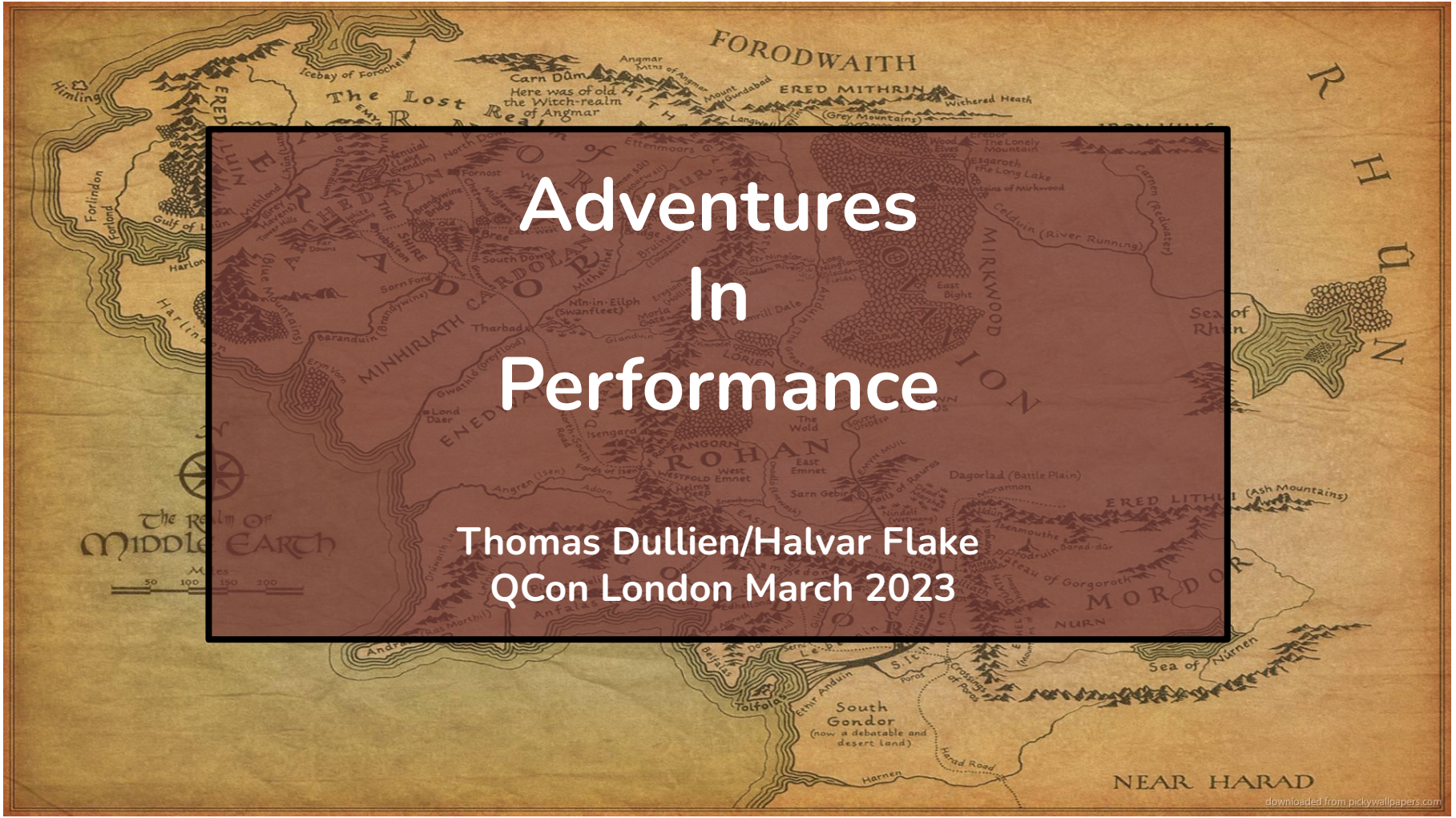


Adventures In Performance

Thomas Dullien/Halvar Flake

QCon London March 2023

Why care about performance? (Business reasons)

Moore's Law
Single-core speedups of 25%-50% per year meant that optimization made no economic sense

Death of Moore's Law

New hardware no longer pays for new features.

On-premise software
The cost of computing was paid by the user of the software, who also provided the hardware.

SaaS

Inefficient code cuts directly into gross margins.

On-premise hardware
Up-front CapEx for datacenters and hardware meant that efficiency gains could not translate to savings.

Cloud

Code that scales nonlinearly also scales **cost** nonlinearly.

Why care about performance? (Business reasons)

Moore's Law
Single-core speedups of 25%-50% per year meant that optimization made no economic sense

Death of Moore's Law

New hardware no longer pays for new features.

On-premise software
The cost of computing was paid by the user of the software, who also provided the hardware.

SaaS

Inefficient code cuts directly into gross margins.

On-premise hardware
Up-front CapEx for datacenters and hardware meant that efficiency gains could not translate to savings.

Cloud

Code that scales nonlinearly also scales **cost** nonlinearly.

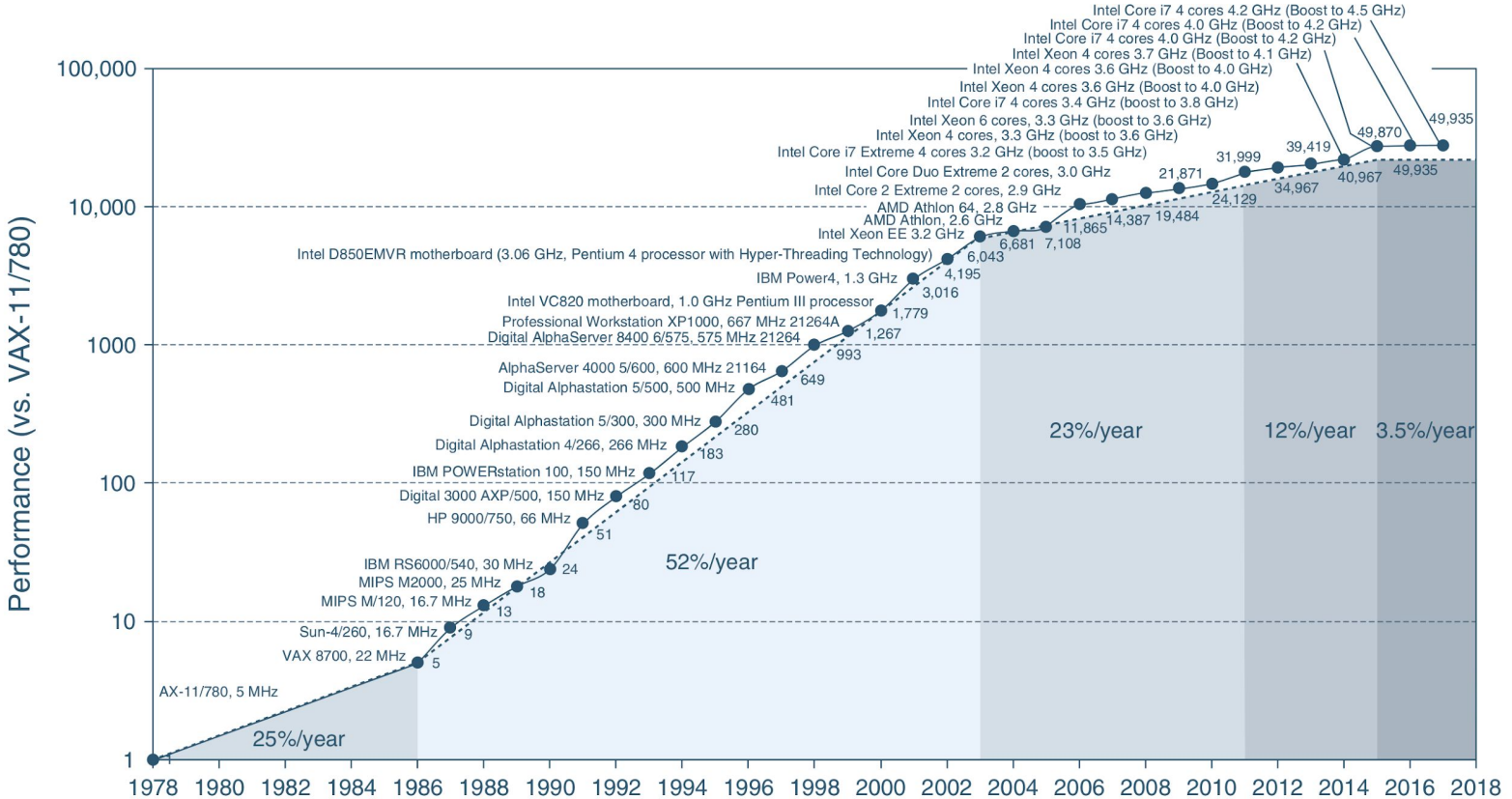
Performance per dollar is not growing much.

