Distinguished Artifact Award! 🏆

µShell

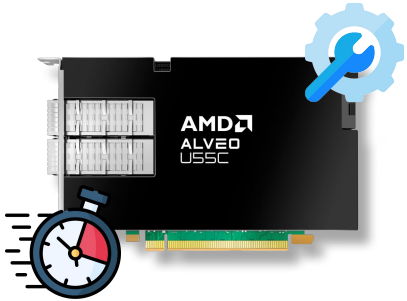
A Microkernel-based FPGA Shell Architecture

Jiyang Chen, **Anubhav Panda**, Harshavardhan Unnibhavi,
Atsushi Koshiba, and Pramod Bhatotia

Systems Research Group
https://dse.in.tum.de/

OSDI '26, Seattle, WA

FPGAs in the cloud



aws

Azure

ByteDance

Alibaba Cloud

FPGAs are **high performant, energy efficient,** and **re-programmable**

FPGAs are **deployed at scale** by major cloud providers

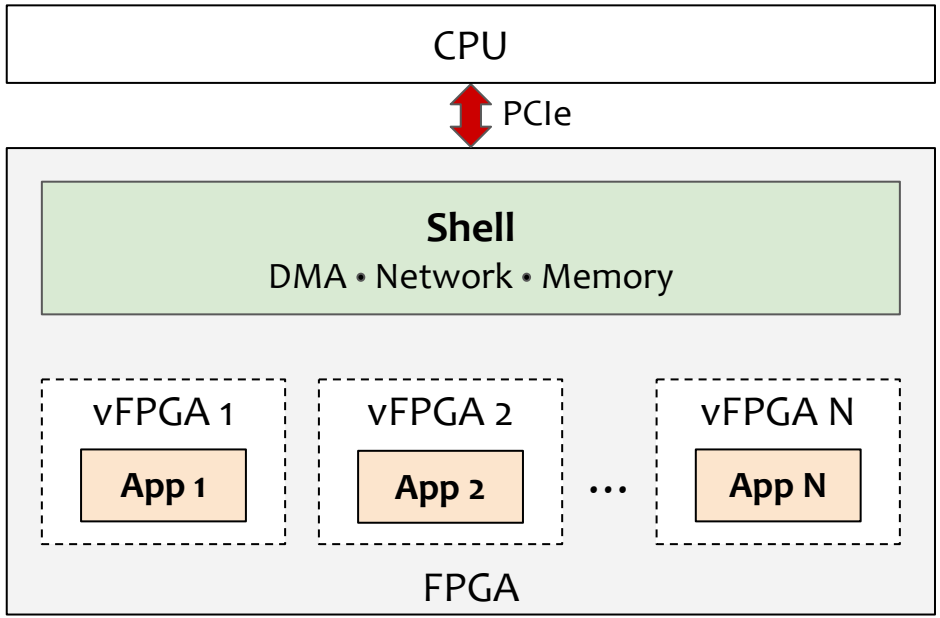
FPGA deployment model

Shell (static)

- Manages apps and provides shared services

vFPGAs (dynamic)

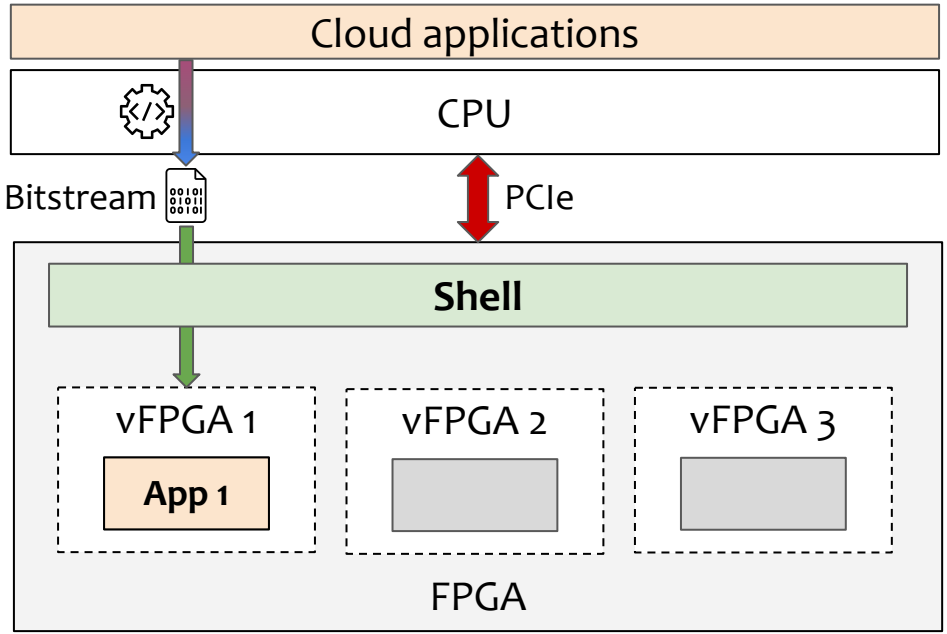
- Reconfigurable regions that deploy and run applications



The shell acts as an OS, managing the FPGA as a set of vFPGAs

Application deployments on FPGA

- **Synthesize:** Applications are synthesized into a single bitstream
- **Deployment:** The shell loads the bitstream onto a vFPGA



