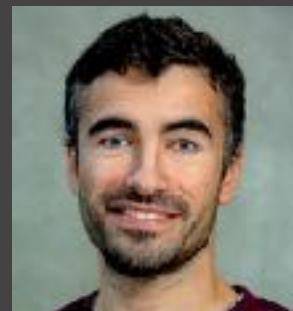


8. Research in Green Software

Sustainable Software Engineering

CS4295



Luís Cruz

L.Cruz@tudelft.nl

1. Energy patterns for mobile apps
2. Energy efficiency across programming languages
3. Carbon-aware datacenters

- 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

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

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

Catalog of energy patterns for mobile applications



Luis Cruz¹  · Rui Abreu²

Published online: 5 March 2019
© Springer Science+Business Media, LLC, part of Springer Nature 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

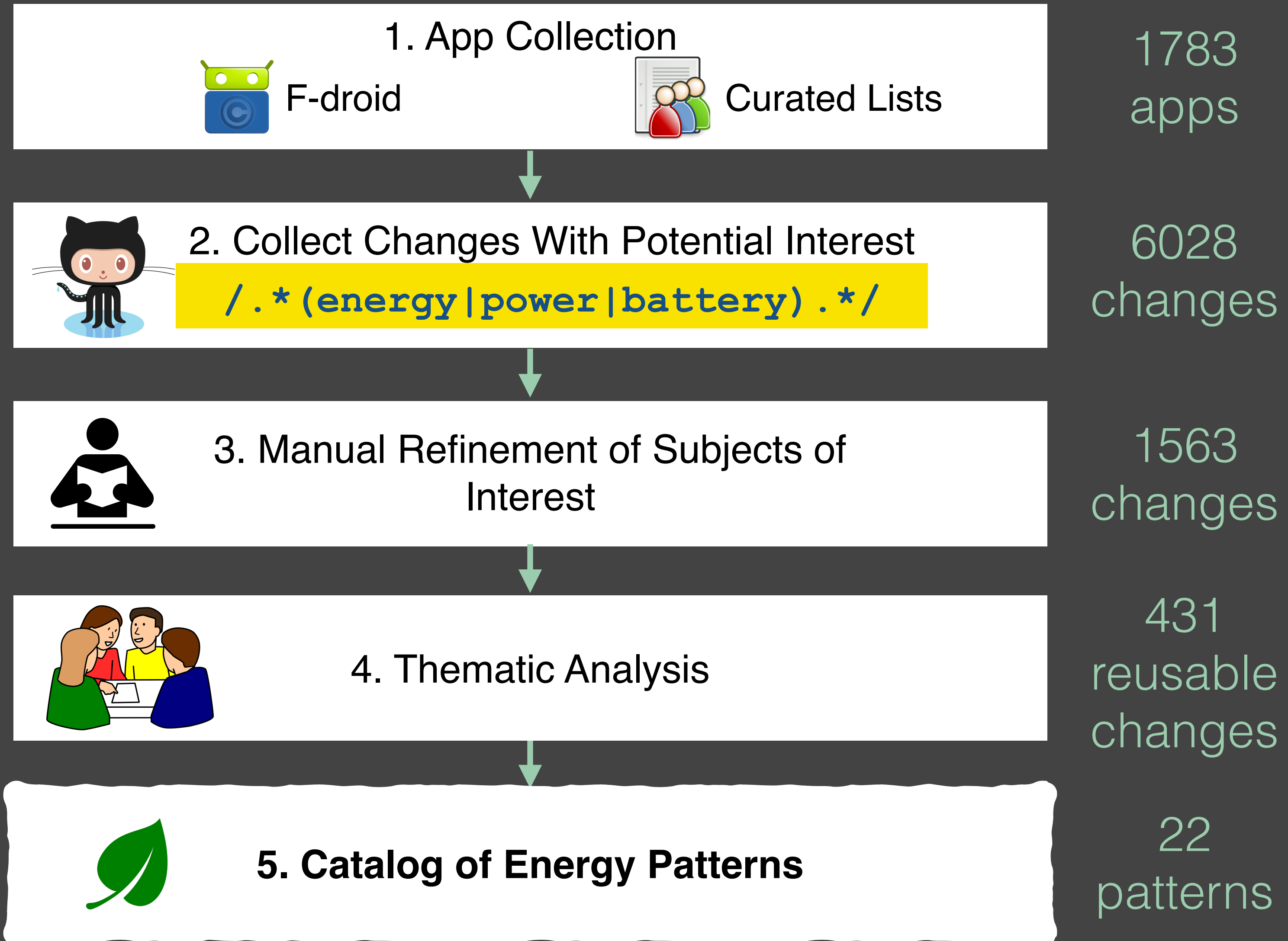
1 Introduction

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
luisacruz@fc.up.pt

Methodology



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).

Energy Patterns for Mobile Apps

A visualization with prevalence and co-occurrence of patterns can be found [here](#).

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

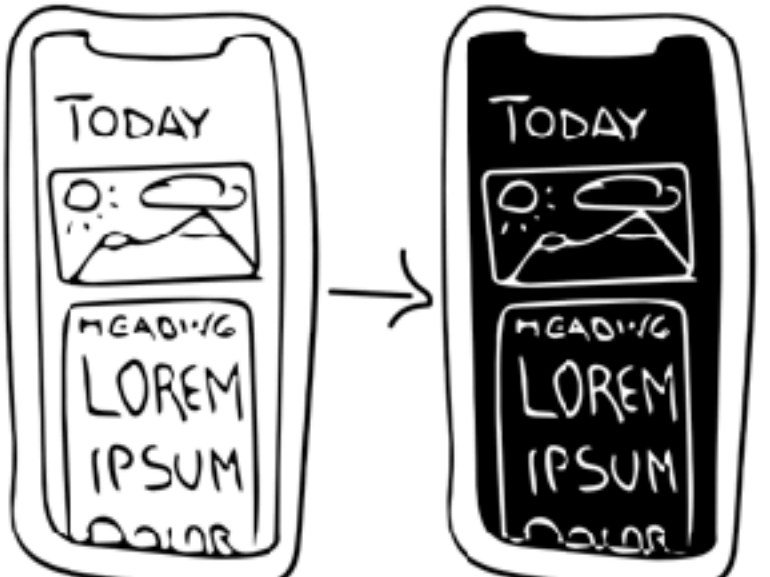
[← show all patterns](#)

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

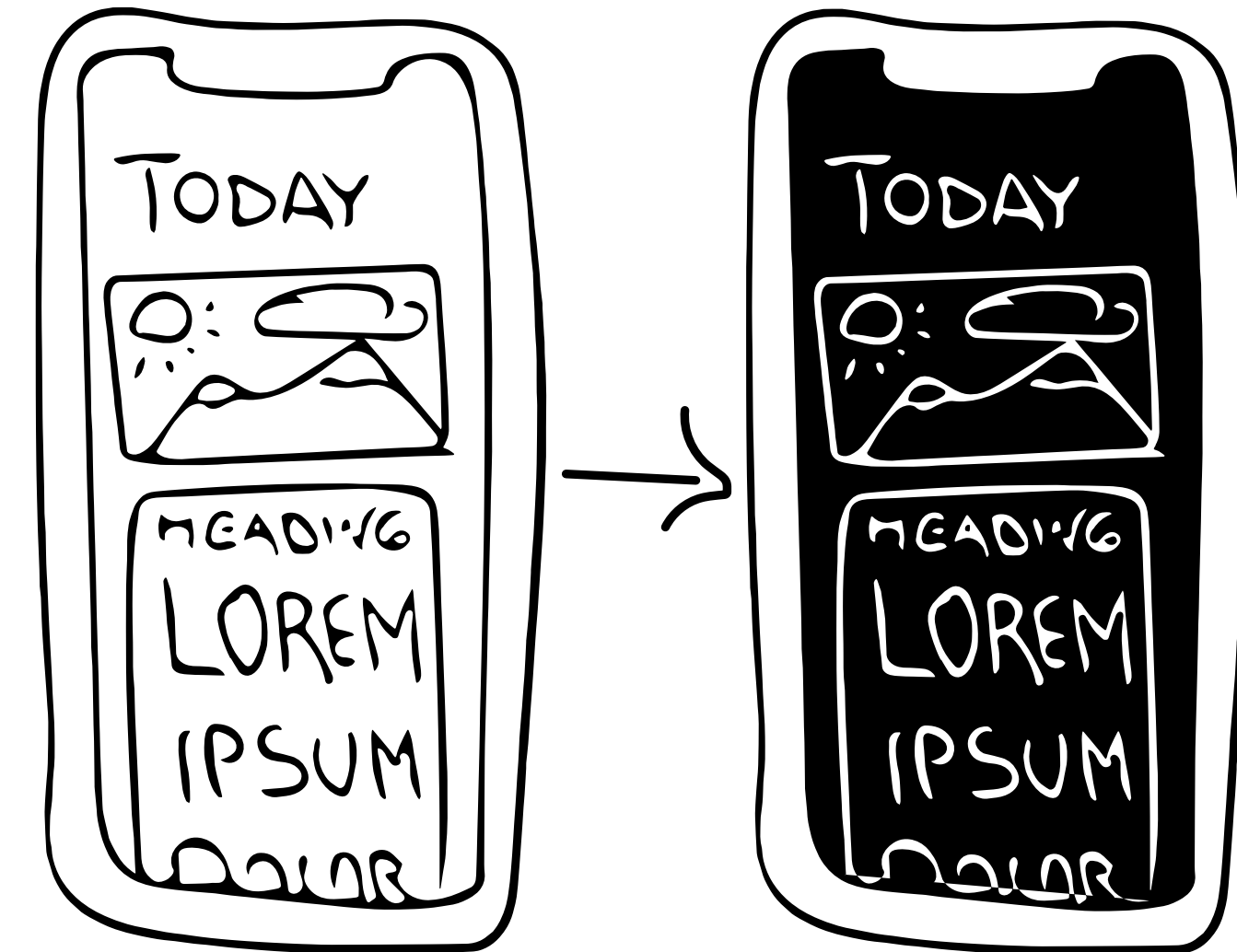
Provide a UI with dark background colors. This is particularly beneficial for mobile devices with AMOLED screens, which are more energy efficient when displaying dark colors. In some cases, it might be reasonable to allow users to choose between a light and a dark theme. The dark theme can also be activated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

