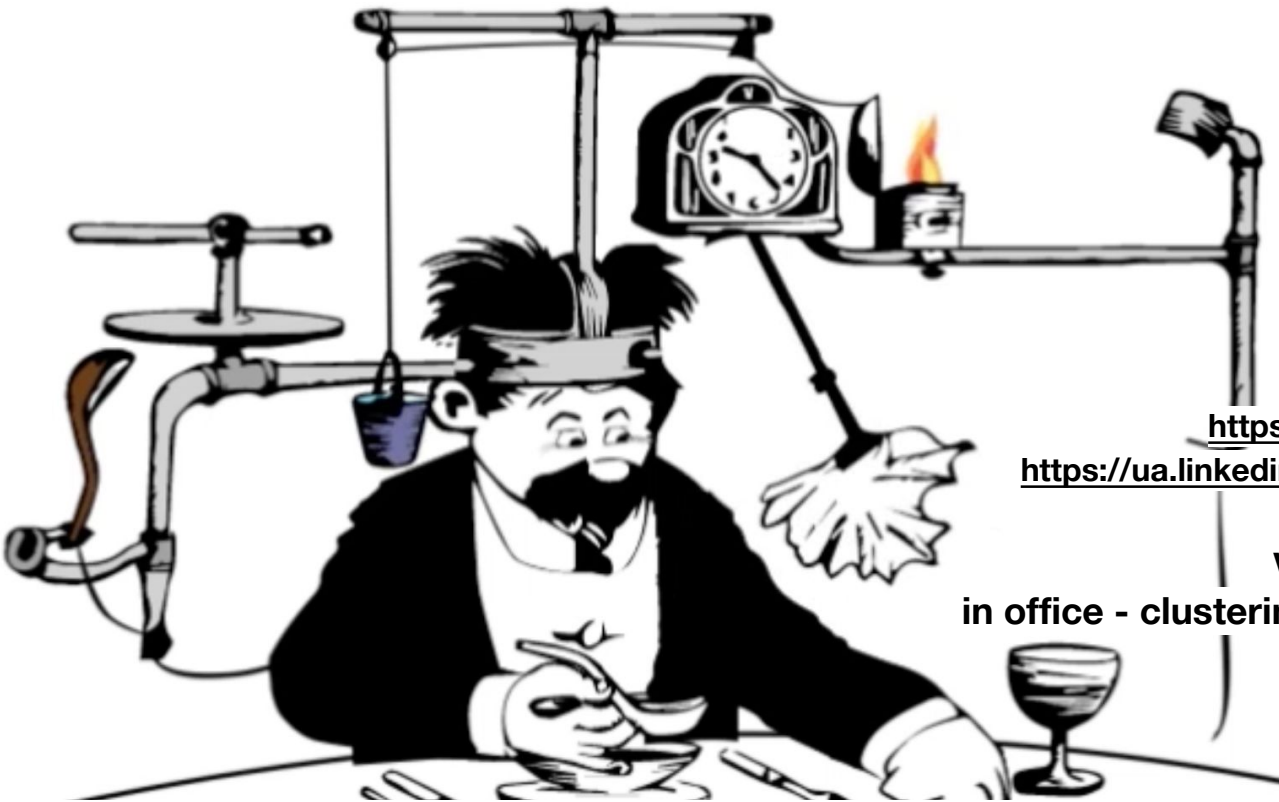


GPar: Unsung Hero of Concurrency in Practice

<http://playstationna.i.lithium.com/t5/image/serverpage/image-id/492281iDADEAFC04868CE5C?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?

GPar's place in this picture

how GPar's handles concurrent/parallel development?

why and when use GPar's?

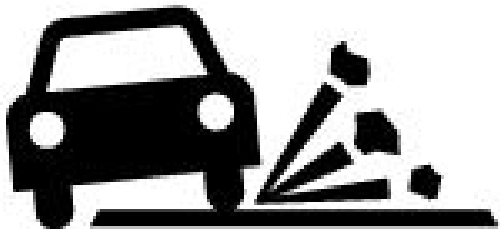
why and when do not use GPar's?



