My personal, biased view of

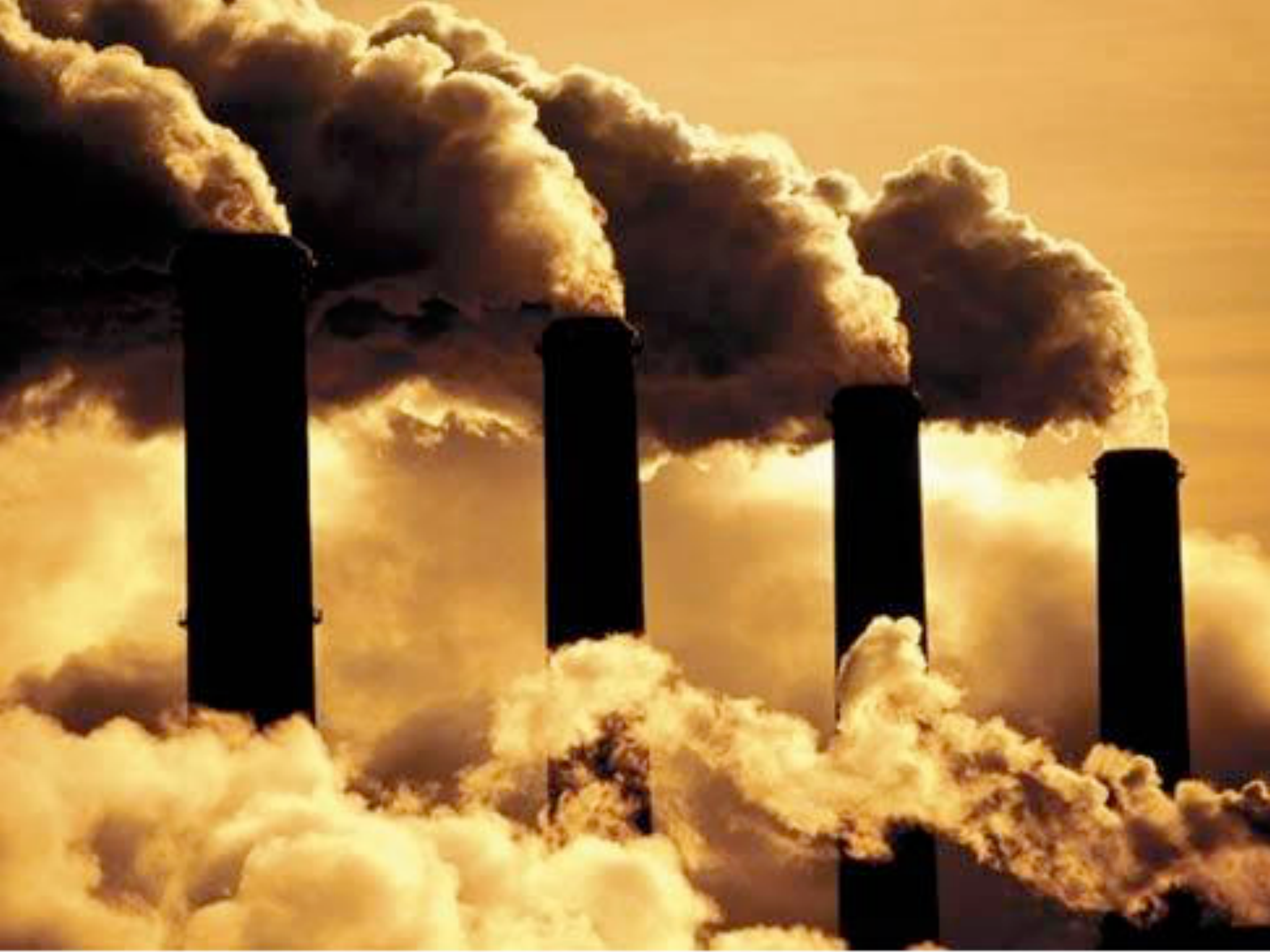
The Last Five Years of Energy Consumption Research

Gustavo Pinto

@gustavopinto

gpinto@ufpa.br

UNIVERSIDADE FEDERAL DO PARÁ



That's really the deep fre data center

By ROB WAUGH
UPDATED: 09:48 GMT, 28 Oc

- **First plant outside U.S.**
- **In northern Sweden 60 year**
- **Other web giants such**
- **Near huge dam on river**
- **Plant will have 14 backu**

Facebook's new server far chosen because the fierce

Facebook looked at severa become faster for users ac

Efficiency: How others can do it

Five things you can do now

At Google, we've spent more than a decade improving the energy efficiency of our data centers, and we've picked up some best practices along the way. Whether you're running a small or large data center, you can apply several simple design choices to improve the efficiency of your facility, reduce costs, and reduce your impact on the environment.

Here are our top five best practices:

Google's Green Data Centers: Network POP Case Study (PDF)

In addition to the large-scale data centers used to deliver our web services, we maintain several small, network point of presences (POPs). See how we applied some of the efficiency best practices during a retrofit to save money and reduce greenhouse gas emissions.

1. **Measure PUE**

You can't manage what you don't measure, so be sure to track your data center's energy use. The industry uses a ratio called Power Usage Effectiveness (PUE) to measure and help reduce the energy used for non-computing functions like cooling and power distribution. To effectively use PUE, it's important to measure often. We sample at least once per second. It's even more important to capture energy data over the entire year, since seasonal weather variations affect PUE. Learn more.

2. **Manage airflow**

Good air flow management is crucial to efficient data center operation. Minimize hot and cold air mixing by using well-designed containment. Then, eliminate hot spots and be sure to use blanking plates (or flat sheets of metal) for any empty slots in your rack. We've found that a little analysis can have big payoffs. For example, thermal modeling using computational fluid dynamics (CFD) can help you quickly characterize and optimize air flow for your facility without having to reorganize your computing room. Learn more.

3. **Adjust the thermostat**

The need to keep data centers at 70°F is a myth. Virtually all equipment manufacturers allow you to run your cold aisle at 80°F or higher. If your facility uses an economizer (which we highly recommend), run elevated cold aisle temperatures to enable more days of "free cooling" and higher energy savings. Learn more.

Uh oh

@gustavopinto


Respect
by battery!
We need
energy
efficient
apps!
Go away
bad
apps!

Waze battery consumption

Moderators: Unholy, Unholy

POSTREPLY Search this topic... Search

16 posts • Page 1 of 2 • 1

Waze battery consumption

QUOTE

by **sfe0** » Sat Jun 02, 2012 10:01 am

Hello all!
Been using Waze for about two weeks now and I really like it. Haven't been stuck in any traffic jam during this time, like I normally do this time of year, but I have received a few warnings from Waze and driven alternative roads with great success. Thank You for that!
But, how come Waze uses so much battery?
I have a SonyEricssonArcS and the SonyEricsson car charger is the only one I've tried this far that can cope with the energy hungry Waze app. All other chargers, 4 of them, can't deliver enough current to keep the phones battery from draining when running Waze. The phone itself gets very hot and from what I've read on the net this isn't a problem just for me.
I also use another similar app in my Xperia and that is the Geocaching app "Neongeo". Neongeo, like Waze, uses GPS, a-GPS, maps, shows me moving around on the map, live internet update and keeps the phones screen always on. Still, it consumes what seems like much less power than Waze and the phone does not get hot at all.
Why?

sfe0

Posts: 4
Joined: Thu May 24, 2012 10:29 am
Has thanked: 0 time
Been thanked: 0 time

Waze battery consumption

Moderators: Unholy, Unholy

POSTREPLY Search this topic... Search

16 posts • Page 1 of 2 • 1

Waze battery consumption

QUOTE

by **sfe0** » Sat Jun 02, 2012 10:01 am

Hello all!
Been using Waze for about two weeks now and I really like it. Haven't been stuck in any traffic jam during this time, like I normally do this time of year, but I have received a few warnings from Waze and driven alternative roads with great success. Thank You for that!
But, how come Waze uses so much battery?
I have a SonyEricssonArcS and the SonyEricsson car charger is the only one I've tried this far that can cope with the energy hungry Waze app. All other chargers, 4 of them, can't deliver enough current to keep the phones battery from draining when running Waze. The phone itself gets very hot and from what I've read on the net this isn't a problem just for me.
I also use another similar app in my Xperia and that is the Geocaching app "Neongeo". Neongeo, like Waze, uses GPS, a-GPS, maps, shows me moving around on the map, live internet update and keeps the phones screen always on. Still, it consumes what seems like much less power than Waze and the phone does not get hot at all.
Why?

sfe0

Posts: 4
Joined: Thu May 24, 2012 10:29 am
Has thanked: 0 time
Been thanked: 0 time

Waze battery consumption

Moderators: Unholy, Unholy

POSTREPLY Search this topic... Search

16 posts • Page 1 of 2 • 1

Waze battery consumption

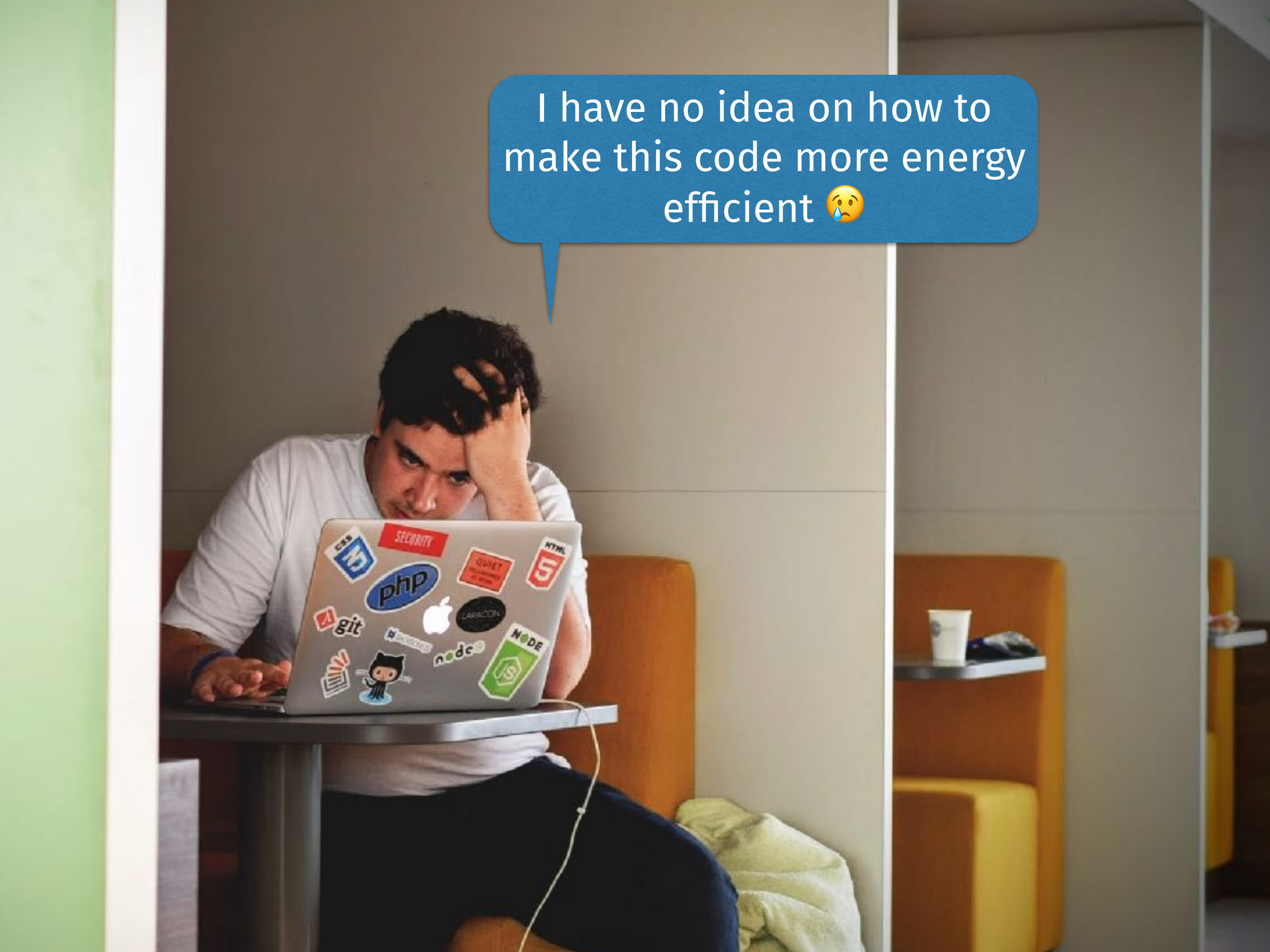
QUOTE

by **sfe0** » Sat Jun 02, 2012 10:01 am

Hello all!
Been using Waze for about two weeks now and I really like it. Haven't been stuck in any traffic jam during this time, like I normally do this time of year, but I have received a few warnings from Waze and driven alternative roads with great success. Thank You for that!
But, how come Waze uses so much battery?
I have a SonyEricssonArcS and the SonyEricsson car charger is the only one I've tried this far that can cope with the energy hungry Waze app. All other chargers, 4 of them, can't deliver enough current to keep the phones battery from draining when running Waze. The phone itself gets very hot and from what I've read on the net this isn't a problem just for me.
I also use another similar app in my Xperia and that is the Geocaching app "Neongeo". Neongeo, like Waze, uses GPS, a-GPS, maps, shows me moving around on the map, live internet update and keeps the phones screen always on. Still, it consumes what seems like much less power than Waze and the phone does not get hot at all.
Why?

sfe0

Posts: 4
Joined: Thu May 24, 2012 10:29 am
Has thanked: 0 time
Been thanked: 0 time


I have no idea on how to make this code more energy efficient 😢
SECURITY
QUIET
HTML
php
git
LARACON
NODE
node

2014

2M+ Users

5M+ Questions

10M+ Answers

50GB+ of data

Mining Questions about Software Energy Consumption

Gustavo Pinto[1,2], Fernando Castor[1], Yu David Liu[2]

[1]Federal University of Pernambuco
Recife, PE, Brazil
{ghlp, castor}@cin.ufpe.br

[2]SUNY Binghamton
Binghamton, NY 13902, USA
davidL@cs.binghamton.edu

ABSTRACT

A growing number of software solutions have been proposed to address application-level energy consumption problems in the last few years. However, little is known about how much software developers are concerned about energy consumption, what aspects of energy consumption they consider important, and what solutions they have in mind for improving energy efficiency. In this paper we present the first empirical study on understanding the views of application programmers on software energy consumption problems. Using StackOverflow as our primary data source, we analyze a carefully curated sample of more than 300 questions and 550 answers from more than 800 users. With this data, we observed a number of interesting findings. Our study shows that practitioners are aware of the energy consumption problems: the questions they ask are not only *diverse* – we found 5 main themes of questions – but also often more *interesting* and *challenging* when compared to the control question set. Even though energy consumption-related questions are popular when considering a number of different popularity measures, the same cannot be said about the quality of their answers. In addition, we observed that some of these answers are often flawed or vague. We contrast the advice provided by these answers with the state-of-the-art research on energy consumption. Our summary of software energy consumption problems may help researchers focus on what matters the most to software developers and end users.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Documentation, Human Factors

Keywords

Software Energy Consumption, Practitioners, Q&A

MSR '14, May 31 – June 1, 2014, Hyderabad, India

1. INTRODUCTION

Nowadays, thanks to the rapid proliferation of mobile phones, tablets, and unwired devices in general, energy efficiency is becoming a key software design consideration where the energy consumption is closely related to battery lifetime. It is also of increasing interest in the non-mobile arena, such as data centers and desktop environments. Energy-efficient solutions are highly sought after across the compute stack, with more established results through innovations in hardware/architecture [2, 14, 28], operating systems [10, 19, 24], and runtime systems [8, 25, 31]. In recent years, there is a growing interest in studying energy consumption from higher layers of the compute stack and most of these studies focus on application software [13, 15, 18, 23, 26, 29]. These approaches complement prior hardware/OS-centric solutions, so that improvements at the hardware/OS level are not cancelled out at the application level, e.g., due to misuses of language/library/application features.

We believe a critical dimension to further improve energy efficiency of software systems is to understand *how software developers think*. The needs of developers and the challenges they face may help energy-efficiency researchers stay focused on the real-world problems. The collective wisdom shared by developers may serve as a practical guide for future energy-aware and energy-efficient software development. The conceptually incorrect views they hold may inspire educators to develop more state-of-the-art curricula.

The goal of this work is to obtain a deeper understanding of (i) whether application programmers are interested in software energy consumption, and, if so, (ii) how they are dealing with energy consumption issues. Specifically, the questions we are trying to answer are:

RQ1 What are the distinctive characteristics of energy-related questions?

RQ2 What are the most common energy-related problems faced by software developers?

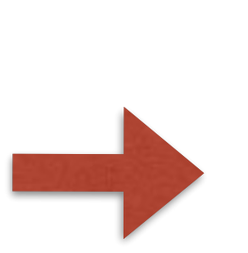
RQ3 According to developers, what are the main causes for software energy consumption?

RQ4 What solutions do developers employ or recommend to save energy?

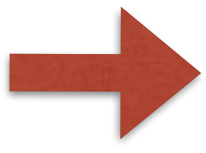
Our study is based on data from StackOverflow, a collaborative development questions and answers (Q&A) website. As one of the most popular forums in the software development world, StackOverflow is often used for software engineering studies [20, 30, 32]. It contains over 2

MSR'14

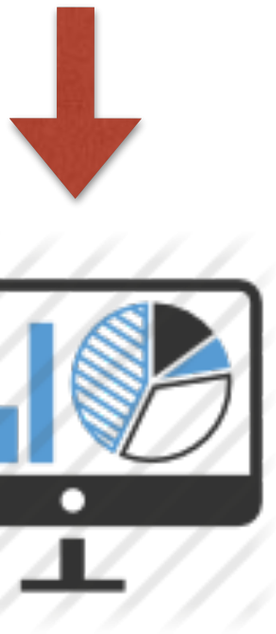
5M Questions



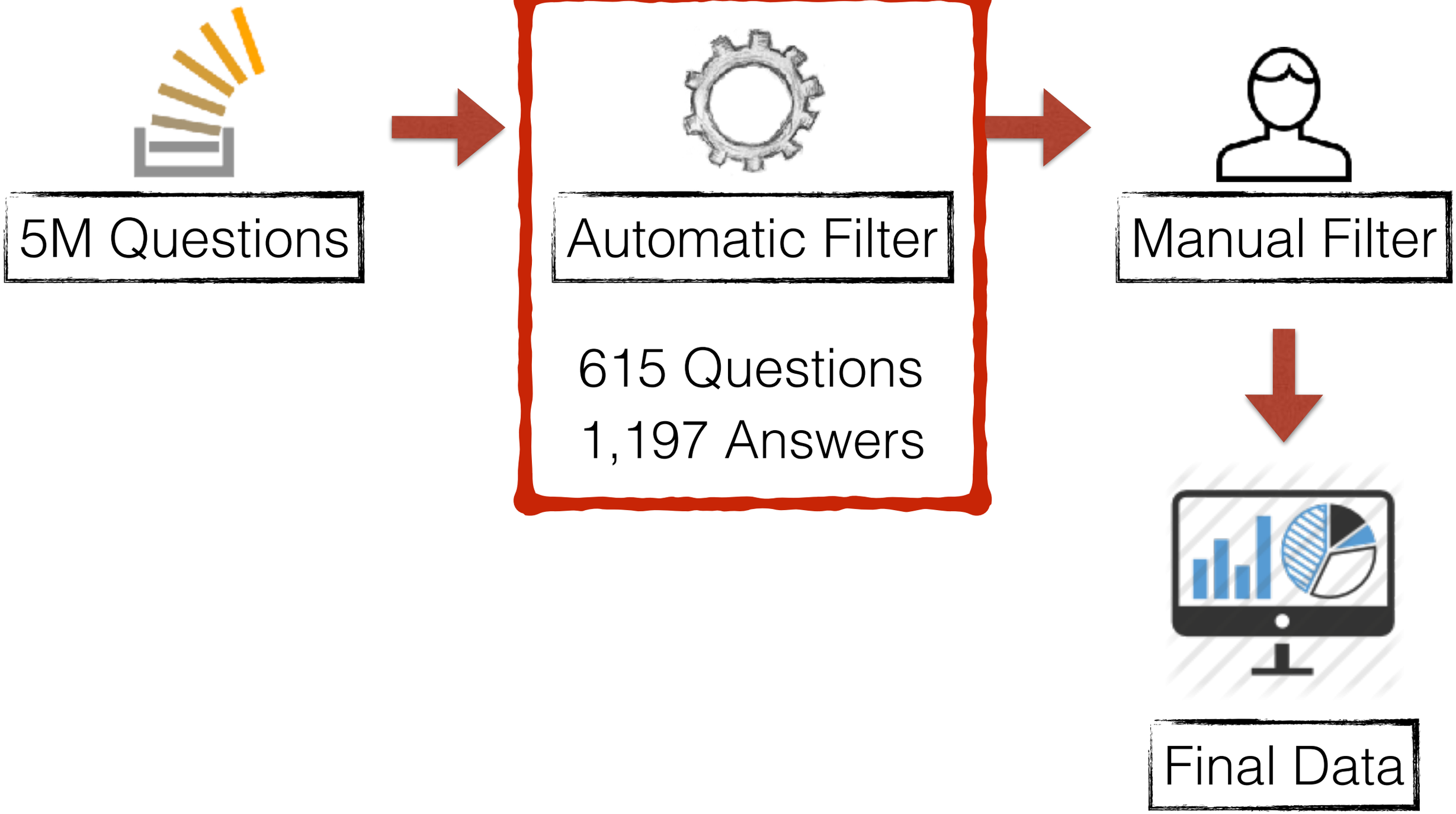
Automatic Filter



Manual Filter



Final Data


5M Questions
Automatic Filter
615 Questions
1,197 Answers
Manual Filter
Final Data

5M Questions

Automatic Filter

Manual Filter

Final Data

325 Questions
558 Answers

Base Group

from 2008 to 2013

Research Questions

- **RQ1:** What are the most common energy-related **problems** faced by software developers?

- **RQ2:** What are the main **causes** for software energy consumption problems?

- **RQ3:** What **solutions** do developers employ or recommend to save energy?

Energy-Related Problems

- Measurements (59/97 — Q/A)
- General Knowledge (40/84 — Q/A)
- Code design (36/133 — Q/A)
- Context-specific (83/110 — Q/A)
- Noise (107/134 — Q/A)

“I want to measure the energy consumption of my own application (which I can modify) [...] on Windows CE 5.0 and Windows Mobile 5/6. Is there some kind of API for this?”

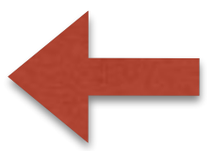
- Measurements (59/97 — Q/A)
- General Knowledge (40/84 — Q/A)
- Code design (36/133 — Q/A)
- Context-specific (83/110 — Q/A)
- Noise (107/134 — Q/A)

“Are there any s/w high level design considerations […] to make the code as power efficient as possible?”

- Measurements (59/97 — Q/A)
- General Knowledge (40/84 — Q/A)
- Code design (36/133 — Q/A)
- Context-specific (83/110 — Q/A)
- Noise (107/134 — Q/A)

Energy-Related Problems

- Measurements (59/97 — Q/A)
- General Knowledge (40/84 — Q/A)
- Code design (36/133 — Q/A)



- Context-specific (83/110 — Q/A)
- Noise (107/134 — Q/A)

— Highest popularity
— Highest A per Q ratio
— Highest success rate

Energy-Related Causes

- Unnecessary resource usage (49 occurrences)
- Fault GPS behavior (42 occurrences)
- Background activities (40 occurrences)
- Excessive synchronization (32 occurrences)
- Background wallpapers (17 occurrences)
- Advertisement (11 occurrences)

"to have a background application that monitors device usage, identifies unused/idle resources, and acts appropriately"

- Unnecessary resource usage (49 occurrences)
- Fault GPS behavior (42 occurrences)
- Background activities (40 occurrences)
- Excessive synchronization (32 occurrences)
- Background wallpapers (17 occurrences)
- Advertisement (11 occurrences)

"When there are bugs that keep the GPS turned on too long they go to the top of the list to get fixed"

- Unnecessary resource usage (49 occurrences)
- Fault GPS behavior (42 occurrences)
- Background activities (40 occurrences)
- Excessive synchronization (32 occurrences)
- Background wallpapers (17 occurrences)
- Advertisement (11 occurrences)

Energy-Related Solutions

- Keep IO to a minimum (29 occurrences)
- Bulk operations (24 occurrences)
- Avoid polling (17 occurrences)
- Hardware Coordination (11 occurrences)
- Concurrent Programming (9 occurrences)
- Race to idle (7 occurrences)

“do not flood the output stream with null values”

- Keep IO to a minimum (29 occurrences)
- Bulk operations (24 occurrences)
- Avoid polling (17 occurrences)
- Hardware Coordination (11 occurrences)
- Concurrent Programming (9 occurrences)
- Race to idle (7 occurrences)

“Don’t transfer say 1 file, and then wait for a bit to do another transfer. Instead, transfer right after the other.”

- Keep IO to a minimum (29 occurrences)
- Bulk operations (24 occurrences)
- Avoid polling (17 occurrences)
- Hardware Coordination (11 occurrences)
- Concurrent Programming (9 occurrences)
- Race to idle (7 occurrences)

Do researchers agree?

- Keep IO to a minimum (29 occurrences)
- Bulk operations (24 occurrences)
- Avoid polling (17 occurrences)
- Hardware Coordination (11 occurrences)
- Concurrent Programming (9 occurrences)
- Race to idle (7 occurrences)

Do researchers agree?

- ✓ Keep IO to a minimum (29 occurrences)
- ✓ Bulk operations (24 occurrences)
- ✓ Avoid polling (17 occurrences)
- ✓ Hardware Coordination (11 occurrences)
- ✗ Concurrent Programming (9 occurrences)
- ✗ Race to idle (7 occurrences)

2015

4Mi+ Users

19Mi+ Repositories

Mining Energy-Aware Commits

Irineu Moura, Gustavo Pinto, Felipe Ebert and Fernando Castor
Federal University of Pernambuco
Informatics Center
{imlm2, ghlp, fe, castor}@cin.ufpe.br

***Abstract*—Over the last years, energy consumption has become a first-class citizen in software development practice. While energy-efficient solutions on lower-level layers of the software stack are well-established, there is convincing evidence that even better results can be achieved by encouraging practitioners to participate in the process. For instance, previous work has shown that using a newer version of a concurrent data structure can yield a 2.19x energy savings when compared to the old associative implementation [75]. Nonetheless, little is known about how much software engineers are employing energy-efficient solutions in their applications and what solutions they employ for improving energy-efficiency. In this paper we present a qualitative study of "energy-aware commits". Using Github as our primary data source, we perform a thorough analysis on an initial sample of 2,189 commits and carefully curate a set of 371 energy-aware commits spread over 317 real-world non-trivial applications. Our study reveals that software developers heavily rely on low-level energy management approaches, such as frequency scaling and multiple levels of idleness. Also, our findings suggest that ill-chosen energy saving techniques can impact the correctness of an application. Yet, we found what we call "energy-aware interfaces", which are means for clients (*e.g.*, developers or end-users) to save energy in their applications just by using a function, abstracting away the low-level implementation details.**

I. Introduction

Thanks to the diversification of modern computing platforms, battery-driven devices such as smartphones, tablets and unwired devices in general are now commonplace in our lives. However, such devices are energy-constrained, as they rely on limited battery power supply. Energy consumption directly affects the perception of users about their quality. For example, in a survey conducted with more than 3,500 respondents from 4 different countries [49], long-lasting battery life has been cited as the most desired feature in a new phone by 71% of the respondents. Likewise, recent research pointed out that battery usage is a key factor for evaluating and adopting mobile applications [86]. As a result, not only researchers [57], [88], [90] but also giants of the software industry [56], [84] have recently started to promote energy-efficient systems.

Traditionally, energy optimization research has focused at the hardware-level (*e.g.*, [52], [59]) and at the system-level (*e.g.*, [54], [78]). Arguably, the strategy of leaving the energy optimization issue to lower-level systems and architecture layers has been successful. The main advantage of this approach is that user applications can be seen as black-boxes, *i.e.*, no prior knowledge of the application source code or its behavior needs to be used to achieve better energy savings.

However, applications also impact energy consumption, although software itself does not consume energy. When one energy-inefficient piece of code is introduced in a system comprising hardware and software components, compilers and runtime systems cannot help much, since they are not aware of the semantics of the program. Unlike performance, where more efficient is always better, sometimes an application will purposefully consume more energy to provide its intended functionality, *e.g.*, by activating an energy-intensive device.

In contrast, energy consumption bottlenecks often stem from inappropriate design choices, as is often the case with performance bugs [61]. Finding and fixing these problems requires human insight. It is not always clear for programmers, however, what they can do from a software engineering perspective to save energy. There seem to be misconceptions and lack of appropriate tools [76].

Recent work [64], [63], [75], [77] has shown that there is ample opportunity to improve energy consumption at the application level. As an example, upgrading the `ConcurrentHashMap` class available in Java 7 to its improved successor, available in Java 8, can yield energy savings of 2.19x [75]. Nevertheless, developing an energy-efficient system is a non-trivial task. Even though researchers have made great strides in empirical studies aiming to understand the impact of different program characteristics in energy consumption (*e.g.*, [64], [77], [80], [89]), these studies do not cover a complete range of language constructs and implementation choices. Therefore, developers still do not fully understand how their source code modifications will impact energy consumption [76].

In spite of this grim scenario, many systems that are energy-efficient are being developed in practice. Starting from this premise, in this paper we focus on a timely but overlooked question:

- What solutions do software developers employ to save energy in practice?

To answer this question, we obtained data from GitHub, the most popular source code hosting website in the software development world. At the time when this paper was written, it contained over 21.1 million software repositories[1] and more than 3.5 million contributors[2]. In addition, GitHub is being used in recent software engineering studies [46], [51], [85].

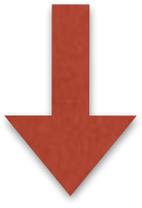
In order to understand what are the energy-efficient solutions that software developers are employing, we started by searching at the commit messages, since programmers usually

[1]https://github.com/features
[2]https://github.com/blog/1470-five-years

MSR'15


GitHub
19M Repos

19M Repos



Query on

GitHub Archive

19M Repos

Query on

GitHub Archive

```
SELECT (repository_url + '/commit/' + payload_commit_id) as commit_link, repository_url,
       repository_owner, repository_name, payload_commit_id, payload_commit_email,
       payload_commit_msg,  payload_commit_name, payload_commit_flag, created_at
       FROM [githubarchive:github.timeline]
       WHERE payload_commit_msg is not null
       AND PARSE_UTC_USEC(created_at) <= PARSE_UTC_USEC ('2014-05-15 23:59:59')
       AND (lower(payload_commit_msg) like '%power consum%'
            OR lower(payload_commit_msg) like '%power efficien%'
            OR lower(payload_commit_msg) like '%power sav%'
            OR lower(payload_commit_msg) like '%save power%'
            OR lower(payload_commit_msg) like '%energy consum%'
            OR lower(payload_commit_msg) like '%energy efficien%'
            OR lower(payload_commit_msg) like '%energy sav%'
            OR lower(payload_commit_msg) like '%save energy%')
       AND type = 'PushEvent'
       GROUP BY commit_link, payload_commit_msg, repository_url, payload_commit_id,
              created_at, repository_owner, repository_name, payload_commit_email,
              payload_commit_name, payload_commit_flag
ORDER BY created_at asc
```

19M Repos

```
SELECT (repository_url + '/commit/' + payload_commit_id) as commit_link, repository_url,
       repository_owner, repository_name, payload_commit_id, payload_commit_email,
       payload_commit_msg,  payload_commit_name, payload_commit_flag, created_at
       FROM [githubarchive:github.timeline]
       WHERE payload_commit_msg is not null
       AND PARSE_UTC_USEC(created_at) <= PARSE_UTC_USEC('2014-05-15 23:59:59')
       AND (lower(payload_commit_msg) like '%power consum%'
            OR lower(payload_commit_msg) like '%power efficien%'
            OR lower(payload_commit_msg) like '%power sav%'
            OR lower(payload_commit_msg) like '%save power%'
            OR lower(payload_commit_msg) like '%energy consum%'
            OR lower(payload_commit_msg) like '%energy efficien%'
            OR lower(payload_commit_msg) like '%energy sav%'
            OR lower(payload_commit_msg) like '%save energy%')
       AND type = 'PushEvent'
       GROUP BY commit_link, payload_commit_msg, repository_url, payload_commit_id,
                created_at, repository_owner, repository_name, payload_commit_email,
                payload_commit_name, payload_commit_flag
       ORDER BY created_at asc
```

Query on

GitHub Archive

2,189 Commits

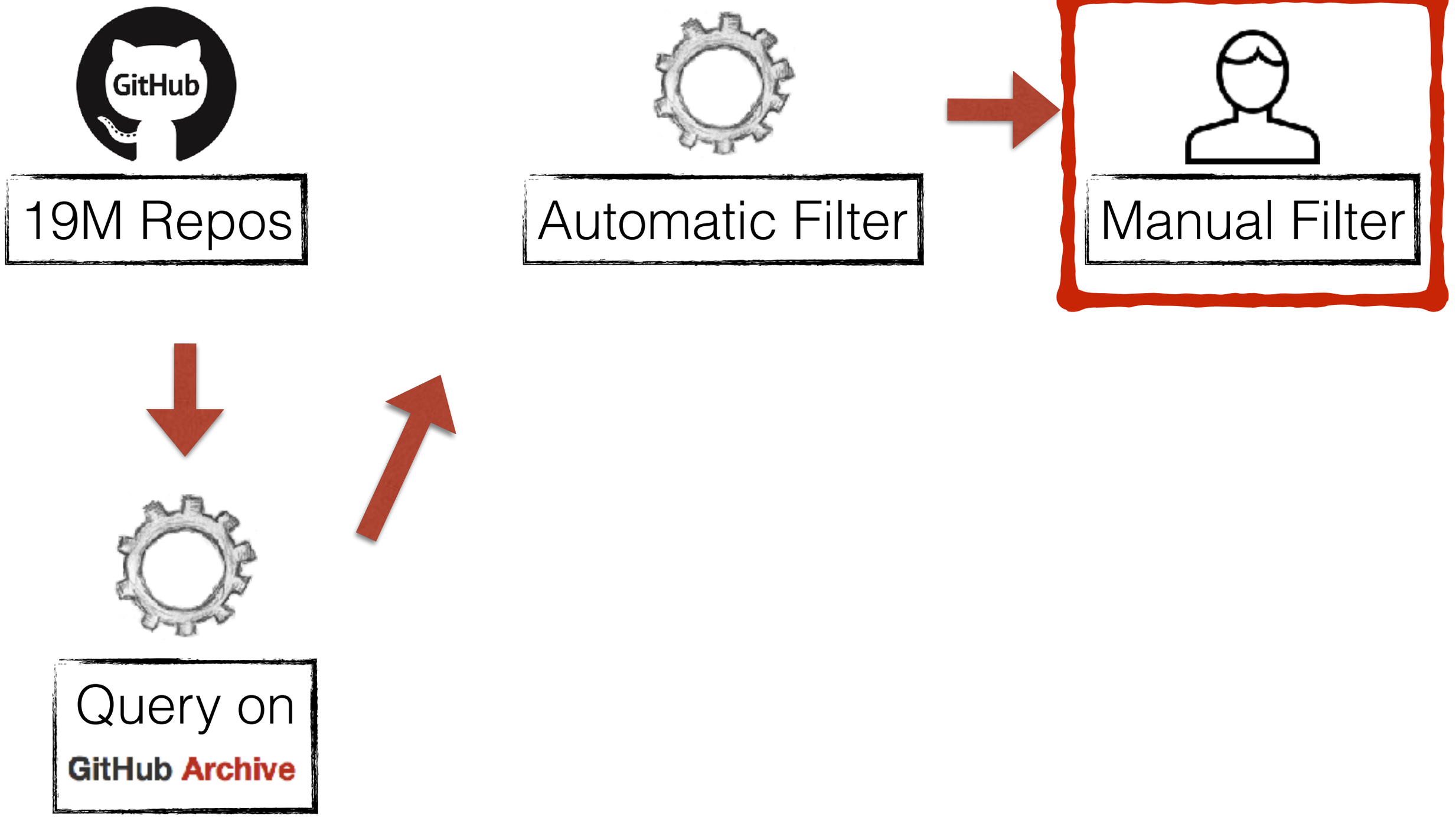
19M Repos

Query on

GitHub Archive

Automatic Filter

1,005 Commits

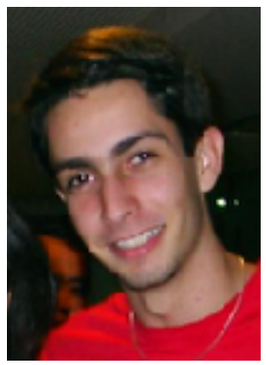

GitHub
19M Repos
Automatic Filter
Manual Filter
Query on
GitHub Archive

19M Repos

Commits
dh7h3
md8ja
j287h
dij873
dj827h
os837
82uan
28a08
2ja82
d0hk0
j29yd
a7jf9
aio92
hnna2

Automatic Filter

Manual Filter



19M Repos

Automatic Filter

Manual Filter

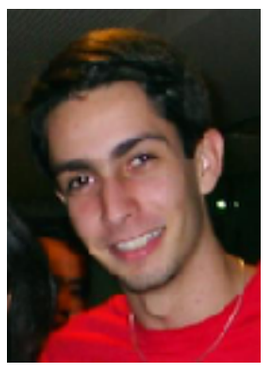
Commits
dh7h3
md8ja
j287h
dij873
dj827h
os837
82uan
28a08
2ja82
d0hk0
j29yd
a7jf9
aio92
hnna2

19M Repos

Commits
dh7h3
md8ja
j287h
dij873
dj827h
os837
82uan
28a08
2ja82
d0hk0
j29yd
a7jf9
aio92
hnna2

Automatic Filter

Manual Filter

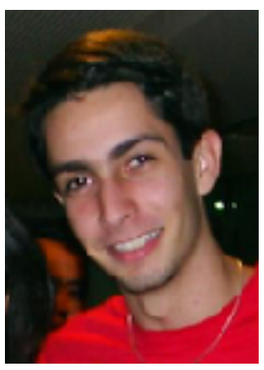


19M Repos

Automatic Filter

Manual Filter

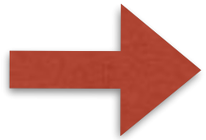
Commits
dh7h3
md8ja
j287h
dij873
dj827h
os837
82uan
28a08
2ja82
d0hk0
j29yd
a7jf9
aio92
hnna2



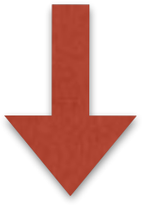
All commits were, at least, double-checked!

19M Repos

Automatic Filter

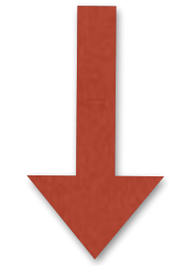


Manual Filter



Query on

GitHub Archive



371
Energy-Aware
Commits

Research Questions

- **RQ1.** What are the **solutions** that developers use to save energy in practice?

- **RQ2.** What software **quality attributes** may be given precedence over energy consumption?

- **RQ3.** How are energy-saving solutions distributed over the **software stack**?

