# A simple, pipelined all-gather algorithm for large irregular problems

Jesper Larsson Träff, Andreas Ripke,
Christian Siebert,
Pavan Balaji, Rajeev Thakur, William Gropp
NEC Labs Europe, ANL, UIUC

# An irregular all-gather data-exchange operation

- A set of processes, 0, …, p-1
- Each has a block of data Bi of (possibly) different size

| Bi |
|---|

All processes have all blocks B0, B1, …, B(p-1)

| B0 | B1 | … | Bi | … | | B(p-1) |
|---|---|---|---|---|---|---|

MPI_Allgatherv(sendbuf,…,recvbuf,…,counts,…);

counts[i] ≈ |Bi|, all processes know the size of all blocks!

MPI_Allgatherv used in numerical libraries eg. PETSc

- Other irregular collectives, eg. MPI_Alltoallw

Irregular collectives algorithmically (much!) more difficult – challenging - than regular collectives:

1. Different amounts of data between processes (in different rounds) → Load imbalance
2. Partial information for the processes (MPI_Alltoallw, MPI_Gatherv, …) → Difficult/expensive to compute schedule
3. Schedules → Optimality is (NP)-hard!

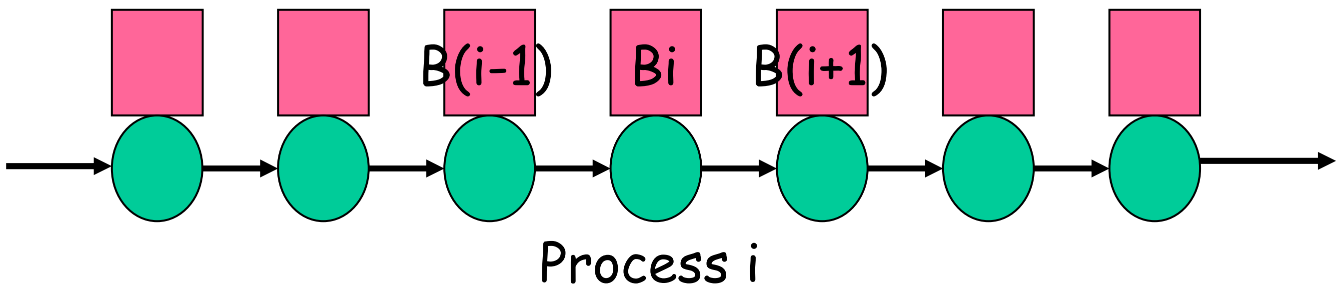
Related work: MPI_Allgather (gossip, broadcast-to-all, …)

- Bruck et al. '97: simultaneous binomial tree algorithm:
- Benson et al. '03: MPI implementation of allgather algorithms on switched networks [non SMP-aware]
- Thakur et al. '04: mpich2 implementations of Bruck and other [non SMP-aware]
- Mamidala et al. '06: SMP implementations
- Träff'06: Graceful degradation from Bruck to linear ring [SMP-aware]

MPI_Allgatherv

- Balaji et al '07: optimization in the context of PETSc

# An algorithm for large, regular all-gather problems




Linear ring: p-1 rounds

- Process i receives block B(i-1-k) and sends block B(i-k) in round k, k=0, …, p-1

## Analysis:

Size per process $mi = |Bi| = m'$ (for regular problem)

Total problem size $m = \sum mi$

- p-1 (regular) communication rounds
- Each round takes time O(mi), total O((p-1)mi) = O(m-m')

Assumptions:

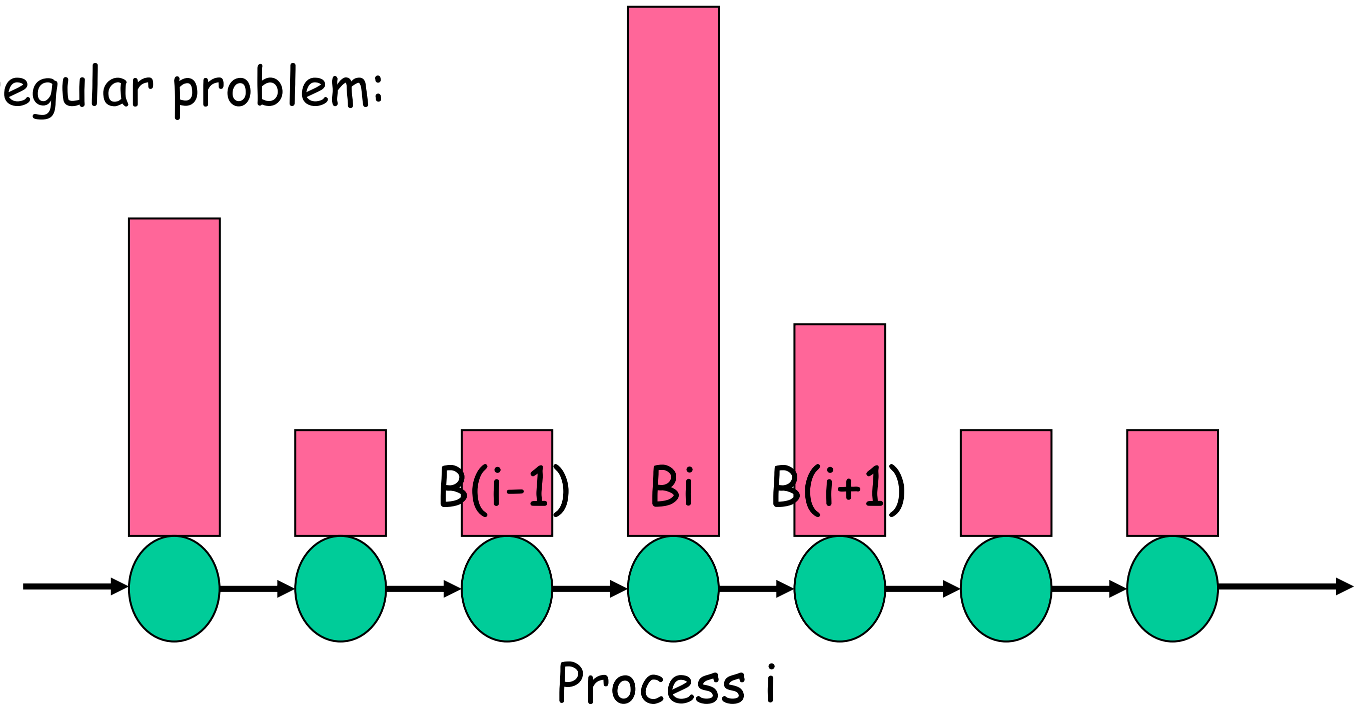
- Processes can send and receive simultaneously
- Cost of sending/receiving data of size m' is O(m')
- Homogeneous communication along ring

NOTE:

For small m' algorithm with log p rounds preferable!

