

(Secure-)System Architectures Rough Overview

Hermann Härtig



Outline

- Objectives
- Architectures, based on
 - safe languages
 - operating systems
 - hardware virtualization
 - micro-kernels
- use cases



Objectives: Security

- confidentiality
no unauthorized access to information
- integrity
no unauthorized, **unnoticed** modification of information
- recoverability
no permanent damage to information
- availability
timeliness of service



Integrity: 2 Common Definitions

- Definition 1:
 - Either information is current, correct, and complete
 - Or it is possible to detect that these properties do not hold
- Definition 2:
 - No damage to information
 - Integrity violation:
 - Detect
 - prevent



Objectives: System Security

- Secure and unsecure applications
- Compatibility:
 - Legacy (insecure) applications
 - Legacy OSes, Hardware drivers
- Flexible sandboxing
- Resource Control
- Simplicity, small trusted computing base



Trusted Computing Base

All parts of a system (hardware and software) that must be relied upon to properly enforce a security policy

TCBs should

- be extremely carefully engineered and
- be small



So far ... and later

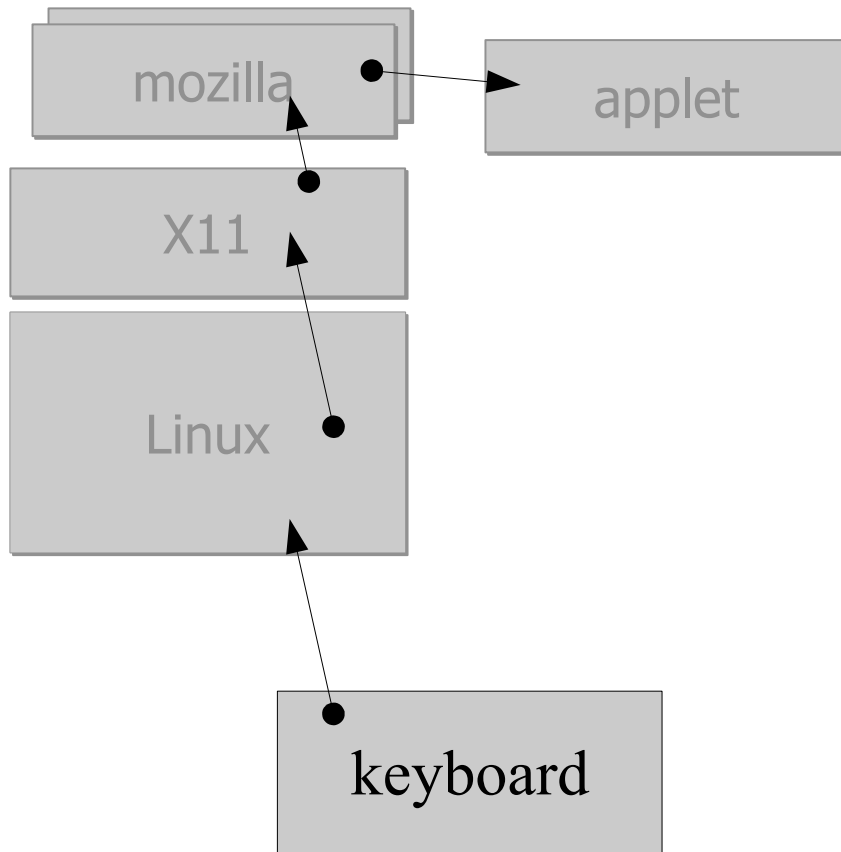
- Security Objectives:
Confidentiality, Integrity, Recoverability,
Availability, ...
- Authenticated Booting, Remote Attestation and
Sealed Memory
- Security Policies and Models:
Multilevel, Chinese Wall, ...
- Security mechanisms:
access control lists, capabilities,
(later: firewalls, network security), ...
- Threats: buffer overflow, covert channels, ...



