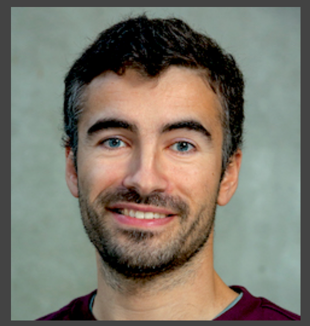
7. Green SE – Research

Sustainable Software Engineering
CS4575


Luís Cruz
L.Cruz@tudelft.nl


Carolin Brandt
C.E.Brandt@tudelft.nl


Enrique Barba Roque
E.BarbaRoque@tudelft.nl

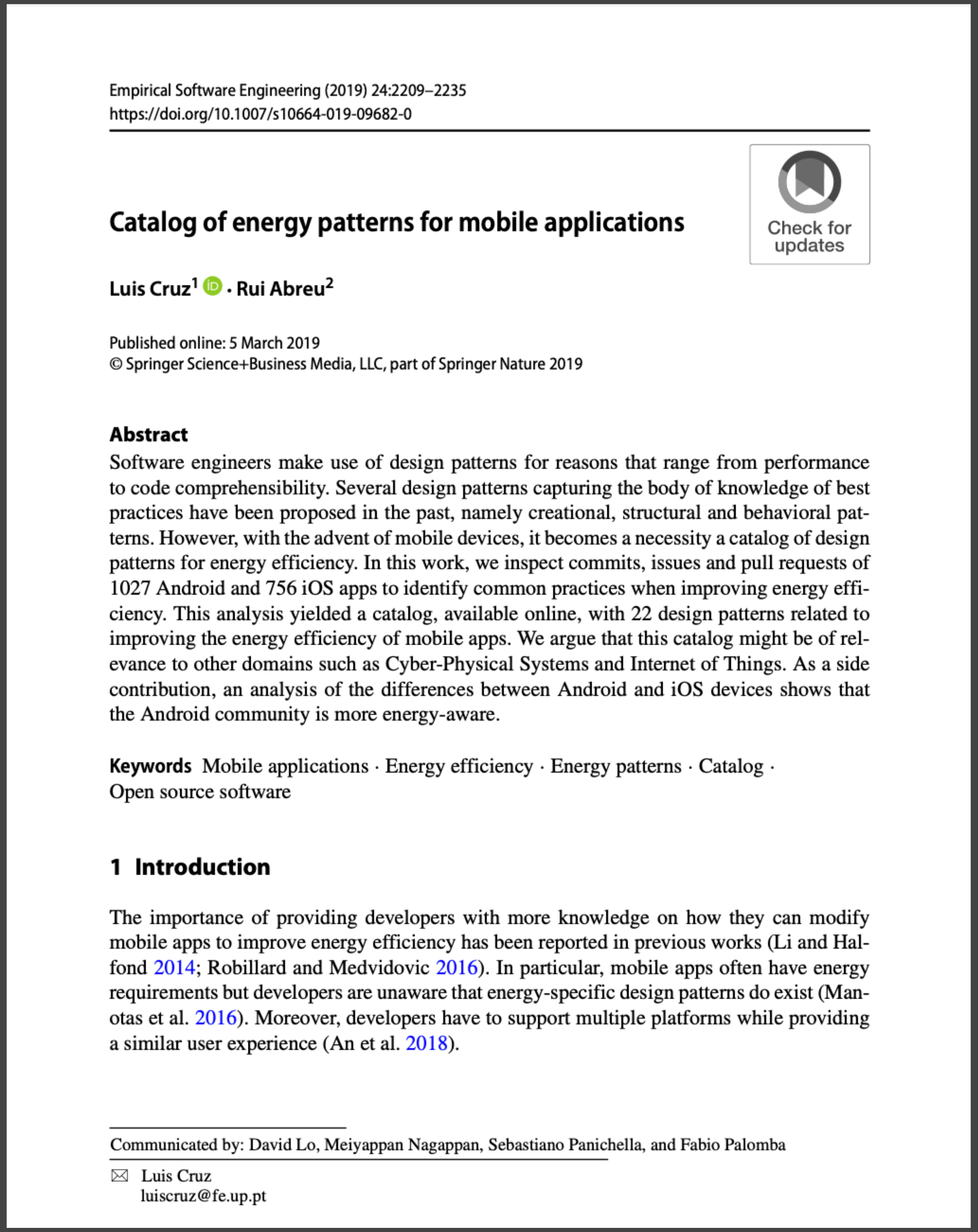
1. Energy patterns for mobile apps
2. Carbon-aware datacenters
3. Energy Regression Testing
4. Debugging Energy with Docker images
5. Energy Efficiency vs Code Quality

- While learning about these works, try to be critical about them and find their pitfalls.

- We have seen that **measuring energy consumption is not trivial**
- It is **not practical** considering that developers have **other priorities** above energy efficiency
- At the same time, every now and then there are some efforts to improve energy efficiency in some cases. This is **time consuming** and **requires expertise**.

- **How can we reuse these efforts?**

Energy Patterns for Mobile Apps



Empirical Software Engineering (2019) 24:2209–2235
https://doi.org/10.1007/s10664-019-09682-0

Check for updates

Catalog of energy patterns for mobile applications

Luis Cruz[1] · Rui Abreu[2]

Published online: 5 March 2019

Abstract
Software engineers make use of design patterns for reasons that range from performance to code comprehensibility. Several design patterns capturing the body of knowledge of best practices have been proposed in the past, namely creational, structural and behavioral patterns. However, with the advent of mobile devices, it becomes a necessity a catalog of design patterns for energy efficiency. In this work, we inspect commits, issues and pull requests of 1027 Android and 756 iOS apps to identify common practices when improving energy efficiency. This analysis yielded a catalog, available online, with 22 design patterns related to improving the energy efficiency of mobile apps. We argue that this catalog might be of relevance to other domains such as Cyber-Physical Systems and Internet of Things. As a side contribution, an analysis of the differences between Android and iOS devices shows that the Android community is more energy-aware.

Keywords Mobile applications · Energy efficiency · Energy patterns · Catalog · Open source software

1 Introduction

The importance of providing developers with more knowledge on how they can modify mobile apps to improve energy efficiency has been reported in previous works (Li and Halfond 2014; Robillard and Medvidovic 2016). In particular, mobile apps often have energy requirements but developers are unaware that energy-specific design patterns do exist (Manotas et al. 2016). Moreover, developers have to support multiple platforms while providing a similar user experience (An et al. 2018).

Communicated by: David Lo, Meiyappan Nagappan, Sebastiano Panichella, and Fabio Palomba

✉ Luis Cruz
luiscruz@fe.up.pt

https://tqrg.github.io/energy-patterns/

Methodology

1. App Collection
   - F-droid
   - Curated Lists

   1783 apps

2. Collect Changes With Potential Interest

   `/.*(energy|power|battery).*/`

   6028 changes

3. Manual Refinement of Subjects of Interest

   1563 changes

4. Thematic Analysis

   431 reusable changes

5. **Catalog of Energy Patterns**

   22 patterns

Thematic Analysis

1. Familiarization with data

2. Generating initial labels

3. Reviewing themes

4. Defining and naming themes

- **Energy Pattern**: *design pattern to improve energy efficiency..*
- 22 energy patterns.
- Each pattern is described by **Context**, **Solution**, **Example**, **References** from literature, and **Occurences** (links to code changes from git repositories).

tqrg.github.io

Energy Patterns for Mobile Apps

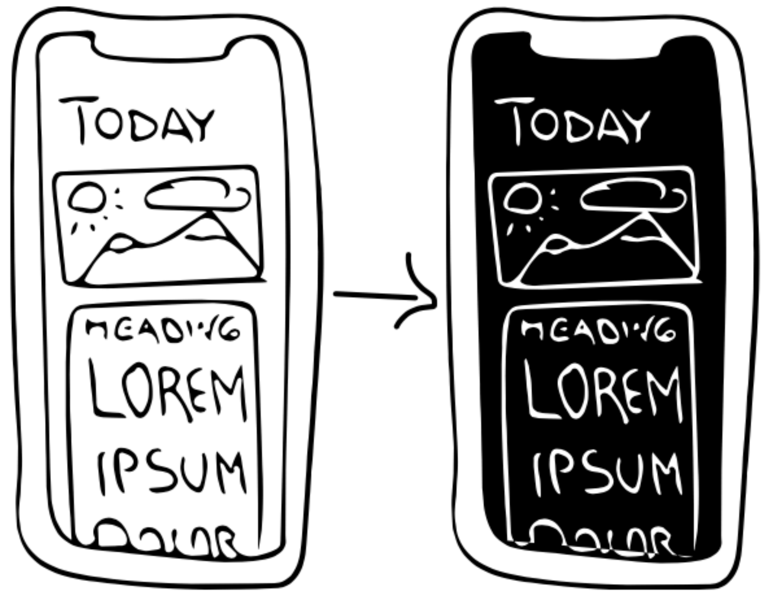
A visualization with prevalence and co-occurence of patterns can be found here.

News This catalog has been **accepted** to the *Journal of Empirical Software Engineering*. Check out the **preprint**.

← show all patterns

Dark UI Colors

Provide a dark UI color theme to save battery on devices with AMOLED screens.



Context

Screen is one of the major source of power consumption on mobile devices. Apps that require heavy usage of screen (e.g., reading apps) can have a big impact on battery life.

Solution

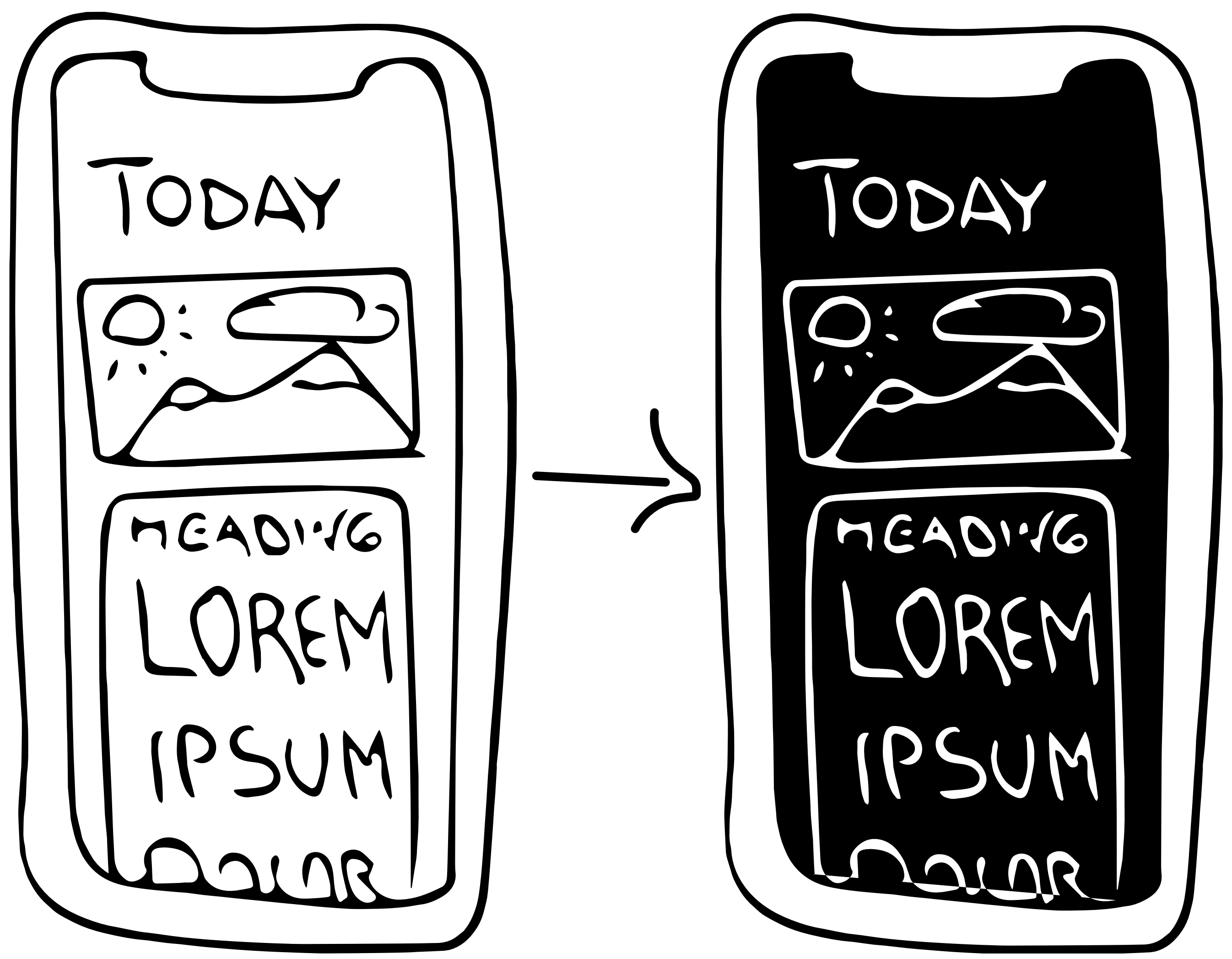
Provide a UI with dark background colors. This is particularly beneficial for mobile devices with AMOLED screens, which are more energy efficient when displaying dark colors. In some cases, it might be reasonable to allow users to choose between a light and a dark theme. The dark theme can also be activated using a special trigger (e.g., when battery is running low).

Display a menu

https://tqrg.github.io/energy-patterns

Dark UI Colors

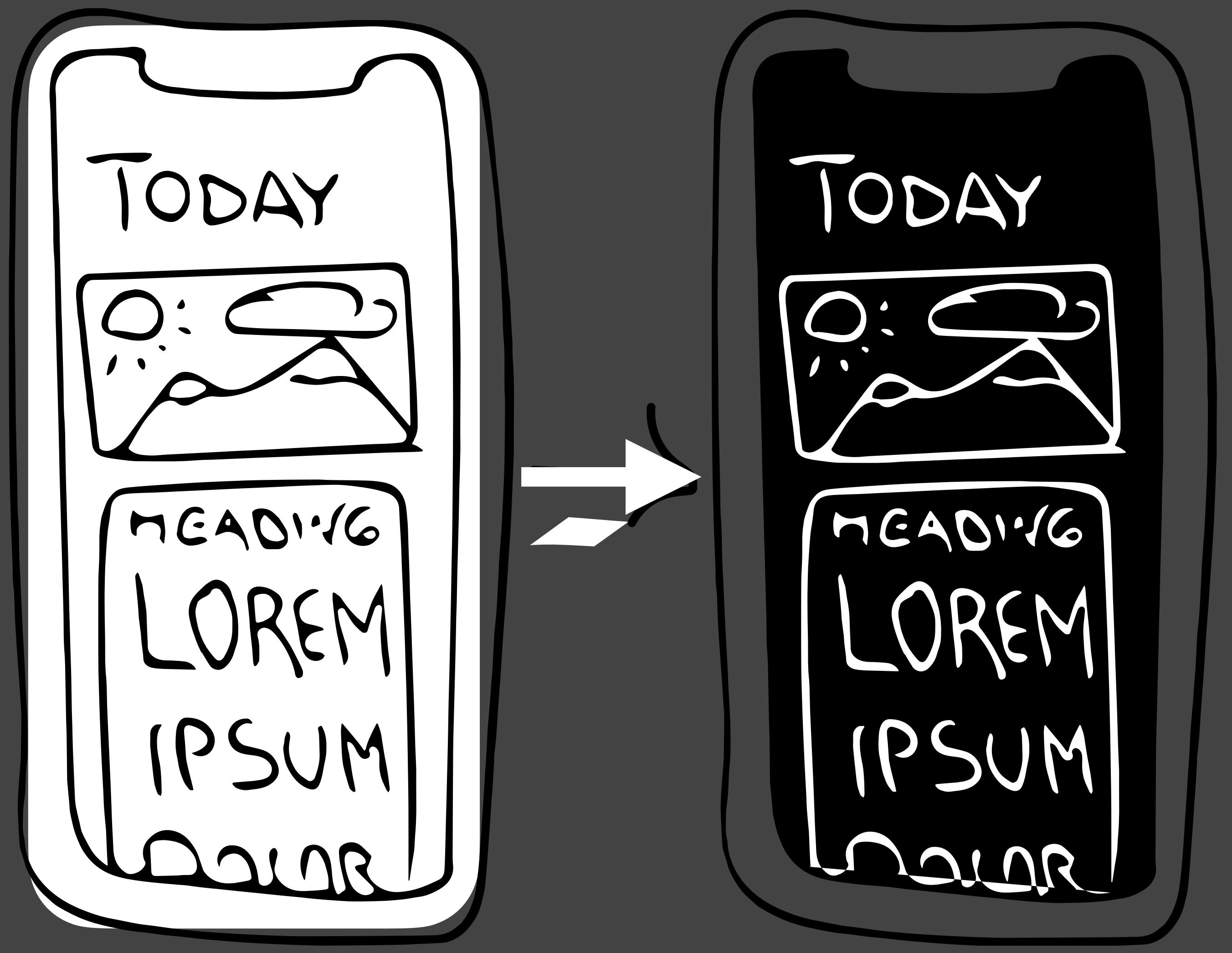
Provide a dark UI color theme to save battery on devices with AMOLED screens.