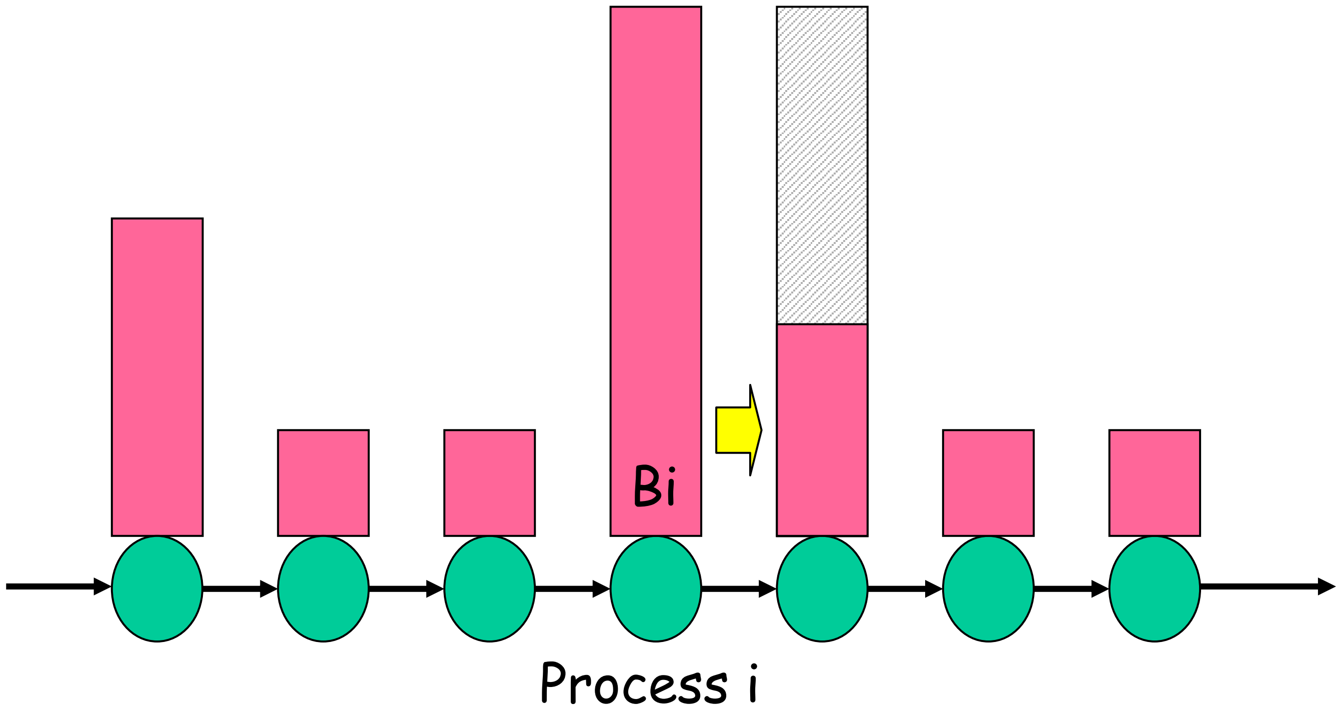
Optimal: no idle time, no superfluous data!

Irregular problem:



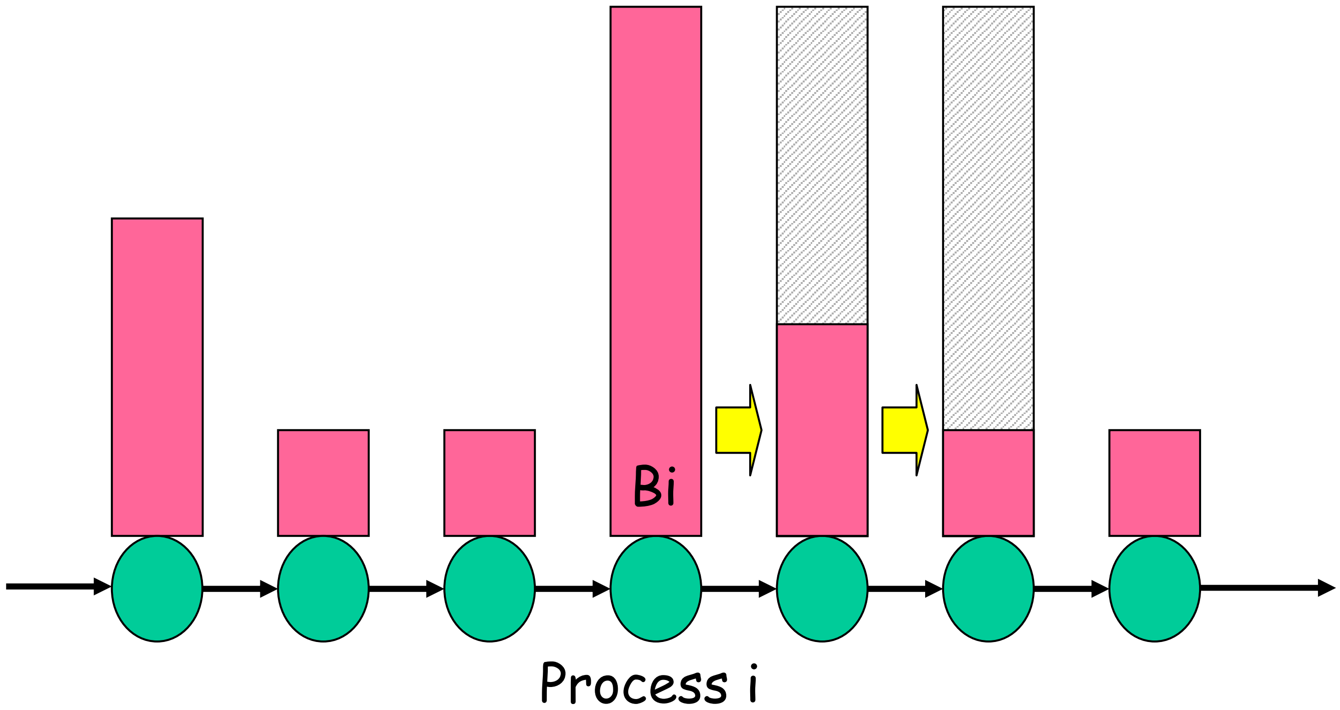

Linear ring: p-1 rounds

- Process i receives block B(i-1-k) from i-1 and sends block B(i-k) to i+1 in round k, k=0, …, p-1



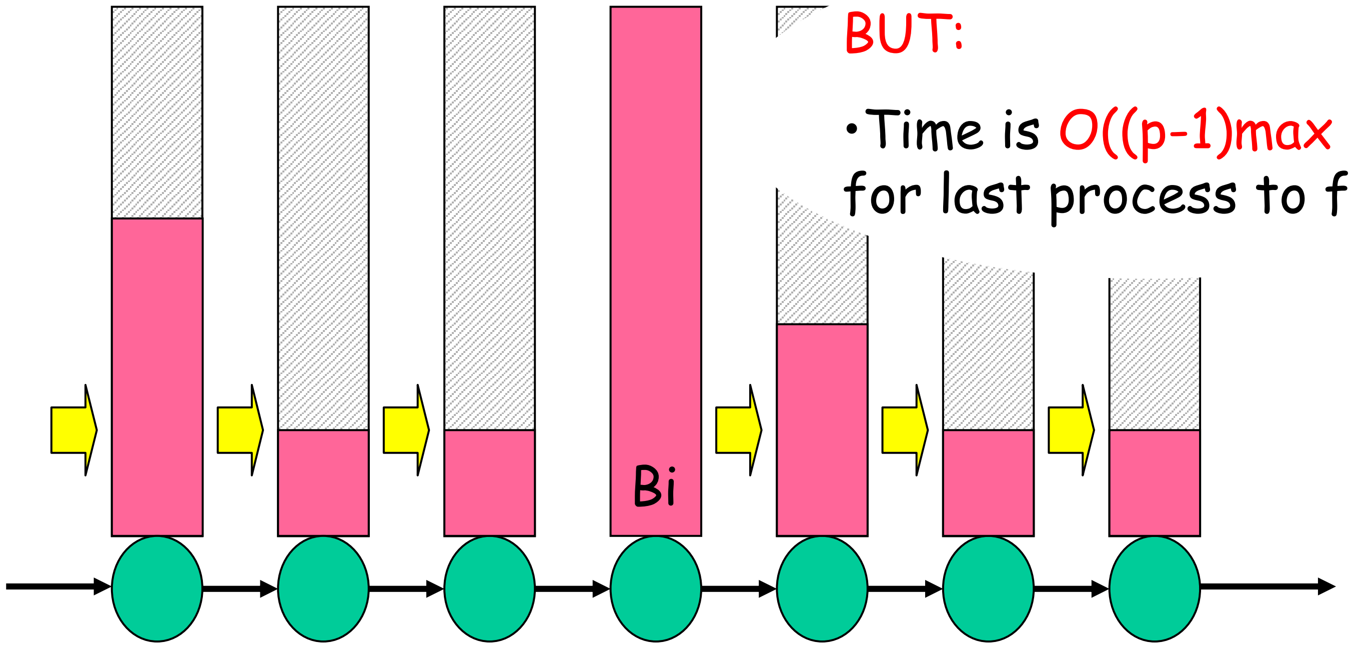

Linear ring: p-1 rounds

•Process i receives block B(i-1-k) from i-1 and sends block B(i-k) to i+1 in round k, k=0, …, p-1




