Stormpot

Stormpot – Object pooling for Java

chrisvest.github.com/stormpot/

Reader

Fork me on GitHub

Stormpot by Chris Vest

Generic object pooling for Java

Stormpot is an object pooling library for Java. Use it to recycle objects that are expensive to create. The library will take care of creating and destroying your objects in the background.

The library is distributed under the Apache 2.0 license. You can get the source, read the documentation and report issues.

```
<dependency>
  <groupId>com.github.chrisvest</groupId>
  <artifactId>stormpot</artifactId>
  <version>2.0</version>
</dependency>
```

Why choose Stormpot?

… over, say, Commons-Pool? Good question! Both libraries are released under the Apache 2.0 license, both have thorough documentation, both have high test coverage (Stormpot have 100%) and both depend on nothing but Java itself.

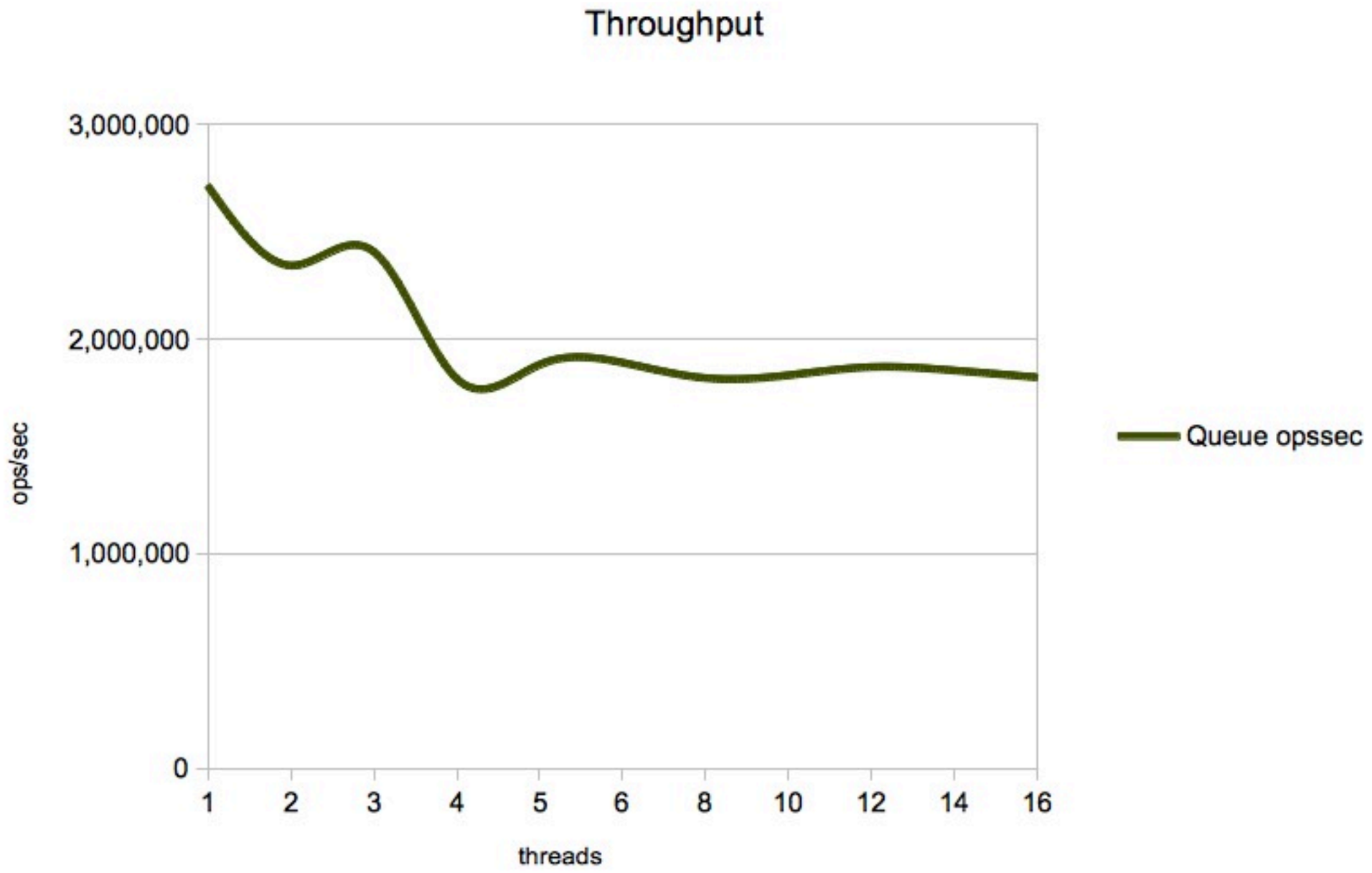
There are differences, though. Stormpot has an invasive but small API, whereas Commons-Pool has a richer and less intrusive API. The different slants in their API design comes from differing focus. Stormpot is designed for performance and a unified API for different implementations. This means that those who use Stormpot can switch between implementations without worrying about compatibility. Commons-Pool, on the other hand, have different APIs for different kinds of pools, so changing pool implementation might mean also having to change the code that uses the pool. On the other hand, the various pool implementations can make special features available that would otherwise be inconvenient to expose through a generic API.

