Specialization and Extension: An Agentic Future

Dan Williams

Keynote for the 1st AgenticOS workshop with ASPLOS

My Perspective as a Systems Researcher

- IBM research ~10 years
- Virginia Tech ~5 years
- Interest/expertise is in Systems
  - Trusted computing
  - Virtualization
  - Kernel extensions

- Have always avoided AI/ML… until now?

Why Agents are Different

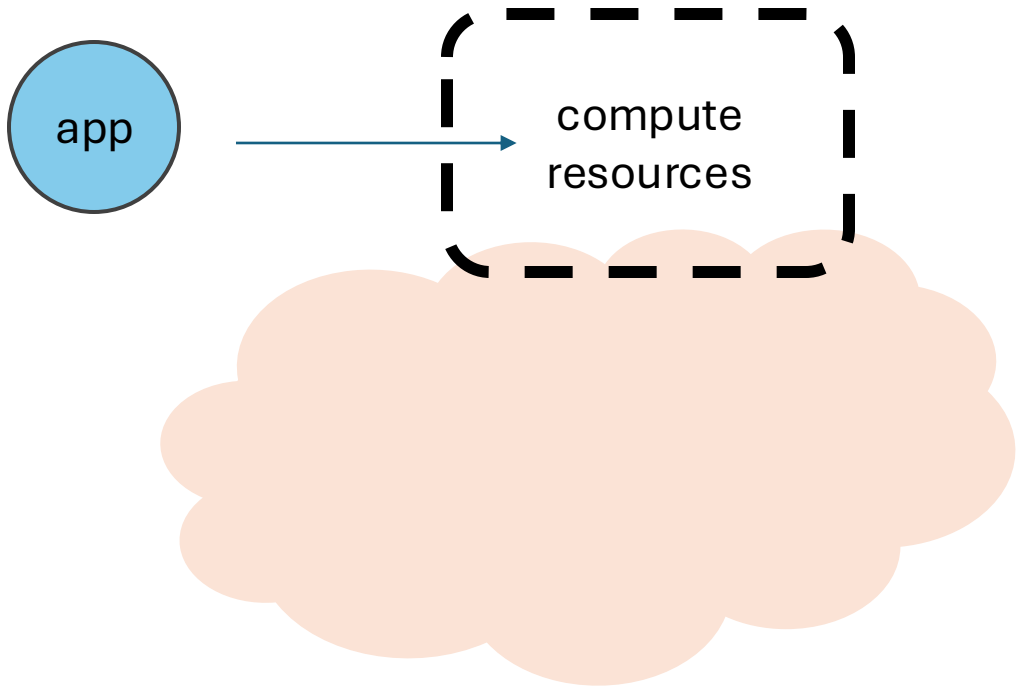
- Impacts of Agentic AI on systems problems that I'm familiar with

- PART 1: The rise and fall of the Unikernel
  - Specialization vs. generalization and how coding agents affect it
- PART 2: The rise of safe kernel extension
  - Safe extension and how agents magnify its importance
- PART 3: Specialized software explosion
  - Future looking challenges

PART 1:
The rise and fall of Unikernels

Context: cloud computing ~ 8-10 years ago

- Pay-as-you go compute resources
- No need for customers to own and manage physical servers

