

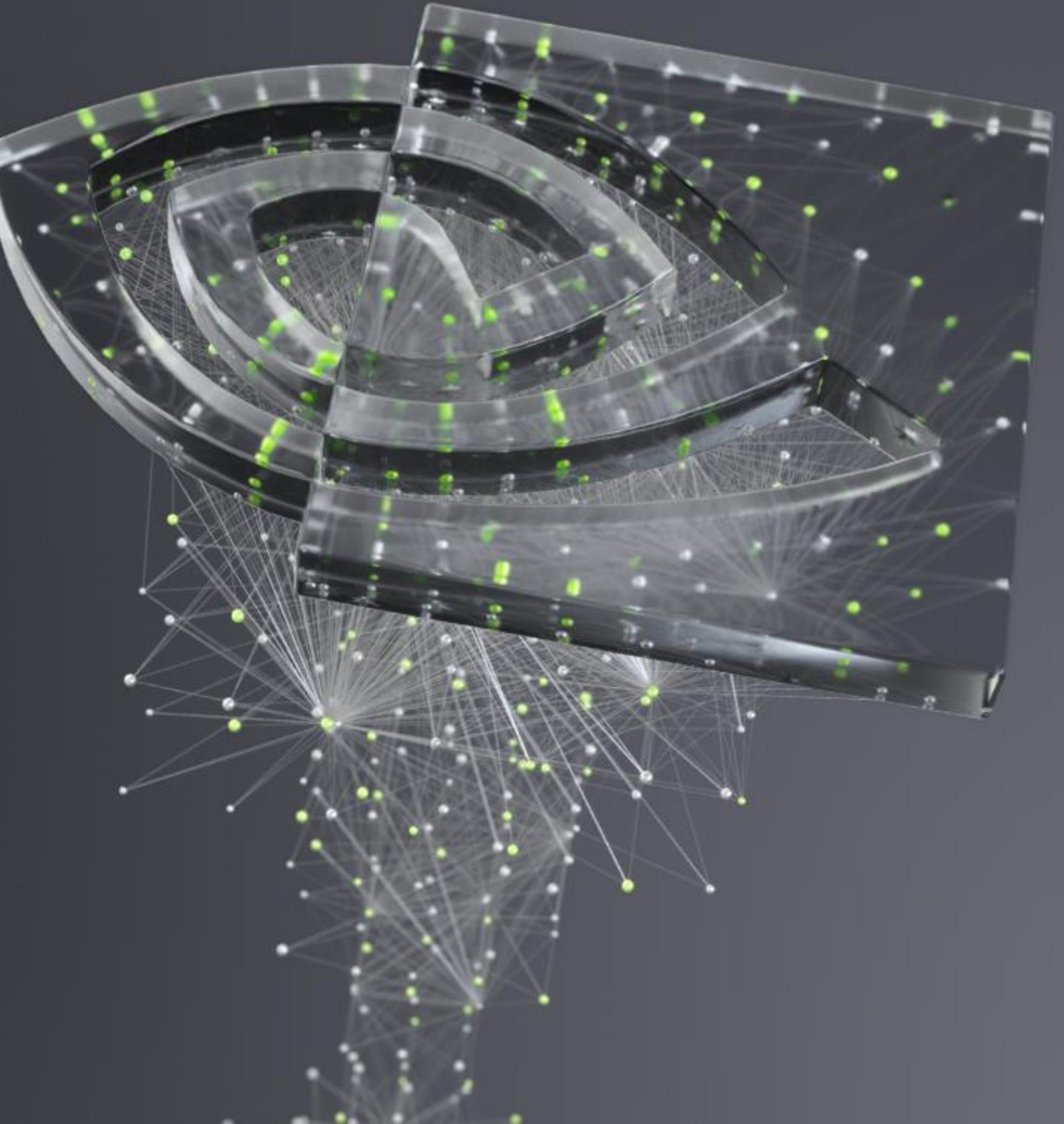


NVIDIA®

# ML FRAMEWORKS INTEROPERABILITY

Christian Hundt, Manager AI Developer Technology

Miguel Martínez, Sr. Deep Learning Data Scientist





# Today's Agenda

- About This Talk
- End-to-End Pipeline
- Common Bottlenecks:
  - *Memory Management*
  - *Data Conversion*
  - *Data Loading*
  - *Data Transfer*
- Summary



# ABOUT THIS TALK

# FRAMEWORK INTEROPERABILITY

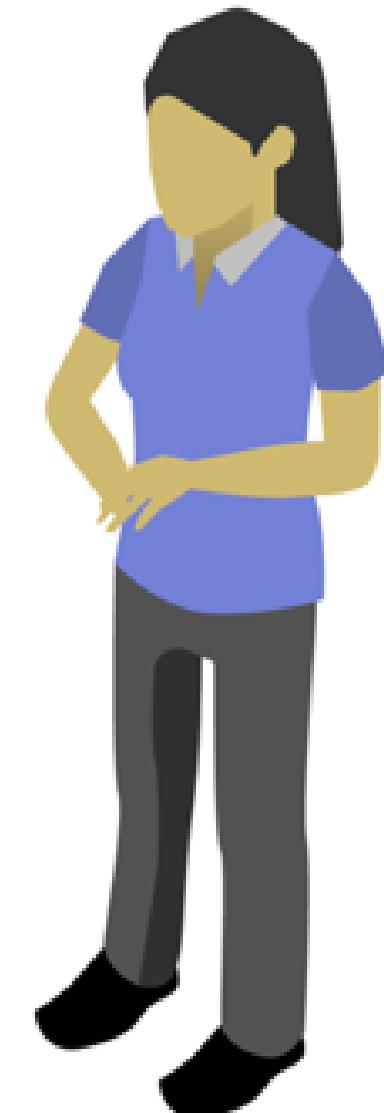
When a single framework is not enough



data  
scientist



CUDA  
developer



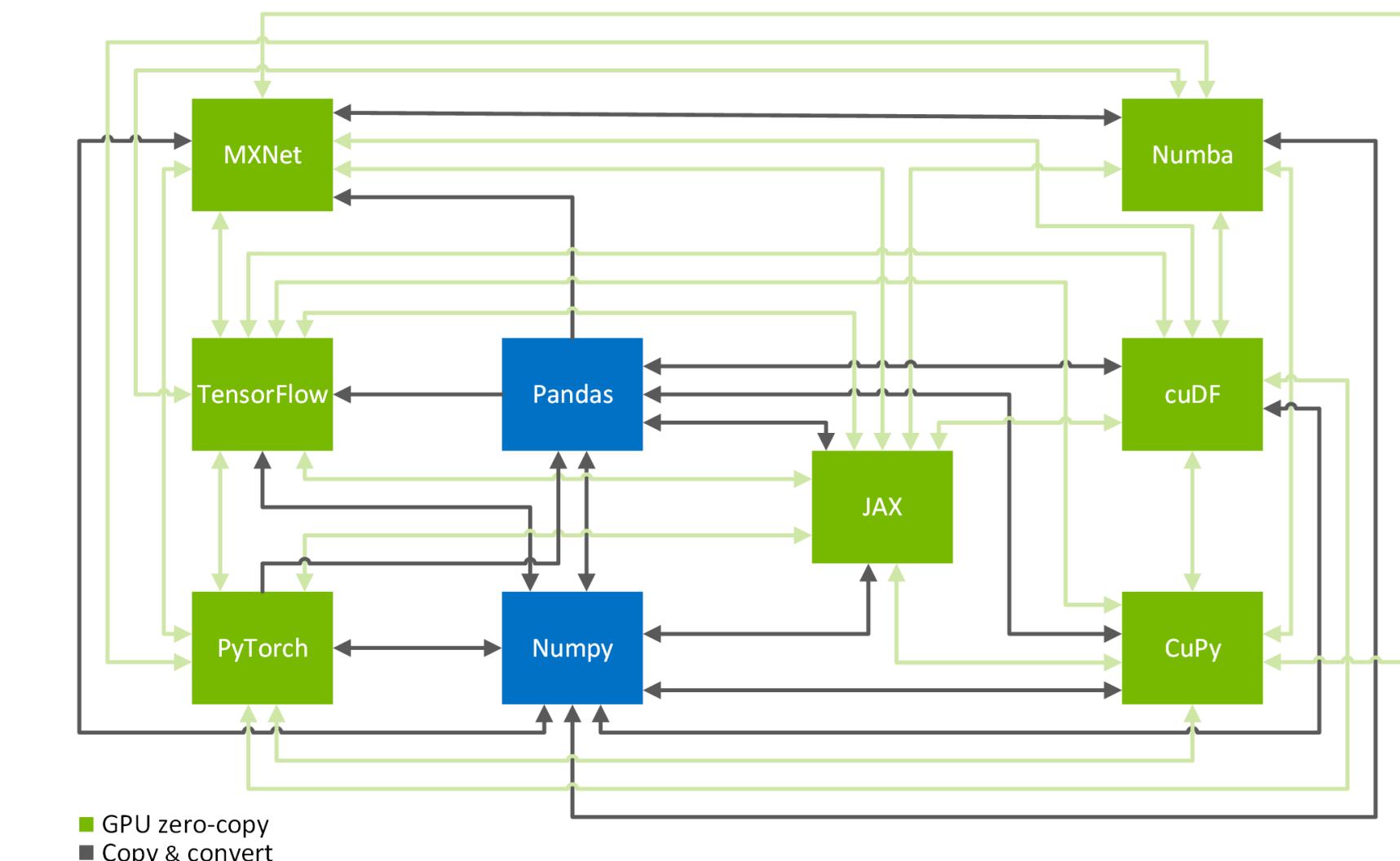
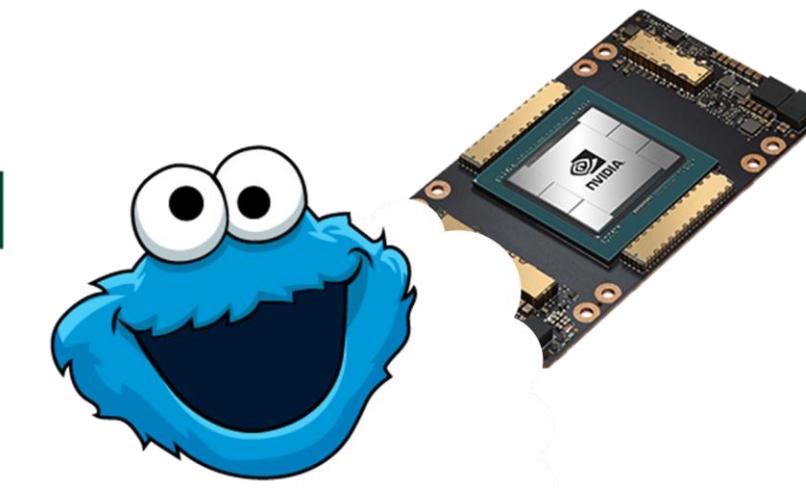
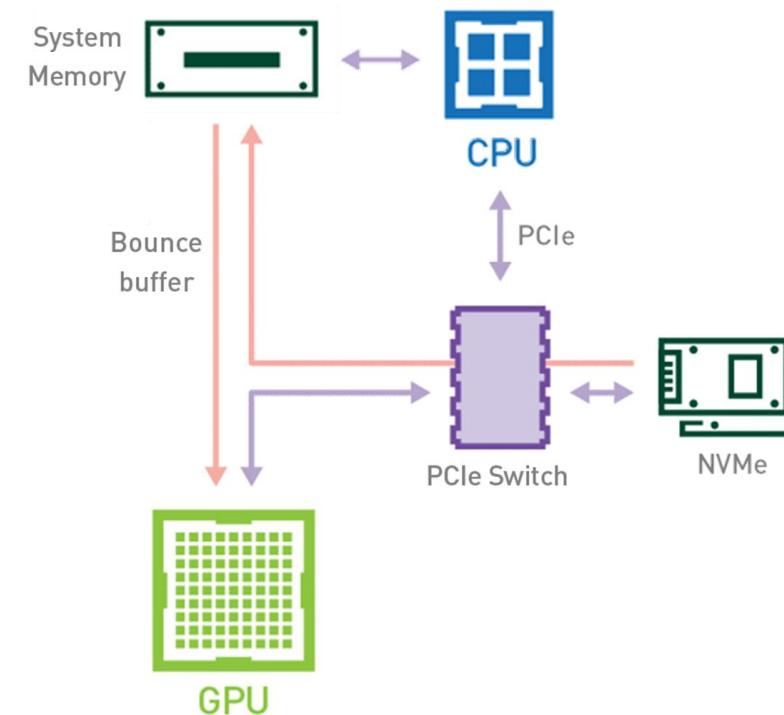
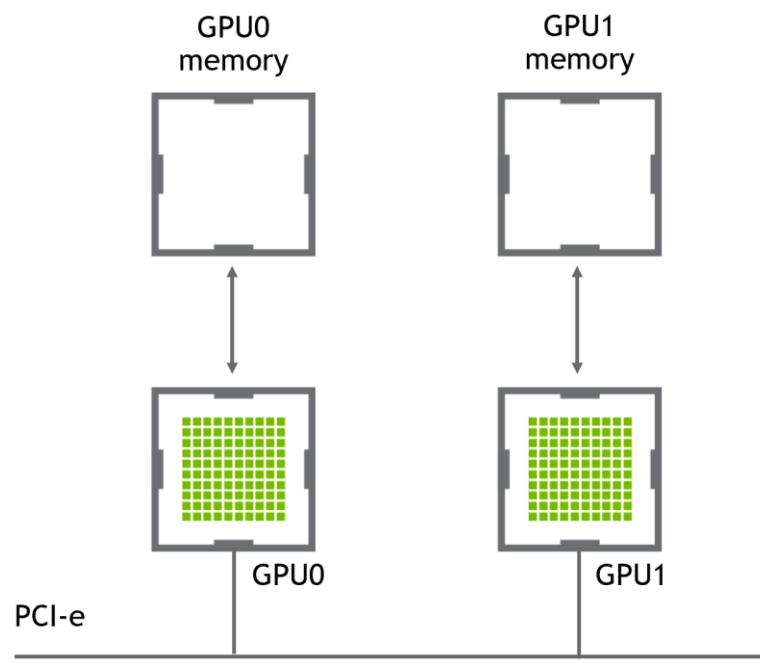
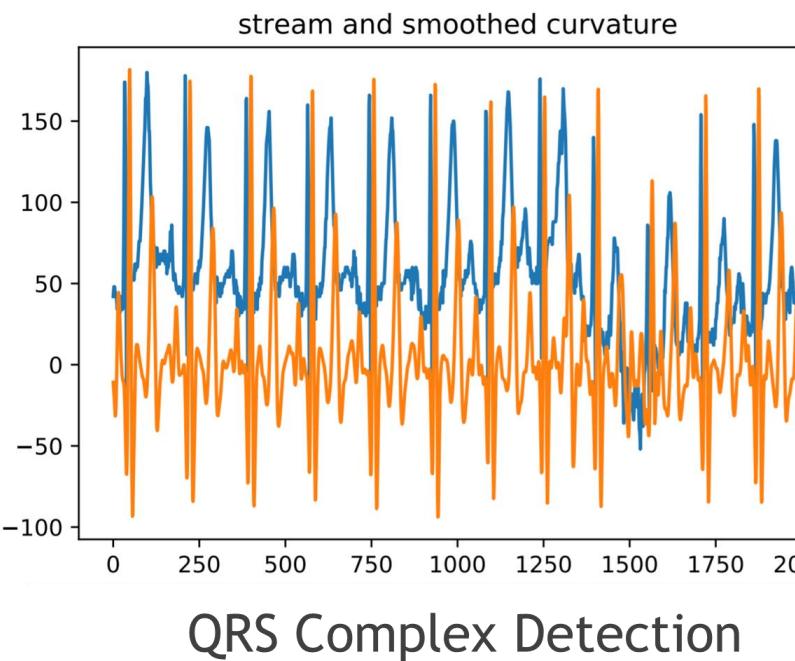
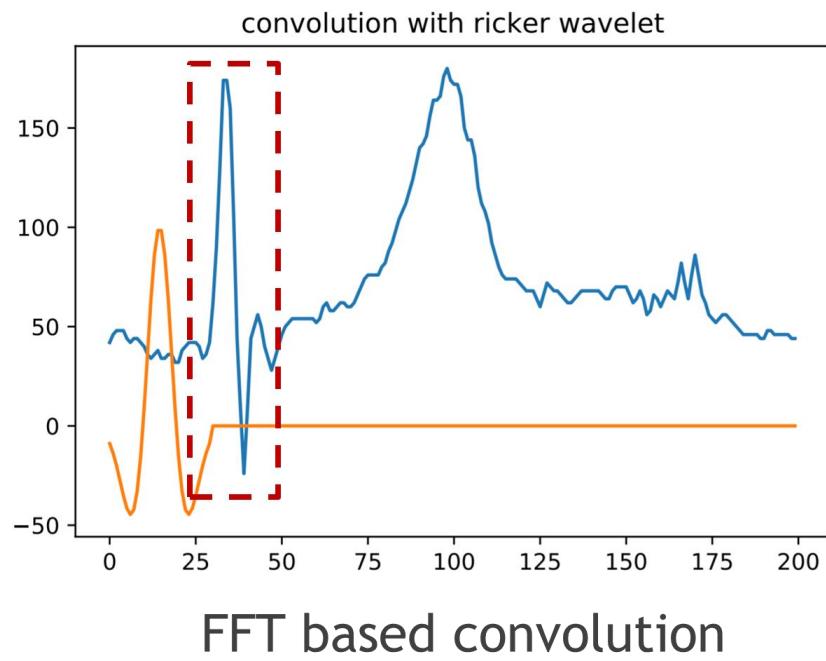
data  
engineer



