Porting GGML to the NUX Kernel Development Framework.

Gianluca Guida, 02/02/2025

About me.

Hello! 👋

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

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

About this talk.

Bringing GGML to a constrained environment.

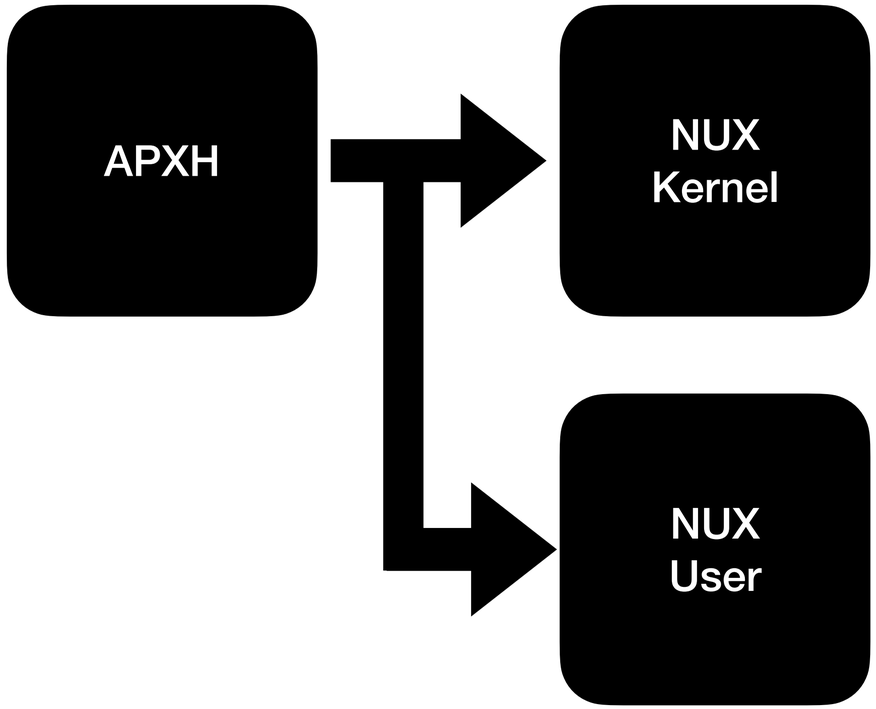
- **Part I: What is NUX?**
- **Part II: What is the minimum requirement to run GGML?**
- **Part III: Porting GGML to NUX.**
- **Part IV: Considerations and Q&A.**

Part I: The NUX kernel framework.

The NUX kernel framework.

NUX systems: an overview.

- **Three main components:**
  1. **APXH (αρχη): The ELF bootloader**
     - **Bootstrap from platform**
     - **Loads Kernel and User binaries.**
     - **Launches the kernel and disappears.**
  2. **NUX kernel: Kernel-Space component**
     - **Handles interrupts, exceptions and syscalls.**
  3. **NUX User: User-Space component**
     - **Requests services to the kernel via syscalls.**
     - **Initial user program running for system initialization or single user binary system.**

