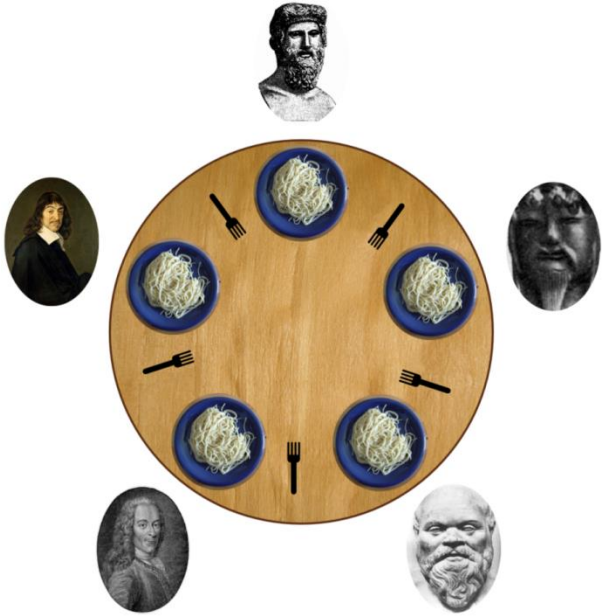
Concurrency in practice



Mikalai Alimenkou

http://xpinjection.com

@xpinjection

Mikalai Alimenkou

@xpinjection

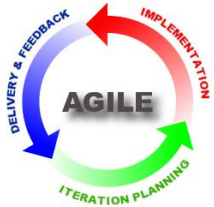
Happy father, Java Tech Lead, ScrumMaster, XP Injection founder, speaker, Agile/XP Coach, conference organizer: @jeeconf, @seleniumcamp, @xpdays_ua, @itbrunch.

Ukraine, Kiev · http://xpinjection.com

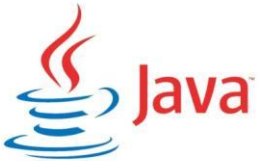
13,550 TWEETS

51 FOLLOWING

1,219 FOLLOWERS



8 YEARS, EXPERT



10 YEARS, TECH LEAD

FOUNDER AND COACH:

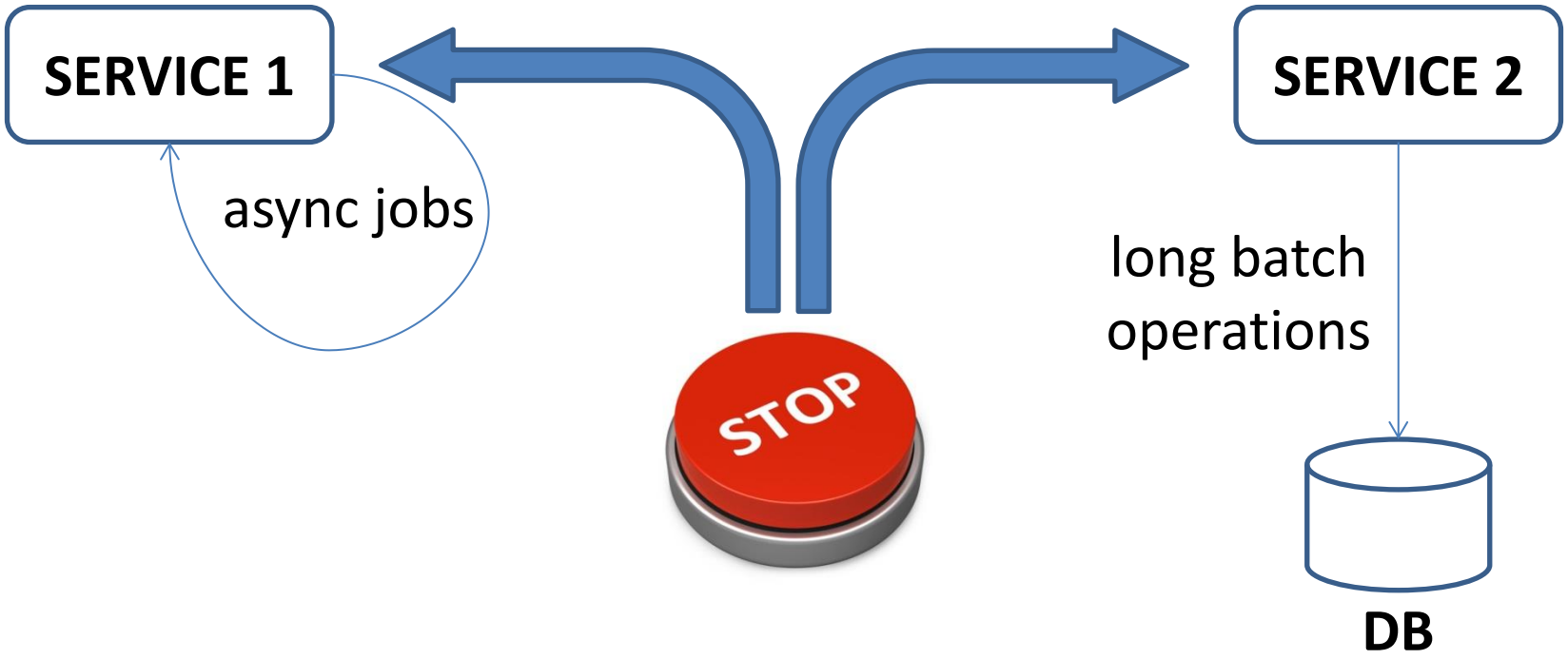
ORGANIZER AND SPEAKER:

Disclaimer

Lets start from practical tasks!


“there’s
no such thing
as a free
lunch.”

