# MPI Process Synchronization in Space and Time

J. Schuchart[1], S. Hunold[2], G. Bosilca[1]

EuroMPI'23

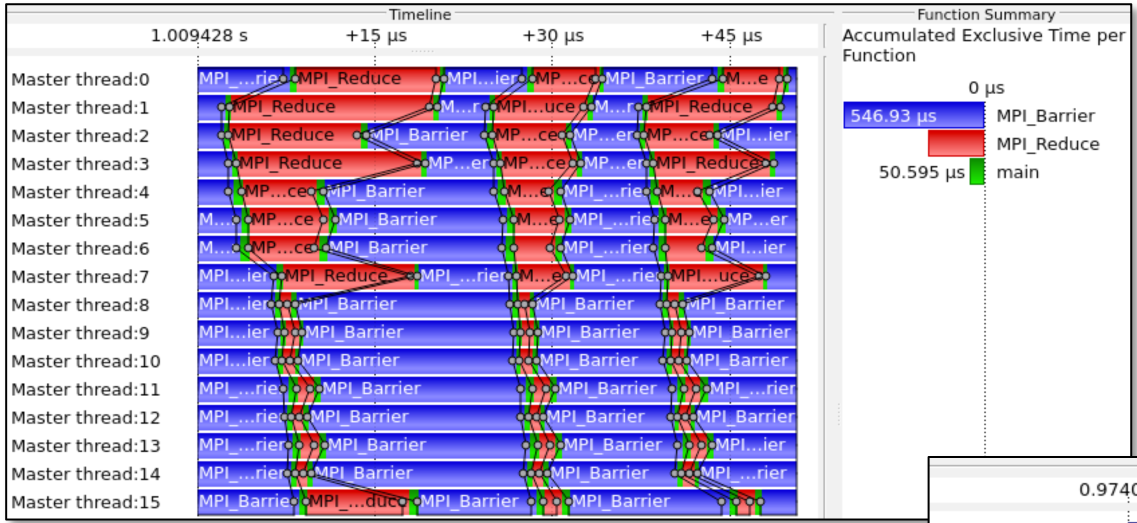
September 11-13, 2023

Bristol, UK

[1] Innovative Computing Laboratory, University of Tennessee, Knoxville, USA
[2] TU Wien, Vienna, Austria

