TECHNISCHE UNIVERSITÄT DRESDEN

**Department of Computer Science** Institute for System Architecture, Operating Systems Group

# SECURITY ARCHITECTURES

**CARSTEN WEINHOLD**

# Motivation

- Common observations:
  - Complex software has security bugs
  - Users are plagued by malware
  - Companies, governments are high value targets
  - Critical data gets stolen
  - User PCs become bots
- Sad truth: threats won‘t go away

# Mobile Platforms

- It‘s all the same for mobile devices
- Malware in Android Store: trojan horse downloaded by millions of users
- Security-critical bugs:
  - Drivers [1,2], USB stacks [6], boot loaders
  - Messaging apps [7], web browser, ...
- „Jailbreaking“ = attack on security:
  - Requires physical access ...
  - ... or visit special website [8]

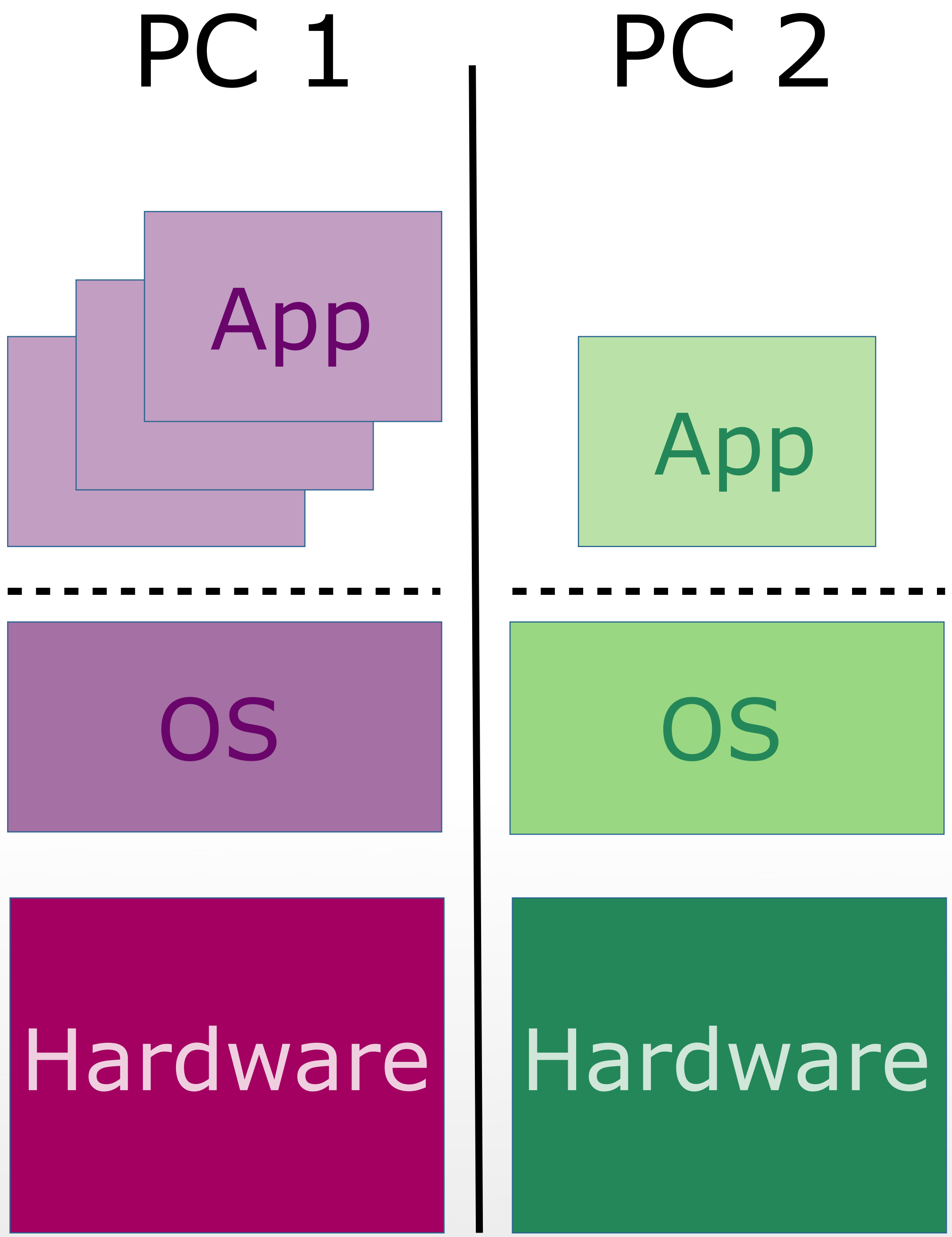
# Classical Architectures

# Isolation

- Isolation in commodity OSes based on user accounts
- Problems:
  - Same privileges for all apps
  - Permissive interfaces (e.g., ptrace to manipulate other address spaces)
  - No isolation within application
- Efforts to restrict privileges:
  - SELinux, AppArmor, Seatbelt, ...

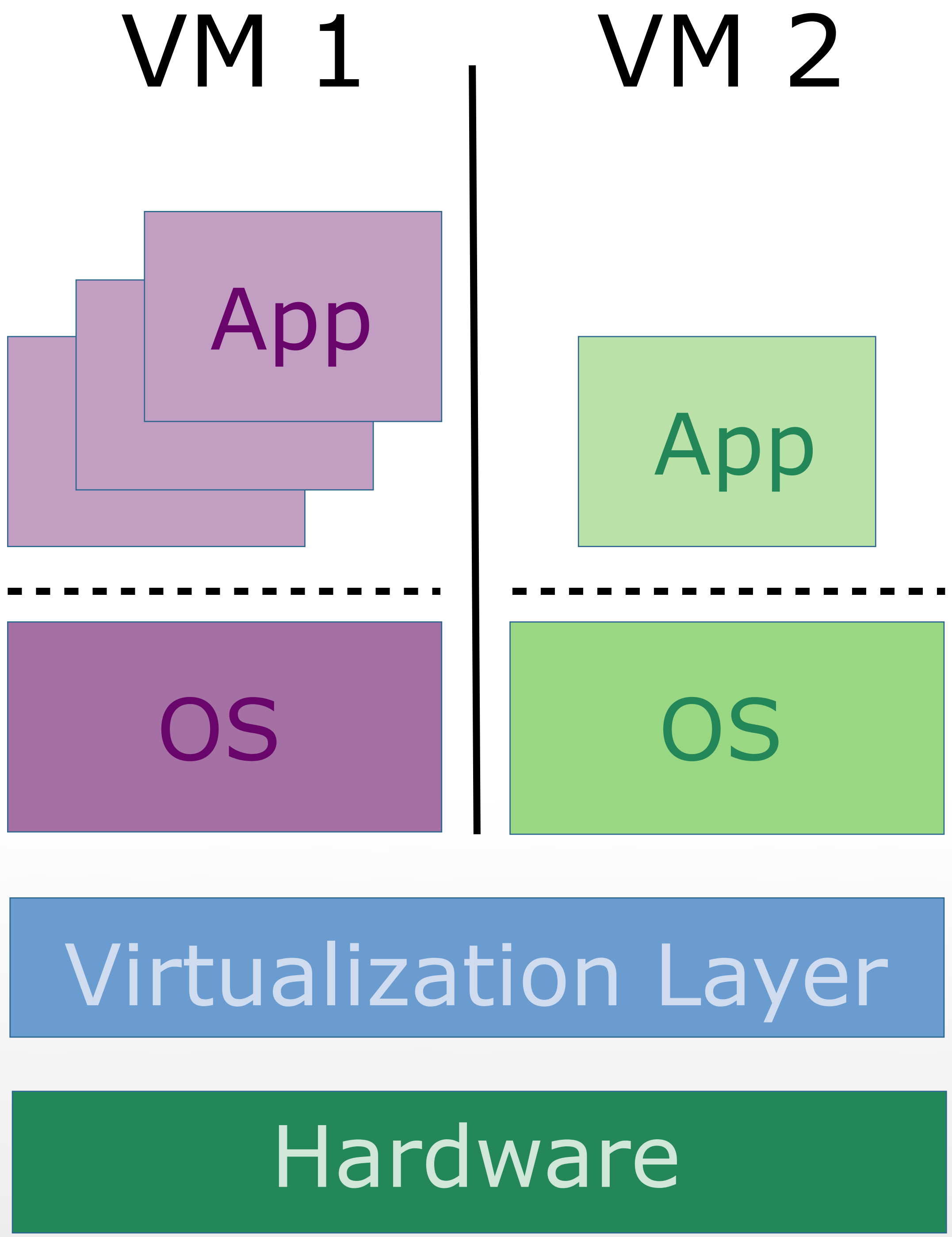
# Physical Isolation

- Separate computers
- Applications and data physically isolated
- Effective, but ...
  - High costs
  - Needs more space
  - Inconvenient
  - Exposure to network may pose threat



