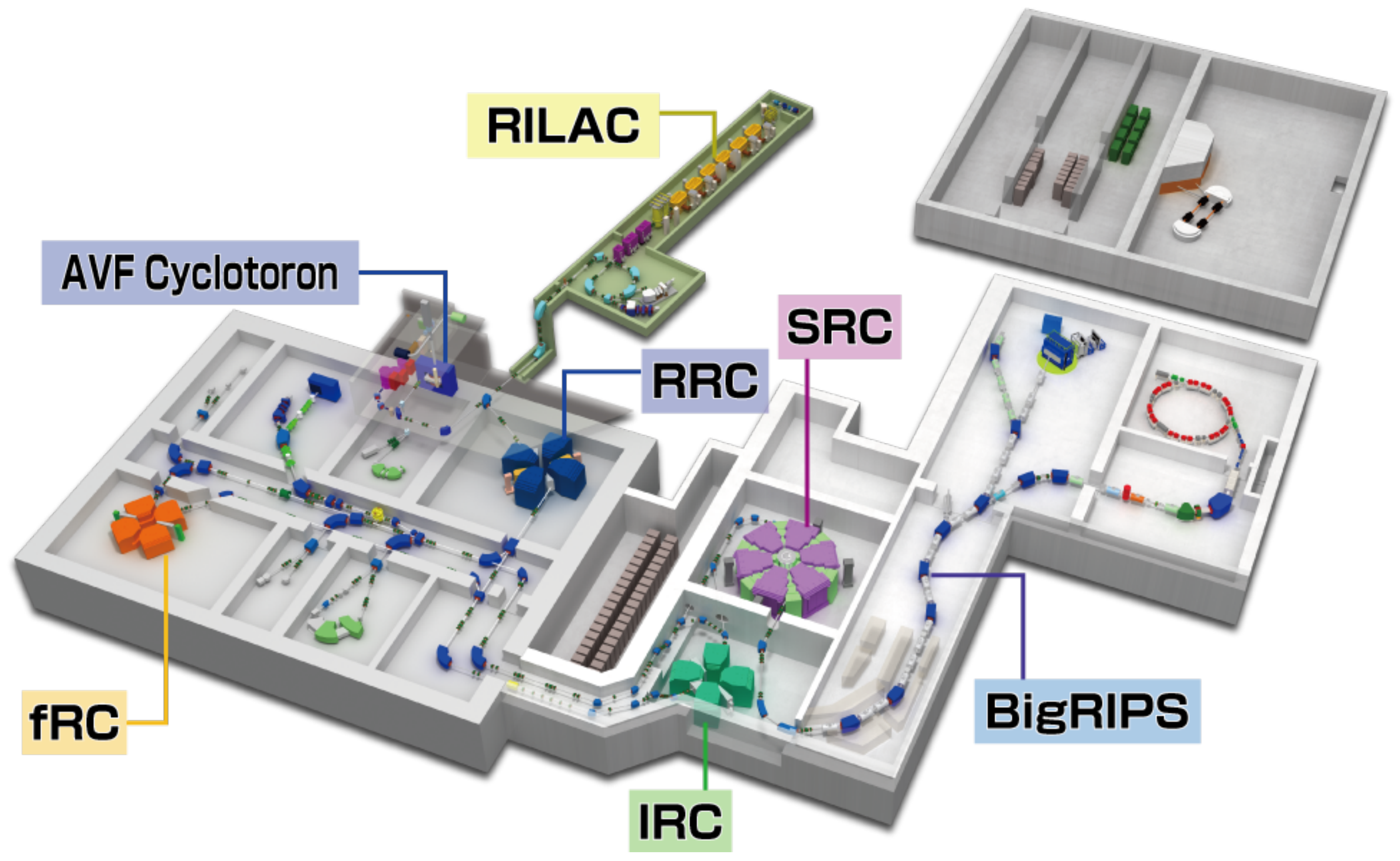
High-speed data processing in the RIBF DAQ system using the Alveo data-center accelerator card

Yuto Ichinohe (RIKEN Nishina Center)

Hidetada Baba (RIKEN Nishina Center), Shoko Takeshige (Rikkyo Univ.), Taku Gunji (CNS)

24th IEEE Real Time Conference — ICISE, Quy Nhon, Vietnam
2024.4.23

RIKEN Radioactive Isotope Beam Factory (RIBF)

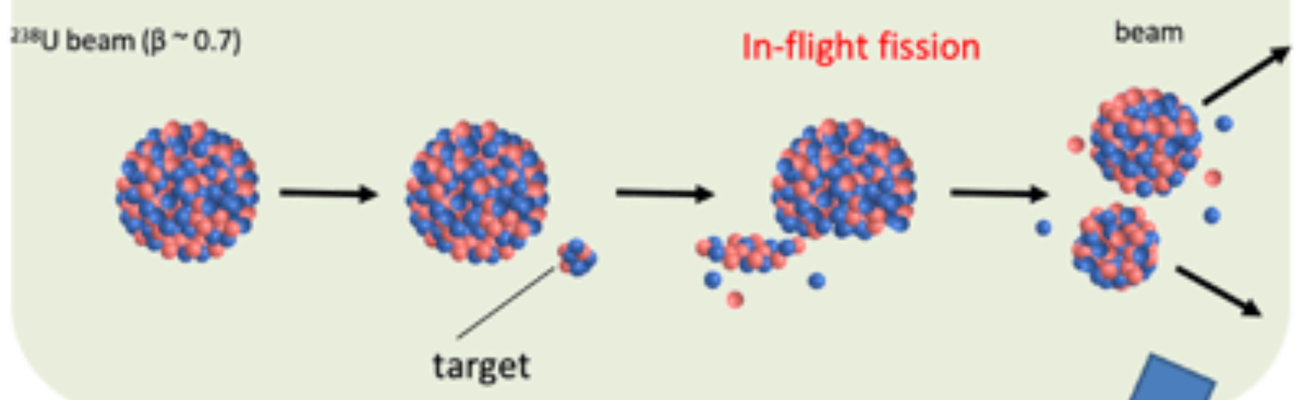


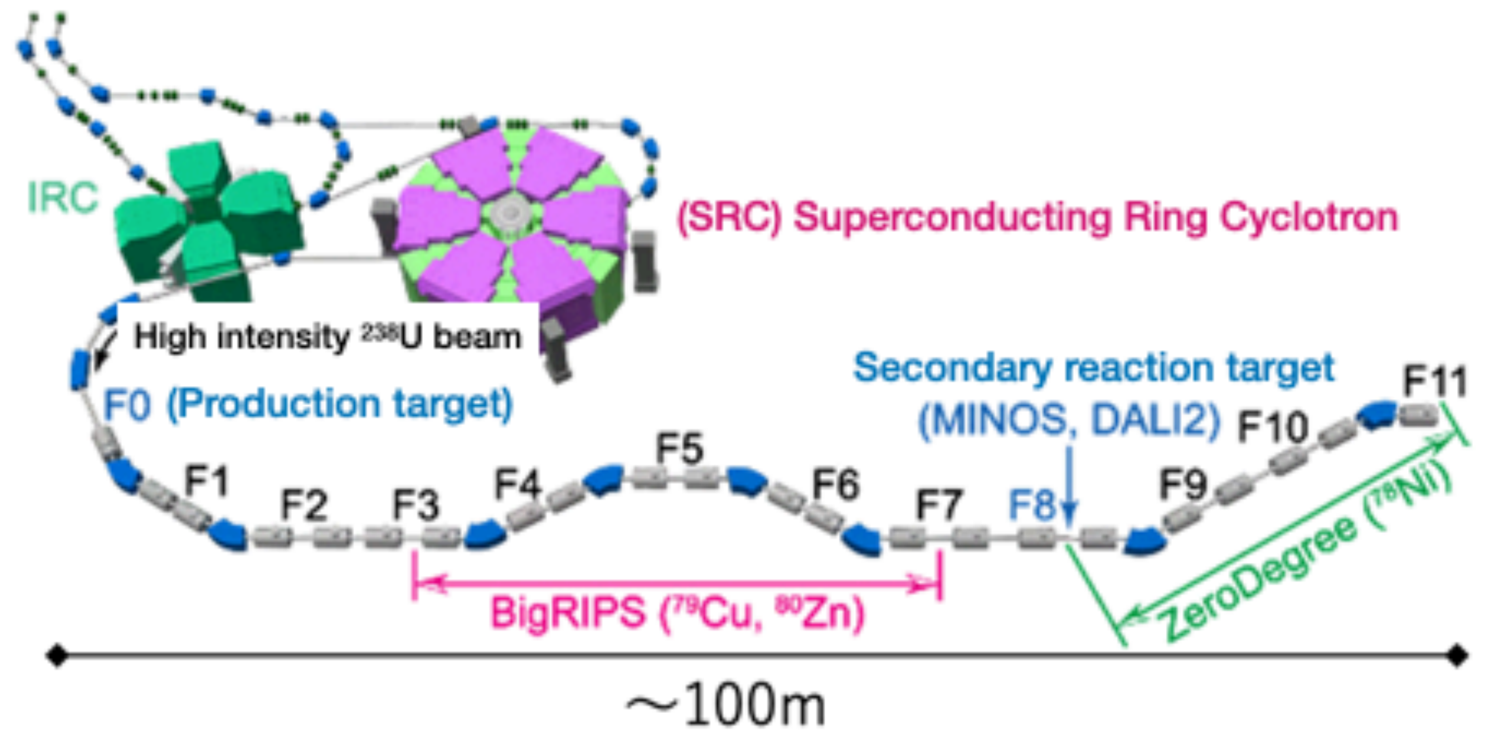
1. **Highest intensity RI beam**
    - secondary beam intensity: < 1e7 cps
    - mixture of multiple species of RIs

2. **RI Identification**
    - BigRIPS (RI fragment separator)

3. **Physics measurement**

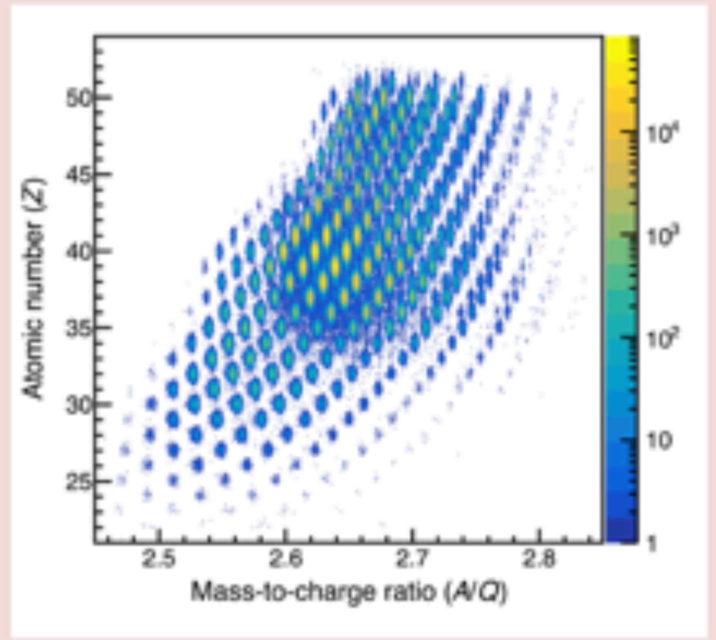
Production of RI beam (highest intensity)