Display a menu

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

Dark UI Colors

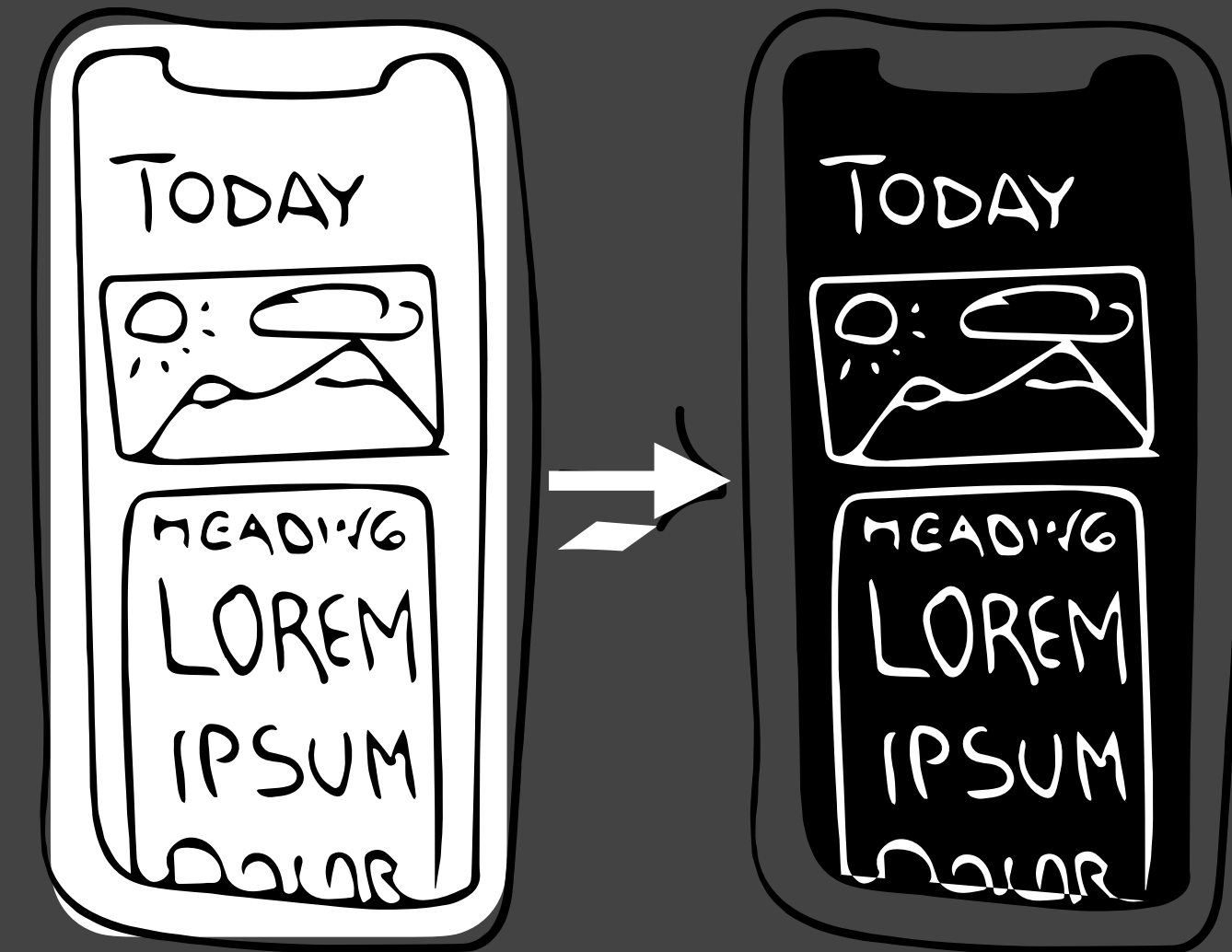
Provide a dark UI color theme to save battery on devices with AMOLED screens.



- **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

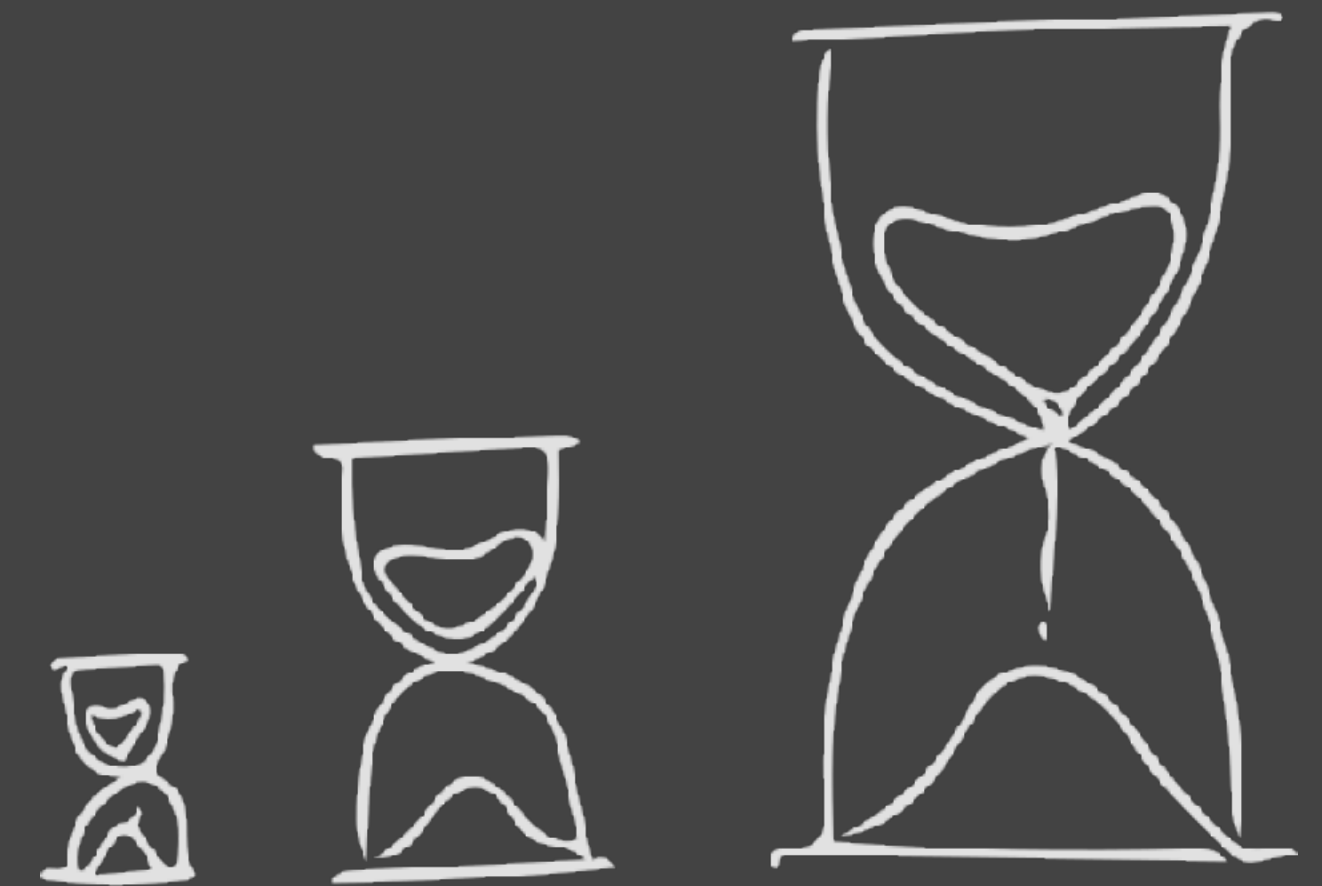
Provide a dark UI color theme to save battery on devices with AMOLED screens.



