Understanding and Overcoming Parallelism Bottlenecks in ForkJoin Applications

@gustavopinto

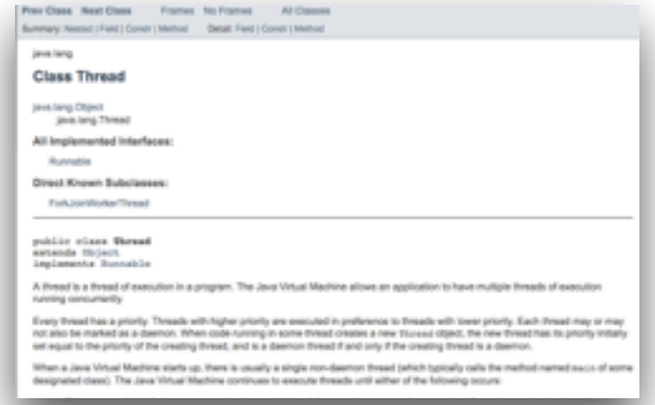
A. Canino

F. Castor

G. Xu

Y. D. Liu

Modern Java applications run on parallel architectures



Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.lang

Class Thread

java.lang.Object
  java.lang.Thread

All Implemented Interfaces:

Runnable

Direct Known Subclasses:

ForkJoinWorkerThread

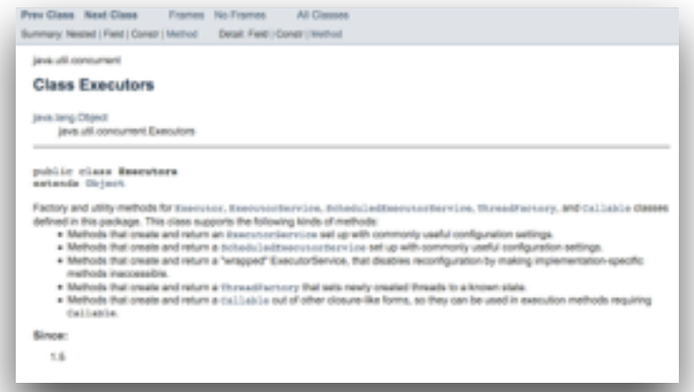
```
public class Thread
extends Object
implements Runnable
```

A thread is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. Each thread may or may not also be marked as a daemon. When code running in some thread creates a new `Thread` object, the new thread has its priority initially set equal to the priority of the creating thread, and is a daemon thread if and only if the creating thread is a daemon.

When a Java Virtual Machine starts up, there is usually a single non-daemon thread (which typically calls the method named `main` of some designated class). The Java Virtual Machine continues to execute threads until either of the following occurs:

java.lang.Thread



Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util.concurrent

Class Executors

java.lang.Object
  java.util.concurrent.Executors

```
public class Executors
extends Object
```

Factory and utility methods for `Executor`, `ExecutorService`, `ScheduledExecutorService`, `ThreadFactory`, and `Callable` classes defined in this package. This class supports the following kinds of methods:

- Methods that create and return an `ExecutorService` set up with commonly useful configuration settings.
- Methods that create and return a `ScheduledExecutorService` set up with commonly useful configuration settings.
- Methods that create and return a "wrapped" ExecutorService, that disables reconfiguration by making implementation-specific methods inaccessible.
- Methods that create and return a `ThreadFactory` that sets newly created threads to a known state.
- Methods that create and return a `Callable` out of other closure-like forms, so they can be used in execution methods requiring `Callable`.

Since:

1.5

java.util.concurrent.Executors

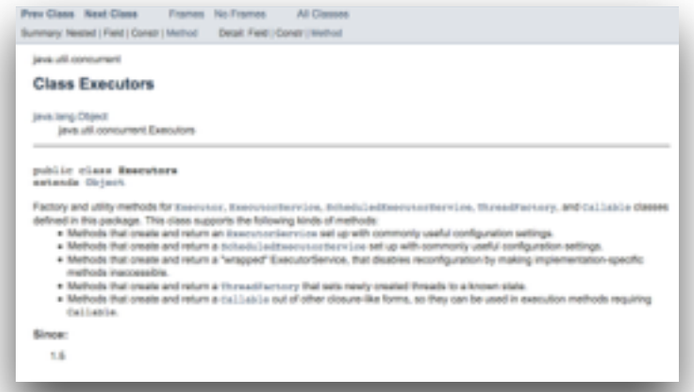
- Widely used
- Low level API
- Error prone

- Well used
- High Level API
- User friendly

Modern Java applications run on parallel architectures

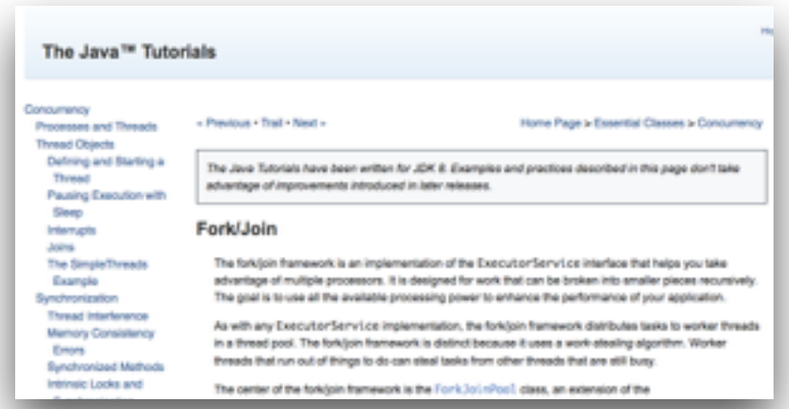
java.lang.Thread

- Widely used
- Low level API
- Error prone



java.util.concurrent.Executors

- Well used
- High Level API
- User friendly



ForkJoin

- Can be more used
- Sophisticated API
- Sophisticated scheduler

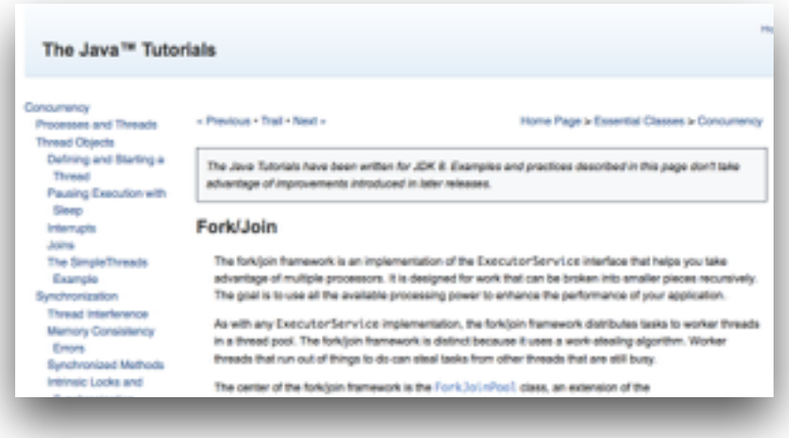
Modern Java applications run on parallel architectures

java.lang.Thread

- Widely used
- Low level API
- Error prone

java.util.concurrent.Executors

- Well used
- High Level API
- User friendly



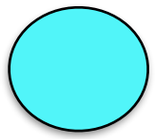
ForkJoin

- Can be more used
- Sophisticated API
- Sophisticated scheduler

Why ForkJoin?

@gustavopinto

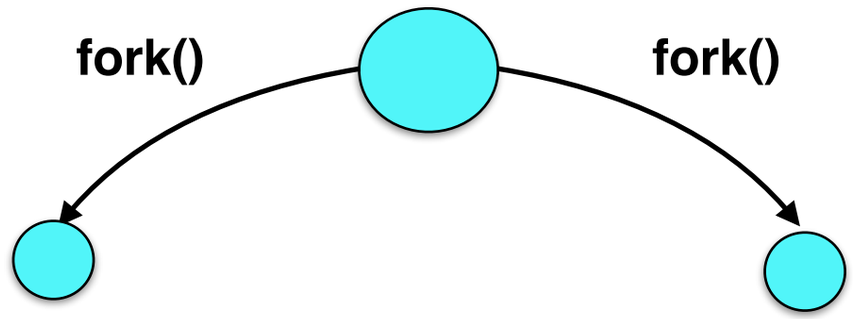
Divider and conquer algorithm



Why ForkJoin?

@gustavopinto

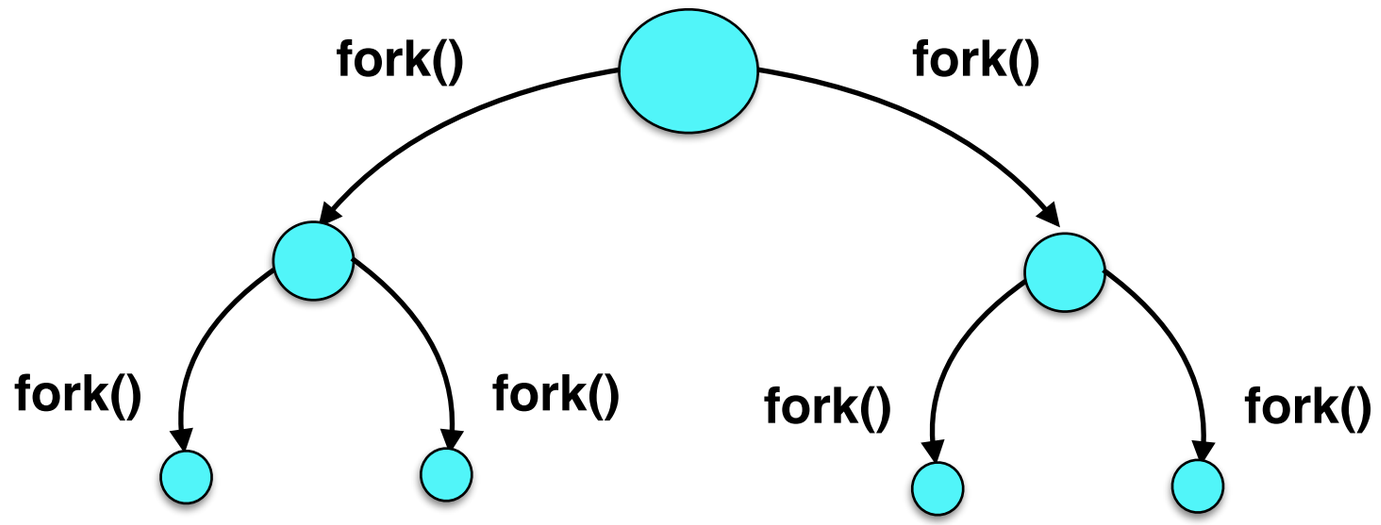
Divider and conquer algorithm



Why ForkJoin?

@gustavopinto

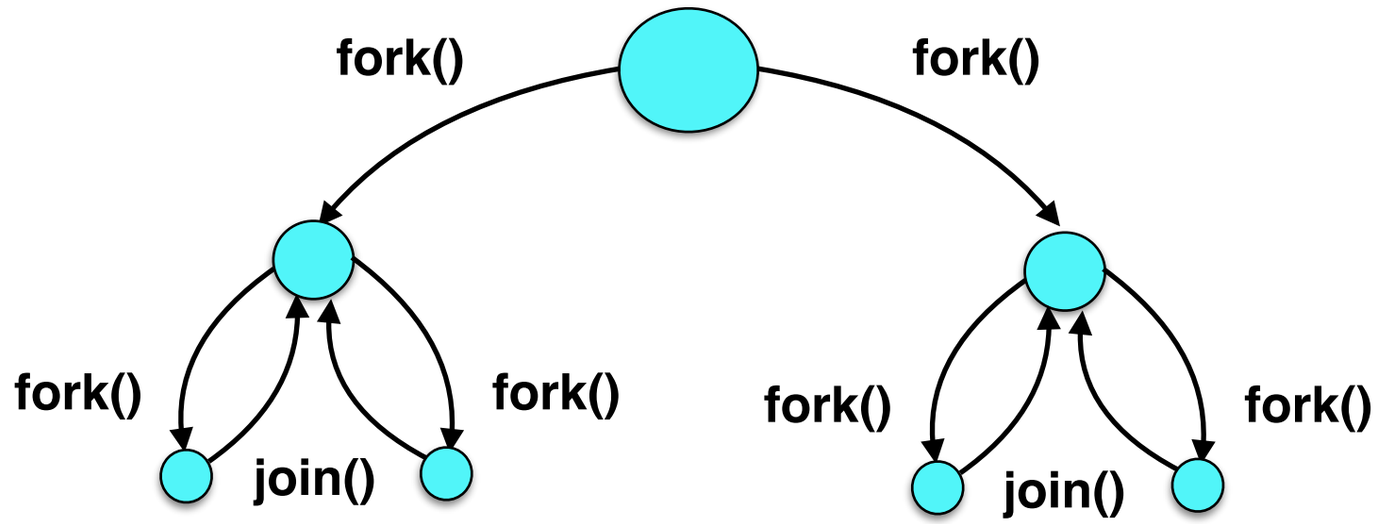
Divider and conquer algorithm



Why ForkJoin?

@gustavopinto

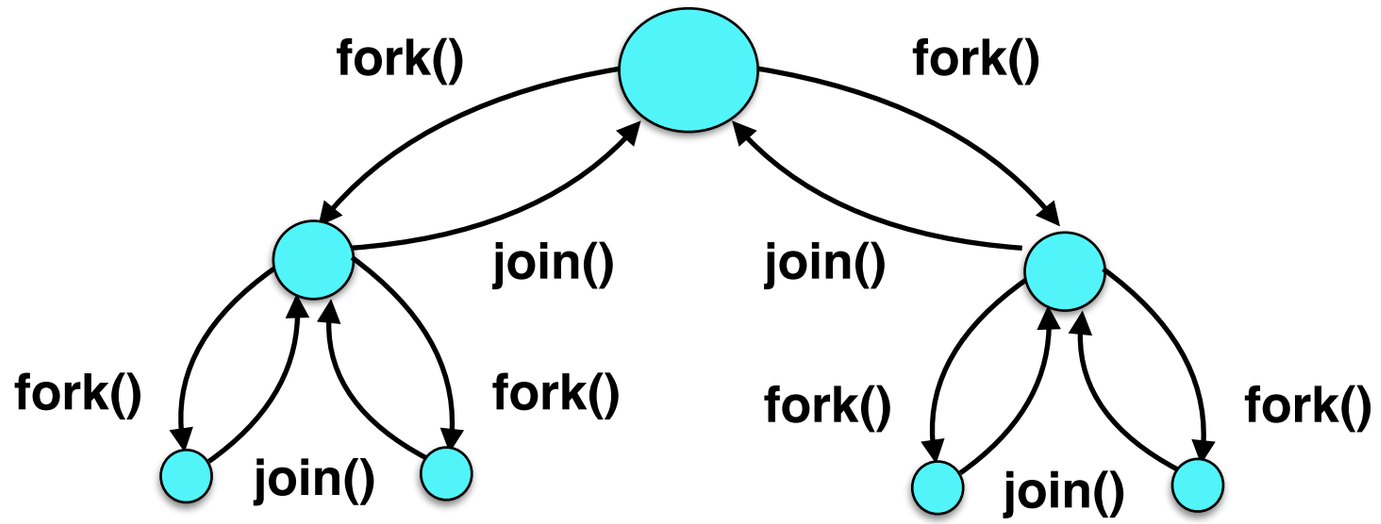
Divider and conquer algorithm



Why ForkJoin?

