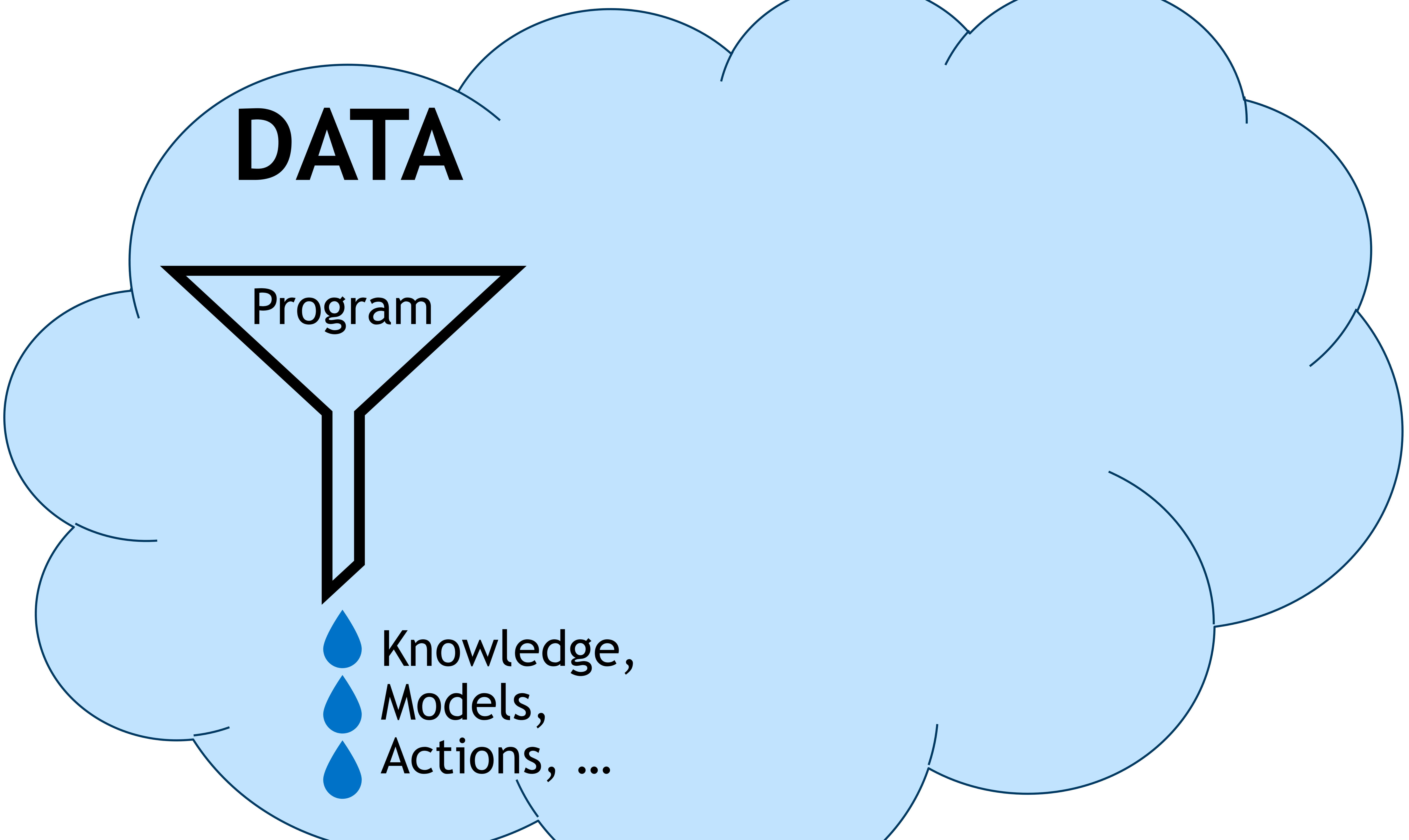
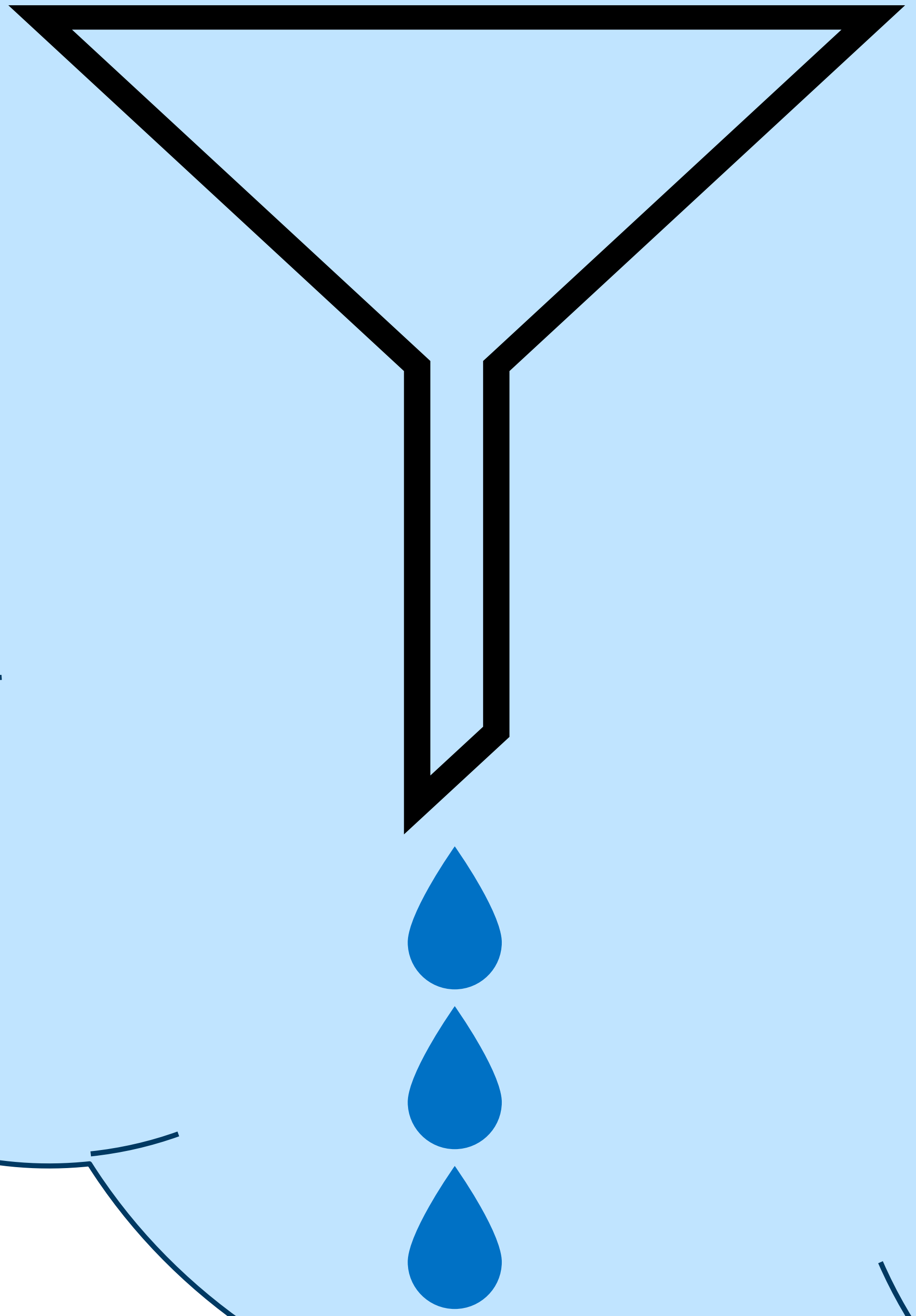
NVIDIA

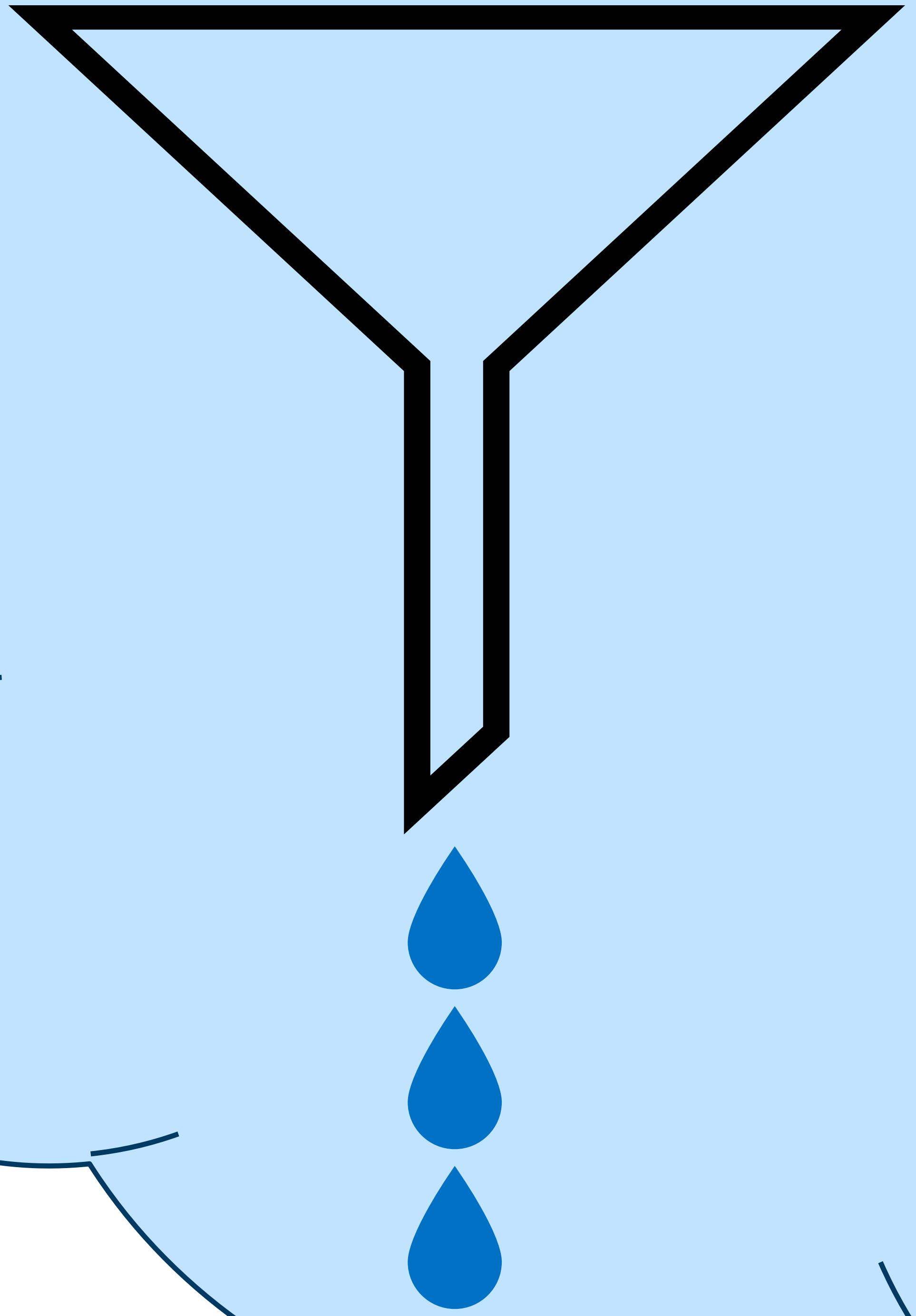
PROGRAMMING SYSTEMS OF DATA

Michael Garland, November 2020


DATA
Program
Knowledge,
Models,
Actions, …



How do we organize the *external data* that a program accesses/acquires?



How do we organize the *internal data* that a program processes/generates?

DATA STRUCTURES

Every programming system provides common building blocks

List

```
>>> powers_of_two()
[1, 2, 4, 8, 16, 32]
```

Dictionary

```
>>> identify_speaker()
{'first name': 'Michael', 'last name': 'Garland'}
```

Array

```
>>> A = numpy.ones((5,8), dtype=int)
array([[1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1]])
```

DATA STRUCTURES

Means by which we compose separate procedures

'A' is produced

```
>>> A = numpy.ones((5,8), dtype=int)
array([[1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1]])
```

'A' is consumed

```
>>> sum(A)
array([5, 5, 5, 5, 5, 5, 5, 5])
```

PROGRAMMING SYSTEM

Good design enables convenient expression of algorithms

```
import numpy as np

def cg_solve(A, b):
    x = np.zeros(A.shape[1])
    r = b - A.dot(x)
    p = r
    rsold = r.dot(r)
    for i in xrange(b.shape[0]):
        Ap = A.dot(p)
        alpha = rsold / (p.dot(Ap))
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = r.dot(r)
        if np.sqrt(rsnew) < 1e-10:
            break
        beta = rsnew / rsold
        p = r + beta * p
        rsold = rsnew
     return x
```

WHERE’S THE PROBLEM?

Convenient Expression

+

Compositional Software

+

High Performance

COSTS OF COMPOSITION

Potential for data copying & conversion at interface boundaries

```
import numpy
from scipy.sparse import csr_matrix

rows    = [0, 0, 1, 1, 2, 2, 2, 3, 3]
columns = [0, 1, 1, 2, 0, 2, 3, 1, 3]
entries = [1, 7, 2, 8, 5, 3, 9, 6, 4]

A = csr_matrix((entries, (rows, columns)),
               shape=(4,4),
               dtype=int)

>>> A.toarray()
array([[1, 7, 0, 0],
       [0, 2, 8, 0],
       [5, 0, 3, 9],
       [0, 6, 0, 4]], dtype=int32)

>>> numpy.sum(A.data)
45
```

Does my matrix data get copied here?

