$Chor_{MPI}$: The Chor of the Message Passing Interface (MPI)

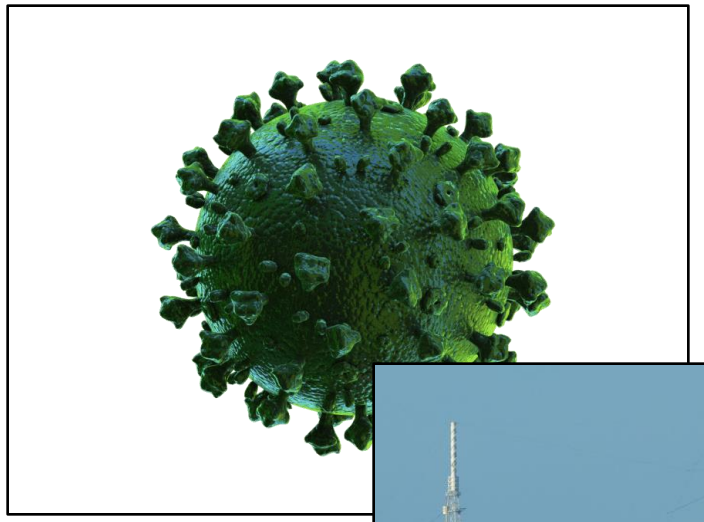
Choreographic Programming Workshop PLDI26

Keith Allen

Joint work with Andrew K. Hirsch and Matt Knepley

Intro to MPI

- Default standard for High-Performance Computing (HPC)
- Used in medical, aerospace, and energy research
- Library specification implemented in different languages
  - C, C++, Fortran
  - OCaml



[1]

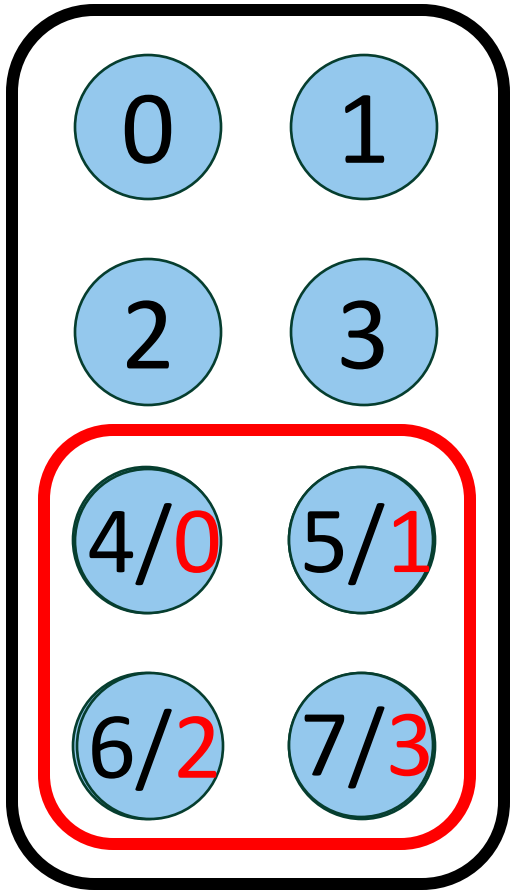
[2]

[3]

