TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science Institute of System Architecture, Operating Systems Group
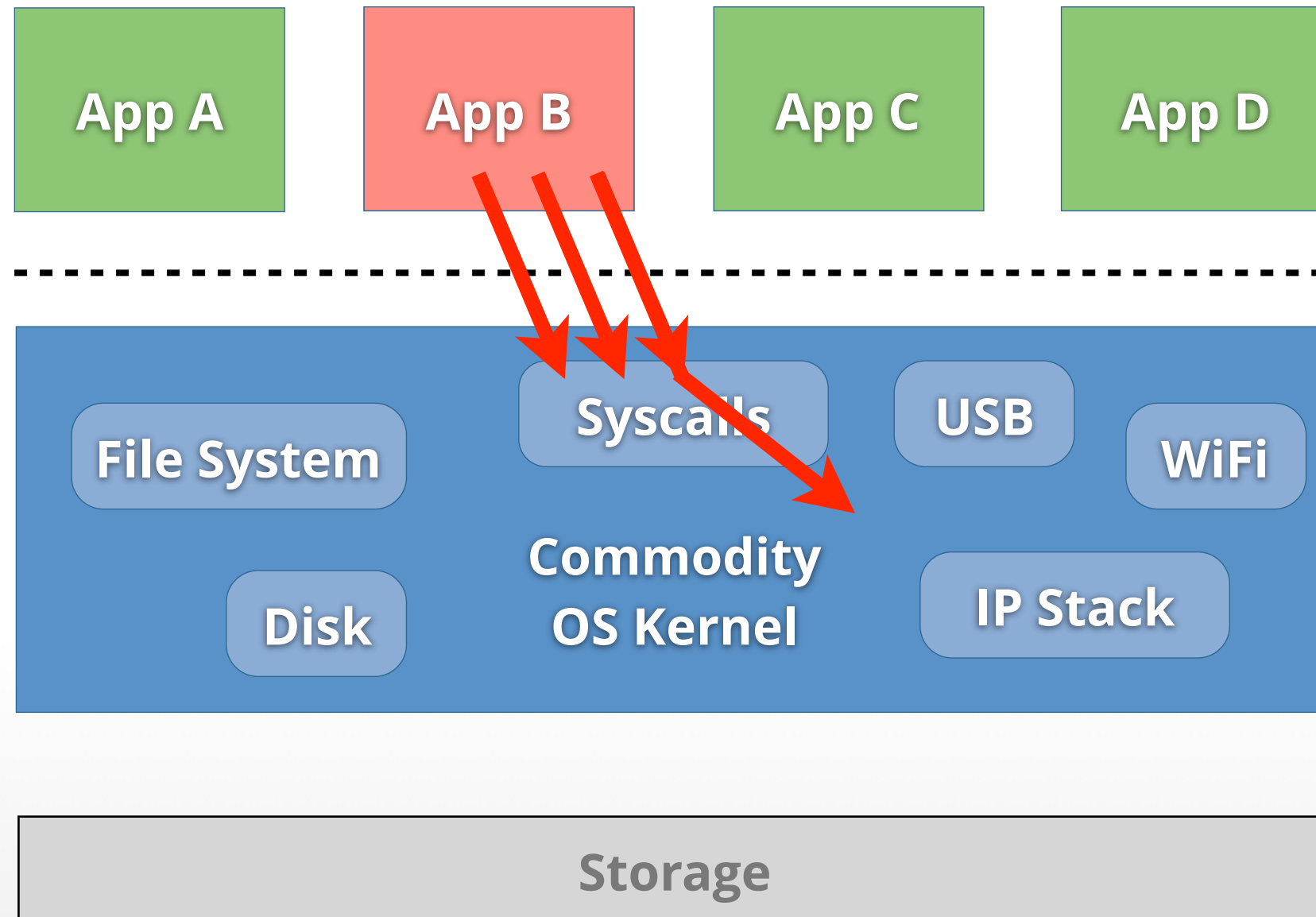
SECURITY ARCHITECTURES
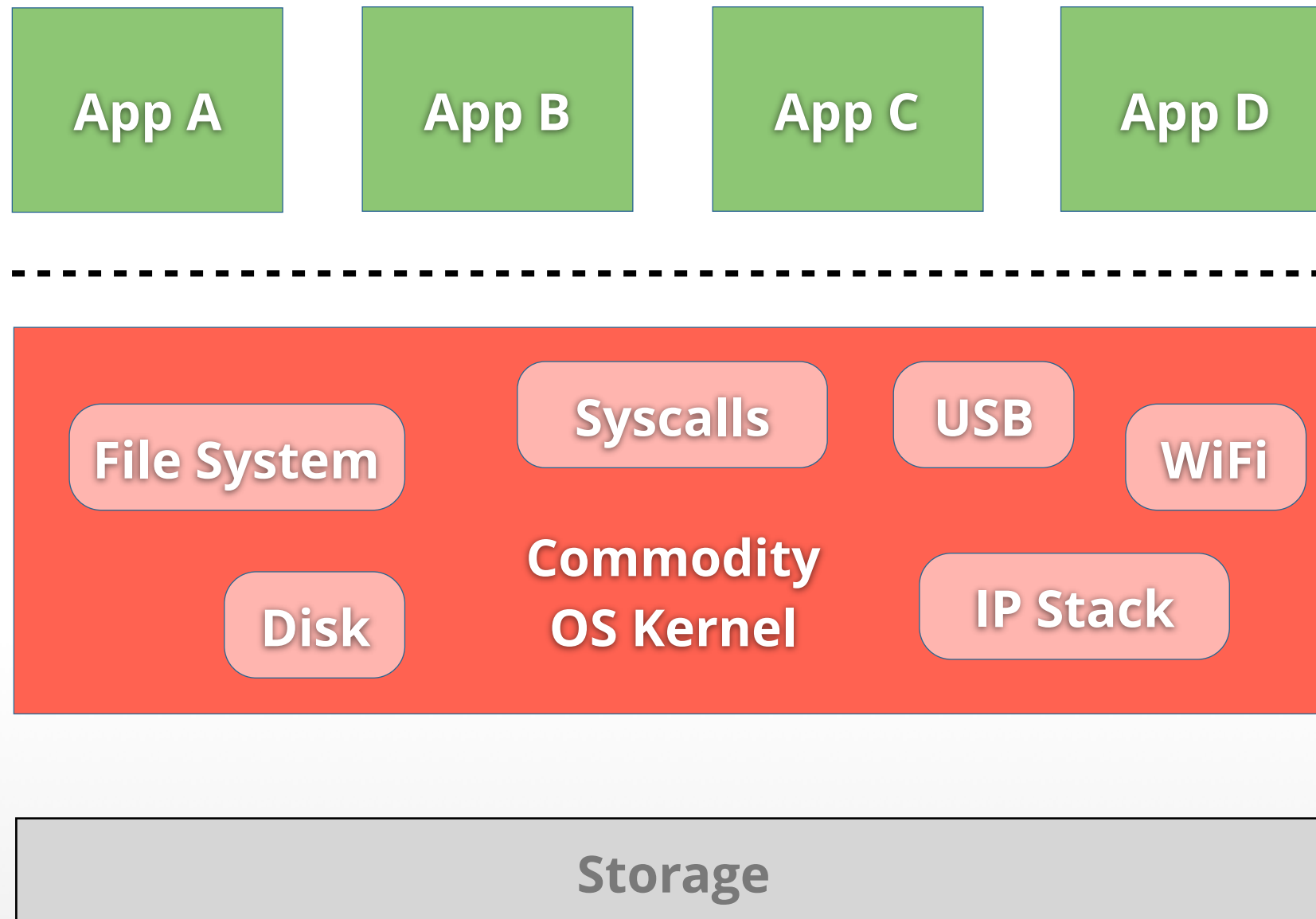
CARSTEN WEINHOLD

CLASSICAL ARCHITECTURES

- Isolation in commodity OSes:

  - Based on user accounts

  - Same privileges for all apps

  - No isolation within applications

  - Permissive interfaces (e.g., ptrace to manipulate other address spaces)

KERNEL ATTACK VECTOR

App A

App B

App C

App D

File System

Syscalls

USB

WiFi

Commodity
OS Kernel

Disk

IP Stack

Storage

- Isolation in commodity OSes:

  - Based on user accounts

  - Same privileges for all apps

  - No isolation within applications

  - Permissive interfaces (e.g., ptrace to manipulate other address spaces)
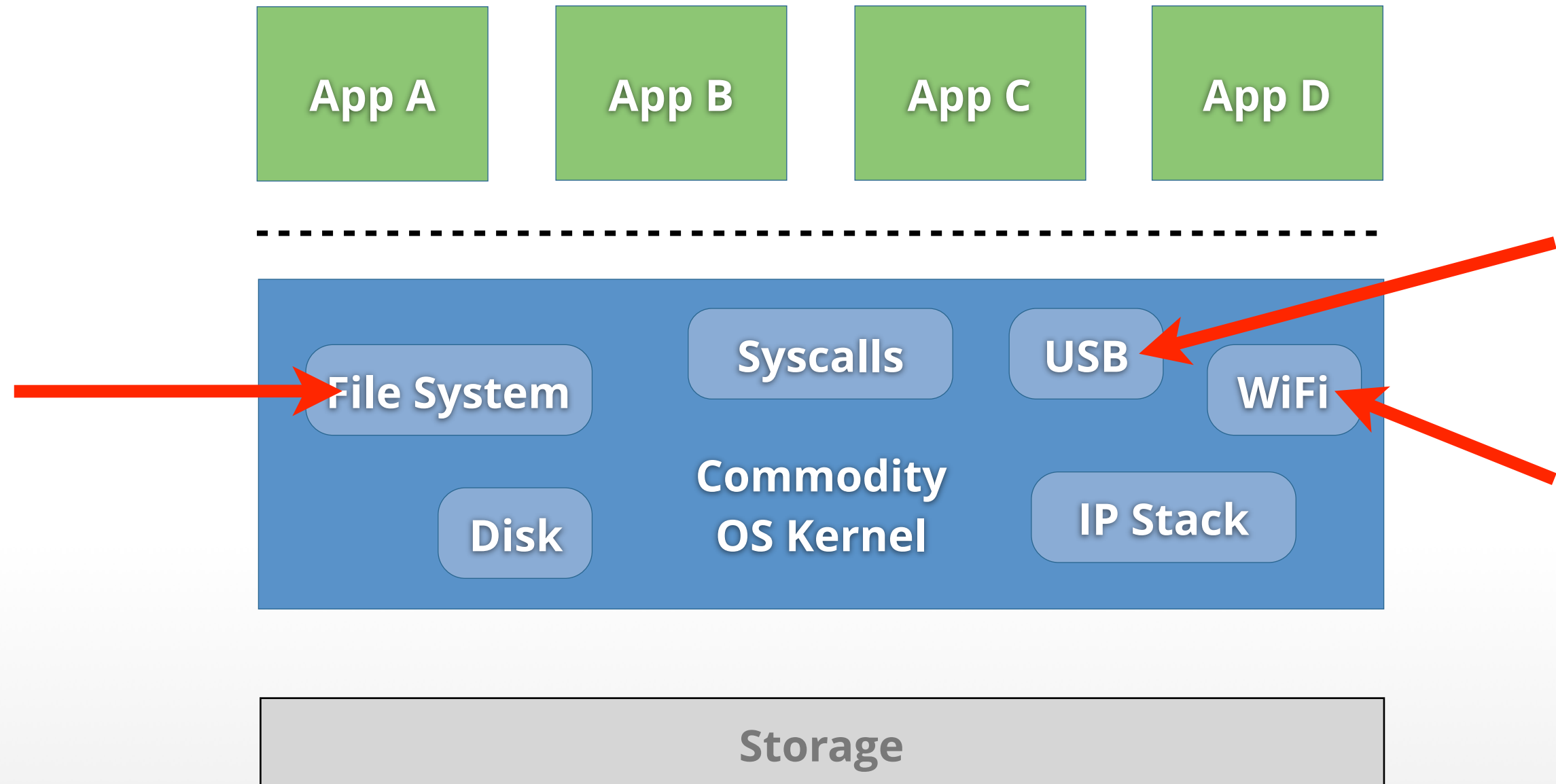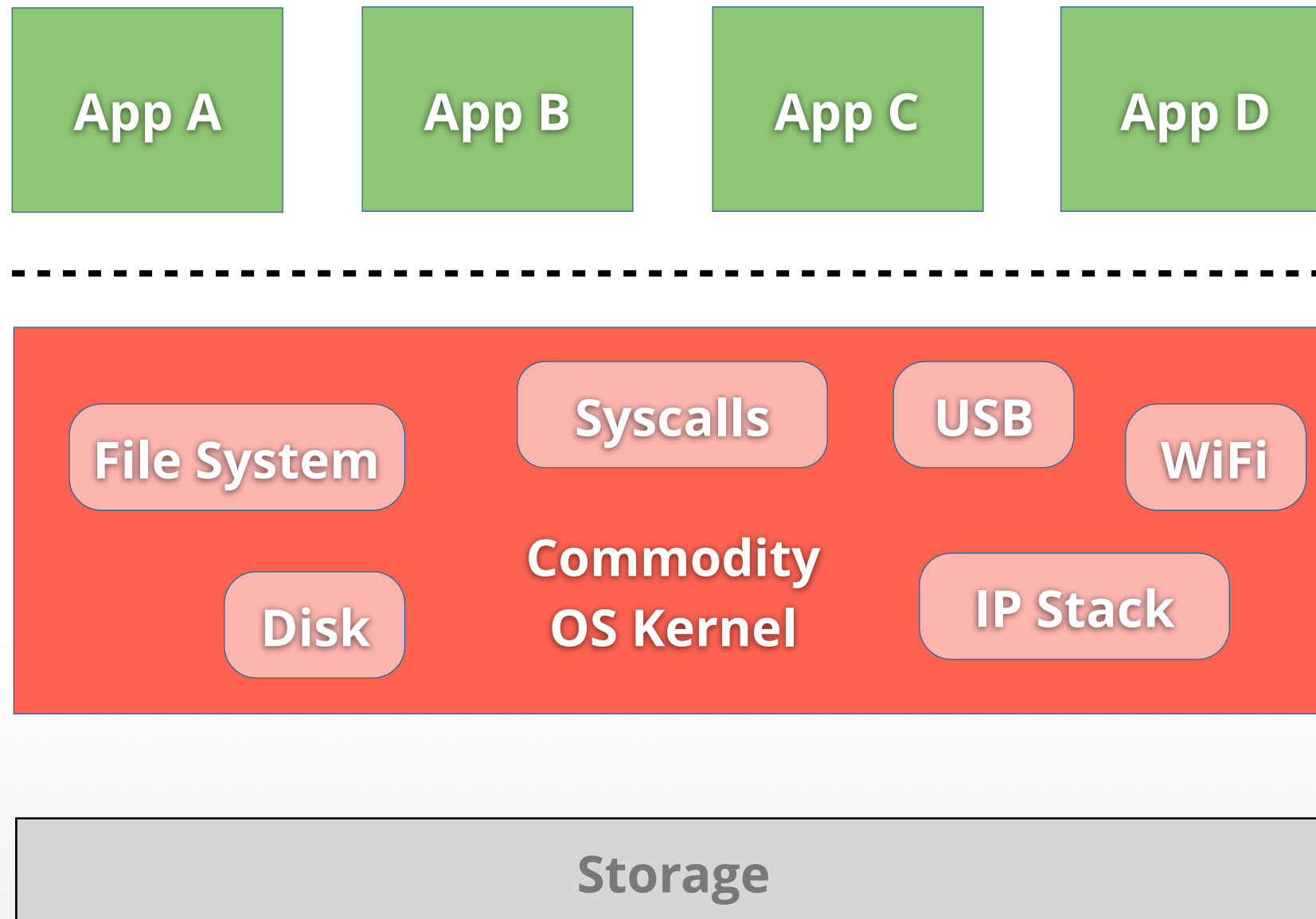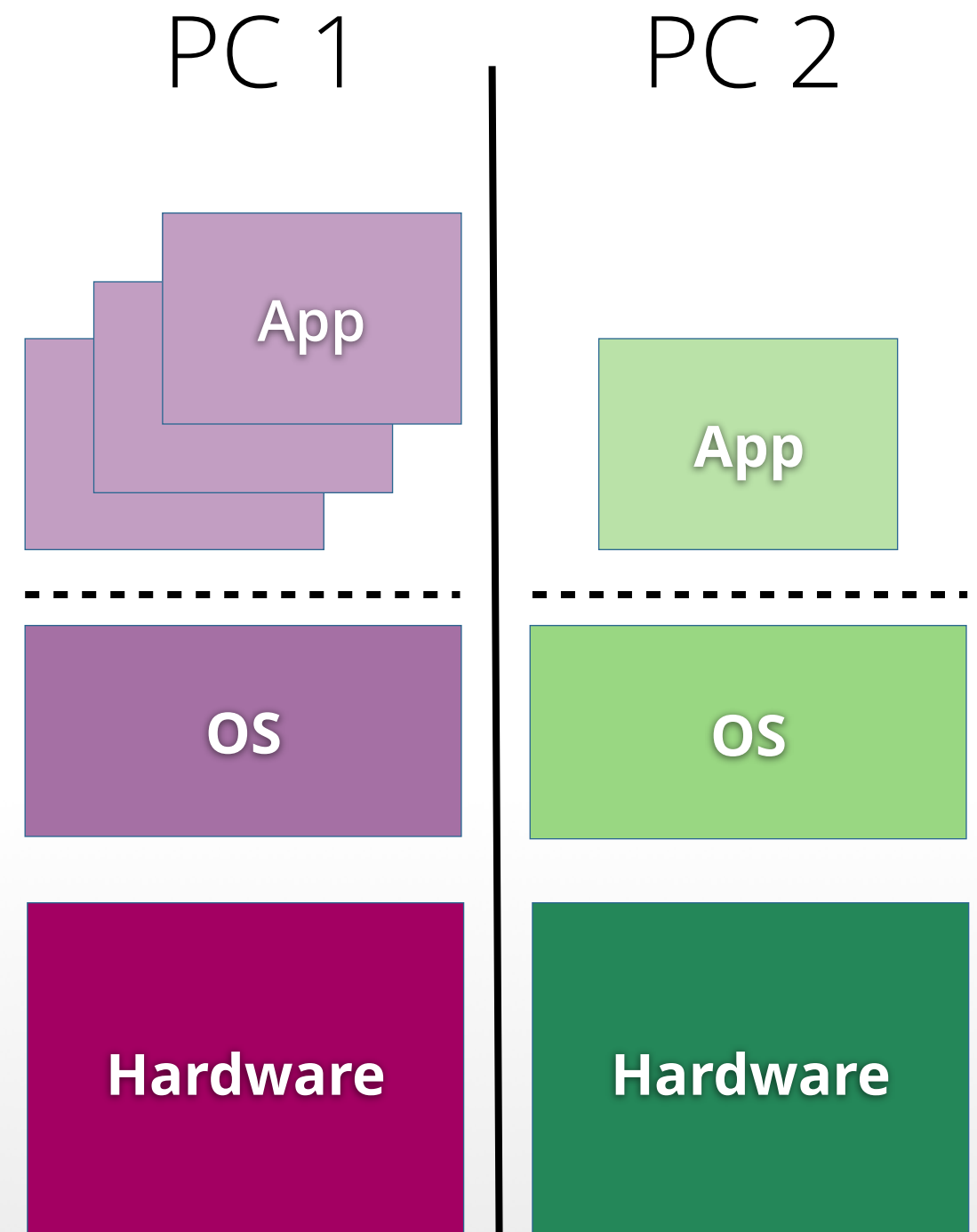
- Efforts to restrict privileges:

  - SELinux, AppArmor, Seatbelt, ...

- Separate computers
- Applications and data physically isolated
- Effective, but …
  - Higher costs
  - Needs more space
  - Inconvenient
  - Exposed to network

- Multiple VMs, OSes
- Isolation enforced by virtualization layer
- Saves space, energy, maintenance effort
- But still ...
  - Switching between VMs is inconvenient
  - Even more code

VM 1 | VM 2

App

App

OS | OS

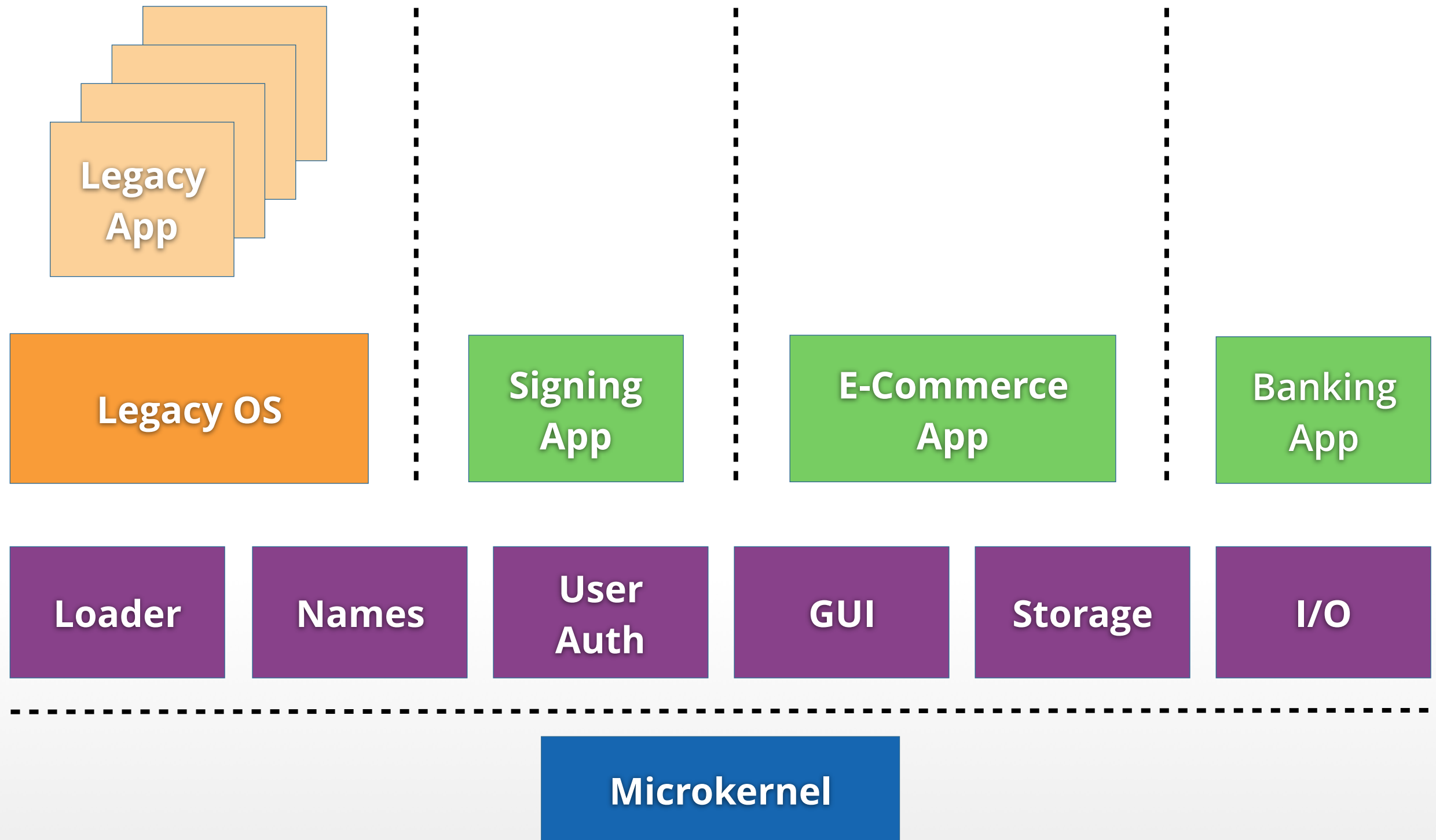**Virtualization Layer**

Hardware
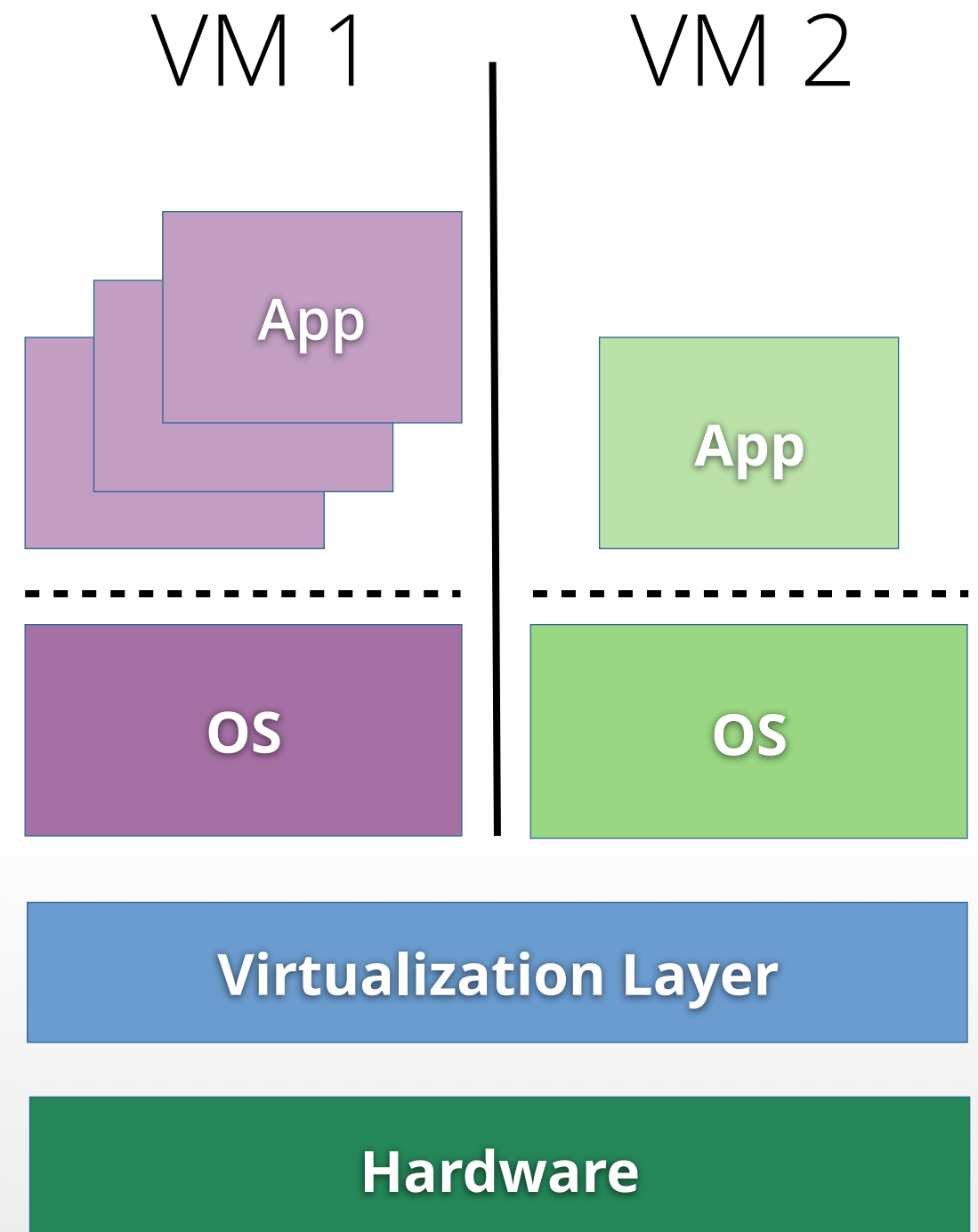
- Huge code bases remain

- Applications still the same

- Many targets to attack:

  - Applications, libraries, commodity OSes

  - Virus scanner, firewall, …

  - Virtualization layer

- High overhead for many VMs

SECURITY ARCHITECTURES

- Protect the user's data

- Secure applications that process data

- Acknowledge different kinds of trust, e.g.:

  - Application **A** trusted to handle its own data, but not the files of application **B**

  - OS trusted to _store_ data, but not to _see_ it

- Identify and secure **TCB:** the **T**rusted **C**omputing **B**ase

- To improve security: Reduce size of TCB = smaller attack surface

- First (incomplete) idea:

  - Remove huge legacy OS from TCB

  - Port application to microkernel-based multi-server OS

  - Remove unneeded libc backends, etc.

