Automated measurement and green optimizations for software

SCI, Blue Angel for Software, CI / CD

GREEN CODING;

Who are we

Your workshop facilitators

- Arne - Green Coding Solutions
- Didi - Green Coding Solutions
- Michelle - Green Coding Solutions
- Aydin - bluehands

Our entry question

What is the goal of this workshop?

We want to make software sustainable

Through technical means - Measurement, Automation, Optimization

- **In this workshop we want to**
  - share successful **experiences and techniques**
  - share current **struggles** and **hurdles**
  - brainstorm new **ideas** and **approaches**
- **To get the workshop started we will present**
  - Our approach for an **open source automated measurement system**
  - Current possible routes for **certificates** / **standards**

We want to make software sustainable

Our definition of a sustainable software and our approach

- A sustainable software knows about it's **energy** and **CO2 emission**
- A sustainable software is constantly **monitored** and **optimized** for savings
- A sustainable software uses the **least amount of resources** for a given task (compared with similar applications)
- A sustainable software is lean and does **combat software bloat**

- **=> Be sure to bring your view of a "sustainable software" in later!**

Our follow-up question

What technical parts do we need to get to a sustainable software?

Let's start with some requirements

What are our goals and what do we need to achieve them

- **Measurement**: We want to look at software and determine it's energy and CO2 consumption
- **Reproducibility**: We want to quantify changes and thus runs must be reproducible and have low StdDev.
- **Optimization**: We want to make changes and compare their effect
- **DevOps / CI/CD integration**: We want to look at changes over time as incremental changes might be negligable
- **Standards**: We need standards that the measurements output to execute on all of the above - Actually ... this is the first!

Stage #1

Standards

Standards - Part #1: SCI

Green Software Foundation's SCI

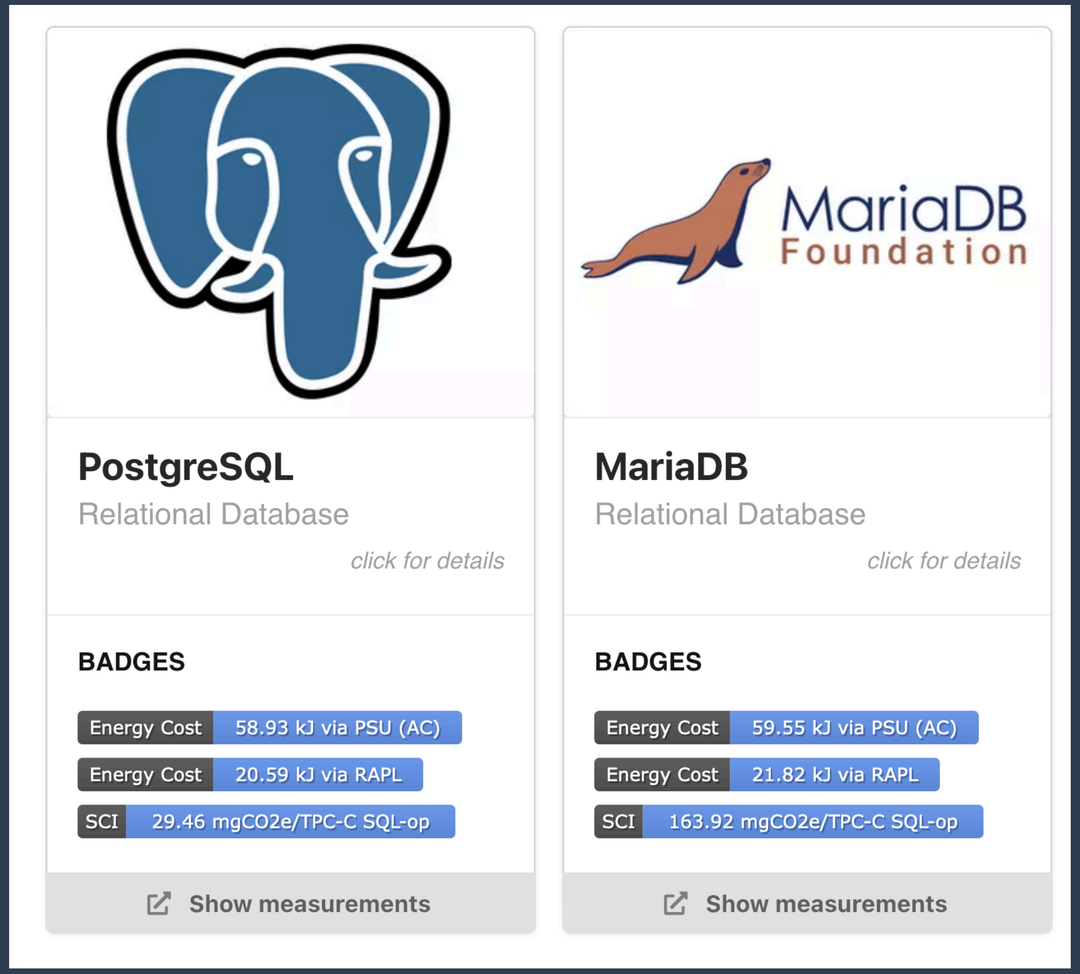
SCI = (E * I) + M per R

- (E) - Energy consumption (kilowatt hours) for different components:
  - Ex. CPU/GPUs, Data storage, Memory, Network
- (I) - Emissions factors
- (M) - Embodied emissions
  - Ex. data for servers, mobile devices and laptops
- (R) - Unit of work / use case

Standards - Part #1: SCI

Green Software Foundation's SCI

- By using standardised or at least identical Benchmarks software becomes comparable through the SCI
- What we want is typically multiple SCIs for one software
- Example **Grafana**
  - Case #1: Importing data
  - Case #2: Displaying 5 graphs
  - Case #3: Making aggregation



Standards - Part #2: Blue Angel

Blue Angel for Software by Umweltbundesamt

SCI = (E * I) + M per R

- (E) - Energy consumption (kilowatt hours) for different components:
  - Ex. CPU/GPUs, Data storage, Memory, Network
- (I) - Emissions factors
- (M) - Embodied emissions
  - Ex. data for servers, mobile devices and laptops
- (R) - Unit of work / use case

Standards - Part #2: Blue Angel for Software

Ressource- and energy-efficient software products (DE-UZ 215)

- **Goals:**
  - Reduce hardware obsolescence
  - Increase transparency about software energy consumption
  - Increase user autonomy
  - Increase modularity and software re-use

Standards - Part #2: Blue Angel for Software

Ressource- and energy-efficient software products (DE-UZ 215)

- **Requirements:**
  - Specification of intended use-case(s)
  - Measurement incl. utilization, energy etc.
  - Formal declaration of no advertisments, security updates, etc.
  - Must run on older hardware
    - Reference system or
    - OS Version compatibility

Good for me.
Good for the environment.

Umwelt Bundesamt

Standards - Part #3: ISO 14001 / GHG Protocol

Environmental management

- ISO 14000 familiy is focused on life cycle assessment
- ISO 14067 provides framework of looking into software as phases
- GHG protocol provides the execution standard
- Result is a life-cycle-assessment for a digital product. Methodology has international accreditation
- => Talk also to our speaker Moritz Bölter for experiences!

Stage #2

Infrastructure reproducibility

Reproducability - Orchestration

Using infrastructure files and git versioning

- Container Files
  - Docker Files
  - Kubernetes Files
  - Impact Framework
- Versioning of all resources and files integrated with Github repository

compose.yml

```yaml
services:
  gcb-wordpress-apache:
    build:
      context: .
      dockerfile: Dockerfile-wordpress
    container_name: gcb-wordpress-apache
    image: gcb_wordpress_apache
    ports:
      - 9875:9875
    restart: always
    environment:
      - WORDPRESS_DB_HOST=gcb-wordpress-mariadb
    depends_on:
      - gcb-wordpress-mariadb
    ...
```

Reproducability

Orchestrating - Tricky topic!

- Orchestrating docker containers
- Orchestrating Kubernetes k3s for smaller setups in deeper docker container
- Orchestrating full K8s with network plane in internal cluster (Not implemented!)
  - => We have seen this very unuseful for actual software optimizations. It is nice for reporting. But not for optimizations as normal observability does give no insights about actions on the system
  - => Share your insights!
- How to handle external networking? Mirror resources locally? Spoof?

