



Open Kernel Labs™

Be open. Be safe.

Open-Source Virtualization: The Right Match for Embedded Linux

Gernot Heiser

Founder and CTO, Open Kernel Labs

Leader, Embedded OS Research, NICTA

Professor of Operating Systems, UNSW

LinuxWorld, August 2008

Outline

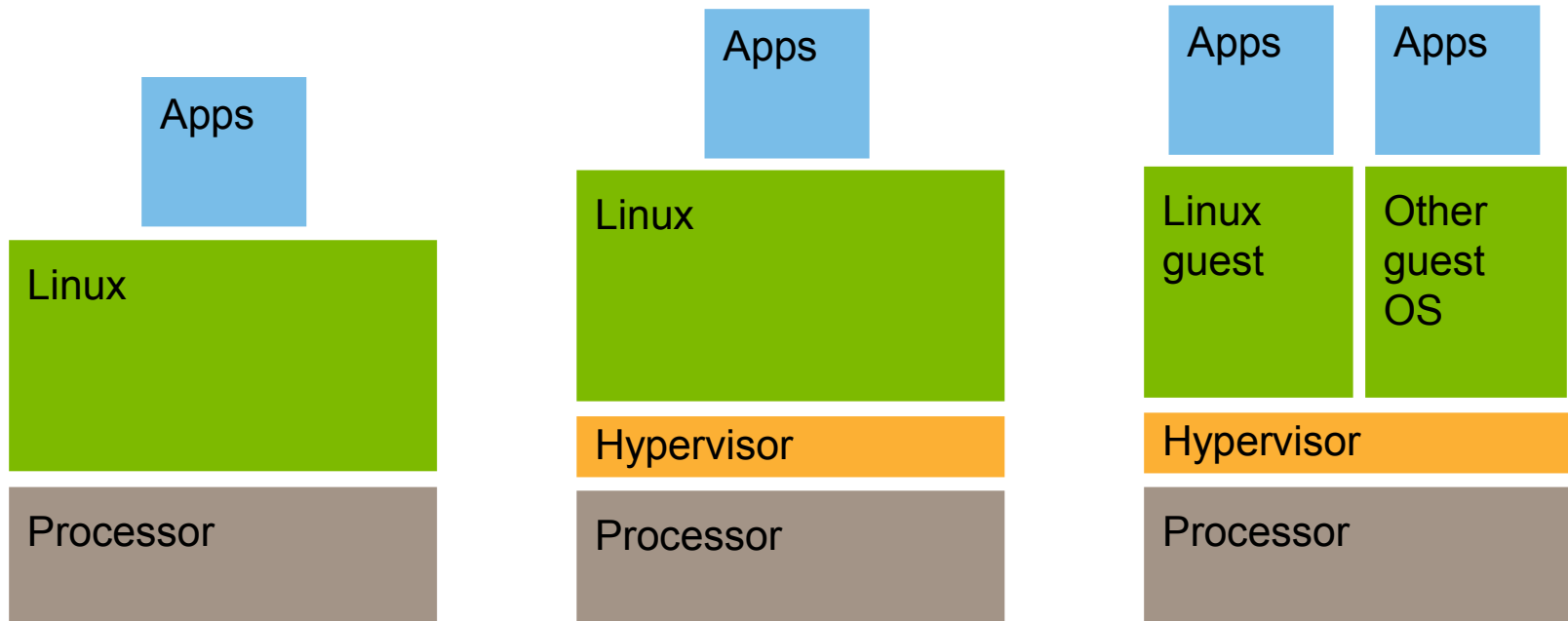


*Be open.
Be safe.*

- *What are virtual machines?*
- Why virtualize embedded Linux?
- How does it differ from enterprise virtualization?
- Embedded virtualization with OKL4

What are Virtual Machines?

- OS normally runs directly on hardware
- Virtualization inserts software layer between OS and hardware
 - called *hypervisor* or *virtual-machine monitor*
 - supports several concurrent OSes, called *guests*

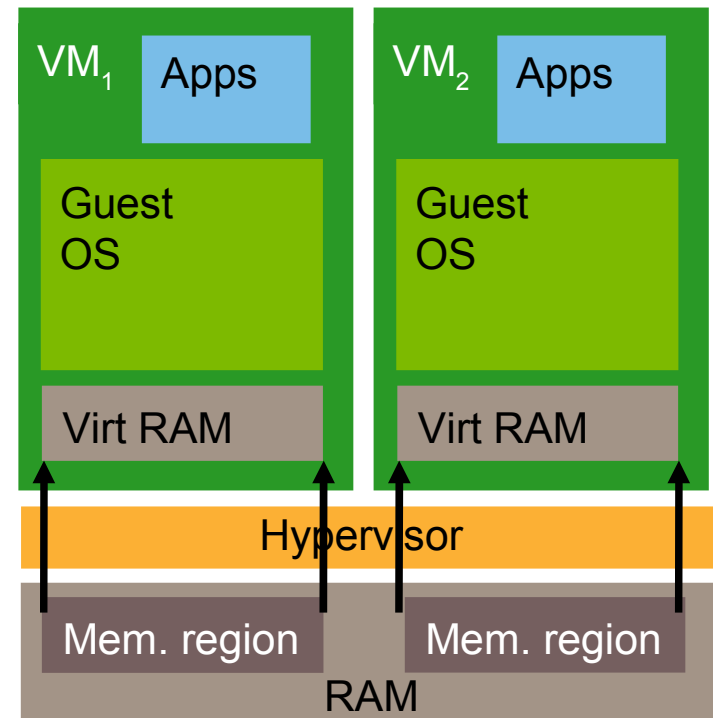


The Hypervisor



Be open.
Be safe.

- Hypervisor partitions and multiplexes hardware between guests
- Hypervisor is in complete control of all physical resources
 - memory, devices, interrupts, CPU time
- Virtual machines access virtual resources
 - mapped to physical resources by hypervisor
- Guest OS executes at lesser privilege
 - only hypervisor runs in most privileged mode
 - essential to ensure hypervisor has control over resources
- Hypervisor schedules virtual machines
 - Performs *world switch* between VMs
 - Each guest OS schedules its apps



Outline



*Be open.
Be safe.*

- What are virtual machines?
- *Why virtualize embedded Linux?*
- How does it differ from enterprise virtualization?
- Embedded virtualization with OKL4

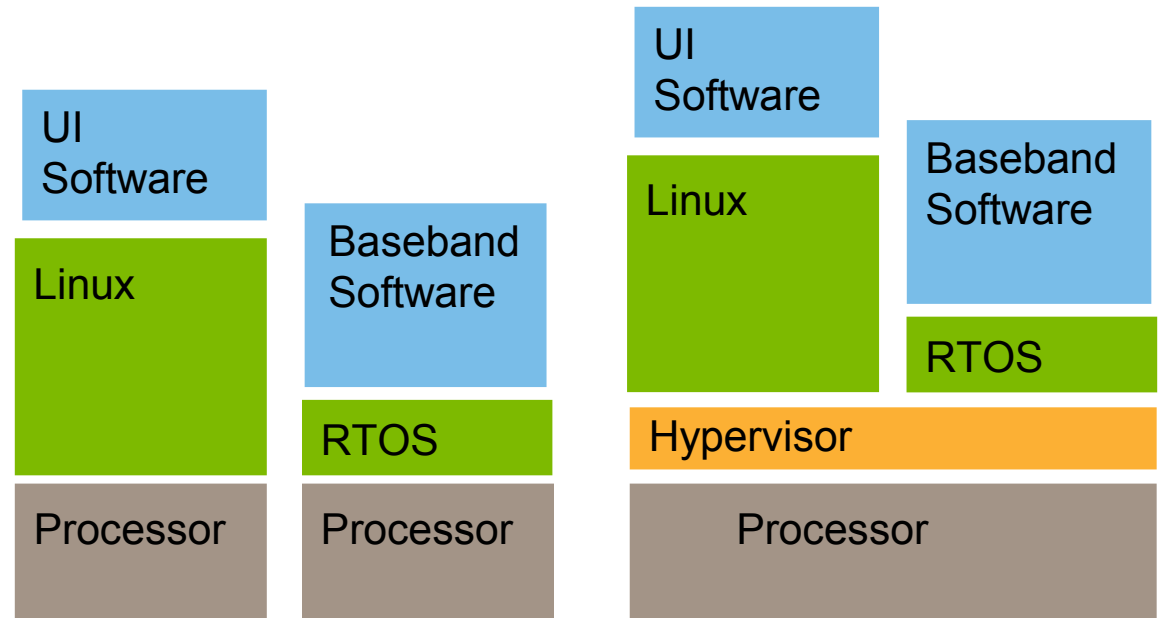
Why Virtualize Embedded Linux?