NUMPY ARRAY INTERFACE

Self-describing data can be reused across software modules without copying

```
>>> A = numpy.zeros((5,8), dtype=int, order='F′ )

array([[0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0]])

>>> A.__array_interface__

{'data': (2137481977920, False),
 'descr': [('', '<i4')],
 'shape': (5, 8),
 'strides': (4, 20),
 'typestr': '<i4',
 'version': 3}
```

Explanation of the array interface: https://numpy.org/doc/stable/reference/arrays.interface.html

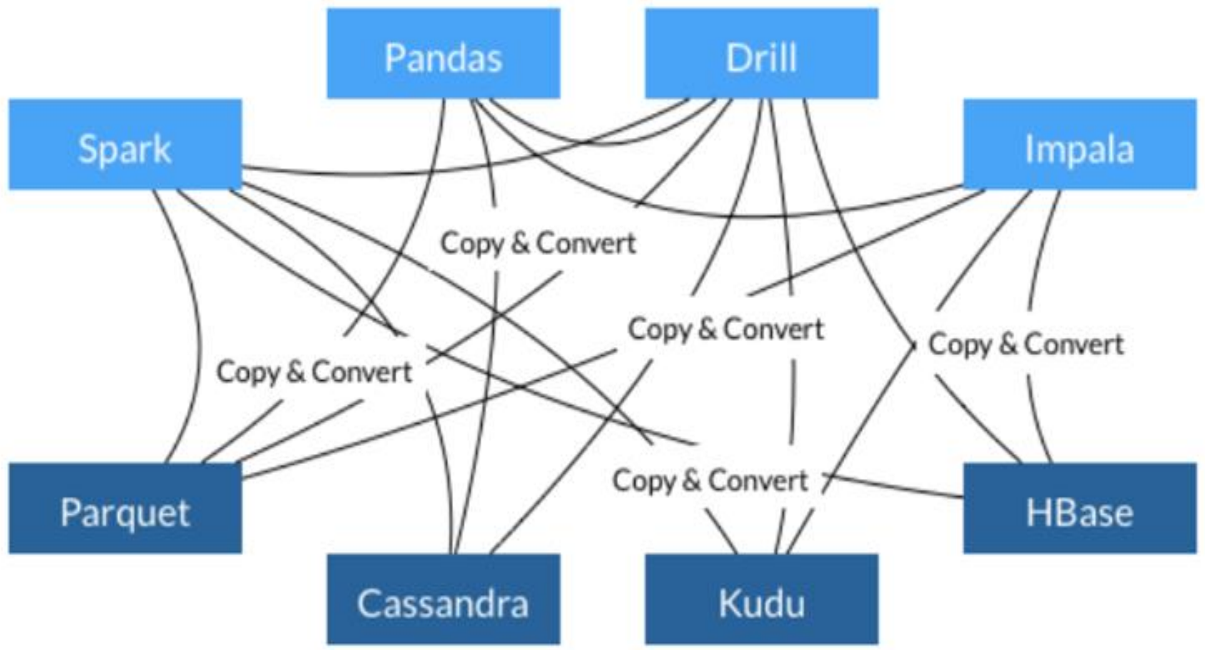
NUMPY ARRAY INTERFACE

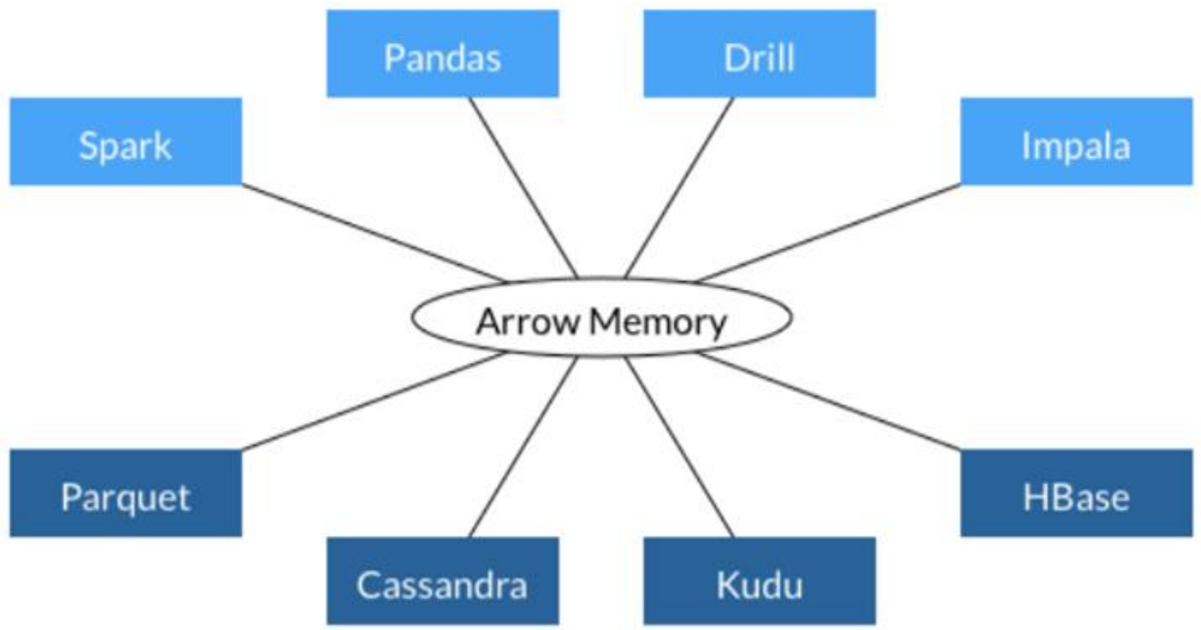
Self-describing data can be reused across software modules without copying

```
>>> A = numpy.zeros((5,8), dtype=int, order='F’ )

array([[0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0]])

>>> A.__array_interface__

{'data': (2137481977920, False),
 'descr': [('', '<i4')],
 'shape': (5, 8),
 'strides': (4, 20),
 'typestr': '<i4',
 'version': 3}
```

See also: Harris, C.R., Millman, K.J., van der Walt, S.J. *et al.* Array programming with NumPy. *Nature* **585**, 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2

Learning from Apache Arrow





- Each system has its own internal memory format
- 70-80% computation wasted on serialization and deserialization
- Similar functionality implemented in multiple projects

- All systems utilize the same memory format
- No overhead for cross-system communication
- Projects can share functionality (eg, Parquet-to-Arrow reader)

Source: From Apache Arrow Home Page - https://arrow.apache.org/

More here: http://rapids.ai

APACHE ARROW

Specifying the memory layout for dataframes

Mortgage ID	Pay Date	Amount
101	12/18/2018	1029.30
102	12/21/2018	1429.31
103	12/14/2018	1289.27
101	01/15/2018	1104.59
102	01/17/2018	1457.15
103	NULL	NULL

Mortgage ID

```
data = [101, 102, 103, 101, 102, 103]
size = 6
type = INT
bitmask = [0x3F] = [0 0 1 1 1 1 1 1]
```

Pay Date

```
data = [1545091200000, 1545350400000, 1544745600000,
        1514764800000, 1516147200000, *garbage* ]
size = 6
type = DATE
bitmask = [0x1F] = [0 0 0 1 1 1 1 1]
```

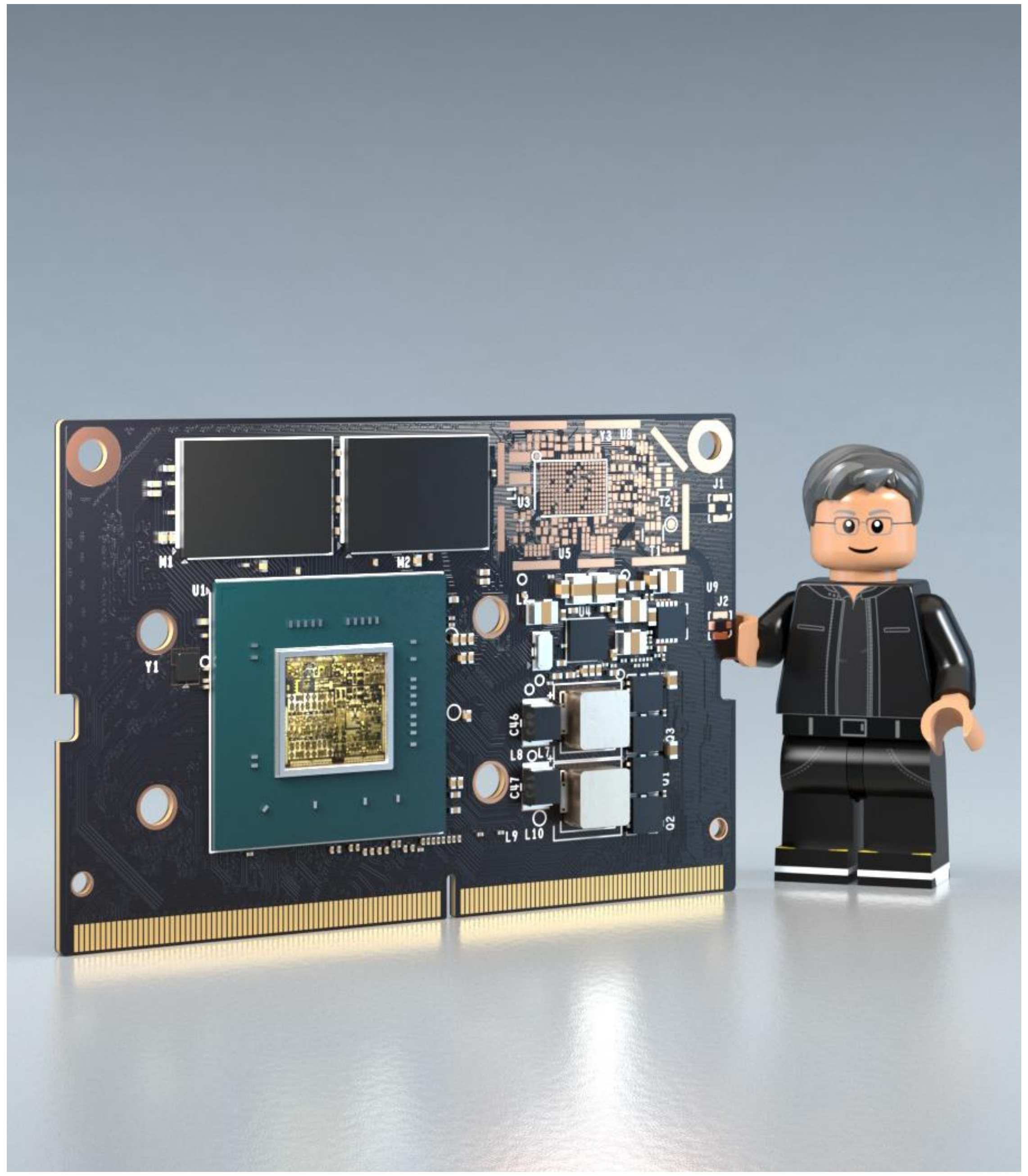
Amount

```
data = [1029.30, 1429.31, 1289.27,
        1104.59, 1457.15, *garbage*]
size = 6
type = FLOAT
bitmask = [0x1F] = [0 0 0 1 1 1 1 1]
```

