

# Energy-Efficient Software

***Datamatikeruddannelsens ERFA dag April 2024***

## The Green Architecture Framework

… And some thoughts about teaching

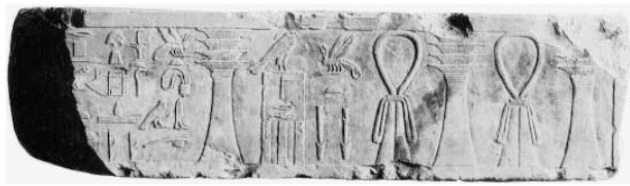
*Henrik Bærbak Christensen*
Associate Professor, Aarhus University
Imhotep

# About Me



- *Henrik Bærbak Christensen*
- Associate professor (lektor) since 2003
  - Collaboration with many Danish IT companies
    - Systematic, Jyske Bank, Terma, Rambøll, KMD, Uber, Kamstrup, …
  - Actually have complex software *in production* ☺
- Faglig koordinator for Master (SWK) / IT-Vest
- Owner of
    - Course development and consultancy
    - http://www.imhotep.dk

AARHUS UNIVERSITY



**Imhotep**
Vidensformidling inden for Softwareudvikling

# Agenda



- … today is two fold
- First: **Present aspects of *Green Architecture FW***
  - *A set of tactics for improving energy-efficiency in software*

The Green Architecture Framework




Talk from GOTO Aarhus 2023

- Second: **Thoughts about teaching it…**
  - *A concrete example of how I have made a major exercise on the topic – and evaluated student's perception*

Teaching Energy-Efficient Software – An Experience Report

# Green Architecture Framework

Part I

# Motivation

- Well… Large and important topic !

> **Sustainability** is a societal goal that relates to the ability of people to safely co-exist on Earth over a long time. Specific definitions of sustainability are difficult to agree on and have varied with literature,

- Climate crisis, green energy, reduce carbon footprint…
- Part of it is of course: *Green computing*
  - *reduce energy consumption and lower carbon emissions from the design, use and disposal of technology products*
  - Encompass the full life-cycle of computing
    - Materials to produce, run, dispose our computational work

# Motivation

- I will (mostly) delimit myself to *energy-efficiency*

**Energy conversion efficiency** ($\eta$) is the ratio between the useful output of an energy conversion machine and the input, in energy terms. The input, as well as the useful output may be chemical, electric power, mechanical work, light (radiation), or heat. The resulting value, $\eta$ (eta), ranges between 0 and 1.[1][2][3]

- *… or*

Literally, it measures the rate of computation that can be delivered by a computer for every watt of power consumed.

- Ala: *Patient Inger's blood-pressure is uploaded to server*
  - Architecture A spends **3.1mJ**; Architecture B spends **6.7mJ**
  - *We prefer architecture A, right?*

# Energy and Power



- We are basically interested in *energy*
  - Energy = Amount of work
- **Energy** is measured in **Joule** (SI unit)
  - 1J work is done when a force of 1 newton displaces a mass 1 meter
    - Newton = force accelerating 1kg by 1m/s^2
- **Power** is measured in **Watt**
  - Power = energy / second; 1 W = 1 J/s
    - Or…
  - 1 Joule is 1 W in 1 second = 1 Ws
  - **1 KWh = 3.6 MJ**

| joule | |
|---|---|
| Unit system | SI |
| Unit of | energy |
| Symbol | J |
| Named after | James Prescott Joule |
| Conversions | |
| 1 J *in* ... | ... *is equal to* ... |
| SI base units | kg·m$^{2}$·s$^{-2}$ |
| CGS units | 1 × 10$^{7}$ erg |
| watt-seconds | 1 W·s |
| kilowatt-hours | ≈2.78 × 10$^{-7}$ kW·h |
| kilocalories (thermochemical) | 2.390 × 10$^{-4}$ $kcal_{th}$ |
| BTUs | 9.48 × 10$^{-4}$ BTU |
| electronvolts | ≈6.24 × 10$^{18}$ eV |

100g Hellmann's Mayonnaise contains 2,965,000 J (0.82 kWh)
About 35 min sweaty bicycling…

# Motivating Example

- Gangnam Style
  - Was shown 1.7 x 10^9 times the first year
  - Energy to stream once is 0.19kWh
  - **Total: 312 GWh**


ELSEVIER
Journal of Systems and Software
Volume 117, July 2016, Pages 185-198

Empirical evaluation of two best practices for energy-efficient software development

Giuseppe Procaccianti, Héctor Fernández, Patricia Lago

PSY - GANGNAM STYLE [Original Video]

  - Danish average house ("parcelhus") yearly electricity consumption
    - 4.4 – 5.0 MWh
  - **~ 70.000 Danish houses**

Morale: None…
But it is a bit thought provoking…

# Energy = Work Done

*Hardware* spends energy, because our *Software* wants work to be done.



# What is using Power?

- Note
  - **CPU drives much else**
    - Heat/fan/ **cooling**
  - Note
    - SSD+DRAM is ‘cheap’ power wise…

## Gaming Computer

Purpose: heavy gaming, heavy graphics editing, overclocking, moderate virtualization, web surfing, listening to music, viewing images, watching high resolution videos

Components

| | |
|---|---|
| High End CPU (Intel Core i7) | 95 W |
| Aftermarket CPU Heatsink Fan | 12 W |
| High End Motherboard | 80 W |
| RAM Modules x 2 | 6 W |
| High End Graphics Card ($251 to $400) | 258 W |
| Dedicated Sound Card | 15 W |
| Solid State Drive | 3 W |
| 3.5" Hard Disk Drive | 9 W |
| Blu ray Drive | 30 W |
| Case Fans x 4 | 24 W |

Gaming PC Power Requirements 532 Watts

~ 210W

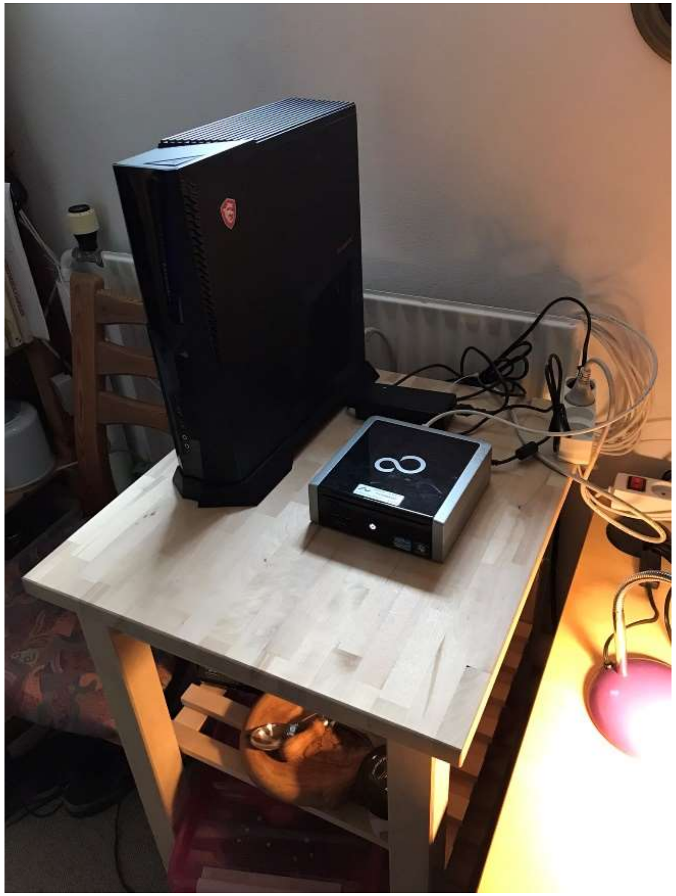
~ 18W

Out-of-box: Network
Devices: Screen, GPS, sensors…

# Examples: My Humble Lab



- The Lab
  - Fujitsu Esprimo Q900 (2012)
  - Fujitsu Primenergy TX 100 (2012)
  - MSI Trident (2020)
- Installed with Ubuntu 22.04 LTS
  - Headless
    - No use for the GeForce RTX™ 2080 Ti ☺
- **Idle** Power Consumption
  - Esprimo: ~ 11 W (plug) / 2.8 W (CPU)
  - MSI: ~ 40 W (plug) / 7.4 W (CPU)
    - At ~95% CPU load@Plug: Esprimo 43W and MSI 160W



