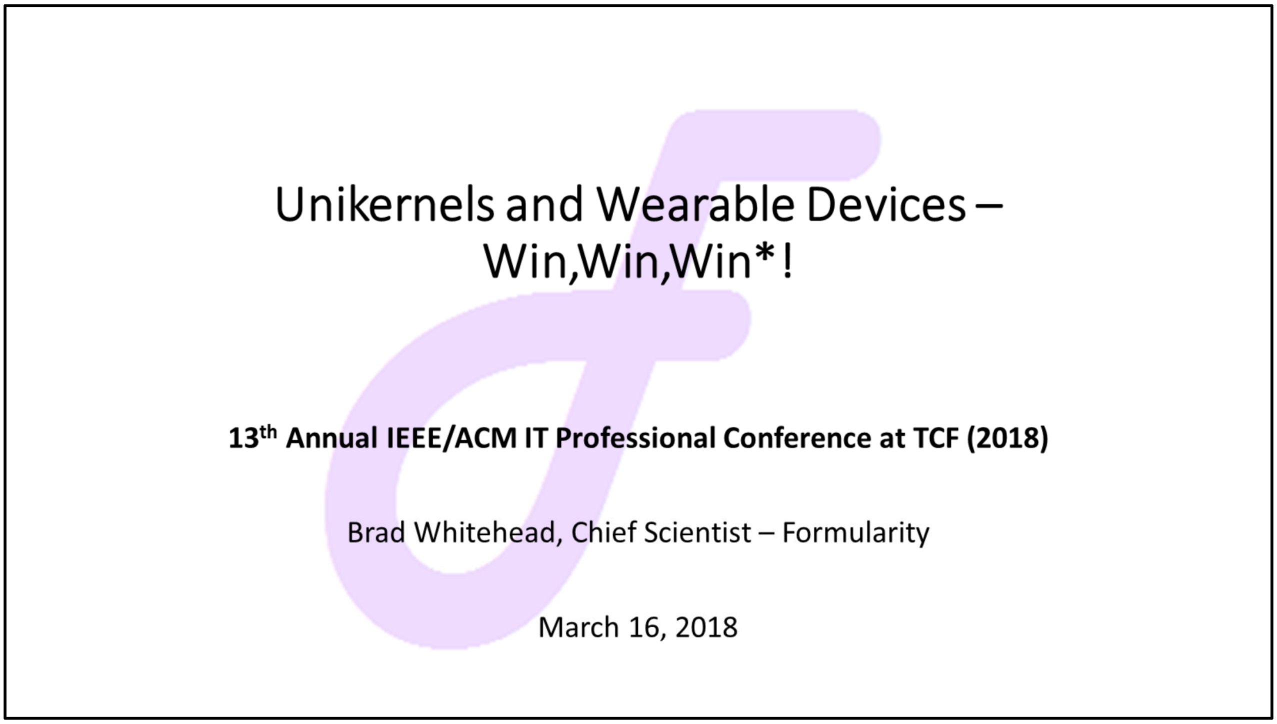


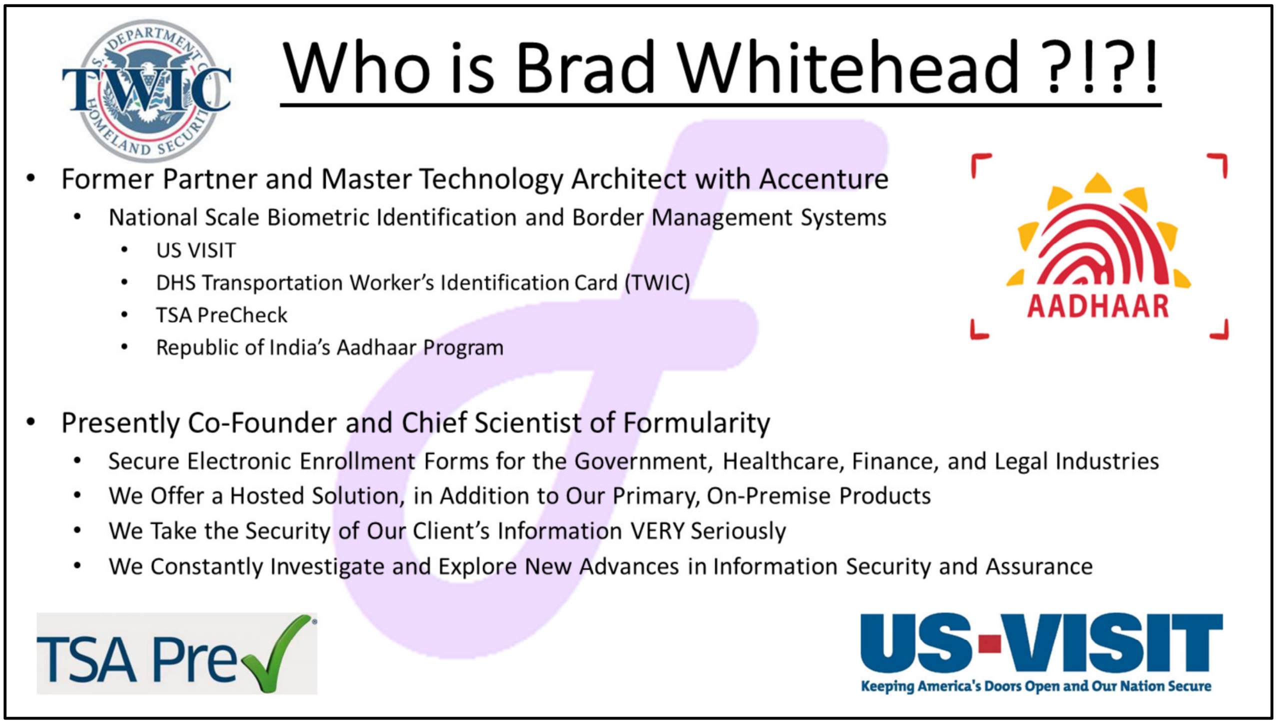
Unikernels and Wearable Devices – Win,Win,Win*!

13th Annual IEEE/ACM IT Professional Conference at TCF (2018)

Brad Whitehead, Chief Scientist – Formularity

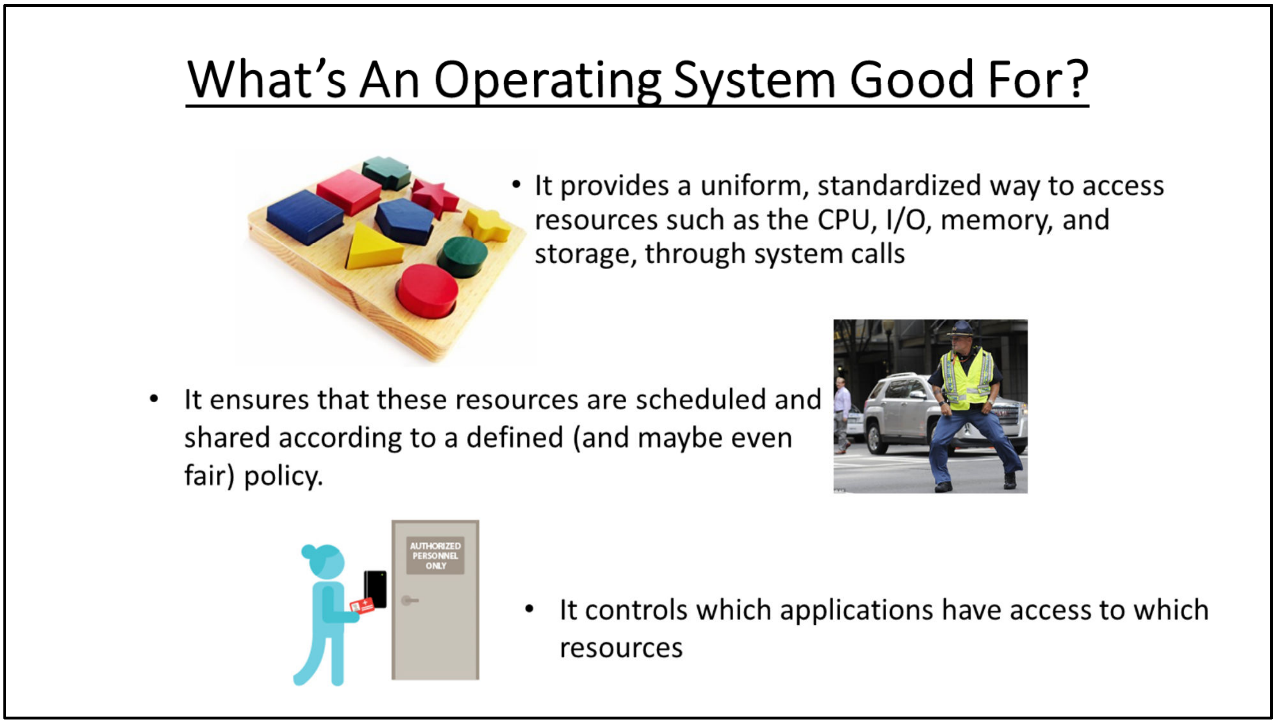
March 16, 2018

Good afternoon. I'm Brad Whitehead. I'm the Chief Scientist at Formularity. Before I get started, I'd like to thank the folks of the IEEE and the ACMfor giving me the opportunity to talk to you today at ITPC 2018. In particular, I'd like to thank David Sol and Al Katz. They have selected an excellent agenda of both interesting and educational presentations. But most importantly, I'd like to thank you for choosing to attend this talk. I hope I make it worth your while! Formularity is a small company and you may not be familiar with us. We develop high security electronic enrollment forms for things like national identity management programs, financial institutions, and national health care program enrollments. We are dedicated to making sure the sensitive personal information you provide on our forms is secure and protected at all times. In addition to our forms being run by our clients in their own data centers, we also offer a hosted solution. Since we are potentially storing valuable, sensitive personal information of our clients' customers, we are extremely concerned about security! We take a number of steps to ensure information remains encrypted; while at rest, while in motion, and especially inbetween ;-) We are constantly reviewing not only the threats, but new technologies that can help mitigate these threats. I'm here today to discuss one of these promising new technologies, unikernels. Unlike typical security measures that impose additional complexity and require additional resources, unikernels are unique in that they significantly simplify the software and operations, and they reduce resource requirements. Hence the title of today's talk – "Win, Win, Win"! To be clear, Formularity is not yet using unikernels in production, but we are experimenting with them and anticipate their use in the future.