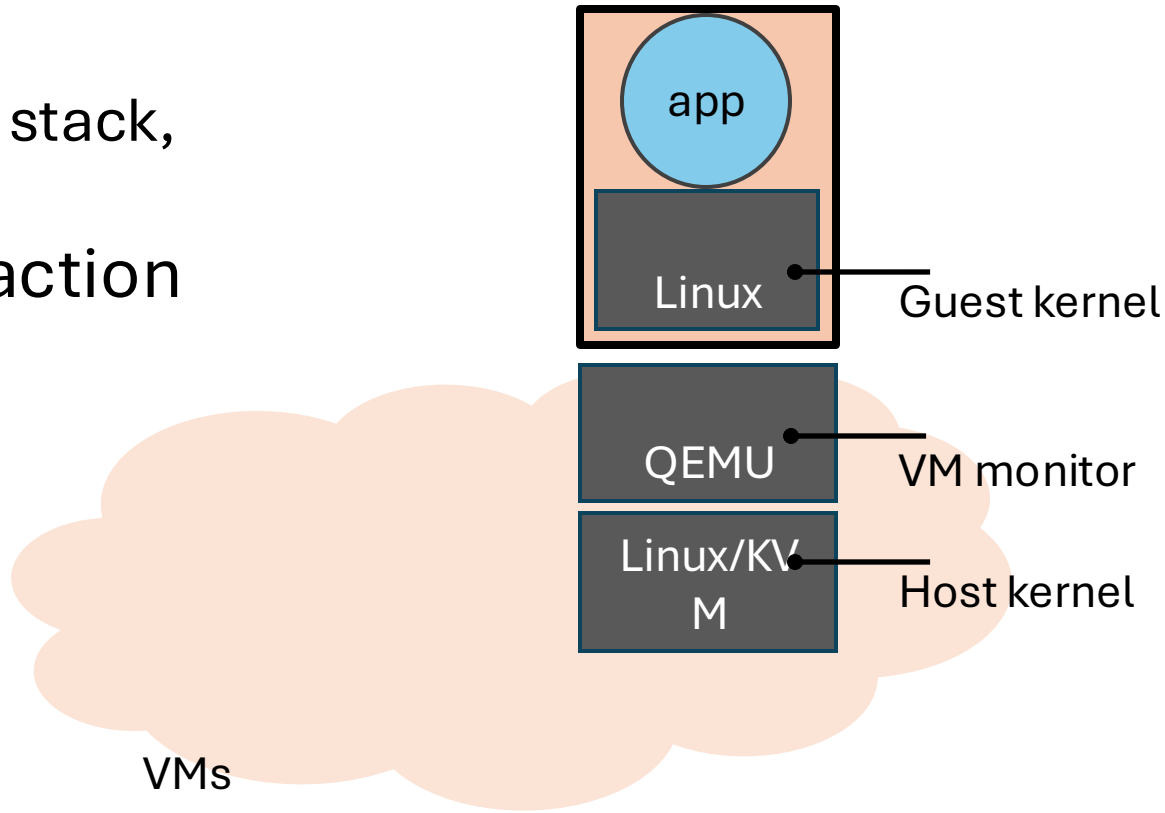


- How resources are exposed to cloud users: **unit of execution**

Unit of execution: the virtual machine (VM)

- Can run **general-purpose** workloads
- Guests/tenants well **isolated**
  - Each has its own filesystem, network stack, etc.
- Run on **thin** virtual hardware abstraction

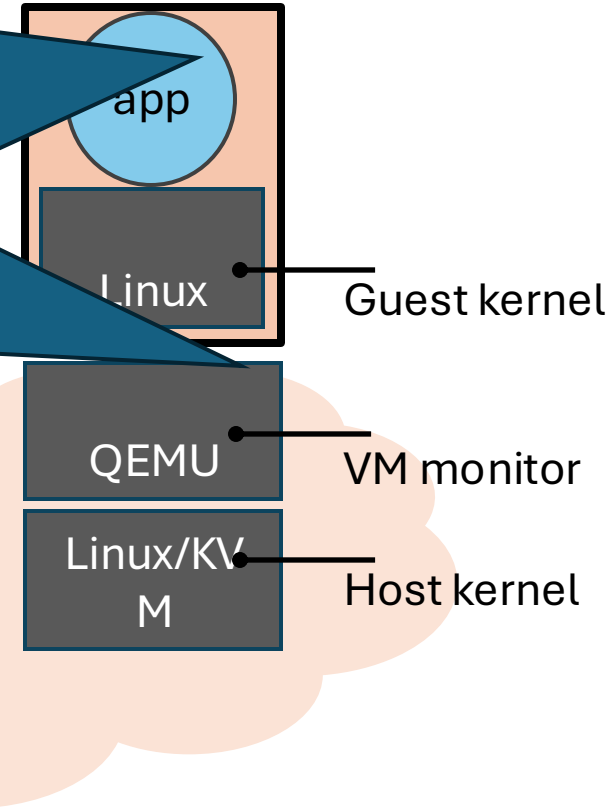
- But VMs are **heavyweight**
  - Image size
  - Startup time
  - Runtime overheads
  - Memory density



Unit of execution: the virtual machine (VM)