**SPOILER ALERT**

# OUTLOOK

Things we will talk about



	user_id	timestamp	source_ip
row_1	6939800	2021-04-12 05:31	42.155.123.142
row_2	4015666	2021-04-12 05:47	67.132.212.125
row_3	5456236	2021-04-12 05:52	81.205.129.121

	Row-wise data layout	Columnar data layout
row_1	6939800	6939800
row_2	2021-04-12 05:31	4015666
row_3	42.155.123.142	5456236
user_id		2021-04-12 05:31
timestamp		2021-04-12 05:47
source_ip		2021-04-12 05:52

# E2E CODE EXAMPLE + CHEAT SHEET

**E2E Workflow.ipynb**

```
[1]: import cudf
subject_cudf = cudf.read_csv("data/subject.csv", dtype='float32')

[2]: import cufilter as cx
from cufilter.charts.datashader import line

# Chart width and height
WIDTH=600
HEIGHT=300

# 20 hours ECG Look Like a mess
subject_cudf['x'] = subject_cudf.index

line_cux = line(x='x', y='data', add_interaction=False)

_=cux.DataFrame.from_dataframe(subject_cudf).dashboard([line_cux])
line_cux.chart.title.text = 'ECG stream'
line_cux.chart.title.align = 'center'
line_cux.chart.width = WIDTH
line_cux.chart.height = HEIGHT
line_cux.view()[0]
```

**ECG stream**

```
[3]: import cupy as cp
from time_series_utils import smoothed_curvature_cupy

# convert to cupy
subject_cupy = cp.fromDlpack(subject_cudf['data'].to_dlpack())

# compute smoothed curvature via ricker wavelet
curvature_cupy = smoothed_curvature_cupy(subject_cupy, window=30)
```

**E2E Workflow.ipynb**

```
[15]: from cufilter.charts import scatter
scatter_chart_cux = scatter(x='x', y='y', pixel_shade_type="linear",
                            legend=False, add_interaction=False)
_=cux.DataFrame.from_dataframe(mu_cudf).dashboard([scatter_chart_cux])
scatter_chart_cux.chart.title.text = 'Latent space'
scatter_chart_cux.chart.title.align = 'center'
scatter_chart_cux.chart.width = WIDTH//2
scatter_chart_cux.chart.height = HEIGHT
scatter_chart_cux.view()[0]
```

**Latent space**

```
[16]: print("Installing hvplot...")
!pip -q install hvplot
Installing hvplot...
```

```
[17]: import hvplot.cudf
import panel

ROWS = 4
COLS = 4

# Let's sample some fake heart beats
with torch.no_grad():
    latents = torch.empty(ROWS * COLS, 2).normal_(mean=0.0, std=0.5).to('cuda')
    samples = model.decode(latents)

heartbeats_cudf = cudf.DataFrame({'x': cp.arange(samples.shape[1])})
heartbeats_gridspec = panel.GridSpec(width=COLS*200, height=ROWS*100)
for i, (sample, latent) in enumerate(zip(samples, latents)):
    heartbeats_cudf['y_%s' % i] = sample
```

**E2E Workflow.ipynb**

## Index

	Pandas	Numpy	cuDF	cuPY	JAX	Numba	TensorFlow	PyTorch	MXNet
Pandas	n/a	code	code	code	code	code	code	code	code
Numpy	code	n/a	code	code	code	code	code	code	code
cuDF	code	code	n/a	code	code	code	code	code	code
cuPY	code	code	code	n/a	code	code	code	code	code
JAX	code	code	code	code	n/a	code	code	code	code
Numba	code	code	code	code	code	n/a	code	code	code
TensorFlow	code	code	code	code	code	code	n/a	code	code
PyTorch	code	code	code	code	code	code	code	n/a	code
MXNet	code	code	code	code	code	code	code	code	n/a

## From Pandas to Numpy

```
[25]: # Option 1: Pandas DataFrame to a Numpy ndarray
src = pd.DataFrame({'x': [1, 2], 'y': [3, 4]})

print(type(dst), "\n", dst)
<class 'numpy.ndarray'>
[[1 3]
 [2 4]]
```

```
[26]: # Option 2: Convert a Pandas DataFrame to a Numpy ndarray
src = pd.DataFrame({'x': [1, 2], 'y': [3, 4]})

dst = src.values # "to_numpy()" is preferred to "values".

print(type(dst), "\n", dst)
<class 'numpy.ndarray'>
[[1 3]
 [2 4]]
```

```
[27]: # Option 3: Convert a Pandas DataFrame to a Numpy recarray
src = pd.DataFrame({'x': [1, 2], 'y': [3, 4]}, index=['a', 'b'])

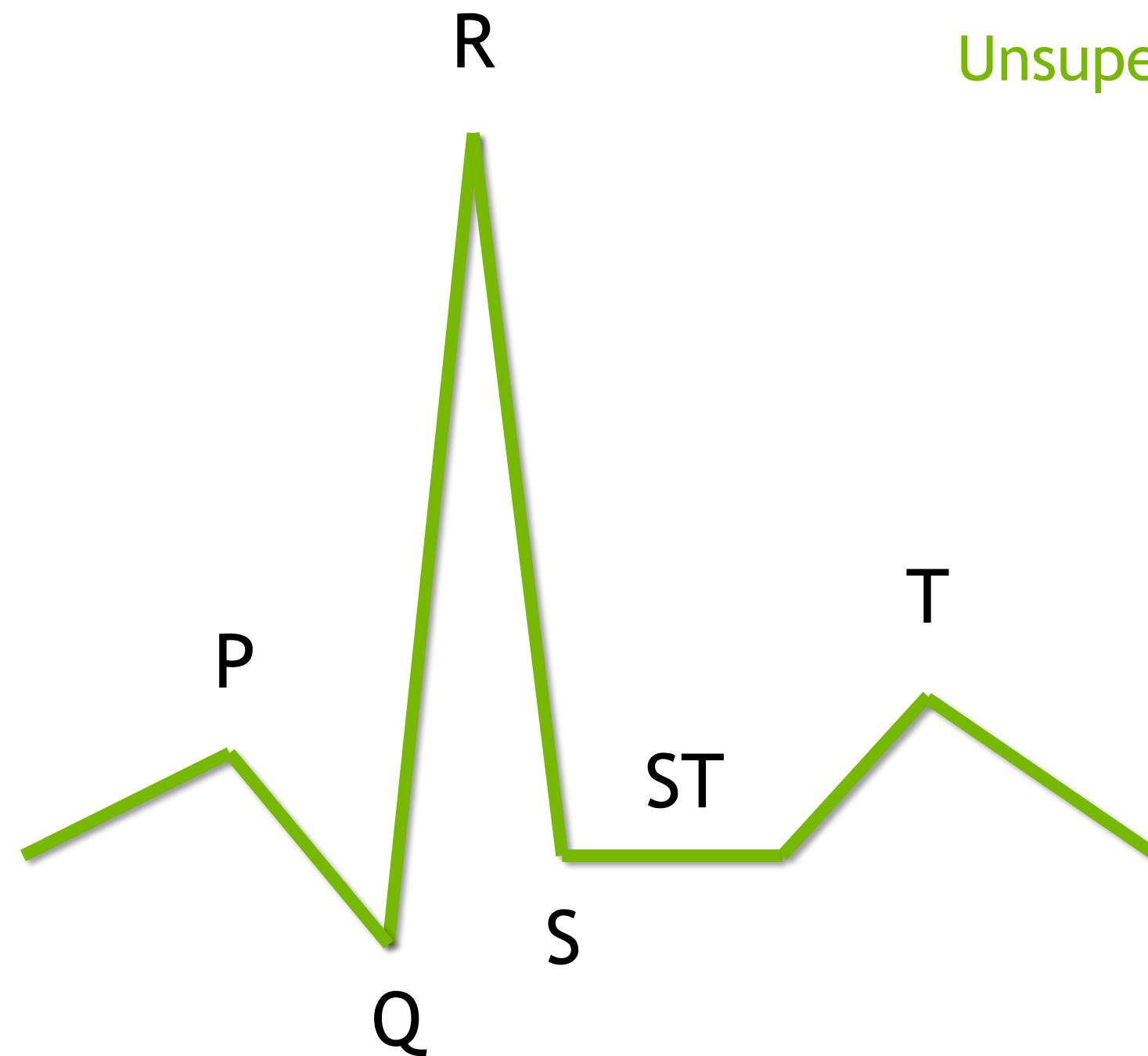
print(type(dst), "\n", dst)
<class 'numpy.recarray'>
[['a', 1, 3] ('b', 2, 4)]
```



E2E PIPELINE

# TOY MODEL

Unsupervised outlier detection



What we have:

- 20 hours stream of continuously measured electrocardiogram (ECG) data.
- Univariate and uniformly sampled time series as CSV on disk.

What we are doing:

- Unsupervised segmentation of ECG stream into ~ 100k heartbeats.
- Training of Variational Autoencoder (VAE) for outlier detection.
- Visualization of generated heartbeats

## *Disclaimer*

*Technical example pipeline demonstrating framework interoperability.  
Not suitable for production in medical environments.*

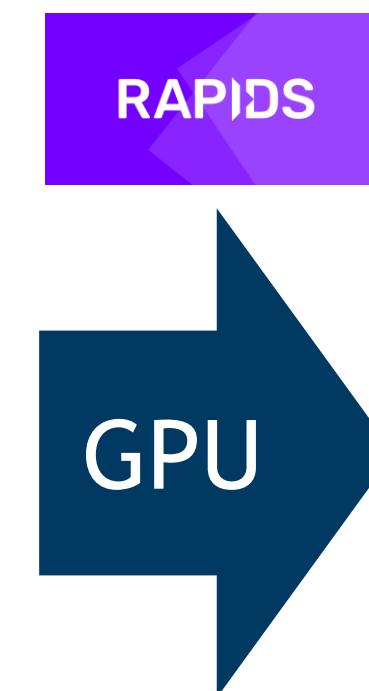
# END-TO-END PIPELINE

Parse ECG from CSV



```
1 data
2 1.3863000000000008e+00
3 1.734900000000000109e+00
4 2.186500000000000110e+00
5 2.7471999999999864e+00
6 3.75190000000000013e+00
7 5.0231000000000342e+00
8 6.4785000000000369e+00
9 8.0025999999999270e+00
10 9.4921000000000648e+00
11 1.08254000000000013e+01
12 1.1867699999999925e+01
13 1.24868000000000057e+01
14 1.25873000000000082e+01
15 1.2142899999999914e+01
```

```
cudf.io.csv.read_csv(filepath_or_buffer,
lineterminator='\n', quotechar='', quoting=0,
doublequote=True, header='infer',
mangle_dupe_cols=True, usecols=None,
sep=',', delimiter=None,
delim_whitespace=False,
skipinitialspace=False, names=None,
dtype=None, skipfooter=0, skiprows=0,
dayfirst=False, compression='infer',
thousands=None, decimal=',',
true_values=None, false_values=None,
nrows=None, byte_range=None,
skip_blank_lines=True, parse_dates=None,
comment=None, na_values=None,
keep_default_na=True, na_filter=True,
prefix=None, index_col=None, **kwargs)
```



	data
0	1.3863
1	1.7349
2	2.1865
3	2.7472
4	3.7519
...	...

