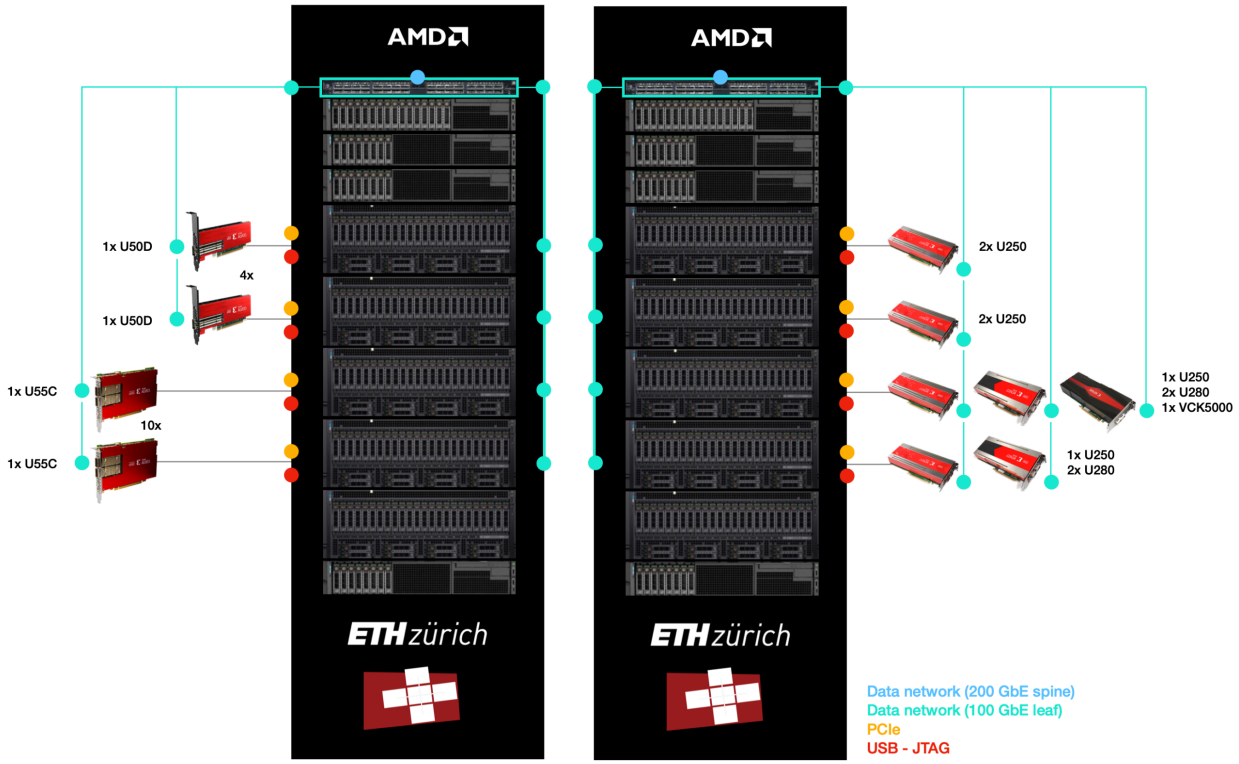
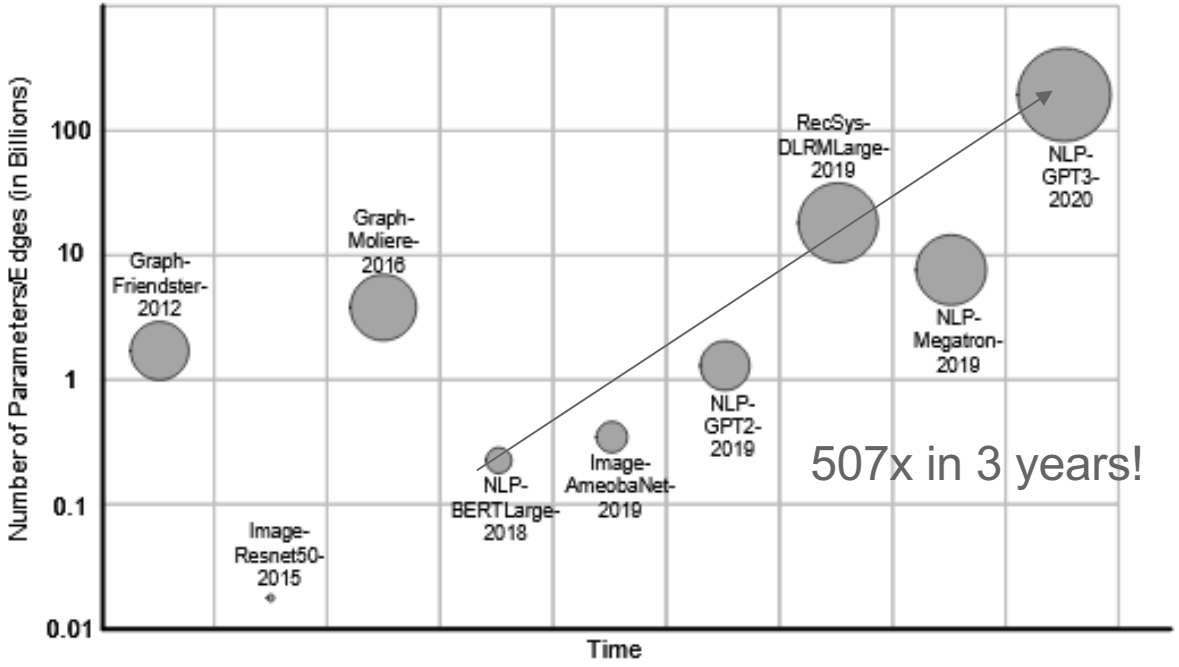
Open Source Resources and Infrastructure for FPGAs

