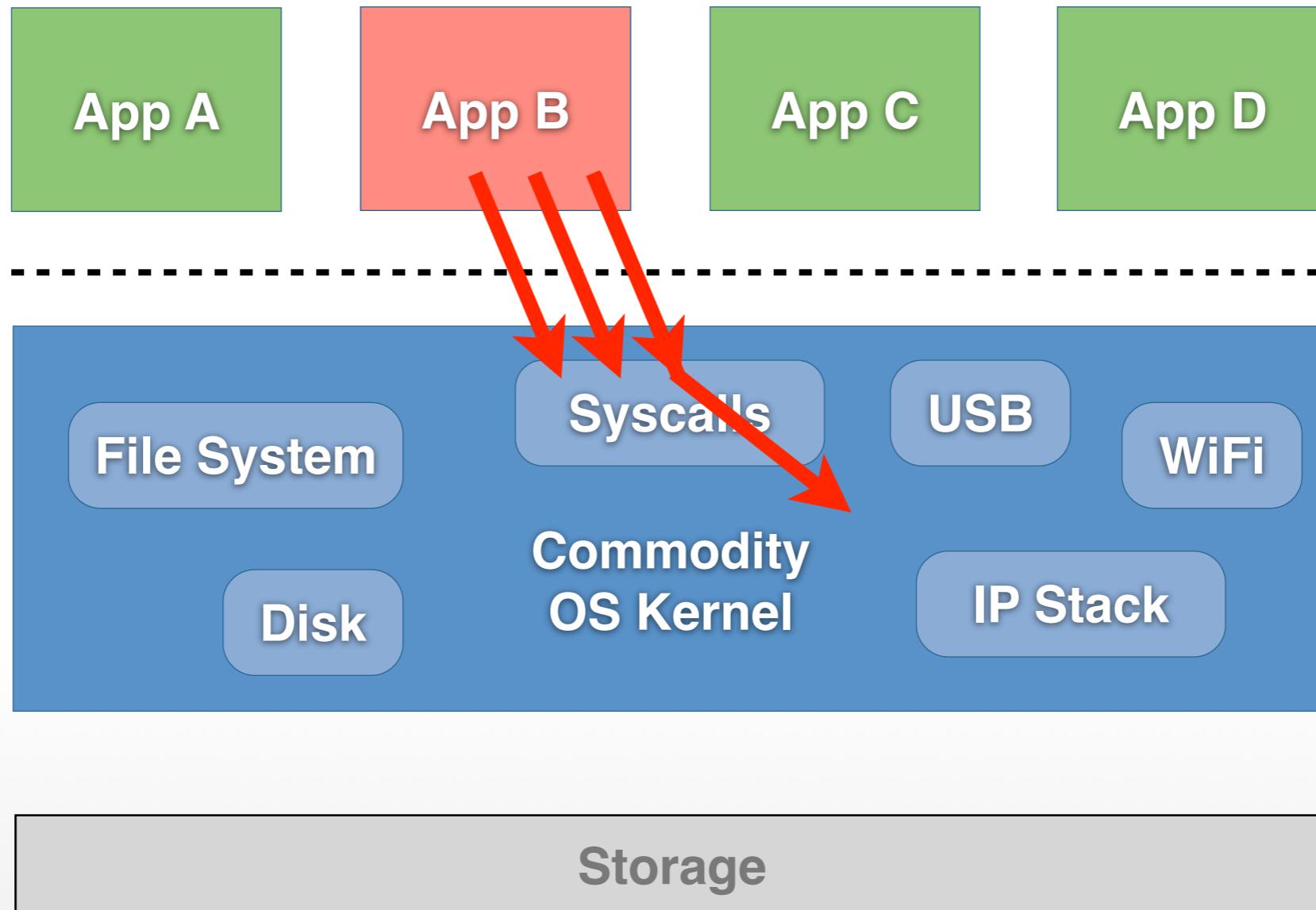


# SECURITY ARCHITECTURES

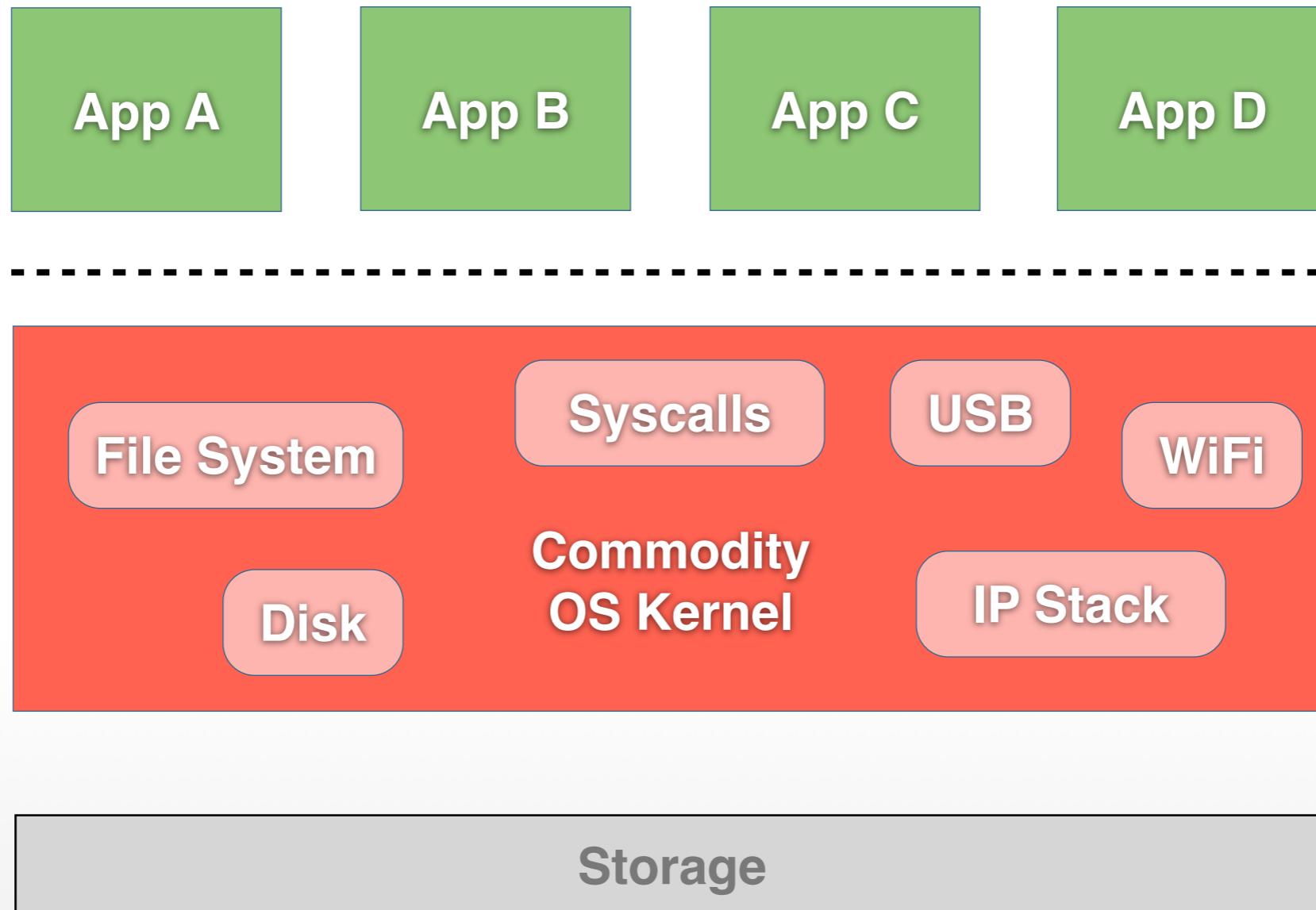
CARSTEN WEINHOLD

# CLASSICAL ARCHITECTURES

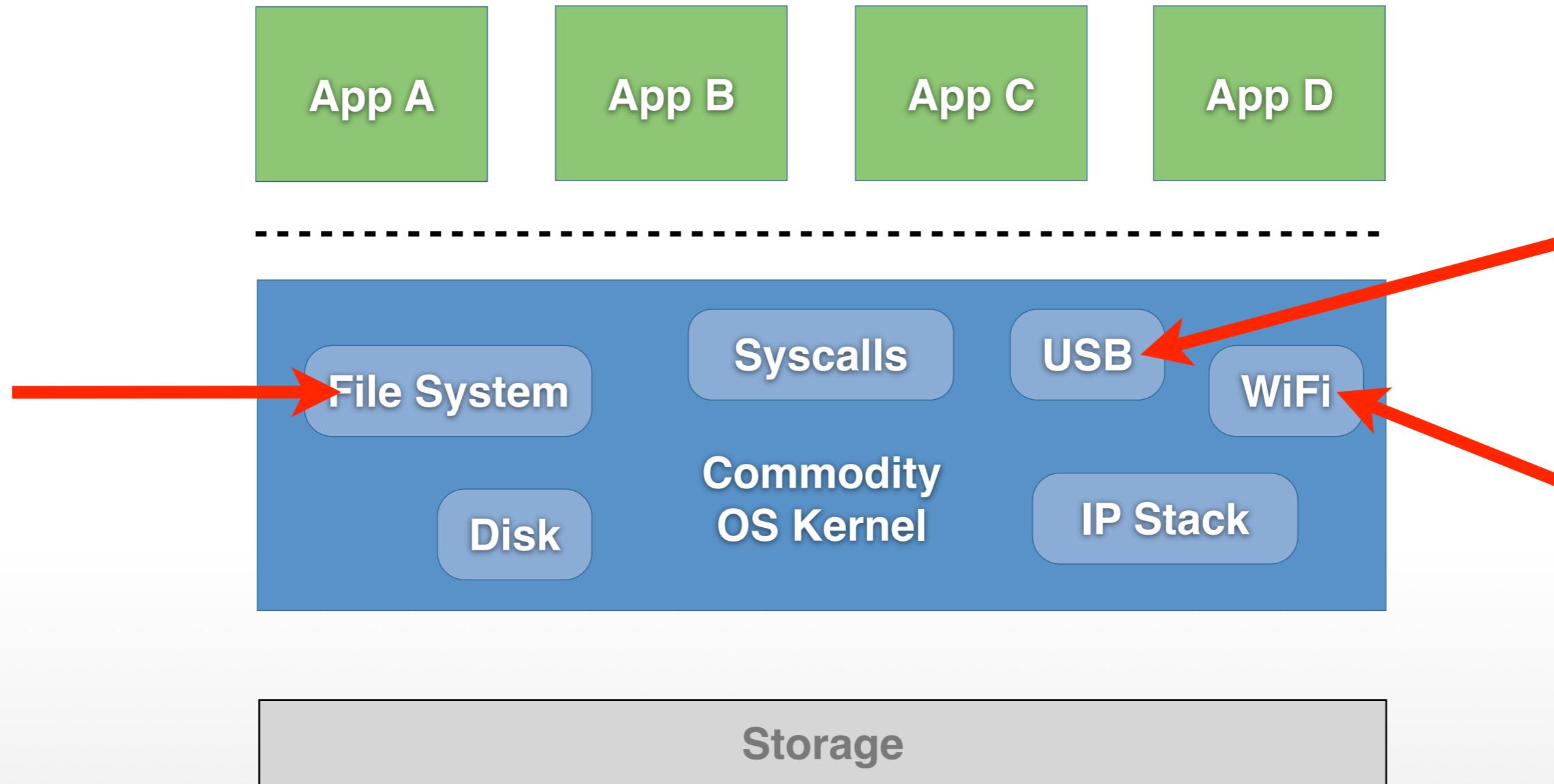
- Isolation in commodity OSes for PCs:
  - Based on user accounts
  - Same privileges for all apps
  - No isolation within applications
  - Permissive interfaces (e.g., ptrace to manipulate other address spaces)