- **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

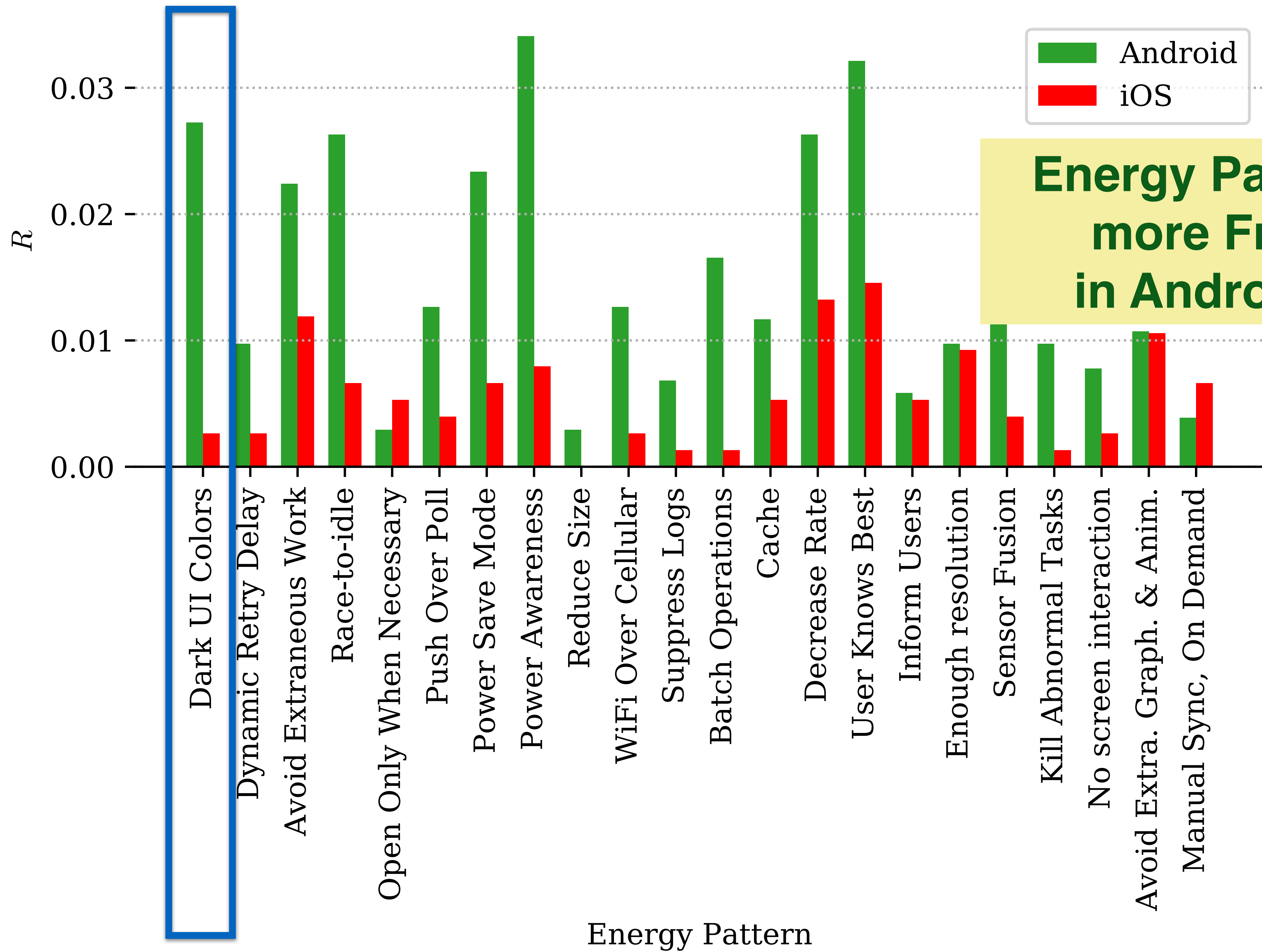


- **Context:** Executing operations separately leads to extraneous tail energy consumptions
- **Solution:** Bundle multiple operations in a single one. [...]
- **Example:** Use system provided APIs to schedule background tasks. These APIs, guarantee that device will exit sleep mode only when there is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

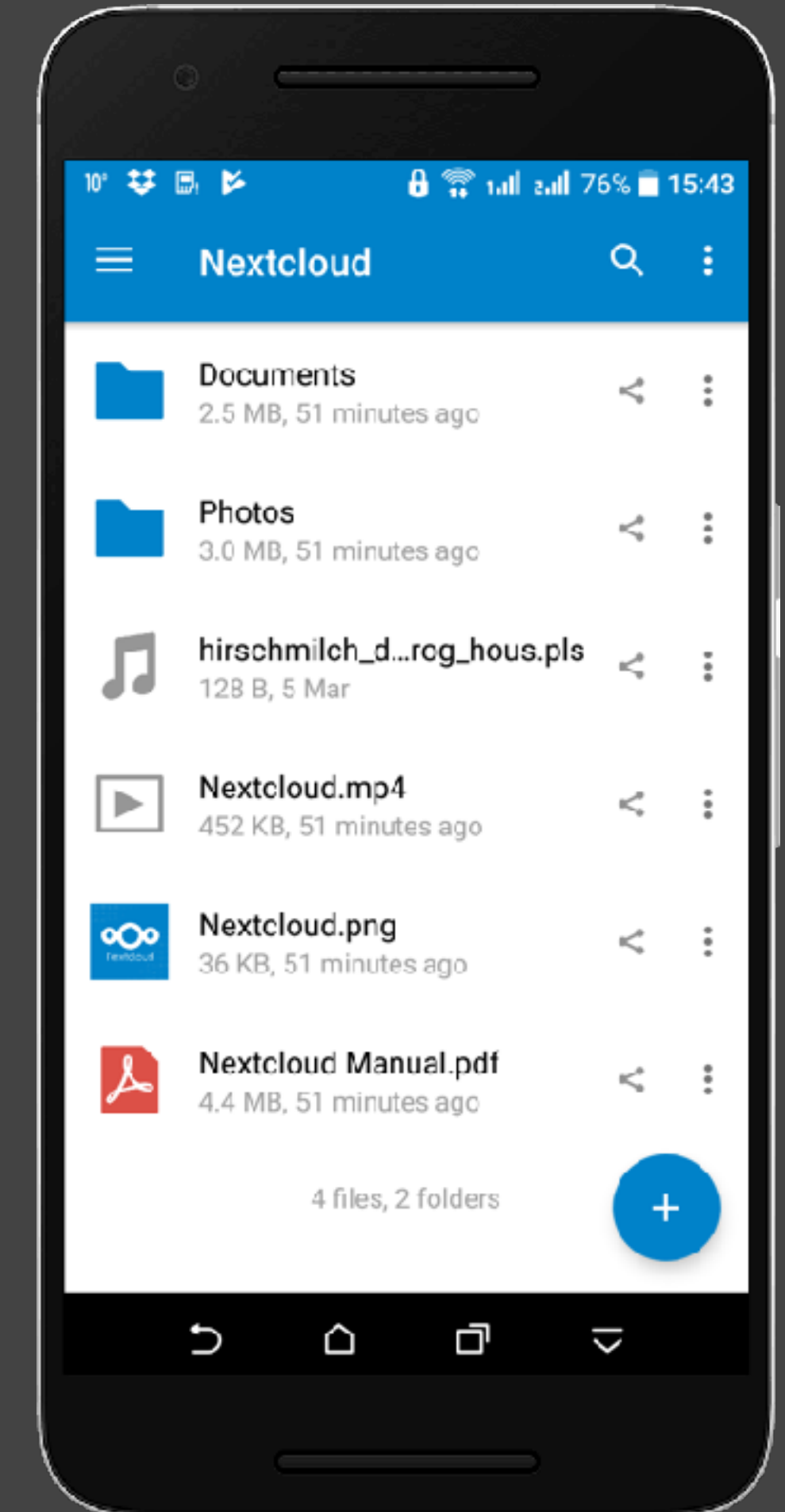
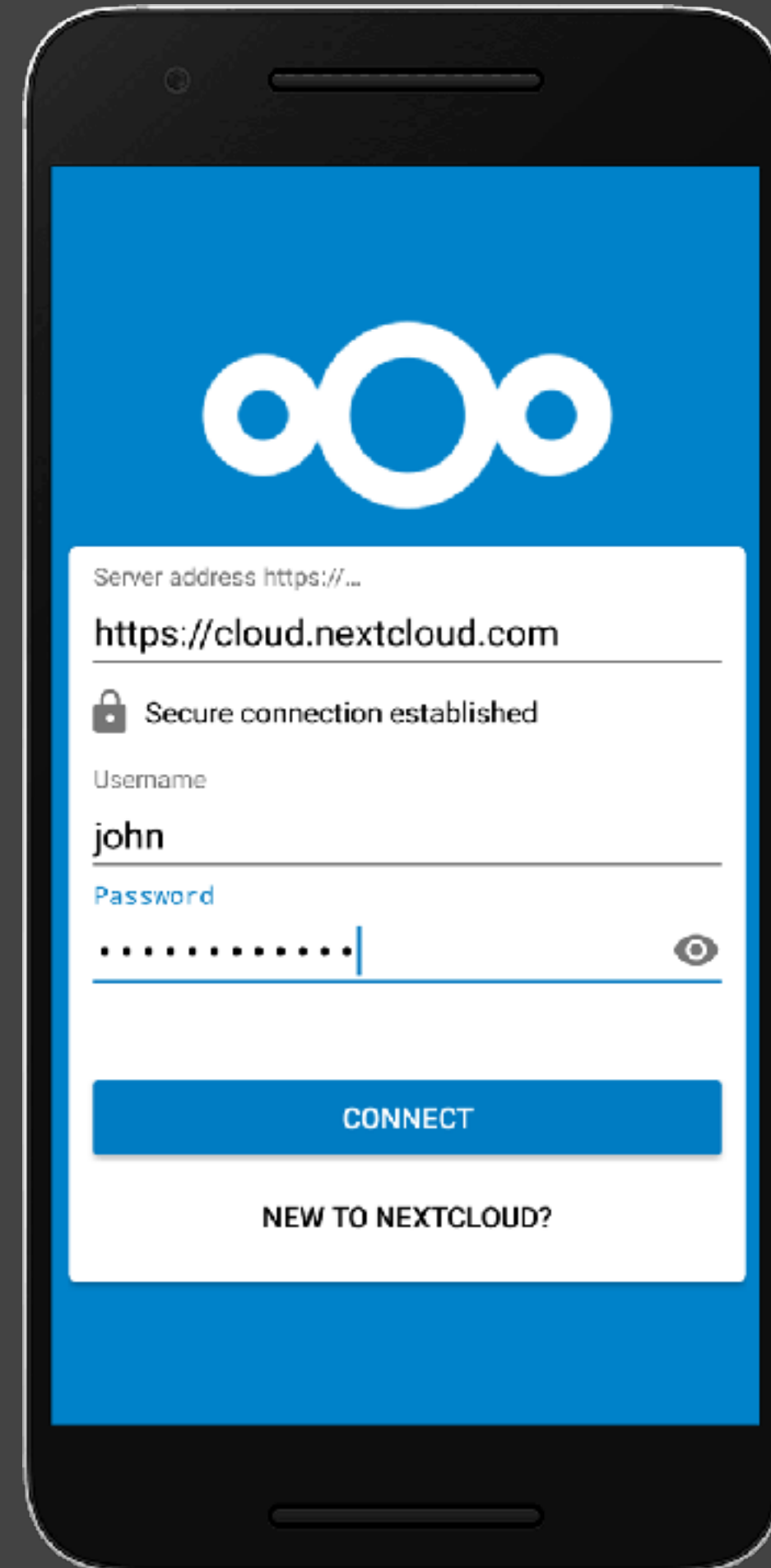
Avoid Extraneous Graphics and Animations