Dario Korolija, Zhenhao He, Wenqi Jiang, Gustavo Alonso



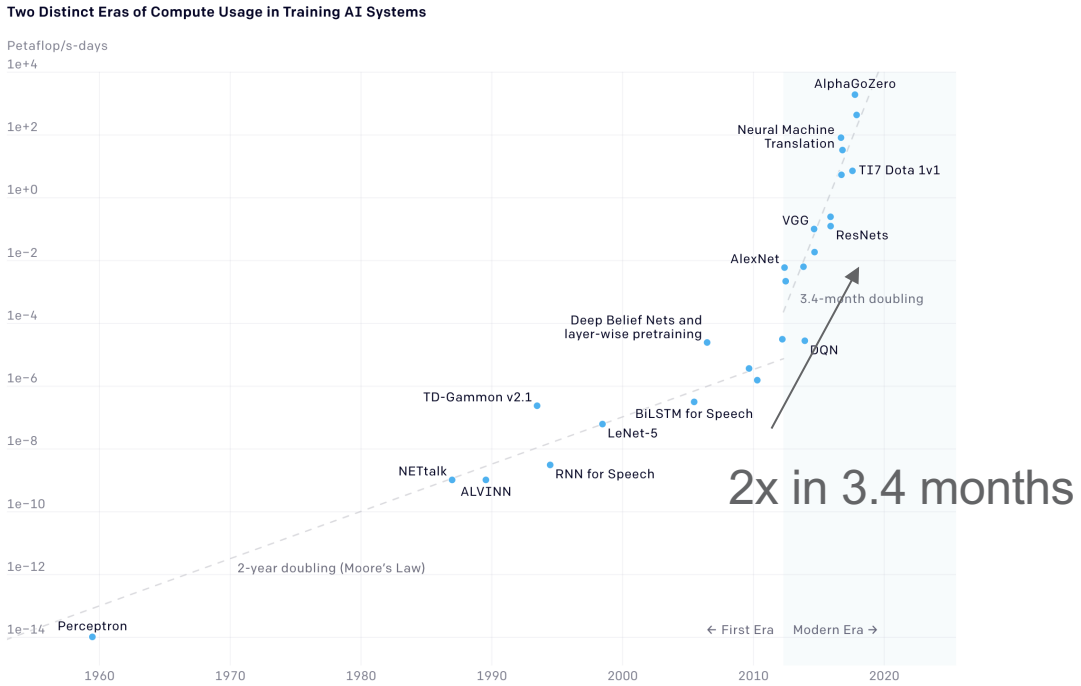
Trends in data centers

- Evolution of computation requirements in modern **HPC** and **datacenter** AI applications



AI application memory requirements

Tearing Down the Memory Wall https://arxiv.org/pdf/2008.10169.pdf



AI application compute requirements

AI and Compute https://openai.com/blog/ai-and-compute

- ***Dynamic workload;***
- ***Heterogeneous devices;***
- ***Distributed computing***

Efficient data processing across data center is important…

Database example

SELECT * FROM T WHERE id=3

- A database will read the table from cloud storage
- Bring it all the way to the local memory, then to the CPU registers
- Just to throw away all tuples but 1
- Creates bottlenecks in storage, network, memory access, data buses, pollutes the caches, CPU cycles, etc.

As the amount of data to process keeps growing, its movement throughout the system has become one of the biggest bottlenecks and source of inefficiencies

→**Smart processing on distributed resources**

Large deployment of FPGAs in the cloud

As consequences of growing compute/storage demands and requirements of efficiency…

For specialization while keeping flexible for dynamic workloads…

Large deployment of FPGAs in the cloud – examples

- FPGAs as in-network accelerator to meet growing compute demands

- Microsoft Catapult Project
  - *https://www.microsoft.com/en-us/research/project/project-catapult/*

