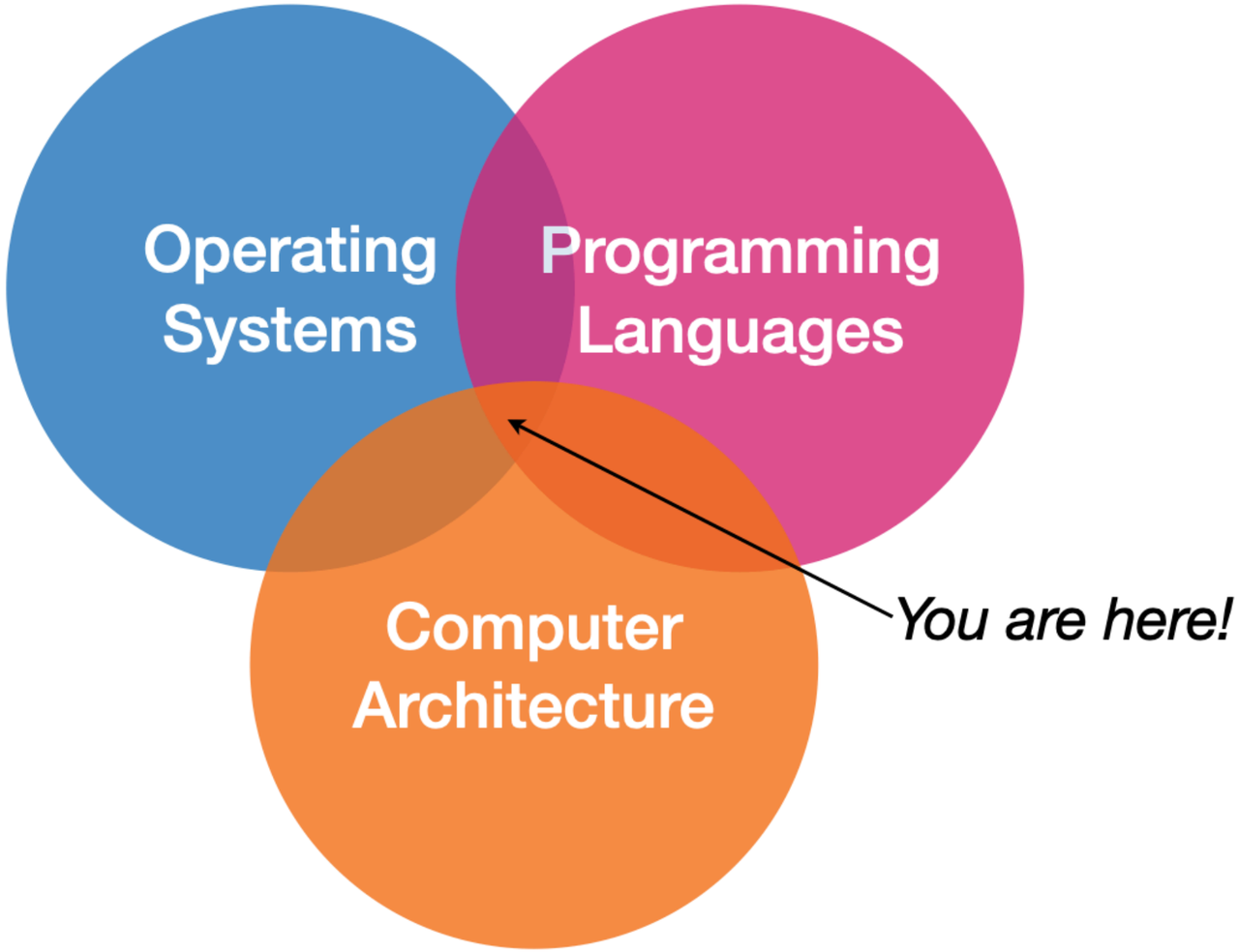
Securing the foundations:

Hardware-assisted secure Unikernels

KC Sivaramakrishnan

Security — A multi-dimensional challenge

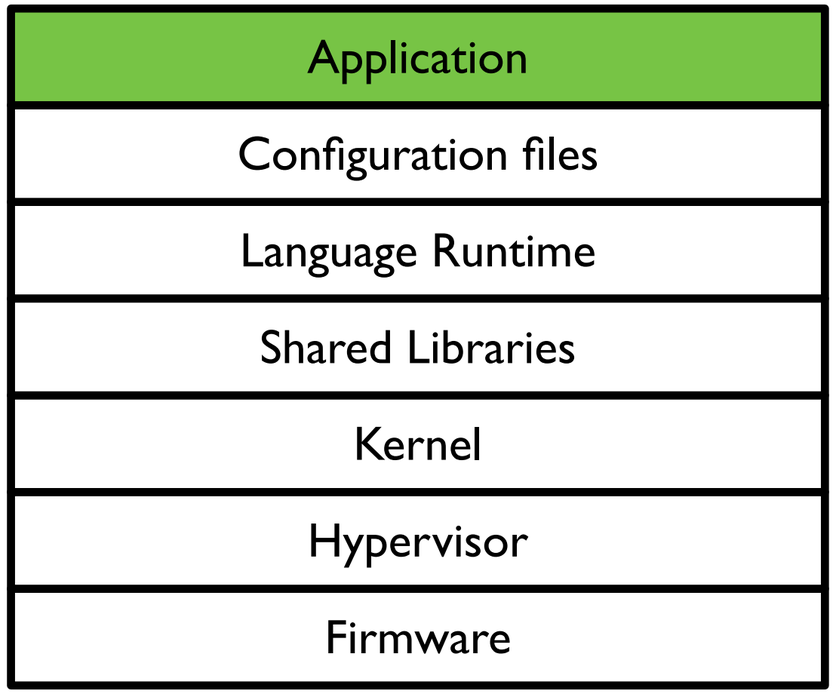
Security — A multi-dimensional challenge



Today

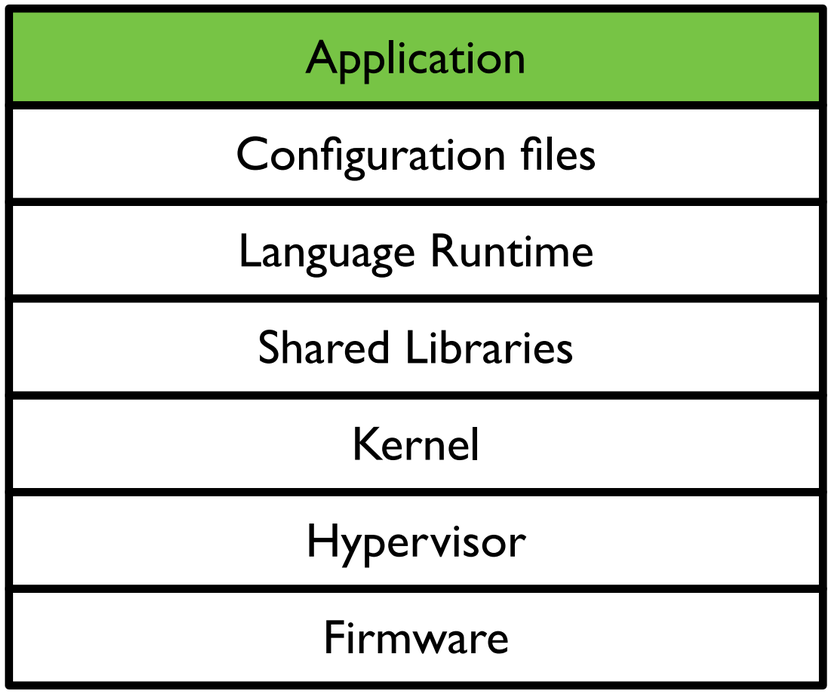
- Operating Systems
  - MirageOS — Small, safer, single-purpose OS
- Memory Safety
  - OCaml — memory-safe programming
- Going beyond memory safety
  - FIDES — Hardware-assisted intra-process isolation

Why do we need an operating system?



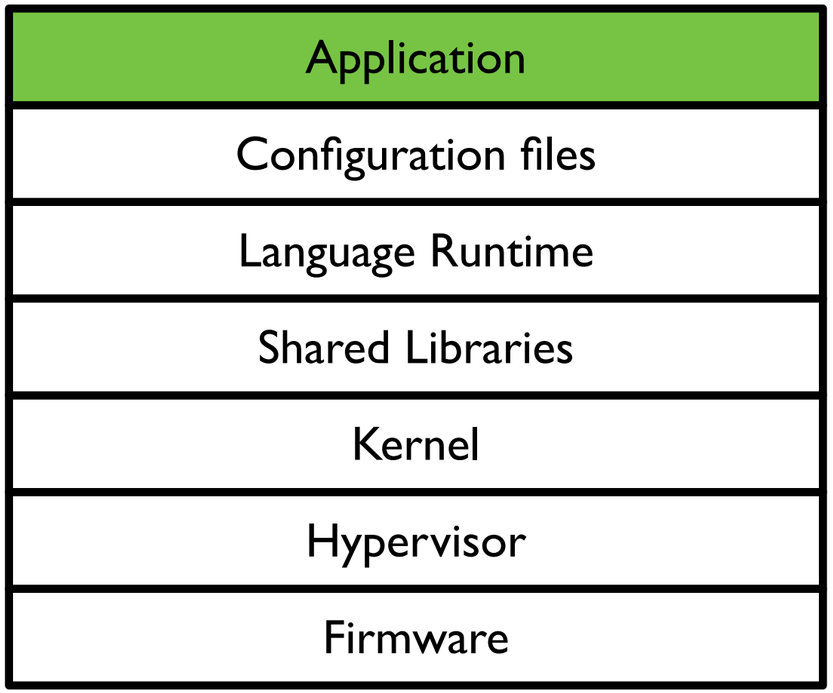
Why do we need an operating system?

- The main goal of an OS is to support running applications
  - **Stability:** most applications are not yet written when the system is deployed
  - **Scalability:** do not rewrite everything for every new hardware device



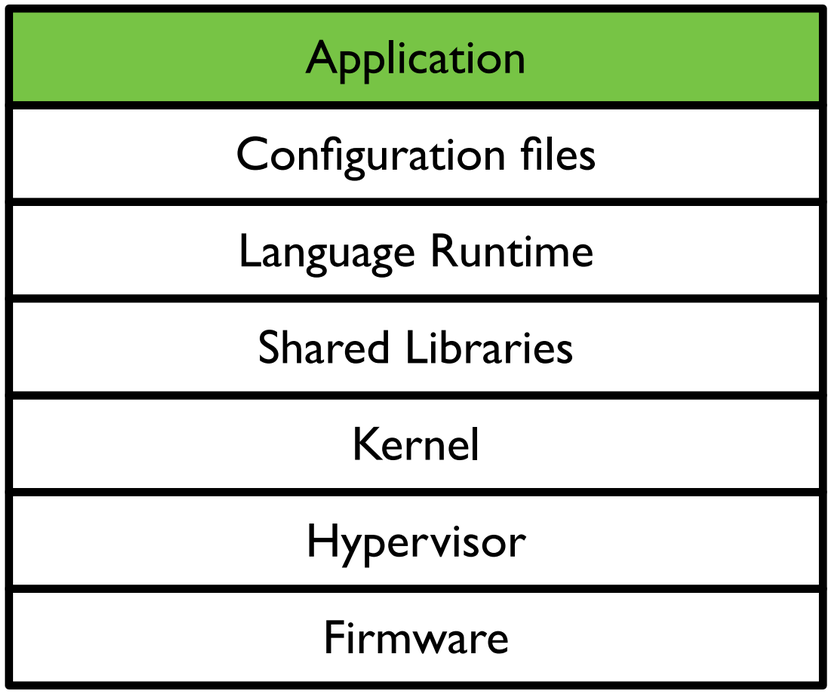
Why do we need an operating system?

- The main goal of an OS is to support running applications
  - **Stability:** most applications are not yet written when the system is deployed
  - **Scalability:** do not rewrite everything for every new hardware device
- OS does this by
  - Providing abstraction over hardware — drivers!
  - **Resource management:** files, users, CPU, memory, network



Why do we need an operating system?

- The main goal of an OS is to support running applications
  - **Stability:** most applications are not yet written when the system is deployed
  - **Scalability:** do not rewrite everything for every new hardware device
- OS does this by
  - Providing abstraction over hardware — drivers!
  - **Resource management:** files, users, CPU, memory, network
- Application code is ***a small %*** of the runtime environment

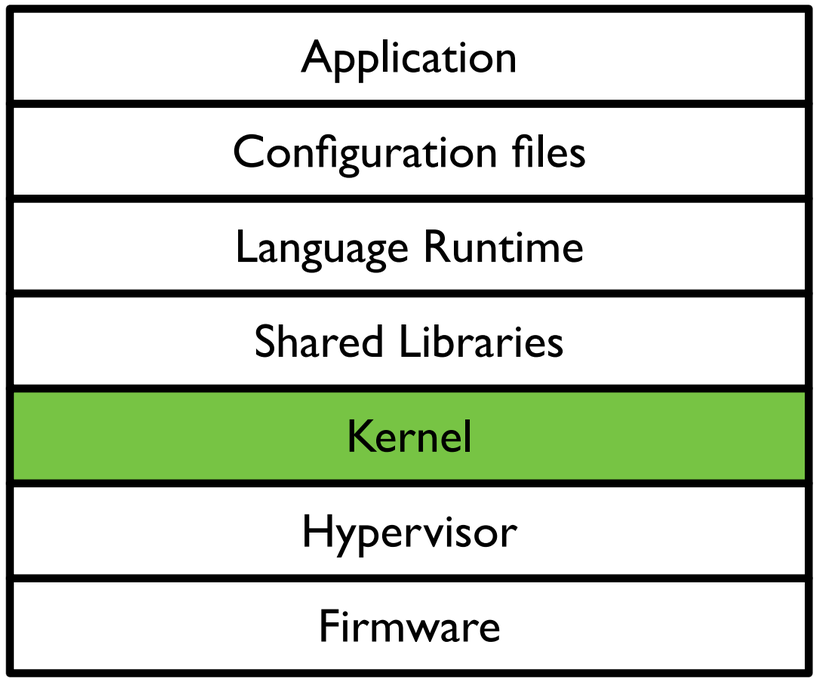


Kernel: A Core OS component

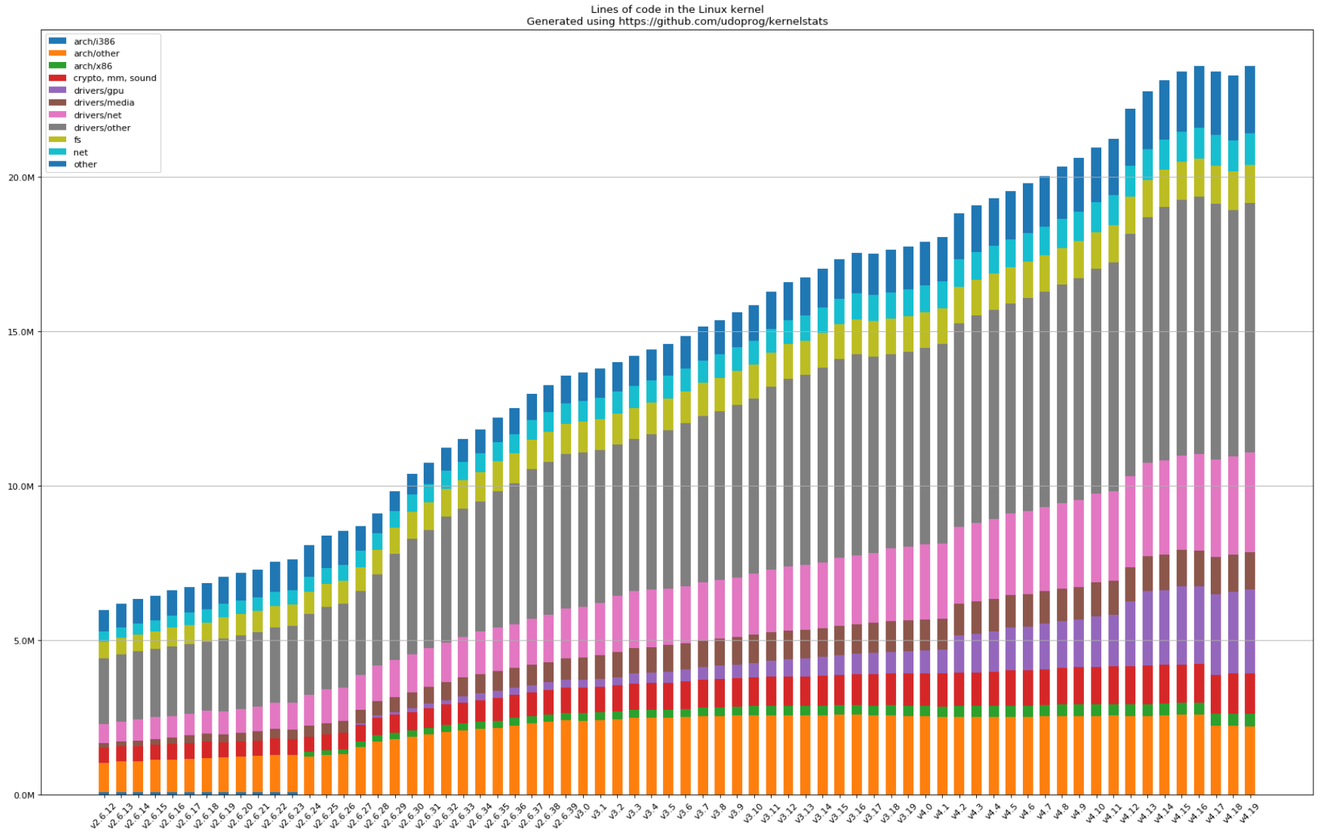
"*True, Linux is monolithic, and I agree that microkernels are nicer... As has been noted (not only by me), the Linux kernel is a minuscule part of a complete system:*

Full sources for Linux currently run to about 200kB compressed. And all of that source is portable, except for this tiny kernel that you can (provably: I did it) re-write totally from scratch in less than a year without having /any/ prior knowledge."

– Linus Torvalds, 1992



Linux Kernel



Drivers!

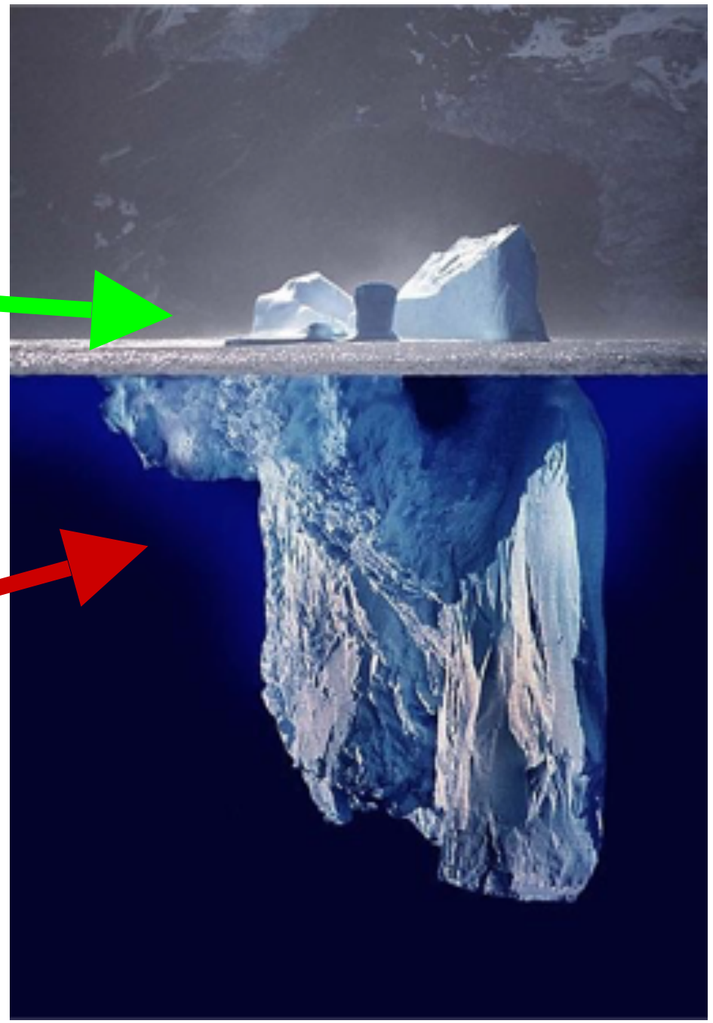
Linux 5.11 has 30.14 million lines of code, 60% drivers

Windows has 50 million lines of code

OS Icebergs

Code you want to run

Code your operating system insists you need!



How do we reduce the OS complexity?

