School of Computer Science & Engineering

Trustworthy Systems Group

In and Around LionsOS

Ivan Velickovic

i.velickovic@unsw.edu.au

Outline

Outline

- Overall design of LionsOS and it’s fundamental parts.

Outline

- Overall design of LionsOS and it’s fundamental parts.
- What progress have we made in the past year?

Outline

- Overall design of LionsOS and it’s fundamental parts.
- What progress have we made in the past year?
- What do we have now?

Outline

- Overall design of LionsOS and it’s fundamental parts.
- What progress have we made in the past year?
- What do we have now?
- ‘Unique’ problems and how we’re solving them.

Outline

- Overall design of LionsOS and it’s fundamental parts.
- What progress have we made in the past year?
- What do we have now?
- ‘Unique’ problems and how we’re solving them.
- Demo.

LionsOS design

- You start with a clean slate. The system designer chooses what components, I/O systems, client programs to have.

Microkit

Microkit

- Programming model is event based.

Microkit

- Programming model is event based.
- Asynchronous or synchronous communication.

Microkit

- Programming model is event based.
- Asynchronous or synchronous communication.
- Microkit lowers the bar to seL4 but does not include drivers or other higher-level OS services.

Microkit - status

Microkit - status

- Multiple releases in the past year
  - 1.3.0 (Jul’24), 1.4.0 (Aug’24), 1.4.1 (Aug’24).

Microkit - status

- Multiple releases in the past year
  - 1.3.0 (Jul'24), 1.4.0 (Aug'24), 1.4.1 (Aug'24).
- Most effort around implementing/upstreaming requested features:
  - Hypervisor/Virtual Machine support.
  - Architecture support (RISC-V merged, x86 still in-progress).
  - More platform support.
  - Various fixes, quality-of-life improvements.

I/O – seL4 Device Driver Framework

I/O – seL4 Device Driver Framework

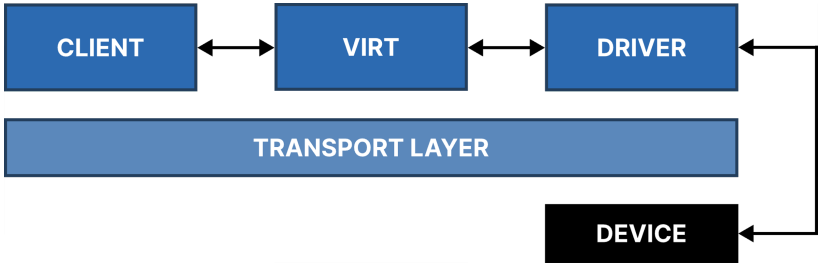
- Designed for performance, modularity and simplicity.

I/O – seL4 Device Driver Framework

- Designed for performance, modularity and simplicity.
- Standard protocol/interface between components for each device class.
  - LionsOS components (e.g file systems) are based off these interfaces.

I/O – seL4 Device Driver Framework

- Designed for performance, modularity and simplicity.
- Standard protocol/interface between components for each device class.
  - LionsOS components (e.g file systems) are based off these interfaces.



I/O - sDDF status

I/O - sDDF status

- Last year we had:
  - Network
  - Timer

I/O - sDDF status

- Last year we had:
  - Network
  - Timer
- In the past year we've added:
  - Block (e.g MMC)
  - UART (non-DMA)
  - I^2C
  - Audio

I/O - sDDF status

- Last year we had:
  - Network
  - Timer
- In the past year we've added:
  - Block (e.g MMC)
  - UART (non-DMA)
  - I^2C
  - Audio
- And have made various progress on:
  - Graphics (2D)
  - GPIO
  - Pinmux
  - Clock