Linear ring: p-1 rounds

•Process i receives block B(i-1-k) from i-1 and sends block B(i-k) to i+1 in round k, k=0, …, p-1




BUT:

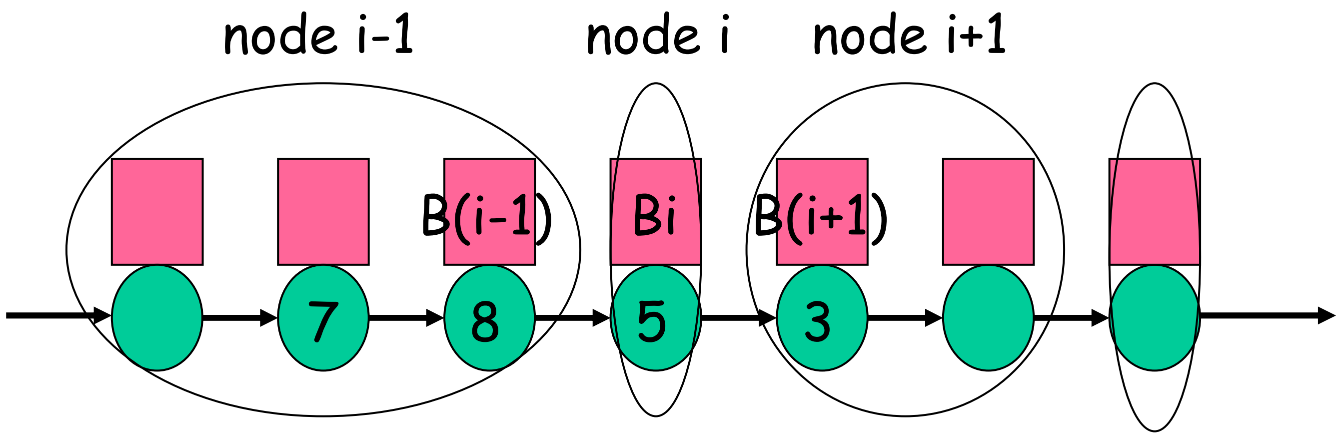
- Time is O((p-1)max |Bi|) for last process to finish

Process i

Linear ring: p-1 rounds

- Process i receives block B(i-1-k) from i-1 and sends block B(i-k) to i+1 in round k, k=0, …, p-1

Observation 1: linear ring works for clustered systems also



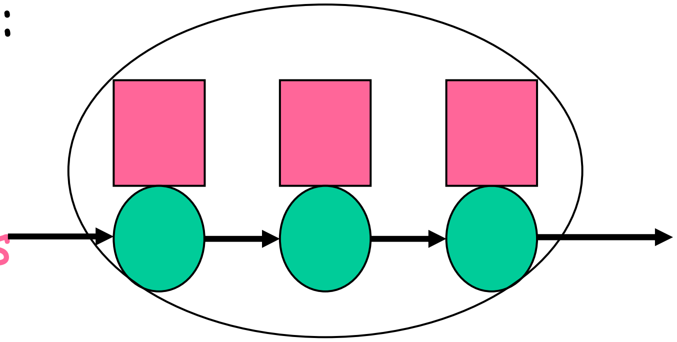

Linear ring: p-1 rounds

•Process i receives block B(i-1-k) from i-1 and sends block B(i-k) to i+1 in round k, k=0, …, p-1

MODIFICATION: Use virtual ranking, one process per node sends, one process per node receives

## Analysis:

- p-1 rounds
- Inter-node connections busy in all rounds: one process per node sends, one process per node receives
- Each node sends and receives (p-1) blocks

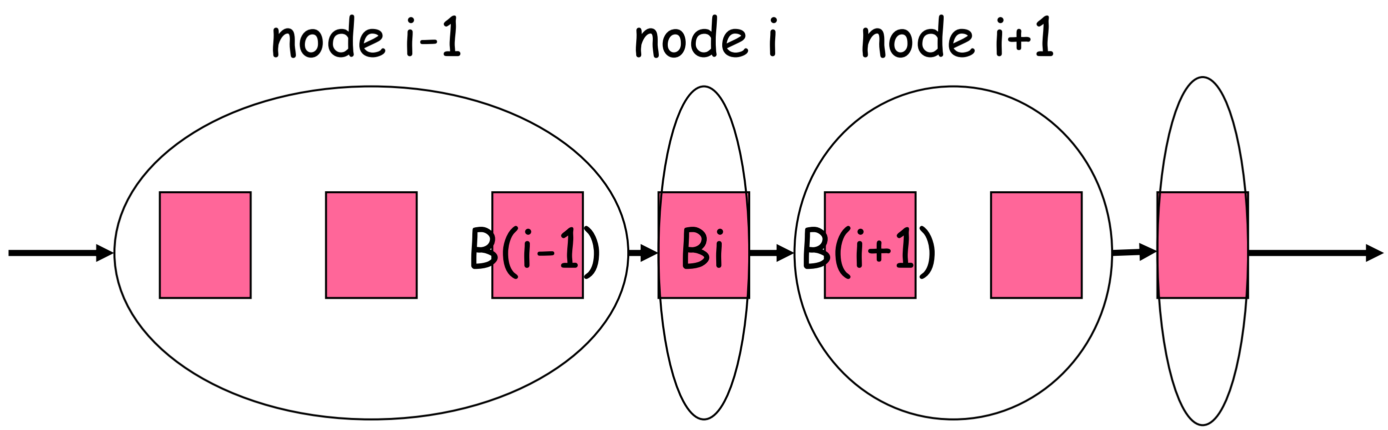


IMPROVEMENT:

a node never receives a block that it already has (replace by intra-node all-gather).