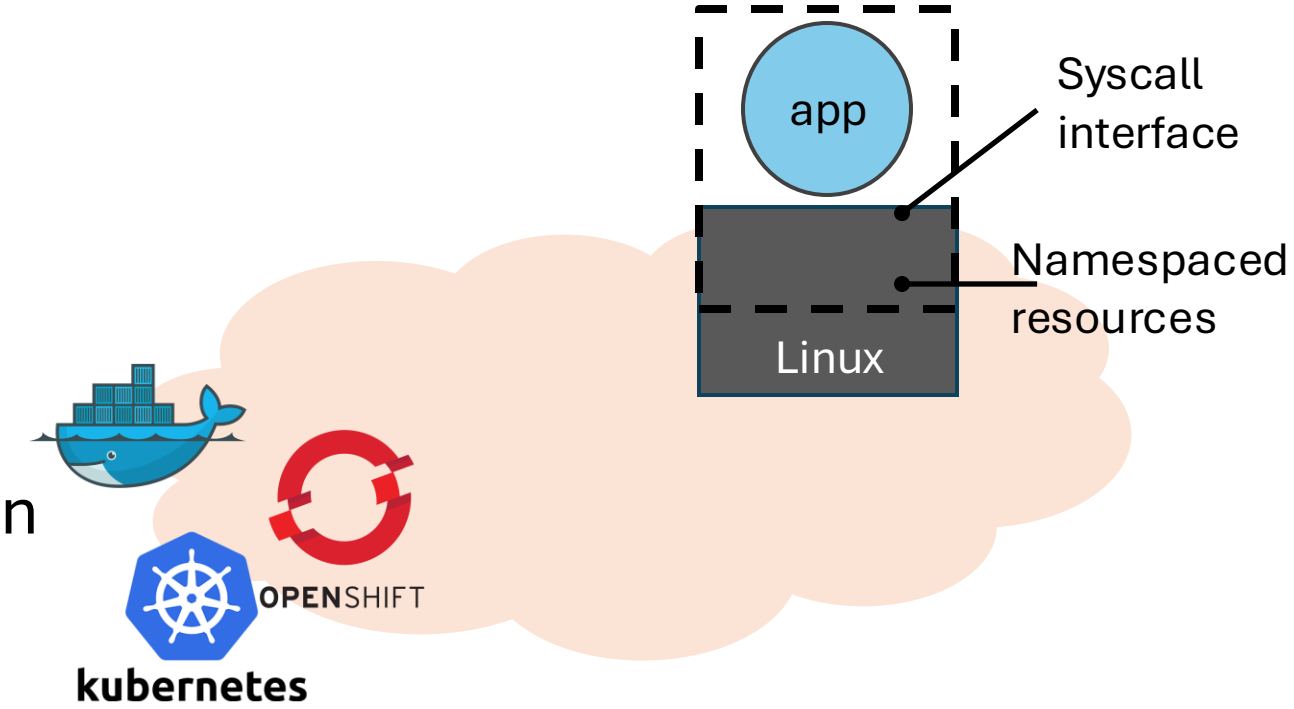
- Can run **ge**
- Guests/**tena**
  - etc.
- Run on **th**
- But VMs are **he**
  - Image size
  - Startup time
  - Runtime overheads
  - Memory density

OK for coarse (hourly) resource consumption, but what about fine grained (ms) serverless

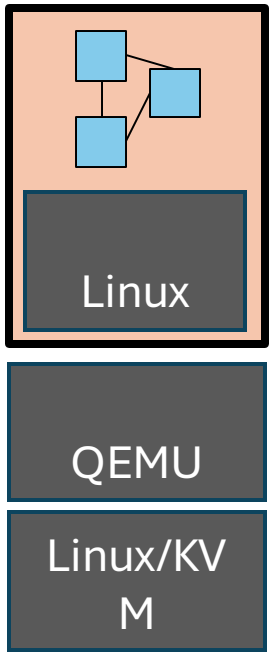


Unit of execution: the container

- Can run **general-purpose** workloads
- Guests/tenants **appear isolated**
  - Namespaced views of filesystem, network stack, etc.
- Containers are **lightweight**
  - Image size
  - Startup time
  - Runtime overheads
  - Memory density
- But run on **wide** POSIX abstraction
  - Poor isolation

