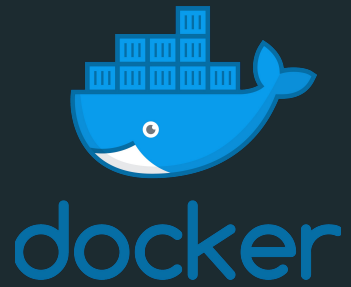
The Operating System in 2018

Justin Cormack



Who am I?

Engineer at Docker in Cambridge, UK. Formerly Unikernel Systems.

Work on security, systems software, LinuxKit, containers

@justincormack

Has anything really changed recently?

The big areas of change and stasis

- Biggest changes, but all still ongoing
  - performance driven changes eg eBPF, userspace networking, NVMe
  - operations, how we use and deploy operating systems
  - emulation and portability
  - slow death of legacy
- Little change so far but signs that it is coming
  - the monolithic OS
  - lack of diversity
  - programming languages
  - security

docker

Performance

Two talks earlier today in this track

- Thomas Graf on eBPF
- Alan Kasindorf on NVMe
- If you missed these, highly recommend the recordings!

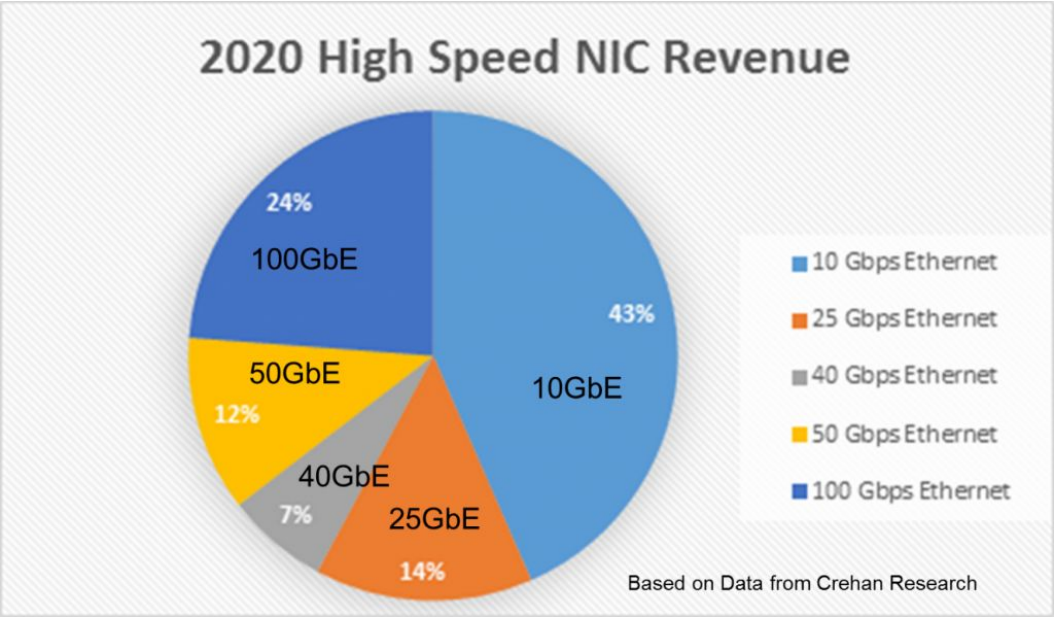
Performance

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

Ken Batcher

Storage and network got *much* faster

- cheap 10 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 is replacing 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

How to fix it 1

Userspace

Avoid the kernel userspace switch latency

- system calls are relatively slow
- run all the code in userspace to avoid switches
- minimal use of the kernel!
- involves writing device drivers in userspace
- DPDK (networking) is the most widely used framework
- also SPDK (NVMe), Snabb (networking)
- userspace drivers getting easier, firmware provides higher level API
- eg Mellanox has a single driver API for 10-100Gb ethernet
- NVMe is widespread standard API for storage

