HYPERVISOR-LESS VIRTIO FOR REAL-TIME AND SAFETY

Maarten Koning
Wind River Fellow

WNDRVR

KVM Forum 2021

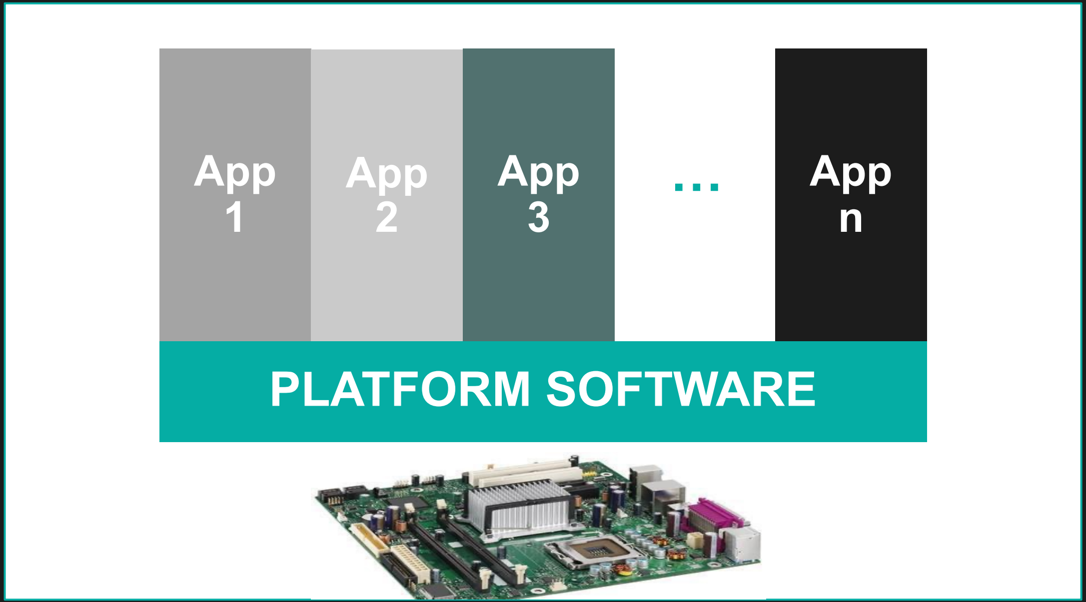
FIXED-FUNCTION EMBEDDED SYSTEMS ARE NOW INTELLIGENT EDGE DEVICES

EMBEDDED SYSTEMS
Black Box Dedicated Systems

Enclosed and Engineered

INTEGRATION PLATFORMS
White Box Partitioned Systems

Open and Orchestrated



Open Architecture / Open Ecosystem

Multi-party Software Integration

WNDRVR

EDGE DEVICES WILL INCREASINGLY CONTAIN LINUX

1. Edge devices have large amounts of open-source middleware & ready-made applications that are increasingly only available for Linux.
2. Board support packages for COTS & reference hw are increasingly only available for Linux.
3. Porting code from Linux is increasingly problematic.

→ **Therefore, edge devices will increasingly contain an instance of Linux.**

QED //

THE INTELLIGENT EDGE REQUIRES REACTIVITY

Emerging Use Cases Demand Low Latency and Accelerated Processing at the Edge.

