Energy-Efficient Software Architecture – For Developers

Henrik Bærbak Christensen
Associate Professor, Computer Science, Aarhus University
hbc@cs.au.dk

Motivation…

goto;

- Well?
  - Heard of a climate crisis, right? ***We need to do something !***
- Context:
  - I am a teacher foremost – *practical design principles that I can apply in my design and coding that reduce energy spending…*
    - *And teach my students to apply in their design and coding*
- Example of a *tactic*: ***Accept Lower Fidelity***
  - Aka ‘do not develop/use features that waste energy’

Example

- Imagine a conference session;
  - you want to ask a question!

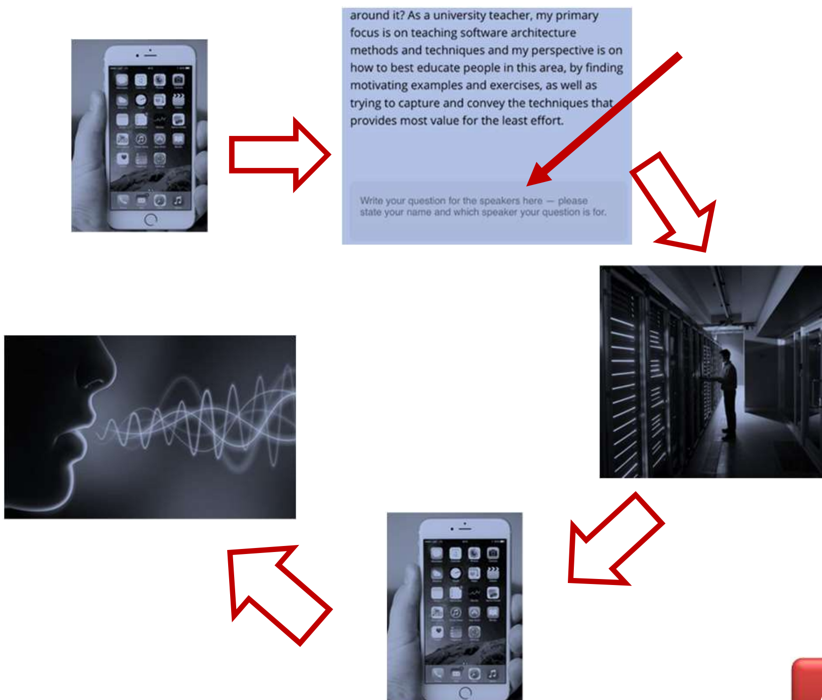
Solution A

- We could – *develop an app…*
  - Spending energy on multiple phones, networks, servers…

Solution B

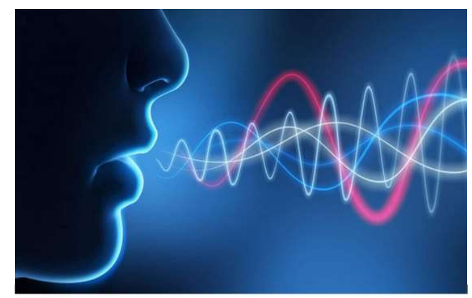
goto;

- Or We could: Just ask…
  - Spending rye-bread energy…



A

B



Tactic:
Accept lower fidelity
Avoid Feature Creep

Motivation

- So – talking about …

  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 literatu

- I will delimit myself to *energy-efficiency*

  Energy conversion efficiency (η) 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, η (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	$kg \cdot m^2 \cdot s^{-2}$
CGS units	1×10^7 erg
watt-seconds	1 W·s
kilowatt-hours	$\approx 2.78 \times 10^{-7}$ kW·h
kilocalories (thermochemical)	2.390×10^{-4} $kcal_{th}$
BTUs	9.48×10^{-4} BTU
electronvolts	$\approx 6.24 \times 10^{18}$ eV

100g Hellmann's Mayonnaise contains 2,965,000 J.
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

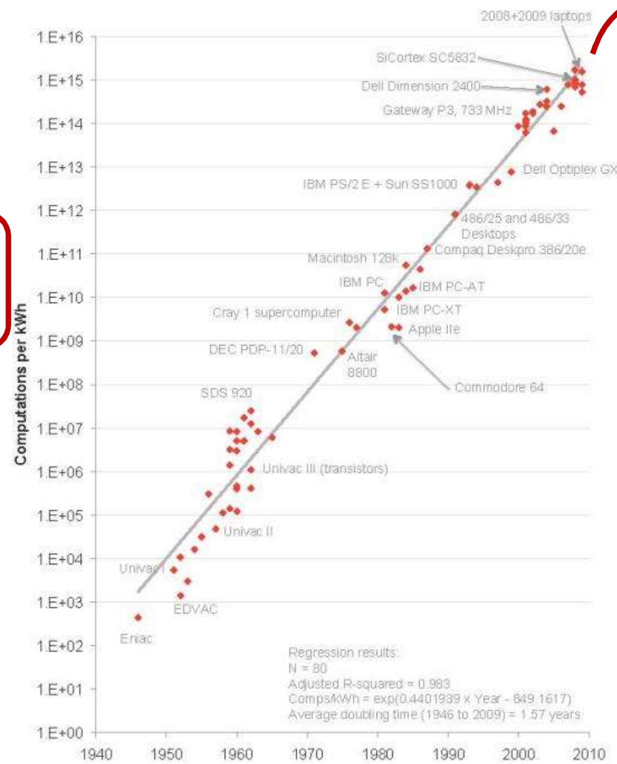
Hardware spends energy, because our *Software* wants work to be done.

Koomey's Law

- Hardware consumes energy
  - But is improving all the time!
    - They are the good guys!

Koomey's law describes a trend in the history of computing hardware: for about a half-century, the number of computations per joule of energy dissipated doubled about every 1.57 years. Professor Jonathan Koomey described the trend in a 2010 paper in which he wrote that "at a fixed computing load, the amount of battery you need will fall by a factor of two every year and a half."[1]

This trend had been remarkably stable since the 1950s (R^2 of over 98%). But in 2011, Koomey re-examined this data[2] and found that after 2000, the doubling slowed to about once every 2.6 years. This is related to the slowing[3] of Moore's law, the ability to build smaller transistors; and the end around 2005 of Dennard scaling, the ability to build smaller transistors with constant power density.



- Intel 12th Gen CPU

Within each model of 12th-generation Intel CPU, you'll find E-cores (Efficiency) and P-cores (Performance) in the CPU package. The relative numbers between these two types of core can vary, but the full Alder Lake CPU die has eight P- and eight E- cores, which is found in the i9 CPU models. The i7 and i5 models have an 8/4 and 6/4 design for P- and E- cores respectively.

Wirth's Law

goto;

- Unfortunately, we as developers and architects are terrible at writing software or writing too much ☹
  - We are the bad guys!

Wirth's law is an adage on computer performance which states that software is getting slower more rapidly than hardware is becoming faster.