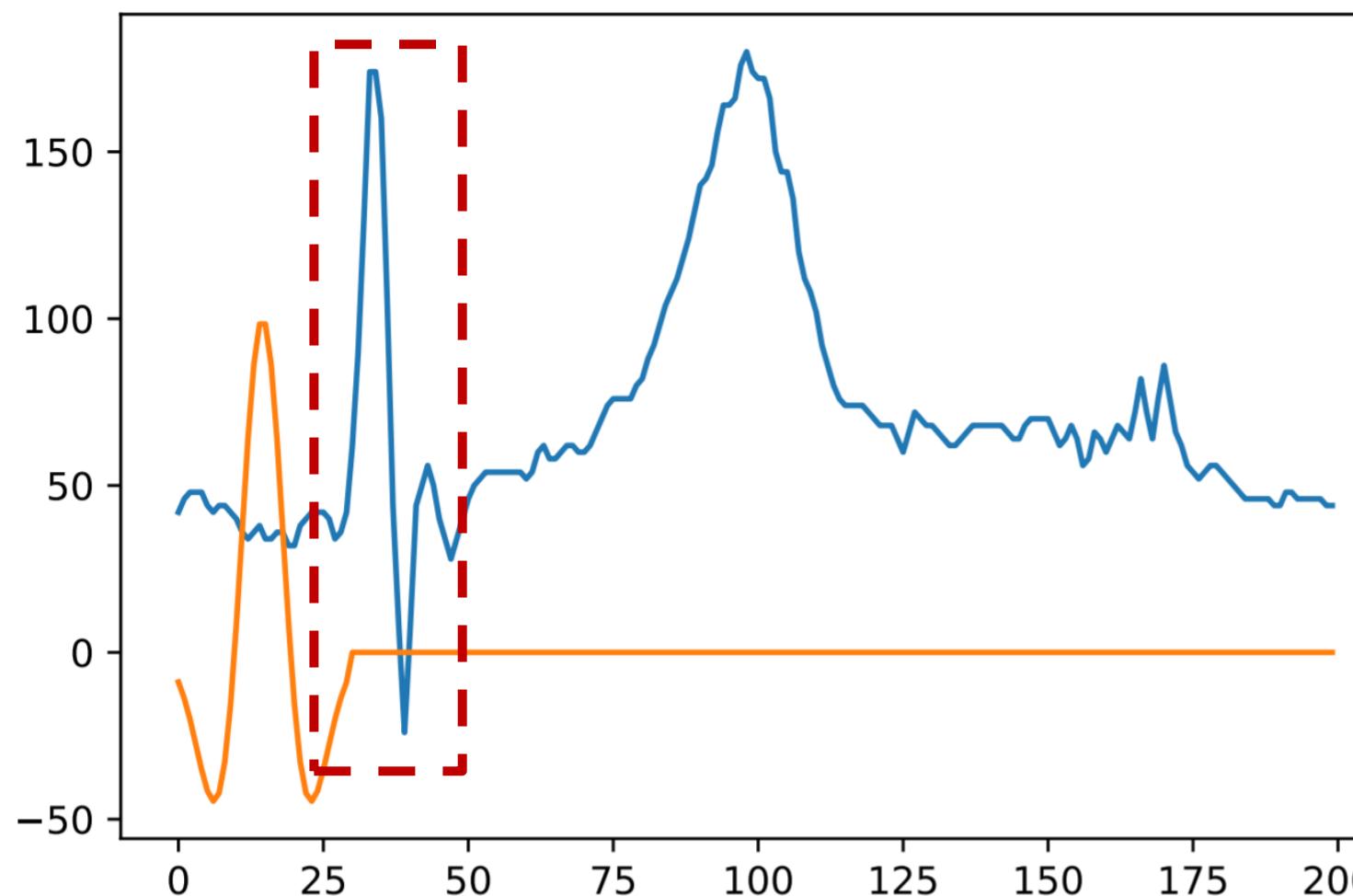
Load a comma-separated-values (CSV) dataset into a DataFrame