#1. Graceful shutdown

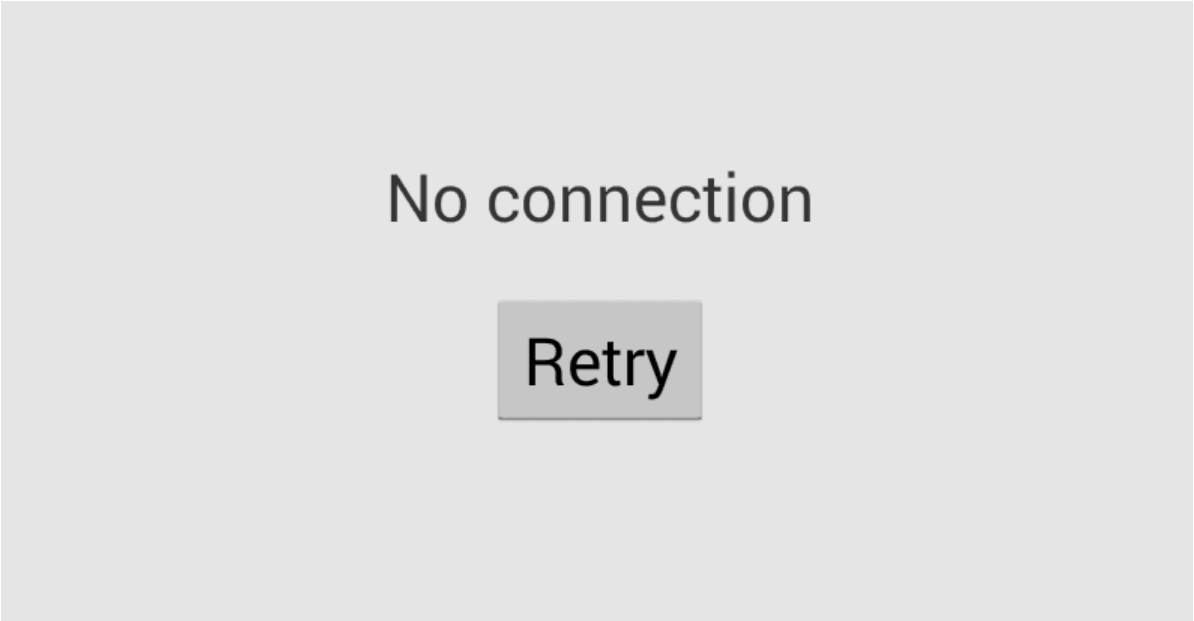


Shared boolean flag

```
private final AtomicBoolean stopped;
```

```
@Override
public void run() {
    while (!stopped.get()) {
        try {
            tryToDoTask();
        } catch (RuntimeException e) {
            onError(e);
        }
    }
}
```

#2. Retry on error task

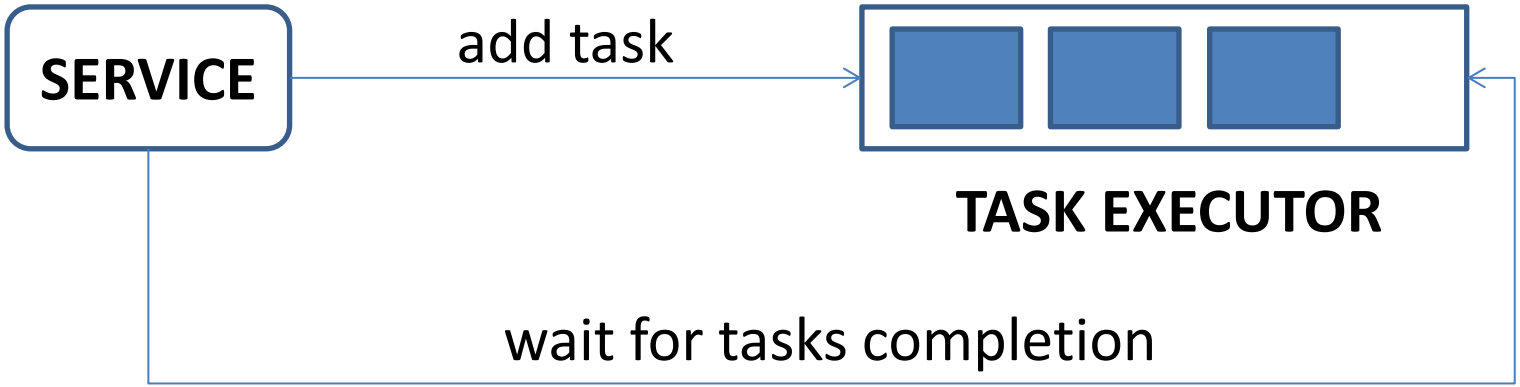


Configurable retry until done

```
private final long retryTimeoutMs;
private final AtomicBoolean stopped;
```

```
public T doTask() {
    do {
        try {
            return tryToDoTask();
        } catch (RuntimeException e) {
            LOG.error("Retry after delay: " +
                    retryTimeoutMs, e);
            sleep();
        }
    } while (!isCancelled());
    throw new AlreadyStoppedException("Task interrupted");
}
```

#3. Concurrent task executor



Submit tasks in single thread

```
private void processBatches(Date date) {
    ConcurrentTasksExecutor<Boolean> tasksExecutor =
            new ConcurrentTasksExecutor<>(executorService, 5);
    tasksExecutor.setTaskWaitTimeout(1, TimeUnit.MINUTES);
    List<SerpBatch> ready = batchProvider.getReadyBatches(date);
    for (SerpBatch batch : ready) {
        tasksExecutor.submit(new BatchProcessTask(batch, date));
    }
    tasksExecutor.waitForAllTasksCompletion();
}
```

Limit concurrent tasks

```
private final ExecutorCompletionService<T> completionService;
private final int concurrentTasksLimit;
private int tasksCount;
```

```
public ConcurrentTasksExecutor(ExecutorService executor, int concurrentTasksLimit) {
    this.completionService = new ExecutorCompletionService<T>(executor);
    this.concurrentTasksLimit = concurrentTasksLimit;
}
```

```
public void submit(Callable<T> task) {
    checkLimit();
    completionService.submit(task);
    tasksCount++;
}
```

