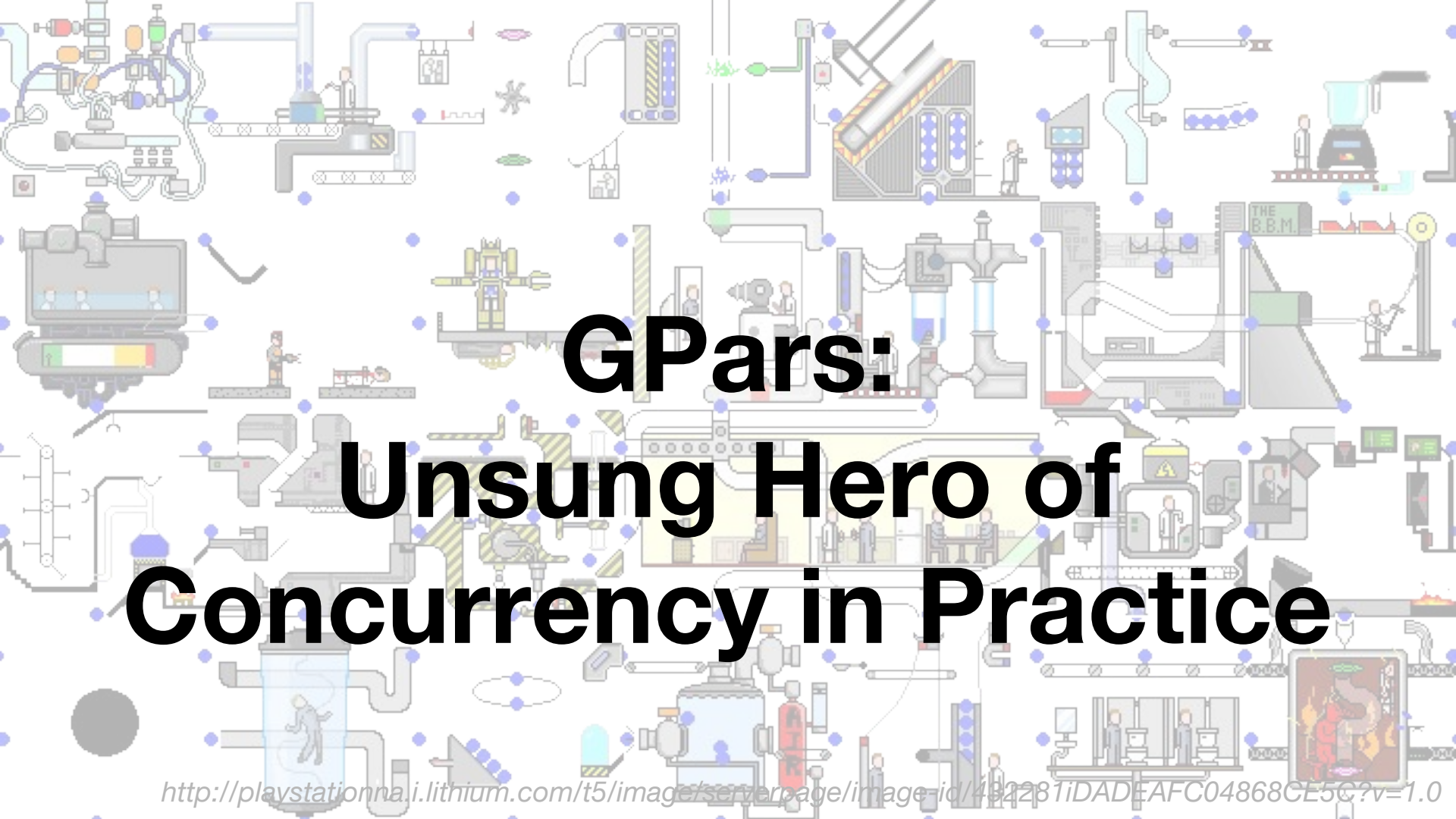
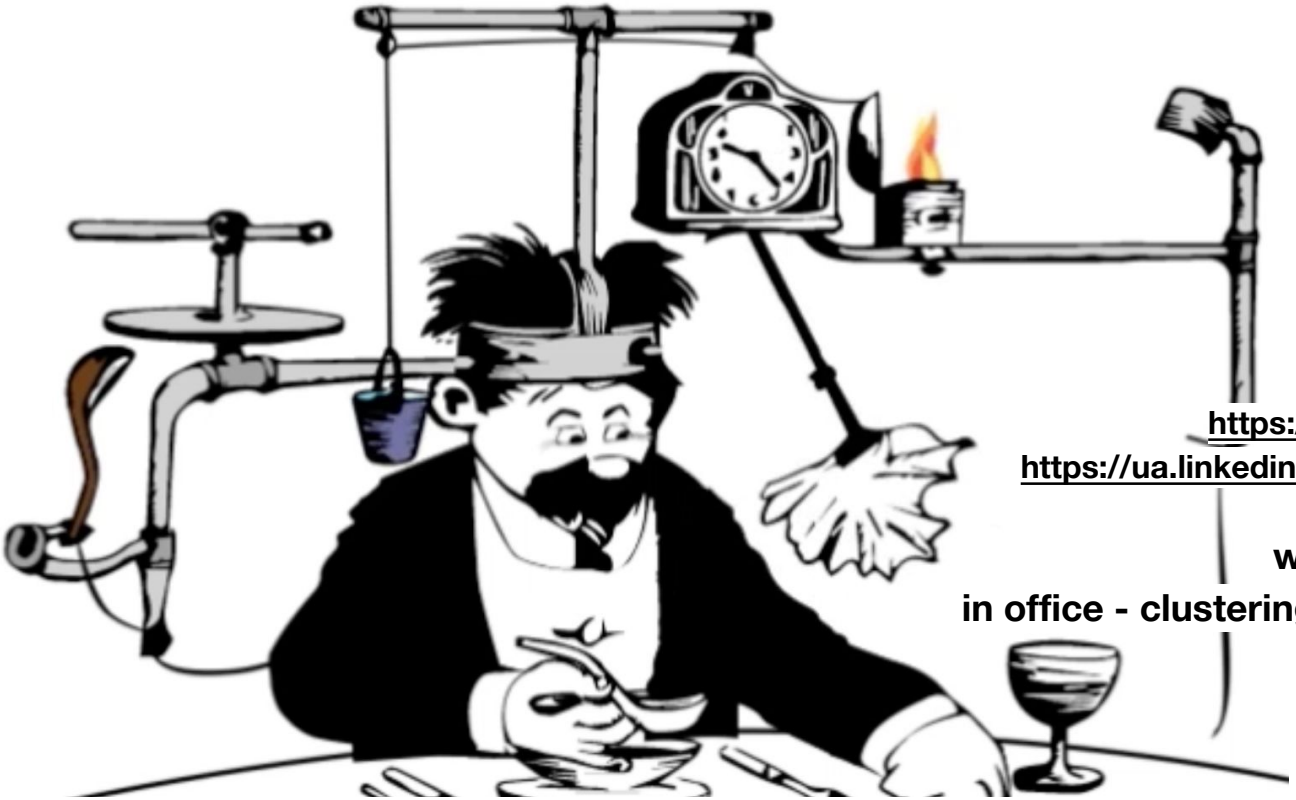