# Virtual Machines

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



# Little Effect ...

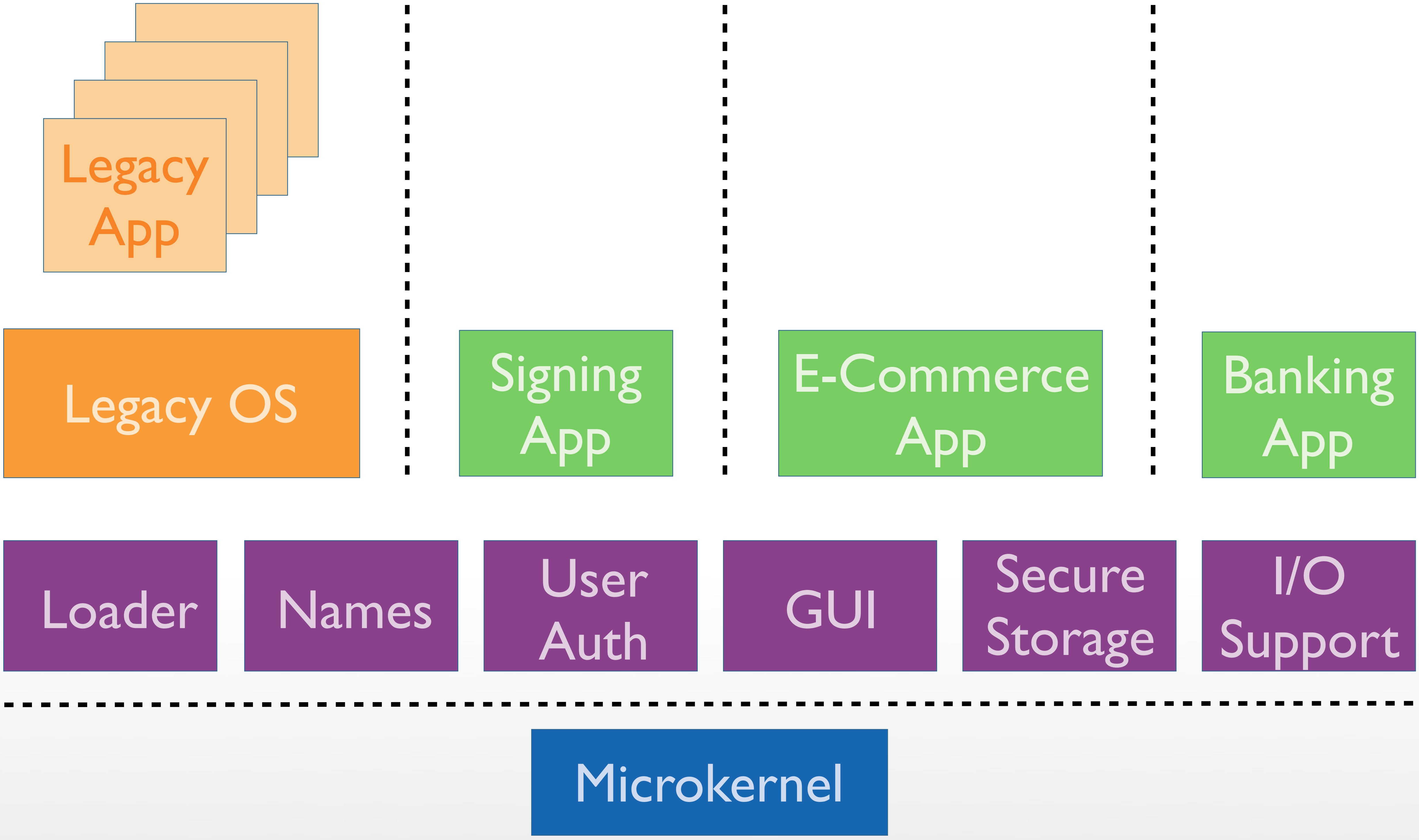
- Huge code bases remain
- Many targets to attack:
  - Application, Commodity OS
  - Virus scanner, firewall, ...
- Expensive communication via (virtual) Ethernet
- High resource consumption even for small applications

# Security Architectures

# TCB--; Security++;

- To further improve security:
  - Reduce size of TCB = attack surface
- First idea:
  - Port application to microkernel-based multi-server OS
  - Remove huge legacy OS from TCB
  - Remove unneeded libc backends, etc.
  - Possible approaches discussed in lecture on „Legacy Containers"

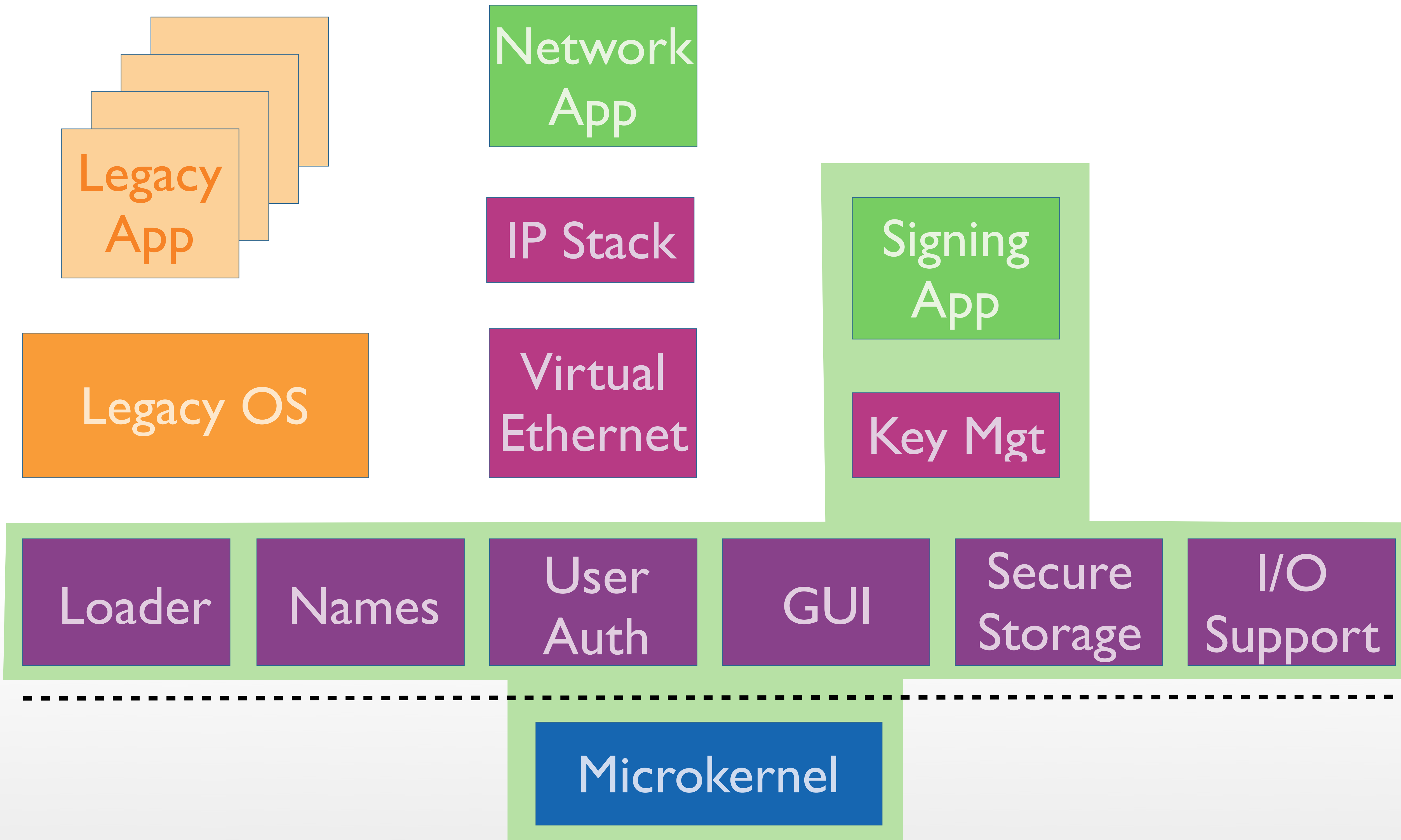
# Nizza Architecture



# Nizza Key Principles

- Nizza architecture based on basic design concepts:
  - Strong isolation
  - Application-specific TCBs
  - Legacy reuse
  - Trusted wrapper
  - Trusted Computing

# App-specific TCBs



# App-specific TCBs

- 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 for strong isolation
  - Well-defined interfaces

TECHNISCHE UNIVERSITÄT DRESDEN

# Splitting Components

# Split Applications

- Problems with porting applications:
  - Dependencies need to be satisfied
  - Can be complex, even infeasible
  - Stripped down applications may lack functionality / usability
- Better idea: split application
  - Make only security-critical parts run on microkernel-based OS
  - Reduces size of TCB even further

# Example 1: eMail

- Critical functionality to support digitally signed e-mails:
  - Handling of signature keys
  - Requesting passphrase to unlock secret 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