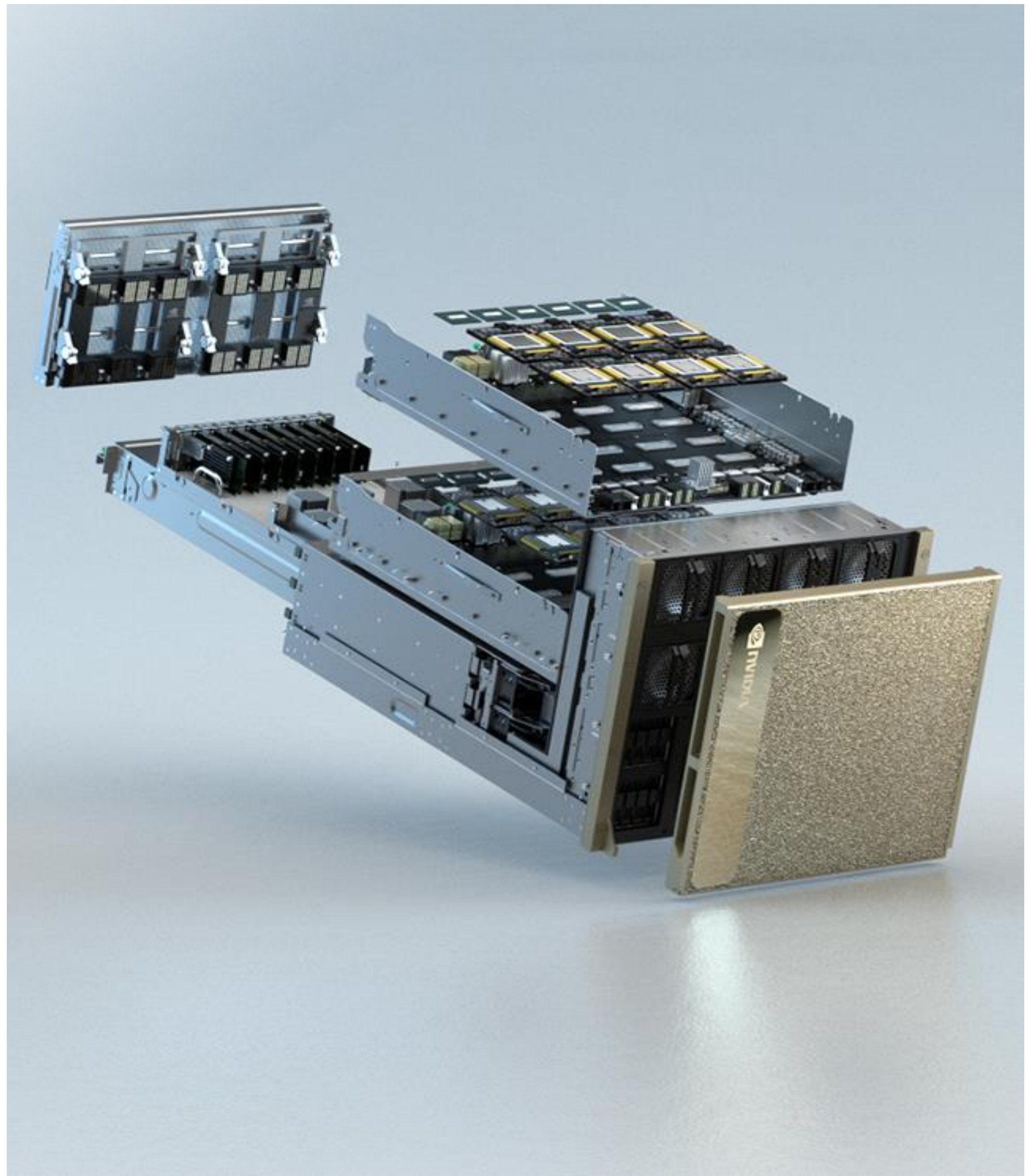
SCALABLE EXECUTION

Achieving high performance at any scale

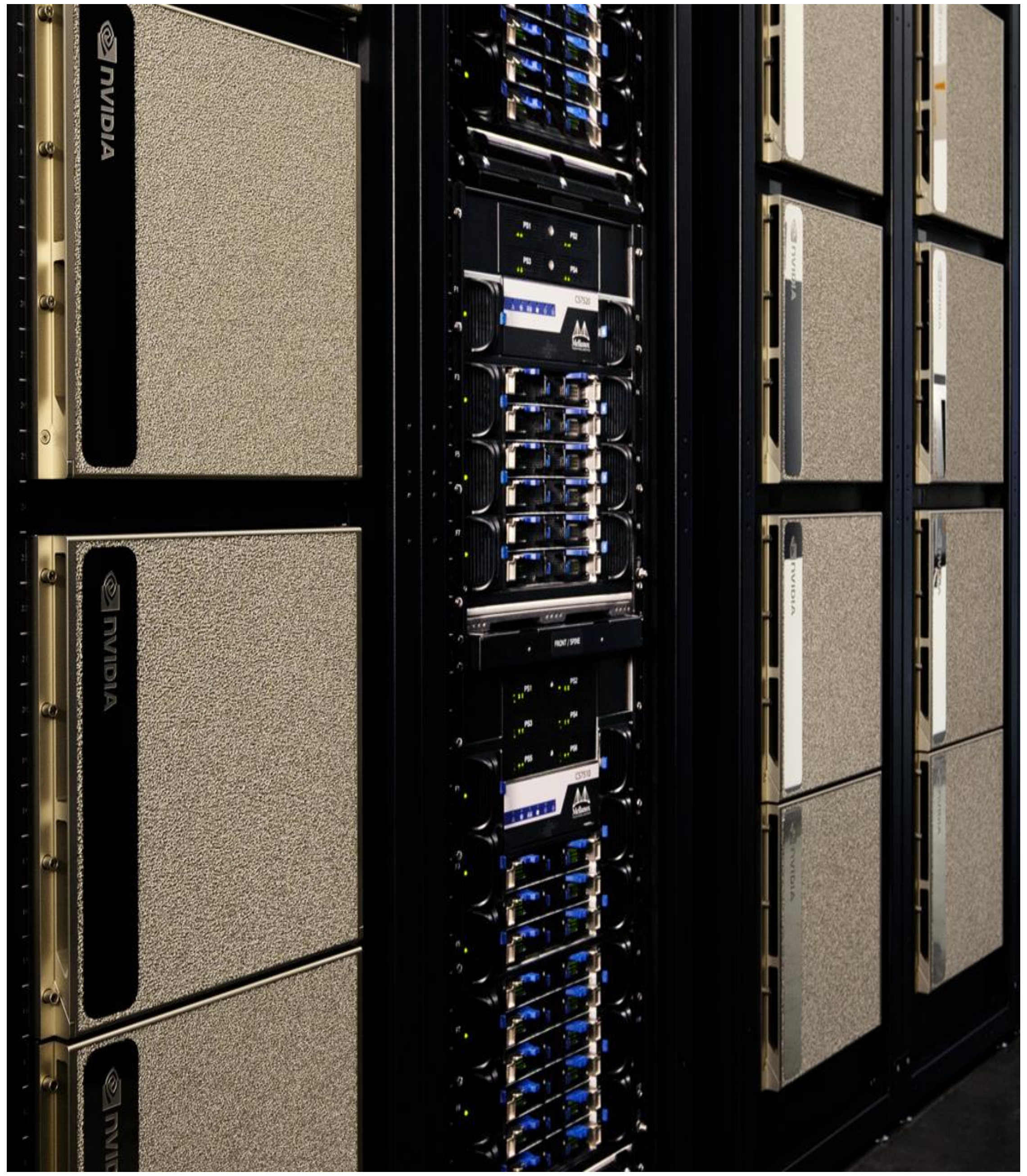
Jetson AGX Xavier



DGX-2



DGX SuperPod



SCALABLE EXECUTION

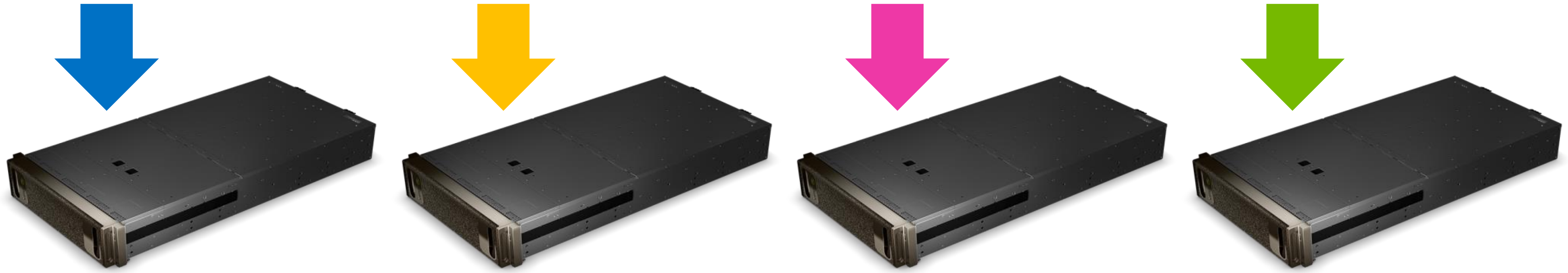
With convenient, compositional software interfaces

```
import numpy as np

def cg_solve(A, b):
    x = np.zeros(A.shape[1])
    r = b - A.dot(x)
    p = r
    rsold = r.dot(r)
    for i in xrange(b.shape[0]):
        Ap = A.dot(p)
        alpha = rsold / (p.dot(Ap))
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = r.dot(r)
        if np.sqrt(rsnew) < 1e-10:
            break
        beta = rsnew / rsold
        p = r + beta * p
        rsold = rsnew
    return x
```

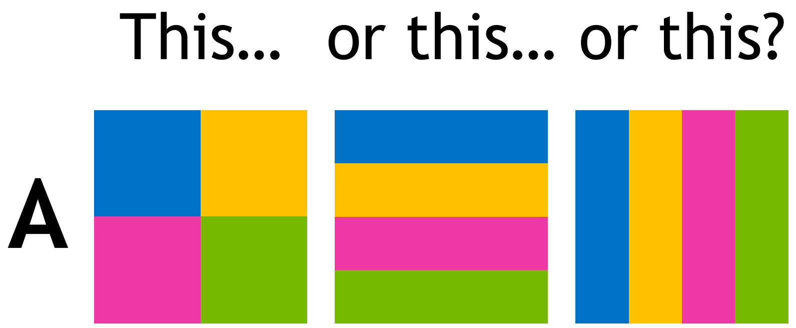
- Many existing interfaces expose ample parallelism
- Data often large enough to warrant large machines
- How can programming systems help provide this experience?

ORGANIZING DATA

Scaling requires partitioning and movement of data



- Data needs to be partitioned and moved between nodes as needed



- Choice may depend on multiple factors, including operations being performed

- Programming systems can help orchestrate the placement and movement of data

LEGATE

LEGATE PROGRAMMING SYSTEM

Couple convenient notation with accelerated libraries via an advanced runtime system

```
try: import legate.numpy as np
except: import numpy as np

def cg_solve(A, b):
    x = np.zeros(A.shape[1])
    r = b - A.dot(x)
    p = r
    rsold = r.dot(r)
    for i in xrange(b.shape[0]):
        Ap = A.dot(p)
        alpha = rsold / (p.dot(Ap))
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = r.dot(r)
        if np.sqrt(rsnew) < 1e-10:
            break
        beta = rsnew / rsold
        p = r + beta * p
        rsold = rsnew
     return x
```

Familiar Domain-Specific Interfaces

Legate

Data-Driven Task Runtime System

Legion

Accelerated Libraries

cuBLAS, cuSPARSE, cuDNN, cuDF, cuGRAPH, cuIO, …

Early Access Release:
http://developer.nvidia.com/legate

MAKE ADOPTION INCREDIBLY SIMPLE

Even if users are a bit arbitrary in their adoption strategy

```
import random, numpy, legate.numpy


# Code written using the NumPy interface provided by "np":
def step1(np, n):      return np.ones(n), np.ones(n)
def step2(np, x, y):   return np.dot(x, y)


# Unorthodox adoption strategy:
numpy_likes = [numpy, legate.numpy]
x,y = step1(random.choice(numpy_likes), 1_000_000_000)
print( step2(random.choice(numpy_likes), x, y) )
```

A BRIEF HISTORY OF LEGION

Designed for HPC, but well-suited to data science

Begun in 2011 at Stanford

Collaboration: Stanford, NVIDIA, Los Alamos

Built for HPC applications on supercomputers

HPC CASE STUDY

S3D port to Legion + Singe kernel DSL

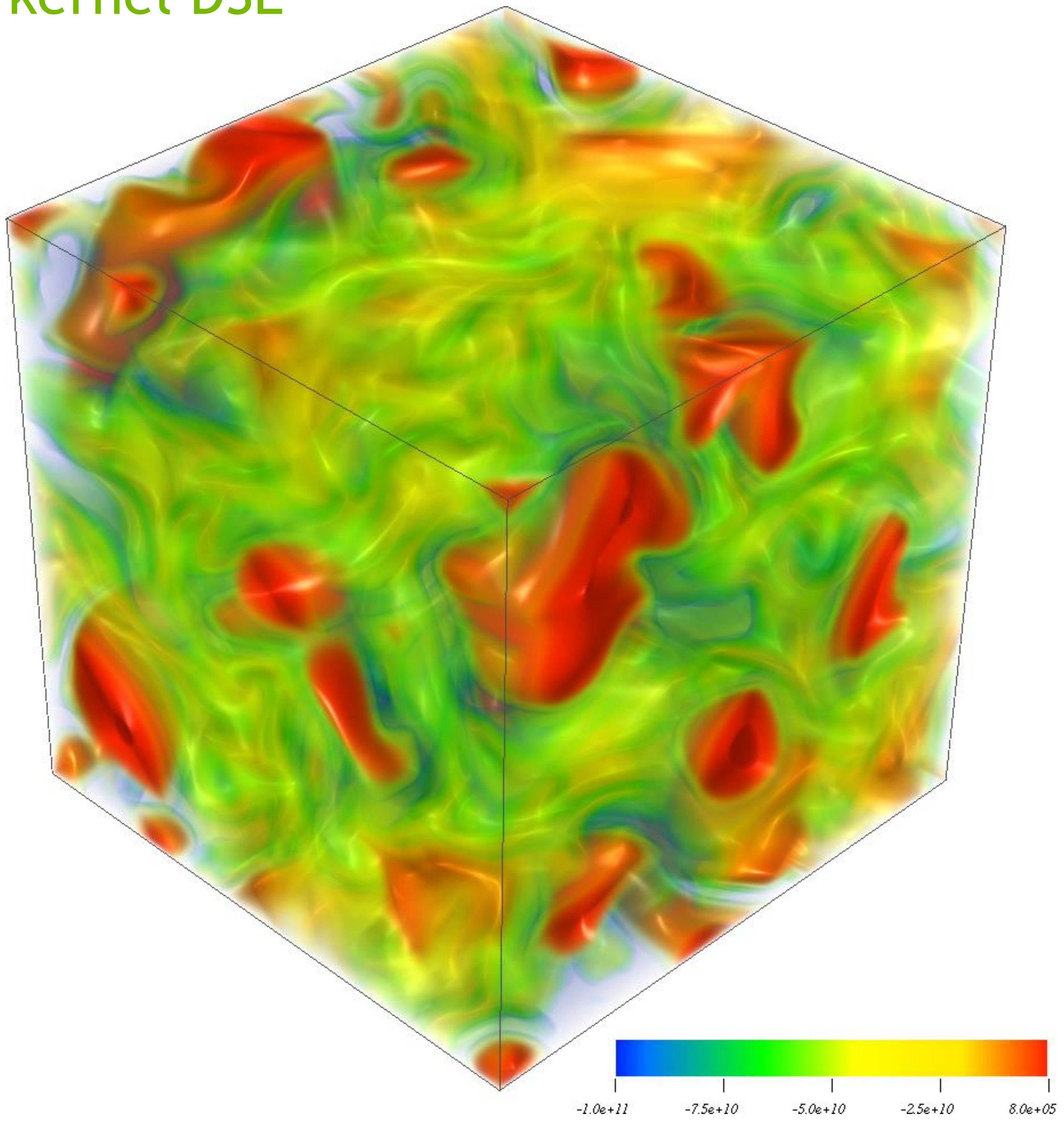
Rewrote 100K lines of S3D combustion application