The adage is named after Niklaus Wirth, a computer scientist who discussed it in his 1995 article "A Plea for Lean Software".[1][2]

- Example:
  - For my students I like an easy, but small, linux desktop: *Lubuntu*
  - First used in 2016, easily ran in a 2GB RAM VM ☺

lubuntu-16.04.6-desktop-amd64	17-03-2023 14:14	Disc Image File	954.368 KB

  - Last 22.04 version, has issues running in a 4GB RAM VM ☹

lubuntu-22.04-desktop-amd64	20-05-2022 10:48	Disc Image File	2.545.182 KB

  - And – *In the old days*

Win98InstallCDImage	11-10-2007 09:36	Disc Image File	114.758 KB

What is using Power?

goto;

- 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

~ 211W

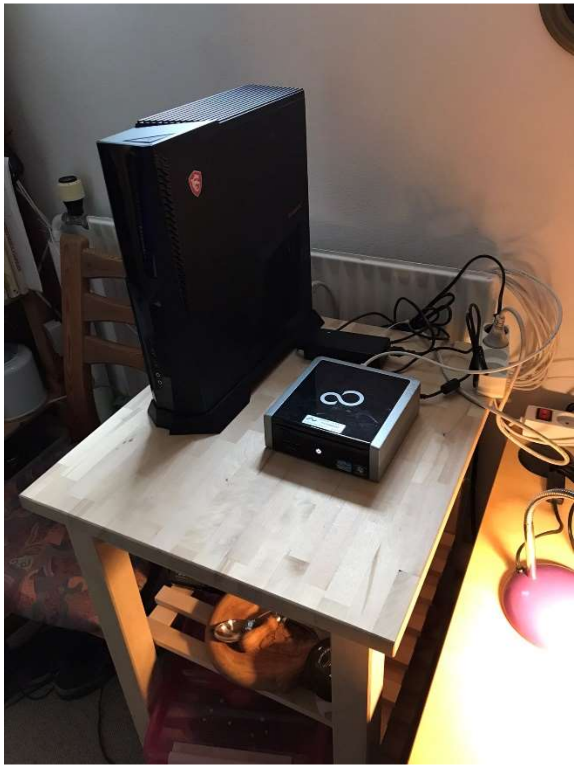
~ 18W

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

Examples: My Humble Lab

goto;

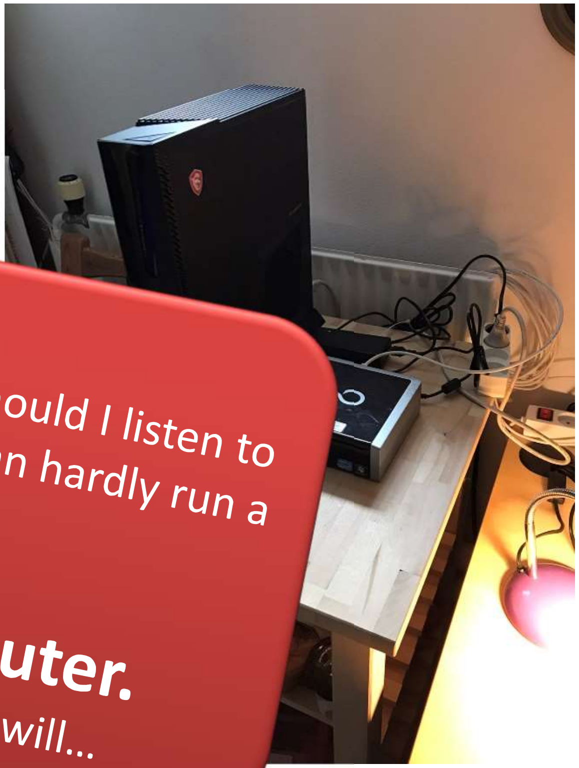
- The Lab
  - Fujitsu Esprimo Q900 (2012)
  - MSI Trident (2020)
- Installed with Ubuntu 22.04 LTS
  - Headless
    - No use for the GeForce RTX™ 2080 Ti ☺
- **Idle** Power Consumption
  - Esprimo: ~ 11 W (plug) / 2.8 W (CPU)
  - MSI: ~ 40 W (plug) / 7.4 W (CPU)
    - At ~95% CPU load@Plug: Esprimo 43W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



Examples: My Humble Lab

goto;

- The Lab
  - Fujits… Q900 (2012)
  - MSI
- Insta…
  - H…
- **Idle** Powe…
  - Esprimo: ~ 11 W (plug) / 2.…
  - MSI: ~ 40 W (plug) / 7.4 W (CPU)
    - At ~95% CPU load@Plug: Esprimo 43W and MSI 160W



Come on, Henrik???

I run 100.000 instances in the cloud, why should I listen to your experiments on a decade old PC that can hardly run a Windows 10 OS ???

Well – **A computer is a computer.**

The data may not transfer, but the trend will…

So… Green Architecting?

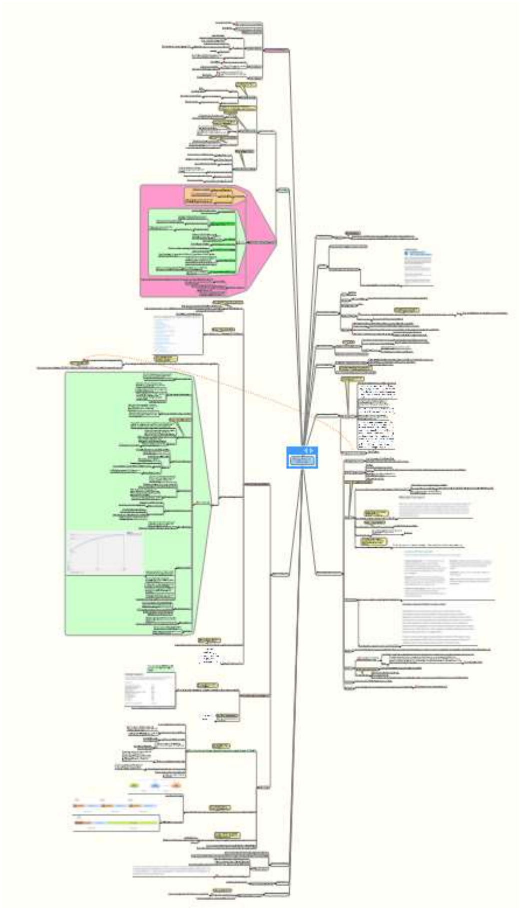
How do I then design my architecture and code, so it spends less energy?