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

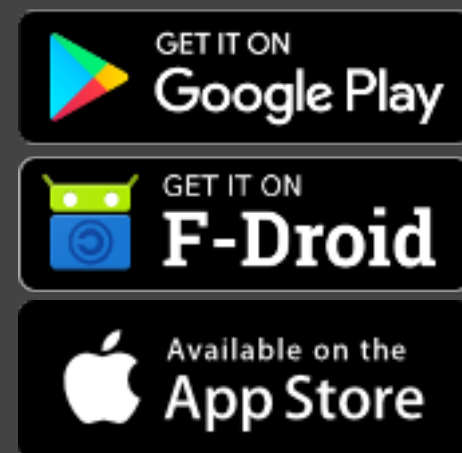
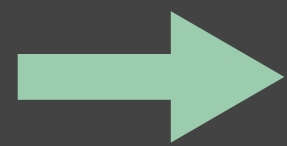
- **Context:** Mobile apps that feature impressive graphics and animations. [...]
- **Solution:** Study the importance of graphics and animations to the user experience and reduce them when applicable. [...]
- **Example:** Resort to low frame rates for animations when possible.



Example case: Nextcloud



FOSS

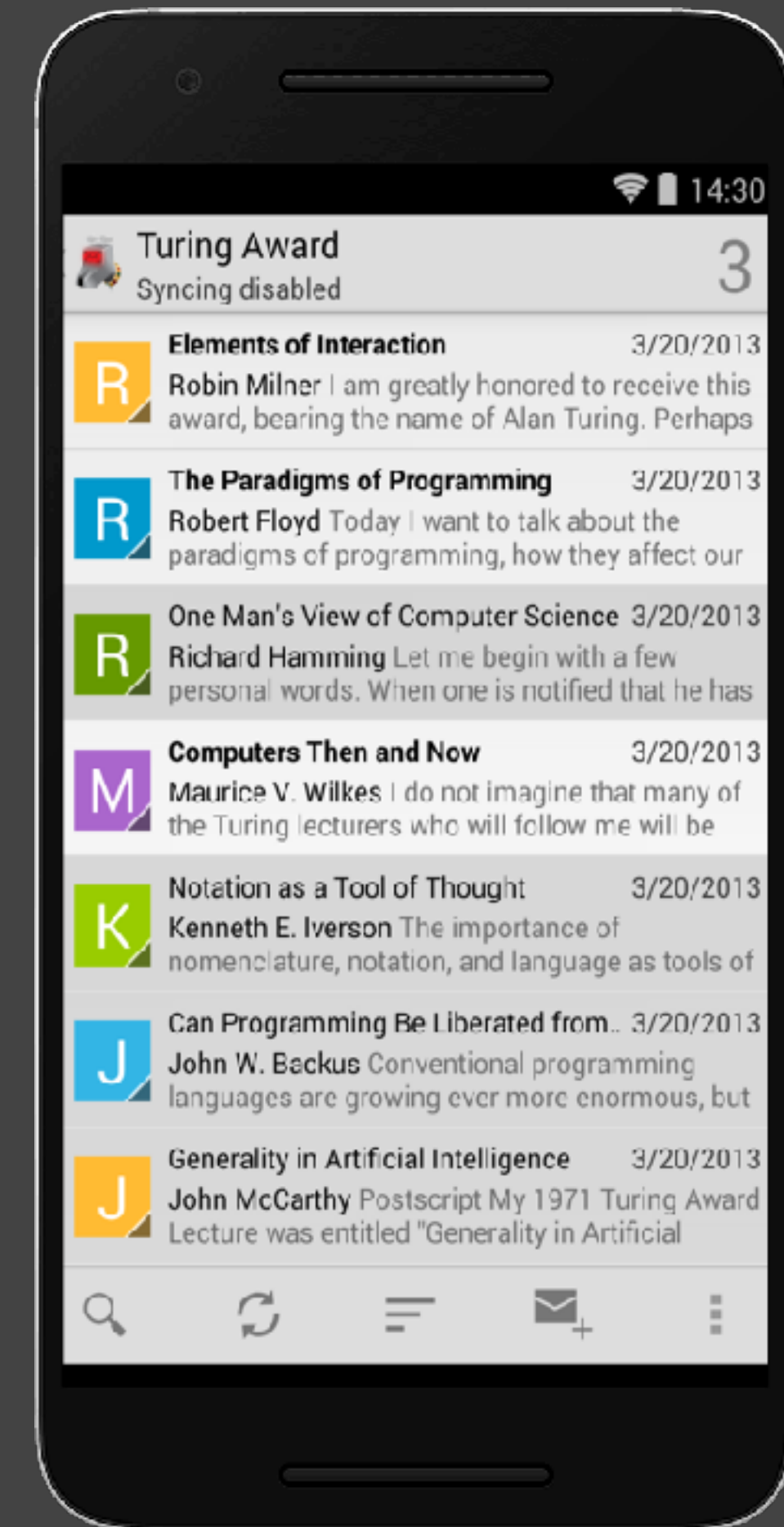