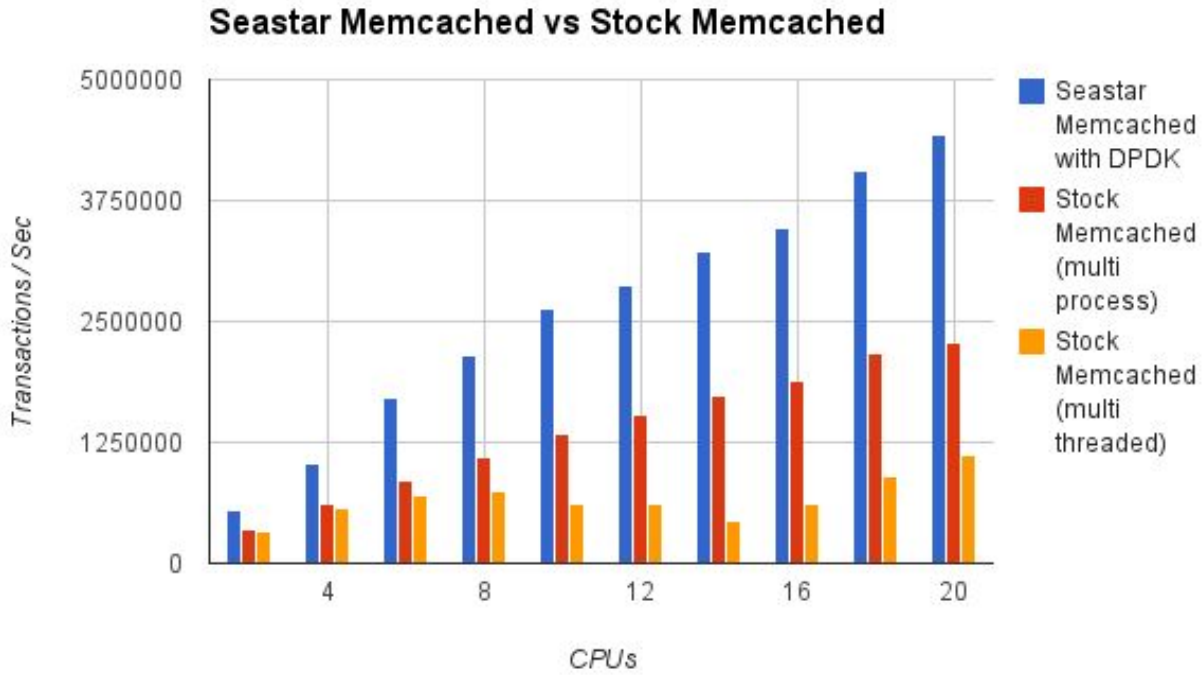
docker

Example: SeaStar

- SeaStar: high performance database application
- Originally the company shipped as a unikernel
- Now a framework hosted in Linux but not using much of Linux
- C++
- DPDK
- userspace TCP stack
- no locking, just message passing with ring buffers
- Cassandra, Memcached and Redis compatible backends
- https://github.com/scylladb/seastar

docker

SeaStar performance



How to fix it 2
Kernel space

Never leave the kernel!

- the context switch is too expensive
- put everything in the kernel?
- the kernel was hard to code for though, C code, modules etc
- create a new safe in-kernel programming language

eBPF is AWS Lambda for the Linux kernel

- attach functions to kernel events
- eBPF is a limited safe language subset, LLVM toolchain
- was very limited, being extended, eg supports function calls now
- XDP, the network framework is the most advanced part so far
- forwarding, filtering, routing, load balancing

