Portable, MPI-Interoperable Coarray Fortran

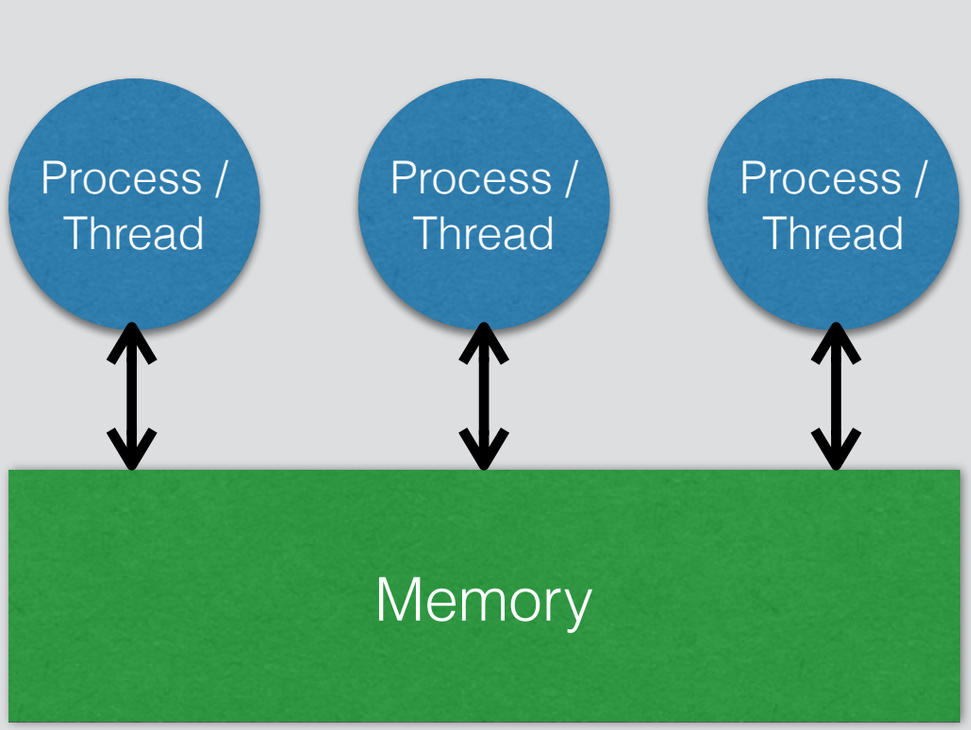
Chaoran Yang,[1] Wesley Bland,[2]
John Mellor-Crummey,[1] Pavan Balaji[2]

[1]Department of Computer Science
Rice University
Houston, TX

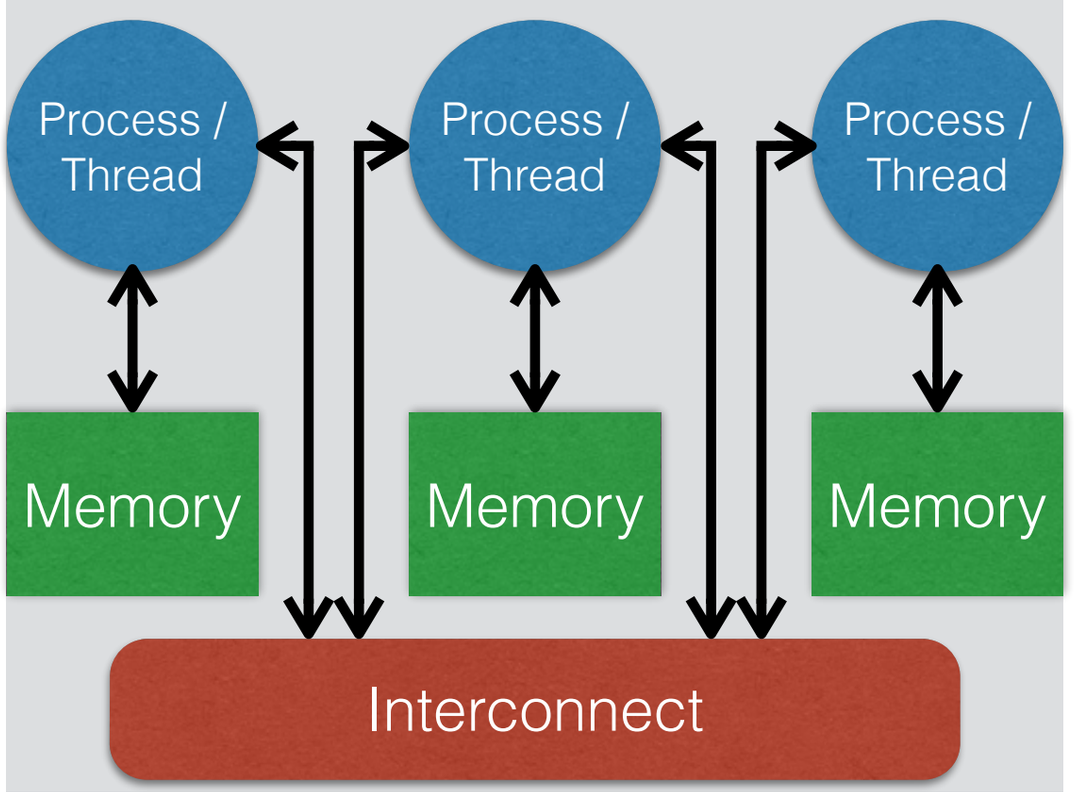
[2]Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL

Parallel Programming Models

- Shared memory
  - Pthread, OpenMP, …
  - “easy” to program
  - hard to scale

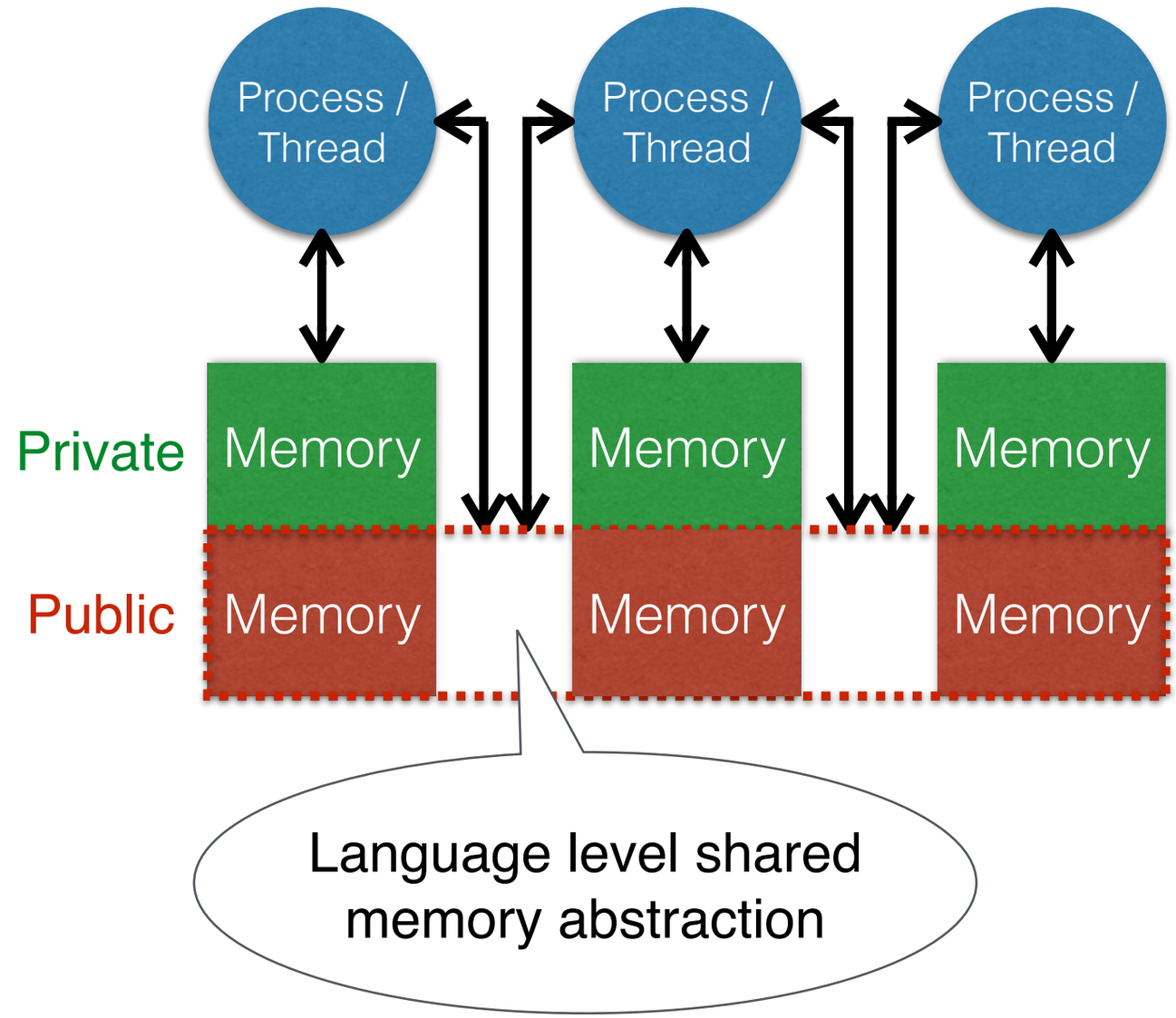


- Message passing
  - MPI, …
  - “scalable”
  - hard to program



Partitioned Global Address Space Languages

- Combine the strength
  - shared memory models
    - “Easy to program”
  - message passing models
    - “Scalable”
- Example PGAS Languages
  - Unified Parallel C (C)
  - Titanium (Java)
  - Coarray Fortran (Fortran)
- Related efforts
  - X10 (IBM)
  - Chapel (Cray)
  - Fortress (Oracle)

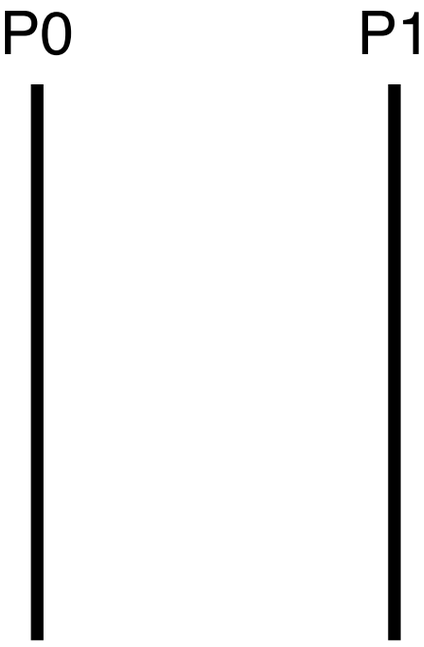


Current status

- The dominance of Message Passing Interface (MPI)
  - Most applications on clusters are written using MPI
  - Many high-level libraries on clusters are built with MPI
- Why people do not adopt PGAS languages?
  - Most PGAS languages are built with a different runtime system e.g. GASNet
  - Hard to adopt new programming models in existing applications **incrementally**

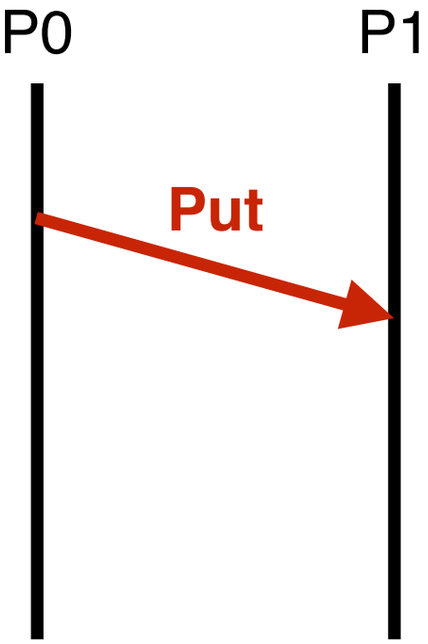
Problem 1: deadlock

```
PROGRAM MAY_DEADLOCK
   USE MPI
   CALL MPI_INIT(E)
   CALL MPI_COMM_RANK(MPI_COMM_WORLD, MY_RANK, E)
   IF (MYRANK .EQ. 0) A(:)[1] = A(:)
   CALL MPI_BARRIER(MPI_COMM_WORLD, IERR)
   CALL MPI_FINALIZE(E)
END PROGRAM
```



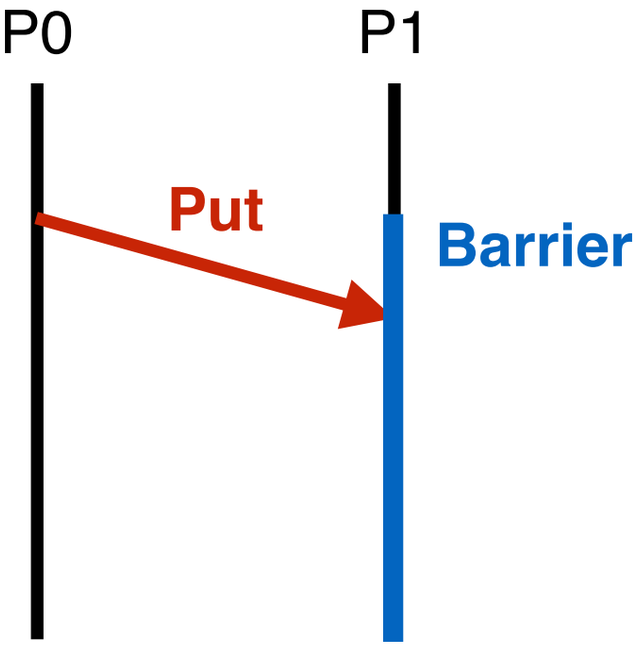
Problem 1: deadlock

```
PROGRAM MAY_DEADLOCK
   USE MPI
   CALL MPI_INIT(E)
   CALL MPI_COMM_RANK(MPI_COMM_WORLD, MY_RANK, E)
   IF (MYRANK .EQ. 0) A(:)[1] = A(:)
   CALL MPI_BARRIER(MPI_COMM_WORLD, IERR)
   CALL MPI_FINALIZE(E)
END PROGRAM
```



