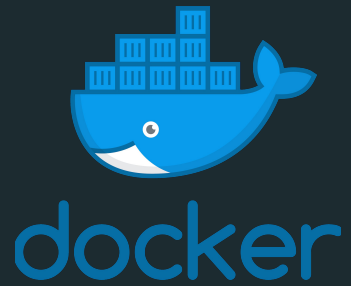
# The Modern Platform in 2020

Justin Cormack



# Who am I?

Engineer at Docker in Cambridge, UK.

Docker ❤️ developers

Work on security, systems software, applications, LinuxKit, containers

@justincormack

# From OS to languages?

# From OS to languages

- I have moved from running QCon Operating System tracks to language tracks over the last few years
- For me not a big change, unikernels are a language approach to the OS, and the language that we use to write the OS are important
- Many of the drivers for change in the two areas are similar...

# Has anything really changed recently?

# Drivers of change

- Performance requirements - going to talk a lot about these
- Hardware is changing because of these. Vectors, GPU, FPGA, we are moving away from the classic PDP-11 C model
- On the other end, we are programming billions of tiny devices
- Security is important. Our software is being attacked faster than ever, and languages can help us.

# The environment

- the two huge C/C++/... compiler projects, gcc and LLVM have become amazingly successful, as has OpenJDK for the JVM.
- gcc witnessed the birth of commercial open source, with Cygnus ("Cygnus your GNU support") being the first real commercial open source company founded in 1989, later acquired by Red Hat.
- Cygnus was so successful that most commercial compiler vendors shut down.
- LLVM, supported by Apple and later others has provided competition.
- These projects have provided ability for lots of languages to thrive.
- Many other languages have their own compilers and interpreters...

# Performance

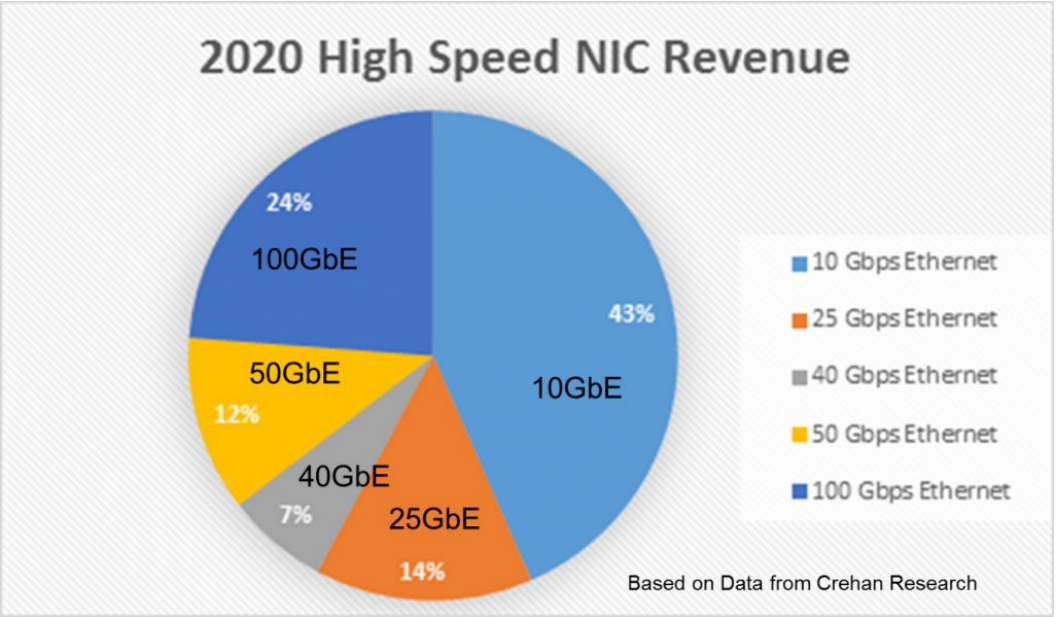
# Performance

*“A supercomputer is a device for turning compute-bound problems into I/O bound problems.”*

Ken Batcher

# Storage and network got *much* faster

- cheap 25 gigabit ethernet
- 100 gigabit ethernet
- millions of packets/sec
- SSD, NVMe, NVDIMM
- millions of IO/sec
- IO bandwidth way up
- clock speeds only doubled
- lots of CPU cores



# This is changing everything

- 1Gb ethernet to 100Gb, two orders of magnitude faster
- SSD seek time two orders of magnitude faster than disk

- Back in the early 2000s in memory databases were the big thing
- C10K, 10 thousand connections on a server, was hard
- epoll was invented to fix this, and events not threads

- SSD can now commit at network wire speed
- C10M is possible now
- every CPU cycle counts, 10GbE is up to 14m packets/s
- only 130 clock cycles per packet!