One application per vFPGA is known as **the monolithic model**

Monolithic deployment model

≠

Real-world cloud applications

Monolithic deployment model

Monolithic

Deployed as a single bitstream

≠

Modular

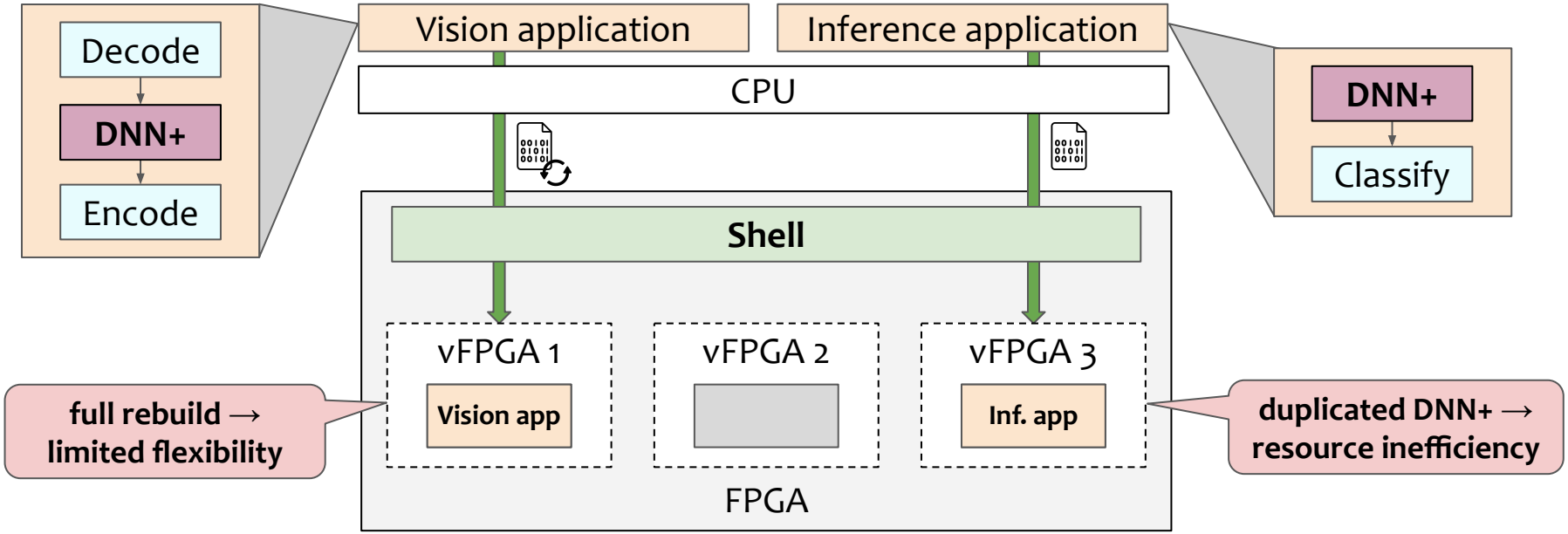
Composed of diverse, independent tasks

Composable

Tasks are shareable & reusable

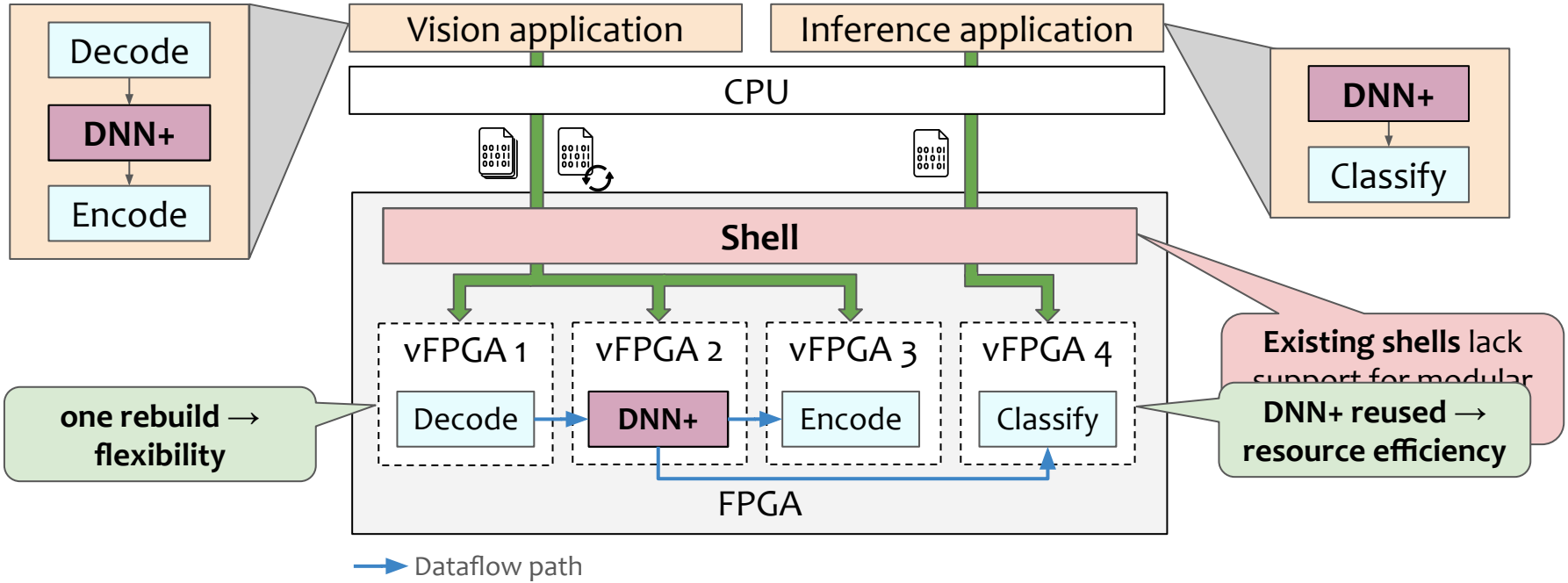
Real-world cloud applications

Monolithic FPGA deployment



Monolithic deployment leads to **limited flexibility** and **resource inefficiency**

The goal: **Modular** FPGA deployment



Modular deployment leads to **flexibility** and **resource efficiency**

Key challenges: What do FPGA shells lack?

1

Isolation

No access control between vFPGAs

2

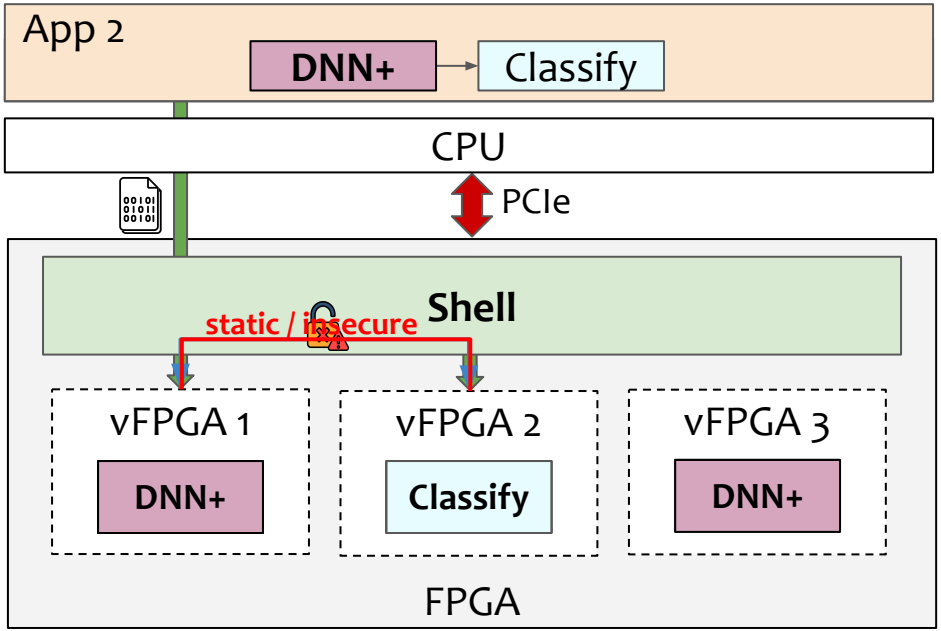
Module composition

Lack of dynamic & secure datapaths

3

Scheduling

No policy to share modules



Problem statement

How to build an FPGA shell for **real-world cloud applications** while ensuring **secure isolation, dynamic composition** and **efficient sharing**?

