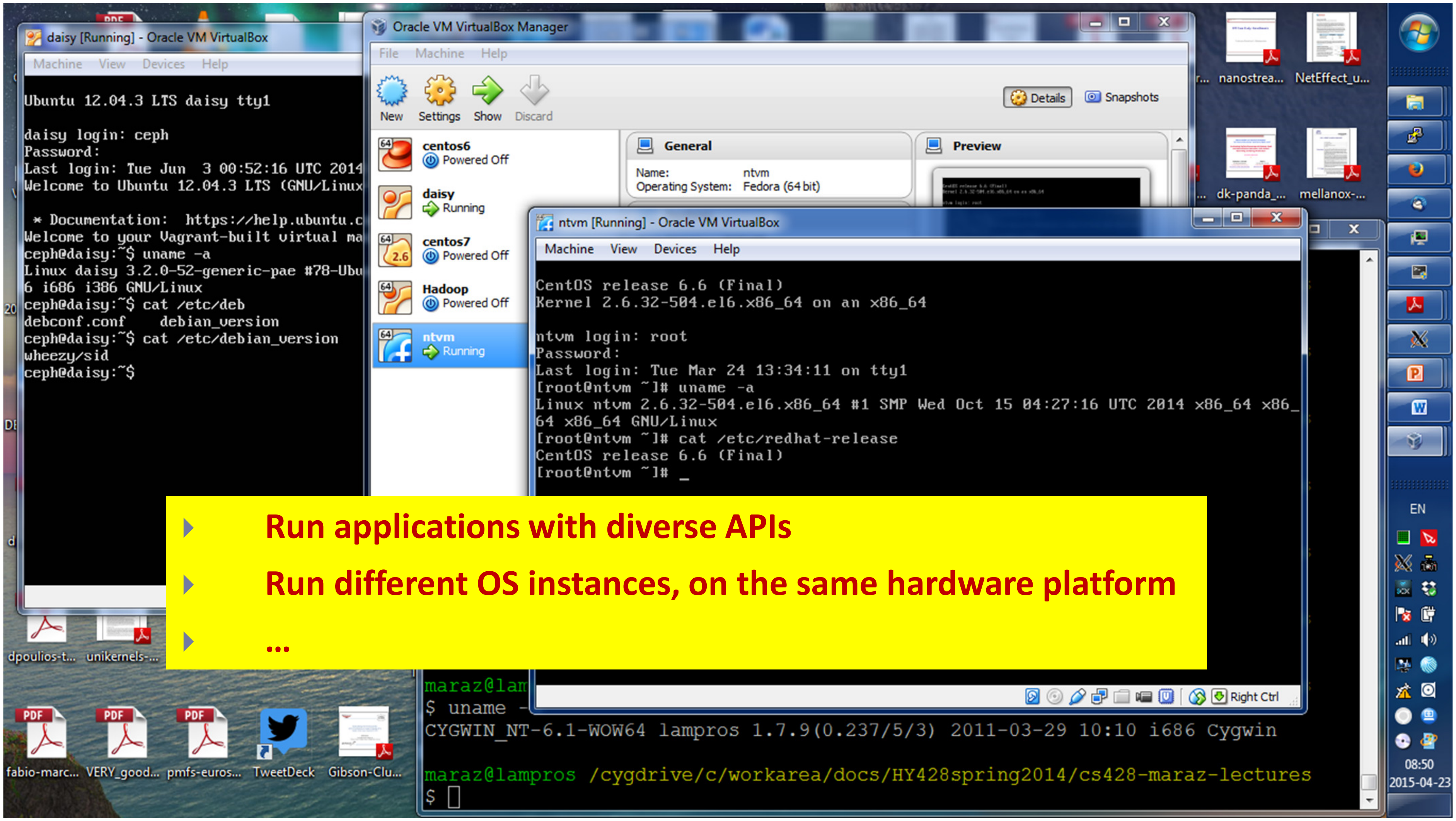
Virtualization for Embedded Systems
Lecture for the Embedded Systems Course
CSD, University of Crete (April 23, 2015)

- Manolis Marazakis (maraz@ics.forth.gr)

Institute of Computer Science (ICS)
Foundation for Research and Technology – Hellas (FORTH)

What does virtualization look like (informally) ?



- Run applications with diverse APIs
- Run different OS instances, on the same hardware platform
- …

Uses of virtual machines

- Multiple (identical) OS'es on same platform
  - The original *raison d'être*
  - These days mostly driven by server consolidation
- Interesting variants of this:
  - Different OSes (e.g. Linux + Windows)
  - Old version of same OS
  - OS debugging (most likely uses Type-II VMM)
  - Checkpoint-restart
    - minimize lost work in case of crash
    - useful for debugging, incl. going backwards in time
    - re-run from last checkpoint to crash, collect traces, invert trace from crash
- Live system migration
  - Load balancing, Environment take-home
- Ship application with complete OS
  - Reduce dependency on environment
- **How about embedded systems?**

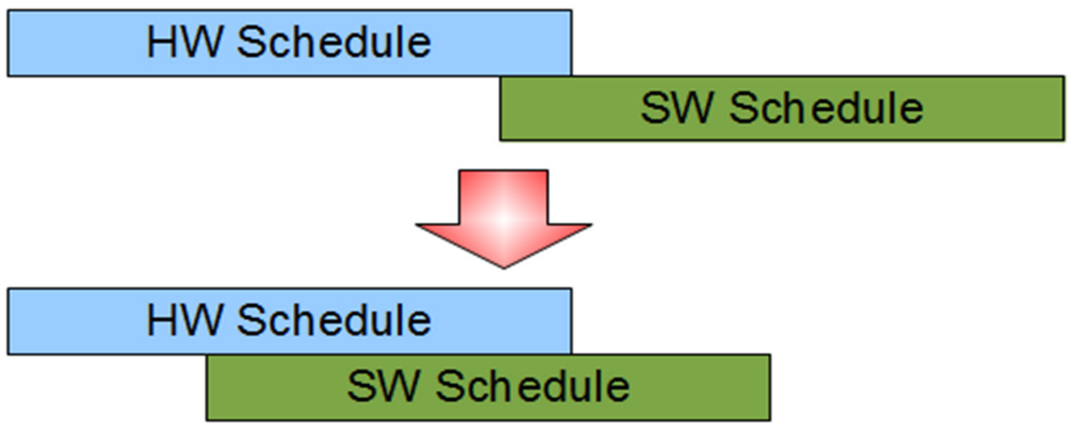
Virtualization to enable h/w-s/w co-design

- How to **co-design/co-develop H/W + S/W** for a system ?
  - Limited availability
  - Bugs in the production environment cannot be reproduced in the laboratory
  - Difficult to debug on-site
  - Narrow time windows
  - Sometimes in a dangerous environment …
- **Debugging** challenges
  - Is it a problem in the driver or in the device?
  - Is the firmware faulty? Is it wrongly loaded/configured?
  - Is the hardware damaged?
  - How can we reproduce the bug?
  - Do we have easy access to the environment?
  - Is it remotely located?