TWIC
U.S. DEPARTMENT OF HOMELAND SECURITY
AADHAAR
TSA Pre✓
US-VISIT
Keeping America's Doors Open and Our Nation Secure

Who is Brad Whitehead ?!?!

- Former Partner and Master Technology Architect with Accenture
  - National Scale Biometric Identification and Border Management Systems
    - US VISIT
    - DHS Transportation Worker's Identification Card (TWIC)
    - TSA PreCheck
    - Republic of India's Aadhaar Program
- Presently Co-Founder and Chief Scientist of Formularity
  - Secure Electronic Enrollment Forms for the Government, Healthcare, Finance, and Legal Industries
  - We Offer a Hosted Solution, in Addition to Our Primary, On-Premise Products
  - We Take the Security of Our Client's Information VERY Seriously
  - We Constantly Investigate and Explore New Advances in Information Security and Assurance



What’s An Operating System Good For?

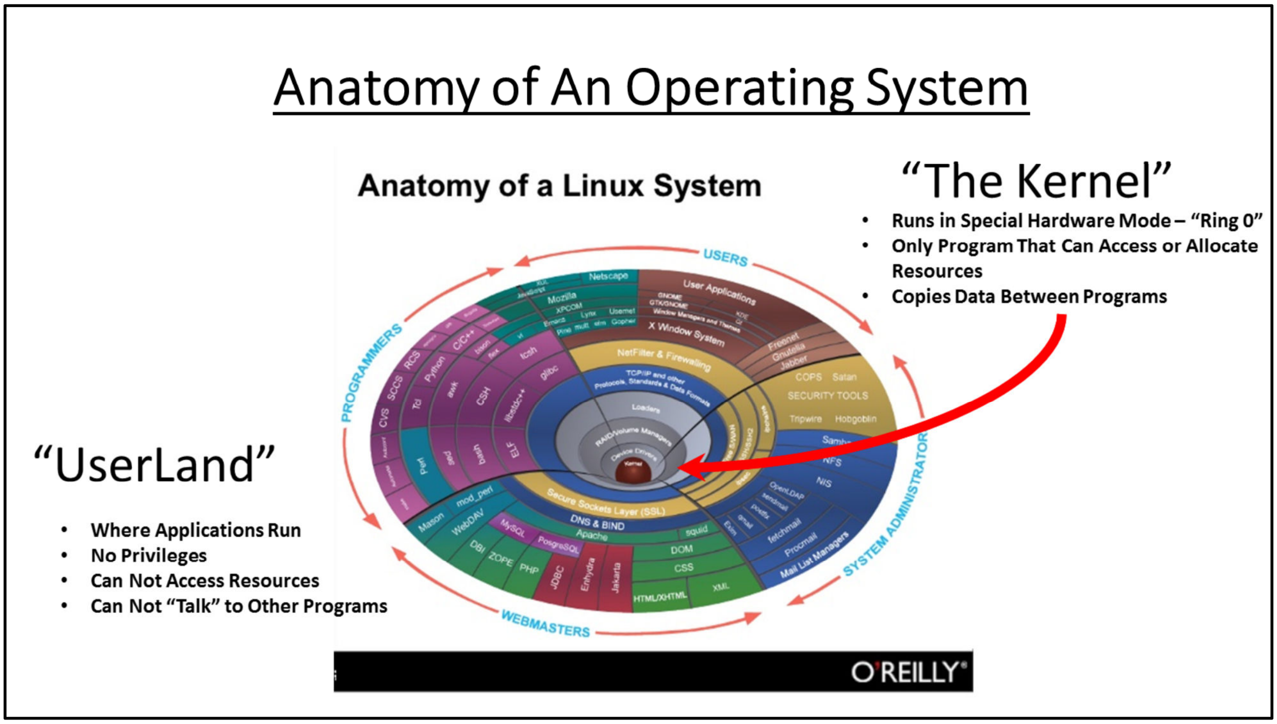
- It provides a uniform, standardized way to access resources such as the CPU, I/O, memory, and storage, through system calls
- It ensures that these resources are scheduled and shared according to a defined (and maybe even fair) policy.
- It controls which applications have access to which resources

What’s an Operating System Good For?

Uniform, standardized access to resources (CPU, I/O, memory, storage, etc.)

Safe sharing of these resources (traffic cop)

Security



Anatomy of An Operating System

“The Kernel”

- Runs in Special Hardware Mode – “Ring 0”
- Only Program That Can Access or Allocate Resources
- Copies Data Between Programs

“UserLand”

- Where Applications Run
- No Privileges
- Can Not Access Resources
- Can Not “Talk” to Other Programs

Anatomy of an Operating System

User Interface (CLI or GUI)

Standard set of tools and applications (“Userland”)

Kernel (privileged, separated from Userland by hardware)

Monolithic (Linux even includes a web server in the kernel!)

Famous flame war between Torvald Linus and Dr. Andrew Tanenbaum, 1992 Usenet

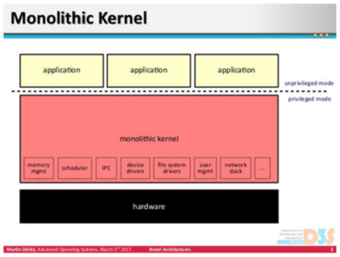
Microkernel (Mach, Minix)

Context switches

data copying

Unikernels – Why we are here today!

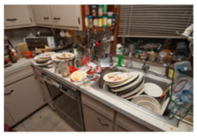
What Kind of Kernel Are You?



Monolithic Kernel

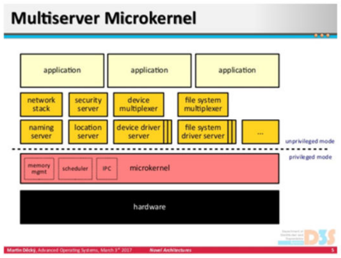
Martin Děcký, Advanced Operating Systems, March 3rd 2017 Novel Architectures 3

- A Monolithic Kernel?
  - Also Known as the “Kitchen Sink” Approach
    (Linux even includes a web server in the kernel!)
  - Famous flame war between Linus Torvald and Dr. Andrew Tanenbaum, 1992 Usenet
  - All Major Commercial OSs Have Monolithic Kernels



"The facts, Ma'am. Just the facts."

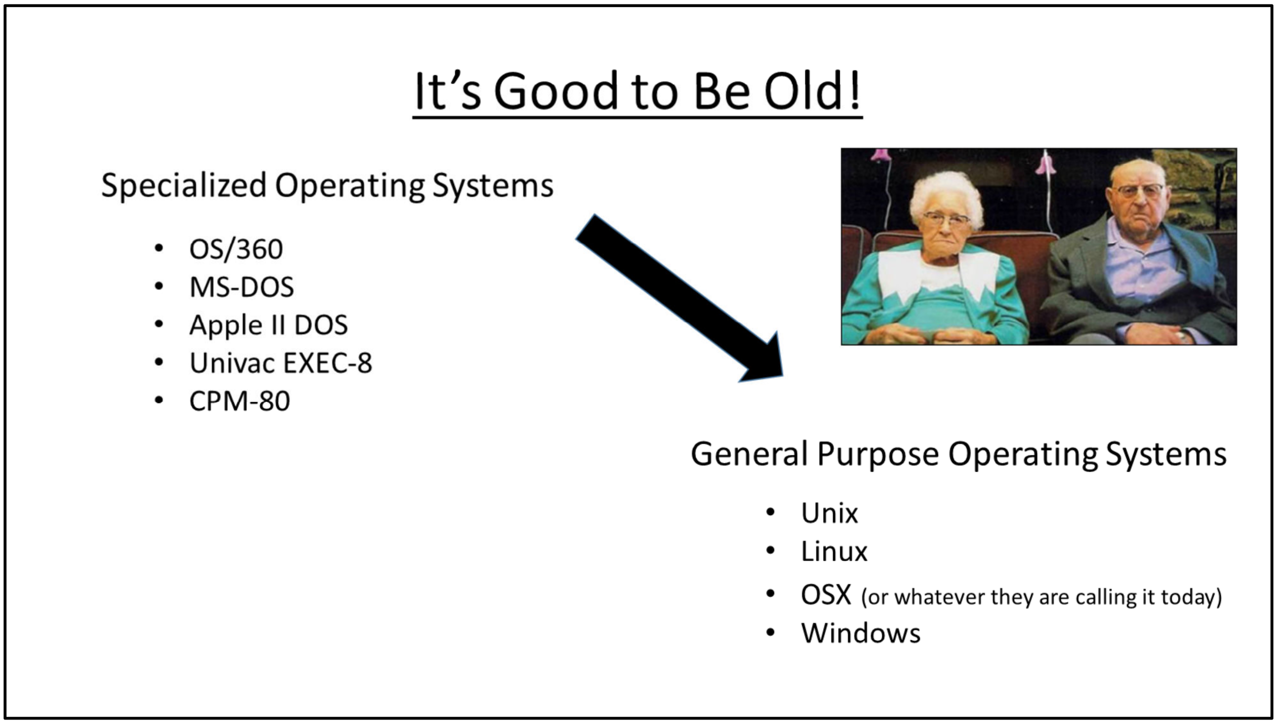
- or a Microkernel (Mach, Minix)?
  - Only Essential Functions in the Kernel
  - Better Flexibility and Security
  - Poorer Performance with Context Switching and Data Copying



Multiserver Microkernel

Martin Děcký, Advanced Operating Systems, March 3rd 2017 Novel Architectures 5

or a Unikernel? – Why We Are Here Today!



It’s Good to Be Old!

Specialized Operating Systems

- OS/360
- MS-DOS
- Apple II DOS
- Univac EXEC-8
- CPM-80

General Purpose Operating Systems

- Unix
- Linux
- OSX (or whatever they are calling it today)
- Windows

It’s Good to Be Old!

- Specialized OSs
  - MVS/360
  - MS-DOS
  - Apple II BASIC
  - CPM-80
- Standardized OSs – Huge Multiuser Monoliths
  - Unix(es)
  - Linux (22 million SLOC and 17 different languages)
    - RHEL - Userland of 420 million SLOC
  - Windows (50 million SLOC)
- Processes and Threads
  - Faster instantiation
  - Shared memory/ no kernel transitions

HUGE General Purpose Operating Systems!