Proposal: µShell

A **microkernel**-based shell for modular FPGA application deployment

Key idea: Applying **microkernel principles** to FPGA shells

Microkernel principles adapted to FPGAs

1 **Capabilities**

Isolation across vFPGAs

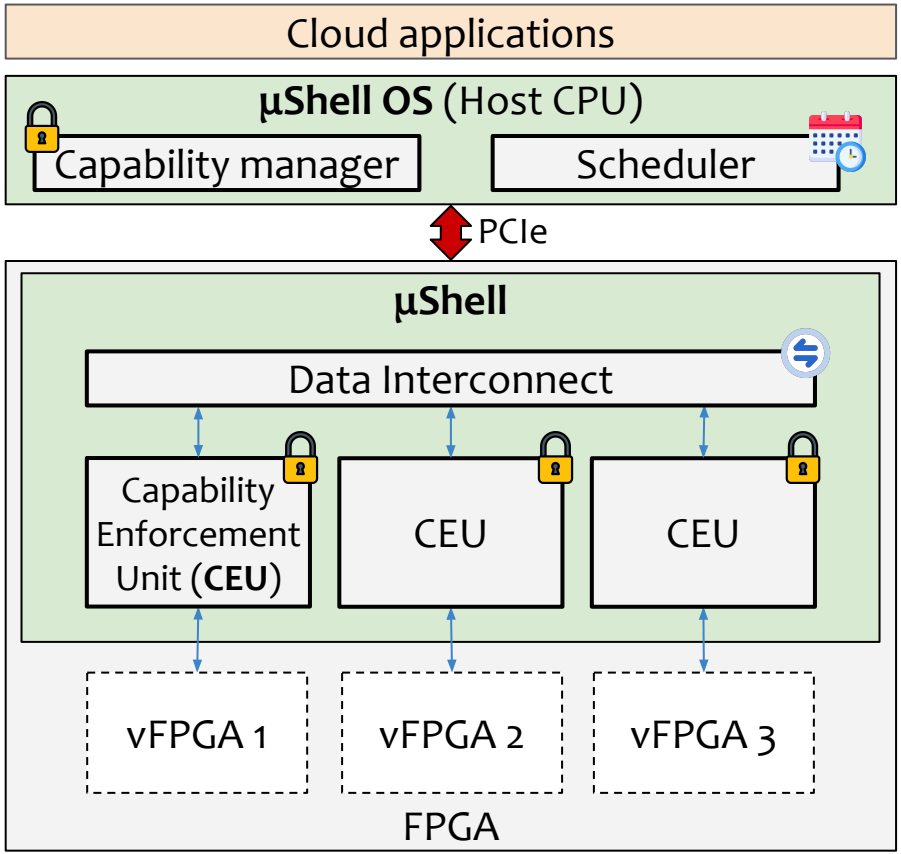
2 **IPC**

Dynamic module composition

3 **Scheduling**

Support of module sharing

µShell overview

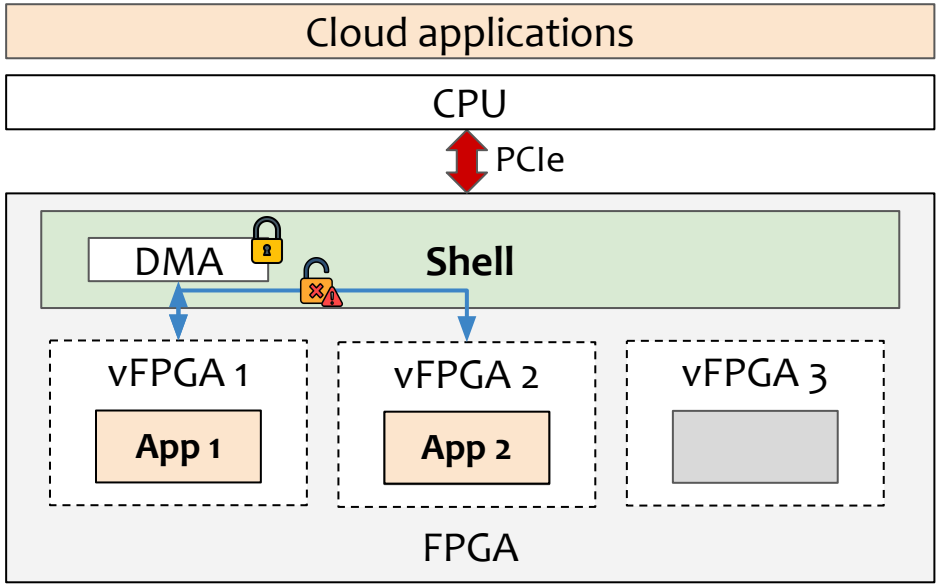


Outline

- ~~Motivation~~
- Design & Workflow
- Evaluation

Challenge #1: **Lack of isolation**

- Existing shells isolate only **memory accesses**
- **No isolation mechanism** over communication between vFPGAs

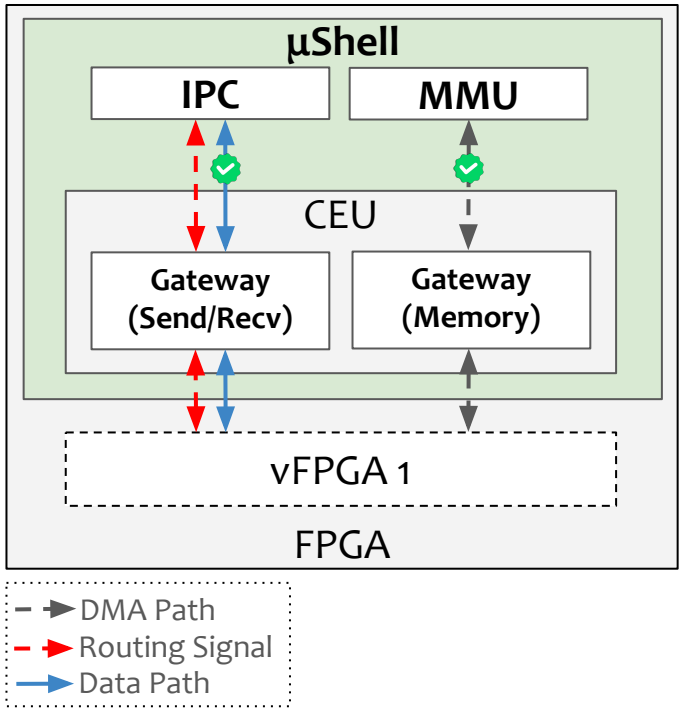


How can we **ensure secure isolation** across vFPGAs and memory accesses?