*Be open.
Be safe.*

Use case 1: Mobile phone processor consolidation

- High-end phones run high-level OS (Linux) on app processor
 - supports complex UI software
- Baseband processing supported by real-time OS (RTOS)
- Medium-range phone needs less grunt
 - can share processor
 - two VMs on one physical processor
 - hardware cost reduction



Why Virtualize Embedded Linux?

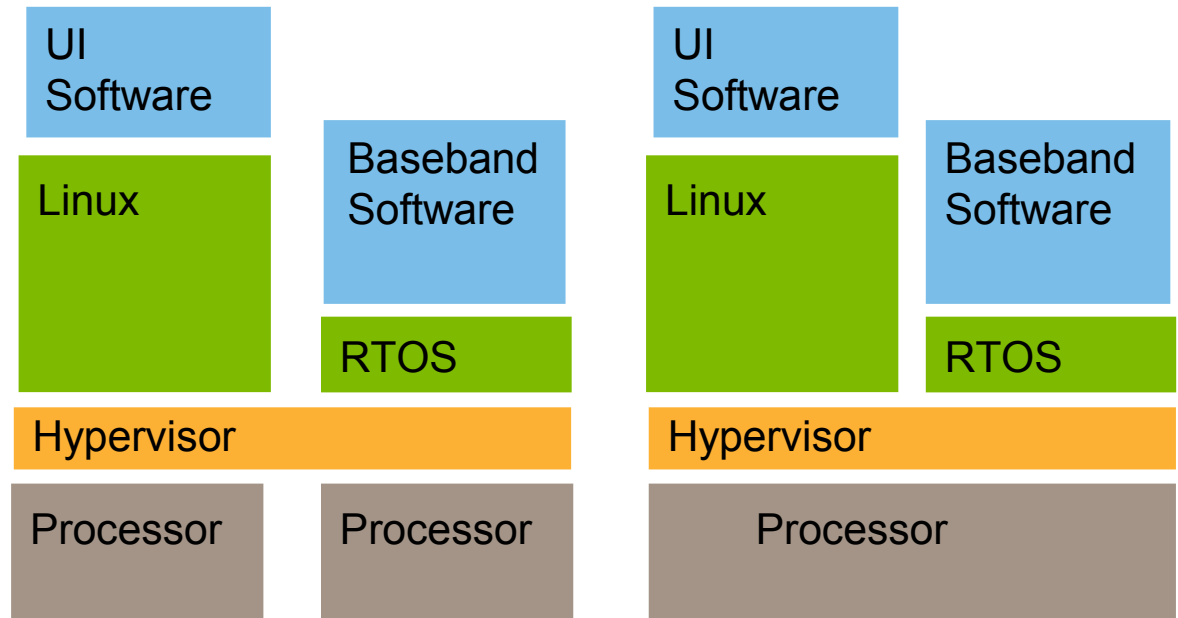


Be open.
Be safe.

Use case 1a: Software architecture abstraction

- Support for *product series*
 - range of related products of varying capabilities
- Same low-level software for high- and medium-end devices
- Benefits:

- time-to-market
- engineering cost



Why Virtualize Embedded Linux?

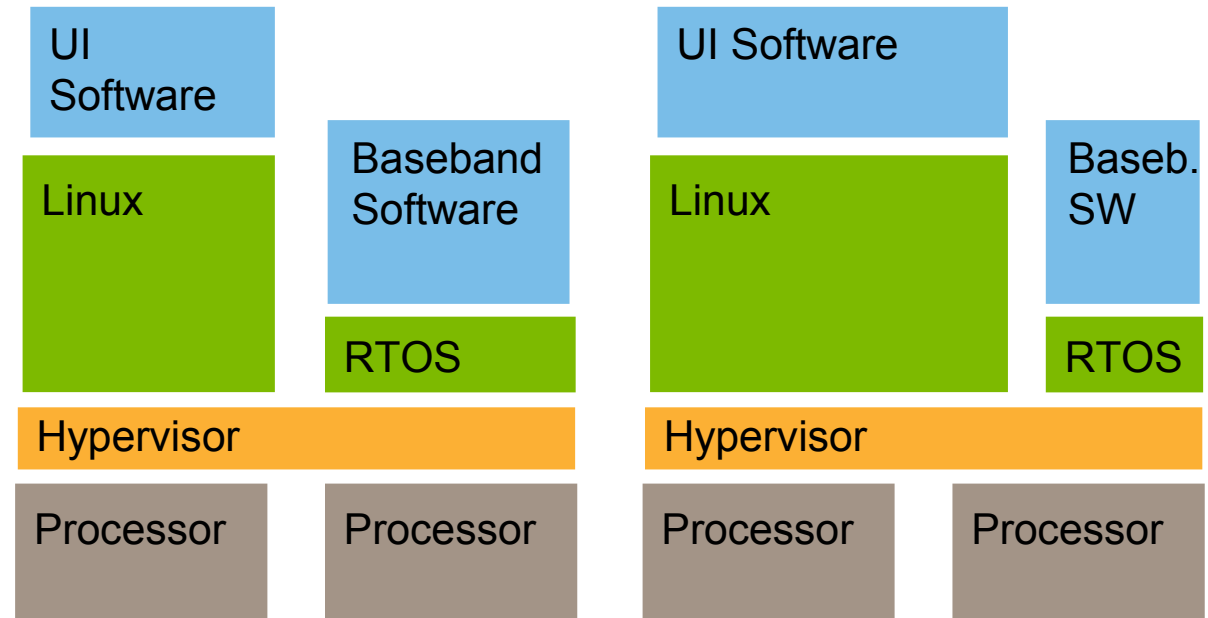


*Be open.
Be safe.*

Use case 1b: Dynamic processor allocation

→ Allocate share of baseband processor to application OS

- Provide extra CPU power during high-load periods (media play)
- Better processor utilisation \Rightarrow higher performance with lower-end hardware
- HW cost reduction



