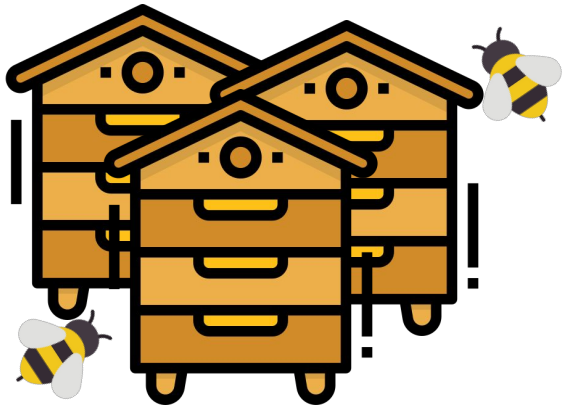
Apiary: An OS for the Modern FPGA

Katie Lim[1,2], Matthew Giordano[1], Irene Zhang[2], Baris Kasikci[1], Tom Anderson[1]

[1]University of Washington, [2]Microsoft Research



Outline

Motivation

Apiary Goals

Use-case Study

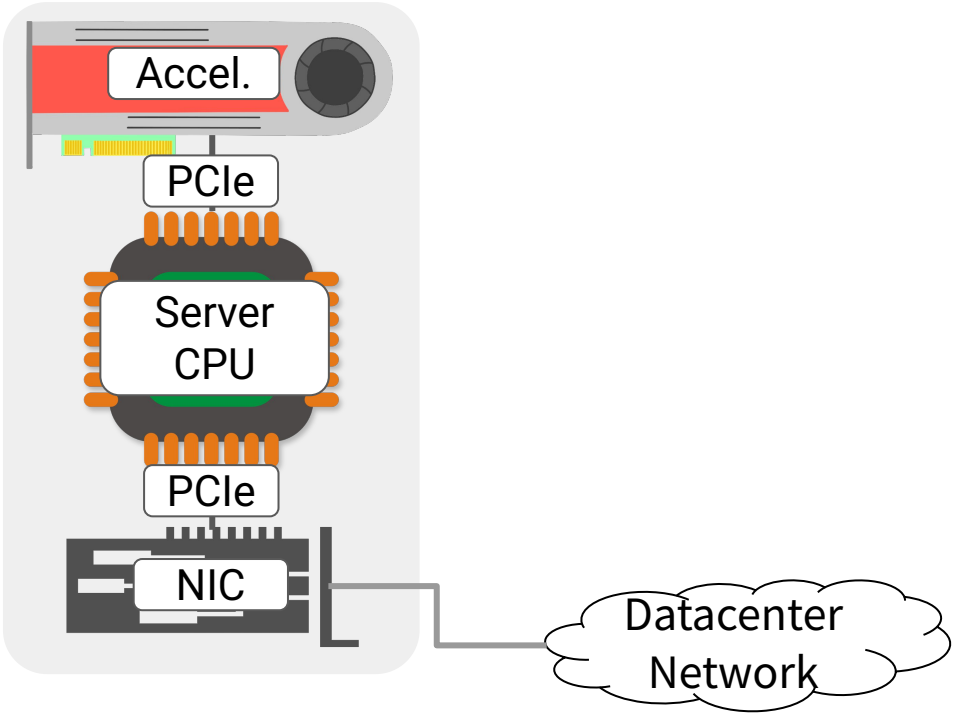
Apiary Architecture

Conclusion

How do you use FPGAs in a datacenter?

PCIe-attached, hosted by CPU

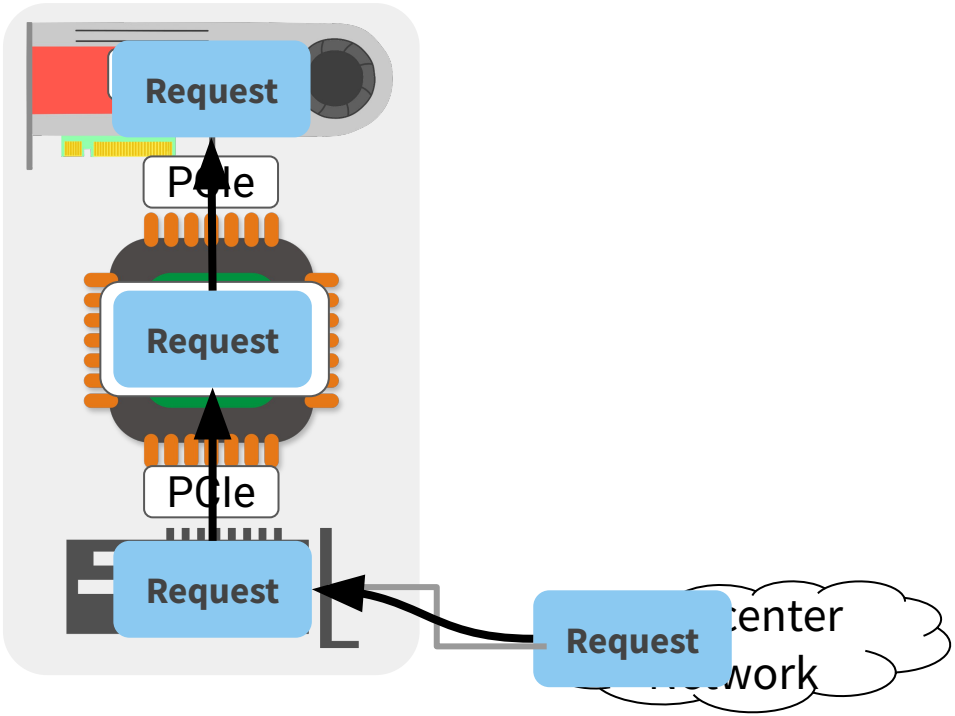
(e.g. AWS F1/F2 EC2)



How do you use FPGAs in a datacenter?

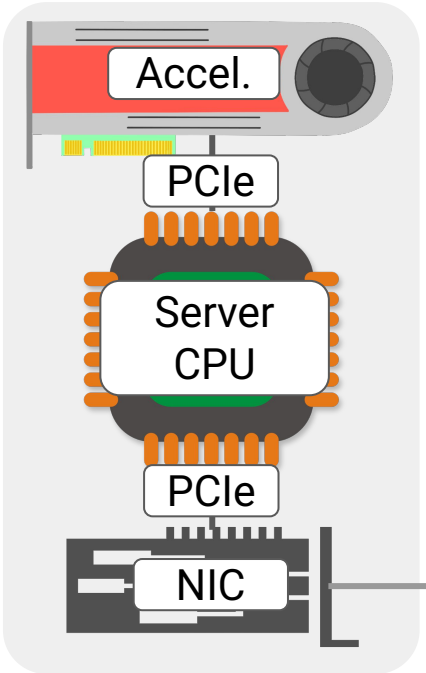
PCIe-attached, hosted by CPU

(e.g. AWS F1/F2 EC2)

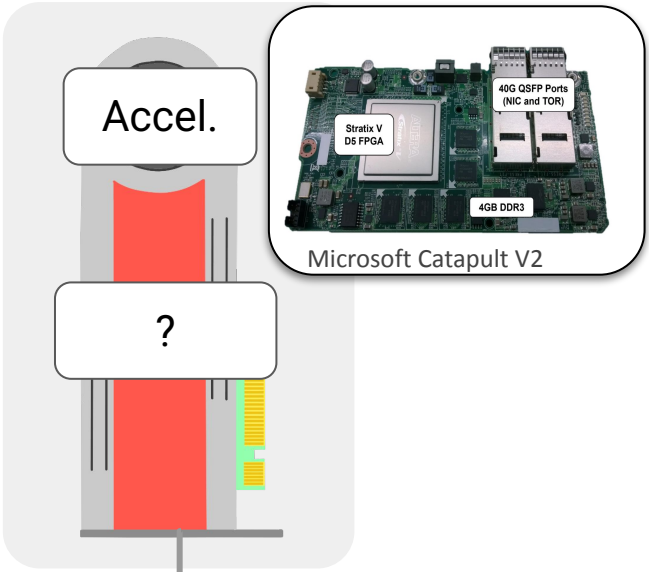


How do you use FPGAs in a datacenter?

PCIe-attached, hosted by CPU
(e.g. AWS F1/F2 EC2)



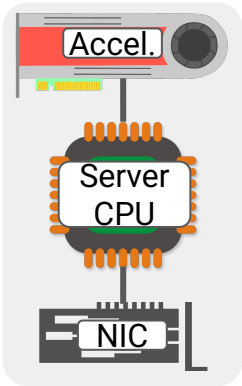
Direct-attached



Datacenter Network

How do you use FPGAs in a datacenter?