# Split eMail Signing

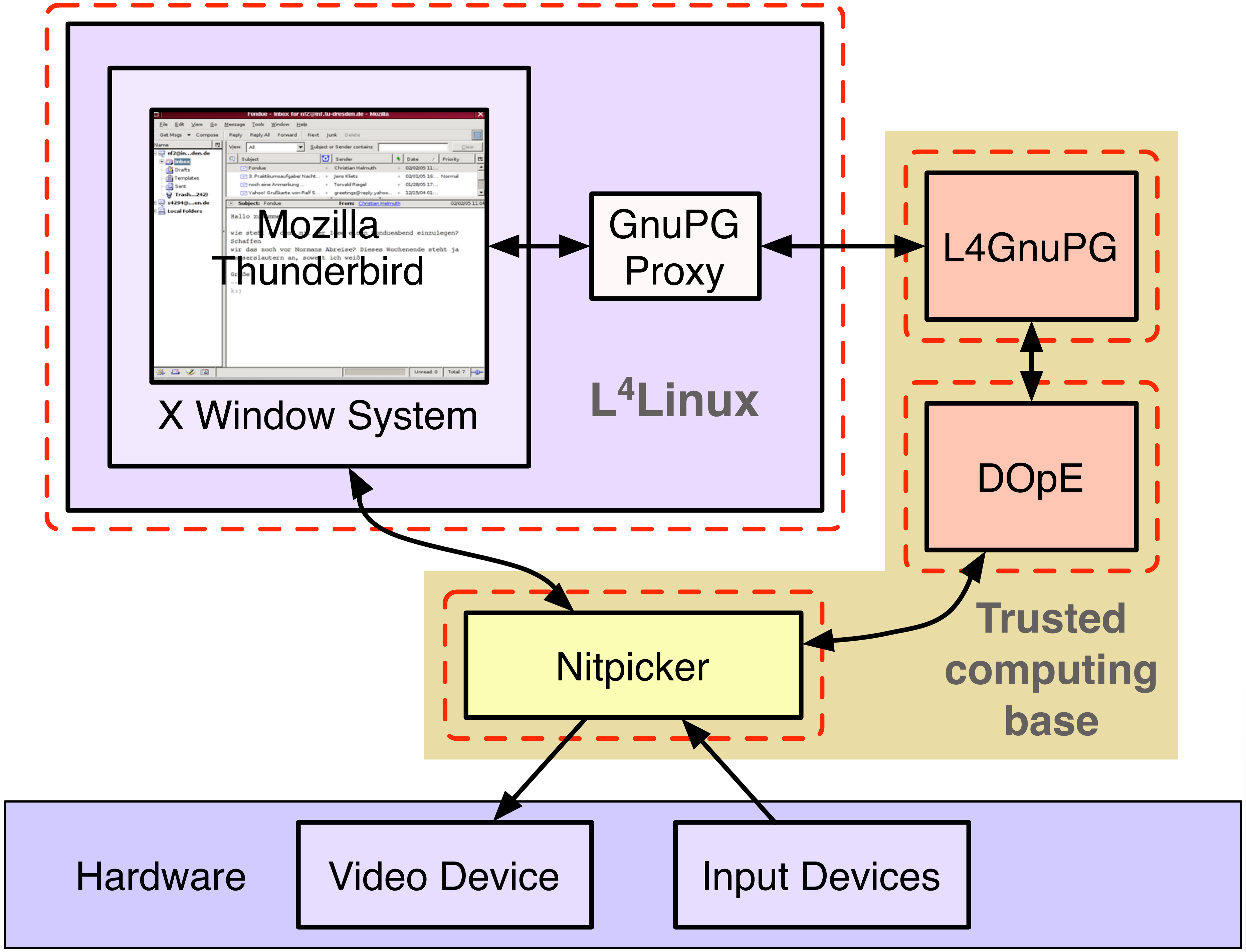



Figure taken from [4]

# Benefit of Splitting

- *>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
- Method not restricted to applications

# Reuse OS Services?

- Beyond applications:
  - New OS? Do not reinvent the wheel!
  - Existing software can be reused:
    - Protocol stacks (e.g., TCP/IP)
    - Commodity OSes (e.g., Linux)
  - Virtualization is important enabler

# Reuse Made Easy

- 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?

# Reuse OS Services

- Network and file system stacks are virtually essential subsystems
- Generally well tested
- Ready for production use
- ... but not bug free:
  - Month of Kernel Bugs 2006 [1,2]:
    - 14 exploitable flaws in file systems: UFS, ISO 9660, Ext3, SquashFS, ...
    - WiFi drivers: remotely exploitable bugs

# Trusted Wrappers

- Complex protocol stacks should not be part of TCB
- Reuse untrusted infrastructure through trusted wrapper:
  - Add security measures around existing APIs
    - Cryptography
    - Redundancy
- Similar approaches: SSL, VPN

# Example 2: VPN

- 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


Image source: http://www.secunet.com/de/das-unternehmen/presse/bilddatenbank/

# SINA Box Overview

- Hardware:
  - Different kinds of networks interfaces:
    - Red: plaintext, no protection
    - Black: encrypted, MACs
- Tamper / EM protected casing
- Software:
  - Minimized and hardened Linux
  - Runs only from CD-ROM or Flash

# Linux is Complex

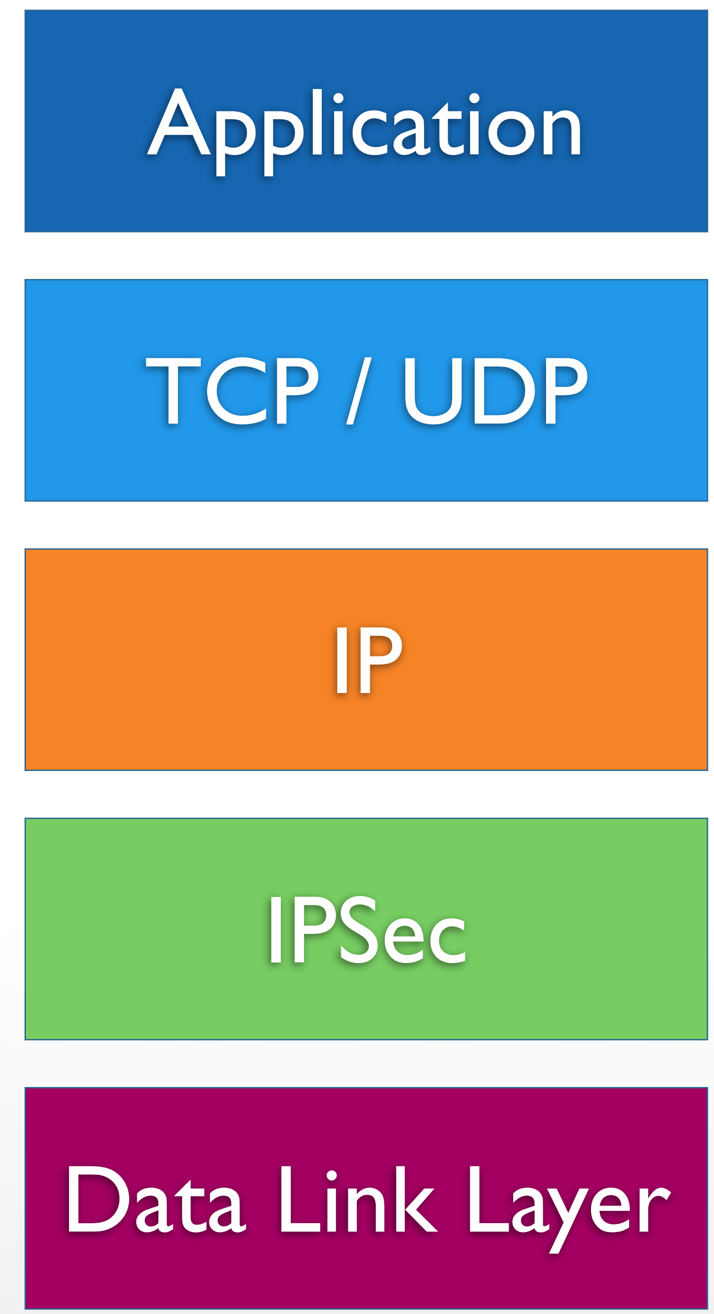
- Linux is complex!
- SLOC for Linux 2.6.18:
  - Architecture specific: 817.880
  - x86 specific: 55.463
  - Drivers: 2.365.256
  - Common: 1.800.587
- Typical config: ~ 2.000.000
- Minimized & hardened: > 500.000

# Mikro-SINA

- Research project Mikro-SINA:
  - Reduce TCB of VPN gateway
  - Enable high-level evaluation for high assurance scenarios
  - Ensure confidentiality and integrity of sensitive data within the VPN
  - Exploit microkernel architecture

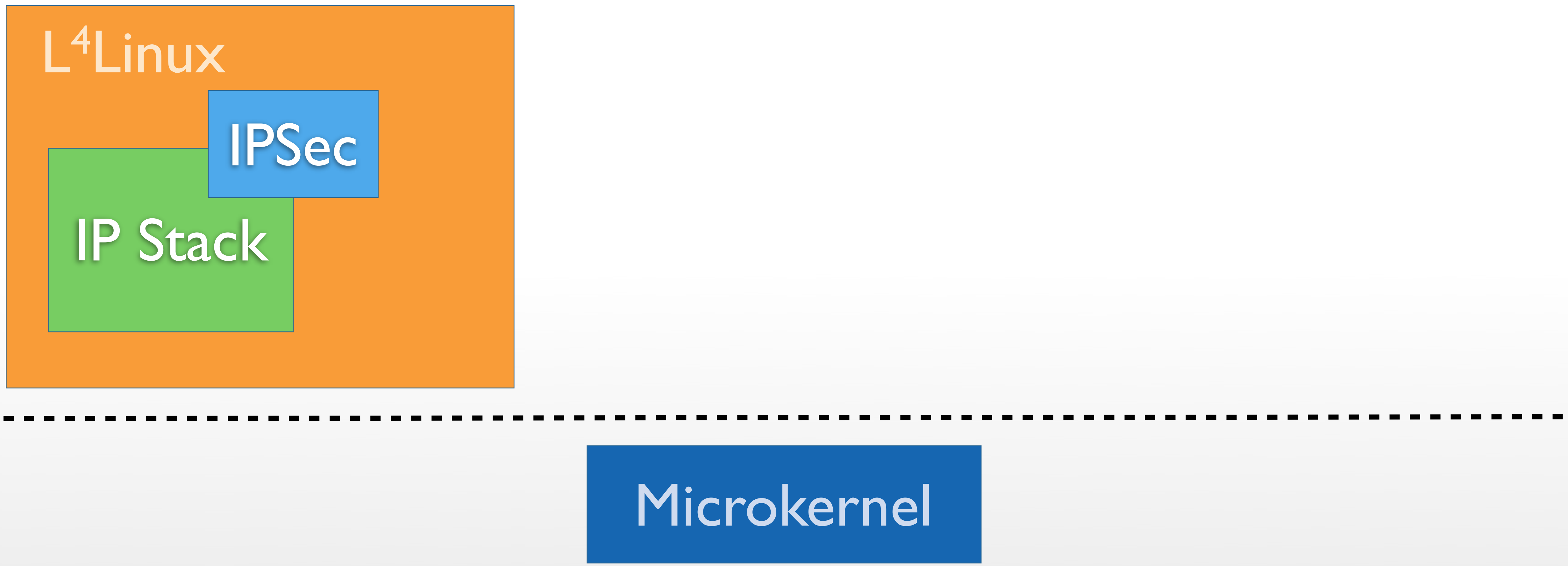
# IPSec in a Nutshell

- Protocol suite for securing IP-based communication
- Authentication header (**AH**)
  - Integrity
  - Authentication
- Encapsulating Security Payload (**ESP**)
  - Confidentiality
- Tunnel mode / transport mode