Writing (and testing) device drivers … without hardware

Shift Left

- Hardware + Software = Complete product
- Feature-complete software by A-0 silicon
- Software needs to happen earlier



[source: PJ Waskiewicz & Shannon Nelson - Linux Plumbers Conference, 2011]

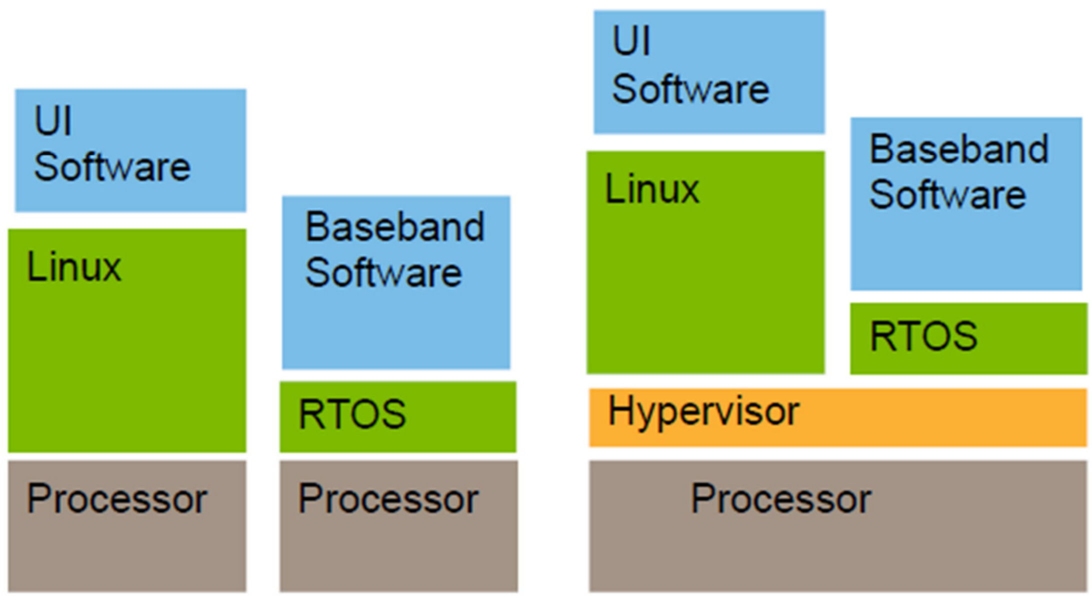
Processor consolidation

Use case 1: Mobile phone processor consolidation

→ High-end phones run high-level OS (Linux) on app processor
  - supports complex UI software

→ Base-band 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



Source: OS course slides by Gernot Heiser – UNSW/NICTA/OKL - 2008

License separation

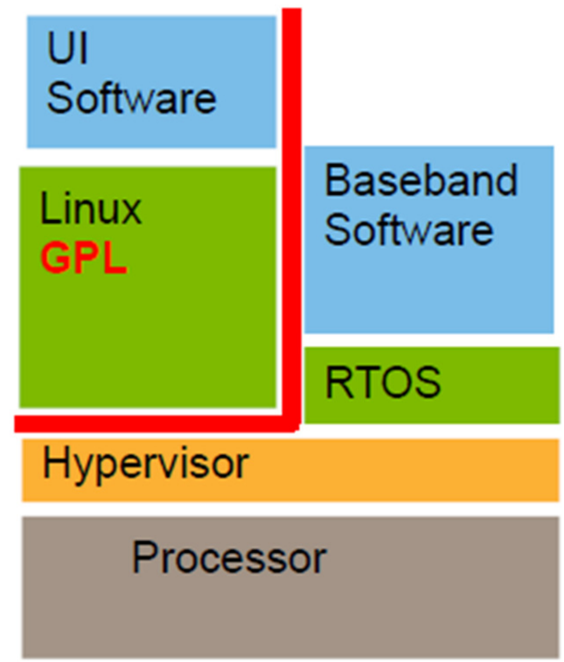
Use case 1a: License separation

→ Linux desired for various reasons
  - familiar, high-level API
  - large developer community
  - free

→ Other parts of system contain proprietary code

→ Manufacturer doesn't want to open-source

→ User VM to contain Linux + GPL



Software architecture abstraction

Use case 1b: Software-architecture abstraction

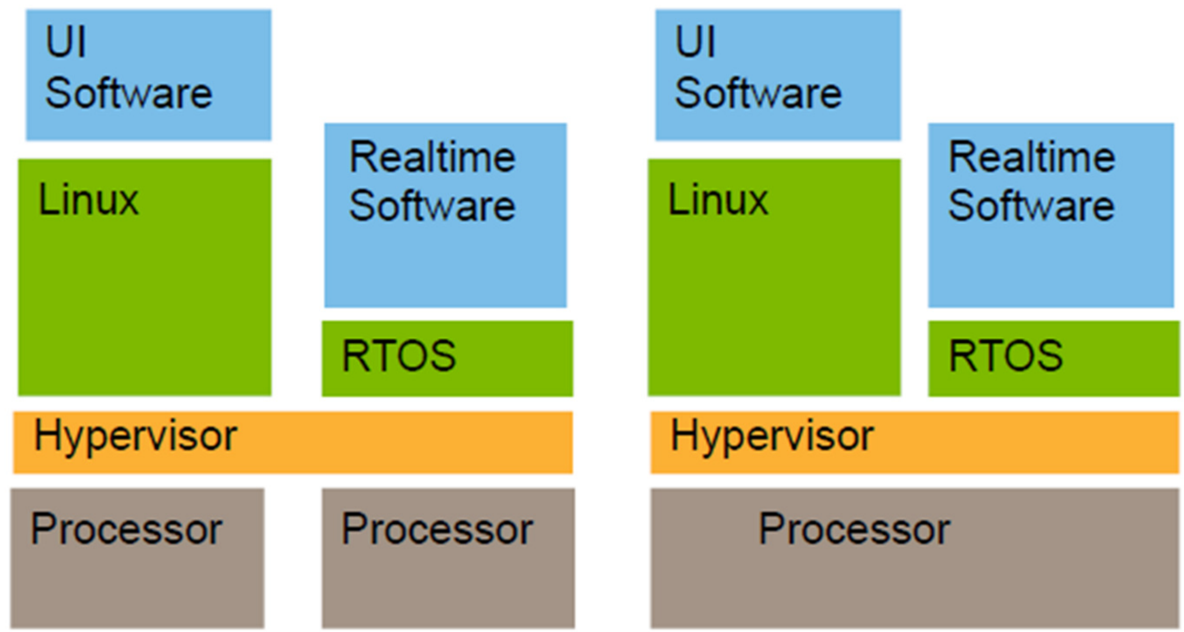
→ Support for *product series*

- range of related products of varying capabilities

→ Same low-level software for high- and medium-end devices

→ Benefits:

- time-to-market
- engineering cost

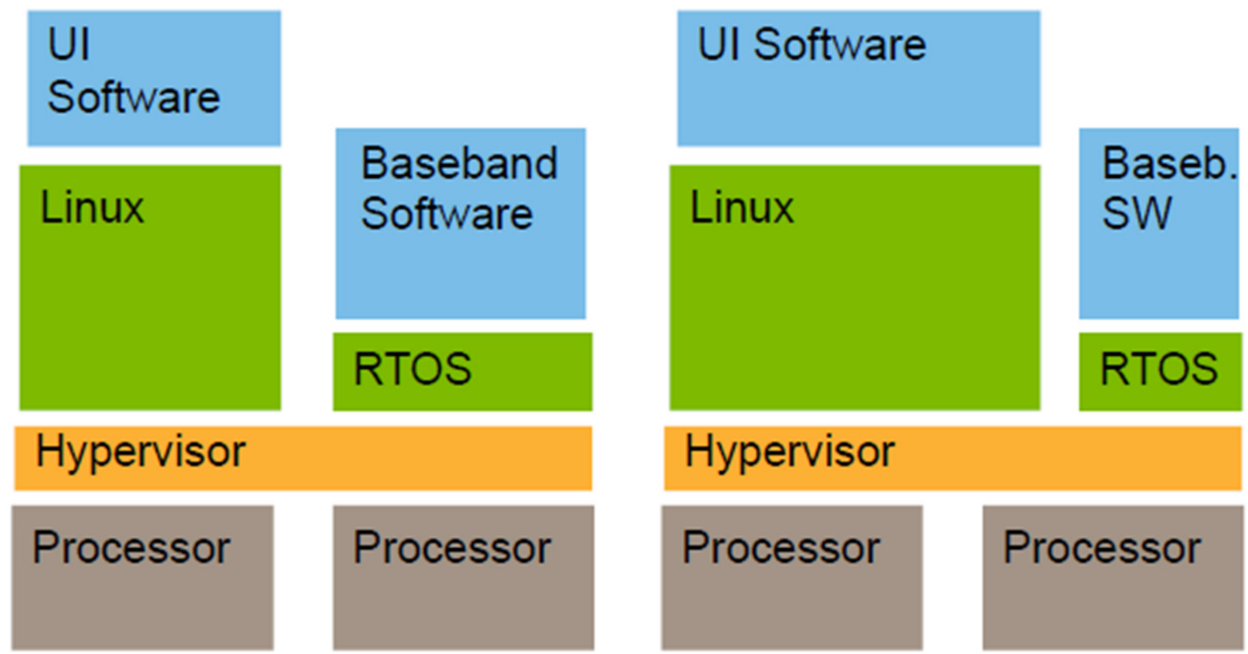


Dynamic processor allocation

Use case 1c: Dynamic processor allocation

→ Allocate share of base-band processor to application OS

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

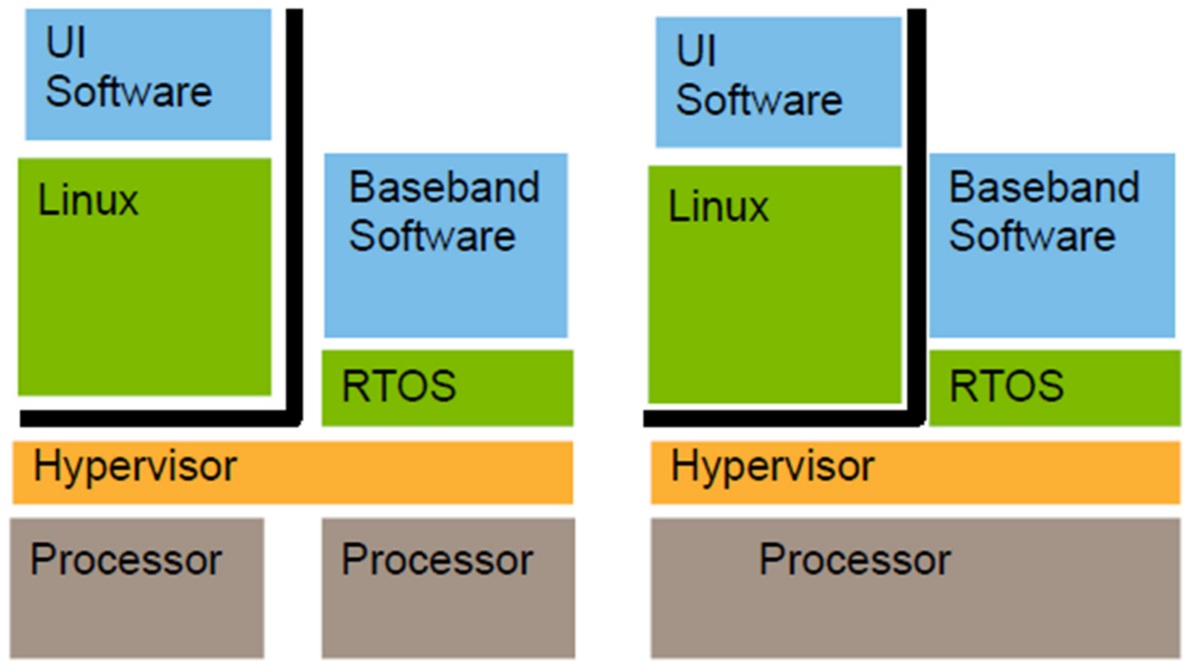


FORTH-ICS Institute of Computer Science

Certification re-use

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



User-configured OS

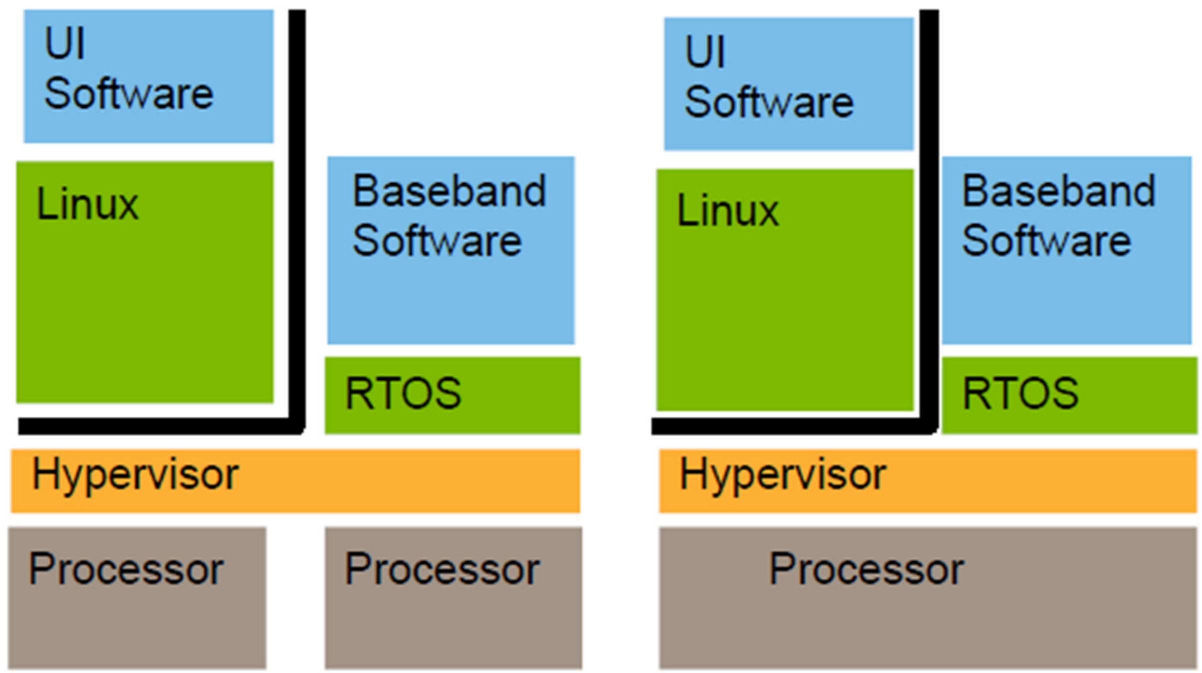
Use case 2a: Open phone with user-configured OS

→ Give users control over the application environment

- perfect match for Linux

→ Requires strong encapsulation of application environment

- without undermining performance!