Up to 6x faster than MPI+vector Fortran

Up to 2.85x faster than MPI+OpenACC

Port from development system to Titan: 14 hours

Second port to Piz Daint: 4 hours

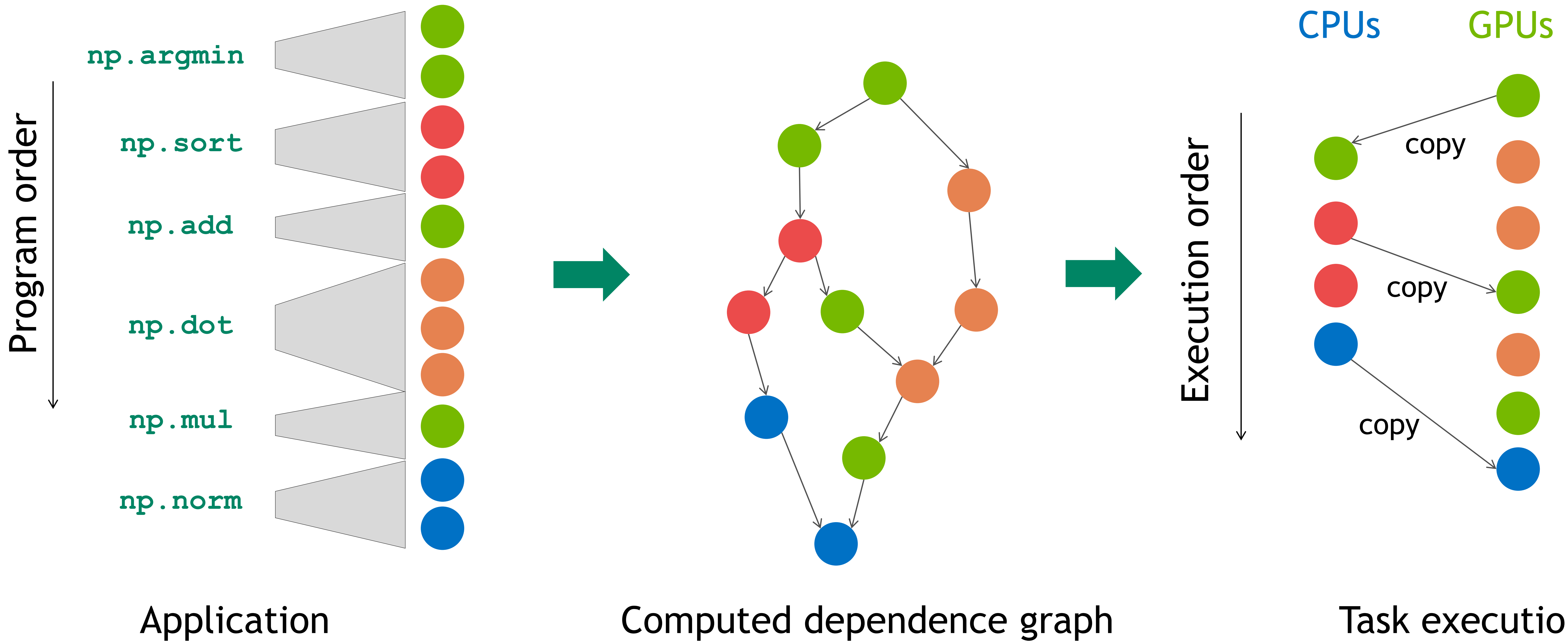


Collaboration amongst:

NVIDIA Los Alamos NATIONAL LABORATORY EST. 1943 Sandia National Laboratories

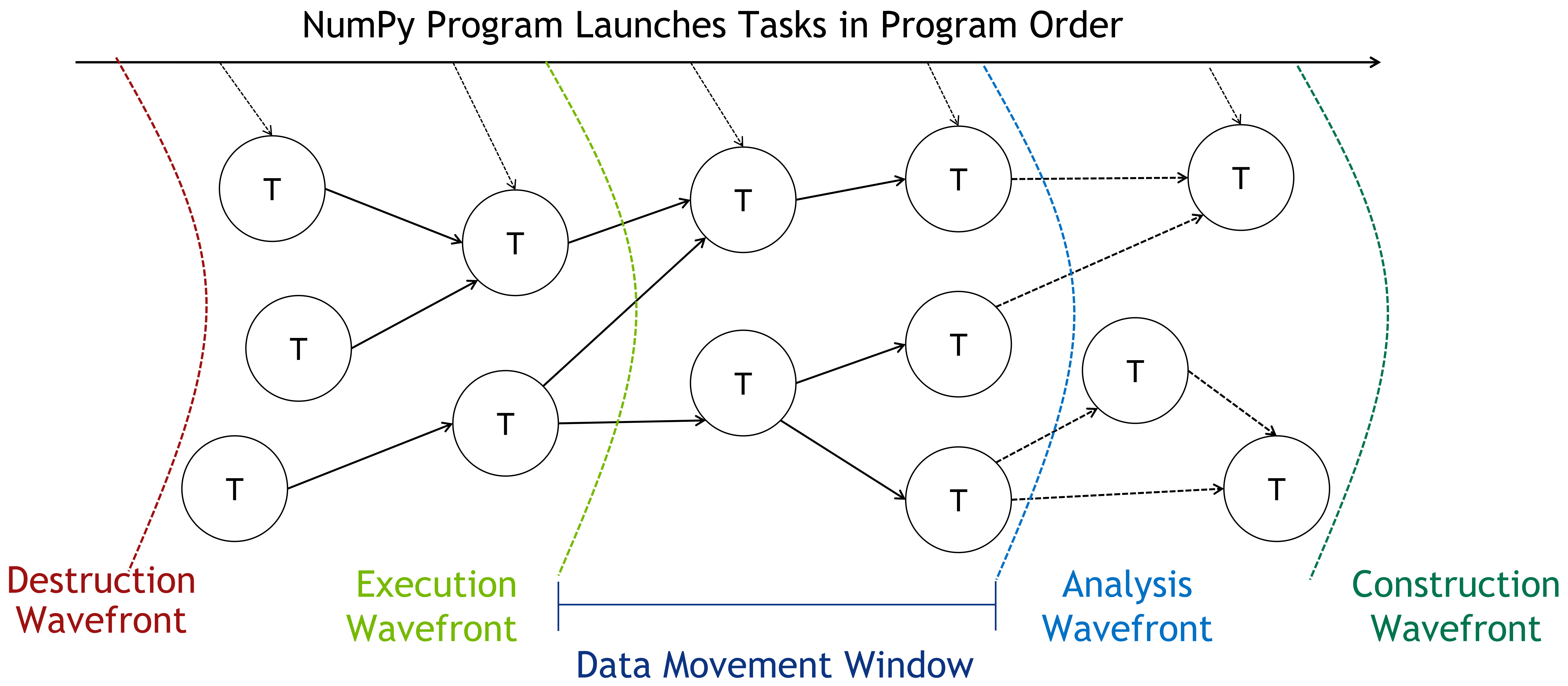
LEGATE ARCHITECTURE

Leveraging task-based runtimes to build a programming system for data analytics



DEFERRED EXECUTION MODEL

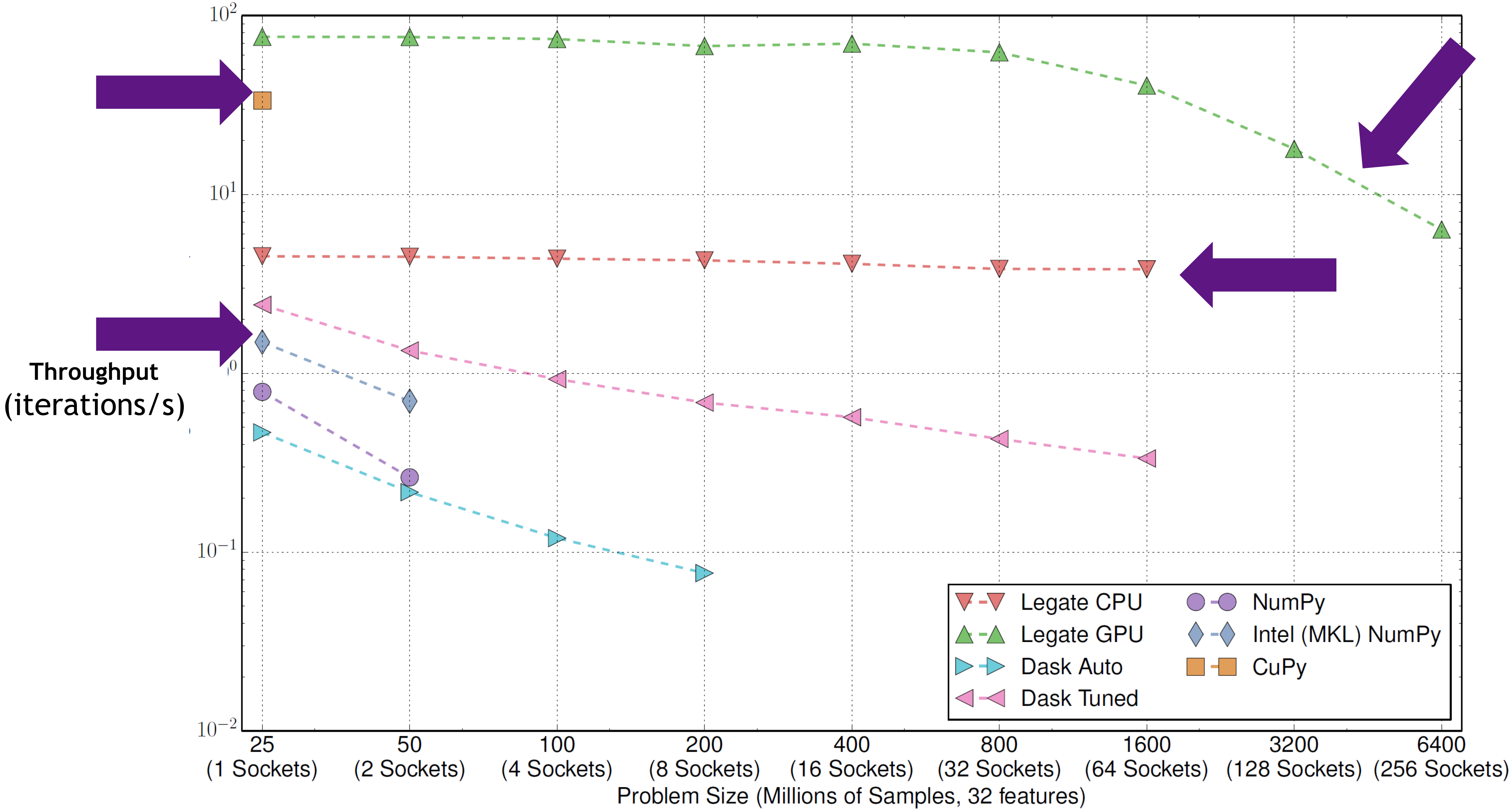
Enables dynamic analysis regardless of application control flow



EXAMPLE: LOGISTIC REGRESSION

```
def logistic_regression(T, features, target, steps,
                        learning_rate, sample, add_intercept=False):
    if add_intercept:
        intercept = np.ones((features.shape[0],1), dtype=T)
        features = np.hstack((intercept, features))

    weights = np.zeros(features.shape[1], dtype=T)

    for step in range(steps):
        scores = np.dot(features, weights)
        predictions = sigmoid(scores)

        error = target - predictions
        gradient = np.dot(error,features)
        weights += learning_rate * gradient

        if step % sample == 0:
            print('Log Likelihood of step '+str(step)+': '+
                  str(log_likelihood(features, target, weights)))

    return weights
```

EXAMPLE: LOGISTIC REGRESSION



FLEXFLOW

FLEXFLOW

Beyond data and model parallelism for deep neural networks

Data-parallel decomposition

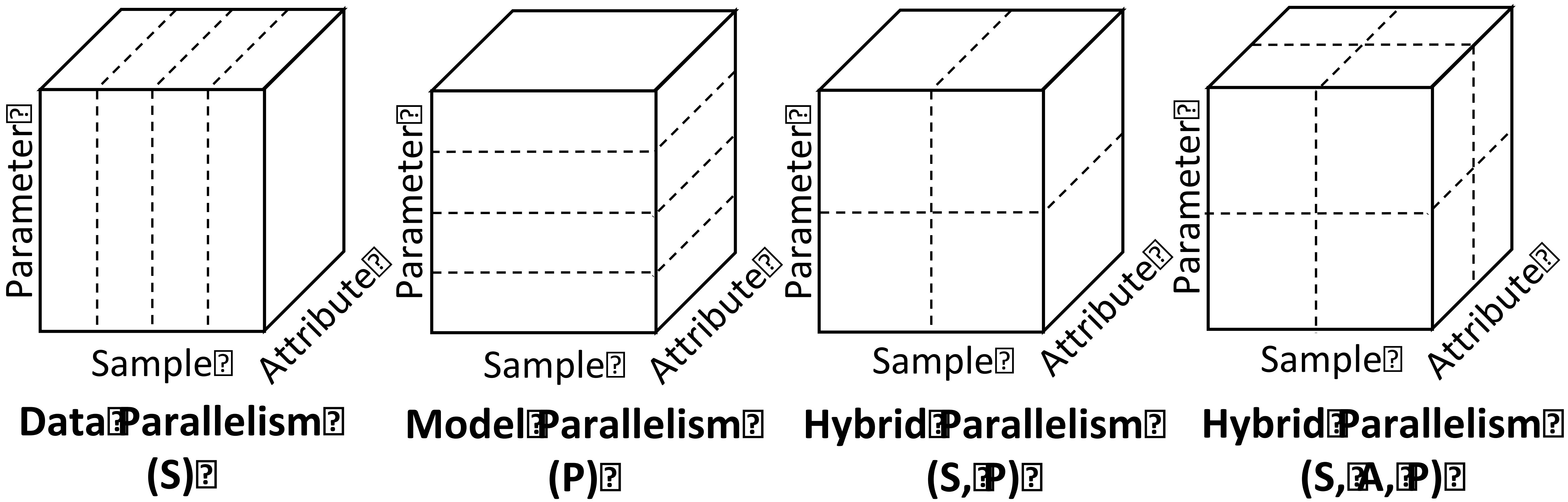
Model-parallel decomposition

Hybrid decompositions (many)

Zhihao Jia, Matei Zaharia, and Alex Aiken.
Beyond Data and Model Parallelism for Deep Neural Networks.
In Proc. 2nd Conf. on Machine Learning and Systems (MLSys), Palo Alto, CA, April 2019.