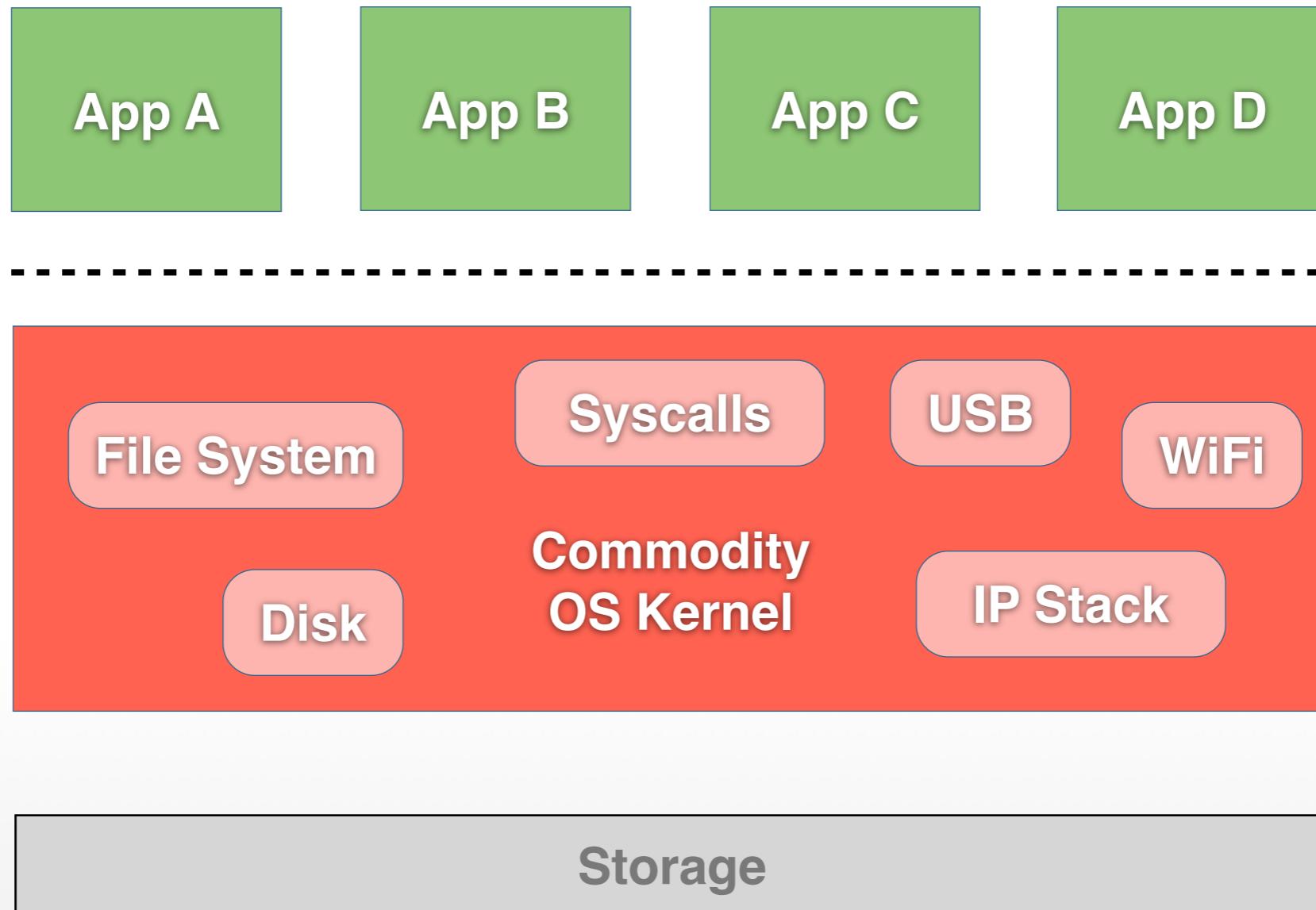
# KERNEL ATTACK



# KERNEL ATTACK

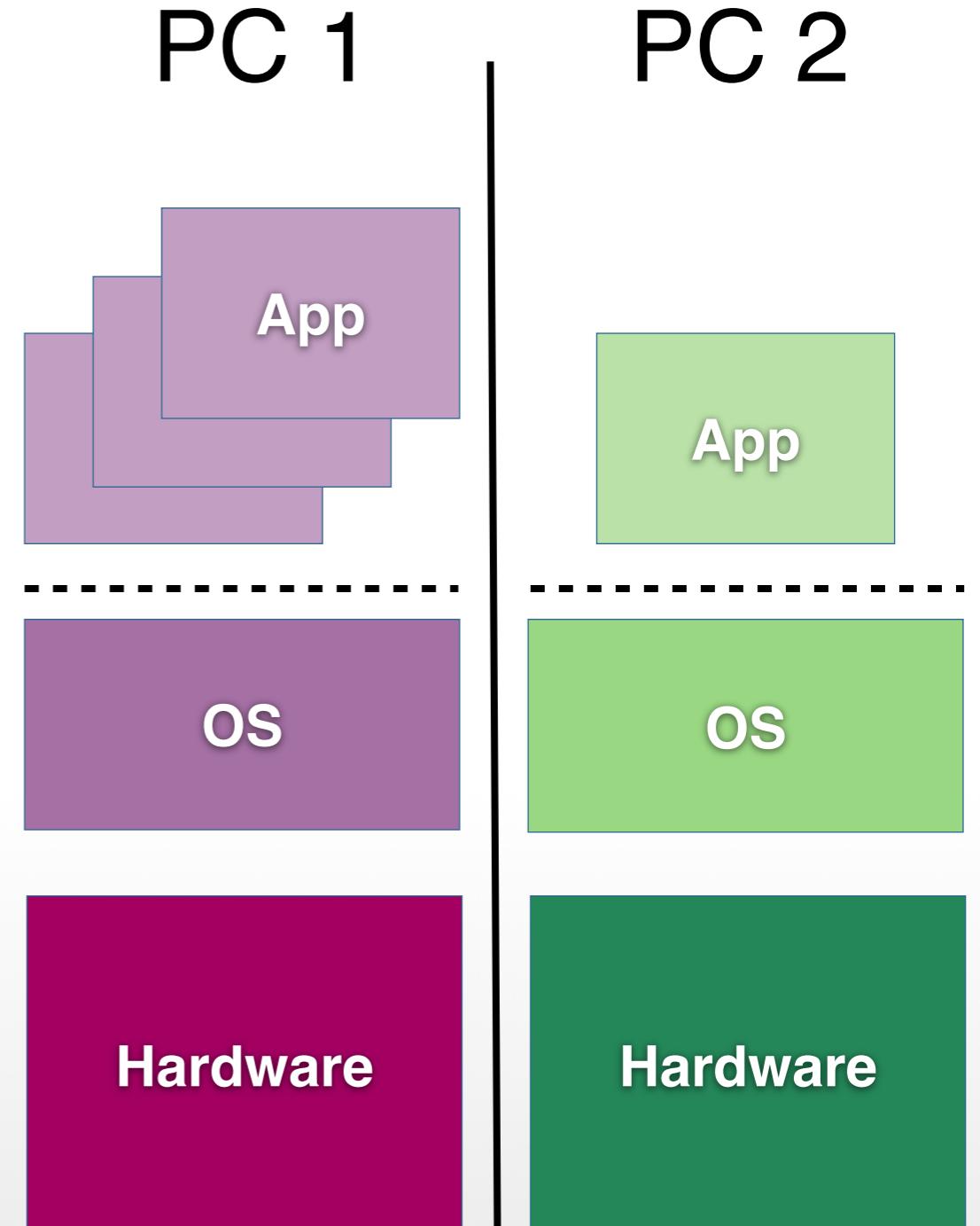


# KERNEL ATTACK

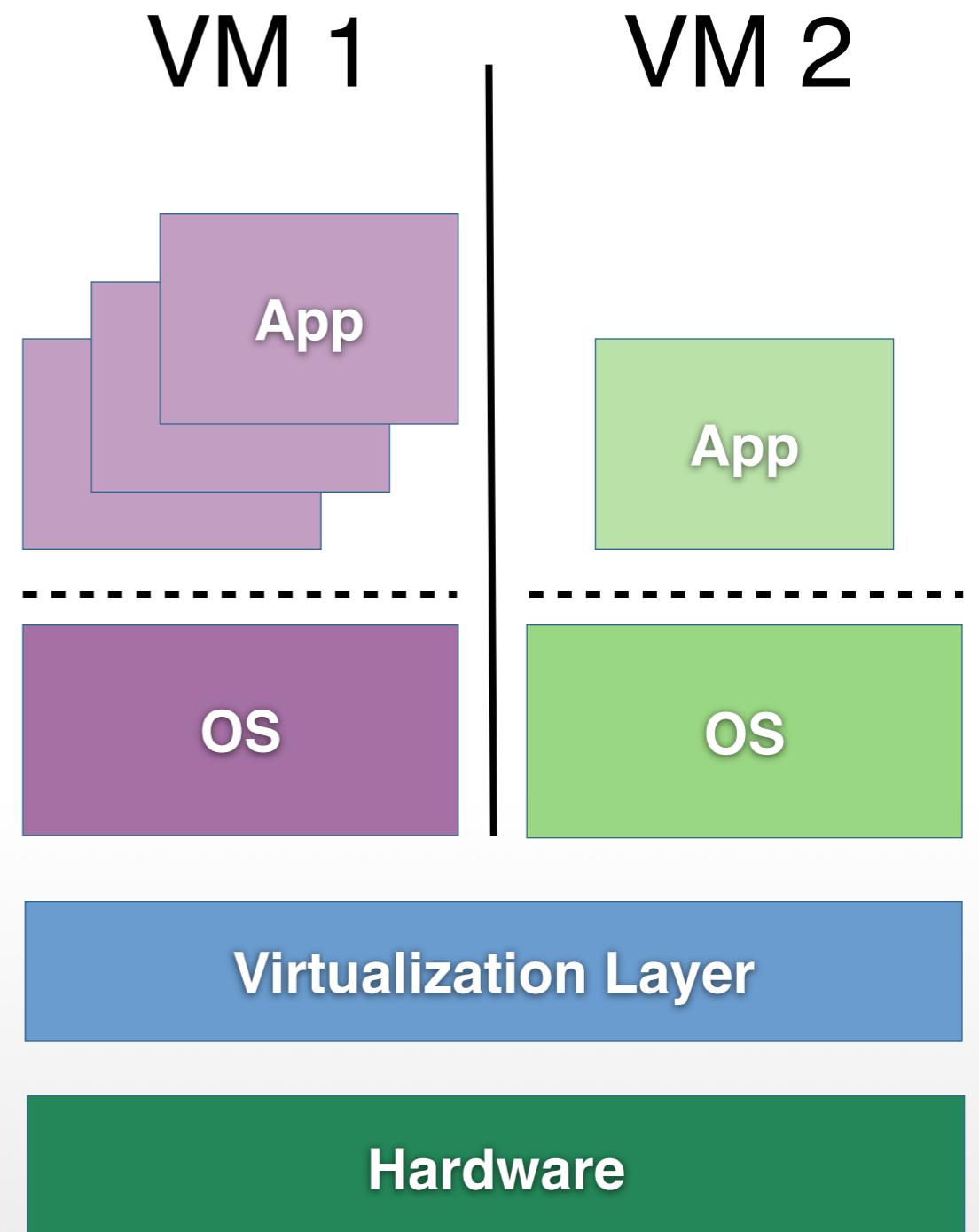


- Isolation in commodity OSes for PCs:
  - Based on user accounts
  - Same privileges for all apps
  - No isolation within applications
  - Permissive interfaces (e.g., ptrace to manipulate other address spaces)
- Efforts to restrict privileges:
  - SELinux, AppArmor, Seatbelt, ...
  - Linux containers,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

- 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



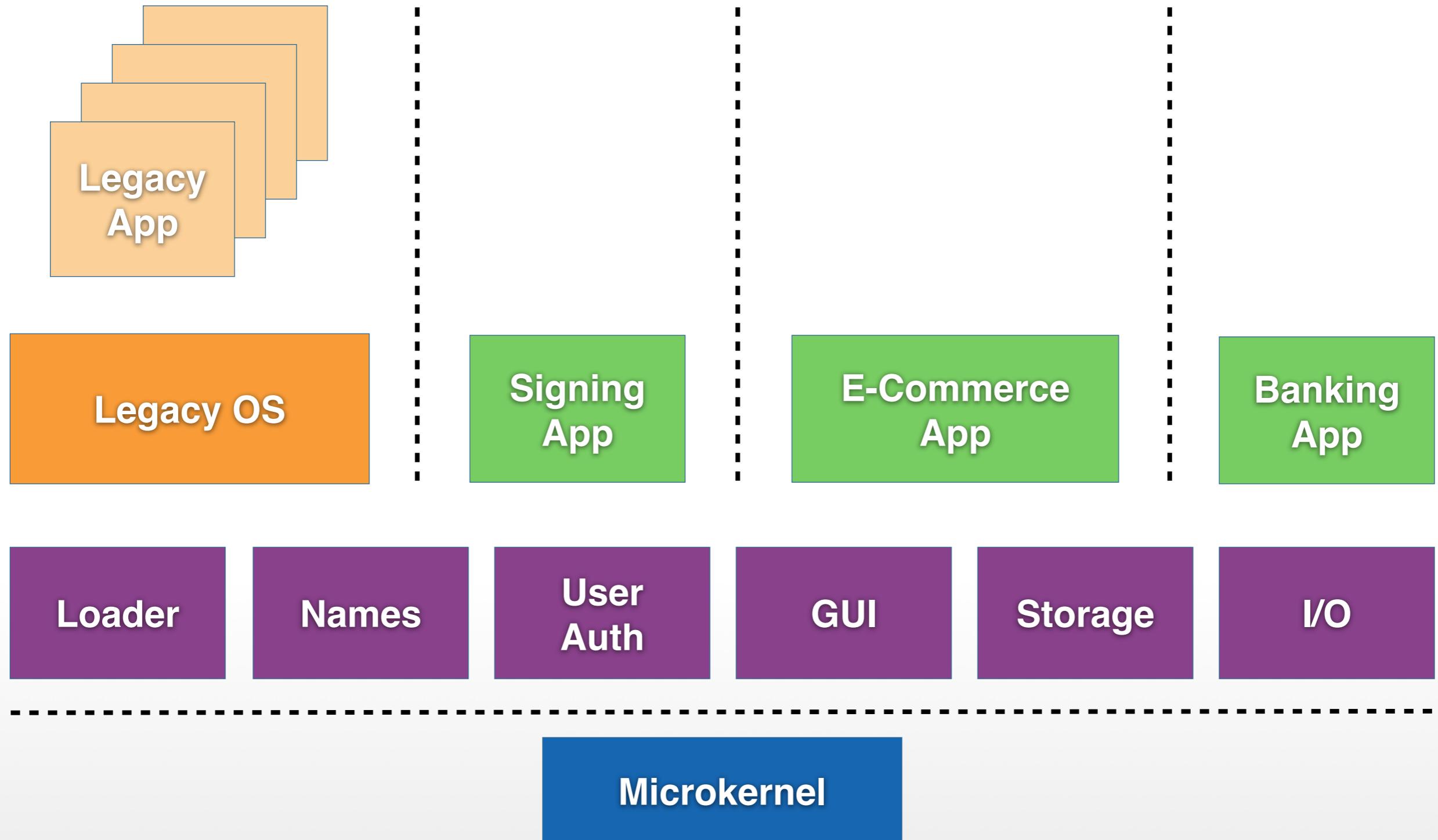
# WHAT IS THE PROBLEM?