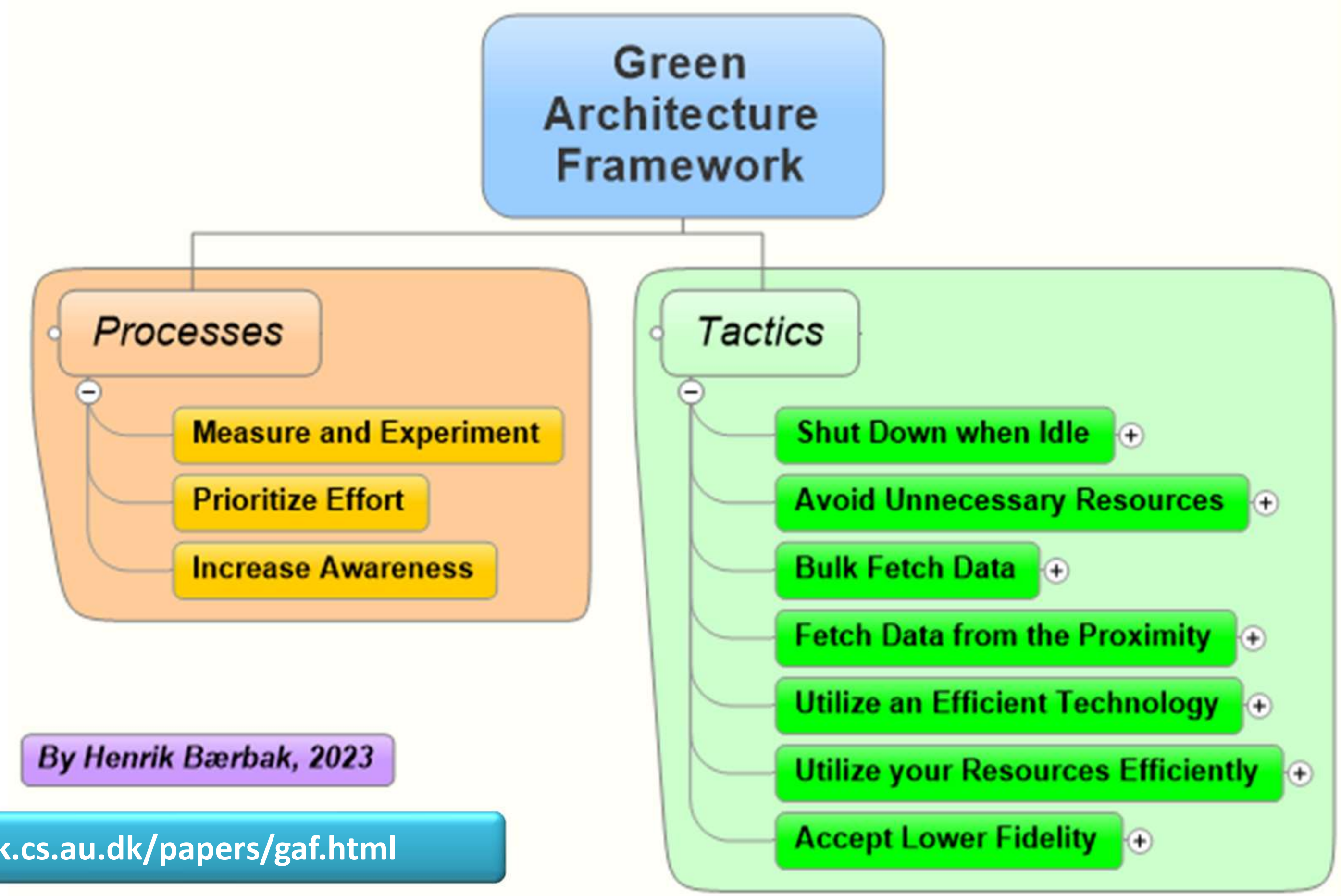
# So… Green Architecting?

How do I then design my architecture and code, so it consumes less energy?

# The Framework



- *The Green Architecture framework ☺*



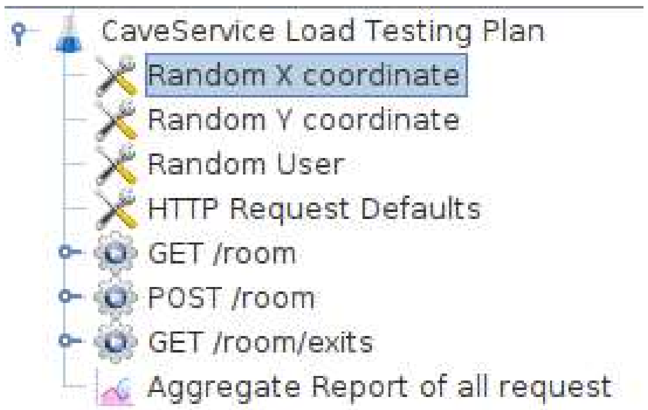

https://baerbak.cs.au.dk/papers/gaf.html


# Processes

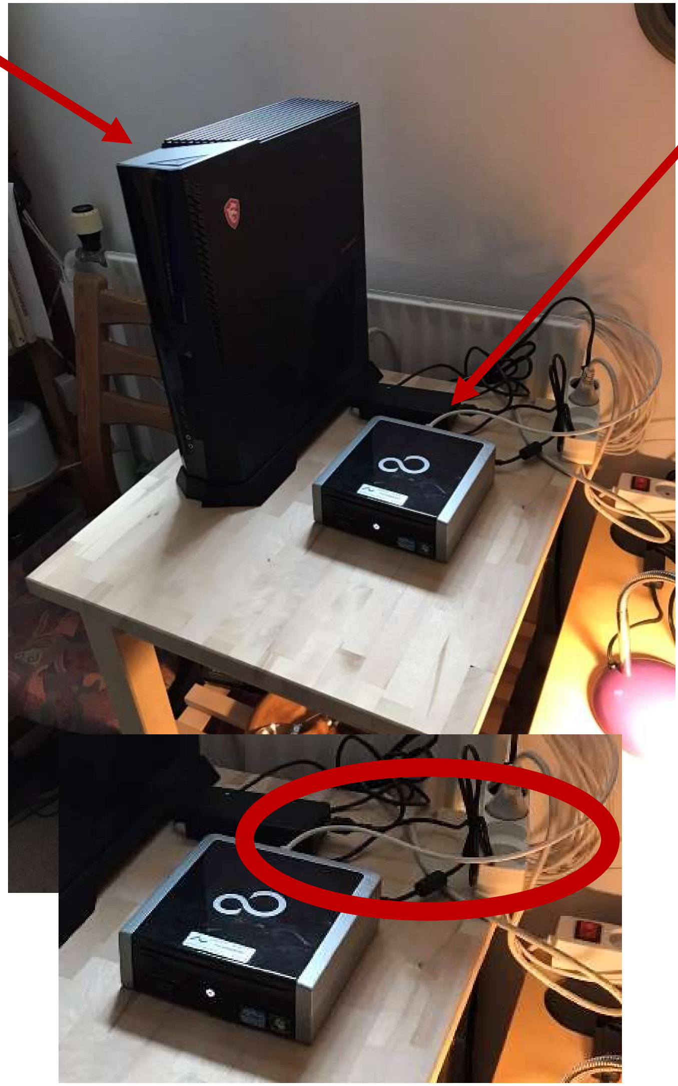
How we design Green Architectures?

