GPU Accelerated Computational Instruments with NVIDIA Holoscan

Adam Thompson | Principal Technical Product Manager
adamt@nvidia.com

Background and Motivations

Scientific Computing is Evolving

Experiments

Simulation

Viz

Edge

HPC+AI

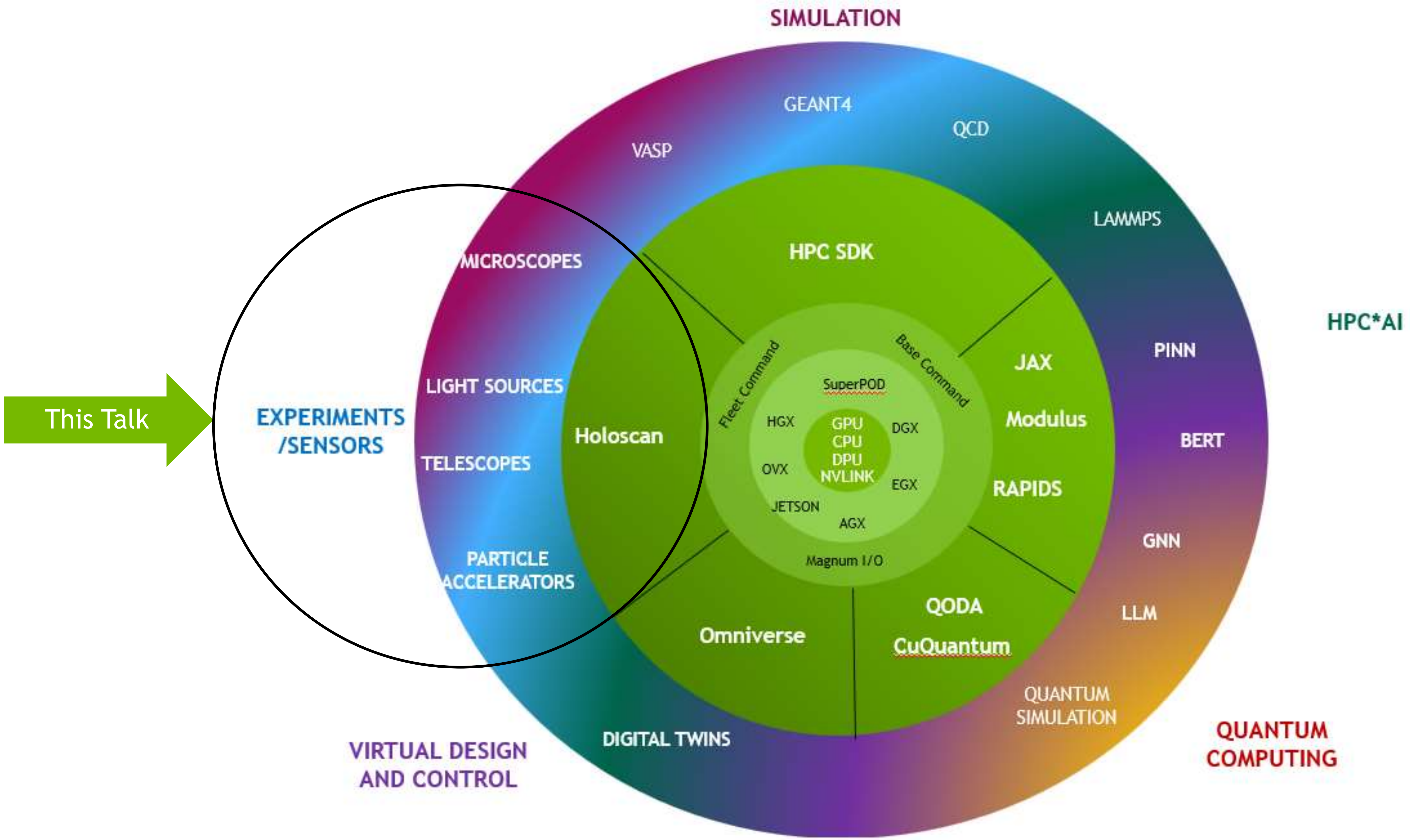
Simulation

Digital Twin

Quantum Computing

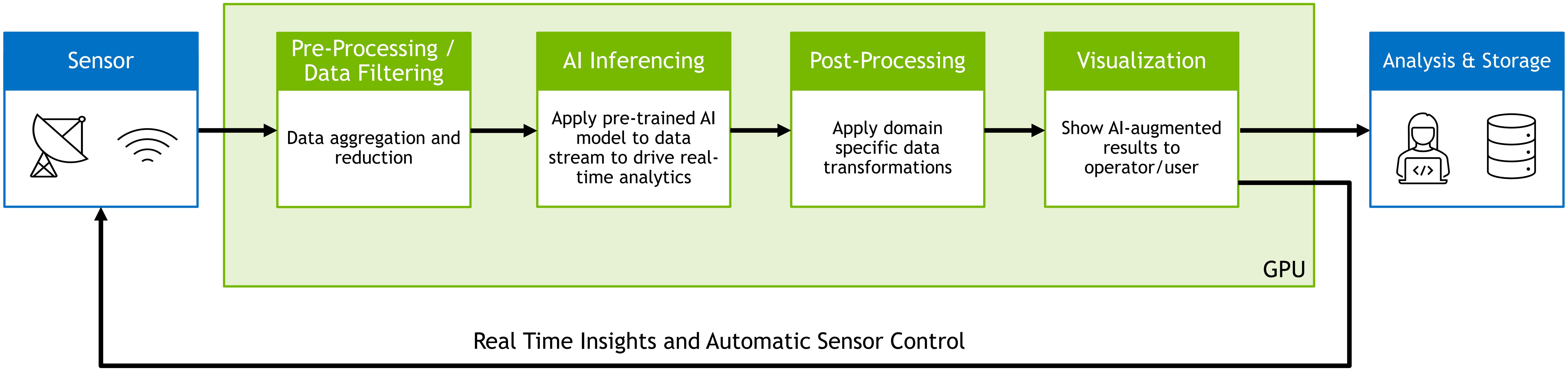
Feature	Pre-Exascale	Post Exascale
Usage	Batch	Interactive & distributed
Workload	Single simulation / ensemble	Simulation/ensembles, AI training and inference, Quantum, Edge, Twin
System Configuration	Homogeneous	Modular Composed Heterogeneous Workflows
Experiments	Offline data analysis for experiments	Part of Real-time Instrument, Steering, and Offline
Digital Twins	Reduced models / in-situ visualization	Interactive combination of simulation and observational data
Quantum Computing	Nascent	National Priority
Programming Models	Fortran, C++, MPI, OpenMP, OpenACC	Standard parallelism support in Fortran, C++, MPI, OpenMP, OpenACC, Python, Julia, Pytorch, Tensorflow, DSLs

Composite Workflows for Advancing Science



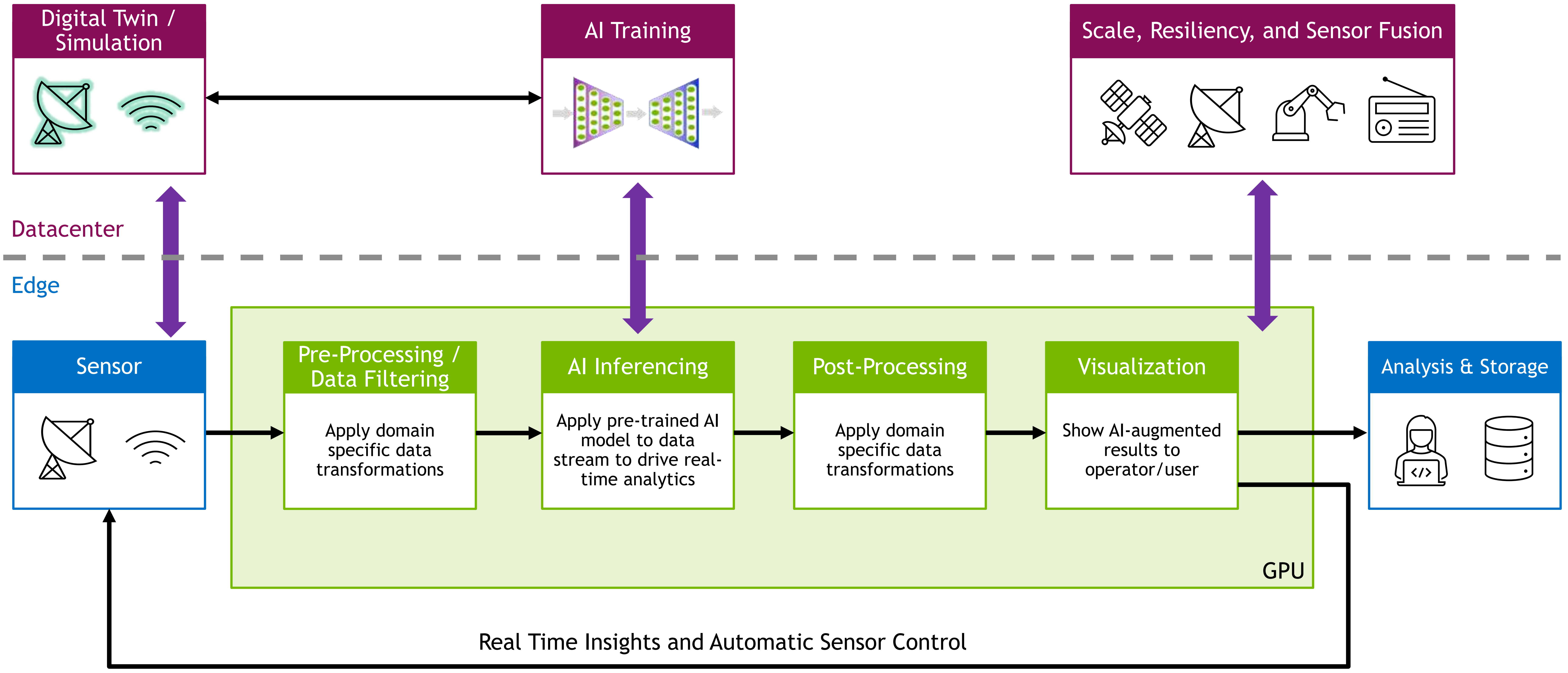
Anatomy of a Sensor Processing Pipeline at the Edge

Domain Agnostic, Software Defined, AI-Enabled, and Scalable



Edge and Datacenter Collaboration

A Vision of Integrated Workflows



The 5 Pillars of Sensor Processing

Holoscan Platform

Sensor to GPU Data Movement

High Bandwidth
Low Latency
GPUDirect
No Retransmits
Sensor Agnostic

Real Time GPU Compute

C++ and Python
Jax, CuPy, Numba
CUDA-X, MatX
DLPack
Bring Your Own Code

Real Time AI Inferencing

Bring Your Own Model
Multi-Model Inference
PyTorch, Tensorflow
TensorRT
TAO

Real Time Visualization

Data Type Agnostic
Data Format Agnostic
Interactive

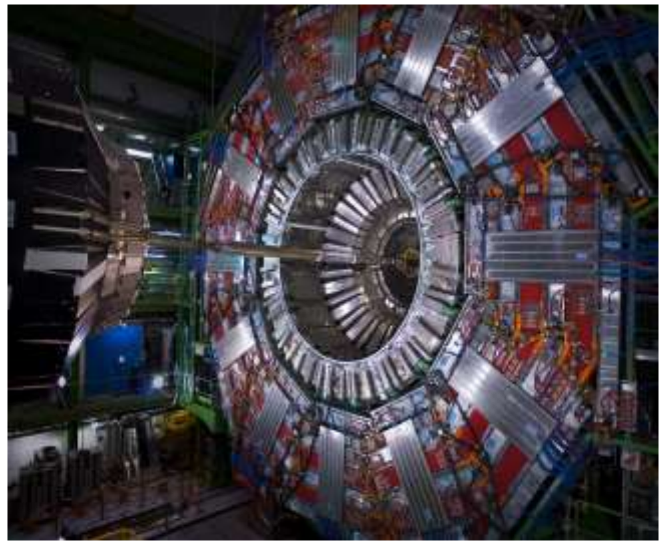
Collaborate with Cloud/Datacenter

Scale Out Compute
Finetune AI Model
Refine Digital Twin
Multi-Sensor Fusion
Resiliency

