Scaling Java Applications Through Concurrency

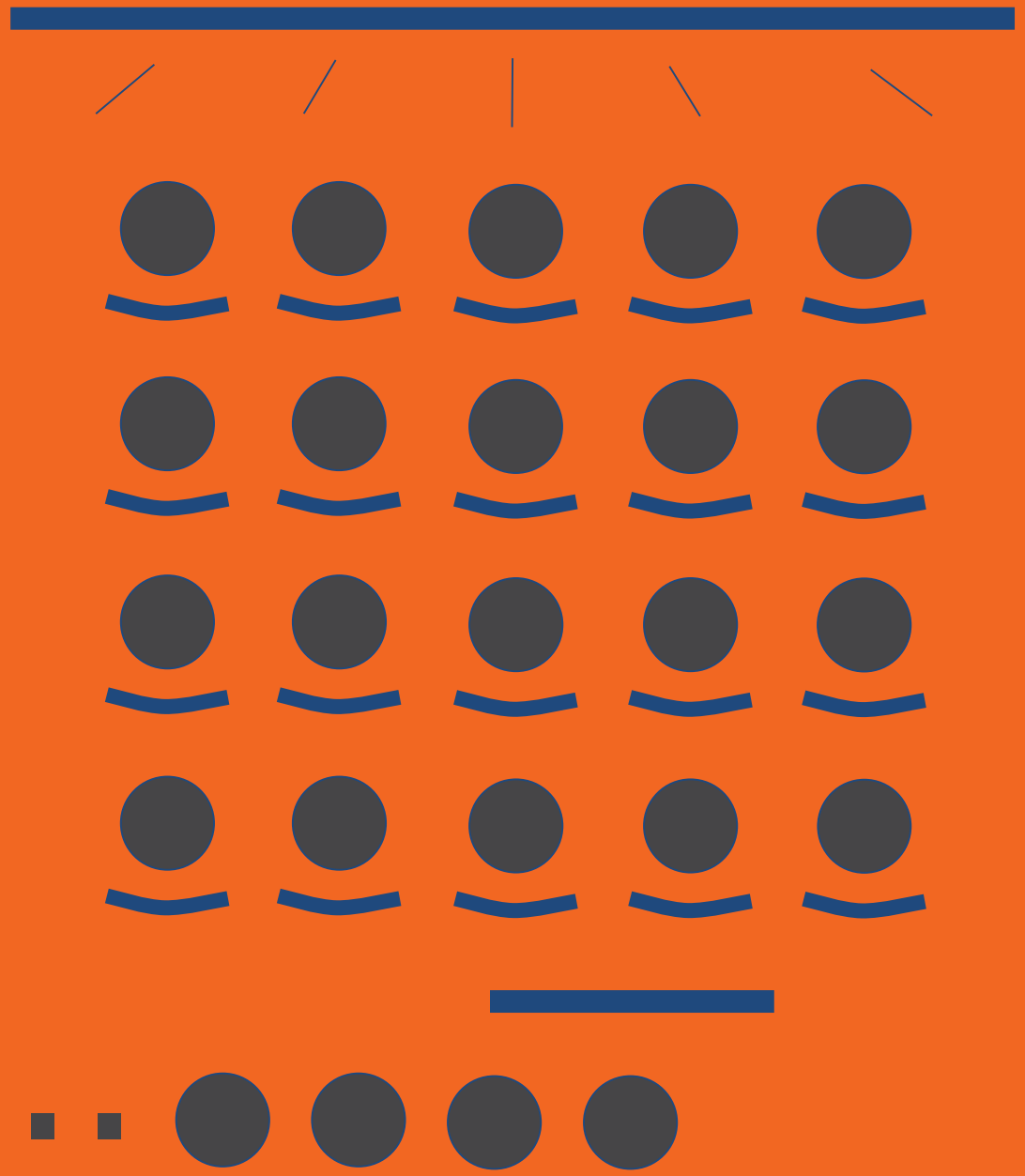
Throttling Incoming Work

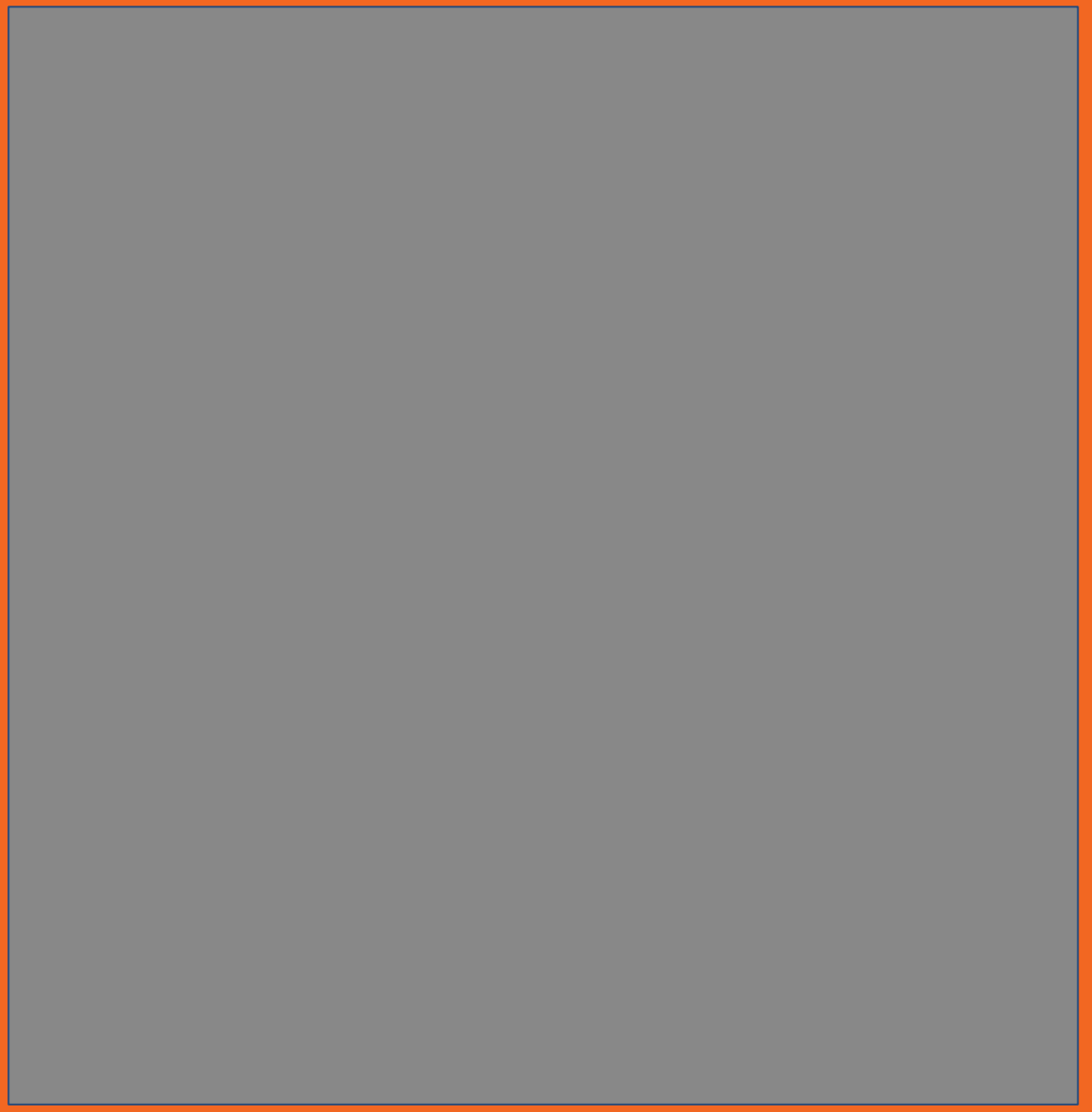
Josh Cummings

@jzheaux | tech.joshcummings.com

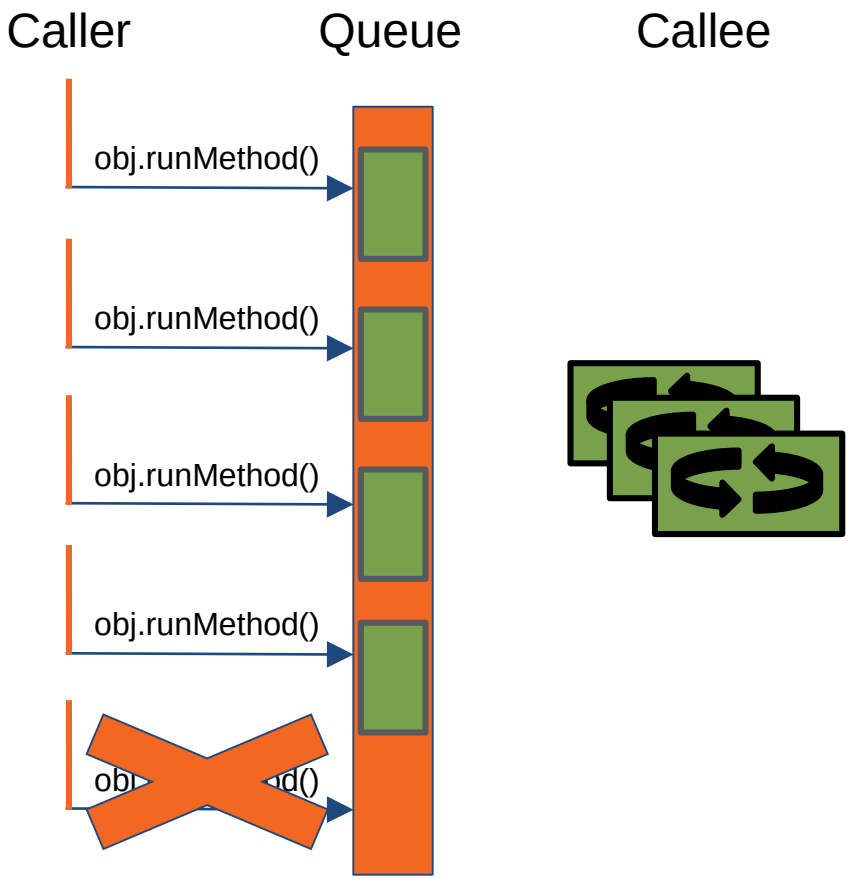
Identity Pipeline: Throttling Work

- Reads in a stream of identity-related data
- **Normalizes and verifies the contents of each identity**
- Merges and persists the identities in an in-memory cache
- Notifies an error queue if any of the above fails
- Records aggregate identity statistics





The Backpressure Pattern



- Service rejects requests after bounded queue fills up
- Client decides what to do at that point; try again or failover to a different service
- Queueing Theory provides a mathematical model (see Little's Law)