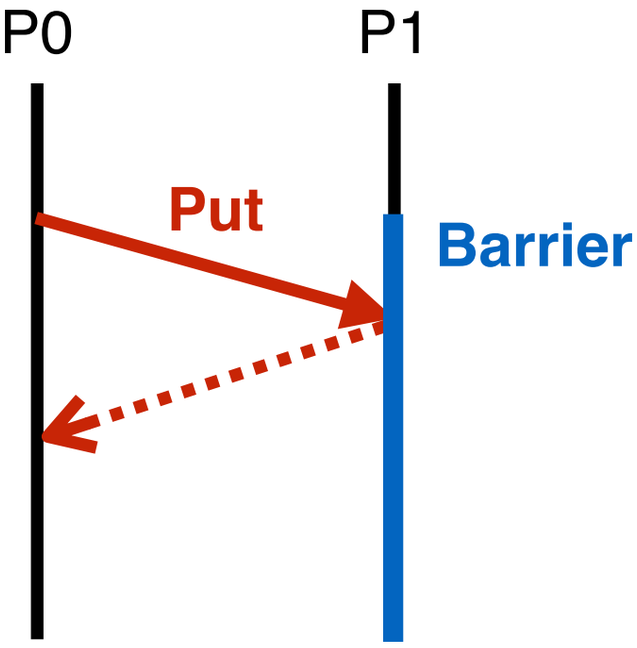
Problem 1: deadlock

```
PROGRAM MAY_DEADLOCK
   USE MPI
   CALL MPI_INIT(E)
   CALL MPI_COMM_RANK(MPI_COMM_WORLD, MY_RANK, E)
   IF (MYRANK .EQ. 0) A(:)[1] = A(:)
   CALL MPI_BARRIER(MPI_COMM_WORLD, IERR)
   CALL MPI_FINALIZE(E)
END PROGRAM
```

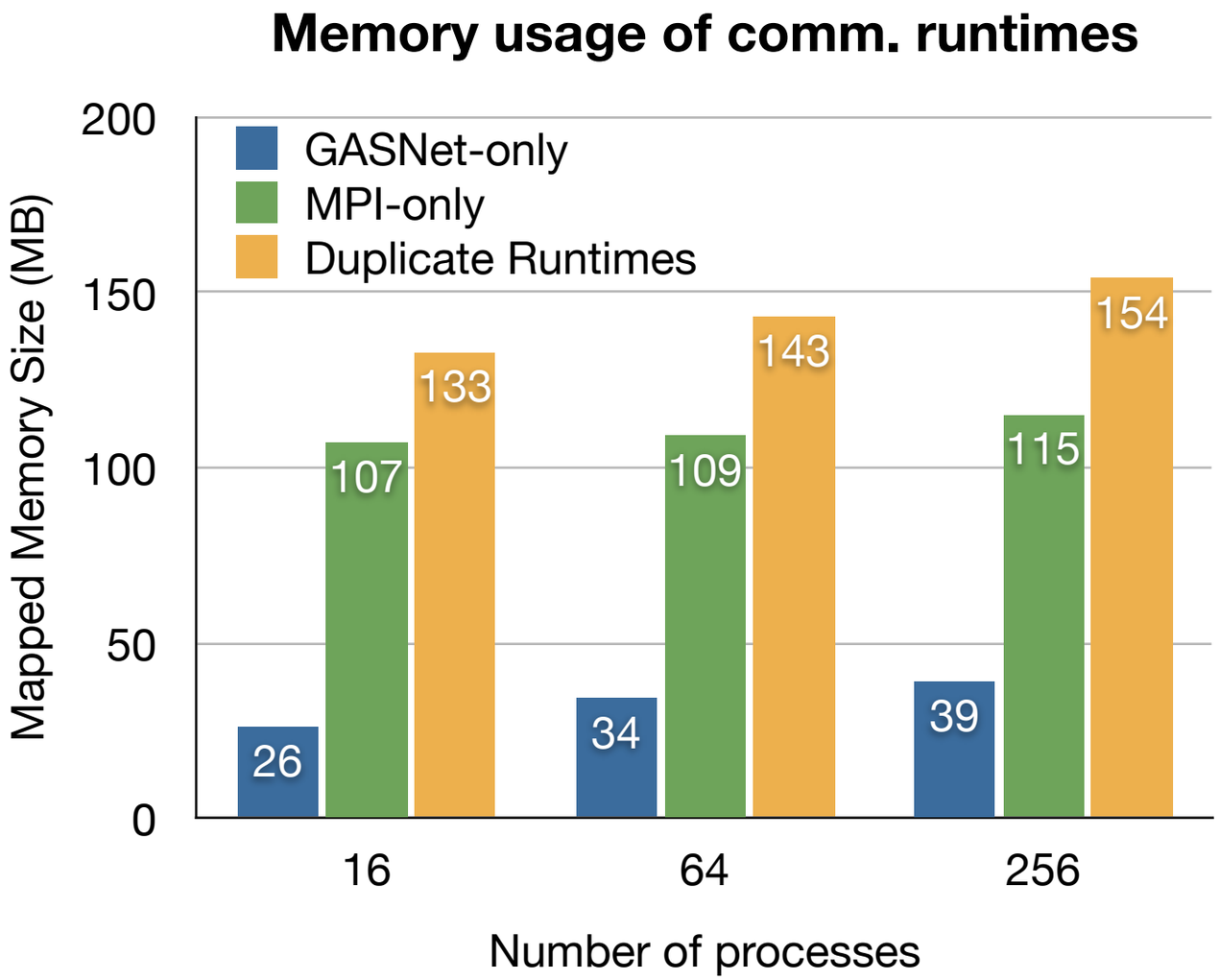


Problem 1: deadlock

```
PROGRAM MAY_DEADLOCK
   USE MPI
   CALL MPI_INIT(E)
   CALL MPI_COMM_RANK(MPI_COMM_WORLD, MY_RANK, E)
   IF (MYRANK .EQ. 0) A(:)[1] = A(:)
   CALL MPI_BARRIER(MPI_COMM_WORLD, IERR)
   CALL MPI_FINALIZE(E)
END PROGRAM
```



Problem 2: duplicates resources



- **Memory usage per process increases as the number of processes increases**
- At larger scale, excessive memory use of duplicate runtimes will hurt scalability

The problem

- PGAS languages DO NOT play well with MPI
  - program may deadlock using MPI and other runtimes
  - program unnecessarily uses duplicated resources

The problem

- PGAS languages DO NOT play well with MPI
  - program may deadlock using MPI and other runtimes
  - program unnecessarily uses duplicated resources

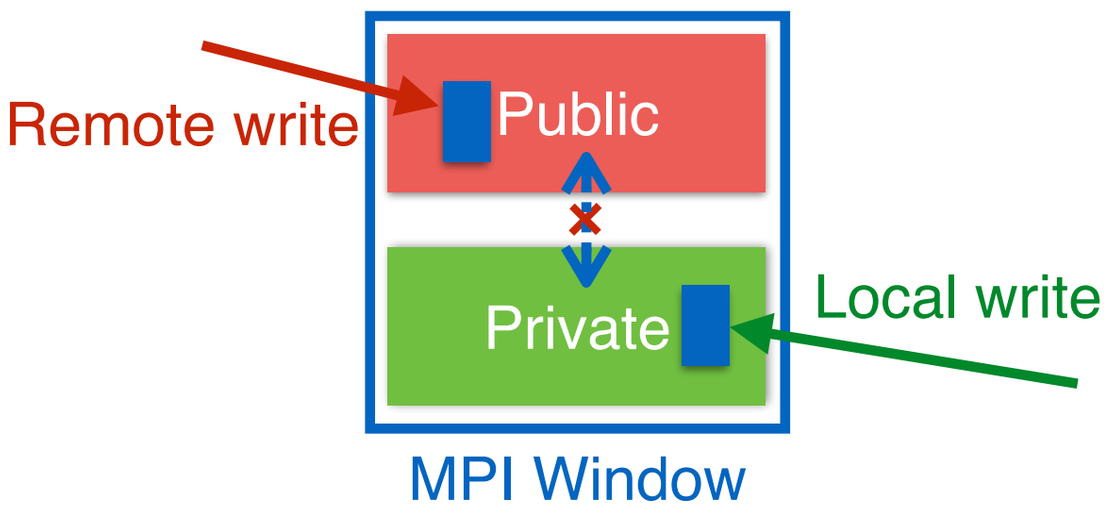
The solution

Build PGAS language runtimes with MPI

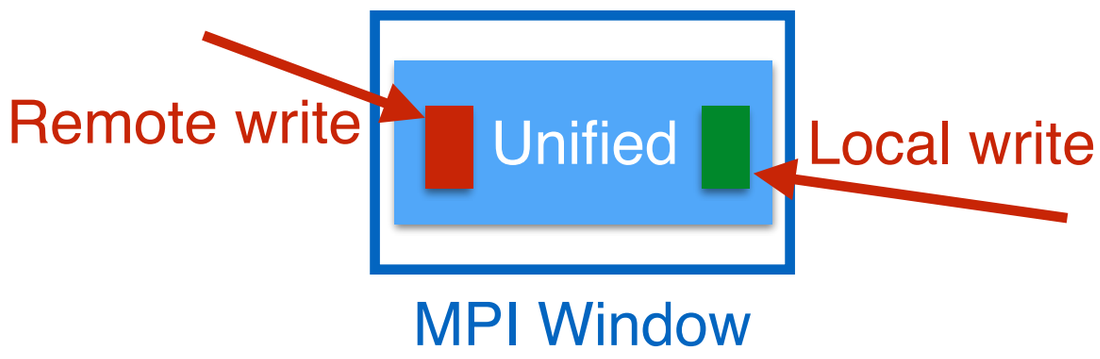
Why people haven't done it?

- Previously MPI was considered insufficient for this goal*
- MPI-3 adds extensive support for Remote Memory Access (RMA)

"Separate" Window (MPI-2)



"Unified' window (MPI-3)



*: D. Bonachea and J. Duell. Problems with using MPI 1.1 and 2.0 as compilation targets for parallel language implementations. *Int. J. High Perform. Comput. Netw.*, 1(1-3):91–99, Aug. 2004.

Question to investigate

- Build PGAS runtimes with MPI-3
  - Does it provide full interoperability?
  - Does it degrade performance?

Question to investigate

- Build PGAS runtimes with MPI-3
  - Does it provide full interoperability?
  - Does it degrade performance?

Approach

- Build a PGAS language (Coarray Fortran) with MPI-3 and evaluate its interoperability and performance

Coarray in Fortran 2008 (CAF)

- Fortran 2008 Standard contains features for parallel programming using a SPMD (Single Program Multiple Data) model
- What is a coarray?
  - extends array syntax with **codimensions**, e.g. **REAL :: X(10,10)[*]**
- How to access a coarray?
  - Reference with [] mean data on specified image, e.g. **X(1,:) = X(1,:)[p]**
  - May be allocatable, structure components, dummy or actual arguments

Coarray Fortran 2.0 (CAF 2.0)

"A rich extension to Coarray Fortran developed at Rice University"

- Teams (like MPI communicator) and collectives
- Asynchronous operations
  - asynchronous copy, asynchronous collectives, and function shipping
- Synchronization constructs
  - events, cofence, and finish

Features in Blue has been adopted by Fortran standard committee

Coarray and MPI-3 RMA

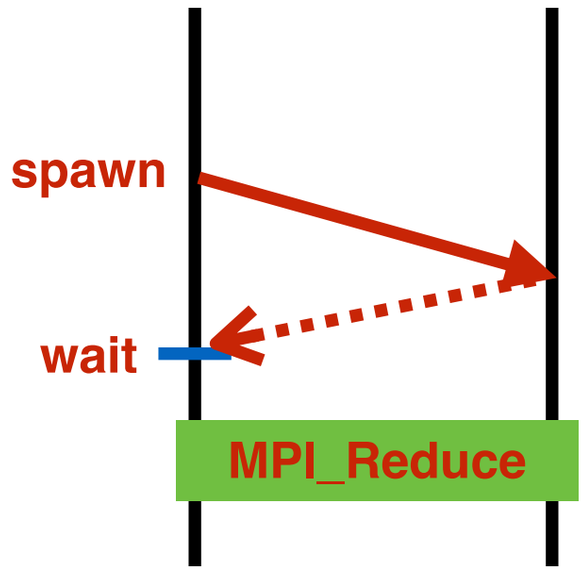
Mapping coarray features to MPI-3 APIs

- Initialization
  - `INTEGER :: A(100,100)[*]`
  - **MPI_WIN_ALLOCATE**, then **MPI_WIN_LOCK_ALL**
- Coarray Read & Write
  - `A(:)[1] = A(:); A(:) = A(:)[2]`
  - **MPI_RPUT; MPI_RGET**
- Synchronization
  - **MPI_WIN_SYNC (local)** & **MPI_WIN_FLUSH (_ALL) (global)**