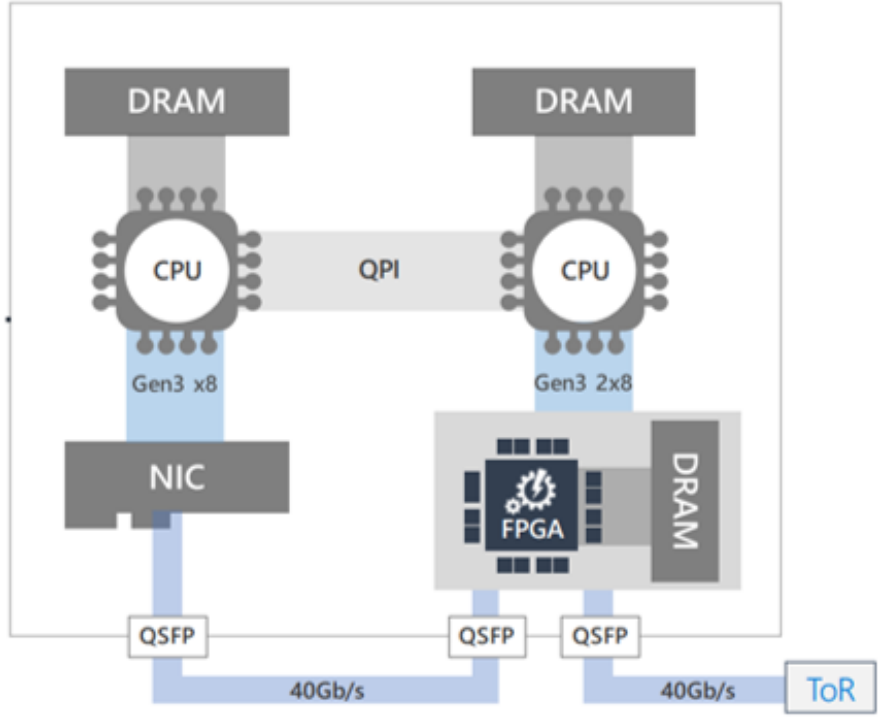
  - Bump-in-the-wire architecture

  - Accelerate machine learning workload, e.g., CNN

  - Accelerate page rank

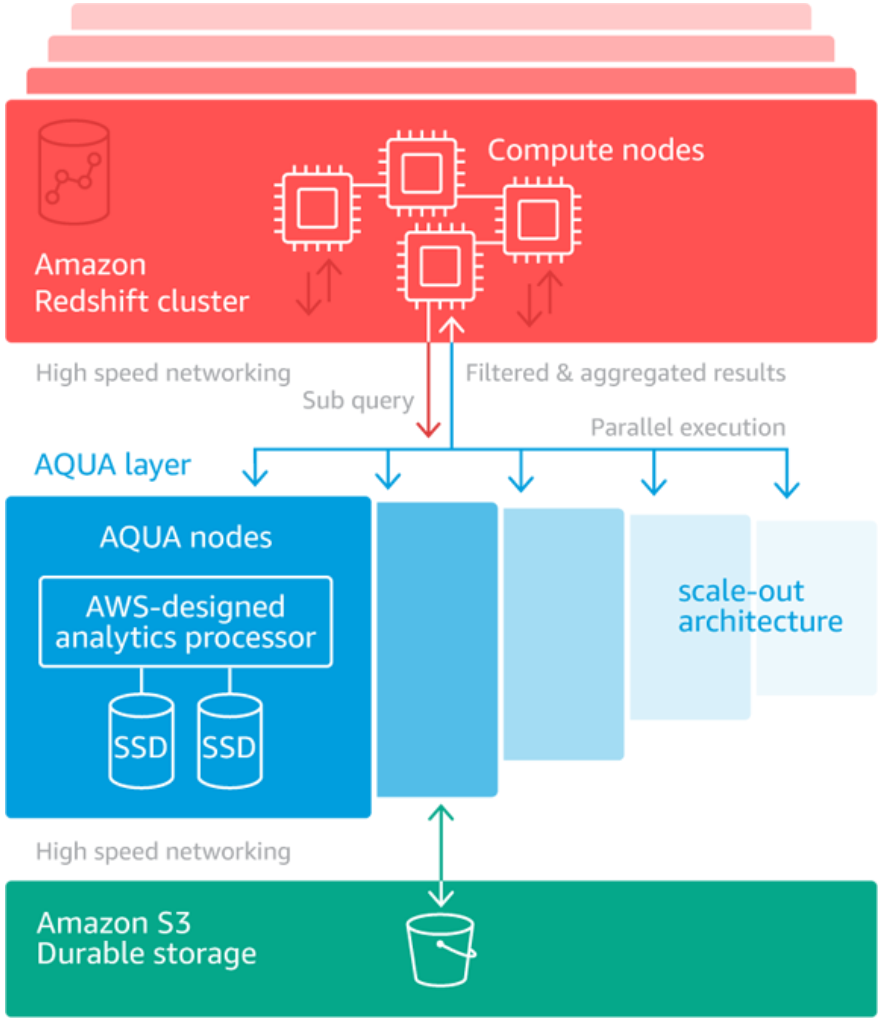
  - Also for QoS, storage acceleration …

  - Production deployed in Bing



Large deployment of FPGAs in the cloud – examples

- FPGAs as smart accelerator for disaggregated resources
- Amazon AQUA
  - *https://aws.amazon.com/blogs/aws/new-aqua-advanced-query-accelerator-for-amazon-redshift/*
  - Analytic engine with FPGAs
  - Pushing computation closer to data
  - Reduce CPU compute requirement
  - Reduce network traffic



Efforts from academia

Similar trends in academia to study how to place FPGAs in a larger system, for performance and efficiency.

Just list a few…

Near-data processing:

KV-Direct (SOSP’17), Caribou (VLDB’17)

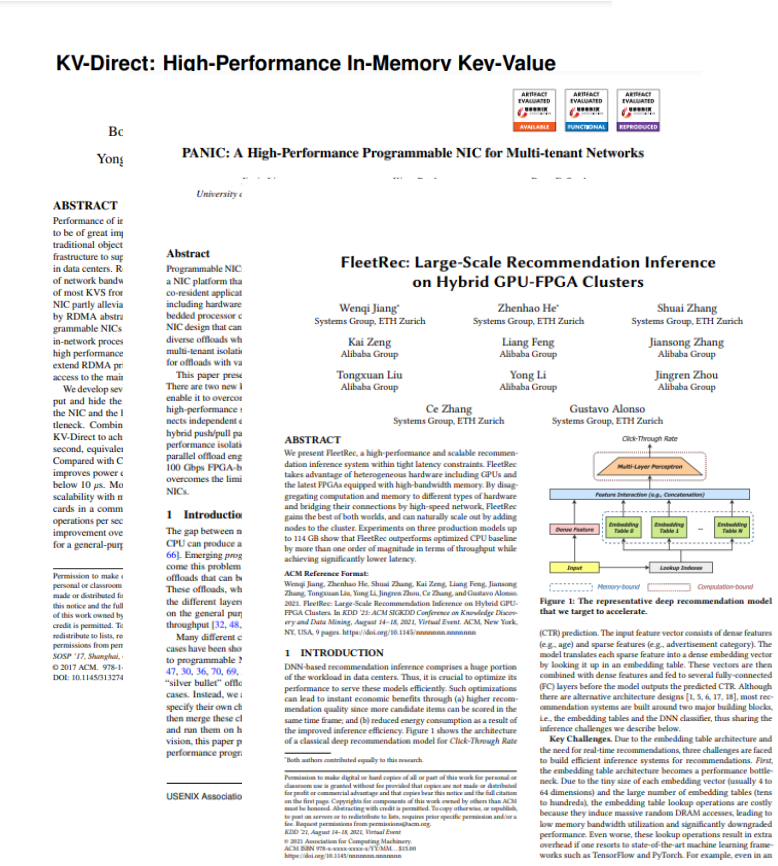
In-network processing

PANIC (OSDI’20), Corundum (FCCM’20)

Strom (EuroSys’20), Farview (CIDR’22)

Distributed computing

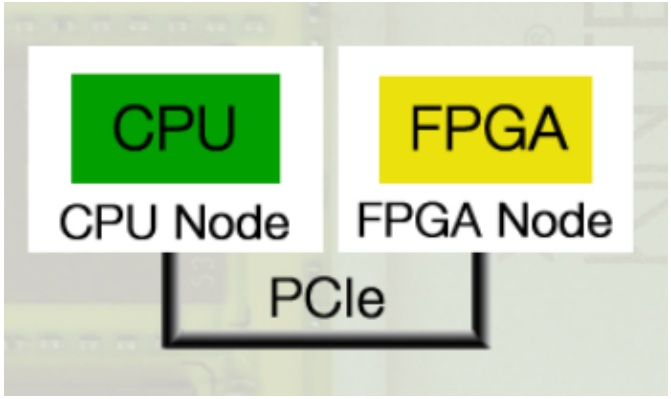
FleetRec (KDD’21), FPDeep (TC’19)

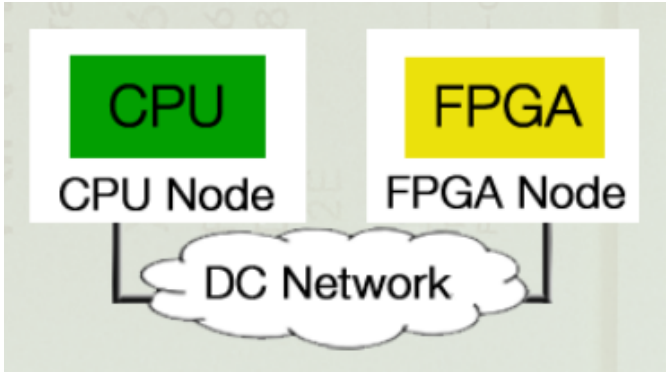


FPGAs in the context of cloud deployment is different…

From the previous examples, we see:

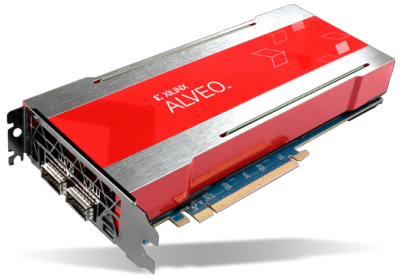
- FPGAs no longer viewed as slave accelerators
  - Connected to CPU host via PCIe
  - For traditional offload workload
- But as first-class citizen for data processing
  - Connected through DC network
    - Low latency
    - High throughput
  - Running DC communication protocol

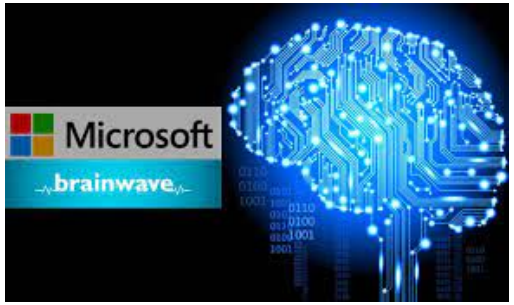




FPGAs ideal for networking + computation

- Direct processing of network data
  - In contrast to GPUs (or other accelerators)
- Pipelined architecture fits network stream processing
  - Finite state machines
- Overlapping networking and computation
  - Store-and-forward not required
  - Batching not required
- Example: Microsoft Brainwave
  - *https://www.microsoft.com/en-us/research/project/project-brainwave/*
  - Real Time DNN
  - Low latency, high efficiency
  - Served in real production





Challenges to build applications on FPGAs

- **Lack of open-source infrastructure for FPGAs**
  - Cloud infrastructure close-source, lacking testbed
  - Comparison to CPUs…
    - Rich APIs
    - Portability across platforms
    - Backward compatibility

- **Lack of support from commonly available development framework**
    - Abstracting data movement through PCIe
    - But not with network…

- **Lack of traditional operating system abstractions**
  - We are used to processes, threads …

- **Sometimes must first build the whole infrastructure before exploring in-network processing/distributed applications**
  - Think about previous research…

Summary of Challenges

- **Lack of open-source infrastructure**
- **Lack of traditional high-level abstractions and interfaces**

Tutorial: Resources for Distributed Applications on FPGA Clusters

- Target: Facilitate practitioner and researcher exploring distributed applications on FPGA clusters
  - **FPGA clusters (HACC)**
    - Data center standard infrastructure
  - **Frameworks and abstractions**
    - Shell support and abstractions for in-network processing, disaggregated computation, distributed applications …
  - **Systems and applications built on top**

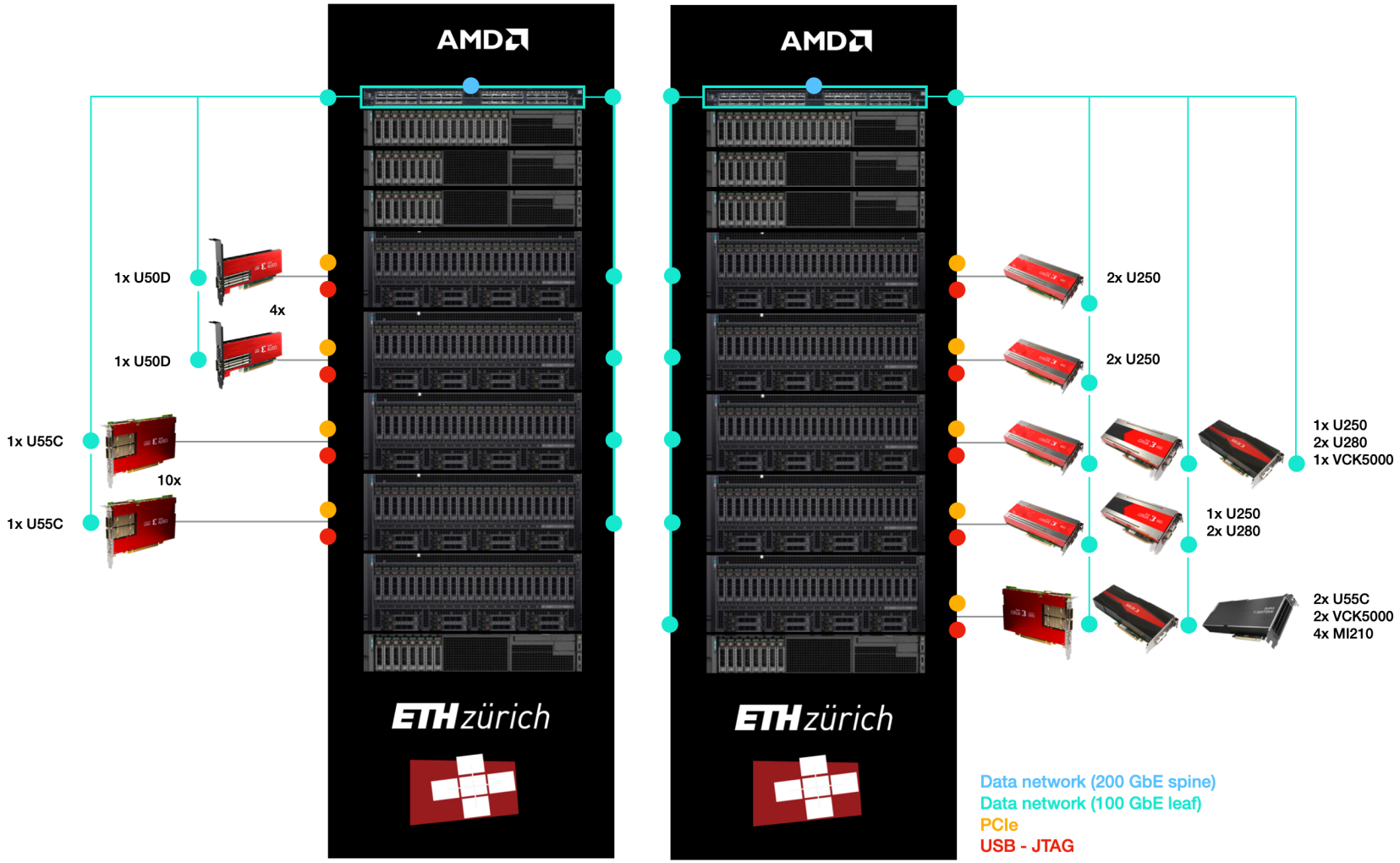
Two working flows:

Infrastructure – HACC cluster

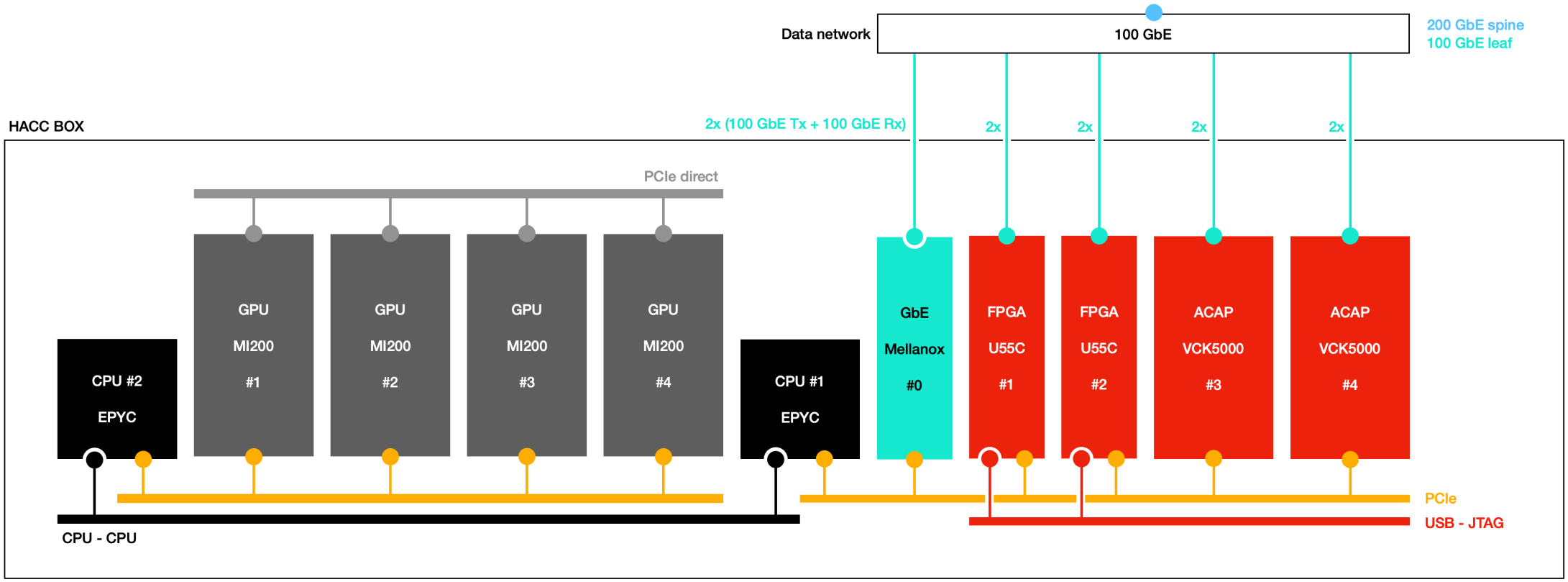
- The Heterogeneous Accelerated Compute Clusters (HACC) program is a unique initiative to support novel research in adaptive compute acceleration for data center settings and high-performance computing (HPC).

- ETH Zurich HACC
  https://systems.ethz.ch/research/data-processing-on-modern-hardware/hacc.html