Reproducability

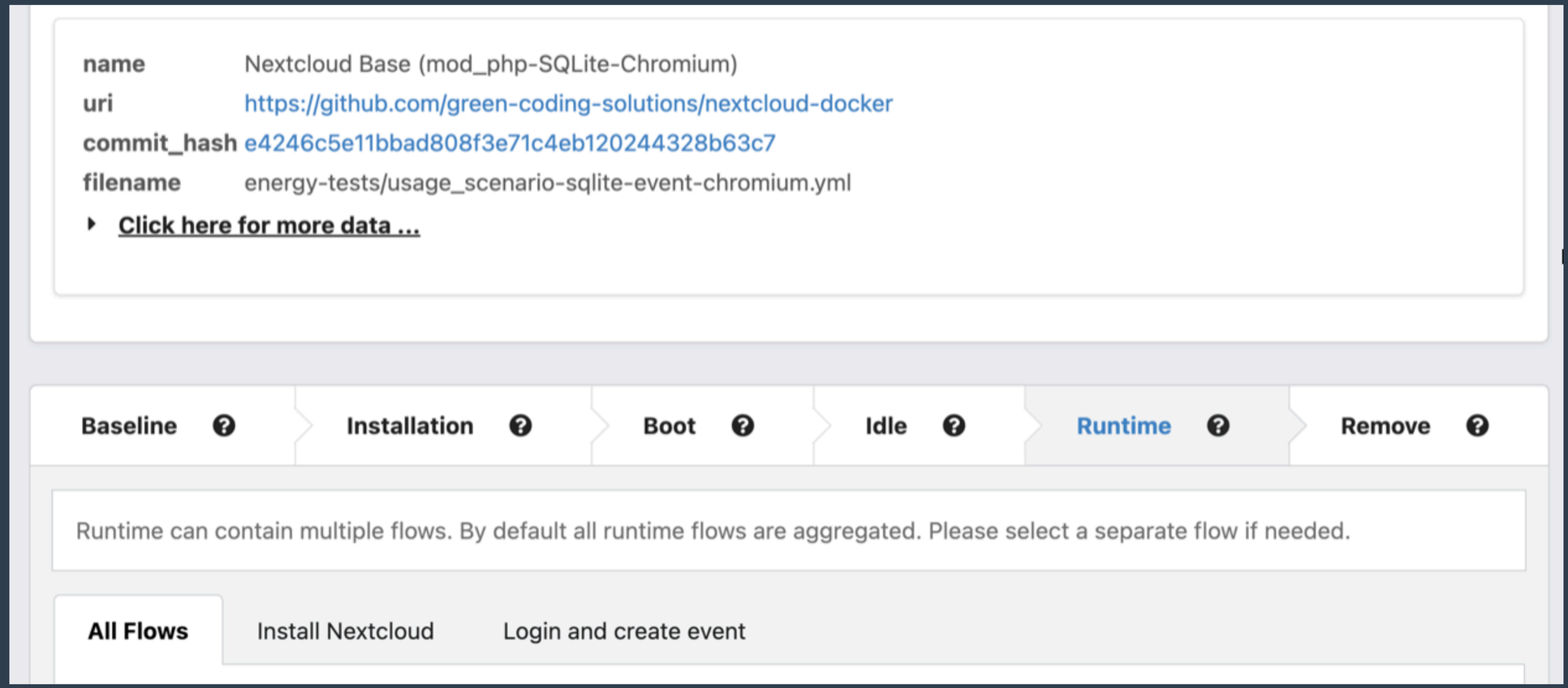
Alternatives to orchestration - System auto discovery

- We have seen this from softwares like Kepler
- We call it: **Monitor mode**
  - Difference to Kepler: Triggers to start / end a measurement frame
  - This allows to have reproducible measurements even in live systems

- => Do you think of any other relevant resources for modern systems? Put it on a note later!

Reproducability - Orchestration

Going through all phases of the Blue Angel for Software

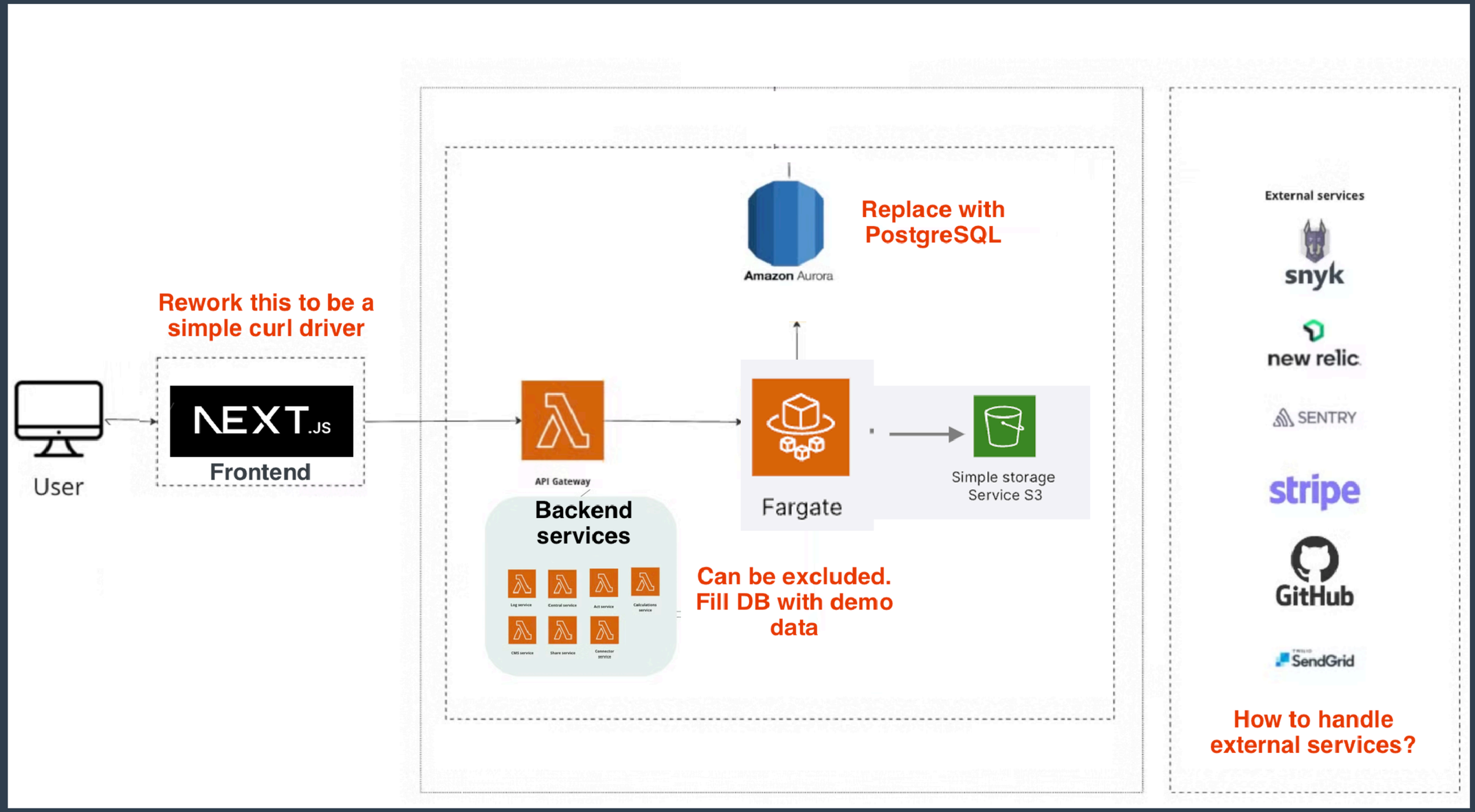


Thinking of software in phases. It adheres to upcoming standards AND It will come in handy later!

Orchestration - Open problems

When looking at cloud architectures for instance

- Keep in mind that optimization can only happen in domains you control or can benchmark
- Even cloud applications can be effectively optimized with slight changes.
- Replace with equivalents (has bias, but will behave comparable)
- Be mindful of what needs to part for minimal use case to keep system error influence low.
- Example ->


Rework this to be a simple curl driver
User
NEXT.JS
Frontend
API Gateway
Backend services
Amazon Aurora
Replace with PostgreSQL
Fargate
Simple storage Service S3
Can be excluded. Fill DB with demo data
External services
snyk
new relic
SENTRY
stripe
GitHub
SendGrid
How to handle external services?