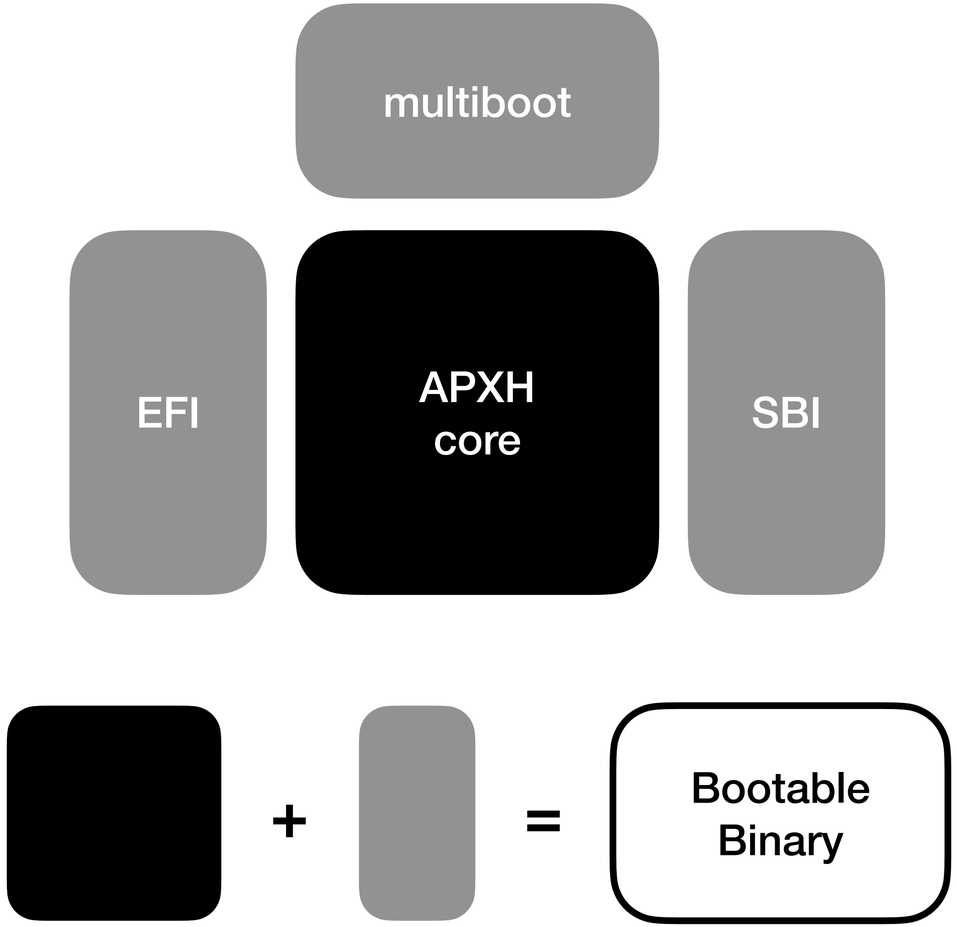


The NUX kernel framework.

APXH: an introduction.

APXH (αρχη): The ELF bootloader

- Supports EFI (RV64 and x86), multiboot, SBI
- Loads a kernel ELF in memory
- Loads a user ELF in memory.
- Program Headers for special boot information and kernel memory layout, e.g.:
    1. Framebuffer
    2. Boot and Platform Information
    3. 1:1 memory map
- Designed to be easily portable to new platforms and architectures.



The NUX kernel framework.

Kernel-code architecture.

libhal: CPU bootstrap and abstraction

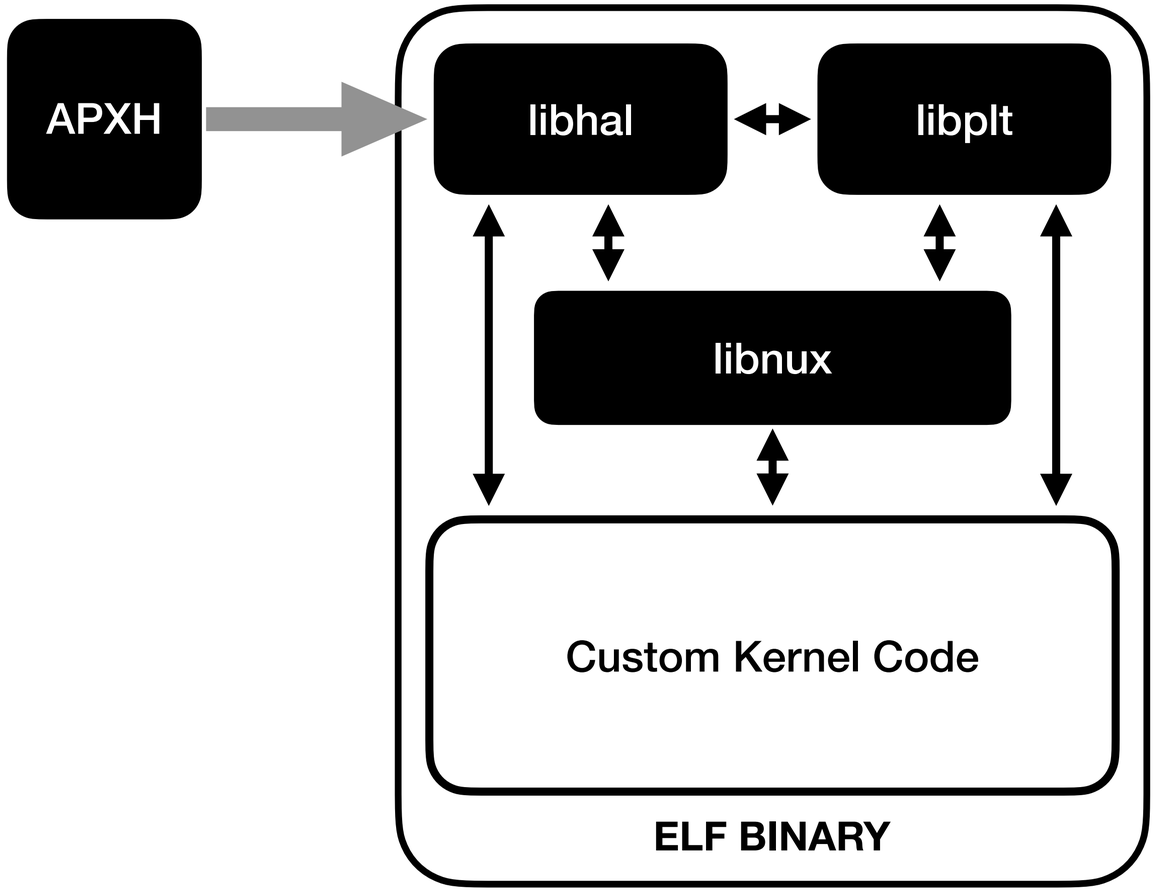
- ***libhal_x86*: i386 and AMD64**
- ***libhal_riscv*: RISCV-64**

libplt: Platform configuration and abstraction

- ***libplt_acpi*: i386 and AMD64**
- ***libplt_sbi*: RISCV-64**

libnux: Higher level functionalities.