Key System Property

- effective separation (partitioning)
- mediated communication
- small trusted computing base

Your password(s), credit card number, ...



see:

Understanding Data Lifetime
via Whole System Simulation
Jim Chow, Ben Pfaff, Tal
Garfinkel, Kevin Christopher,
and Mendel Rosenblum,
Stanford University
Usenix Security 04



Safe languages

- All applications are written in a “safe” language.
- Mechanisms are enforced by compiler and/or interpreter.



Examples for Language-based architectures

- Java-based systems
typesicherer Speicher
only “Byte Code” is allowed
JVM enforces
- Burroughs B 7700
all applications written in “Burroughs
extended ALGOL”, OS in ESPOL
only binary programs produced by BEA
compiler are executed
enforced by the OS
- MS Singularity



Properties of Language-based architectures

- closed systems, only one language
- sometimes “non-safe” languages are preferable
(e.g., device drivers, speed, ...)
- how to determine whether or not some binary program was produced from any program in a particular, safe language (Java: “Byte Code verifiers”) ?

- very common, e.g. in telephones



OS-based Separation

Three Variants

- see one common OS instance:
use processes and existing mechanisms (ACLs, ...) to establish mediation and mediated communication
add more elaborate mechanisms (SE-Linux)
- see “their own” instance of the OS:
runs the OS at user level (user mode Linux)
provide completely separate machines (for example: separate Linux machines)
- add another abstraction (zones, jails, containers, ...)



Properties of all variants

- homogeneous
- base OS (for example Linux) part of the TCB



additional abstraction: zones, jails, ...

- separate name space per user: “chroot”
- restrict communication to within one partition
- bind resources (IP addresses) to one zone
- separate kernel resources (process table, /proc, ...)
- separate “root user”
- partition resources (Memory, CPU, devices, ...)



Examples & use cases

one OS: ... common case

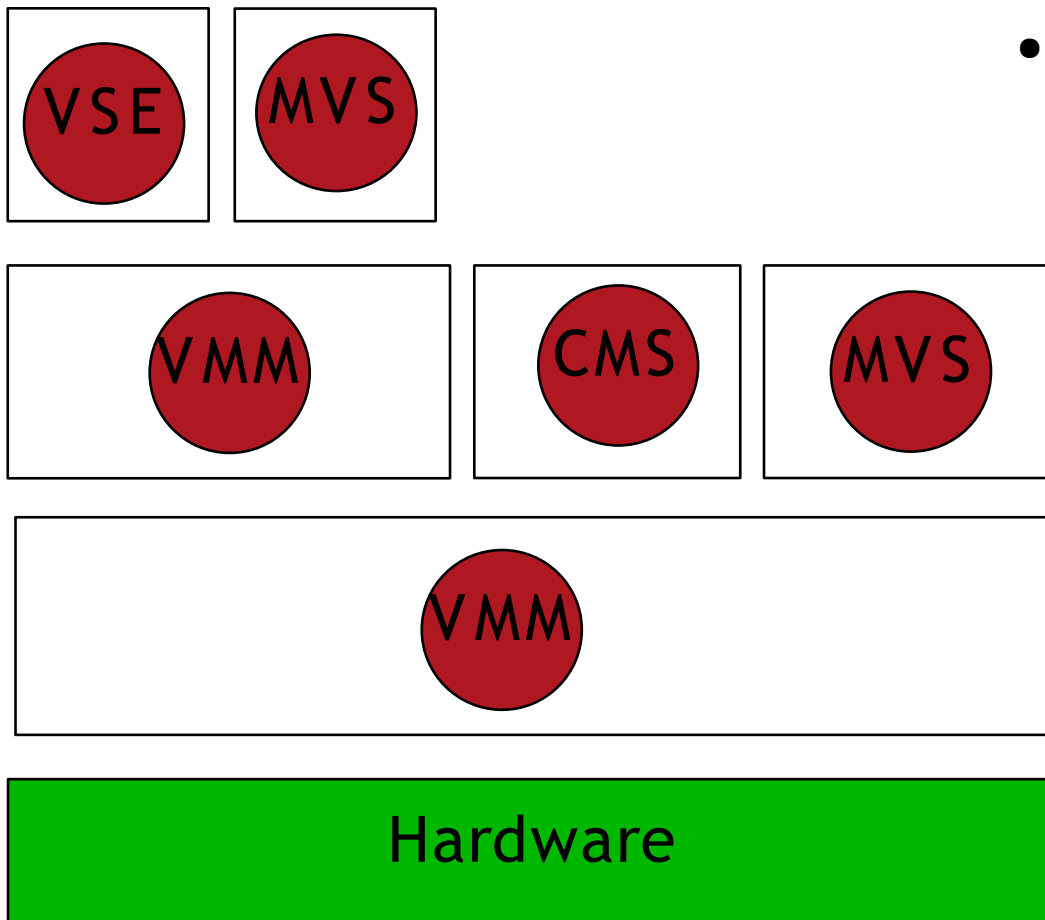
additional abstraction:

- examples:
 - Sun Solaris Zones
 - FreeBSD Jails
 - ...
- use cases:
 - server isolation

OS virtualization: User Mode Linux, use case ?

Hardware Virtualization (VM Monitors)

- VM emulate physical machines
- **different** legacy OSes can be used





Hardware Virtualization

- Users see their own full hardware system
- can use (within limits: any) legacy OS
- a VMM (Virtual Machine Monitor) provides virtual CPU, Memory, devices, ...



Hardware architectures and VM

- hardware architectures provide privilege levels to separate trusted (OS) and untrusted SW(applications)
- some *sensitive* instructions must be available to trusted software exclusively
- untrusted software ideally raises exceptions if sensitive instructions are executed
- however, some architectures overload sensitive instructions(different semantics)
- e.g., popf in X86



The three variants of “Hardware Virtualization”

- “faithful” or “full” virtualization:
 - Emulation (Qemu), slow
 - hardware support needed (at least for X86)
- binary patching:
before loading or at run time:
patch critical instructions in used OS
- “Para-Virtualization”:
change legacy OS at source level



Examples for Virtual Machines

- VM for IBM 360, ...
full hardware architecture support
- VAX VM
- Virtual-Box, VMWare, Connectix, ...
VMM needs to locate and replace sensitive instructions
- Terra (Stanford) (?)
- Xen (“Paravirtualization”)
operating systems need to be modified by hand
- intel Vanderpool / AMD Pacifica