I/O

I/O

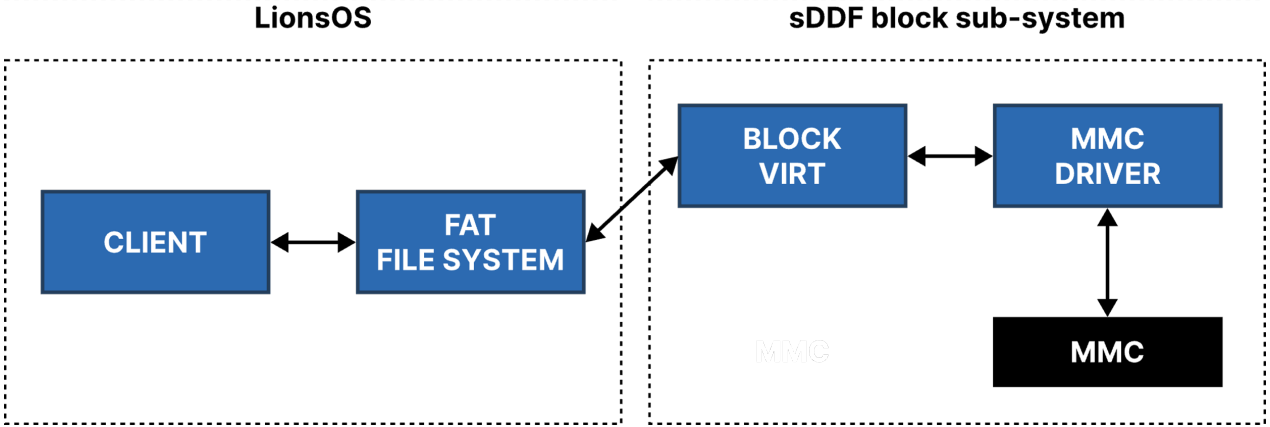
- In general, anything to the left of the virtualiser is not part of sDDF and would be considered part of the OS.
  - For example, file systems are a ‘client’ of the block sub-system.

I/O

- In general, anything to the left of the virtualiser is not part of sDDF and would be considered part of the OS.
  - For example, file systems are a ‘client’ of the block sub-system.
- Primarily have been focused on sDDF I/O design.
  - Starting to design higher-level I/O layers, such as file systems.

I/O

- In general, anything to the left of the virtualiser is not part of sDDF and would be considered part of the OS.
  - For example, file systems are a ‘client’ of the block sub-system.
- Primarily have been focused on sDDF I/O design.
  - Starting to design higher-level I/O layers, such as file systems.



Virtualisation

- Main updates are that:
  - RISC-V support is now working.
    - Although it depends on seL4 changes that are yet to be mainlined.
    - Plan to create an RFC and start upstreaming the changes.
  - virtIO devices.
    - We now have virtIO implementations for console, block, network and audio devices.
  - Multi-core guests are being worked on as well.

Static architecture, but not a static system

Static architecture, but not a static system

- When people hear ‘static architecture’, they may think you setup the system and then you can’t touch it anymore.

Static architecture, but not a static system

- When people hear ‘static architecture’, they may think you setup the system and then you can’t touch it anymore.
- But, there is a spectrum.

Static architecture, but not a static system

- When people hear ‘static architecture’, they may think you setup the system and then you can’t touch it anymore.
- But, there is a spectrum.
- Two approaches to address this:
  - Template protection domains.
  - Introducing more dynamism into sDDF, specifically with hot-plugging.
    - Able to eject/insert MMC cards in a live system.
    - Yet to apply hot-plugging to other device classes.

What have we made so far?

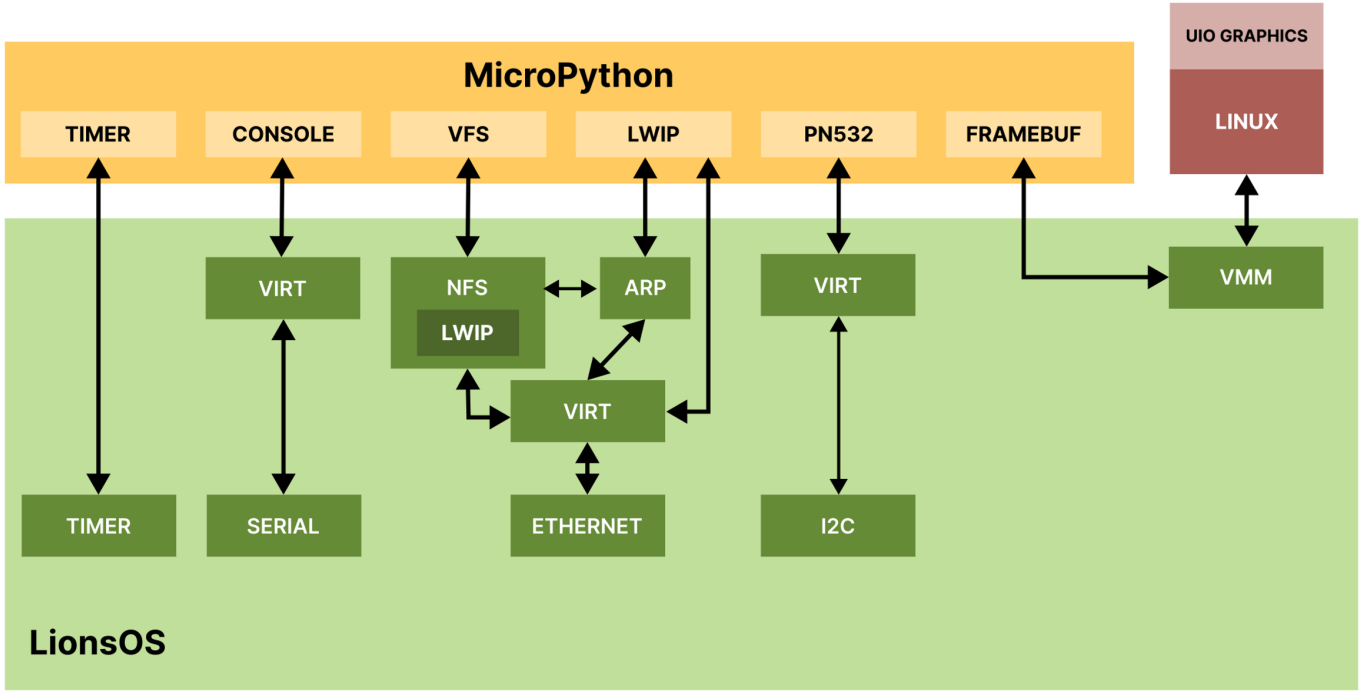
What have we made so far?

