Tech Talks

# Chasing Races

## Deterministic Simulation Testing in XTDB

James Henderson, XTDB Head of Engineering

2025-12-05

# Coming up:

- The **value** and **cost** of a test
- What we test in XTDB, and how
- DST: the theory
- What this looks like in Kotlin

# The Value and Cost of a Test

# Value ∝ ?

- **False negative** rate
- **Probability** of a bug in the code-under-test
- **Impact** of a bug in the code-under-test
- Documentation value
- others...?

# Cost ∝ ?

- **Time** to write it (obviously)
- **False positive** rate
- **Maintenance** cost
- Execution time
- Setup
- others...?

# What we test on XTDB, and how?

- ‘Highly general’ code: example-based **unit tests**
  - e.g. bitemporal resolution, what to compact
- Complex outputs: **regression tests**
  - e.g. SQL planner, indexer
- Combinatorial explosions: **property tests**
  - e.g. polymorphic Arrow vectors
- Concurrent/distributed code: **deterministic simulation tests** ← you are here
- Everything else: mostly example-based **integration tests**
  - using the in-memory, throwaway node setup - highly recommend

# Aside: Allen intervals

- Useful in our unit tests + property tests

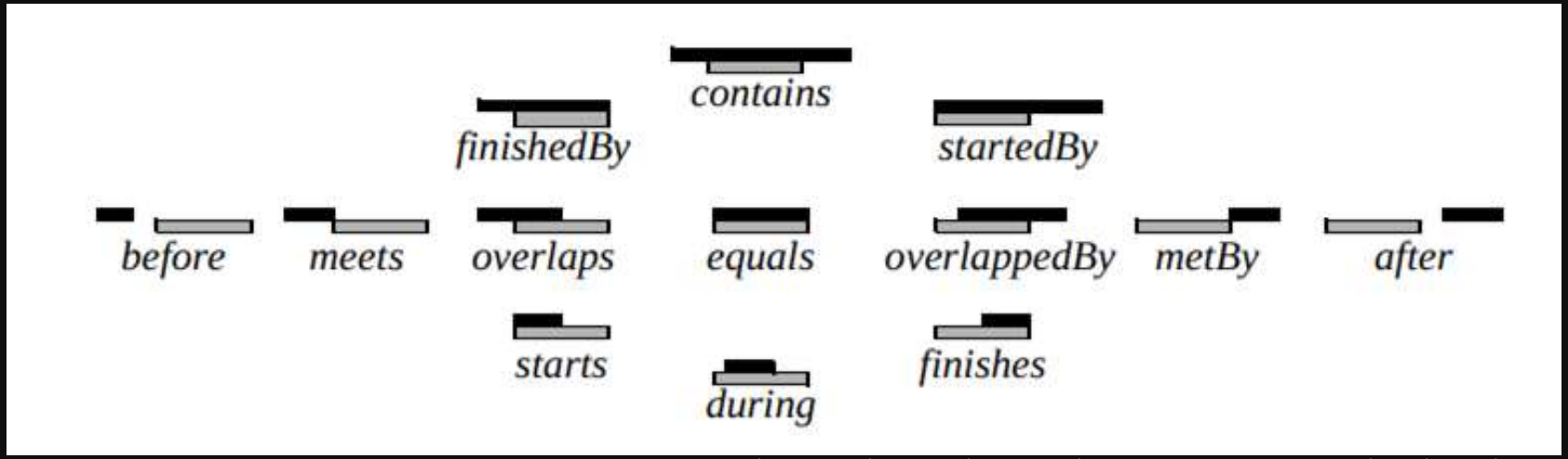



image: "*Object-Relational Indexing for General Interval Relationships*", Kriegel et al.

# *Before* you DST:

- DST is *relatively* complex (compared to unit/integration/property)

- Like any risk, ***avoid*** before you ***mitigate***:
  - Immutability by default
  - Avoid sharing mutable state between threads
  - Separate pure code from side-effects
  - “Communicating sequential processes” (CSP), actors

... a.k.a. ‘Functional Programming Best Practices’

```
sealed interface Event
class FetchReq(val key: Key, val res: CompletableDeferred<Value>) : Event
class FetchDone(val key: Key, val value: Value) : Event

val fetchCh = Channel<Event>()

init {
    scope.launch {
        val state = State(/* ... */)

        for (event in fetchCh) {
            when (event) {
                is FetchReq -> /* ... */
                is FetchDone -> /* ... */
            }
        }
    }
}

suspend fun get(key: Key): Value {
    val res = CompletableDeferred<Value>()
    fetchCh.send(FetchReq(key, res))
    return res.await()
}
```

# Deterministic Simulation Testing (DST)

Problem:

- Concurrent code is non-deterministic - timing dependent
- Vast number of potential ‘interleavings’
- Anything you’d normally do to isolate the issue (e.g. logging, debugger) often makes the issue go away

Solution: just run all the possible interleavings, see if anything breaks!

