MACHINA

Lessons And Insights From Reimplementing the Mach Microkernel.

Gianluca Guida, 01/02/2025

About me.

Hello! 👋

- Italian in Cambridge (England)
- Hypervisors, Operating Systems, Security
- Currently at Rivos Inc.
- Past employers amongst others: HP, Bromium, Citrix, XenSource
- Ask me about synthesizers!

NB: This talk is about a personal project. Not affiliated with my current or past employers.

About this talk.

Why would anyone reimplement Mach?

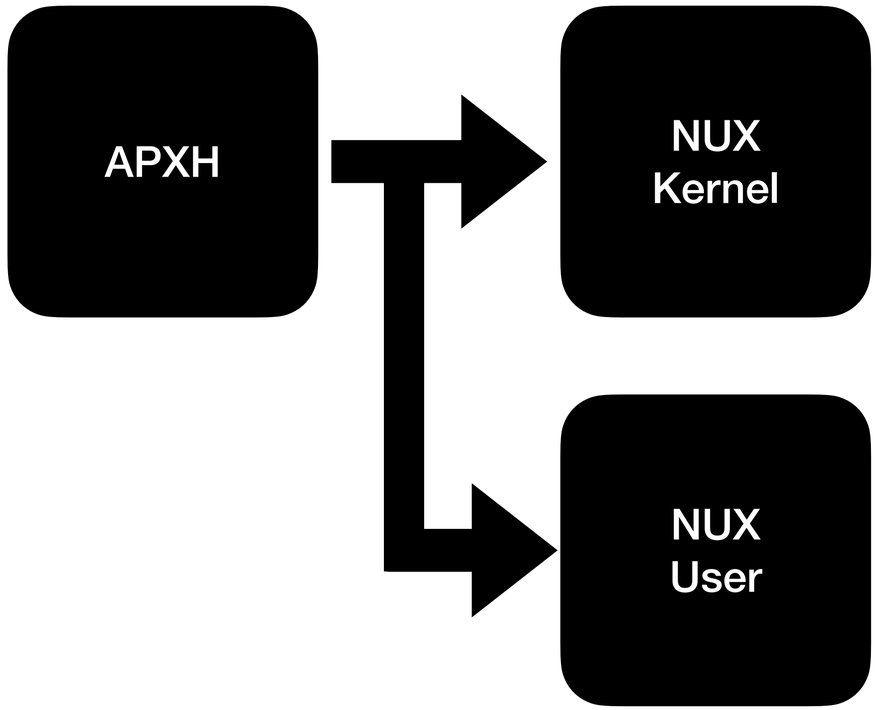
- Part I: History And Motivation
- Part II: A Brief Introduction to Mach
- Part III: The MACHINA reimplementation.
- Part IV: Lessons learned.

Part I: History and Motivation.

History and Motivation.

A Brief Introduction To NUX.

- **NUX: a kernel framework for prototyping OSes *quickly*.**
  - **https://nux.tlbflush.org**
  - **Original motivation to port Murgia Hack (https://mhsys.org) to modern hardware.**
  - **Underlying architectural assumptions similar to MH kernel.**
  - **Result: MH can now run on AMD64 and RISCV64.**
- **Tomorrow's talk in AI devroom will be more detailed about NUX and its architecture.**



History and Motivation.

Mach as a stress test.

- Porting MH to NUX not hard
  - MH has no kernel threads.
  - MH has one user thread per process.
- Mach is possibly the farthest thing from MH kernel
  - Uses kernel threads.
  - Requires implementation of dynamic, refcounted objects.
  - Rich VM that interacts in almost *mysterious* ways.
  - Extensive use of threads in userspace.

History and Motivation.

Mach as a personal *unfinished business.*

- Been interested in the Mach microkernel since the 1990s
  - Only nostalgia can make the memory of downloading GNU Hurd via modem and compiling it on a 486 a beautiful one
- Mach's schism between documentation and code:
  - Documents such as "Mach 3 Kernel Principles" underlines a clean, beautiful architecture.
  - Code is — *for lack of better euphemisms* — hard to follow.
- StoMach's 20 years anniversary!
  - My personal branch of GNU Mach, presented in 2005 at the Hurd Meeting in Madrid.
  - Introduced a COM interface in the device server, allowed to use OSKit drivers.

History and Motivation.

Strategies for NUX-based Mach.

- Two ways I could go on porting Mach to NUX:
    1. Implement a NUX arch in Mach
        - Mach was famous for its portability
        - Arch-dependent interface well separated.
        - The main difficult thing is creating a kernel thread abstraction on top of NUX. Doable but *unnatural*.
    2. Reimplement Mach on top of NUX.
        - Hardest, longest road.
        - *Understand by reimplementing*. Could finally answer many questions I have about this microkernel.
            - Does the code really have to be that complicated and *difficult to read*?
            - Mach was a pioneer on many modern OSes ideas. What choices wouldn't be made today?

History and Motivation.

Strategies for NUX-based Mach.

- Two ways I could go on porting Mach to NUX:
  1. Implement a NUX arch in Mach
     - Mach was famous for its portability
     - Arch-dependent interface well separated.
     - The main difficult thing is creating a kernel thread abstraction on top of NUX. Doable but *unnatural*.
  2. Reimplement Mach on top of NUX.

