Project A9N

Empowering the Future: A Modern Microkernel-Based Architecture Built for Speed, Flexibility, and Security

Rekka IGUMI / horizon; 2025-02-16

Table of Contents

01

Introduction

Dive into Microkernel World

About Me

Rekka IGUMI

@horizon2k38

- 19 years old
- Low Level,
  Software Architecture
- Microkernel Lover


entries count [007F]
entries count [0080]
entries count [0081]
entries count [0082]
entries count [0083]
entries count [0084]
08:12:28 [469/724]
kernel: A9N v0.2.1
architecture: x86_64
creator: horizon2k38
project: mitoujr
about: A9N is a kernel built on HAL <Hardware Abstraction Layer> and microkernel.
It combines high portability, stability, and scalability.
[0000000000][KERNEL][-] start A9N kernel ...
[0000000001][KERNEL][-] kernel entry : 0xffff800000209ae0
[0000000002][KERNEL][-] boot info address : 0xffff8000bff176d0
[0000000003][KERNEL][-] init local variables ...
[0000000004][KERNEL][-] init architecture ...
[0000000005][KERNEL][-] - init HAL
[0000000006][KERNEL][-] - init io
[0000000007][KERNEL][-] - init serial
[0000000008][KERNEL][-] - init logger
[0000000009][HAL][-] call global ctors ...
[0000000010][HAL][-] arch_info [0] : 0x00000000bfb7e014
[0000000011][HAL][-] arch_info [1] : 0x00000000deadbeaf
[0000000012][HAL][-] arch_info [2] : 0x00000000deadbeaf
[0000000013][HAL][-] arch_info [3] : 0x00000000deadbeaf
[0000000014][HAL][-] arch_info [4] : 0x00000000deadbeaf
[0000000015][HAL][-] arch_info [5] : 0x00000000deadbeaf
[0000000016][HAL][-] arch_info [6] : 0x00000000deadbeaf
[0000000017][HAL][-] arch_info [7] : 0x00000000deadbeaf
[0000000018][HAL][-] configure cpu features ...
[0000000019][HAL][000] configure cpu features ...
[0000000020][HAL][000] configure cpu core ...
[0000000021][HAL][000] init Local APIC
[0000000022][HAL][000] APIC base address : 0x00000000fee00000
[0000000023][HAL][000] init 8259 PIC
[0000000024][HAL][000] init ACPI
[0000000025][HAL][000] signature : RSD PTR
[0000000026][HAL][000] checksum : 150
[0000000027][HAL][000] oem_id : BOCHS
[0000000028][HAL][000] revision : 2
[0000000029][HAL][000] rsdt_address : 0xbfb7d074
[0000000030][HAL][000] length : 0x00000024
[0000000031][HAL][000] xsdt_address : 0x00000000bfb7d0e8
[0000000032][HAL][000] extended_checksum : 174
[0000000033][HAL][000] signature : XSDT
[0000000034][HAL][000] length : 0x0000004c
[0000000035][HAL][000] revision : 1
[0000000036][HAL][000] checksum : 209
[0000000037][HAL][000] oem_id : BOCHS
[0000000038][HAL][000] oem_table_id : BXPC
[0000000039][HAL][000] oem_revision : 0x00000001
[0000000040][HAL][000] creator_id : 0x20202020
[0000000041][HAL][000] creator_revision : 0x01000013
[0000000042][HAL][000] signature : FACP
[0000000043][HAL][000] length : 0x00000074
[0000000044][HAL][000] revision : 1
[0000000045][HAL][000] checksum : 203
[0000000046][HAL][000] oem_id : BOCHS
[0000000047][HAL][000] oem_table_id : BXPC
[0000000048][HAL][000] oem_revision : 0x00000001
[0000000049][HAL][000] creator_id : 0x43505842
[0000000050][HAL][000] creator_revision : 0x00000001
[0000000051][HAL][000] configure FADT address
[0000000052][HAL][000] signature : APIC
[0000000053][HAL][000] length : 0x00000078
"horizon-mac-14.local" 08:12 16-Feb-25

A9N + NUN + KOITO

Secure Operating System With a Capability-Based Microkernel

HAL

for Hardware-Independent Kernel

3rd-Gen Microkernel

Ultra-Fast IPC + Capability + Virtualization

OS Framework

Easier System Construction

User-Level Servers

Integration of Existing System Resources
for Fault-Tolerant System

Too Many Technical Terms …

02

Background

What is Difference?

What is Kernel ?

