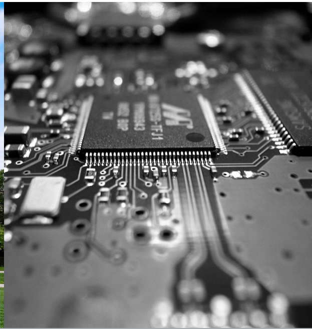
Adding Fast GPU Derived Datatype Handing to Existing MPIs

University of New Mexico CS Colloquium, May 5, 2021

Carl Pearson (Postdoc, Sandia National Labs)
Kun Wu, Wen-Mei Hwu, I-Hsin Chung, Jinjun Xiong



ILLINOIS
Electrical & Computer Engineering
GRAINGER COLLEGE OF ENGINEERING

Carl Pearson

- Postdoctoral Appointee, Center for Computing Research, Sandia National Labs
  - Multi-GPU Communication
  - Sparse matrix operations
- Ph.D from University of Illinois Electrical and Computer Engineering

cwpearson
cwpears@sandia.gov
carlpearson.net

Outline

- Stencil code
    - Domain Decomposition
- Stencil code and supercomputing
    - Processes, MPI, and GPUs
- Non-contiguous data
    - Where it comes from and why it matters
- TEMPI's approach to derived type handling
    - Translation
    - Canonicalization
    - Kernel Selection
- Some Performance Results
- How TEMPI works with MPI

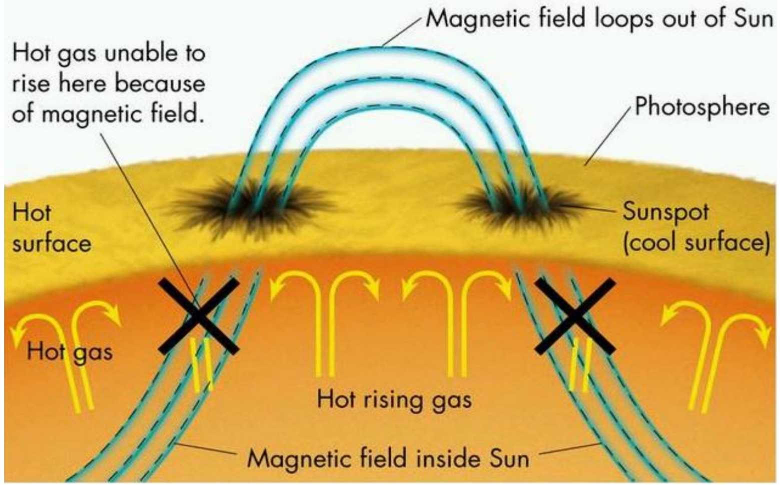
CS 375

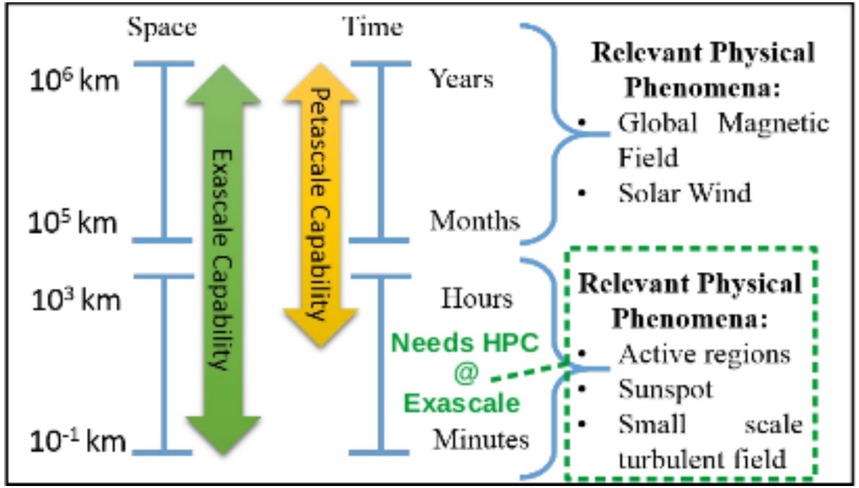
CS 341, 442, 471, 475, 481

CS 151, 241, 251

Astaroth[1] 3D Stellar Simulation Code

- Aalto Univ. / Univ. of Helsinki
- 2E+11 gridpoints on 3072 GPUS (256^3 / GPU)
- First direct numerical simulation of relevant high-resolution phenomena



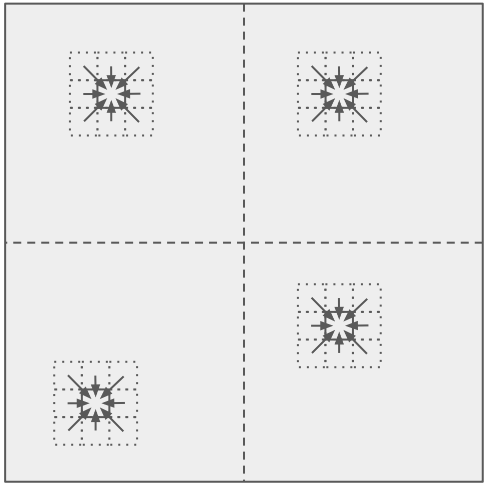


Figures from a talk by Anjum, O. 2017

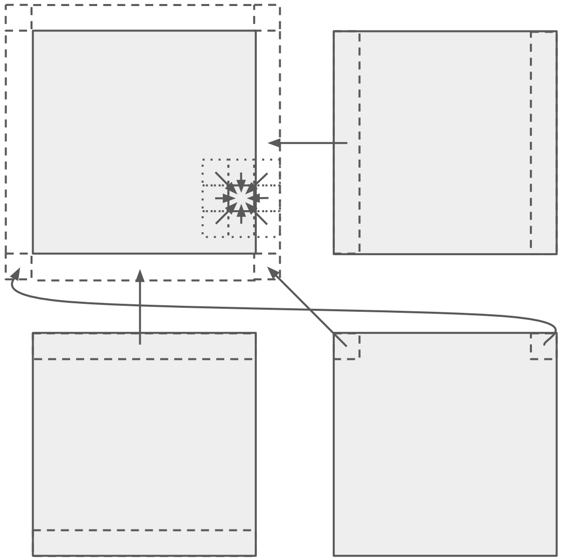
[1] Pekkilä, J., Väisälä, M. S., Käpylä, M. J., Käpylä, P. J., & Anjum, O. (2017). Methods for compressible fluid simulation on GPUs using high-order finite differences. *Computer Physics Communications*, *217*, 11-22.

Distributed Stencil

- Inside of sun as a grid of points
  - Each point has associated physical properties
- Value at next timestep computed from neighbor's current values



domain decomposition of grid



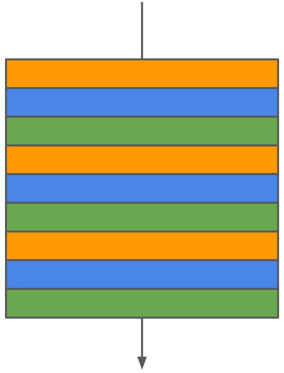
Sub-grids with boundary communication

Parallel Computing with Processes

- Program code
- Program state
- Isolation
  - Each process thinks it is alone on the computer
  - Errors in one process do not affect others

Parallel Computing with Processes

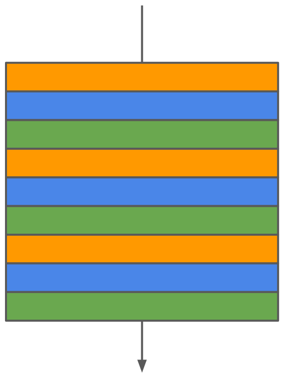
- Program code
- Program state
- Isolation
  - Each process thinks it is alone on the computer
  - Errors in one process do not affect others



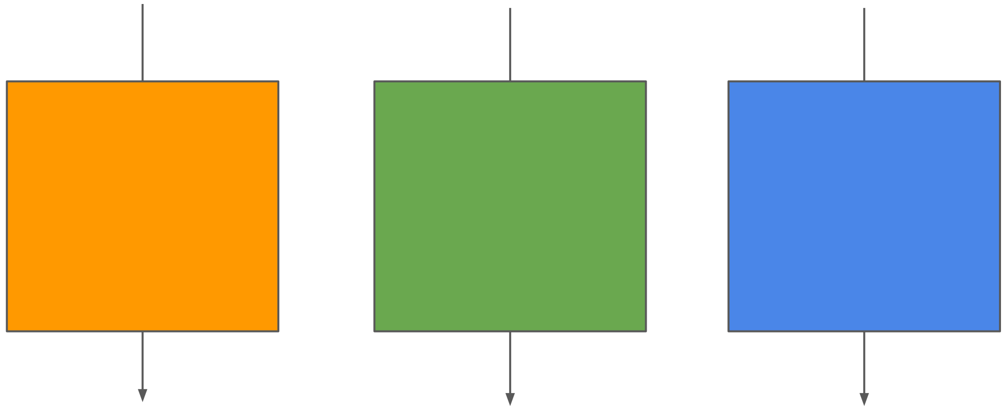
One CPU

Parallel Computing with Processes

- Program code
- Program state
- Isolation
  - Each process thinks it is alone on the computer
  - Errors in one process do not affect others



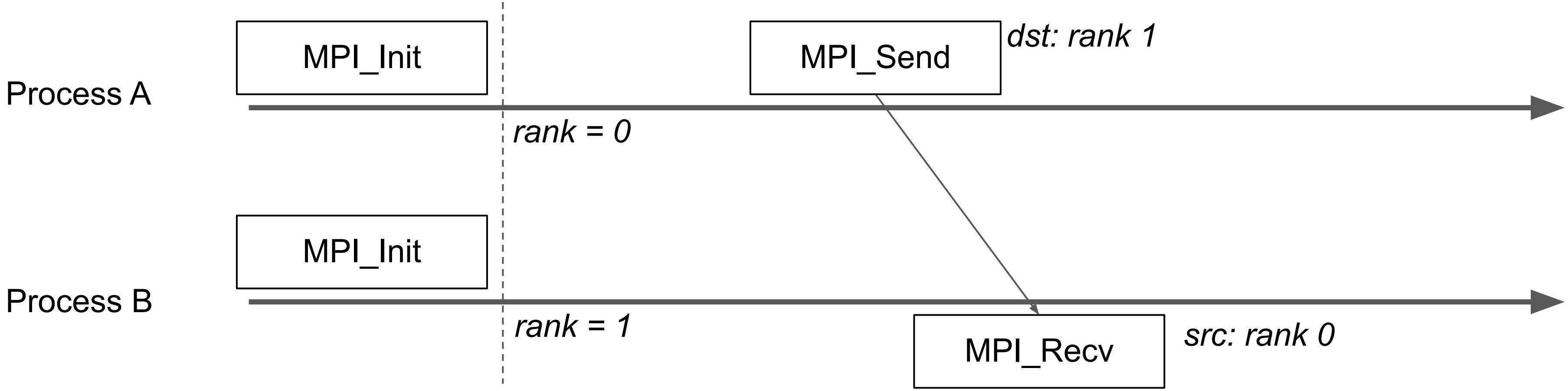
One CPU



Three CPUs (more done in the same amount of time)