- Reference system, aka Kitty.

What have we made so far?

- Reference system, aka Kitty.
- Web server, serving the seL4 website (https://sel4.systems).

Reference system (Kitty)



Lessons learned

Lessons learned

- Friction is too high, need to still lower the barrier to entry.
  - Combining various components and I/O sub-systems in a single system does not scale well.

Lessons learned

- Friction is too high, need to still lower the barrier to entry.
  - Combining various components and I/O sub-systems in a single system does not scale well.
- With a static architecture, some of the run-time complexity in a typical OS shifts to build-time in LionsOS.

Lowering friction – driver VMs

Lowering friction – driver VMs

- Re-use drivers from Linux via a virtual machine.
  - ...where the concessions to performance and security are acceptable.
  - Helps prototype without writing a bunch of drivers.

Lowering friction – driver VMs

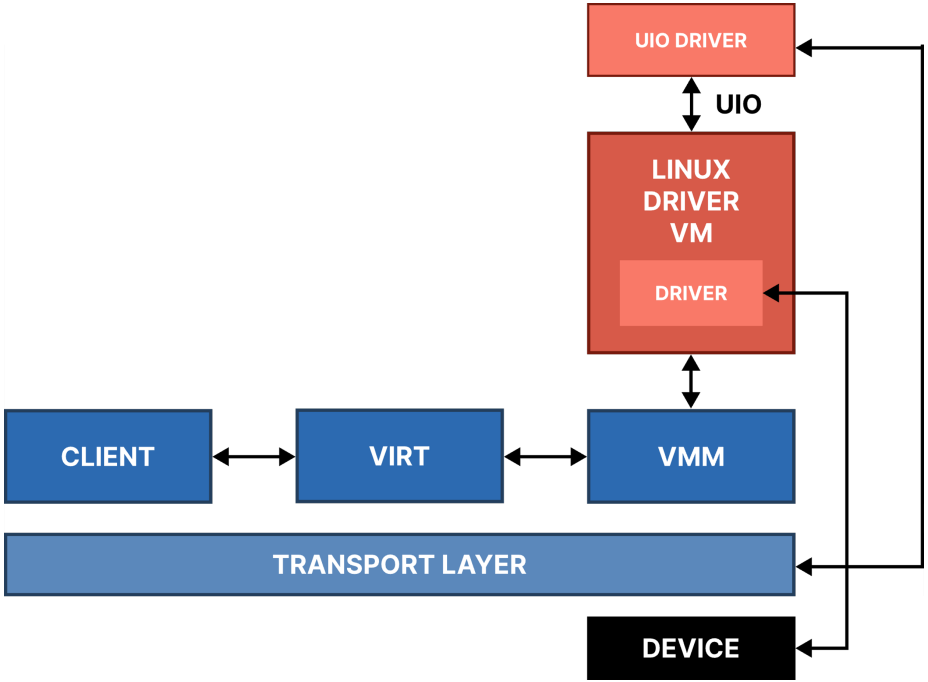
- Re-use drivers from Linux via a virtual machine.
  - ...where the concessions to performance and security are acceptable.
  - Helps prototype without writing a bunch of drivers.
- Convert sDDF protocol to Linux system calls/APIs.