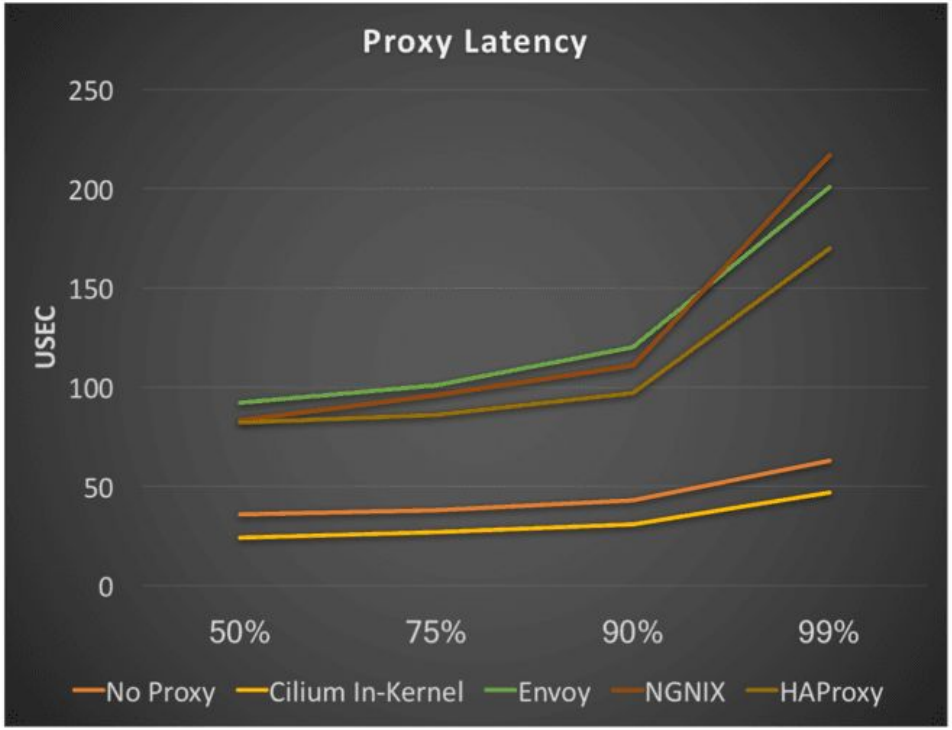
docker

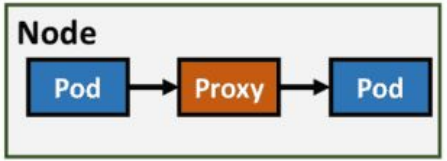
Example: Cilium

- working on a full in kernel datapath for networking
- Linux has in kernel TCP so can terminate in kernel
- Can transparently bypass TCP for local sockets
- Much faster than mixed kernel/userspace dataplane eg Nginx, Envoy
- https://github.com/cilium/cilium

docker

Cilium performance





Notes:

- This excludes parsing logic and policy rule execution
- Istio may route requests through mixer which will add latency

More on eBPF

- if you missed Thomas Graf's talk catch up later!
- XDP eXpress Data Path provides a high performance, programmable network data path in the kernel using eBPF
- networking is the most mature part of the eBPF in kernel stack
- ready for production on a modern kernel
- expect more use in other kernel subsystems soon
- provides a new set of APIs as well as traditional Posix

docker

Choosing userspace or kernel

Kernel space or userspace?

- userspace
  - use any programming language, tooling
  - debugging and programming is a little more like what you are used to
  - shortage of comprehensive libraries
- kernel space
  - you can reuse much of the Linux kernel infrastructure
  - more limited tooling

Both are getting better fast! Most people doing high performance work are doing one or the other now for new projects.

docker

Operations

docker

The modern Unix system

- Development of Unix started in 1969
- Public release 1973
- SVR 4.0 1988 led the the growth of the Unix workstation
- Sun, Apollo, Silicon Graphics; SunOS, Irix, ...
- “3M Computer” a Megabyte of memory, a Megapixel display, a Megaflop of compute, for under a Megapenny, ie $10,000
- NetBSD, Slackware, Debian, 1993; Red Hat Linux, 1995

Our modern systems look little different, other than having more packages.

Cattle not pets

- operations has changed a huge amount too in the decade
- the vast majority of operating systems never have a person log in
- most are created via APIs and automation
- immutable infrastructure: build once, then deploy
- tooling for automated installs not manual tweaking
- move away from the Sun workstation model of the 1990s
- tooling has been slow to change

docker

Immutable delivery at Netflix

“In the cloud, we know exactly what we want a server to be, and if we want to change that we simply terminate it and launch a new server with a new AMI.”