Solution #1: Capability-based isolation

The capability-based isolation works in **two parts**

- Hardware capability control (**CEU**)
  - **Memory gateways** verify memory access requests
  - **Send/Receive gateways** verify vFPGA to vFPGA requests

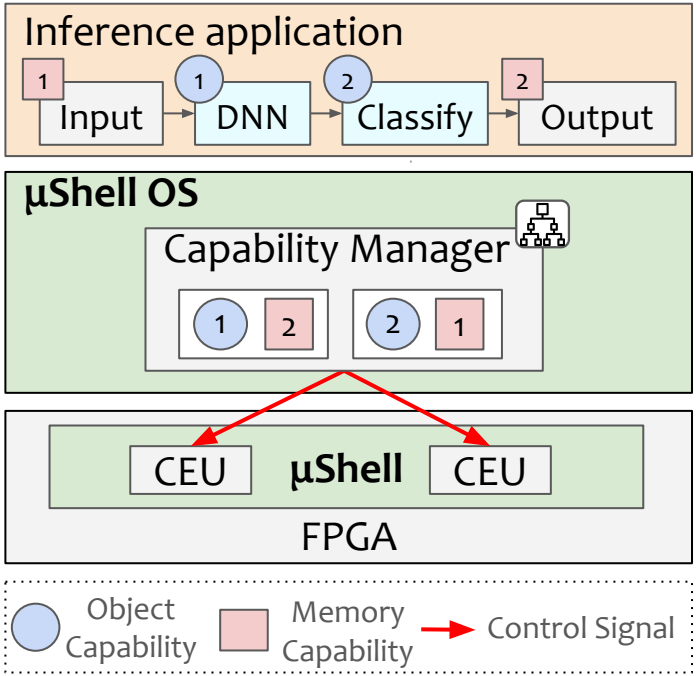


Capabilities provide **fine-grained isolation** across modules and memory

Solution #1: Capability-based isolation

The capability-based isolation works in **two parts**

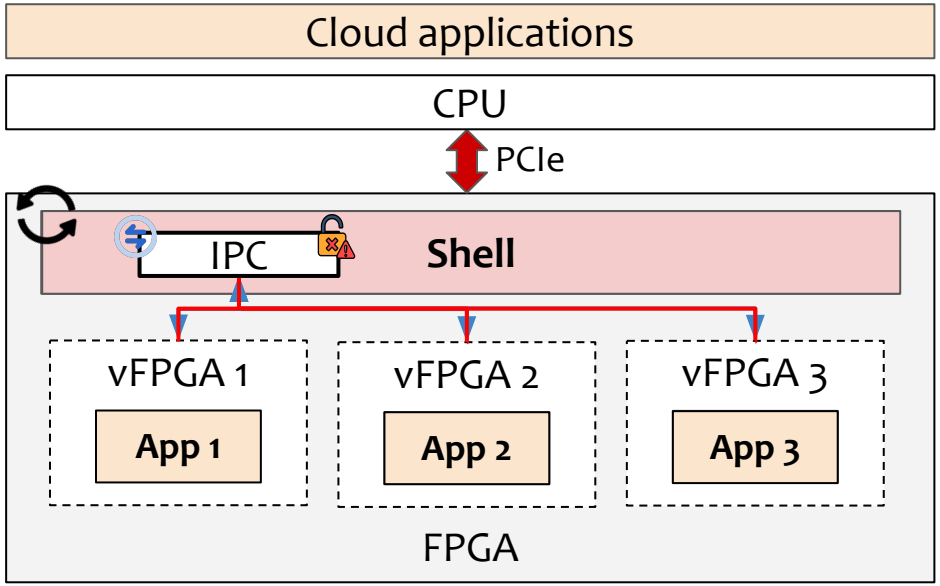
- Software capability control
  - **Object** and **memory** capabilities are managed at the OS level
  - **Capability manager** enforces them to the hardware



Software-managed, hardware-enforced capabilities provide fine-grained isolation

Challenge #2: **Lack of IPC**

- Static interconnects require **recompiling the entire shell**
- Dynamic interconnects target networking only, or **lack fine-grained isolation**

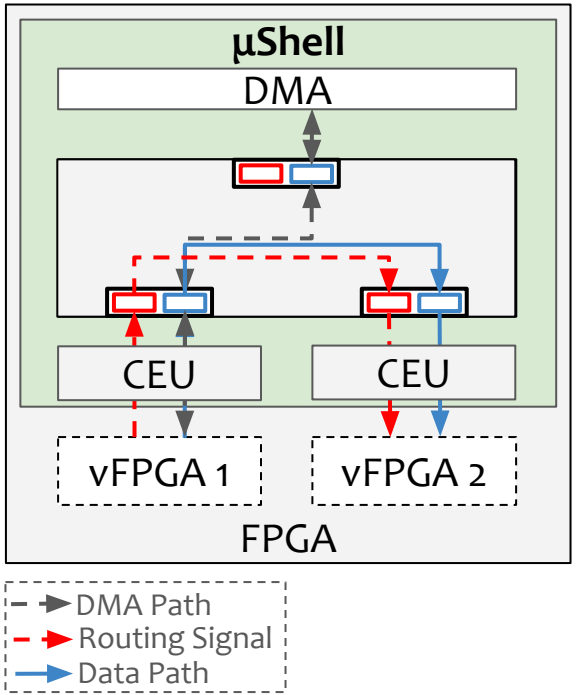


How can we ensure **secure and flexible communication** between modules?