Heart of Operating System

Application

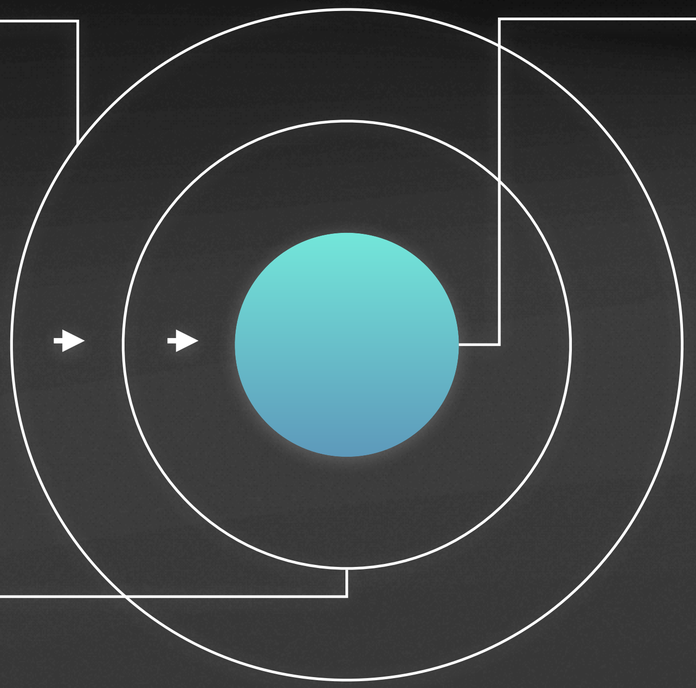
User-Level Software

- Use OS-Level APIs

Operating System

"System" Software

- Use Kernel-Level APIs



Kernel

Provides the most primitive API

- Closest to Hardware
- High Privilege-Level
- Stability is Indispensable

Typical Kernel / OS

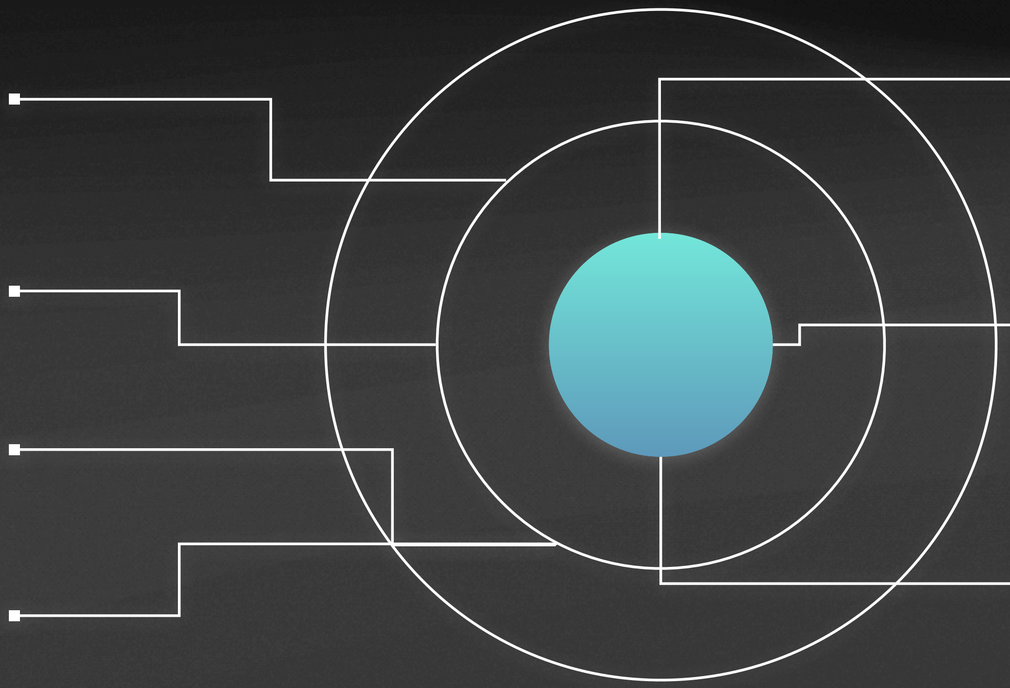
Commonly Used

Windows

macOS

Debian

Fedora



NT Kernel

ntoskrnl.exe

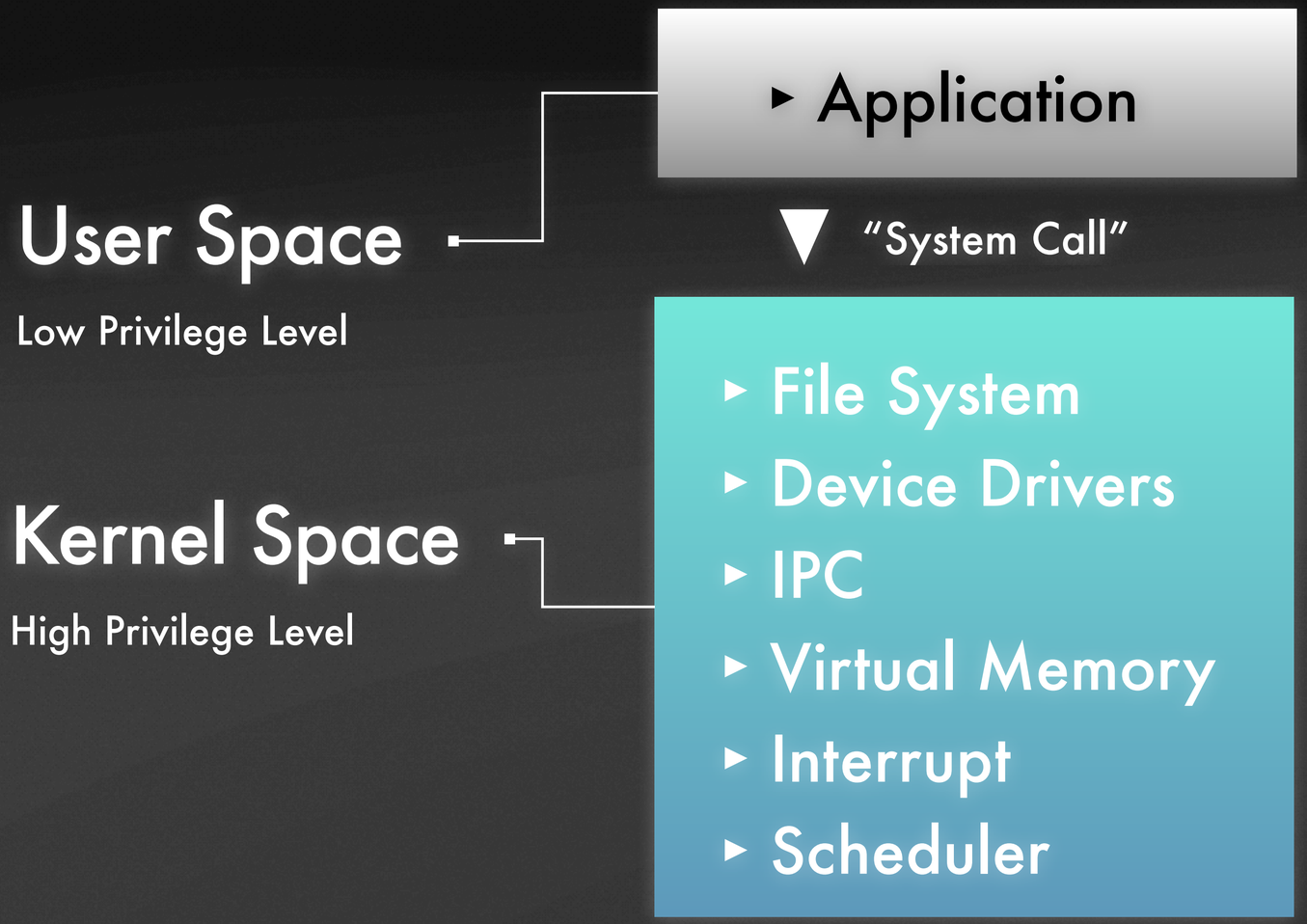
XNU Kernel

Linux Kernel

vmlinux

What is Monolithic Kernel?

Commonly Used Kernel Design



User Space

Low Privilege Level

Kernel Space

High Privilege Level

Monolithic Kernel

Single Layer, All-Inclusive Kernel

- Comprehensive Kernel Space
- Less Fault Isolation
- High Performance
- "Fixed" Policies