# You Must Measure!

- Load Generator
  - Apache JMeter™




- Cabled network
  - No other traffic



- Target Machine
  - Ubuntu Server with PowerStat
  - And target app
    - container
  - And nothing else ☺

# Increase Awareness



- Increase Awareness
  - All architects/developers/ stakeholders informed about how to increase energy-efficiency


- From my kitchen. Which one is 2W and which 40W?

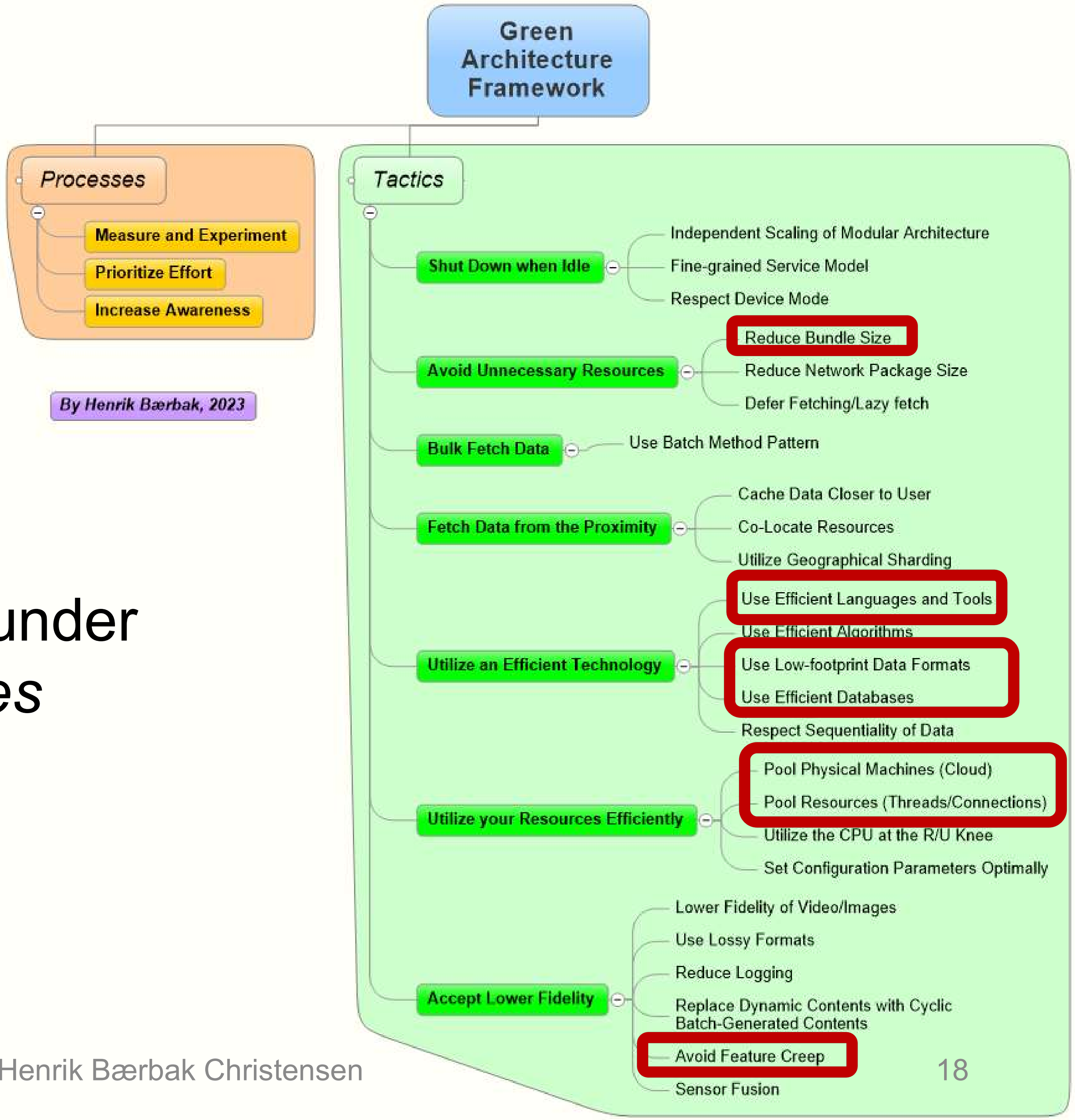

# Tactics

How do we then *do*
Green Architecting?



- Set of *tactics*
  - *Architectural design decision to impact energy-efficiency*

- Quite a lot of tactics under *seven main categories*



# Avoid Unnecessary Resources



- *"Don't put things in your suitcase, that will not be used"*
- **Reduce Bundle Size**
  - Example: Docker base image for Java

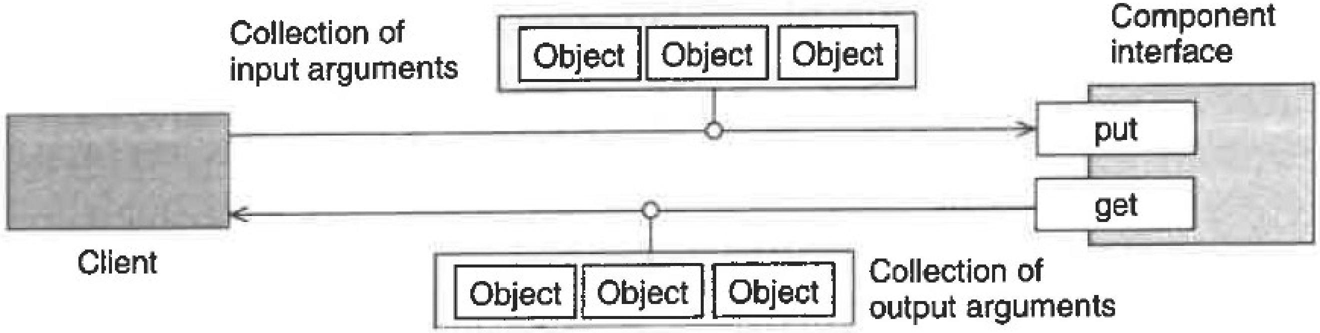
```
adoptopenjdk/openjdk11     alpine-jre      bfb0a8fceb23   10 months ago   149MB
openjdk                    11-jre-slim     764a04af3eff   12 months ago   223MB
openjdk                    11              47a932d998b7   12 months ago   654MB
```

  - That is: **4.3 times bigger image** to transfer and load for the "same" Java Runtime Environment for my server…

# Bulk Fetch Data



- *“Buy 50 things at the super market once, instead of making 50 trips buying a single thing”*
  - POSA4(2007): **Batch Method**

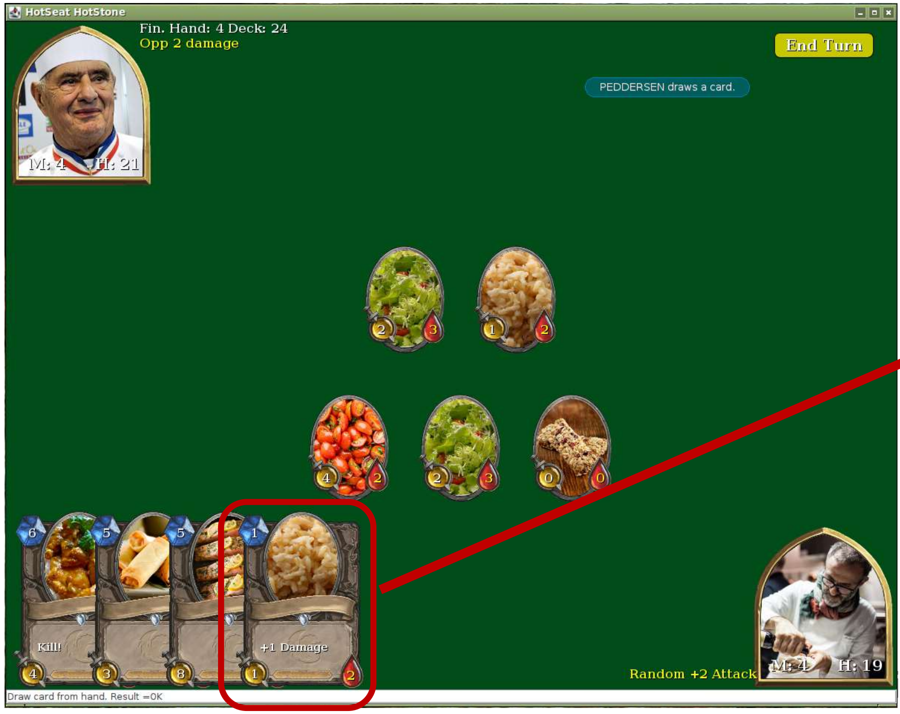



- **Iterator pattern over network** is an energy *anti pattern*
  - *getNext()* across the network is a *chatty interface*
  - Use *pagination* instead – bulk fetch next 50 items in one chunk