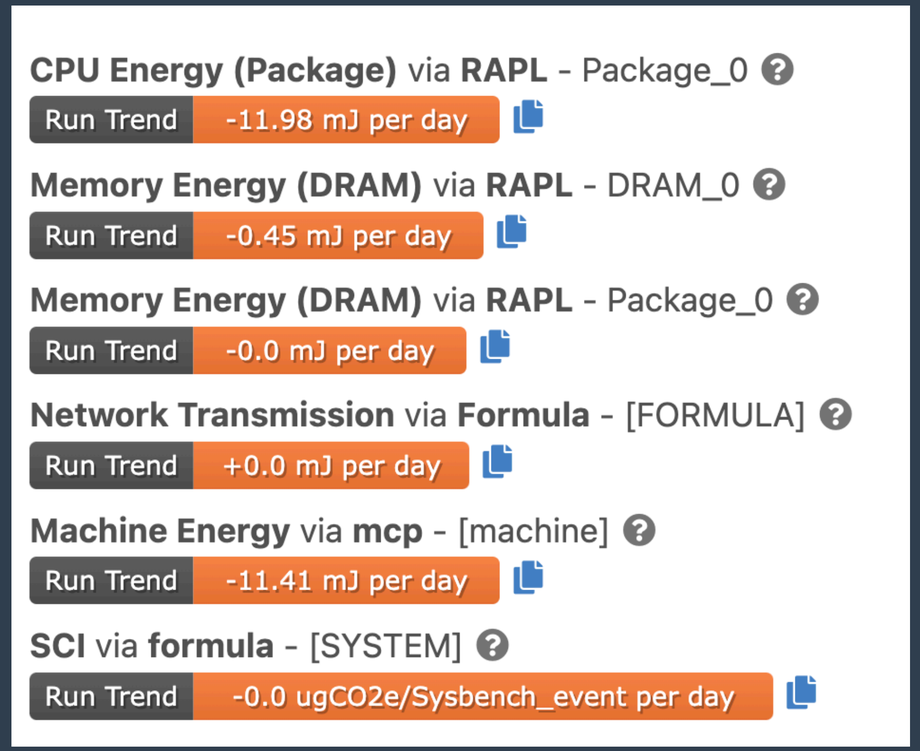
Stage #3

Measurement

Measurement

There is so much to measure. How to select?

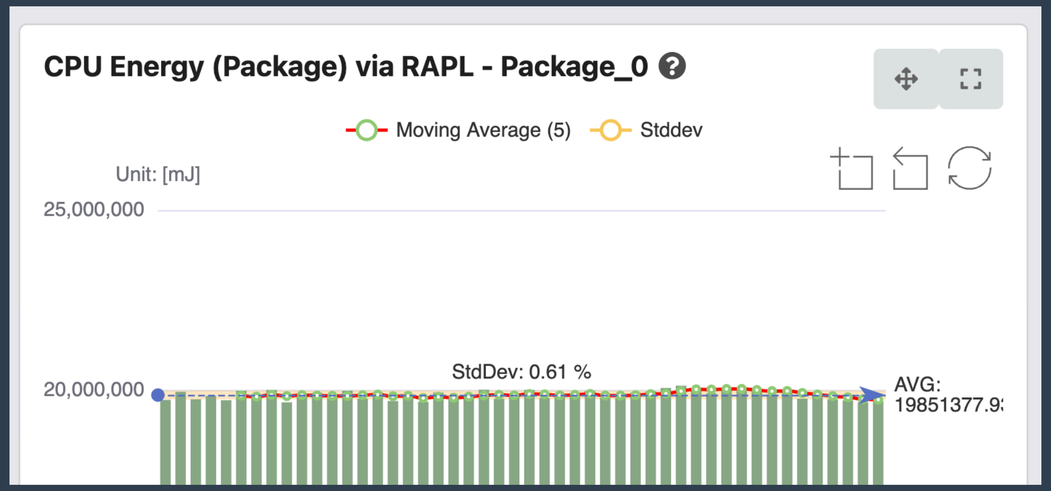
- We opted for a modular reporter architecture with sane defaults



Measurements - Reproducability

Maintaining measurement accuracy

- Temperature Control, Resource utilization control
- Off-the shelf OS with timers removed



Control workloads every 6h - StdDev < 1% between measurements

```
Running System Checks
Checking db online                                : OK
Checking single energy scope machine provider : OK
Checking tmpfs mount                              : OK
Checking < 5% CPU utilization                     : WARN (Your system
 seems to be busy. Utilization is above 5%. Consider terminating
some processes for a more stable measurement.)
Checking single energy scope machine provider
Checking tmpfs mount for low filesystem latency : OK
Checking 1GB free hdd space                       : OK
Checking docker daemon                            : OK
Checking running containers                       : OK
Checking current temperature"                     : OK
```

Guard clauses and sanity checks

Measurement-Reproducability

Running test workloads and setting best practices

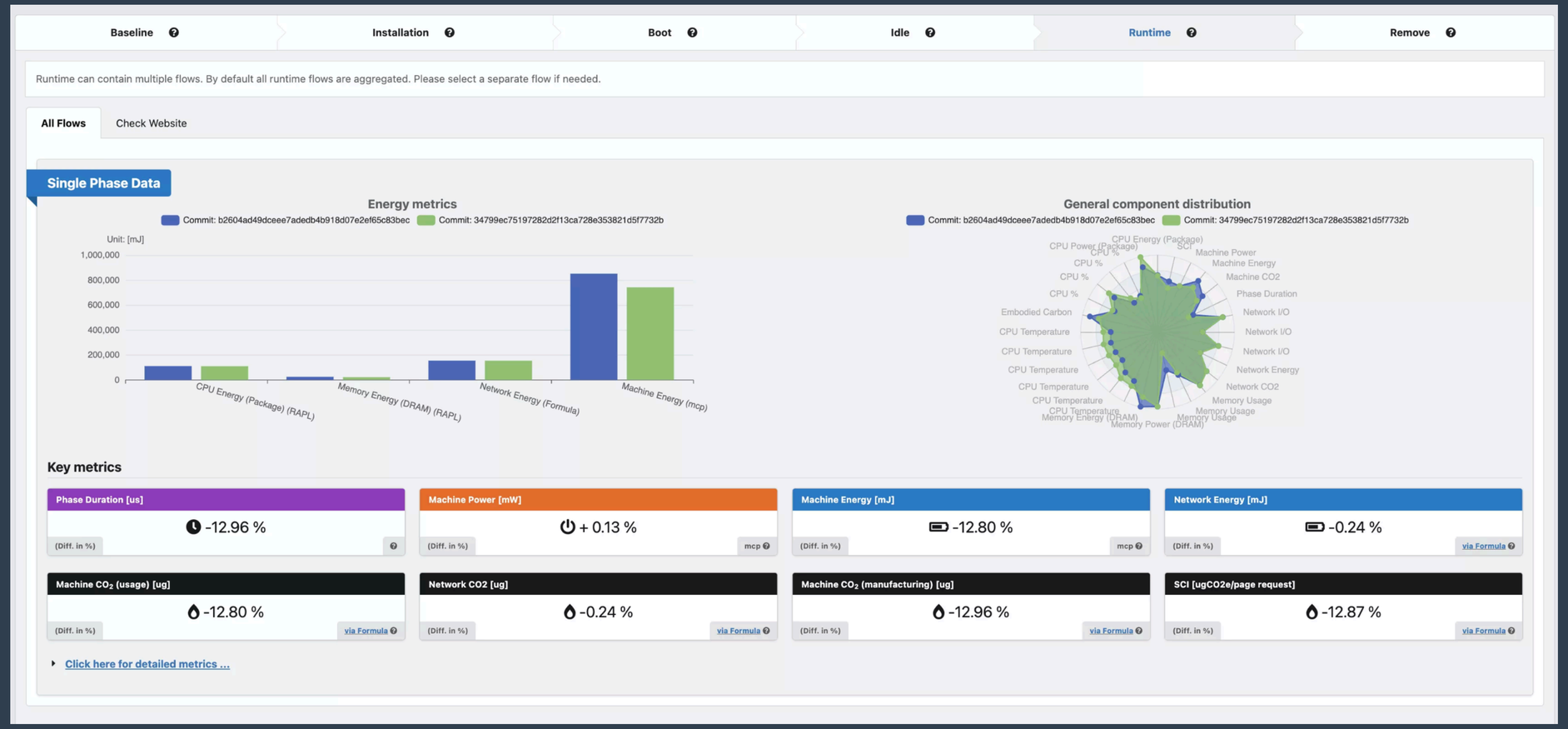
List of best practices

1. Never compare between machines to judge your software
2. An application should NEVER come to the bounds of its resources
3. The application you want to test must run at least twice as long as the minimal resolution
4. When running tests your disk load should not go over 50%
5. Limit amount and resolution of Metric Providers to what you absolutely need
6. Always check STDDEV
7. Design representative *Standard Usage Scenarios*
8. Pin your dependencies
9. Use temperature control and validate measurement std.dev.
10. Trigger test remotely or keep system inactive
11. Your system should not overheat
12. Mount your `/tmp` on `/tmpfs`
13. Turn logging off
14. Use `--docker-prune`
15. Use non standard sampling intervals and avoid undersampling
16. System Check Threshhold

- Not be read on this slide! But it is a long list and growing ... Contribute!
- https://docs.green-coding.io/docs/measuring/best-practices/