RQ1: Solutions

- Frequency and voltage scaling (50 occurrences)
- Use power efficient library/device (45 occurrences)
- Disabling features or devices (42 occurrences)
- Energy bug fix (26 occurrences)
- Low power idling (22 occurrences)
- Timing out (16 occurrences)

RQ1: Solutions

- Frequency and voltage scaling (50 occurrences)

Reduce Wifi voltage for power savings Browse files

Should be beneficial for a wifi only device

10.1

Metallice authored on Jun 27, 2012 1 parent 3cee2da commit 34f4738019a9e04c882ffa25bed1b40b81017c51

Showing 1 changed file with 1 addition and 1 deletion. Unified Split

2 arch/arm/mach-omap2/board-espresso-wifi.c View

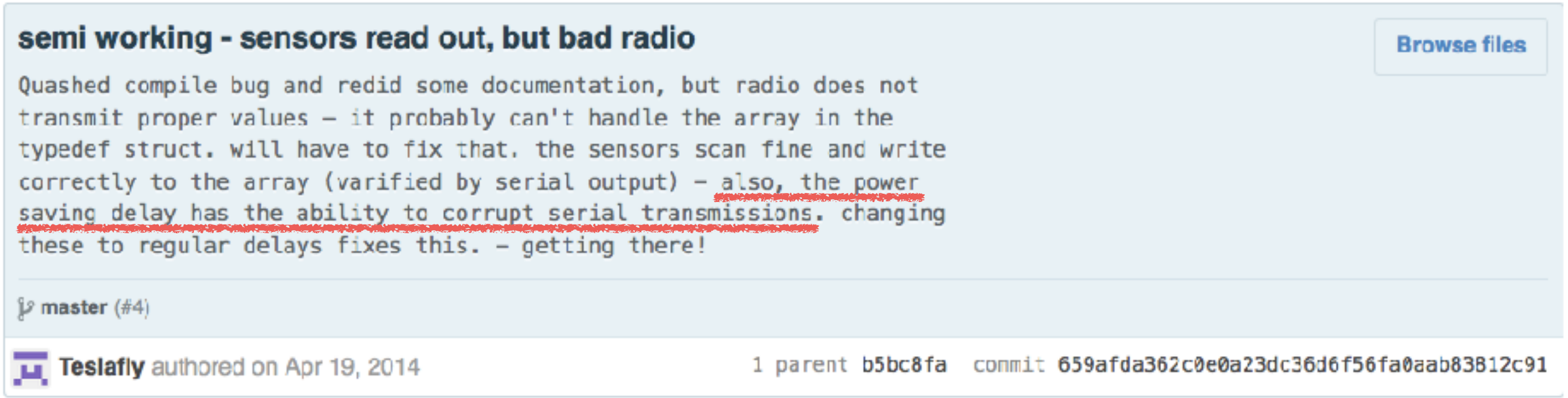
```
@@ -230,7 +230,7 @@ static struct regulator_init_data espresso_vmmc5 = {
230 230
231 231  static struct fixed_voltage_config espresso_vwlan = {
232 232          .supply_name            = "vwl1271",
233     -        .microvolts             = 2000200, /* 2.0V */
    233 +        .microvolts             = 1800200, /* 2.0V */ /*1.8V Metallice*/
234 234          .startup_delay          = 70000, /* 70msec */
235 235          .enable_high            = 1,
236 236          .enabled_at_boot        = 0,
```

RQ2: Quality Attributes

- Correctness (7 occurrences)
- Responsiveness (6 occurrences)
- Performance (3 occurrences)
- No actual power saving (3 occurrences)
- Miscellaneous (3 occurrences)

RQ2: Quality Attributes

- Correctness (7 occurrences)



semi working - sensors read out, but bad radio

Browse files

```
Quashed compile bug and redid some documentation, but radio does not
transmit proper values - it probably can't handle the array in the
typedef struct. will have to fix that. the sensors scan fine and write
correctly to the array (varified by serial output) - also, the power
saving delay has the ability to corrupt serial transmissions. changing
these to regular delays fixes this. - getting there!
```

master (#4)

Teslafly authored on Apr 19, 2014 — 1 parent b5bc8fa commit 659afda362c0e0a23dc36d6f56fa0aab83812c91

RQ2: Quality Attributes

- Responsiveness (6 occurrences)

cpufreq: boost the sampling rate on touch event Browse files

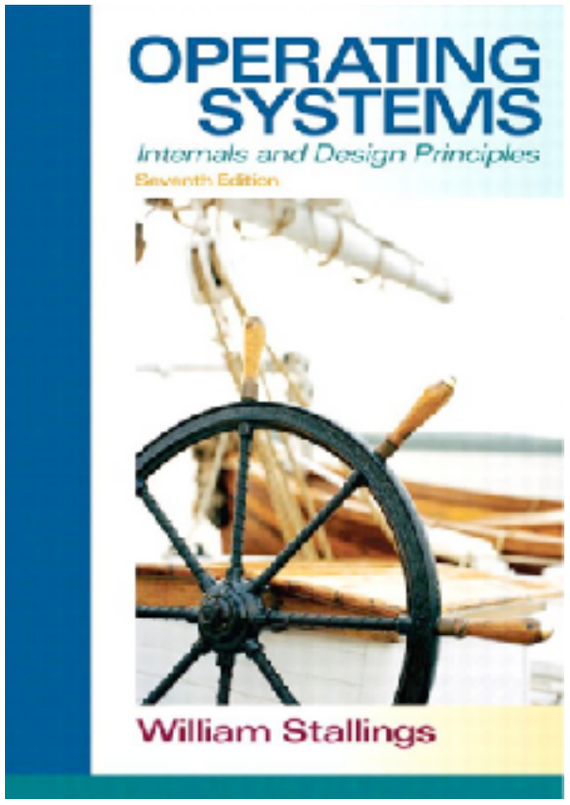
```
For better Ux responsiveness ondemand sampling rate
needs to be 20ms. But, a 20ms sampling rate increases power
consumption. So, boost the sampling rate to 20ms on every
touch event for 2.5 ms and later reset to default rate.
Also, change sampling down factor proportional to the
sampling rate.

Change-Id: I111b1cf3b8ed133347149afc34d329d0384ecfcb
Signed-off-by: Narayanan Gopalakrishnan <nargop@codeaurora.org>
```

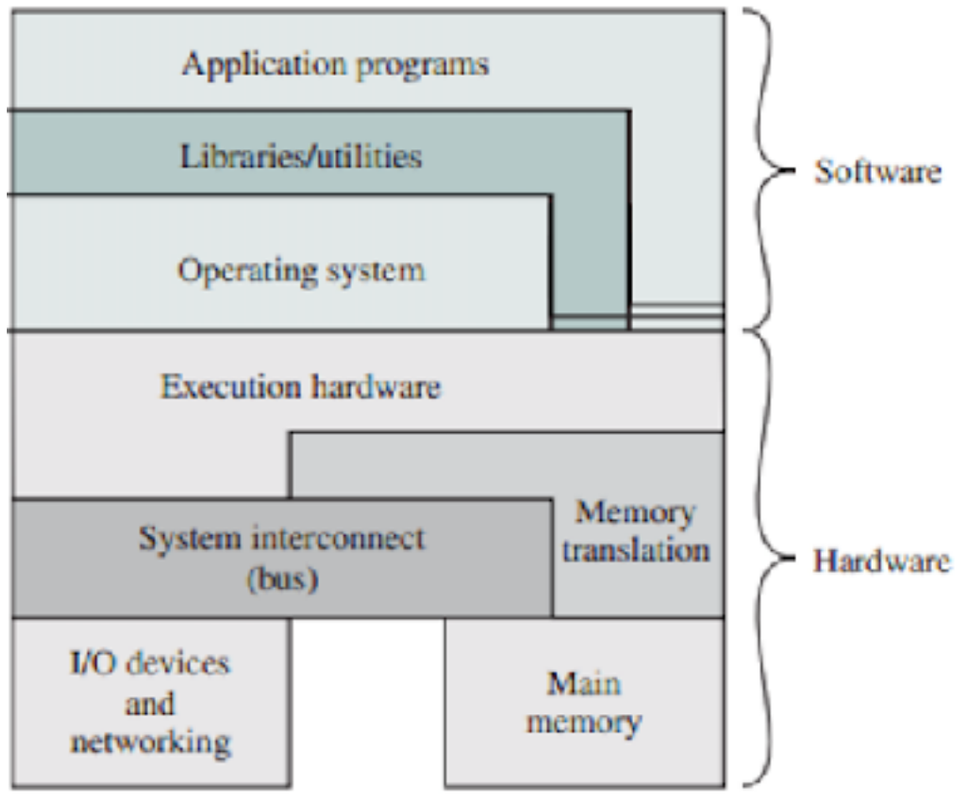
cm-10.1_3.0 + jellybean + jellybean-stable + mr1-staging

Narayanan Gopalakrishnan authored on Aug 9, 2012 — 1 parent b56043a commit 694447e65dd4af2af7193021ab67c4cad91ff412

intervigilium committed on Oct 5, 2012

RQ3: Software Stack

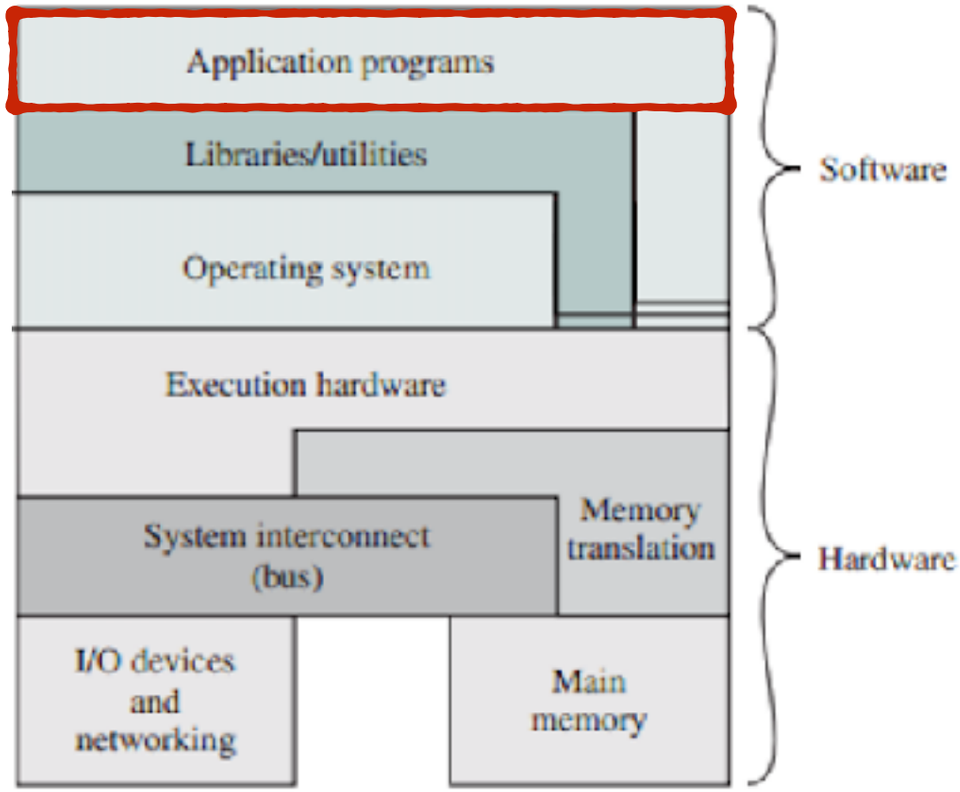


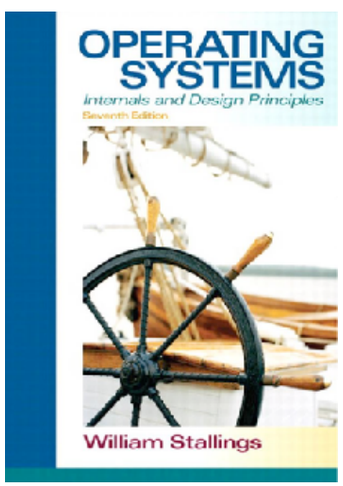


RQ3: Software Stack

88 Commits

Application includes embedded applications, desktop application, and mobile applications.



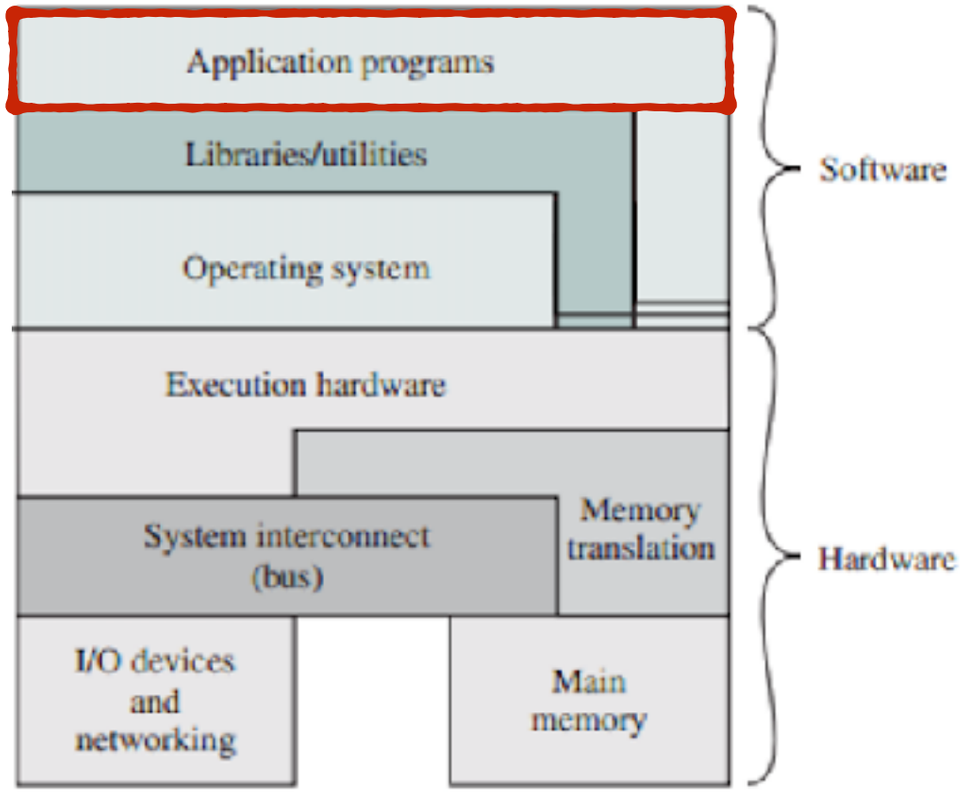


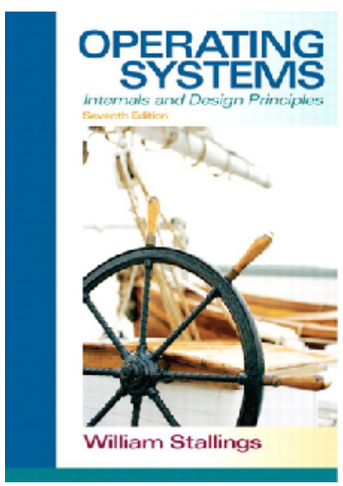
RQ3: Software Stack

88 Commits

42 Embedded

Application includes embedded applications, desktop application, and mobile applications.





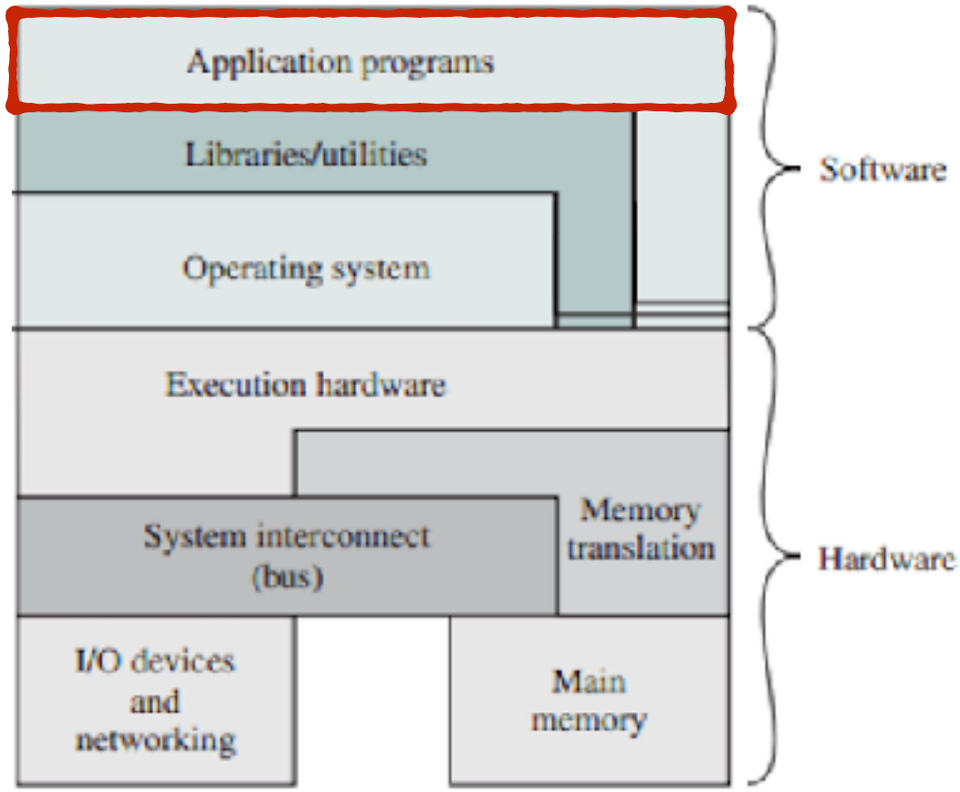
RQ3: Software Stack

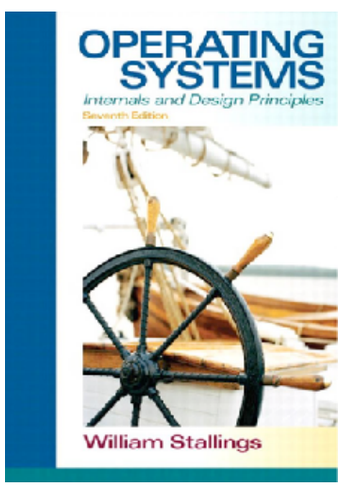
88 Commits

42 Embedded

21 Arduino

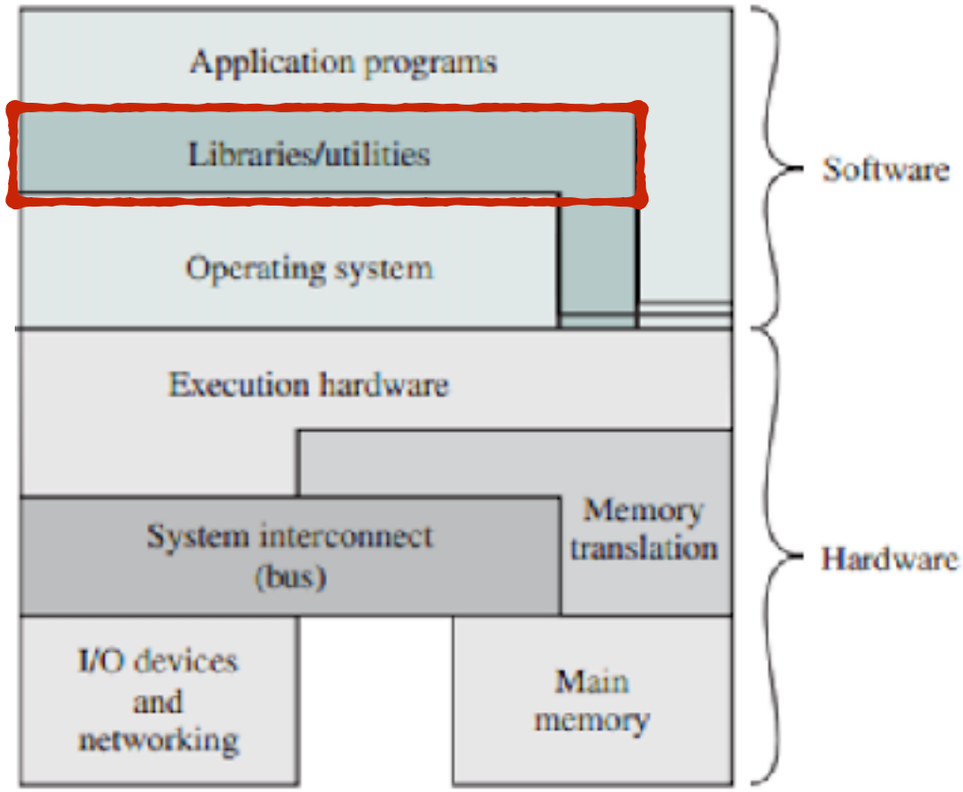
Application includes embedded applications, desktop application, and mobile applications.



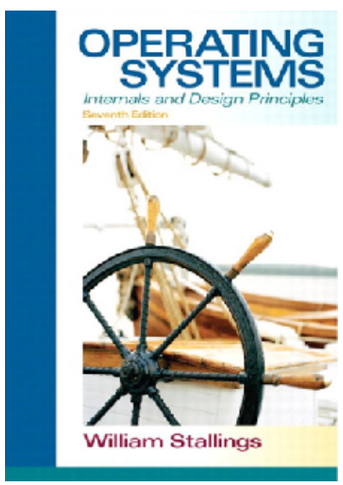


RQ3: Software Stack

50 Commits



Libraries/Utilities include scripts and embedded libraries.

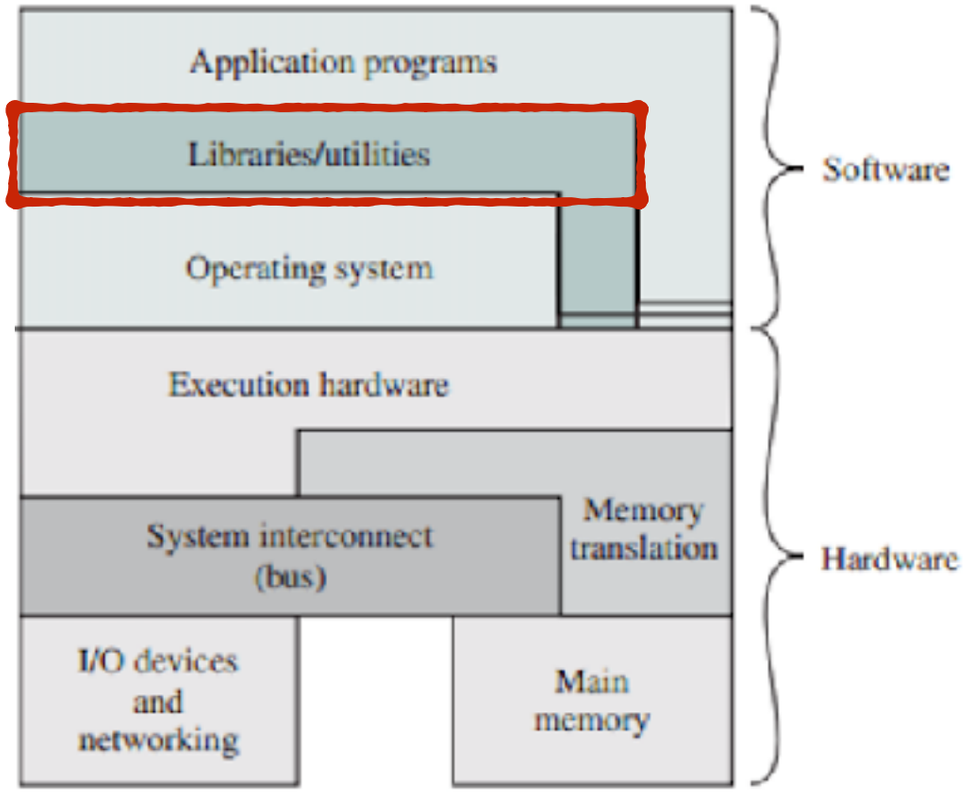


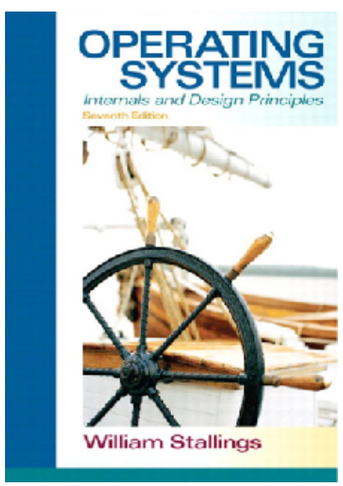
RQ3: Software Stack

50 Commits

41 scripts

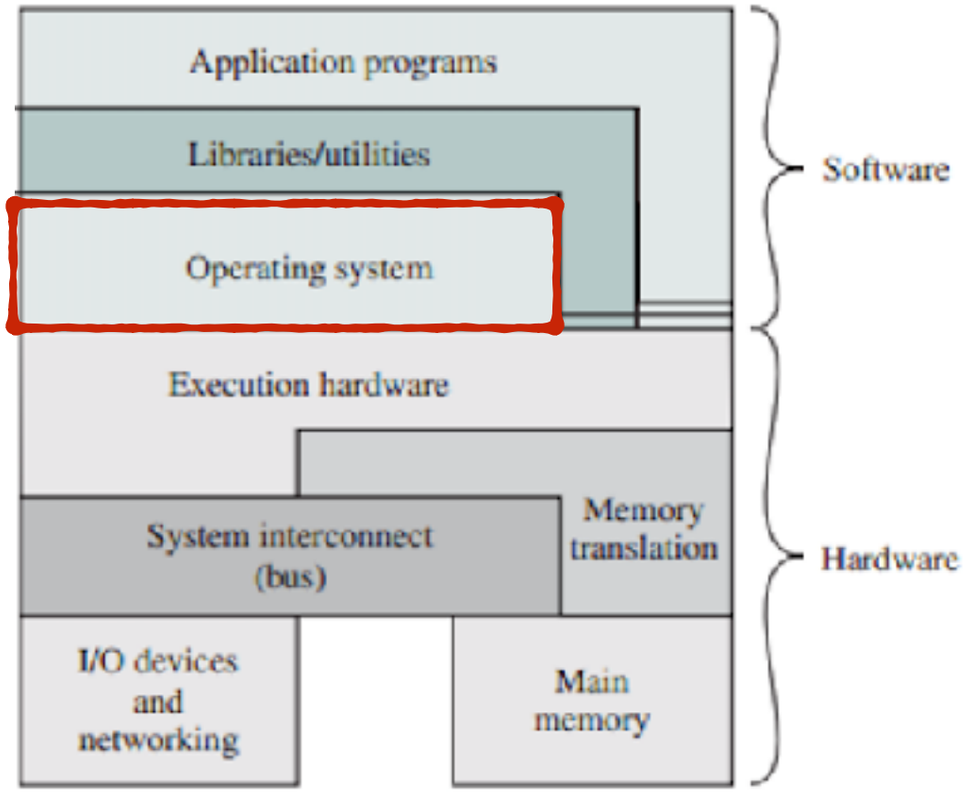
Libraries/Utilities include scripts and embedded libraries.



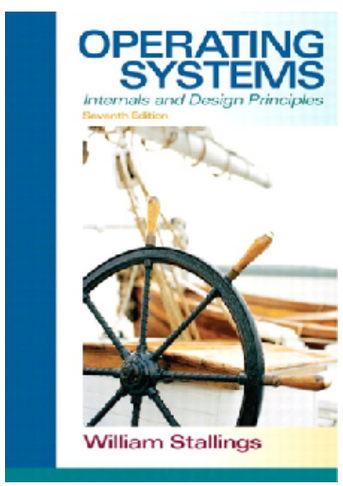


RQ3: Software Stack

142 Commits



Operating System includes Kernels, Embedded Kernels, Drivers and Firmwares

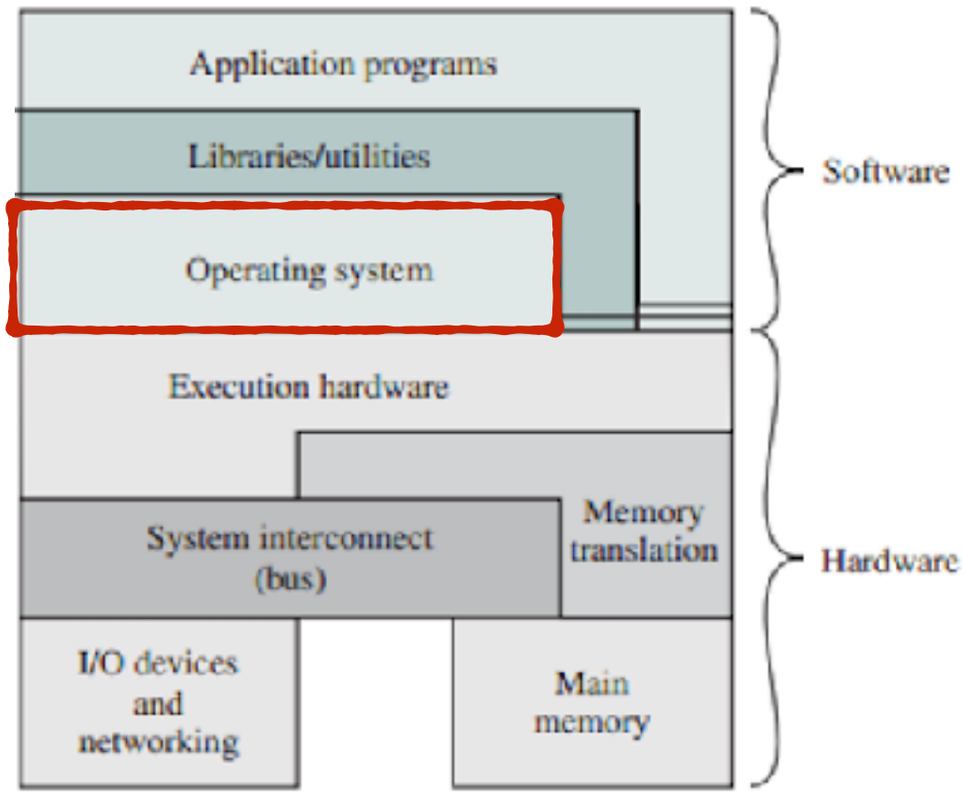


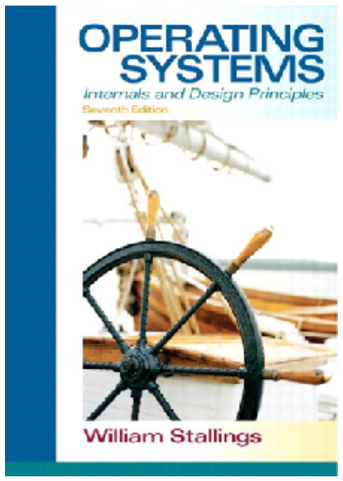
RQ3: Software Stack

142 Commits

69 Kernels

Operating System includes Kernels, Embedded Kernels, Drivers and Firmwares





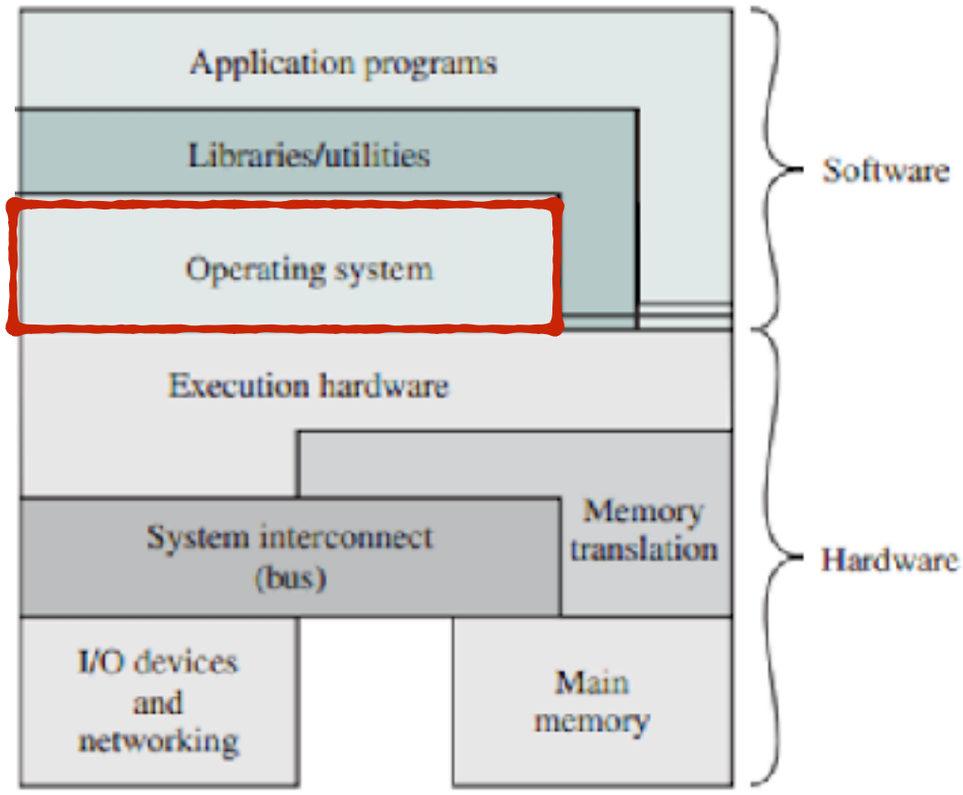
RQ3: Software Stack

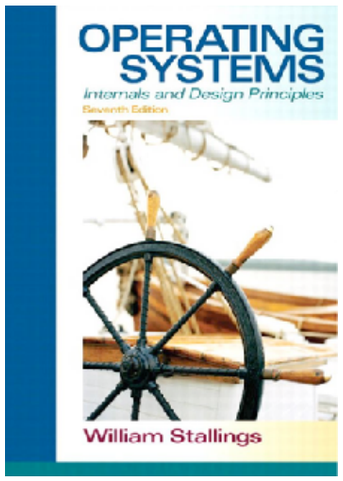
142 Commits

69 — OS Kernel

54 — Drivers

Operating System includes Kernels, Embedded Kernels, Drivers and Firmwares



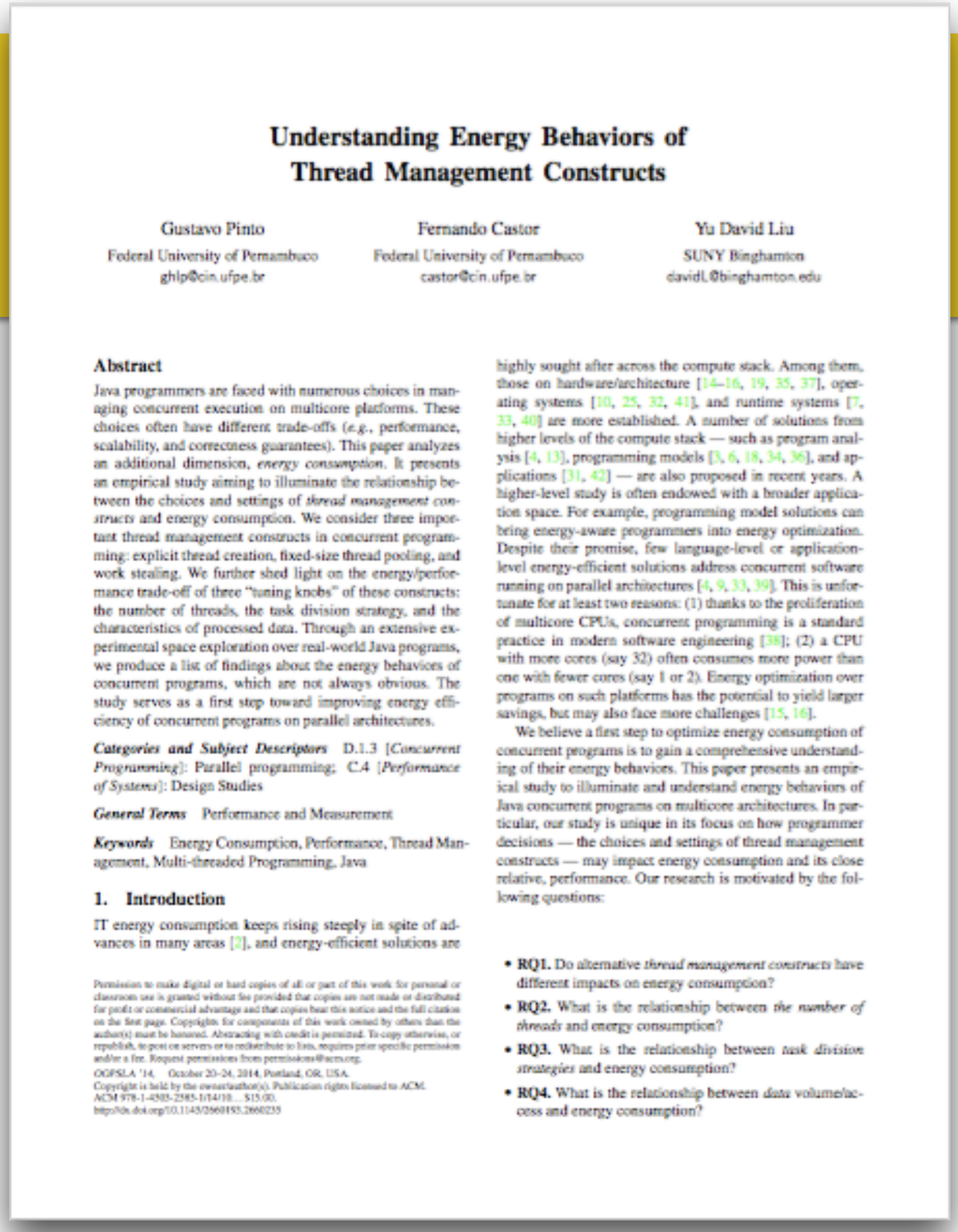


2014

- **Explicit threading (the Thread-style):** Using the *java.lang.Thread* class

- **Thread pooling (the Executor-style):** Using the *java.util.concurrent.Executor* framework

- **Working Stealing (the ForkJoin-style):** Using the *java.util.concurrent.ForkJoin* framework



Understanding Energy Behaviors of Thread Management Constructs

Gustavo Pinto
Federal University of Pernambuco
ghlp@cin.ufpe.br

Fernando Castor
Federal University of Pernambuco
castor@cin.ufpe.br

Yu David Liu
SUNY Binghamton
davidL@binghamton.edu

Abstract

Java programmers are faced with numerous choices in managing concurrent execution on multicore platforms. These choices often have different trade-offs (*e.g.*, performance, scalability, and correctness guarantees). This paper analyzes an additional dimension, *energy consumption*. It presents an empirical study aiming to illuminate the relationship between the choices and settings of *thread management constructs* and energy consumption. We consider three important thread management constructs in concurrent programming: explicit thread creation, fixed-size thread pooling, and work stealing. We further shed light on the energy/performance trade-off of three "tuning knobs" of these constructs: the number of threads, the task division strategy, and the characteristics of processed data. Through an extensive experimental space exploration over real-world Java programs, we produce a list of findings about the energy behaviors of concurrent programs, which are not always obvious. The study serves as a first step toward improving energy efficiency of concurrent programs on parallel architectures.

Categories and Subject Descriptors D.1.3 [*Concurrent Programming*]: Parallel programming; C.4 [*Performance of Systems*]: Design Studies

General Terms Performance and Measurement

Keywords Energy Consumption, Performance, Thread Management, Multi-threaded Programming, Java

1. Introduction

IT energy consumption keeps rising steeply in spite of advances in many areas [2], and energy-efficient solutions are highly sought after across the compute stack. Among them, those on hardware/architecture [14–16, 19, 35, 37], operating systems [10, 25, 32, 41], and runtime systems [7, 33, 40] are more established. A number of solutions from higher levels of the compute stack — such as program analysis [4, 13], programming models [3, 6, 18, 34, 36], and applications [31, 42] — are also proposed in recent years. A higher-level study is often endowed with a broader application space. For example, programming model solutions can bring energy-aware programmers into energy optimization. Despite their promise, few language-level or application-level energy-efficient solutions address concurrent software running on parallel architectures [4, 9, 33, 39]. This is unfortunate for at least two reasons: (1) thanks to the proliferation of multicore CPUs, concurrent programming is a standard practice in modern software engineering [38]; (2) a CPU with more cores (say 32) often consumes more power than one with fewer cores (say 1 or 2). Energy optimization over programs on such platforms has the potential to yield larger savings, but may also face more challenges [15, 16].