  - Possible approaches discussed in lecture on „Legacy Reuse"
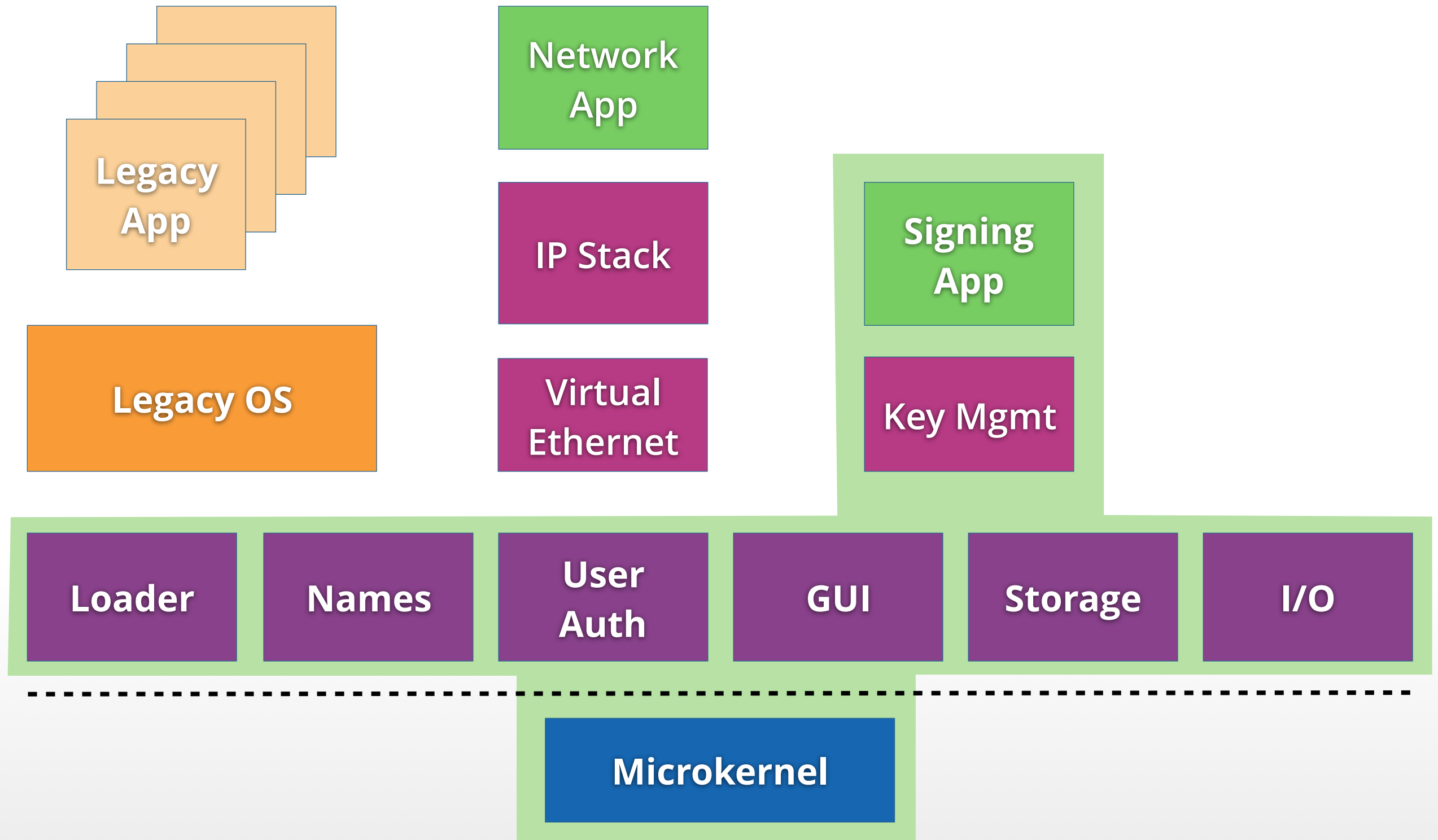
Nizza architecture: fundamental concepts:

- Strong isolation

- Application-specific TCBs

- Legacy reuse

- Trusted wrappers

- Trusted computing

- Reflects **Principle of Least Privilege**

- TCB of an application includes only components its security relies upon

- TCB does not include unrelated applications, services, libraries

- Reflects **Principle of Least Privilege**

- TCB of an application includes only components its security relies upon

- TCB does not include unrelated applications, services, libraries

- Mechanisms:

  - Address spaces + IPC control for isolation

  - Well-defined interfaces

SPLITTING COMPONENTS

- Problems with porting applications:

    - Dependencies need to be satisfied

    - Can be complex, require lots of code

    - Stripped down applications may lack functionality / usability

- Better idea: split application

    - Make only security-critical parts run on microkernel-based OS

    - Parts of application removed from TCB

Digitally signed e-mails, what's critical?

- Handling of signature keys

- Requesting passphrase to unlock signature key

- Presenting e-mail message:

  - Before sending: „**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **S**ign"

  - After receiving: verify signature, identify sender

Mozilla Thunderbird

GnuPG

X Window System
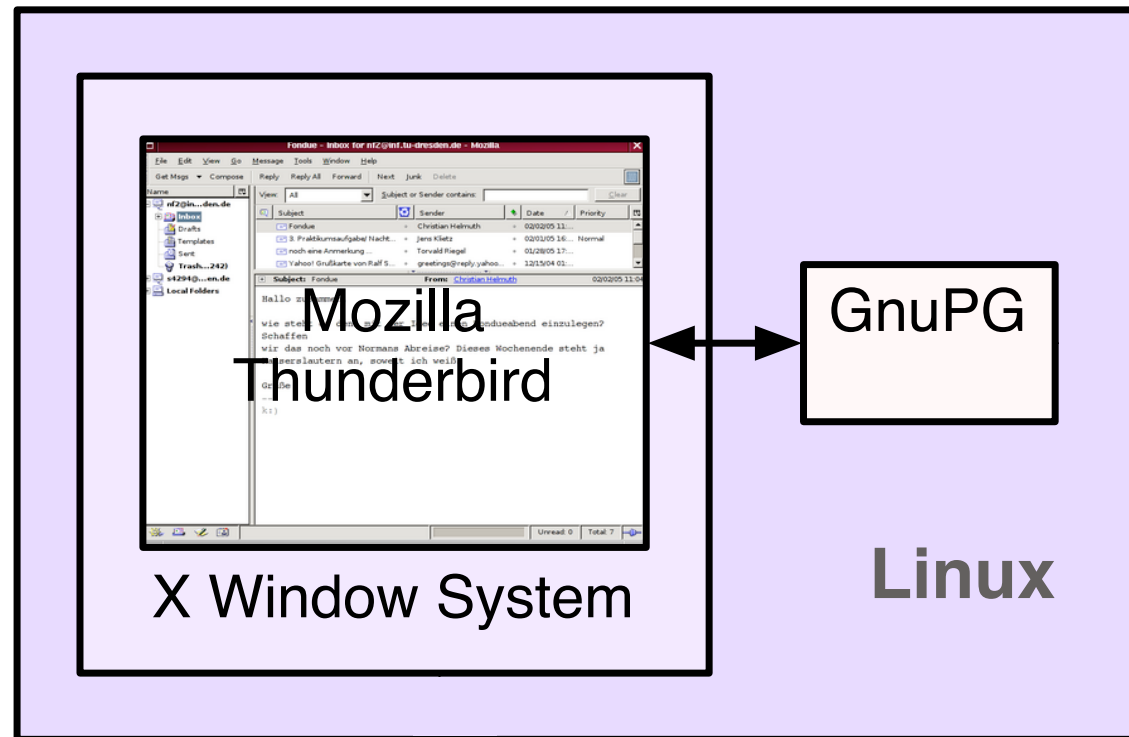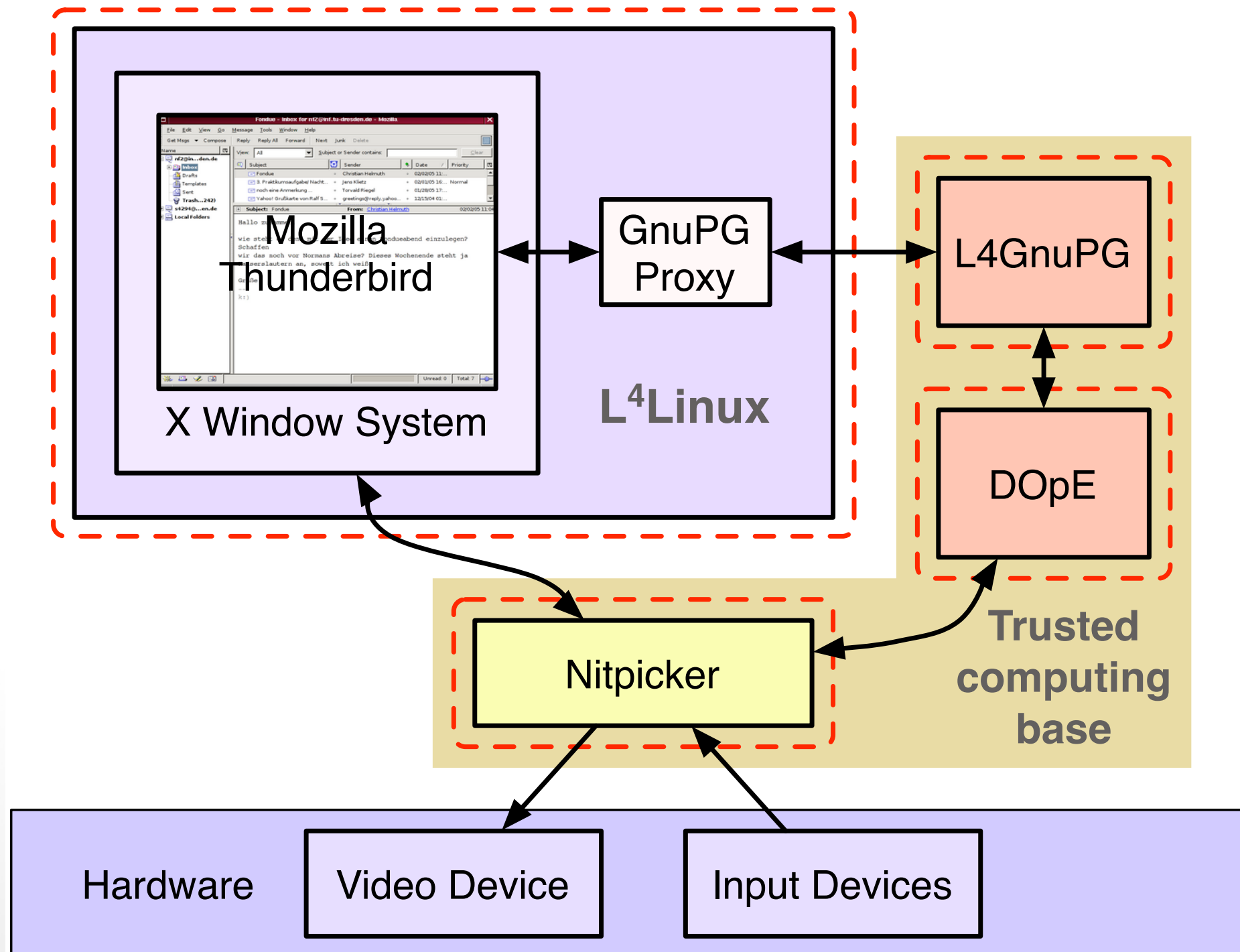
Linux

Image source: [5]

- *1,500,000*+ SLOC no longer in TCB:
  - Linux kernel, drivers, X-Server
  - C and GUI libraries, Thunderbird, …
- TCB size reduced to ~*150,000* SLOC:
  - GNU Privacy Guard, e-mail viewer
  - Basic L4 system
- At least 10 times less code in TCB

- Splitting works for applications

- What about the complex and useful infrastructure of commodity OSes?

  - Drivers (see previous lectures)

  - Protocol stacks (e.g., TCP/IP)

  - File systems

- Starting point: Virtualized commodity OS

- Run legacy OS in VM

- Reuse service: net, files, …

- Legacy infrastructure isolated from applications

- But:

  - Applications still depend on legacy services … in TCB?

  - Interfaces reused, security issues as well?

App

Legacy OS

Basic Services

Microkernel

- Network and file system stacks are virtually essential subsystems

- Generally well tested

- Ready for production use

- ... but not bug free [1,2]:

  - Linux file systems (UFS, ISO 9660, Ext3, SquashFS, ...): bug hunt of just 1 month yielded 14 exploitable flaws

  - WiFi drivers: remotely exploitable bugs

- Complex protocol stacks should not be part of TCB (for confidentiality + integrity)

- Reuse untrusted infrastructure through **Trusted Wrapper:**

  - Add security around existing APIs

    - Cryptography

    - Additional checks (may require copy of critical data, if original data cannot be trusted)

- General idea similar to TLS, VPN

VPN: Confidentiality, Integrty, ~~Availability~~
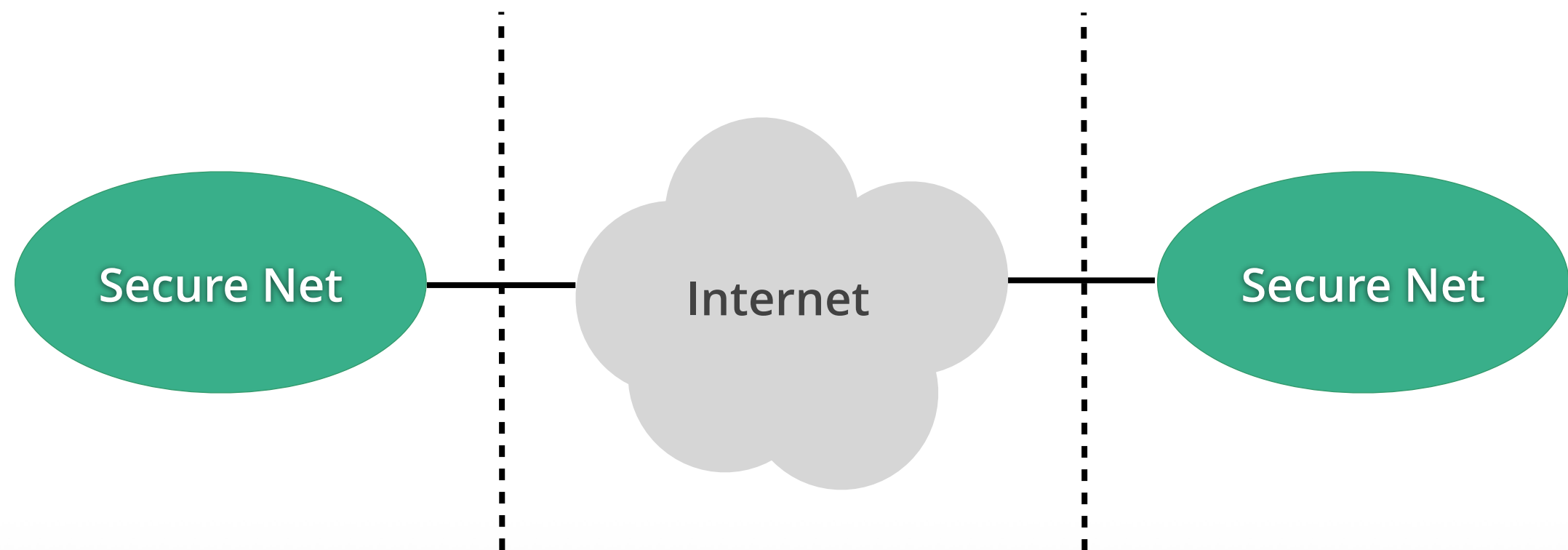
- SINA box used by German „BSI":

- VPN gateway

- Implements IPSec & PKI

- Intrusion detection & response

- Used for secure access to government networks, e.g., in German embassies

- Differently trusted network interfaces:

  - **Red:** plaintext, no protection

  - **Black:** encryption + authentication codes

Internet	↔	Ethernet (external)	↔	VPN Software	↔	Ethernet (internal)	↔	

- VPN Software:

  - Based on minimized and hardened Linux

  - Runs only from CD-ROM or Flash

- Linux is complex!