Netflix Building with Legos, 2011

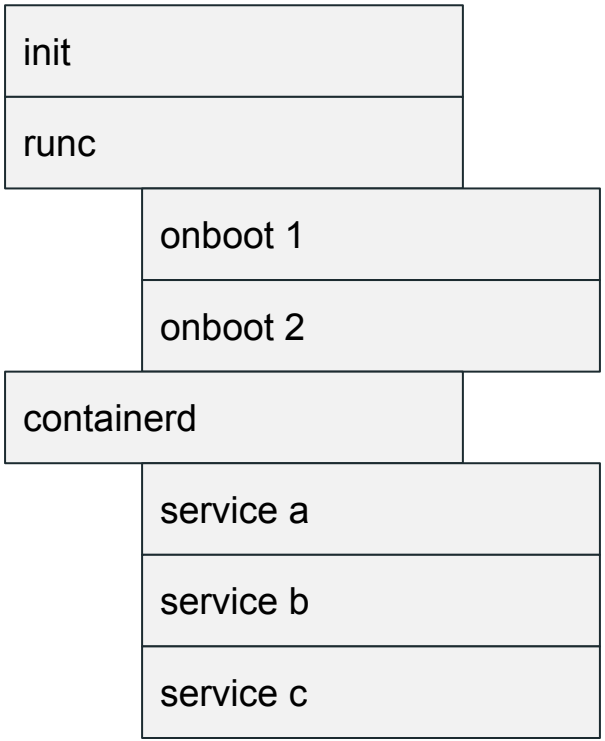
docker

LinuxKit

LinuxKit

- it is a kit, with enough pieces to get you started
- everything can easily be replaced if required
- designed to be built and tested in a CI pipeline
- build times just a minute or so
- test locally then ship to production
- minimal so boots fast
- small so secure and does not need updating so much

LinuxKit startup



sequential startup
eg network configuration, disks

services start up in parallel after initialization

same design as pods in Kubernetes

Configure this from a yaml file

```yaml
kernel:
  image: linuxkit/kernel:4.9.60
  cmdline: "console=tty0 console=ttyS0 console=ttyAMA0"
init:
  - linuxkit/init:42a92119e1ca10380e0d33e26c0cbcf85b9b3558
  - linuxkit/runc:817fdc592eac6cb7804fa1721a43a7f6e23fb50f
  - linuxkit/containerd:82be2bbb7cf83bab161ffe2a64624ba1107725ff
onboot:
  - name: dhcpcd
    image: linuxkit/dhcpcd:48831507404049660b960e4055f544917d90378e
    command: ["/sbin/dhcpcd", "--nobackground", "-f", "/dhcpcd.conf", "-1"]
services:
  - name: getty
    image: linuxkit/getty:6af22c32c98536a79230eef000e9abd06b037faa
  - name: redis
    image: redis:4.0-alpine
    capabilities:
     - CAP_NET_BIND_SERVICE
     - CAP_CHOWN
     - CAP_SETUID
     - CAP_SETGID
     - CAP_DAC_OVERRIDE
```

important differences

- root filesystem is immutable
- can run from ISO, initramfs, squashfs, ...
- no package manager
- no possibility to update at runtime
- replace with a new image to update software
- if you want dynamic services you use Docker or Kubernetes on top
- removes all complexity of install, update, reboot
- from scratch re-imagining of the modern Linux userspace

Practicalities

Simple tooling for lots of use cases

- Tooling can build most kinds of image needed to boot VMs or bare metal
  - ISO for EFI or BIOS
  - raw disk images
  - AWS AMIs
  - GCP disk format
  - QCOW2 for qemu and KVM
  - VHD
  - VMDK
  - raw kernel and initramfs
  - Raspberry Pi3 image

docker

Simple tooling for lots of use cases

- Simple build, push, run workflow for many common use cases
  - AWS
  - GCP
  - Azure
  - OpenStack
  - Vcenter
  - Packet.net iPXE
  - Hyperkit for MacOS
  - Hyper-V for Windows
  - KVM for Linux
  - VMware Fusion
  - Virtualbox

Simple tooling for lots of use cases

Generally (example Google Cloud)

```
linuxkit build file.yml
linuxkit push gcp filename
linuxkit run gcp filename
```

Some platforms have additional options. You can always use the native tooling to expose all the options.

docker

Demo

Emulation

Linus decided Linux would have a stable ABI

“If a change results in user programs breaking, it's a bug in the kernel. We never EVER blame the user programs. How hard can this be to understand?”

