Towards 1000x Speedup for HEP Workloads with Heterogeneous Programmable Datacenters

Anton Burtsev, Alex Veidenbaum
aburtsev@uci.edu, alexv@ics.uci.edu
University of California, Irvine
March, 2018

A Roadmap for HEP Software and Computing R&D for the 2020s

HEP Software Foundation[1]

Abstract: Particle physics has an ambitious and broad experimental programme for the coming decades. This programme requires large investments in detector hardware, either to build new facilities and experiments, or to upgrade existing ones. Similarly, it requires commensurate investment in the R&D of software to acquire, manage, process, and analyse the shear amounts of data to be recorded. In planning for the HL-LHC in particular, it is critical that all of the collaborating stakeholders agree on the software goals and priorities, and that the efforts complement each other. In this spirit, this white paper describes the R&D activities required to prepare for this software upgrade.

Compute Ex #1: Exploratory Data Analysis

Fast Access to Columnar, Hierarchically Nested Data via Code Transformation

Jim Pivarski, Peter Elmer
Physics Department
Princeton University
Princeton, NJ, 08544
pivarski@princeton.edu, peter.elmer@cern.ch

Brian Bockelman, Zhe Zhang
Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588
bbockelm@cse.unl.edu, zhan0915@huskers.unl.edu

***Abstract*—Big Data query systems represent data in a columnar format for fast, selective access, and in some cases (e.g. Apache Drill), perform calculations directly on the columnar data without row materialization, avoiding runtime costs.**

However, many analysis procedures cannot be easily or efficiently expressed as SQL. In High Energy Physics, the majority of HEP data cross-linking is limited to small, disconnected graphs called "events" no larger than hundreds of kilobytes.

Furthermore, most of the well-known database systems use SQL or an SQL variant as the query language, but even the simplest HEP analysis functions are awkward and possibly

Compute Ex #1: Exploratory Data Analysis

- Dataset:
  - 5.4 million events (simulated Drell-Yan collisions)
  - Typical analysis will involve 10 such datasets

  - Float: 5.4*4 = 21.6MB x 10 = 216MB
  - Double: 432MB

max p_T in C++

```
float maximum = 0.0;
for (i=0; i < muons.size(); i++)
  if (muons[i]->pt > maximum)
    maximum = muons[i]->pt;
fill_histogram(maximum);
```

eta of best by p_T in C++

```
float maximum = 0.0;
Muon* best = nullptr;
for (i=0; i < muons.size(); i++)
  if (muons[i]->pt > maximum) {
    maximum = muons[i]->pt;
    best = muon; }
if (best != nullptr)
  fill_histogram(best->eta);
```

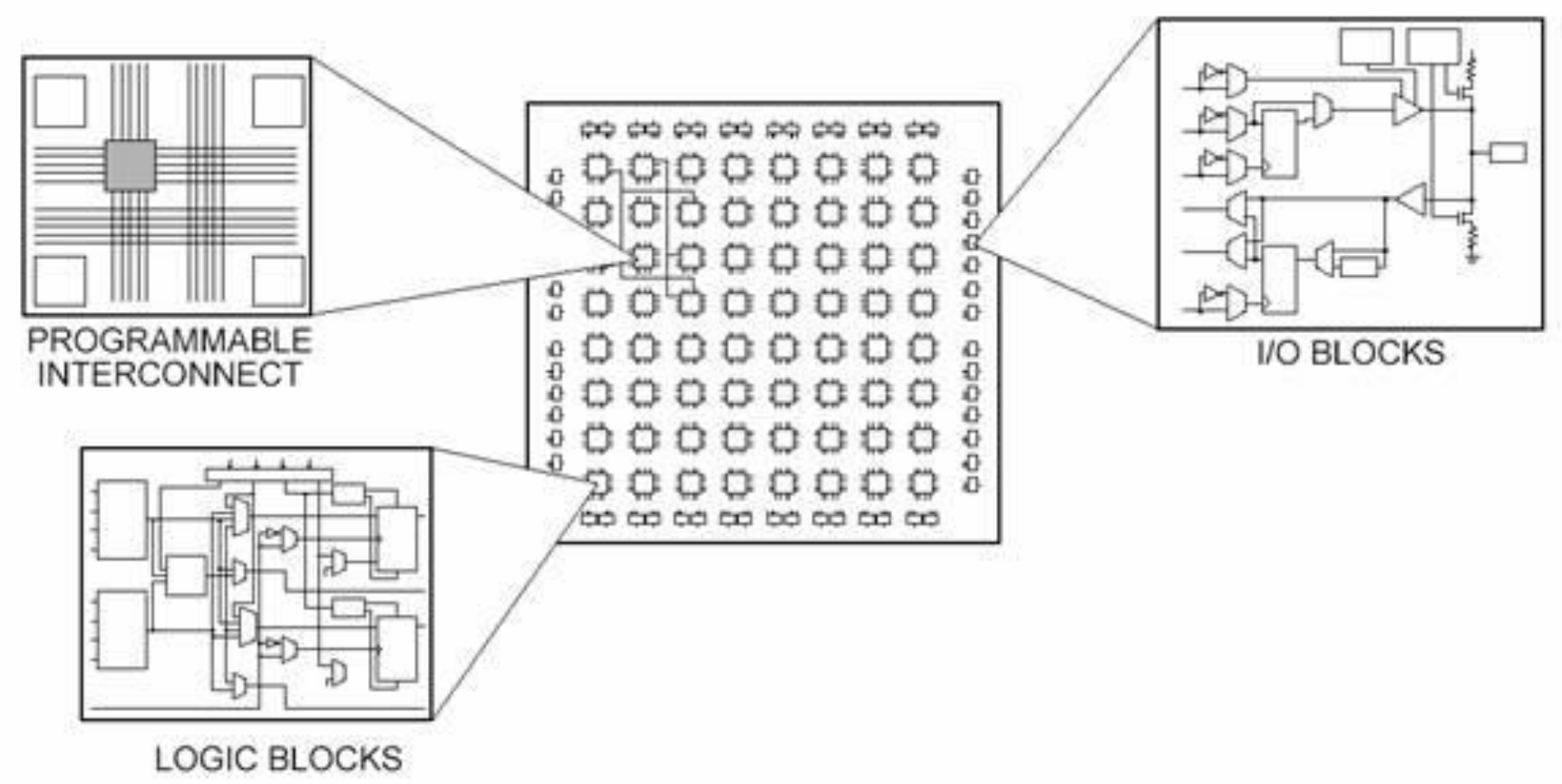
mass of pairs in C++

```
int n = muons.size();
for (i=0; i < n; i++)
  for (j=i+1; j < n; j++) {
    Muon* m1 = muons[i];
    Muon* m2 = muons[j];
    double mass = sqrt(
      2*m1->pt*m2->pt*(
-     cosh(m1->eta - m2->eta) -
)     cos(m1->phi - m2->phi)));
    fill_histogram(mass); }
```

