Architecting for Sustainability

Software Architecture (IN4315)

@luismcruz
L.Cruz@tudelft.nl
https://luiscruz.github.io/

If you are worried about someone or you are in need of guidance yourself, don't wait. We may not physically be together at EWI but we are a community in which there is always someone to talk to. You can contact:

- Your own academic counsellor by making an appointment here or writing an email to ac-msc-eemcs@tudelft.nl
- One of the student psychologists at psychologen@tudelft.nl
- Your own GP or the SGZ.
- There is also a hotline for suicide prevention: www.113.nl or call 0800-0113 (this is anonymous)
- Is there an emergency on campus? Call 015 – 278 8888 of 112 (24/7)
- MoTiv has group and individual consultations - call 015 2006060 (16:00 – 18:00 en 20:00 – 22:00)

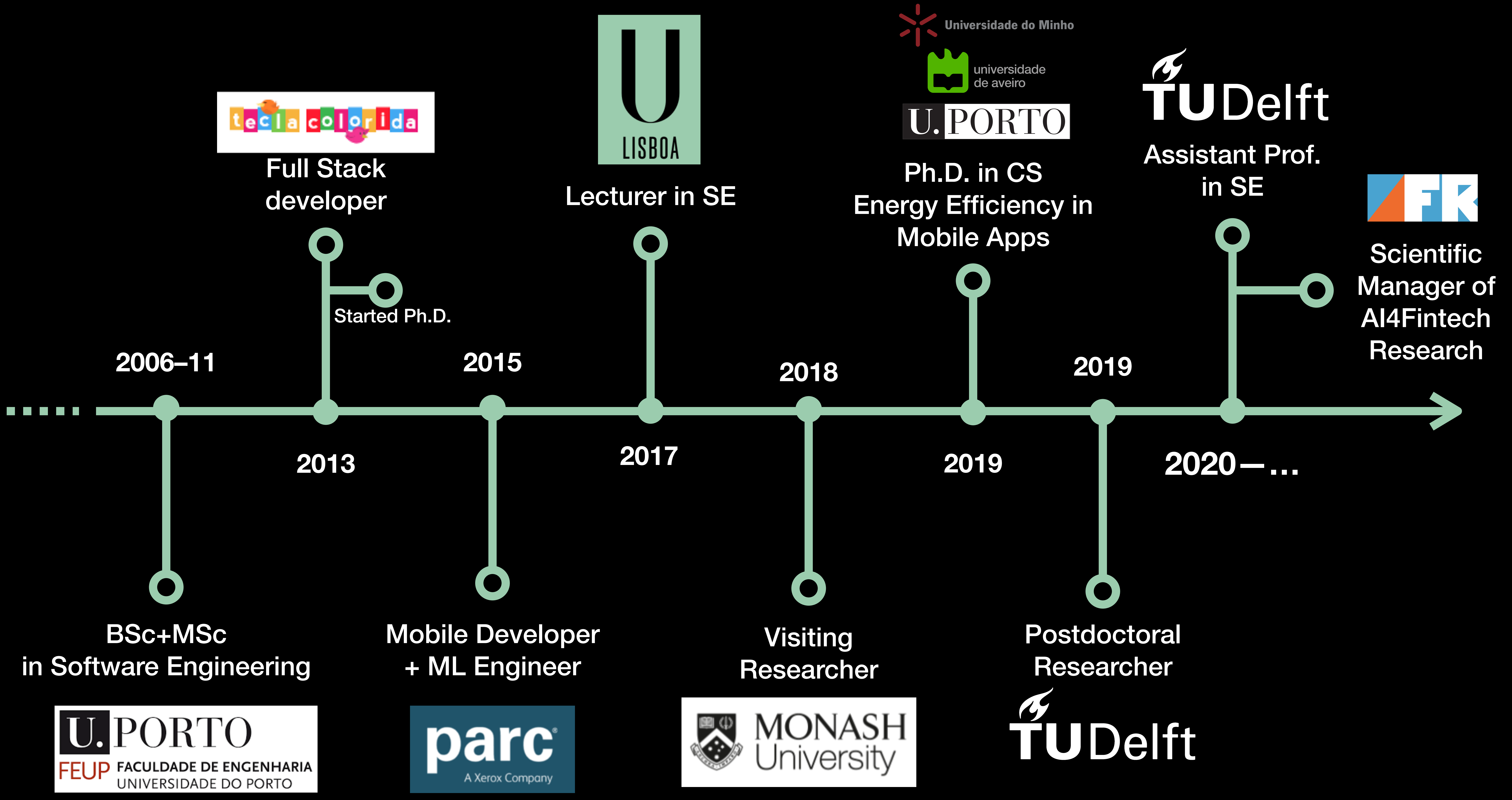
We are all looking for a bit of balance within the limitations of the current situation. Variety is, after all, the spice of life and this not only applies to food!

How do you balance between studying and time off? Finding time to enjoy your courses, getting to know people in a project, managing your time: you can find all kinds of tips, tools and support at TUD's wellbeing and study page.

Are you looking for online social events to meet up with fellow students? Check out CH's calendar, ETV's calendar or Delft/SEA. Activities are in English and open to everybody.

Looking for ways to energize yourself by trying something new? There is a well being week with all kinds of different activities starting from February onwards. X has all kinds of online courses on offer, also the fields and courts are open again. You can even go to a Gardening Quiz. Also, MoTiv offers inspiration workshops in May.

Opening up and talking to the people around you is a vital part of keeping your perspective. Finding students to work together with, meeting up frequently to check in on each other. For practical tips on taking initiative and finding balance between studying and time off: you are also welcome to make an appointment with an academic counsellor at EWI.


2006–11
BSc+MSc
in Software Engineering
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO
2013
tecla colorida
Full Stack
developer
Started Ph.D.
2015
Mobile Developer
+ ML Engineer
parc
A Xerox Company
2017
U
LISBOA
Lecturer in SE
2018
Visiting
Researcher
MONASH
University
2019
Universidade do Minho
universidade
de aveiro
U.PORTO
Ph.D. in CS
Energy Efficiency in
Mobile Apps
2019
Postdoctoral
Researcher
TUDelft
2020—...
TUDelft
Assistant Prof.
in SE
Scientific
Manager of
AI4Fintech
Research

AI for Fintech Research (AFR)

AI for Fintech Research (AFR)

- Collaboration between **ING** and **TU Delft**.

AI for Fintech Research (AFR)

- Collaboration between **ING** and **TU Delft**.
- Artificial Intelligence, Data Analytics, and Software Analytics in the context of **FinTech**.

AI for Fintech Research (AFR)

- Collaboration between **ING** and **TU Delft**.
- Artificial Intelligence, Data Analytics, and Software Analytics in the context of **FinTech**.
- Officially started in January 2020.

AI for Fintech Research (AFR)

- Collaboration between **ING** and **TU Delft**.
- Artificial Intelligence, Data Analytics, and Software Analytics in the context of **FinTech**.
- Officially started in January 2020.
- 11 research tracks, 5 years.

AI for Fintech Research (AFR)

- Collaboration between **ING** and **TU Delft**.
- Artificial Intelligence, Data Analytics, and Software Analytics in the context of **FinTech**.
- Officially started in January 2020.
- 11 research tracks, 5 years.
- Website: https://se.ewi.tudelft.nl/ai4fintech/

AI for Fintech Research (AFR)

- Collaboration between **ING** and **TU Delft**.
- Artificial Intelligence, Data Analytics, and Software Analytics in the context of **FinTech**.
- Officially started in January 2020.
- 11 research tracks, 5 years.
- Website: https://se.ewi.tudelft.nl/ai4fintech/
- Twitter: @Ai4Fintech

Outline

1. Sustainable Software Engineering.

Outline

1. Sustainable Software Engineering.

2. Measuring Energy Consumption.

Outline

Outline

1. Sustainable Software Engineering.

2. Measuring Energy Consumption.

3. Energy Patterns
(Additionally/Maybe: Energy-efficient Programming Languages)

Outline

1. Sustainable Software Engineering.

2. Measuring Energy Consumption.

3. Energy Patterns
(Additionally/Maybe: Energy-efficient Programming Languages)

4. Wrap-up

Software Sustainability

Sustainability Design and Software: The Karlskrona Manifesto

Christoph Becker, Faculty of Information, University of Toronto, Toronto, ON, Canada, christoph.becker@utoronto.ca

Ruzanna Chitchyan, Dept of Computer Science, University of Leicester, Leicester, UK, rc256@leicester.ac.uk

Leticia Duboc, Dept of Inf. & Computer Science, State Univ. of Rio de Janeiro, Rio de Janeiro, Brazil, leticia@ime.uerj.br

Steve Easterbrook, Dept of Computer Science, University of Toronto, Toronto, ON, Canada, sme@cs.utoronto.ca

Birgit Penzenstadler, Institute for Software Research, University of California, Irvine, Irvine, California, US, bpenzens@uci.edu

Norbert Seyff, Dept of Informatics, University of Zurich, Zurich, Switzerland, seyff@ifi.uzh.ch

Colin C. Venters, School of Computing & Engineering, University of Huddersfield, Huddersfield, UK, c.venters@hud.ac.uk

***Abstract*—Sustainability has emerged as a broad concern for society. Many engineering disciplines have been grappling with challenges in how we sustain technical, social and ecological systems. In the software engineering community, for example, maintainability has been a concern for a long time. But too often, these issues are treated in isolation from one another. Misperceptions among practitioners and research communities persist, rooted in a lack of coherent understanding of sustainability, and how it relates to software systems research and practice. This article presents a cross-disciplinary initiative to create a common ground and a point of reference for the global community of research and practice in software and sustainability, to be used for effectively communicating key issues, goals, values and principles of sustainability design for software-intensive systems. The centrepiece of this effort is the *Karlskrona Manifesto for Sustainability Design*, a vehicle for a much needed conversation about sustainability within and beyond the software community, and an articulation of the fundamental principles underpinning design choices that affect sustainability. We describe the motivation for developing this manifesto, including some considerations of the genre of the manifesto as well as the dynamics of its creation. We illustrate the collaborative reflective writing process and present the current edition of the manifesto itself. We assess immediate implications and applications of the articulated principles, compare these to current practice, and suggest future steps.**

I. Introduction

It is clear that society is facing major sustainability challenges that require long-term, joined-up thinking. How do we sustain our technical infrastructures, given how much we rely on them for everything from communication and navigation through to storing health records, identifying security threats, and keeping the lights on? How do we sustain prosperity in society, given the erosion of trust in our political institutions and a growing inequality in ownership of resources? And, above all, how do we sustain the planetary systems that support life on earth, in the face of accumulation of pollutants, species loss, and accelerating climate change?

The discipline of Software Engineering (SE) has a major role to play in sustainability, because of the extent to which software systems mediate so many aspects of our lives. However, software practice has a tendency to focus only on the immediate effects and tangible benefits of software products and platforms. SE research has, for the most part, focused on increasing the reliability, efficiency and cost-benefit relation of software products for their owners, through a focus on processes, methods, models and techniques to create, verify and validate software systems and keep them operational.

The lack of long-term thinking in software engineering research and practice has been critiqued throughout the history of the discipline. For example, software maintenance and evolution were raised as concerns even at the very first software engineering conference [1]. Since then, efforts to increase the maintainability of software products and facilitate their evolution have often focused on improving architecture, decreasing lifecycle costs and managing technical debt [2]. Neumann has criticized the lack of long-term thinking over security considerations in SE [3]. For our digital information assets, some now speak of a 'digital dark age' [4], where, having discarded analog media in preference for digital, we now find that many of these assets become unreadable, due, in part, to the rapid lifecycles of software technology.

While progress has been made on design for maintainability of software *per se*, considerations that extend beyond immediate software product qualities and user benefits are generally treated as secondary concerns, optional qualities to be addressed only after the system under design has been shown to be a success in terms of technical and/or marketing criteria. The larger impact of software artefacts on society and the natural environment is not routinely analyzed. But by trading off longer-term sustainability questions for shorter-term success criteria, we accumulate threats to sustainability. We argue that this trade-off itself is unnecessary. As Neumann

(Becker, 2015)

Software Sustainability

- Individual. **Mental and physical well-being, education, freedom, self-respect, mobility, agency.**

Sustainability Design and Software: The Karlskrona Manifesto

Christoph Becker
Faculty of Information
University of Toronto
Toronto, ON, Canada
christoph.becker@utoronto.ca

Ruzanna Chitchyan
Dept of Computer Science
University of Leicester
Leicester, UK
rc256@leicester.ac.uk

Leticia Duboc
Dept of Inf. & Computer Science
State Univ. of Rio de Janeiro
Rio de Janeiro, Brazil
leticia@ime.uerj.br

Steve Easterbrook
Dept of Computer Science
University of Toronto
Toronto, ON, Canada
sme@cs.utoronto.ca

Birgit Penzenstadler
Institute for Software Research
University of California, Irvine
Irvine, California, US
bpenzens@uci.edu

Norbert Seyff
Dept of Informatics
University of Zurich
Zurich, Switzerland
seyff@ifi.uzh.ch

Colin C. Venters
School of Computing & Engineering
University of Huddersfield
Huddersfield, UK
c.venters@hud.ac.uk

***Abstract*—Sustainability has emerged as a broad concern for society. Many engineering disciplines have been grappling with challenges in how we sustain technical, social and ecological systems. In the software engineering community, for example, maintainability has been a concern for a long time. But too often, these issues are treated in isolation from one another. Misperceptions among practitioners and research communities persist, rooted in a lack of coherent understanding of sustainability, and how it relates to software systems research and practice. This article presents a cross-disciplinary initiative to create a common ground and a point of reference for the global community of research and practice in software and sustainability, to be used for effectively communicating key issues, goals, values and principles of sustainability design for software-intensive systems. The centrepiece of this effort is the *Karlskrona Manifesto for Sustainability Design*, a vehicle for a much needed conversation about sustainability within and beyond the software community, and an articulation of the fundamental principles underpinning design choices that affect sustainability. We describe the motivation for developing this manifesto, including some considerations of the genre of the manifesto as well as the dynamics of its creation. We illustrate the collaborative reflective writing process and present the current edition of the manifesto itself. We assess immediate implications and applications of the articulated principles, compare these to current practice, and suggest future steps.**

I. Introduction

It is clear that society is facing major sustainability challenges that require long-term, joined-up thinking. How do we sustain our technical infrastructures, given how much we rely on them for everything from communication and navigation through to storing health records, identifying security threats, and keeping the lights on? How do we sustain prosperity in society, given the erosion of trust in our political institutions and a growing inequality in ownership of resources? And, above all, how do we sustain the planetary systems that support life on earth, in the face of accumulation of pollutants, species loss, and accelerating climate change?

The discipline of Software Engineering (SE) has a major role to play in sustainability, because of the extent to which software systems mediate so many aspects of our lives. However, software practice has a tendency to focus only on the immediate effects and tangible benefits of software products and platforms. SE research has, for the most part, focused on increasing the reliability, efficiency and cost-benefit relation of software products for their owners, through a focus on processes, methods, models and techniques to create, verify and validate software systems and keep them operational.

The lack of long-term thinking in software engineering research and practice has been critiqued throughout the history of the discipline. For example, software maintenance and evolution were raised as concerns even at the very first software engineering conference [1]. Since then, efforts to increase the maintainability of software products and facilitate their evolution have often focused on improving architecture, decreasing lifecycle costs and managing technical debt [2]. Neumann has criticized the lack of long-term thinking over security considerations in SE [3]. For our digital information assets, some now speak of a 'digital dark age' [4], where, having discarded analog media in preference for digital, we now find that many of these assets become unreadable, due, in part, to the rapid lifecycles of software technology.

While progress has been made on design for maintainability of software *per se*, considerations that extend beyond immediate software product qualities and user benefits are generally treated as secondary concerns, optional qualities to be addressed only after the system under design has been shown to be a success in terms of technical and/or marketing criteria. The larger impact of software artefacts on society and the natural environment is not routinely analyzed. But by trading off longer-term sustainability questions for shorter-term success criteria, we accumulate threats to sustainability. We argue that this trade-off itself is unnecessary. As Neumann

(Becker, 2015)

Software Sustainability

- Individual. **Mental and physical well-being, education, freedom, self-respect, mobility, agency.**
- Social. **Social equity, justice, employment, democracy.**

Sustainability Design and Software: The Karlskrona Manifesto

Christoph Becker, Faculty of Information, University of Toronto, Toronto, ON, Canada, christoph.becker@utoronto.ca

Ruzanna Chitchyan, Dept of Computer Science, University of Leicester, Leicester, UK, rc256@leicester.ac.uk

Leticia Duboc, Dept of Inf. & Computer Science, State Univ. of Rio de Janeiro, Rio de Janeiro, Brazil, leticia@ime.uerj.br

Steve Easterbrook, Dept of Computer Science, University of Toronto, Toronto, ON, Canada, sme@cs.utoronto.ca

Birgit Penzenstadler, Institute for Software Research, University of California, Irvine, Irvine, California, US, bpenzens@uci.edu

Norbert Seyff, Dept of Informatics, University of Zurich, Zurich, Switzerland, seyff@ifi.uzh.ch

Colin C. Venters, School of Computing & Engineering, University of Huddersfield, Huddersfield, UK, c.venters@hud.ac.uk

***Abstract*—Sustainability has emerged as a broad concern for society. Many engineering disciplines have been grappling with challenges in how we sustain technical, social and ecological systems. In the software engineering community, for example, maintainability has been a concern for a long time. But too often, these issues are treated in isolation from one another. Misperceptions among practitioners and research communities persist, rooted in a lack of coherent understanding of sustainability, and how it relates to software systems research and practice. This article presents a cross-disciplinary initiative to create a common ground and a point of reference for the global community of research and practice in software and sustainability, to be used for effectively communicating key issues, goals, values and principles of sustainability design for software-intensive systems. The centrepiece of this effort is the *Karlskrona Manifesto for Sustainability Design*, a vehicle for a much needed conversation about sustainability within and beyond the software community, and an articulation of the fundamental principles underpinning design choices that affect sustainability. We describe the motivation for developing this manifesto, including some considerations of the genre of the manifesto as well as the dynamics of its creation. We illustrate the collaborative reflective writing process and present the current edition of the manifesto itself. We assess immediate implications and applications of the articulated principles, compare these to current practice, and suggest future steps.**

I. Introduction

It is clear that society is facing major sustainability challenges that require long-term, joined-up thinking. How do we sustain our technical infrastructures, given how much we rely on them for everything from communication and navigation through to storing health records, identifying security threats, and keeping the lights on? How do we sustain prosperity in society, given the erosion of trust in our political institutions and a growing inequality in ownership of resources? And, above all, how do we sustain the planetary systems that support life on earth, in the face of accumulation of pollutants, species loss, and accelerating climate change?

The discipline of Software Engineering (SE) has a major role to play in sustainability, because of the extent to which software systems mediate so many aspects of our lives. However, software practice has a tendency to focus only on the immediate effects and tangible benefits of software products and platforms. SE research has, for the most part, focused on increasing the reliability, efficiency and cost-benefit relation of software products for their owners, through a focus on processes, methods, models and techniques to create, verify and validate software systems and keep them operational.

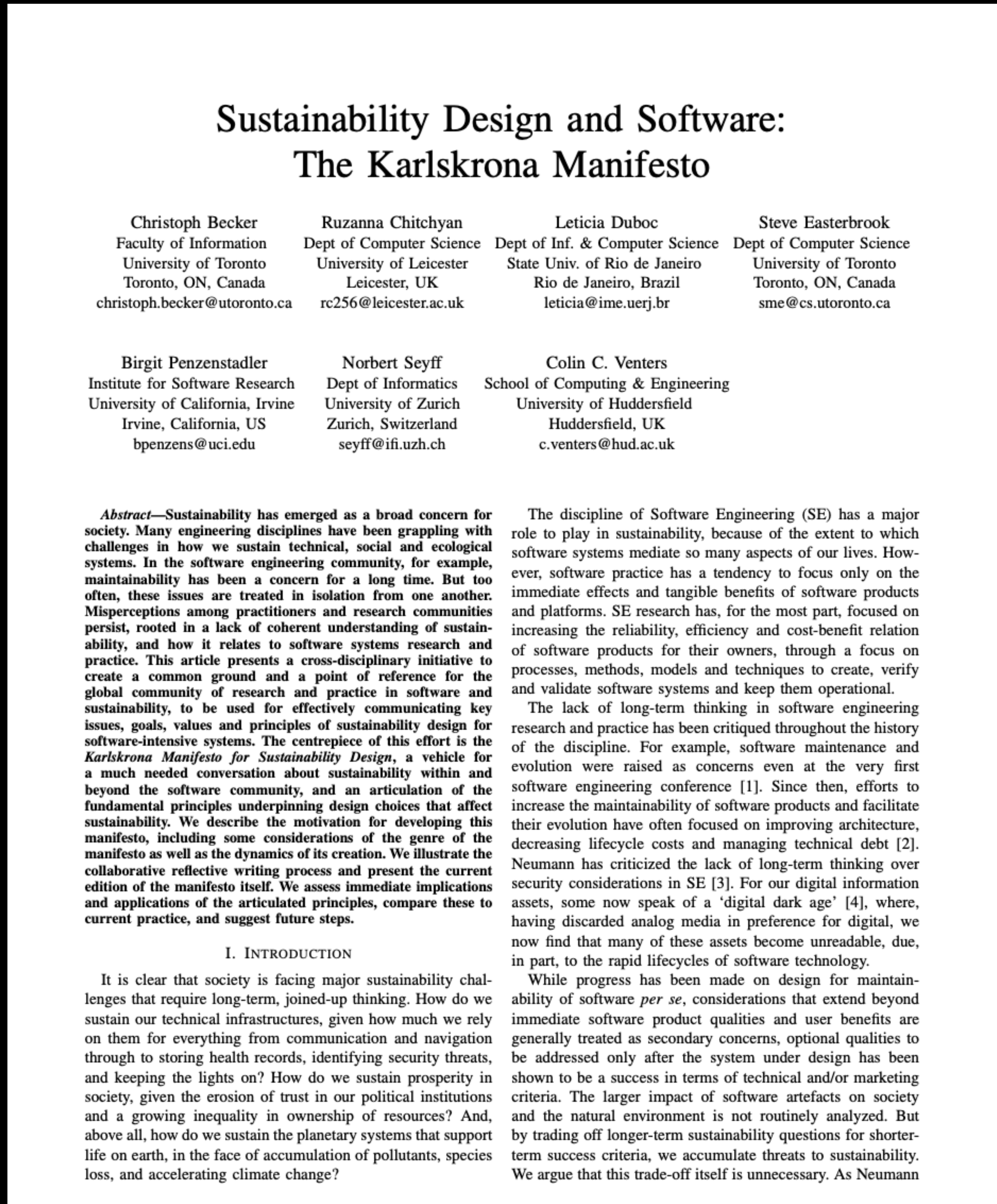
The lack of long-term thinking in software engineering research and practice has been critiqued throughout the history of the discipline. For example, software maintenance and evolution were raised as concerns even at the very first software engineering conference [1]. Since then, efforts to increase the maintainability of software products and facilitate their evolution have often focused on improving architecture, decreasing lifecycle costs and managing technical debt [2]. Neumann has criticized the lack of long-term thinking over security considerations in SE [3]. For our digital information assets, some now speak of a 'digital dark age' [4], where, having discarded analog media in preference for digital, we now find that many of these assets become unreadable, due, in part, to the rapid lifecycles of software technology.

While progress has been made on design for maintainability of software *per se*, considerations that extend beyond immediate software product qualities and user benefits are generally treated as secondary concerns, optional qualities to be addressed only after the system under design has been shown to be a success in terms of technical and/or marketing criteria. The larger impact of software artefacts on society and the natural environment is not routinely analyzed. But by trading off longer-term sustainability questions for shorter-term success criteria, we accumulate threats to sustainability. We argue that this trade-off itself is unnecessary. As Neumann

(Becker, 2015)

Software Sustainability

- Individual. **Mental and physical well-being, education, freedom, self-respect, mobility, agency.**
- Social. **Social equity, justice, employment, democracy.**
- Technical. **Maintenance, innovation, obsolescence, data integrity.**



Sustainability Design and Software: The Karlskrona Manifesto

Christoph Becker
Faculty of Information
University of Toronto
Toronto, ON, Canada
christoph.becker@utoronto.ca

Ruzanna Chitchyan
Dept of Computer Science
University of Leicester
Leicester, UK
rc256@leicester.ac.uk

Leticia Duboc
Dept of Inf. & Computer Science
State Univ. of Rio de Janeiro
Rio de Janeiro, Brazil
leticia@ime.uerj.br

Steve Easterbrook
Dept of Computer Science
University of Toronto
Toronto, ON, Canada
sme@cs.utoronto.ca

Birgit Penzenstadler
Institute for Software Research
University of California, Irvine
Irvine, California, US
bpenzens@uci.edu

Norbert Seyff
Dept of Informatics
University of Zurich
Zurich, Switzerland
seyff@ifi.uzh.ch

Colin C. Venters
School of Computing & Engineering
University of Huddersfield
Huddersfield, UK
c.venters@hud.ac.uk

***Abstract*—Sustainability has emerged as a broad concern for society. Many engineering disciplines have been grappling with challenges in how we sustain technical, social and ecological systems. In the software engineering community, for example, maintainability has been a concern for a long time. But too often, these issues are treated in isolation from one another. Misperceptions among practitioners and research communities persist, rooted in a lack of coherent understanding of sustainability, and how it relates to software systems research and practice. This article presents a cross-disciplinary initiative to create a common ground and a point of reference for the global community of research and practice in software and sustainability, to be used for effectively communicating key issues, goals, values and principles of sustainability design for software-intensive systems. The centrepiece of this effort is the *Karlskrona Manifesto for Sustainability Design*, a vehicle for a much needed conversation about sustainability within and beyond the software community, and an articulation of the fundamental principles underpinning design choices that affect sustainability. We describe the motivation for developing this manifesto, including some considerations of the genre of the manifesto as well as the dynamics of its creation. We illustrate the collaborative reflective writing process and present the current edition of the manifesto itself. We assess immediate implications and applications of the articulated principles, compare these to current practice, and suggest future steps.**

I. Introduction

It is clear that society is facing major sustainability challenges that require long-term, joined-up thinking. How do we sustain our technical infrastructures, given how much we rely on them for everything from communication and navigation through to storing health records, identifying security threats, and keeping the lights on? How do we sustain prosperity in society, given the erosion of trust in our political institutions and a growing inequality in ownership of resources? And, above all, how do we sustain the planetary systems that support life on earth, in the face of accumulation of pollutants, species loss, and accelerating climate change?