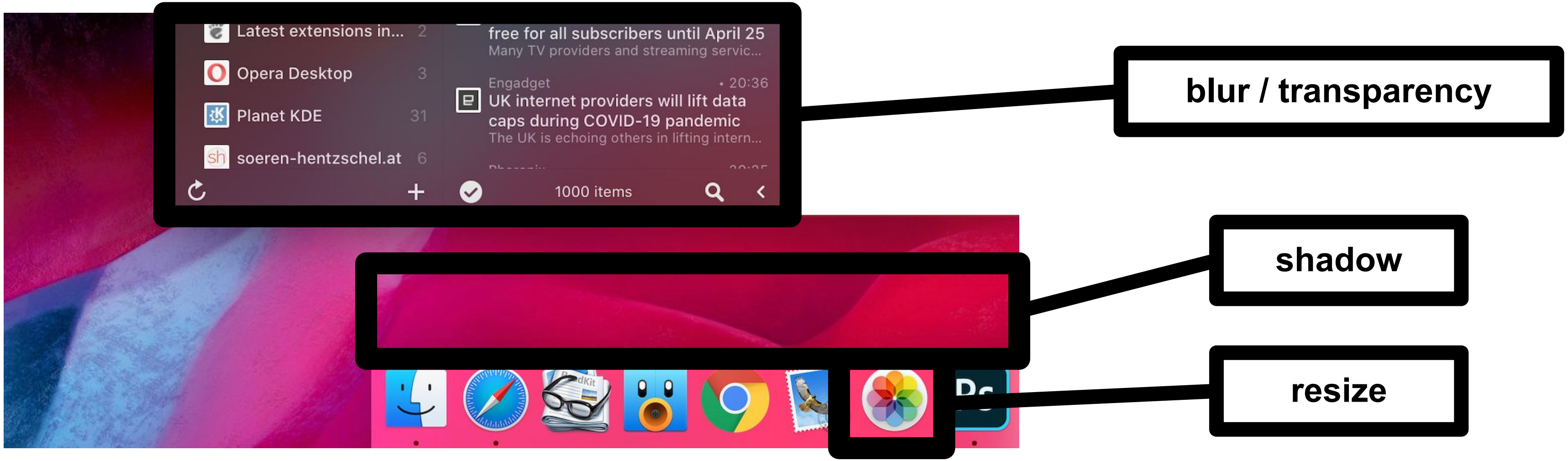
MPI - Message Passing Interface

- Selectively break isolation between processes
- Coordination: “have you given me updated boundary values?”
- Communication: “here are updated boundary values for you.”



GPU Computing (Graphics Processing Unit)

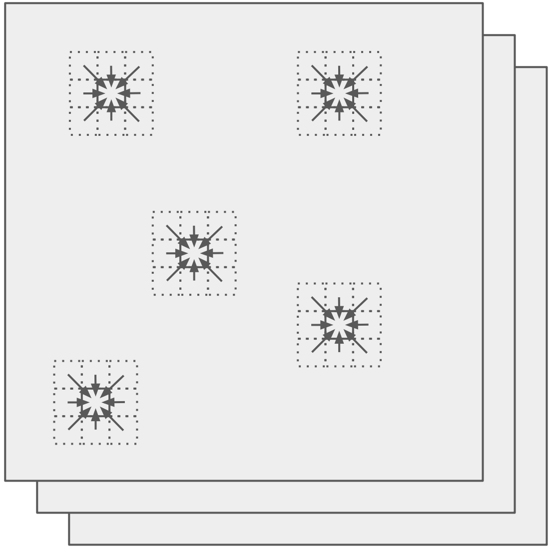
- Special hardware to render displayed images faster
  - Quickly apply same operation to many pixels

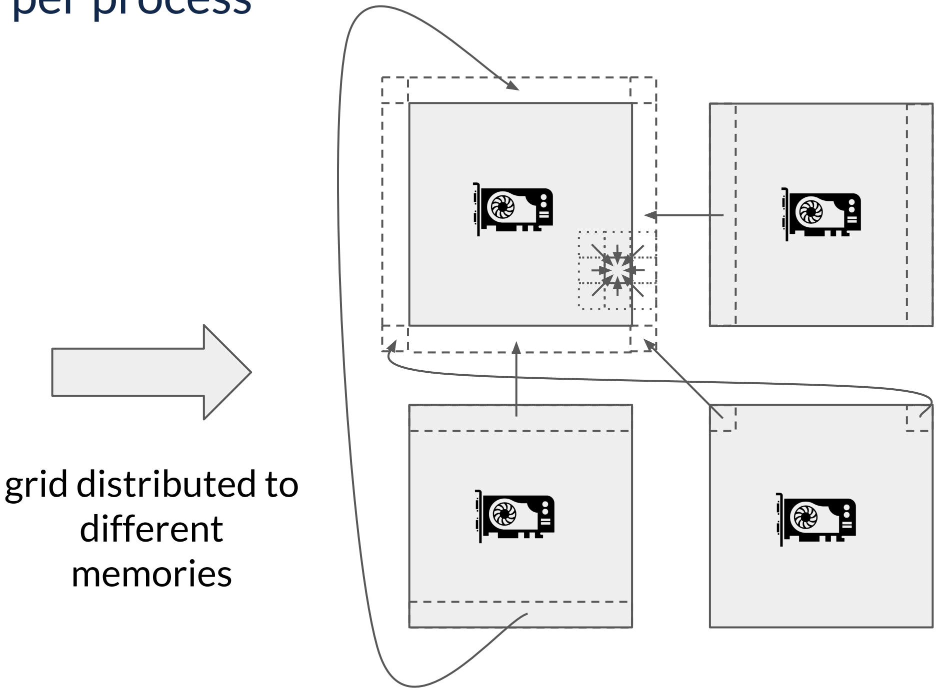


- Turns out, many scientific codes have similar properties
- Blur: create new pixel value based on neighbors
- Stencil: create new gridpoint value based on neighbors

Distributed Stencil

- one sub-grid per GPU, one GPU per process



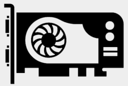


Contiguous & Non-contiguous Data

- “row-major” storage

X

0x0	0x8	. . .
0x0+1X		
0x0+2X		
0x0+3X		
0x0+4X		



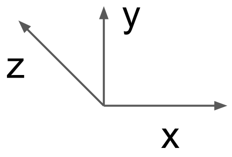
“top edge” - contiguous

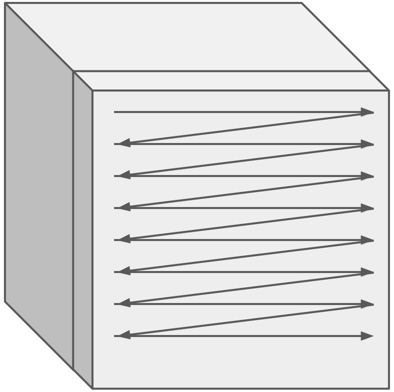
0x0	0x8	0x16	. . .

“left edge” - non-contiguous

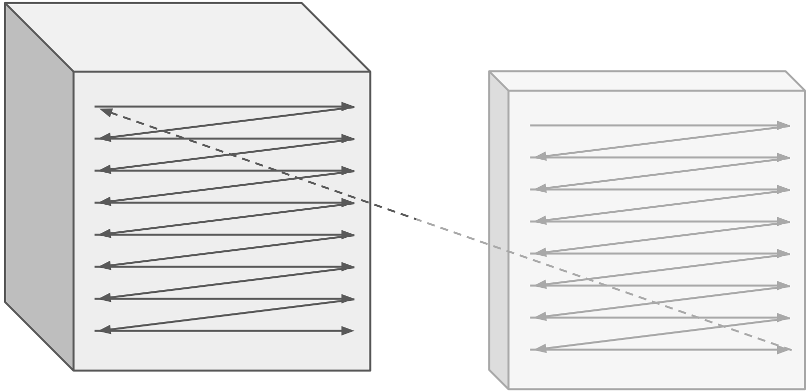
0x0		0x0+X	. . .