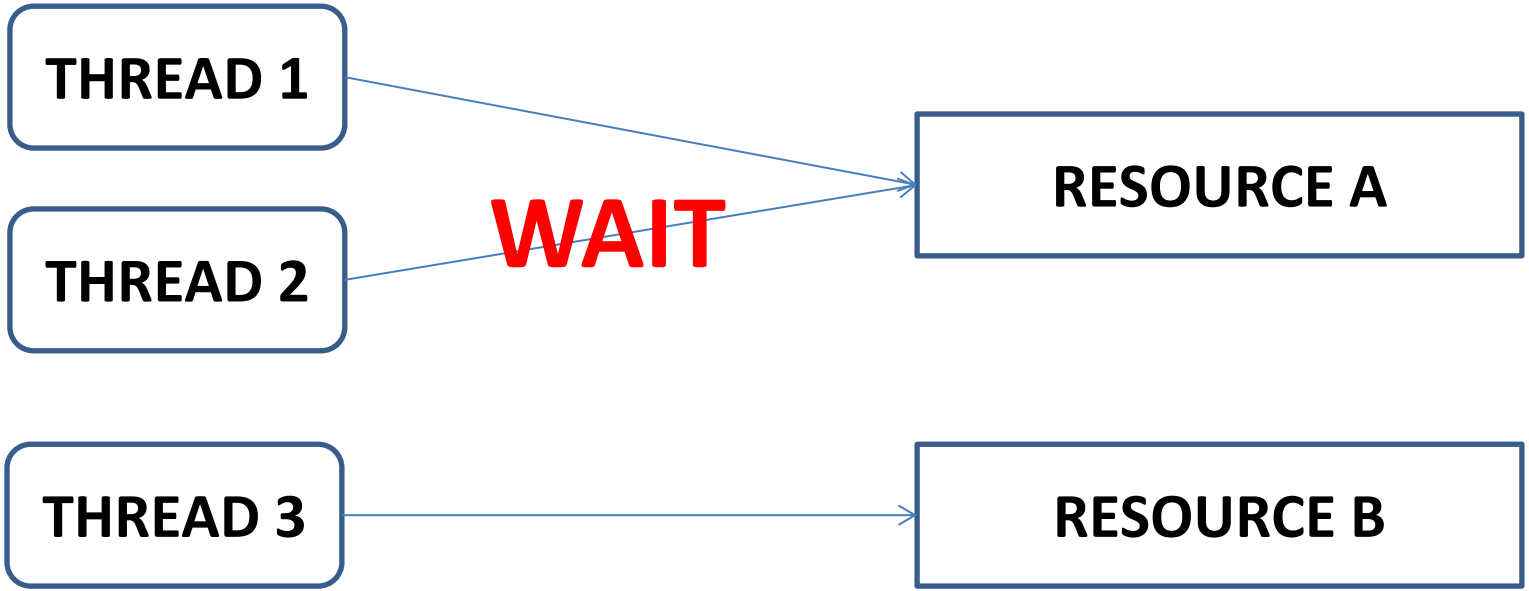
```
private void checkLimit() {
    if (tasksCount >= concurrentTasksLimit) {
        waitForTaskCompletion();
    }
}
```

Wait for single or all tasks

```
public List<T> waitForAllTasksCompletion() {
    List<T> results = new ArrayList<T>(tasksCount);
    int waitedTasksCount = tasksCount;
    for (int i = 0; i < waitedTasksCount; i++) {
        results.add(waitForTaskCompletion());
    }
    return results;
}
```

```
public T waitForTaskCompletion() {
    try {
        return tryToGetNextTaskResult();
    } catch (InterruptedException e) {
        throw new IllegalStateException("Unexpected error", e);
    }
}
```

#4. Resource locking



Simple map to store locks

```
private final ConcurrentMap<T, ResourceLock> locks =
        new ConcurrentHashMap<>();
```

```
private ResourceLock createResourceLock(T resource) {
    ResourceLock resourceLock = locks.get(resource);
    if (resourceLock == null) {
        ReadWriteLock lock = createReadWriteLock(resource);
        resourceLock = new ResourceLock(lock);
        locks.put(resource, resourceLock);
    }
    return resourceLock;
}
```

```
protected ReadWriteLock createReadWriteLock(T resource) {
    return new ReentrantReadWriteLock();
}
```

Count lock usages to remove it

```
private void decreaseAccessCount(T resource,
                                 ResourceLock resourceLock) {
    resourceLock.counter--;
    if (resourceLock.counter == 0) {
        locks.remove(resource);
    }
}
```

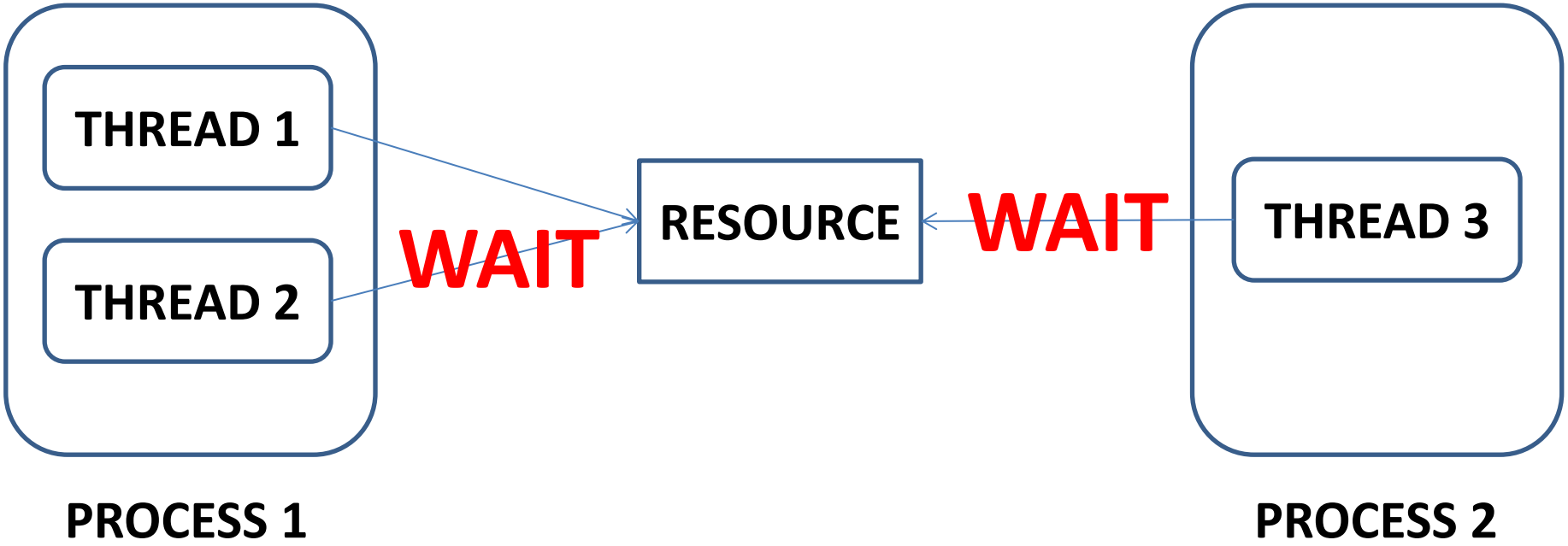
```
private static class ResourceLock {
    private int counter;
    private final ReadWriteLock rwLock;

    public ResourceLock(ReadWriteLock rwLock) {
        this.rwLock = rwLock;
    }
}
```