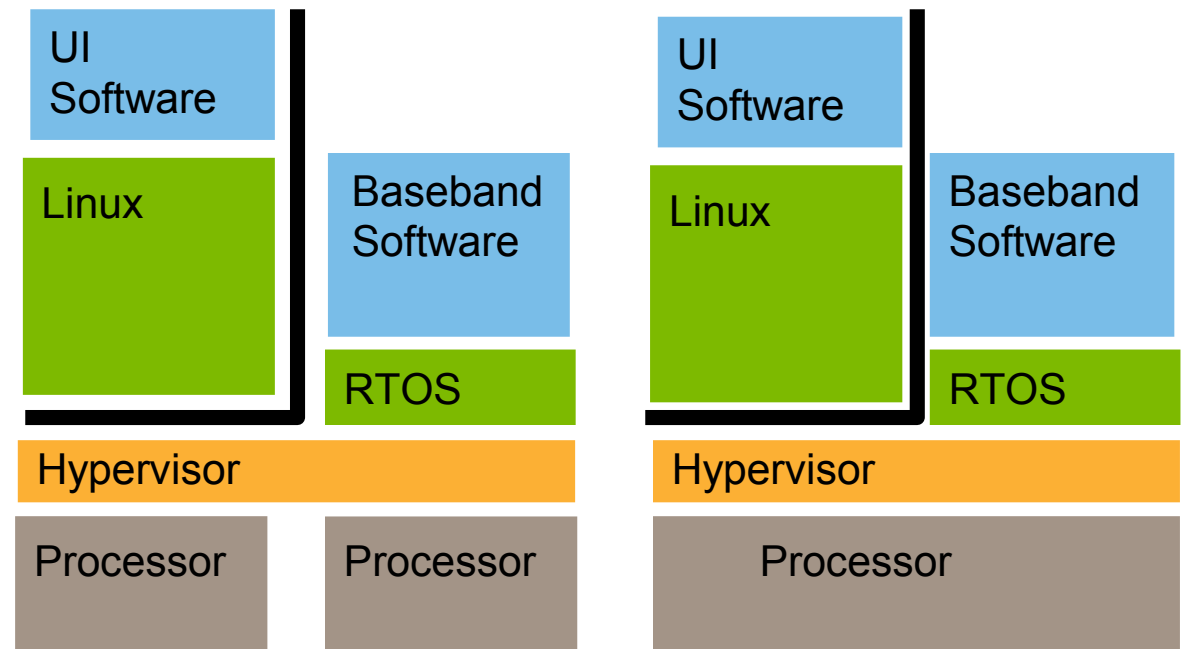
Why Virtualize Embedded Linux?



*Be open.
Be safe.*

Use case 2: Certification re-use

- Phones need to be certified to comply with communication standards
- Any change that (potentially) affects comms needs re-certification
- UI part of system changes frequently
- Encapsulation of UI
 - provided by VM
 - avoids need for costly re-certification



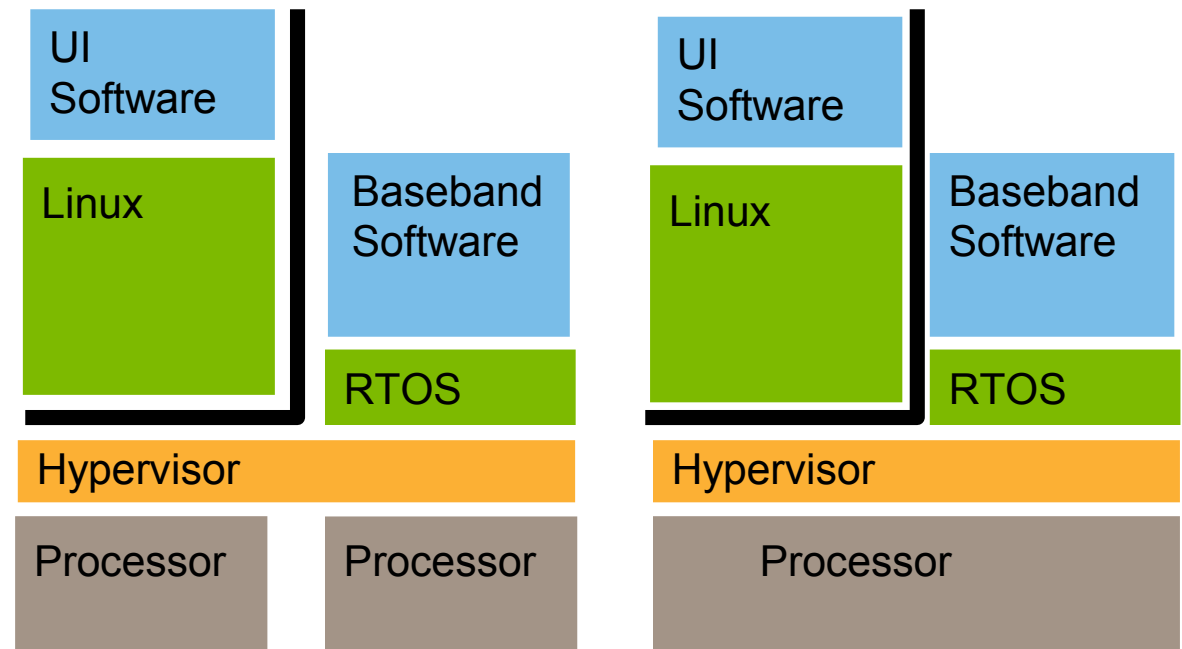
Why Virtualize Embedded Linux?



*Be open.
Be safe.*

Use case 2a: Open phone with user-configured OS

- Give users control over the application environment
 - perfect match for Linux
- Requires strong encapsulation of application environment
 - without undermining performance!



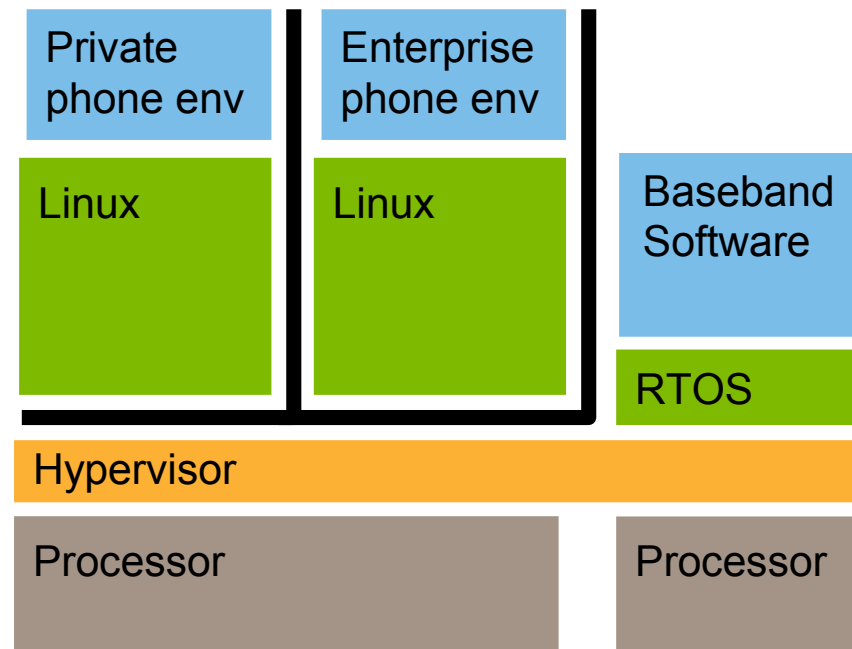
Why Virtualize Embedded Linux?