Holoscan – NVIDIA's AI-Enabled Streaming Sensor Platform

NVIDIA Holoscan Platform

Enabling Real Time, AI-Enabled Streaming Analytics at Any Scale



AGX Orin
Embedded

IGX Orin
Enterprise Edge

NVIDIA AI

MGX / DGX
HPC

Grace Hopper
Simulation

Sensor and Domain Agnostic

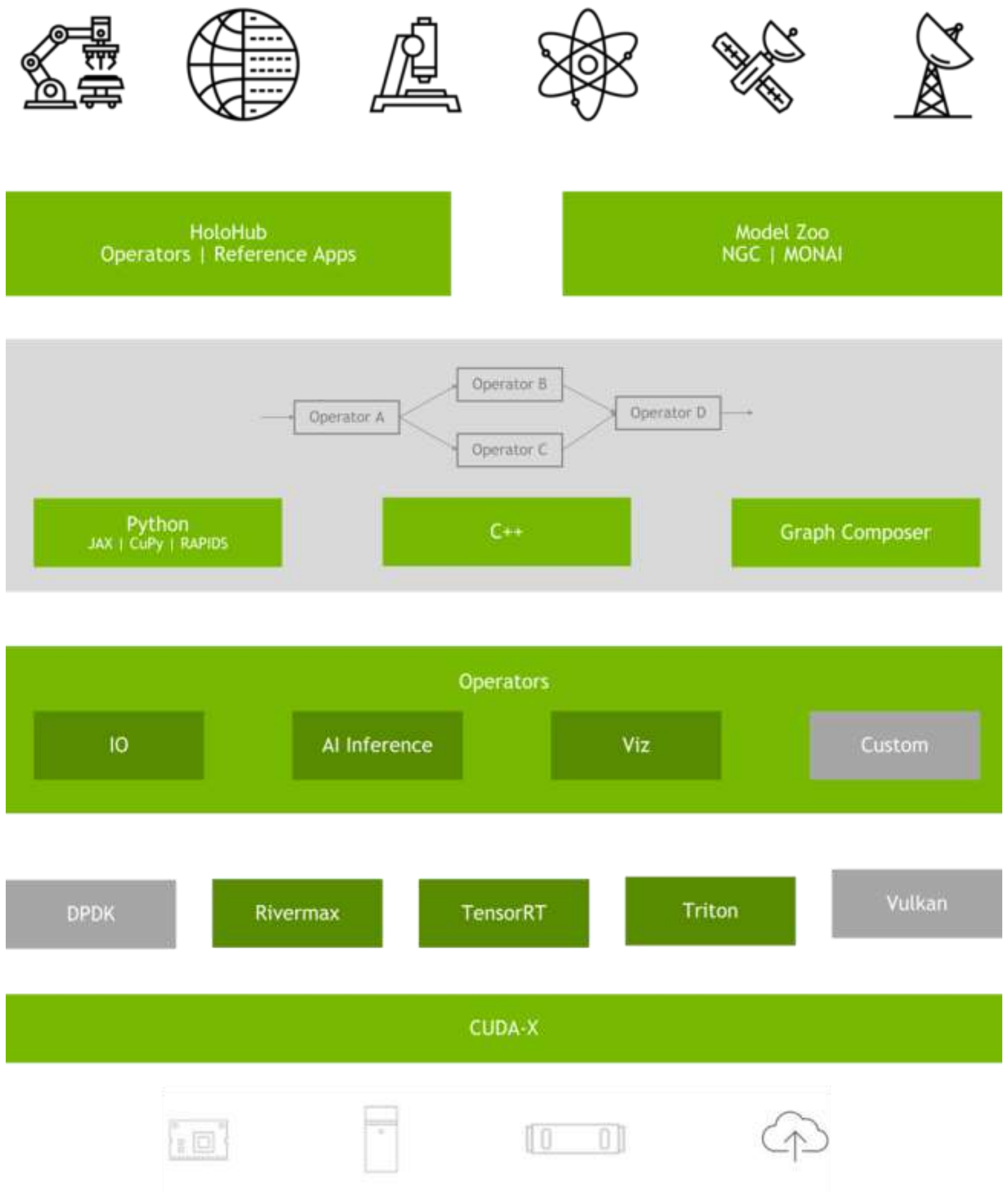
Software Defined

Low Latency, High Throughput

Scalable from Edge to Datacenter

NVIDIA Holoscan

SDK for Building AI-Enabled Sensor Processing Applications



Features

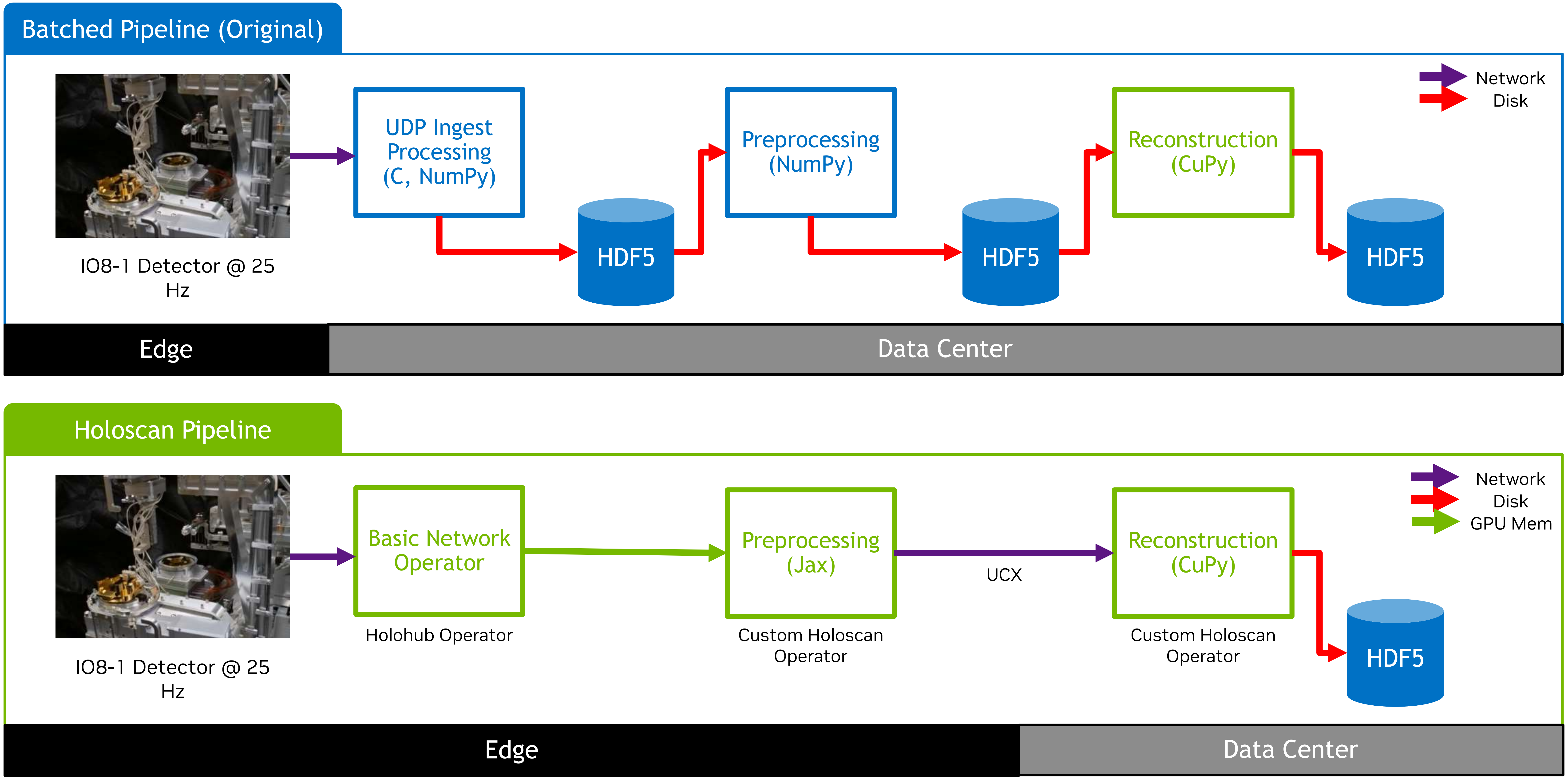
- C++ and Python APIs for domain agnostic sensor data processing workflows
- Multi-Node and Multi-GPU support with advanced pipeline scheduling options and network-aware data movement
- AI Inference with pluggable backends such as ONNX, Torchscript, and TensorRT
- Scalable from IGX Orin (ARM + GPU) to DGX (x86 + A100)
- Apache 2 Licensed and Available on GitHub

Benefits

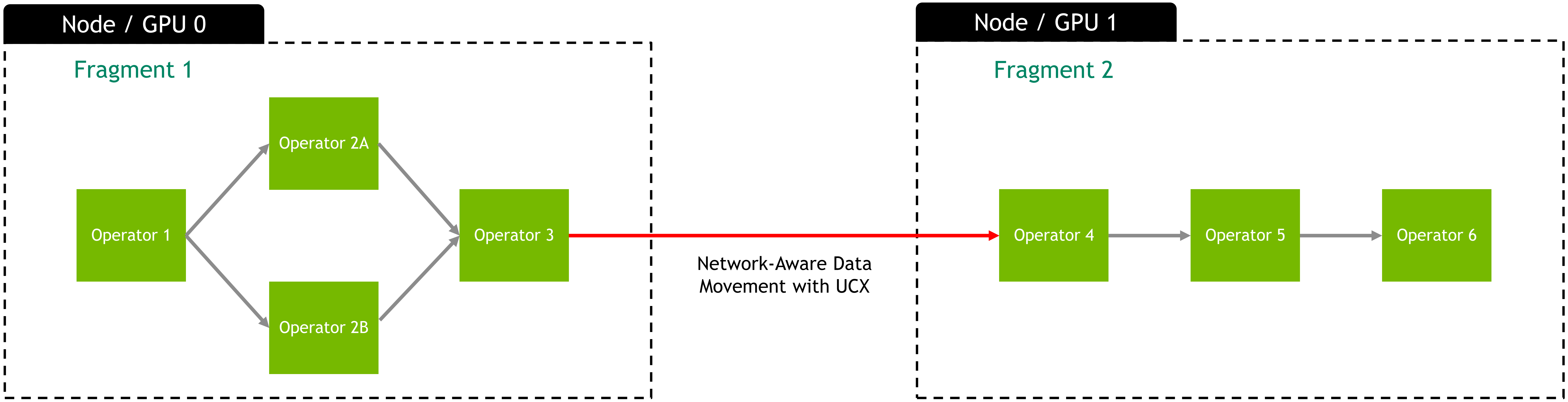
- Simplifies sensor I/O to GPU
- Simplifies the deployment of an AI model in a streaming pipeline
- Provides customizable, reusable, and flexible components to build and deploy GPU-accelerated algorithms
- Scale workloads with Holoscan's distributed computing features
- Deploy to the Cloud with Holoscan Cloud Native and Holoscan App Packager

Holoscan Ptychography Pipeline

Collaboration with Diamond Light Source – **4x Reduction in User Wait Time with Holoscan**



Holoscan Fundamentals



Holoscan Applications are built by forming a graph of either core or custom Holoscan Operators. Operators are the fundamental unit of work in Holoscan and can define I/O, AI inferencing, visualization, and accelerated computing functions

Holoscan Fragments define hardware locality of a given series of connected Operators. Data movement within a Fragment is facilitated via shared GPU pointers

Schedulers

Greedy Scheduler

Uses single CPU thread to launch operators in a pipeline sequentially

Multi-Threaded Scheduler

Can pin operators to a specific CPU thread for async execution and pipeline parallelism