- Linux
  - Kernel – 22 Million SLOC
    - Written in 17 Different Programming Languages
  - Red Hat Enterprise Linux UserLand – 420 Million SLOC
- Windows
  - 50 Million SLOC

These OSs:

- Take Seconds or Minutes to Boot!
- Require Hundreds of Megabytes of Memory!
- Consume Watts of Power!

And Thinking About the Number of Undiscovered Errors Makes My Head Hurt!

Processes and Threads, Oh Dear!!

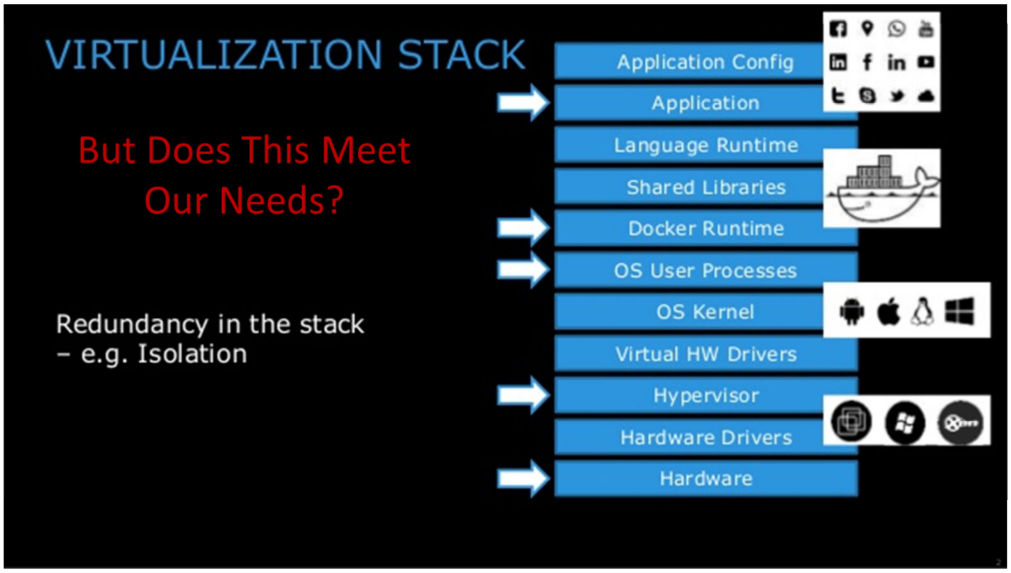
(or “Mr. Gorbachev, Tear Down This Wall”)

- Operating System Processes Isolate Memory and Communications between Programs
  - However, Processes are heavy and take time to initiate
  - And there are significant performance penalties when information has to be transferred between applications because the kernel must be involved
- Threads Remove this Isolation
  - Faster to initiate
  - Every Thread in a Program has access to all the information in the Program

“Threads” 16bit.com

In Effect, Developers
Have Voted
That They Want Less Protection Than Operating Systems Initially Provided

So What Do We Have Today?



VIRTUALIZATION STACK

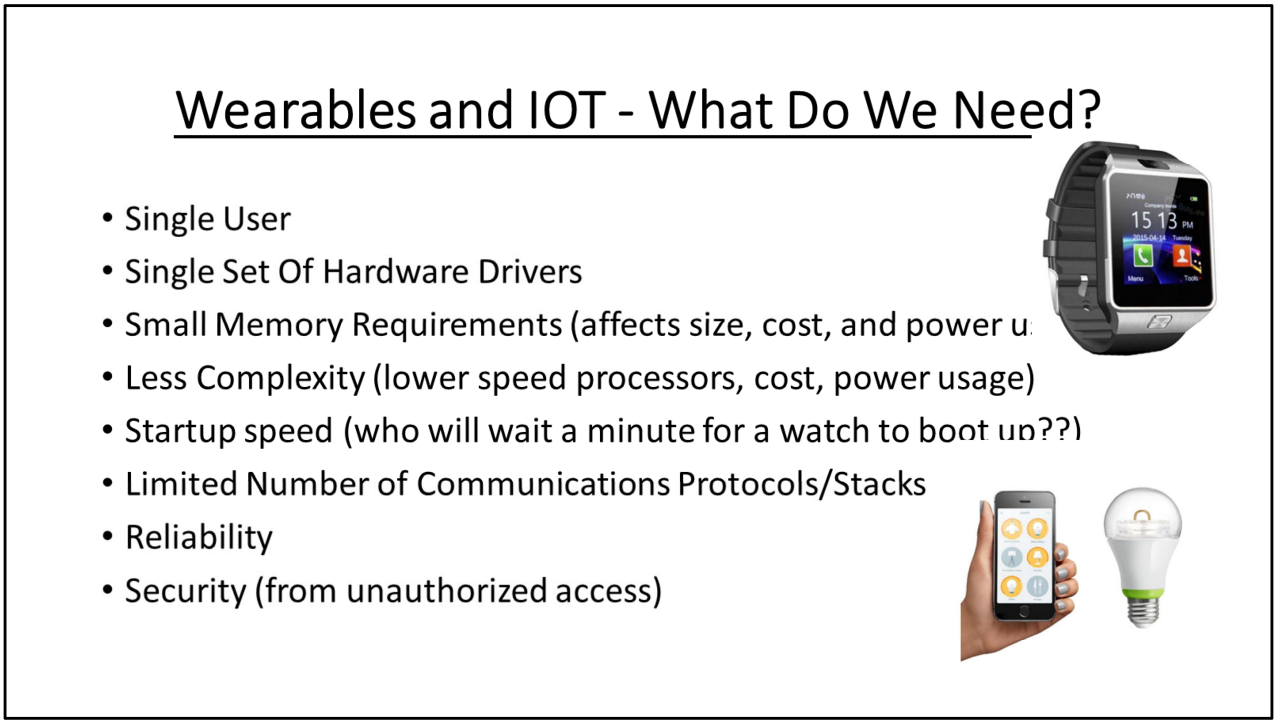
But Does This Meet Our Needs?

Redundancy in the stack
– e.g. Isolation

Application Config
Application
Language Runtime
Shared Libraries
Docker Runtime
OS User Processes
OS Kernel
Virtual HW Drivers
Hypervisor
Hardware Drivers
Hardware

State-of-the-Art

- Shared Servers (Hardware)
- Hypervisor (running an OS)
- Virtual Machines (each running an OS)
- Containers (each with a User Land)
- Single Application (and user)

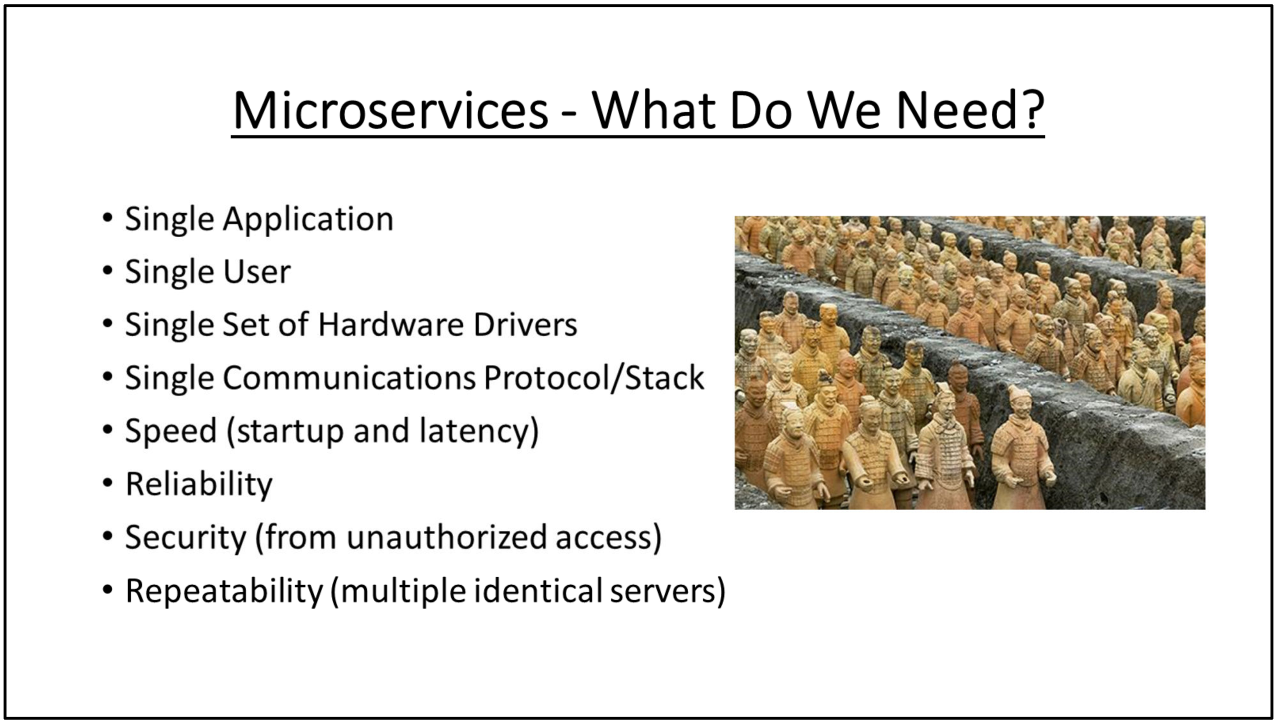


Wearables and IOT - What Do We Need?