- SLOC for Linux 2.6.18:

  - Architecture specific: 817,880

  - x86 specific: 55,463

  - Drivers: 2,365,256

  - Common: 1,800,587

- Typical config: ~ 2,000,000

- Minimized & hardened: ~ 500,000

- Research project „Mikro-SINA"

- Goals:

  - Reduce TCB of VPN gateway software

  - Enable high-level evaluation for high assurance scenarios

  - Ensure confidentiality and integrity of sensitive data within the VPN

  - Exploit microkernel architecture

- Protocol suite for securing IP-based communication

- Authentication header (**AH**)
  - Integrity
  - Authentication

- Encapsulating Security Payload (**ESP**)
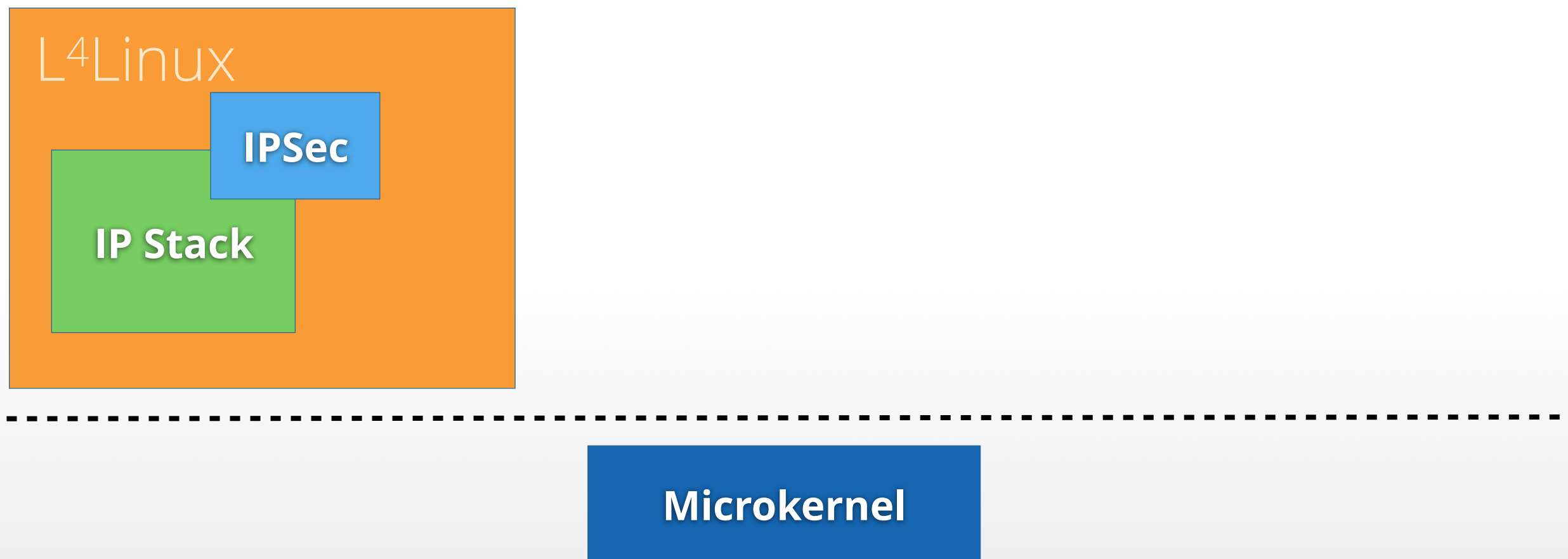 - Confidentiality
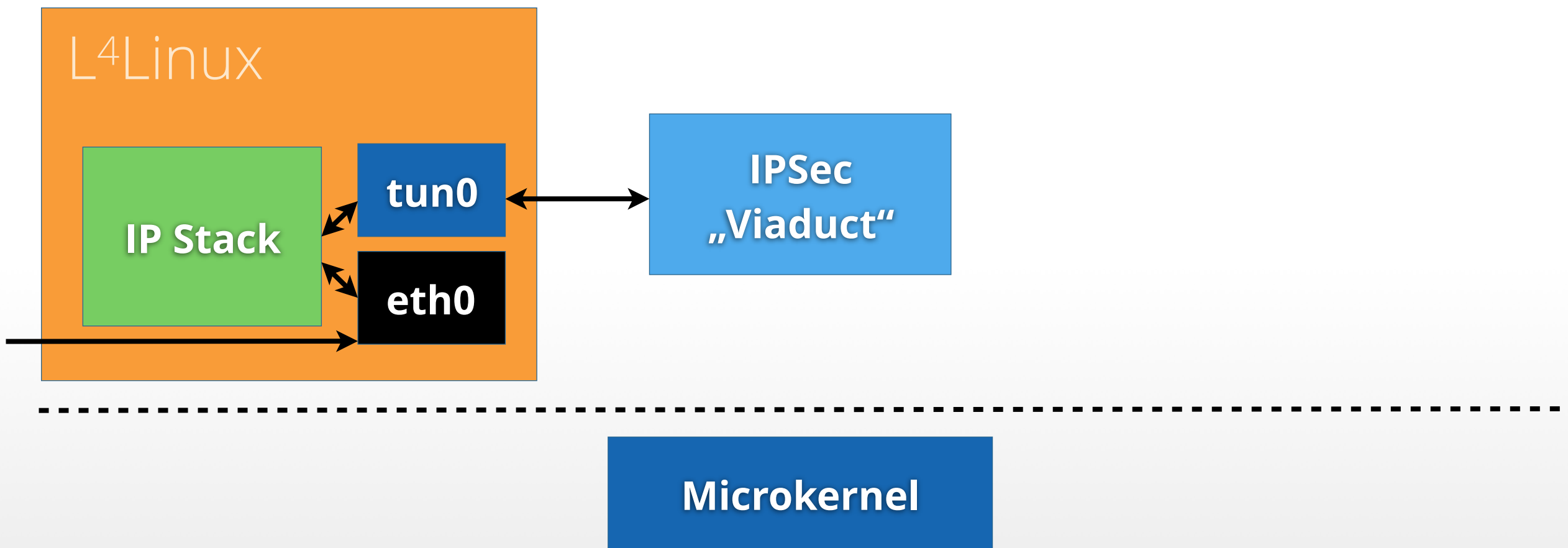
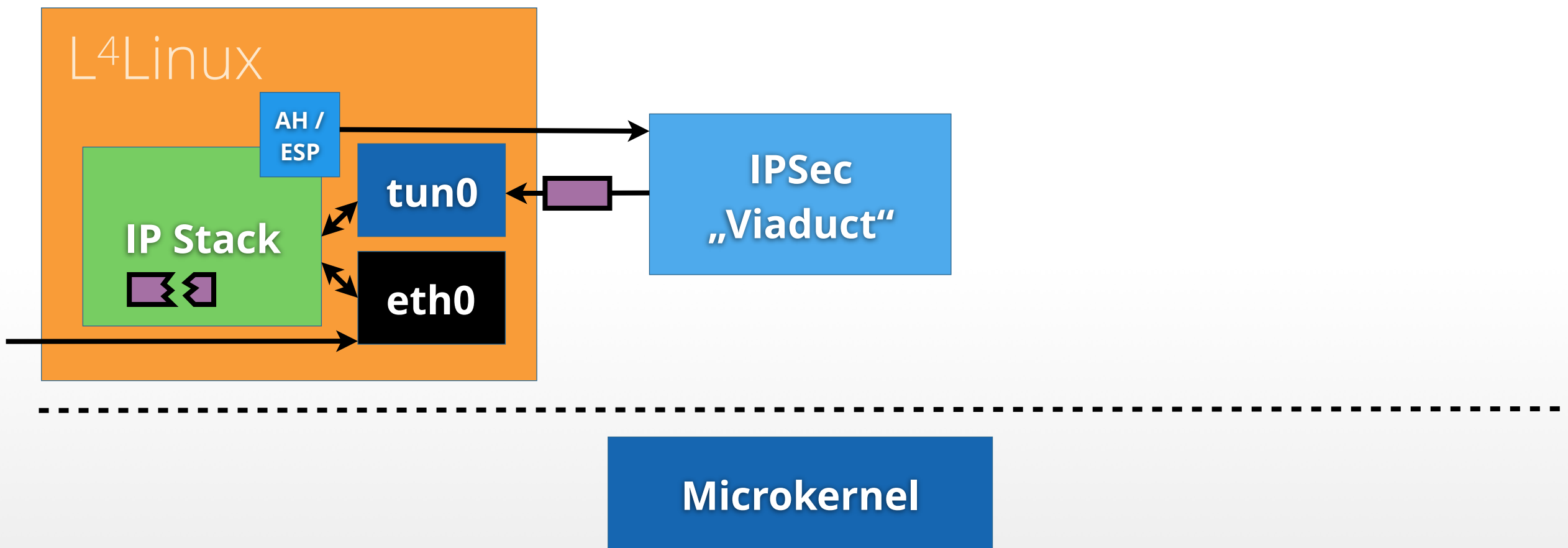- Key management / exchange

Application

TCP / UDP

IP

IPSec

Link Layer

- IPSec is security critical component

- ... but is integrated into Linux kernel

L⁴Linux
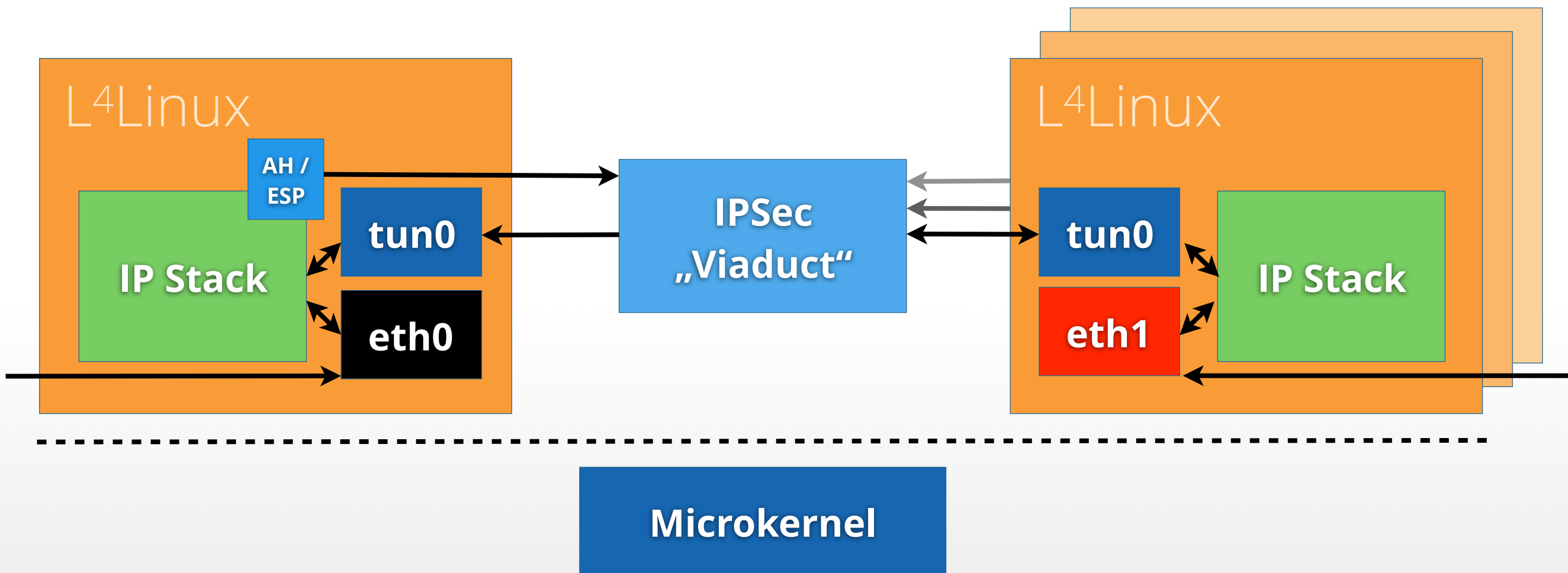
IPSec

IP Stack

Microkernel

- **Idea:** Isolate IPSec in „Viaduct"

