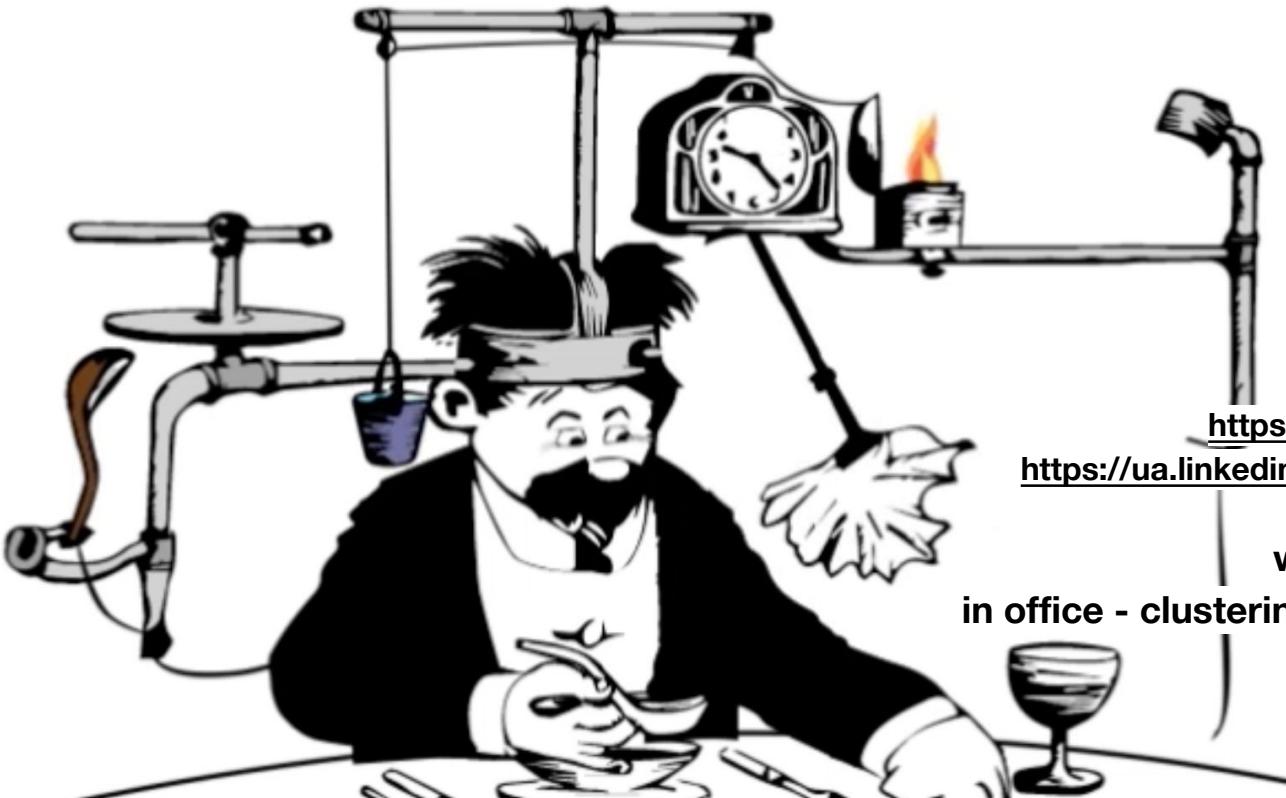


GPar: Unsung Hero of Concurrency in Practice



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

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?



HOW TO STAY
FOCUSED

guideline #2 for developing concurrent code

MURPHY'S LAW:

Anything that can go wrong will go wrong.

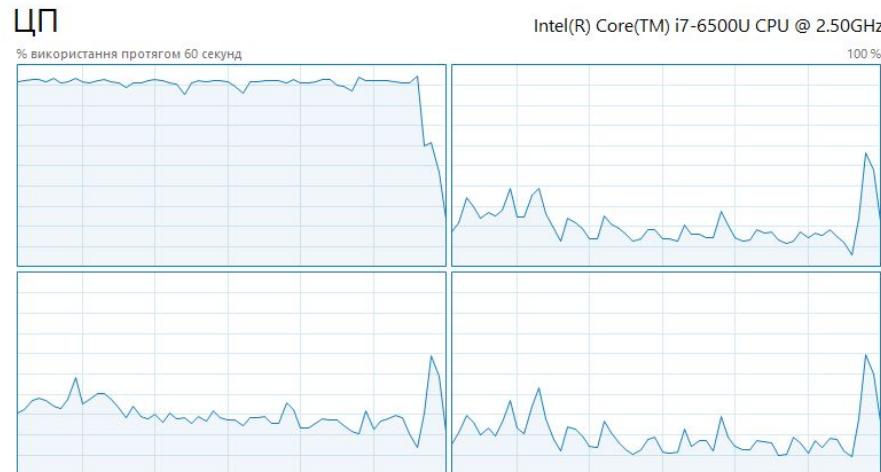


```
public class Holder {  
    private int n;  
  
    public Holder(int n) { this.n = n; }  
  
    public void assertSanity() {  
        if (n != n) {  
            throw new AssertionError("Even it can go wrong!");  
        }  
    }  
  
    public class Initializer {  
  
        public Holder holder;  
  
        public void init() { holder = new Holder(42); }  
    }  
}
```

guideline #1 for developing concurrent code

MOORE'S LAW:

Computers will get exponentially faster.



Використання	Швидкість	Максимальна швидкість:
23%	2,46 ГГц	2,59 ГГц
Процеси	Потоки	Дескриптори
165	2292	73885
Час роботи		
0:01:01:26		
Сокети:	1	
Ядра:	2	
Логічних процесорів:	4	
Віртуалізація:		Вимкнuto
Підтримка технології Hyper-V:	Так	
Кеш 1 рівня:	128 КБ	
Кеш 2 рівня:	512 КБ	
Кеш 3 рівня:	4,0 МБ	

The **Thread** class

802

Thinking in Java

Bruce Eckel

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!),
*///:-
```

```
    print(this + " completed");
}
public String toString() {
    return String.format("%12s", name);
}