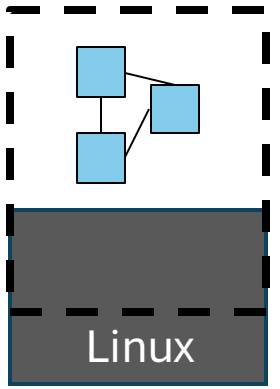


In search of a better unit of execution…



VMs

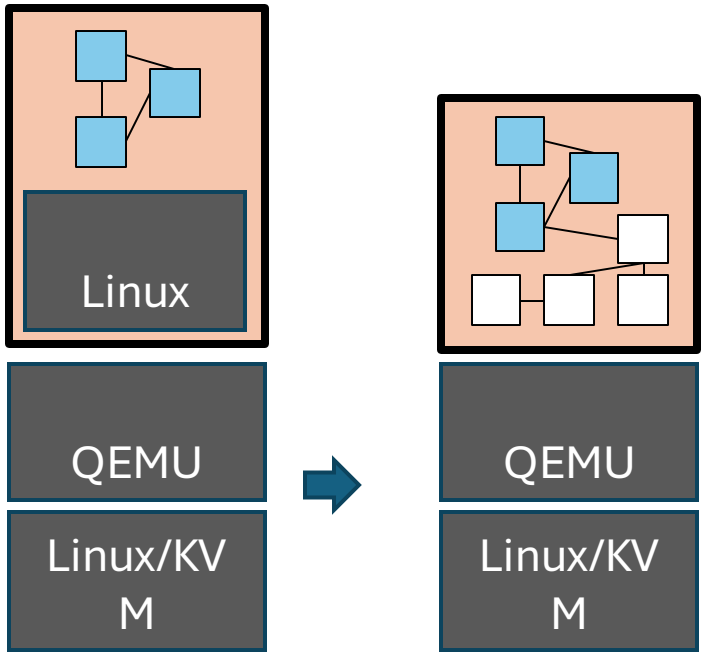
well-isolated heavyweight



Containers

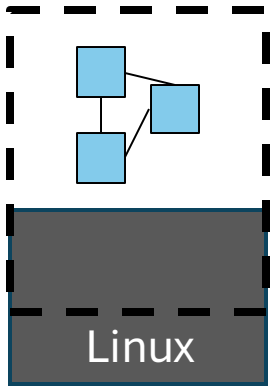
lightweight poorly isolated

In search of a better unit of execution…



VMs

Unikernels



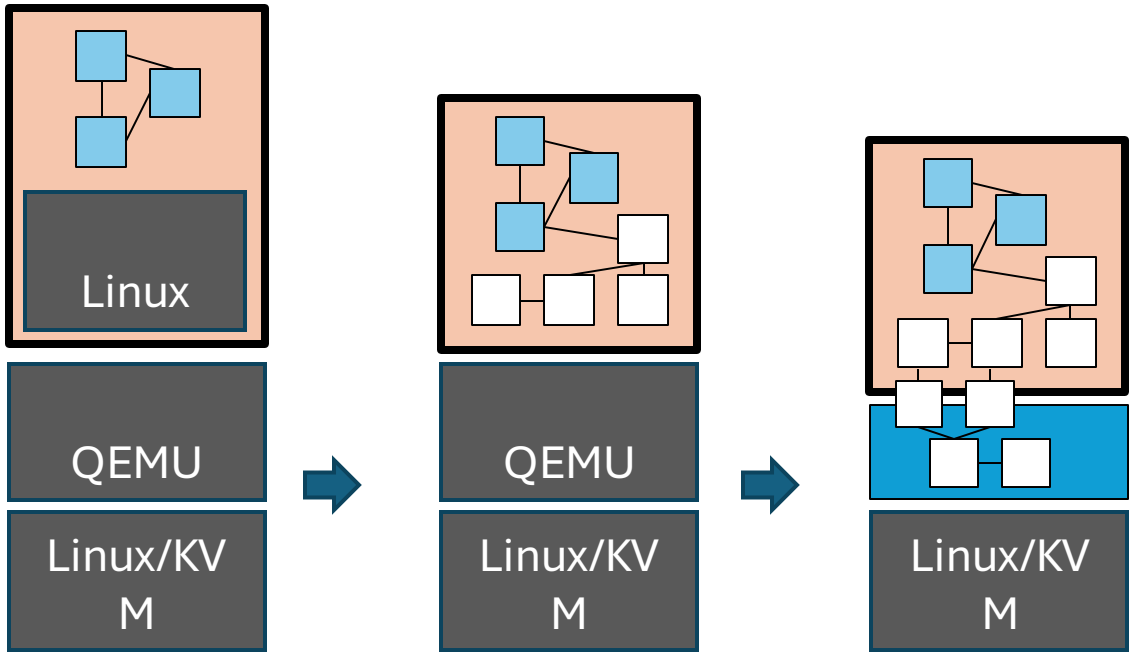
Containers

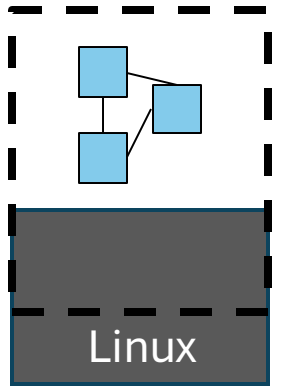
well-isolated heavyweight

well-isolated lightweight

lightweight poorly isolated

In search of a better unit of execution…

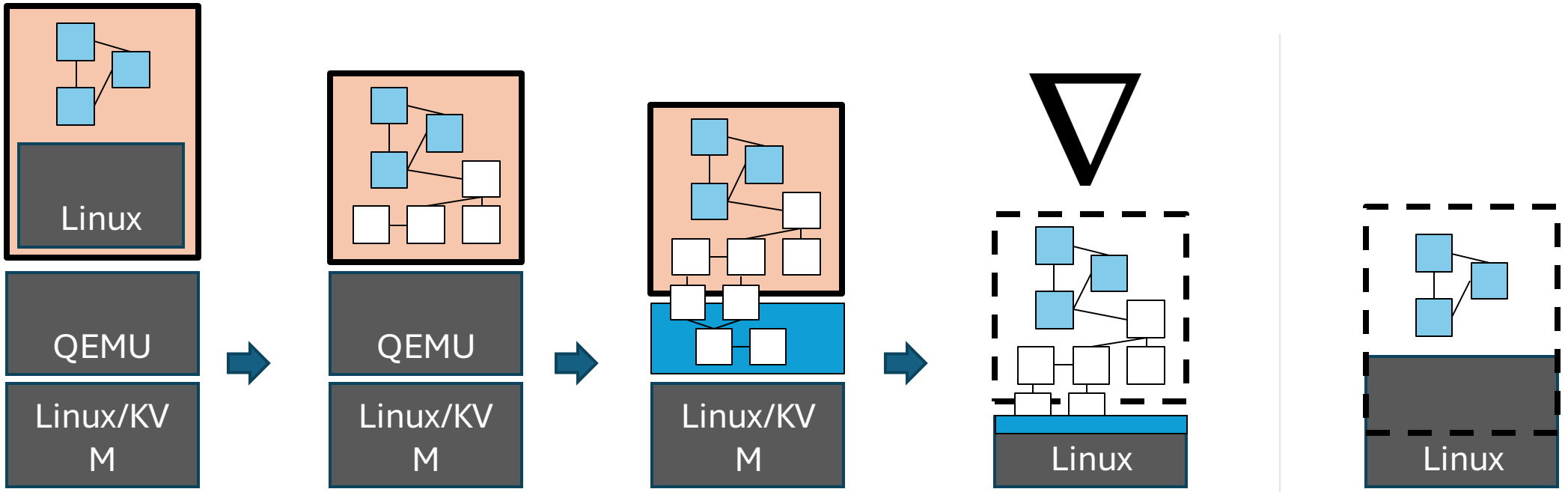




VMs | Unikernels | **Unikernels with unikernel monitors** | Containers

well-isolated heavyweight → **well-isolated lightweight** | **lightweight poorly isolated**

In search of a better unit of execution…