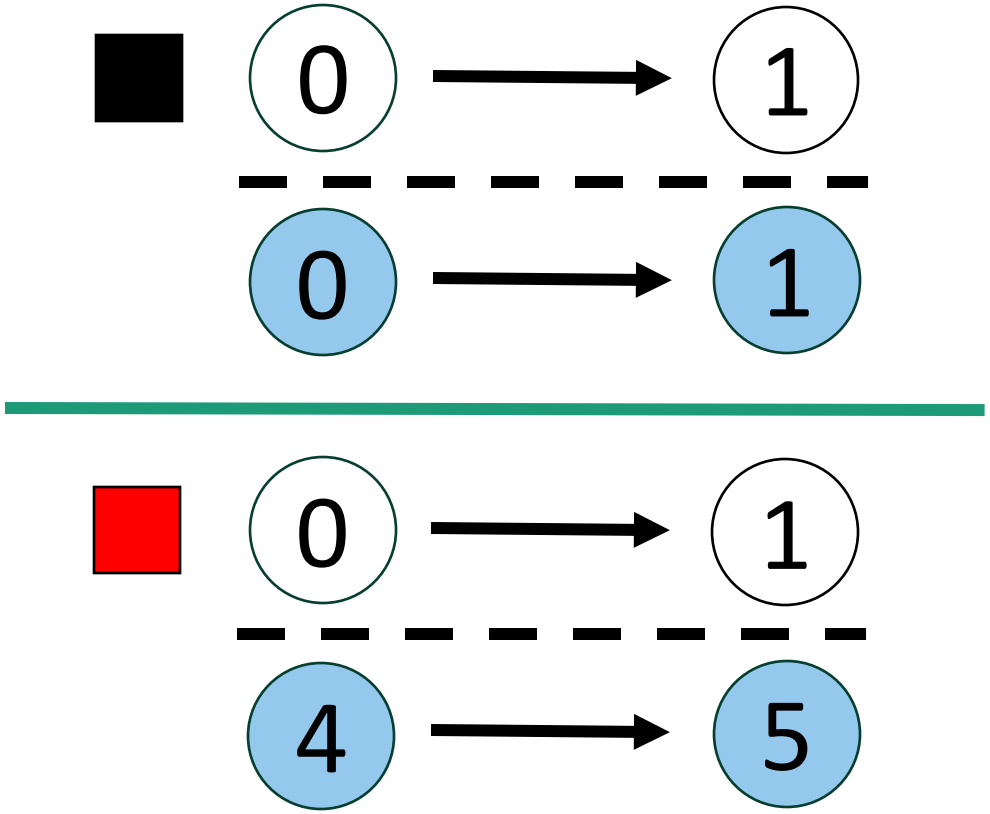
[1] Covid19 HPC Consortium
[2] NASA/Bill Ingalls
[3] Argonne National Laboratory