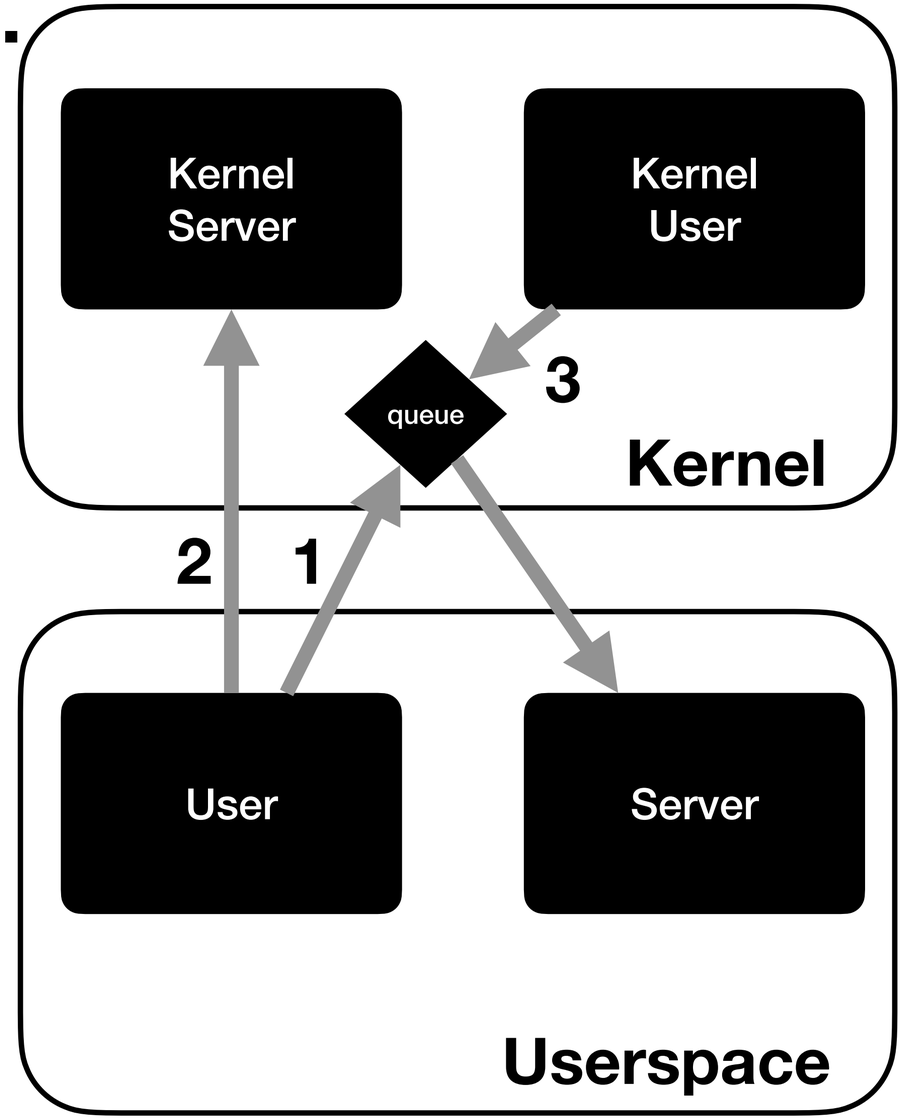
     Of course I chose this!
     - Hardest, longest road.
     - *Understand by reimplementing*. Could finally answer many questions I have about this microkernel.
       - Does the code really have to be that complicated and *difficult to read*?
       - Mach was a pioneer on many modern OSes ideas. What choices wouldn't be made today?

Part II: A Brief Introduction to Mach

A brief introduction to Mach

IPC: Mach Kernel and User interaction.

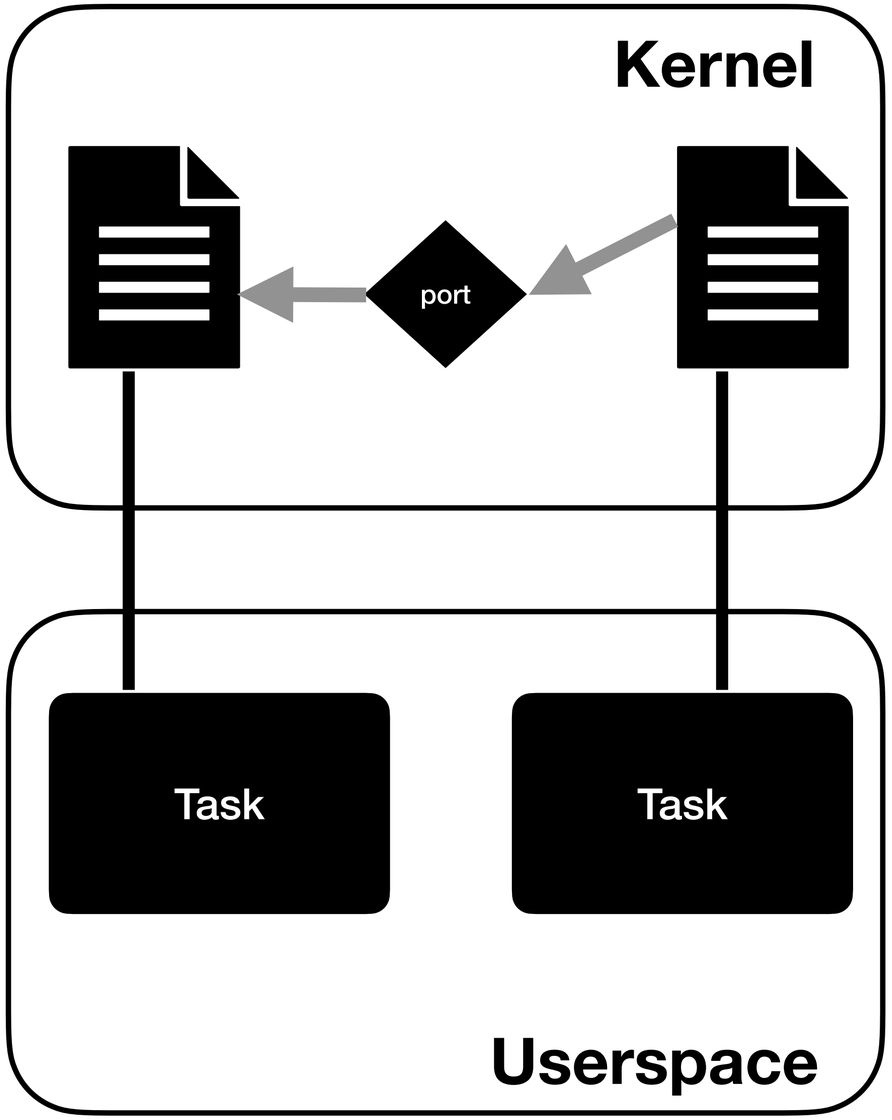
- Mach is famously based on IPCs.
- It is a client-based architecture, and client is called *user*.
- Minimally, the only required system calls are those to send messages.
    - Most of the kernel services are also exported via syscalls, for performance reasons.
- Three different modes supported:
    1. Userspace to Userspace
        - This is the default. Two threads can communicate queueing messages in the kernel.
    2. *Kernel Server*
        - Kernel can receive messages from userspace.
    3. *Kernel User*
        - Kernel can send messages to the userspace.