PCIe-attached, hosted by CPU



Pros:

✅ Full OS service support

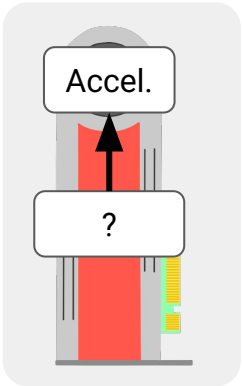
✅ Flexible CPU “fallback”

Cons:

❌ Higher and unpredictable latency

❌ “Wasting” CPU capacity

Direct-attached



Pros:

✅ Short, predictable path to accelerator

✅ Improved efficiency

Cons:

❌ No familiar OS abstractions

❌ Less flexible hardware infrastructure

Outline

Motivation

Apiary Goals

Use-case Study

Apiary Architecture

Conclusion

Apiary Goals

Design a system to provide OS features without software or CPU assistance

- **High-level interfaces:** common set of OS-like services for productivity and portability
- **Modularity:** applications and services can compose with each other and be easily added or removed
- **Isolation:** prevent unintended or malicious interactions between elements

Focus on hardware interactions **within** the FPGA, because of direct-attached

- Prior work assumes CPU-mediation and focuses on CPU-FPGA interactions

Apiary System Model

- Microkernel: all processing elements should be able to compose with each other via message passing over a generic interconnect
- Elements are both application accelerators and services
  - Application accelerators: video encoder, compression, ML, etc.
  - Services: TCP stack, memory allocation, storage, etc.
- Application accelerators may have multiple users or may misbehave