# END-TO-END PIPELINE

Band-Pass Filter

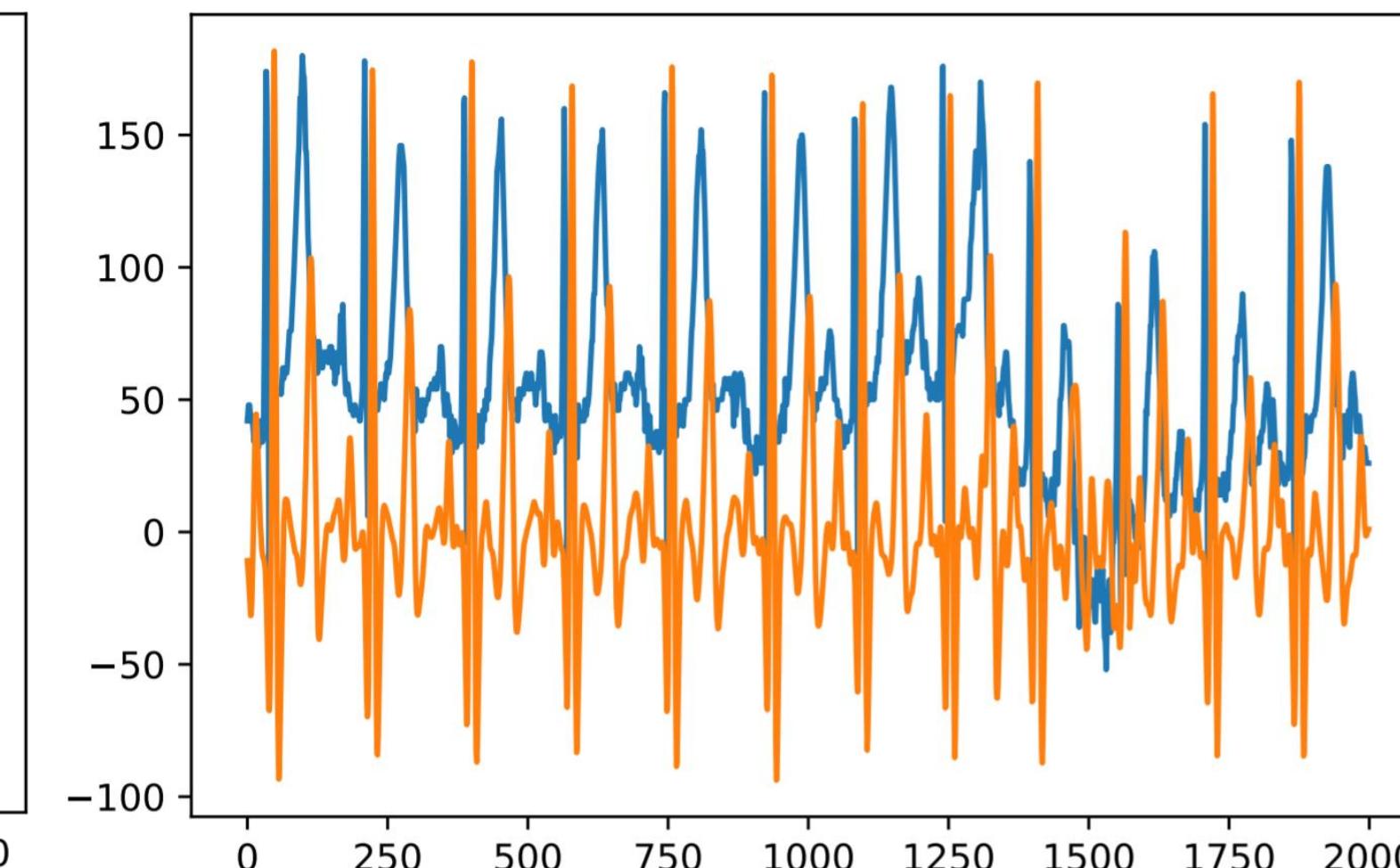


convolution with ricker wavelet



FFT based convolution

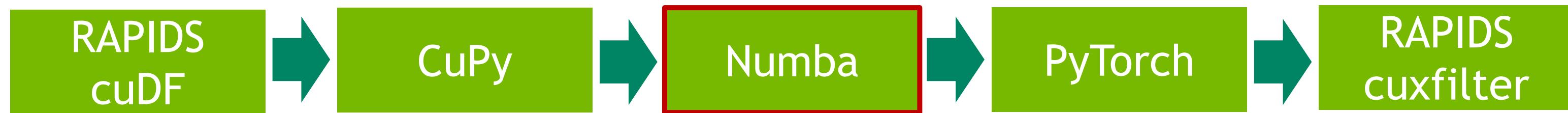
stream and smoothed curvature



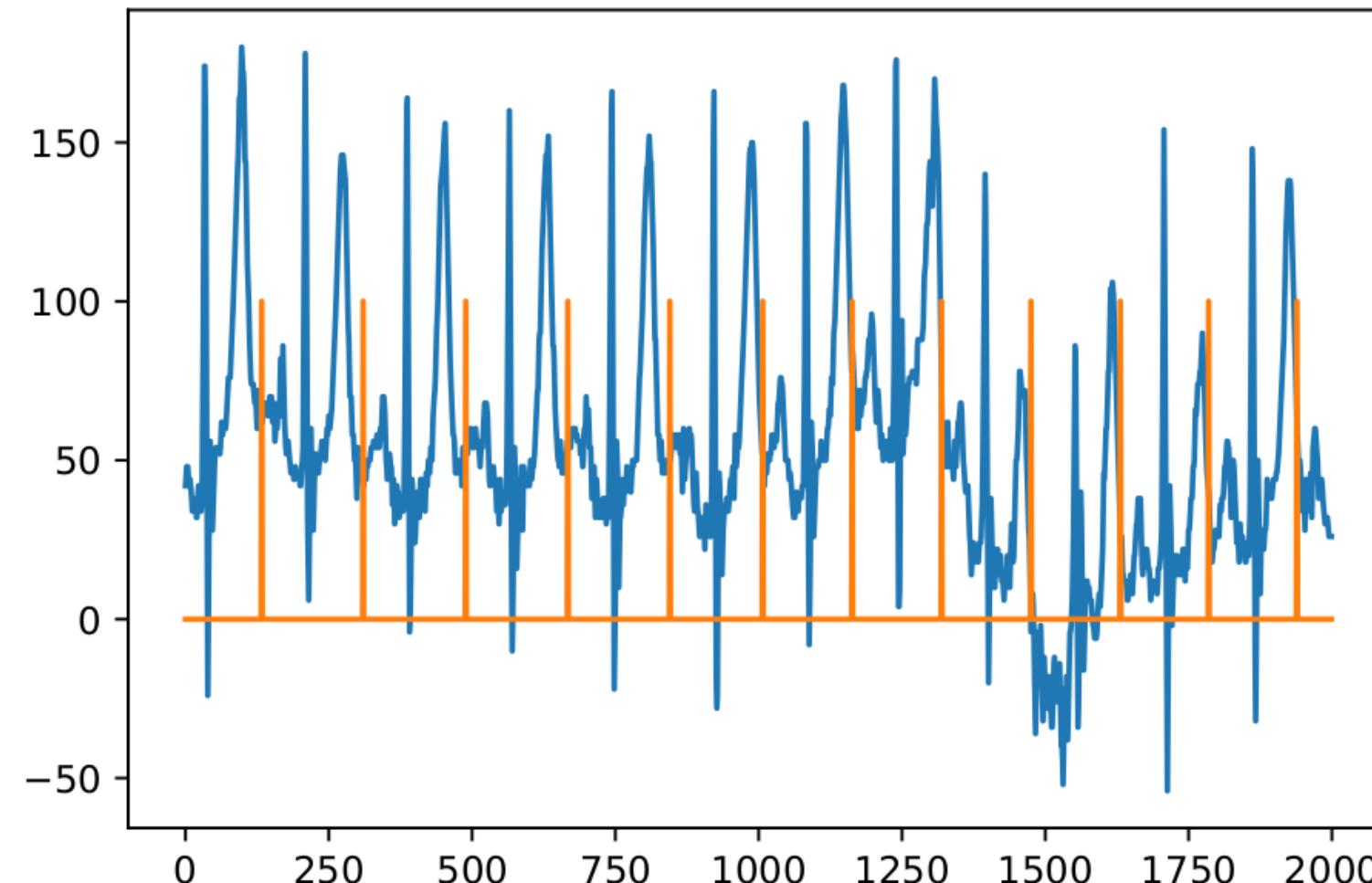
QRS Complex Detection

# END-TO-END PIPELINE

Non-trivial Preprocessing

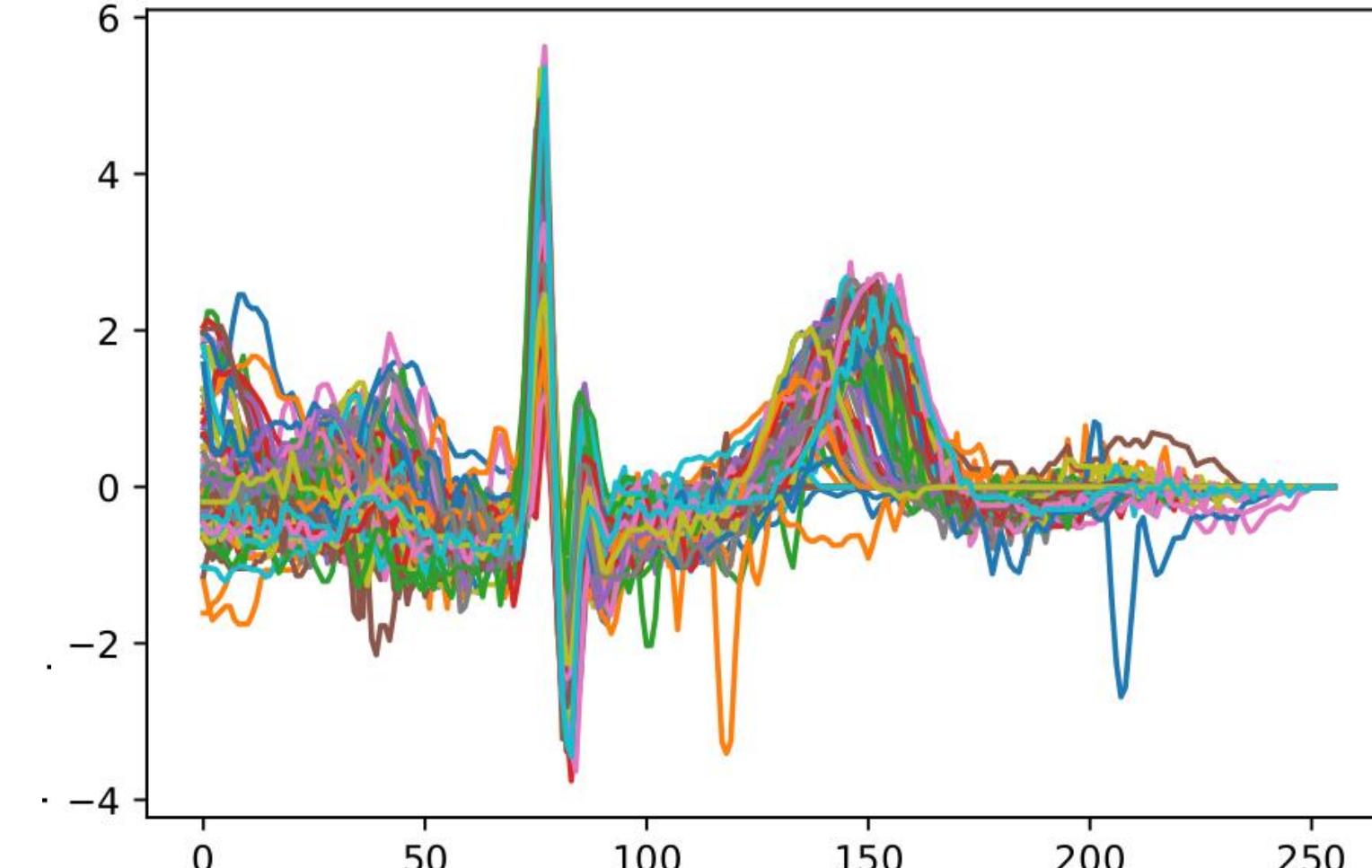


segmentation gates



1D non-max suppression

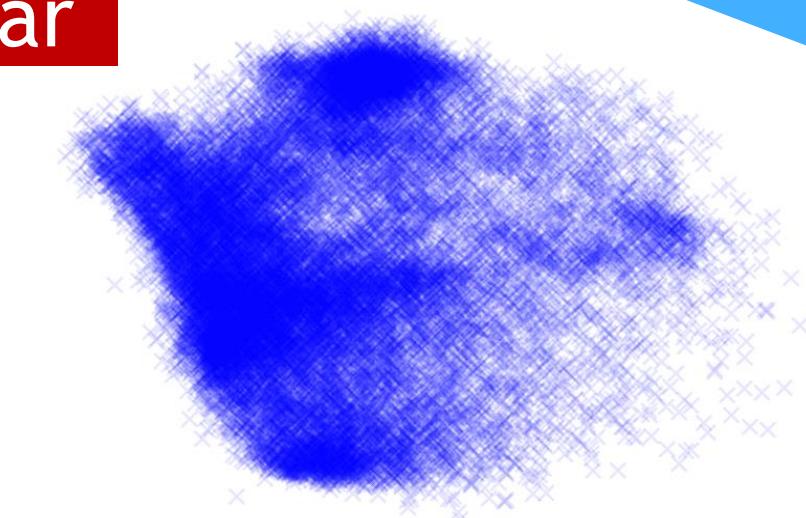
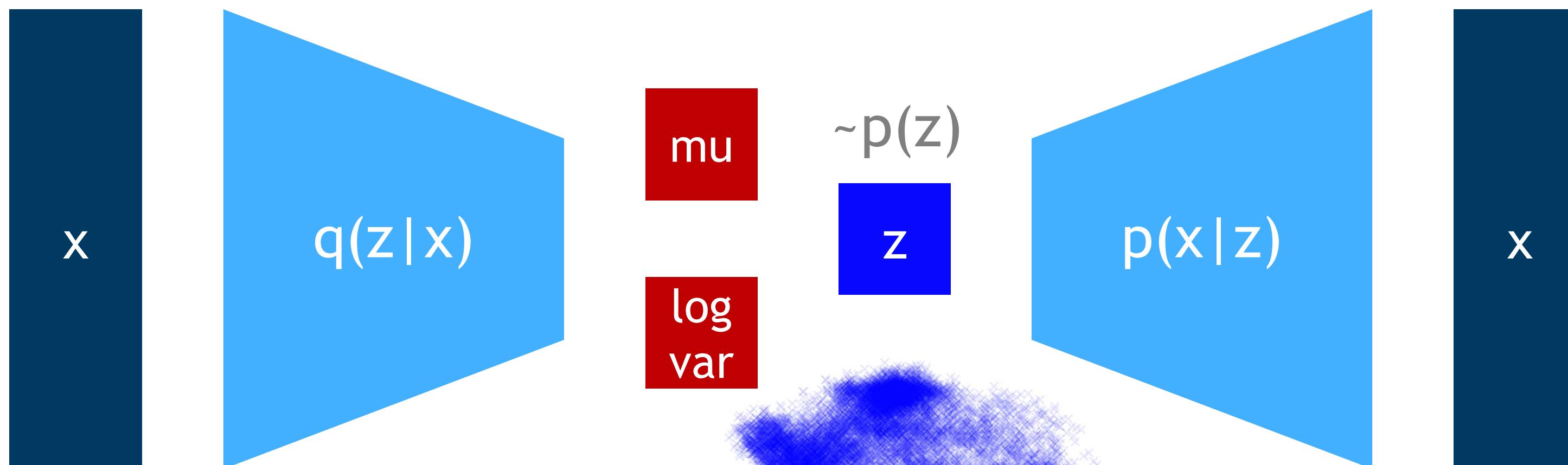
a few heartbeats



normalization and embedding

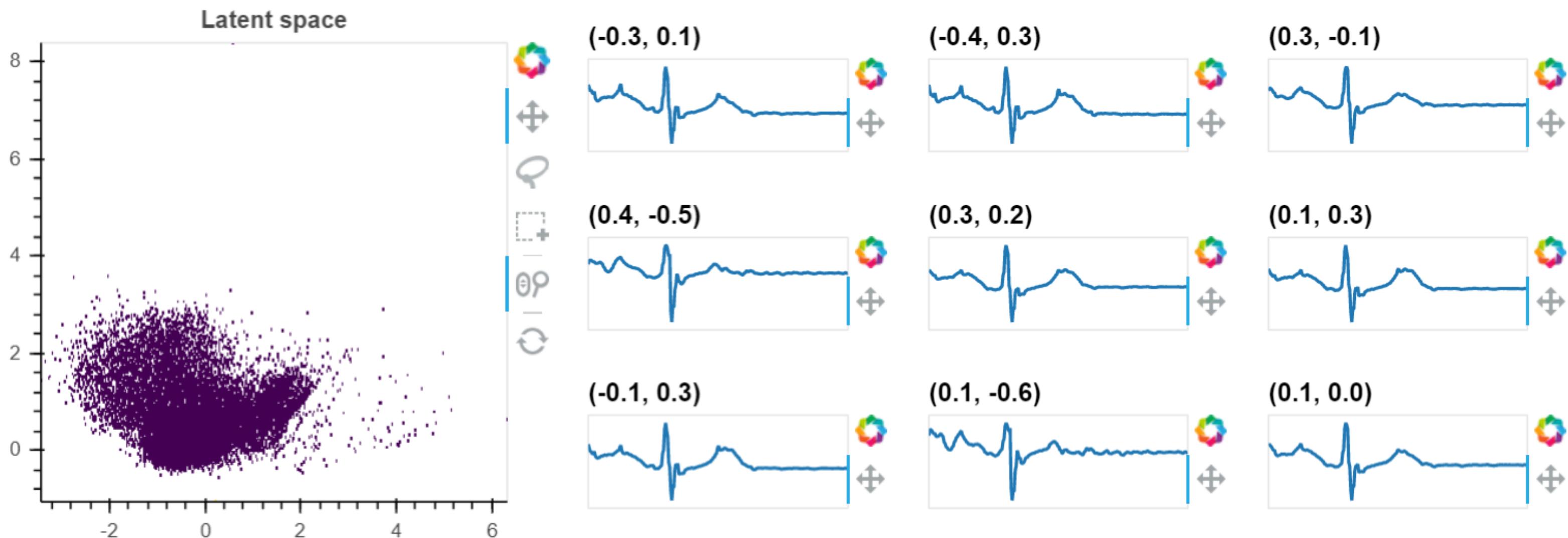
# END-TO-END PIPELINE

Variational Autoencoder (VAE)



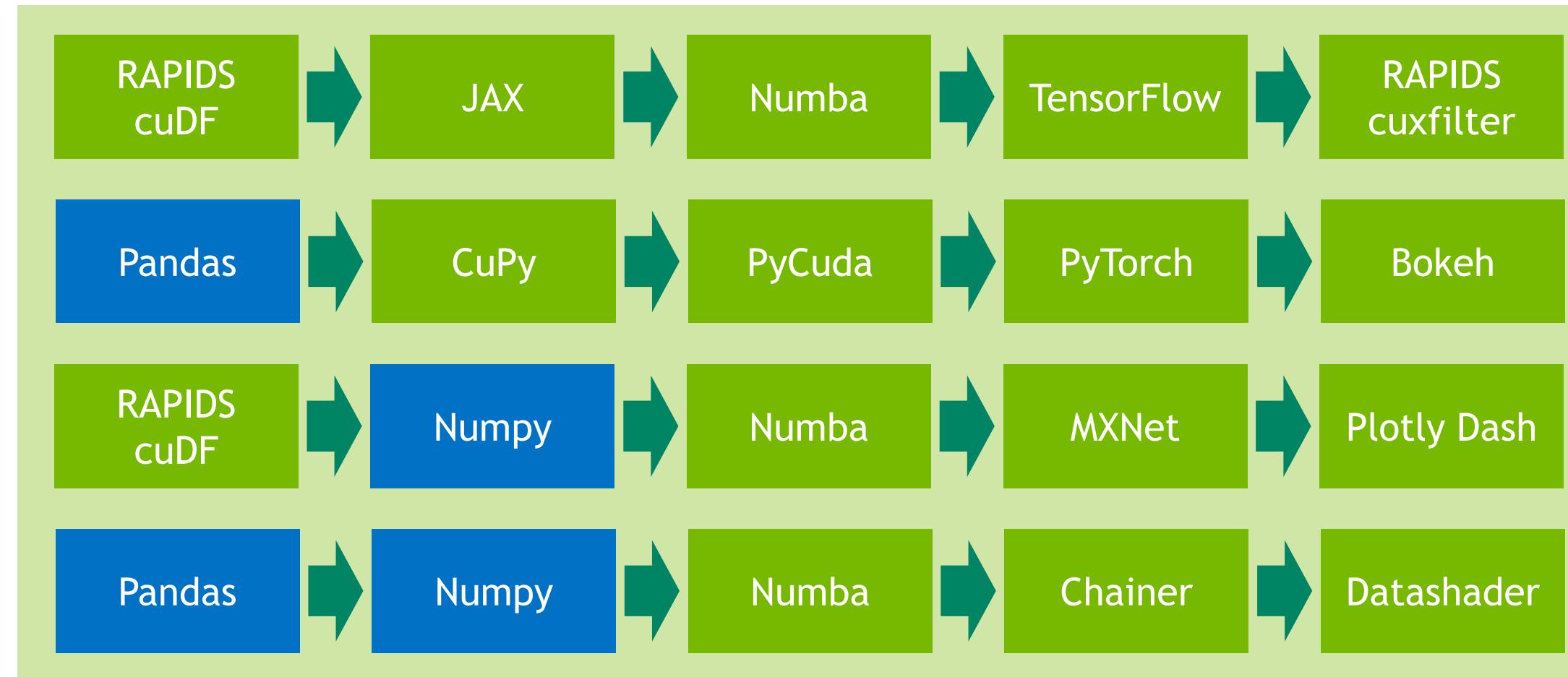
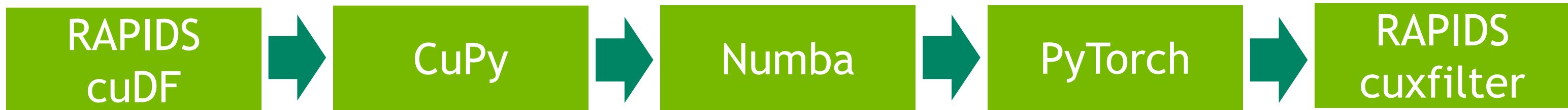
# END-TO-END PIPELINE

Visualization



# MIX AND MATCH WORKFLOWS

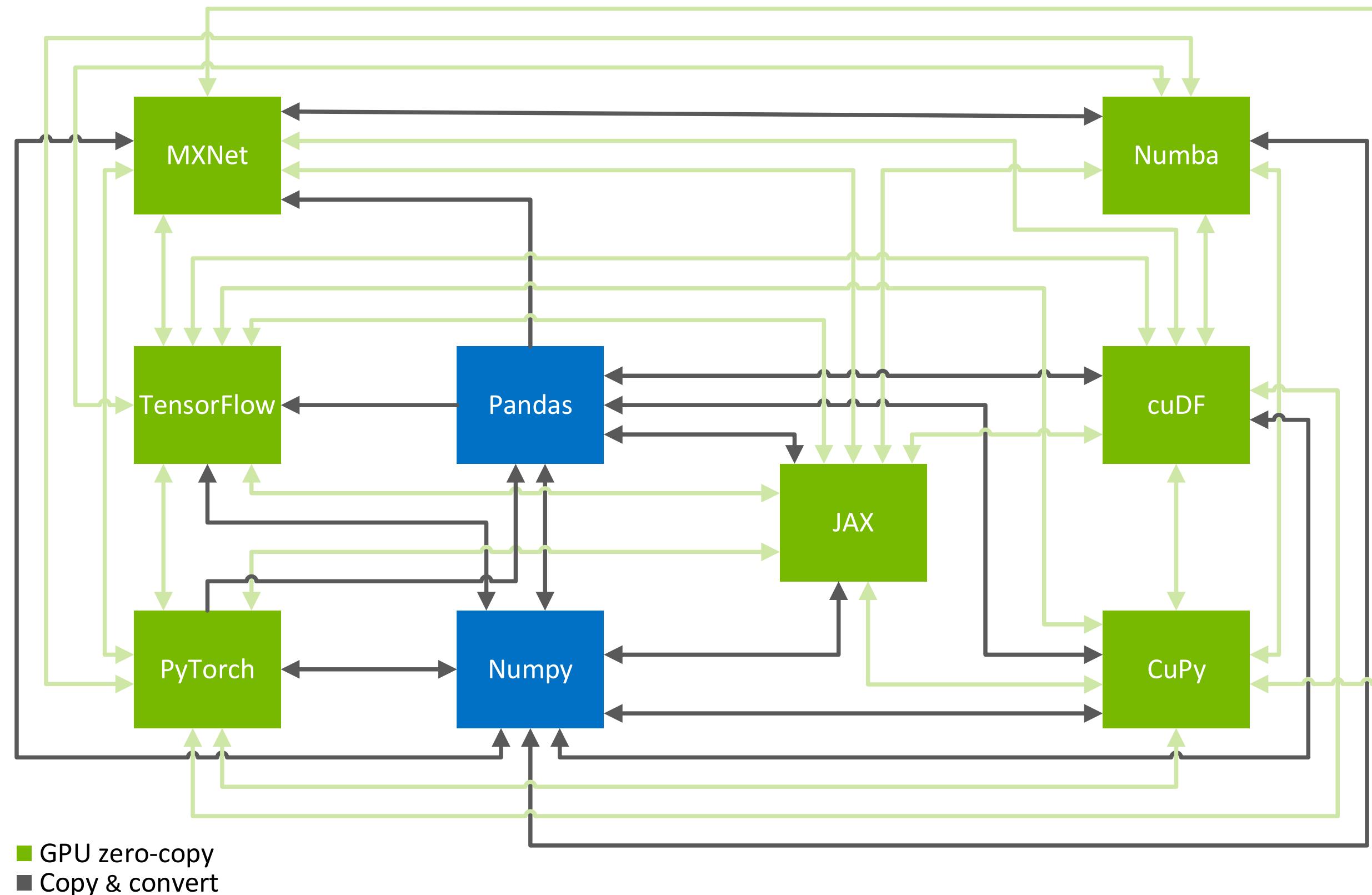
Endless possibilities!



And many others!