# Storage is changing as fast

- Solid state storage has replaced spinning rust everywhere
- Laptops
- Databases
- Most non-archival storage soon
- Latency driven

Next stage is NV-Dimm

- Flash in memory form factor
- 10x capacity of RAM, lower power consumption
- Latency little higher, write directly from CPU not via RAM.
- Cache-line addressable



# Power and heterogeneity

# Power consumption of computers

- data centres consume roughly 3% of electricity, a share that is growing
- large cloud providers are more efficient than traditional centres
- we need to make our use of computers a lot more efficient

- also, don't do unnecessary computation
- also, use computers powered by renewable energy



# Understanding Sources of Inefficiency in General-Purpose Chips

Rehan Hameed[1], Wajahat Qadeer[1], Megan Wachs[1], Omid Azizi[1], Alex Solomatnikov[2], Benjamin C. Lee[1], Stephen Richardson[1], Christos Kozyrakis[1] and Mark Horowitz[1]

[1]Dept. of Electrical Engineering
Stanford University, Stanford, CA
{rhameed, wqadeer, wachs, oazizi,
bcclee, steveri, kozyraki, horowitz}@stanford.edu

[2]Hicamp Systems,
Menlo Park, CA
solomatnikov@gmail.com

## ABSTRACT

Due to their high volume, general-purpose processors, and now chip multiprocessors (CMPs), are much more cost effective than ASICs, but lag significantly in terms of performance and energy efficiency. This paper explores the sources of these performance and energy overheads in general-purpose processing systems by quantifying the overheads of a 720p HD H.264 encoder running on a general-purpose CMP system. It then explores methods to eliminate these overheads by transforming the CPU into a specialized system for H.264 encoding. We evaluate the gains from customizations useful to broad classes of algorithms, such as SIMD units, as well as those specific to particular computation, such as customized storage and functional units.

The ASIC is 500x more energy efficient than our original four-processor CMP. Broadly applicable optimizations improve

## 1. INTRODUCTION

Most computing systems today are power limited, whether it is the 1W limit of a cell phone, or the 100W limit of a server. Since technology scaling no longer provides the energy savings, it once did [1], designers must turn to other techniques for continued performance improvements and tractable energy costs. One attractive option is to understand and to incorporate sources of ASIC efficiency, since general-purpose processors can be outclassed by three orders of magnitude in both performance and energy efficiency by ASIC designs [5].

The desire to achieve ASIC-like compute efficiencies with microprocessor-like application development cost is pushing designers to explore two new areas. One area aims to create CPU designs with much lower energy per instruction [6], while the other aims to create new design methodologies to reduce the cost

# What can we do now?

- use hardware that we have now
  - vectorization (AVX-512 was a GPU design!)
  - other CPU accelerators such as crypto
  - GPU
  - FPGA
- our programming languages don't expose these, our compilation targets don't include these in general
- see 1.40pm *Java for GPUs and FPGAs*, Juan Jose Fumero Alfonso
- factors of 10 or even more in efficiency for suitable code!

# We need languages that map better

- JIT is a great solution to extract parallel, vector and GPU friendly code
- right now much vector code is hand written with intrinsics! This is close to going back to writing assembly.
  - eg see simdjson https://github.com/lemire/simdjson parse JSON at gigabytes a second.
- GPU code has special toolchains such as CUDA.
- Julia is doing great work in exploring this space for numerical work.

# Languages that output languages

# Languages are generating more languages

- We have frameworks that generate configuration Yaml from code, such as Pulumi and jk
- We have languages that generate hardware description languages, such as Chisel, which is Scala used to generate hardware description graph, to design chips.
- We have frameworks such as Tensorflow that generate computation graphs for ML applications, from Swift or Python.
- Language technology is ending up everywhere in our stacks...