We believe a first step to optimize energy consumption of concurrent programs is to gain a comprehensive understanding of their energy behavior. This paper presents an empirical study to illuminate and understand energy behaviors of Java concurrent programs on multicore architectures. In particular, our study is unique in its focus on how programmer decisions — the choices and settings of thread management constructs — may impact energy consumption and its close relative, performance. Our research is motivated by the following questions:

- **RQ1.** Do alternative *thread management constructs* have different impacts on energy consumption?
- **RQ2.** What is the relationship between *the number of threads* and energy consumption?
- **RQ3.** What is the relationship between *task division strategies* and energy consumption?
- **RQ4.** What is the relationship between *data* volume/access and energy consumption?

OOPSLA '14, October 20–24, 2014, Portland, OR, USA.

ACM 978-1-4503-2585-1/14/10…$15.00.
http://dx.doi.org/10.1145/2660193.2660235

OOPSLA'14

Benchmarks

- **Embarrassingly parallel:** spectralnorm, sunflow, n-queens
- **Leaning parallel:** xalan, knucleotide, tomcat
- **Leaning serial:** mandelbrot, largestImage
- **Embarrassingly serial:** h2

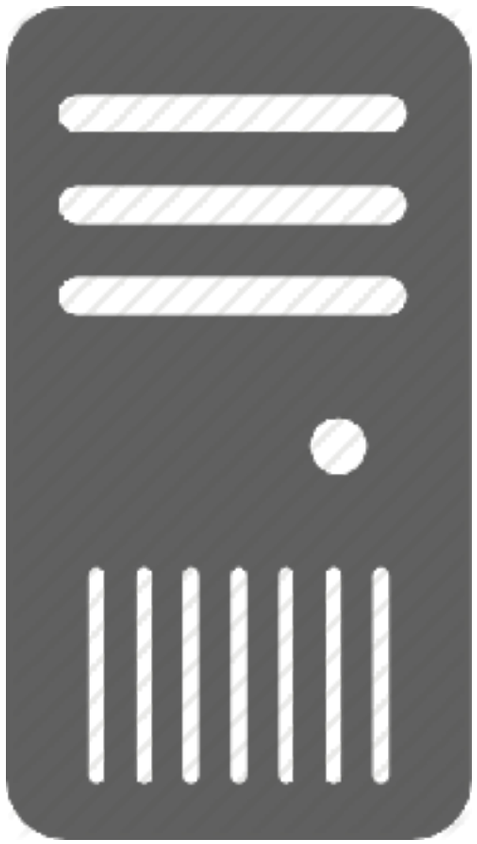
Benchmarks

- **Embarrassingly parallel:** spectralnorm, sunflow, n-queens
- **Leaning parallel:** xalan, knucleotide, tomcat
- **Leaning serial:** mandelbrot, largestImage
- **Embarrassingly serial:** h2

Micro-benchmarks

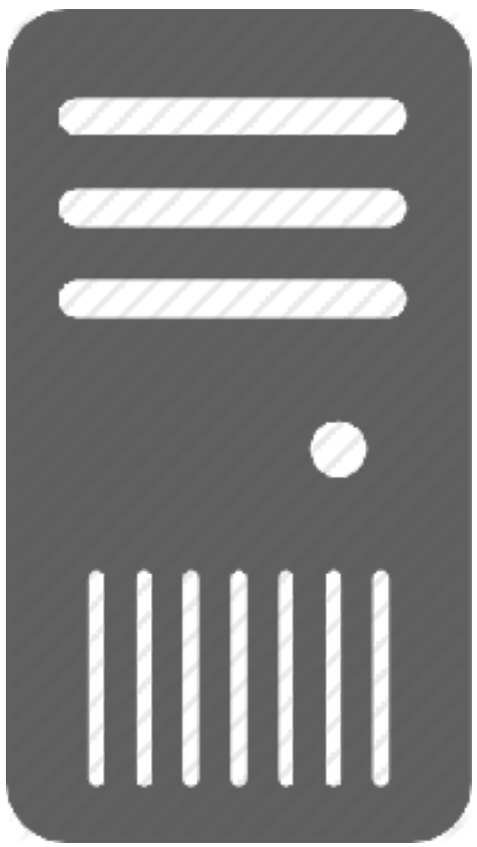
DaCapo benchmarks

Experimental Environment

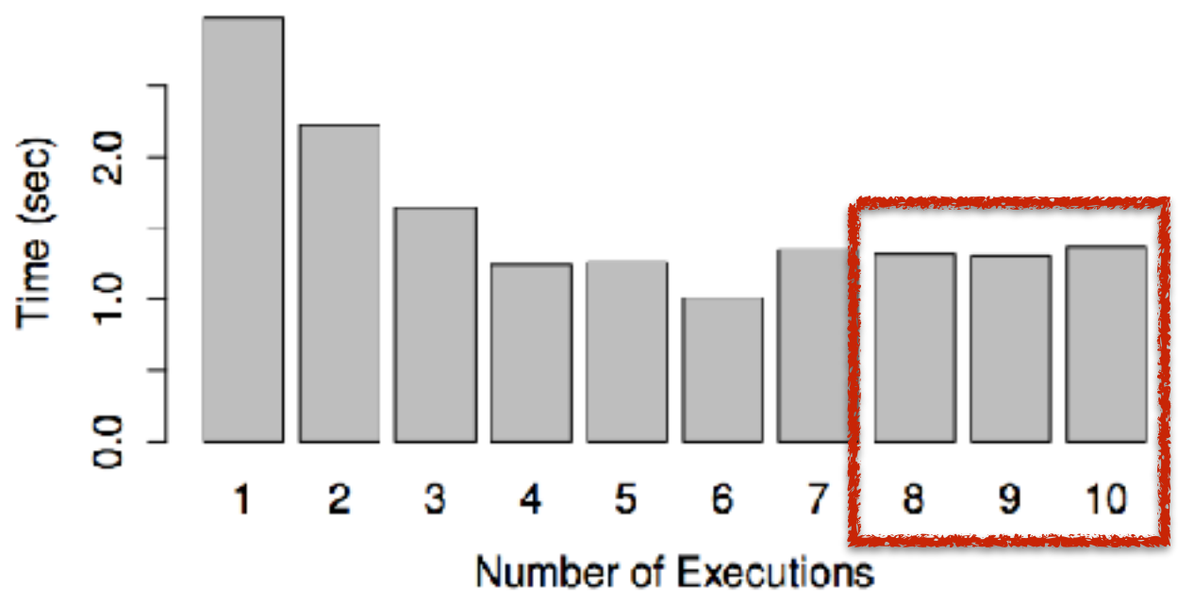


A 2×16-core AMD CPUs, running Debian Linux, 64GB of memory, JDK version 1.7.0 11, build 21, “ondemand” governor

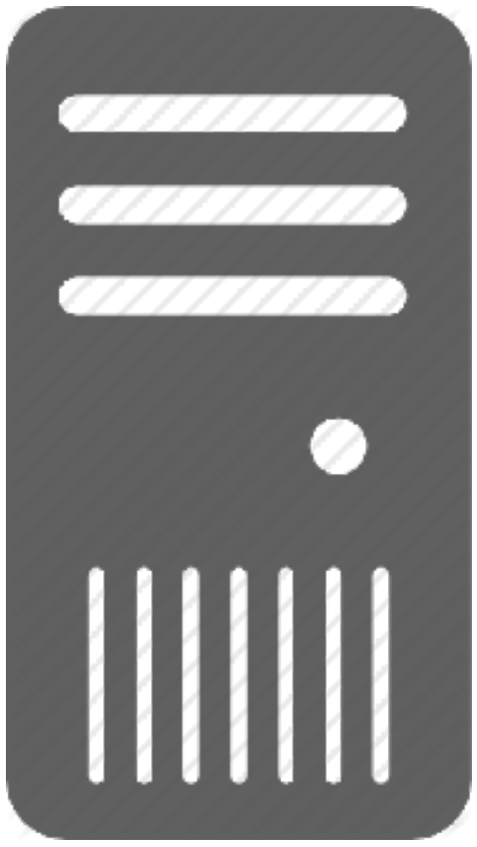
Experimental Environment



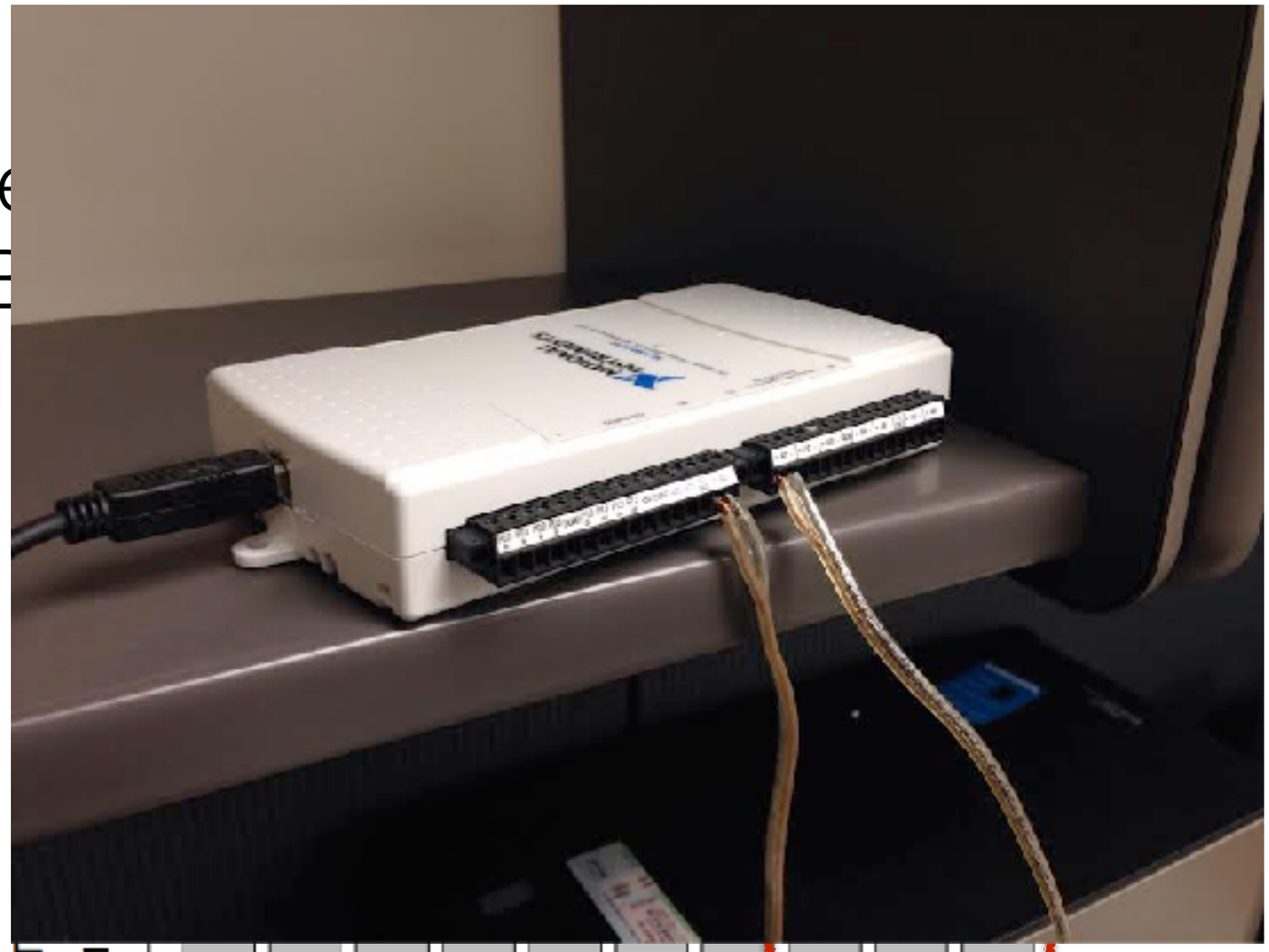
A 2×16-core AMD CPUs, running Debian Linux, 64GB of memory, JDK version 1.7.0 11, build 21, “ondemand” governor

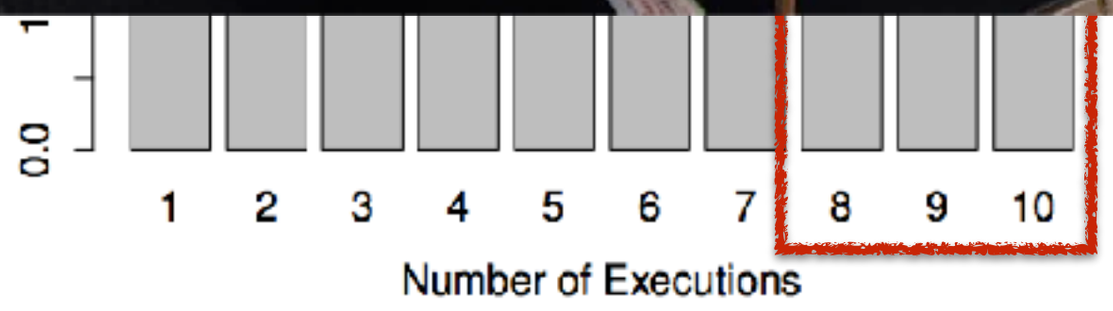


Experimental Environment

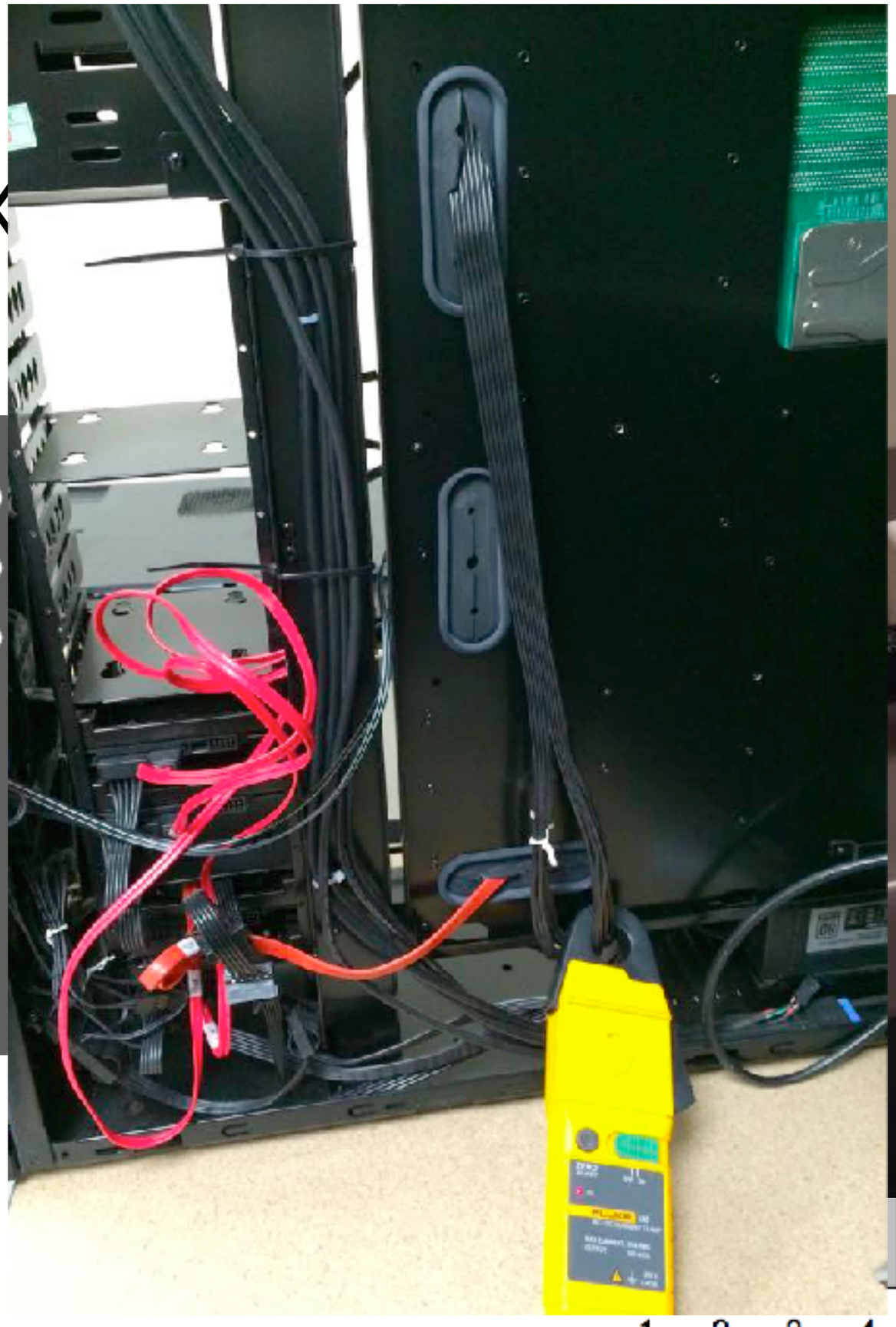


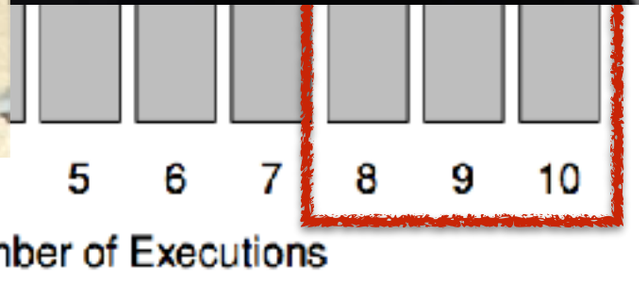
A 2×16-core
Linux, 64GB
11, build 21





Ex





Methodology

Starting from the Thread-style

```
class Main {
   int counter = 0; int coords[DATAN];
   void main() {
      for (int i = 0; i < THREADN; i++)
        (new Bucket()).start();
   }
   class Bucket extends Thread {
     ...
     public void run() { while(counter<DATAN){ dowork
          (); }}
     public void dowork() {
        int start;
        synchronized (Main.this) {
          if (counter >= DATAN) return;
          start=counter; counter+=DATAN/TASKN;
        }
        for (int j = 0; j < DATAN/TASKN; j++) {
          render(coords[start + j]);
        } //end for
}}}
```

Methodology

Then to the Executors-style

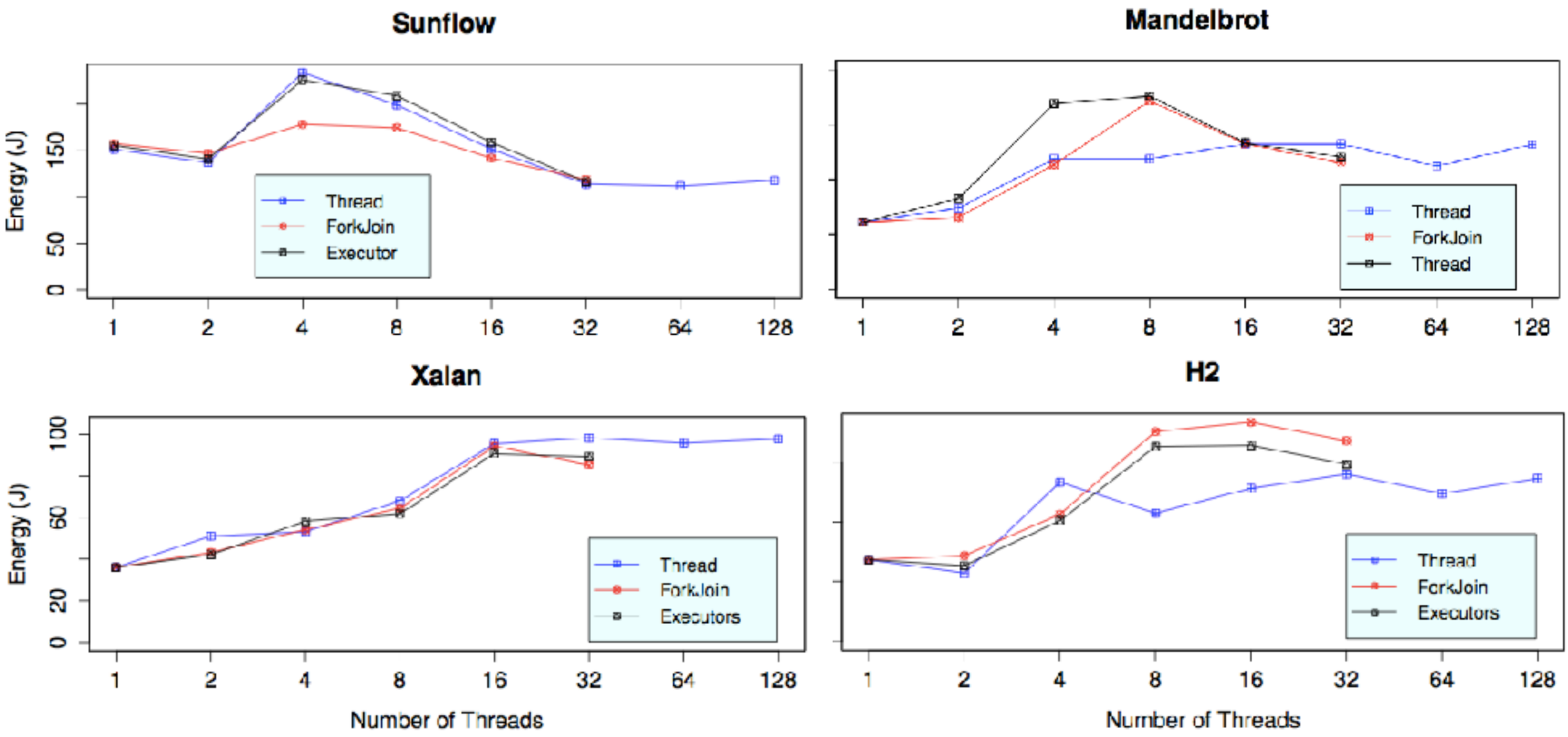
```
class Main {
   int counter = 0; int coords[DATAN];
   void main() {
      ExecutorService es = Executors.
           newFixedThreadPool(THREADN);
      for (int i = 0; i < TASKN; i++)
        es.execute(new Bucket());
   }
   class Bucket extends Thread {
     ...
     public void run() { dowork(); }
}}
```

Methodology

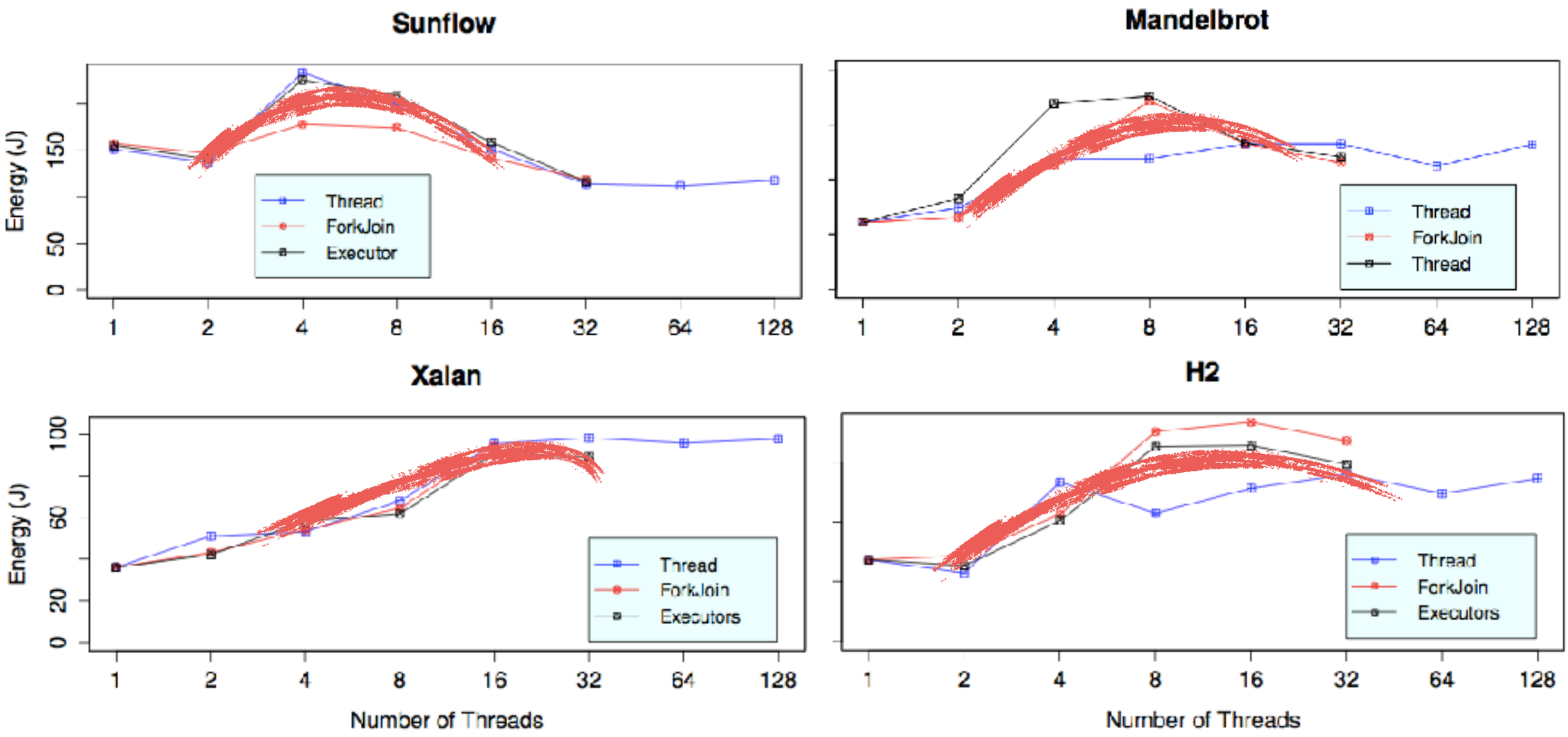
And finally, the ForkJoin-style

```
class Main {
   int counter = 0; int coords[DATAN];
   void main() {
       (new ForkJoinPool(THREADN)).submit(new Bucket
           (1));
   }
   class Bucket extends RecursiveAction {
     ...
     int taskcounter;
     Bucket(int taskcounter) {
        this.taskcounter = taskcounter;
     }
     public void compute() {
        if (taskcounter <= TASKN) {
            (new Bucket(taskcounter+1)).fork();
            dowork();
        }
}}}
```

Energy Consumption When Varying the Number of Threads



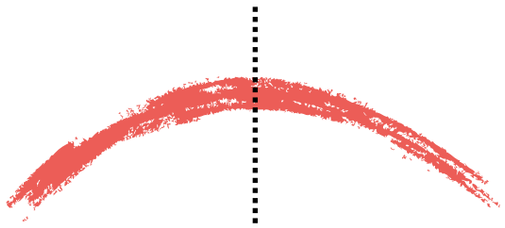
The Λ Curve


Sunflow
Energy (J)
0
50
150
Thread
ForkJoin
Executor
1
2
4
8
16
32
64
128
Mandelbrot
Thread
ForkJoin
Thread
Xalan
Energy (J)
0
20
60
100
Thread
ForkJoin
Executors
Number of Threads
H2
Thread
ForkJoin
Executors
Number of Threads

The Λ Curve

The Λ Curve

The Λ Curve



- More cores idle
- CPU frequency at a lower level

The Λ Curve

- More cores idle
- CPU frequency at a lower level

- More threads used, program completes sooner
- The greater the ratio between speedup and power, the steeper the \

Copying vs Sharing

```
import static Arrays.*;
class Task extends RecursiveAction{
  public Task (User[] u) {}
  protected void compute() {
    if (u.length < N) { local(u); }
    else {
      int split = u.length / 2;

      User[] u1 = copyOfRange(u, 0,
          split);
      User[] u2 = copyOfRange(u,
          split, u.length);

      invokeAll(new Task(u1),
                new Task(u2));
    }
  }
}
```

```
class Task extends RecursiveAction{
  public Task (User[] u, int from,
      int to) {}
  protected void compute() {
    if (to - from < N)
      local(u, from, to);
    else {
      int split = (from + to) / 2;

      invokeAll(
      new Task(u, from, split),
      new Task(u, from + split, to)
      );
    }
  }
}
```

Copying vs Sharing

```
import static Arrays.*;
class Task extends RecursiveAction{
  public Task (User[] u) {}
  protected void compute() {
    if (u.length < N) { local(u); }
    else {
      int split = u.length / 2;

      User[] u1 = copyOfRange(u, 0,
          split);
      User[] u2 = copyOfRange(u,
          split, u.length);

      invokeAll(new Task(u1),
                new Task(u2));
    }
  }
}
```

```
class Task extends RecursiveAction{
  public Task (User[] u, int from,
      int to) {}
  protected void compute() {
    if (to - from < N)
      local(u, from, to);
    else {
      int split = (from + to) / 2;

      invokeAll(
      new Task(u, from, split),
      new Task(u, from + split, to)
      );
    }
  }
}
```

Copying

Copying vs Sharing

```
import static Arrays.*;
class Task extends RecursiveAction{
  public Task (User[] u) {}
  protected void compute() {
    if (u.length < N) { local(u); }
    else {
      int split = u.length / 2;

      User[] u1 = copyOfRange(u, 0,
          split);
      User[] u2 = copyOfRange(u,
          split, u.length);

      invokeAll(new Task(u1),
                new Task(u2));
    }
  }
}
```

```
class Task extends RecursiveAction{
  public Task (User[] u, int from,
      int to) {}
  protected void compute() {
    if (to - from < N)
      local(u, from, to);
    else {
      int split = (from + to) / 2;

      invokeAll(
      new Task(u, from, split),
      new Task(u, from + split, to)
      );
    }
  }
}
```

Copying

Sharing

Copying vs Sharing

```
import static Arrays.*;
class Task extends RecursiveAction{
  public Task (User[] u) {}
  protected void compute() {
    if (u.length < N) { local(u); }
    else {
      int split = u.length / 2;

      User[] u1 = copyOfRange(u, 0,
          split);
      User[] u2 = copyOfRange(u,
          split, u.length);

      invokeAll(new Task(u1),
                new Task(u2));
    }
  }
}
```

```
class Task extends RecursiveAction{
  public Task (User[] u, int from,
      int to) {}
  protected void compute() {
    if (to - from < N)
      local(u, from, to);
    else {
      int split = (from + to) / 2;

      invokeAll(
      new Task(u, from, split),
      new Task(u, from + split, to)
      );
    }
  }
}
```

Copying

Sharing

±15% of energy savings!

2016

Bad programmers worry about the code. Good programmers worry about **data structures** and their relationships.

Linus Tolvards

A Comprehensive Study on the Energy Efficiency of Java's Thread-Safe Collections

Gustavo Pinto[1] Kenan Liu[2] Fernando Castor[3] Yu David Liu[2]
[1]UFPA, Brazil [2]SUNY Binghamton, USA [3]UFPE, Brazil

that each collection implements, *e.g.*, a collection implementation that performs traversals very efficiently can be more than an order of magnitude less efficient than another implementation of the same collection when it comes to insertions.

I. Introduction

A question that often arises in software development forums is: "since Java has so many collection implementations, which one should I use?"[1]. Answers to this question come in different flavors: these collections serve for different purposes and have different characteristics in terms of performance, scalability and thread-safety. In this study, we consider one additional attribute: *energy efficiency*. In an era where mobile platforms are prevalent, there is considerable evidence that battery usage is a key factor for evaluating and adopting mobile applications [1]. Energy consumption estimation tools do exist [2], [3], [4], but they do not provide direct guidance on *energy optimization*, *i.e.*, bridging the gap between understanding where energy is consumed and understanding how the code can be modified in order to reduce energy consumption.

Energy optimization is traditionally addressed by hardware-level (*e.g.*, [5], [6]) and system-level approaches (*e.g.*, [7], [8]). However, it has been gaining momentum in recent years through application-level software engineering techniques (*e.g.*, [9], [10], [11]). The overarching premise of this emerging direction is that the high-level knowledge from software engineers on application design and implementation can make significant impact on energy consumption, as confirmed

[1]http://stackoverflow.com/search?q=which+data+structure+use+java+ic question

cent empirical studies [12], [13], [14], [15]. Moreover, y efficiency, similarly to performance and reliability, is a nic property. Thus, it must be tackled across multiple of the system stack. The space for application-level y optimization, however, is diverse.

this paper, we elucidate the energy consumption of ent Java thread-safe collections running on parallel ar- tures. This is a critical direction at the junction of data- ive computing and parallel computing, which deserves investigation due to at least three reasons:

- Collections are one of the most important building blocks of computer programming. Multiplicity — a collection may hold many pieces of data items — is the norm of their use, and it often contributes to significant memory pressure, and performance problems in general, of modern applications where data are often intensive [16], [17].
- Not only high-end servers but also desktop machines, smartphones, and tablets need concurrent programs to make best use of their multi-core hardware. A CPU with more cores (say 32) often dissipates more power than one with fewer cores (say 1 or 2) [18]. In addition, in mobile platforms such as Android, due to responsiveness requirements, concurrency in the form of asynchronous operations is the norm [19].
- Mainstream programming languages often provide a number of implementations for the same collection and these implementations have potentially different characteristics in terms of energy efficiency [20], [21].

Through extensive experiments conducted in a multi-core environment, we correlate energy behaviors of 13 thread-safe implementations of Java collections, grouped by 3 well-known interfaces (`List`, `Set`, and `Map`), and their turning knobs. Our research is motivated by the following questions:

RQ1. Do different implementations of the same collection have different impacts on energy consumption?

RQ2. Do different operations in the same implementation of a collection consume energy differently?

RQ3. Do collections scale, from an energy consumption perspective, with an increasing number of concurrent threads?

RQ4. Do different collection configurations and usages have different impacts on energy consumption?

In order to answer **RQ1** and **RQ2**, we select and analyze the behaviors of three common operations — traversal, insertion and removal — for each collection implementation. To answer

ICSME'16

In case you are a Java programmer...

```
List<Object> lists = ...;
```

- **ArrayList**
- **LinkedList**

```
List<Object> lists = new ArrayList<>();
```

Thread 

```
List<Object> lists = new ArrayList<>();
```

Thread

List<Object> lists = ...;

Non Thread-Safe

- ArrayList
- LinkedList

Thread-Safe

- Vector
- Collections.synchronizedList()
- CopyOnWriteArrayList

```
List<Object> lists = new Vector<>();
```

Thread 

```
List<Object> lists = new Vector<>();
```

Thread 

Thread-safe!

List<Object> lists = new Vector<>();

Thread 

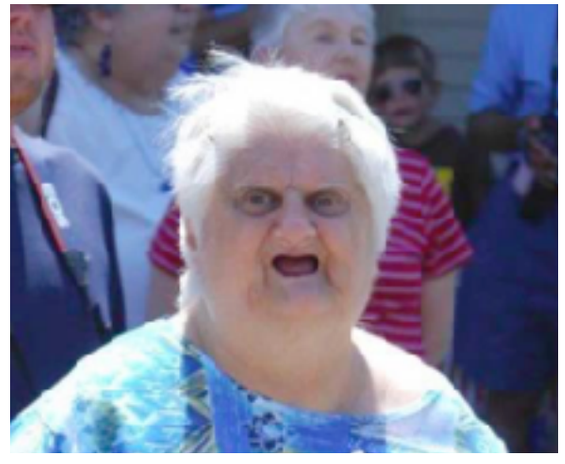
```
List<Object> lists = new Vector<>();
```

```
public class Vector<E> {
    public synchronized void addElement(E obj) {..
    public synchronized boolean removeElement(Object obj) {..
    public synchronized E get(int index) {..
    ...
}
```

Thread

```
List<Object> lists = new Vector<>();
```

```
public class Vector<E> {
    public synchronized void addElement(E obj) {..
    public synchronized boolean removeElement(Object obj) {..
    public synchronized E get(int index) {..
    ...
}
```



```
List<Object> lists = new CopyOnWriteArrayList<>();
```

Thread 

 Thread-safe!

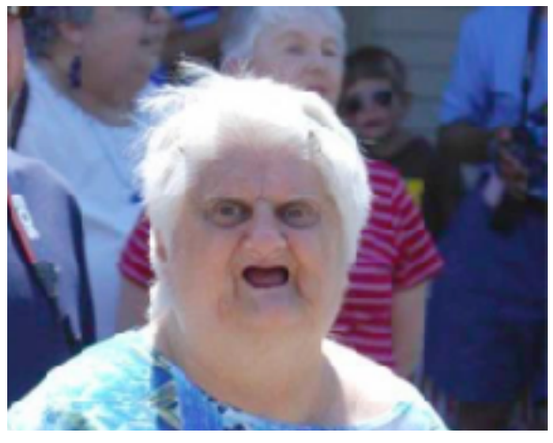
```
List<Object> lists = new CopyOnWriteArrayList<>();
```

Thread

Thread-safe!

List<Object> lists = new CopyOnWriteArrayList<>();

```
public class CopyOnWriteArrayList<E> {

    private E get(Object[] a, int index) {
        return (E) a[index];
    }

    public boolean add(E e) {
        final ReentrantLock lock = this.lock;
        lock.lock();
        try {
            Object[] elements = getArray();
            int len = elements.length;
            Object[] newElements = Arrays.copyOf(elements, len + 1);
            newElements[len] = e;
            setArray(newElements);
            return true;
        } finally {
            lock.unlock();
        }
    }

}
```



List<Object> lists = ...;

Non Thread-Safe

- ArrayList
- LinkedList

Thread-Safe

- Vector
- Collections.synchronizedList()
- CopyOnWriteArrayList

```
List<Object> lists = ...;

Set<Objects> sets = ...;

Map<Object, Object> maps = ...;
```

List<Object> lists = ...;

Set<Objects> sets = ...;

Map<Object, Object> maps = ...;

List<Object> lists = ...;

Set<Objects> sets = ...;

Map<Object, Object> maps = ...;

A Comprehensive Study on the Energy Efficiency of Java's Thread-Safe Collections

Gustavo Pinto[1] Kenan Liu[2] Fernando Castor[3] Yu David Liu[2]
[1]IFPA, Brazil [2]SUNY Binghamton, USA [3]UFPE, Brazil

***Abstract*—Java programmers are served with numerous choices of collections, varying from simple sequential ordered lists to sophisticated hashtable implementations. These choices are well-known to have different characteristics in terms of performance, scalability, and thread-safety, and most of them are well studied. This paper analyzes an additional dimension, *energy efficiency*. We conducted an empirical investigation of 16 collection implementations (13 thread-safe, 3 non-thread-safe) grouped under 3 commonly used forms of collections (lists, sets, and mappings). Using micro- and real world-benchmarks (TOMCAT and XALAN), we show that our results are meaningful and impactful. In general, we observed that simple design decisions can greatly impact energy consumption. In particular, we found that using a newer hashtable version can yield a 2.19x energy savings in the micro-benchmarks and up to 17% in the real world-benchmarks, when compared to the old associative implementation. Also, we observed that different implementations of the same thread-safe collection can have widely different energy consumption behaviors. This variation also applies to the different operations that each collection implements, *e.g.*, a collection implementation that performs traversals very efficiently can be more than an order of magnitude less efficient than another implementation of the same collection when it comes to insertions.**

I. INTRODUCTION

A question that often arises in software development forums is: "since Java has so many collection implementations, which one should I use?"[1]. Answers to this question come in different flavors: these collections serve for different purposes and have different characteristics in terms of performance, scalability and thread-safety. In this study, we consider one additional attribute: *energy efficiency*. In an era where mobile platforms are prevalent, there is considerable evidence that battery usage is a key factor for evaluating and adopting mobile applications [1]. Energy consumption estimation tools do exist [2], [3], [4], but they do not provide direct guidance on *energy optimization*, *i.e.*, bridging the gap between understanding where energy is consumed and understanding how the code can be modified in order to reduce energy consumption.