GPars: Unsung Hero of Concurrency in Practice

http://playstationna.i.lithium.com/t5/image/serverpage/image-id/432281iDADEAFC04868CE5C?v=1.0



Yaroslav Yermilov

Senior Software Engineer
EPAM Systems

https://yermilov.github.io/
https://twitter.com/yermilov17
https://www.facebook.com/yaroslav.yermilov
https://ua.linkedin.com/pub/yaroslav-yermilov/58/682/506

work for EPAM Systems since 2011
in office - clustering, Big Data and automated testing
out of office - Groovy

https://motherboard.vice.com/en_us/article/inside-rube-goldberg-machine-youtube-video-artist-joseph-herschers-bedroom-workshop

Focus for this talk

two guidelines for developing concurrent code

how Java handles concurrent/parallel development?

GPars place in this picture

how GPars handles concurrent/parallel development?

why and when use GPars?

why and when do not use GPars?

http://www.theverge.com/2015/4/22/8381963/rube-goldberg-machine-contest-history-ideas

guideline #2 for developing concurrent code

MURPHY'S LAW:

Anything that can go wrong will go wrong.

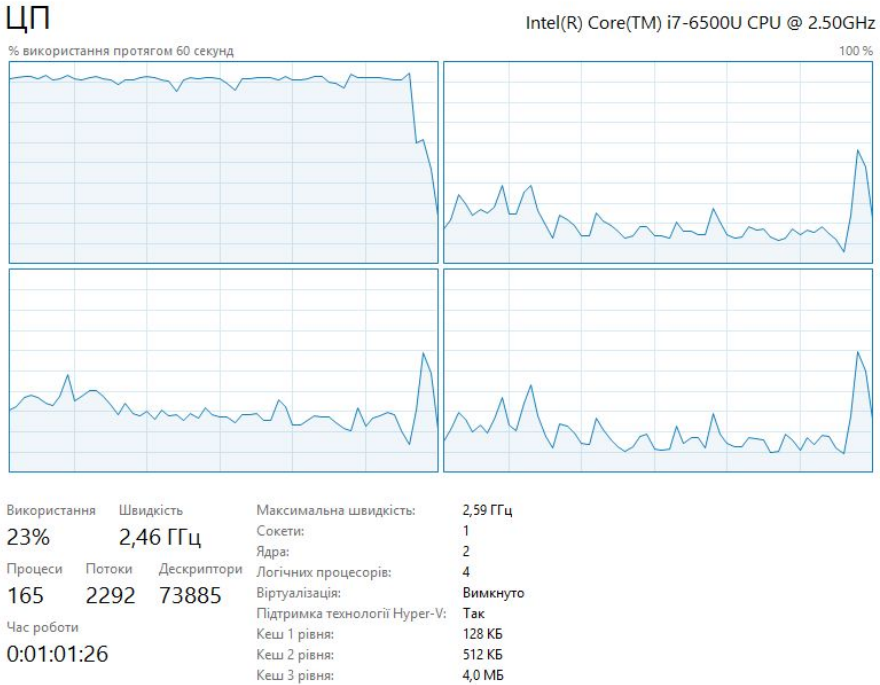
```
public class Holder {
    private int n;

    public Holder(int n) { this.n = n; }

    public void assertSanity() {
        if (n != n) {
            throw new AssertionError("Even it can go wrong!");
        }
    }
}

public class Initializer {

    public Holder holder;

    public void init() { holder = new Holder(42); }
}
```

http://www.systemcomic.com/2009/04/27/the-system-197

guideline #1 for developing concurrent code

MOORE'S LAW:

Computers will get exponentially faster.