# Not just the high performance, the small

# Reprogramming the small

- efficiency in servers is important, but there are vast numbers of smaller devices
- scavenged power and ultra low power especially important, so accelerators just as important in longer run
- over 20bn Arm chips, mostly microcontrollers ship every year
- over time microcontrollers are moving from 8 bit to 32 bit RISC
- too cheap and low power to run Linux, historically programmed in C or assembly.
- we need higher level languages to enable broader base of programmers and lots of new applications.
- see 4.10pm *Tiny Go, Small is Going Big* by Ron Evans

# Accelerators on microcontrollers

- as microcontrollers becoming exposed over networks they need encryption and authentication
- software encryption is very slow
- even areas such as toys are seeing this, a microcontroller in a toy may need to communicate with an App on a phone
- so accelerators for encryption are growing in use
- also AI apps. Running models on microcontrollers is feasible if heavily optimised, even down to 1 bit operations
- microcontrollers with AI acceleration to do basic processing becoming more common.

# Web Assembly

# What is so different about the Wasm platform?

- surely we have had platforms like the JVM before?
- first, it has shipped in the browser, the first new platform there since the JVM and Flash were removed. No longer a JavaScript only world.
- because of the draw of the web platform, almost every language has a Wasm strategy now, so it is becoming a universal target.
- we have learned from previous designs and have an open process to add features to benefit all targets.
- interesting new platforms like CloudFlare's web workers based on Wasm.
- 2.55pm *Build your own WebAssembly Compiler* Colin Eberhardt

# Multi and mixed language support

- one dream for Wasm is that we will get mixed language interoperability.
- this is a really hard problem! languages have different concepts!
- Python does not understand Rust linearity restrictions!
- ownership and garbage collection are hard problems
- the JVM worked around this by forcing all languages into one model to a large extent.
- making this work will be hard but a lot of people are going to try!