Profiling Tools

Data Flow Tracking

Tracks data latency as a packet/frame moves through a Holoscan application

Nsight Systems

Observe overall application behavior and performance

Holohub

A Repository for Hosting Sample Holoscan Applications and Operators

Repository: https://github.com/nvidia-holoscan/holohub

Sensor I/O

Basic Network Operator

Linux Sockets

Advanced Network Operator

DPDK, GPUDirect RDMA

AI + Sensor Processing

SDR FM Demodulation

FM Demodulation + Speech to Text Transcription

Face Detection

TAO Pretrained Model on Video Stream

GPU Accelerated Sensor Processing

Simple Radar Application

Python and C++ Examples on Traditional Radar Pipeline

Orthorectification

GPU Accelerated Orthorectification with OptiX

SAR Image Formation

Python and C++ Examples for Multiple Algorithms

Who Is Using Holoscan

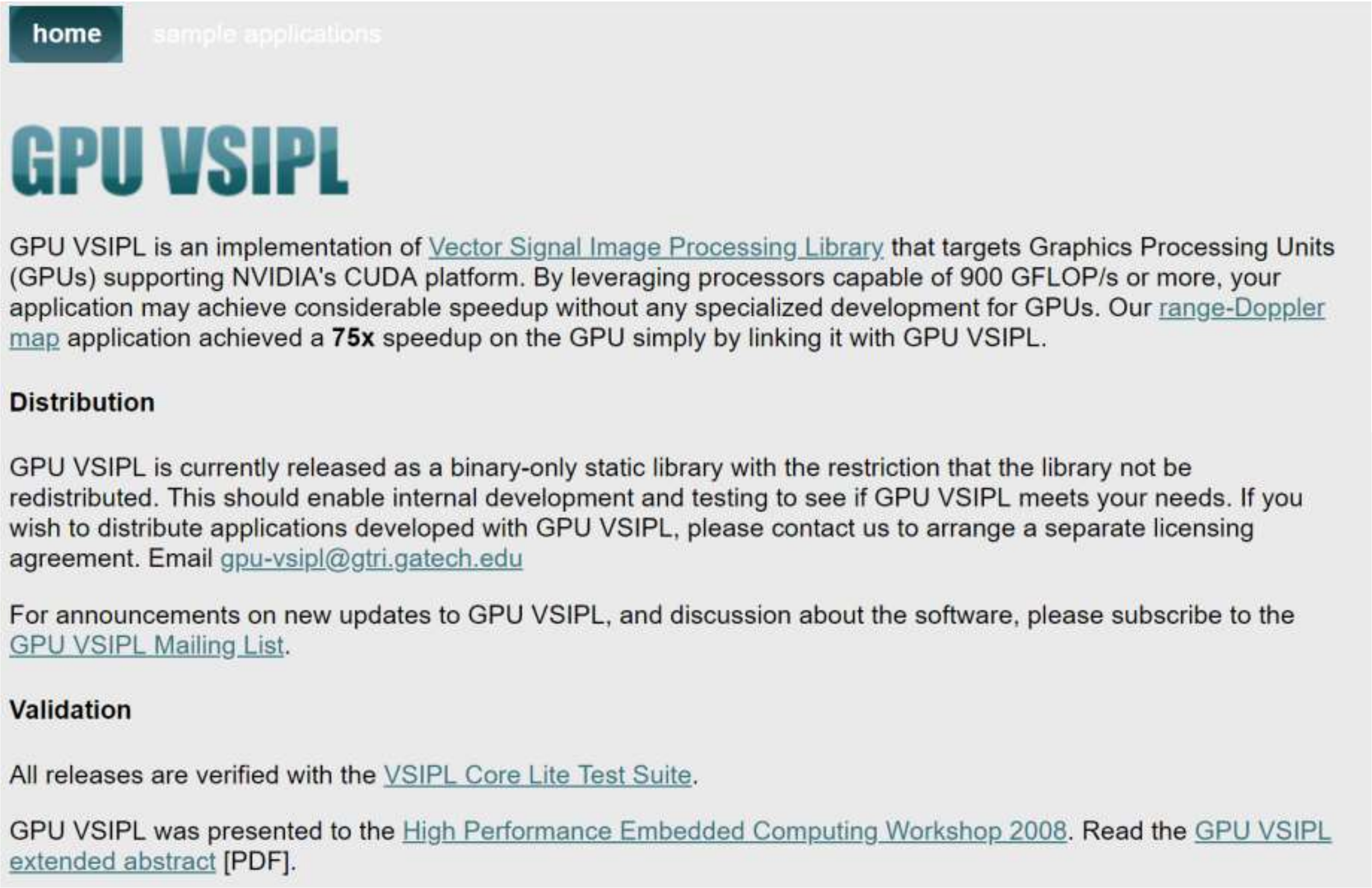
A Glimpse at Initial Applications

Customer	Domain	Application	Why Holoscan?
Diamond Light Source	Scientific Computing	Ptychography – High Resolution X-Ray	Batched -> Streaming Processing
Argonne National Laboratory	Scientific Computing	X-Ray Photon Correlation Spectroscopy	Batched -> Streaming Processing
Lawerence Berkeley National Laboratory	Scientific Computing	4D STEM Microscopy	Science Programmable Edge
Lawrence Livermore National Laboratory	Scientific Computing	High Speed Instrument Command and Control	Integration with Existing Instrument Frameworks
SETI / Breakthrough Listen #1	Radio Astronomy	Correlation and Digital Beamforming	High Speed I/O to GPU Compute
SETI / Breakthrough Listen #2	Radio Astronomy	Narrowband ML Inferencing	Online ML Inferencing
Analog Devices	Test & Measurement	Platform Enablement	High Speed I/O to GPU Compute
Georgia Tech Research Institute #1	Aerospace and Defense	Radar Signal Processing	High Speed I/O to GPU Compute
Georgia Tech Research Institute #2	Aerospace and Defense	Automatic Emitter Identification	Online ML Inferencing

Prototyping and Developing GPU Accelerated Applications

History of Signal Processing on NVIDIA GPUs

High Level Abstractions to Fast Compute



home sample applications

GPU VSIPL

GPU VSIPL is an implementation of Vector Signal Image Processing Library that targets Graphics Processing Units (GPUs) supporting NVIDIA's CUDA platform. By leveraging processors capable of 900 GFLOP/s or more, your application may achieve considerable speedup without any specialized development for GPUs. Our range-Doppler map application achieved a **75x** speedup on the GPU simply by linking it with GPU VSIPL.

Distribution

GPU VSIPL is currently released as a binary-only static library with the restriction that the library not be redistributed. This should enable internal development and testing to see if GPU VSIPL meets your needs. If you wish to distribute applications developed with GPU VSIPL, please contact us to arrange a separate licensing agreement. Email gpu-vsipl@gtri.gatech.edu

For announcements on new updates to GPU VSIPL, and discussion about the software, please subscribe to the GPU VSIPL Mailing List.

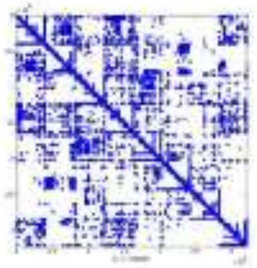
Validation

All releases are verified with the VSIPL Core Lite Test Suite.

GPU VSIPL was presented to the High Performance Embedded Computing Workshop 2008. Read the GPU VSIPL extended abstract [PDF].

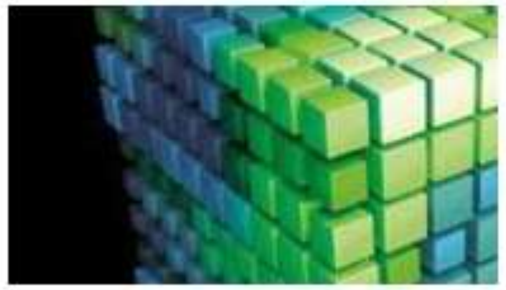
cuFFT

GPU-accelerated library for Fast Fourier Transforms



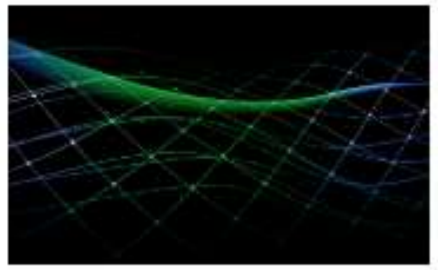
cuSPARSE

GPU-accelerated BLAS for sparse matrices



cuBLAS

GPU-accelerated standard BLAS library



cuSOLVER

Dense and sparse direct solvers for Computer Vision, CFD, Computational Chemistry, and Linear Optimization applications

cuSignal – Python GPU-Accelerated Signal Processing Library

cuSignal - Selected Algorithms

GPU-accelerated SciPy Signal (Python)

Convolution
- Convolve/Correlate
- FFT Convolve
- Convolve/Correlate 2D

Filtering and Filter Design
- Resampling – Polyphase, Upfirdn, Resample
- Hilbert/Hilbert 2D
- Wiener
- Firwin, FIR Filter

Waveform Generation
- Chirp
- Square
- Gaussian Pulse

Window Functions
- Kaiser
- Blackman
- Hamming
- Hanning

Spectral Analysis
- Periodogram
- Welch
- Spectrogram

Phased Array
- Ambgfun
- MVDR

Peak Finding

Full List of Supported Functions – cuSignal Docs

SciPy Signal – Polyphase Resampler

```
import numpy as np
from scipy import signal

start = 0
stop = 10
num_samps = int(1e8)
resample_up = 2
resample_down = 3

cx = np.linspace(start, stop, num_samps, endpoint=False)
cy = np.cos(-cx**2/6.0)

%%timeit
cf = signal.resample_poly(cy, resample_up, resample_down, window=('kaiser', 0.5))
```

2x Xeon E5-2600: 2.36 seconds

cuSignal – Polyphase Resampler

```
import cupy as cp
import cusignal

start = 0
stop = 10
num_samps = int(1e8)
resample_up = 2
resample_down = 3

cx = cp.linspace(start, stop, num_samps, endpoint=False)
cy = cp.cos(-cx**2/6.0)

%%timeit
cf = cusignal.resample_poly(cy, resample_up, resample_down, window=('kaiser', 0.5))
```

NVIDIA A100: 4.69 milliseconds, 503x SciPy Signal (CPU)

Speed of Light Performance - A100

timeit (7 runs); Benchmarked with ~1e8 sample signals, float64

Method	SciPy Signal (ms)	cuSignal (ms)	Speedup (xN)
fftconvolve	27300	46.6	585.8
correlate	4020	28.3	142.0
resample	14700	15.4	954.5
resample_poly	2360	4.6	513.0
welch	4870	23.5	207.2
spectrogram	2520	13.2	190.9
convolve2d	8410	6.04	1392.3

Learn more about cuSignal functionality and performance by browsing the notebooks

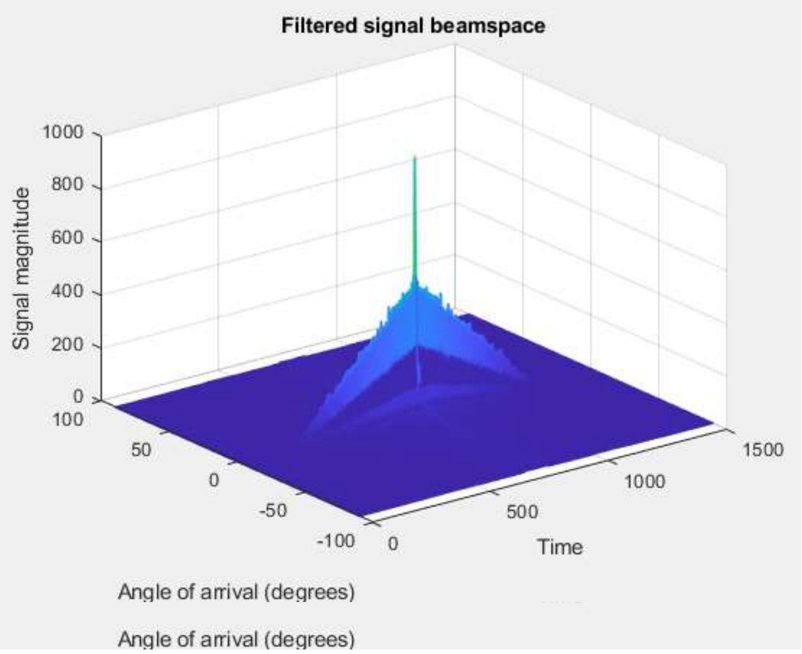
Digital Beamforming Example – Georgia Tech Research Institute