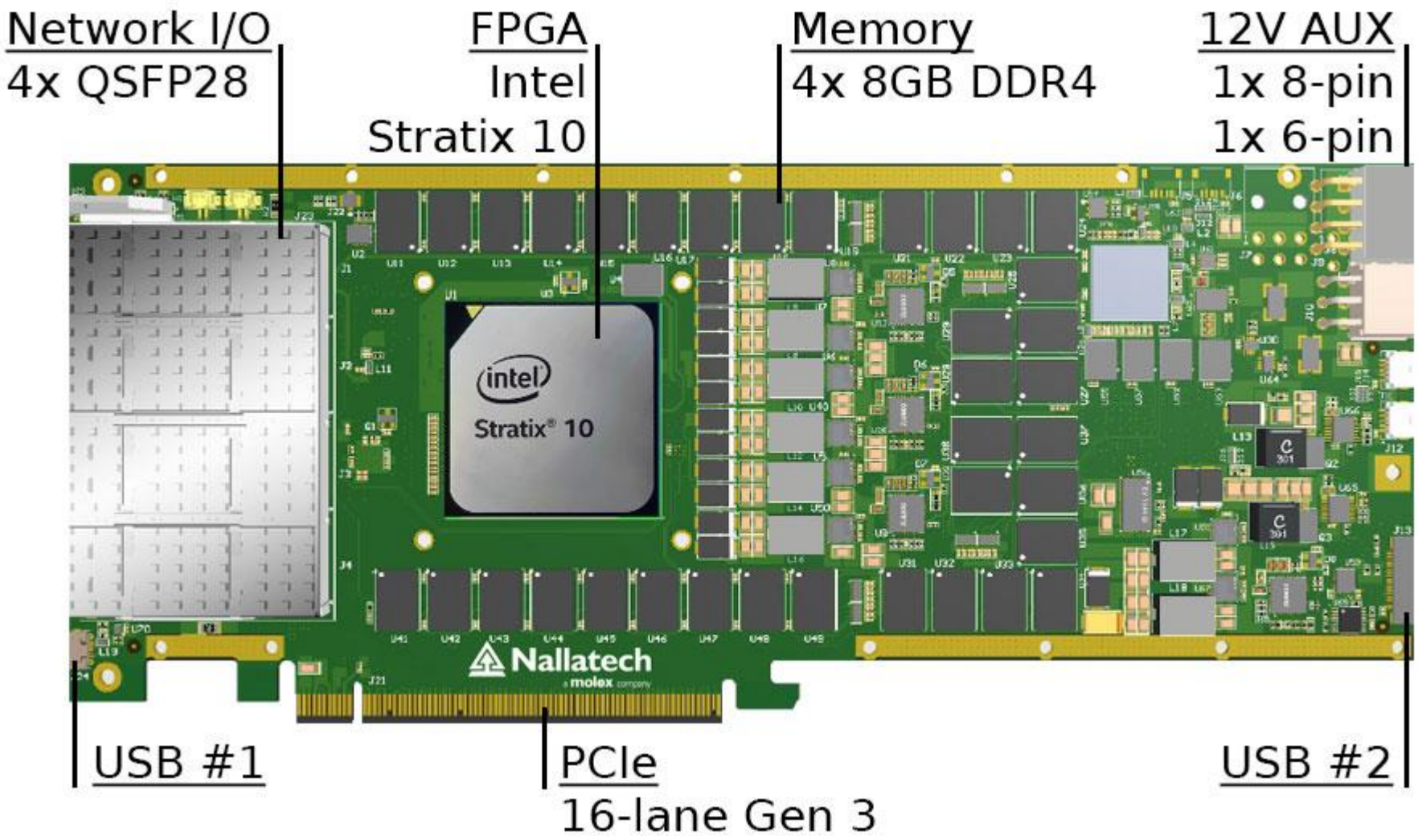
p_T sum of pairs in C++

```
int n = muons.size();
for (i=0; i < n; i++)
  for (j=i+1; j < n; j++) {
    Muon* m1 = muons[i];
    Muon* m2 = muons[j];
    double s = m1->pt + m2->pt;
    fill_histogram(s); }
```