Ingredient 1: Library OS

Library operating systems

Library operating systems

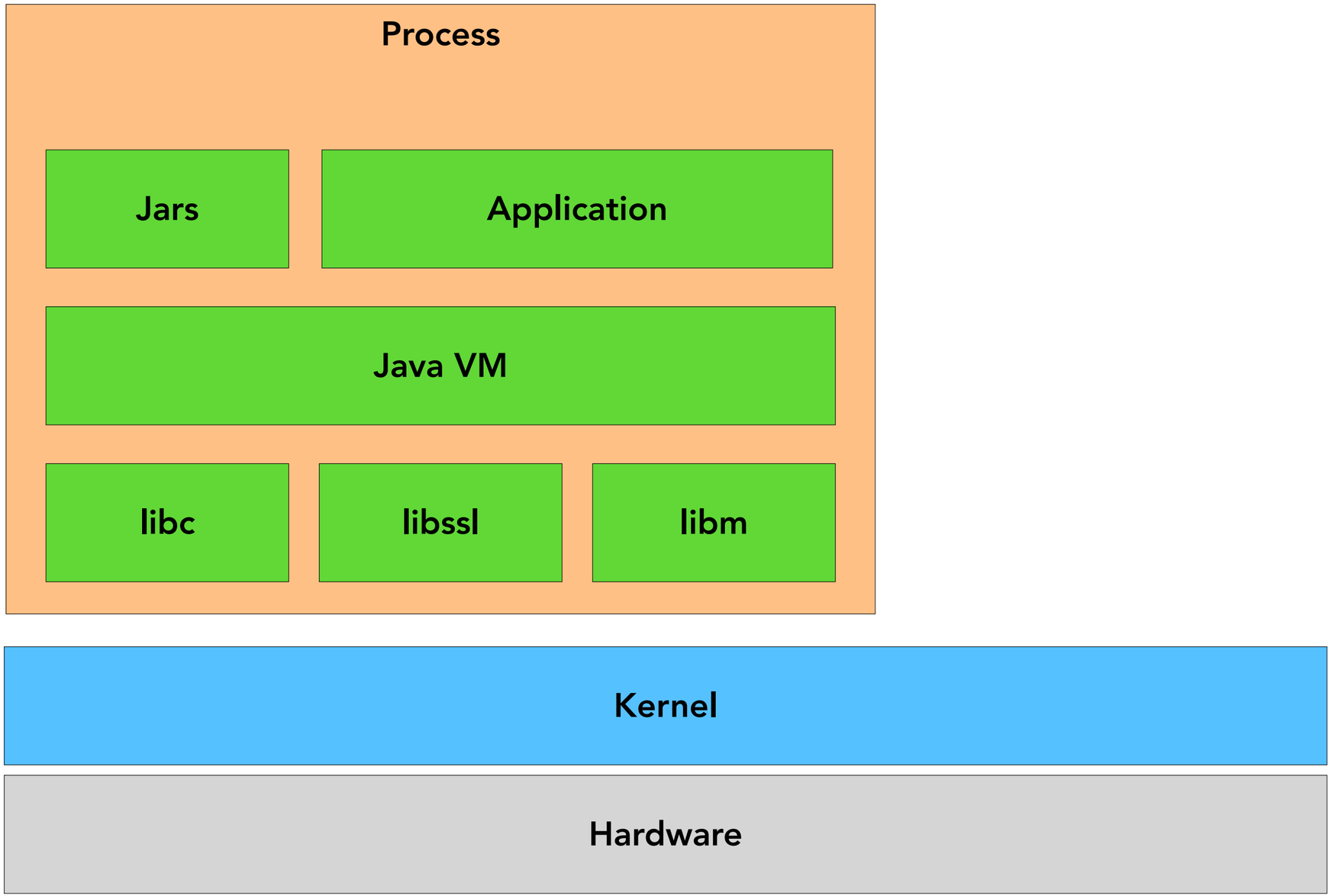
- Kernel functionality is broken up from its ***monolith*** into many ***individual libraries.***
  - There is no ambient kernel; just ***function calls*** are left.

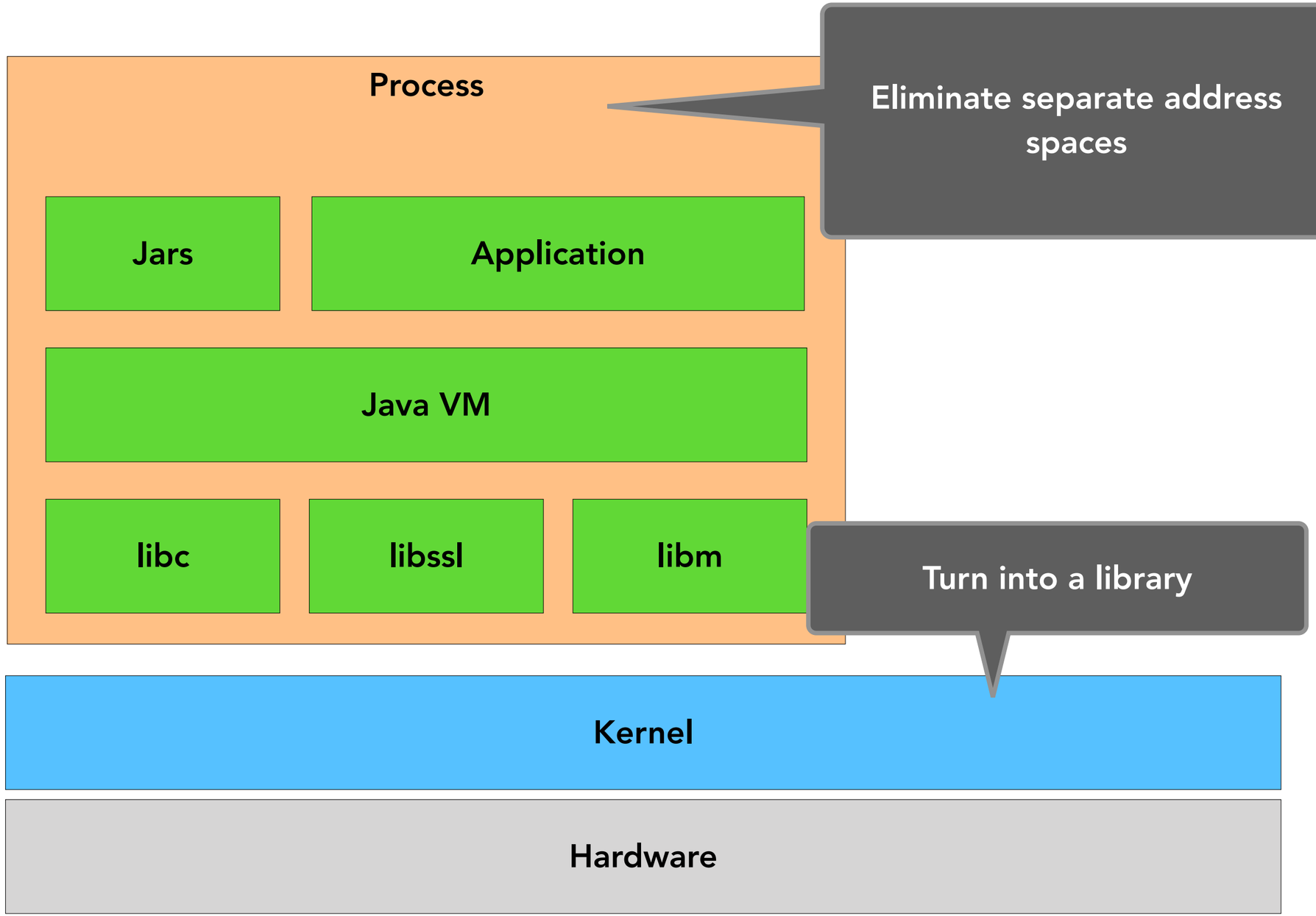
Library operating systems

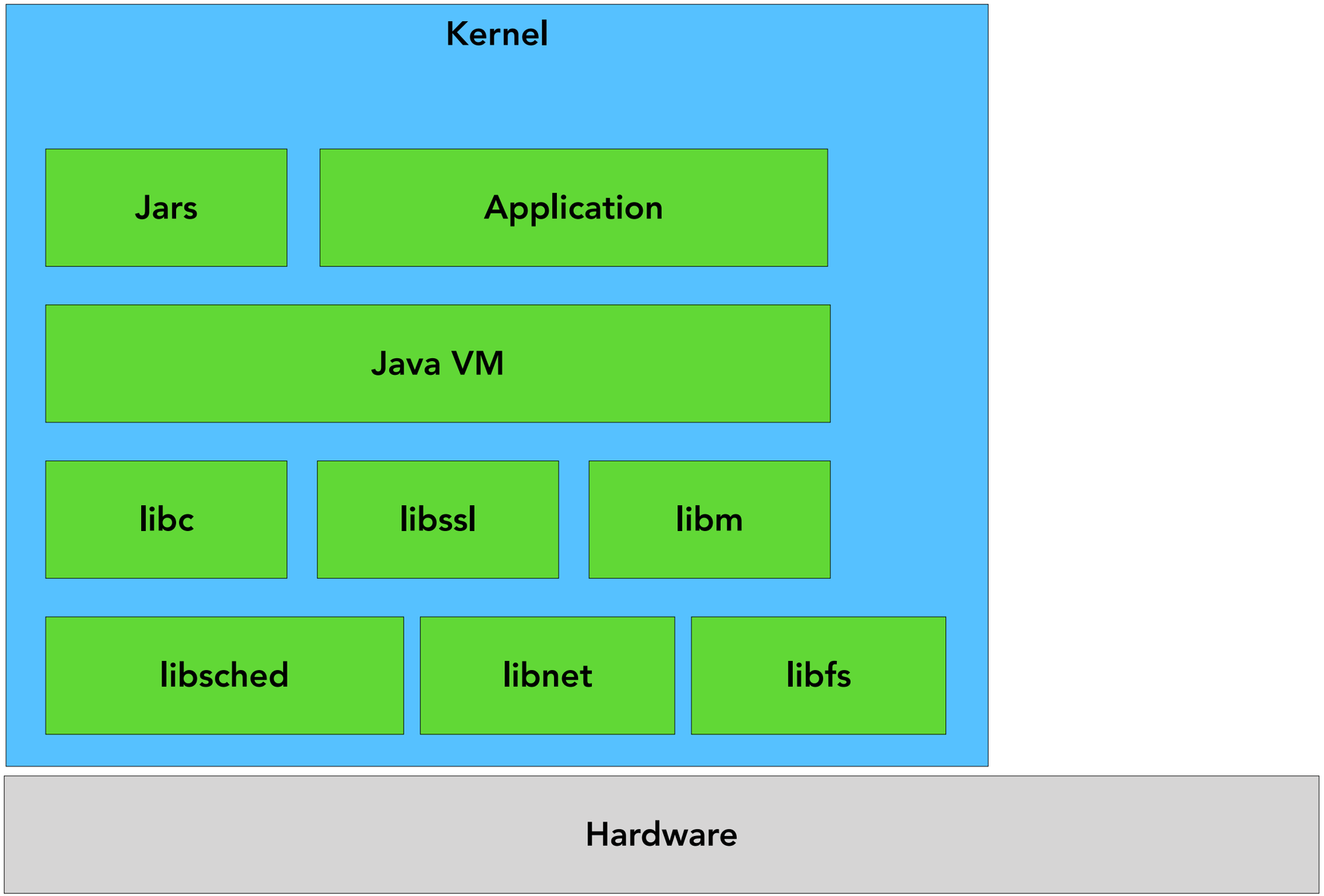
- Kernel functionality is broken up from its ***monolith*** into many ***individual libraries.***
  - There is no ambient kernel; just ***function calls*** are left.
- Device drivers, schedulers, networking, and storage stacks are ***directly linked*** to the application
  - Eliminate the need for an intermediary kernel layer.
  - Applications **select libraries they need** with a small boot layer and jump straight into the code.

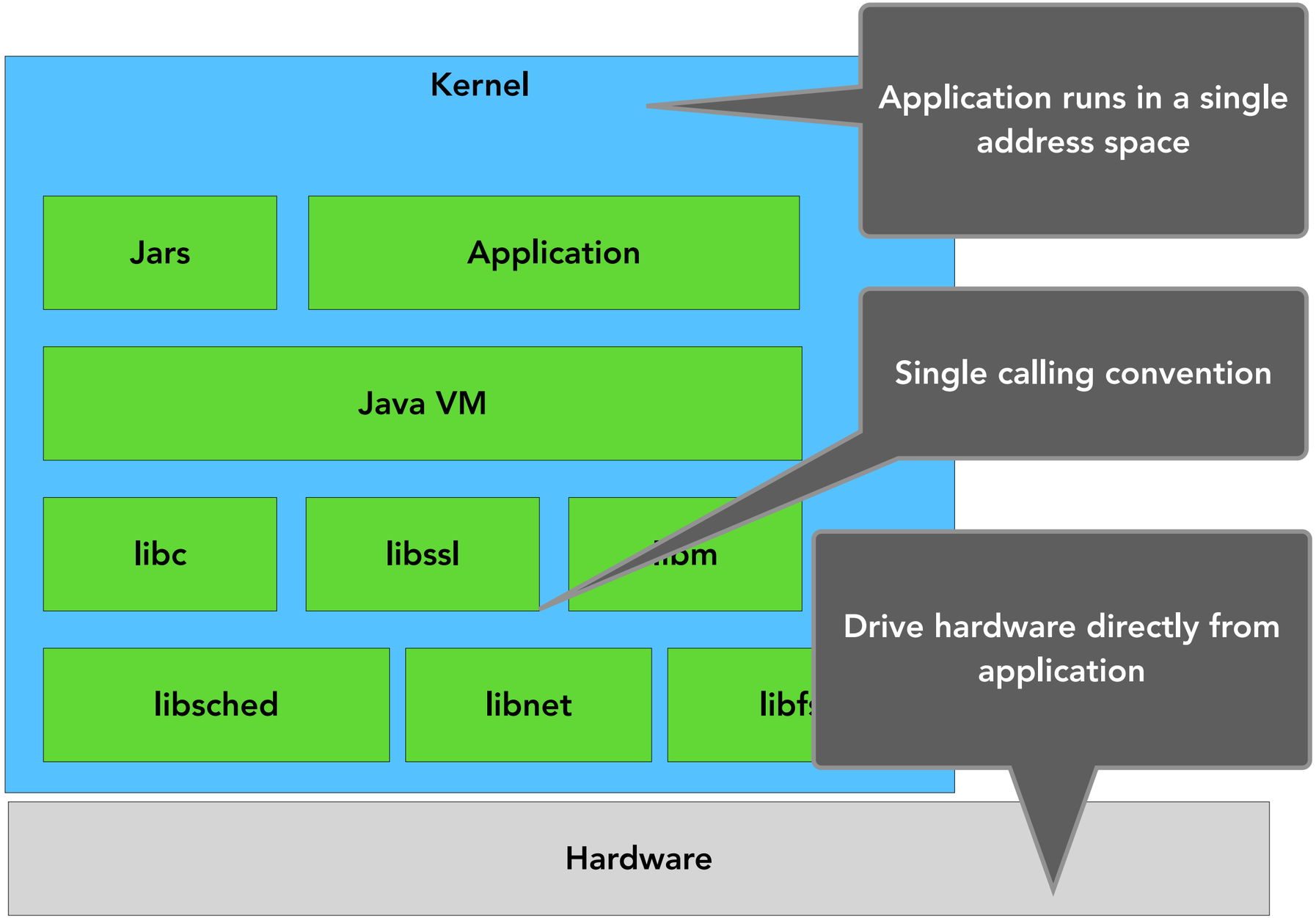
Library operating systems

- Kernel functionality is broken up from its ***monolith*** into many ***individual libraries.***
  - There is no ambient kernel; just ***function calls*** are left.
- Device drivers, schedulers, networking, and storage stacks are ***directly linked*** to the application
  - Eliminate the need for an intermediary kernel layer.
  - Applications **select libraries they need** with a small boot layer and jump straight into the code.
- Hardware is driven directly from the application, usually in a single address space.


Process
Jars
Application
Java VM
libc
libssl
libm
Kernel
Hardware


Process
Eliminate separate address spaces
Jars
Application
Java VM
libc
libssl
libm
Turn into a library
Kernel
Hardware


Kernel
Jars
Application
Java VM
libc
libssl
libm
libsched
libnet
libfs
Hardware


Kernel
Jars
Application
Java VM
libc
libssl
libm
libsched
libnet
libf
Hardware
Application runs in a single address space
Single calling convention
Drive hardware directly from application

Library operating systems

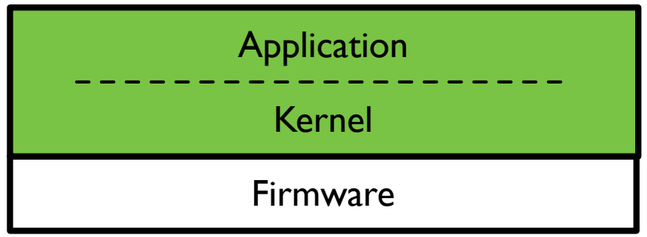
In the 1990s, we had:

- **Nemesis:** Cambridge/Glasgow
- **Exokernel:** MIT

Neither succeeded outside of academia due to the device drivers needing to be updated regularly to stay relevant.

Became popular in niche areas (network appliances or high-frequency trading).

Library operating systems



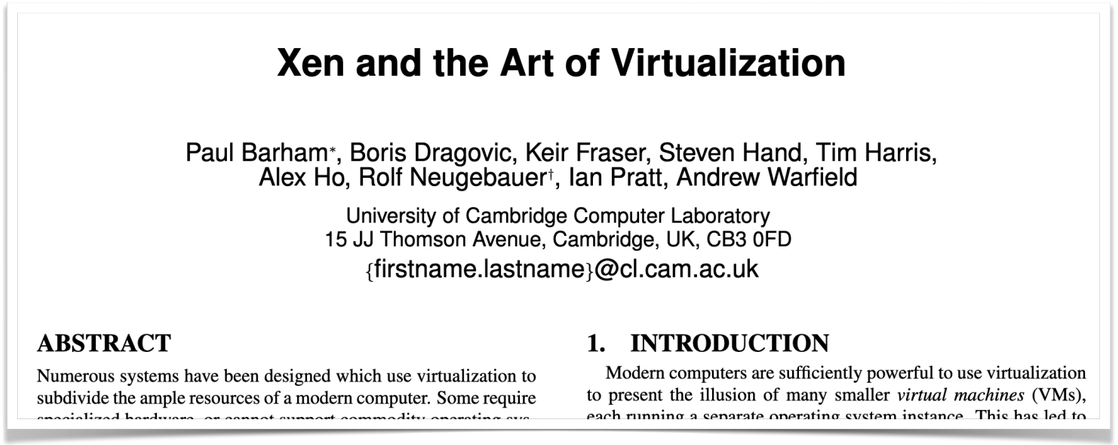
Pros: application-level control of hardware, small attack surface, high-performance.

Cons: There is no kernel protection internally, and device drivers all need to be rewritten from a normal kernel.

Ingredient 2: Virtualisation

Virtualisation

- In the 2000s, hardware vendors added extensions that allow the creation of virtual versions of physical resources, such as servers, networks, and storage devices.
- It enables multiple virtual machines (VMs), with their own operating systems, to **run in isolation, side-by-side, on the same physical hardware**.
- Hypervisor (aka VMM) — creates and runs virtual machines



Xen and the Art of Virtualization

Paul Barham*, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris,
Alex Ho, Rolf Neugebauer†, Ian Pratt, Andrew Warfield

University of Cambridge Computer Laboratory
15 JJ Thomson Avenue, Cambridge, UK, CB3 0FD
{firstname.lastname}@cl.cam.ac.uk

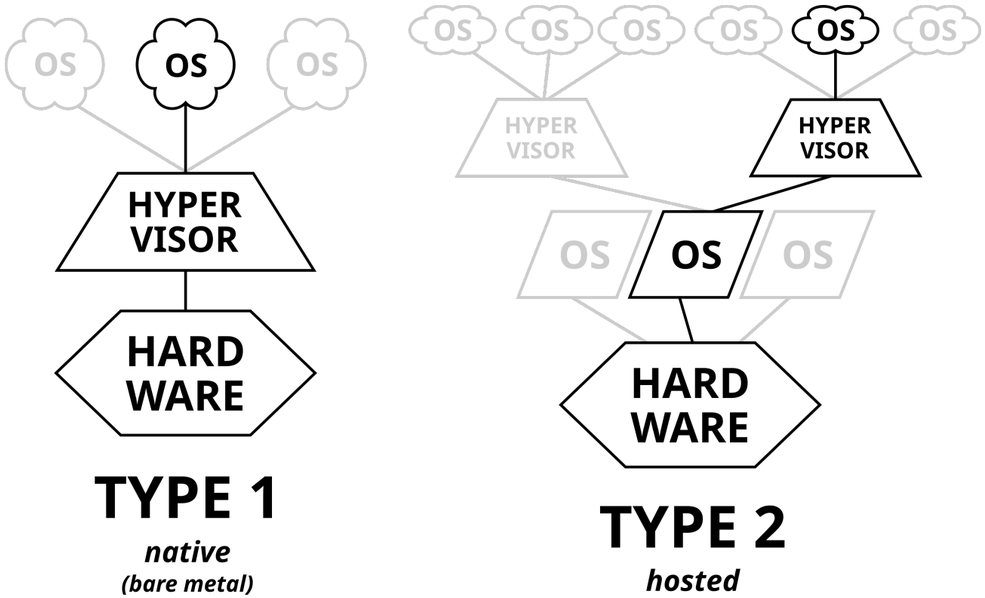
ABSTRACT

Numerous systems have been designed which use virtualization to subdivide the ample resources of a modern computer. Some require

1. INTRODUCTION

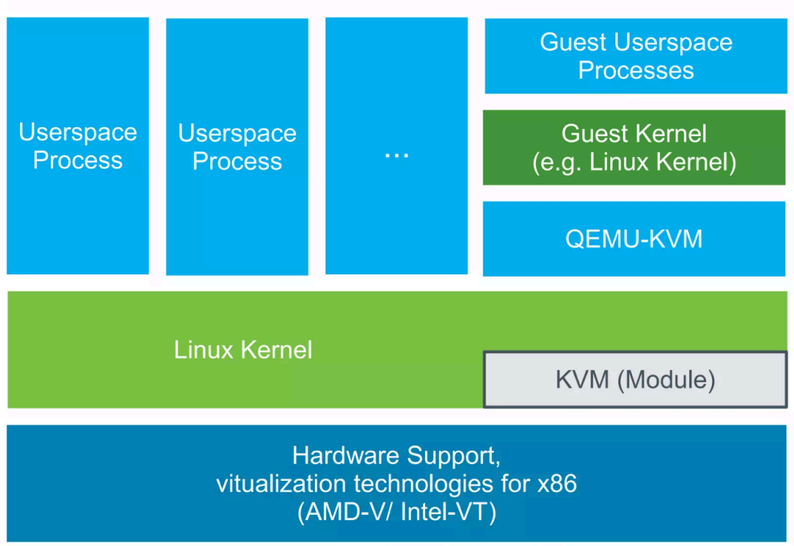
Modern computers are sufficiently powerful to use virtualization to present the illusion of many smaller *virtual machines* (VMs),

Virtualisation



- **Type 1** — KVM (converts Linux to a type 1 hypervisor), VMware ESXi, Microsoft Hyper-V, Citrix XenServer
- **Type 2** — VirtualBox, VMware Workstation, Microsoft Virtual PC

Linux KVM



- Turns Linux into a Type 1 VMM
- QEMU emulates CPUs and missing hardware
- **VirtIO** — virtualisation of networks and disk device drivers
  - *Can take advantage of Linux Kernel's vast driver support!*

Library operating systems

Cons: There is no kernel protection internally, and ~~device drivers all need to be rewritten from a normal kernel.~~

Ingredient 3: OCaml

Memory safety

Library operating systems

Cons: There is no kernel protection internally, and ~~device drivers all need to be rewritten from a normal kernel.~~

Memory safety

Library operating systems

Cons: There is no kernel protection internally, and ~~device drivers all need to be rewritten from a normal kernel.~~

Microsoft: 70 percent of all security bugs are memory safety issues

Percentage of memory safety issues has been hovering at 70 percent for the past 12 years.