VMs | Unikernels | **Unikernels with unikernel monitors** | **Unikernels as processes** | Containers

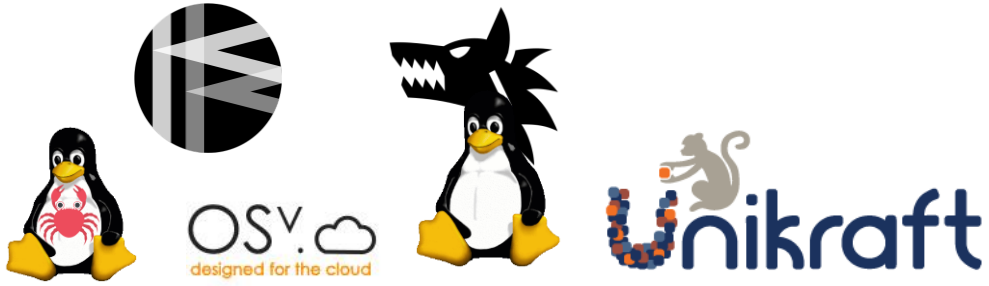
well-isolated heavyweight → **well-isolated lightweight** | **lightweight poorly isolated**

Unikernels and Compatibility

- Language-specific
  - MirageOS (OCaml)
  - IncludeOS (C++)

- Legacy-oriented
  - Rumprun (NetBSD-based)
  - **Lupine Linux**
  - Hermitux
  - OSv
  - Unikraft

Claim binary compatibility with Linux



Issues with "general" unikernels

- Unikernel benefits erode with generality
  - E.g., Static analysis/ whole-system optimization
- Lack full Linux support
  - Missing system calls, non-PIE incompatibilities, TLS incompatibilities, fork, /proc, signals, etc.
- Curated applications
  - Out of date?