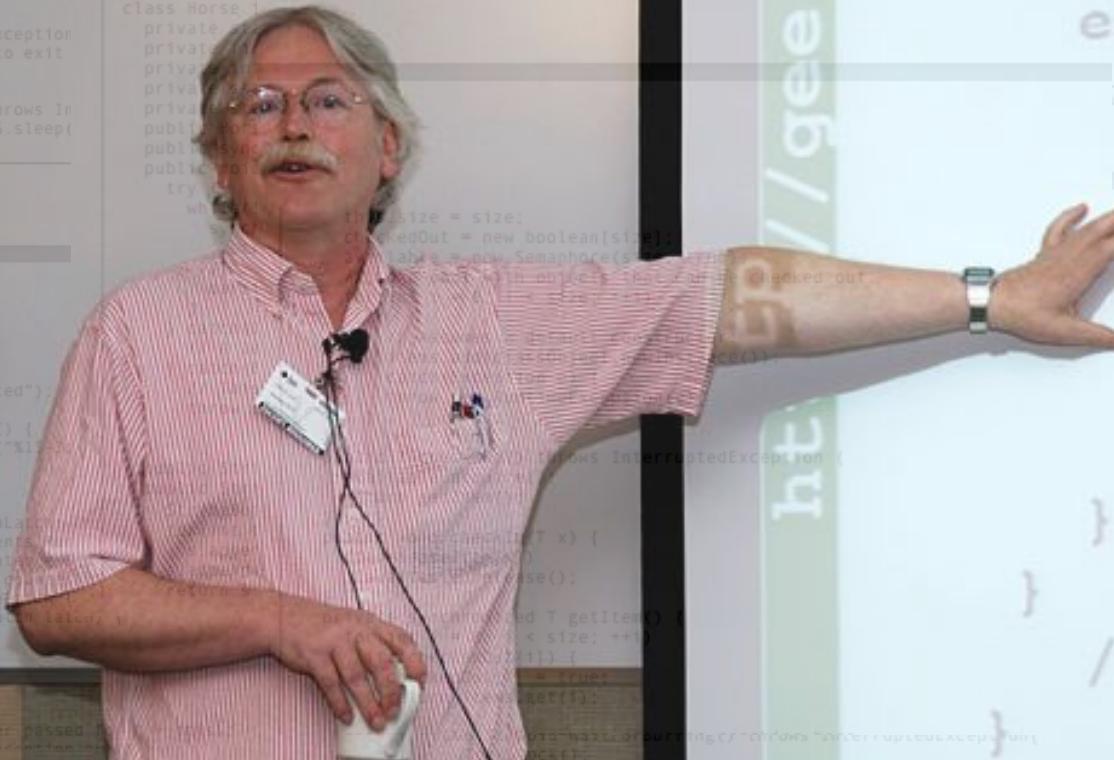
// Waits on the CountDownLatch
class WaitingTask implements Runnable {
    private static int counter = 0;
    private final int id = counter++;
    private final CountDownLatch latch;
    WaitingTask(CountDownLatch latch) {
        this.latch = latch;
    }
    public void run() {
        try {
            latch.await();
            print("Latch barrier passed " + id);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

ht

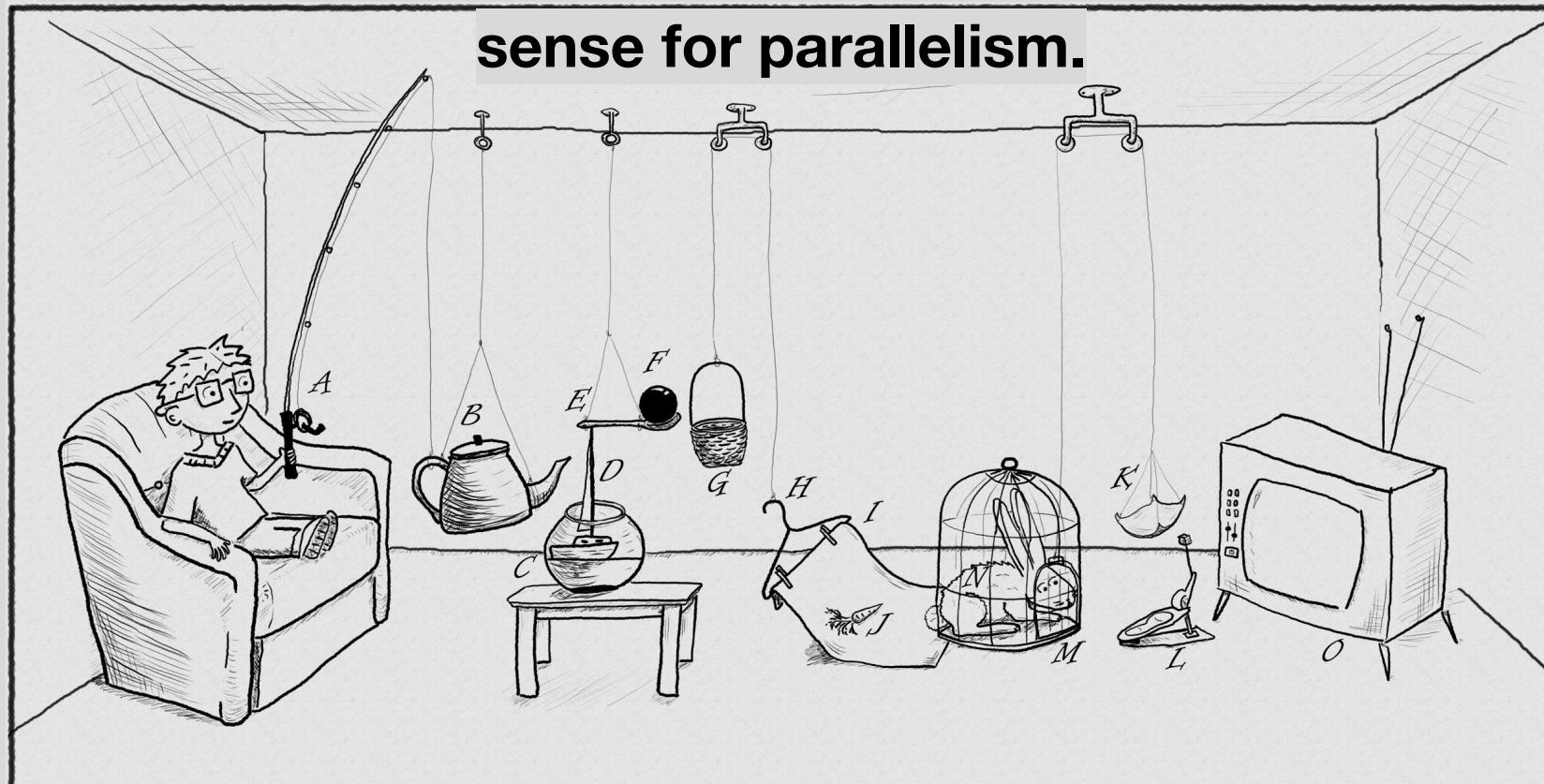
```
//: concurrency/Pool.java
// Using a Semaphore inside a Pool, to restrict
// the number of tasks that can use a resource.
import java.util.concurrent.*;
import java.util.*;

public class Pool<T> {
    private int size;
    private List<T> items = new ArrayList<T>();
    private volatile boolean[] checkedOut;
    private Semaphore available;
    public Pool(Class<T> classObject, int size) {
```

Thinking in Java



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

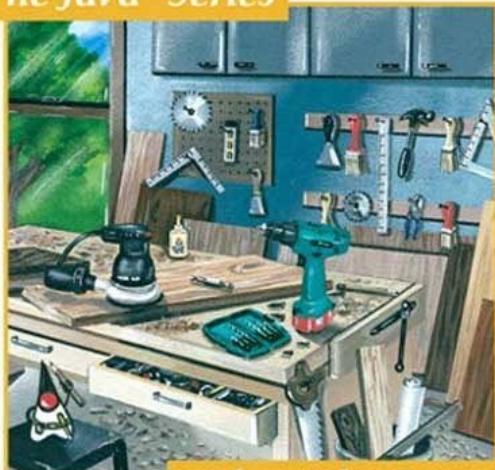


Joshua Bloch

Revised and
Updated for
Java SE 6

Effective Java™ Second Edition

The Java™ Series



...from the Source



10 Concurrency.....	259
Item 66: Synchronize access to shared mutable data.....	259
Item 67: Avoid excessive synchronization	265
Item 68: Prefer executors and tasks to threads.....	271
Item 69: Prefer concurrency utilities to <code>wait</code> and <code>notify</code>	273