A brief introduction to Mach

IPC: Port, Port Rights, Port Sets.

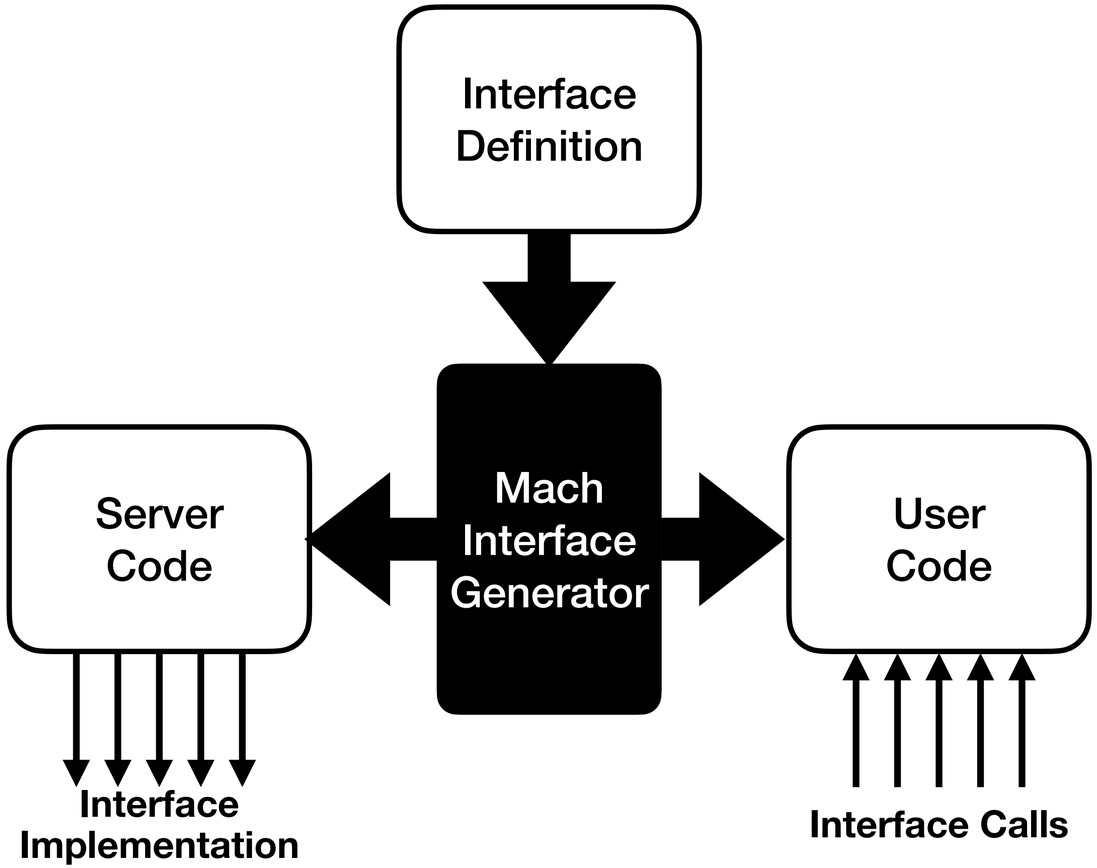
- IPC messages are sent from an end point to another.
- The end point is called a *Port* in Mach.
- Ports are kernel object, and live *refcounted* in the kernel.
- Ports do not have a *global* name space, but each task has a *local name space.*
- For each entry in the name space we have (simplified):
  - Port Right (Send, Recv, Send Once)
  - Port to send the message
- Send port right are effectively *moved* to the task receiving the message
  - Can be cloned though, so can send any number of messages
- Send Once — hence the name — send only once, and the port right self-destruct on sending.
- For each port, there's only one Receive right.
  - Whoever has the right, can receive the messages sent to the port.
  - Receive Rights can be collated into *Port Sets*, so that a single receive request can receive messages from multiple ports.



A brief introduction to Mach

IPC: MIG and User/Server Interface.

