TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Operating Systems Group

INFLUENTIAL OPERATING SYSTEM RESEARCH: SECURITY MECHANISMS AND HOW TO USE THEM

CARSTEN WEINHOLD

OVERVIEW

- Fundamental Concepts and Building Blocks
- Problems in Practice
- Security Architectures
- The Way Forward?

FUNDAMENTAL CONCEPTS AND BUILDING BLOCKS

CAPABILITIES

A capability is an unforgeable (immutable) token (piece of data) of authentication for some system resource possessed by a process. Possession itself grants access.

- First described by Dennis & Horn in 1966
- Managed and protected by system
- Attached to process, but cannot be forged by it
- Can be shared, transferred, inherited

ACCESS CONTROL LISTS

An ACL is an out-of-process entity that allows to control access to some system resource. Access is granted after proactive checking by the enforcing system.

- Maps identifiers to access rights
- Attached to objects to be access by processes
- Managed by system, cannot be forged by process
- Usually whitelist, but can also include blacklist semantics

ROLE-BASED ACCESS CONTROL

- Programs always work in a role
- Program can drop to a lower role but not elevate to a higher role
- Higher role programs start lower role programs
- Roles can be selectively inheritable
- Example: SELinux
  - Every program effectively needs a policy
  - Huge maintenance burden and/or trust in vendor/distributor

RULE-SET BASED ACCESS CONTROL

- Combines all previous approaches (best of all worlds)
- Rules can be combined to sets
- Sets can be (selectively) inherited

PROCESSOR MODES

- Invented for MULTICS to enable multiprogramming environment
- Idea of rings of privilege that hold CPU instructions
- Inner rings can use instructions in the outer rings, but not vice versa
- Allows to implement memory protection via hardware-enforced addressing schemes

PROCESSOR MODES: X86-64

- Ring 3: User mode
- Ring 2: Unused
- Ring 1: Unused
- Ring 0: Kernel mode
- Ring „-1“: Hypervisor mode
- Also:
  - System management mode (SMM)
  - Secure enclave („SGX mode“)

MEMORY PROTECTION

- Segmentation
- Paging
- Capability-Based Addressing

SEGMENTATION

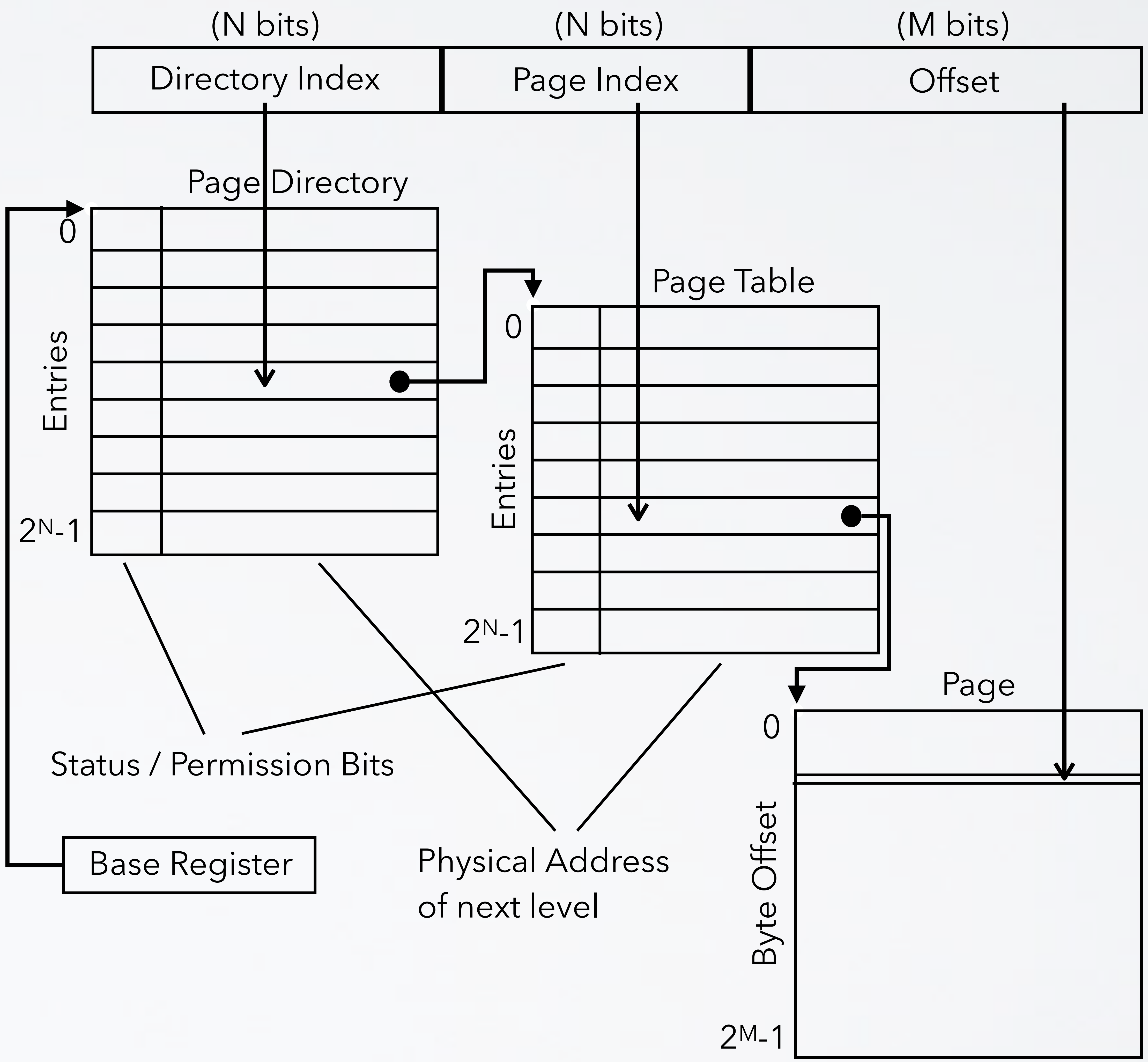
- Flat (virtual) address space partitioning
- First implemented in the Burroughs B5500, but also in MULTICS, IBM System/38, Intel 80286
- Addresses relative to segment base register:
  `address = segment + offset`
- Segment limit register marks size of segment
- Memory segmentation visible to the process
- RAM and file-system address spaces can be merged