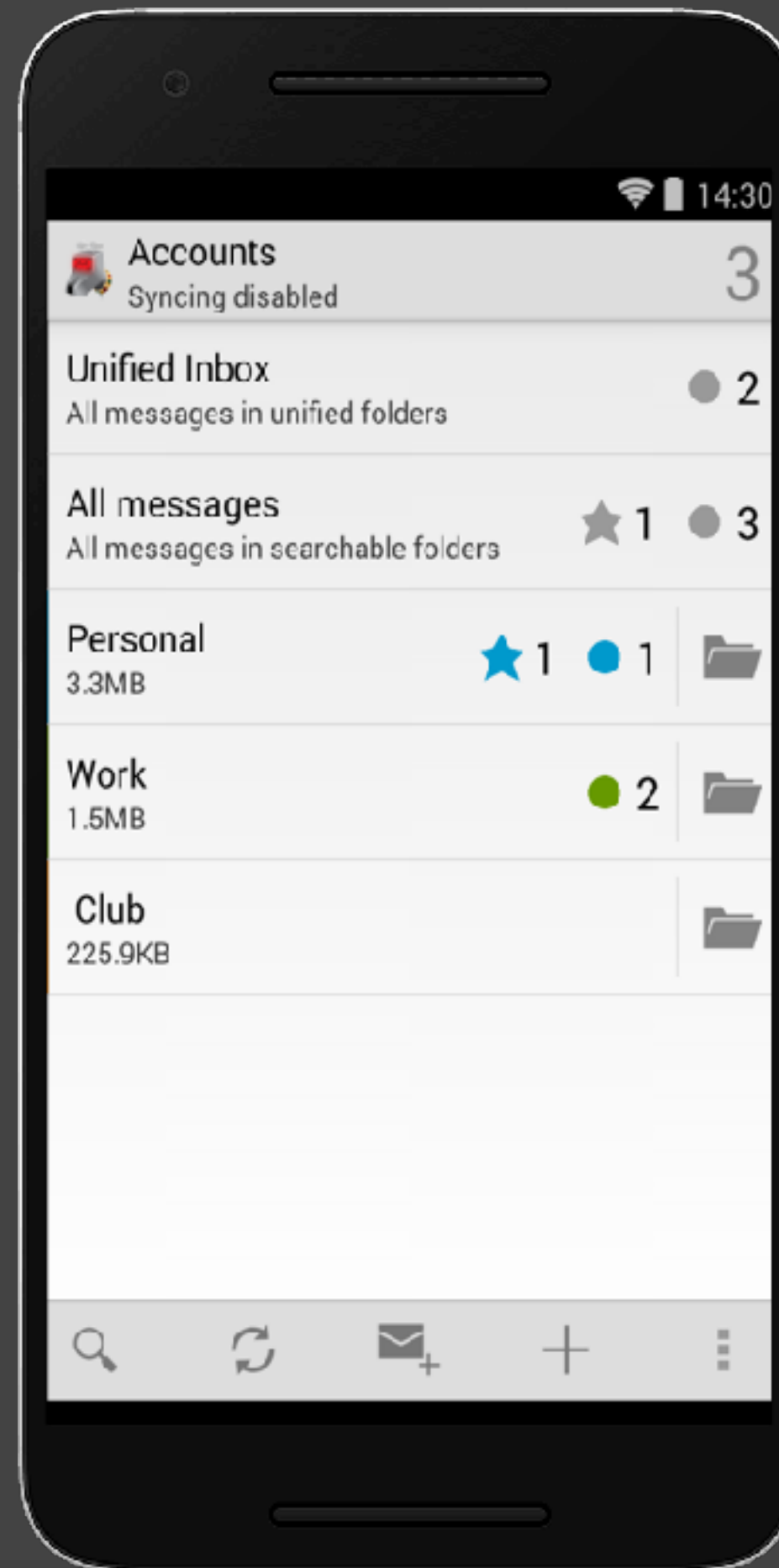
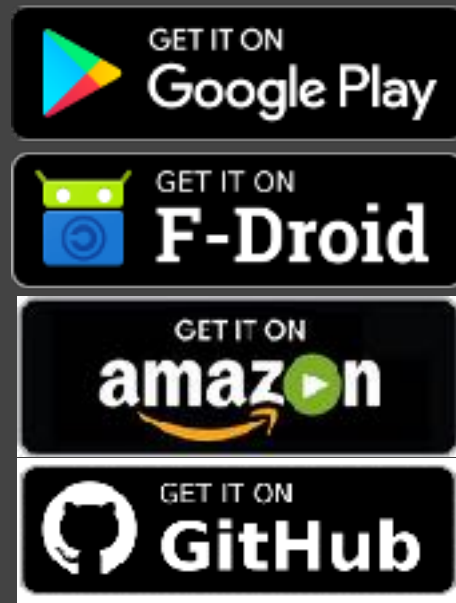


Example case: Nextcloud

- 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



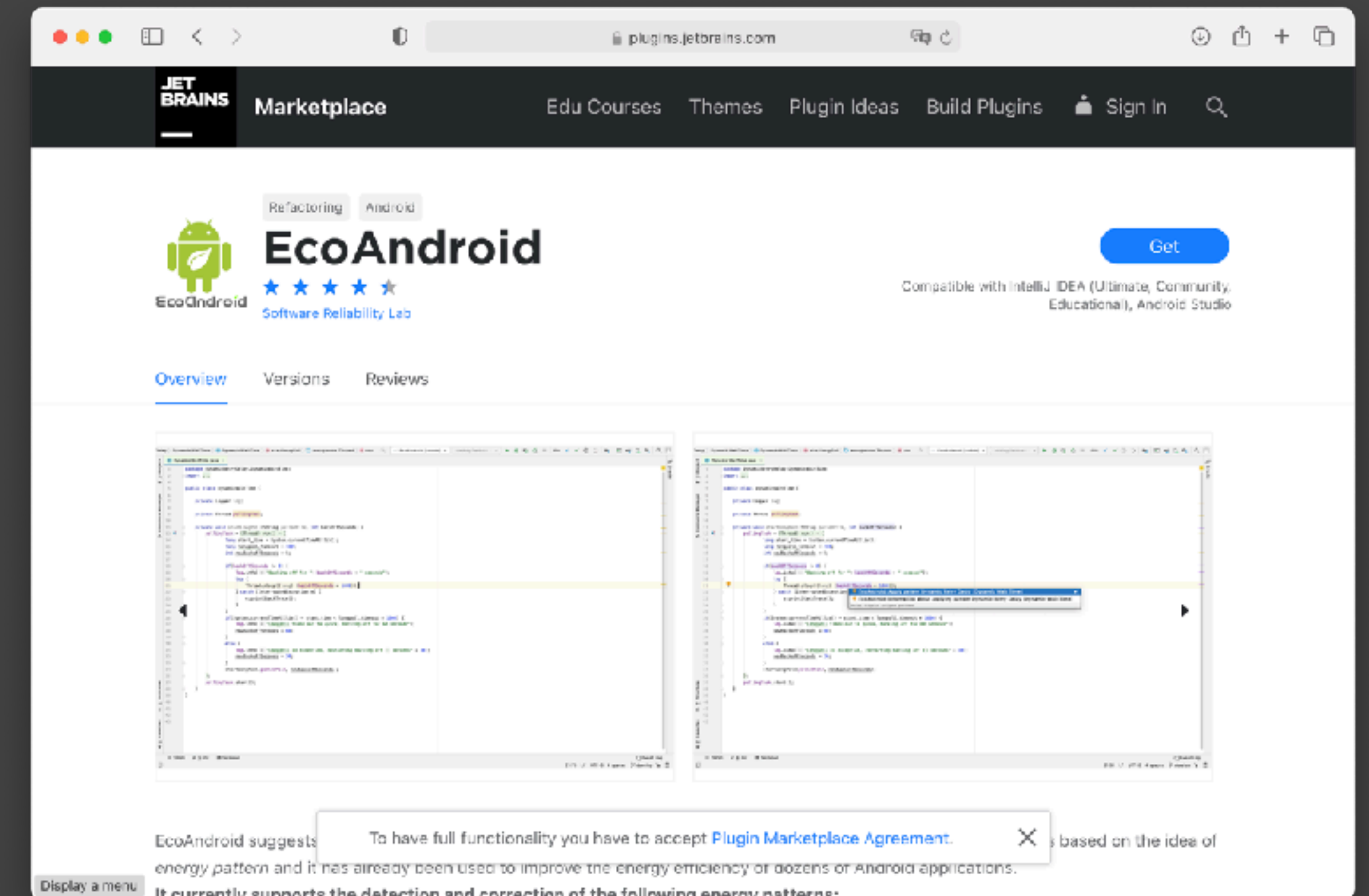
Example case: K-9 mail

- 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



Energy Efficiency Across Programming Languages

Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva

Energy Efficiency across Programming Languages

How Do Energy, Time, and Memory Relate?