Cost of computing moved to software vendors.

Efficiency gains translate to $ saved.

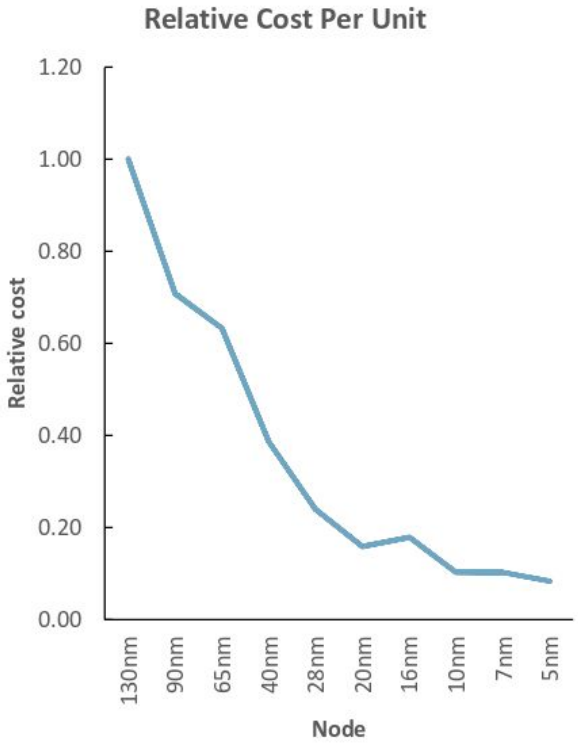
After 40 years of relative unimportance, **efficiency is important again.**

Death of Moore: No more single core speedup



Death of Moore: No more falling transistor cost

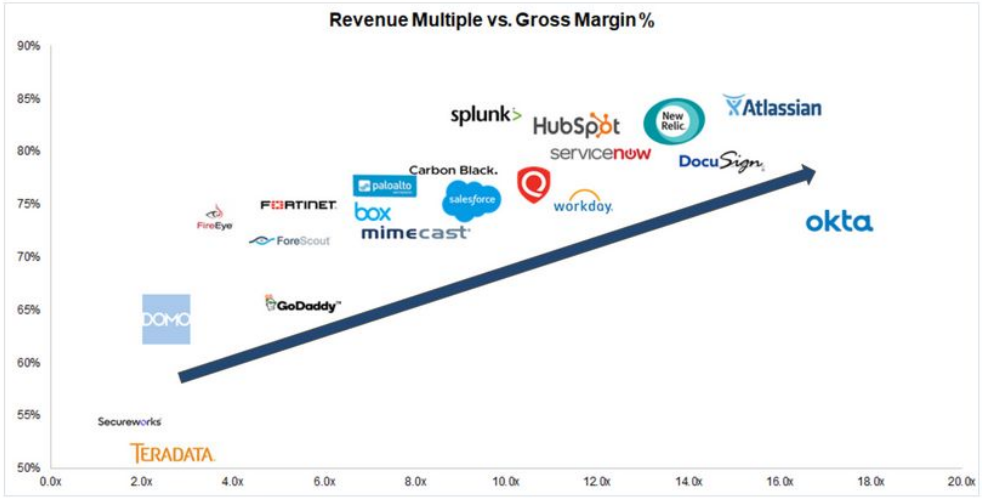
Per-transistor unit costs are not falling any more.



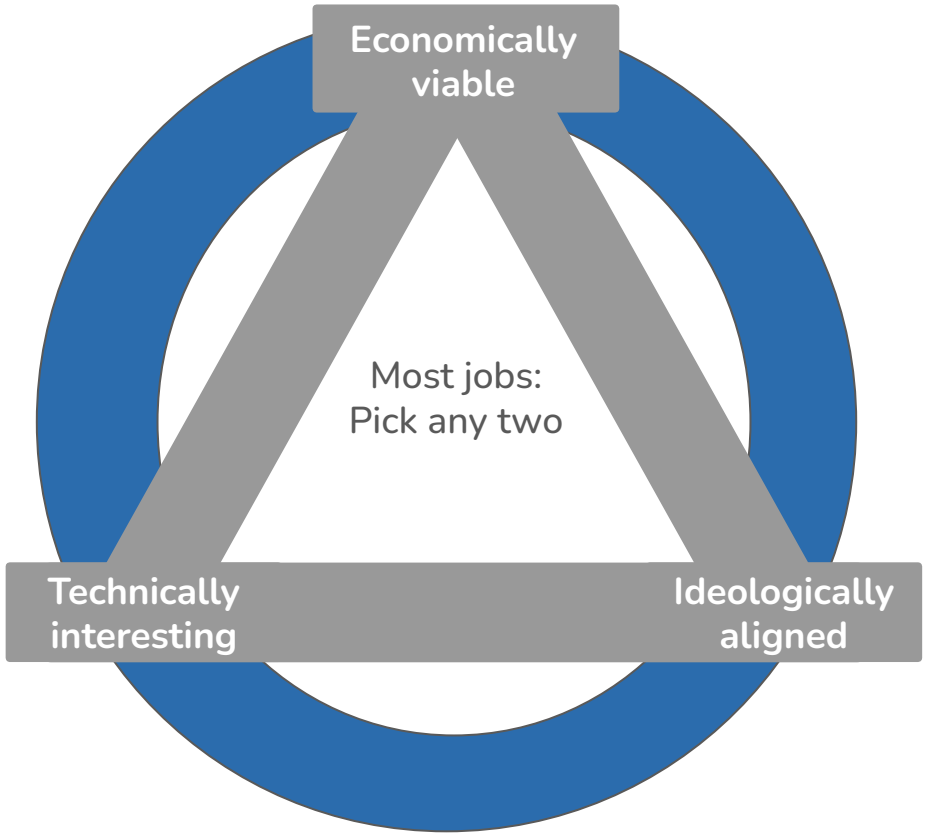
Importance of Gross Margins

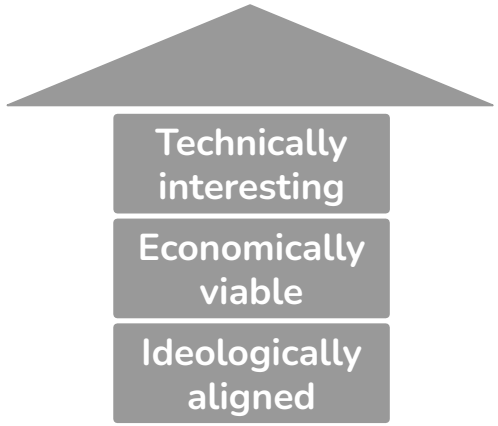
- COGS for SaaS companies principally consists of two components:
  - SRE's
  - Cloud costs

- Improving Gross Margin from 70% to 80% adds the same to the valuation as 2.5x ARR.