- **Context:** […] Apps that require heavy usage of screen can have a substantial negative impact on battery life.
- **Solution:** Provide a UI theme with dark background colors. […]
- **Example:** In a reading app, provide a theme with a dark background using light colors to display text. […]

Dark UI Colors

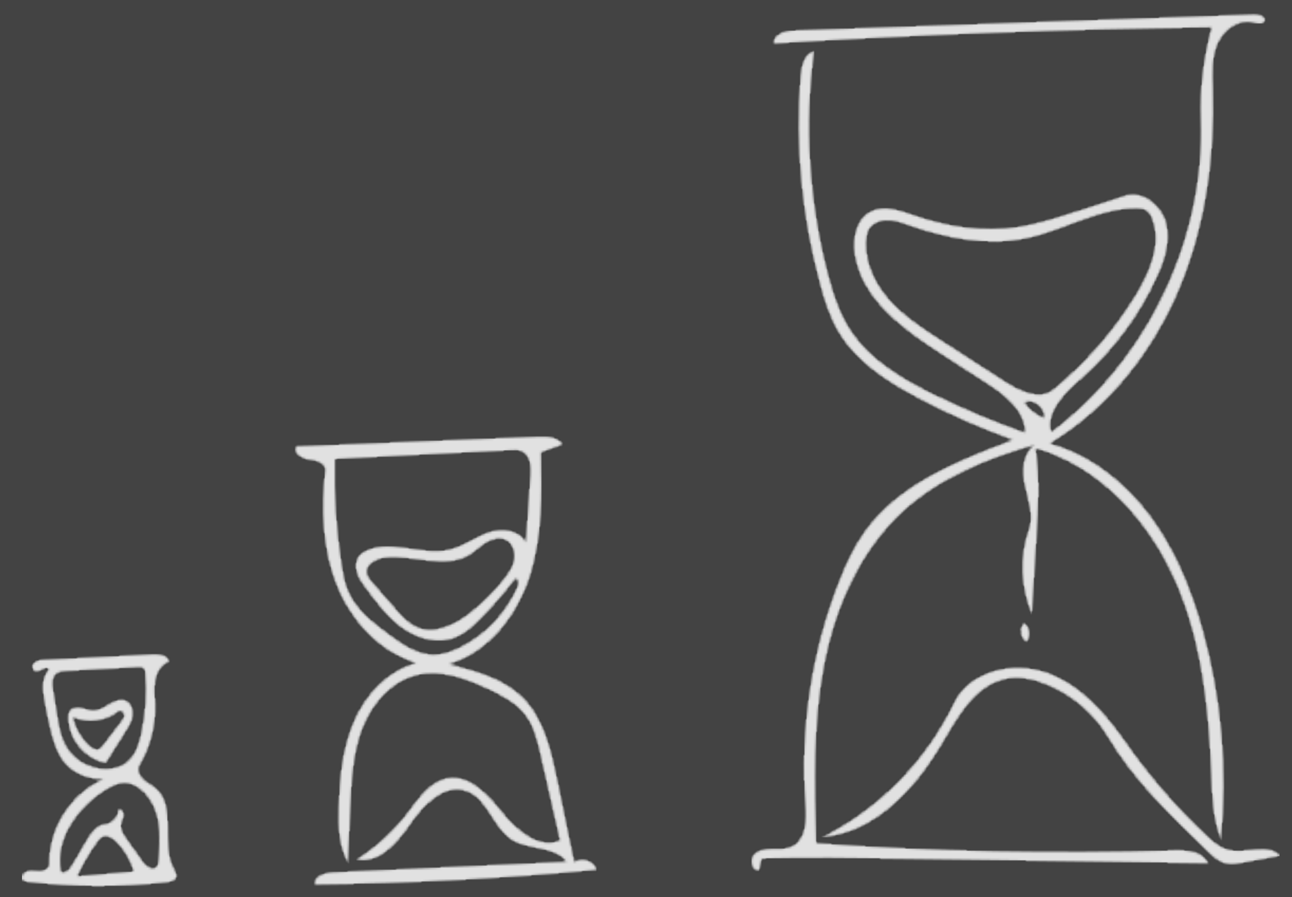
Provide a dark UI color theme to save battery on devices with AMOLED screens.



- **Context:** […] Apps that require heavy usage of screen can have a substantial negative impact on battery life.
- **Solution:** Provide a UI theme with dark background colors. […]
- **Example:** In a reading app, provide a theme with a dark background using light colors to display text. […]

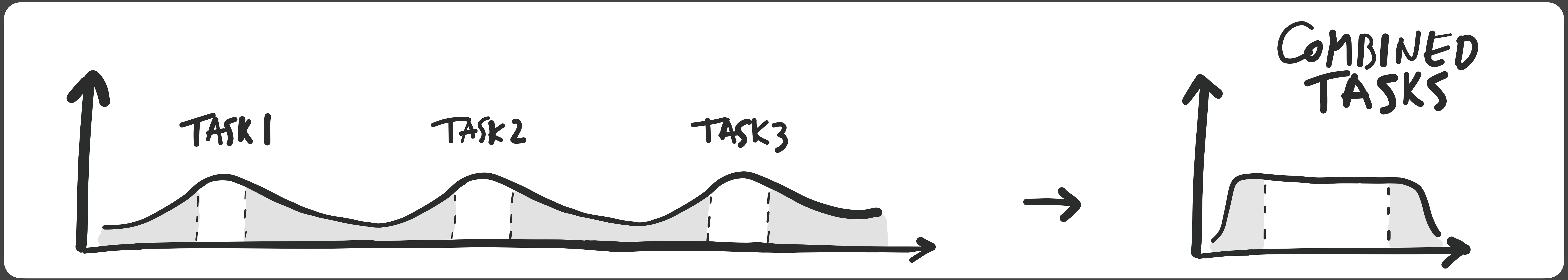
Dynamic Retry Delay

Whenever an attempt to access a resource fails, increase the time interval before retrying.



- **Context:** […] In a mobile app, when a given resource is unavailable, the app will unnecessarily try to connect the resource for a number of times, leading to unnecessary power consumption.
- **Solution:** Increase retry interval after each failed connection. […]
- **Example:** Consider a mobile app that provides a news feed and the app is not able to reach the server to collect updates. […] use the Fibonacci series to increase the time between attempts.

Batch Operations

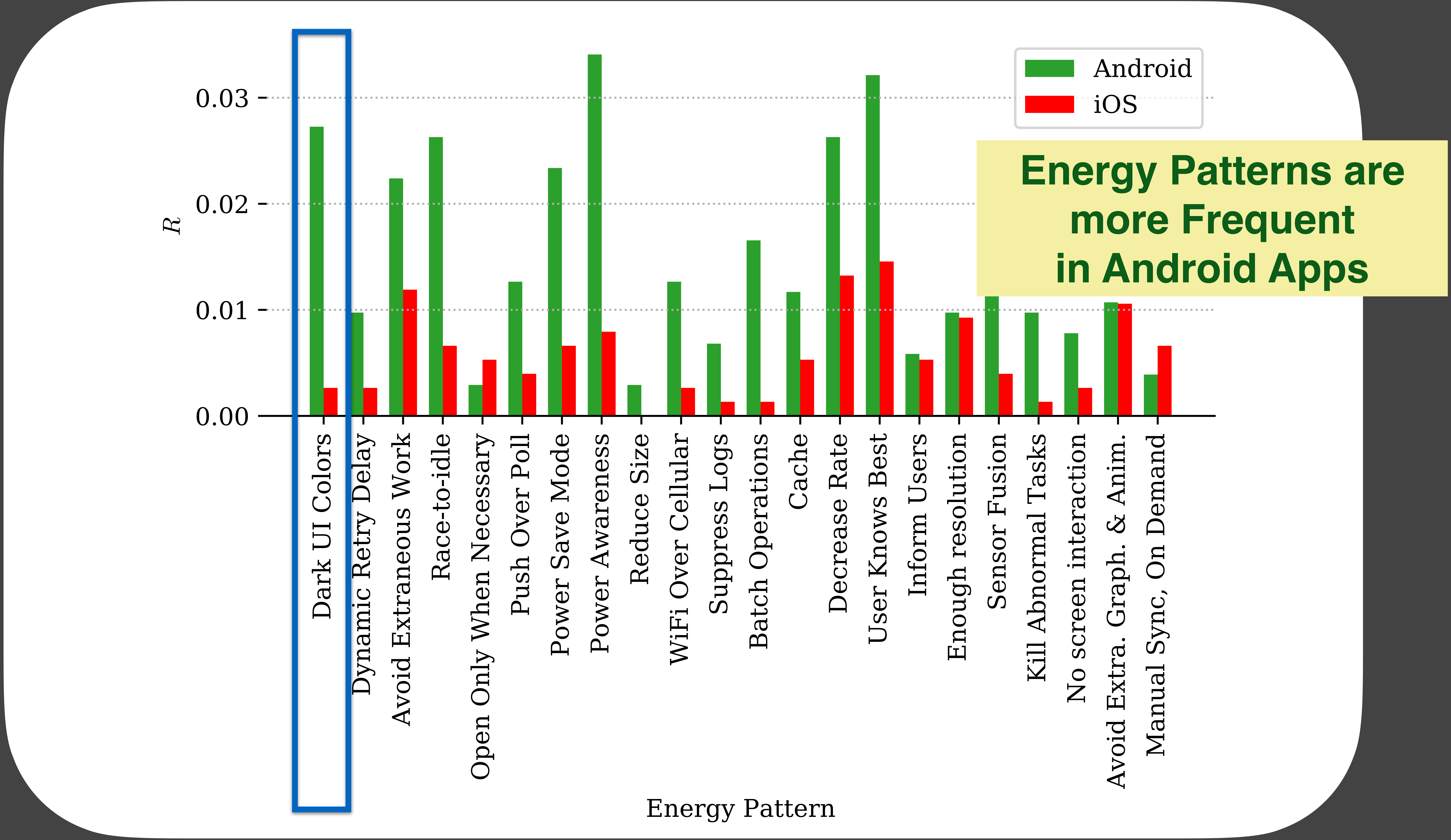


- **Context:** Executing operations separately leads to extraneous tail energy consumptions
- **Solution:** Bundle multiple operations in a single one. […]
- **Example:** Use system provided APIs to schedule background tasks. These APIs, guarantee that device will exit sleep mode only when there is a reasonable amount of work to do or when a given task is urgent. […]

Avoid Extraneous Graphics and Animations

Despite being important to improve user experience, graphics and animations are battery intensive and should be used with moderation.

- **Context:** Mobile apps that feature impressive graphics and animations. […]
- **Solution:** Study the importance of graphics and animations to the user experience and reduce them when applicable. […]
- **Example:** Resort to low frame rates for animations when possible.

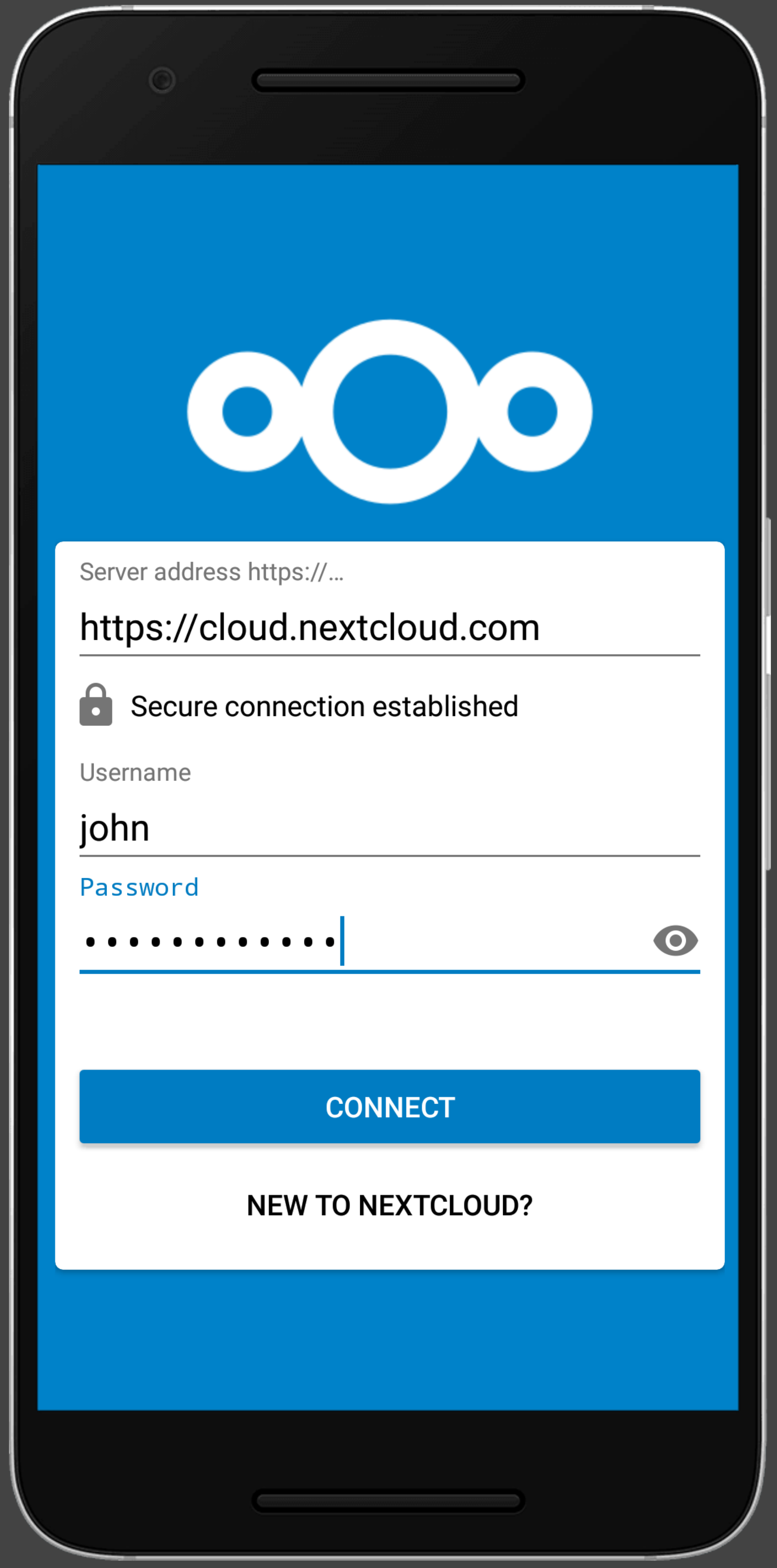

Android
iOS
Energy Patterns are more Frequent in Android Apps
R
0.03
0.02
0.01
0.00
Dark UI Colors
Dynamic Retry Delay
Avoid Extraneous Work
Race-to-idle
Open Only When Necessary
Push Over Poll
Power Save Mode
Power Awareness
Reduce Size
WiFi Over Cellular
Suppress Logs
Batch Operations
Cache
Decrease Rate
User Knows Best
Inform Users
Enough resolution
Sensor Fusion
Kill Abnormal Tasks
No screen interaction
Avoid Extra. Graph. & Anim.
Manual Sync, On Demand
Energy Pattern