Rui Pereira
HASLab/INESC TEC
Universidade do Minho, Portugal
rui.pereira@di.uminho.pt

Marco Couto
HASLab/INESC TEC
Universidade do Minho, Portugal
marco.couto@inesctec.pt

Francisco Ribeiro, Rui Rua
HASLab/INESC TEC
Universidade do Minho, Portugal
fribeiro@di.uminho.pt
rrua@di.uminho.pt

Jácome Cunha
NOVA LINCS, DI, FCT
Univ. Nova de Lisboa, Portugal
jacume@fct.unl.pt

João Paulo Fernandes
Release/LISP, CISTIC
Universidade de Coimbra, Portugal
jpf@dei.uc.pt

João Saraiva
HASLab/INESC TEC
Universidade do Minho, Portugal
saraiva@di.uminho.pt

Abstract

This paper presents a study of the runtime, memory usage and energy consumption of twenty seven well known software languages. We monitor the performance of such languages using ten different programming problems, expressed in each of the languages. Our results show interesting findings, such as, slower/faster languages consuming less/more energy, and how memory usage influences energy consumption. We show how to use our results to provide software engineers support to decide which language to use when energy efficiency is a concern.

CCS Concepts • Software and its engineering → Software performance; General programming languages;

Keywords Energy Efficiency, Programming Languages, Language Benchmarking, Green Software

ACM Reference Format:

Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2017. Energy Efficiency across Programming Languages: How Do Energy, Time, and Memory Relate?. In *Proceedings of 2017 ACM SIGPLAN International Conference on Software Language Engineering (SLE'17)*. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3136014.3136031>

1 Introduction

Software language engineering provides powerful techniques and tools to design, implement and evolve software languages. Such techniques aim at improving programmers

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SLE'17, October 25–26, 2017, Vancouver, Canada
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5523-6/17/10...\$15.00
<https://doi.org/10.1145/3136014.3136031>

productivity - by incorporating advanced features in the language design, like for instance powerful modular and type systems - and at efficiently execute such software - by developing, for example, aggressive compiler optimizations. Indeed, most techniques were developed with the main goal of helping software developers in producing faster programs. In fact, in the last century *performance* in software languages was in almost all cases synonymous of *fast execution time* (embedded systems were probably the single exception).

In this century, this reality is quickly changing and software energy consumption is becoming a key concern for computer manufacturers, software language engineers, programmers, and even regular computer users. Nowadays, it is usual to see mobile phone users (which are powerful computers) avoiding using CPU intensive applications just to save battery/energy. While the concern on the computers' energy efficiency started by the hardware manufacturers, it quickly became a concern for software developers too [28]. In fact, this is a recent and intensive area of research where several techniques to analyze and optimize the energy consumption of software systems are being developed. Such techniques already provide knowledge on the energy efficiency of data structures [15, 27] and android language [25], the energy impact of different programming practices both in mobile [18, 22, 31] and desktop applications [26, 32], the energy efficiency of applications within the same scope [8, 17], or even on how to predict energy consumption in several software systems [4, 14], among several other works.

An interesting question that frequently arises in the software energy efficiency area is whether a *faster program* is also an *energy efficient program*, or not. If the answer is yes, then optimizing a program for speed also means optimizing it for energy, and this is exactly what the compiler construction community has been hardly doing since the very beginning of software languages. However, energy consumption does not depends only on execution time, as shown in the equation $E_{energy} = T_{time} \times P_{power}$. In fact, there are several research works showing different results regarding

- Is a faster programming language also more energy efficient?
- Comparing different programming languages is not an easy task.
 - They differ in many mechanisms:
 - Interpreted vs Compiled
 - Optimisations at the compiler level
 - Virtual machine
 - Garbage collector
 - Available libraries

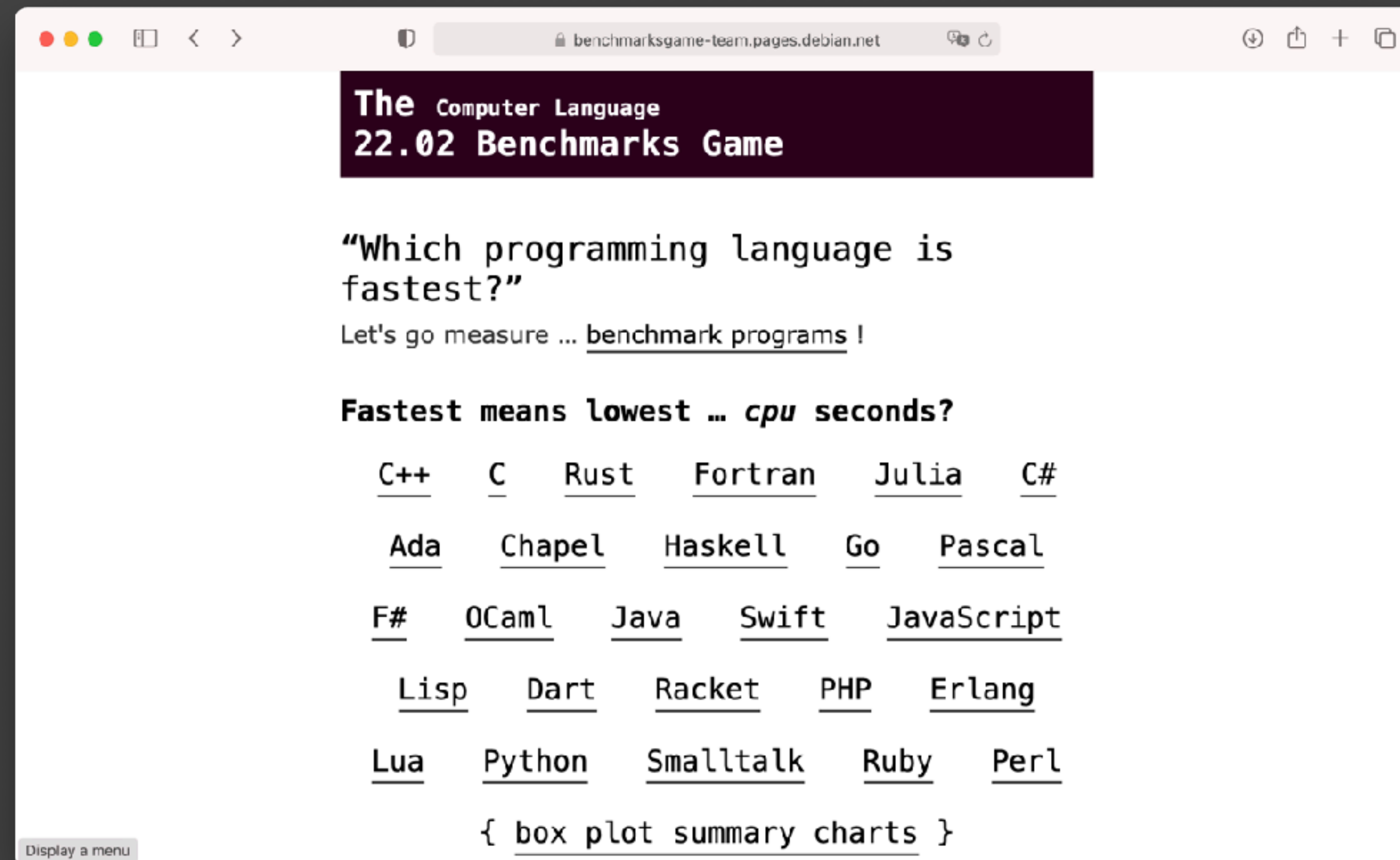
Research Questions

- **Can we compare** the energy efficiency of software languages?
- Is the **faster** language always the most **energy efficient**?
- ~~How does **memory usage** relate to **energy consumption**? (We don't cover this one)~~
- Can we automatically decide what is the best programming language considering **energy**, **time**, and **memory usage**?
- How do the results of our energy consumption analysis of programming languages gathered from rigorous **performance benchmarking solutions compare to** results of **average day-to-day solutions**?