# MIX AND MATCH WORKFLOWS

Use the right tool, for the right job, in the right way

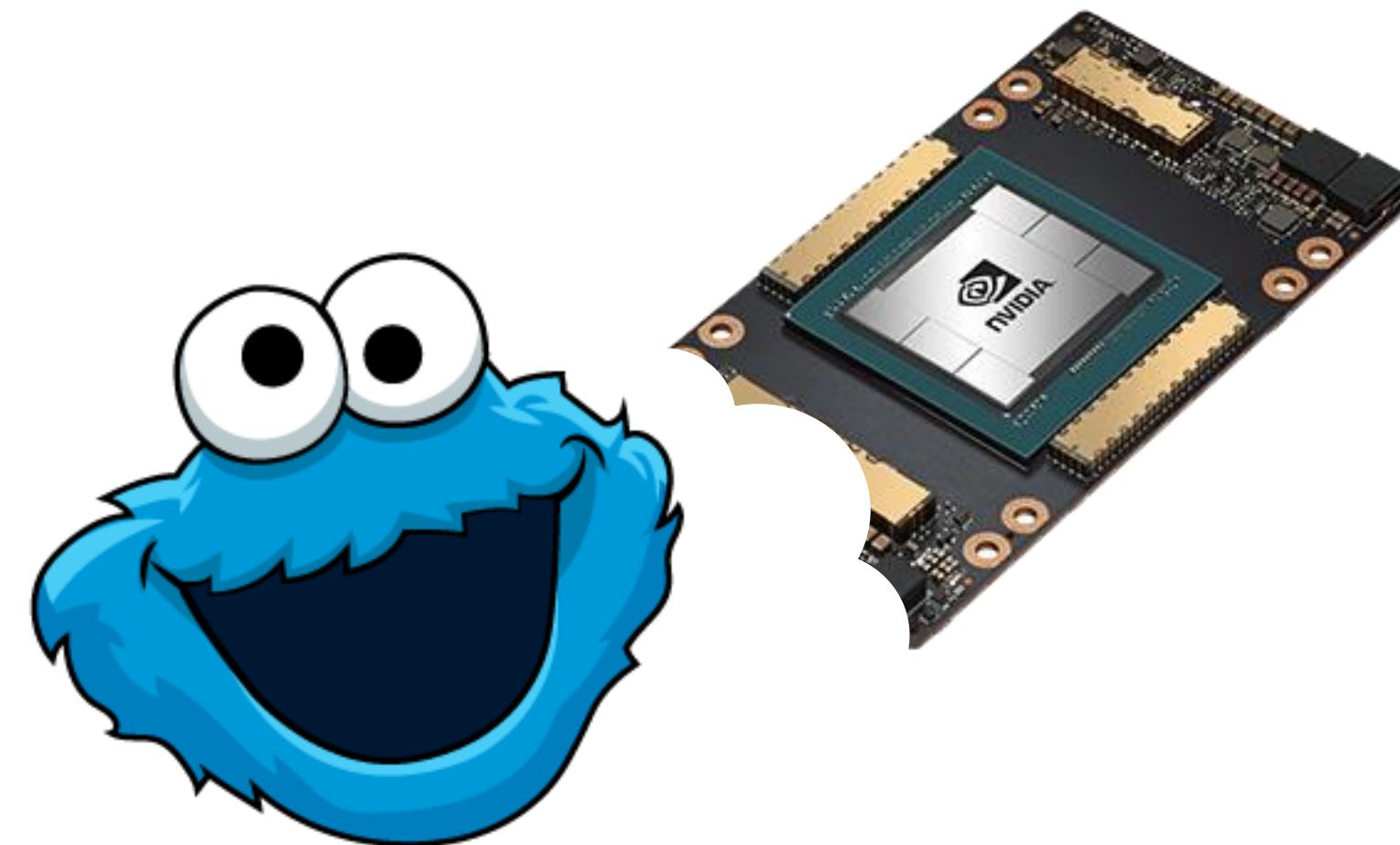




# MEMORY MANAGEMENT BOTTLENECK

# MEMORY MANAGEMENT BOTTLENECK

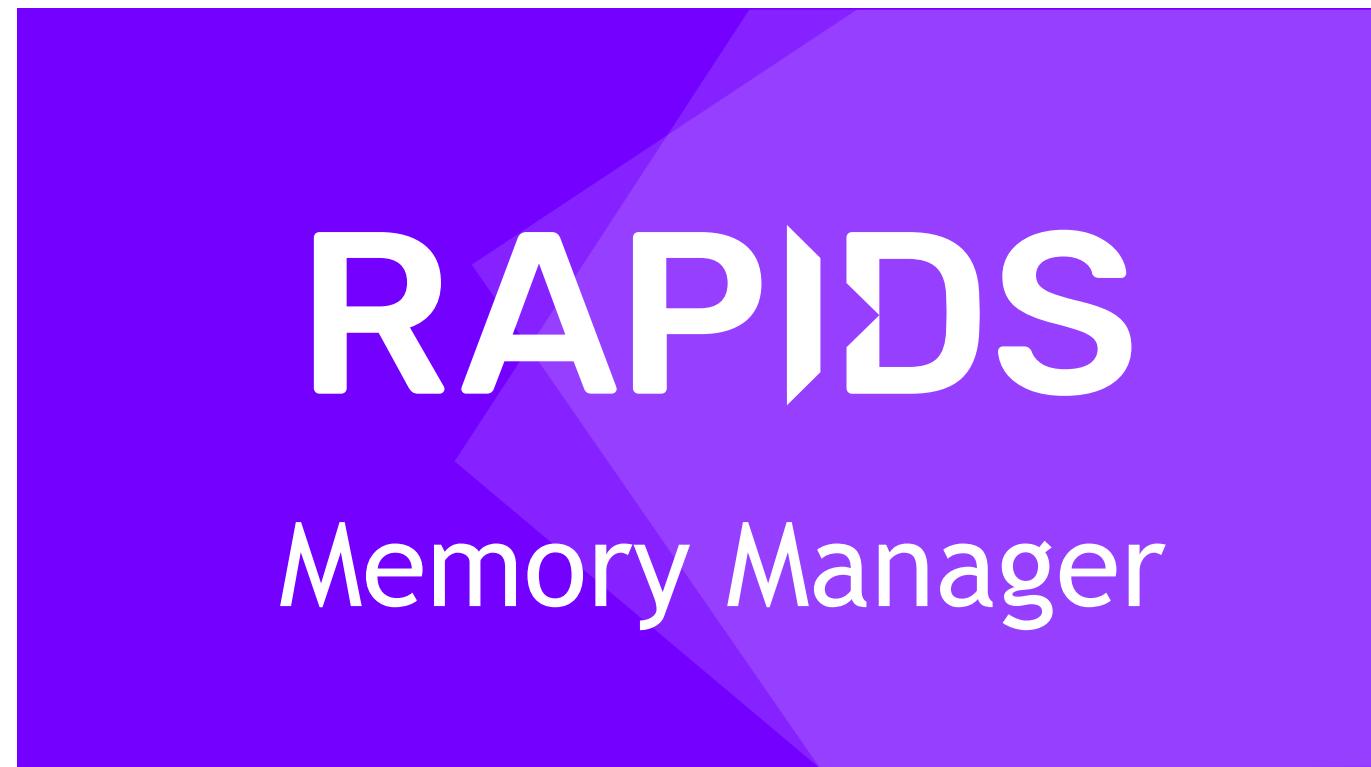
Memory managers are greedy



Gimme some more... memory!

# MEMORY MANAGEMENT ~~BOTTLENECK~~

One memory manager to rule them all?



CuPy



- Open-source external allocator interface (EAI)

```
void* allocate(std::size_t bytes, cudaStream_t stream)
void deallocate(void* p, std::size_t bytes, cudaStream_t stream)
```
- Follows open-closed principle:

*software entities should be **open** for extension  
but **closed** for modification.*
- Makes it trivial to layer additional functionality like logging, leak checking, limiting, etc.
- Zero-code integration with RAPIDS:

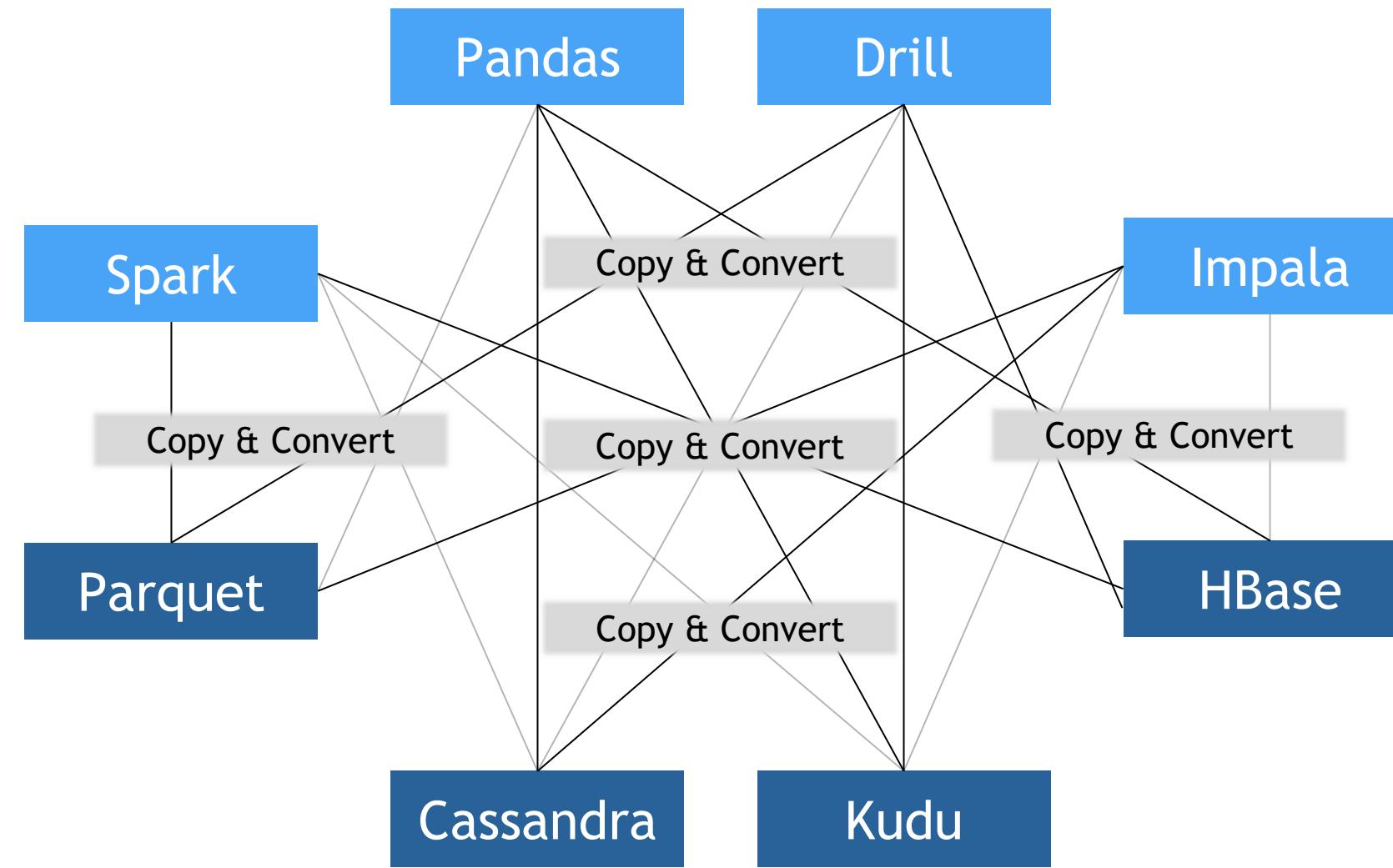
```
import cudf before using CuPy or Numba.
```



# DATA CONVERSION BOTTLENECK

# DATA CONVERSION BOTTLENECK

Copy and convert is expensive



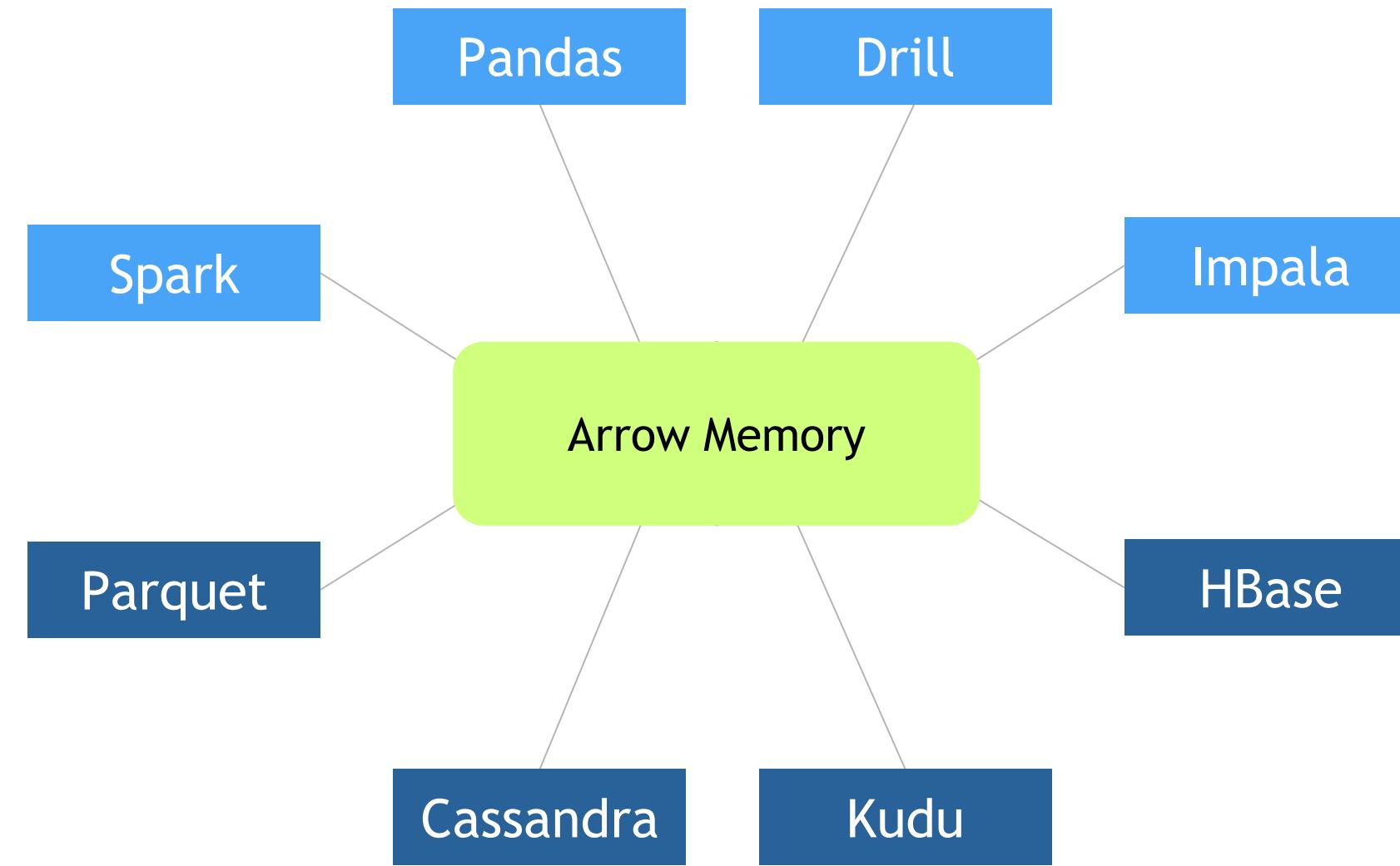
Each system has its own internal memory format.

Similar functionality implemented in multiple projects.

~70% computation wasted on serialization & deserialization.

# DATA CONVERSION ~~BOTTLENECK~~

Learning from Apache Arrow



All systems utilize the same memory format.

Projects can share functionality.

No overhead for cross-system communication.

# COLUMNAR DATA LAYOUT

A perfect match for data science

Columnar layout leverages GPU strengths.

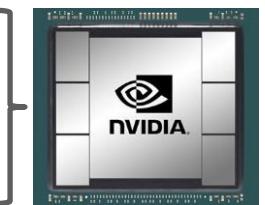
CPU Single Instruction Multiple Data (SIMD) instructions also take advantage of it.

Consistency with CPU version simplifies development and conversion.

Emphasis on zero-copy and shallow-copy operations minimizes a core bottleneck.

	user_id	timestamp	source_ip
row_1	6939800	2021-04-12 05:31	42.155.123.142
row_2	4015666	2021-04-12 05:47	67.132.212.125
row_3	5456236	2021-04-12 05:52	81.205.129.121

	Row-wise data layout	Columnar data layout
row_1	6939800	6939800
row_2	2021-04-12 05:31	4015666
row_3	42.155.123.142	5456236
row_1	4015666	2021-04-12 05:31
row_2	2021-04-12 05:47	2021-04-12 05:47
row_3	81.205.129.121	2021-04-12 05:52
row_1	5456236	42.155.123.142
row_2	2021-04-12 05:52	67.132.212.125
row_3	81.205.129.121	81.205.129.121



# DLPACK

Sharing tensors the easiest way

DLPack is an open in-memory tensor structure which enables:

- Easier sharing of tensors and operators between deep learning frameworks.
- Easier wrapping of vendor level operator implementations, allowing collaboration when introducing new devices/ops.
- Quick swapping of backend implementations, like different version of BLAS.
- For final users, this could bring more operators, and possibility of mixing usage between frameworks.