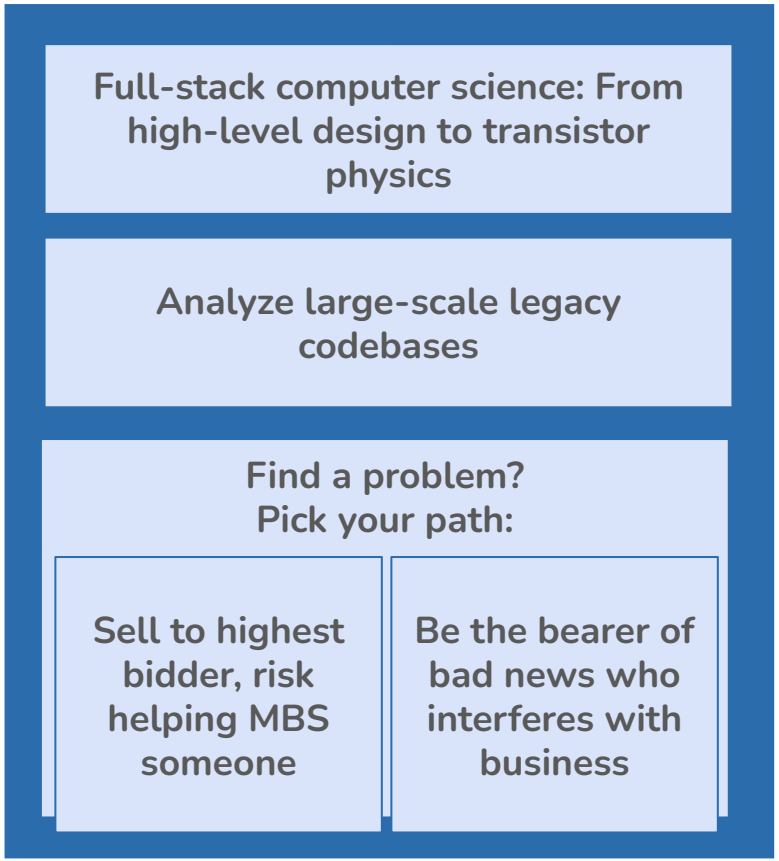
Why care about performance? (Personal reasons)

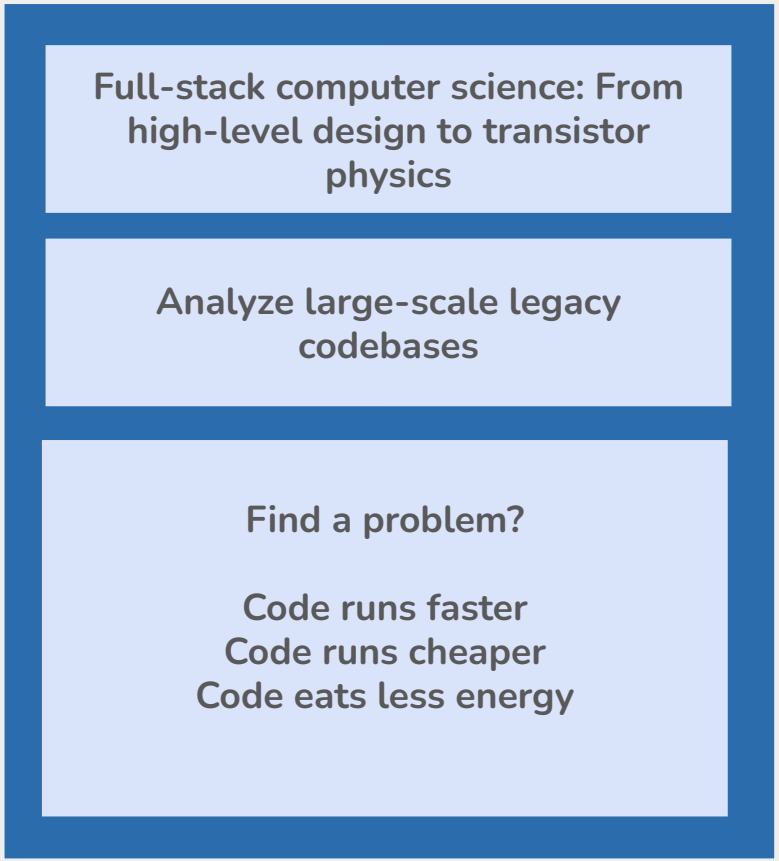


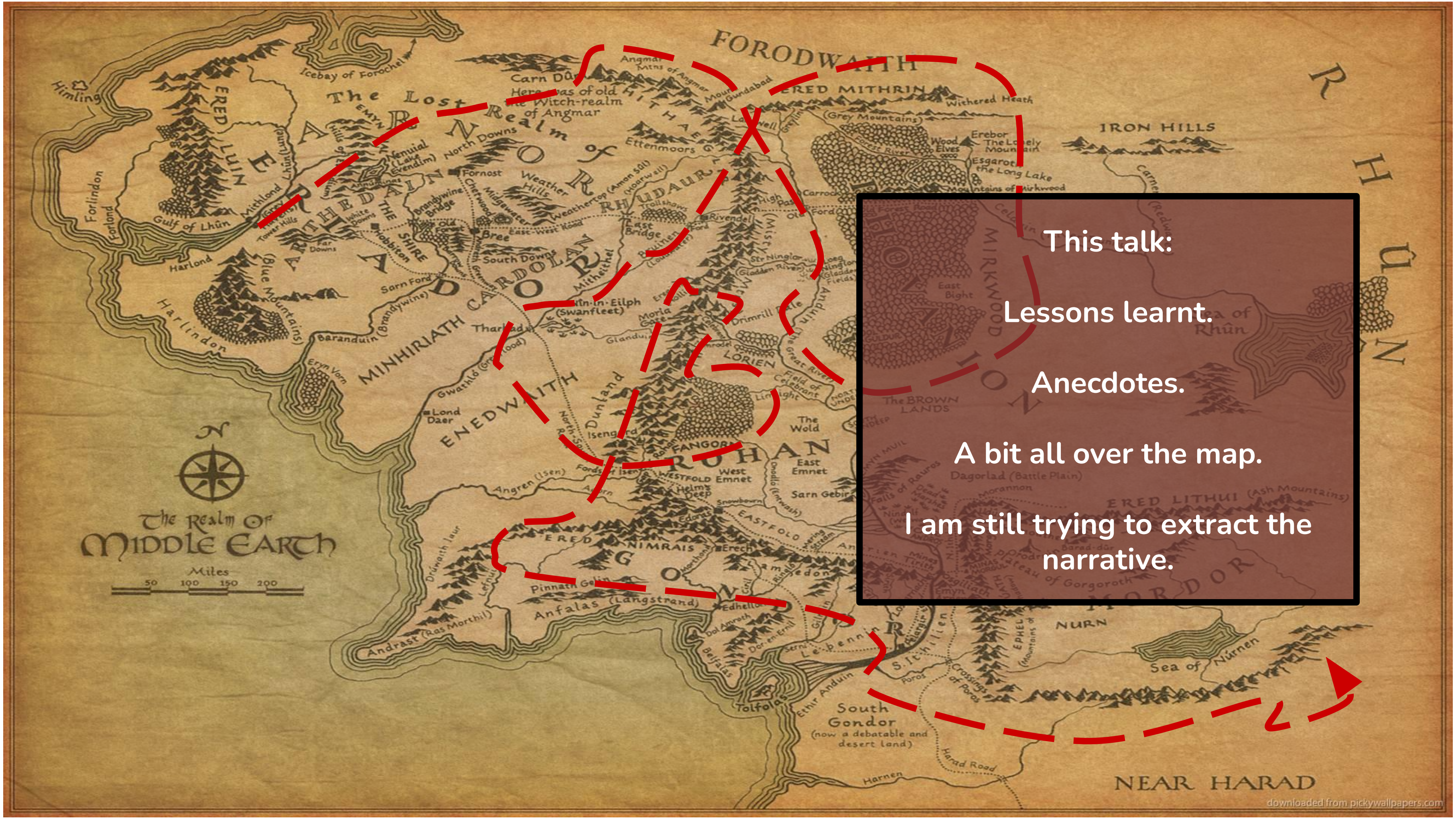


Perf work:
Good alignment

My path: Spy vs. Spy security => Performance engineering







This talk:

Lessons learnt.