The discipline of Software Engineering (SE) has a major role to play in sustainability, because of the extent to which software systems mediate so many aspects of our lives. However, software practice has a tendency to focus only on the immediate effects and tangible benefits of software products and platforms. SE research has, for the most part, focused on increasing the reliability, efficiency and cost-benefit relation of software products for their owners, through a focus on processes, methods, models and techniques to create, verify and validate software systems and keep them operational.

The lack of long-term thinking in software engineering research and practice has been critiqued throughout the history of the discipline. For example, software maintenance and evolution were raised as concerns even at the very first software engineering conference [1]. Since then, efforts to increase the maintainability of software products and facilitate their evolution have often focused on improving architecture, decreasing lifecycle costs and managing technical debt [2]. Neumann has criticized the lack of long-term thinking over security considerations in SE [3]. For our digital information assets, some now speak of a 'digital dark age' [4], where, having discarded analog media in preference for digital, we now find that many of these assets become unreadable, due, in part, to the rapid lifecycles of software technology.

While progress has been made on design for maintainability of software *per se*, considerations that extend beyond immediate software product qualities and user benefits are generally treated as secondary concerns, optional qualities to be addressed only after the system under design has been shown to be a success in terms of technical and/or marketing criteria. The larger impact of software artefacts on society and the natural environment is not routinely analyzed. But by trading off longer-term sustainability questions for shorter-term success criteria, we accumulate threats to sustainability. We argue that this trade-off itself is unnecessary. As Neumann

(Becker, 2015)

Software Sustainability

- Individual. **Mental and physical well-being, education, freedom, self-respect, mobility, agency.**
- Social. **Social equity, justice, employment, democracy.**
- Technical. **Maintenance, innovation, obsolescence, data integrity.**
- Economic. **Wealth creation, prosperity, profitability, capital investment, income.**

Sustainability Design and Software: The Karlskrona Manifesto

Christoph Becker, Faculty of Information, University of Toronto, Toronto, ON, Canada, christoph.becker@utoronto.ca
Ruzanna Chitchyan, Dept of Computer Science, University of Leicester, Leicester, UK, rc256@leicester.ac.uk
Leticia Duboc, Dept of Inf. & Computer Science, State Univ. of Rio de Janeiro, Rio de Janeiro, Brazil, leticia@ime.uerj.br
Steve Easterbrook, Dept of Computer Science, University of Toronto, Toronto, ON, Canada, sme@cs.utoronto.ca
Birgit Penzenstadler, Institute for Software Research, University of California, Irvine, Irvine, California, US, bpenzens@uci.edu
Norbert Seyff, Dept of Informatics, University of Zurich, Zurich, Switzerland, seyff@ifi.uzh.ch
Colin C. Venters, School of Computing & Engineering, University of Huddersfield, Huddersfield, UK, c.venters@hud.ac.uk

Abstract—**Sustainability has emerged as a broad concern for society. Many engineering disciplines have been grappling with challenges in how we sustain technical, social and ecological systems. In the software engineering community, for example, maintainability has been a concern for a long time. But too often, these issues are treated in isolation from one another. Misperceptions among practitioners and research communities persist, rooted in a lack of coherent understanding of sustainability, and how it relates to software systems research and practice. This article presents a cross-disciplinary initiative to create a common ground and a point of reference for the global community of research and practice in software and sustainability, to be used for effectively communicating key issues, goals, values and principles of sustainability design for software-intensive systems. The centrepiece of this effort is the *Karlskrona Manifesto for Sustainability Design*, a vehicle for a much needed conversation about sustainability within and beyond the software community, and an articulation of the fundamental principles underpinning design choices that affect sustainability. We describe the motivation for developing this manifesto, including some considerations of the genre of the manifesto as well as the dynamics of its creation. We illustrate the collaborative reflective writing process and present the current edition of the manifesto itself. We assess immediate implications and applications of the articulated principles, compare these to current practice, and suggest future steps.**

I. Introduction

It is clear that society is facing major sustainability challenges that require long-term, joined-up thinking. How do we sustain our technical infrastructures, given how much we rely on them for everything from communication and navigation through to storing health records, identifying security threats, and keeping the lights on? How do we sustain prosperity in society, given the erosion of trust in our political institutions and a growing inequality in ownership of resources? And, above all, how do we sustain the planetary systems that support life on earth, in the face of accumulation of pollutants, species loss, and accelerating climate change?

The discipline of Software Engineering (SE) has a major role to play in sustainability, because of the extent to which software systems mediate so many aspects of our lives. However, software practice has a tendency to focus only on the immediate effects and tangible benefits of software products and platforms. SE research has, for the most part, focused on increasing the reliability, efficiency and cost-benefit relation of software products for their owners, through a focus on processes, methods, models and techniques to create, verify and validate software systems and keep them operational.

The lack of long-term thinking in software engineering research and practice has been critiqued throughout the history of the discipline. For example, software maintenance and evolution were raised as concerns even at the very first software engineering conference [1]. Since then, efforts to increase the maintainability of software products and facilitate their evolution have often focused on improving architecture, decreasing lifecycle costs and managing technical debt [2]. Neumann has criticized the lack of long-term thinking over security considerations in SE [3]. For our digital information assets, some now speak of a ‘digital dark age’ [4], where, having discarded analog media in preference for digital, we now find that many of these assets become unreadable, due, in part, to the rapid lifecycles of software technology.

While progress has been made on design for maintainability of software *per se*, considerations that extend beyond immediate software product qualities and user benefits are generally treated as secondary concerns, optional qualities to be addressed only after the system under design has been shown to be a success in terms of technical and/or marketing criteria. The larger impact of software artefacts on society and the natural environment is not routinely analyzed. But by trading off longer-term sustainability questions for shorter-term success criteria, we accumulate threats to sustainability. We argue that this trade-off itself is unnecessary. As Neumann

(Becker, 2015)

Software Sustainability

- Individual. **Mental and physical well-being, education, freedom, self-respect, mobility, agency.**
- Social. **Social equity, justice, employment, democracy.**
- Technical. **Maintenance, innovation, obsolescence, data integrity.**
- Economic. **Wealth creation, prosperity, profitability, capital investment, income.**
- Environmental. **Ecosystems, raw resources, climate change, food production, water, pollution, waste.**

Sustainability Design and Software: The Karlskrona Manifesto

Christoph Becker, Faculty of Information, University of Toronto, Toronto, ON, Canada, christoph.becker@utoronto.ca

Ruzanna Chitchyan, Dept of Computer Science, University of Leicester, Leicester, UK, rc256@leicester.ac.uk

Leticia Duboc, Dept of Inf. & Computer Science, State Univ. of Rio de Janeiro, Rio de Janeiro, Brazil, leticia@ime.uerj.br

Steve Easterbrook, Dept of Computer Science, University of Toronto, Toronto, ON, Canada, sme@cs.utoronto.ca

Birgit Penzenstadler, Institute for Software Research, University of California, Irvine, Irvine, California, US, bpenzens@uci.edu

Norbert Seyff, Dept of Informatics, University of Zurich, Zurich, Switzerland, seyff@ifi.uzh.ch

Colin C. Venters, School of Computing & Engineering, University of Huddersfield, Huddersfield, UK, c.venters@hud.ac.uk

***Abstract*—Sustainability has emerged as a broad concern for society. Many engineering disciplines have been grappling with challenges in how we sustain technical, social and ecological systems. In the software engineering community, for example, maintainability has been a concern for a long time. But too often, these issues are treated in isolation from one another. Misperceptions among practitioners and research communities persist, rooted in a lack of coherent understanding of sustainability, and how it relates to software systems research and practice. This article presents a cross-disciplinary initiative to create a common ground and a point of reference for the global community of research and practice in software and sustainability, to be used for effectively communicating key issues, goals, values and principles of sustainability design for software-intensive systems. The centrepiece of this effort is the *Karlskrona Manifesto for Sustainability Design*, a vehicle for a much needed conversation about sustainability within and beyond the software community, and an articulation of the fundamental principles underpinning design choices that affect sustainability. We describe the motivation for developing this manifesto, including some considerations of the genre of the manifesto as well as the dynamics of its creation. We illustrate the collaborative reflective writing process and present the current edition of the manifesto itself. We assess immediate implications and applications of the articulated principles, compare these to current practice, and suggest future steps.**

I. Introduction

It is clear that society is facing major sustainability challenges that require long-term, joined-up thinking. How do we sustain our technical infrastructures, given how much we rely on them for everything from communication and navigation through to storing health records, identifying security threats, and keeping the lights on? How do we sustain prosperity in society, given the erosion of trust in our political institutions and a growing inequality in ownership of resources? And, above all, how do we sustain the planetary systems that support life on earth, in the face of accumulation of pollutants, species loss, and accelerating climate change?

The discipline of Software Engineering (SE) has a major role to play in sustainability, because of the extent to which software systems mediate so many aspects of our lives. However, software practice has a tendency to focus only on the immediate effects and tangible benefits of software products and platforms. SE research has, for the most part, focused on increasing the reliability, efficiency and cost-benefit relation of software products for their owners, through a focus on processes, methods, models and techniques to create, verify and validate software systems and keep them operational.

The lack of long-term thinking in software engineering research and practice has been critiqued throughout the history of the discipline. For example, software maintenance and evolution were raised as concerns even at the very first software engineering conference [1]. Since then, efforts to increase the maintainability of software products and facilitate their evolution have often focused on improving architecture, decreasing lifecycle costs and managing technical debt [2]. Neumann has criticized the lack of long-term thinking over security considerations in SE [3]. For our digital information assets, some now speak of a 'digital dark age' [4], where, having discarded analog media in preference for digital, we now find that many of these assets become unreadable, due, in part, to the rapid lifecycles of software technology.

While progress has been made on design for maintainability of software *per se*, considerations that extend beyond immediate software product qualities and user benefits are generally treated as secondary concerns, optional qualities to be addressed only after the system under design has been shown to be a success in terms of technical and/or marketing criteria. The larger impact of software artefacts on society and the natural environment is not routinely analyzed. But by trading off longer-term sustainability questions for shorter-term success criteria, we accumulate threats to sustainability. We argue that this trade-off itself is unnecessary. As Neumann

(Becker, 2015)

Software Sustainability

- Individual. **Mental and physical well-being, education, freedom, self-respect, mobility, agency.**
- Social. **Social equity, justice, employment, democracy.**
- Technical. **Maintenance, innovation, obsolescence, data integrity.**
- Economic. **Wealth creation, prosperity, profitability, capital investment, income.**
- Environmental. **Ecosystems, raw resources, climate change, food production, water, pollution, waste.**

Sustainability Design and Software: The Karlskrona Manifesto

Christoph Becker, Faculty of Information, University of Toronto, Toronto, ON, Canada, christoph.becker@utoronto.ca

Ruzanna Chitchyan, Dept of Computer Science, University of Leicester, Leicester, UK, rc256@leicester.ac.uk

Leticia Duboc, Dept of Inf. & Computer Science, State Univ. of Rio de Janeiro, Rio de Janeiro, Brazil, leticia@ime.uerj.br

Steve Easterbrook, Dept of Computer Science, University of Toronto, Toronto, ON, Canada, sme@cs.utoronto.ca

Birgit Penzenstadler, Institute for Software Research, University of California, Irvine, Irvine, California, US, bpenzens@uci.edu

Norbert Seyff, Dept of Informatics, University of Zurich, Zurich, Switzerland, seyff@ifi.uzh.ch

Colin C. Venters, School of Computing & Engineering, University of Huddersfield, Huddersfield, UK, c.venters@hud.ac.uk

***Abstract*—Sustainability has emerged as a broad concern for society. Many engineering disciplines have been grappling with challenges in how we sustain technical, social and ecological systems. In the software engineering community, for example, maintainability has been a concern for a long time. But too often, these issues are treated in isolation from one another. Misperceptions among practitioners and research communities persist, rooted in a lack of coherent understanding of sustainability, and how it relates to software systems research and practice. This article presents a cross-disciplinary initiative to create a common ground and a point of reference for the global community of research and practice in software and sustainability, to be used for effectively communicating key issues, goals, values and principles of sustainability design for software-intensive systems. The centrepiece of this effort is the *Karlskrona Manifesto for Sustainability Design*, a vehicle for a much needed conversation about sustainability within and beyond the software community, and an articulation of the fundamental principles underpinning design choices that affect sustainability. We describe the motivation for developing this manifesto, including some considerations of the genre of the manifesto as well as the dynamics of its creation. We illustrate the collaborative reflective writing process and present the current edition of the manifesto itself. We assess immediate implications and applications of the articulated principles, compare these to current practice, and suggest future steps.**

I. Introduction

It is clear that society is facing major sustainability challenges that require long-term, joined-up thinking. How do we sustain our technical infrastructures, given how much we rely on them for everything from communication and navigation through to storing health records, identifying security threats, and keeping the lights on? How do we sustain prosperity in society, given the erosion of trust in our political institutions and a growing inequality in ownership of resources? And, above all, how do we sustain the planetary systems that support life on earth, in the face of accumulation of pollutants, species loss, and accelerating climate change?

The discipline of Software Engineering (SE) has a major role to play in sustainability, because of the extent to which software systems mediate so many aspects of our lives. However, software practice has a tendency to focus only on the immediate effects and tangible benefits of software products and platforms. SE research has, for the most part, focused on increasing the reliability, efficiency and cost-benefit relation of software products for their owners, through a focus on processes, methods, models and techniques to create, verify and validate software systems and keep them operational.

The lack of long-term thinking in software engineering research and practice has been critiqued throughout the history of the discipline. For example, software maintenance and evolution were raised as concerns even at the very first software engineering conference [1]. Since then, efforts to increase the maintainability of software products and facilitate their evolution have often focused on improving architecture, decreasing lifecycle costs and managing technical debt [2]. Neumann has criticized the lack of long-term thinking over security considerations in SE [3]. For our digital information assets, some now speak of a 'digital dark age' [4], where, having discarded analog media in preference for digital, we now find that many of these assets become unreadable, due, in part, to the rapid lifecycles of software technology.

While progress has been made on design for maintainability of software *per se*, considerations that extend beyond immediate software product qualities and user benefits are generally treated as secondary concerns, optional qualities to be addressed only after the system under design has been shown to be a success in terms of technical and/or marketing criteria. The larger impact of software artefacts on society and the natural environment is not routinely analyzed. But by trading off longer-term sustainability questions for shorter-term success criteria, we accumulate threats to sustainability. We argue that this trade-off itself is unnecessary. As Neumann

(Becker, 2015)

Individual Sustainability

“How Was Your Weekend?” Software Development Teams Working From Home During COVID-19

Courtney Miller*, Paige Rodeghero‡, Margaret-Anne Storey§, Denae Ford¶ and Thomas Zimmermann‖

* New College of Florida, FL, USA. Email: courtney.miller17@ncf.edu
‡ Clemson University, SC, USA. Email: prodegh@clemson.edu
§ University of Victoria, BC, Canada. Email: mstorey@uvic.ca
¶ Microsoft Research, WA, USA. Email: denae@microsoft.com
‖ Microsoft Research, WA, USA. Email: tzimmer@microsoft.com

arXiv:2101.05877v2 [cs.SE] 11 Feb 2021

***Abstract*—The mass shift to working at home during the COVID-19 pandemic radically changed the way many software development teams collaborate and communicate. To investigate how team culture and team productivity may also have been affected, we conducted two surveys at a large software company. The first, an exploratory survey during the early months of the pandemic with 2,265 developer responses, revealed that many developers faced challenges reaching milestones and that their team productivity had changed. We also found through qualitative analysis that important team culture factors such as communication and social connection had been affected. For example, the simple phrase “How was your weekend?” had become a subtle way to show peer support.**

In our second survey, we conducted a quantitative analysis of the team cultural factors that emerged from our first survey to understand the prevalence of the reported changes. From 608 developer responses, we found that 74% of these respondents missed social interactions with colleagues and 51% reported a decrease in their communication ease with colleagues. We used data from the second survey to build a regression model to identify important team culture factors for modeling team productivity. We found that the ability to brainstorm with colleagues, difficulty communicating with colleagues, and satisfaction with interactions from social activities are important factors that are associated with how developers report their software development team's productivity. Our findings inform how managers and leaders in large software companies can support sustained team productivity during times of crisis and beyond.

I. Introduction

As COVID-19 spread globally, many companies, including Google, Microsoft, Twitter, Amazon, and Facebook, instructed their software developers to go home and work remotely [1]. Entire software development teams that used to work predominantly in-person suddenly had to pivot their work and quickly establish effective remote collaboration and communication.

Prior research has studied how working from home (WFH) affects productivity [2], [3]. While regular WFH is not the same as WFH during a pandemic, the COVID-19 pandemic has created a natural experiment for researchers to study

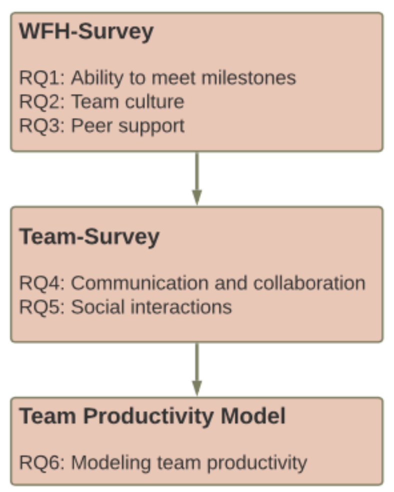


Fig. 1. Methodology Flow Chart

are recent papers that already began investigating the impacts of this unique work setting. Bao *et al.* performed a case study using automated trace data, along with other metrics, to determine how productivity has been affected. They found that productivity was affected in various ways depending on the productivity metrics used [4]. Ralph *et al.* performed an international large-scale questionnaire survey of developer well-being and productivity and found that productivity and well-being are closely related, and both are currently suffering [5]. As insightful as these works are at providing empirical evidence of factors affecting individual developer productivity, they lack a deeper understanding of a major responsibility of industrial software developers—collaborating with a team.

In our work, we identify factors that affect software development team productivity such as team culture factors, including communication, camaraderie, and team cohesion [6], [7]. We hypothesize these factors are at particular risk of being disrupted by this unexpected shift to WFH. Thus, we

(Miller et al., 2021)
Preprint: https://arxiv.org/pdf/2101.05877.pdf

Individual Sustainability

- Survey with 600+ developers

“How Was Your Weekend?” Software Development Teams Working From Home During COVID-19

Courtney Miller*, Paige Rodeghero‡, Margaret-Anne Storey§, Denae Ford¶ and Thomas Zimmermann‖

* New College of Florida, FL, USA. Email: courtney.miller17@ncf.edu
‡ Clemson University, SC, USA. Email: prodegh@clemson.edu
§ University of Victoria, BC, Canada. Email: mstorey@uvic.ca
¶ Microsoft Research, WA, USA. Email: denae@microsoft.com
‖ Microsoft Research, WA, USA. Email: tzimmer@microsoft.com

arXiv:2101.05877v2 [cs.SE] 11 Feb 2021

***Abstract*—The mass shift to working at home during the COVID-19 pandemic radically changed the way many software development teams collaborate and communicate. To investigate how team culture and team productivity may also have been affected, we conducted two surveys at a large software company. The first, an exploratory survey during the early months of the pandemic with 2,265 developer responses, revealed that many developers faced challenges reaching milestones and that their team productivity had changed. We also found through qualitative analysis that important team culture factors such as communication and social connection had been affected. For example, the simple phrase “How was your weekend?” had become a subtle way to show peer support.**

In our second survey, we conducted a quantitative analysis of the team cultural factors that emerged from our first survey to understand the prevalence of the reported changes. From 608 developer responses, we found that 74% of these respondents missed social interactions with colleagues and 51% reported a decrease in their communication ease with colleagues. We used data from the second survey to build a regression model to identify important team culture factors for modeling team productivity. We found that the ability to brainstorm with colleagues, difficulty communicating with colleagues, and satisfaction with interactions from social activities are important factors that are associated with how developers report their software development team’s productivity. Our findings inform how managers and leaders in large software companies can support sustained team productivity during times of crisis and beyond.

I. Introduction

As COVID-19 spread globally, many companies, including Google, Microsoft, Twitter, Amazon, and Facebook, instructed their software developers to go home and work remotely [1]. Entire software development teams that used to work predominantly in-person suddenly had to pivot their work and quickly establish effective remote collaboration and communication.

Prior research has studied how working from home (WFH) affects productivity [2], [3]. While regular WFH is not the same as WFH during a pandemic, the COVID-19 pandemic has created a natural experiment for researchers to study

Fig. 1. Methodology Flow Chart

are recent papers that already began investigating the impacts of this unique work setting. Bao *et al.* performed a case study using automated trace data, along with other metrics, to determine how productivity has been affected. They found that productivity was affected in various ways depending on the productivity metrics used [4]. Ralph *et al.* performed an international large-scale questionnaire survey of developer well-being and productivity and found that productivity and well-being are closely related, and both are currently suffering [5]. As insightful as these works are at providing empirical evidence of factors affecting individual developer productivity, they lack a deeper understanding of a major responsibility of industrial software developers—collaborating with a team.

In our work, we identify factors that affect software development team productivity such as team culture factors, including communication, camaraderie, and team cohesion [6], [7]. We hypothesize these factors are at particular risk of being disrupted by this unexpected shift to WFH. Thus, we

(Miller et al., 2021)
Preprint: https://arxiv.org/pdf/2101.05877.pdf

Individual Sustainability

- Survey with 600+ developers
- Pandemic remote work isn't remote work.

arXiv:2101.05877v2 [cs.SE] 11 Feb 2021

“How Was Your Weekend?” Software Development Teams Working From Home During COVID-19

Courtney Miller*, Paige Rodeghero‡, Margaret-Anne Storey§, Denae Ford¶ and Thomas Zimmermann‖

* New College of Florida, FL, USA. Email: courtney.miller17@ncf.edu
‡ Clemson University, SC, USA. Email: prodegh@clemson.edu
§ University of Victoria, BC, Canada. Email: mstorey@uvic.ca
¶ Microsoft Research, WA, USA. Email: denae@microsoft.com
‖ Microsoft Research, WA, USA. Email: tzimmer@microsoft.com

***Abstract*—The mass shift to working at home during the COVID-19 pandemic radically changed the way many software development teams collaborate and communicate. To investigate how team culture and team productivity may also have been affected, we conducted two surveys at a large software company. The first, an exploratory survey during the early months of the pandemic with 2,265 developer responses, revealed that many developers faced challenges reaching milestones and that their team productivity had changed. We also found through qualitative analysis that important team culture factors such as communication and social connection had been affected. For example, the simple phrase “How was your weekend?” had become a subtle way to show peer support.**

In our second survey, we conducted a quantitative analysis of the team cultural factors that emerged from our first survey to understand the prevalence of the reported changes. From 608 developer responses, we found that 74% of these respondents missed social interactions with colleagues and 51% reported a decrease in their communication ease with colleagues. We used data from the second survey to build a regression model to identify important team culture factors for modeling team productivity. We found that the ability to brainstorm with colleagues, difficulty communicating with colleagues, and satisfaction with interactions from social activities are important factors that are associated with how developers report their software development team's productivity. Our findings inform how managers and leaders in large software companies can support sustained team productivity during times of crisis and beyond.

Fig. 1. Methodology Flow Chart

I. INTRODUCTION

As COVID-19 spread globally, many companies, including Google, Microsoft, Twitter, Amazon, and Facebook, instructed their software developers to go home and work remotely [1]. Entire software development teams that used to work predominantly in-person suddenly had to pivot their work and quickly establish effective remote collaboration and communication.

Prior research has studied how working from home (WFH) affects productivity [2], [3]. While regular WFH is not the same as WFH during a pandemic, the COVID-19 pandemic has created a natural experiment for researchers to study WFH on a much larger scale than previously possible. For

are recent papers that already began investigating the impacts of this unique work setting. Bao *et al.* performed a case study using automated trace data, along with other metrics, to determine how productivity has been affected. They found that productivity was affected in various ways depending on the productivity metrics used [4]. Ralph *et al.* performed an international large-scale questionnaire survey of developer well-being and productivity and found that productivity and well-being are closely related, and both are currently suffering [5]. As insightful as these works are at providing empirical evidence of factors affecting individual developer productivity, they lack a deeper understanding of a major responsibility of industrial software developers—collaborating with a team.