Written by **Catalin Cimpanu,** Contributor

Feb. 11, 2019 at 7:48 a.m. PT

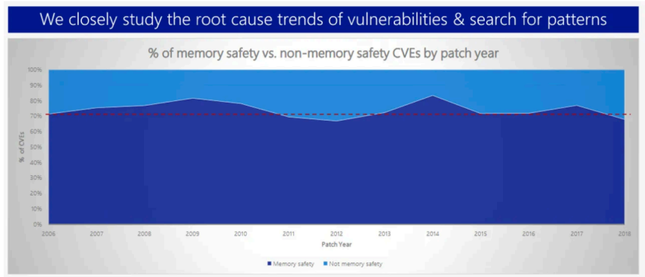


Image: Matt Miller

/ related

Worried about the Windows BitLocker recovery bug? 6 things you need to know

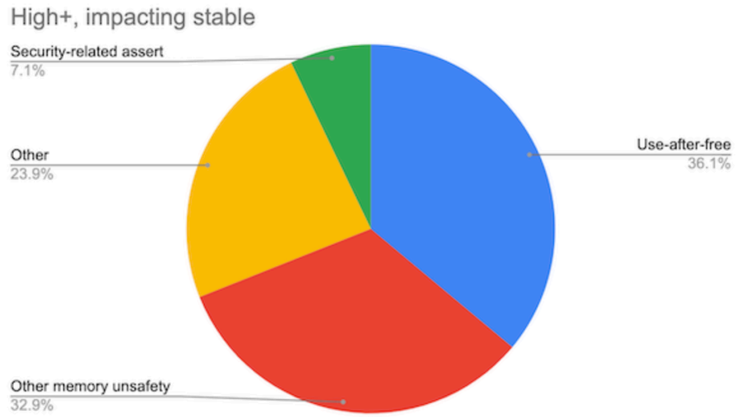
The Windows 10 clock is ticking: 5 ways to save your old PC in 2025 (most are free)

Memory safety

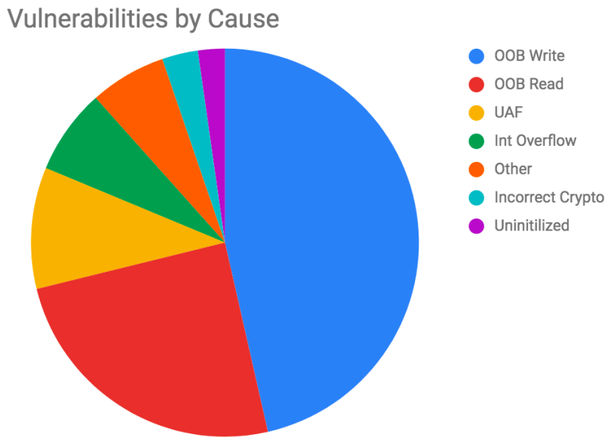
The Chromium project finds that around 70% of our serious security bugs are memory safety problems. Our next major project is to prevent such bugs at source.

The problem

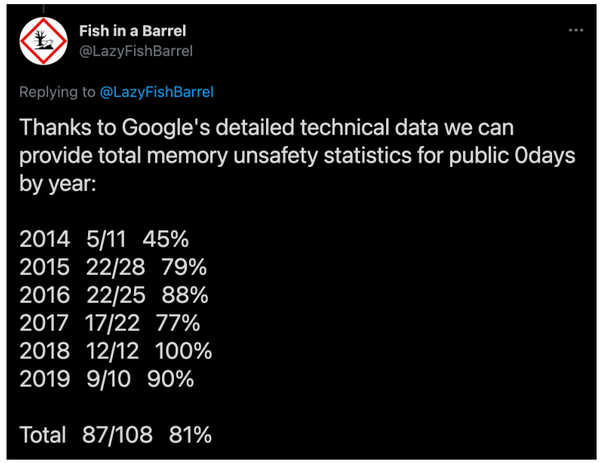
Around 70% of our high severity security bugs are memory unsafety problems (that is, mistakes with C/C++ pointers). Half of those are use-after-free bugs.



Memory safety



90% of Android vulnerabilities are memory safety issues



80% of the exploited vulnerabilities of known 0-days were memory safety issues

Memory safety

The Case for Memory Safe Roadmaps

Why Both C-Suite Executives and Technical Experts Need to Take Memory Safe Coding Seriously

Publication: December 2023

United States Cybersecurity and Infrastructure Security Agency
United States National Security Agency
United States Federal Bureau of Investigation
Australian Signals Directorate's Australian Cyber Security Centre
Canadian Centre for Cyber Security
United Kingdom National Cyber Security Centre
New Zealand National Cyber Security Centre
Computer Emergency Response Team New Zealand

THE WHITE HOUSE

MENU

FEBRUARY 26, 2024

Press Release: Future Software Should Be Memory Safe

ONCD › BRIEFING ROOM › PRESS RELEASE

Leaders in Industry Support White House Call to Address Root Cause of Many of the Worst Cyber Attacks

Read the full report here

Memory safety and Programming Languages

- Unsafe languages
  - C, C++, Assembly, Objective-C

Memory safety and Programming Languages

- Unsafe languages
  - C, C++, Assembly, Objective-C
- Safe languages
  - With the help of a garbage collector (GC) — JavaScript, Python, Java, Go, OCaml, …
  - Without a GC — Rust

Memory safety and Programming Languages

- Unsafe languages
  - C, C++, Assembly, Objective-C
- Safe languages
  - With the help of a garbage collector (GC) — JavaScript, Python, Java, Go, OCaml, …
  - Without a GC — Rust
- Unsafe parts of safe languages
  - Unsafe Rust, unsafe package in Go, Obj in OCaml

Memory safety and Programming Languages

- Unsafe languages
  - C, C++, Assembly, Objective-C
- Safe languages
  - With the help of a garbage collector (GC) — JavaScript, Python, Java, Go, OCaml, …
  - Without a GC — Rust
- Unsafe parts of safe languages
  - Unsafe Rust, unsafe package in Go, Obj in OCaml

Library operating systems

Cons: ~~There is no kernel protection internally, and device drivers all need to be rewritten from a normal kernel.~~

industrial-strength, pragmatic, functional programming language

industrial-strength, pragmatic, functional programming language

Industry

Projects

industrial-strength, pragmatic, functional programming language

Industry

Projects

Higher-order functions

Hindley-Milner Type Inference

Powerful module system

industrial-strength, pragmatic, functional programming language

Industry

FACEBOOK
Tarides
Bloomberg
docker
ahrefs
SimCorp
Microsoft
Tezos
Jane Street

Projects

hack
flow
COMPCERT

Higher-order functions

Hindley-Milner Type Inference

Powerful module system

Functional core with imperative and object-oriented features

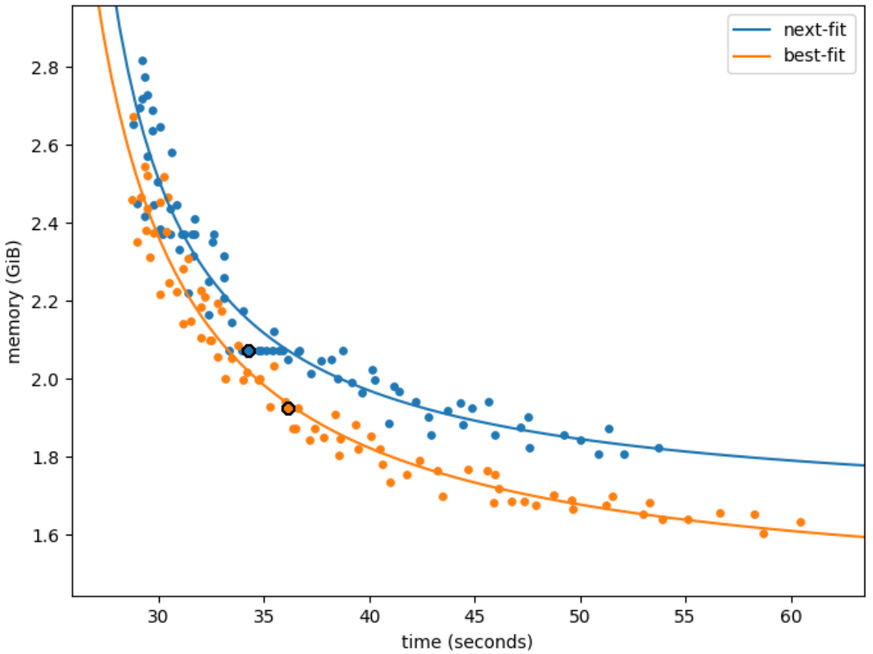
Native (x86, Arm, Power, RISC-V), JavaScript, WebAssembly

OCaml Performance

- GC is tuned for low-latency
  - If your application can tolerate 1 ms latency, then OCaml is a good fit
  - 95% of code that we write fit this model

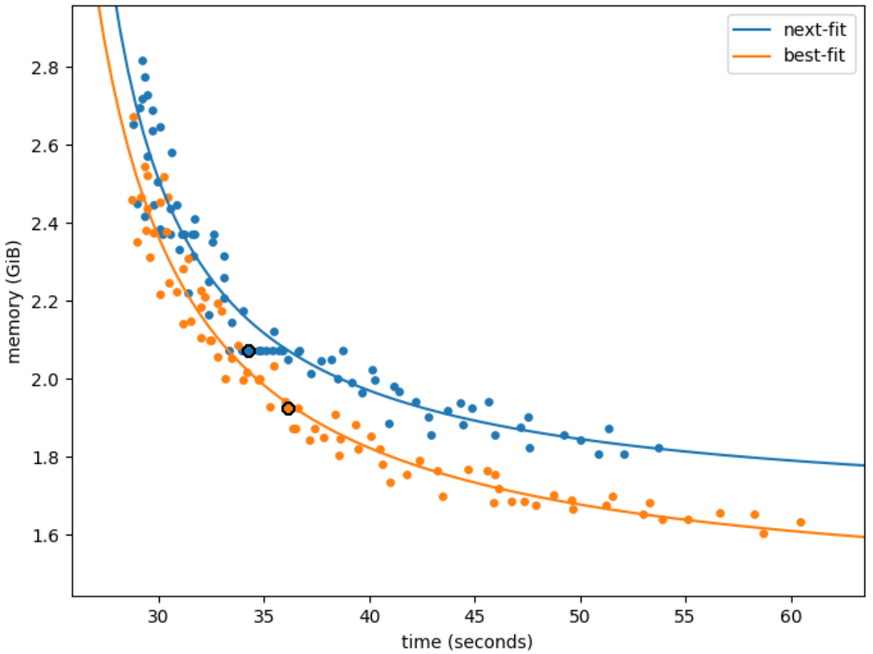
OCaml Performance

- GC is tuned for low-latency
  - If your application can tolerate 1 ms latency, then OCaml is a good fit
  - 95% of code that we write fit this model
- GC is a tradeoff between space and time



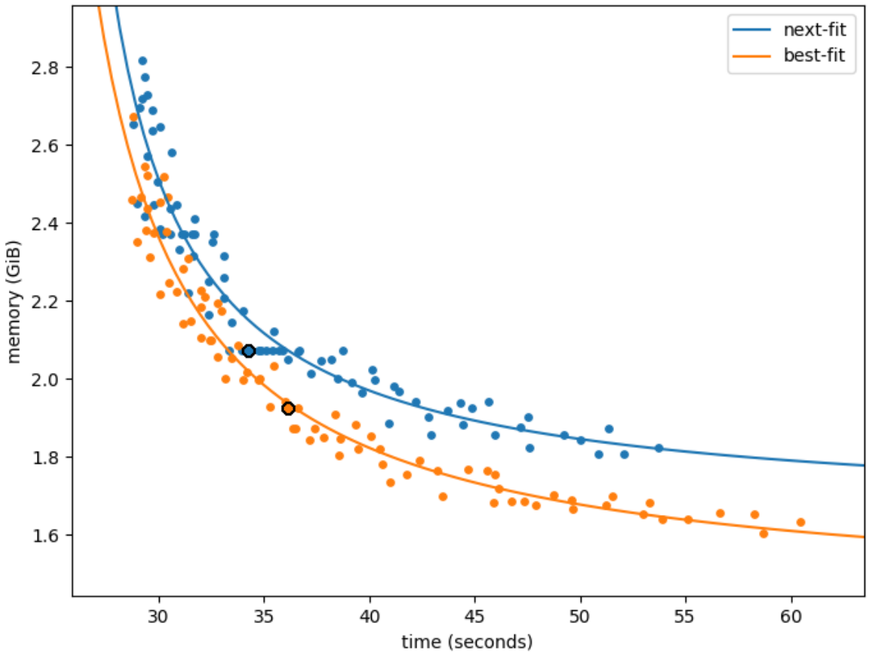
OCaml Performance

- GC is tuned for low-latency
  - If your application can tolerate 1 ms latency, then OCaml is a good fit
  - 95% of code that we write fit this model
- GC is a tradeoff between space and time
- OCaml is typically 1.5x to 2x slower than C for algorithmic workloads
  - Python will be 10x to 100x slower than C

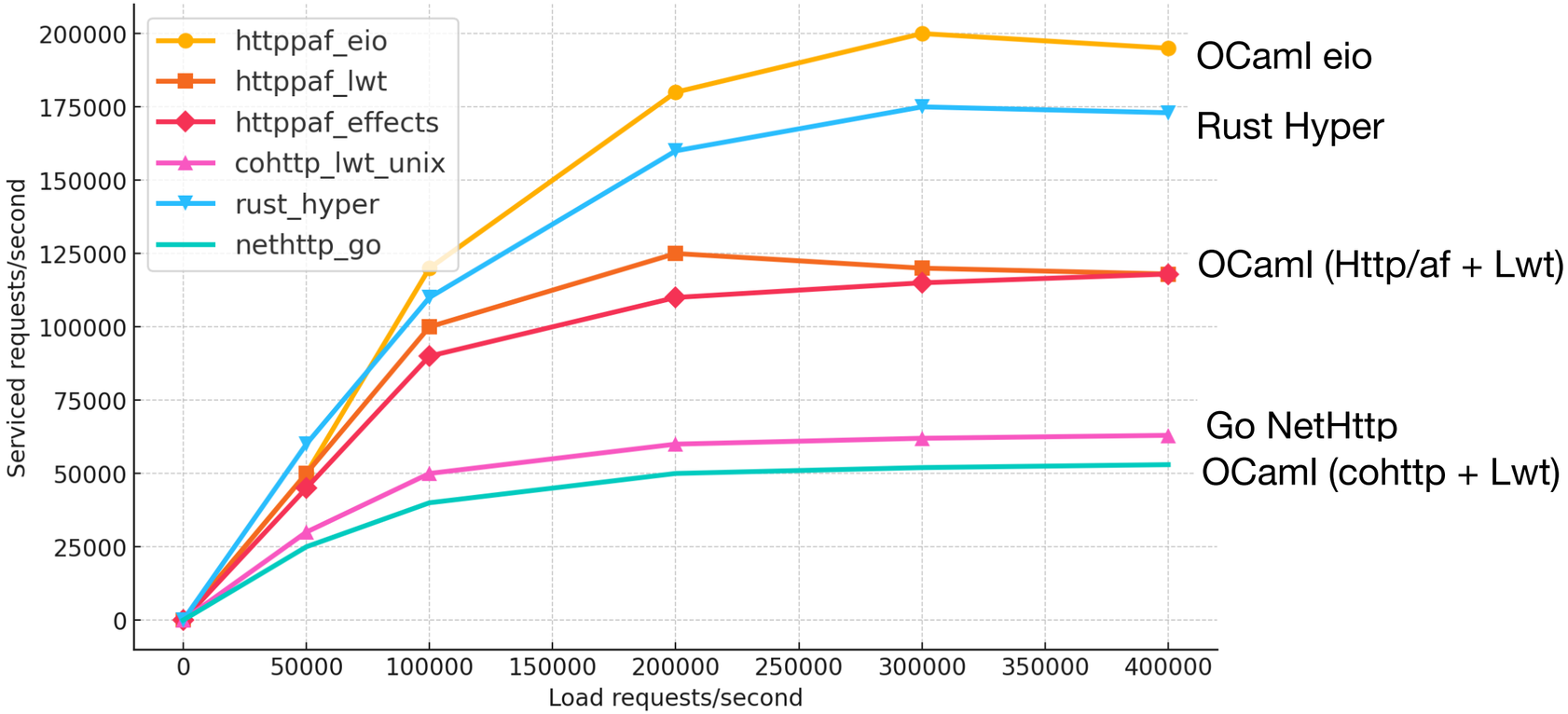


OCaml Performance

- GC is tuned for low-latency
  - If your application can tolerate 1 ms latency, then OCaml is a good fit
  - 95% of code that we write fit this model
- GC is a tradeoff between space and time
- OCaml is typically 1.5x to 2x slower than C for algorithmic workloads
  - Python will be 10x to 100x slower than C
- Fast FFI to C for speed



OCaml Performance — Web Server

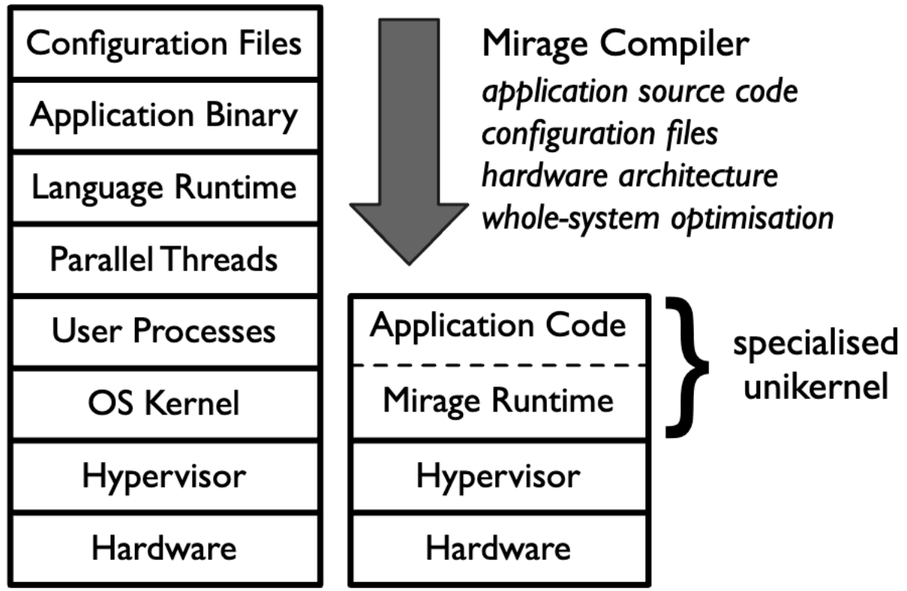


https://github.com/ocaml-multicore/eio

MirageOS = Library OS + Virtualisation + OCaml

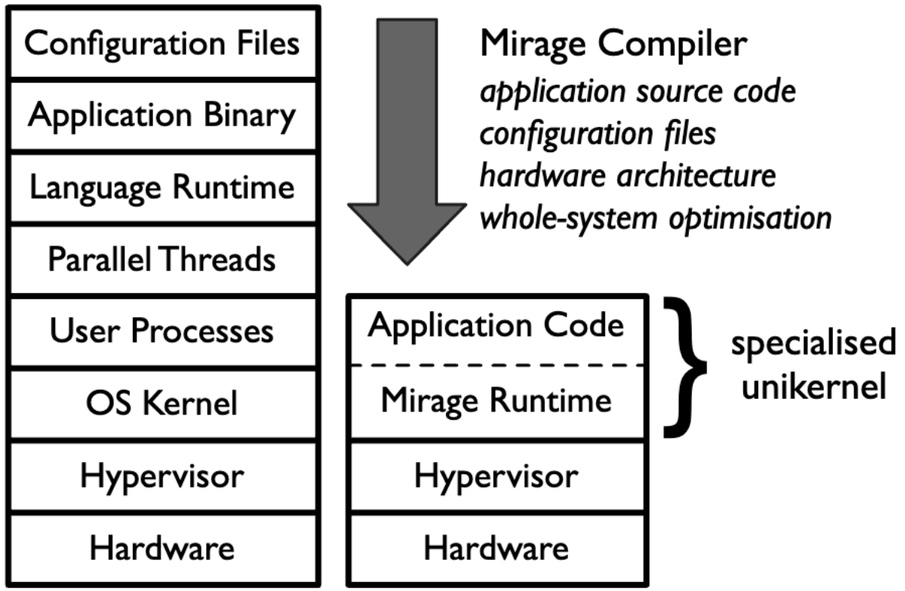
MirageOS Unikernels

- MirageOS is a **library OS** and a **compiler** that can build specialised images containing only the runtime environment needed by the application
  - Cut the complexity by designing the layers as independent type-safe libraries.



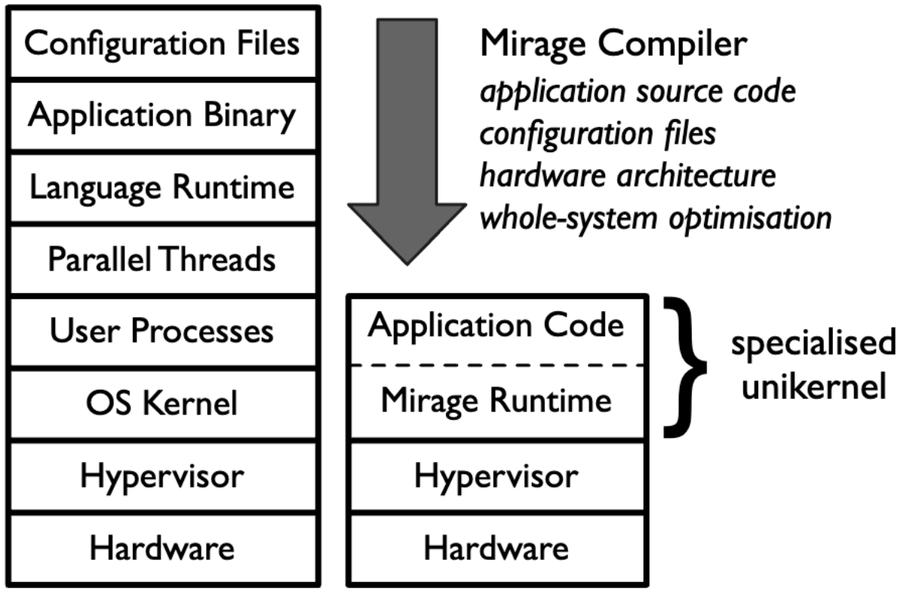
MirageOS Unikernels

- MirageOS is a **library OS** and a **compiler** that can build specialised images containing only the runtime environment needed by the application
  - Cut the complexity by designing the layers as independent type-safe libraries.
- The MirageOS compiler transforms an application manifest into a **specialised image.**
  - Rely on the OCaml compiler for modular static analysis, dead-code elimination, etc.