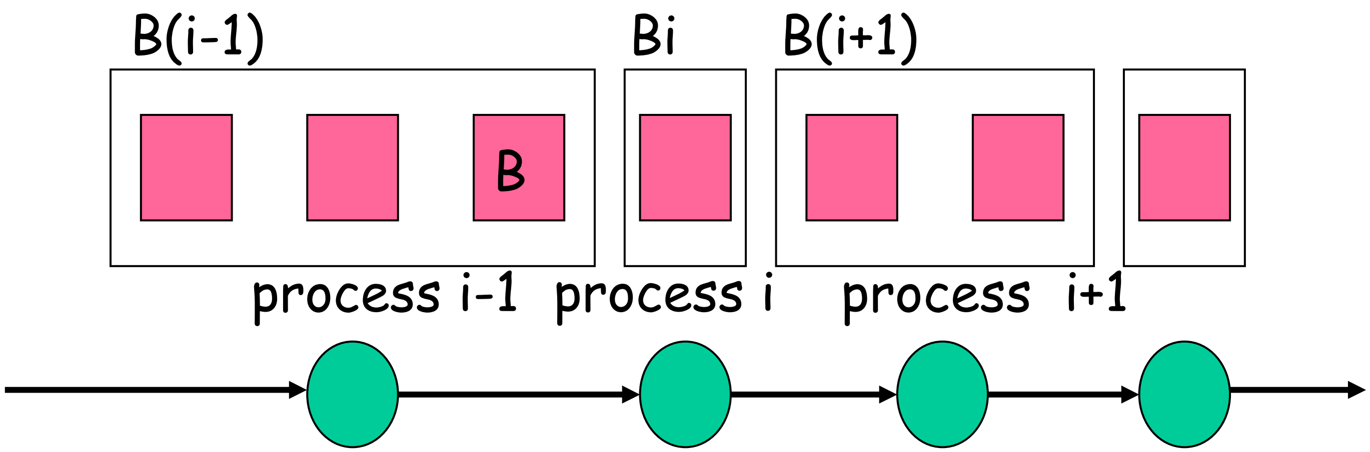
## Observation 2:

linear ring on cluster solves irregular problem over nodes




Irregular problem can be solved by simulating clustered algorithm

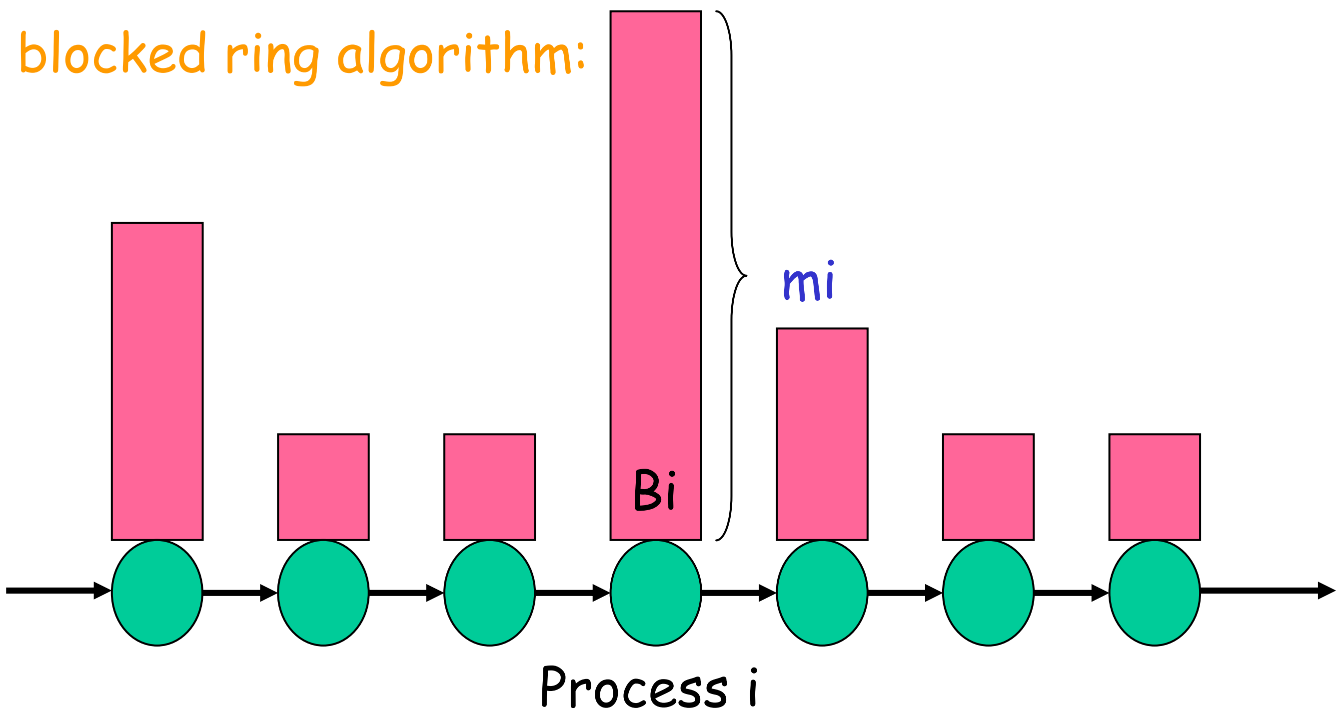
Handle each block Bi as node of ceil(Bi/B) regular blocks of some size B



# The algorithm for large, irregular all-gather problems

The blocked/pipelined ring algorithm

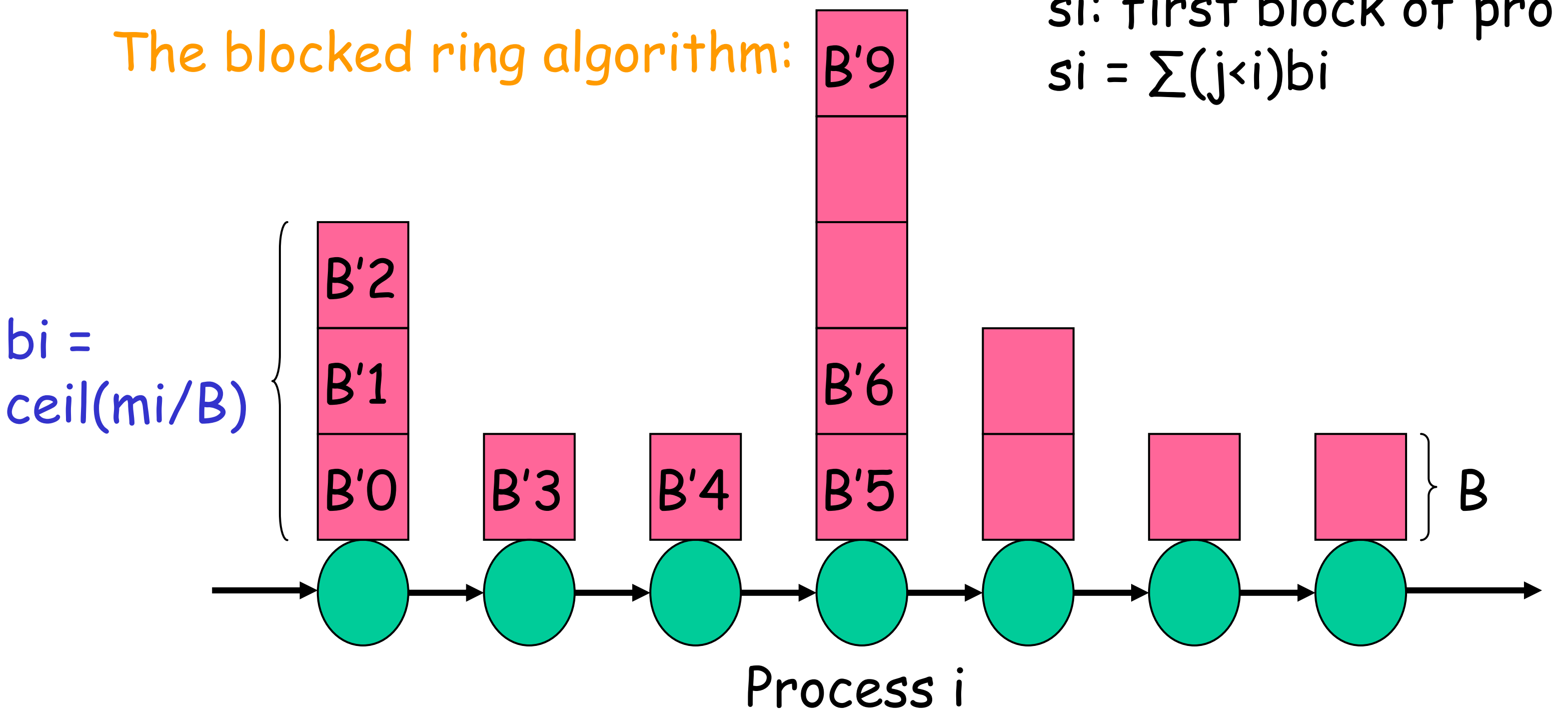
## The blocked ring algorithm:




1. For each process i, cut data Bi into bi = max(1,ceil(mi/B)) blocks B'j of some chosen size (at most) B

## The blocked ring algorithm:

si: first block of process i,
$si = \sum(j<i)bi$

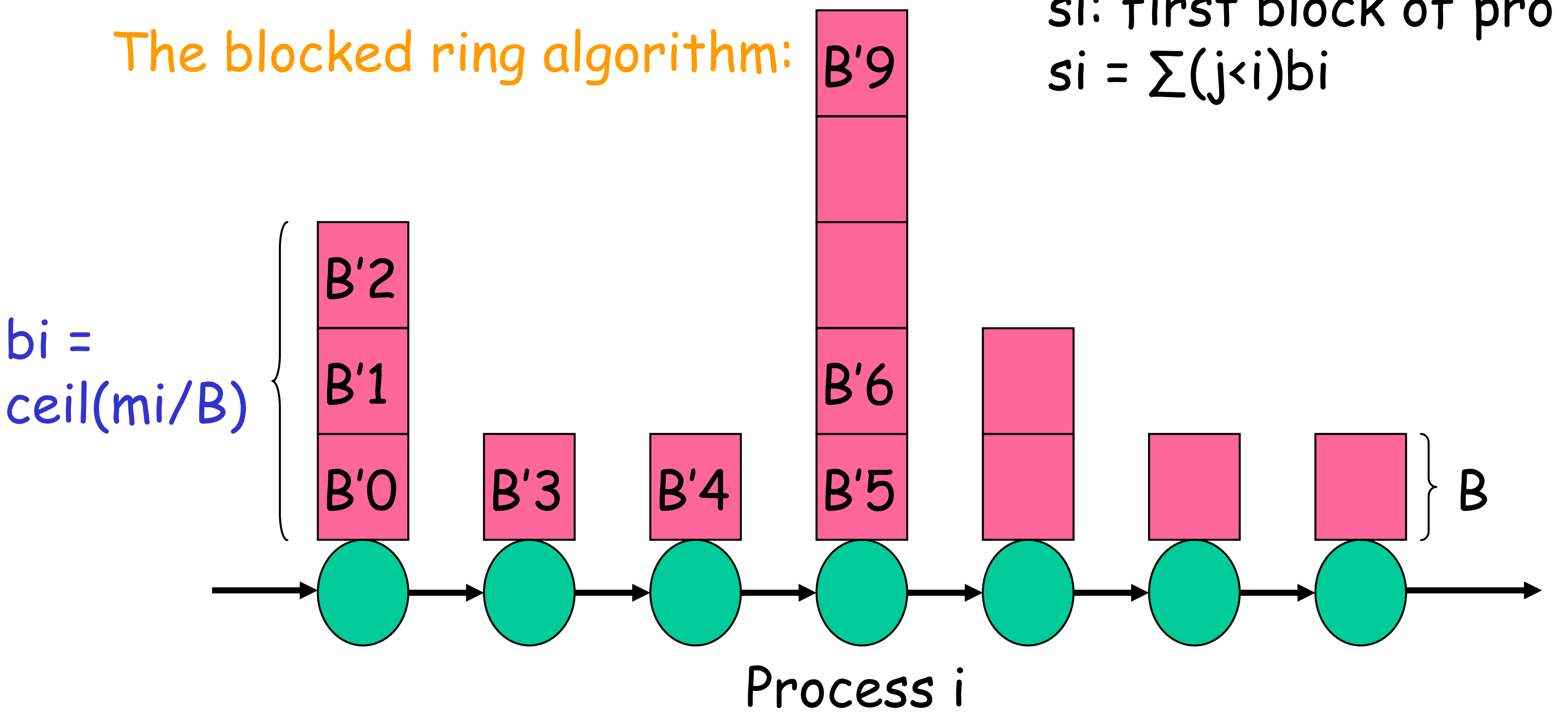



1. For each process i, cut data Bi into $bi = \max(1,\mathrm{ceil}(mi/B))$ blocks B'j of some chosen size (at most) B – each process has at least one block

Total number of blocks $b = \sum bi$

## The blocked ring algorithm:

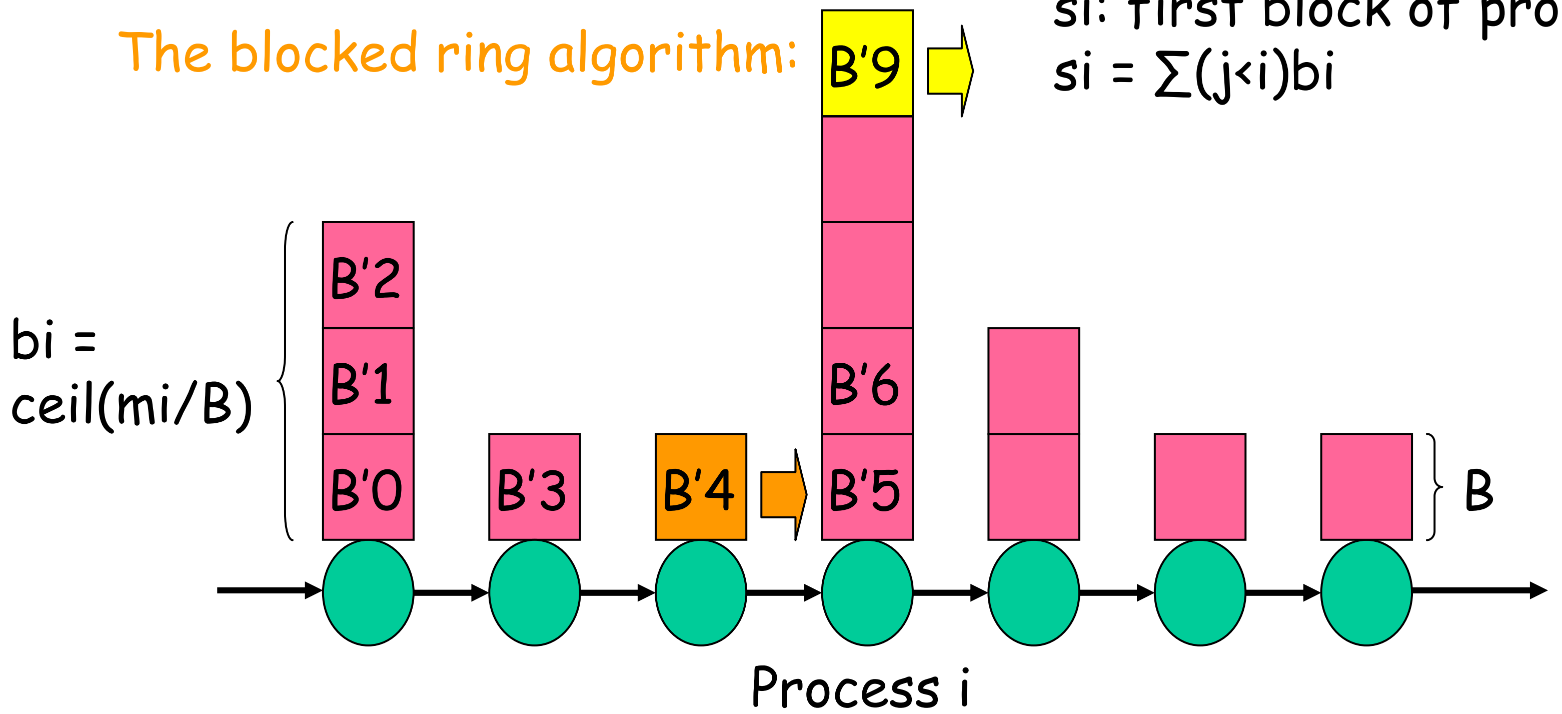
si: first block of process i,
si = ∑(j<i)bi