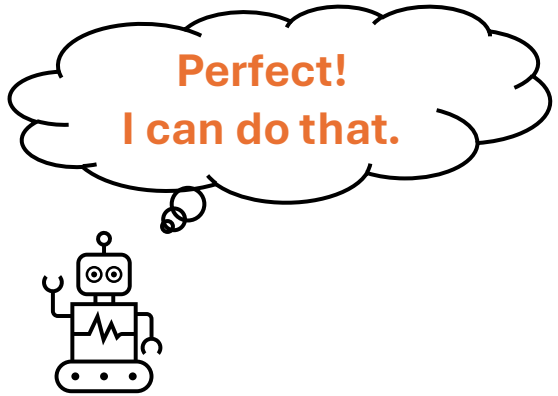
Specialized code has been costly to develop and/or maintain

- Requires language enthusiast community (e.g., Mirage OCaml)
- Or maintainer critical mass (e.g., Linux)

Specialization is ***practically*** infeasible

Specialization is ***practically*** infeasible

Unless...



~~Cambrian~~ Specialized Unikernel Explosion

- Coding agents could cause an explosion of new implementations

- Challenges:
  - Is it reasonable to rewrite/specialize very large, complex software?
  - Is the code any good? Is it safe?

~~Cambrian~~ Specialized Unikernel Explosion

- Coding agents could cause an explosion of new implementations

- Challenges:
  - Is it reasonable to rewrite/specialize very large, complex software?
  - Is the code any good? Is it safe?

Stay tuned for extensions!

PART 2:
The rise of safe kernel extensions

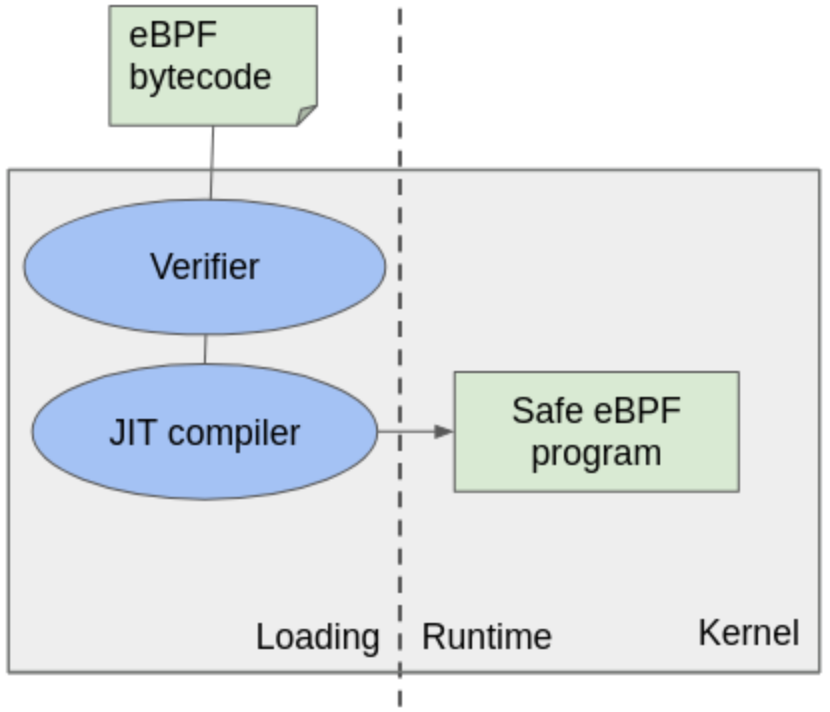
The kernel extension ecosystem

- Emerging use cases for custom code in the kernel
  - Networking: packet processing or filtering
  - Observability: collect metrics
  - Tracing/Profiling: follow execution to debug problems
  - Security: gather context for policies
  - Application acceleration
- Kernel module? Too risky
- Userspace? Too slow

- Need a way to ***safely*** run third-party code in the kernel…

BPF: verified safe kernel extensions

- User-supplied programs that run at **hook points** in kernel
  - Programs specified in eBPF bytecode that is amenable to verification
- Verified by **in-kernel verifier**
  - Form of symbolic execution to check for safety (e.g., no NULL dereference, resource safety, termination, etc.)
- In-kernel **JIT compiler** converts to native binary



Issues to solve

- Safety
  - Unsafe nesting
  - Interactions with kernel environment
  - Long-running extensions
- Performance
  - Kernel hooks
  - Pairwise extension patterns
  - Selective extension

