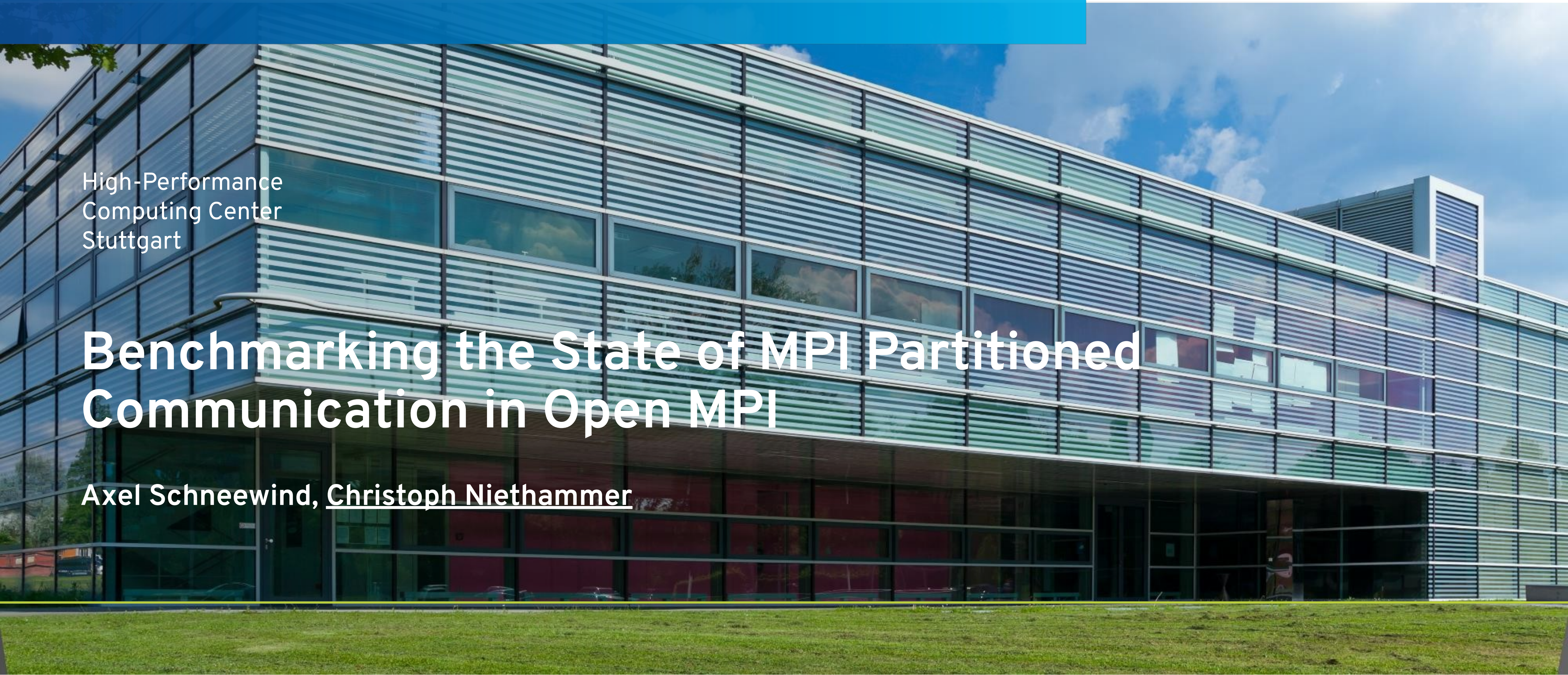


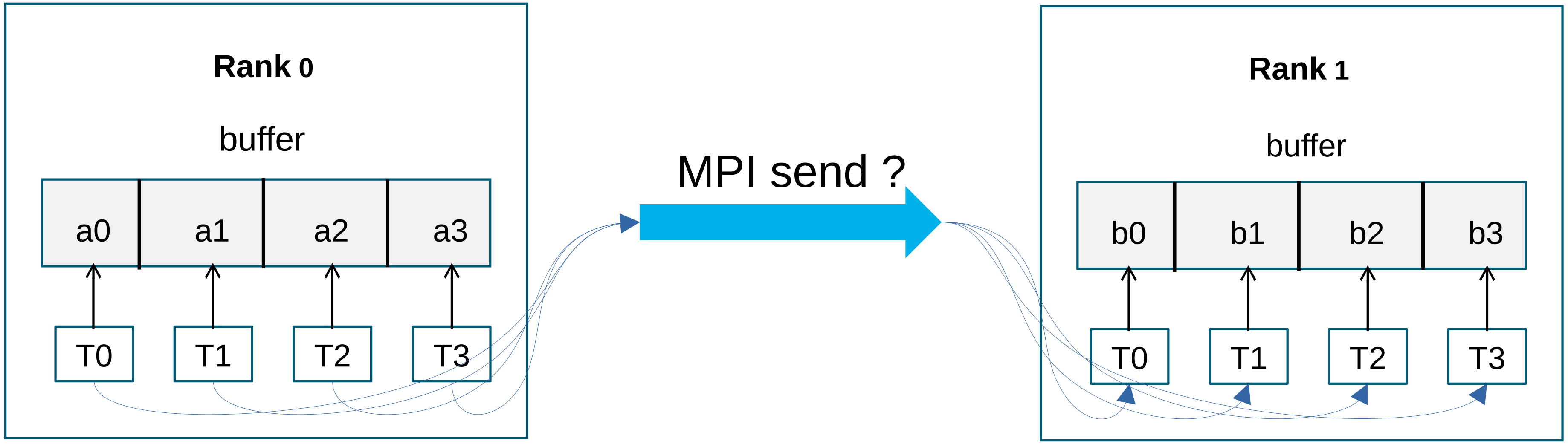
High-Performance
Computing Center
Stuttgart

