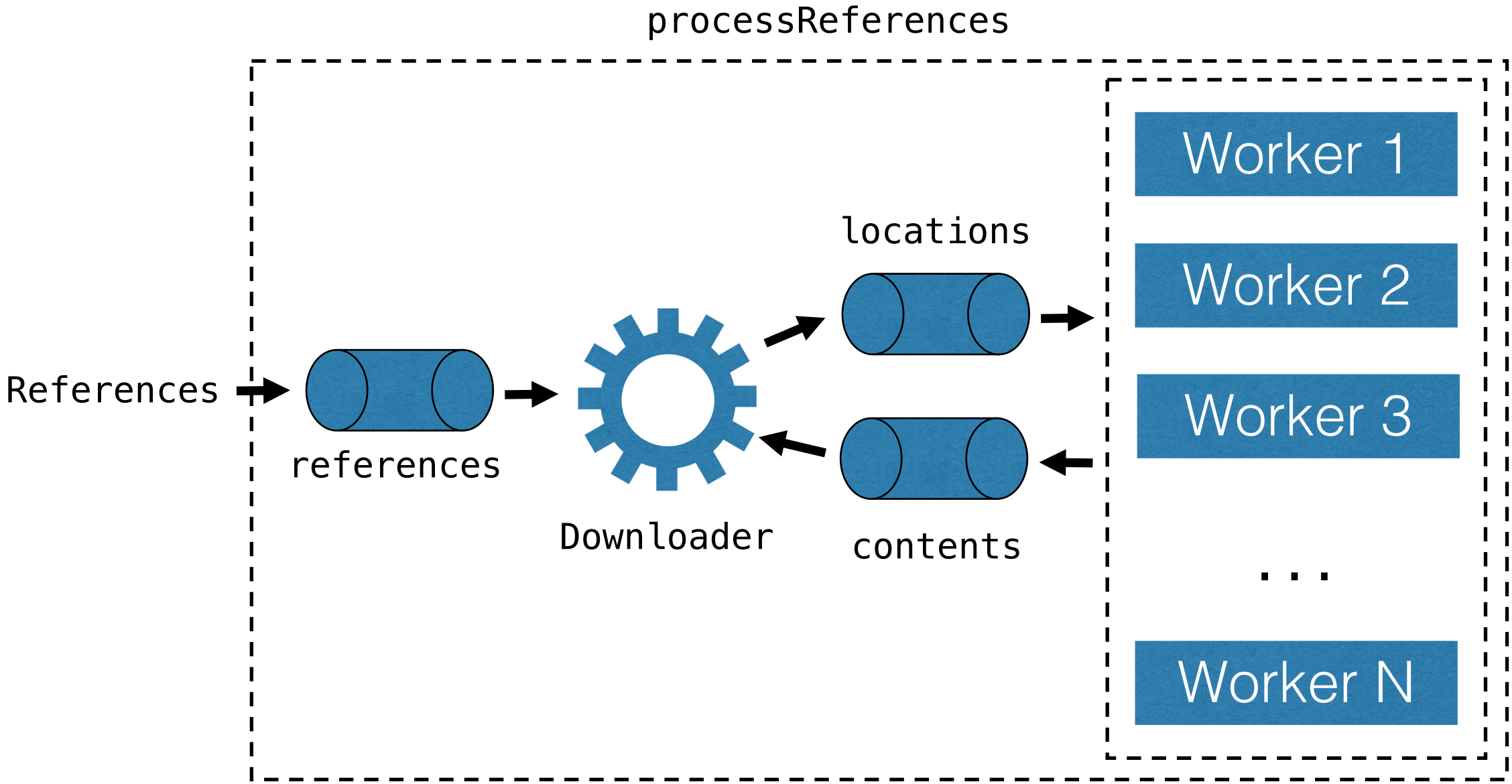
```
fun CoroutineScope.processReferences(
    references: ReceiveChannel<Reference>
)
```

```
fun CoroutineScope.processReferences(
    references: ReceiveChannel<Reference>
)
```