Developer and GPU Speedups with ~4 Hours of Work

MATLAB*

~174 seconds

```matlab
function [filteredData] = TDBeamform(elemLocations, samplingFrequency, steerDirection, data)

assert(size(elemLocations, 1) == size(data, 2), "elemLocations row size must equal data column size");
numElements = size(elemLocations, 1);

% Compute the time delays
C = 2.997e8;
steerDirection = steerDirection./sqrt(sum(steerDirection.^2));
sampleDelays = (samplingFrequency./C).*(elemLocations*steerDirection)';

% Compute the delay filters
numTaps = 301;
halfLength = idivide(int32(numTaps), int32(2));

tapLocations = double(-halfLength:(numTaps - halfLength - 1))';
delayFilterTaps = repmat(tapLocations, [1, numElements]) + sampleDelays;

delayFilters = sinc(delayFilterTaps);

% apply the delay filters
filteredData = Filter(data, delayFilters);
filteredData = mean(filteredData, 2);

end
```

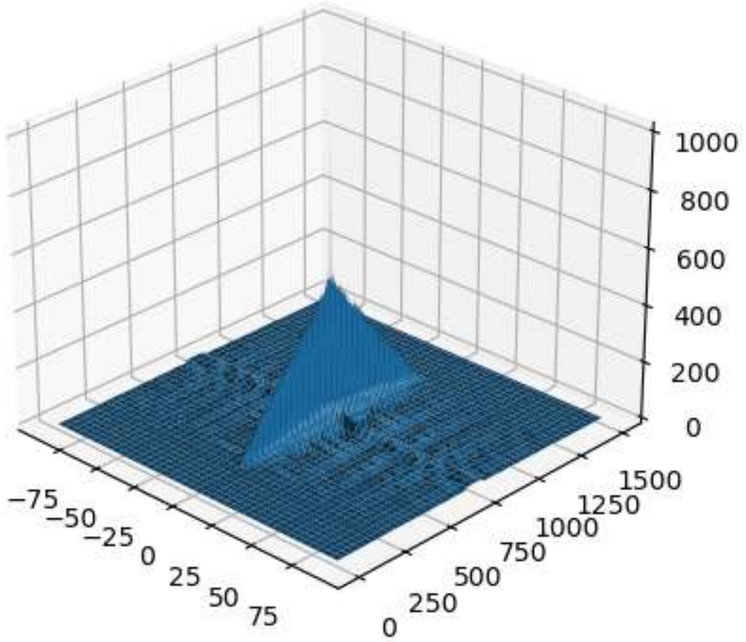


~150x Speedup

CuPy/cuSignal

P100* – 3.16 seconds | A100 – 1.15 seconds

```python
def TDBeamform(elemLocations, samplingFrequency, steerDirection, data):
    numElements = elemLocations.shape[0]
    steerDirection = steerDirection / cp.sqrt(cp.sum(cp.power(steerDirection, 2),axis=0))
    sampleDelays = (fc / C) * cp.matmul(elemLocations, steerDirection)

    numTaps = 301
    halfLength = numTaps // 2

    tapLocations = cp.expand_dims(cp.arange(-halfLength, numTaps - halfLength), 1)
    delayFilterTaps = cp.tile(tapLocations, (1, numElements)) + cp.ravel(sampleDelays)

    delayFilters = cp.sinc(delayFilterTaps)

    filteredData = cp.mean(cusignal.fftconvolve(inSignal.T, delayFilters.T, mode='same', axes=1), axis=0)
    return filteredData
```



*Double Precision, Profiled on same node, Intel 2x Xeon E5-2600 with P100

Get Started

NOTE: As of CuPy v13, cuSignal has Transitioned to cupyx.scipy.signal

[deprecated]: https://github.com/rapidsai/cusignal

414,220 Anaconda Downloads

702 GitHub Stars

43 Contributors

https://github.com/cupy/cupy

pip install cupy-cuda12x

MatX: A C++ Header Only Library for GPU-Accelerated Numerical Computing

Bridging Flexibility, Ease-of-Use, and Performance

GPU-Accelerated Numerical Computing Software for Every Type of Developer

PYTHONISTA

CuPy

Numba

PyTorch

TensorFlow

Python Bindings to CUDA

LLVM to CUDA

C/C++ GURU

cuBLAS

cuSOLVER

cuFFT

cuRAND

CUTLASS

Optimized CUDA Libraries

MatX – C++17 Template Library for Numerical Computing

Design Overview

Features

👍 Ease of Use:

Straightforward programming model with familiar interfaces (MATLAB/Python-like)

Wraps existing libraries like cuFFT, CUTLASS, and cuRAND

Easily customizable

🔥 High Performance:

Prioritization of efficiently handling streaming data

Separates allocation and processing

Key Concepts

MatX leverages CUDA Managed Memory, freeing the developer from worrying about data locality

Compute operations are performed on arbitrary-rank tensors -- lightweight descriptors of data either on host or device. Tensors are accepted in *all* MatX functions (like a NumPy ndarray)

Zero data movement view manipulations (clone, slice, permute) that can be chained together at compile-time

Supports many transforms: FFT, convolution, filtering, GEMM, pointwise operators, and contraction

Supports host and device execution with minimal code changes

MatX API Examples

Initialize a 2D tensor with data:	`A = {{1, 2, 3}, {4, 5, 6}};`
Add two tensor element-wise and scale:	`(A = (A + B) / 5.0).run();`
Perform a traditional GEMM:	`(C = matmul(A, B)).run();`
Perform an in-place FFT:	`(A = fft(A)).run();`
Batch sort a 4D tensor by rows:	`(t4_sort = sort(t4)).run();`

MatX/Python Comparison

FFT Based Resampler - No Windowing

Python

```python
import numpy as np
from numpy import fft as fft

N = min(num_samp, num_samp_resamp)
nyq = N // 2 + 1

# Create an empty vector with num_samps elements
sig = np.empty(num_samp)

# Real to complex FFT, time to freq domain
fft_sig = fft.rfft(sig)

# Slice
slice_sig = fft_sig[0:nyq]

# Complex to real IFFT
resamp_sig = fft.irfft(slice_sig, num_samp_resamp)
```

MatX

```cpp
uint32_t N = std::min(num_samp, num_samp_resamp);
uint32_t nyq = N / 2 + 1;

auto sigView        = make_tensor<float, 1>({num_samp});
auto sigViewComplex = make_tensor<complex, 1>({num_samp/2+1});
auto resampView     = make_tensor<float, 1>({num_samp_resamp});

// Real to Complex FFT, time to freq domain
(sigViewComplex = fft(sigView)).run(stream);

// Slice to half spectrum based on num_samp_resamp
auto sliceView = slice(sigViewComplex, {0}, {nyq});

// Complex to Real IFFT, back to time domain
(resampView = ifft(sliceView)).run(stream);
```

5.36s (Xeon E5-2698v4 @ 2.20GHz)

5.48ms (V100)

~1000x improvement!

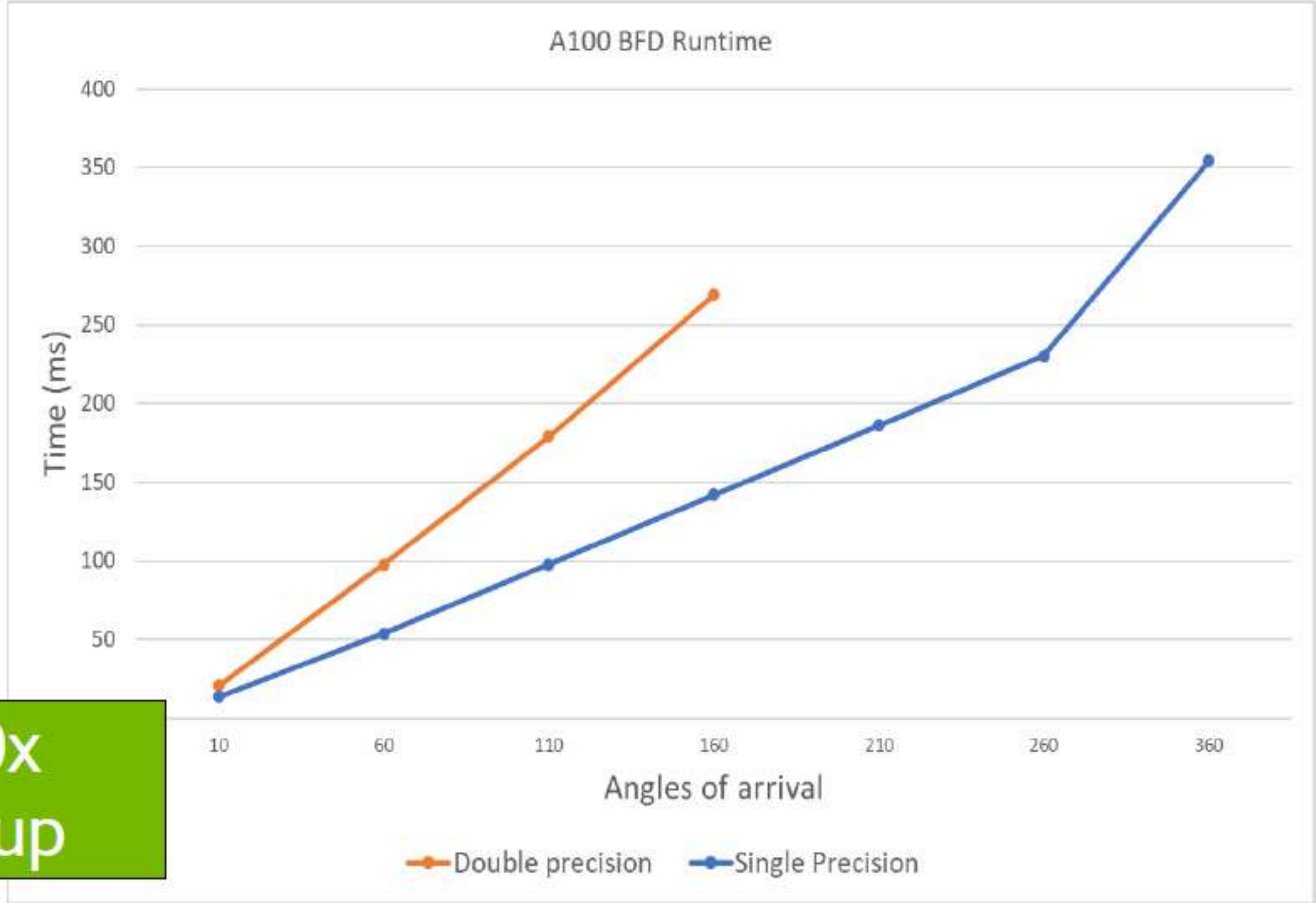
Revisiting Digital Beamforming with GTRI

Performance Optimizations with MatX

MatX (C++ and additional batching)

```cpp
steerDirection(steerDir, numAoas).run(stream);

matmul(steeredEl, steerDir, elLocations, stream);

auto sampClone    = sampleDelays.Clone({matx::matxKeepDim, numTaps, matx::matxKeepDim});

auto tapLocations = matx::range<0>({numTaps, 1}, -halfLength, 1);
auto reptaps      = matx::repmat(tapLocations, {1, elCount}) + sampClone;
(delayFilterTaps  = matx::sin(M_PI * reptaps) / (M_PI * reptaps)).run(stream);

(dftPermute = delayFilterTapsFull.Permute({0,2,1})).run(stream);

fft(fft_filt, dftPermute, fft_len, stream);

(fft1_mult = r2cop(fft_filt, fft_len) * fft_sig).run(stream);

ifft(fft1_mult, fft1_mult, 0, stream);

 mean(means, fft1_mult.Permute({0,2,1}), stream);

(matchedFilter = matx::repmat(matx::conj(matx::reverseX(matSignal)), {numAoas, 1})).run(stream);

fft(matchedFilterFreq, matchedFilter, fft_len, stream);

fft(means, means, 0, stream);

(matchedFilterFreq = matchedFilterFreq * means).run(stream);

ifft(matchedFilterFreq, matchedFilterFreq, 0, stream);
```

~1500x Speedup



"The A100 results are more than enough for a real-time processor, which is our ultimate goal" - Tim Andersen, GTRI

Get Started

Join the MatX Community!