Questions to Ponder

How do elements communicate…

- …generically
- …scalably
- …safely

How do we create abstractions for…

- …concurrency
- …diverse application accelerators
- …I/O

What is our isolation model for…

- …memory
- …I/O
- …failures

Outline

Motivation

Apiary Goals

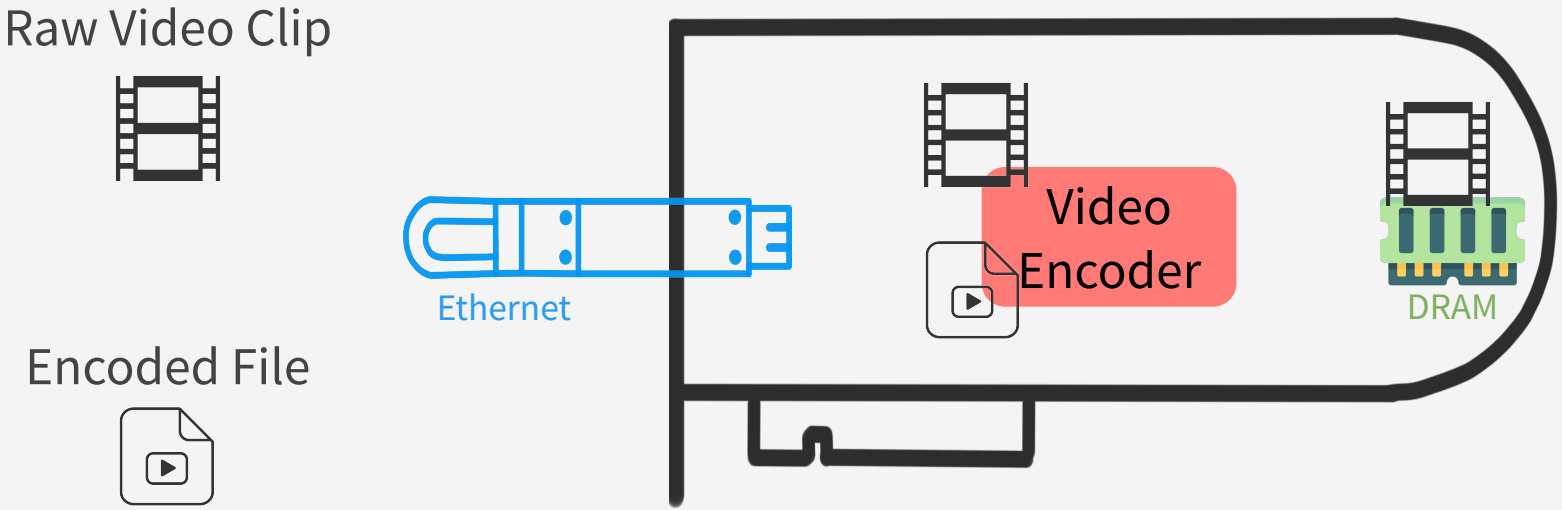
Use-case Study

Apiary Architecture

Conclusion

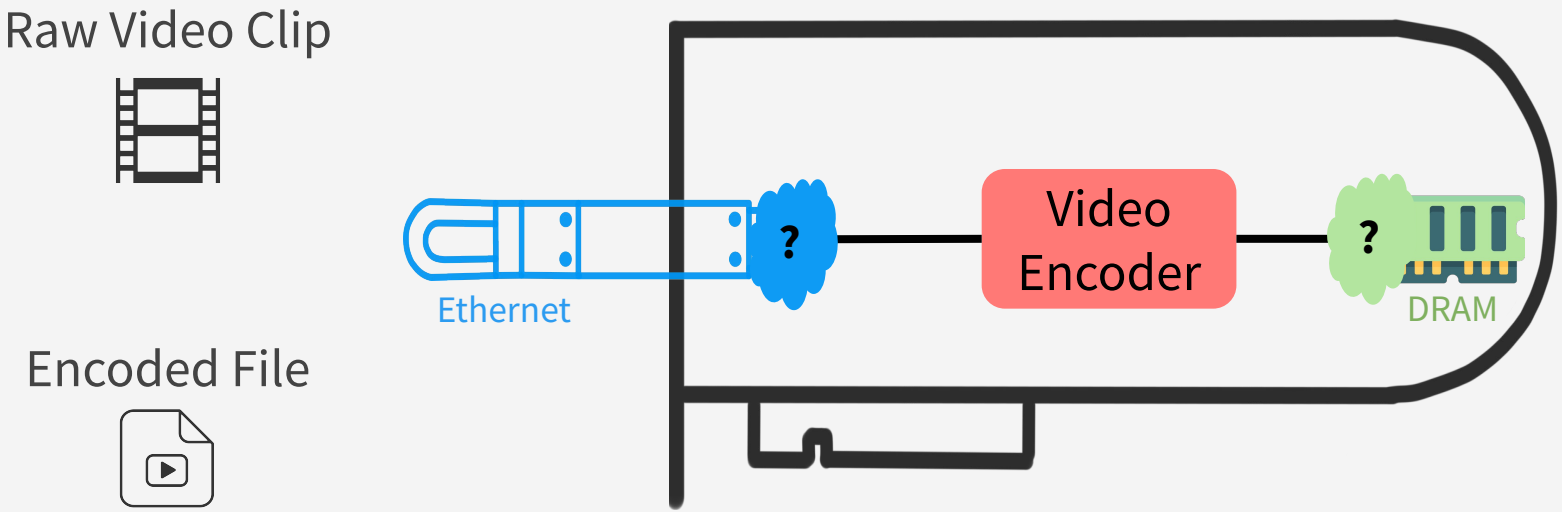
Getting Up and Running

- Imagine we're accelerating video encoding in a video processing pipeline



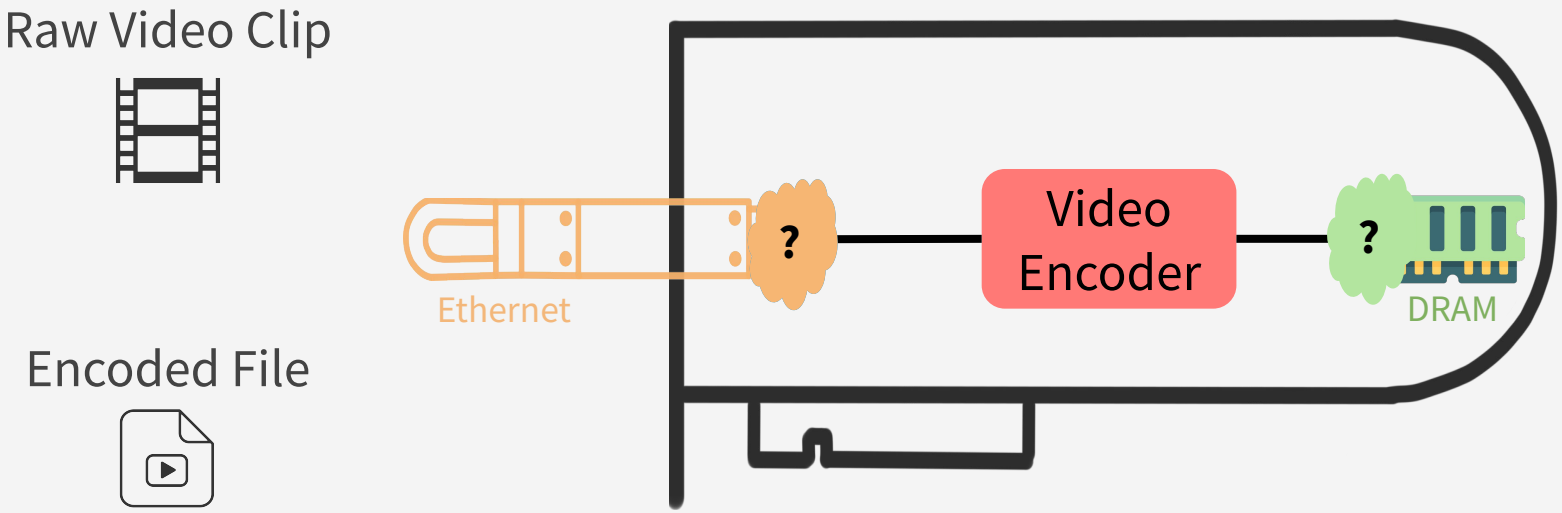
Getting Up and Running

- I/O interfaces are board-specific



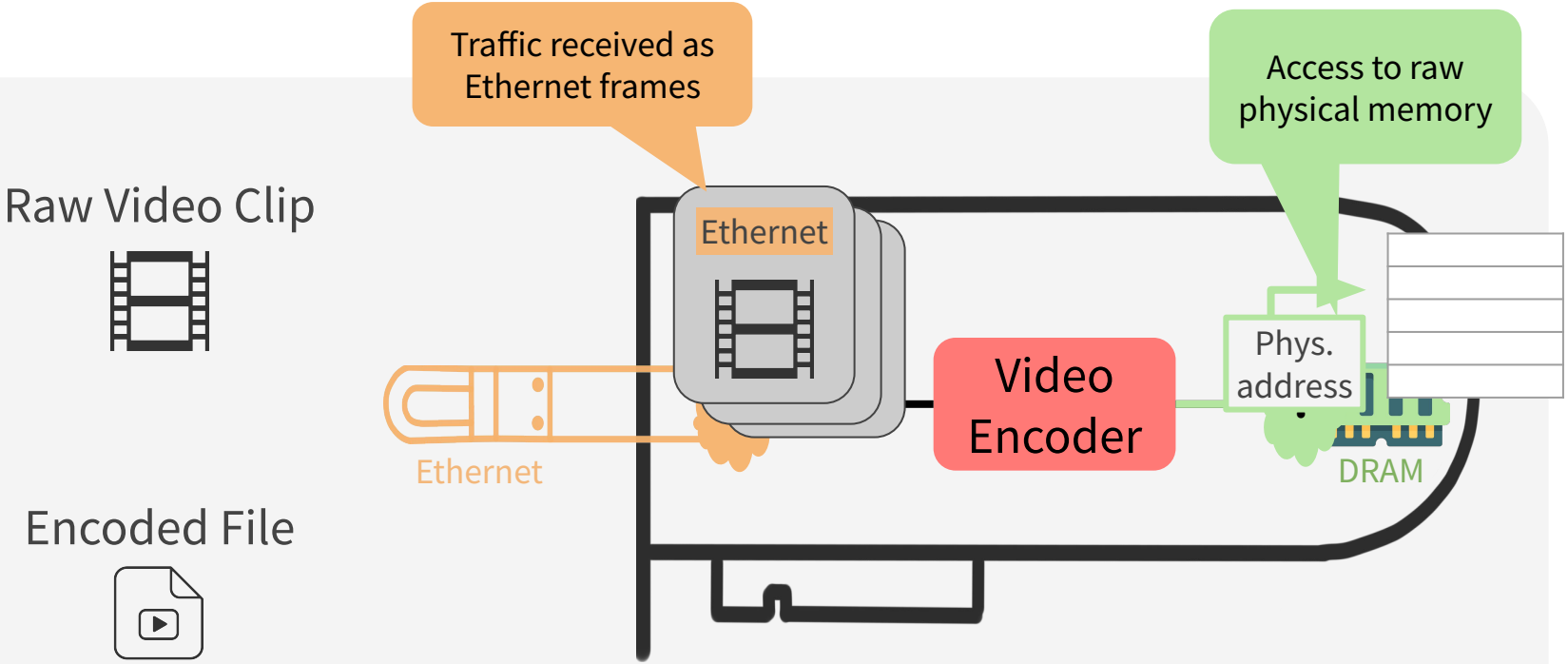
Getting Up and Running

- I/O interfaces are board-specific



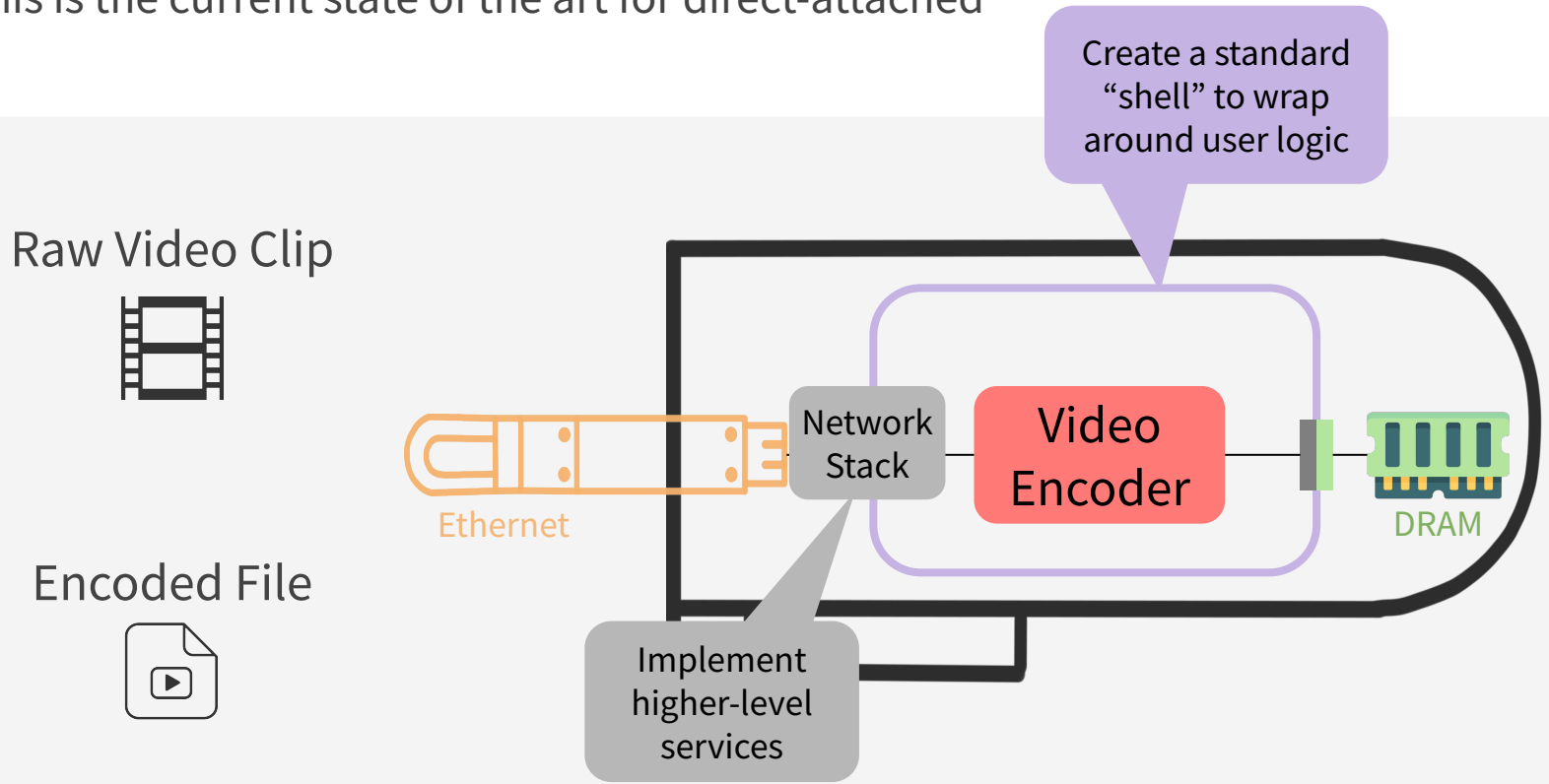
Getting Up and Running

- I/O is low-level



Getting Up and Running

- This is the current state of the art for direct-attached



Inter“process” communication

- **Network on chip (NoC):** Router-based hardware interconnect within a system on chip (SoC) that routes messages between processing elements
- Can use a variety of topologies, routing algorithms
  - Common for hardware efficiency: 2D mesh, lossless, static routing
- Beneficial for scalability, modularity

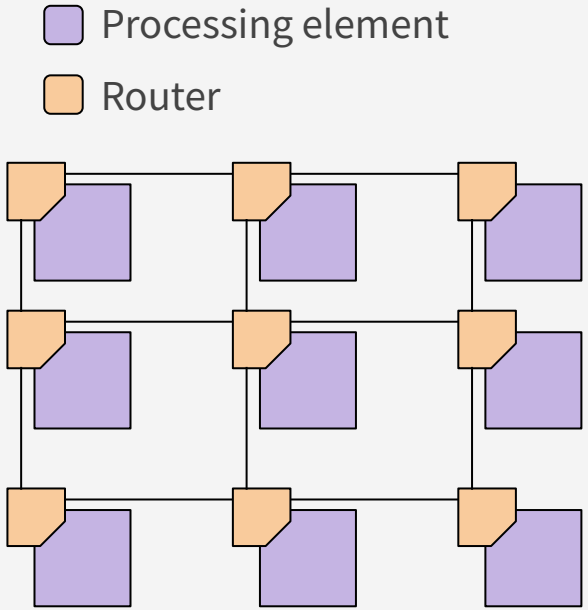
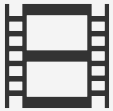