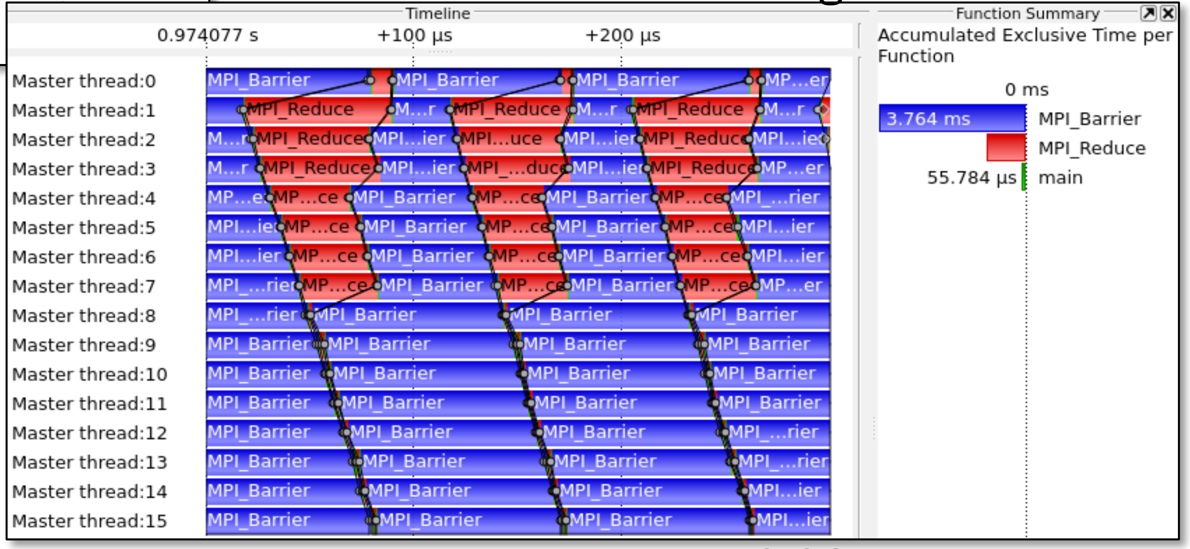
# A Tale of Two Barriers

Barrier: linear




Reported Latency: 2.78us

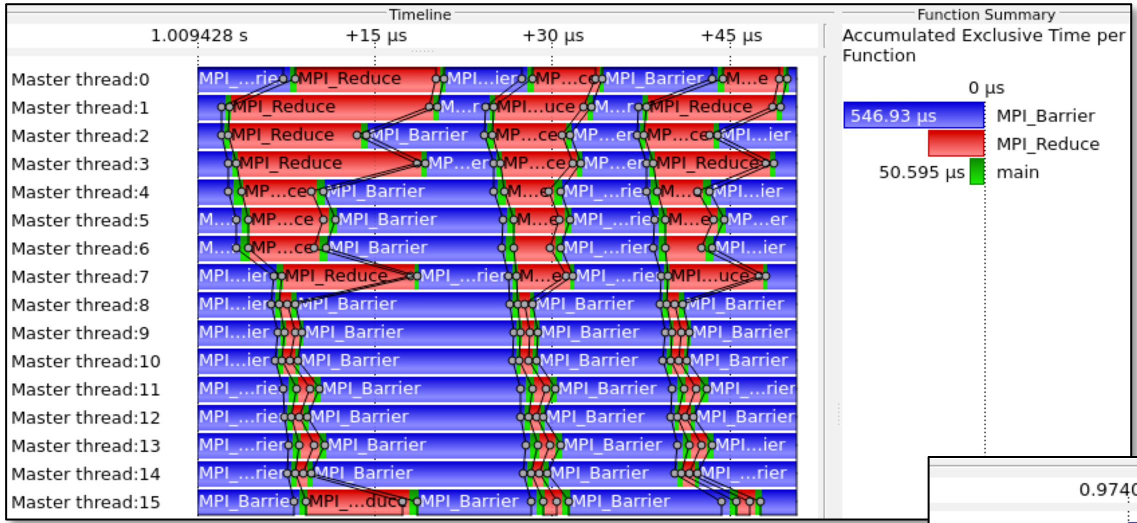
Barrier: double-ring




Reported Latency: 9.98us

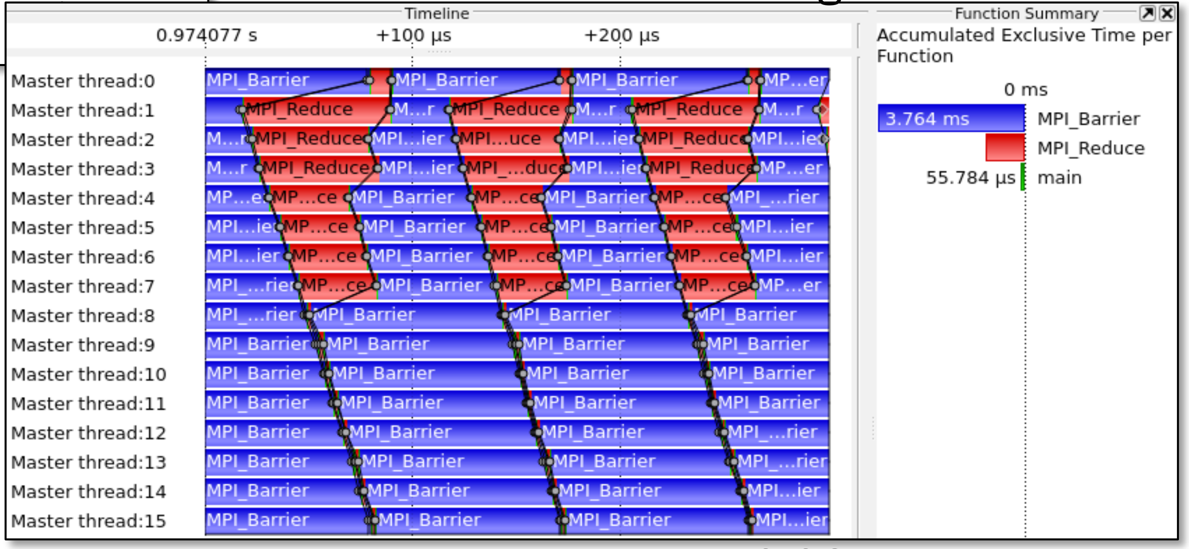
# A Tale of Two Barriers

Barrier: linear




Reported Latency: 2.78us

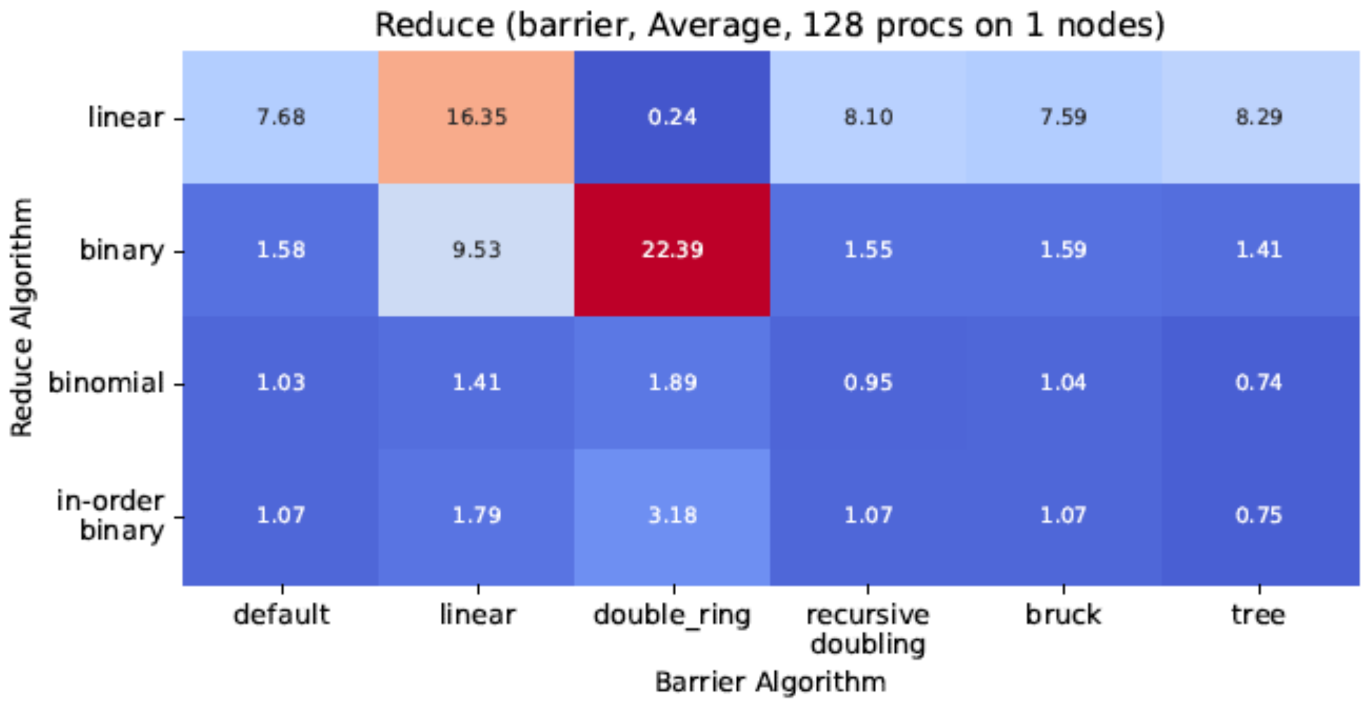
Barrier: double-ring



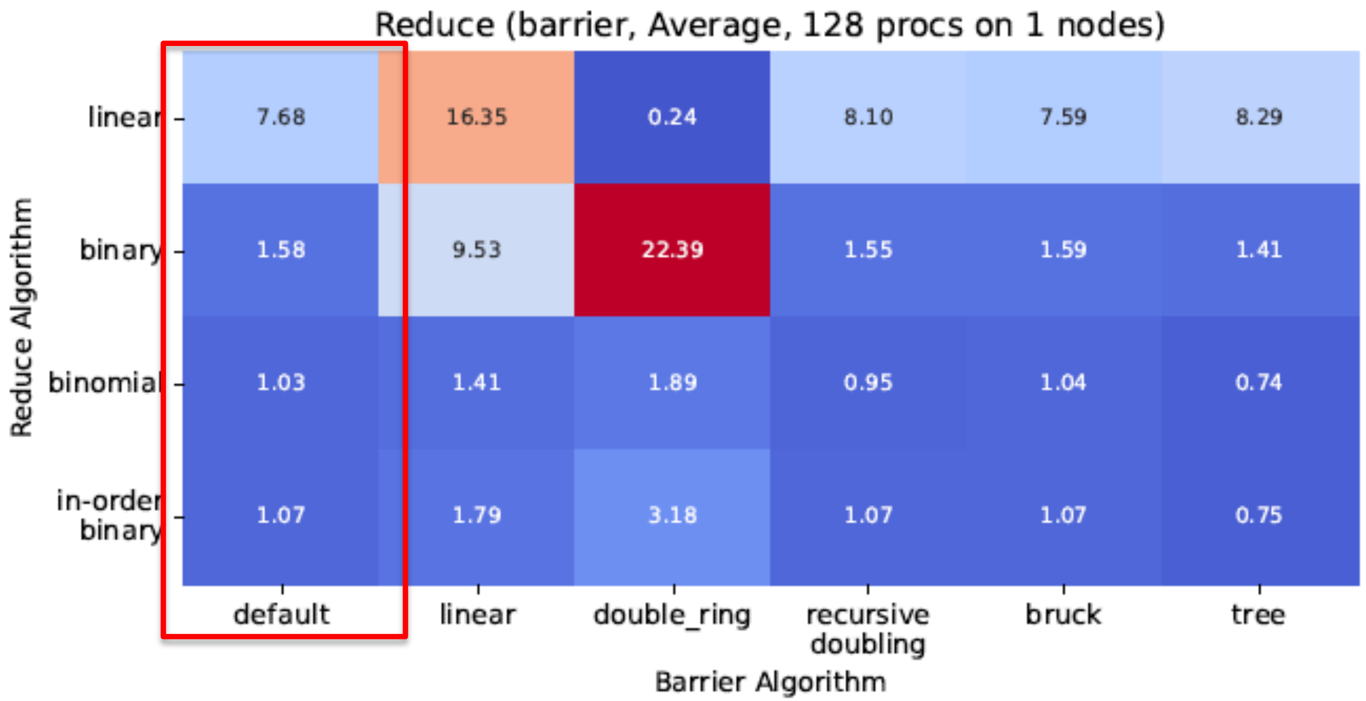

Reported Latency: 9.98us

Arbitrary
Arrival
Pattern

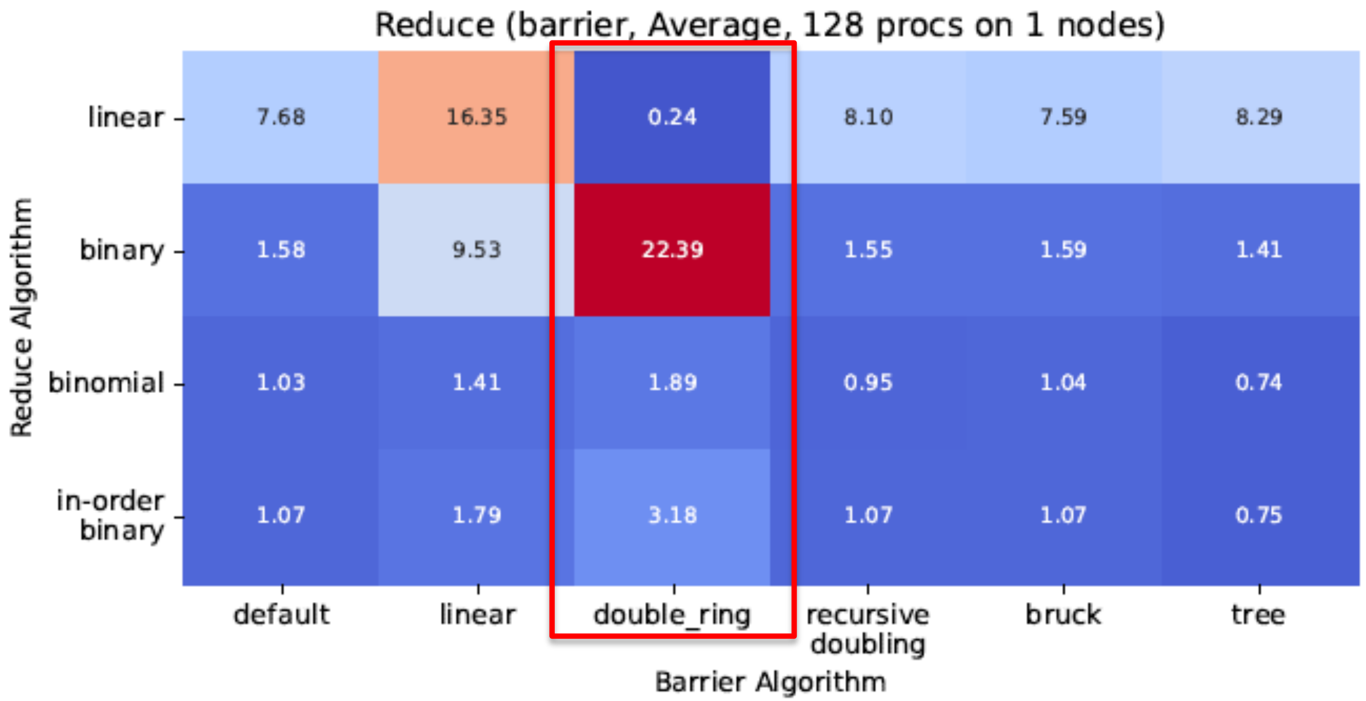