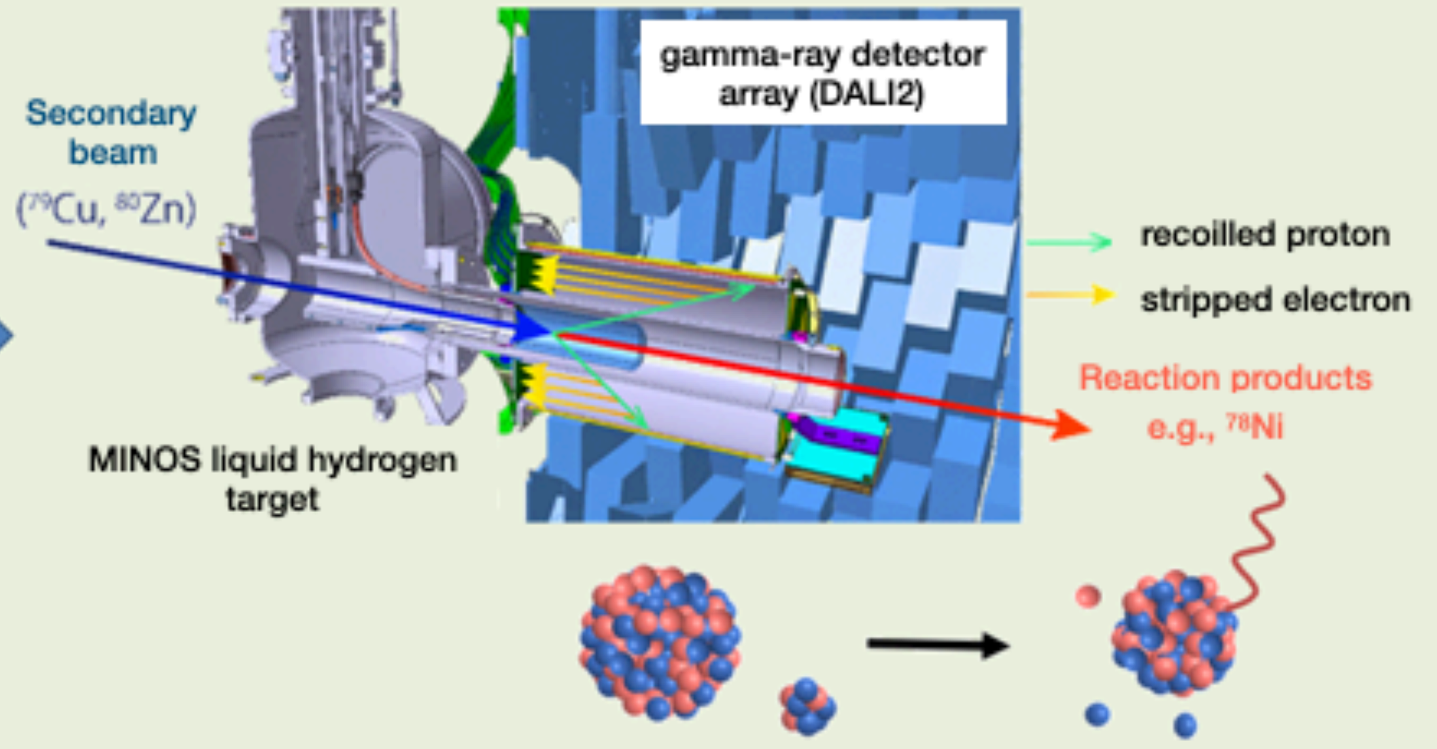
IRC
(SRC) Superconducting Ring Cyclotron
High intensity ^{238}U beam
F0 (Production target)
Secondary reaction target
(MINOS, DALI2)
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
BigRIPS (^{79}Cu, ^{80}Zn)
ZeroDegree (^{78}Ni)
~100m

Particle Identification



Beamline detectors to identify various RIs

Nuclear reaction measurement

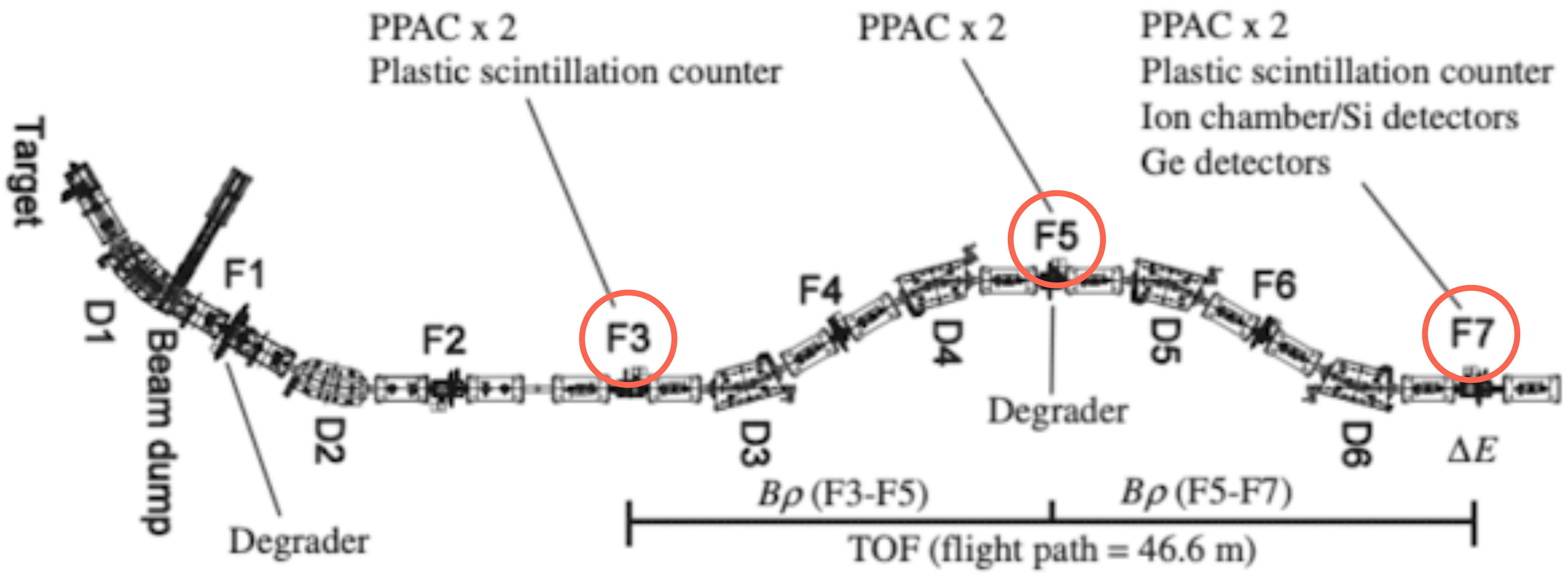


Secondary beam (^{79}Cu, ^{80}Zn)
gamma-ray detector array (DALI2)
recoilled proton
stripped electron
Reaction products e.g., ^{78}Ni
MINOS liquid hydrogen target

RIs transported to main detectors

BigRIPS

RI beam **P**rojectile-fragment **S**eparator

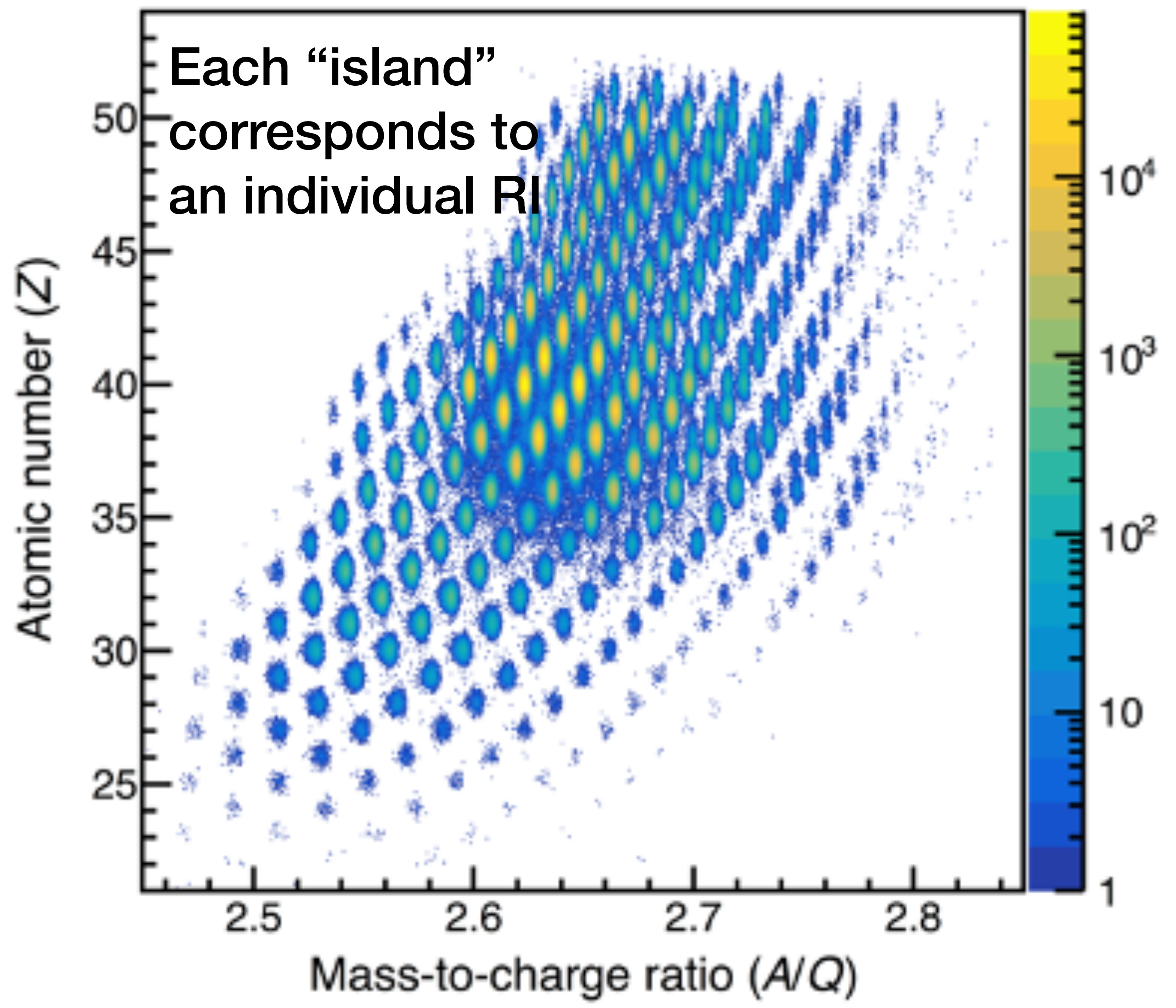


Beamline detectors for PID
- Plastic scintillator (F3, F7)
- PPAC (F3, F5, F7)
- Ion chamber (F7)

Concept of the PID using BigRIPS
- TOF between F3 Plastic and F7 Plastic → β
- Particle transfer from F3 PPAC to F5 PPAC → Bρ
- Energy loss in F7 IC + β → Z
- Bρ + β → A/Q

Extract desired RIs using the particle identification diagram for physics measurement (offline analysis)