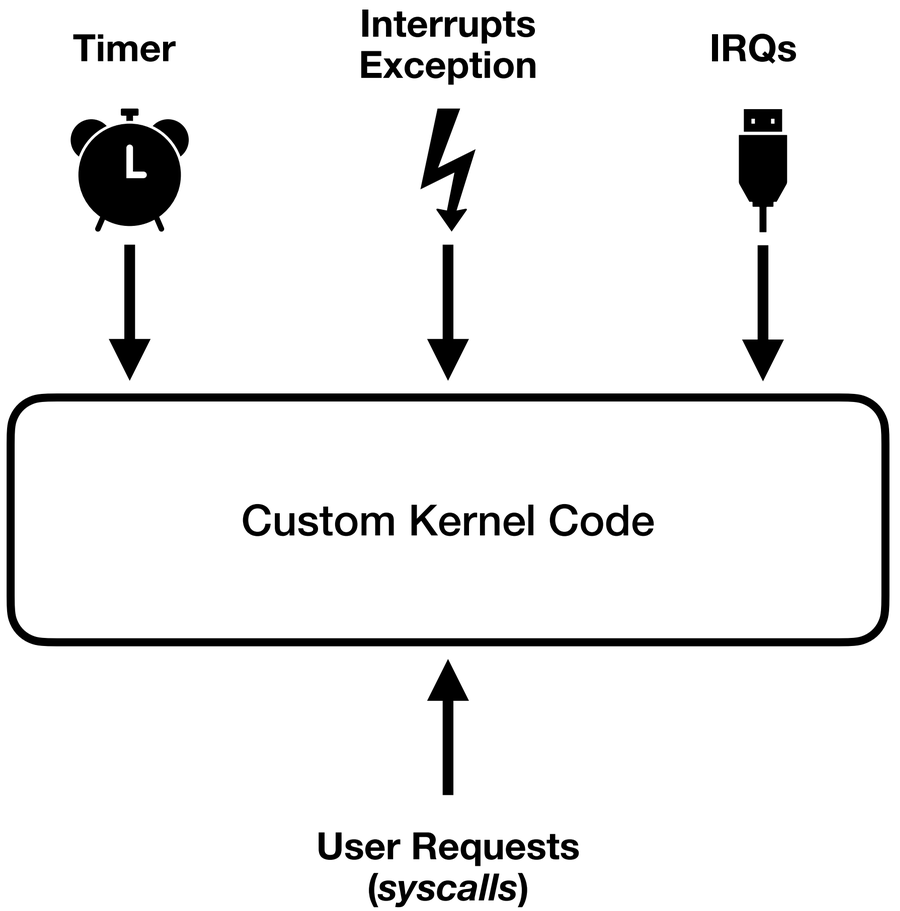
- **memory allocators and mapping**
- **user stack frames**
- **global TLB coherency**
- ***printf*, panics, etc.**



The NUX kernel framework.

The Custom Kernel Code world-view.

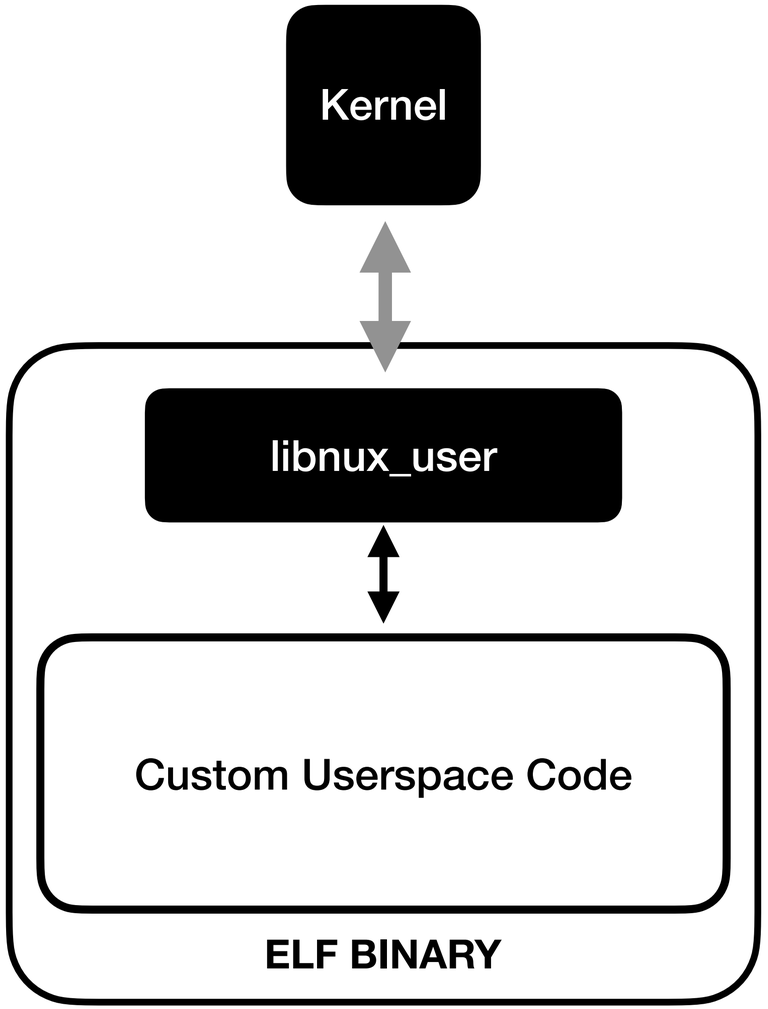
- A kernel in NUX is defined by its *entry* functions.
- Two initialisation *entries*:
  - `main()`: Bootstrap CPU initialisation
  - `main_ap()`: Secondary CPUs initialisation.
- Runtime *entries*:
  - `entry_ipi()`: Inter-processor Interrupt entry.
  - `entry_alarm()`: Timer entry.
  - `entry_ex()`: Exceptions entry.
  - `entry_pf()`: Page-Fault entry.
  - `entry_irq()`: IRQ entry.
  - `entry_sysc()`: User requests entry.