Measurements - Comparison

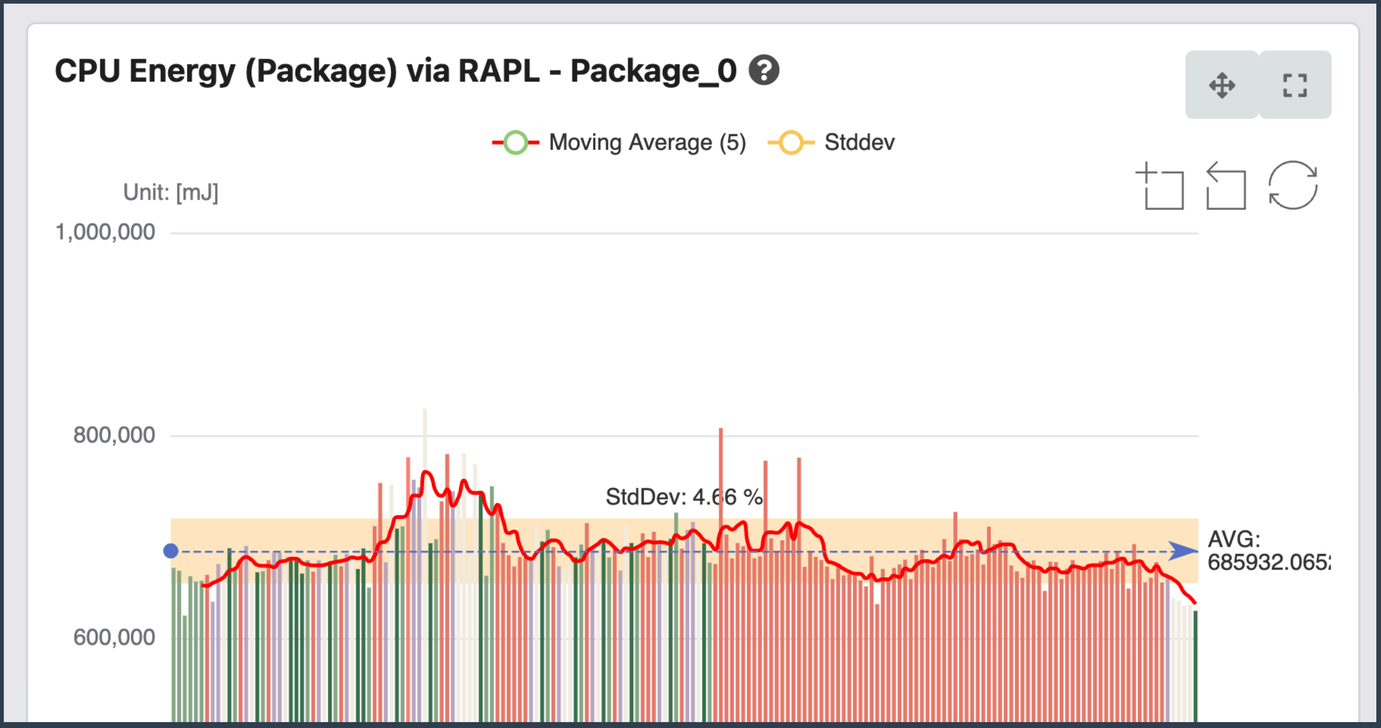
Comparing between Machines, Between repositories, Between Runs ...



GREEN CODING;

Measurements - Comparison

Behaviour of software over time



See changes over time with moving average

Showing regression trend

GREEN CODING;

Stage #4

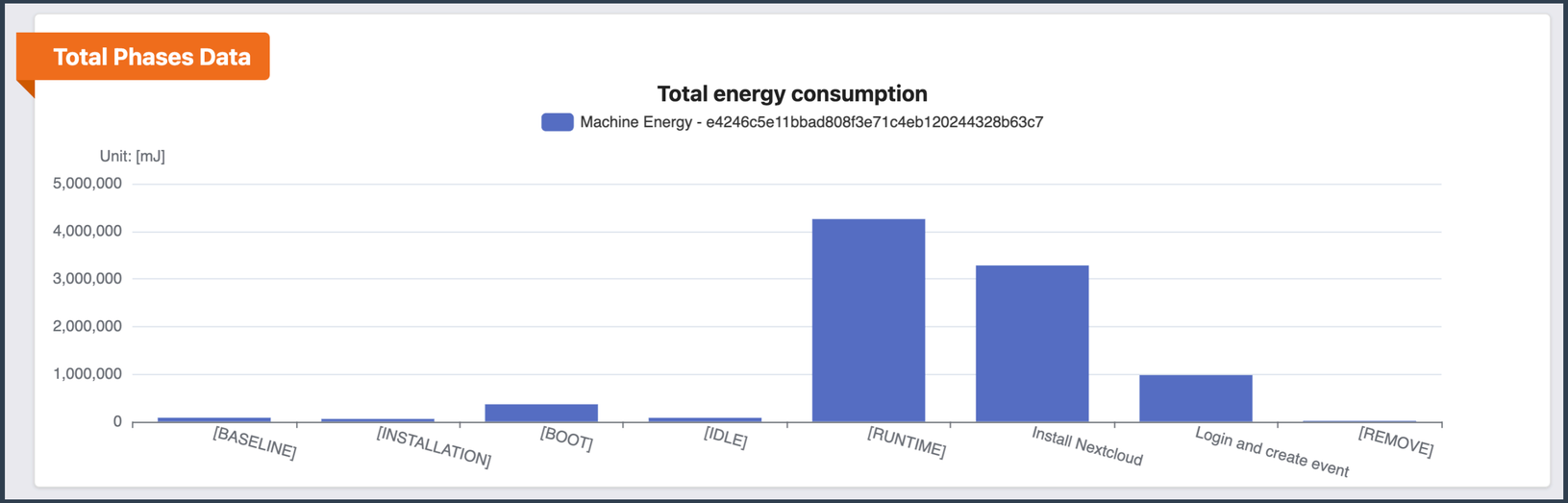
Optimizations

Optimizations

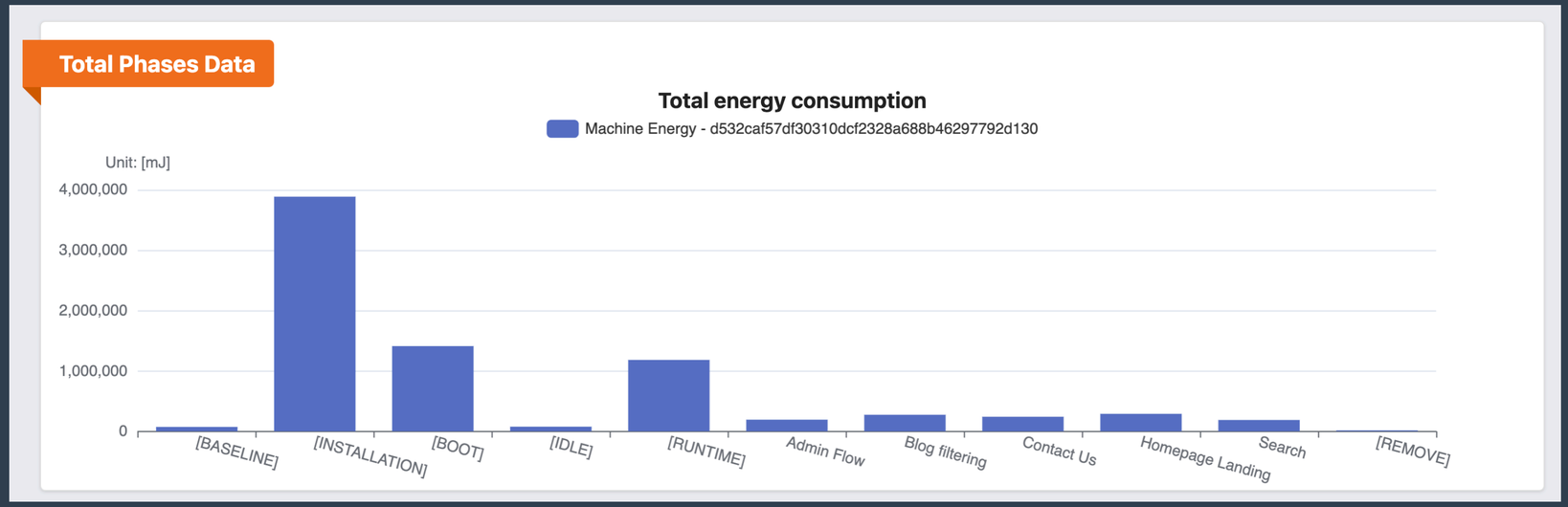
Split into four technical domains of optimizations