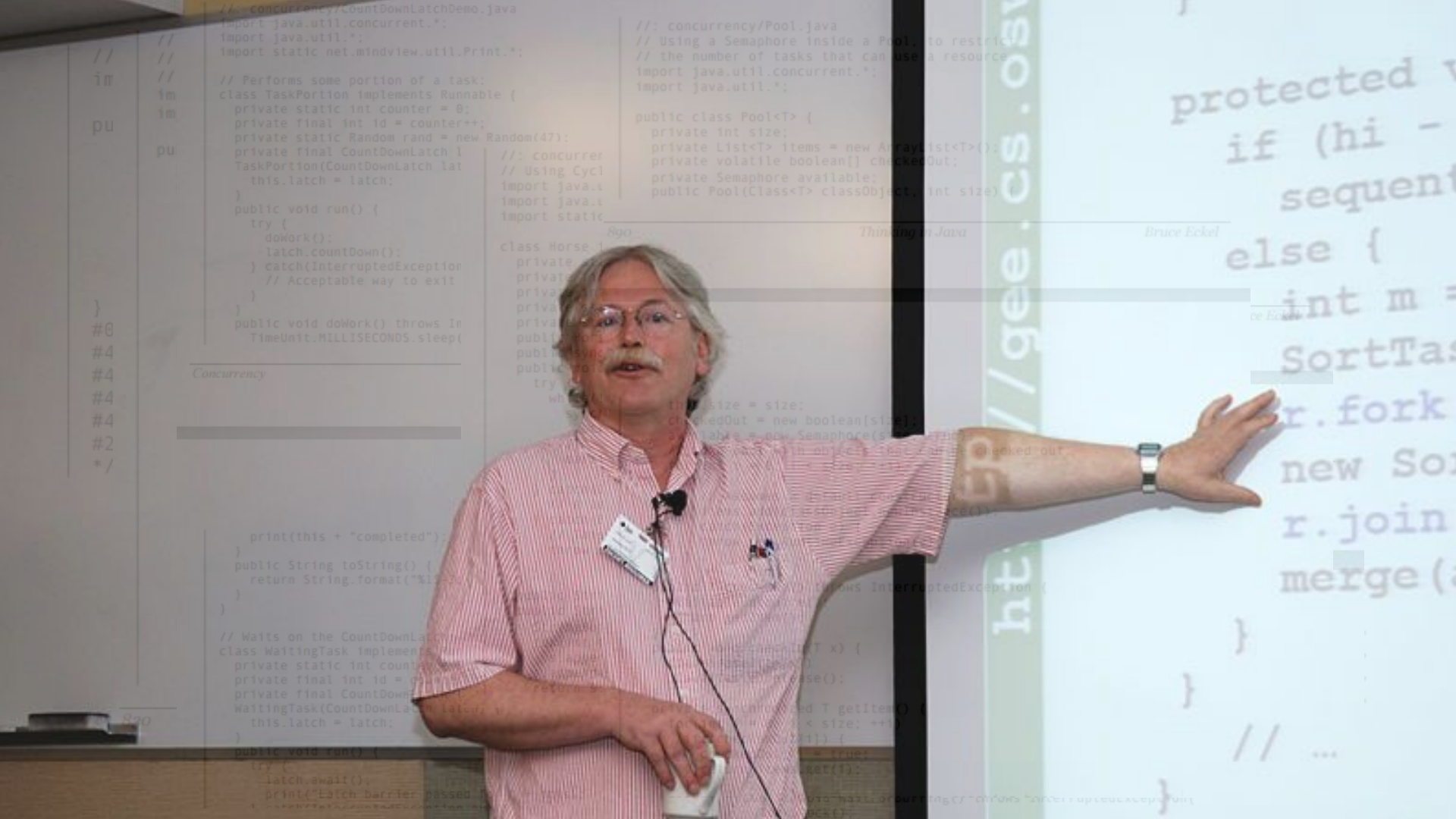


http://www.systemcomic.com/2009/04/27/the-system-197

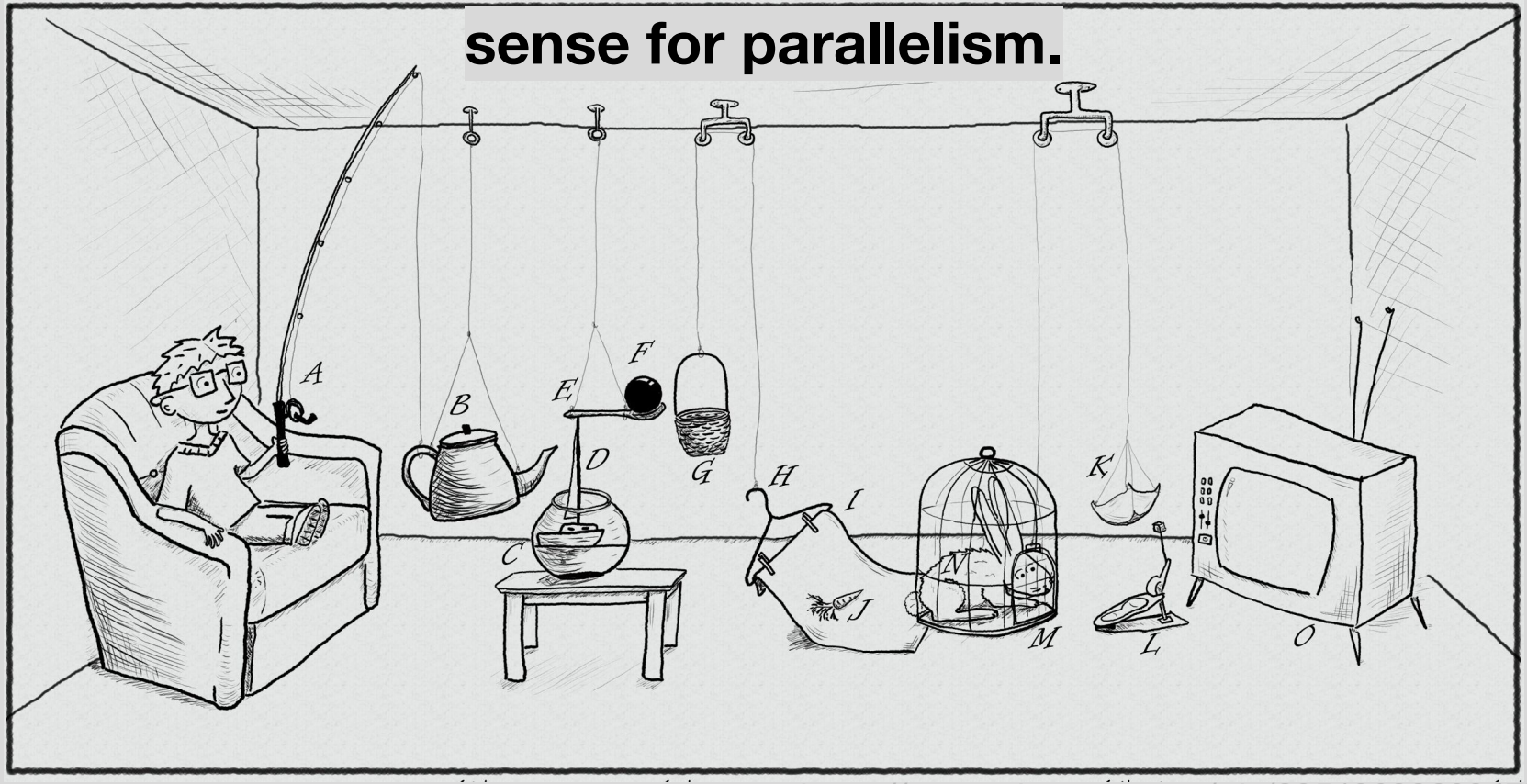
The **Thread** class

The traditional way to turn a **Runnable** object into a working task is to hand it to a **Thread** constructor. This example shows how to drive a **Liftoff** object using a **Thread:**