- IPSec packets sent/received through TUN/TAP device

- Problem: Routers can fragment IPSec packets on the way

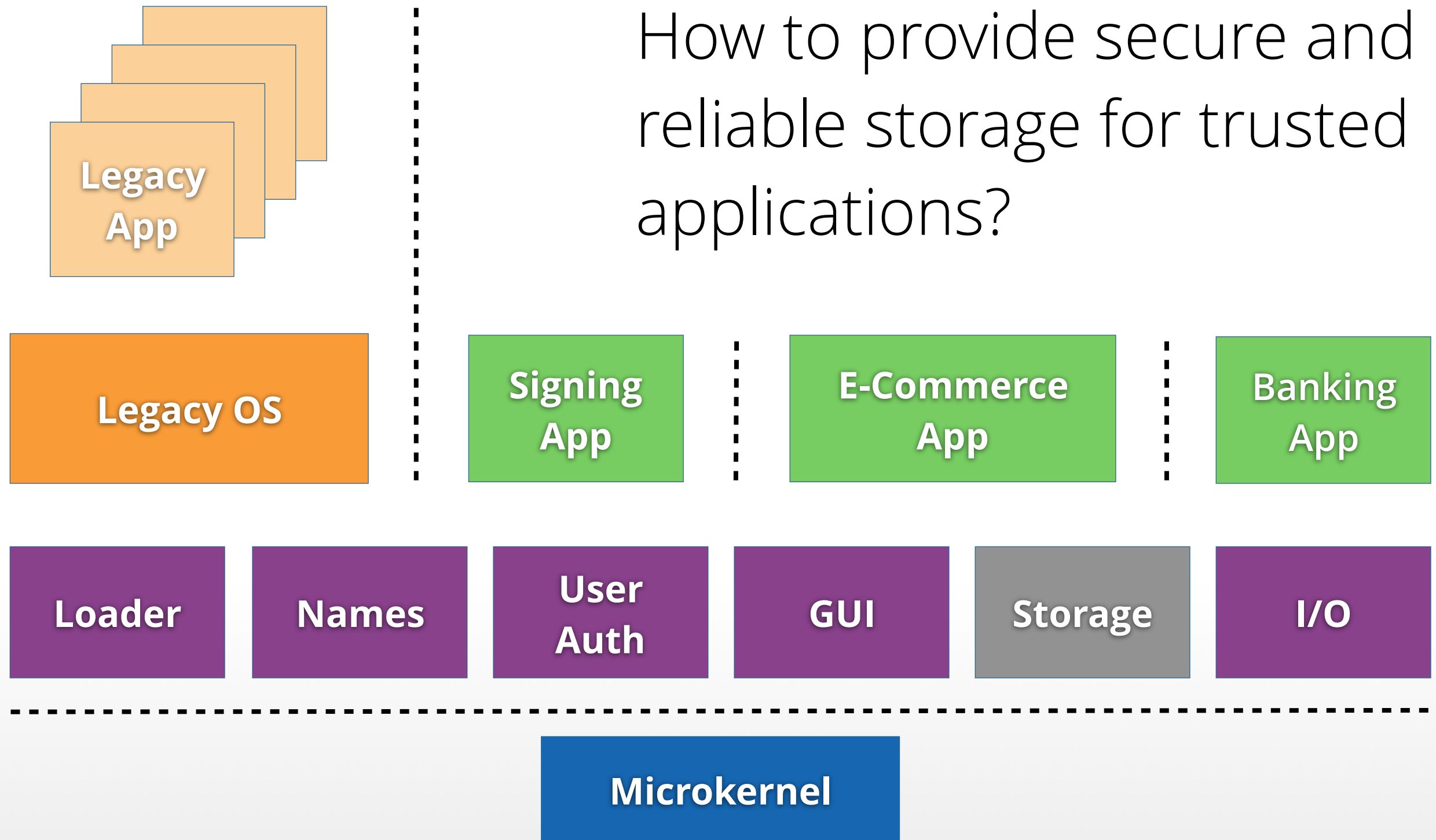- Let L^{4}Linux reassemble them

- Untrusted L⁴Linux must not see both plaintext and encrypted data

- Dedicated L⁴Linux for black/red networks

- Result: Trusted Wrapper for VPN

- Small TCB (see [6] for details):

  - 5,000 SLOC for „Viaduct"

  - Fine grain isolation

  - Principle of least privilege

- Extensive reuse of legacy code:

  - Drivers

  - IP stack

TECHNISCHE
UNIVERSITÄT
DRESDEN

How to provide secure and reliable storage for trusted applications?

Legacy App

Legacy OS

Signing App

E-Commerce App

Banking App

Loader

Names
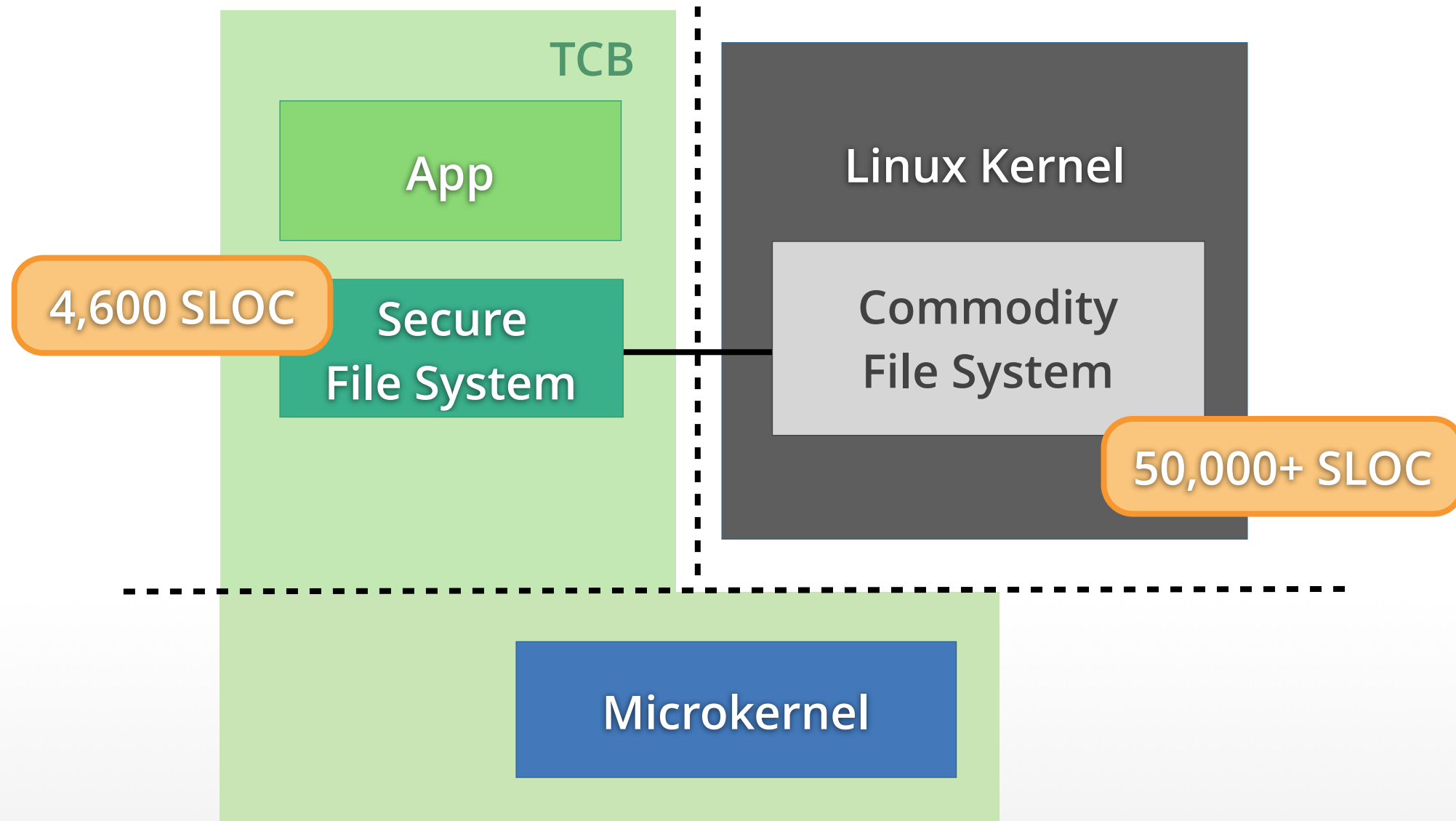
User Auth

GUI

Storage

I/O

Microkernel

VPFS: Confidentiality, Integrity, ~~Availability~~
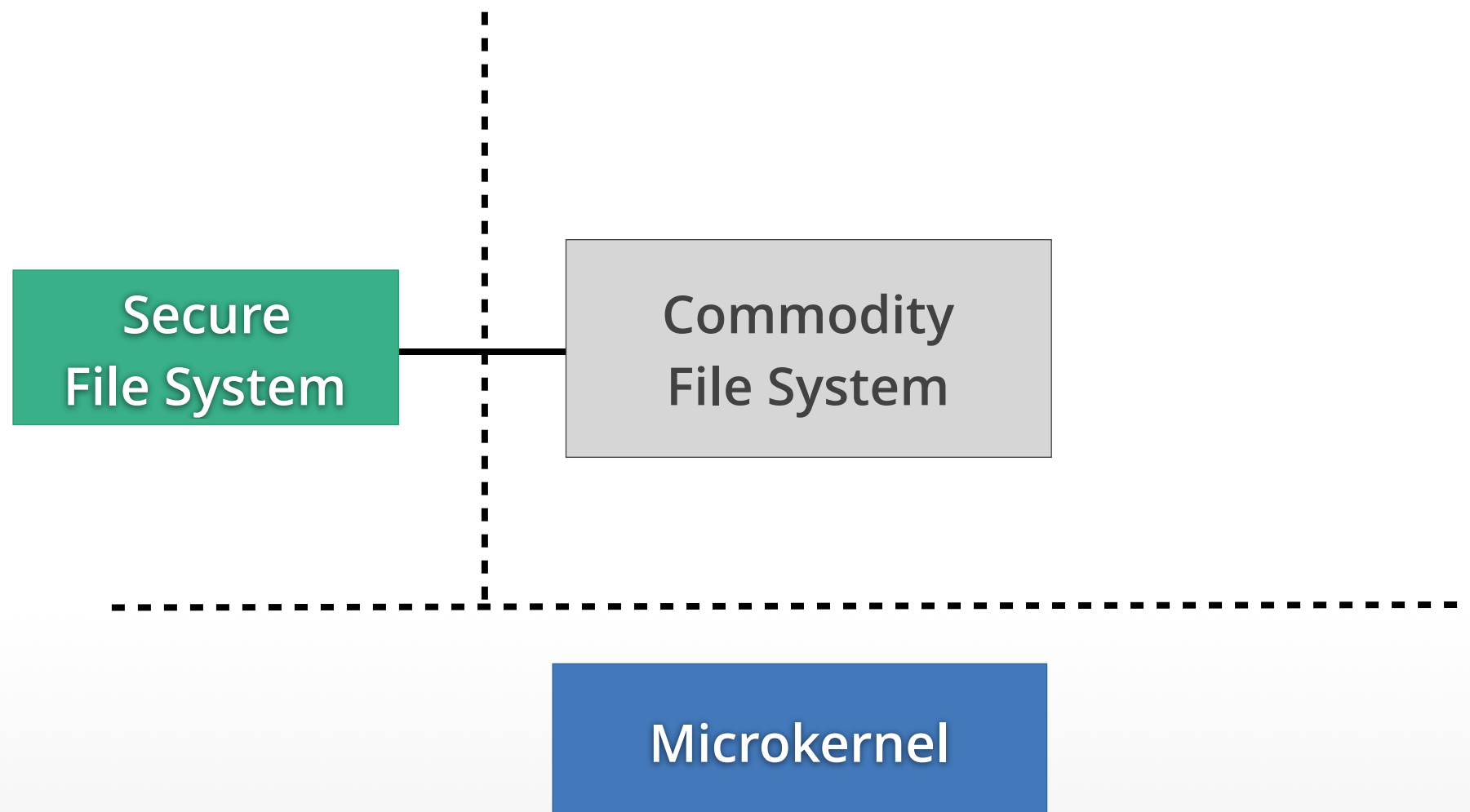
See [3] for details