In our work, we identify factors that affect software development team productivity such as team culture factors, including communication, camaraderie, and team cohesion [6], [7]. We hypothesize these factors are at particular risk of being disrupted by this unexpected shift to WFH. Thus, we investigated the effects of WFH on teams and answered the

(Miller et al., 2021)
Preprint: https://arxiv.org/pdf/2101.05877.pdf

Individual Sustainability

- Survey with 600+ developers
- Pandemic remote work isn't remote work.
- *"We have lost somewhere between 20%-40% effectiveness in use of time. In order to keep up, people are working longer hours. We are starting to see burnout."* (participant 1384)

"How Was Your Weekend?" Software Development Teams Working From Home During COVID-19

Courtney Miller*, Paige Rodeghero‡, Margaret-Anne Storey§, Denae Ford¶ and Thomas Zimmermann‖

* New College of Florida, FL, USA. Email: courtney.miller17@ncf.edu
‡ Clemson University, SC, USA. Email: prodegh@clemson.edu
§ University of Victoria, BC, Canada. Email: mstorey@uvic.ca
¶ Microsoft Research, WA, USA. Email: denae@microsoft.com
‖ Microsoft Research, WA, USA. Email: tzimmer@microsoft.com

arXiv:2101.05877v2 [cs.SE] 11 Feb 2021

***Abstract*—The mass shift to working at home during the COVID-19 pandemic radically changed the way many software development teams collaborate and communicate. To investigate how team culture and team productivity may also have been affected, we conducted two surveys at a large software company. The first, an exploratory survey during the early months of the pandemic with 2,265 developer responses, revealed that many developers faced challenges reaching milestones and that their team productivity had changed. We also found through qualitative analysis that important team culture factors such as communication and social connection had been affected. For example, the simple phrase "How was your weekend?" had become a subtle way to show peer support.**

In our second survey, we conducted a quantitative analysis of the team cultural factors that emerged from our first survey to understand the prevalence of the reported changes. From 608 developer responses, we found that 74% of these respondents missed social interactions with colleagues and 51% reported a decrease in their communication ease with colleagues. We used data from the second survey to build a regression model to identify important team culture factors for modeling team productivity. We found that the ability to brainstorm with colleagues, difficulty communicating with colleagues, and satisfaction with interactions from social activities are important factors that are associated with how developers report their software development team's productivity. Our findings inform how managers and leaders in large software companies can support sustained team productivity during times of crisis and beyond.

Fig. 1. Methodology Flow Chart

I. Introduction

As COVID-19 spread globally, many companies, including Google, Microsoft, Twitter, Amazon, and Facebook, instructed their software developers to go home and work remotely [1]. Entire software development teams that used to work predominantly in-person suddenly had to pivot their work and quickly establish effective remote collaboration and communication.

Prior research has studied how working from home (WFH) affects productivity [2], [3]. While regular WFH is not the same as WFH during a pandemic, the COVID-19 pandemic has created a natural experiment for researchers to study WFH on a much larger scale than previously possible. For

are recent papers that already began investigating the impacts of this unique work setting. Bao *et al.* performed a case study using automated trace data, along with other metrics, to determine how productivity has been affected. They found that productivity was affected in various ways depending on the productivity metrics used [4]. Ralph *et al.* performed an international large-scale questionnaire survey of developer well-being and productivity and found that productivity and well-being are closely related, and both are currently suffering [5]. As insightful as these works are at providing empirical evidence of factors affecting individual developer productivity, they lack a deeper understanding of a major responsibility of industrial software developers—collaborating with a team.

In our work, we identify factors that affect software development team productivity such as team culture factors, including communication, camaraderie, and team cohesion [6], [7]. We hypothesize these factors are at particular risk of being disrupted by this unexpected shift to WFH. Thus, we investigated the effects of WFH on teams and answered the

(Miller et al., 2021)
Preprint: https://arxiv.org/pdf/2101.05877.pdf

Individual Sustainability

- Survey with 600+ developers
- Pandemic remote work isn't remote work.
- *"We have lost somewhere between 20%-40% effectiveness in use of time. In order to keep up, people are working longer hours. We are starting to see burnout."* (participant 1384)
- **A few proposed practices:**
  Build and maintain team culture.
  Include social activities as part of "work."
  Be mindful of other people's time.
  Actively work to be inclusive.

arXiv:2101.05877v2 [cs.SE] 11 Feb 2021

"How Was Your Weekend?" Software Development Teams Working From Home During COVID-19

Courtney Miller*, Paige Rodeghero‡, Margaret-Anne Storey§, Denae Ford¶ and Thomas Zimmermann‖
* New College of Florida, FL, USA. Email: courtney.miller17@ncf.edu
‡ Clemson University, SC, USA. Email: prodegh@clemson.edu
§ University of Victoria, BC, Canada. Email: mstorey@uvic.ca
¶ Microsoft Research, WA, USA. Email: denae@microsoft.com
‖ Microsoft Research, WA, USA. Email: tzimmer@microsoft.com

***Abstract*—The mass shift to working at home during the COVID-19 pandemic radically changed the way many software development teams collaborate and communicate. To investigate how team culture and team productivity may also have been affected, we conducted two surveys at a large software company. The first, an exploratory survey during the early months of the pandemic with 2,265 developer responses, revealed that many developers faced challenges reaching milestones and that their team productivity had changed. We also found through qualitative analysis that important team culture factors such as communication and social connection had been affected. For example, the simple phrase "How was your weekend?" had become a subtle way to show peer support.**

In our second survey, we conducted a quantitative analysis of the team cultural factors that emerged from our first survey to understand the prevalence of the reported changes. From 608 developer responses, we found that 74% of these respondents missed social interactions with colleagues and 51% reported a decrease in their communication ease with colleagues. We used data from the second survey to build a regression model to identify important team culture factors for modeling team productivity. We found that the ability to brainstorm with colleagues, difficulty communicating with colleagues, and satisfaction with interactions from social activities are important factors that are associated with how developers report their software development team's productivity. Our findings inform how managers and leaders in large software companies can support sustained team productivity during times of crisis and beyond.

Fig. 1. Methodology Flow Chart

I. INTRODUCTION

As COVID-19 spread globally, many companies, including Google, Microsoft, Twitter, Amazon, and Facebook, instructed their software developers to go home and work remotely [1]. Entire software development teams that used to work predominantly in-person suddenly had to pivot their work and quickly establish effective remote collaboration and communication.

Prior research has studied how working from home (WFH) affects productivity [2], [3]. While regular WFH is not the same as WFH during a pandemic, the COVID-19 pandemic has created a natural experiment for researchers to study WFH on a much larger scale than previously possible. For

are recent papers that already began investigating the impacts of this unique work setting. Bao *et al.* performed a case study using automated trace data, along with other metrics, to determine how productivity has been affected. They found that productivity was affected in various ways depending on the productivity metrics used [4]. Ralph *et al.* performed an international large-scale questionnaire survey of developer well-being and productivity and found that productivity and well-being are closely related, and both are currently suffering [5]. As insightful as these works are at providing empirical evidence of factors affecting individual developer productivity, they lack a deeper understanding of a major responsibility of industrial software developers—collaborating with a team.

In our work, we identify factors that affect software development team productivity such as team culture factors, including communication, camaraderie, and team cohesion [6], [7]. We hypothesize these factors are at particular risk of being disrupted by this unexpected shift to WFH. Thus, we investigated the effects of WFH on teams and answered the

(Miller et al., 2021)
Preprint: https://arxiv.org/pdf/2101.05877.pdf

Green Software Engineering

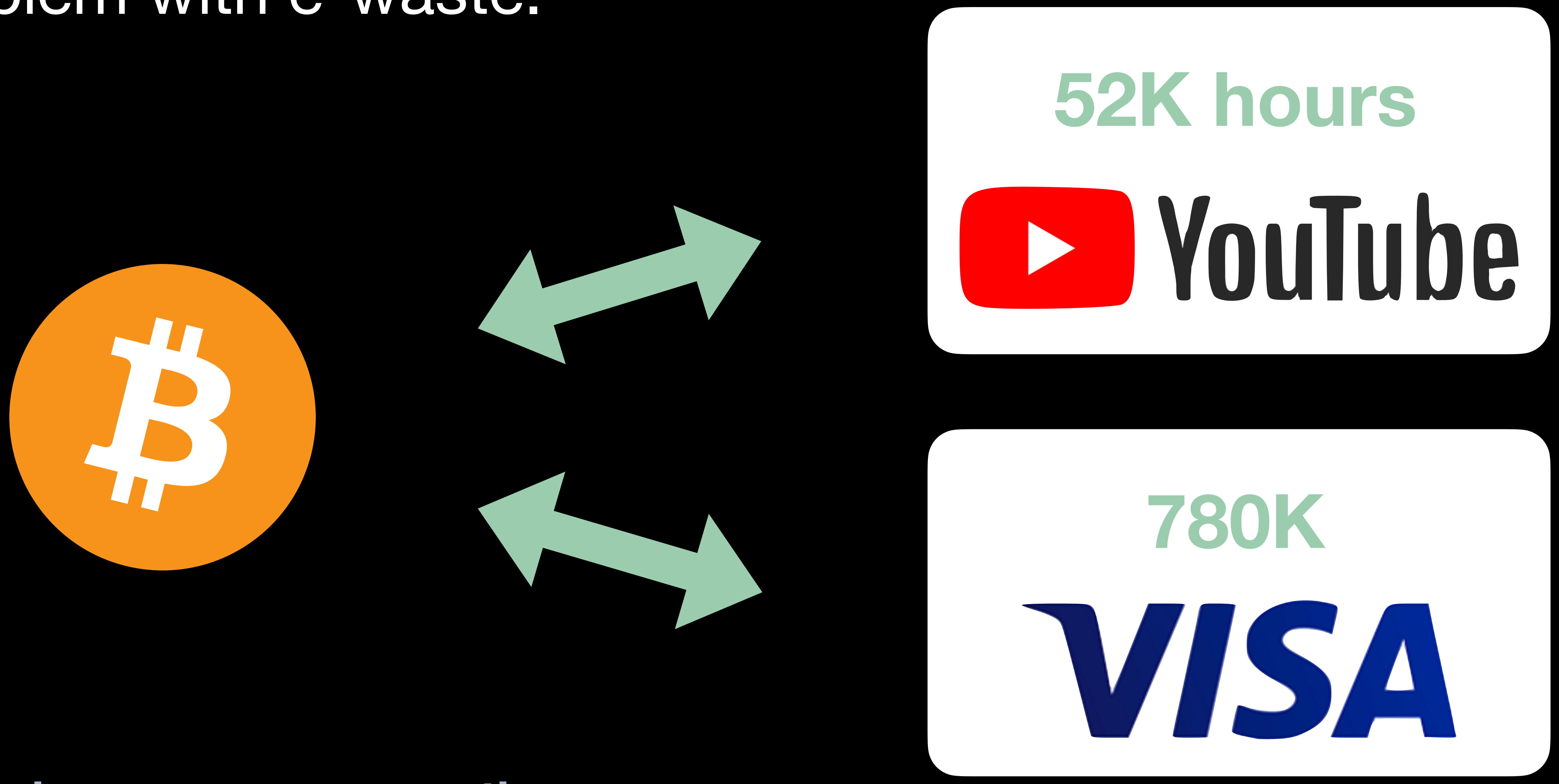
- What is it?

Bitcoin example

- No central authority — consensus algorithm.
- One transaction — multiple agents to compute a hash that validates the transaction.
- Several discussions regarding social sustainability.
- **How does bitcoin transactions compare to traditional centralised transactions w.r.t. environmental impact?**

Bitcoin example

- Annual energy consumption equivalent to Chile’s (77.78 TWh).
- Problem with e-waste.



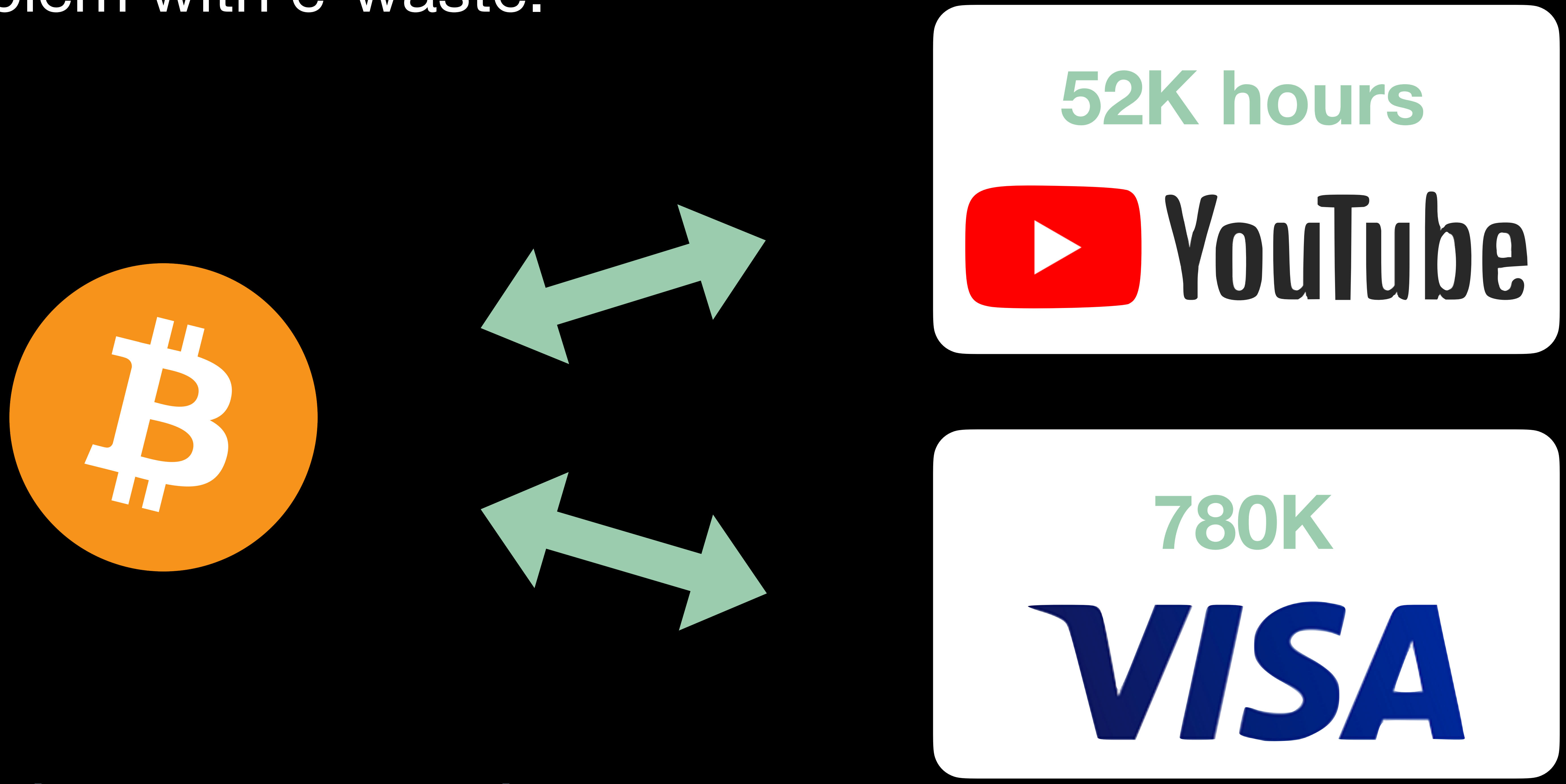
https://digiconomist.net/bitcoin-energy-consumption

https://cbeci.org/cbeci/comparisons

Bitcoin example

higher than the Netherlands' (120TWh💰 > 111TWh🇳🇱)

- Annual energy consumption ~~equivalent to Chile's (77.78 TWh)~~.
- Problem with e-waste.



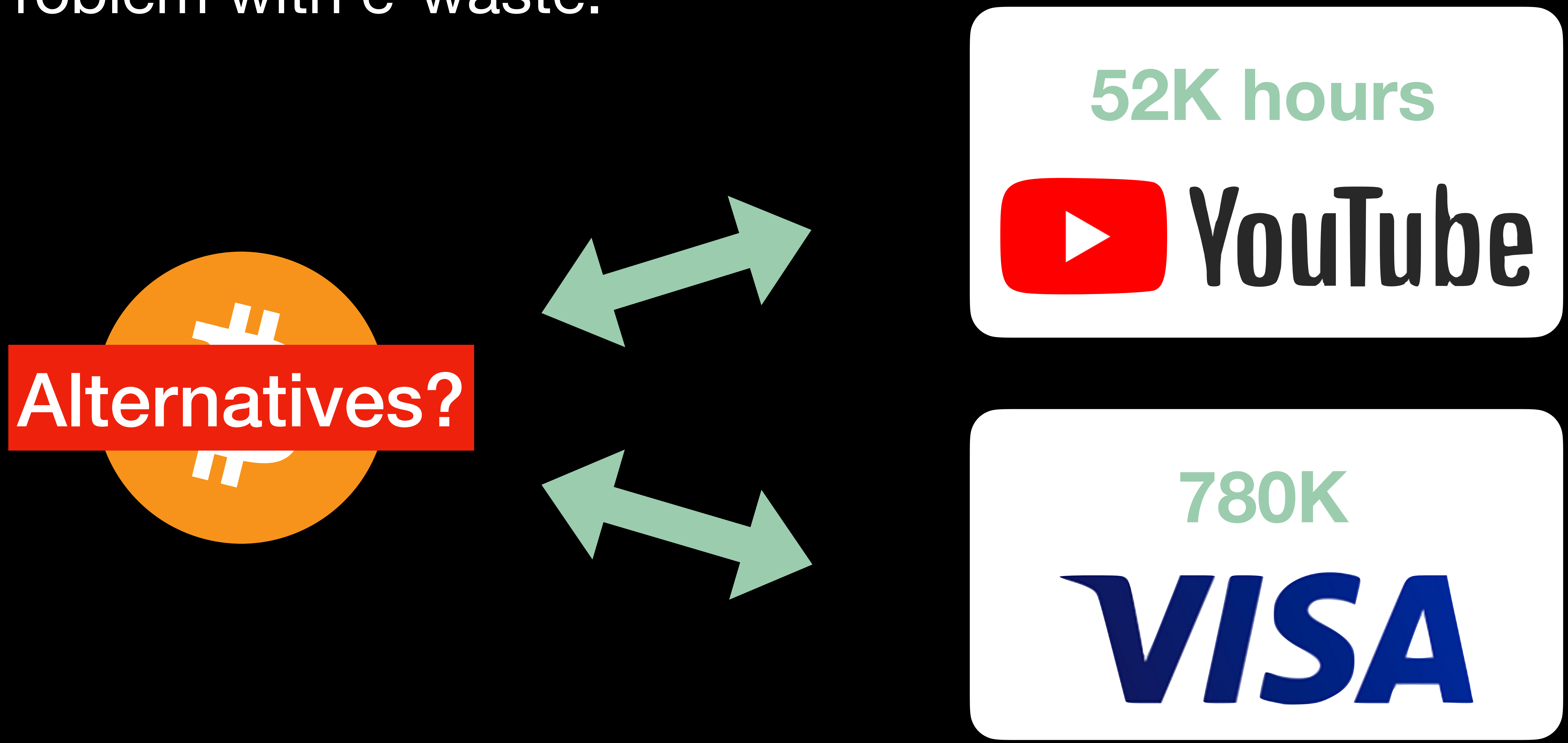
https://digiconomist.net/bitcoin-energy-consumption

https://cbeci.org/cbeci/comparisons

Bitcoin example

higher than the Netherlands' (120TWh💰 > 111TWh🇳🇱)

- Annual energy consumption ~~equivalent to Chile's (77.78 TWh)~~.
- Problem with e-waste.



https://digiconomist.net/bitcoin-energy-consumption

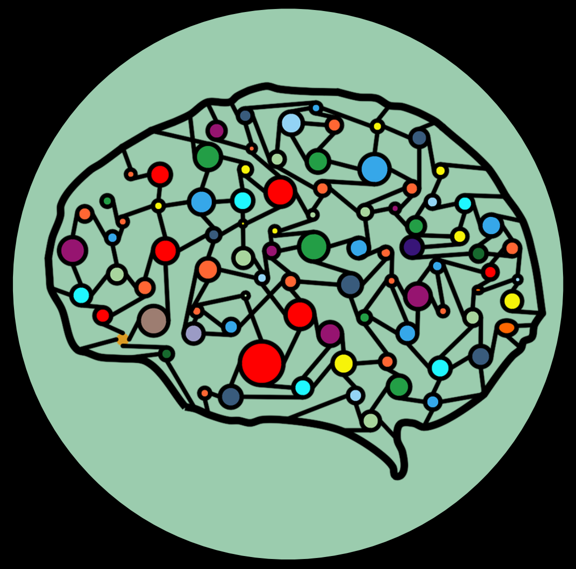
https://cbeci.org/cbeci/comparisons

Training Neural Networks

- Deep Learning in NLP.

- Training and tuning an NLP model is comparable to the **CO2 emission** of a **normal car** throughout its lifetime.

- Researchers should prioritize developing efficient models and hardware.



(Strubell, 2019)

Energy and Policy Considerations for Deep Learning in NLP

Emma Strubell **Ananya Ganesh** **Andrew McCallum**
College of Information and Computer Sciences
University of Massachusetts Amherst
{strubell, aganesh, mccallum}@cs.umass.edu

Abstract

Recent progress in hardware and methodology for training neural networks has ushered in a new generation of large networks trained on abundant data. These models have obtained notable gains in accuracy across many NLP tasks. However, these accuracy improvements depend on the availability of exceptionally large computational resources that necessitate similarly substantial energy consumption. As a result these models are costly to train and develop, both financially, due to the cost of hardware and electricity or cloud compute time, and environmentally, due to the carbon footprint required to fuel modern tensor processing hardware. In this paper we bring this issue to the attention of NLP researchers by quantifying the approximate financial and environmental costs of training a variety of recently successful neural network models for NLP. Based on these findings, we propose actionable recommendations to reduce costs and improve equity in NLP research and practice.

1 Introduction

Advances in techniques and hardware for training deep neural networks have recently enabled impressive accuracy improvements across many fundamental NLP tasks (Bahdanau et al., 2015; Luong et al., 2015; Dozat and Manning, 2017; Vaswani et al., 2017), with the most computationally-hungry models obtaining the highest scores (Peters et al., 2018; Devlin et al., 2019; Radford et al., 2019; So et al., 2019). As a result, training a state-of-the-art model now requires substantial computational resources which demand considerable energy, along with the associated financial and environmental costs. Research and development of new models multiplies these costs by thousands of times by requiring retraining to experiment with model architectures and hyperparameters. Whereas a decade ago most

Consumption	CO_2e (lbs)
Air travel, 1 person, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experiments	78,468
Transformer (big)	192
w/ neural arch. search	626,155

Table 1: Estimated CO_2 emissions from training common NLP models, compared to familiar consumption.[1]

NLP models could be trained and developed on a commodity laptop or server, many now require multiple instances of specialized hardware such as GPUs or TPUs, therefore limiting access to these highly accurate models on the basis of finances.

Even when these expensive computational resources are available, model training also incurs a substantial cost to the environment due to the energy required to power this hardware for weeks or months at a time. Though some of this energy may come from renewable or carbon credit-offset resources, the high energy demands of these models are still a concern since (1) energy is not currently derived from carbon-neural sources in many locations, and (2) when renewable energy is available, it is still limited to the equipment we have to produce and store it, and energy spent training a neural network might better be allocated to heating a family's home. It is estimated that we must cut carbon emissions by half over the next decade to deter escalating rates of natural disaster, and based on the estimated CO_2 emissions listed in Table 1,

[1]Sources: (1) Air travel and per-capita consumption: https://bit.ly/2Hw0xWc; (2) car lifetime: https://bit.ly/2Qbr0w1.

3645

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 3645–3650
Florence, Italy, July 28 - August 2, 2019. ©2019 Association for Computational Linguistics

Green Field

- There is no awareness of the energy consumption.
- There is little information about the energy consumption of our decisions and practices as software architects and developers.
- Little is known about our footprint as users and developers. E.g, watch a movie in streaming platforms.

Greenpeace Report

- IT sector consumes 7% of global electricity (2017).
- *“The continued **lack of transparency** by many companies regarding their **energy demand** and the **supply of electricity** powering their data centers remains a **significant threat** to the sector’s long-term **sustainability**.”*
- Report provides an analysis of environmental sustainability of tech providers in different angles. **Transparency, Commitment, Energy Efficiency, Renewable Procurement, Advocacy**.

Gary Cook, Jude Lee, Tamina Tsai, Ada Kongn, John Deans, Brian Johnson, Elizabeth Jardim, and Brian Johnson. 2017. Clicking Clean: Who is winning the race to build a green internet? Technical report, Greenpeace.