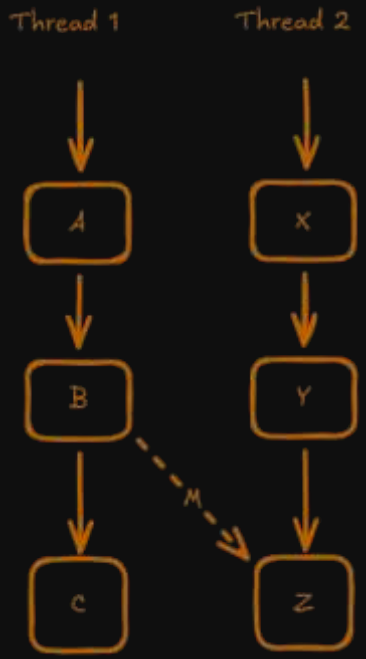
Thank you for coming to my TED talk

# ‘happens before’

1. If events A and B are in the same thread, and A comes before B in that thread, then A→B
2. If A is the sending of a message by one process, and B is the receiving of that same message by another process, then A→B
3. If A→B and B→C, then A→C (‘transitivity’)

‘happens before’ is only a **partial order**.

If !(A→B) and !(B→A) then A and B are **concurrent**, and can happen in either order.




*“Time, Clocks, and the Ordering of Events in a Distributed System”* (Lamport, 1978)

# DST: Trying all the total orderings

Aims:

- Choose an ordering, run it, then test properties of the system
- If an ordering fails any of its properties, allow exact re-running
  - e.g. with a debugger attached, logging turned up.
- Low false positive and false negative rates
  - Test the ‘real code’ as much as possible
  - … so need to minimise performance overhead
- Keep code-under-test simple
  - i.e. introducing DST shouldn’t require complex changes to the underlying code

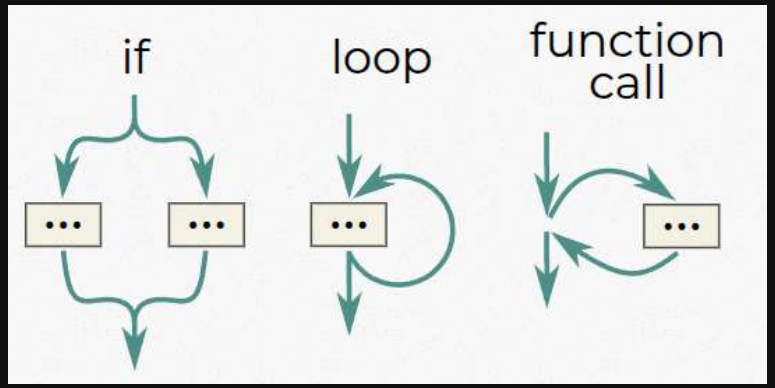
# Coroutines and Continuations:

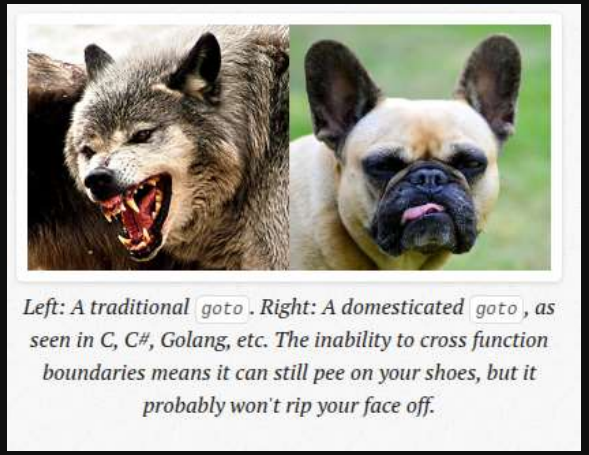
```
fun calculateSum(a: Int, b: Int): Int = a + b
```

```
suspend fun calculateSum(client: HttpClient, a: Int, b: Int): Int =
    client.get("http://example.com/sum?a=$a&b=$b").body()
```

# Aside: ‘structured’ concurrency

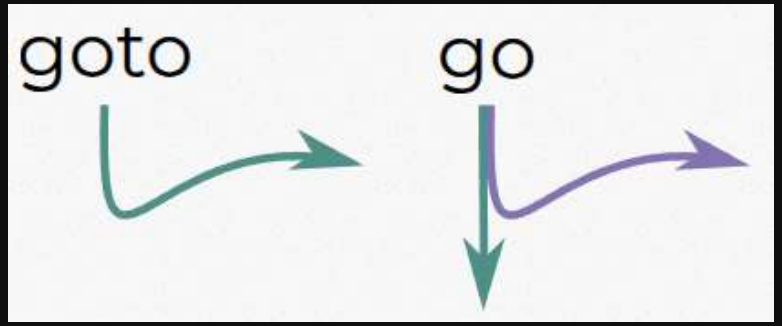
a.k.a. “Go Statement Considered Harmful”