VPFS: Confidentiality, Integrity, ~~Availability~~

See [3] for details

Signing App

Office App

Banking App

Secure File System

Secure File System

Secure File System

Secure File System Proxy

Secure File System Proxy

Secure File System Proxy

Commodity File System
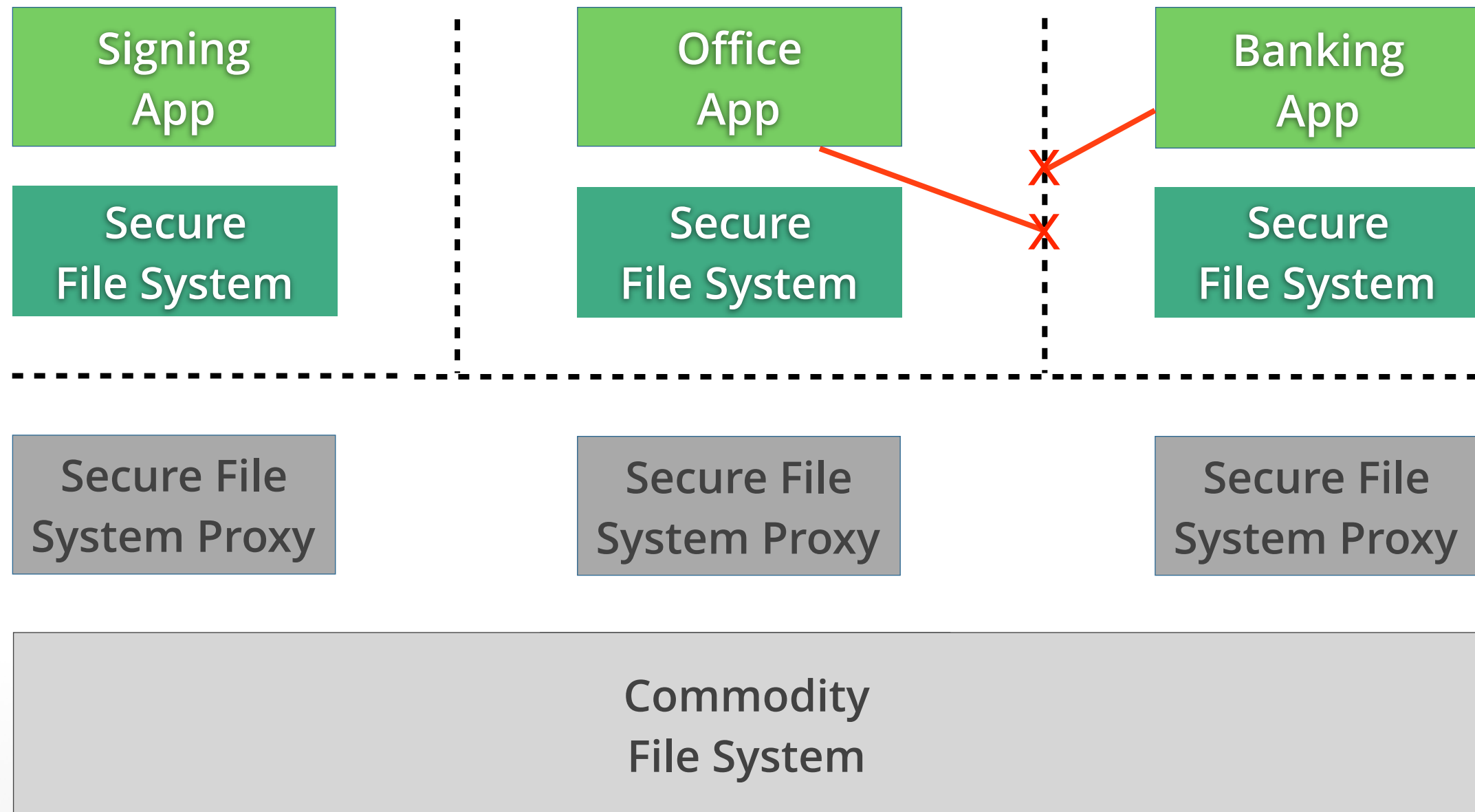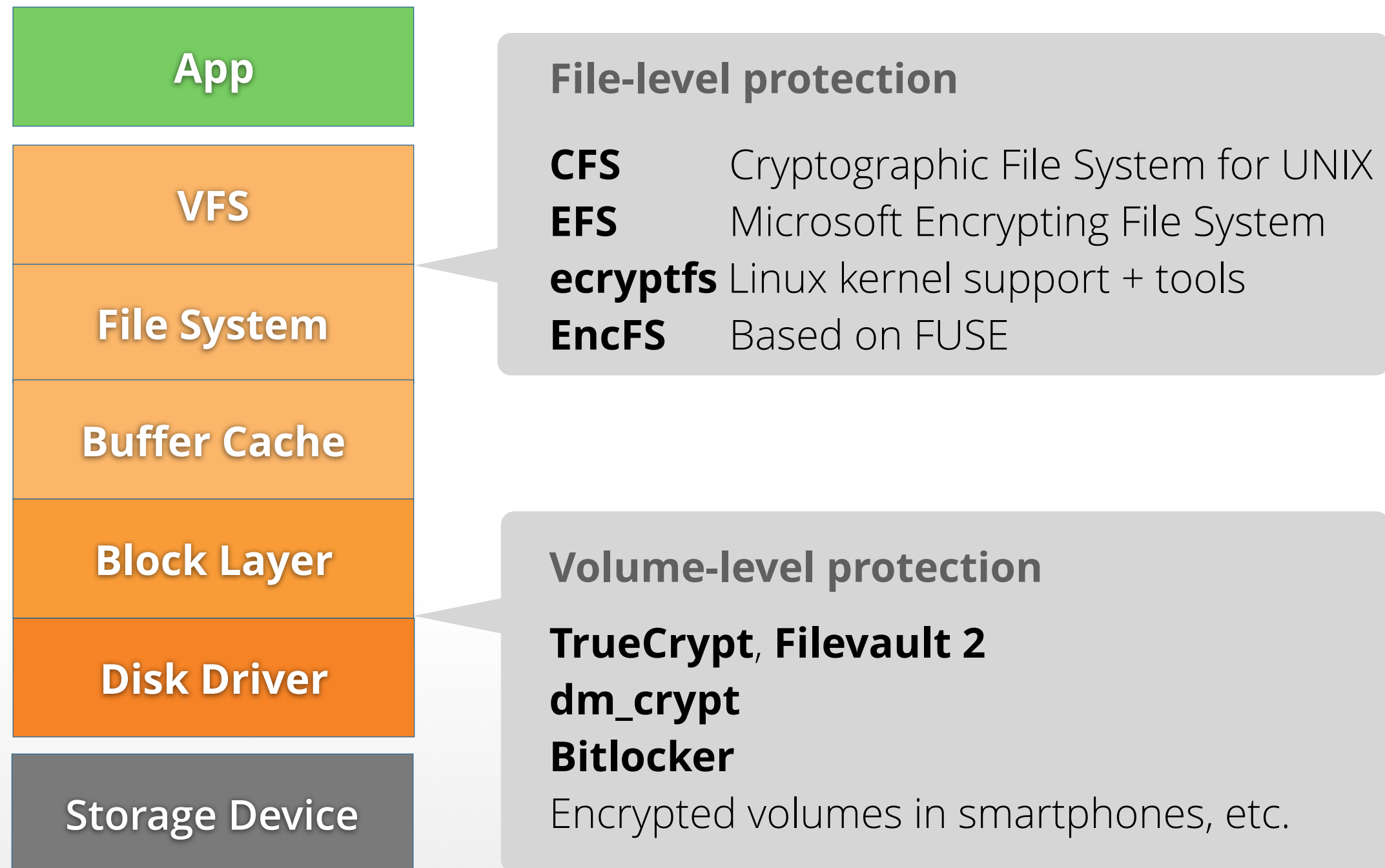
Isolate applications and their private storage: configure communication capabilities such that each application can access its private instance of the secure file system exclusively

- **Confidentiality:** only authorized applications can access file system, all untrusted software cannot get any useful information

- **Integrity:** all data and meta data is correct, complete, and up to date; otherwise report integrity error

- **Recoverability:** damaged data in untrusted file system can be recovered

POPULAR SOLUTIONS

App

| VFS |
| File System |
| Buffer Cache |
| Block Layer |
| Disk Driver |
| Storage Device |

File-level protection

CFS Cryptographic File System for UNIX
EFS Microsoft Encrypting File System
ecryptfs Linux kernel support + tools
EncFS Based on FUSE

Volume-level protection

TrueCrypt, **Filevault 2**
dm_crypt
Bitlocker
Encrypted volumes in smartphones, etc.

- First end of design space: Protect at block layer

  - Transparent encryption of all data and metadata

  - Block-level integrity ???

  - Most parts of file system stack are part of TCB

  - Attack surface still big

App
VFS
File System
Buffer Cache
Block Layer
Protection
Disk Driver
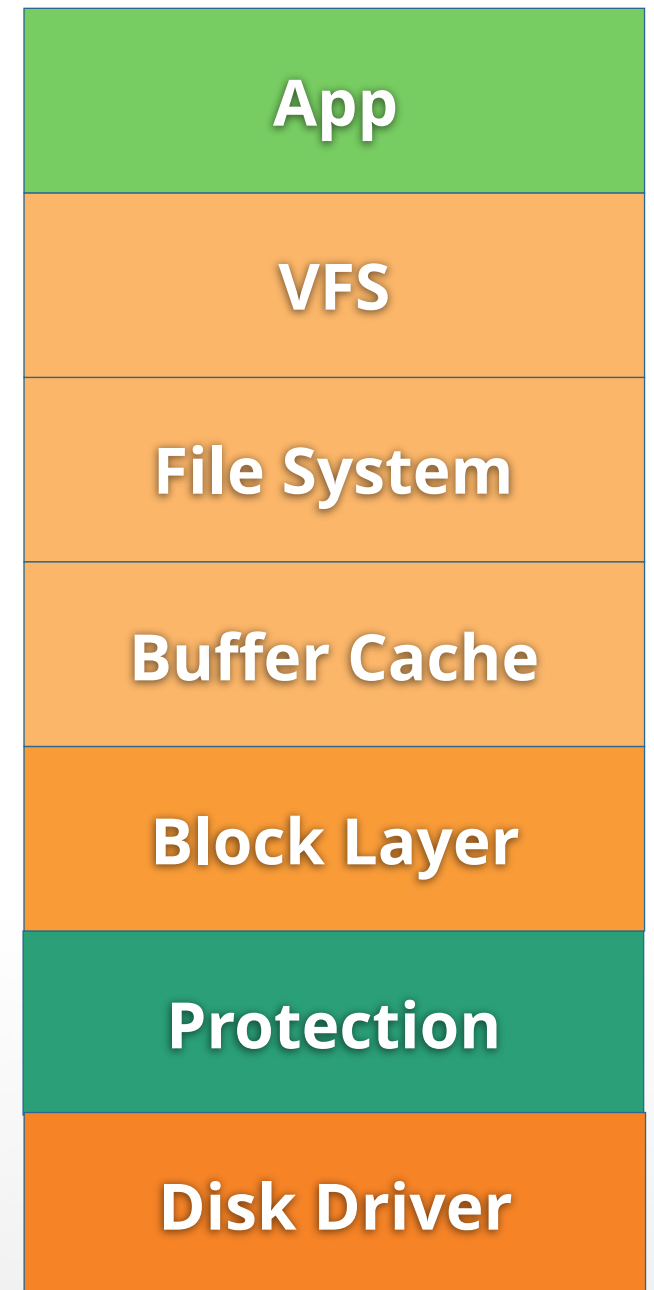
- Second end of design space: Protect individual files

  - Stacked file system

  - Encrypt all data and some metadata (directories, …)