Solution #2: Secure IPC mechanism

Data Interconnect dynamically routes data streams

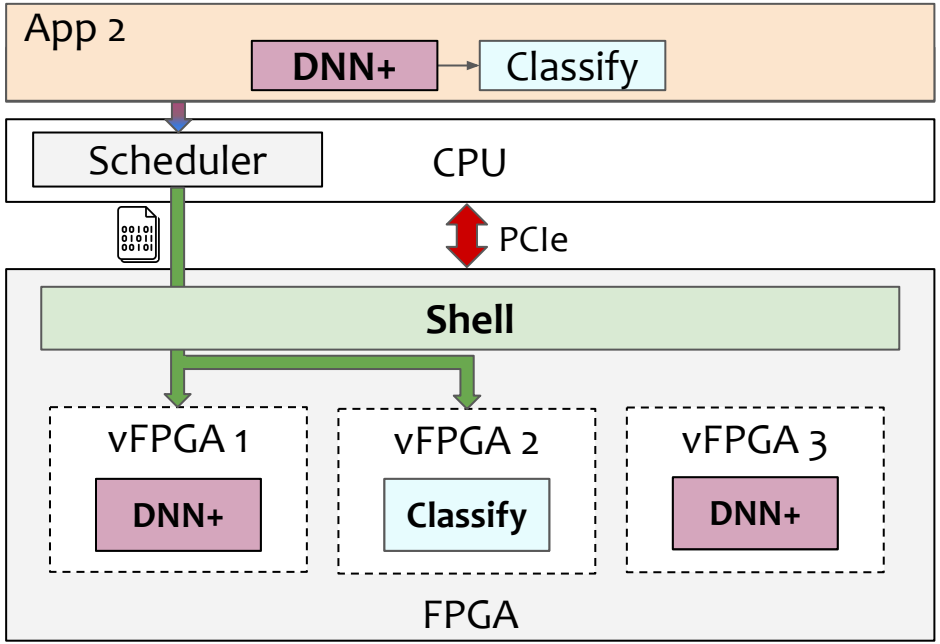
- vFPGA-to-memory communication
  - The interconnect links the vFPGA to the DMA engine
- Cross vFPGA communication
  - The interconnect sets a direct path between vFPGAs



µShell's IPC mechanism ensures secure and flexible communication between modules

Challenge #3: **Lack of scheduling**

- Existing schedulers are unaware of shared modules, causing **unnecessary reconfigurations**
- Existing schedulers also **underutilize FPGA resources**



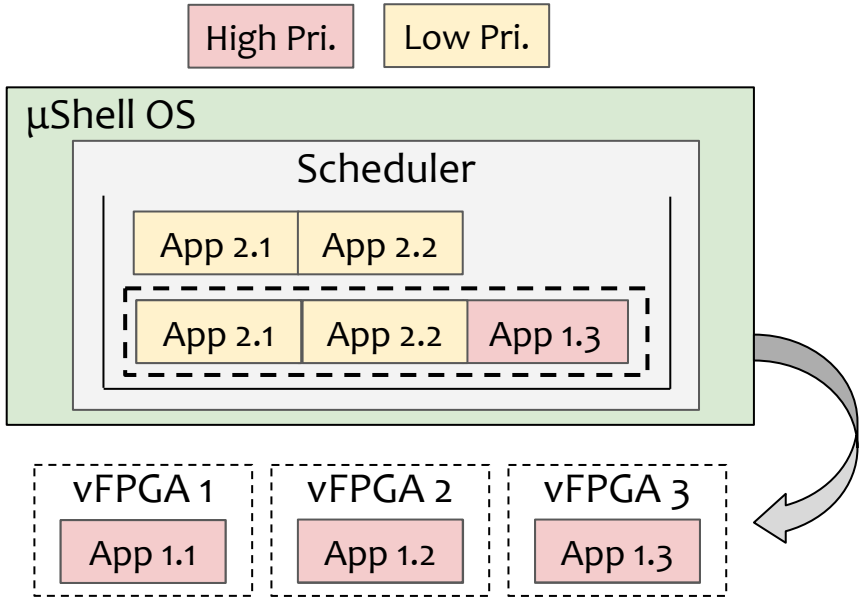
How can scheduling **avoid reconfigurations** and **optimize FPGA resources**?

Solution #3: Component-aware scheduling

The scheduling algorithm generates a schedule based on

- Higher priority
  - App 1 occupies all 3 vFPGAs

→ leads to a priority based schedule

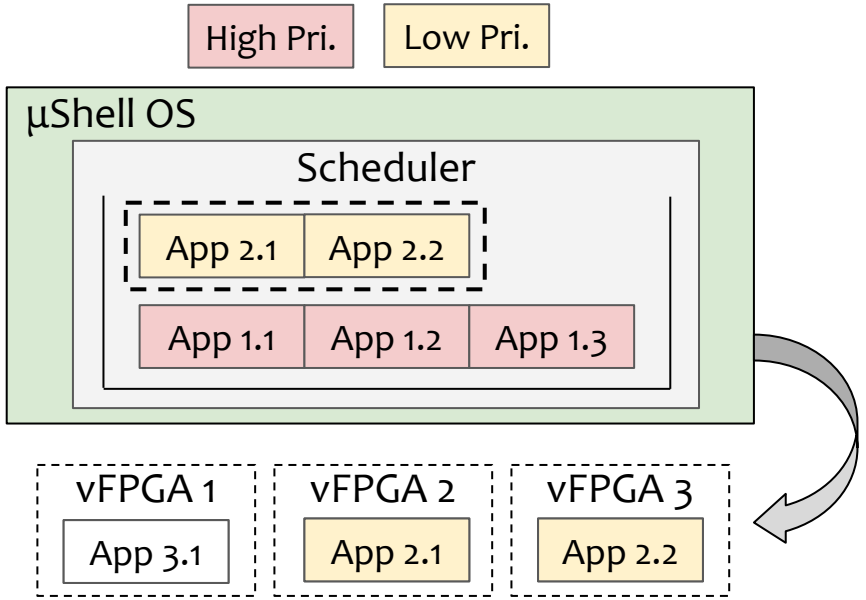


Solution #3: Component-aware scheduling

The scheduling algorithm generates a schedule based on

- Higher priority
- Limited priority inversion
  - Idle vFPGAs filled by App 2 and App 3