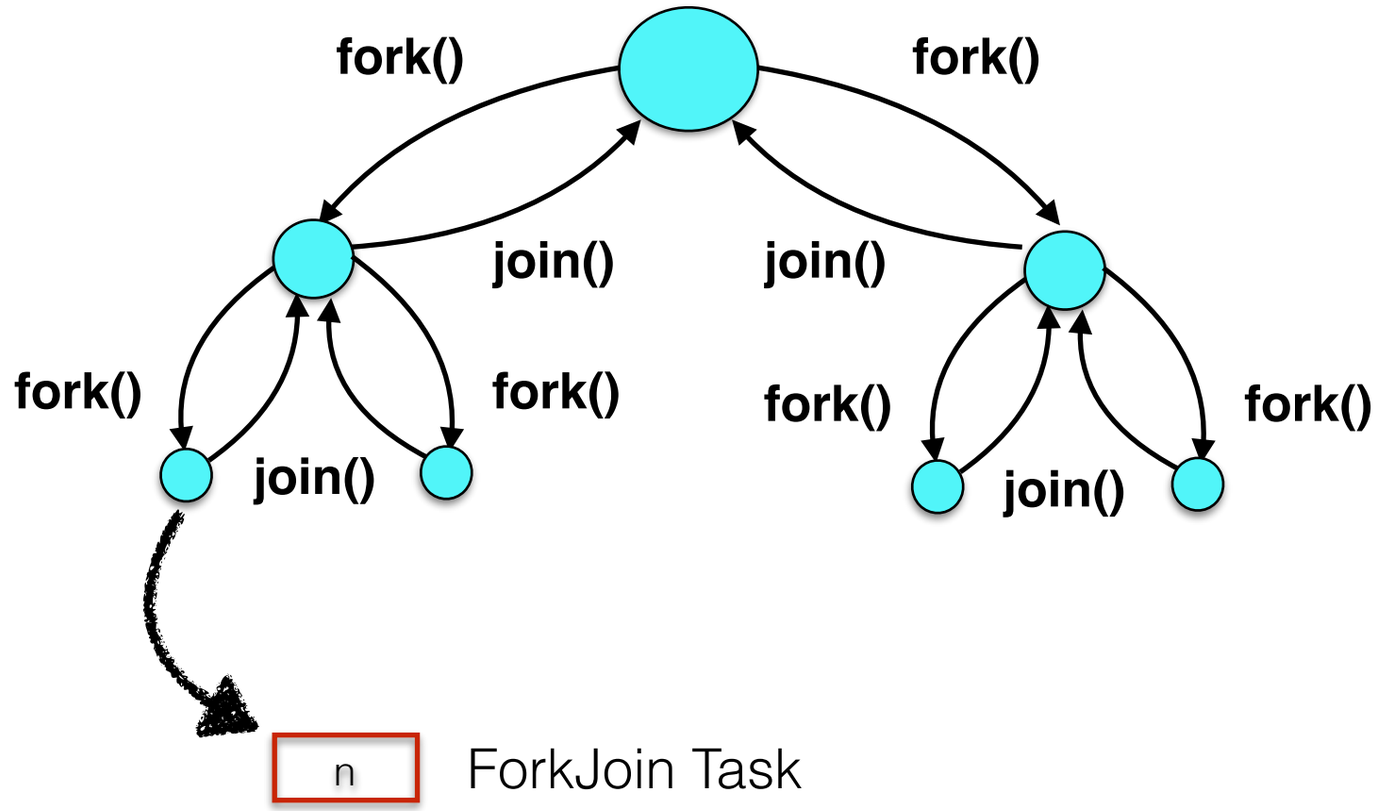
@gustavopinto

Divider and conquer algorithm



Why ForkJoin?

Divider and conquer algorithm

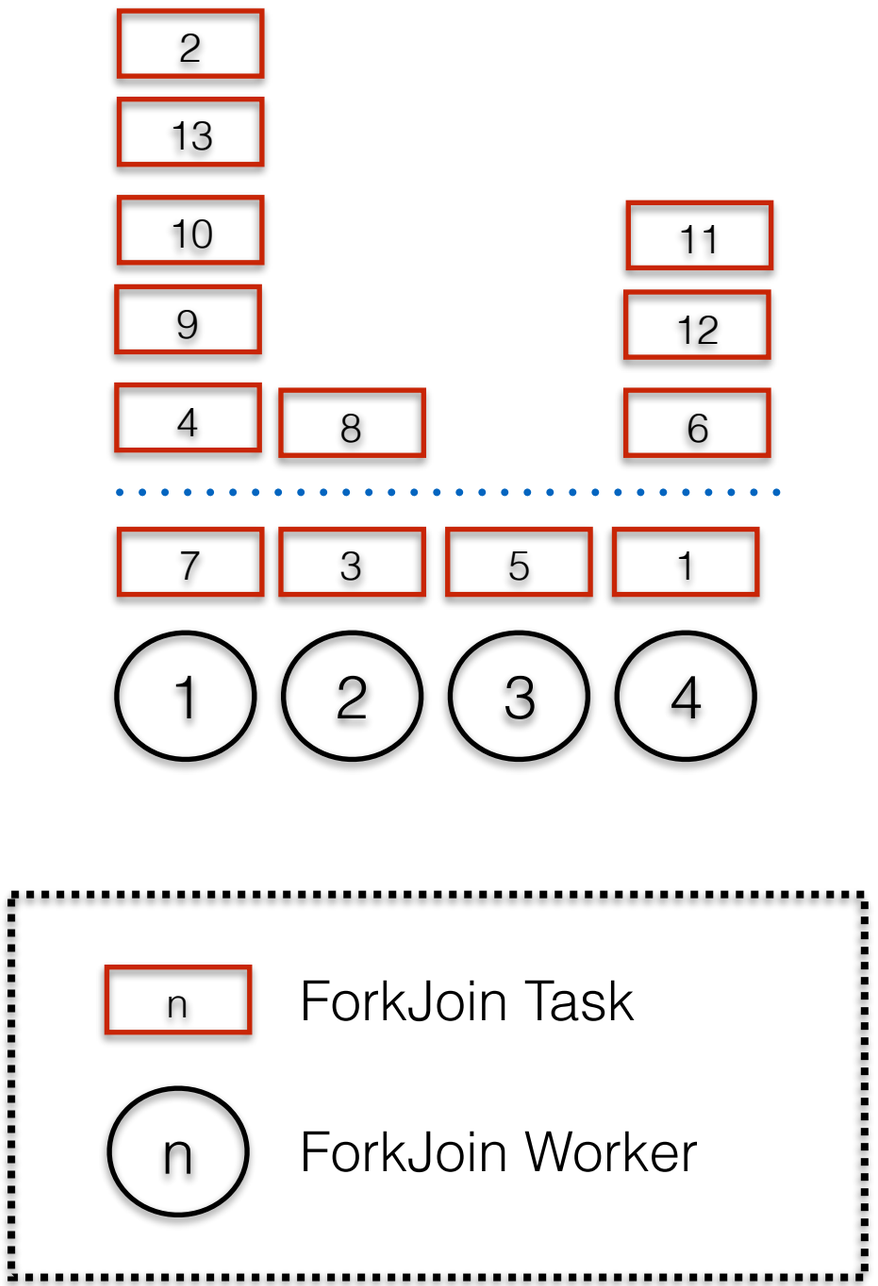


@gustavopinto

Why ForkJoin?

@gustavopinto

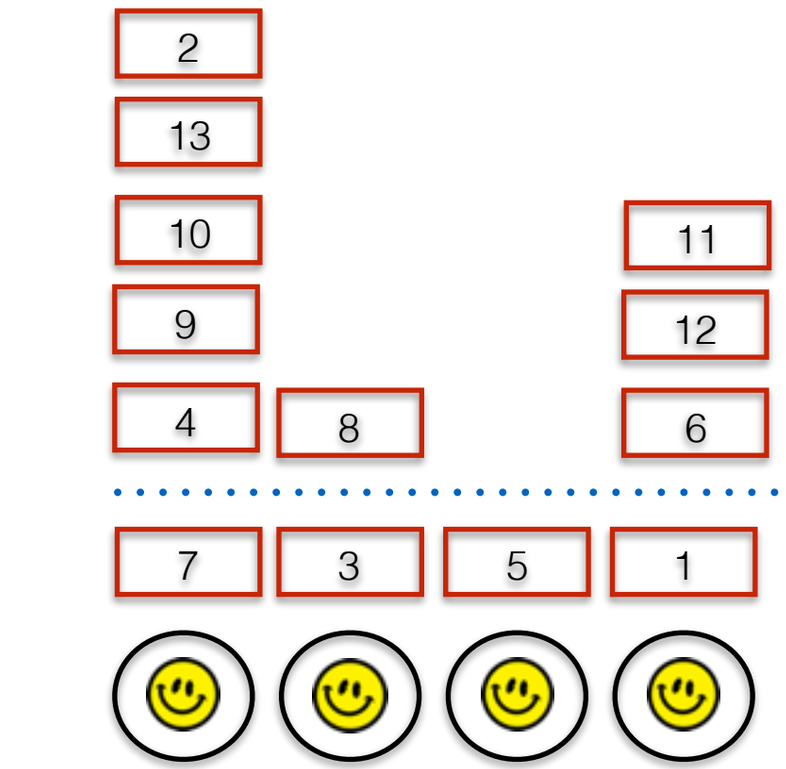
Work Stealing



Why ForkJoin?

@gustavopinto

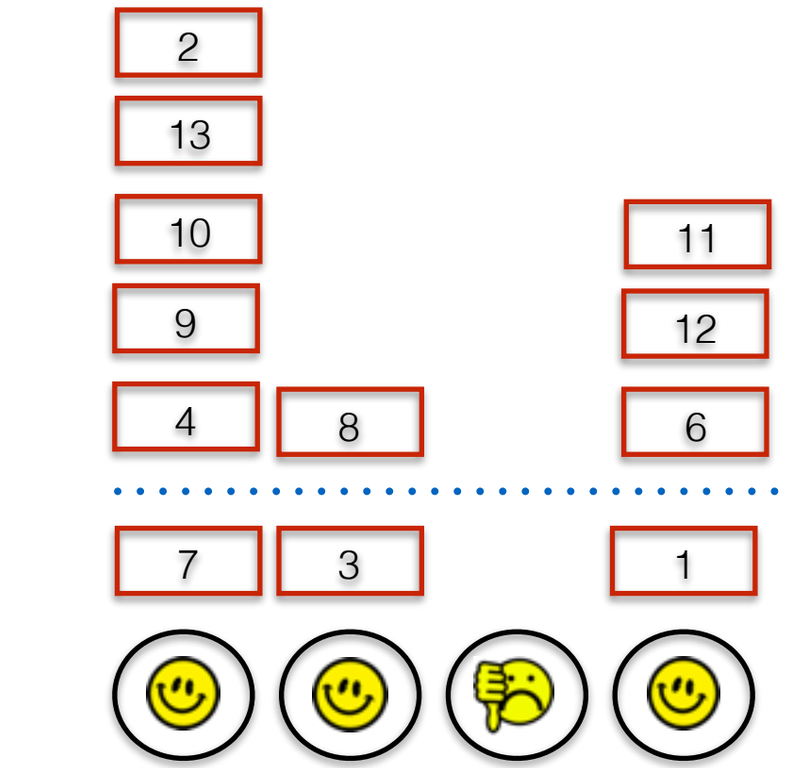
Work Stealing



Why ForkJoin?

@gustavopinto

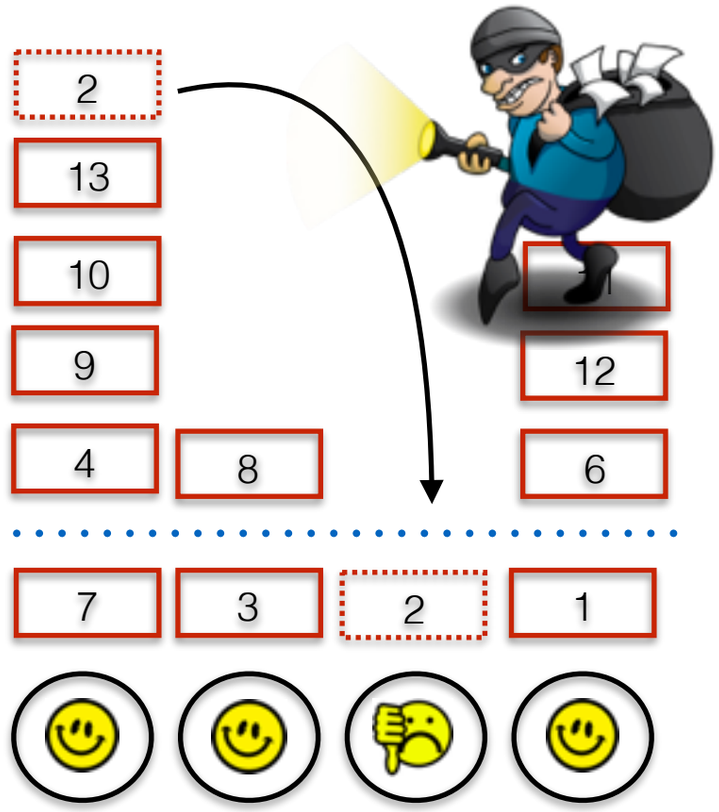
Work Stealing



Why ForkJoin?

@gustavopinto

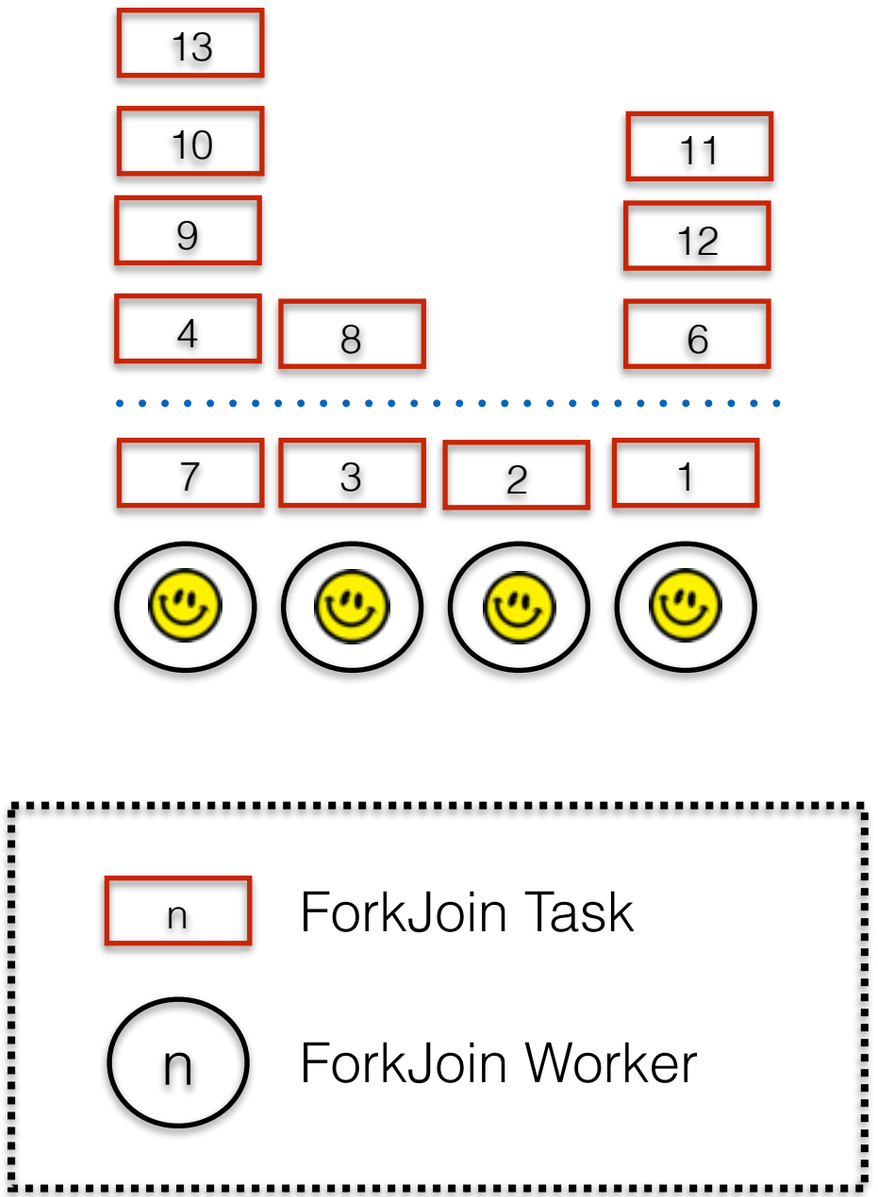
Work Stealing



Why ForkJoin?