“Row-major” Storage in 3D





z = 0 plane

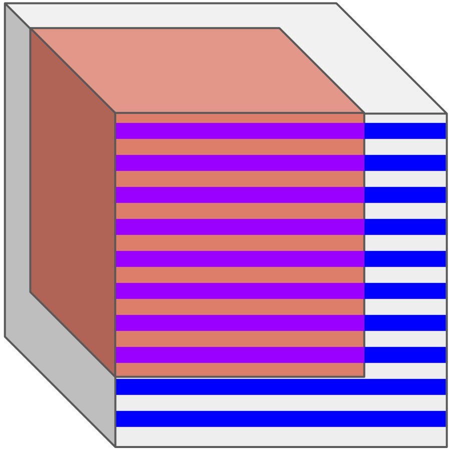


z = 1 plane

z = 0 plane

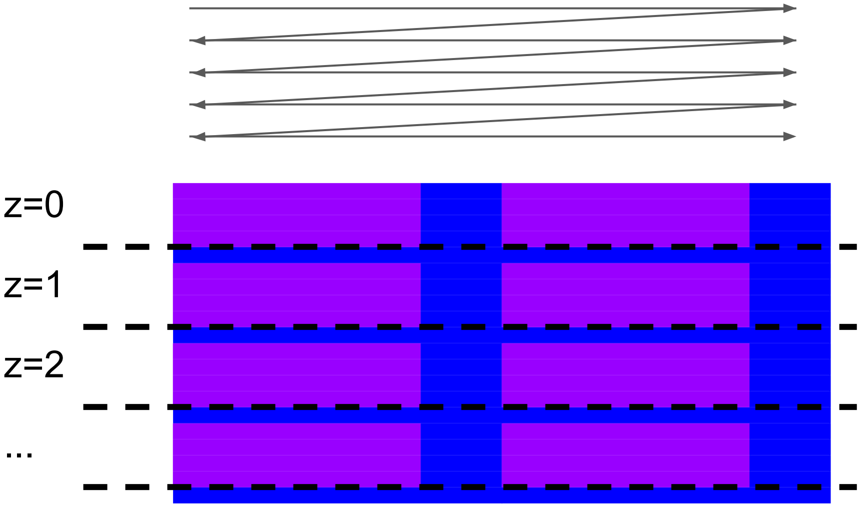
3D Non-contiguous Example

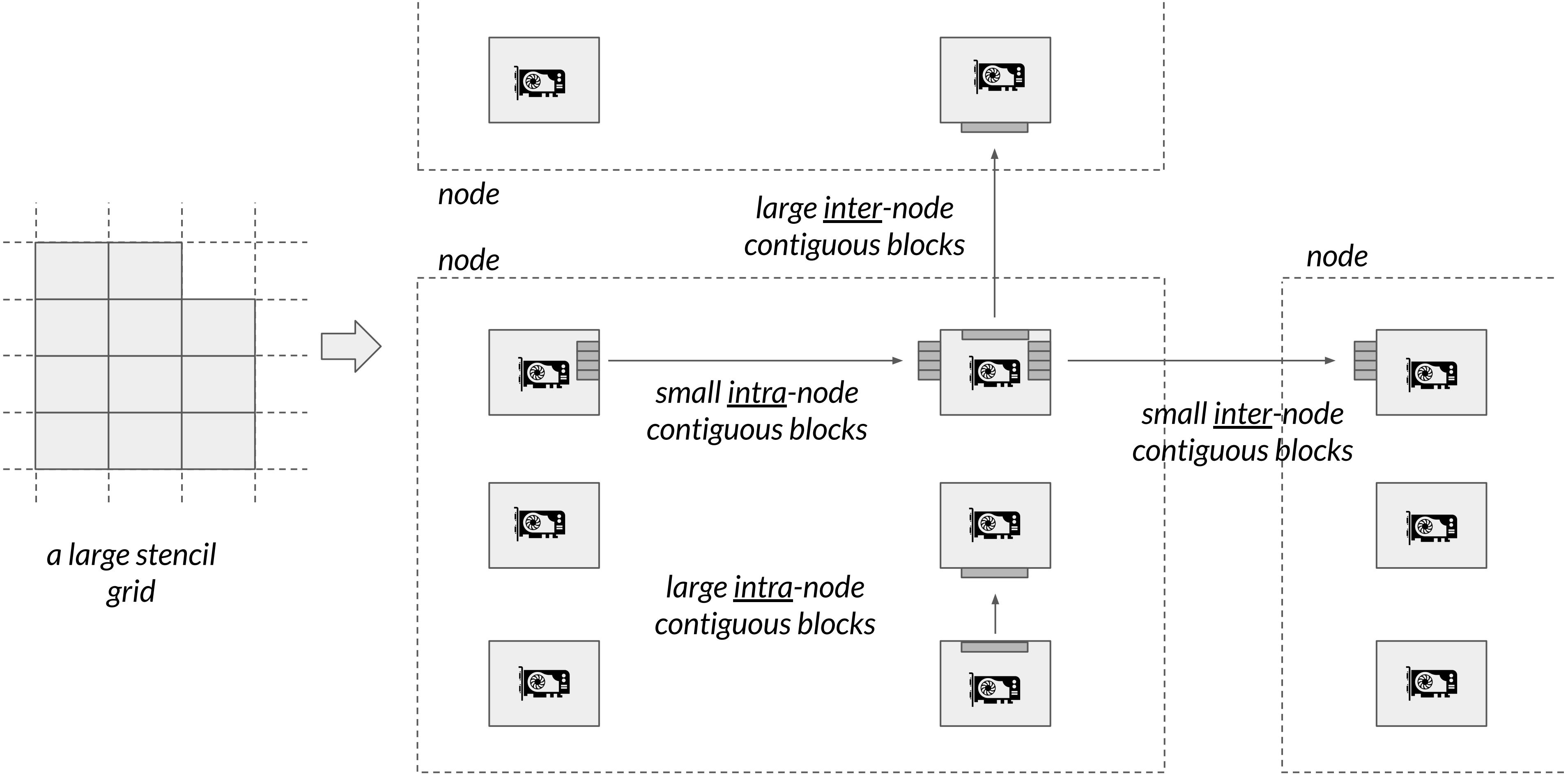
3D Region



(z = 0)

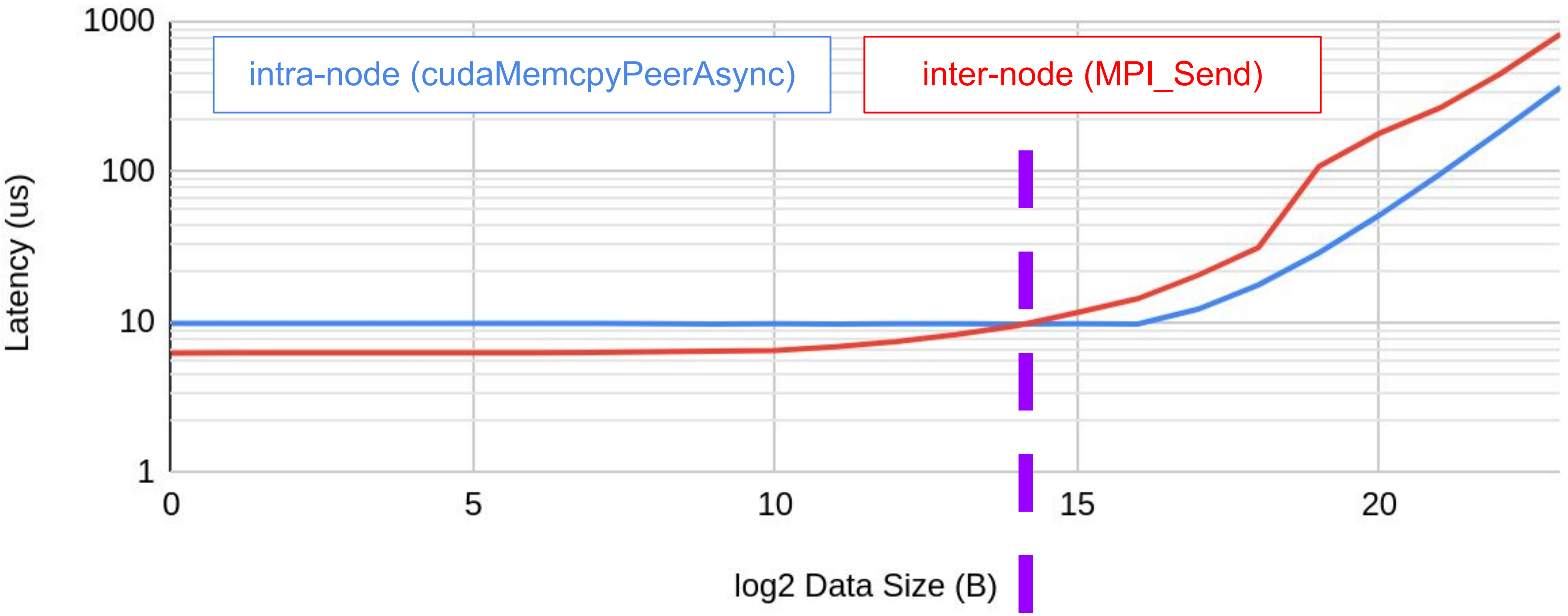
(storage order)





a large stencil grid
node
node
large inter-node contiguous blocks
node
small intra-node contiguous blocks
small inter-node contiguous blocks
large intra-node contiguous blocks

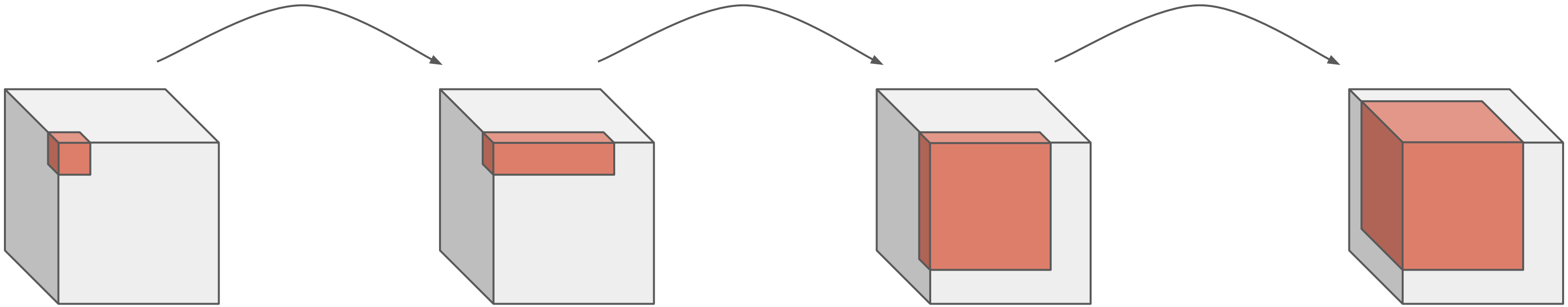
Latency vs Contiguous Size



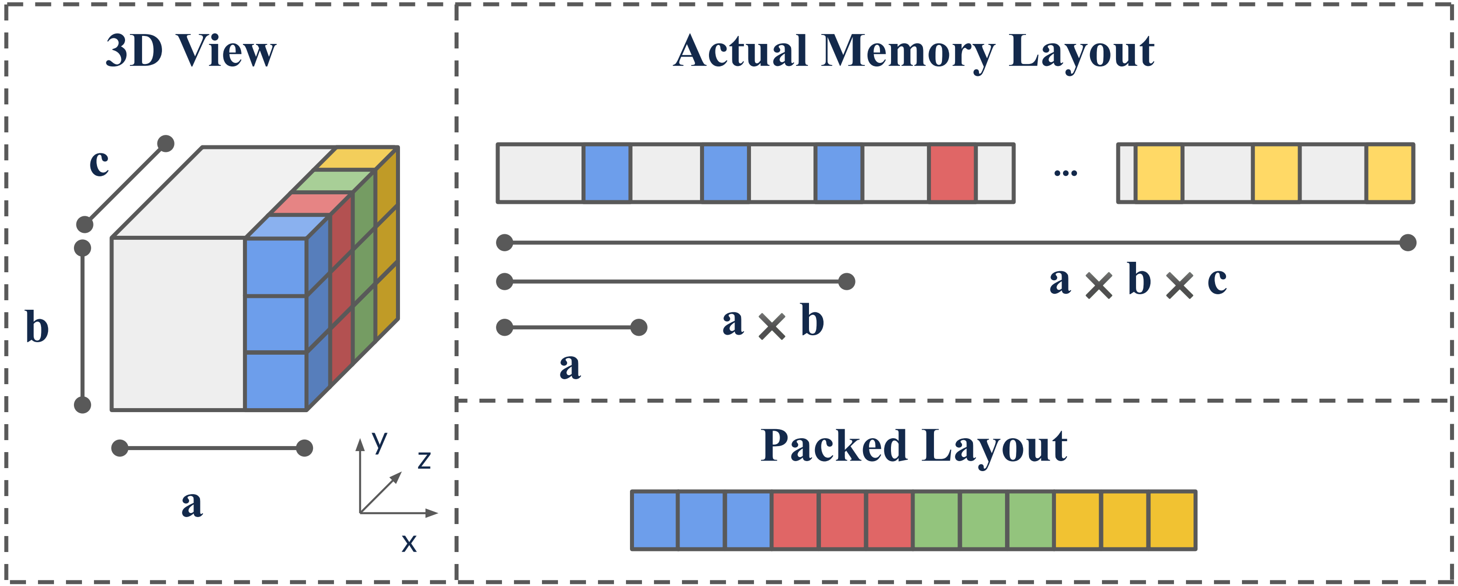
small blocks - relatively slow (latency)

large blocks - relatively fast (bandwidth)