Benchmarking the State of MPI Partitioned Communication in Open MPI

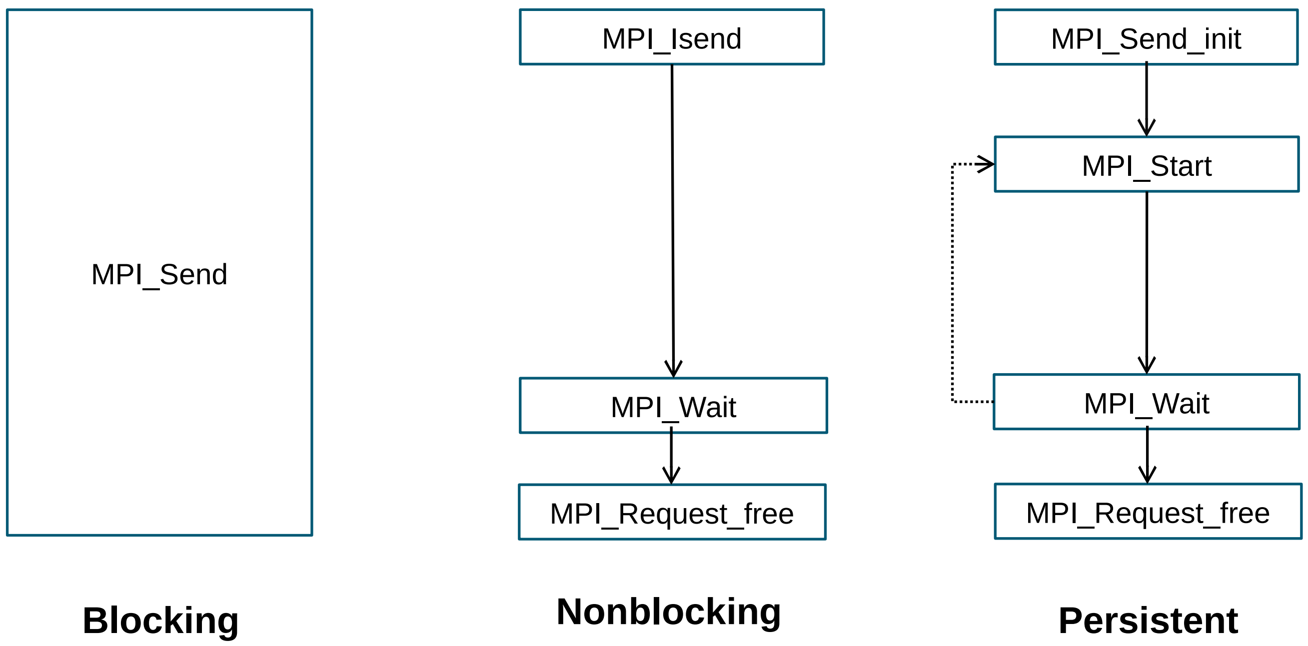
Axel Schneewind, Christoph Niethammer

Motivation: Hybrid Programming

In hybrid MPI+X programs send buffers are filled and consumed by multiple „X“ (e.g. threads)



MPI: Transfer mechanisms on the sender side



MPI-4.1: Restrictions for blocking/nonblocking send operations

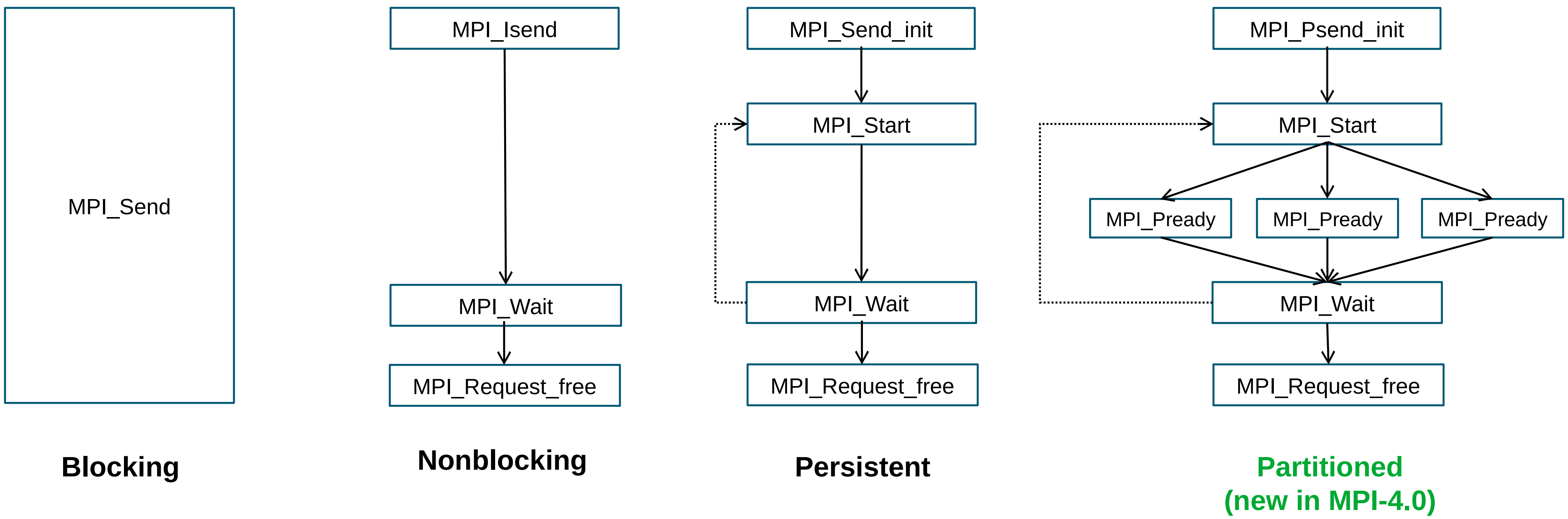
HLRS

Order (3.5). Messages are *nonovertaking*: If a sender sends two messages in succession to the same destination, and both match the same receive, then this operation cannot receive the second message if the first one is still pending. [analogously for receive operations]

Order (3.7.4). Nonblocking communication operations are *ordered* according to the execution order of the calls that initiate the communication.

Progress (3.7.4).
A call to MPI_WAIT that completes a receive will eventually terminate and return if a matching send has been started, unless the send is satisfied by another receive. [analogously for sender]

MPI: Transfer mechanisms on the sender side



- No such restrictions for MPI_Start/MPI_Pready!
- Reducing number of necessary completion tests (MPI_Test / MPI_Wait)
- Calls to MPI_Pready might cause earlier message progress ...

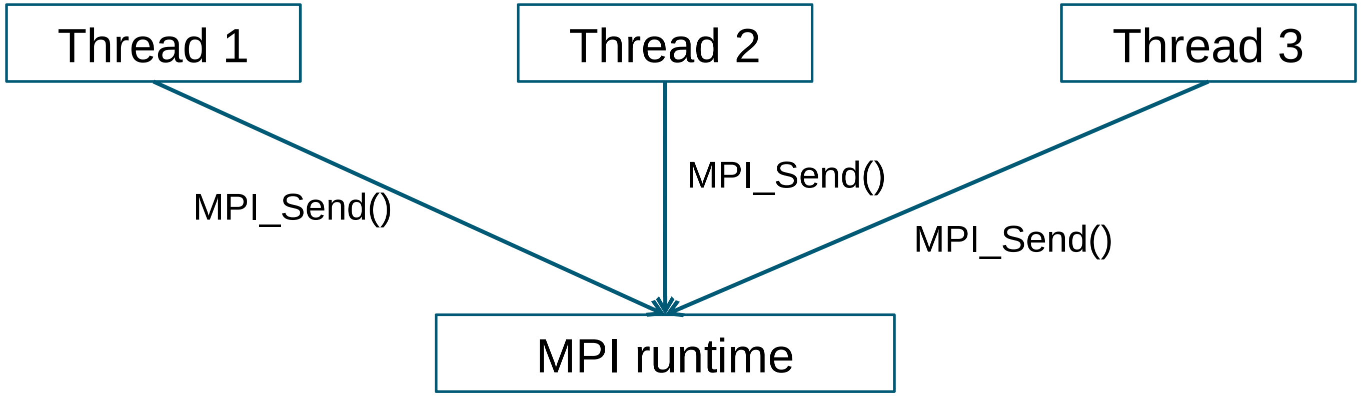
MPI-4.1: Partitioned Communication

Rationale. Partitioned communication is designed to provide opportunities for MPI implementations to optimize data transfers.