PAGING

- Hierarchical, per-process mapping of virtual memory to physical memory at page granularity
- First implemented in the Atlas Computer (1959/62), but also in IBM System/370, Intel IA-32 (since 80386), …
- 2 (or 3+) protection domains: (VM) / Kernel / User-space
- Page sizes of limited variability (e.g., 4 kiB normal page and 4 MiB super page)

PAGING

- OS manages page tables
- Physical data layout and current consumption invisible to process
- Status and Permission bits in table entries specify access rights:
  - User vs kernel mode
  - Read/write/execute



CAPABILITY-BASED ADDRESSING

- Form of object-based addressing: every access to memory referenced through a capability
- No unrestricted pointer operations allowed in user space
- Single-address space possible ⇒ no context switches
- Possible implementations:
  - Store capabilities in protected memory area, modify through privileged process
  - Extend memory with „capability bits“ to mark protected locations (recent example: CHERI capabilities)

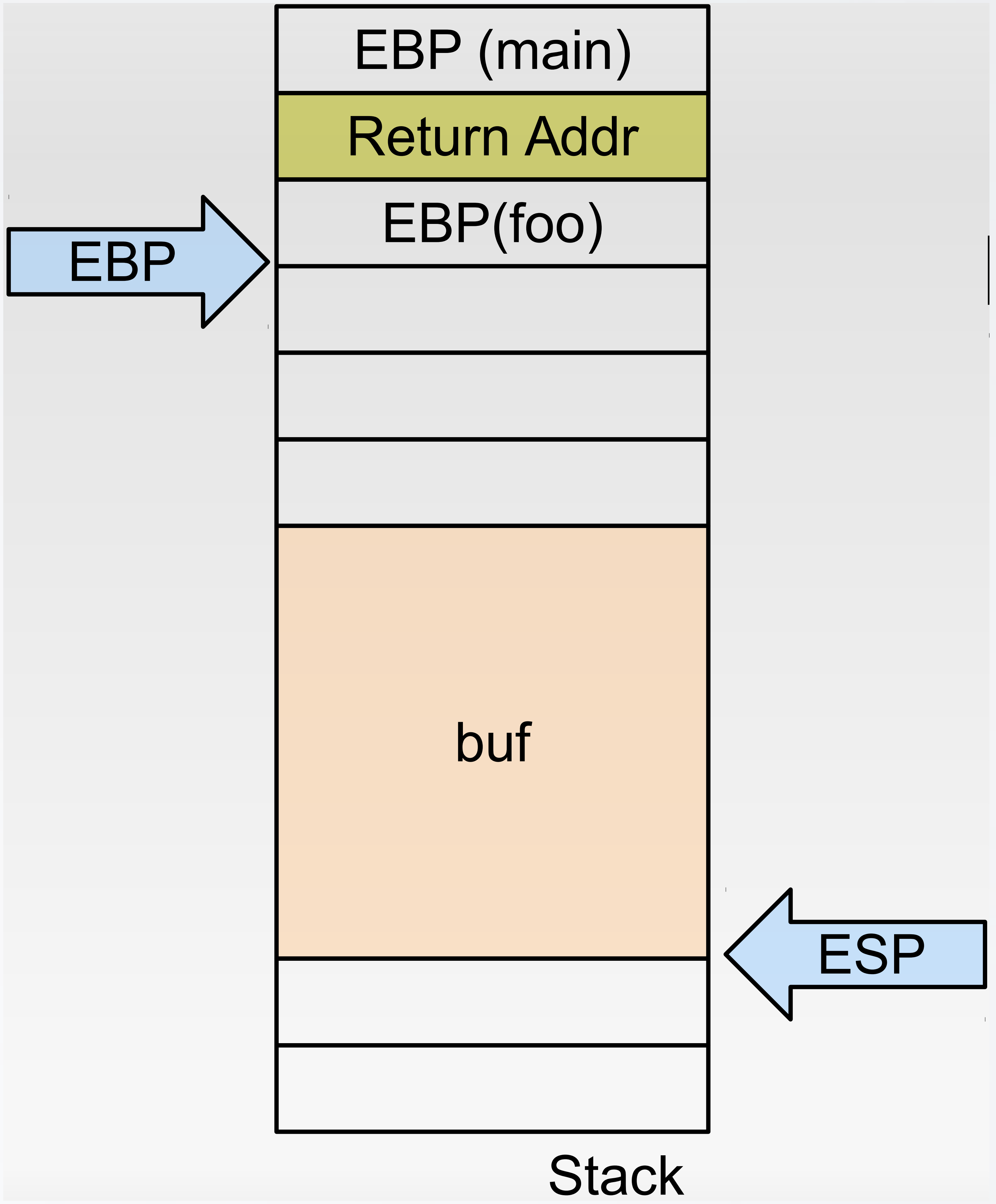
PROBLEMS IN PRACTICE

YOU ARE LEAVING THE SAFE

- No (or only limited) protection within a process
- Programs can read / write within their own address space
- Use of pointers unsafe in: native code / C / C++, …