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 {

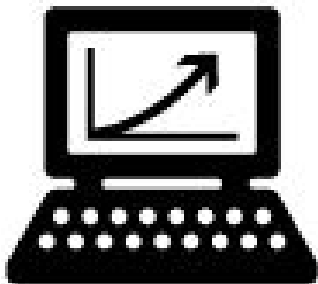
    public Holder holder;

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

guideline #1 for developing concurrent code

MOORE'S LAW:

Computers will get exponentially faster.

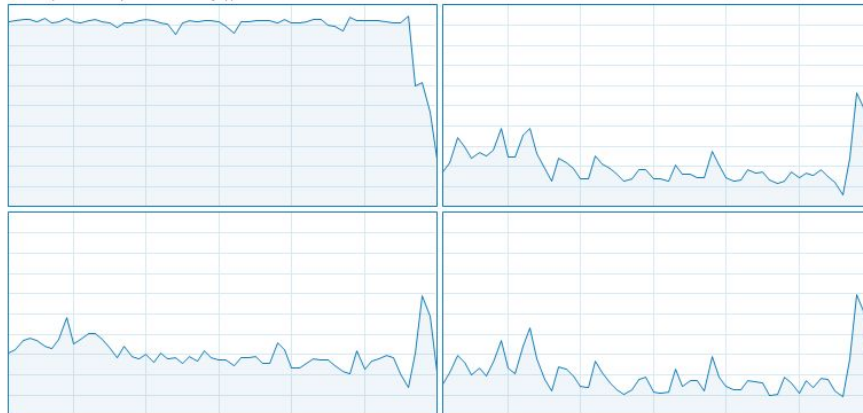


ЦП

Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz

% використання протягом 60 секунд

100 %



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

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/CountDownLatchDemo.java
import java.util.concurrent.*;
import java.util.*;
import static net.mindview.util.Print.*;

// Performs some portion of a task:
class TaskPortion implements Runnable {
    private static int counter = 0;
    private final int id = counter++;
    private static Random rand = new Random(47);
    private final CountDownLatch l
    TaskPortion(CountDownLatch lat
        this.latch = latch;
    }
    public void run() {
        try {
            doWork();
            latch.countDown();
        } catch (InterruptedException
            // Acceptable way to exit
        }
    }
    public void doWork() throws Ir
        TimeUnit.MILLISECONDS.sleep(

```

Concurrency

```
//: concurrer
// Using Cycl
import java.
import java.
import static
```

```
class Horse {
    private
    private
    private
    private
    private
    public
    public
    public
    try
    while

```