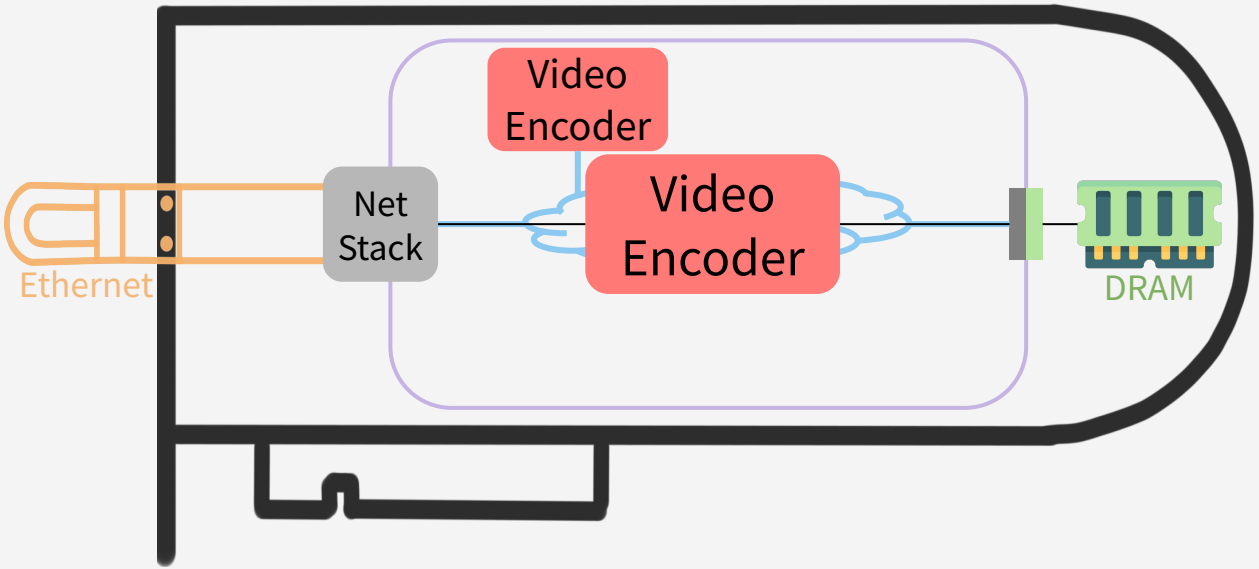
Diagram of a 2D-Mesh NoC

Adding Communication

Raw Video Clip

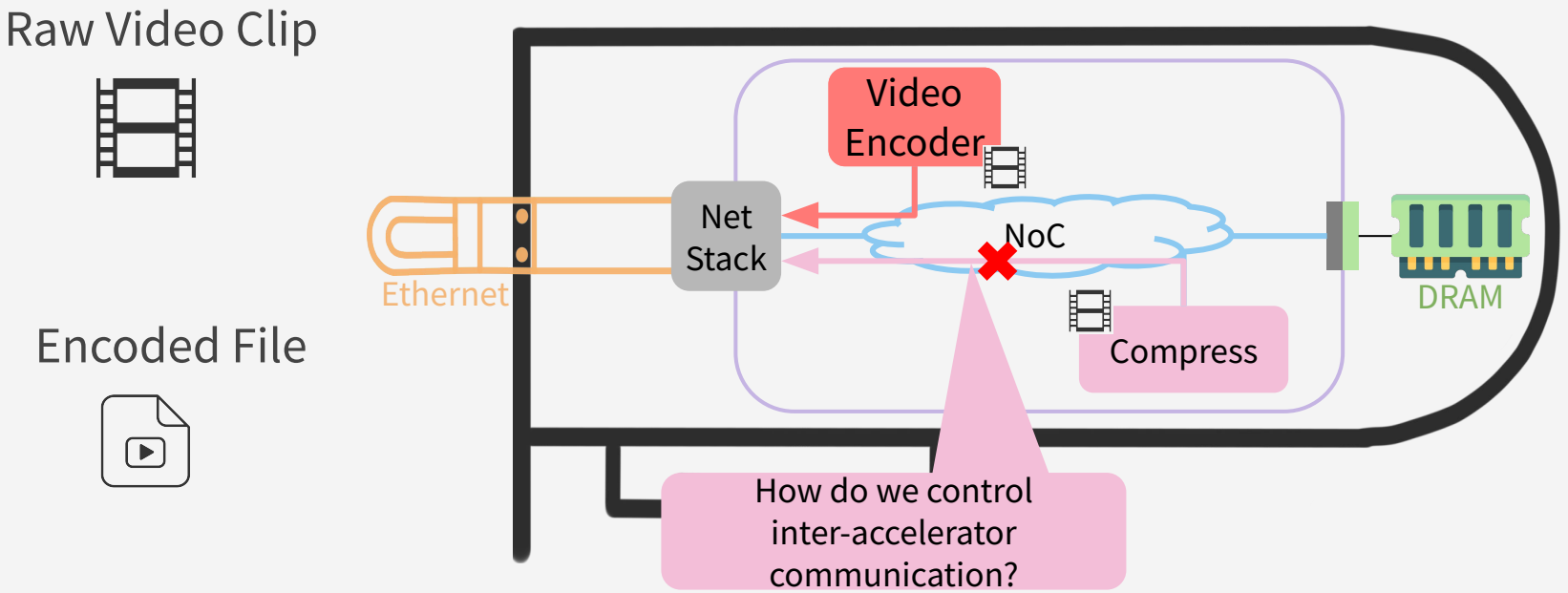


Encoded File

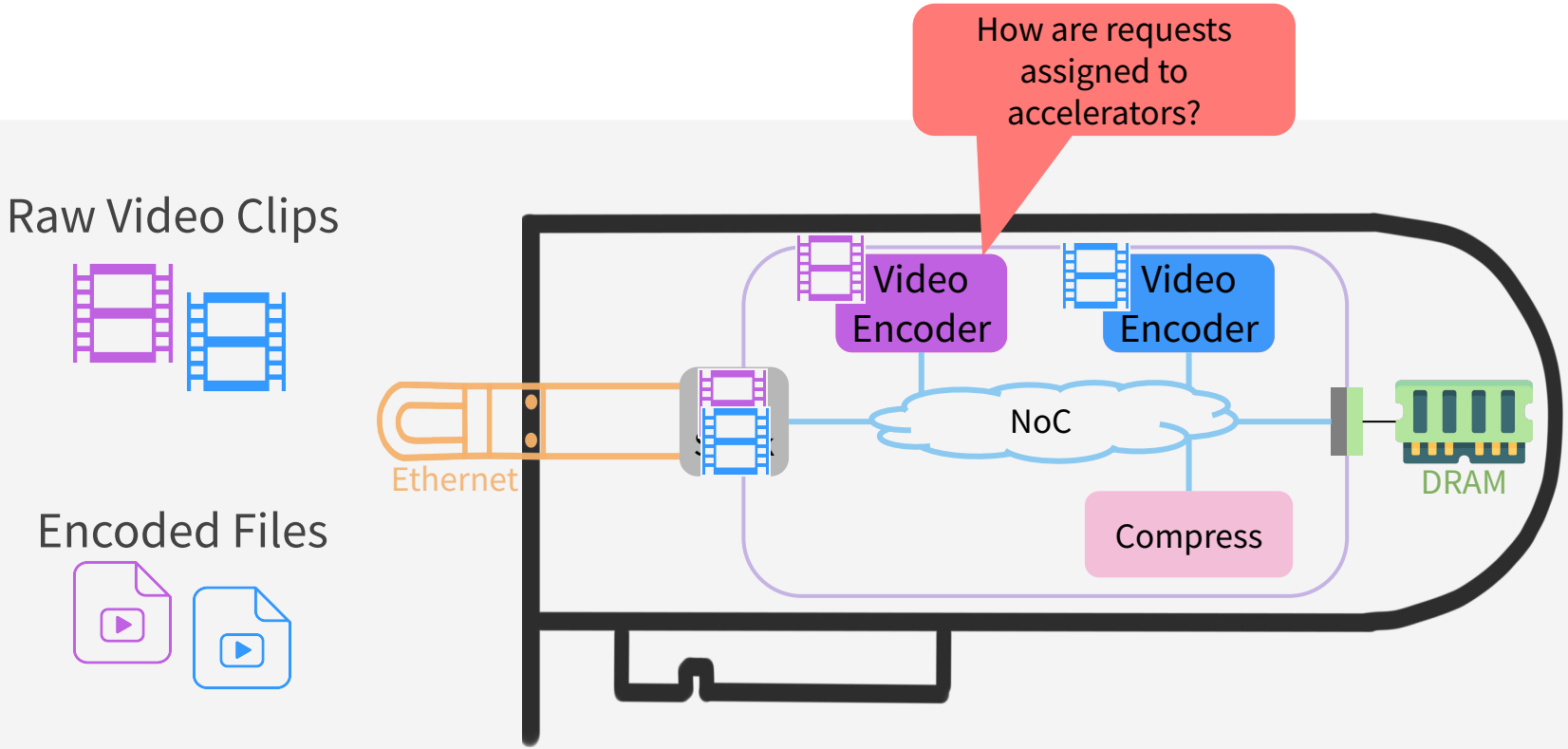


How do we prevent unintentional accelerator communication?

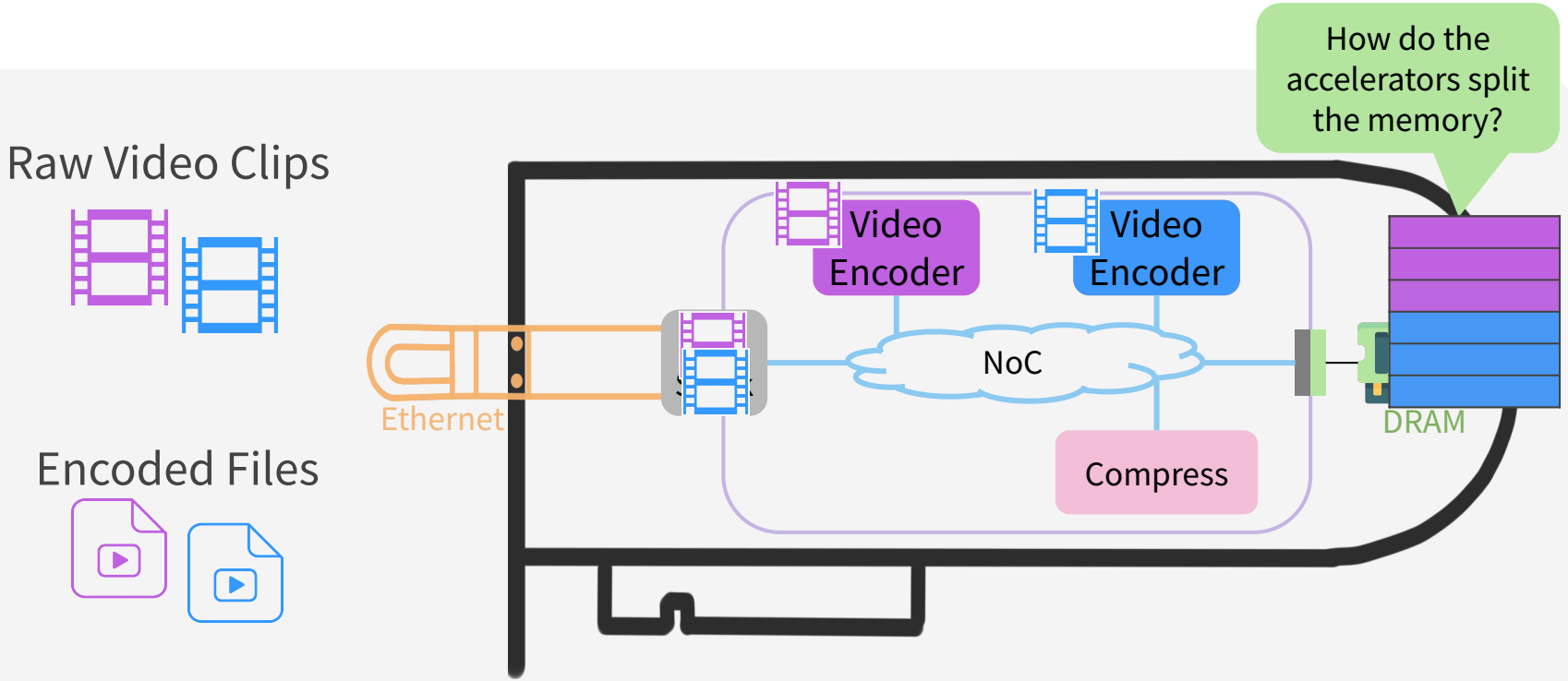
- Imagine introducing a third-party accelerator