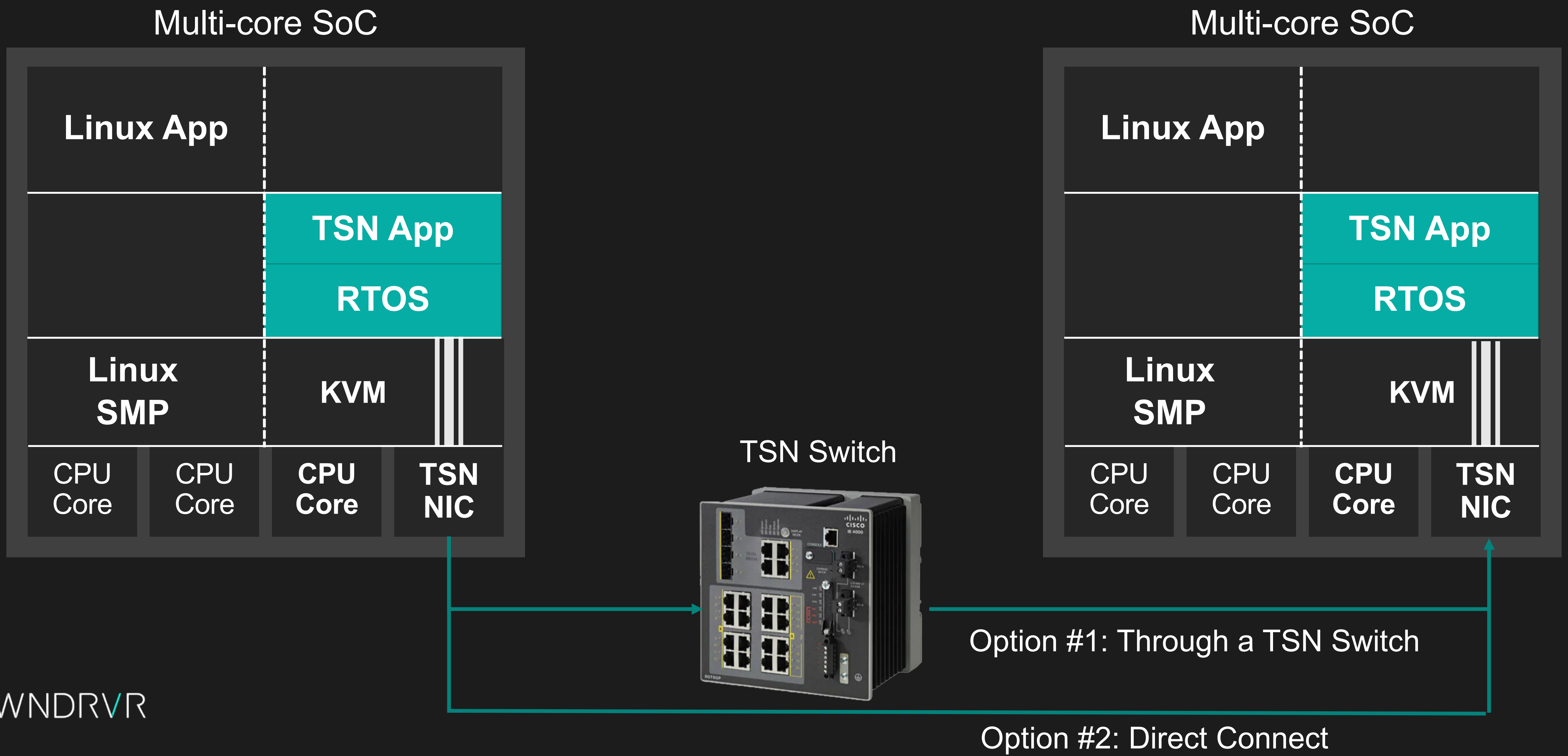
Edge Infrastructure:	**Wireless** vRAN, vEPC	**Wireline** PON	**uCPE** SD-WAN	**IP Enterprise Services**
Autonomous Devices:	**Drones**	**Autonomous Vehicles**	**Industry Robots**	**Medical**
Immersive Experiences:	**Virtual Reality**	**Augmented Reality**	**360 Video**	**Wearable Cognitive Assistance**
IoT & Analytic:s	**Industrial Sensors**	**Home Devices**	**Retail**	**Healthcare**

WNDRVR

IF EDGE DEVICES WILL CONTAIN LINUX, WHERE WILL THE REAL-TIME AND SAFETY WORKLOADS RUN?