- Single User
- Single Set Of Hardware Drivers
- Small Memory Requirements (affects size, cost, and power u
- Less Complexity (lower speed processors, cost, power usage)
- Startup speed (who will wait a minute for a watch to boot up??)
- Limited Number of Communications Protocols/Stacks
- Reliability
- Security (from unauthorized access)

Unikernels – Let’s Cut Out The Middle

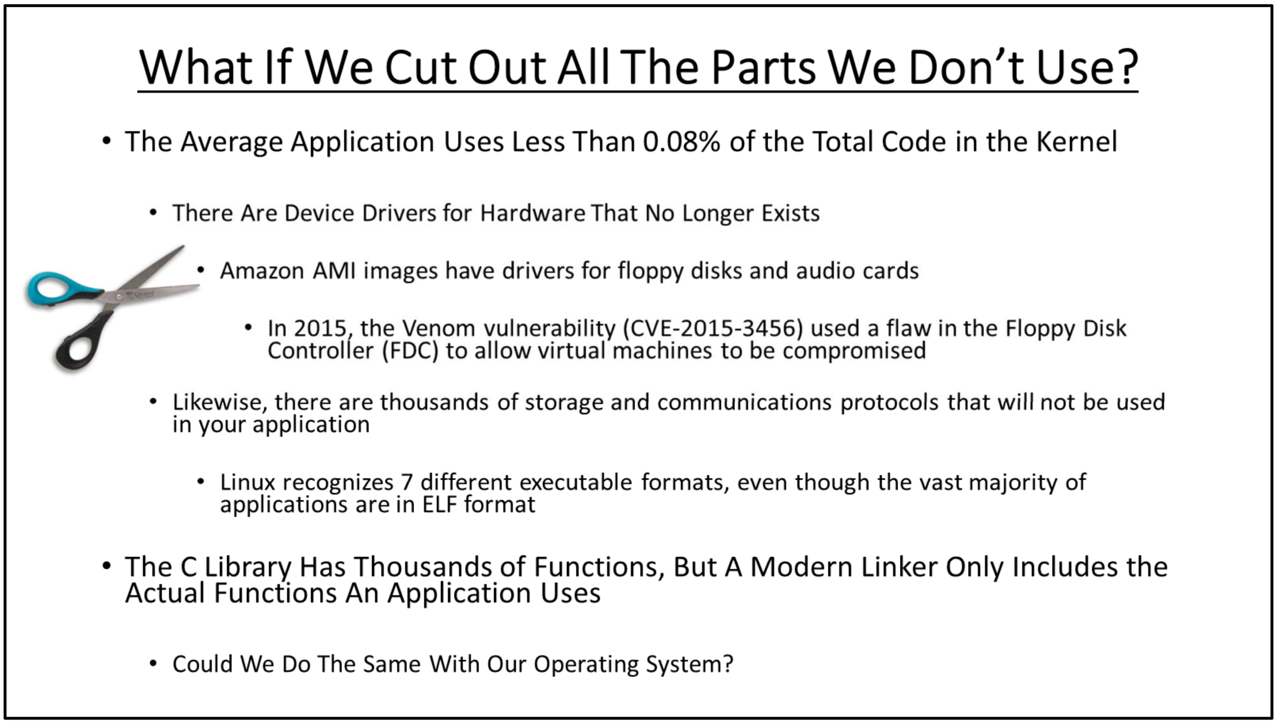
- IOT/Wearables
  - Dedicated hardware
  - Unikernel
  - Single Application



Microservices - What Do We Need?

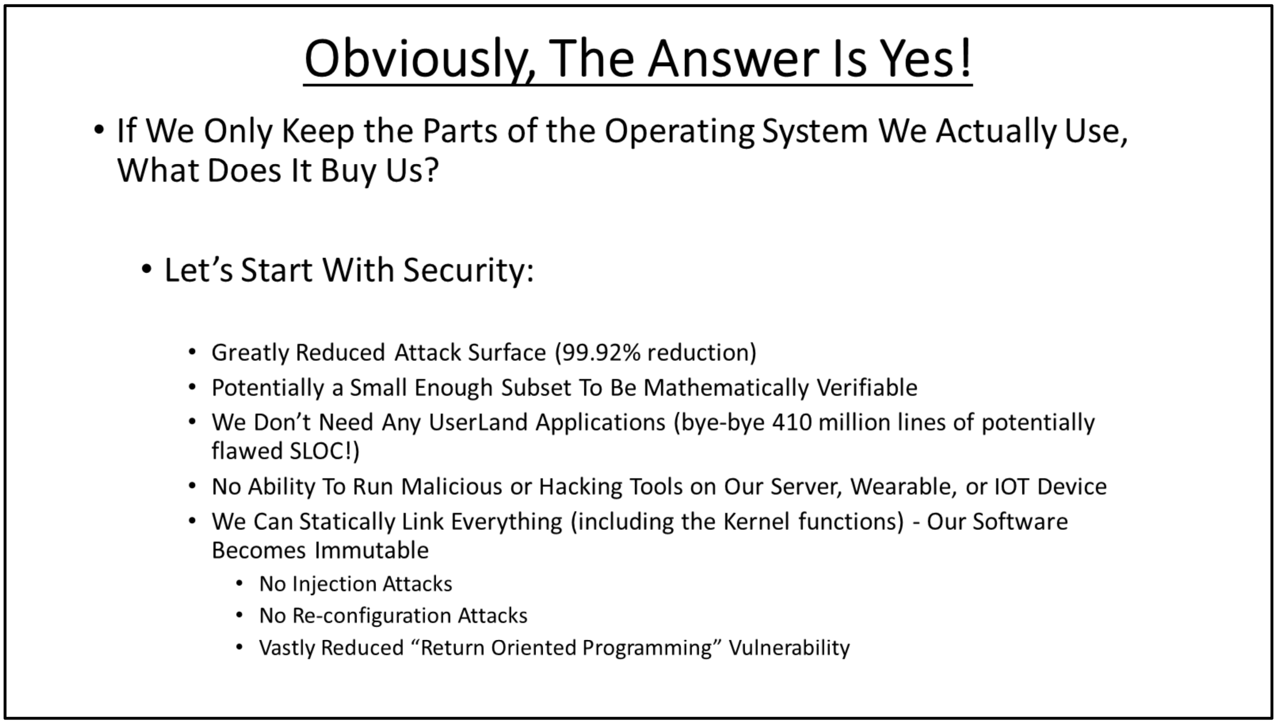
- Single Application
- Single User
- Single Set of Hardware Drivers
- Single Communications Protocol/Stack
- Speed (startup and latency)
- Reliability
- Security (from unauthorized access)
- Repeatability (multiple identical servers)

Unikernels – Let’s Cut Out The Middle
Microservices
Shared Servers (Hardware)
Hypervisor (running an OS)
Virtual Machine (running a unikernel OS)
Single Application



What If We Cut Out All The Parts We Don’t Use?

- The Average Application Uses Less Than 0.08% of the Total Code in the Kernel
  - There Are Device Drivers for Hardware That No Longer Exists
    - Amazon AMI images have drivers for floppy disks and audio cards
      - In 2015, the Venom vulnerability (CVE-2015-3456) used a flaw in the Floppy Disk Controller (FDC) to allow virtual machines to be compromised
  - Likewise, there are thousands of storage and communications protocols that will not be used in your application
    - Linux recognizes 7 different executable formats, even though the vast majority of applications are in ELF format
- The C Library Has Thousands of Functions, But A Modern Linker Only Includes the Actual Functions An Application Uses
  - Could We Do The Same With Our Operating System?

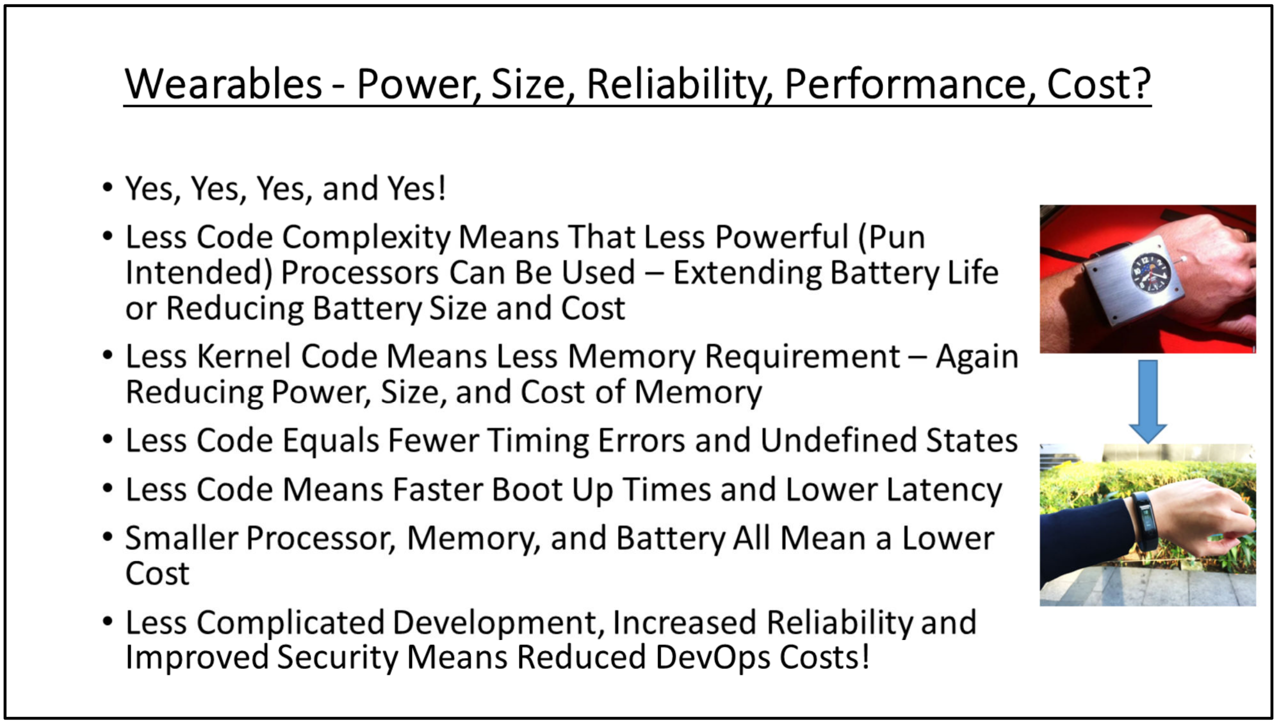


Obviously, The Answer Is Yes!