Methodology

The Computer Language Benchmarks Game

- <https://benchmarksgame-team.pages.debian.net/benchmarksgame/>



Problems in the Computer Language Benchmarks Game

Benchmark	Description	Input
n-body	Double precision N-body simulation	50M
fannkuch-redux	Indexed access to tiny integer sequence	12
spectral-norm	Eigenvalue using the power method	5,500
mandelbrot	Generate Mandelbrot set portable bitmap file	16,000
pidigits	Streaming arbitrary precision arithmetic	10,000
regex-redux	Match DNA 8mers and substitute magic patterns	fasta output
fasta	Generate and write random DNA sequences	25M
k-nucleotide	Hashtable update and k-nucleotide strings	fasta output
reverse-complement	Read DNA sequences, write their reverse-complement	fasta output
binary-trees	Allocate, traverse and deallocate many binary trees	21
chameneos-redux	Symmetrical thread rendezvous requests	6M
meteor-contest	Search for solutions to shape packing puzzle	2,098
thread-ring	Switch from thread to thread passing one token	50M

- 27 Programming languages across different paradigms
 - **Functional** (e.g., Ocaml, F#, Haskell)
 - **Imperative** (e.g., C, Go, Python)
 - **Object-oriented** (e.g., C++, C#, Java)
 - Scripting (or **interpretative**) (e.g., JavaScript, Python, Ruby)
 - (These are not mutual exclusive)
- **Intel RAPL**'s C library to measure energy consumption

- Execute each benchmark solution 10 times.
 - Collect energy data and timestamps.
- **Two-minute interval** between executions

binary-trees

	Energy (J)	Time (ms)	Ratio (J/ms)	Mb
(c) C	39.80	1125	0.035	131
(c) C++	41.23	1129	0.037	132
(c) Rust ↓ ₂	49.07	1263	0.039	180
(c) Fortran ↑ ₁	69.82	2112	0.033	133
(c) Ada ↓ ₁	95.02	2822	0.034	197
(c) Ocaml ↓ ₁ ↑ ₂	100.74	3525	0.029	148
(v) Java ↑ ₁ ↓ ₁₆	111.84	3306	0.034	1120
(v) Lisp ↓ ₃ ↓ ₃	149.55	10570	0.014	373
(v) Racket ↓ ₄ ↓ ₆	155.81	11261	0.014	467
(i) Hack ↑ ₂ ↓ ₉	156.71	4497	0.035	502
(v) C# ↓ ₁ ↓ ₁	189.74	10797	0.018	427
(v) F# ↓ ₃ ↓ ₁	207.13	15637	0.013	432
(c) Pascal ↓ ₃ ↑ ₅	214.64	16079	0.013	256
(c) Chapel ↑ ₅ ↑ ₄	237.29	7265	0.033	335
(v) Erlang ↑ ₅ ↑ ₁	266.14	7327	0.036	433
(c) Haskell ↑ ₂ ↓ ₂	270.15	11582	0.023	494
(i) Dart ↓ ₁ ↑ ₁	290.27	17197	0.017	475
(i) JavaScript ↓ ₂ ↓ ₄	312.14	21349	0.015	916
(i) TypeScript ↓ ₂ ↓ ₂	315.10	21686	0.015	915
(c) Go ↑ ₃ ↑ ₁₃	636.71	16292	0.039	228
(i) Jruby ↑ ₂ ↓ ₃	720.53	19276	0.037	1671
(i) Ruby ↑ ₅	855.12	26634	0.032	482
(i) PHP ↑ ₃	1,397.51	42316	0.033	786
(i) Python ↑ ₁₅	1,793.46	45003	0.040	275
(i) Lua ↓ ₁	2,452.04	209217	0.012	1961
(i) Perl ↑ ₁	3,542.20	96097	0.037	2148
(c) Swift		n.e.		

binary-trees

	Energy (J)	Time (ms)	Ratio (J/ms)	Mb
(c) C	39.80	1125	0.035	131
(c) C++	41.23	1129	0.037	132
(c) Rust ↓ ₂	49.07	1263	0.039	180
(c) Fortran ↑ ₁	69.82	2112	0.033	133
(c) Ada ↓ ₁	95.02	2822	0.034	197
(c) Ocaml ↓ ₁ ↑ ₂	100.74	3525	0.029	148
(v) Java ↑ ₁ ↓ ₁₆	111.84	3306	0.034	1120
(v) Lisp ↓ ₃ ↓ ₃	149.55	10570	0.014	373
(v) Racket ↓ ₄ ↓ ₆	155.81	11261	0.014	467
(i) Hack ↑ ₂ ↓ ₉	156.71	4497	0.035	502
(v) C# ↓ ₁ ↓ ₁	189.74	10797	0.018	427
(v) F# ↓ ₃ ↓ ₁	207.13	15637	0.013	432
(c) Pascal ↓ ₃ ↑ ₅	214.64	16079	0.013	256
(c) Chapel ↑ ₅ ↑ ₄	237.29	7265	0.033	335
(v) Erlang ↑ ₅ ↑ ₁	266.14	7327	0.036	433
(c) Haskell ↑ ₂ ↓ ₂	270.15	11582	0.023	494
(i) Dart ↓ ₁ ↑ ₁	290.27	17197	0.017	475
(i) JavaScript ↓ ₂ ↓ ₄	312.14	21349	0.015	916
(i) TypeScript ↓ ₂ ↓ ₂	315.10	21686	0.015	915
(c) Go ↑ ₃ ↑ ₁₃	636.71	16292	0.039	228
(i) Jruby ↑ ₂ ↓ ₃	720.53	19276	0.037	1671
(i) Ruby ↑ ₅	855.12	26634	0.032	482

Normalized global results for Energy, Time, and Memory.

Total					
	Energy (J)		Time (ms)		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

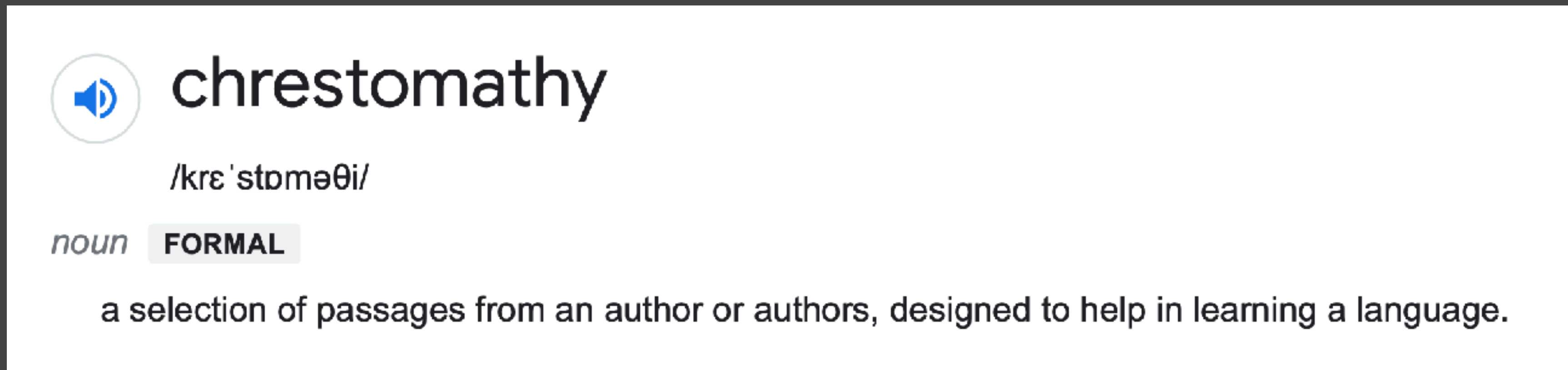
Critical thinking

- There is no doubt this is an excellent study. Yet, as any excellent study, there's a lot we can **discuss** and **criticise** constructively.
- What kind of issues you see in drawing conclusions from such a table of results?
 - Is the benchmark representative of the most common usage behaviour?
 - Are the implemented solutions representative?
 - Does it make sense to use the average to compare energy consumption across different problems?
 - ...

Reproducing with Rosetta Code

(?)

- Rosetta Code is a **programming chrestomathy repository**



The screenshot shows a dictionary entry for the word "chrestomathy". It includes a speaker icon for audio pronunciation, the phonetic transcription "/krɛ'stɒməθi/", the part of speech "noun", and a "FORMAL" label. The definition is "a selection of passages from an author or authors, designed to help in learning a language."

- 900 usecases/tasks solved across 700 different programming languages
- **Purpose:** if you know a programming language we can easily learn how the same task is solved in a language you are not familiar with.

Remove-duplicates

	Energy (J)	Time (ms)
(c) Rust	0.01	1
(c) C++	0.12	5
(c) C	0.14	10
(c) Go	0.32	13
(i) Lua	0.51	21
(i) Perl	1.31	53
(i) JavaScript	1.73	73
(v) Erlang	2.36	96
(v) Java	2.96	214
(i) PHP	2.99	121
(i) Python	4.93	206
(i) Ruby	6.13	259
(v) Racket	7.54	318

Rosetta Code global ranking based on Energy.