CONTENTS

Item 70: Document thread safety	278
Item 71: Use lazy initialization judiciously	282
Item 72: Don't depend on the thread scheduler	286
Item 73: Avoid thread groups	288



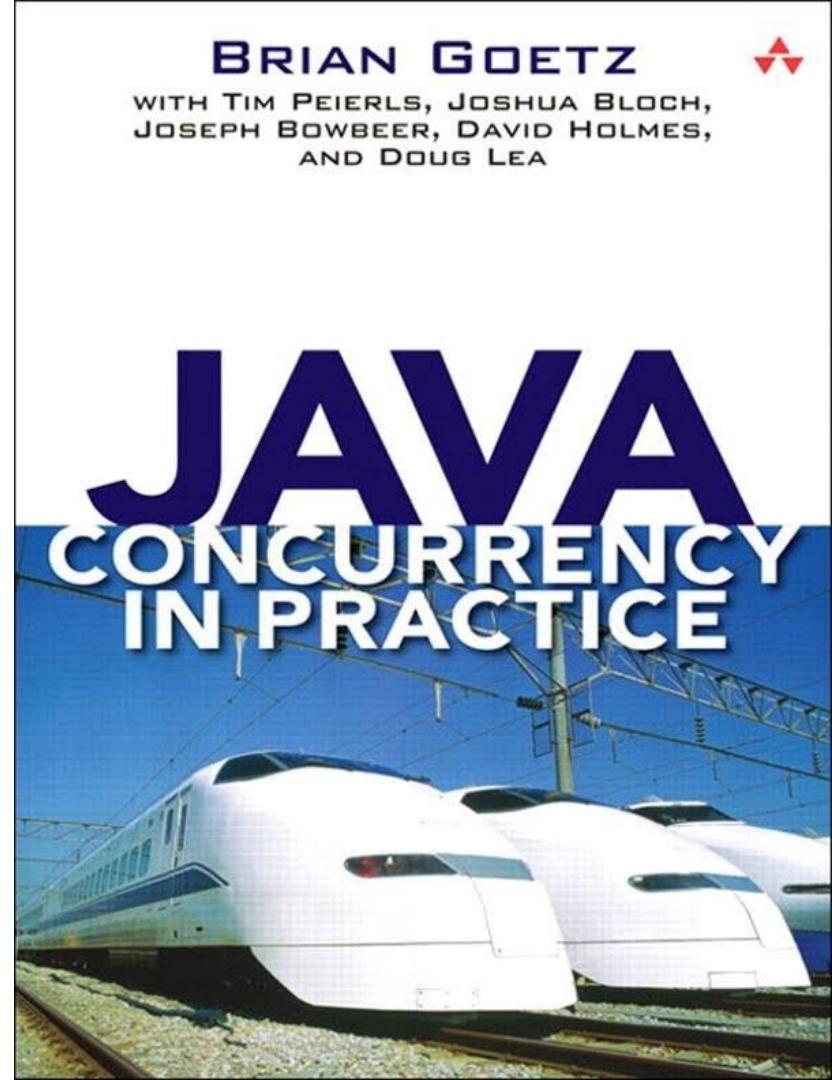
If multiple threads access the same mutable state variable without appropriate synchronization, your program is broken.

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()));

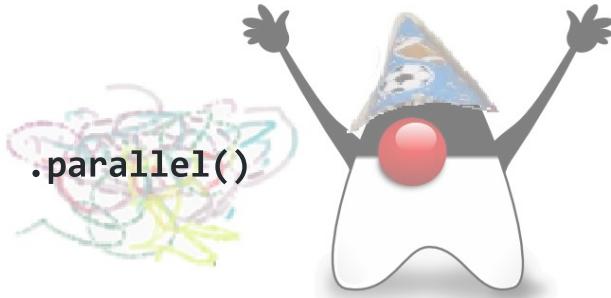
    leftTask.fork(); rightTask.fork();