Anecdotes.

A bit all over the map.

I am still trying to extract the narrative.

A few anecdotes...

Historical

- Your language is designed for computers that are extinct
- Your DB and application is designed for computers that are extinct

Organisational

- Your org chart matters
- Companies cannot buy something that has negative cost
- Compilation time & tragedy of the commons

Mathematical

- Benchmarking is a statistical nightmare

Technical

- Libraries dominate apps
- GC is a (high) tax

A few anecdotes...

Historical

- **Your language is designed for computers that are extinct**
- Your DB and application is designed for computers that are extinct

Organisational

- Your org chart matters
- Companies cannot buy something that has negative cost
- Compilation time & tragedy of the commons

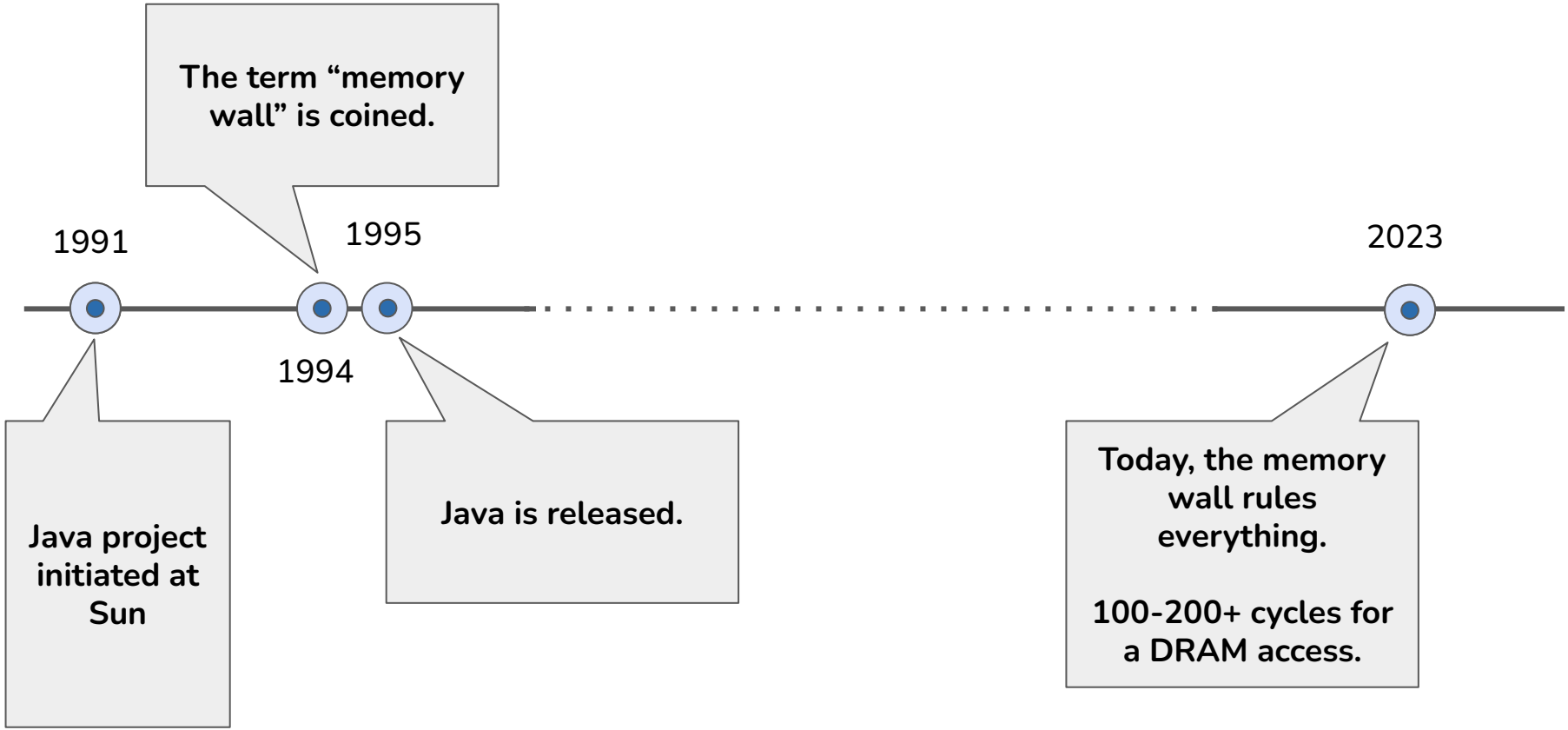
Mathematical

- Benchmarking is a statistical nightmare

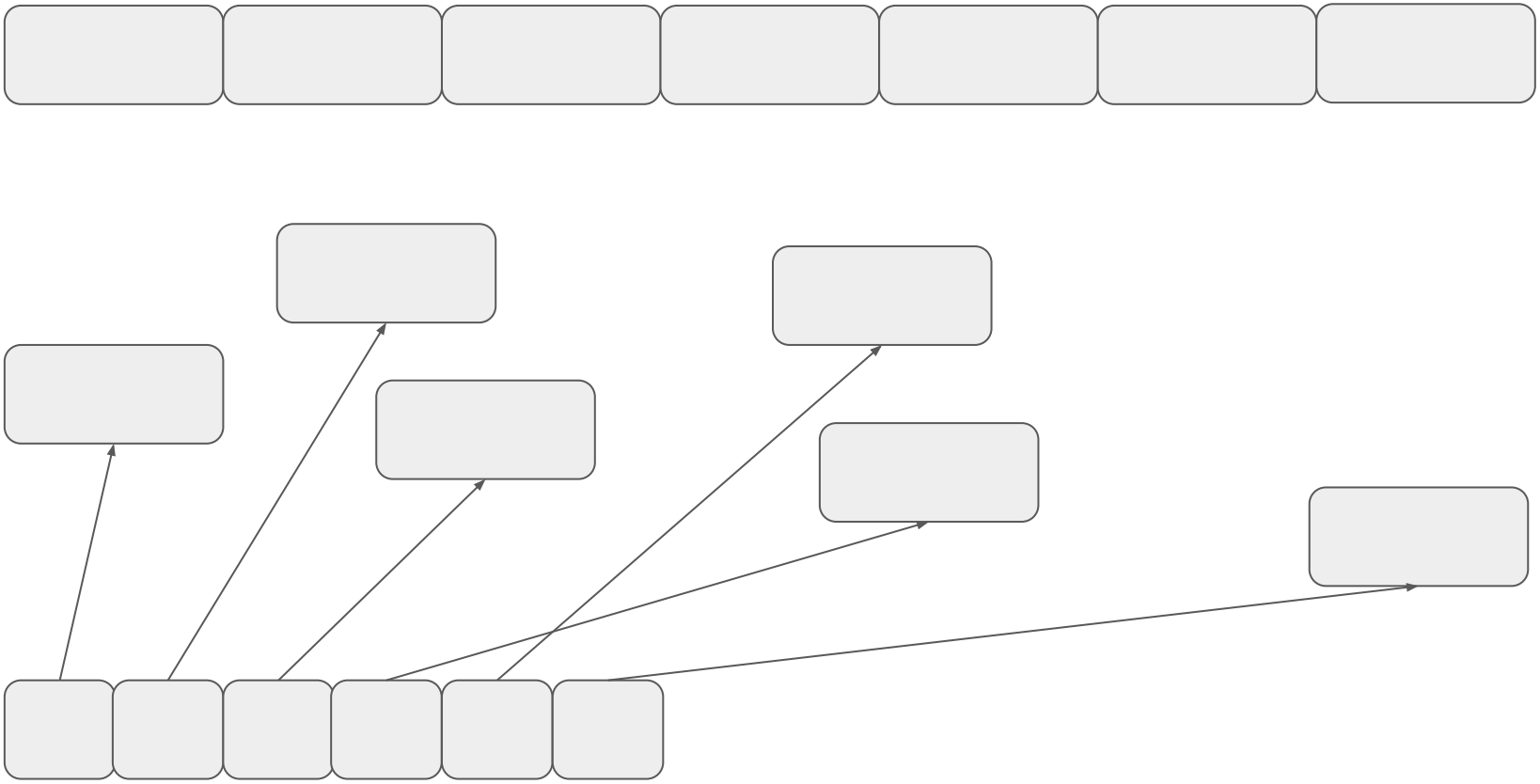
Technical

- Libraries dominate apps
- GC is a (high) tax

Timeline of the Java Programming Language



Array-of-struct in C vs. Object[] in Java



Assumptions baked into the language

- Traversing large linked graph structures on the heap is a reasonable thing to do (GC).
- Dereferencing a pointer does not come with a significant performance hit.
- Correct assumptions in 1991!

A few anecdotes...

Historical

- Your language is designed for computers that are extinct
- **Your DB and application are designed for computers that are extinct**

Organisational

- Your org chart matters
- Companies cannot buy something that has negative cost
- Compilation time & tragedy of the commons

Mathematical

- Benchmarking is a statistical nightmare

Technical

- Libraries dominate apps
- GC is a (high) tax

Properties of spinning disks

- 150-180 IOPS
- Seeks are very expensive
- Layout on the disk important: Seek times can be shorter when data is nearby
- Latency for seek/read in the multi-ms range
- Little internal parallelism - just a few seeks in the queue are useful
- Multiple threads contending the same HDD will ruin performance as disk heads need to seek between two places

Properties of NVMe SSDs

- 170000 IOPS (1000x more!) - 11x EBS gp2
- Seeks are very very cheap
- Internal row buffers
- Near-instant random writes (buffered by internal DRAM)
- Latency for seek/read under 0.3ms
- Significant internal parallelism - queue depth of 32-64 or higher necessary for optimal performance

mmap and large readaheads, fixed-size thread pools

A surprising number of storage systems follow this pattern:

- Fixed-size threadpools (~#cores)
- mmap-backed storage
- Reliance on large readaheads

Makes (some amount) of sense if you are on a spinning disk.

Pathology of that approach

- Servicing a page-fault is inherently blocking I/O
- Takes ~0.3ms to do end-to-end handling (thread resumes)
- A single thread can thus only originate 3000 IOPS, tops
- To saturate a 170k IOPS drive, you need 56 threads constantly hitting page faults (!)

- For blocking I/O, thread pools are usually too small, and the net result is a system that is slow, and nobody knows why (CPU and disks are both twiddling thumbs)