- **Part #1:** Algorithmical observability based optimizations
  - Using metrics that the OS already provides like CPU%, Memory, time etc.
- Part **#2:** Algorithmical profiling based optimizations
  - Using PMUs or deep profiling like IPC, Energy, access-times etc.
- **Part #3**: Changes over time
- **Part #4:** Targeted Code quality / review based optimizations through AI
  - Identifying code quality of specific parts of the application

Phases interplay already reveals a lot info



Boot and Build compared to Runtime is already low



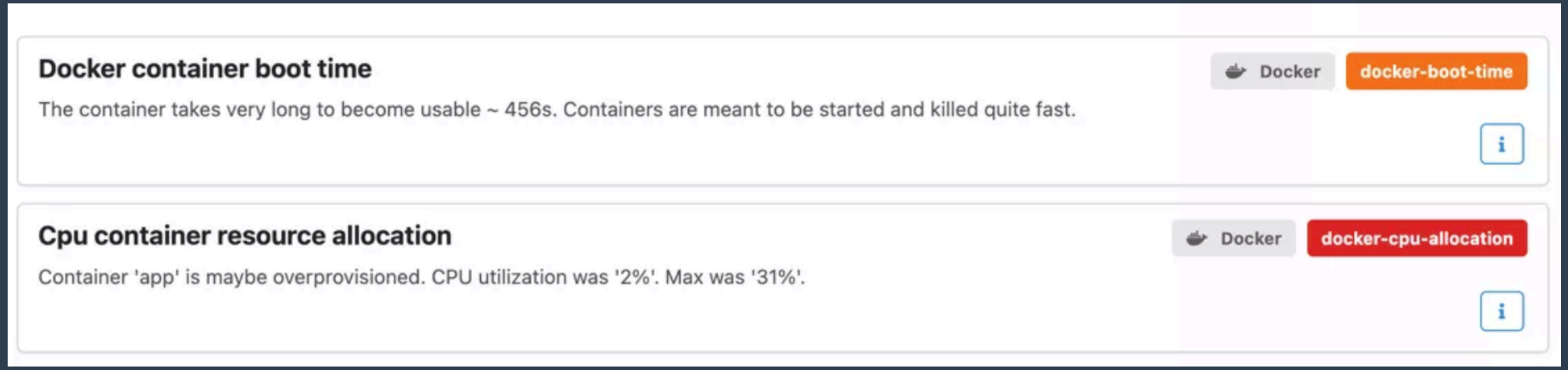
Boot compared to runtime very bad. Problematic in FAAS and CI/CD cases where caching is possible

GREEN CODING;

Optimizations - Part #1

Algorithmical observability based optimizations

- Once we orchestrate the application, we can tune it's resource consumption behaviour



Example from Green Metrics Tool for Container Boot and CPU resouce metrics

Optimizations - Part #1

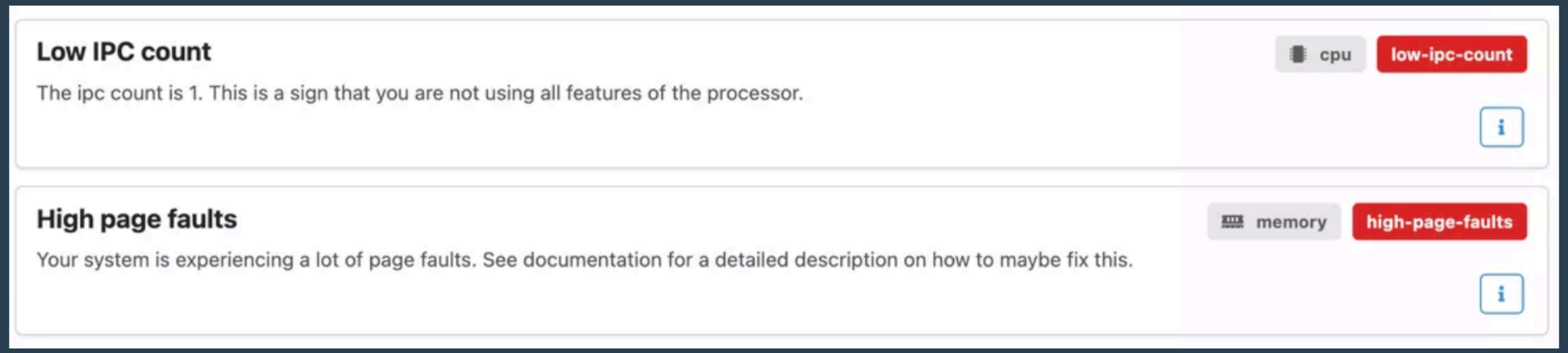
Algorithmical observability based optimizations

- We can determine over-provisioning by testing with lower limits and comparing results
- We can determine boot vs. runtime duration because we know an average request length (FAAS)
- We can look at image sizes, image layer architecture that breaks caching
- And many more from best practice lists like the Green Software Foundation, Digital Sustainability Center and many more ...

Optimizations - Part #2

Algorithmical profiling based optimizations

- We need to instrument an application by capturing tracepoints, patching the runtime, polling PMUs etc. to make this work.
- Also we must change characteristics of the OS and see how the application behaves

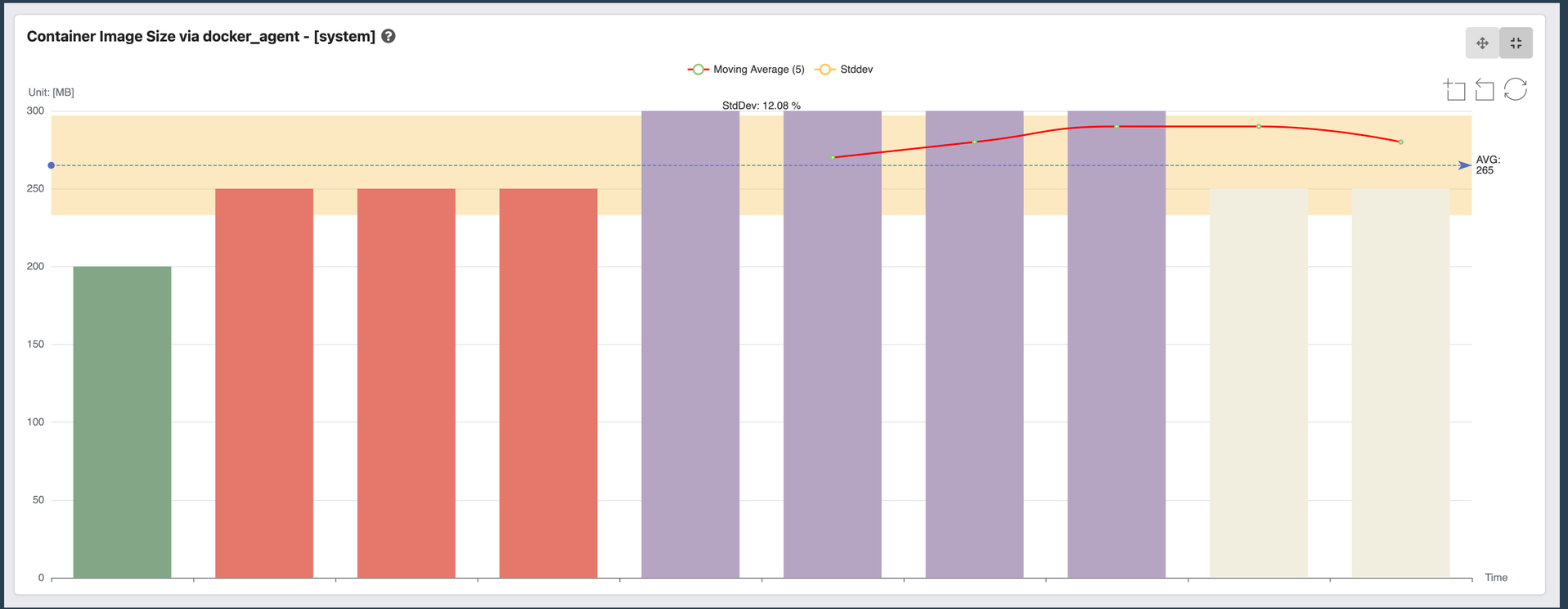


- But the benefits are high! In example: Utilizing processor in full can bring 8x improvement