Linus Torvalds

This gave a stable emulation target

- Linux emulation originally implemented on NetBSD in 1995
- Solaris implementation in 2004
- ported to FreeBSD in 2006
- reimplemented and updated on SmartOS in 2015
- Windows Subsystem for Linux launched in 2016
- gVisor, userspace emulation open sourced in 2018

Linux ABI is still huge (long tail) but possible to do this with hard work.

docker

What does this mean?

- non performance critical software can be emulated elsewhere
- this is increasingly used for security isolation
- for high security applications this will become increasingly important
- gVisor is used for Google App Engine
- also just useful, eg for running existing code on Windows
  - WSL becoming increasingly good
  - integrating better with Windows programs

Don't miss Adin Scannell's gVisor talk at 4.10 today!

docker

Longer term implications

- Linux code no longer needs to run on Linux
- we can migrate to new platforms more easily in future
- highest performing applications will always be written natively
- but we can port the rest to new platforms using emulation
- platforms like App Engine and Lambda will not have to be based on Linux in the long run, they can move to new operating systems

Portability

Server hardware is becoming more standard

- The vast majority (over 70%) of Linux is drivers
- Most of these are obsolete on a server
- Linux however is also an OS for many other use cases
- Windows is now based on a slim modern portable OS called OneCore
- Many applications run on VMs anyway, little real hardware
- Most modern hardware presents a software interface
- eg SATA, USB, NVMe, SR-IOV
- Mellanox has a single driver for 10, 100, future ethernet cards
- Servers are all 64 bit
- Arm64 and AMD64 servers look the same, other than instructions

Operating systems used to be multi user

- the design of Unix was around sharing scarce resources
- computers were expensive so shared between people
- now we all have lots of computers which we do not share
- there are only a few "scarce" resources left
  - memory
  - I/O bandwidth
- for most applications these are not the limiting factors
- operating systems do not guarantee what applications actually want
  - tail latency
  - SLAs

docker

Death of legacy

- A server OS does not need legacy drivers
- There were hundreds of different 10 and 100Mb ethernet cards
- Only a handful of 10Gb+ manufacturers
- Building a new server OS is easier than ever in theory
- It hasn't happened though...
- GPUs and other specialist hardware make it more difficult