- 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 Trusted Computing Base

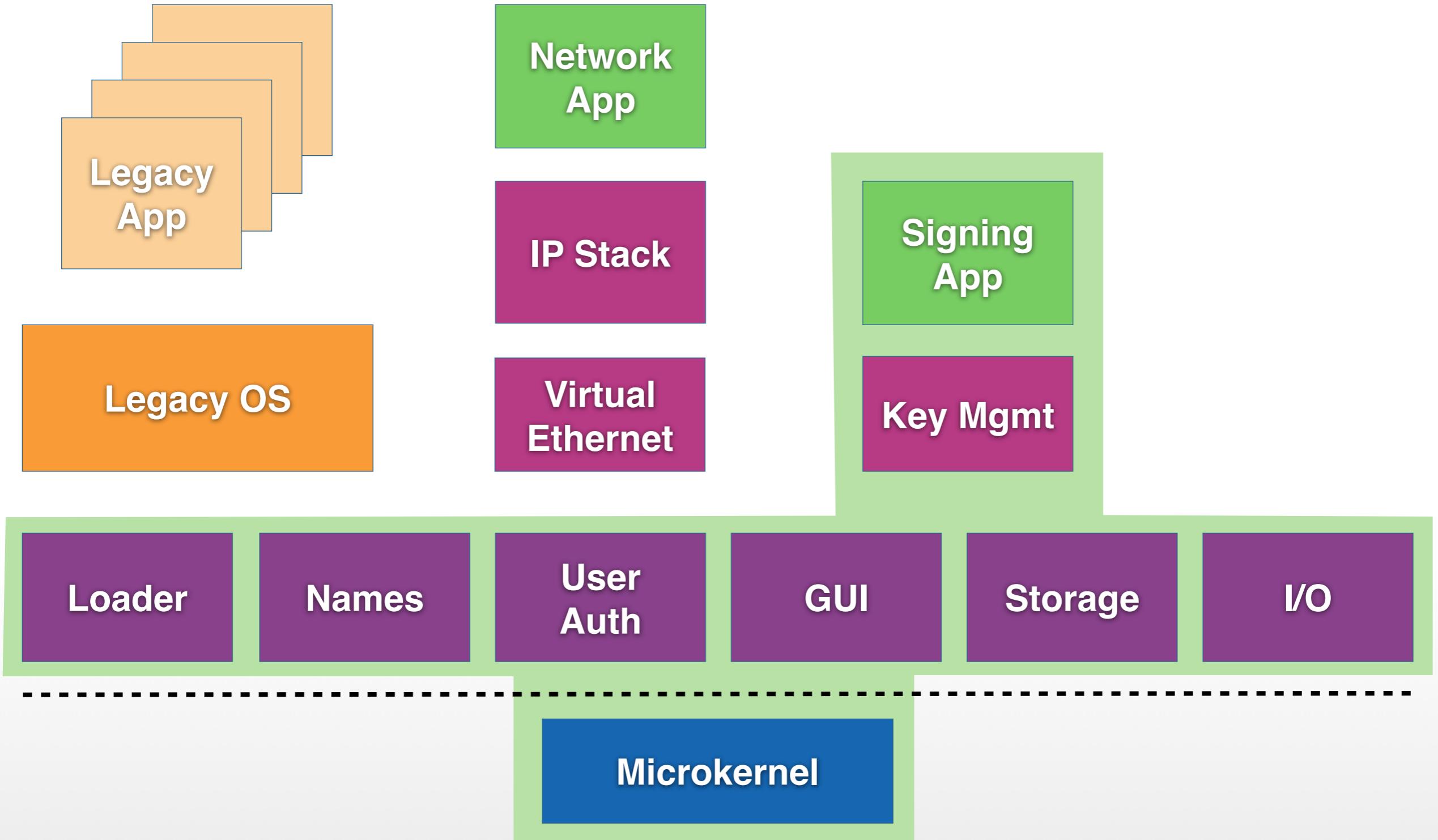
- To improve security: Reduce size of TCB = smaller attack surface
- First (incomplete) idea:
  - Remove huge legacy OS from TCB
  - Port application to microkernel-based multi-server OS
  - Remove unneeded libc backends, etc.
  - Possible approaches discussed in lecture on „Legacy Reuse“



## **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: „**What You See Is What You Sign**“
  - After receiving: verify signature, identify sender

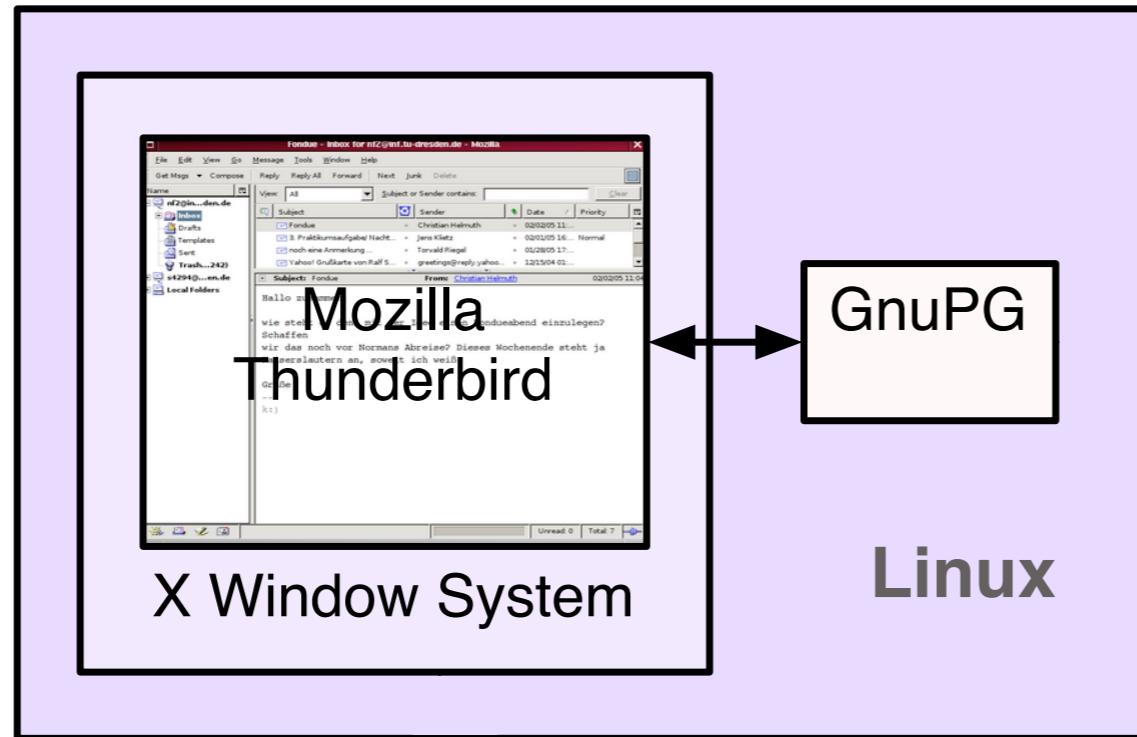


Image source: [5]

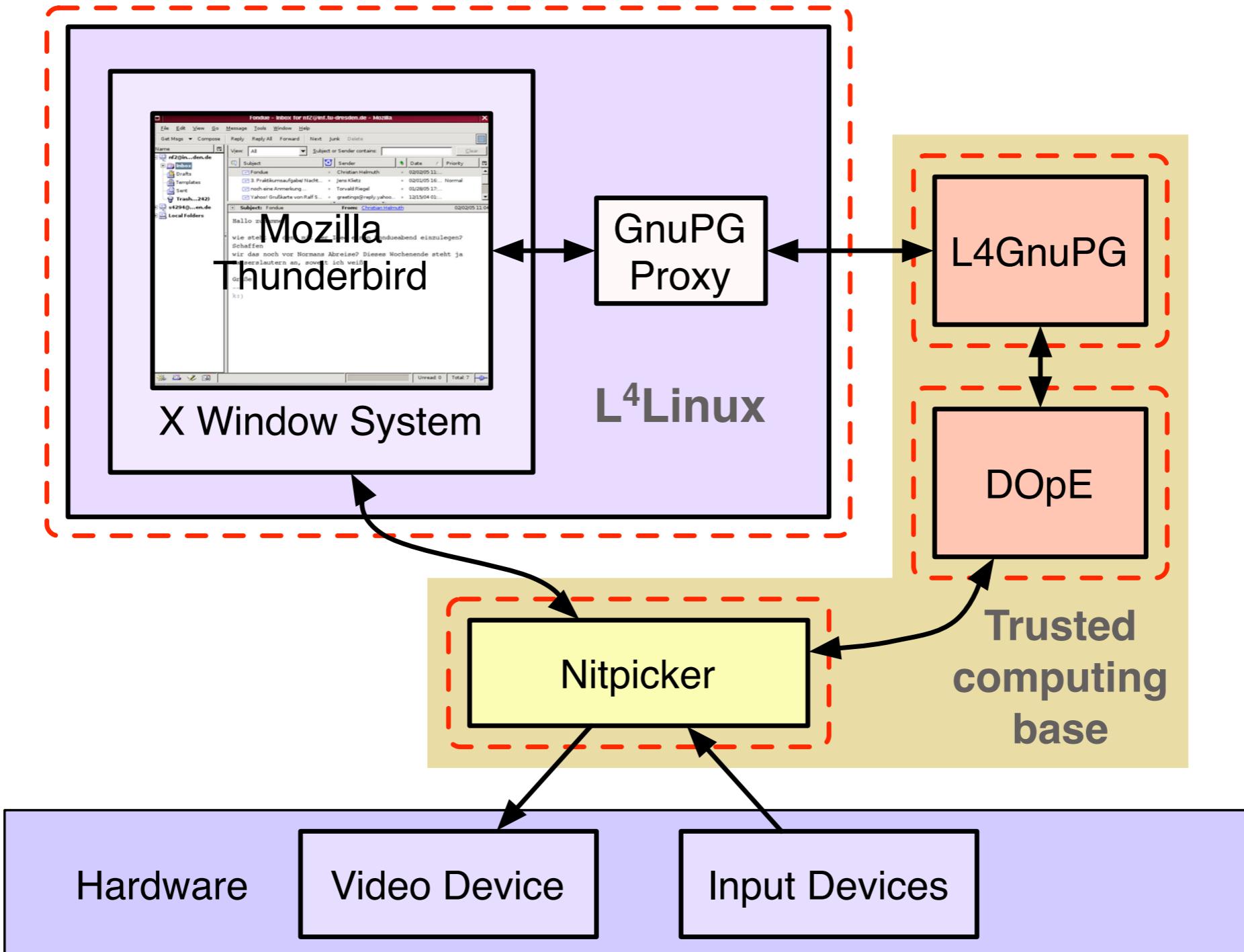


Image source: [5]

- 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

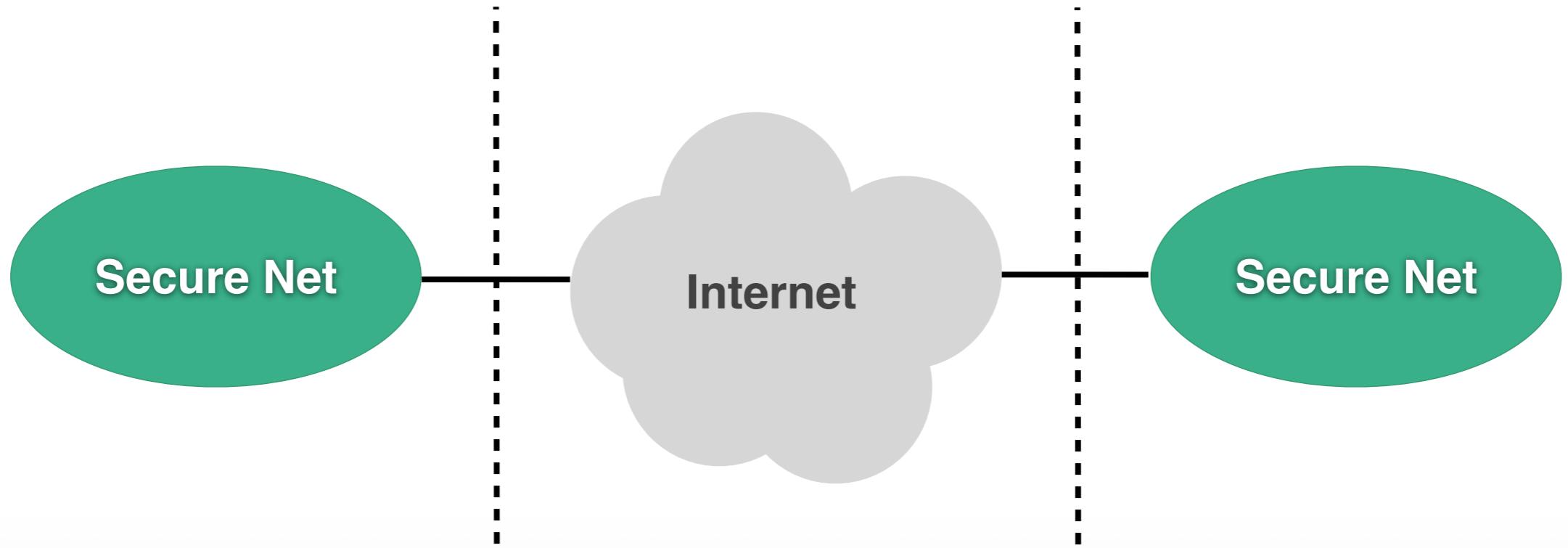
- Run legacy OS in VM
- Reuse service: net, files, ...
- Legacy infrastructure isolated from applications
- But:
  - Applications still depend on legacy services ... in TCB?
  - Interfaces reused, security issues as well?