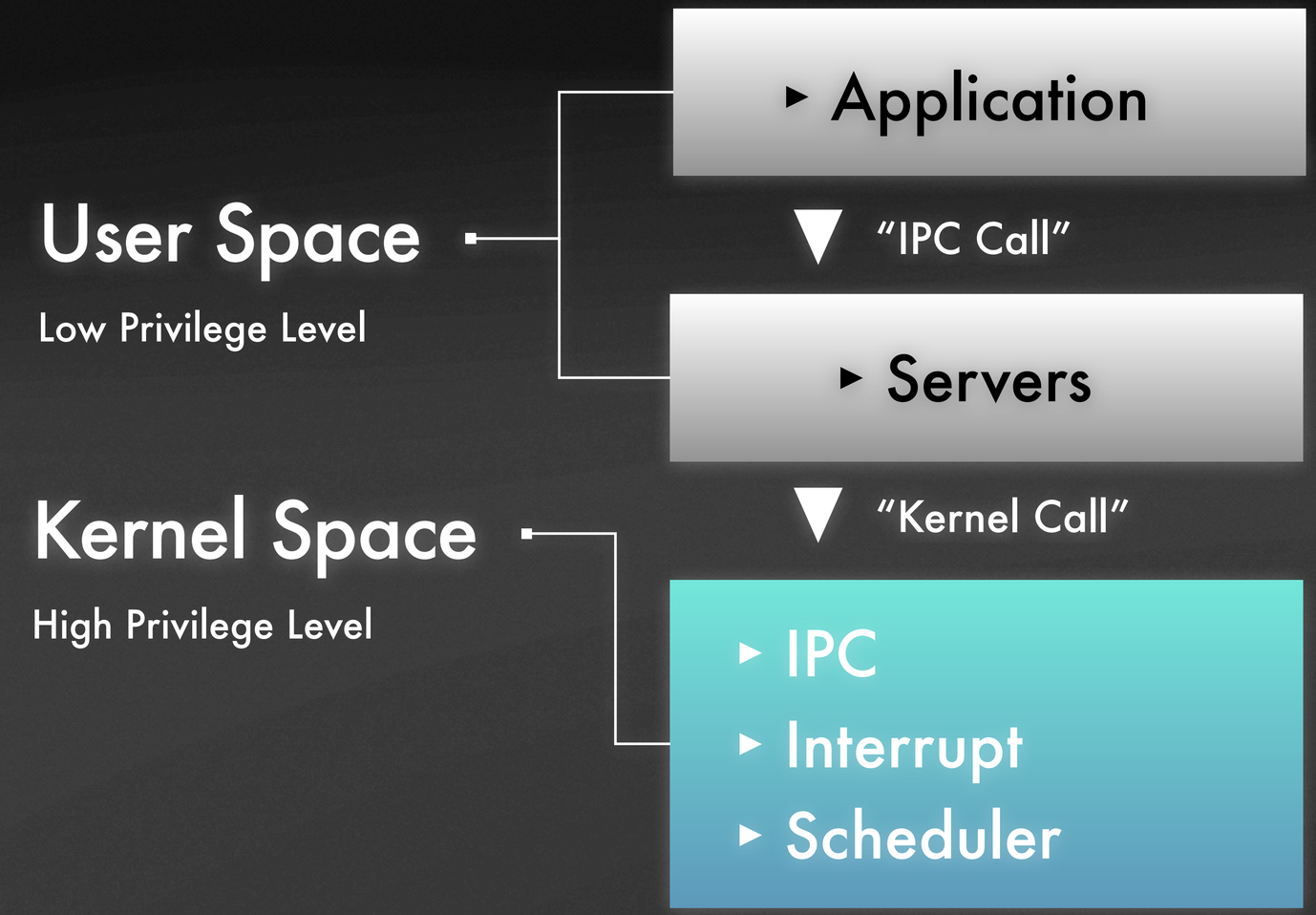
Limitations of the Existing System

Lose Scalability and Security

- Increased computer penetration
  - As Predicted by Moore's Law, the Number of Transistors Per Price is Increasing Year by Year
  - We are Becoming a Society Where Each Communicates with the Other
- In 2024, The Linux kernel Will be as Large as 20 Million+ LoCs!
  - Huge TCB, Huge Attack Surfaces
  - Bloated Code Causes Loss of Maintainability

What is Microkernel?

Differences in the Responsibilities of the Kernel



Microkernel

Minimal Kernel, Modular Services

- Minimalist Kernel Design
- Better Modularity, and Security
- Typically Incurs Overhead by IPC

Memory Management Model in Monolithic Kernel

Challenges Faced by Many Microkernels

- Application

▼ "System Call"

- Virtual Memory
  - `kmalloc`
- Kernel Object
  - SLAB Allocator
- Frame
  - Buddy System

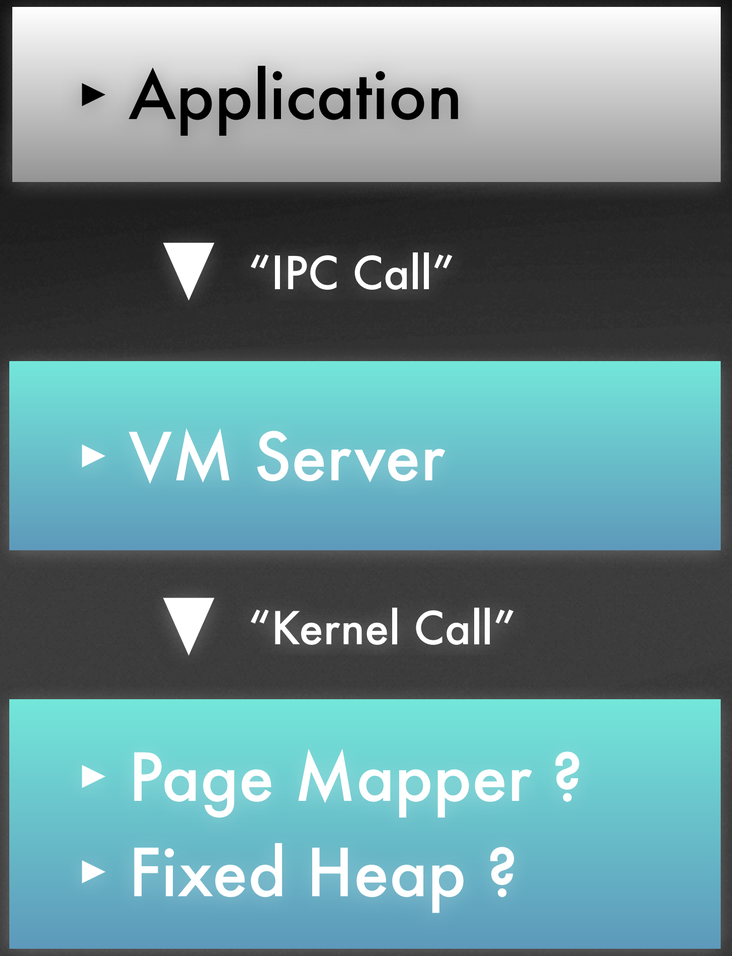
"Fixed" Management

Fixed and Inflexible Mechanisms

- Kernel has its own memory allocation policy
- Changing the allocation policy is difficult
  - Optimizing for specific use cases is challenging

Memory Management Model in Microkernel

Challenges Faced by Many Microkernels



Management Trade-offs

Issues with Previous Systems

- Kernel cannot dynamically allocate its own data
- Fixed heap limits prevent scalability
- Users can access kernel allocation information
  - User-level memory management servers hold too much authority

New Method is Needed!

03

Project

A Modern Microkernel Architecture

Microkernel History

Evolution of Microkernel

Generation #1

e.g., Mach (3.0+), Chorus

- Replace UNIX Pipe with IPC Mechanism
- Low-Performance

Generation #2

e.g., L3, L4
(Alpha, Hazelnut, Pistachio, Fiasco)