MirageOS Unikernels

- MirageOS is a **library OS** and a **compiler** that can build specialised images containing only the runtime environment needed by the application
  - Cut the complexity by designing the layers as independent type-safe libraries.
- The MirageOS compiler transforms an application manifest into a **specialised image.**
  - Rely on the OCaml compiler for modular static analysis, dead-code elimination, etc.
- Rely on the OCaml runtime as the sole trusted runtime environment (and selected C bindings)



Available Libraries

Network:
Ethernet, IP, UDP, TCP, HTTP 1.0/1.1/2.0, ALPN, DNS, ARP, DHCP, SMTP, IRC, cap-n-proto, emails
Storage:
block device, Ramdisk, Qcow, B-trees, VHD, Zlib, Gzip, Lzo, Git, Tar, FAT32
Data-structures:
LRU, Rabin's fingerprint, bloom filters, adaptative radix trees, discrete interval encoding trees
Security:
x.509, ASN1, TLS, SSH
Crypto:
hashes, checksums
Ciphers (AES, 3DES, RC4, ChaCha20/Poly1305)
AEAD primitives (AES-GCM, AES-CCM)
Public keys (RSA, DSA, DH)
Fortuna

- Reimplemented in OCaml
- TLS: "rigorous engineering"
  - same pure code to generate test oracles, verify oracle against real-world TLS traces and the real implementation
  - Use Fiat (Coq extraction) for crypto primitives.

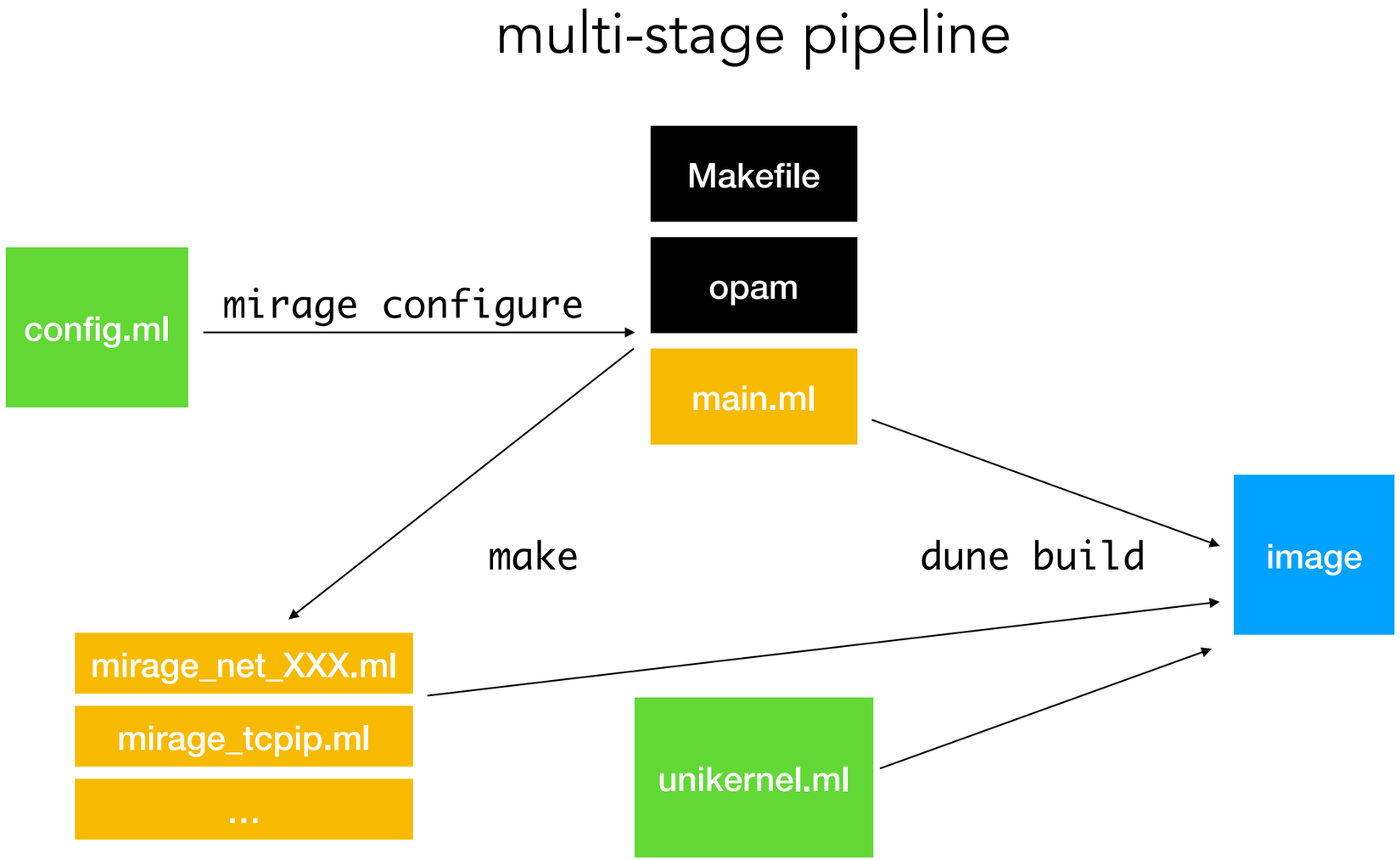
Not-quite-so-broken TLS: lessons in re-engineering a security protocol specification and implementation

David Kaloper-Meršinjak†, Hannes Mehnert†, Anil Madhavapeddy and Peter Sewell
University of Cambridge Computer Laboratory
first.last@cl.cam.ac.uk
† *These authors contributed equally to this work*

What is a MirageOS Unikernel?

- A statically compiled ELF binary
- Executed as a virtual machine
  - Solo5 is the host system process (“tender”)
    - Provides the platform-specific details for MirageOS applications to interact with the underlying hardware or virtualisation frameworks
  - Supports — KVM, Xen, virtio, muen, Linux Seccomp
- Can also be executed as a Unix process
  - Useful for debugging and development

MirageOS Compiler



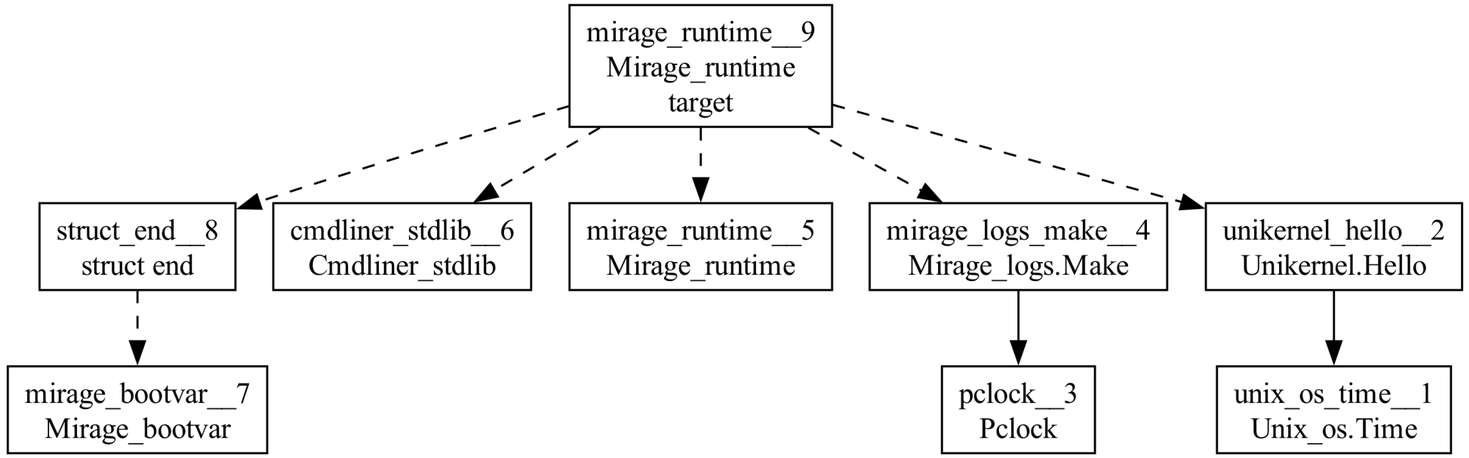
Hello Unikernel — unikernel.ml

```
open Lwt.Infix

module Hello (Time : Mirage_time.S) = struct
  let start _time =
    let rec loop = function
      | 0 -> Lwt.return_unit
      | n ->
          Logs.info (fun f -> f "hello");
          Time.sleep_ns (Duration.of_sec 1) >>= fun () -> loop (n - 1)
    in
    loop 4
end
```

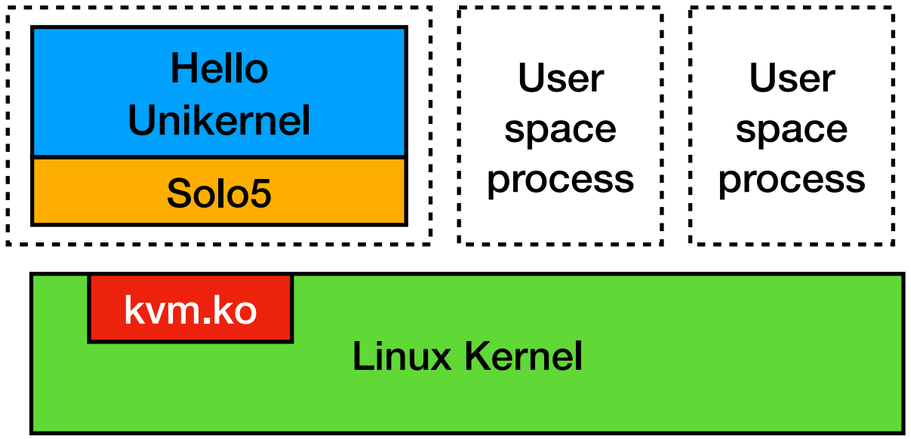
Hello unikernel — Unix backend

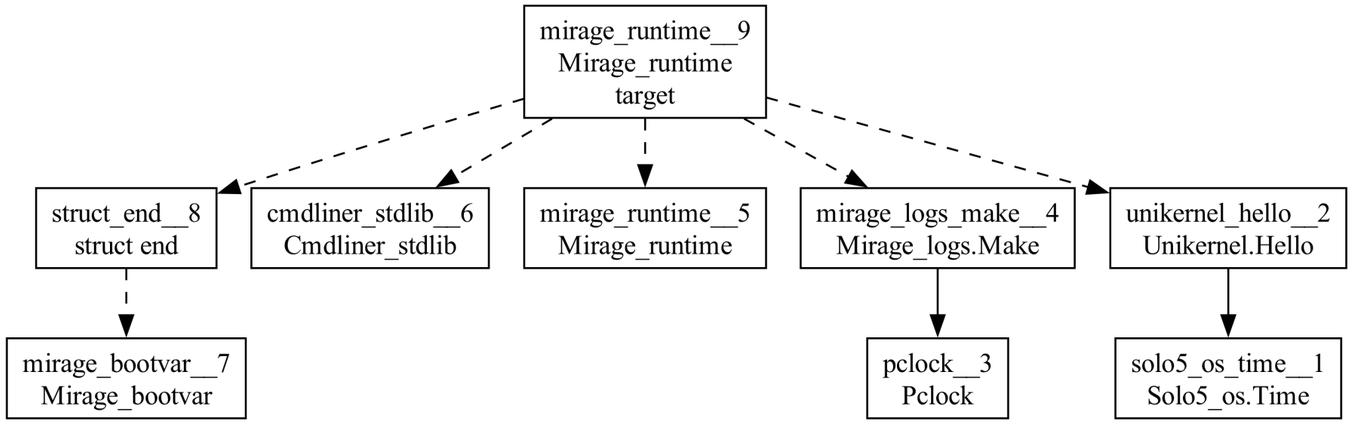
```
$ mirage configure -t unix
$ make
$ ./dist/hello
2024-11-25T17:04:16+05:30: [INFO] [application] hello
2024-11-25T17:04:17+05:30: [INFO] [application] hello
2024-11-25T17:04:18+05:30: [INFO] [application] hello
2024-11-25T17:04:19+05:30: [INFO] [application] hello
```



Hello unikernel — solo5-hvt on kvm

```
$ mirage configure -t hvt
$ make
$ solo5-hvt -- dist/hello.hvt
             |      ___|
  __|  _ \  |  _ \ __ \
\__ \ (   | | (   |  ) |
____/\___/ _|\___/____/
Solo5: Bindings version v0.9.0
Solo5: Memory map: 512 MB addressable:
Solo5:   reserved @ (0x0 - 0xfffff)
Solo5:       text @ (0x100000 - 0x1c4fff)
Solo5:     rodata @ (0x1c5000 - 0x1f5fff)
Solo5:       data @ (0x1f6000 - 0x289fff)
Solo5:       heap >= 0x28a000 < stack < 0x20000000
2024-11-25T11:47:10-00:00: [INFO] [application] hello
2024-11-25T11:47:11-00:00: [INFO] [application] hello
2024-11-25T11:47:12-00:00: [INFO] [application] hello
2024-11-25T11:47:13-00:00: [INFO] [application] hello
Solo5: solo5_exit(0) called
```





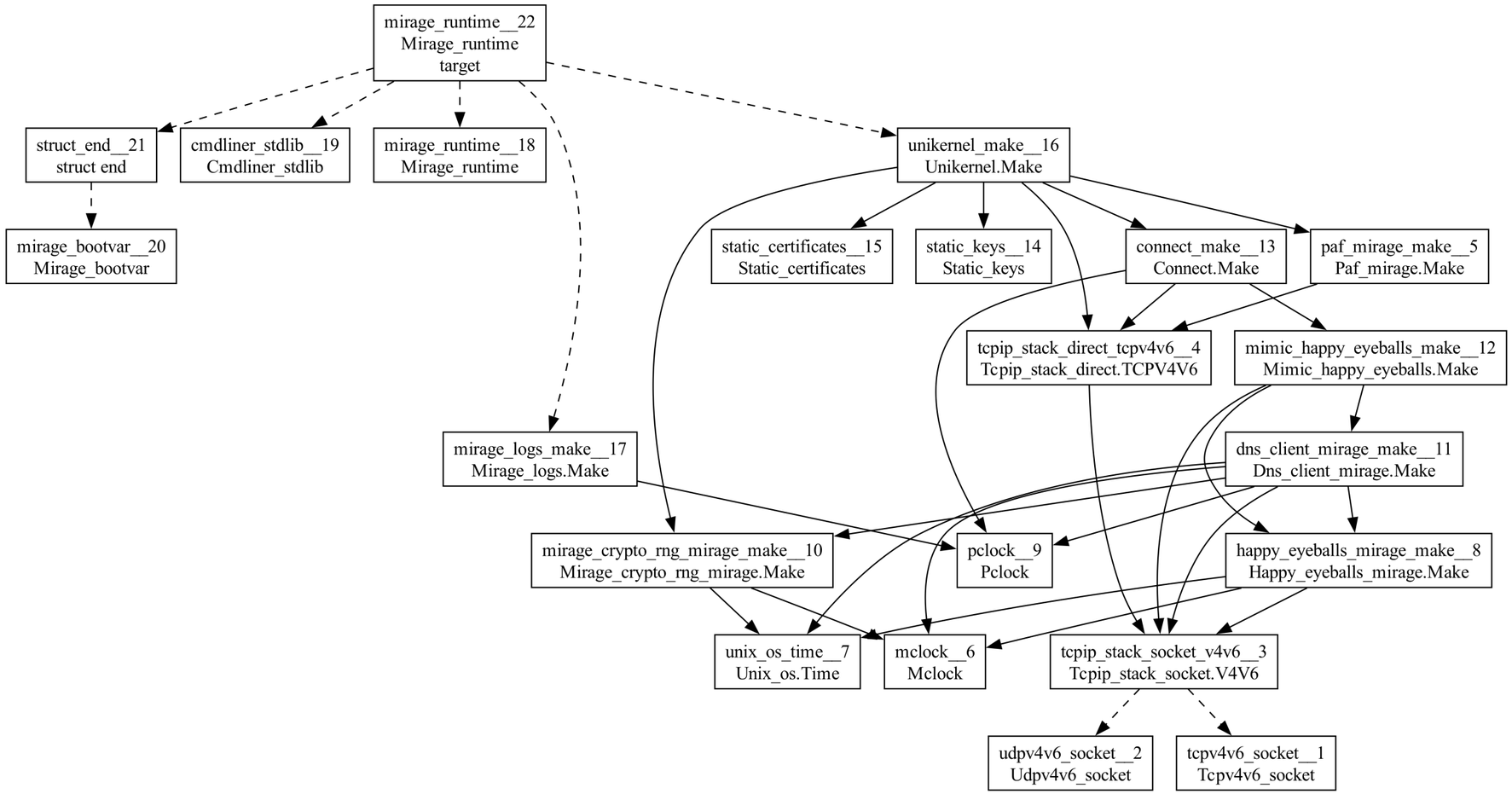
mirage.io website

- A full-fledged https server
- Uses TLS encryption

```
module Make
    (Random : Mirage_crypto_rng_mirage.S)
    (Certificate : Mirage_kv.RO)
    (Key : Mirage_kv.RO)
    (Tcp : Tcpip.Tcp.S with type ipaddr = Ipaddr.t)
    (Connect : Connect.S)
    (HTTP_server : Paf_mirage.S) =
struct
```

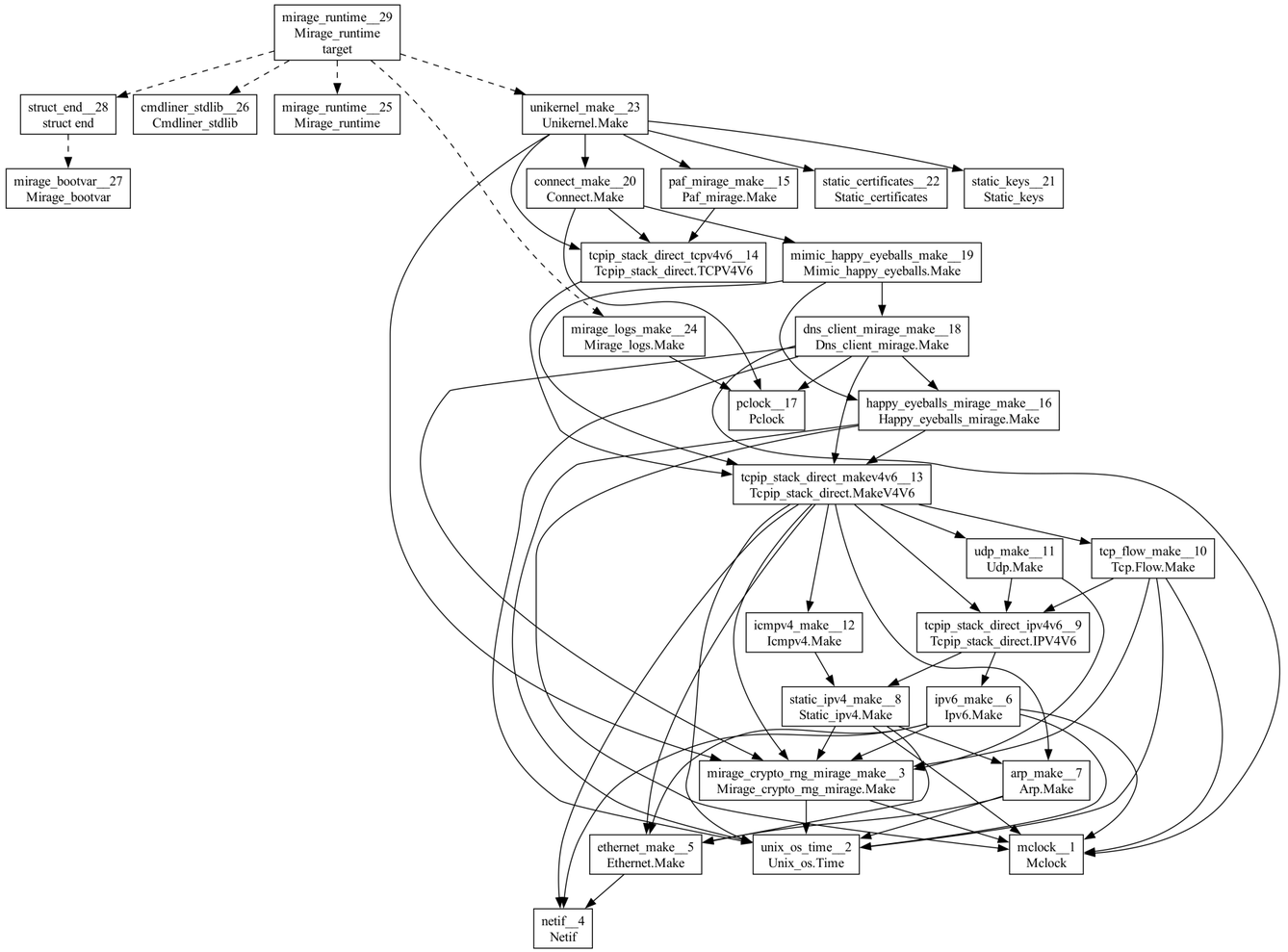
mirage.io website

```
$ mirage configure -t unix --net=host
```