- If We Only Keep the Parts of the Operating System We Actually Use, What Does It Buy Us?
  - Let’s Start With Security:
    - Greatly Reduced Attack Surface (99.92% reduction)
    - Potentially a Small Enough Subset To Be Mathematically Verifiable
    - We Don’t Need Any UserLand Applications (bye-bye 410 million lines of potentially flawed SLOC!)
    - No Ability To Run Malicious or Hacking Tools on Our Server, Wearable, or IOT Device
    - We Can Statically Link Everything (including the Kernel functions) - Our Software Becomes Immutable
      - No Injection Attacks
      - No Re-configuration Attacks
      - Vastly Reduced “Return Oriented Programming” Vulnerability

Security

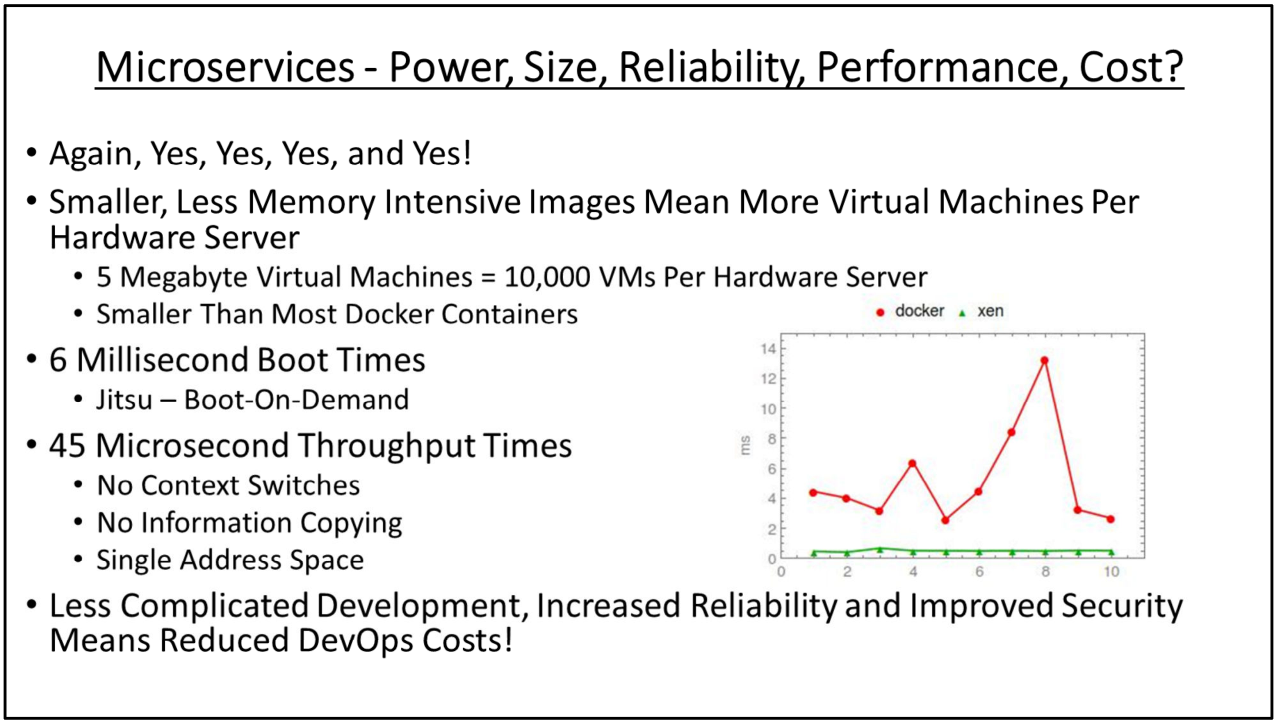
Reduced attack surface – .08% of typical
Small enough to possibly be mathematically verified
No tools (no shell, etc.)
No standard system calls – uses ASLR library calls
Immutable – modules and dynamic libraries can’t be added



Wearables - Power, Size, Reliability, Performance, Cost?

- Yes, Yes, Yes, and Yes!
- Less Code Complexity Means That Less Powerful (Pun Intended) Processors Can Be Used – Extending Battery Life or Reducing Battery Size and Cost
- Less Kernel Code Means Less Memory Requirement – Again Reducing Power, Size, and Cost of Memory
- Less Code Equals Fewer Timing Errors and Undefined States
- Less Code Means Faster Boot Up Times and Lower Latency
- Smaller Processor, Memory, and Battery All Mean a Lower Cost
- Less Complicated Development, Increased Reliability and Improved Security Means Reduced DevOps Costs!

Reduced Cost
- IOT/Wearable
  - Less processing speed
  - Less memory
  - Less power/longer battery life



Microservices - Power, Size, Reliability, Performance, Cost?

- Again, Yes, Yes, Yes, and Yes!
- Smaller, Less Memory Intensive Images Mean More Virtual Machines Per Hardware Server
  - 5 Megabyte Virtual Machines = 10,000 VMs Per Hardware Server
  - Smaller Than Most Docker Containers
- 6 Millisecond Boot Times
  - Jitsu – Boot-On-Demand
- 45 Microsecond Throughput Times
  - No Context Switches
  - No Information Copying
  - Single Address Space
- Less Complicated Development, Increased Reliability and Improved Security Means Reduced DevOps Costs!

Reduced Cost

Microservices

Smaller instances or more VMs per instance - 5MB per VM, 10K VMs/hardware server

Higher performance

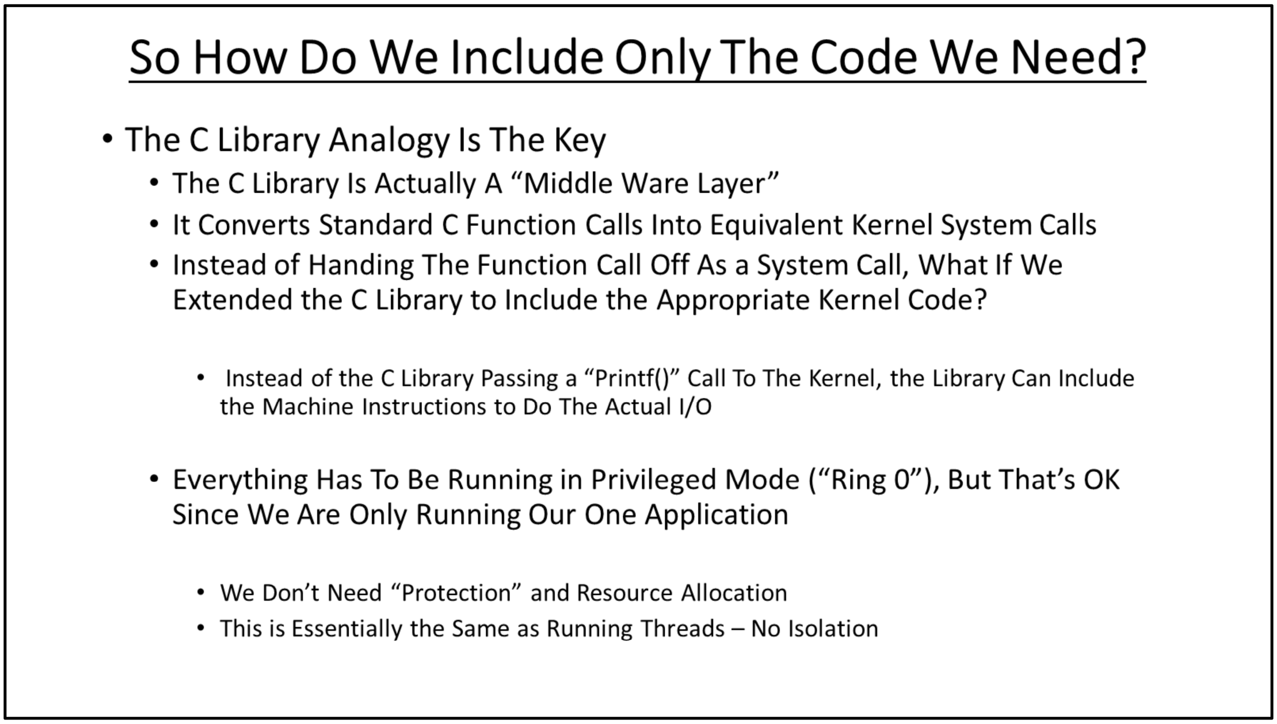
6 millisecond boot

No context switches

No memory copying between kernel and applications

Server-less Functions (with servers)! – 45 microsecond response

Jitsu



So How Do We Include Only The Code We Need?

- The C Library Analogy Is The Key
  - The C Library Is Actually A “Middle Ware Layer”
  - It Converts Standard C Function Calls Into Equivalent Kernel System Calls
  - Instead of Handing The Function Call Off As a System Call, What If We Extended the C Library to Include the Appropriate Kernel Code?
    - Instead of the C Library Passing a “Printf()” Call To The Kernel, the Library Can Include the Machine Instructions to Do The Actual I/O
  - Everything Has To Be Running in Privileged Mode (“Ring 0”), But That’s OK Since We Are Only Running Our One Application
    - We Don’t Need “Protection” and Resource Allocation
    - This is Essentially the Same as Running Threads – No Isolation

Where do unikernels come from?

Decomposing existing monolithic operating systems

BSD Unix/Mach microkernel – a large number of OS functions have been moved out of the kernel

Anykernel (NetBSD)

Specialized language libraries (OCaml, Haskell, Erlang)

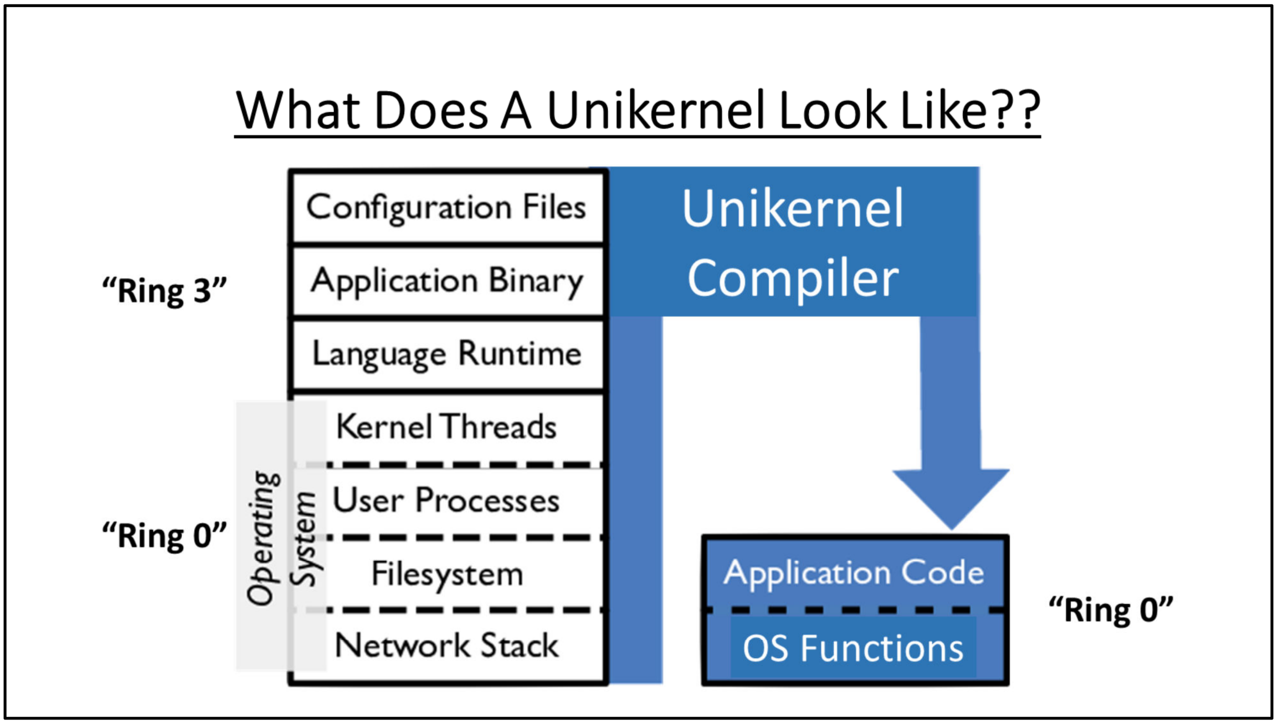
This Is How The “Library Operating System” Concept Started

- Common Operating System Functions, Drivers, and Protocols Were Written As a Library of Functions
- The First Libraries Were Written in Functional Languages
  - MirageOS – OCaml
  - HaLVM – Haskell
  - Ling – Erlang
- These Proved the Concept – More Secure, Higher Performance, Increased Reliability, Reduced Resources, Single Address Space, Reduced or Eliminated Scheduling
- Limited to New Application Development
- When You Link The “Library Operating System” Functions to Your Application, You Have a Single Executable That Runs Directly on Hardware Or a Hypervisor…

…You Have A Unikernel!