*Be open.
Be safe.*

Use case 2b: Phone with private and enterprise environment

- Work phone environment integrated with enterprise IT system
- Private phone environment contains sensitive personal data
- Mutual distrust between the environments ⇒ strong isolation needed



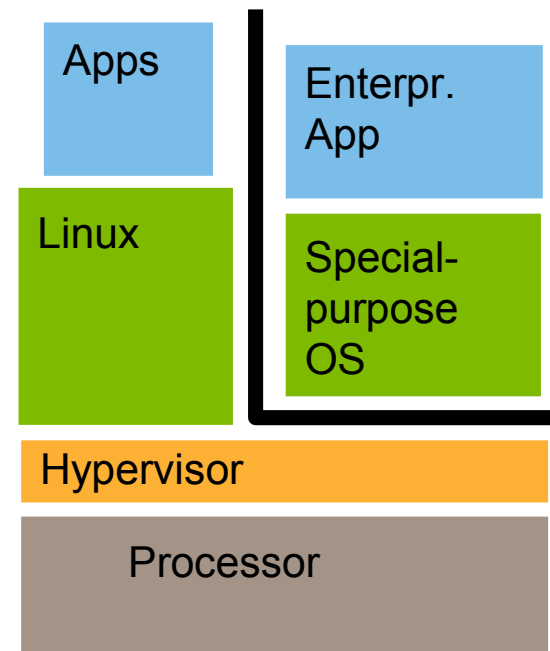
Why Virtualize Embedded Linux?



*Be open.
Be safe.*

Use case 3: Mobile internet device (MID) with enterprise app

- MID is open device, controlled by owner
- Enterprise app is closed and controlled by enterprise IT dept
- Hypervisor provides isolation



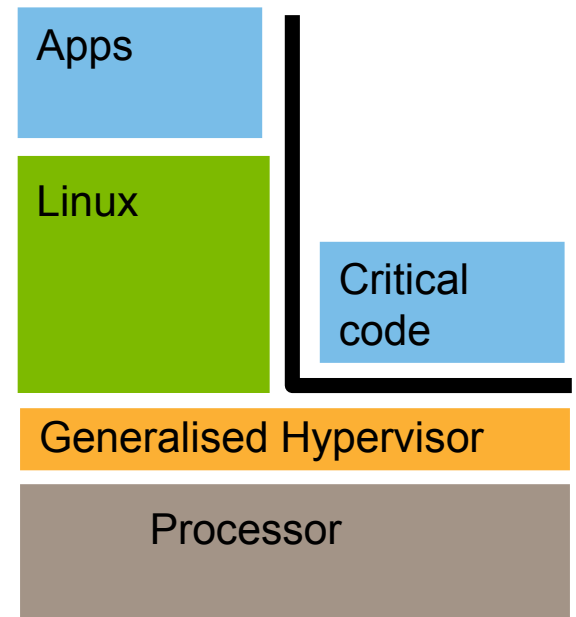
Why Virtualize Embedded Linux?



*Be open.
Be safe.*

Use case 3a: Environment with minimal *trusted computing base* (TCB)

- Minimise exposure of highly security-critical service to other code
- Avoid even an OS, provide minimal trusted environment
 - need a minimal programming environment
 - goes beyond capabilities of normal hypervisor
 - requires basic OS functionality



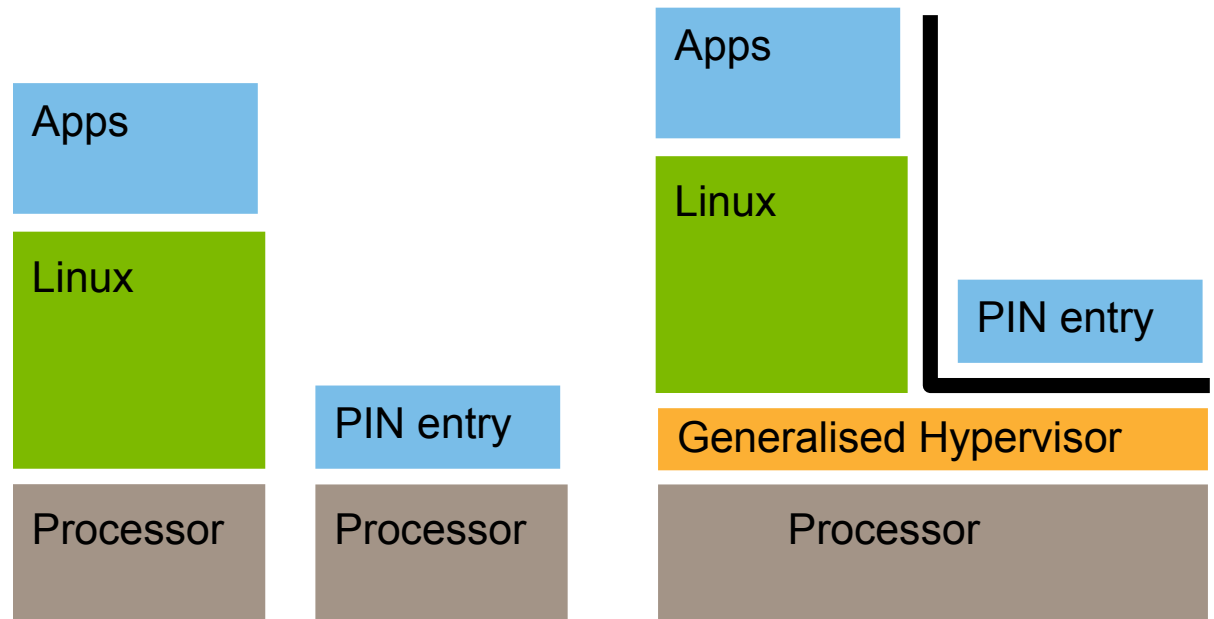
Why Virtualize Embedded Linux?



Be open.
Be safe.

Use case 3b: Point-of-sale (POS) device

- May be stand-alone or integrated with other device (eg phone)
- Financial services providers require strong isolation
 - dedicated processor for PIN/key entry
 - use dedicated *virtual processor* ⇒ HW cost reduction



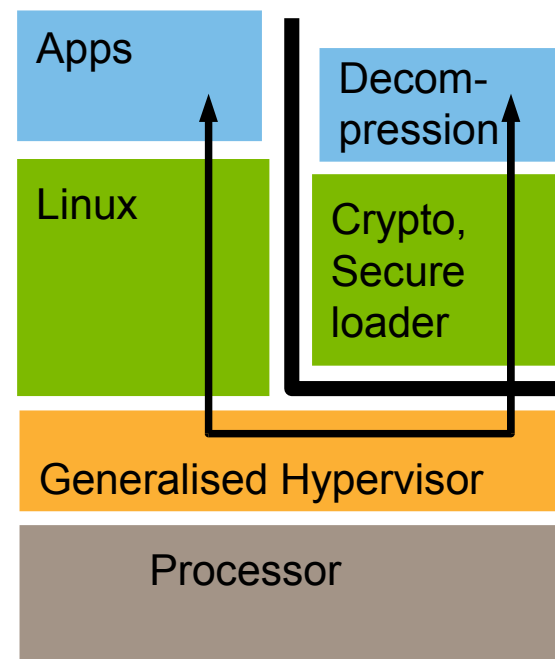
Why Virtualize Embedded Linux?