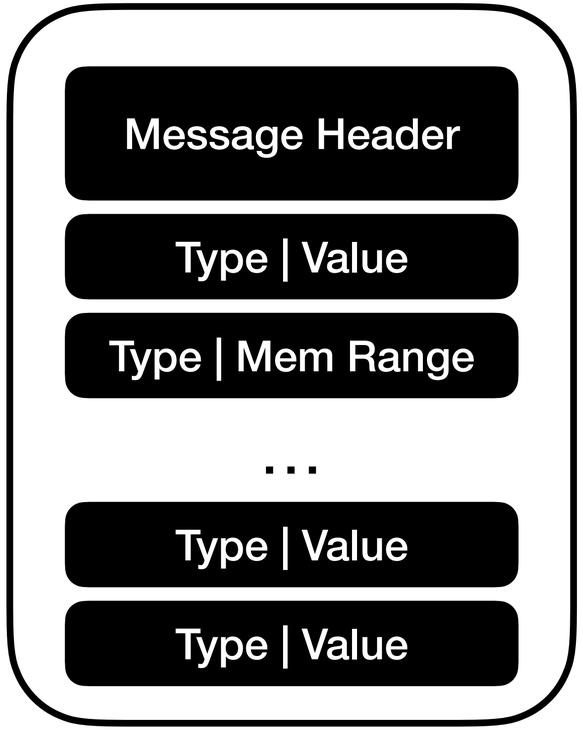
- Mach provides a standard Interface Definition Language and a tool to generate *user* and *server* code: MIG, or Mach Interface Generator.
- The *user* part translates C function calls into *a kernel-defined* message format.
- The *server* part does the opposite, from messages to function calls.



A brief introduction to Mach

IPC: The Mach Message Format

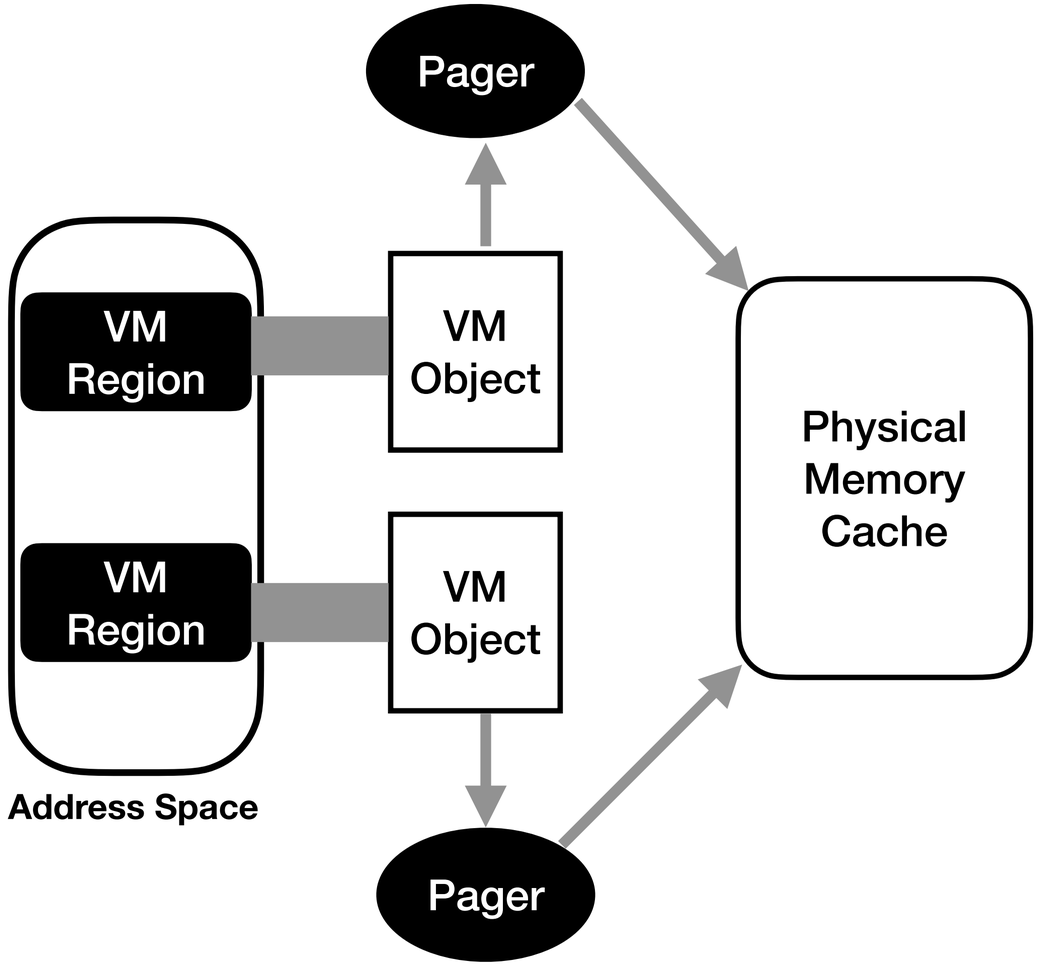
- Mach Messages can be of two types:
  - A. Simple Messages
    - Can be copied directly in the port queue. Data passed has no meaning for the kernel.
  - B. Complex Messages
    - Need to be *parsed* by the kernel.
    - May contain *address ranges* that have to be copied to the receiving task.
    - May contain *port rights* transferred from a task to another.
- The existence of complex messages implies that:
  - MIG and the Kernel are tightly coupled.
  - Message passing cannot be a simple *fast copy*.



A brief introduction to Mach

VM: If you thought the IPC was complex.