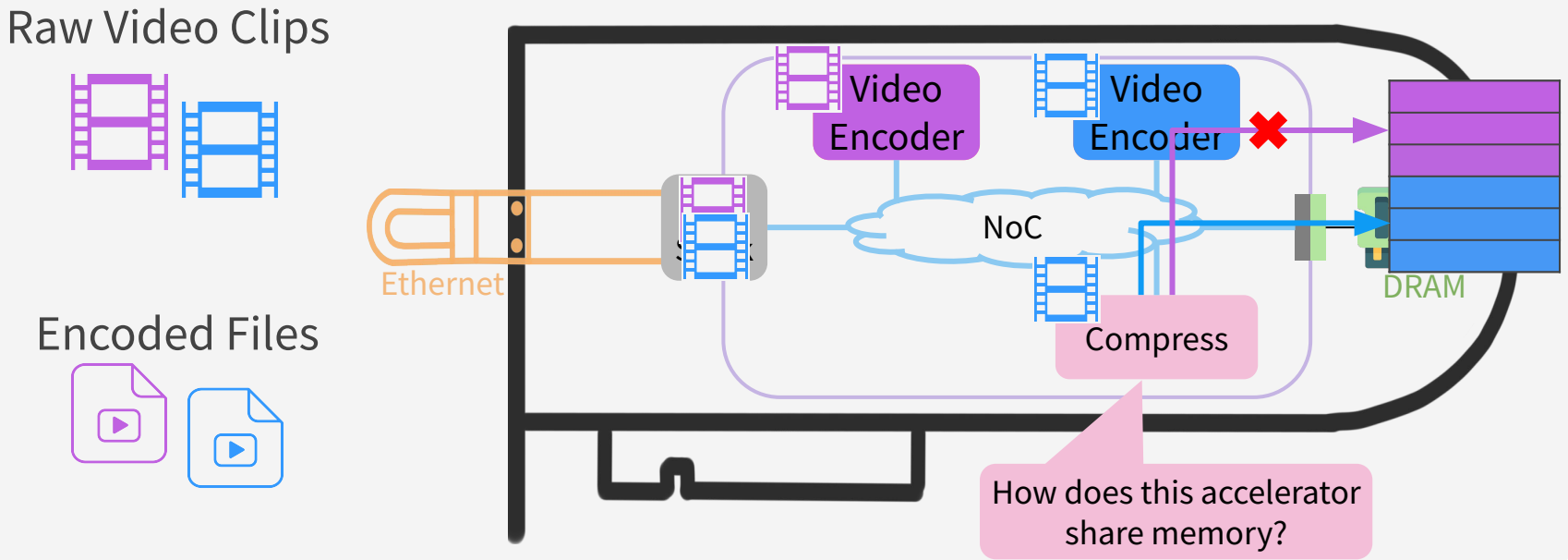
How do we handle more requests?



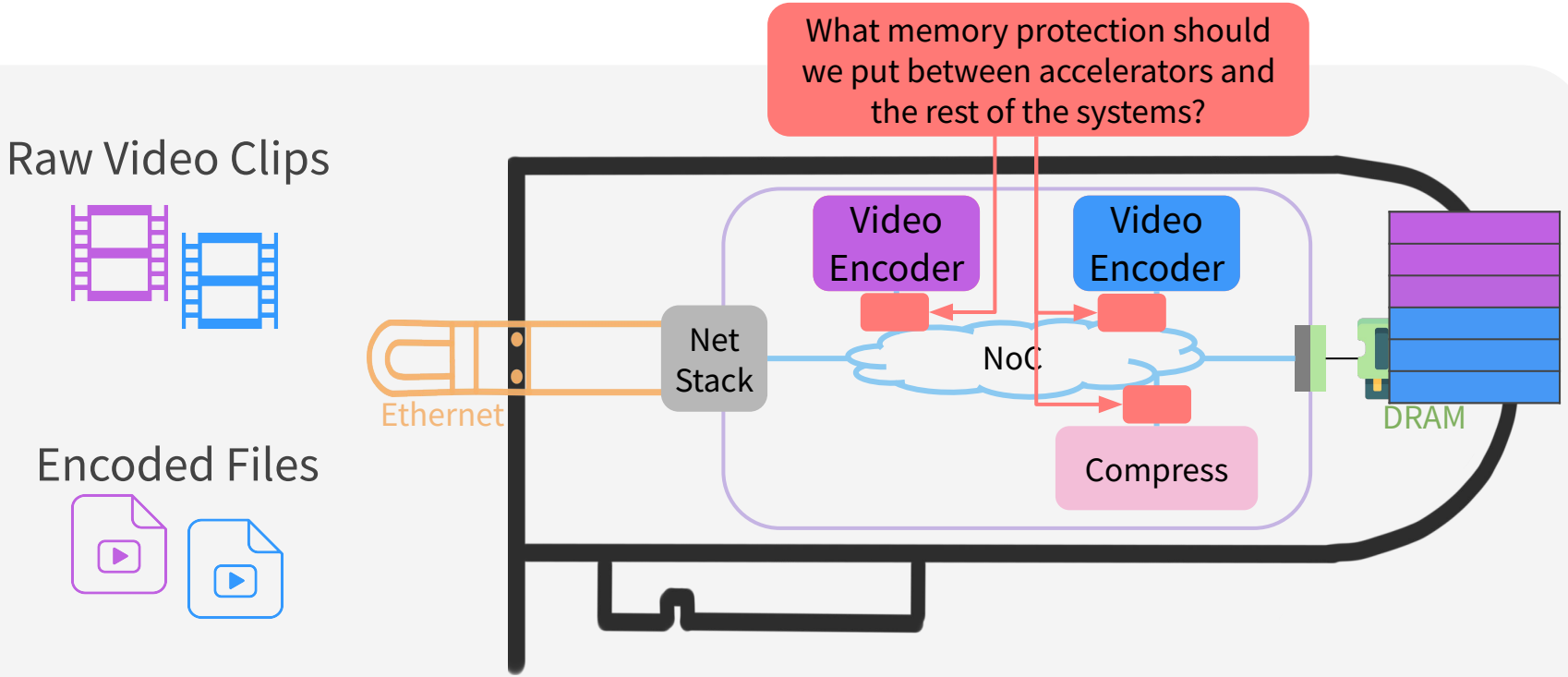
How do we handle more requests?



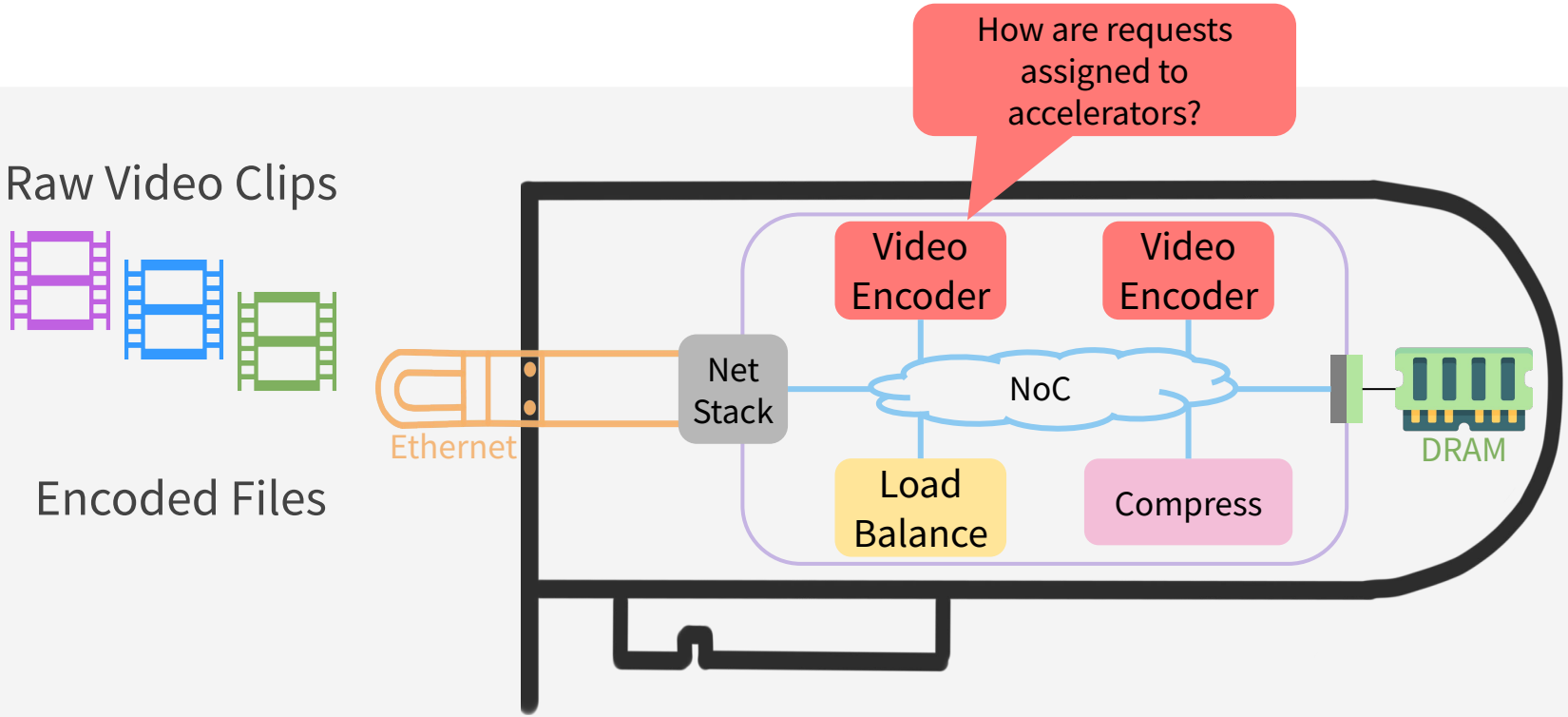
How do we control memory access?