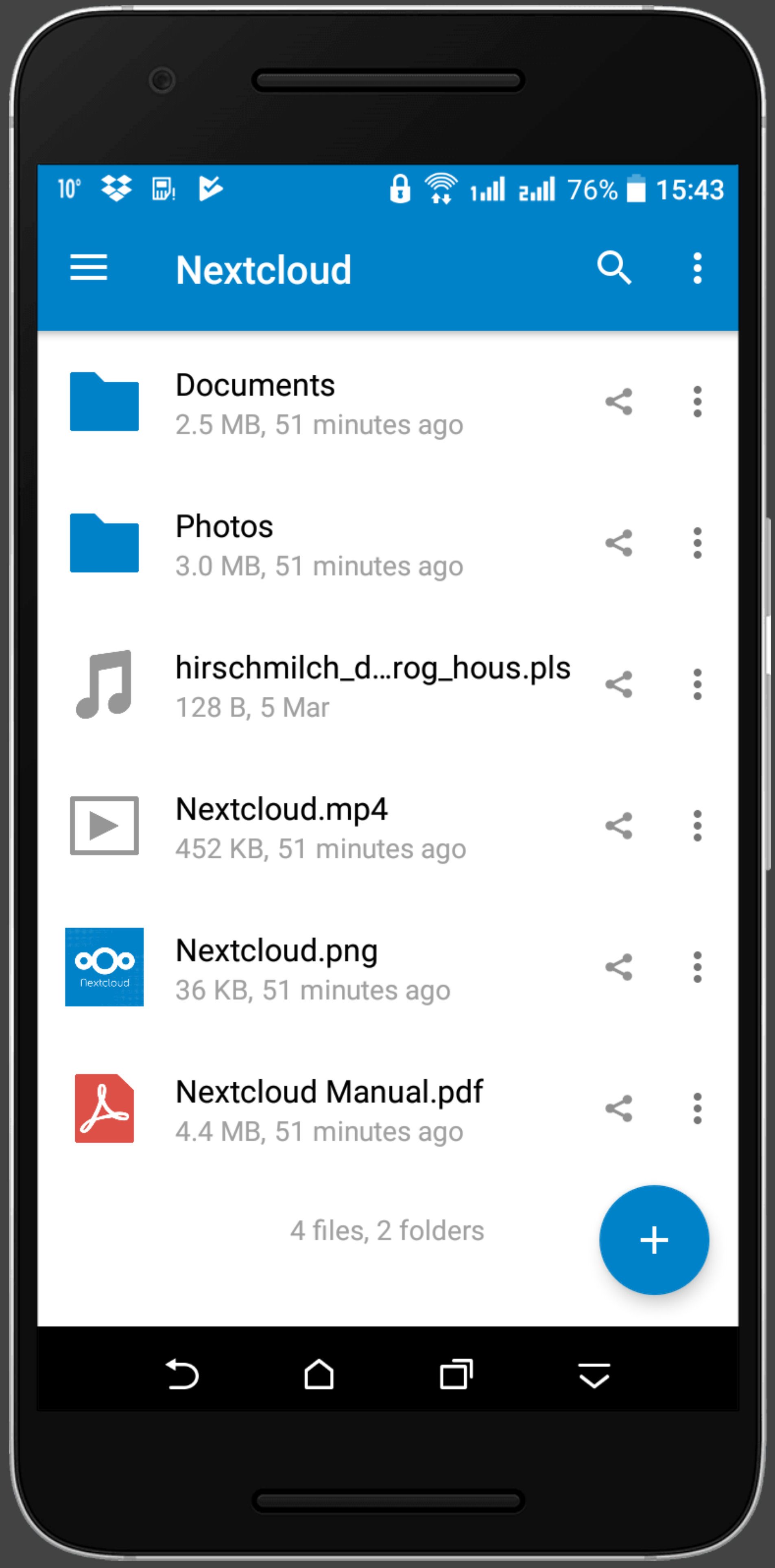
Example case: Nextcloud

FOSS → GET IT ON F-Droid

GET IT ON Google Play

Available on the App Store



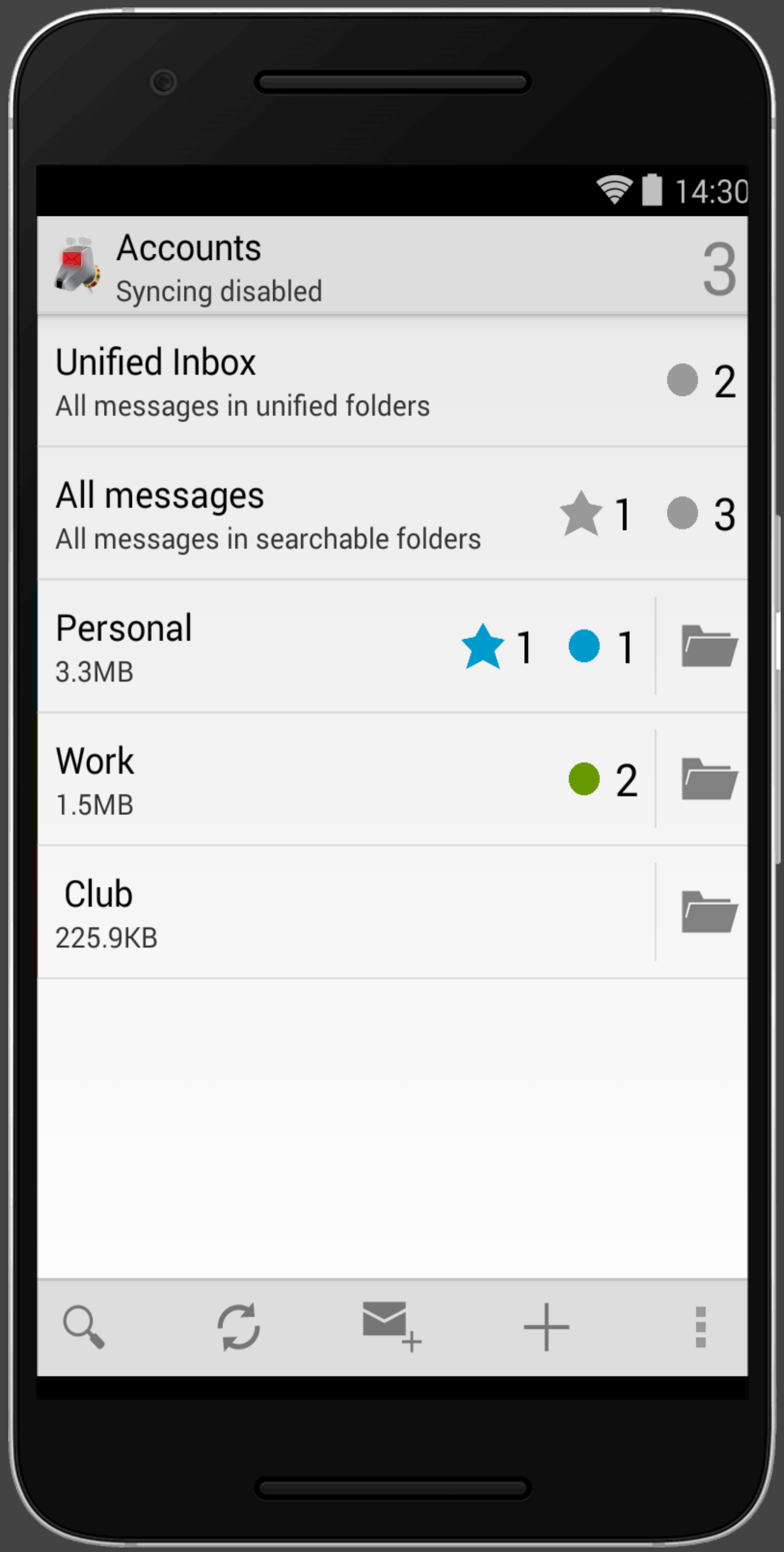


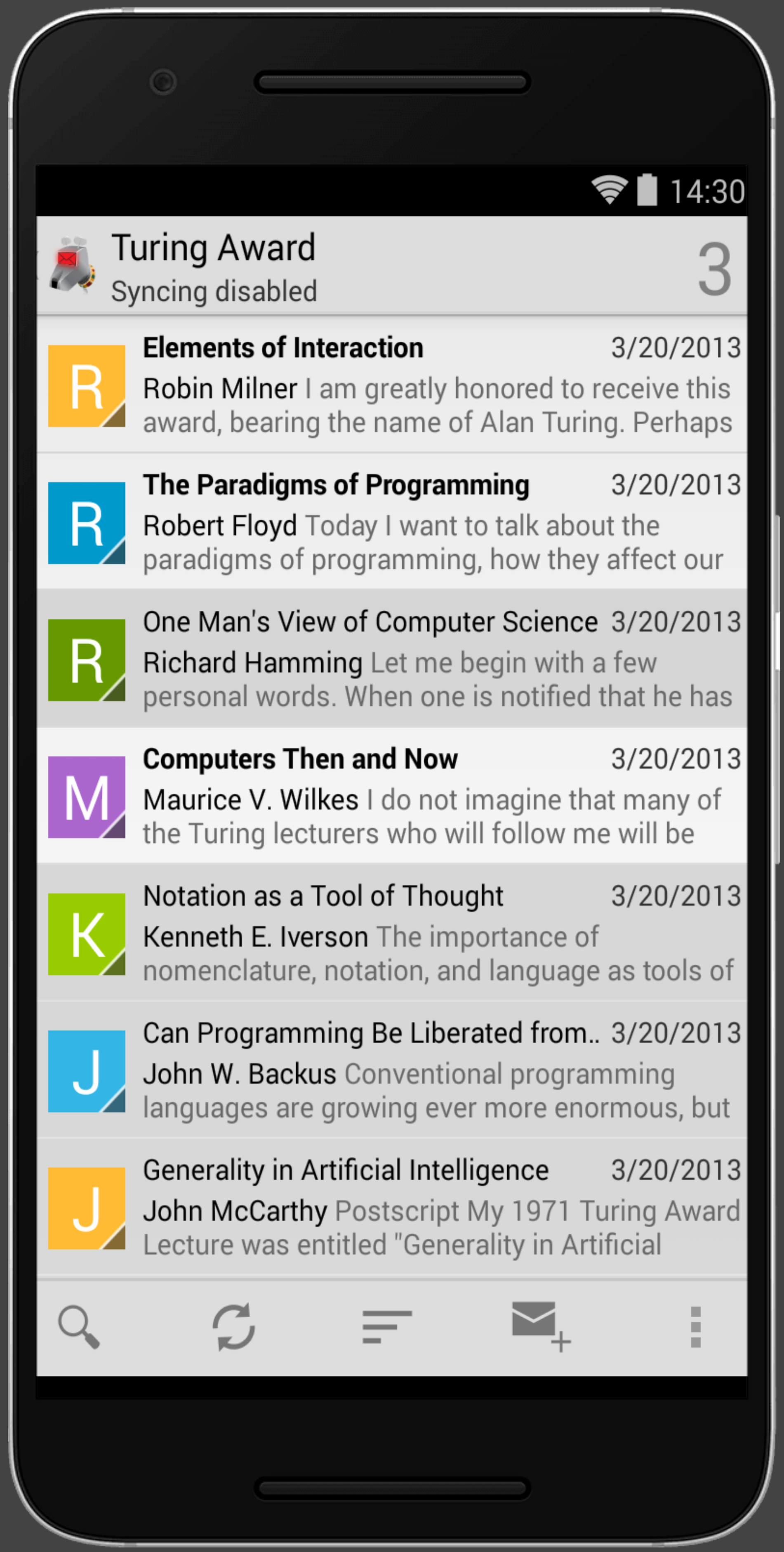
Example case: Nextcloud

- Users complain that sometimes they go on a trip and Nextcloud drains their battery. Users consider uninstalling the app when battery life is essential.
- File sync can be energy-greedy. **Send large files to the server, long 3G/4G data connections.**
- It is mostly used for backup. **No real-time collaboration is needed.**
- Energy requirements vary depending on context and user. Some days you really need all the battery you can get.
- https://github.com/nextcloud/android/commit/8bc432027e0d33e8043cf40192203203a40ca29c

Solutions?

Example case: K-9 mail





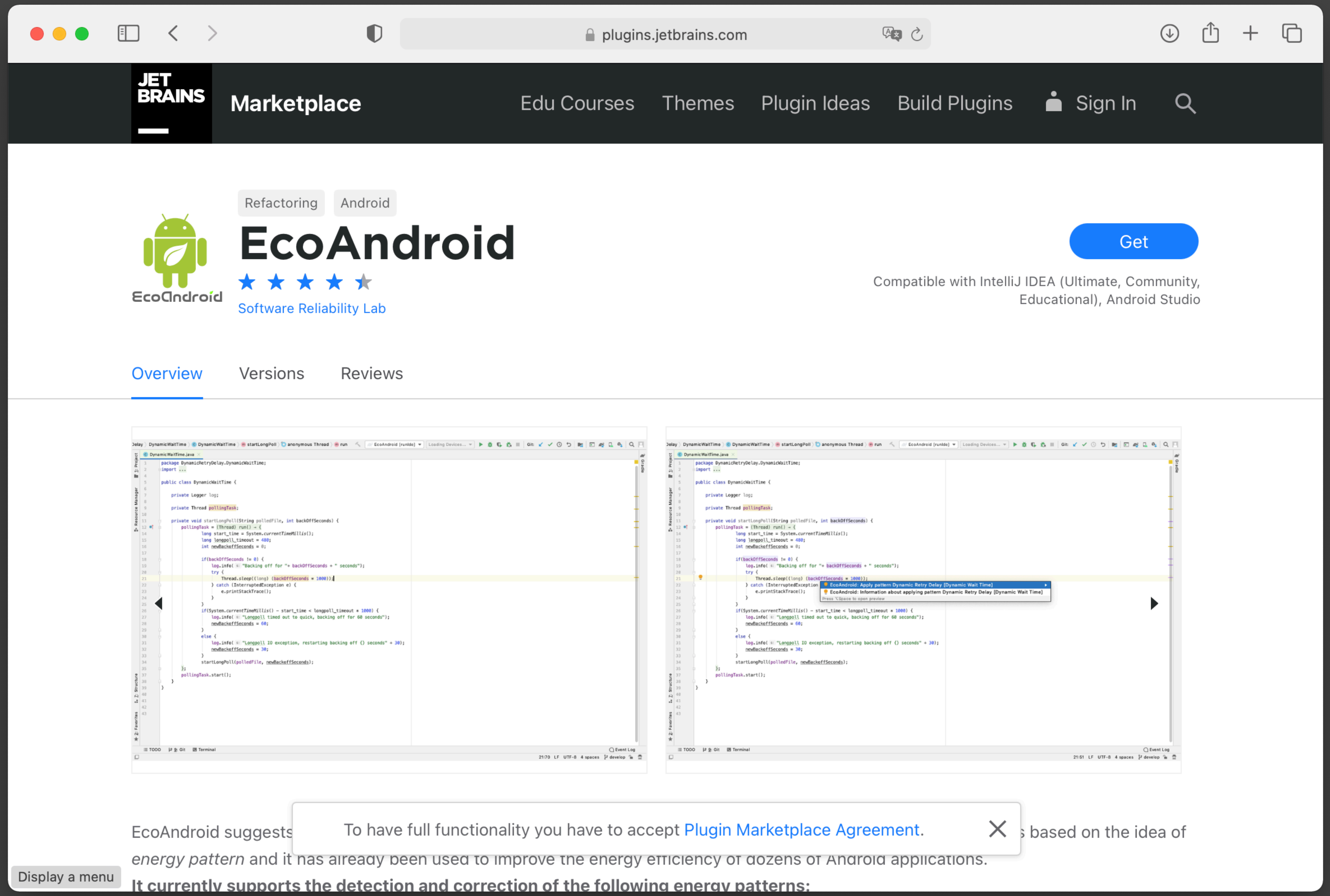
Example case: K-9 mail

- Some users noticed that K-9 mail was spending more energy than usual. 🙉
- A user that was having issues with a personal mail server noticed that K-9 mail was the one of the most energy-greedy apps. **IMAP IDLE protocol for real-time notifications.**
- When a connection is not possible, the app automatically retries later.
- https://github.com/k9mail/k-9/commit/86f3b28f79509d1a4d613eb39f60603e08579ea3

Solutions?

EcoAndroid

- Plugin for IntelliJ (Android Studio)
    - Dynamic Retry Delay
    - Push Over Poll
    - Reduce Size
    - Cache
    - Avoid Graphics and Animations



Carbon-Aware Computing for Datacenters

Ana Radovanovic´, Ross Koningstein, Ian Schneider, Bokan Chen, Alexandre Duarte, Binz Roy, Diyue Xiao, Maya Haridasan, Patrick Hung, Nick Care, Saurav Talukdar, Eric Mullen, Kendal Smith, MariEllen Cottman, and Walfredo Cirne

1

Carbon-Aware Computing for Datacenters

Ana Radovanović, Ross Koningstein, Ian Schneider, Bokan Chen, Alexandre Duarte, Binz Roy, Diyue Xiao, Maya Haridasan, Patrick Hung, Nick Care, Saurav Talukdar, Eric Mullen, Kendal Smith, MariEllen Cottman, and Walfredo Cirne

arXiv:2106.11750v1 [cs.DC] 11 Jun 2021

***Abstract*—The amount of CO_2 emitted per kilowatt-hour on an electricity grid varies by time of day and substantially varies by location due to the types of generation. Networked collections of warehouse scale computers, sometimes called Hyperscale Computing, emit more carbon than needed if operated without regard to these variations in carbon intensity. This paper introduces Google's system for Carbon-Intelligent Compute Management, which actively minimizes electricity-based carbon footprint and power infrastructure costs by delaying temporally flexible workloads. The core component of the system is a suite of analytical pipelines used to gather the next day's carbon intensity forecasts, train day-ahead demand prediction models, and use risk-aware optimization to generate the next day's carbon-aware Virtual Capacity Curves (VCCs) for all datacenter clusters across Google's fleet. VCCs impose hourly limits on resources available to temporally flexible workloads while preserving overall daily capacity, enabling all such workloads to complete within a day. Data from operation shows that VCCs effectively limit hourly capacity when the grid's energy supply mix is carbon intensive and delay the execution of temporally flexible workloads to "greener" times.**

***Index Terms*—Datacenter computing, carbon- and efficiency-aware compute management, power management.**

I. INTRODUCTION

Demand for computing resources and datacenter power worldwide has been continuously growing, now accounting for approximately 1% of total electricity usage [1]. Between 2010 and 2018, global datacenter workloads and compute instances increased more than sixfold [1]. In response, new methodologies for increasing datacenter power and energy efficiency are required to limit their growing environmental, economic and performance impacts [2], [3].

The datacenter industry has the potential to facilitate carbon emissions reductions in electricity grids. A considerable fraction of compute workloads have flexibility in both when and where they run. Given that emissions from electricity production vary substantially by time and location [4]–[7], we can exploit load flexibility to consume power where and when the grid is less carbon intensive. By effectively managing its load, the datacenter industry can contribute to a more robust, resilient, and cost-efficient energy system, facilitating grid decarbonization. Electric grid operators, in turn, can possibly benefit by as much as EUR 1B/year [8].