2. Run linear pipe on blocks, in round k process i receives block B'(si-1-k) from i-1 and sends block B'(si+bi-1-k) to i+1

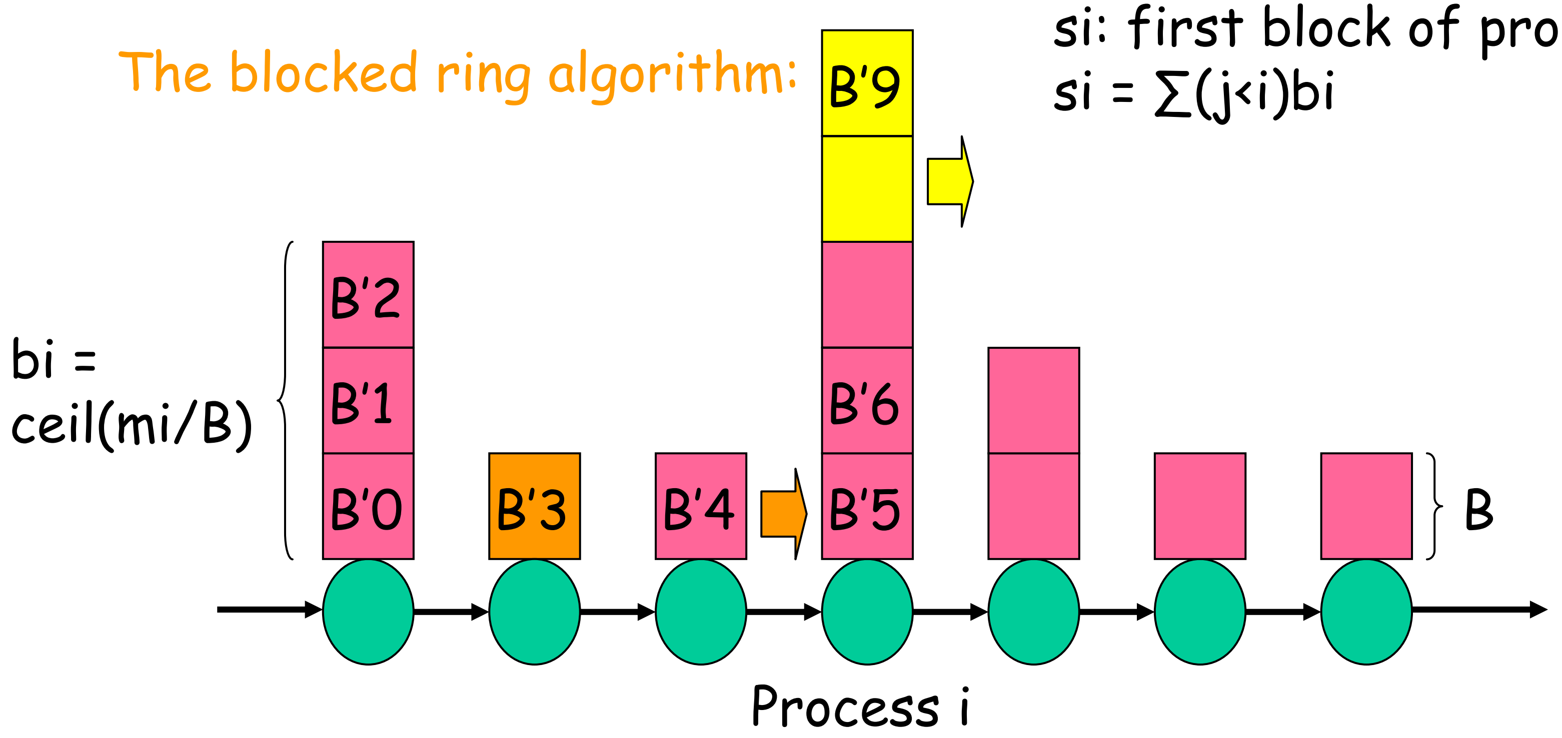
The blocked ring algorithm:

si: first block of process i,
si = ∑(j<i)bi




2. Run linear pipe on blocks, in round k process i receives block B'(si-1-k) from i-1 and sends block B'(si+bi-1-k) to i+1

The blocked ring algorithm:

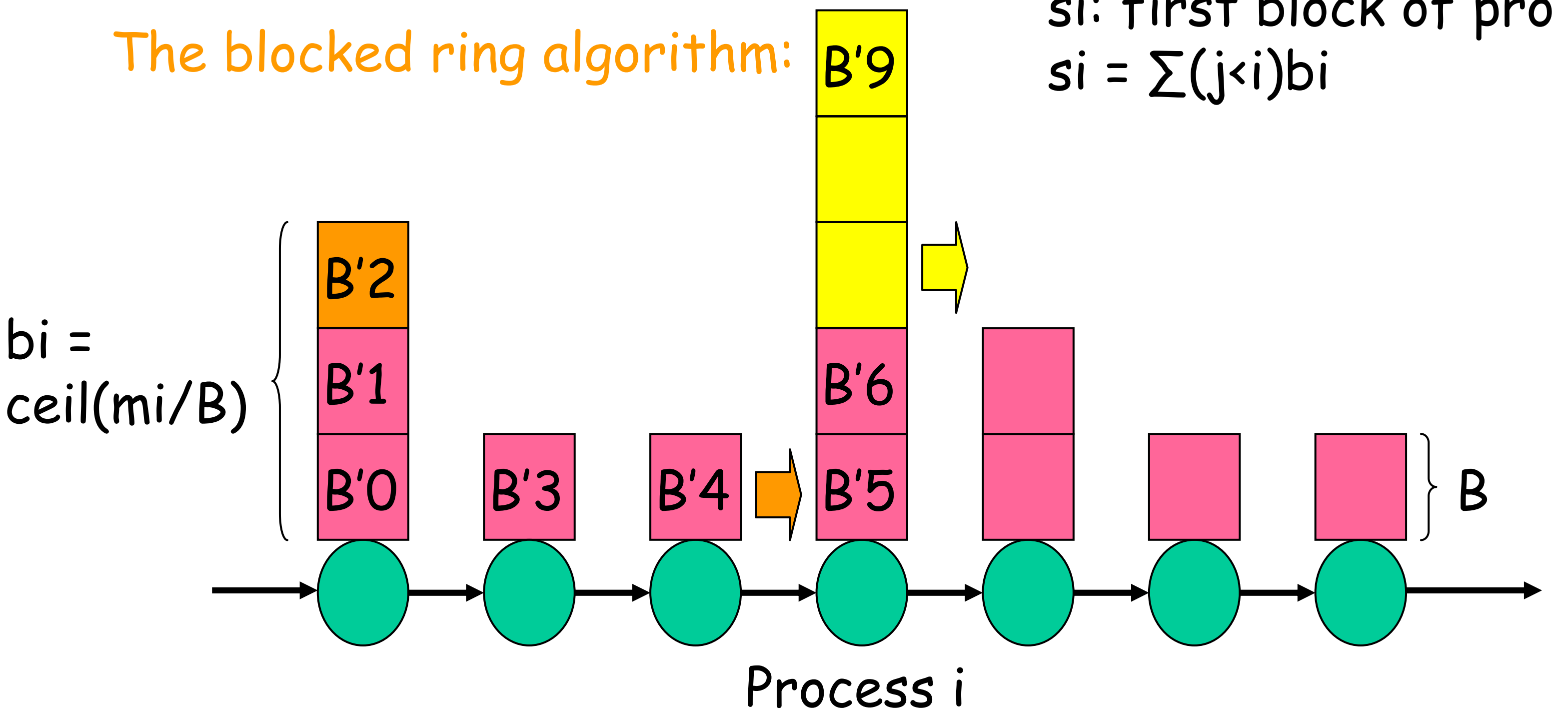



si: first block of process i,
si = ∑(j<i)bi

2. Run linear pipe on blocks, in round k process i receives block B'(si-1-k) from i-1 and sends block B'(si+bi-1-k) to i+1