FPGA Filed-programmable gate array

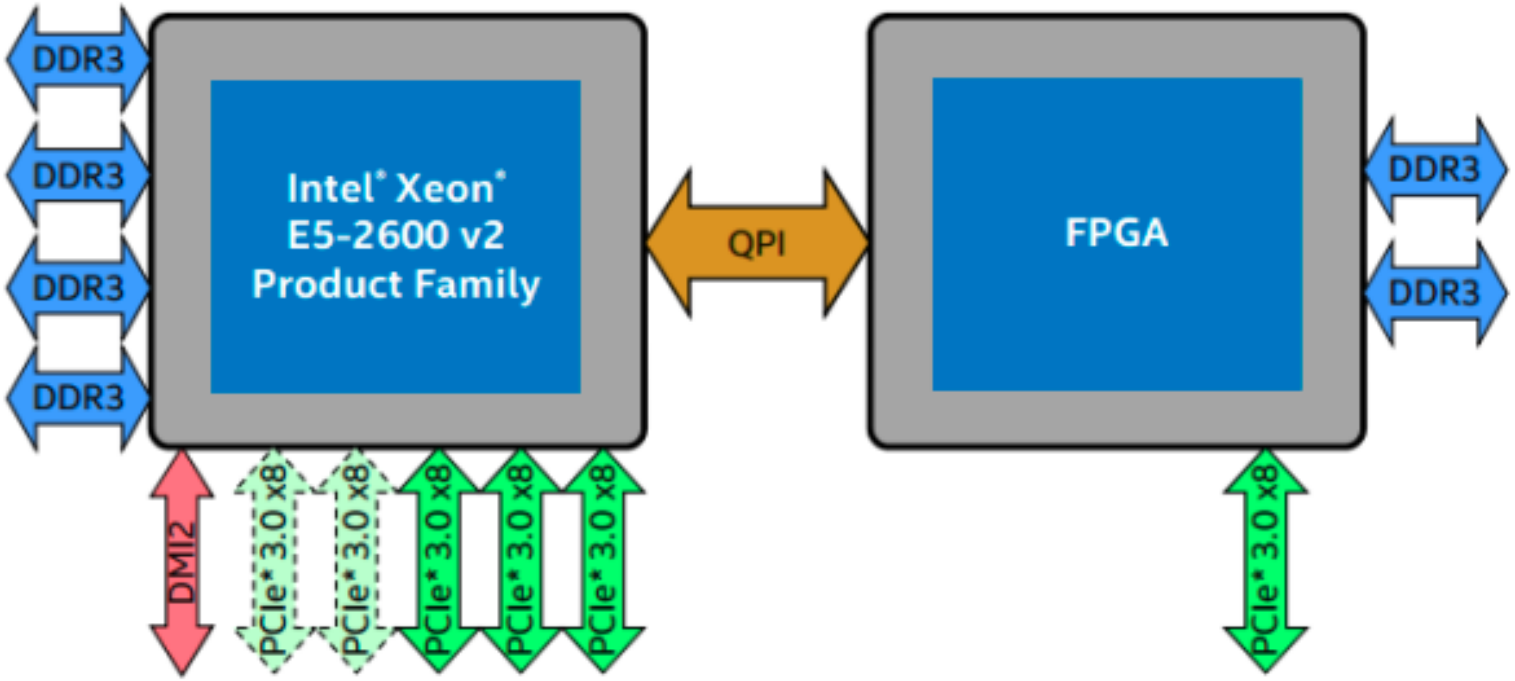


Intel Stratix 10 FPGA



https://www.altera.com/products/fpga/stratix-series/stratix-10/overview.html

Intel HARP: Cache-coherent FPGA



Processor	Intel® Xeon® E5-26xx v2 Processor
FPGA Module	Altera Stratix V
QPI Speed	6.4 GT/s full width (target 8.0 GT/s at full width)
Memory to FPGA Module	2 channels of DDR3 (up to 64 GB)
Expansion connector to FPGA Module	PCIe 3.0 x8 lanes - maybe used for direct I/O e.g. Ethernet
Features	Configuration Agent, Caching Agent,, (optional) Memory Controller
Software	Accelerator Abstraction Layer (AAL) runtime, drivers, sample applications

max p_T in C++

```
float maximum = 0.0;
for (i=0; i < muons.size(); i++)
  if (muons[i]->pt > maximum)
    maximum = muons[i]->pt;
fill_histogram(maximum);
```

eta of best by p_T in C++

```
float maximum = 0.0;
Muon* best = nullptr;
for (i=0; i < muons.size(); i++)
  if (muons[i]->pt > maximum) {
    maximum = muons[i]->pt;
    best = muon; }
if (best != nullptr)
  fill_histogram(best->eta);
```