MPI is free to choose **how many transfers** to do within a partitioned communication send independent of how many partitions are reported as ready to MPI through MPI_PREADY calls.
Aggregation of partitions is permitted but not required.
Ordering of partitions is permitted but not required.

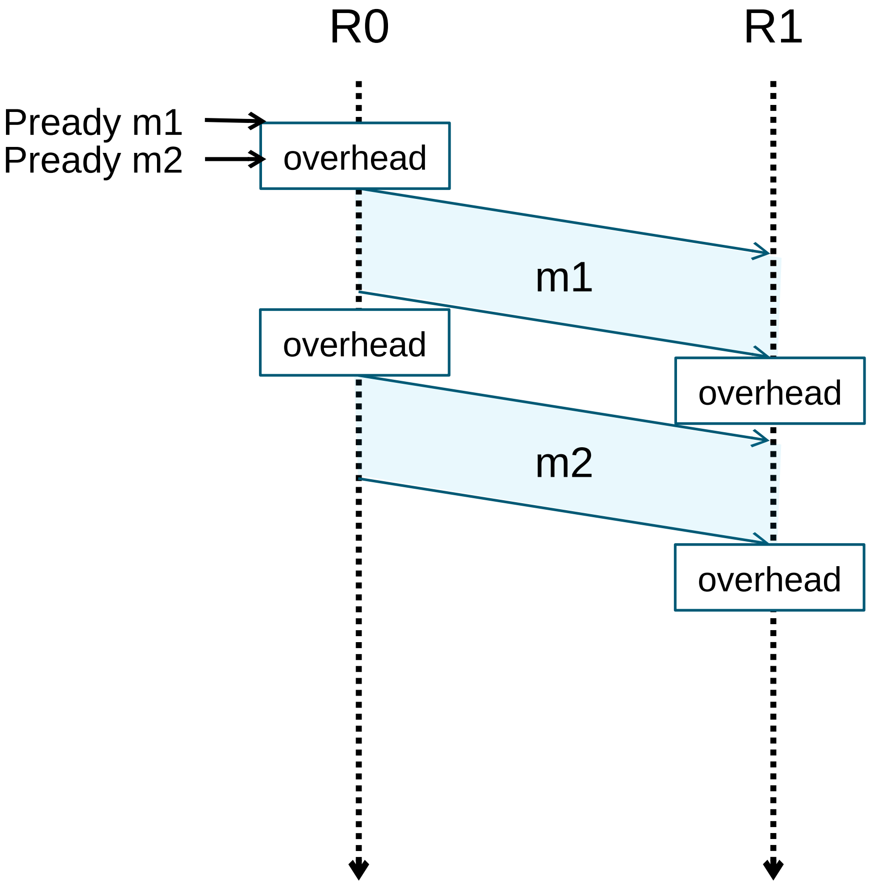
A naive implementation can simply wait for the entire message buffer to be marked ready before any transfer(s) occur and could wait until the completion function is called on a request before transferring data. [...]. (*End of rationale.*)

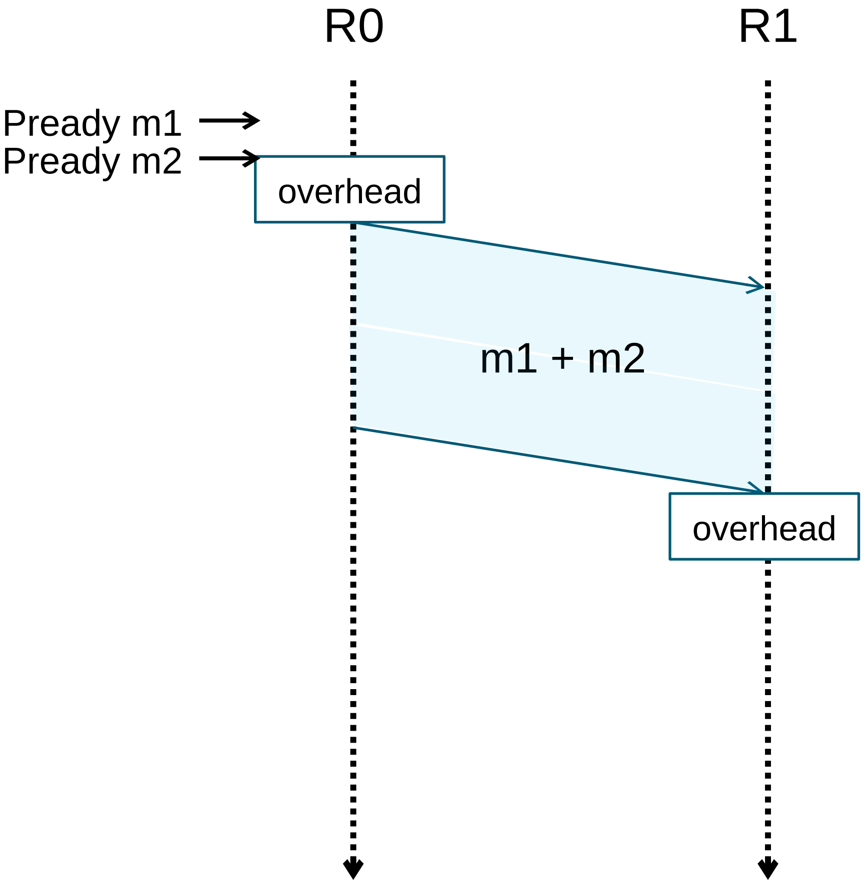
Possible performance benefits – Less locking overhead



- Pready not required to initiate individual sends with all their overheads and locking needs in the MPI runtime