	Final Grade	Clean Energy Index	Natural Gas	Coal	Nuclear
Adobe	B	23%	37%	23%	11%
Alibaba.com	D	24%	3%	67%	3%
amazon.com web services	C	17%	24%	30%	26%
Apple	A	83%	4%	5%	5%
Baidu 百度	F	24%	3%	67%	3%
Facebook	A	67%	7%	15%	9%
Google	A	56%	14%	15%	10%
hp	C	50%	17%	27%	5%

	Final Grade	Clean Energy Index	Natural Gas	Coal	Nuclear
IBM	C	29%	29%	27%	15%
Microsoft	B	32%	23%	31%	10%
NAVER	C	2%	19%	39%	31%
ORACLE	D	8%	26%	36%	25%
salesforce	B	43%	12%	16%	15%
SAMSUNG 삼성SDS	D	11%	19%	29%	31%
Tencent 腾讯	F	24%	3%	67%	3%

(Greenpeace, 2017)

(Side note)

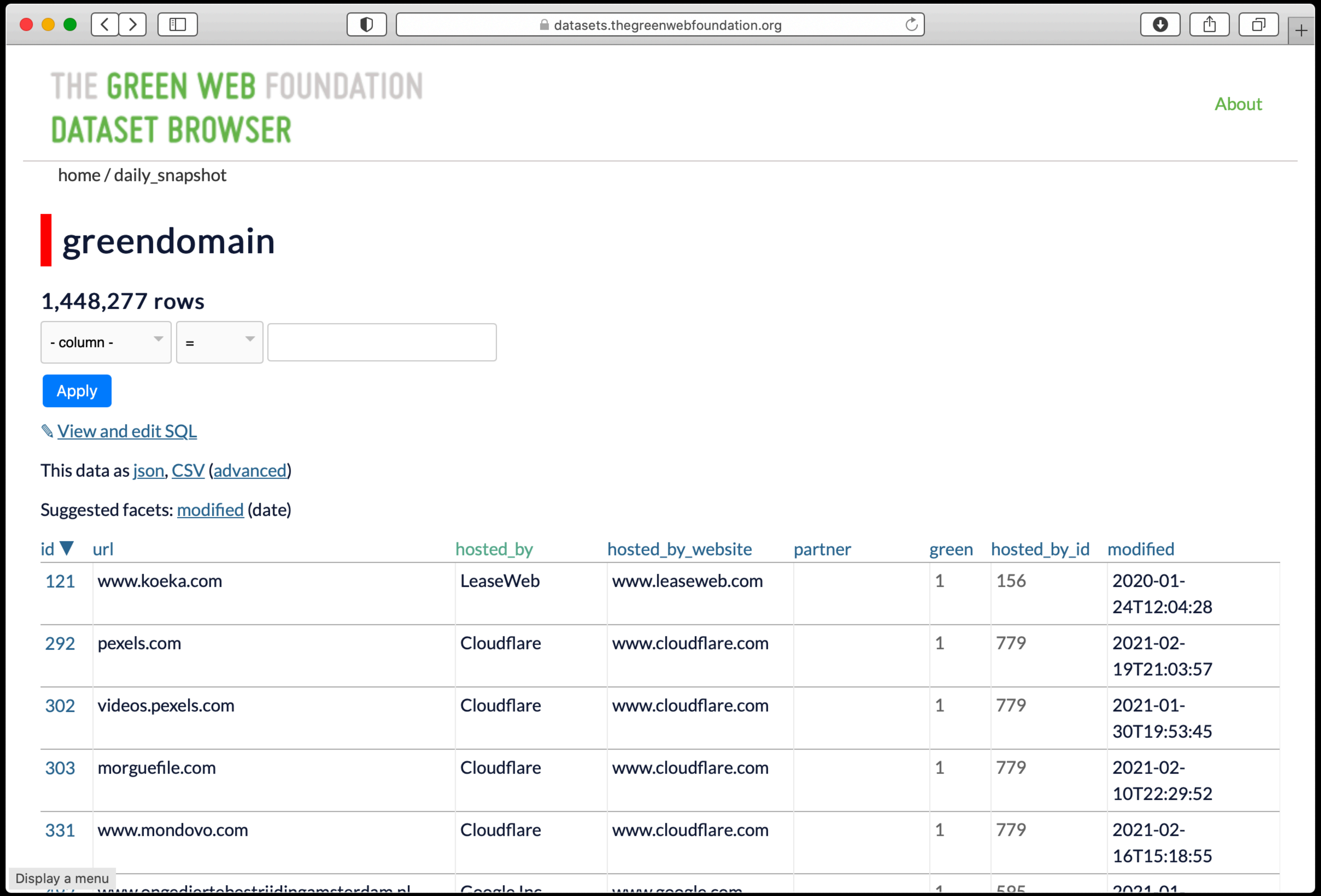
Video Streaming

	Final Grade	Clean Energy Index	Natural Gas	Coal	Nuclear
Afreeca.com	F	2%	19%	39%	31%
Amazon Prime	C	17%	24%	30%	26%
HBO	D	22%	20%	25%	25%
Hulu	F	20%	30%	29%	20%
Netflix	D	17%	24%	30%	26%
Pooq.co.kr	F	2%	19%	39%	31%
Vevo	F	27%	15%	32%	26%
Vimeo	D	47%	13%	20%	19%
YouTube	A	56%	15%	14%	10%

Music/Audio Streaming

	Final Grade	Clean Energy Index	Natural Gas	Coal	Nuclear
iTunes	A	83%	4%	5%	5%
NPR	F	17%	24%	30%	26%
Pandora	F	13%	32%	20%	27%
SoundCloud	F	17%	24%	30%	26%
Spotify	D	56%	15%	14%	10%
Podbbang	F	2%	19%	39%	31%

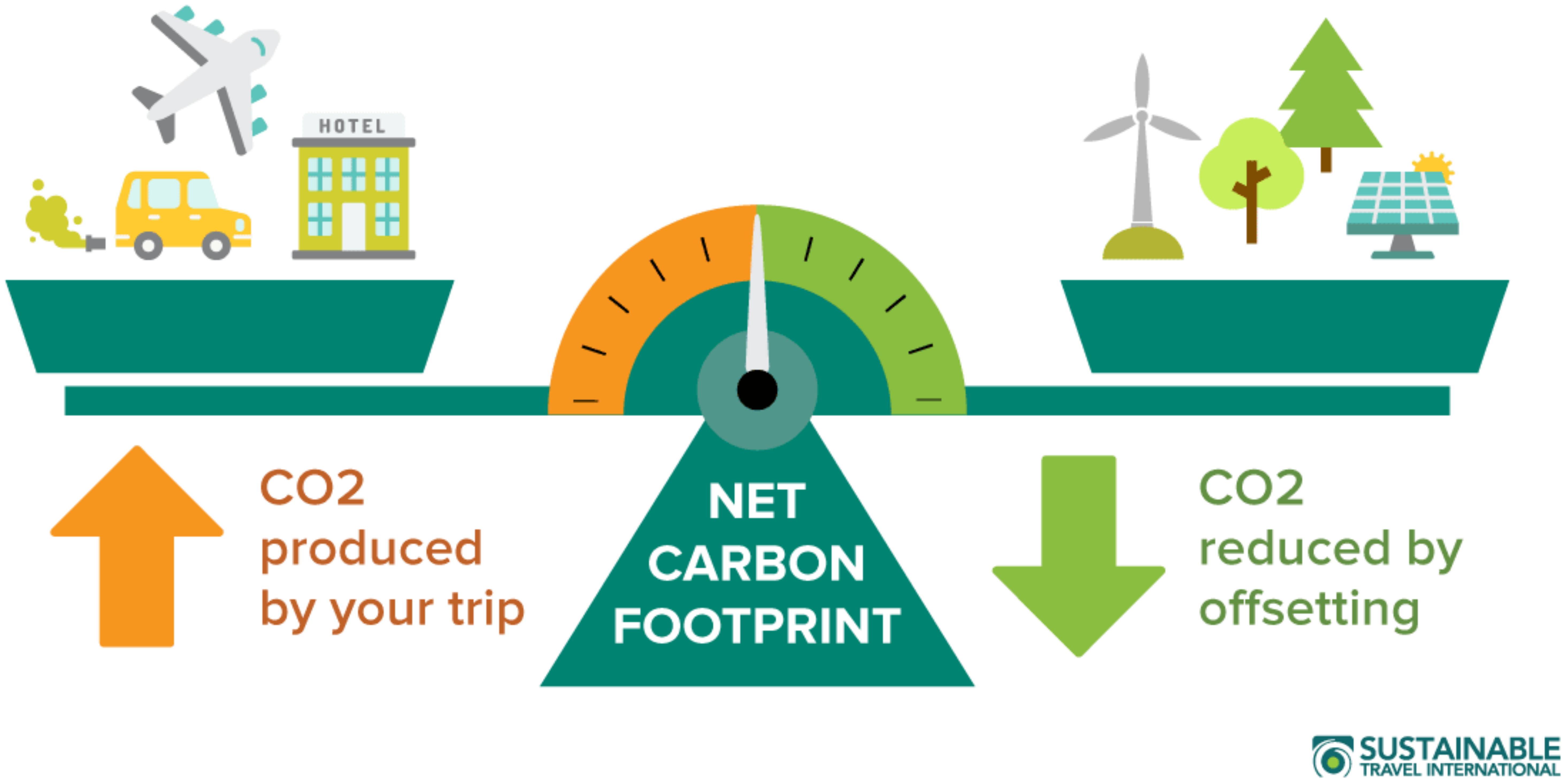
How to check for green hosts



THE GREEN WEB FOUNDATION

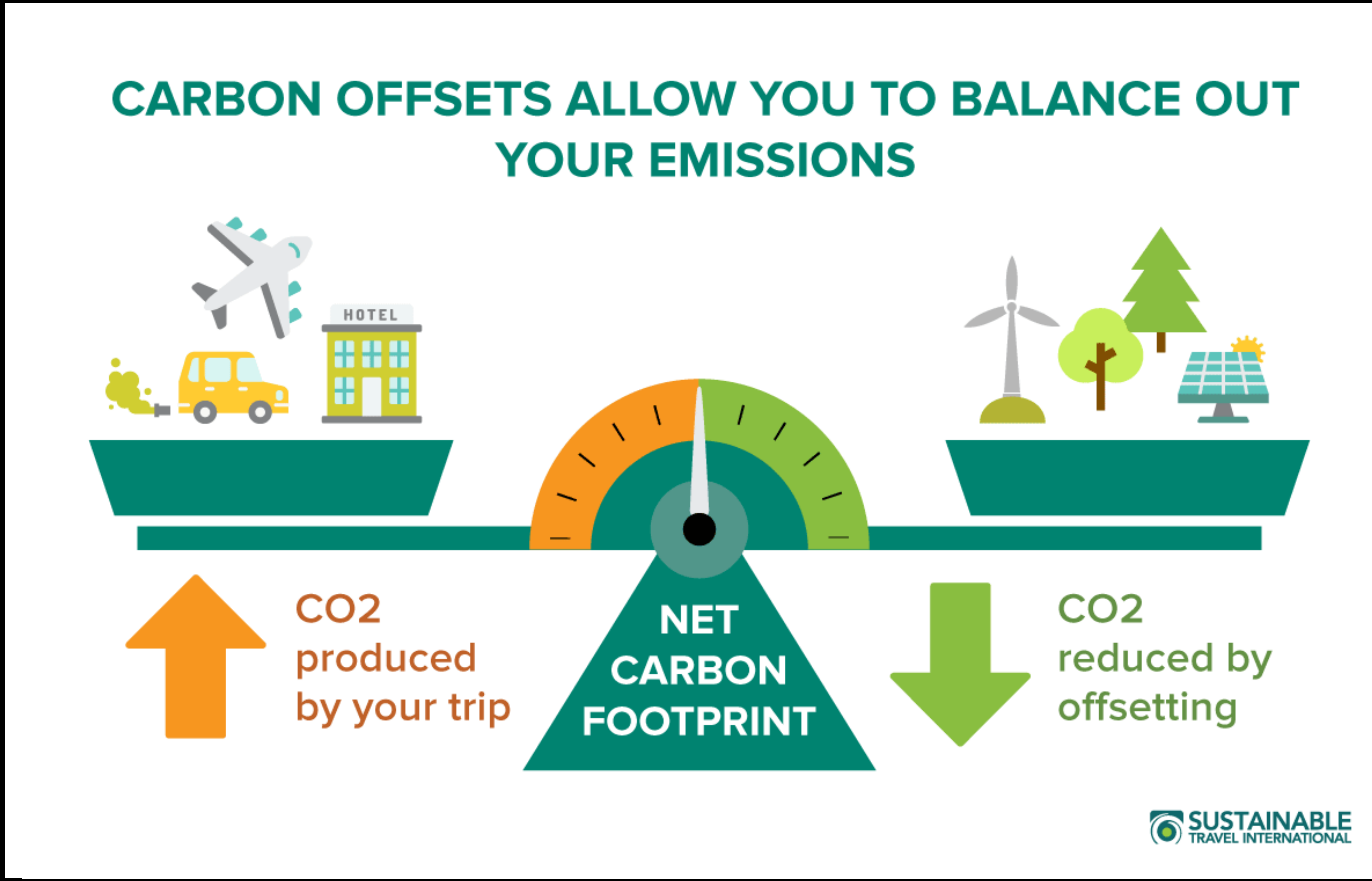
- Greendomain database. **Provided by The Green Web Foundation.**
- https://datasets.thegreenwebfoundation.org/daily_snapshot/greendomain

CARBON OFFSETS ALLOW YOU TO BALANCE OUT YOUR EMISSIONS?



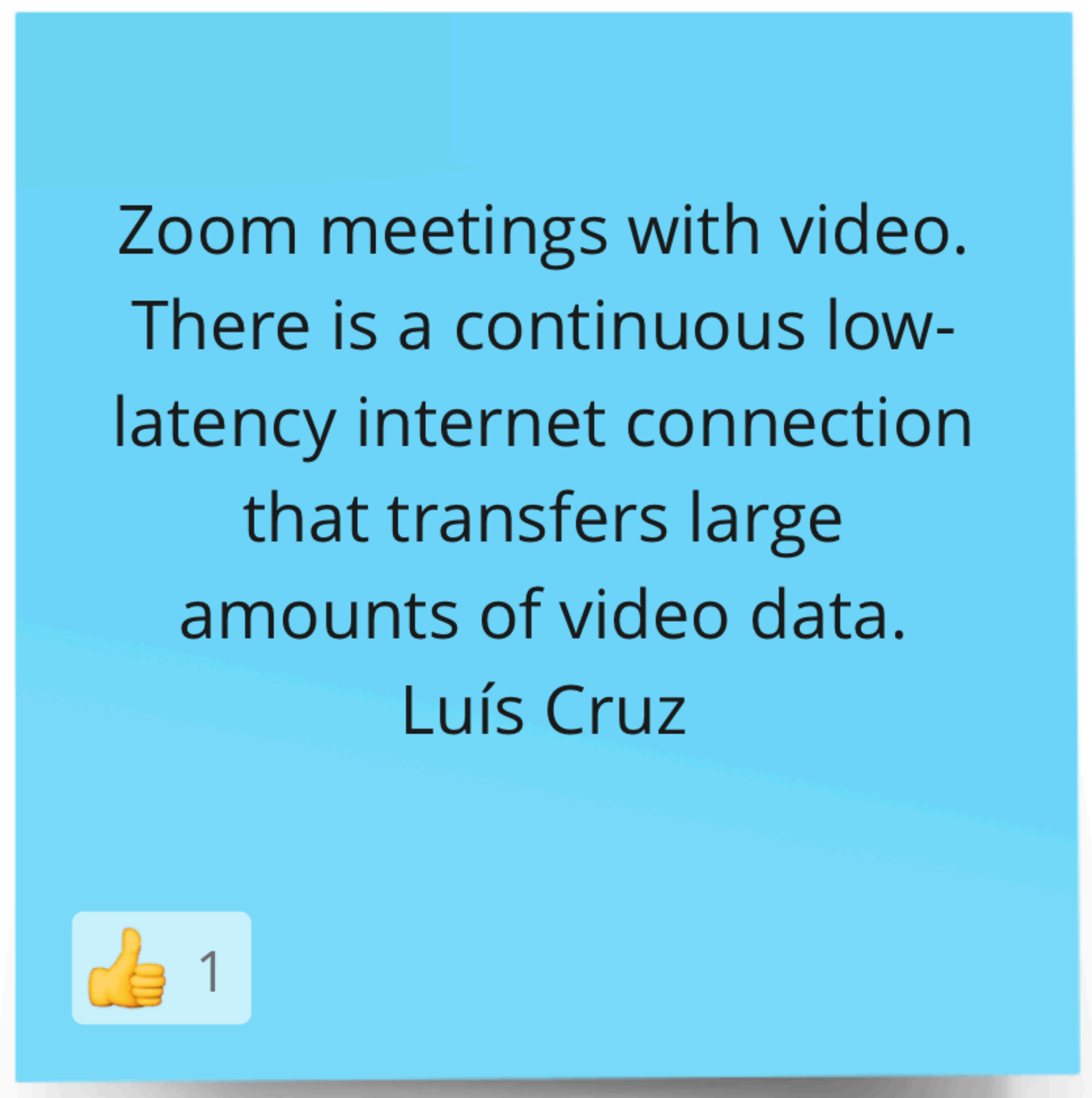
- **Not really**: Carbon offset is an easy strategy that allows large polluting services to simply throw money at the moment.

- It practice, it boils down to creating **monocultures of trees** in **underdeveloped countries**.



Checkpoint 1

- From all the things you do as a Computer Science expert, name a few that you find to be carbon intensive.
- Add a sticky note with a brief answer (200 max). Add your name in the end.
- Upvote other answers using the thumbs up 👍 emoji.
- Some sticky notes will be selected for discussion.
  - What is the trade-off between carbon intensity and usefulness?
  - How could we measure?
- Miro board: https://edu.nl/8b639



edu.nl/8b639

Sources of Energy Consumption

Sources of Energy Consumption

Execution

Sources of Energy Consumption

Execution	Development

Sources of Energy Consumption

Execution	Development	Infrastructure

How to Measure Energy Consumption

How to Measure Energy Consumption

1. Create a scenario.

How to Measure Energy Consumption

1. Create a scenario.

2. Execute and **collect power data**.

How to Measure Energy Consumption

1. Create a scenario.
2. Execute and **collect power data**.
3. Implement energy improvement.

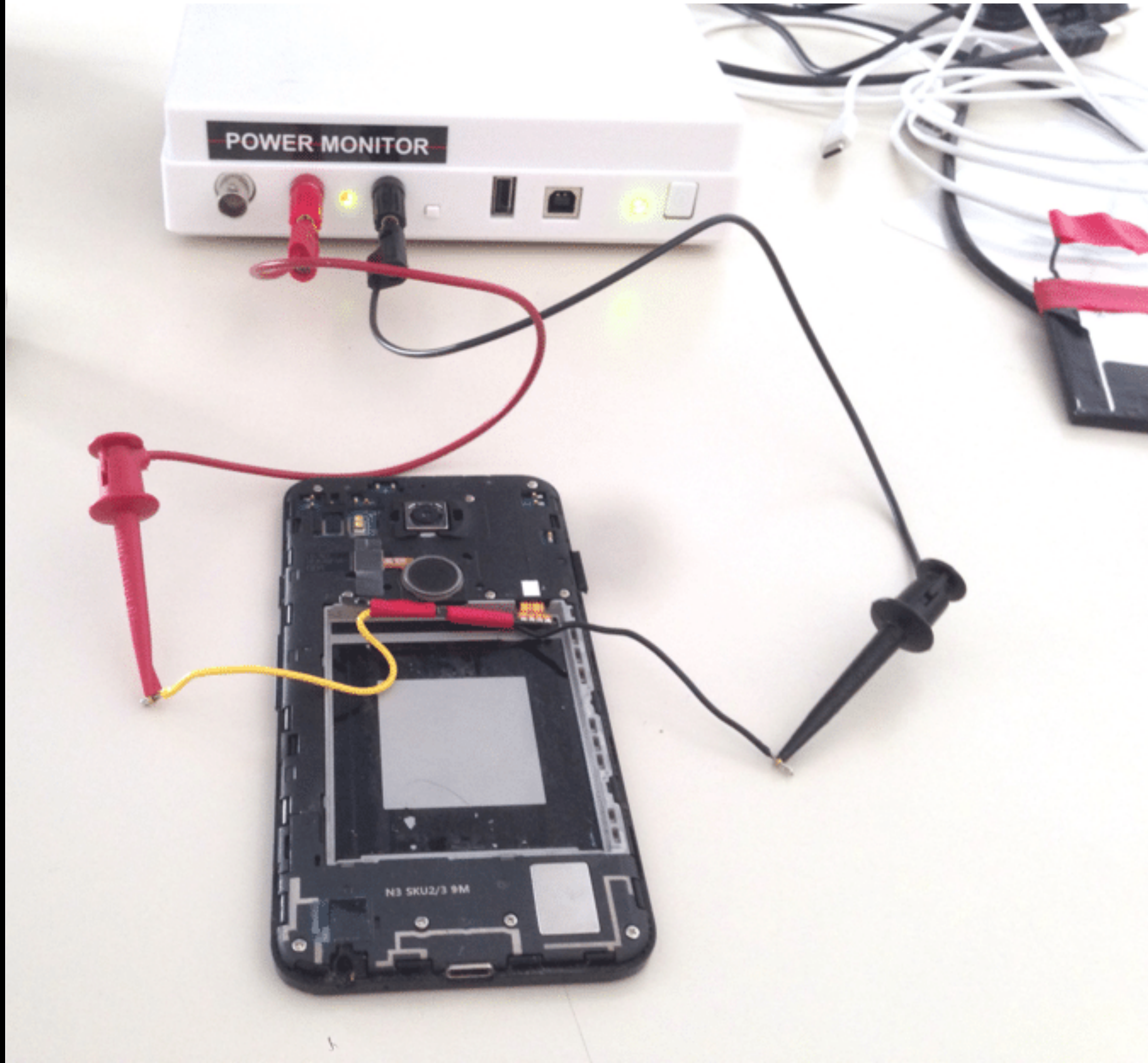
How to Measure Energy Consumption

1. Create a scenario.
2. Execute and **collect power data**.
3. Implement energy improvement.
4. Execute and collect power data.

How to Measure Energy Consumption

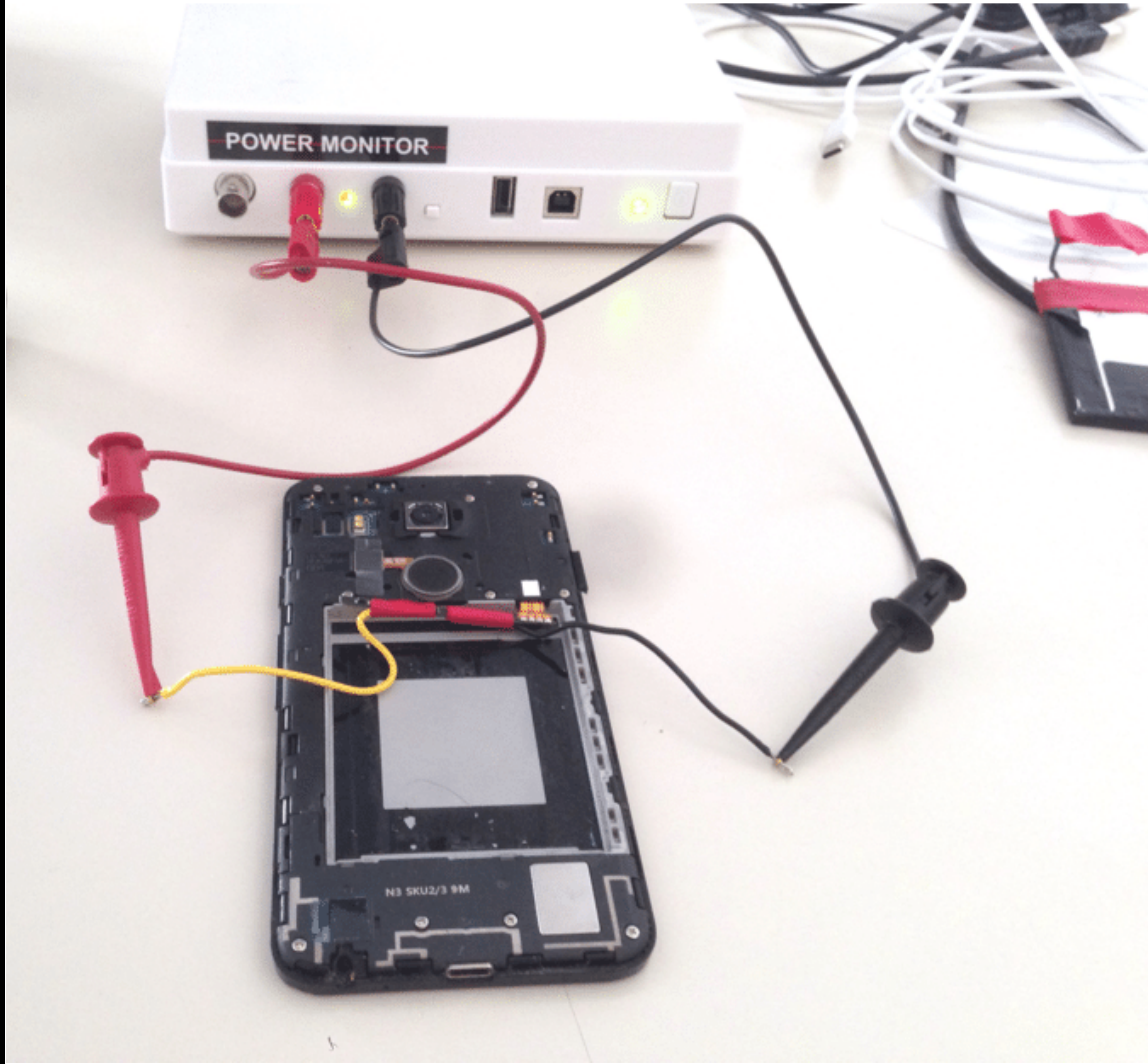
1. Create a scenario.
2. Execute and **collect power data**.
3. Implement energy improvement.
4. Execute and collect power data.
5. Analyse and compare results.