- Solved IPC Performance
- Fixed Kernel Memory

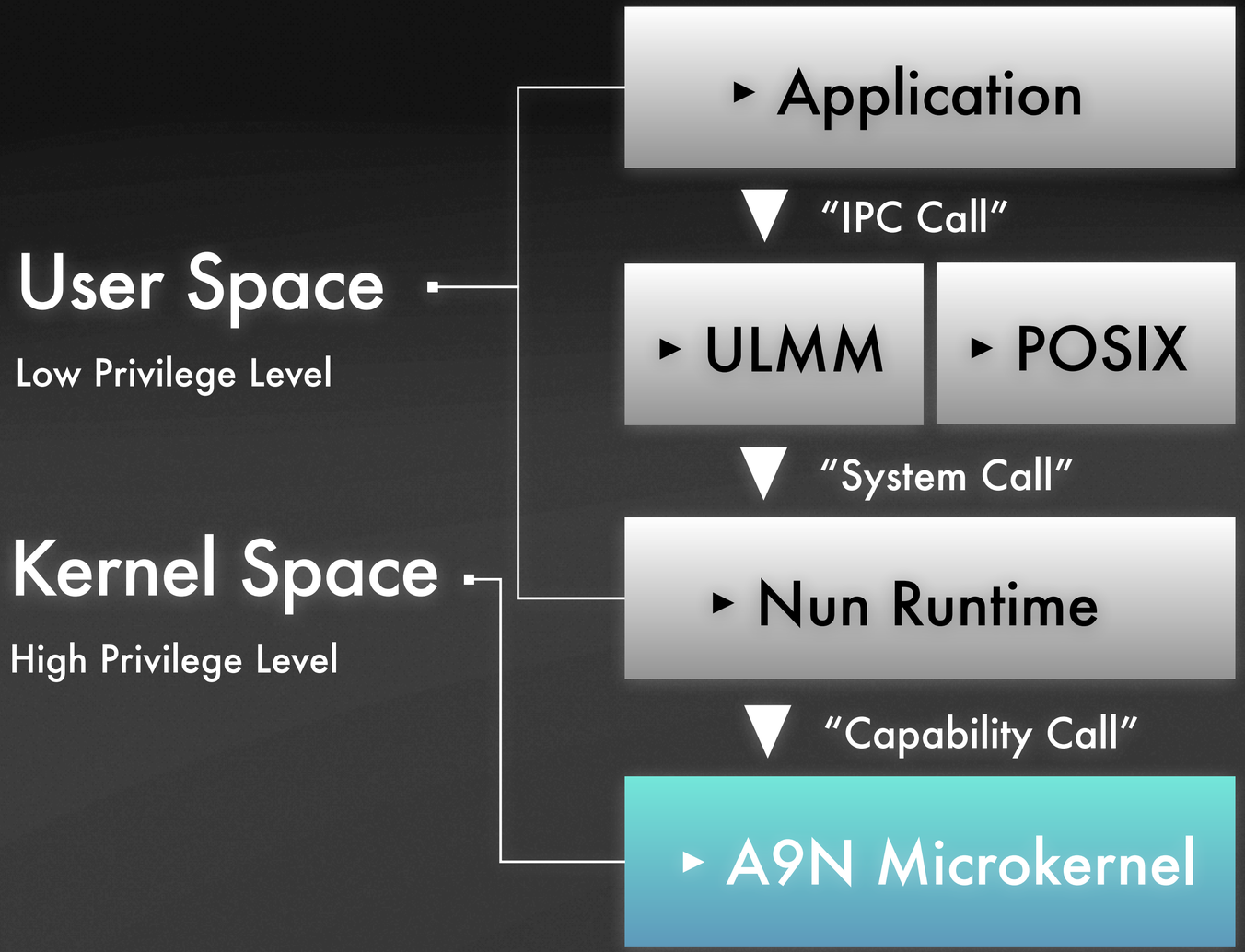
Generation #3

e.g., L4 (OKL4, Fiasco.OC, seL4), Zircon

- Capability-Based Security
- Flexible Kernel Memory

A9N + NUN + KOITO Architecture

A Modern Microkernel Architecture



User Space

Low Privilege Level

Kernel Space

High Privilege Level

User-Level Syscall

Provides Compatibility with Existing Systems

User-Level MM

100% User-Decided Policy;
Buddy System, SLAB/SLOB/SLUB

3rd-Gen Microkernel

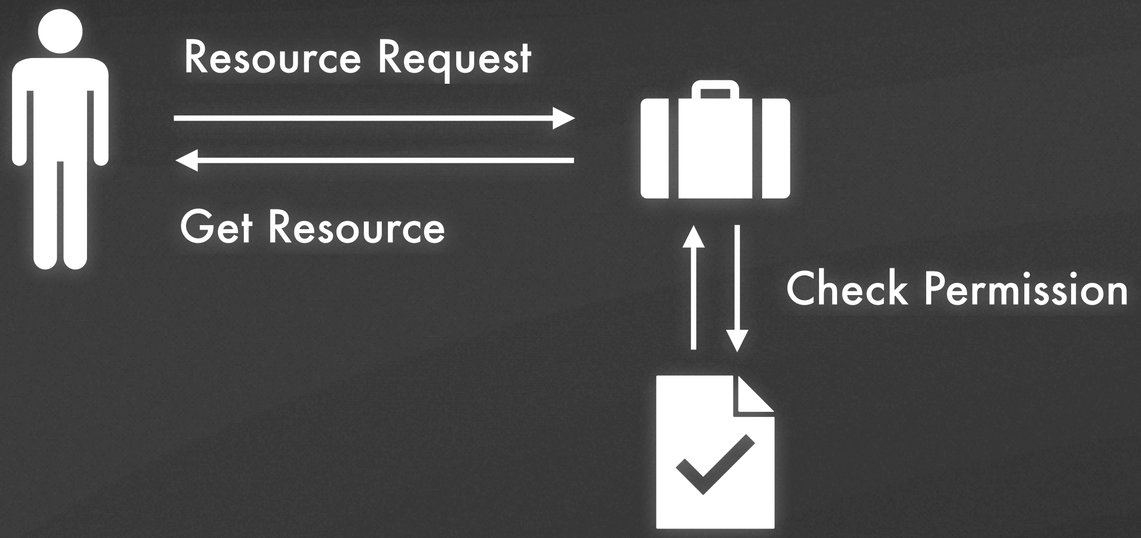
Capability-Based Microkernel with HAL
Ultra-Fast IPC

What is Capability?