MPI Crash Course

Groups



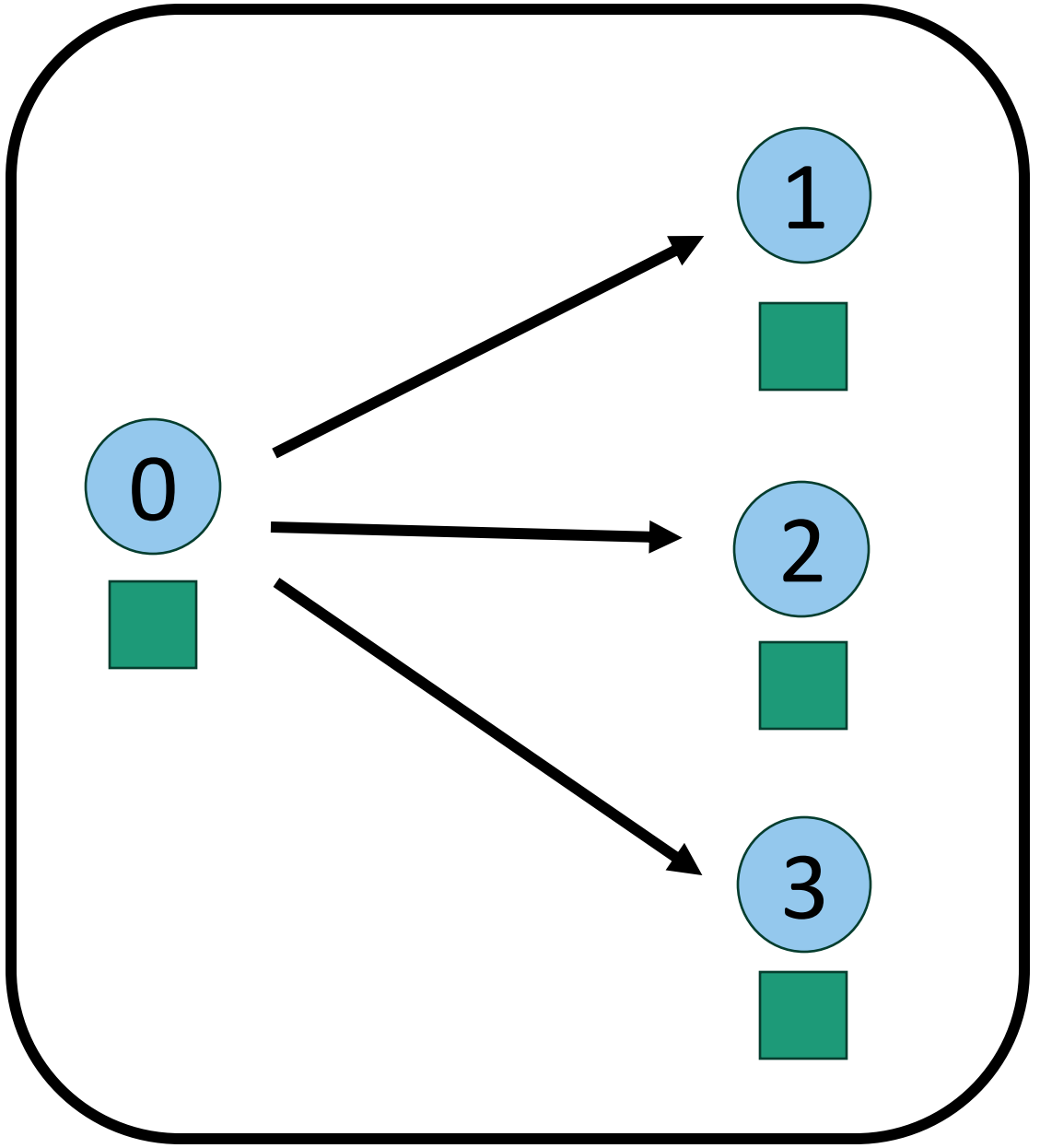
Point to Point Comm.