Energy optimization is traditionally addressed by hardware-level (*e.g.*, [5], [6]) and system-level approaches (*e.g.*, [7], [8]). However, it has been gaining momentum in recent years through application-level software engineering techniques (*e.g.*, [9], [10], [11]). The overarching premise of this emerging direction is that the high-level knowledge from software engineers on application design and implementation can make significant impact on energy consumption, as confirmed by recent empirical studies [12], [13], [14], [15]. Moreover, energy efficiency, similarly to performance and reliability, is a systemic property. Thus, it must be tackled across multiple levels of the system stack. The space for application-level energy optimization, however, is diverse.

In this paper, we elucidate the energy consumption of different Java thread-safe collections running on parallel architectures. This is a critical direction at the junction of data-intensive computing and parallel computing, which deserves more investigation due to at least three reasons:

- Collections are one of the most important building blocks of computer programming. Multiplicity — a collection may hold many pieces of data items — is the norm of their use, and it often contributes to significant memory pressure, and performance problems in general, of modern applications where data are often intensive [16], [17].
- Not only high-end servers but also desktop machines, smartphones, and tablets need concurrent programs to make best use of their multi-core hardware. A CPU with more cores (say 32) often dissipates more power than one with fewer cores (say 1 or 2) [18]. In addition, in mobile platforms such as Android, due to responsiveness requirements, concurrency in the form of asynchronous operations is the norm [19].
- Mainstream programming languages often provide a number of implementations for the same collection and these implementations have potentially different characteristics in terms of energy efficiency [20], [21].

Through extensive experiments conducted in a multi-core environment, we correlate energy behaviors of 13 thread-safe implementations of Java collections, grouped by 3 well-known interfaces (`List`, `Set`, and `Map`), and their turning knobs. Our research is motivated by the following questions:

RQ1. Do different implementations of the same collection have different impacts on energy consumption?

RQ2. Do different operations in the same implementation of a collection consume energy differently?

RQ3. Do collections scale, from an energy consumption perspective, with an increasing number of concurrent threads?

RQ4. Do different collection configurations and usages have different impacts on energy consumption?

In order to answer **RQ1** and **RQ2**, we select and analyze the behaviors of three common operations — traversal, insertion and removal — for each collection implementation. To answer

[1]http://stackoverflow.com/search?q=which+data+structure+use+java+ic question

16 Collections

List
ArrayList
Vector
Collections.syncList()
CopyOnWriteArrayList

Set
LinkedHashSet
Collections.syncSet()
CopyOnWriteArraySet
ConcurrentSkipListSet
ConcurrentHashSet
ConcurrentHashSetV8

Map
LinkedHashMap
Hashtable
Collections.syncMap()
ConcurrentSkipListMap
ConcurrentHashMap
ConcurrentHashMapV8

16 Collections

List
ArrayList
Vector
Collections.syncList()
CopyOnWriteArrayList

Set
LinkedHashSet
Collections.syncSet()
CopyOnWriteArraySet
ConcurrentSkipListSet
ConcurrentHashSet
ConcurrentHashSetV8

Map
LinkedHashMap
Hashtable
Collections.syncMap()
ConcurrentSkipListMap
ConcurrentHashMap
ConcurrentHashMapV8

Non thread-safe

Thread-safe

16 Collections

List
ArrayList
Vector
Collections.syncList()
CopyOnWriteArrayList

Set
LinkedHashSet
Collections.syncSet()
CopyOnWriteArraySet
ConcurrentSkipListSet
ConcurrentHashSet
ConcurrentHashSetV8

Map
LinkedHashMap
Hashtable
Collections.syncMap()
ConcurrentSkipListMap
ConcurrentHashMap
ConcurrentHashMapV8

Java 7

Java 8

16 Collections

List
ArrayList
Vector
Collections.syncList()
CopyOnWriteArrayList

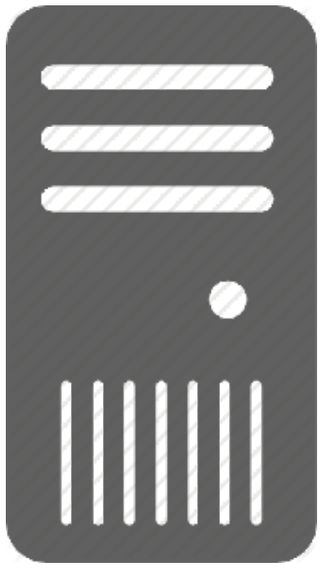
Set
LinkedHashSet
Collections.syncSet()
CopyOnWriteArraySet
ConcurrentSkipListSet
ConcurrentHashSet
ConcurrentHashSetV8

Map
LinkedHashMap
Hashtable
Collections.syncMap()
ConcurrentSkipListMap
ConcurrentHashMap
ConcurrentHashMapV8

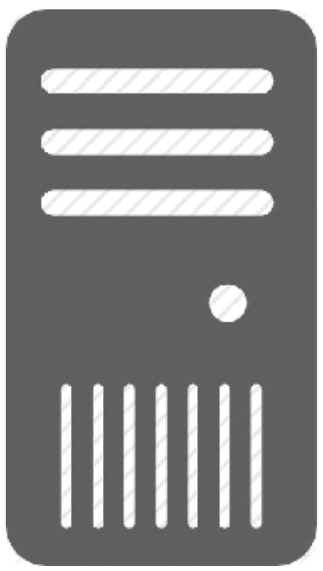
x 3 Operations

Traversal	Insertion	Removal

2 Environments

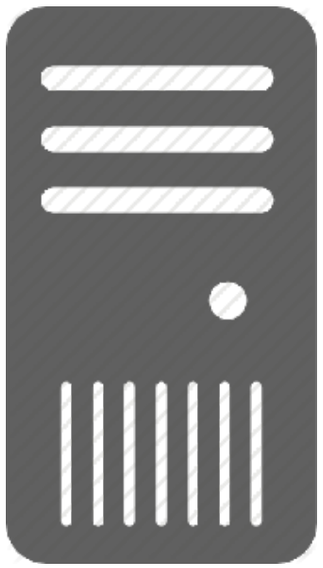


AMD CPU: A 2×16-core, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0_11, build 21.



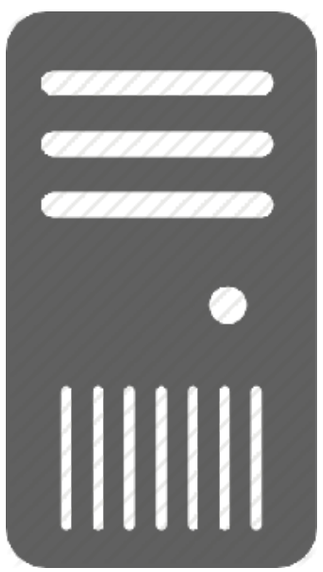
Intel CPU: A 2×8-core (32-cores w/ hyper-threading), running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0_71, build 14.

2 Environments



AMD CPU: A 2×16-core, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0 11, build 21.

Hardware-based energy measurement

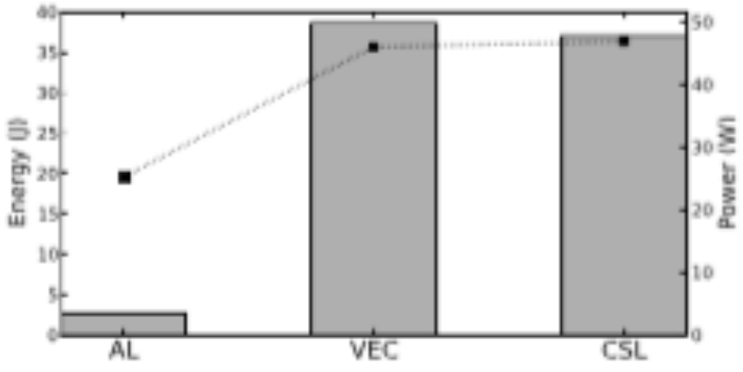


Intel CPU: A 2×8-core (32-cores w/ hyper-threading), running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0 71, build 14.

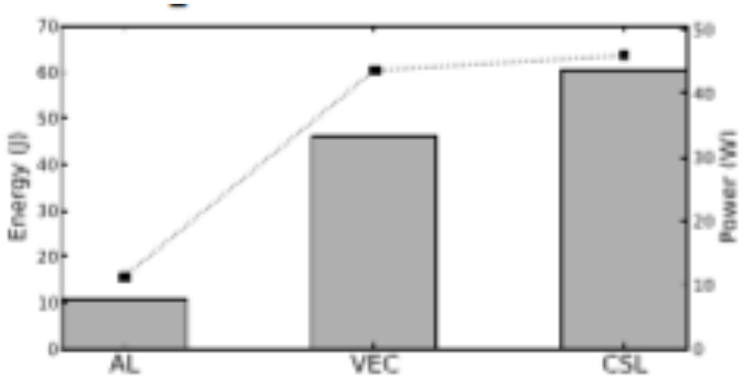
Software-based energy measurement

Lists

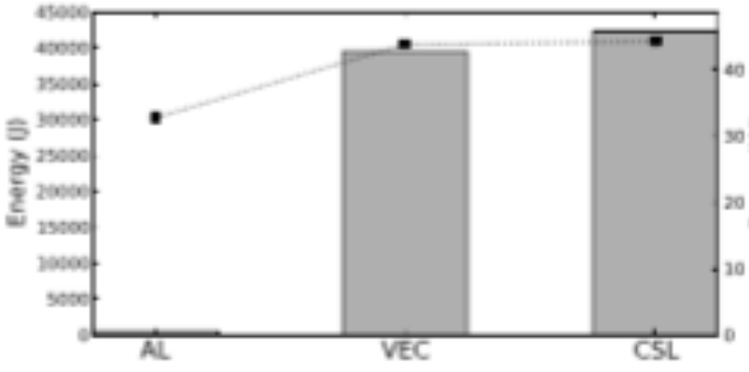
AMD CPU



Traversal

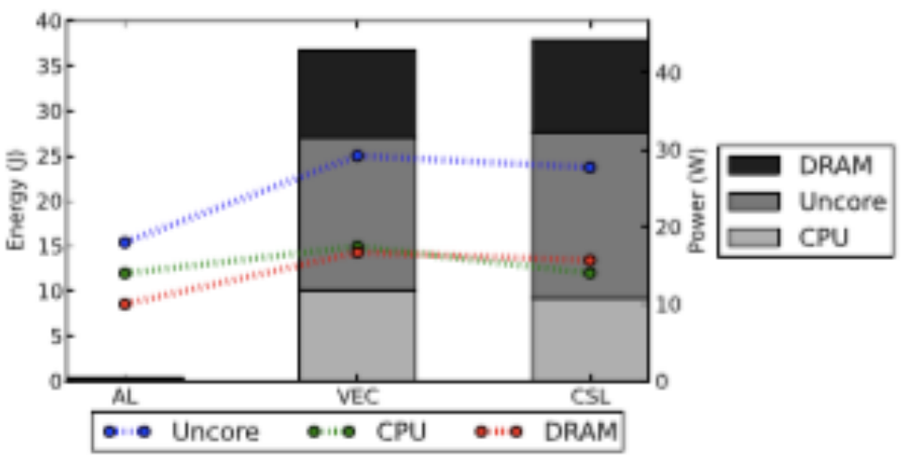


Insertion

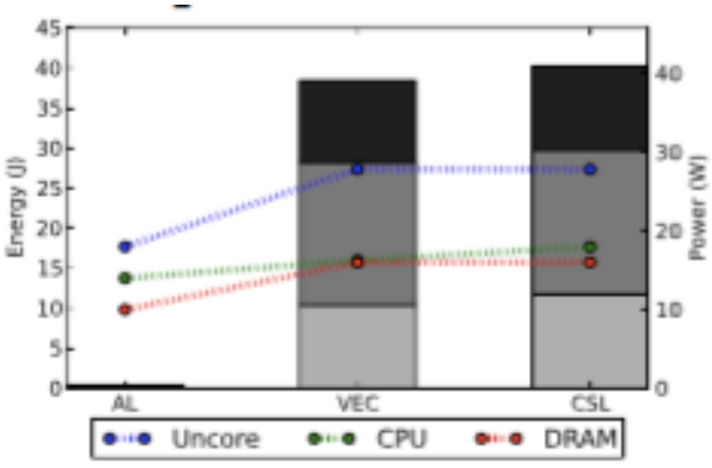


Removal

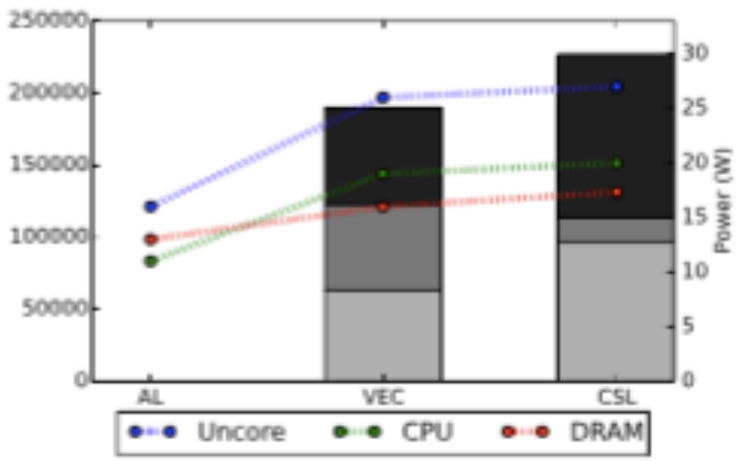
Intel CPU



Traversal

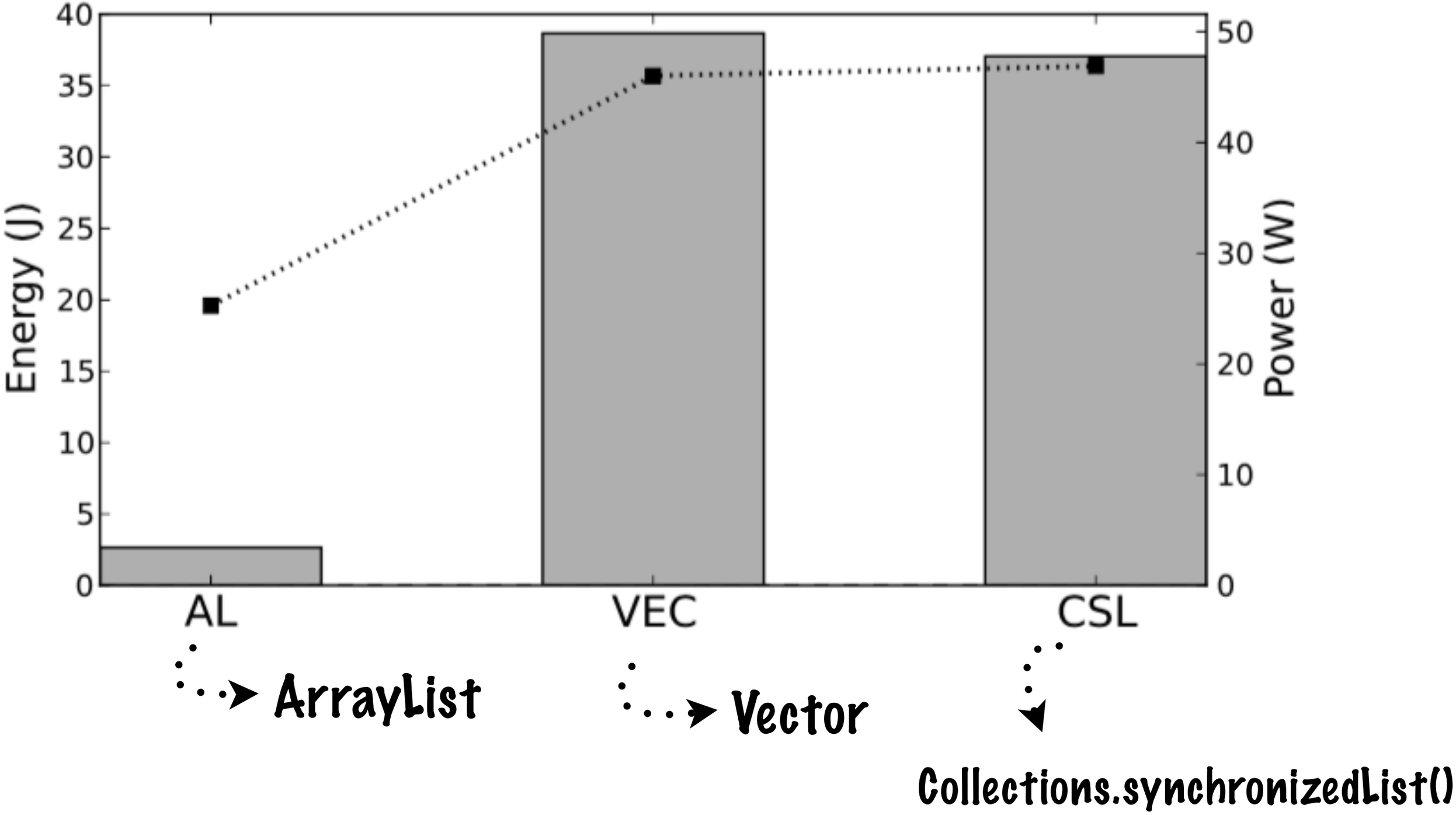


Insertion

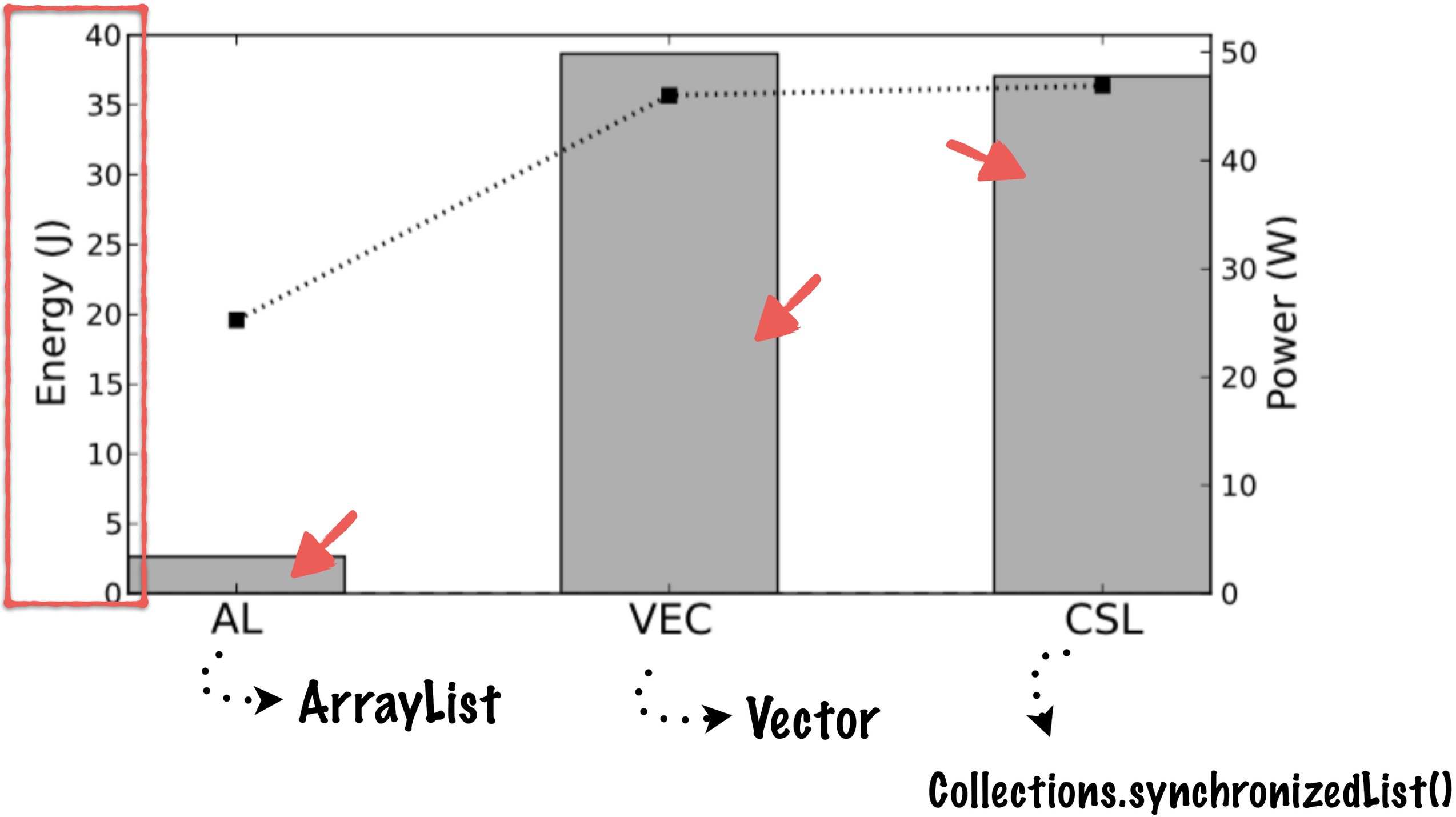


Removal

Lists

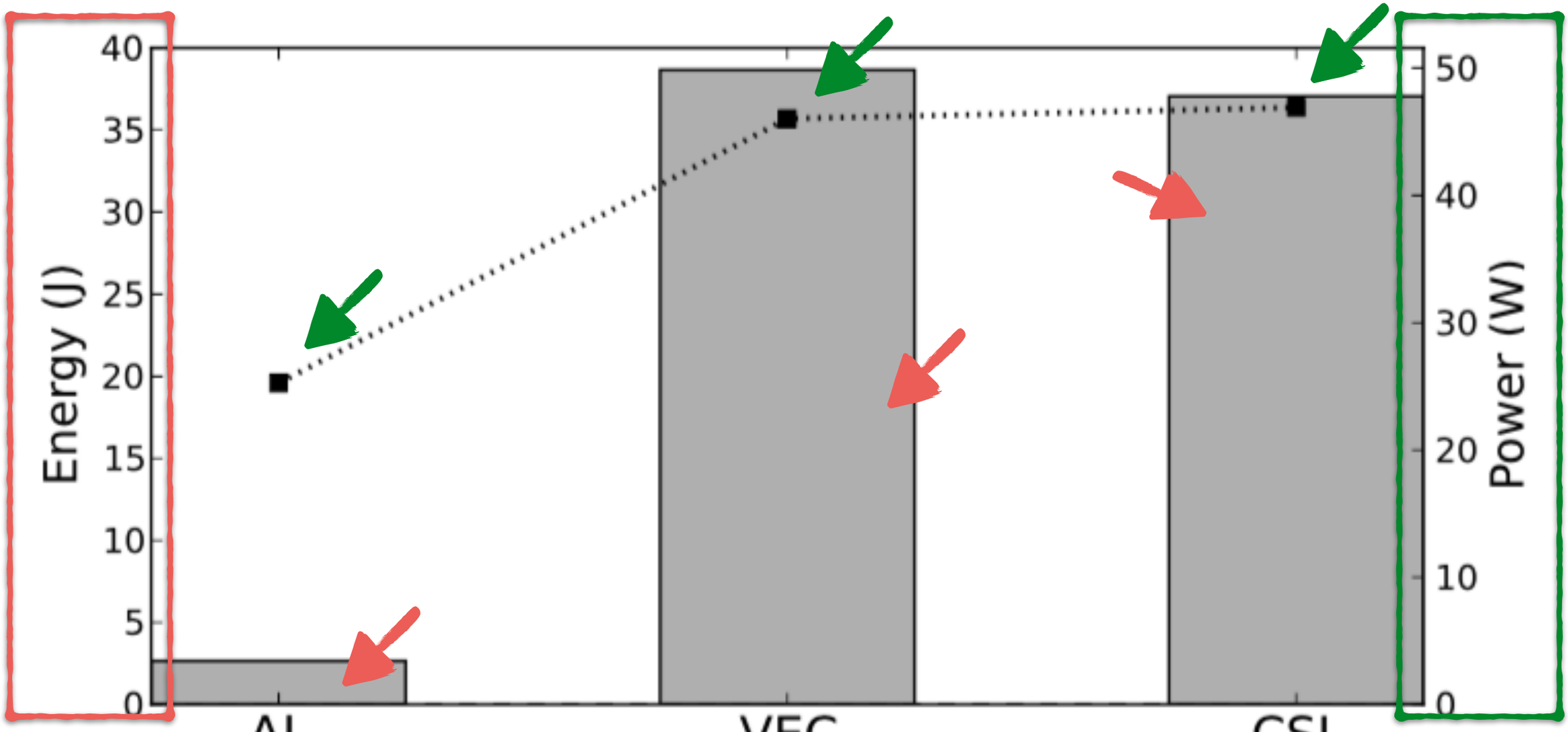

Energy (J)
Power (W)
0
5
10
15
20
25
30
35
40
0
10
20
30
40
50
AL
VEC
CSL
ArrayList
Vector
Collections.synchronizedList()

Energy (Joules)



Energy (Joules)

Power (Watts)



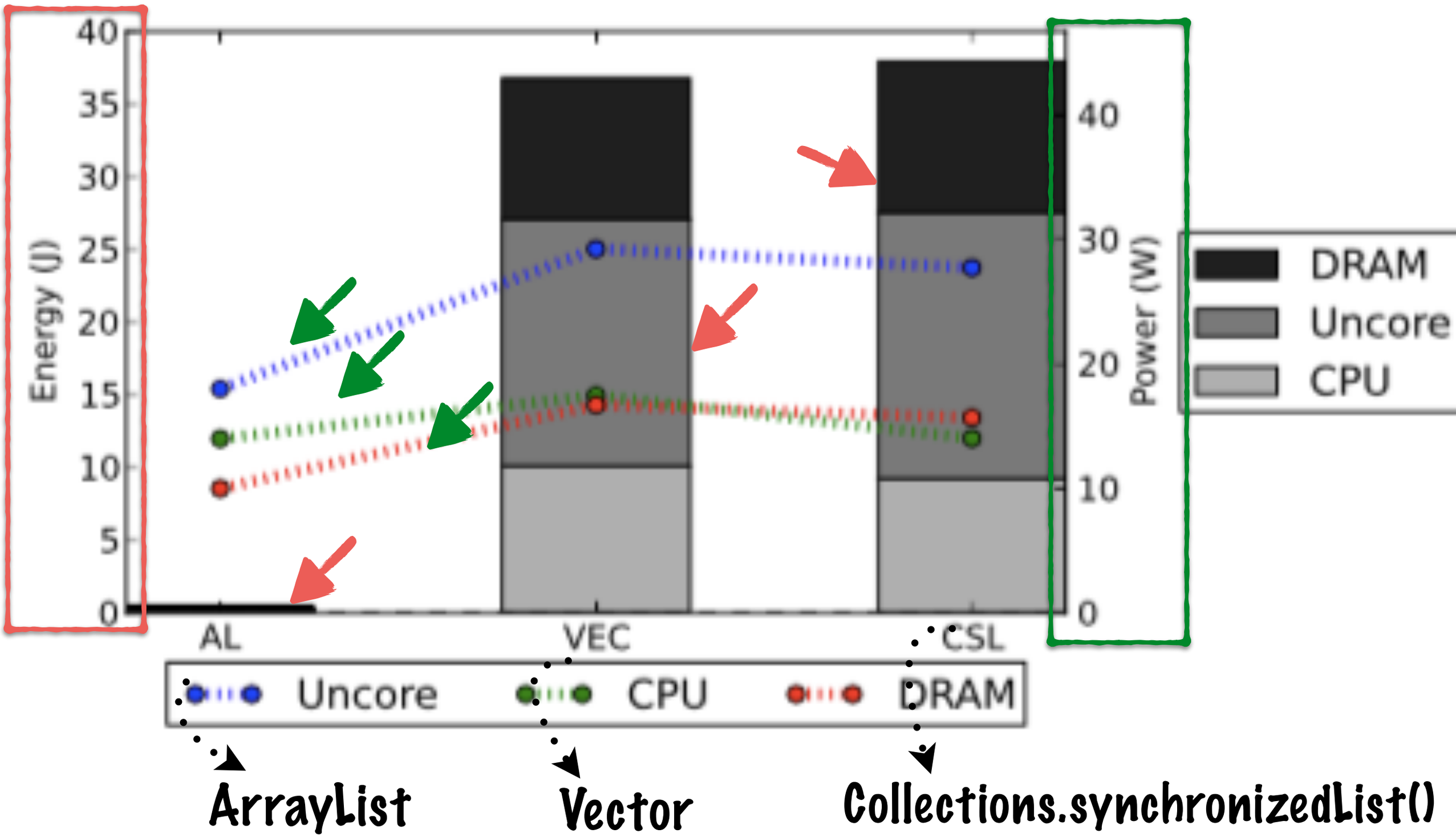
ArrayList

Vector

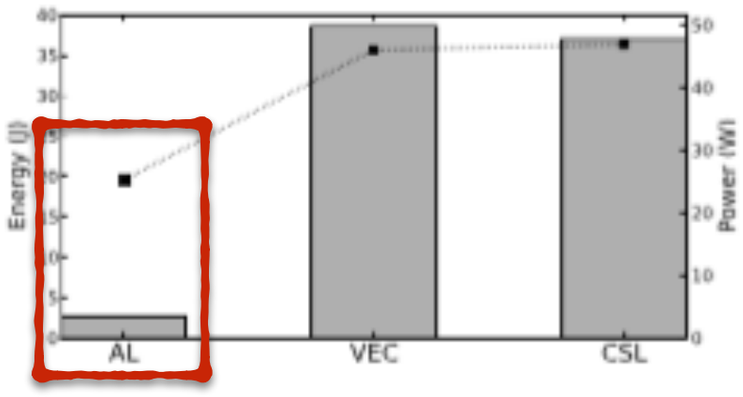
Collections.synchronizedList()

Energy (Joules)

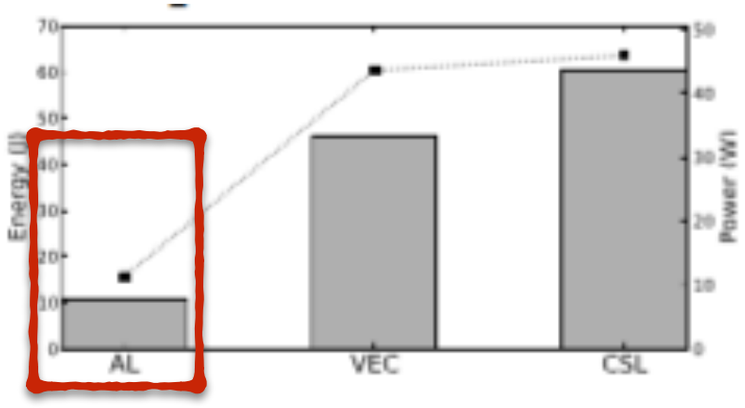
Power (Watts)