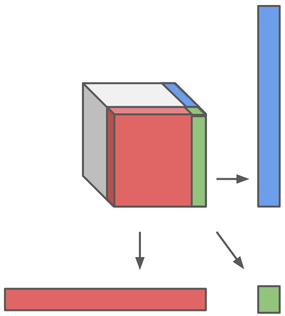
MPI Derived Datatypes



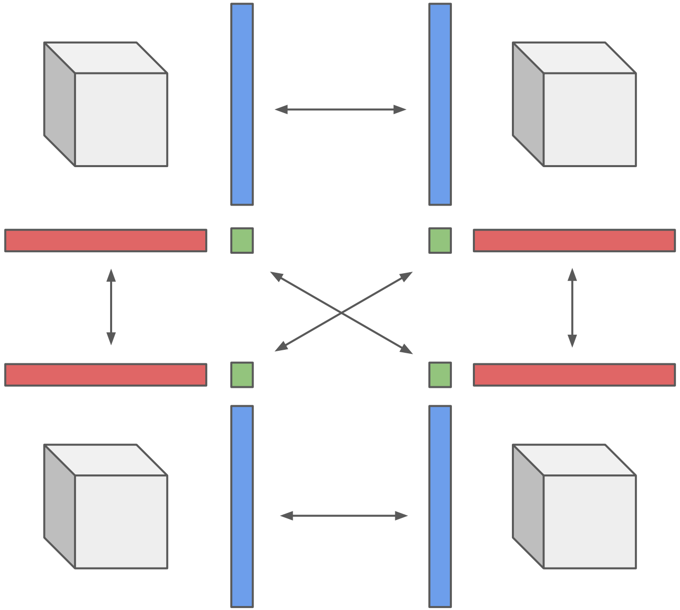
Many Small Blocks into Few Large Blocks



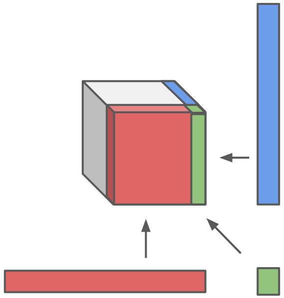
Pack / Alltoallv / Unpack



MPI_Packs

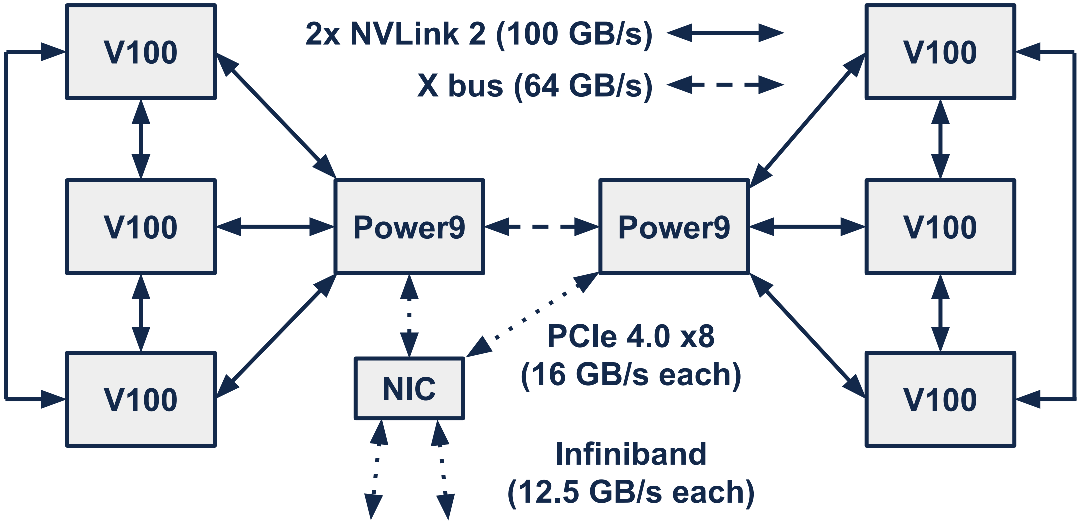


MPI_Neighbor_alltoallv



MPI_Unpacks

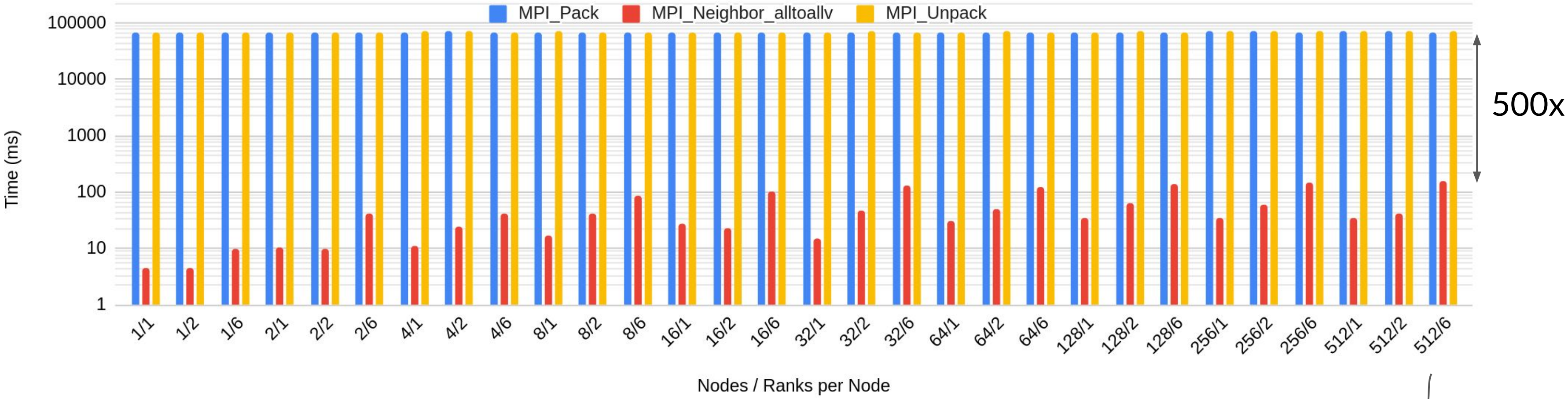
OLCF Summit



Summit Node
(bidirectional bandwidth)

The Problem (1/2)

- Halo exchange with MPI derived types

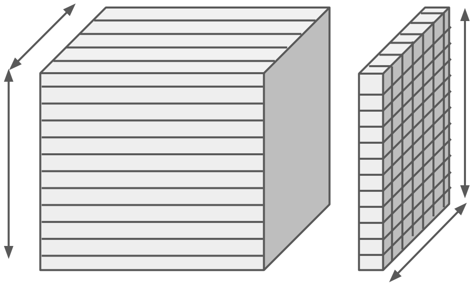


- 73.7 MiB/rank
  - MPI_Neighbor_alltoallv = ~500 MB/s/rank
  - MPI_Pack / MPI_Unpack = **~1 MB/s**

The Problem (2/2)

- Most of the “non-contiguousness” is in one dimension
- 3,145,728 contiguous blocks (~20us per block)
- one cudaMemcpyAsync per block

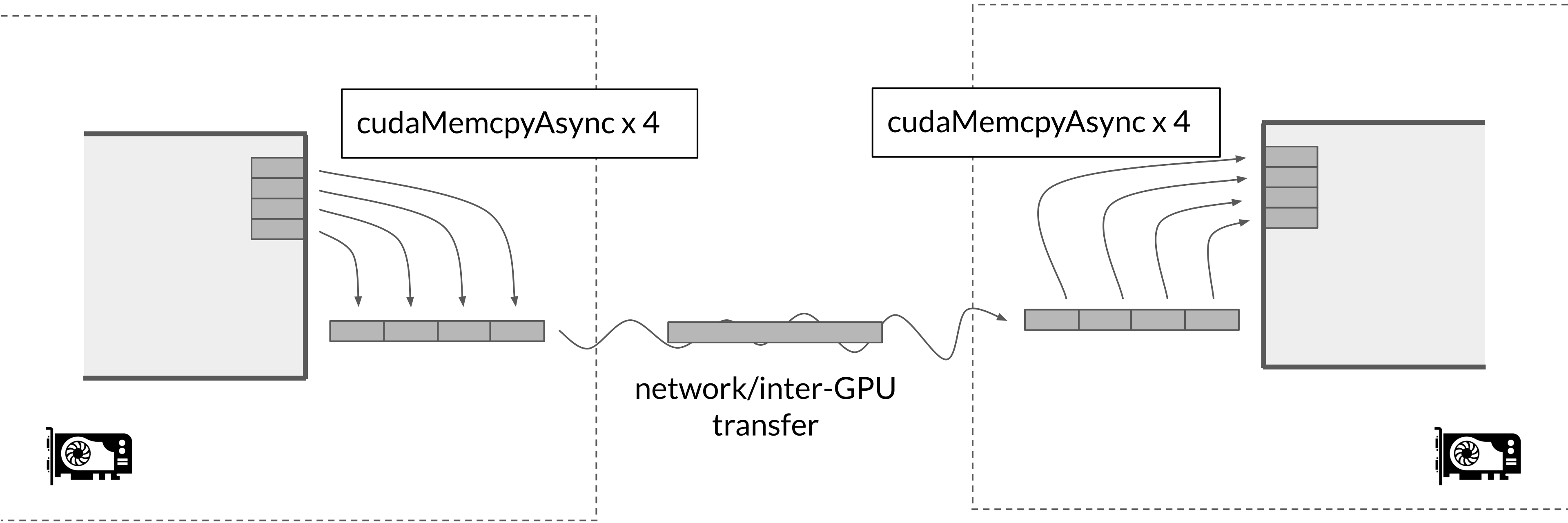
256 contiguous blocks
on each of these faces