Zhihao Jia, Sina Lin, Charles R. Qi, and Alex Aiken.
Exploring Hidden Dimensions in Parallelizing Convolutional Neural Networks.
In Proc. Int'l Conf. on Machine Learning (ICML), Stockholm, Sweden, July 2018

SEARCHING FOR STRATEGIES



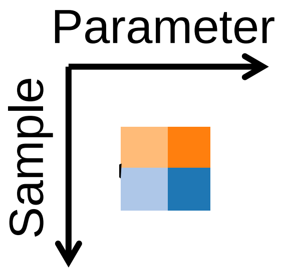
Example parallelization strategies for 1D convolution

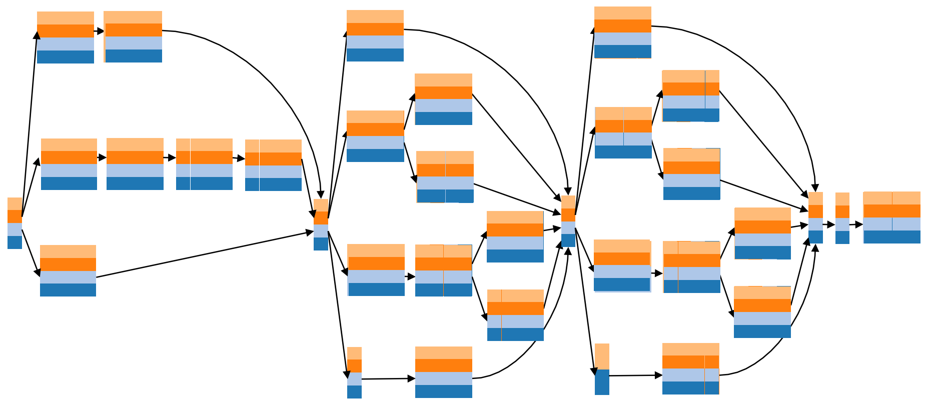
Different strategies perform the same computation.

CASE STUDY

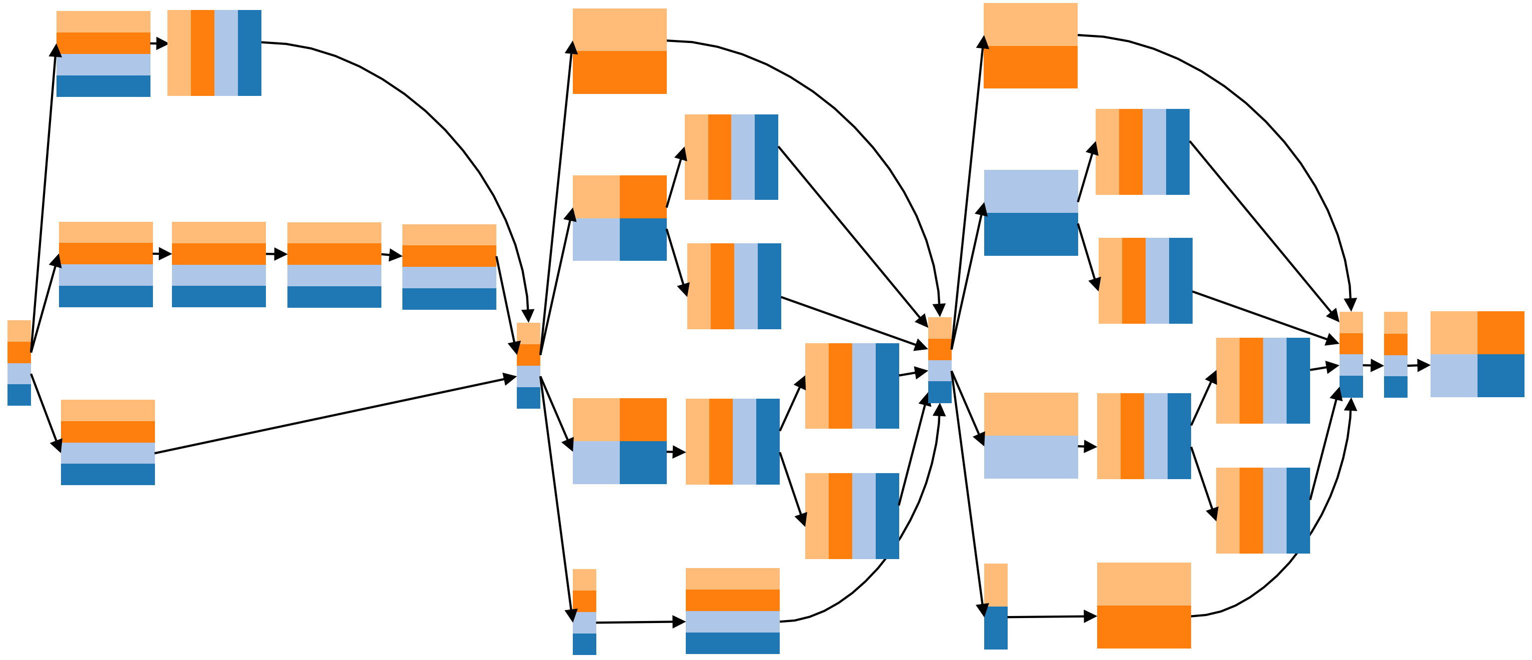
Inception-v3 DNN training on ImageNet

- GPU 1
- GPU 2
- GPU 3
- GPU 4





Data parallelism

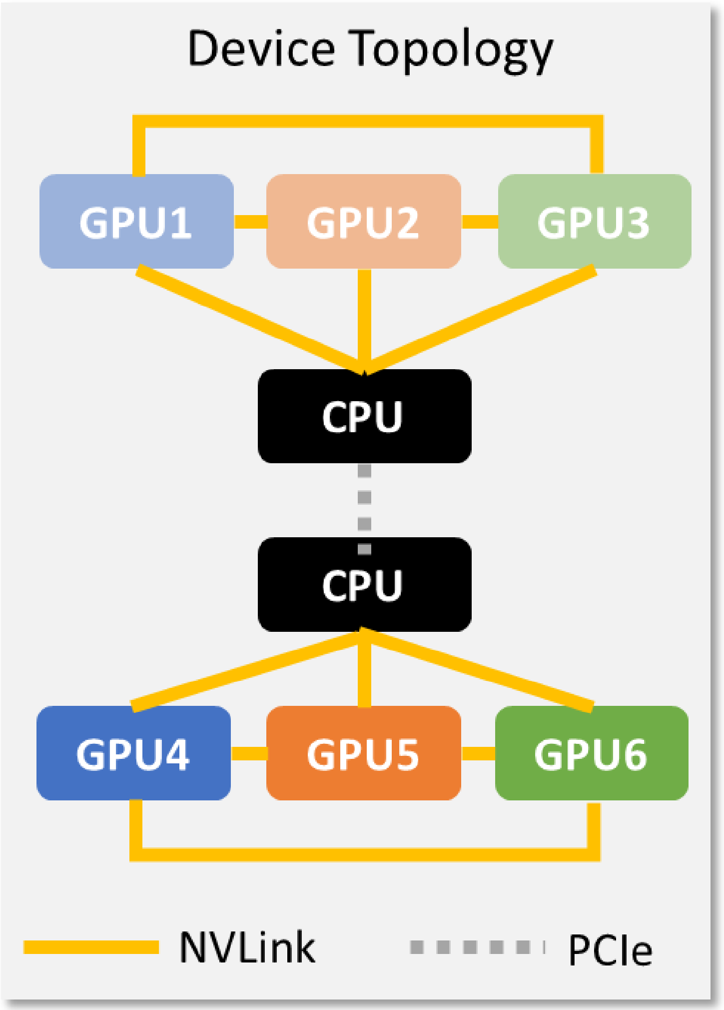


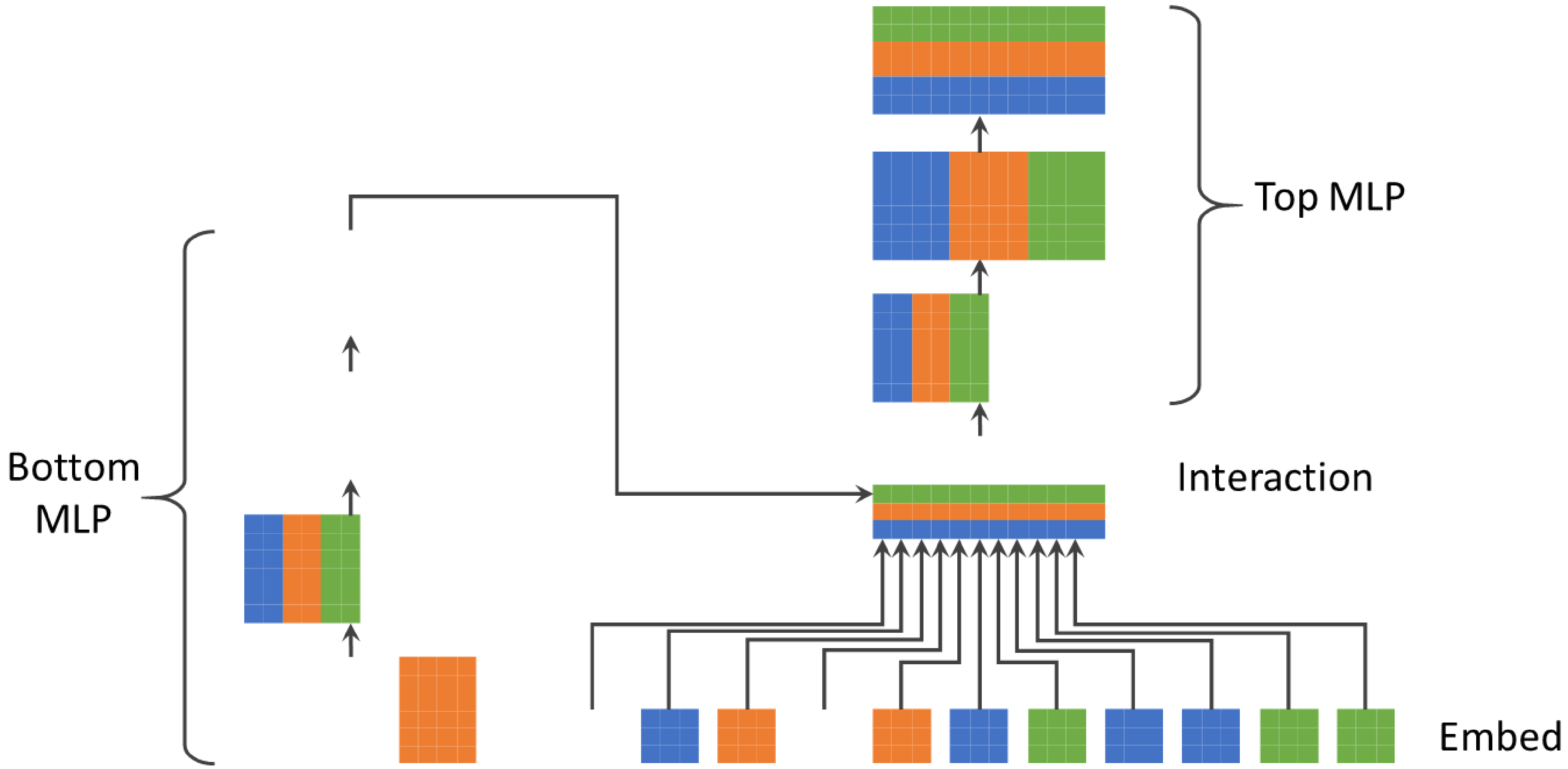
A parallelization strategy found by SOAP search procedure