Privilege Management with the Object-Capability Model

ACL

Permission-Based Access Control

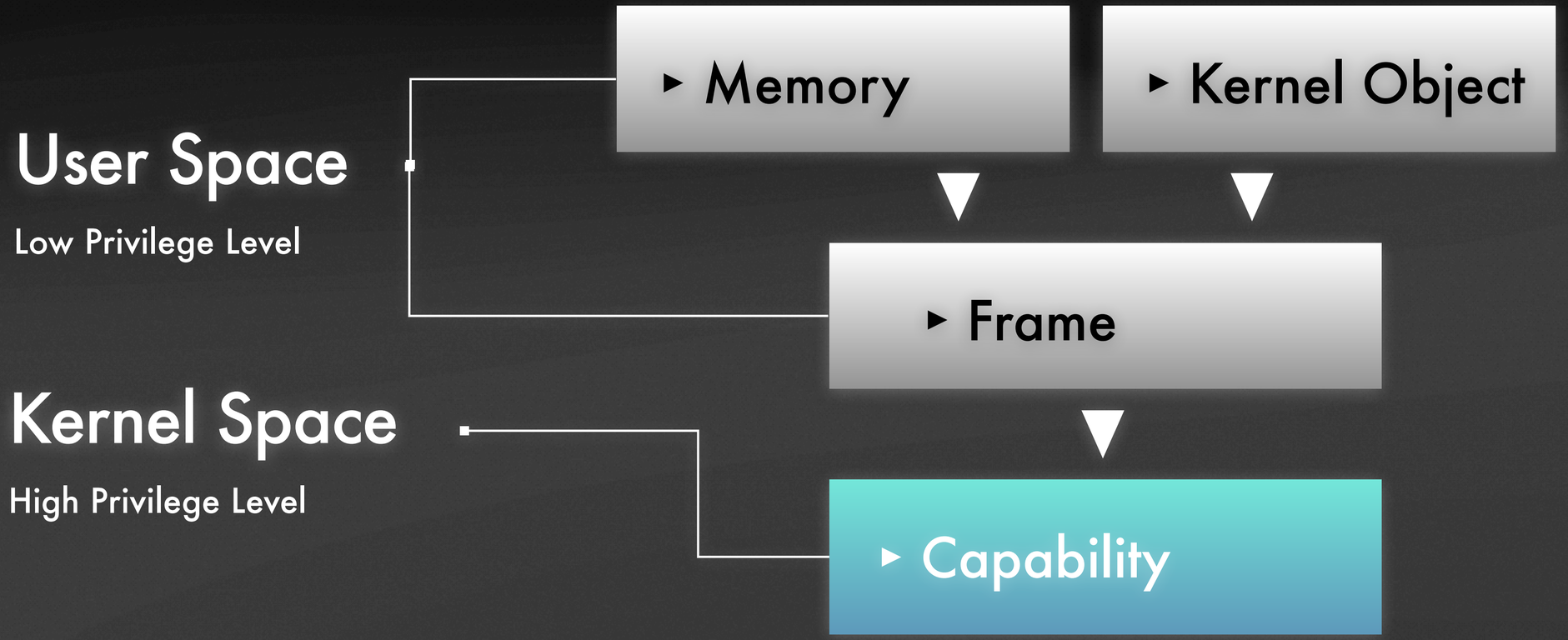


Capability

Token-Based Access Control

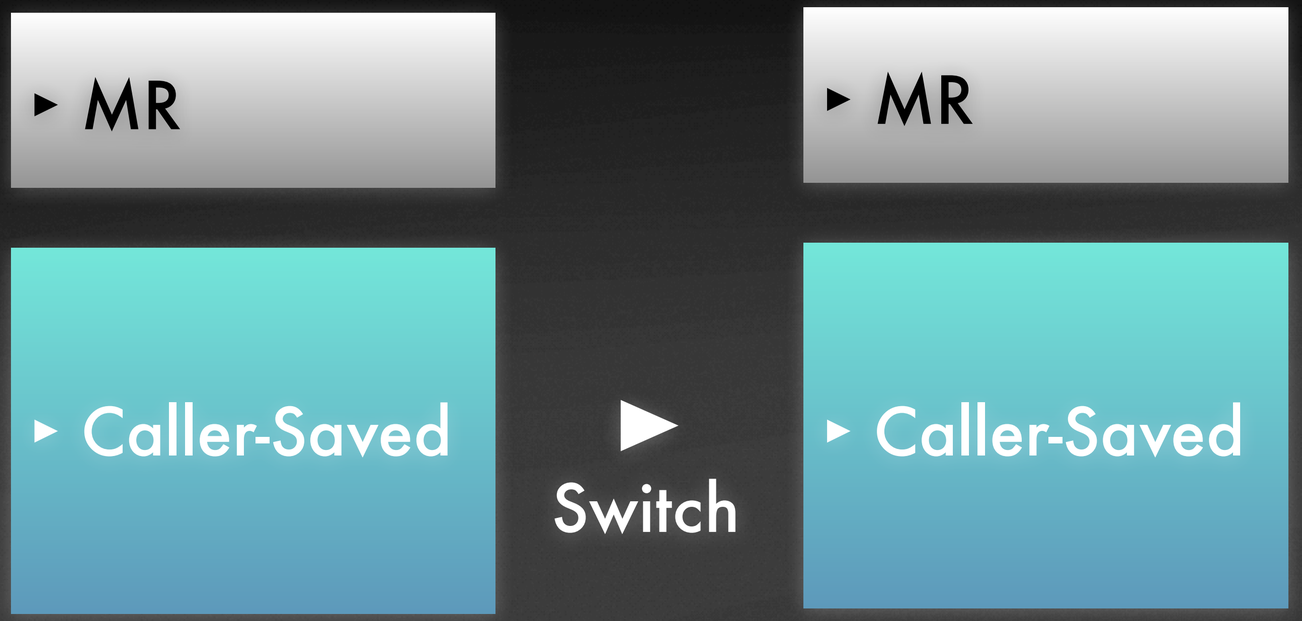
Capability-Based Memory Management

Secure and Maintainable Mechanism



Implementation of Fast IPC

from L4 History



- Sender Context
- Receiver Context
- Quantum Remains
- When in a Waiting

Use Case Optimization

Optimization for Common Use Case :
- Call, Reply-Receive

Message Registers

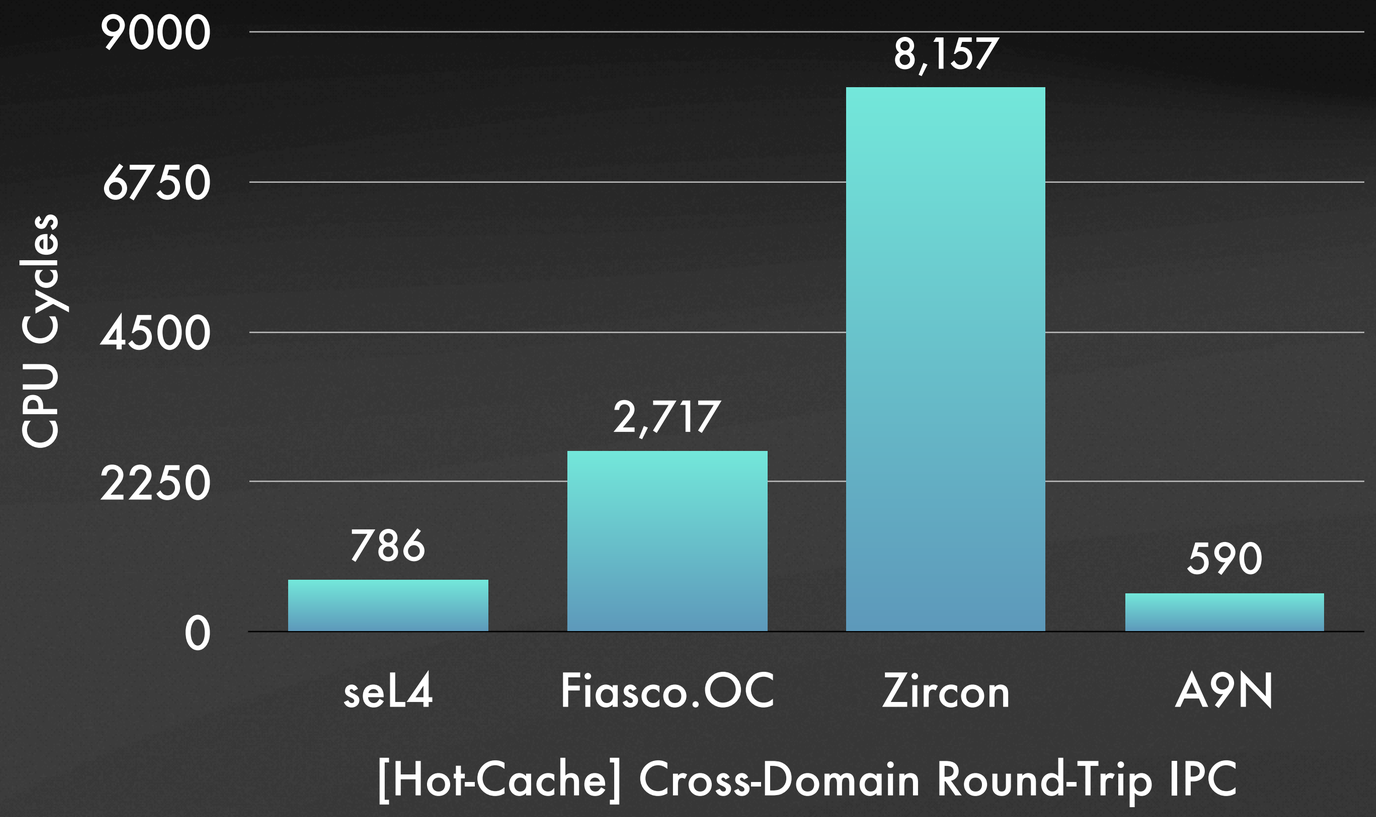
When the Number of Messages Exceeds the Available Hardware Registers, They are Stored in the Message Buffer (on RAM, Like UTCB on L4).

Direct Context Switch

Reduce Scheduler Invocation Costs

IPC Benchmark

Worlds Fastest Microkernel



cf., [Zeyu Mi et al., 2019] [seL4, 2022]

Fastest IPC !