# Bulk Fetch Data

Imhotep Vidensformidling inden for Softwareudvikling

- *“Buy 50 things at the super market once, instead of making 50 trips buying a single thing”*
- Example
  - Classic OO is often a very *fine-grained API*




```
public interface Card extends Effectable, Identifiable, Attributable {
  7 implementations  ≗ Henrik Bærbak Christensen
  String getName();
  7 implementations  ≗ Henrik Bærbak Christensen
  int getManaCost();
  8 implementations  ≗ Henrik Bærbak Christensen
  int getAttack();
  7 implementations  ≗ Henrik Bærbak Christensen
  int getHealth();
  7 implementations  ≗ Henrik Bærbak Christensen
  boolean isActive();
  7 implementations  ≗ Henrik Bærbak Christensen
  Player getOwner();
}
```

The UI needs to get all cards’ data from the server when redrawing UI…

# Example of A/B Architecture



- The Card interface
  - **Two** remote implementations

```java
public interface Card extends Effectable, Identifiable, Attributable {
  7 implementations  Henrik Bærbak Christensen
  String getName();
  7 implementations  Henrik Bærbak Christensen
  int getManaCost();
  8 implementations  Henrik Bærbak Christensen
  int getAttack();
  7 implementations  Henrik Bærbak Christensen
  int getHealth();
  7 implementations  Henrik Bærbak Christensen
  boolean isActive();
  7 implementations  Henrik Bærbak Christensen
  Player getOwner();
}
```

Broker pattern
Classic (RMI-like)

```java
@Override
public String getName() {
  return requestor.sendRequestAndAwaitReply(cardId,
          OperationNames.CARD_GET_NAME, String.class);
}

@Override
public int getManaCost() {
  return requestor.sendRequestAndAwaitReply(cardId,
          OperationNames.CARD_GET_MANA_COST, int.class);
}

@Override
public int getAttack() {
  return requestor.sendRequestAndAwaitReply(cardId,
          OperationNames.CARD_GET_ATTACK, int.class);
}
```

Broker pattern
Batch Method

```java
@Override
public String getName() {
  // eternal caching
  if (name != null) return name;
  name = fetchCardFromCache().getName();
  return name;
}

@Override
public int getManaCost() { return fetchCardFromCache().getManaCost(); }

@Override
public int getAttack() { return fetchCardFromCache().getAttack(); }
```


```java
private Card fetchCardFromCache() {
  long now = System.currentTimeMillis();
  if (cachedCard == null || now > timestamp + FeatureFlag.CACHE_EXPIRE
    cachedCard = requestor.sendRequestAndAwaitReply(cardId,
            OperationNames.CARD_GET_PODO, CardClientPODO.class);
    timestamp = now;
    cacheRefresh++;
  } else
    cacheHit++;
  return cachedCard;
}
```

# Bulk Fetch Data



- *“Buy 50 things at the super market once, instead of making 50 trips buying a single thing”*
- Comparison
  - Classic Broker (ala Java RMI)
    - 5.66W (σ 0.90W)
  - Batch Method Broker (5sec Caching)
    - 4.12W (σ 0.79W)
    - (Reducing number of network calls to 43%)
  - Saving **27.2% energy**

- ***And this is on the server side only!***

# Utilize an Efficient Technology



- *“Switch the 20 W halogen bulb to a 4 W LED bulb”*
- **Use Efficient Languages and Tools**
  - (This 2017 study used rather unrealistic benchmark programs)
    - Mandelbrot???

| | 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 |
| (v) Erlang | 42.23 |
| (i) Lua | 45.98 |
| (i) Jruby | 46.54 |
| (i) Ruby | 69.91 |
| (i) Python | 75.88 |
| (i) Perl | 79.58 |

**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

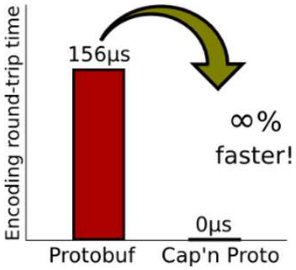
João Saraiva
HASLab/INESC TEC
Universidade do Minho, Portugal
saraiva@di.uminho.pt

Own experiment of a 3 endpoint REST Service impl:
Java (baseline)
Go (-3.5% energy)
Scala (+27% energy)
**Python (+162%, 2½x)**

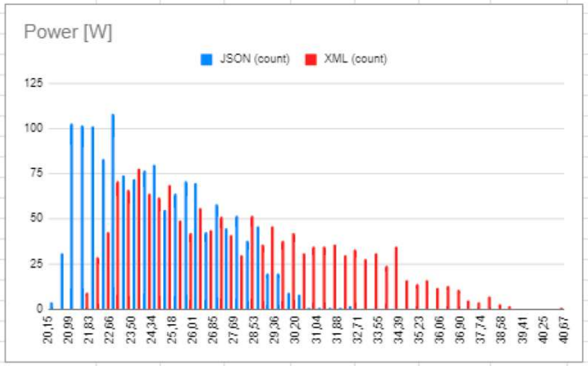
# Utilize an Efficient Technology



- *“Switch the 20 W halogen bulb to a 4 W LED bulb”*
- **Use Low-footprint Data Formats**
  - *XML is much more verbose than JSON*
  - Binary formats: ProtoBuf, Cap’n Proto




– Part-time students did a XML versus JSON experiment




JSON: 24.5W (σ 2.5w)
XML: 27.9W (σ 4.1W)
*That is 12.2% saved energy by using JSON over XML*

# Utilize an Efficient Technology



- *“Switch the 20 W halogen bulb to a 4 W LED bulb”*
- **Use Efficient Databases**
    - If only a ‘blob storage’ / key-value store is necessary then pick one, rather than a SQL or a MongoDB database

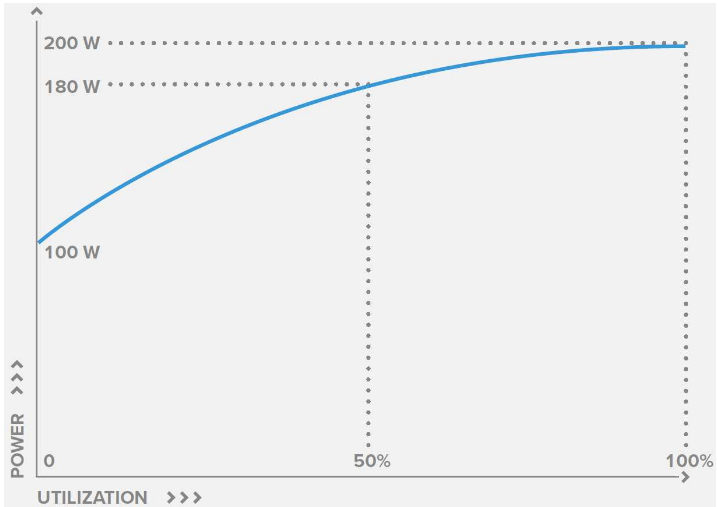
  – Example
    - REST service (three endpoints: One POST and two GET)
    - Comparing the four approaches' power

| | | |
|---|---|---|
| – Fake in-memory db: | ~ 11.8W σ 0.3W | (- 44.6%) |
| – Redis db: | ~ 14.7W σ 0.3W | (- 31.0%) |
| – Mongo db (naive): | ~ 21.3W σ 0.5W | (baseline) |
| – Mongo db (optimized): | ~ 20.9W σ 0.2W | (- 1.9%) |