Collect Power Data



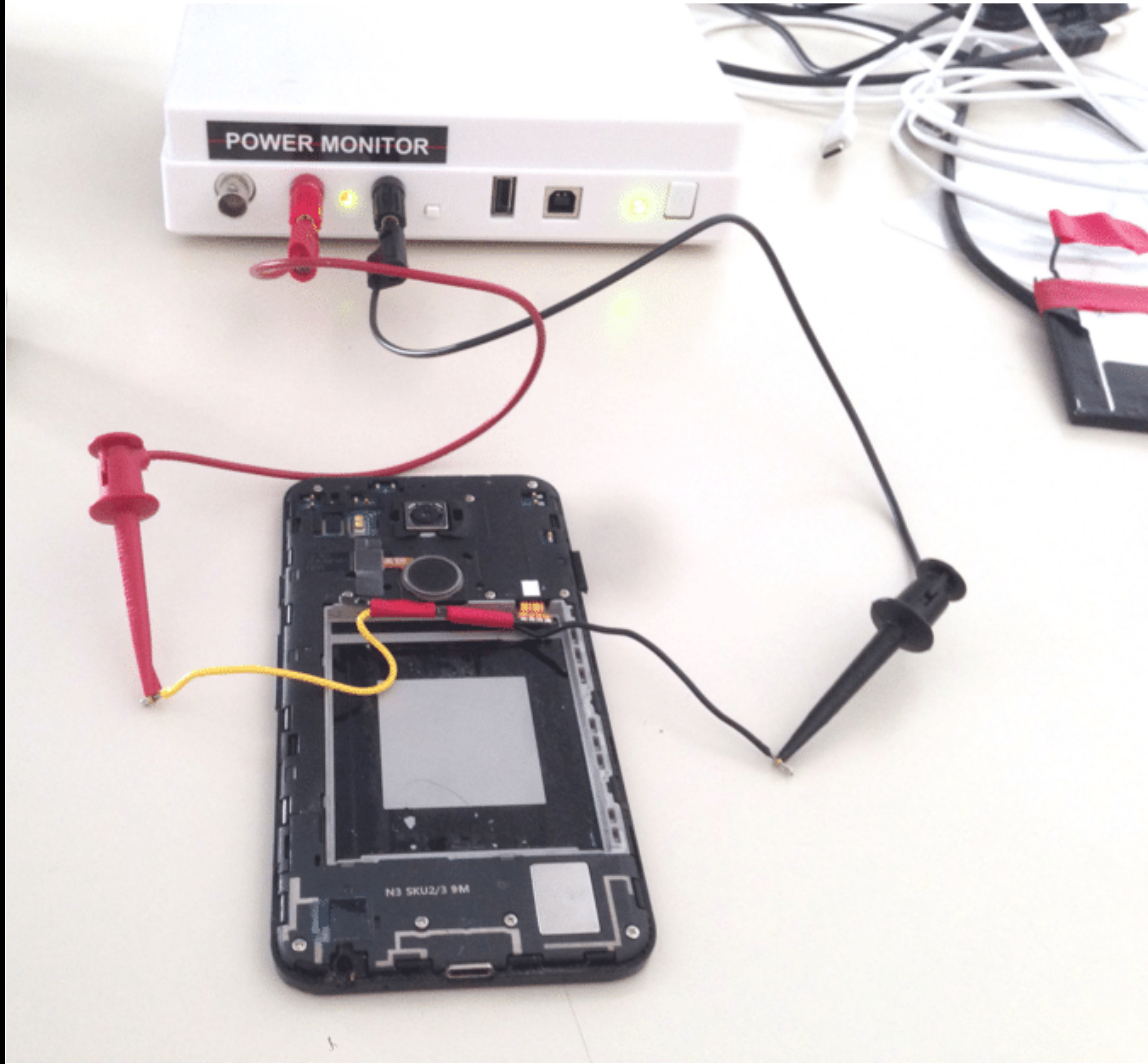
Collect Power Data

- Electricity bill 🤑



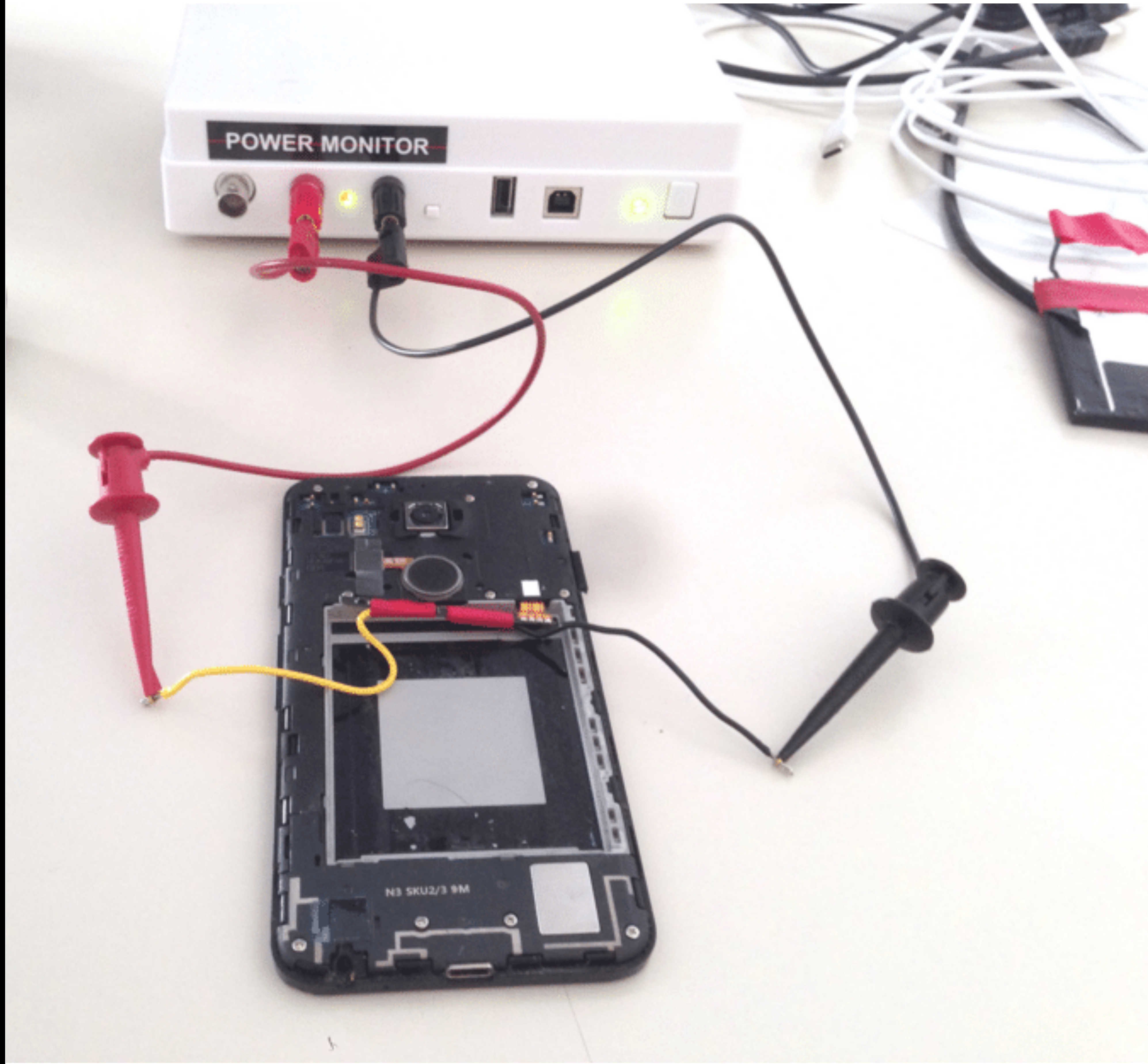
Collect Power Data

- Electricity bill 🤑
- Execution time



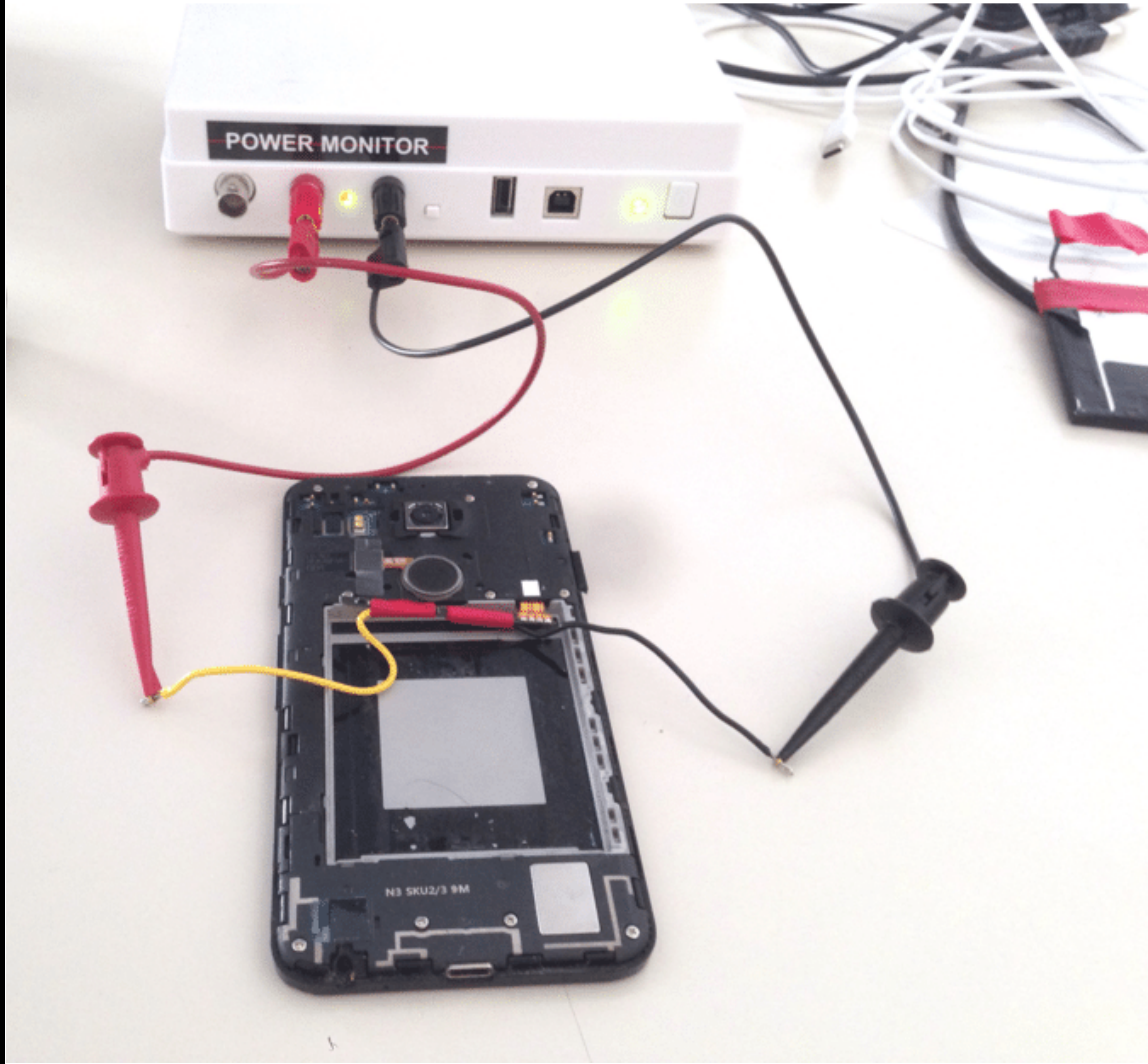
Collect Power Data

- Electricity bill 🤑
- Execution time
- Estimation tools (a.k.a. energy profilers)



Collect Power Data

- Electricity bill 🤑
- Execution time
- Estimation tools (a.k.a. energy profilers)
- Power Monitors (e.g., Monsoon)



Estimation tools

Estimation tools

- Windows Energy Estimation Engine (E3)
  7-day dump `> powercfg.exe /srumutil`
  https://edu.nl/mdkvc

Estimation tools

- Windows Energy Estimation Engine (E3)
  7-day dump `> powercfg.exe /srumutil`
  https://edu.nl/mdkvc

- Intel RAPL (Linux and Mac).

Estimation tools

- Windows Energy Estimation Engine (E3)
  7-day dump `> powercfg.exe /srumutil`
  https://edu.nl/mdkvc

- Intel RAPL (Linux and Mac).

- **Powerstat (Linux)** 🟢
  https://edu.nl/edcb9

Estimation tools

- Windows Energy Estimation Engine (E3)
  7-day dump `> powercfg.exe /srumutil`
  https://edu.nl/mdkvc

- Intel RAPL (Linux and Mac).

- **Powerstat (Linux)** 🟢
  https://edu.nl/edcb9

- Powermetrics (Mac)

Estimation tools

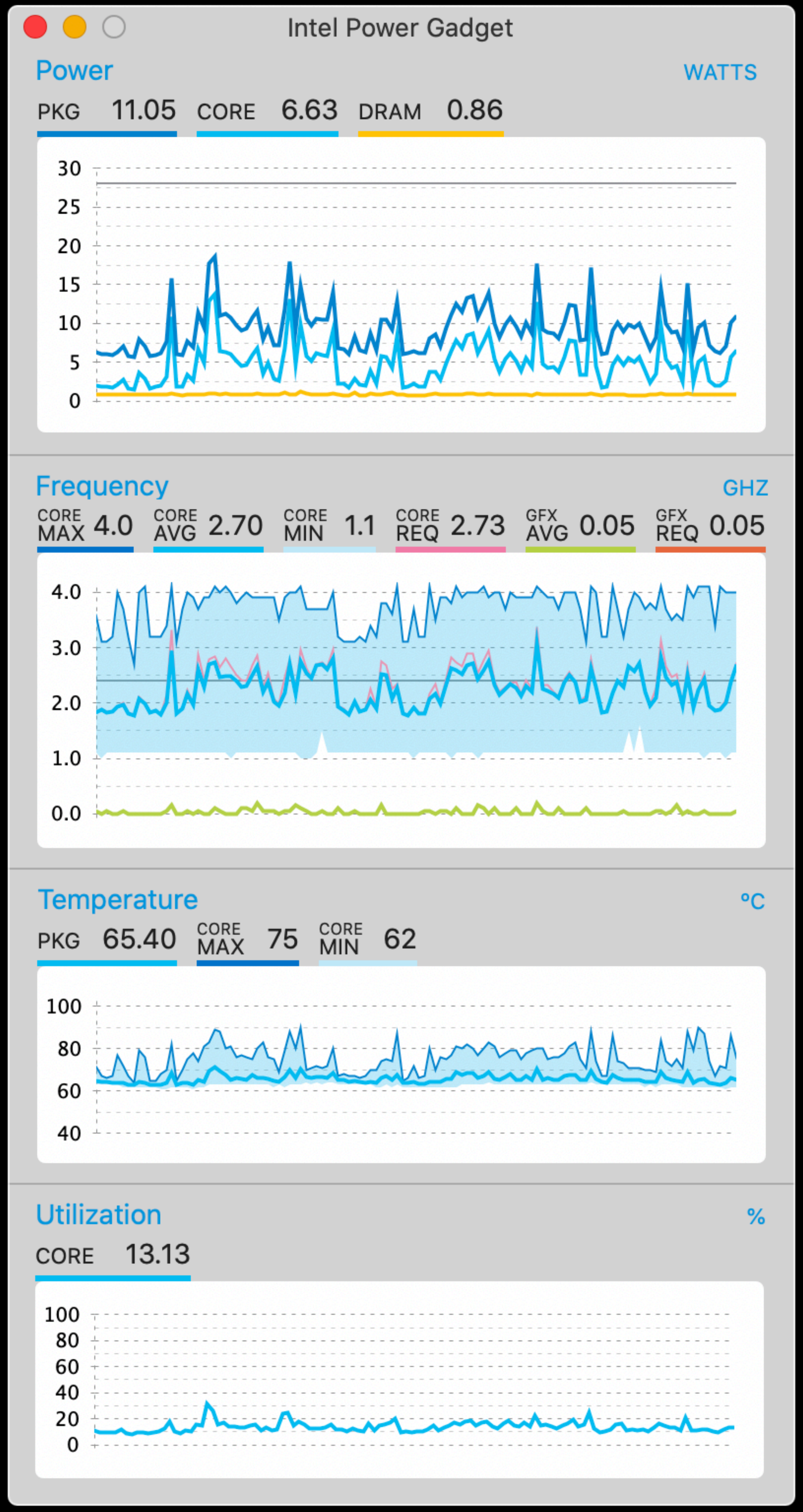
- Windows Energy Estimation Engine (E3)
  7-day dump `> powercfg.exe /srumutil`
  https://edu.nl/mdkvc

- Intel RAPL (Linux and Mac).

- **Powerstat (Linux)** 🟢
  https://edu.nl/edcb9

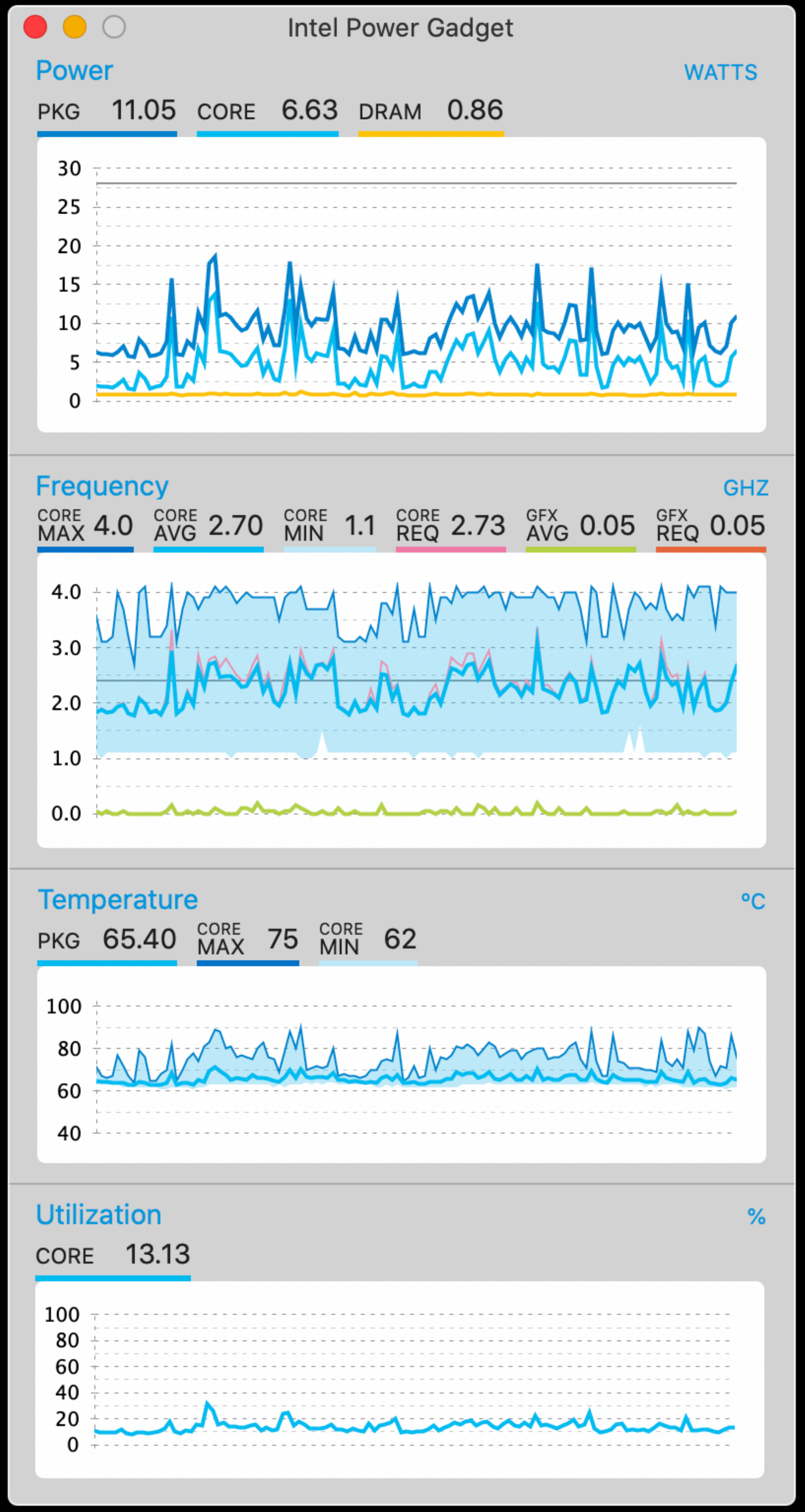
- Powermetrics (Mac)

- **Intel PowerGadget (Windows and Mac)** 🟢
  https://edu.nl/xmdvd



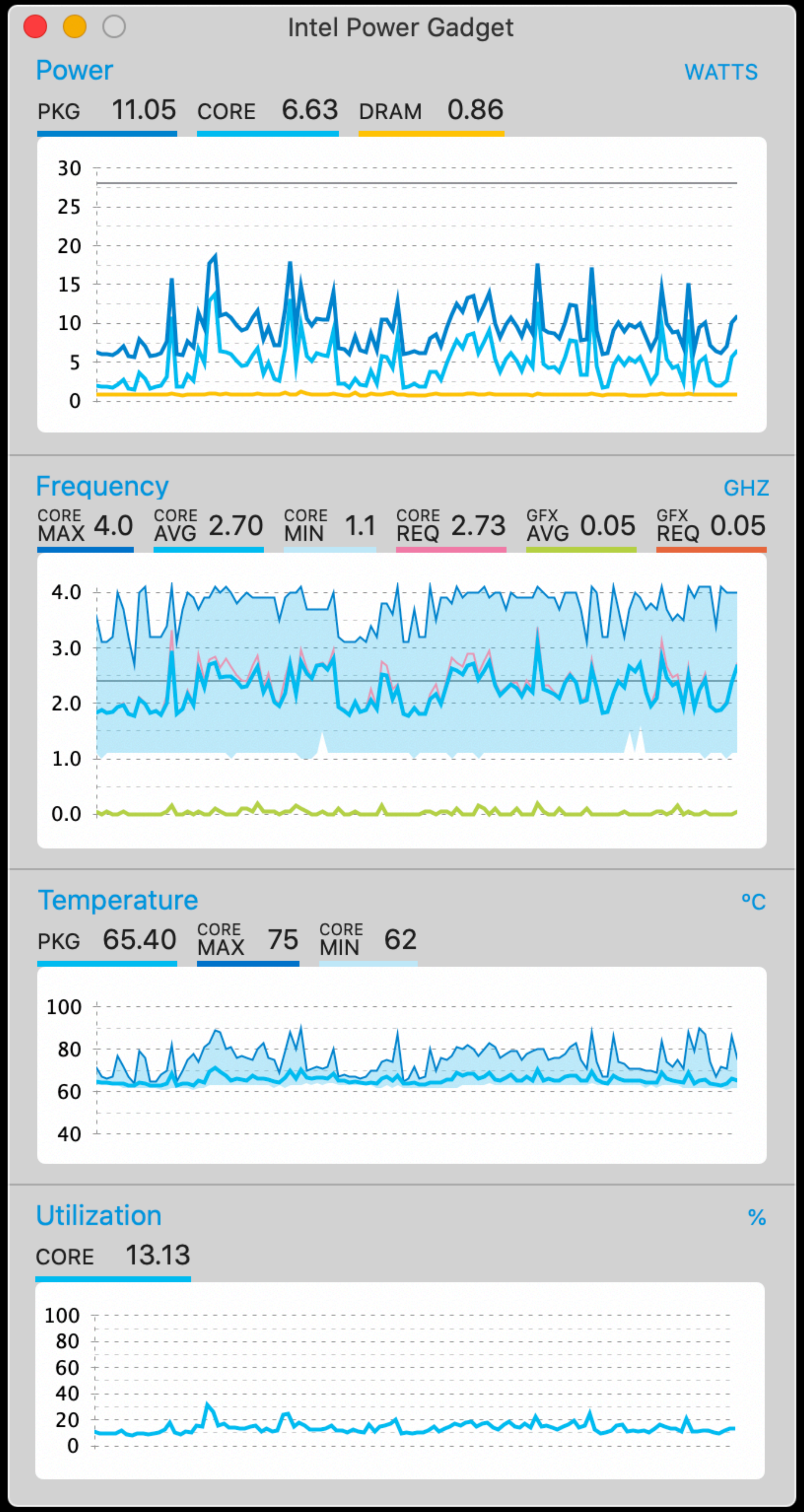
Estimation tools

- Windows Energy Estimation Engine (E3)
  7-day dump `> powercfg.exe /srumutil`
  https://edu.nl/mdkvc
- Intel RAPL (Linux and Mac).
- **Powerstat (Linux)** 🟢
  https://edu.nl/edcb9
- Powermetrics (Mac)
- **Intel PowerGadget (Windows and Mac)** 🟢
  https://edu.nl/xmdvd
- Intel PowerLog. CLI tool shipped with PowerGadget. Measure any given bash command.
  > `/Applications/Intel\ Power\ Gadget/PowerLog -cmd <CMD>`