Rosetta Code Global Ranking

Position	Language
1	C
2	Pascal
3	Ada
4	Rust
5	C++, Fortran
6	Chapel
7	OCaml, Go
8	Lisp
9	Haskell, JavaScript
10	Java
11	PHP
12	Lua, Ruby
13	Perl
14	Dart, Racket, Erlang
15	Python

Revisiting Research Questions

- **Can we compare** the energy efficiency of software languages?
- Is the **faster** language always the most **energy efficient**?
- ~~How does **memory usage** relate to **energy consumption**?~~
- Can we automatically decide what is the best programming language considering **energy, time, and memory usage**?
- How do the results of our energy consumption analysis of programming languages gathered from rigorous **performance benchmarking solutions compare to** results of **average day-to-day solutions**?
 - What would happen if we cherry picked the tasks?

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

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

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

Abstract—The amount of CO₂ 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

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, phthang@google.com, ncare@google.com, stalukdar@google.com, ericmullen@google.com, kendalsmith@google.com, mcottman@google.com, walfredo@google.com)

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.

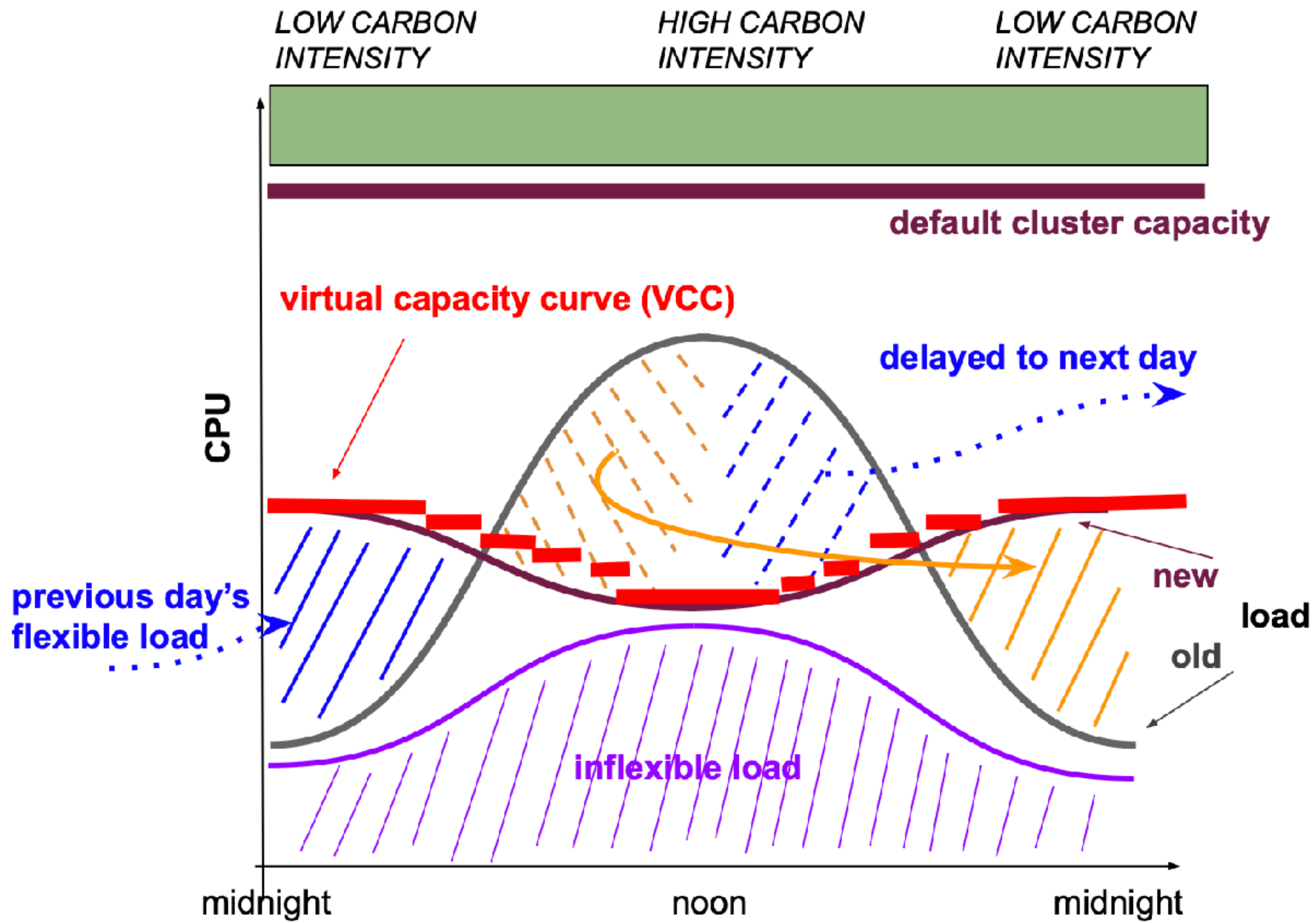
This paper describes the methodology and principles behind Google's system for Carbon-Intelligent Compute management, which reduces grid carbon emissions from Google's datacenter electricity use and reduces operating costs by increasing resource and power efficiency. To accomplish this goal, the system 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.

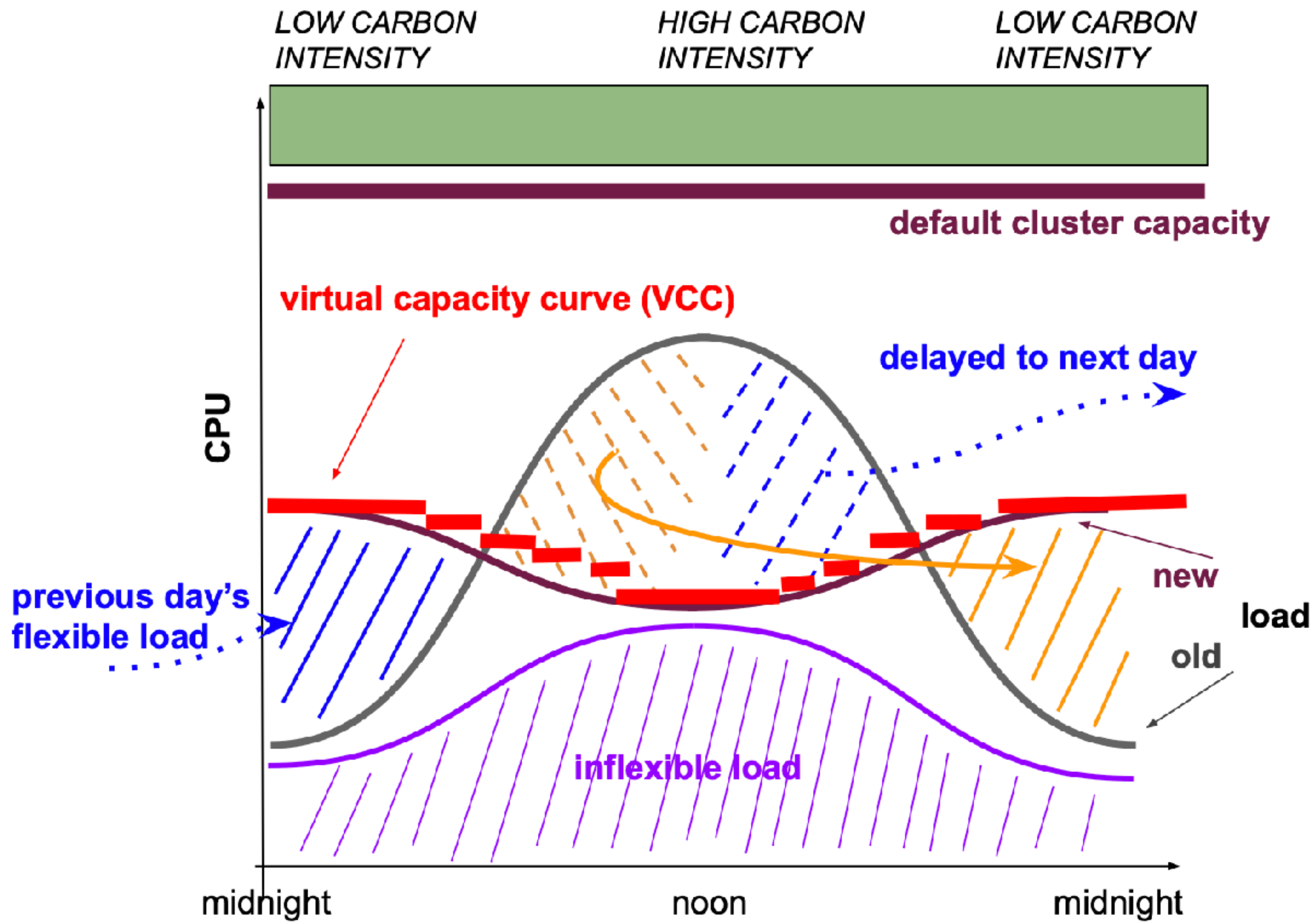
The cluster-level VCCs are pushed to all of Google's datacenter clusters prior to the start of the next day, where they

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



- 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



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!
- Factors in **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 facilities' 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 might lead to shifting 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 falls 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?