Optimizations - Part #3

Changes over time

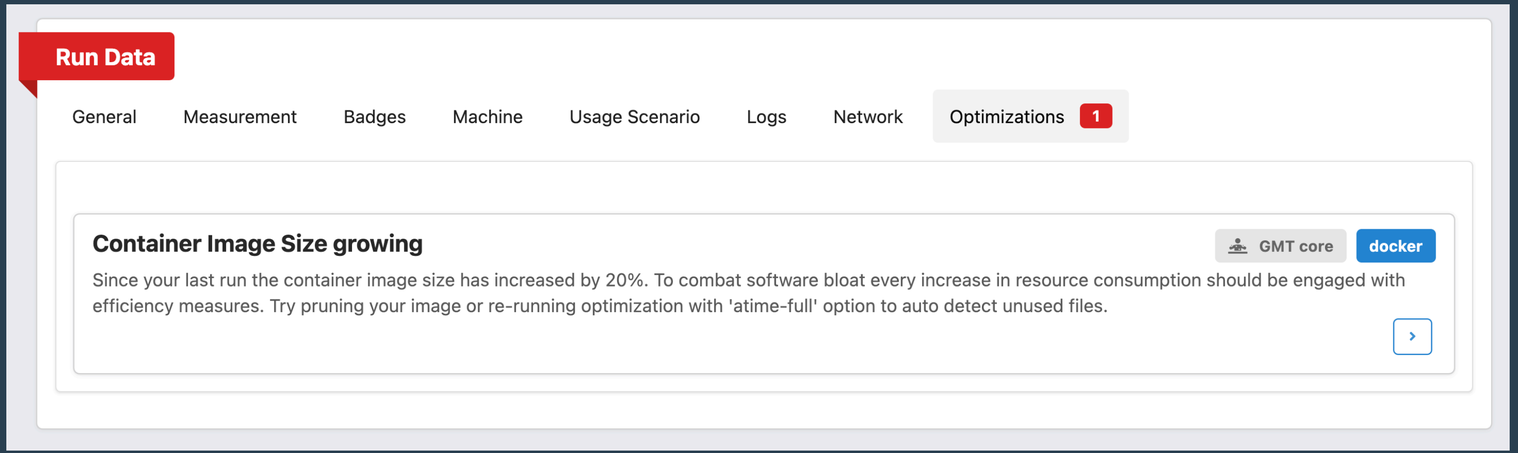
An important domain of Green Coding is also fighting software bloat

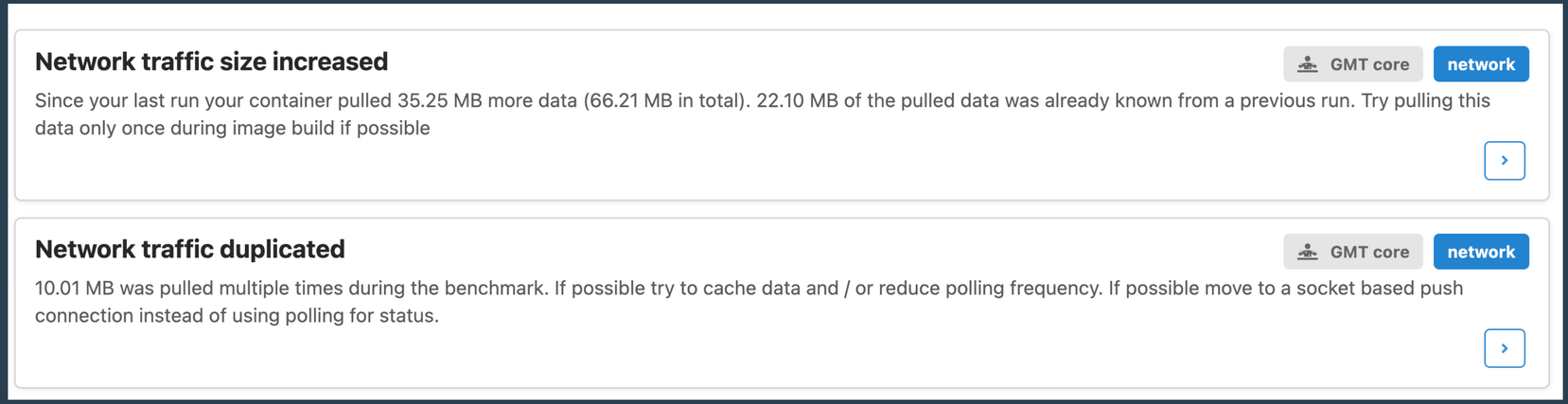


Docker image size between different releases of a software

Optimizations - Part #3

Changes over time

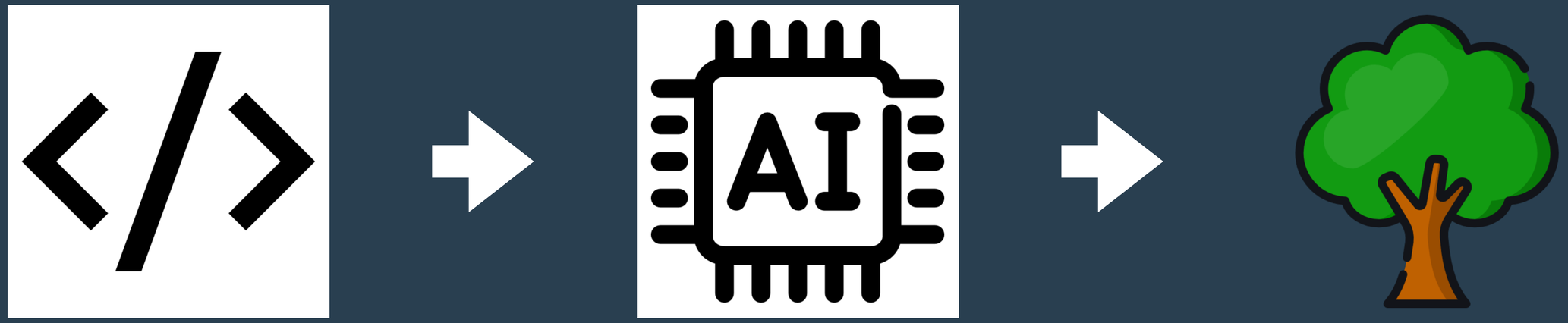




Example for intercepted and analysed network traffic

Optimizations - Part #4 - AI

AI Optimizations - What everyone wants



Code

Optimizations - Part #4

AI Optimizations - What everyone wants

Don't we know this from the Simpsons already?

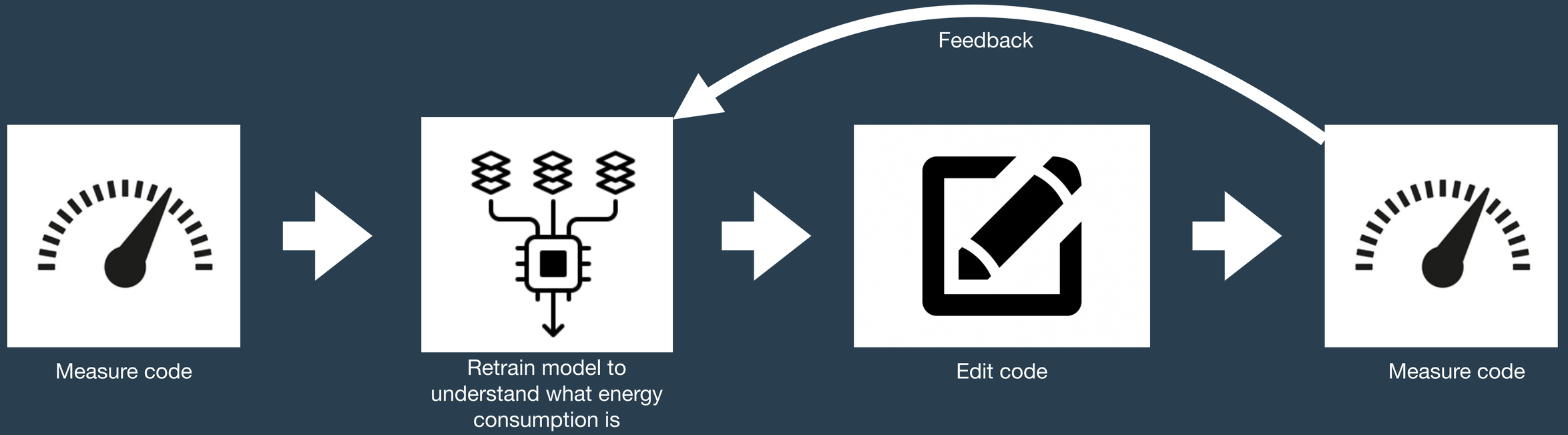
Optimizations - Part #4

AI Optimizations - This doesn’t really work that well

- AI models no understanding of “green”
- Ai models can not consume large complex projects
- Performance Engineering ≠ Green Coding
- Modern systems use loads of libraries and functionalities that change depending on the system. Some operations great on X86 but really bad on ARM
- Normally ends up being Lint++