# Choice of Barrier Algorithms Matters



# Choice of Barrier Algorithms Matters



# Choice of Barrier Algorithms Matters

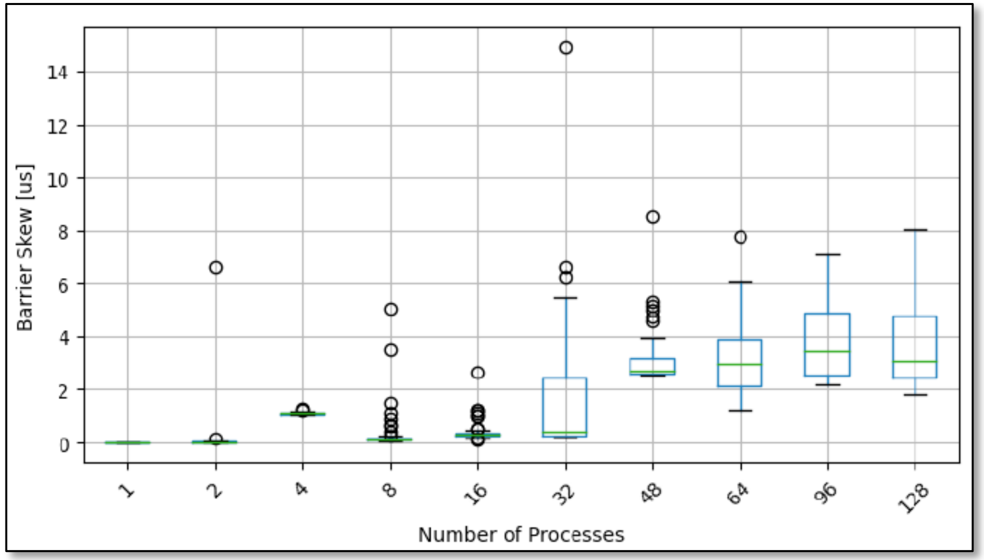


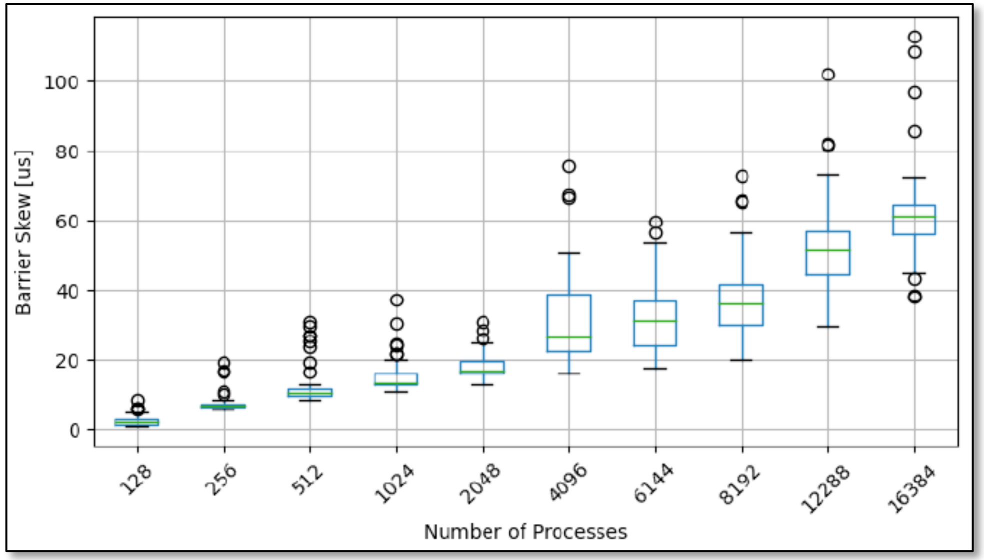
We have been benchmarking collective operations wrong for decades.

# Barriers only Synchronize in Space

$$\sigma = \max_{t_0 \ldots t_p} - \min_{t_0 \ldots t_p}$$

- Barrier Skew:
  difference between minimum and maximum barrier time
- Barriers do not guarantee any time synchronization
  - But that is what we want for benchmarks!






ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

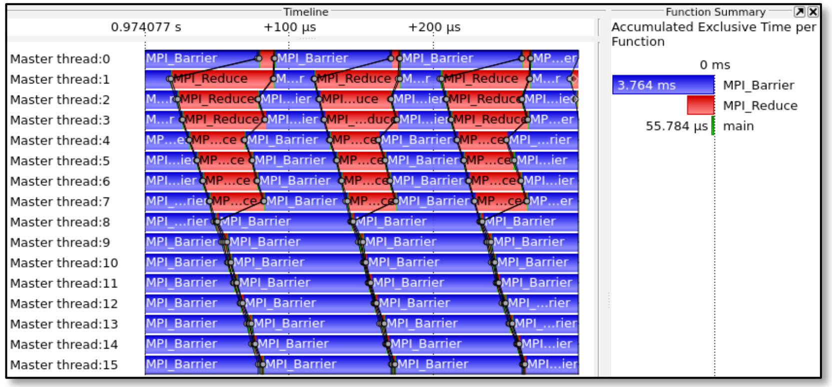
# What we care about

## What we care about:

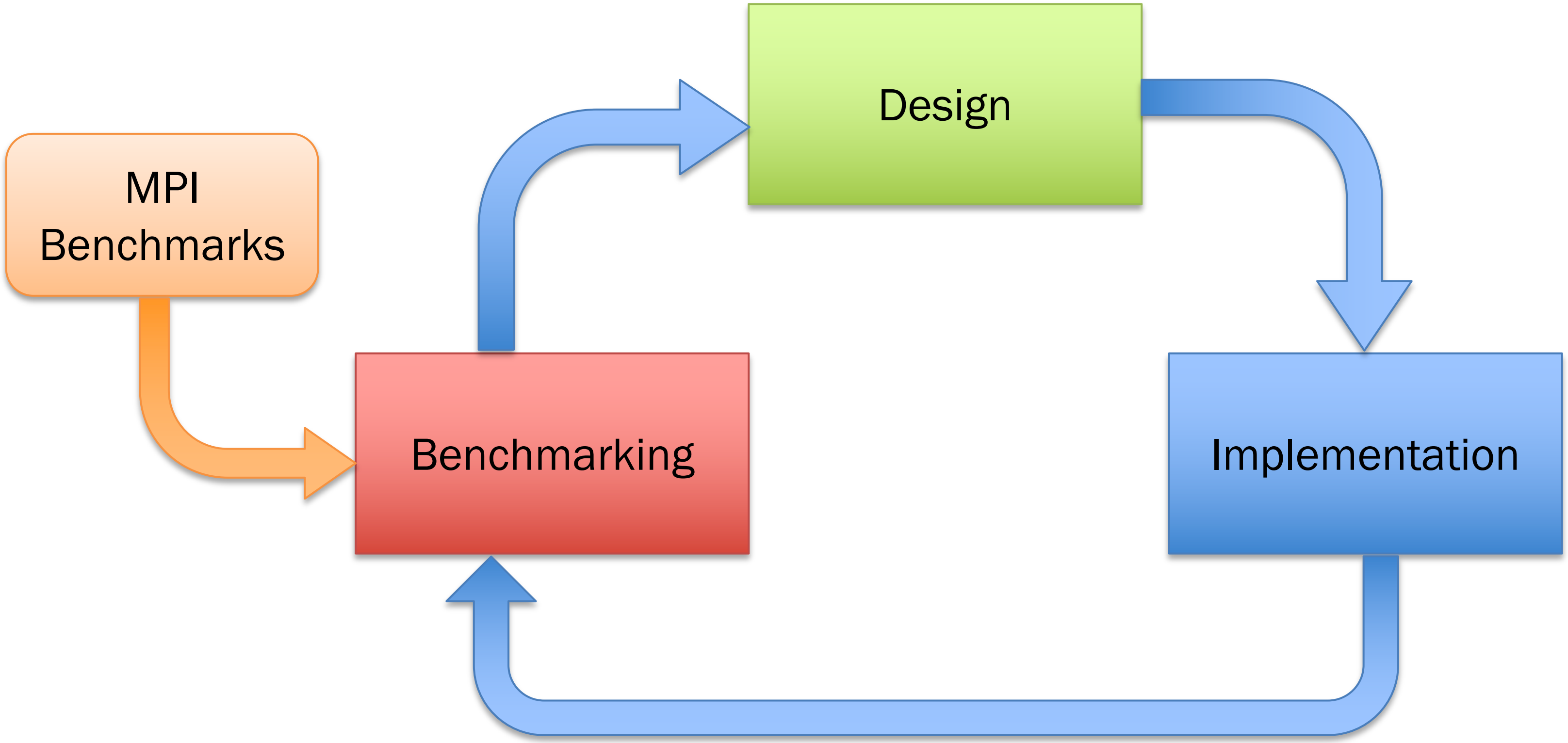
- Latency
- Latency hiding
- Bandwidth
- Concurrency
- Resource usage
- Hardware features
- Correctness

## What we *don't* care about:

- Impact of barrier algorithms
  (unless we're benchmarking barrier latency)



# Why should we care?




MVAPICH

# Why don't we care?

- We know this is bad, but all common benchmarks do it anyway
- Proper implementation is not trivial

LLNL mpiBench

```
__TIME_START__;
for (i = 0; i < p->iter; i++) {
    MPI_Reduce(sbuffer, rbuffer, p->count,
               p->type, p->reduceop, p->root,
               p->comm);
    __BAR__(p->comm);
}
__TIME_END__;
```

Intel MPI Benchmarks

```
for (i = 0; i < ITERATIONS->n_sample; i++) {
    t1 = MPI_Wtime();
    MPI_ERRHAND(MPI_Reduce((char*)c_info->s_buffer + i % ITERATIONS->s_cache_iter * ITERATIONS->s_offs,
                           (char*)c_info->r_buffer + i % ITERATIONS->r_cache_iter * ITERATIONS->r_offs,
                           s_num,
                           c_info->red_data_type, c_info->op_type,
                           root,
                           c_info->communicator));
    t2 = MPI_Wtime();
    *time += (t2 - t1);

    /*  CHANGE THE ROOT NODE */
    root = (root + c_info->root_shift) % c_info->num_procs;
    IMB_do_n_barriers(c_info->communicator, c_info->sync);
}
```

OSU Micro-Benchmarks

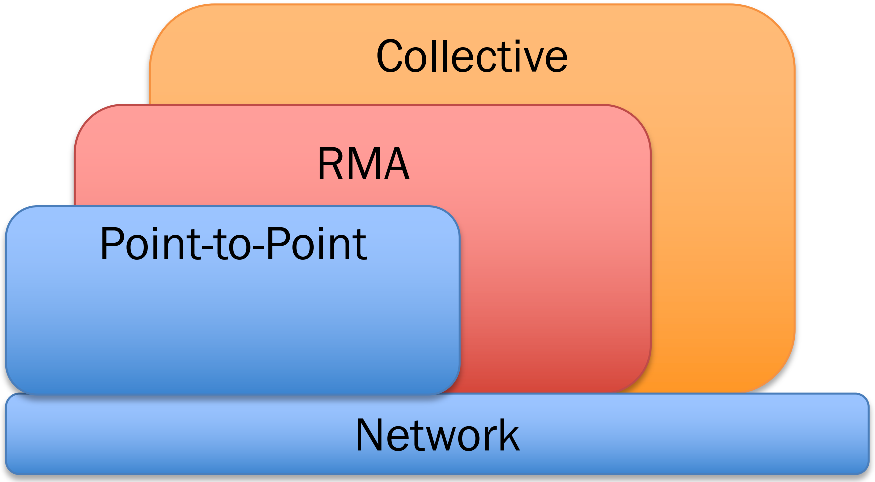
```
MPI_CHECK(MPI_Barrier(MPI_COMM_WORLD));

t_start = MPI_Wtime();

MPI_CHECK(MPI_Reduce(sendbuf, recvbuf, num_elements,
                     omb_curr_datatype, MPI_SUM, 0,
                     MPI_COMM_WORLD));
t_stop = MPI_Wtime();
```

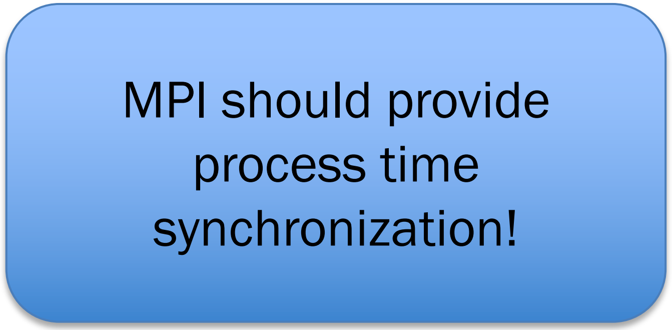
# How can we care?

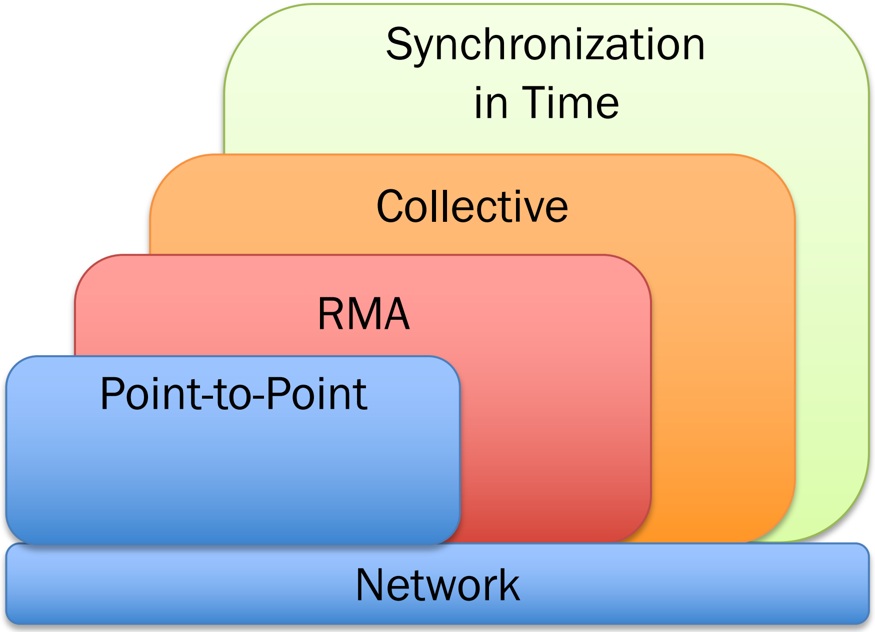
- MPI provides abstractions on many levels
- Abstract away the hard stuff
  - No user should have to implement reductions
  - Or broadcast
  - Or alltoall
  - Or message matching