*Be open.
Be safe.*

Use case 4: IP protection in set-top box

- STB runs Linux for UI, but also contains highly valuable IP
 - highly-efficient, proprietary compression algorithm
- Operates in hostile environment
 - reverse engineering of algorithms
- Need highly-trustworthy code that
 - loads code from Flash into on-chip RAM
 - decrypts code
 - runs code protected from interference



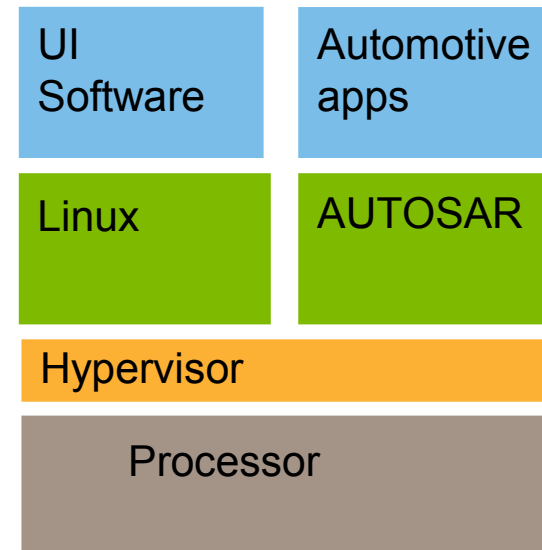
Why Virtualize Embedded Linux?



Be open.
Be safe.

Use case 5: Automotive control and infotainment

- Trend to processor consolidation in automotive industry
 - top-end cars have > 100 CPUs!
 - cost, complexity and space pressures to reduce by an order of magnitude
 - AUTOSAR OS standard addressing this for control/convenience function
- Increasing importance of *Infotainment*
 - driver information and entertainment function
 - not addressed by AUTOSAR
- Increasing overlap of infotainment and control/convenience
 - eg park-distance control using infotainment display
 - benefits from being located on same CPU



Outline



*Be open.
Be safe.*

- What are virtual machines?
- Why virtualize embedded Linux?
- *How does it differ from enterprise virtualization?*
- Embedded virtualization with OKL4

Enterprise vs Embedded Virtualization

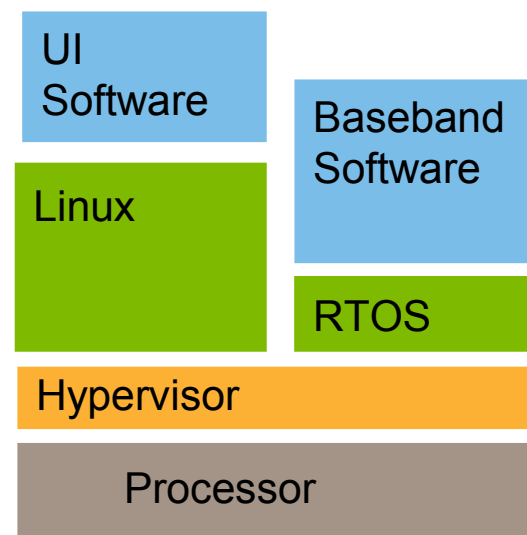
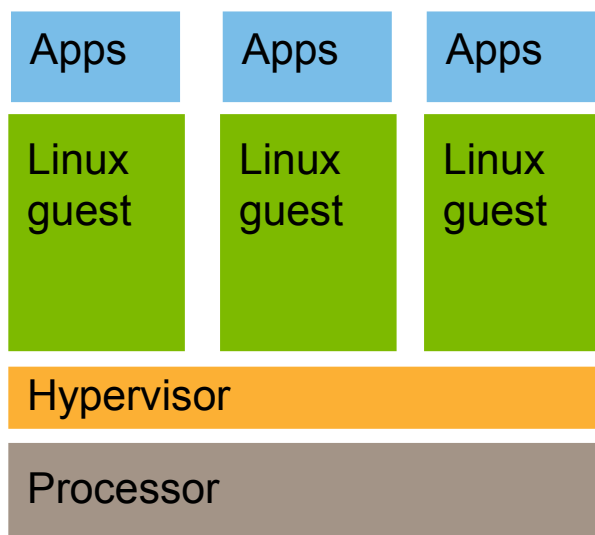


*Be open.
Be safe.*

Homogenous vs heterogenous guests

- Enterprise: many Linux VMs
- hypervisor size irrelevant
 - VMs scheduled round-robin

- Embedded: 1 Linux + 1 RTOS
- hypervisor resource-constrained
 - interrupt latencies matter



Enterprise vs Embedded Virtualization

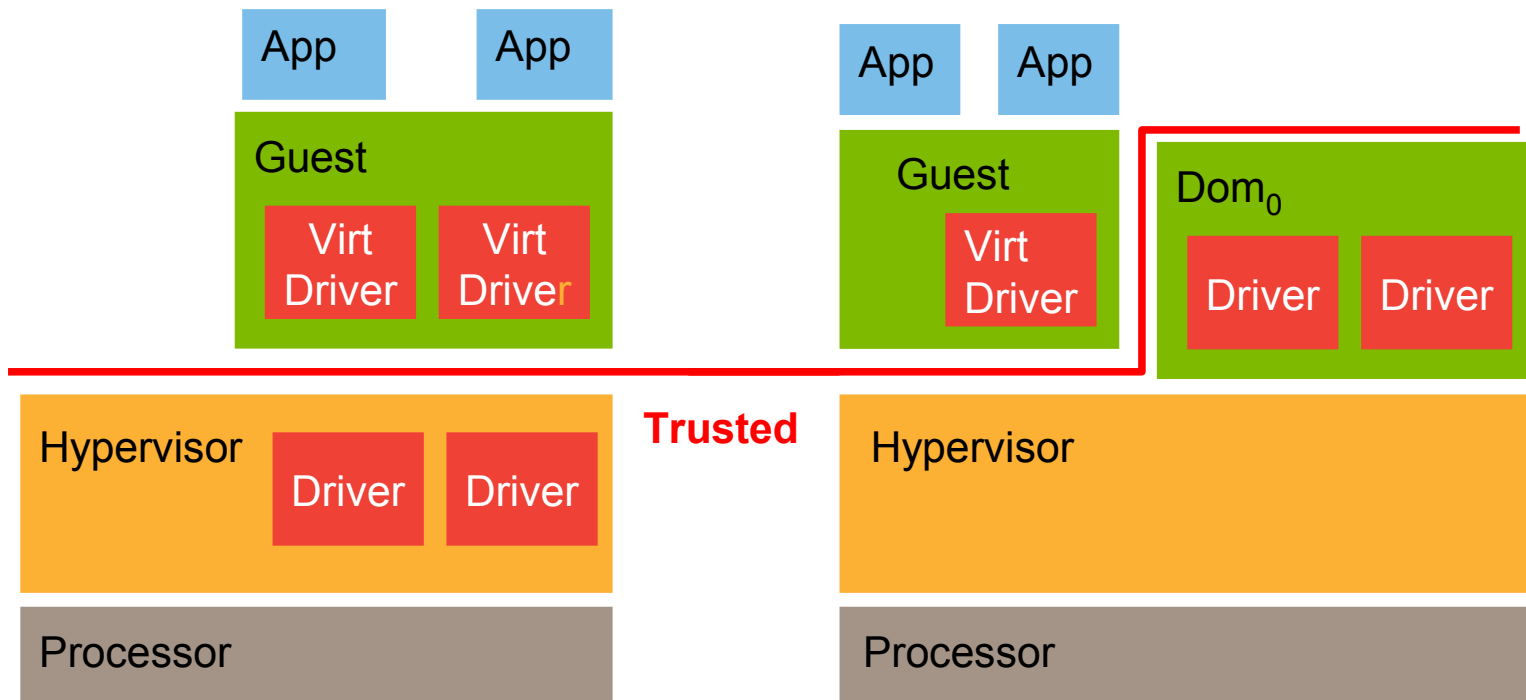


*Be open.
Be safe.*

Devices in enterprise-style virtual machines

- Hypervisor owns all devices
- Drivers in hypervisor
 - need to port all drivers
 - huge TCB

- Drivers in privileged guest OS
 - can leverage guest's driver support
 - still huge TCB!