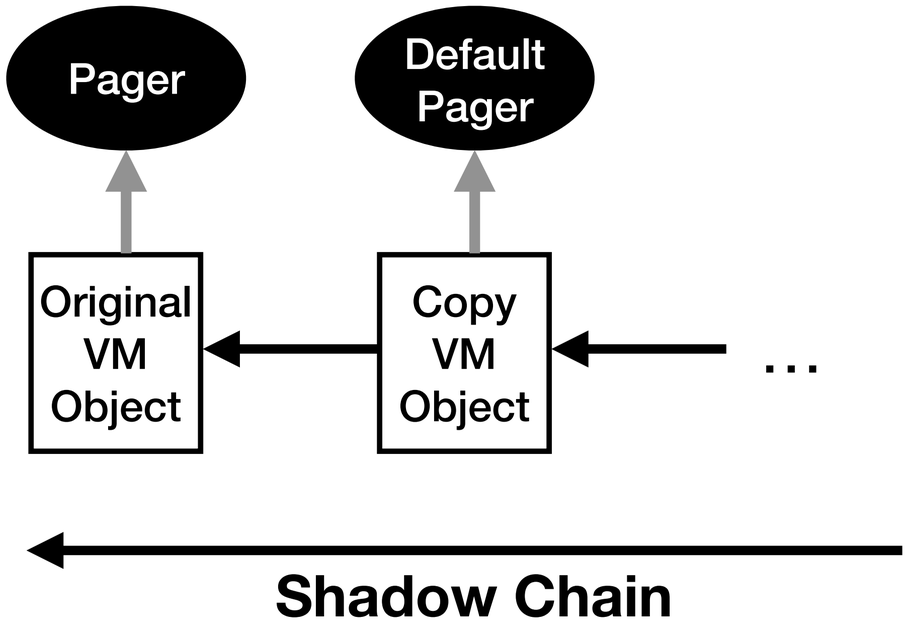
- Mach allows to map parts of *VM objects* into a task's address space.
- Each object has an associated external *pager* that supplies pages requested.
  - This is a *Kernel User* IPC.
- Pagers supply the pages requested to a memory cache.
  - This is a *Kernel Server* IPC.
- There's a special pager, called the *default pager*, that supplies zeroed pages initially.



A brief introduction to Mach

VM: Copy on Write Structures

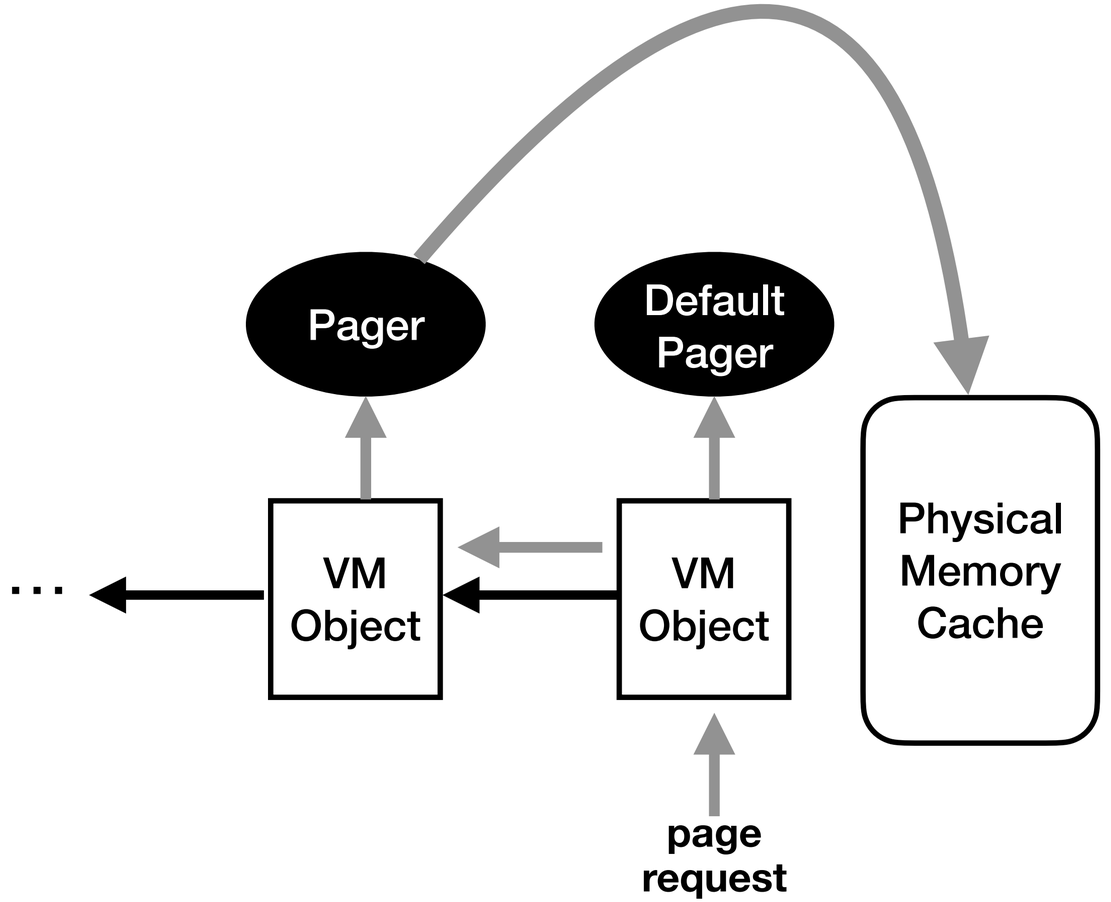
- Mach aggressively uses *copy-on-write* when copying parts of a VM region between address spaces.
- When VM object *A* is copied with copy on write, a new, empty VM object *B* is created.
  - VM Object A *shadows* VM Object B.
- VM Object A or B can further be copied, creating a *shadow chain* between VM objects.



A brief introduction to Mach

VM: Copy on Write Dynamics I

- When a VM object needs to *retrieve* a page (e.g., page fault)
    1. The current pager is checked for the missing page
    2. If pager doesn't have that page, the request moves to the shadow
    3. If the shadow VM object has the page resident, return the page. Otherwise search the pager.
    4. Whichever pager has the page, adds the page to the *requesting* VM object cache.