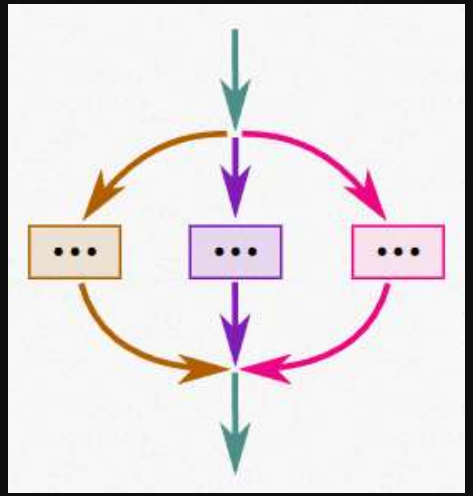



# Aside: ‘structured’ concurrency

a.k.a. “Go Statement Considered Harmful”



# Coroutines and Continuations:

```
suspend fun calculateSum(client: HttpClient, a: Int, b: Int): Int =
    client.get("http://example.com/sum?a=$a&b=$b").body()
```

```
fun calculateSum(
    client: HttpClient, a: Int, b: Int,
    cont: Continuation<Int>
): Unit {
    client.get(
      "http://example.com/sum?a=$a&b=$b",
      Continuation { res -> cont.resume(res.body()) }
    )
}
```

# CoroutineDispatcher

```
public abstract class CoroutineDispatcher {

    /**
     * Requests execution of a runnable [block].
     *
     * The dispatcher guarantees that [block] will eventually execute,
     * typically by dispatching it to a thread pool, using a dedicated thread,
     * or just executing the block in place.
     *
     * This method should guarantee that the given [block] will be eventually invoked,
     * otherwise the system may reach a deadlock state and never leave it.
     */
    public abstract fun dispatch(context: CoroutineContext, block: Runnable)
}
```

# CoroutineDispatcher

```
class DeterministicDispatcher(private val rand: Random) : CoroutineDispatcher() {

    private val jobs = mutableSetOf<Runnable>()
    private var running = false

    override fun dispatch(context: CoroutineContext, block: Runnable) {
        jobs.add(block)

        if (!running) {
            running = true
            while (true) {
                val job = jobs.randomOrNull(rand) ?: break
                jobs.remove(job)
                job.run()
            }
            running = false
        }
    }
}
```

# Isolating the observable side-effects

Goal:

> every time there’s a observable side-effect,
> give the Dispatcher chance to re-order.

# Isolating the observable side-effects

Goal:

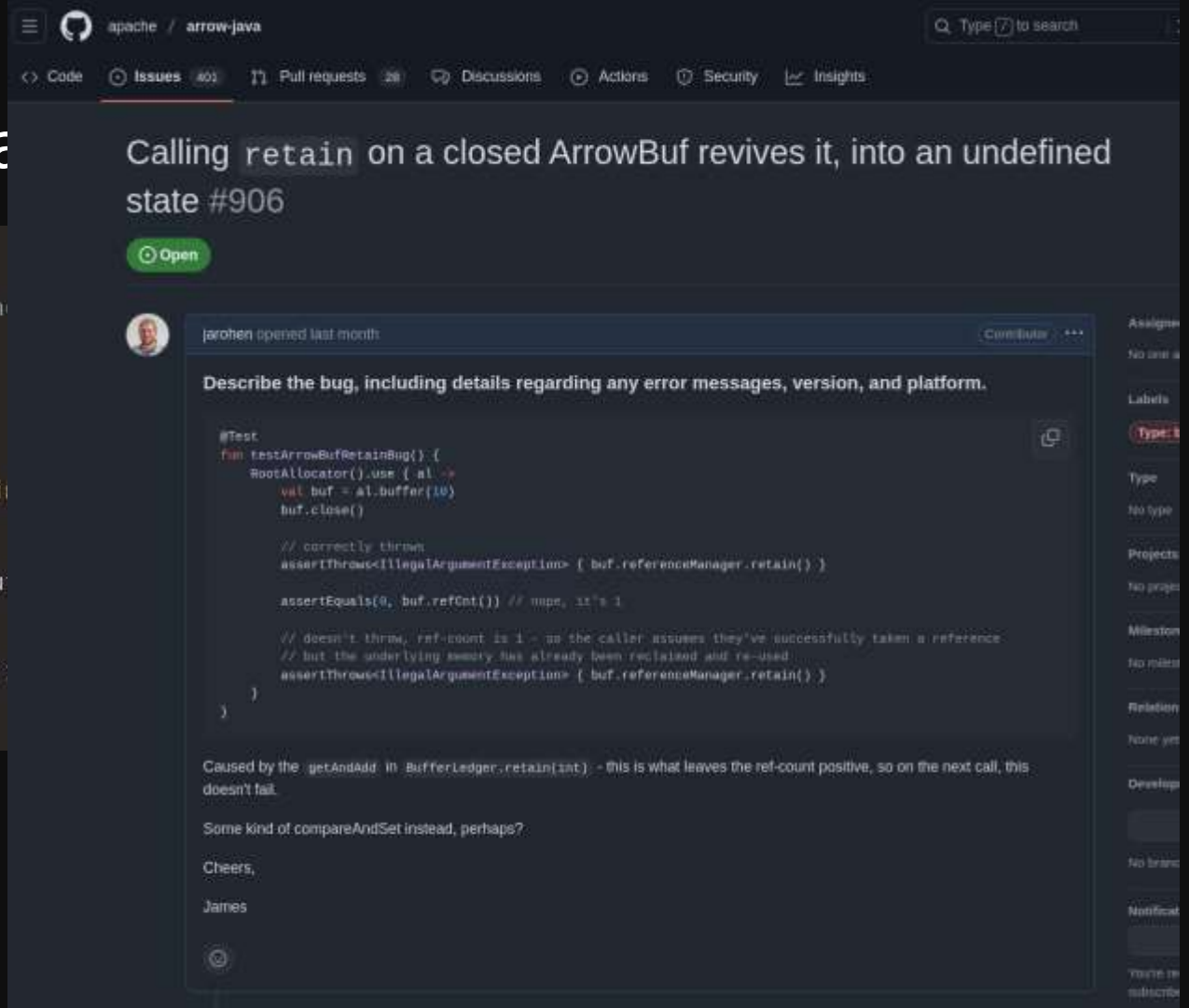
> every time there's a observable side-effect,
> give the Dispatcher chance to re-order.

```
interface Driver : AutoCloseable {
    suspend fun executeJob(job: Job): TriesAdded
    suspend fun appendMessage(triesAdded: TriesAdded): Log.MessageMetadata
}
```