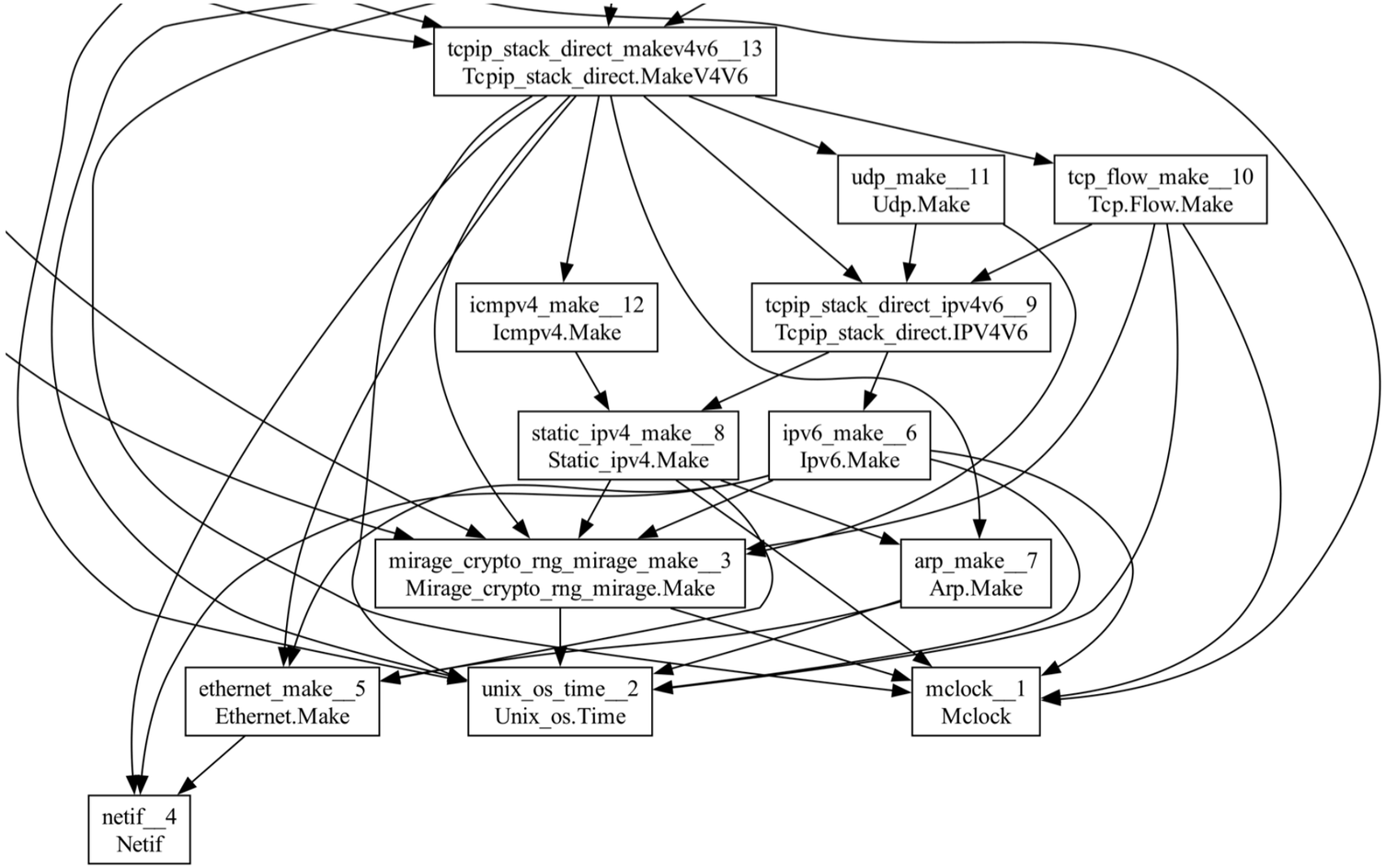
mirage.io website

```
$ mirage configure -t unix --net=direct
```



mirage.io website

```
$ mirage configure -t unix --net=direct
```



MirageOS Compiler

- Remove dead code and inline code across traditionally opaque layer
  - Resulting images usually have a size of **a few MiB**.
  - Our HTTPS web server which runs mirage.io is **only 10 MiB!**

MirageOS Compiler

- Remove dead code and inline code across traditionally opaque layer
  - Resulting images usually have a size of **a few MiB**.
  - Our HTTPS web server which runs mirage.io is **only 10 MiB!**
- Configuration can be partially evaluated at compile-time
  - Extreme specialisation enables a **boot time of a few ms**.

MirageOS Compiler

- Remove dead code and inline code across traditionally opaque layer
  - Resulting images usually have a size of **a few MiB**.
  - Our HTTPS web server which runs mirage.io is **only 10 MiB!**
- Configuration can be partially evaluated at compile-time
  - Extreme specialisation enables a **boot time of a few ms**.
- If something (e.g. networking) is not used, it will not be available at runtime
  - Minimal runtime environments use **a few MiB of RAM**.

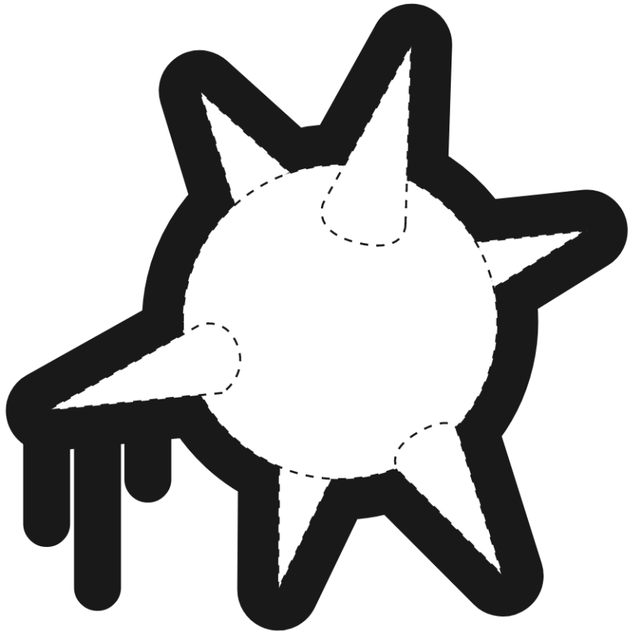
MirageOS Compiler

- Remove dead code and inline code across traditionally opaque layer
  - Resulting images usually have a size of **a few MiB**.
  - Our HTTPS web server which runs mirage.io is **only 10 MiB!**
- Configuration can be partially evaluated at compile-time
  - Extreme specialisation enables a **boot time of a few ms**.
- If something (e.g. networking) is not used, it will not be available at runtime
  - Minimal runtime environments use **a few MiB of RAM**.
- The kernel and user space share the same address space
  - Many runtime checks are removed, so static analysis is critical.

MirageOS Usecases

Bitcoin Piñata

- https://hannes.robur.coop/Posts/Pinata
- 1.1 MB Unikernel, which ran from 2015 to 2018
- Hold the key to 10 bitcoins (peak worth $165k)
  - Now worth ~$1M
- A successful authenticated TLS session reveals the private Bitcoin key
- 500,000 accesses to the Piñata website, more than 150,000 attempts at connecting to the Piñata bounty
- The bitcoins were safe!



Nitrokey NetHSM

- NitroKey is developing NetHSM, a new HSM solution to manage cryptographic keys securely.
- The software implementation should be easy to customise and offer superior security
  - It should also be easily auditable by anyone to eliminate backdoors.
- The NetHSM should meet high-performance requirements, allowing its use in low-power hardware security devices and highly efficient cloud-based solutions.
- They chose to use MirageOS running on the Muen micro-kernel

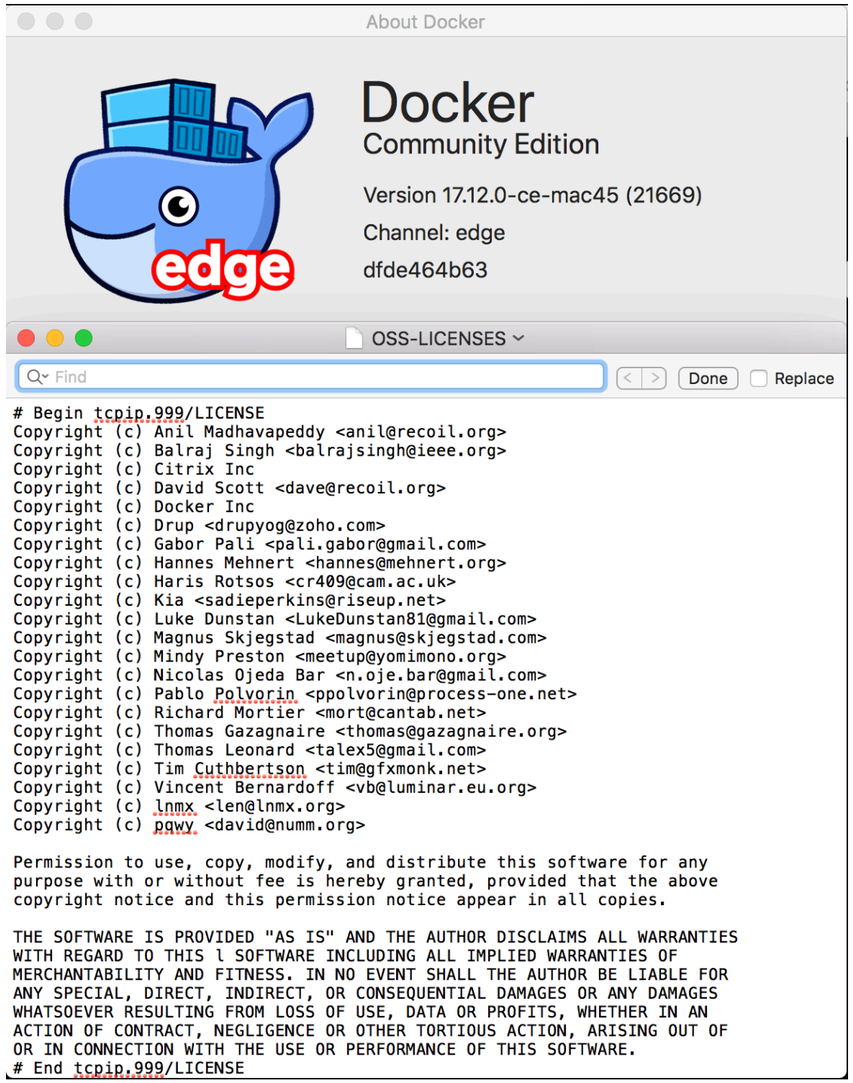
NetHSM - The Trustworthy, Open Hardware Security Module That Just Works

https://www.nitrokey.com/products/nethsm

Docker for Mac

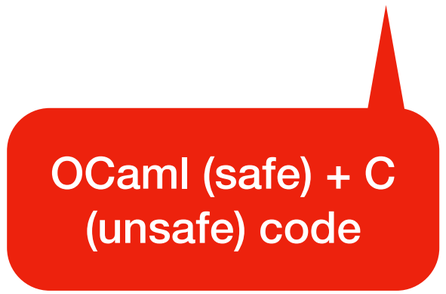
MirageOS libraries used by millions of users

- Normally Docker use Linux namespaces and other Linux features
- On macOS
  - Docker daemon runs in a light Linux VM (using hypervisor.framework)
  - Docker client is a Mac application
- MirageOS libraries are used to translate semantics differences between platforms:
  - **volumes:** FUSE format + fsevent/inotify
  - **network:** Linux ethernet packets to MacOS network syscalls



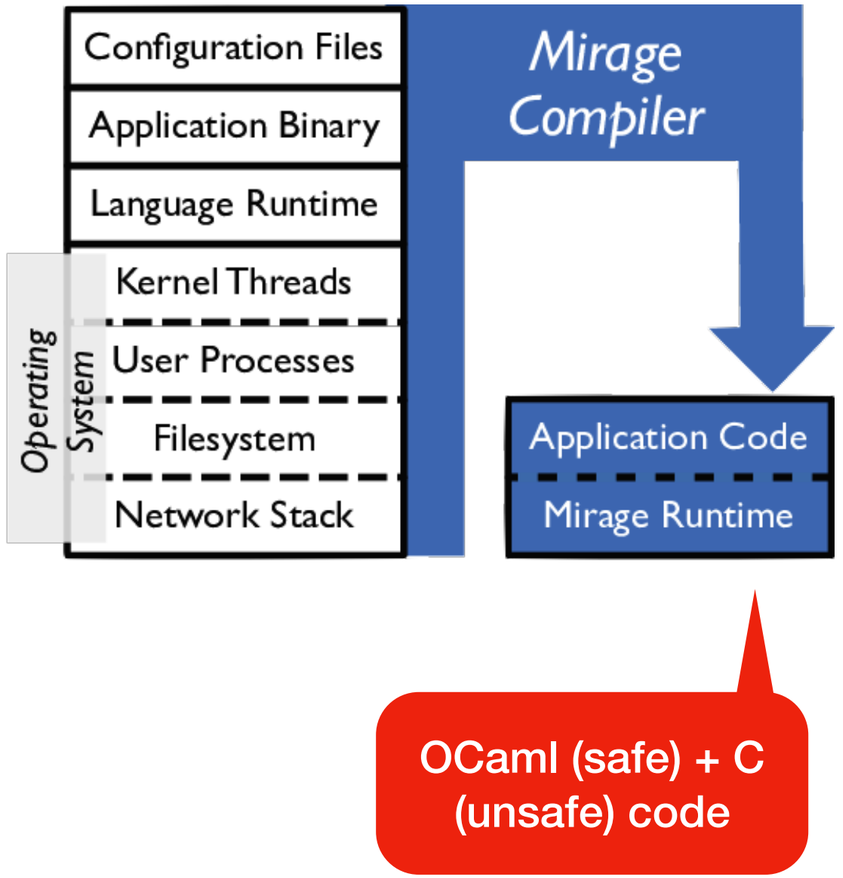
MirageOS Challenges

- Rewrite your applications in OCaml!
- No inter-unikernel isolation
  - No separate kernel vs user space
  - No separation between different bits of the user space (no process abstraction)
- Linking external C libraries
  - Legacy C code is unavoidable — crypto, drivers, sqlite, …
  - may have memory vulnerabilities, may harm Unikernel safety



MirageOS Challenges

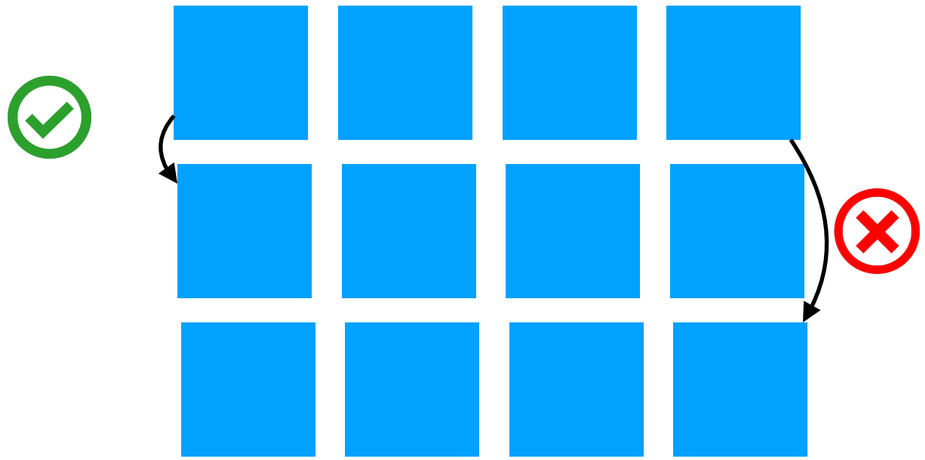
- Rewrite your applications in OCaml!
- No inter-unikernel isolation
  - No separate kernel vs user space
  - No separation between different bits of the user space (no process abstraction)
- Linking external C libraries
  - Legacy C code is unavoidable — crypto, drivers, sqlite, …
  - may have memory vulnerabilities, may harm Unikernel safety



Can we provide fault isolation within Unikernels?

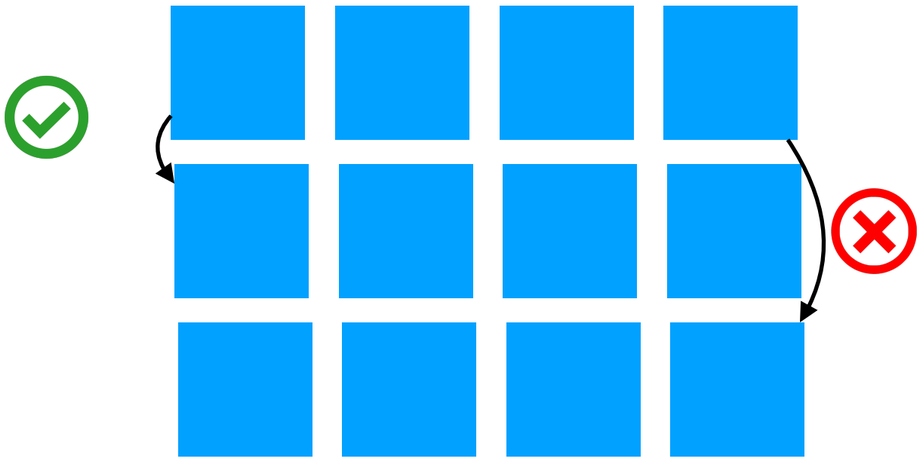
Compartments / SFI — overview

- Compartments offer *intra-process* isolation
  - Functions mapped to compartments
  - Restrict *control flow* and *data access* across security boundaries



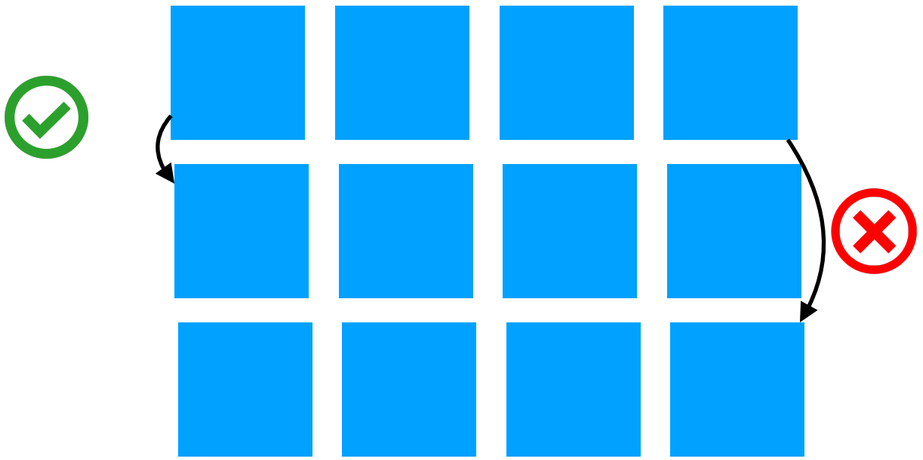
Compartments / SFI — overview

- Compartments offer *intra-process* isolation
  - Functions mapped to compartments
  - Restrict *control flow* and *data access* across security boundaries
- Control flow restricted by
  - Whitelisted PC ranges
  - Shadow stack to prevent ROP attacks



Compartments / SFI — overview

- Compartments offer *intra-process* isolation
  - Functions mapped to compartments
  - Restrict *control flow* and *data access* across security boundaries
- Control flow restricted by
  - Whitelisted PC ranges
  - Shadow stack to prevent ROP attacks
- Data access restricted by
  - VMM tricks (or) fat pointers (or) capabilities (à la CHERI)

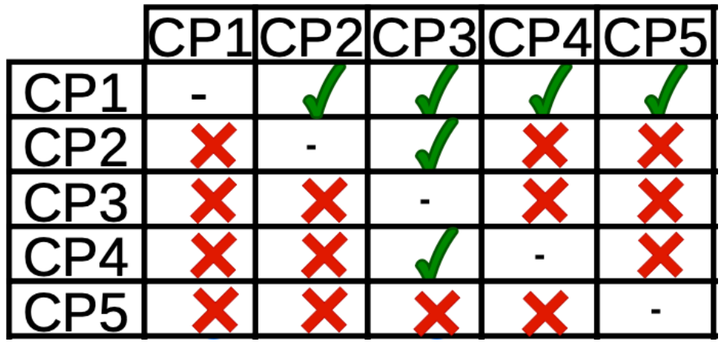


FIDES — Secure compartments

- Security-hardened Shakti RISC-V processor + MirageOS unikernels
  - https://gitlab.com/shaktiproject

FIDES — Secure compartments

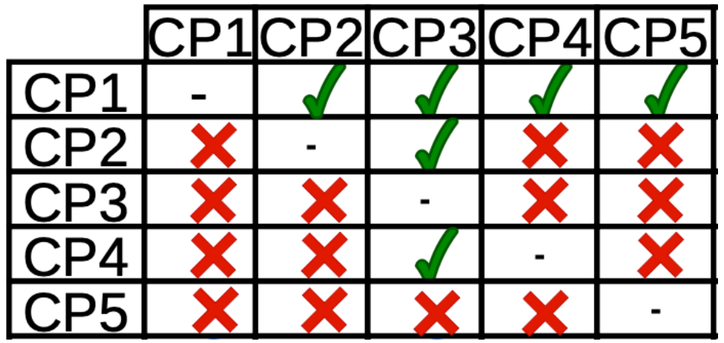
- Security-hardened Shakti RISC-V processor + MirageOS unikernels
  - https://gitlab.com/shaktiproject
- Intra-process compartments
  - Vulnerabilities in C do not affect OCaml



Access Matrix

FIDES — Secure compartments

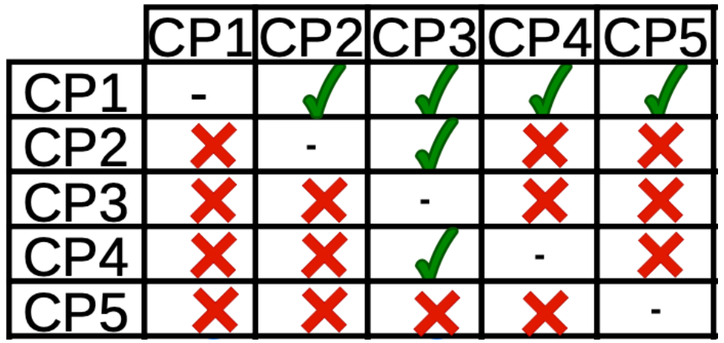
- Security-hardened Shakti RISC-V processor + MirageOS unikernels
  - https://gitlab.com/shaktiproject
- Intra-process compartments
  - Vulnerabilities in C do not affect OCaml
- Compartment access matrix defined at *link time*
  - Run *unmodified* OCaml and C code



Access Matrix

FIDES — Secure compartments

- Security-hardened Shakti RISC-V processor + MirageOS unikernels
  - https://gitlab.com/shaktiproject
- Intra-process compartments
  - Vulnerabilities in C do not affect OCaml
- Compartment access matrix defined at *link time*
  - Run *unmodified* OCaml and C code
- Small extension to hardware and software
  - Two new instructions added to RISC-V ISA: **Val** and **Checkcap**
  - Modification to LLVM and OCaml compiler to emit these instructions