  - More flexibility for integrity

  - Most parts of file system stack <u>not</u> part of TCB
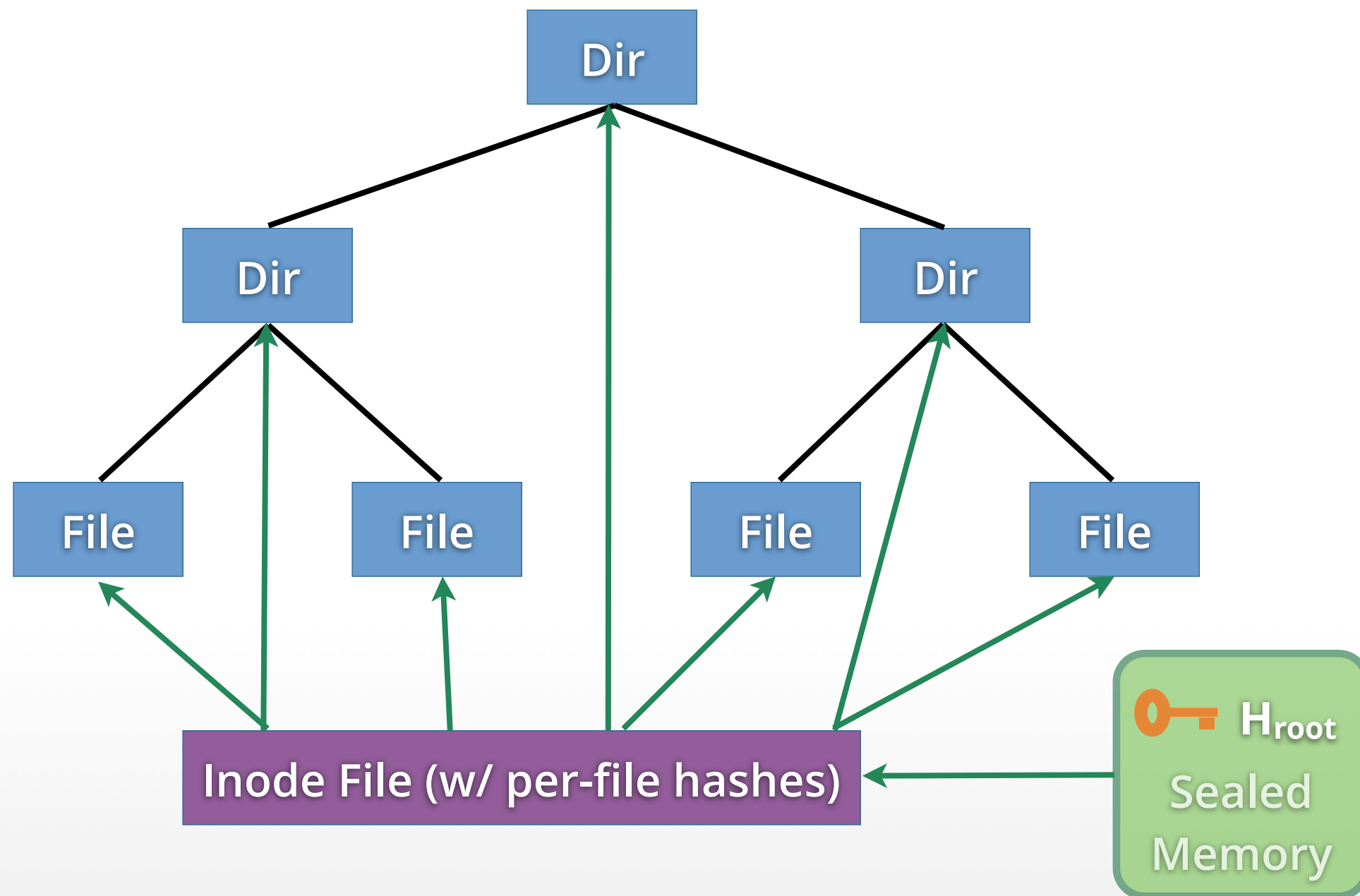
  - Ideal for trusted wrapper
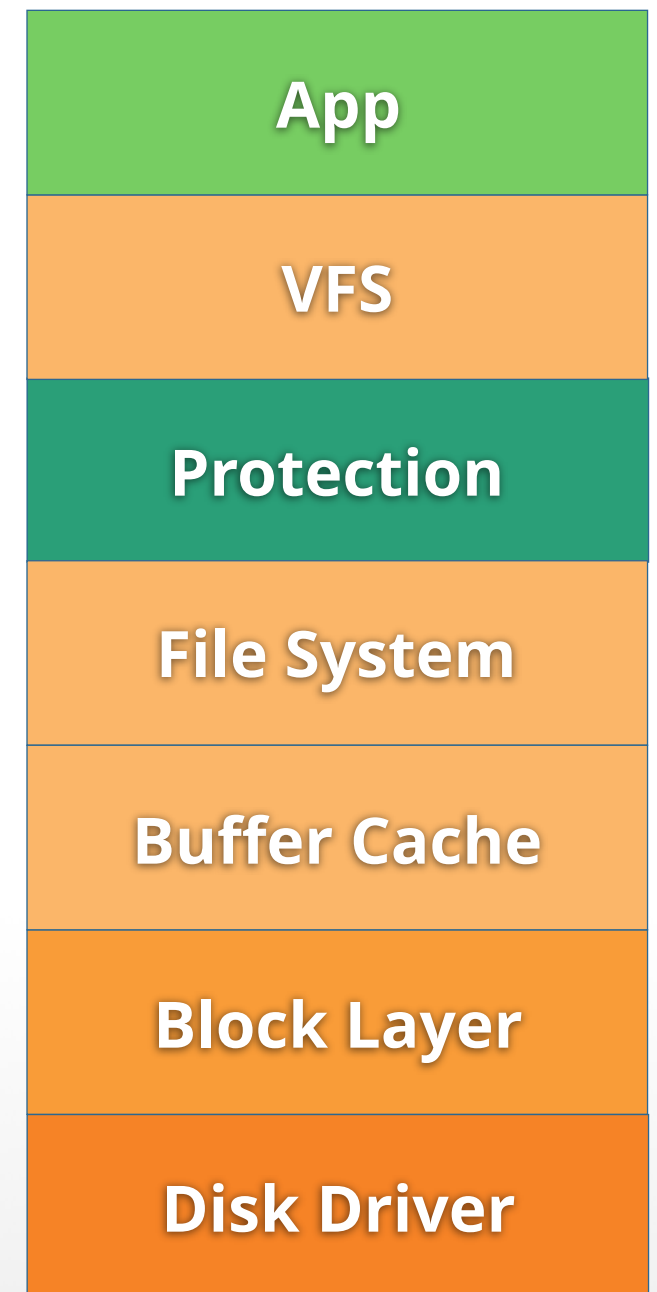
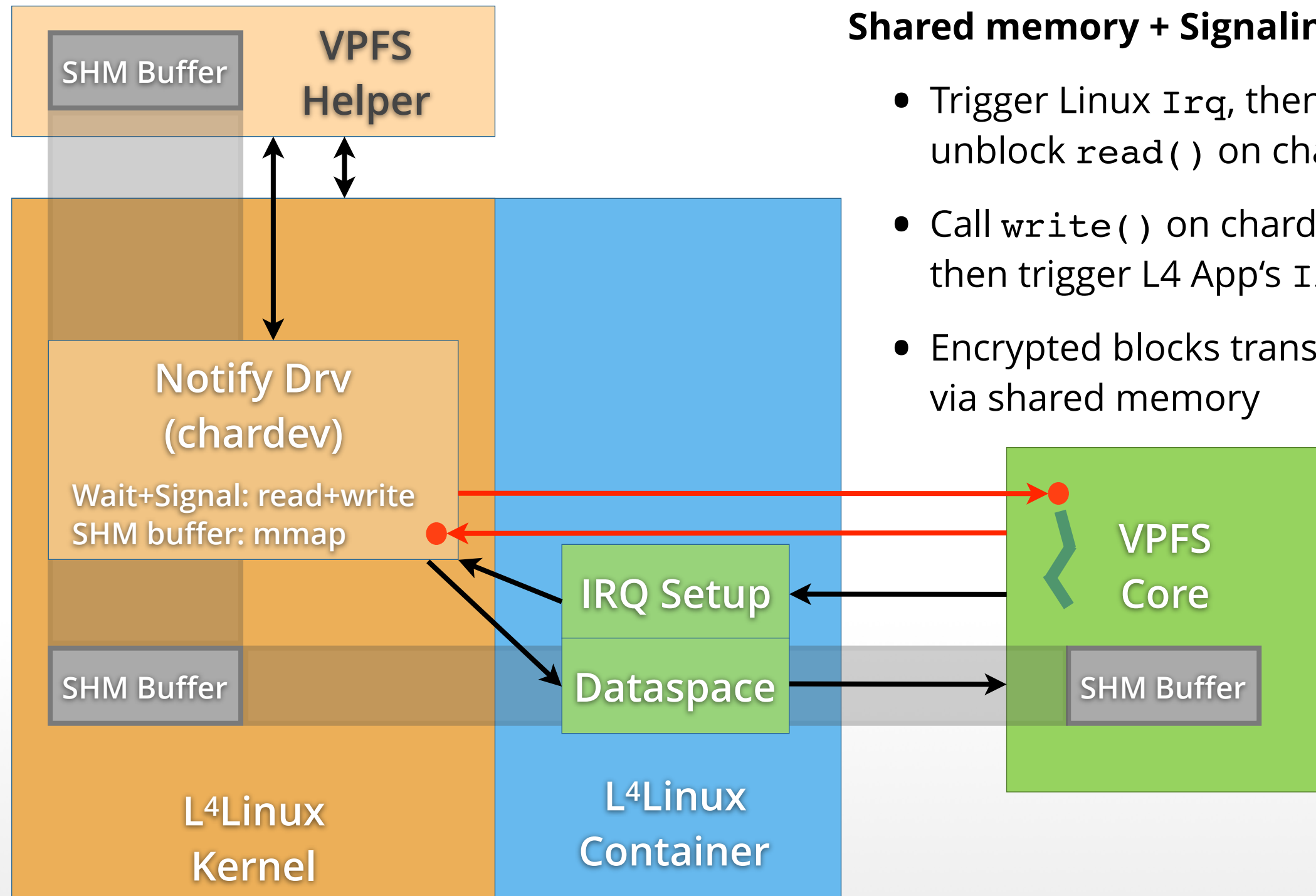| App |
| --- |
| VFS |
| Protection |
| File System |
| Buffer Cache |
| Block Layer |
| Disk Driver |
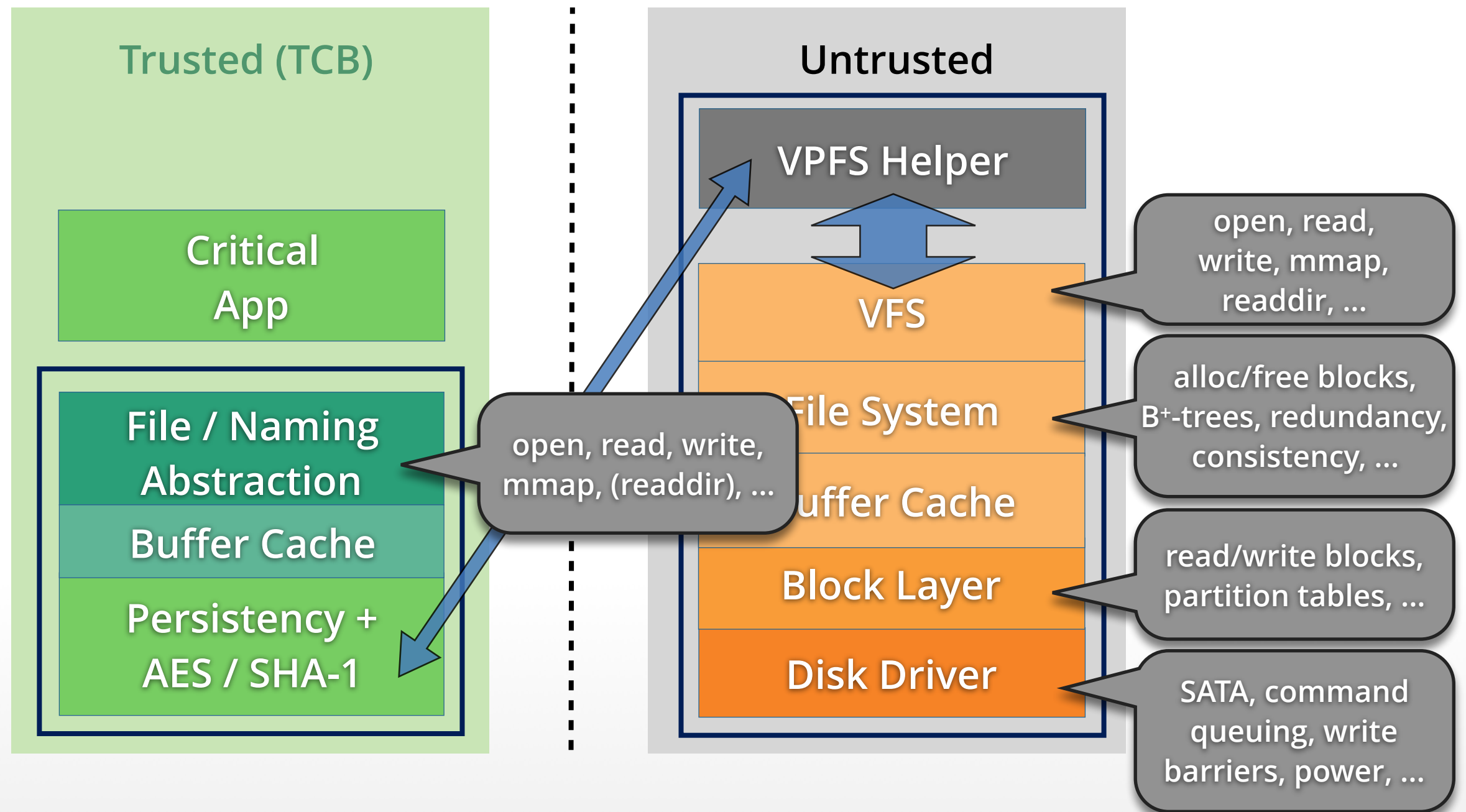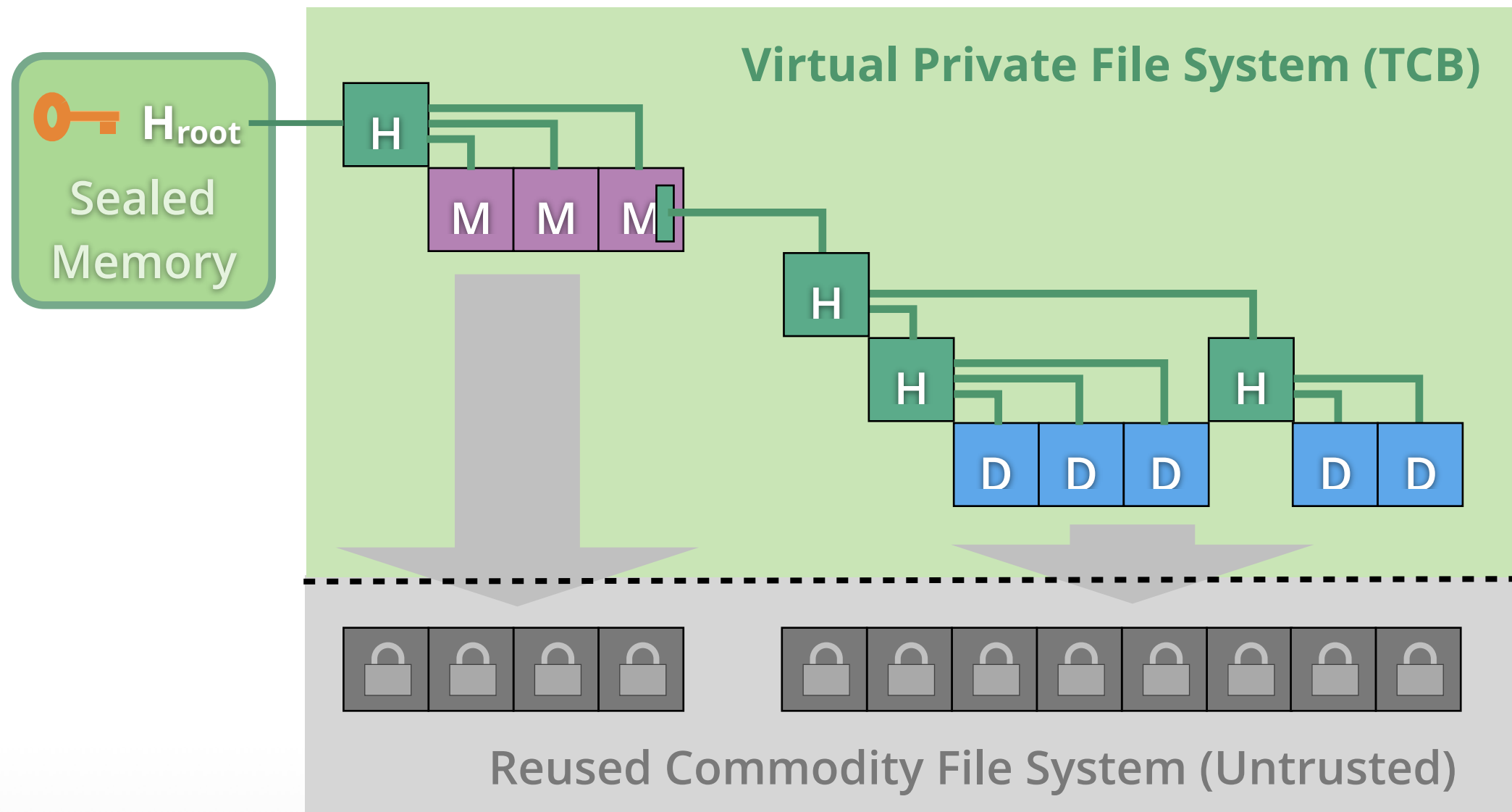
- Encrypted files in commodity file system

- Merkle **hash tree** to detect tampering

- Trusted part of VPFS enforces security:

    - Encryption / decryption on the fly

    - Plaintext only in trusted buffer cache

    - Files in untrusted commodity file system store encrypted blocks

    - Hash tree protects integrity of complete file system

    - Single hash of root node stored securely

- VPFS reuses Linux file system stack:

  - Drivers, block device layer

  - Optimizations (buffer cache, read ahead, write batching, ...)

 - Allocate / free disk storage for files

- Cooperation: proxy driver in L^{4}Linux

Shared memory + Signaling:

- Trigger Linux `Irq`, then unblock `read()` on chardev

- Call `write()` on chardev, then trigger L4 App's `Irq`

- Encrypted blocks transferred via shared memory

Labels within the figure:

- SHM Buffer
- VPFS Helper
- Notify Drv (chardev)
- Wait+Signal: read+write
- SHM buffer: mmap
- SHM Buffer
- L⁴Linux Kernel
- IRQ Setup
- Dataspace
- L⁴Linux Container
- VPFS Core
- SHM Buffer

- Trusted wrappers for file systems work!

- VPFS is general purpose file system

- Significant reduction in code size:

  - Untrusted Linux file system stack comprises **50,000+** SLOC

  - VPFS adds **4,000** to **4,600** SLOC to application TCB [3]

  - jVPFS adds another **350** SLOC for secure journaling to protect against crashes [4]

USER INTERFACES

- Isolated applications run in different domains of trust, but separate screens are inconvenient

- The Nitpicker solution [5]:

  - Let all windows share the same screen ...