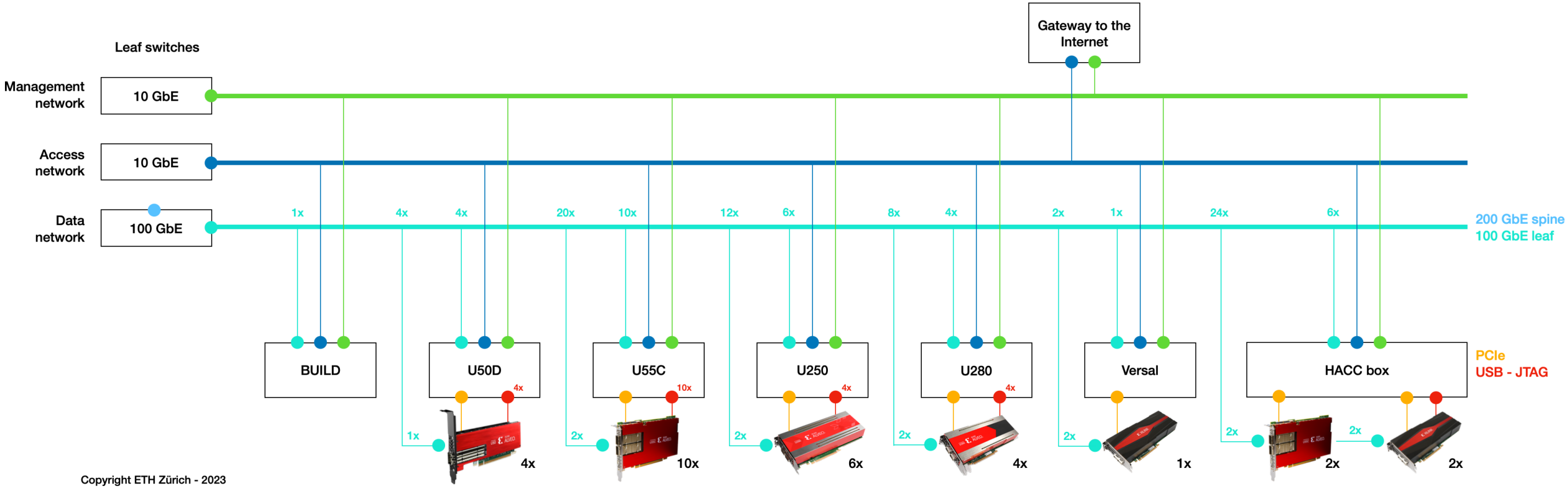
Introduction to HACC cluster



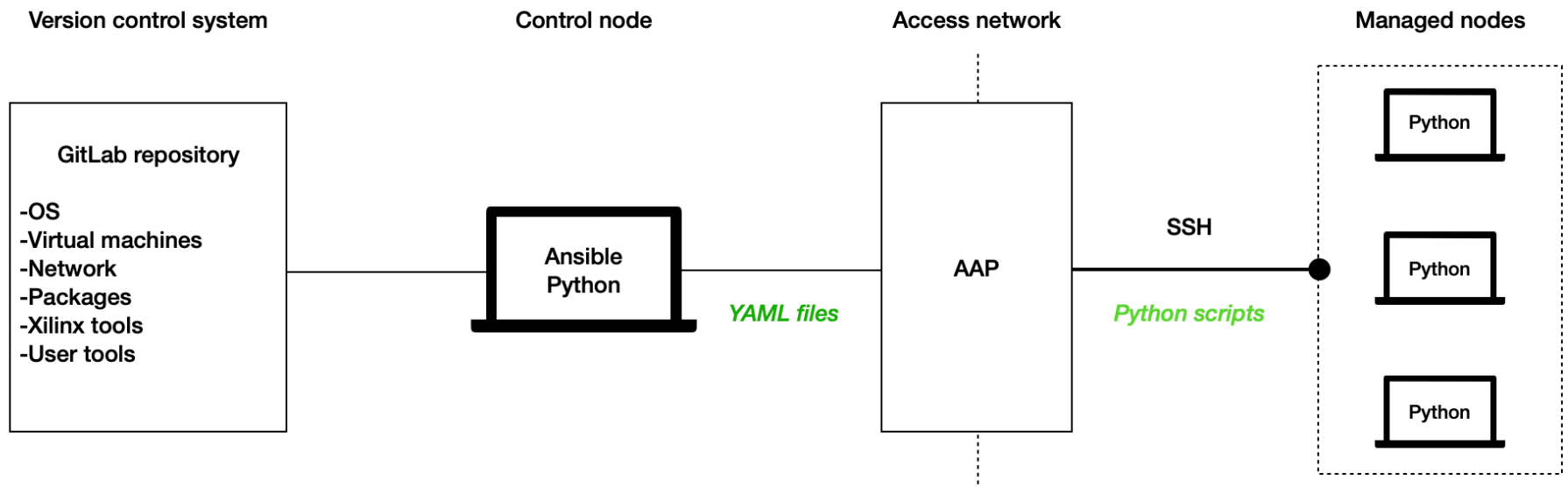
Overview (HACC boxes)



Overview



Operating the cluster



Infrastructure as Code (IaC)

-IaaS playbooks
-PaaS playbooks

Ansible Automation Platform (AAP) and pipelines

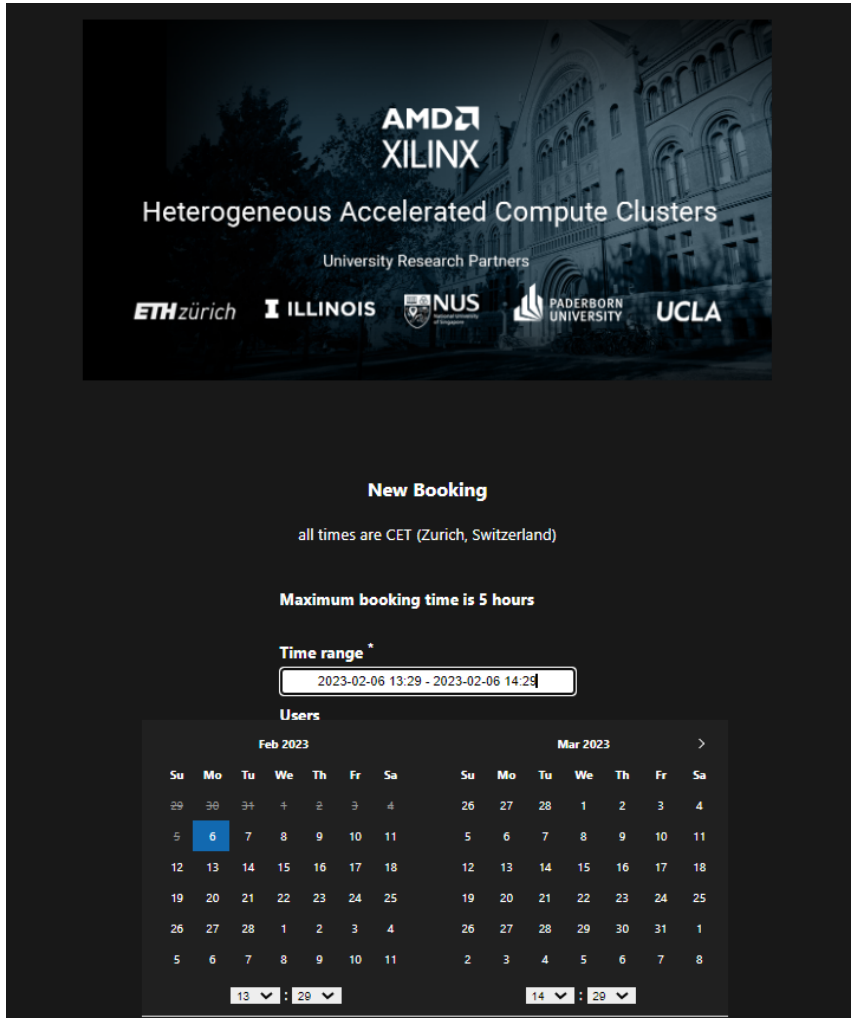
-Daily operations with GitLab pipelines
-On demand operations

Infrastructure

-Reset to a defined state
-Xilinx releases

Booking system

- Reserving a specific VM/device for a specific period
  - Maximum 5 hours per reservation
- During a reservation, only the selected user can connect to the VM/device
- User can choose different workflows when login
  - Vitis workflow
  - Coyote workflow



User access

- Access requires registration
  - ETH users contact Gustavo Alonso
  - All others through AMD Xilinx (HACC program)
  - Users get guest account at ETH (renewable)

Build server (no FPGA)

Dell Power Edge R740

- 2 x Intel Xeon Gold 6248 2,5 GHz, 20C/40T
- 12 x 32 GB DDR4
- 6 x 960 GB SSD
- **Mellanox Connect X-5, single port (100Gb)**
- Intel 10 Gbs card

Large server for compilation, project development, and support of cluster activities

Large enough to support many concurrent users

Nodes with Multiple FPGAs (Hypervisor)

Systems@ETH Zürich

Nodes with 2 FPGAs

Dell Power Edge R740

- **2 x Alveo U250**
- 2 x Intel Xeon Gold 6234 3,3 GHz, 8C/16T
- 12 x 32 GB DDR4
- 2 x 96GB SSD
- **2 x Mellanox Connect X-5, single port (100Gb)**
- Intel 10 Gbs card