1. On Linux when it has sufficient reactivity
   → "software-based partitioning" (Linux-only approach)
2. Beside Linux in a virtual machine
   → "virtualization-based partitioning" (Hypervisor needed)
3. Beside Linux on a compute island
   → "physical partitioning" (Hypervisor-less)
4. Beside Linux on a borrowed core
   → "whiteboard partitioning" (Be careful!)

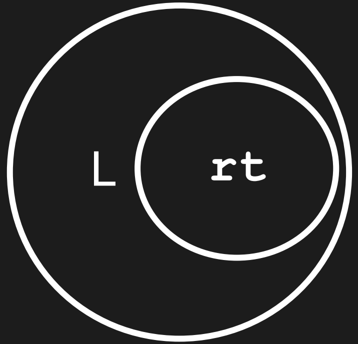
USE CASE: KVM WITH CORE ISOLATION FOR TIME SENSITIVE NETWORKING



WNDRVR

REALTIME AND SAFETY WORKLOADS WITH LINUX *

1. Core Reservation (Software partitioning)



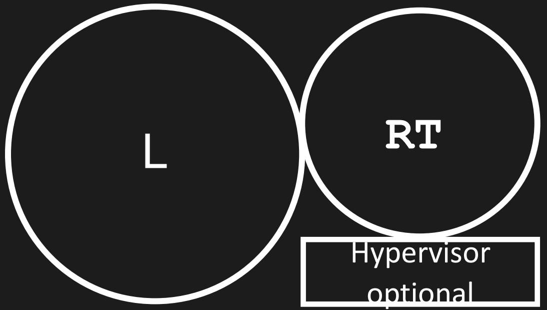
1a: User-level process

1b: unikernel

1c: KVM + rt workload

(PREEMPT_RT helpful)

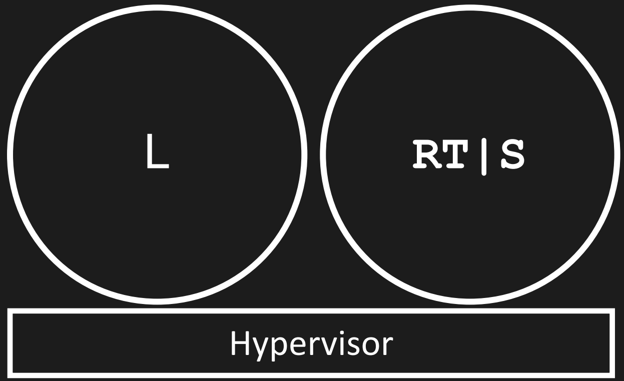
2. Core Offload (Whiteboard partitioning)



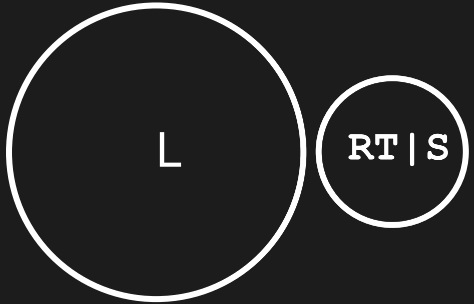
2a: Unsupervised AMP

2b: Partially-supervised AMP

3. Mixed-Criticality (Virtual Partitioning)



4. Compute Islands (Physical Partitioning)




```
     L: Linux general-purpose workload
    rt: soft realtime workload
    RT: hard realtime workload
     S: safety workload
 Green: standard practice
Orange: Less commonly seen
        (OS research / new / future)
```

* Assuming Linux cannot yet achieve certification to run safety workloads.