Profiler Live Demo

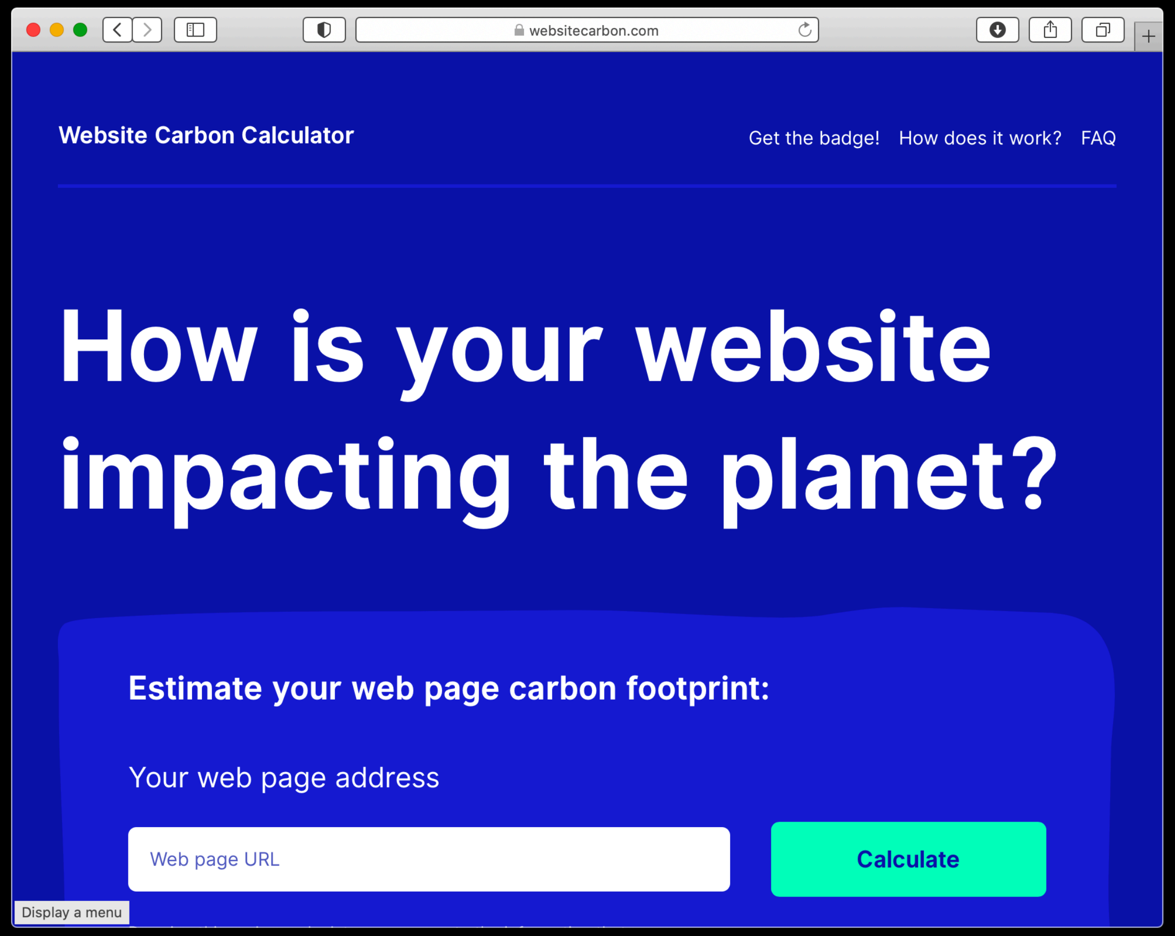
Profiler Live Demo



Other estimation tools

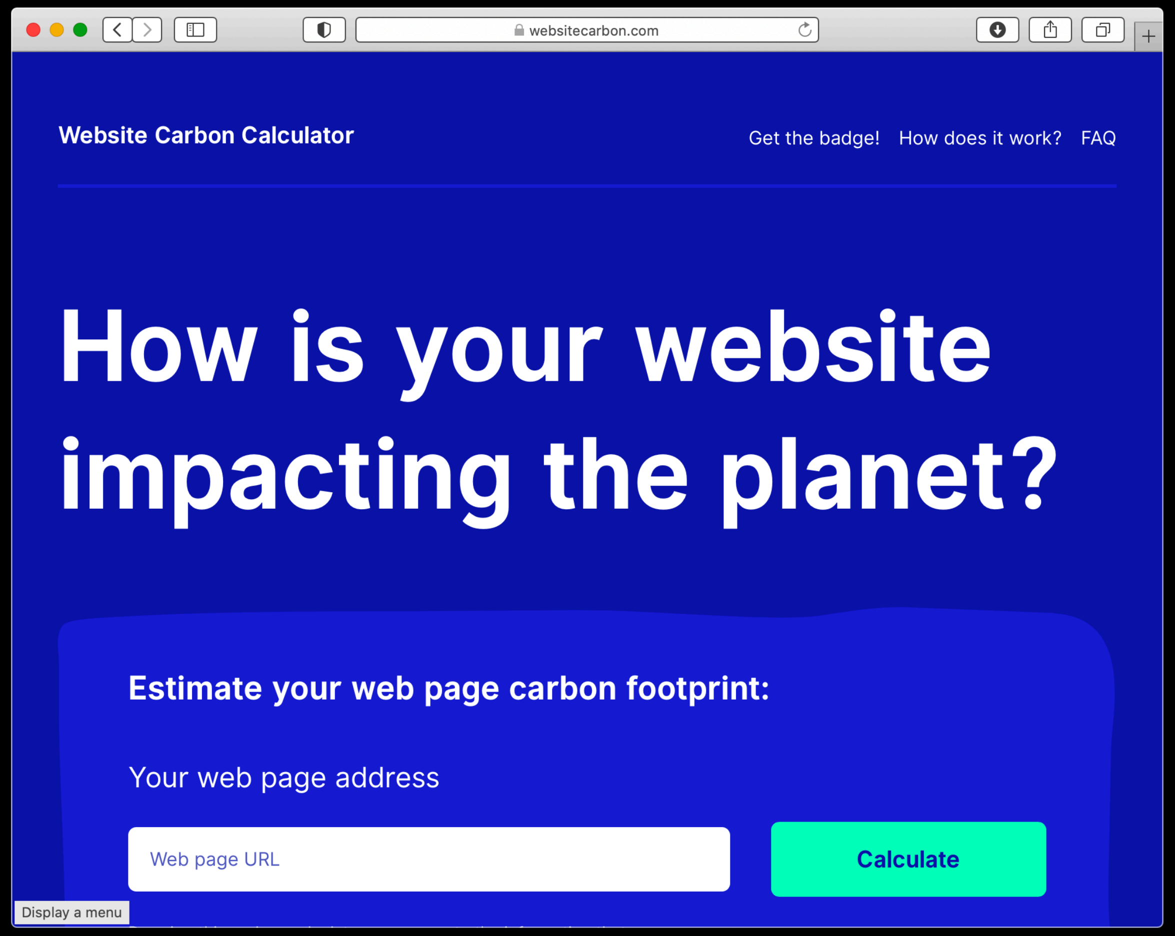
Other estimation tools

- Website Carbon Calculator. **#LetsGreenTheWeb**
  https://www.websitecarbon.com

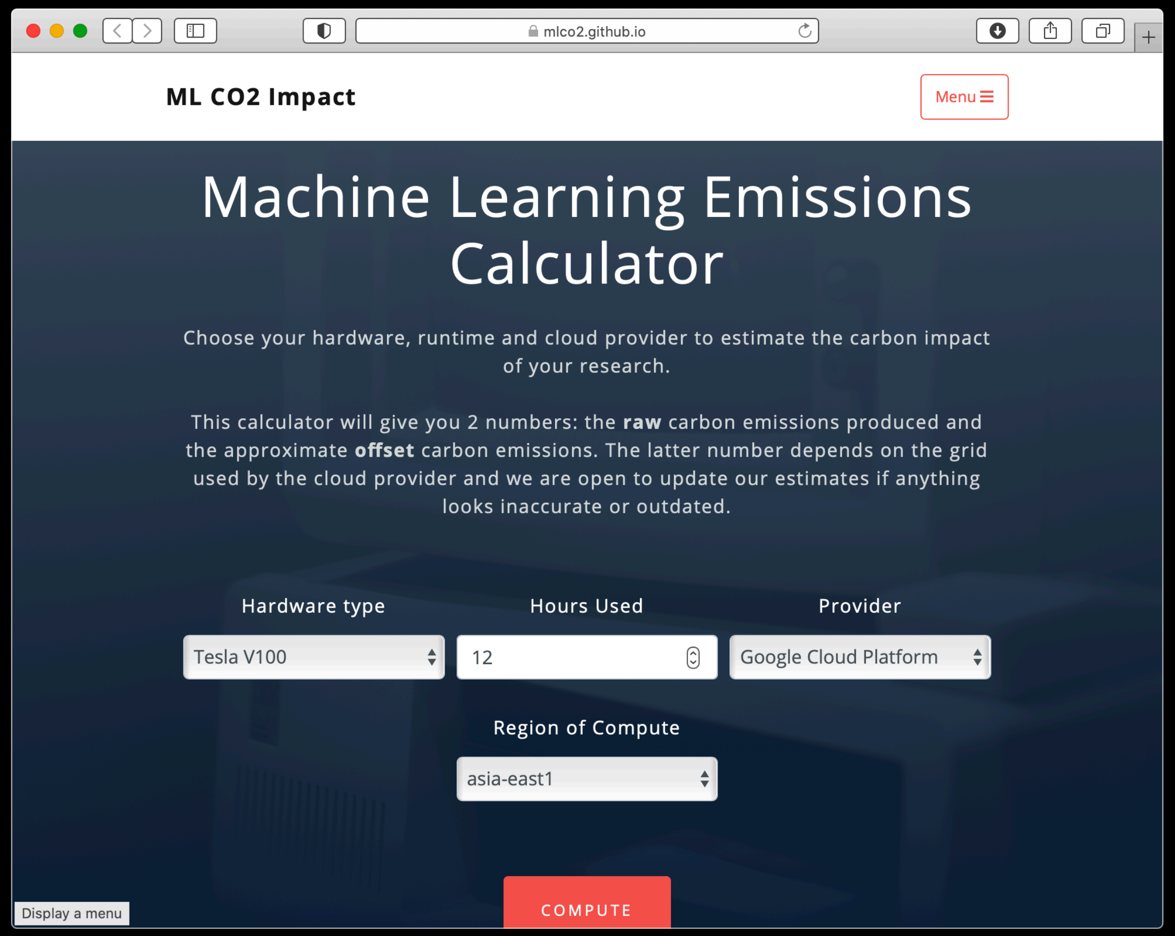


Other estimation tools

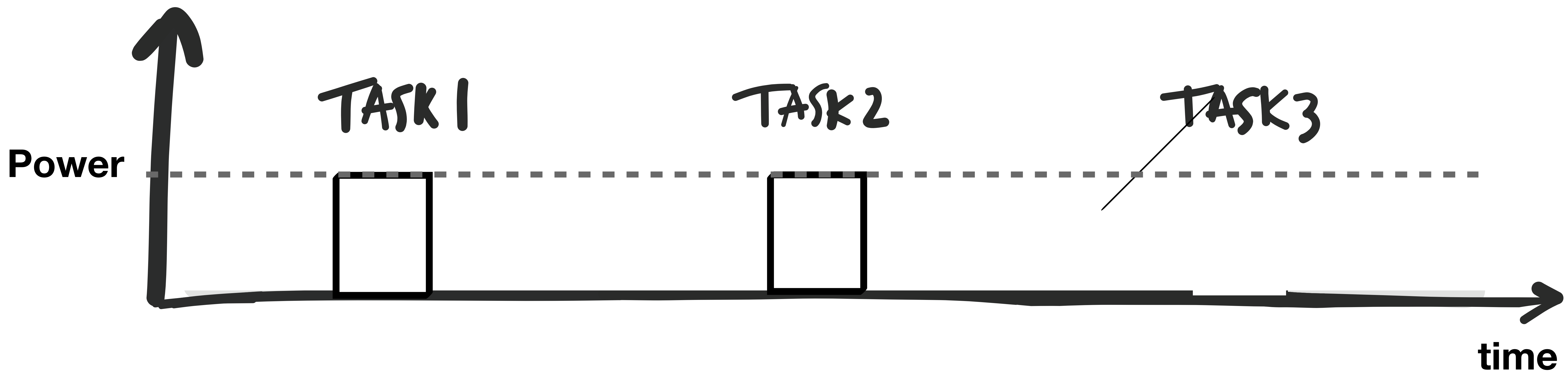
- Website Carbon Calculator. **#LetsGreenTheWeb**
  https://www.websitecarbon.com



- ML CO2 Impact. **Extra: it generates badges in LaTeX for ML projects.**
  https://mlco2.github.io/impact/

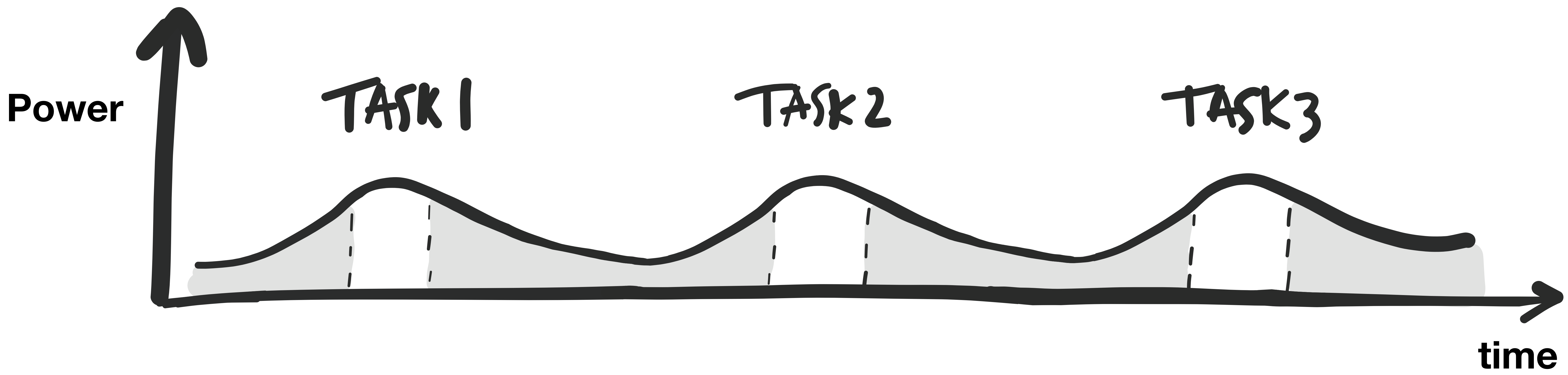


Going from Power samples to Energy Consumption

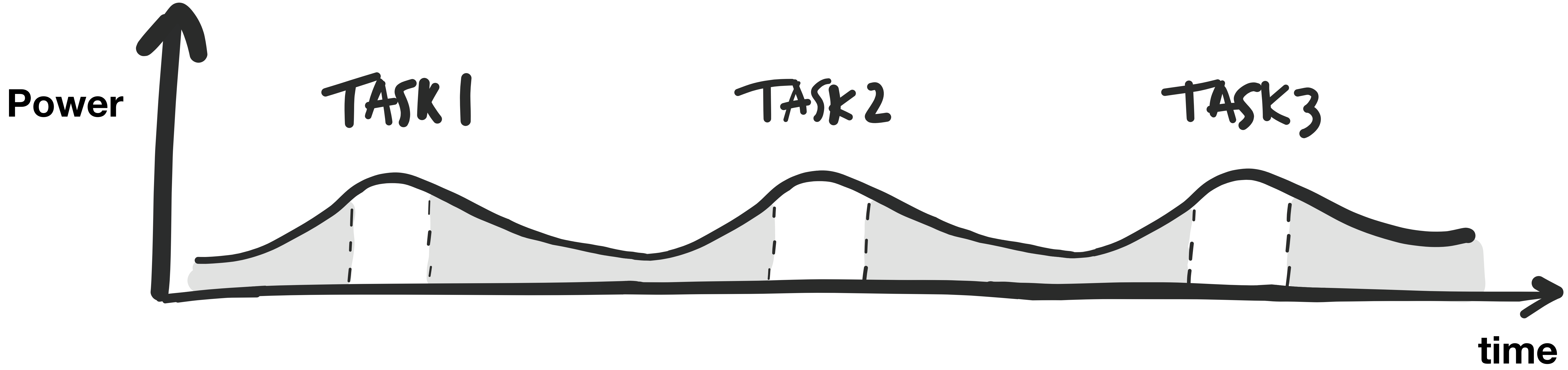


$$W = P \times \Delta t$$ 👎

Going from Power samples to Energy Consumption



Going from Power samples to Energy Consumption



$$W = \int_{t_1}^{t_2} P(t) \cdot \mathrm{d}t$$ 👍

Checkpoint 2

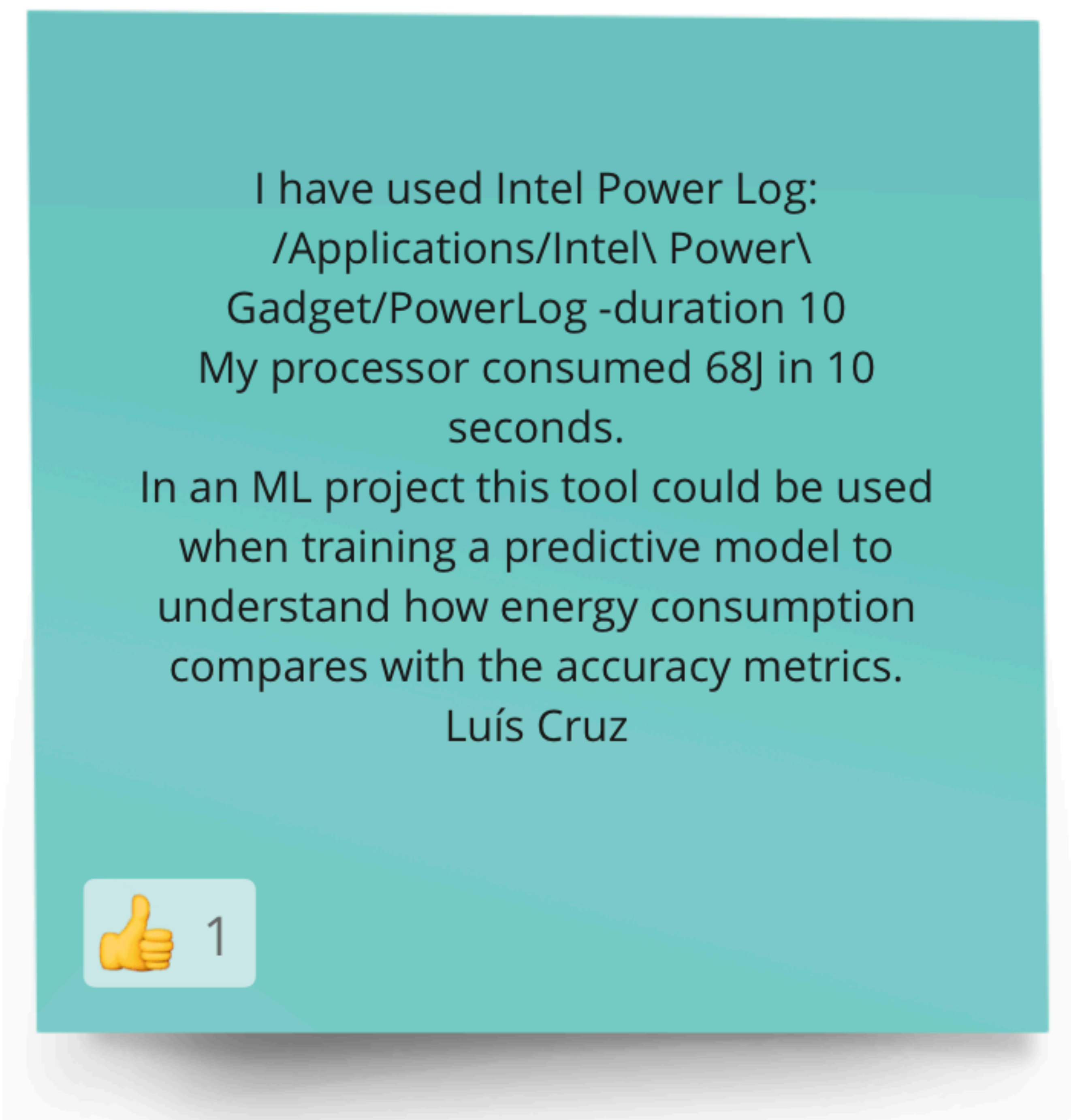
I have used Intel Power Log:
/Applications/Intel\ Power\ Gadget/PowerLog -duration 10
My processor consumed 68J in 10 seconds.
In an ML project this tool could be used when training a predictive model to understand how energy consumption compares with the accuracy metrics.
Luís Cruz

👍 1

edu.nl/8b639

Checkpoint 2

- Use an energy profiler of your choice to collect power measurements from your computer.



edu.nl/8b639

Checkpoint 2

- Use an energy profiler of your choice to collect power measurements from your computer.
  - Recommended energy profilers: Energy Profilers: **Intel Power Gadget** (Mac/Windows) or **Powerstat** (Linux).

I have used Intel Power Log:
/Applications/Intel\ Power\ Gadget/PowerLog -duration 10
My processor consumed 68J in 10 seconds.
In an ML project this tool could be used when training a predictive model to understand how energy consumption compares with the accuracy metrics.
Luís Cruz

👍 1

edu.nl/8b639

Checkpoint 2

- Use an energy profiler of your choice to collect power measurements from your computer.
    - Recommended energy profilers: Energy Profilers: **Intel Power Gadget** (Mac/Windows) or **Powerstat** (Linux).
- Use a sticky note to explain the following: (400 chars max)

I have used Intel Power Log:
/Applications/Intel\ Power\ Gadget/PowerLog -duration 10
My processor consumed 68J in 10 seconds.
In an ML project this tool could be used when training a predictive model to understand how energy consumption compares with the accuracy metrics.
Luís Cruz

👍 1

edu.nl/8b639

Checkpoint 2

- Use an energy profiler of your choice to collect power measurements from your computer.
  - Recommended energy profilers: Energy Profilers: **Intel Power Gadget** (Mac/Windows) or **Powerstat** (Linux).
- Use a sticky note to explain the following: (400 chars max)
  - How you used the tool and what was the final energy consumption.

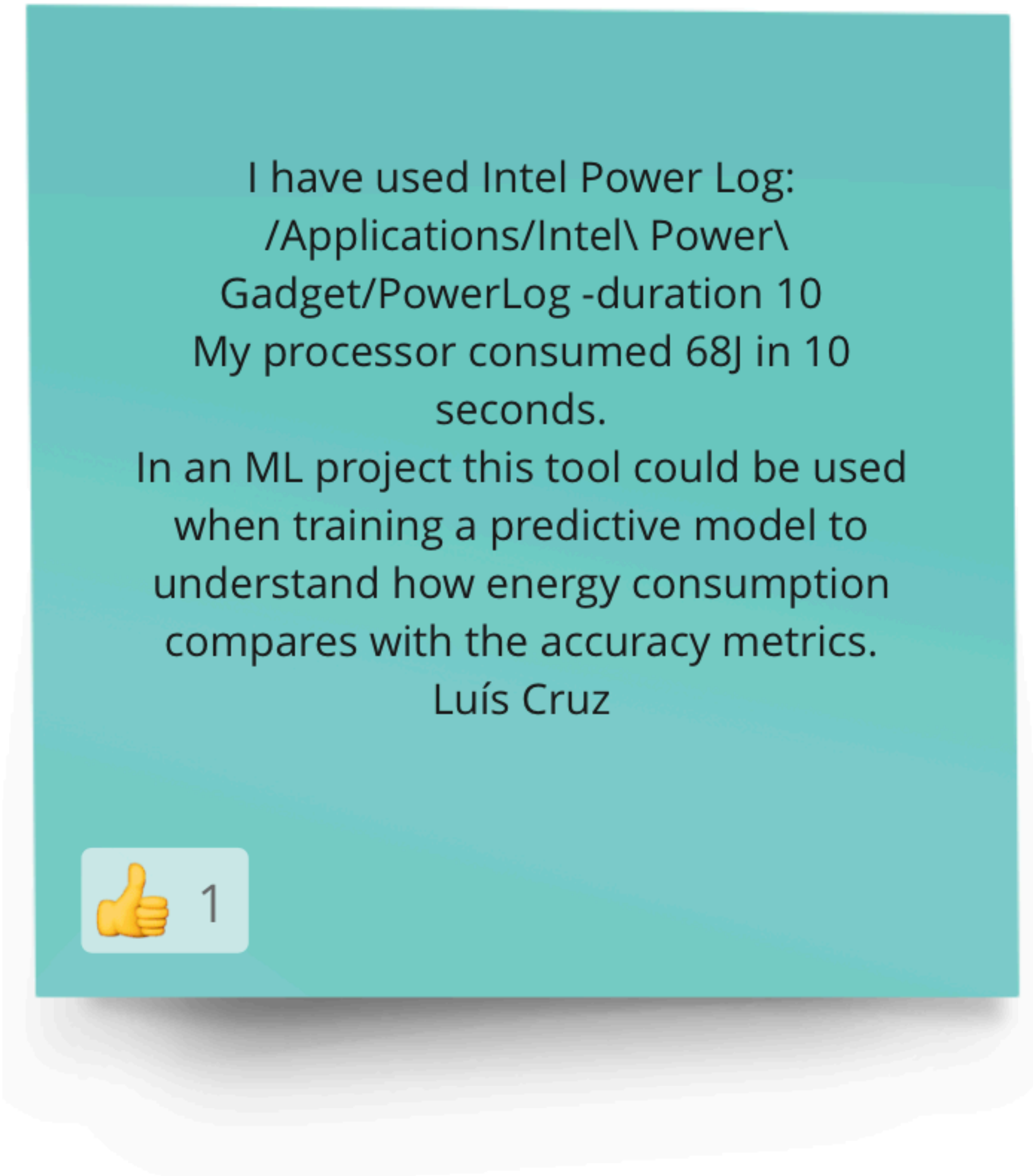
I have used Intel Power Log:
/Applications/Intel\ Power\
Gadget/PowerLog -duration 10
My processor consumed 68J in 10
seconds.
In an ML project this tool could be used
when training a predictive model to
understand how energy consumption
compares with the accuracy metrics.
Luís Cruz