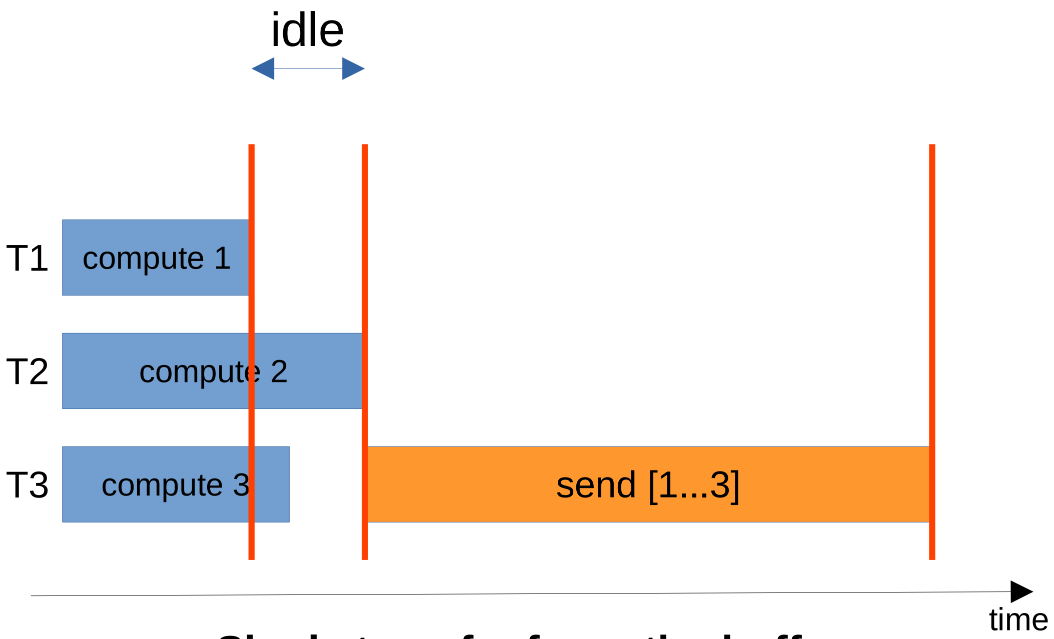
Possible optimization – Message aggregation



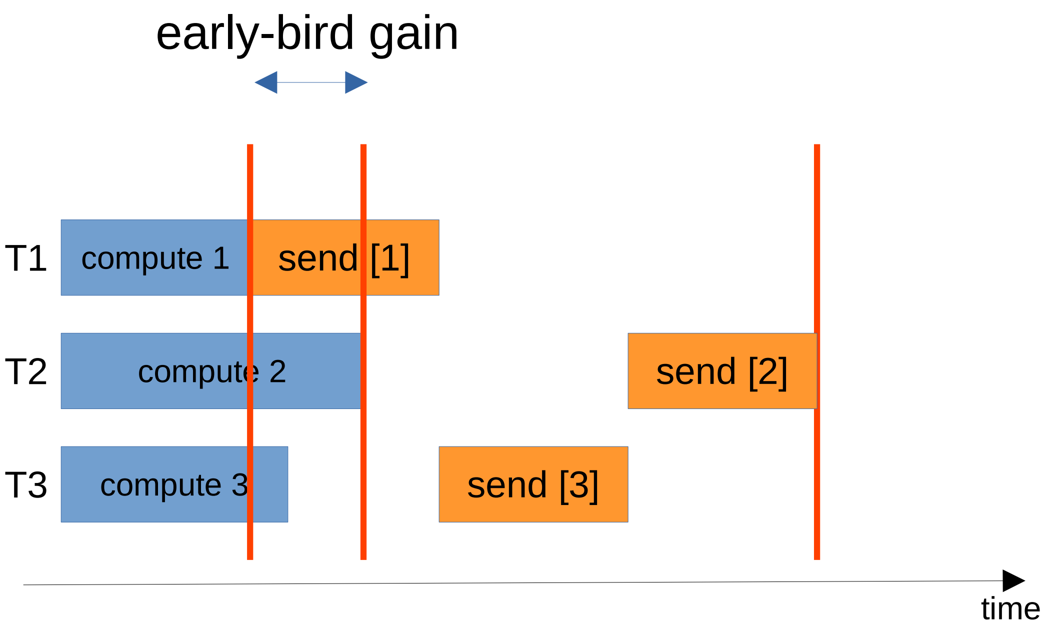


- Pready allows aggregation

Possible performance benefits – Early-bird effect



Single transfer for entire buffer



One transfer per partition

- Pready allows early-bird with low-overhead individual transfers

MPI-4.1: Partitioned Communication

H L R S

Rationale. Partitioned communication is designed to provide opportunities for MPI implementations to optimize data transfers.

MPI is free to choose **how many transfers** to do within a partitioned communication send independent of how many partitions are reported as ready to MPI through MPI_PREADY calls.
Aggregation of partitions is permitted but not required.
Ordering of partitions is permitted but not required.

A naive implementation can simply wait for the entire message buffer to be marked ready before any transfer(s) occur and could wait until the completion function is called on a request before transferring data. [...]. (*End of rationale.*)

So, let's see how this holds up in reality 3 years after MPI 4.0 came out …

Goals for our benchmarking

HLRS

Measure **effective bandwidth** depending on

- Partition size
- Transfer mechanism
- Varying thread counts for partition marking
- Varying orders of marking partitions as ready

Benchmarking scheme

```
# initialization

for i in 0...num_iterations:
    MPI_Barrier()   # synchronization

    start_time[i] = MPI_Wtime()

    for p in 0...partition_count in some order:
        send/receive partition p        # start transfer

    MPI_Wait(request)           # completion

    end_time[i] = MPI_Wtime()

# cleanup

bandwidth = buffer_size / mean(start_time - end_time)
```

Transfer mechanisms

HLRS

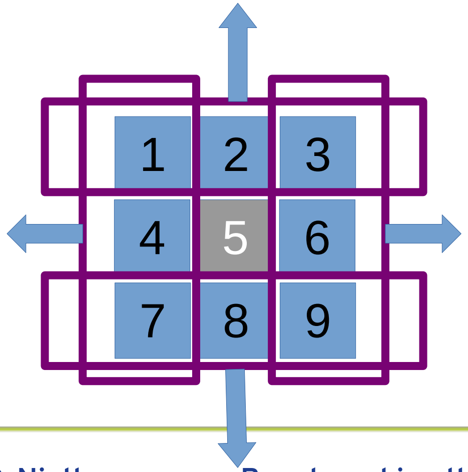
Mechanism	Initialization	Partition ready	Completion	Freeing
Send	-	MPI_Send (p)	-	-
Persistent	MPI_Send_init (p)	MPI_Start (p)	MPI_Waitall	MPI_Request_free (p)
Isend	-	MPI_Isend (p)	MPI_Waitall	MPI_Request_free (p)
Psend	MPI_Psend_init	MPI_Pready (p)	MPI_Wait	MPI_Request_free

operations marked with (p) are performed for each partition

Partition ready marking orders

Partitions marked ready in different orders:

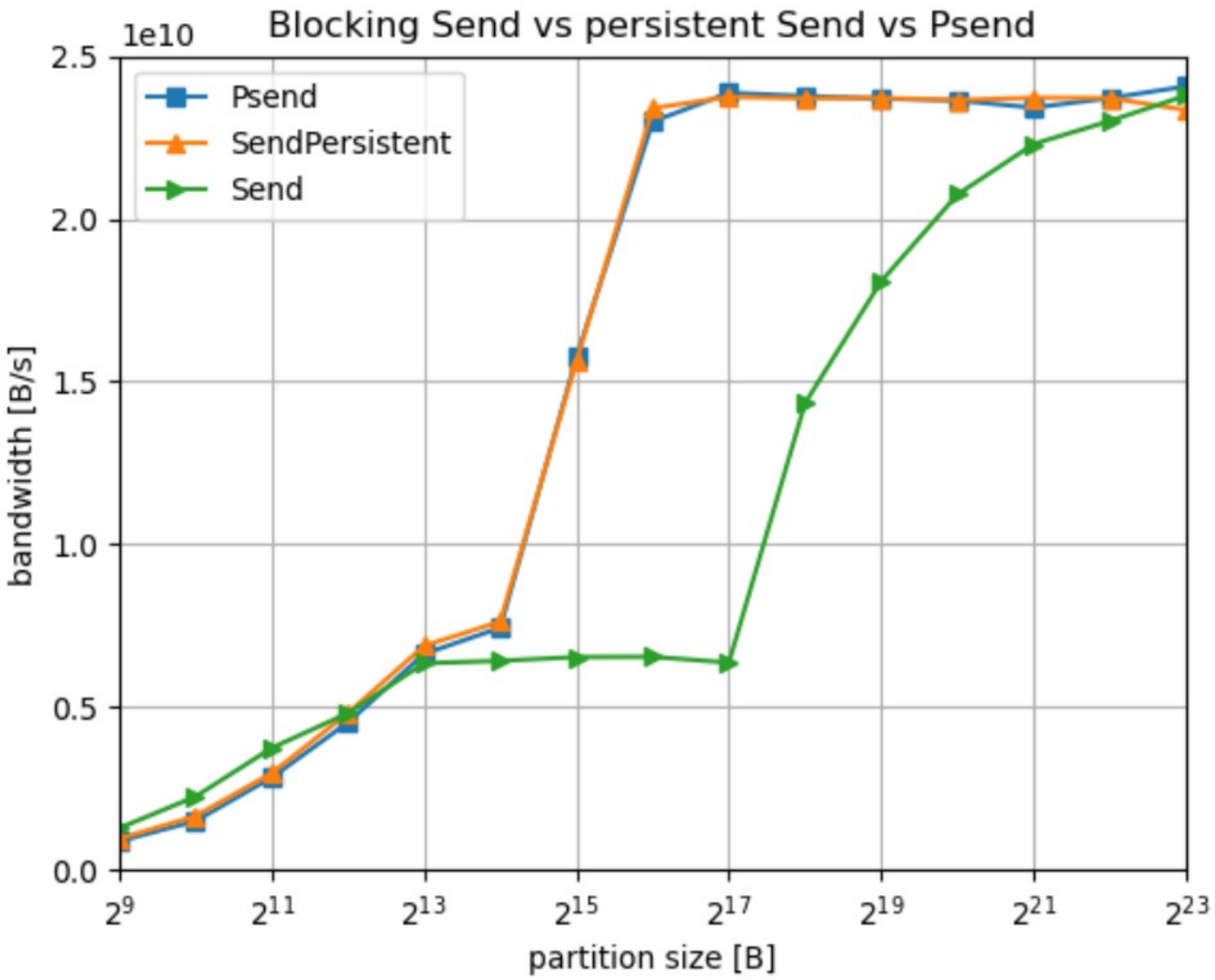
- Left-to-right: 1 2 3 4 5 6 7 8
- Stride: 1 5 2 6 3 7 4 8
- Randomized: 8 7 3 2 5 1 6 4
- Neighbourhood exchange pattern



Experimental setup

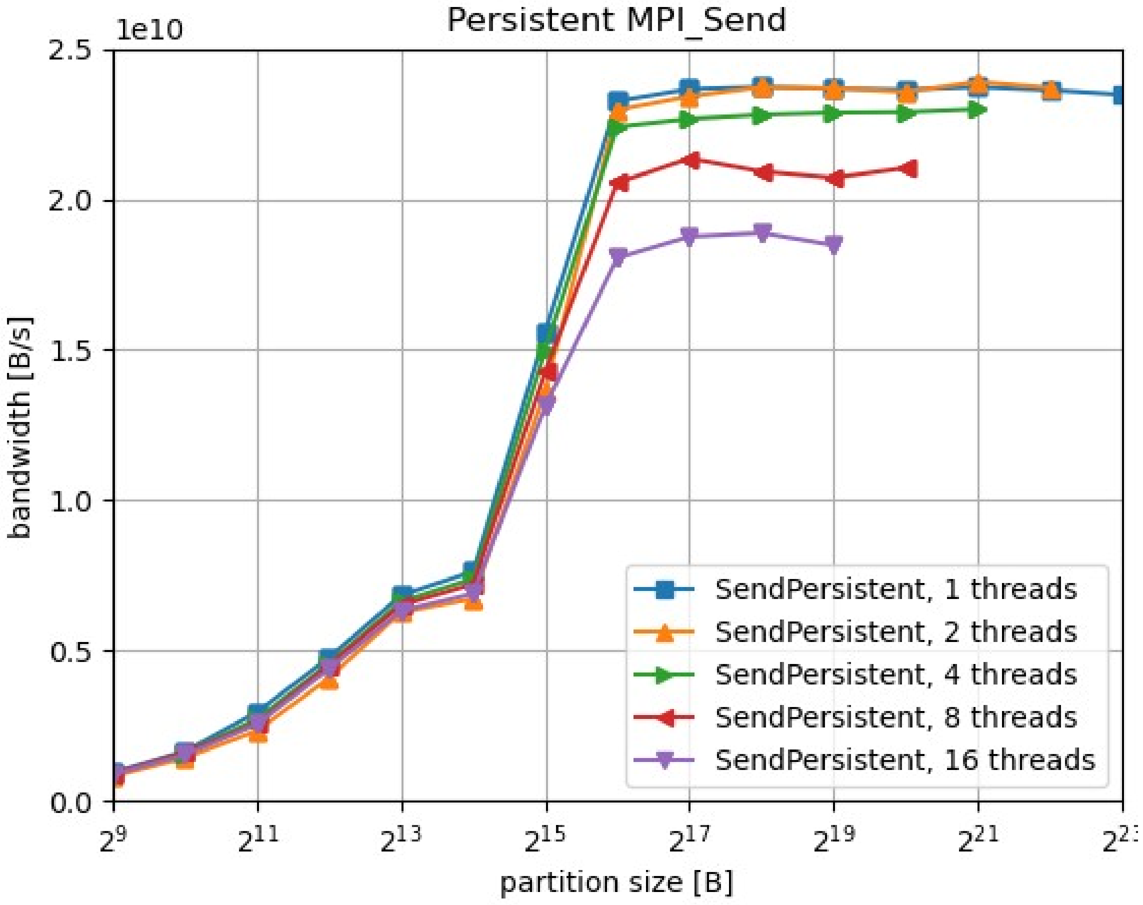
- 2 nodes on HLRS/HAWK with one MPI process each
  - Max. Bandwidth: 200Gbit/s = 25GB/s
- Multithreading with OpenMP (gcc)
- Open MPI 5.0.3
- **Buffer size:** 8MiB
- **Partition sizes**: 512B, 1024B, ..., 8MiB