The blocked ring algorithm:

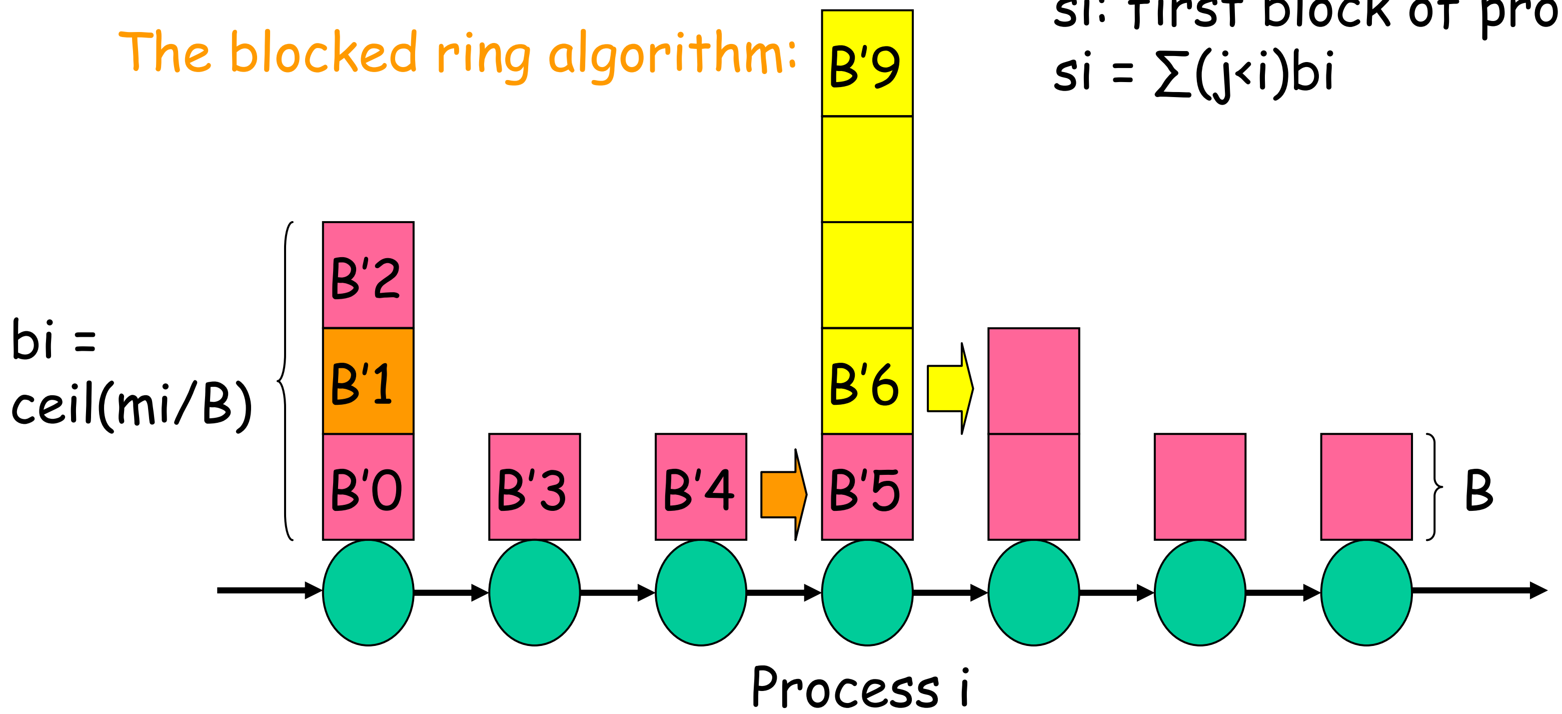
si: first block of process i,
si = ∑(j<i)bi




2. Run linear pipe on blocks, in round k process i receives block B'(si-1-k) from i-1 and sends block B'(si+bi-1-k) to i+1

The blocked ring algorithm:

si: first block of process i,
si = ∑(j<i)bi



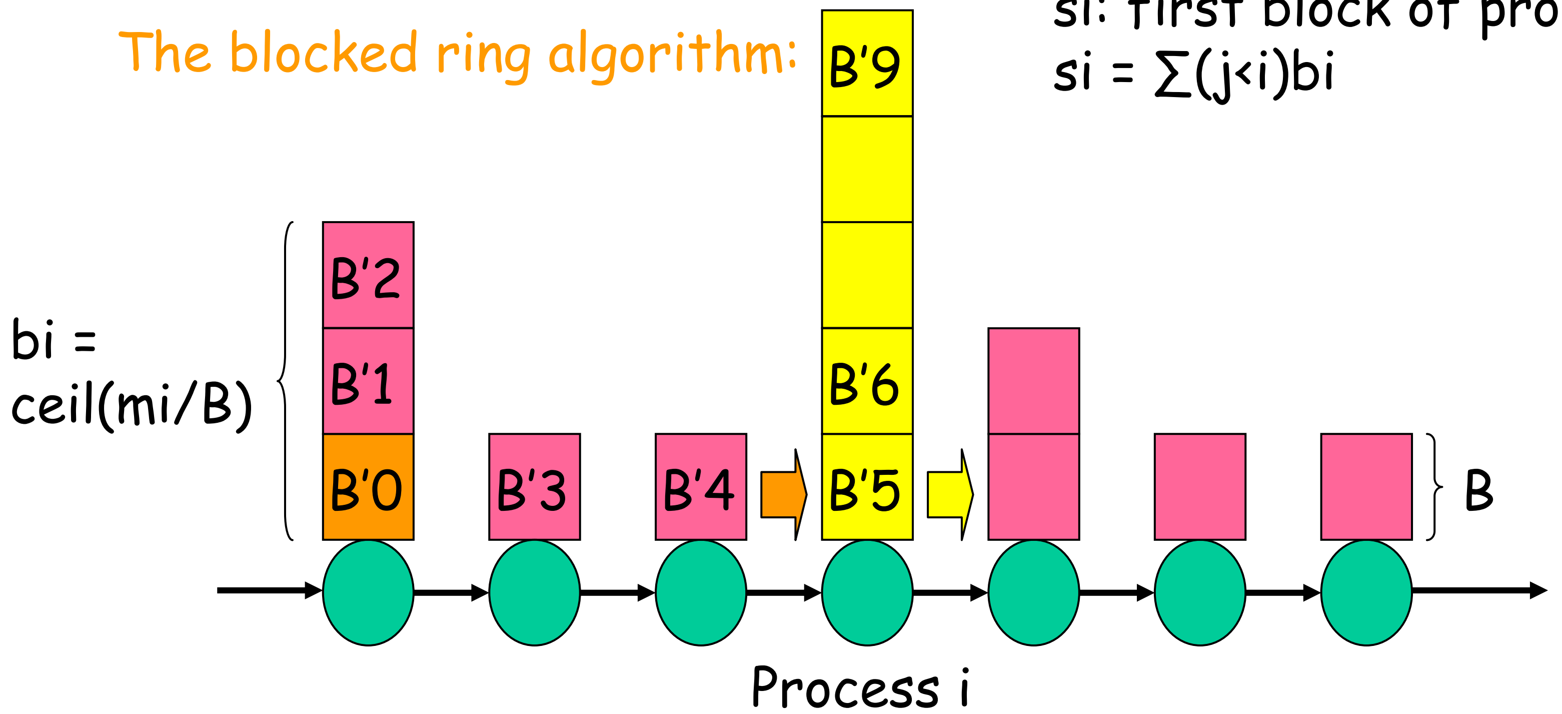

2. Run linear pipe on blocks, in round k process i receives block B'(si-1-k) from i-1 and sends block B'(si+bi-1-k) to i+1