## From cuDF to CuPy

```
# Convert a cuDF DataFrame to a CuPy ndarray
src = cudf.DataFrame({'x': [1, 2], 'y': [3, 4]})
dst = cp.from_dlpack(src.to_dlpack())

print(type(dst), "\n", dst)

<class 'cupy.core.core.ndarray'>
[[1 3]
 [2 4]]
```

## From CuPy to PyTorch

```
# Convert a CuPy ndarray to a PyTorch Tensor
src = cp.array([[1, 2], [3, 4]])
dst = torch.from_dlpack(src.toDlpack())

print(type(dst), "\n", dst)

<class 'torch.Tensor'>
tensor([[1, 2],
        [3, 4]], device='cuda:0')
```

# CUDA ARRAY INTERFACE 3.0

## Seamless Ingestion

The `__cuda_array_interface__` attribute returns a dictionary (`dict`) that must contain the following entries:

**shape: (integer, ...)**

A tuple of int (or long) representing the size of each dimension.

**typestr: str**

The type string. This has the same definition as `typestr` in the numpy array interface.

**data: (integer, boolean)**

The data is a 2-tuple. The first element is the data pointer as a Python `int` (or `long`). The data must be device-accessible. For zero-size arrays, use 0 here. The second element is the read-only flag as a Python `bool`.

**version: integer**

An integer for the version of the interface being exported. The current version is 3.



# NUMBA INGESTION

## CuPy->Numba Example

```
from numba import cuda
import math

@cuda.jit
def segment_kernel(signal, window, out):

    base, step = cuda.grid(1), cuda.gridsize(1)
    for position in range(base, signal.shape[0]-window+1, step):
        accum = -math.inf
        for index in range(window):
            value = signal[position+index]
            accum = value if value > accum else accum
        # am I the maximum in my neighborhood?
        out[position] = 1 if accum == signal[position+(window+1)//2] else 0

def segment_numba(signal, window):

    out = cp.empty(signal.shape[0]-window+1, dtype=cp.int64)
    segment_kernel[80*32, 64](signal, window, out)
    cuda.synchronize()

    # fixes sequences of the following form 0000011000-> 0000010000
    out[1:] *= (cp.diff(out) == 1)

    return out
```

signal and out treated as being equivalent to  
numba.cuda.cudadrv.devicearray.DeviceNDArray



signal and out of type  
cupy.core.core.ndarray



## The Consortium for Python Data API Standards

Open consortium to create n-dimensional array and dataframe API standards.

Multidimensional arrays (tensors) fragmentation:

- NumPy, CuPy, Tensorflow, PyTorch, etc.

Dataframes fragmentation:

- Pandas, cuDF, Modin, Dask, Vaex, etc.

Cross-project and cross-ecosystem alignment on APIs, data exchange mechanisms and other related topics.

November 10<sup>th</sup>, 2020: Published the first version of its array API standard: <https://data-apis.org/array-api/latest>

### Consortium Members

Apache Arrow	GeoPandas	MXNet	PyTorch
Apache Spark	Ibis	NumPy	TensorFlow
cuDF	JAX	OmniSciDB	Vaex
CuPy	Koalas	ONNX	XArray
Dask	Modin	Pandas	...

HOW STANDARDS PROLIFERATE:  
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)

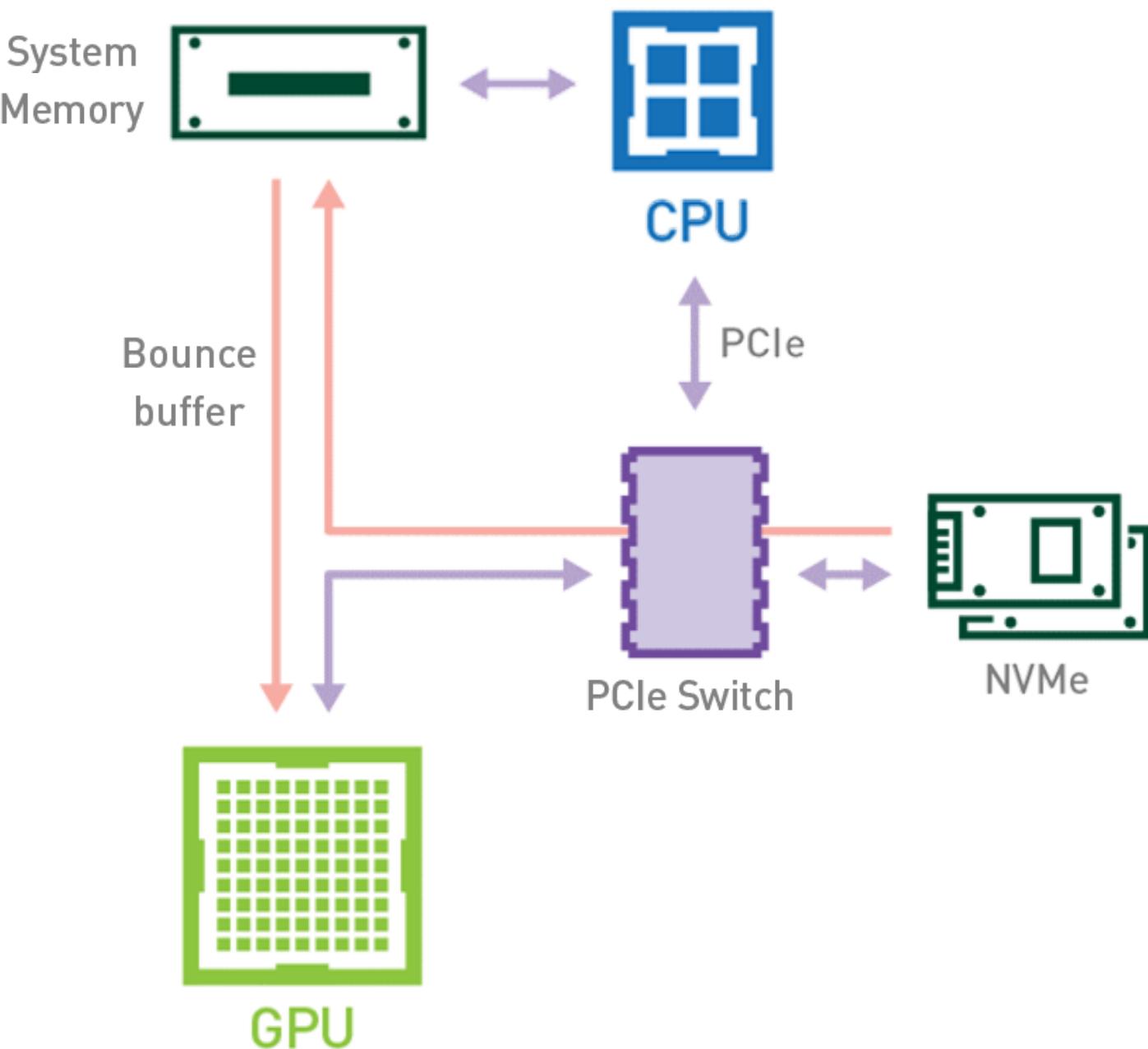




# DATA LOADING BOTTLENECK

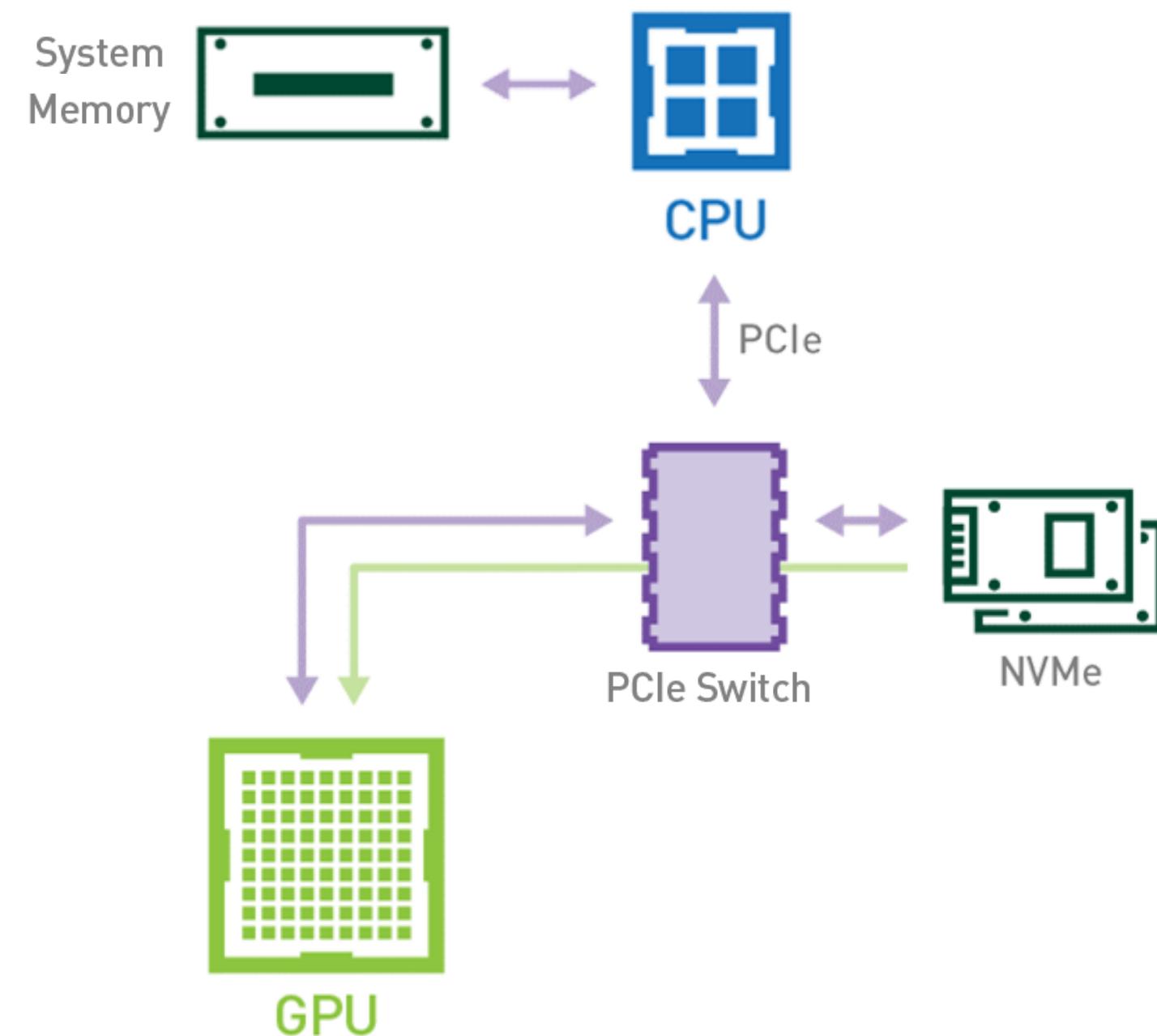
# DATA LOADING BOTTLENECK

## Without GPUDirect Storage



# DATA LOADING ~~BOTTLENECK~~

## With GPUDirect Storage

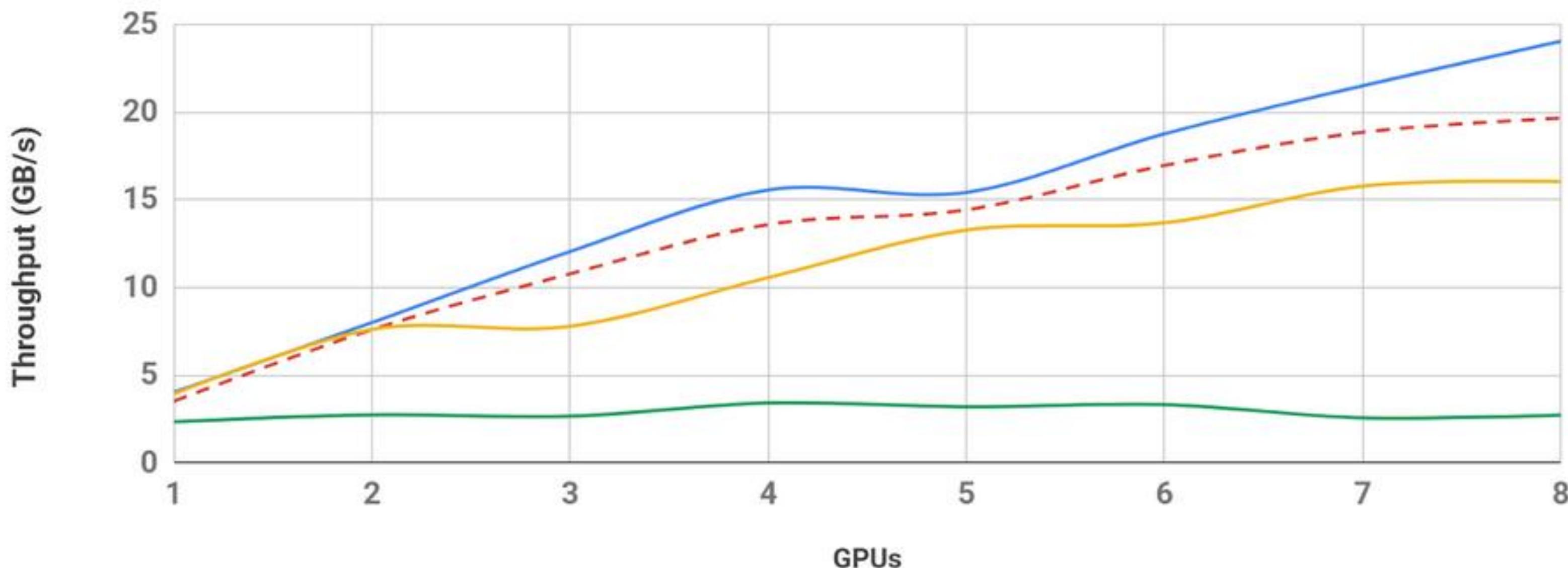


A Direct Path Between Storage and GPU Memory

# DATA LOADING ~~BOTTLENECK~~ RAPIDS cuIO

## GPUs vs Throughput

- GDS (NVMe ->GPU)
- mmap (Read from Active page cache) + cudaMemcpy + cudaMalloc Memory
- mmap (Read from Disk) + cudaMemcpy + cudaMalloc Memory
- mmap + cudaMallocManaged (Stock Bits)

