adesso

Wirth's Law in the Age of Cloud and Green IT

Addressing Software Complexity for Sustainable Performance

Speaker

Christian Dähn

Chief Architect for Government Clouds

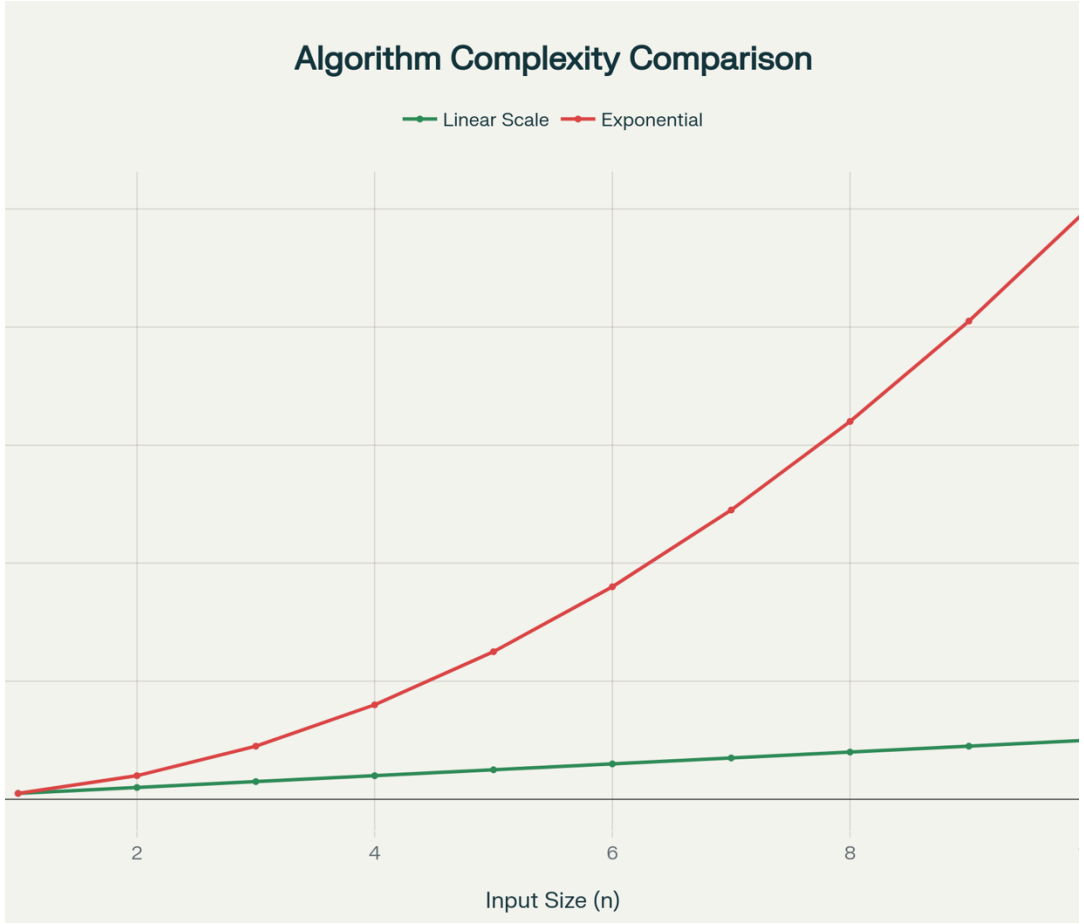
Senior C++ Developer
Industrial OCR and image analysis systems
AI & ML developer
Embedded development (ASM, C/C++)

Wirth's Law

Software gets slower more rapidly than hardware becomes faster.