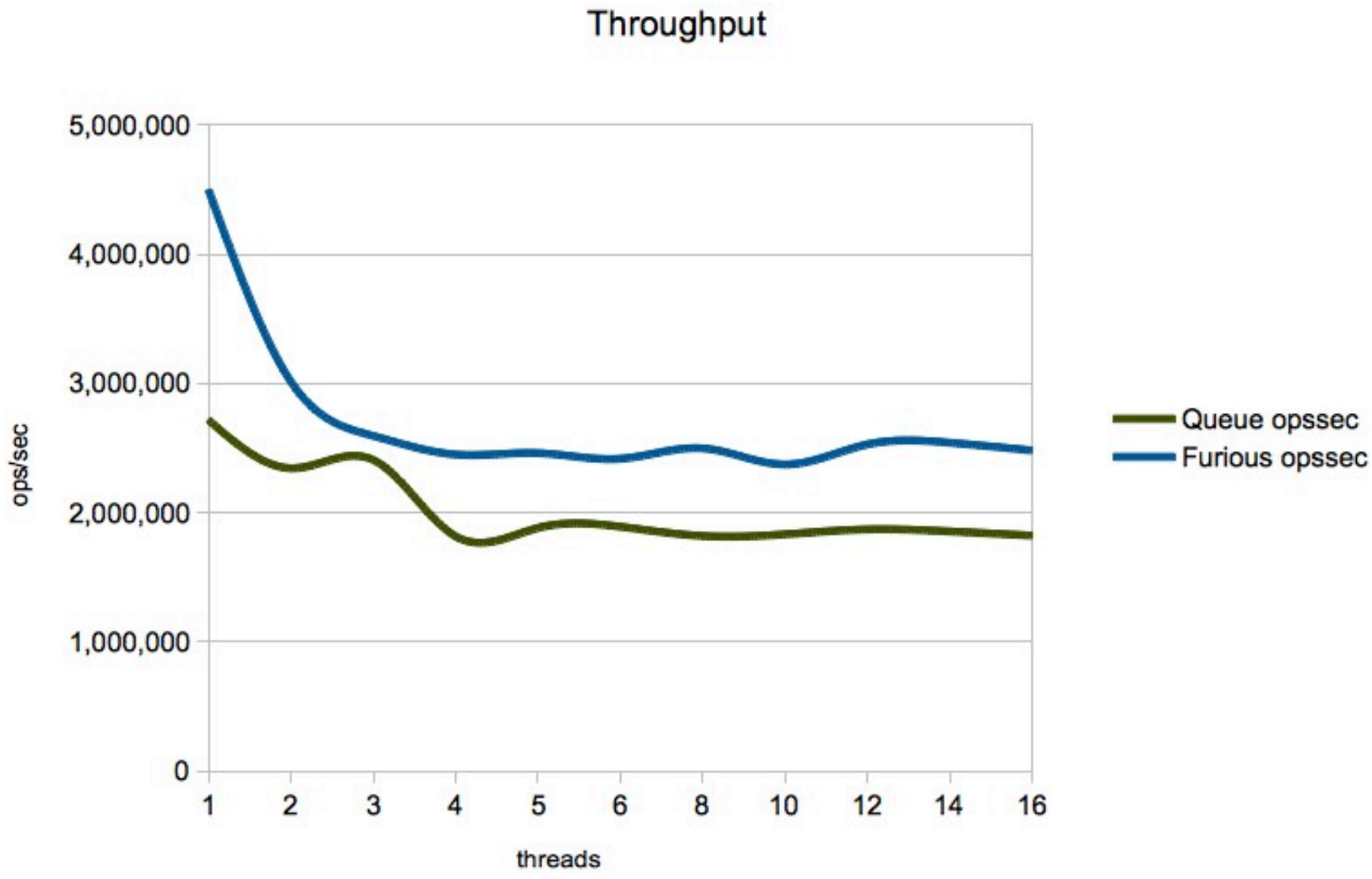
Getting started

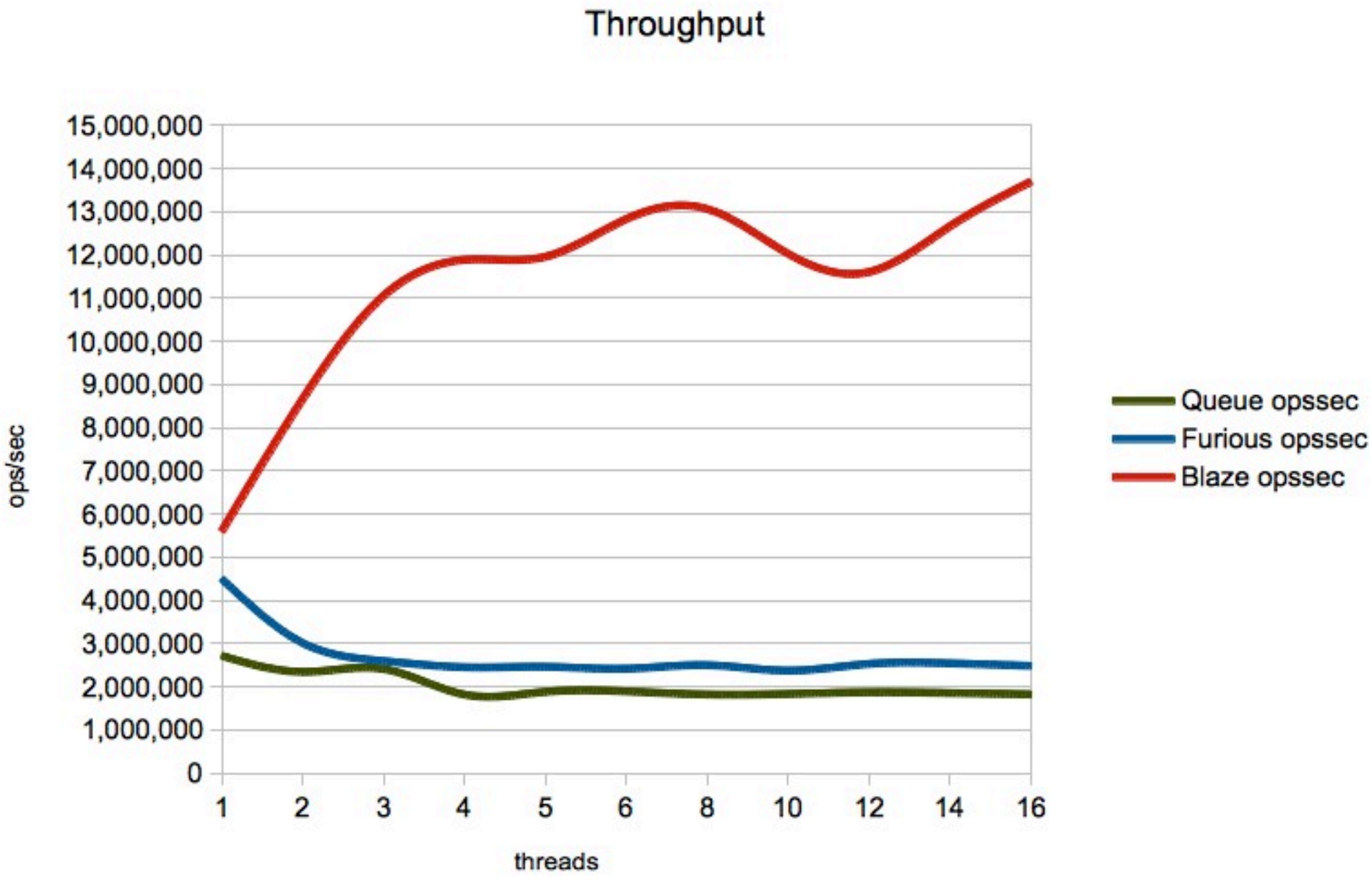
You have decided to use Stormpot for something and want to get started. In this section, I will show you the minimum amount of code it takes, to get started with Stormpot. I mentioned above that Stormpot has an invasive API. This means that there are things you need to do to your code before you can use it. Specifically, there are two interfaces that you need to implement: Poolable, for the objects that you want pooled, and Allocator, which is a kind of factory for your Poolables. Their minimum correct implementations look like this:

```
// MyPoolable.java - minimum Poolable implementation
import stormpot.Poolable;
import stormpot.Slot;

public class MyPoolable implements Poolable {
  private final Slot slot;
  public MyPoolable(Slot slot) {
    this.slot = slot;
  }

  public void release() {
    slot.release(this);
  }
}
```