Feeding the beast

Modern SSDs are performance beasts, and you need to think carefully about the best way to feed them.

What about “cloud SSD”?

- Cloud-attached storage is an entirely different beast
  - HDD: High-latency, low concurrency
  - SSD: Low-latency, high concurrency
  - Cloud-storage: High-latency, extremely high concurrency
- Very few DBMS are optimized to operate in the “high-latency, near limitless concurrency” paradigm.
- Don’t expect the same codebase to be useful for all three paradigms.

A few lessons...

Historical

- Your language is designed for computers that are extinct
- Your DB and application is designed for computers that are extinct

Organisational

- Your org chart matters
- Companies cannot buy something that has negative cost
- Compilation time & tragedy of the commons

Mathematical

- Benchmarking is a statistical nightmare

Technical

- **Libraries dominate apps**
- GC is a (high) tax

Common libraries quickly dominate the largest app

There’s a finite number of pieces of code that globally eat the most CPU:

- Garbage collectors
- Allocators
- Compression (zlib etc.)
- FFMpeg etc.

In almost every large-sized org, the CPU cost of a common library will eclipse the cost of the most heavyweight app.

Unrealized vision: Performance bug bounties on GitHub

- Dream for optimyze was a global profiling SaaS
- Global view on which code eats CPU time

If you have a global view on which code eats CPU time, you can create bug bounties for FOSS libraries:

- “Improve this loop, earn $50k”
- Global savings for all users can be many million $

A few anecdotes...

Historical

- Your language is designed for computers that are extinct
- Your DB and application is designed for computers that are extinct

Organisational

- Your org chart matters
- Companies cannot buy something that has negative cost
- Compilation time & tragedy of the commons

Mathematical

- Benchmarking is a statistical nightmare

Technical

- Libraries dominate apps
- GC is a (high) tax

GC is a high tax

- Common libraries dominate apps
- GC is part of every app in a given runtime
- GC is expensive because traversing graphs on the heap is bad for locality

Common to see 10-20% of all CPU cycles in garbage collection (!)

“When we need to reduce CPU usage, we do memory profiling”

Anonymous high-ranking engineer at Ride-Sharing company

Wrestling the GC is commonplace

- Java folks become experts at tuning GCs
- High-performance Java folks become experts at avoiding allocations altogether
- Go folks end up analyzing the results of the escape analysis

Don’t worry about memory management, they said. The garbage collector will do it for you, they said.

Rust gave the most important contribution:
“You can have memory safety without having to have a GC.”

A few lessons...

Historical

- Your language is designed for computers that are extinct
- Your DB and application is designed for computers that are extinct

Organisational

- **Your org chart matters**
- Companies cannot buy something that has negative cost
- Compilation time & tragedy of the commons