It is a Vast Field…

- Lots of literature ☹
  - And I am still a novice
    - So there is probably a lot of essentials out there that I am missing…

- But – there are some central lessons and tactics…

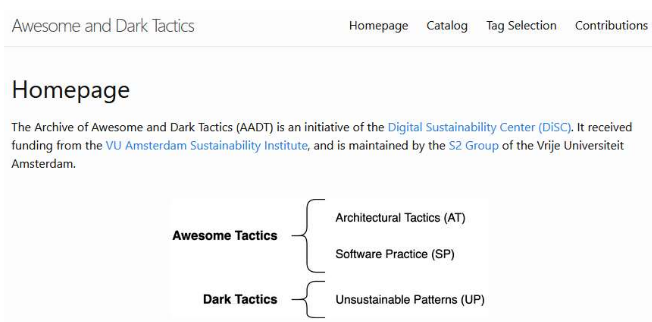
- … that I have tried to distill…

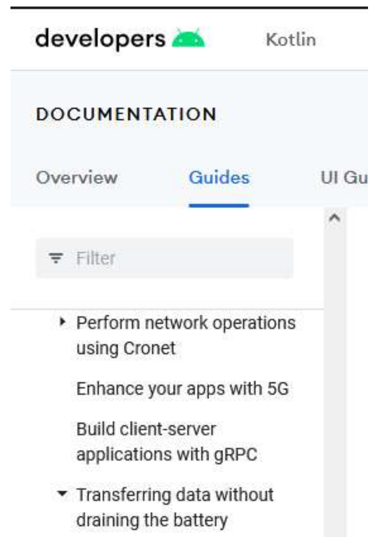


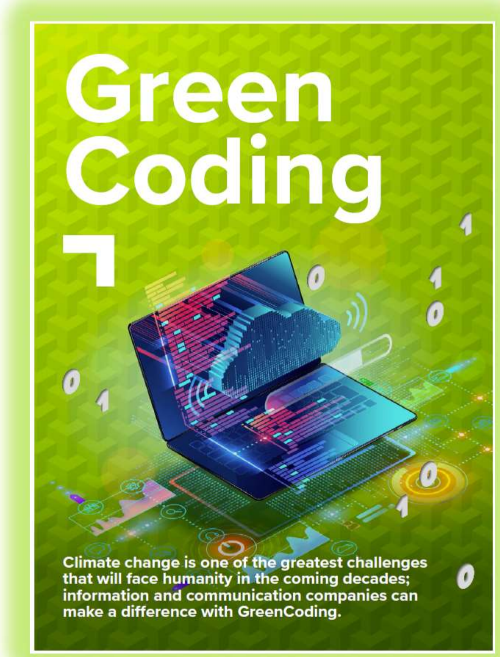
My Primary Inspirations

goto;

- The most interesting I have stumbled upon…





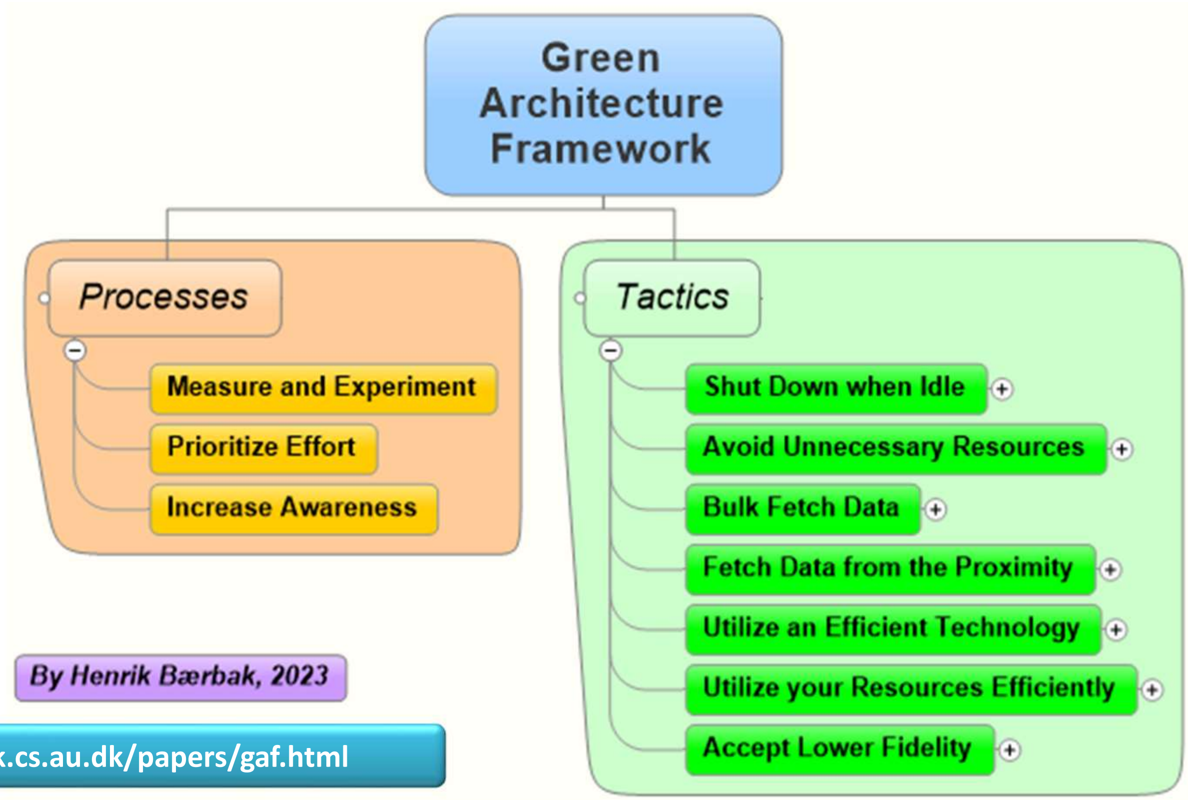


Principles of Green Software Engineering

The Framework

goto;

- *The Green Architecture framework ☺*



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

Processes

How we design Green Architectures?

Measure and Experiment

goto;

- *You need to measure!*

> percent). The lesson is that you can't manage it if you don't measure it!

- *You need to experiment!*

> - We can, with a small effort in experimentation and prototyping, and small design changes, substantially improve an application's energy use.

Tactics: Bulk Fetch Data
+ Low Foot-print Data Formats

Managing Energy Consumption as an Architectural Quality Attribute

Rick Kazman, Serge Haziyev, Andriy Yakuba, and Damian A. Tamburri

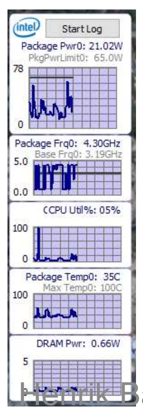
SEPTEMBER/OCTOBER 2018 | IEEE SOFTWARE

Table 2. The differences between the experiments.

Setup	Description	Consumption per hour (Wh)	Total energy savings (%)
Original	Plaintext payload and a 15-min polling interval	0.0998	0
Experiment 1	Binary format	0.0917	8
	Binary format + bug fix	0.0527	47
Experiment 2	A polling interval of 1 h	0.0137	$\boxed{86}$