# Utilize your Resources Efficiently

- *“Prepare several items in the oven at the same time”*

- An *idling* computer spends between 1/4 - 2/3 power compared to a *busy* computer
  - The *non-proportionality of energy consumption*
- Which means:
  - **Per-transaction energy cost is lowering as the computer is more heavily utilized!**

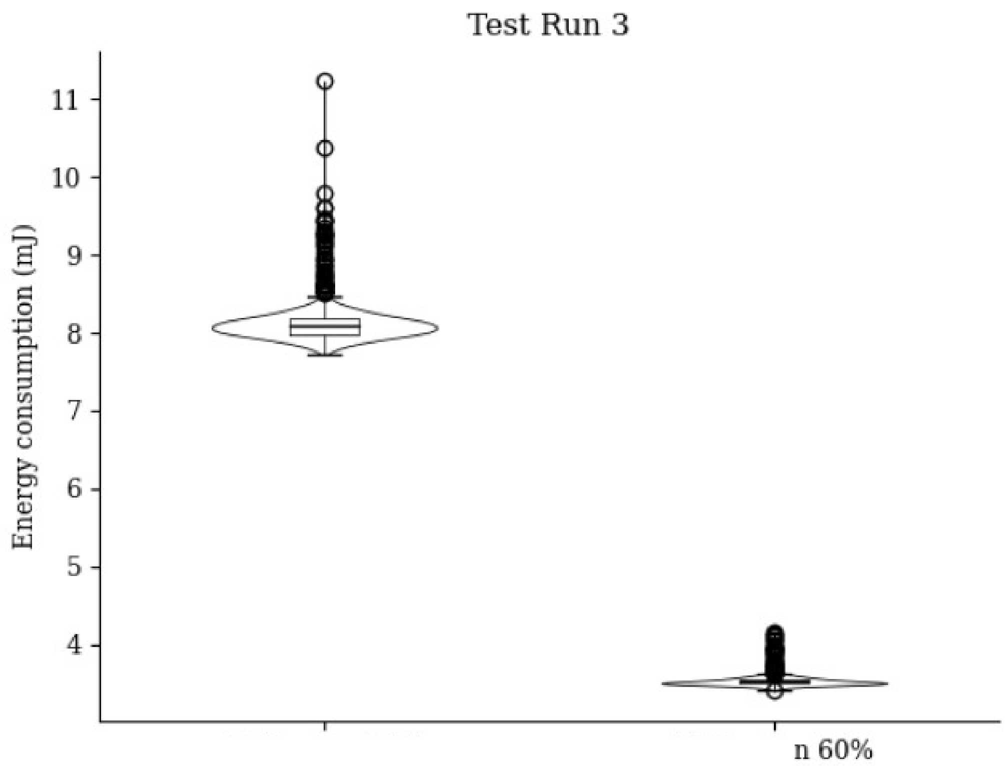


# From My Part-Time Students



Table 10 summarize the results related to energy measurement for the 4 test runs. As expected, the lower utilization leads to lower Power consumption, but since the throughput is reduced by a factor 6, and the power is only reduced by a factor of around 2.5, the energy per request is lower for the higher utilization. The energy cost per request for the high utilization is between 43.3% and 43.5% of the cost for running at lower utilization(baseline). Making a request at 10% will cost more than twice (2.3) the energy compared to making it at 60% utilization.

| # | Type | Start | Energy | | | |
|---|---|---|---|---|---|---|
| | | | Mean [W] | Stddev [W] | mJ per request | |
| 2 | Utilization 10% | 01:08:43 | 2.5 | 0.066 | 8.11 | baseline |
| | Utilization 60% | 01:40:42 | 6.45 | 0.134 | 3.52 | 0.434 |
| 3 | Utilization 10% | 02:46:41 | 2.5 | 0.072 | 8.12 | baseline |
| | Utilization 60% | 02:13:41 | 6.47 | 0.137 | 3.53 | 0.434 |
| 4 | Utilization 10% | 03:51:40 | 2.5 | 0.0735 | 8.11 | baseline |
| | Utilization 60% | 03:18:41 | 6.44 | 0.136 | 3.51 | 0.433 |
| 5 | Utilization 10% | 04:23:40 | 2.49 | 0.0665 | 8.08 | baseline |
| | Utilization 60% | 04:55:39 | 6.45 | 0.137 | 3.52 | 0.435 |




**Conclusion:**

As the $H_0$ hypothesis can be rejected, there is a statistically significant power saving of 56.5-56.7% per request when running at 60% utilization compared to running at 10% utilization.

# Utilize your Resources Efficiently



- *"Prepare several items in the oven at the same time"*
- **Pool Physical Machines (Cloud)**
  - Host a lot of VM on same physical machine means *when A is not using the CPU, then B have it*
    - – *Cloud centers are better at that than on-premise*

- **Pool Resources (Threads/Connections)**
  - – Threads and connections are expensive to create and deallocate
    - Pool them

Own experiment: Three-tier system with MariaDB storage. A) Naïve 'connection-pr-request' connector; B) C3P0 'pool'.
Pooled connection spent **about 62.5% less energy**.

Naïve: 192tps
C3P0: 514 tps

Disclaimer: Naïve had coarse-grained locks applied…

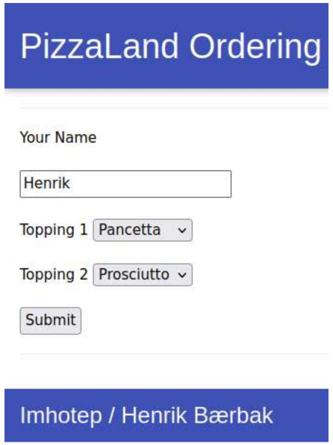
# Accept Lower Fidelity

**Imhotep** Vidensformidling inden for Softwareudvikling

- **Avoid Feature Creep**

‘PizzaLand’ Experiment:
A ‘core’ REST based pizza ordering system with ordering and inventory system in MariaDB; deployed on a 2012 i5 CPU @ 2.5GHz/4 core + 8GB DDR3 RAM