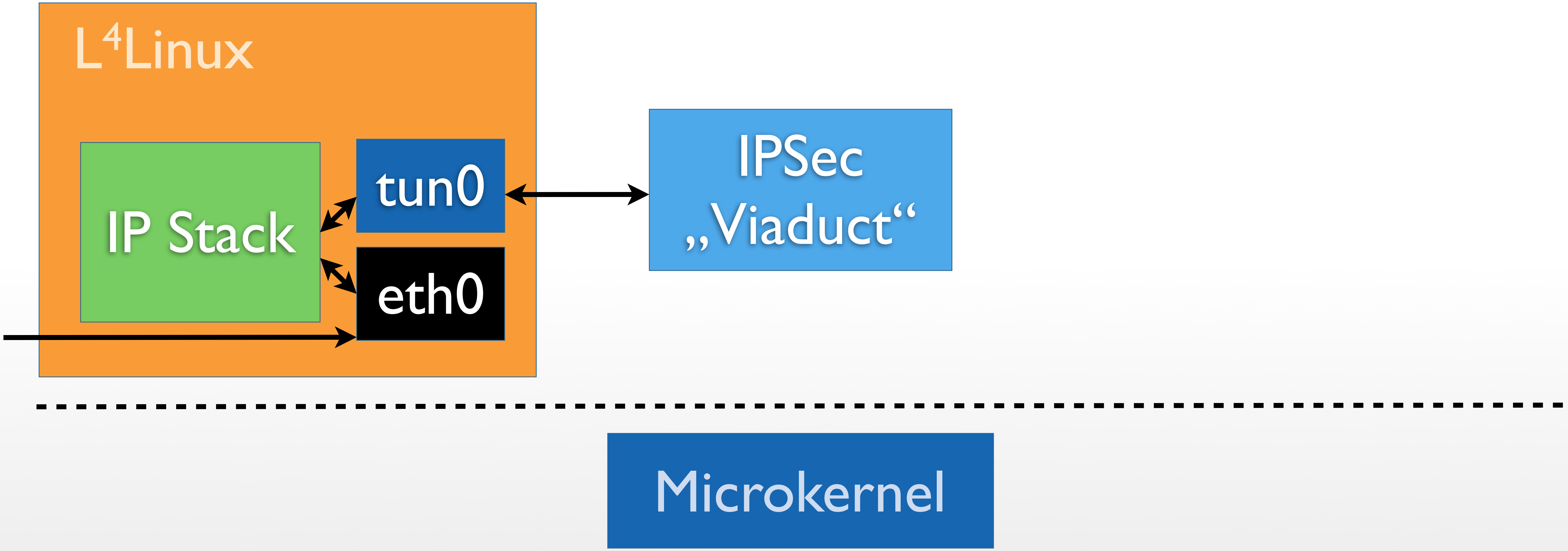
# IPSec in $L^4$Linux

- IPSec is security critical component
- … but is integrated into Linux kernel



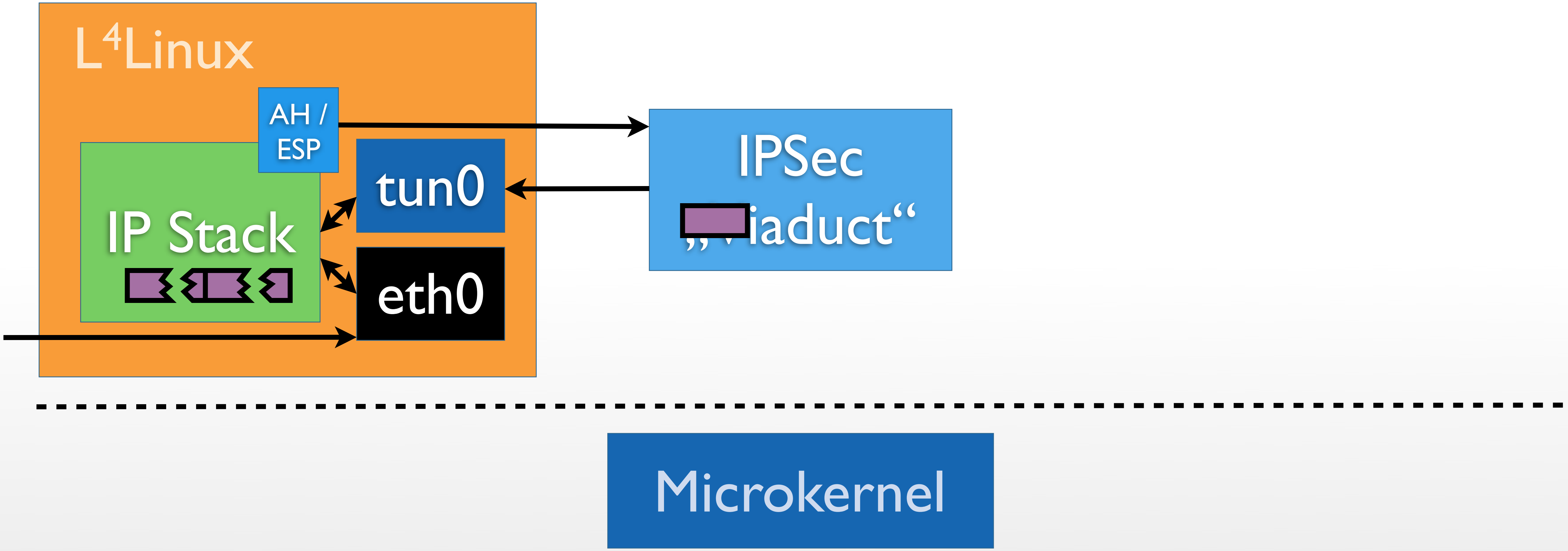
# IPSec „Viaduct"

- Better: isolate IPSec in „Viaduct"
- IPSec packets sent/received through TUN/TAP device



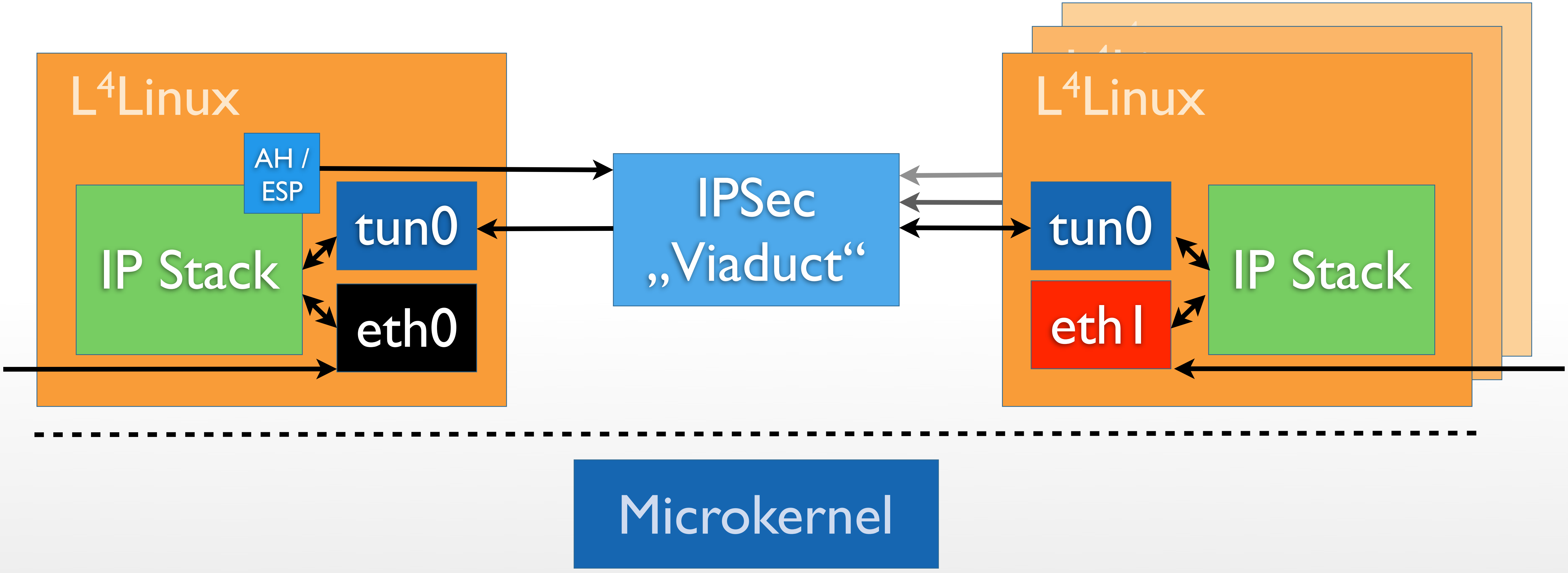
# Fragmentation

- Problem: Routers can fragment IPSec packets on the way
- Let L4Linux reassemble them



# Confidentiality

- Untrusted L4Linux must not see both plaintext and encrypted data
- Dedicated L4Linux for black/red networks