1.1k GitHub Stars

21 Contributors

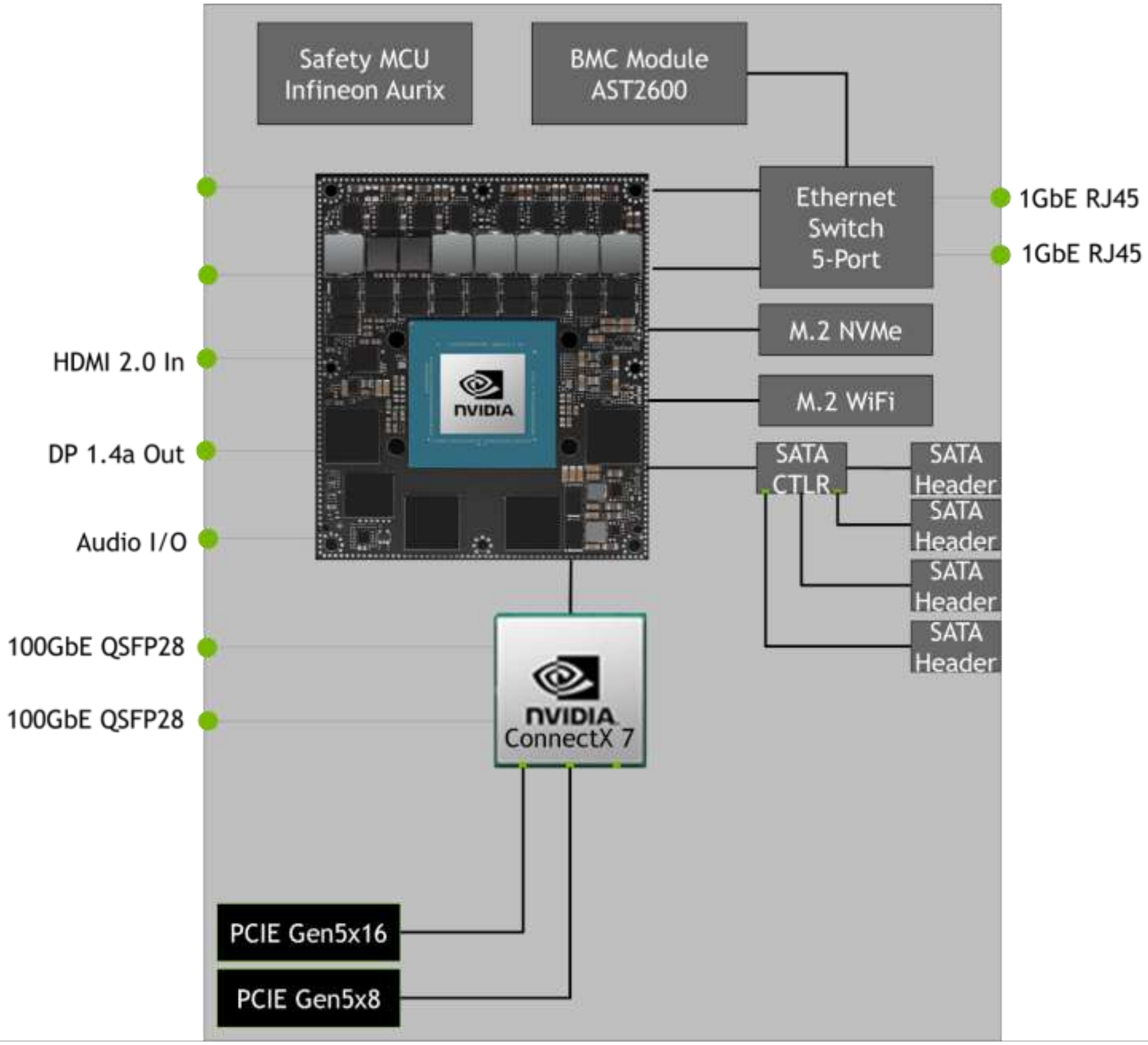
BSD-3 Licensed

https://github.com/NVIDIA/MatX

IGX – Embedded Platform for Enterprise Edge AI Processing

NVIDIA IGX Platform for Industrial-Grade Edge AI

HARDWARE

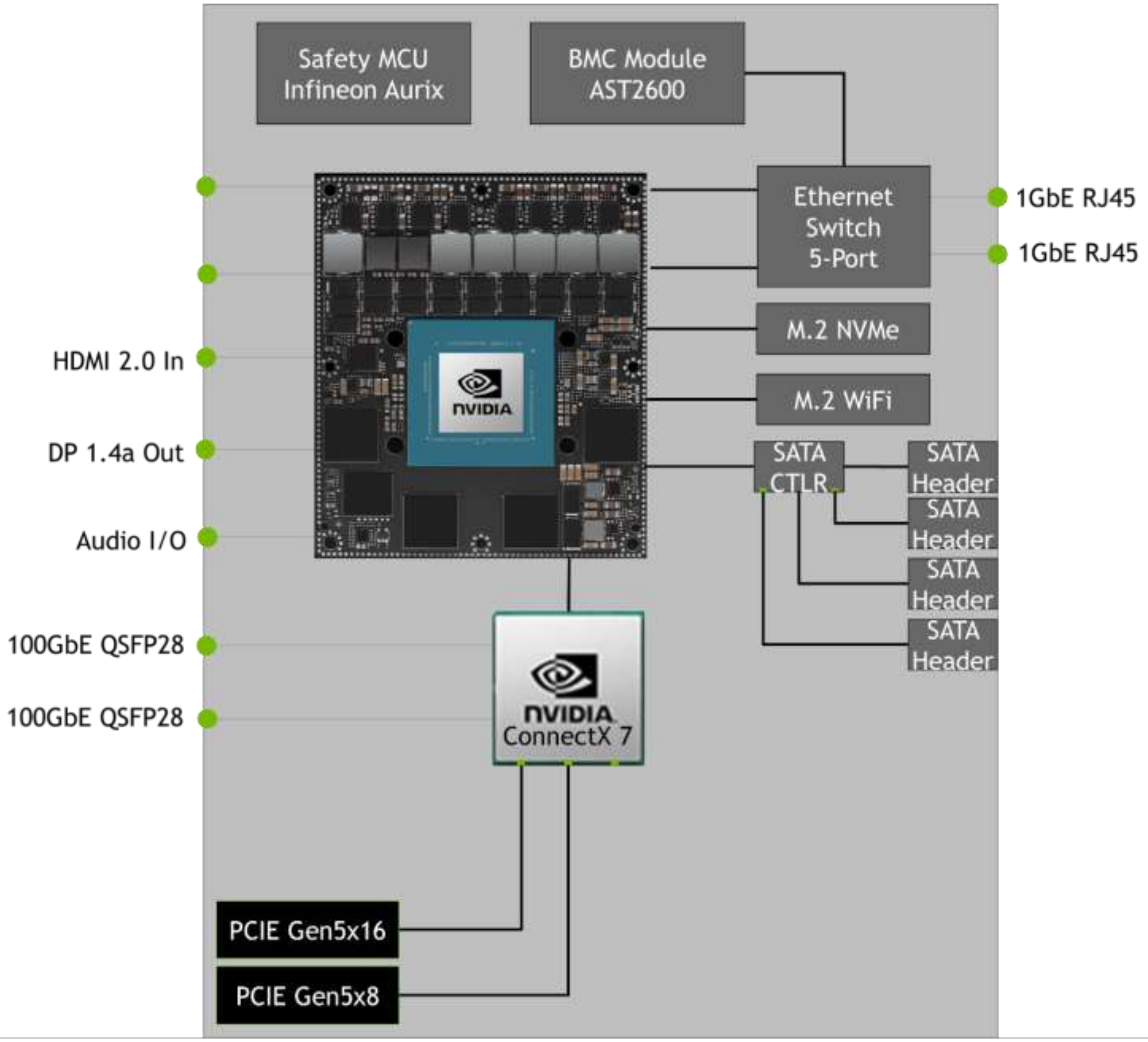


SOFTWARE

- CPU: 12-core Arm
- GPU: NVIDIA Ampere 2,048 CUDA, 64 Tensor
- Optional Discrete GPU card : NVIDIA A6000
- Smart NIC: NVIDIA ConnectX-7 (200 Gb/s of networking speed, ideal for ingesting high frame rate video)

NVIDIA IGX Platform for Industrial-Grade Edge AI

HARDWARE



FUNCTIONALITY

- **Safety MCU** – Certified safety RTOS monitoring your platform
  - Active attestation - all the sensors on IGX and the SoM being terminated at the sMCU and monitored in real-time. This is a prerequisite for Safety

- **BMC** – Designed to allow updates to all of the key components inc. Orin SoM, CX7, Safety MCU, Ethernet Switch, PCIe Switch, Ethernet Controller(s), and SATA Controller
  - Prerequisite for OTA for OS & MCU

- **ConnectX-7**
  - High Bandwidth networking –ability to ingest "raw" video at high FPS eg. Industrial Defect camera running @ 500-1000FPS. You can't compress these types of feeds so you need ultra-high BW to cope with them.
  - TLS, MACSec or IPSec offload engines for Ground-Cloud/Cloud-Ground data protection/encryption
  - Rivermax support for GPUDirect UDP / GPUDirect RDMA
  - Precision Time protocol (PTP) support for accurate timing
  - PCIe Switch – Add capabilities beyond Orin SoM, ie Add discrete GPU

Bring You Own Sensor to Holoscan

Basic Network Operator Rx/Tx

Focus on **Simplicity**: Ingest UDP Ethernet to GPU at < 10Gbps

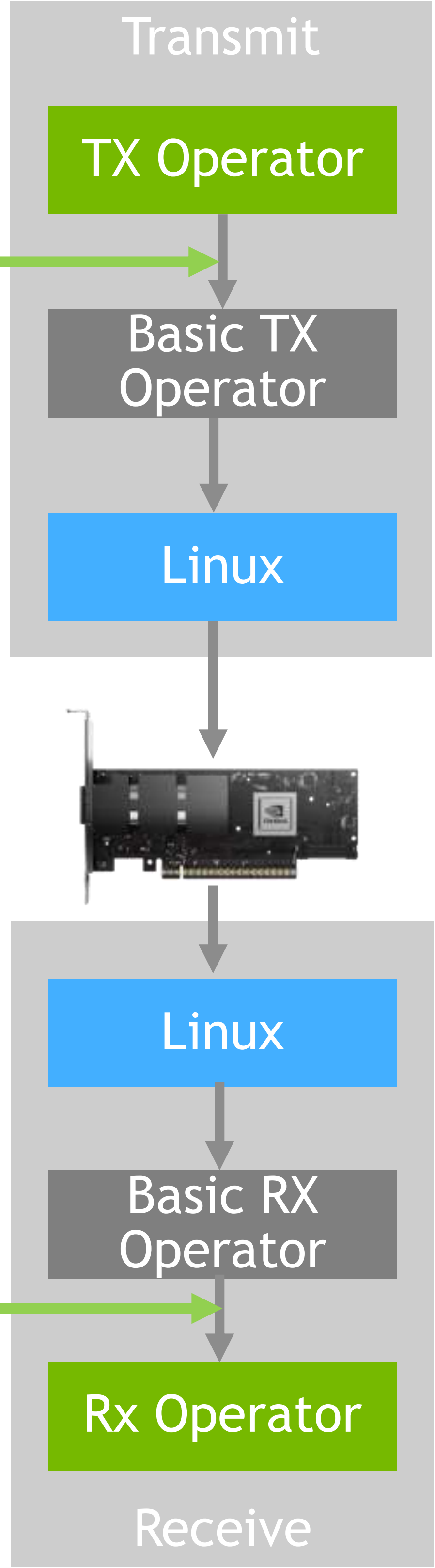
- Sockets provide a common interface to send and receive data to/from an abstract sink/source
  - Works on all Linux distributions and network cards
  - Supports both streaming and datagram protocols
  - Kernel provides protocol stacks
    - User doesn’t need to worry about retransmits, headers etc
  - Ideal for simple use cases requiring easy portability

- Cannot achieve line rate on modern NICs
  - All packets go through have at least one copy
  - User-space to kernel-space context switches
  - Small number of threads
  - No GPUDirect

- Get started with Basic Network Operator on Holohub

```
struct NetworkOpBurstParams {
  uint8_t *data;
  uint32_t len;
  uint32_t num_pkts;
};
```