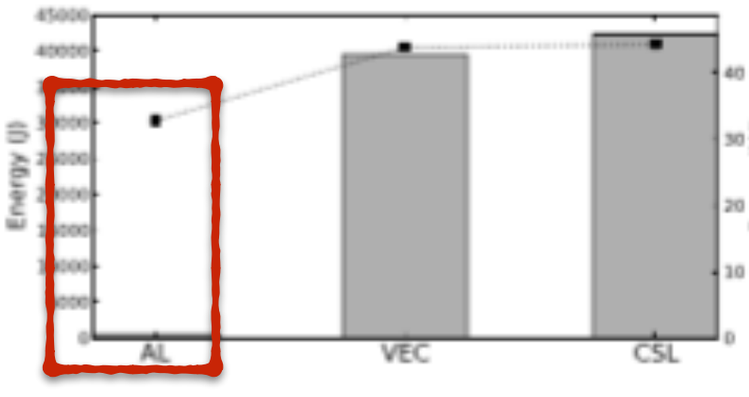
AMD CPU



Traversal

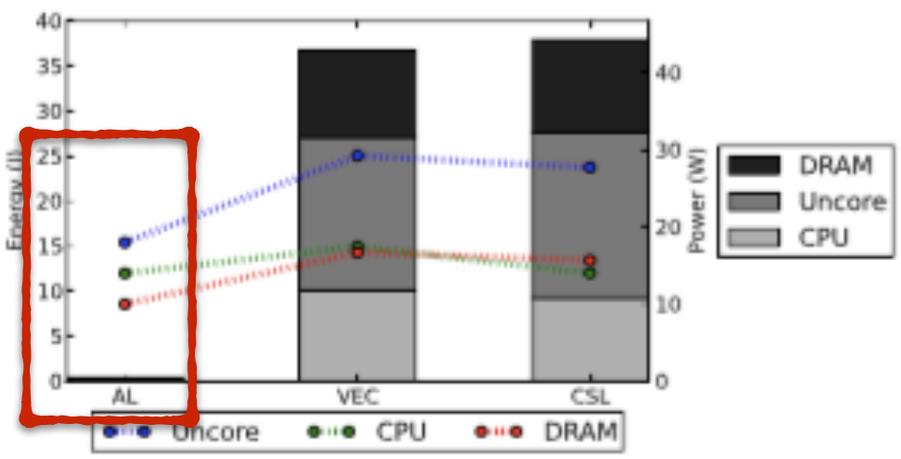


Insertion

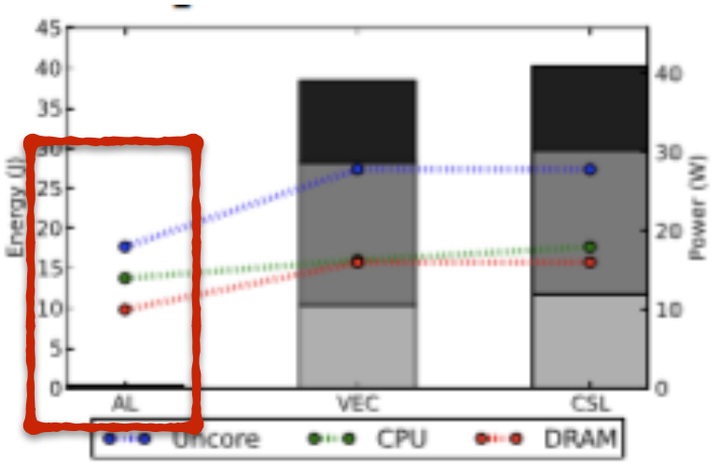


Removal

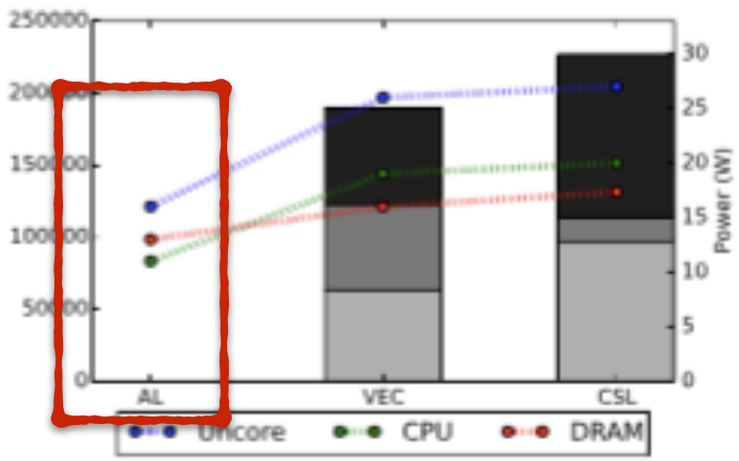
Intel CPU



Traversal

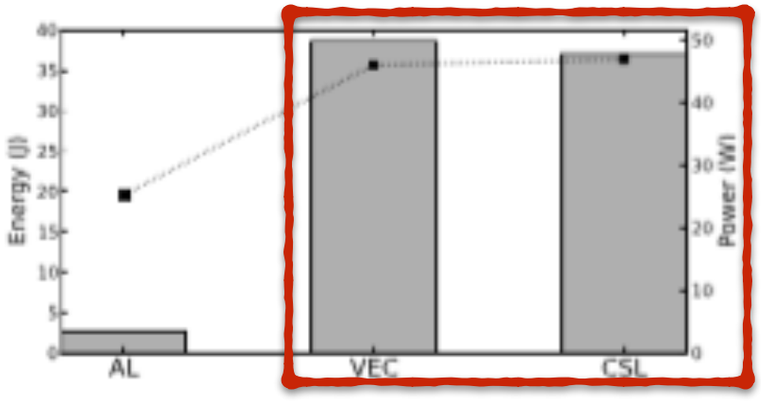


Insertion

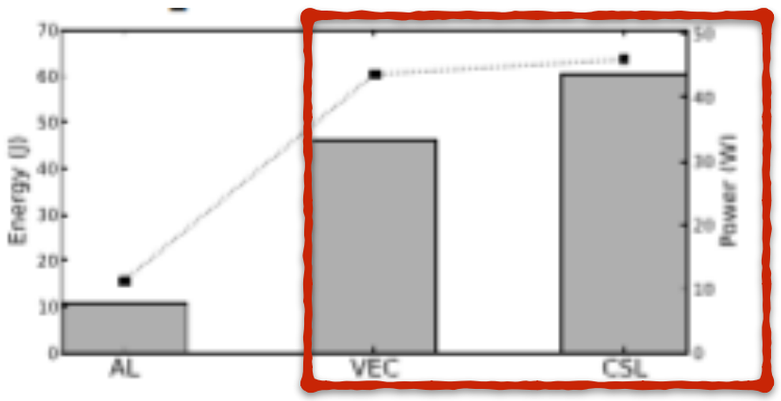


Removal

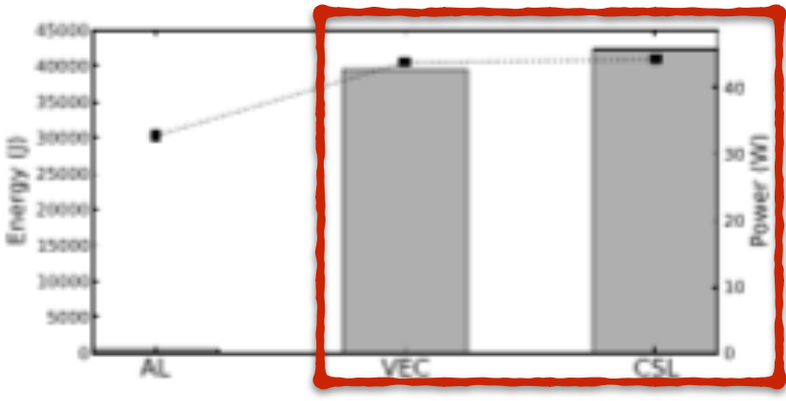
AMD CPU



Traversal

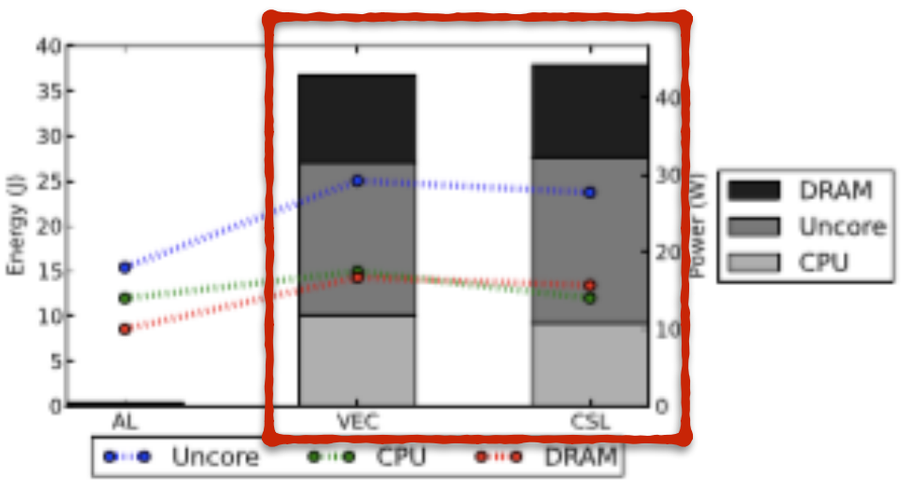


Insertion

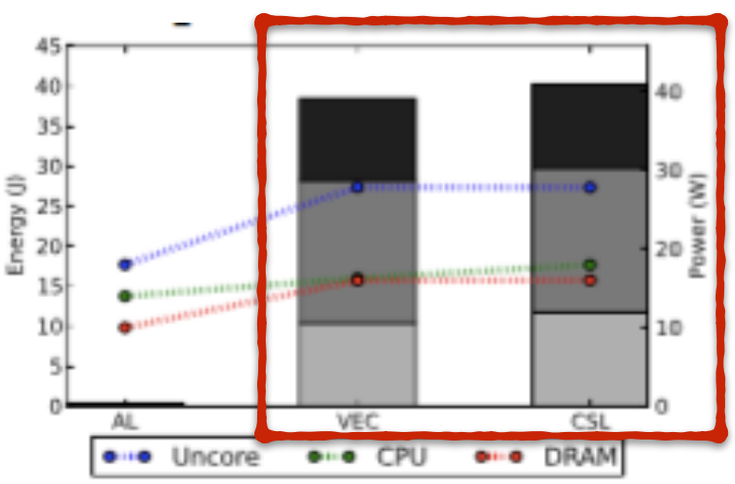


Removal

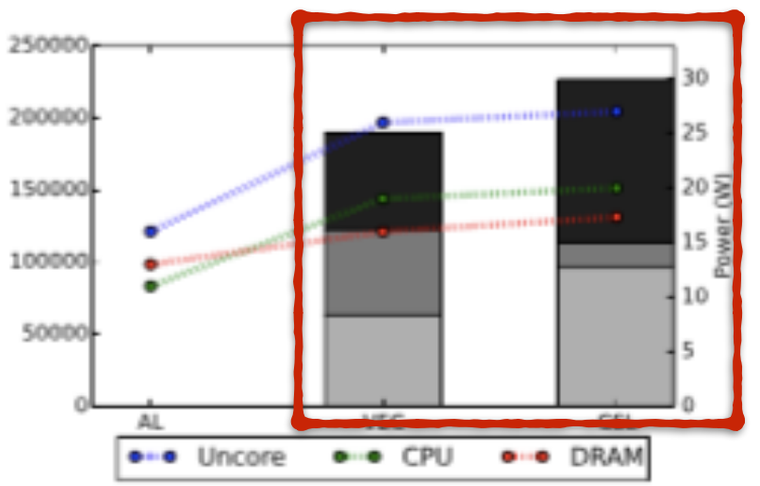
Intel CPU



Traversal



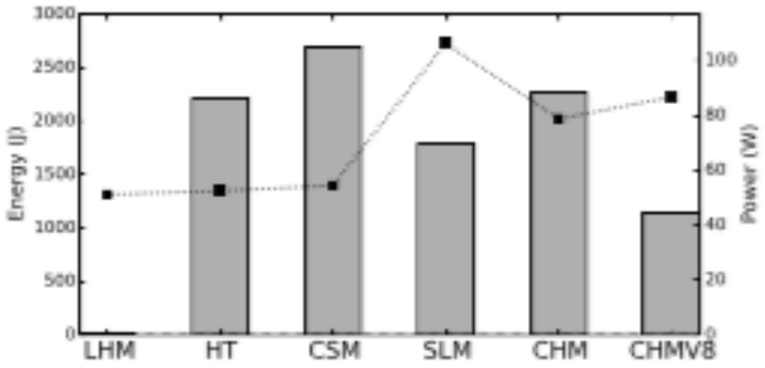
Insertion



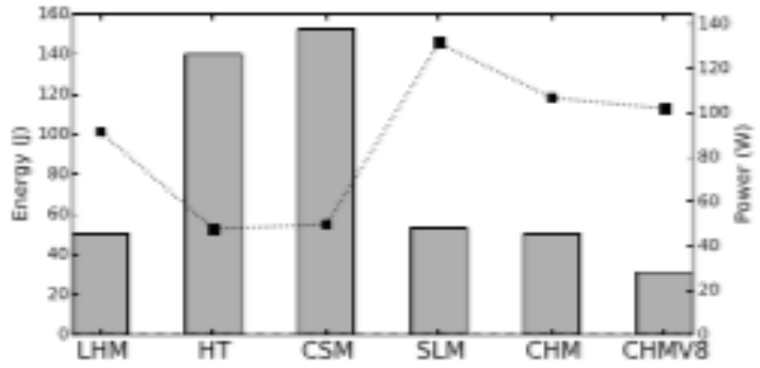
Removal

Maps

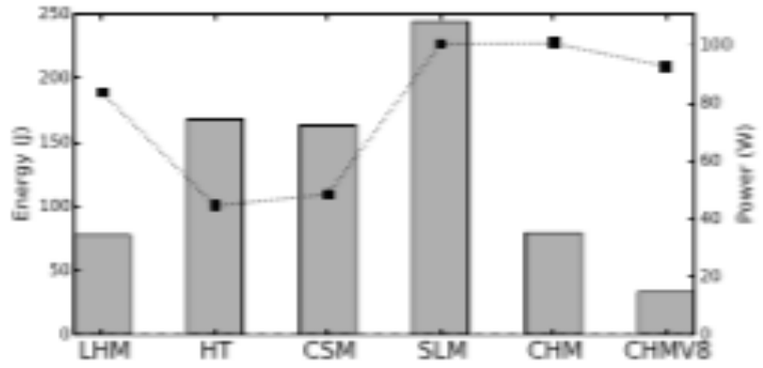
AMD CPU



Traversal

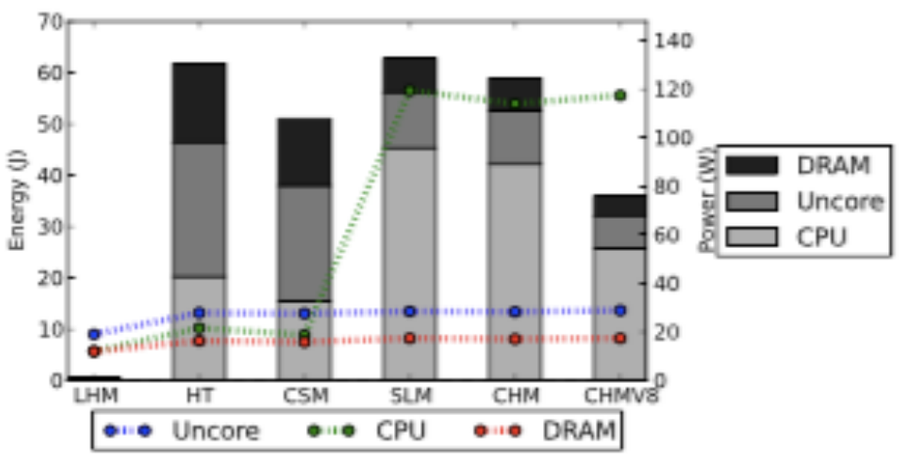


Insertion

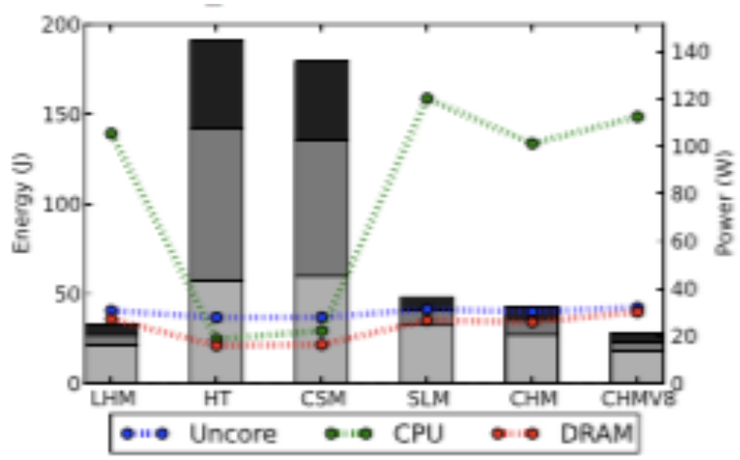


Removal

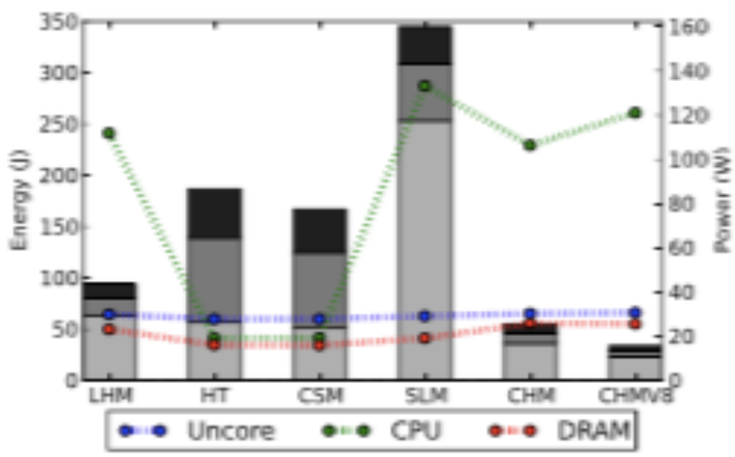
Intel CPU



Traversal



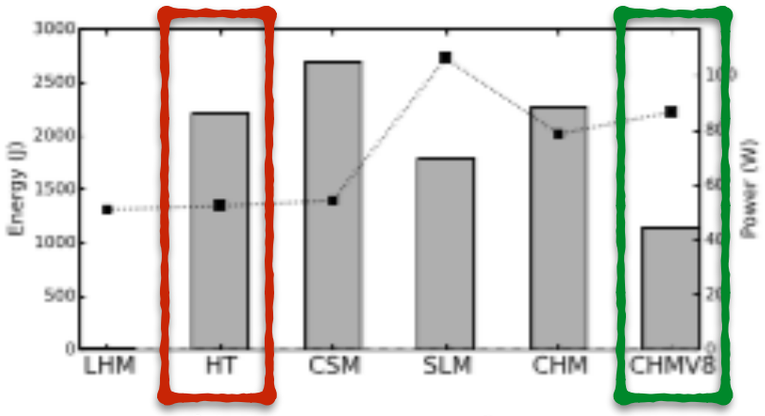
Insertion



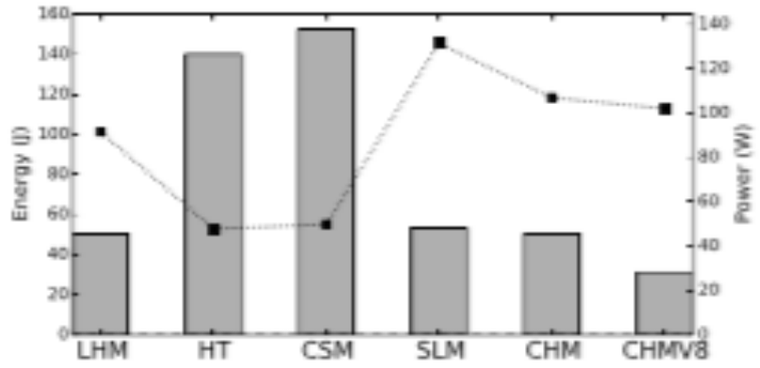
Removal

Maps

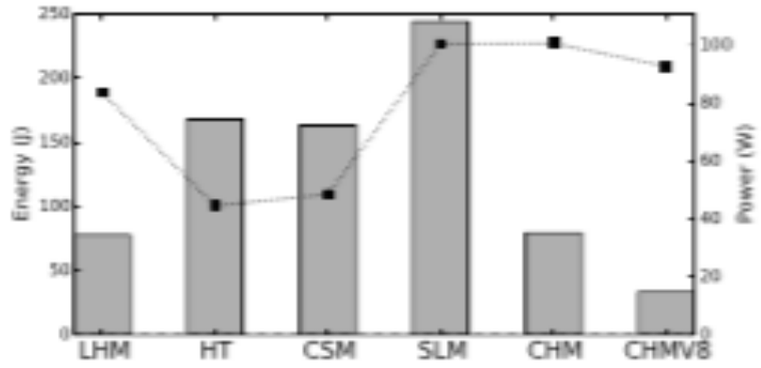
AMD CPU



Traversal

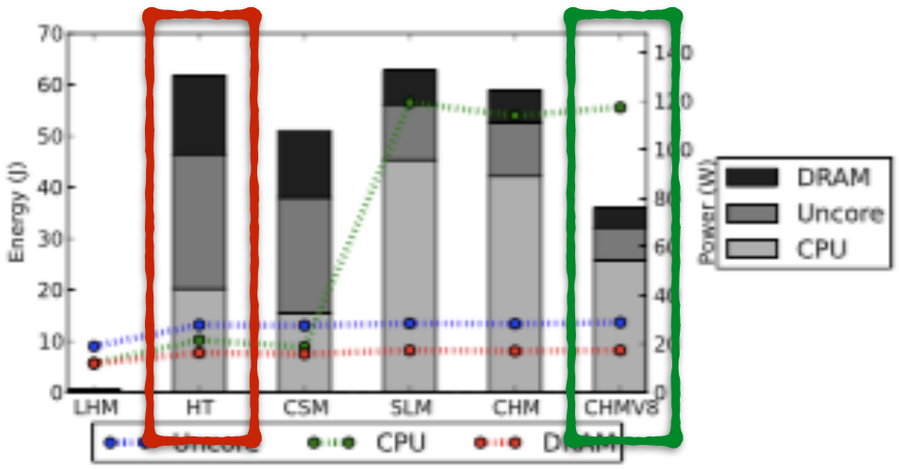


Insertion

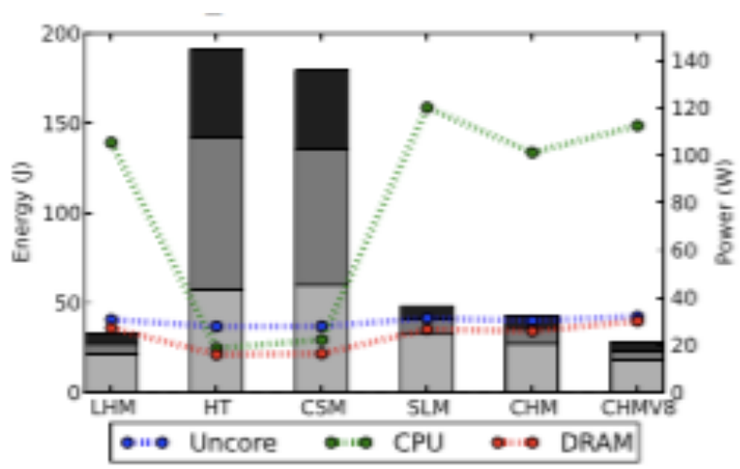


Removal

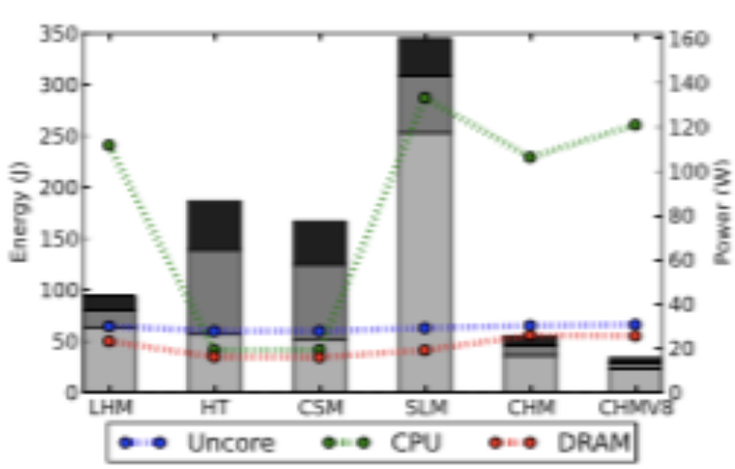
Intel CPU



Traversal



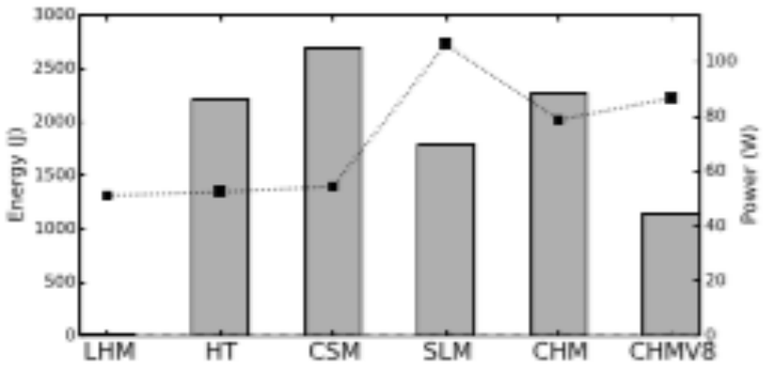
Insertion



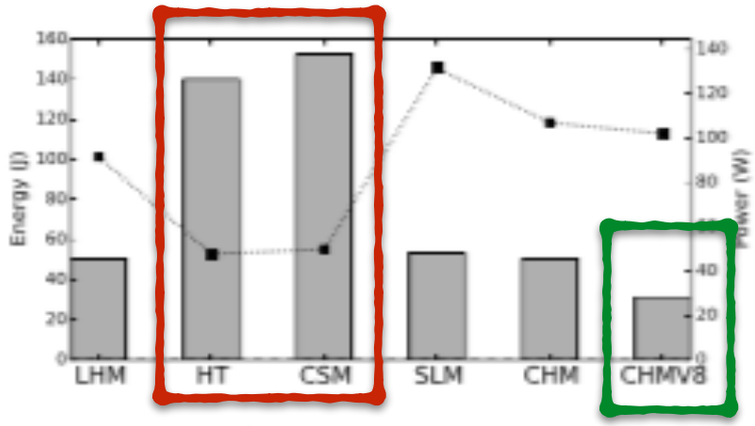
Removal

Maps

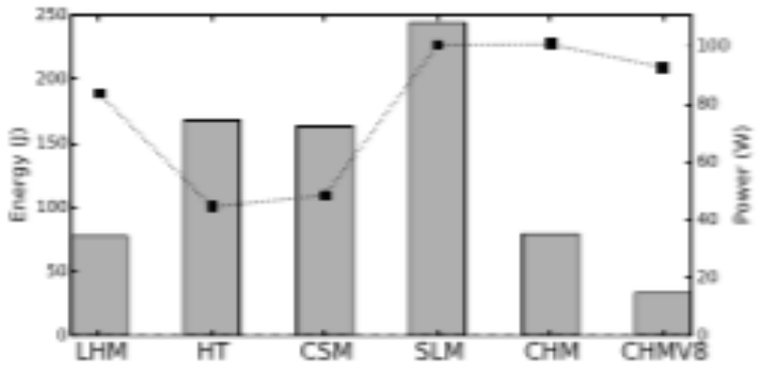
AMD CPU



Traversal

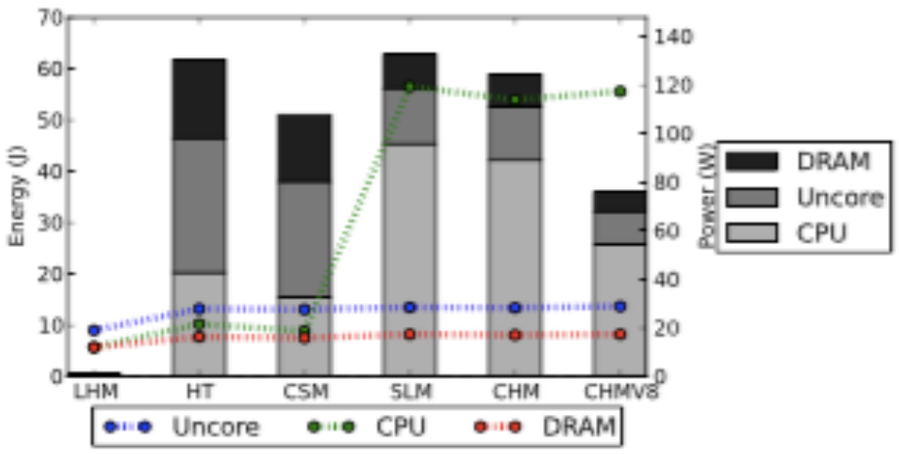


Insertion

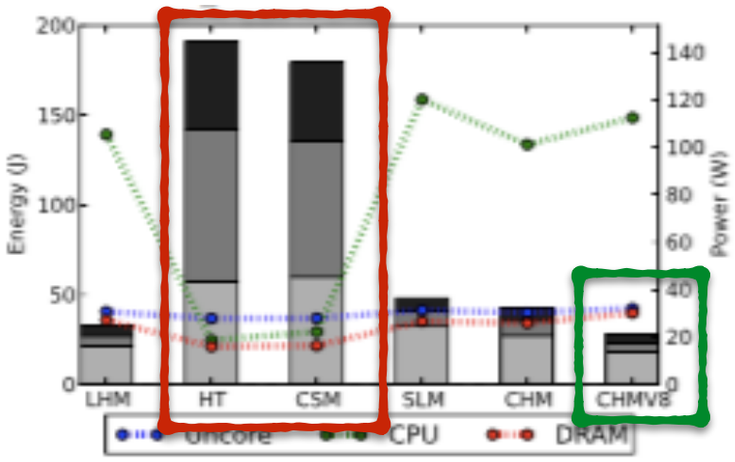


Removal

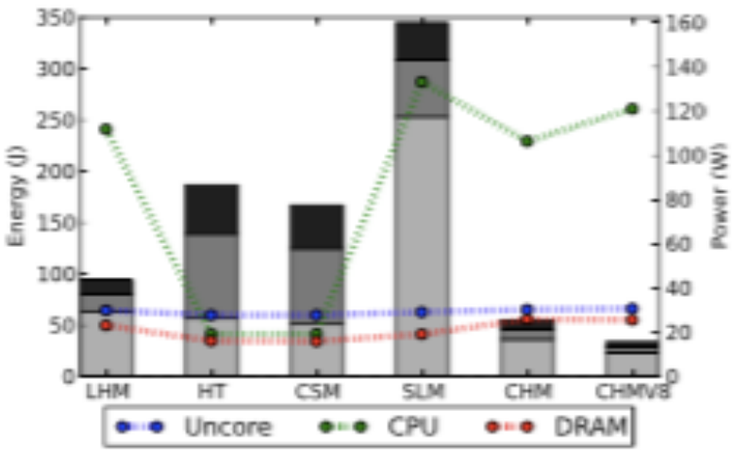
Intel CPU



Traversal



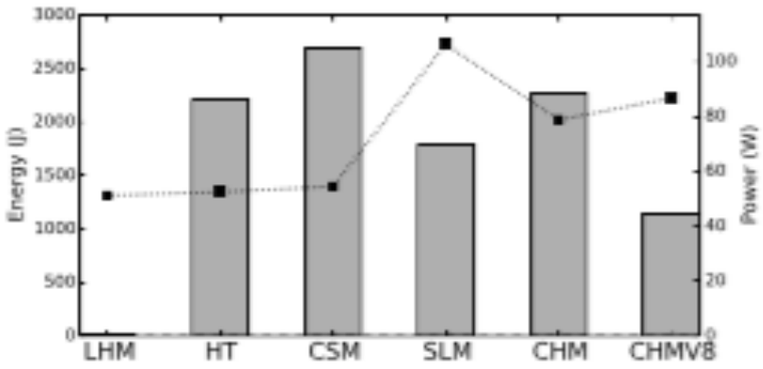
Insertion



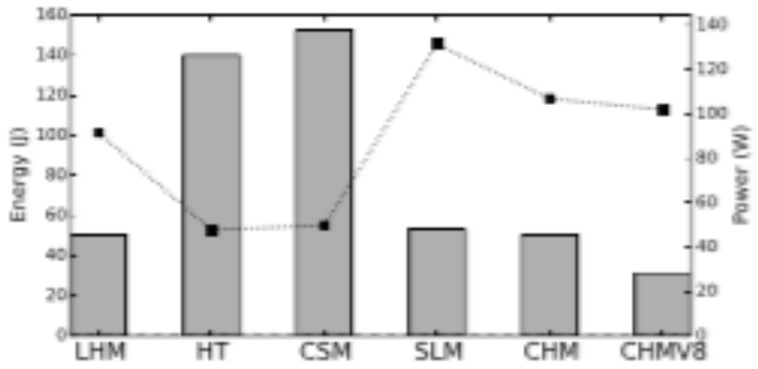
Removal

Maps

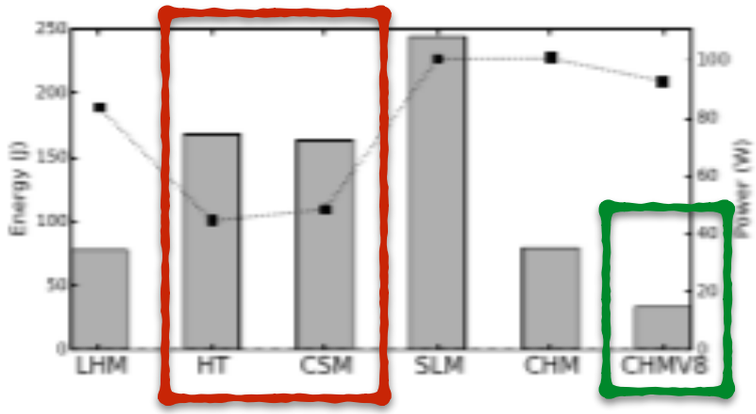
AMD CPU



Traversal

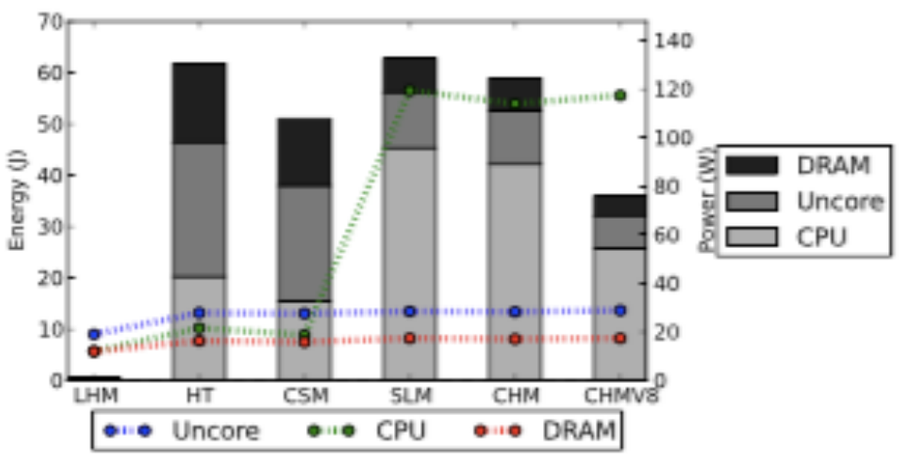


Insertion

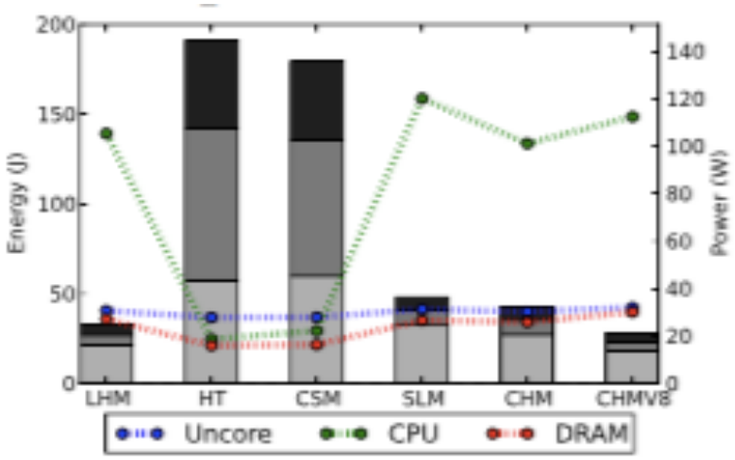


Removal

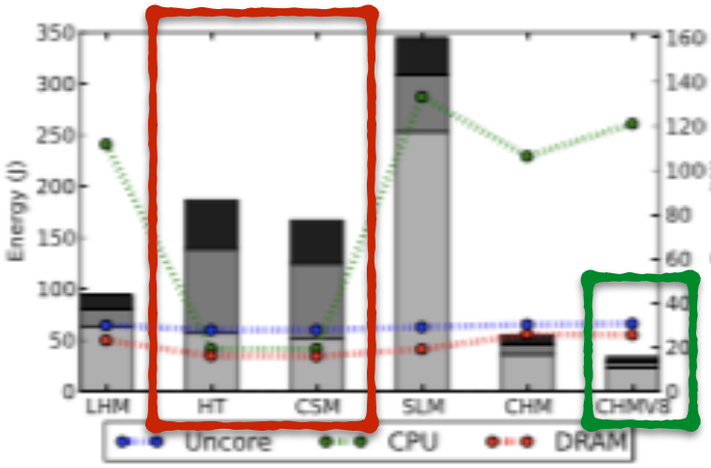
Intel CPU



Traversal



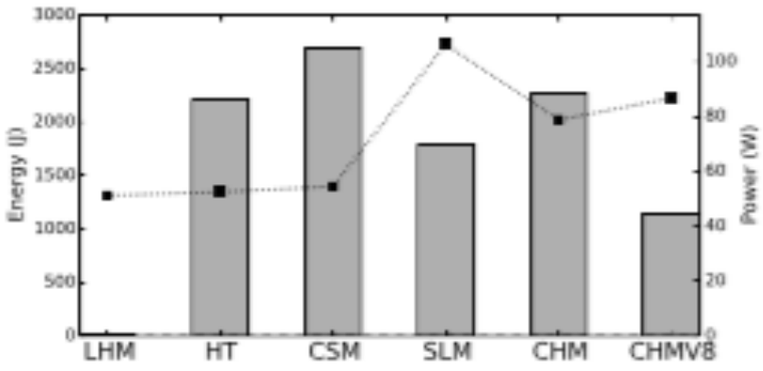
Insertion



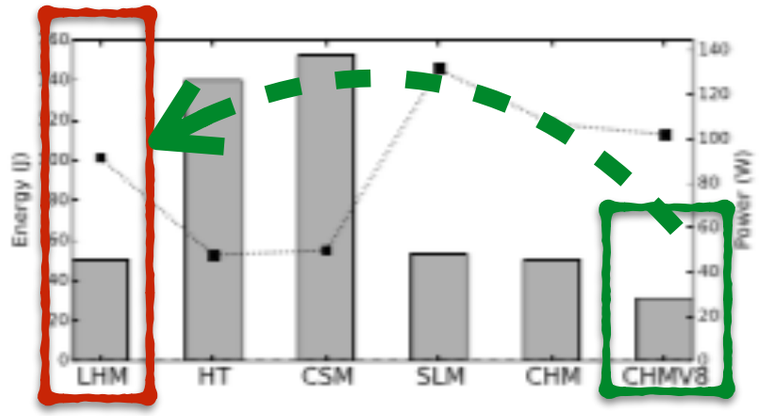
Removal

Maps

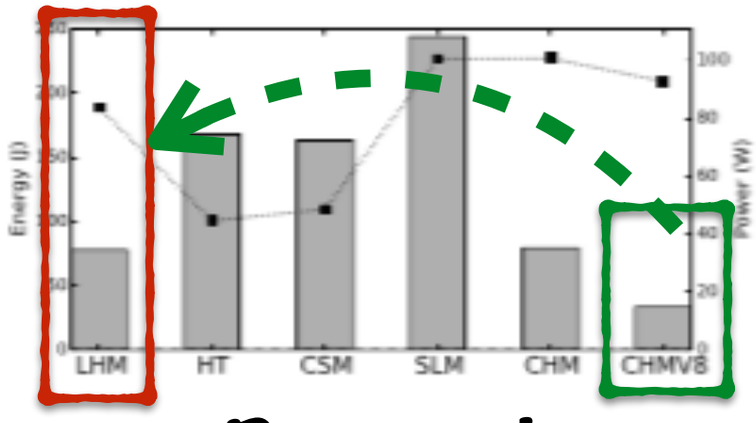
AMD CPU



Traversal

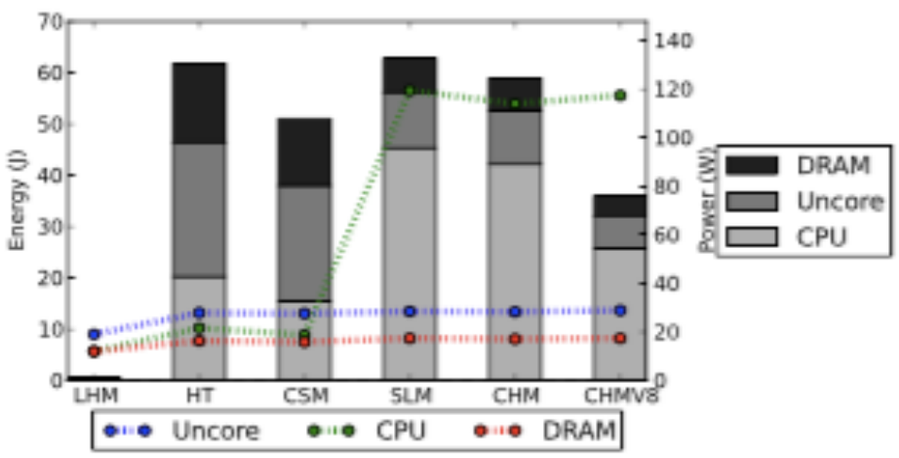


Insertion

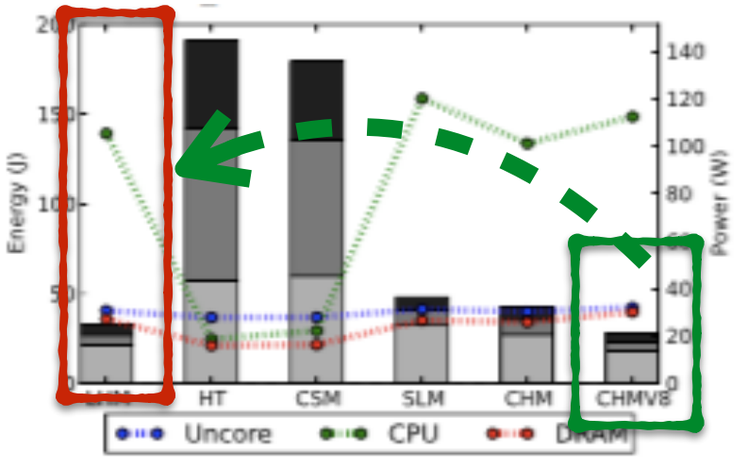


Removal

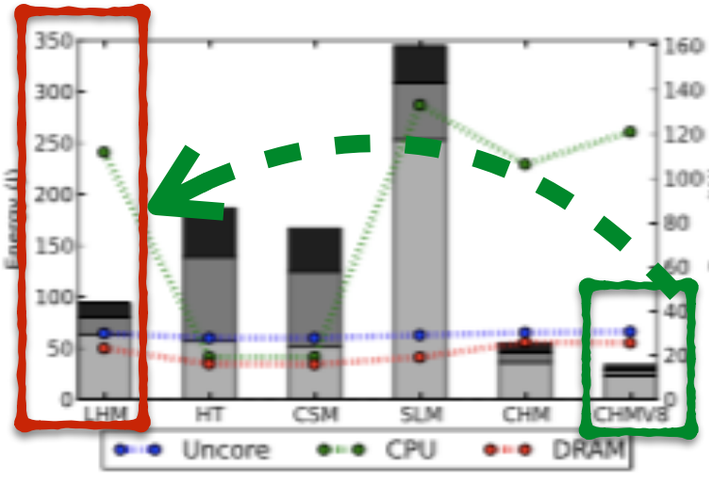
Less energy than the non thread-safe implementation!