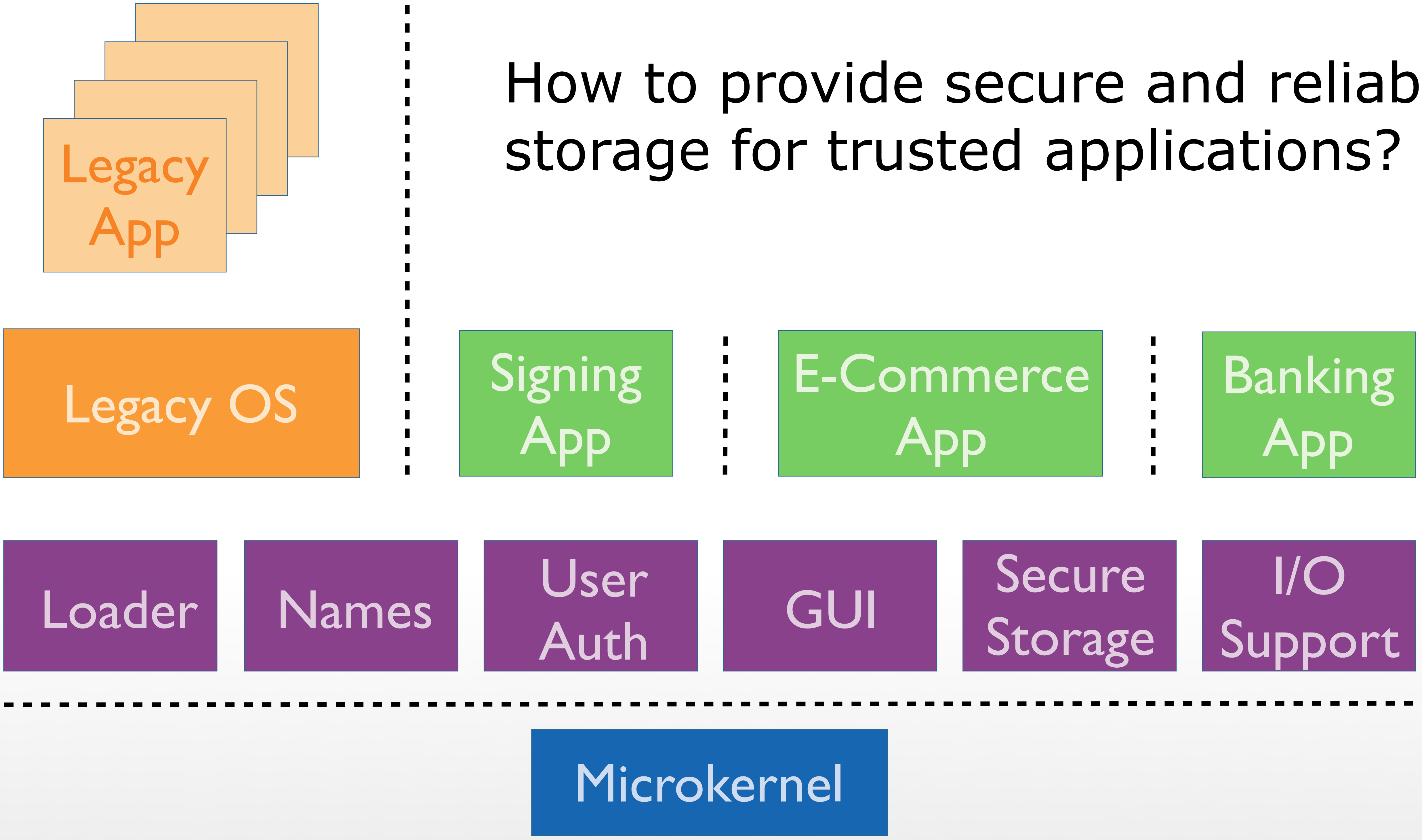


# Mikro-SINA Results

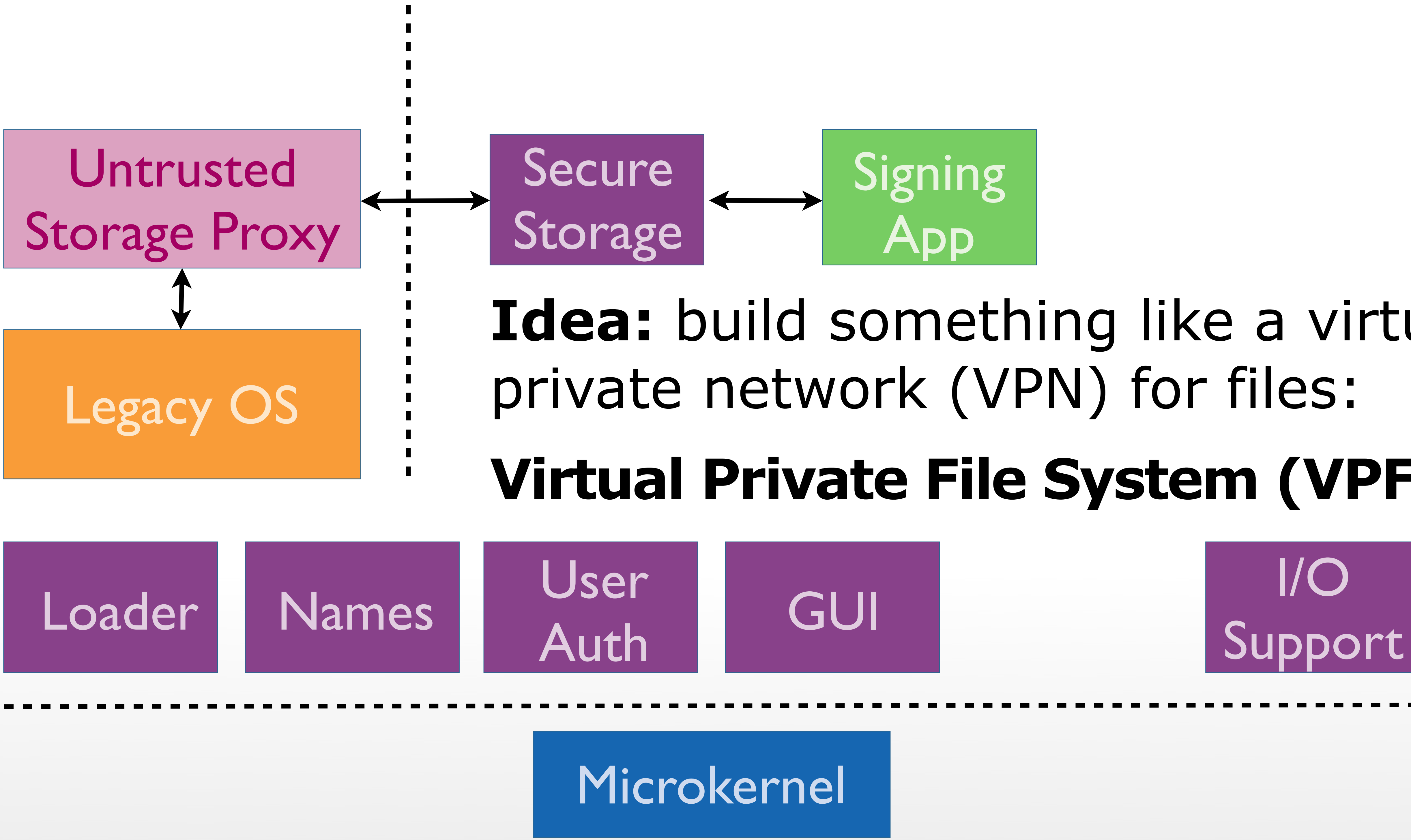
- Trusted wrapper for VPN
- Small TCB:
  - 5.000 SLOC for „Viaduct"
  - Fine grain isolation
  - Principle of least privilege
- Extensive reuse of legacy code (Drivers, IP stack, ...)
- More details in [5]

# Example 3: Storage

How to provide secure and reliable storage for trusted applications?