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

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



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



```
.parallel()  
.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

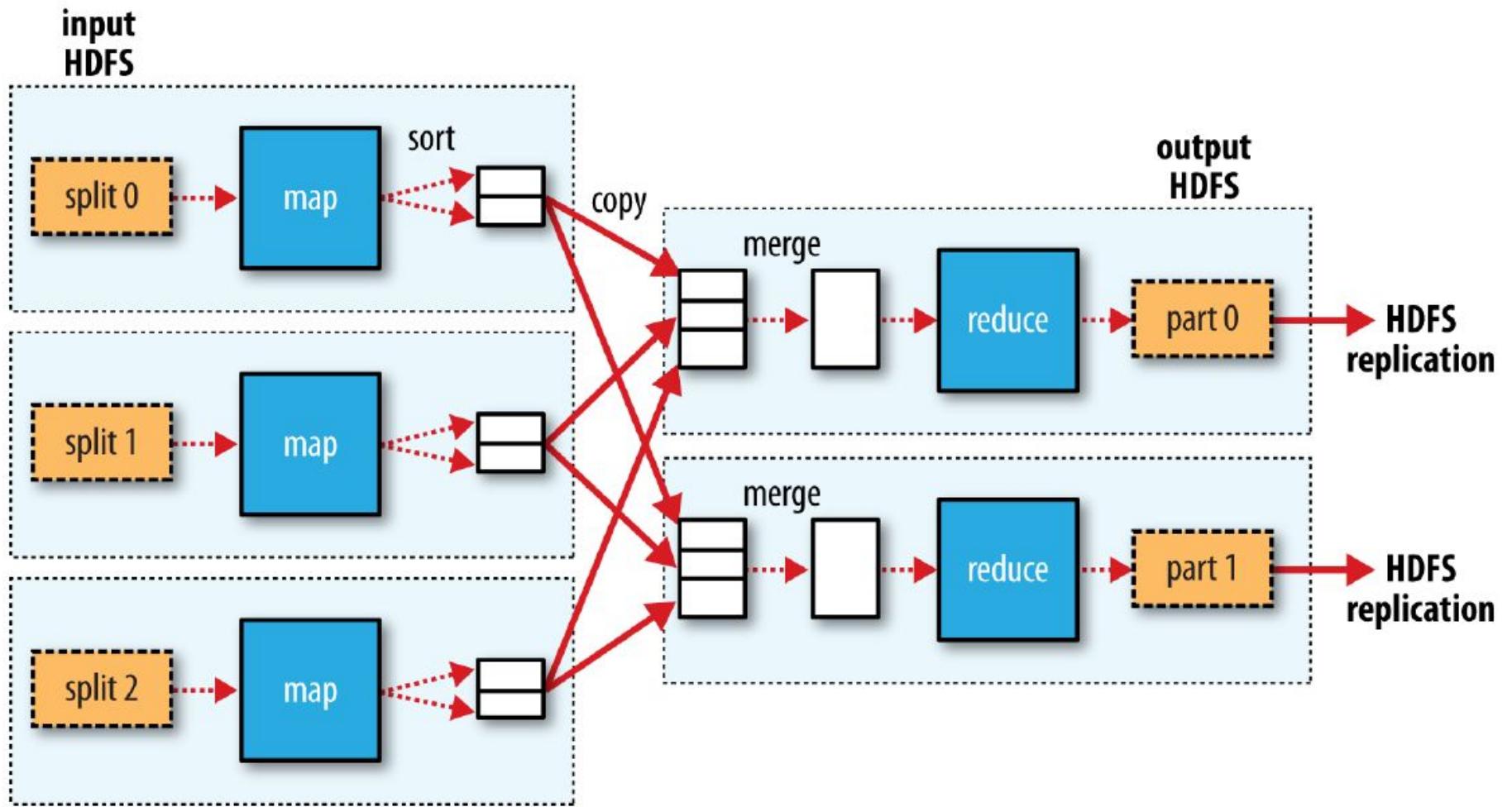
In Spring's approach to building RESTful web services, HTTP requests are handled by a controller. These components are easily identified by the `@RestController` annotation, and the `GreetingController` below handles `GET` requests for `/greeting` by returning a new instance of the `Greeting` class:

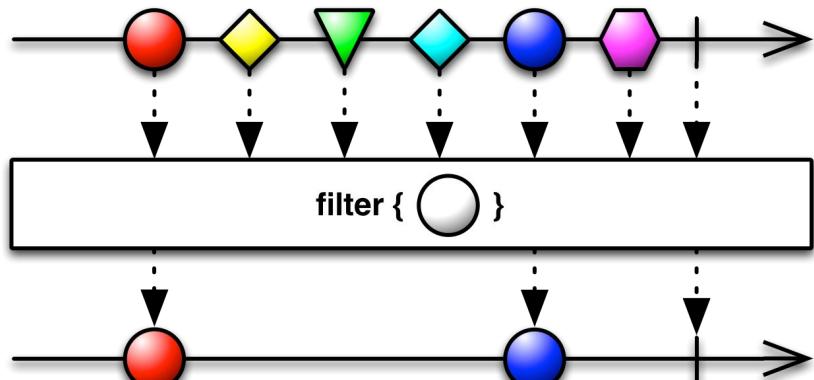
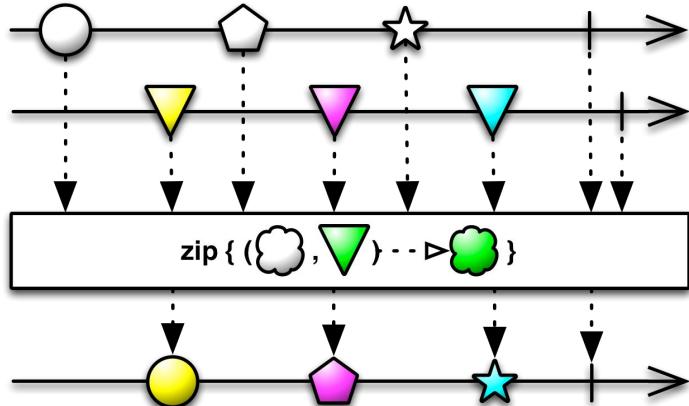
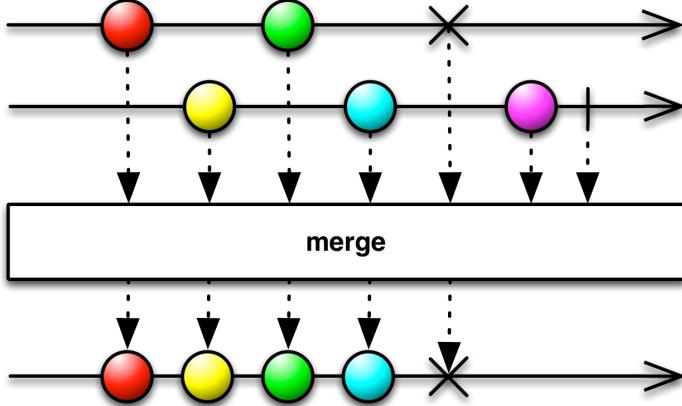
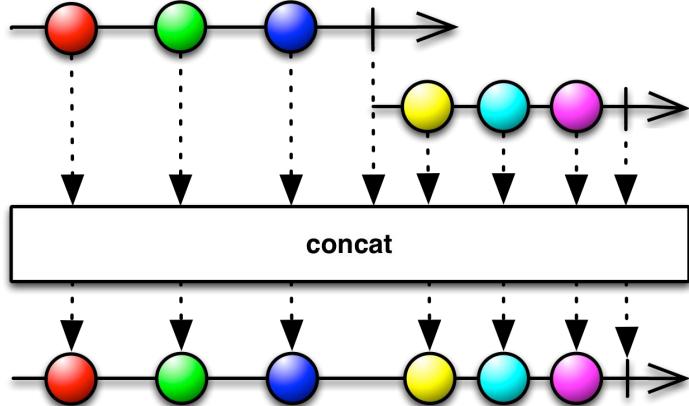
src/main/java/hello/GreetingController.java