Wirth's Law & The Lost Linearity

THESIS: The Past: Linear scaling with complexity: O(n) vs. Today: Exponential complexity: O(n^2)



The Past:

Software in the 90‘s on DOS machines = single threaded, nearly linear dependency to cpu clock rate

e.g.: DOS Game on 486 @ 33 MHz has 25 fps, but >50 fps @ 66 MHz

Today:

Software often multithreaded, multitasking OS, runs on CPU & GPU = overhead by multithreading + abstraction layers + async I/O

Why the Complexity Exploded

- **Multithreading Overhead:**
  - more CPU cores != more speed
  - more coordination, more waiting for locks, more 'management' work by the operating system
- **Abstraction Layers:**
  - e.g. frameworks, containers, virtual machines
  - additional computational costs per layer – often with complexity O(n^2)
- **Distributed Systems:**
  - interprocess & network communication (with serialization and en/decrypting overhead)
  - RPC and microservice architectures add unpredictable latency

My First Story: The Complexity Trap

Project: PHP-based worktime tracking and work shift management system of a government agency with >10.000 users.

- Starting point: 1 Server → Fast App with < 1s response time
- After 8 years: 24+ Servers → Slow App with > 10s response time

First examination results:

Issue #1: SQL Query with 12.000 lines!
Issue #2: PHP-Monolith with code inside MySQL DB
Issue #3: Each session reads all user data on each request of the UI
Issue #4: Nobody ever checked for slow queries and architecture flaws

Key problem:

Architecture with no segregation of responsibilities and no support for (real) concurrent workloads.

My Second Story: The Cloud Shock

Project: Java-based self-written ERP-System with >5.000 users migrated from on-premises VMs to cloud with „shift-left“ approach.

- Starting point: 5-10 Server on-premises → budget of 250.000 EUR/year and 4 admins
- After migration: all VMs cloud hosted → bill of 1.2 Mio EUR/year for cloud + some on-premises servers + 4 admins

First examination results:

Issue #1: High RAM Usage of Java-App, memory leaks of Tomcat
Issue #2: High idle CPU load
Issue #3: Default Tomcat-Config & Spring (Boot) Setup
Issue #4: Full-blown Linux server OS inside VM

Key problem:

Lift-and-shift workloads often miss 72% potential CO2 reduction (and accordingly the targeted cost reduction).
→ see: 451 Research for AWS, "The carbon reduction opportunity of moving to AWS," 2021

The Deeper Cost:
The Green IT Connection

Pay-per-use means pay-per-waste.

Cloud: You pay for what you use

→ vCPU+vRAM+Storage+DB+Traffic accounted per minute

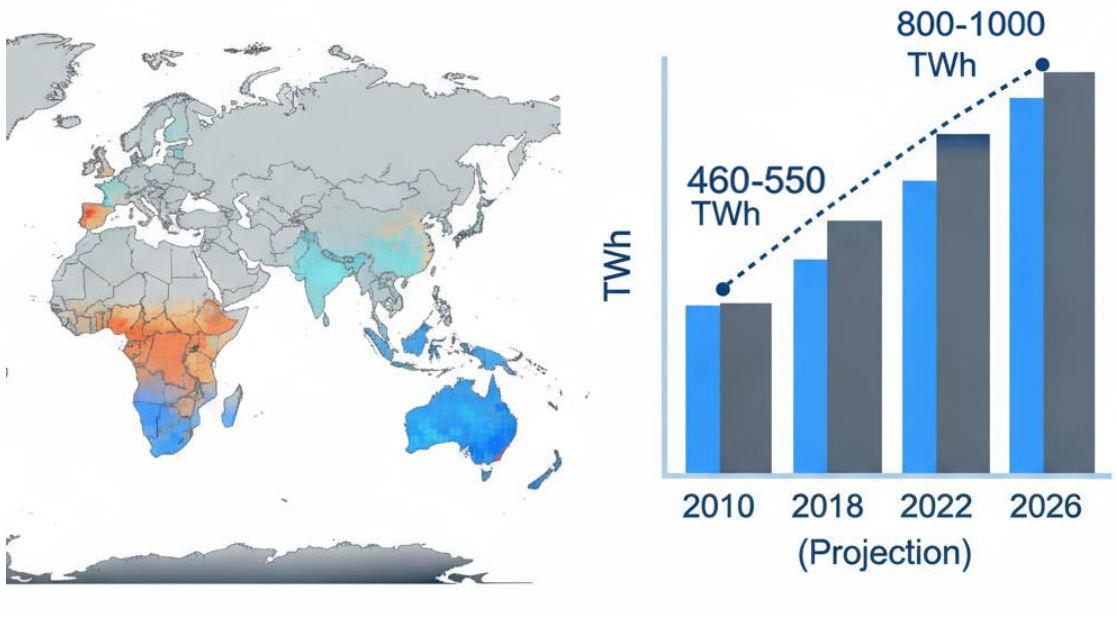
→ every component in cloud environments MUST be resource-efficient