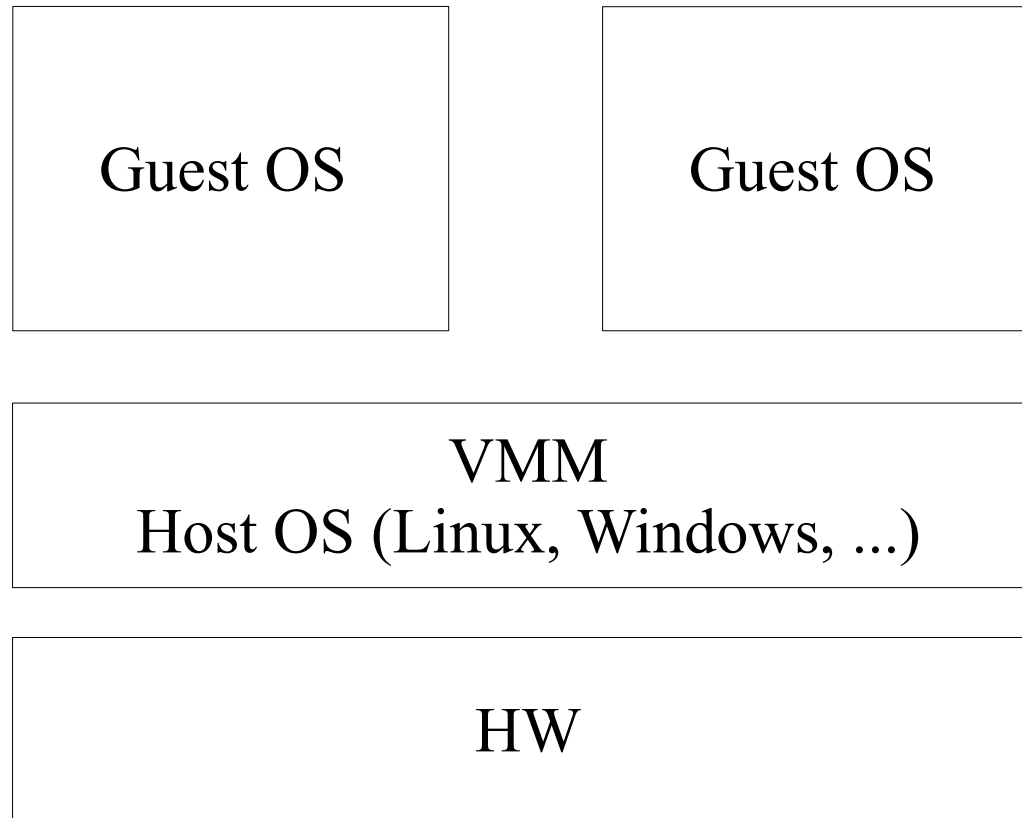


Properties of Virtual Machines

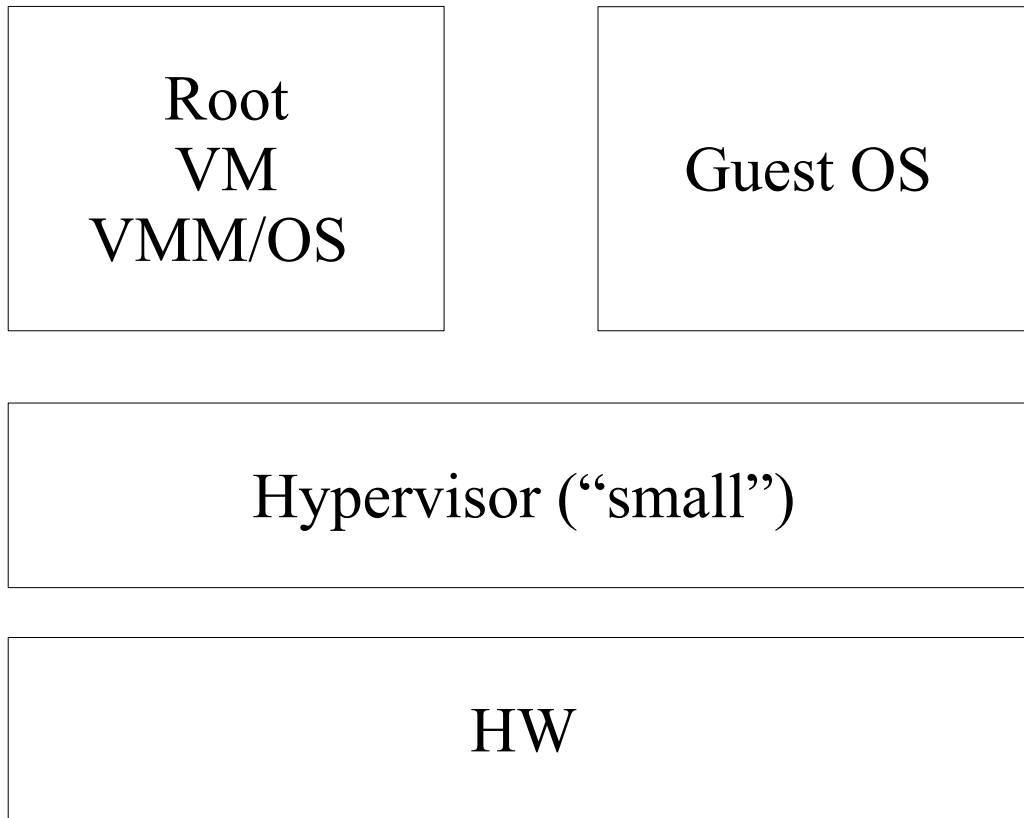
- VMM mostly part of TCB (can be large)
- complete separation,
communication by devices and drivers
large interfaces
sometimes specific communication channels
- two architectures ...