- 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 [11]
  - Bugs in BSD Jails [12]

- Complex I/O and protocol stacks should not be part of TCB (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, Integrity, ~~Availability~~



# CASE STUDY: SINA BOX

- 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/>

- Differently trusted network interfaces:
  - **Red:** plaintext, no protection
  - **Black:** encryption + authentication codes



- VPN Software:
  - Based on minimized and hardened Linux
  - Runs only from read-only storage

- 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

**Released date:  
20 Sep 2006**

- Linux is even more complex in 2024!
- SLOC for Linux 6.7.1:
  - Architecture specific: 1,729,519
  - x86 specific: 316,544
  - Drivers: 17,771,667
  - fs/btrfs: 106,335

**No need to update every year 😊**  
**It keeps growing.**

- 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

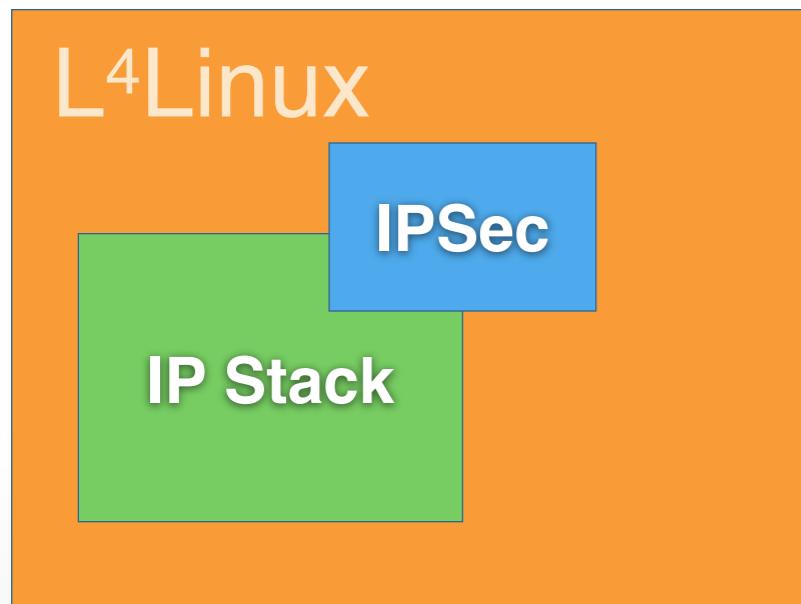
TCP / UDP

IP

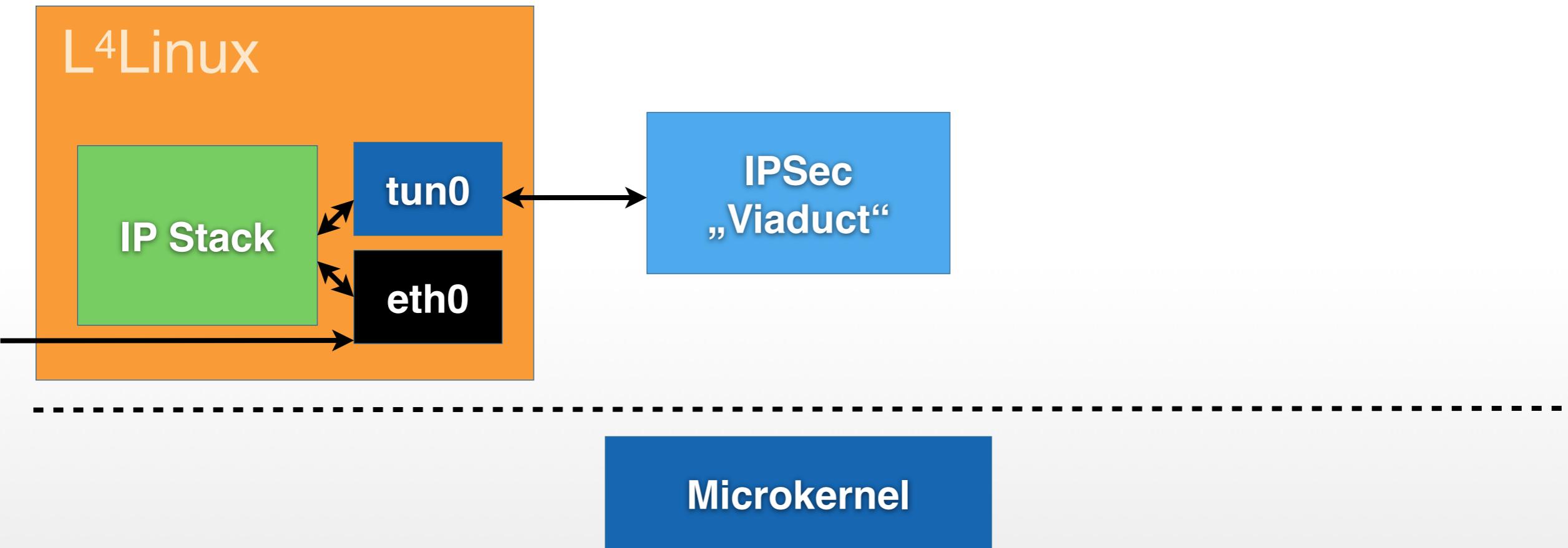
IPSec

Link Layer

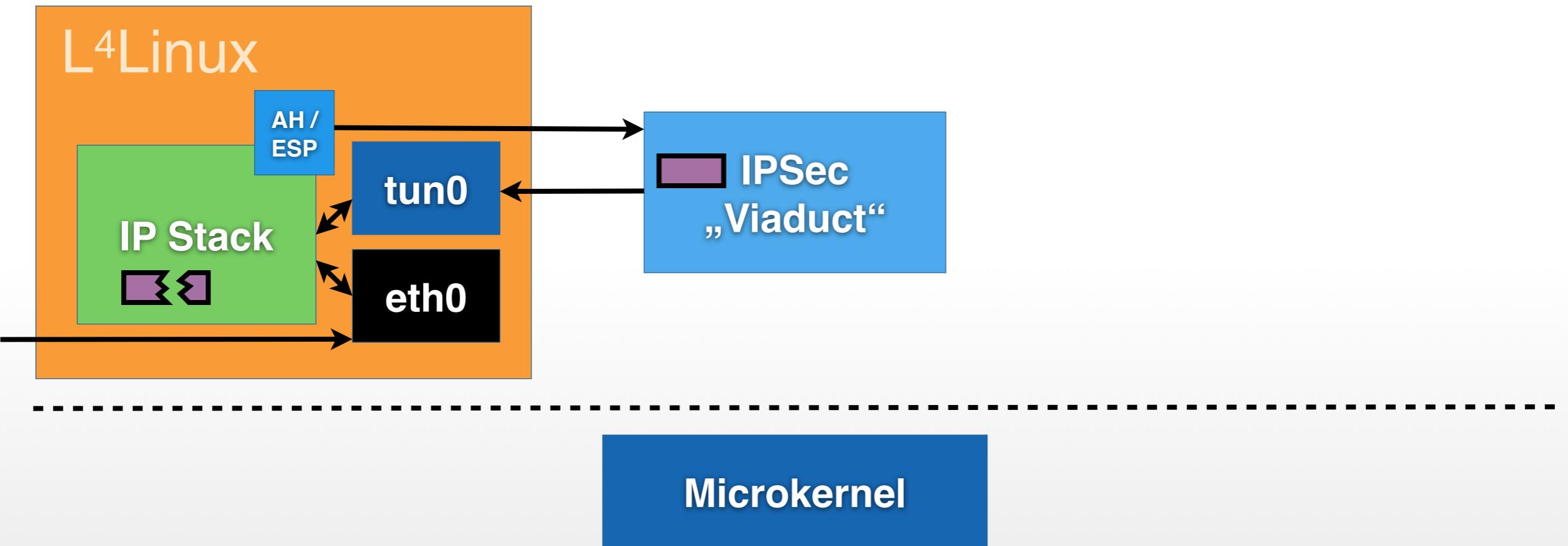
- IPSec is security critical component
- ... but is integrated into Linux kernel



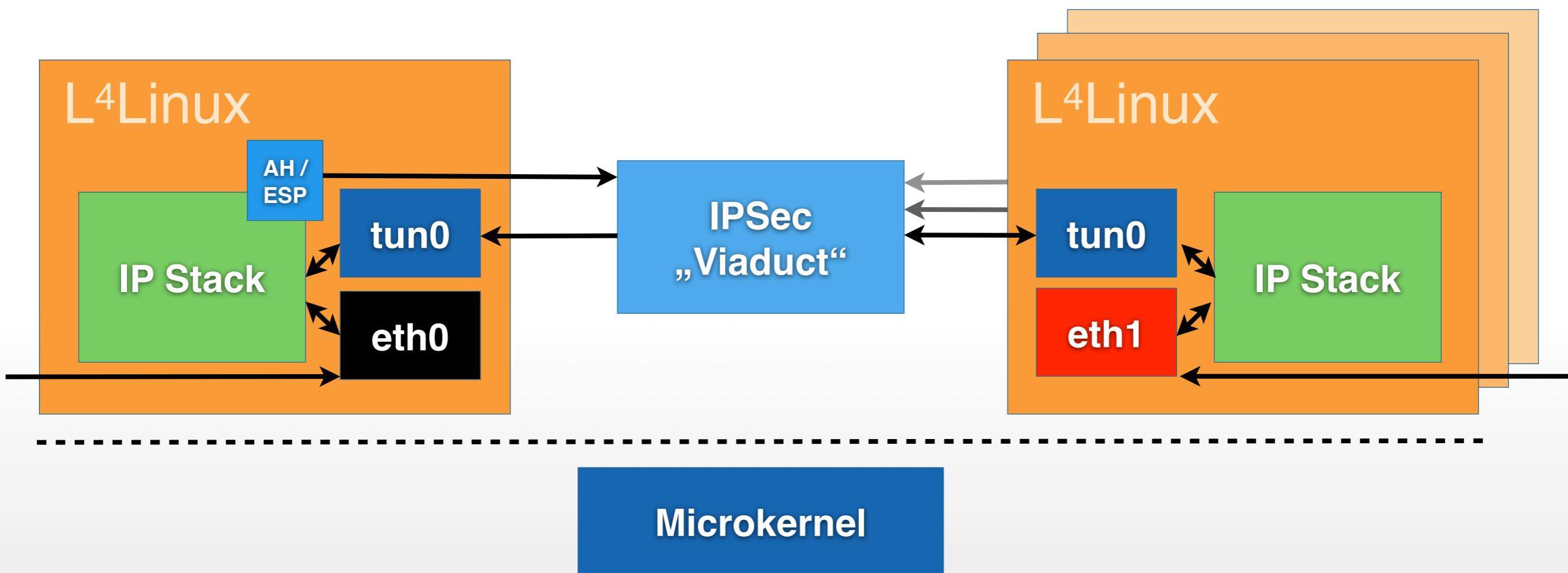
- **Idea:** Isolate IPsec in „Viaduct“
- IPsec packets sent/received through TUN/TAP device



- Problem: Routers can fragment IPSec packets on the way
- Let L<sup>4</sup>Linux reassemble them

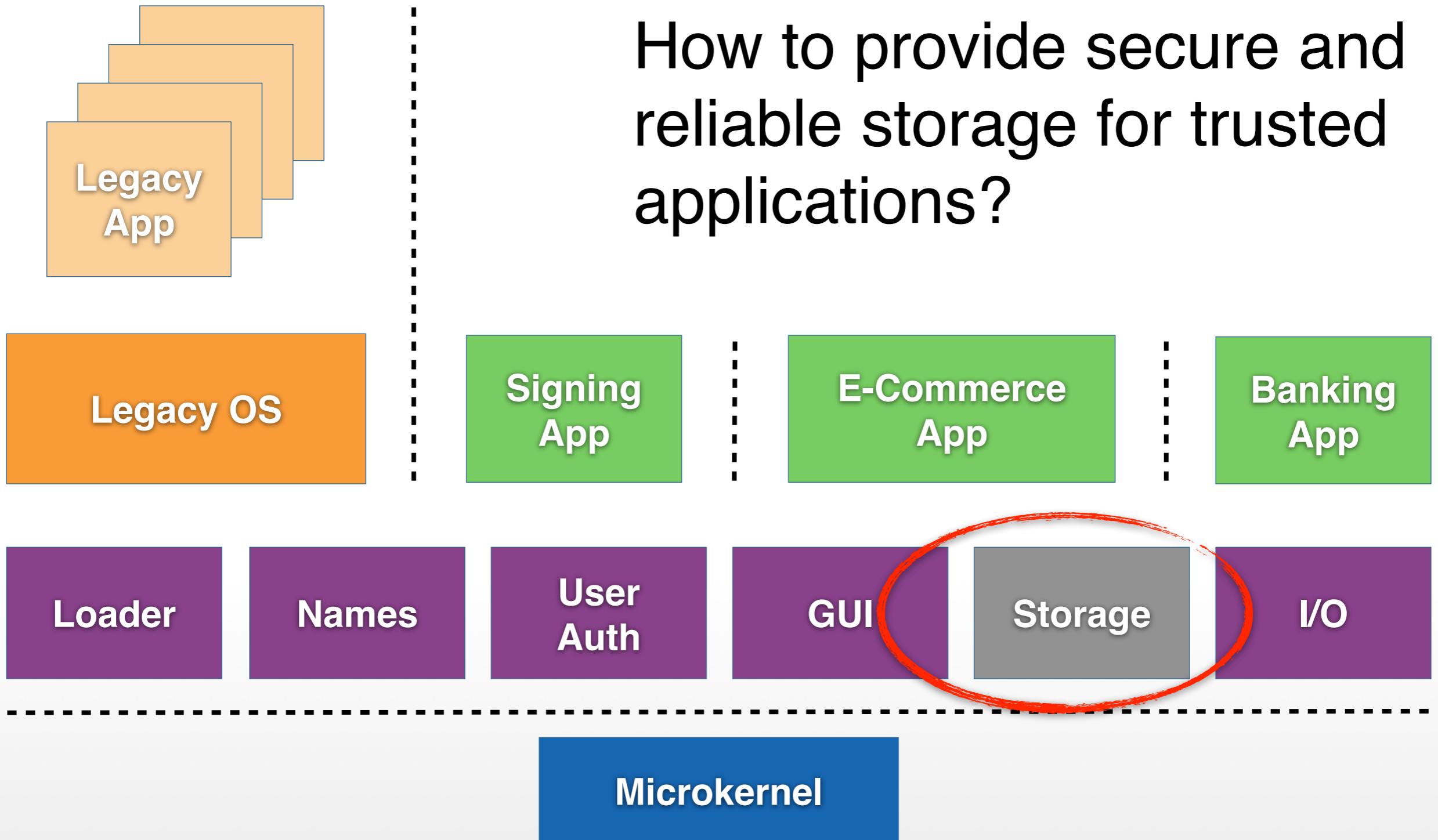


- Untrusted L<sup>4</sup>Linux instances must not see both plaintext and encrypted data
- Dedicated L<sup>4</sup>Linux for black/red networks