Particle identification diagram



Real-time data processing in RIBF DAQ

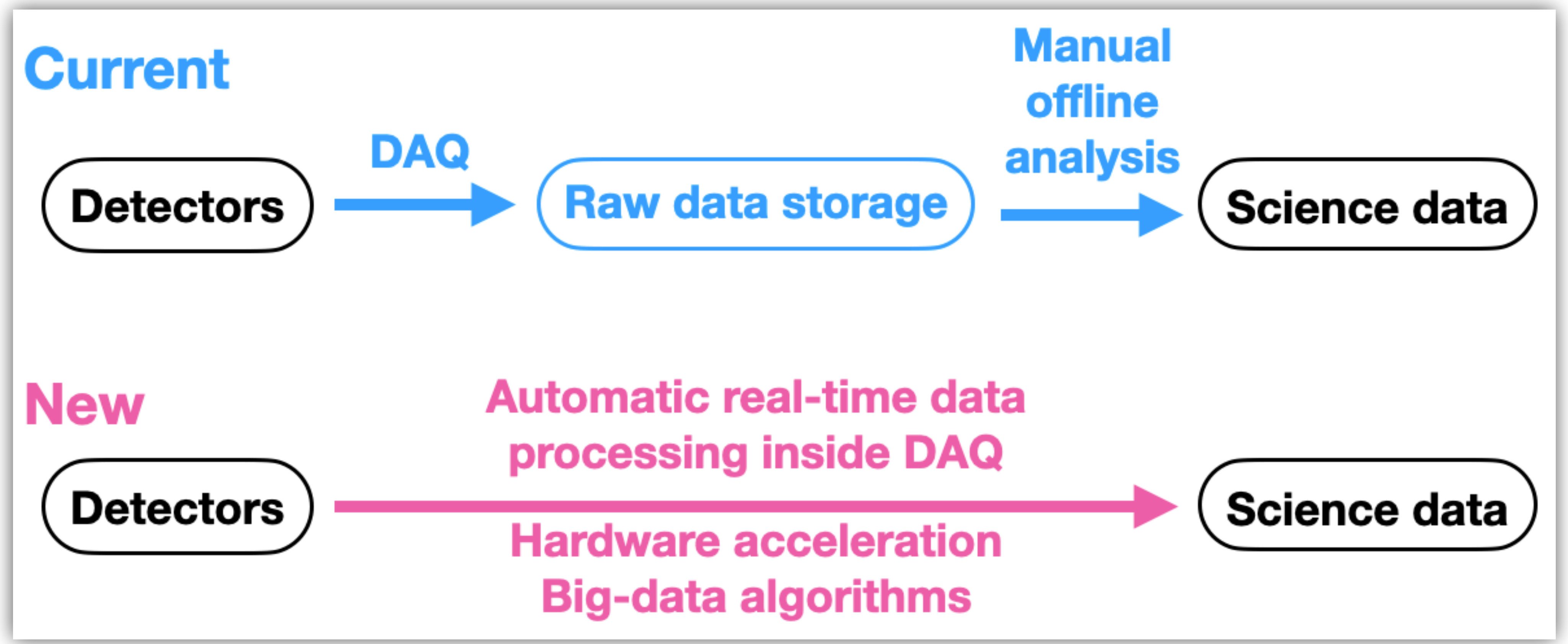
Goal: Streaming "physical" quantities

e.g., TOF, Position, ΔE (calibrated, analyzed), PID info

1. Speed-up of data analysis
   - Currently, the same procedure is performed individually by each experimental group (redundant) → standard PID without overheads

2. More "physical" triggers
   - Currently, simple discriminator triggers are mostly used (low level) → trigger based on e.g., PID information

3. Easier simultaneous multiple experiments
   - Currently, PID DAQ system can only be used exclusively (inefficient) → the official PID DAQ stream which can be subscribed by multiple experimental groups at the same time



What kind of hardware is suitable?

- FPGA may be the choice for real-time analysis of streaming data
- Manually implementing a complicated task such as PID in FPGA with HDLs is nightmare …

→ **AMD (Xilinx) Alveo series**

AMD (Xilinx) Alveo series

"Adaptable Accelerator Cards for Data Center Workloads"

- Enhancing the host server capability with FPGA through easily installable PCI Express interface

Alveo U50 (~$3500)

- Parallelly accessible 8GB (256MB x 32) HBM
    - High-bandwidth, large data can be stored close to FPGA
- Direct external connectivity with a QSFP28 port

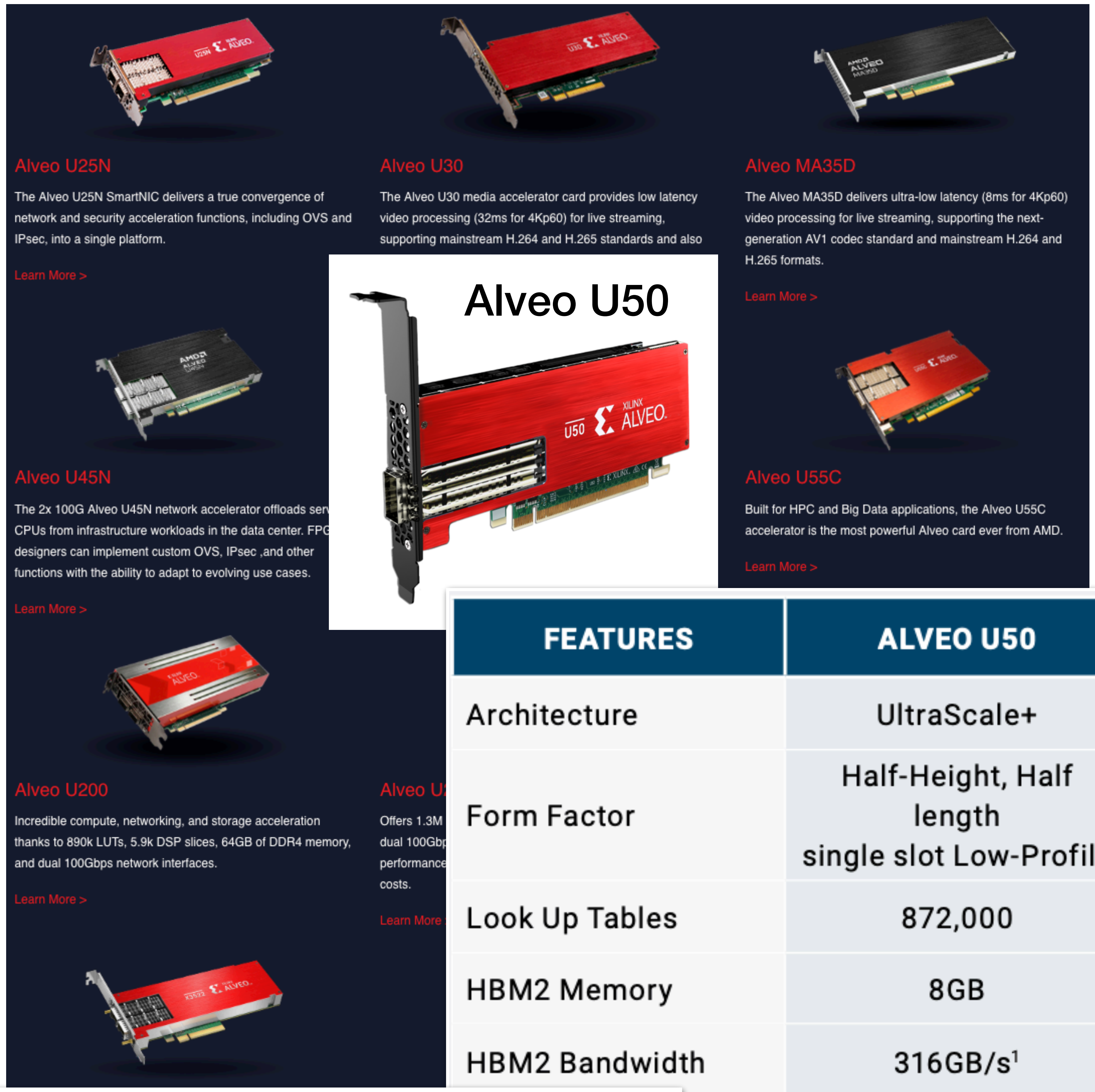
Vitis Unified Software Platform

- Covers most of the development flow of applications that invokes FPGA kernels from the host CPU (C++ simulation, HLS, RTL / C++ co-simulation)

<u>1. Most of the application framework are provided</u>

<u>2. C++ codes are automatically converted to RTL by HLS</u>

- Users can focus only on thinking how to exploit the FPGA power and writing C++ codes



Alveo U25N

The Alveo U25N SmartNIC delivers a true convergence of network and security acceleration functions, including OVS and IPsec, into a single platform.

Learn More >

Alveo U30

The Alveo U30 media accelerator card provides low latency video processing (32ms for 4Kp60) for live streaming, supporting mainstream H.264 and H.265 standards and also

Alveo MA35D

The Alveo MA35D delivers ultra-low latency (8ms for 4Kp60) video processing for live streaming, supporting the next-generation AV1 codec standard and mainstream H.264 and H.265 formats.

Learn More >

Alveo U45N

The 2x 100G Alveo U45N network accelerator offloads serv CPUs from infrastructure workloads in the data center. FPG designers can implement custom OVS, IPsec ,and other functions with the ability to adapt to evolving use cases.

Learn More >

Alveo U55C

Built for HPC and Big Data applications, the Alveo U55C accelerator is the most powerful Alveo card ever from AMD.

Learn More >

Alveo U200

Incredible compute, networking, and storage acceleration thanks to 890k LUTs, 5.9k DSP slices, 64GB of DDR4 memory, and dual 100Gbps network interfaces.