@gustavopinto

Work Stealing



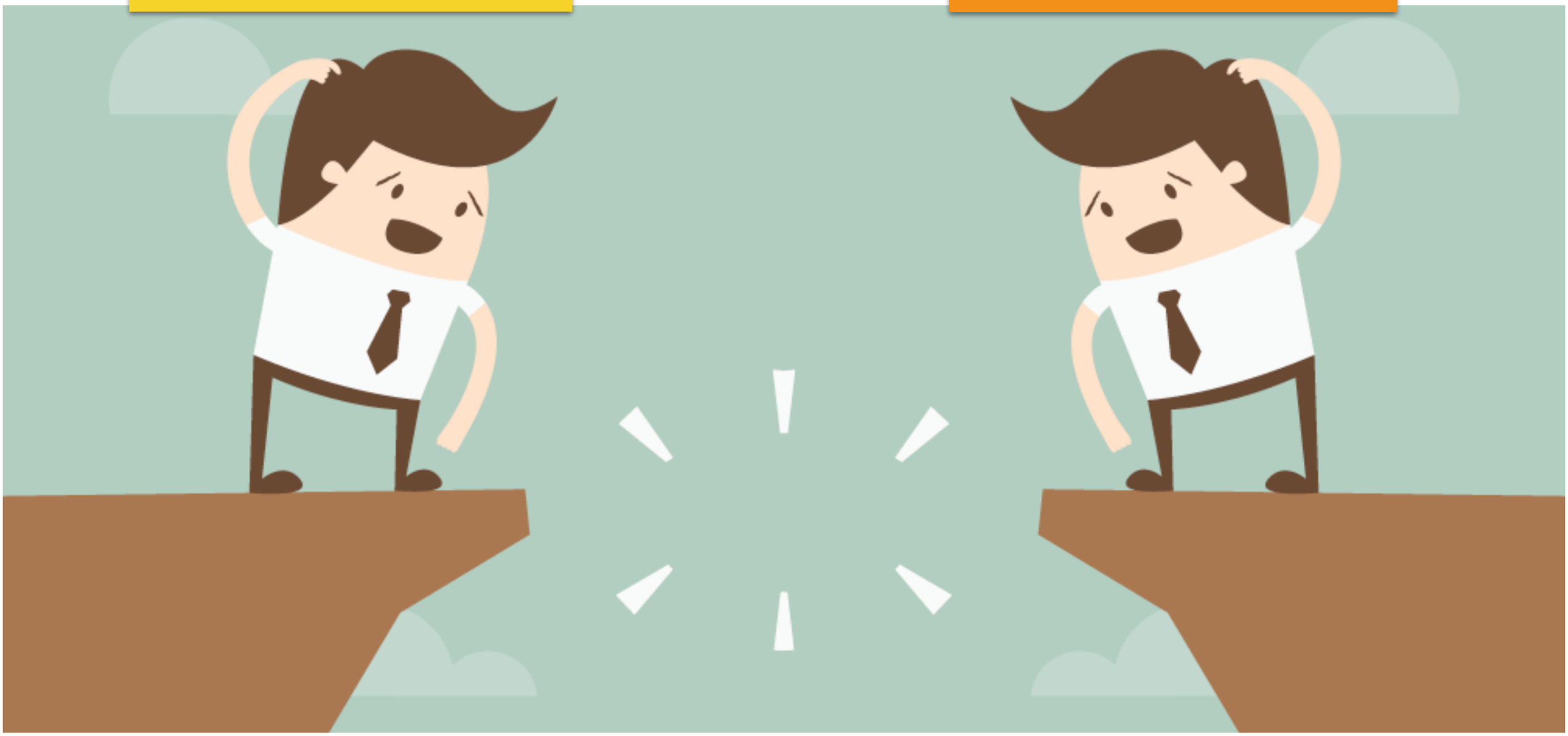
Why ForkJoin?

@gustavopinto

Bedrock for higher-level Java concurrent libraries

System Programmers

Application Programmers



Application Programmers

ForkJoin Applications

What are the ForkJoin applications?

1. Search for java.util.concurrent.ForkJoinPool
2. Investigate if ForkJoin is indeed used
3. Filter out class assignments and pet projects
4. Try to build and run the code

@gustavopinto

Application Programmers

ForkJoin Applications

@gustavopinto

What are the ForkJoin applications?

1. Search for java.util.concurrent.ForkJoinPool
2. Investigate if ForkJoin is indeed used
3. Filter out class assignments and pet projects
4. Try to build and run the code

30 projects selected

e.g., ~ 380KLoC

ecco	ejisto	mandelbrot	knn
jacer	conflate	cq4j	mywiki
exhibitor	warp	lowlatency	…

System Programmers

ForkJoin Applications

@gustavopinto

What about that higher level libs?

1. 330K lines of code
2. 21k commits
3. 470 source code contributors
4. Written (mostly) in Scala and Java
5. Well-known and well-used

This paper

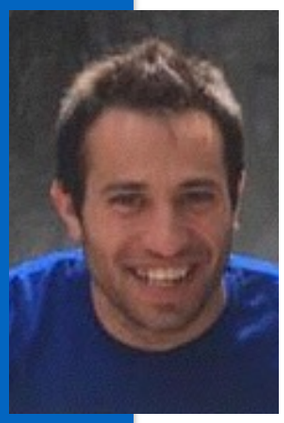
A depth-oriented study and restructuring of the akka message passing algorithm

A breadth-oriented study of 30 real-world ForkJoin open-source projects

A refactoring tool (the first aimed at improving energy consumption of parallel systems)

@gustavopinto

Understanding Parallelism Bottlenecks



Tree: f7da13b878 ▾ FastenTest / src / core / forkjoin / ForkJoinPerformer.java Find file Copy path

NesterovAndrey commit f7da13b on Jun 13

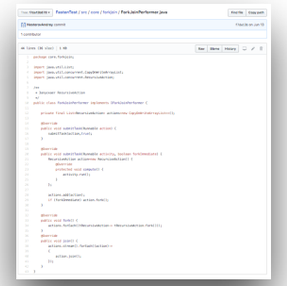
1 contributor

44 lines (36 sloc) | 1 KB Raw Blame History

```java
package core.forkjoin;

import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;
import java.util.concurrent.RecursiveAction;

/**
 * Запускает RecursiveAction
 */
public class ForkJoinPerformer implements IForkJoinPerformer {

    private final List<RecursiveAction> actions=new CopyOnWriteArrayList<>();

    @Override
    public void submitTask(Runnable action) {
        submitTask(action,true);
    }

    @Override
    public void submitTask(Runnable activity, boolean forkImmediate) {
        RecursiveAction action=new RecursiveAction() {
            @Override
            protected void compute() {
                activity.run();
            }
        };

        actions.add(action);
        if (forkImmediate) action.fork();
    }

    @Override
    public void fork() {
        actions.forEach((tRecursiveAction-> tRecursiveAction.fork()));
    }
    @Override
    public void join() {
        actions.stream().forEach((action)->
        {
            action.join();
        });
    }
}
```

@gustavopinto

Understanding Parallelism Bottlenecks

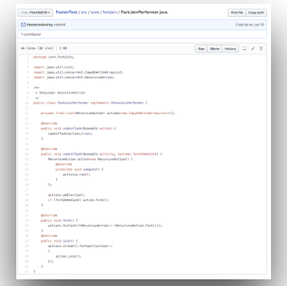


v0

...

@gustavopinto

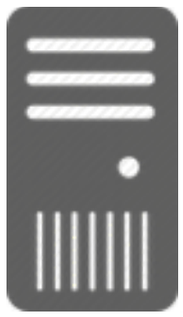
Understanding Parallelism Bottlenecks



v0

...

For each version, we measured execution time and energy consumption



Intel CPU: A 2×8-core (32-cores w/ hyper-threading), running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0_71, build 14.

JRapl: Software-based energy measurement

@gustavopinto

Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #1: Centralized pooling

actors process their own messages

there is no side effect

Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #1: Centralized pooling

actor

mailbox

actors process their own messages

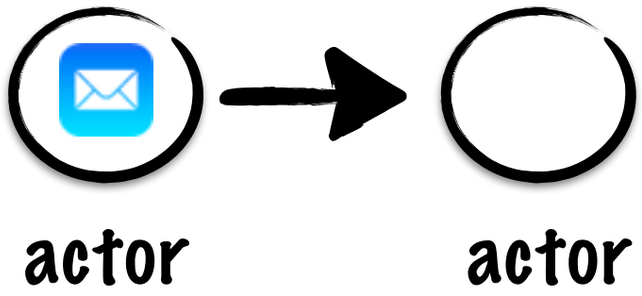
there is no side effect

actors exchange, but do not share the same message

Overcoming Parallelism Bottlenecks

@gustavopinto

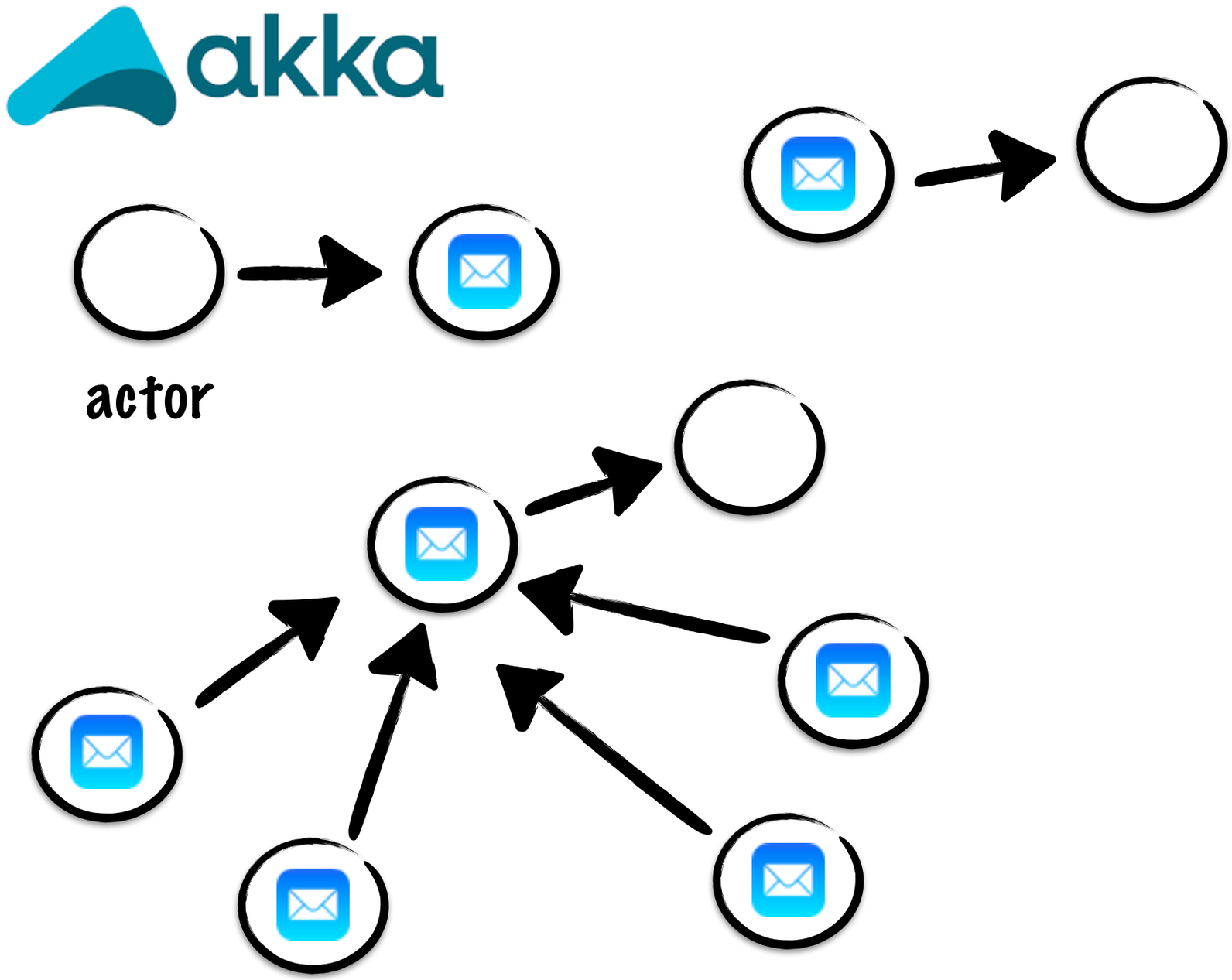
Bottleneck #1: Centralized pooling



Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #1: Centralized pooling



Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #1: Centralized pooling

(Centralized)

actor

actor

Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #1: Centralized pooling

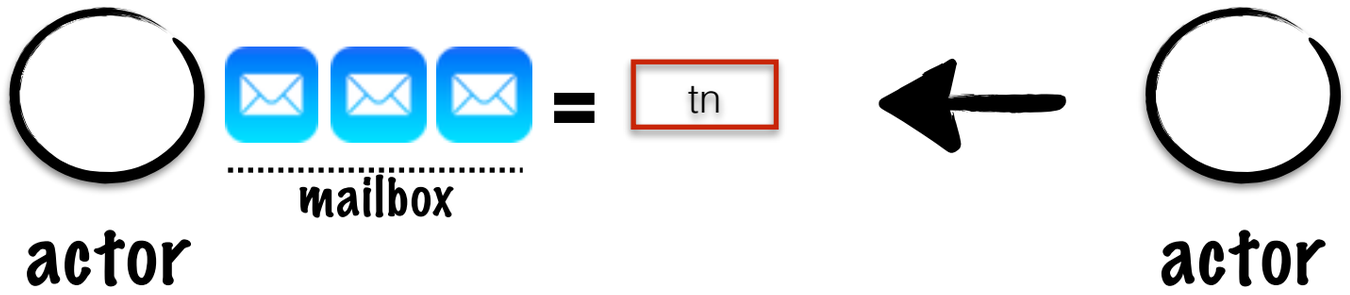
(Centralized)

Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #1: Centralized pooling

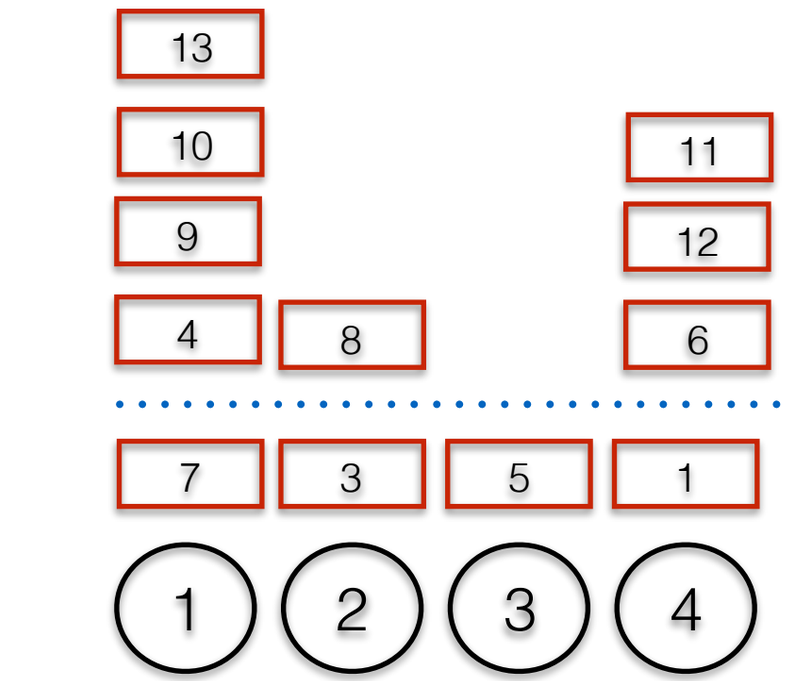
(Centralized)



Overcoming Parallelism Bottlenecks

@gustavopinto

Work Stealing

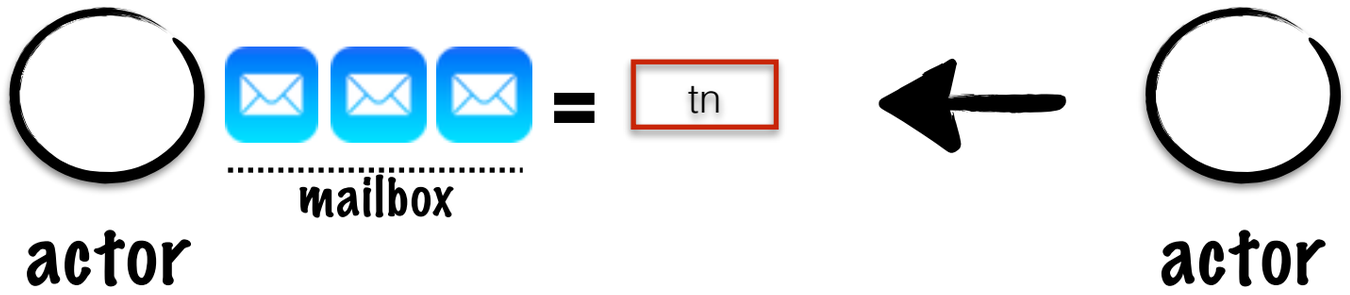


Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #1: Centralized pooling

(Centralized)

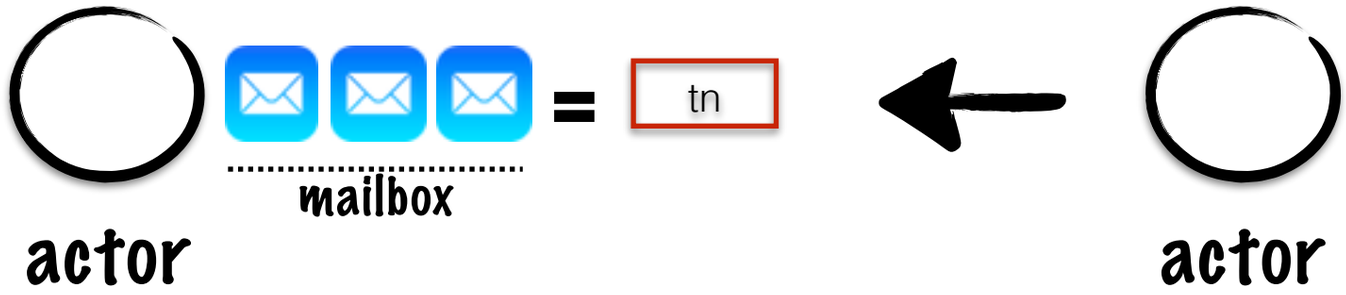


Overcoming Parallelism Bottlenecks

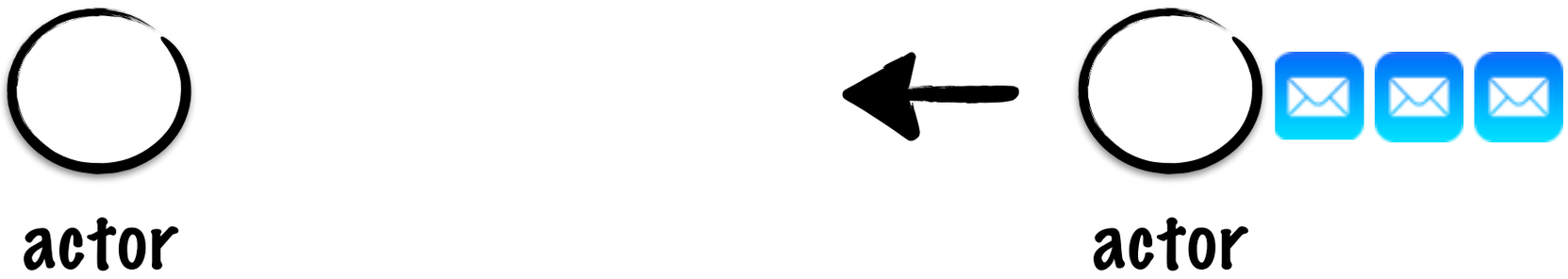
@gustavopinto

Bottleneck #1: Centralized pooling

akka (Centralized)



akka (DEcentralized)

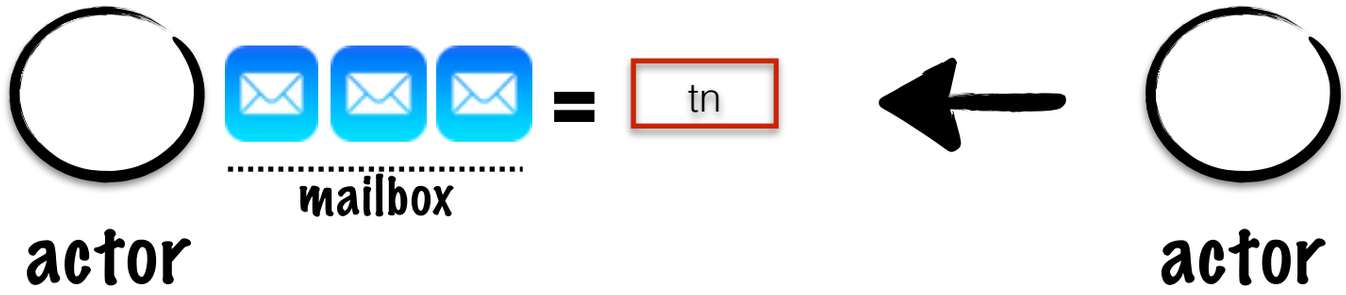


Overcoming Parallelism Bottlenecks

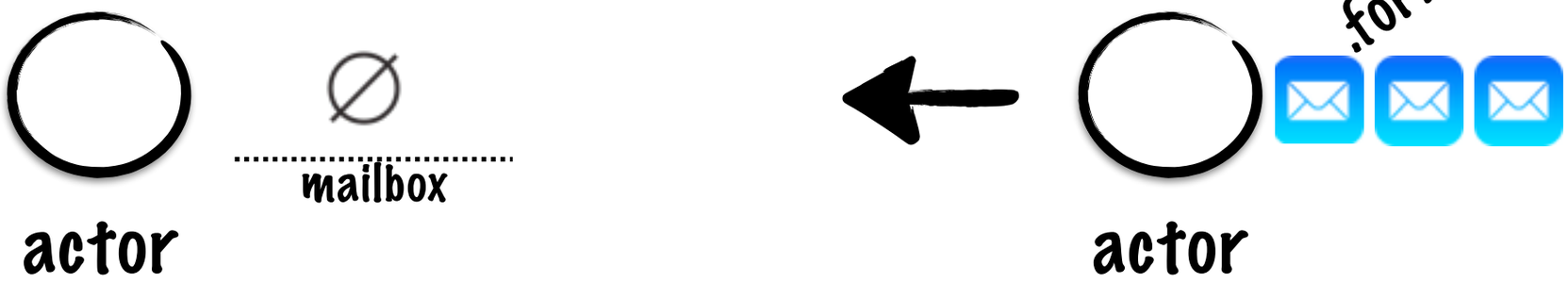
@gustavopinto

Bottleneck #1: Centralized pooling

akka (Centralized)



akka (DEcentralized)

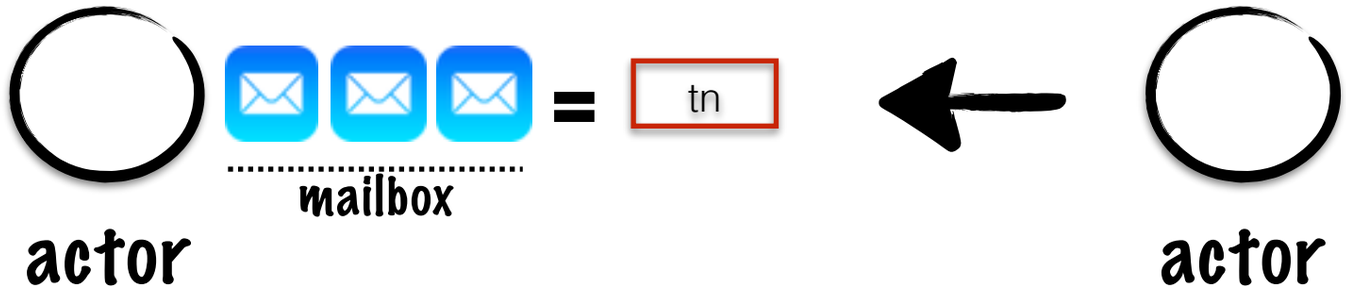


Overcoming Parallelism Bottlenecks

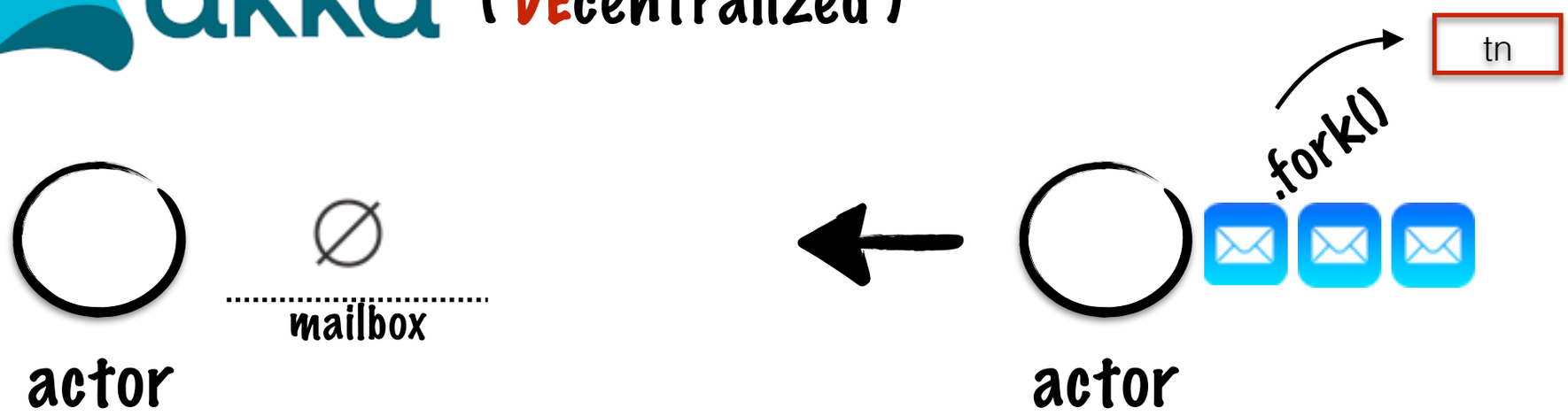
@gustavopinto

Bottleneck #1: Centralized pooling

akka (Centralized)



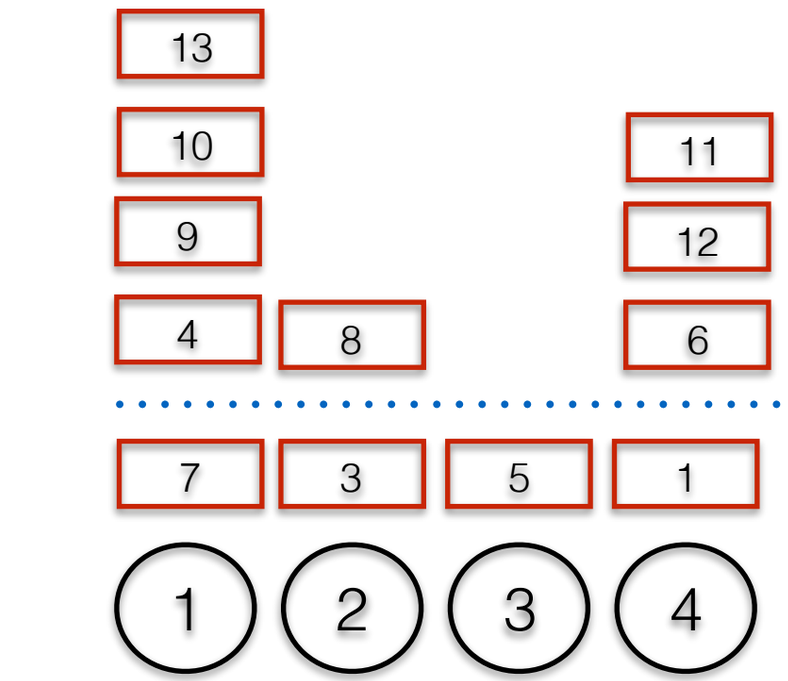
akka (DEcentralized)