- Mock `Driver` uses `yield()`'s to *ensure* suspension

# Spot the bug (Apa

```
/**
 * Increment the ledger's referen
 * by the given amount.
 */
fun retain(increment: Int) {
  require(increment > 0) { "retai

  val originalReferenceCount = bu

  require(originalReferenceCount
}
```



# Yeah, ok - but why's that a race?

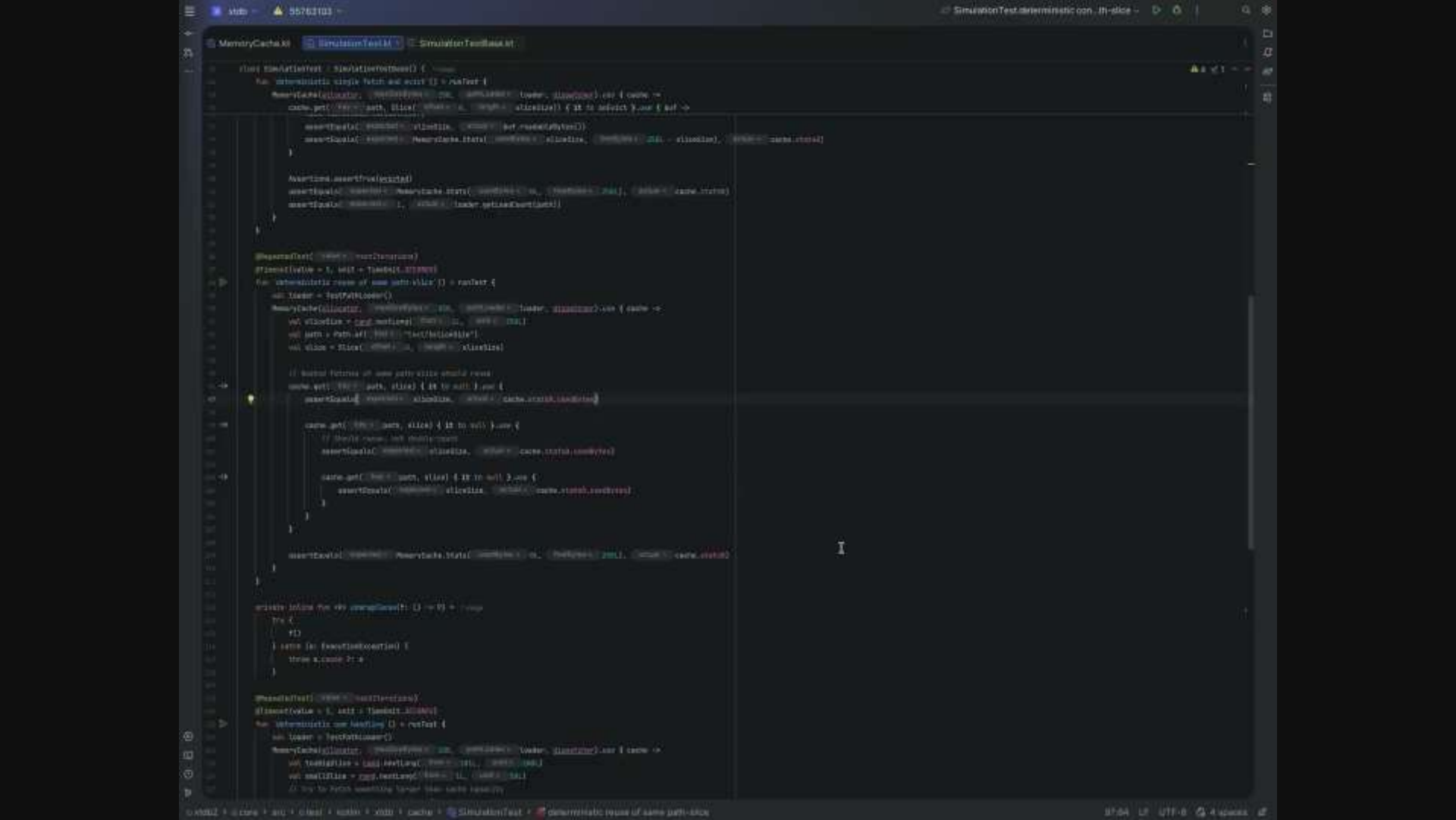
- Two threads trying to increment the ref-count:
  - Buffer ref-count → 0, fair game for cleanup.
  - T1 tries, (correctly) gets the exception - evicts it from our cache.
  - T2 tries at that exact moment - still in cache, but ***no exception*** (ref-count = 1, thanks T1)
  - T2 reads freed memory - gonna have a *bad time*.