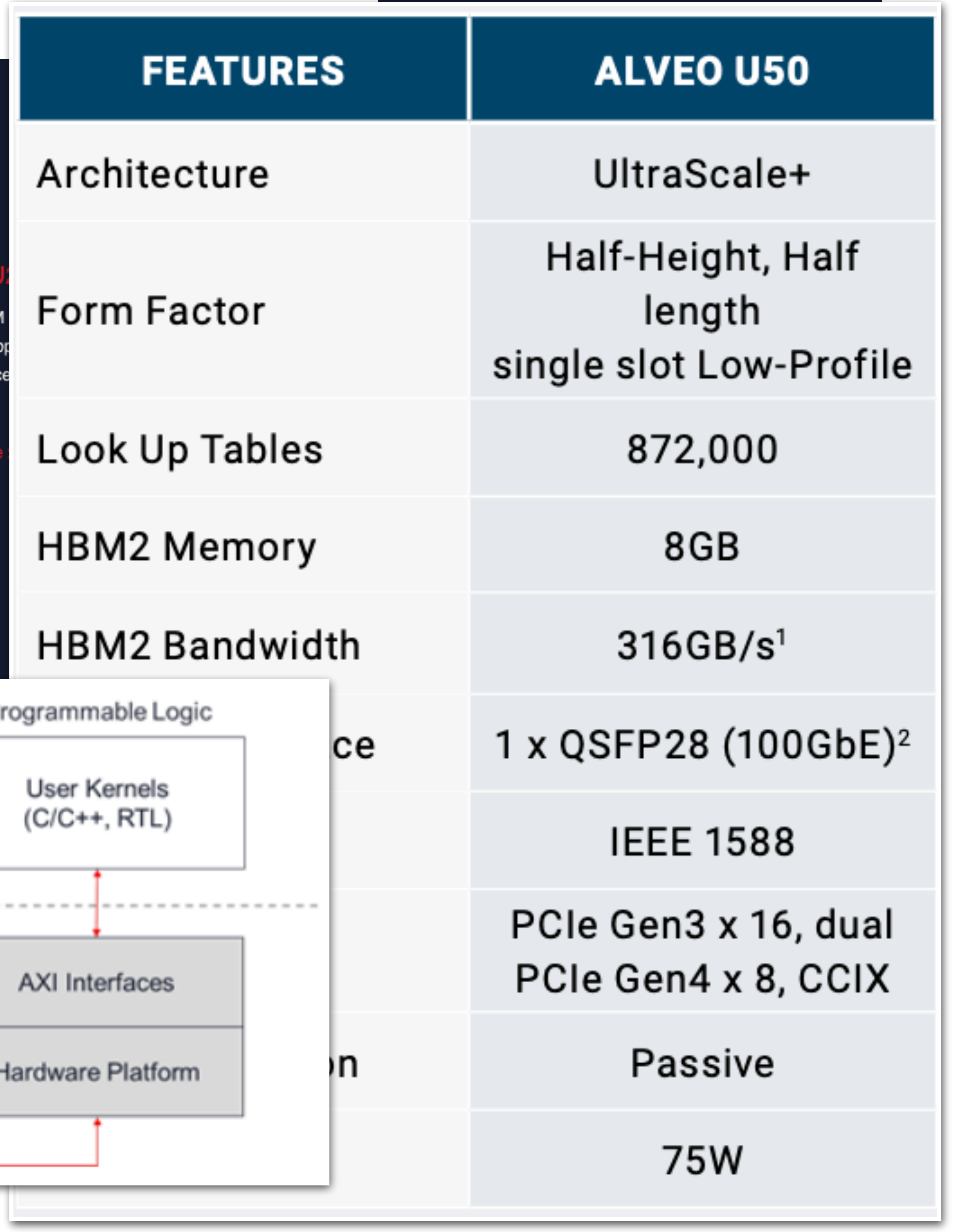
Learn More >

Alveo U

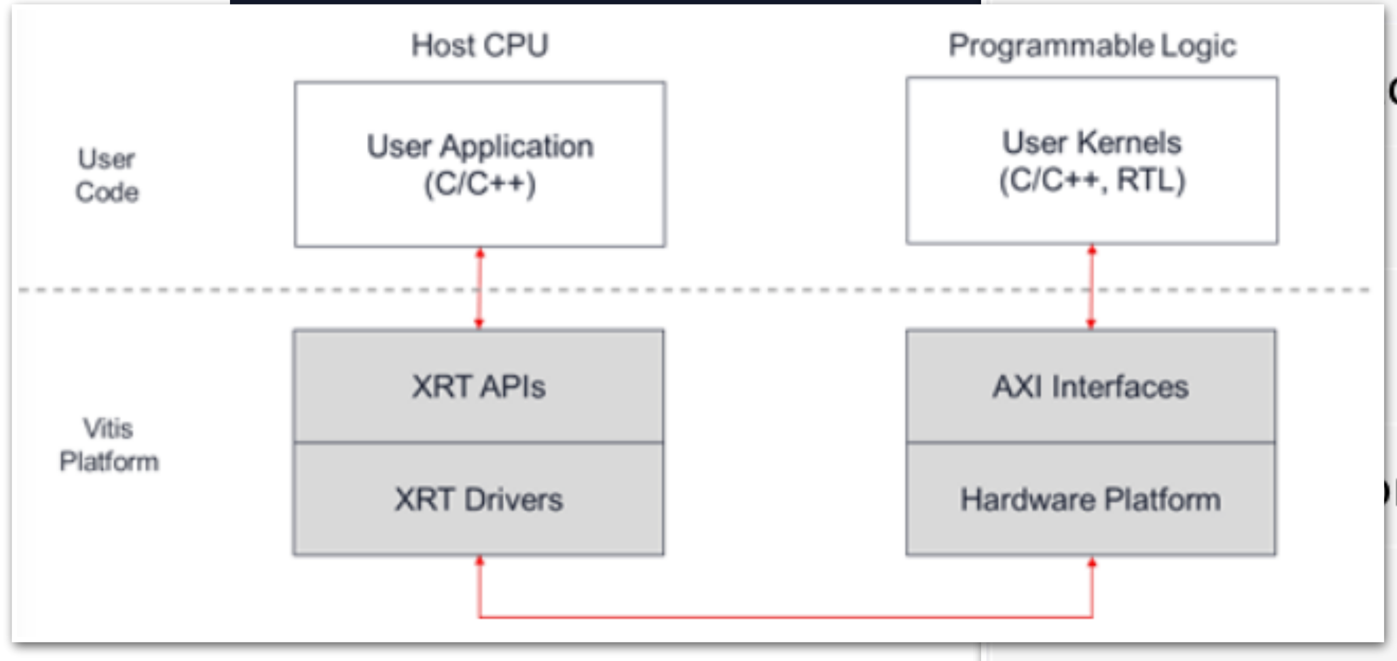
Offers 1.3M dual 100Gbp performance costs.

Learn More

Alveo U50



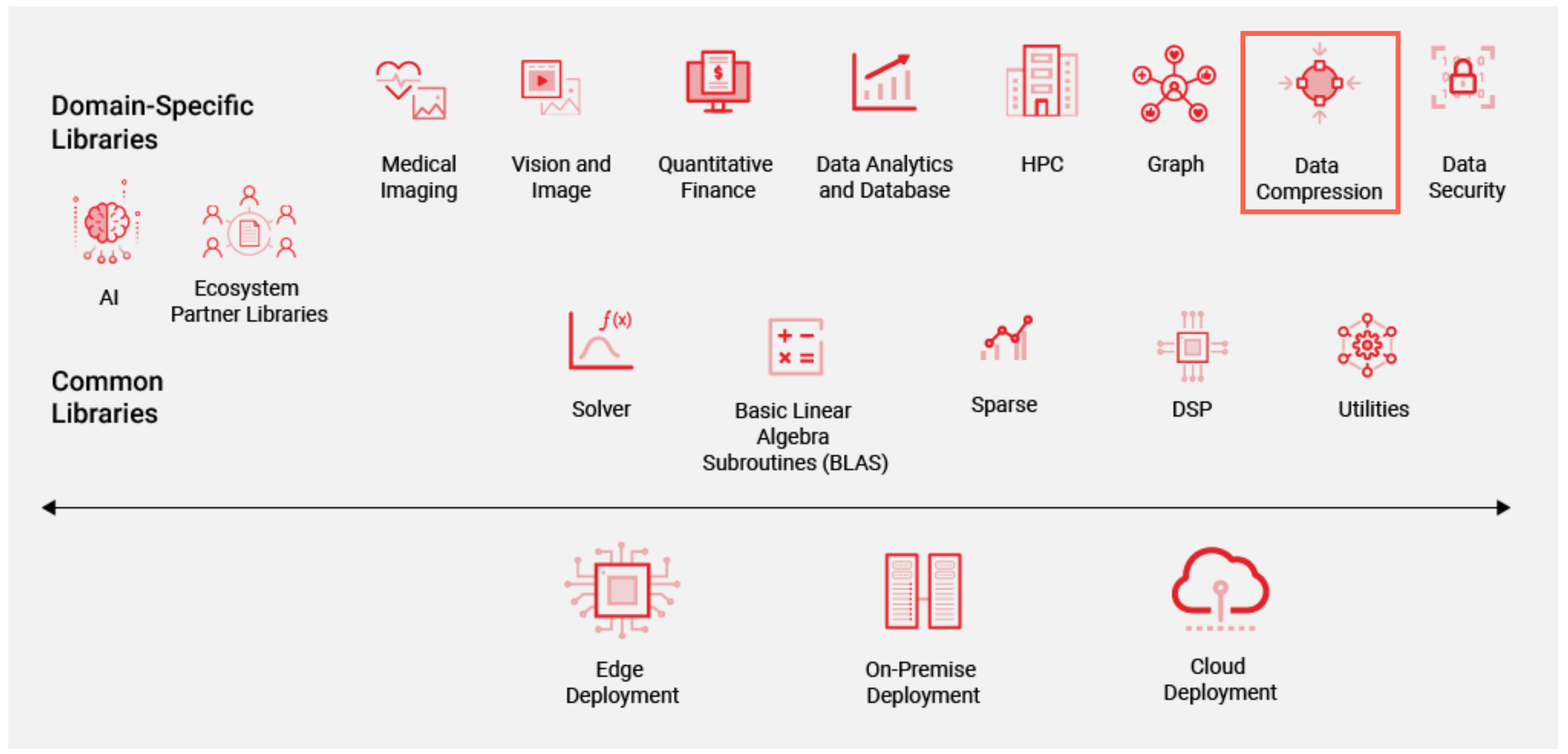
FEATURES	ALVEO U50
Architecture	UltraScale+
Form Factor	Half-Height, Half length single slot Low-Profile
Look Up Tables	872,000
HBM2 Memory	8GB
HBM2 Bandwidth	316GB/s[1]
ce	1 x QSFP28 (100GbE)[2]
	IEEE 1588
	PCIe Gen3 x 16, dual PCIe Gen4 x 8, CCIX
n	Passive
	75W



Benchmark: Hardware acceleration of GZIP

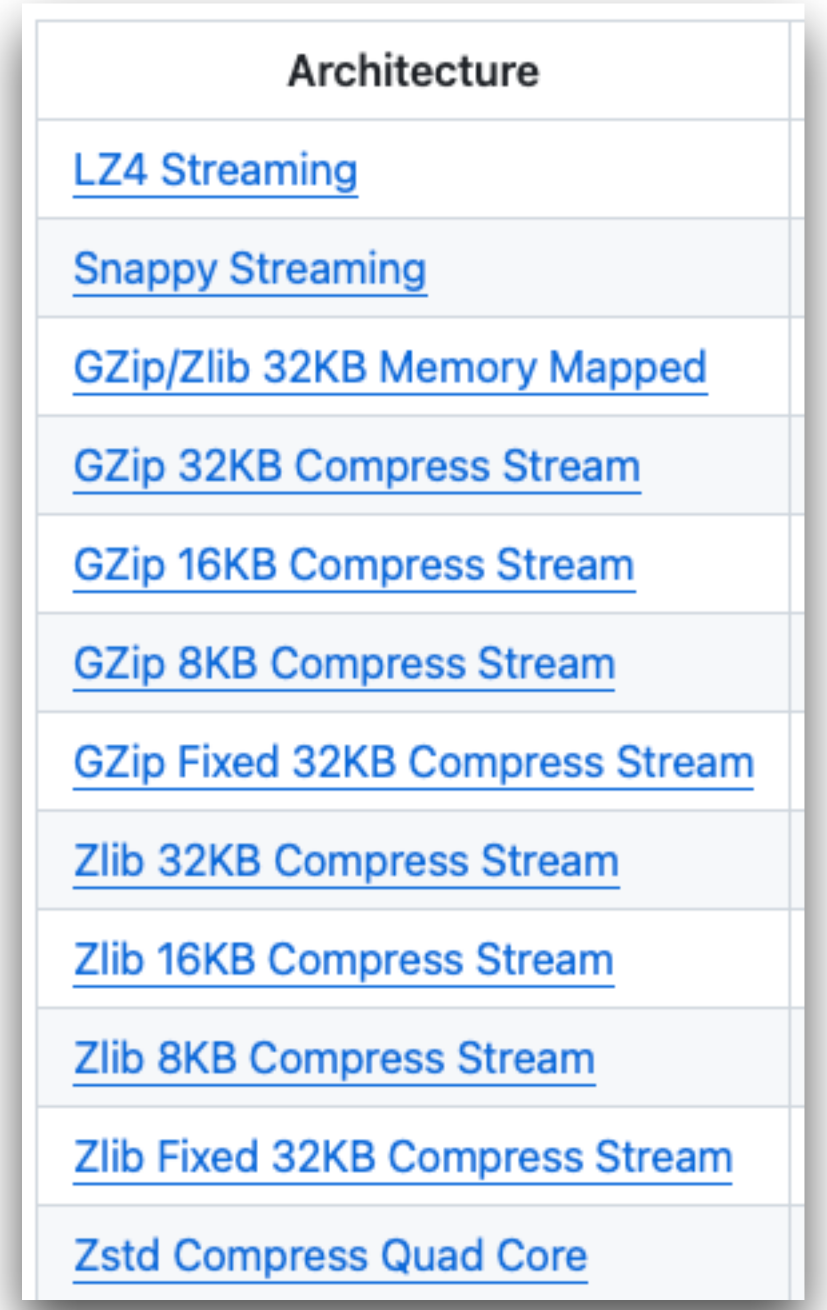
Sample hardware-accelerated codes of major tasks/libraries are available