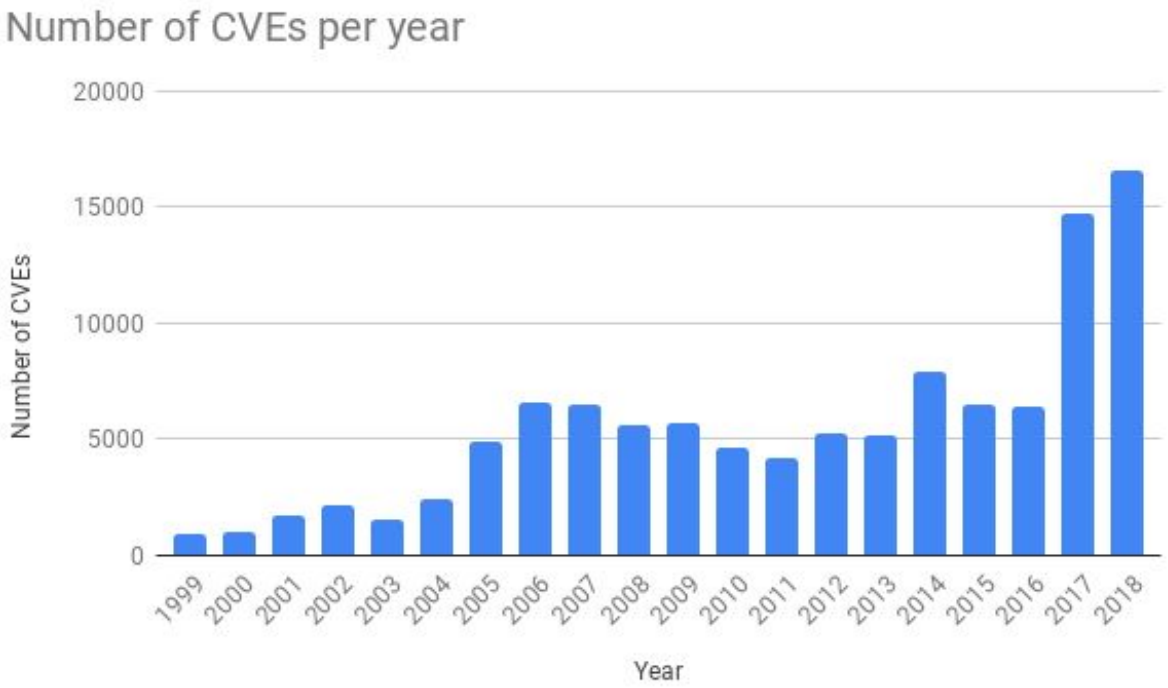
# Safety

# The rise of automation in security testing

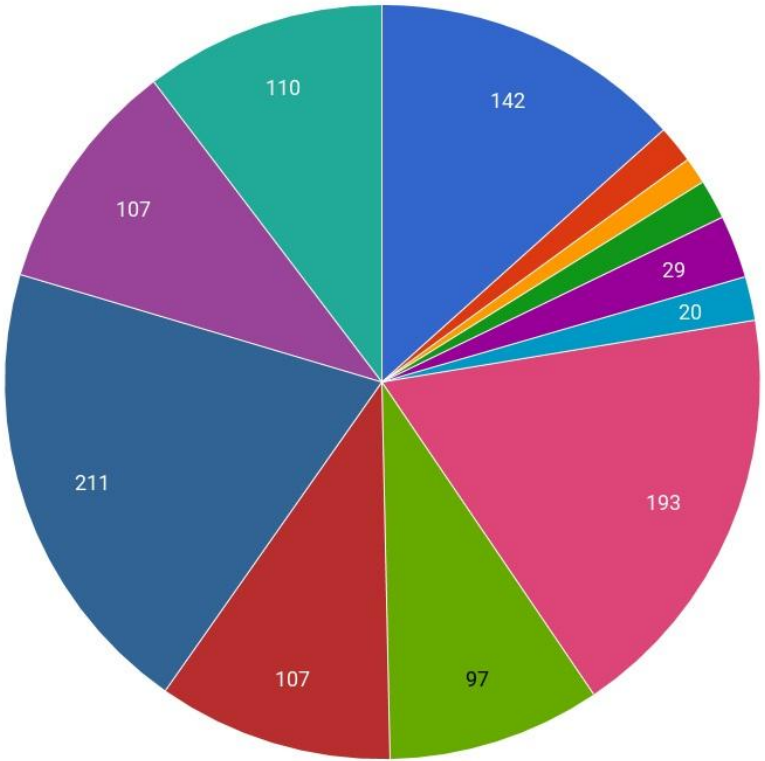
- the first highly successful fuzz tester, American Fuzzy Lop was released in 2013
- fuzz testing is now available as a service and works really well, see Fuzzbuzz, OSS-Fuzz from Google, Microsoft Security Risk Detection
- also tools such as Semmle, bought by GitHub last year, which given one security issue finds more related ones



# Better at finding issues



# What kind of issues?




- heap buffer overflows
- global buffer overflows
- stack buffer overflows
- use after frees
- uninitialized memory
- stack overflows
- timeouts
- ooms
- leaks
- ubsan
- unknown crashes
- other (e.g. assertions)

docker

# Right now, memory safety issues dominate

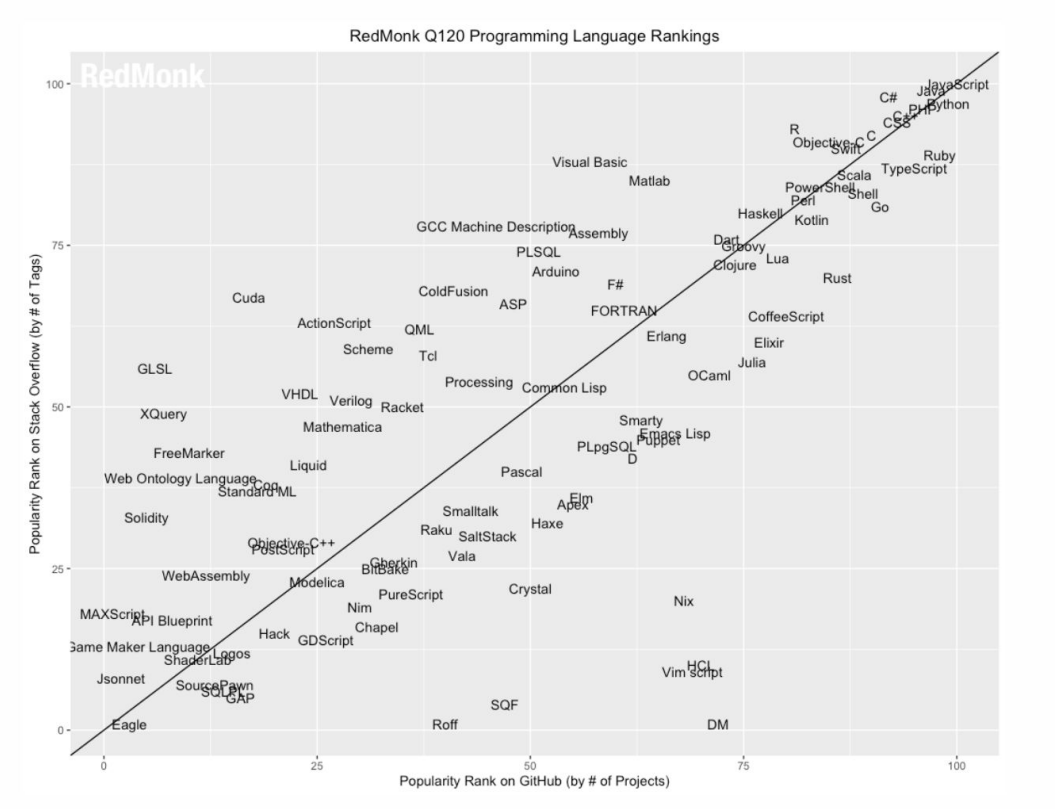
- estimates are that 70% of security issues are memory safety
- pretty much every language created in the last 20-30 years is memory safe. We have huge amounts of critical infrastructure in C!
- CVEs will continue until we can migrate to newer languages
- but languages can get us more than memory safety