***Handles 51,800 orders per hour!***



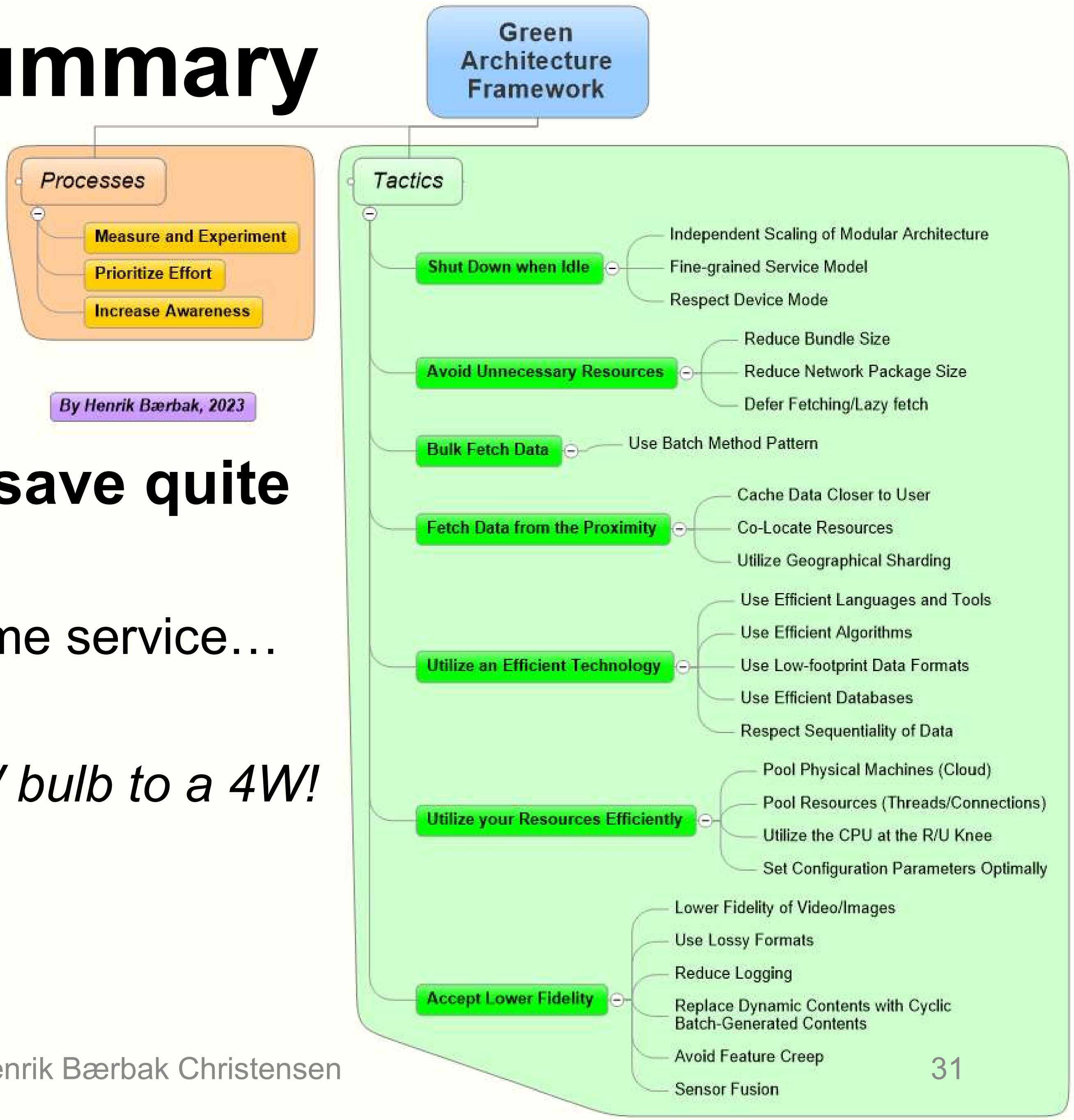

**Write results to file / Read from file**

**Filename** /home/csdev/proj/evuproject/energy-pizzaland/c3p0-monolith.jtl **Browse...** **Log/Display Only:** ☐ **Errors** ☐ **Successes** **Configure**

| Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Maximum | Error % | Throughput | Received K... | Sent KB/sec |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GET /order | 779170 | 4 | 1 | 2 | 3 | 95 | 0 | 283 | 0.00% | 485.2/sec | 218.99 | 62.55 |
| POST /order | 23180 | 37 | 24 | 76 | 118 | 215 | 9 | 414 | 0.00% | 14.4/sec | 4.19 | 3.76 |
| POST /finish | 23084 | 6672 | 6514 | 12010 | 13349 | 15160 | 44 | 24888 | 0.00% | 14.4/sec | 3.43 | 2.62 |
| TOTAL | 825434 | 191 | 1 | 4 | 62 | 8371 | 0 | 24888 | 0.00% | 514.0/sec | 226.60 | 68.93 |

# Summary

- There are a lot of tactics…
- **… and they actually save quite a lot of energy!**
  - While providing the same service…
  - *Let us change the 20W bulb to a 4W!*



# … and Teaching it

Part II

# Context



- Continuing Education / Part-Time Education
  - Master i IT. *Students are full-time developers*

It-vest
samarbejdende universiteter

- Fagpakke:
  - Software Arkitektur i Praksis
    - One week of teaching

| When | What |
|---|---|
| Week 13 | Introduction to the advanced course. Introduction to Performance Engineering. Queue Theory. **Easter.** |
| Week 15 | Performance testing and patterns for performance. |
| Week 16 | Cloud Computing, Virtualization and Container Technology. |
| Week 17 | Big data and NoSQL. |
| Week 18 | Energy-Efficient Software and Architecture. |
| Week 19 | Architectural Erosion. Technical Debt. |
| Week 20 | MicroServices. Architectural Conformance. |

    - One (large) exercise

- Mandatory 4: Performance Testing TeleMed (18/4).
- Mandatory 5: TeleMed in the Cloud. (9/5).
- Mandatory 6: Energy Efficient TeleMed. (1/6).

# Theory…

## SAiP Weekplan 12

The learning goals for Week 18 are:

Energy-Efficient Software and Architecture.

**Literature:**

- [Bass et al., 2021] Chapter 6 (only QAS part).
- [Christensen, 2023] Draft. (Online)
- [Suárez et al., 2021] Cursory, but the best paper I have read so far. It is the main source of inspiration fo
- [Kazman et al., 2018] Cursory - focus on key learnings. On BS.
- [Cruz, 2021] Mandatory. Read it in the 'Note 1' sense to be applied in the M6 mandatory. (Online)
- [Hussain] Optional. (Online)
- [Lago] Optional. (Online)
- [Montagliani, 2020] Optional. Relevant for App developers. (Online)

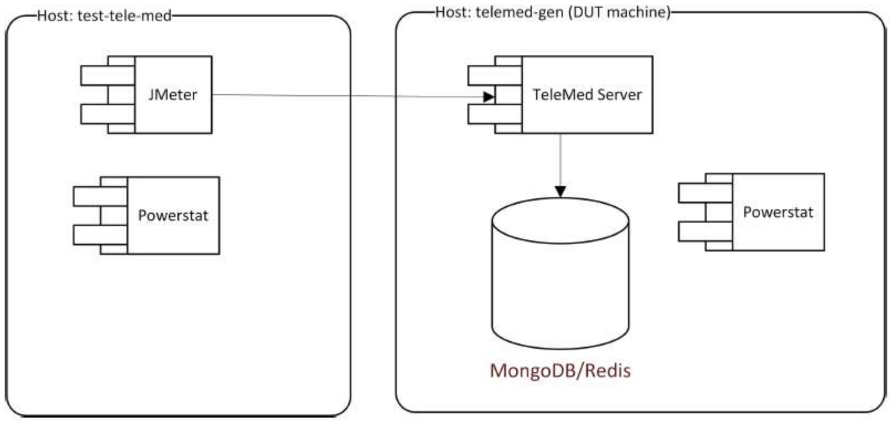
**Slides:**

- W12-1 Energy Efficiency (powerpoint) // (PDF)
- W12-1b Energy Efficiency Long (powerpoint) // (PDF)
- W12-2 Measuring Energy (powerpoint) // (PDF)
- W12-3 Database Experiment (powerpoint) // (PDF)
- W12-4 Logging Experiment (powerpoint) // (PDF)
- W12-5 Language Experiment (powerpoint) // (PDF)
- W12-6 Microservice Experiment (powerpoint) // (PDF)
- W12-7 Batch Method Experiment (powerpoint) // (PDF)

Topics:
- GAF
- Setting up a lab
- Statistical methods and tools

# … and Practice

- TeleMed as basis system *Well known to students!*
  - A ‘scaled down’ system for tele medicine
    - *Knud measures his blood pressure; uploads to central medical information system for review by general practitioner*
- Prerequisites
  - Build a two machine lab
  - Enhance a JMeter load script
- Exercises
  - A/B of MongoDB / Redis
  - A/B of Logging / No Logging
  - Free exercise



# Examples of Work

Imhotep Vidensformidling inden for Softwareudvikling

- DB Choice
  - Redis is the better choice

Redis uses: (10.4 - 9.69) / 10.4 = 6.9% less power than MongoDB for CPU measurements.