Furthermore, shifting execution of flexible workloads in time and space can decrease peak demand for resources and power. Since datacenters are planned based on peak power and resource usage, smaller peaks reduce the need for more capacity. Not only does this save money, it also reduces environmental impacts.

This paper describes the methodology and principles behind Google's system for Carbon-Intelligent Compute management, which reduces grid carbon emissions from Google's datacenter electricity use and reduces operating costs by increasing resource and power efficiency. To accomplish this goal, the system harnesses the temporal flexibility of a significant fraction of Google's internal workloads that tolerate delays as long as their work gets completed within 24 hours. Typical examples of such workloads are data compaction, machine learning, simulation, and data processing (e.g., video processing) pipelines – many of the tasks that make information found through Google products more accessible and useful. Note that other loads include user-facing services (Search, Maps and YouTube) that people rely on around the clock, and our cloud customers' workloads running in allocated Virtual Machines (VMs), which are not temporally flexible and therefore not affected by the new system.

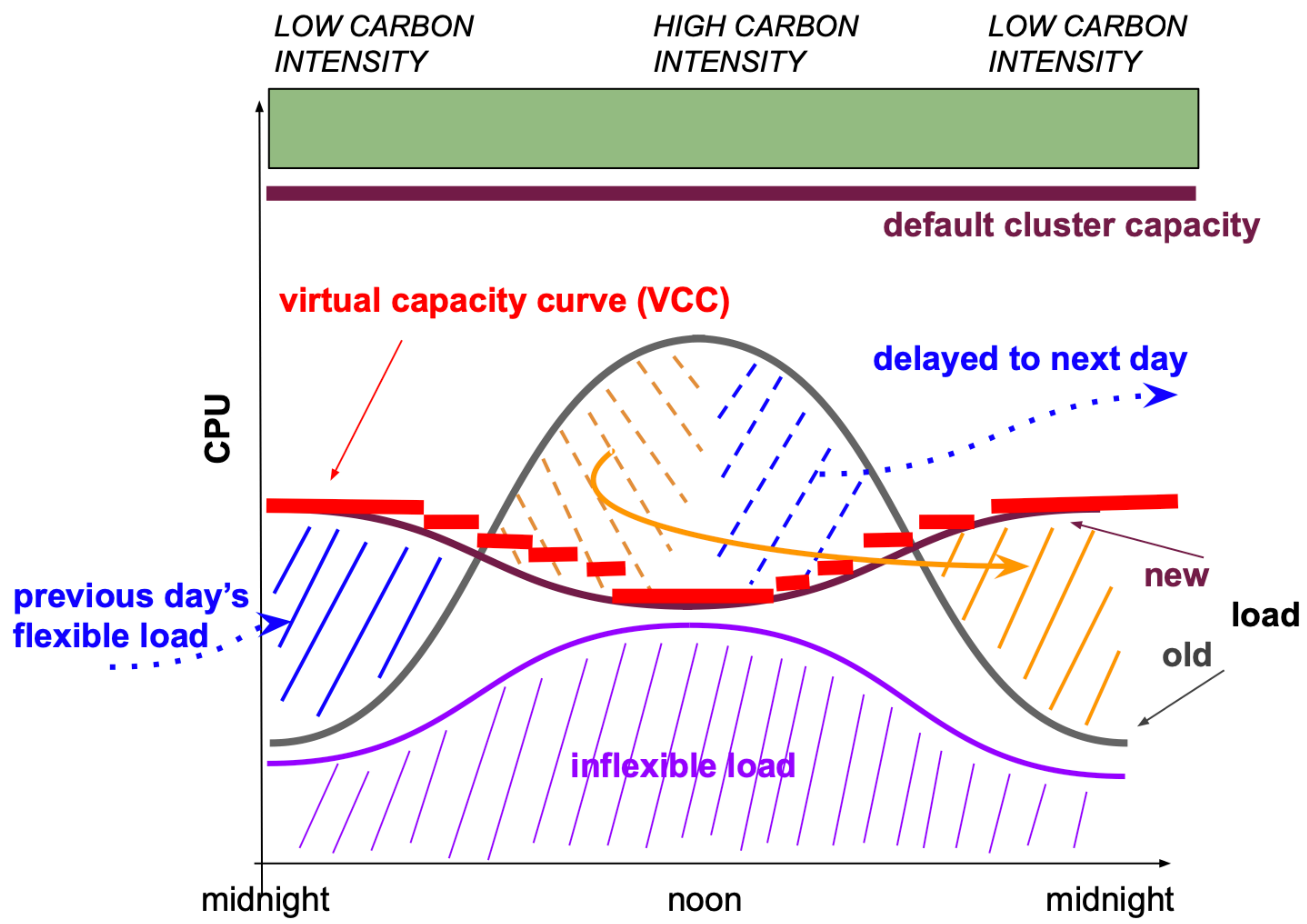
Workloads are comprised of compute jobs. The system needs to consider compute jobs' arrival patterns, resource usage, dependencies and placement consequences, which generally have high uncertainty and are hard to predict (i.e., we do not know in advance what jobs will run over the course of the next day). Fortunately, in spite of high uncertainties at the job level, Google's flexible resource usage and daily consumption at a cluster-level and beyond have demonstrated to be quite predictable within a day-ahead forecasting horizon. The aggregate outcome of job scheduling ultimately affects global costs, carbon footprint, and future resource utilization. The workload scheduler implementation must be simple (i.e., with as little as possible computational complexity in making placement decisions) to cope with the high volume of job requests.

The core of the carbon-aware load shaping mechanism is a set of cluster-level [9] Virtual Capacity Curves (VCCs), which are hourly resource usage limits that serve to shape each cluster resource and power usage profile over the following day. These limits are computed using an optimization process that takes account of aggregate flexible and inflexible demand predictions and their uncertainty, hourly carbon intensity forecasts [10], explicit characterization of business and environmental targets, infrastructure and workload performance expectations, and usage limits set by energy providers for different datacenters across Google's fleet.

The cluster-level VCCs are pushed to all of Google's datacenter clusters prior to the start of the next day, where they

The authors are with Google, Inc. Mountain View, CA, 94043 (Email: anaradovanovic@google.com, ross@google.com, ischneid@google.com, bokanchen@google.com, alexandredu@google.com, binzroy@google.com, diyuexiao@google.com, haridasan@google.com, phfhung@google.com, ncare@google.com, stalukdar@google.com, ericmullen@google.com, kendalsmith@google.com, meacottman@google.com, walfredo@google.com) .

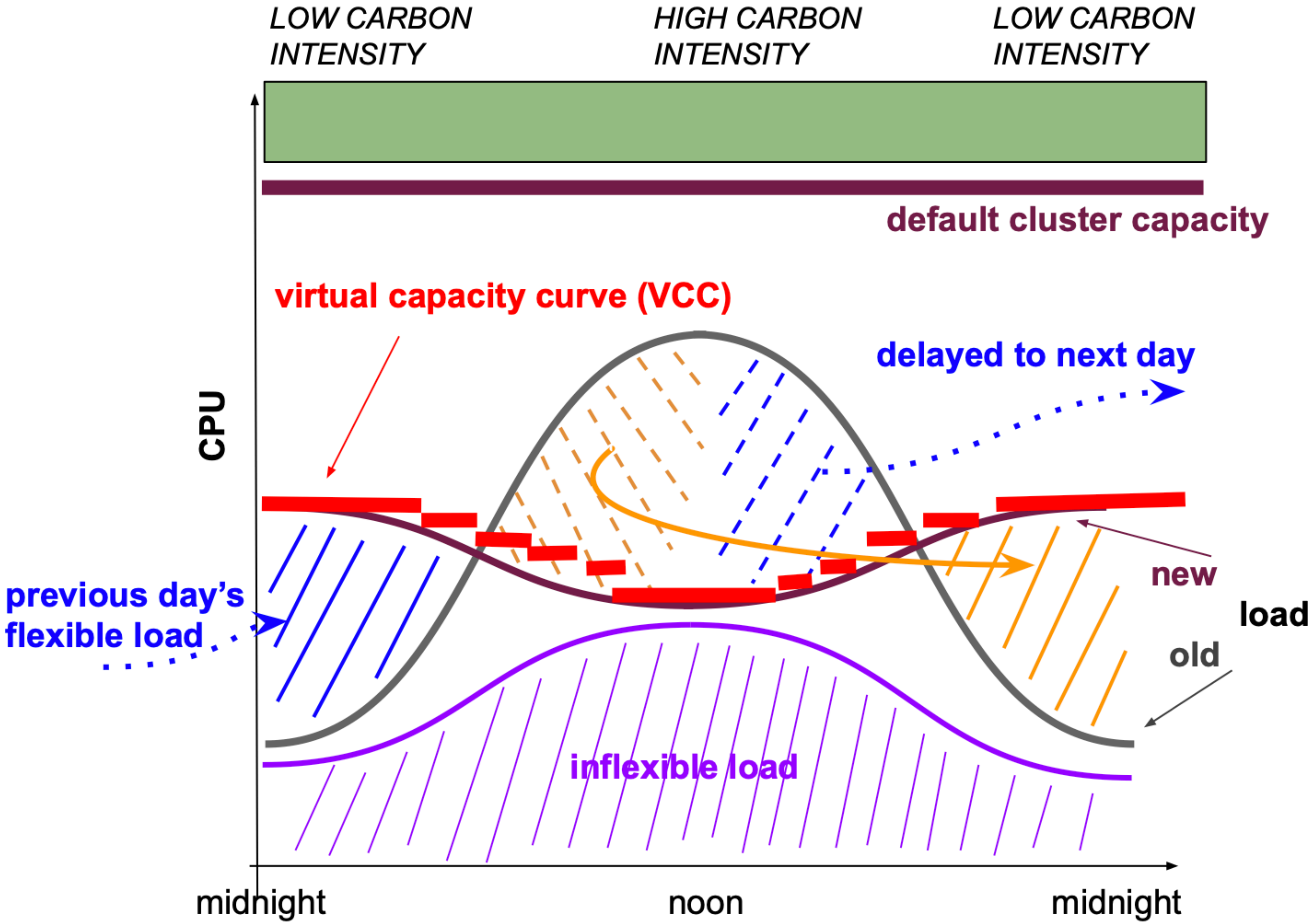
https://sites.google.com/view/energy-efficiency-languages


LOW CARBON INTENSITY
HIGH CARBON INTENSITY
LOW CARBON INTENSITY
default cluster capacity
virtual capacity curve (VCC)
delayed to next day
CPU
previous day's flexible load
new
load
old
inflexible load
midnight
noon
midnight

- Google’s Carbon-Intelligent Computing System (CICS)
- Main idea: use carbon-intensity data to shift datacenter jobs in time
- Typically, job schedulers use a metric of **cluster capacity** to schedule a job in a particular cluster.
    - CICS overrides this metric with the **virtual capacity curve (VCC)** that factors in Carbon intensity
    - When a new job comes in, the scheduler estimates its CPU load and power usage and assigns it to a cluster if the VCC is not exceeded.

- Jobs are divided between **flexible** and **inflexible**.
    - **Flexible** load is considered shapeable/shiftable as long as its total **daily compute (CPU) demand** is preserved

- The system needs to consider that, while running a job, the virtual capacity curve (VCC) might drop. Hence, this job should not start in the first place.
    - They forecast VCC for the next day



LOW CARBON INTENSITY
HIGH CARBON INTENSITY
LOW CARBON INTENSITY
default cluster capacity
virtual capacity curve (VCC)
delayed to next day
CPU
previous day's flexible load
new
load
old
inflexible load
midnight
noon
midnight

Virtual Cluster Capacity (VCC)

- Aims at reducing the peak load at carbon intensive hours but in total it should allow for the some amount of daily computation!
- Considers **Carbon intensity**
  - Using data from electricityMap.org
- It does not use carbon intensity directly.
  Carbon intensity is converted to a cost (kgCO2 -> $$)
  - This way, they can factor in other metrics that can also be converted to money. E.g., the cost saved by preventing peak load
    - Peak workload entails extra cost at the infrastructure level (e.g., control faclities' temperature).
  - By using money cost instead of carbon cost, they have more data available.

Virtual Cluster Capacity (VCC)

- If the forecast of VCC fails it shift flexible workload more than expected.
- This happens because the amount of workload forecasted was below the workload needed, and VCC was “aggressively” low.
  - The systems fall back to the real cluster capacity for **1 week** until results start being realistic again.

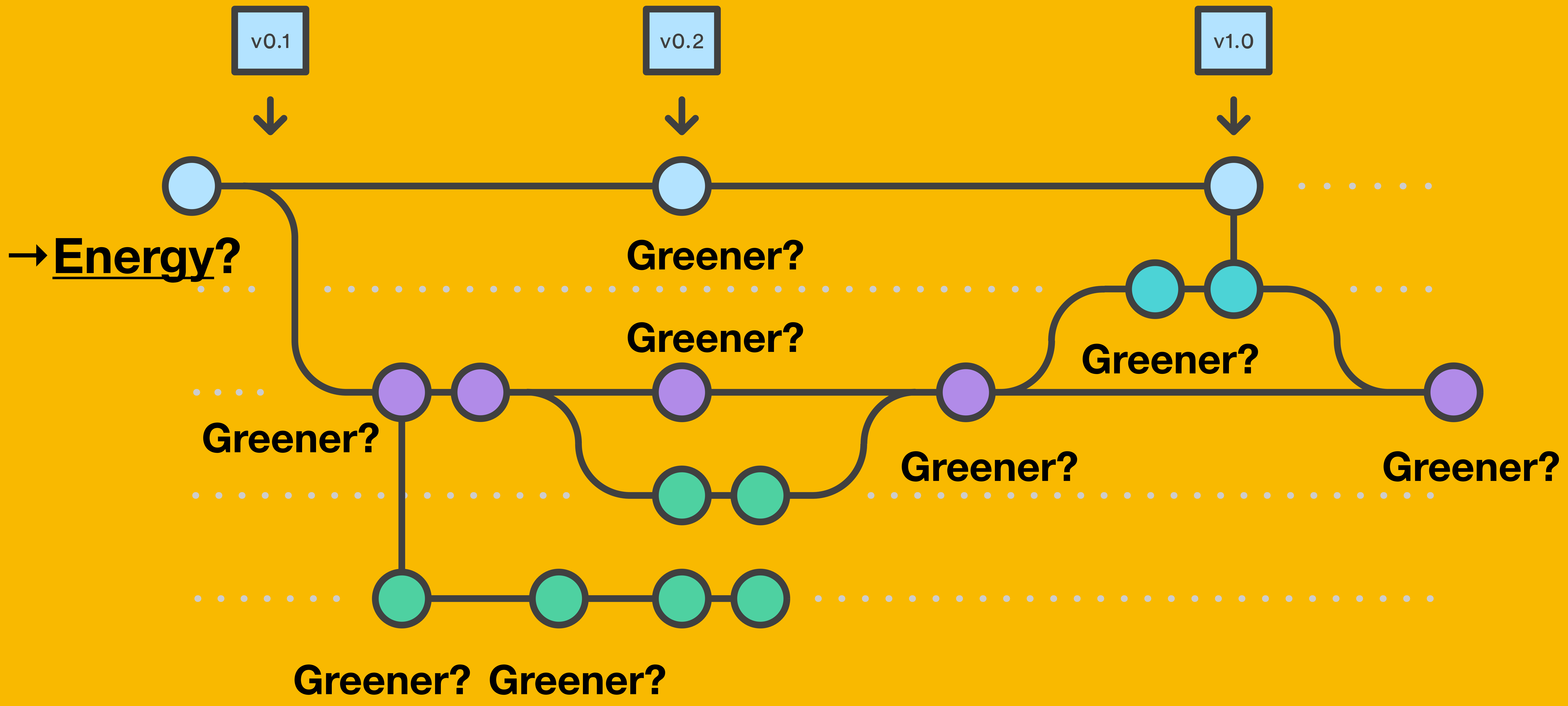
Next steps

- They will consider spatial flexibility.
  - I.e., tasks that can be shifted over time and over space.
    - It needs to factor in relocation overheads, though.

Critical questions

- Who defines what is **flexible** and **inflexible**?
- How do you estimate the **CPU load** of a given task?