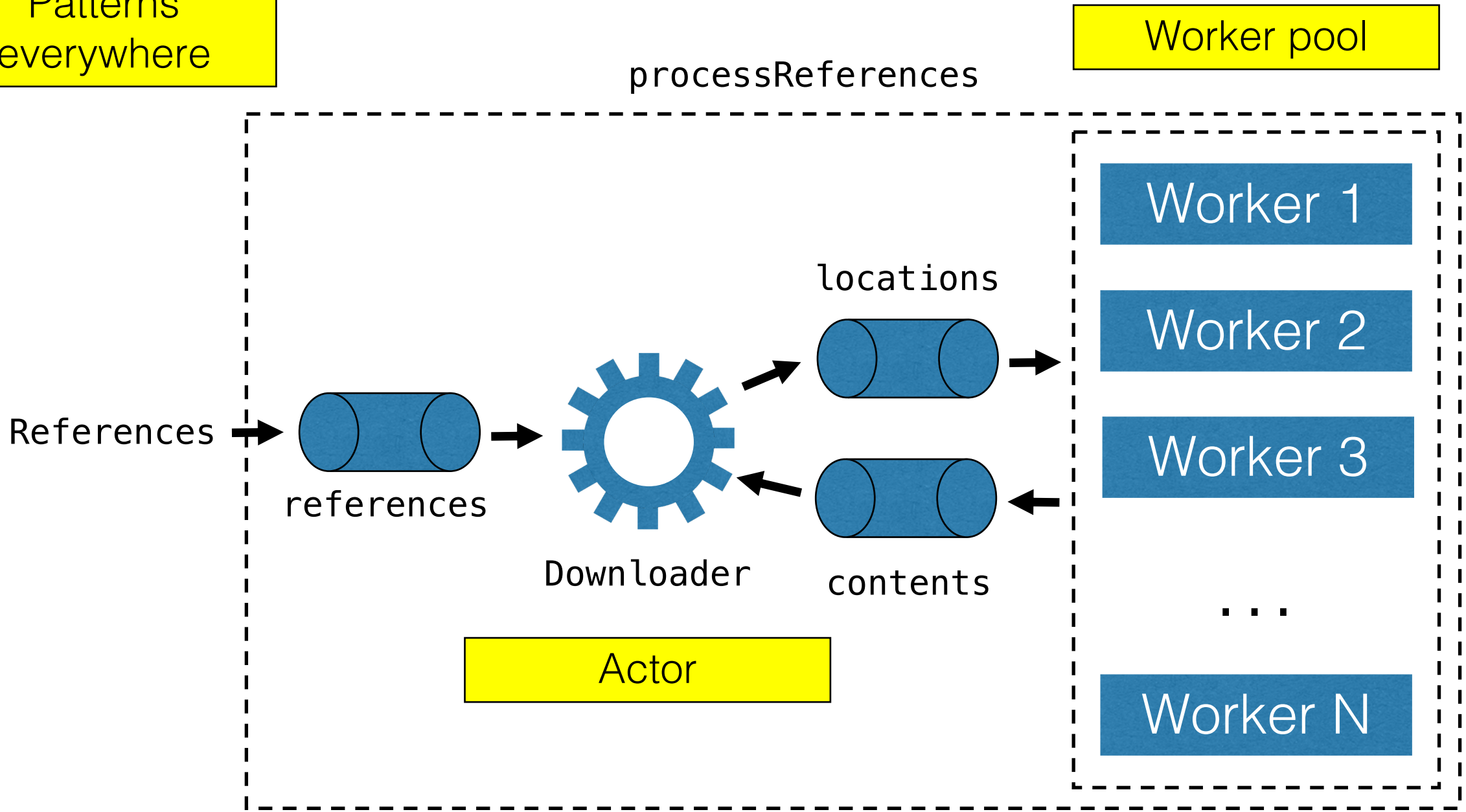
```
fun CoroutineScope.processReferences(
    references: ReceiveChannel<Reference>
) {
    val locations = Channel<Location>()
    val contents = Channel<LocContent>()
    repeat(N_WORKERS) { worker(locations, contents) }
    downloader(references, locations, contents)
}
```

```
fun CoroutineScope.processReferences(
    references: ReceiveChannel<Reference>
) {
    val locations = Channel<Location>()
    val contents = Channel<LocContent>()
    repeat(N_WORKERS) { worker(locations, contents) }
    downloader(references, locations, contents)
}
```


References
references
Downloader
locations
contents
Worker 1
Worker 2
Worker 3
...
Worker N


processReferences
References
references
Downloader
locations
contents
Worker 1
Worker 2
Worker 3
...
Worker N

Patterns everywhere



```
fun CoroutineScope.processReferences(…)
```

Root CoroutineScope

```
class SomethingWithLifecycle {

}
```

```
class SomethingWithLifecycle : CoroutineScope {

}
```

```
class SomethingWithLifecycle : CoroutineScope {
    override val coroutineContext: CoroutineContext
        get() = …
}
```

```
class SomethingWithLifecycle : CoroutineScope {
    private val job = Job()

    override val coroutineContext: CoroutineContext
        get() = …
}
```

```
class SomethingWithLifecycle : CoroutineScope {
    private val job = Job()

    fun dispose() { … }

    override val coroutineContext: CoroutineContext
        get() = …
}
```

```
class SomethingWithLifecycle : CoroutineScope {
    private val job = Job()

    fun close() { … }

    override val coroutineContext: CoroutineContext
        get() = …
}
```

```
class SomethingWithLifecycle : CoroutineScope {
    private val job = Job()

    fun close() {
        job.cancel()
    }

    override val coroutineContext: CoroutineContext
        get() = …
}
```

```
class SomethingWithLifecycle : CoroutineScope {
    private val job = Job()

    fun close() {
        job.cancel()
    }

    override val coroutineContext: CoroutineContext
        get() = job
}
```

```
class SomethingWithLifecycle : CoroutineScope {
    private val job = Job()

    fun close() {
        job.cancel()
    }

    override val coroutineContext: CoroutineContext
        get() = job + Dispatchers.Main
}
```

```
class SomethingWithLifecycle : CoroutineScope {
    …
    override val coroutineContext: CoroutineContext
        get() = job + Dispatchers.Main

    fun doSomething() {

    }
}
```

```
class SomethingWithLifecycle : CoroutineScope {
    …
    override val coroutineContext: CoroutineContext
        get() = job + Dispatchers.Main

    fun doSomething() {
        launch { … }
    }
}
```

```
class SomethingWithLifecycle : CoroutineScope {
    …
    override val coroutineContext: CoroutineContext
        get() = job + Dispatchers.Main

    fun doSomething() {
        processReferences(references)
    }
}
```

Never leak any coroutines

KotlinConf

suspend vs scope

```
suspend fun downloadContent(location: Location): Content
```

Does something long &
waits for it to complete without blocking

```
suspend fun downloadContent(location: Location): Content
```

```
suspend fun downloadContent(location: Location): Content
```

```
fun CoroutineScope.processReferences(…)
```

```
suspend fun downloadContent(location: Location): Content
```

```
fun CoroutineScope.processReferences(…)
```

Launches new coroutines &
quickly returns, does not wait for them

KotlinConf

Takeaway

Coroutines are like light-weight threads

Coroutines are **NOT** like threads

Coroutines are **NOT** like threads

Rethink the way you structure your code

Thank you

Any questions?

Roman Elizarov
elizarov@
@relizarov