Access Matrix

Threat model

- Source code is untrusted
  - Inline assembly and use of Obj.magic trusted
- All code is compiled with FIDES C and OCaml compiler
  - Compiler instrumentation added by FIDES is correct
  - OCaml runtime is trusted
- Binary executable cannot be tampered with
- Hardware attacks rowhammer, fault attacks, side-channels are out of scope

FIDES Guarantees

- **Control-flow integrity**
  - The control flow in every execution of the program respects the compartment access matrix
- **Memory safety**
  - No memory errors; all references point to valid memory
  - Pointers cannot be forged

FIDES — Challenges and opportunities

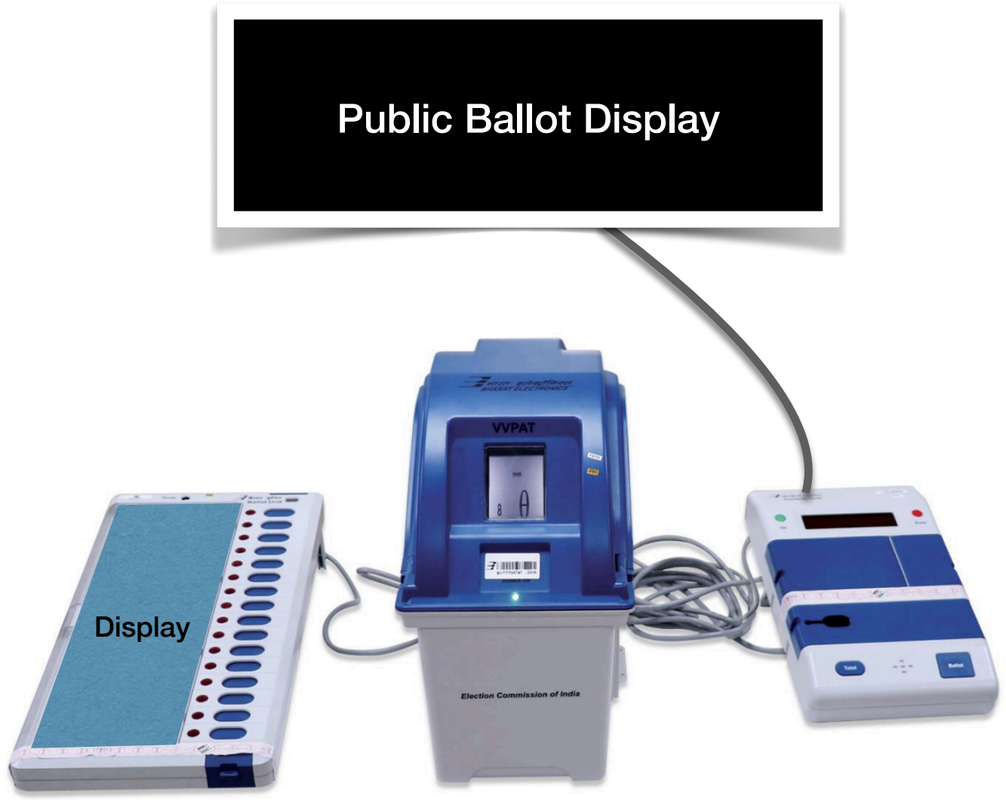
- OCaml offers memory safety
  - Hardware-accelerated fat pointers *only* for C code
    - Fine-grained data compartments
    - No fat pointers for OCaml code
  - Pay attention to FFI boundaries

FIDES — Challenges and opportunities

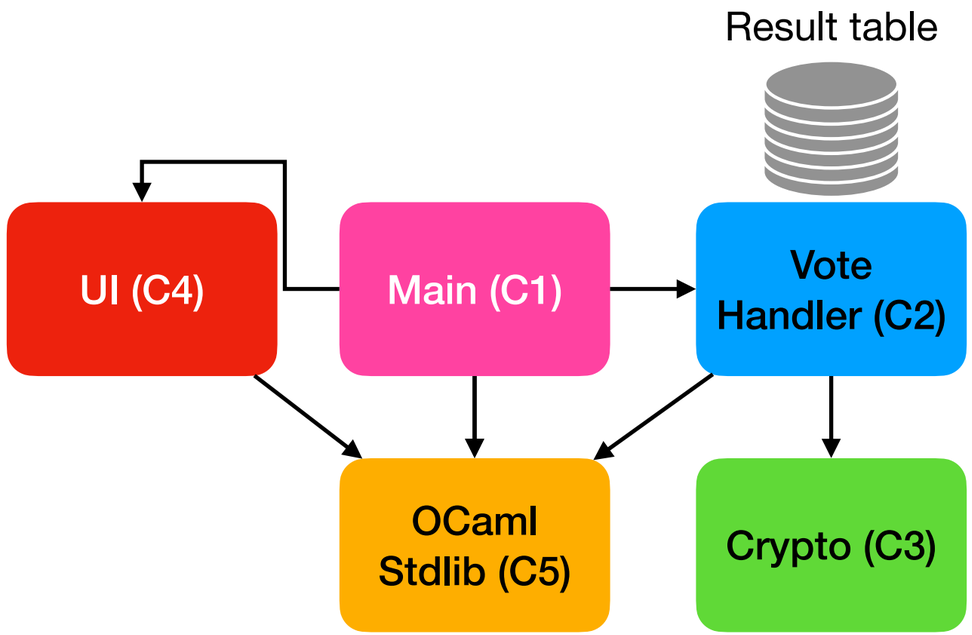
- OCaml offers memory safety
  - Hardware-accelerated fat pointers *only* for C code
    - Fine-grained data compartments
    - No fat pointers for OCaml code
  - Pay attention to FFI boundaries
- **FIDES code compartment must now handle FP features!**
  - Higher-order functions, tail calls, exceptions

Remote Voting Machine (RVM)

- Aim to address voter absenteeism amongst migrant voters
  - 300 million people don’t vote
- Enable migrant voters to be able to vote from a different constituency
- Voting machine is more complex!
  - “Discussion on improving voter participation of domestic migrants using remote voting”, Election Commission of India, 2022

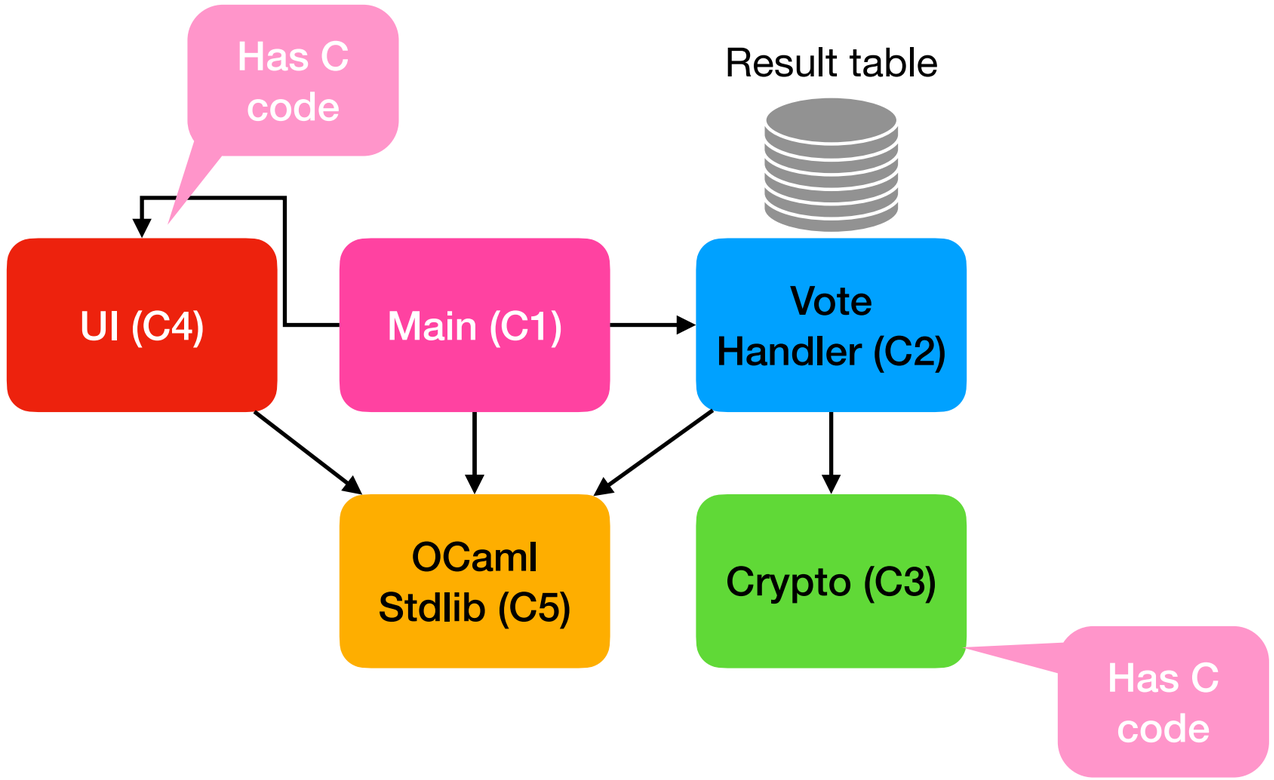


Compartments for RVM



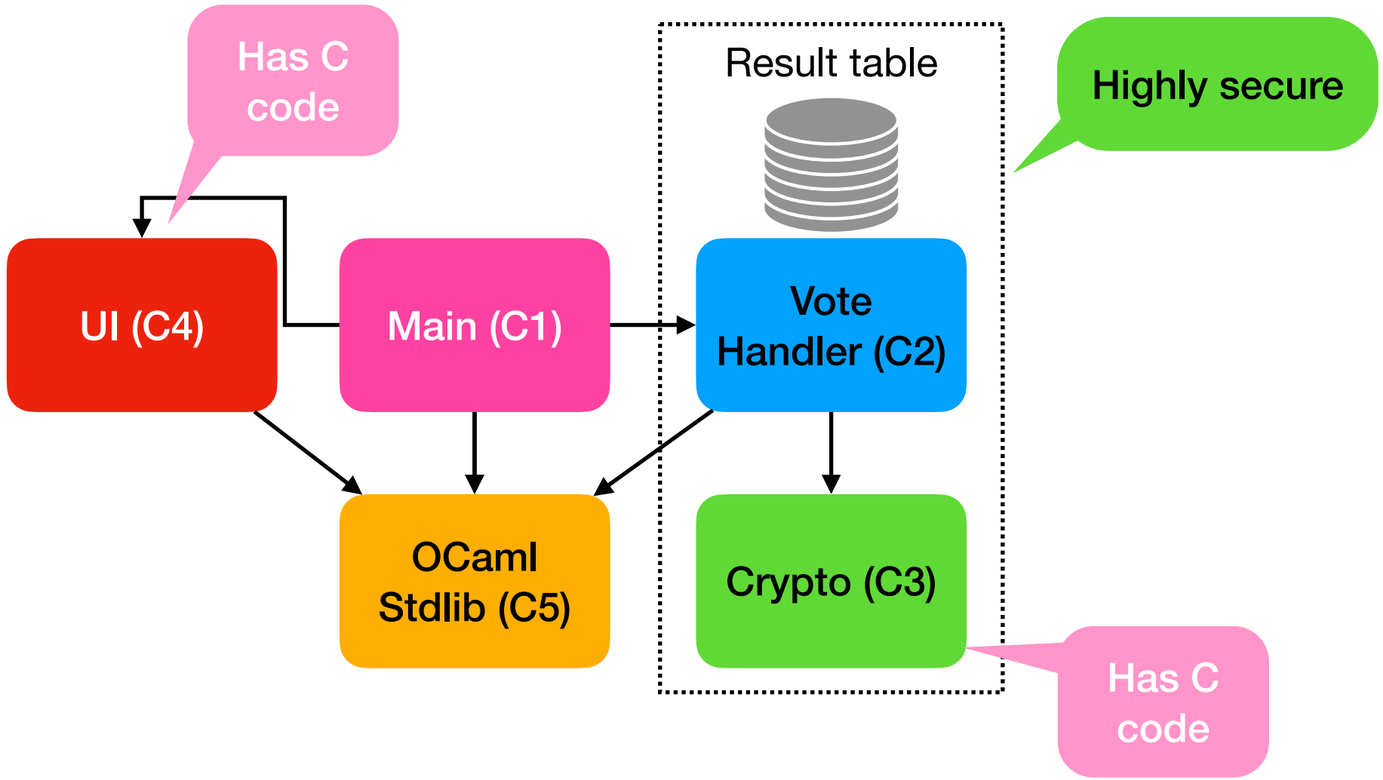
indicates call is allowed

Compartments for RVM



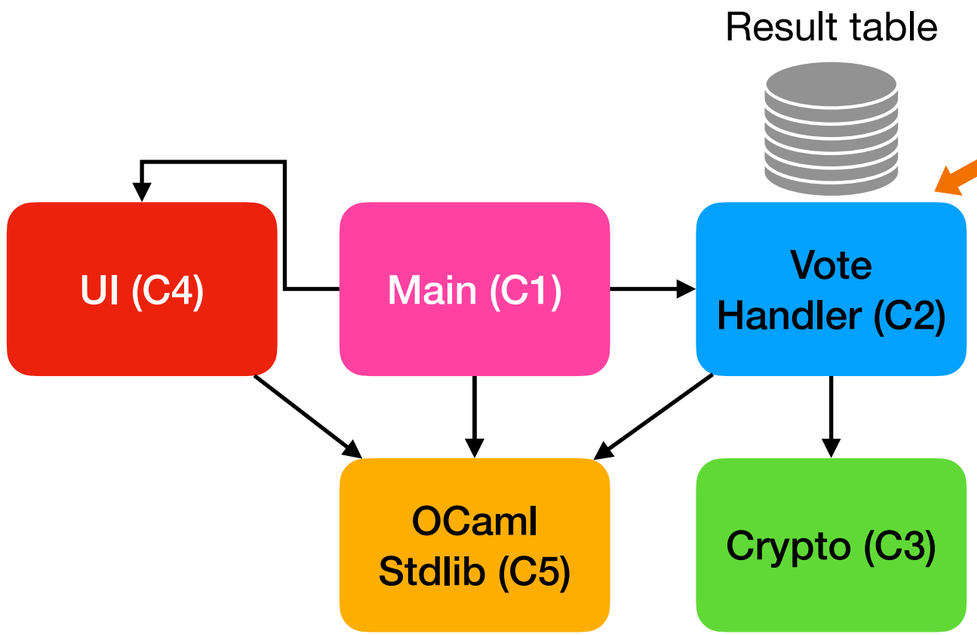
→ indicates call is allowed

Compartments for RVM



indicates call is allowed

Higher-order functions

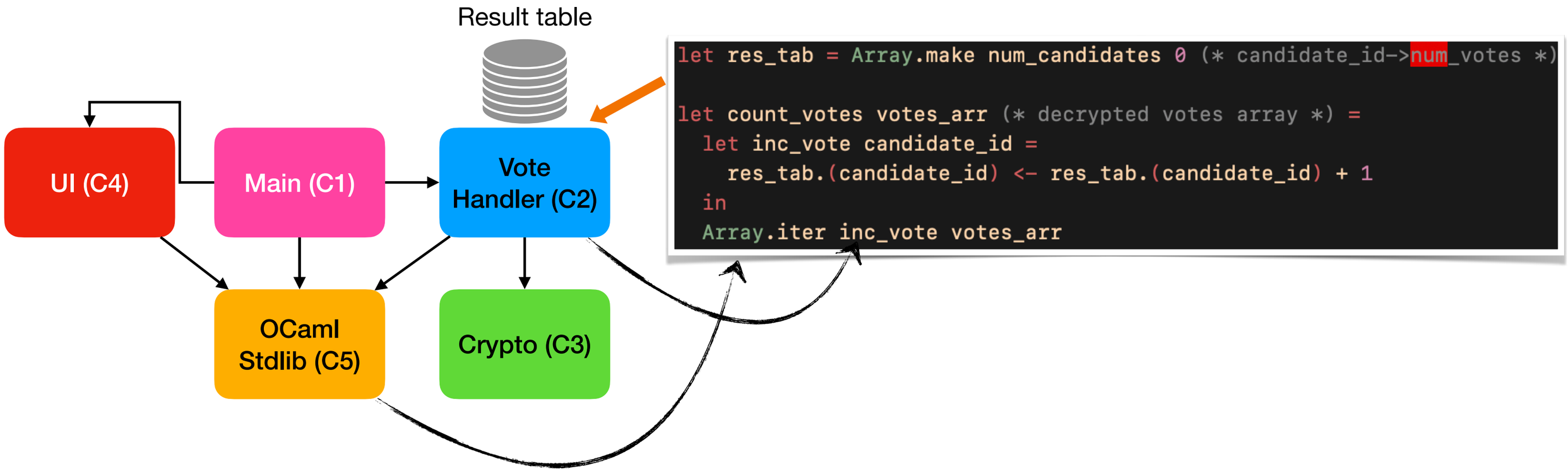



```
let res_tab = Array.make num_candidates 0 (* candidate_id->num_votes *)

let count_votes votes_arr (* decrypted votes array *) =
  let inc_vote candidate_id =
    res_tab.(candidate_id) <- res_tab.(candidate_id) + 1
  in
  Array.iter inc_vote votes_arr
```

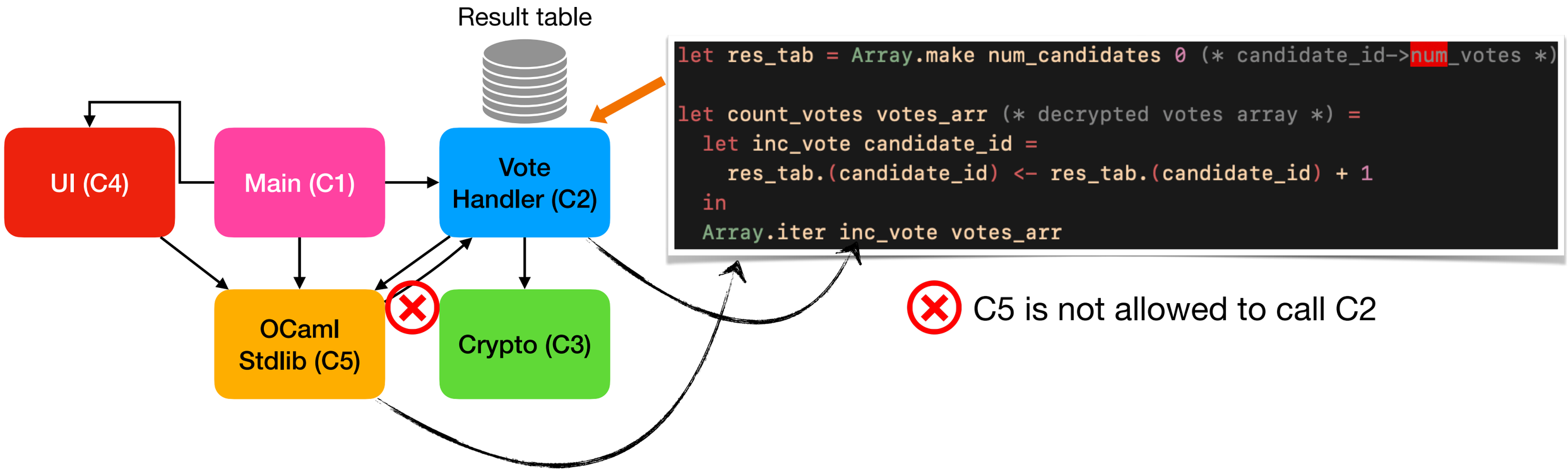
→ indicates call is allowed

Higher-order functions

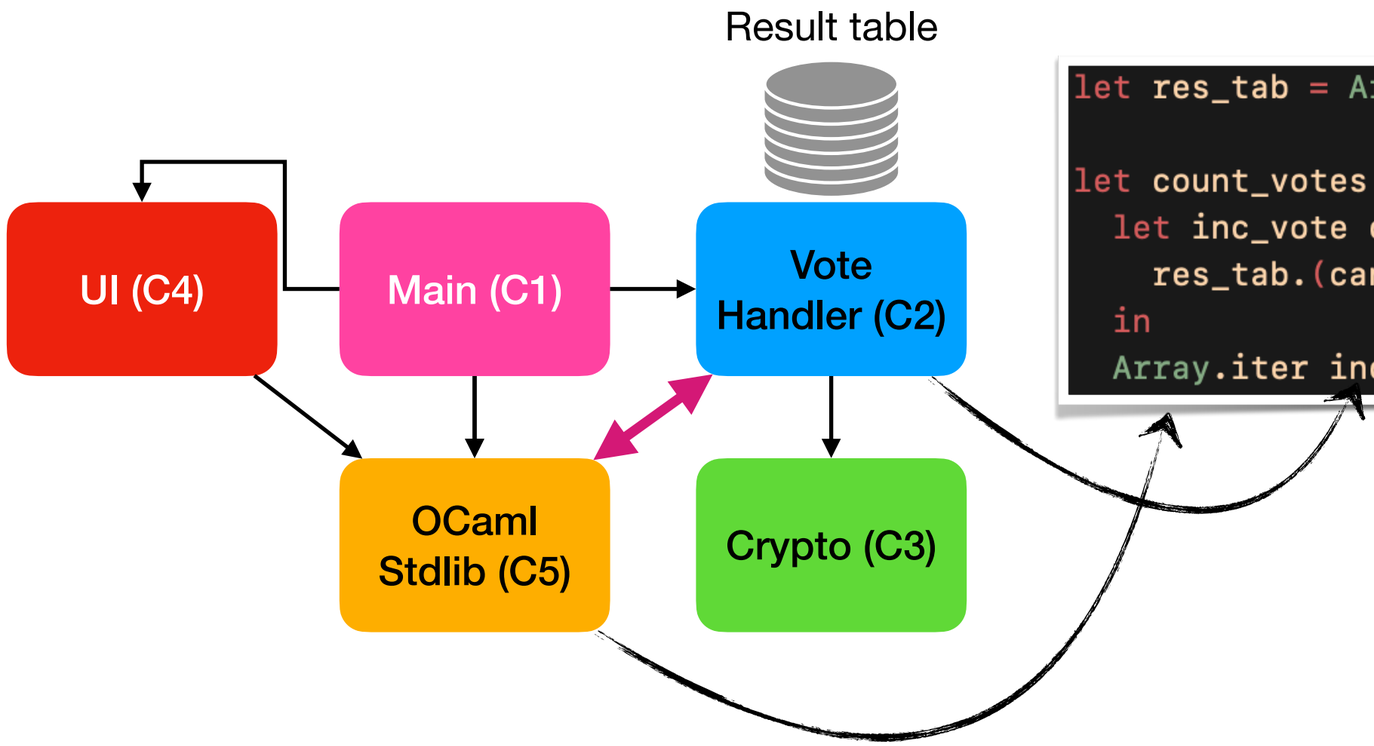


indicates call is allowed

Higher-order functions



Higher-order functions — Idea 1

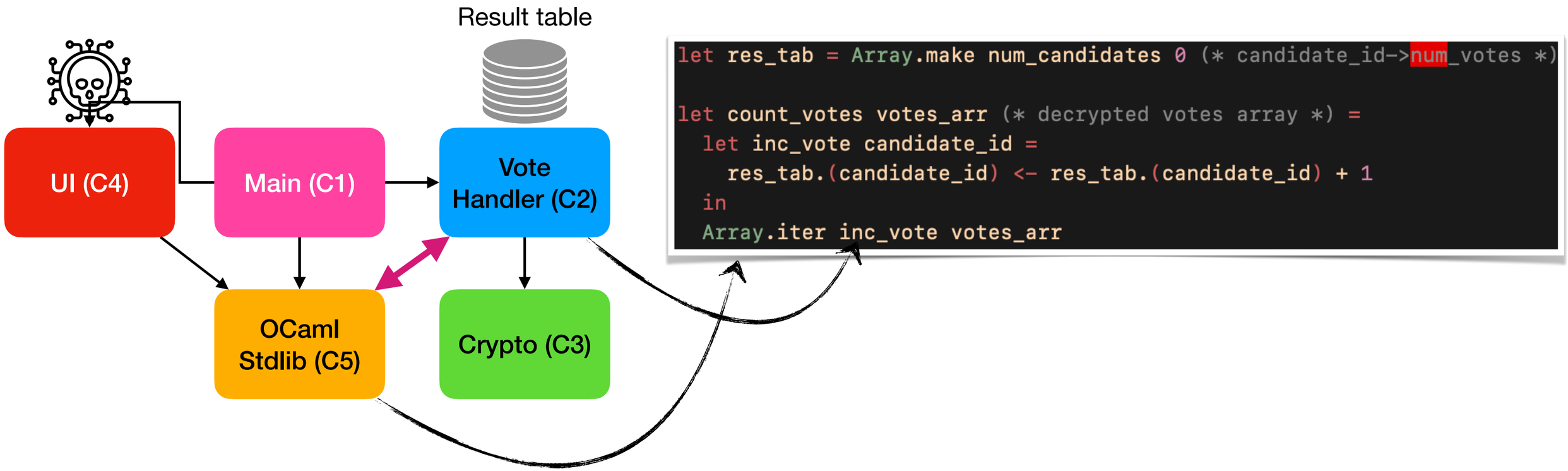



```
let res_tab = Array.make num_candidates 0 (* candidate_id->num_votes *)

let count_votes votes_arr (* decrypted votes array *) =
  let inc_vote candidate_id =
    res_tab.(candidate_id) <- res_tab.(candidate_id) + 1
  in
  Array.iter inc_vote votes_arr
```