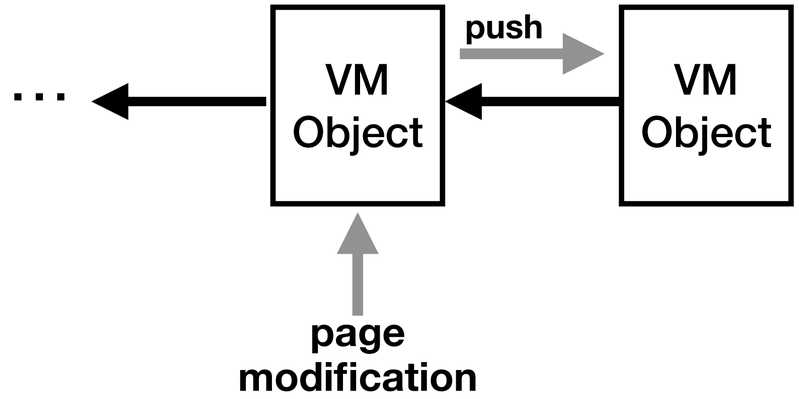
A brief introduction to Mach

VM: Copy on Write Dynamics II

- When a page in a VM object is modified:

  1. Before allowing modifications, the current page is *pushed* to the object we are shadowing.

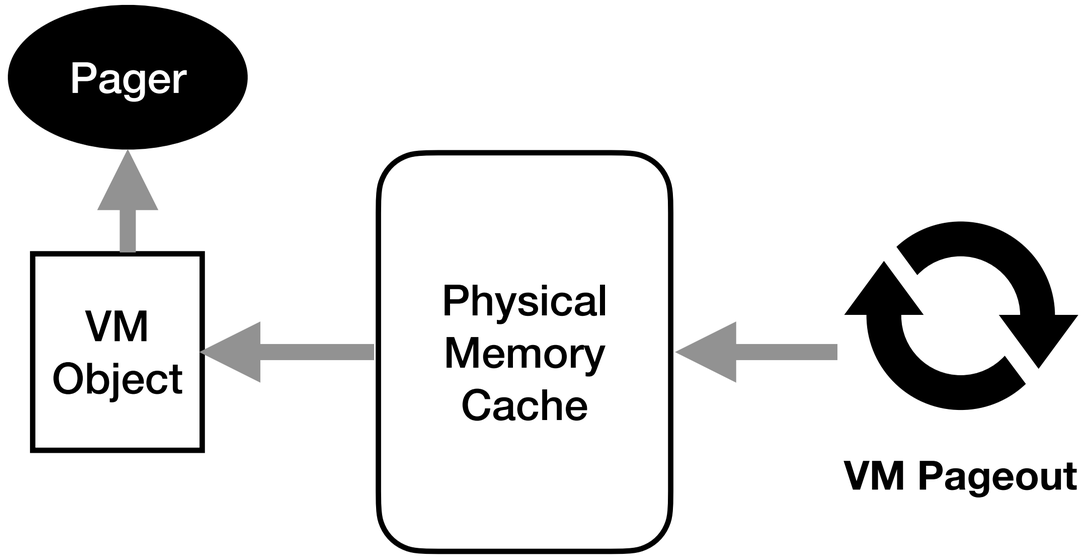
  2. The VM object obtains a copy of the page.



A brief introduction to Mach

VM: Paging Out

- Kernel tries to maintain a certain number of pages free.
- When memory is low, the VM pageout scans memory in cache and instructs VM Objects to page out least used pages.



A brief introduction to Mach

VM: Complications, as if more were needed.

- The *copy-on-write* mechanism described is only *one of three* mechanisms supported.
  A. MEMORY_OBJECT_COPY_DELAY: The mechanism described
  B. MEMORY_OBJECT_COPY_CALL: Notify pager before copying.
     - From Mach 3 Kernel Principles, 1992: "*(Important note: This feature is scheduled for replacement. It is un-tested and believed not to work.)*"
     - It is still there…
  C. MEMORY_OBJECT_COPY_NONE: Always make physical copies of data.
- When switching between COPY_DELAY to COPY_NONE, the kernel has to fetch all pages swapped out, and make physical copies.
  - Only seen in a 1994 commit in the *ext2fs* translator of the GNU Hurd.

A brief introduction to Mach

VM: External VM Interface

- **Mach User interface does not think in term of VM objects, but in terms of Address Ranges.**
- **This means that operations issued from the user to the VM subsystem might spawn *multiple* VM objects in a single operation.**
- **This also includes address ranges passed in messages.**

Part III: The MACHINA reimplementation.

The MACHINA reimplementation.

Reimplementation principles.

- I hope I convinced you how complicated this is.
- Actively avoided looking at Mach source code.
  - Wanted to reproduce the interface, and see what the code would look like.
- Divide and Conquer: implement the two complex system, IPC and VM, separately.
  - In Mach, they are actually tightly coupled, as messages sent to the kernel are themselves an address space range, so subject to the VM object logic.
  - In MACHINA, messages are sent through a special buffer, always mapped and shared between kernel and user.