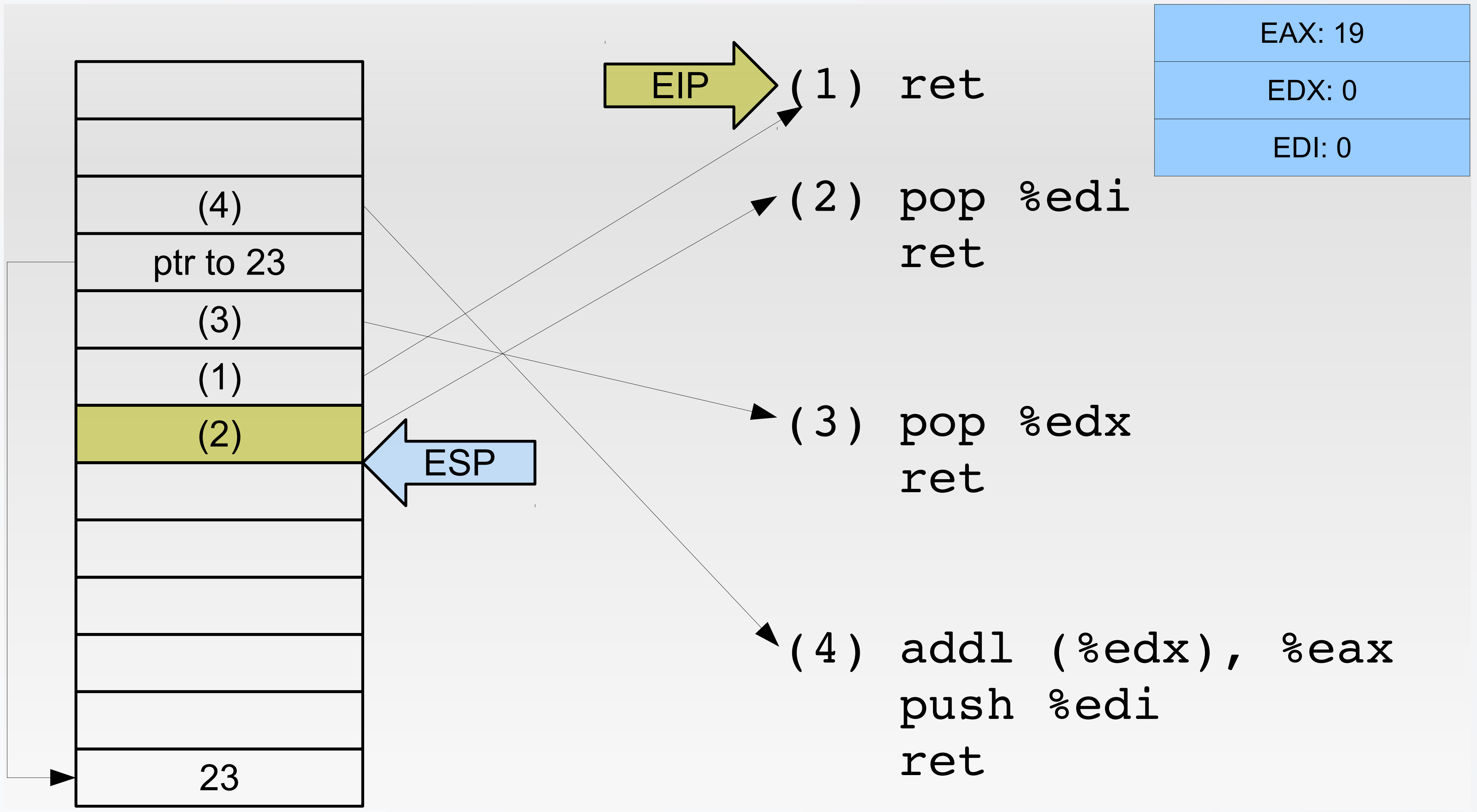
BUFFER OVERFLOW ATTACKS

- Stack overflows smash may return addresses, jump anywhere on `ret`
- Overflows on heap may overwrite:
  - Function pointers
  - VTable pointers
  - Memory management information
- Partial mitigations:
  - Canaries (but may be guessable)
  - Shadow stacks for return addresses



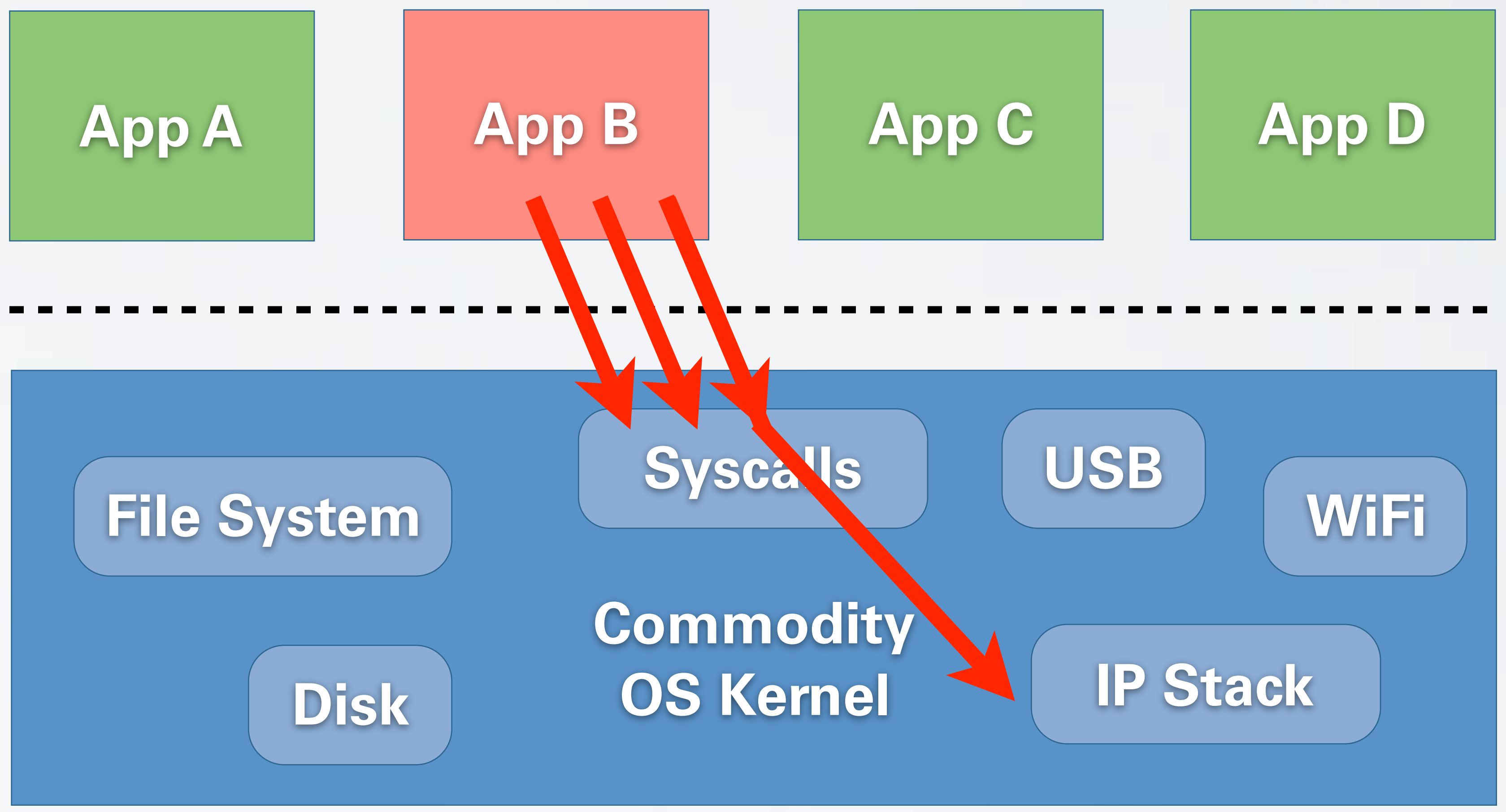
ROP ATTACKS

- Write XOR execute semantics makes code injection attacks useless
- But return instruction still allows unrestricted jumps to arbitrary addresses: Return-oriented programming (ROP)