mass of pairs in C++

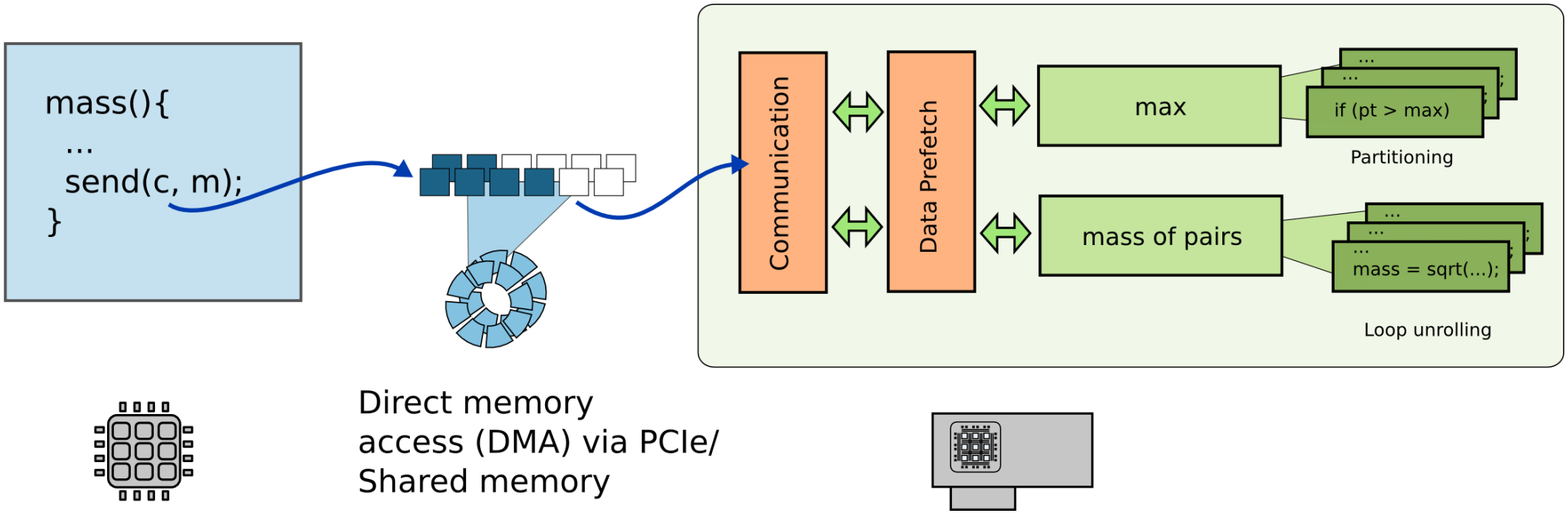
```
int n = muons.size();
for (i=0; i < n; i++)
  for (j=i+1; j < n; j++) {
    Muon* m1 = muons[i];
    Muon* m2 = muons[j];
    double mass = sqrt(
      2*m1->pt*m2->pt*(
-     cosh(m1->eta - m2->eta) -
)     cos(m1->phi - m2->phi)));
    fill_histogram(mass); }
```

p_T sum of pairs in C++

```
int n = muons.size();
for (i=0; i < n; i++)
  for (j=i+1; j < n; j++) {
    Muon* m1 = muons[i];
    Muon* m2 = muons[j];
    double s = m1->pt + m2->pt;
    fill_histogram(s); }
```

FPGA acceleration

- Parallel pipelines
  - Partition the input
  - Unroll loops
- Reconfigurable with partial reconfiguration



FPGA vs GPU

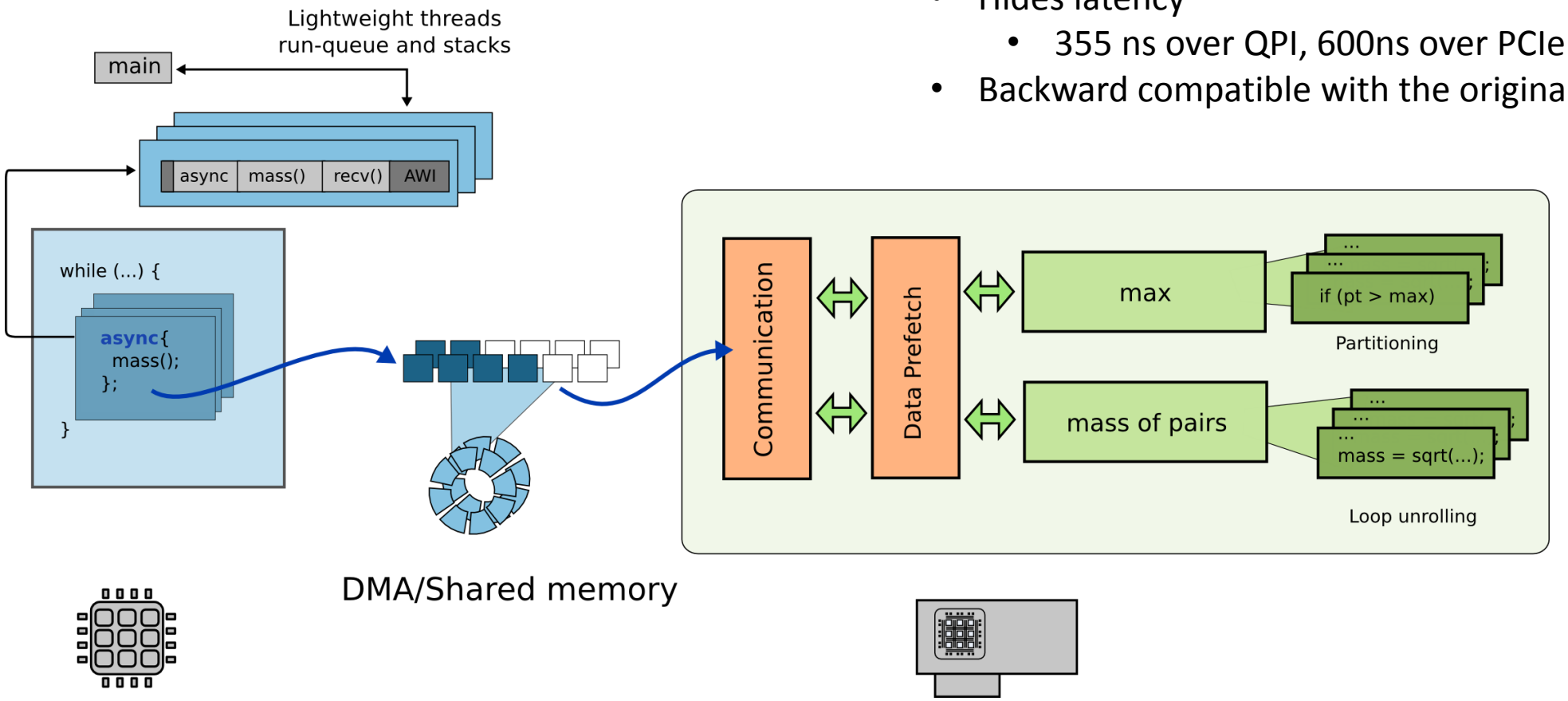
- NVidia Tesla V100 GPU
  - 15 TFLOP single point
  - 60GFLOP per watt