Active Messages

"Lightweight, low-level, asynchronous, remote procedure calls"

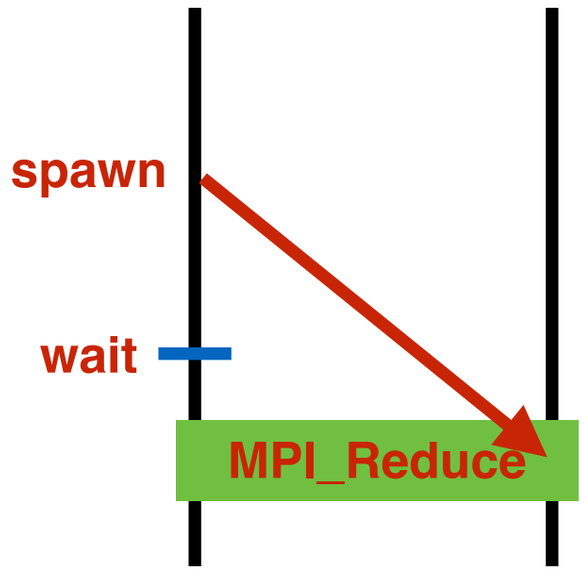
- Many CAF 2.0 features are built on top of Active Messages
- MPI does not provide an implementation of Active Messages
- Emulate Active Messages with **MPI_Send** and **MPI_Recv** routines
  - hurt performance - cannot overlap communication with AM handlers
  - hurt interoperability - could cause deadlock



Active Messages

"Lightweight, low-level, asynchronous, remote procedure calls"

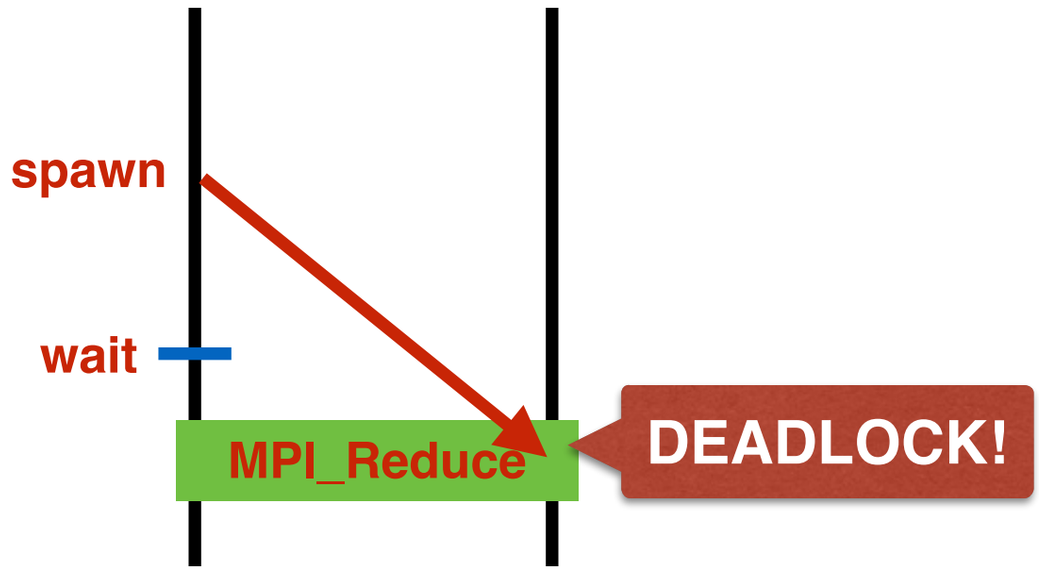
- Many CAF 2.0 features are built on top of Active Messages
- MPI does not provide an implementation of Active Messages
- Emulate Active Messages with **MPI_Send** and **MPI_Recv** routines
  - hurt performance - cannot overlap communication with AM handlers
  - hurt interoperability - could cause deadlock



Active Messages

"Lightweight, low-level, asynchronous, remote procedure calls"

- Many CAF 2.0 features are built on top of Active Messages
- MPI does not provide an implementation of Active Messages
- Emulate Active Messages with **MPI_Send** and **MPI_Recv** routines
  - hurt performance - cannot overlap communication with AM handlers
  - hurt interoperability - could cause deadlock



CAF 2.0 Events

"similar to counting semaphores"

Maps to Active Messages

- **CALL event_notify(event, n)**
  - Need to ensure all previous asynchronous operations have completed before the notification

```
for each window
   MPI_Win_sync(win)
for each dirty window
   MPI_Win_flush_all(win)
AM_Request(…) // MPI_Isend
```

- **CALL event_wait(event, n)**
  - Also serves as a compiler barrier (prevent compiler from reorder operations upward)

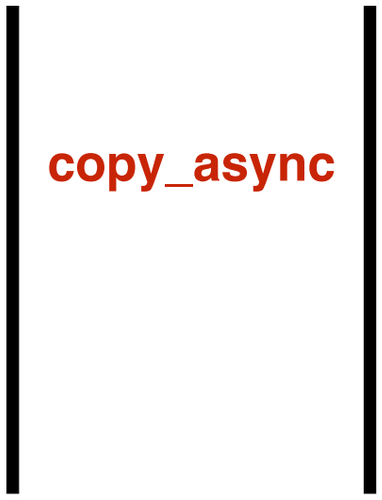
```
while (count < n)
   for each window
      MPI_Win_sync(win)
   AM_Poll(…) // MPI_Iprobe
```