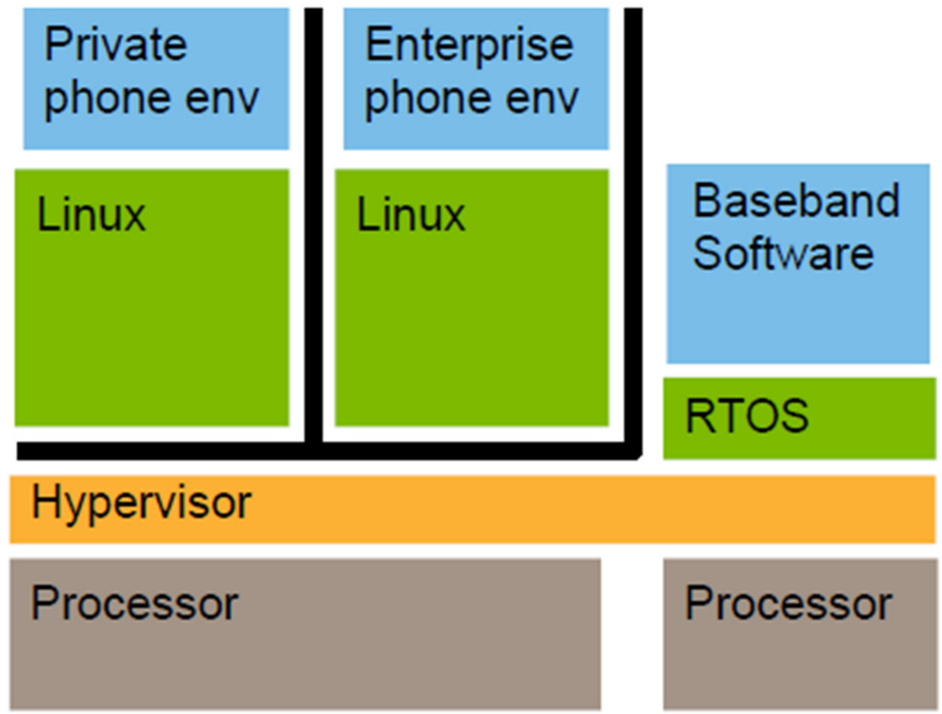
Personal + Enterprise environment

Use case 2b: Phone with private and enterprise environment

→ Work phone environment integrated with enterprise IT system

→ Private phone environment contains sensitive personal data

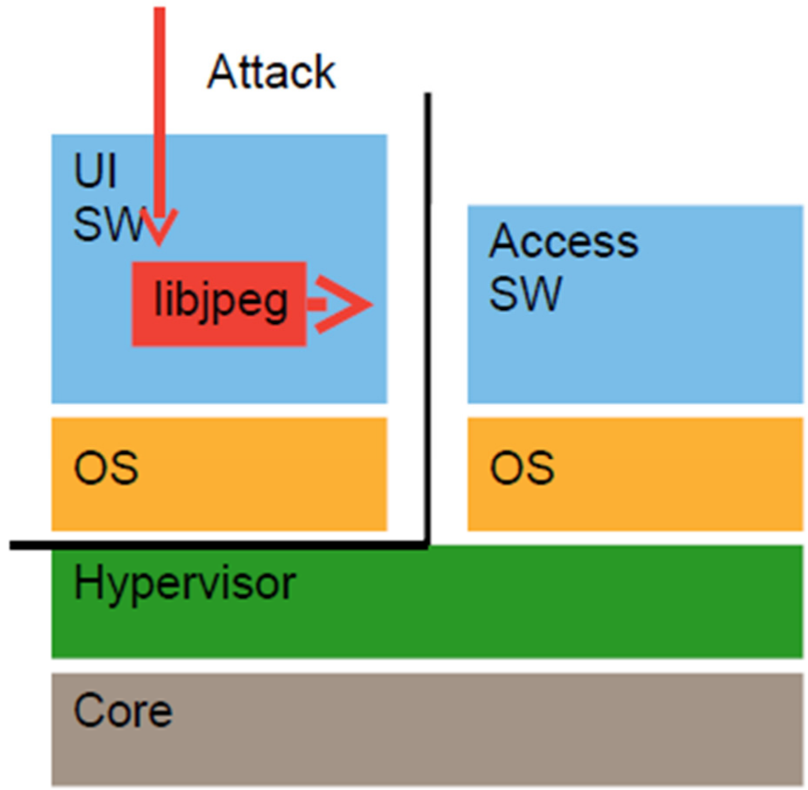
→ Mutual distrust between the environments ⇒ strong isolation needed



Separate systems code from apps

Use case 2c: Security

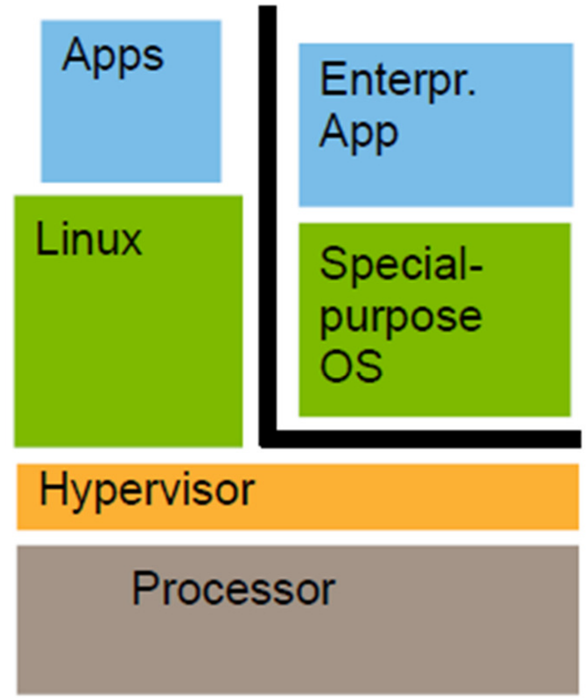
- → Protect against exploits
- → Modem software attacked by UI exploits
  - Compromised application OS could compromise RT side
  - Could have serious consequences
    - e.g. jamming cellular network
- → Virtualization protects
  - Separate apps and system code into different VMs



Isolation of Personal from Enterprise functions

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 department
- → Hypervisor provides isolation