The two architectures: Hosted VMM and ..



The two architectures: ... Hypervisor-based



- hypervisor:
enforces separation
provides basic
mechanisms
- Root domain:
contains largest part
controls whole
machine



Server use cases

- “server consolidation” (very successful !!!)
 - many virtual servers on one physical machine
 - “as secure as separate physical machines”
- data centers:
migrate VM from one physical location to another

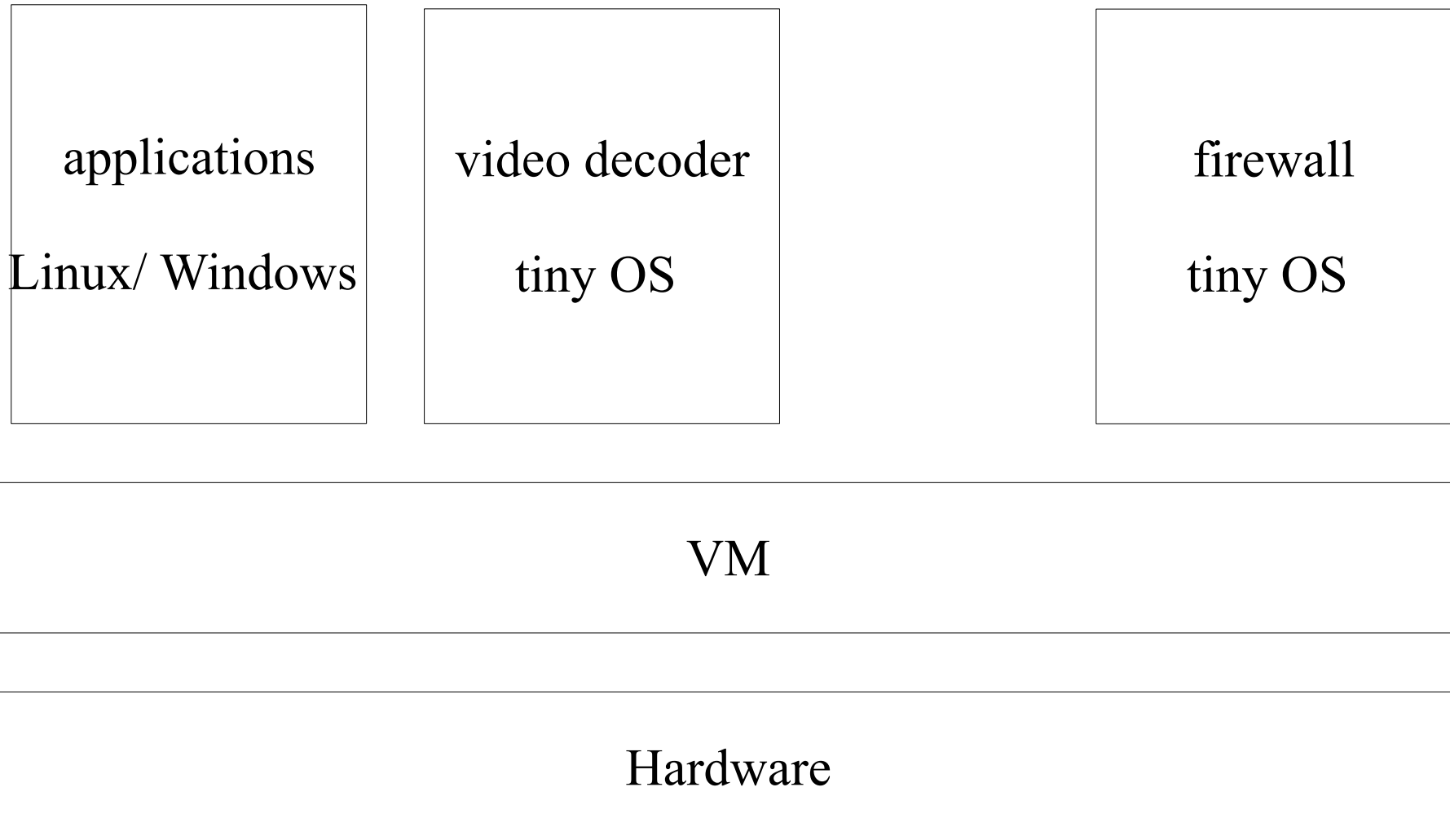