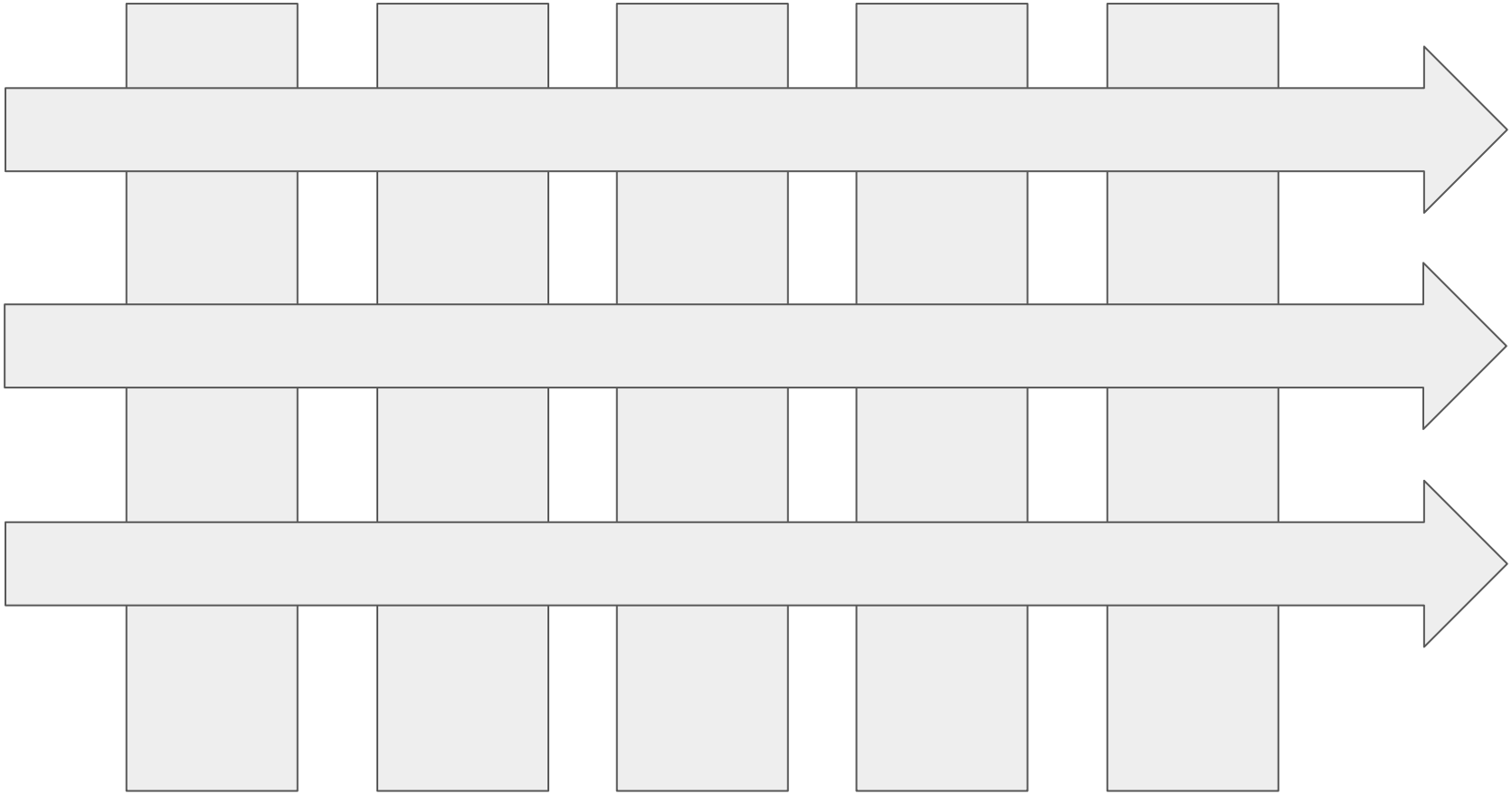
Mathematical

- Benchmarking is a statistical nightmare

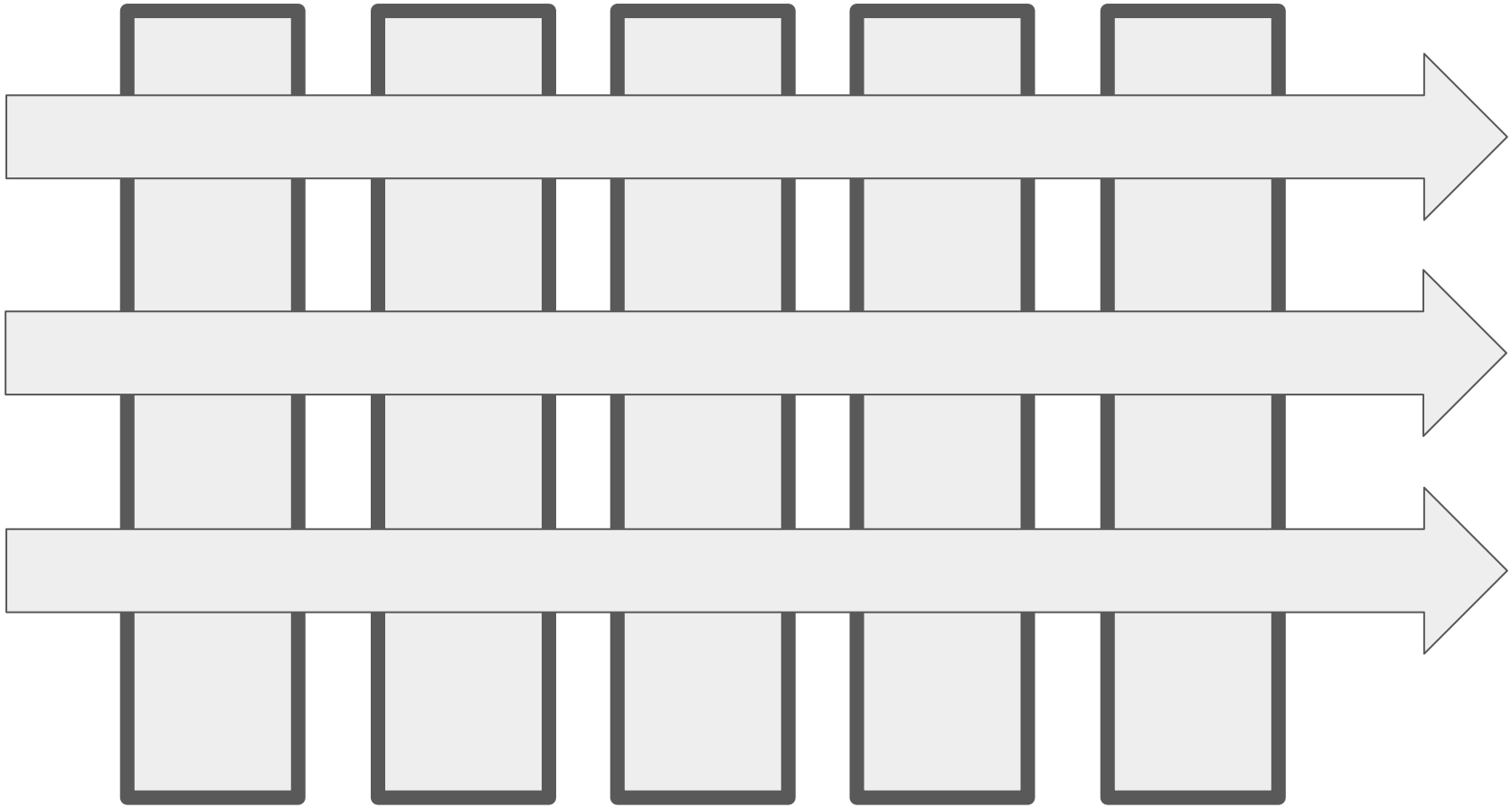
Technical

- Libraries dominate apps
- GC is a (high) tax

Horizontal vs vertical emphasis in organisation



Google: Strong vertical, very prescriptive, big monorepo, big services



Other places: More horizontal orientation, two pizza teams, separate repos & infra

Horizontal emphasis is for flexibility, vertical is for efficiency

Vertical organisations:

- Identify performance hog
- Fix the library
- Reap benefits everywhere

Horizontal organisations:

- Identify performance hog
- Work with 100+ teams to get the change integrated into 100 repos/places

A few anecdotes...

Historical

- Your language is designed for computers that are extinct
- Your DB and application is designed for computers that are extinct

Organisational

- Your org chart matters
- **Companies cannot buy something that has negative cost**
- Compilation time & tragedy of the commons

Mathematical

- Benchmarking is a statistical nightmare

Technical

- Libraries dominate apps
- GC is a (high) tax
- Temporal dependencies are bad
- “Stupid is beautiful”

Cut-of-savings contracts fail on accounting & legal hurdles

Performance work lends itself to value-based pricing (cut-of-savings). Unfortunately, lots of technical and legal hurdles:

- Nobody knows how to budget for a contract that has guaranteed net-negative-cost, accounting goes crazy.
- Legal departments are skittish because they don't know what the legal exposure is (in terms of $).