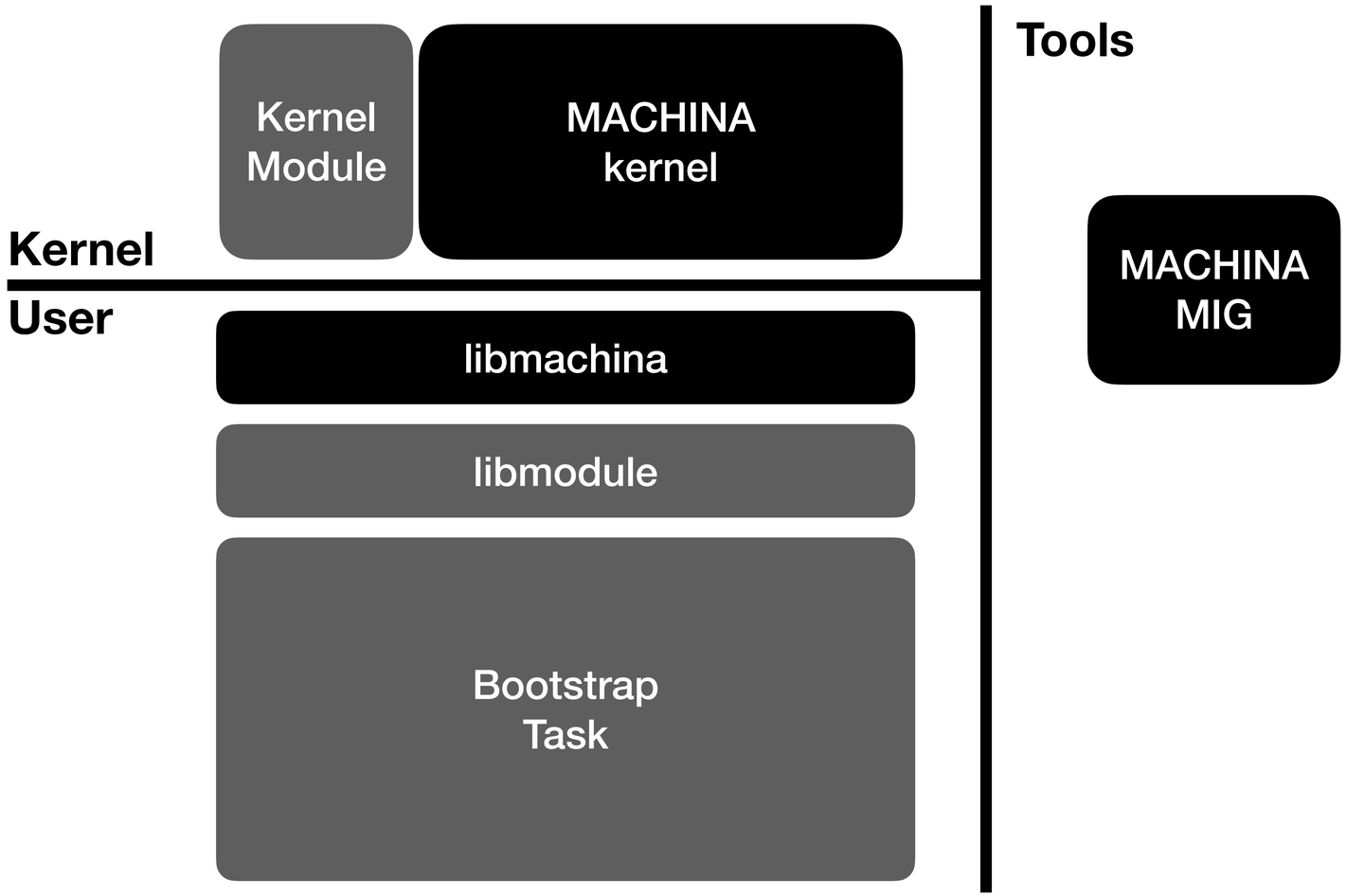
The MACHINA reimplementation.

Modularity

- **MACHINA includes the concept of modules:**
  - **The kernel core defines the kernel objects and their interactions.**
  - **A module defines the actual user interface:**
    - **Kernel IPC Interfaces**
    - **Extra System Calls**
  - **Modules currently being developed:**
    - ***test*: a testing module for development**
    - ***mach3:* based off CMU Mach defs files and headers, implements classic Mach.**

The MACHINA reimplementation.

Software stack.



The MACHINA reimplementation.

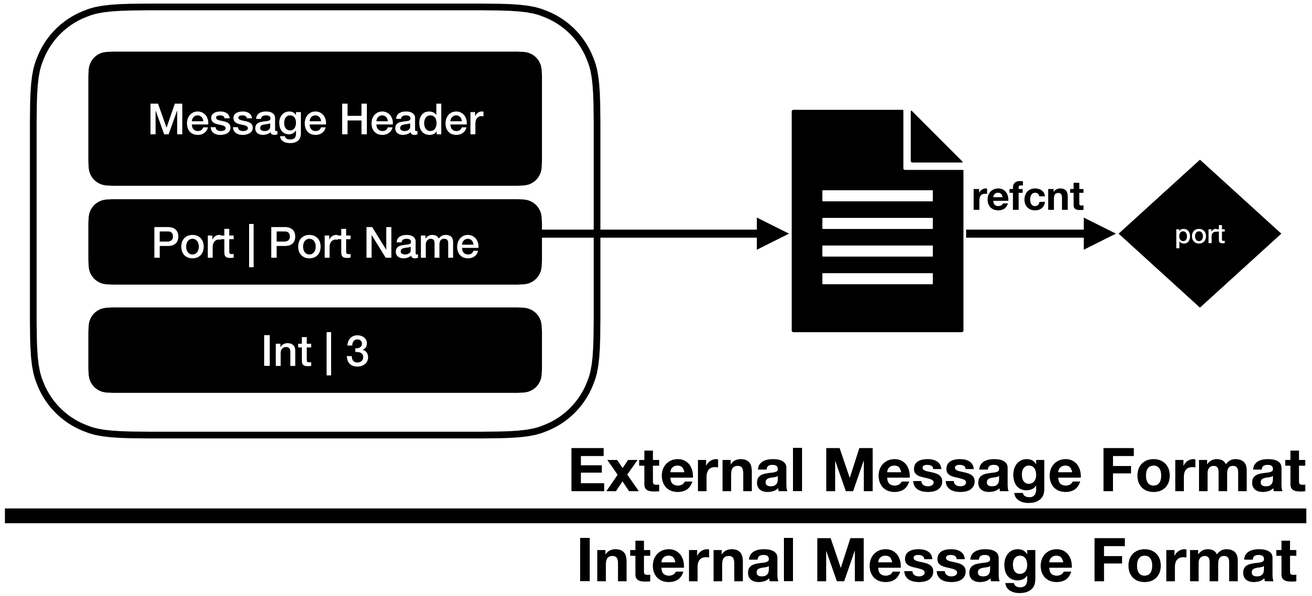
MACHINA IPC implementation.

- Two types of messages:
  1. “External Format”

     Reference to kernel objects are task-local.