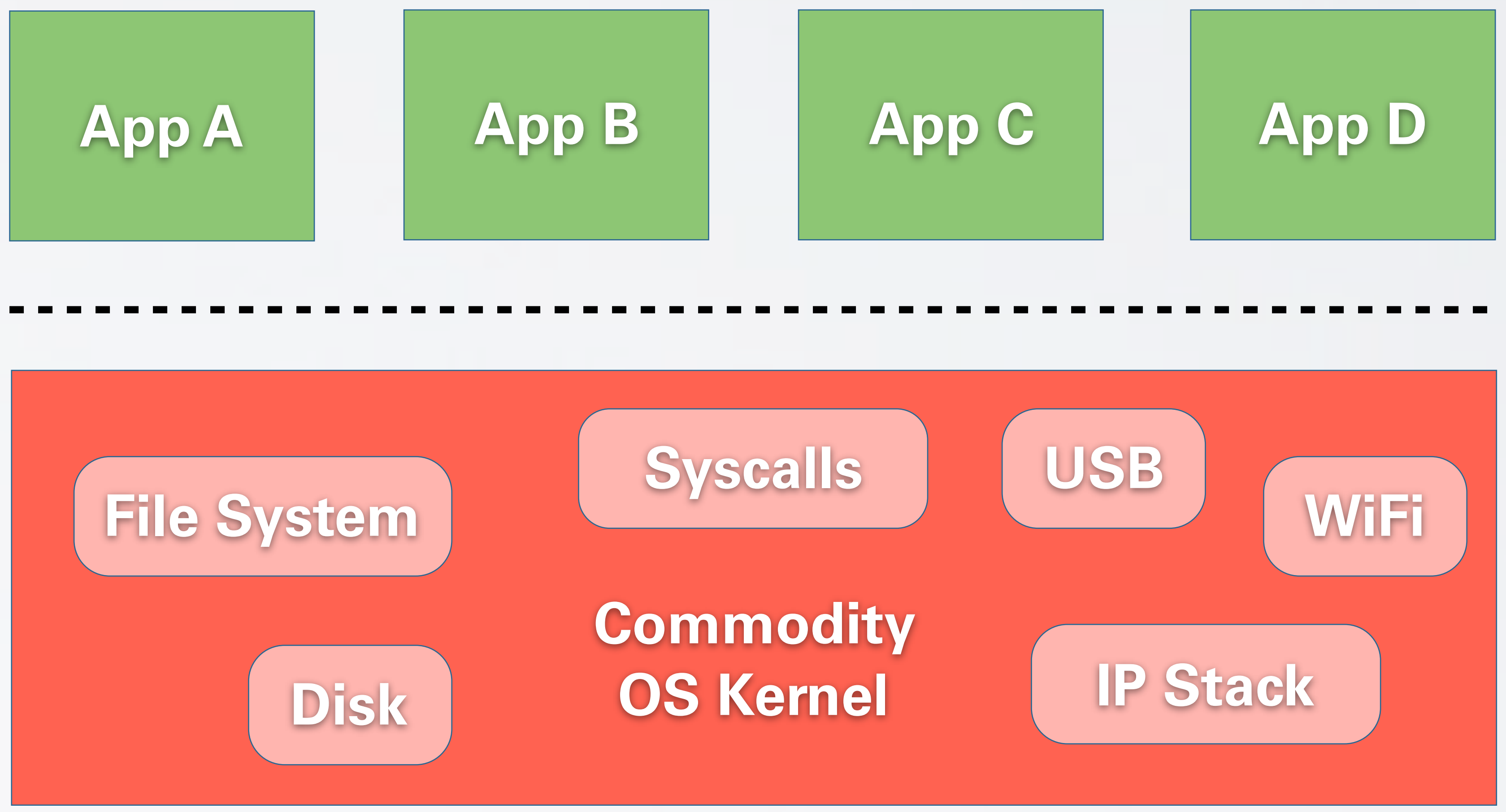
MONOLITHIC KERNEL

- All security mechanisms are implemented / managed by operating system kernel
  - Capabilities, ACLs, …
  - Memory protection, …
- All other OS functionality, too
- Huge codebase, large attack surface exposed to applications