256 x 256
contiguous blocks
on this face

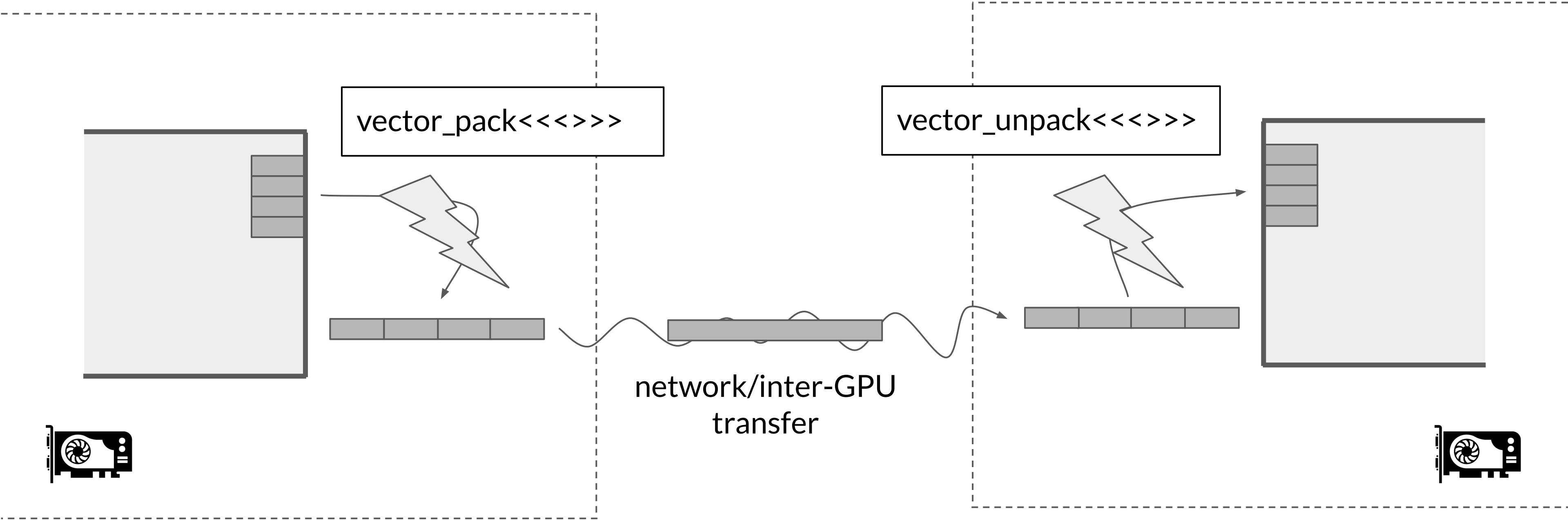
(recall that halo space breaks up otherwise contiguous directions)

MPI_Send (SpectrumMPI)

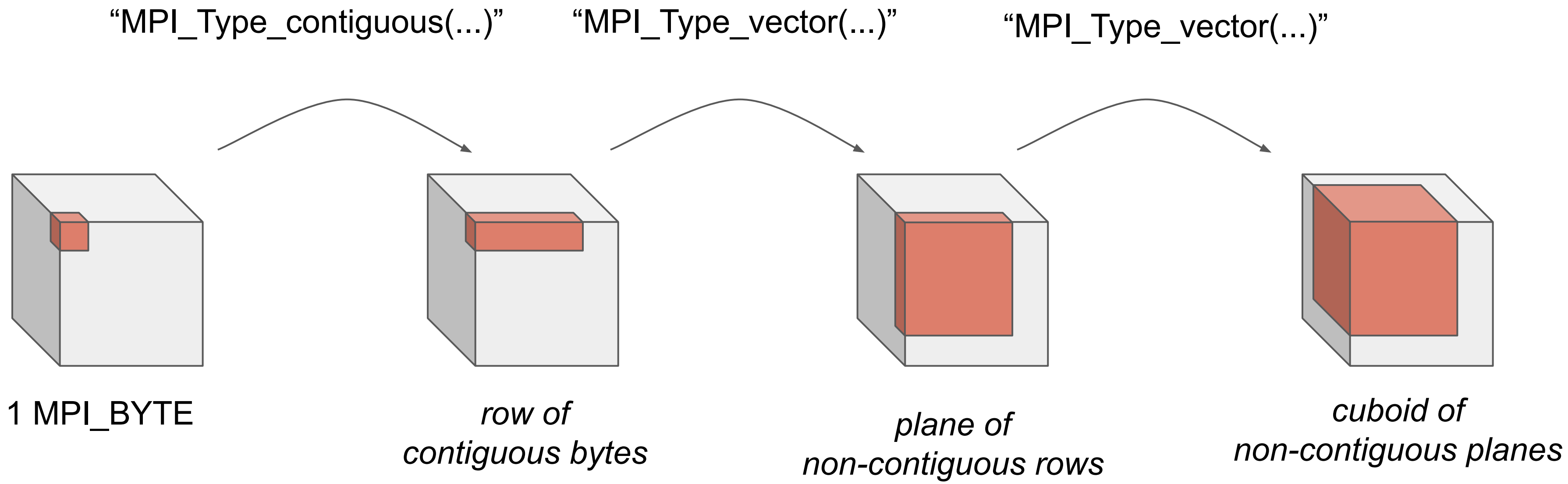


A Solution

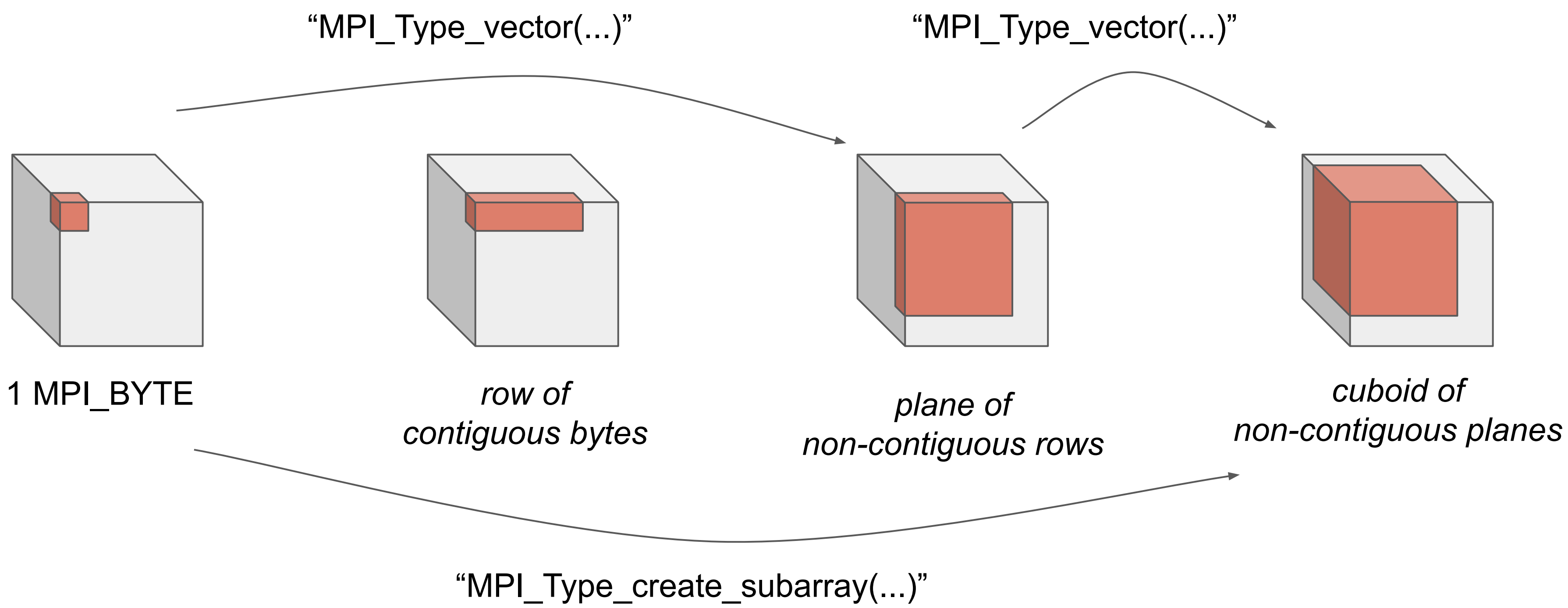
- GPU Kernels to pack non-contiguous data



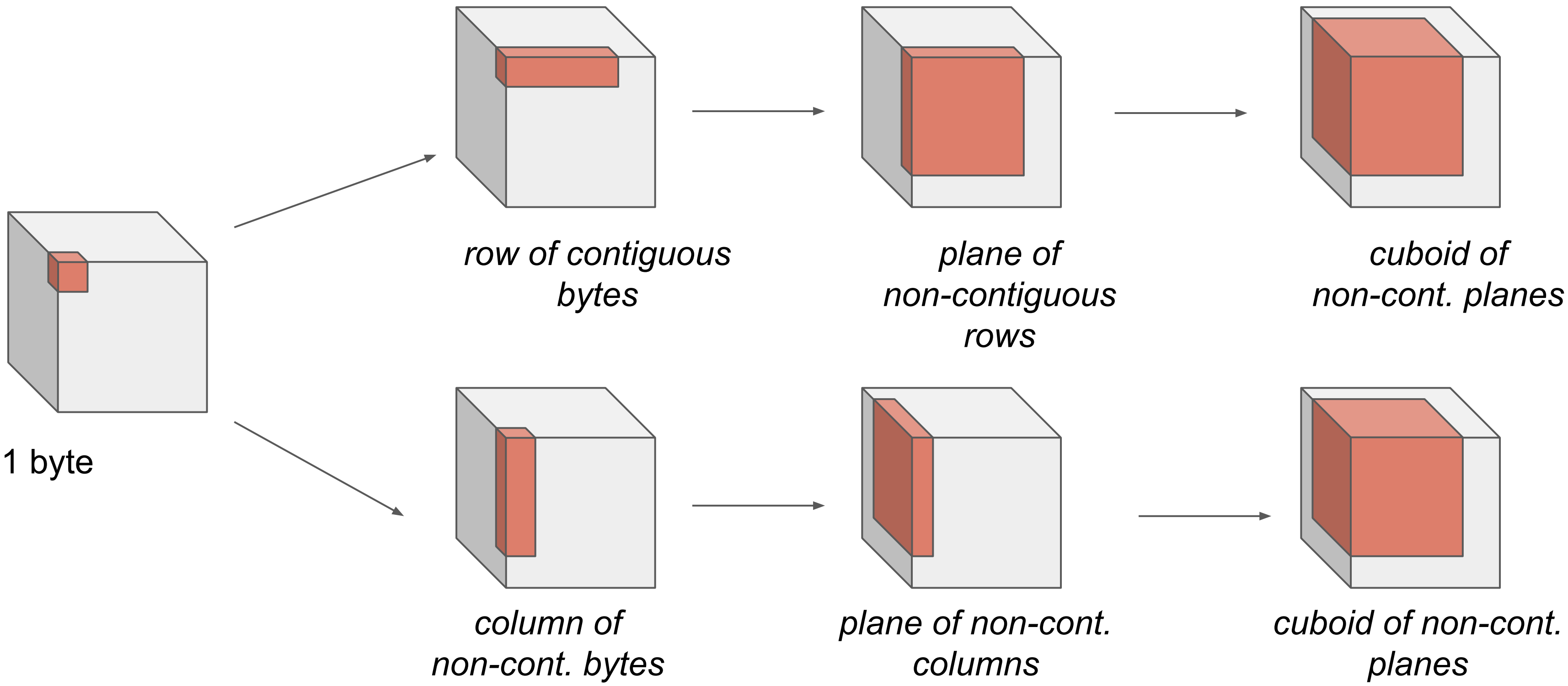
MPI Derived Datatypes: Revisited



MPI Derived Datatypes: Revisited



MPI Derived Datatypes: Revisited



TEMPI Datatype Handling

MPI_Type_commit()

↓

Translation
Convert MPI Derived Datatype into internal representation (IR)