```
//: 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) {

```

890

Thinking in Java

Bruce Eckel

ce Eckel

//gee.cs.osv

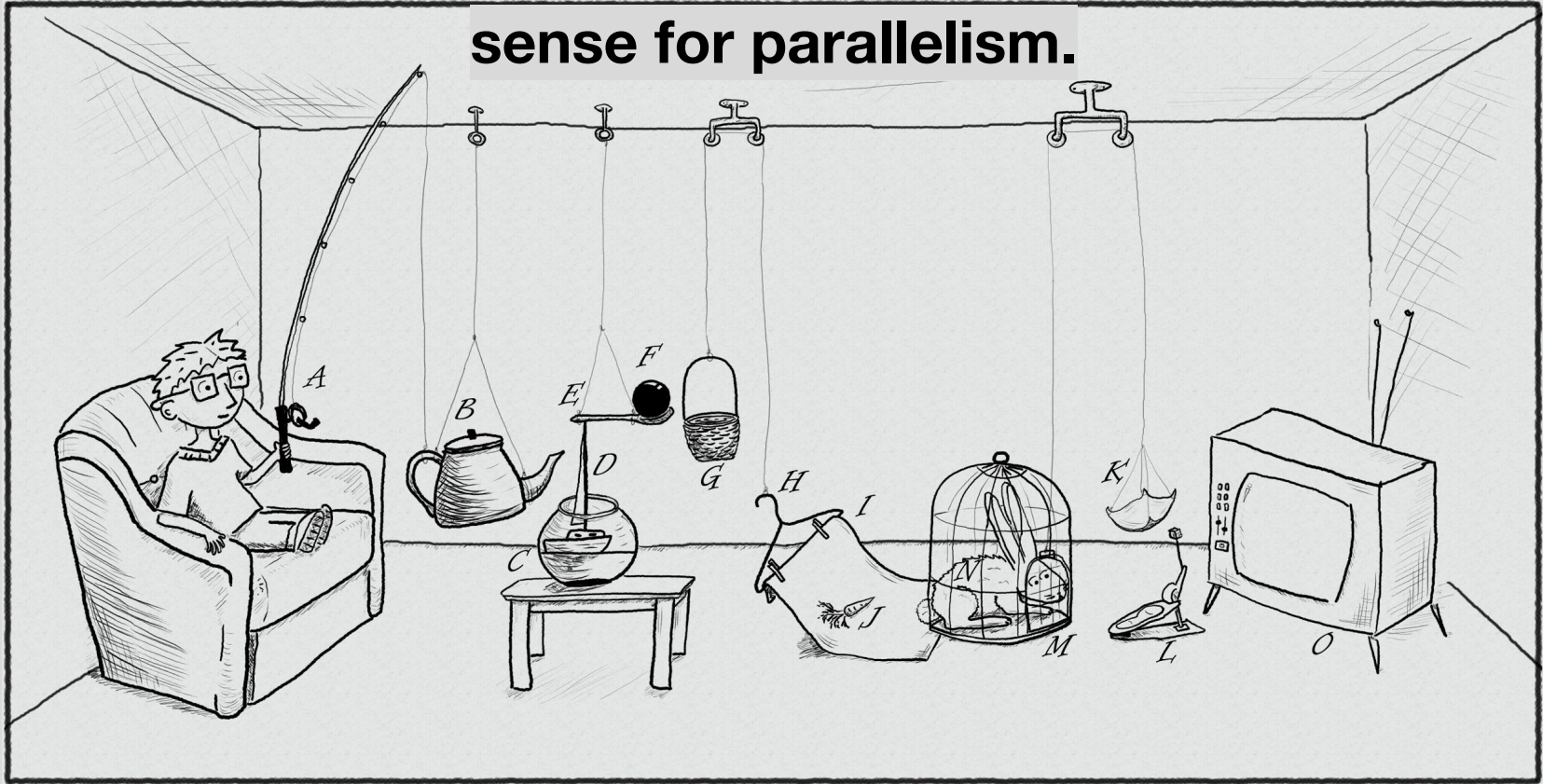
```
protected v
if (hi -
sequent
else {
int m =
SortTas
r.fork
new So
r.join
merge (
}
}
// ...
}
```