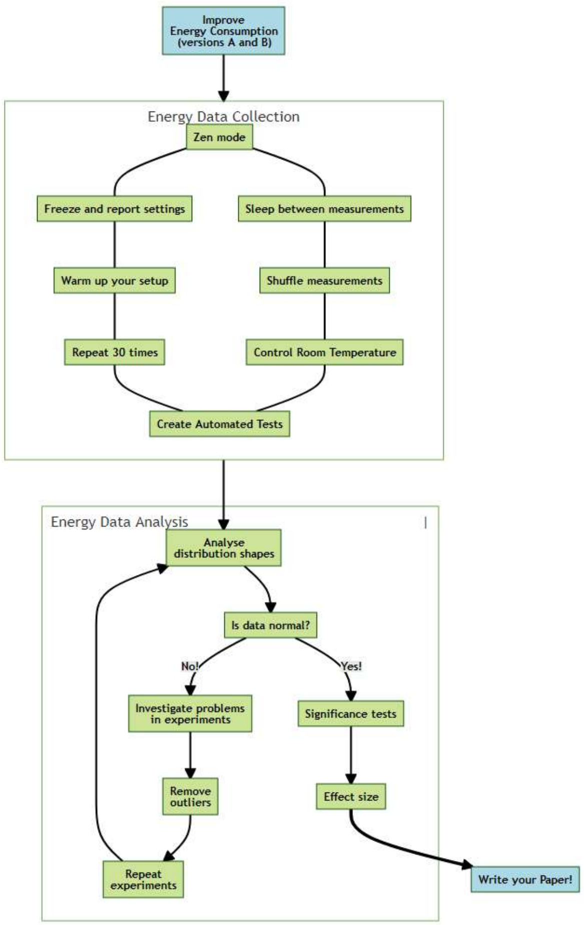
Measurements

- Measure *the wall power*
  - "Absolute truth"
    - I use a 'Nedis smart plug'
      - Manual read out ☹
- Measure the 'on-chip' power
  - **RAPL**: *Running Average Power Limit*
    - *Only CPU (and DRAM) is measured*
- Virtual Machines?
  - No luck!
    - Cost correlate ☺



Be Systematic

- As in Physics
  - *Control the environment, reduce error sources*
  - *Make many experiments, large sample size*
  - *Use proper statistical methods*



- Luiz Cruz (2021)
  - https://luiscruz.github.io/2021/10/10/scientific-guide.html

Prioritize Effort

goto;

- Prioritize Effort
  - Know the usage profile
    - (by measurements ☺)
  - And invest your effort where it counts
    - Those user stories that are executed the most and that can be optimized the most – are the ones to spend your effort on optimizing
      - Sounds reasonable, but you need to know the usage profile ☺

Increase Awareness

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

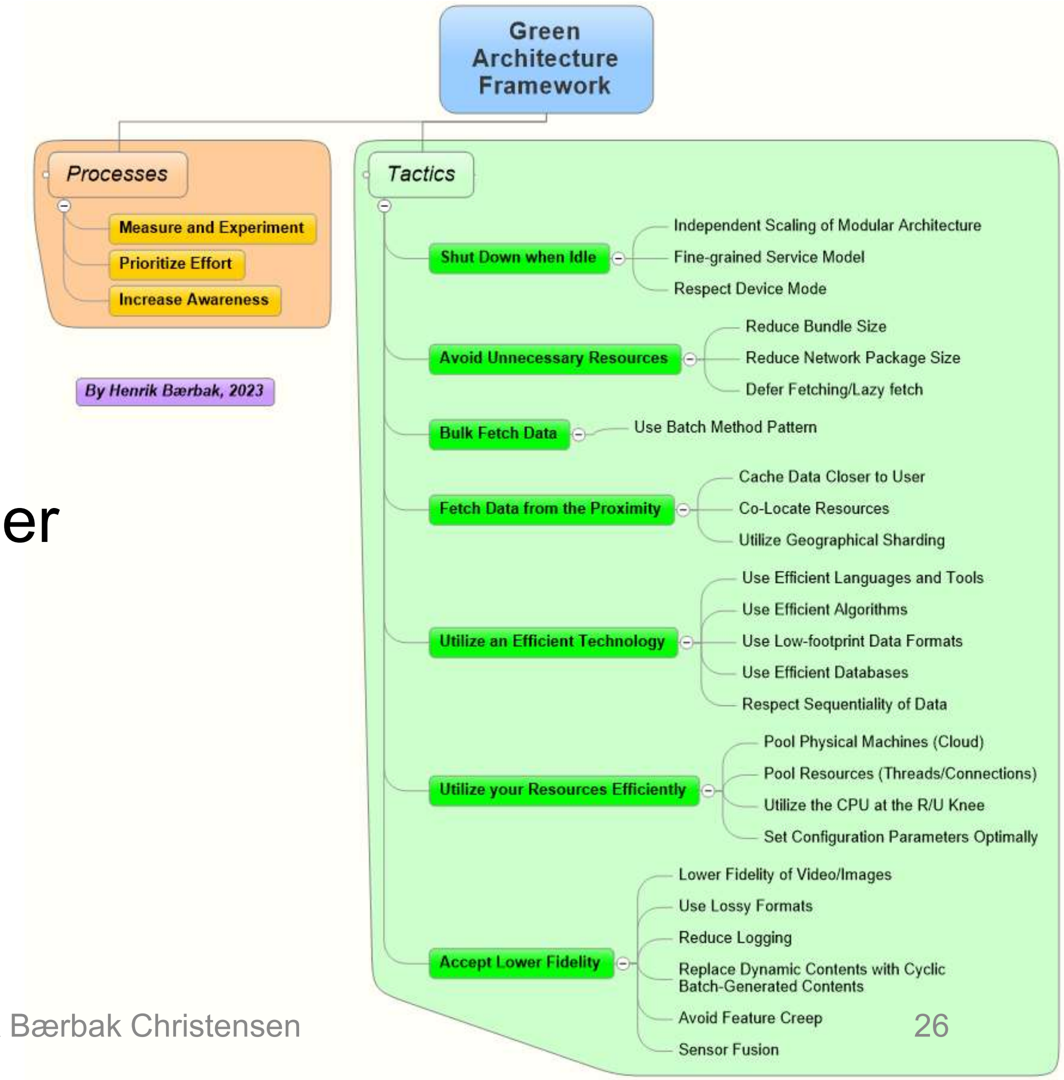
- From my kitchen. Which one is 2W and which 40W?
  - You have to tell the kids which one to prefer ☺

Tactics

How do we then *do* Green Architecting?