```
@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));
    }
}
```

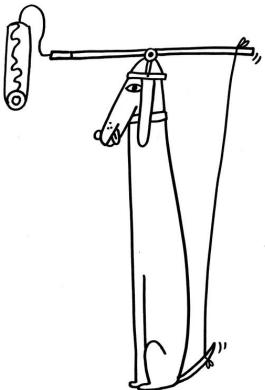








HOW TO BRUSH
YOUR TEETH



HOW TO UTILIZE YOUR
DOG'S MINDLESS JOY

GPars:

data parallelism

map/reduce

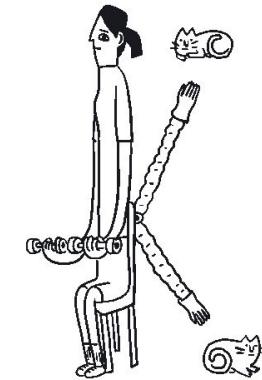
fork/join

asynchronous closures

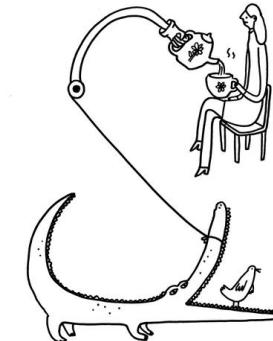
agents

actors

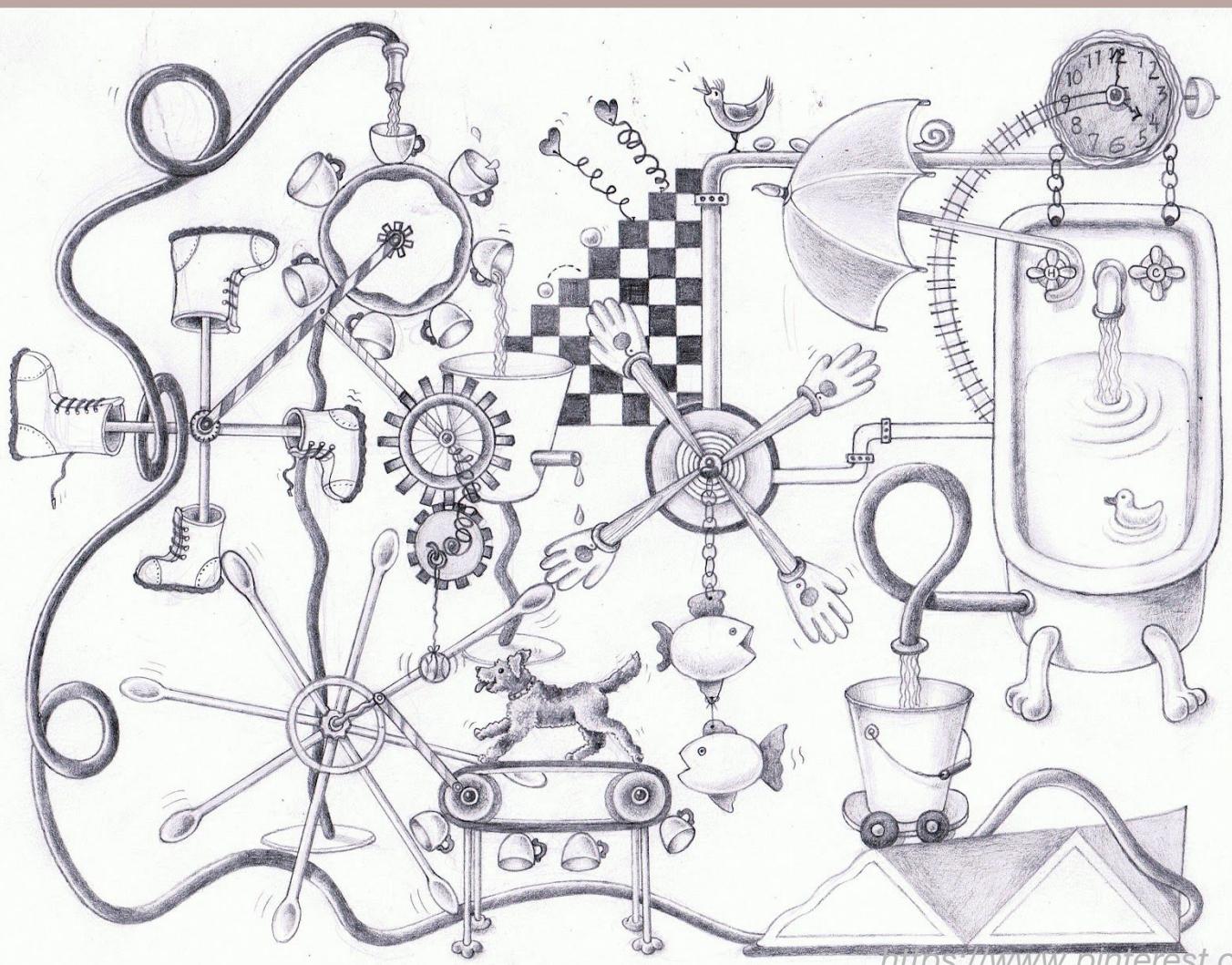
dataflows



HOW TO PET YOUR PETS
WHILE DOING YOUR REPS

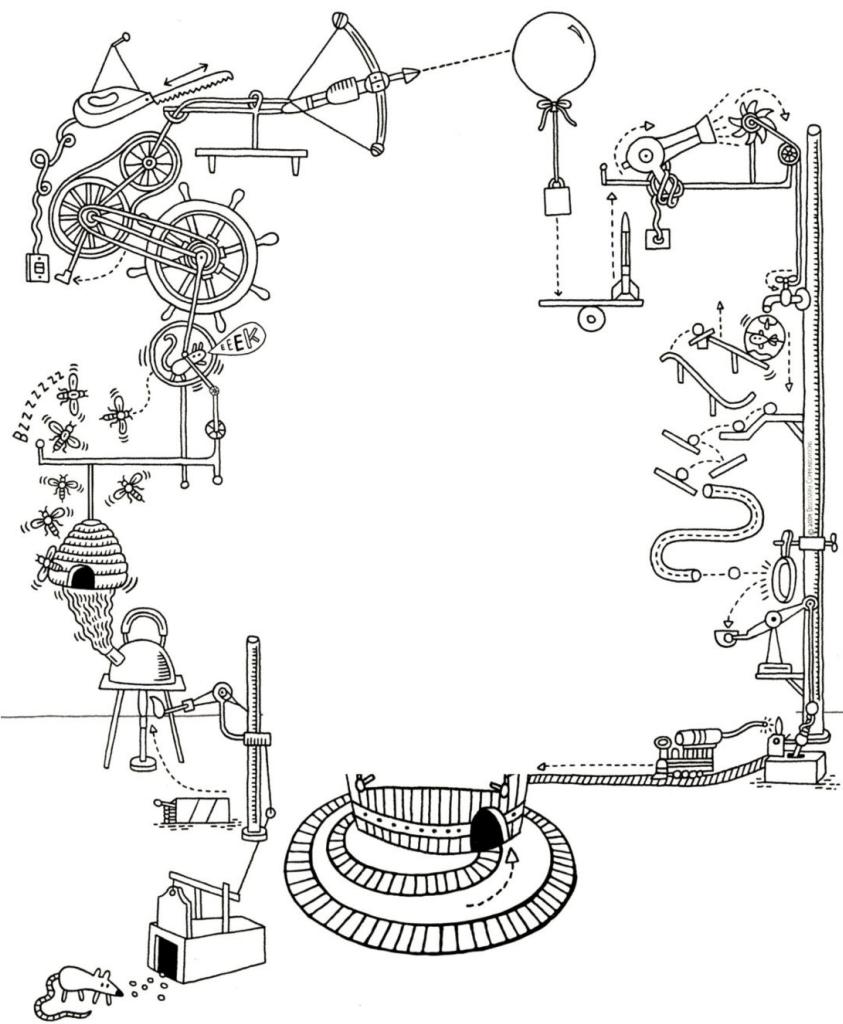


HOW TO SERVE TEA



DEMO

JAVA OR GROOVY?



```
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;
}
```

```
@Immutable class ImmutableGroovyPerson {  
  
    String name  
    Collection<String> tweets  
}
```

**CLICK TO PLACE
YOUR
AD
HERE!**