# Split File System

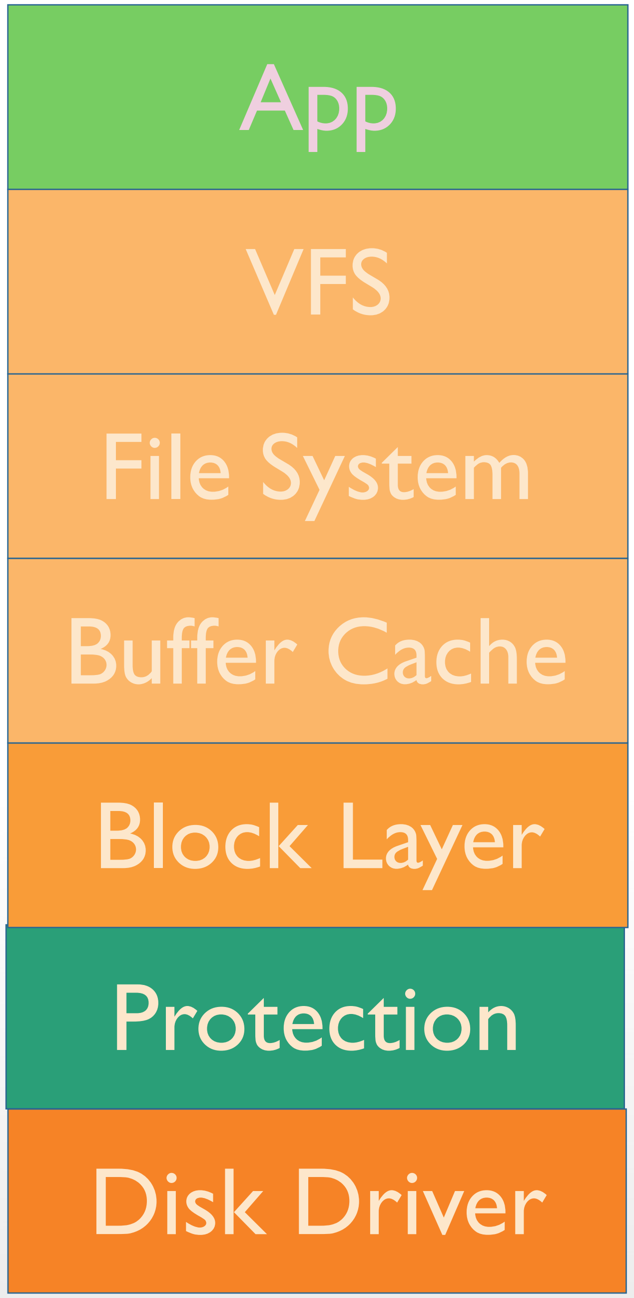



**Idea:** build something like a virtual private network (VPN) for files:

**Virtual Private File System (VPFS)**

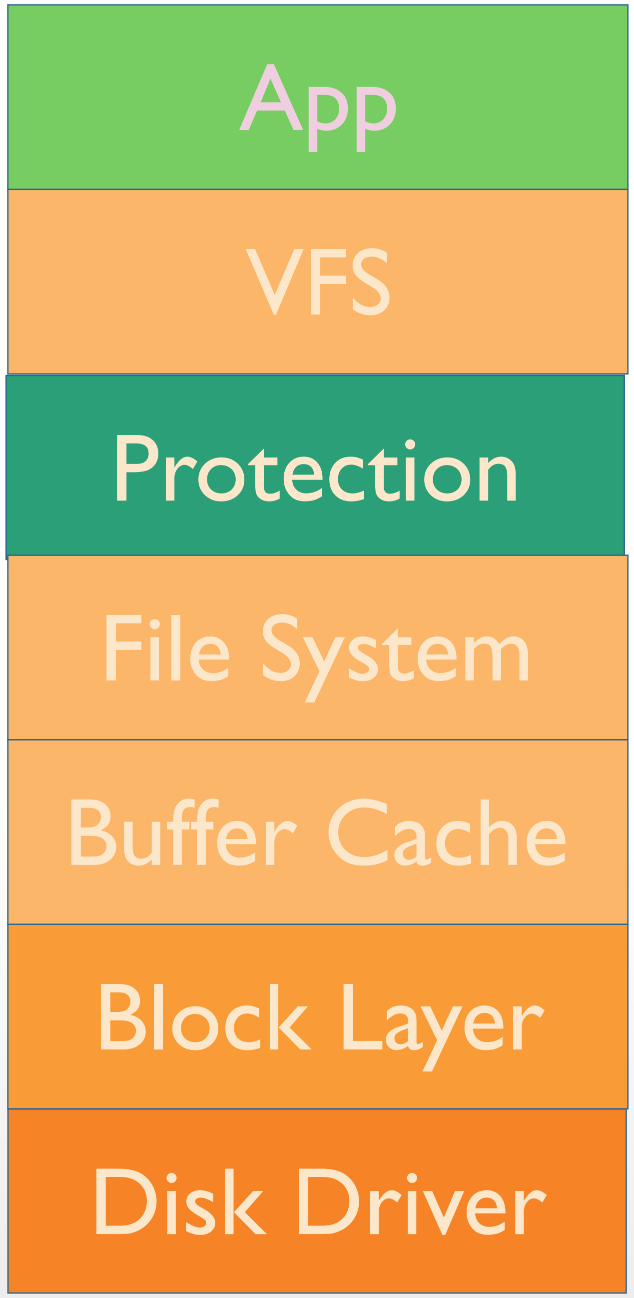
# Design Space

- First end of design space: Protect whole file system at block layer:
  - Common solution (e.g., dm_crypt in Linux)
  - Easy protection for all data
  - Requires a complete file system in the TCB
  - Limit reuse

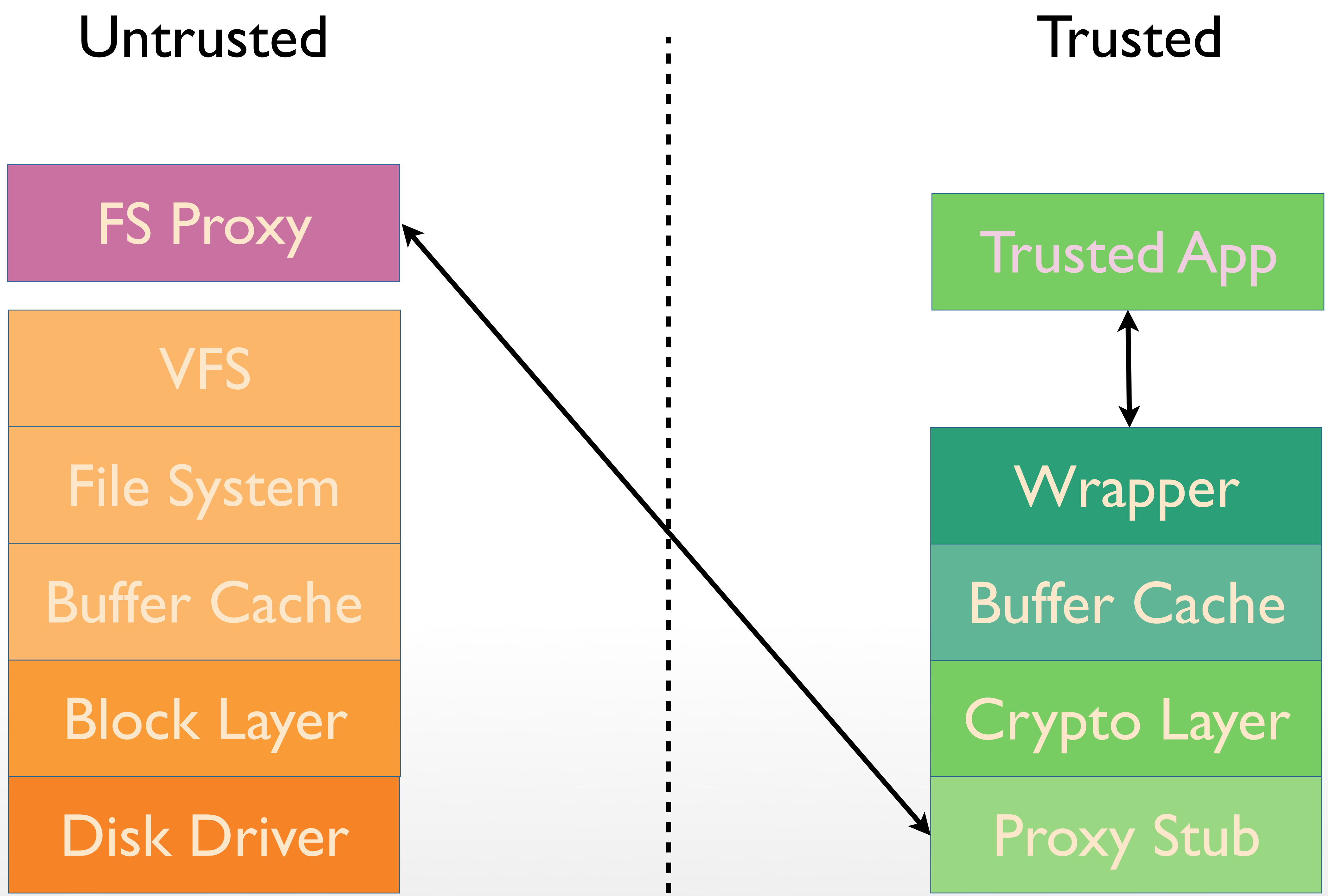


# Design Space

- Second end of design space: Protect individual files near VFS / API layer:
  - Stacked file system (e.g., ecryptfs in Linux)
  - Flexible protection policies
  - Most parts of file system stack not part of TCB
  - Ideal for trusted wrapper