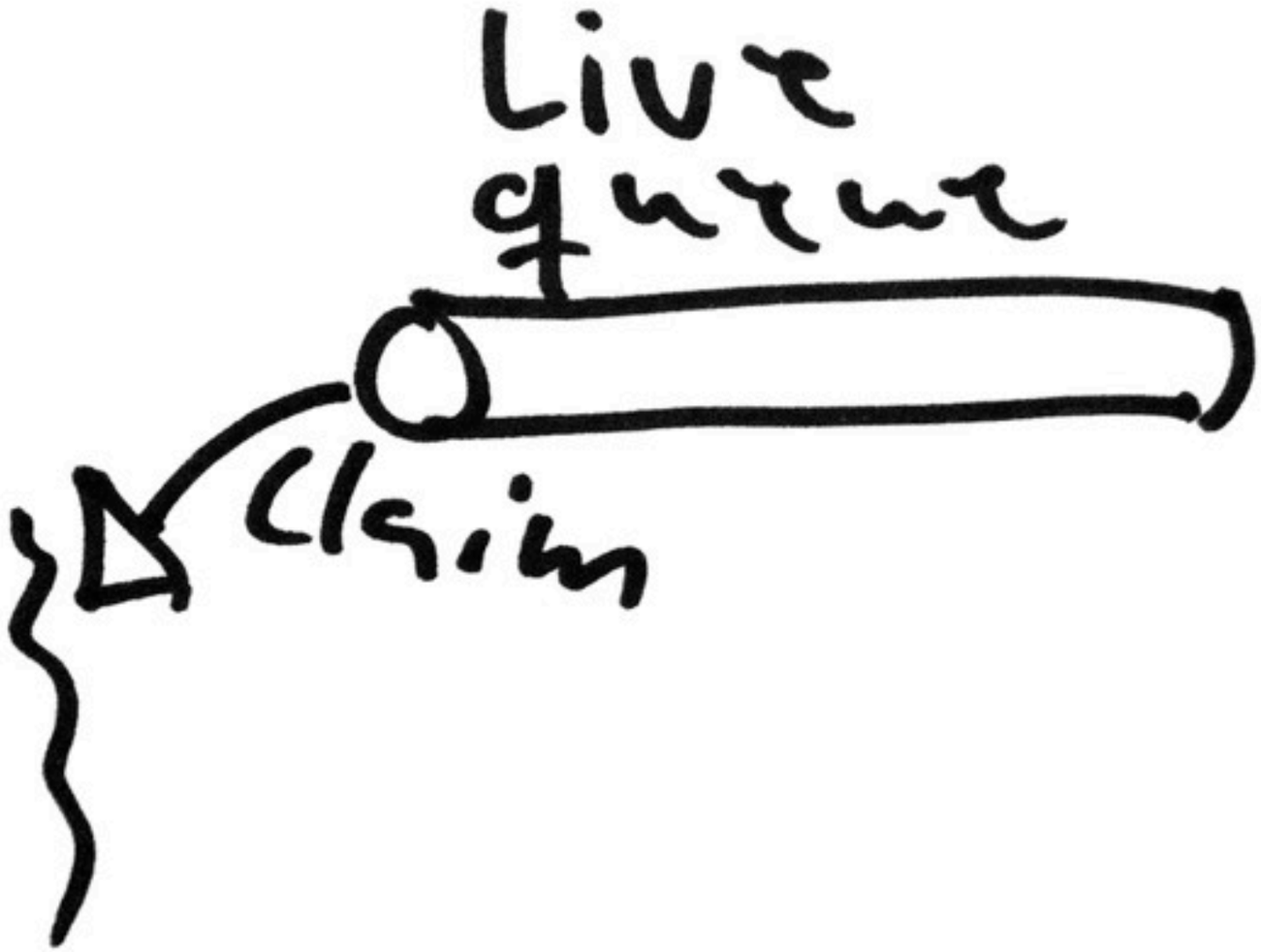
```
// MyAllocator.java - minimum Allocator implementation
import stormpot.Allocator;
import stormpot.Slot;

public class MyAllocator implements Allocator<MyPoolable> {
  public MyPoolable allocate(Slot slot) throws Exception {
    return new MyPoolable(slot);
  }

  public void deallocate(MyPoolable poolable) throws Exception {
    // Nothing to do here
    // But it's a perfect place to close sockets, files, etc.
  }
}
```