What has not (yet) changed

Declining number of operating systems

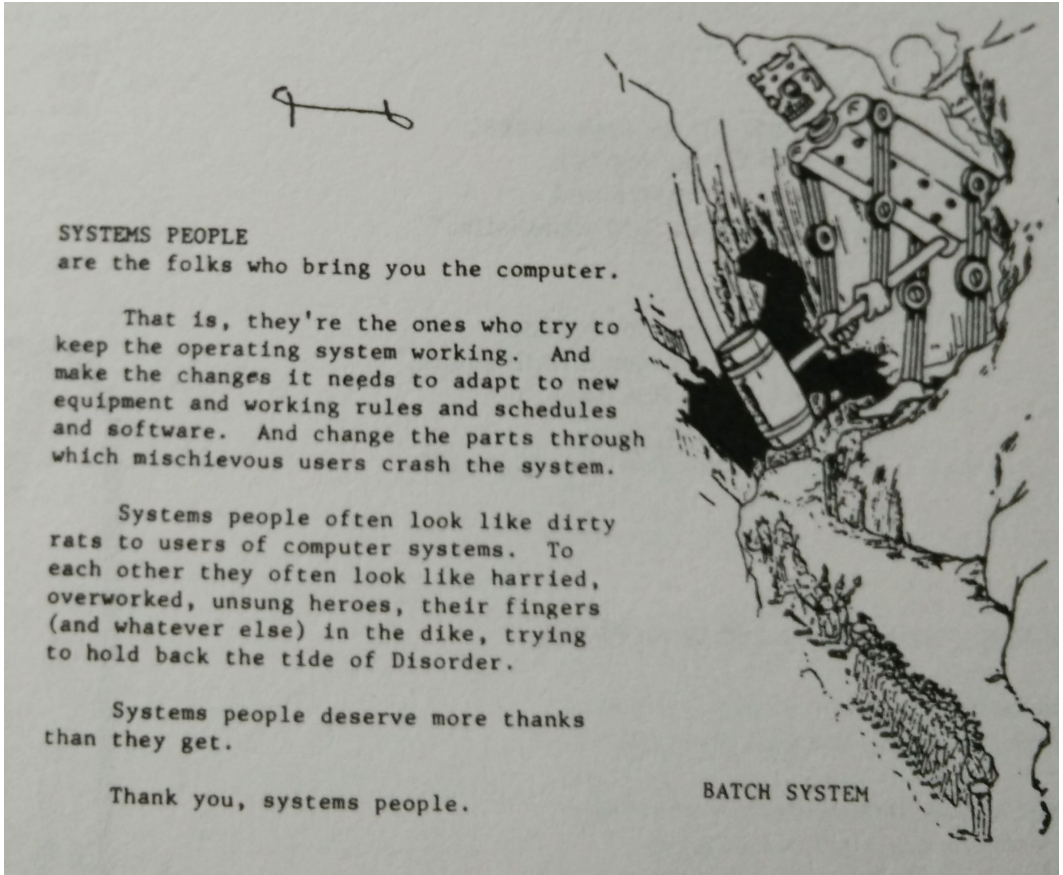
- Only three operating systems with significant market share
  - Linux, Android
  - Windows
  - iOS, MacOS
- For server applications only two have significant market share
  - Linux
  - Windows
- Linux on Azure
  - 2015 25% Linux
  - 2017 40% Linux
  - 2018 over 50% Linux

Lack of diversity

Heading towards a monoculture

- convenient that everything runs Linux, but limits new ideas
- many great systems are under used or unknown
- new kinds of system are hard to introduce

Always seen as people apart



Computer Lib, Ted Nelson, 1974

docker

Contributor diversity poor too

Creator of Linux Apologizes For Being a Jerk and Says He's Working on 'Understanding Emotions'

Vice, 2018

Linus Torvalds, the creator of the open source Linux operating system, apologized for his "lifetime of not understanding emotions" and "flippant attacks in emails."

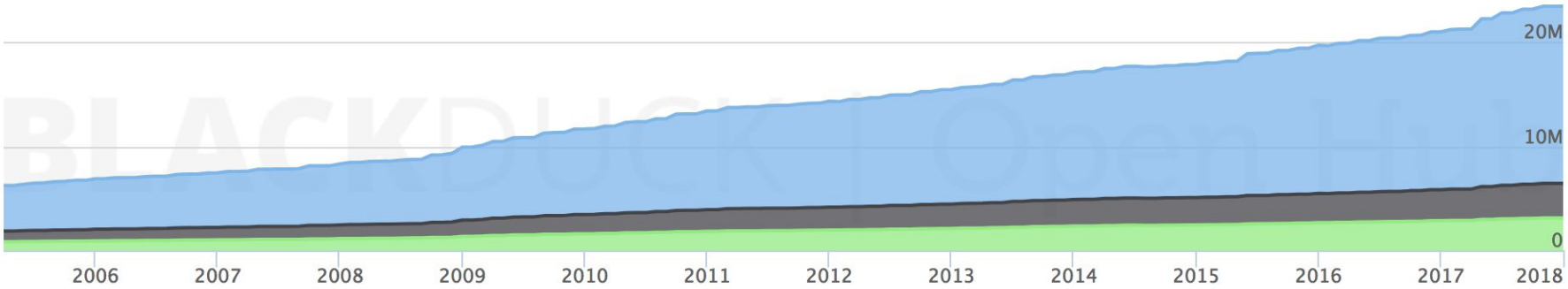
SHARE TWEET

docker

The last monolith

Lines of code in the Linux kernel

Windows is around 50 million... a Linux distro is over 500 million lines



Windows git repo

- 3.5 million files
- 270GB
- 8,421 pushes per day (on average)
- 2,500 pull requests, with 6,600 reviewers per work day (on average)
- 4,352 active topic branches
- 1,760 official builds per day