Nodes with 3 FPGAs

Dell Power Edge R940

- **1 x Alveo U250 + 2 x Alveo U280**
- 2 x 2 x 2 x Intel Xeon Gold 6234 3,3 GHz, 8C/16T
- 24 x 16 GB DDR4
- 2 x 96GB SSD
- **2 x Mellanox Connect X-5, single port (100Gb)**
- Intel 10 Gbs card

Nodes with single FPGA (Bare-metal)

AMD EPYC

- 32 CPU cores
- 64 GB DDR4
- **Mellanox Connect X-5, single port (100Gb)**
- Intel 10 Gbs card
- **1 x Alveo U55C/U50D**

Fast transitioning between Vitis and Coyote workflows

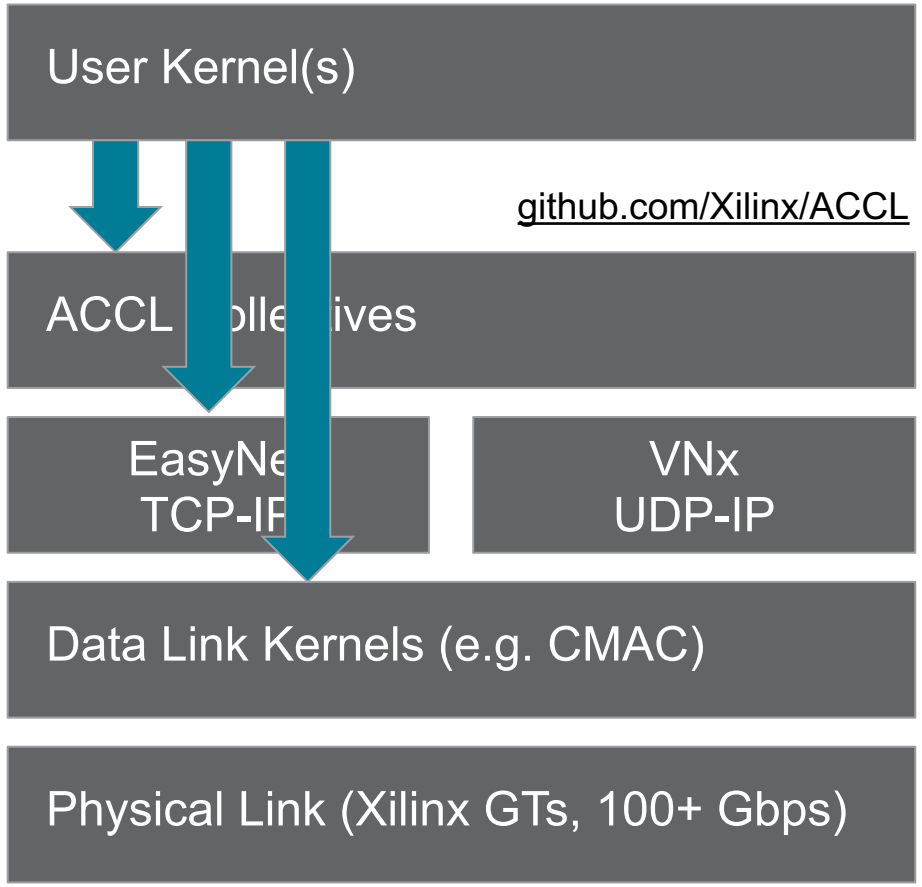
- Hot-plug boot
  - Use Linux capabilities to re-enumerate PCI devices on the fly
  - without the need for cold or warm rebooting of the system

- Cold boot
  - Powering off and on the machine
  - Resets all the hardware peripherals (including all PCI devices)
  - Xilinx accelerator cards: causes the base shell be flashed from PROM
  - This operation is required to revert a server to the Vitis workflow

- Warm boot
  - Restarts the system without interrupting the power
  - Xilinx accelerator cards: re-enumerates the number of PCI functions
  - This operation is required to bring a server to the bare-metal workflow

Vitis flow

- **Data Link Kernels**
  - Ethernet or Aurora
  - Wraps MRMAC (Versal) or CMAC (UltraScale) and provides AXI Stream interface to upper layers
  - Limited reliability support e.g. FEC
- **Network (IP) and Transport Kernels**
  - UDP with VNx – low footprint, unreliable
  - TCP with EasyNet – higher footprint, reliable
- **Collective Offload Kernel(s)**
  - ACCL
- All kernels are Vitis-compatible, portable across Alveo range
- Full freedom to construct application on top of any layer



github.com/Xilinx/ACCL

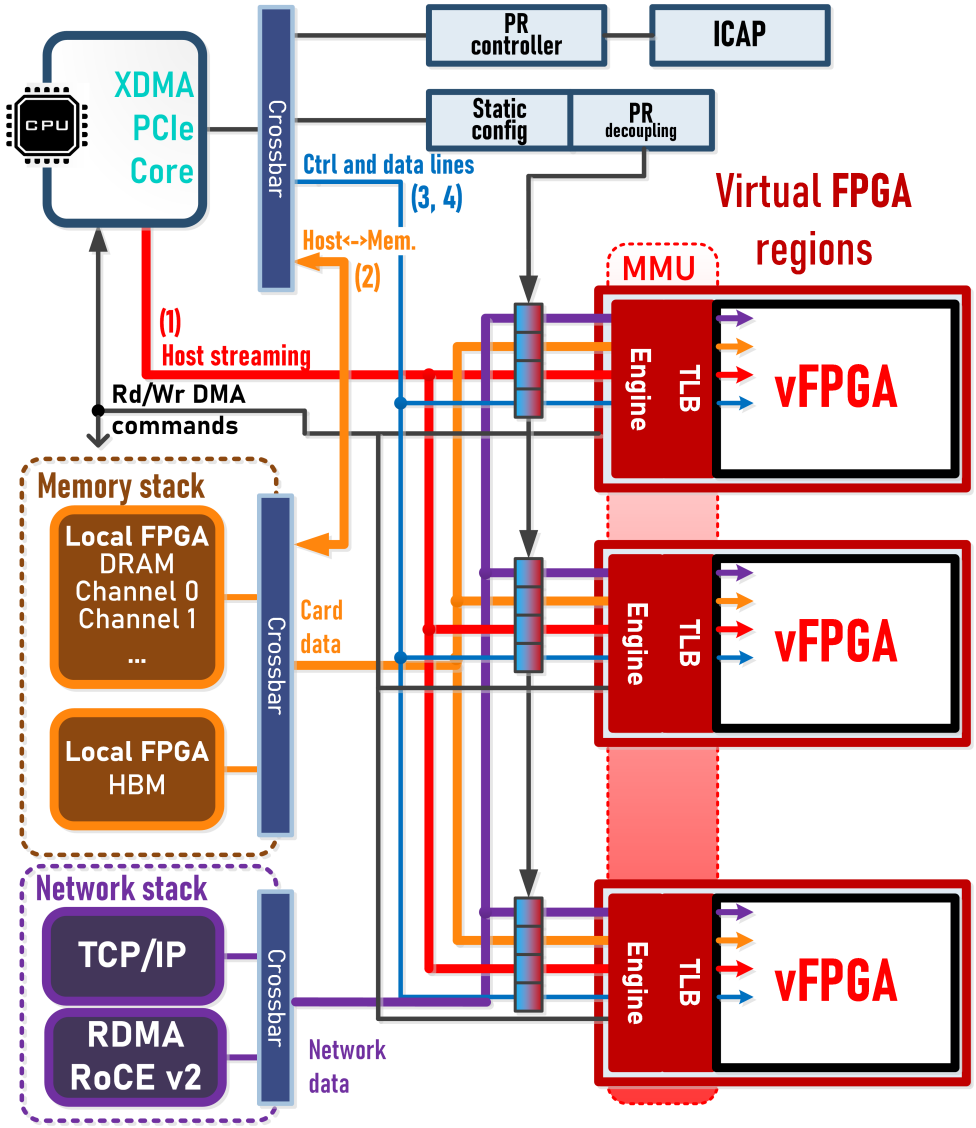
github.com/Xilinx/xup_vitis_network_example