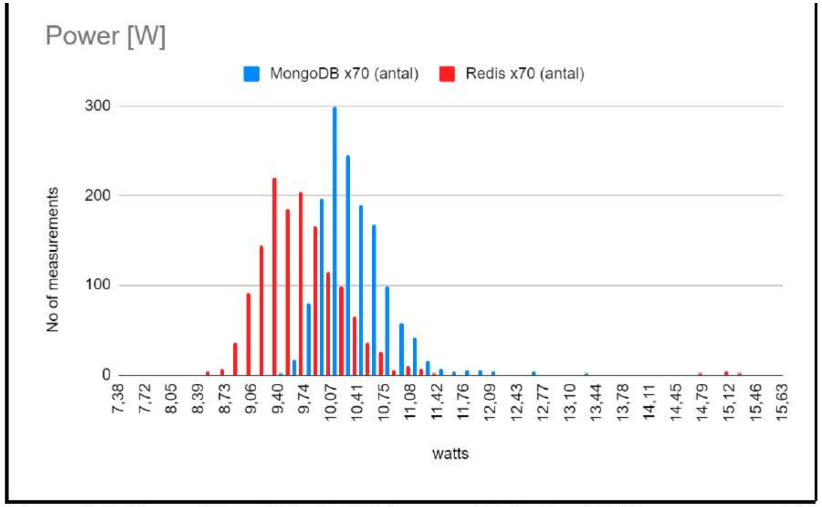
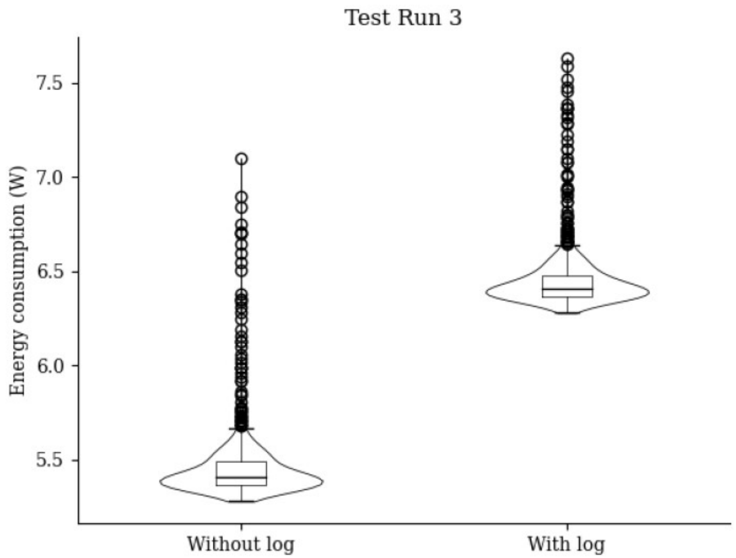



Figure 2-2: MongoDB and Redis CPU power distribution (1440 measurements).

- Logging / No Logging

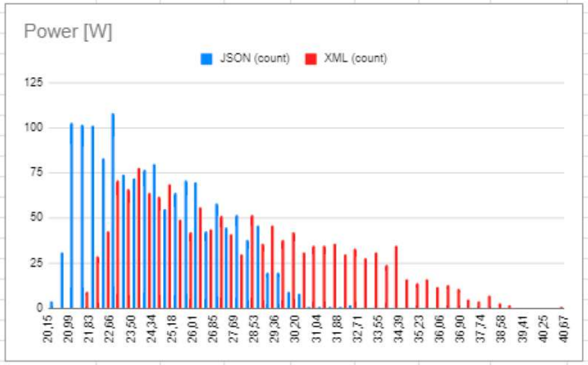



**Conclusion:**

As $H_0$ hypothesis is rejected, **there is a statistically significant power increase of 20-21% in mJ per request** when enabling logging. There is still a difference in memory usage, which might impact the result, even though none of the present data indicates it.

# Free Exercise

1. *Energy Proportionality*: Try to run TeleMed at two different points on the energy curve, for instance loading the target machine at 10% and at 90%; and compare the energy-per-transaction cost.
2. *Docker virtualization*: Compare TeleMed running as a Gradle task versus running it as a docker container (multistage) or as a stack using Docker Swarm.
3. *JVM or Not*: Compile TeleMed using GraalVM to get a native image and compare that with running using the standard JVM.
4. *Choice of Measurement Data Format*: Write a new `Builder` implementation in another format than HL7/XML, and compare that to the given one.
5. *Choice of package size*: Rewrite client-server communication so instead of sending a measurement right away, they are cached at the client side and only transmitted when N measurements have been made (N = 5 or 10 or ?). Compare to the given TeleMed.
6. *SQL Storage Engine*: Write an implementation of `XDSBackend` that stores measurements in a classic SQL database like MySQL, MariaDB, or similar. Compare that implementation to the MongoDB (or Redis) implementation. Any SQL wizards are free to share an implementation on the Forum.
7. *Choice of language*: Compare the Java based TeleMed (configured for in-memory storage) to an alternative TeleMed written in your favorite language.
8. Any other energy-efficiency experiment that you may come up with; also experiments on other code bases than TeleMed. It must, however, be approved by me before you start.




JSON: 24.5W (σ 2.5w)
XML: 27.9W (σ 4.1W)
*That is 12.2% saved energy by using JSON over XML*

# Questionnaire



- Questionnaire to part-time students

**Question 3**

Kursets artikler samt obligatorisk opgave omkring energi-effektiv arkitektur ar øget min opmærksomhed på, forskellige arkitekturer og implementationsvalg har stor betydning for energ

| | | |
|---|---|---|
| I meget høj grad | 6 | (42,86 %) |
| I høj grad | 6 | (42,86 %) |
| I middel grad | 2 | (14,29 %) |
| I mindre grad | 0 | (0 %) |
| I lav grad | 0 | (0 %) |
| Ved ikke | 0 | (0 %) |

**Question 4**

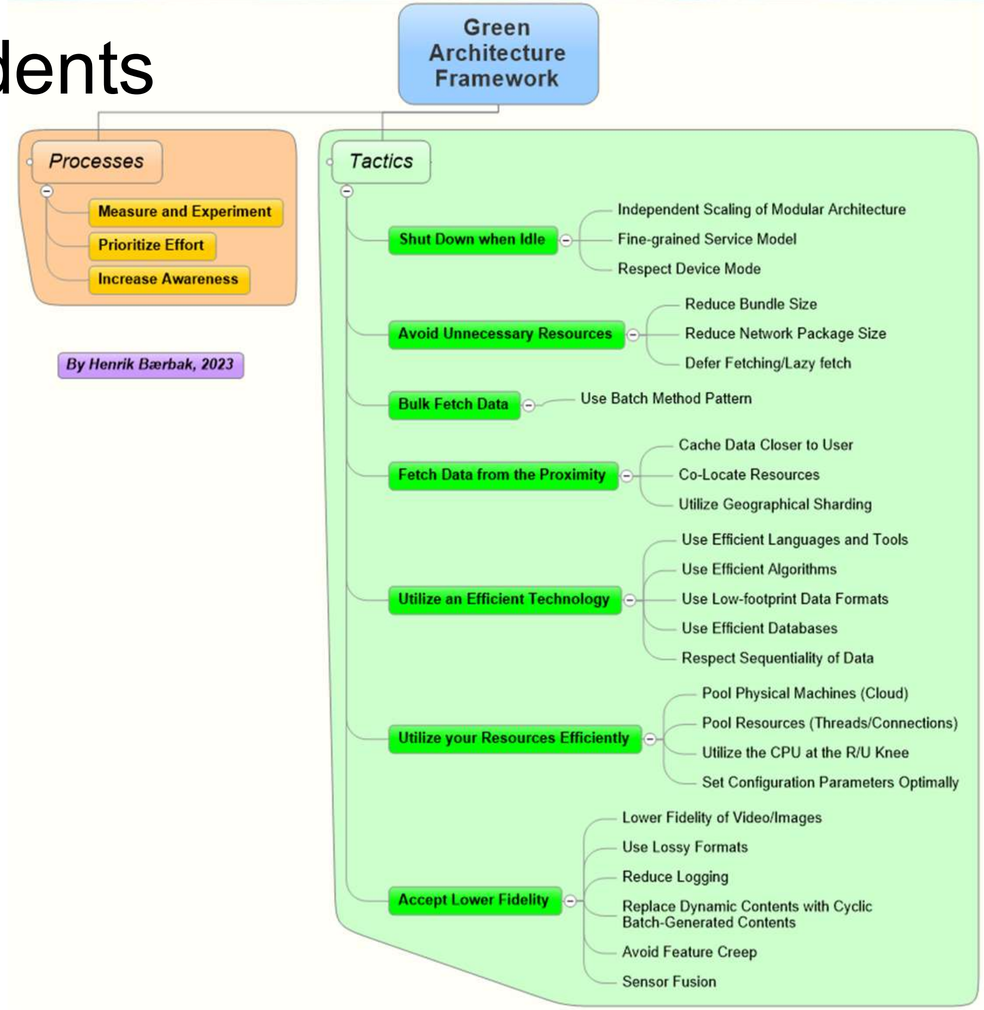
Kursets fokus på energi-effektiv arkitektur vil indgå i, og væ e inspiration til, mit fremtidige arkitektur og softwareudviklingsarbejde, således at energiforbruget søges ergi besparelse (brug af teorien).