CAF 2.0 Asynchronous Operations

- **copy_async(dest, src, dest_event, src_event, pred_event)**

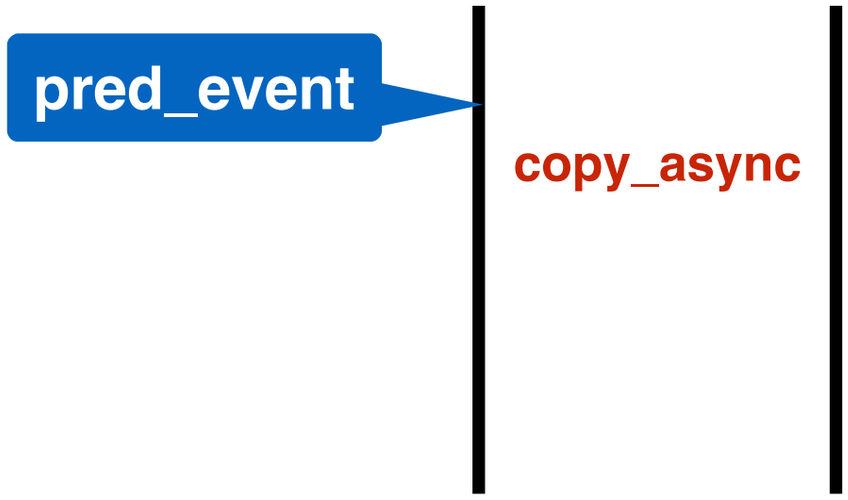
CAF 2.0 Asynchronous Operations

- **copy_async(dest, src, dest_event, src_event, pred_event)**



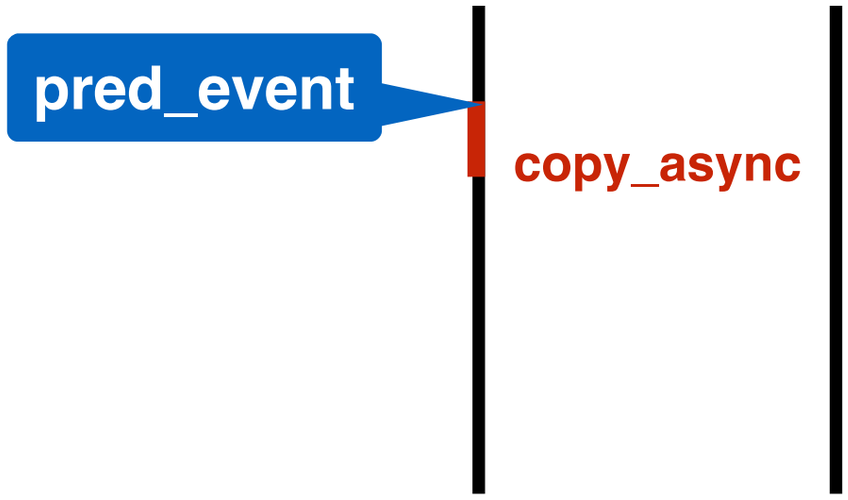
CAF 2.0 Asynchronous Operations

- **copy_async(dest, src, dest_event, src_event, pred_event)**



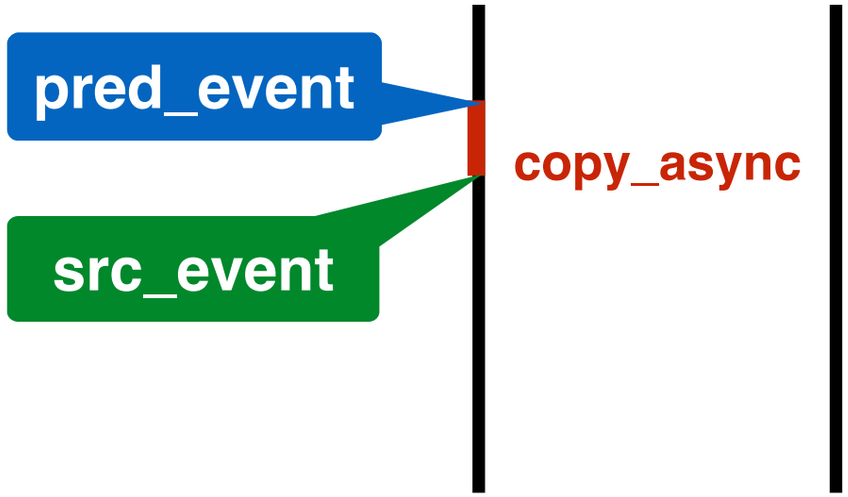
CAF 2.0 Asynchronous Operations

- **copy_async(dest, src, dest_event, src_event, pred_event)**



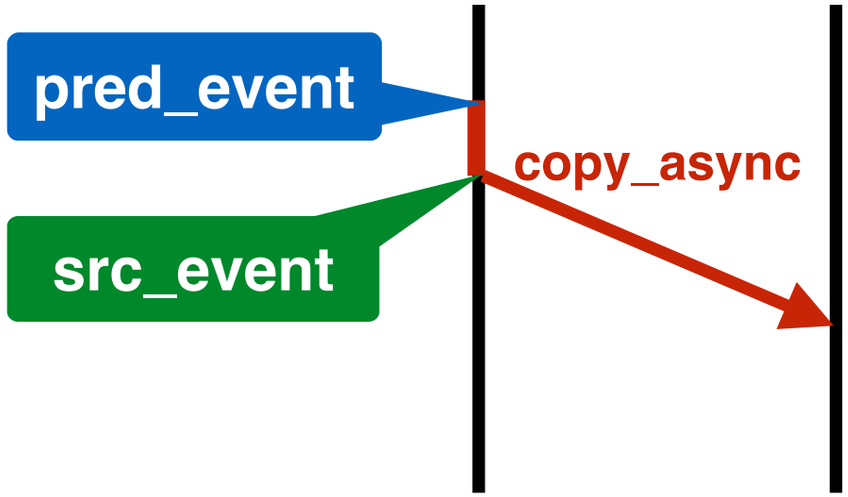
CAF 2.0 Asynchronous Operations

- **copy_async(dest, src, dest_event, src_event, pred_event)**



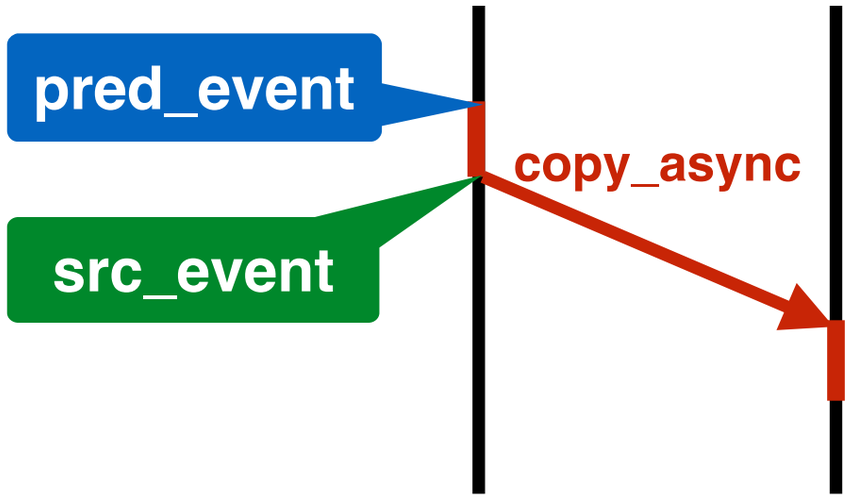
CAF 2.0 Asynchronous Operations

- **copy_async(dest, src, dest_event, src_event, pred_event)**



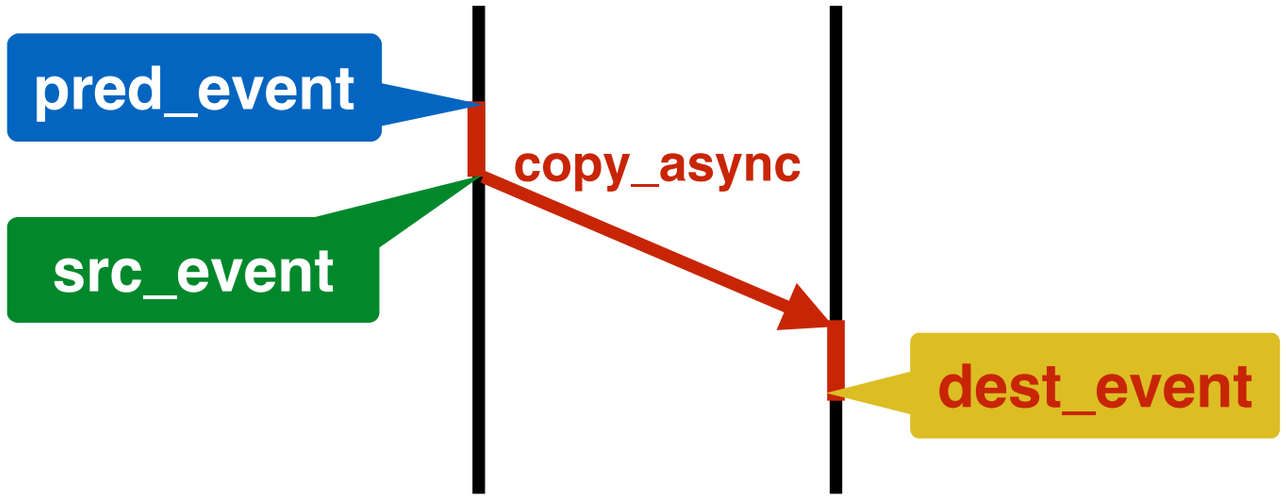
CAF 2.0 Asynchronous Operations

- **copy_async(dest, src, dest_event, src_event, pred_event)**



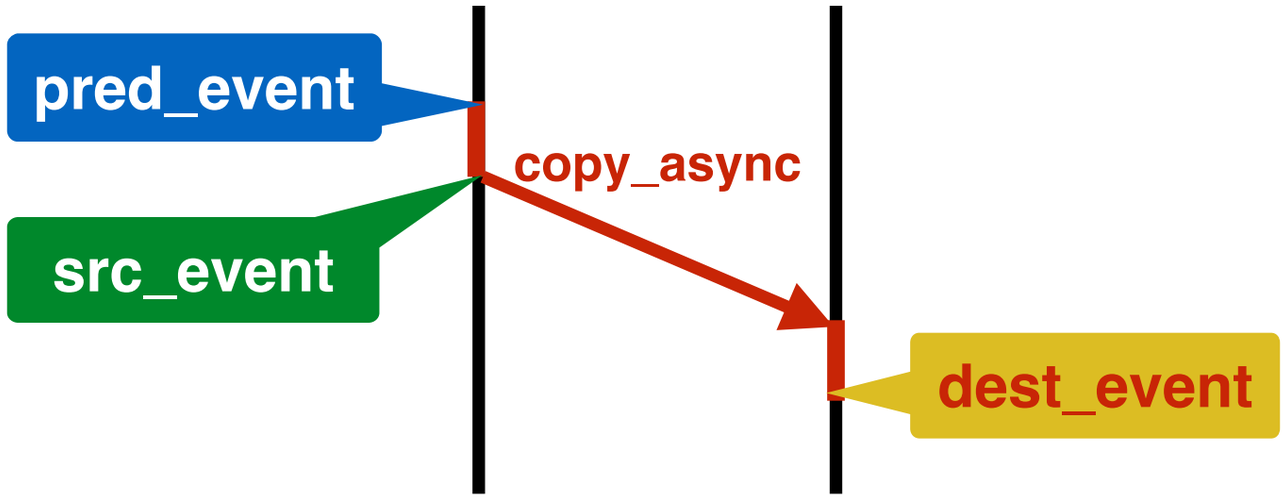
CAF 2.0 Asynchronous Operations

- **copy_async(dest, src, dest_event, src_event, pred_event)**



CAF 2.0 Asynchronous Operations

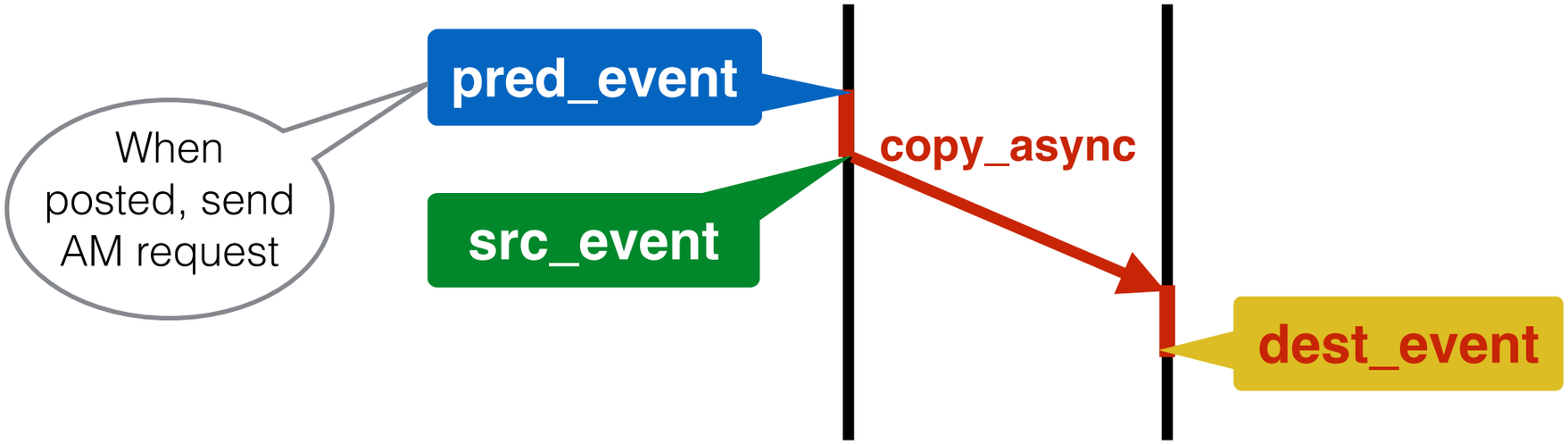
- **copy_async(dest, src, dest_event, src_event, pred_event)**



- Map **copy_async** to **Active Message**

CAF 2.0 Asynchronous Operations

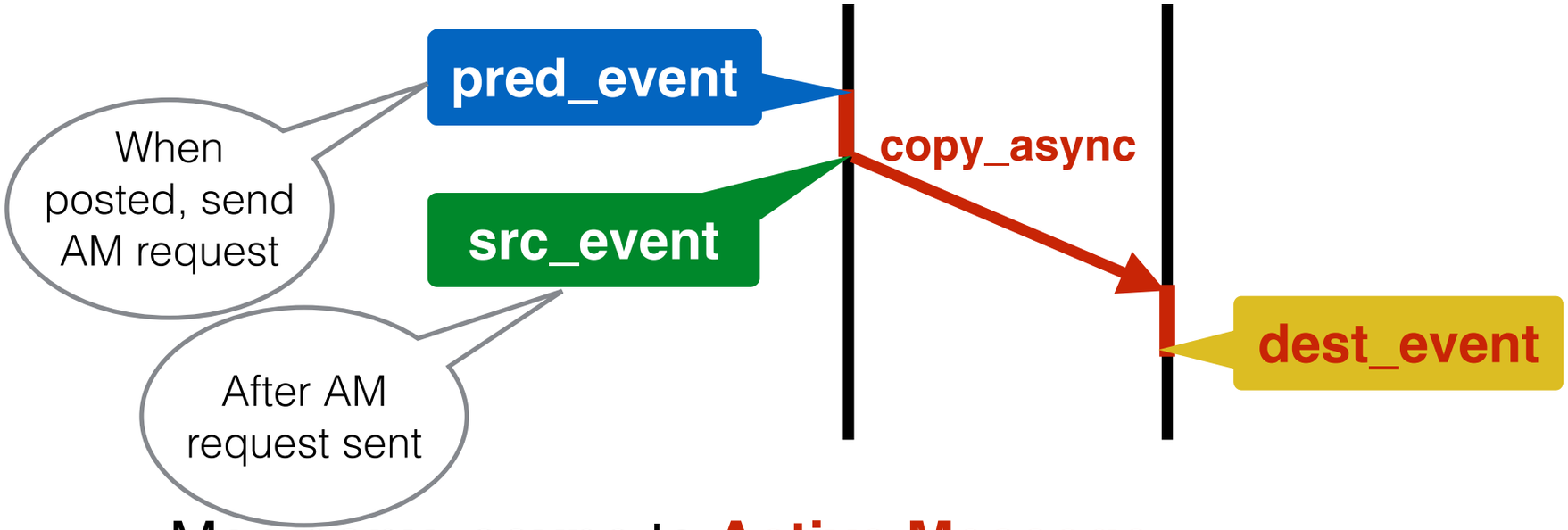
- **copy_async(dest, src, dest_event, src_event, pred_event)**



- Map **copy_async** to **Active Message**

CAF 2.0 Asynchronous Operations

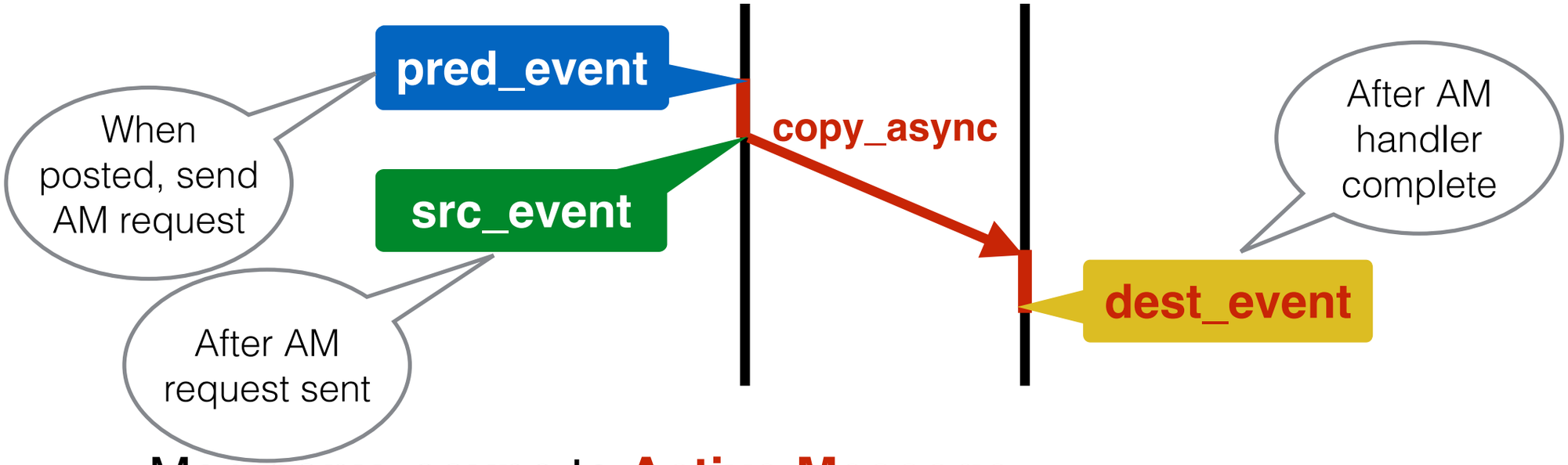
- **copy_async(dest, src, dest_event, src_event, pred_event)**



- Map **copy_async** to **Active Message**

CAF 2.0 Asynchronous Operations

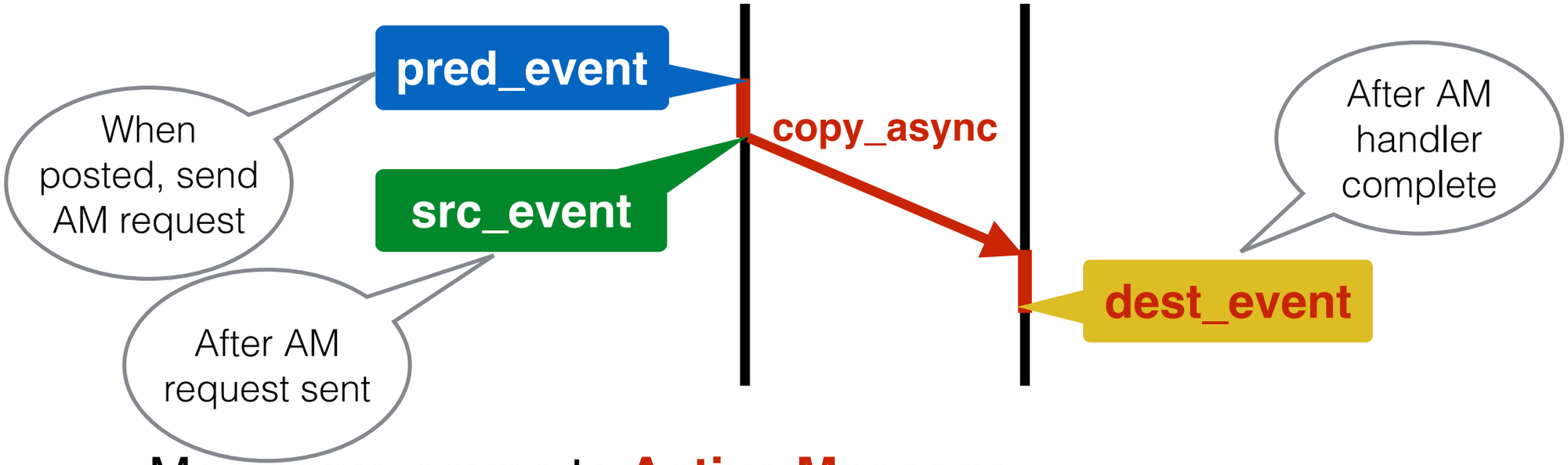
- **copy_async(dest, src, dest_event, src_event, pred_event)**