Overcoming Parallelism Bottlenecks

@gustavopinto

Work Stealing

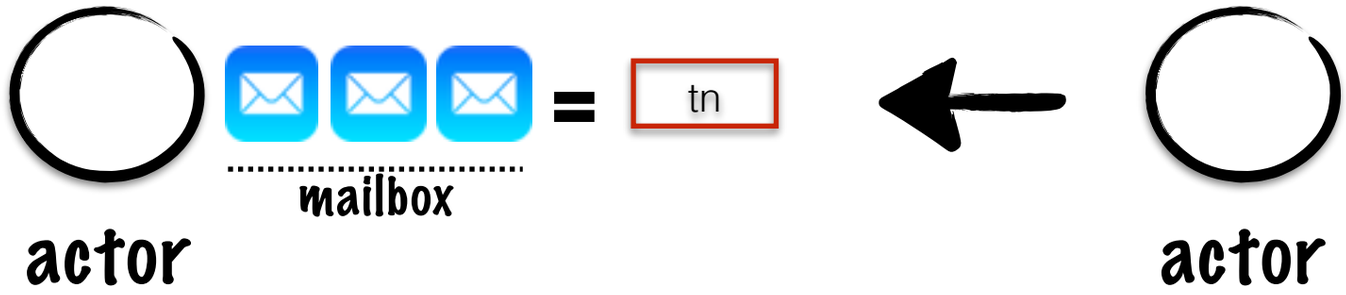


Overcoming Parallelism Bottlenecks

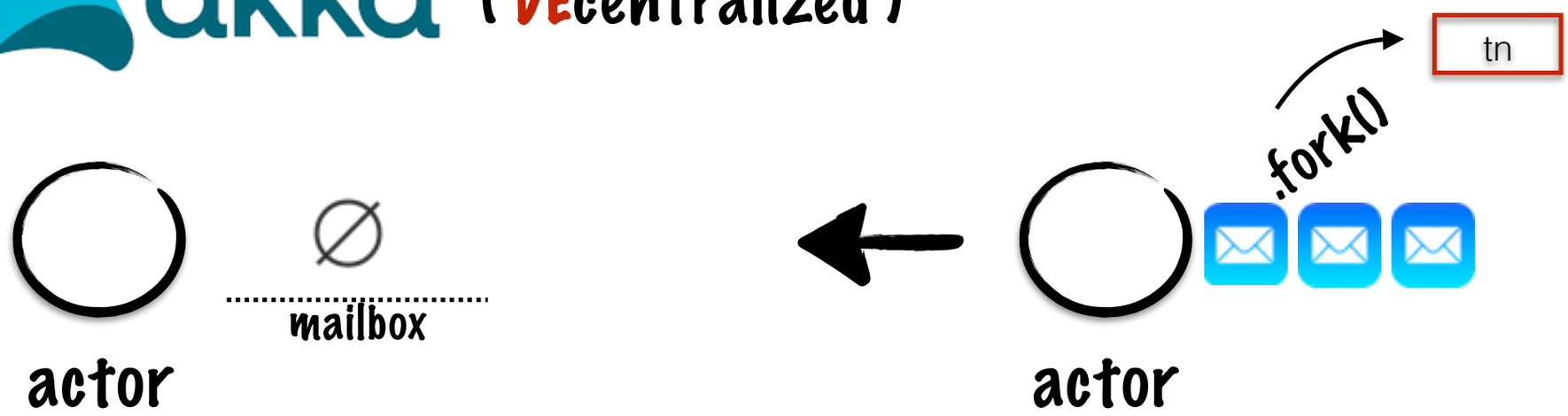
@gustavopinto

Bottleneck #1: Centralized pooling

akka (Centralized)



akka (DEcentralized)



Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #1: Centralized pooling

benchmark	Runtime (ms)				
	original	σ	custom	σ	speedup
max-throughput	1,861.8	433.4	1,833.0	417.8	1.0×
single-ping	11,657.7	643.8	8,979.6	1,815.5	1.3×
ping-throughput	2,314.9	183.0	701.5	84.7	3.3×
single-producer	4,008.3	1,273.8	4,960.3	2,002.8	0.8×
multi-producer	7,120.3	1,143.9	8,219.5	2,327.0	0.8×
middle-man	3,757.3	195.27	1744.1	195.4	2.1×
mediator	4,633.3	241.13	724.3	123.6	6.4×

Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #1: Centralized pooling

akka

benchmark	Runtime (ms)				
	original	σ	custom	σ	speedup
max-throughput	1,861.8	433.4	1,833.0	417.8	1.0×
single-ping	11,657.7	643.8	8,979.6	1,815.5	1.3×
ping-throughput	2,314.9	183.0	701.5	84.7	3.3×
single-producer	4,008.3	1,273.8	4,960.3	2,002.8	0.8×
multi-producer	7,120.3	1,143.9	8,219.5	2,327.0	0.8×
middle-man	3,757.3	195.27	1744.1	195.4	2.1×
mediator	4,633.3	241.13	724.3	123.6	6.4×

Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #1: Centralized pooling

benchmark	Runtime (ms)				
	original	σ	custom	σ	speedup
max-throughput	1,861.8	433.4	1,833.0	417.8	1.0×
single-ping	11,657.7	643.8	8,979.6	1,815.5	
ping-throughput	2,314.9	183.0	701.5	84.7	3.3x
single-producer	4,008.3	1,273.8	4,960.3	2,002.8	
multi-producer	7,120.3	1,143.9	8,219.5	2,327.0	0.8×
middle-man	3,757.3	195.27	1744.1	195.4	
mediator	4,633.3	241.13	724.3	123.6	6.4x

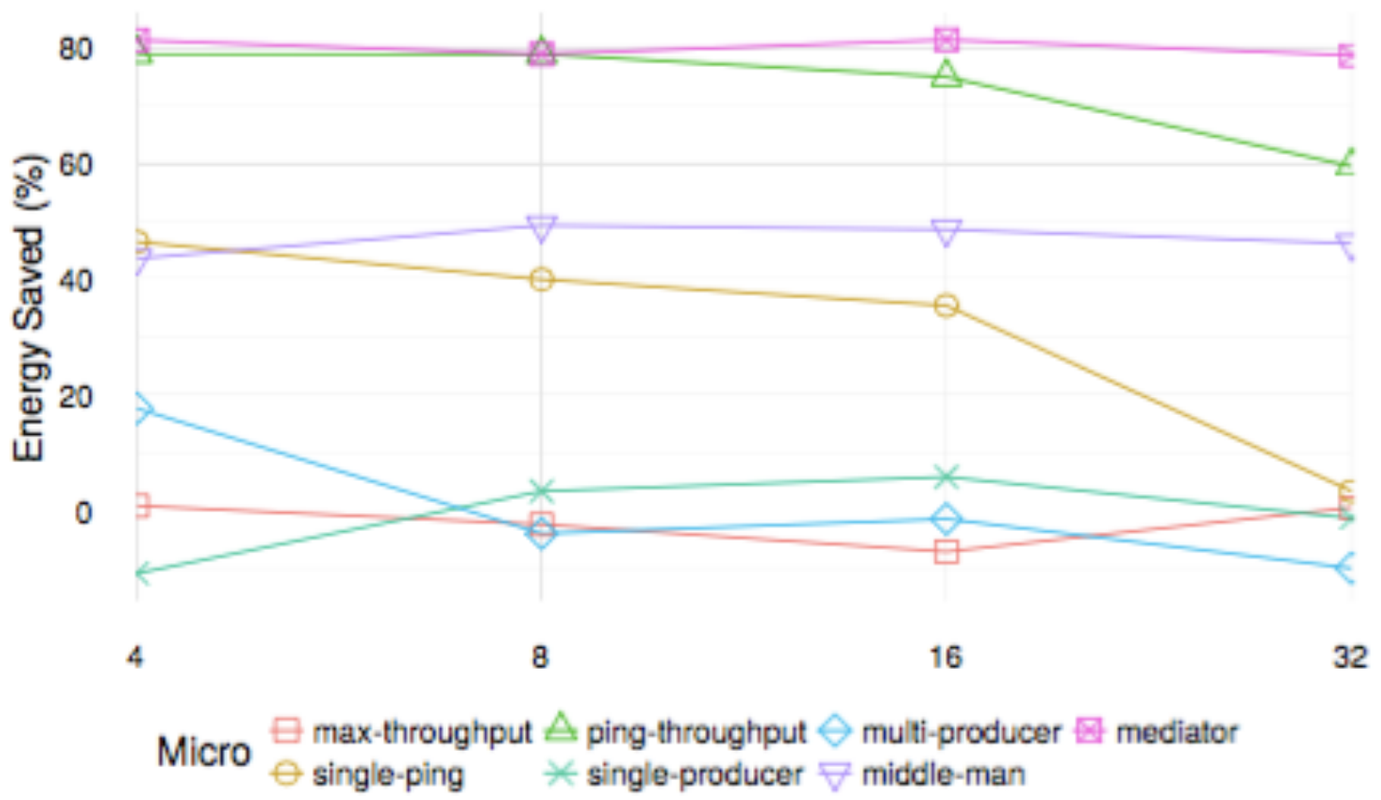
Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #1: Centralized pooling

akka

benchmark	Runtime (ms) original	σ	custom	σ	speedup
max-throughput	1,861.8	433.4	1,833.0	417.8	1.0×
single-ping	11,657.7	643.8	8,979.6	1,815.5	
ping-throughput	2,314.9	183.0	701.5	84.7	3.3x
single-producer	4,008.3	1,273.8	4,960.3	2,002.8	
multi-producer	7,120.3	1,143.9	8,219.5	2,327.0	0.8×
middle-man	3,757.3	195.27	1744.1	195.4	
mediator	4,633.3	241.13	724.3	123.6	6.4x



Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #2: Copy on Fork

t1 = 

t2 = first half

t3 = second half

Overcoming Parallelism Bottlenecks

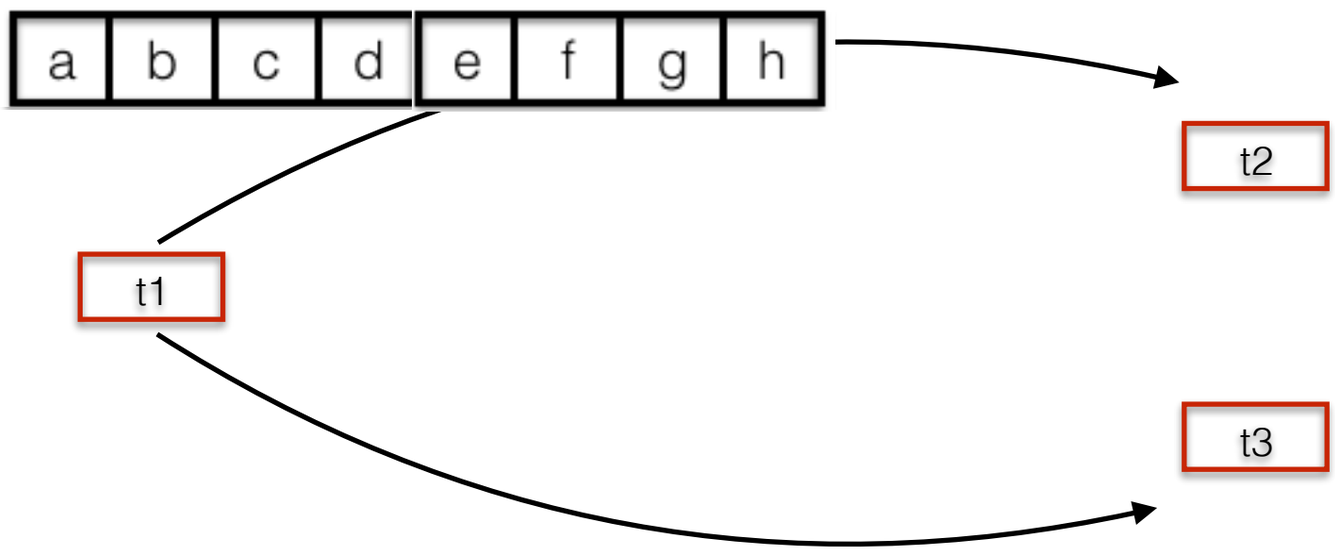
@gustavopinto

Bottleneck #2: Copy on Fork

t1 = 

t2 = first half

t3 = second half



Overcoming Parallelism Bottlenecks

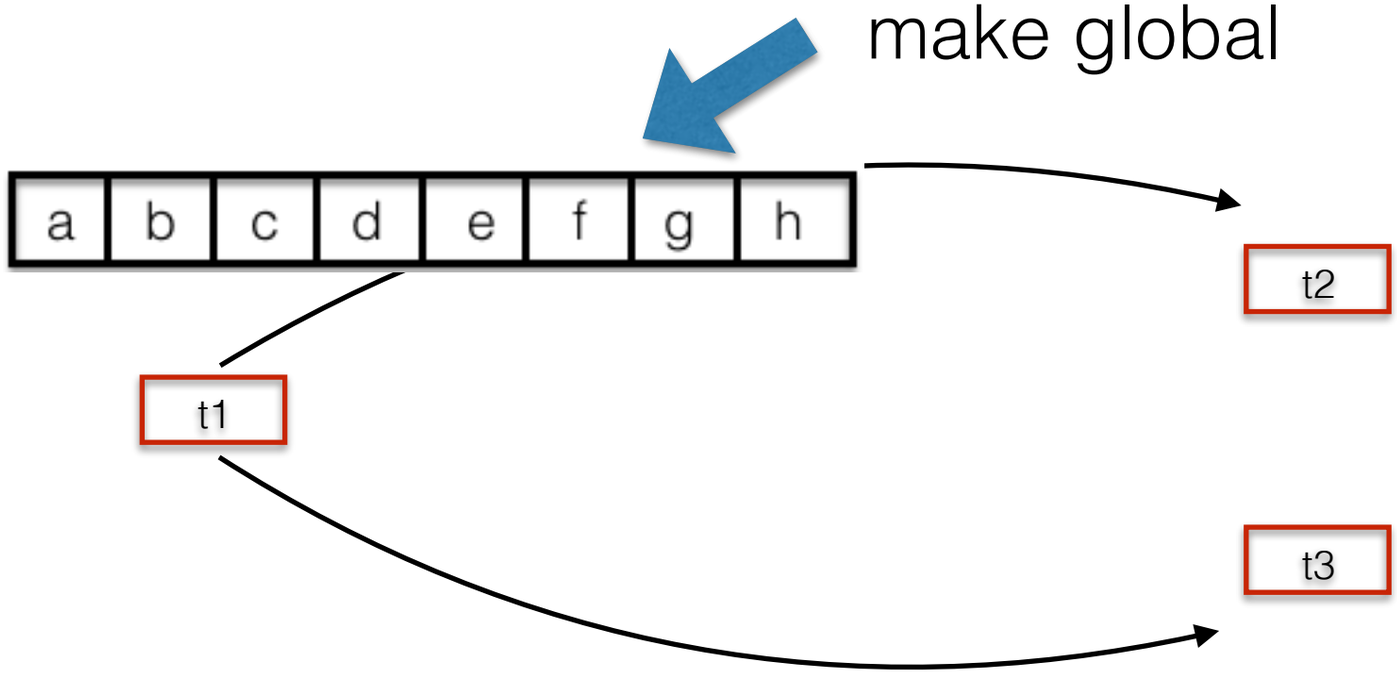
@gustavopinto

Bottleneck #2: Copy on Fork

t1 = 

t2 = first half

t3 = second half



Overcoming Parallelism Bottlenecks

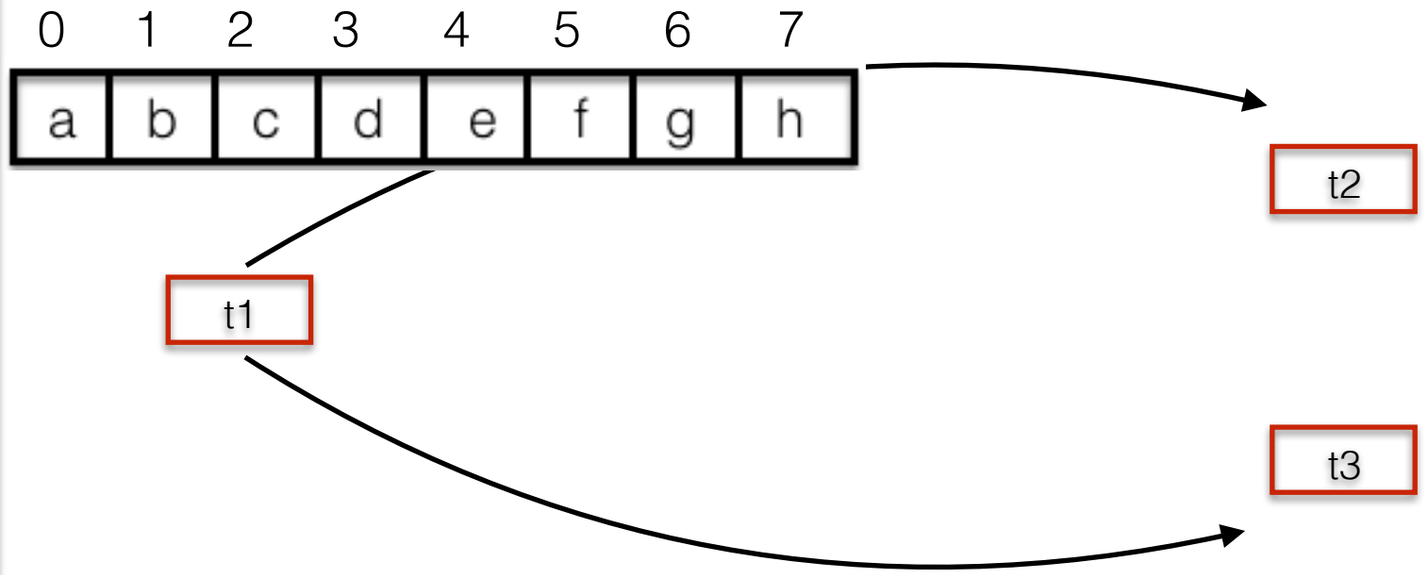
@gustavopinto

Bottleneck #2: Copy on Fork

t1 = | a | b | c | d | e | f | g | h |

t2 = first half

t3 = second half



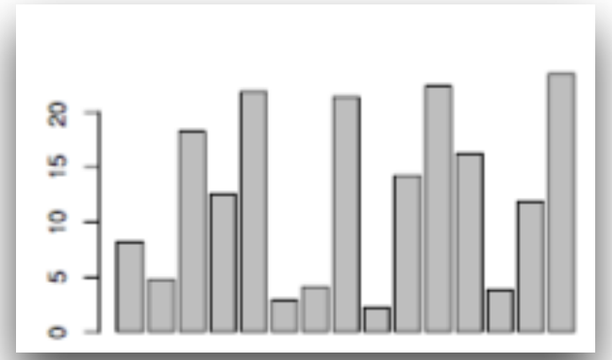
Overcoming Parallelism Bottlenecks

Bottleneck #2: Copy on Fork

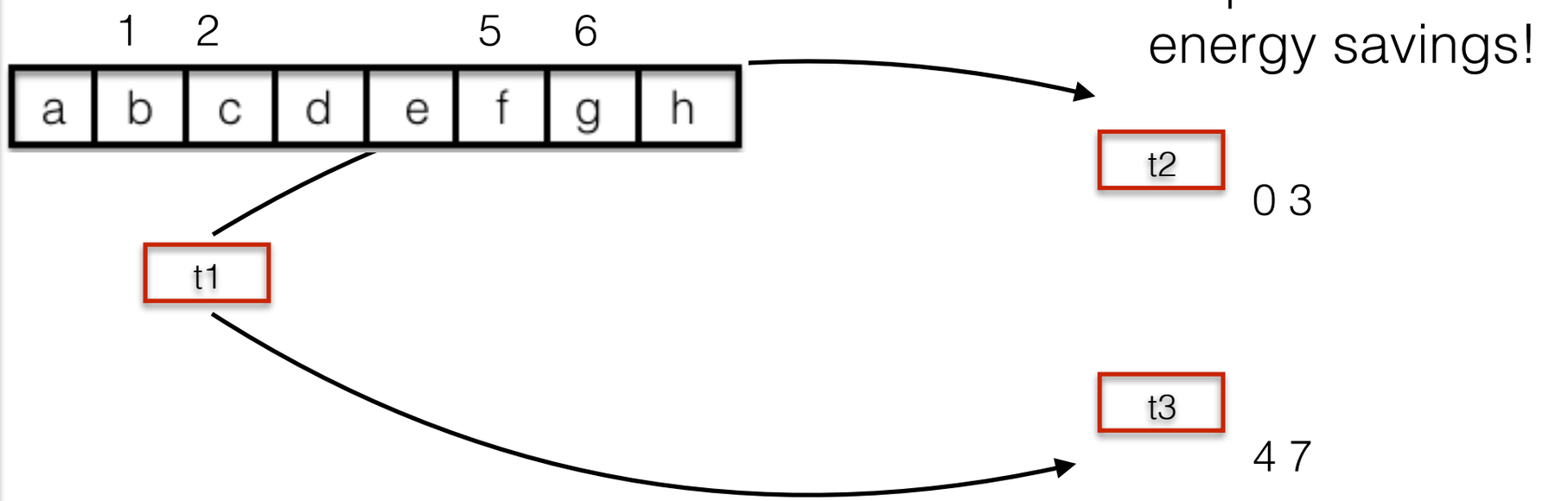
t1 = | a | b | c | d | e | f | g | h |

t2 = first half

t3 = second half



Up to 20% of energy savings!



@gustavopinto

Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #3: Copy on Join

t1 = | a | b | c | d |

t2 = | e | f | g | h |

t3 = t1 + t2

Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #3: Copy on Join

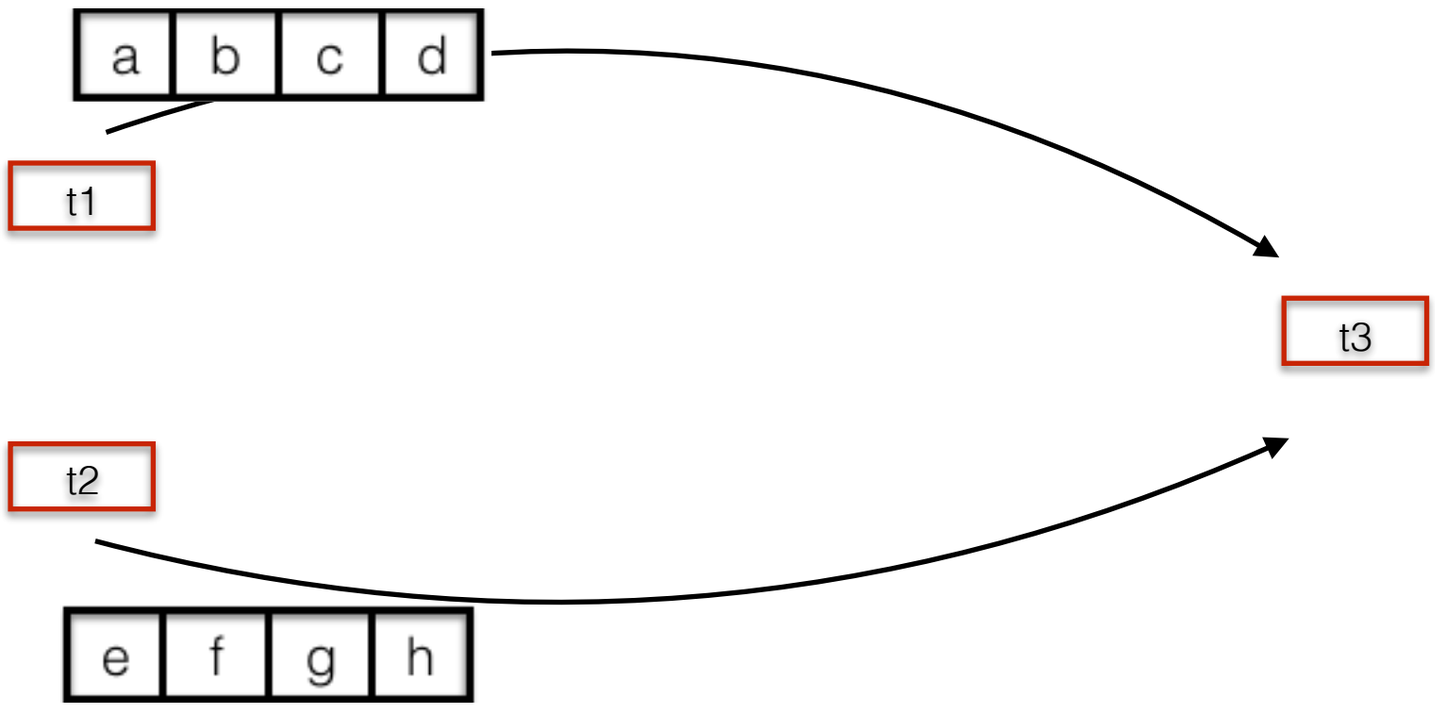
t1 =

a	b	c	d

t2 =

e	f	g	h

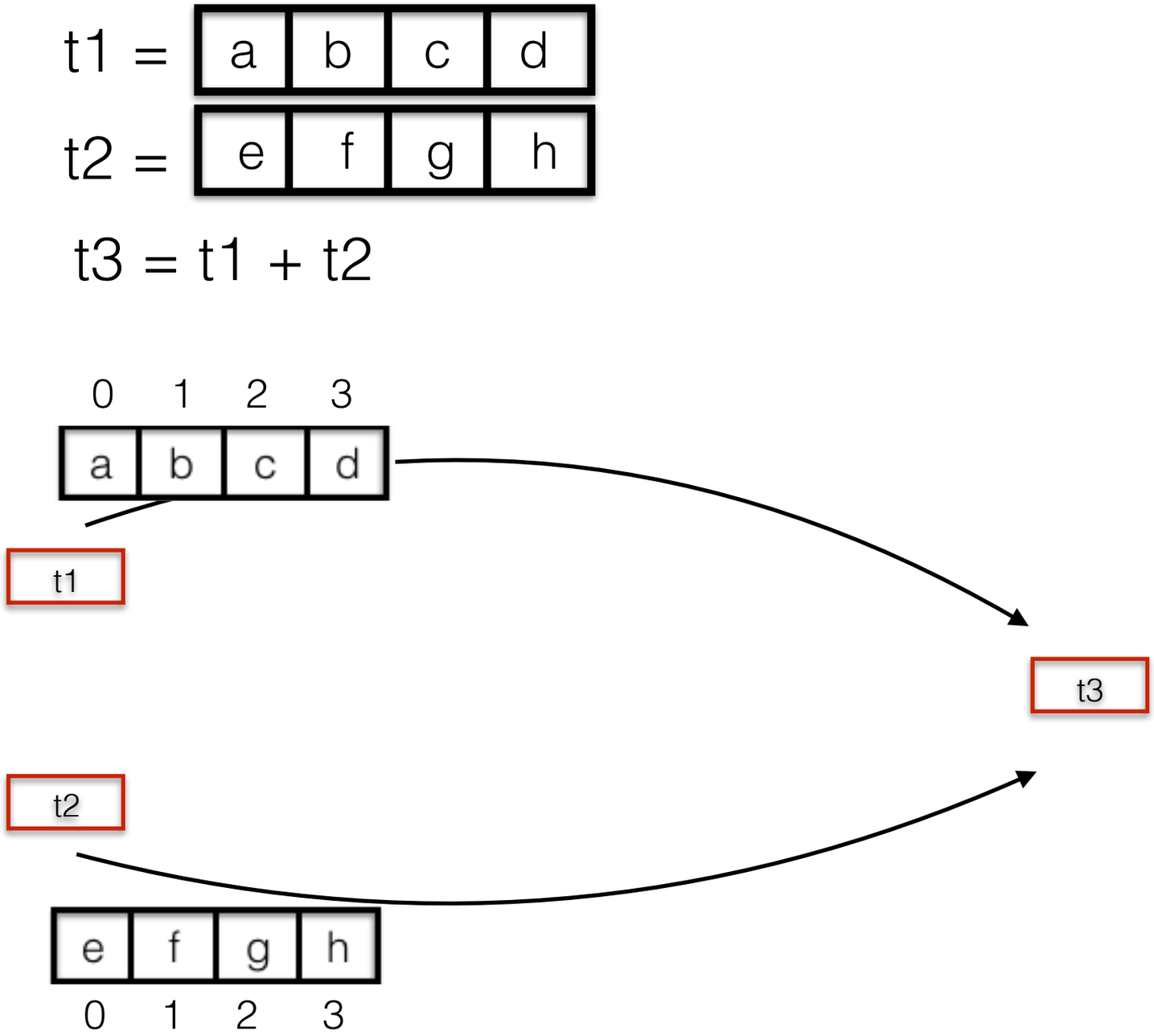
t3 = t1 + t2



Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #3: Copy on Join



Overcoming Parallelism Bottlenecks

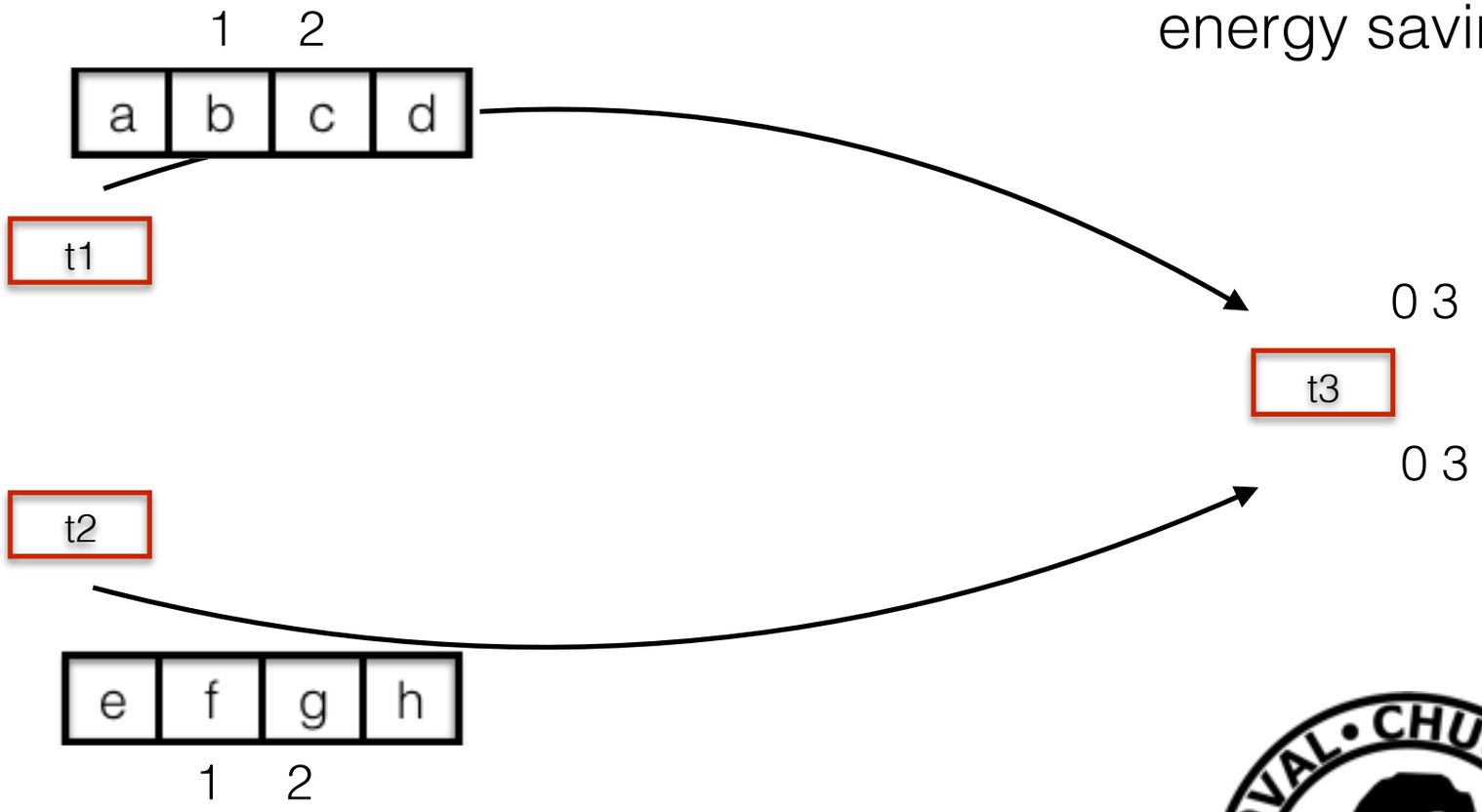
@gustavopinto

Bottleneck #3: Copy on Join

t1 = | a | b | c | d |

t2 = | e | f | g | h |

t3 = t1 + t2



Up to 12% of energy savings!

Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #4: Scattered Data

t1 = **abababababab** …

a = memory copies for a subtask

b = forks the subtask

Overcoming Parallelism Bottlenecks

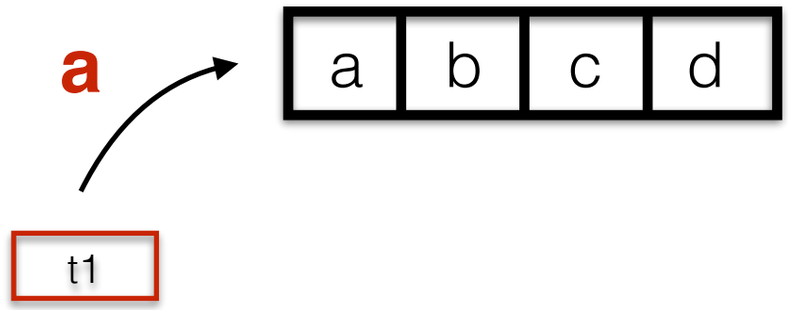
@gustavopinto

Bottleneck #4: Scattered Data

t1 = **a**b**a**b**a**b**a**b …

a = memory copies for a subtask

b = forks the subtask



Overcoming Parallelism Bottlenecks

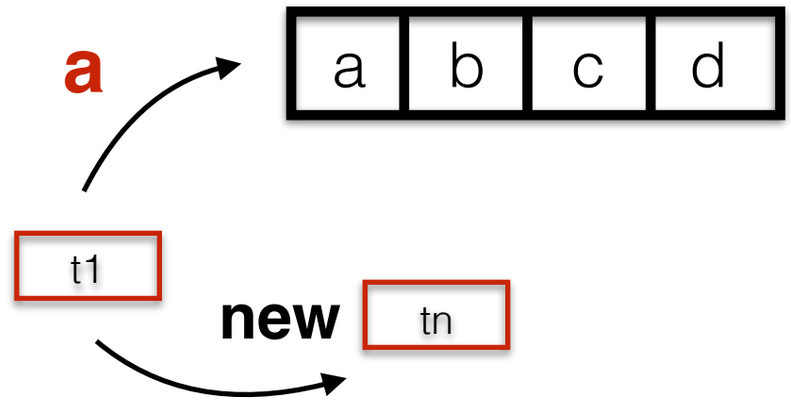
@gustavopinto

Bottleneck #4: Scattered Data

t1 = **abababababab** …

a = memory copies for a subtask

b = forks the subtask



Overcoming Parallelism Bottlenecks

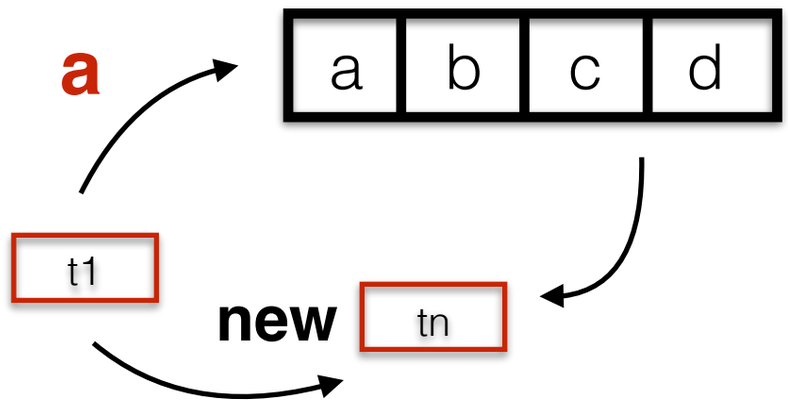
@gustavopinto

Bottleneck #4: Scattered Data

t1 = **abababababab** …

a = memory copies for a subtask

b = forks the subtask



Overcoming Parallelism Bottlenecks

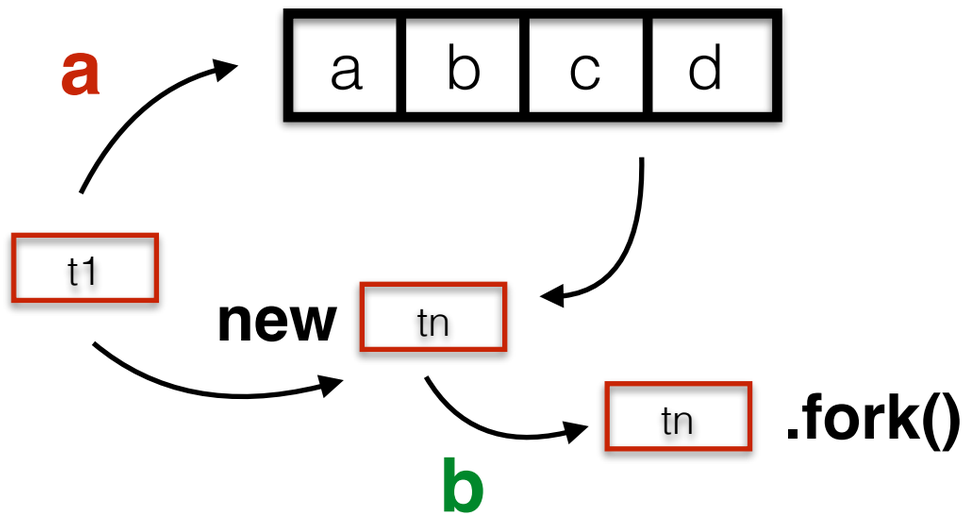
@gustavopinto

Bottleneck #4: Scattered Data

t1 = **abababababab** …

a = memory copies for a subtask

b = forks the subtask



Overcoming Parallelism Bottlenecks

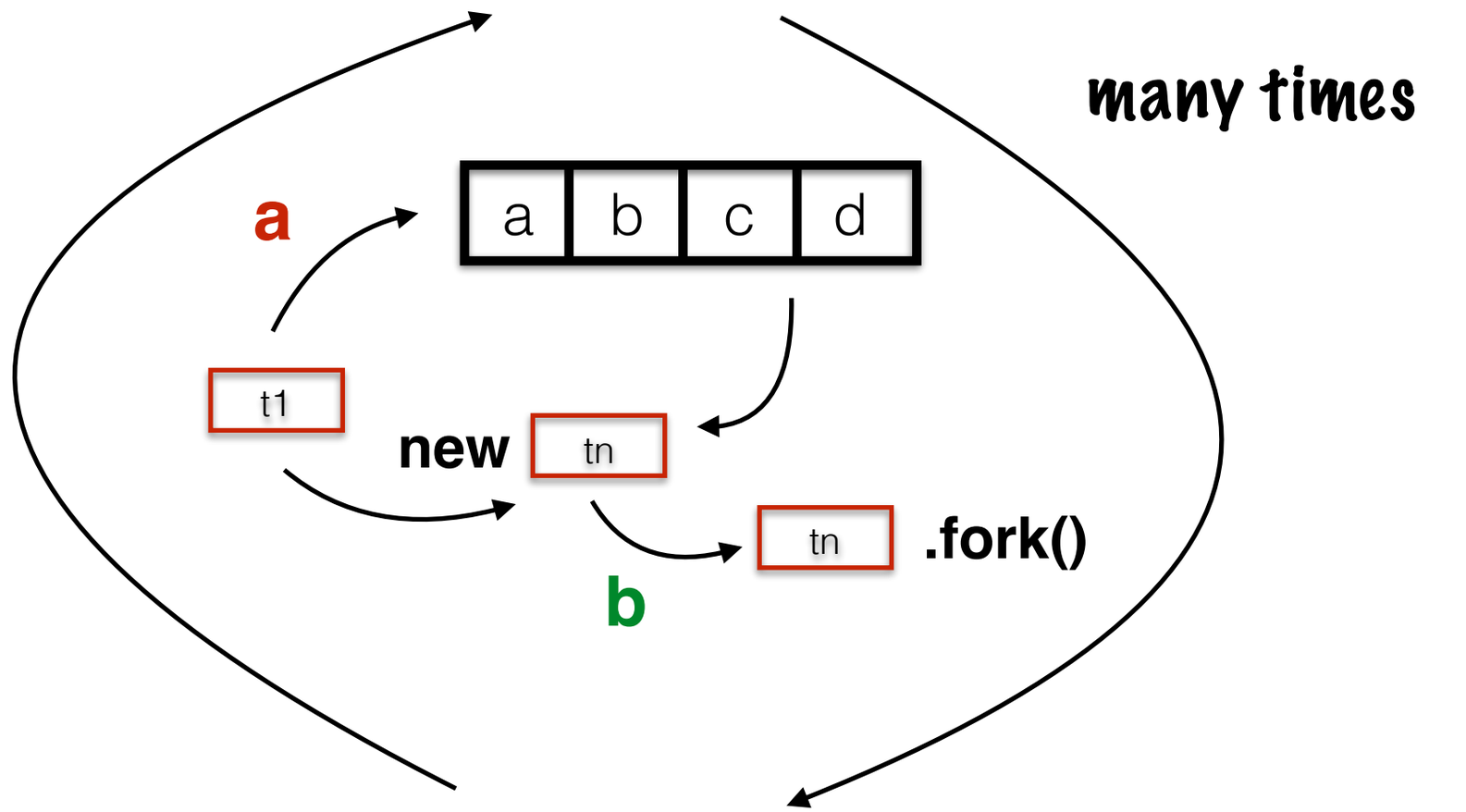
@gustavopinto

Bottleneck #4: Scattered Data

t1 = **abababab** …

a = memory copies for a subtask

b = forks the subtask



Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #4: Scattered Data

t1 = **abababababab** …

a = memory copies for a subtask

b = forks the subtask

t1 = **aaaabbbb**…

Overcoming Parallelism Bottlenecks

@gustavopinto

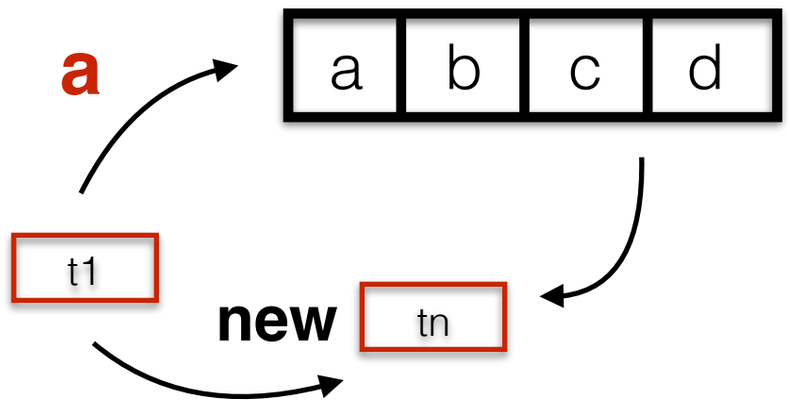
Bottleneck #4: Scattered Data

t1 = **abababababab** …

a = memory copies for a subtask

b = forks the subtask

t1 = **aaaabbbb**…



Overcoming Parallelism Bottlenecks

@gustavopinto

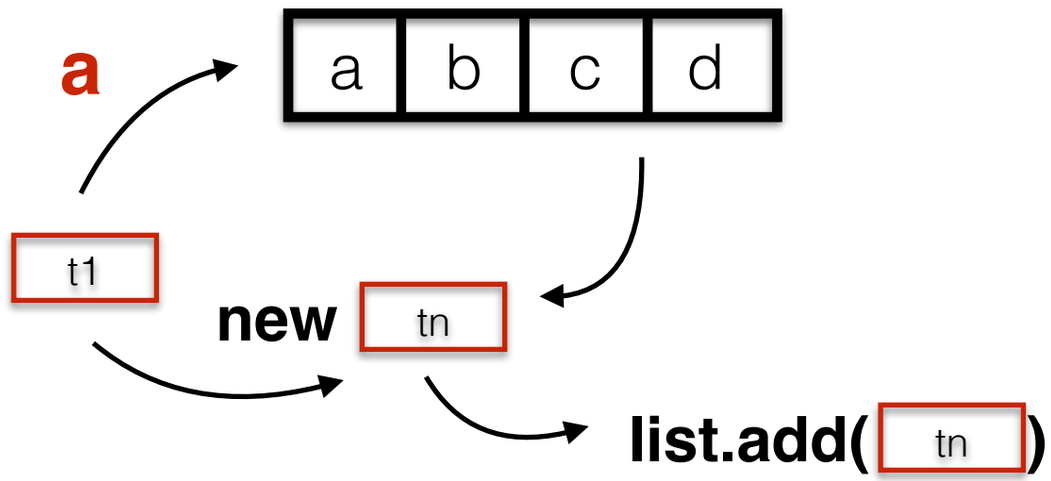
Bottleneck #4: Scattered Data

t1 = **abababababab** …

a = memory copies for a subtask

b = forks the subtask

t1 = **aaaabbbb**…



Overcoming Parallelism Bottlenecks

@gustavopinto

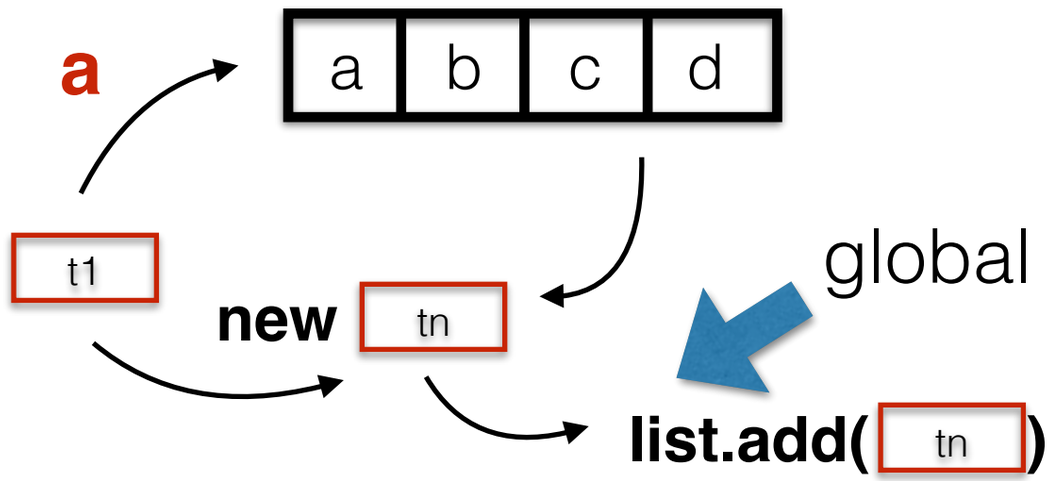
Bottleneck #4: Scattered Data

t1 = **abababab** …

a = memory copies for a subtask

b = forks the subtask

t1 = **aaaabbbb**…



Overcoming Parallelism Bottlenecks

@gustavopinto

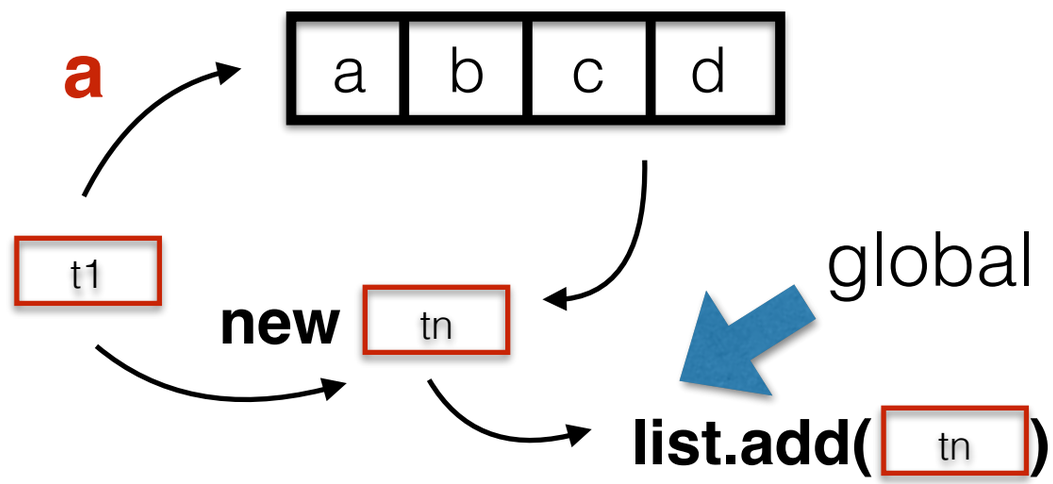
Bottleneck #4: Scattered Data

t1 = **abababababab** …

a = memory copies for a subtask

b = forks the subtask

t1 = **aaaabbbb**…



after creating the objects..

```
for task in list:
    tn .fork()
```

b

Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #4: Scattered Data

t1 = **abababababab** …

a = memory copies for a sub

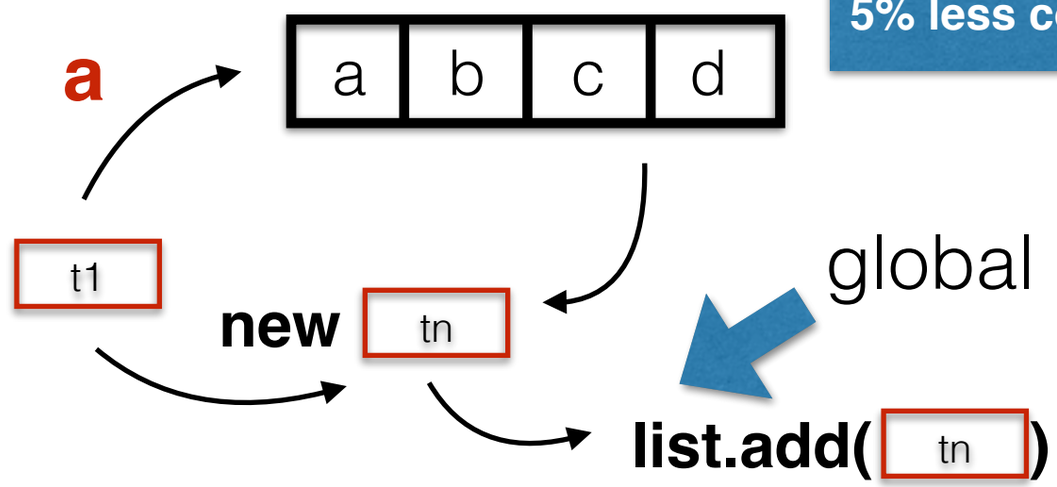
b = forks the subtask

t1 = **aaaabbbb**…

10% of energy savings

3% less cache misses

5% less context switches



after creating the objects..

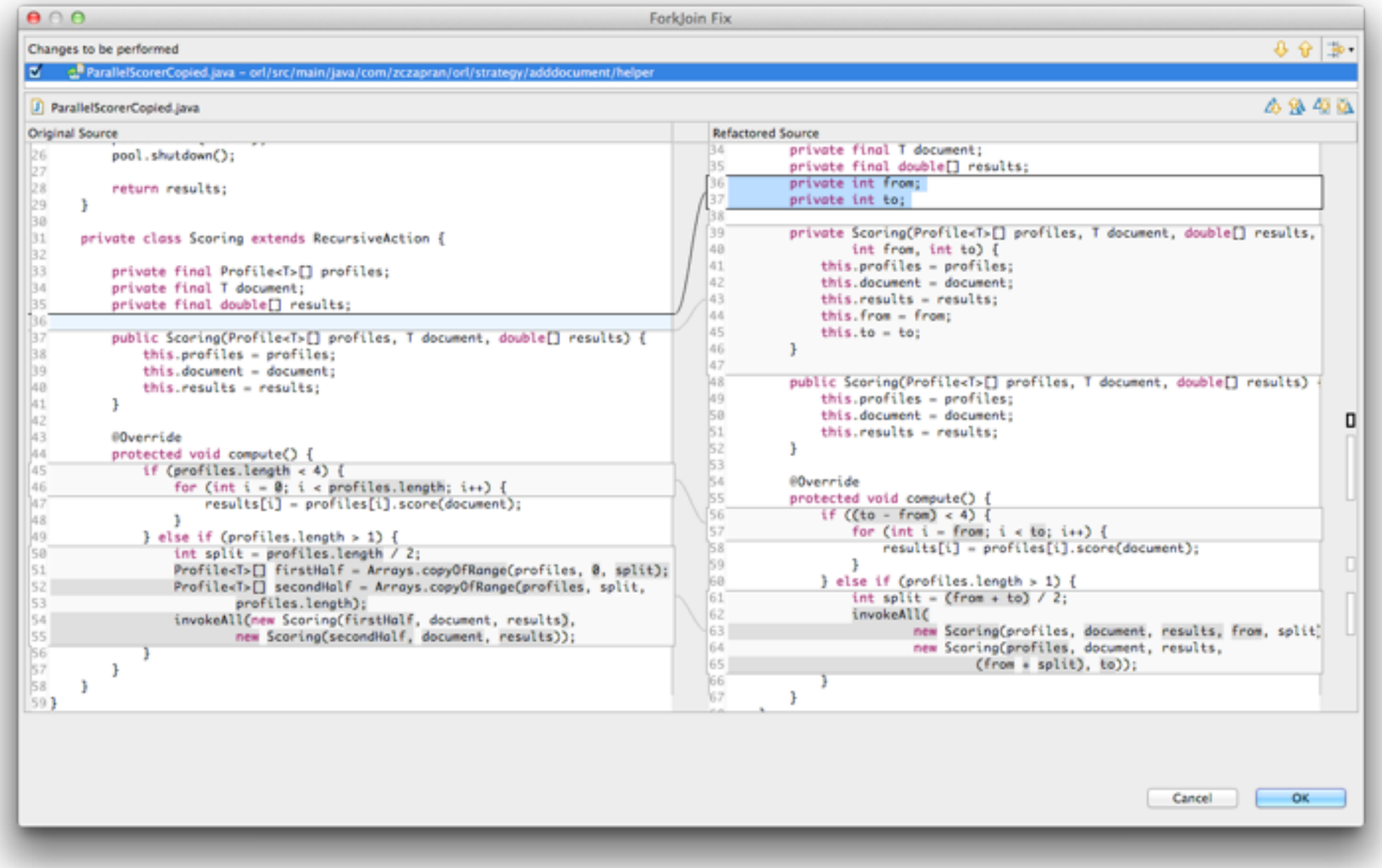
```
for task in list:
    tn .fork()
```

b

Overcoming Parallelism Bottlenecks

@gustavopinto

Automating Bottleneck #2: Copy on Fork



Overcoming Parallelism Bottlenecks

@gustavopinto

Patching Bottleneck #2: Copy on Fork

Proposed #1 **Improving ForkJoin usage for better performance** on Dec 5, 2014
czbabl/itemupdown

Merged #1 **Improving ForkJoin usage for better performance** on Dec 5, 2014
toby1984/jAcer

Proposed #1 **Improving ForkJoin usage for better performance** on Dec 5, 2014
nikkrichko/Educational

Merged #1 **Improving ForkJ**
mayukh42/scalatuts

Merged #1 **Improving ForkJ**
tfreese/knn

Closed #1 **Improving ForkJ**
TarantulaTechnology/

Proposed #1 **Improving ForkJ**
statsbiblioteket/doms-

Merged #1 **Improving ForkJ**
2sbsbsb/ForkAndJoin

Proposed #206 **Improving ForkJoin usage for better performance** on De
Netflix/exhibitor