Reduce synchronization penalty

```
private final Striped<Lock> stripes = Striped.lazyWeakLock(64);
```

```
private ReadWriteLock allocateNamedLock(T resource) {
    Lock stripeLock = stripes.get(resource);
    stripeLock.lock();
    try {
        ResourceLock resourceLock = createResourceLock(resource);
        resourceLock.counter++;
        return resourceLock.rwLock;
    } finally {
        stripeLock.unlock();
    }
}
```

#5. Inter-process named locks



Locks are stored in simple map

```
private final ConcurrentMap<String, InterProcessLock> locks =
        new ConcurrentHashMap<>();
```

```
@Override
public boolean lock(String resource, int timeoutMs) {
    InterProcessLock lock = createLock(lockPath + "/" + resource);
    boolean acquired = lock(resource, lock, timeoutMs);
    if (acquired) {
        locks.put(resource, lock);
    }
    return acquired;
}
```

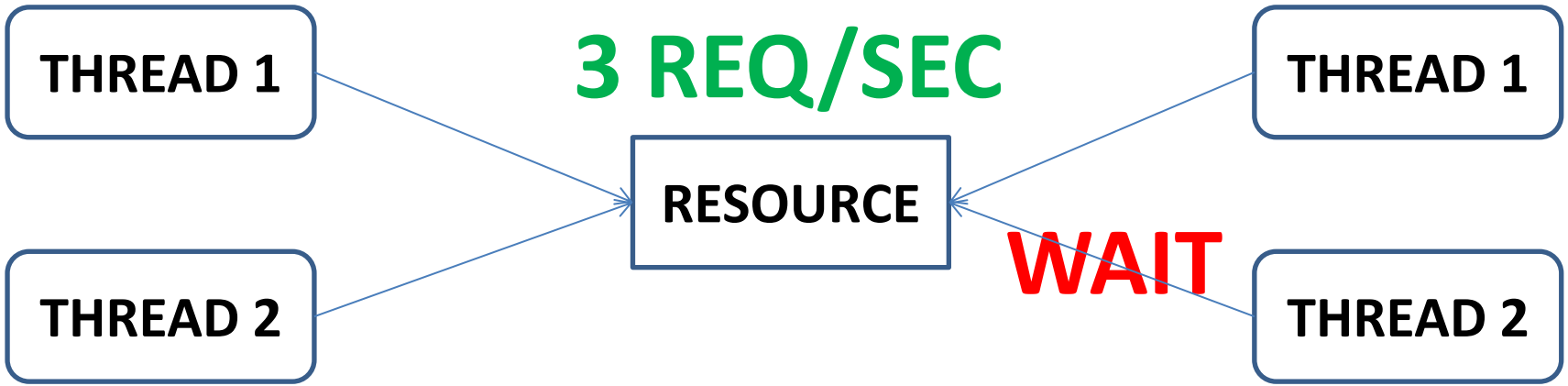
```
@Override
public void unlock(String resource) {
    InterProcessLock lock = locks.remove(resource);
    if (lock == null) {
        LOG.warn("Release of non locked resource: " + resource);
    } else {
        release(resource, lock);
    }
}
```

On error throw own exception

```
private boolean lock(String resource, InterProcessLock lock,
                     int timeoutMs) {
    try {
        return lock.acquire(timeoutMs, TimeUnit.MILLISECONDS);
    } catch (Exception e) {
        throw new ResourceLockFailureException(resource, e);
    }
}
```

```
private void release(String resource, InterProcessLock lock) {
    try {
        lock.release();
    } catch (Exception e) {
        throw new ResourceLockFailureException(resource, e);
    }
}
```

#6. Limit resource access rate



Use Semaphore as limiter

```
private final Semaphore[] ring = new Semaphore[RING_SIZE];
private final int maxRatePerSecond;
```

```
public BlockingRateLimitAccessController(int maxRatePerSecond) {
    this.maxRatePerSecond = maxRatePerSecond;
    for (int i = 0; i < ring.length; i++) {
        ring[i] = new Semaphore(maxRatePerSecond);
    }
}
```

```
public void acquireAccess() {
    while (!acquire()) {}
}
```

Work with current semaphore

```
private boolean acquire() {
    try {
        return tryToAcquire(getActiveRingCell());
    } catch (InterruptedException e) {
        return false;
    }
}
```

```
private boolean tryToAcquire(int cell) throws InterruptedException {
    Semaphore semaphore = ring[cell];
    long timeoutMs = calculateDelay();
    return semaphore.tryAcquire(timeoutMs, TimeUnit.MILLISECONDS);
}
```

```
private int getActiveRingCell() {
    return (int) ((Clock.getTimeMillis() / 1000) % RING_SIZE);
}
```