- BLAS, Data science (random forest, SVM, K-means), compression, Matrix decomposition etc…



Example: **GZIP compression**

- Powerful compression
  - ~12 sec for 2.5 GB compression
  - cf. ~75 sec with 3.7 GHz Core i9
- Decompression is slower than CPU…
- Data size limited by HBM (compressed + decompressed < 8 GB)



Architecture
LZ4 Streaming
Snappy Streaming
GZip/Zlib 32KB Memory Mapped
GZip 32KB Compress Stream
GZip 16KB Compress Stream
GZip 8KB Compress Stream
GZip Fixed 32KB Compress Stream
Zlib 32KB Compress Stream
Zlib 16KB Compress Stream
Zlib 8KB Compress Stream
Zlib Fixed 32KB Compress Stream
Zstd Compress Quad Core

Confirmed the effect of hardware acceleration (although there are room for improvements…) → **Practical application**

RIBF PID using Alveo

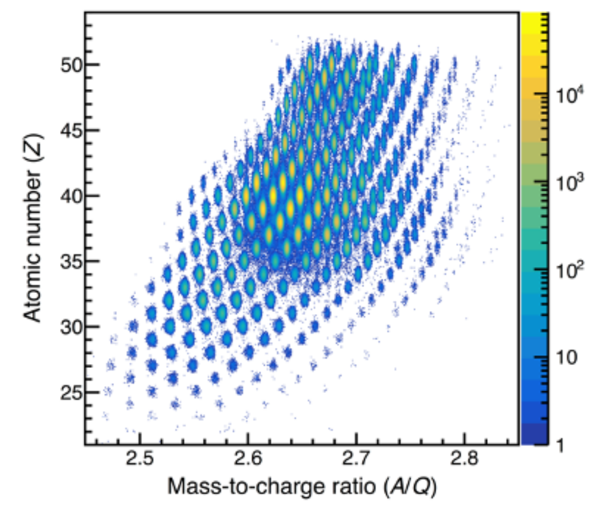
Tentative goal: Reproducing PID results identical to those derived by *anaroot*

(standard software for the RIBF data analysis)

Formulation (TOF-Bρ-ΔE method)

1. Input: raw data segment of *PPAC3, PPAC5, PPAC7, PL3, PL7, IC7*

2. Output: two double values corresponding to *A/Q & Z*

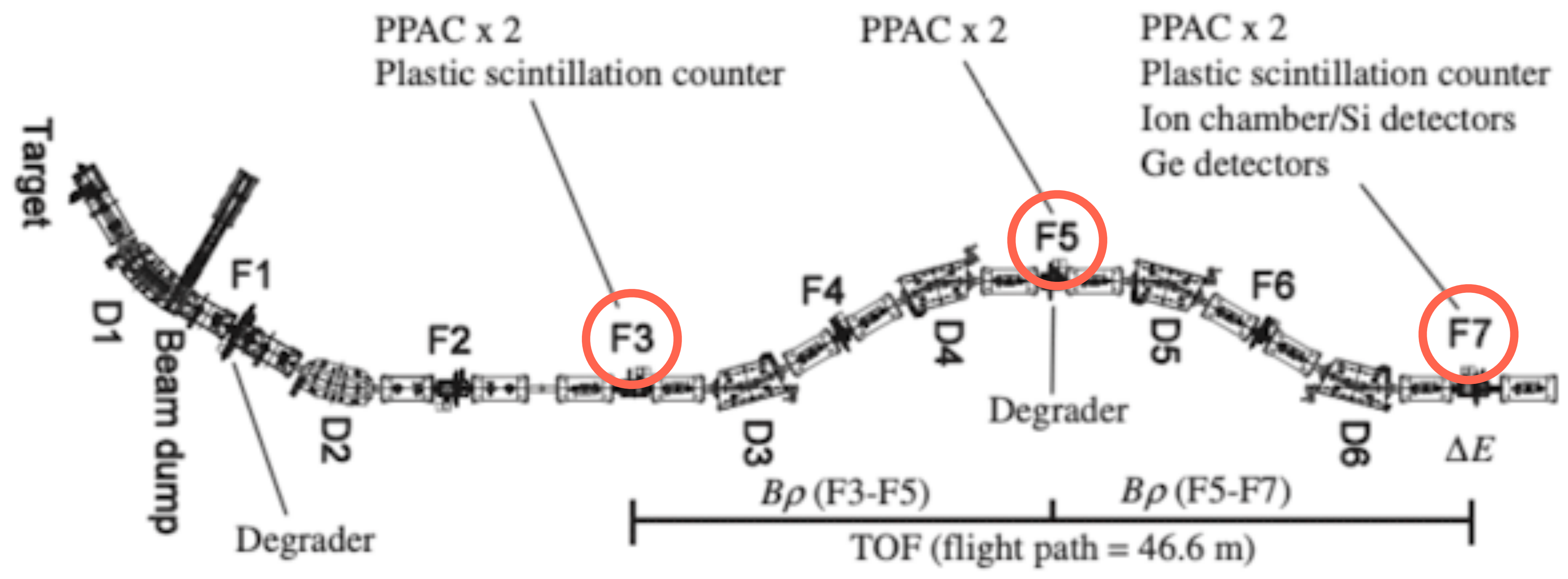


3. PPAC: 4 PPACs / FP (F3, F5, F7)
- Raw data → positions of interaction
- interaction positions + detector positions → position & direction of charged particles (least square) at each focal plane
- Particle transfer between two focal planes → **Bρ**

4. Plastic scintillator: 2 PMs / FP (F3, F7)
- passage time of RI (average of two PMs)
- passage time difference → TOF → **β**

5. Ion chamber: 6 ICs / FP (F7)
- **ΔE** (correct for pedestal + geometric mean + linear transformation)

6. PID
- β + Bρ → **A/Q**
- ΔE + β + Bethe-Bloch formula → **Z**



High-level synthesis overview

Starting from C++ codes based on *anaroot*…

<u>Step 1: Refactoring of the C++ codes such that the codes conform to the specification of HLS toolkit</u>

- ROOT dependency is removed
- Dynamic memory allocation is removed

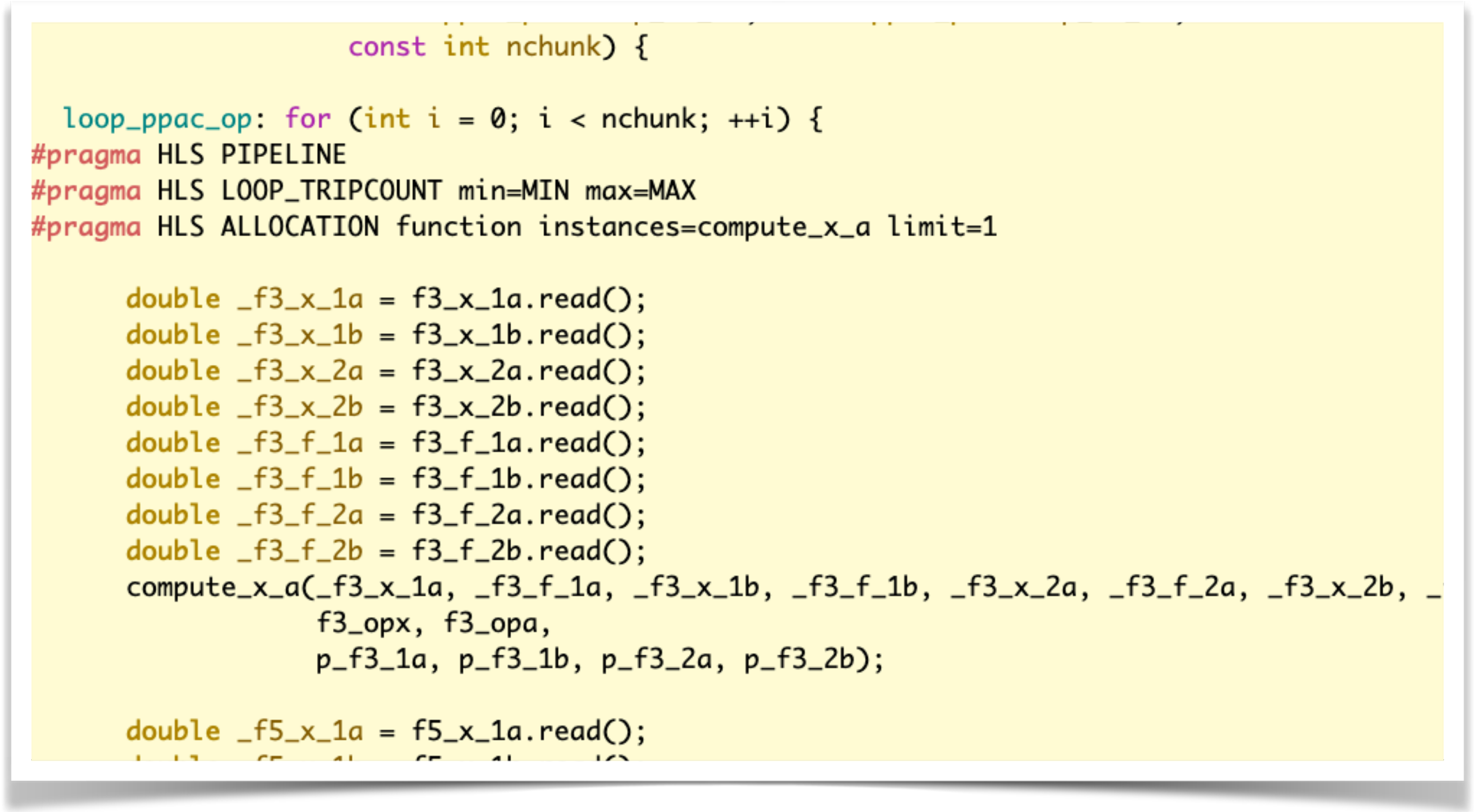
<u>Step 2: Tuning the C++ codes to assist the toolkit in inducing efficient RTL codes</u>

- Adding compiler directives (e.g., HLS PIPELINE: making a for loop pipelined, HLS INLINE: making a function in-line)
- Dataflow — splitting a task into smaller sub-tasks and connect them using pipeline registers (assisting task-parallelization)