The NUX kernel framework.

User-code architecture.

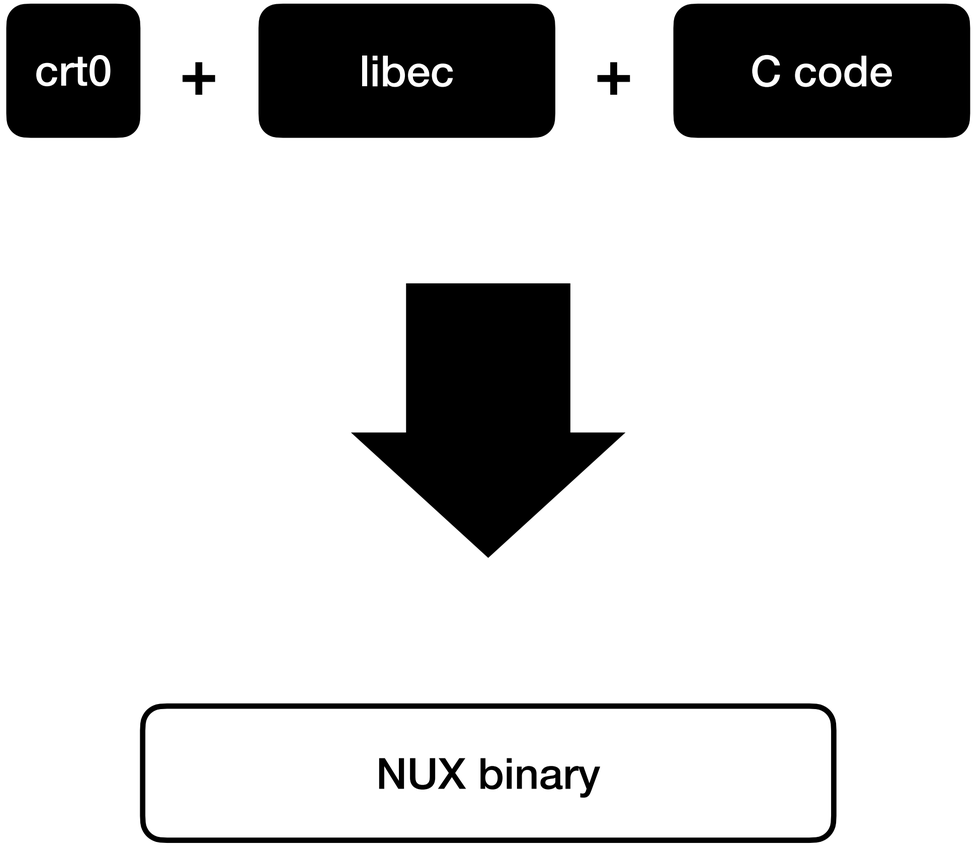
- For user-space code, the NUX framework provides *libnux_user*
- *libnux_user* goal is to provide a common interface to issue syscalls.
- As loaded, NUX only has a single userspace binary loaded at boot time.
- Kernel can implement its own mechanism to load additional user programs.



The NUX kernel framework.

libec: the unsung hero of NUX.

- **NUX provides its own *libc to create binaries.***
- ***libec* (lib embedded-C) is a simple, minimal libc.**
  - **based on NetBSD libc for ease of porting**
  - **strict adherence to C-standard *not* a goal.**
  - **limited to functions deemed useful for kernels and small binaries.**

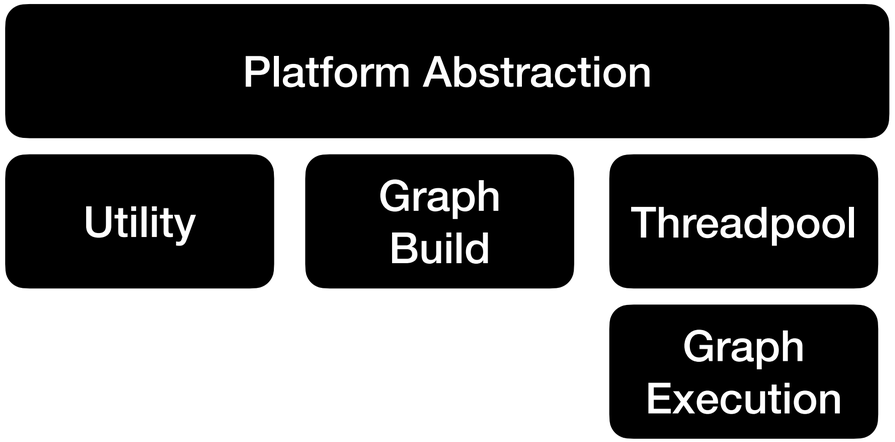


Part II: Running GGML.