The largest git repo on the planet

docker

The last bastion of C

Operating systems are C code

- come to Bryan Cantrill's talk at 5.25 for more on this!
- other languages have not made much impact
- applies to userspace too

docker

Security

Security

- security as a driver for operating system change is slow
- Meltdown and Spectre maybe changed that a bit
- prevalent encryption will too, as key management becomes common
- security has yet to change the design space a lot
- secure IoT is one space that will change
- demand for security and privacy slowly rising
  - still denial this will change how applications are built
- Linux has preferred agility over security

docker

Windows is probably more secure now

Trustworthy Computing is computing that is as available, reliable and secure as electricity, water services and telephony... Security models should be easy for developers to understand and build into their applications.

Bill Gates, Trustworthy Computing, 2002

Unikernels: the radical answer

Unikernels

- operating system as a library you link to your application
- boot your application directly on a VM or hardware
- just run your application, nothing else
- specialise everything for a single application
- not monolithic
- pick and choose different implementations from libraries
- pick the language you want to use

docker

Unikernels: successes

- Microsoft shipped SQL Server for Linux as a unikernel
- growing open source communities around two projects
  - Mirage (OCaml)
  - IncludeOS (C++)
- commercial company NanoVMs
- other smaller projects
- many closed source internal projects inside companies

docker

The Library OS

Unlike the rest of the world

- operating system code not designed for reuse
- designed to run in a peculiar environment
- not enough system libraries for fast development
- most OS code is in C, developers want C++, Rust, Go, OCaml, ...
- you can borrow code from the BSDs, eg TCP stacks
- unikernels are building those libraries too
- as more people work with these tools they get better!
- have not changed the world yet...

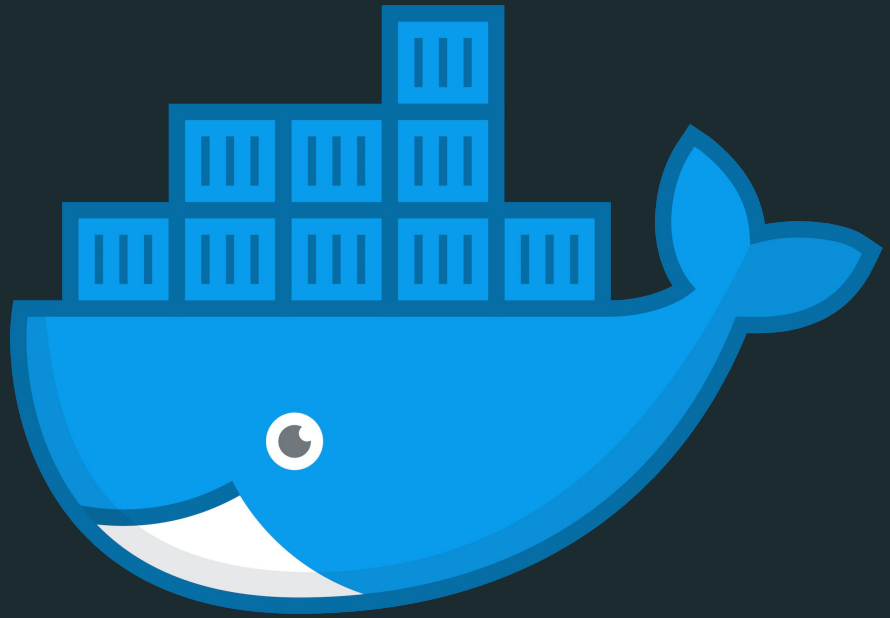
docker

Summary

Summary

- Operating systems did change after all!
- Performance meant we created two new ways to run code
  - userspace, self contained using little of OS
  - in kernel eBPF, Lambda for Linux
- Operations changing uses, more like LinuxKit than the 1990s
- emulation is making code more portable
- security will lead to the next changes
- unikernels are coming along, interest growing
- diversity will fight the monoculture, make that happen!

docker



THANK YOU