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

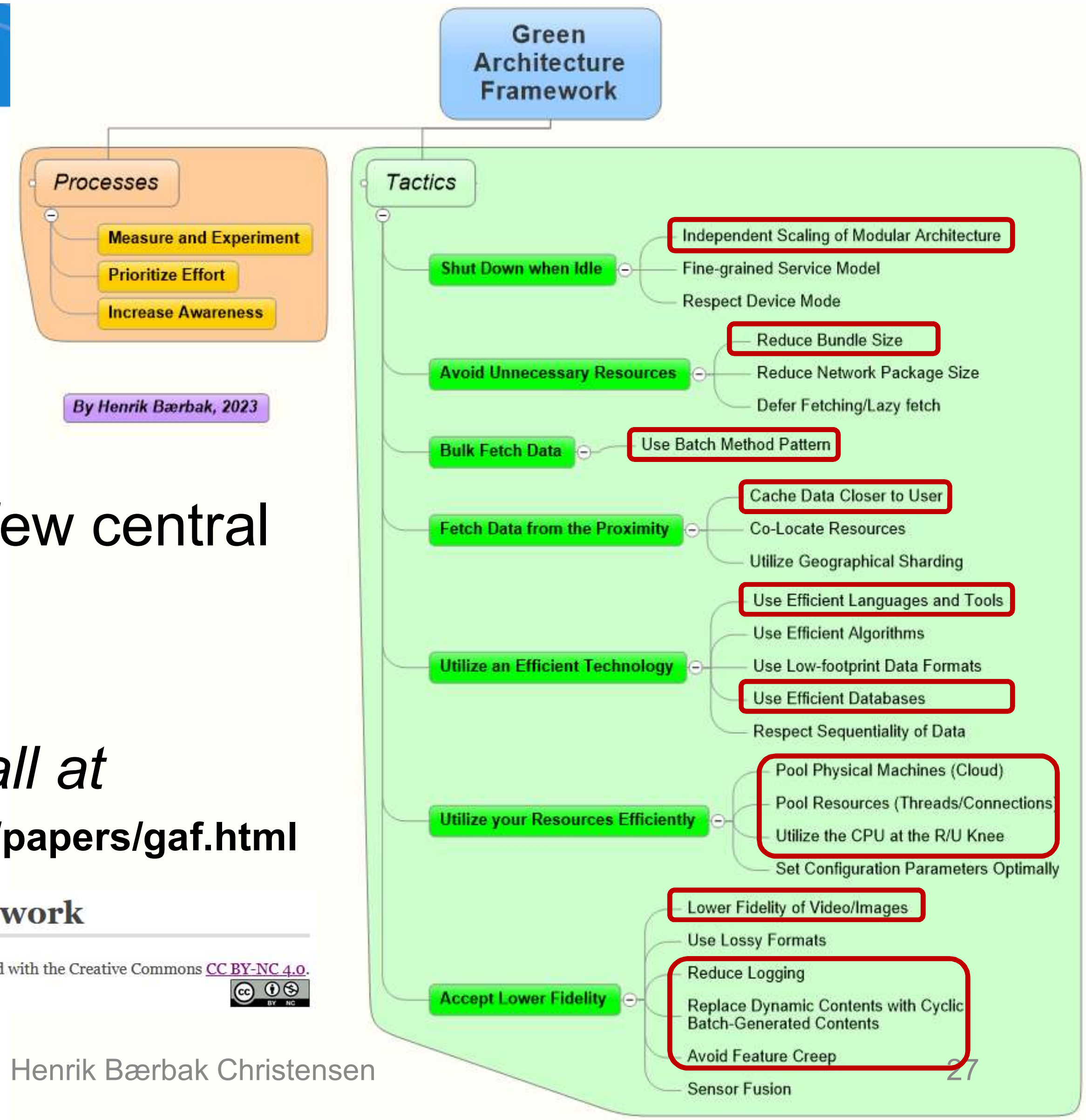
- Quite a lot of tactics under *seven main categories*



- I will only discuss a few central tactics…

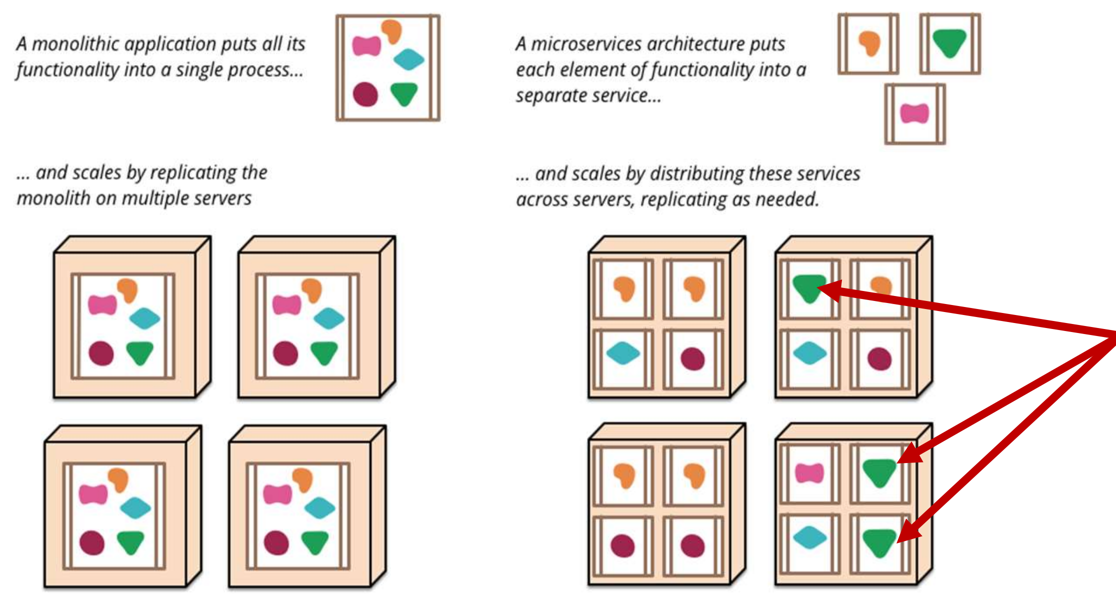
- *Find draft paper on all at*
  - **https://baerbak.cs.au.dk/papers/gaf.html**

The Green Architecture Framework



Shut Down when Idle

- *“Turn off the lights in the bathroom, when you leave”* ☺
- **Independent Scaling of Modular Architecture**
  - Microservices versus Monolith



Turn off VMs when not needed

Shut Down when Idle

- *“Turn off the lights in the bathroom, when you leave”* ☺
- **Independent Scaling of Modular Architecture**
  - Microservices versus Monolith
  - *Elasticity* – adjust services to current load

Horizontal Pod Autoscaling

In Kubernetes, a *HorizontalPodAutoscaler* automatically updates a workload resource (such as a Deployment or StatefulSet), with the aim of automatically scaling the workload to match demand.

If the load decreases, and the number of Pods is above the configured minimum, the HorizontalPodAutoscaler instructs the workload resource (the Deployment, StatefulSet, or other similar resource) to scale back down.

Shut Down when Idle

- *"Turn off the lights in the bathroom, when you leave"* ☺
- **Independent Scaling of Modular Architecture**
  - Micr… versus Monolith
  - Ela… …rrent load

Ho…

In K… (suc… to r… …the …fulSet,

Beware!
If you do *not* use that scaling…
Experiment: 'PizzaLand' [two bounded contexts]
Monolith: 20.3 mJ pr REST call
Microservice: 36.6 mJ pr REST call
Overhead of 80.3% (x1.8 more energy)

Avoid Unnecessary Resources

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

```
FROM adoptopenjdk/openjdk11:alpine-jre
```

173MB

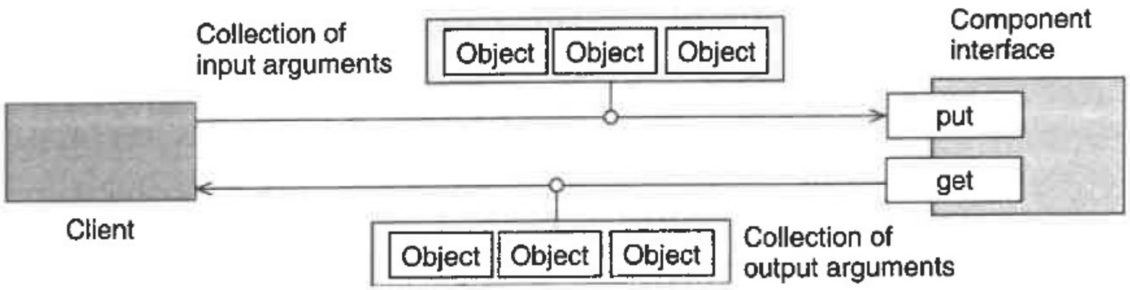
```
FROM henrikbaerbak/jdk11-gradle68
```