Throughput
3,000,000
2,000,000
1,000,000
0
ops/sec
1
2
3
4
5
6
8
10
12
14
16
threads
Queue opssec


Throughput
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
ops/sec
1
2
3
4
5
6
8
10
12
14
16
threads
Queue opssec
Furious opssec


Throughput
15,000,000
14,000,000
13,000,000
12,000,000
11,000,000
10,000,000
9,000,000
8,000,000
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
ops/sec
1
2
3
4
5
6
8
10
12
14
16
threads
Queue opssec
Furious opssec
Blaze opssec

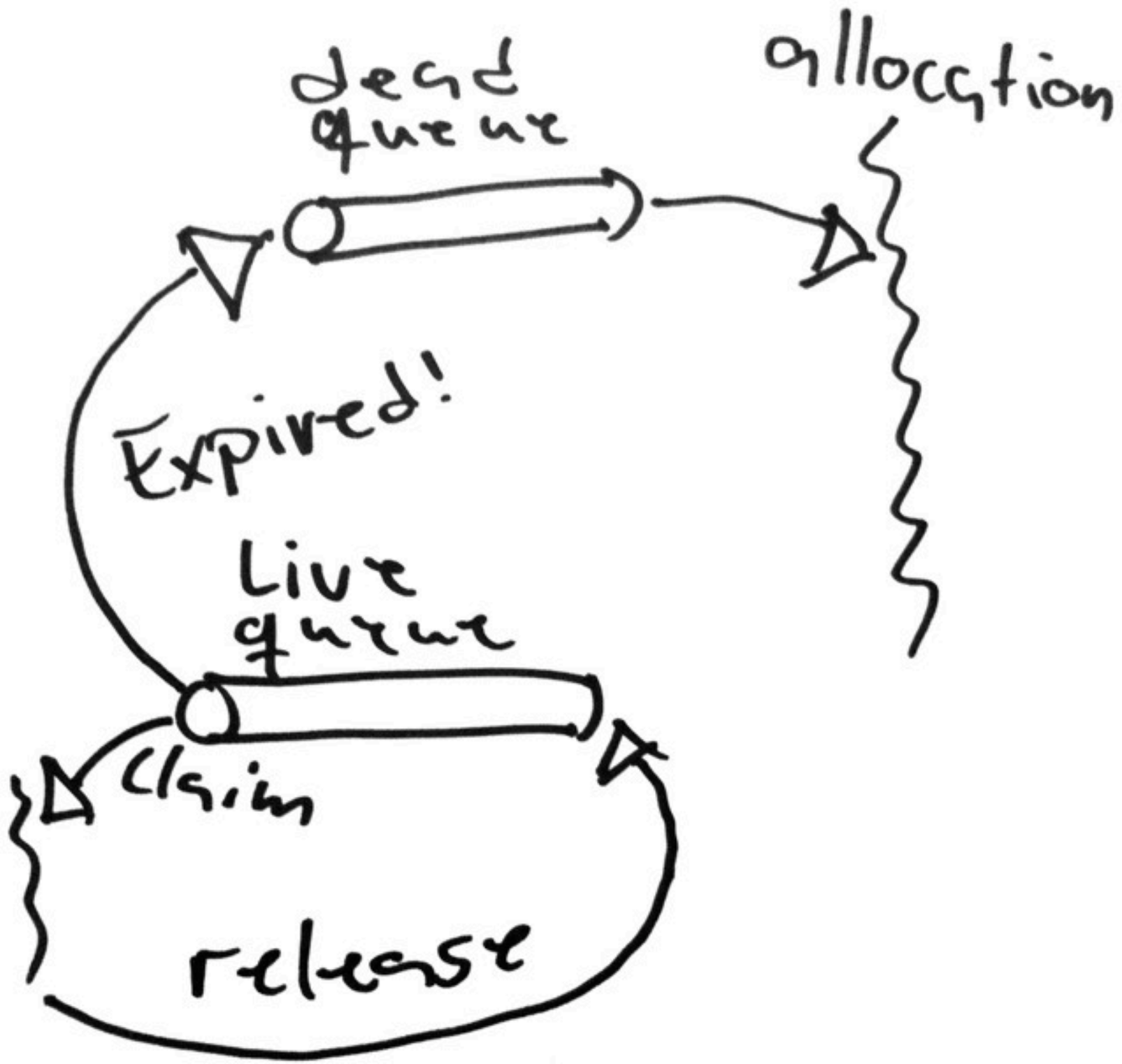
QueuePool

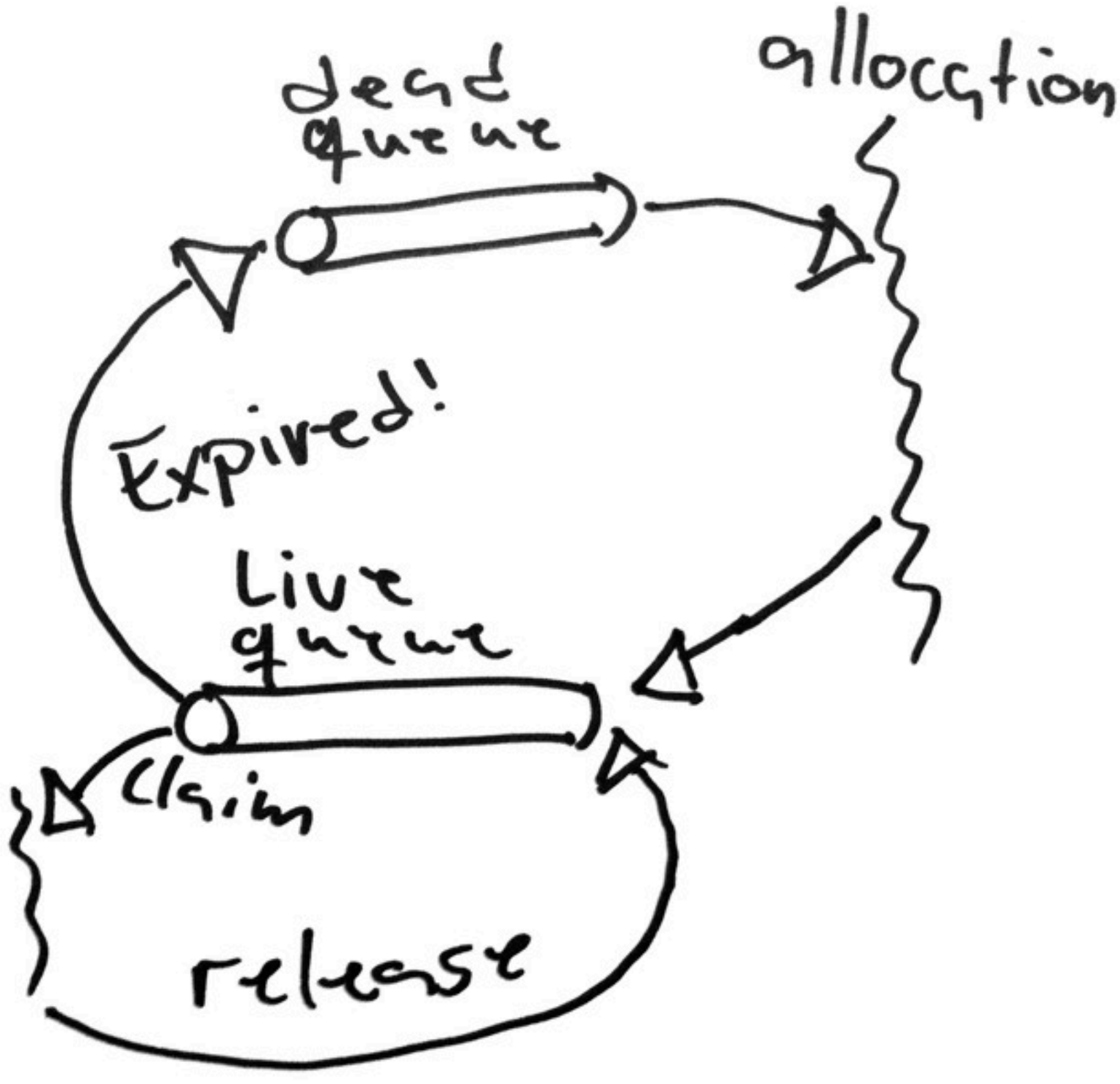

Live
queue
Claim


Live
queue
claim
release