Running GGML.

Architecture of a minimal GGML setup.

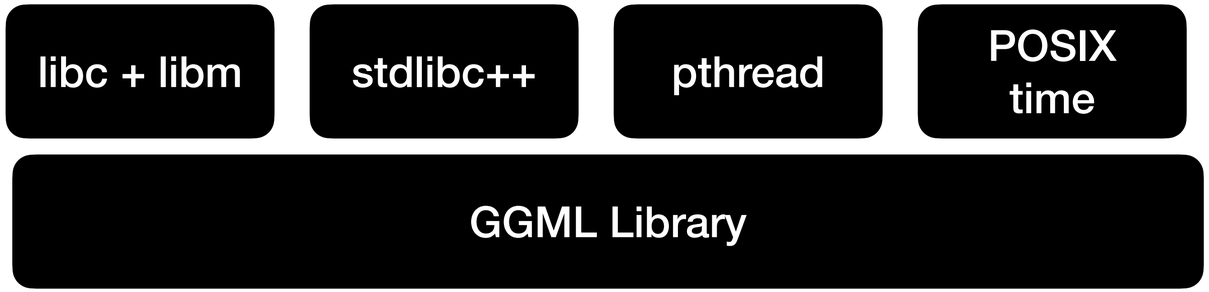
- A minimal library build has:
  - Platform abstraction
    - memory mgmt, time, etc.
  - Threadpool implementation
  - Computation Graph construction
  - Interpreter/VM
    - executes graph ops on threadpool.
  - Utility functions (including File I/O)



Running GGML.

Software dependencies.

- At minimum GGML requires
  - C++ runtime.
  - Small subset of C++ standard library.
    - vectors, iterators, etc.
  - A fairly complete libc (*qsort*, *malloc/free, file I/O)*
  - libm for floating point maths.
  - pthreads
  - Some POSIX functionalities (time)

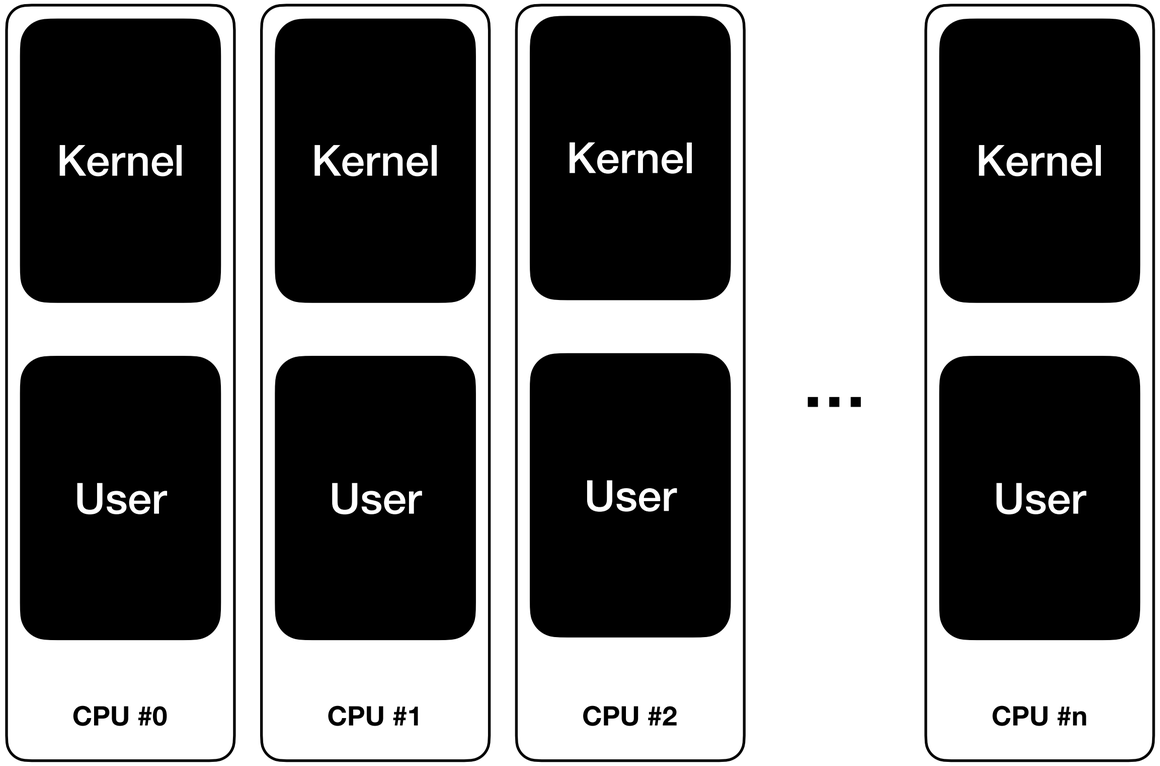


Part III: Porting GGML to NUX.

Porting GGML.

Model of a NUX system.