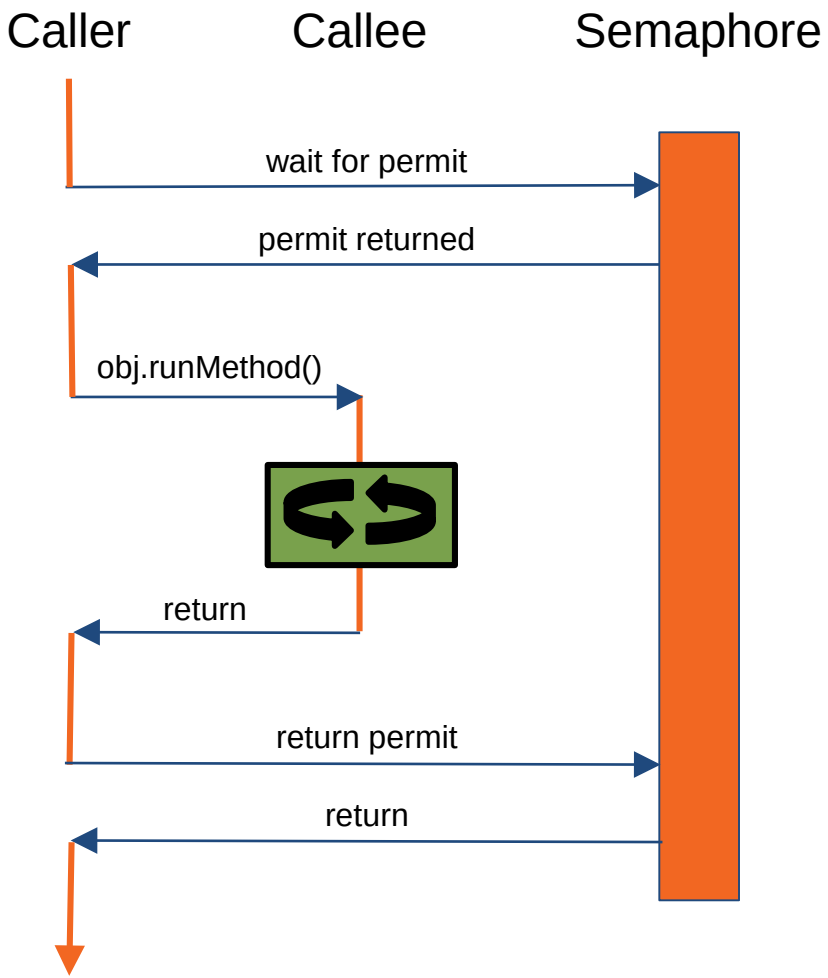
ThreadPoolExecutor, part II

- Configurable for backpressure by supplying an ArrayBlockingQueue where the capacity is the in-flight max
- Defaults to throw a RejectedExecutionException if no handler is provided
- Max and core pool size can be throttled at runtime

```
// limit the length of the pool's internal queue
// and configure a rejection handler for when
// backpressure is applied

ThreadPoolExecutor pool =
        new ThreadPoolExecutor(5, 5, -1,
                TimeUnit.MILLISECONDS,
                new
ArrayBlockingQueue<>(capacity),
                new
RejectionExecutionHandler() {
                        public void
rejectedExecution(...) {
                                // fail/try
another service
                        }
                });
```

Demo

The Semaphore Pattern



- Semaphore has as a given number of permits that can be issued on its lock
- Each client blocks at a “ticket booth”, waiting for a permit
- Only a given number of callers can invoke the method at a time