# How can we care?

- MPI provides abstractions on many levels
- Abstract away the hard stuff
  - No user should have to implement reductions
  - Or broadcast
  - Or alltoall
  - Or message matching

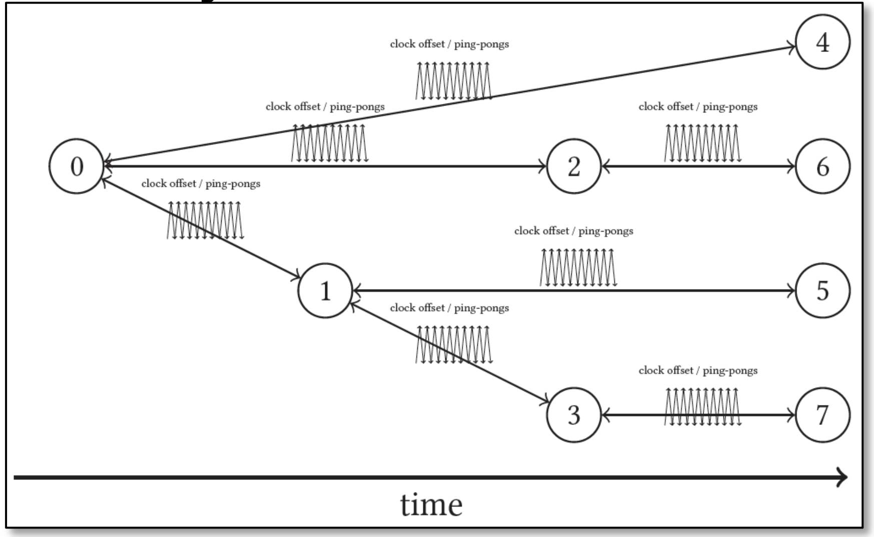


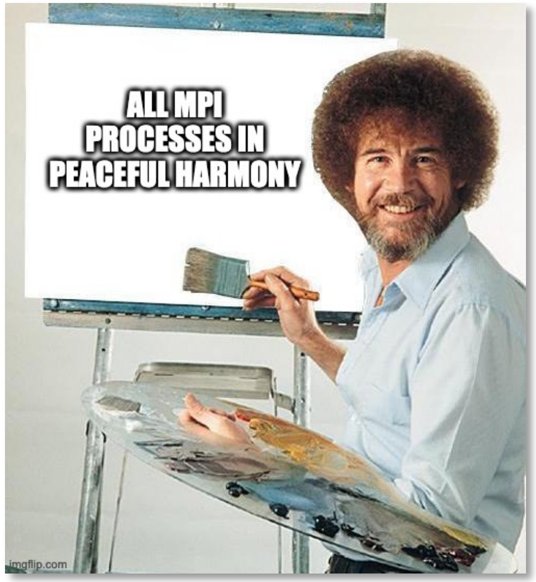



# Process Synchronization in Time

- Few machines provide global clocks (MPI_WTIME_IS_GLOBAL)
  - Only have the work
- All others: Synchronization in 2 steps:

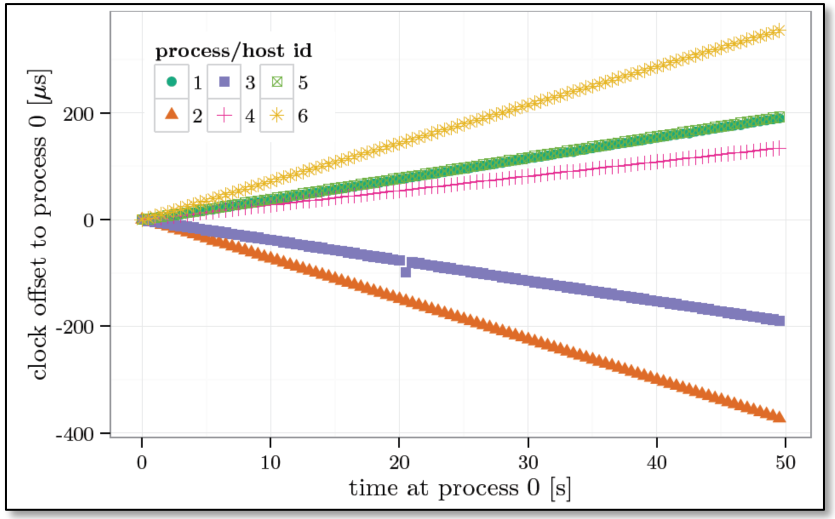
1. Ensure synchronized virtual clocks
2. Ensure synchronized execution



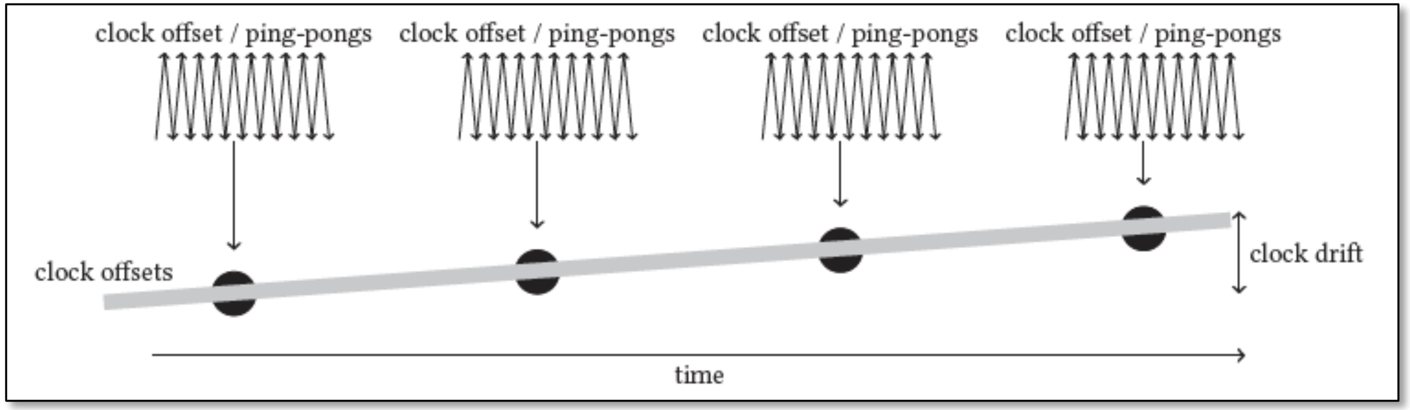



# Challenge: Clock Drift

- Clocks run at different speeds (1-10ppm)
- Impacted by temperature and manufacturing differences
- Drift correction:
  - Include drift in local clock synchronization
  - Periodically re-synchronize




S. Hunold and A. Carpen-Amarie. 2015. On the Impact of Synchronizing Clocks and Processes on Benchmarking MPI Collectives. EuroMPI '15. https://doi.org/10.1145/2802658.2802662

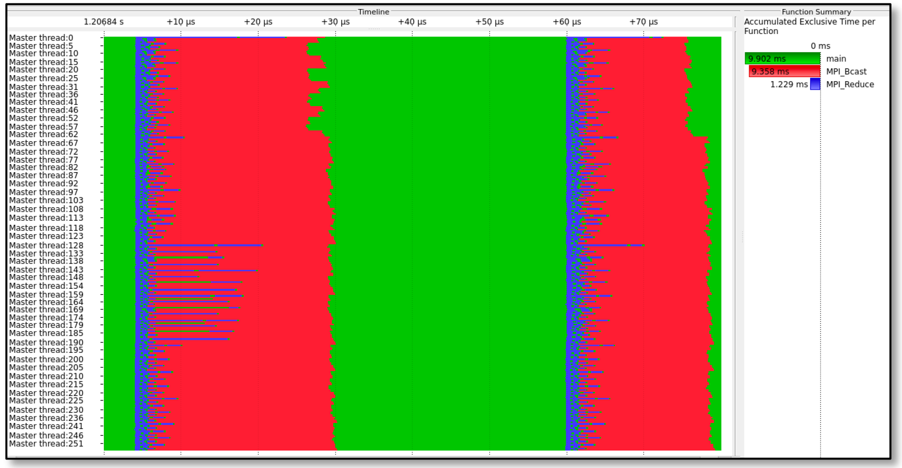


# Introducing: MPIX_Harmonize

- Synchronizes internal clocks in regular intervals
  - To correct for clock drift
  - Every few seconds (automatically adjustable?)
- Attempts to block processes until a common point in time
  - **flag == true** if calling process reached deadline
  - **flag == false** if calling process missed deadline
  - Application deals with synchronization misses
- Processes resume execution in harmony