Energy Regression Testing



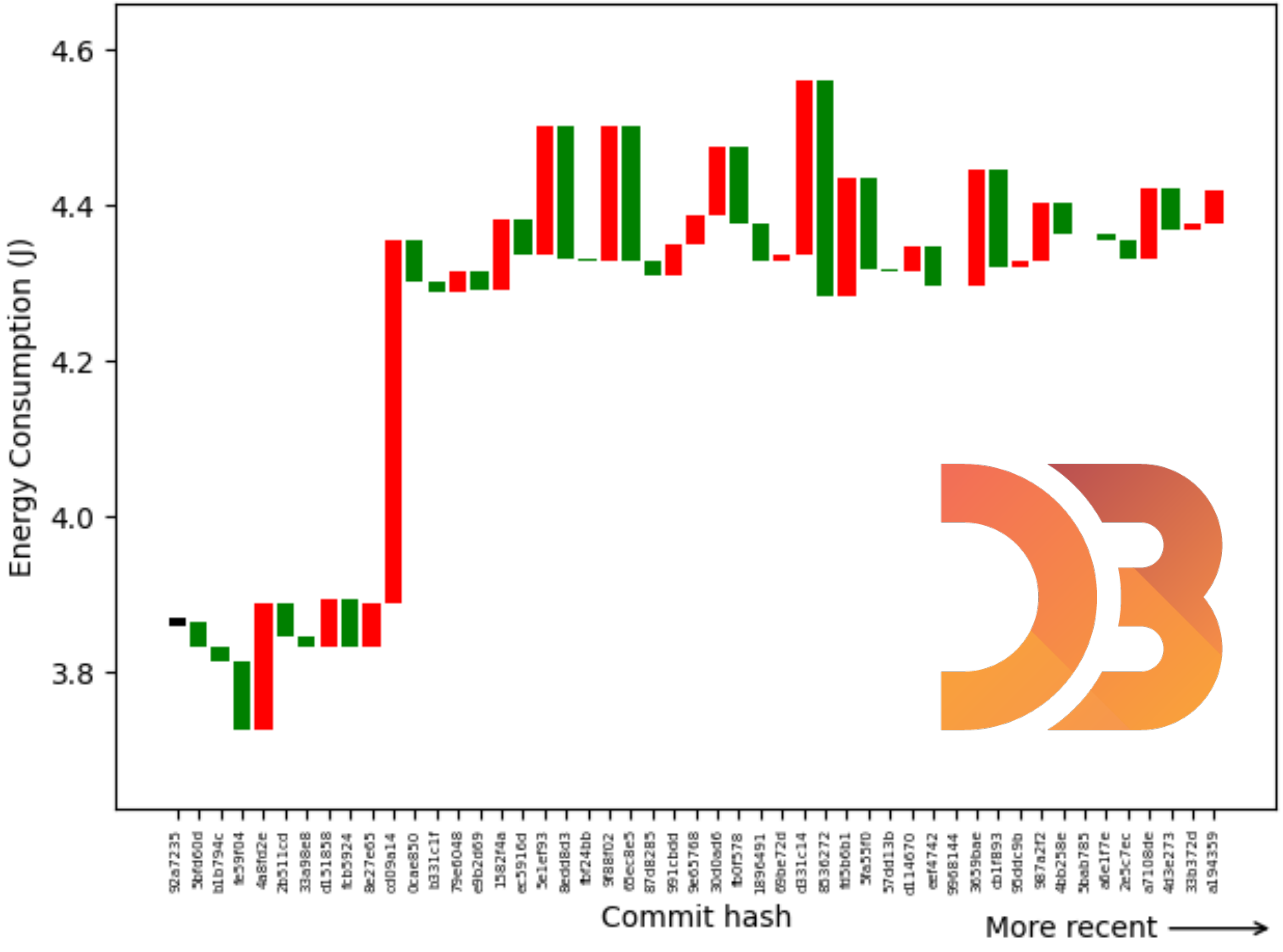
E-compare

```
steps:
  - uses: actions/checkout@v4
  - name: Set up Python
  - uses: actions/setup-python@v4
    with:
      python-version: '3.x'
  - run:  python -m pip install --upgrade pip
  - run: pip install -r requirements.txt
  - run: pip install pytest pytest-cov
+     - uses: koenhagen/measure-energy-action@v0.16
+       with:
+         GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          run: pytest tests.py --doctest-modules --cov=com
```

Github Actions

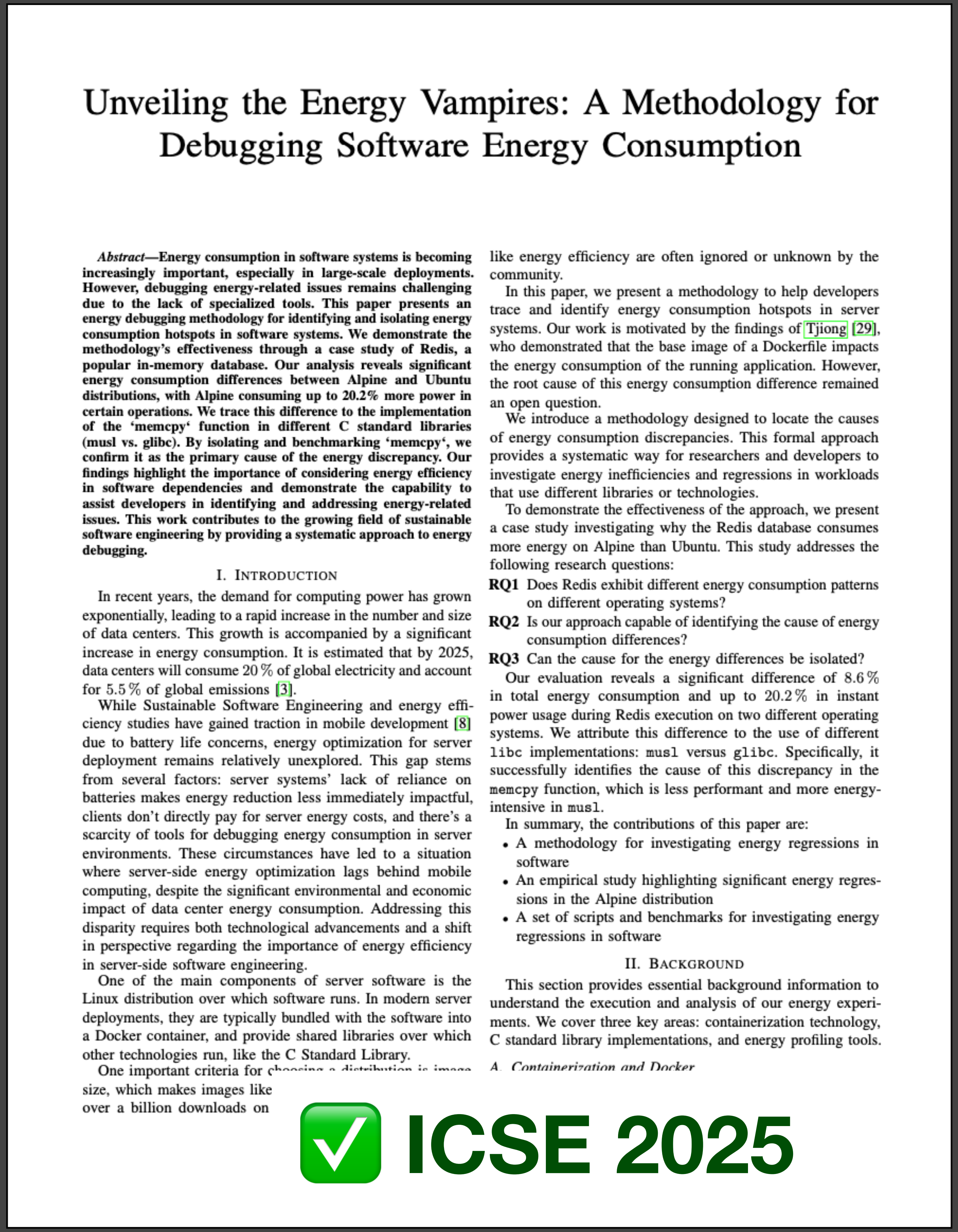
➜ https://koenhagen.github.io/E-Compare/

Energy Regression Testing



4. Debugging Energy with Docker Images

Unveiling the Energy Vampires
A methodology for debugging Software Energy Consumption



Unveiling the Energy Vampires: A Methodology for Debugging Software Energy Consumption

***Abstract*—Energy consumption in software systems is becoming increasingly important, especially in large-scale deployments. However, debugging energy-related issues remains challenging due to the lack of specialized tools. This paper presents an energy debugging methodology for identifying and isolating energy consumption hotspots in software systems. We demonstrate the methodology's effectiveness through a case study of Redis, a popular in-memory database. Our analysis reveals significant energy consumption differences between Alpine and Ubuntu distributions, with Alpine consuming up to 20.2% more power in certain operations. We trace this difference to the implementation of the 'memcpy' function in different C standard libraries (musl vs. glibc). By isolating and benchmarking 'memcpy', we confirm it as the primary cause of the energy discrepancy. Our findings highlight the importance of considering energy efficiency in software dependencies and demonstrate the capability to assist developers in identifying and addressing energy-related issues. This work contributes to the growing field of sustainable software engineering by providing a systematic approach to energy debugging.**

I. Introduction

In recent years, the demand for computing power has grown exponentially, leading to a rapid increase in the number and size of data centers. This growth is accompanied by a significant increase in energy consumption. It is estimated that by 2025, data centers will consume 20 % of global electricity and account for 5.5 % of global emissions [3].

While Sustainable Software Engineering and energy efficiency studies have gained traction in mobile development [8] due to battery life concerns, energy optimization for server deployment remains relatively unexplored. This gap stems from several factors: server systems' lack of reliance on batteries makes energy reduction less immediately impactful, clients don't directly pay for server energy costs, and there's a scarcity of tools for debugging energy consumption in server environments. These circumstances have led to a situation where server-side energy optimization lags behind mobile computing, despite the significant environmental and economic impact of data center energy consumption. Addressing this disparity requires both technological advancements and a shift in perspective regarding the importance of energy efficiency in server-side software engineering.

One of the main components of server software is the Linux distribution over which software runs. In modern server deployments, they are typically bundled with the software into a Docker container, and provide shared libraries over which other technologies run, like the C Standard Library.

One important criteria for [illegible] size, which makes images like over a billion downloads on

like energy efficiency are often ignored or unknown by the community.

In this paper, we present a methodology to help developers trace and identify energy consumption hotspots in server systems. Our work is motivated by the findings of Tjiong [29], who demonstrated that the base image of a Dockerfile impacts the energy consumption of the running application. However, the root cause of this energy consumption difference remained an open question.

We introduce a methodology designed to locate the causes of energy consumption discrepancies. This formal approach provides a systematic way for researchers and developers to investigate energy inefficiencies and regressions in workloads that use different libraries or technologies.

To demonstrate the effectiveness of the approach, we present a case study investigating why the Redis database consumes more energy on Alpine than Ubuntu. This study addresses the following research questions:

RQ1 Does Redis exhibit different energy consumption patterns on different operating systems?

RQ2 Is our approach capable of identifying the cause of energy consumption differences?

RQ3 Can the cause for the energy differences be isolated?

Our evaluation reveals a significant difference of 8.6 % in total energy consumption and up to 20.2 % in instant power usage during Redis execution on two different operating systems. We attribute this difference to the use of different `libc` implementations: `musl` versus `glibc`. Specifically, it successfully identifies the cause of this discrepancy in the `memcpy` function, which is less performant and more energy-intensive in `musl`.

In summary, the contributions of this paper are:

- A methodology for investigating energy regressions in software
- An empirical study highlighting significant energy regressions in the Alpine distribution
- A set of scripts and benchmarks for investigating energy regressions in software

II. Background

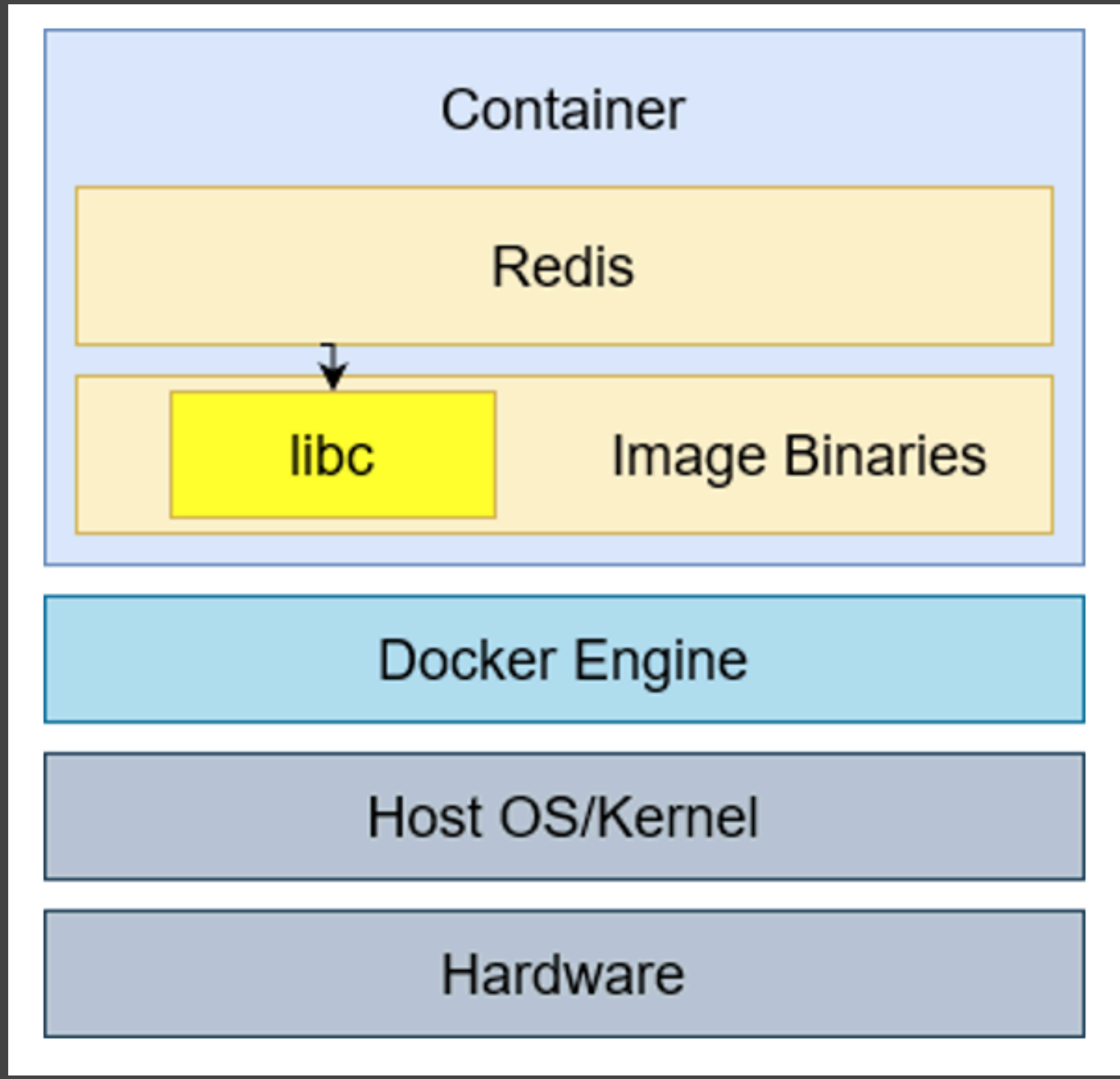
This section provides essential background information to understand the execution and analysis of our energy experiments. We cover three key areas: containerization technology, C standard library implementations, and energy profiling tools.