1.2x faster

CASE STUDY

DLRM training on Criteo Kaggle dataset



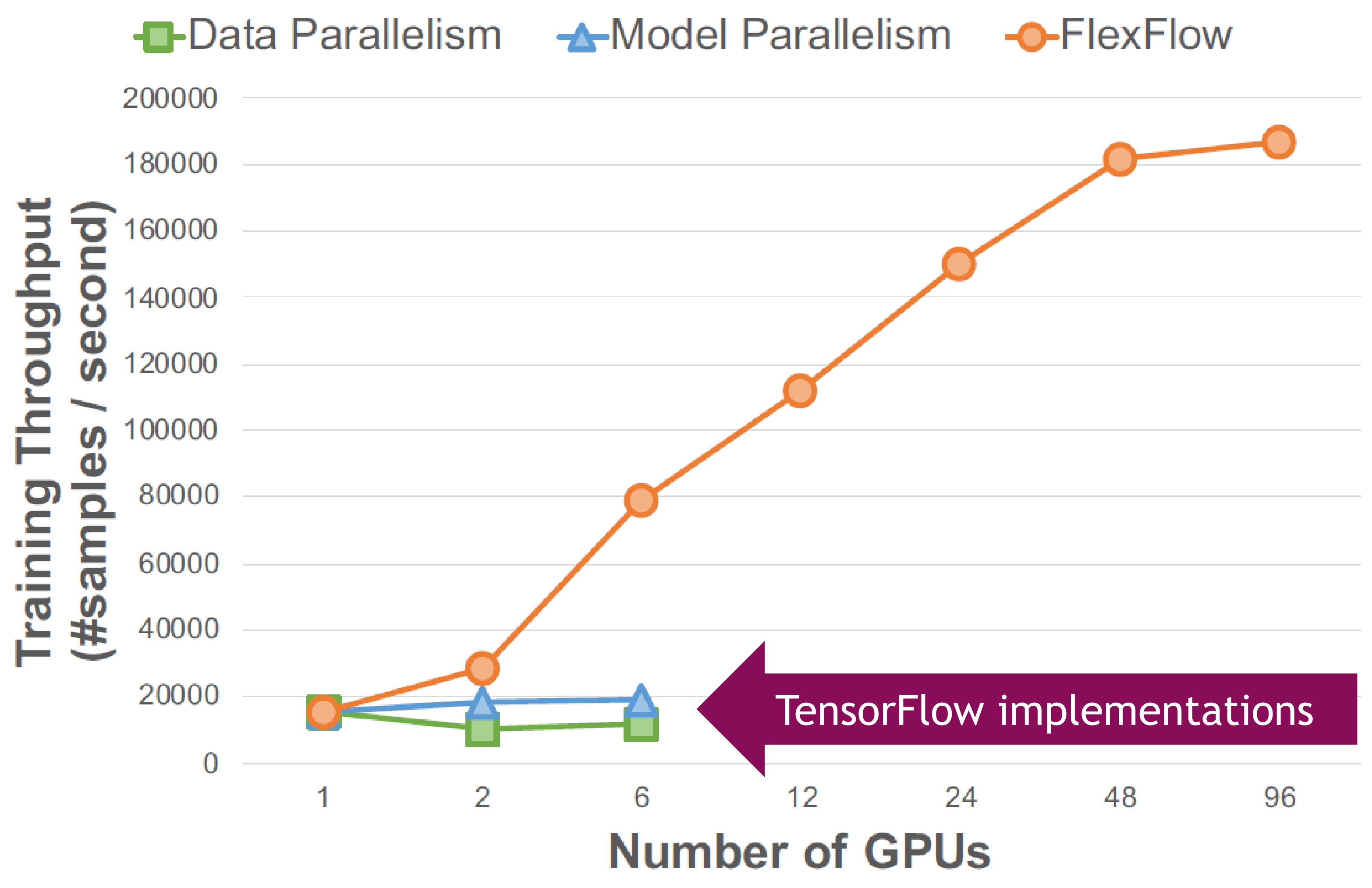


Layers	Communication	Compute	Strategy
Embed	Heavy	Light	Parallelism across **Operators**
Interaction	Light	Heavy	Parallelism across **Samples**
MLP	Heavy	Heavy	Parallelism across **Samples** and **Parameters**

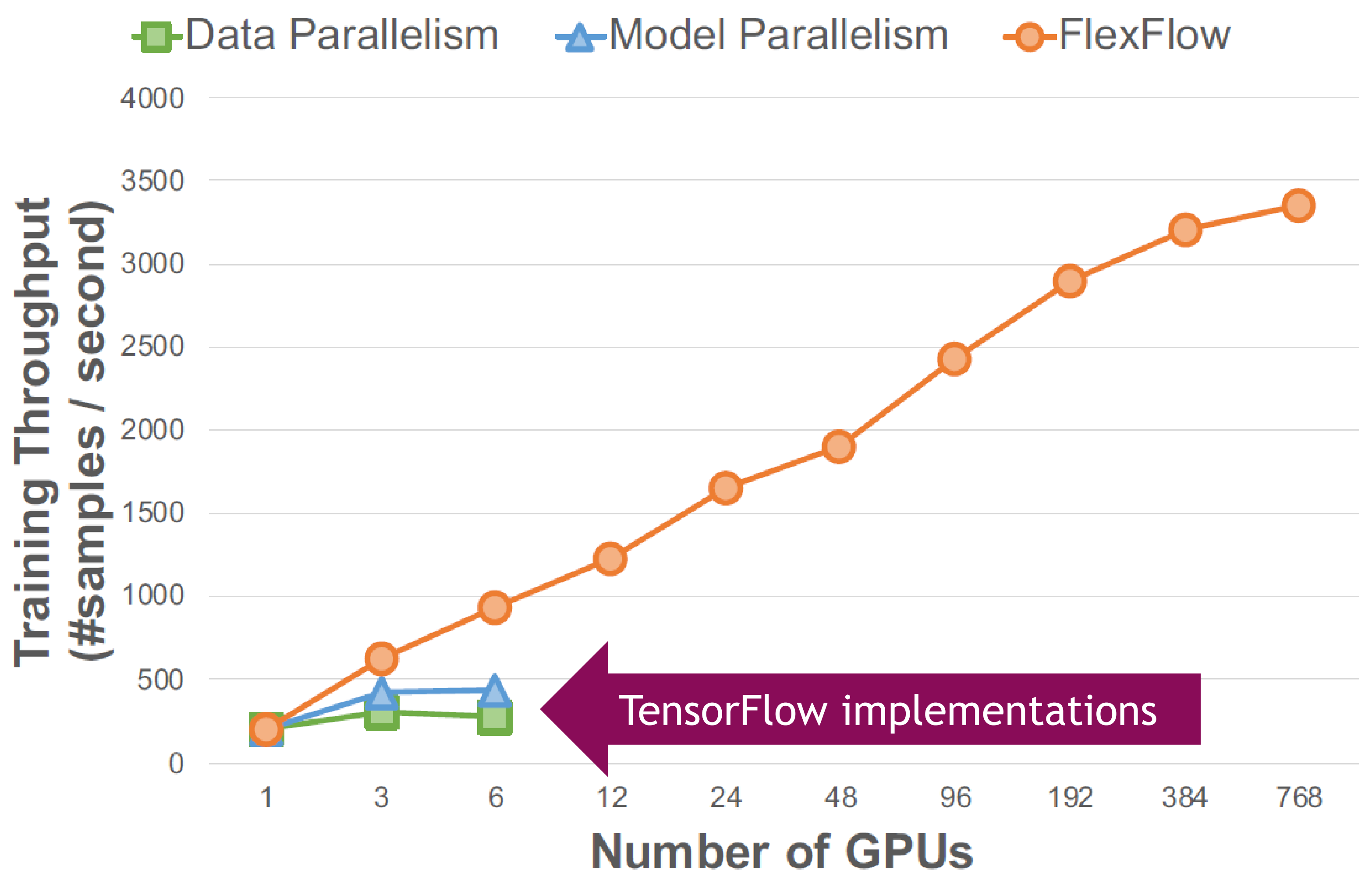
DLRM - https://github.com/facebookresearch/dlrm

SCALING STUDY

Hybrid parallelism enabled good



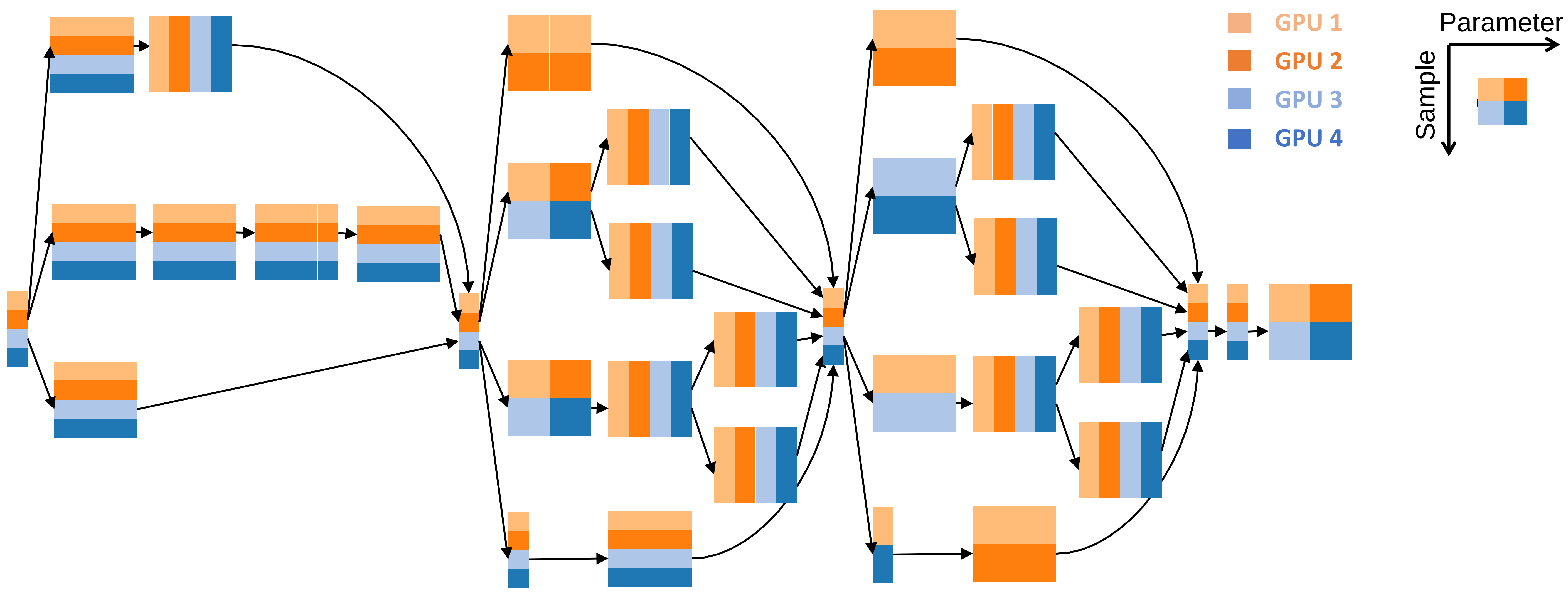
(a) DLRM
https://github.com/facebookresearch/dlrm



(b) Candle Uno
https://github.com/ECP-CANDLE/Benchmarks

Training performance for DLRM and Candle Uno on the Summit supercomputer.

ML: Find the right architecture and parallel decomposition



HPC: Design efficient programming system for executing them

LEGION RUNTIME

QUESTION

What makes a runtime system like Legion well-suited to this problem?

```
try: import legate.numpy as np
except: import numpy as np

def cg_solve(A, b):
    x = np.zeros(A.shape[1])
    r = b - A.dot(x)
    p = r
    rsold = r.dot(r)
    for i in xrange(b.shape[0]):
        Ap = A.dot(p)
        alpha = rsold / (p.dot(Ap))
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = r.dot(r)
        if np.sqrt(rsnew) < 1e-10:
            break
        beta = rsnew / rsold
        p = r + beta * p
        rsold = rsnew
     return x
```

Domain-Specific Program Interfaces

Legate

Data-Driven Runtime System

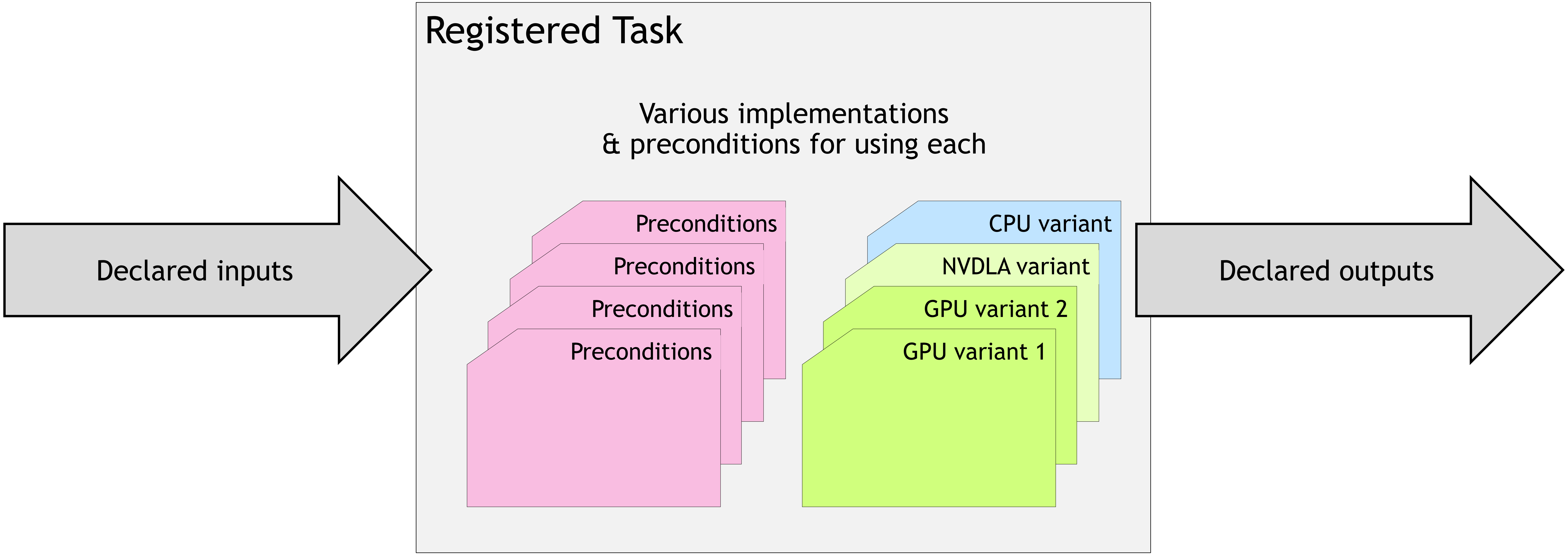
Legion

Accelerated Libraries

cuBLAS, cuDNN, cuDF, cuSPARSE, cuGRAPH, cuIO, …

TASK MODEL

Generalize ubiquitous programming concept: procedures



Task ≈ procedure + stronger encapsulation + descriptive dependencies

Remote Procedure Call Model

Focus on allowing arbitrary procedure signatures and transport of whatever data that implies