"You may be able to do this if you're Bain and have golfed with the CEO for 2 decades, but not as a startup."

A few anecdotes...

Historical

- Your language is designed for computers that are extinct
- Your DB and application is designed for computers that are extinct

Organisational

- Your org chart matters
- Companies cannot buy something that has negative cost
- Compilation time & tragedy of the commons

Mathematical

- Benchmarking is a statistical nightmare

Technical

- Libraries dominate apps
- GC is a (high) tax

Nobody likes long compilation times

- Go compiler devs are zealous about this
- Upstream package builders have limited computational resources

Tragedy of the commons:

- If a library runs on 1000 cores for 1 year, 1% performance improvement are worth 10 core-years.
- It would make global sense to compile certain heavy libraries (zlib etc.) for **weeks** if 1% extra perf could be obtained.

The person compiling the code has no incentive to speed up everybody's binary. Classical tragedy of the commons.

On x86_64, everybody compiles for the wrong uArch

- Upstream packages are compiled for the lowest-common denominator
- Your cloud instance almost certainly has a newer uArch

You can often get measurable speedups by rebuilding for your uArch. Complication: Cloud instances don’t map 1:1 to uArch.

(I wish mainstream Linux distros would have uArch-specific packages.)

A few lessons...

Historical

- Your language is designed for computers that are extinct
- Your DB and application is designed for computers that are extinct

Organisational

- Your org chart matters
- Companies cannot buy something that has negative cost
- Compilation time & tragedy of the commons

Mathematical

- Benchmarking is a statistical nightmare

Technical

- Libraries dominate apps
- GC is a (high) tax

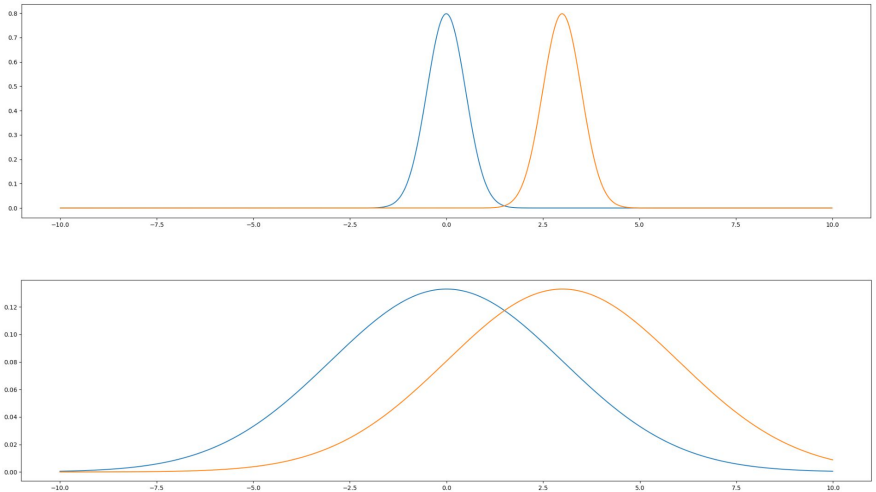
Benchmarking is a statistical nightmare

In theory, every organisation would want benchmarks to run as part of CI/CD, to help improve performance and prevent performance regressions.

In theory, determining “does this change make my code faster” should be easy to do, right? I mean, classical statistical hypothesis testing, right?

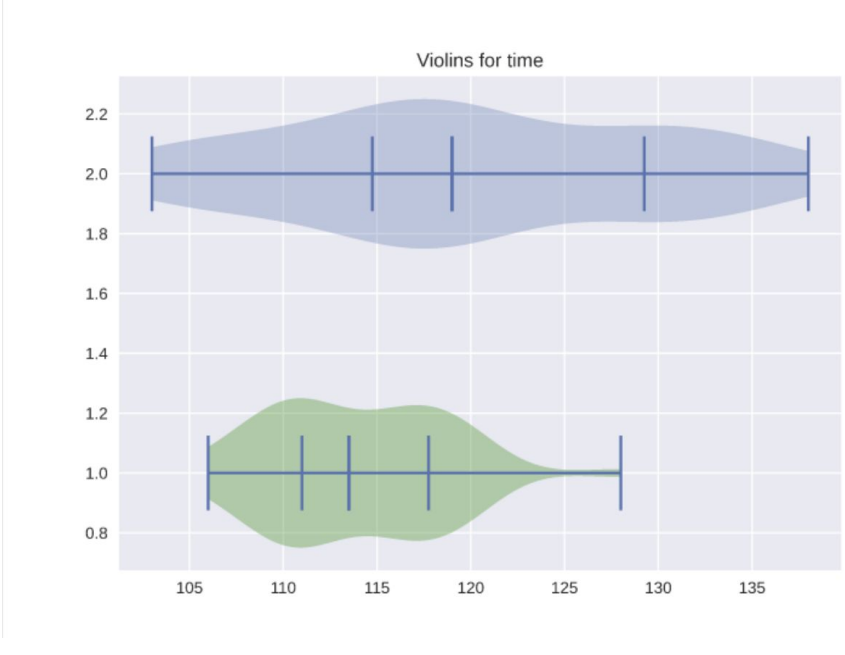
In practice, it is super rare to see an organisation that runs statistically sound benchmarks in CI/CD that aren’t plagued by a dozen validity problems. (good examples: Clickhouse and Mongo)

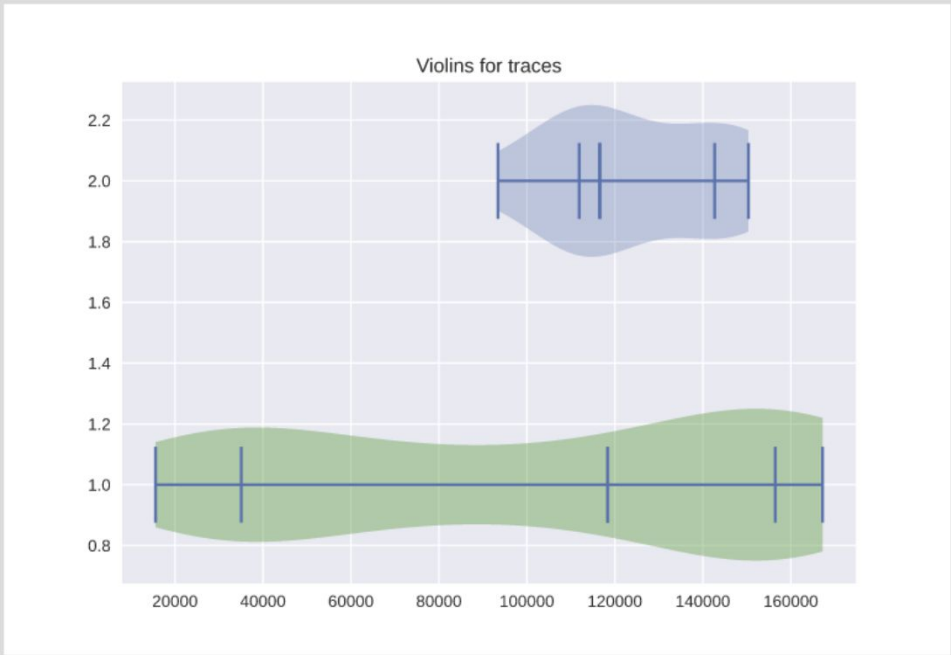
Problem 1: Variance is your enemy, particularly for small effect sizes



High variance in measurements means it is harder to tell if your change improves things, but people do not fear variance enough.

Problem 2: Does this look normal to you?