- Map **copy_async** to **Active Message**

CAF 2.0 Asynchronous Operations

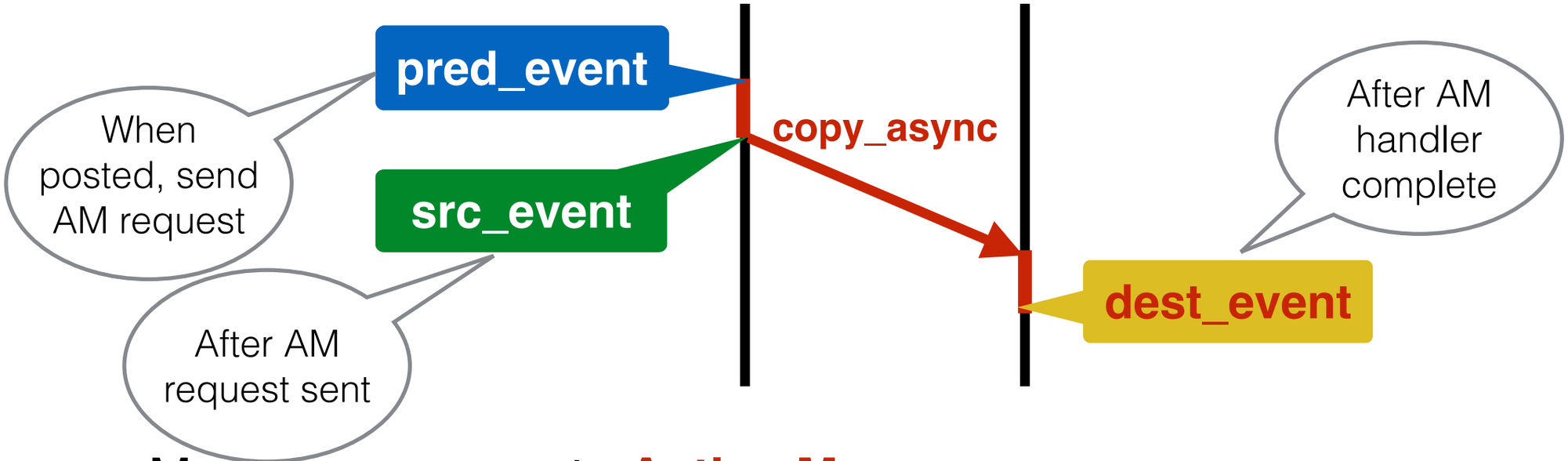
- **copy_async(dest, src, dest_event, src_event, pred_event)**



- Map **copy_async** to **Active Message**
  - MPI does not have AM support

CAF 2.0 Asynchronous Operations

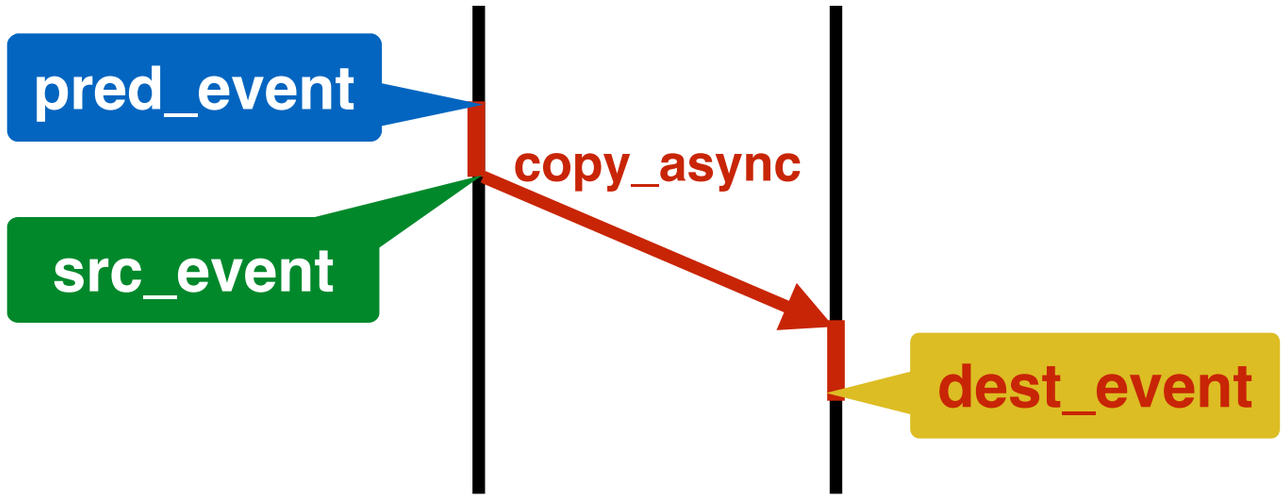
- **copy_async(dest, src, dest_event, src_event, pred_event)**



- Map **copy_async** to **Active Message**
  - MPI does not have AM support
- Map **copy_async** to **MPI_RPUT** (or **MPI_RGET**)

CAF 2.0 Asynchronous Operations

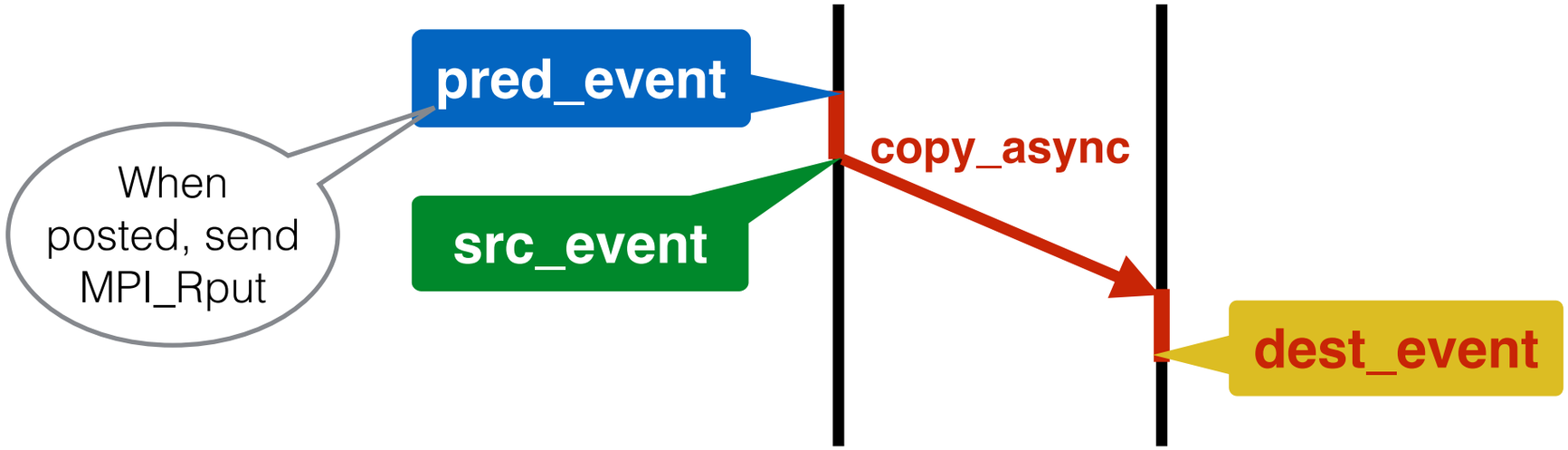
- **copy_async(dest, src, dest_event, src_event, pred_event)**



- Map **copy_async** to **Active Message**
  - MPI does not have AM support
- Map **copy_async** to **MPI_RPUT** (or **MPI_RGET**)

CAF 2.0 Asynchronous Operations

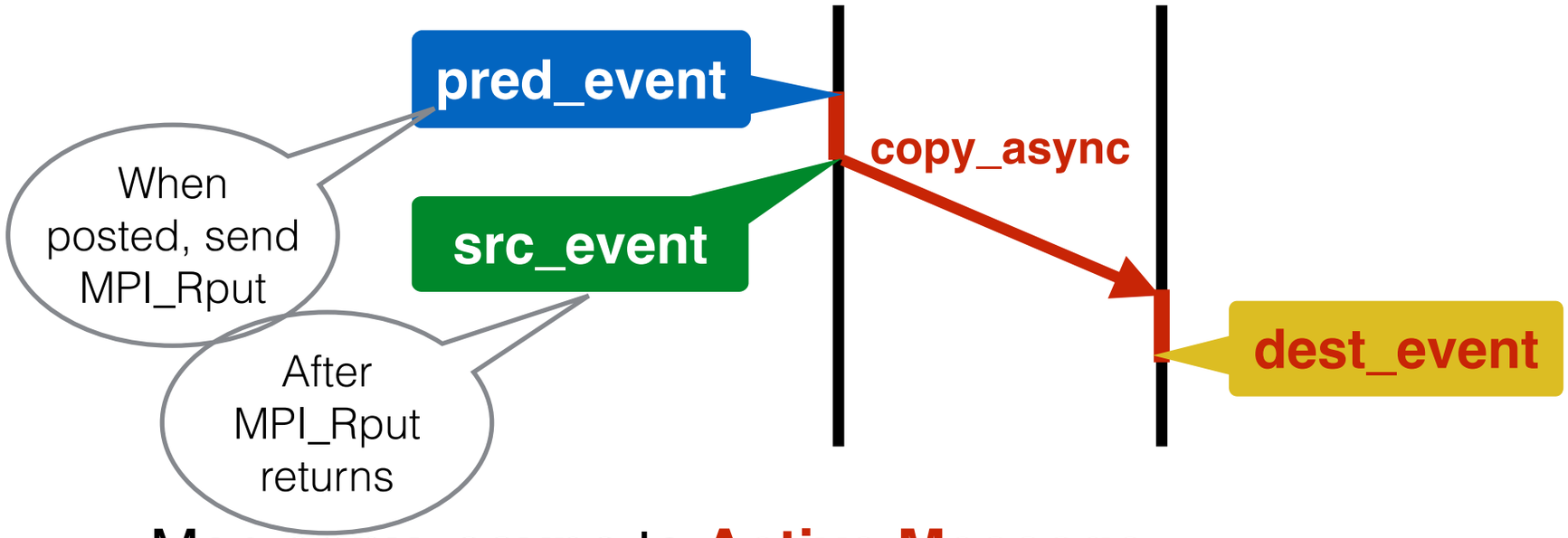
- **copy_async(dest, src, dest_event, src_event, pred_event)**



- Map **copy_async** to **Active Message**
  - MPI does not have AM support
- Map **copy_async** to **MPI_RPUT** (or **MPI_RGET**)

CAF 2.0 Asynchronous Operations

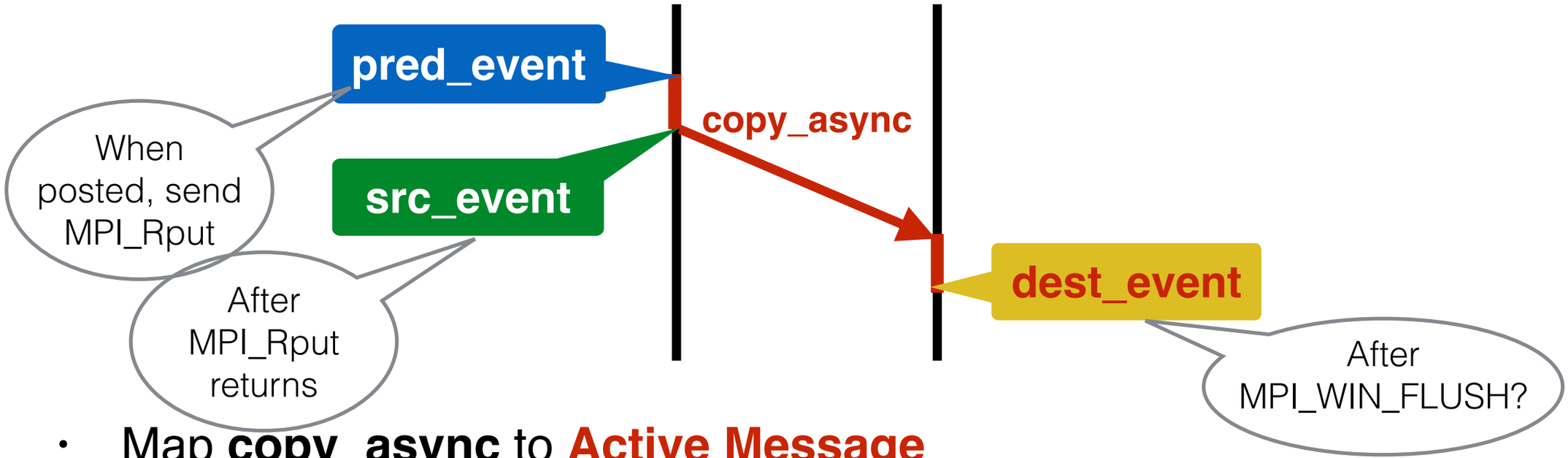
- **copy_async(dest, src, dest_event, src_event, pred_event)**