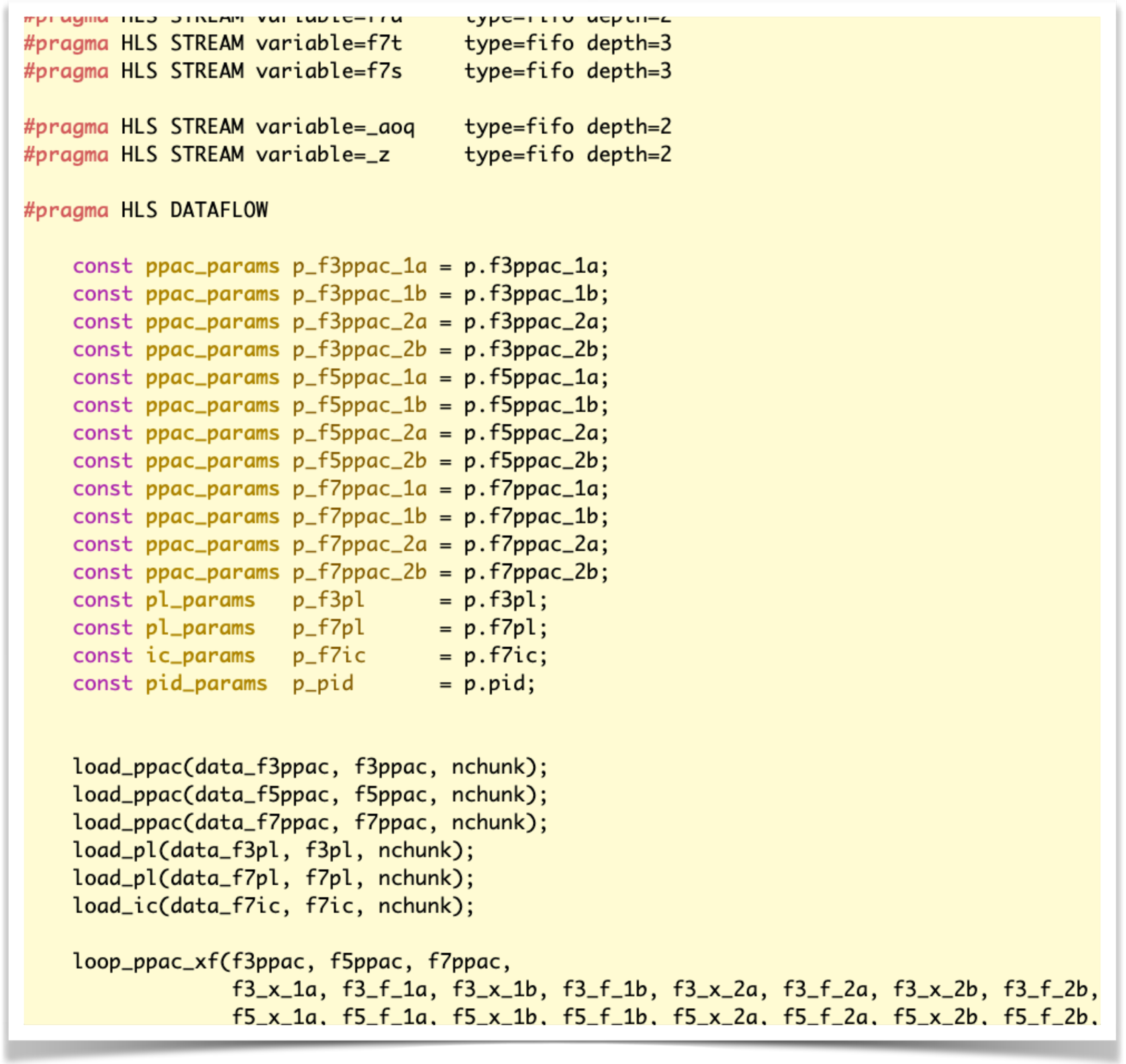
<u>Step 3: Converting the codes into RTL codes using the Vitis HLS toolkit</u>

RTL codes can be obtained

- can be used as same as those generated from HDL codes



```
                  const int nchunk) {

  loop_ppac_op: for (int i = 0; i < nchunk; ++i) {
#pragma HLS PIPELINE
#pragma HLS LOOP_TRIPCOUNT min=MIN max=MAX
#pragma HLS ALLOCATION function instances=compute_x_a limit=1

      double _f3_x_1a = f3_x_1a.read();
      double _f3_x_1b = f3_x_1b.read();
      double _f3_x_2a = f3_x_2a.read();
      double _f3_x_2b = f3_x_2b.read();
      double _f3_f_1a = f3_f_1a.read();
      double _f3_f_1b = f3_f_1b.read();
      double _f3_f_2a = f3_f_2a.read();
      double _f3_f_2b = f3_f_2b.read();
      compute_x_a(_f3_x_1a, _f3_f_1a, _f3_x_1b, _f3_f_1b, _f3_x_2a, _f3_f_2a, _f3_x_2b, _
                  f3_opx, f3_opa,
                  p_f3_1a, p_f3_1b, p_f3_2a, p_f3_2b);

      double _f5_x_1a = f5_x_1a.read();
```

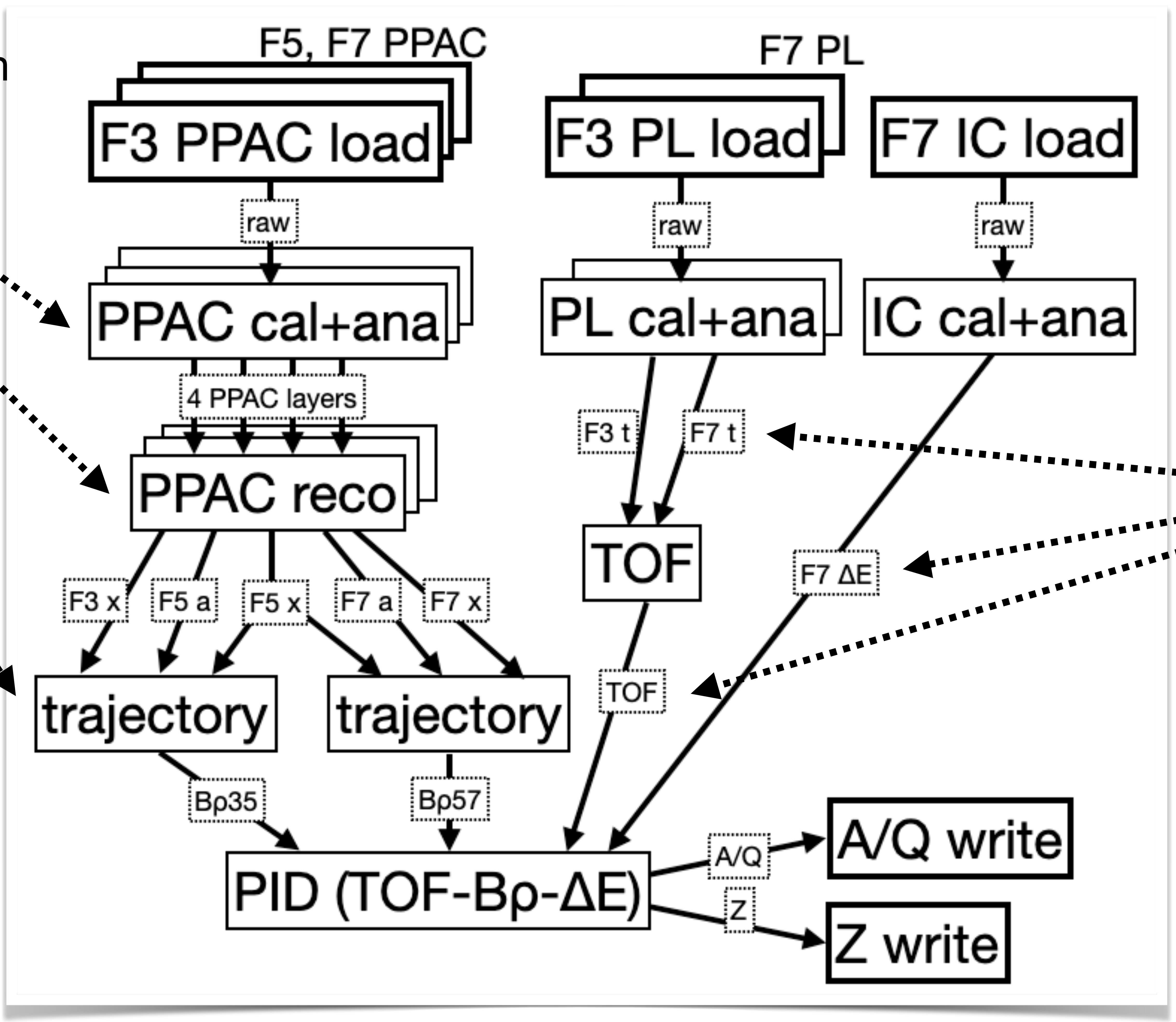


```
#pragma HLS STREAM variable=f7t     type=fifo depth=3
#pragma HLS STREAM variable=f7s     type=fifo depth=3

#pragma HLS STREAM variable=_aoq    type=fifo depth=2
#pragma HLS STREAM variable=_z      type=fifo depth=2

#pragma HLS DATAFLOW

    const ppac_params p_f3ppac_1a = p.f3ppac_1a;
    const ppac_params p_f3ppac_1b = p.f3ppac_1b;
    const ppac_params p_f3ppac_2a = p.f3ppac_2a;
    const ppac_params p_f3ppac_2b = p.f3ppac_2b;
    const ppac_params p_f5ppac_1a = p.f5ppac_1a;
    const ppac_params p_f5ppac_1b = p.f5ppac_1b;
    const ppac_params p_f5ppac_2a = p.f5ppac_2a;
    const ppac_params p_f5ppac_2b = p.f5ppac_2b;
    const ppac_params p_f7ppac_1a = p.f7ppac_1a;
    const ppac_params p_f7ppac_1b = p.f7ppac_1b;
    const ppac_params p_f7ppac_2a = p.f7ppac_2a;
    const ppac_params p_f7ppac_2b = p.f7ppac_2b;
    const pl_params   p_f3pl      = p.f3pl;
    const pl_params   p_f7pl      = p.f7pl;
    const ic_params   p_f7ic      = p.f7ic;
    const pid_params  p_pid       = p.pid;


    load_ppac(data_f3ppac, f3ppac, nchunk);
    load_ppac(data_f5ppac, f5ppac, nchunk);
    load_ppac(data_f7ppac, f7ppac, nchunk);
    load_pl(data_f3pl, f3pl, nchunk);
    load_pl(data_f7pl, f7pl, nchunk);
    load_ic(data_f7ic, f7ic, nchunk);

    loop_ppac_xf(f3ppac, f5ppac, f7ppac,
                 f3_x_1a, f3_f_1a, f3_x_1b, f3_f_1b, f3_x_2a, f3_f_2a, f3_x_2b, f3_f_2b,
                 f5_x_1a, f5_f_1a, f5_x_1b, f5_f_1b, f5_x_2a, f5_f_2a, f5_x_2b, f5_f_2b,
```