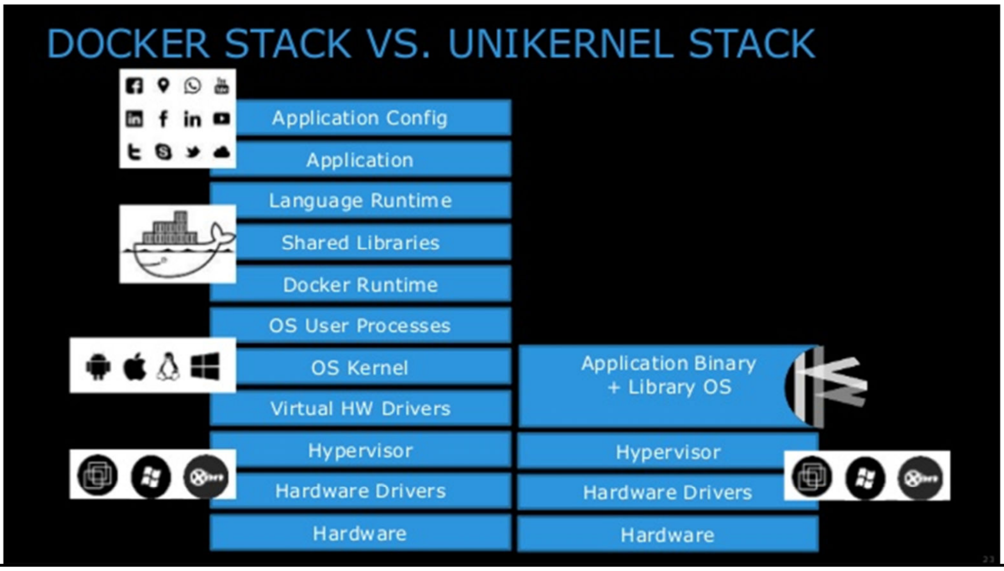
“Of Course It Runs NetBSD!”

- NetBSD, a Version of Unix, is Famous For Its Ability To Be Ported To New Hardware
- It’s a Monolithic Kernel, But Internally Its Been Structured Into Well Defined Functions and Layers
- Library of NetBSD Functions Have Been Created, Called “The AnyKernel” Concept
- The AnyKernel Concept Allows Existing Application Code, Designed For the Linux or Unix Operating System To Be Statically Linked With Operating System Functions and Drivers, Forming A Unikernel!

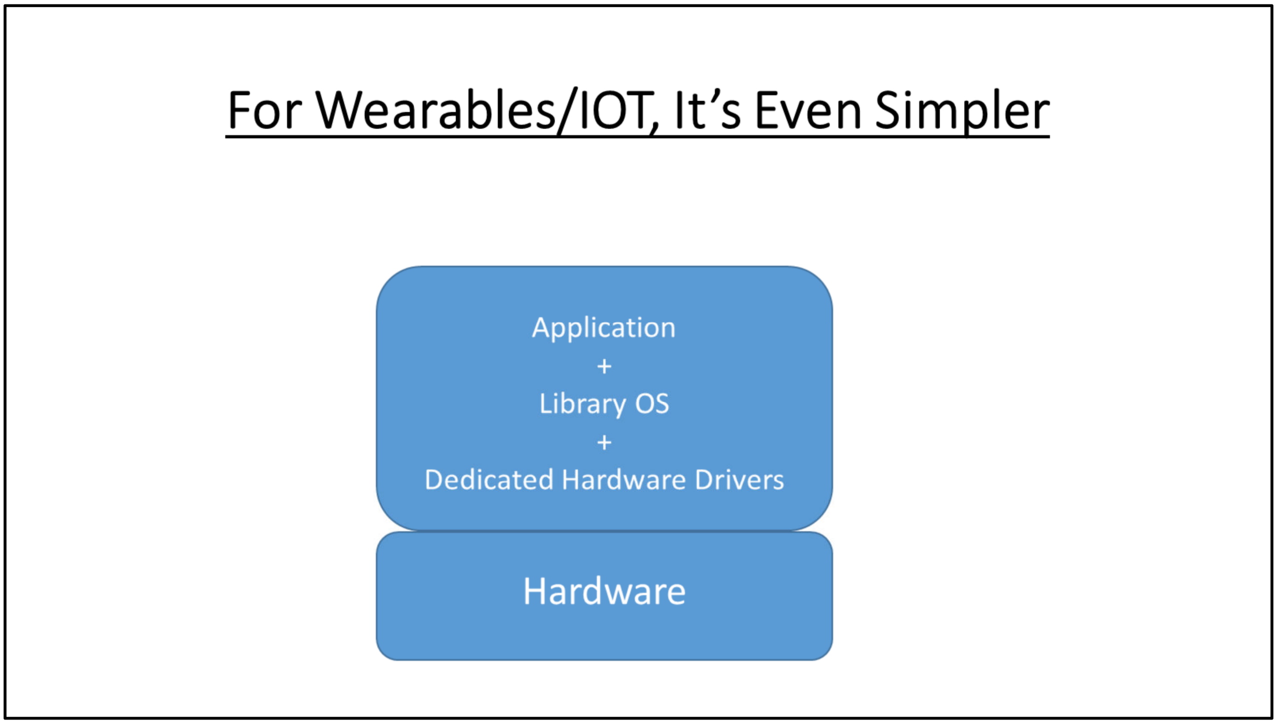


What Does A Unikernel Look Like??

Microservices - Now What Do We Have?