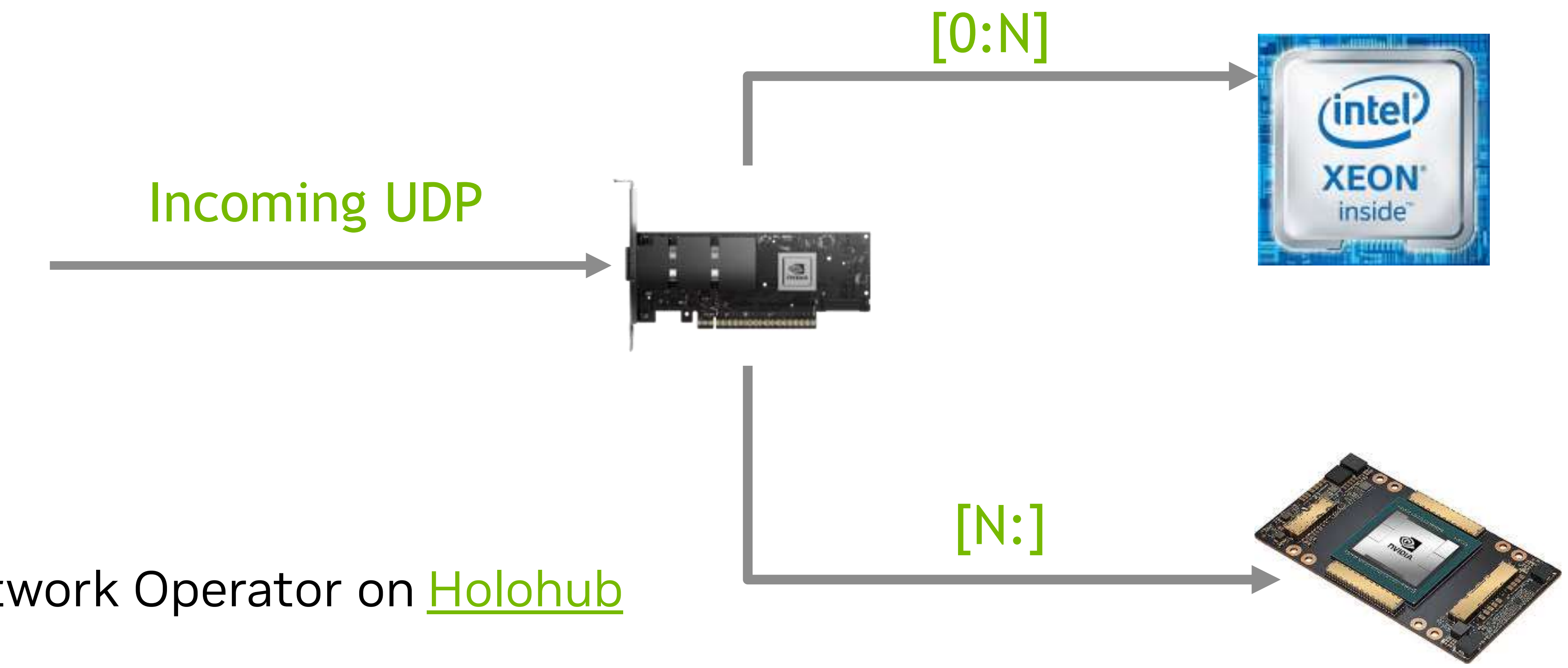
Simple message to send/receive



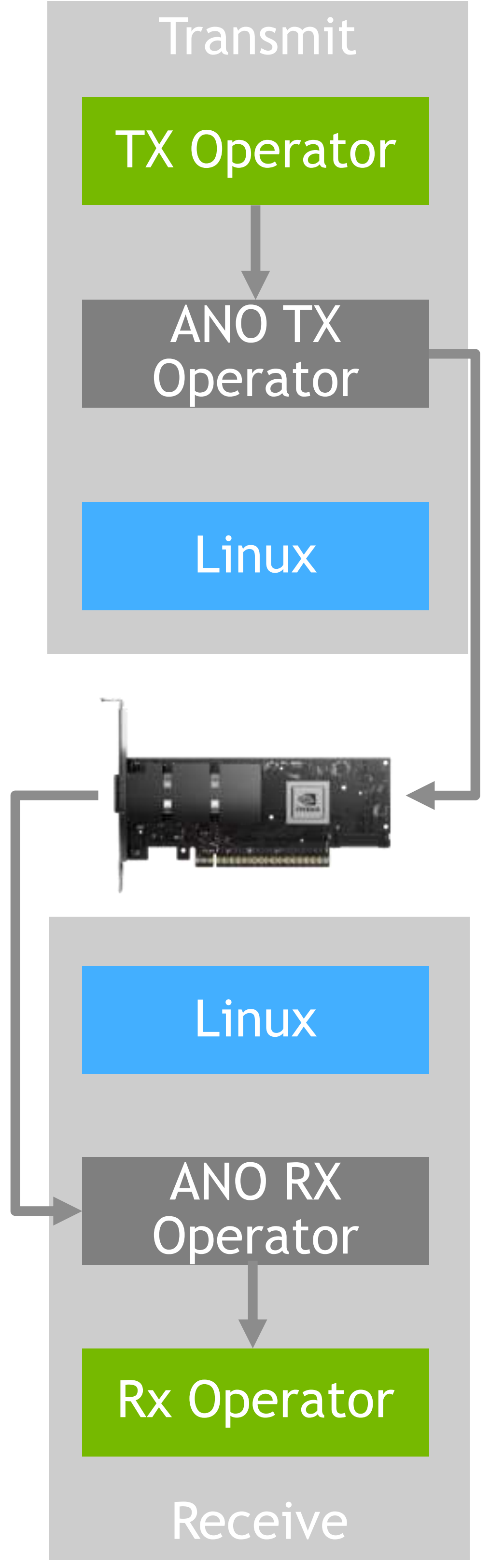
Advanced Network Operator

Focus on **Performance**: Ingest UDP Ethernet to GPU at Line Rate

- Bypasses Linux kernel for access directly to NIC DMA buffers
  - Achieves peak rates on any modern NIC
    - Scalable number of CPU cores to handle traffic in parallel
  - Utilizes DPDK for packet processing and IPC
    - Works on any NIC supported by DPDK
    - More libraries can be supported for new use cases without changing the API
  - **Zero-copy interface from NIC into user buffers or directly to GPU using GPUDirect via standard UDP packets**
    - No RDMA protocol necessary (RoCE/iWARP)
- GPUDirect supported with any legacy sending protocol (UDP, Ethernet, VITA-49, etc)
  - Different modes: **Header-Data Split (HDS), Batched,** or **Persistent Kernel**
- **Python bindings in progress**



- Get started with Advanced Network Operator on [Holohub]



5G Instrumentation with Holoscan

End-to-End Signal Processing at 120Gbps+

Holoscan Transmit Software Pipeline

Recorded 120Gbps I/Q RF Data → File Read → Advanced Network Operator TX → UDP Ethernet

Holoscan Receive Software Pipeline

UDP Ethernet → Advanced Network Operator RX → Pointer → Digital Down Conversion → Pointer → Error and Spectrum Calculation → Pointer → NATS Sender → Low Speed Network → Web GUI Visualization

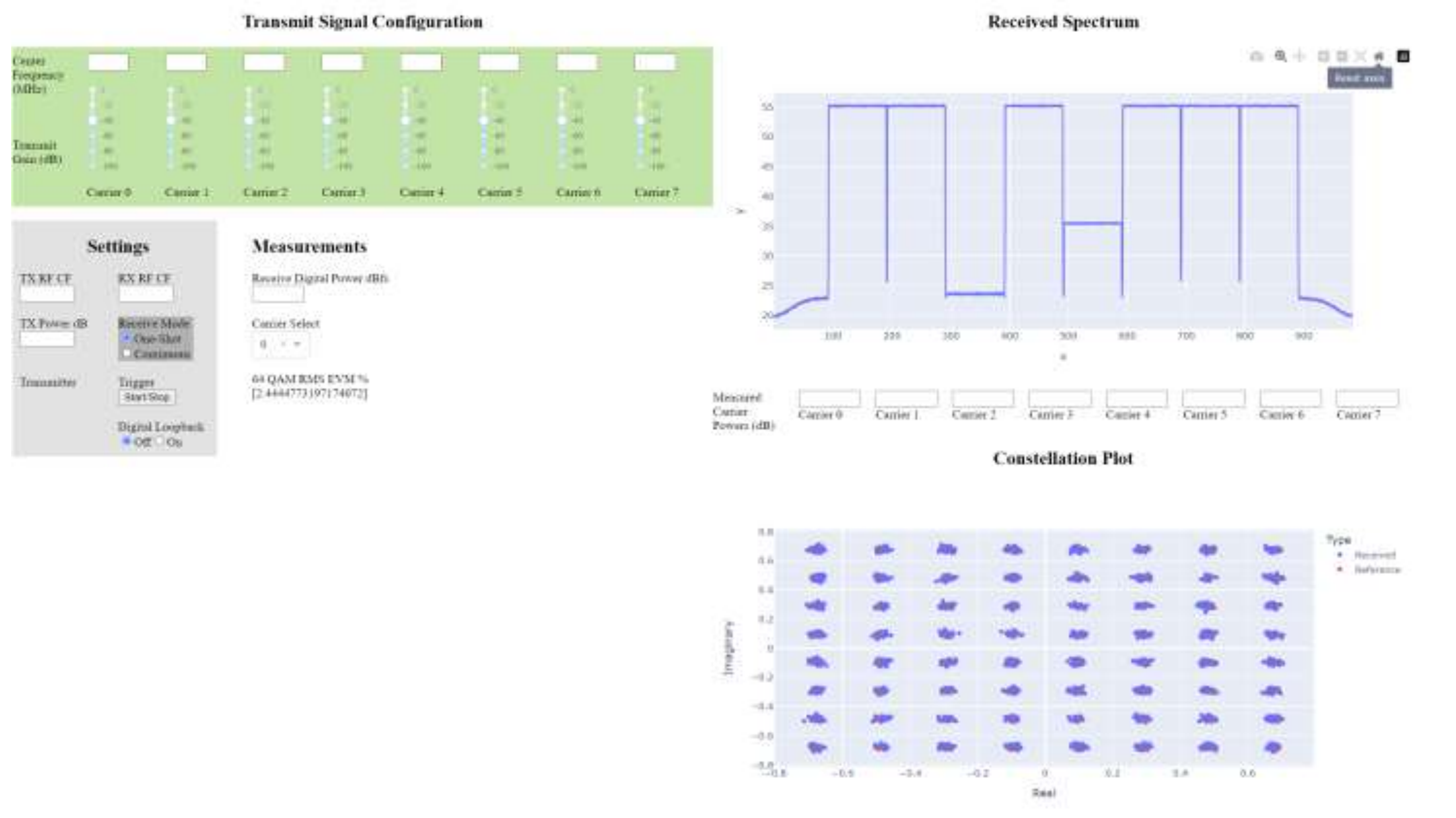
Hardware Configuration

Transmitter
IGX (Xavier, RTX6000)

UDP Ethernet

Receiver
Sensor Processor
IGX (Xavier, RTX6000)