MPI Crash Course

Collective Communication

Broadcast: Root sends same value to all members

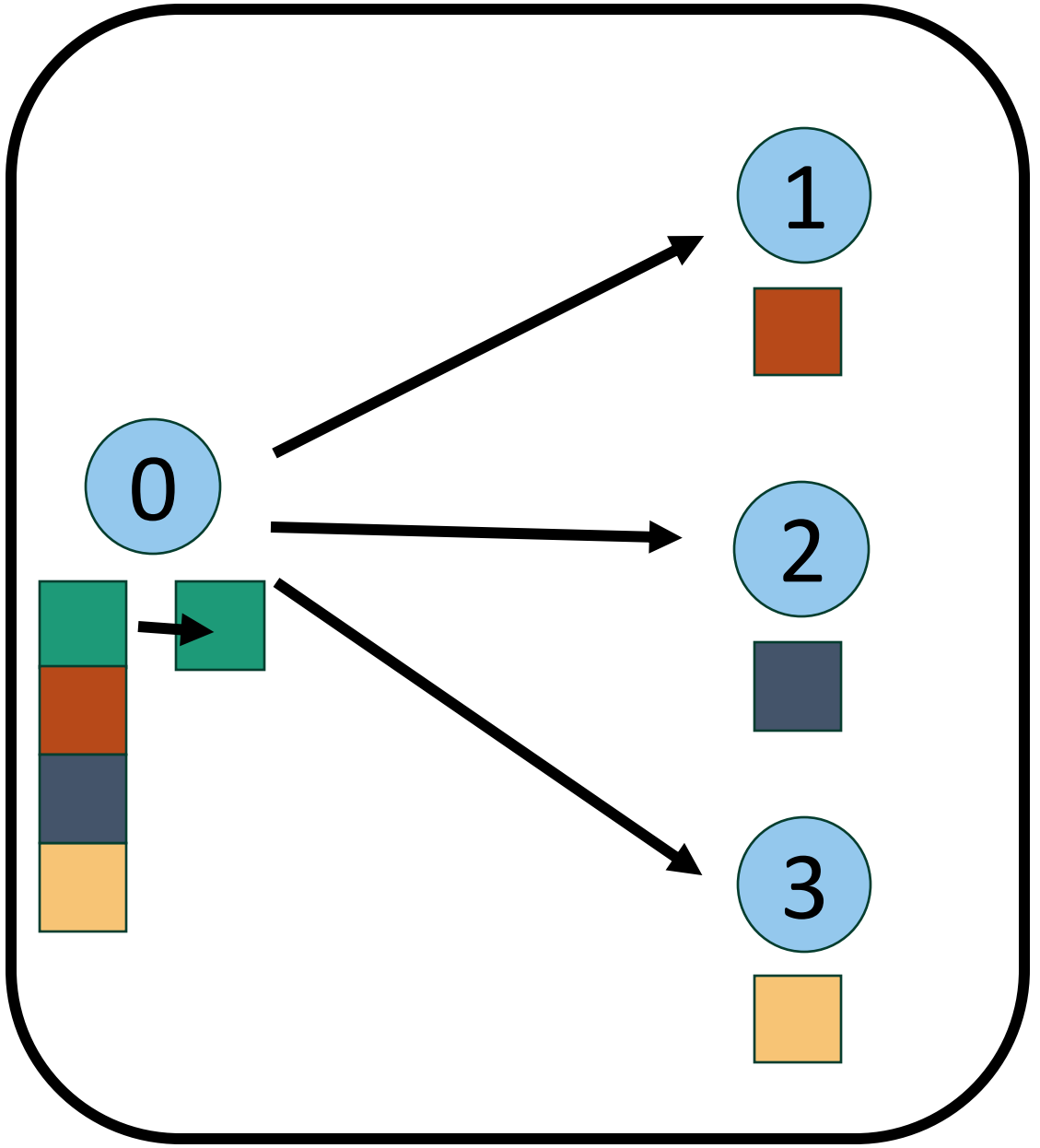


MPI Crash Course

Collective Communication

Broadcast: Root sends same value to all members

Scatter: Root splits and sends an array



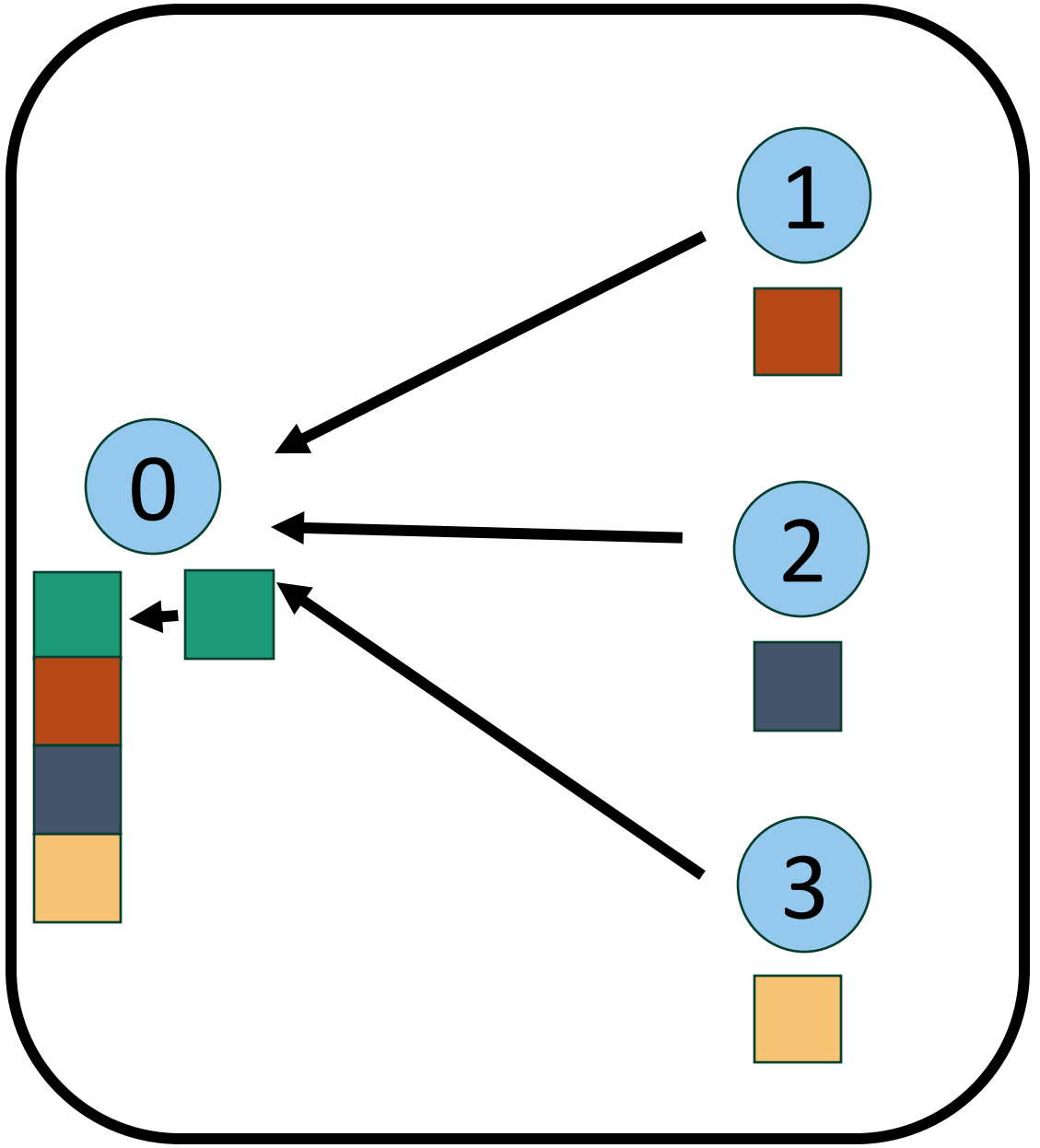
MPI Crash Course

Collective Communication