→ leads to resource efficiency

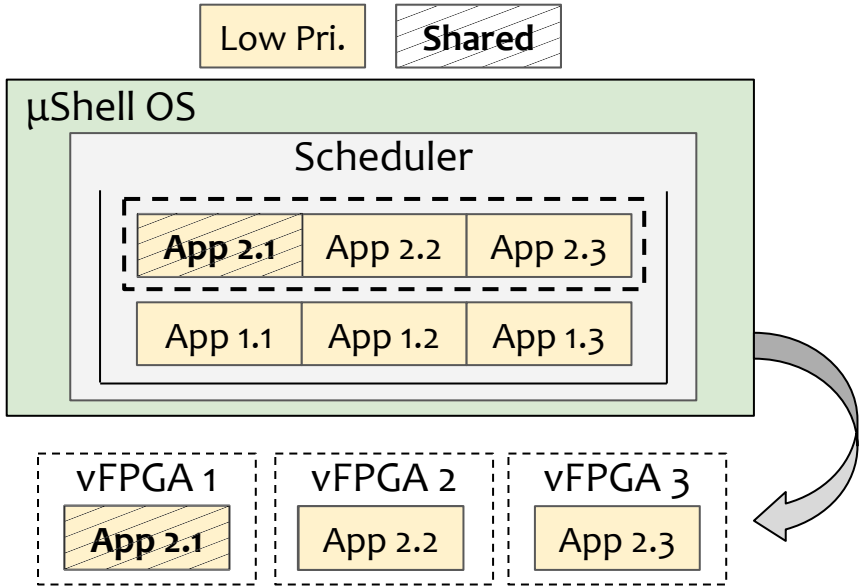


Solution #3: Component-aware scheduling

The scheduling algorithm generates a schedule based on

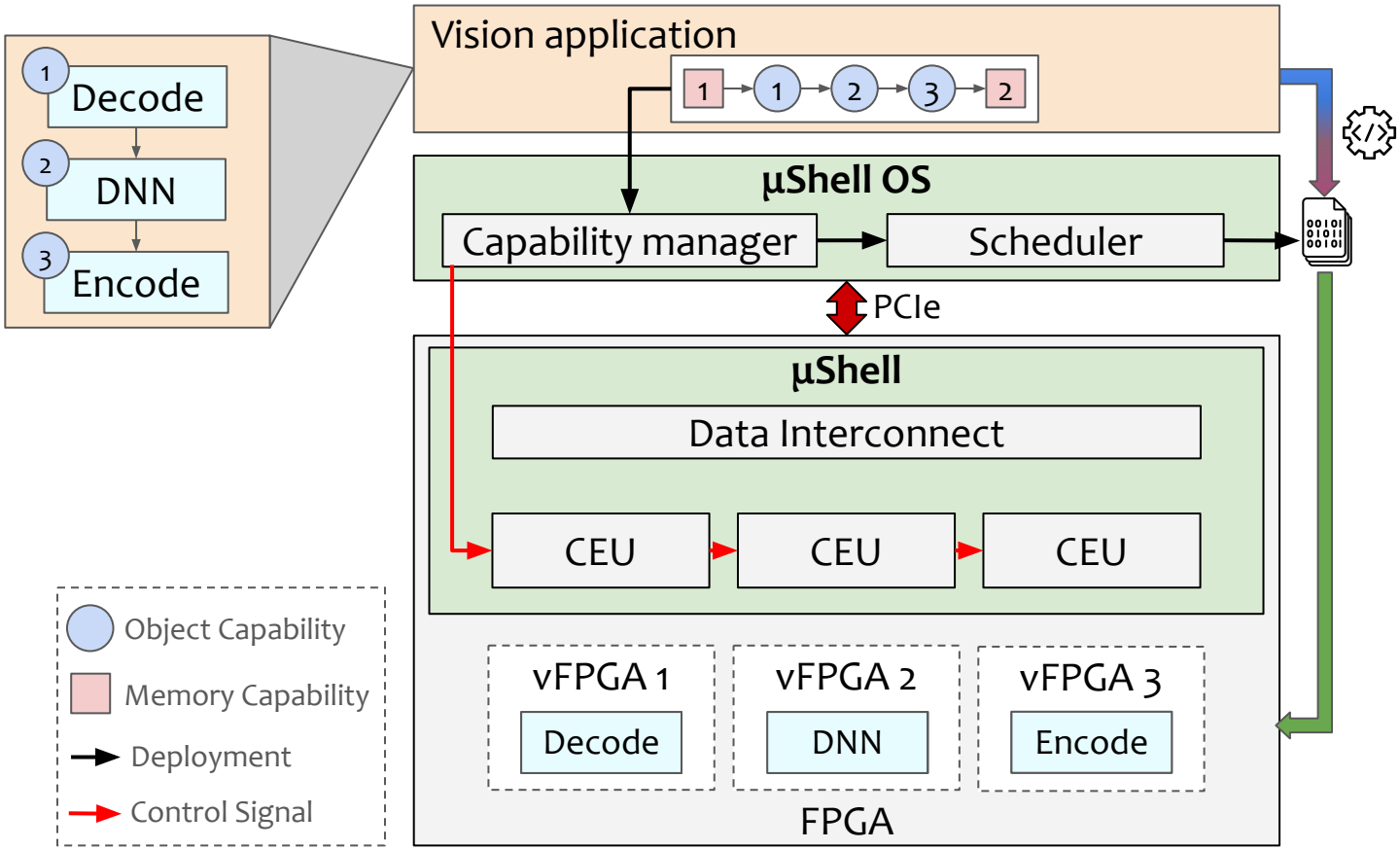
- Higher priority
- Limited priority inversion
- Component overlap
  - App 2 runs by reusing the **shared module**