# Beyond memory safety

- the next frontier is safe concurrency
- we are increasingly writing highly concurrent programs
- that is partly to increase resource efficiency, and because the world we interface with works like that
- deadlocks and race conditions and locking bugs can be fixed by languages and types too!
- These are correctness bugs and security issues.
- 5.25pm Pony, Types and Garbage Collection, Sophia Drossopoulou
- Pony is a little known but really lovely language used in areas such as high performance streaming applications, which offers totally safe concurrency.

# Languages everywhere

# Languages everywhere



# Why so many?

- there are more than that!
- languages are our tools as programmers, and the more kinds of thing we try to do the more languages we build
- as we try to do more complex things we learn more about how we can use languages to do these.
- we want to be able to manipulate our languages with code, and so much more, and the languages we have are not great for that.



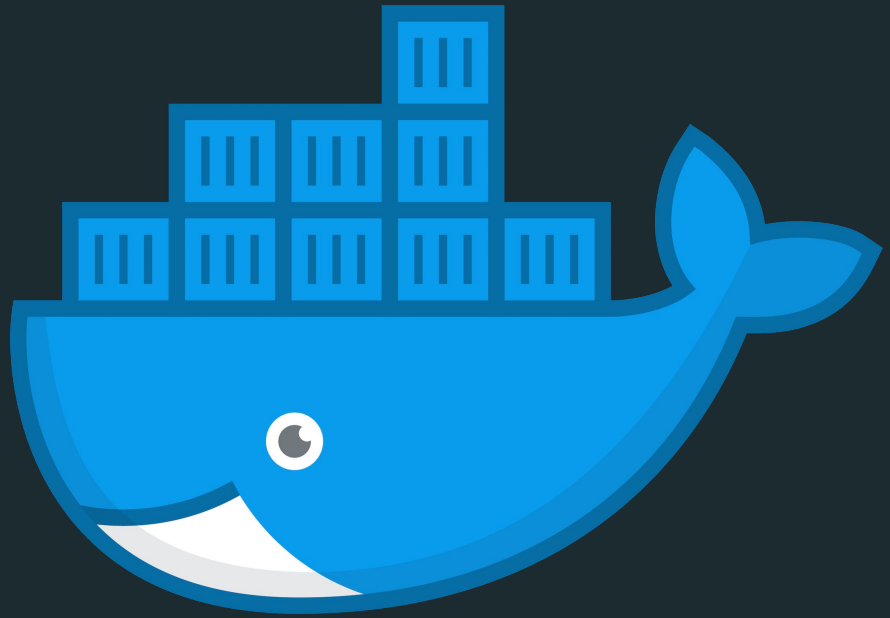
# What next in languages?

- this post "What next?" by Graydon Hoare has some ideas http://graydon2.dreamwidth.org/253769.html
- I think the bugs and security issues that it causes and the coding difficulties mean that we need to get rid of "undefined behaviour" next.
- this is part of the journey to being able to reason about what code does, which is necessary for many uses doing transformation of code correctly, and formal methods, another frontier.
- we need to make language technology more accessible (like the Wasm talk later does!), make small languages (Tiny Go!), make safe languages (Pony!) and make it easier to work with real modern hardware (TornadoVM!)

# Language shapes the way we think

- understanding the different approaches languages take is one of the most important things about learning to deeply understand programming.
- Many projects and products are building languages and language technology deeply into them to add powerful new features.
- Hardware and software and power consumption and performance are going to be driven by new things we do with programming languages.





# THANK YOU