Intel N150@3.6GHz 91.4ns

ACI : Application Capability Interface

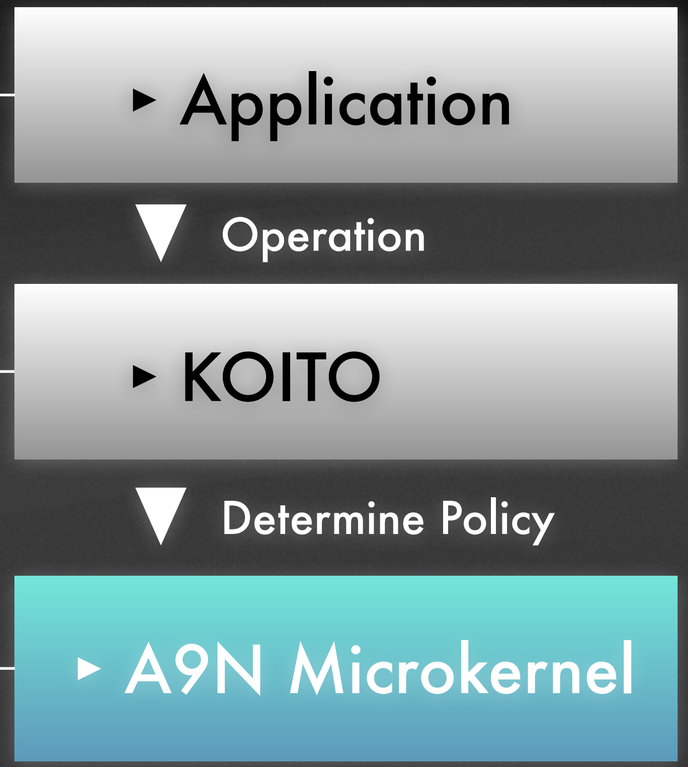
Completely User-Defined Policy

OS Layer

Determine the Course of Operation

Kernel Layer

Only the “Mechanism” of Operation is Realized

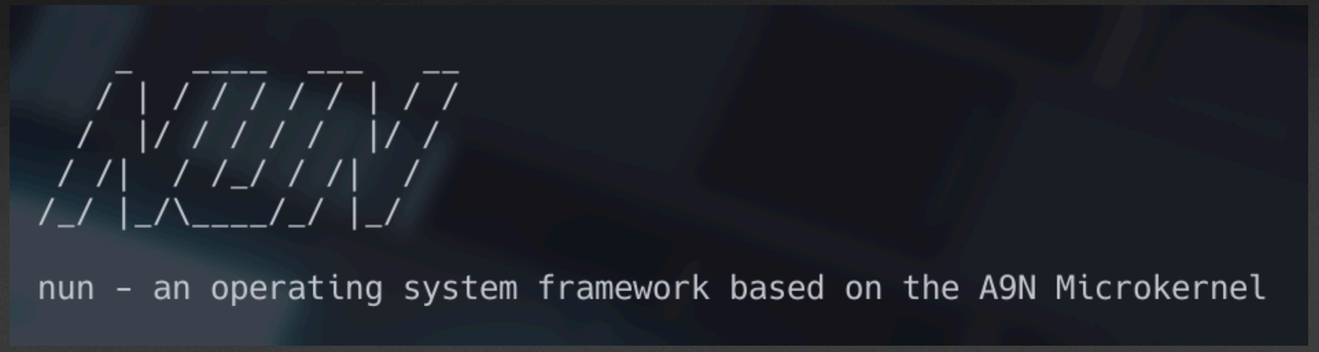


Conventions

Order of Storing Capability,
IPC Buffer Address,
Message Unification, …

Nun : OS Framework on A9N

Powered by Rust



Useful Runtime

Runtime to Build Init Executable for A9N
Written in Rust!

Advanced Abstraction

Runs the Smallest OS with Only 7 Lines

Demo #2: Hello, World! on Nun + A9N

Rust Live Coding (1 min)

Demo #3: A9N Manual

Show a few Pages

KVM (Kernel-Based Virtual Machine)

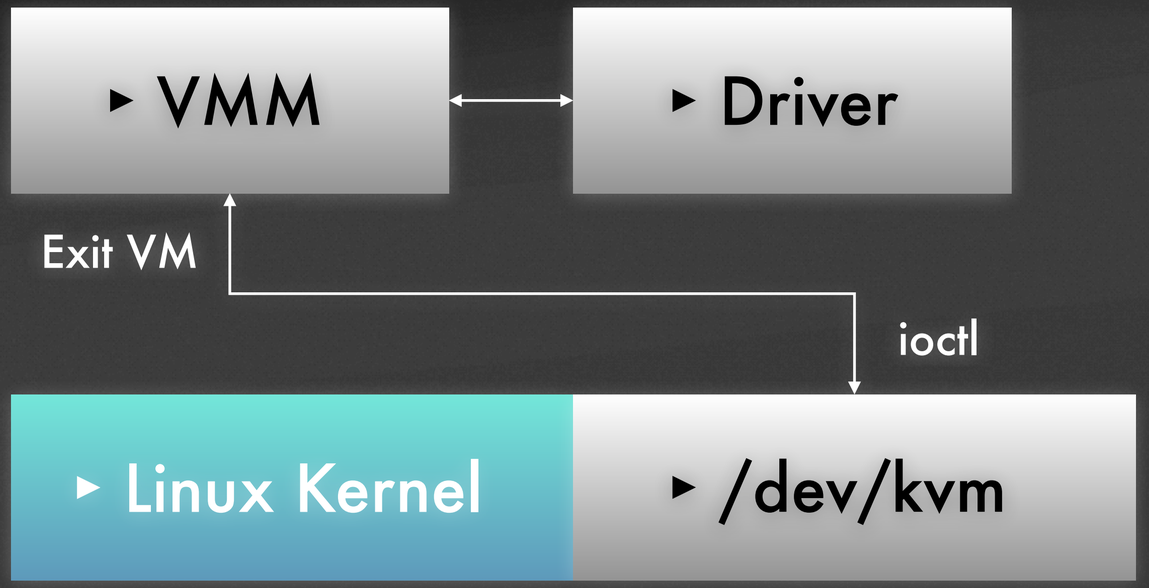
What is Difference ?

User-Level VMM

User : Control Hypervisor via `ioctl(2)` Syscall

Huge TCB

TCB Bloat Caused by the Linux Kernel



Implementation of Hypervisor with “Amateras”