Traversal



Insertion



Removal

Case Study

the DaCapo benchmark suite

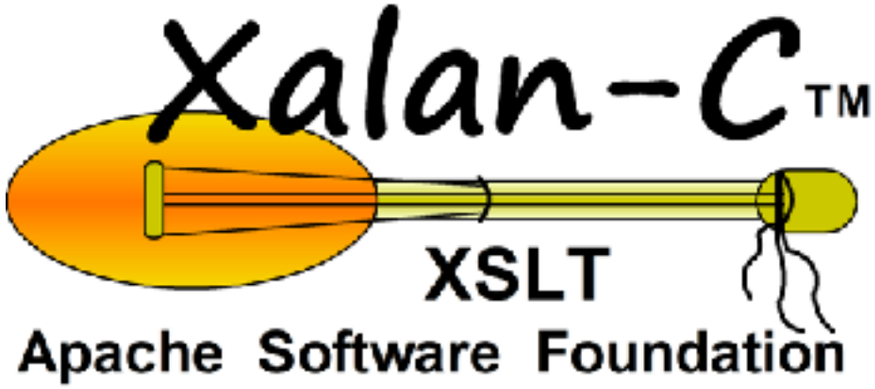
Tomcat

- A web server
- More than 170K lines of Java code
- More than 300 Hashtables

Tomcat

- A web server
- More than 170K lines of Java code
- More than 300 Hashtables

Xalan

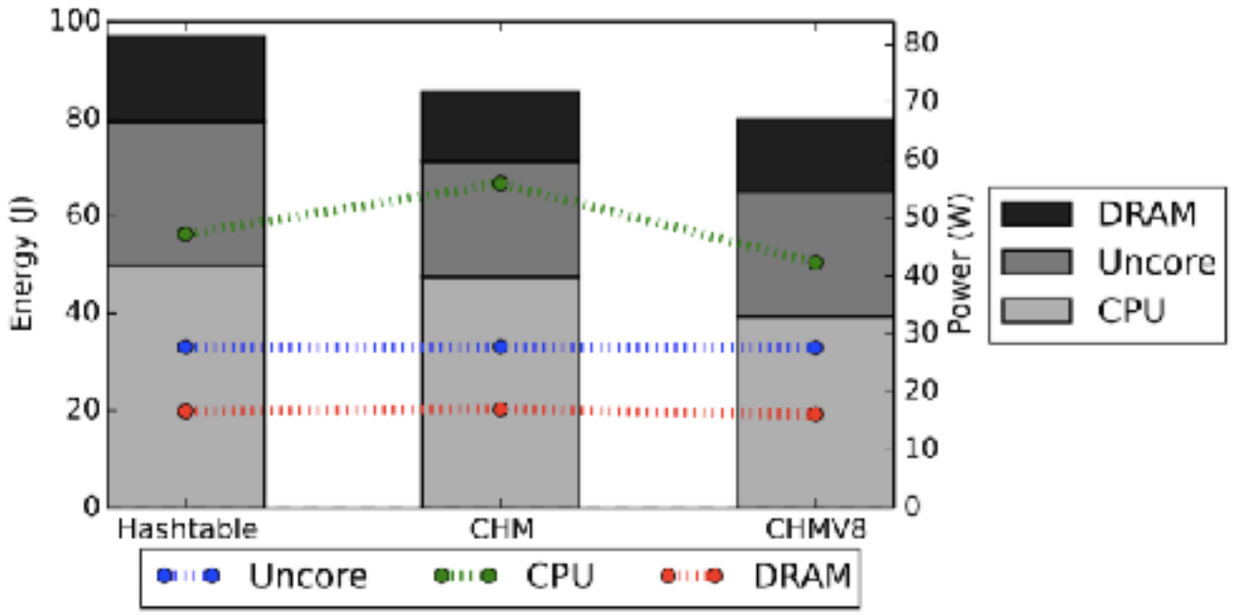


- Parses XML in HTML documents
- More than 188K lines of Java code
- More than 140 Hashtables

Task:

For each Hashtable instance, change it for a ConcurrentHashMap one. Do it again for ConcurrentHashMapV8

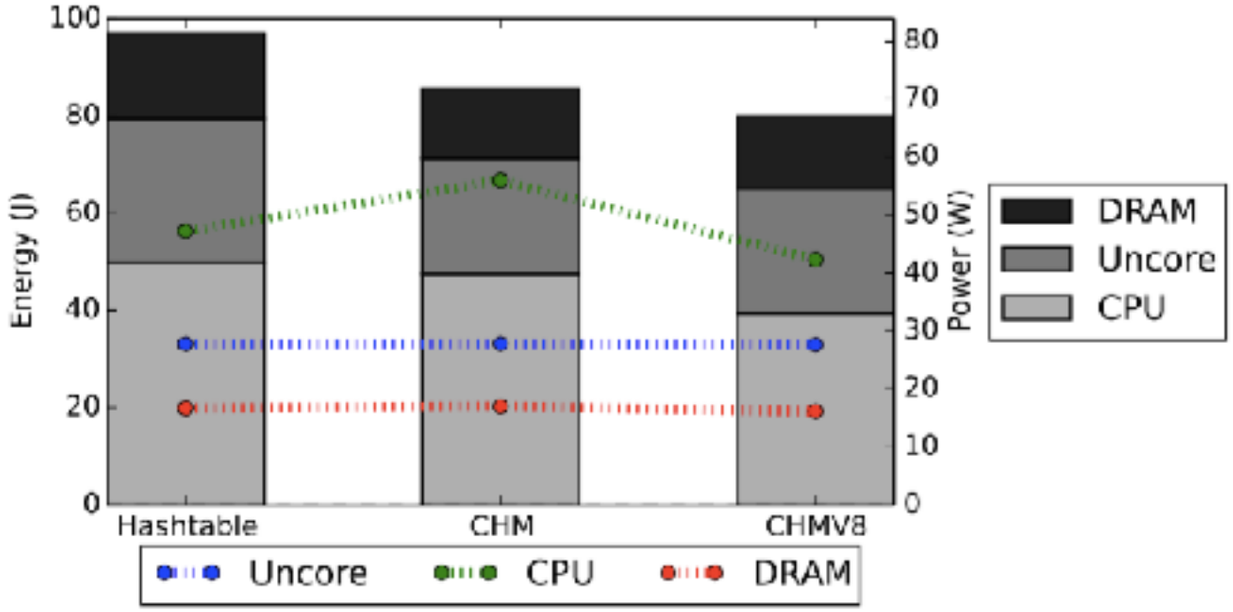
Tomcat



Hashtable to CHM: -12.21%

Hashtable to CHM8: -17.82%

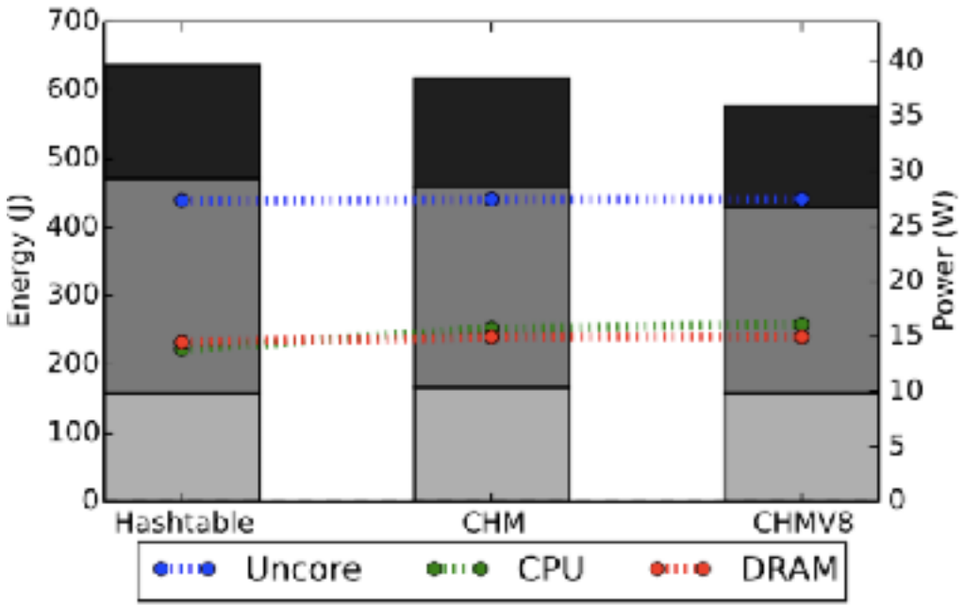
Tomcat



Hashtable to CHM: -12.21%

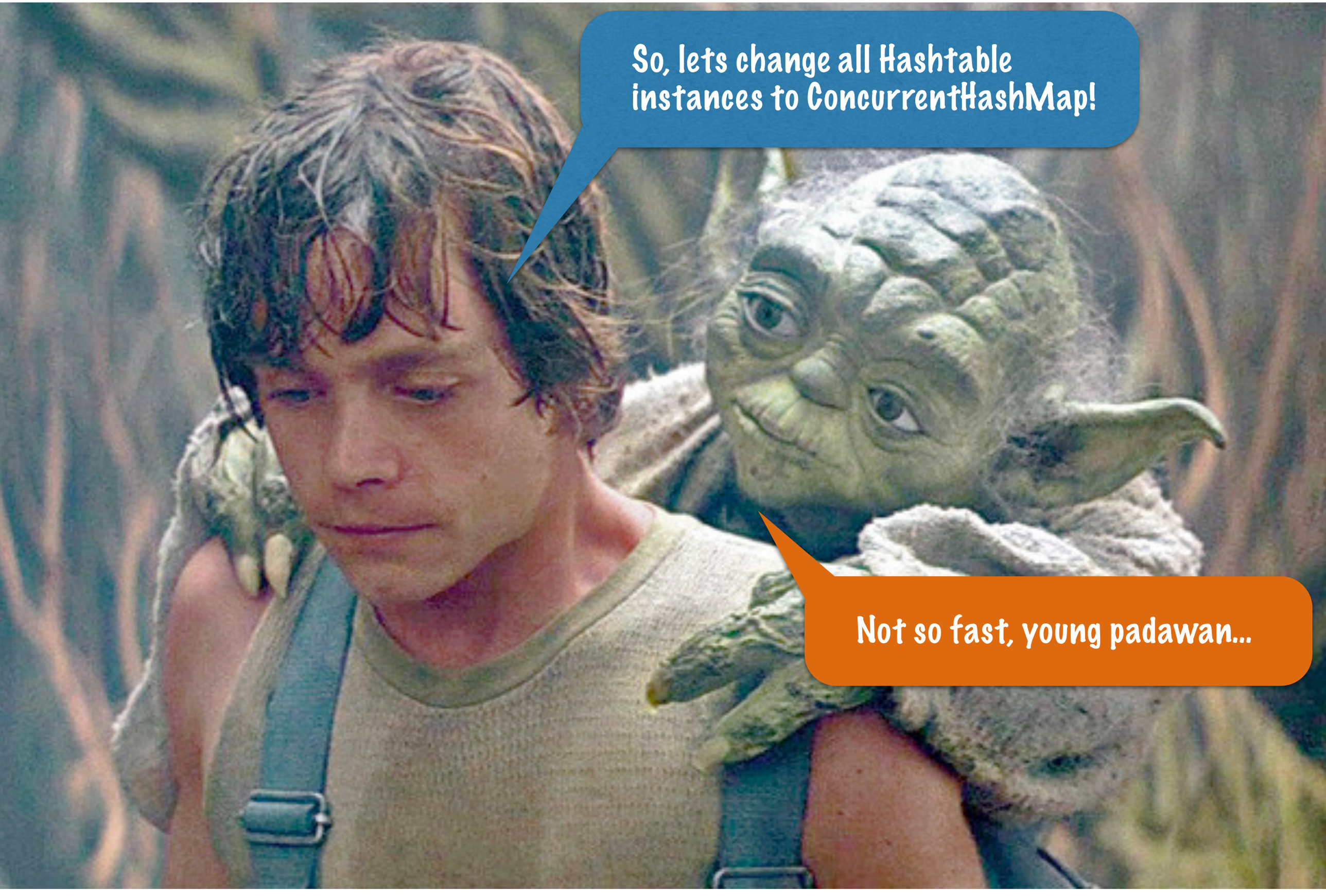
Hashtable to CHM8: -17.82%

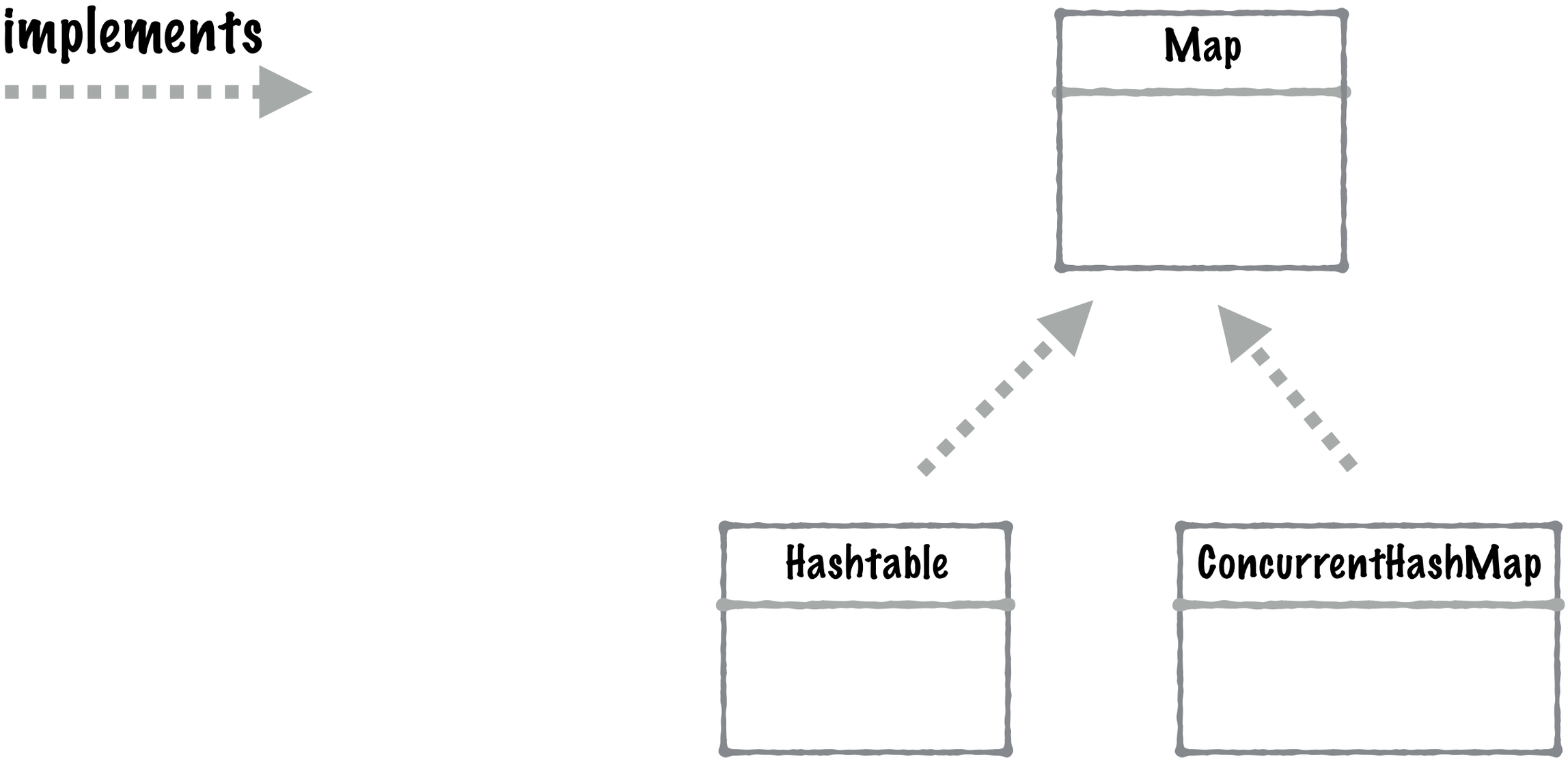
Xalan

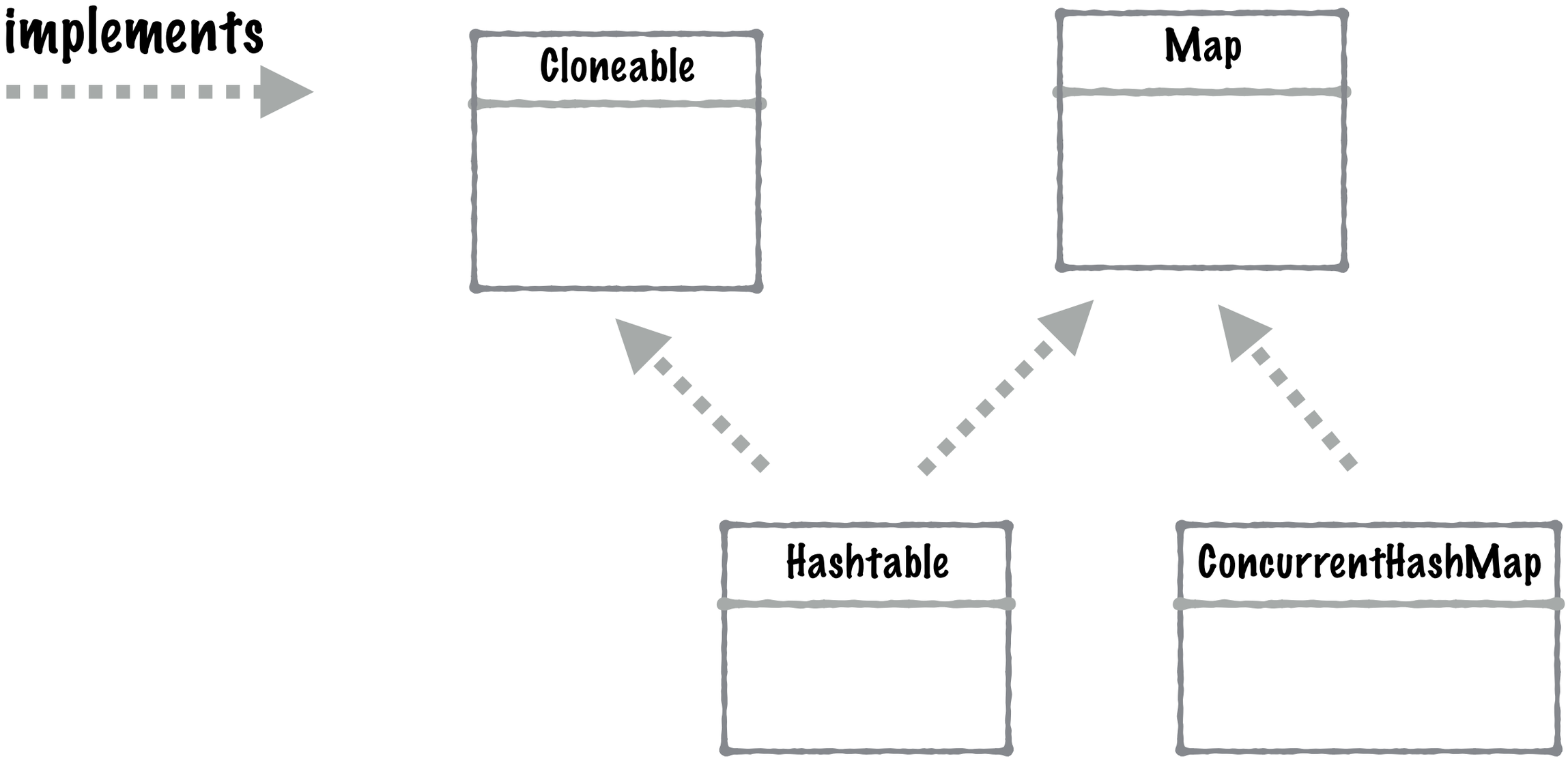


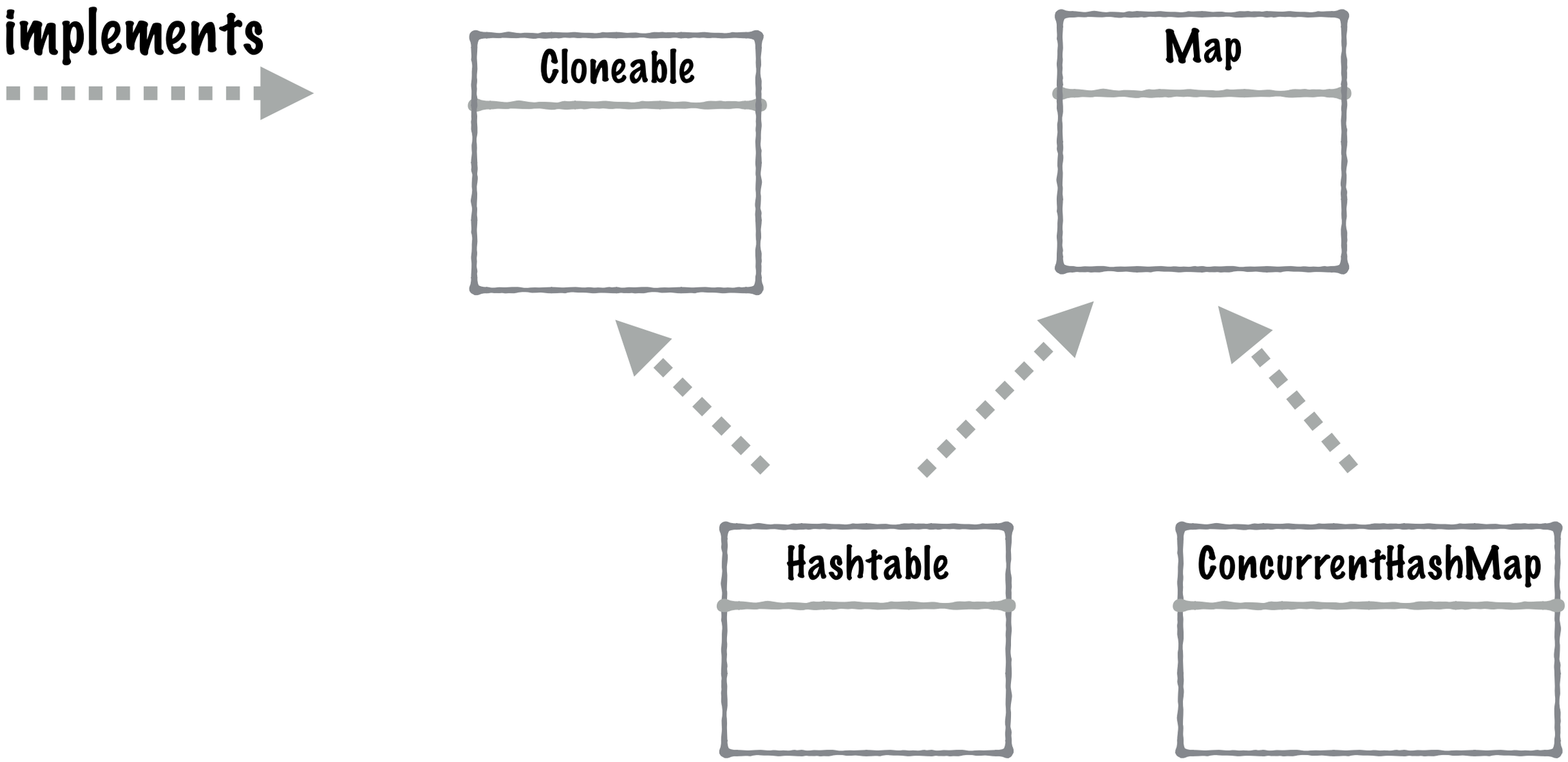
Hashtable to CHM: -5.82%

Hashtable to CHM8: -9.32%


So, lets change all Hashtable instances to ConcurrentHashMap!
Not so fast, young padawan...

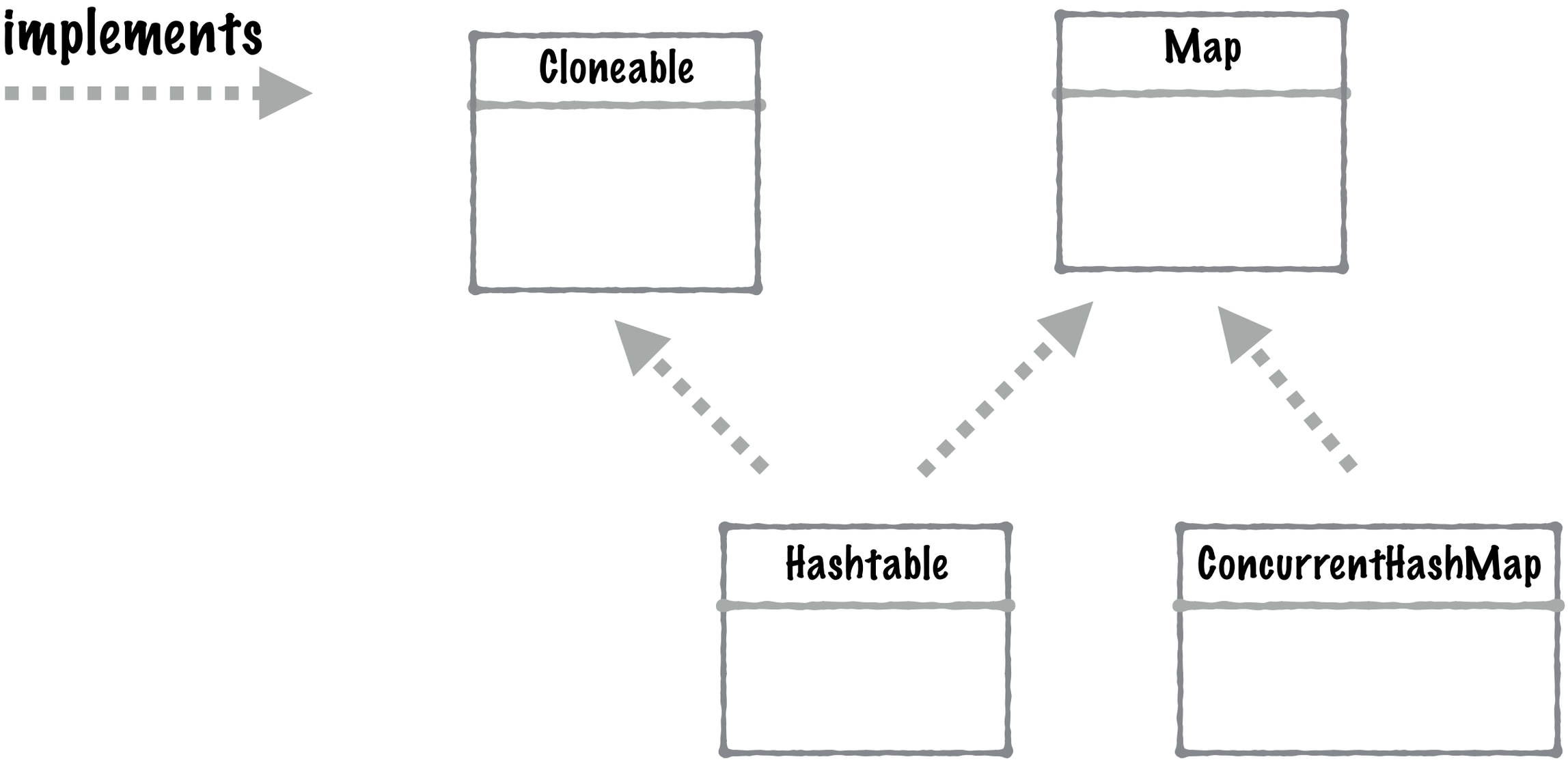

implements
Map
Hashtable
ConcurrentHashMap


implements
Cloneable
Map
Hashtable
ConcurrentHashMap



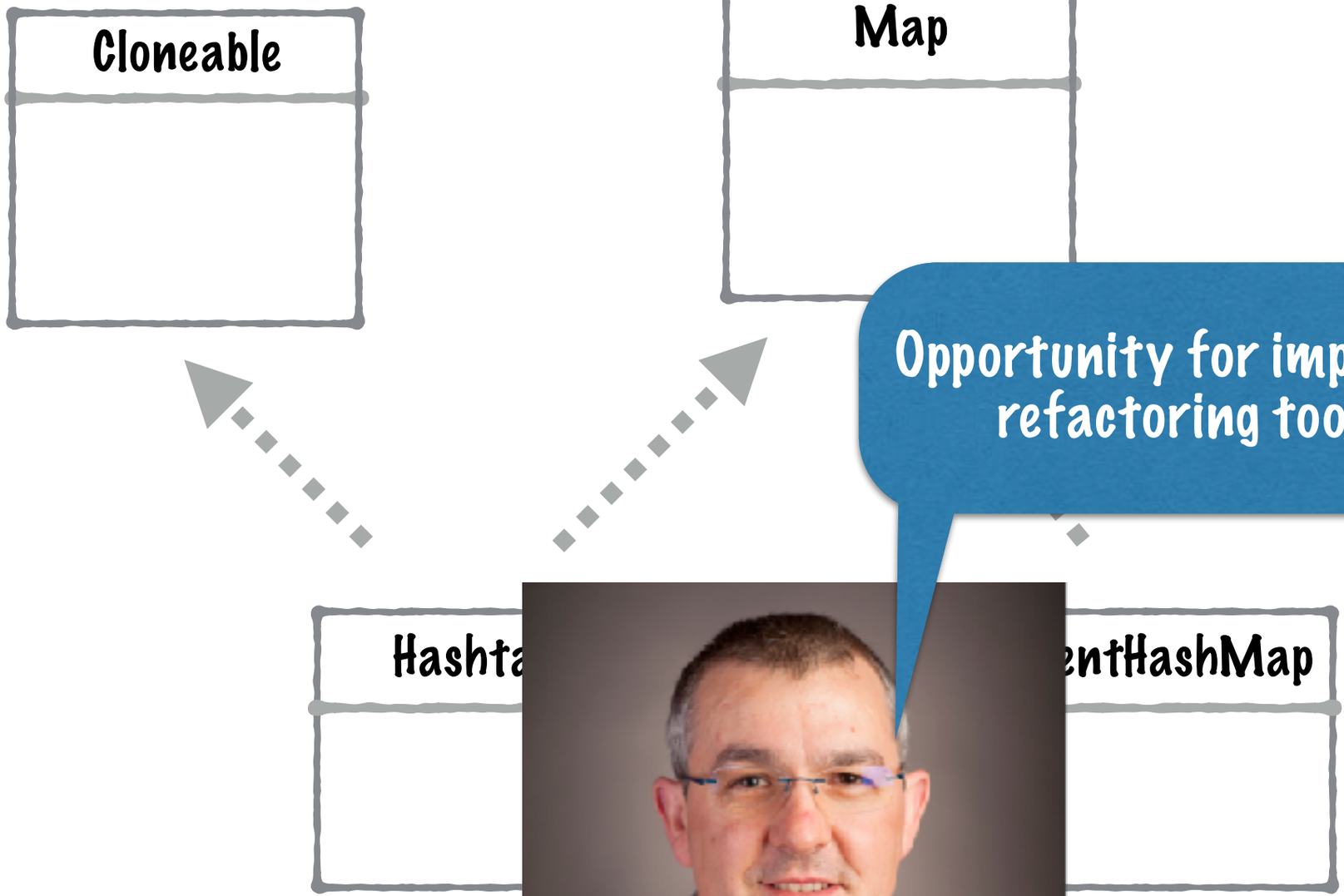

```
Map<X,Y> obj = new Hashtable<>();
obj.clone();
```

// works fine




```
Map<X,Y> obj = new Hashtable<>();
obj.clone();                                  // works fine

Map<X,Y> obj = new ConcurrentHashMap<>();
obj.clone();                                  // compiler error
```

implements



Opportunity for improving refactoring tools!

Danny Dig

```
Map<X,Y> obj = new Hashtable<>();
obj.clone();
```

ks fine

```
Map<X,Y> obj = new ConcurrentHashMap<>();
obj.clone();                                  // compiler error
```

2017

What are the most common bottlenecks in Java parallel computations?

Understanding and Overcoming Parallelism Bottlenecks in ForkJoin Applications

Gustavo Pinto[1] Anthony Canino[2] Fernando Castor[3] Harry Guoqing Xu[4] Yu David Liu[2]
[1]UFPA, Brazil [2]SUNY Binghamton, USA [3]UFPE, Brazil [4]UC Irvine, USA

Abstract—FORKJOIN framework is a widely used parallel programming framework upon which both core concurrency libraries and real-world applications are built. Beneath its simple and user-friendly APIs, FORKJOIN is a sophisticated managed parallel runtime unfamiliar to many application programmers: the framework core is a *work-stealing* scheduler, handles *fine-grained* tasks, and sustains the pressure from automatic memory management. FORKJOIN poses a unique gap in the compute stack between high-level software engineering and low-level system optimization. Understanding and bridging this gap is crucial for the future of parallelism support in JVM-supported applications.

This paper describes a comprehensive study on parallelism bottlenecks in FORKJOIN applications, with a unique focus on how they interact with underlying system-level features, such as work stealing and memory management. We identify 6 bottlenecks, and found that refactoring them can significantly improve performance and energy efficiency. Our field study includes an in-depth analysis of AKKA — a real-world actor framework — and 30 additional open-source FORKJOIN projects. We sent our patches to the developers of 15 projects, and 7 out of the 9 projects that replied to our patches have accepted them.

I. INTRODUCTION

Modern Java applications predominately run on parallel architectures, whose performance and energy efficiency critically depend on efficient thread management. FORKJOIN [1] is an influential parallel framework at the core of Java concurrency design. It not only provides thread management to numerous real-world applications, but also serves as the bedrock for higher-level Java concurrent libraries [2]. The impact of FORKJOIN also goes beyond Java applications *per se*, as several new programming languages [3], [4], [5] continue to operate on Java Virtual Machines (JVMs) and rely on FORKJOIN for thread management. FORKJOIN is known for its intuitive programming interface, particularly suitable for programming task-parallel and data-parallel jobs that have a divide-and-conquer nature.

FORKJOIN employs a *work-stealing* runtime [1]. While work stealing provides many benefits in resource utilization and scalability, efficient stealing dictates careful coordination across the layers of applications, runtime systems, and the OS. System-level performance and energy optimizations for C-based work-stealing programs are not new [6], [7], [8], [9], but combining work stealing with a Java-like managed runtime and, more importantly, reorienting it to application programming comes with a distinct set of unique challenges:

- *Thread Management*: work stealing by nature is "decentralized coordination", where threads coordinate on system utilization but thread management decisions are made by individual threads. This is in contrast with existing approaches either lacking coordination (*e.g.*, Java `Thread` objects) or requiring centralized management (*e.g.*, thread pooling).
- *Synchronization Management*: work stealing presents a unique flavor in the management of synchronization and thread states. Unfortunately, these features conflict with traditional approaches such as locks (*e.g.*, synchronized methods) and explicit thread state management (*e.g.*, sleep) [10], leading to erratic performance surprising to application programmers. This problem is exacerbated by the large legacy code base of Java applications and libraries.
- *Data Management*: the Java runtime primarily allocates objects in the heap, and deallocation is managed by garbage collection. This is in sharp contrast with C-based work-stealing frameworks [11], where data are routinely represented as arrays of primitive data types. As a result, the allocation and distribution of data among worker threads plays a pivotal role in application performance.

Do these challenges introduce bottlenecks in modern parallel applications running on FORKJOIN? How severe are these bottlenecks in terms of performance and energy efficiency? Is there generalizable wisdom that can be shared with FORKJOIN programmers to avoid the bottlenecks?

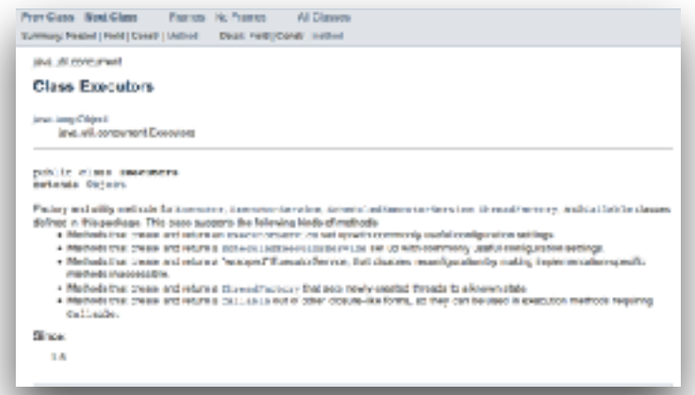
This Paper We present the first empirical study to bridge the gap between modern software engineering and work-stealing systems in the context of FORKJOIN. It aims at providing a better understanding—as well as raising the awareness—of the subtleties and common performance pitfalls in FORKJOIN programming through a comprehensive study of characteristics and behaviors of real-world FORKJOIN applications. We identify potential bottlenecks against parallelism in these applications, illustrate their impact on systems performance and energy, and demonstrate how such bottlenecks can be overcome through refactoring.

Our study follows a unique cross-layer approach: it is *application-driven* and *system-aware*. On the one hand, we are more interested in how real-world applications built on FORKJOIN behave—and how their performance can be improved through application-level programming—rather than an "under-the-hood" system-level optimization. On the other hand, we are aimed at finding the root causes of the bottlenecks on the systems stack, such as how each bottleneck may potentially hamper the desired behavior of the work stealing scheduler, garbage collector, and underlying hardware. This

ASE'17

Modern Java applications run on parallel architectures

java.lang.Thread



java.util.concurrent.Executors

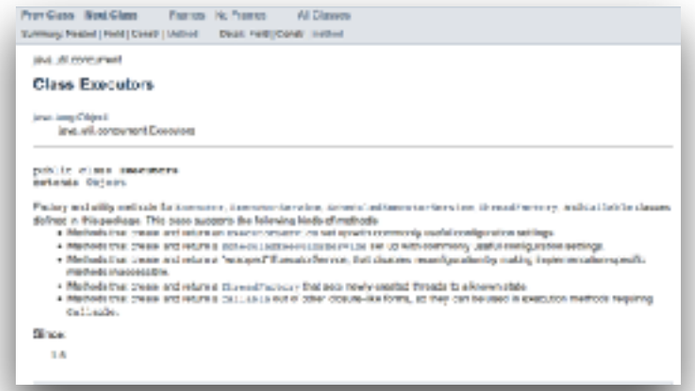
- Widely used
- Low level API
- Error prone

- Well used
- High Level API
- User friendly

Modern Java applications run on parallel architectures

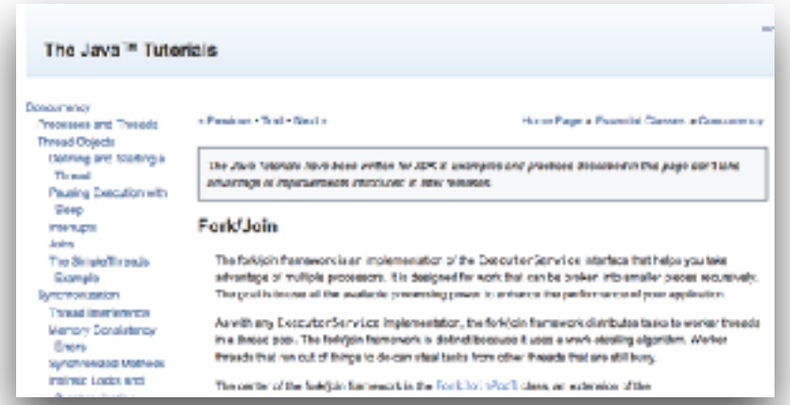
java.lang.Thread

- 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

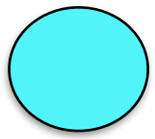
ForkJoin

- Can be more used
- Sophisticated API
- Sophisticated scheduler

Why ForkJoin?

@gustavopinto

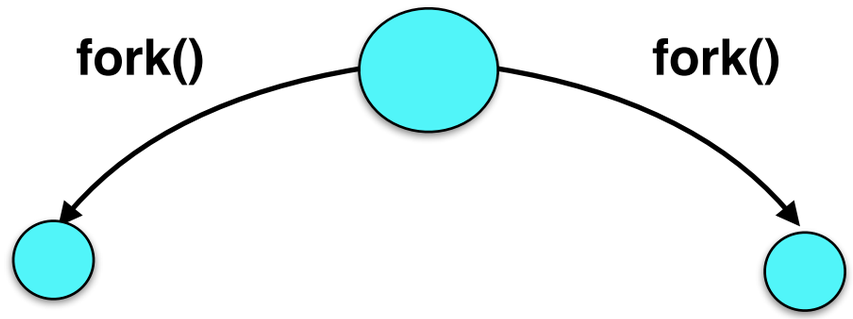
Divider and conquer algorithm



Why ForkJoin?

@gustavopinto

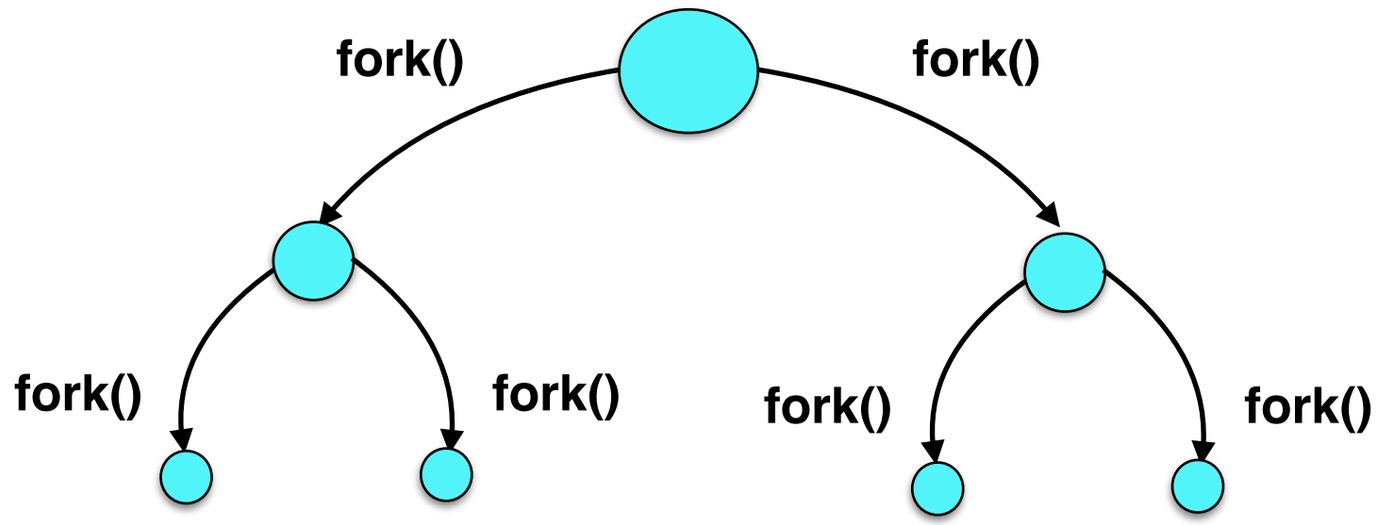
Divider and conquer algorithm



Why ForkJoin?

@gustavopinto

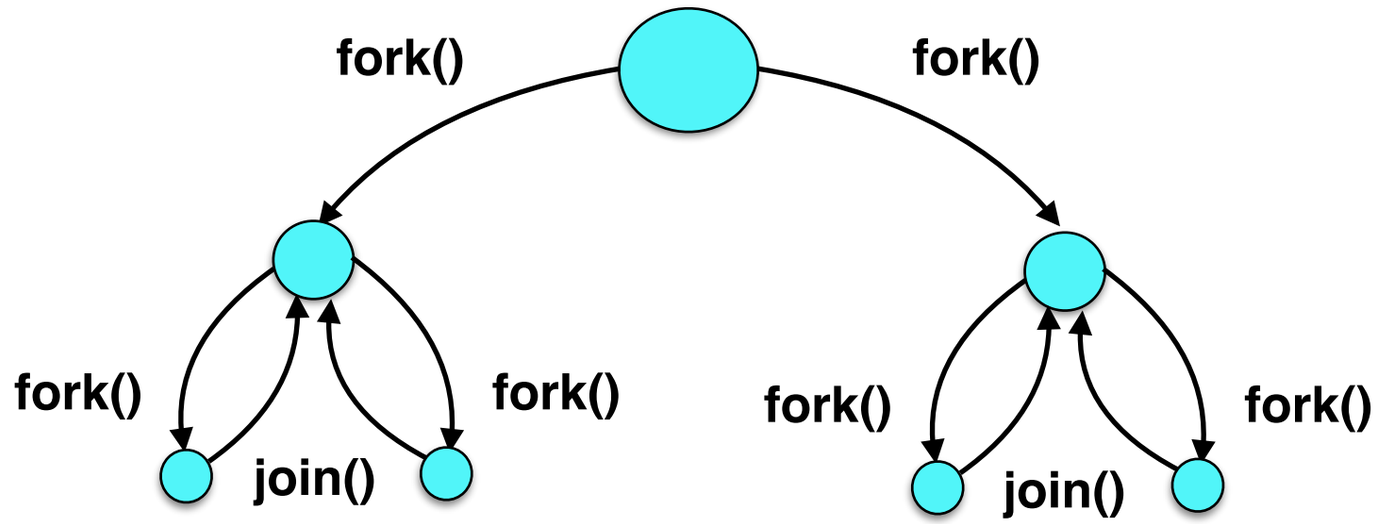
Divider and conquer algorithm



Why ForkJoin?