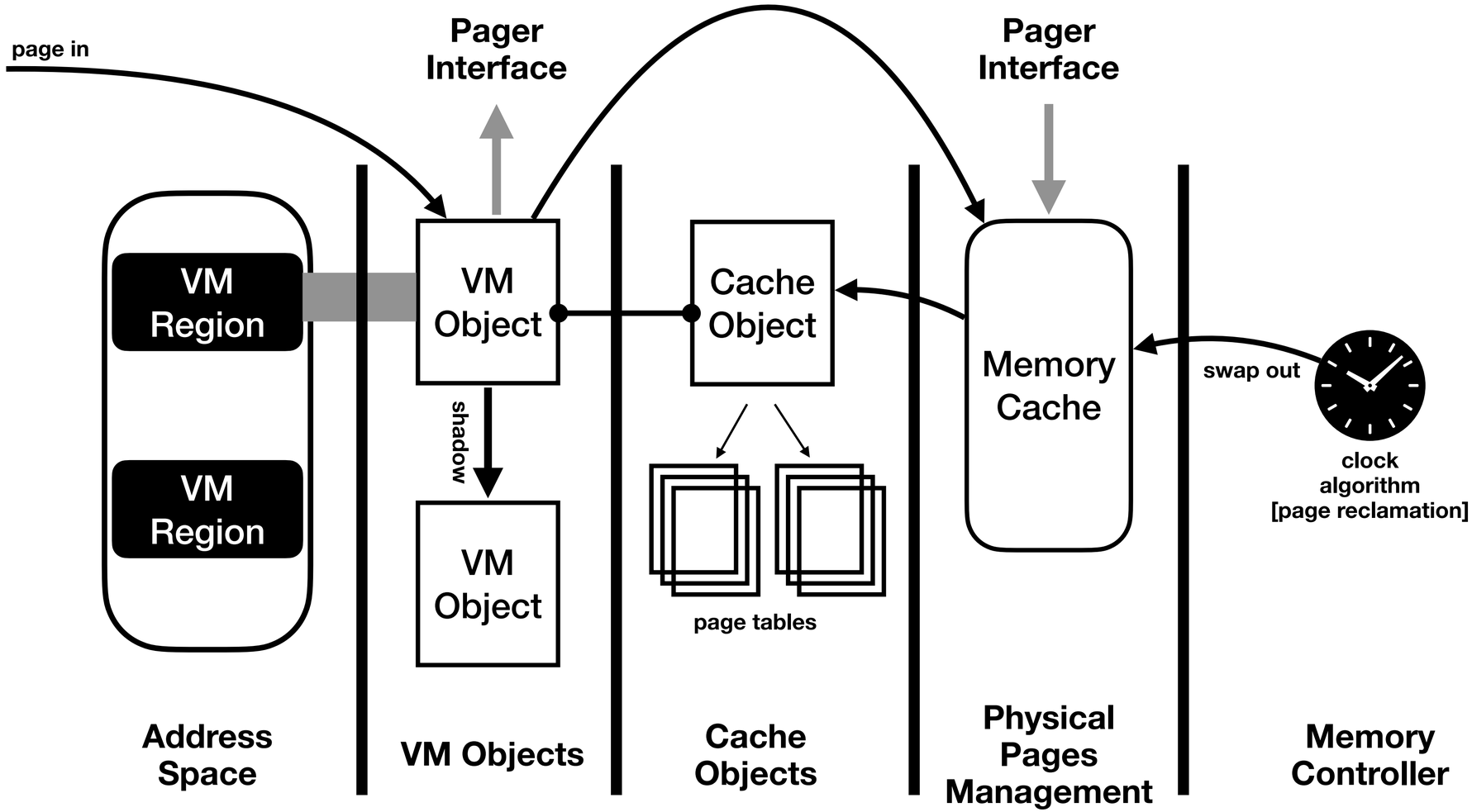
  2. “Internal Format”

     References to kernel objects are reference counted pointers.
- When receiving a message:
  - The message is translated from External to Internal, then queued.
- When sending a message:
  - The message is translated from Internal to External, then copied to the thread’s message buffer.



The MACHINA reimplementation.

MACHINA VM implementation.



Part IV: Lessons learned.

Lessons learned.

Is it worth it?

- I *knew* Mach was complicated. Now I know much better *why*.
- Does rewriting makes sense? Seems like it:
  - Code, at least when it comes to locking objects and following code path, much *simpler* — although *incomplete*.
  - Early benchmarks of MACHINA IPC (no optimizations attempted) seem to be at least on par with more mature Mach implementation.
  - Code being simpler, it is easier to modify.
  - Having the IPC kernel interface defined in modules allow to have a way to generate *Mach-like* systems, and further expose the flexibility of the Mach design

Lessons learned.

Is it a ‘modern’ design?

- Many choices wouldn’t be made today:
  - IPC would probably be done on a ring buffer, rather than on a queue of a controllable maximum message count.
  - In modern microkernel system, the userspace is usually trusted to remember which object it mapped at which address.

    Having the user interface for the VM operate at VM object, and not generic address range, would simplify the VM architecture by a lot.
  - Port Rights counters are really complicated. Although it is so linked to the nature of Mach that it’s difficult to say how it would be done differently.

MACHINA: Current Status

Incomplete but core functional.

- Core is fairly complete. Hardest parts implemented first.
  - Port and other kernel object handling is implemented.
  - IPC misses *notifications.*
  - VM is implemented. External pager interface currently unused.
- Missing parts:
  - Many functionalities regarding task, thread, host, etc necessary to have a running system.
  - Mach3 module currently off-branch.
- What does it do:
  - Not very much, except booting a test bootstrap that stresses IPCs and VMs.

Thank you!

For more information:

https://tlbflush.org

https://nux.tlbflush.org

https://github.com/glguida/machina