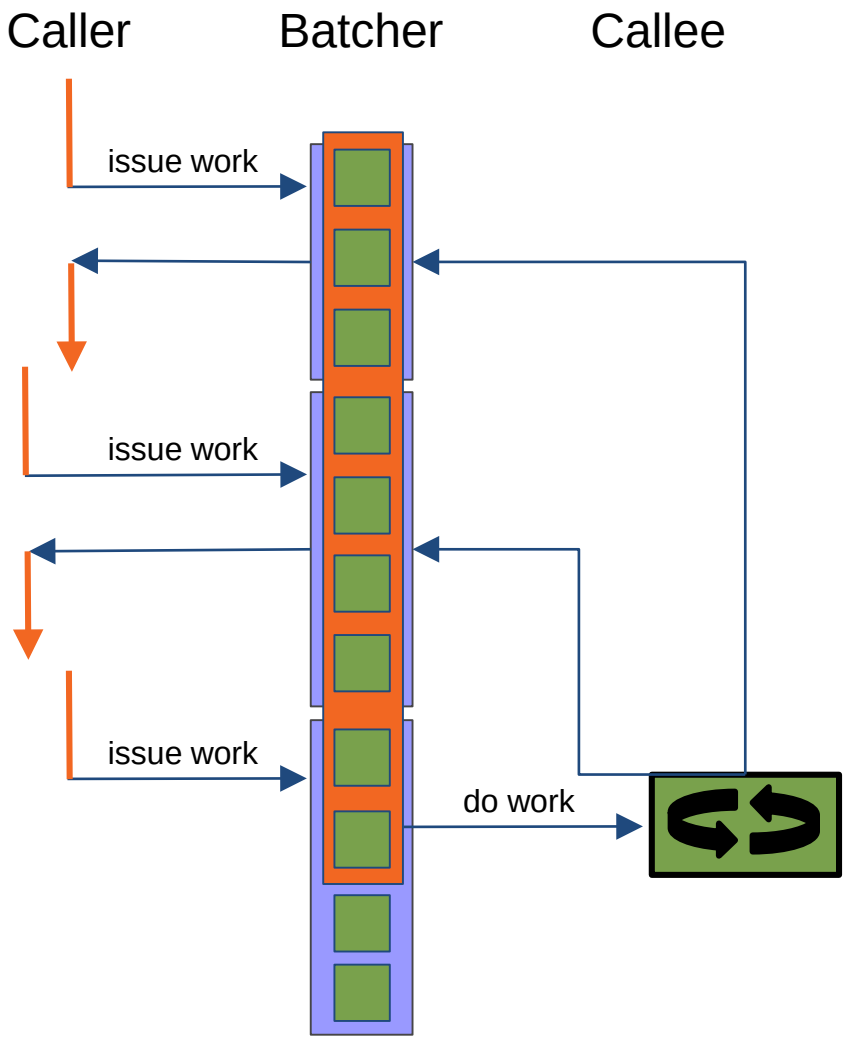
Semaphore

- Similar *happens-before* semantics to ReentrantLock, just allowing more than one thread to obtain the lock at once
- Similar acquisition and release patterns to ReentrantLock, like trial locking and timed waits
- Number of available permits can be changed at runtime if Semaphore is subclassed

```
// ask for a permit to call the method, and
then
// call it

Semaphore ticketBooth = new
Semaphore(5);


public void verifyAddresses(List addresses) {
        ticketBooth.acquire();
        try {

        delegate.verifyAddresses(addresses)
;
        } finally {
                ticketBooth.release();
        }
}
```

Demo

The Batching Pattern



- Batch according to a specific size with a timeout
- Callers can ignore, block, or provide callback to continue once their payload has been processed
- Like Backpressure, model the traffic in order to get optimal batch size and timeout

CyclicBarrier

- Like CountDownLatch, but reusable
- New arrivals should block
- A runnable can be specified to run each time the barrier is broken
- Useful when repeated groups of multiple threads must wait for one another

```
// guests explore the museum, the guide
// responds once all are done

CyclicBarrier tour = new
CyclicBarrier(numberOfGuests,this::alertGuid
e);


public void arrive(Guest guest) {
            // explore room
      tour.await(3000); // wait for others
}

public void alertGuide() {
            guide.wakeup(this); // guide to next
room
}           }
}
```

Demo

CyclicBarrier

CountDownLatch

Phaser

Phaser

- Arrivals may choose to block and wait or simply announce their arrival
- Separates the concepts of registration and arrival to allow for myriad sophisticated completion patterns
- Supports tiers of Phasers to reduce contention in cases of a large number of registrants

```
// guests arrive at each table, the waiter
// responds as each table fills up

Phaser waiter =
        new Phaser(numberOfGuests + 1);


public void arrive(Guest guest) {
        waiter.arrive(); // sit at table, etc...
}

public void arrive(Waiter waiter) {
        waiter.arriveAndAwaitAdvance();
    // now take orders...
}
```

```
        }
}
```

Demo

Review

- Applications can be made more resilient by considering arrival rate, concurrency limits, and bulk requests
- The Java Concurrency API offers several classes that can cap arrival rates in order to increase overall throughput
- Whatever solution is chosen, modeling actual traffic levels is critical