↓

Canonicalization
Convert semantically-equivalent IR to simplified form

↓

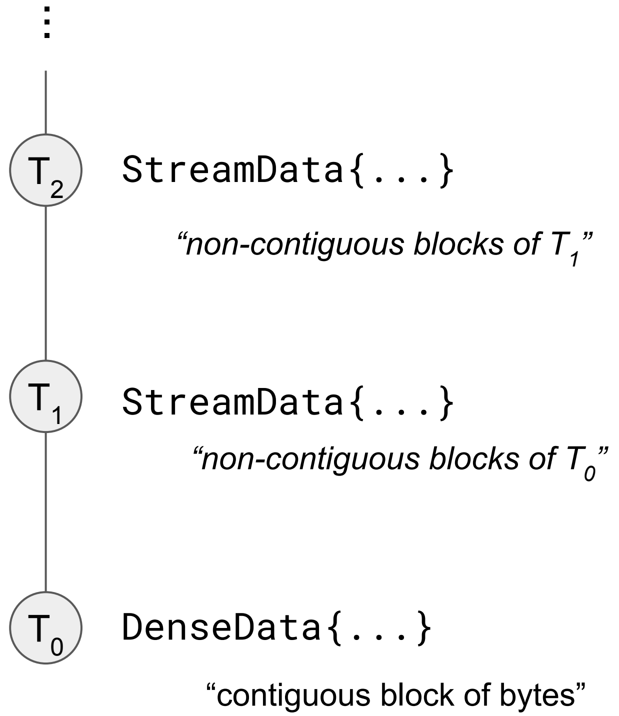
Kernel Selection
Choose packing/unpacking kernel for future operations

IR - “Internal Representation”

```
StreamData {
  integer offset; // offset (B) of the first element
  integer stride; // pitch (B) between element
  integer count;  // number of elements
}

DenseData {
  integer offset; // offset (B) of the first byte
  integer extent; // number of bytes
}
```

Hierarchy of StreamData, rooted at DenseData



Examples

TEMPI Datatype Handling

Translation
Convert MPI Derived Datatype into internal representation (IR)

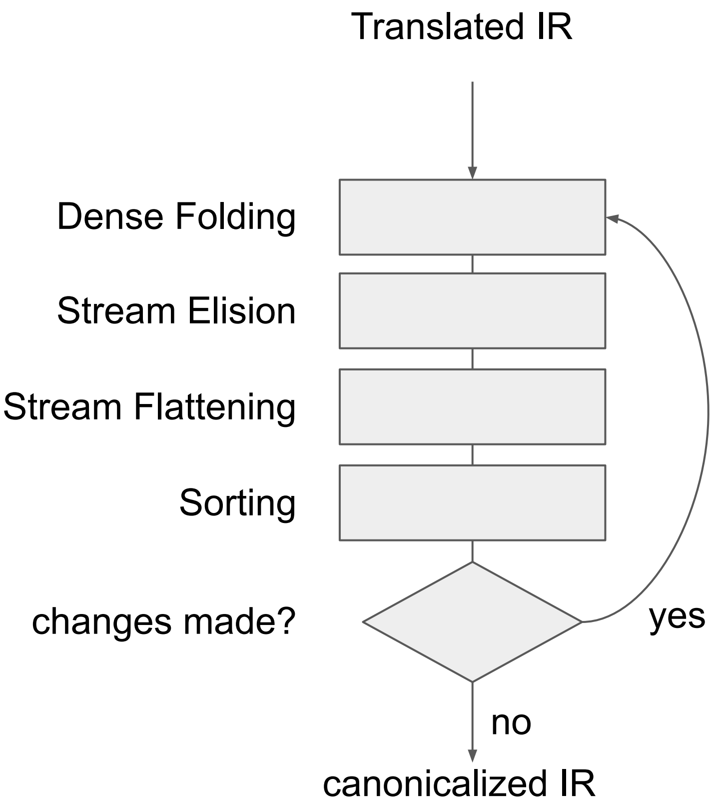
↓

Canonicalization
Convert semantically-equivalent IR to simplified form

↓

Kernel Selection
Choose packing/unpacking kernel for IR

Canonicalization



Canonicalization: Stream Elision

An MPI vector will commonly have a block of 1 child element

...

StreamData{offset:X, count:5, stride:4}

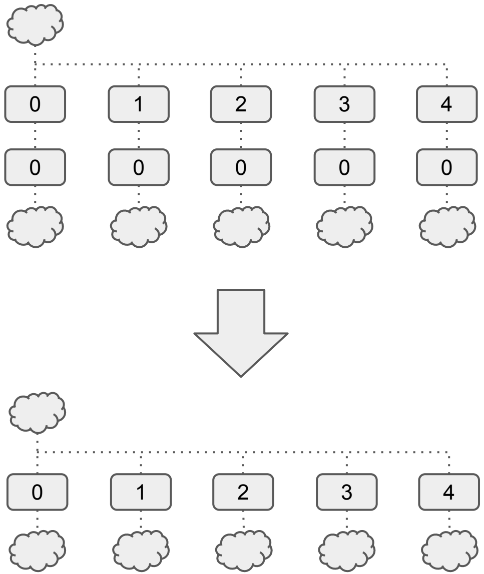
StreamData{offset:Y, count:1, stride:_}

...

...

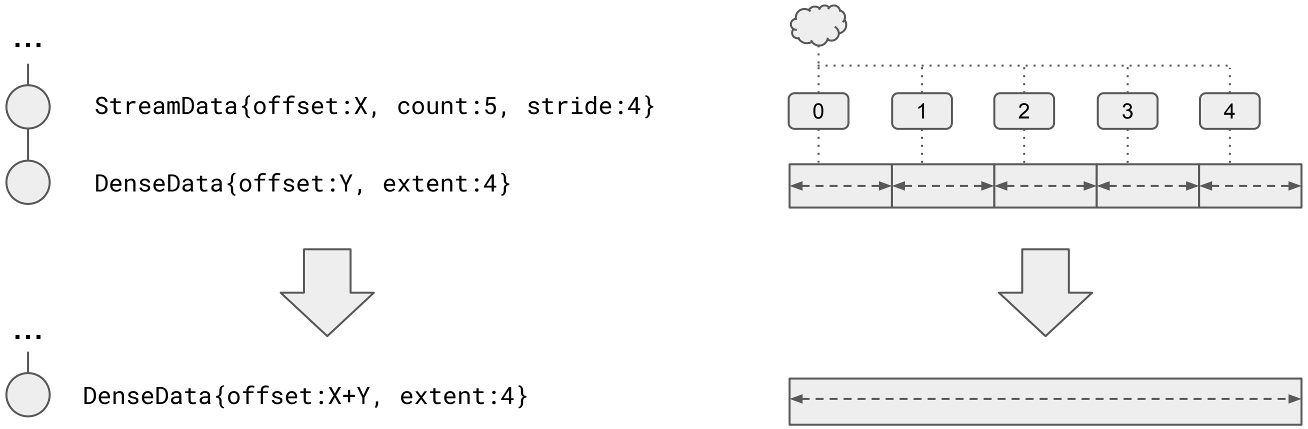
StreamData{offset:X+Y, count:5, stride:4}

...



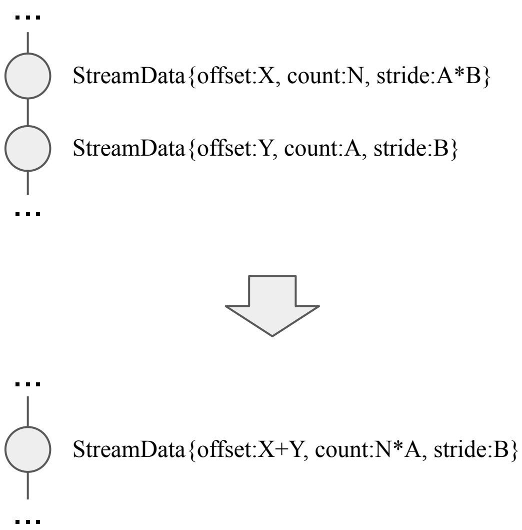
Canonicalization: Dense Folding

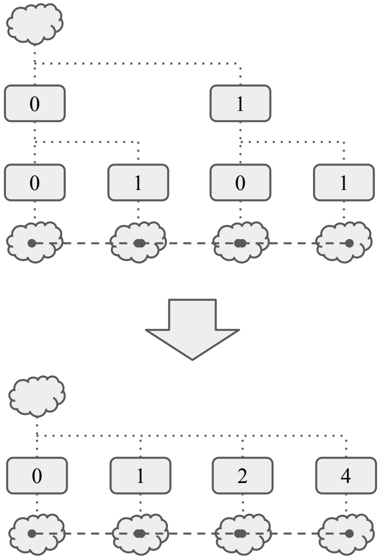
When a stream is actually multiple dense elements
A parent type of an MPI named type



Canonicalization: Stream Flattening

When two streams of three elements is one stream of six elements





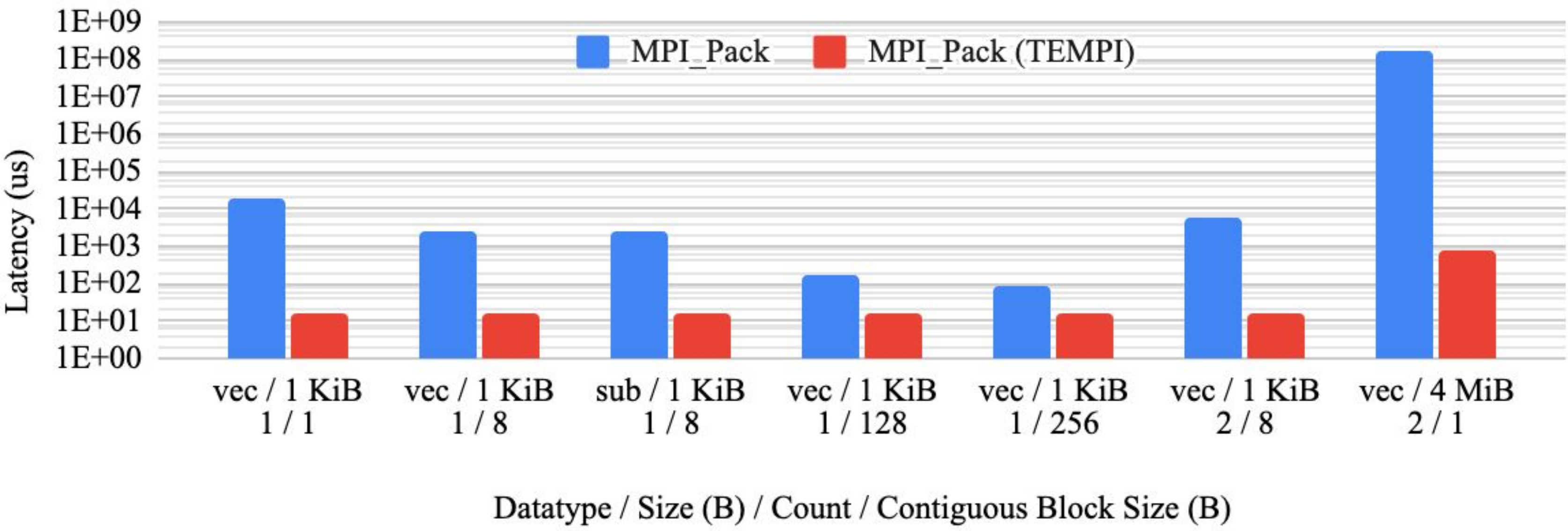
TEMPI Datatype Handling

Translation
Convert MPI Derived Datatype into internal representation (IR)