Broadcast: Root sends same value to all members

Scatter: Root splits and sends an array

Gather: Root collects and combines arrays from all members



Is deadlock freedom of MPI programs important?

Yes!

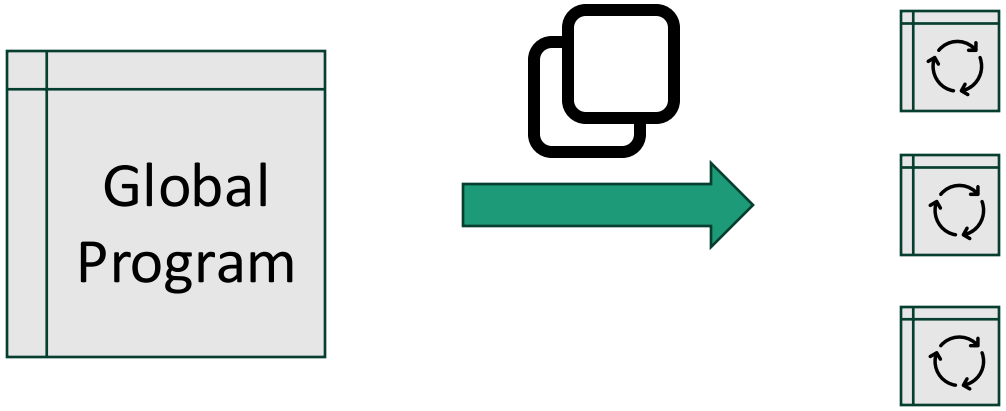
MPI programs answer important questions and are expensive to run

Why Choreographic Programming with MPI?