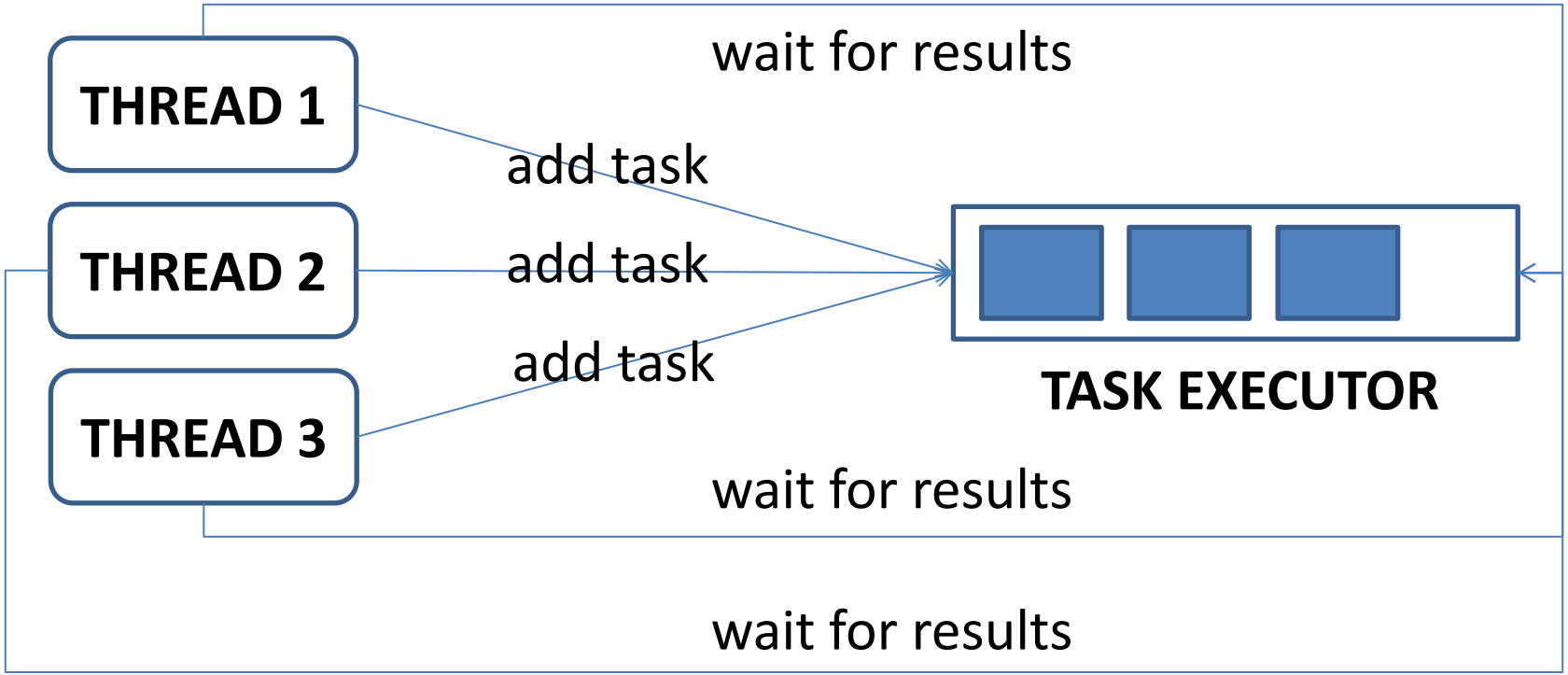
Reset not used semaphores

```
public void clean() {
    int activeCell = getActiveRingCell();
    int left = (activeCell - SAFE_OFFSET + RING_SIZE) % RING_SIZE;
    int right = (activeCell + SAFE_OFFSET + 1) % RING_SIZE;
    for (int cell = right; cell != left; cell = nextCellInRing(cell)) {
        reset(cell);
    }
}
```

```
private int nextCellInRing(int cell) {
    return (cell + 1) % RING_SIZE;
}
```

```
private void reset(int cell) {
    Semaphore semaphore = ring[cell];
    semaphore.drainPermits();
    semaphore.release(maxRatePerSecond);
}
```

#7. Transparent batching



Just add task to task executor

```
@Override
protected void index(Document document) {
    try {
        taskExecutor.addTask(document).get(taskWaitTimeout,
                TimeUnit.MILLISECONDS);
    } catch (ExecutionException e) {
        throw new IllegalStateException("Can't index page",
                e.getCause());
    } catch (Exception e) {
        throw new IllegalStateException("Can't index page", e);
    }
}
```

Tasks processed in batches

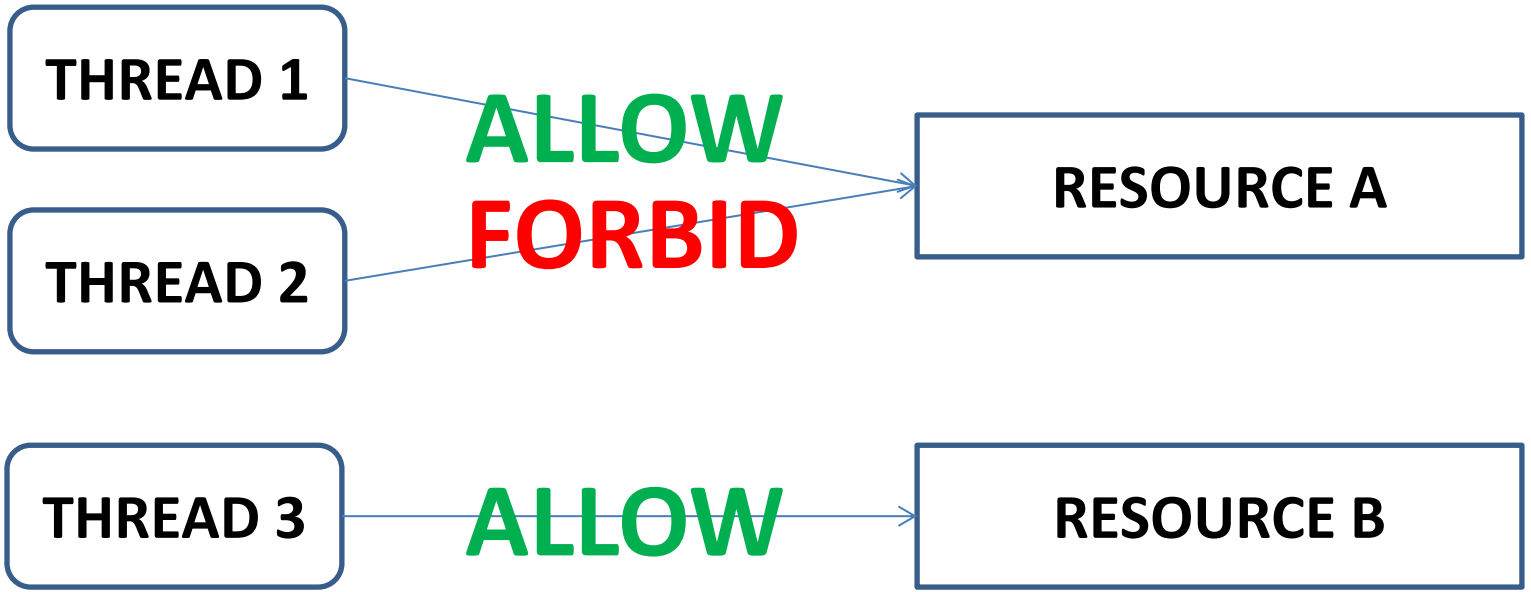
```
private class BatchTaskMonitor implements Runnable {
    @Override
    public void run() {
        while (!closed && !Thread.interrupted()) {
            tryToProcessNextBatch();
        }
    }

    private void tryToProcessNextBatch() {
        try {
            waitForNextBatch();
            processBatch();
            releaseTasks();
        } catch (Exception e) {
            failTasks(e);
        } finally {
            clearBatch();
        }
    }
}
```