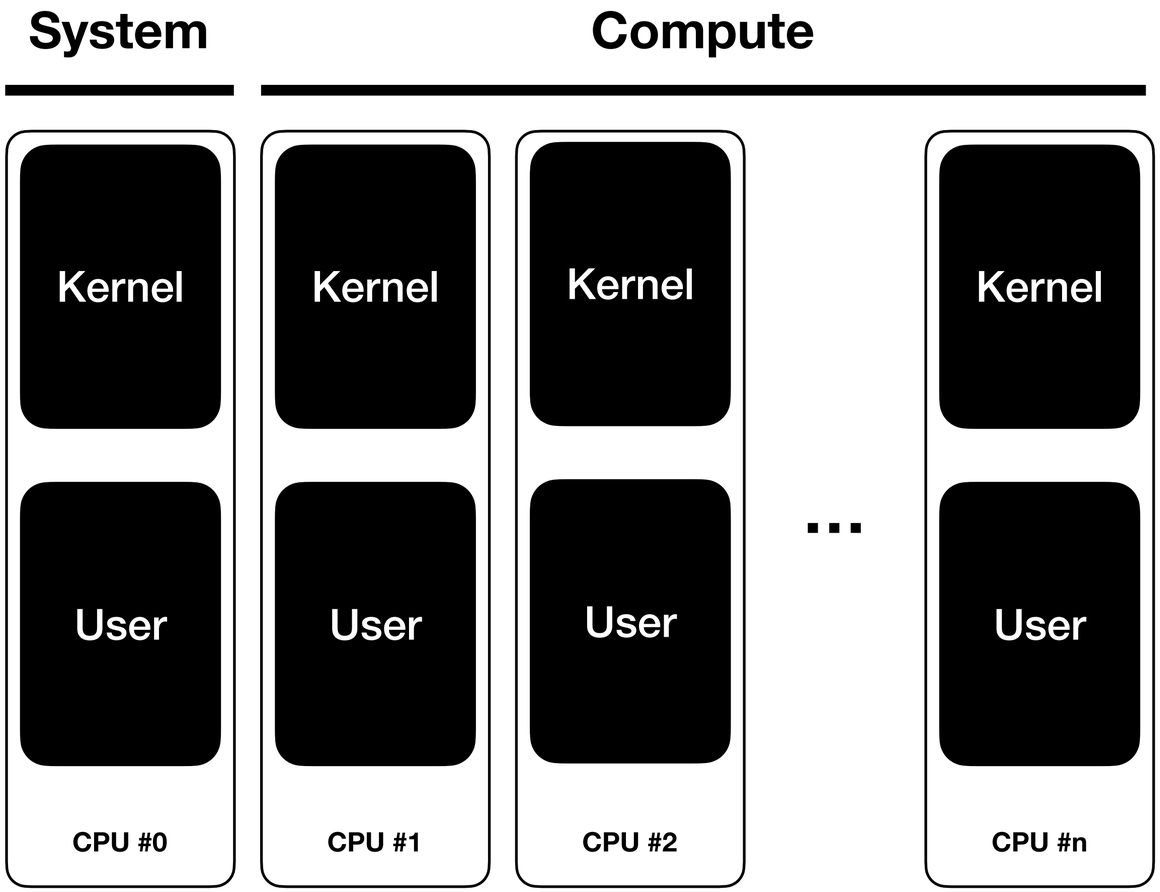
- At boot, each CPU has the same Kernel/User image.
- Images are created by APXH and shared amongst all CPUs.
- Code can of course be specialised per-CPU.



Porting GGML.

Architectural choices for the port.

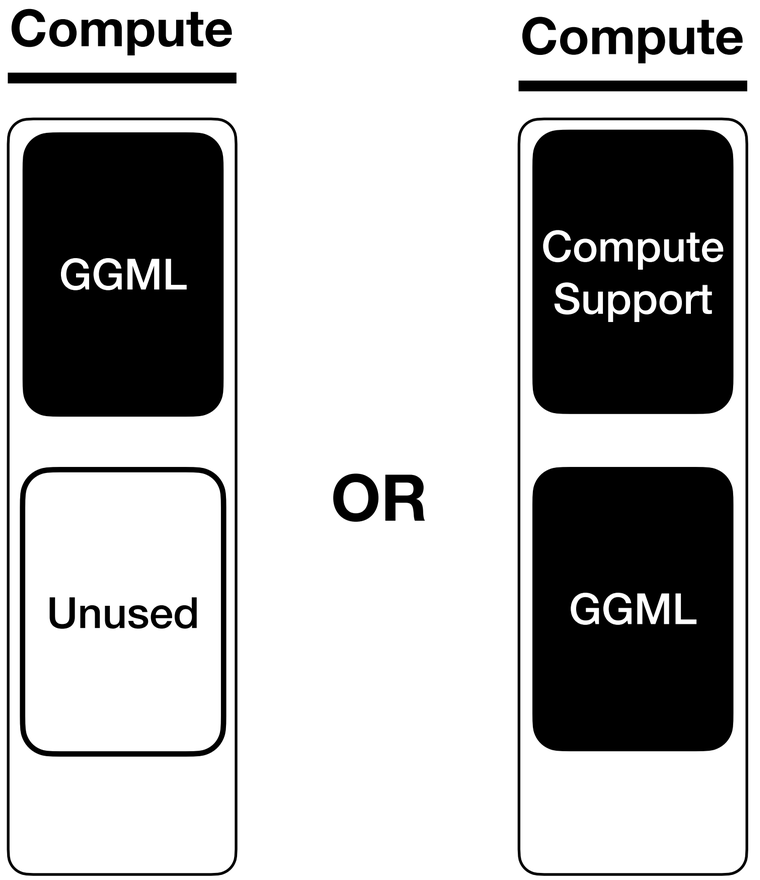
- **NUX layout is incredibly flexible.**
- **Goal is a compute platform for GGML.**
- **Dedicating entire CPUs to compute threads allows them to run without scheduling issues.**
- **System should communicate with external world so at least one CPU should be left to implement a minimal OS — *unikernels anyone?***