# VPFS Architecture

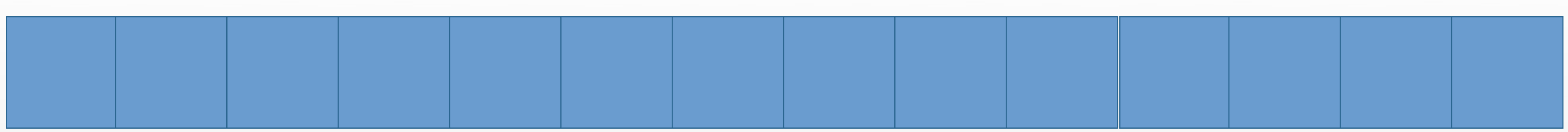


# Protection Goals

- **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 from trusted backup

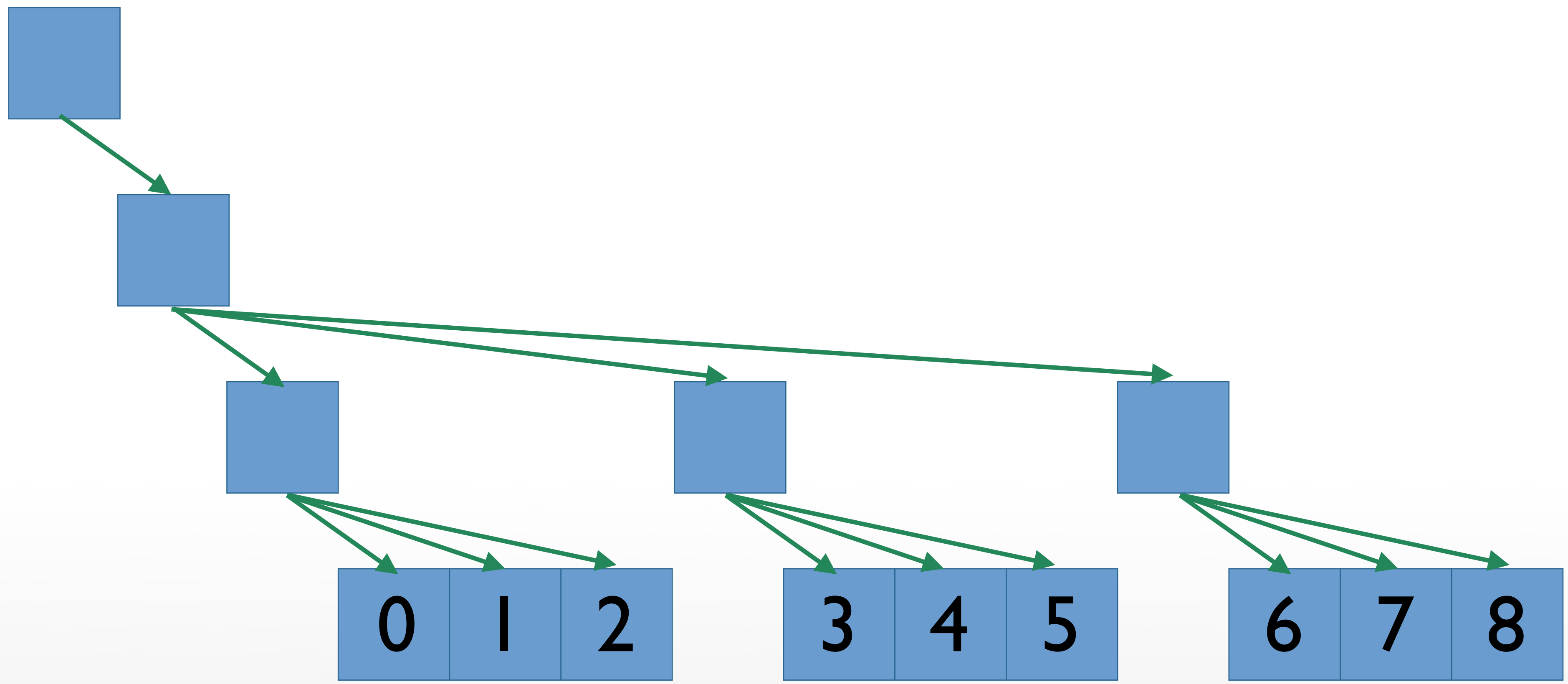
# File Encryption

- Files in untrusted legacy file system are arrays of encrypted blocks
- Trusted part of VPFS takes care of encryption / decryption on the fly
- Only buffer cache contains plaintext

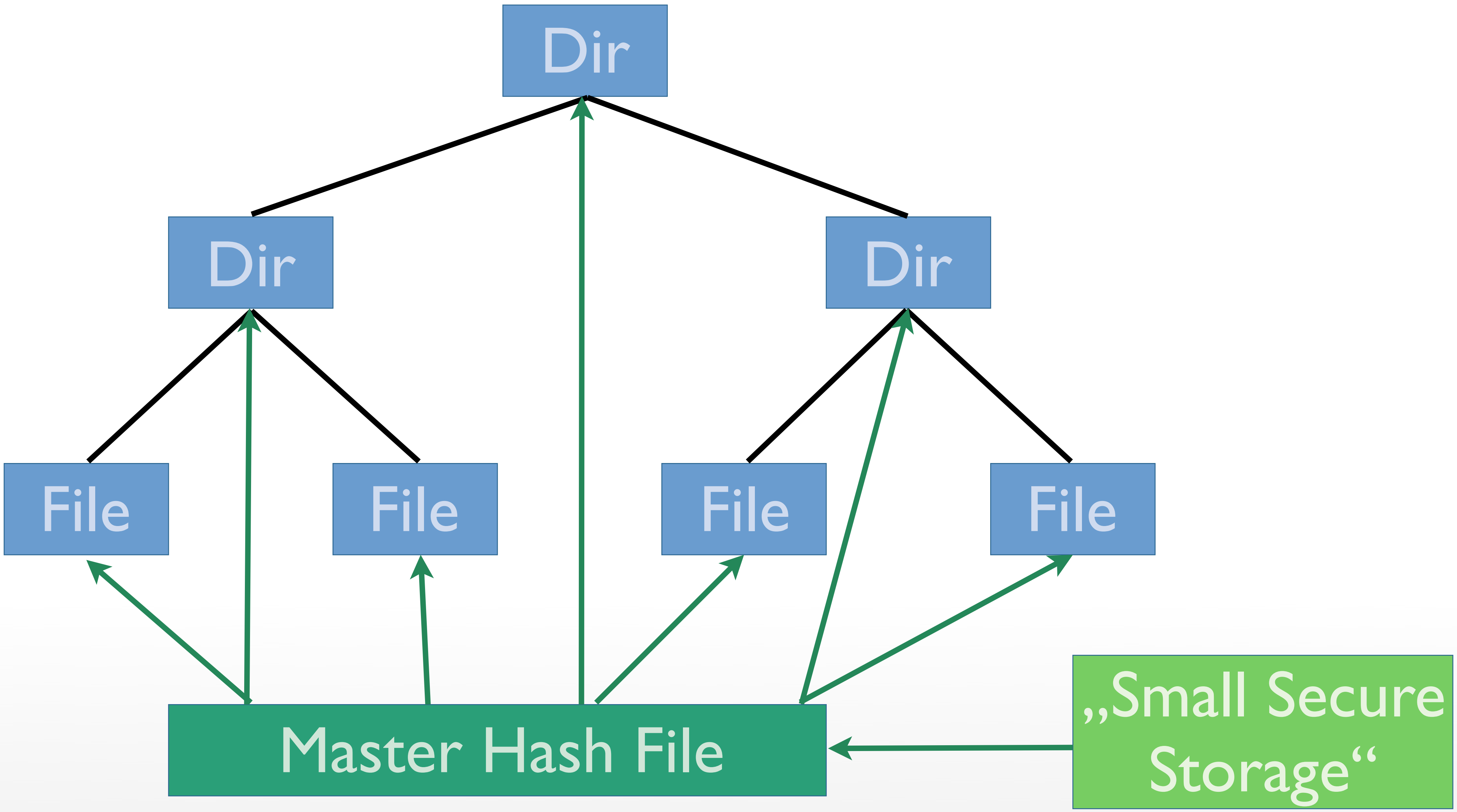


# File Integrity

- Hash tree embedded in files
- Parents authenticate child nodes



# Meta Data Integrity



# Maximizing Reuse

- VPFS reuses Linux file system stack:
  - Drivers, block device layer
  - Optimizations (buffer cache, read ahead, write batching, ...)
  - Allocate / free disk storage for files
- What about metadata?
  - Naming and lookup functionality
  - Directories and hierarchies

# Validity Checks

- How to trust untrusted meta data?
  - „File exists" / „File does not exist":
    - Validated inside TCB using cryptographic proof and hash tree
    - Efficient solution possible
  - Directory listings:
    - Efficient solution requires functionality to be implemented in TCB
- Details in [3]

# VPFS Summary

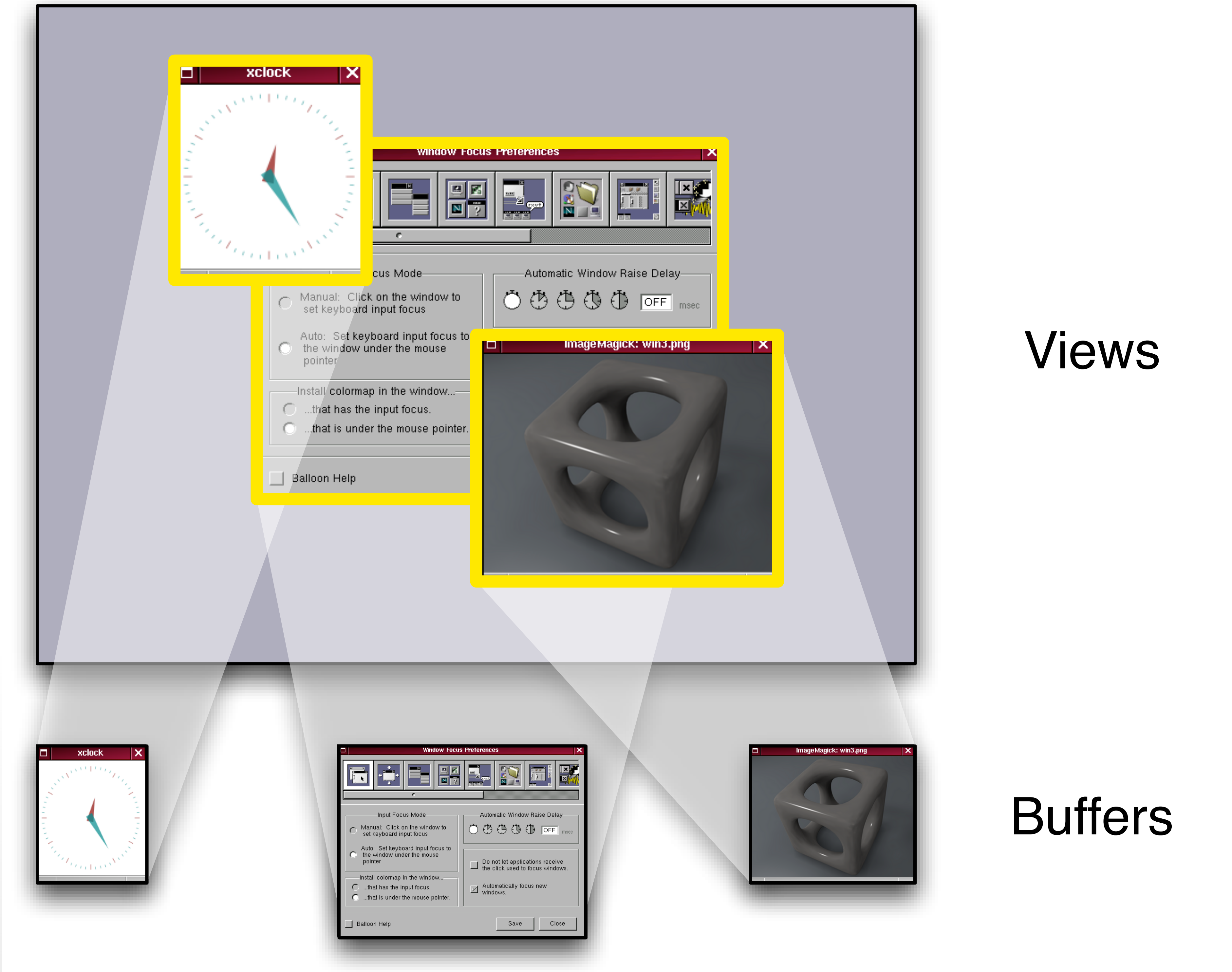
- Trusted wrapper shown to work for file systems
- VPFS is general purpose file system
- Significant reduction in code size:
  - VPFS adds 4,000 to 4,600 SLOC to application TCB
  - Standard Linux file system stack comprises >50,000 SLOC