Lowering friction – driver VMs

- Re-use drivers from Linux via a virtual machine.
  - ...where the concessions to performance and security are acceptable.
  - Helps prototype without writing a bunch of drivers.
- Convert sDDF protocol to Linux system calls/APIs.
- Currently, UIO drivers exist for the block and audio device classes.

Lowering friction – driver VMs

- Re-use drivers from Linux via a virtual machine.
  - ...where the concessions to performance and security are acceptable.
  - Helps prototype without writing a bunch of drivers.
- Convert sDDF protocol to Linux system calls/APIs.
- Currently, UIO drivers exist for the block and audio device classes.



Lowering friction – driver VMs

Lowering friction – driver VMs

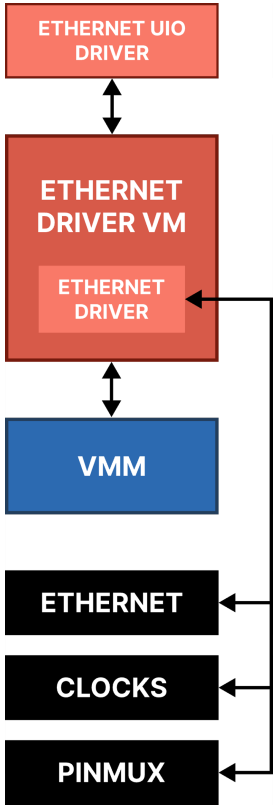
- Using multiple driver VMs presents a problem.

Lowering friction – driver VMs

- Using multiple driver VMs presents a problem.
- Linux will attempt to initialise clocks/pinmux for the passed through device.

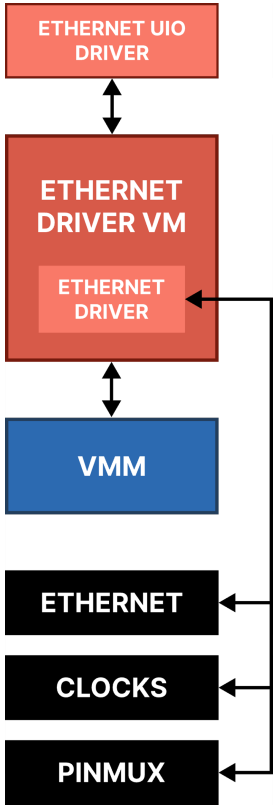
Lowering friction – driver VMs

- Using multiple driver VMs presents a problem.
- Linux will attempt to initialise clocks/pinmux for the passed through device.



Lowering friction – driver VMs

- Using multiple driver VMs presents a problem.
- Linux will attempt to initialise clocks/pinmux for the passed through device.
- Trap and emulate access to clocks/pinmux devices and rely on native drivers instead.



Lowering friction – build time tooling

Lowering friction – build time tooling

- Two aspects:
  - Managing the System Description Format (SDF) file to give to the Microkit tool.
  - Information flow from the design of the system to component code.