The proposed function MPIX_Harmonize has the following signature:

```
int MPIX_Harmonize(MPI_Comm comm, int *flag);
```



# MPIX_Harmonize Algorithm

- Periodically resynchronize clocks
  - Or if a process previously missed a deadline
- Process 0 broadcasts new deadline
- Processes block until deadline, return success or failure

**Algorithm 1** Algorithm for `MPIX_Harmonize`.

```
Require: comm                                          ▷ Input communicator
Require: outflag                                       ▷ Output flag
slack ← sync_slack(comm)
flag ← 0                                               ▷ Local and global state
root ← 0
if last_sync_failed(comm) then
    flag ← SYNC_FAILED                                 ▷ Locally failed
end if
if elapsed_since_last_sync(comm) > 1.0s then
    flag ← flag | SYNC_EXPIRED                         ▷ Locally expired
end if
flag ← MPI_Reduce(flag, MPI_BOR, root, comm)
if root = comm_rank(comm) then
    if flag then                                       ▷ Global State
        sync_time ← −1.0                               ▷ Trigger clock sync
        if flag | SYNC_FAILED then
            slack ← slack × 1.5                        ▷ Increase slack
            sync_slack(comm) ← slack
        end if
    else
        sync_time ← global_time() + slack
    end if
end if
sync_time ← MPI_Bcast(sync_time, root, comm)
if sync_time < 0.0 then                                ▷ Negative time triggers sync
    sync_clocks();
    MPI_Reduce(0, root, comm)                          ▷ Reduce clock-sync jitter
    last_sync_time(comm) ← global_time()
    sync_time ← global_time() + slack                  ▷ New sync time
    sync_time ← MPI_Bcast(sync_time, root, comm)
end if
if sync_time < global_time() then
    outflag ← 0                                        ▷ Missed deadline
    last_sync_failed(comm) ← TRUE                      ▷ Store for next call
else
    outflag ← 1                                        ▷ Success, wait for sync time
    while sync_time > global_time() do
    end while
end if
```

# Usage in Collective Benchmarks

1. Harmonize processes
2. Perform operation under test
3. Discard invalid measurements

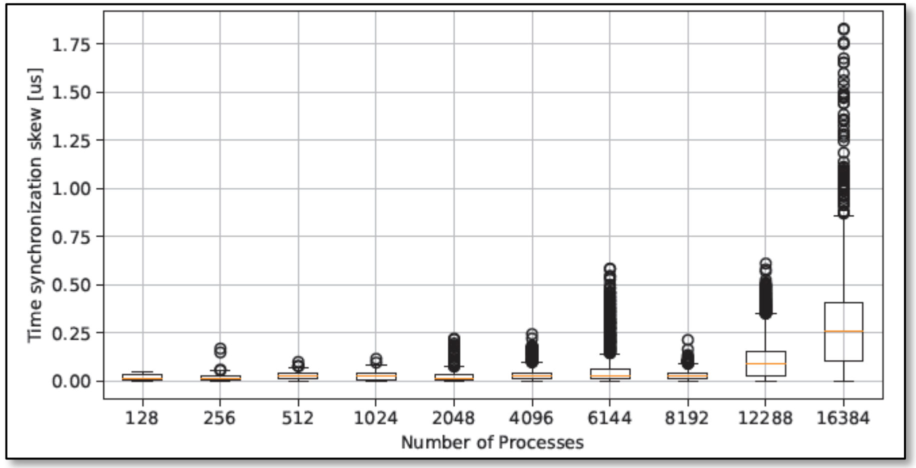
```
for (int i = 0; i < NITER; ++j) {
  int flag, check_idx;
  /* synchronize in space and time */
  MPIX_Harmonize(comm, &flag);
  /* take measurement */
  double begin = MPI_Wtime();
  MPI_Reduce(..., comm);
  check_idx = i%CHECK_EVERY;
  check_time[check_idx] = MPI_Wtime() - begin;
  check_flags[check_idx] = flag;
  if (i == NITER-1 ||
      check_idx == CHECK_EVERY-1)) {
    /* discard invalid timings */
    MPI_Allreduce(MPI_IN_PLACE, check_flags,
                  check_idx,
                  MPI_INT, MPI_LAND, comm);
    for (int k = 0; k < check_idx; ++k) {
      if (check_flags[k]) { /* valid */
        wtime += check_time[k];
        valid_iterations++;
      }
    }
  }
}
```

# Evaluation

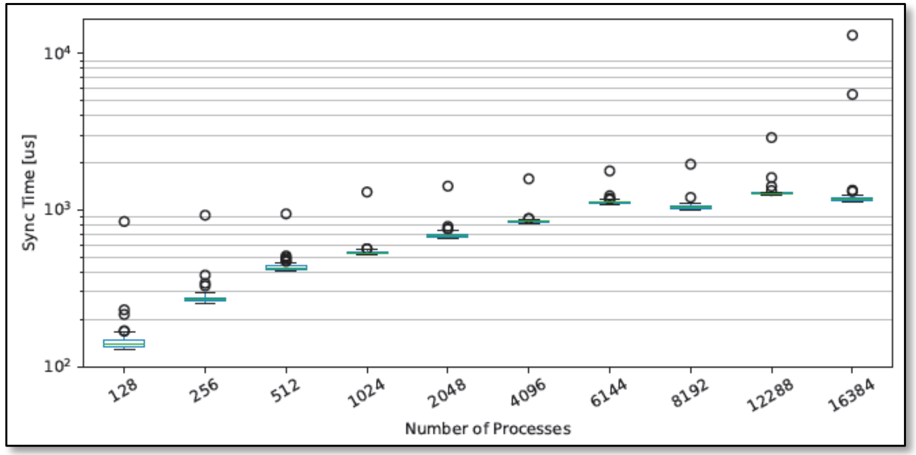
- All experiments performed on *Hawk* installed at HLRS, Stuttgart, Germany
- 2x64 core AMD EPYC Rome
- 200Gbit/s ConnectX-6
- Up to 64 nodes (64k processes)

# Cost of Process Synchronization

- Synchronization Skew:
  - Majority: < 0.5us
  - Outliers: < 2us
  - ~ 1/100 of barrier

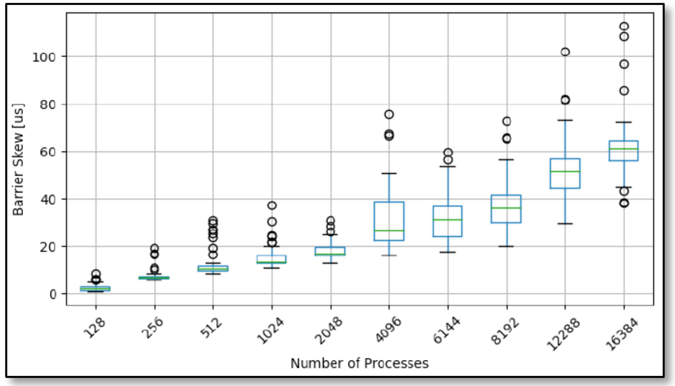


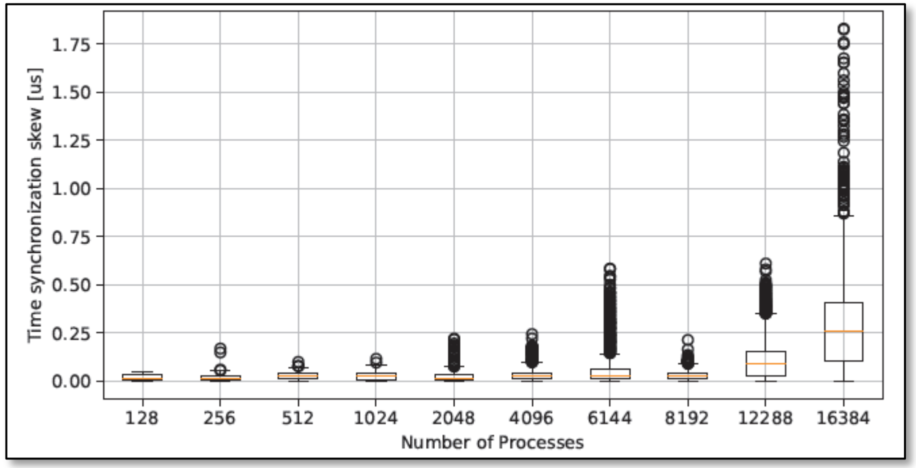

- Clock Synchronization <1ms
  - First synchronization more costly due to connection setup
  - 0.1 – 1% overhead if done every second