Realized RTL dataflow

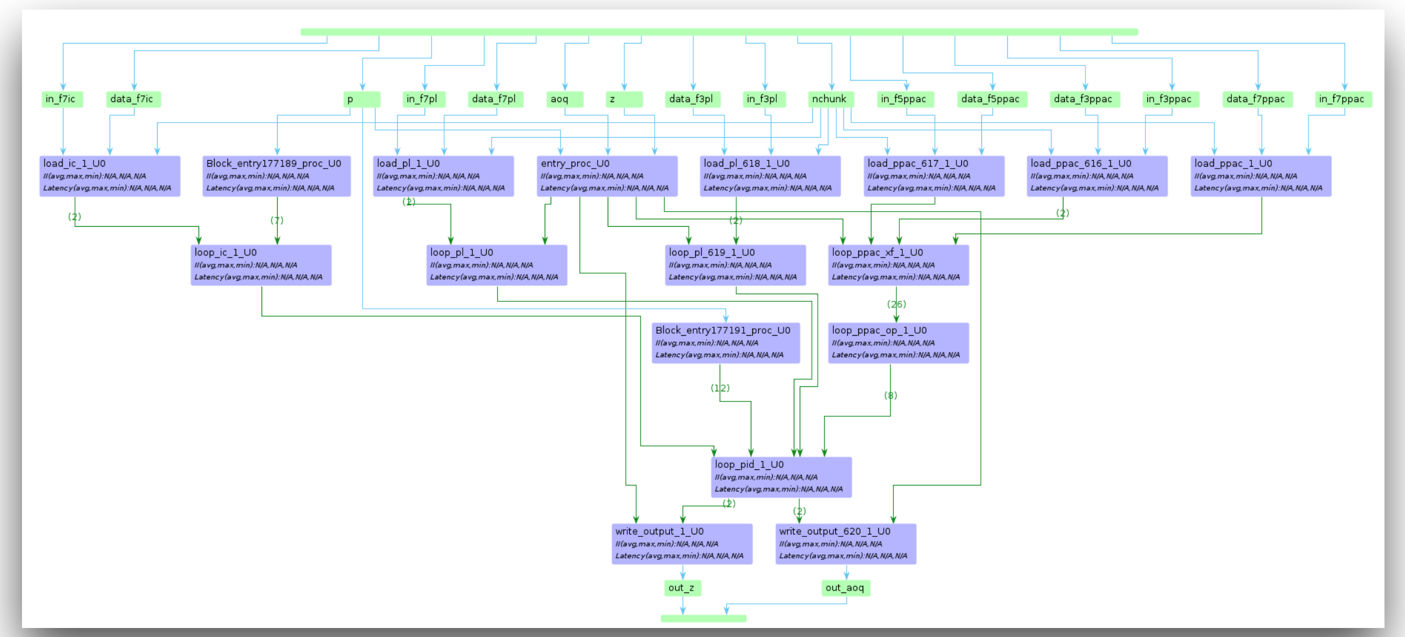
Independent task

- each is implemented as an <u>individual</u> FPGA design
- <u>every</u> task runs parallelly



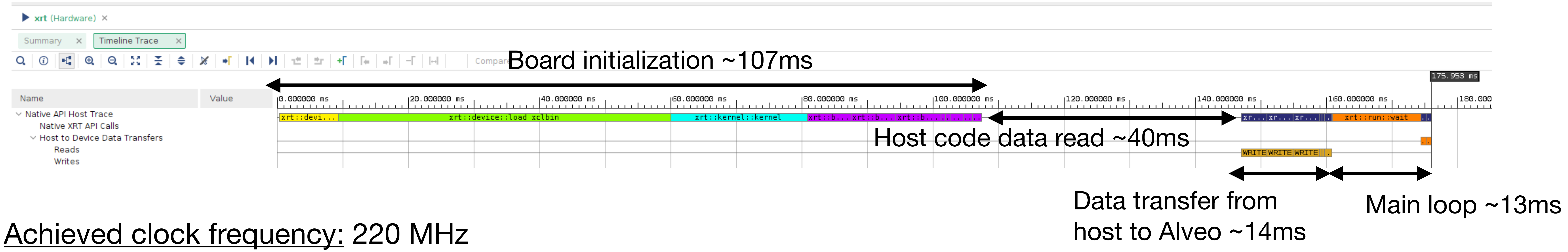
Pipeline register (FIFO)

Output from the official dataflow viewer



Successfully implemented

Performance



<u>Achieved clock frequency:</u> 220 MHz

<u>Achieved latency</u>

- ~1000 clks → ~4.3 us @ 220 MHz
- x10 slower compared to the latency 450 ns with CPU (core i9-10900X, single thread)

<u>Achieved pipeline processing</u>

- II = 5 clks → throughput 44 MHz @ 220 MHz
- cf. CPU throughput: 2.2kHz → **x20 throughput**

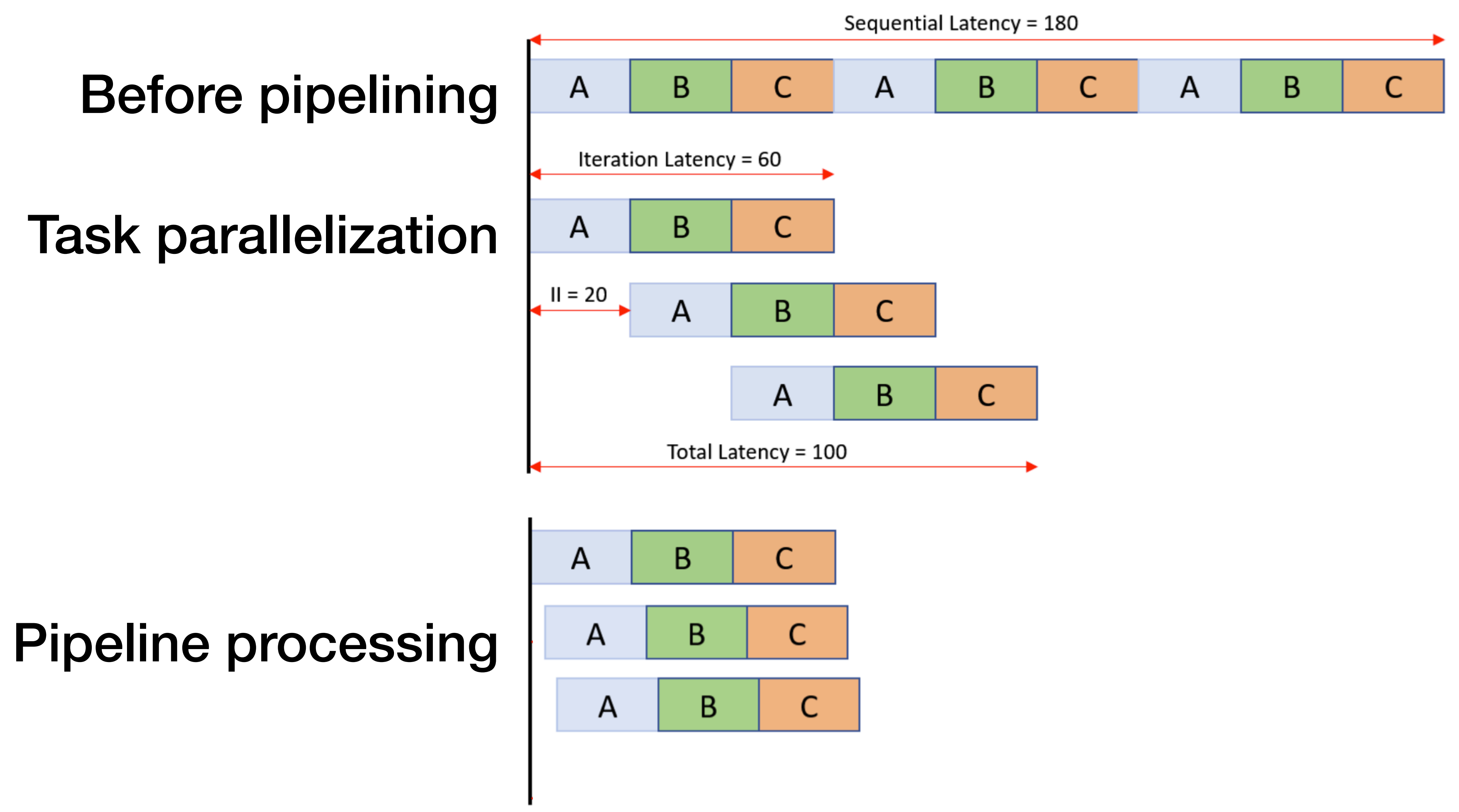
- **~40 MHz data acquisition is possible** (cf. secondary beam intensity: << 1e7 cps)
- **PID (physical) trigger is possible if ~5 us delay is acceptable**
- **Overheads (board init., data transfer etc.) should be removed, e.g. by data streaming, for practical use**

Implications

1. ~40 MHz data acquisition is possible
2. PID trigger is possible if ~5 us delay is acceptable

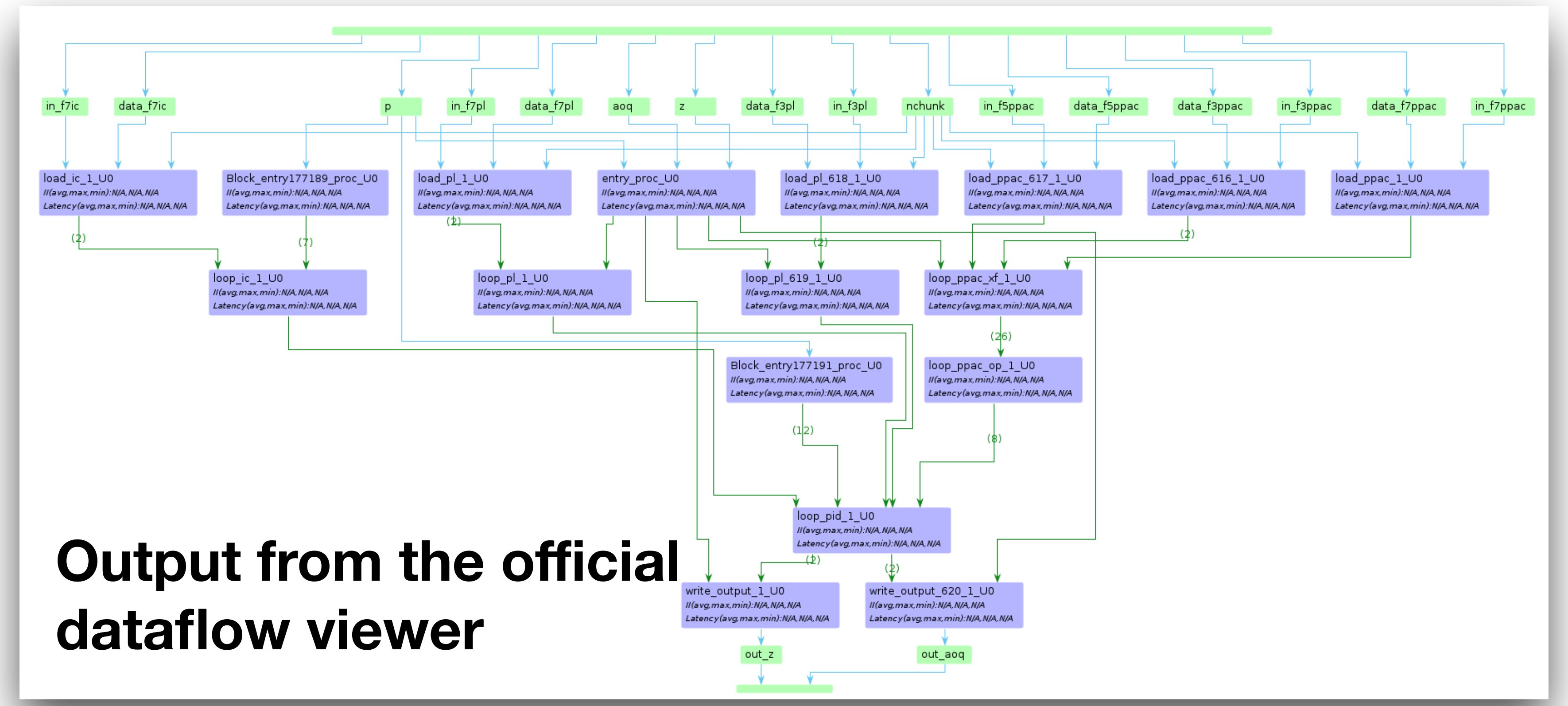
Huge speed-up (throughput) can be expected