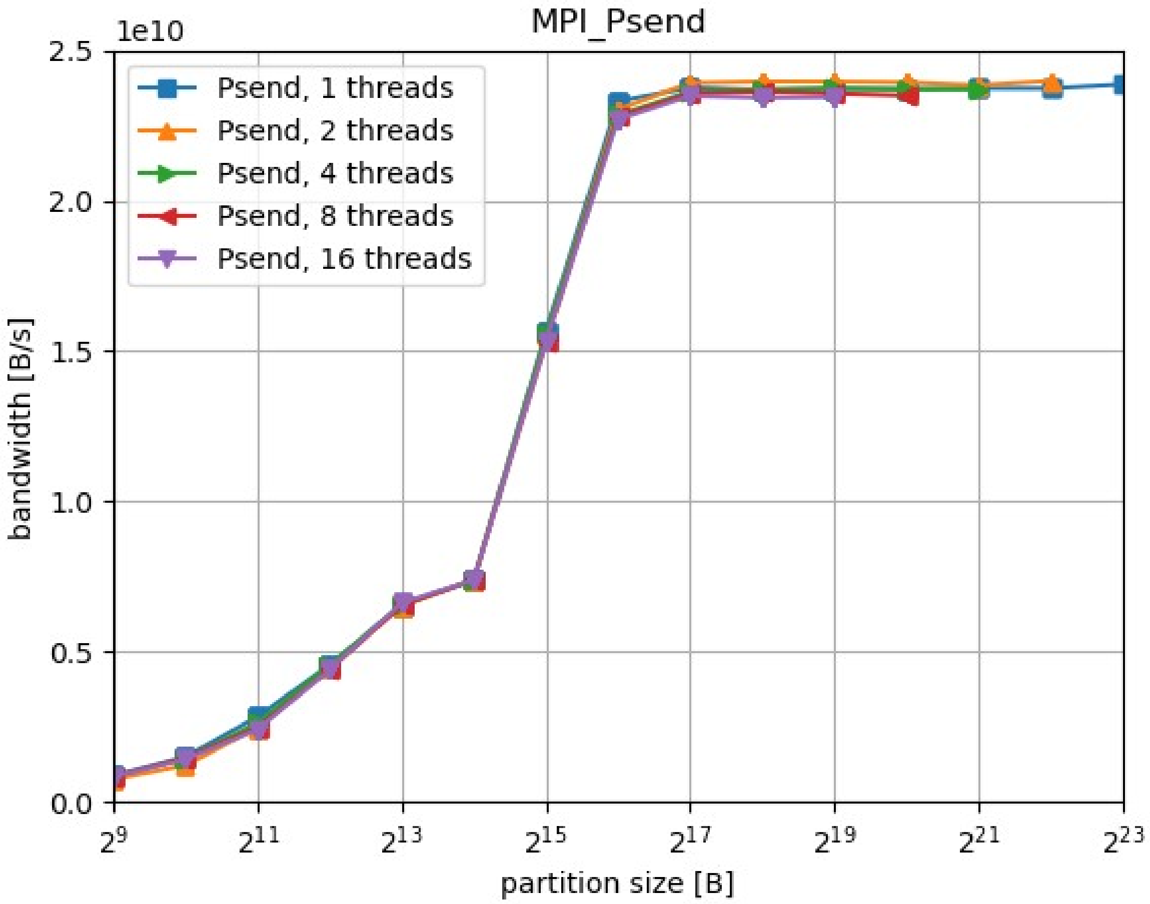
Results for pure MPI



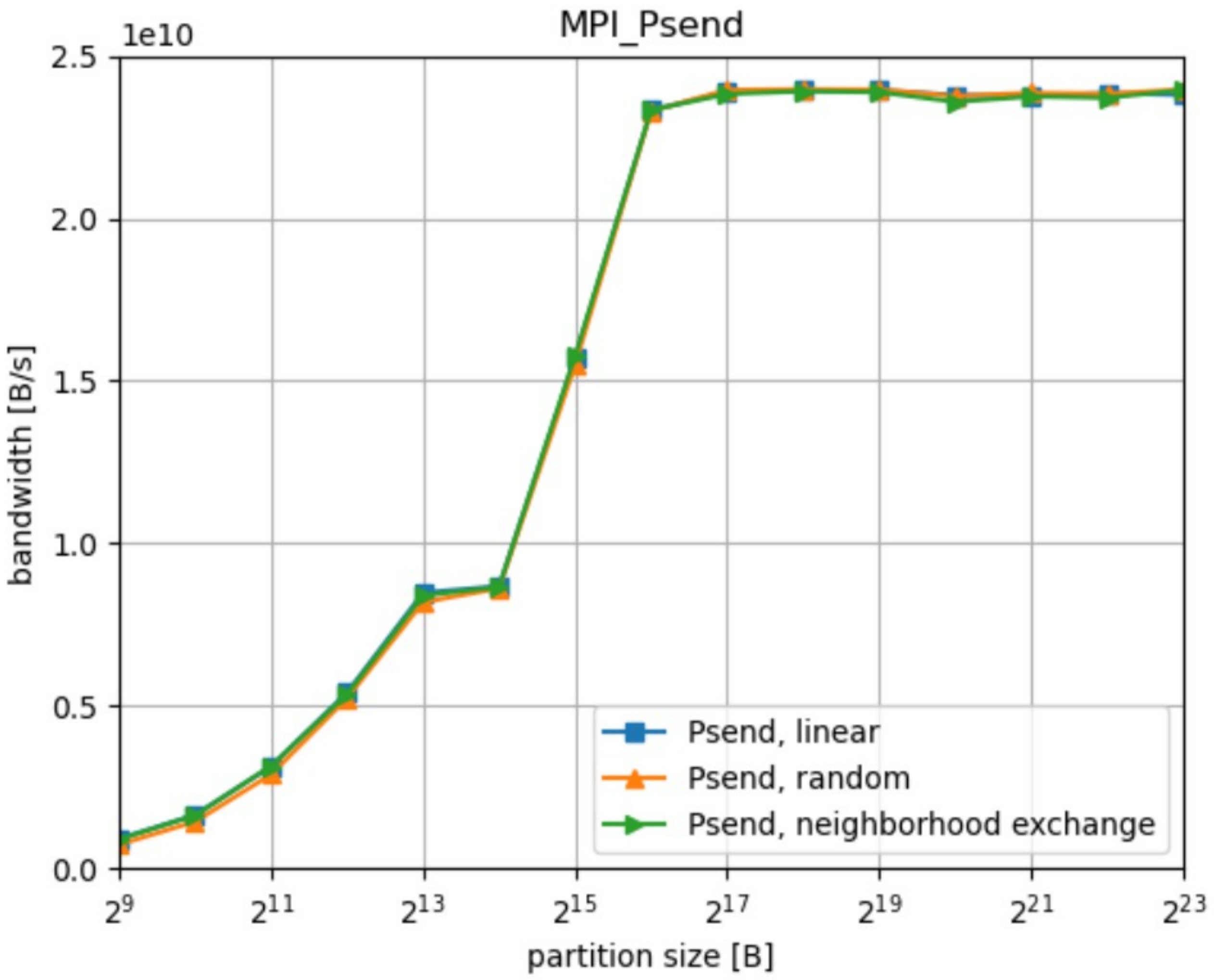
Results: Persistent send vs Psend (multithreaded)