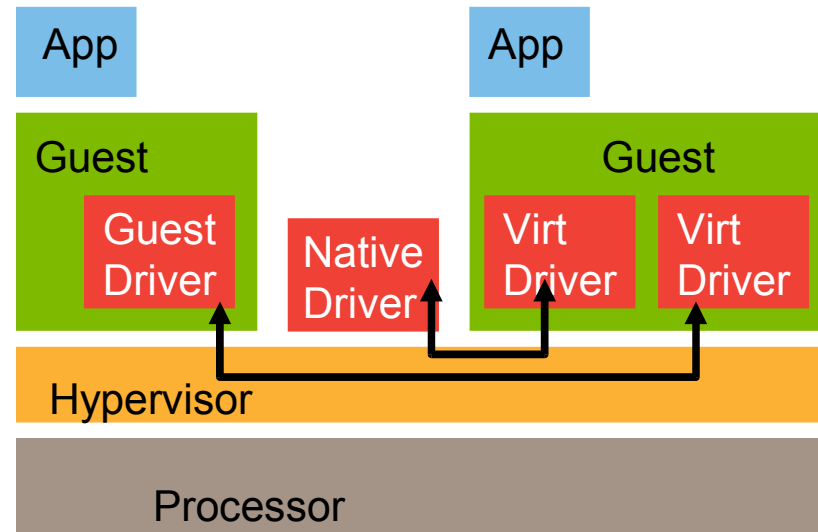
Enterprise vs Embedded Virtualization



*Be open.
Be safe.*

Devices in embedded virtual machines

- Some devices owned by particular VM
- Some devices shared
- Some devices too sensitive to trust guest
- Driver OS way too resource-hungry
- Need small TCB
- Need to share drivers
 - native minimal TCB drivers
 - guest drivers



Core Difference: Isolation vs Cooperation



*Be open.
Be safe.*

Enterprise

- Independent services
- Emphasis on isolation
- Inter-VM communication is secondary
 - performance secondary
- VMs connected to Internet (and thus to each other)
 - can freely communicate with each other and the rest of the world

Embedded

- Integrated system
- Cooperation with protection
- Inter-VM communication is critically important
 - performance crucial
- VMs are subsystems accessing shared (but restricted) resources
 - communication must be controlled

- Need: **Strong protection, yet Highly-efficient (low-latency, high-bandwidth) communication**
- This doesn't fit the virtual machine model!
- Need a generalisation of VMs

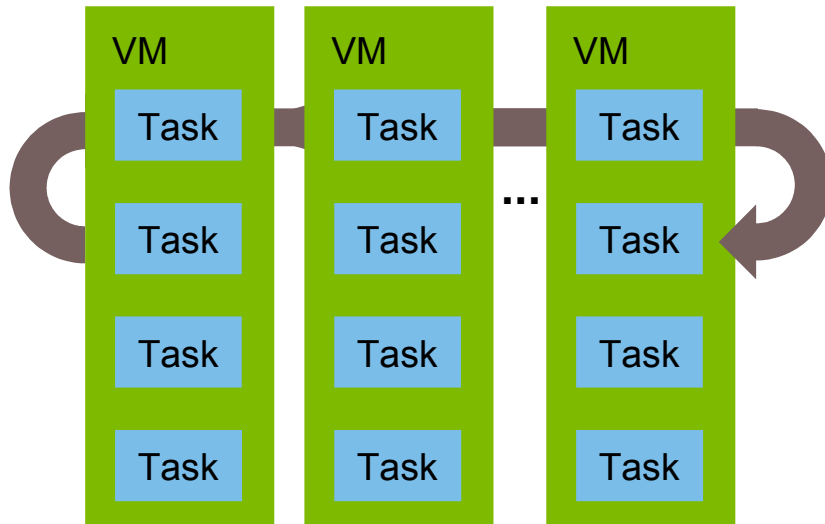
Isolation vs Cooperation: Scheduling



*Be open.
Be safe.*

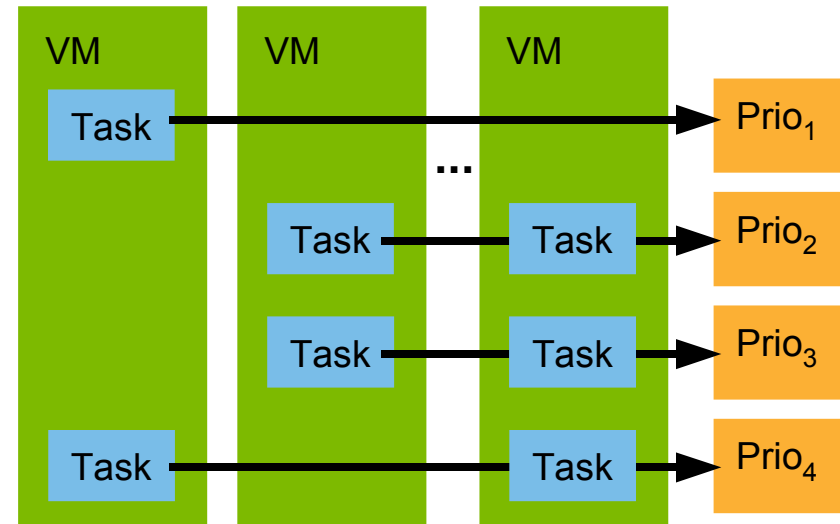
Enterprise

- Round-robin scheduling of Vms
- Guest OS schedules its apps



Embedded

- Global view of scheduling
- Schedule threads, not VMs



- This doesn't fit the virtual machine model!
- Need a generalisation of VMs

Inter-VM Communication Control



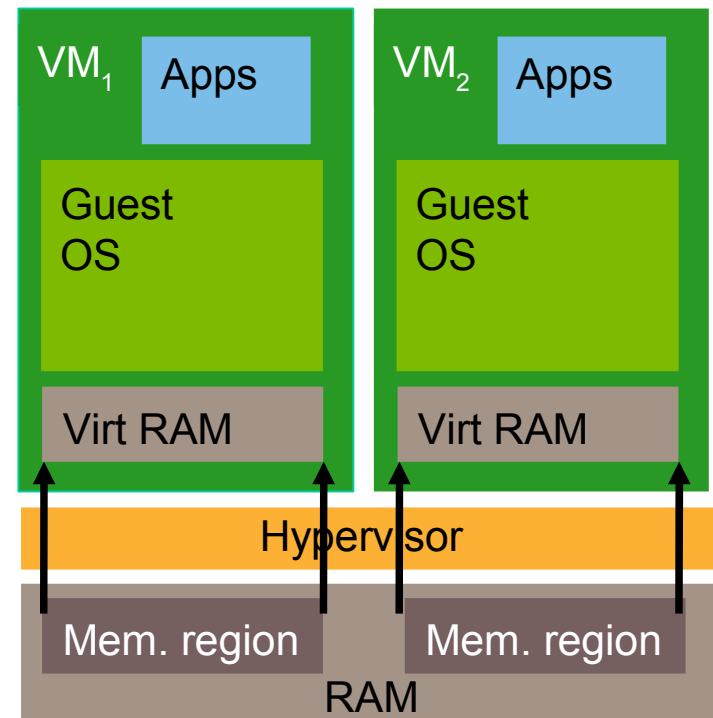
*Be open.
Be safe.*

Modern embedded systems are multi-user devices!