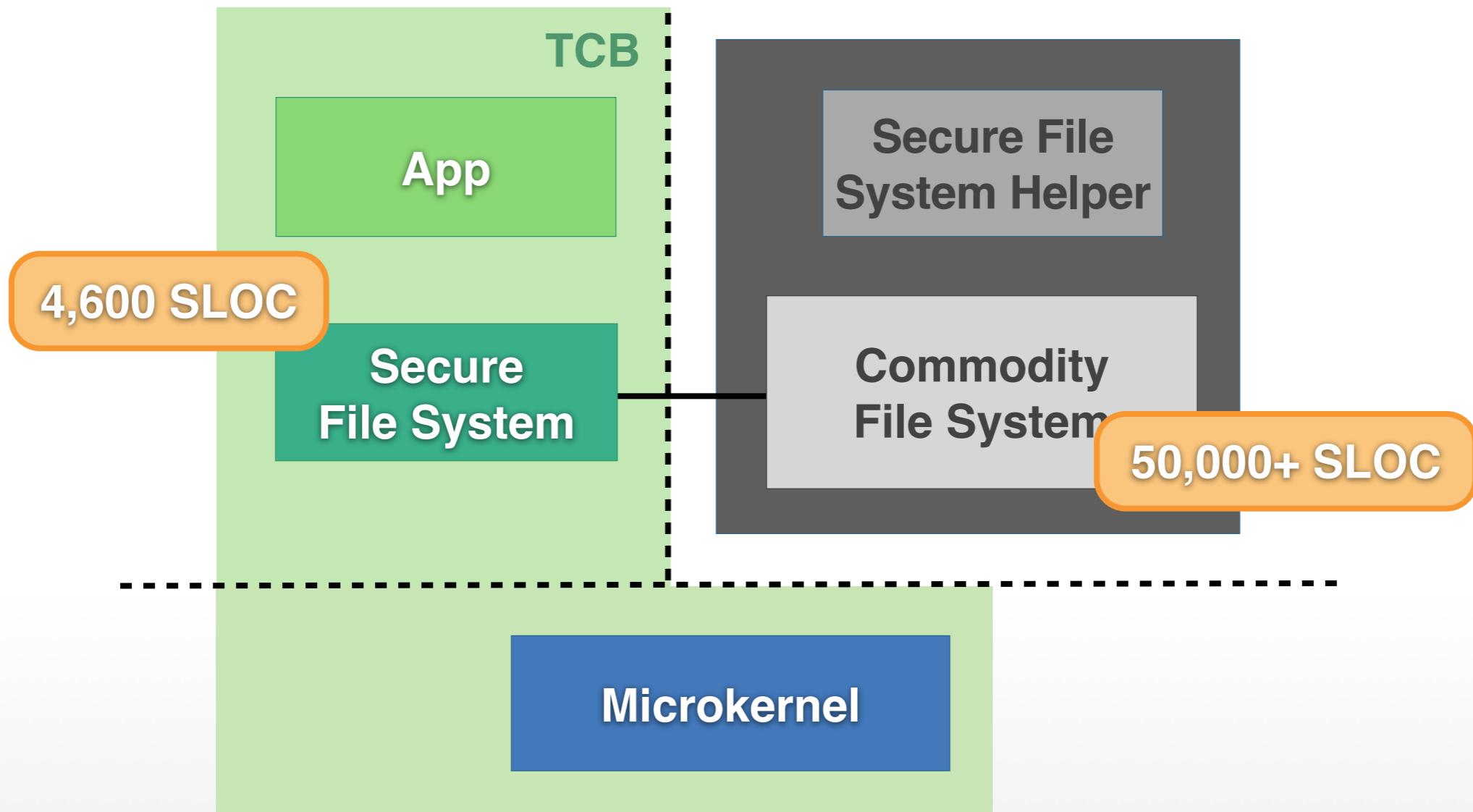


- Result: Trusted Wrapper for VPN
- Small TCB (see [6] for details):
  - 5,000 SLOC for „Viaduct“
  - Fine grain isolation
  - Principle of least privilege
- Extensive reuse of legacy code:
  - Drivers
  - IP stack

# EXAMPLE 3: STORAGE

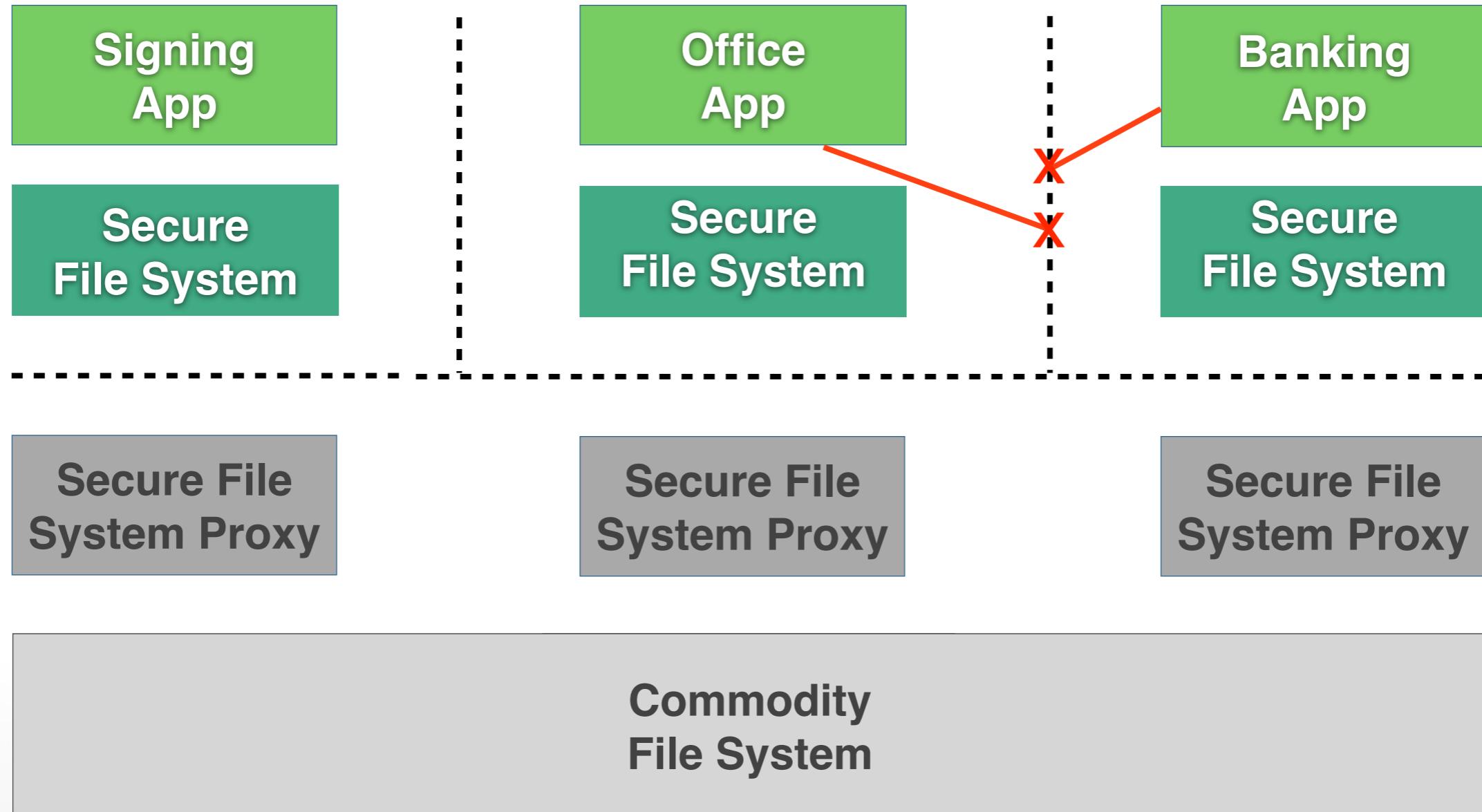


**VPFS: Confidentiality, Integrity, Availability**



See [3] for details

# VPFS STACK

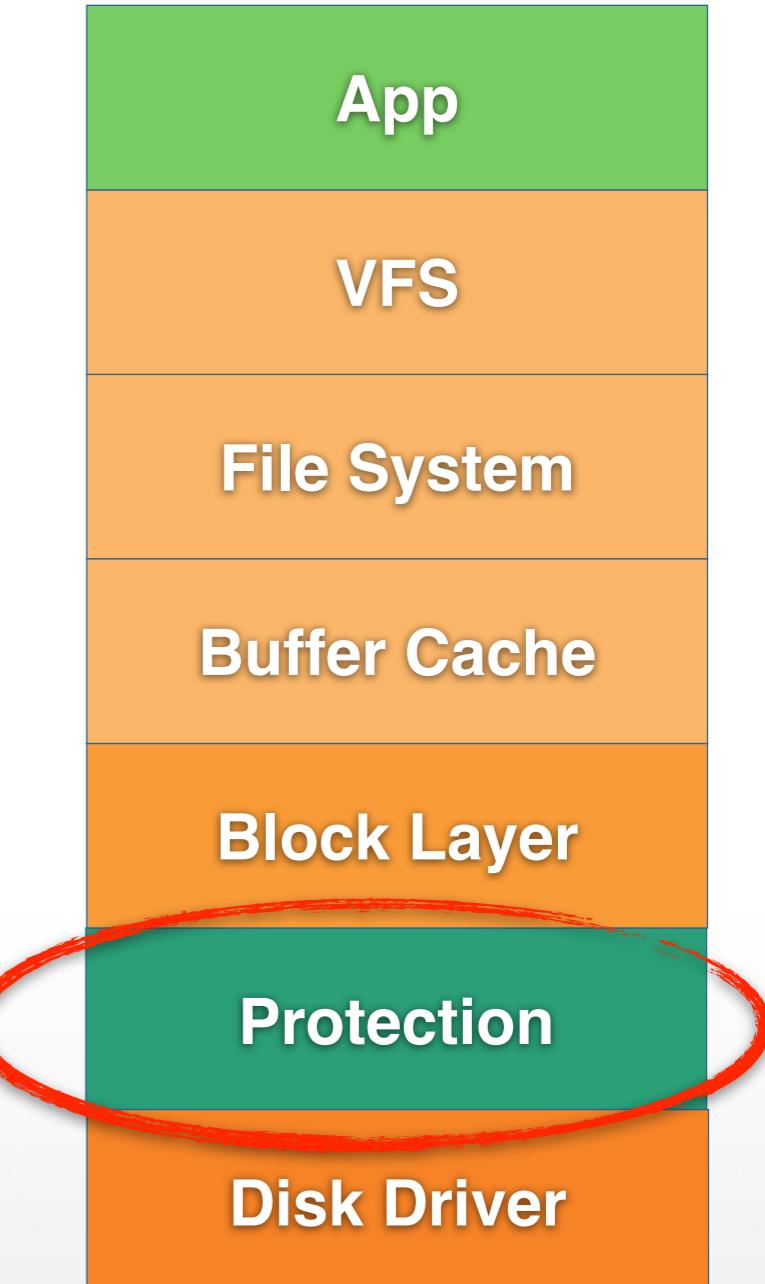


Isolate applications and their private storage: configure communication capabilities such that each application can access its private instance of the secure file system exclusively

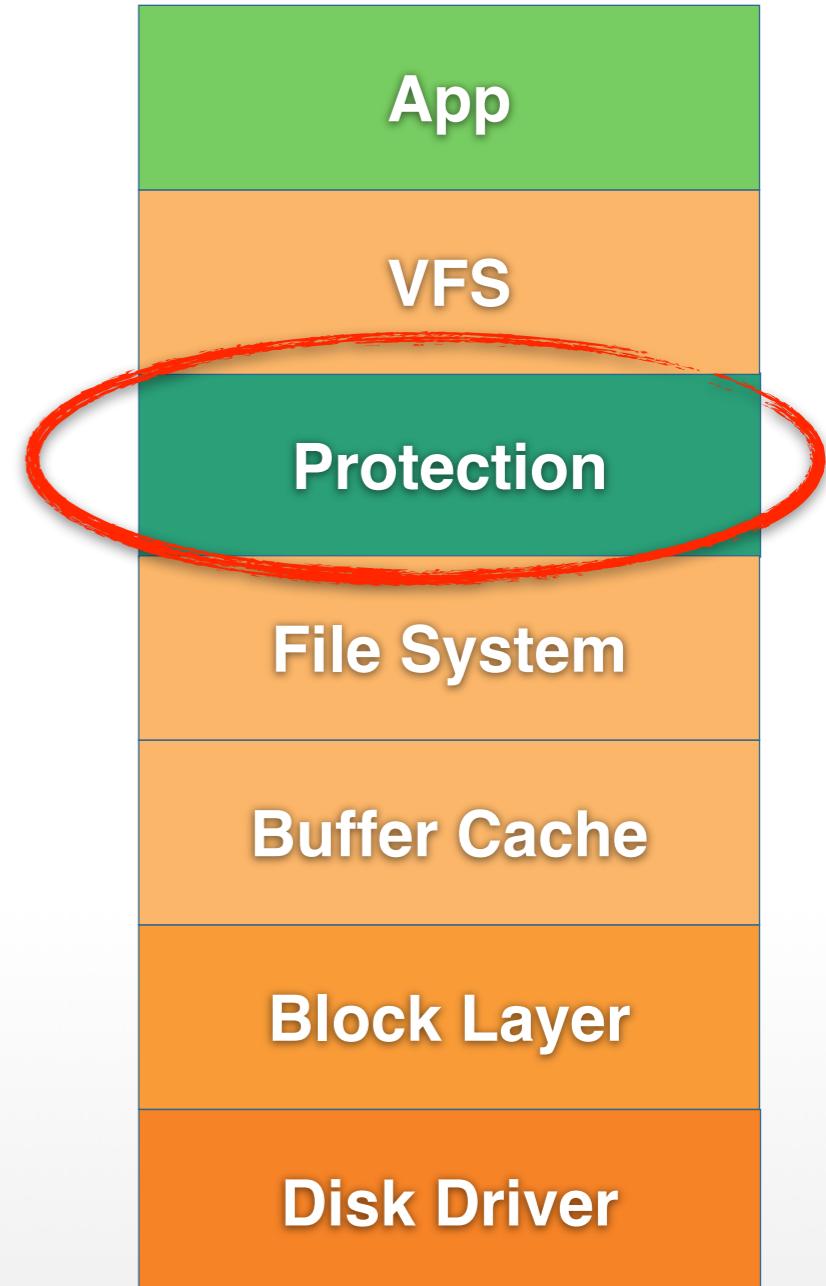
- **Confidentiality:** only authorized applications can access file system, all untrusted software cannot get any useful information
- **Integrity:** all data and meta data is correct, complete, and up to date; otherwise report integrity error
- **Recoverability:** damaged data in untrusted file system can be recovered from trusted backup