WNDRVR

REALTIME & SAFETY WORKLOADS WITH LINUX

10s of uSec-ish soft realtime required:

→ **deploy native workload as a Linux* process thread on a reserved core(s).**
→ *research: deploy realtime workload using Linux* KVM vCPU on a reserved core(s).*

uSec-ish hard realtime required:

→ **deploy RT workload beside Linux on a compute island <u>or</u> in a VM with a RT hypervisor.**
→ *research: deploy realtime workload beside Linux on a core(s) offloaded from Linux.*

Safety required:

→ deploy safety workload on a compute island <u>or</u> in a VM with a safety hypervisor.
→ *research: deploy safety workload on Safety Linux on a reserved core(s).*

(*) Linux PREEMPT_RT patch required, else expect
100s of uSec-ish for tuned Linux (PREEMPT_VOLUNTARY) or
mSec-ish for untuned (PREEMPT_NONE) Linux.

WNDRVR

BUT HOW TO SHARE RESOURCES BETWEEN RUNTIMES?

When workloads run on different runtimes in the same SoC, we need the runtimes to integrate for the purposes of:

1. printf(), console and debug access
2. read/write of Linux file systems from auxiliary runtimes
3. intra-SoC messaging between Linux and auxiliary runtimes

The “de facto” approach is to use TCP/IP for this over an on-chip or on-board ethernet switch - or via a virtual ethernet driver.

However, TCP/IP is a WAN protocol which is a heavyweight intra-SoC solution for these local runtime integration needs.

WHY USE VIRTIO FOR INTRA-SoC WORKLOAD INTEGRATION?

1. virtio is already available both in Linux and in many runtimes
2. virtio is an open specification that is transport independent
3. virtio has AF_VSOCK which is similar to AF_INET
   → our experiments show it is 10x faster than TCP/IP over virtio
4. virtio can be run over shared memory without a hypervisor
   → so-called ***"hypervisor-less virtio"***
5. virtio has low-level devices and higher-level services too

HYPERVISOR-LESS VIRTIO

Define and prototype a framework for using virtio as a communication infrastructure, while removing the constraints usually associated with the presence of a hypervisor.

Hypervisor-less virtio PoC:

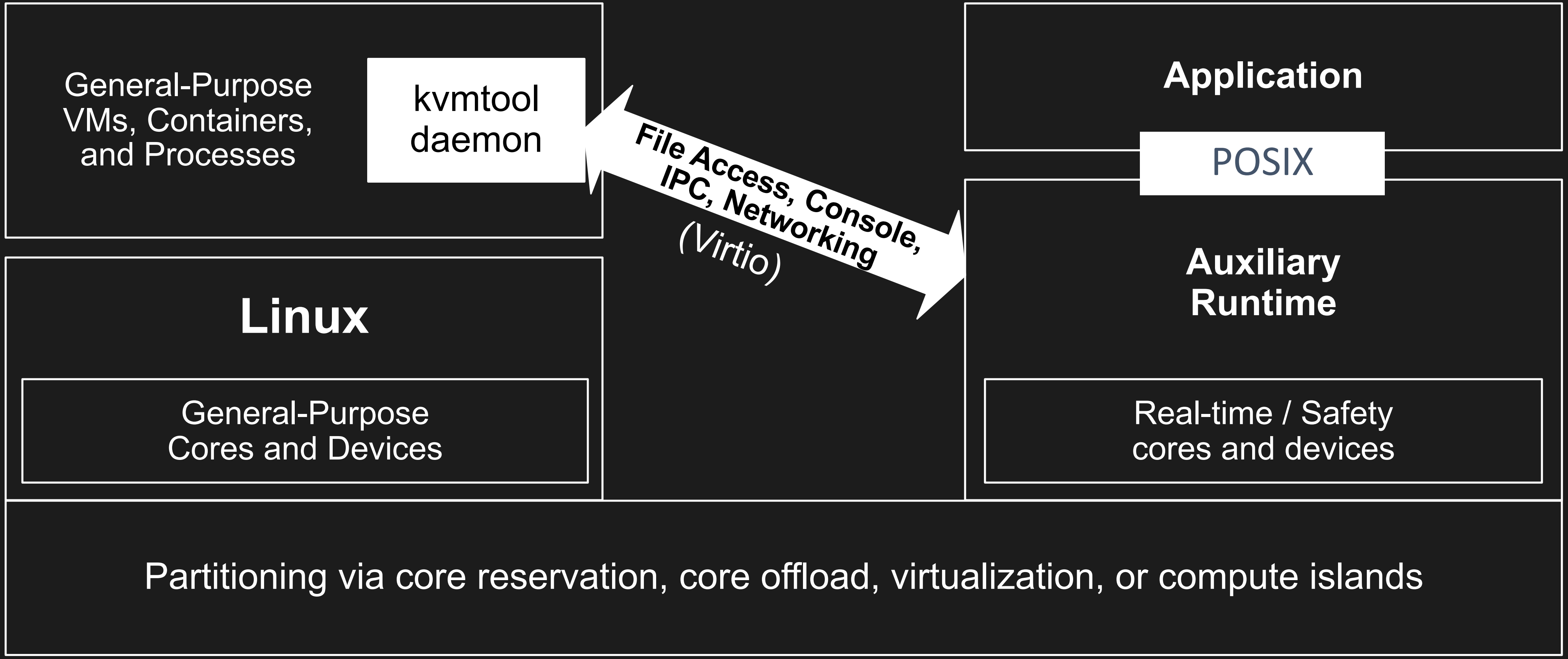
- 64-bit Intel x86_64 and ARM support
- Hardware notifications
- Selected Linux kvmtool AKA “lkvm” as the virtio back-end