- Intel Stratix 10 FPGA
  - 10 TFLOP single point
  - 80GFLOP per watt

More control

- Low-latency communication via DMA or shared memory with the main program
  - Simple ring-buffer optimized for the number of cache-coherence or PCIe transactions
- Data prefetching from the host (CPU) and device (FPGA) memories and even from NVMe
- Direct communication over the network and with NVMe

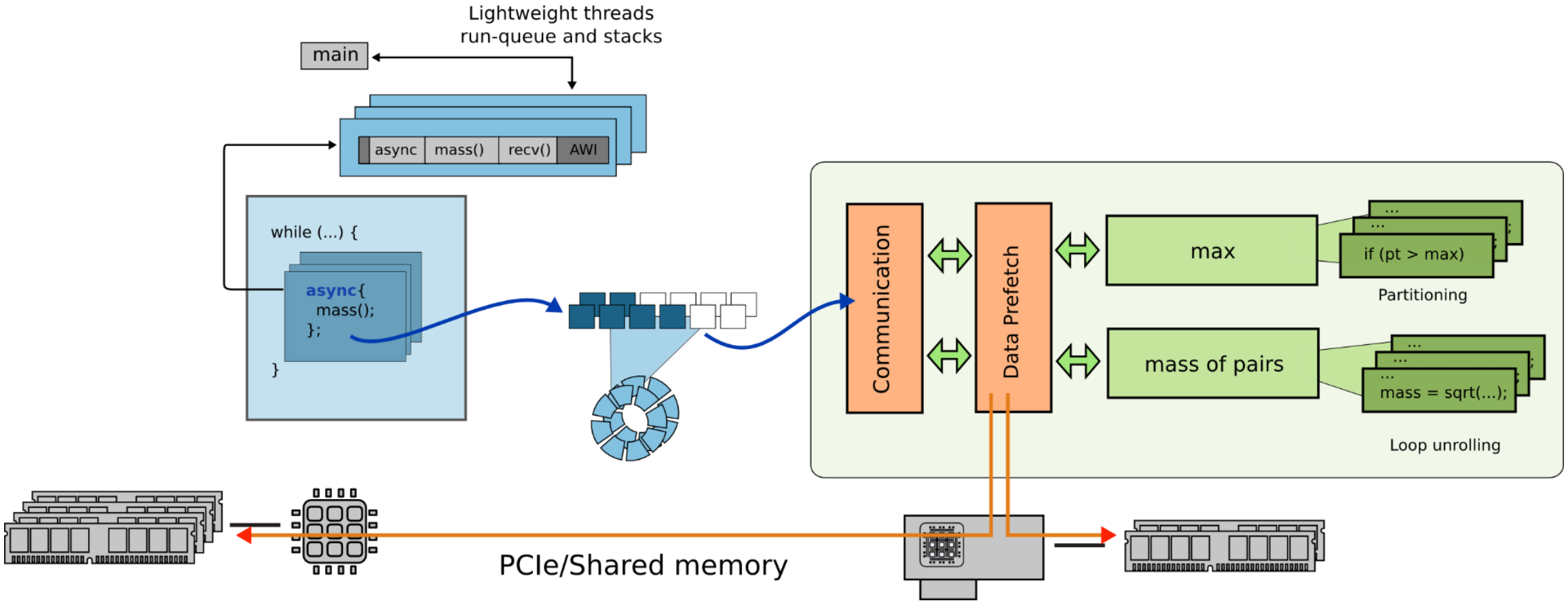
Integration with existing programs: asynchronous runtime



- Hides latency
  - 355 ns over QPI, 600ns over PCIe
- Backward compatible with the original code