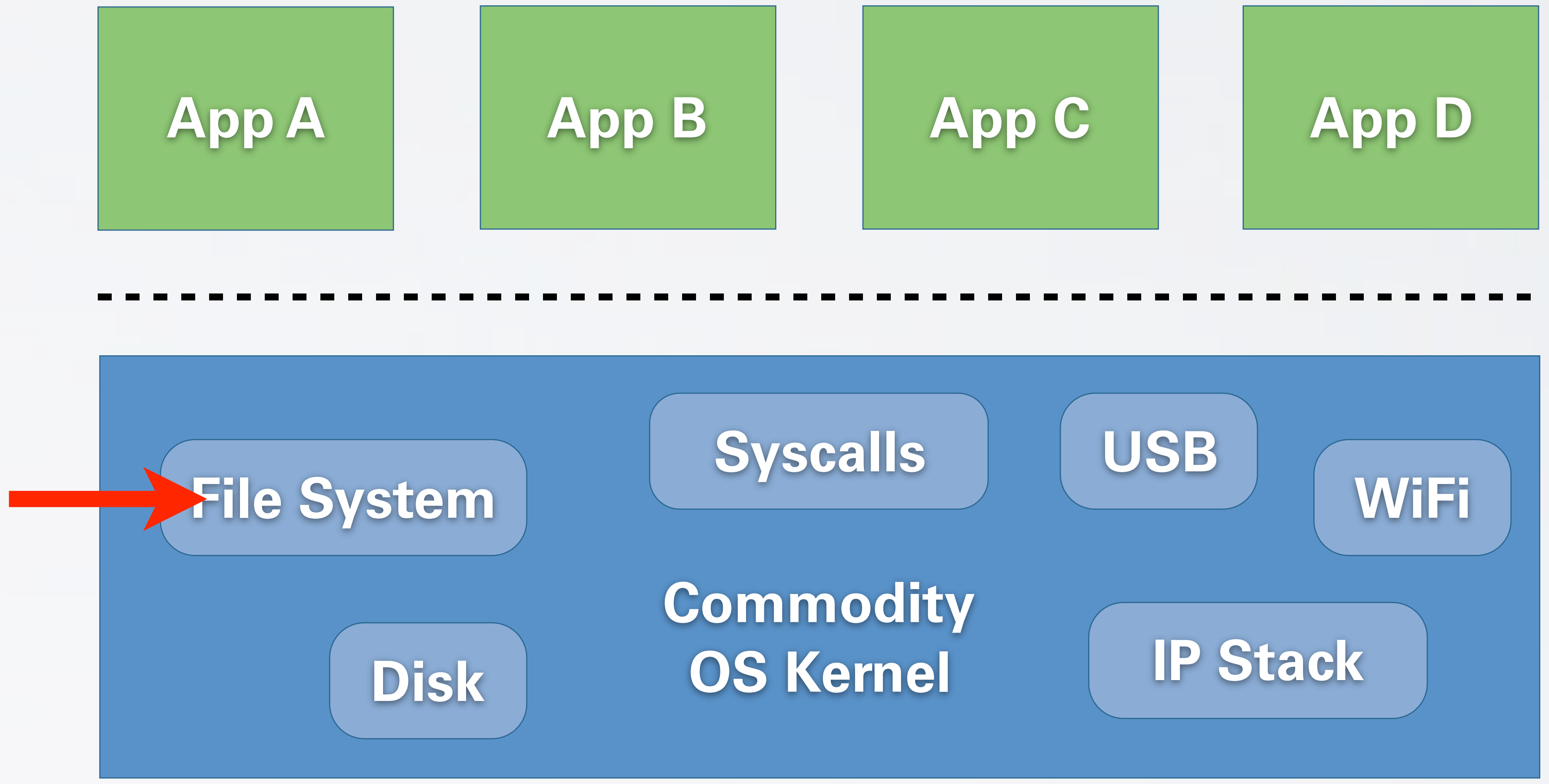
MONOLITHIC KERNEL

- All security mechanisms are implemented / managed by operating system kernel
  - Capabilities, ACLs, …
  - Memory protection, …
- All other OS functionality, too
- Huge codebase, large attack surface exposed to applications
- No protection within kernel



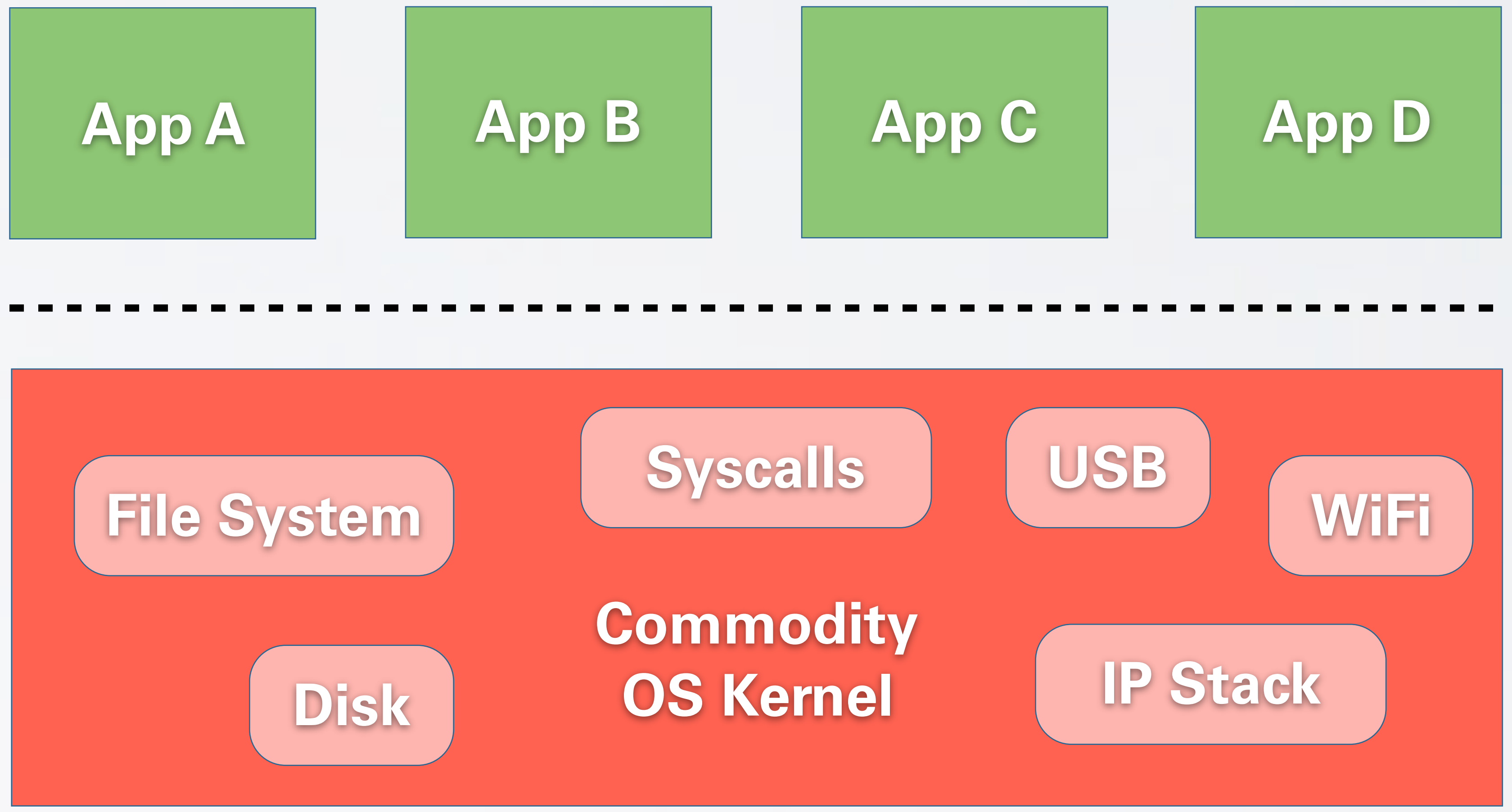
MONOLITHIC KERNEL

- Not only malicious applications
- Operating system kernel also exposed to untrusted input
  - Network packets and protocols
  - Thunderbolt, USB, other buses
  - File-system images
- One exploitable bug: kernel and all applications compromised