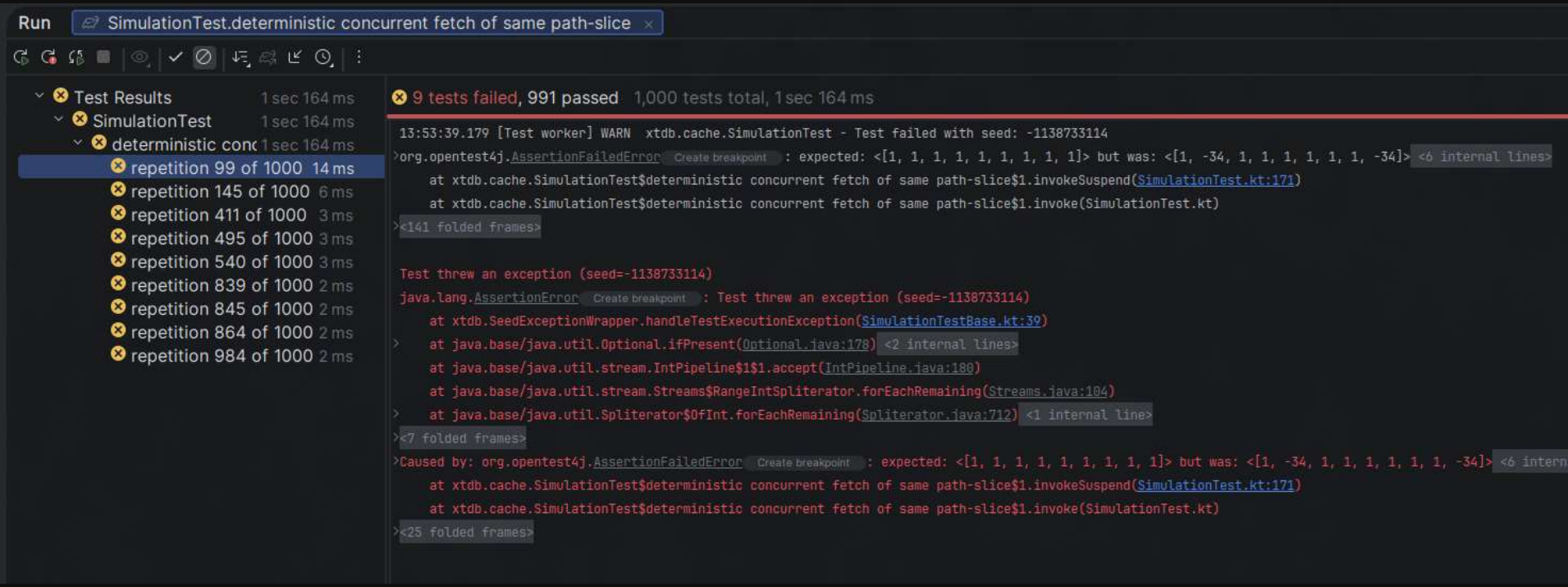
- Check the ref-count first, and then try to increase it?
  - That's also a race