924MB

  - That is: 4.3 times bigger image to transfer and load ☹
    - For the exact same server in java 11…
- Many other examples
  - Javascript tree-shaking, ProGuard, GraalVM, *network payload…*

Bulk Fetch Data

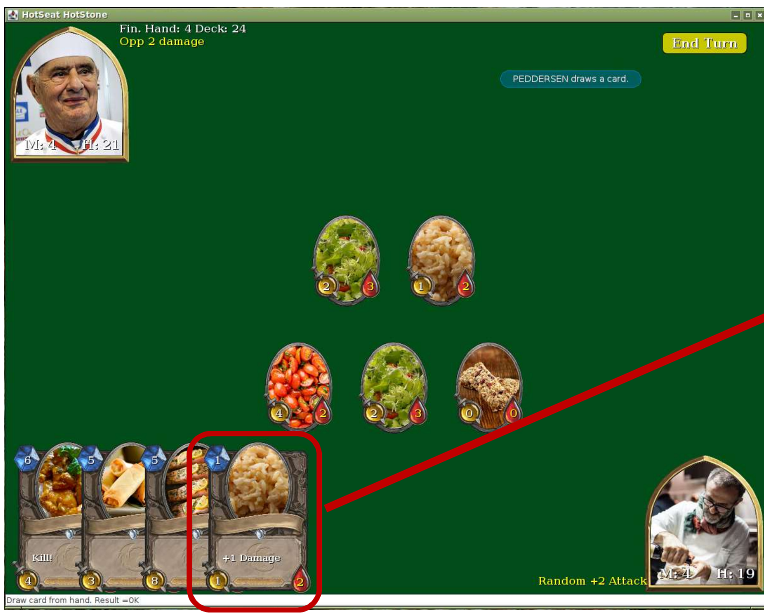
- *"Buy 50 things at the super market once, instead of making 50 trips buying a single thing"*
  - POSA4(2007): **Batch Method**



- **Iterator pattern** 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

- *"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 card data from the server when redrawing UI…

Example of A/B Architecture

goto;

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

```
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)