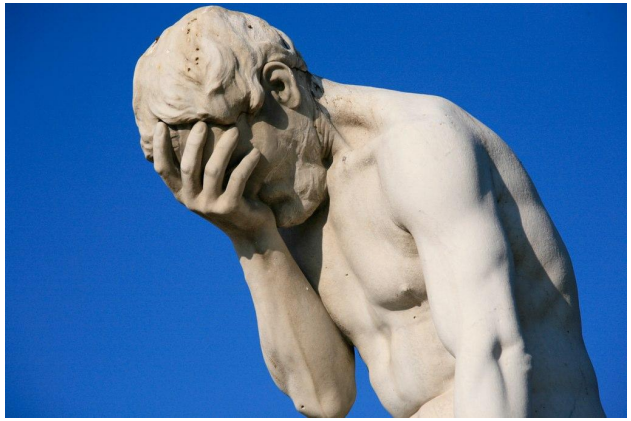


Memory hierarchies, caches etc. all create complicated multi-modal and pathological distributions. Goodbye parametric testing :-/ and therefore goodbye sample efficiency.

Problem 3: CPU internals mean your benchmark runs aren't IID

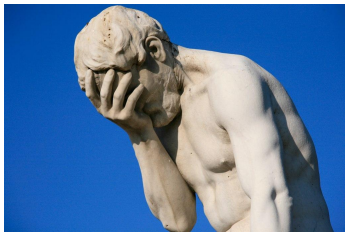
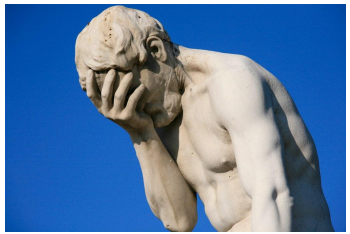
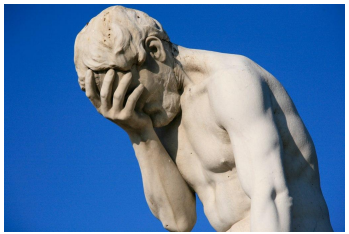
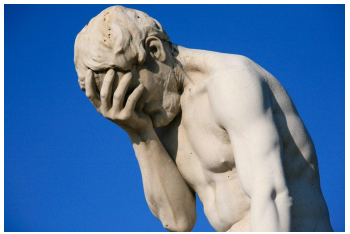
Running a benchmark multiple times "trains" the various predictors. Solution: Random interleaving of benchmark runs. [1]

Frequency boosts and -scaling happens.



Problem 4: ASLR, caches, noisy neighbors on cloud instances

- ASLR can interact poorly with caches, causing +/- 10% noise for particularly unlucky memory layouts
- Cloud instances can have noisy neighbors that run cache benchmarks or accidentally exhaust memory bandwidth

Problem 5: Controlling for all of these means you're not measuring what matters

- Running single-tenant on bare metal and disabling frequency scaling controls variance.
- Unfortunately, your measurements are no longer representative of what happens in production.

Results

- Approximately nobody has statistically reliable benchmarks that show improvement or regression on CI/CD
- Cost of running enough experiments on commit to establish “this makes something faster” with small-enough confidence interval is often prohibitive
- Overall seems to not bother anyone :-)

Resources

- MongoDB have written candidly about their work here: [2]
- Clickhouse have written candidly about their work here: [3]
- Andrey Akinshin has an excellent (if heavy) blog on nonparametric statistics for performance here: [4]

Concrete advice?

What are the takeaways from all of these lessons?

Advice to the practitioner

Technical

1. **Know your napkin math.**
   Almost all performance analysis begins by identifying a discrepancy between what napkin math says and what happens in real systems. “This should not be slow” is the start of most adventures.

2. **Accept that tooling is nascent and disjoint.**
   I ended up starting a company because I needed a simple fleet-wide CPU profiler. The tools you need are nascent and ill-fitting.

3. **Always measure.**
   Performance problems are almost always a murder mystery, and quite often the perpetrator is not the usual suspect.

4. **So. Many. Low. Hanging. Fruit.**
   Most environments leave 20-30%+ of easy wins on the table. That’s real money.

Advice to the practitioner

Organisational

1. **Try to establish a north-star metric**
   For digital goods providers, you really want to work toward a "cost-per-unit-served". Fixes incentives.

2. **Dedicated teams and "optimization weeks" have good results**
   Doing a "hack week" focused on identifying and fixing low-hanging fruit yields good results.
   Dedicated optimization teams can really pay for themselves in larger orgs.

3. **Few other areas on software engineering have such clear success metrics.**
   Many engineers enjoy the clarity and game-like nature.

4. **The organisational importance will only grow over time.**
   With the end of free money, margins matter, and I would not be surprised if efficiency becomes a board-level topic on par with security etc.

Advice to the practitioner

Mathematical

1. **Methods of last resort are your first resort :-(**
   Pathologies of the underlying distributions force use of non-parametric methods.

2. **Low statistical power of these tests requires many runs to obtain reasonable confidence bounds**
   Cost of doing scientifically sound benchmarking might be too high to do on each commit.

Advice to the practitioner

Historical

1. **Be aware of drastic changes in computer "geometry"**
   A 1000x increase in IOPS, a change in cost for a previously-expensive operation, a change in SotA for data compression has drastic downstream effects.

2. **Code and configs outlive hardware, often by decades**
   Code either gets replaced quickly or lives almost forever. 90% of the code you write today will be gone in 5 years, and 10% will still be there in 20 years. I guess that approximately all tuning parameters older than 3 years are wrong.

Technical outlook?

What tools are missing? If I could dream, what tools would I want?

All that, and a pony

Diagnosing performance issues currently requires integrating different tools:

- CPU and memory profilers
- Distributed tracing
- Data from the OS scheduler about threads
- (...)

Visualisation and UI for these things is disjoint, not performant, and generally janky.

The tools I wish I had

There are different tools I wish I had, for different purposes:

- **CO2 reduction:**
  - Global profiling SaaS database with a statistically significant sample of all workloads in the world (FOSS performance bounties)
- **Cost accounting:**
  - Profiling of CPU, IO, network traffic to assign resources to code
  - Integration with a metric about “units served” to calculate cost breakdown
- **Latency analysis:**
  - Combination of CPU profiling, distributed tracing, and scheduler events, all tied together over the network, with zero deployment friction and negligible impact
  - GPU-accelerated UI to visualize that data in time (Perfetto++)
- **Cluster-wide truly causal profiling:**
  - Some approaches to causal profiling (“which line of code needs to be made faster”) exist (Coz etc.)
  - “Not causal enough” (can’t tell me for example that I am hitting too many page faults)
  - Not cluster-wide
  - “Why is this request slow?” / “Why is this request expensive?”

Big hurdle to adoption is deployment friction. Frictionless needs to be the goal.


Ok … and now?
What’s next?
FORODWAITH
RHUN
MIRKWOOD
RHOVANION
ENEDWITH
MINHIRIATH
ERIADOR
ARNOR
GONDOR
MORDOR
ERED LITHUI
NURN
Sea of Rhun
Sea of Nurnen
NEAR HARAD
South Gondor
(now a debatable and desert land)
The Realm of MIDDLE EARTH
downloaded from pickywallpapers.com


Questions?
FORODWAITH
ERED MITHRIN
IRON HILLS
RHÛN
ARNOR
The Lost Realm of
ARTHEDAIN
RHUDAUR
CARDOLAN
MINHIRIATH
ENEDWAITH
GONDOR
ANGMAR
The Realm of
MIDDLE EARTH
MORDOR
NURN
NEAR HARAD
Dagorlad (Battle Plain)
Sea of Rhûn
Sea of Núrnen
South Gondor
(now uninhabitable and desert land)
downloaded from pickywallpapers.com