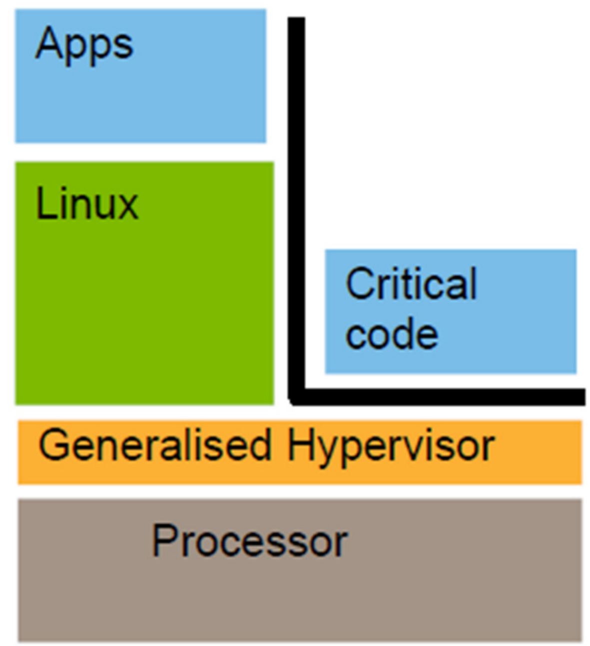
Minimal Trusted Computing Base

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

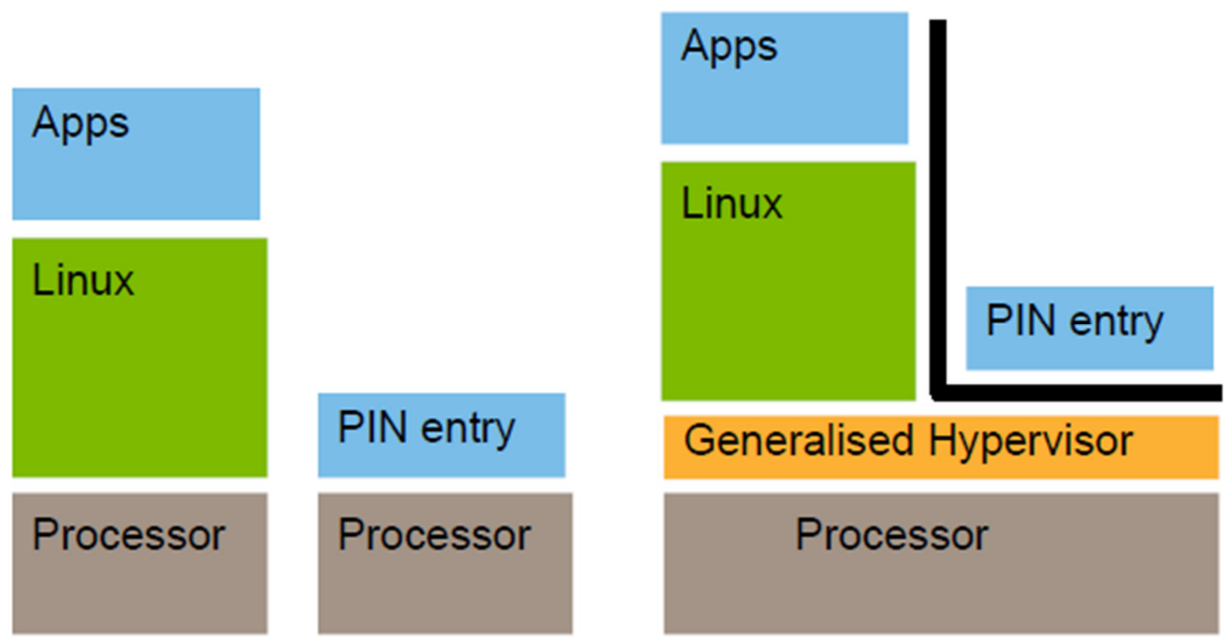
- need a minimal programming environment
- goes beyond capabilities of normal hypervisor
- requires basic OS functionality



Secure payments

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



Digital Rights Management … on open device

Use case 4: DRM on open device

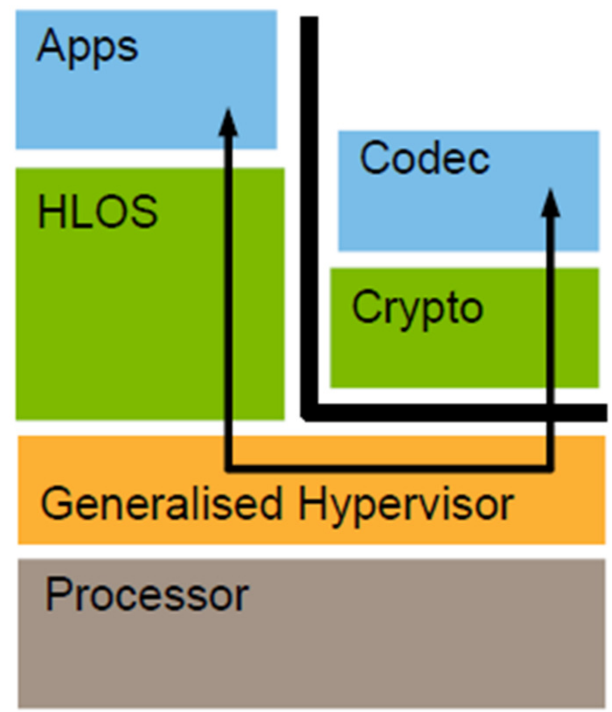
→ Device runs Linux as app OS, uses Linux-based media player

→ DRM must not rely on Linux

→ Need trustworthy code that

- loads media content into on-chip RAM
- decrypts and decodes content
- allows Linux-based player to disply

→ Need to protect data from guest OS

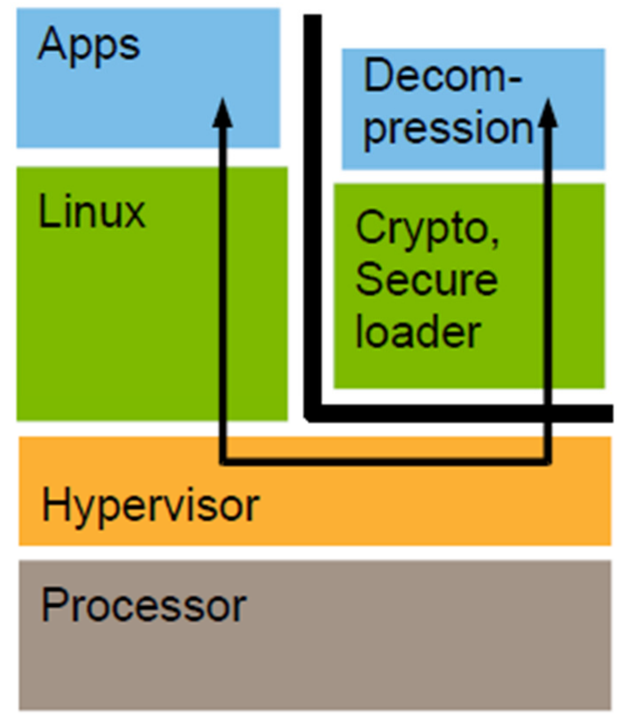


FORTH-ICS
Institute of Computer Science

IP protection

Use case 4a: 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



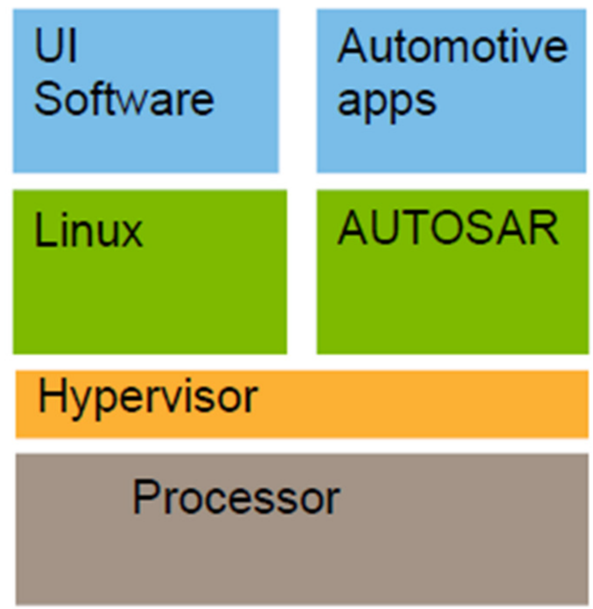
Processor consolidation: control + infotainment