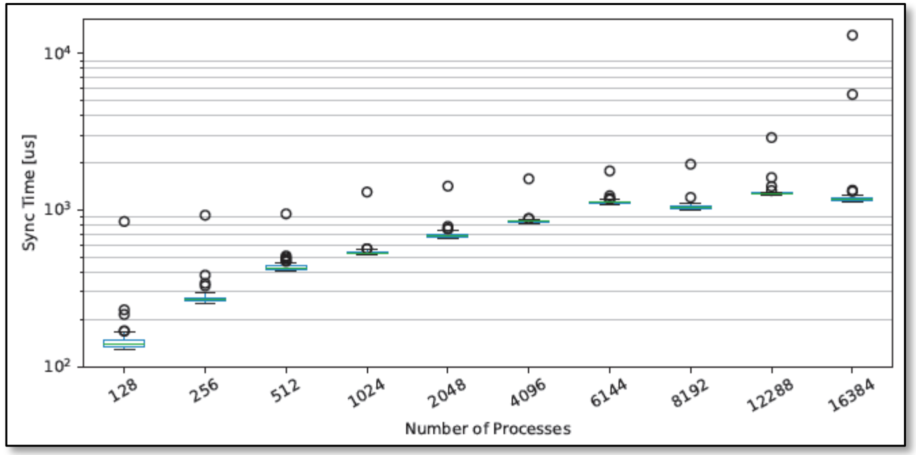
# Cost of Process Synchronization

- Synchronization Skew:
  - Majority: < 0.5us
  - Outliers: < 2us
  - ~ 1/100 of barrier



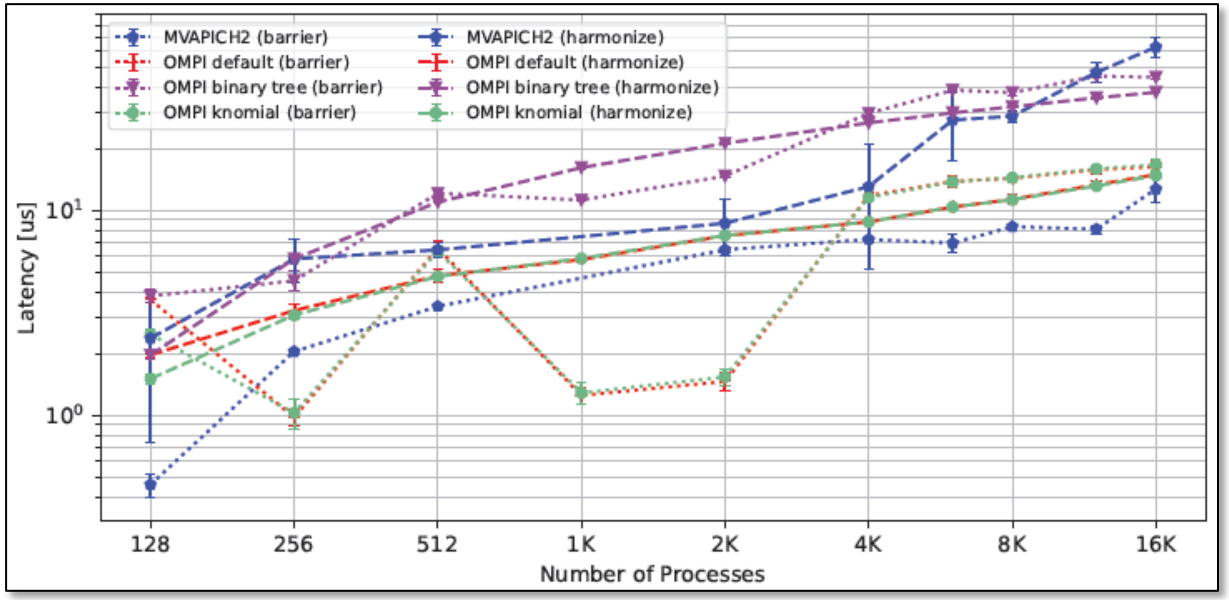




- Clock Synchronization <1ms
  - First synchronization more costly due to connection setup
  - 0.1 – 1% overhead if done every second



# Impact on MPI_Bcast

- 4B messages, process scaling
- Open MPI: knomial impacted by barrier algorithm
- MVAPICH: lower reported latency with barrier-synchronization

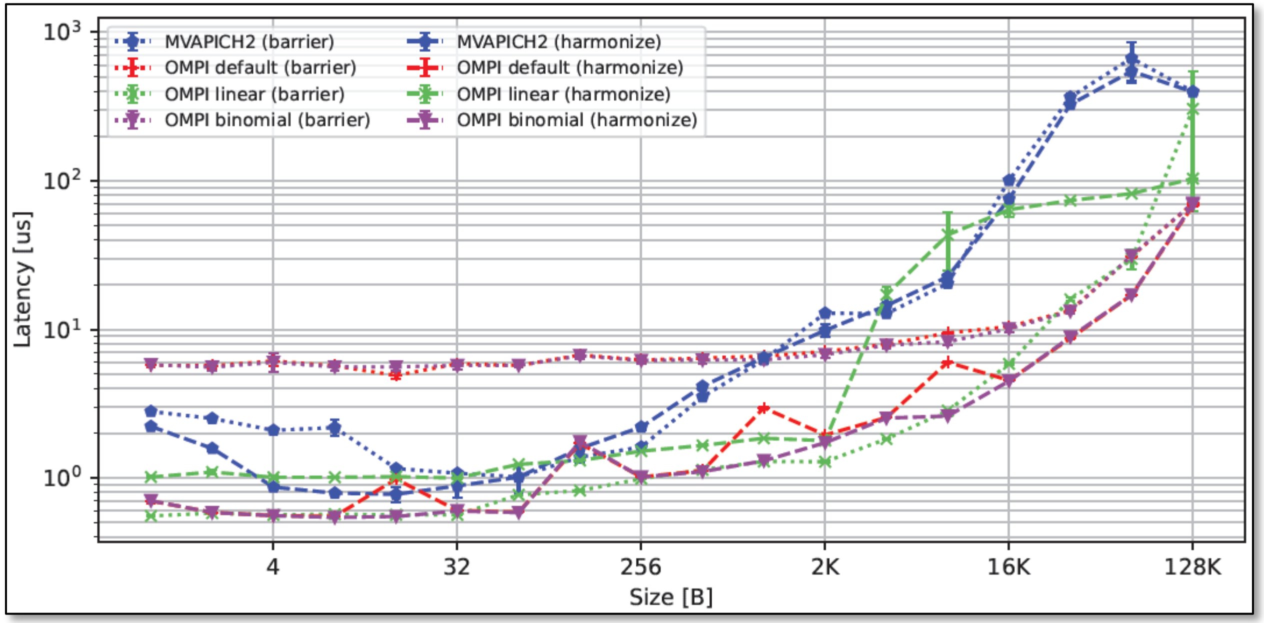



Synchronization does not impact algorithm performance, but latency reported by OSU benchmarks.