Desktop/Laptop use cases

- different trust level on your desktop
 - rubbish machine for use of internet
 - firewall on separate VM
 - game VM
 - Media machine
- “my old PC”

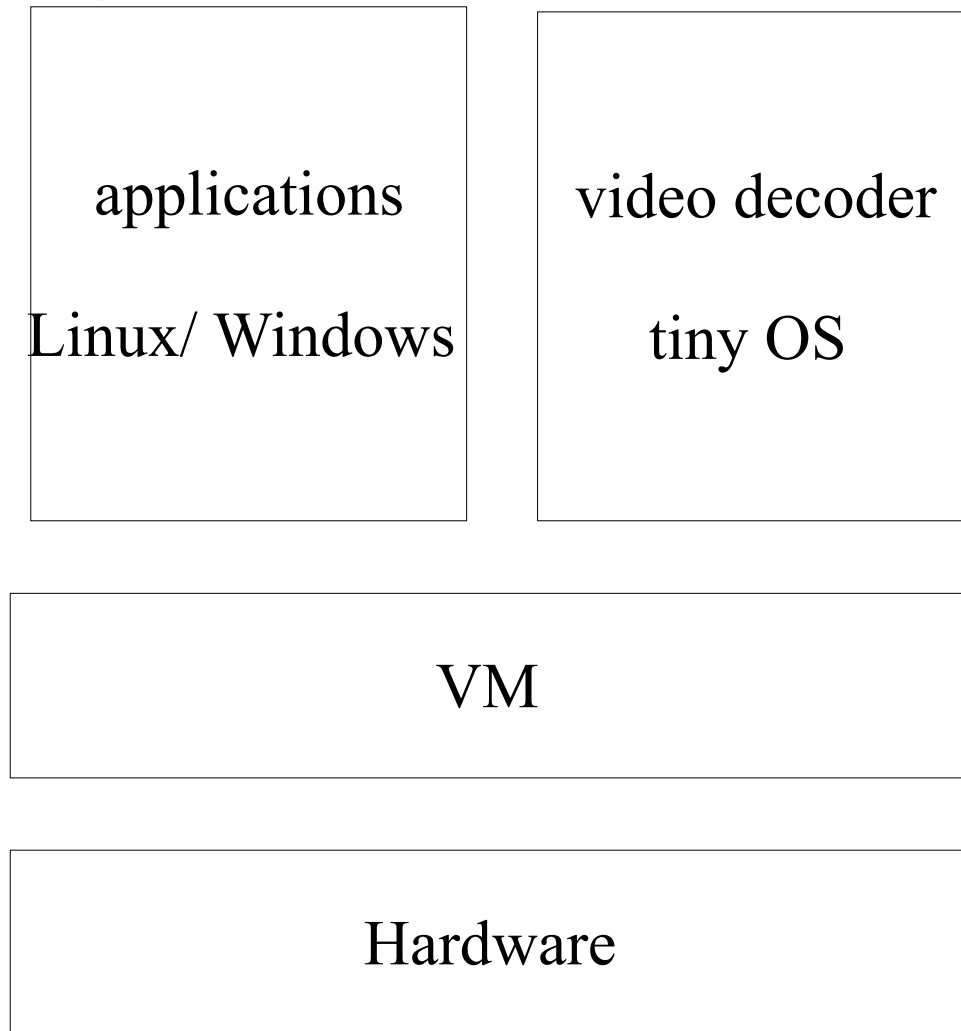


Small special purpose operating systems





VM and Authenticated Booting



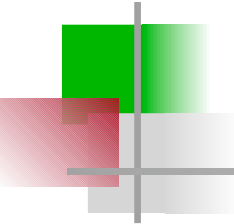
- company provides decoder keys only for (HW, VM, tiny OS, mediaplayer)
- TPM/Sealed memory delivers keys only to ...
- counter in TPM allows to play exactly n times