```
print(this + "completed");
}
public String toString() {
    return String.format("%12s",
}
}
// Waits on the CountDownLatch
class WaitingTask implements
    private static int counter = 0;
    private final int id = counter++;
    private final CountDown
    WaitingTask(CountDownLac
        this.latch = latch;
    }
    public void run() {
        try {
            latch.await();
            print("Latch barrier passed");

```

890

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



Joshua Bloch

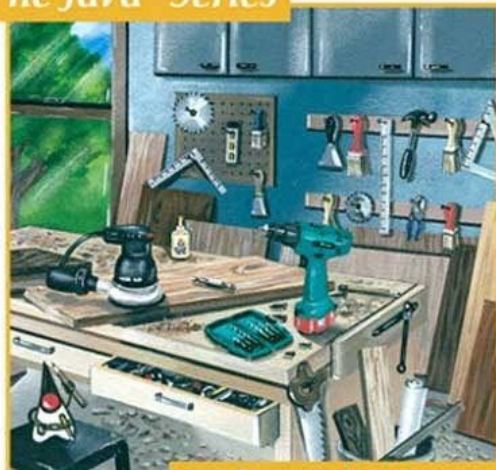
Revised and
Updated for
Java SE 6



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 wait and notify.....	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

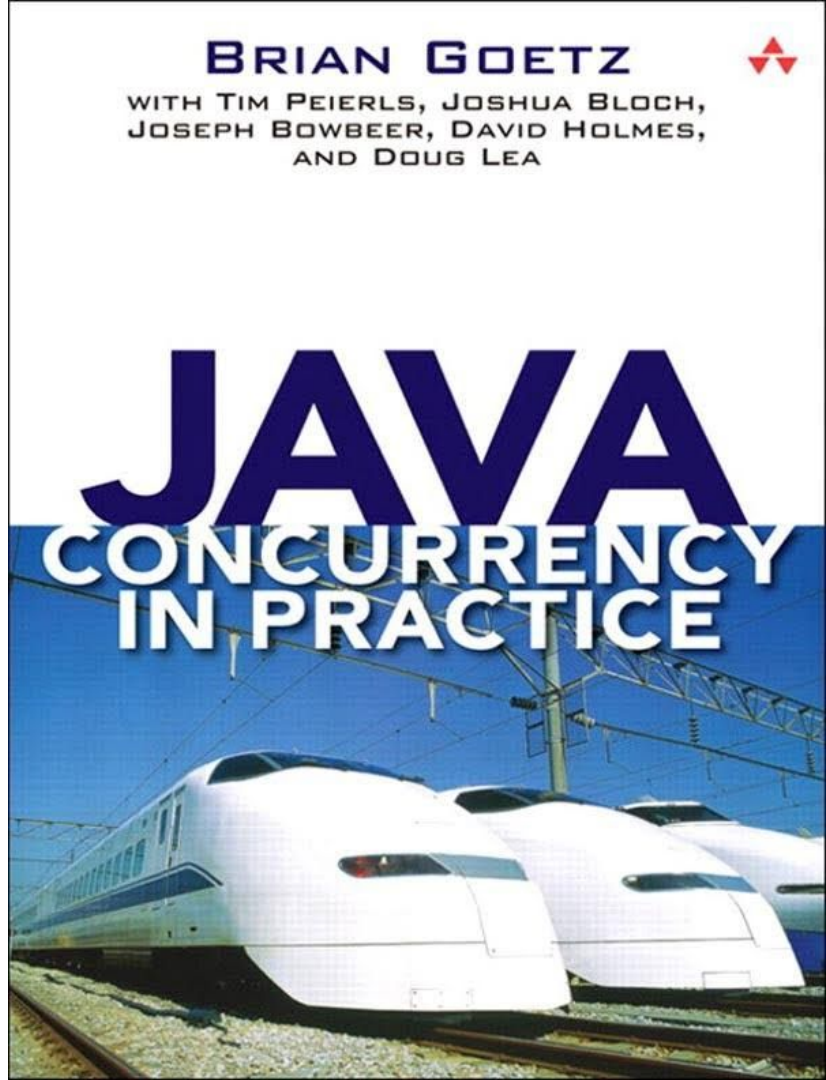
If multiple threads access the same mutable state variable without appropriate synchronization, your program is broken.

There are three ways to fix it:

Don't share the state variable across threads

Make the state variable immutable

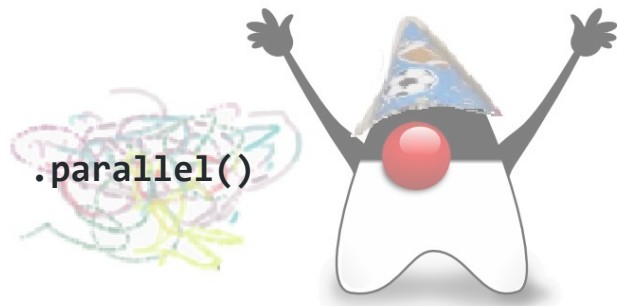
Use synchronization whenever accessing the state variable



```
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:

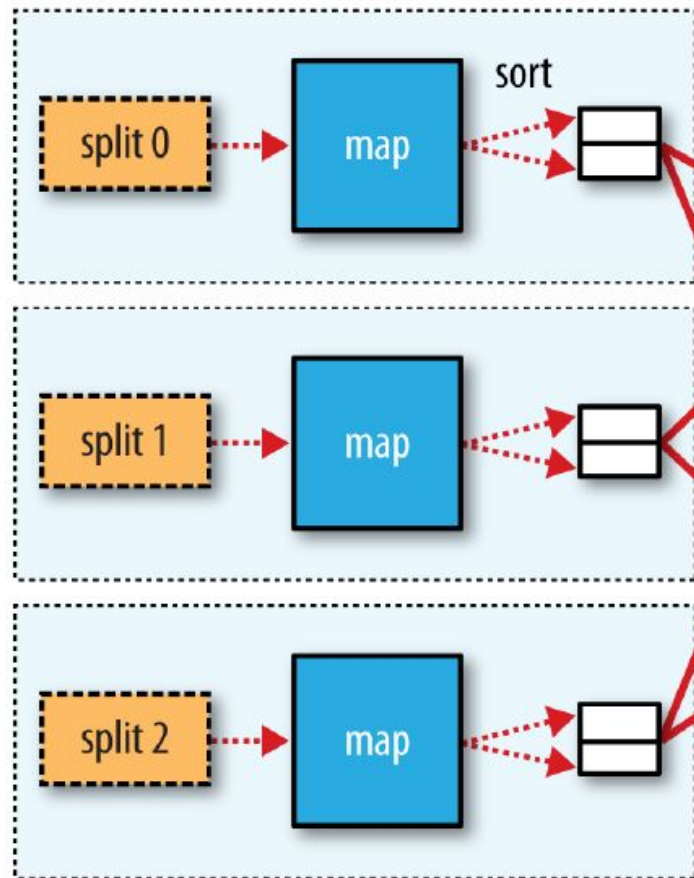
```
src/main/java/hello/GreetingController.java
```

```
@RestController
public class GreetingController {

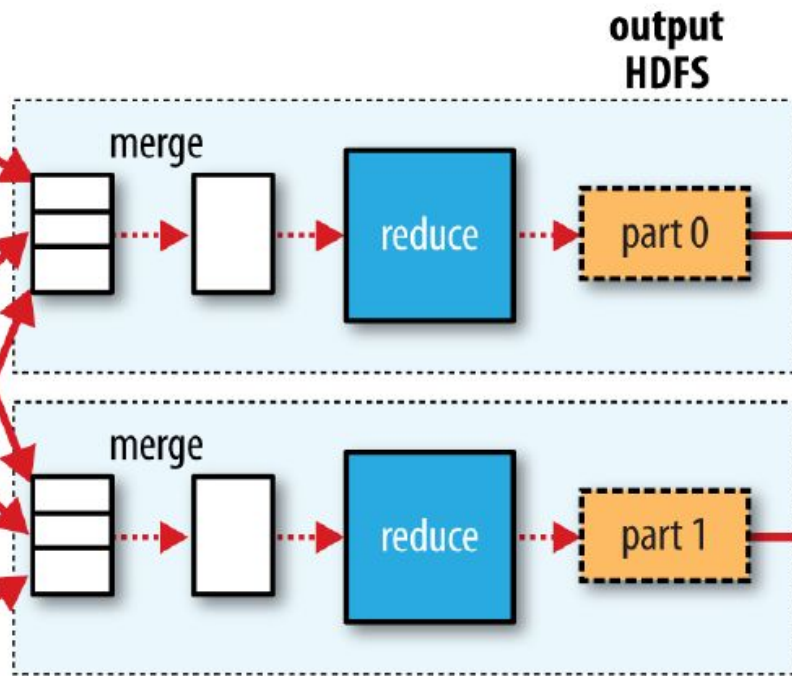
    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

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

**input
HDFS**



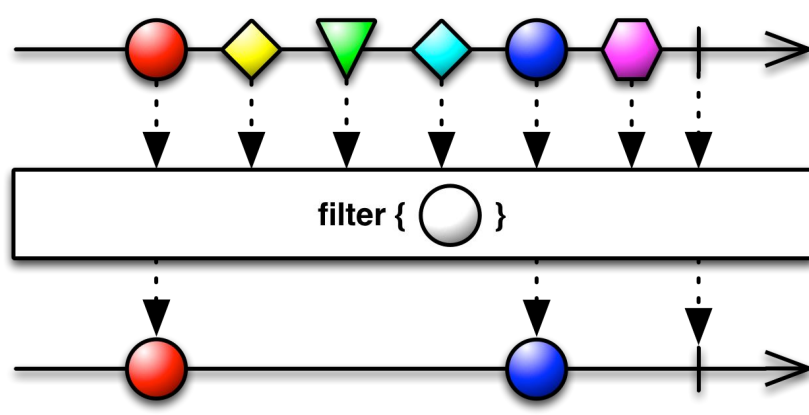