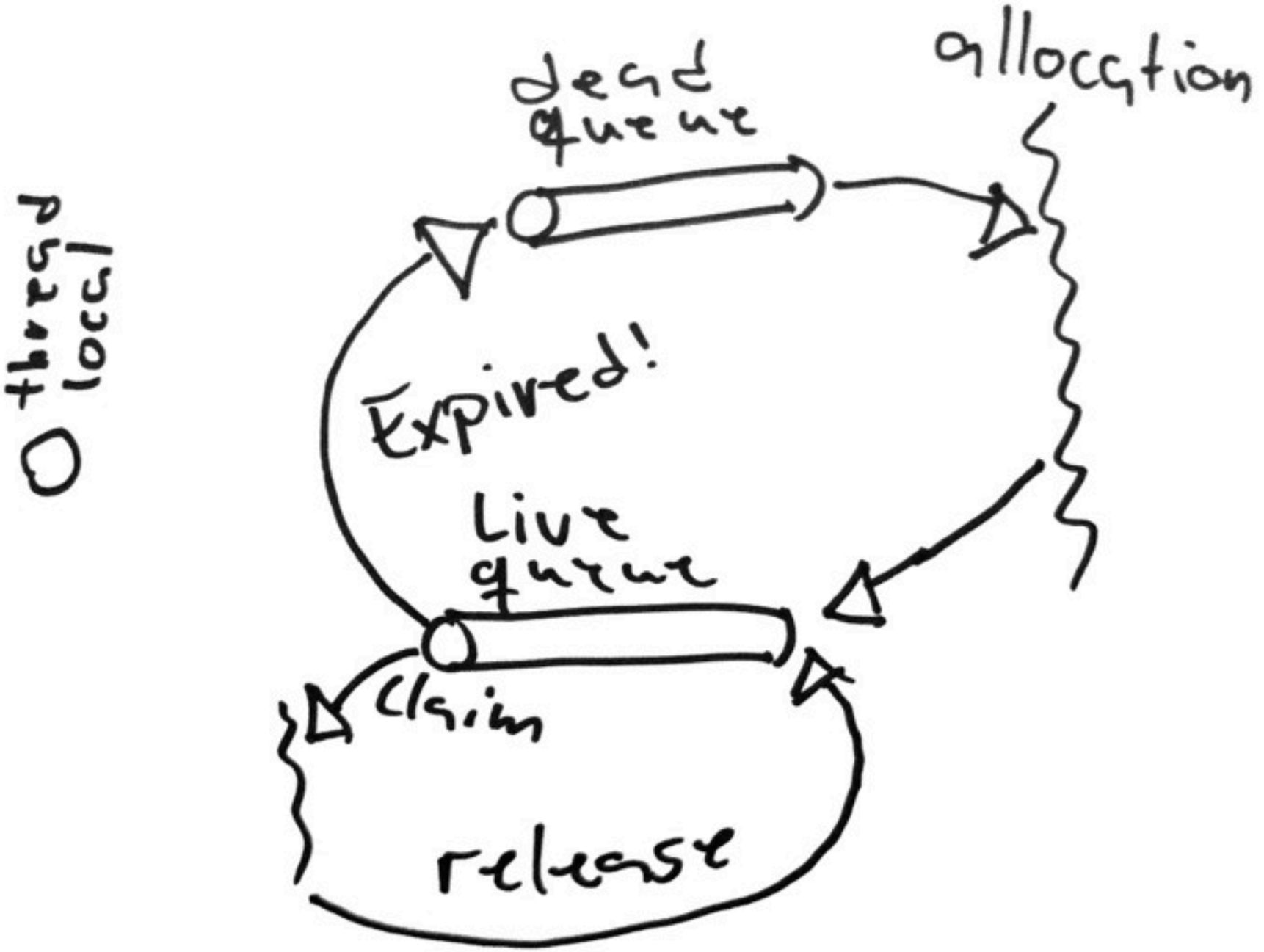

Expired!
Live
queue
claim
release

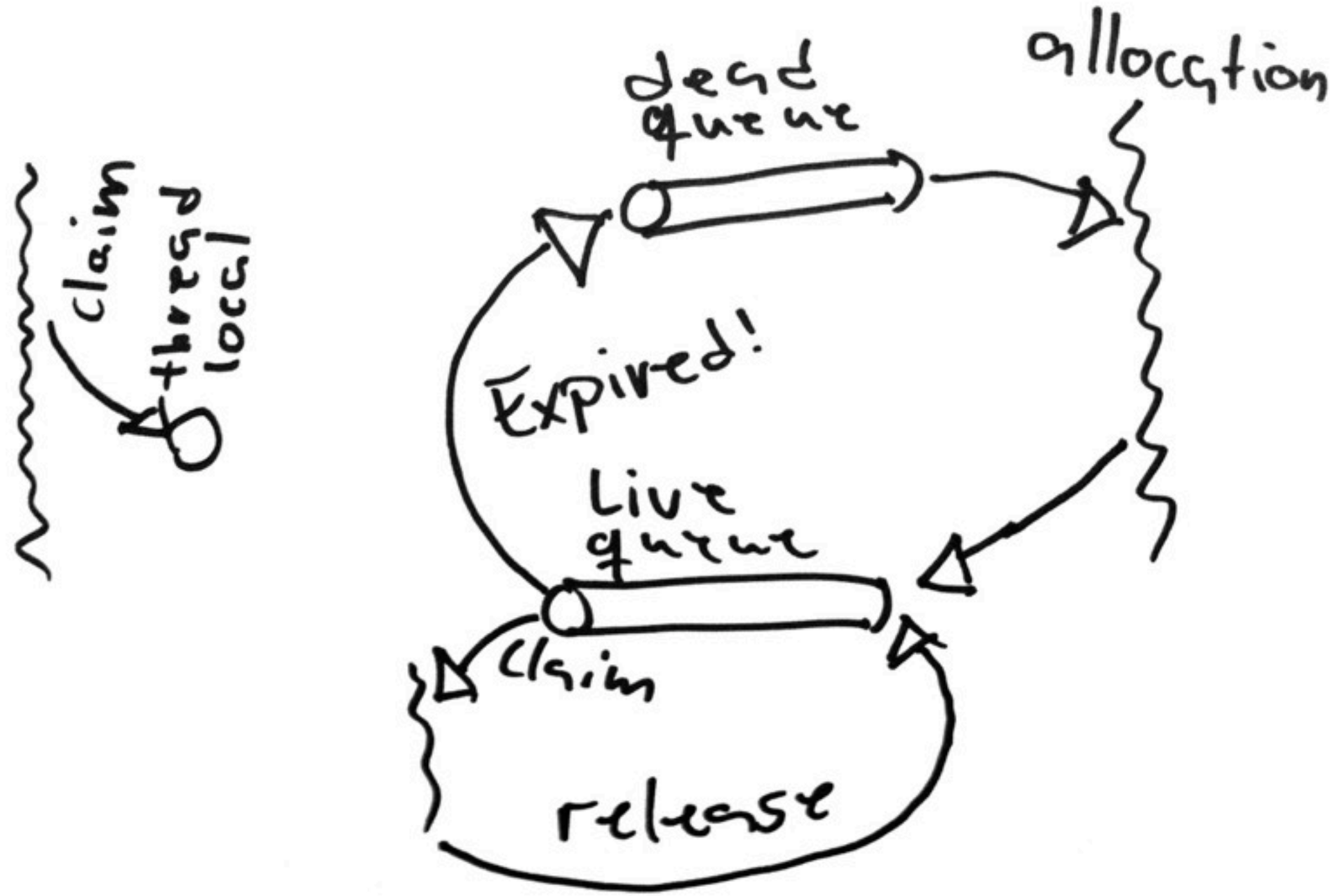

dead
queue
Expired!
live
queue
claim
release

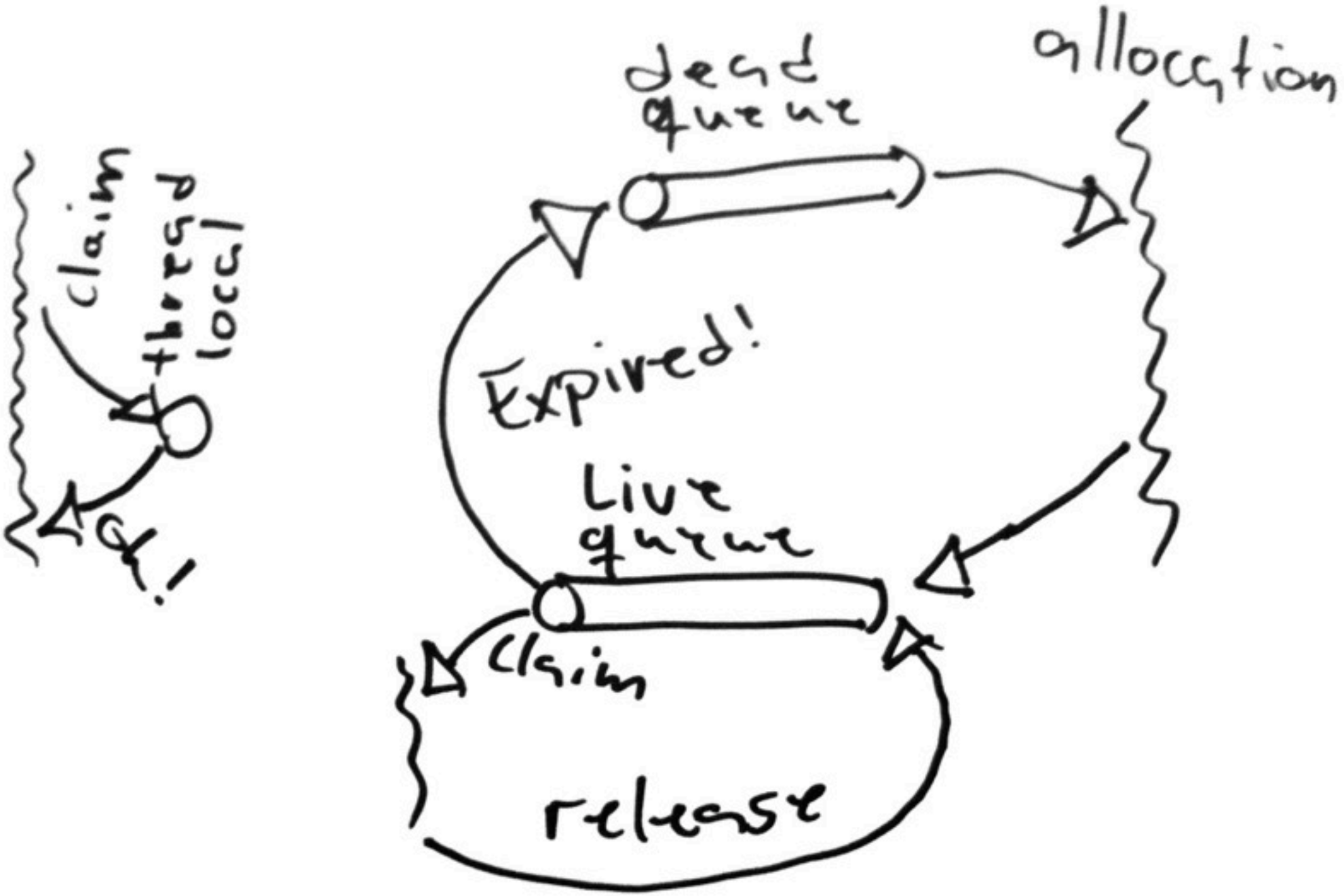

dead
queue
allocation
Expired!
Live
queue
Claim
release

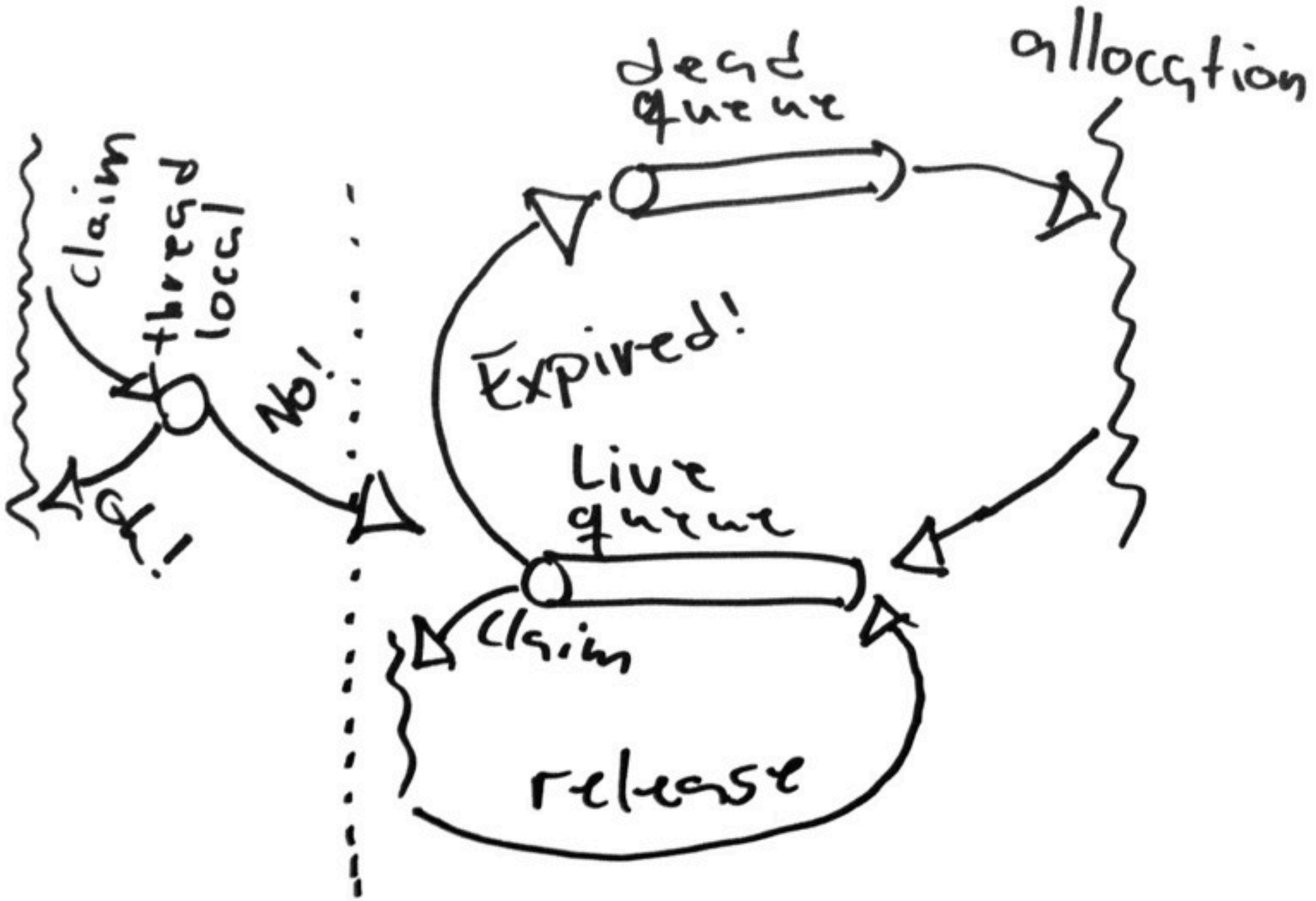