```
@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

```
@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(); }
```

```
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
    - 4.12W (σ 0.79W)
    - (Reducing number of network calls to 43%)
  – Saving **27% energy**

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

Fetch Data from the Proximity

- *"Have a stock of supplies to avoid a lot of trips to the super market"*

- **Cache Data Closer to User**
  - The *Batch Method Broker* is one such example
  - Content-Delivery-Networks (CDN)
    - Store web contents (caching) physically near to the users to provide faster load times by avoiding "long distance network transmission"

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

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

goto;

- *“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

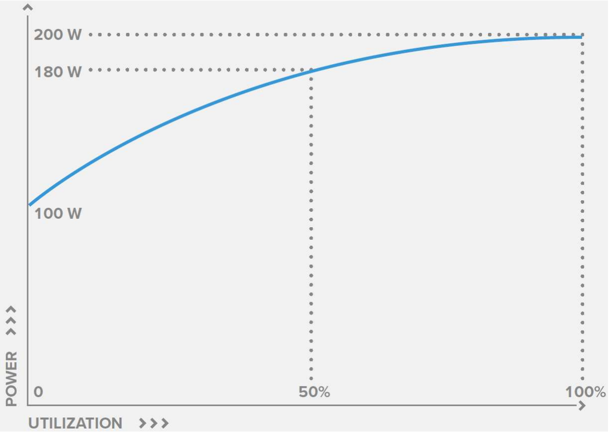
– Fake in-memory db:	~ 11.8W σ 0.3W	(- 44.6%)
– Redis db:	~ 14.7W σ 0.3W	(- 31.0%)
– Mongo db (naive):	~ 21.3W σ 0.5W	(baseline)
– Mongo db (optimized):	~ 20.9W σ 0.2W	(- 1.9%)

Utilize your Resources Efficiently

goto;

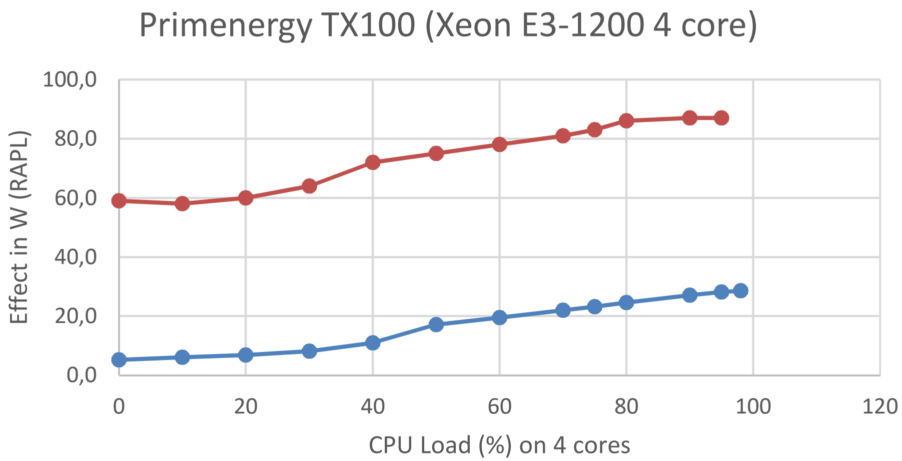
- *"Prepare several items in the oven at the same time"*
- An *idling* computer spends between 1/4 - 2/3 power compared to a *busy* computer
    - The *non-proportionality of energy consumption*

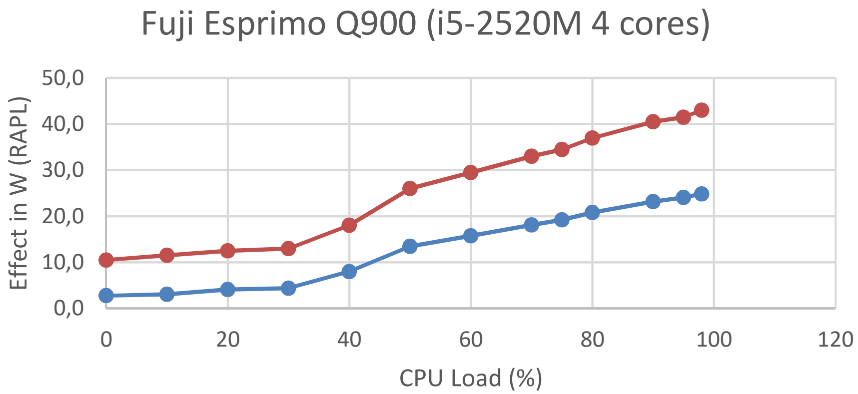
- Which means:
    - **Per-transaction energy cost is lowering as the computer is more heavily utilized**



Utilize your Resources Efficiently

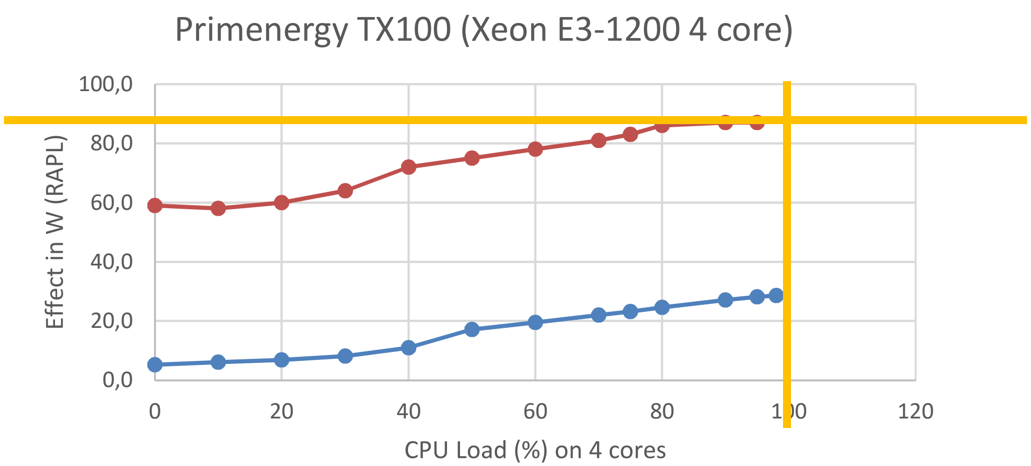
- *"Prepare several items in the oven at the same time"*
- My own measurements
  - Red Line = on-the-wall power
  - Blue Line = on-chip power (RAPL)





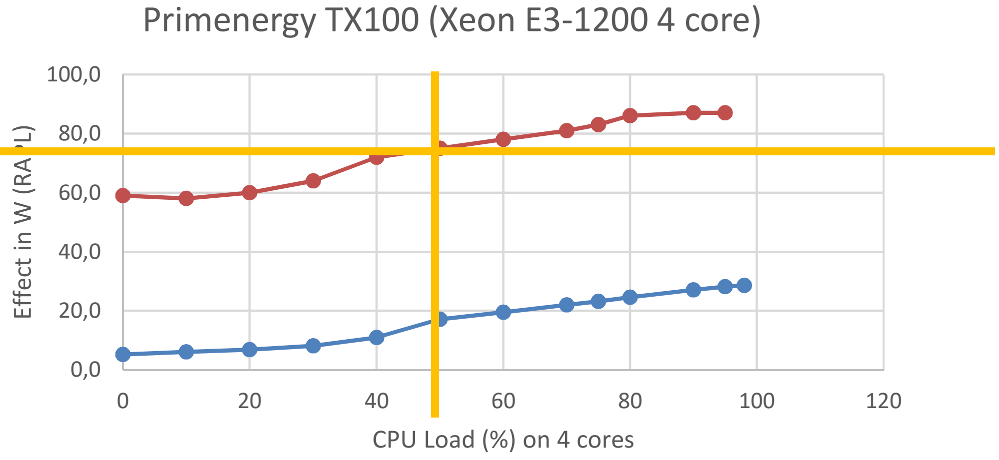
Utilize your Resources Efficiently

- *"Prepare several items in the oven at the same time"*
- Imagine a *single* server
  - Handling 2.000 tps at 100% CPU load
    - Result: 2.000 tps spending **90 W**



Utilize your Resources Efficiently

- *“Prepare several items in the oven at the same time”*
- Change to *Horizontal Scaling: Two servers*
  - Handling 1.000 tps **each** at 50% CPU load
    - Result: 2.000 tps spending 2 x 75 W = **150 W**

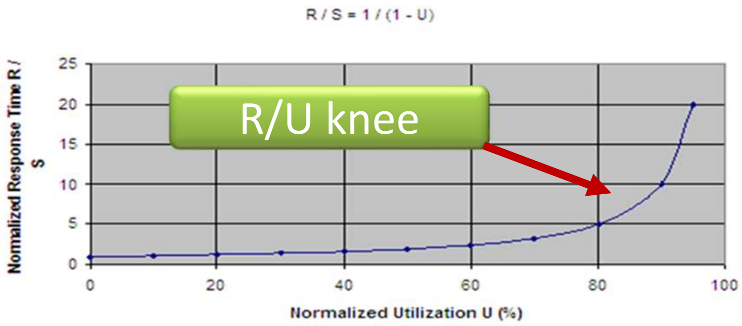


Morale: Make your CPU as much work as possible!
90 W versus 150 W

Utilize your Resources Efficiently

goto;

- *"Prepare several items in the oven at the same time"*
- **Utilize the CPU at the R/U knee**
  - Queue Theory:
    - Response time gets *very* long when we approach 100% CPU load
      - R = S/(1-U)
        - » R = 10ms at 0%
        - » R = 50ms at 80% **(5x !)**
- So…
  - Max CPU load while having reasonable response times means
    - CPU around 70% - 95% (depending on…)



Utilize your Resources Efficiently