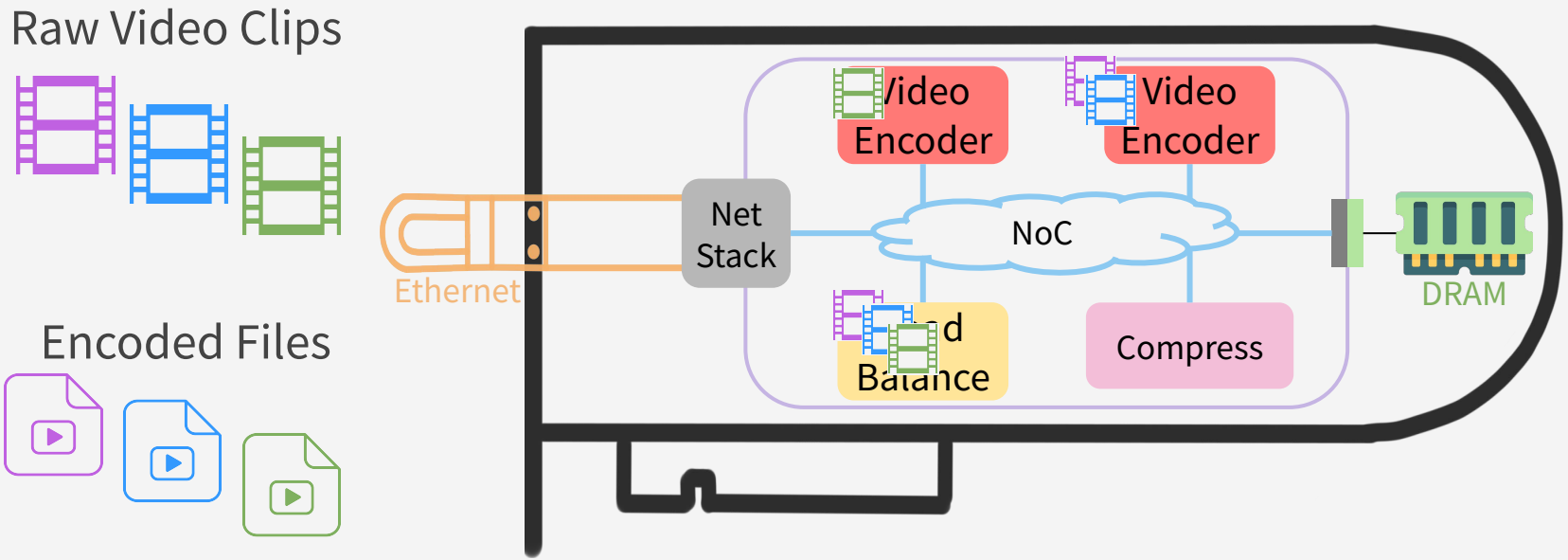
How do we control memory access?



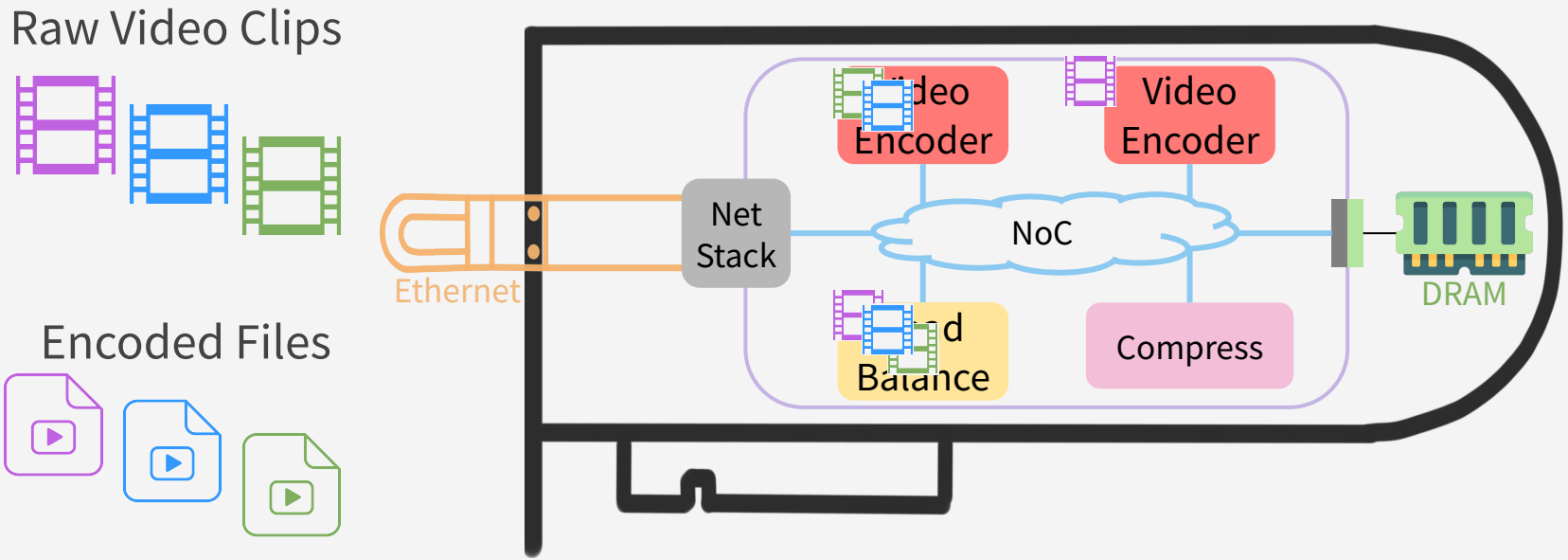
What if there are more streams than accelerators?



What if there are more streams than accelerators?

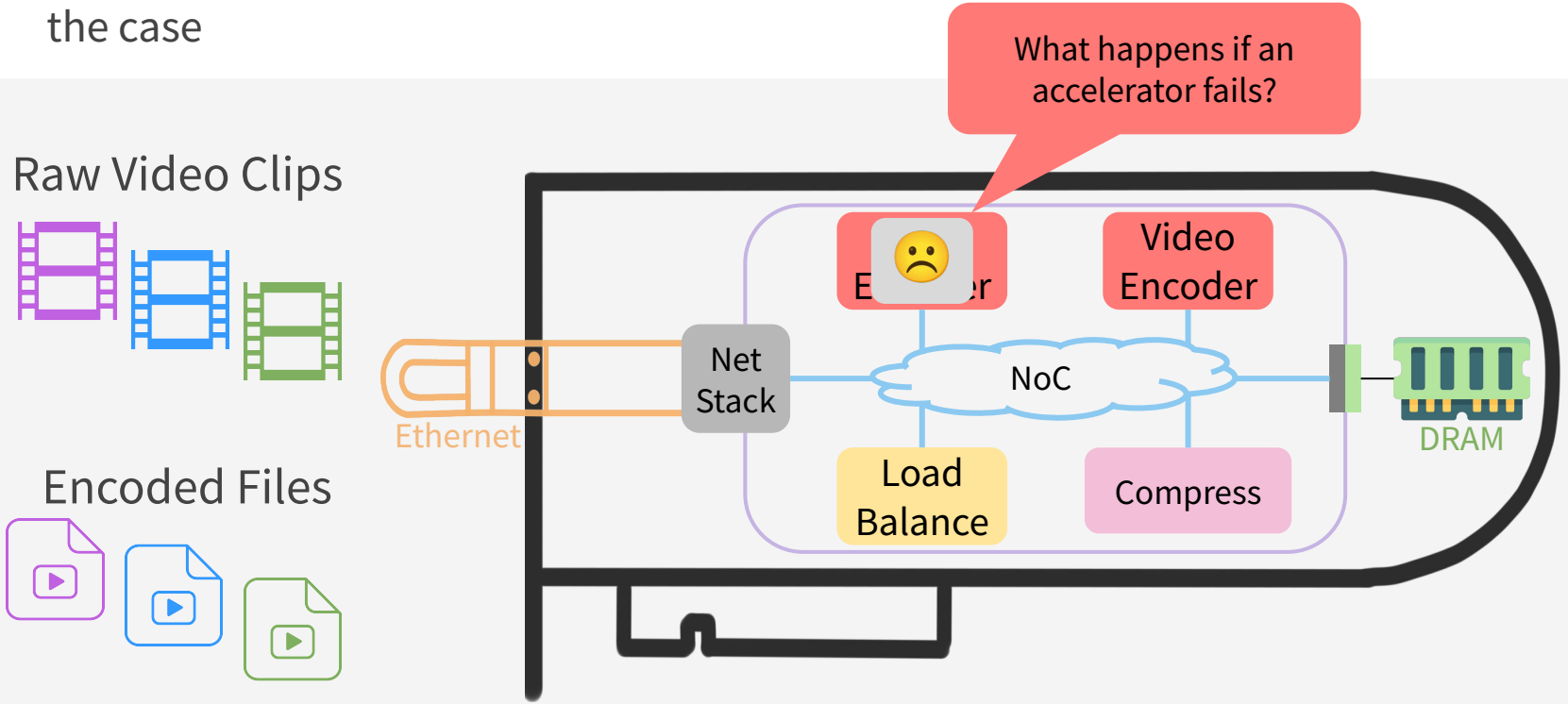


What if there are more streams than accelerators?



What happens on failures?