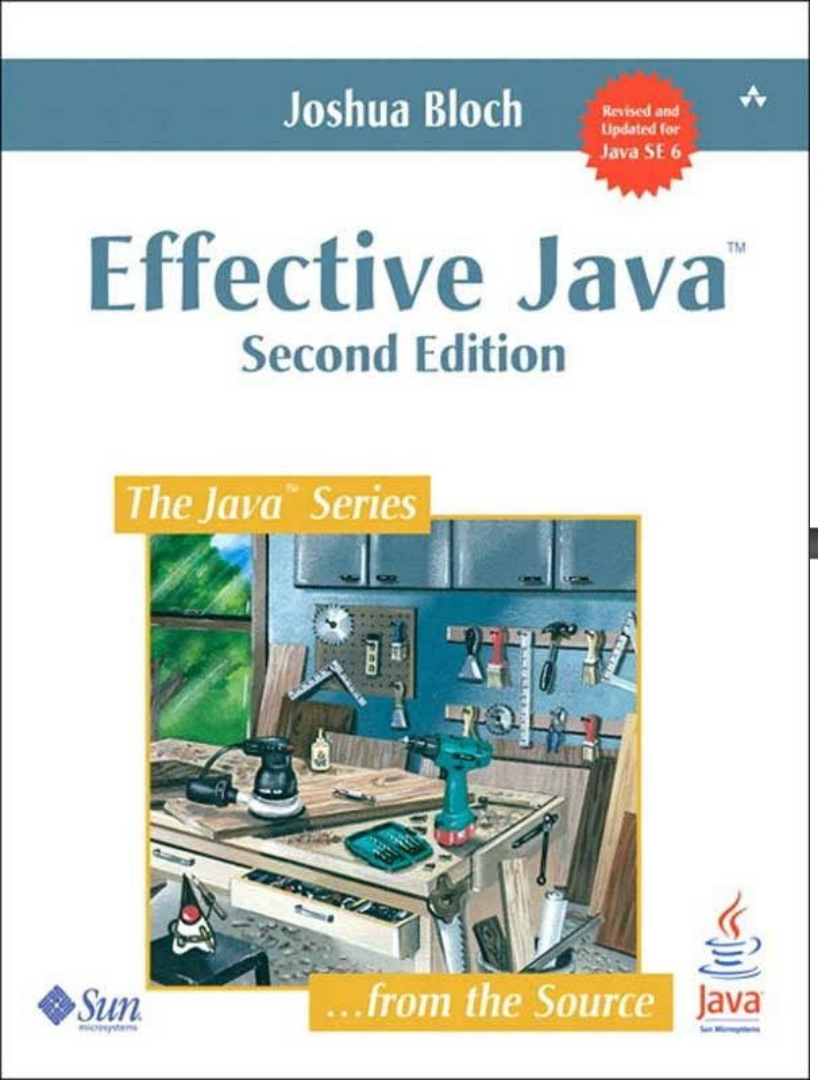
```
//: concurrency/BasicThreads.java
// The most basic use of the Thread class.

public class BasicThreads {
  public static void main(String[] args) {
    Thread t = new Thread(new LiftOff());
    t.start();
    System.out.println("Waiting for LiftOff");
  }
} /* Output: (90% match)
Waiting for LiftOff
#0(9), #0(8), #0(7), #0(6), #0(5), #0(4), #0(3), #0(2), #0(1),
#0(Liftoff!),
*///:~
```


Concurrency
Thinking in Java
Bruce Eckel
http://gee.cs.oswego.edu

The traditional thread-based concurrency model built into Java doesn't match well with the natural human sense for parallelism.



https://www.pinterest.com/pin/395190936024778319/



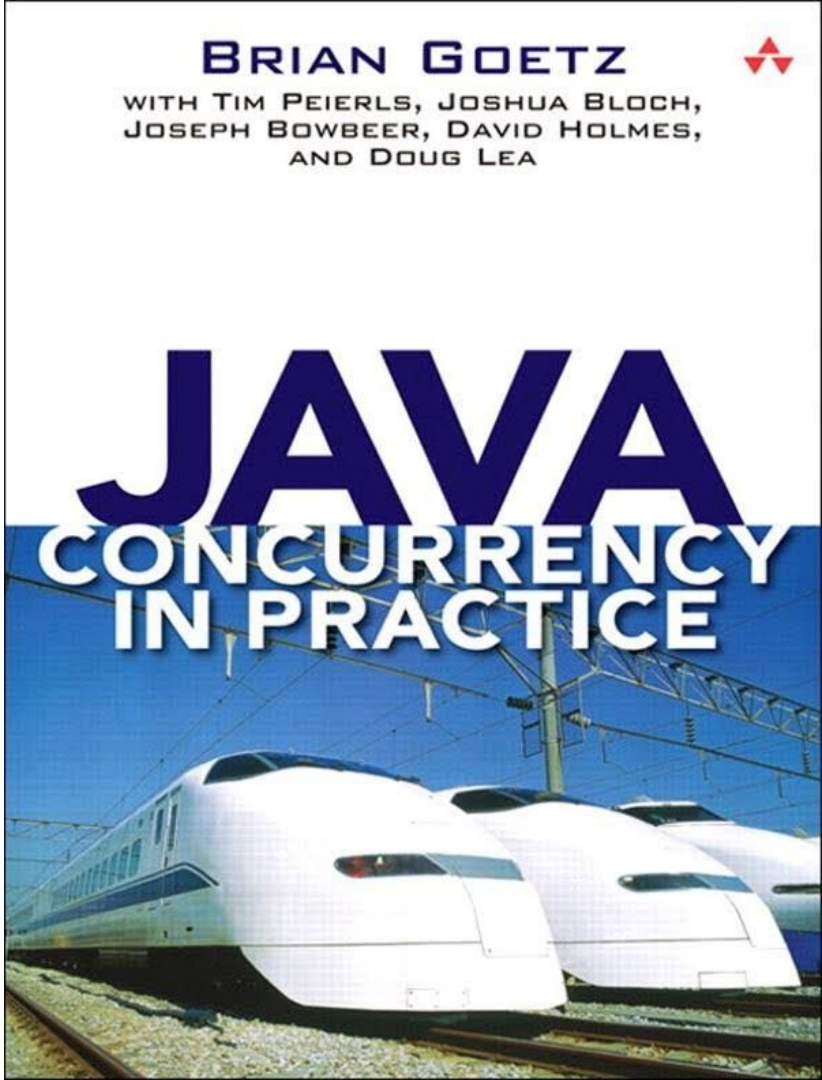
If multiple threads access the same mutable state variable without appropriate synchronization, your program is broken.