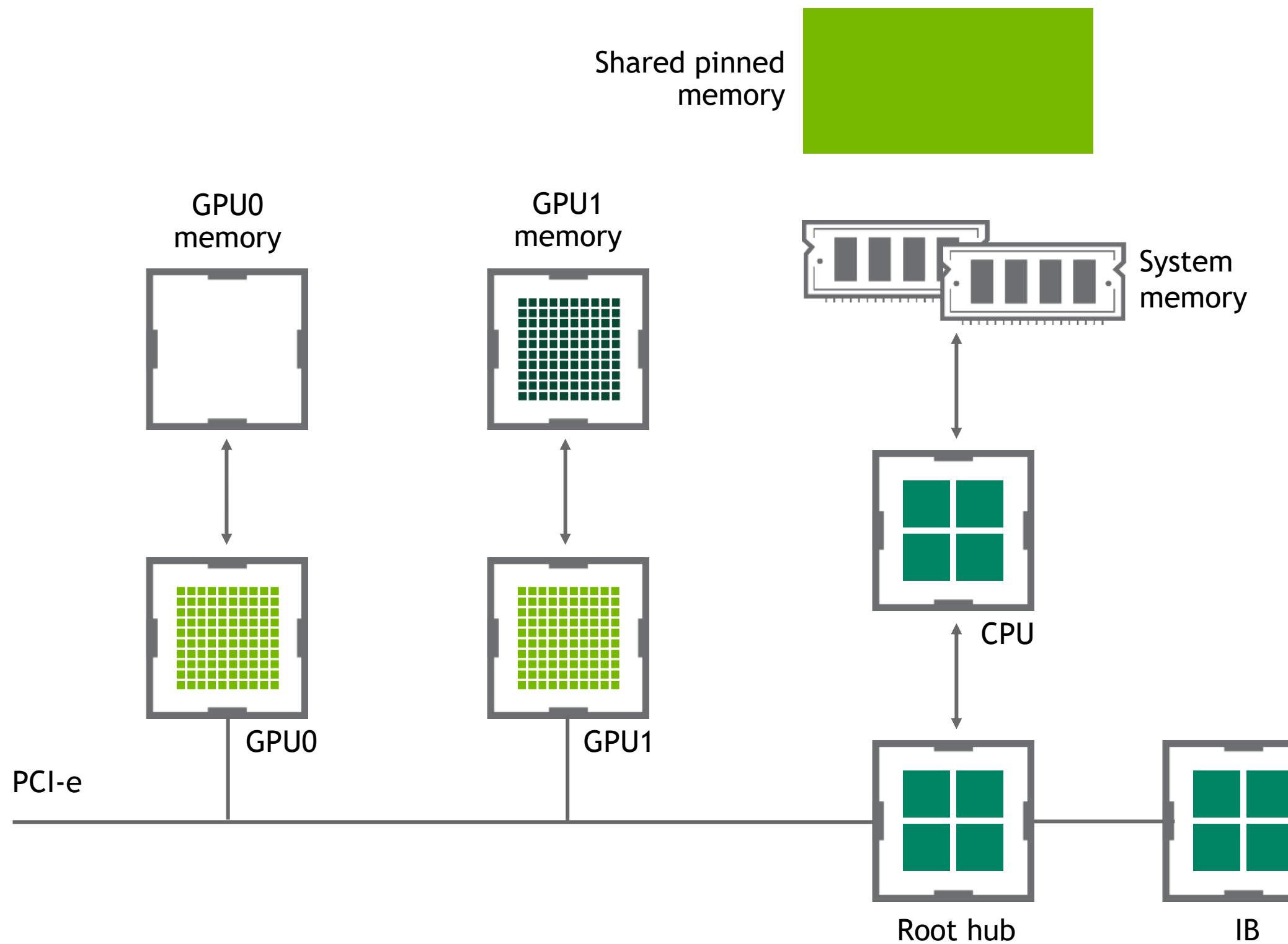




# DATA TRANSFER BOTTLENECK

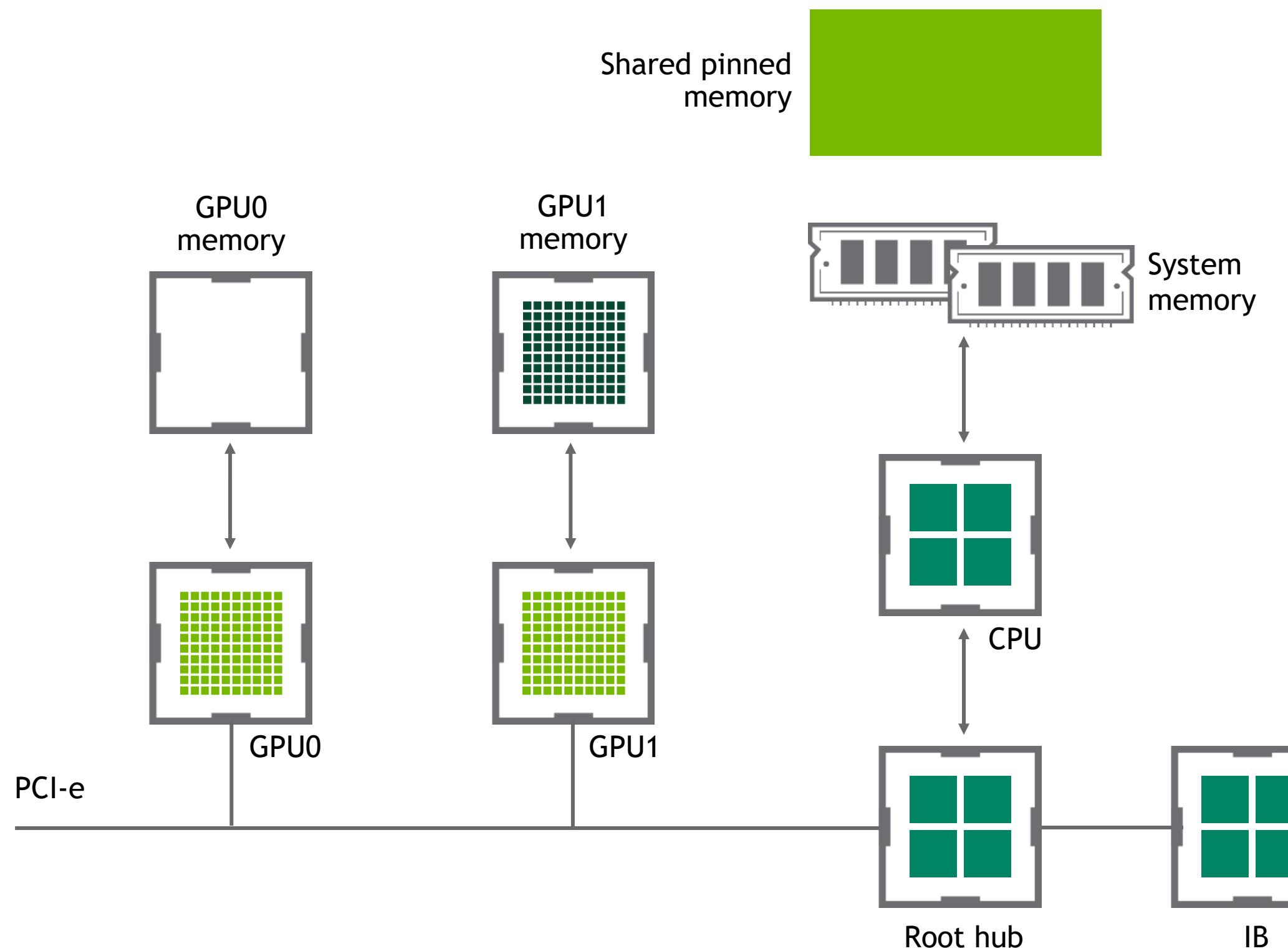
# PEER TO PEER DATA TRANSFER BOTTLENECK

## Without GPUDirect



# PEER TO PEER DATA TRANSFER ~~BOTTLENECK~~

With GPUDirect



**GPUDirect P2P Transfer:** Data is copied directly to another GPU memory.

**GPUDirect P2P Access:** GPU Peers can directly access and transfer data between them if they are on the same root hub.

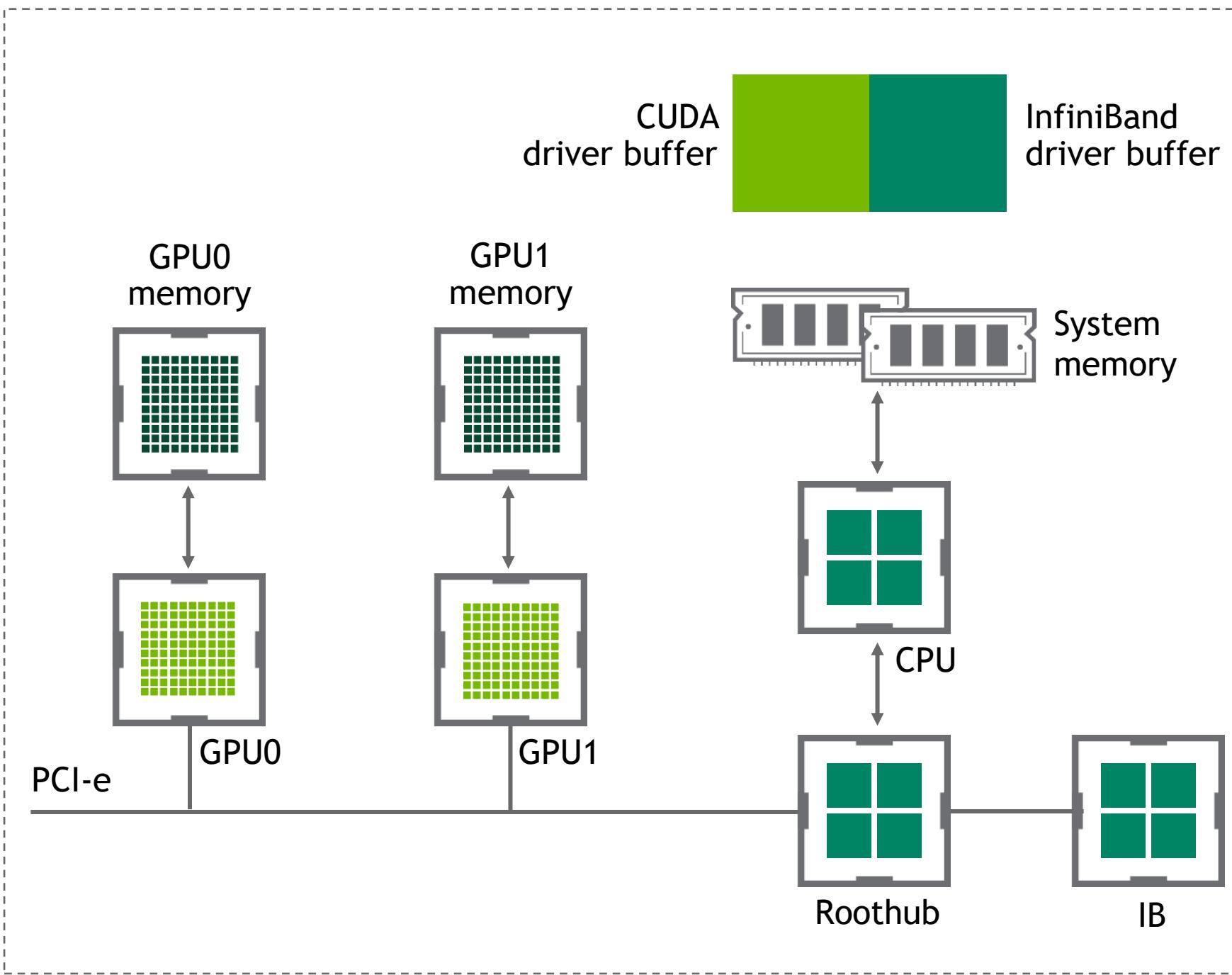
Compute latency is minimized because no CPU involvement is required to drive staging data.

Removes bottlenecks in PCIe topology.