- Map **copy_async** to **Active Message**
  - MPI does not have AM support
- Map **copy_async** to **MPI_RPUT** (or **MPI_RGET**)

CAF 2.0 Asynchronous Operations

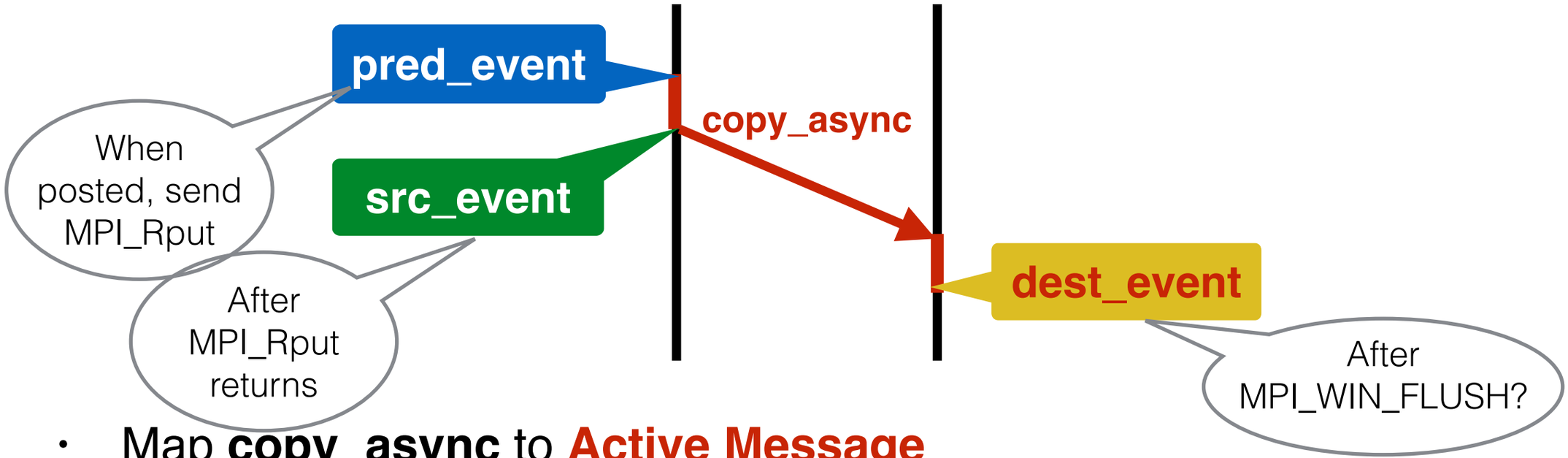
- **copy_async(dest, src, dest_event, src_event, pred_event)**



- Map **copy_async** to **Active Message**
  - MPI does not have AM support
- Map **copy_async** to **MPI_RPUT** (or **MPI_RGET**)

CAF 2.0 Asynchronous Operations

- **copy_async(dest, src, dest_event, src_event, pred_event)**



- Map **copy_async** to **Active Message**
  - MPI does not have AM support
- Map **copy_async** to **MPI_RPUT** (or **MPI_RGET**)
  - No asynchronous synchronization operation in MPI

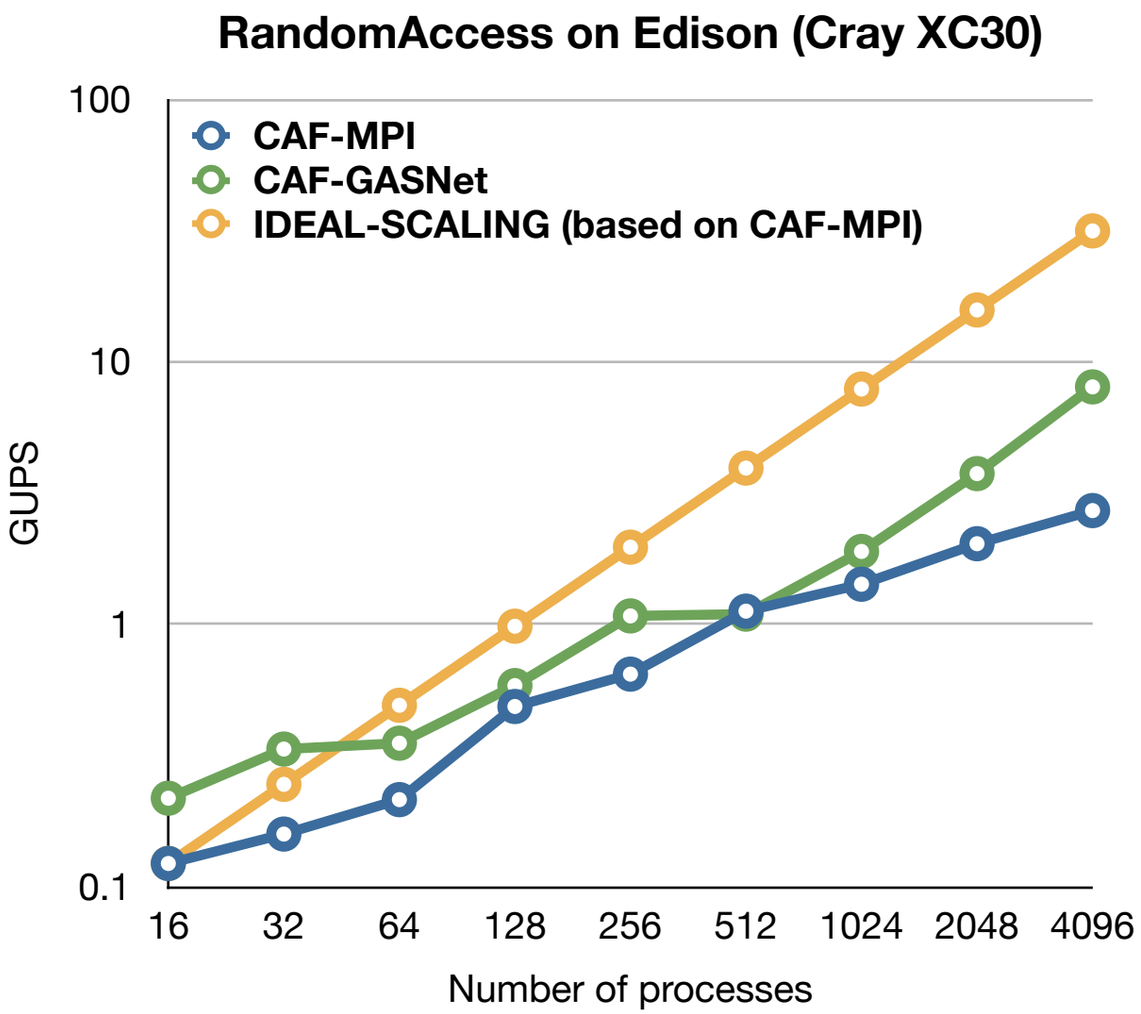
Evaluation

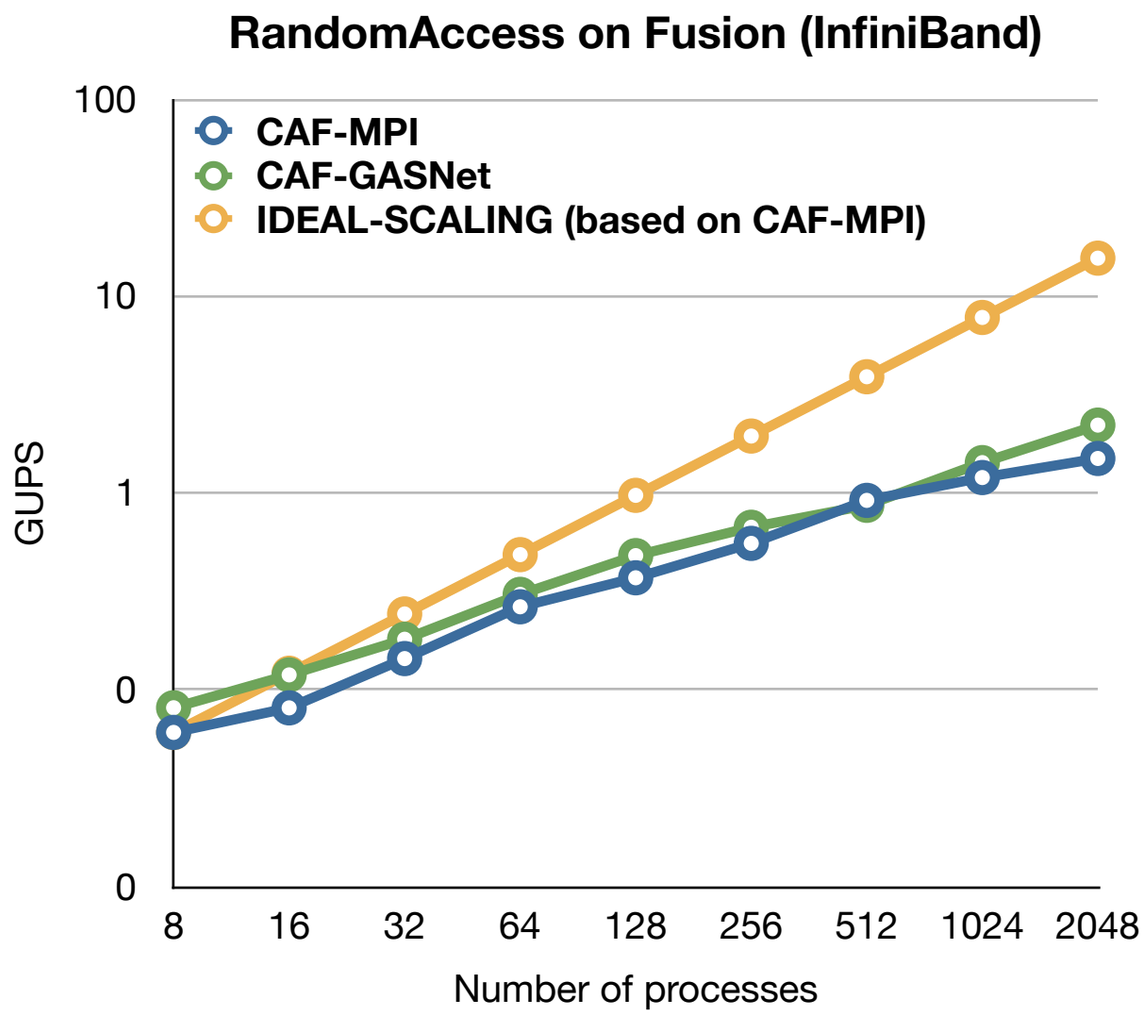
- 2 machines
  - Cluster (InfiniBand) and Cray XC30
- 3 benchmarks and 1 mini-app
  - **RandomAccess**, **FFT**, **HPL**, and **CGPOP**
- 2 implementations
  - **CAF-MPI** and **CAF-GASNet**

System	Nodes	Cores / Node	Memory / Node	Interconnect	MPI Version
Cluster (Fusion)	320	2x4	32GB	InfiniBand QDR	MVAPICH2-1.9
Cray XC30 (Edison)	5,200	2x12	64GB	Cray Aries	CRAY MPI-6.0.2

RandomAccess

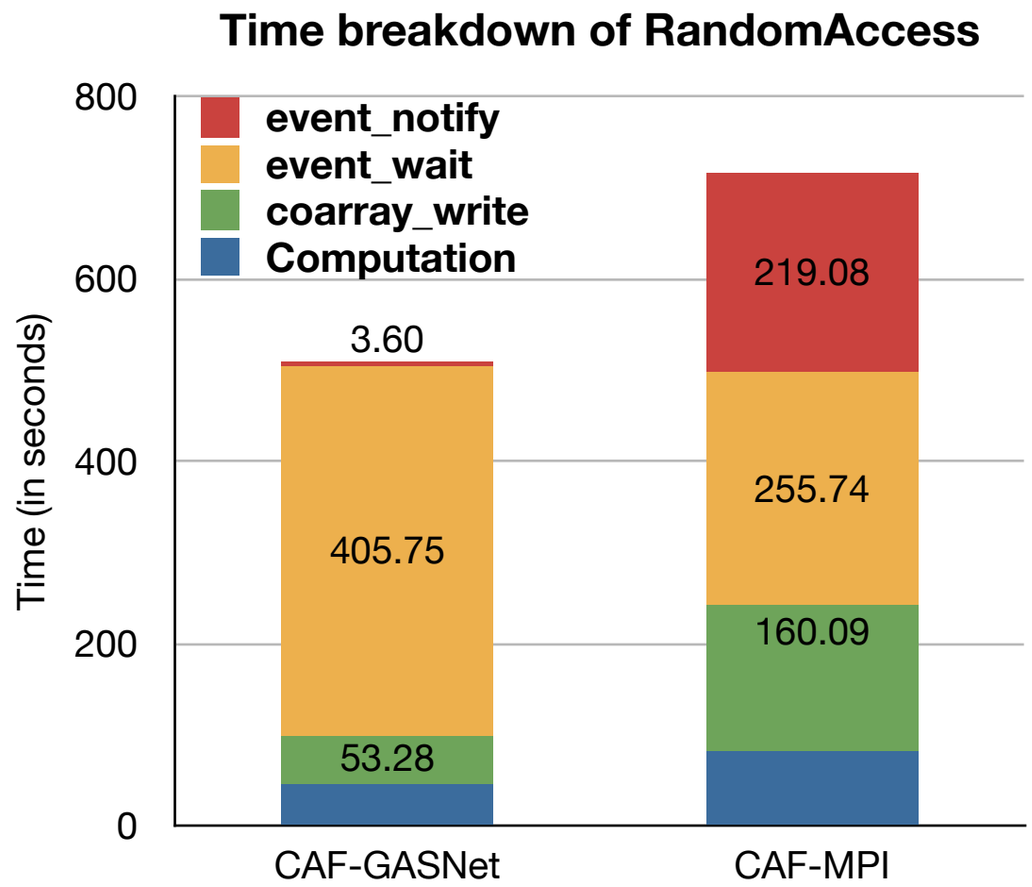
"Measures worst case system throughput"