There are three ways to fix it:

Don’t share the state variable across threads

Make the state variable immutable

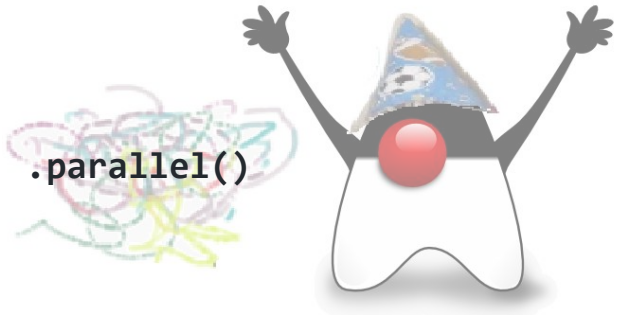
Use synchronization whenever accessing the state variable



```
public class MergeSortTask<T extends Comparable<T>> extends RecursiveTask<List<T>> {
   @Override
   protected List<T> compute() {
       if (list.size() < 2) { return list; }
       if (list.size() == 2) {
           if (list.get(0).compareTo(list.get(1)) != 1) {
               return list;
           } else {
               return asList(list.get(1), list.get(0));
           }
       }

       MergeSortTask<T> leftTask = new MergeSortTask<>(list.subList(0, list.size() / 2));
       MergeSortTask<T> rightTask = new MergeSortTask<>(list.subList(list.size() / 2,
list.size()));

       leftTask.fork(); rightTask.fork();

       List<T> left = leftTask.join();
       List<T> right = rightTask.join();

       return merge(left, right);
   }
```

http://blog.takipi.com/forkjoin-framework-vs-parallel-streams-vs-executorservice-the-ultimate-benchmark/

```
Optional<Status> mostPopularTweet = tweets.stream()
```




```
        .filter(tweet -> tweet.getText().toLowerCase().contains(topic.toLowerCase()))
        .filter(tweet -> !tweet.isRetweet())
        .max(comparingInt(tweet -> tweet.getFavoriteCount() + tweet.getRetweetCount()));
```

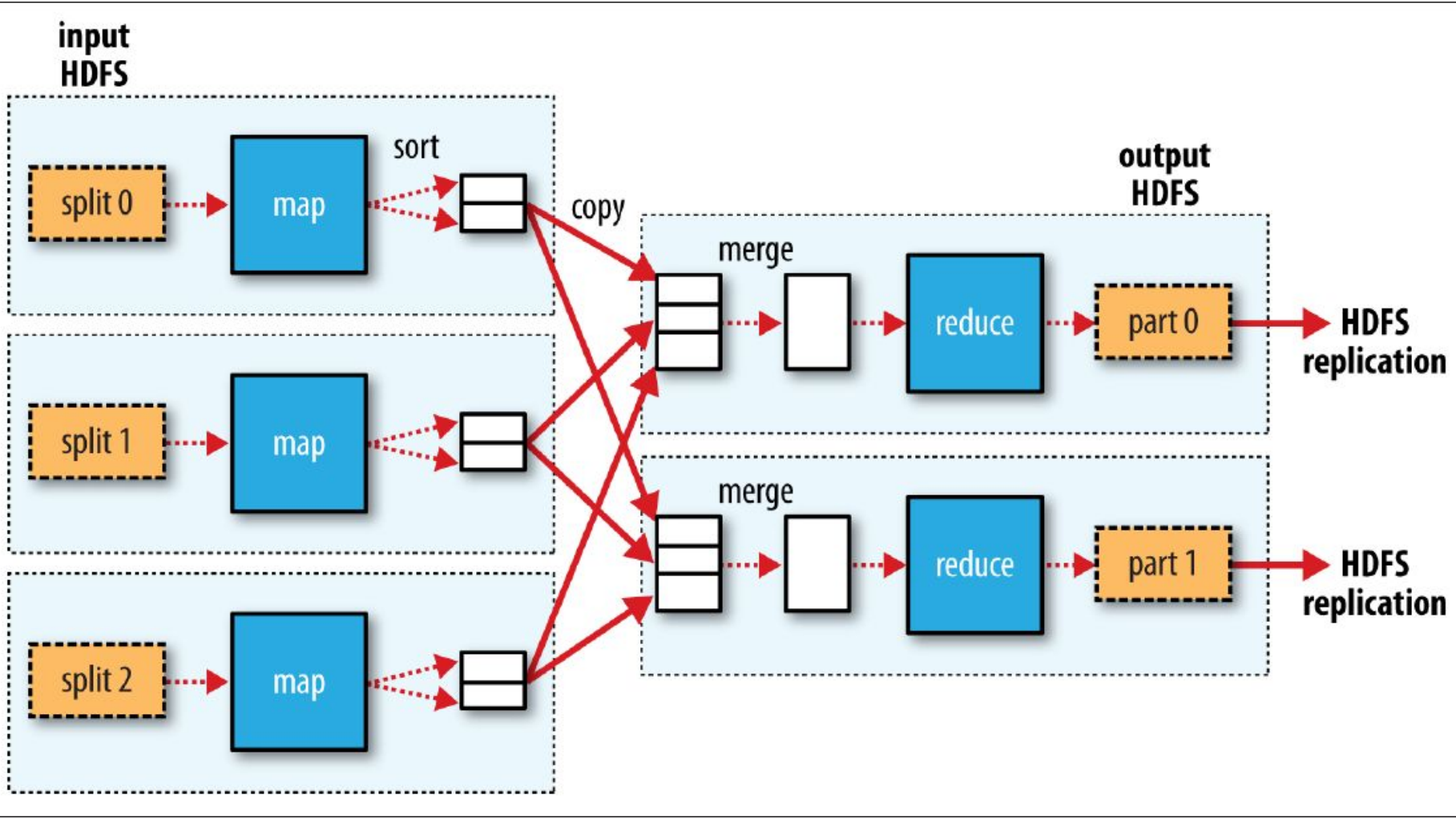
Next you create the resource controller that will serve these greetings.