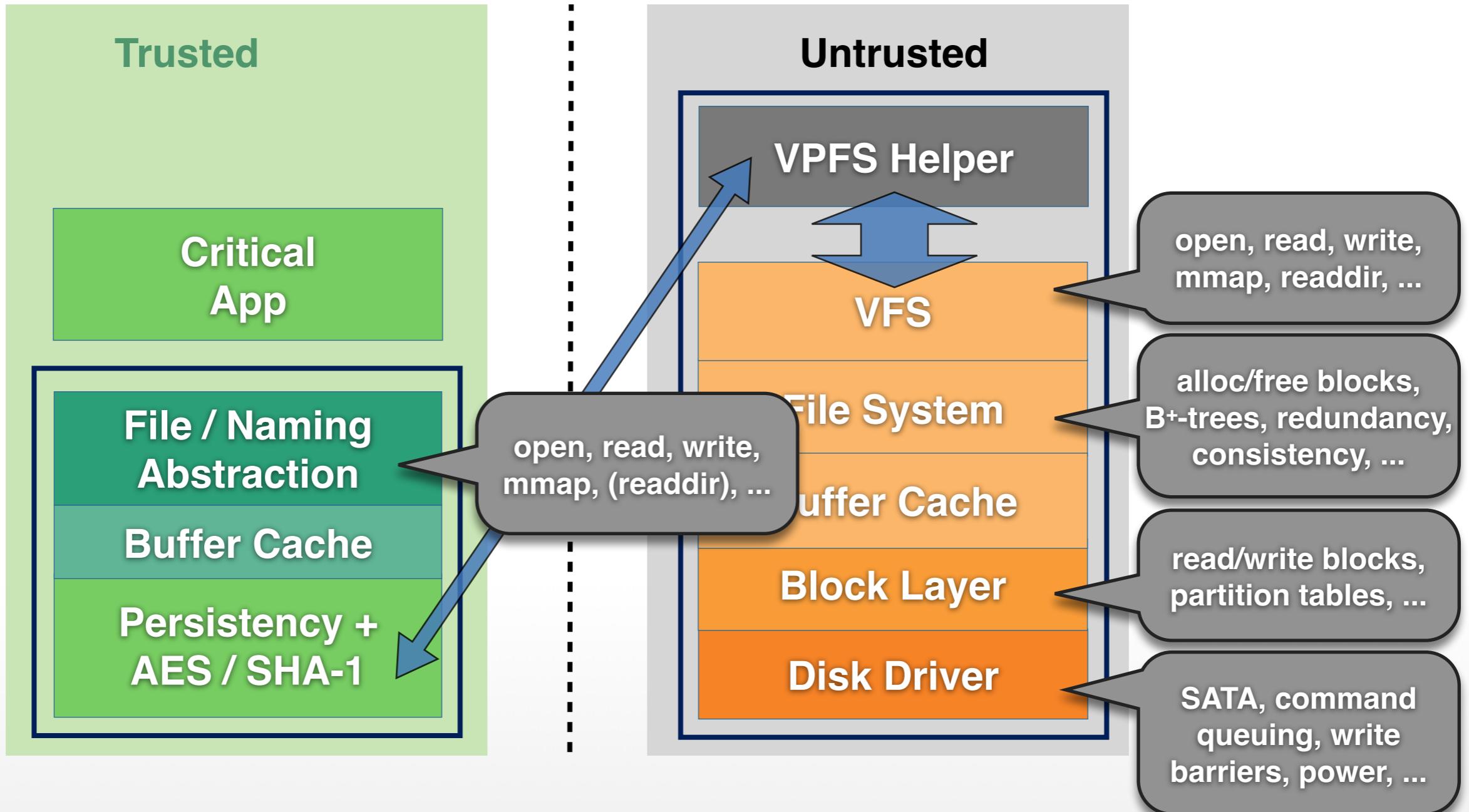


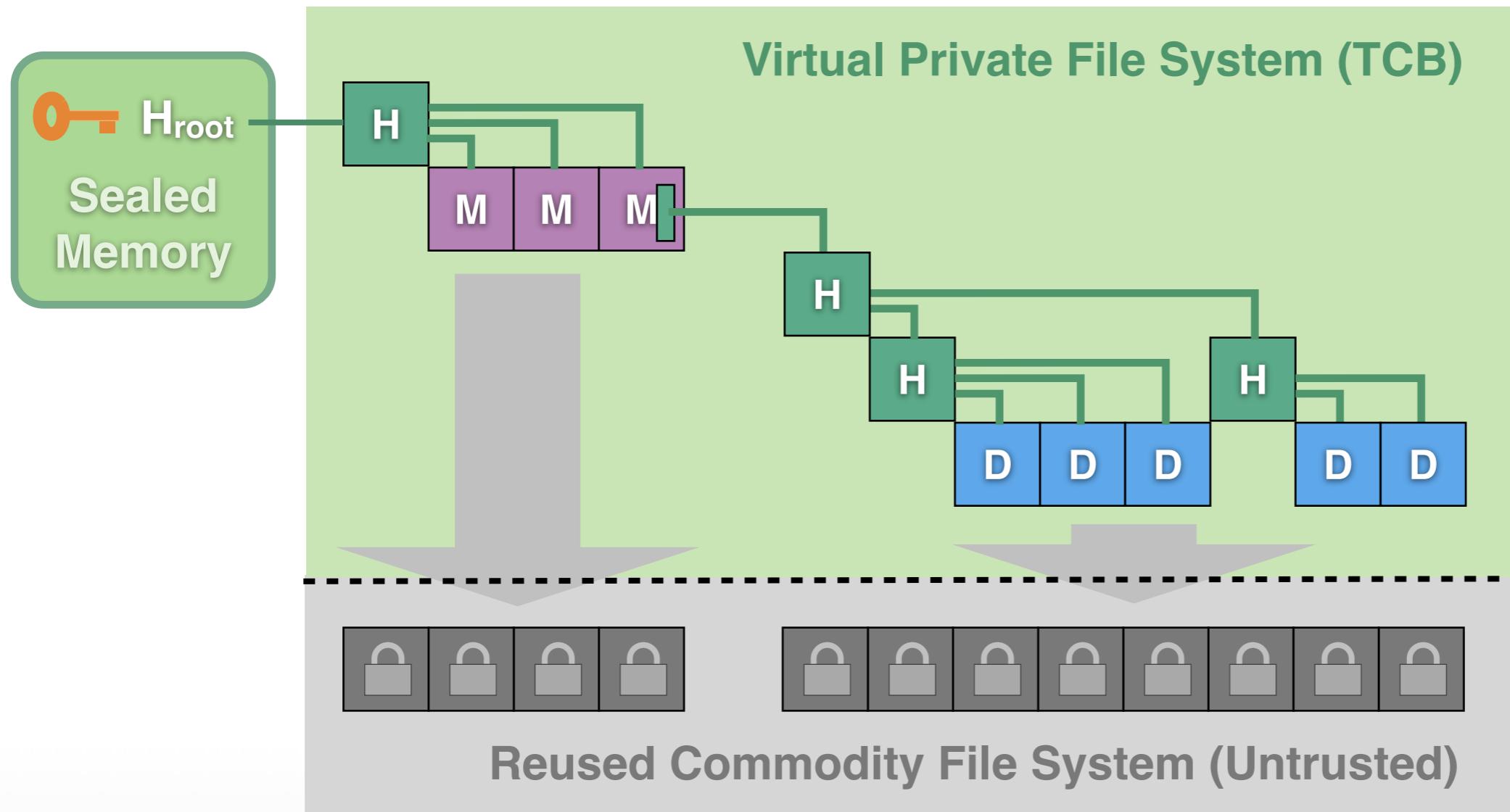
- First end of design space:  
Protect at block layer
  - Transparent encryption of all data and metadata
  - Block-level integrity ???
  - Most parts of file system stack are part of TCB
  - Attack surface still big



- 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 not part of TCB
  - Ideal for trusted wrapper

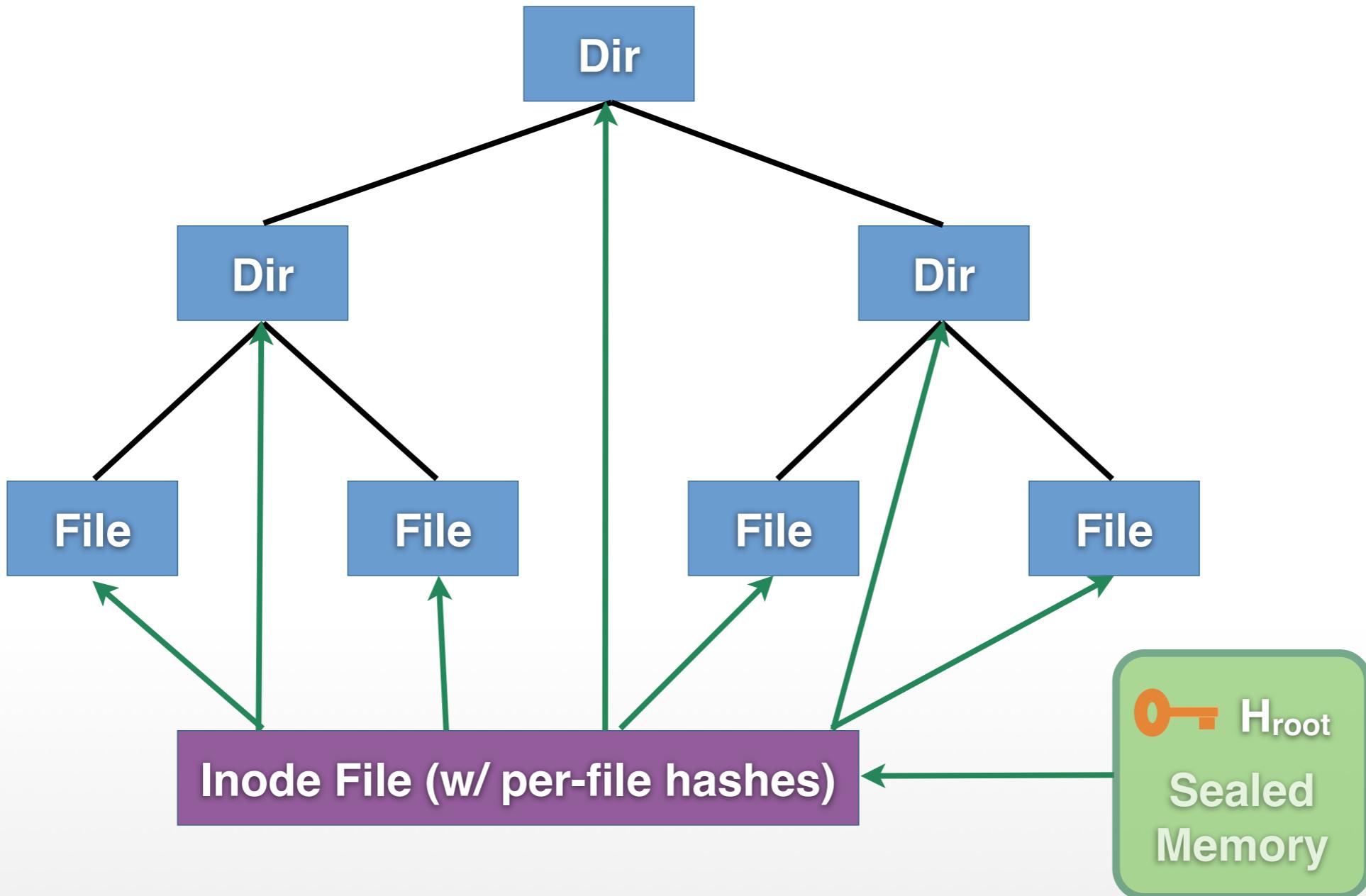




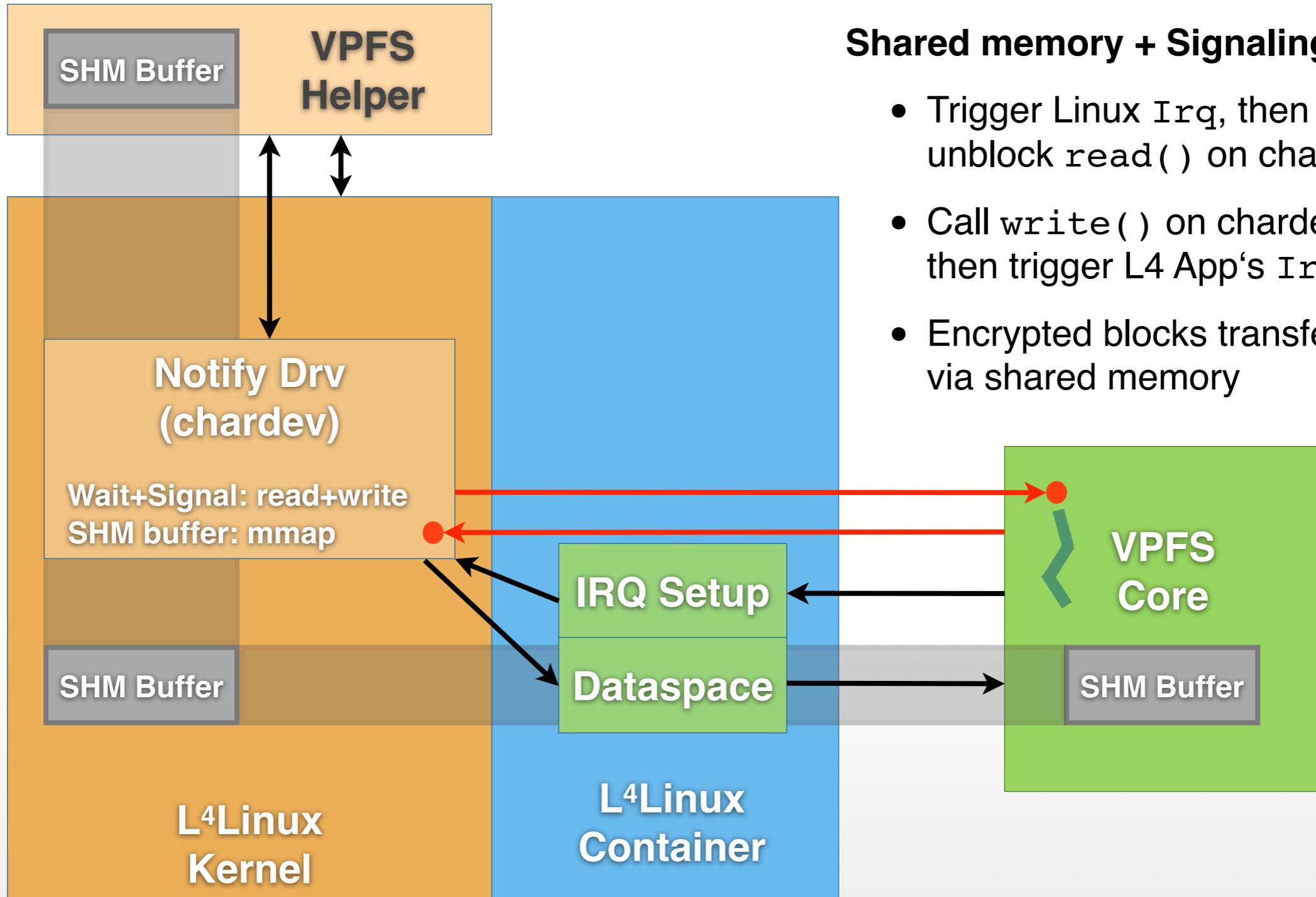


- Encrypted files in commodity file system
- Merkle **hash tree** to detect tampering