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

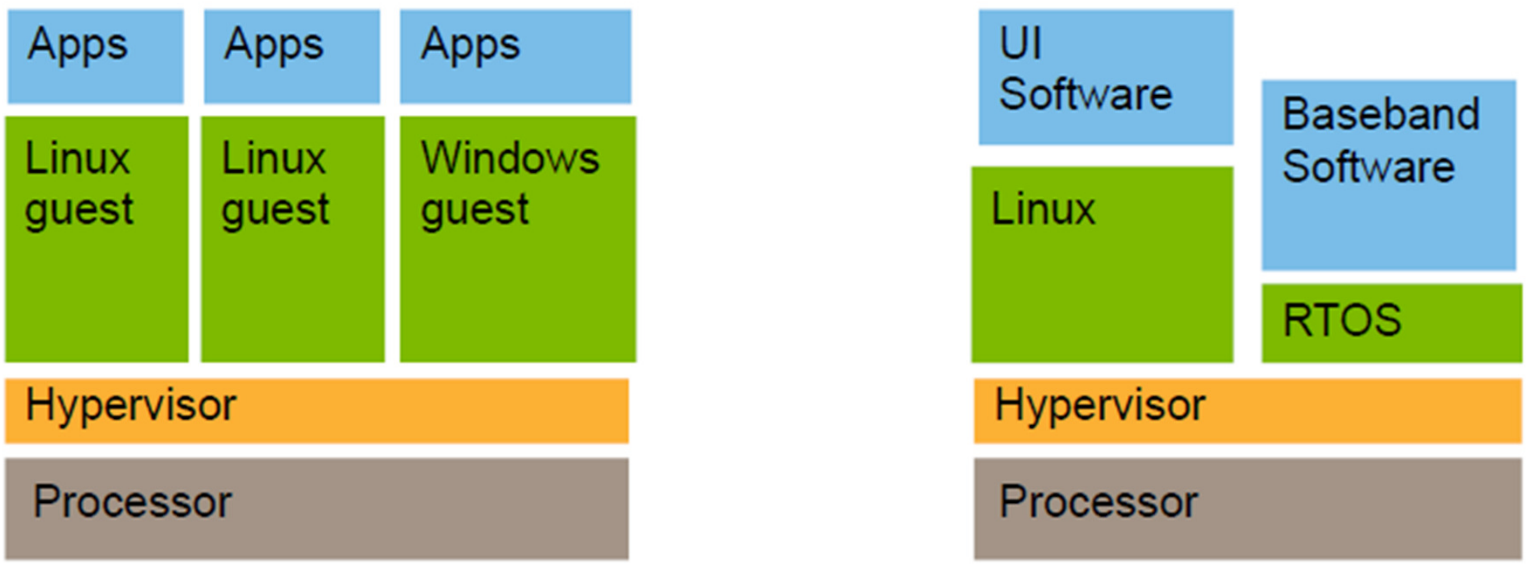
→ Increasing overlap of infotainment and control/convenience
- eg park-distance control using infotainment display
- benefits from being located on same CPU



Enterprise vs Embedded Systems VMs

Homogenous vs heterogenous guests

- Enterprise: many similar guests
  - hypervisor size irrelevant
  - VMs scheduled round-robin
- Embedded: 1 HLOS + 1 RTOS
  - hypervisor resource-constrained
  - interrupt latencies matter



Isolation vs Cooperation

Enterprise

→ Independent services
→ Emphasis on isolation
→ Inter-VM communication is secondary
  - performance secondary
→ VMs connected to Internet (and thus to each other)

Embedded

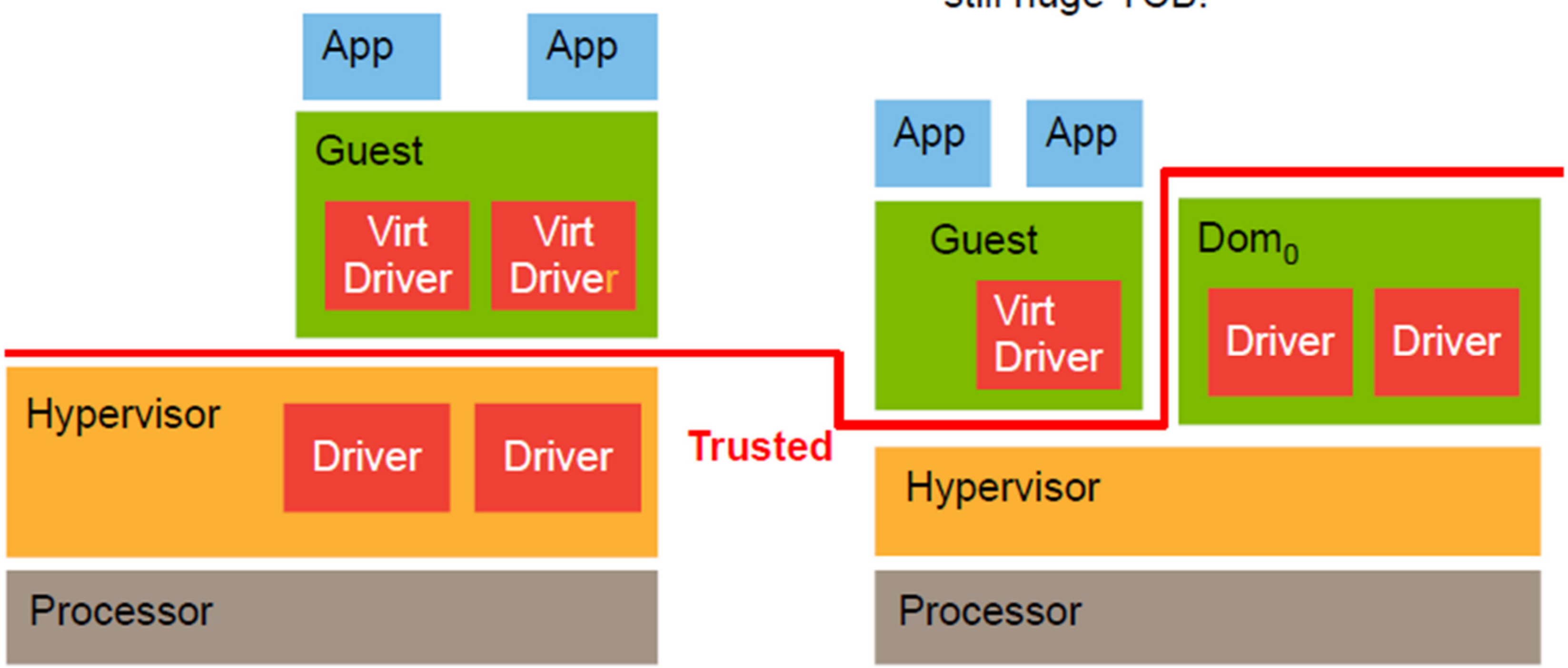
→ Integrated system
→ Cooperation with protection
→ Inter-VM communication is critically important
  - performance crucial
→ VMs are subsystems accessing shared (but restricted) resources

Isolation vs Cooperation : Scheduling

This image cannot currently be displayed.