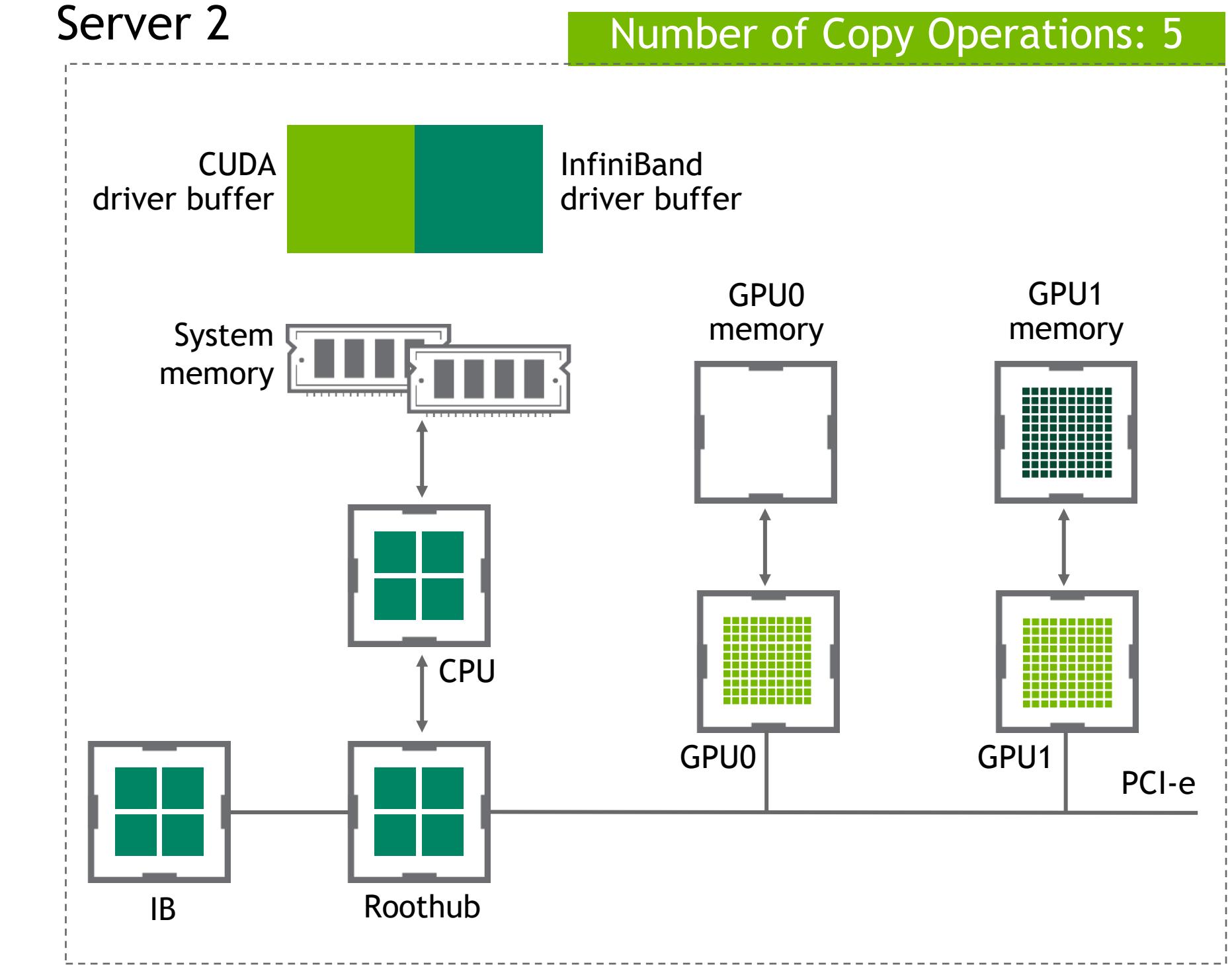
# INTER-NODE DATA TRANSFER BOTTLENECK

## Without GPUDirect RDMA

Server 1



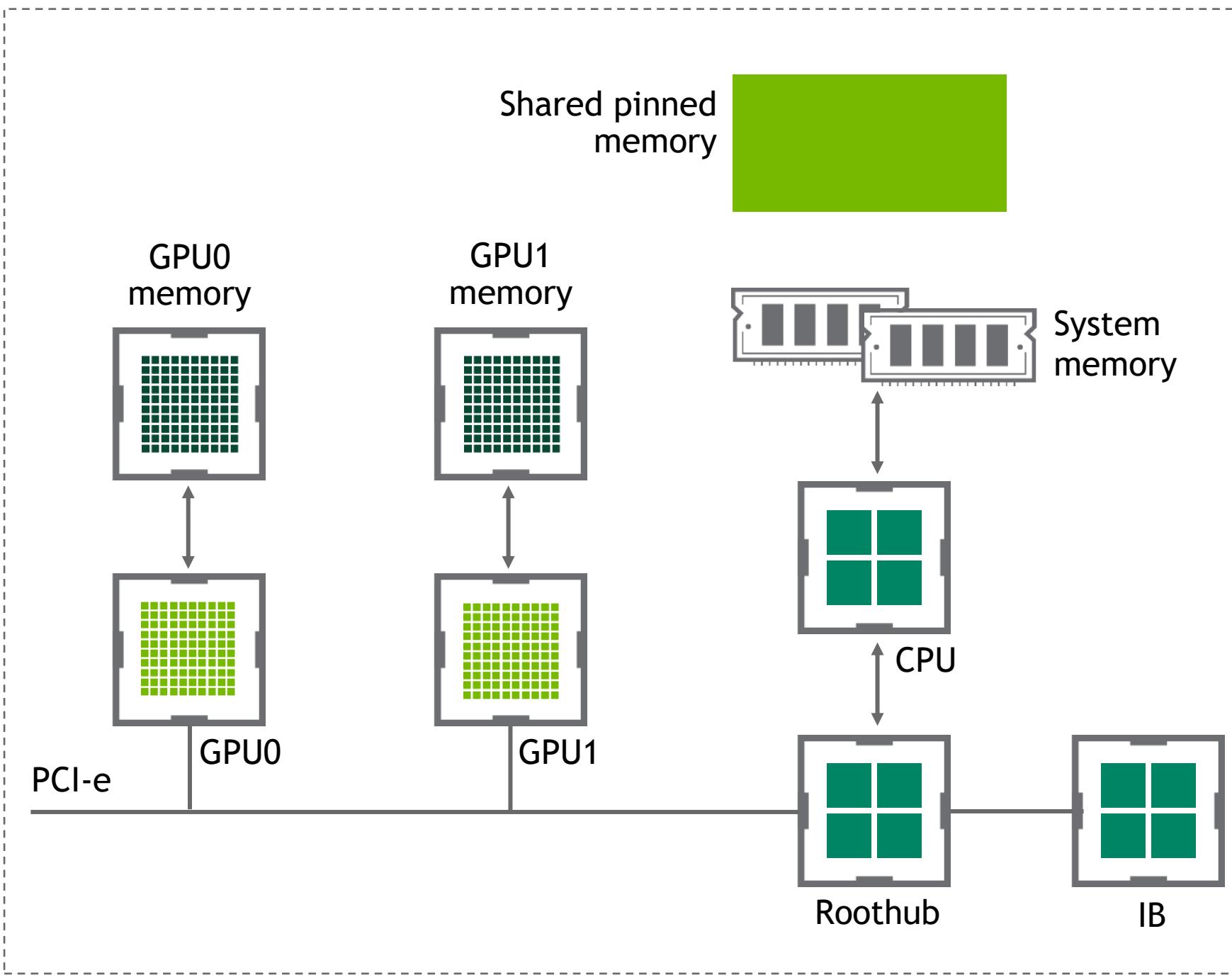
Server 2



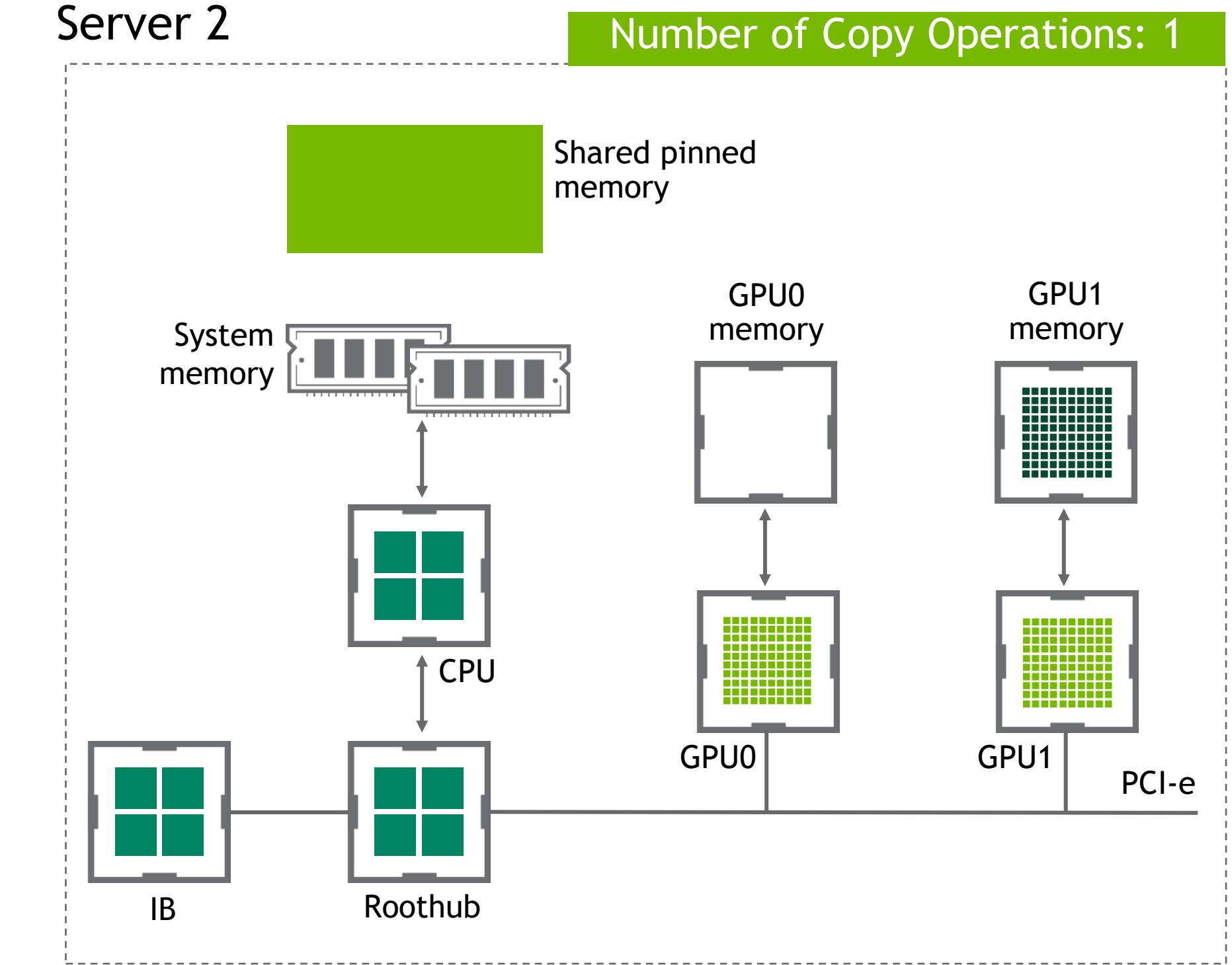
# INTER-NODE DATA TRANSFER ~~BOTTLENECK~~

## With GPUDirect RDMA

Server 1

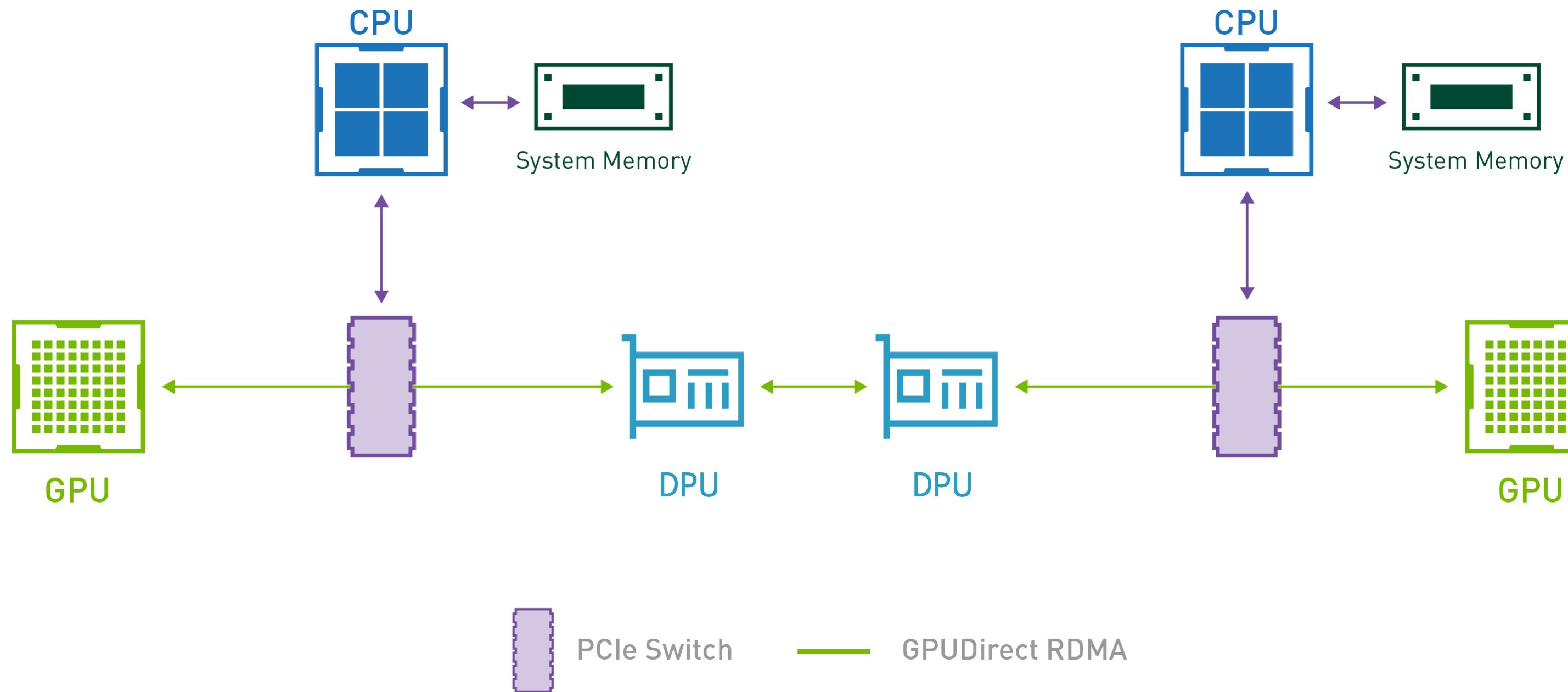


Server 2



# INTER-NODE DATA TRANSFER ~~BOTTLENECK~~

GPUDirect RDMA: Direct Communication between NVIDIA GPUs





# SUMMARY

# ML FRAMEWORKS INTEROPERABILITY

What we have seen today

Interoperability matters.

A single framework is not enough anymore.

Complex workloads make use of multiple libraries.

Interoperability via DLPack and CUDA Array Interface (CAI).

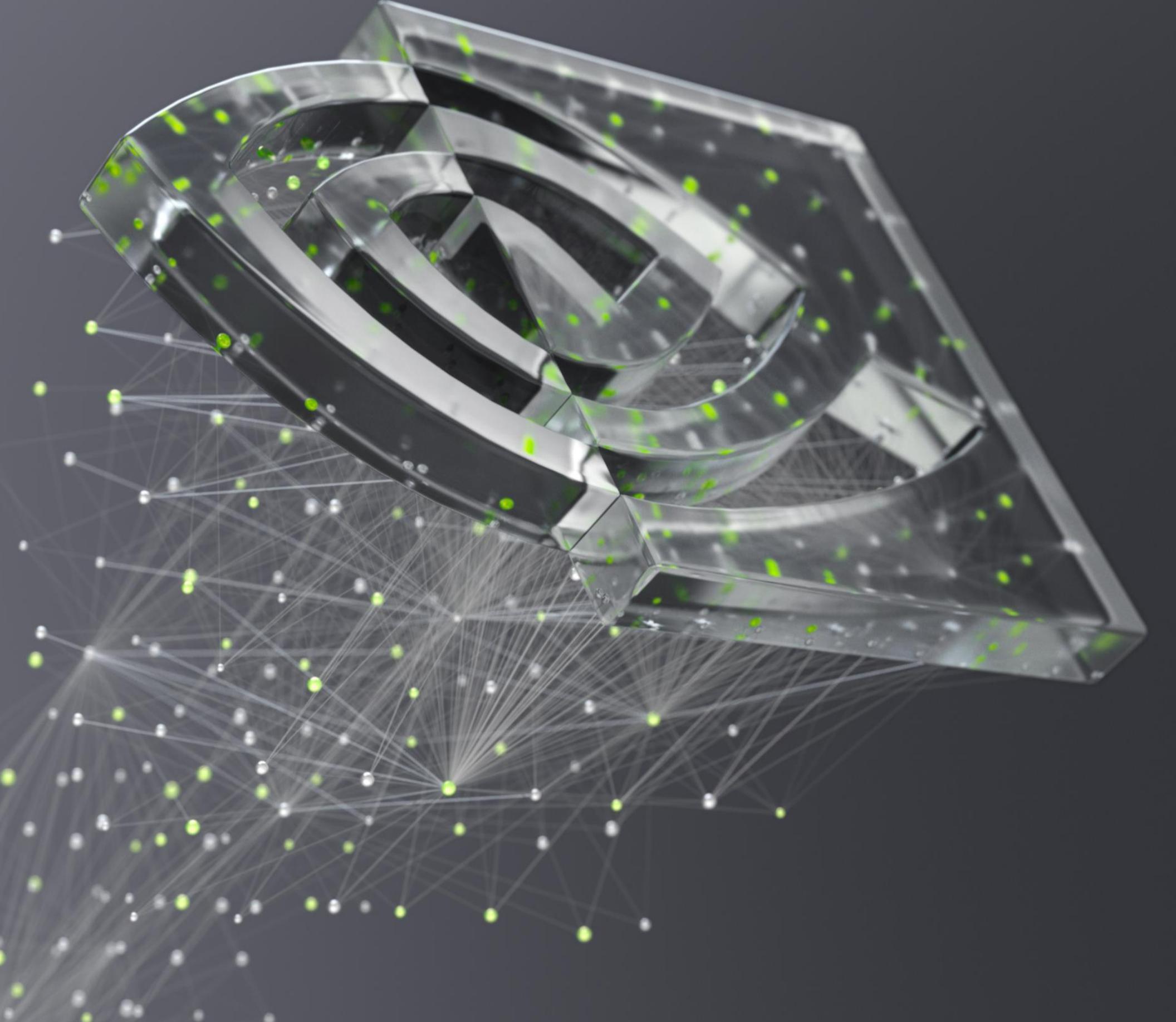
Zero-copy and no data conversion is the goal. Not always possible, yet.

Memory management, data loading and data transfer are common bottlenecks.

That's all folks!

# RESOURCES

- [Blog Post, Part 1: Memory Layouts and Memory Pools](#)
- [Blog Post, Part 2: Data Loading and Data Transfer Bottlenecks](#)
- [Blog Post, Part 3: Zero-Copy in Action using an E2E Pipeline](#)
- [GTC Video Recording | NVIDIA On-Demand](#)
- [Interoperability Cheat Sheet Notebook](#)
- [End-2-End Example Notebook](#)



*Thank  
you!*



NVIDIA®