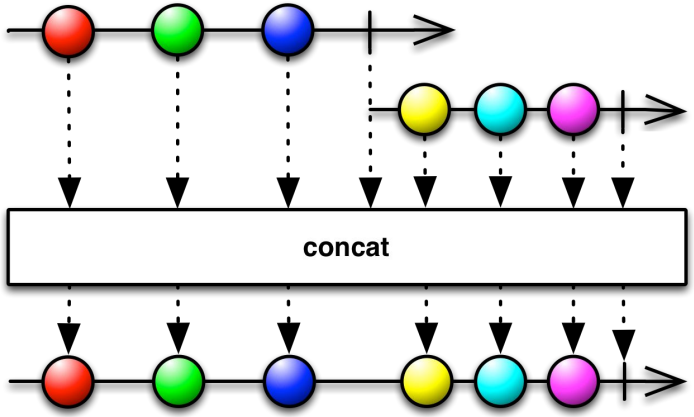
Create a resource controller

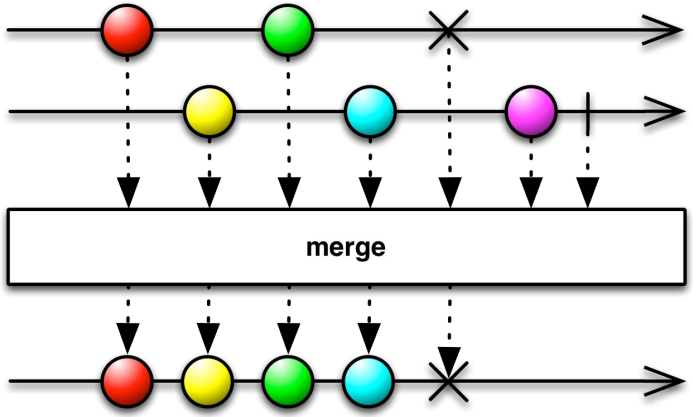
In Spring's approach to building RESTful web services, HTTP requests are handled by a controller. These components are easily identified by the `@RestController` annotation, and the `GreetingController` below handles `GET` requests for `/greeting` by returning a new instance of the `Greeting` class:

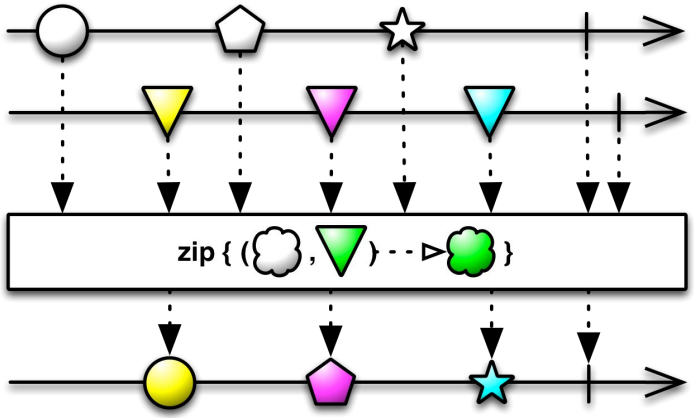
`src/main/java/hello/GreetingController.java`

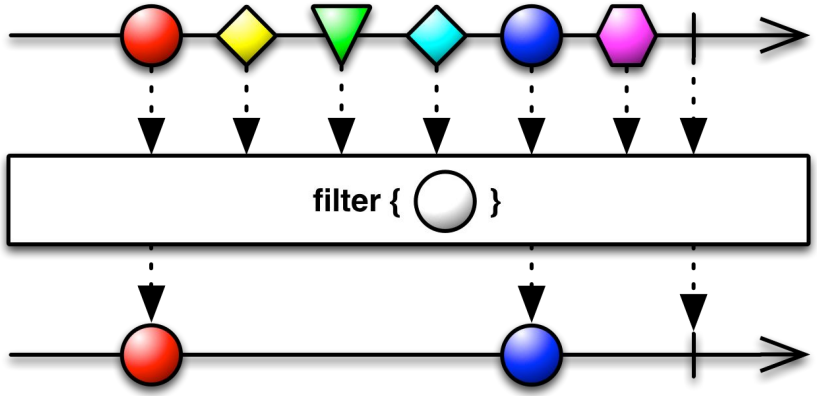
```
@RestController
public class GreetingController {

  private static final String template = "Hello, %s!";
  private final AtomicLong counter = new AtomicLong();

  @RequestMapping("/greeting")
  public Greeting greeting(@RequestParam(value="name", defaultValue="World") String name) {
      return new Greeting(counter.incrementAndGet(), String.format(template, name));
  }
}
```


input
HDFS
split 0
map
sort
split 1
map
split 2
map
copy
merge
reduce
part 0
output
HDFS
HDFS
replication
merge
reduce
part 1
HDFS
replication



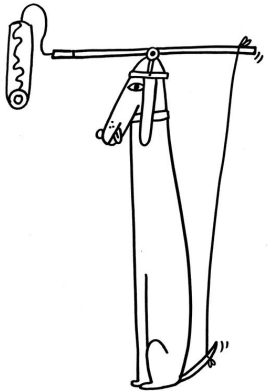






http://reactivex.io/RxJava/2.x/javadoc/io/reactivex/Flowable.html


akka



GPars:

data parallelism

map/reduce

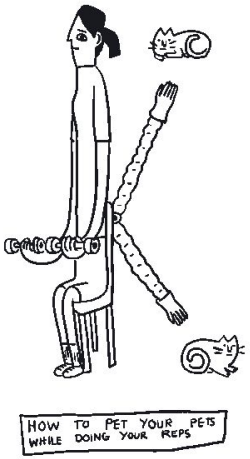
fork/join

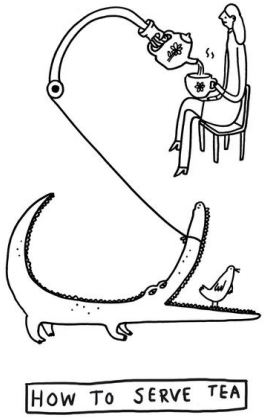
asynchronous closures

agents

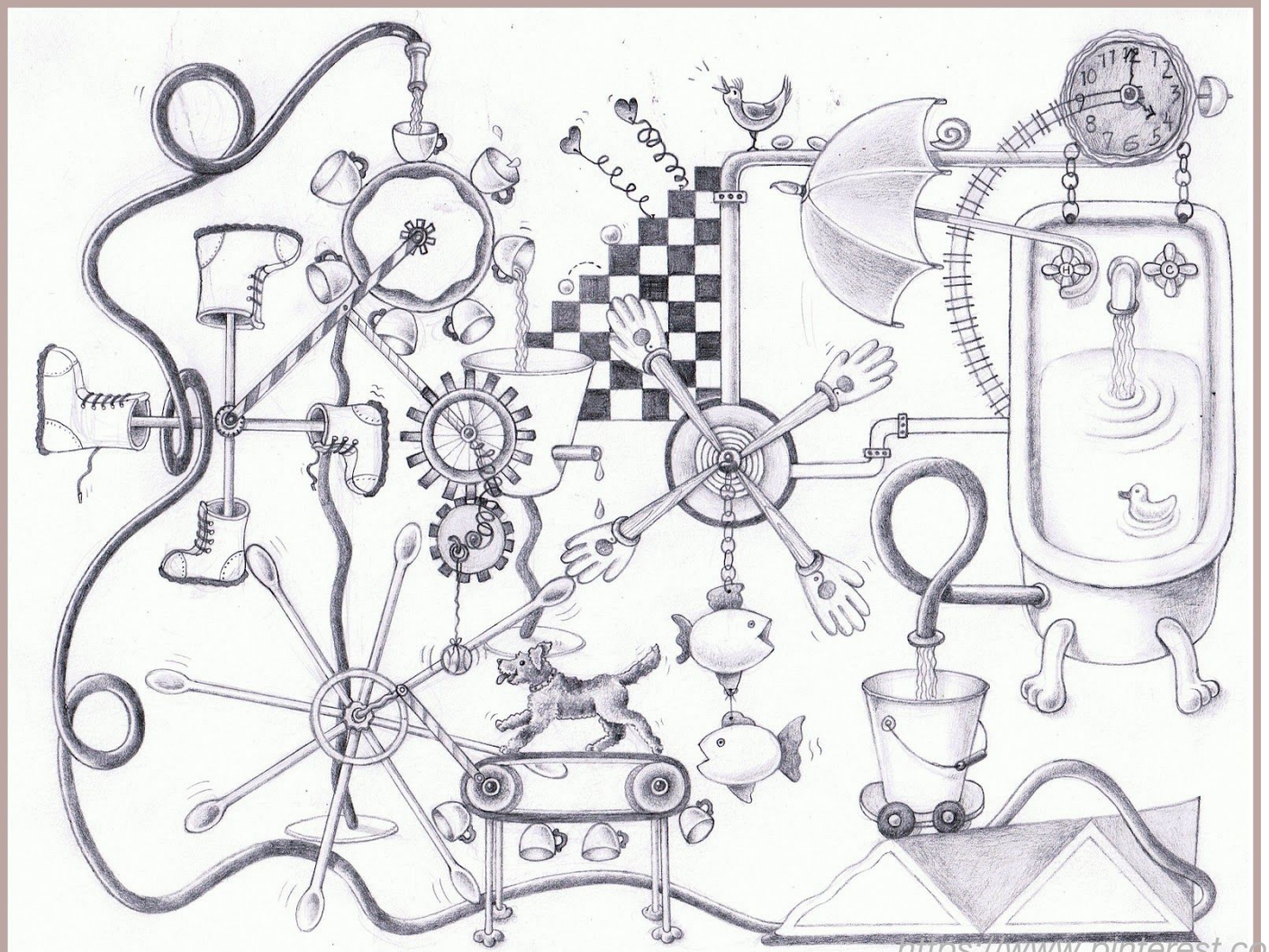
actors

dataflows



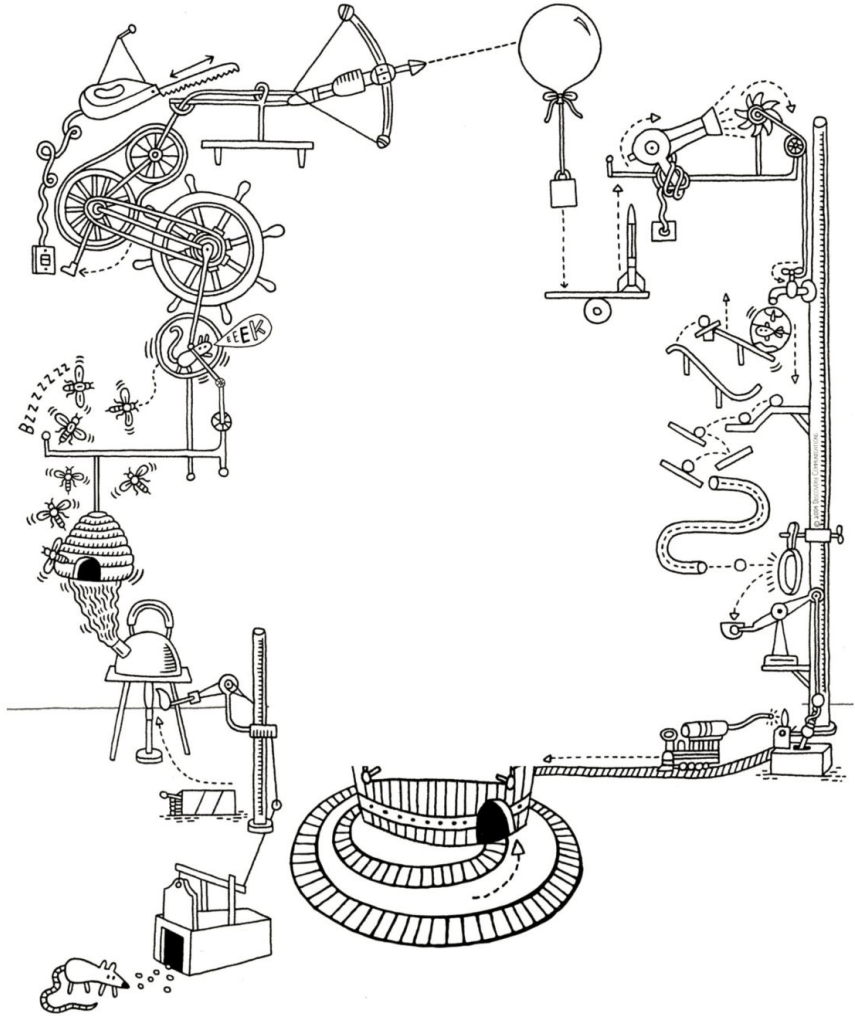


http://www.theverge.com/2015/4/22/8381963/rube-goldberg-machine-contest-history-ideas



DEMO

JAVA
OR
GROOVY?



http://forevercuriousmuseum.org/events/rube-goldberg-machine-summer-camp-july-28-29-30-2015/

```java
public final class ImmutableJavaPerson {

    private final String name;

    private final Collection<String> tweets;

    public ImmutableJavaPerson(String name, Collection<String> tweets) {
        this.name = name;
        this.tweets = new ArrayList<>(tweets);
    }

    public String getName() {
        return name;
    }

    public Collection<String> getTweets() {
        return unmodifiableCollection(tweets);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
```

```
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        ImmutableJavaPerson that = (ImmutableJavaPerson) o;

        if (name != null ? !name.equals(that.name) : that.name != null) return false;
        return tweets != null ? tweets.equals(that.tweets) : that.tweets == null;
    }

    @Override
    public int hashCode() {
        int result = name != null ? name.hashCode() : 0;
        result = 31 * result + (tweets != null ? tweets.hashCode() : 0);
        return result;
    }
}
```

```groovy
@Immutable class ImmutableGroovyPerson {

   String name
   Collection<String> tweets
}
```

```
def thread1 = Thread.start {
   println "Hello from ${Thread.currentThread().name}"
}


def thread2 = Thread.startDaemon {
   println "Hello from ${Thread.currentThread().name}"
}


[ thread1, thread2 ]*.join()
```