- 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



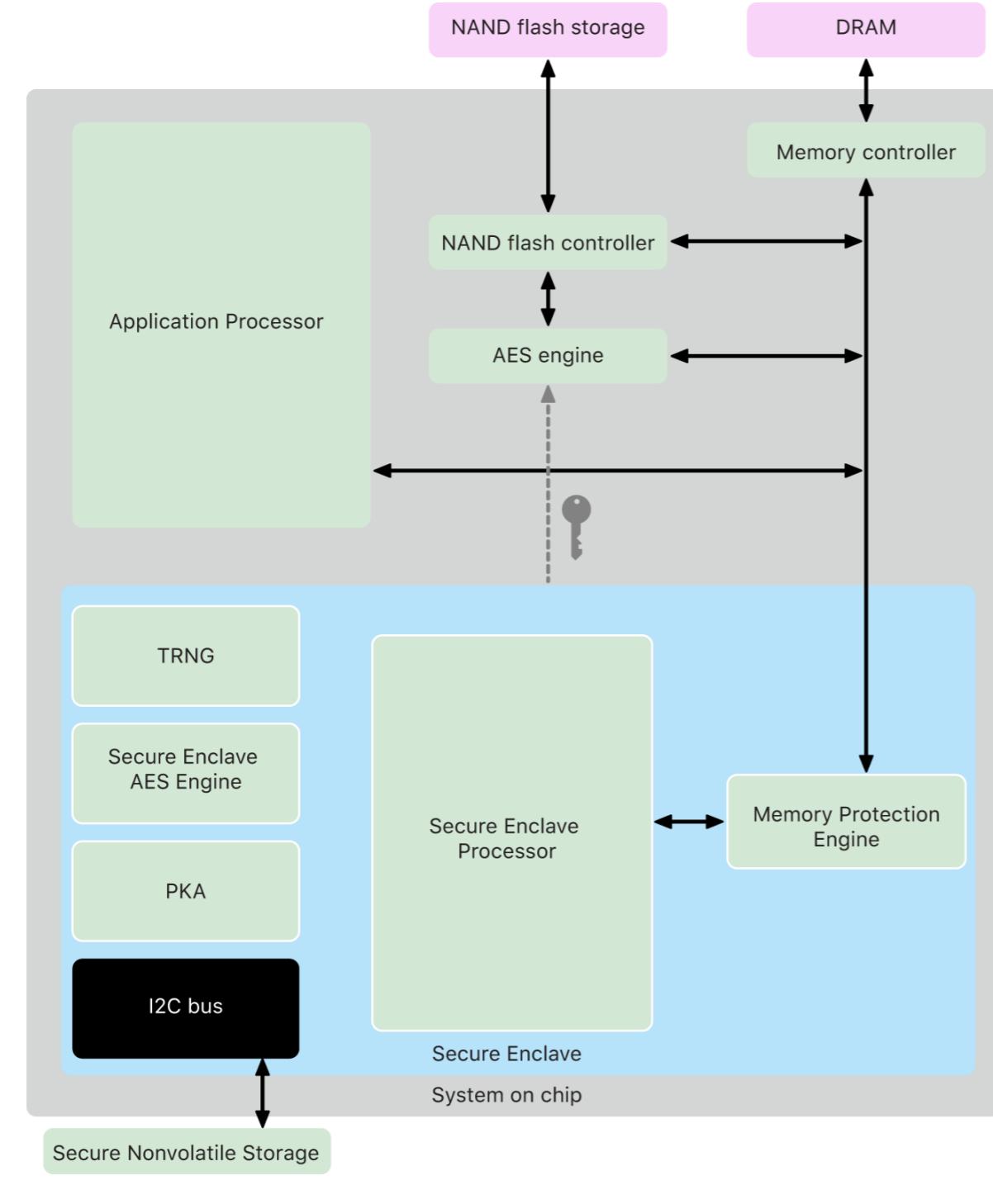
- VPFS reuses Linux file system stack:
  - Drivers, block device layer
  - Optimizations (buffer cache, read ahead, write batching, ...)
  - Allocate / free disk storage for files
- Cooperation: proxy driver in L<sup>4</sup>Linux



- 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]

- 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

# BRIEFLY: HARDWARE

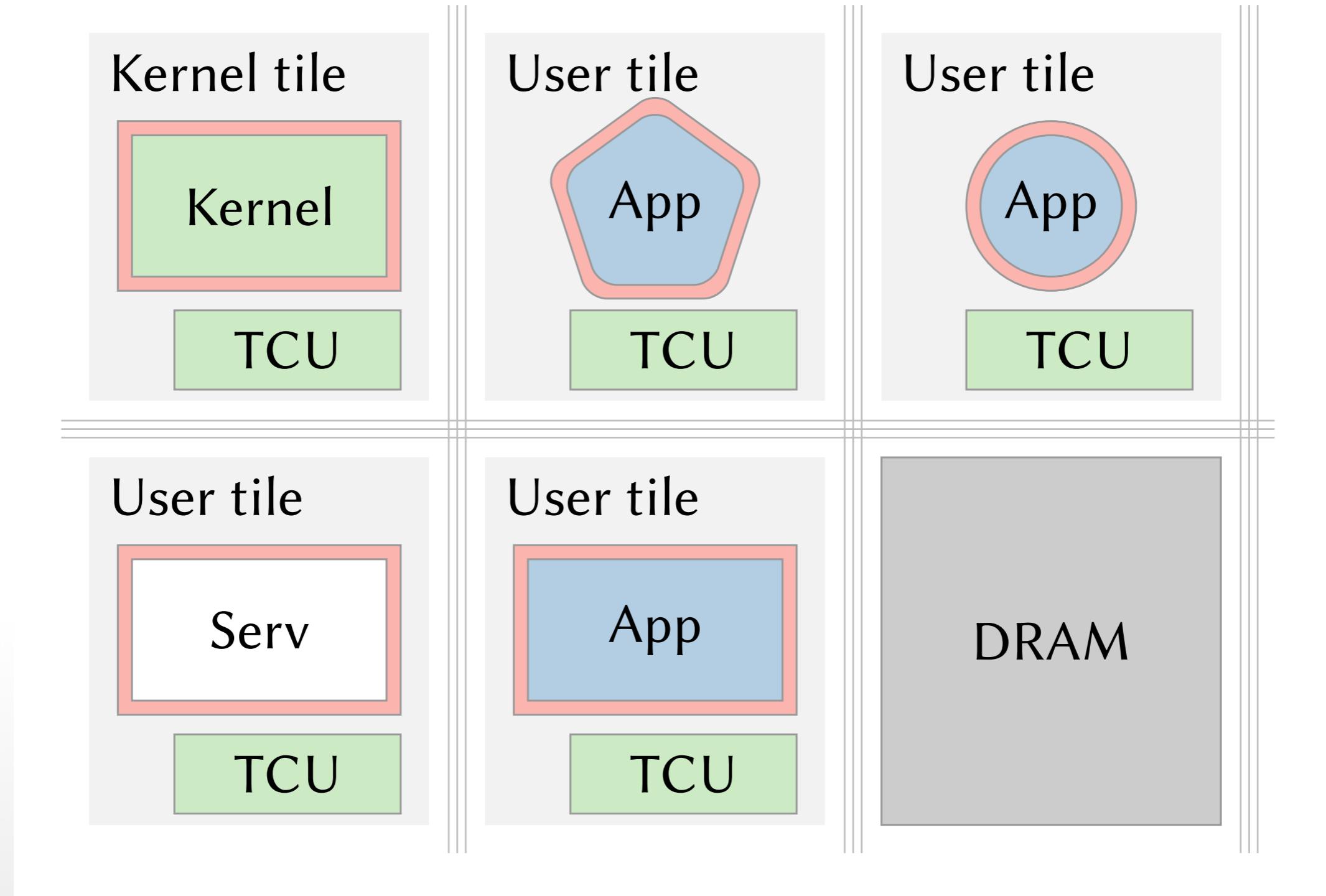


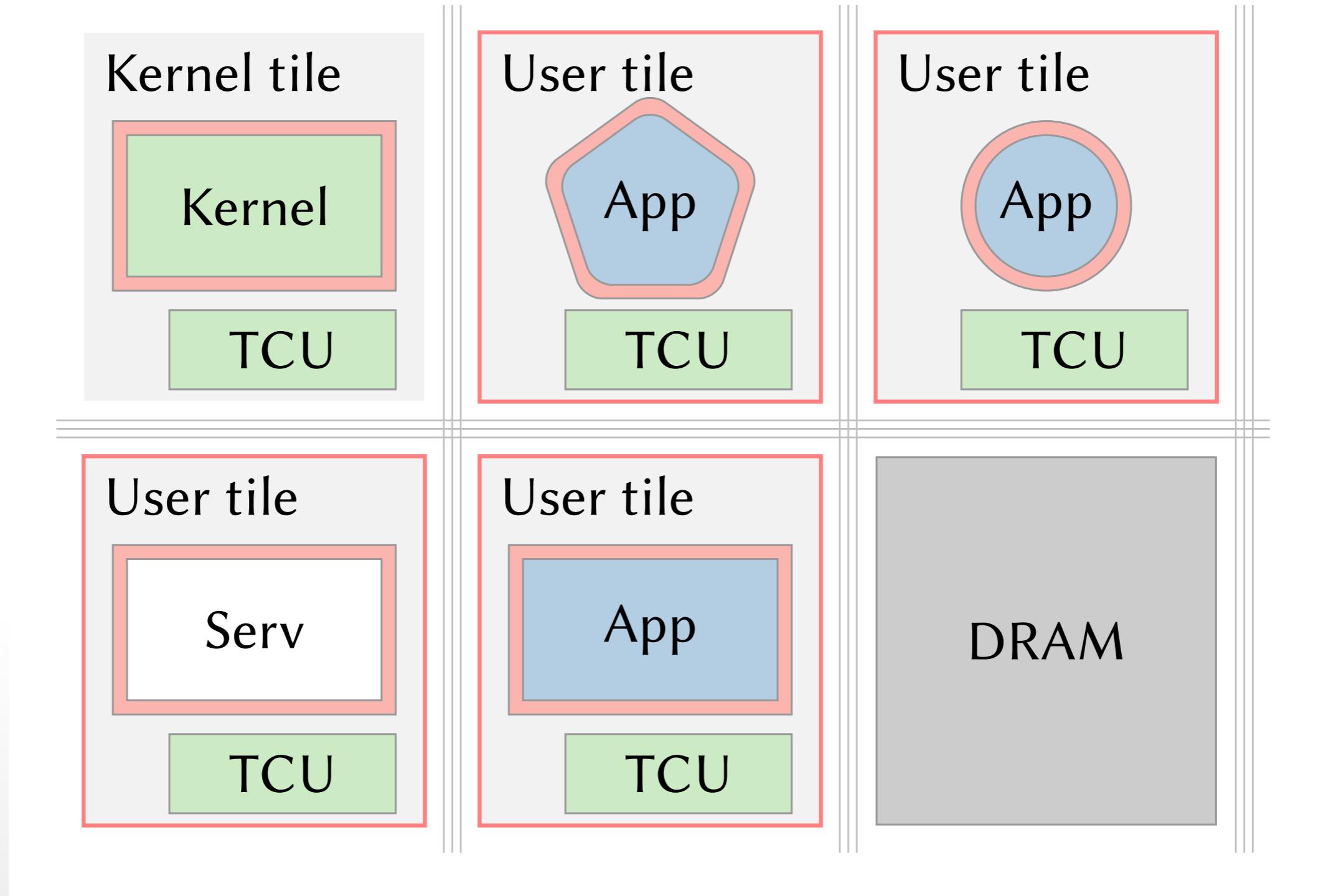
Apple devices have "Secure Enclave Processor (SEP)" running a dedicated service OS fully isolated from the application processor hardware.

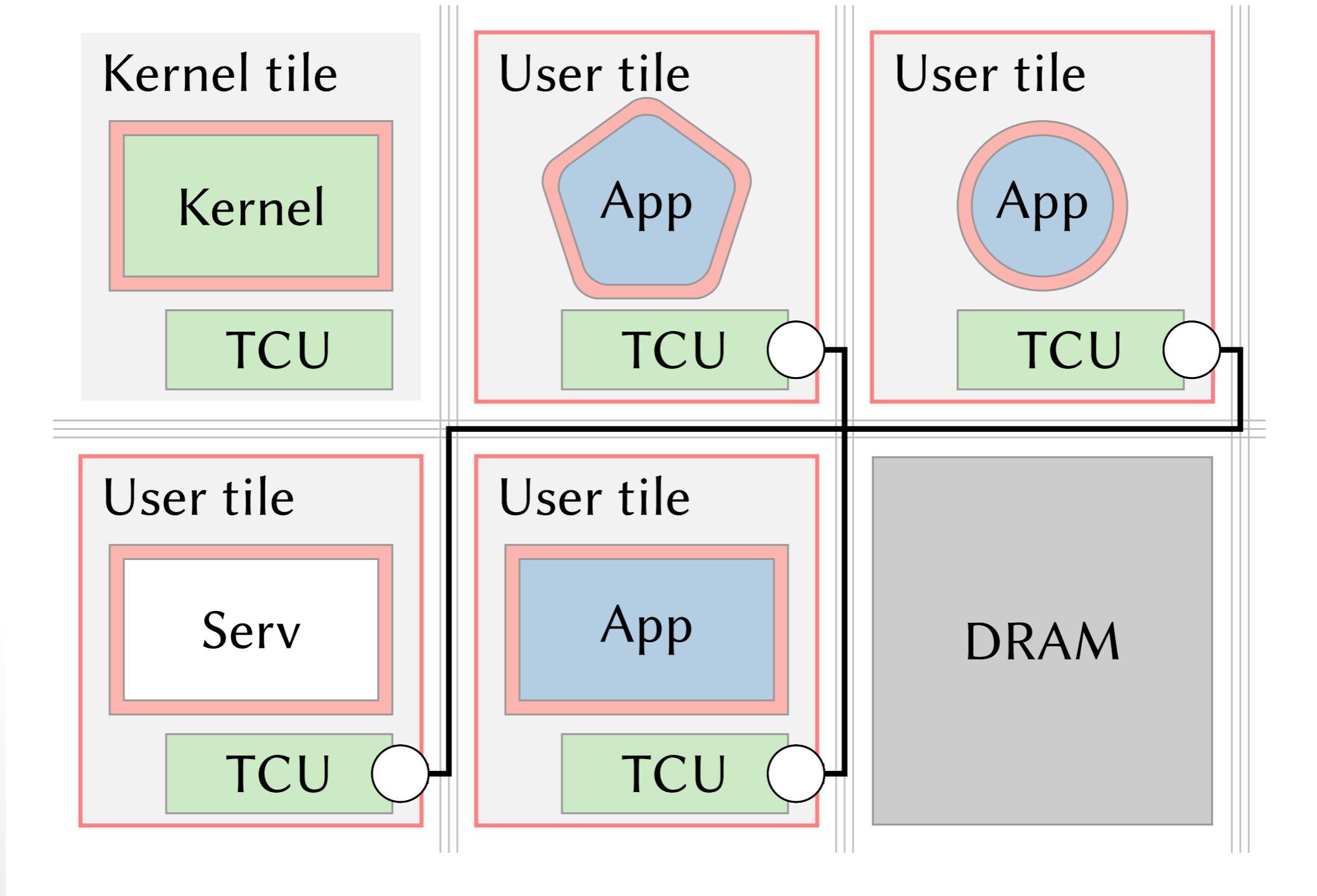
### The SEP runs **sepOS**:

"The Secure Enclave firmware is based on a version of the L4 microkernel customized by Apple."