Devices in enterprise virtual machines

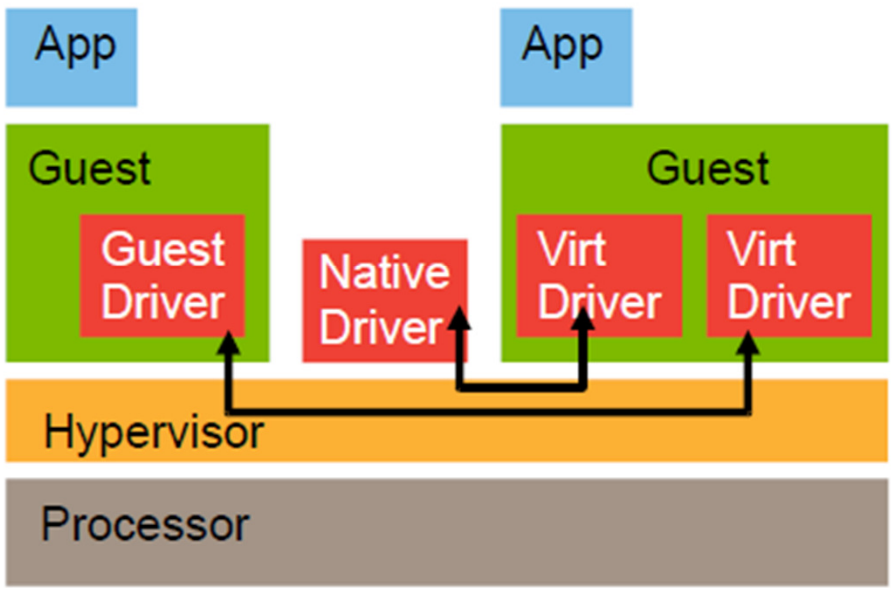
- → Hypervisor owns all devices
- → Drivers in hypervisor
  - need to port all drivers
  - huge TCB
- → Drivers in privileged guest OS
  - can leverage guest's driver support
  - need to trust driver OS
  - still huge TCB!



Devices in embedded virtual machines

- Some devices owned by particular VM
- Some devices shared
- Some devices too sensitive to trust any guest
- Driver OS too resource hungry

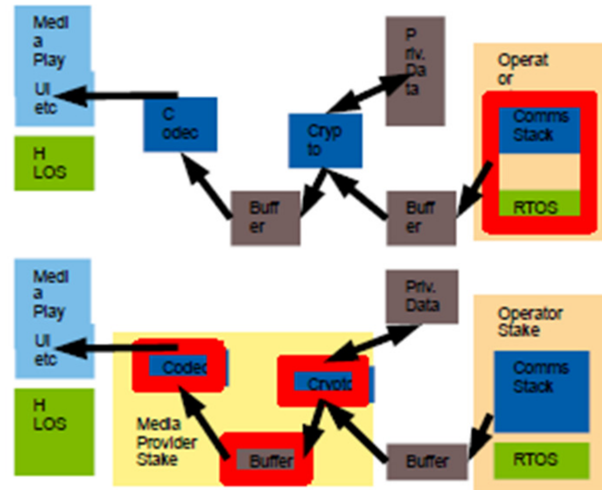
- Use isolated drivers
  - protected from other drivers
  - protected from guest OSes

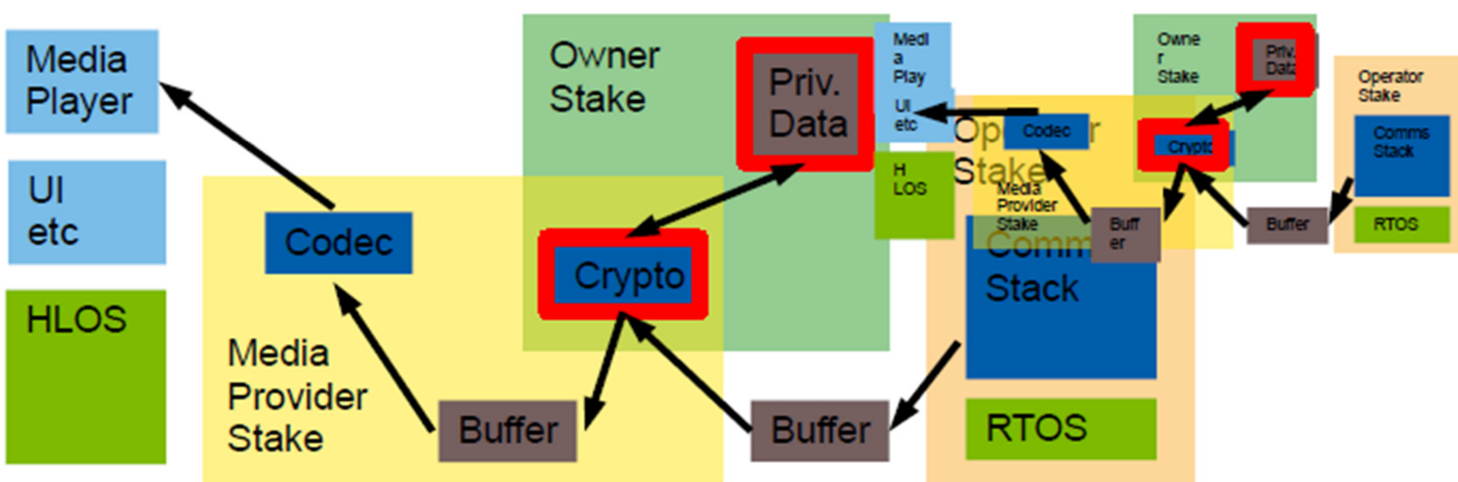


Inter-VM Communication

Modern embedded systems are multi-user devices!

- Eg a phone has three *classes* of “users”:
  - the network operator(s)
    - assets: cellular network
  - content providers
    - media content
  - the owner of the physical device
    - assets: private data, access keys



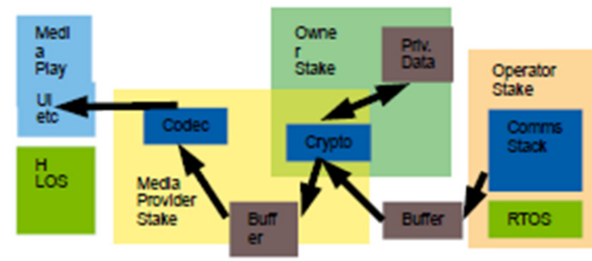


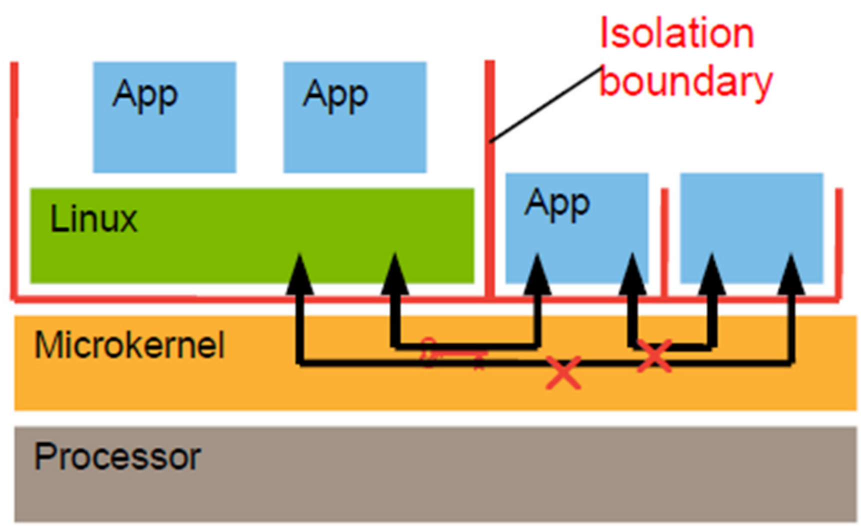
Inter-VM Communication

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

- → Different "users" are mutually distrusting
- → Need strong protection / information-flow control between them
- → Isolation boundaries ≠ VM boundaries
  - some are much smaller than VMs
    - individual buffers, programs
  - some contain VMs
  - some overlap VMs
- → Need to define information flow between isolation domains