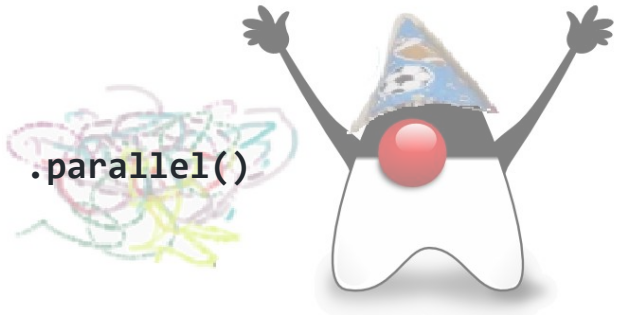
```
def process = (['git', 'status']).execute([], new File('.'))
def processOutput = new StringWriter()
process.consumeProcessOutput processOutput, processOutput
process.waitFor()
println processOutput.toString().trim()
```

```
class SynchronizedCounter {

    int atomicCounter
    int counter

    @Synchronized
    int incrementAndGet() {
        atomicCounter = atomicCounter + 1
        return atomicCounter
    }

    @WithReadLock
    int value() {
        counter
    }

    @WithWriteLock
    void increment() {
        counter = counter + 1
    }
}
```

```
def findMostPopularTweet = { Map params ->
   String topic = params.about
   Collection tweets = params.from

   tweets.findAll({ Status tweet ->
       tweet.text.toLowerCase().contains(topic.toLowerCase())
   }).findAll({ Status tweet ->
       !tweet.retweet
   }).collect ({ Status tweet ->
       [
           user: tweet.user.screenName,
           text: tweet.text,
           favourited: tweet.favoriteCount,
           retweeted: tweet.retweetCount
       ]
   }).max({
       it.favourited + it.retweeted
   })
}



def result = findMostPopularTweet about: 'gpars', from: jeeconfTweets
```

```
def findMostPopularTweet = { Map params ->
   String topic = params.about
   Collection tweets = params.from

   tweets.findAll({ Status tweet ->
       tweet.text.toLowerCase().contains(topic.toLowerCase())
   }).findAll({ Status tweet ->
       !tweet.retweet
   }).collect ({ Status tweet ->
       [
           user: tweet.user.screenName,
           text: tweet.text,
           favourited: tweet.favoriteCount,
           retweeted: tweet.retweetCount
       ]
   }).max({
       it.favourited + it.retweeted
   })
}

ParallelEnhancer.enhanceInstance(jeeconfTweets)
jeeconfTweets.makeConcurrent()

def result = findMostPopularTweet about: 'gpars', from: jeeconfTweets
```

```
Optional<Status> mostPopularTweet = tweets.stream()
```




```
        .filter(tweet -> tweet.getText().toLowerCase().contains(topic.toLowerCase()))
        .filter(tweet -> !tweet.isRetweet())
        .max(comparingInt(tweet -> tweet.getFavoriteCount() + tweet.getRetweetCount()));
```

WITH
POOL

```groovy
def findMostPopularTweet = { Map params ->
    String topic = params.about
    Collection tweets = params.from

    withPool {
        tweets.parallel.filter({ Status tweet ->
            tweet.text.toLowerCase().contains(topic.toLowerCase())
        }).filter({ Status tweet ->
            !tweet.retweet
        }).map({ Status tweet ->
            [
                user: tweet.user.screenName,
                text: tweet.text,
                favourited: tweet.favoriteCount,
                retweeted: tweet.retweetCount
            ]
        }).max({
            it.favourited + it.retweeted
        })
    }
}

def result = findMostPopularTweet about: 'gpars', from: jeeconfTweets
```

```
withPool {

    Closure fetchLatestTweets = twitter.&fetchLatestTweets.asyncFun()
    Closure determineTopic = topics.&determineTopic.asyncFun()
    Closure aggregateTopics = topics.&aggregateTopics.asyncFun()
    Closure fetchNewsAbout = news.&fetchNewsAbout.asyncFun()
    Closure filterMostImportant = news.&filterMostImportant.asyncFun()


    Promise topics = determineTopic(fetchLatestTweets)


    Promise aggregatedTopics = topics.get().inject({ t1, t2 -> aggregateTopics(t1, t2) })


    aggregatedTopics.then {
        fetchNewsAbout(it)
    }.then {
        filterMostImportant(it)
    }.then {
        println it
    }.join()
}
```

```
withPool {

    Closure fetchLatestTweets = twitter.&fetchLatestTweets.asyncFun()
    Closure determineTopic = topics.&determineTopic().asyncFun()
    Closure aggregateTopics = topics.&aggregateTopics.gmemoize()
    Closure fetchNewsAbout = news.&fetchNewsAbout.asyncFun()
    Closure filterMostImportant = news.&filterMostImportant.asyncFun()


    Promise topics = determineTopic(fetchLatestTweets)


    Promise aggregatedTopics = topics.get().inject({ t1, t2 -> aggregateTopics(t1, t2) })


    aggregatedTopics.then {
        fetchNewsAbout(it)
    }.then {
        filterMostImportant(it)
    }.then {
        println it
    }.join()
}
```

AGENTS

```
def stringGuard = new Agent<String>()
28.times {
    Thread.start {
        stringGuard { updateValue(it + '|' + Thread.currentThread().name) }
    }
}
println stringGuard.val

def listGuard = new Agent<List<String>>()
43.times {
    Thread.start {
        listGuard { it.add(Thread.currentThread().name) }
    }
}
listGuard.valAsync {
    println it
}
```

```
final Agent<String> stringGuard = new Agent<>();
stringGuard.send(new MessagingRunnable<String>() {
    @Override
    protected void doRun(String value) {
        stringGuard.updateValue(value + '|' + Thread.currentThread().getName());
    }
});
System.out.println(stringGuard.getVal());

final Agent<List<String>> listGuard = new Agent<>();
listGuard.send(new ArrayList<>());
listGuard.send(new MessagingRunnable<List<String>>() {
    @Override
    protected void doRun(List<String> value) {
        value.add(Thread.currentThread().getName());
    }
});
listGuard.valAsync(new MessagingRunnable<List<String>>() {
    @Override
    protected void doRun(List<String> value) {
        System.out.println(value);
    }
});
```

ACTORS

DATAFLOWS

GPARS PROS

GPARS CONS

FURTHER DIVE

QA

THANK YOU