Porting GGML.

How to structure GGML in a Compute CPU.

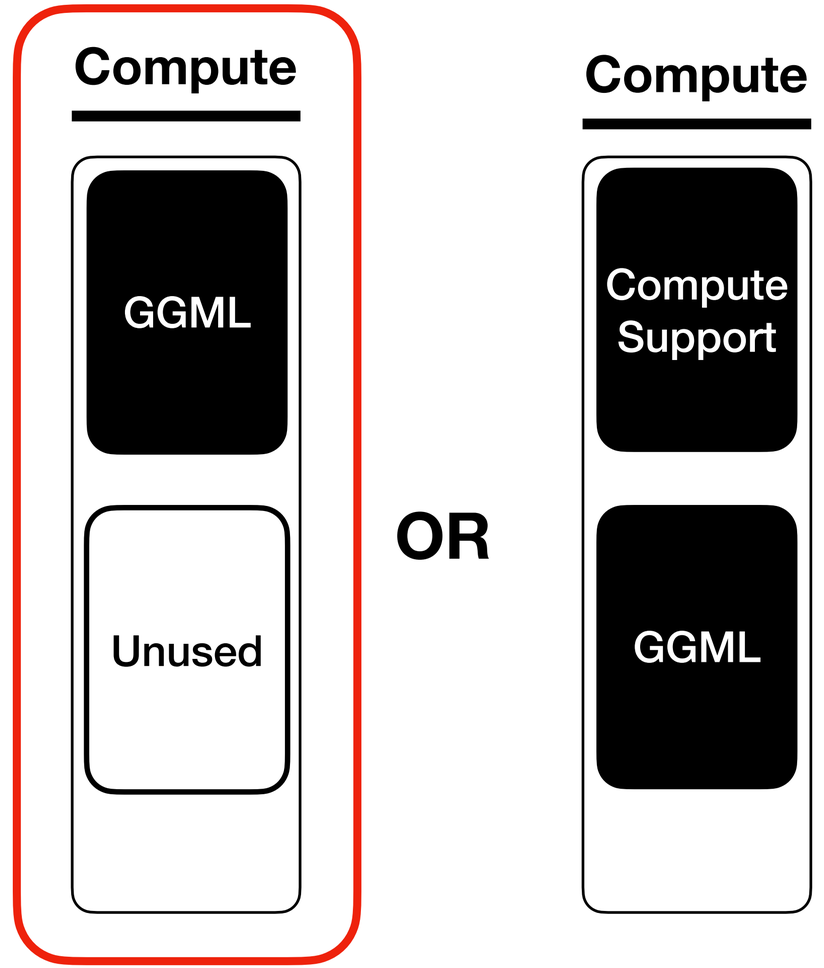
- GGML graph compute should run in kernel or in user space?
- GGML in kernel (*bare metal*):
  - GGML runs uninterrupted
  - No syscalls latency for system operations
  - High-level code to be ported in an usually minimal environment
- GGML in userspace:
  - Code is separated in its own address space
  - Easier to port libraries (e.g., using *newlib* instead of *libec*)
  - Potential latency by interrupts interrupting the compute and need for syscalls



Porting GGML.

How to structure GGML in a Compute CPU.

- GGML graph compute should run in kernel or in user space?
- GGML in kernel (*bare metal*):
  - GGML runs uninterrupted
  - No syscalls latency for system operations
  - High-level code to be ported in an usually minimal environment
- GGML in userspace:
  - Code is separated in its own address space
  - Easier to port libraries (e.g., using *newlib* instead of *libec*)
  - Potential latency by interrupts interrupting the compute and need for syscalls

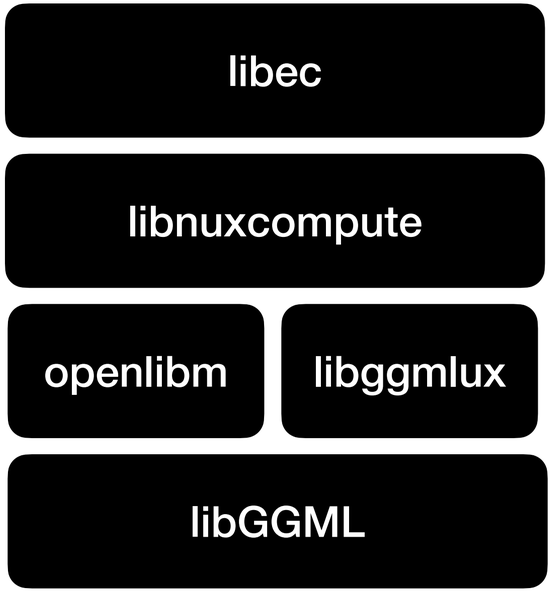


Solution Implemented

Porting GGML.