MPI programmers already use psuedo-choreographic style

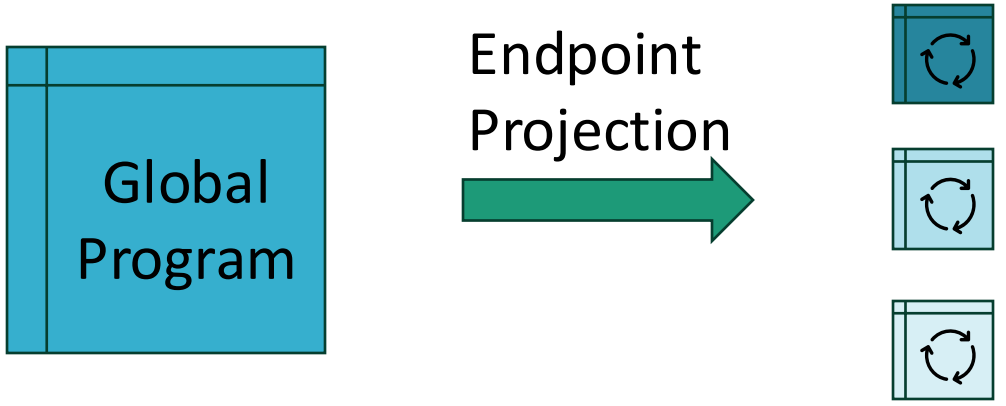
MPI vs Choreographic Programming

MPI



Each process runs a **copy** of the global program

Chor. Prog.



Each process runs a **projected version** of the global program

MPI vs Choreographic Programming

MPI
(mpi.tutorial.com)

```
...
int number;
if (world_rank == 0) {
    number = data;
    MPI_Send(&number, 1, MPI_INT,
      1, 0, MPI_COMM_WORLD);
} else if (world_rank == 1) {
    MPI_Recv(&number, 1, MPI_INT,
      0, 0, MPI_COMM_WORLD,
      MPI_STATUS_IGNORE);
}
```

Group

Sender

Msg

Receiver

$Chor_{MPI}$

```
...
g.0.(data) ~> 1
```

λ_{MPI}

`g = MPI_COMM_WORLD`

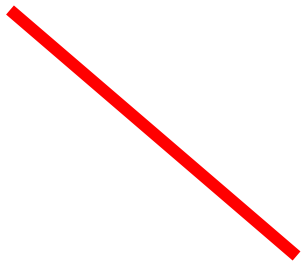
p_0)...
send data tag t to 1 of g

p_1)...
recv <tag t, from 0> of g

MPI vs Choreographic Programming

MPI (from mpi.tutorial.com)

```
...
  MPI_Bcast(data, num_elements,
  MPI_INT, 0, MPI_COMM_WORLD);
```



Root: Broadcast to group
Non-Root: Receive from root

$Chor_{MPI}$

```
...
data ~>br MPI_COMM_WORLD
```

Collaborating with Matt Knepley
(Core Developer of PETSc) at UB

We are building a $Chor_{MPI}$, a choreographic HPC language, based on MPI, to ensure deadlock freedom.

A functional subset of
the MPI standard

Network Language: λ_{MPI}

- Concurrent lambda calculus
- Key features
  - Blocking/Non-blocking Point to Point Comm.
  - Collective Communication
  - Extensive group calculus

- Model of MPI
- Mechanized proofs in Rocq
  - Work in progress!

```
send v tag t to q of g
recv <tag t, from p> of g

broadcast e tag t to g
scatter n of e tag t to g
gather n of e tag t and g

gunion g1 g2
gsplit f g
gshuffle a g
```

Chor_{MPI}

- Work in progress choreographic langauge

`send v tag t to q of g` `recv <tag t, from p> of g`	`g.p.v ~> q`
`broadcast e tag t to g`	`e ~>br g`
`scatter n of e tag t to g`	`e/n ~>sc g`
`gather n of e tag t and g`	`e/n ~>ga g`

Chor_{MPI}

Faceted Values

$^{facv}[e_0; \ldots; e_n]^i$: facval(g, t)

$g = {}^{G}[6,3,0,2]^i$

$^{facv}[[0;1];[2;3];[4;5];[6;7]]$
: facval(g, array(nat)

$^{facv}[17;18;19;20]$
: facval(g, nat)

Chor_{MPI}

SIMD Block `SIMD (p ∈ g, x ∈ facv){e}`

Expression `e` is run at each process `p` in `g` which has access to `x`, the processes piece of the facet value `facv`.