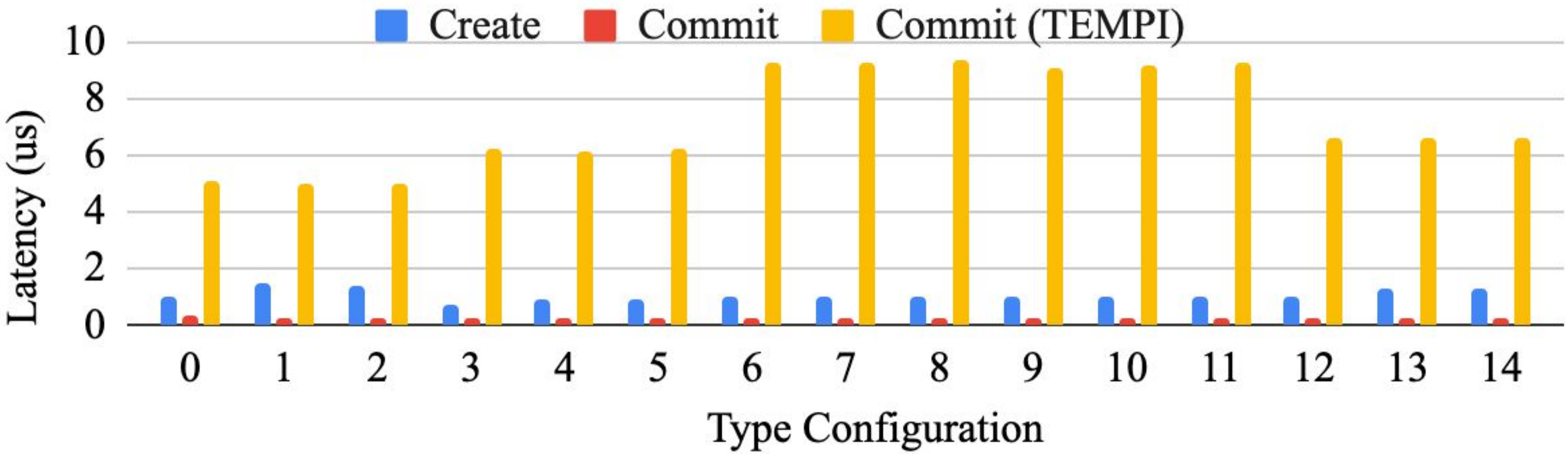
Canonicalization
Convert semantically-equivalent IR to simplified form

Kernel Selection
Choose packing/unpacking kernel for IR

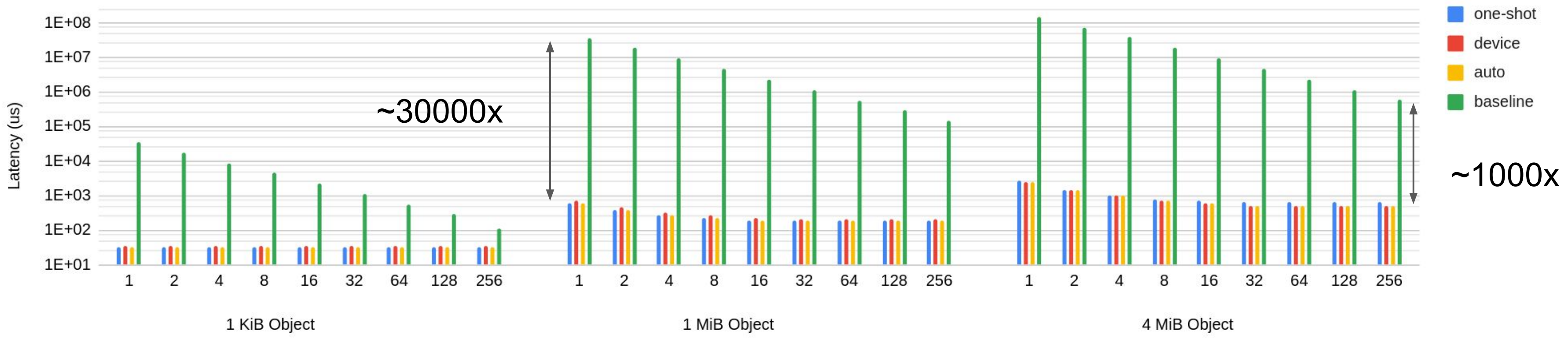
MPI_Pack Results



MPI_Type_commit Time



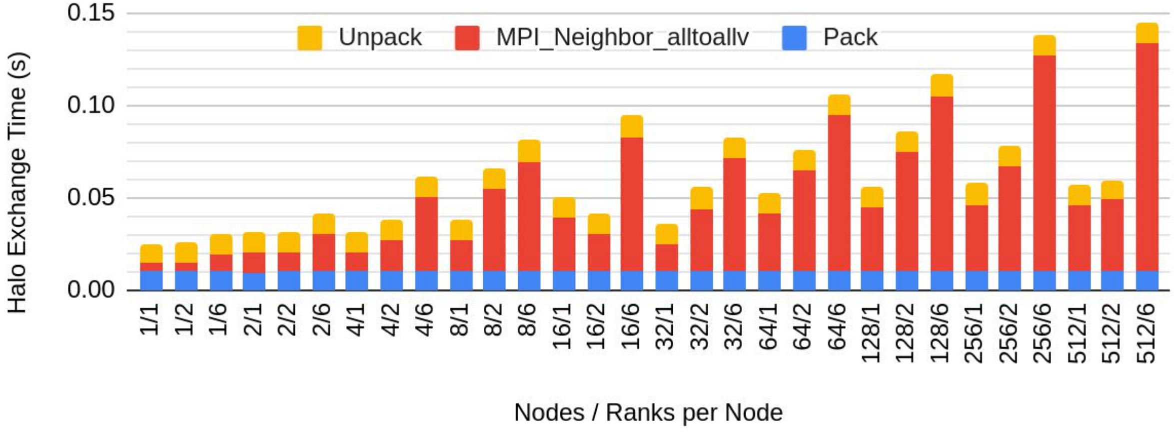
MPI_Send / MPI_Recv



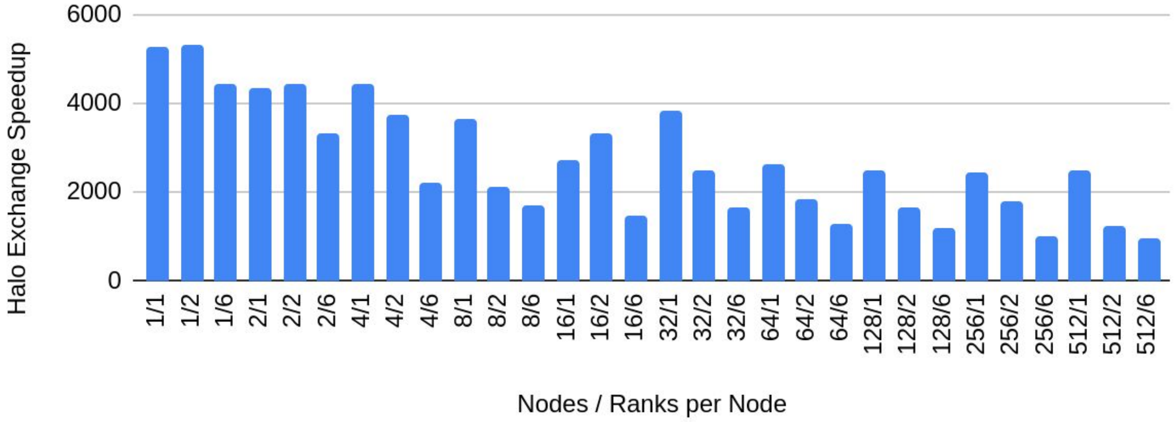
MPI_Send/Recv Latency for 2D objects with different block sizes