```
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

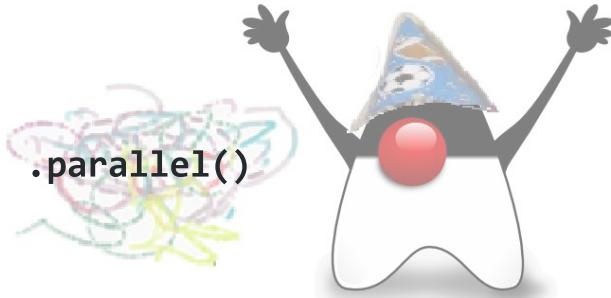
    tweets.findAll{ Status tweet ->
        tweet.text.toLowerCase().contains(topic.toLowerCase())
    }.findAll{ Status tweet ->
        !tweet.retweet
    }.collect { Status tweet ->
        [
            user: tweet.user.screenName,
            text: tweet.text,
            favourited: tweet.favoriteCount,
            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()
```



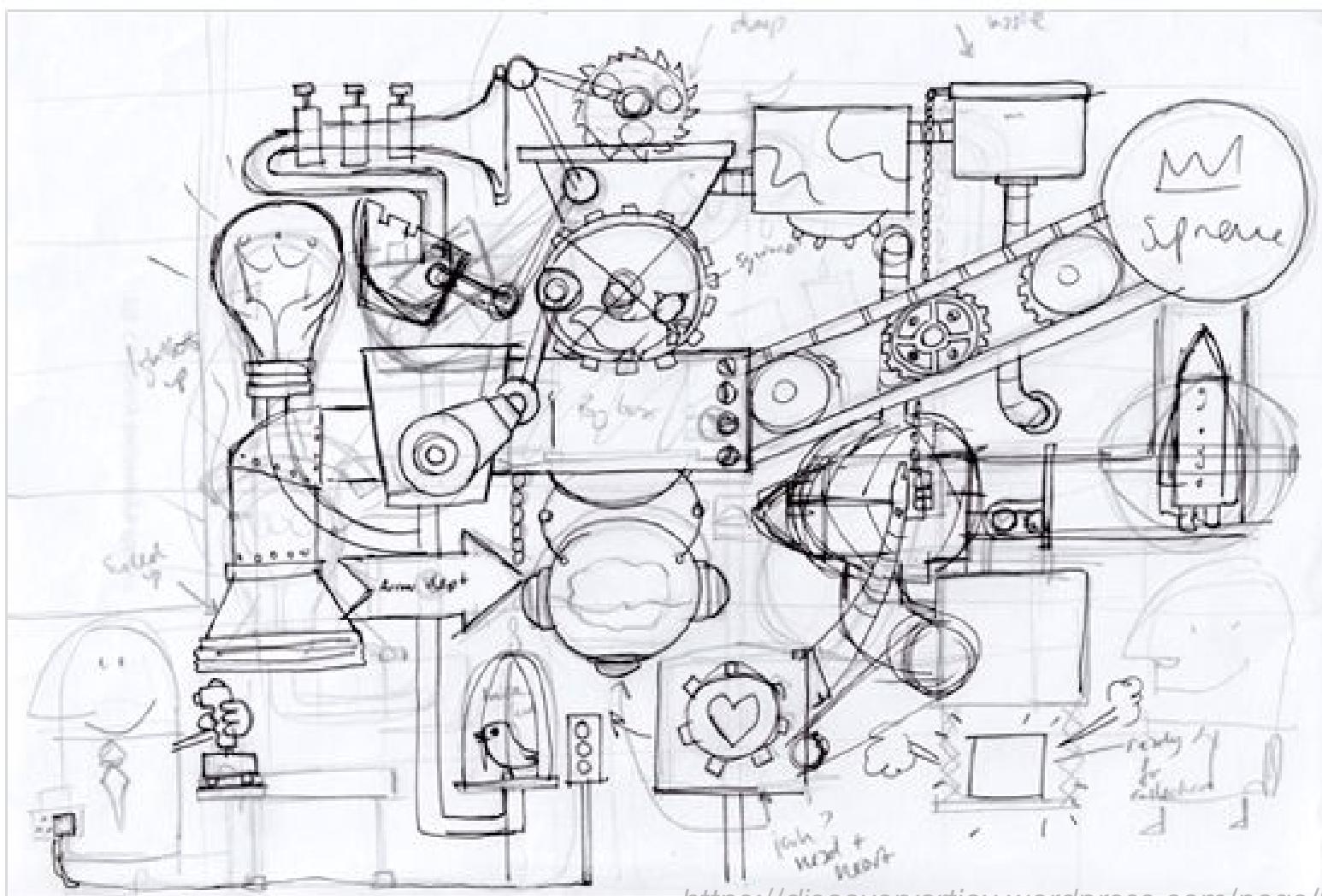
```
.parallel()  
.filter(tweet -> tweet.getText().toLowerCase().contains(topic.toLowerCase()))  
.filter(tweet -> !tweet.isRetweet())  
.max(comparingInt(tweet -> tweet.getFavoriteCount() + tweet.getRetweetCount()));
```

```
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
```

WITH POOL



```
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()  
}
```

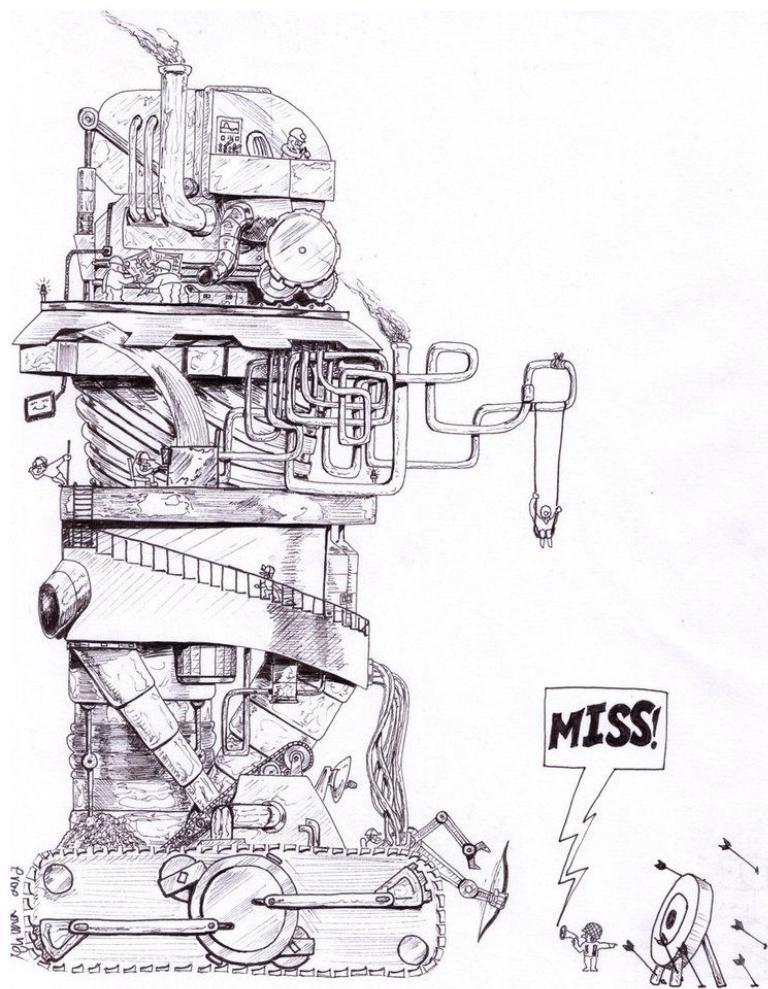
```
withPool {
```

```
    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

<http://www.keywordsuggests.com/DT5zG9YMrnEK707aqUa2lhkNs5vUwu3mV6WrrLHR0soHddyR42EFvbEkYg9kaqswD%7CyremGnEFLO%7CzJXdp0SMQ/>