SIMD results in a faceted value with type `facval(g, t)` where `e : t`

Chor_{MPI} Role Call Example

Scatter an array of 0's, add your rank in the group to the 0, and gather all the answer to the root

```
let g = G[6,3,0,2]i in
  let facv = (g.0.[0;0;0;0]j)/1 ~>sc g in
    SIMD (p ∈ g, x ∈ facv) {
      [(rank g) + (aget x 0)]k
    }/1 ~>ga g
```

- 4 process group
- global 6 is root

- array of 0's located at g.0
- scatter results in a facval

- gather takes a facval

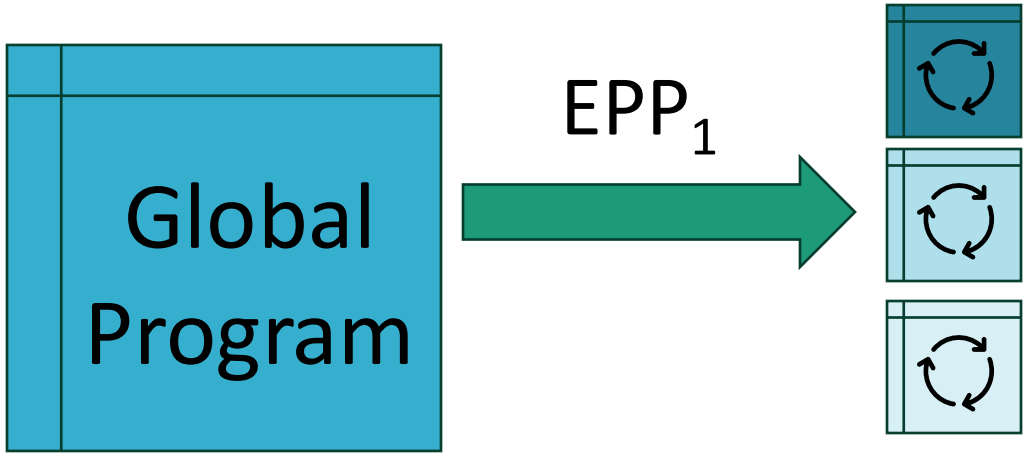
- add your rank
- aget since receiving an array

MLV and SIMD Values

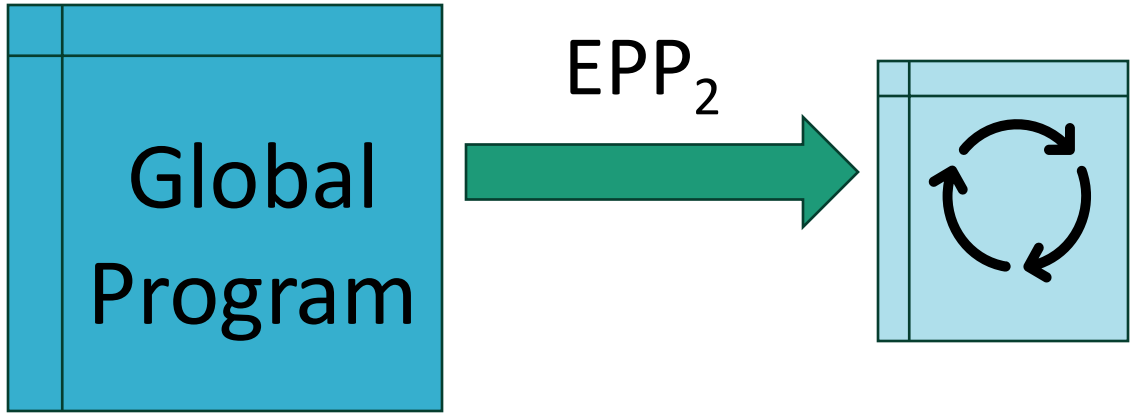
- Multiply Located Values (MLV)
  - Broadcast
  - Seen in He-λsmall [Bates and Near 2024] and λ_{QC} [Samuelson et al. 2025]
- Single Instruction Multi. Data (SIMD)
  - Scatter and Gather
  - Faceted Values
  - Symphony [Sweet et al. 2023]