@gustavopinto

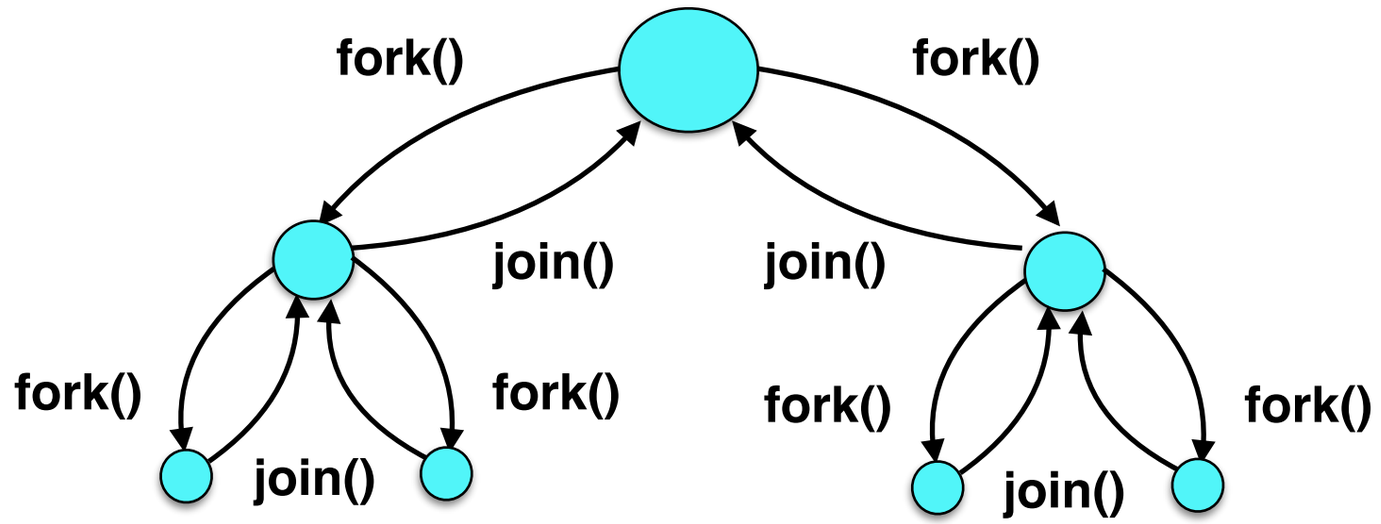
Divider and conquer algorithm



Why ForkJoin?

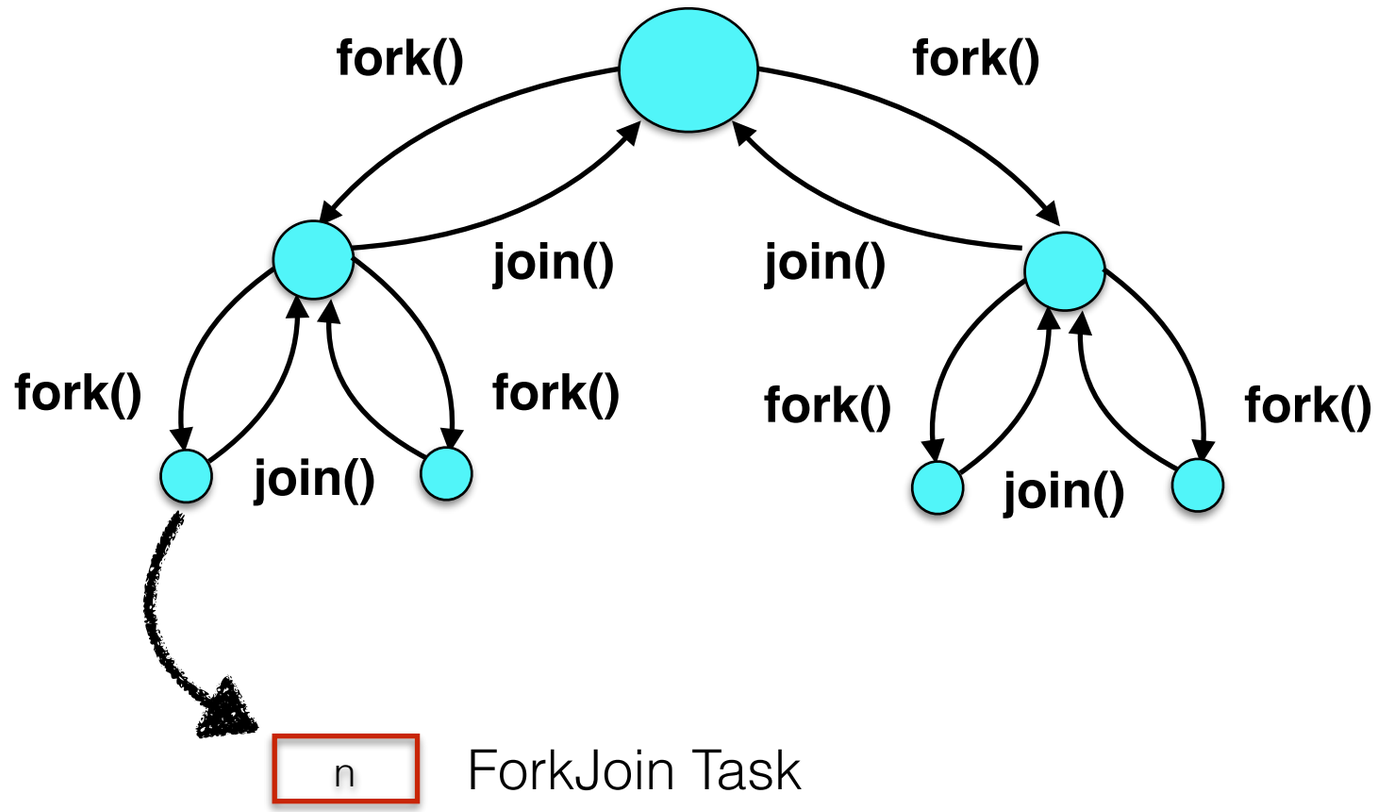
@gustavopinto

Divider and conquer algorithm



Why ForkJoin?

Divider and conquer algorithm

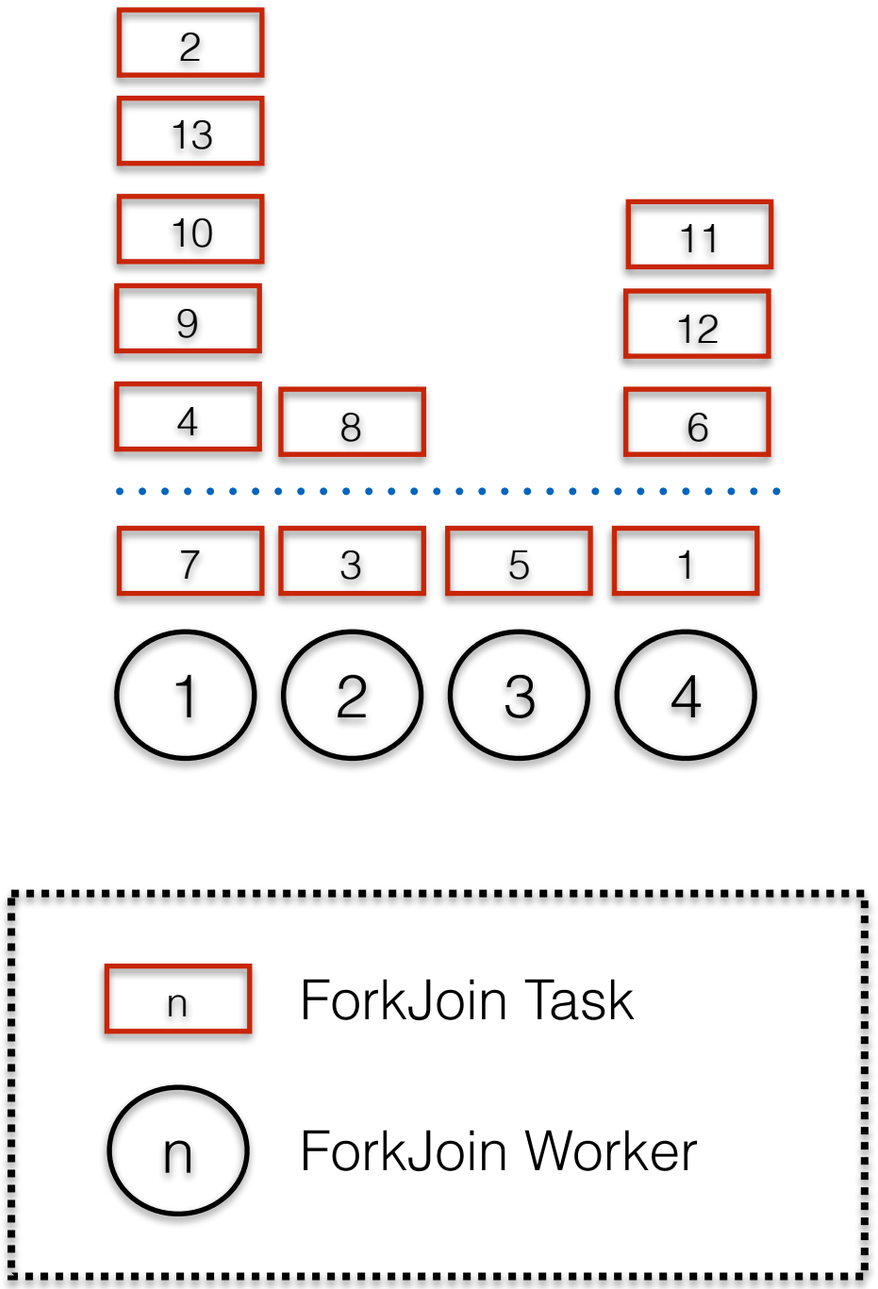


@gustavopinto

Why ForkJoin?

@gustavopinto

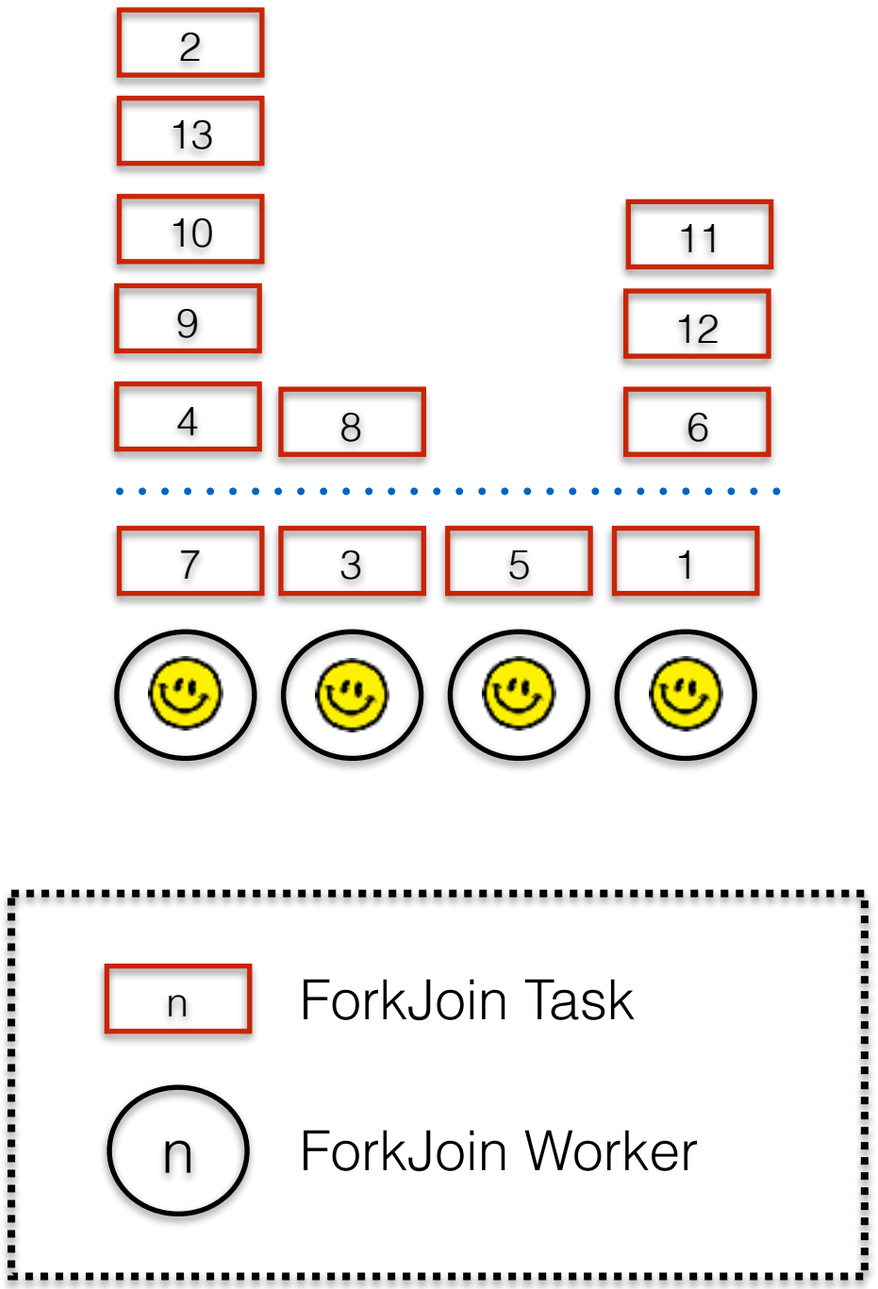
Work Stealing



Why ForkJoin?

@gustavopinto

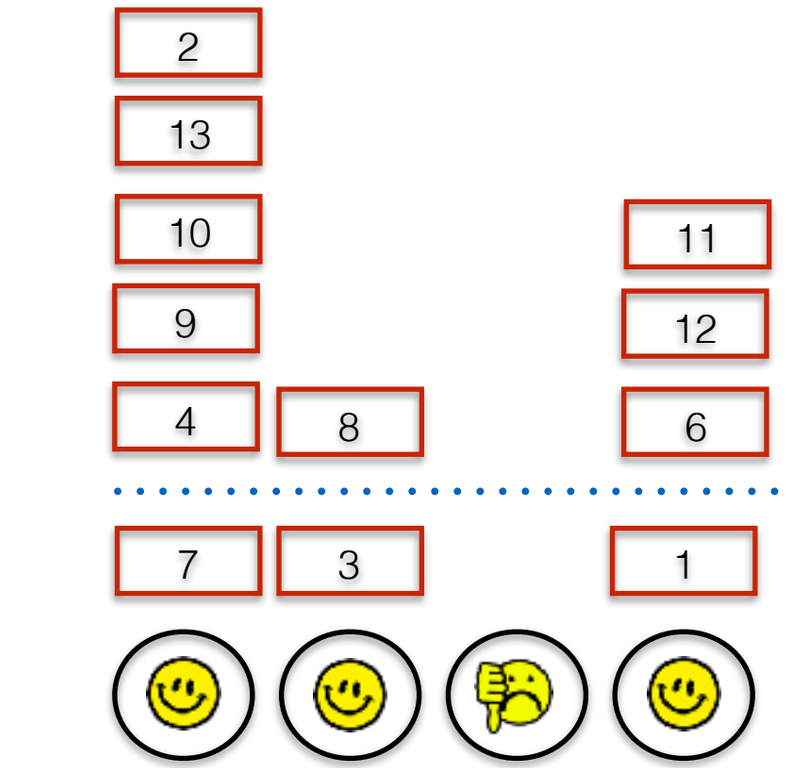
Work Stealing



Why ForkJoin?

@gustavopinto

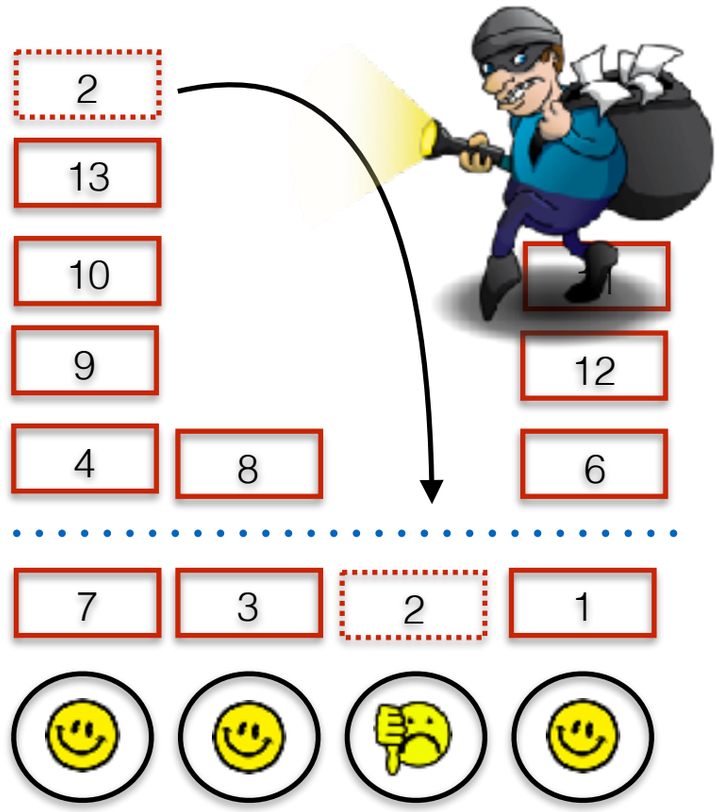
Work Stealing



Why ForkJoin?

@gustavopinto

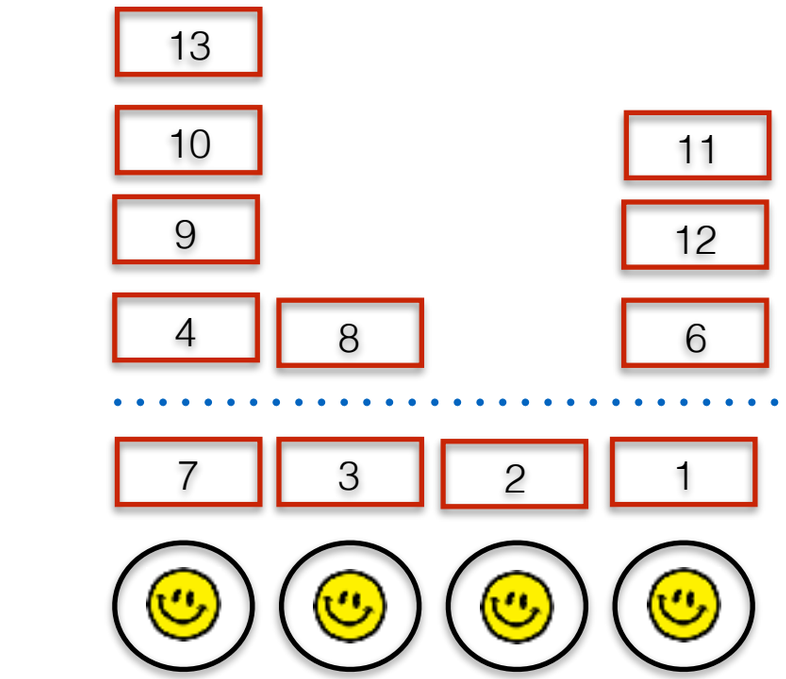
Work Stealing



Why ForkJoin?

@gustavopinto

Work Stealing



ForkJoin Applications

@gustavopinto

Our corpus of data

System Programmers

akka

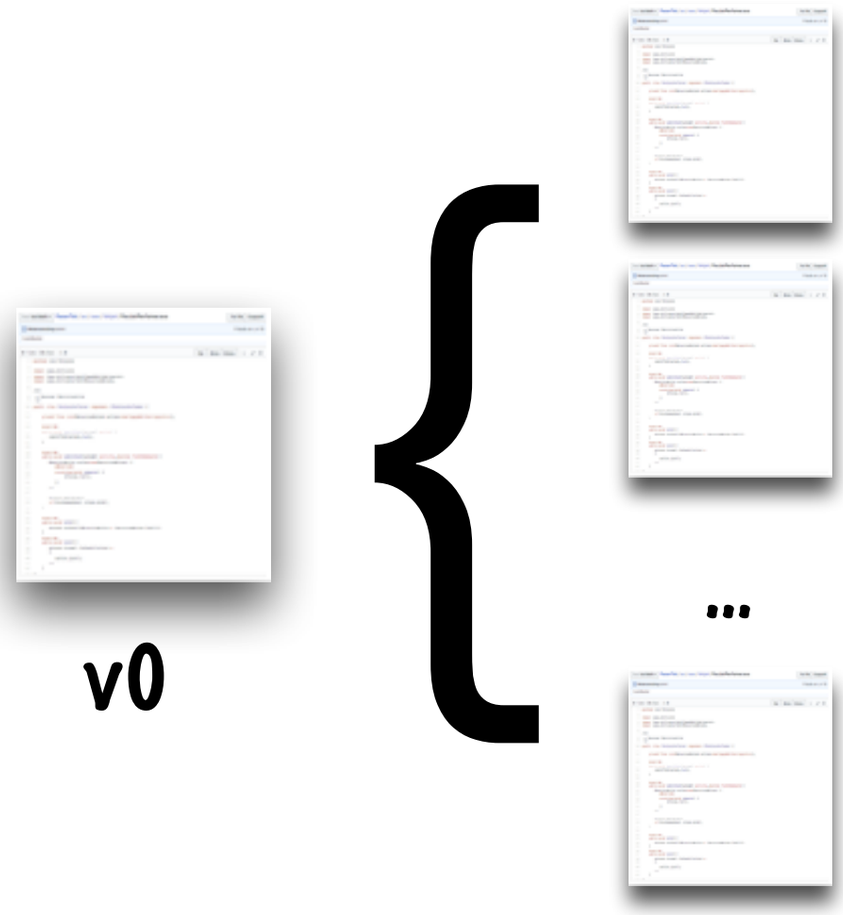
+

Application Programmers

THE CLOC PROGRAM.

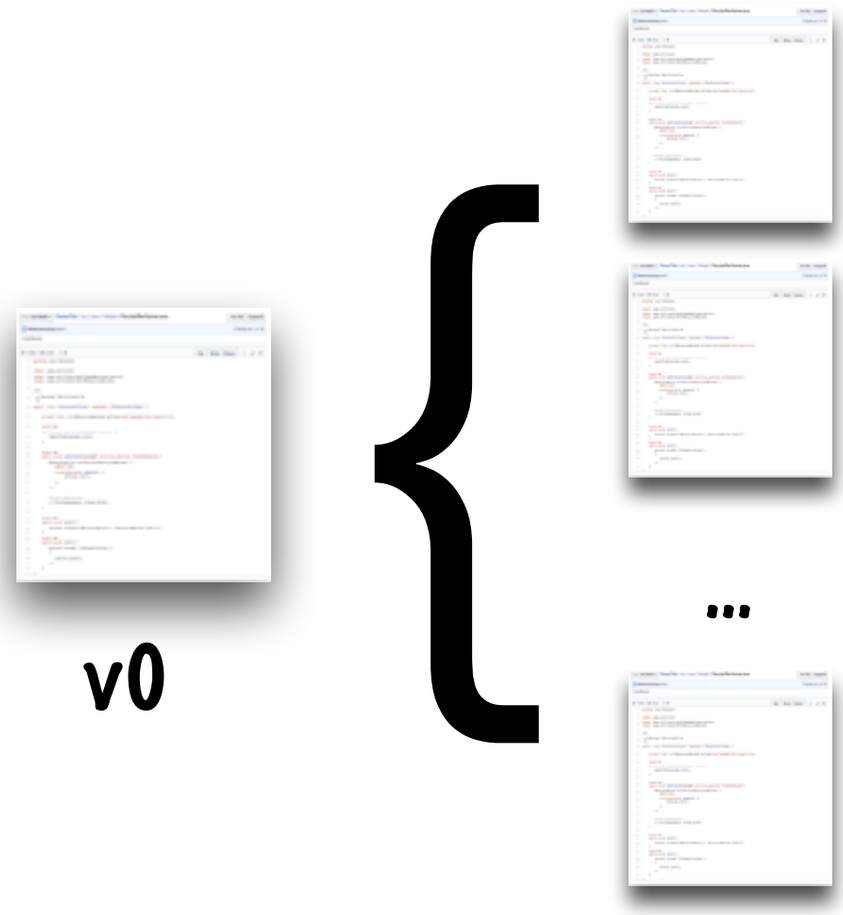
Projects	# LoC	# Commits	# Bottlenecks
Akka	326,341	20,759	1
itemupdown	4,925	2	2
jAcer	4,476	35	2
educational	1,323	7	2
scalatuts	253	5	2
knn	3,099	27	2
doms-transformers	3,714	254	2
ForkAndJoinUtility	127	12	2
Solitaire	1,527	39	2
mywiki	1,920	17	2
MagicSquares	664	153	2
ejisto	12,330	274	2, 3
exhibitor	15,314	701	2, 3, 4
cq4j	5,815	23	2, 3
netflixoss	231,361	1	2, 3
javaOneBR-2012	518	4	2, 3
jadira	46,095	630	3
ecco	5,849	119	3
conflate	934	9	3
bazzar-base	7,766	15	3, 4
DocumentIndexing	1,127	1	4
CSSTProto	10,721	17	4
Fibonacci	79	2	5
Mandelbrot	1,442	30	5
Solitaire	1,527	39	5
Matrices	2,356	15	5
LockedBasedGrid	1,390	1	5
Basic-Blocks	4,821	41	5
warp	15,287	338	6
j7cc	5,110	76	6
lowlatency	3,018	18	6

Understanding Parallelism Bottlenecks

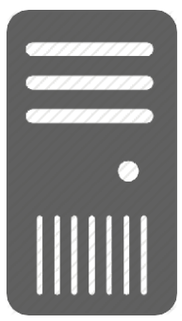


@gustavopinto

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, 2.60GHz, 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

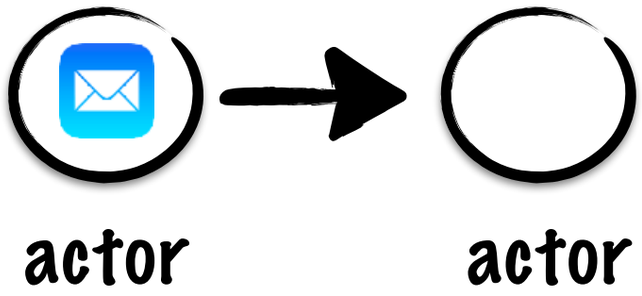
actors exchange, but do not share the same message

Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #1: Centralized pooling

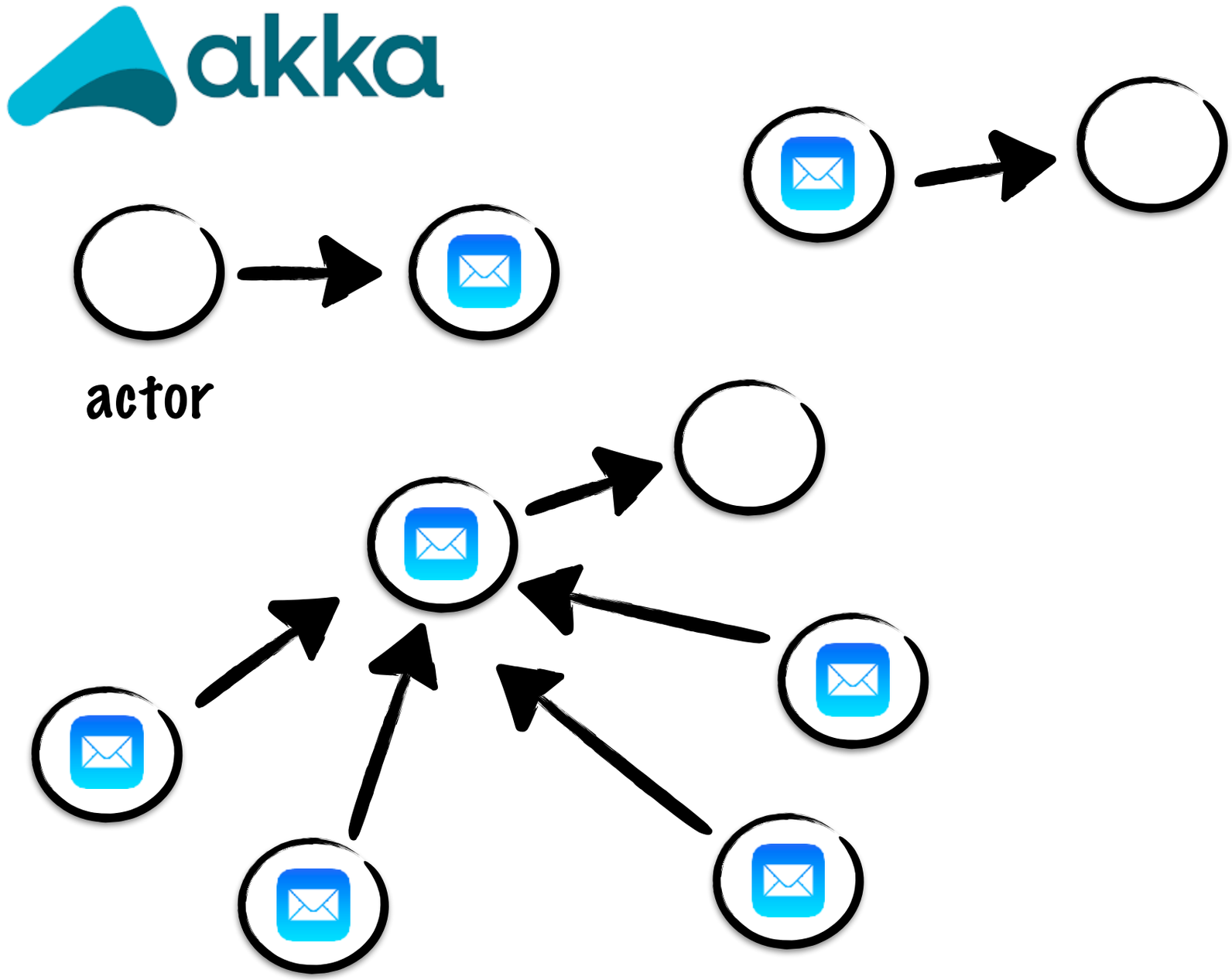
akka



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

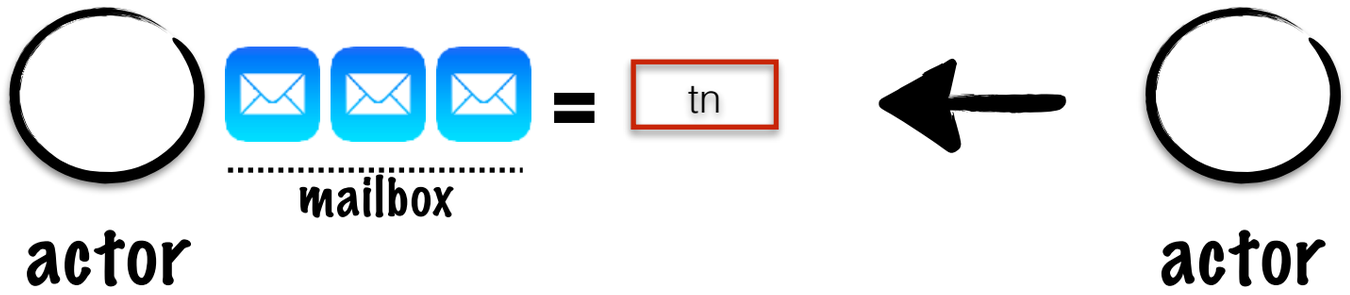
(Centralized)

Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #1: Centralized pooling

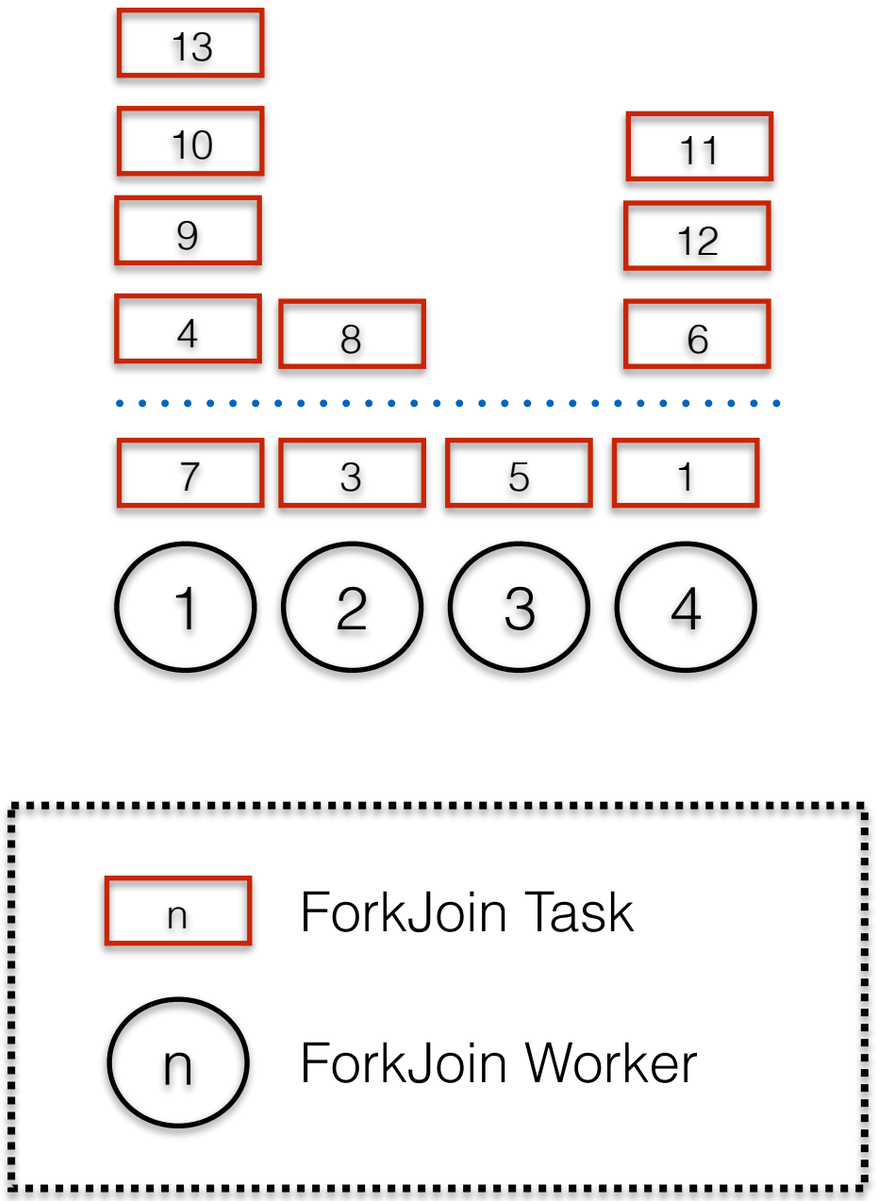
(Centralized)



Overcoming Parallelism Bottlenecks

@gustavopinto

Work Stealing

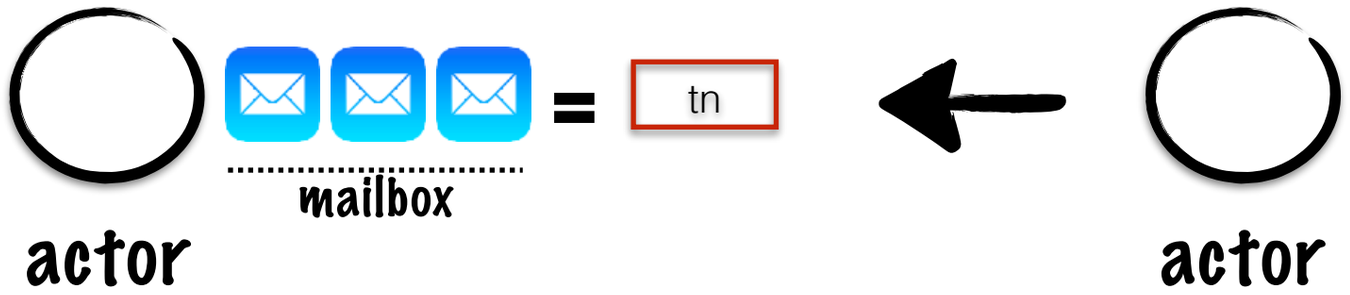


Overcoming Parallelism Bottlenecks

@gustavopinto

Bottleneck #1: Centralized pooling

(Centralized)

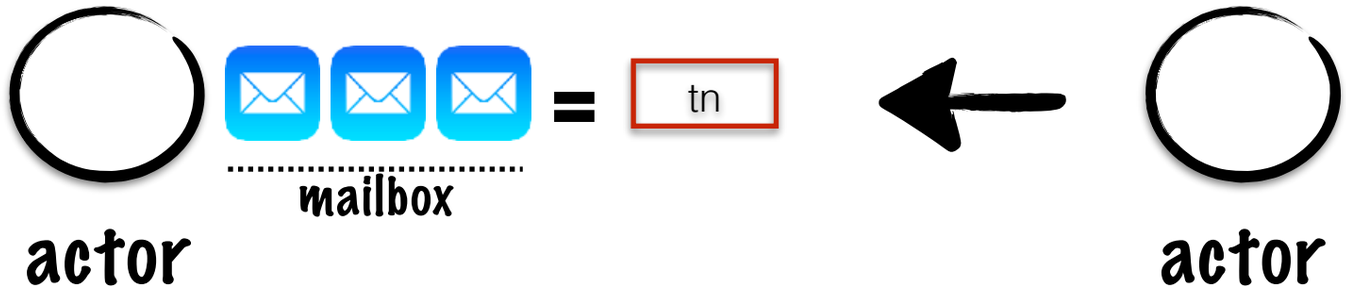


Overcoming Parallelism Bottlenecks

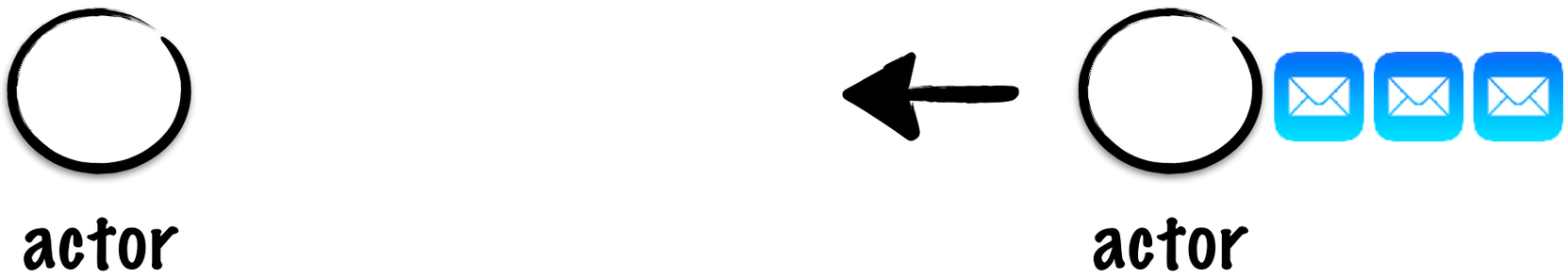
@gustavopinto

Bottleneck #1: Centralized pooling

akka (Centralized)



akka (DEcentralized)