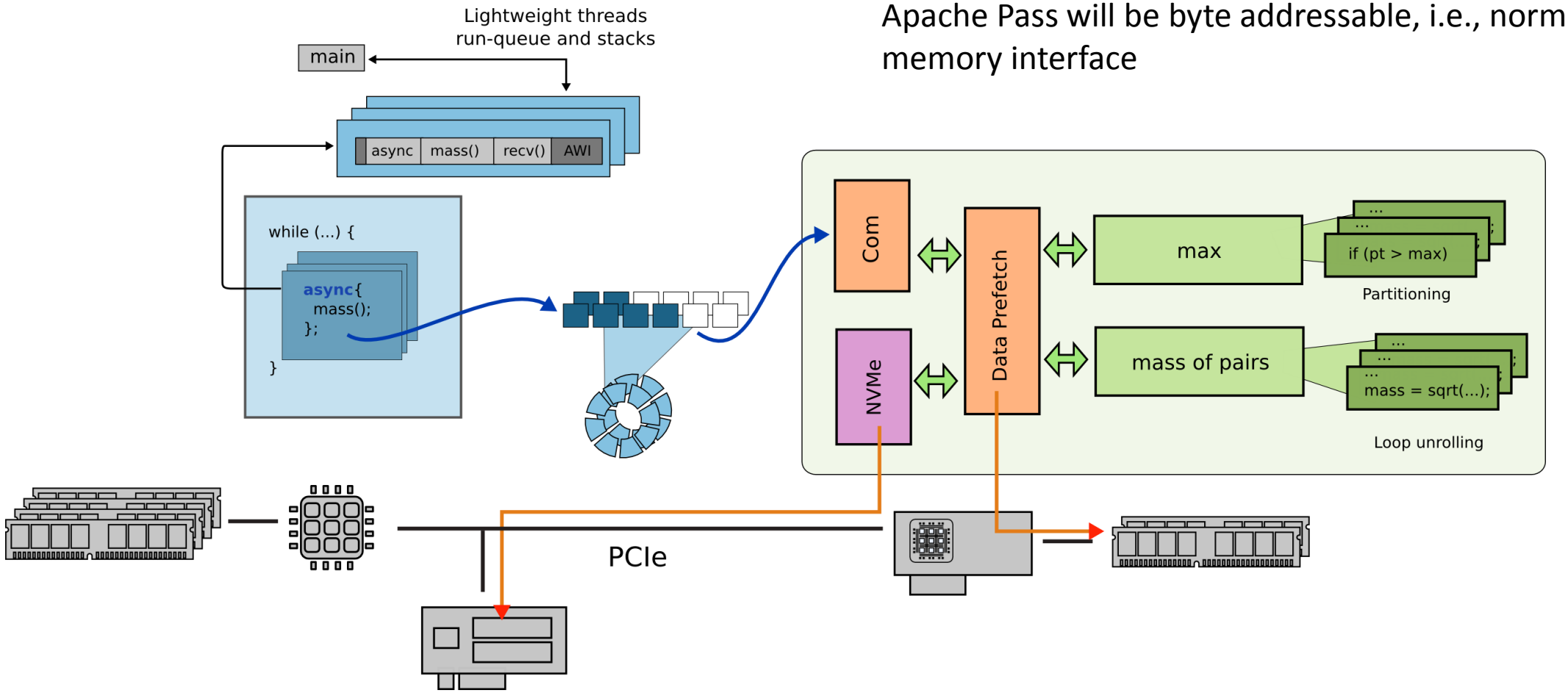
Data prefetching

- FPGA has
  - 6MB of fast block RAM
  - 4GB of DRAM
- Program a custom prefetch logic that is aware of the data layout