| | | |
|---|---|---|
| I meget høj grad | 1 | (7,14 %) |
| I høj grad | 2 | (14,29 %) |
| I middel grad | 8 | (57,14 %) |
| I mindre grad | 2 | (14,29 %) |
| I lav grad | 0 | (0 %) |
| Ved ikke | 1 | (7,14 %) |

**Question 5**

Jeg forventer, at fremtidige projekter som jeg måtte indgå i, i højere g ad vil forsøge at lave konkrete målinger og eksperimenter med energi forbrug (brug af praktiske eksperimenter).

| | | |
|---|---|---|
| I meget høj grad | 0 | (0 %) |
| I høj grad | 2 | (14,29 %) |
| I middel grad | 6 | (42,86 %) |
| I mindre grad | 5 | (35,71 %) |
| I lav grad | 1 | (7,14 %) |
| Ved ikke | 0 | (0 %) |



# Experience

- Awareness ☺
  - It has been an eye-opener for me to do experiments
  - All students report the same ‘Aha’ experience
    - *21% energy increase, just spent on logging ???*
- Labor intensive to do concrete work ☹
  - You need **two ‘raw’ machines** to do serious work
    - And preferably on a cabled network
  - You need to **control all parameters** (load, temperature, …)
  - You need to do **a lot of measurements** (~20 minutes)
    - … and often have a ‘ups, I forgot …’ and have to repeat it



# Experience

- When comparing architecture/design A and B
  - You have to **design and implement both A and B** ☹
    - … which takes time to do…
- Example
  - I have my CaveService coded in
    - Java
    - Go
    - Scala
    - Python
    - (Rust)

*Spark - A micro framework for creating web applications in Kotlin and Java 8 with minimal effort*

Scalatra

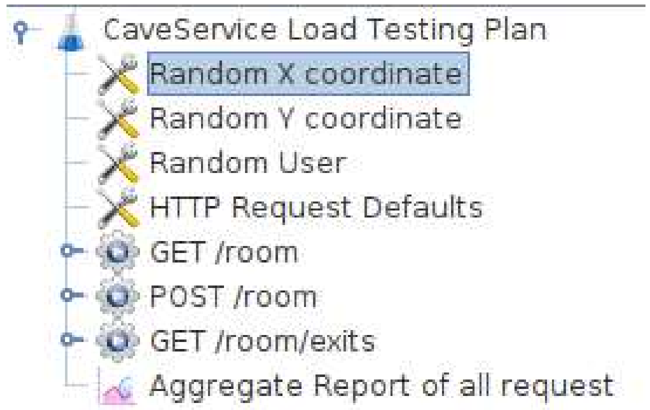
Gin Web Framework

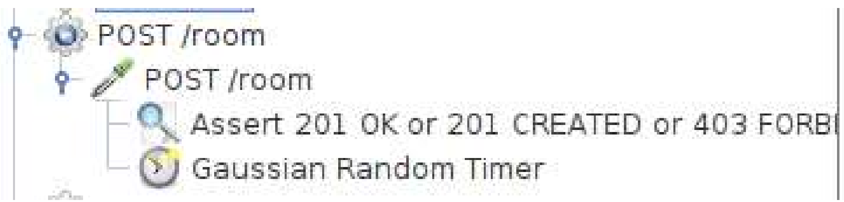
Waitress

Each one required learning a language PLUS learning a WebServer framework (SparkJava, Gin, Scalatra, Waitress)!

# Experience

- Reproducible Lab
  - How to run an experiment 1.000 times in controlled manner?
    - **Load Generators**, like Apache JMeter + load scripts
    - I was fortunate that the course already had introduced that…

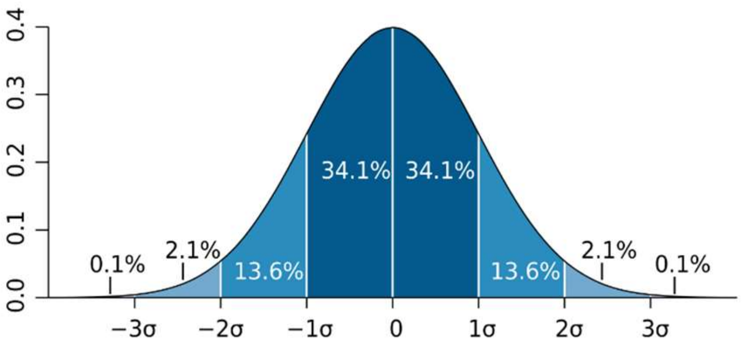


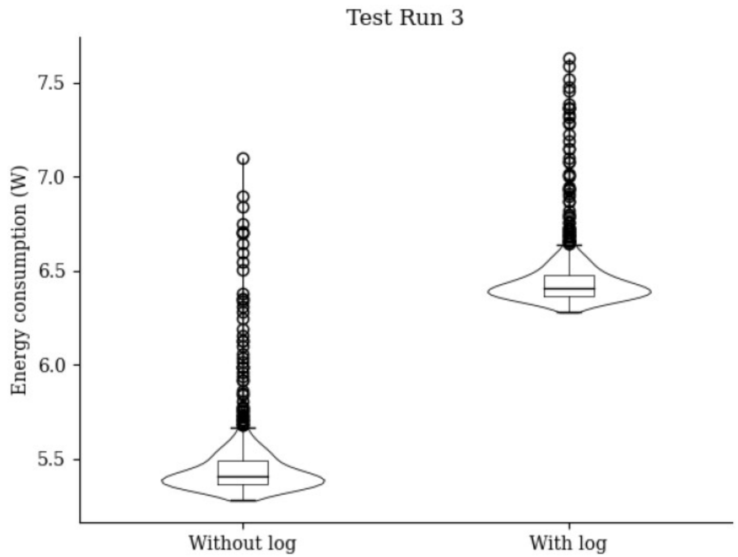




- Tooling is heavy
  - Getting to know JMeter, powerstat, statistics tools, plotting libraries, …

# Experience

- IT people do not know how to make experiments ☹
  - This is **physics:** Measuring variables in a system
    - Central limit theorem (= do *many* experiments)
    - Do statistics analysis
      - Std dev, violin plots, Welch T-tests, …



# Conclusion

- I think my main points are
  - **It is important!**
    - **The house is on fire!**
    - And software has to contribute
  - It is an eye-opener
    - *Don't do Python if you love your kids ☺!*
  - It has a quite labor intensive do to well… ☹
    - Ideally, you will benefit from a dedicated lab
      - But a room, two old PC's and a switch, and a booking plan will do…
  - … and I would be happy to exchange ideas
    - hbc@cs.au.dk