A. Containerization and Docker

✅ ICSE 2025

Data Centers and Docker

- Containers are the most popular way to deploy apps into the cloud
- Base image is an important choice when building an image
- Criteria
  - Linux distribution and binaries
  - Image size
  - **Energy?**

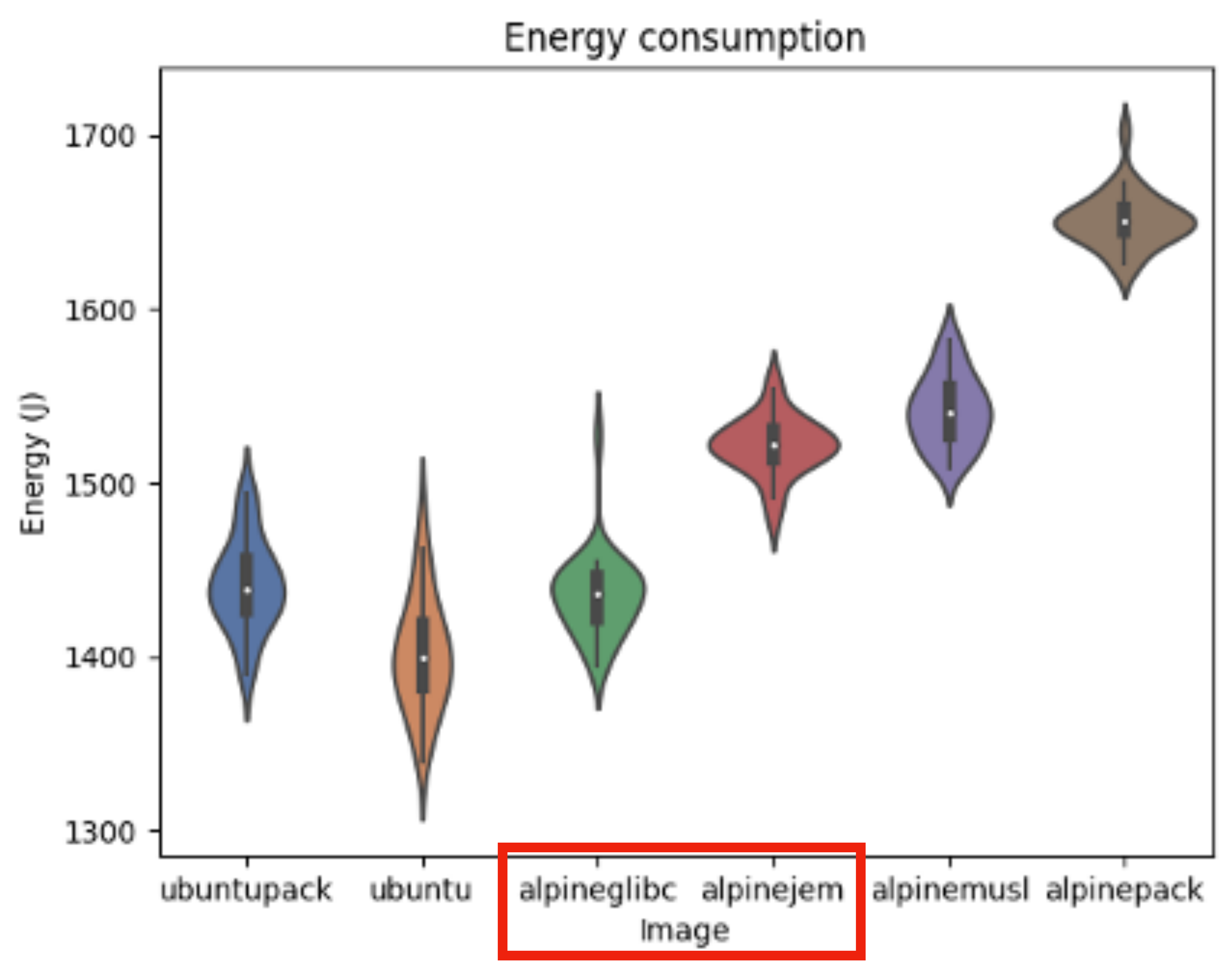


Redis energy consumption

- Ubuntu/Alpine base images have different energy consumption for Redis

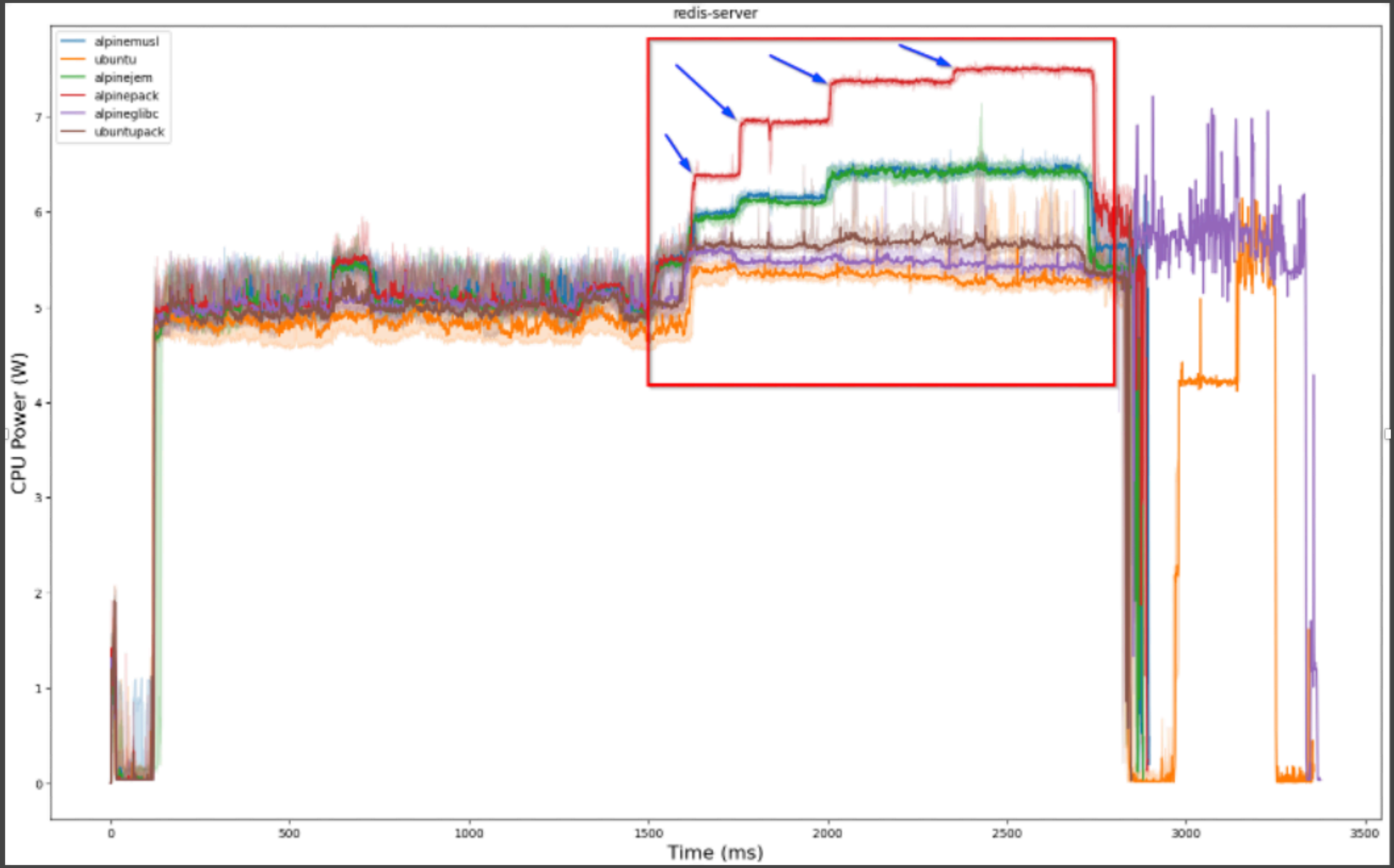
Label	Base image	libc	Redis version	Allocator
ubuntupack	Ubuntu	*glibc*	6.0.16	jemalloc
ubuntu	Ubuntu	*glibc*	7.2.4	jemalloc
alpineglibc	Alpine	*glibc*	7.2.4	jemalloc
alpinejem	Alpine	*musl*	7.2.4	jemalloc
alpinemusl	Alpine	*musl*	7.2.4	musl allocator
alpinepack	Alpine	*musl*	7.0.15	musl allocator

Image	Time (s)	Energy (J)
ubuntupack	281.62	1441.42
ubuntu	295.27	1401.54
alpineglibc	284.16	1435.28
alpinejem	284.15	1521.59
alpinemusl	284.94	1541.85
alpinepack	286.53	1651.11



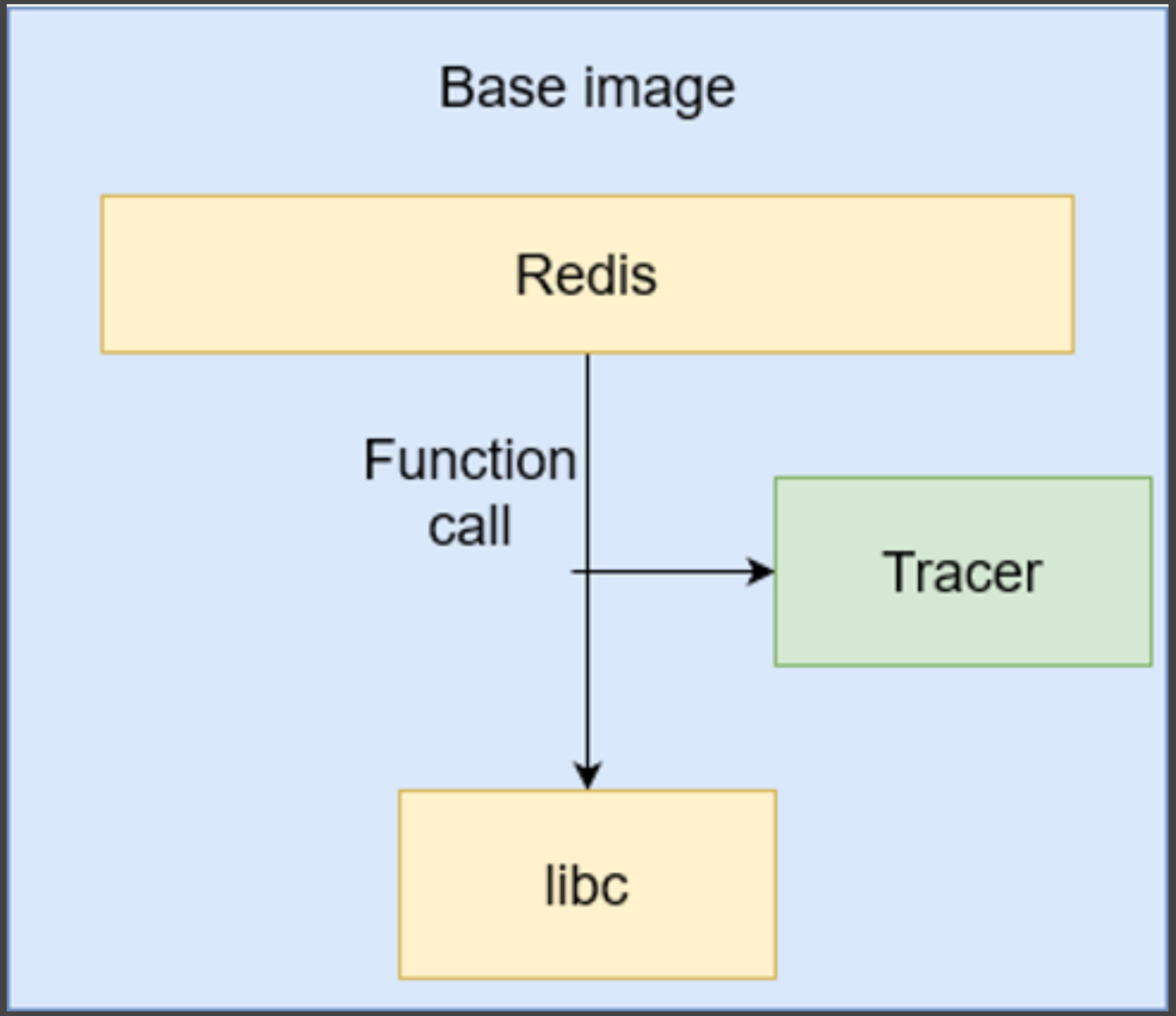
Redis Energy Consumption

- RQ1: What is the difference in energy consumption between glibc and musl?
- RQ2: Can we use tracing to compare energy consumption in shared libraries?
- RQ3: Can we verify the origin of energy consumption differences by recreating the workload behavior closely?



Tracing

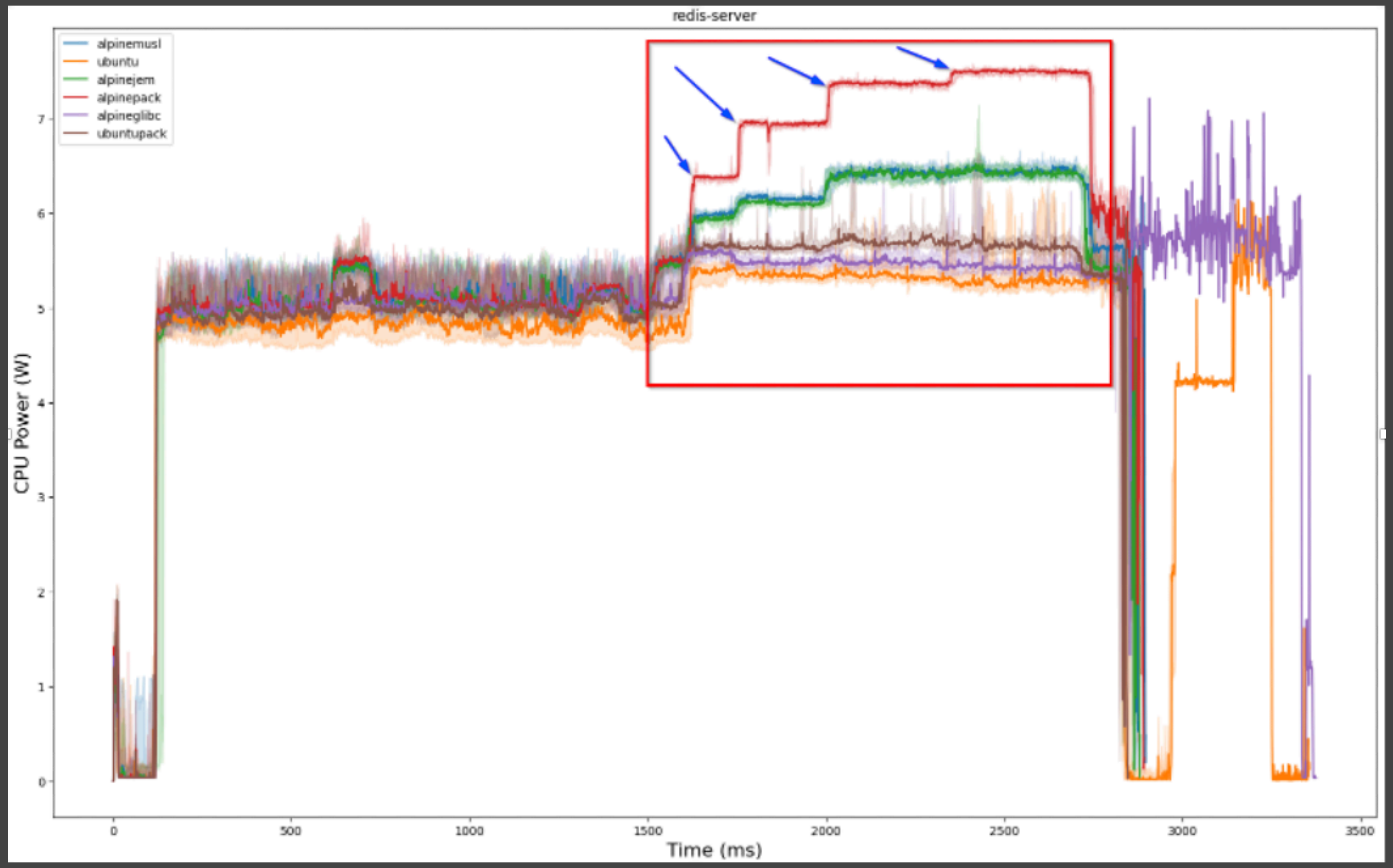
- Find *libc* function running at the end of the benchmark
- Tracing: Capture calls made to *libc*
  - Introduces non-linear overhead



Total time	Self time	Calls	Function
40.000 s	40.000 s	4	pthread_cond_timedwait
13.857 s	13.857 s	3	pthread_cond_wait
13.515 s	13.515 s	99494	epoll_wait
6.599 s	6.599 s	115270590	memcpy
6.150 s	6.150 s	500006	write
2.218 s	2.218 s	501557	read
239.204 ms	239.204 ms	2340922	strchr
221.763 ms	221.763 ms	1776711	strcasecmp
152.807 ms	152.807 ms	1201218	gettimeofday
151.164 ms	151.164 ms	1307754	memcmp
117.885 ms	117.885 ms	898262	clock_gettime
45.398 ms	45.398 ms	99495	localtime_r
26.472 ms	26.472 ms	1552	close
25.426 ms	25.426 ms	132144	memmove
16.300 ms	16.300 ms	5033	setsockopt

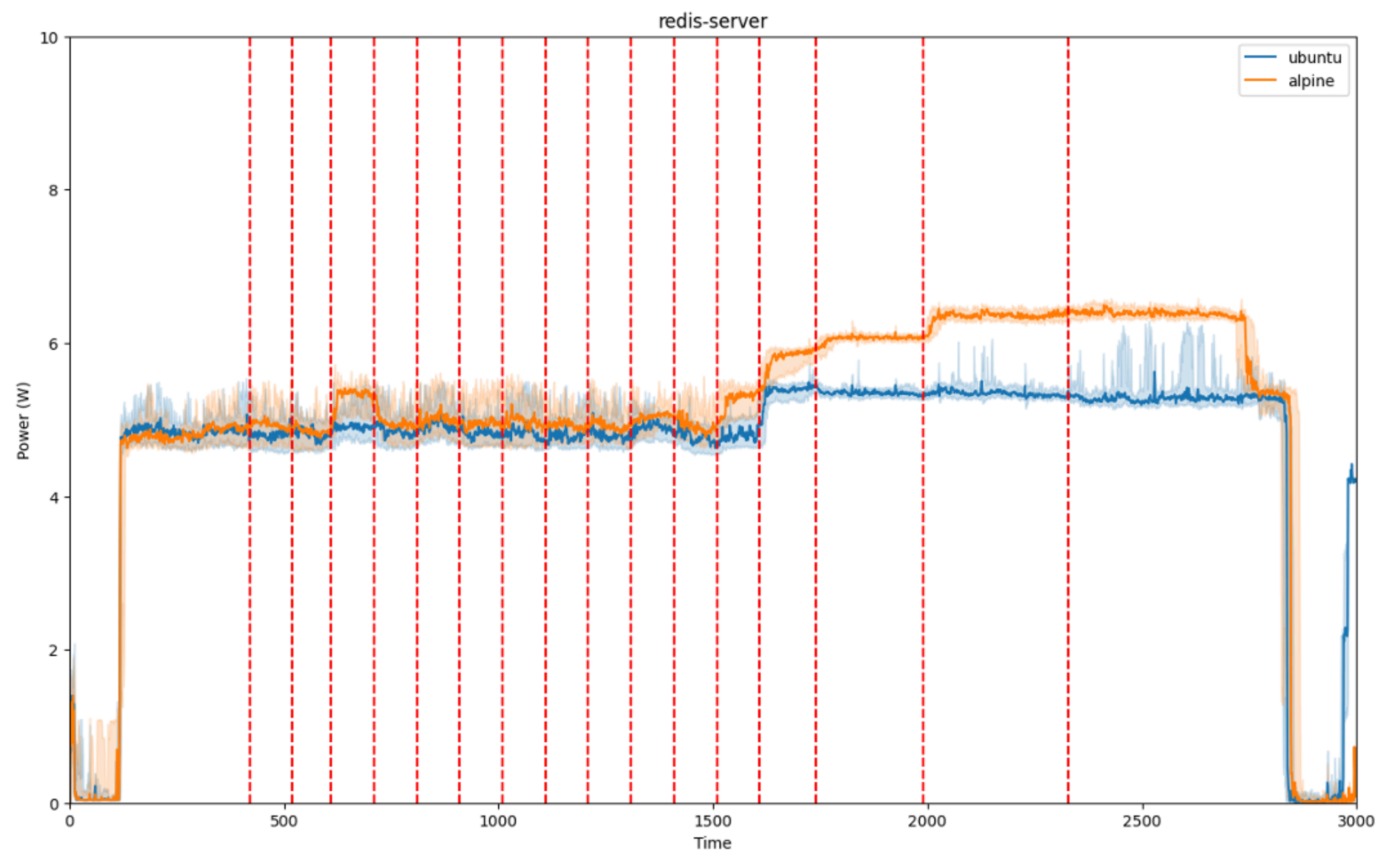
Synchronization

- Tracing slows down execution and increase energy consumption
- Run separately and synchronize with logs