Halo Exchange

latency



speedup



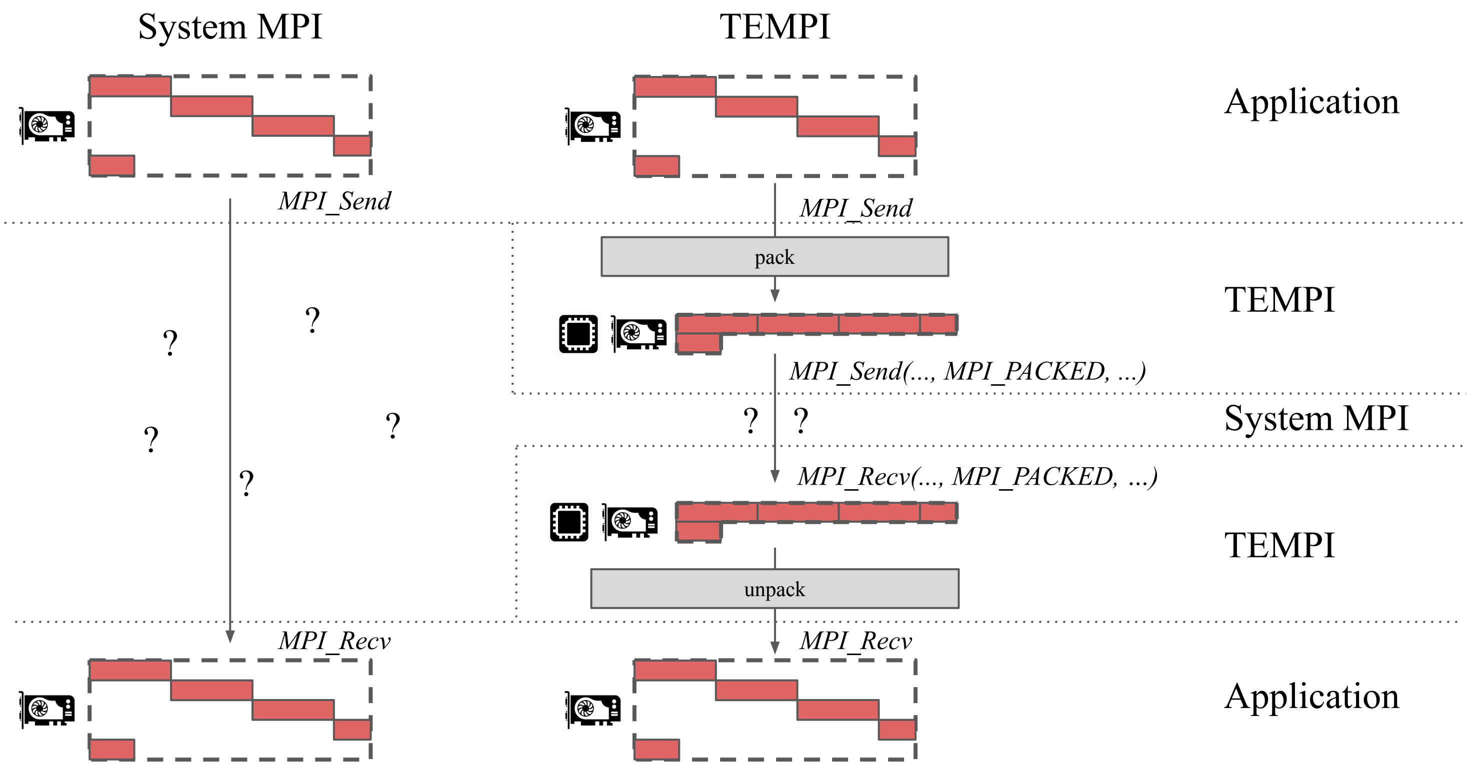
A Practical Challenge

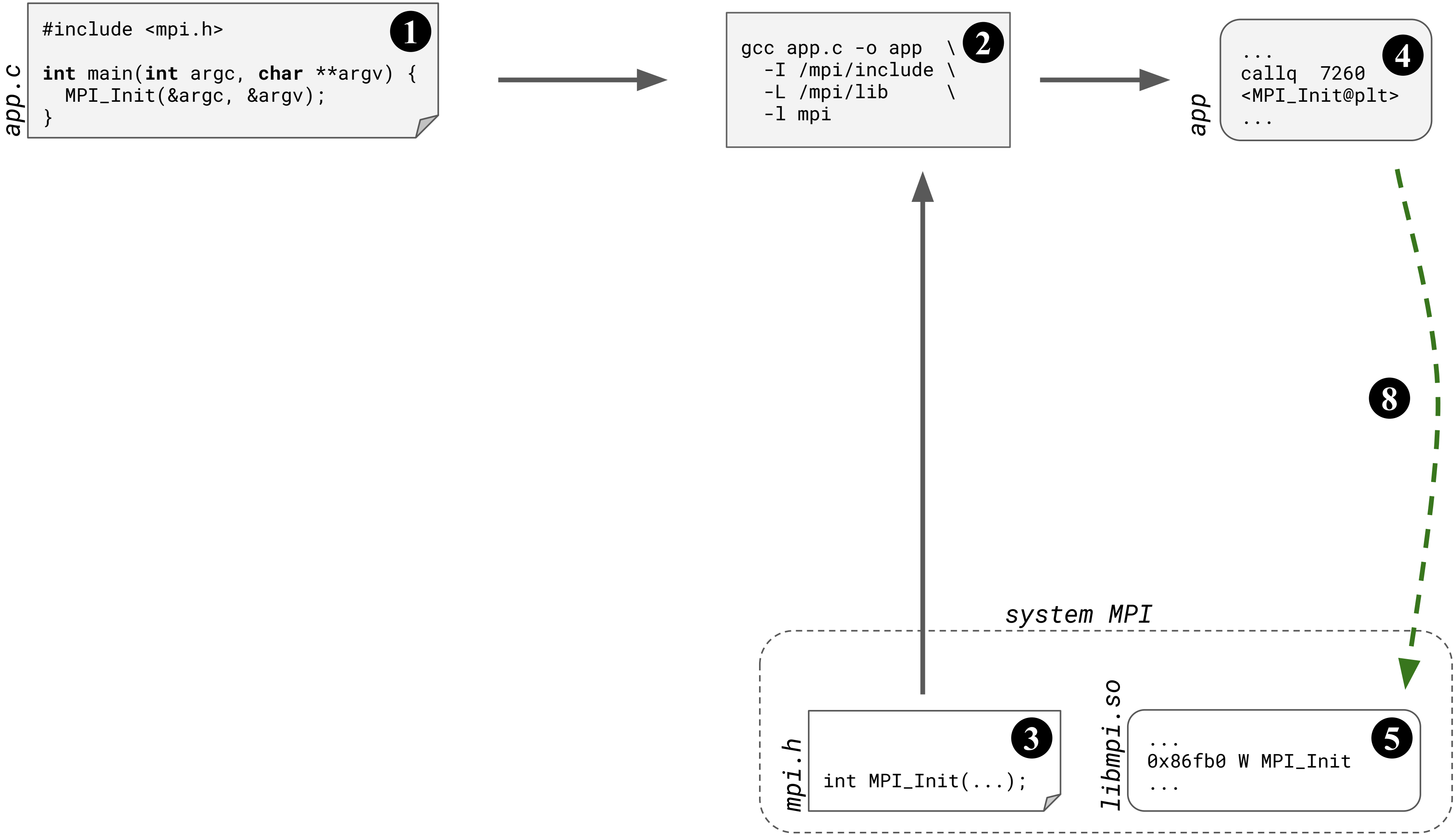
- Large-scale systems are tightly controlled
  - Can't just make whatever changes you want
- Usually one MPI (or maybe two) are deployed on the system
  - Rarely bugfixed (if ever)
  - Even more rarely are new features added
- Difficult or impossible to make experimental modifications

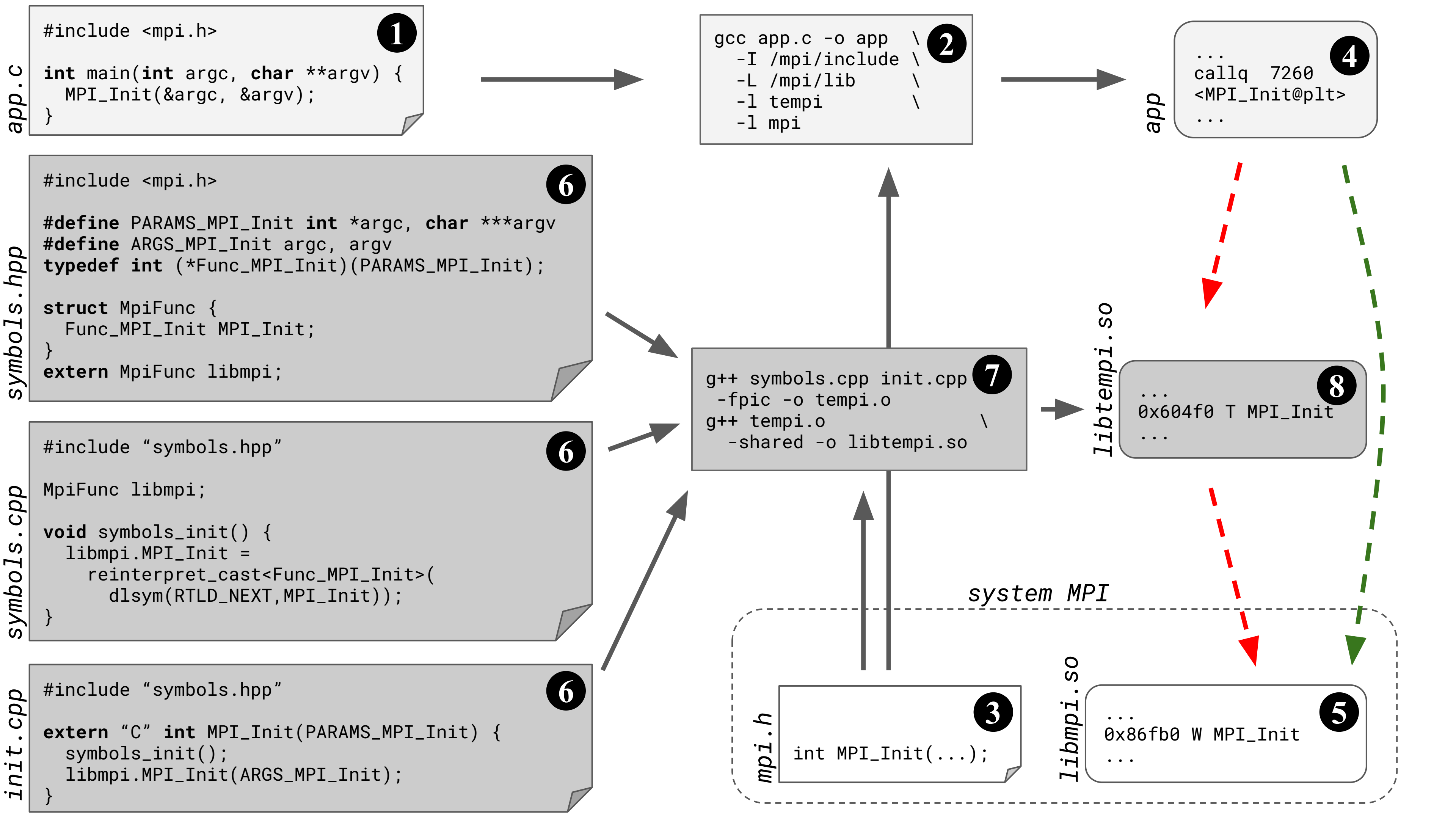
- MPI has a well-defined standard
  - Take advantage of this + how OS loads libraries to inject modifications

TEMPI

- "Temporary MPI" / "Topology Experiments for MPI" / plural of tempo (speed)
- MPI interposer




app.c
#include <mpi.h>
int main(int argc, char **argv) {
MPI_Init(&argc, &argv);
}
1
gcc app.c -o app \
-I /mpi/include \
-L /mpi/lib \
-l mpi
2
app
...
callq 7260
<MPI_Init@plt>
...
4
8
system MPI
mpi.h
int MPI_Init(...);
3
libmpi.so
...
0x86fb0 W MPI_Init
...
5


app.c
#include <mpi.h>
int main(int argc, char **argv) {
MPI_Init(&argc, &argv);
}
1
gcc app.c -o app \
-I /mpi/include \
-L /mpi/lib \
-l tempi \
-l mpi
2
app
...
callq 7260
<MPI_Init@plt>
...
4
symbols.hpp
#include <mpi.h>
#define PARAMS_MPI_Init int *argc, char ***argv
#define ARGS_MPI_Init argc, argv
typedef int (*Func_MPI_Init)(PARAMS_MPI_Init);
struct MpiFunc {
Func_MPI_Init MPI_Init;
}
extern MpiFunc libmpi;
6
symbols.cpp
#include “symbols.hpp”
MpiFunc libmpi;
void symbols_init() {
libmpi.MPI_Init =
reinterpret_cast<Func_MPI_Init>(
dlsym(RTLD_NEXT,MPI_Init));
}
6
init.cpp
#include “symbols.hpp”
extern “C” int MPI_Init(PARAMS_MPI_Init) {
symbols_init();
libmpi.MPI_Init(ARGS_MPI_Init);
}
6
g++ symbols.cpp init.cpp
-fpic -o tempi.o
g++ tempi.o \
-shared -o libtempi.so
7
libtempi.so
...
0x604f0 T MPI_Init
...
8
system MPI
mpi.h
int MPI_Init(...);
3
libmpi.so
...
0x86fb0 W MPI_Init
...
5

Conclusion

- Stencil code
- How they are parallelized
- Why non-contiguous data matters
- New MPI derived datatype approach
- A way to deploy experimental changes to systems.

Thank You

Slides & Paper: carlpearson.net > “Talks” > Click the May 5th link