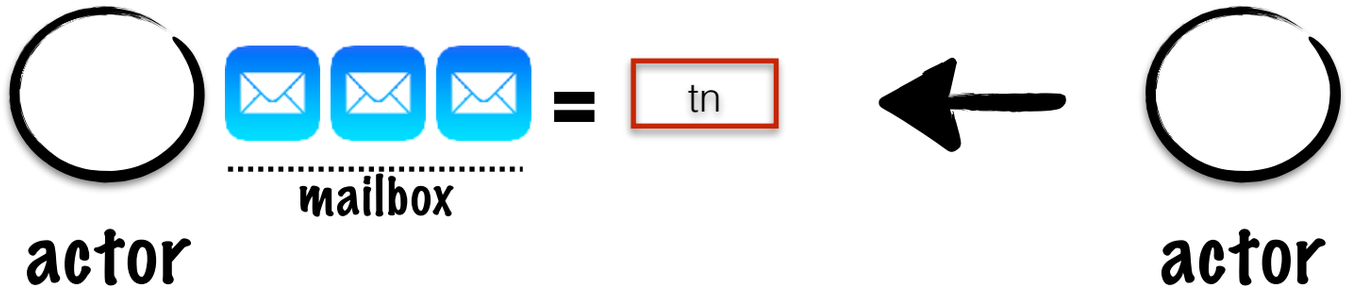


Overcoming Parallelism Bottlenecks

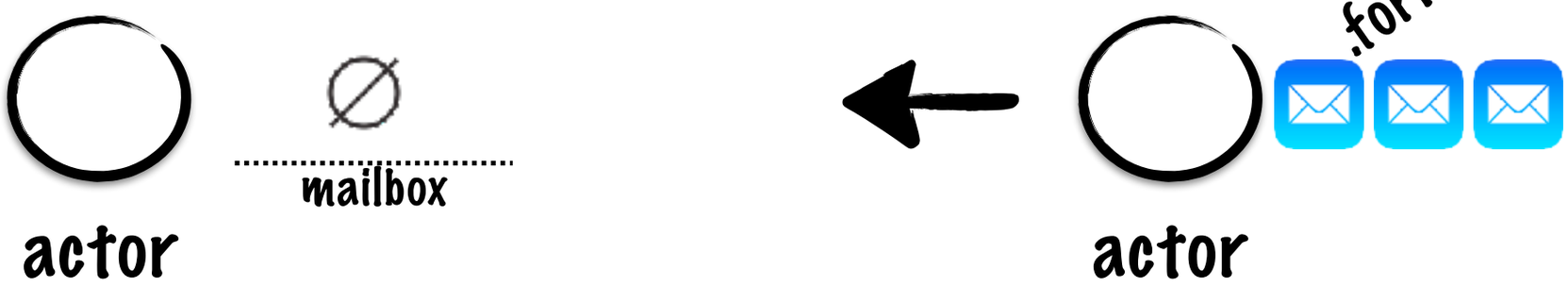
@gustavopinto

Bottleneck #1: Centralized pooling

akka (Centralized)



akka (DEcentralized)

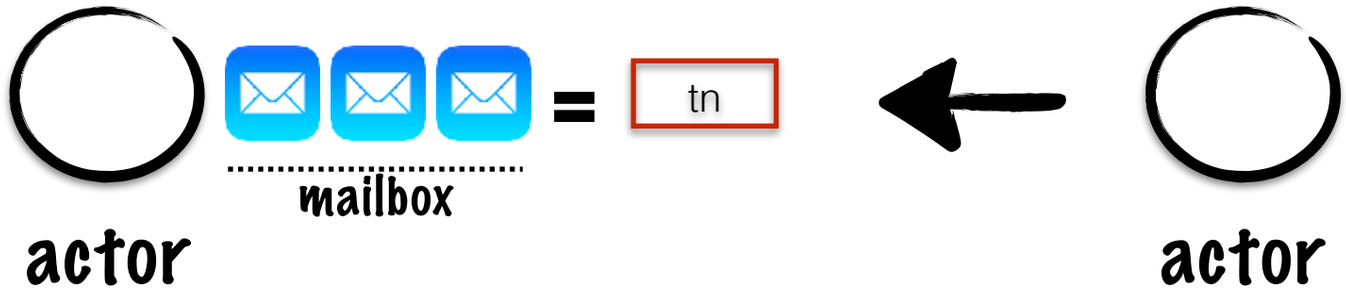


Overcoming Parallelism Bottlenecks

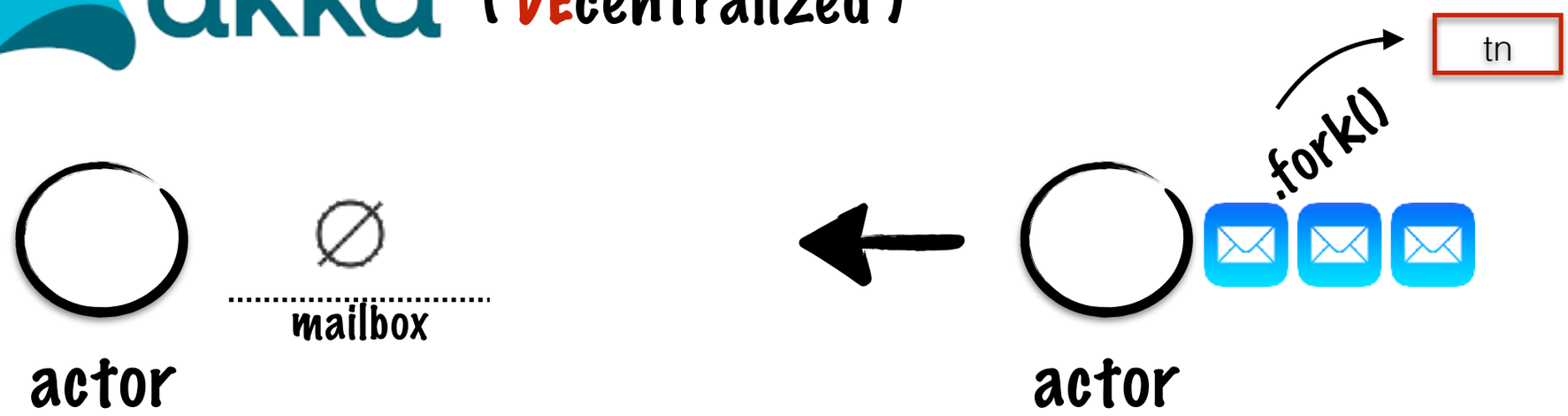
@gustavopinto

Bottleneck #1: Centralized pooling

akka (Centralized)



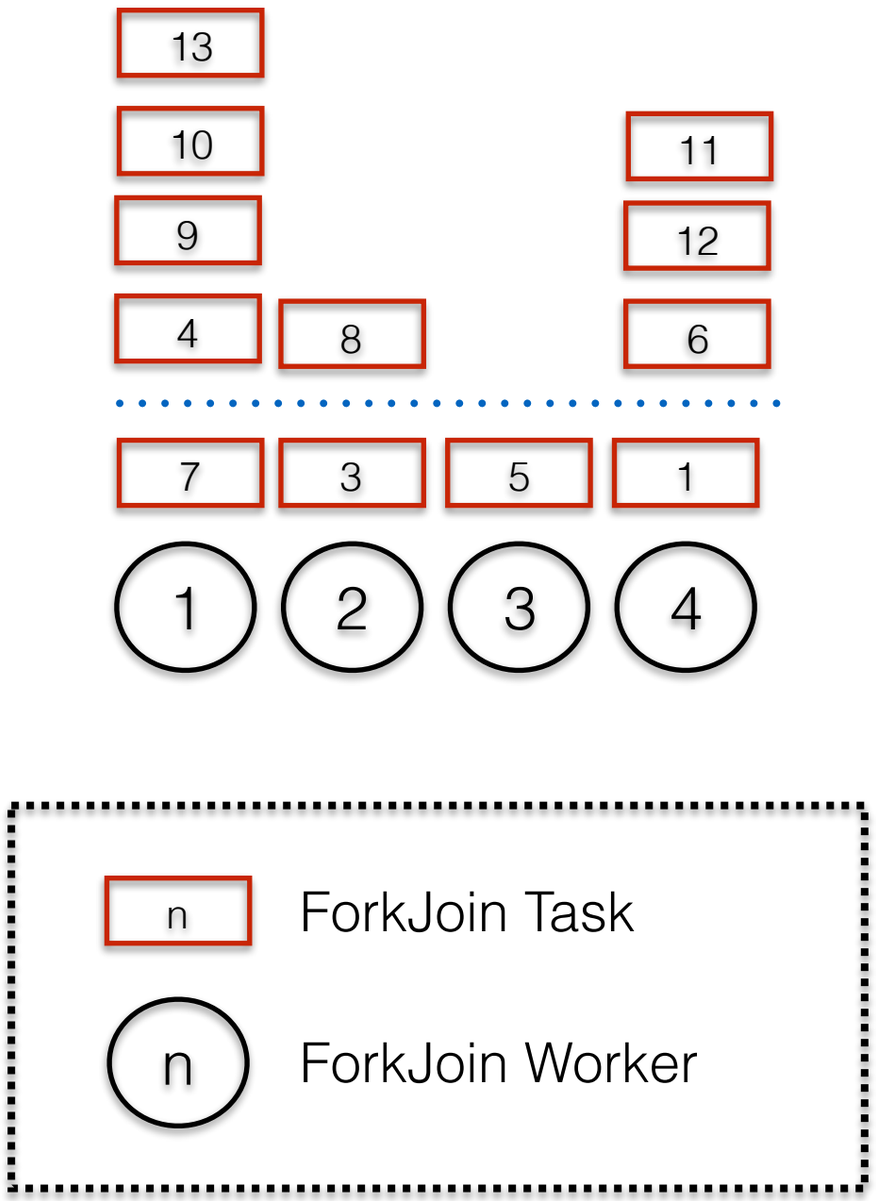
akka (DEcentralized)



Overcoming Parallelism Bottlenecks

@gustavopinto

Work Stealing

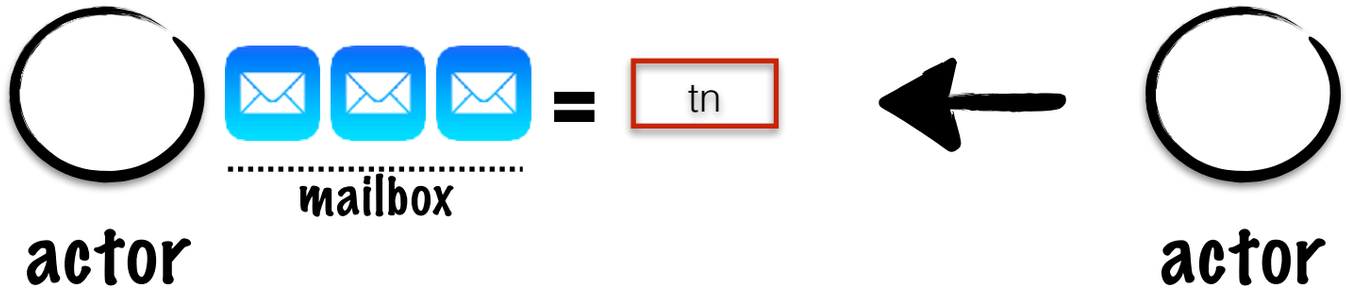


Overcoming Parallelism Bottlenecks

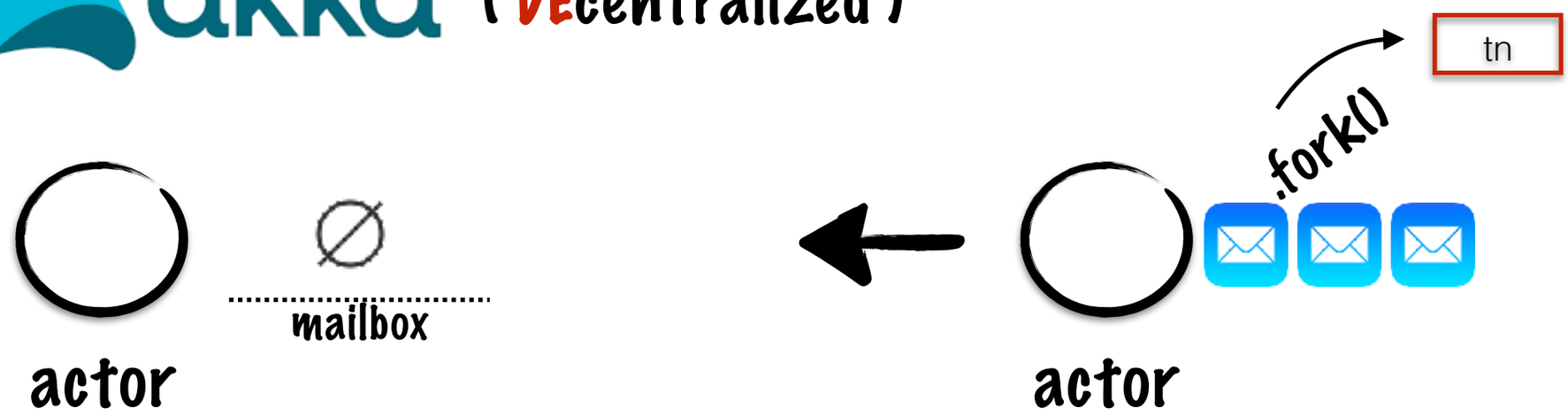
@gustavopinto

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

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

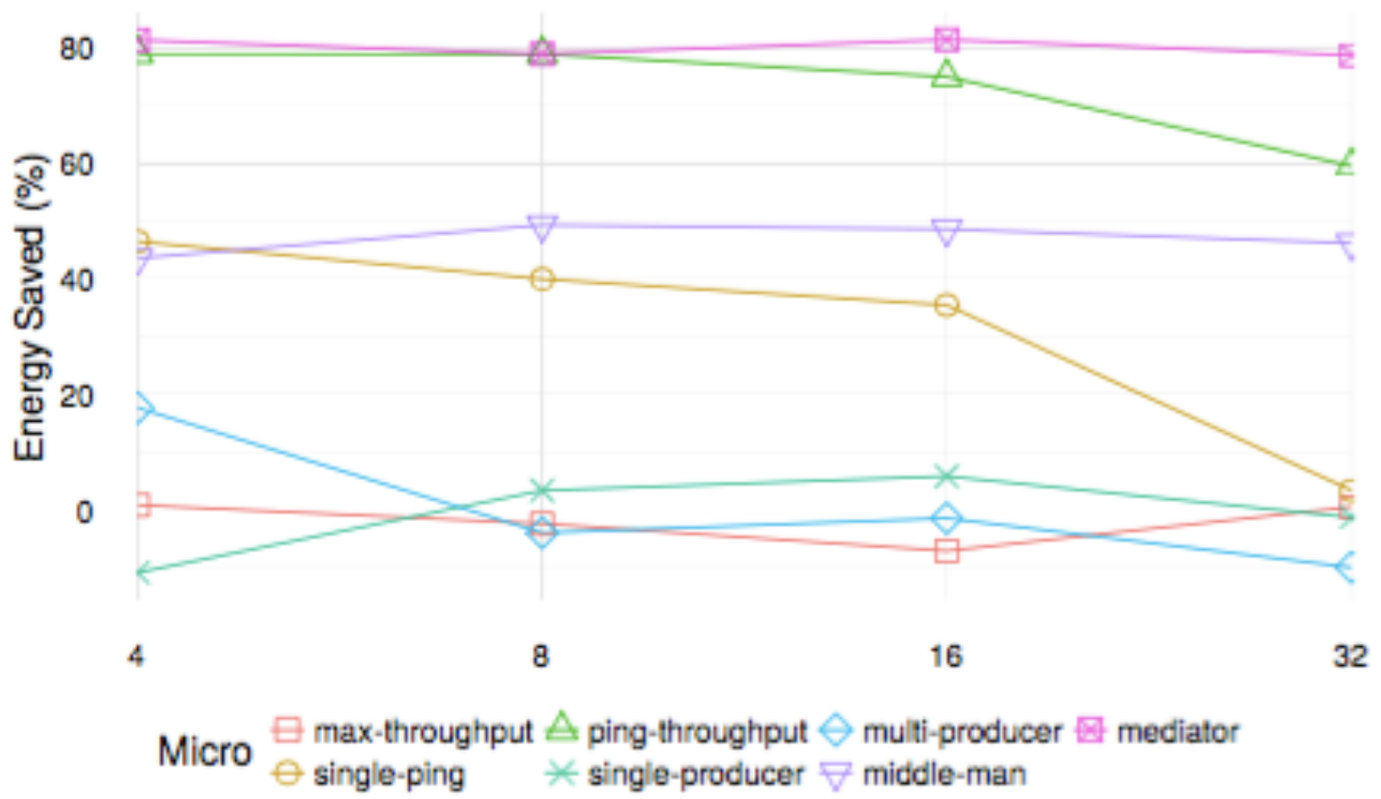
Overcoming Parallelism Bottlenecks

@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

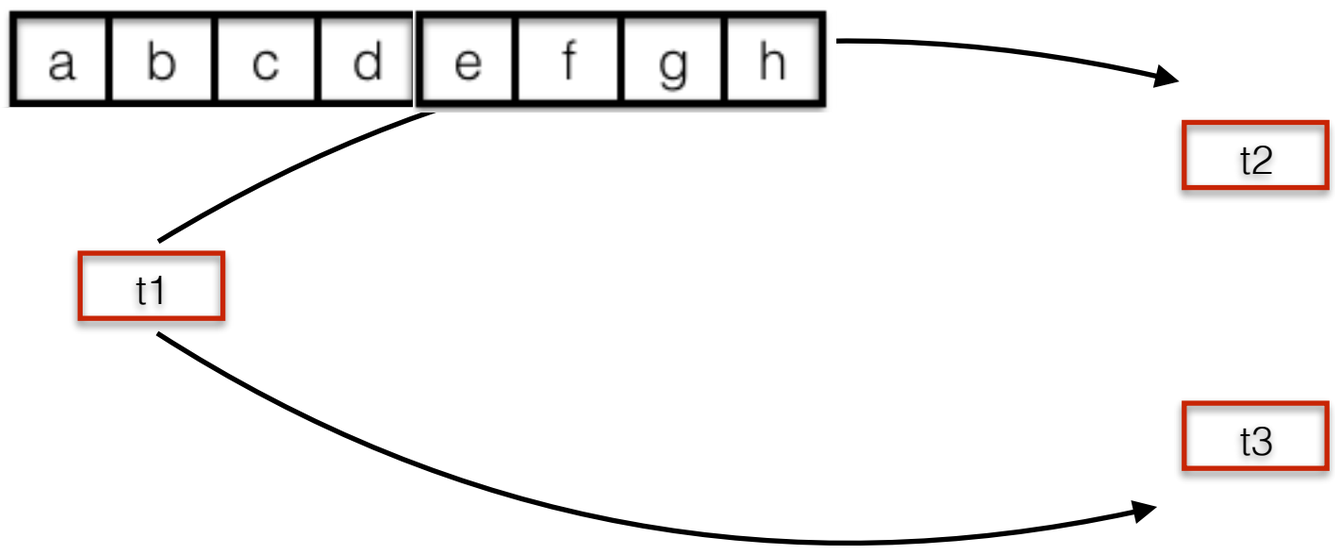
@gustavopinto

Bottleneck #2: Copy on Fork

t1 = 

t2 = first half

t3 = second half



Overcoming Parallelism Bottlenecks

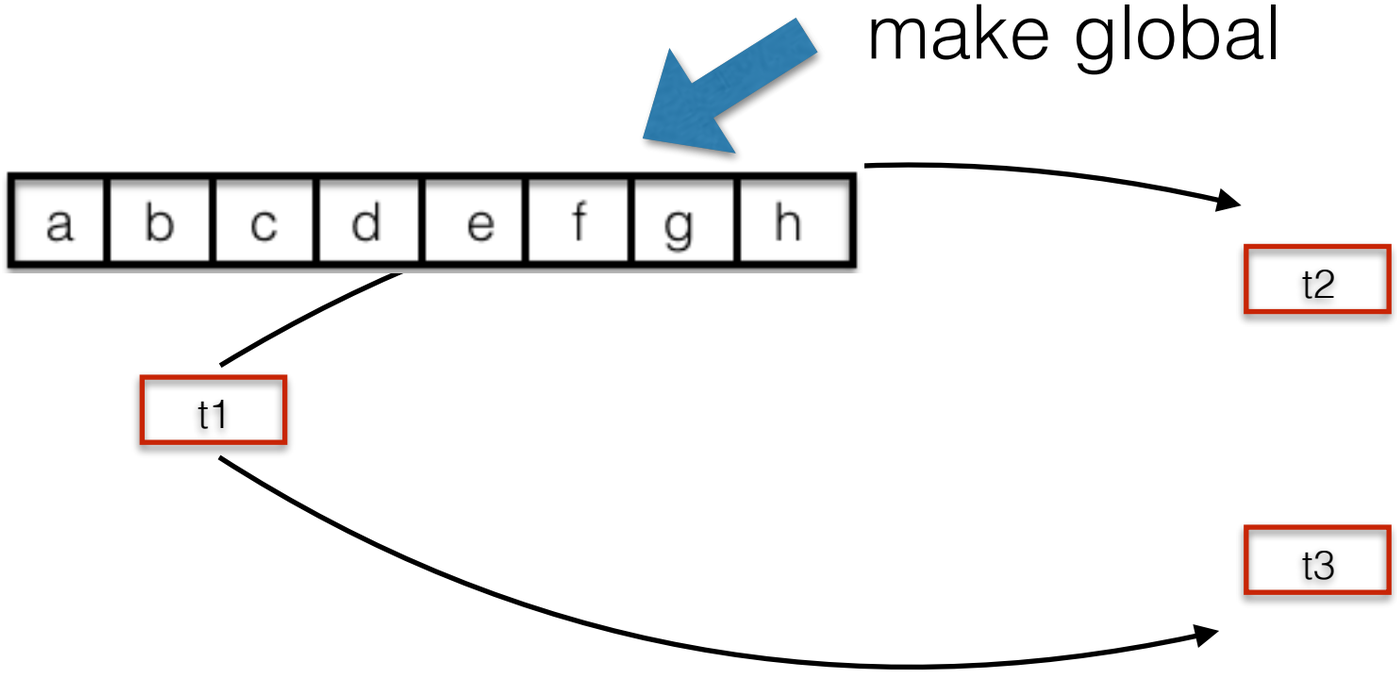
@gustavopinto

Bottleneck #2: Copy on Fork

t1 = 

t2 = first half

t3 = second half



Overcoming Parallelism Bottlenecks

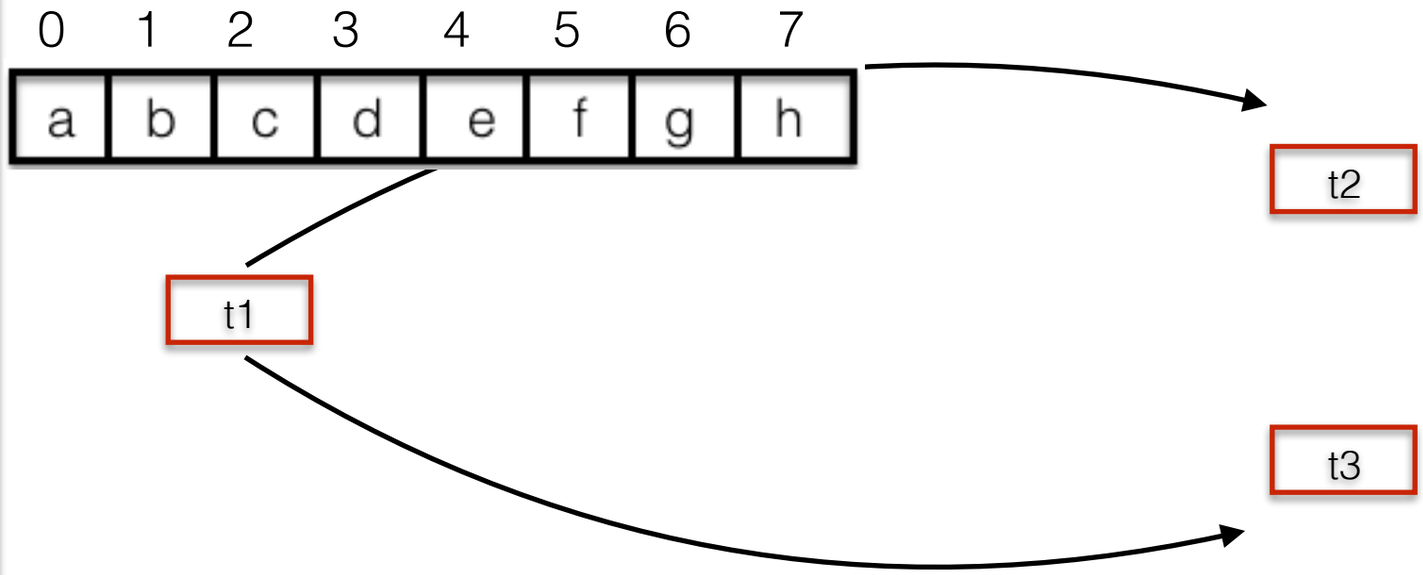
@gustavopinto

Bottleneck #2: Copy on Fork

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

t2 = first half

t3 = second half



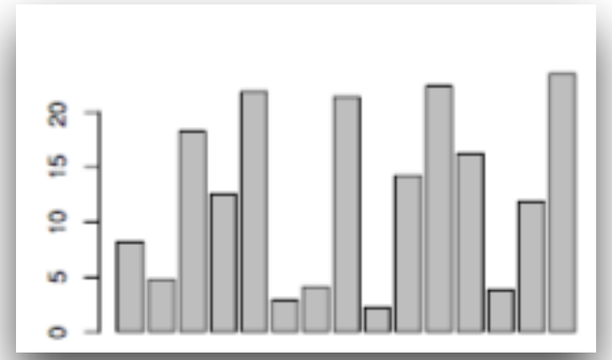
Overcoming Parallelism Bottlenecks

Bottleneck #2: Copy on Fork

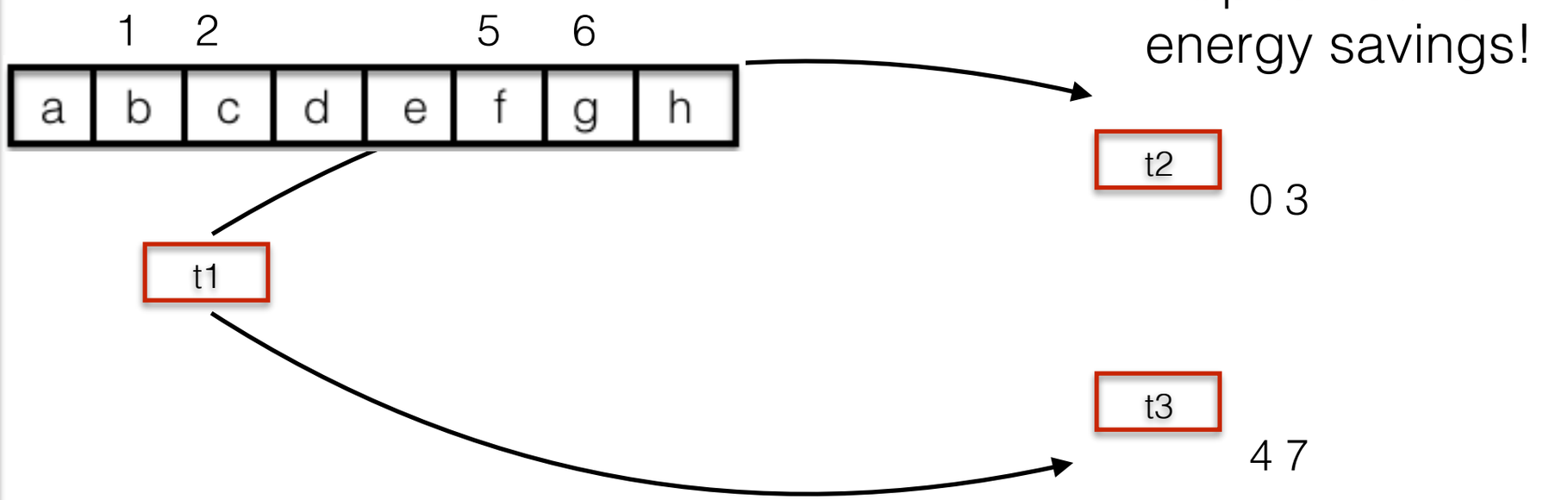
t1 = | a | b | c | d | e | f | g | h |

t2 = first half

t3 = second half



Up to 20% of energy savings!

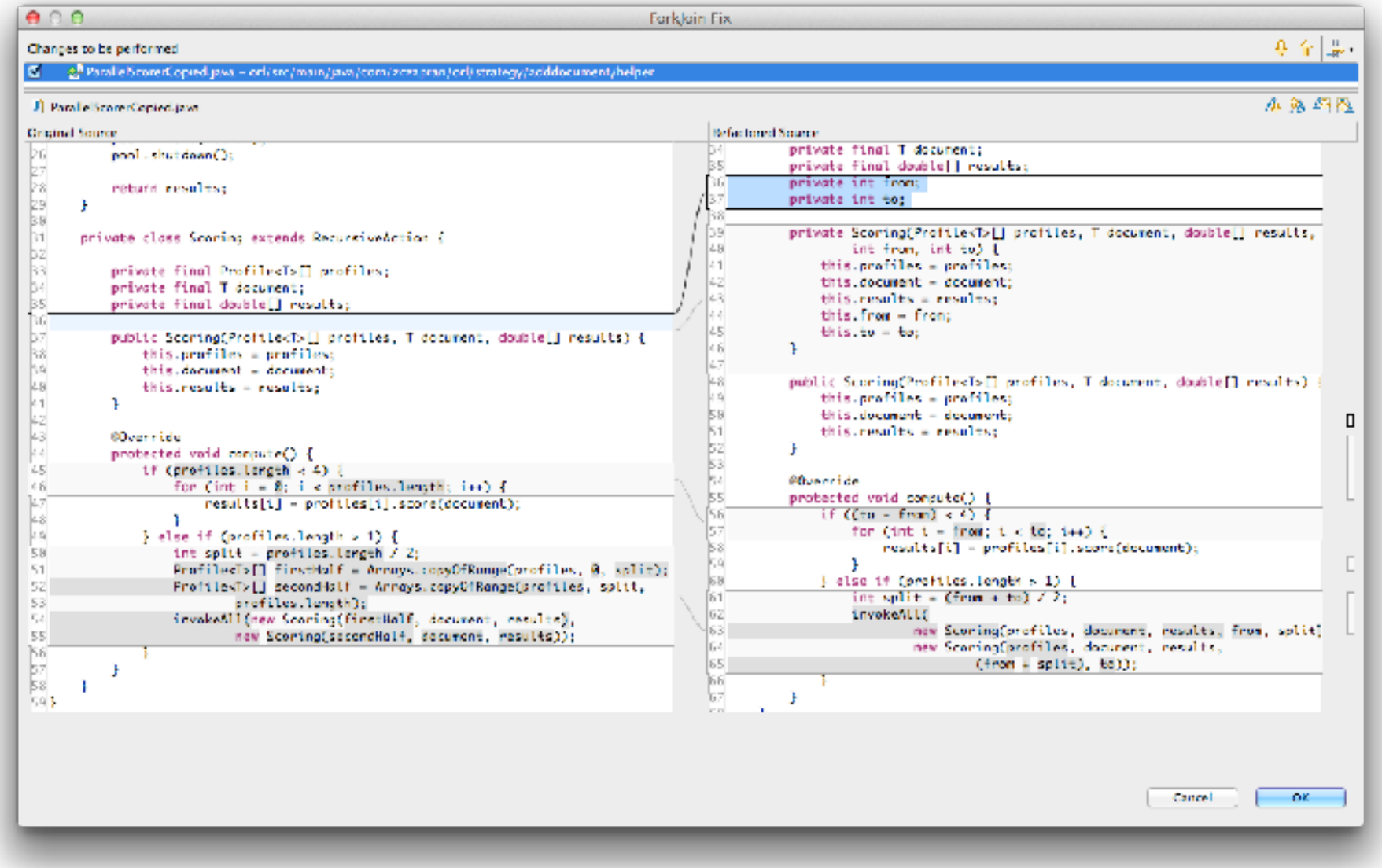


@gustavopinto

Overcoming Parallelism Bottlenecks

@gustavopinto

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/n

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

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

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

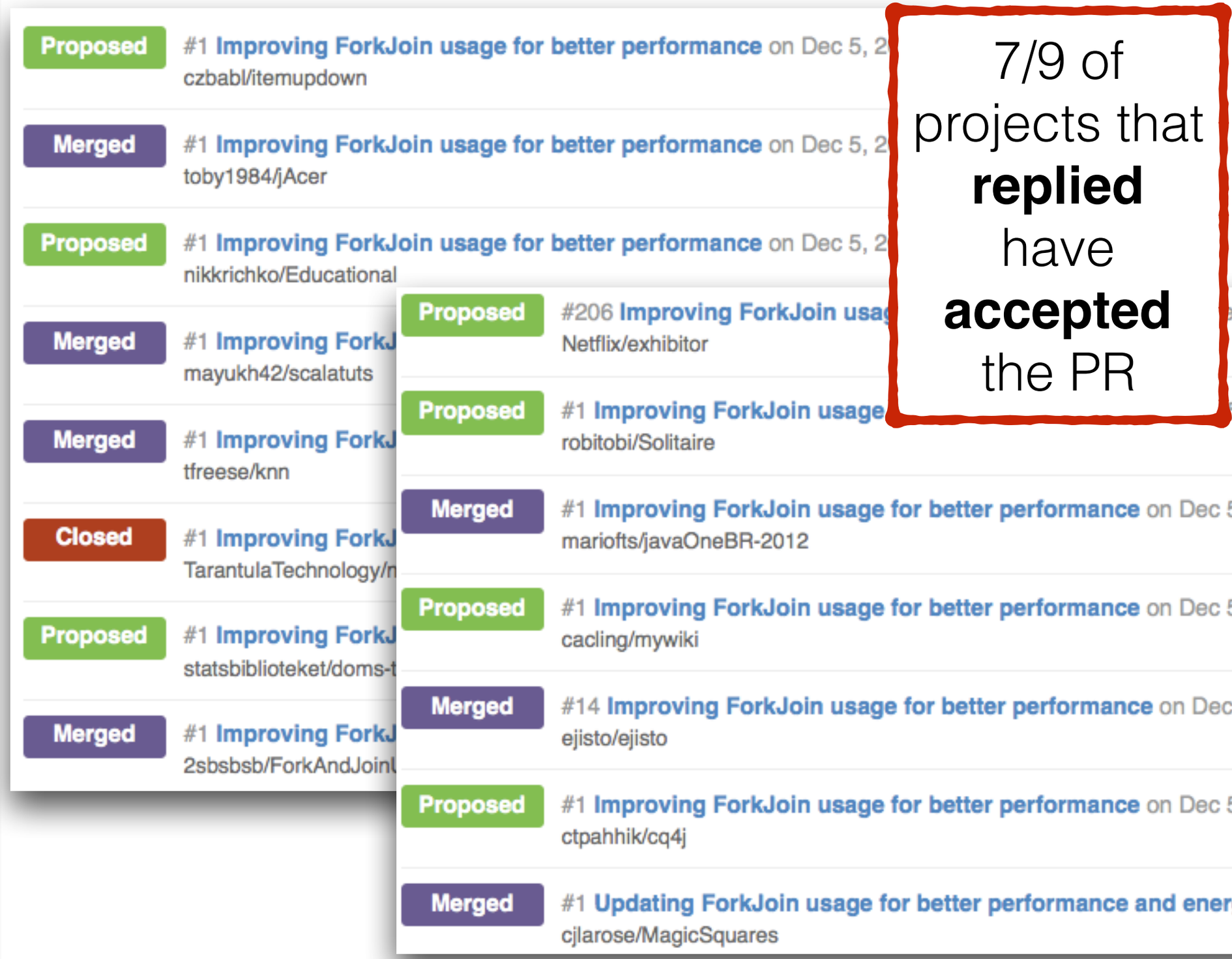
Proposed #1 **Improving ForkJoin usage for better performance** on Dec 5
ctpahhik/cq4j

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

Overcoming Parallelism Bottlenecks

@gustavopinto

Patching Bottleneck #2: Copy on Fork



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


WHAT'S
NEXT

Next?

Better tool support

Refactoring

Testing

Debugging

Visualization

Estimation

....

Books, cookbooks, guidelines

Energy Efficiency: A New Concern for Application Software Developers

Gustavo Pinto
Federal Institute of Pará
gustavo.pinto@ifpa.edu.br

Fernando Castor
Federal University of Pernambuco
castor@cin.ufpe.br

ABSTRACT

Energy efficiency is a problem that must be addressed at all levels of the software stack. However, developing energy-efficient software is not an easy task. In this paper we argue that this is mostly due to two main problems: the lack of knowledge and the lack of tools. These problems prevent software developers from identifying, refactoring, fixing, and removing energy consumption hotspots. We review how current research in the area of software engineering is tackling these two problems. Furthermore, based on an investigation on the problems faced by energy-aware developers, we discuss avenues for future research in the area.

Keywords

Software Energy Consumption, Lack of Knowledge, Lack of Tools.

1. INTRODUCTION

The prevalence and ubiquity of mobile computing platforms such as smartphones, tablets, smartwatches, and smartglasses, changed the way people use and interact with software. In particular, these platforms share a common yet challenging requirement: they are battery-driven. As users interact with them, they tend to be less available, since even simple well-optimized operations (e.g., texting a friend) consume energy. At the same time, wasteful, poorly-optimized software can deplete a device's battery much faster than necessary. Heavy resource usage has been shown to be one of the reasons leading to poor apps reviews in online app stores [19].

This concern, however, pertains not only to mobile platforms. Big players of the software industry are also reaching the same conclusion, as stated in one of the few energy efficient software development guides: "*Even small inefficiencies in apps add up across the system, significantly affecting battery life, performance, responsiveness, and temperature*"[1]. Corporations that maintain data centers struggle with soaring energy costs. These costs can be attributed in part to overprovisioning with severs constantly operating under their maximum capacity (e.g., America's data centers are wasting huge amount of energy [13]), and to the developers of the apps running on these data centers generally not taking energy into consideration [33].

Unfortunately, during the last decades, little attention has been placed on creating techniques, tools, and processes to empower software developers to better understand and use energy resources. As a consequence, software developers still lack textbooks, guidelines, courses, and tools to refer to when dealing with energy consumption issues [33, 42]. Moreover, most of the research that connects computing and energy efficiency has concentrated on the lower levels of the hardware and software stack. However, recent studies show that these lower level solutions do not capture the whole picture [2, 9, 22], when it comes to energy consumption. Although software systems do not consume energy themselves, they affect hardware utilization, leading to indirect energy consumption.

1.1 How is software related to energy consumption?

In general, energy consumption E is an accumulation of power dissipation P over time t, that is, $E = P \times t$. Power P is measured in watts, whereas energy E is measured in joules. As an example, if one operation takes 10 seconds to complete and dissipates 5 watts, it consumes 50 joules of energy. In particular, when taking about software energy consumption, we should pay attention to:

- a given software under execution,
- on a given hardware platform,
- on a given context,
- during a given time.

To understand the importance of a *hardware platform*, consider an application that uses network. Any commodity smartphone nowadays supports, at least, WiFi, 3G, and 4G. A recent study observed that 3G can consume about 1.7x more energy than WiFi, whereas 4G can consume about 1.3x

[1] https://developer.apple.com/library/content/documentation/Performance/Conceptual/power_efficiency_guidelines_osx/index.html#//apple_ref/doc/uid/TP40013929

© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

CACM'2017

My personal, biased view of

The Last Five Years of Energy Consumption Research

Gustavo Pinto

@gustavopinto

gpinto@ufpa.br

UNIVERSIDADE FEDERAL DO PARÁ