What does GGML in Kernel looks like?

- *libnuxcompute*: library for managing CPUs for uniterrupted computation.
- *openlibm*: [Julia Project] a library that implements libm in an extremely portable way.
- *libggmlux*: Where all the dirty work is done:
  - C++ runtime
  - libc extensions to libec
  - stdlibc++ minimal implementation
  - pthread to nuxcompute mapping
  - ggml_time to libnux mapping



Porting GGML.

Putting it all together.

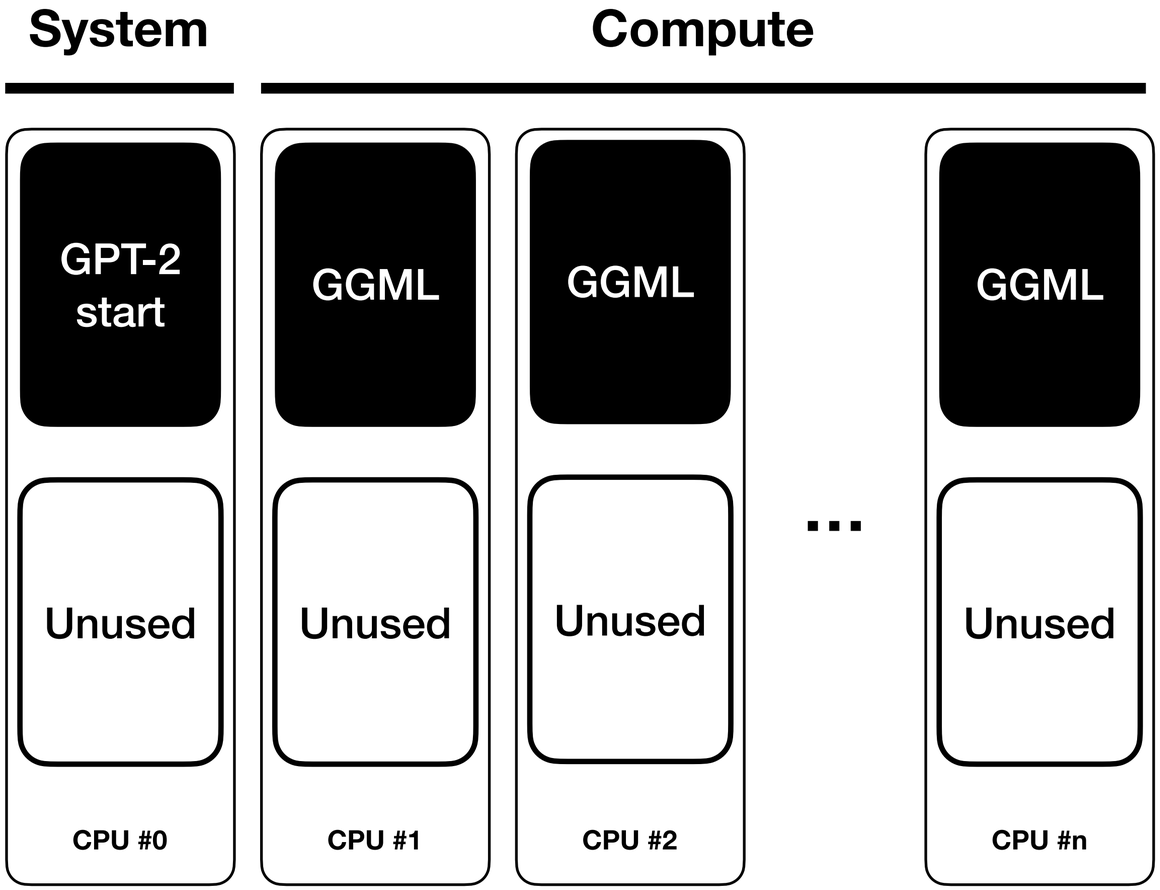
Code implementing this at

https://github.com/glguida/blasbare

Name derives from earlier experiments of porting BLAS to NUX

Prototype, *not* production ready!

Can run a simple GPT-2 model derived from GGML's examples/gpt-2



Part IV: Final Considerations

Final Considerations

- Porting GGML to constrained and embedded systems is much easier than expected!
  - Architecture is sane.
- Some simple but possibly intrusive modifications to GGML codebase might ease efforts similar to this:
  - Separate different sections (e.g., file I/O) in different files. So their compilation can be more easily disabled.
  - Allow reimplementing a separate threadpool implementation (e.g., static and not depending from pthread) without resorting to #ifdef's.
  - Platform abstraction could be modified to allow to implement GGML on different platform without resorting to modifying ggml.c or simulating a POSIX/pthread interface.

Thank you!

For more information:

https://tlbflush.org

https://nux.tlbflush.org

https://github.com/glguida/blasbare