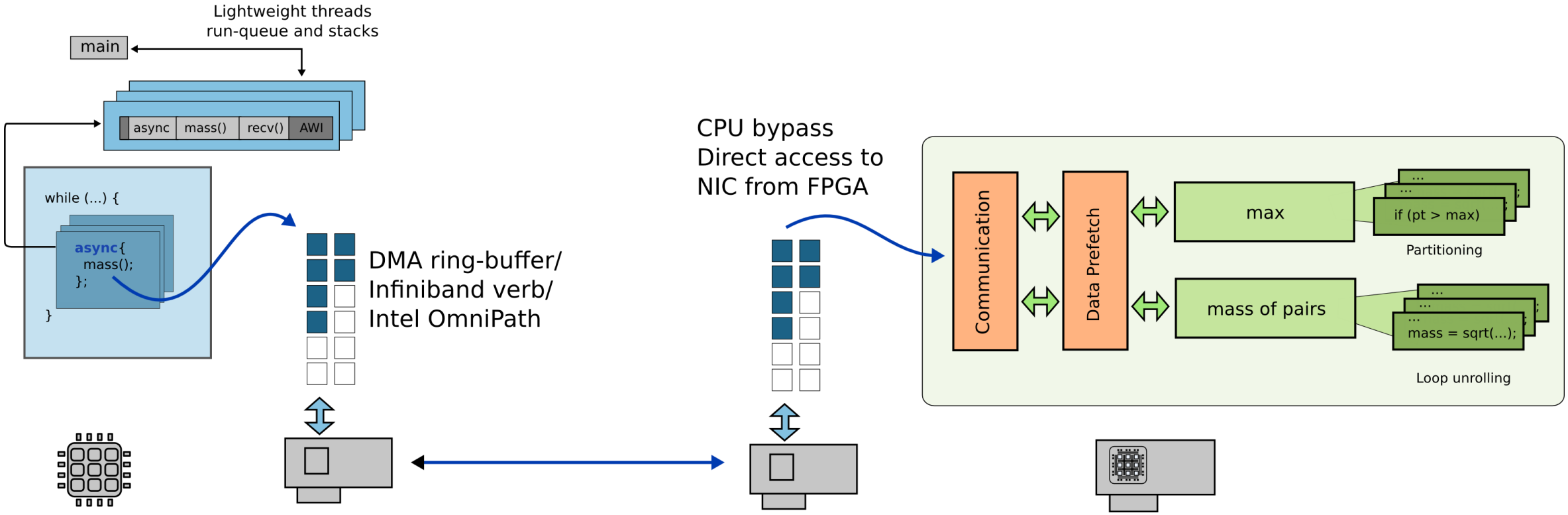


Direct access to NVMe

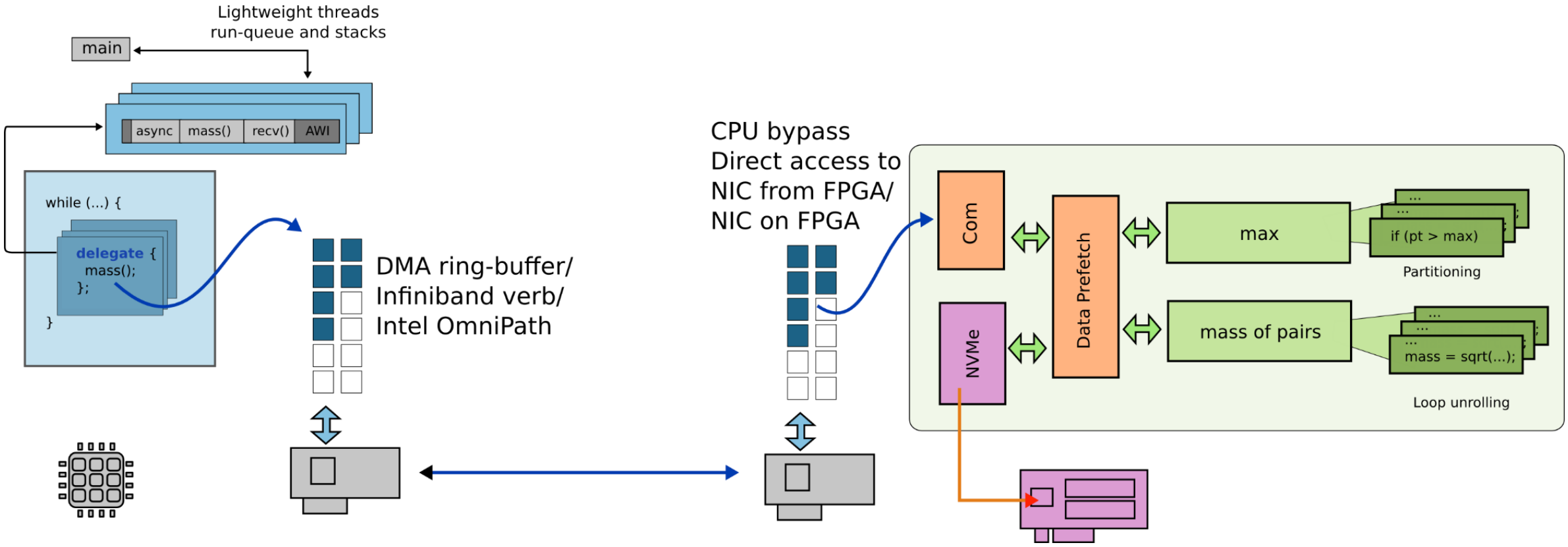
- Direct access to storage devices
  - NVMe is a simple ring-based protocol
  - Easy to program in FPGA
- Emerging non-volatile DIMMs, e.g., Intel 3D Xpoint Apache Pass will be byte addressable, i.e., normal memory interface