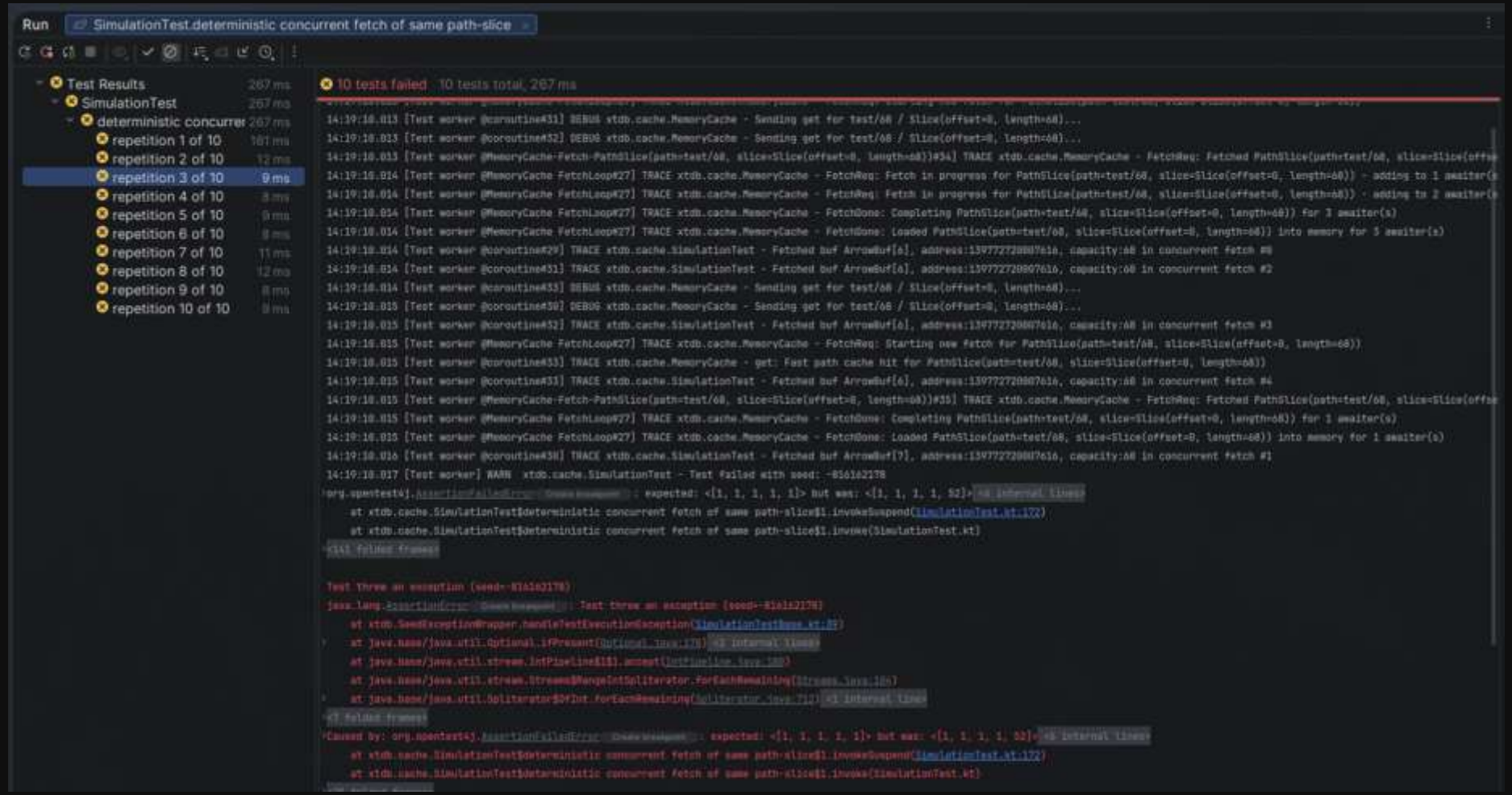
# Now for a test:

```kotlin
@RepeatableSimulationTest
fun `deterministic concurrent fetch of same path-slice`(iteration: Int) = runTest {
    MemoryCache(/* ... */).use { cache ->
        val path = Path.of("...")

        // Launch multiple concurrent fetches of the same path-slice
        val deferreds = async(dispatcher) {
            (1..5).map { i ->
                async(dispatcher) {
                    cache.get(path).use { buf ->
                        assertEquals(/* ... */)
                        yield()
                        buf.getByte(0)
                    }
                }
            }
        }.await()

        val results = deferreds.awaitAll()

        assertEquals(/* all results are the same */)
    }
}
```

```kotlin
class DeterministicDispatcher(private val rand: Random) : CoroutineDispatcher() {

    private val jobs = mutableSetOf<Runnable>()
    private var running = false

    override fun dispatch(context: CoroutineContext, block: Runnable) {
        jobs.add(block)

        if (!running) {
            running = true
            while (true) {
                val job = jobs.randomOrNull(rand) ?: break
                jobs.remove(job)
                job.run()
            }
            running = false
        }
    }
}
```




Run
SimulationTest.deterministic concurrent fetch of same path-slice
Test Results 1 sec 164 ms
SimulationTest 1 sec 164 ms
deterministic conc 1 sec 164 ms
repetition 99 of 1000 14 ms
repetition 145 of 1000 6 ms
repetition 411 of 1000 3 ms
repetition 495 of 1000 3 ms
repetition 540 of 1000 3 ms
repetition 839 of 1000 2 ms
repetition 845 of 1000 2 ms
repetition 864 of 1000 2 ms
repetition 984 of 1000 2 ms
9 tests failed, 991 passed 1,000 tests total, 1 sec 164 ms
13:53:39.179 [Test worker] WARN xtdb.cache.SimulationTest - Test failed with seed: -1138733114
org.opentest4j.AssertionFailedError Create breakpoint : expected: <[1, 1, 1, 1, 1, 1, 1, 1]> but was: <[1, -34, 1, 1, 1, 1, 1, -34]> <6 internal lines>
at xtdb.cache.SimulationTest$deterministic concurrent fetch of same path-slice$1.invokeSuspend(SimulationTest.kt:171)
at xtdb.cache.SimulationTest$deterministic concurrent fetch of same path-slice$1.invoke(SimulationTest.kt)
<141 folded frames>
Test threw an exception (seed=-1138733114)
java.lang.AssertionError Create breakpoint : Test threw an exception (seed=-1138733114)
at xtdb.SeedExceptionWrapper.handleTestExecutionException(SimulationTestBase.kt:39)
at java.base/java.util.Optional.ifPresent(Optional.java:178) <2 internal lines>
at java.base/java.util.stream.IntPipeline$1$1.accept(IntPipeline.java:180)
at java.base/java.util.stream.Streams$RangeIntSpliterator.forEachRemaining(Streams.java:104)
at java.base/java.util.Spliterator$OfInt.forEachRemaining(Spliterator.java:712) <1 internal line>
<7 folded frames>
Caused by: org.opentest4j.AssertionFailedError Create breakpoint : expected: <[1, 1, 1, 1, 1, 1, 1, 1]> but was: <[1, -34, 1, 1, 1, 1, 1, -34]> <6 intern
at xtdb.cache.SimulationTest$deterministic concurrent fetch of same path-slice$1.invokeSuspend(SimulationTest.kt:171)
at xtdb.cache.SimulationTest$deterministic concurrent fetch of same path-slice$1.invoke(SimulationTest.kt)
<25 folded frames>