  → leveraging its existing support for console, 9p file system, vsock and virtio-net.

  → added new MMIO over shared memory transport

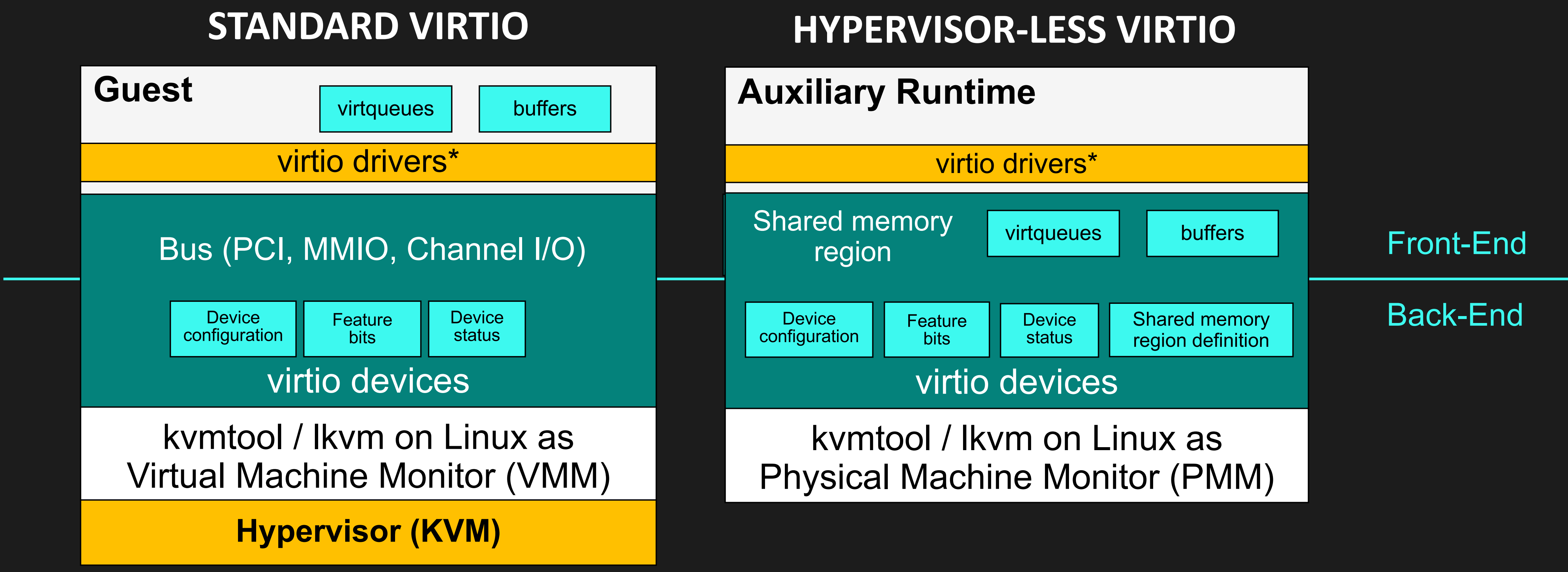
  → enabled /dev/{vhost-vsock,vhost-net} for vhost offload without workload virtualization.