Configurable batching strategy

```
public Future<T> addTask(T data) {
    BatchTask<T> task = new BatchTask<T>(data);
    tasks.add(task);
    return task;
}
```

```
private void waitForNextBatch() throws InterruptedException {
    BatchTask<T> newTask = tasks.poll(TASK_WAIT_TIMEOUT,
            TimeUnit.SECONDS);
    Validate.notNull(newTask, "Task waiting timeout exceed");
    batch.add(newTask);
    int delay = batchDelayStrategy.getBatchWaitDelay(
            newTask.getCreatedAt(), tasks.size() + 1);
    if (delay > 0) {
        Thread.sleep(delay);
    }
}
```

#8. Resource access control



Access success is based on time

```
private final ConcurrentMap<String, AccessTimestamp> accessLog;
private final String resourceId;
private final AccessTimestamp currentAccessTime;
private final AccessTimestamp lastAccessTime;
private AccessTimestamp successAccessTime;
```

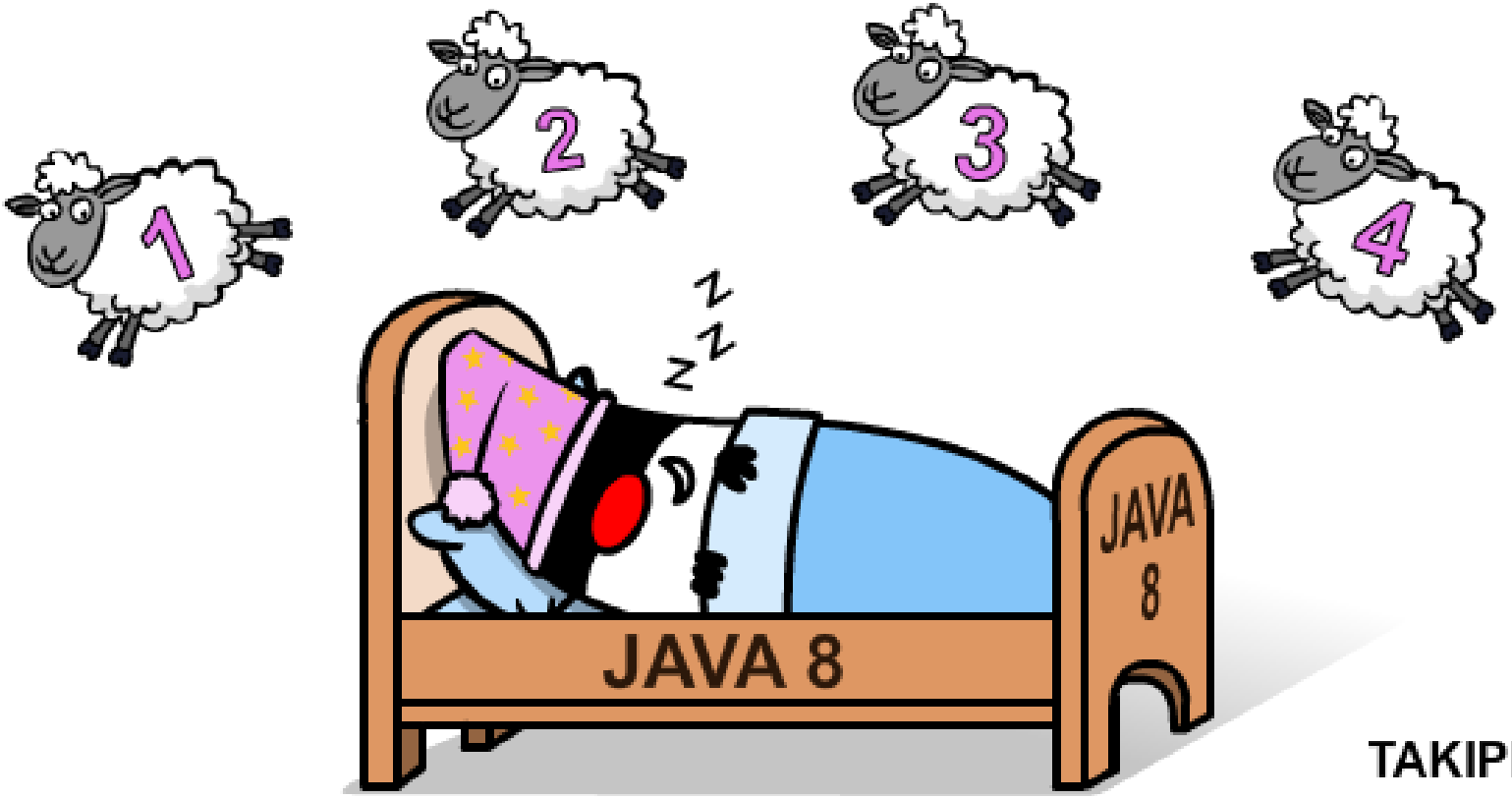
```
public AccessRequest(String resourceId,
        ConcurrentMap<String, AccessTimestamp> accessLog) {
    this.resourceId = resourceId;
    this.accessLog = accessLog;
    this.currentAccessTime = new AccessTimestamp(Clock.getTimeMillis());
    this.lastAccessTime = accessLog.putIfAbsent(resourceId,
            currentAccessTime);
}
```

Check for successful access

```
public boolean isTimeoutExceed(int accessDelay) {
    return lastAccessTime.isTimeoutExceed(currentAccessTime,
            accessDelay);
}
```

```
public boolean logSuccessAccess() {
    AccessTimestamp oldAccessTime = lastAccessTime == null ?
            currentAccessTime : lastAccessTime;
    boolean stillActual = accessLog.replace(resourceId,
            oldAccessTime, currentAccessTime);
    if (stillActual) {
        successAccessTime = currentAccessTime;
        return true;
    }
    return false;
}
```

#9. Sharded quick counters



LongAdder algorithm

```
public void add(long x) {
    Cell[] as; long b, v; HashCode hc; Cell a; int n;
    if ((as = cells) != null || !casBase(b = base, b + x)) {
        boolean uncontended = true;
        int h = (hc = threadHashCode.get()).code;
        if (as == null || (n = as.length) < 1 ||
            (a = as[(n - 1) & h]) == null ||
            !(uncontended = a.cas(v = a.value, v + x)))
            retryUpdate(x, hc, uncontended);
    }
}
```

Lets review useful tools!