→ leads to high flexibility

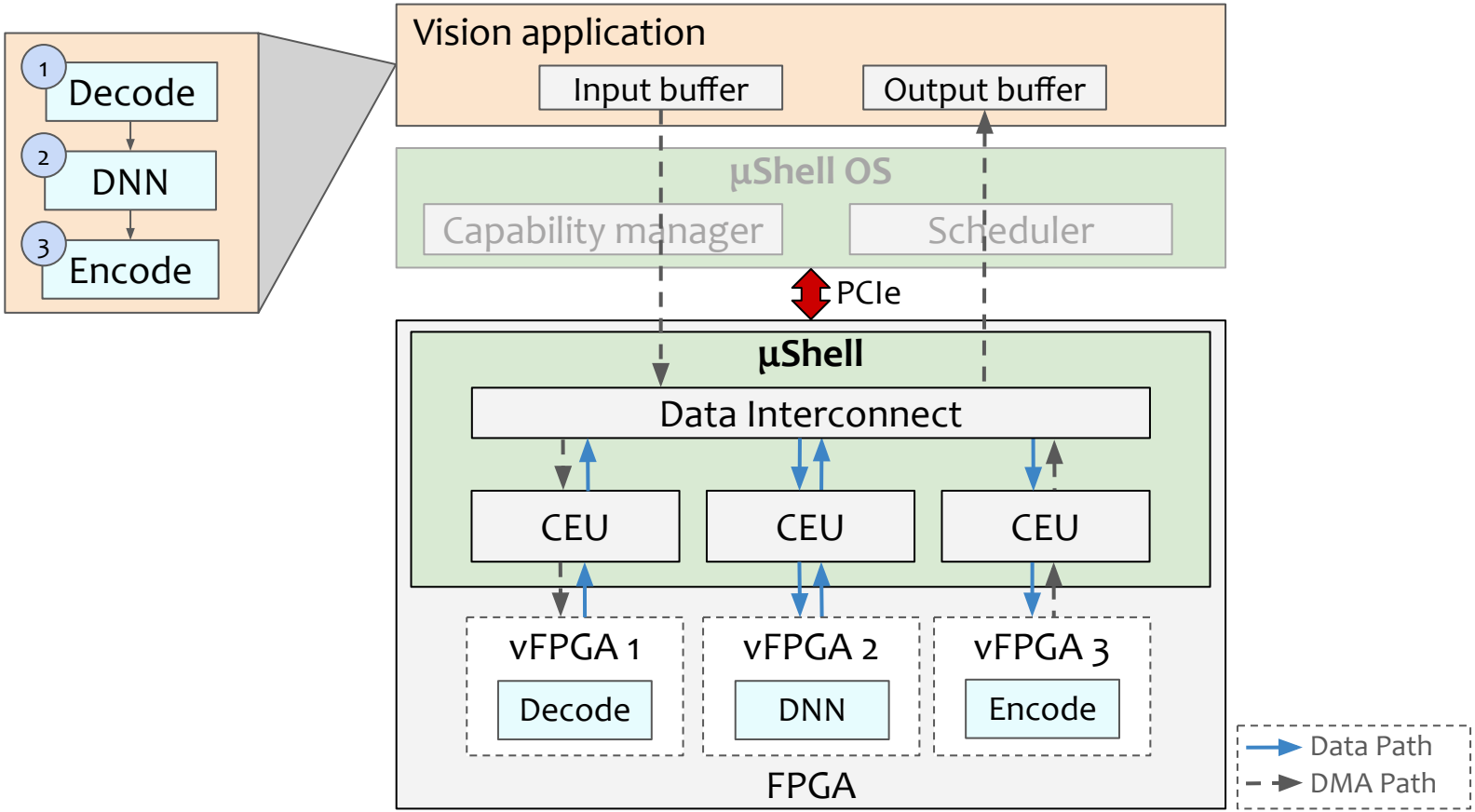


Component-aware scheduling ensures **priority, resource efficiency,** and **flexibility**

End-to-end workflow



End-to-end workflow



Outline

- ~~Motivation~~
- ~~Design & Workflow~~
- Evaluation

Evaluation

1. **End-to-end Performance:**
   *Does µShell's modular deployment **preserve performance** compared to monolithic shells?*

2. **Deployment costs:**
   Can µShell l**ower deployment cost** by reusing already-deployed modules?

3. **Scheduler effectiveness:**
   *Can µShell's component-aware scheduler **reduce reconfigurations** and **meet deadlines**?*

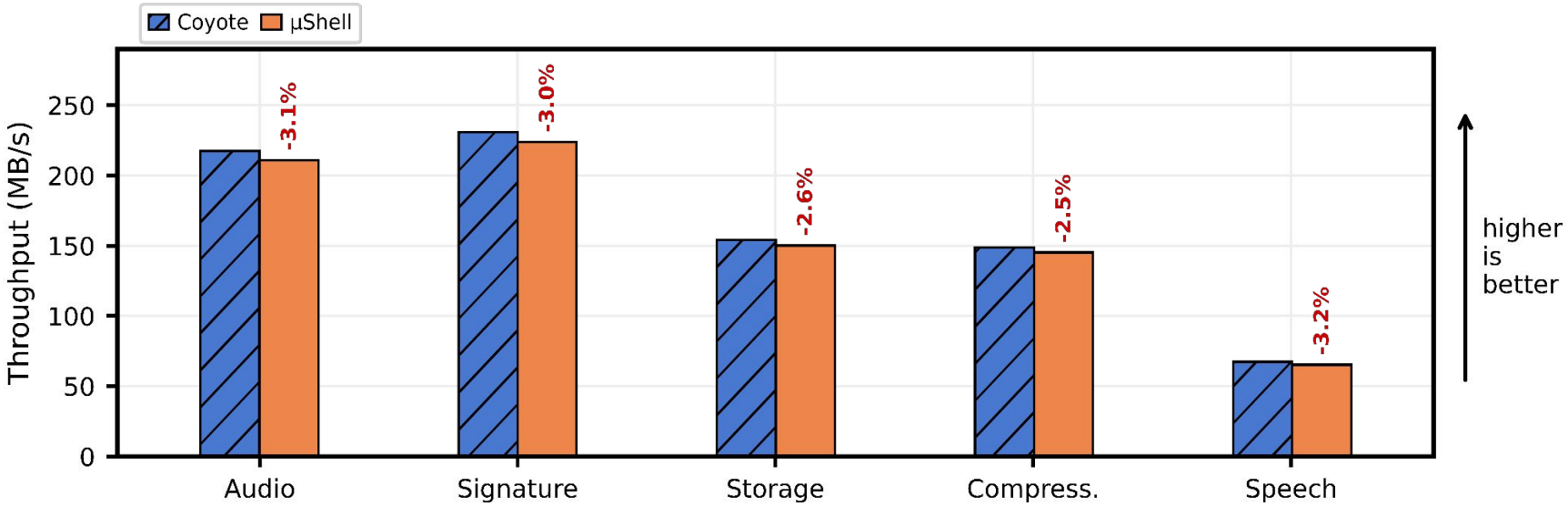
More evaluation metrics in the paper!