👍 1

edu.nl/8b639

Checkpoint 2

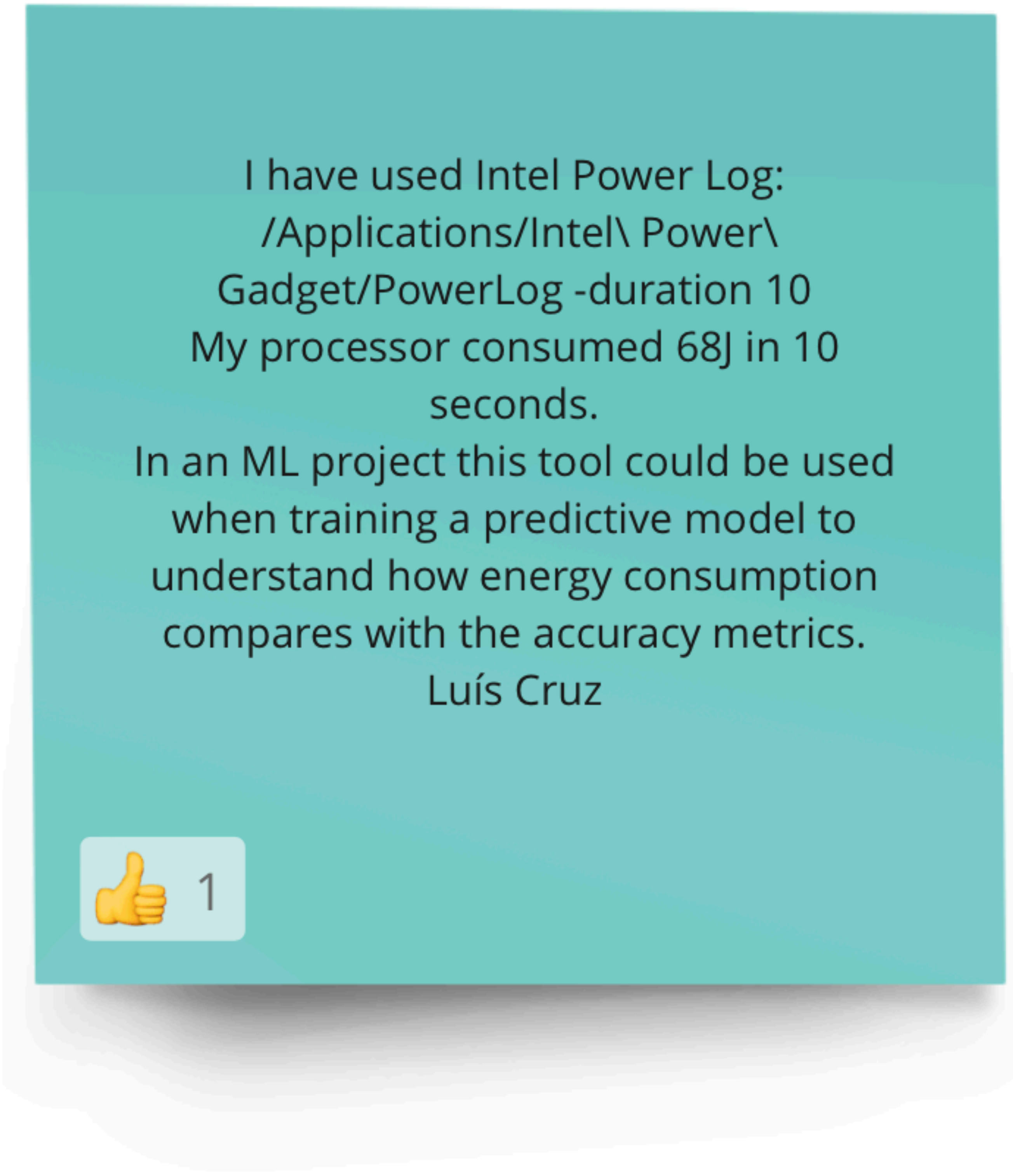
- Use an energy profiler of your choice to collect power measurements from your computer.
  - Recommended energy profilers: Energy Profilers: **Intel Power Gadget** (Mac/Windows) or **Powerstat** (Linux).
- Use a sticky note to explain the following: (400 chars max)
  - How you used the tool and what was the final energy consumption.
  - How the profiler could be used to test energy efficiency in a software project.



edu.nl/8b639

Checkpoint 2

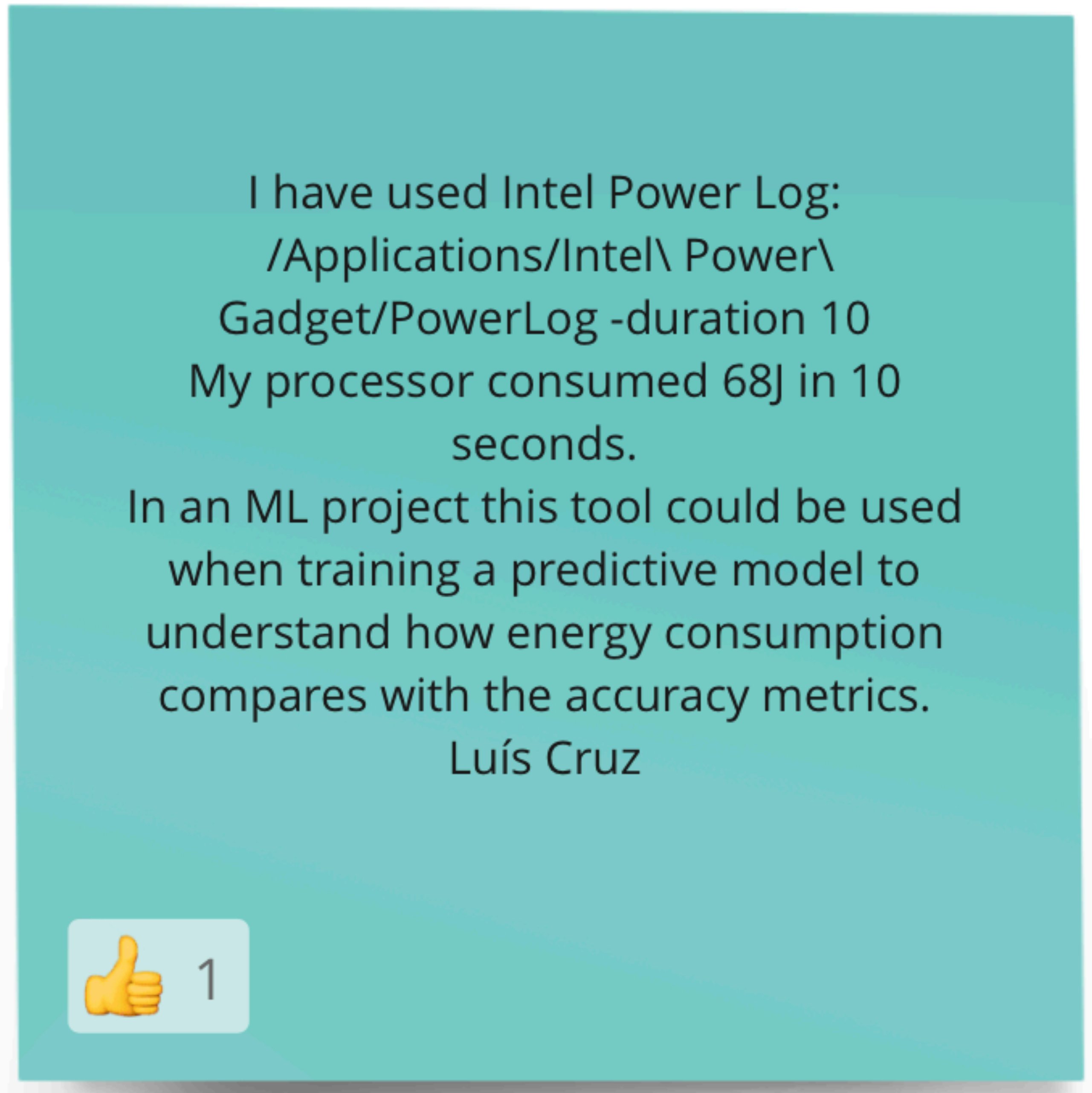
- Use an energy profiler of your choice to collect power measurements from your computer.
  - Recommended energy profilers: Energy Profilers: **Intel Power Gadget** (Mac/Windows) or **Powerstat** (Linux).
- Use a sticky note to explain the following: (400 chars max)
  - How you used the tool and what was the final energy consumption.
  - How the profiler could be used to test energy efficiency in a software project.
- Read some of your colleague's answers and upvote your favourite with the emoji 👍.



edu.nl/8b639

Checkpoint 2

- Use an energy profiler of your choice to collect power measurements from your computer.
  - Recommended energy profilers: Energy Profilers: **Intel Power Gadget** (Mac/Windows) or **Powerstat** (Linux).
- Use a sticky note to explain the following: (400 chars max)
  - How you used the tool and what was the final energy consumption.
  - How the profiler could be used to test energy efficiency in a software project.
- Read some of your colleague's answers and upvote your favourite with the emoji 👍.
- Miro board: https://edu.nl/8b639



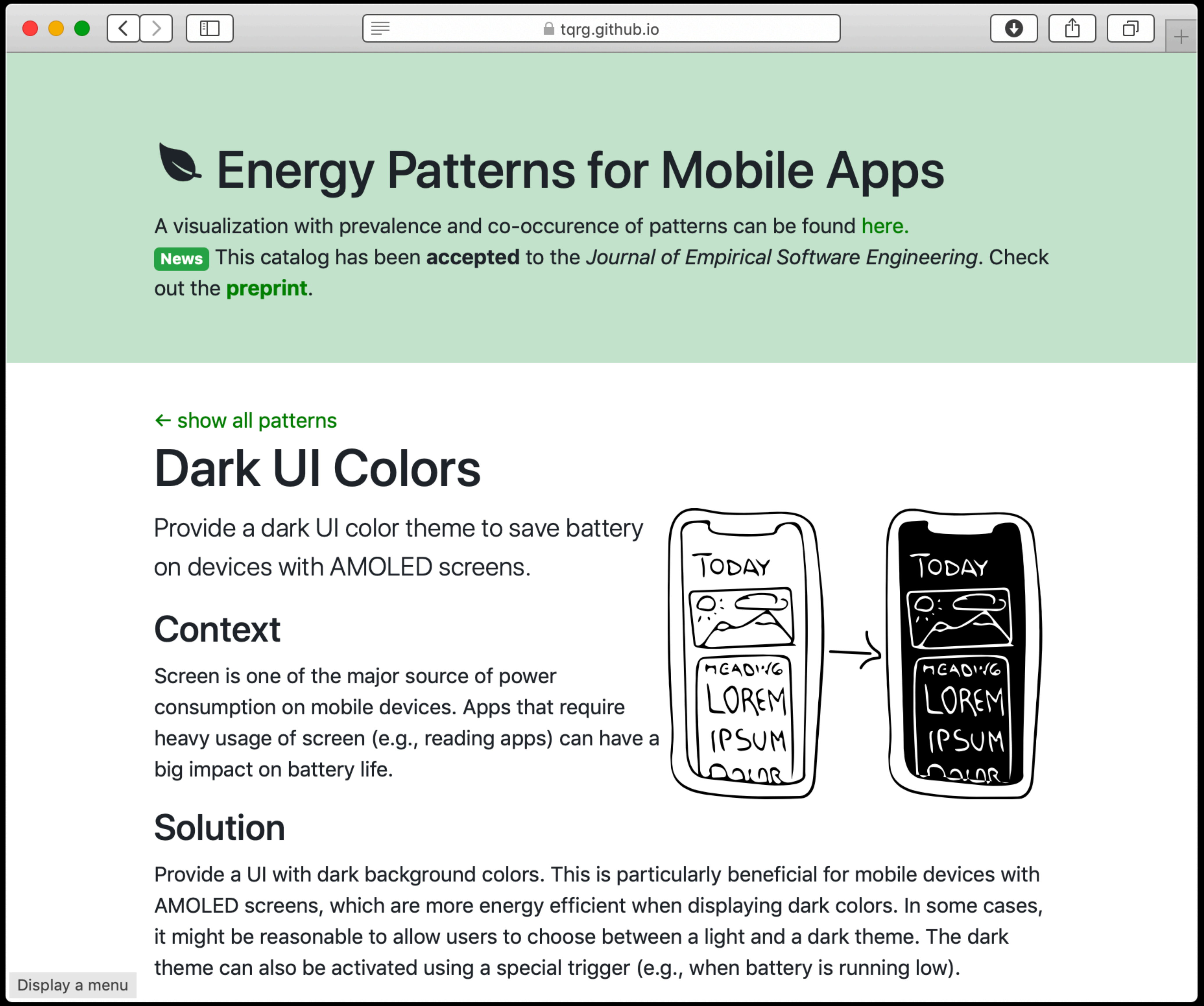
edu.nl/8b639

- Measuring energy consumption is difficult!
- Solutions?

- Measuring energy consumption is difficult!
- Solutions?
- **Software Design Pattern**
  *General, **reusable solution** to a recurrent problem within a given context in software design.*

- Measuring energy consumption is difficult!
- Solutions?
- **Software Design Pattern**
  General, ***reusable solution*** *to a recurrent problem within a given context in software design.*
- **Energy Pattern**
  Design pattern to ***improve energy efficiency.***

Energy Patterns for Mobile



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

Methodolgy

Methodolgy

1. App Collection

F-droid

Curated Lists

1783 apps

Methodolgy

1. App Collection

F-droid   Curated Lists

1783 apps

↓

2. Collect Changes With Potential Interest

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

6028 changes

Methodolgy

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

Methodolgy

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

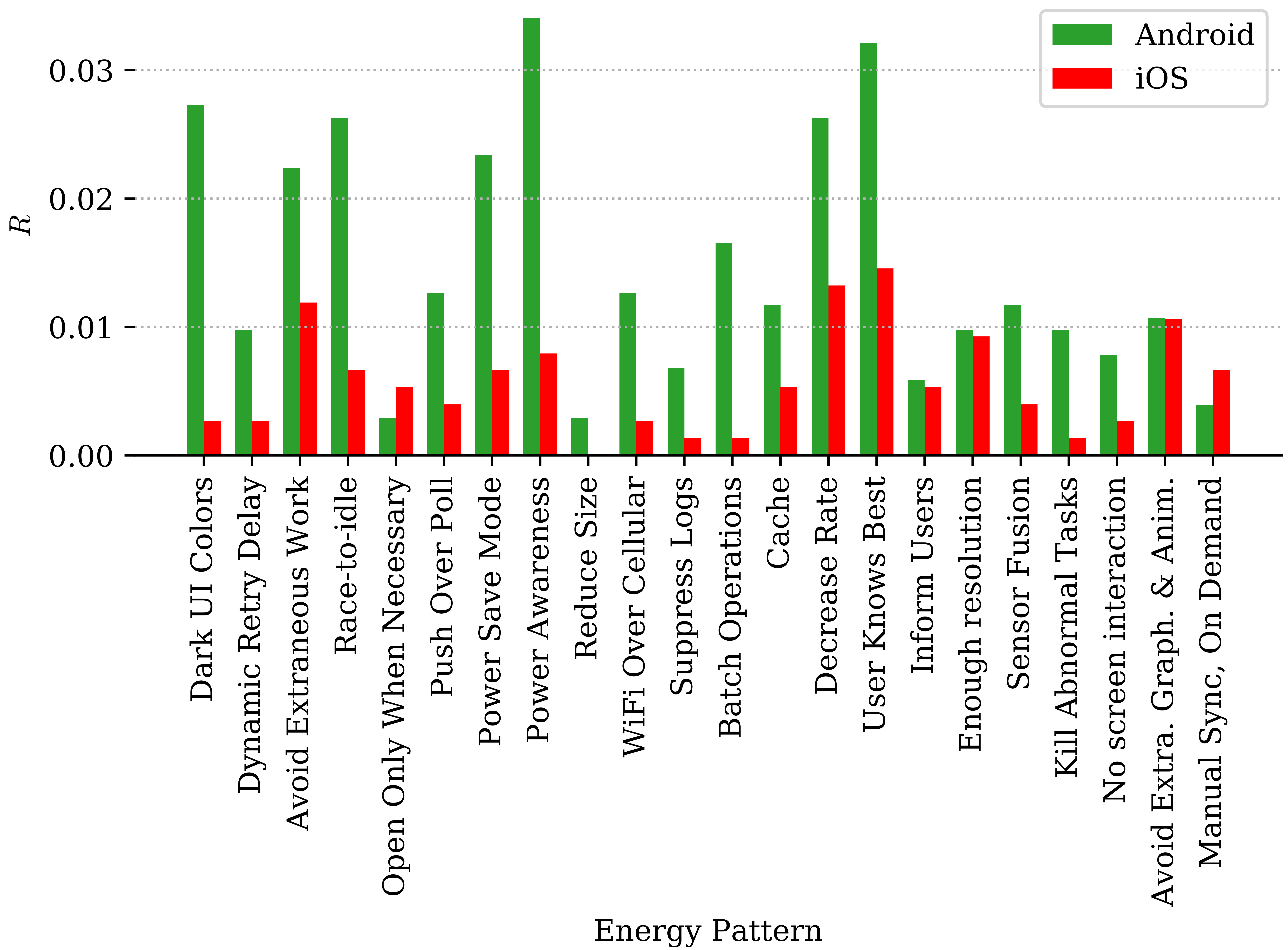
   1563 changes

4. Thematic Analysis

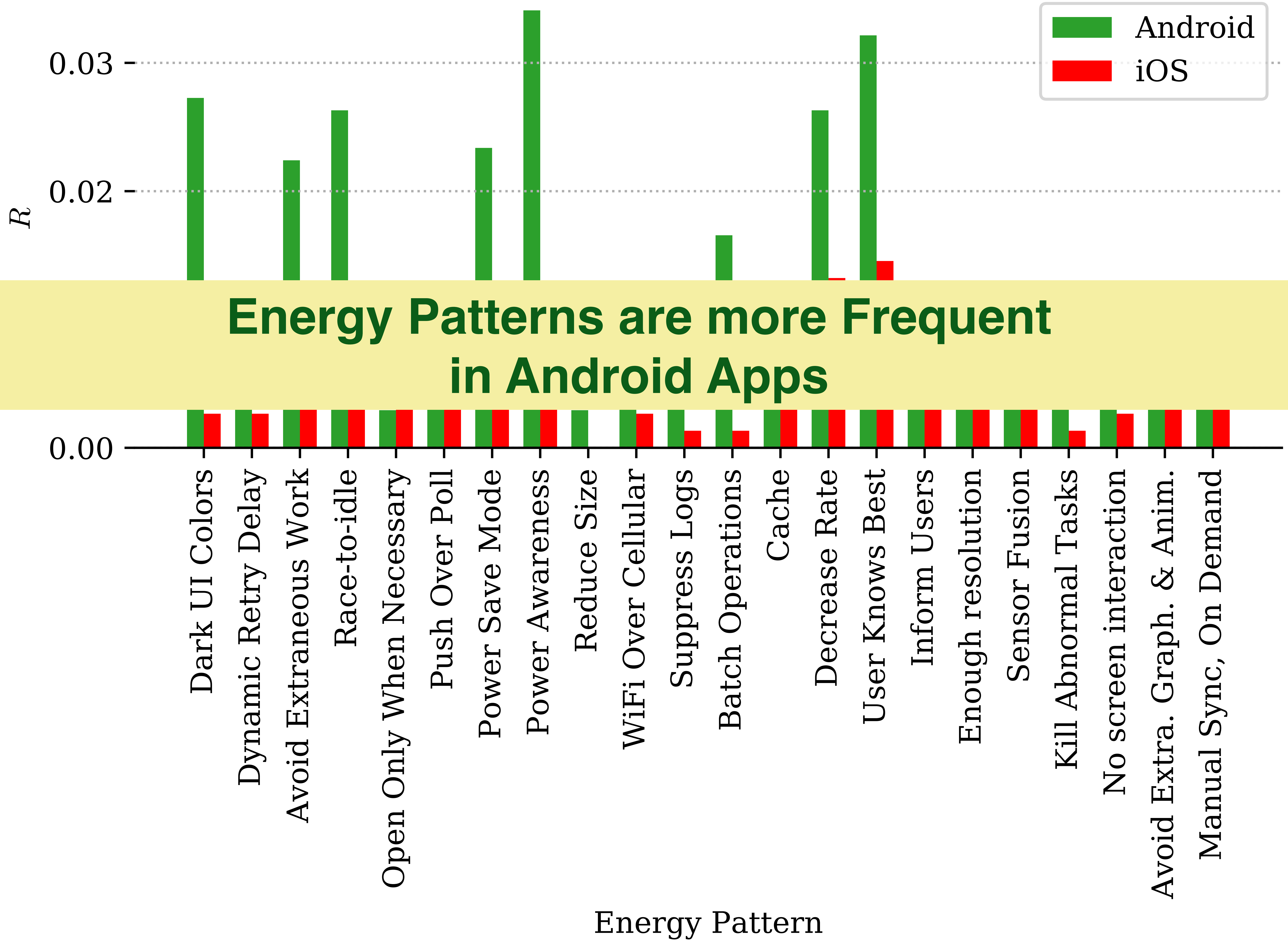
   431 reusable changes

Methodolgy

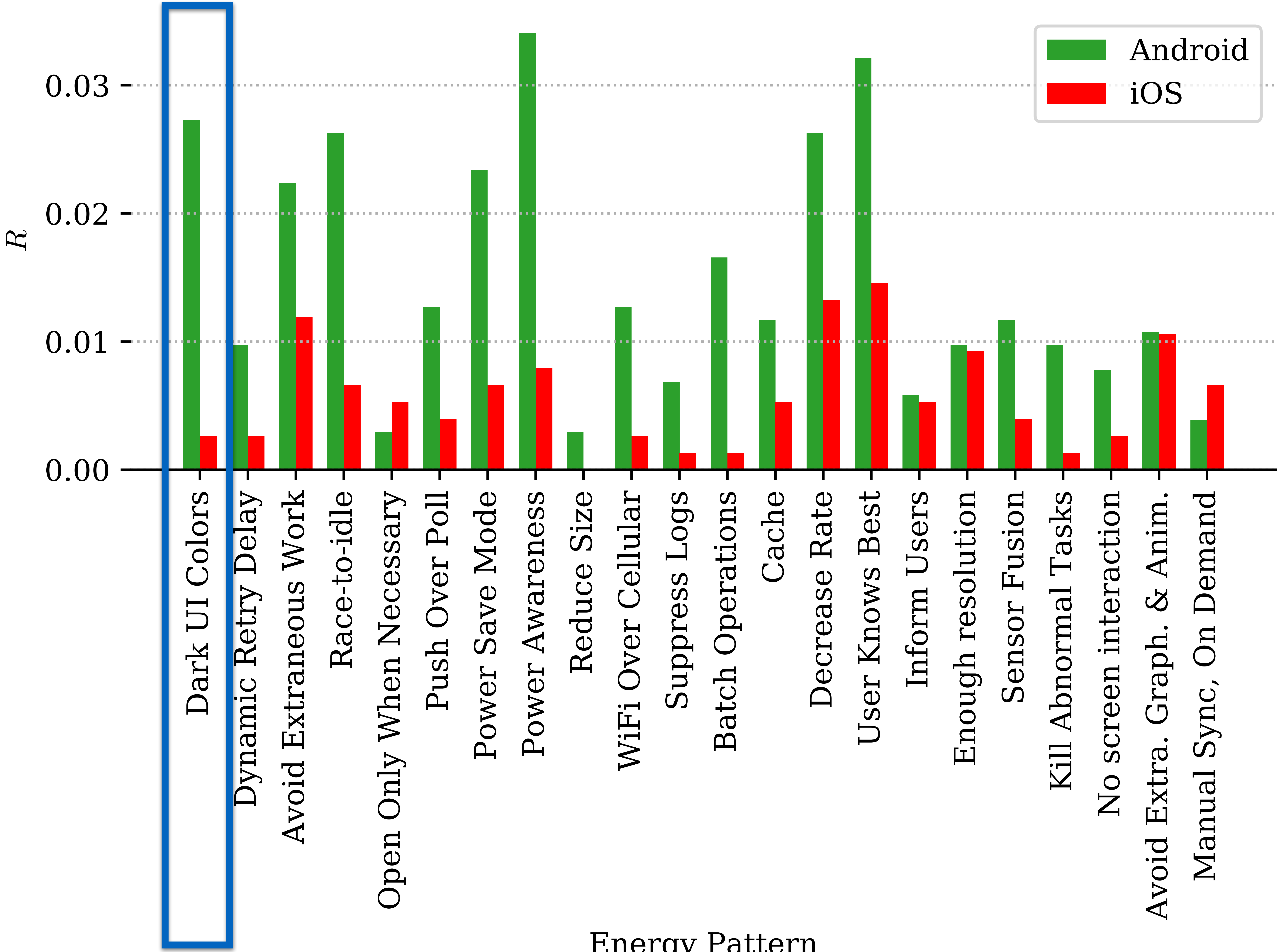
1. App Collection F-droid Curated Lists	1783 apps
2. Collect Changes With Potential Interest `/.*(energy|power|battery).*/`	6028 changes
3. Manual Refinement of Subjects of Interest	1563 changes
4. Thematic Analysis	431 reusable changes
5. Catalog of Energy Patterns	22 patterns