  - ... but securely:

    - Make windows & applications identifiable

    - Prevent them from spying on each other: route input securely, no screenshots
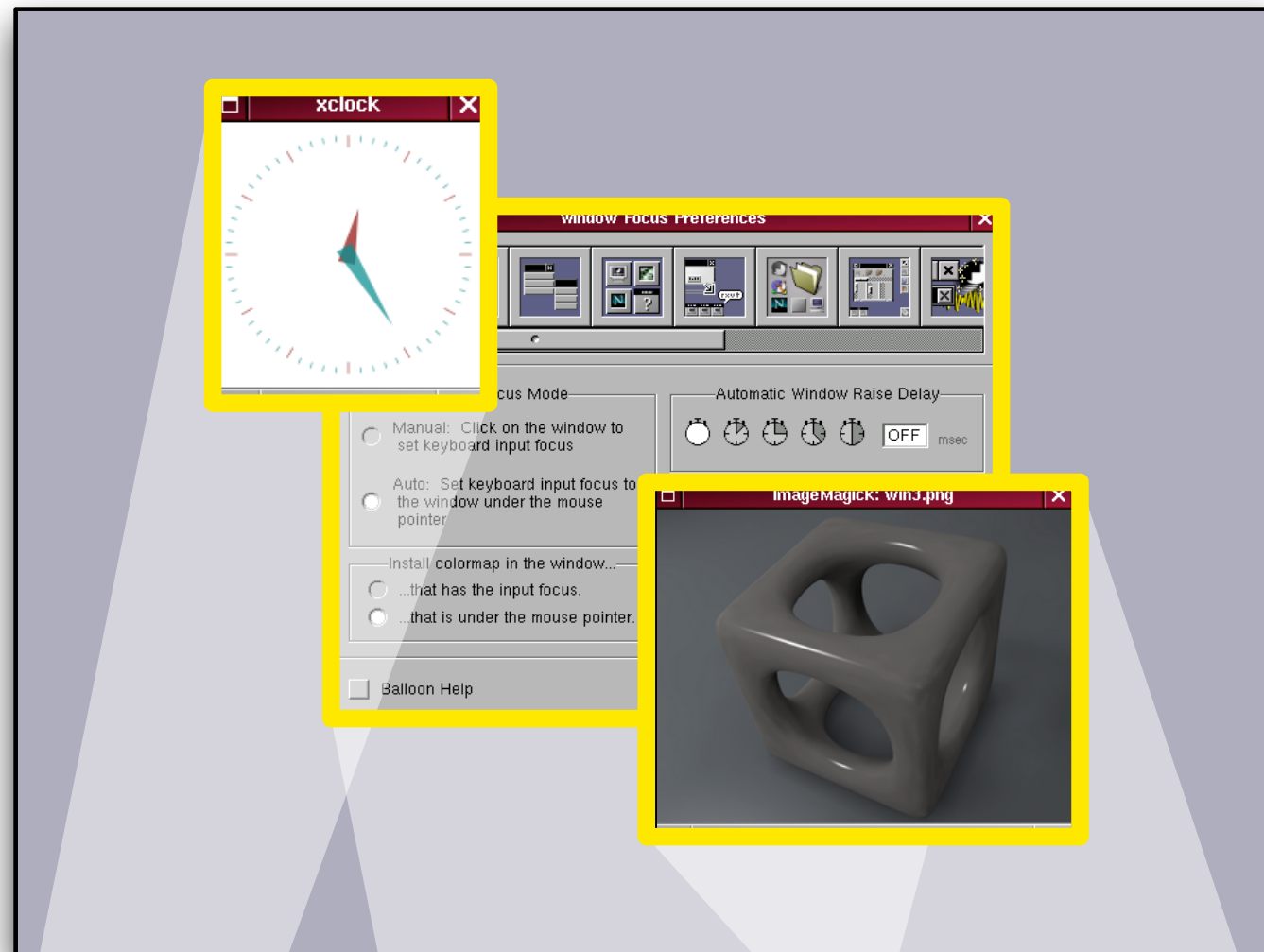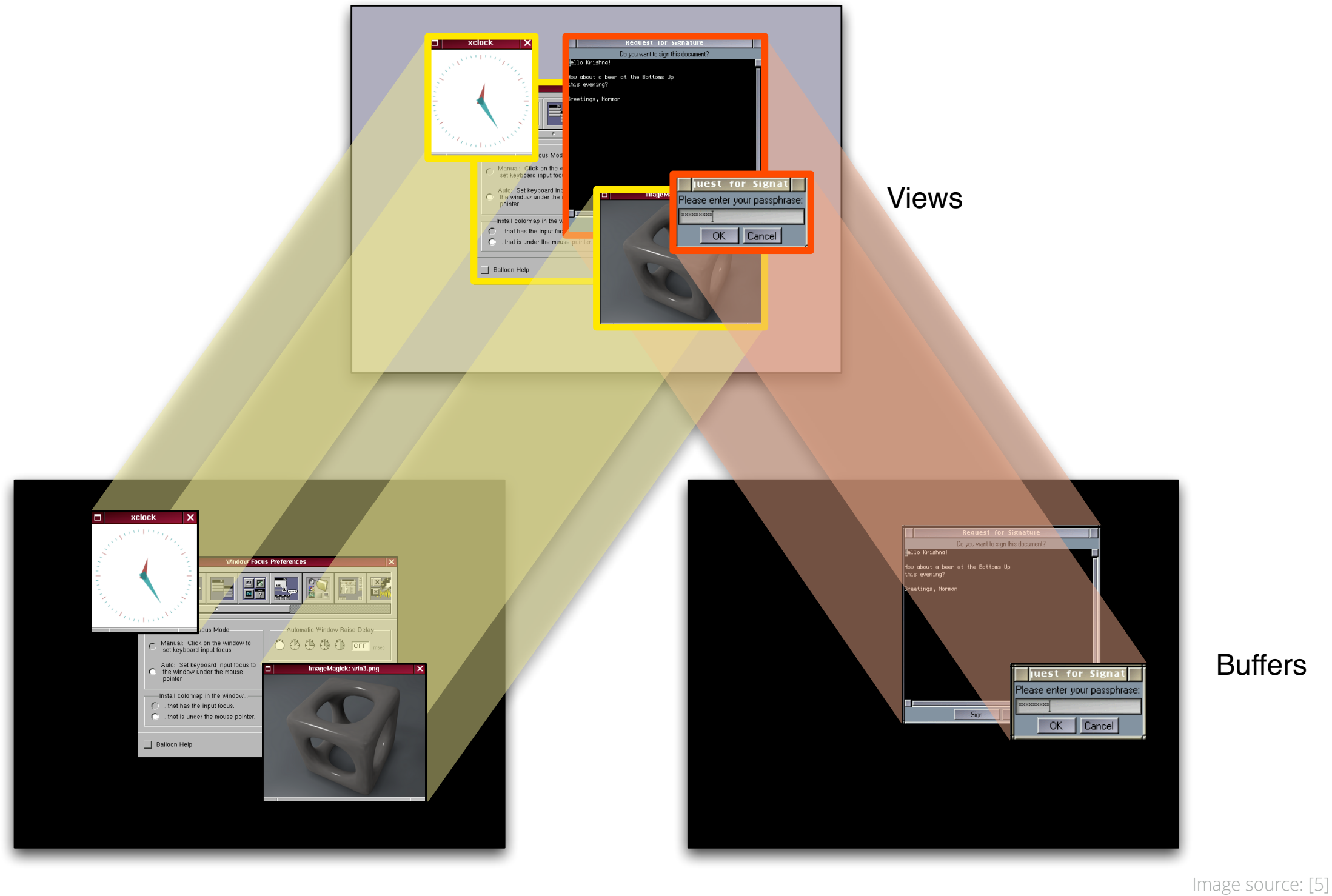
Views

Buffers

Image source: [5]
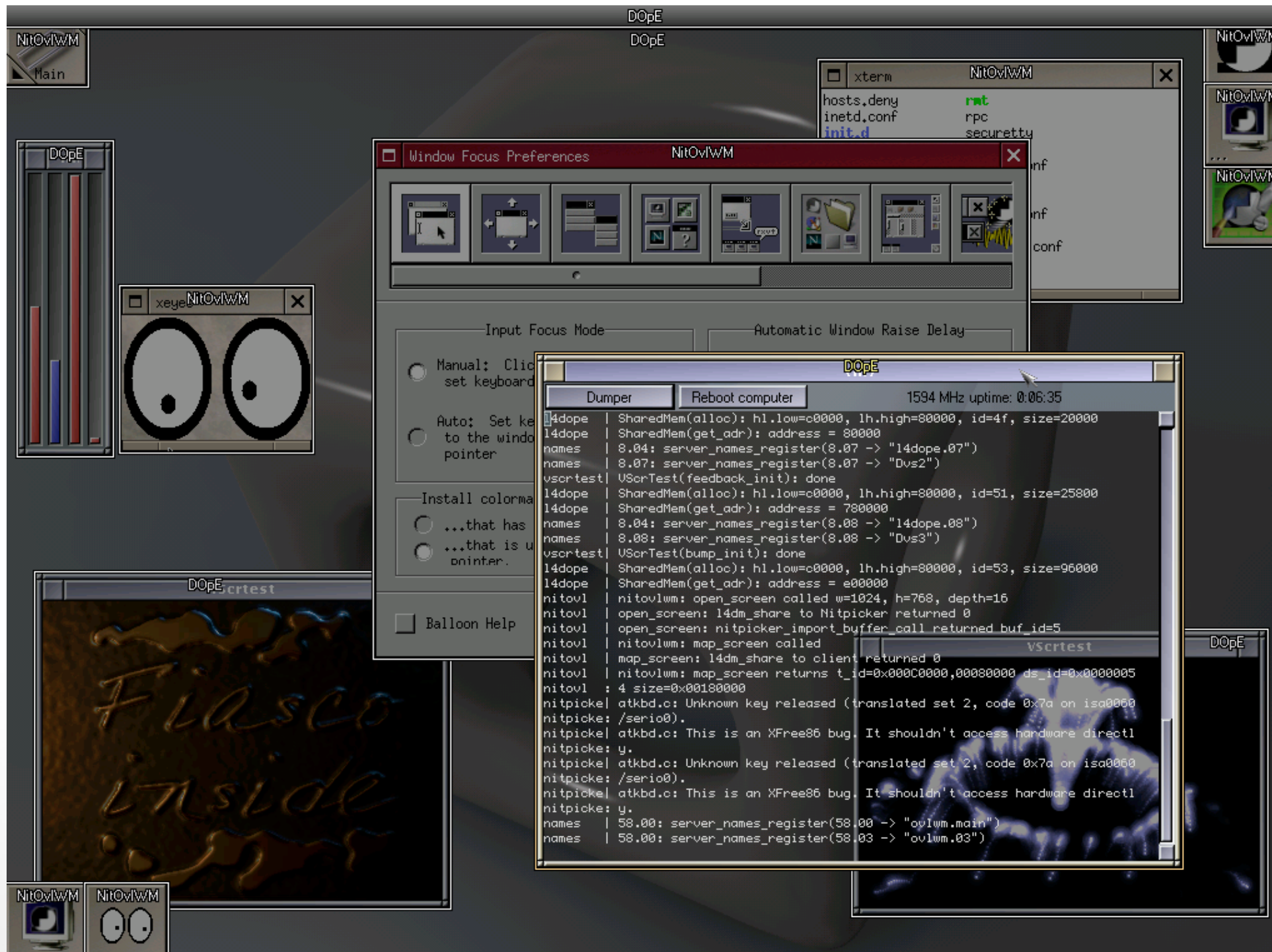
Views
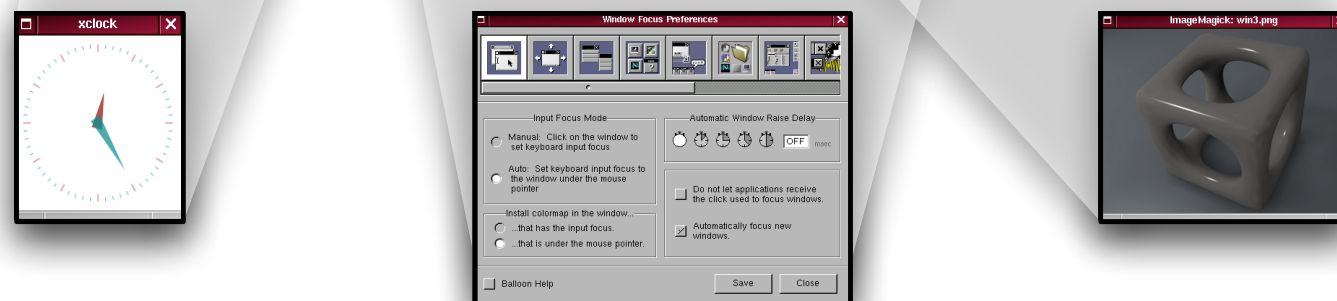
Buffers

Image source: [5]

DEMO

Image source: [5]

- Secure reuse of untrusted legacy infrastructure

- Split apps + OS services for smaller TCB

- Nizza secure system architecture:

  - Strong isolation

  - Application-specific TCBs

  - Legacy Reuse

  - Trusted Wrapper

- [1] http://www.heise.de/newsticker/Month-of-Kernel-Bugs-Ein-Zwischenstand--/meldung/81454

- [2] http://projects.info-pull.com/mokb/

- [3] Carsten Weinhold and Hermann Härtig, „VPFS: Building a Virtual Private File System with a Small Trusted Computing Base", Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems, April 2008, Glasgow, Scotland UK

- [4] Carsten Weinhold and Hermann Härtig, „jVPFS: Adding Robustness to a Secure Stacked File System with Untrusted Local Storage Components", Proceedings of the 2011 USENIX Annual Technical Conference, Portland, OR, USA, June 2011

- [5] Norman Feske and Christian Helmuth, „A Nitpicker's guide to a minimal-complexity secure GUI", ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference, 2005, Washington, DC, USA

- [6] Christian Helmuth, Alexander Warg, Norman Feske, „Mikro-SINA - Hands-on Experiences with the Nizza Security Architecture", D.A.CH Security 2005, 2005, Darmstadt, Germany

- [7] http://support.apple.com/kb/HT4013

- [8] http://support.apple.com/kb/HT3754

- [9] http://jailbreakme.com

- [10] Asmussen et al.: „M3: A Hardware/OS Co-Design to Tame Heterogeneous Manycores", ASPLOS'16