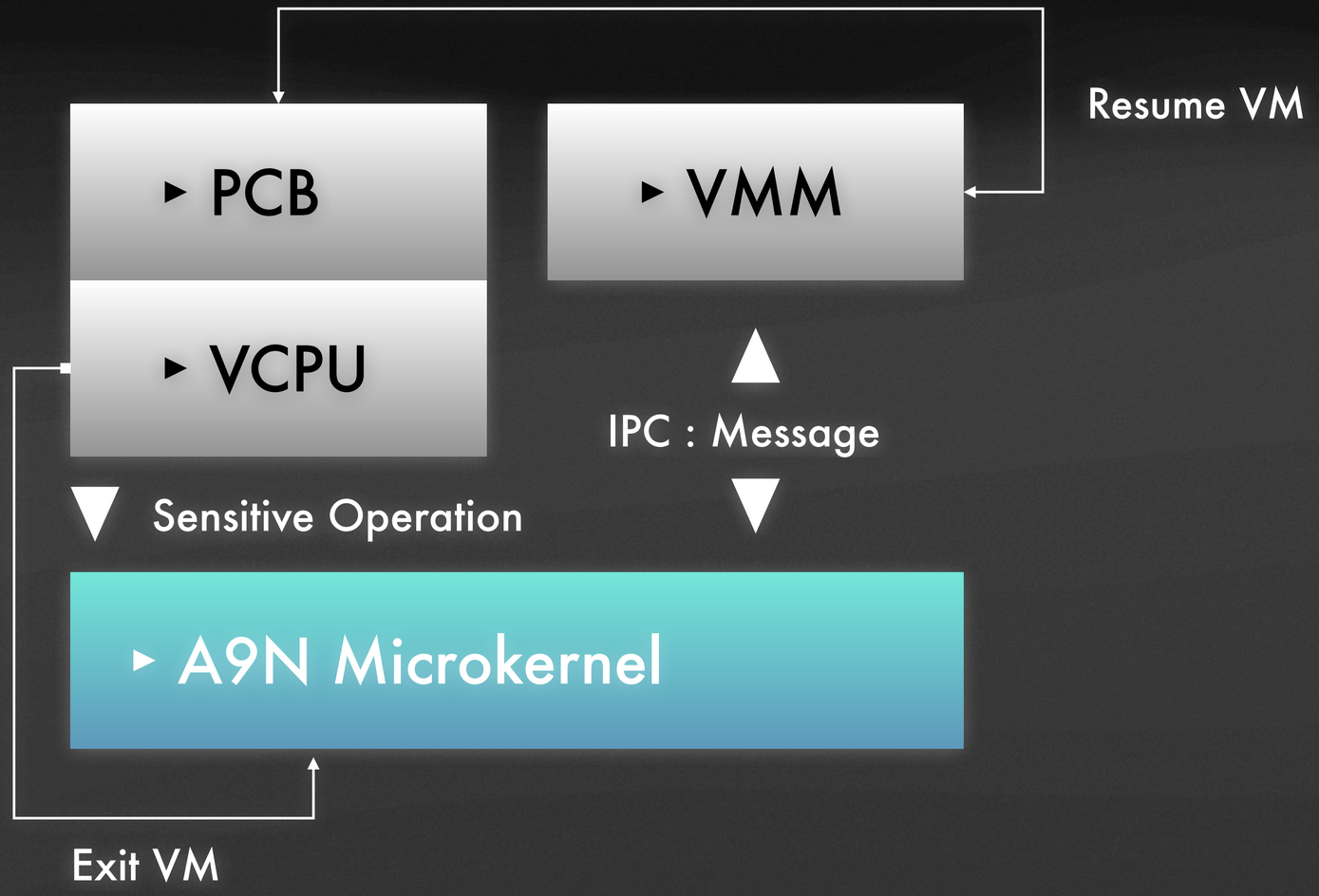
IPC-Based Virtualization Mechanisms

User-Level VMM

Emulate IO, Device and some sensitive operations.
Microkernel-Based :
TCB Much Smaller than KVM!

VCPU Capability

Control Virtual Machine;
“VM Exit Reason” is sent to VMM via IPC



Precedent : Zircon

by Google

3rd-Gen Microkernel

- Rights (Capability) -Based Security
- 50kLoC

Memory Management

- Operation Through Rights
- Kernel-Level PageTable Management

Userland

- Fuchsia OS by Google

Precedent : seL4

by NICTA

3rd-Gen Microkernel

- Capability-Based Security
- Formal Verification (Isabelle / HOL)
- 16kLoC (x86_64)

Memory Management

- No Kernel Heap!
- Operate Through Capability
- Architecture-Dependent API

Userland

- Lions OS?

KOITO on A9N

by horizon2k38

3rd-Gen Microkernel

- Ultra-Fast IPC + Capability-Based Security
- 7kLoC (Kernel) + 7kLoC (x86_64 HAL)
- Hardware-Independent API

Memory Management

- No Kernel Heap!
- Operate Through Capability!
- Full User-Level Memory Management

Userland

- POSIX-Compatible Server
- Running Virtualized Linux

04

Future

What We are Going to Implement

Our Future

Scalable Computing World : Truly Ubiquitous Computing

- Smooth Transition to a Secure System
  - A Small TCB and Capabilities Support Critical Components
  - The POSIX Sever Serves as the Foundation for Securing Existing Systems
  - Virtualization Technology Isolates Threats from the System
- Seamless Interconnection of All Device!
  - High Portability Facilitates Support for a Wide Range of Device

References

- [Blackham & Heiser., 2012] Bernard Blackham & Gernot Heiser. (2012).
  Correct, fast, maintainable: choose any three!. (APSYS '12).

- [Bomberger et al., 1992] Allen C. Bomberger, William S. Frantz, Ann C. Hardy, Norman Hardy, Charles Landau & Jonathan Shapiro. (1992).
  The KeyKOS nanokernel architecture.

- [Elphinstone & Heiser, 2013] Kevin Elphinstone & Gernot Heiser. (2013).
  From L3 to seL4 what have we learnt in 20 years of L4 microkernels?. (SOSP '13).

- [Steinberg & Kauer., 2010] Udo Steinberg & Bernhard Kauer. (2010).
  NOVA: a microhypervisor-based secure virtualization architecture. (EuroSys '10).

- [Zeyu Mi et al., 2019] Zeyu Mi, Dingji Li, Zihan Yang, Xinran Wang & Haibo Chen (2019).
  "SkyBridge: Fast and Secure Inter-Process Communication for Microkernels". (EuroSys`20).

- [seL4, 2022] https://sel4.systems/About/Performance/ (2022-10-09). "seL4 Performance".

Repository

A9N Microkernel

https://github.com/horizon2038/A9N

Nun OS Framework

https://github.com/horizon2038/Nun

A9NLoader (x86_64)

https://github.com/horizon2038/A9NLoader

1

Additional Resources

liba9n : Kernel Support Library for C++20

Modern Embedded C++

- `liba9n::std`
- `liba9n::option<T>`
- `liba9n::result<T, E>`
- `liba9n::not_null<T>`

liba9n::option<T>

Modern Error Handling for C++

option example

```cpp
inline semantic_version::semantic_version(const char *version)
{
    auto captured_parse_pre_release =
        [this](const char *pre_release) -> liba9n::option<const char *>
    {
        return parse_pre_release(pre_release);
    };

    auto captured_parse_build_meta_data =
        [this](const char *build_meta_data) -> liba9n::option<const char *>
    {
        return parse_build_meta_data(build_meta_data);
    };

    parse_base(version)
        .and_then(captured_parse_pre_release)
        .and_then(captured_parse_build_meta_data);
}
```

Null-Safe Type

- Equivalent to Haskell's `Maybe` and Rust's `Option<T>`
- Represents a value that holds either a valid result or an error
- Supports monadic operations as seen in FP languages

liba9n::result<T, E>

Modern Error Handling for C++

result example

```cpp
kernel_result process_manager::try_direct_schedule_and_switch(process &target_process)
{
    return a9n::hal::current_local_variable()
        .and_then(
            [&](cpu_local_variable *local_variable) -> hal::hal_result
            {
                return scheduler_core.try_direct_schedule(&target_process)
                    .transform_error(
                        [&](scheduler_error e) -> hal::hal_error
                        {
                            return hal::hal_error::TRY_AGAIN;

                            switch_to_idle();
                        }
                    )
                    .and_then(
                        [&](process *next_process) -> hal::hal_result
                        {
                            // yield quantum to next process
                            next_process->quantum += local_variable->current_process->quantum;

                            process &preview_process        =
                              *local_variable->current_process;
                            local_variable->current_process = next_process;
                            local_variable->is_idle         = false;

                            return a9n::hal::switch_context(preview_process, *next_process);
                        }
                    );
            }
        )
        .transform_error(convert_hal_to_kernel_error);
}
```