# Impact on MPI_Reduce Benchmarks

- Some algorithms in Open MPI show large spread between barrier-synchronization and MPIX_Harmonize (binomial)
- Open MPI default heuristic needs to be re-evaluated

MPI_Reduce on 16k processes

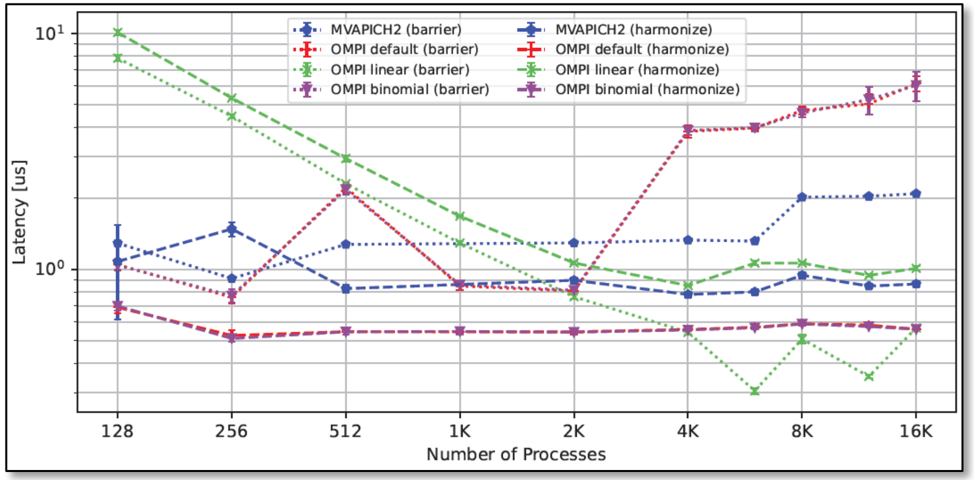



Synchronization does not impact algorithm performance, but latency reported by OSU benchmarks.

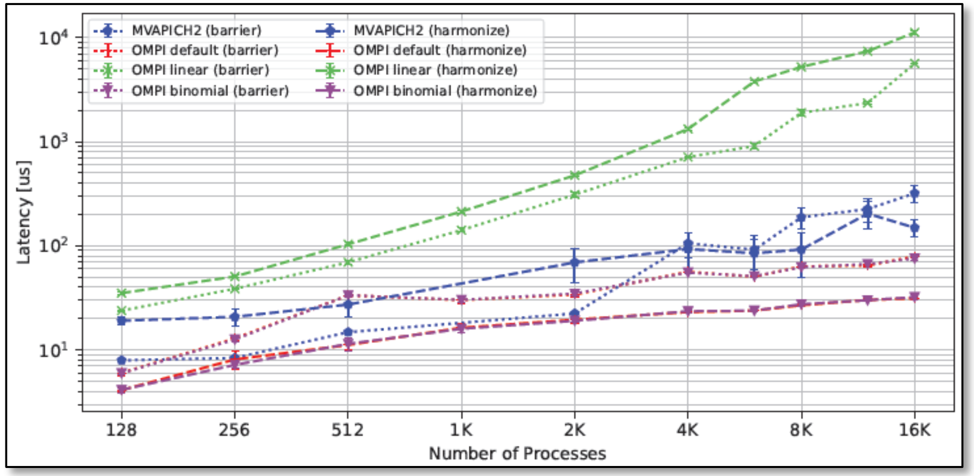
# Detour: What do we actually measure?

- The **mean of means** is not a good metric
  - Mean over all processes
  - Is that really representative?
- Tightly coupled applications sensitive to imbalances
- Better: **max of means**?

$$t_{avg} = \frac{1}{p}\sum_{p}\left(\frac{1}{N}\sum_{i=0}^{N} t_i\right)$$




Process scaling, 4B messages, MEAN

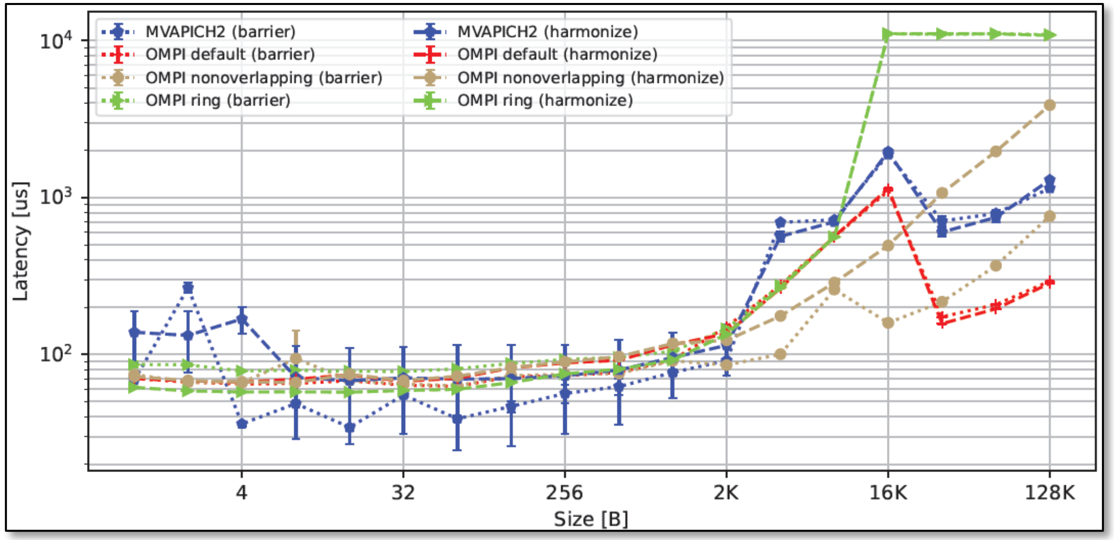



Process scaling, 4B messages, MAX

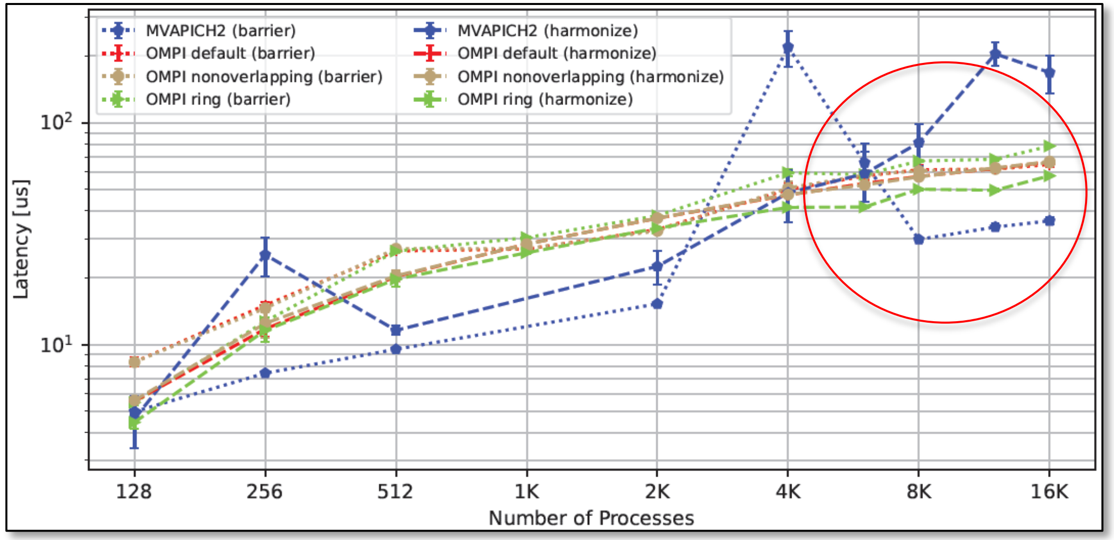
# Impact on MPI_Allreduce

- Impact less pronounced on non-rooted collectives
- Open MPI & MVAPICH2 heuristics should be re-evaluated

Message scaling, 16k processes




4B messages, process scaling



# Summary

- Proper time synchronization is hard but important
- Barriers introduce arbitrary arrival patterns
- MPI should provide synchronization infrastructure

**MPIX_Harmonize**

Process Synchronization in Space **and Time**

with local synchronization check

# Acknowledgements


- HLRS: access to Hawk
- Exascale Computing Project (17-SC-20-SC)
- Austrian Science Fund (FWF): project P 33884-N