vs

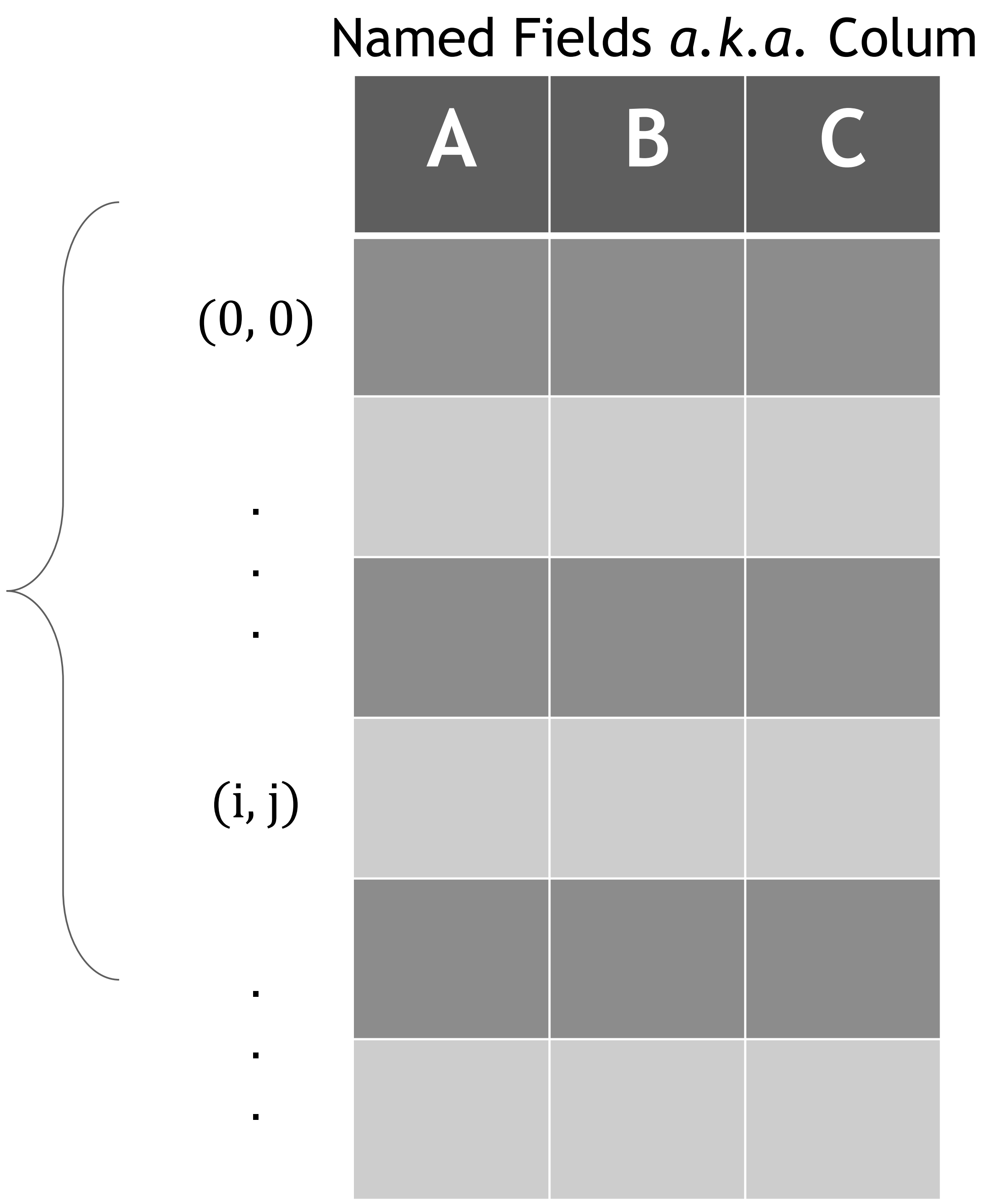
Task Model

Prescribe a data model for parameters to enable runtime to do more on our behalf

LEGION DATA MODEL

Application data logically organized into a tabular representation

Entries addressed by
n-D integer indices

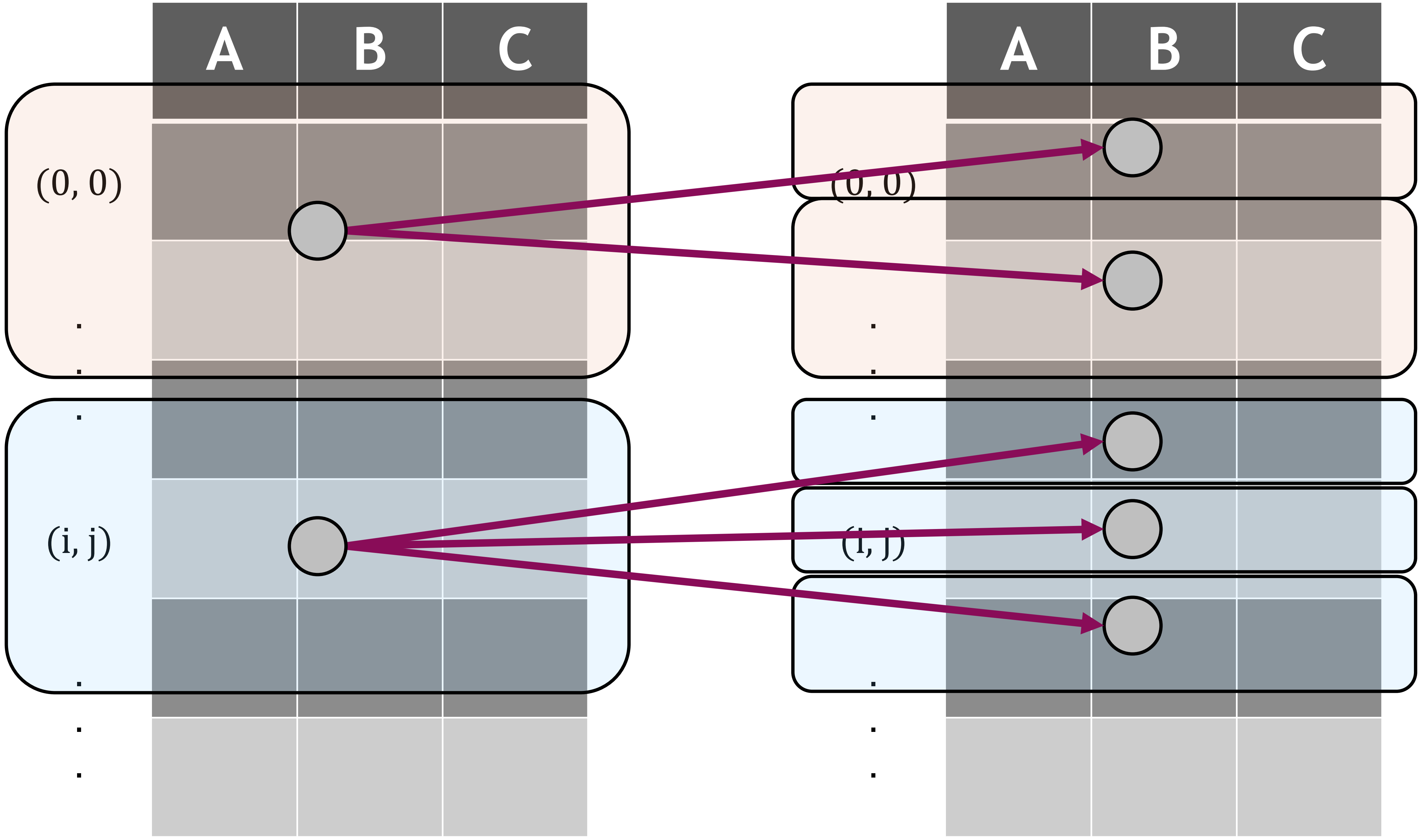


Each field is a
(potentially sparse)
n-D array

LEGION DATA MODEL

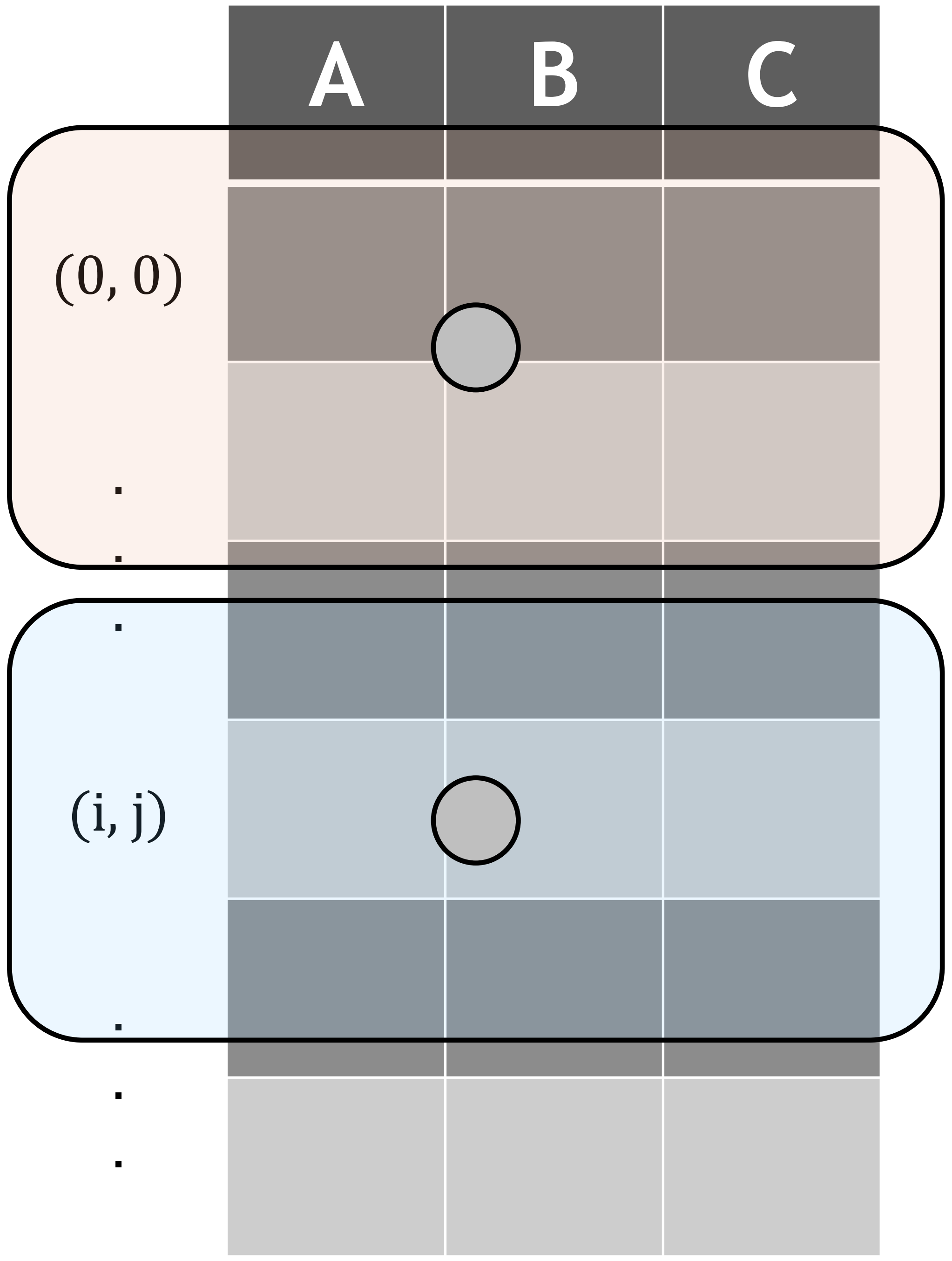
Hierarchical data decomposition

Fine-grained association of pieces of data with tasks



LEGION DATA MODEL

Hierarchical data decomposition



Individual operations specify the partitioning they intend to use.

LEGION DATA MODEL

Exposing data to runtime yields several benefits

Runtime manages versioning, replication, sharding, and movement of data

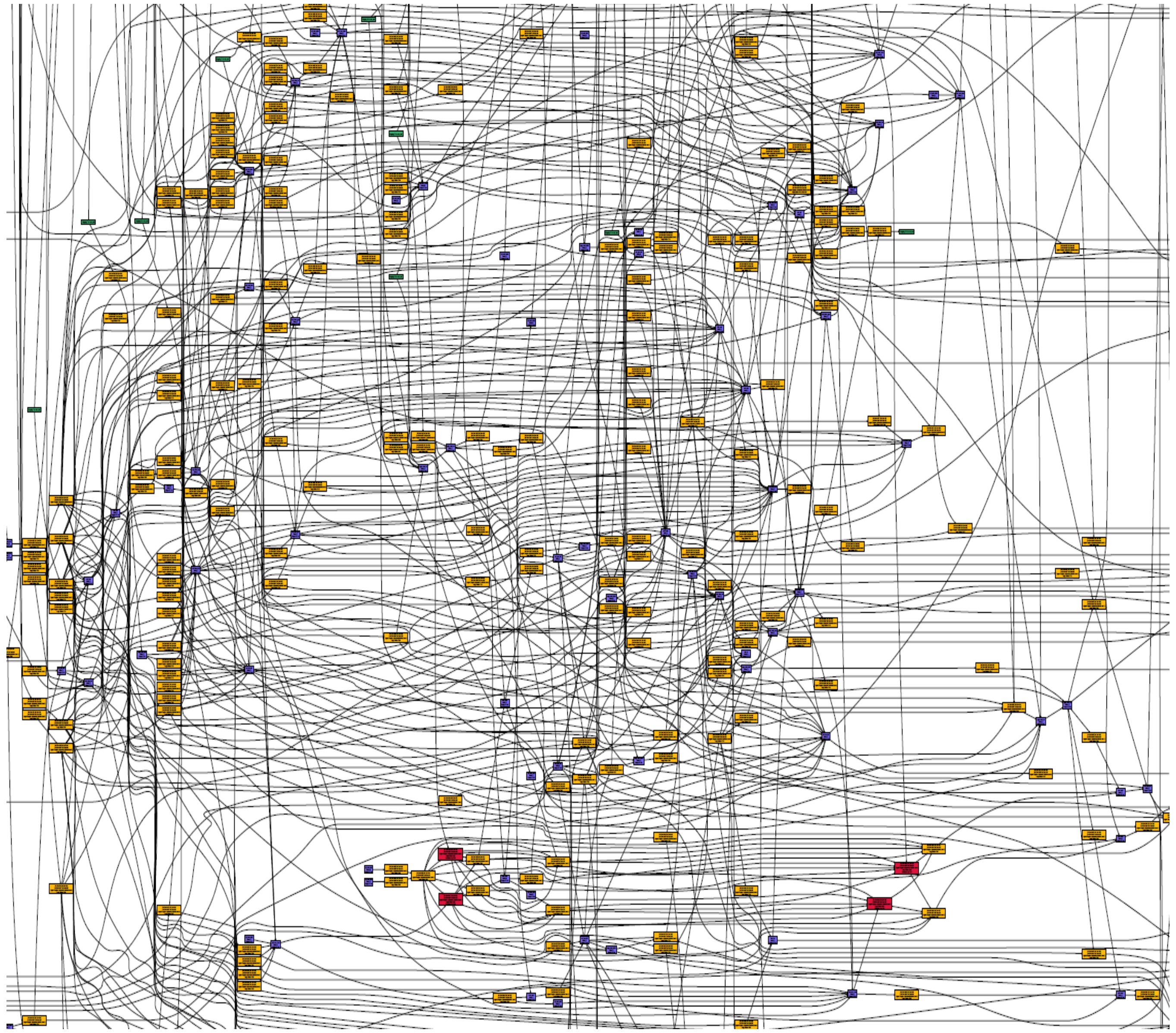
Operations can specify preferred data decomposition independently rather than working with a fixed decomposition