Algebraic Data Type

- Equivalent to Haskell's `Either` and Rust's `Result<T>`
- Represents a common way to express an invalid value
- Supports monadic operations as seen in FP languages

liba9n::not_null<T>

The Power of Null-Safety

not_null.hpp

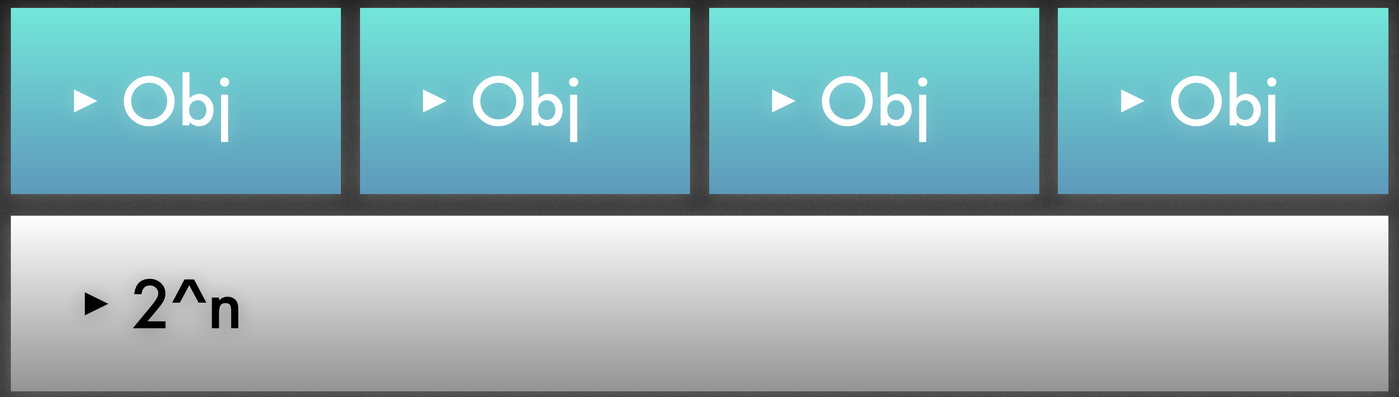
```cpp
template<typename T>
class not_null
{
  public:
    template<typename U = T>
        requires(liba9n::std::is_convertible_v<
                 liba9n::std::remove_reference_t<liba9n::std::remove_pointer_t<U>> *,
                 TPointer>)
    constexpr not_null(const U &reference);
    template<typename U = T>
        requires(liba9n::std::is_convertible_v<
                 liba9n::std::remove_reference_t<liba9n::std::remove_pointer_t<U>> *,
                 TPointer>)
    constexpr not_null(U &&reference);
    constexpr not_null(const not_null &other) = default;
    constexpr not_null(not_null &&other)      = default;
    template<typename U>
        requires(liba9n::std::is_convertible_v<typename not_null<U>::TPointer, TPointer>)
    constexpr not_null(const not_null<U> &other);
    template<typename U>
        requires(liba9n::std::is_convertible_v<typename not_null<U>::TPointer, TPointer>)
    constexpr not_null(not_null<U> &&other);

    template<typename U = T>
        requires(liba9n::std::is_convertible_v<
                 liba9n::std::remove_reference_t<liba9n::std::remove_pointer_t<U>> *,
                 TPointer>)
    constexpr not_null &operator=(U &&reference);
    template<typename U = T>
        requires(liba9n::std::is_convertible_v<
                 liba9n::std::remove_reference_t<liba9n::std::remove_pointer_t<U>> *,
                 TPointer>)
    constexpr not_null &operator=(const U &reference);
    constexpr not_null &operator=(const not_null &other) = default;
    constexpr not_null &operator=(not_null &&other)      = default;
    template<typename U>
        requires(liba9n::std::is_convertible_v<typename not_null<U>::TPointer, TPointer>)
    constexpr not_null &operator=(const not_null<U> &other);
    template<typename U>
        requires(liba9n::std::is_convertible_v<typename not_null<U>::TPointer, TPointer>)
    constexpr not_null &operator=(not_null<U> &&other);
    constexpr TReference operator*() const noexcept;
    constexpr TPointer operator->() const noexcept;
    constexpr explicit operator bool() const noexcept = delete;
    // member methods
    constexpr TReference get() const noexcept;
};
```

Semantic Pointer Wrapper

- Pointer Passing Comes with Responsibilities
- Constructible Only by Reference!
- Effective When Used with `liba9n::result` and `liba9n::option`

User-Level Memory Management

SLAB Allocator

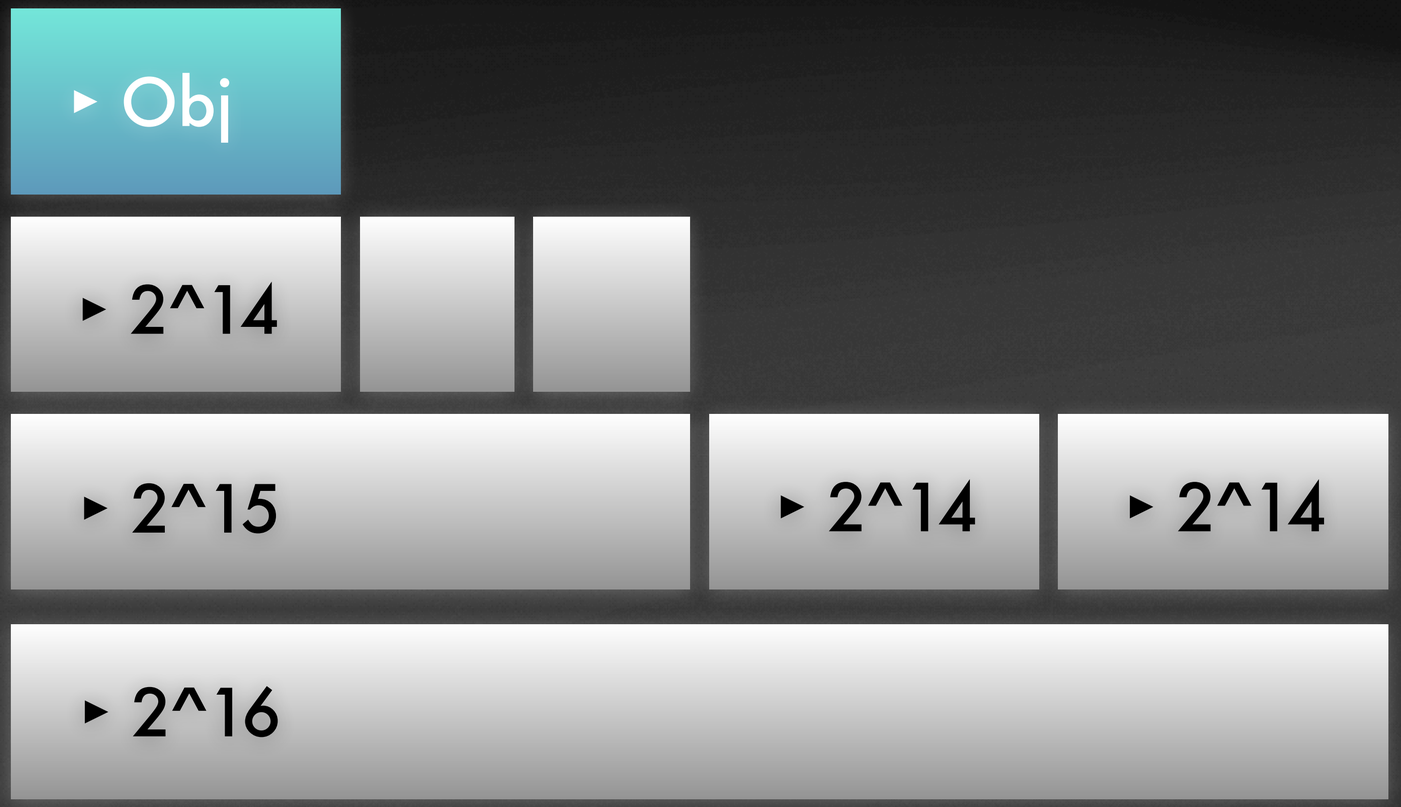


1/n Mapping

The Kernel Object is 2^n bytes, and the Original Generic is Also 2^n bytes. Therefore, It Can be Implemented by Simply Splitting the Generic by the Required Kernel Object.

User-Level Memory Management

Buddy System



Recursive Mapping

In the Case of Buddy System, There are Multiple Types of Kernel Objects to be Converted.
Since All Kernel Objects Including Generic are 2^n bytes, the Requirement is Achieved by Splitting Generic into Child Generic of 1/2 Size and Splitting them Further.