H L R S

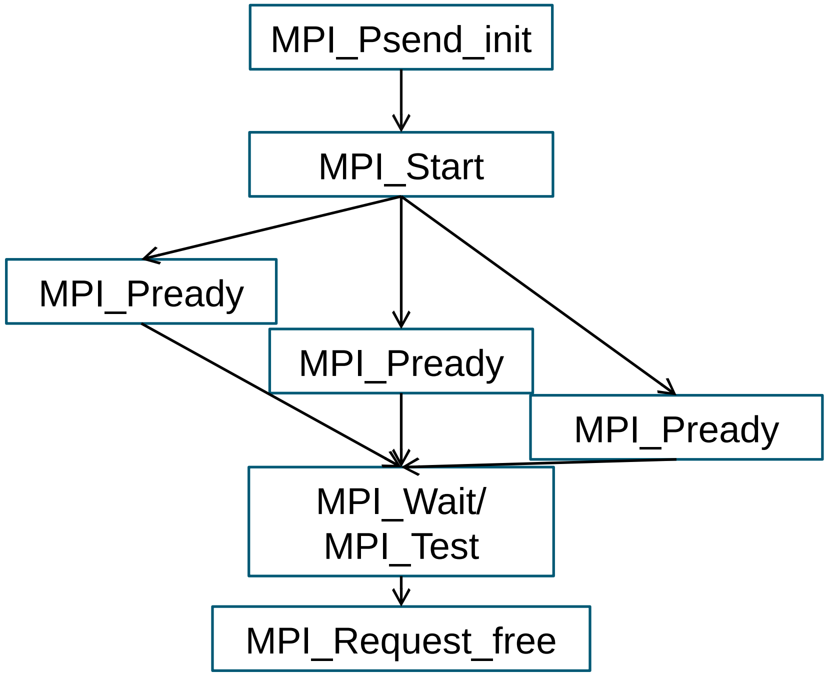




Results: Psend with different send patterns



Current implementations: OpenMPI/5.0.3

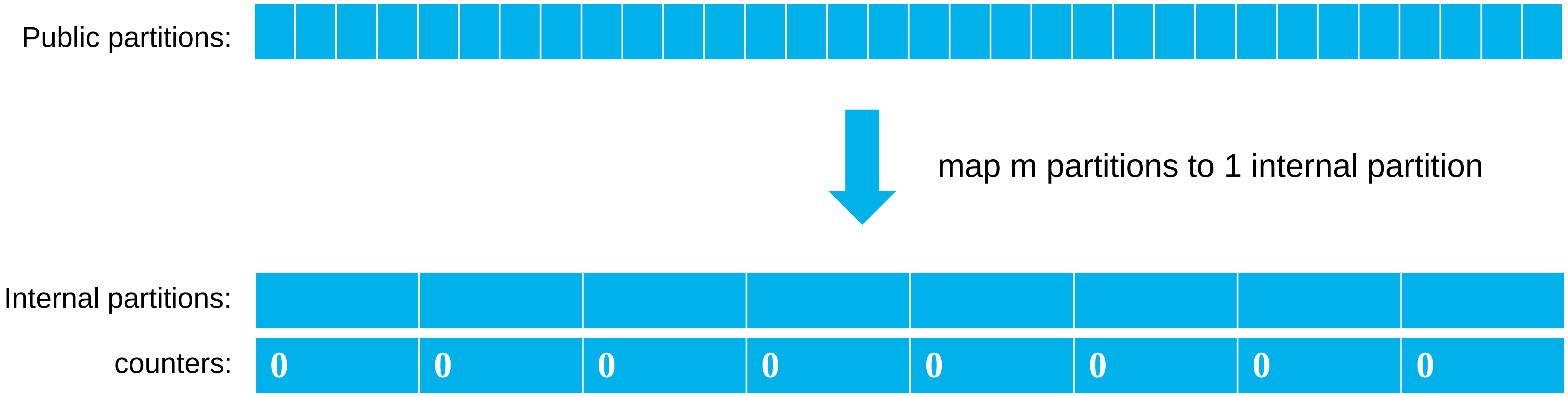



```
MPI_Start():
    for partition p:
        flag[p] = 0;

MPI_Pready(p):
    flag[p] = 1;

progress():
    for p with flag[p] == 1:
        isend(request[p]);
        flag[p] = 2;
```