Run
SimulationTest.deterministic concurrent fetch of same path-slice
Test Results 267 ms
SimulationTest 267 ms
deterministic concurrer 267 ms
repetition 1 of 10 181 ms
repetition 2 of 10 12 ms
repetition 3 of 10 9 ms
repetition 4 of 10 8 ms
repetition 5 of 10 9 ms
repetition 6 of 10 8 ms
repetition 7 of 10 11 ms
repetition 8 of 10 12 ms
repetition 9 of 10 8 ms
repetition 10 of 10 9 ms
10 tests failed 10 tests total, 267 ms
14:19:10.013 [Test worker @coroutine#31] DEBUG xtdb.cache.MemoryCache - Sending get for test/68 / Slice(offset=0, length=68)...
14:19:10.013 [Test worker @coroutine#52] DEBUG xtdb.cache.MemoryCache - Sending get for test/68 / Slice(offset=0, length=68)...
14:19:10.013 [Test worker @MemoryCache-Fetch-PathSlice(path=test/68, slice=Slice(offset=0, length=68))#54] TRACE xtdb.cache.MemoryCache - FetchReq: Fetched PathSlice(path=test/68, slice=Slice(offse
14:19:10.014 [Test worker @MemoryCache FetchLoop#27] TRACE xtdb.cache.MemoryCache - FetchReq: Fetch in progress for PathSlice(path=test/68, slice=Slice(offset=0, length=68)) - adding to 1 awaiter(s
14:19:10.014 [Test worker @MemoryCache FetchLoop#27] TRACE xtdb.cache.MemoryCache - FetchReq: Fetch in progress for PathSlice(path=test/68, slice=Slice(offset=0, length=68)) - adding to 2 awaiter(s
14:19:10.014 [Test worker @MemoryCache FetchLoop#27] TRACE xtdb.cache.MemoryCache - FetchDone: Completing PathSlice(path=test/68, slice=Slice(offset=0, length=68)) for 3 awaiter(s)
14:19:10.014 [Test worker @MemoryCache FetchLoop#27] TRACE xtdb.cache.MemoryCache - FetchDone: Loaded PathSlice(path=test/68, slice=Slice(offset=0, length=68)) into memory for 3 awaiter(s)
14:19:10.014 [Test worker @coroutine#29] TRACE xtdb.cache.SimulationTest - Fetched buf ArrowBuf[6], address:139772720807616, capacity:68 in concurrent fetch #0
14:19:10.014 [Test worker @coroutine#51] TRACE xtdb.cache.SimulationTest - Fetched buf ArrowBuf[6], address:139772720807616, capacity:68 in concurrent fetch #2
14:19:10.014 [Test worker @coroutine#33] DEBUG xtdb.cache.MemoryCache - Sending get for test/68 / Slice(offset=0, length=68)...
14:19:10.015 [Test worker @coroutine#30] DEBUG xtdb.cache.MemoryCache - Sending get for test/68 / Slice(offset=0, length=68)...
14:19:10.015 [Test worker @coroutine#52] TRACE xtdb.cache.SimulationTest - Fetched buf ArrowBuf[6], address:139772720807616, capacity:68 in concurrent fetch #3
14:19:10.015 [Test worker @MemoryCache FetchLoop#27] TRACE xtdb.cache.MemoryCache - FetchReq: Starting new fetch for PathSlice(path=test/68, slice=Slice(offset=0, length=68))
14:19:10.015 [Test worker @coroutine#53] TRACE xtdb.cache.MemoryCache - get: Fast path cache hit for PathSlice(path=test/68, slice=Slice(offset=0, length=68))
14:19:10.015 [Test worker @coroutine#31] TRACE xtdb.cache.SimulationTest - Fetched buf ArrowBuf[6], address:139772720807616, capacity:68 in concurrent fetch #4
14:19:10.015 [Test worker @MemoryCache-Fetch-PathSlice(path=test/68, slice=Slice(offset=0, length=68))#35] TRACE xtdb.cache.MemoryCache - FetchReq: Fetched PathSlice(path=test/68, slice=Slice(offse
14:19:10.015 [Test worker @MemoryCache FetchLoop#27] TRACE xtdb.cache.MemoryCache - FetchDone: Completing PathSlice(path=test/68, slice=Slice(offset=0, length=68)) for 1 awaiter(s)
14:19:10.015 [Test worker @MemoryCache FetchLoop#27] TRACE xtdb.cache.MemoryCache - FetchDone: Loaded PathSlice(path=test/68, slice=Slice(offset=0, length=68)) into memory for 1 awaiter(s)
14:19:10.016 [Test worker @coroutine#30] TRACE xtdb.cache.SimulationTest - Fetched buf ArrowBuf[7], address:139772720807616, capacity:68 in concurrent fetch #1
14:19:10.017 [Test worker] WARN xtdb.cache.SimulationTest - Test failed with seed: -816162178
org.opentest4j.AssertionFailedError: expected: <[1, 1, 1, 1, 1]> but was: <[1, 1, 1, 1, 52]>
at xtdb.cache.SimulationTest$deterministic concurrent fetch of same path-slice$1.invokeSuspend(SimulationTest.kt:172)
at xtdb.cache.SimulationTest$deterministic concurrent fetch of same path-slice$1.invoke(SimulationTest.kt)
Test Threw an exception (seed=-816162178)
java.lang.AssertionError: Test threw an exception (seed=-816162178)
at xtdb.SeedExceptionWrapper.handleTestExecutionException(SimulationTestBase.kt:87)
at java.base/java.util.Optional.ifPresent(Optional.java:178)
at java.base/java.util.stream.IntPipeline$1$1.accept(IntPipeline.java:180)
at java.base/java.util.stream.Streams$RangeIntSpliterator.forEachRemaining(Streams.java:104)
at java.base/java.util.Spliterator$OfInt.forEachRemaining(Spliterator.java:711)
Caused by: org.opentest4j.AssertionFailedError: expected: <[1, 1, 1, 1, 1]> but was: <[1, 1, 1, 1, 52]>
at xtdb.cache.SimulationTest$deterministic concurrent fetch of same path-slice$1.invokeSuspend(SimulationTest.kt:172)
at xtdb.cache.SimulationTest$deterministic concurrent fetch of same path-slice$1.invoke(SimulationTest.kt)