#1. Thread factory builder

```
BasicThreadFactory factory =
        new BasicThreadFactory.Builder()
      .namingPattern("Parser %d")
      .daemon(true)
      .priority(Thread.MAX_PRIORITY)
      .build();
return Executors.newFixedThreadPool(10, factory);
```

#2. ConcurrentUtils

```
public static <K, V> V putIfAbsent(ConcurrentMap<K, V> map,
                                   K key, V value) {
    V result = map.putIfAbsent(key, value);
    return result != null ? result : value;
}
```

```
public static void throwCause(ExecutionException ex) {
    if (ex.getCause() instanceof RuntimeException) {
        throw (RuntimeException) ex.getCause();
    }
    if (ex.getCause() instanceof Error) {
        throw (Error) ex.getCause();
    }
}
```

#3. TimedSemaphore

```
private final TimedSemaphore semaphore =
        new TimedSemaphore(1, TimeUnit.SECONDS, 10);
```

```
@Override
public void run() {
    while (!stopped) {
        try {
            semaphore.acquire();
        } catch (InterruptedException e) {
            throw new IllegalStateException("ERROR", e);
        }
        updateStats();
    }
}
```

#4. TimeLimiter

```
KeywordChecker target = new KeywordChecker();
TimeLimiter limiter = new SimpleTimeLimiter(executorService);
KeywordChecker proxy = limiter.newProxy(target,
        KeywordChecker.class, 50, TimeUnit.MILLISECONDS);
try {
    proxy.checkKeyword("JEEConf");
} catch (UncheckedTimeoutException e) {
    throw new IllegalArgumentException("Complex keyword", e);
}
```

#5. RateLimiter

```
private final RateLimiter rateLimiter =
        RateLimiter.create(5, 10, TimeUnit.SECONDS);
```

```
public void submitTasks(List<Runnable> tasks,
                        Executor executor) {
    for (Runnable task : tasks) {
        rateLimiter.acquire();
        executor.execute(task);
    }
}
```

#6. Striped

```
private final Striped<Lock> stripes = Striped.lazyWeakLock(64);
```

```
private ReadWriteLock allocateNamedLock(T resource) {
    Lock stripeLock = stripes.get(resource);
    stripeLock.lock();
    try {
        ResourceLock resourceLock = createResourceLock(resource);
        resourceLock.counter++;
        return resourceLock.rwLock;
    } finally {
        stripeLock.unlock();
    }
}
```

#7. ListenableFuture

```
ListenableFuture<QueryResult> future = getUser("bob");
Futures.addCallback(future, new FutureCallback<QueryResult>() {
    @Override
    public void onSuccess(QueryResult result) {
        storeInCache(result);
    }

    @Override
    public void onFailure(Throwable t) {
        reportDataAccessError(t);
    }
});
```

#8. Uninterruptibles

```
public static void awaitUninterruptibly(CountDownLatch latch)
public static boolean awaitUninterruptibly(CountDownLatch latch,
      long timeout, TimeUnit unit)
public static void joinUninterruptibly(Thread toJoin)
public static <V> V getUninterruptibly(Future<V> future)
      throws ExecutionException
public static <V> V getUninterruptibly(
      Future<V> future, long timeout,  TimeUnit unit)
          throws ExecutionException, TimeoutException
public static void joinUninterruptibly(Thread toJoin, long timeout,
                                       TimeUnit unit)
public static <E> E takeUninterruptibly(BlockingQueue<E> queue)
public static <E> void putUninterruptibly(BlockingQueue<E> queue,
                                          E element)
public static void sleepUninterruptibly(long sleepFor, TimeUnit unit)
```

#9. CLHQueueLock

```
public class CLHQueueLock {
    private final AtomicReference<Qnode> tail = new AtomicReference<>();

    private final ThreadLocal<Qnode> myNode = new ThreadLocal<Qnode>() {
        @Override
        protected Qnode initialValue() {
            return new Qnode();
        }
    };

    private final ThreadLocal<Qnode> myPred = new ThreadLocal<Qnode>();
```

```
    public CLHQueueLock() {
        Qnode qnode = new Qnode();
        qnode.locked = false;
        tail.set(qnode);
    }
```

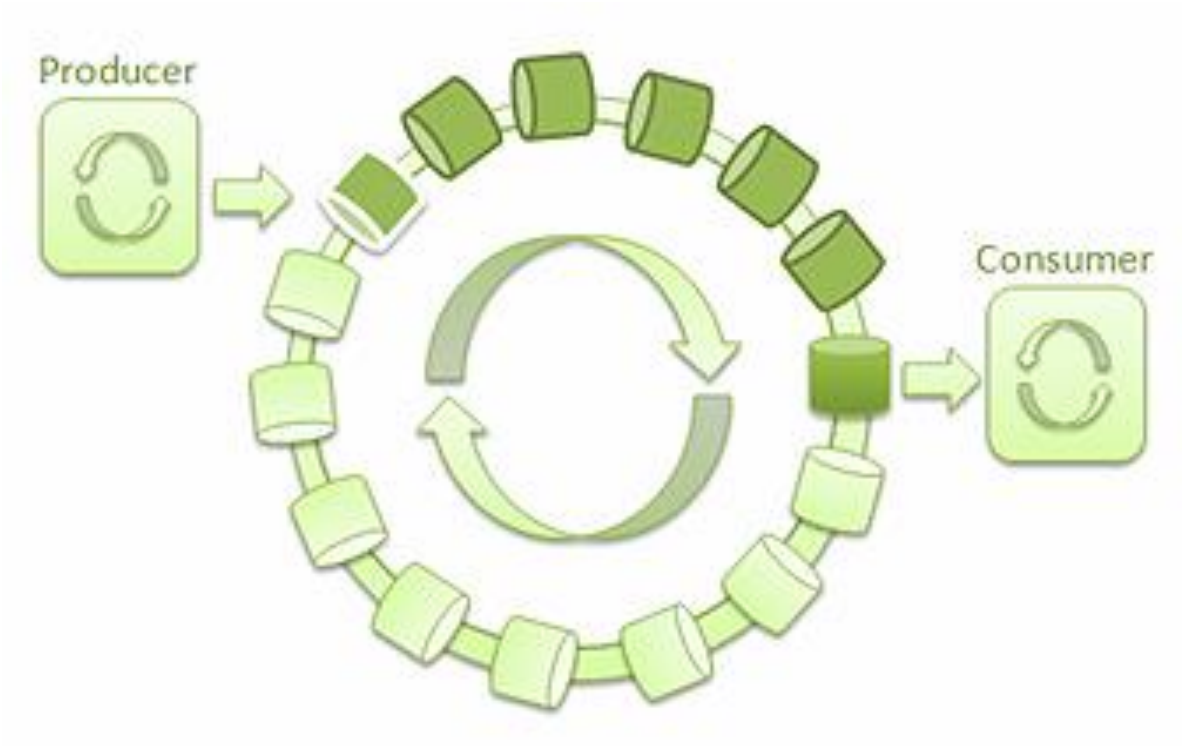
```
    private static final class Qnode {
        private volatile boolean locked = true;
    }
```