GENERALIZED HYPERVISOR-LESS VIRTIO ARCHITECTURE



WNDRVR

SIMILARITIES / DIFFERENCES



* File system (9P), Console (serial), Network (virtual ethernet), IPC (vsock)

WNDRVR

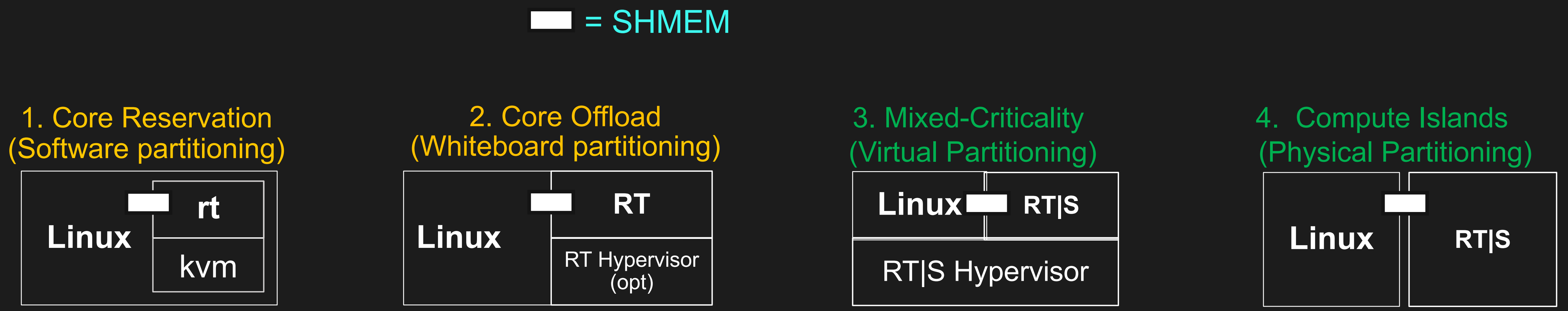
HYPERVISOR-LESS VIRTIO SHARED MEMORY LAYOUT

Shared Memory
Device <n> shared memory
Device <n> header
...
Device 0 shared memory
Device 0 header
DTB fragment

Per-device shared memory	
virtio console	24 KB
vsock	64 KB
9p	24 KB
virtio net	64 KB

WNDRVR

HYPERVISOR-LESS VIRTIO SCENARIOS



The Linux non-realtime / non-safety services are provided to apps on auxiliary runtimes via:

- `open()/close()/read()/write()/ioctl()/…` for serial and file system access
- `socket()/bind()/connect()/accept()/sendto()/recvfrom()/…` for IPC
- PoC development strategy:
  - step 1: enable printf() and file access from auxiliary runtimes using virtio
  - step 2: enable AF_INET socket family over virtio ethernet
  - step 3: switch to AF_VSOCK to remove IP stack requirement for auxiliary runtimes

HOW IT WORKS

1. In a hypervisor-less deployment, hardware mechanisms are used to signal device configuration and to send virtqueue notifications.

2. Upon receiving the hardware notification from the virtio front-end (i.e. the auxiliary runtime), Linux notifies the user-level PMM (kvmtool daemon) .