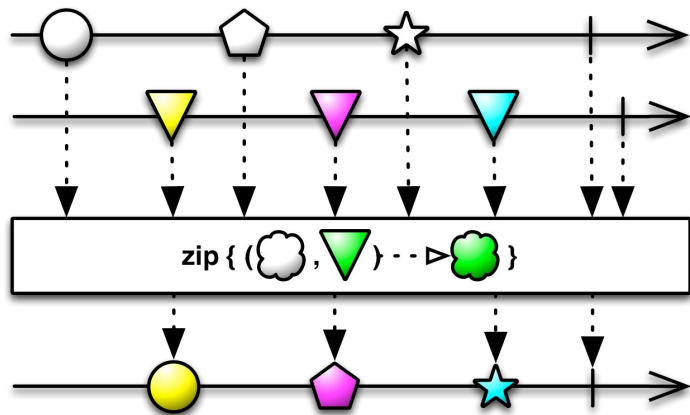
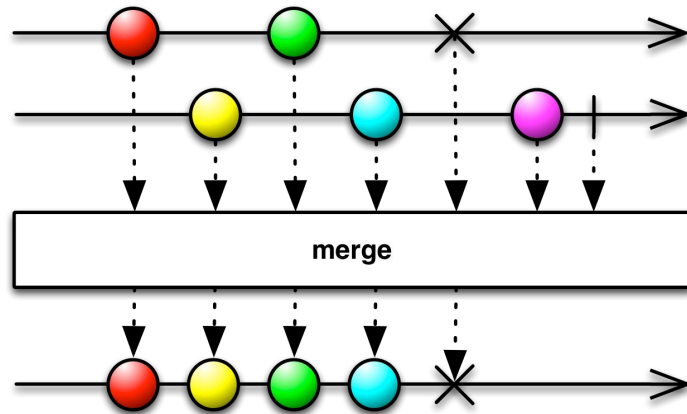
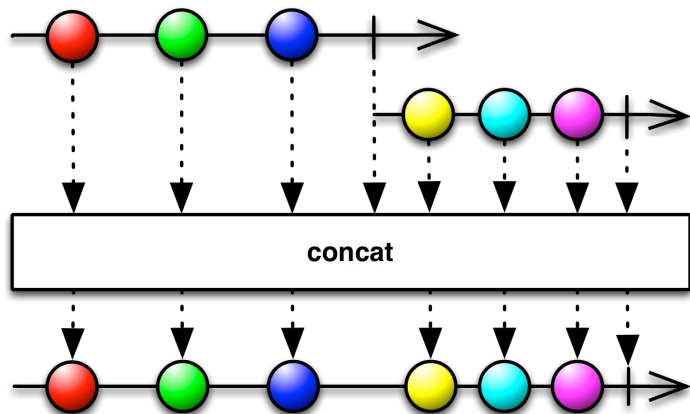
copy



**output
HDFS**

**HDFS
replication**

**HDFS
replication**

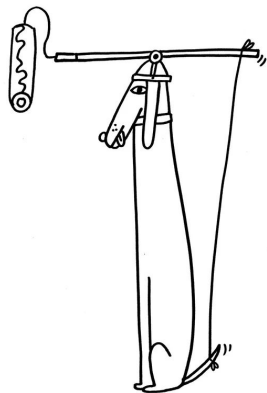




akka



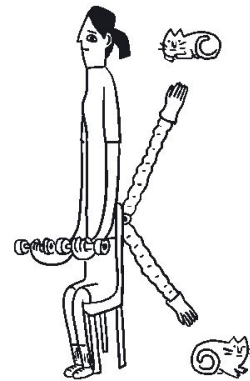
HOW TO BRUSH
YOUR TEETH



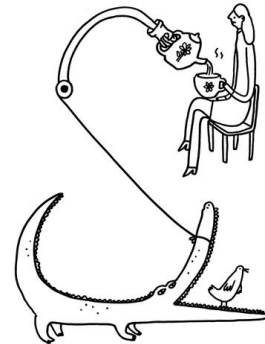