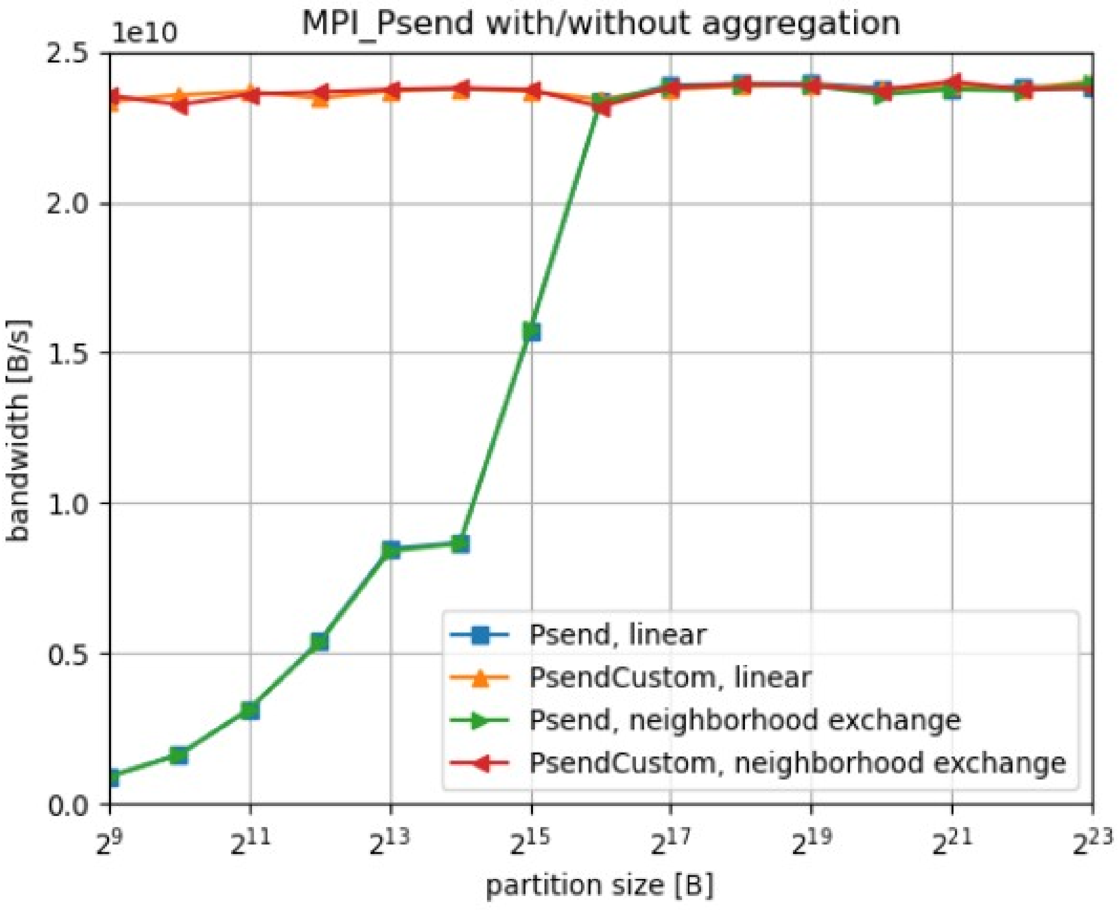
Simple message aggregation

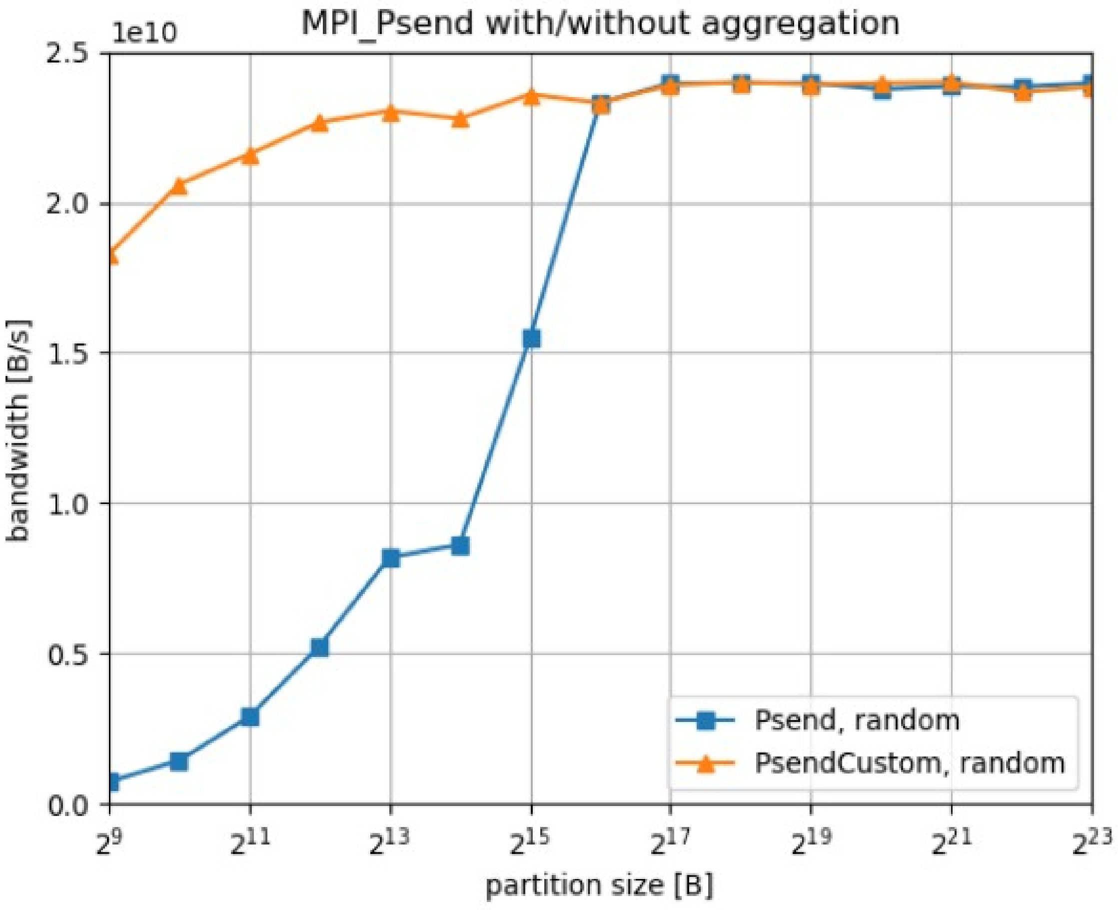



```
Pready(p):
    increment counter of internal partition of p
    if counter >= m:
        mark internal partition as ready
```

Results: With simple message aggregation

(Internal partition size 64kB)





Conclusion and outlook

- OpenMPI:
  - Psend performs similar to persistent Send when single-threaded
  - Psend performs better in multi-threaded case
  - Send pattern does not influence performance
  - Psend implemented currently via nonblocking sends
  - ... no further optimizations such as message aggregation
- Message aggregation:
  - Mapping partitions to larger messages can improve performance for regular and random orderings
- What's next:
  - Implement our message aggregation into Open MPI
  - Test alternative aggregation schemes/algorithms

HLRS

High-Performance
Computing Center
Stuttgart

Thanks for your attention!

Axel Schneewind, Christoph Niethammer