3. Upon being notified via an eventFd, the PMM determines the state of the virtio device using the device status field and its registry values and handles the request.

4. If the PMM can offload processing to vhost, it will act as a proxy between vhost services and the auxiliary runtime by capturing and relaying notifications.

VSOCK-ONLY HYPERVISOR-LESS VIRTIO

A TCP/IP port to vsock port PMM proxy (a la socat & ncat) between the host and the auxiliary runtime enables them to use vsock instead of TCP/IP.

→ yet still be reached using TCP/IP from Linux.

Example use cases:

- debug an auxiliary runtime with GDB using a vsock GDB server on the auxiliary runtime.
- access a shell on the auxiliary runtie using telnet or ssh with a vsock telnet/ssh daemon.
- enable auxiliary runtimes to leverage Linux file systems using vsock-based 9p or nfs clients.
- connect a vsock-based client/server on an auxiliary runtime to a Linux TCP/IP server/client

→ With this approach there may be less need to safety-certify an IP stack for a safety island if it is less expensive to certify virtio vsock.

WNDRVR

SIDE NOTE ON THE PERFORMANCE OF VIRTIO MMIO WITH MSIs WITH A HYPERVISOR

	TRAP (R)	TRAP (W)	CHECK IRQ (R)	ACK IRQ (W)	NOTIFY (W)	IRQ (host signal)	MSI (host signal)
virtio MMIO without MSIs	652633	652638	652615	652615	329666	660911	0
virtio MMIO with MSIs	20	66	0	0	591161	0	1.182M

- IRQ: 1.3M more traps, 1M more memory accesses → 600K fewer host signals
- MSI: 2x the number of host signals is due to 80%+ higher bandwidth

WNDRVR

SIDE NOTE ON THE PERFORMANCE OF VIRTIO/PCI VS VIRTIO/MMIO WITH A HYPERVISOR

Test	Virtio PCI	Virtio MMIO without MSI	Virtio MMIO with MSI
TCP_RR (host -> guest)	20182	11009	20352
TCP_RR (guest -> host)	20463	10955	20058

TCP_RR measures round trip latency (more trans/s = lower latency)

Host is a Walnut Canyon system with Ubuntu

Guest is Yocto Linux running via LKVM

→ Virtio MMIO + MSI (Message Signaled Interrupts) is as fast as virtio over PCI

WNDRVR

CONCLUSIONS

1. There are use cases for auxiliary runtimes at the edge; those runtimes need way to integrate with Linux ; and virtio can help - such as for console, network, file systems and IPC.
2. Compute islands can remove the need for virtualization to enable real-time or safety workloads with Linux-based systems – and they can still use virtio for multi-OS integration using hypervisor-less virtio.
3. Virtio over MMIO with MSIs is as fast as virtio over PCI (and has a smaller implementation making it potentially more suitable for safety cert).
4. AF_VSOCK sockets can be 10x faster than AF_INET TCP/IP sockets and also AF_VSOCK also has a much smaller implementation.

KVMTOOL LINKS

kvmtool was forked to enable its use as an hypervisor-less virtio back-end. It is on the OpenAMP GitHub since this work is being done as part of the OpenAMP Application Services Working Group activities:

https://github.com/OpenAMP/kvmtool

More info on OpenAMP activities is here:

https://www.openampproject.org/news/

MMIO MSI support for kvmtool is here:

https://github.com/OpenAMP/kvmtool/tree/mmio_msi https://vmsplice.net/~stefan/stefanha-kvm-forum-2015.pdf

redhat.

virtio-vsock

Zero-configuration host/guest communication

Stefan Hajnoczi <stefanha@redhat.com>

KVM Forum 2015

THANK YOU

WNDRVR Studio

THE PLATFORM FOR MISSION-CRITICAL INTELLIGENT SYSTEMS

Take a tour of the capabilities at:

www.windriver.com/studio/tour