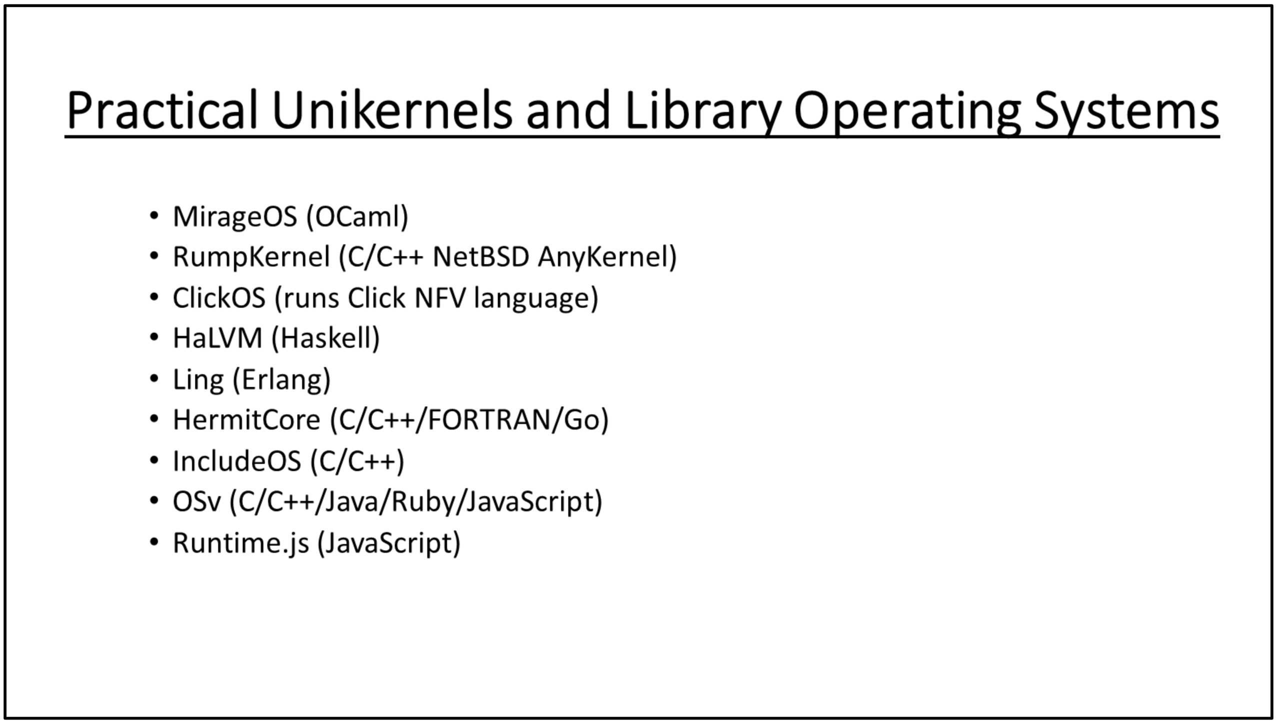
DOCKER STACK VS. UNIKERNEL STACK



For Wearables/IOT, It’s Even Simpler

Application
+
Library OS
+
Dedicated Hardware Drivers

Hardware

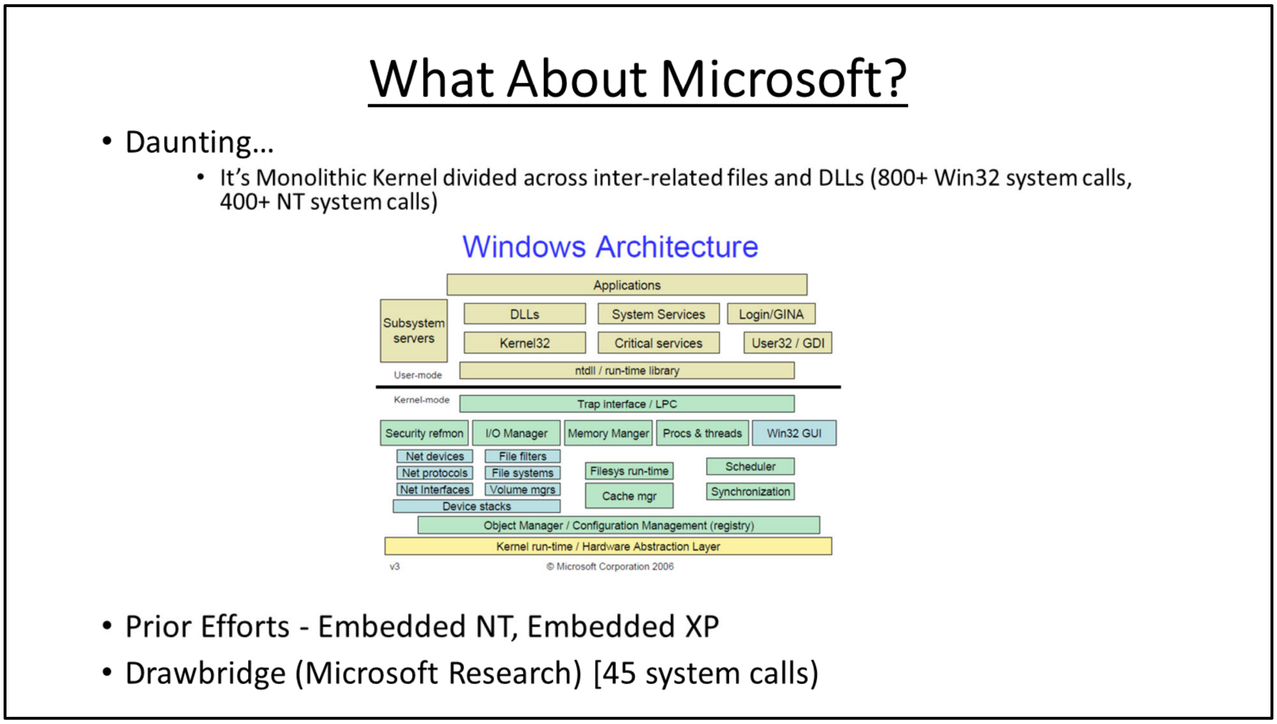


Practical Unikernels and Library Operating Systems

- MirageOS (OCaml)
- RumpKernel (C/C++ NetBSD AnyKernel)
- ClickOS (runs Click NFV language)
- HaLVM (Haskell)
- Ling (Erlang)
- HermitCore (C/C++/FORTRAN/Go)
- IncludeOS (C/C++)
- OSv (C/C++/Java/Ruby/JavaScript)
- Runtime.js (JavaScript)

Practical Unikernels and Library Operating Systems

MirageOS
RumpKernel
ClickOS (runs Click NFV language)
HaLVM (Haskell)
HermitCore (C/C++/FORTRAN/Go)
IncludeOS (C/C++)
OSv (C/C++/Java/Ruby/JavaScript)
Runtime.js (JavaScript)



What About Microsoft?

- Daunting…
  - It’s Monolithic Kernel divided across inter-related files and DLLs (800+ Win32 system calls, 400+ NT system calls)

Windows Architecture

Applications

Subsystem servers

DLLs

System Services

Login/GINA

Kernel32

Critical services

User32 / GDI

User-mode

ntdll / run-time library

Kernel-mode

Trap interface / LPC

Security refmon

I/O Manager

Memory Manger

Procs & threads

Win32 GUI

Net devices

Net protocols

Net Interfaces

File filters

File systems

Volume mgrs

Device stacks

Filesys run-time

Cache mgr

Scheduler

Synchronization

Object Manager / Configuration Management (registry)

Kernel run-time / Hardware Abstraction Layer

v3

© Microsoft Corporation 2006

- Prior Efforts - Embedded NT, Embedded XP
- Drawbridge (Microsoft Research) [45 system calls)

Windows is hopeless! (Not Really)

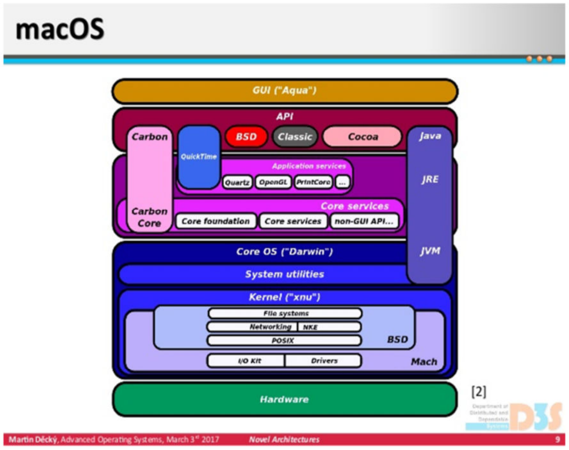
Monolithic kernel divided across inter-related files and DLLs (800+ Win32 system calls, 400+ NT system calls)

Embedded NT, embedded XP

Drawbridge (Microsoft Research) [45 system calls)

Apple Might Be Best Poised to Create a Commercial Unikernel

- OSX Started with Mach Microkernel
- Most of OSX UserLand is BSD
- However…
  - Performance Modifications Have Made OSX a Monolithic Kernel
  - Apple Has Shown No Interest In Non-Apple Hardware or Data Center Environments

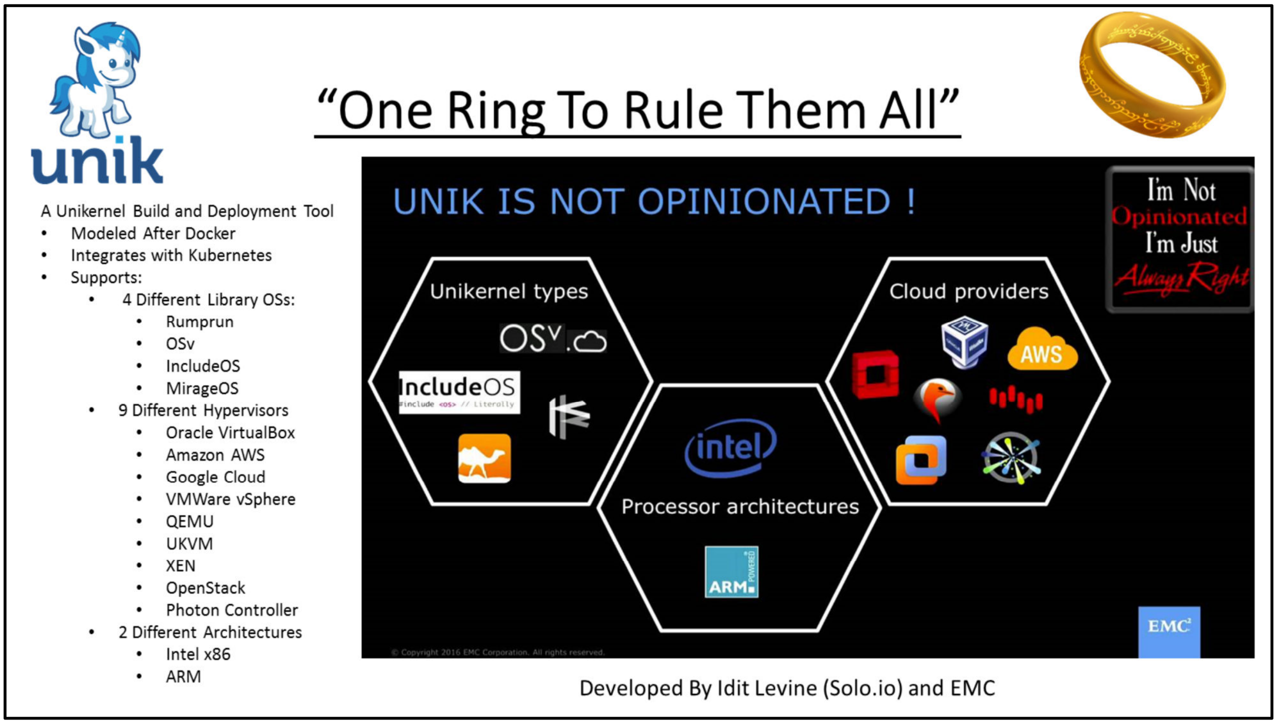


macOS

[2]

Martin Děcký, Advanced Operating Systems, March 3rd 2017 Novel Architectures 9

Apple is best poised to create a commercial unikernel, but ???



“One Ring To Rule Them All”

unik

A Unikernel Build and Deployment Tool

- Modeled After Docker
- Integrates with Kubernetes
- Supports:
  - 4 Different Library OSs:
    - Rumprun
    - OSv
    - IncludeOS
    - MirageOS
  - 9 Different Hypervisors
    - Oracle VirtualBox
    - Amazon AWS
    - Google Cloud
    - VMWare vSphere
    - QEMU
    - UKVM
    - XEN
    - OpenStack
    - Photon Controller
  - 2 Different Architectures
    - Intel x86
    - ARM

UNIK IS NOT OPINIONATED !
Unikernel types
OSv
IncludeOS
#include <os> // Literally
Processor architectures
intel
ARM
Cloud providers
AWS
I'm Not Opinionated I'm Just Always Right
EMC²
© Copyright 2016 EMC Corporation. All rights reserved.