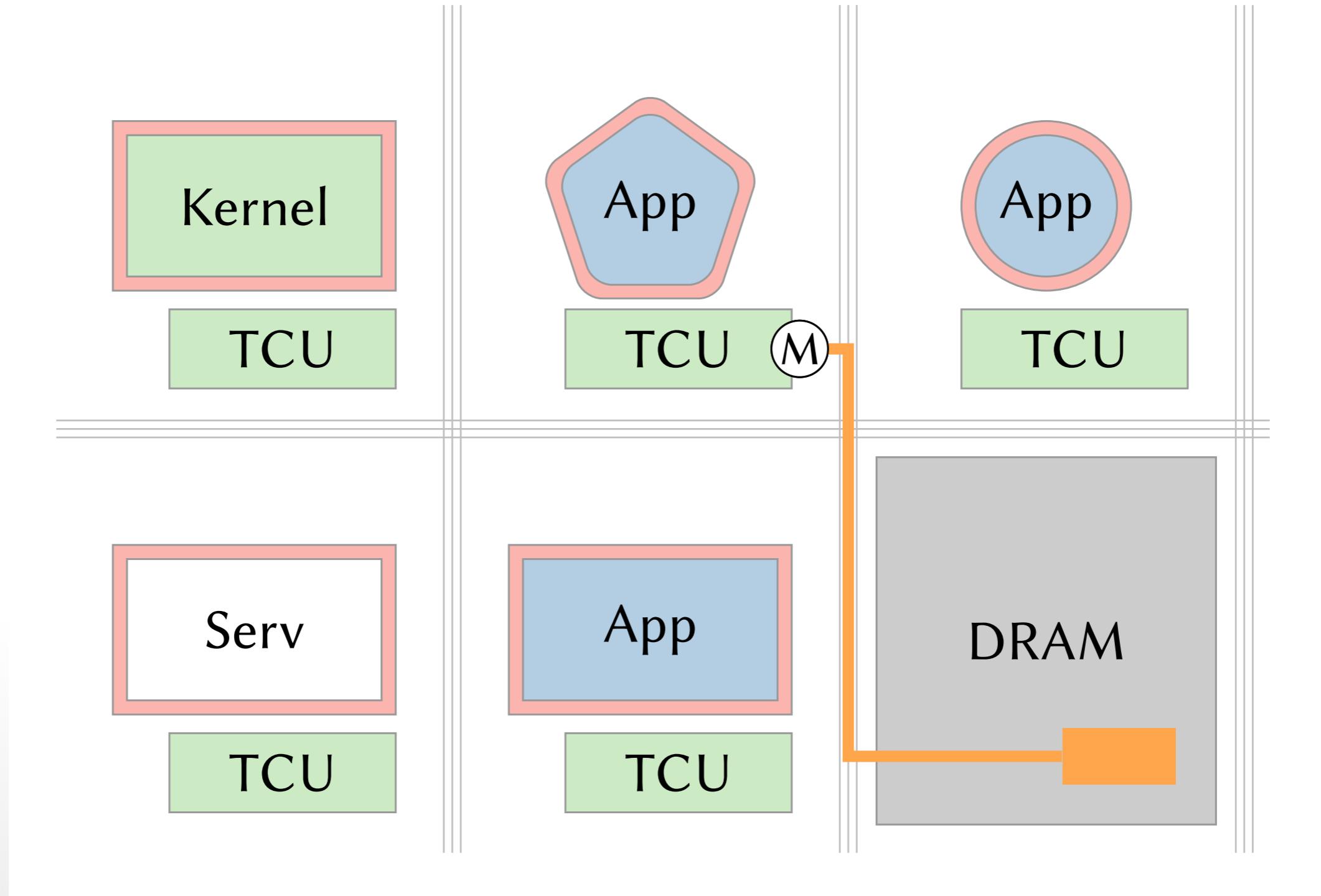
Source: Apple Support Documentation  
<https://support.apple.com/guide/security/secure-enclave-sec59b0b31ff/web>

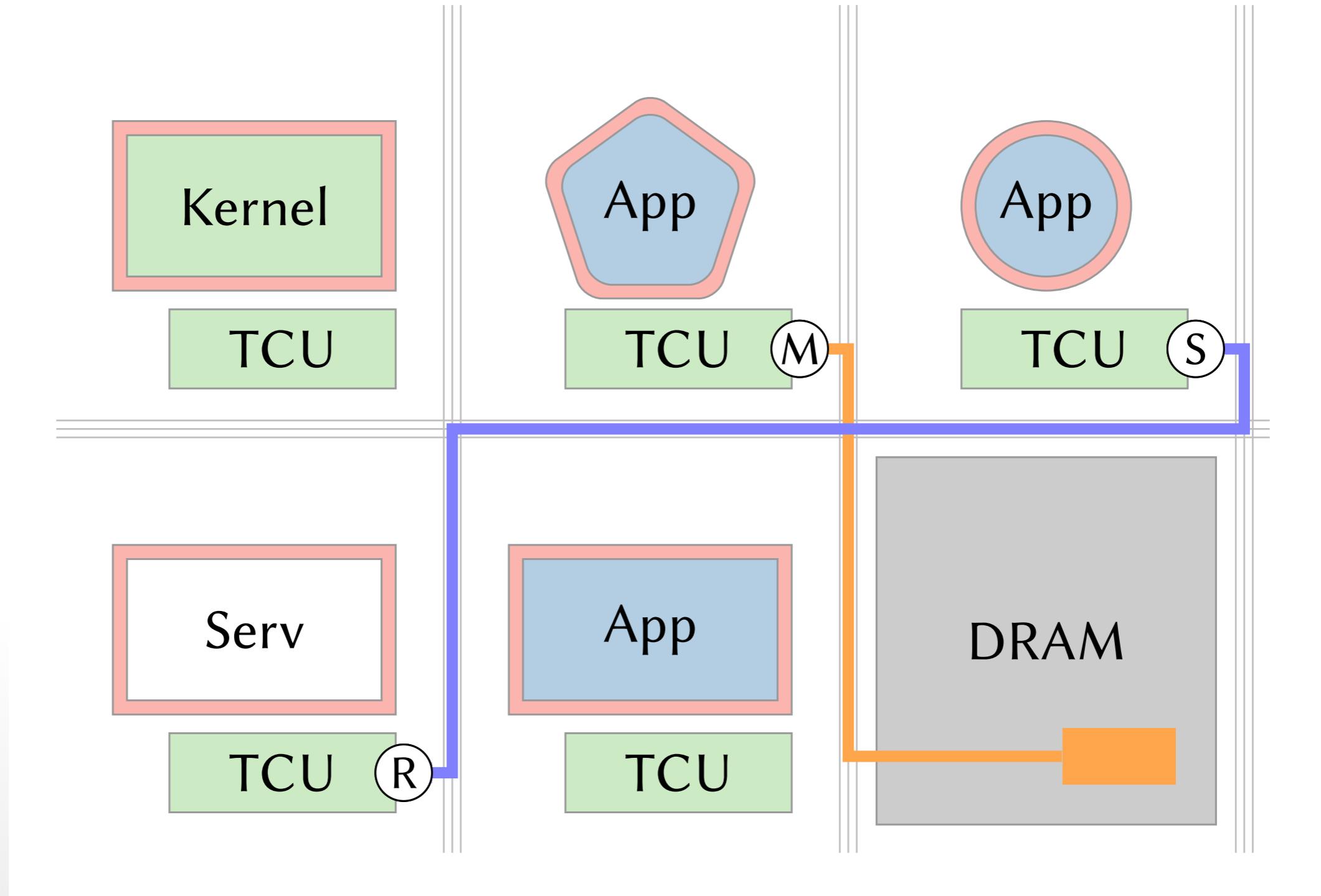




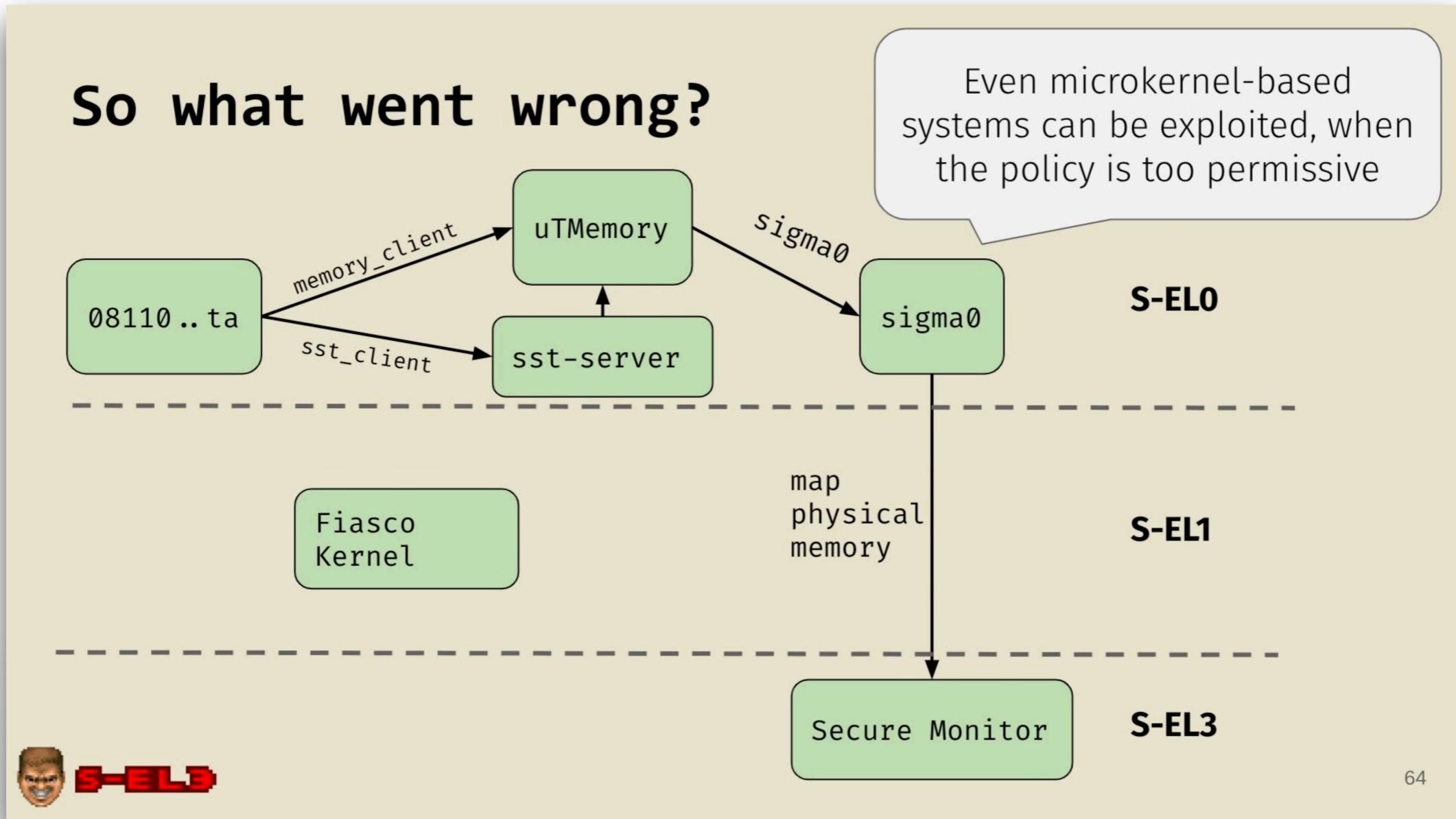


# SNEAK PEEK: M<sup>3</sup>





**CONFIGURE YOUR  
CAPABILITIES CORRECTLY**



[13] 39C3 talk by 0ddc0de, gannimo, Philipp: "Not To Be Trusted - A Fiasco in Android TEEs",  
<https://app.media.ccc.de/v/39c3-not-to-be-trusted-a-fiasco-in-android-tees>

- [1] <http://www.heise.de/newsticker/Month-of-Kernel-Bugs-Ein-Zwischenstand--/meldung/81454>
- [2] <http://projects.info-pull.com/mokb/> [defunct]
- [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> [defunct]
- [10] Asmussen et al.: „M3: A Hardware/OS Co-Design to Tame Heterogeneous Manycores“, ASPLOS'16
- [11] Artenstein: „BroadPwn: Remotely Compromising Android and iOS via a Bug in the Broadcom Wi-Fi Chipset“, <https://www.blackhat.com/docs/us-17/thursday/us-17-Artenstein-Broadpwn-Remotely-Compromising-Android-And-iOS-Via-A-Bug-In-Broadcoms-Wifi-Chipsets-wp.pdf>
- [12] Ilya van Sprudel, Michael Smith: “Escaping Containment: A Security Analysis of FreeBSD Jails”, <https://app.media.ccc.de/v/39c3-escaping-containment-a-security-analysis-of-freebsd-jails>
- [13] 0ddc0de, gannimo, Philipp: “Not To Be Trusted - A Fiasco in Android TEEs”, <https://app.media.ccc.de/v/39c3-not-to-be-trusted-a-fiasco-in-android-tees>