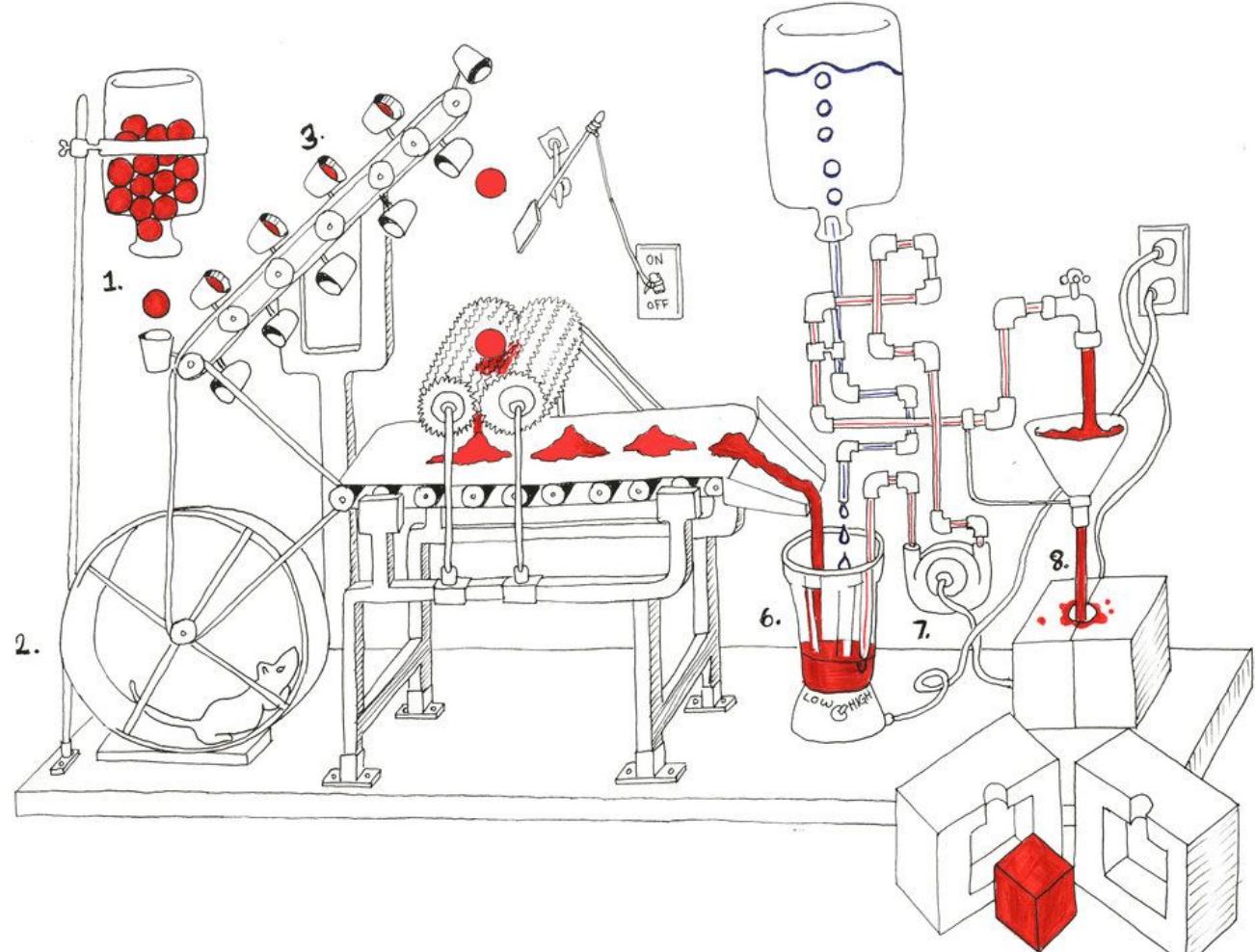
```
def stringGuard = new Agent<String>()
28.times {
    Thread.start {
        stringGuard { updateValue(it + '|' + Thread.currentThread().name) }
    }
}
println stringGuard.val

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



```
def main = actor {
    loop {
        fetchLatestNews.send()
        react { latestNews ->
            extractText latestNews

            react { text -> save text }
        }
    }
}

def fetchLatestNews = reactor {
    news.fetchLatest()
}

def extractText = messageHandler {
    when { Tweet tweet -> reply tweet.status }
    when { FacebookPost facebookPost -> reply facebookPost.post }
    when { BlogArticle blogArticle -> reply blogArticle.text }
}

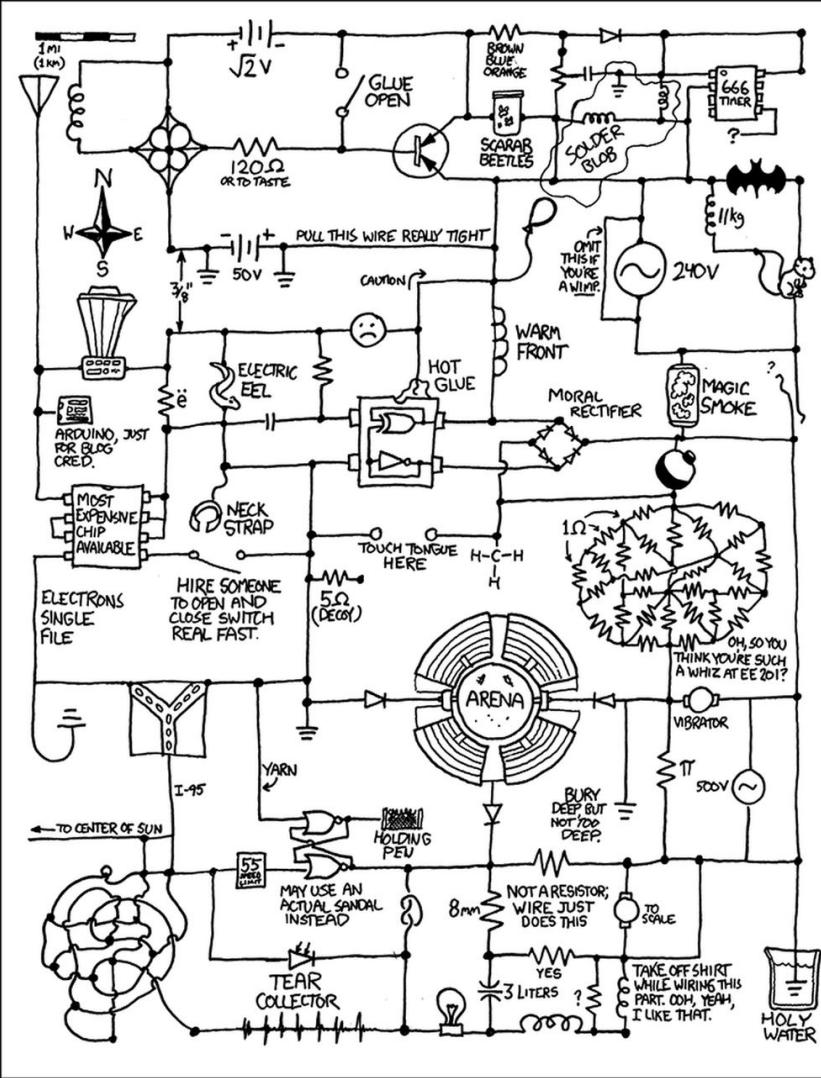
def save = reactor { text ->
    println text
}
```

```
final Closure handler = new ReactorMessagingRunnable() {

    @Override
    protected Object doRun(Object argument) {
        return news.fetchLatest();
    }
};

final Actor actor = new ReactiveActor(handler);
actor.start();
```

```
public class MessageHandler extends DynamicDispatchActor {  
  
    public void onMessage(final Tweet tweet) {  
        replyIfExists(tweet.getStatus());  
    }  
  
    public void onMessage(final FacebookPost facebookPost) {  
        replyIfExists(facebookPost.getPost());  
    }  
  
    public void onMessage(final BlogArticle blogArticle) {  
        replyIfExists(blogArticle.getText());  
    }  
}
```



DATAFLOWS

<http://www.projectroomseattle.org/programs-content/2014/6/preparing-for-the-failure-variet...the-rube-goldberg-confessionals>