goto;

- *“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 29% less energy**.

Accept Lower Fidelity

- *“Turn the room temperature down from 21° to 19°”*
- **Replace Dynamic Contents with Batch**
  - Change webpage dynamic content (expensive) with *batch once-per-hour (or per-day) computation* of a static webpage (cheap)

- **Lower Fidelity of Video/Images**
  - Use 720p instead of 1080p (halves the size)
  - Downscale images server side
    - Use JPEG rather than GIF/PNG

Accept Lower Fidelity

- *"Turn the room temperature down from 21° to 19°"*
- **Reduce Logging**
    - Do we *really* need all that data with 30 log msg for every method call?
    - ELK stacks are notoriously power hungry!

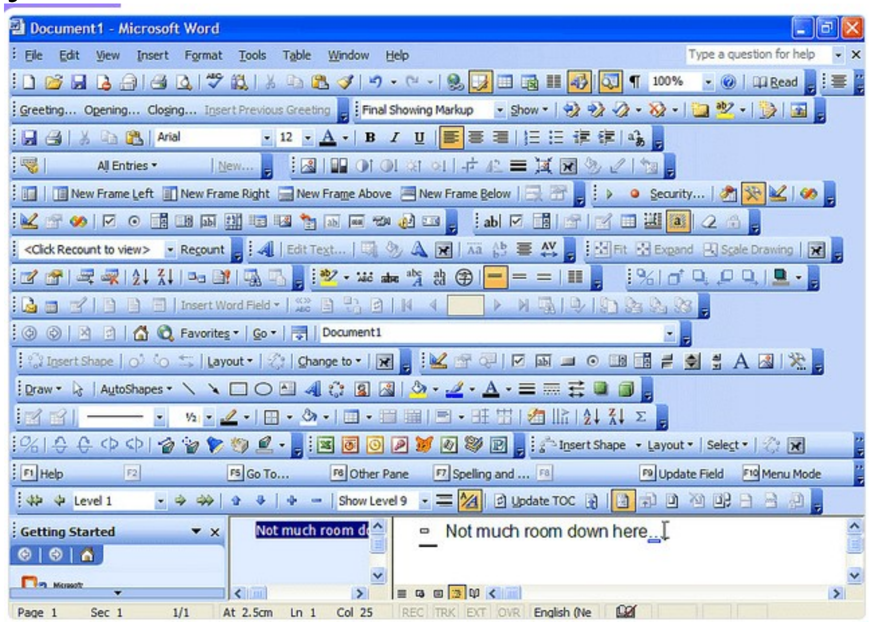
Own experiment: -11.6% energy by remove logging on a simple REST service (3 endpoints)

– But this is a really hard trade-off with 'monitorability' QA
    - When the 267 servers crash, the one important piece of info that you need to understand why – is just the one log message, you *did not make* ☹

Accept Lower Fidelity

goto;

- *"Turn the room temperature down from 21° to 19°"*
- **Avoid Feature Creep**
  - Do we really need it all???



Accept Lower Fidelity

- **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**

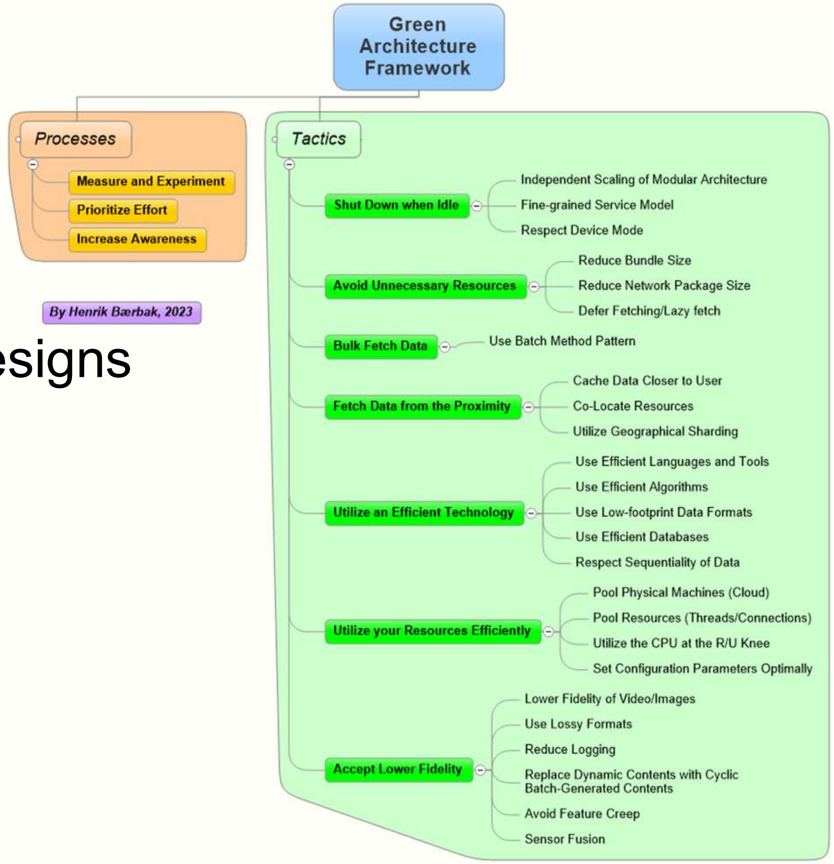
Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received K...	Sent KB/sec
GET /order	779170	4	1	2	3	95	0	283	0.00%	485.2/sec	218.99	62.55
POST /order	23180	37	24	76	118	215	9	414	0.00%	14.4/sec	4.19	3.76
POST /finish	23084	6672	6514	12010	13349	15160	44	24888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

Discussion

Summary

goto;

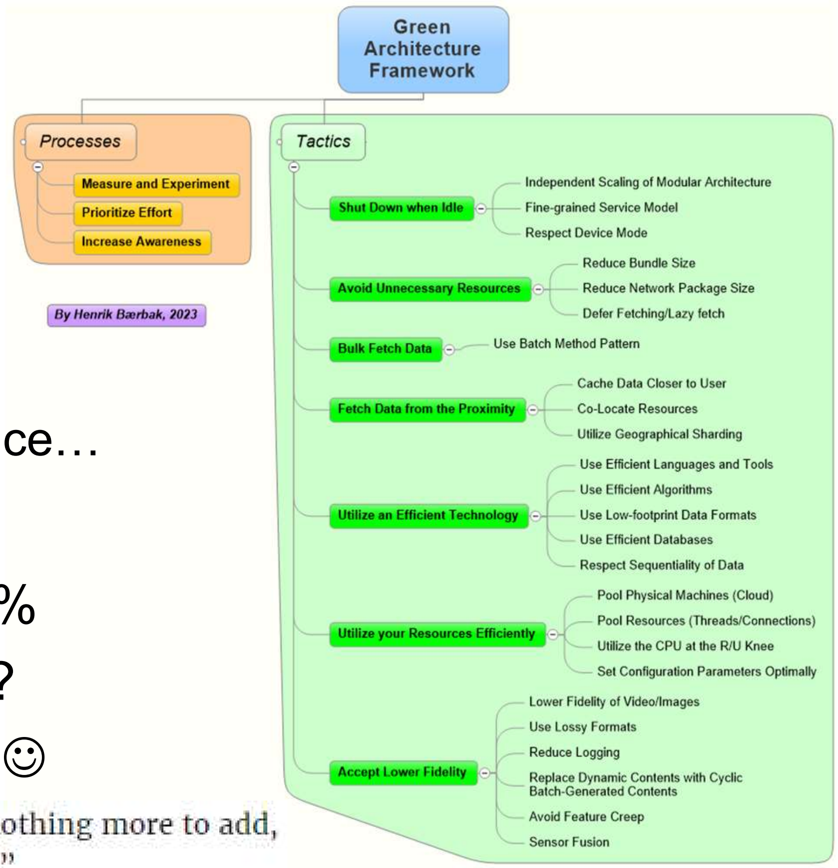
- Tactics
  - Design decisions
    - To reduce energy consumption
  - Categories cover a lot of more specific decisions to make and designs to explore
- Remember
  - *Experiment and measure !!!*
    - The obviously good design may turn out to be a bad design when put to the test…



Summary

goto;

- However, the all require an investment
  - *More complex code*
    - Batch Method took some time!
    - Rewriting code base to Go also
    - Or change monolith to microservice…
- Low Hanging Fruits (?)
  - Get utilization of CPUs up to ~75%
  - Do you really need all those logs?
  - Start learning Go, C++, Java, …, ☺
  - ARM?



"Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away."

— Antoine de Saint-Exupéry, Airman's Odyssey

Would you like to know more?

goto;

Grab a course at Master of IT

- Software construction
- Cyber Security
- IT-architecture
- Digital transformation
- Data Science
- IT-management

Read more

Winner of the Informatics Europe 2020 Best Practices in Education Award

AARHUS UNIVERSITET
AALBORG UNIVERSITET
SDU

Don’t forget to
vote for this session
in the **GOTO Guide app**