BPFflow - Preventing information leaks from eBPF

Enabling BPF Runtime policies for better BPF management

Rahul Tiwari
rahult@vt.edu
Virginia Tech
Blacksburg, VA, USA

Raj Sahu
Virginia Tech
Blacksburg, VA, USA
raj.sahu@vt.edu

Dan Williams
Virginia Tech
Blacksburg, VA, USA
djwillia@vt.edu

mezone

Siddharth Chintamaneni
Virginia Tech
Blacksburg, VA, USA
sidchintamaneni@vt.edu

Sai Roop Somaraju
Virginia Tech
Blacksburg, VA, USA

Dan Williams
Virginia Tech
Blacksburg, VA, USA

Kernel extension verification is untenable

Overflowing the ke

Jinghao Jia
University of Illinois
Urbana-Champaign, IL, USA
jinghao7@illinois.edu

Raj Sahu
Virginia Tech
Blacksburg, VA, USA
rjsu26@vt.edu

Adam Oswald
Virginia Tech
Blacksburg, VA, USA
adamoswald@vt.edu

Dan Williams
Virginia Tech
Blacksburg, VA, USA
djwillia@vt.edu

Michael V. Le
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA
mvle@us.ibm.com

Tianyin Xu
University of Illinois
Urbana-Champaign, IL, USA
tyxu@illinois.edu

ARTIFACT EVALUATED usenix AVAILABLE

ARTIFACT EVALUATED usenix FUNCTIONAL

Fast (Trapless) Kernel Probes Everywhere

Jinghao Jia†, Mich[illegible]

†University of Illinois

Pairwise BPF Programs Should Be Optimized Together

Dan Williams
Virginia Tech
Blacksburg, VA, USA
djwillia@vt.edu

Eliminating eBPF Tracing Overhead on Untraced Processes

Milo Craun
Virginia Tech
Blacksburg, VA, USA
miloc@vt.edu

Khizar Hussain
Virginia Tech
Blacksburg, VA, USA
khizar@vt.edu

Uddhav Gautam
Virginia Tech
Blacksburg, VA, USA
upgautam@vt.edu

Zhengjie Ji
Virginia Tech
Blacksburg, VA, USA
zhengjie@vt.edu

Tanuj Rao
Virginia Tech
Blacksburg, VA, USA
tansanrao@vt.edu

Dan Williams
Virginia Tech
Blacksburg, VA, USA
djwillia@vt.edu

Safe extensions beyond kernel BPF

- Bpftime enables efficient and safe **userspace** extension
- Rex enables similar safety guarantees via Rust and a kernel runtime instead of BPF
- Application-specific kernel extensions (e.g., BMC memcached cache)

Extending Applications Safely and Efficiently

Yusheng Zheng, *UC Santa Cruz*
Tong Yu, *eunomia-bpf Community*
Yiwei Yang, *UC Santa Cruz*
Yanpeng Hu, *ShanghaiTech University*
Xiaozheng Lai, *South China University of Technology*
Dan Williams, *Virginia Tech*
Andi Quinn, *UC Santa Cruz*

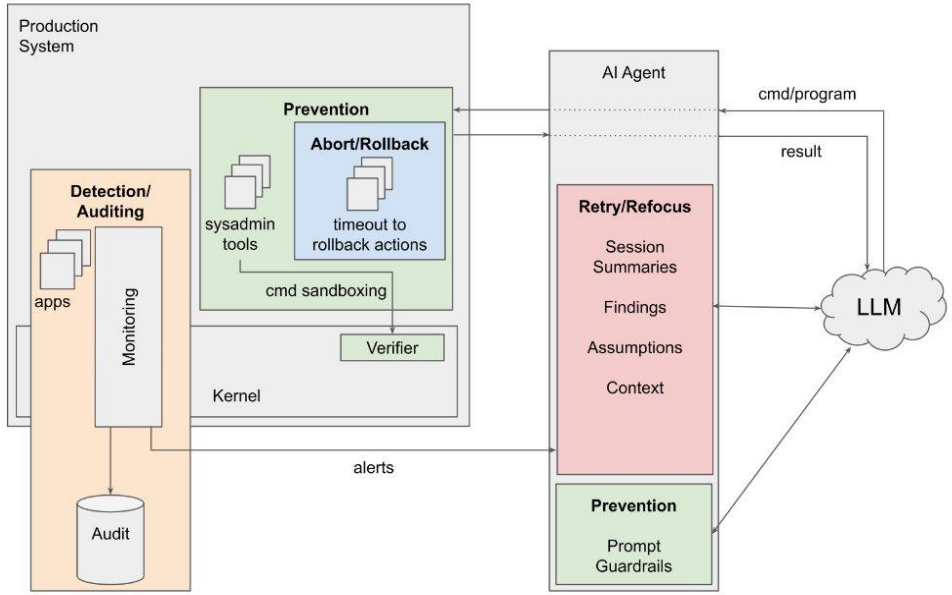
Rex: Closing the language-verifier gap with safe and usable kernel extensions