→ indicates call is allowed

Higher-order functions — Idea 1

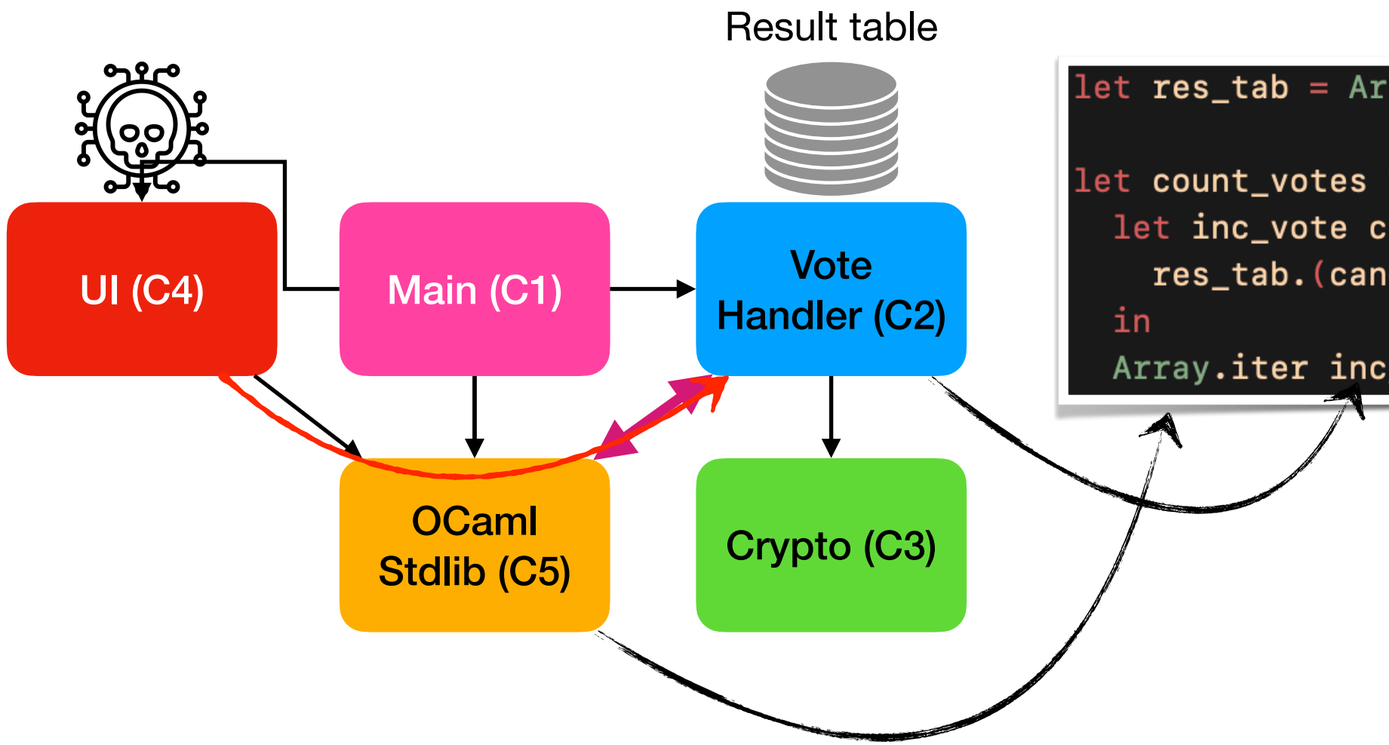



```
let res_tab = Array.make num_candidates 0 (* candidate_id->num_votes *)

let count_votes votes_arr (* decrypted votes array *) =
  let inc_vote candidate_id =
    res_tab.(candidate_id) <- res_tab.(candidate_id) + 1
  in
  Array.iter inc_vote votes_arr
```

→ indicates call is allowed

Higher-order functions — Idea 1



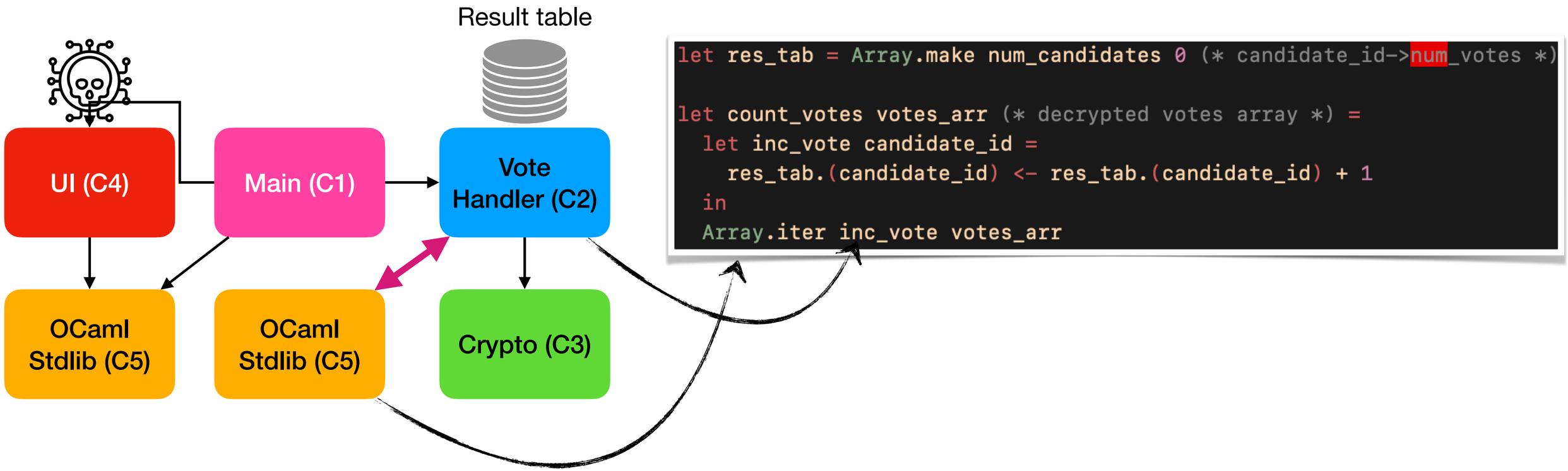

```
let res_tab = Array.make num_candidates 0 (* candidate_id->num_votes *)

let count_votes votes_arr (* decrypted votes array *) =
  let inc_vote candidate_id =
    res_tab.(candidate_id) <- res_tab.(candidate_id) + 1
  in
  Array.iter inc_vote votes_arr
```

“Confused Deputy” attack

→ indicates call is allowed

Higher-order functions — Idea 2

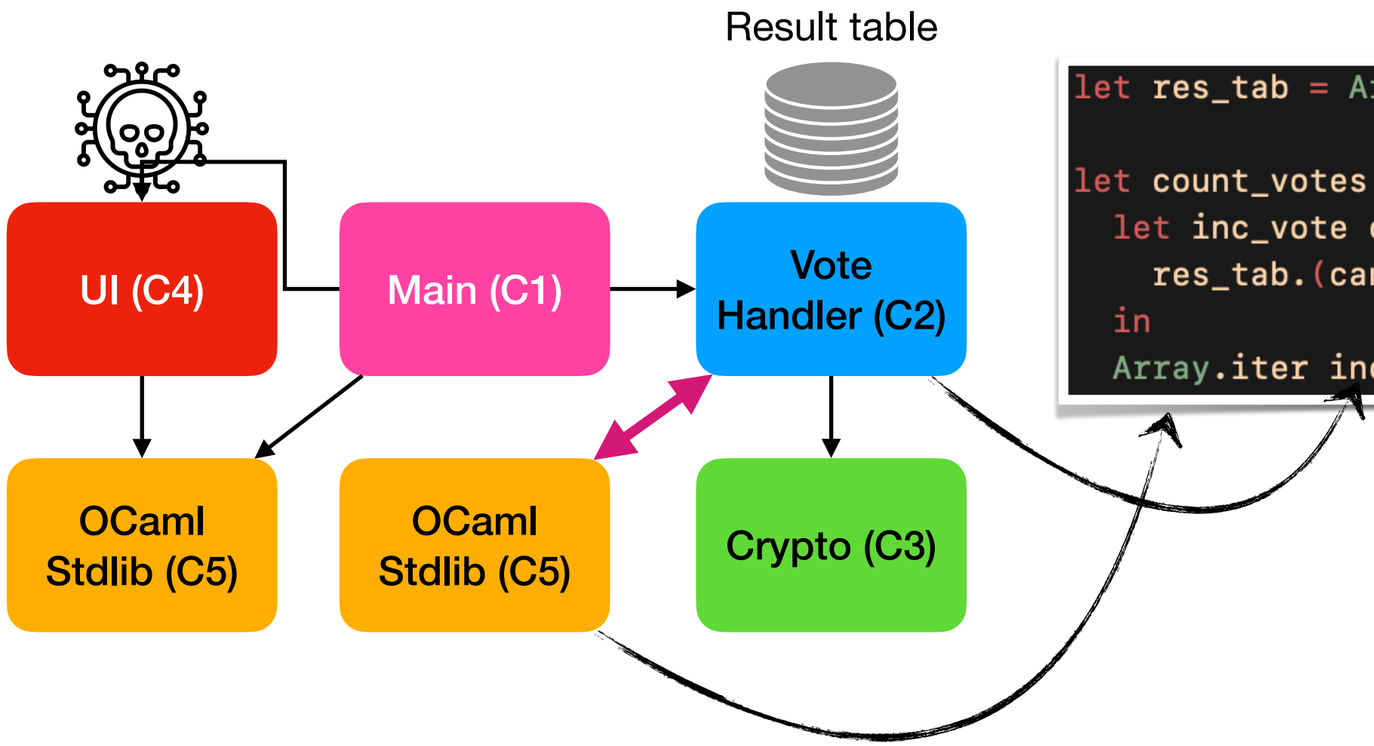



```
let res_tab = Array.make num_candidates 0 (* candidate_id->num_votes *)

let count_votes votes_arr (* decrypted votes array *) =
  let inc_vote candidate_id =
    res_tab.(candidate_id) <- res_tab.(candidate_id) + 1
  in
  Array.iter inc_vote votes_arr
```

indicates call is allowed

Higher-order functions — Idea 2



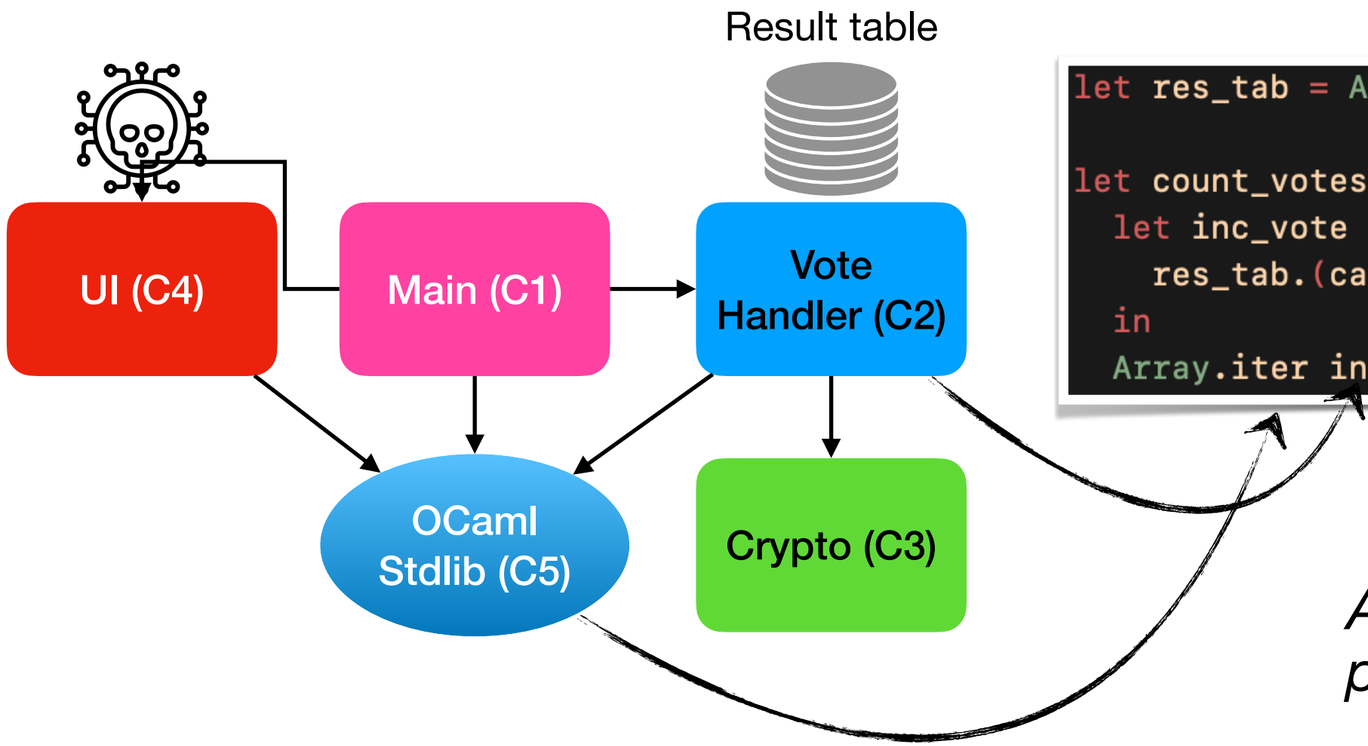

```
let res_tab = Array.make num_candidates 0 (* candidate_id->num_votes *)

let count_votes votes_arr (* decrypted votes array *) =
  let inc_vote candidate_id =
    res_tab.(candidate_id) <- res_tab.(candidate_id) + 1
  in
  Array.iter inc_vote votes_arr
```

- ⊗ Limited compartment resource
- ⊗ Shared state?

→ indicates call is allowed

Fluid compartments




```
let res_tab = Array.make num_candidates 0 (* candidate_id->num_votes *)

let count_votes votes_arr (* decrypted votes array *) =
  let inc_vote candidate_id =
    res_tab.(candidate_id) <- res_tab.(candidate_id) + 1
  in
  Array.iter inc_vote votes_arr
```

A fluid compartment inherits the policies of the caller

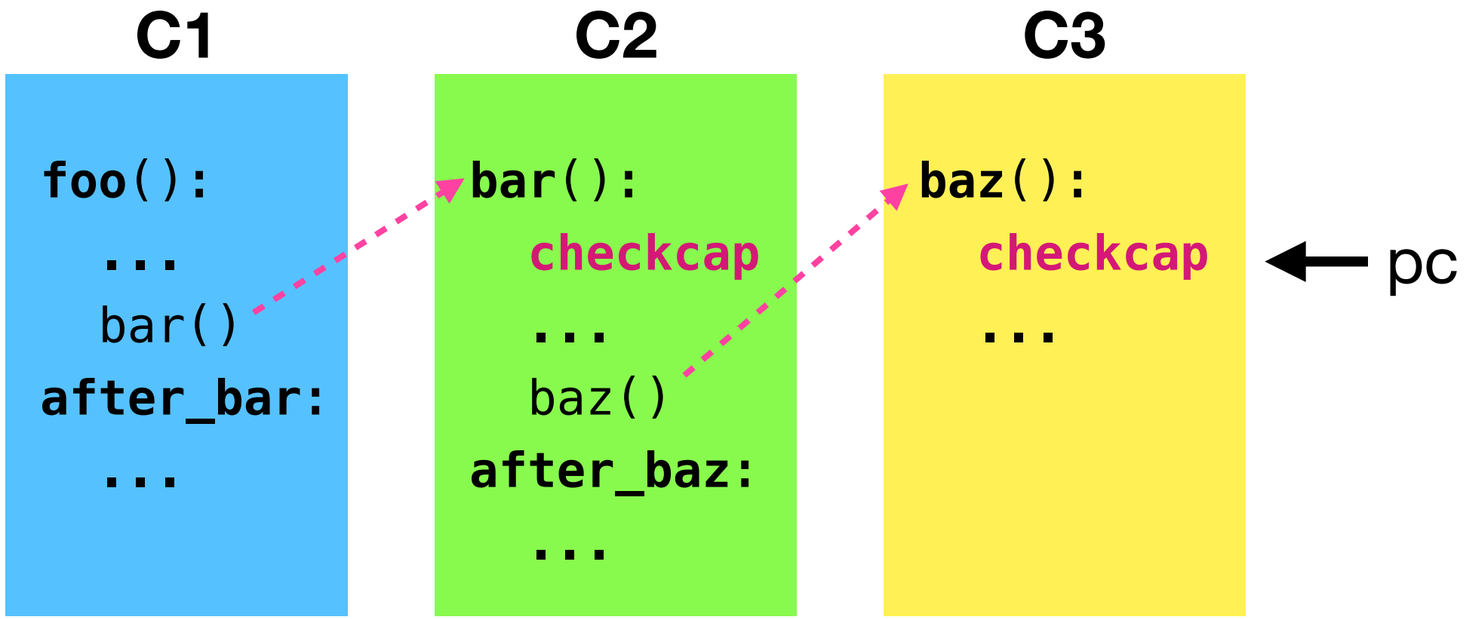
indicates call is allowed

Shadow stack

- Stores the return addresses for inter-compartment calls
- ***Inaccessible from user-code***
  - Maintained and validated by hardware

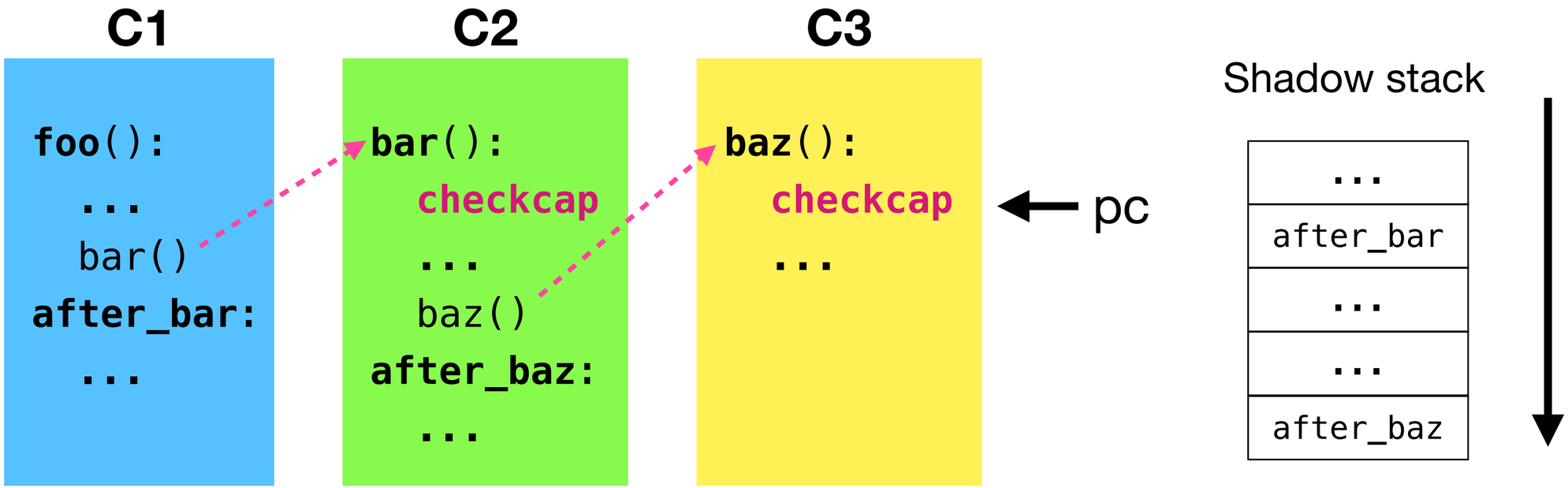
Shadow stack

- Stores the return addresses for inter-compartment calls
- ***Inaccessible from user-code***
  - Maintained and validated by hardware