- cf. CPU: enhancing clock frequency
- cf. GPU: data-parallelization



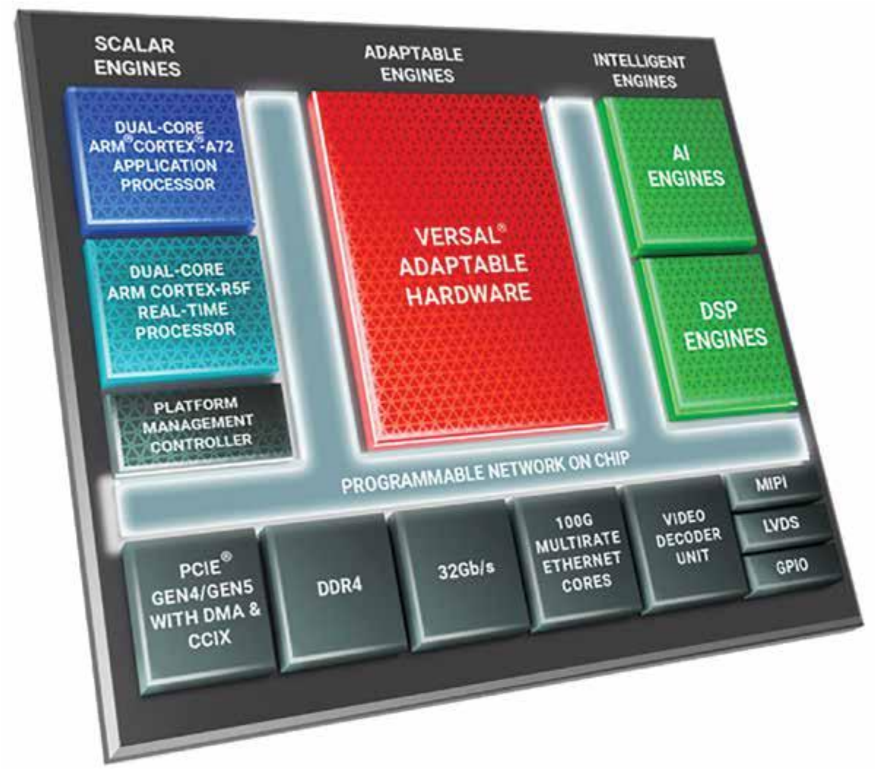
FPGA: task-parallelization
- Rather complicated tasks can be realized as dedicated hardware
- Independent, multiple kind of tasks can be completely parallelized
- Pipelined with II~a few leads to huge gain in throughput

→ advantageous when the task consists of multiple, rather complex subtasks, which need to be executed in an organized manner


Output from the official dataflow viewer

Streaming data can be analyzed with very little overheads
- Direct external connectivity via QSFP28
- Pipeline buffering without register/memory read/write

Current status / Future plans


SCALAR ENGINES
ADAPTABLE ENGINES
INTELLIGENT ENGINES
DUAL-CORE ARM CORTEX-A72 APPLICATION PROCESSOR
DUAL-CORE ARM CORTEX-R5F REAL-TIME PROCESSOR
PLATFORM MANAGEMENT CONTROLLER
VERSAL® ADAPTABLE HARDWARE
AI ENGINES
DSP ENGINES
PROGRAMMABLE NETWORK ON CHIP
PCIE® GEN4/GEN5 WITH DMA & CCIX
DDR4
32Gb/s
100G MULTIRATE ETHERNET CORES
VIDEO DECODER UNIT
MIPI
LVDS
GPIO

Communication using direct I/O

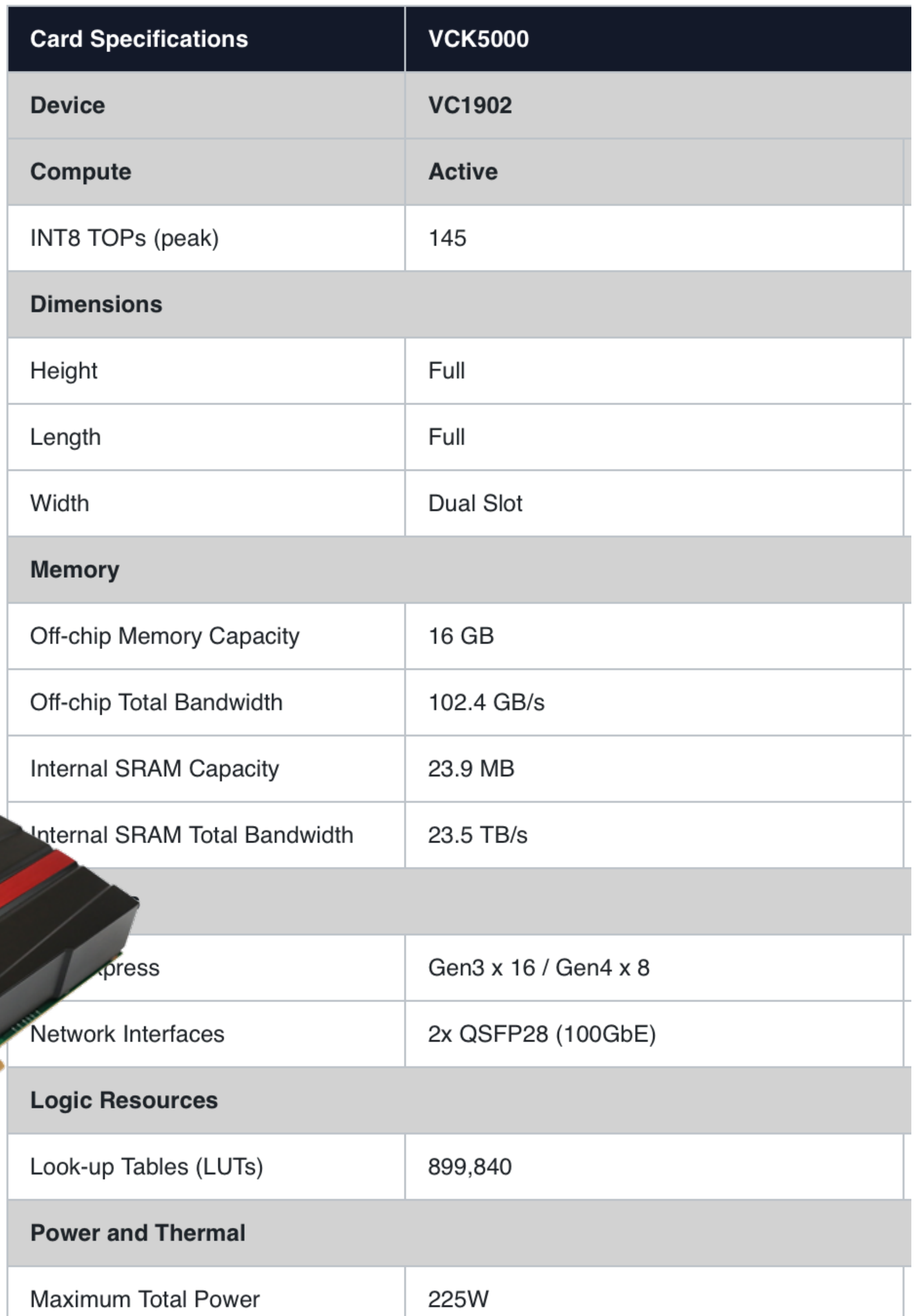
- QSFP28 port + Xilinx Aurora 64B/66B kernel
    - 100 Gbps achieved (loopback)
- Will try communication using two boards

Drift chamber data analysis

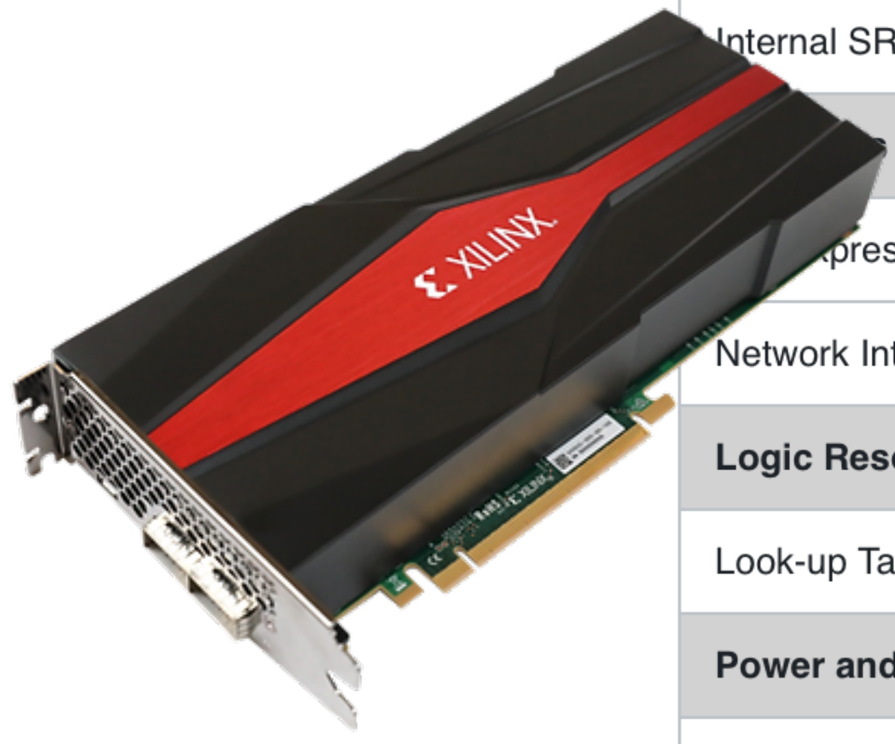
- Variable-length data / loops, nested loops
    - May not be suitable for hardware implementation
- Exploring smarter ways of implementation
    - Currently CPU-only processing is still faster
    - Machine learning / AI (using Versal?)

Versal VCK 5000

- FPGA + AI Core (matrix computation engine; ~ GPU) + QSFP28 x 2
- “GPU that can accept 100 Gbps direct data stream (?)”



Card Specifications	VCK5000
Device	VC1902
Compute	Active
INT8 TOPs (peak)	145
Dimensions	
Height	Full
Length	Full
Width	Dual Slot
Memory	
Off-chip Memory Capacity	16 GB
Off-chip Total Bandwidth	102.4 GB/s
Internal SRAM Capacity	23.9 MB
nternal SRAM Total Bandwidth	23.5 TB/s
press	Gen3 x 16 / Gen4 x 8
Network Interfaces	2x QSFP28 (100GbE)
Logic Resources	
Look-up Tables (LUTs)	899,840
Power and Thermal	
Maximum Total Power	225W


VCK5000

Summary

Exploring the capability of Alveo in the RIBF DAQ data analysis

- Achieved x20 throughput compared to CPU for PID
- Huge speed-up (throughput) can be expected if task-parallelization is possible
- Direct external connectivity may allow streaming data to be analyzed with very little overheads

Will continue to explore further possibilities

- External communication
- For what kind of tasks does Alveo/Versal hardware acceleration is suitable?
- Versal AI core