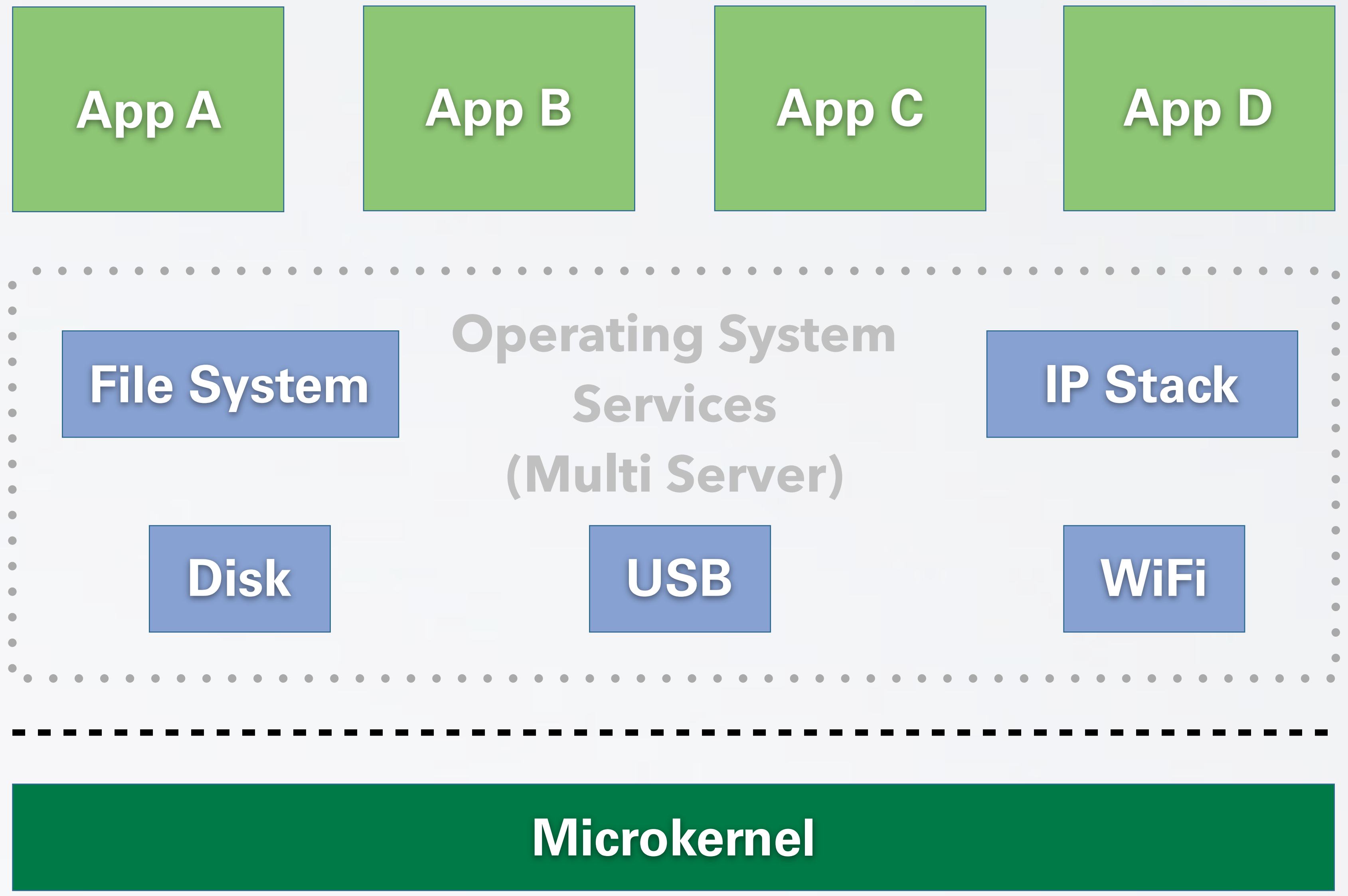
MONOLITHIC KERNEL

- Not only malicious applications
- Operating system kernel also exposed to untrusted input
  - Network packets and protocols
  - Thunderbolt, USB, other buses
  - File-system images
- One exploitable bug: kernel and all applications compromised



MICROKERNEL ARCHITECTURE

- Microkernels split OS into small and isolated components
- Better containment of faults and attacks (assuming safe interfaces)
- But:
  - Restricting interaction of components is still a big problem
  - Does not help against physical attacks (access to device)