HOW TO UTILIZE YOUR
DOG'S MINDLESS JOY

GPar:

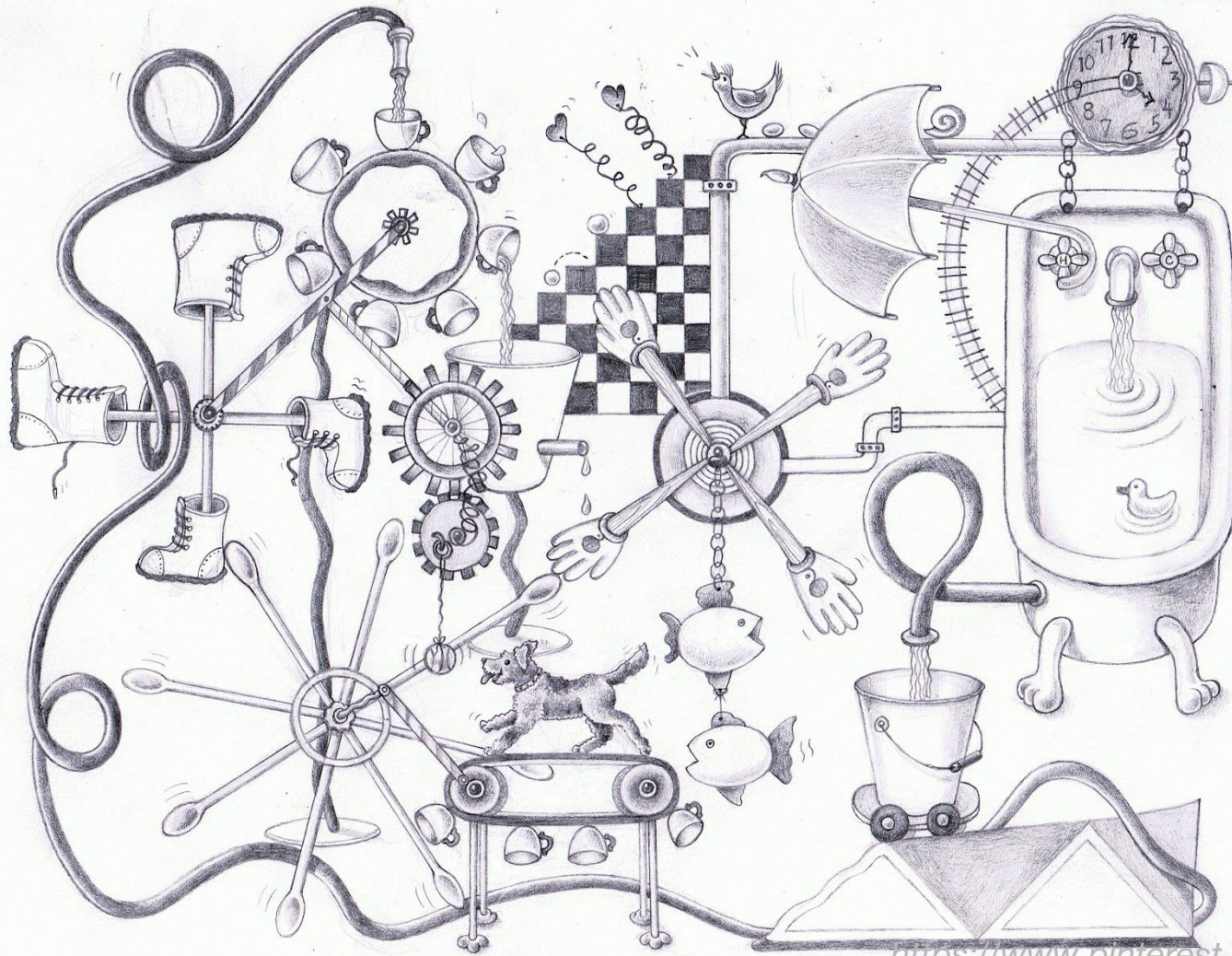
- data parallelism**
- map/reduce**
- fork/join**
- asynchronous closures**
- agents**
- actors**
- dataflows**



HOW TO PET YOUR PETS
WHILE DOING YOUR REPS

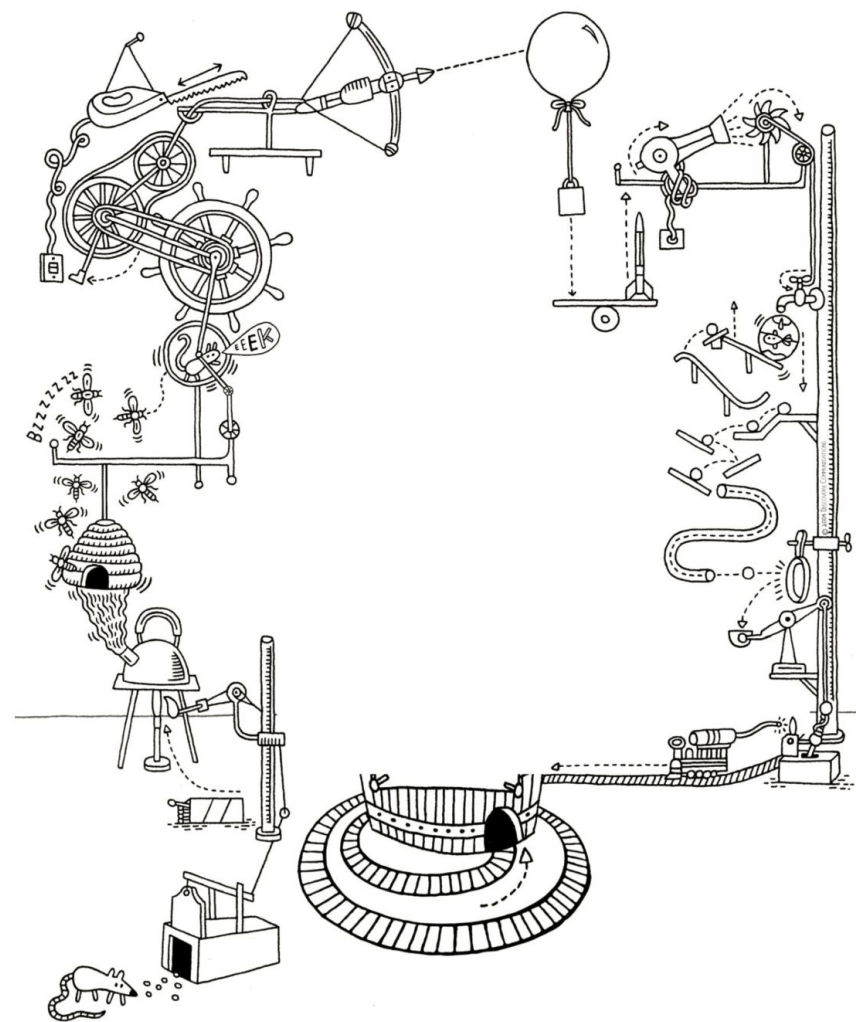


HOW TO SERVE TEA



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()

```

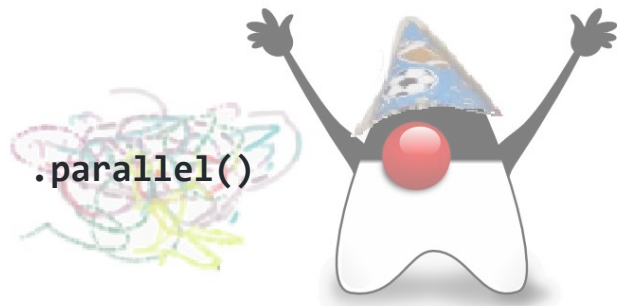


```

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,
        favoured: tweet.favoriteCount,
        retweeted: tweet.retweetCount
      ]
    }).max({
      it.favoured + it.retweeted
    })
  }
}

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

```