Spin lock implementation

```
public void lock() {
    final Qnode localNode = myNode.get();
    localNode.locked = true;
    final Qnode pred = tail.getAndSet(localNode);
    myPred.set(pred);
    while (pred.locked) ;
}
```

```
public void unlock() {
    myNode.get().locked = false;
    myNode.set(myPred.get());
}
```

#10. Disruptor



#11. ZooKeeper + Curator

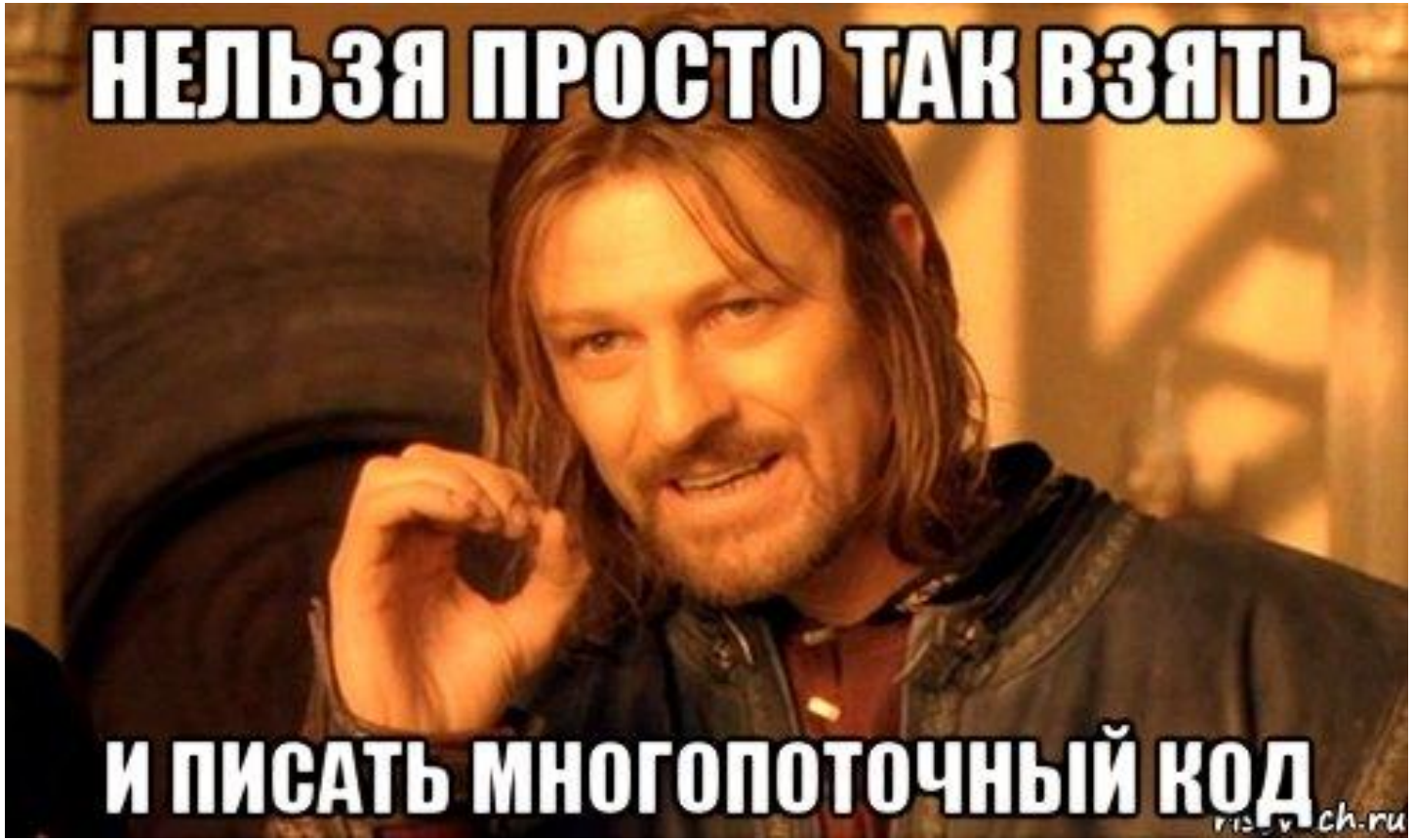
> "Guava is to Java what Curator is to ZooKeeper"
> Patrick Hunt, ZooKeeper committer

- *Elections*
- *Locks*
- *Barriers*
- *Counters*
- *Caches*
- *Nodes*
- *Queues*

Useful links

- https://code.google.com/p/guava-libraries/
- http://commons.apache.org/proper/commons-lang/
- http://dou.ua/lenta/articles/clh-lock/
- http://habrahabr.ru/company/luxoft/blog/157273/
- http://habrahabr.ru/post/130113/
- http://lmax-exchange.github.io/disruptor/
- http://zookeeper.apache.org/
- http://curator.apache.org/
- http://en.wikipedia.org/wiki/Ctrie
- https://code.google.com/p/guava-libraries/wiki/ListenableFutureExplained
- https://guava-libraries.googlecode.com/files/guava-concurrent-slides.pdf

And remember…



@xpinjection

http://xpinjection.com

mikalai.alimenkou@xpinjection.com