Endpoint Projection

- Plan to explore 2 types of EPP



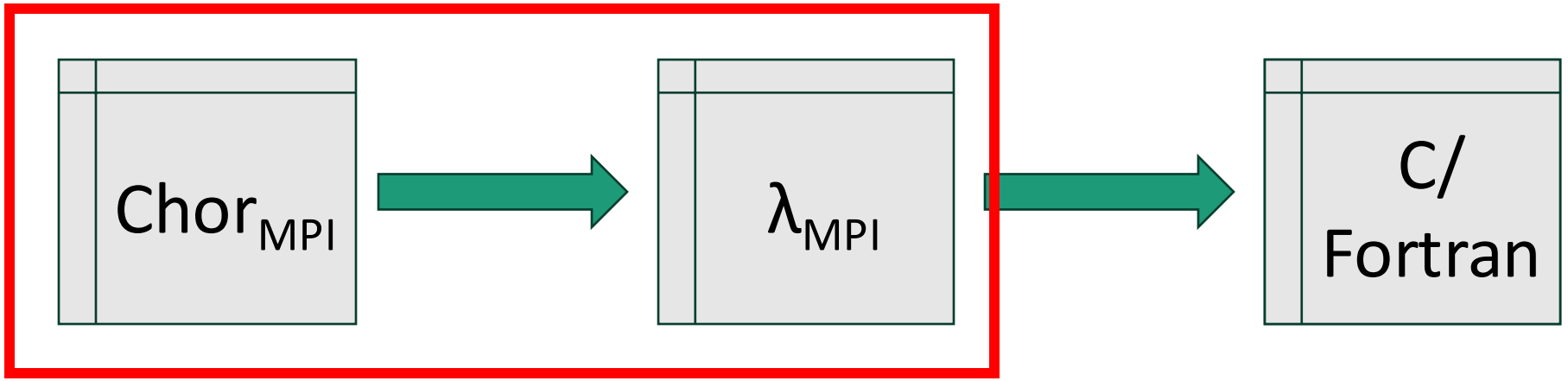
Classic choreographic projection



Choreographic reasoning, with MPI style programs

Compiler Project

Working with two undergrad interns



Goal : Show Chor$_{MPI}$ and λ_{MPI} do not add unreasonable overhead

Quick Summary

- Unverified MPI code is being used in important computations
- MPI programmers are already thinking and writing in a psuedo-choreograhic style
- We are building Chor_{MPI}, which will ensure deadlock freedom, with minimal interruptions to the MPI workflow
  - Work in progress!

Picture Citations

- [1] Covid19 HPC Consortium, https://covid19-hpc-consortium.org/.
- [2] Ingalls, Bill and NASA. *Artemis II Launch*. 2026. *NASA,* https://www.nasa.gov/image-detail/amf-nhq202604010308/.
- [3] Argonne National Laboratory. 2024. Argonne National Laboratory, https://www.anl.gov/article/no-power-no-operator-no-problem-argonne-test-facility-simulates-nuclear-reactors-to-explore.

$Chor_{MPI}$: The Chor of the Message Passing Interface (MPI)

Choreographic Programming Workshop PLDI26

Keith Allen

Joint work with Andrew K. Hirsch and Matt Knepley

Group Calc

- Three default groups
  - Global group $^{G}[0,1,\ldots,(n-1)]^{i}$
  - Empty group $^{G}[\,]^{i}$
  - Singleton group $^{G}[m]^{i}$
- MPI Group Constructors
  - Union, difference, intersection
  - Union, split, and shuffle

Group Constructors

- Union (identical to MPI)
  - `gunion` ^{G}`[0,3,5]` ^{G}`[3,1,4] ==>` ^{G}`[0,3,5,1,4]`
- Split
  - `gsplit (fun _ x := x mod 2)` ^{G}`[6,3,0,2] ==>`
    `[`^{G}`[6,0]; `^{G}`[3,2]]`
- Shuffle
  - `gshuffle [3;1;2;0]` ^{G}`[6,3,0,2] ==>` ^{G}`[2,3,0,6]`

Can build all group constructors