Proposed #1 **Improving ForkJoin usage for better performance** on Dec 5
robitobi/Solitaire

Merged #1 **Improving ForkJoin usage for better performance** on Dec 5
mariofts/javaOneBR-2012

Proposed #1 **Improving ForkJoin usage for better performance** on Dec 5
cacling/mywiki

Merged #14 **Improving ForkJoin usage for better performance** on Dec
ejisto/ejisto

Proposed #1 **Improving ForkJoin usage for better performance** on Dec 5
ctpahhik/cq4j

Merged #1 **Updating ForkJoin usage for better performance and ener**
cjlarose/MagicSquares

Overcoming Parallelism Bottlenecks

@gustavopinto

Patching Bottleneck #2: Copy on Fork

Proposed #1 **Improving ForkJoin usage for better performance** on Dec 5, 2
czbabl/itemupdown

Merged #1 **Improving ForkJoin usage for better performance** on Dec 5, 2
toby1984/jAcer

Proposed #1 **Improving ForkJoin usage for better performance** on Dec 5, 2
nikkrichko/Educational

Merged #1 **Improving ForkJ**
mayukh42/scalatuts

Merged #1 **Improving ForkJ**
tfreese/knn

Closed #1 **Improving ForkJ**
TarantulaTechnology/

Proposed #1 **Improving ForkJ**
statsbiblioteket/doms-

Merged #1 **Improving ForkJ**
2sbsbsb/ForkAndJoin

Proposed #206 **Improving ForkJoin usag**
Netflix/exhibitor

Proposed #1 **Improving ForkJoin usage**
robitobi/Solitaire

Merged #1 **Improving ForkJoin usage for better performance** on Dec 5
mariofts/javaOneBR-2012

Proposed #1 **Improving ForkJoin usage for better performance** on Dec 5
cacling/mywiki

Merged #14 **Improving ForkJoin usage for better performance** on Dec
ejisto/ejisto

Proposed #1 **Improving ForkJoin usage for better performance** on Dec 5
ctpahhik/cq4j

Merged #1 **Updating ForkJoin usage for better performance and ener**
cjlarose/MagicSquares

7/9 of projects that **replied** have **accepted** the PR

Why ForkJoin?

Work Stealing

n ForkJoin Task

n ForkJoin Worker

Understanding Parallelism Bottlenecks

For each version, we measured execution time and energy consumption

Intel CPU: A 2×8-core (32-cores w/ hyper-threading), running Debian, 2.600Hz, with 64GB of memory. JDK version 1.7.0_71, build 14.

JRapl: Software-based energy measurement

@gustavopinto

Overcoming Parallelism Bottlenecks

Bottleneck #4: Scattered Data

t1 = abababababab …

a = memory copies for a sub

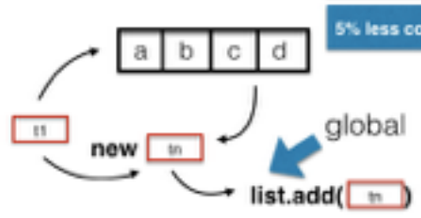
b = forks the subtask

10% of energy savings

t1 = aaaabbbb…

3% less cache misses

5% less context switches

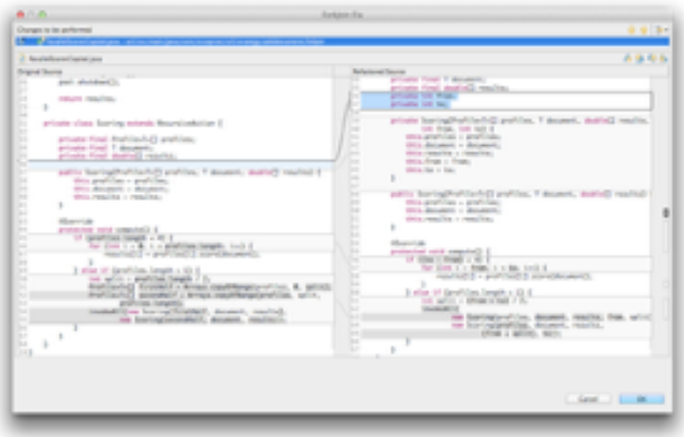


after creating the objects..

```
for task in list:
    tn.fork()
```

Overcoming Parallelism Bottlenecks

Automating Bottleneck #2: Copy on Fork



@gustavopinto

Why ForkJoin?

Work Stealing

ForkJoin Worker

Understanding Parallelism Bottlenecks

For each version, we measured execution time

Intel CPU: A 2×8-core (32-cores w/ hyper-threading) running Debian, 2.600Hz, with 64GB of memory. JDK version 1.7.0_71, build 14.

JRapl: Software-based energy measurement

@gustavopinto

Questions?

@gustavopinto

gpinto@ufpa.br

Bottleneck #4: Scattered Data

Overcoming Parallelism Bottlenecks

t1 = abababababab ...

a = memory copies for a sub

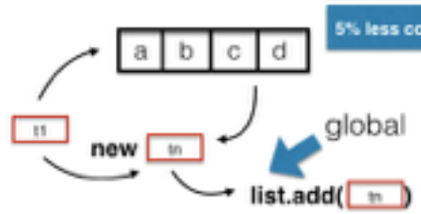
b = forks the subtask

10% of energy savings

t1 = aaaabbbb...

3% less cache misses

5% less context switches

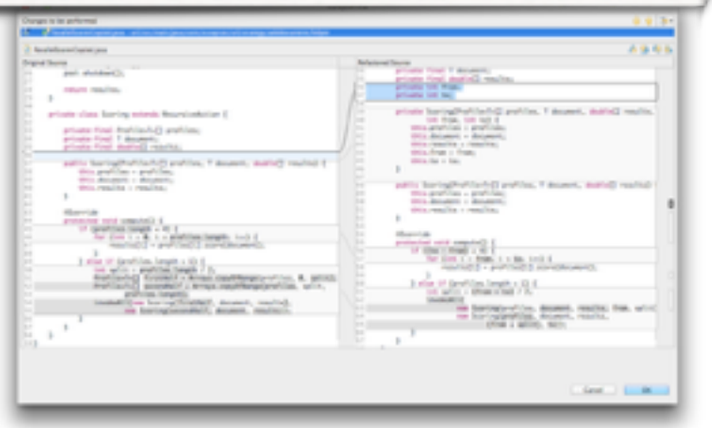


after creating the objects..

for task in list:
tn .fork()

Automating Bottleneck #2: Copy on Fork

Overcoming Parallelism Bottlenecks



@gustavopinto

Understanding and Overcoming Parallelism Bottlenecks in ForkJoin Applications

@gustavopinto

A. Canino

F. Castor

G. Xu

Y. D. Liu