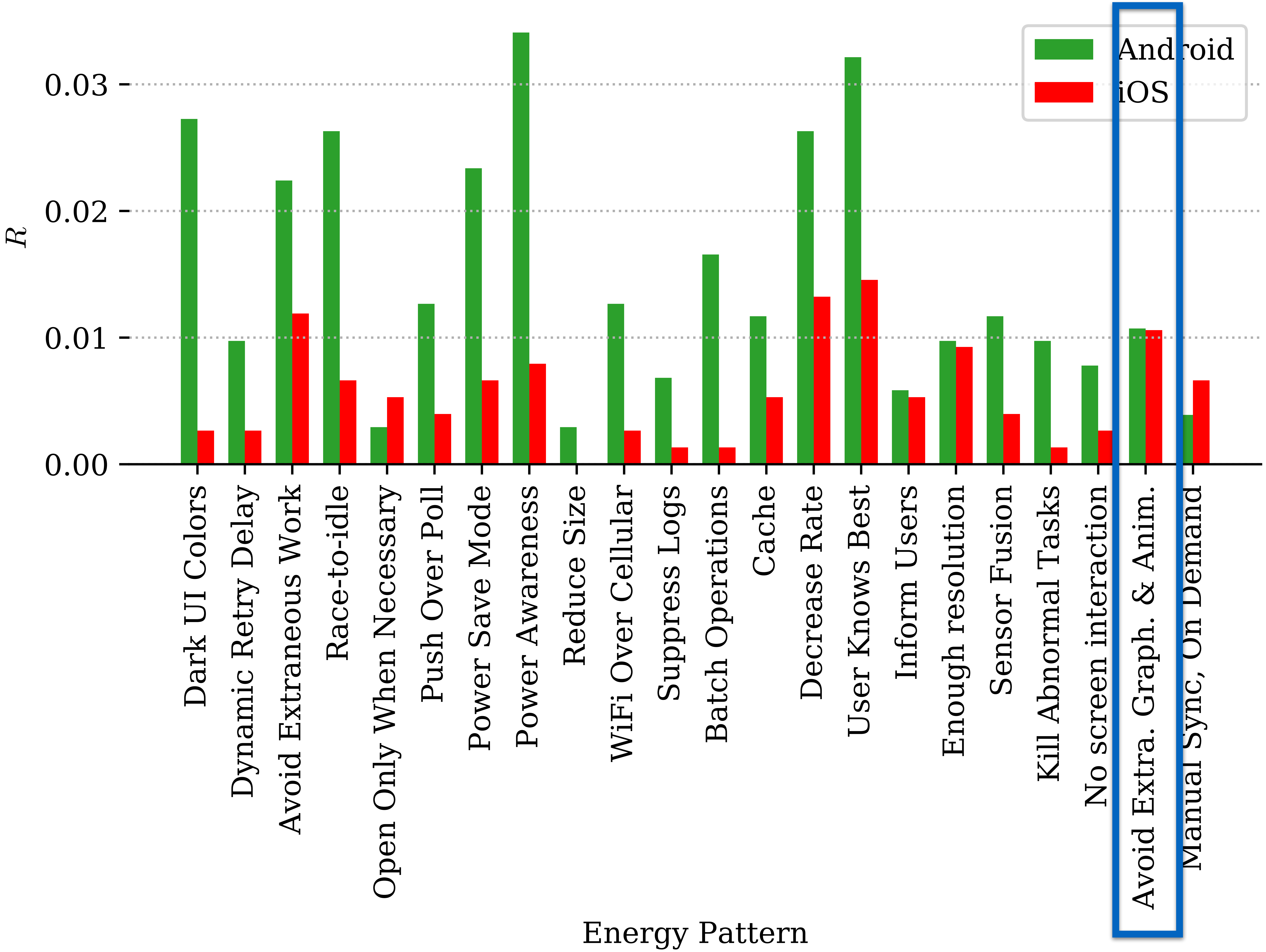


Android
iOS
R
0.03
0.02
0.01
0.00
Dark UI Colors
Dynamic Retry Delay
Avoid Extraneous Work
Race-to-idle
Open Only When Necessary
Push Over Poll
Power Save Mode
Power Awareness
Reduce Size
WiFi Over Cellular
Suppress Logs
Batch Operations
Cache
Decrease Rate
User Knows Best
Inform Users
Enough resolution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



Energy Patterns are more Frequent in Android Apps


Android
iOS
0.03
0.02
0.01
0.00
R
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



0.03
0.02
0.01
0.00
R
Android
iOS
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

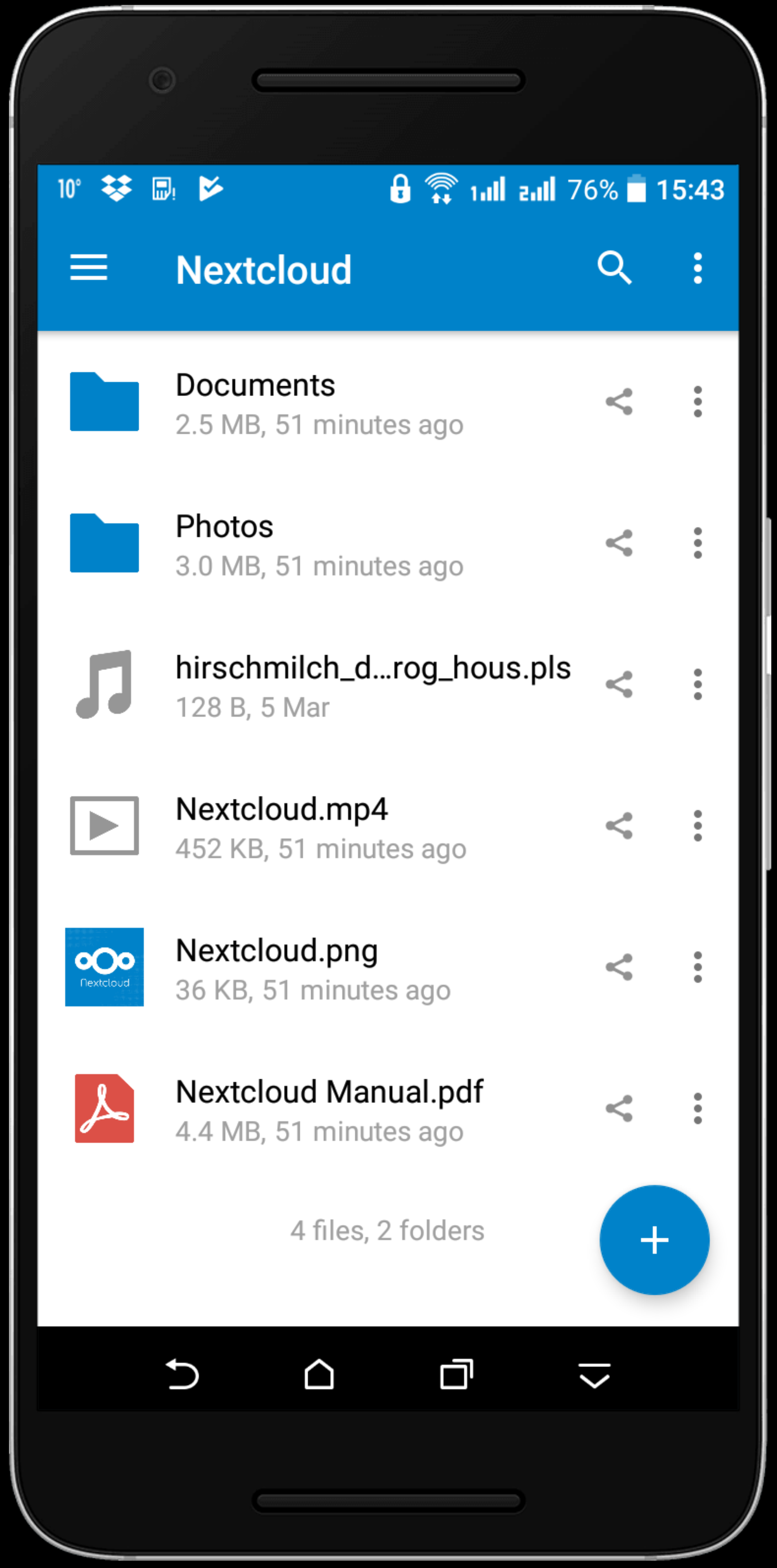
Example case: Nextcloud

Nextcloud

GET IT ON Google Play

GET IT ON F-Droid

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.

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.**

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.**

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.**

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.**

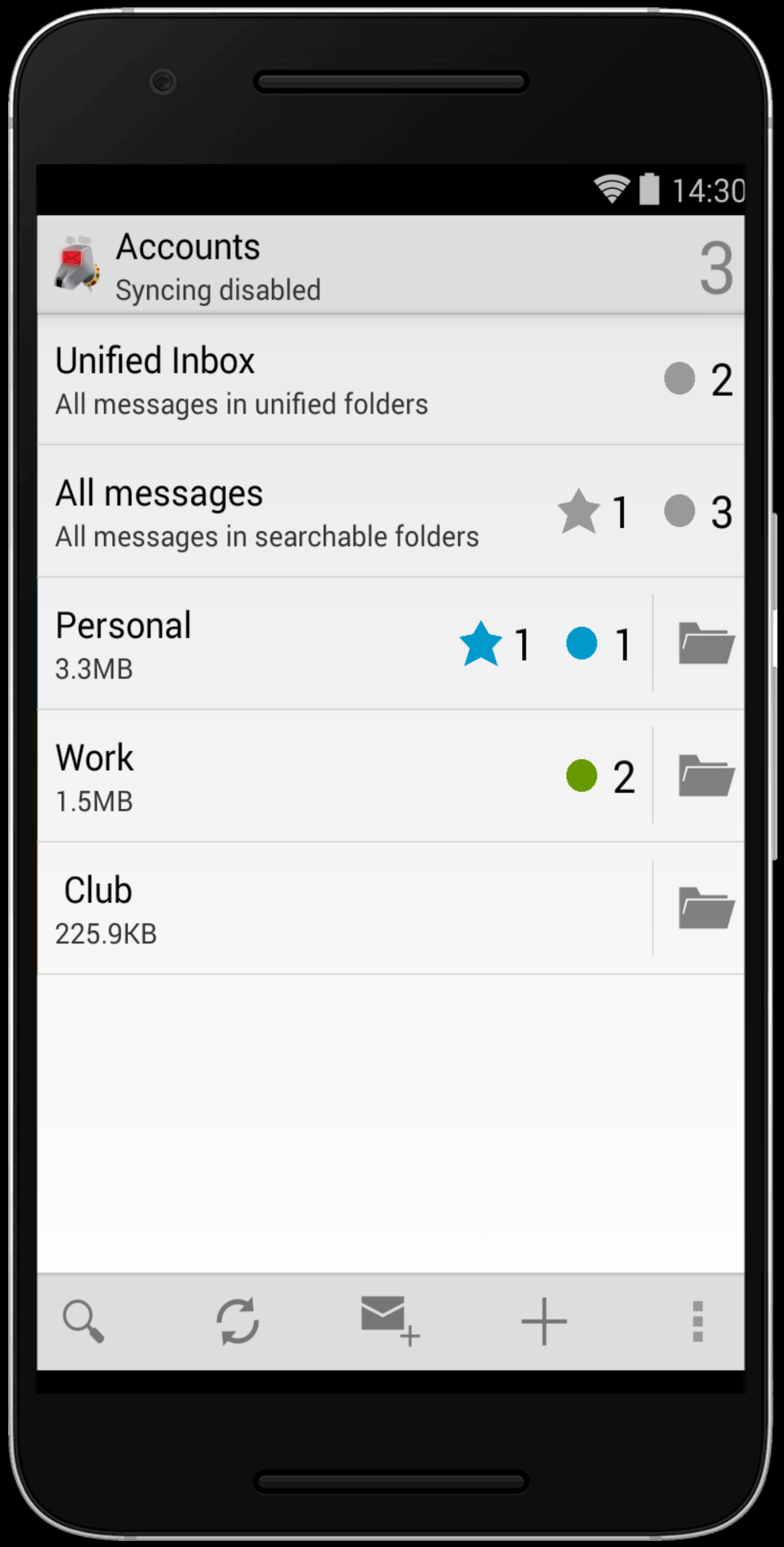
Solutions?

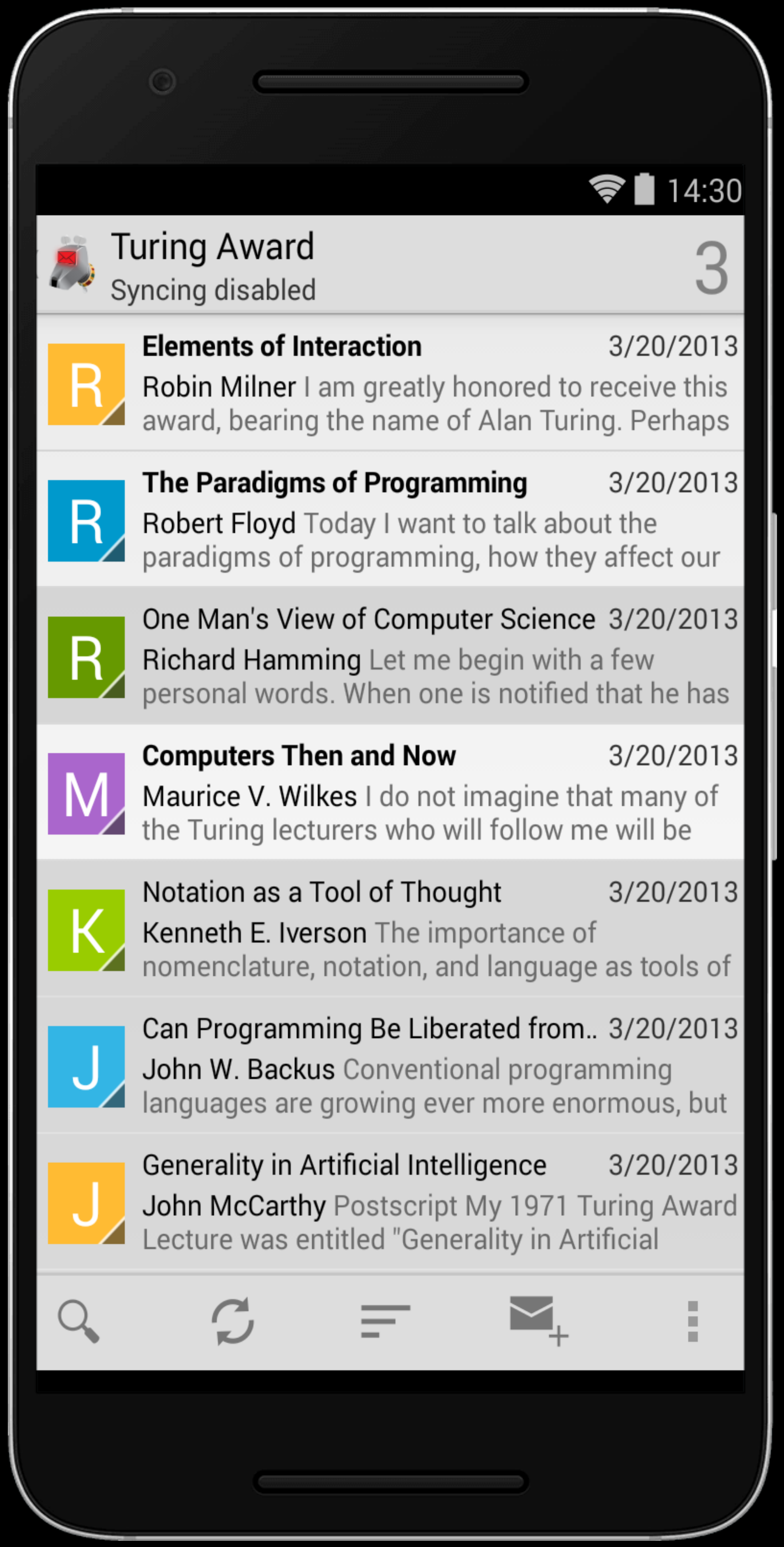
Example case: Nextcloud

- 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.

Example case: K-9 mail

- Some users noticed that K-9 mail was spending more energy than usual. 🙉

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 one of the most energy-greedy apps. **IMAP IDLE protocol for real-time notifications.**

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 one of the most energy-greedy apps. **IMAP IDLE protocol for real-time notifications.**
- When connection is not possible the app automatically retries later.

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 one of the most energy-greedy apps. **IMAP IDLE protocol for real-time notifications.**
- When connection is not possible the app automatically retries later.
- https://github.com/k9mail/k-9/commit/86f3b28f79509d1a4d613eb39f60603e08579ea3

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 one of the most energy-greedy apps. **IMAP IDLE protocol for real-time notifications.**

- When connection is not possible the app automatically retries later.

- https://github.com/k9mail/k-9/commit/86f3b28f79509d1a4d613eb39f60603e08579ea3

Solutions?

Which programming languages are most energy efficient?

Energy Efficiency across Programming Languages

How Do Energy, Time, and Memory Relate?

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

Marco Couto
HASLab/INESC TEC
Universidade do Minho, Portugal
marco.l.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
jacome@fct.unl.pt

João Paulo Fernandes
Release/LISP, CISUC
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 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 [2, 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 $E_{nergy} = T_{ime} \times P_{ower}$. In fact, there are several research works showing different results regarding

SLE'17, October 23–24, 2017, Vancouver, Canada

ACM ISBN 978-1-4503-5525-4/17/10…$15.00
https://doi.org/10.1145/3136014.3136031

256

(Pereira, 2017)

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

The Computer Language Benchmarks Game

https://edu.nl/9fxcv

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

(Pereira, 2017)

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

(Pereira, 2017)

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

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

(Pereira, 2017)

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

Checkpoint 3

To measure the pattern Dark Color UI, I would create two UIs themes for the same app and I would generate automated user interaction scripts that would work for 10 minutes. These scripts would have to run in the exact same way throughout multiple executions. I would use an energy profiler that would collect energy data during the experiment and I would compare the data from the two UIs.
Luís Cruz

👍 1

edu.nl/8b639

Checkpoint 3

- Check the Catalog of Energy Patterns for Mobile Apps. https://tqrg.github.io/energy-patterns/

To measure the pattern Dark Color UI, I would create two UIs themes for the same app and I would generate automated user interaction scripts that would work for 10 minutes. These scripts would have to run in the exact same way throughout multiple executions. I would use an energy profiler that would collect energy data during the experiment and I would compare the data from the two UIs.

Luís Cruz

👍 1

edu.nl/8b639

Checkpoint 3

- Check the Catalog of Energy Patterns for Mobile Apps. https://tqrg.github.io/energy-patterns/
- Choose one energy pattern and do one of the following exercises (max. 500 chars):

To measure the pattern Dark Color UI, I would create two UIs themes for the same app and I would generate automated user interaction scripts that would work for 10 minutes. These scripts would have to run in the exact same way throughout multiple executions. I would use an energy profiler that would collect energy data during the experiment and I would compare the data from the two UIs.
Luís Cruz

1

edu.nl/8b639

Checkpoint 3

- Check the Catalog of Energy Patterns for Mobile Apps. https://tqrg.github.io/energy-patterns/
- Choose one energy pattern and do one of the following exercises (max. 500 chars):
    - a) Explain how the pattern would help in one of your previous projects. Sticky note in yellow (preferably).

To measure the pattern Dark Color UI, I would create two UIs themes for the same app and I would generate automated user interaction scripts that would work for 10 minutes. These scripts would have to run in the exact same way throughout multiple executions. I would use an energy profiler that would collect energy data during the experiment and I would compare the data from the two UIs.
Luís Cruz

👍 1

edu.nl/8b639

Checkpoint 3

- Check the Catalog of Energy Patterns for Mobile Apps. https://tqrg.github.io/energy-patterns/
- Choose one energy pattern and do one of the following exercises (max. 500 chars):
  - a) Explain how the pattern would help in one of your previous projects. Sticky note in yellow (preferably).
  - b) Explain how would you measure the energy-improvement from a particular pattern. Sticky note in green (preferably).

To measure the pattern Dark Color UI, I would create two UIs themes for the same app and I would generate automated user interaction scripts that would work for 10 minutes. These scripts would have to run in the exact same way throughout multiple executions. I would use an energy profiler that would collect energy data during the experiment and I would compare the data from the two UIs.
Luís Cruz

👍 1

edu.nl/8b639

Checkpoint 3

- Check the Catalog of Energy Patterns for Mobile Apps. https://tqrg.github.io/energy-patterns/
- Choose one energy pattern and do one of the following exercises (max. 500 chars):
  - a) Explain how the pattern would help in one of your previous projects. Sticky note in yellow (preferably).
  - b) Explain how would you measure the energy-improvement from a particular pattern. Sticky note in green (preferably).
- Read some of your colleague's answers and upvote your favourite with the emoji 👍.

To measure the pattern Dark Color UI, I would create two UIs themes for the same app and I would generate automated user interaction scripts that would work for 10 minutes. These scripts would have to run in the exact same way throughout multiple executions. I would use an energy profiler that would collect energy data during the experiment and I would compare the data from the two UIs.

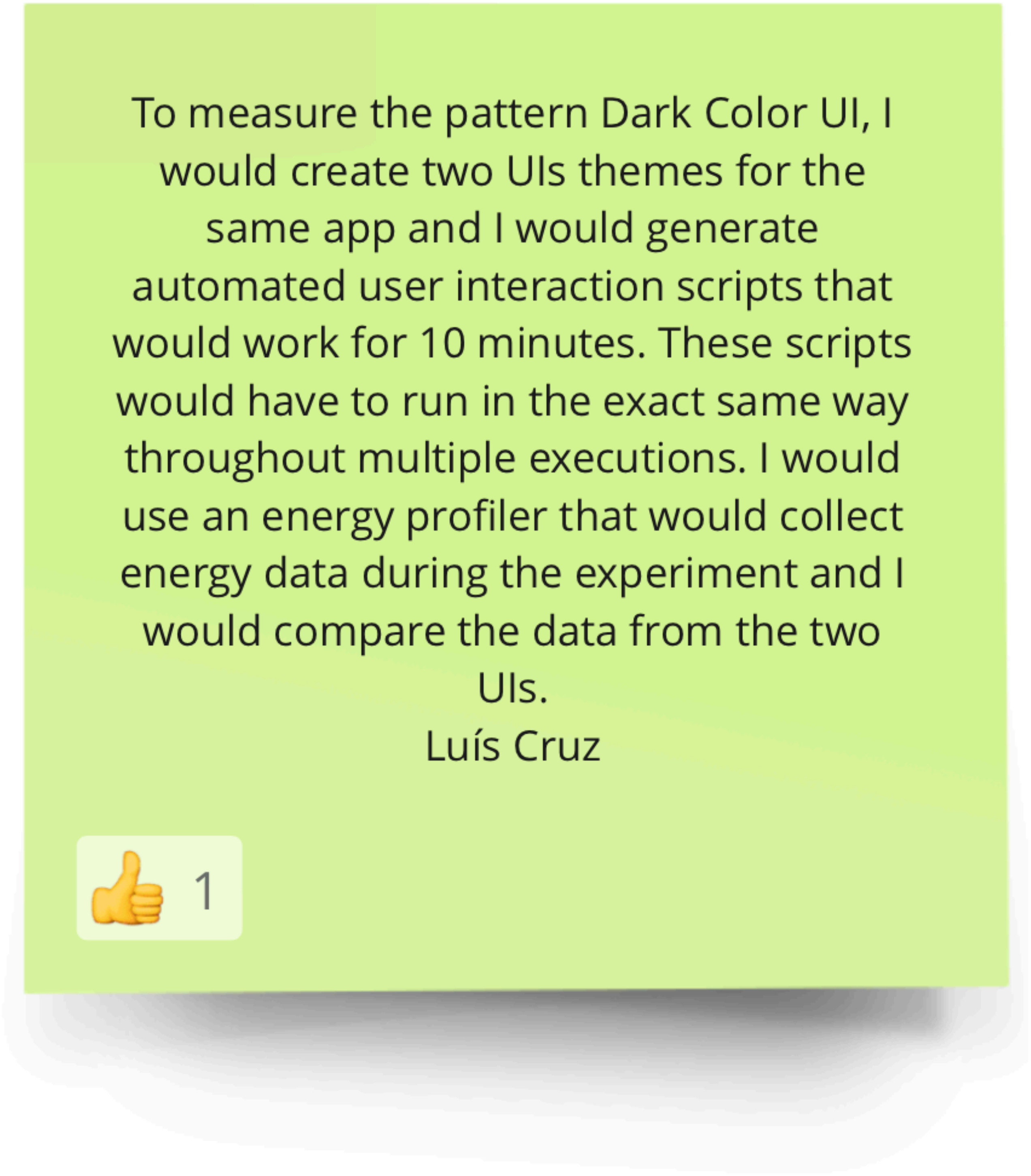
Luís Cruz

👍 1

edu.nl/8b639

Checkpoint 3

- Check the Catalog of Energy Patterns for Mobile Apps. https://tqrg.github.io/energy-patterns/
- Choose one energy pattern and do one of the following exercises (max. 500 chars):
  - a) Explain how the pattern would help in one of your previous projects. Sticky note in yellow (preferably).
  - b) Explain how would you measure the energy-improvement from a particular pattern. Sticky note in green (preferably).
- Read some of your colleague's answers and upvote your favourite with the emoji 👍.
- Miro board: https://edu.nl/8b639



edu.nl/8b639

Green Open Field

- Several research opportunities. **Growing niche.**
- Integrate systems with **energy consumption feedback.** (**Inform users**)
- Energy-efficiency at different levels. **API, programming language, IDE, user, developer, etc.**
- Impact of different architectures. **Controlling for implementation, hardware and feature set is not trivial.**
- Domain-specific energy patterns. **So far, only mobile.**
- Green AI, E-waste, Green deFi, Green Mobile Computing…
- Many initiatives are emerging to address Sustainable IT: ClimateAction.tech; #LetsGreenTheWeb, TheGreenWebFoundation, #11at11, etc.

Assignment

- **A)** Analyze the change history of the project and **find code changes** that are related to green computing. Present and discuss the rationale behind those changes.
- **B)** Recommend **energy improvements** to be implemented in the project (development, source, infrastructure). Implement them, if possible.
- **C) Measure** the energy consumption of potential **hotspots**. (Using an energy profiler)

  Output: Two-page essay with all the rationale behind the study

- Critical thinking is a big plus. A few things to help:
  - Is it always possible to reduce energy consumption?
  - What are the trade-offs of improving energy efficiency?
  - What are the implications on UX or business metrics?
  - Would automation tools help?
  - What is missing in the project to improve energy efficiency?

Wrap-up

- What is Sustainable Software
- What is Green Software?
- How can we measure energy consumption?
- What are the sources of energy consumption in software engineering?
- What is an energy pattern?
- What are the common trade-offs when improving energy efficiency?

Architecting for Sustainability

Software Architecture (IN4315)

@luismcruz

L.Cruz@tudelft.nl

https://luiscruz.github.io/