Shadow stack

- Stores the return addresses for inter-compartment calls
- ***Inaccessible from user-code***
  - Maintained and validated by hardware



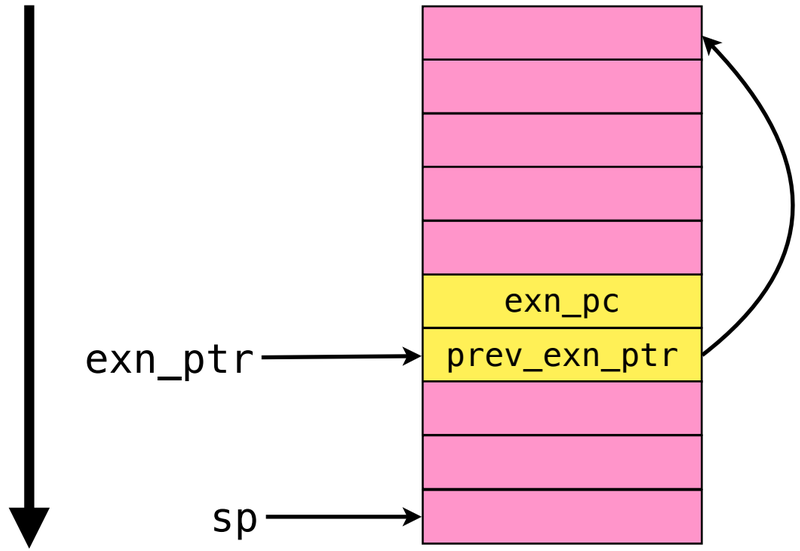
Non-call-return control flow

- Typical compartment schemes handle only call-return sequence

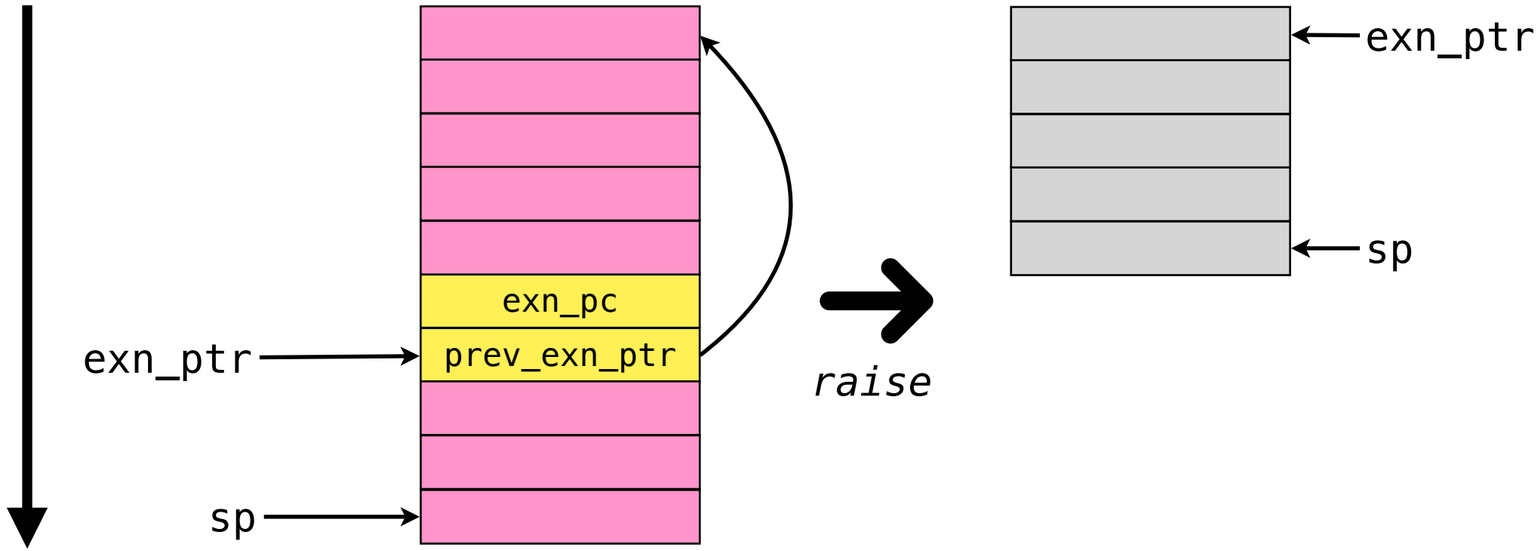
Non-call-return control flow

- Typical compartment schemes handle only call-return sequence
- OCaml has several non-call-return control-flow operations
  - *Tail calls, exceptions, effect handlers!*
  - Need to manage the shadow stack carefully

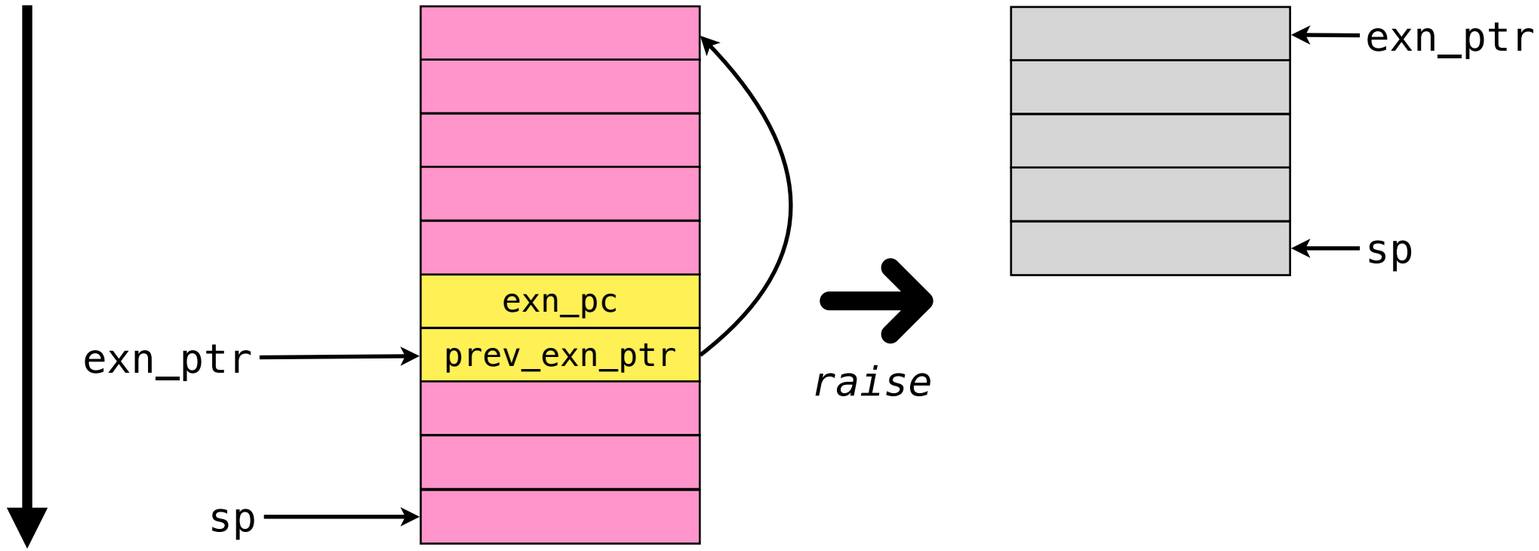
Exceptions and shadow stacks



Exceptions and shadow stacks

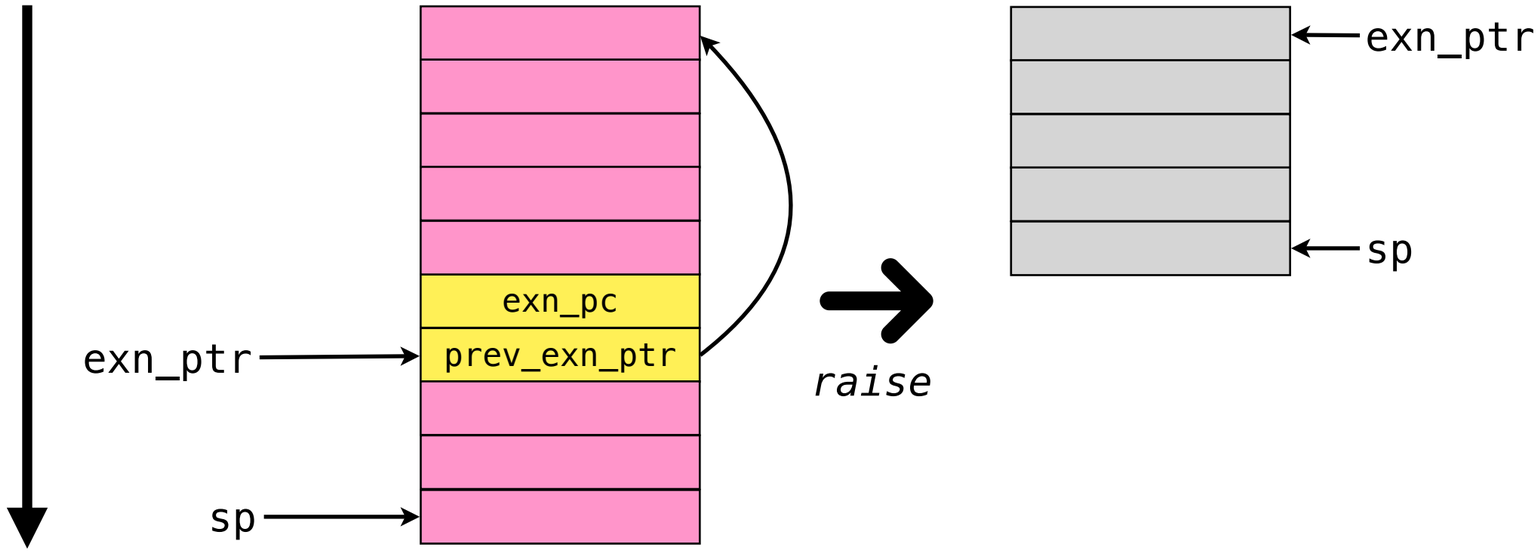


Exceptions and shadow stacks



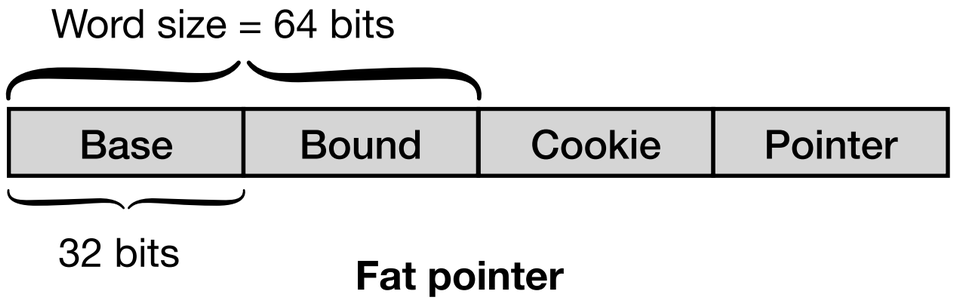
- Exceptions may be thrown across compartments
  - Need to unwind shadow stack appropriately
  - **Challenge:** Detect when intra-compartment exceptions are raised

Exceptions and shadow stacks



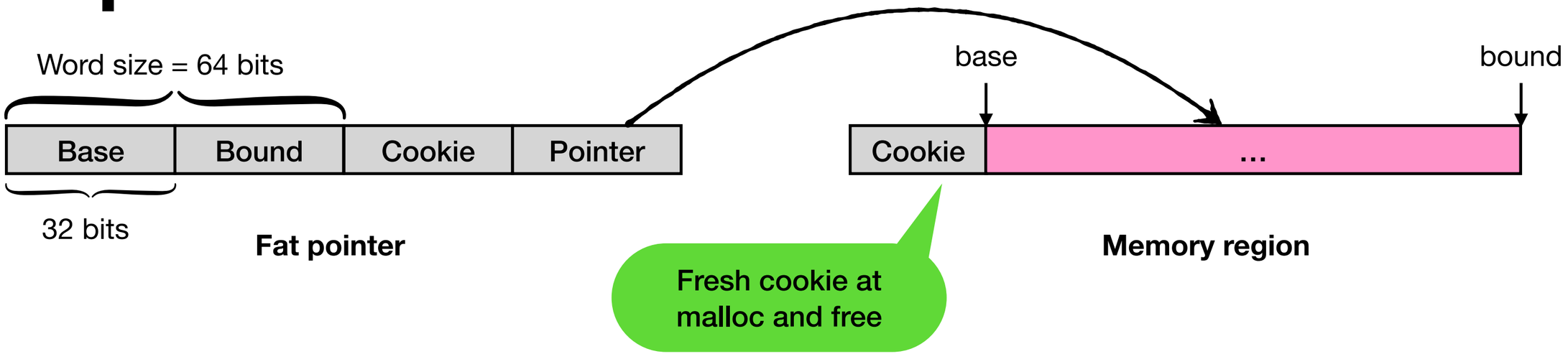
- Exceptions may be thrown across compartments
  - Need to unwind shadow stack appropriately
  - **Challenge:** Detect when intra-compartment exceptions are raised
- **Solution:** Security monitor (SM) updates last `exn_pc` to a special routine

Fat pointers

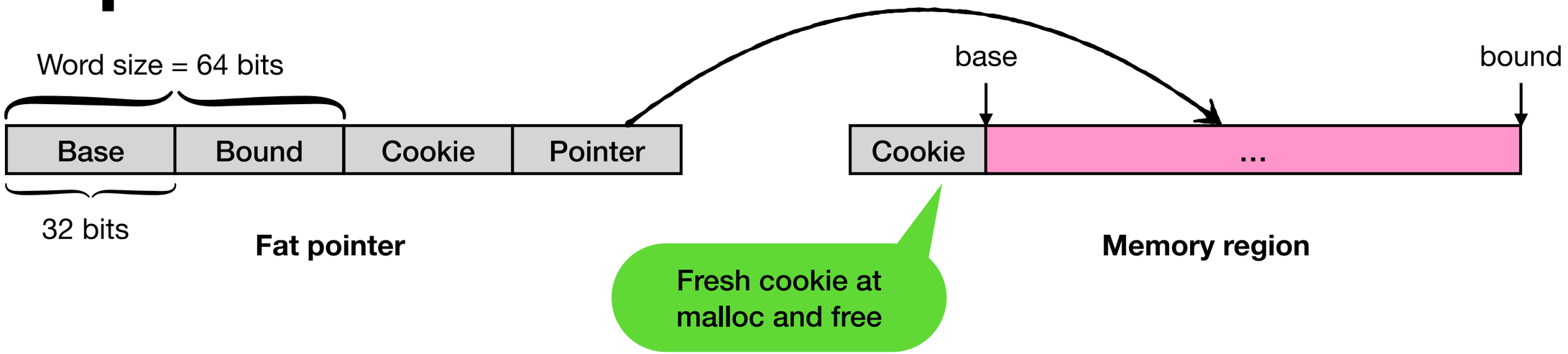


Fat pointer

Fat pointers

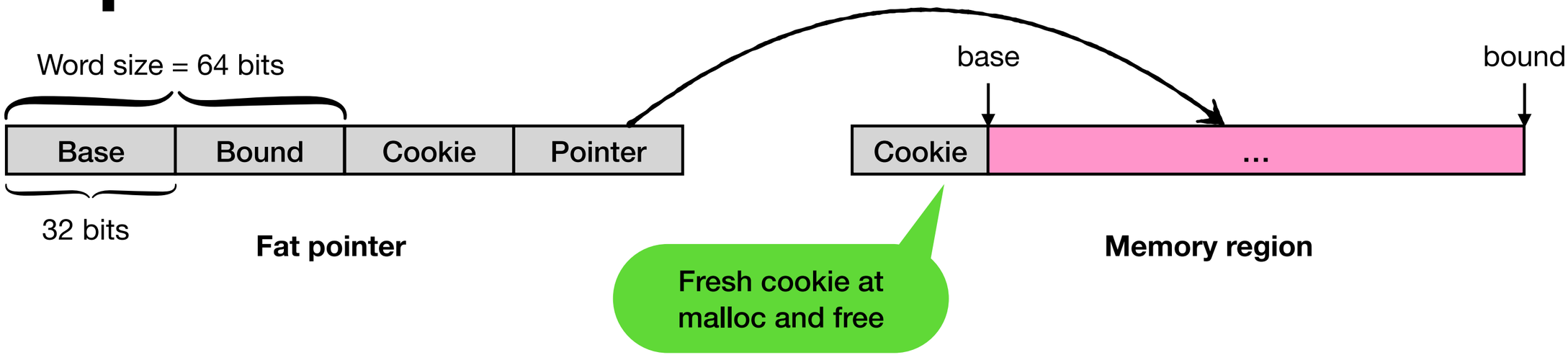


Fat pointers



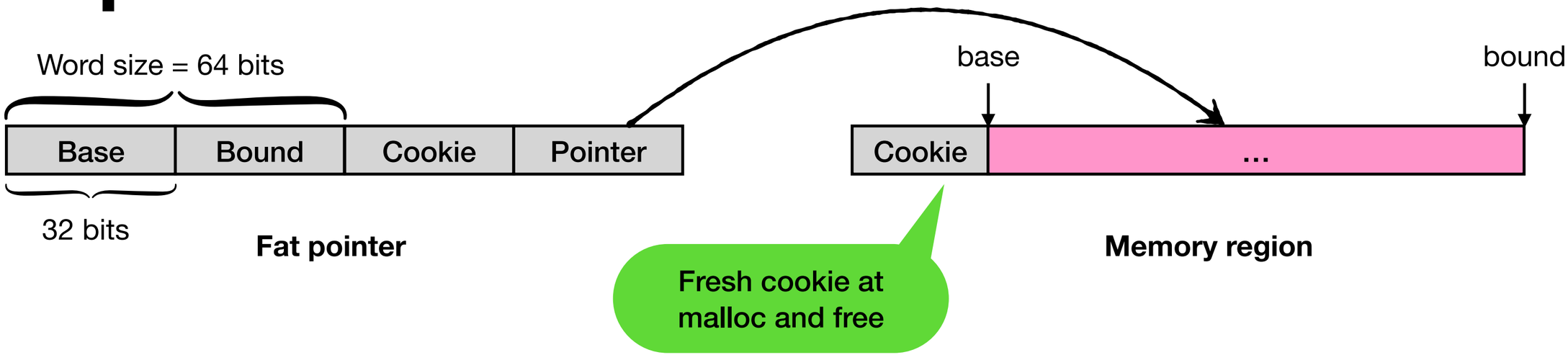
- Fat pointers into the stack have frame scope
  - Each frame has a cookie *freshened* at call and return

Fat pointers



- Fat pointers into the stack have frame scope
  - Each frame has a cookie *freshened* at call and return
- **val** instruction validates fat pointer before access

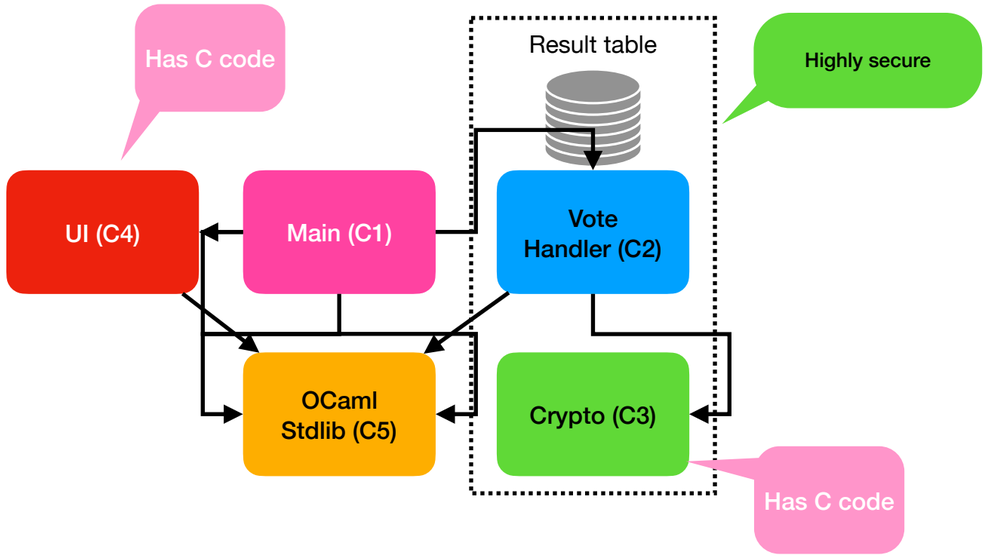
Fat pointers



- Fat pointers into the stack have frame scope
  - Each frame has a cookie *freshened* at call and return
- **val** instruction validates fat pointer before access
- OCaml does not use fat pointers
  - At FFI, use OCaml object header info to create fat pointer
  - Use a special cookie that skips temporal validation

Evaluation

- Compiler changes
  - ~300 lines for OCaml, ~2300 lines for LLVM
- Protyped on Xilinx Artix-7 AC701 FPGA
  - 38.2K LUTs (+6.1% over base)
  - 17.4K registers (+6.0% over base)
- Performance on voting application
  - 4% increase in code size
  - 23% increase in instruction cycle count



Limitations

- **Features:** Effect handlers, parallelism

Limitations

- **Features:** Effect handlers, parallelism
- OCaml runtime is trusted
  - **WIP:** Verified garbage collector for OCaml

Limitations

- **Features:** Effect handlers, parallelism
- OCaml runtime is trusted
  - **WIP:** Verified garbage collector for OCaml
- Data compartments are too weak
  - Objects shared across compartments remain accessible forever
  - Revocation through ownership and borrowing à la Rust
    - modal types in OCaml

Limitations

- **Features:** Effect handlers, parallelism
- OCaml runtime is trusted
  - **WIP:** Verified garbage collector for OCaml
- Data compartments are too weak
  - Objects shared across compartments remain accessible forever
  - Revocation through ownership and borrowing à la Rust
    - modal types in OCaml
- Hardware is exotic
  - Arm MTE for fat pointers in C?

Security — A multi-dimensional challenge