dead
queue
allocation
Expired!
Live
queue
Claim
release

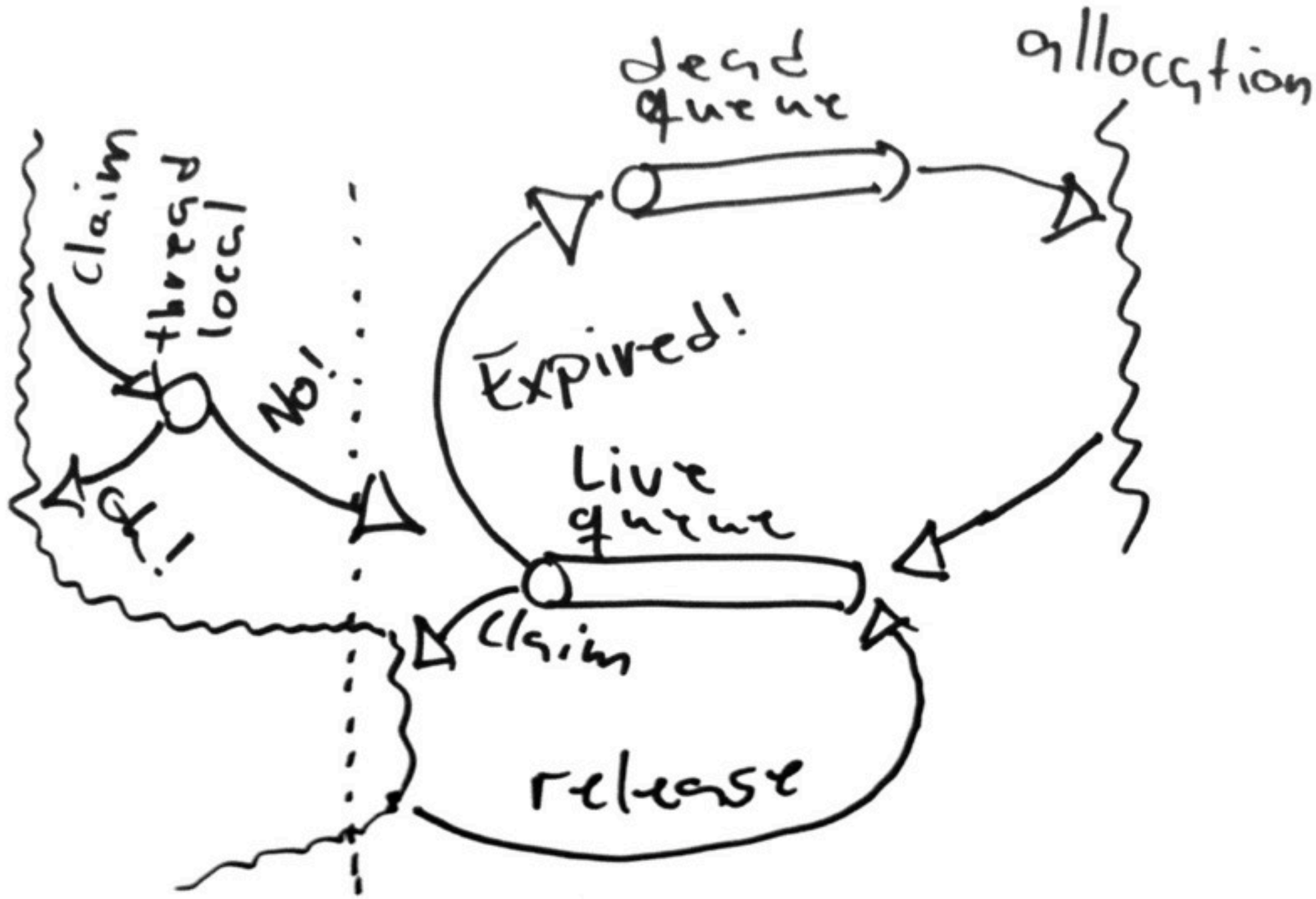
BlazePool


dead queue
allocation
Expired!
Live queue
claim
release


claim
thread
local
dead
queue
allocation
Expired!
Live
queue
claim
release


claim
thread local
ok!
dead queue
allocation
Expired!
Live queue
claim
release


claim
thread local
No!
Ok!
dead queue
allocation
Expired!
Live queue
claim
release


dead queue
allocation
claim
thread local
No!
Expired!
Live queue
claim
release

Contention

=

Slow

Stripe memory access!

if you need the performance
... and you can afford the complexity!

?