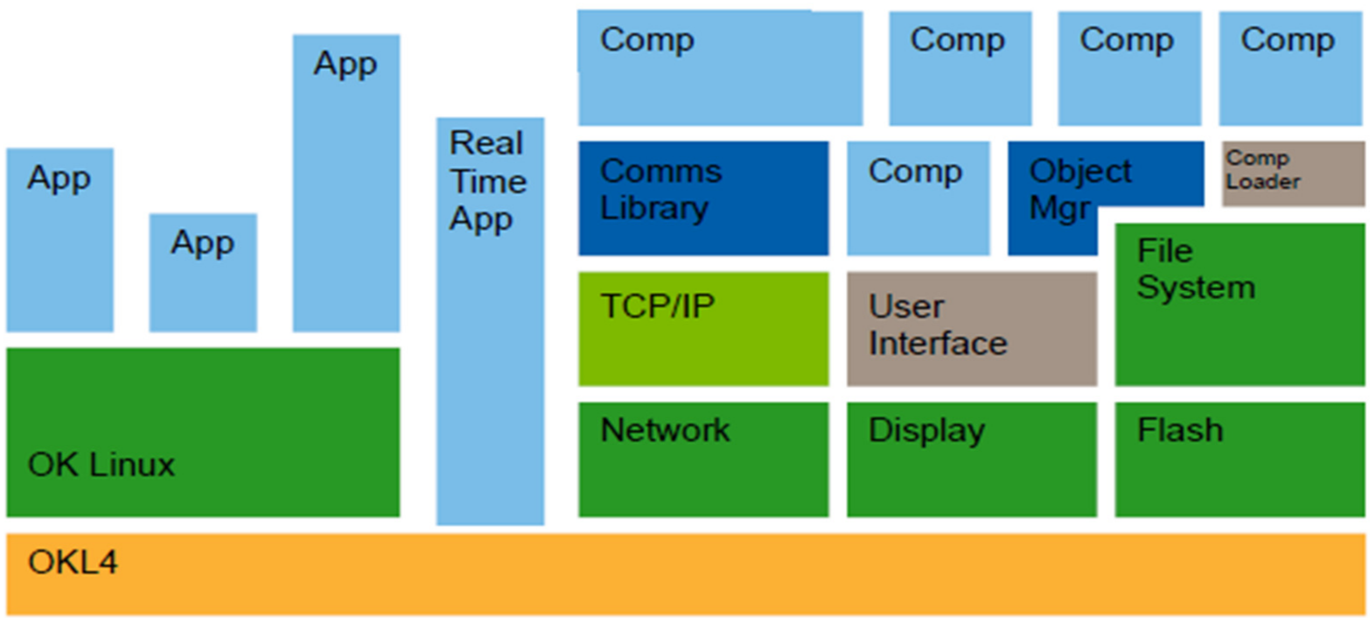




FORTH-ICS Institute of Computer Science

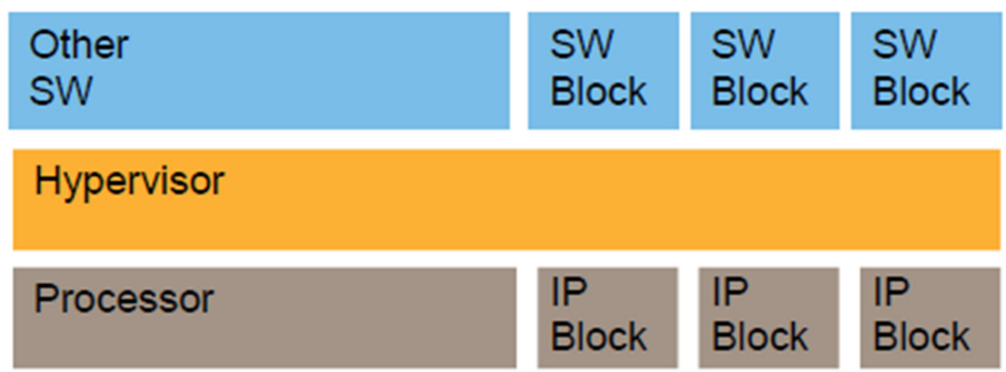
High safety & reliability requirements

- → Software complexity is mushrooming in embedded systems too
  - millions of lines of code
- → Some have very high safety or reliability requirements
- → Need divide-and-conquer approach to software reliability
  - Highly componentised systems to enable fault tolerance



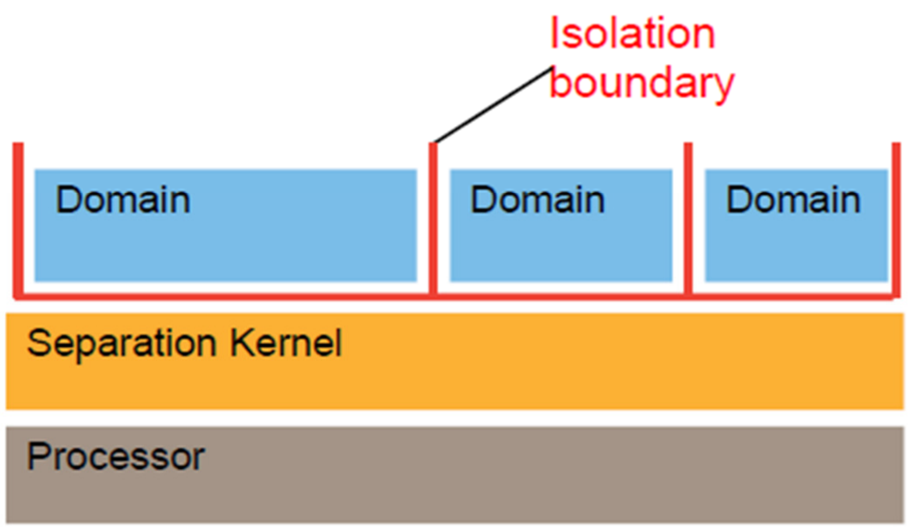
Componentization for IP Blocks

- → Match HW IP blocks with SW IP blocks
- → HW IP owner provides matching SW blocks
  - encapsulate SW to ensure correct operation
  - Stable interfaces despite changing HW/SW boundary



Componentization for Security

- *MILS architecture*: multiple independent levels of security
- Approach to making security verification of complex systems tractable
- *Separation kernel* provides strong security isolation between subsystems
- High-grade verification requires small components



ARM TrustZone

- ARM TrustZone extensions introduce:
  - new processor mode: *monitor*
    - similar to VT-x root mode
    - banked registers (PC, LR)
    - can run unmodified guest OS binary in non-monitor kernel mode
  - new privileged instruction: SMI
    - enters monitor mode
  - new processor status: *secure*
  - partitioning of resources
    - memory and devices marked secure or insecure
    - in secure mode, processor has access to all resources
    - in insecure mode, processor has access to insecure resources only
  - monitor switches world (secure ↔ insecure)
  - really only supports one virtual machine (guest in insecure mode)
    - need another hypervisor and para-virtualization for multiple guests