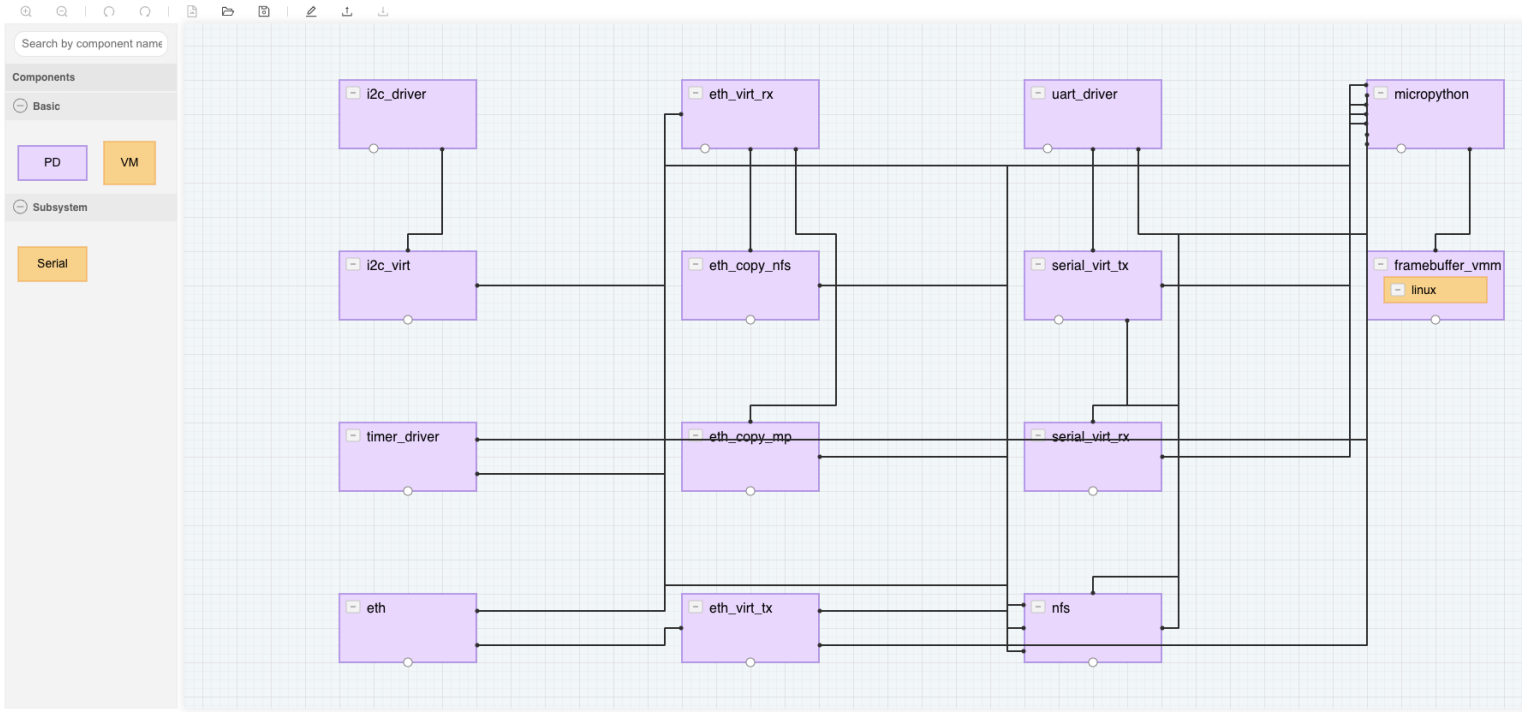
Lowering friction – build time tooling

- Two aspects:
  - Managing the System Description Format (SDF) file to give to the Microkit tool.
  - Information flow from the design of the system to component code.
- Working on tools/libraries to allow creating LionsOS systems based on higher-level description.

Lowering friction – build time tooling

- Two aspects:
  - Managing the System Description Format (SDF) file to give to the Microkit tool.
  - Information flow from the design of the system to component code.
- Working on tools/libraries to allow creating LionsOS systems based on higher-level description.
- Starting to receive internal use, still experimental.

System visualiser/editor



Lowering friction – legacy layer

Lowering friction – legacy layer

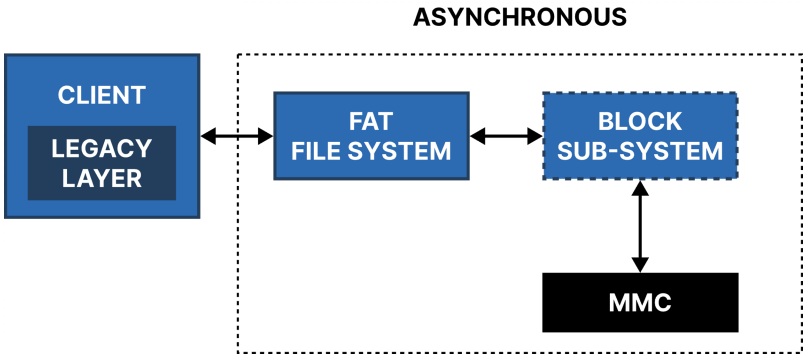
- Legacy and off-the-shelf libraries tend to expect some kind of ‘POSIX-like’ interface.
  - For example, our network file system uses an off-the-shelf library which expect certain POSIX APIs.

Lowering friction – legacy layer

- Legacy and off-the-shelf libraries tend to expect some kind of ‘POSIX-like’ interface.
  - For example, our network file system uses an off-the-shelf library which expect certain POSIX APIs.
- ‘POSIX-like’ vs full POSIX
  - Support for blocking I/O that people are used to programming with such as `read()`, `write()`, `send()`.
  - Unlikely to have versions of `fork()` or `exec()`.