- Eg a phone has three *classes* of “users”:
 - the owner of the physical device
 - assets: private data, access keys
 - the network operator(s)
 - assets: cellular network
 - content providers
 - media content
- They are mutually distrusting
 - Need to protect integrity and confidentiality against *internal* exploits
 - Need control over *information flow*
 - strict control over who has access to what
 - strict control over communication channels

What is *Not* Different: Isolation

- Hypervisor partitions and multiplexes hardware between guests
- Hypervisor is in complete control of all physical resources
 - memory, devices, interrupts, CPU time
- Virtual machines access virtual resources
 - mapped to physical resources by hypervisor
- Guest OS executes at lesser privilege
 - *only hypervisor runs in privileged mode*
 - essential to ensure hypervisor has control
 - *Isolation is inherent in VM concept, it isn't optional!*
- Some products violate this!
 - Run guests in most privileged mode
 - *This is OS co-location, not virtual machines!*
 - Virtualize only interrupts, not system!



OS Co-location Doesn't Help Much



*Be open.
Be safe.*

Does it support these use cases?

- 1: Mobile phone processor consolidation **Not really**
- 1a: Software architecture abstraction **Maybe**
- 1b: Dynamic processor allocation **Maybe**
- 2: Certification re-use **No**
- 2a: Open phone with user-configured OS **No**
- 2b: phone with private and enterprise environment **No**
- 3: Mobile internet device (MID) with enterprise app **No**
- 3a: Environment with TCB **No**
- 3b: Point-of-sale (POS) device **No**
- 4: IP protection in set-top box **No**
- 5: Automotive control and infotainment **No**

OS Co-location — Why Bother?



Be open.
Be safe.

- Typical argument: *Need to run guest in kernel mode for low interrupt latencies*
- This doesn't hold water!
- Actions required on interrupt
 - Enter interrupt mode — *in average faster from user mode (always preemptible)*
 - Save executing thread state — *same in both cases*
 - Restore interrupt handler state — *same in both cases*
 - Switch addressing context — *this can be saved by running guests in kernel mode*
 - Exit interrupt mode — *a few cycles slower if going back to user mode*
- Other actions required as a result of the design of your system
 - These are independent on whether the guest runs in user or kernel mode
- Total possible savings by running guest in kernel mode
 - one switch of addressing context, plus a few cycles for return to user mode
 - cost on RISC processors: < 20 cycles (50 nsec on 400MHz ARM)
 - typical total worst-case interrupt latency: 5 μ sec
- *Running guests in kernel mode is outright stupid!*

Summary of Required Properties



*Be open.
Be safe.*

- Fast context switches
- High-performance communication
- Encapsulated device drivers
- Highly trustworthy privileged code
 - small and clean
 - open source
- Minimal-TCB execution environment
- Fine-grained access/communications control
- Ideally: Answer to future security/safety challenges

This goes well beyond the classical model of a hypervisor!

- Embedded hypervisor must be augmented by general OS capabilities
- High-performance microkernel is a suitable platform

Outline



*Be open.
Be safe.*

- What are virtual machines?
- Why virtualize embedded Linux?
- How does it differ from enterprise virtualization?
- *Embedded virtualization with OKL4*

Generalized Hypervisor: OKL4 Microkernel



*Be open.
Be safe.*

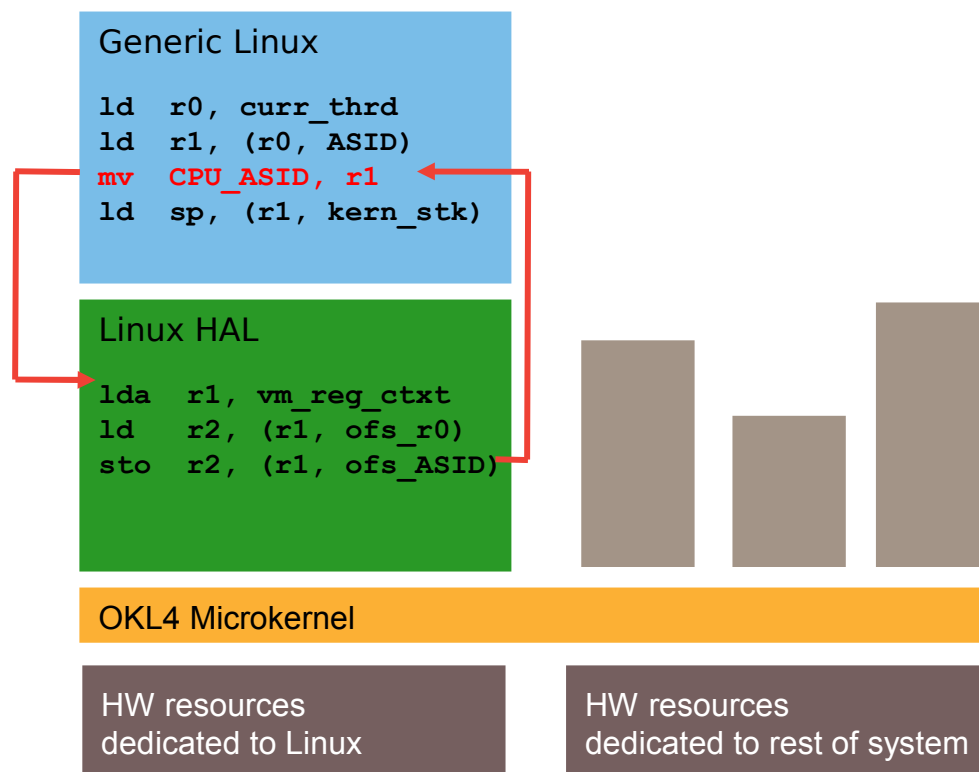
- Small kernel providing core functionality
 - 10 kLOC
 - no other code running in privileged mode, not even device drivers
 - provide mechanisms for building arbitrary systems on top
 - services can have 15 kLOC TCB
- Fast message-passing IPC operation
 - performance close to the hardware limit
- Shared memory for bulk memory transfer
- 10-year track record of high-performance Linux virtualization
- Suitable as a basis for general operating systems
- Powerful and sophisticated protection system
- *Open source!*

Microkernel as Hypervisor: OK Linux on OKL4



Be open.
Be safe.

- Linux ported to OKL4 “architecture”
 - Mapping to OKL4 API
 - Linux runs in user mode
- Microkernel reflects traps back to Linux HAL
 - Performs virtualization
 - Returns to trapping code
- HAL mirrors virtual state
 - Efficient virtualization
 - Synchronize state lazily
- Linux is securely restricted to HW resources dedicated to it

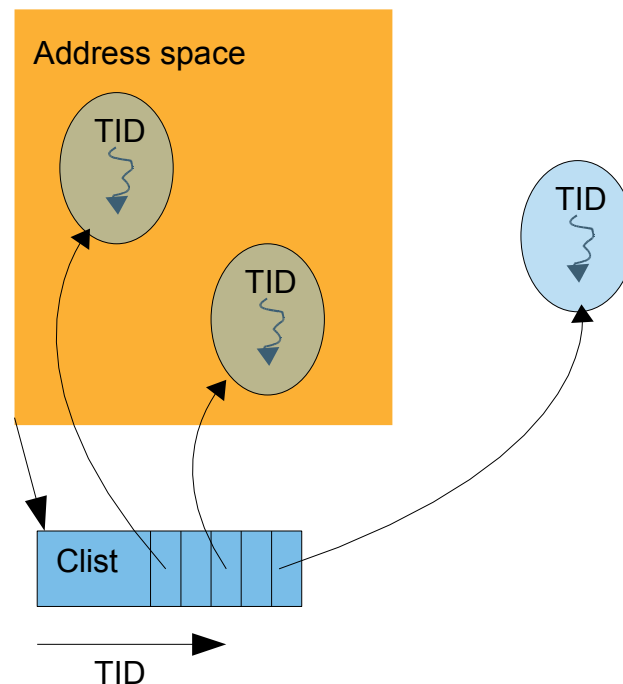


OKL4 Resource and Security Management



Be open.
Be safe.