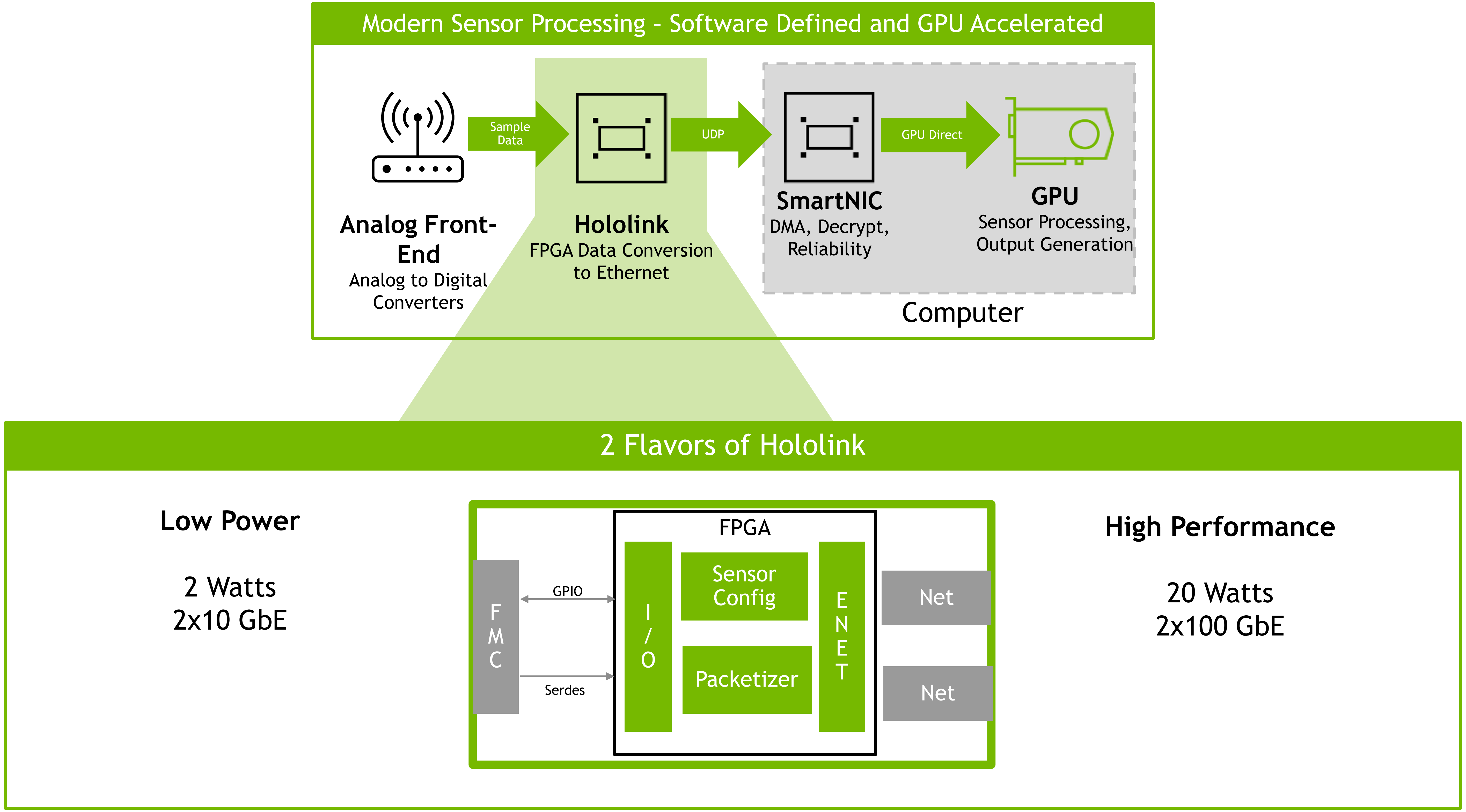
Real Time Visualization



Software Defined, Scalable Sensors with Hololink

Hololink

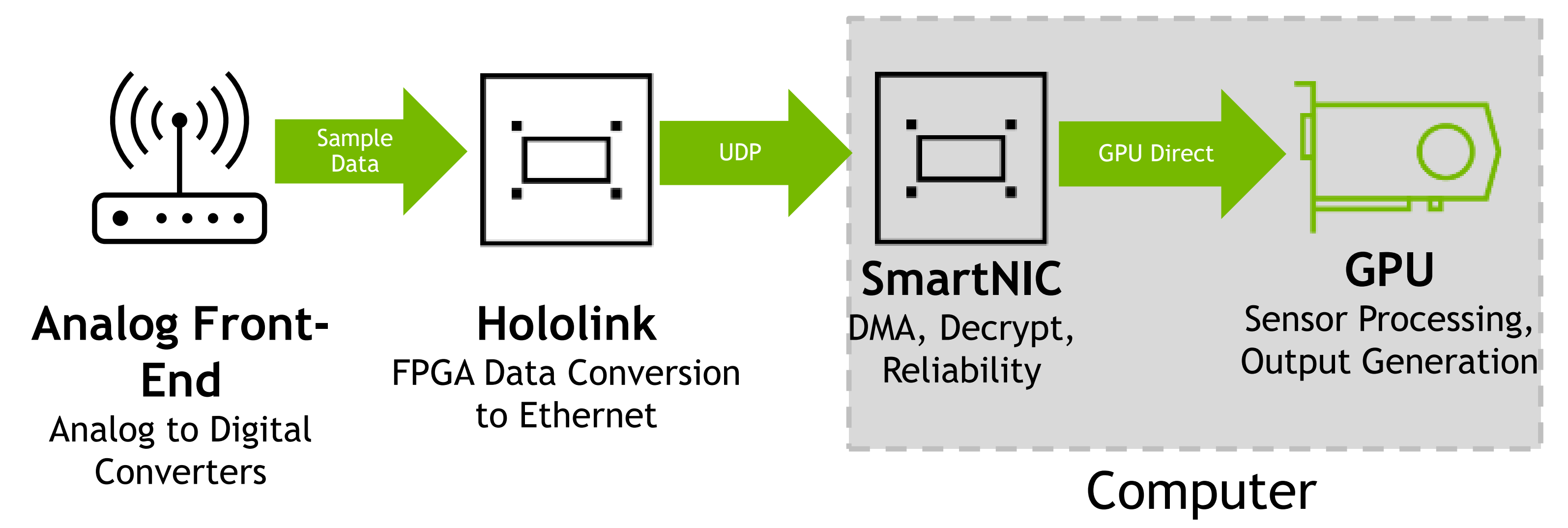
Combining Specialized Sensor I/O with GPU Computing and Holoscan



Sensor Plug and Play with Holoscan Operators and Hololink

Enabling Domain Agnostic Rapid Sensor Processing Design and Deployment with Holoscan

Modern Sensor Processing - Software Defined and GPU Accelerated



Holoscan Operators Deliver Plug-and-Play Capabilities with Data Converter DevKit

Operators can wrap existing packet processing libraries like DPDK, DOCA, and Rivermax

Abstracts sensor data movement from developer regardless of sensor data type

Unlocks immediate productivity, allowing sensor processing engineers to focus on new science

AD9986 Demo with Hololink 100G and IGX

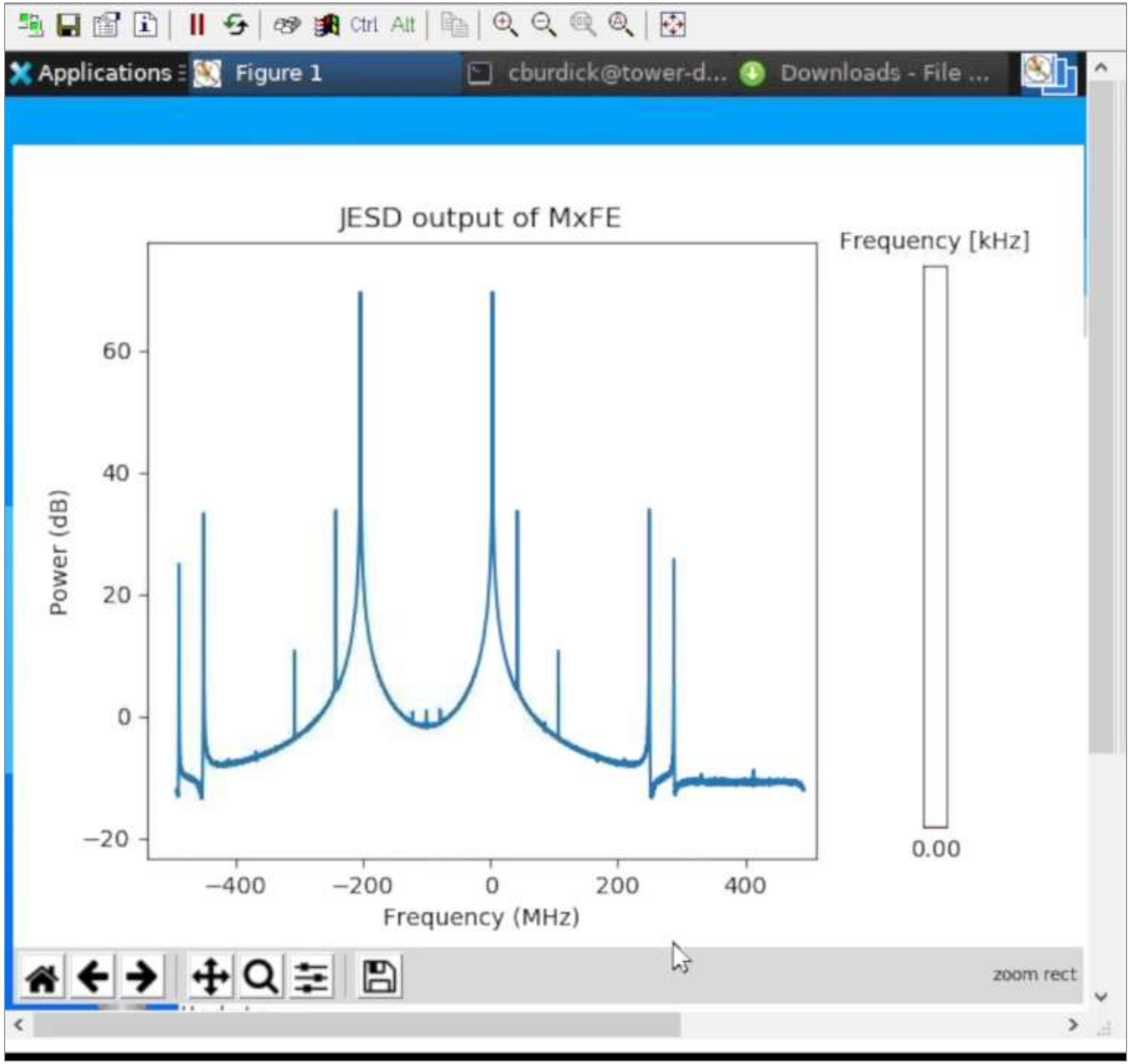
Playback of sinusoid from ADI 9986 MxFE with loopback

Packetization with Hololink

PSD generation with Holoscan

Currently operating at ~68Gbps due to MxFE channel limitations (2x at 34Gbps).

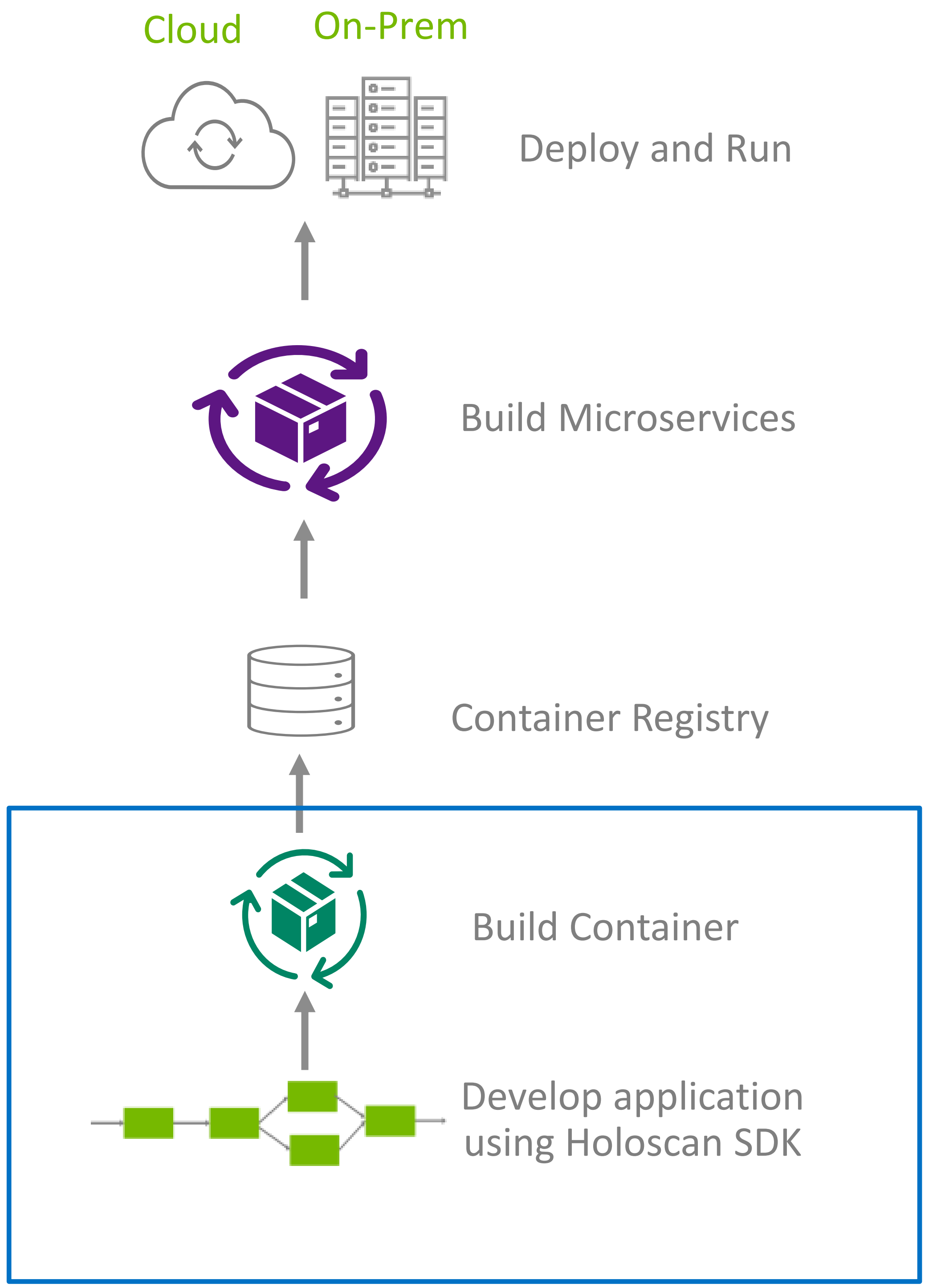
Successfully demonstrated 200Gbps on Hololink alone in loopback