Micro-kernel-based architectures

Principles:

- small kernel with minimal functionality
- all other functionality provided by components/servers running at user level and encapsulated by address spaces
- reuse legacy OS using (para-)virtualization



(Expected) Properties

- Robustness
crashes in components (drivers) do not crash the whole system)
- Security:
smaller TCB
TCB application specific
- Performance:
slightly slower (due to more context switches)



Examples for micro-kernel-based architectures

- L4: Fiasco, Pistachio, OKL4, Nova, ...
- Pike OS (SysGo, early derivative of L4)
- Integrity OS (Green Hills)
- EXO kernel
- Perseus (Bochum, also based on L4)
- EROS (John Hopkins University)
- Microsoft NGSCB (stopped)
- (Trusted) MACH (until 97)
- MANY research projects



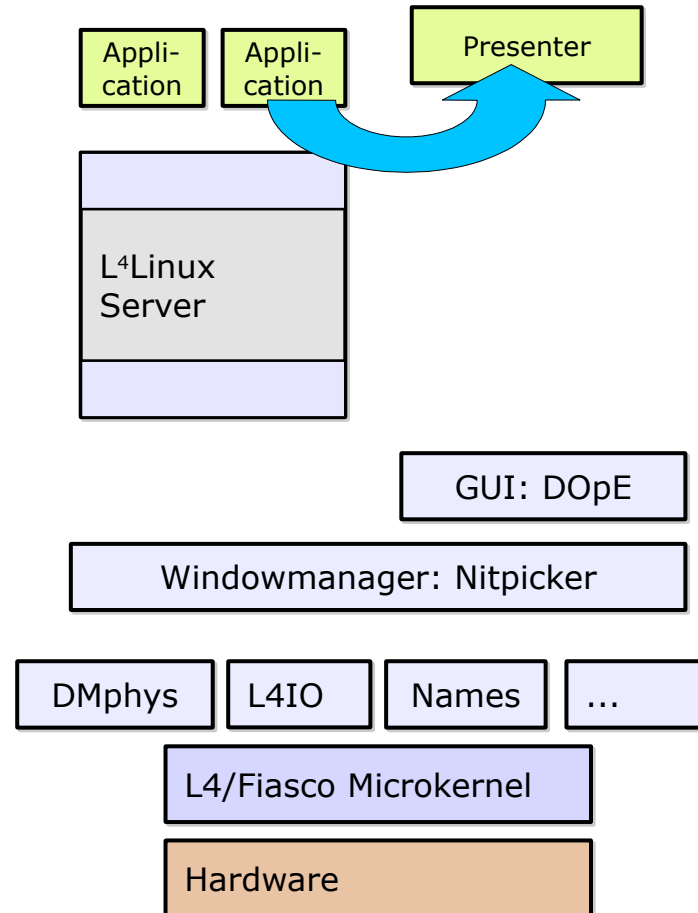
L4 Kernel functionality

kernel provides only inevitable mechanisms
no policies enforced by the kernel

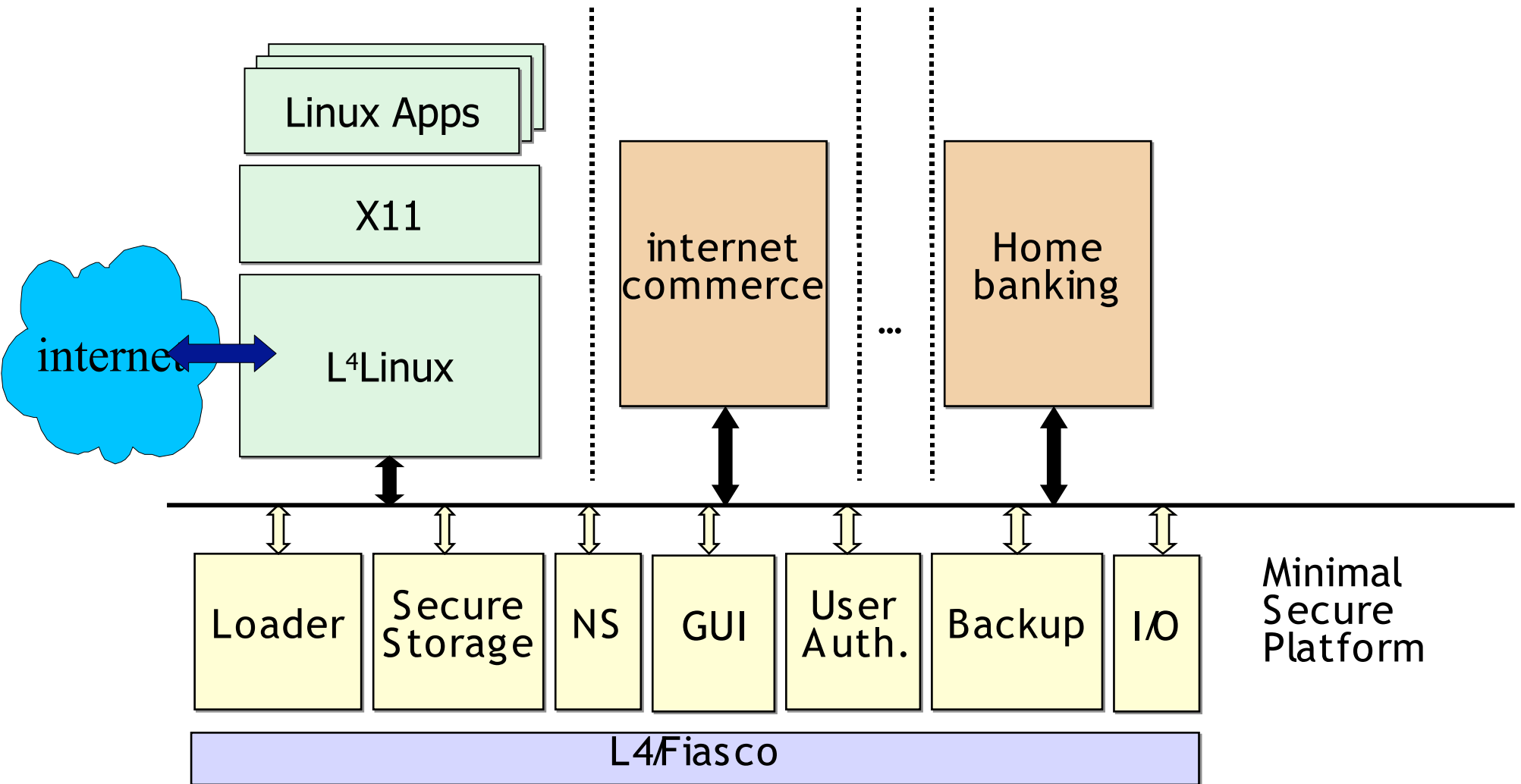
what is inevitable?

- address spaces
- threads & scheduling
- inter-process communication (IPC)

An example configuration



An example configuration





Papers to read

- Terra: A Virtual Machine-Based Platform for Trusted Computing
Garfinkel et al.
SOSP 03
ACM
- Härtig, Hohmuth, Feske, Helmuth, Lackorzynski, Mehnert, Peter:
The Nizza Secure-System Architecture.
International Conference on Collaborative Computing: Networking, Applications and Worksharing (our Webpage)



More on L4 at TU Dresden

- Lectures
 - Micro-kernel-Based Operating Systems
 - Micro-Kernel Construction
- Komplexpraktikum