Jinghao Jia*, Ruowen Qin*, Milo Craun†, Egor Lukiyanov†, Ayush Bansal*, Minh Phan*
Michael V. Le‡, Hubertus Franke‡, Hani Jamjoom‡, Tianyin Xu*, Dan Williams†
*University of Illinois Urbana-Champaign †Virginia Tech ‡IBM T.J. Watson Research Center

Safe extension of software is becoming **practical** and **general**

Already necessary for some Agentic Tasks

- Performance debugging agent
- Require privilege on system
- Verified extensions (like BPF) **prevent** some undesirable outcomes
- One part of a larger system to support privileged performance debugging agents



Specialization is coming

- Recall: **specialization** carries benefits
- **Extension** provides a practical means to specialize complex software
- **Safe** extension rules out some code issues

~~Cambrian~~ Specialized Unikernel Explosion

- Coding agents could cause an explosion of new implementations
- Challenges:
  - Is it reasonable to rewrite/specialize very large, complex software?
  - Is the code any good? Is it safe?

PART 3:
The Specialized Software Explosion

Specialized Extension Explosion

- Coding agents could cause an explosion of new specialized implementations *and/or extensions that specialize existing software*

- Further challenges:
  - When to extend vs. rewrite?
  - How to manage ecosystem of extensions?
  - How to monitor/debug deployments with heterogeneous implementations?
    - Cross-implementation logging

Cross-implementation Logging

- QUIC transport layer protocol is implemented in userspace
  - Fast innovation cycle
  - Lots of implementation heterogeneity
- qlog attempts shared structured logging, but implementations do not use consistently
- Given the QUIC protocol, can a coding agent provide consistent, cross-implementation logging?
  - QUIC specification is key

MsQuic	MIT License	C	designed to be a general purpose QUIC library. Windows and cross platform by .NET. Rust and C layers available are available, as well as conver wrapper classes.
QUIC Library (mvfst)	MIT License	C++	mvfst (Pronounced move fast) is a client and se implementation of IETF QUIC protocol in C++ b
LiteSpeed QUIC Library (lsquic)	MIT License	C	This is the QUIC and HTTP/3 implementation us LiteSpeed Web Server and OpenLiteSpeed.
ngtcp2	MIT License	C	This is a QUIC library that's crypto library agnos works with OpenSSL or GnuTLS. For HTTP/3, it r separate library like nghttp3.
Quiche	BSD-2-Clause License	Rust	Socket-agnostic and exposes a C API for use in applications.
quicly	MIT License	C	This library is the QUIC implementation for the server.
quic-go	MIT License	Go	This library provides QUIC support for Go.
Quinn	Apache License 2.0 MIT License	Rust	An async-friendly QUIC implementation in Rust
Neqo	Apache License 2.0 MIT License	Rust	This implementation from Mozilla is planned to integrated in Necko, a network library used in t web browser
aioquic	BSD-3-Clause License	Python	This library features an I/O-free API suitable for in both clients and servers.
picoquic	MIT License	C	A minimal implementation of QUIC aligned with specifications
pquic	MIT License	C	An extensible QUIC implementation that include virtual machine that is able to dynamically load as plugins
quic	BSD-3-Clause License	Haskell	This package implements QUIC based on Haske lightweight threads.
netty-incubator-codec-quic	Apache License 2.0	Java	This package implements QUIC in netty based Quiche implementation.
nodejs-quic	MIT License	NodeJs	This experimental package implements QUIC fo
s2n-quic	Apache License 2.0	Rust	Open-source Rust implementation from Amazon Services

Other Discussion points

- Evolving software towards extensibility vs. rewrite
  - General extensibility vs. "parametric customization interfaces" [1]
  - Where should "safety" be enforced?
- Homogeneity vs. Heterogeneity
  - Is agent-derived code homogeneous or heterogeneous?
  - Merits of software diversity (e.g., ASLR)
- Importance of specification
  - Grounding across implementations
  - Opportunities for verification
- Death of abstraction layers?
  - A collapse of the ossified software stack?

[1]: Druschel et al. Extensible Kernels are Leading OS Research Astray. HotOS '97

Summary

- Coding agents may enable practical and widespread software specialization
- Safe extension provides a path forward to mitigate complexity and safety challenges
- How to prepare for the specialized software explosion?