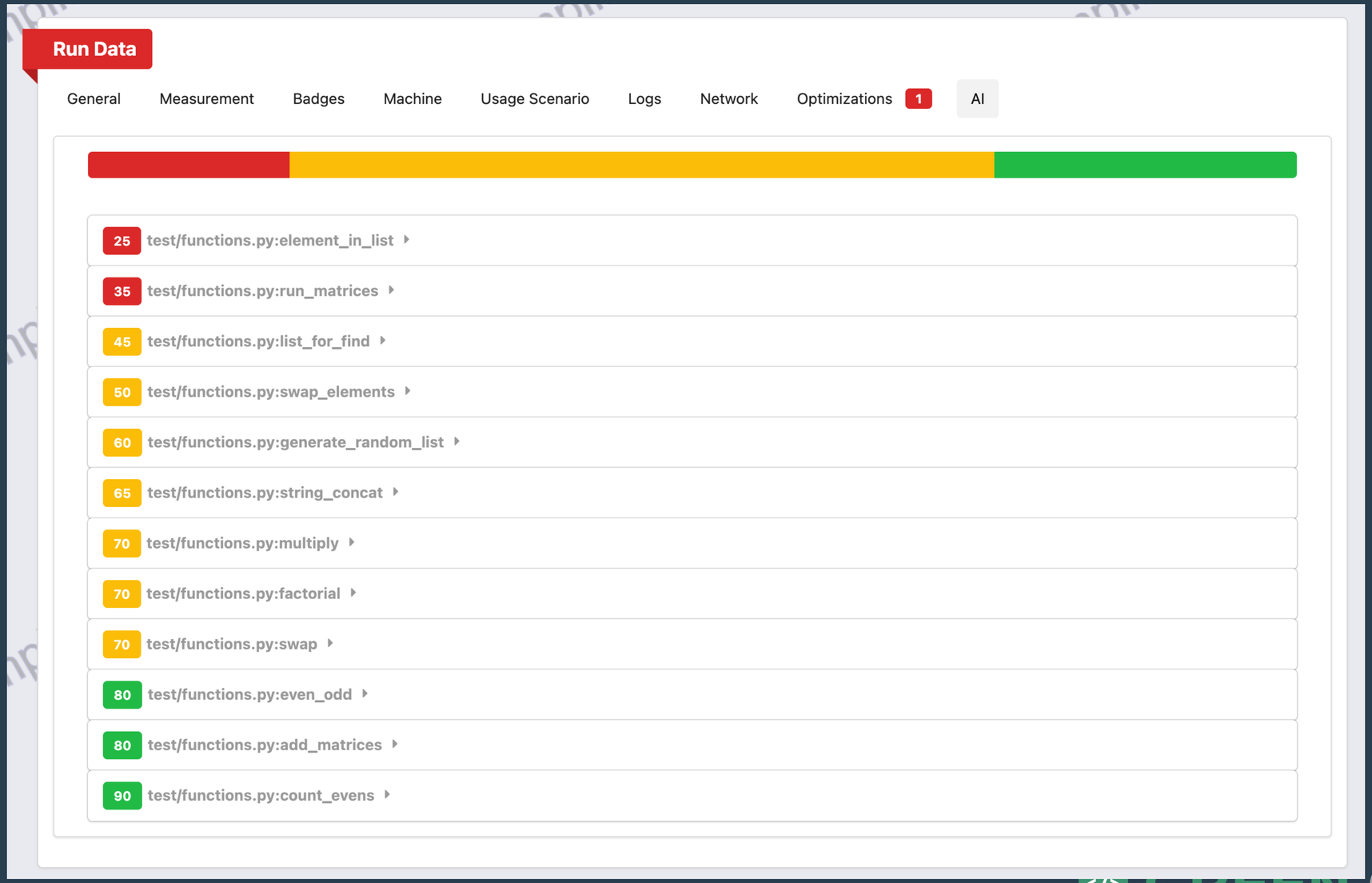
Optimizations - Part #4

AI Optimizations - What we are working on - WIP



Problem: You need a lot of compute

Optimizations - Part #4 - Our current state



Optimizations - Part #4 - Our current state

45 **test/functions.py:list_for_find**

You code

```
def list_for_find(length):
    randome_elements = random.sample(range(MIN, MAX), 100)
    list_seq = list(generate_random_list(length))
    counter = 0
    for ele in randome_elements:
        if ele in list_seq:
            counter += 1
    return counter
```

ChatGPT

To properly assess the performance and optimization potential of the provided piece of code, we would need to consider several factors, such as code efficiency, use of data structures, algorithmic complexity, code readability, and maintainability. As the code snippet is incomplete and lacks context (e.g., definitions of `MIN`, `MAX`, and `generate_random_list`), I can only provide a general assessment based on the visible portion.

Here's a brief analysis:

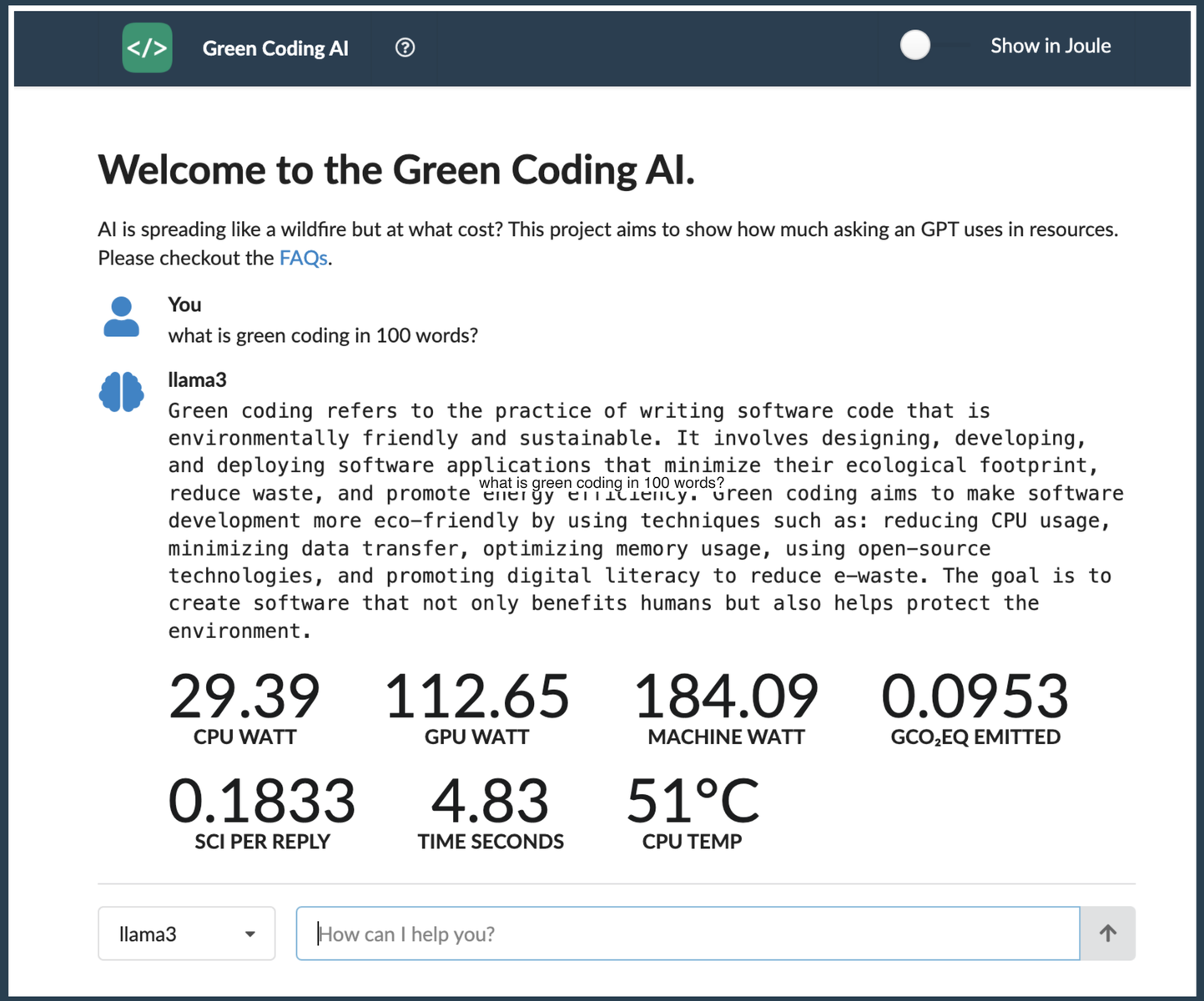
1. **Algorithm Efficiency**: The `if ele in list_seq` operation is O(n) for a list, where n is the length of `list_seq`. This means if `list_seq` is long, the operation will be slow since it checks every element until a match is found.
2. **Use of Data Structures**: Using a list for `list_seq` is not optimal for membership tests. A set would be more efficient, as membership tests are O(1) on average for a set.
3. **Randomness**: The `random.sample` function is used correctly to obtain unique elements, assuming `MIN` and `MAX` are properly defined.
4. **Variable Naming**: The names `randome_elements` and `list_seq` could be more descriptive. There is also a typo in `randome_elements`; it should be `random_elements`.
5. **Magic Numbers**: The '100' in `random.sample(range(MIN, MAX), 100)` is a magic number and its significance is not clear without additional context.
6. **Error Handling**: There is no error handling for potential issues such as the `generate_random_list` function returning a non-list type or the range for `random.sample` being smaller than the sample size desired.
7. **Code Readability**: The code is relatively straightforward and readable, but could benefit from comments explaining the purpose and logic.