```
final Dataflow dataflow = new Dataflows()

task {
    dataflow.numberOfInteractions = dataflow.retweets + dataflow.facebookLikes
}

task {
    dataflow.retweets = twitter.numberOfRetweets(article)
}

task {
    dataflow.facebookLikes = facebook.numberOfLikes(article)
}

println dataflow.numberOfInteractions
```

```
final DataflowVariable<Integer> numberOfInteractions = new DataflowVariable<>();
final DataflowVariable<Integer> retweets = new DataflowVariable<>();
final DataflowVariable<Integer> facebookLikes = new DataflowVariable<>();

task((Runnable) () -> {
    try {
        numberOfInteractions.bind(retweets.getVal() + facebookLikes.getVal());
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }
});

task(() -> retweets.bind(twitter.numberOfRetweets(article)));

task(() -> facebookLikes.bind(facebook.numberofLikes(article)));

System.out.println(numberOfInteractions.getVal());
```

```
def retweets = new DataflowQueue()
def facebookLikes = new DataflowQueue()
def numberOfInteractions = new DataflowQueue()

operator(
    inputs: [ retweets, facebookLikes ],
    outputs: [ numberOfInteractions ],
    { retweetsCount, facebookLikesCount ->
        numberOfInteractions << retweetsCount + facebookLikesCount
    }
)

task {
    ALL_ARTICLES.each { article -> retweets << twitter.numberOfRetweets(article) }
}

task {
    ALL_ARTICLES.each { article -> facebookLikes << facebook.numberofLikes(article) }
}
```

```
final DataflowQueue retweets = new DataflowQueue();
final DataflowQueue facebookLikes = new DataflowQueue();
final DataflowQueue numberOfInteractions = new DataflowQueue();

operator(
   .asList(retweets, facebookLikes),
   .asList(numberOfInteractions),
    new DataflowMessagingRunnable(2) {

    @Override
    protected void doRun(Object... arguments) {
        getOwningProcessor().bindOutput((Integer)arguments[0] + (Integer)arguments[1]);
    }
});

task(() -> ALL_ARTICLES.forEach(a -> retweets.bind(twitter.numberOfRetweets(a))));

task(() -> ALL_ARTICLES.forEach(a -> facebookLikes.bind(facebook.numberOfLikes(a))));
```

WHY USE GPARS?

TO IMPROVE PERFORMANCE OF SEQUENTIAL CODE

SIMPLIFY DEVELOPMENT OF CONCURRENT/PARALLEL CODE

USE CLOSE-TO-DOMAIN ABSTRACTIONS

REDUCE NUMBER OF ERRORS

INTEGRATE DIFFERENT ABSTRACTIONS EASILY

WHEN DO USE GPARS? - PARALLEL STREAMS

FURTHER DIVE

Examples from slides

Groovy and Concurrency with GPars by Paul King

Gpars: Concurrency in Java & Groovy by Ken Kousen

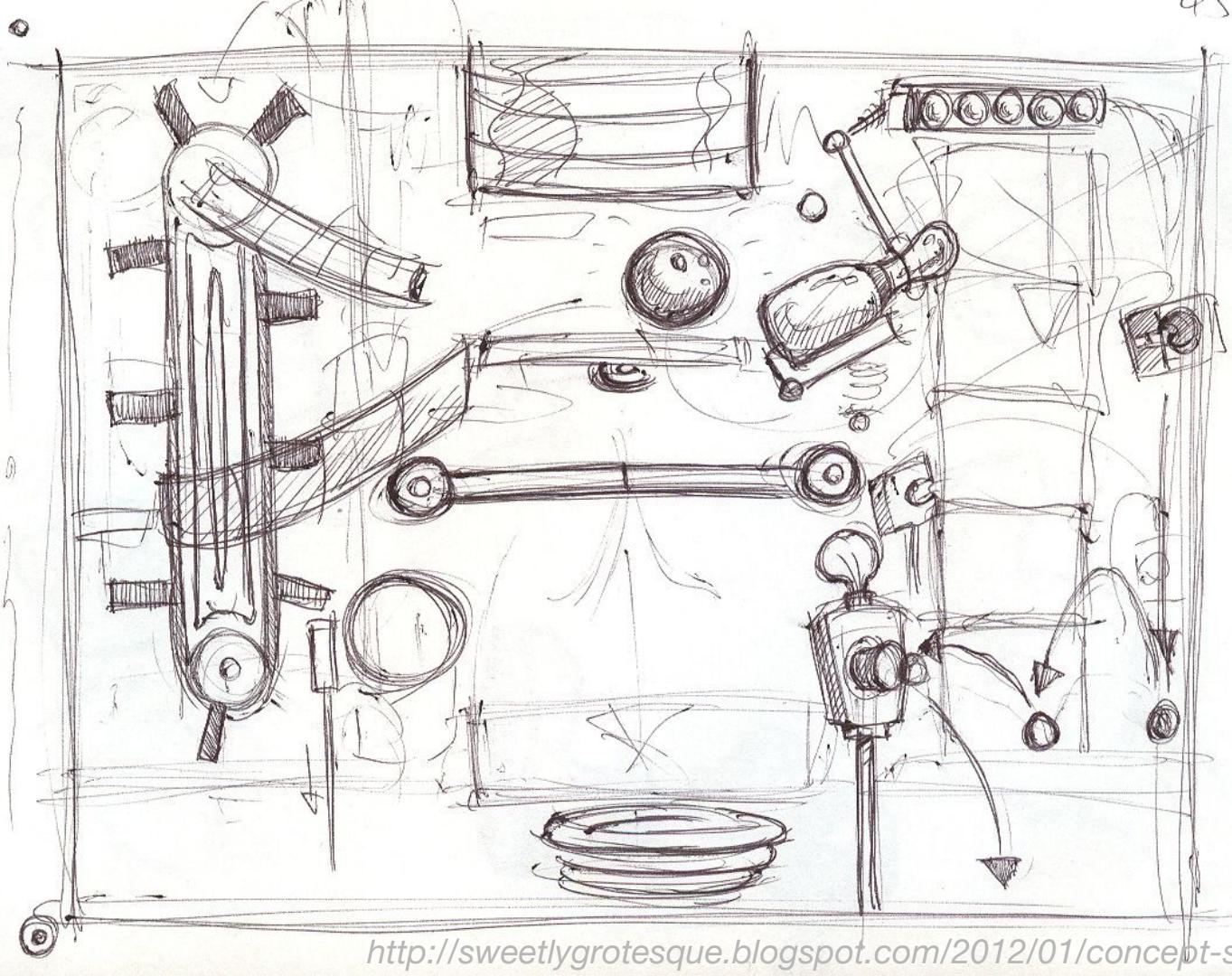
Official site

Sources

Slack channel

The GPars User Guide

'Concurrent Groovy with GPars' chapter from 'Groovy in Action' book



Q/A

**THANK
YOU!**