- Resources controlled by kernel-protected *capabilities*
- Capability conveys privilege
- Efficient resource delegation by providing set of caps
- Subject to system-wide security and resource-management policies defined by system designer
- *Secure HyperCell™ technology* provides security management framework
- Work in progress on extending this to military-strength security

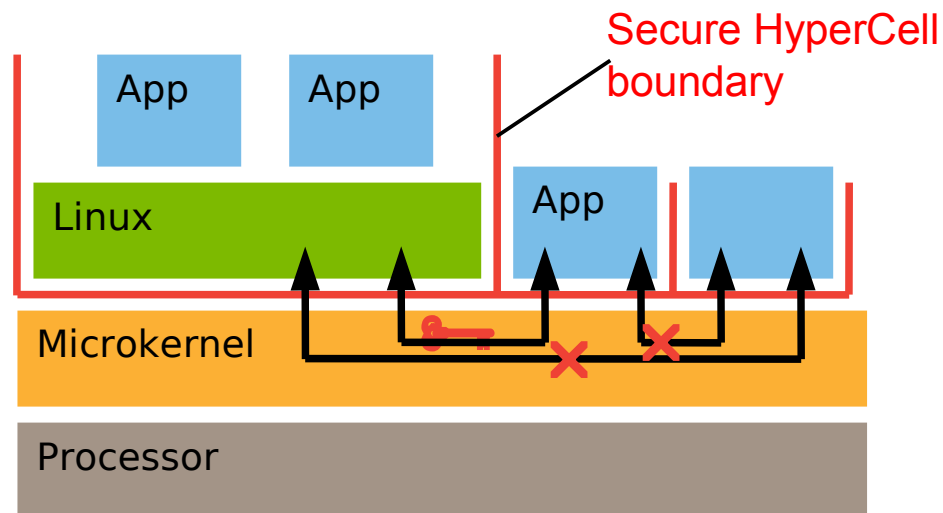


Critical Service with Small TCB



Be open.
Be safe.

- Run normal apps on Linux
- Run security-critical apps native on OKL4, in separate Secure HyperCell
 - crypto services
 - e-cash...
- Security policy defines allowed communication channels
- *TrustZone-like isolation*
 - but more general
 - eg arbitrary number of cells, more than two security levels
 - no hardware requirements beyond MMU
- TCB: <15kLOC

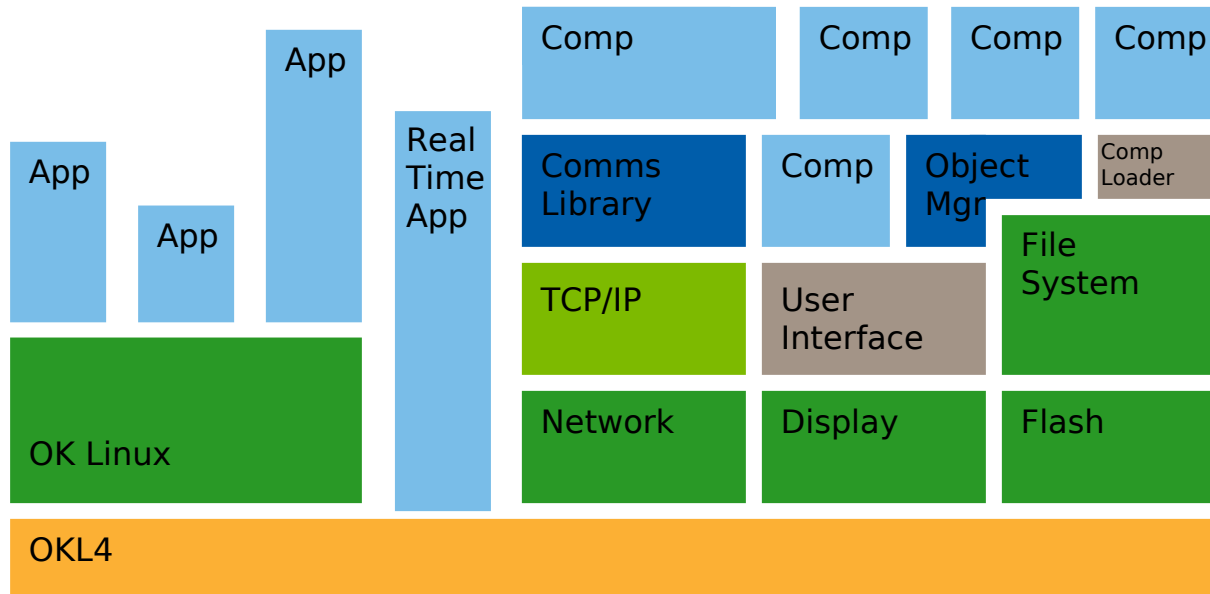


Other Benefits: Hybrid System: Linux + Native



*Be open.
Be safe.*

- Linux for traditional apps
- Highly-componentised environment
 - Fine-grained isolation of subsystems
 - Internal protection boundary
 - Fault isolation, fault tolerance



ARM11-like Performance on ARM9



Be open.
Be safe.

Benchmark	Native	Virtualized	Ratio
<i>Imbench latencies in microseconds, smaller is better</i>			
lat_ctx -s 0 1	11	20	0.55
lat_ctx -s 0 2	262	5	52.4
lat_ctx -s 0 10	298	45	6.62
lat_ctx -s 4 1	48	58	0.83
lat_ctx -s 4 10	419	203	2.06
lat_fifo	509	49	10.39
lat_pipe	509	49	10.39
lat_unix	1015	77	13.18

**Native Linux (2.6.23) vs OK Linux on OKL4 2.1
XScale PXA255 @ 200MHz**

Deployed in 100+ Million Devices!



*Be open.
Be safe.*

- Mobile phones
 - Toshiba W47T CDMA phone selling in Japan late 2006
 - 3G phones from HTC, LG, Sony Ericsson etc shipping since July 2007
 - Linux phones later this year



HTC 8700



- Products in other industry verticals in pipeline
 - HD IP-TV settop box announced, to ship this year
 - etc, etc...



OKL4 Reliability and Security Roadmap



*Be open.
Be safe.*

- Military-strength security and isolation
 - based on further development of capability-based access control
 - kernel resources also subject to system's resource management policies
 - mathematical isolation proofs
- Mathematical proof of implementation correctness
 - kernel proven free of security-relevant bugs
 - possible due to small size of privileged-mode code base
- Strict analysis of real-time properties
 - provide real guarantees for worst-case latencies
- Enabled by close collaboration with NICTA
 - Australian ICT Research Centre of Excellence

The Open Kernel Advantage



*Be open.
Be safe.*

- *Open source*
- Mature: deployed on 100+ million devices
- No compromises: secure *and* fast
- Encapsulated device drivers
- Unbeaten performance — for ten years!
- ARM9 context switches as fast as ARM11
- Unique page-table compression technology for reduced memory use
- Active developer community
- Teaching platform in many universities
- Migration path to highly-componentized, fault-tolerant system
- Information-flow control mechanisms
- Aggressive roadmap to military-style security and formal correctness proofs
- Direct IP pipeline from world-class research center (NICTA)

Thank You



*Be open.
Be safe.*



Open Kernel LabsTM

Be open. Be safe.

Dr. Gernot Heiser, CTO

Open Kernel Labs
gernot@ok-labs.com