# User Interfaces

# Sharing The Screen

- Isolated applications run in different domains of trust, but separate screens are inconvenient
- The Nitpicker solution [4]:
  - 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

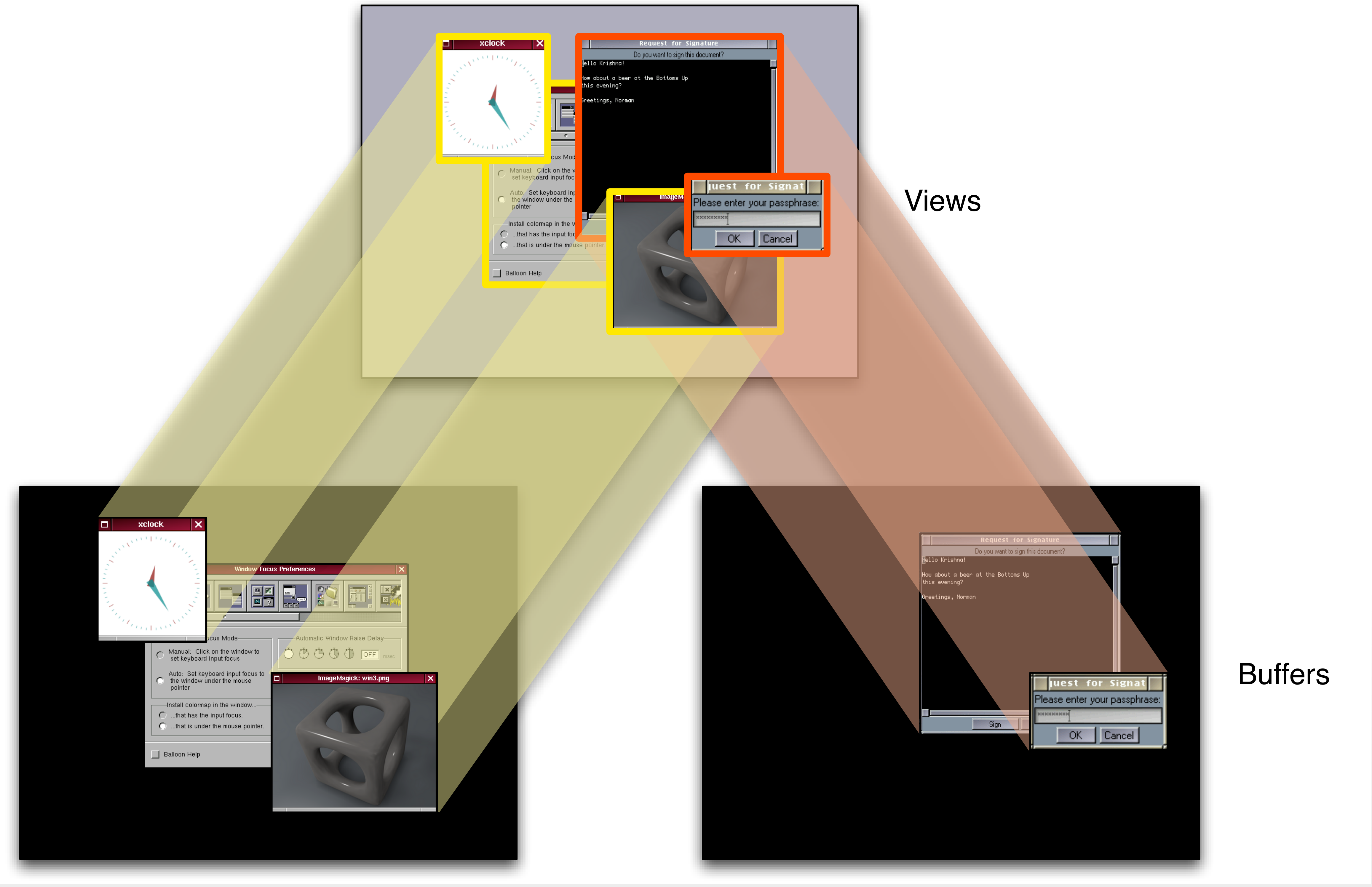
# Concepts



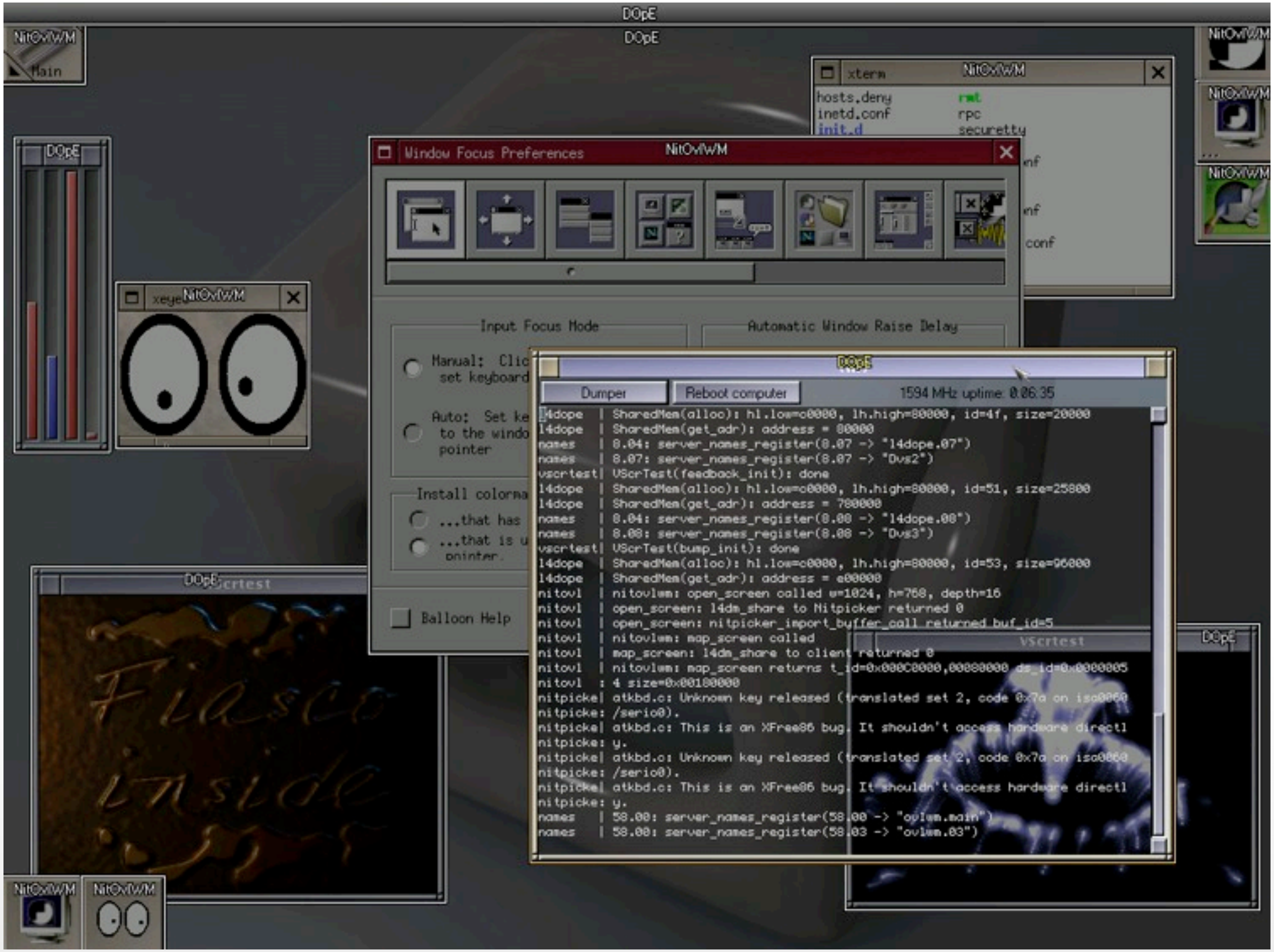

Views

Buffers

# How It Works


Views
Buffers
xclock
Request for Signature
Do you want to sign this document?
Hello Krishna!
How about a beer at the Bottoms Up this evening?
Greetings, Norman
Please enter your passphrase:
OK
Cancel
Window Focus Preferences
ImageMagick: win3.png
Balloon Help

# Nitpicker In Action



# Demo

# Summary

- Secure reuse of untrusted legacy infrastructure
- Splitting of applications and OS services to reduces size of TCB
- Nizza secure system architecture:
  - Strong isolation
  - Application-specific TCBs
  - Legacy Reuse
  - Trusted Wrapper

# Coming up next ...

- Next week, January 25th:
  - Lecture on *„Trusted Computing"*:
    - Where does VPFS store its secrets?
    - How does VPFS detect corrupt data?
    - How can we trust in what Nitpicker shows on the screen?
- Also next week:
  - Exercise „Capability Systems"

# References


- [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 2008, 2008, Glasgow, Scotland UK
- [4] 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
- [5] Christian Helmuth, Alexander Warg, Norman Feske, „Mikro-SINA - Hands-on Experiences with the Nizza Security Architecture“, D.A.CH Security 2005, 2005, Darmstadt, Germany
- [6] http://support.apple.com/kb/HT4013
- [7] http://support.apple.com/kb/HT3754
- [8] http://jailbreakme.com