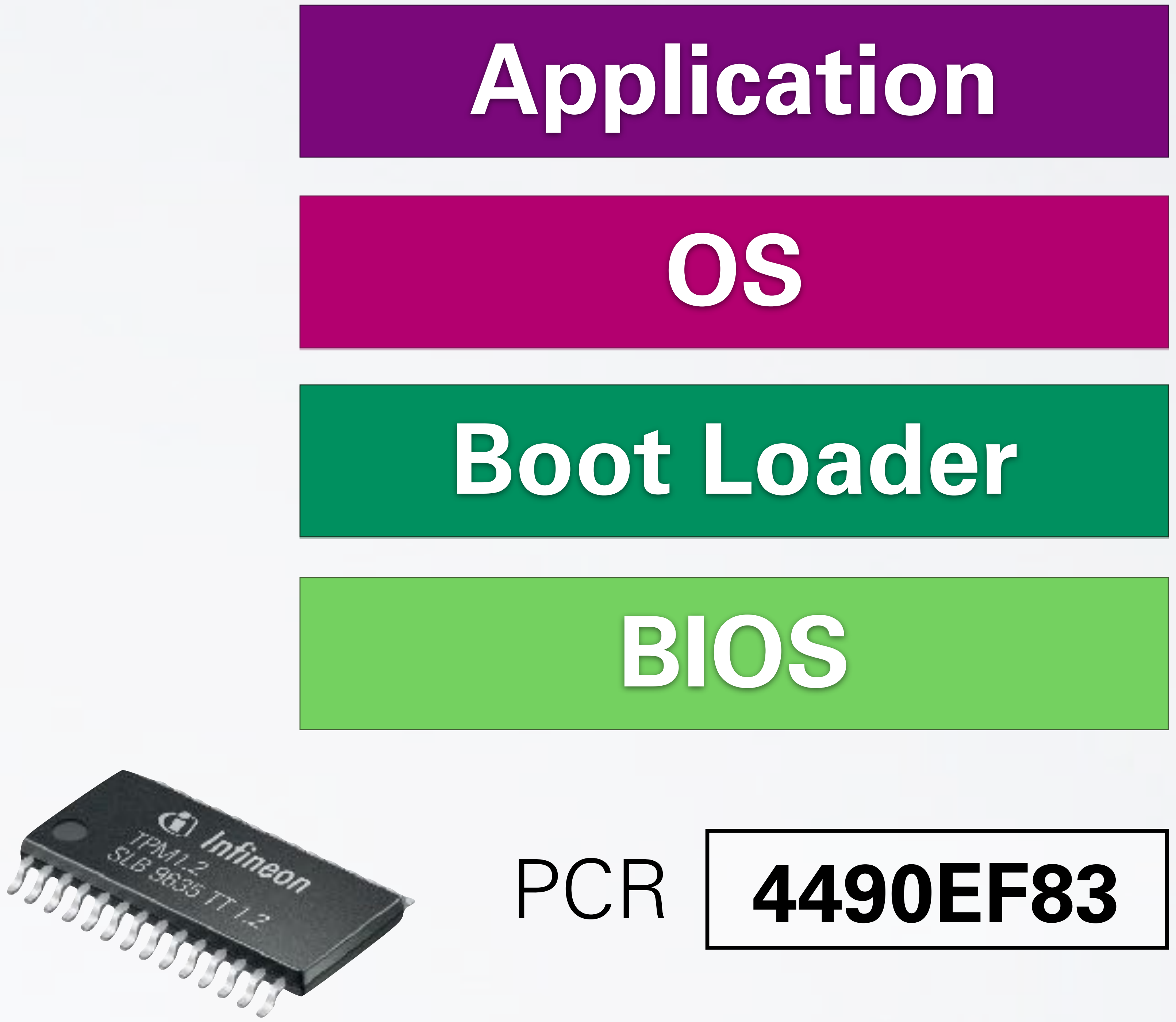
SECURITY ARCHITECTURES

SECURE BOOTING

- Software integrity rooted in hardware:
  - Only load and run software that matches a pre-determined checksum or public key (of the vendor)
  - If software does not match, refuse to load
  - Checksum or public key „fused" into hardware, cannot be exchanged
- Popular in system-on-chip (SOC) architectures, especially smartphones and tables
- Concept can be extended

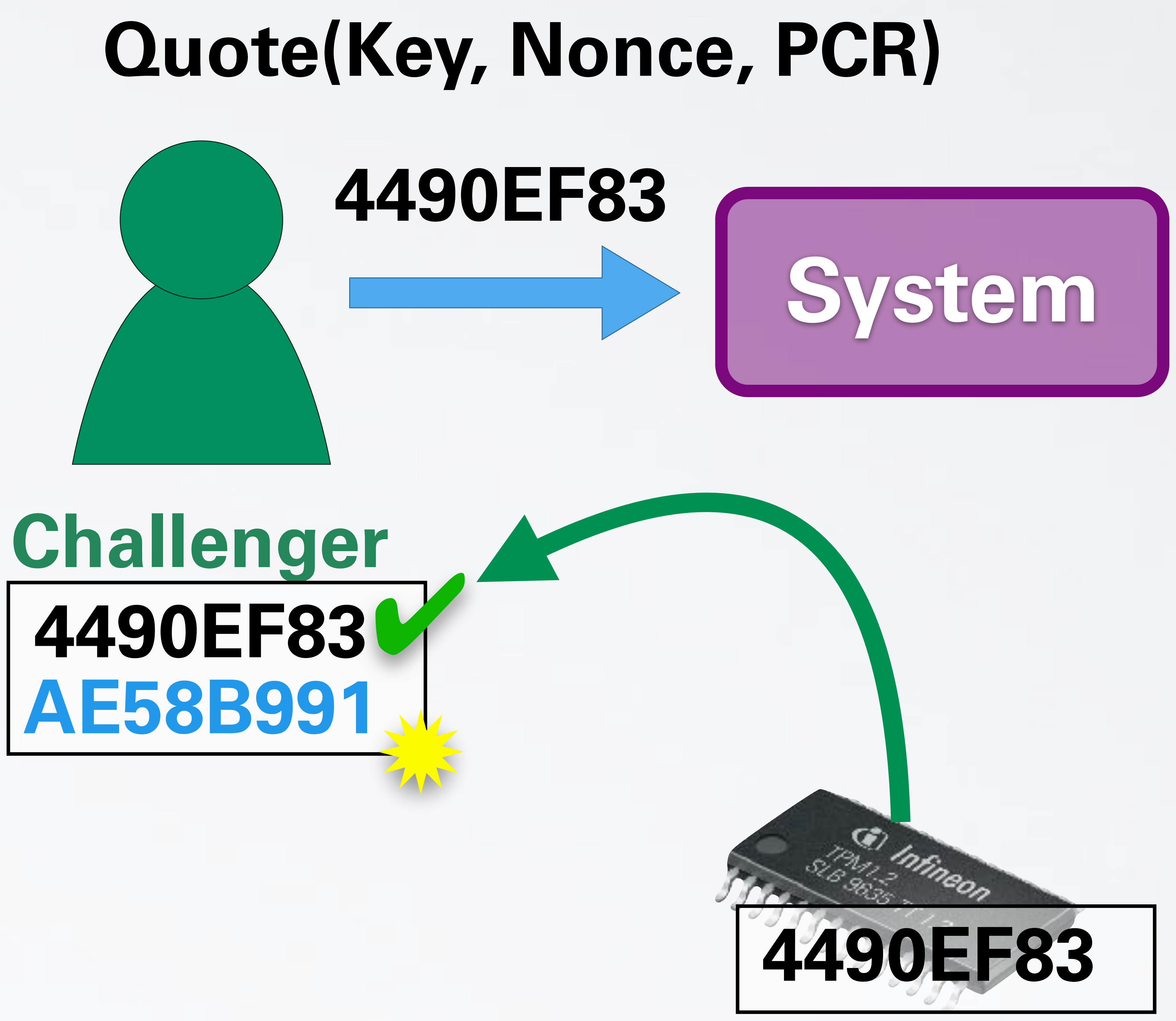
HARDWARE TRUST ANCHOR

- **Authenticated Booting:** Record the chain of trust in a tamper-proof hardware register, use as identity of loaded software stack
- **Sealed memory:** Encrypt data such that it will only be released, if expected software is running
- **Remote attestation:** Securely report identity to remote party



REMOTE ATTESTATION

- Challenger sends a random nonce to the system, he/she wants to have attested
- Challenged system responds with quote: identity of loaded software stack (PCR) + nonce, all signed using a private key
- Challenger check PCR signature based on known public key