- Hardware is typically assumed to be correct all the time, but that's not always the case



Outline

Motivation

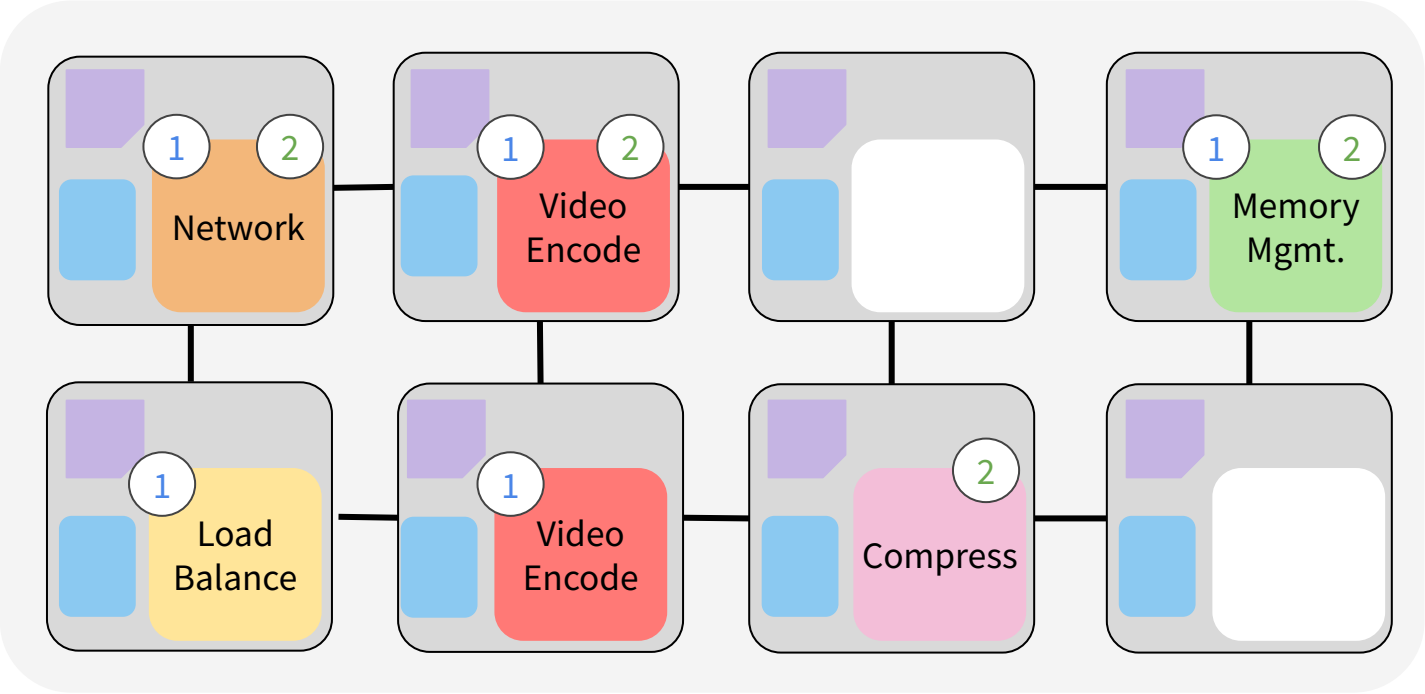
Apiary Goals

Use-case Study

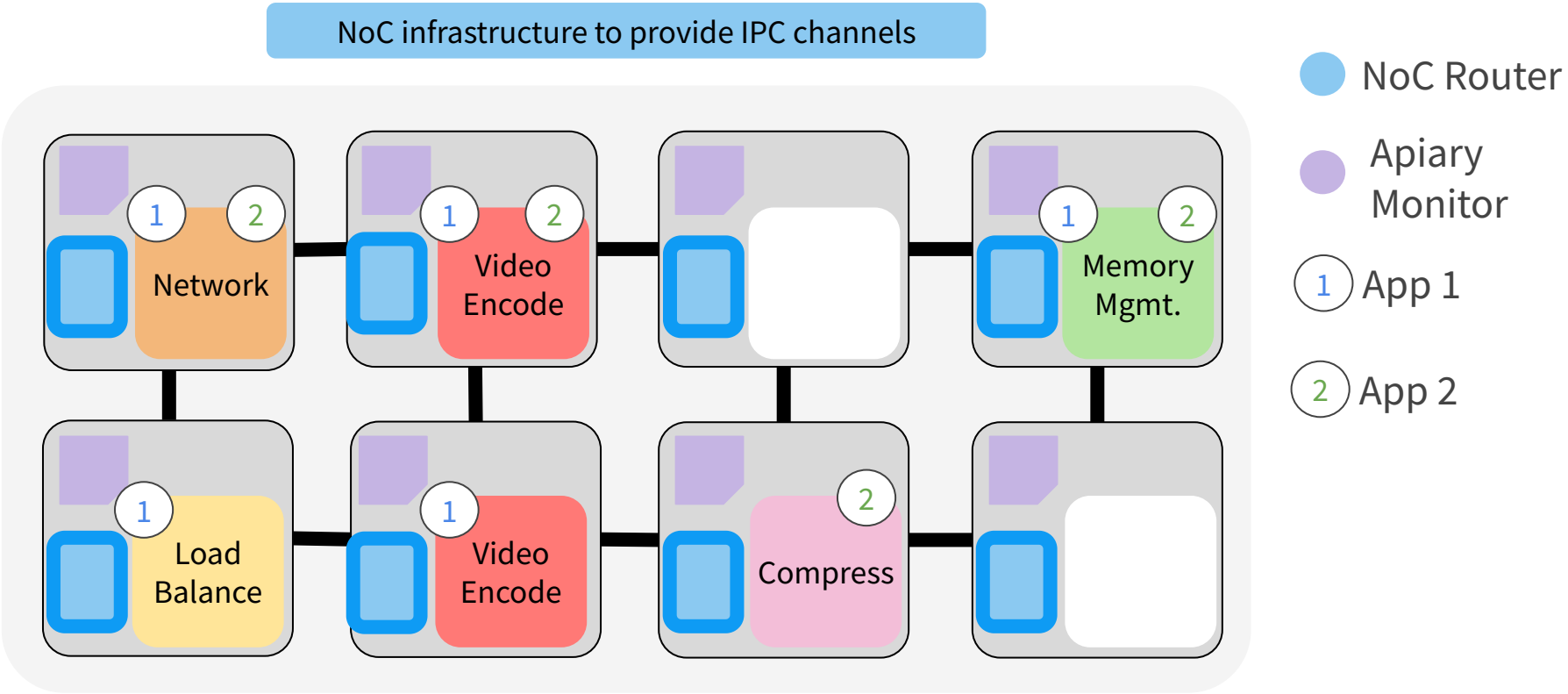
Apiary Architecture

Conclusion

Apiary Architecture Overview

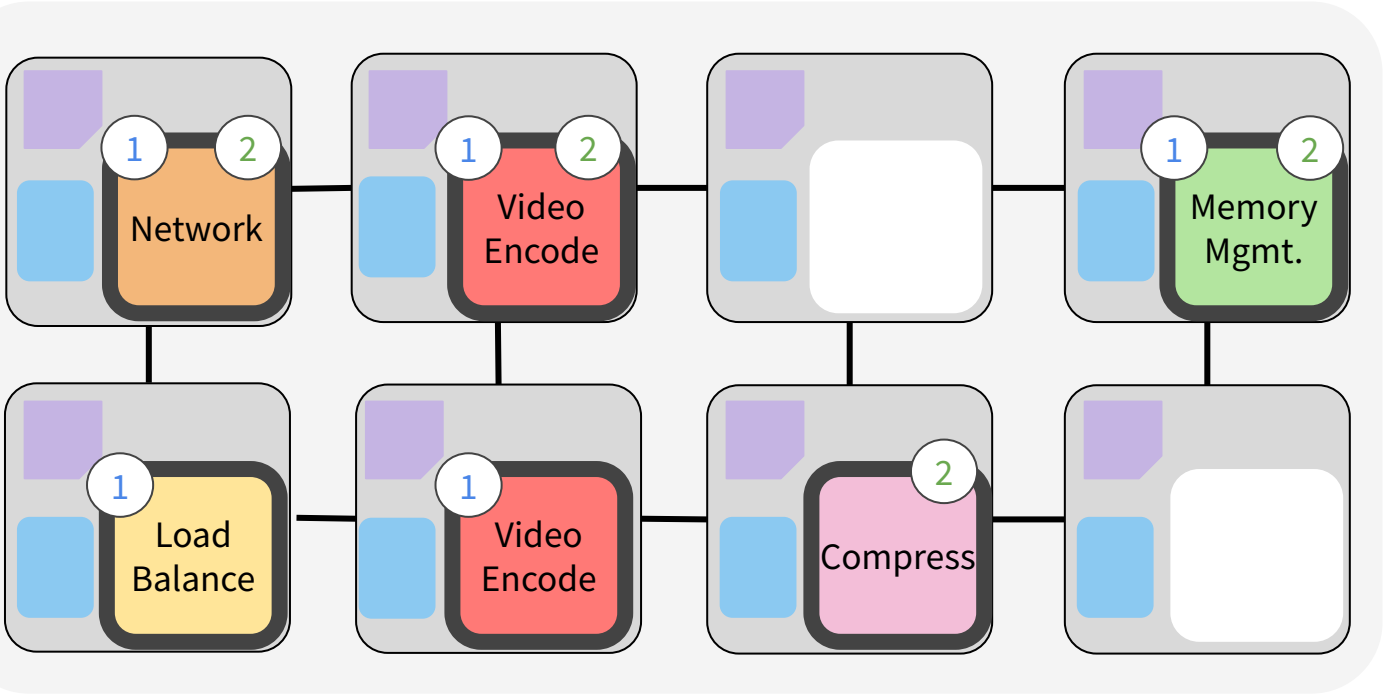


Apiary Architecture Overview



Apiary Architecture Overview

Tile logic is similar to a process or thread



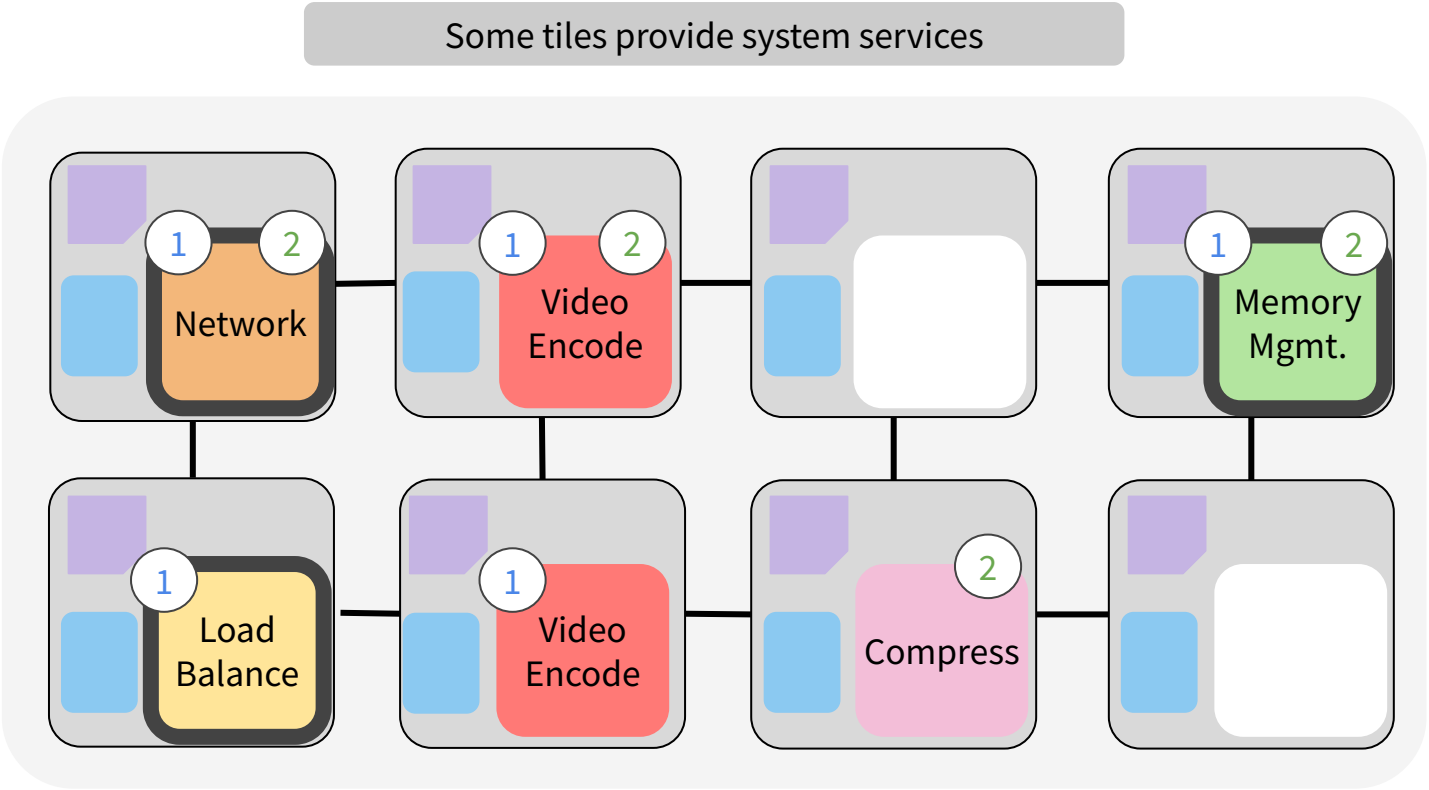
NoC Router

Apiary Monitor

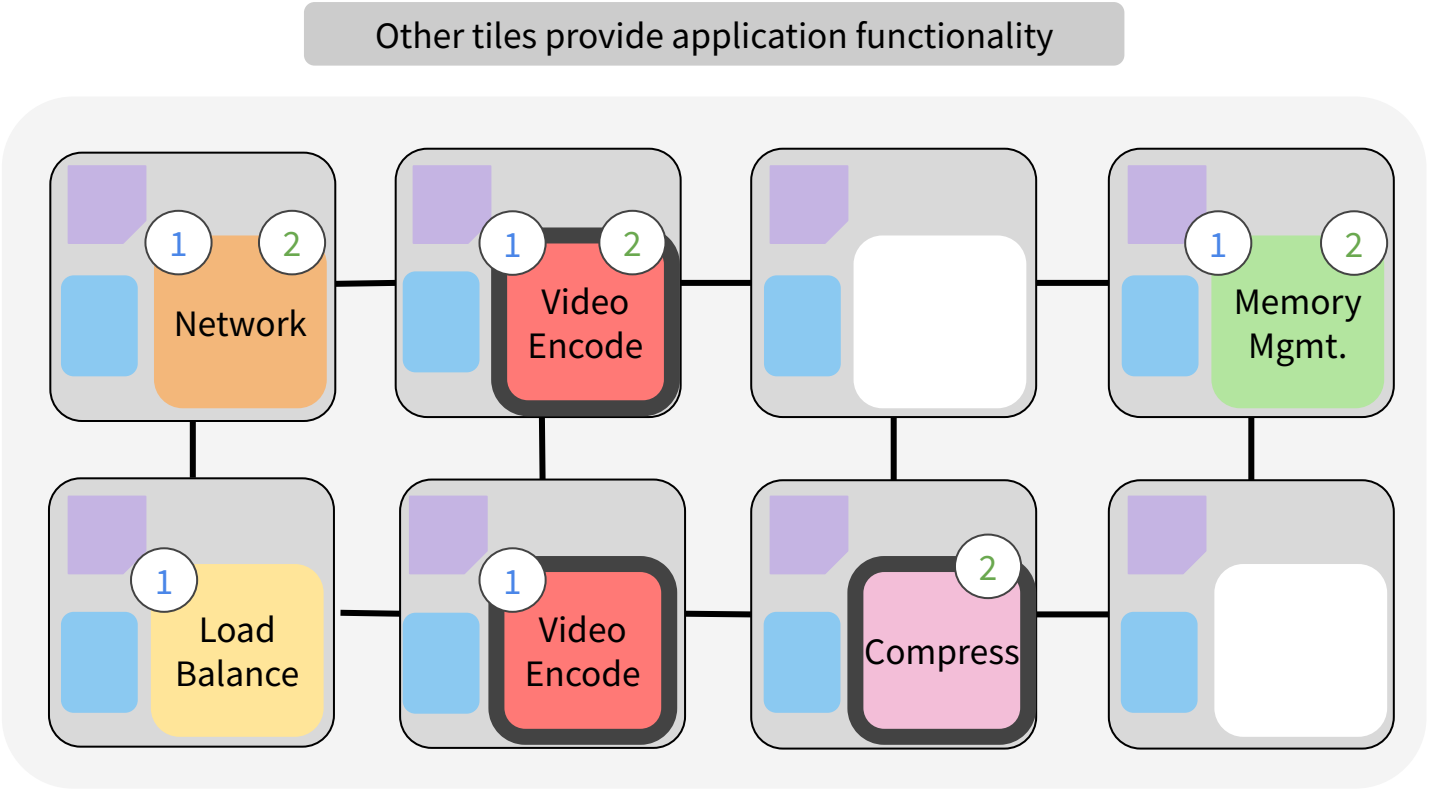
1 App 1

2 App 2

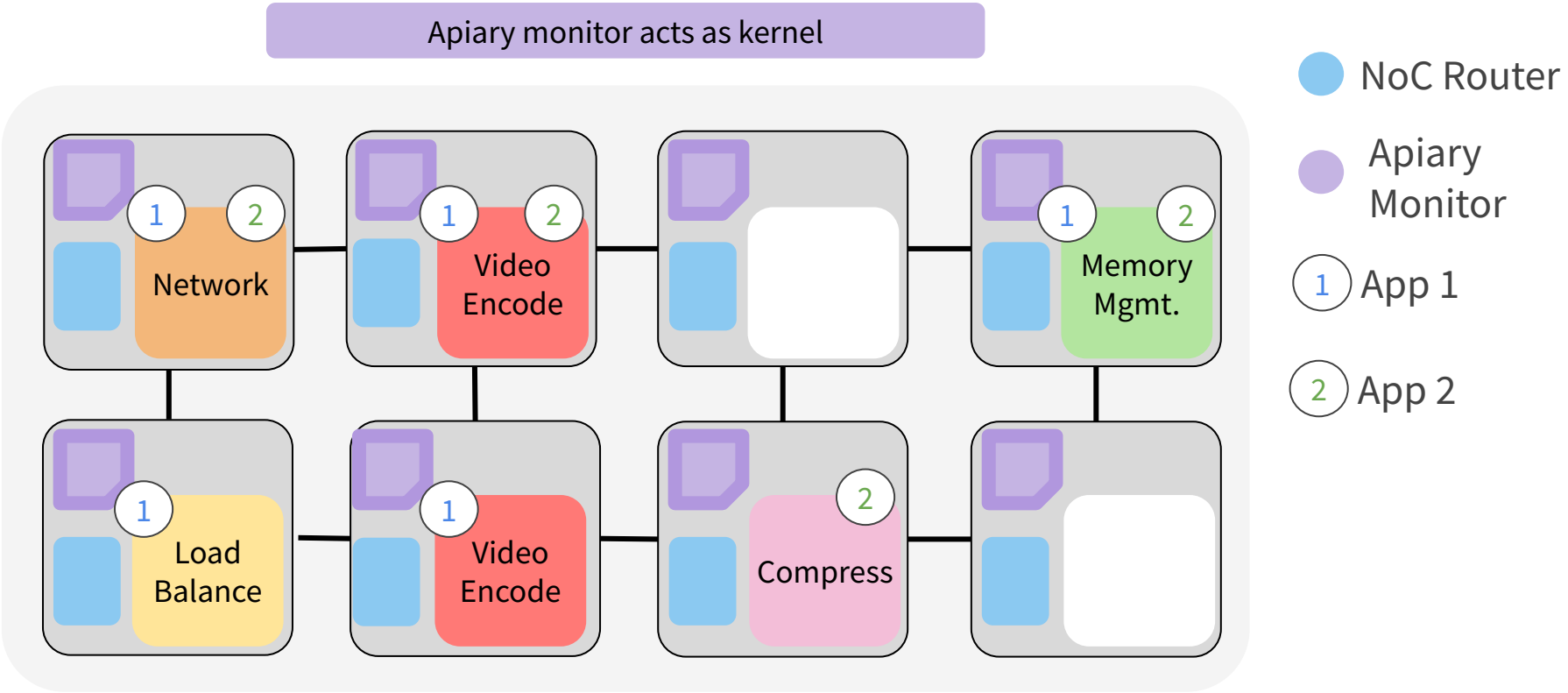
Apiary Architecture Overview



Apiary Architecture Overview



Apiary Architecture Overview



Conclusion

- FPGAs have been deployed in the datacenter, but are difficult to develop on due to lack of infrastructure and growing complexity
- We invite people to join us in thinking about these issues in order to unlock the potential of FPGAs. My personal favorites:
  - What less “conventional” OS abstractions could be relevant here?
  - How do you virtualize elements?
  - What does failure in one element mean for other elements?
- If you’re interested in hardware system services: Apiary’s network stack is already built and open-source (https://github.com/beehive-fpga/beehive)
  - Lim et al., “Beehive: A Modular Flexible Network Stack for Direct Attached Accelerators.” MICRO ‘24