Developed By Idit Levine (Solo.io) and EMC

One Ring to Rule Them All

Unik

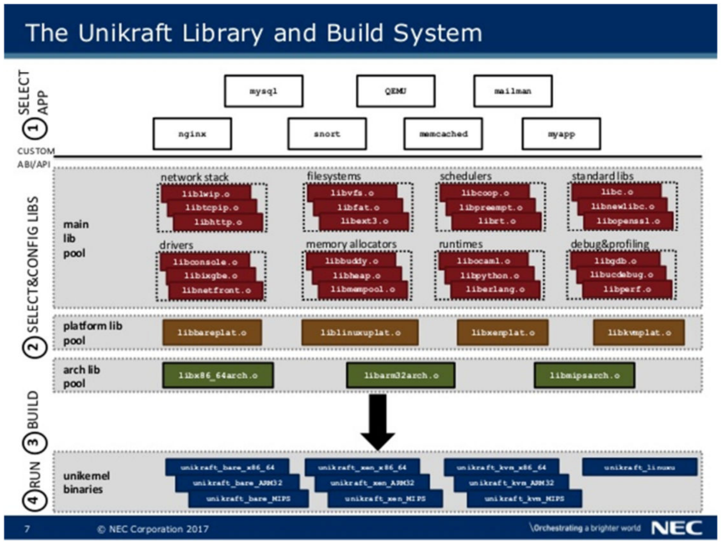
Multiple Unikernels/Languages

VM Images for multiple hypervisors and bare metal

Deploys unikernels for Kubernetes management

Another Possibility

- UniKraft
  - Similar To UniK
  - Being Developed By NEC
  - A Xen-Sponsored Project
  - Developing a Standardized Set of OS Libraries
  - Build and Deployment Tool



The Unikraft Library and Build System

① SELECT APP

mysql
QEMU
mailman
nginx
snort
memcached
myapp

CUSTOM ABI/API

② SELECT&CONFIG LIBS

main lib pool

network stack: liblwip.o, libtcpip.o, libhttp.o
filesystems: libvfs.o, libfat.o, libext3.o
schedulers: libcoop.o, libpreempt.o, librt.o
standard libs: libc.o, libnewlibc.o, libopenssl.o
drivers: libconsole.o, libixgbe.o, libnetfront.o
memory allocators: libbuddy.o, libheap.o, libmempool.o
runtimes: libocaml.o, libpython.o, liberlang.o
debug&profiling: libgdb.o, libucdebug.o, libperf.o

platform lib pool: libbareplat.o, liblinuxplat.o, libxenplat.o, libkvmplat.o

arch lib pool: libx86_64arch.o, libarm32arch.o, libmipsarch.o

③ BUILD

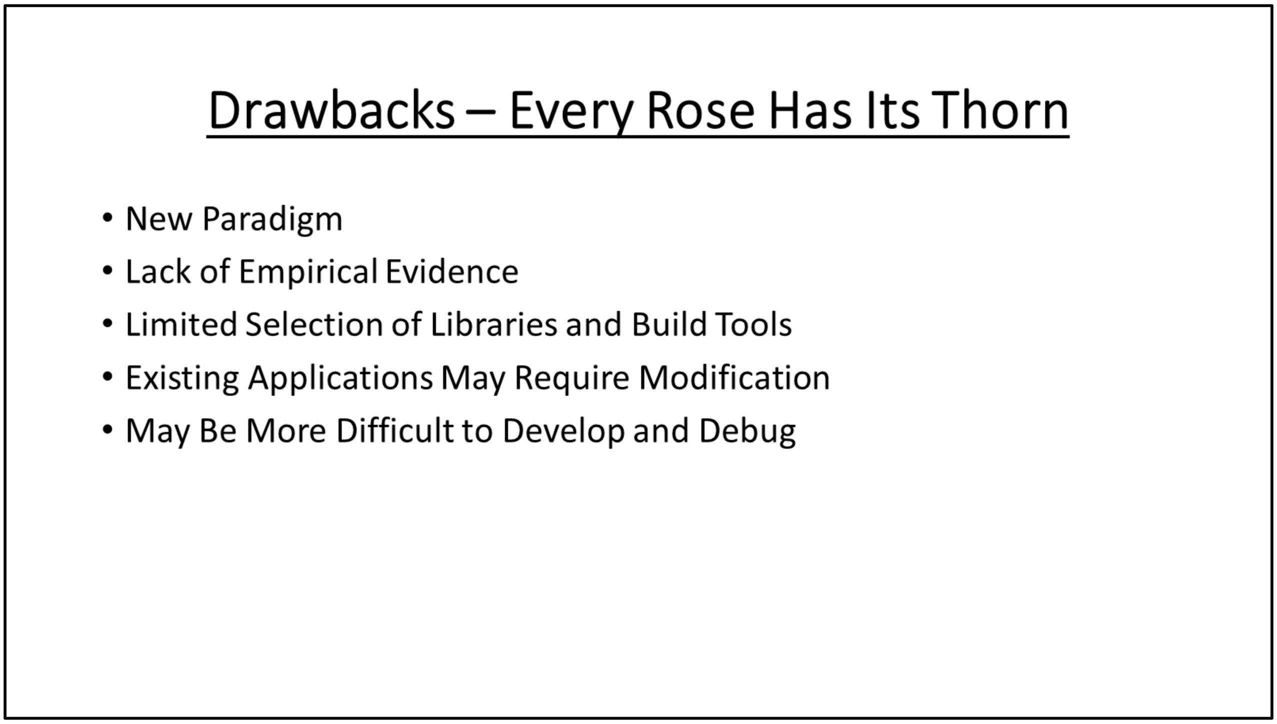
④ RUN

unikernel binaries: unikraft_bare_x86_64, unikraft_bare_ARM32, unikraft_bare_MIPS, unikraft_xen_x86_64, unikraft_xen_ARM32, unikraft_xen_MIPS, unikraft_kvm_x86_64, unikraft_kvm_ARM32, unikraft_kvm_MIPS, unikraft_linuxu

7 © NEC Corporation 2017 \Orchestrating a brighter world NEC

On the Horizon – UniKraft

A framework to collect existing unikernel “libraries”
A “build tool” to build new unikernels
“CPAN” or “NPM” for unikernels



Drawbacks – Every Rose Has Its Thorn

- New Paradigm
- Lack of Empirical Evidence
- Limited Selection of Libraries and Build Tools
- Existing Applications May Require Modification
- May Be More Difficult to Develop and Debug

Drawbacks?

Hardware or hypervisor specific drivers
Existing applications may not run correctly in a shared memory model

DEMO TIME

- Rumprun Build Tool
- NetBSD Library
- C “Hello World” Program
- QEMU Virtual Machine

cloudera

Demo (Rumprun)

Here What Everybody’s Been Waiting For!!!

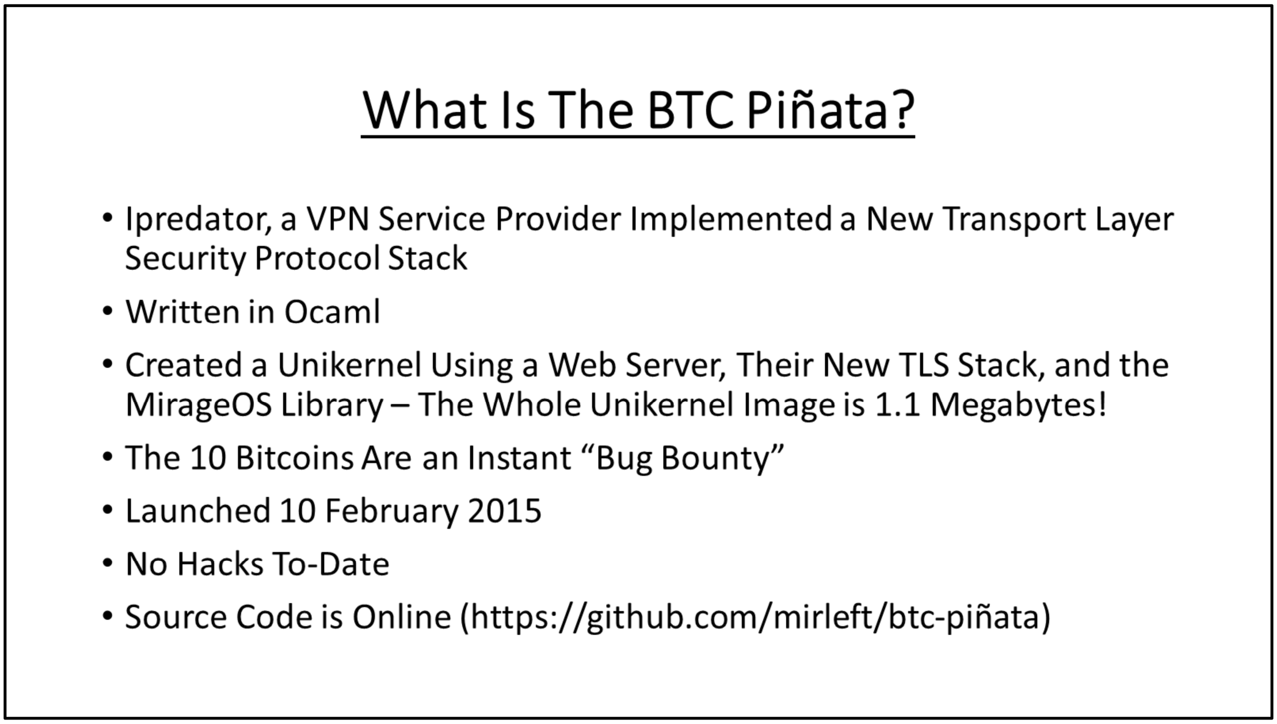
Bitcoin Giveaway Time!!!!!!

Do You Have Your Raffle Ticket?

“BTC Piñata”

- This Application Holds 10 Bitcoins ($82,019.80USD, 3-15-18 16:35)
- Uses TLS Mutual Authentication
- Public Key Certificate is Published
- Guess the Secret Key and the Bitcoins Are Yours
  - Secret Key is RSA 4096 – Billions of Years to Guess!!
- Have To Hack The BTC Piñata Server Instead
- http://ownme.ipredator.se

“If You Smash It, You Get To Keep the Pieces”



What Is The BTC Piñata?

- Ipredator, a VPN Service Provider Implemented a New Transport Layer Security Protocol Stack
- Written in Ocaml
- Created a Unikernel Using a Web Server, Their New TLS Stack, and the MirageOS Library – The Whole Unikernel Image is 1.1 Megabytes!
- The 10 Bitcoins Are an Instant "Bug Bounty"
- Launched 10 February 2015
- No Hacks To-Date
- Source Code is Online (https://github.com/mirleft/btc-piñata)

BTC Piñata

PWN2OWN ~ $100K

IPredator

Prove out new TLS implementation

OCaml/MirageOS

1.1 Mb image size (available on GitHub)

BTC Piñata Online Demo

Copies of the slides and the talking points may be downloaded from the
Formularity website:

https://formularity.com

OK, at this point, hopefully I've demonstrated the security, performance, and resource savings of unikernels.   Given the security problems of current full operating system IOT devices, I truly believe that unikernels are the single most effective base for acceptable IOT device security. Thank you! Copies of these slides and my talking notes will be available on the Formularity website later today, as well as through the ITPC 2018 website. Are there any questions?...