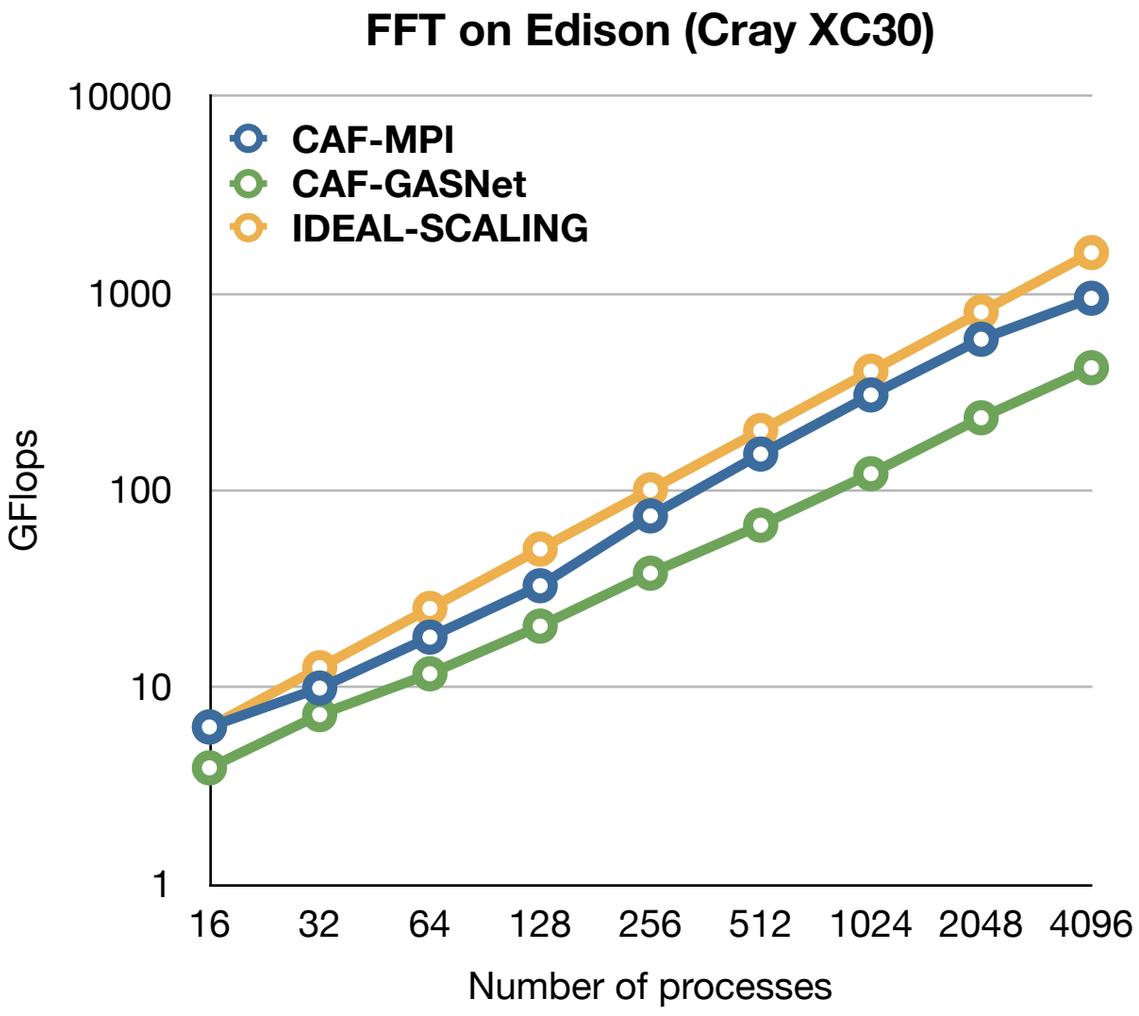
Performance Analysis of RandomAccess

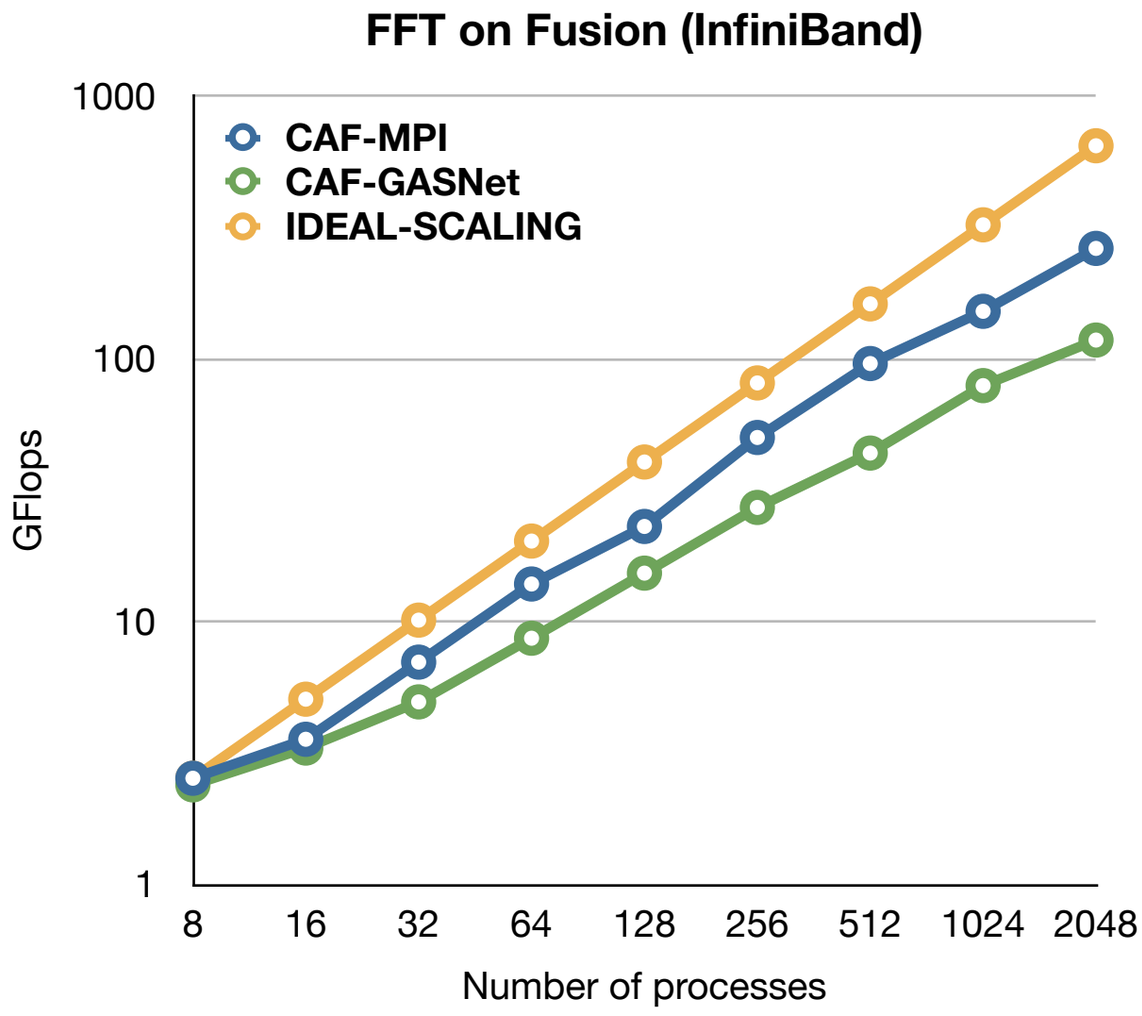
- The time spent in communication are about the same
- **event_notify** is slower in CAF-MPI because of **MPI_WIN_FLUSH_ALL**
- **MPI_WIN_FLUSH_ALL** performs **MPI_WIN_FLUSH** one by one



FFT

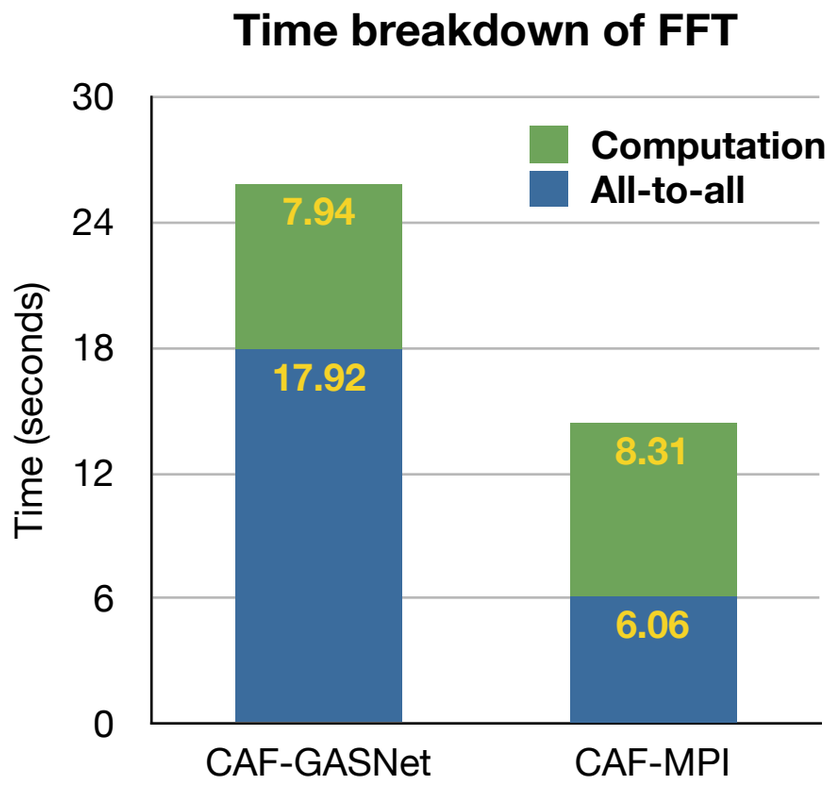
"Measures all-to-all communication"