Considering these points and assuming the unseen parts of the code (like the `generate_random_list` function) are implemented with similar considerations, I would give the visible code a rating of around 45 out of 100. With optimization, such as using a set for `list_seq` and cleaning up the code, the rating could be much higher.

Rating: 45

Improve this with AI

Our showcase for testing inferencing cost of Open Source AI models



GREEN CODING;

Different models have vastly different costs. Try the playground to get a glimps!

You

what is green coding in 100 words?

tinyllama

Green Coding, also known as Lightweight Coding, is a method for software development that simplifies the process of writing and maintaining code without compromising performance or functionality. It involves breaking down complex algorithms into smaller, more manageable components called "steps" to simplify the programming process while still ensuring high-quality code. In essence, green coding reduces the number of lines of code required for a given function, thereby reducing overall code size and improving software performance. The technique is highly effective for developing web applications or large data processing systems as it helps in identifying the most critical parts of code while also ensuring that they are efficient enough to be executed efficiently. In this way, green coding facilitates a more efficient development process and ultimately provides developers with better quality code that meets performance requirements.

28.75	98.61	170.31	0.0273
CPU WATT	GPU WATT	MACHINE WATT	GCO_2EQ EMITTED

0.0545	1.49	51°C
SCI PER REPLY	TIME SECONDS	CPU TEMP

https://green-coding.ai

Now we want your opinions and ideas

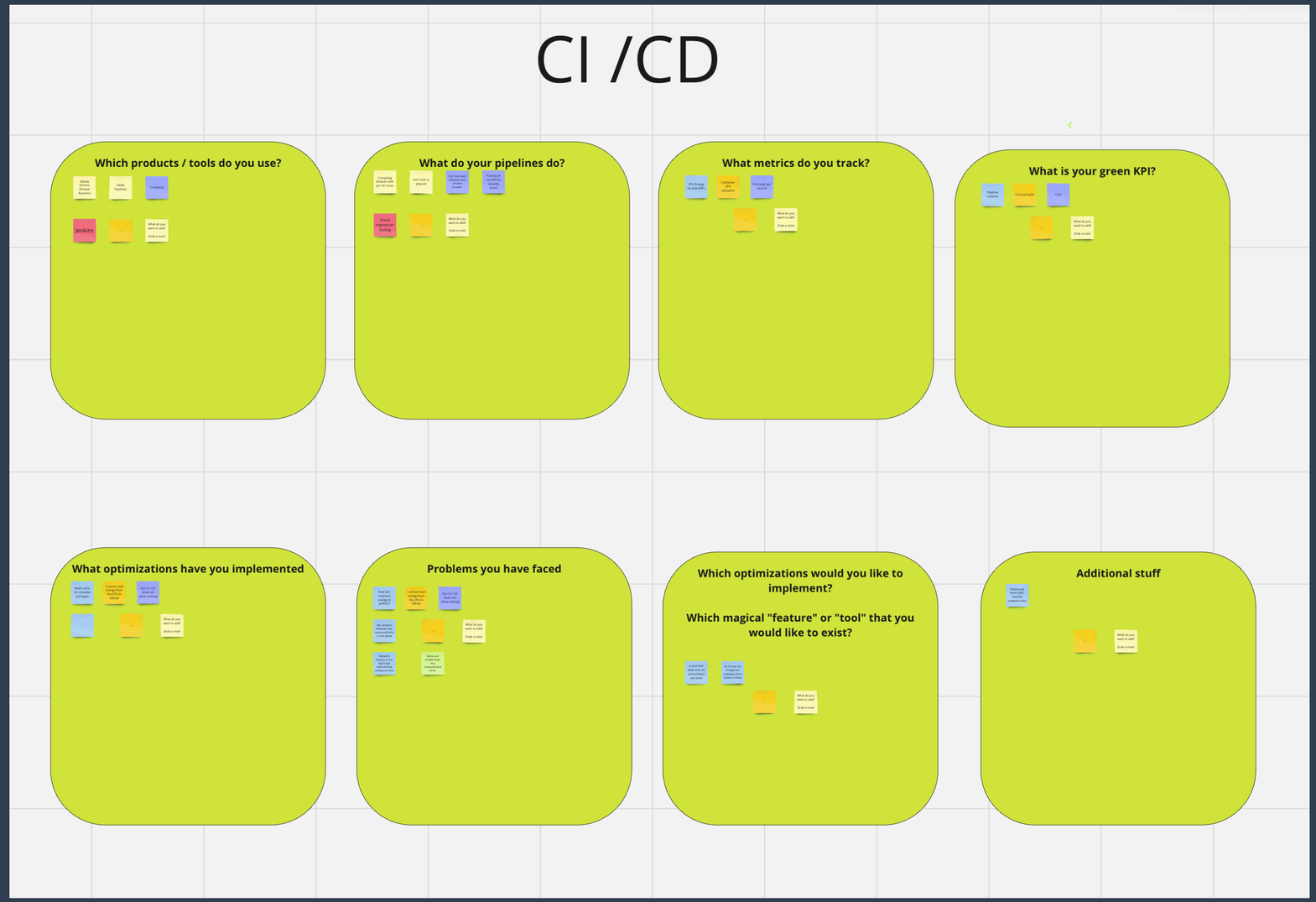
Let's talk about how to implement and automate optimizations

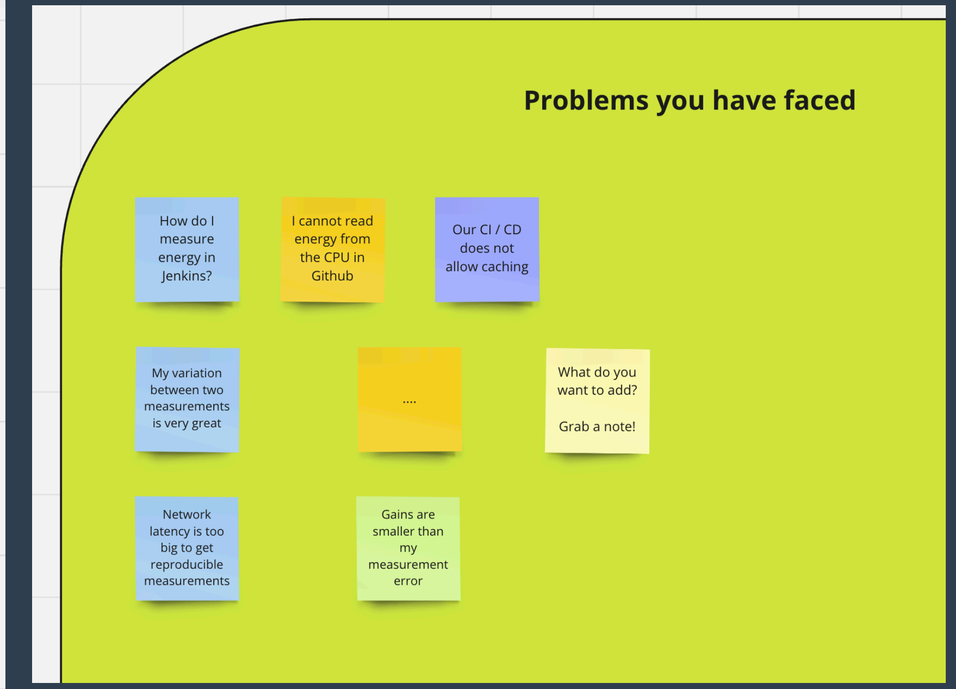
- We have prepared a Miro board at: https://t.ly/hTuwW
- Questions are:
  - What tools are you using?
  - What green optimizations have you implemented already?
  - What problems have you faced?
  - What magical tool would you wish for?
- Then in one hour: We deep dive into the questions and discuss them in the groups. Please also pick out Post-its you want to talk about and start a discussion :)

GREEN CODING;

Now we want your opinions and ideas

Miro Board Preview





Close up of one box

GREEN CODING;

Now we want your opinions and ideas

Preview: Questions of the Miro Board

- Which products / tools do you use?
- What does your software do? / What do your pipelines do?
- What is your green KPI? / What constiutes a green software for you?
- What optimizations have you implemented
- Problems you have faced
- Which optimizations would you like to implement? / Which magical "feature" or "tool" that you would like to exist?