github.com/fpgasystems/Vitis_with_100Gbps_TCP-IP

github.com/Xilinx/AlveoLink

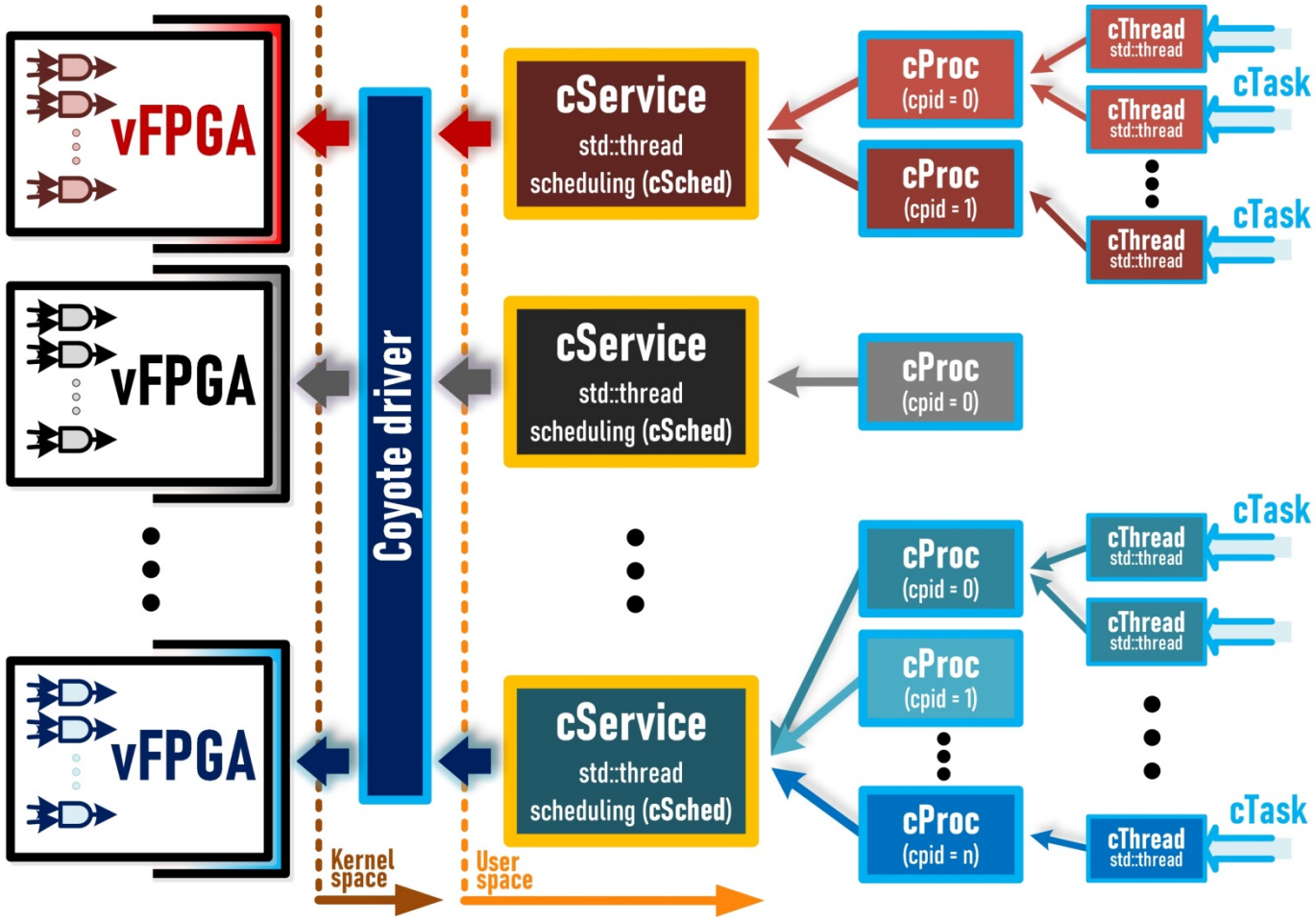
Coyote flow

- **Microkernel for FPGAs**
  - Multiple isolated untrusted vFPGAs
  - Spatial and temporal sharing (PR)
  - Dynamic reconfiguration (scheduler)
  - Virtualized memory
  - Unified memory (HMM)
  - HBM and DRAM striping service
  - Shared TCP/IP service
  - Shared RDMA service
  - Unified logic interface -> portability
  - RTL and HLS support
  - Runs on u50, u200, u250, u280, u55c, vcu118, *Enzian*



Coyote flow

- **User space abstractions**
  - **cSched**- Coyote scheduler, reconfiguration controller
  - **cProcess** - Coyote process, multiple can run within a single vFPGA
  - **cThread** - Coyote thread, multiple can run within a single cProc. Task level parallelism
  - **cTask** - Coyote task, arbitrary user variadic function executed by cThreads
  - **cService** - Coyote library daemon, background service, UDS for IPC



Systems@ETH zürich