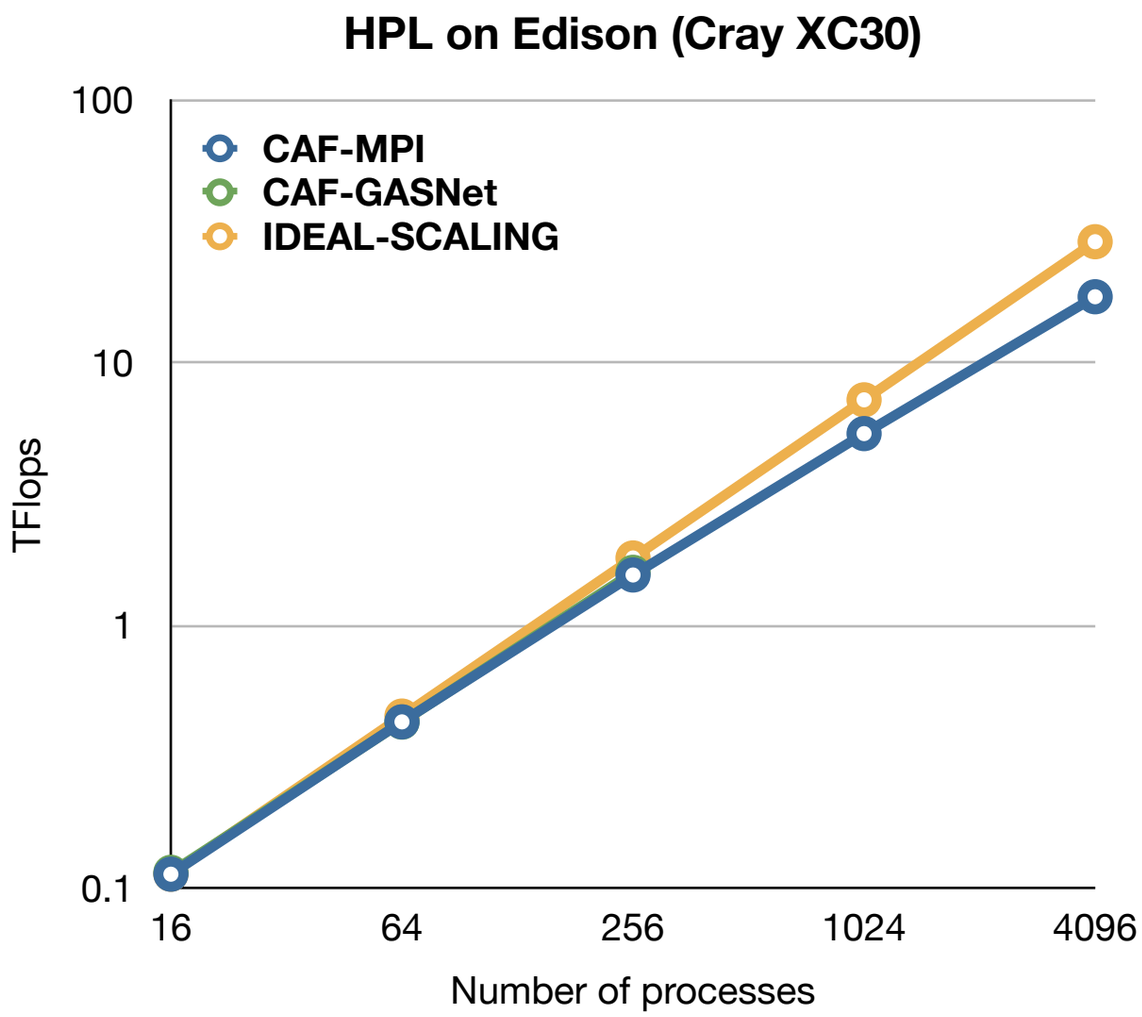
Performance Analysis of FFT

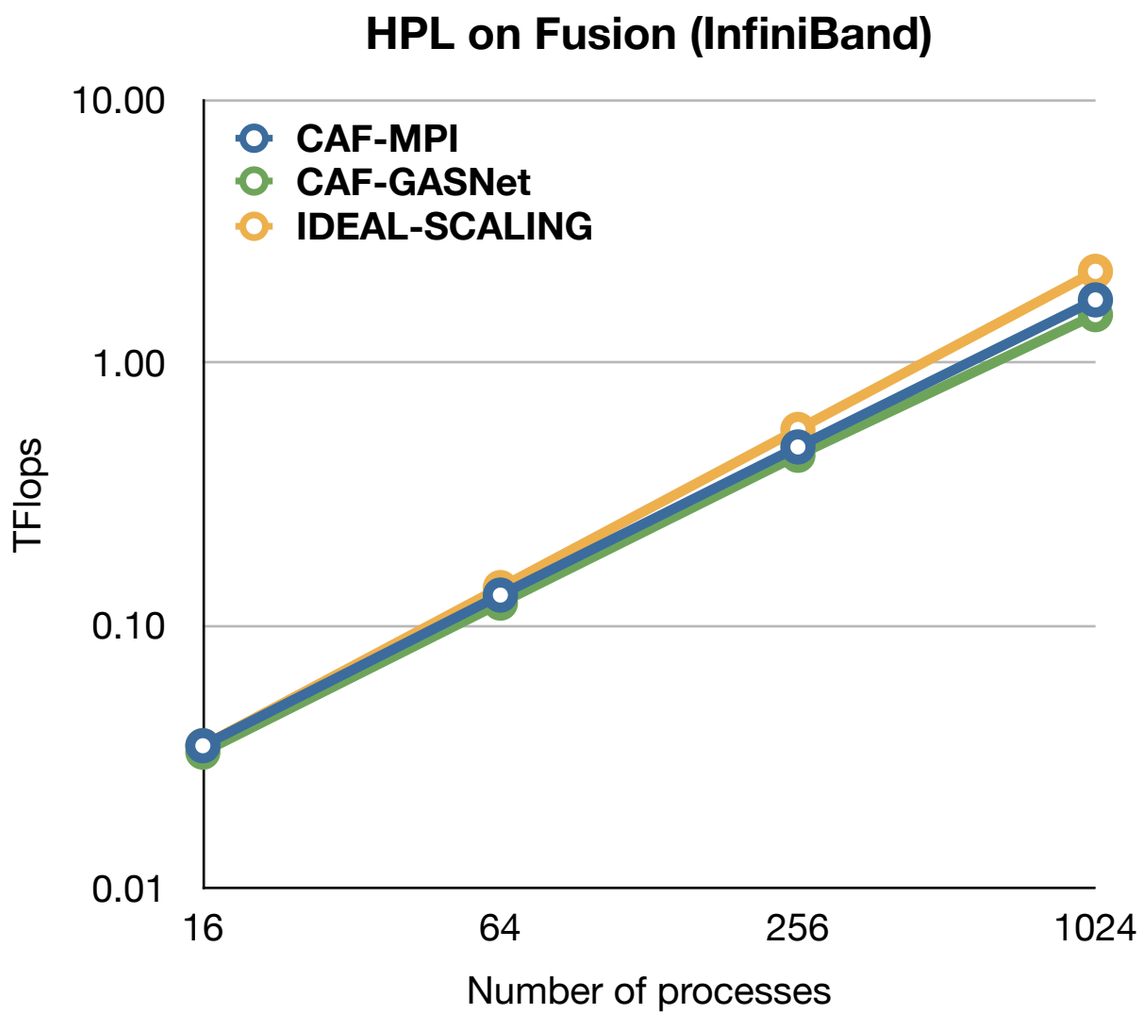
- The CAF 2.0 version of FFT solely uses ALLtoALL for communication
- CAF-MPI performs better because of fast all-to-all implementation



High Performance Linpack

"computation intensive"





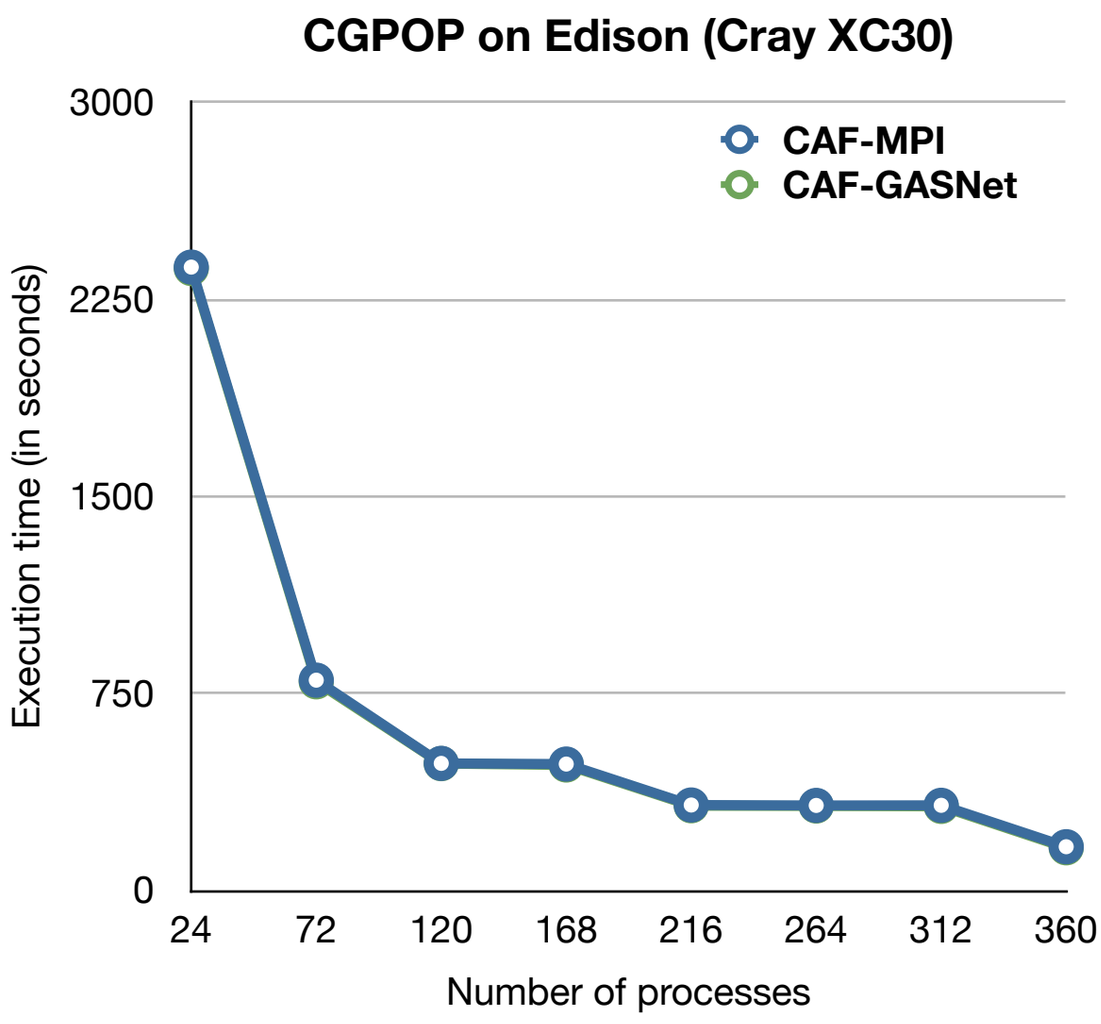
CGPOP

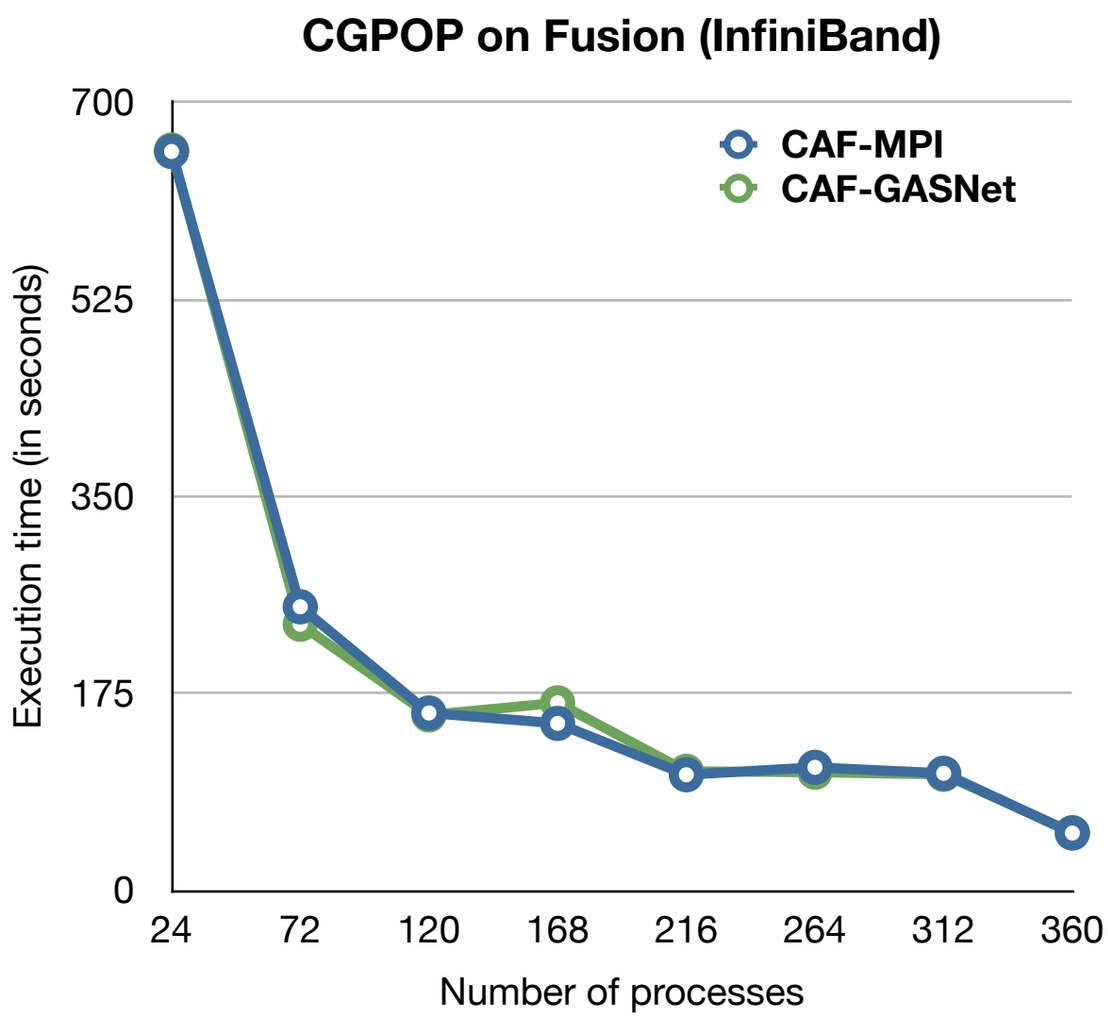
"A CAF+MPI hybrid application"

- The conjugate gradient solver from LANL Parallel Ocean Program 2.0
  - performance bottleneck of the full POP 2.0 application
- Performs linear algebra computation interspersed with two comm. steps:
  - **GlobalSum**: a 3-word vector sum (MPI_Reduce)
  - **UpdateHalo**: boundary exchange between neighboring subdomains (CAF)

Andrew I Stone, John M. Dennis, Michelle Mills Strout, "Evaluating Coarray Fortran with the CGPOP Miniapp"

CGPOP


CGPOP on Edison (Cray XC30)
CAF-MPI
CAF-GASNet
Execution time (in seconds)
3000
2250
1500
750
0
24
72
120
168
216
264
312
360
Number of processes


CGPOP on Fusion (InfiniBand)
CAF-MPI
CAF-GASNet
Execution time (in seconds)
700
525
350
175
0
24
72
120
168
216
264
312
360
Number of processes

Conclusions

- The benefits of building runtime systems on top of MPI
  - Interoperable with numerous MPI based libraries (Fortran 2008)
  - Deliver performance comparable to runtimes built with GASNet
  - MPI's rich interface is time-saving
- What current MPI RMA lacks
  - **MPI_WIN_RFLUSH** - overlap synchronization with computation
  - **Active Messages** - full interoperability