Scaling Sensor Processing Pipelines

Holoscan Application Packager and Runner

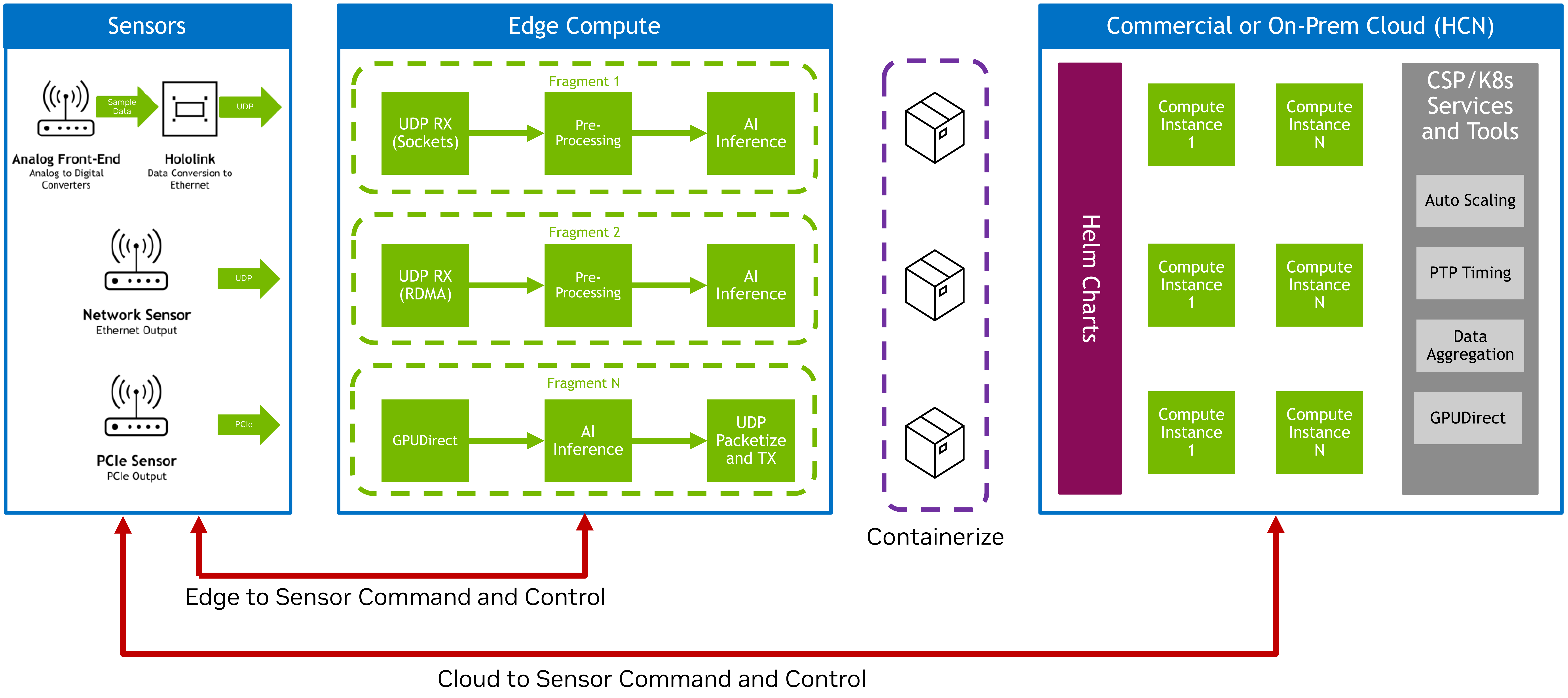
Simplifies Containerization, Packaging, and Testing of Holoscan Applications



- **Application Packager**: command line tool to package and containerize a given Holoscan application with support for both C++ and Python
  - Supports cross compilation, meaning development can be on an x86 platform while deployment is on an Arm system (e.g. IGX Orin)
- **Application Runner**: command line tool to run and test containerized Holoscan applications
  - Abstracts internal packaging details
- Both the Packager and Runner are **Open Container Initiative (OCI) compliant** and compatible with Docker, Kubernetes, and containerd

Rapid Data Analysis and Real Time Steering

Resilient Workflows with Holoscan and Holoscan Cloud Native



NVIDIA

Contributing and Getting Started

Getting Started with Holoscan

Holoscan References

https://github.com/nvidia-holoscan/holoscan-sdk

`docker pull nvcr.io/nvidia/clara-holoscan/holoscan:v1.0.0`

`pip install holoscan`

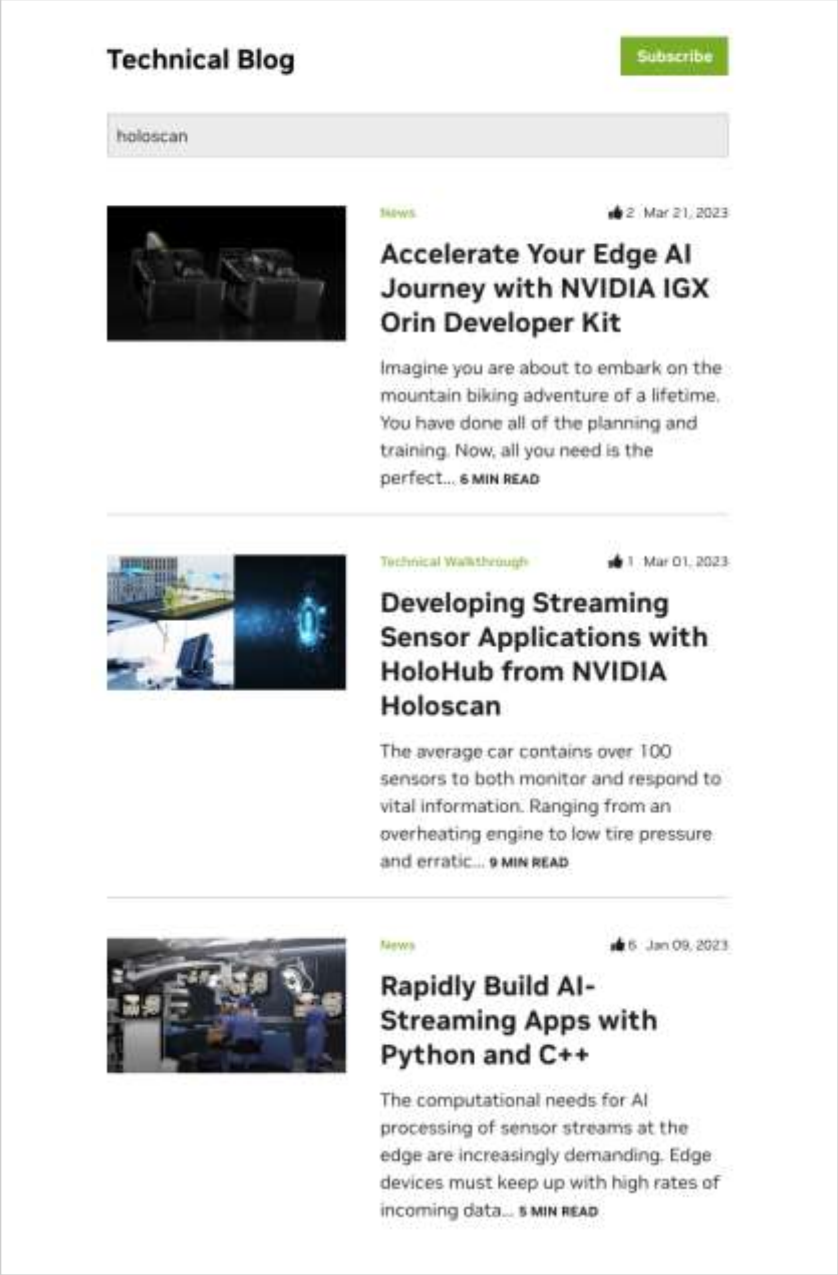
Debian Packages available on NGC

https://docs.nvidia.com/clara-holoscan/sdk-user-guide/index.html

Learn More about NVIDIA Holoscan

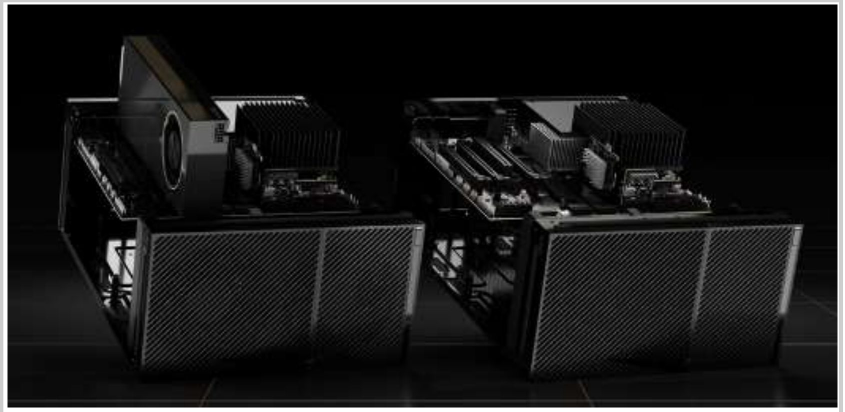
NVIDIA Holoscan Webpage

https://developer.nvidia.com/holoscan-sdk



Technical Blogs

https://developer.nvidia.com/blog/



EDOM TECHNOLOGY CO., LTD.

LEADTEK 丽台科技

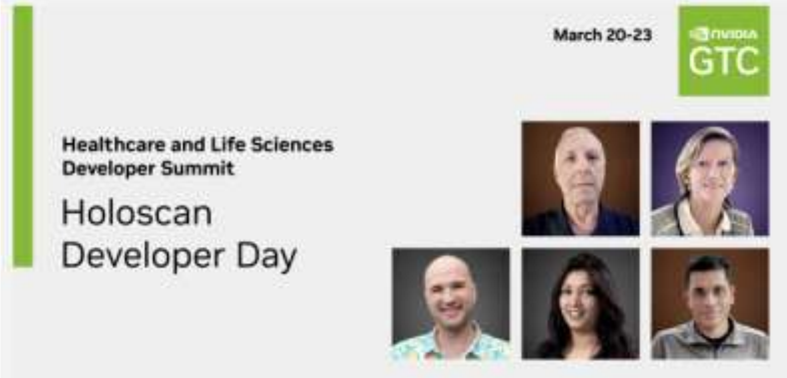
MACNICA

PNY

RYOYO

Order a DevKit

https://www.nvidia.com/en-us/edge-computing/products/igx/



On-Demand

Cosmo GI Genius Session - S51262
Learn How to Deploy and Deploy Real-Time AI Applications in Healthcare

MagicLeap Session - S52046
Digital Twins in Augmented Reality with Omniverse, Holoscan, and Magic Leap

GTC Spring 2023

Session Talks and Special Events

… And the Advanced Network Operator

ANO Benchmark App: https://shorturl.at/sAM28

ANO Holoscan Operator: https://shorturl.at/rsDFZ

Example RADAR App: https://shorturl.at/sxKPW

NVIDIA