Remote access over the network

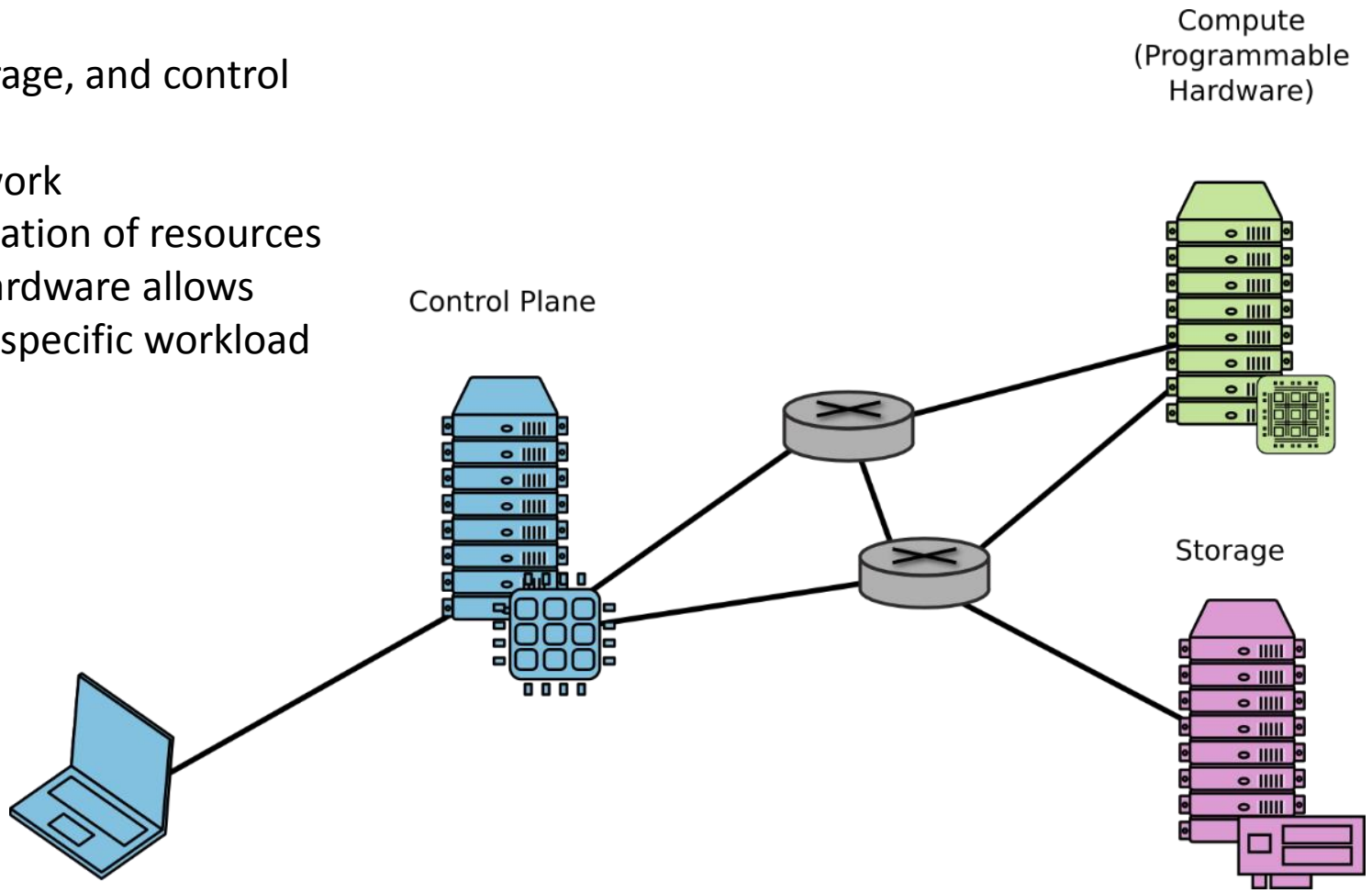


Collocating compute and storage



Disaggregated programmable datacenter

- Pools of compute, storage, and control plane servers
  - Low-latency network
- Flexible, dynamic allocation of resources
  - Programmable hardware allows optimization of a specific workload



Example applications

Parallelized Kalman-Filter-Based Reconstruction of Particle Tracks on Many-Core Processors and GPUs

Giuseppe **Cerati**[4,a], Peter **Elmer**[2,b], Slava **Krutelyov**[1,c], Steven **Lantz**[3,d], Matthieu **Lefebvre**[2,e], Mario **Masciovecchio**[1,f], Kevin **McDermott**[3,g], Daniel **Riley**[3,h], Matevž **Tadel**[1,i], Peter **Wittich**[3,j], Frank **Würthwein**[1,k], and Avi **Yagil**[1,l]

[1] *UC San Diego, La Jolla, CA, United States of America 92093*
[2] *Princeton University, Princeton, NJ, United States of America 08544*
[3] *Cornell University, Ithaca, NY, United States of America 14853*
[4] *Fermilab, Batavia, IL, United States of America 60510-5011*

Abstract. For over a decade now, physical and energy constraints have limited clock speed improvements in commodity microprocessors. Instead, chipmakers have been pushed into producing lower-power, multi-core processors such as Graphical Processing Units (GPU), ARM CPUs, and Intel MICs. Broad-based efforts from manufacturers and developers have been devoted to making these processors user-friendly enough to

Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Nuclear Instruments and Methods in Physics Research A 506 (2003) 250–303
NUCLEAR INSTRUMENTS & METHODS IN PHYSICS RESEARCH Section A
www.elsevier.com/locate/nima

GEANT4—a simulation toolkit

S. Agostinelli[ae], J. Allison[as,*], K. Amako[e], J. Apostolakis[a], H. Araujo[aj], P. Arce[l,m,x,a], M. Asai[g,ai], D. Axen[i,t], S. Banerjee[bi,l], G. Barrand[an], F. Behner[l], L. Bellagamba[c], J. Boudreau[bd], L. Broglia[ar], A. Brunengo[c], H. Burkhardt[a], S. Chauvie[bj,bl], J. Chuma[h], R. Chytracek[a], G. Cooperman[az], G. Cosmo[a], P. Degtyarenko[d], A. Dell'Acqua[a,i], G. Depaola[y], D. Dietrich[af], R. Enami[ab], A. Feliciello[bj], C. Ferguson[bh], H. Fesefeldt[l,o], G. Folger[a], F. Foppiano[ac], A. Forti[as], S. Garelli[ac], S. Giani[a], R. Giannitrapani[bo], D. Gibin[m,bc], J.J. Gómez

Cling – The New Interactive Interpreter for ROOT 6

V Vasilev[1], Ph Canal[2], A Naumann[1], P Russo[2]
1 CERN, PH-SFT, Geneva, Switzerland
[2] FermiLab, P.O. Box 500 Batavia, IL, US

E-mail: vasil.georgiev.vasilev@cern.ch

Abstract. Cling is an interactive C++ interpreter, built on top of Clang and LLVM compiler infrastructure. Like its predecessor Cint, Cling realizes the *read-print-evaluate-loop* concept, in order to leverage rapid application development. Implemented as a small extension to LLVM and Clang, the interpreter reuses their strengths such as the praised concise and expressive compiler diagnostics. We show how to match the interpreter concept to the compiler library and generalize common set of requirements for building up an interactive interpreter. We reason the design and implementation decisions as solution to the challenge of implementing interpreter behaviour as an extension of the compiler library. We present the new features, e.g. how C++11 will come to Cling and how Cint-specific extensions are being adopted. We clarify the state of integration in the ROOT framework and the induced change set. We explain how ROOT dictionaries are simplified due to the new interpreter.

Discussion

- We need help with understanding
  - Sizes of HEP datasets
  - Shape of the computation, e.g., similar to mass of pairs, but for Kalman Filter and Monte Carlo