Lowering friction – legacy layer

- Legacy and off-the-shelf libraries tend to expect some kind of ‘POSIX-like’ interface.
  - For example, our network file system uses an off-the-shelf library which expect certain POSIX APIs.
- ‘POSIX-like’ vs full POSIX
  - Support for blocking I/O that people are used to programming with such as `read()`, `write()`, `send()`.
  - Unlikely to have versions of `fork()` or `exec()`.

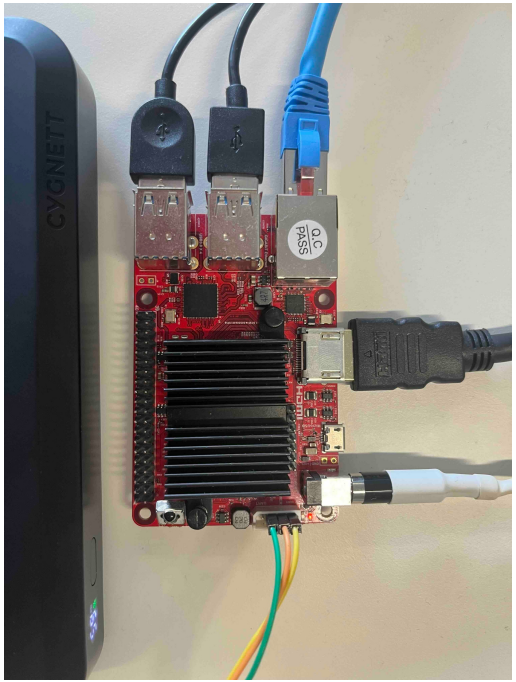


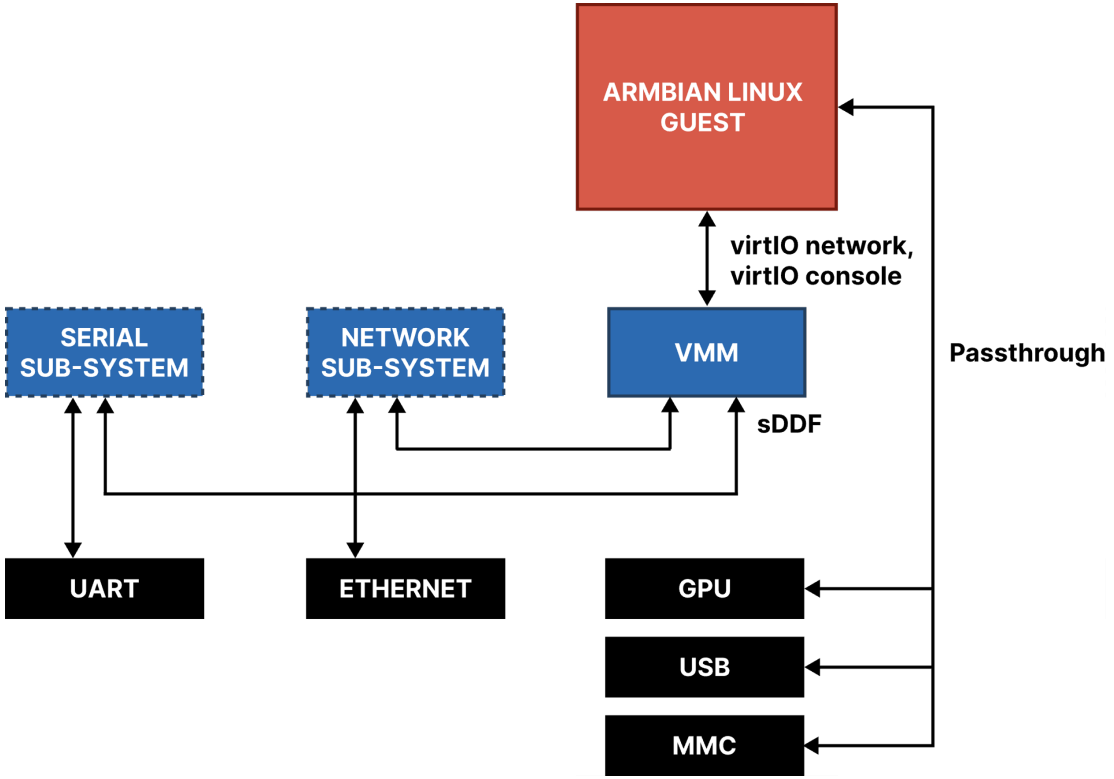
Demo

Demo

But, I lied

Demo system





Thanks! Questions?