→ software must be designed to prevent CPU load during idle state and inefficient code

Global Reality Check: Datacenter Footprint

Key statistics

- 2022 Consumption: 460–550 TWh (nearly France's electricity demand)
- Ireland: Datacenters use 18% of national grid (2023)
- Projection: Global load could double by 2026 without efficiency gains.



The macro backdrop for Wirth's Law:
Our software choices show up in national energy statistics.

Measuring Sustainability: PUE, SCI & Workload Efficiency

Key metrics

- PUE (Power Usage Effectiveness): Hyperscaler ~1.1 vs. on-premises ~1.6
- SCI (Software Carbon Intensity): `SCI=(E*I)/R` (kg CO2 per user action)
- Workload Efficiency: CPU utilization, autoscaling adherence, idle hours

Effects of workload tuning
Studies by McKinsey and Uptime Institute in 2023 show software-driven workload tuning delivers **10–30% energy and cost reductions**, even in efficient facilities.

Strategy 1: Analyze & Optimize

You cannot optimize what you do not measure.

- PUE (Power Usage Effectiveness): Hyperscaler ~1.1 vs. on-premises ~1.6
- SCI (Software Carbon Intensity): `SCI=(E*I)/R` (kg CO2 per user action)
- Workload Efficiency: CPU utilization, autoscaling adherence, idle hours

The Quick Win: Configuration is Code

- Tomcat Configuration
  - **Before:** Default Config → increasing RAM-usage = high costs
  - **After:** garbage collector & stack sizes optimized → costs reduced by over 60%, cpu-hours by almost 50%.
- SPRING framework
  - **Before:** full blown config with default settings
  - **After:** switched to Spring Boot with customized profile → reduced RAM-footprint by 25-30%, faster startup-times, lower memory costs

Strategy 2: Architect for Simplicity

The Architectural Fix: Modularization & CQRS for PHP app

Key Architectural Changes:

- Modularization:
  Breaking the monolith into smaller, independent modules.
- CQRS:
  Separating 'write' operations from 'read' operations for independent optimization.
- MVC & Template framework:
  Separating code for generating UI and business logic.

Result:

- From 24+ Servers -> 1 VM
- From Seconds -> Milliseconds (page load times)

Strategy 3: Modernize Platforms

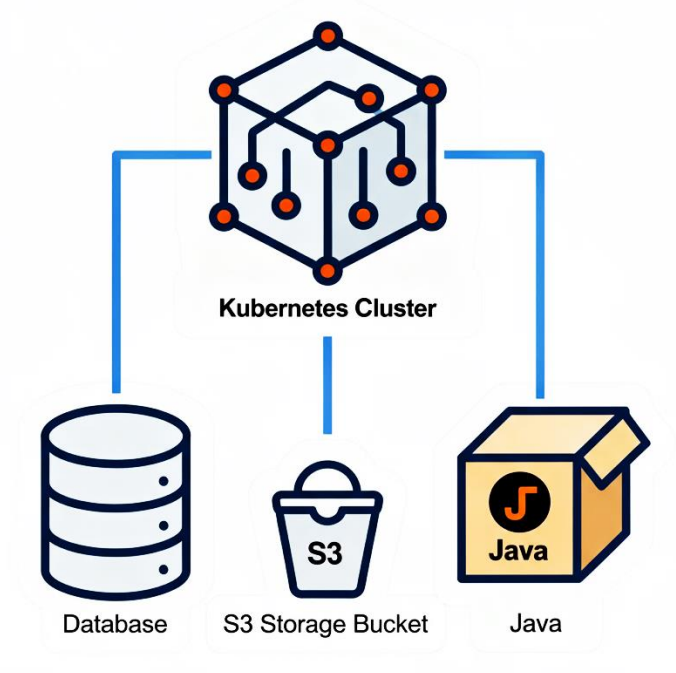
Leveraging Modern Technologies

Key Infrastructure Changes:

- Containerisation:
  Deploy the Java ERP as Containers in Kubernetes, instead of VMs
- Kubernetes & Stages:
  New Stages for Development, Testing and Production with same cluster config and auto-scaling
- Managed Services:
  Replace self hosted DB-, Web- and Storage-Servers by managed cloud-services like PostgreSQL-as-a-Service and S3

Result:

- Increased availability and service stability
- Price drop by 10-30% thanks to more efficient ressource management and cheaper managed services



Summary: Your Key Takeaways

Key Lesssons

Wirth's Law is more relevant than ever.

In the cloud, pay-per-use means pay-per-waste.

We must measure, not guess.

Configuration is code.

More hardware is a band-aid, good architecture is the cure.

A Challenge For You

Three Questions for Your Projects

Do we measure?

Is our architecture helping us, or hurting us?

Are we efficient with our organisation's money and our planet‘s resources?

Thank You

Time for your questions

Bibliography

Foundational Concepts

> Wirth, N. (1995). A Plea for Lean Software. IEEE Computer, 28(2), 64–68.
> Basili, V., Briand, L., & Thomas, W. (2010). Understanding and Managing Complexity in Software. Journal of Systems and Software, 83(2), 207–217.

Software Complexity & Architecture

> Naumann, S., Dick, M., Kern, E., & Johann, T. (2011). Green Software Engineering: Supporting Sustainable Software Development. IEEE IT Professional, 13(1), 43–50.
> Zimmermann, O., & Schär, S. (2021). Sustainable Software Architecture: Analyze and Reduce Technical Debt. Springer.
> Evans, E. (2003). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley.
> Fowler, M. (2011). CQRS. martinfowler.com.
> Kruchten, P., Nord, R., & Ozkaya, I. (2006). Technical Debt: From Metaphor to Theory and Practice. IEEE Software, 29(6), 18–21.

Datacenter Energy & Carbon Footprint

> International Energy Agency (2024). Electricity 2024. IEA Publications.
> International Energy Agency (2024). Data Centres and AI: Tracking Clean Energy Progress. IEA Publications.
> Masanet, E., Shehabi, A., Lei, N., Smith, S., & Koomey, J. (2020). Recalibrating Global Data Center Energy-Use Estimates. Science, 367(6481), 984–986.
> Central Statistics Office Ireland (2024). Information Sector Electricity Consumption 2023.
> Energinet (2024). Electricity Consumption by Sector.
> Uptime Institute (2023). Global Data Center Survey 2023.
> Google (2023). Data Centers: Efficiency. sustainability.google.
> AWS (2022). Sustainability Report. Amazon Web Services.
> SPEC (2023). SPECpower_ssj2008 Benchmark Results. Standard Performance Evaluation Corporation.

Bibliography

Sustainable Architecture & Operations

> Harman, M., Lakhotia, K., Singer, J., & White, D. (2021). Cloud Engineering for Sustainability. IEEE Transactions on Software Engineering, 47(4), 658–673.
> McKinsey & Company (2023). Laying the Foundation for Zero-Carbon Digital Infrastructure.
> 451 Research (2021). The Carbon Reduction Opportunity of Moving to AWS. Commissioned by AWS.
> Cloud Native Computing Foundation (2022). Cloud Native Sustainability Survey.
> Google (2021). Data Centers Efficiency Best Practices.

Measurement Frameworks

> Green Software Foundation (2022). Software Carbon Intensity (SCI) Specification v1.0.
> ENTSO-E (2023). Electricity Maps: Carbon Intensity Data.
> Gregg, B. (2020). Systems Performance: Enterprise and the Cloud (2nd ed.). Addison-Wesley.
> Pereira, L. A. M., et al. (2017). Energy Efficiency Across Programming Languages: How Do Energy, Time, and Memory Relate? MSR 2017.
> The Shift Project (2021). Lean ICT: Towards Digital Sobriety Update.