# The fix

```
/**
 * Increment the ledger's reference count for associated underlying memory chunk
 * by the given amount.
 */
fun retain(increment: Int) {
  require(increment > 0) { "retain($increment) argument is not positive" }

  while (true) {
    val currentRefCount = bufRefCnt.get()
    check(currentRefCount > 0)
    if (bufRefCnt.compareAndSet(currentRefCount, currentRefCount + increment))
      return
  }
}
```

Run SimulationTest.deterministic concurrent fetch of same path-slice

Test Results 1 sec 229 ms
SimulationTest 1 sec 229 ms
deterministic concu 1 sec 229 ms
repetition 1 of 1000 185 ms
repetition 2 of 1000 4 ms
repetition 3 of 1000 4 ms
repetition 4 of 1000 7 ms
repetition 5 of 1000 5 ms
repetition 6 of 1000 4 ms
repetition 7 of 1000 5 ms
repetition 8 of 1000 4 ms
repetition 9 of 1000 4 ms
repetition 10 of 1000 5 ms
repetition 11 of 1000 4 ms
repetition 12 of 1000 4 ms
repetition 13 of 1000 4 ms
repetition 14 of 1000 4 ms
repetition 15 of 1000 4 ms
repetition 16 of 1000 3 ms
repetition 17 of 1000 3 ms
repetition 18 of 1000 3 ms
repetition 19 of 1000 2 ms
repetition 20 of 1000 2 ms
repetition 21 of 1000 3 ms
repetition 22 of 1000 4 ms
repetition 23 of 1000 4 ms
repetition 24 of 1000 4 ms
repetition 25 of 1000 4 ms
repetition 26 of 1000 3 ms
repetition 27 of 1000 2 ms
repetition 28 of 1000 2 ms
repetition 29 of 1000 2 ms
repetition 30 of 1000 2 ms
repetition 31 of 1000 2 ms
repetition 32 of 1000 2 ms
repetition 33 of 1000 3 ms

1,000 tests passed 1,000 tests total, 1 sec 229 ms

xtdb2 › core › src › test › kotlin › xtdb › cache › SimulationTest.kt

# Closing thoughts

- FP Best Practices™ are still as true as ever
  - Immutability by default
  - Separate data and behaviour
  - Separate pure from side-effecting code, test them in isolation → simpler tests
- “Communicating sequential processes” / “actors” have been a boon
  - If you’re going to have mutable state, definitely don’t have *shared* mutable state
- … but if you do have shared mutable state, then DST

# Thank you
# Merry Christmas!

...

@jms (JUXT)
@jhenderson (Grid)
@jarohen (everywhere else)