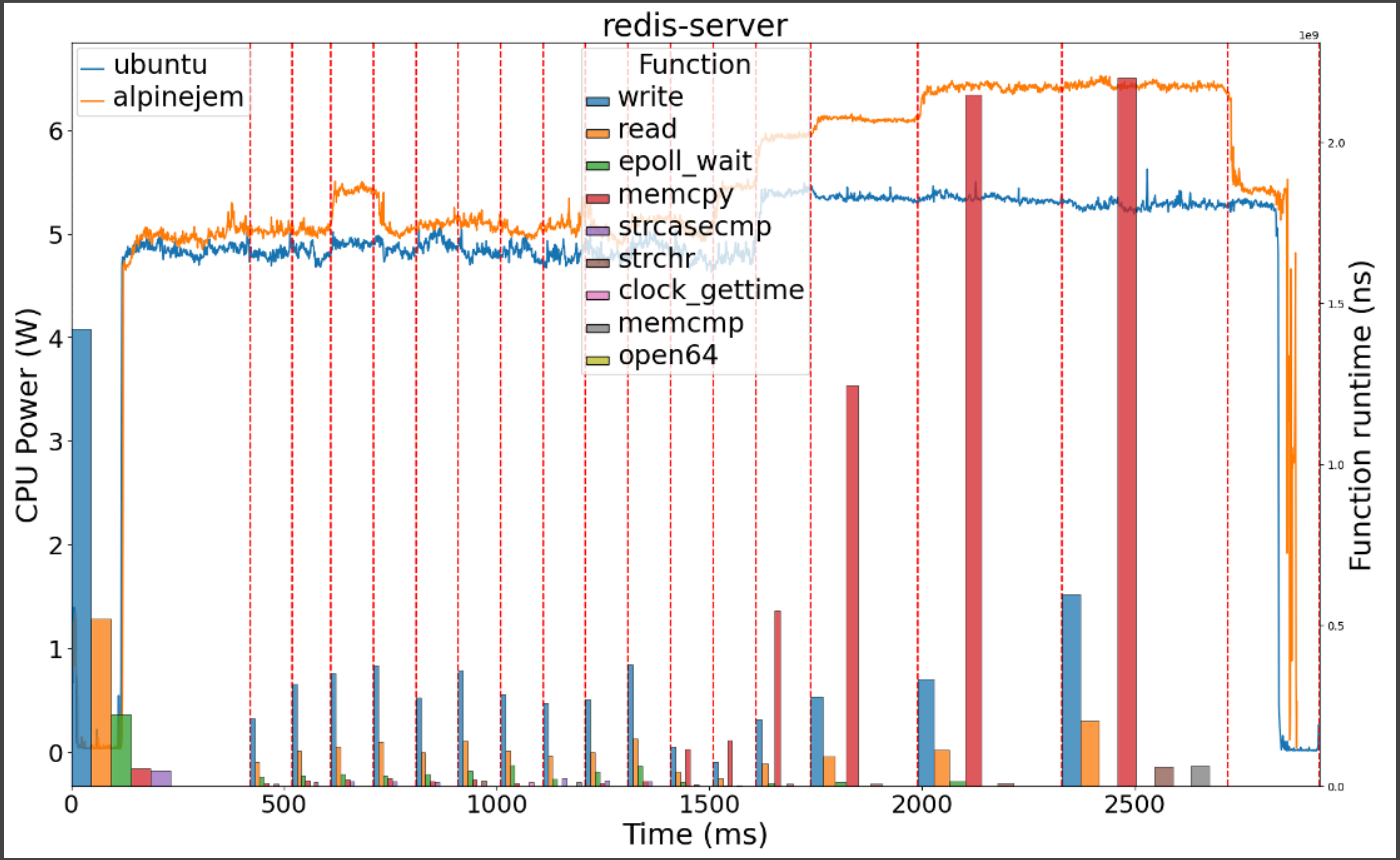


Total time	Self time	Calls	Function
40.000 s	40.000 s	4	pthread_cond_timedwait
13.857 s	13.857 s	3	pthread_cond_wait
13.515 s	13.515 s	99494	epoll_wait
6.599 s	6.599 s	115270590	memcpy
6.150 s	6.150 s	500006	write
2.218 s	2.218 s	501557	read
239.204 ms	239.204 ms	2340922	strchr
221.763 ms	221.763 ms	1776711	strcasecmp
152.807 ms	152.807 ms	1201218	gettimeofday
151.164 ms	151.164 ms	1307754	memcmp
117.885 ms	117.885 ms	898262	clock_gettime
45.398 ms	45.398 ms	99495	localtime_r
26.472 ms	26.472 ms	1552	close
25.426 ms	25.426 ms	132144	memmove
16.300 ms	16.300 ms	5033	setsockopt

Tracing Analysis

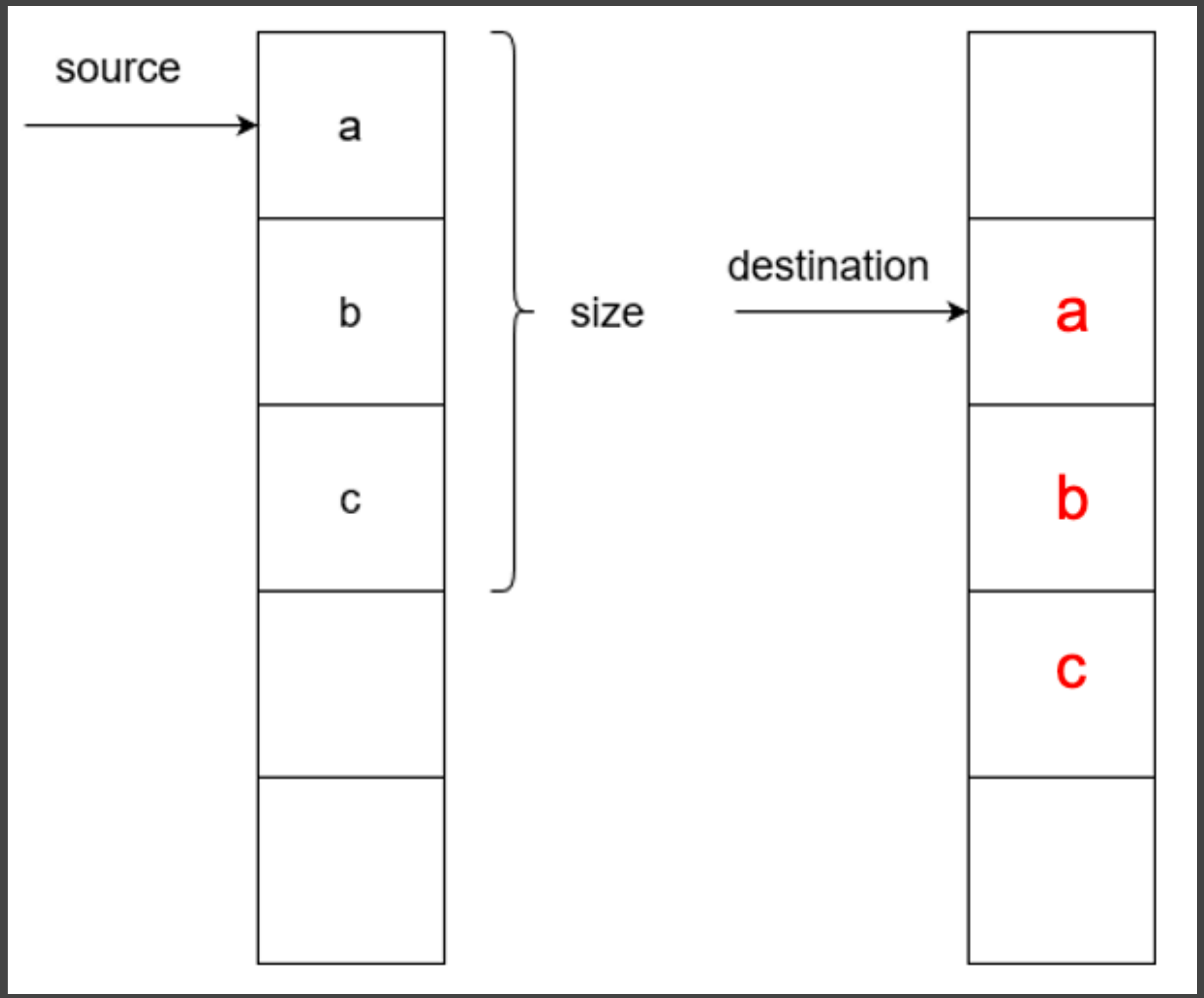


Tracing Analysis

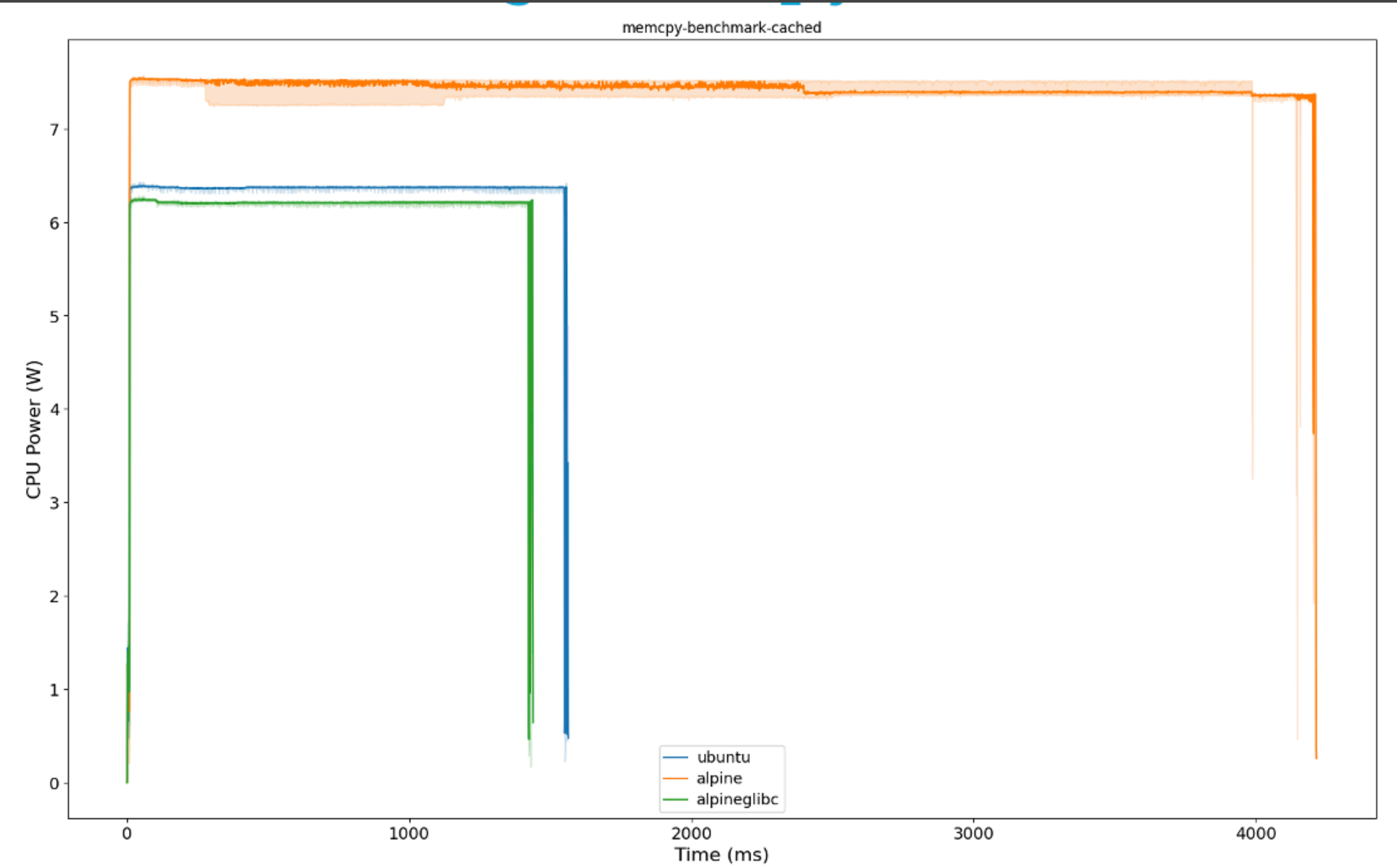


Benchmarking *memcpy*

- Copy bytes from one part of memory to another
- Depending on function parameters, different assembly-level optimizations
  - Some not included in *musl*



Benchmarking *memcpy*



- Copy 4 bytes from fixed pointer to moving pointer
- (100,300,500,600) x 1M requests
- 15% more power and 3X time

Image	Time (s)	Energy (J)
alpine	406.10	2977.45
ubuntu	155.37	972.76
alpineglibc	142.53	869.53

Conclusions

- Significant difference in energy consumption for *memcpy*
    - 8.6% difference for Redis workload and 13% in our benchmark

- musl trades performance for a smaller codebase
    - No official documentation
    - No awareness about energy performance

5. Energy Efficiency vs Code Quality

Measuring Maintainability

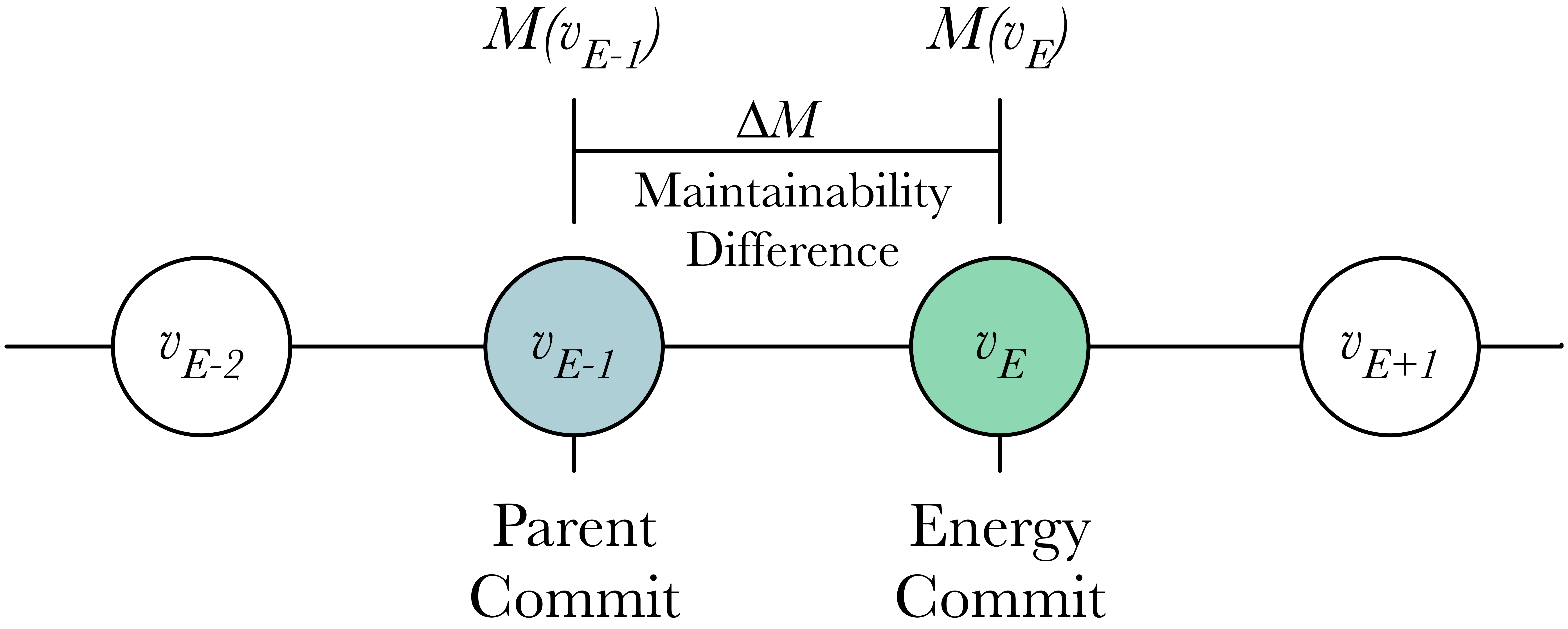
- According to ISO/IEC 25010, Maintainability is "*the degree of effectiveness and efficiency with which a software product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements*"