Copy data when needed; reformat data when profitable

Compose libraries even when they don’t agree on storage format

AUTOMATED ANALYSIS & SCHEDULING

Tasks graphs quickly grow too complex for manual scheduling

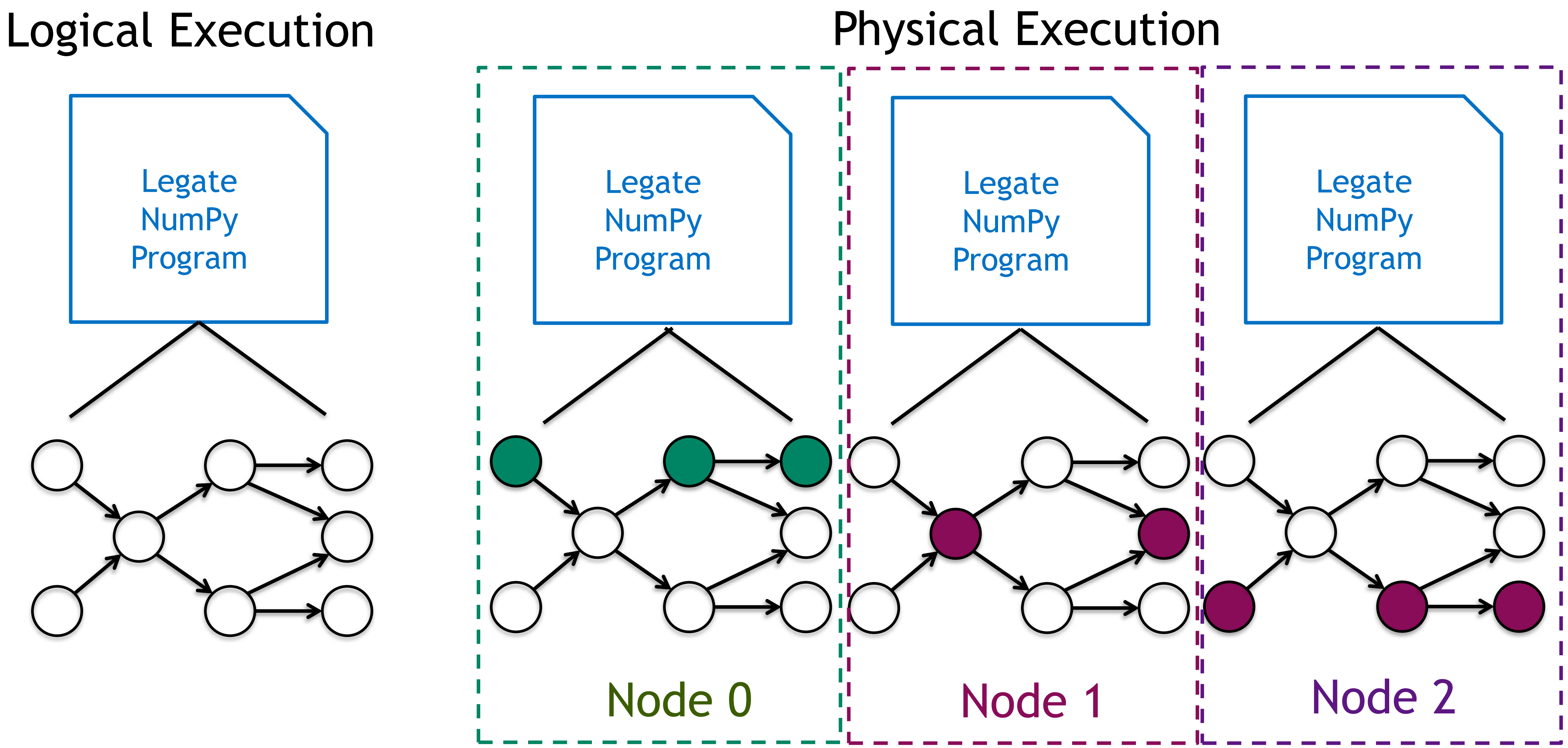


PENNANT mini-app task graph

Showing one time-step on one node

CONTROL REPLICATION

Avoid scaling bottleneck of having a single control thread



Transparently run N copies of your program

Each node performs 1/N-th of dependence analysis
No sequential bottleneck

Some minimal requirements for this to work

MAPPERS

Mapping and scheduling machine independent programs

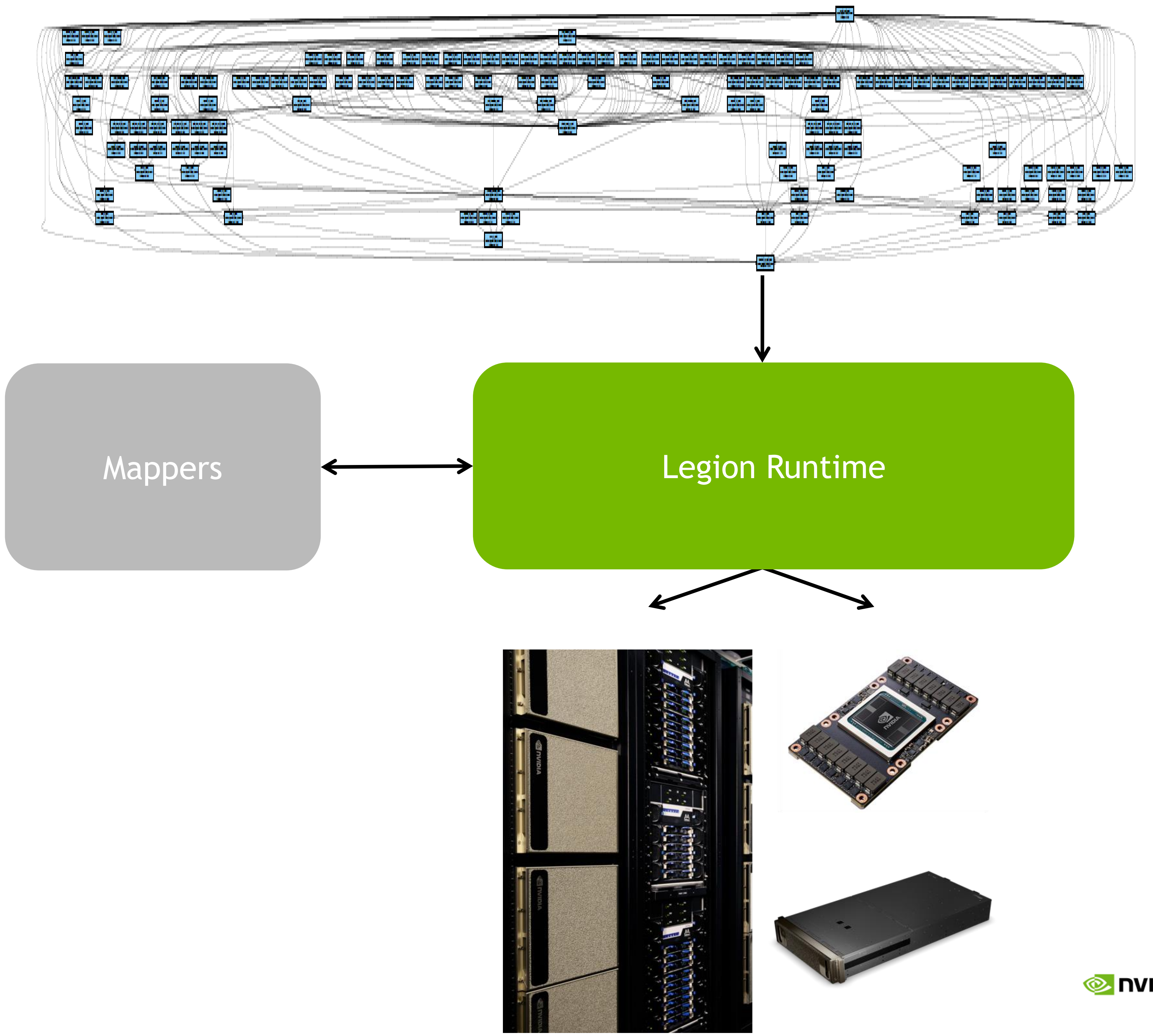
Legion programs are *machine-independent*

Mapping policies are *machine-dependent:*

- Where do tasks run? Which variant?
- Where is data placed?
- How should data be laid out?

Learnable Heuristics

Mapping decisions never change semantics



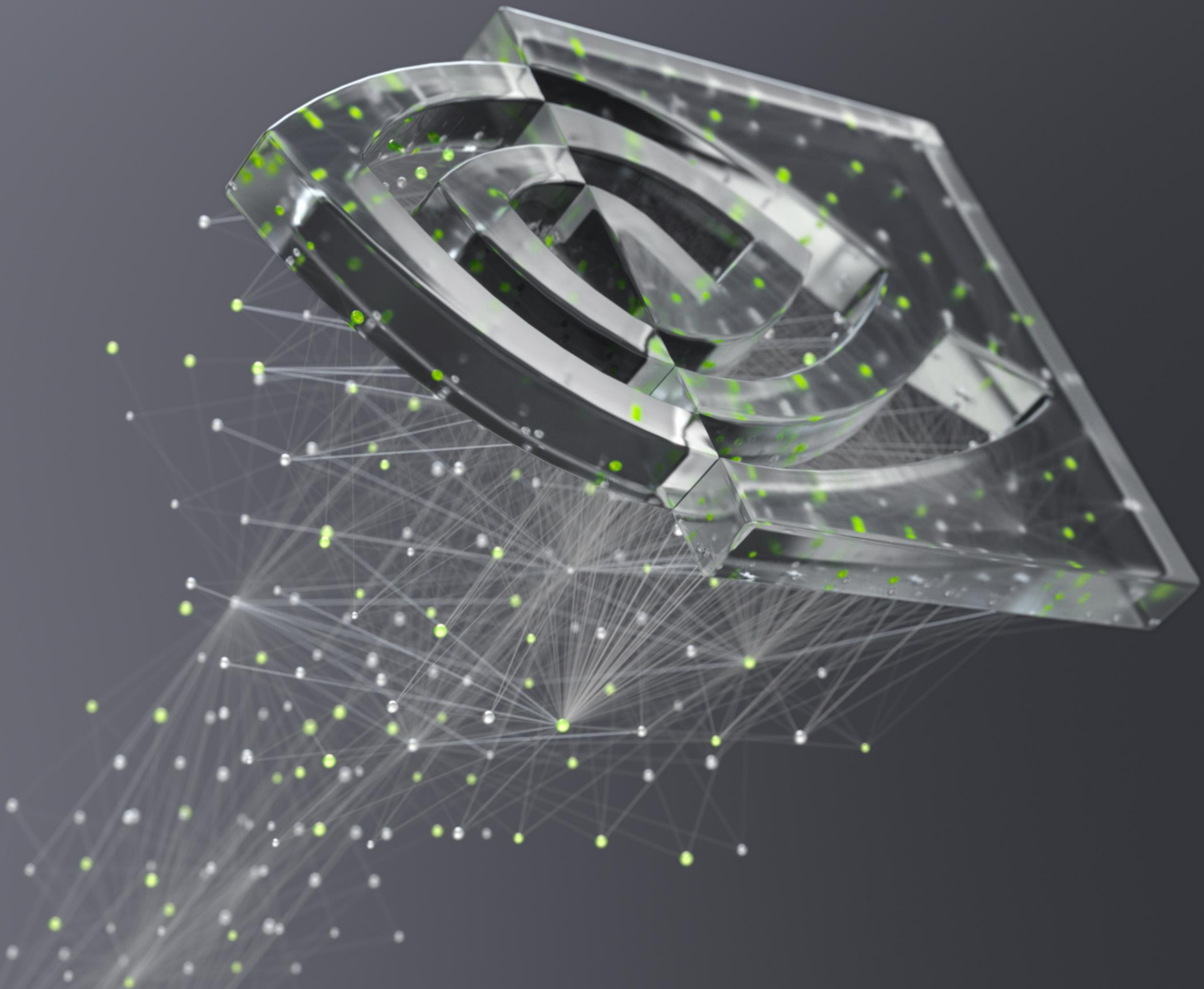
SUMMARY

Machine learning programs are awash in data

Programming systems can help efficiently organize this data

Delivering convenience + compositionality + performance

Data-centric task-based runtime systems are particularly well suited



NVIDIA