Evaluation

- Platform
  - AMD EPYC 7413 (NixOS, Linux 6.9)
  - AMD Alveo U280
- Baseline
  - Coyote v2 (monolithic deployment model)
- Application benchmarks
  - Audio processing
  - Digital signature
  - Secure storage
  - Signed compression
  - Speech recognition

Evaluation: **Performance**

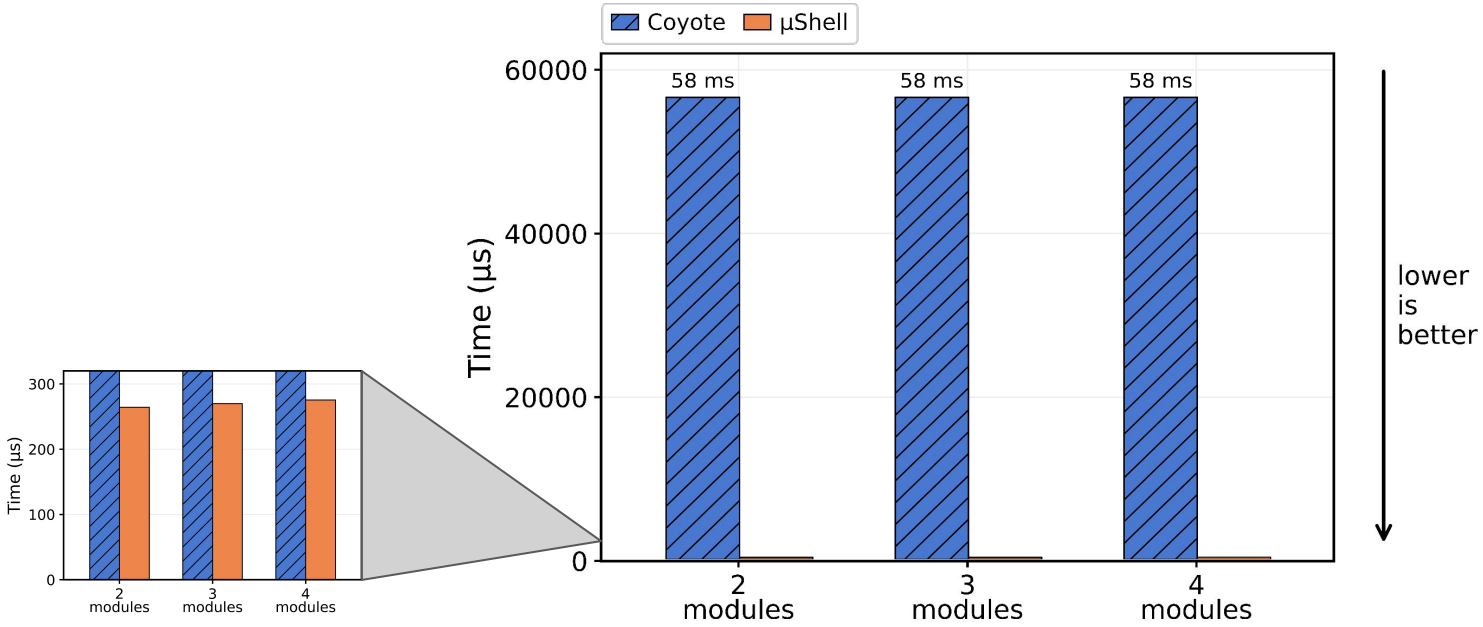
Does splitting an app across vFPGAs hurt throughput?



µShell delivers modular deployment with near-identical throughput

Evaluation: **Deployment costs**

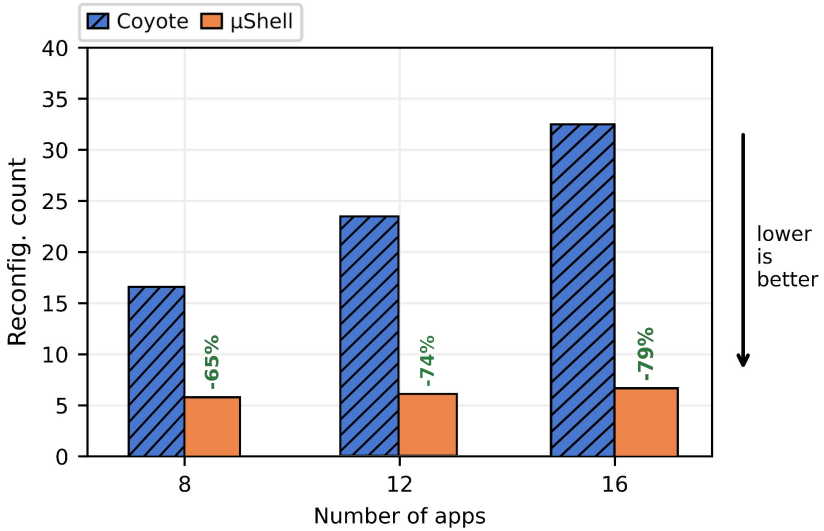
Does splitting an app across vFPGAs affect deployment time?

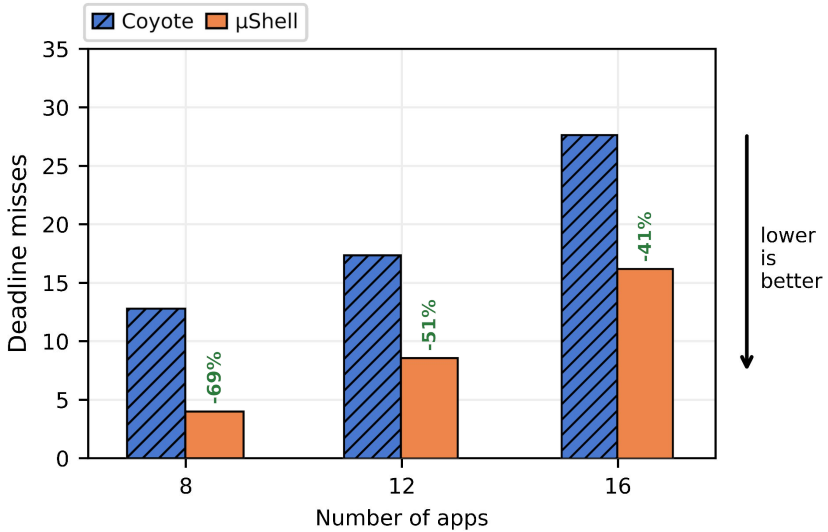


µShell reuses deployed modules, cutting deployment cost by orders of magnitude

Evaluation: **Scheduling**

Does our scheduler reduce reconfiguration counts and meet deadlines?





µShell reduces reconfigurations by up to **79%** and deadline misses by up to **69%**

Summary

Can we build an FPGA shell for real-world cloud applications?

µShell: A microkernel-based FPGA shell architecture

- **Capability-enforced isolation** across modules and memory
- **Secure, flexible IPC** between modules
- **Component-aware scheduling** for resource efficiency

github.com/TUM-DSE/microShell

anubhav.panda@tum.de

jiyang.chen@tum.de

Looking for jobs!