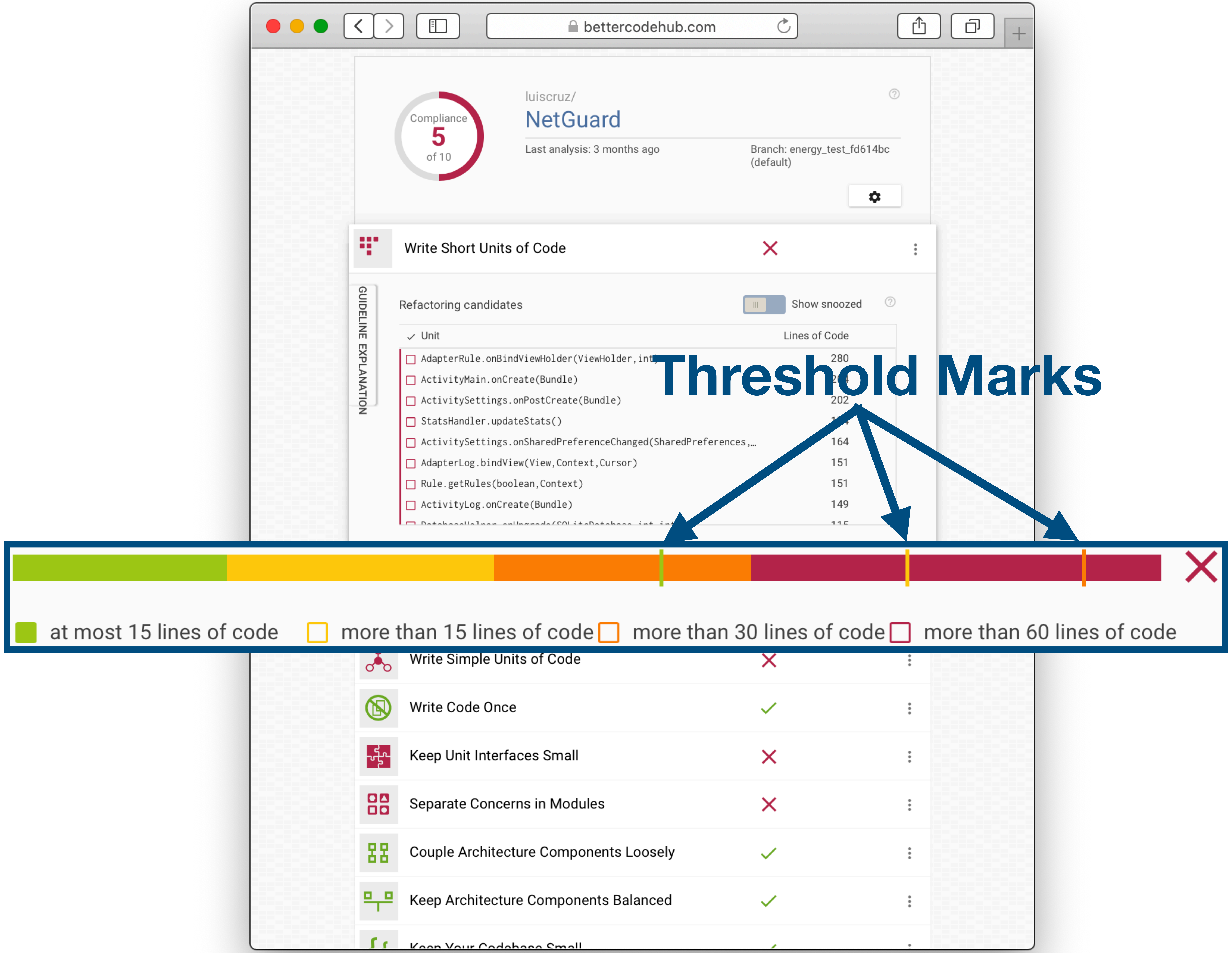
- We use the code analysis tool Better Code Hub to assess maintainability

- Better Code Hub maps the ISO/IEC 25010 standard on maintainability into a set of guidelines derived from static analysis

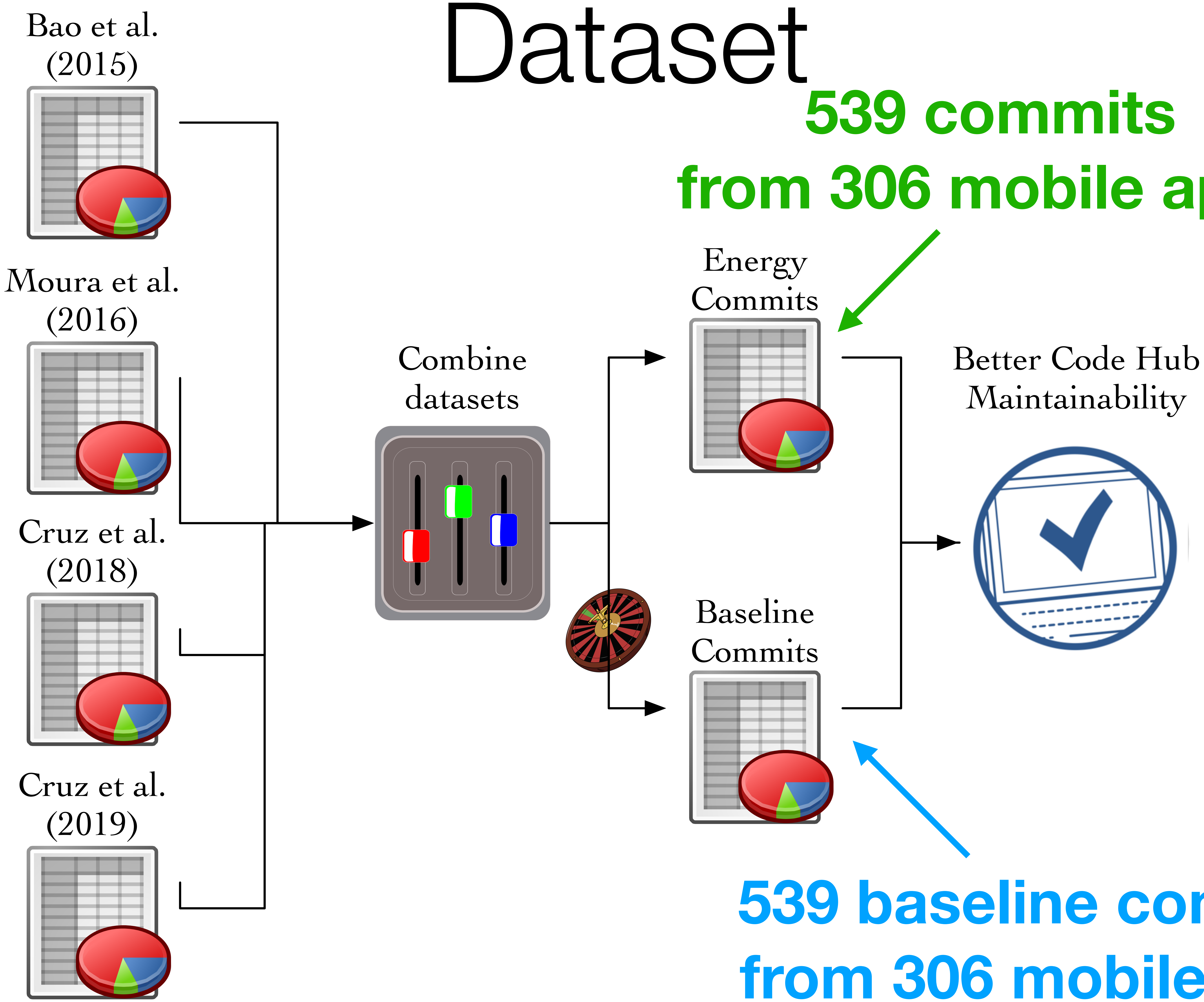
Maintainability of Energy Changes

- What is the impact of making energy-oriented code changes on the maintainability of mobile apps?




bettercodehub.com
Compliance
5
of 10
luiscruz/
NetGuard
Last analysis: 3 months ago
Branch: energy_test_fd614bc (default)
Write Short Units of Code
GUIDELINE EXPLANATION
Refactoring candidates
Show snoozed
Unit
Lines of Code
AdapterRule.onBindViewHolder(ViewHolder,int
280
ActivityMain.onCreate(Bundle)
ActivitySettings.onPostCreate(Bundle)
202
StatsHandler.updateStats()
ActivitySettings.onSharedPreferenceChanged(SharedPreferences,...
164
AdapterLog.bindView(View,Context,Cursor)
151
Rule.getRules(boolean,Context)
151
ActivityLog.onCreate(Bundle)
149
Threshold Marks
at most 15 lines of code
more than 15 lines of code
more than 30 lines of code
more than 60 lines of code
Write Simple Units of Code
Write Code Once
Keep Unit Interfaces Small
Separate Concerns in Modules
Couple Architecture Components Loosely
Keep Architecture Components Balanced

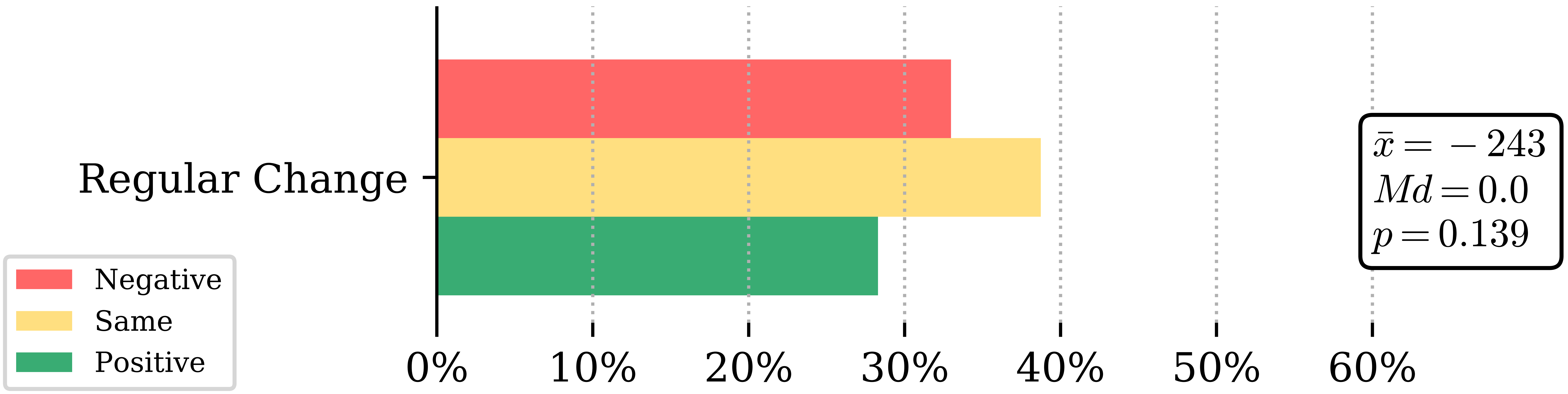
Energy Code Changes Dataset



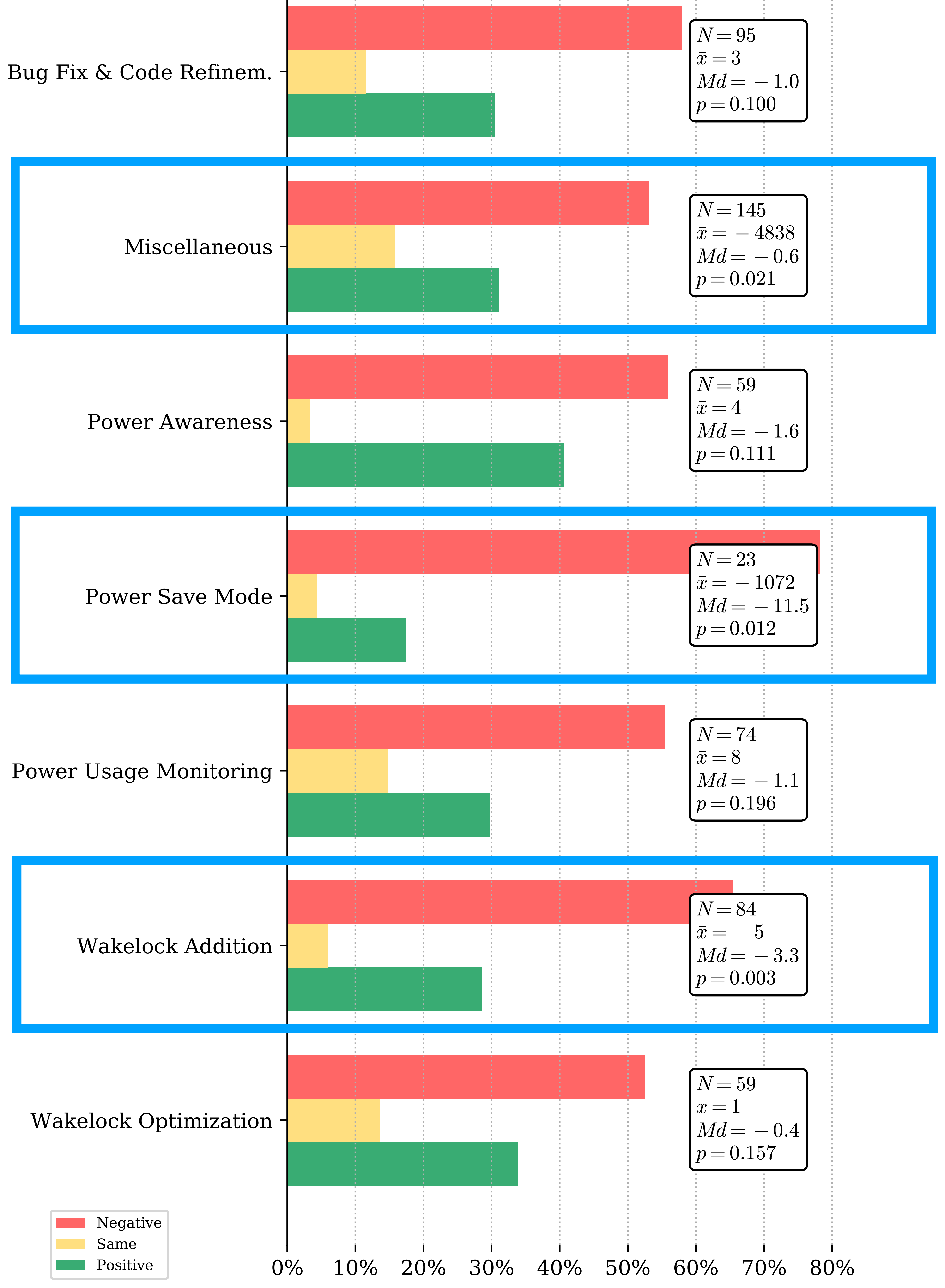
539 commits from 306 mobile apps

539 baseline commits from 306 mobile apps

Impact of energy changes on maintainability



Which energy patterns are more likely to affect maintainability?



Typical maintainability issue I

https://github.com/einmalfel/PodListen/commit/2ed5a65

4 changed files with **28 additions** and **0 deletions**.

```
...
     private synchronized void readPreference(Key key) {
       switch (key) {
+        case AUTO_DOWNLOAD_AC:
+          autoDownloadACOnly = sPrefs.getBoolean(Key.AUTO_DOWNLOAD_AC.toString(), false);
+          if (!autoDownloadACOnly) {
+            context.sendBroadcast(new Intent(DownloadReceiver.UPDATE_QUEUE_ACTION));
+          } else if (!DownloadReceiver.isDeviceCharging()) {
+            DownloadReceiver.stopDownloads(null);
+          }
+          break;
         case PLAYER_FOREGROUND:
           playerForeground = sPrefs.getBoolean(Key.PLAYER_FOREGROUND.toString(), false);
           break;
...
```

Typical maintainability issue II

https://github.com/mozilla/MozStumbler/commit/6ea0268

5 changed files with **66 additions** and **14 deletions**.

```diff
+
+            // each time we reconnect, check to see if we're suppose to be
+            // in power saving mode.  if not, start the scanning. TODO: we
+            // shouldn't just stopScanning if we find that we are in PSM.
+            // Instead, we should see if we were scanning do to an activity
+            // recognition.  If we were, don't stop.
+            if (mConnectionRemote != null) {
+                try {
+                    if (mPrefs.getPowerSavingMode()) {
+                        mConnectionRemote.stopScanning();
+                    } else {
+                        mConnectionRemote.startScanning();
+                    }
+                } catch (RemoteException e) {
+                    Log.e(LOGTAG, "", e);
+                }
+            }
             updateUI();
         }
```