Remote Attestation with Challenge/Response

HARDWARE-BASED ISOLATION

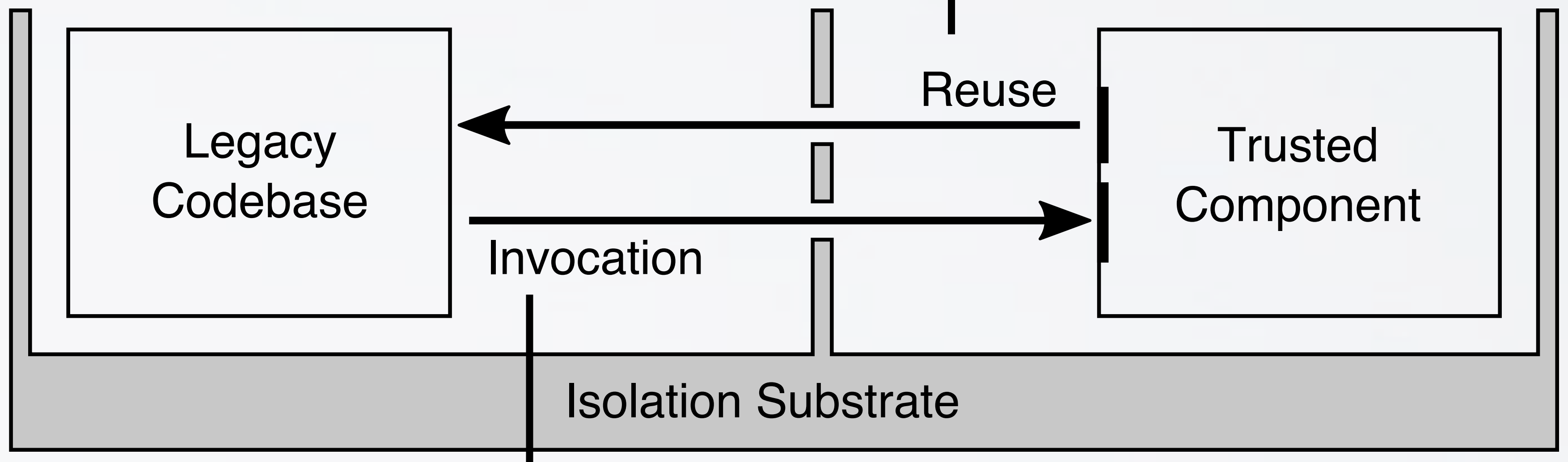
- Apple Security Processor
- ARM TrustZone
- Intel SGX

THE WAY FORWARD?

STRUCTURAL TEMPLATE

- **Isolation Substrate:** Spatial and temporal isolation
- **Legacy Codebase:** „old" code following monolithic design
- **Trusted Component:** Smaller, more secure, or just „my own"
- **Communication:** secure interaction between legacy code-base and trusted component

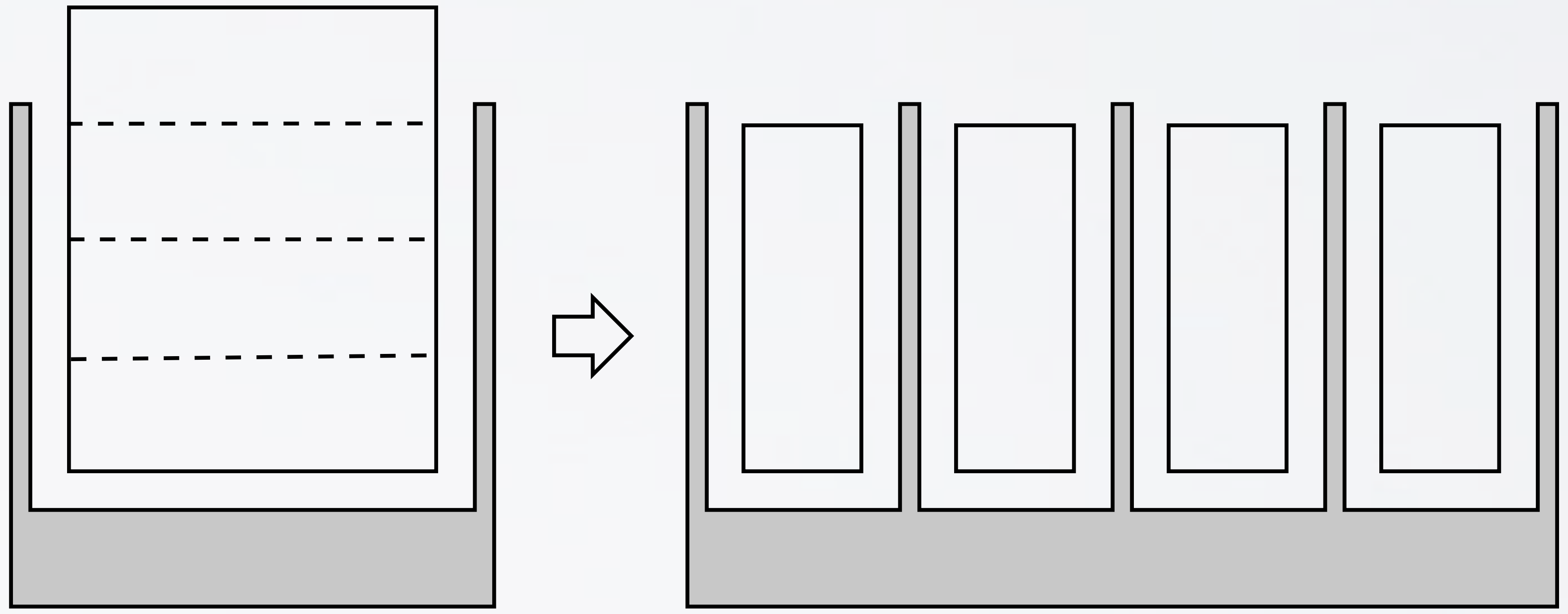
Secure reuse of legacy infrastructure by trusted component, usually involving cryptographic protection of data and/or extra security checks at interface boundary



Service invocation from untrusted legacy codebase, usually to protect a cryptographic secret or perform some other security-critical operation within the trusted component

VERTICAL TO LATERAL

- „Instead of vertically stacked libraries, we envision applications to be horizontal aggregates of communicating components, individually isolated from one another and mutually distrusting"
- Privileges of each component should be minimal (POLA)



EXAMPLE: SMART GRID