NEC

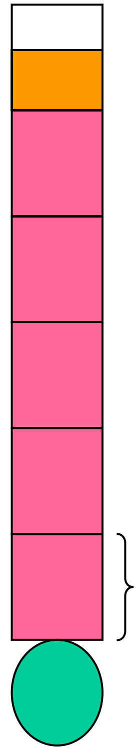
The blocked ring algorithm:

si: first block of process i,
si = ∑(j<i)bi




2. Run linear pipe on blocks, in round k process i receives block B'(si-1-k) from i-1 and sends block B'(si+bi-1-k) to i+1

The blocked ring algorithm:



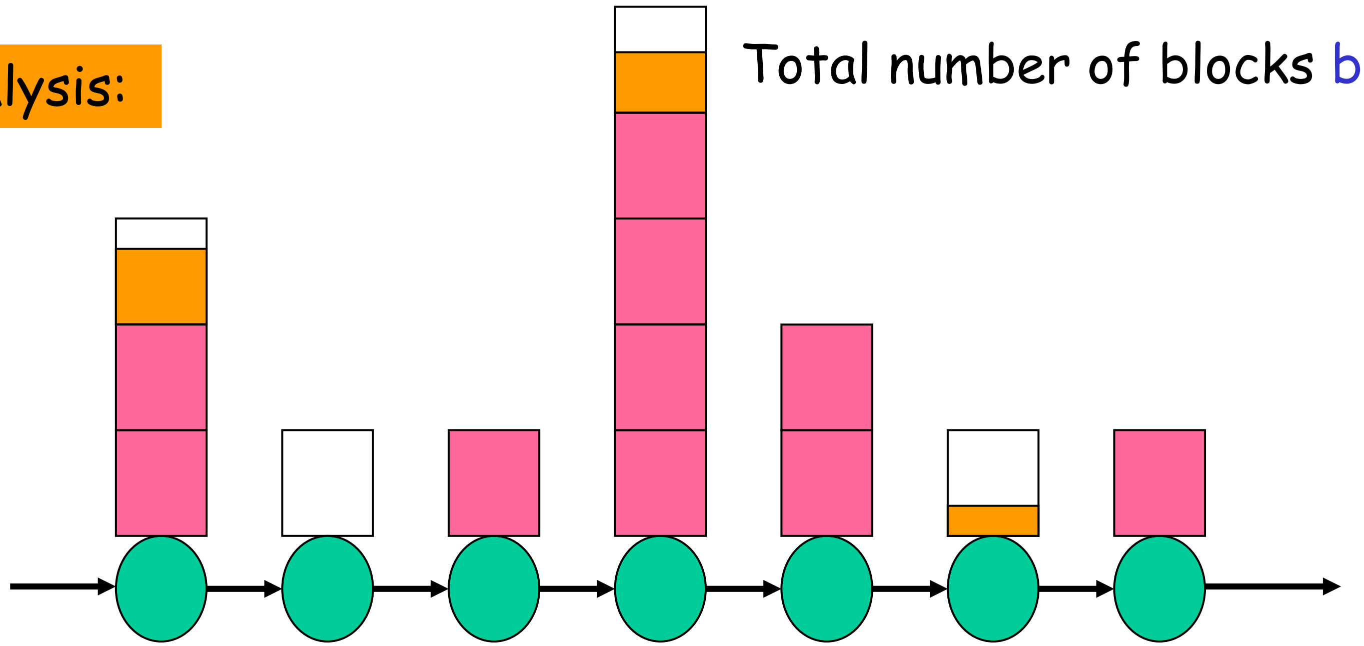

At most one block per process may not be full (may be empty)

MODIFICATION:

- Empty blocks are neither sent nor received
- Only actual data of partial blocks is sent/received
- No process receives a block it already has

Analysis:

Total number of blocks $b = \sum bi$




•b-1 (almost) regular communication rounds

•Each round takes time O(B), total time O((b-1)B) = O((p+∑floor(Bi/B))B) ≈ O(m)

MUCH BETTER than O((p-1)max(mi)) of linear ring without blocking

## General principle:

1. Regular collective operation solves similar irregular problem on clustered system
2. By simulation, algorithm for regular problem on clustered system can be converted to algorithm for irregular problem. ASSUMPTION: communication capability of node and processor similar, eg. one ported
3. Irregular operation (on processes) remains irregular problem on clustered system

⇒ Blocked ring algorithm also works for MPI_Allgatherv on clustered system

## Choosing the block size B:

1. All mi>0: choose B=min(mi) – smallest data of some process (not too small, threshold).
2. Fixed block size
3. Some mi=0: number of rounds is m/B+(p-z)/2, z number of mi=0, assuming partial blocks half full. Time per round in linear model is $\alpha+\beta B$, best block size $\approx \sqrt{[2\alpha m/\beta(p+z-2)]}$

1. All processes busy in all rounds, for regular problems algorithm identical to linear ring
2. Simple solution - can lead to load imbalance for some distributions
3. Optimizes pipelining effect, for extreme problems with m0=m, mi=0 similar to pipelined broadcast. Linear model not accurate enough!

# Experimental results

Comparison of blocked ring algorithm to standard ring:

- Performance
- Load balance
- Effect of block size B

## Target systems

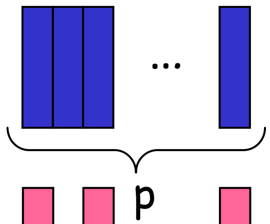
- NEC SX-8 - up to 30 nodes used
- Linux clusters with IB and Gig. Ethernet - 16 nodes, 24 nodes
- IBM Blue Gene/P - up to 4096 processes
- SiCortex 5832 - 5784 processes

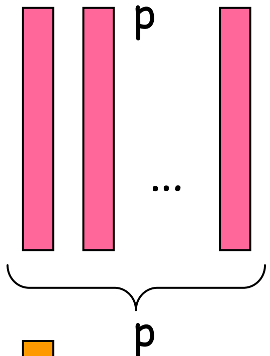


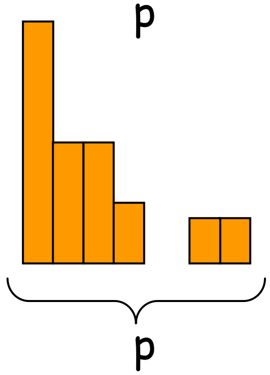

NEC

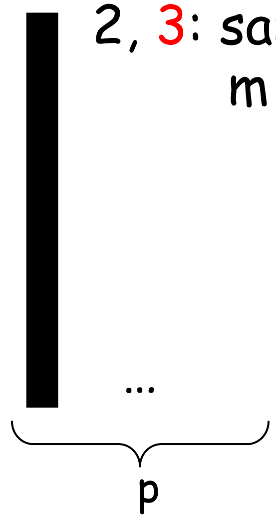
Distributions

1. Regular
2. Broadcast
3. Spike
4. Half
5. Linear
6. Geometric

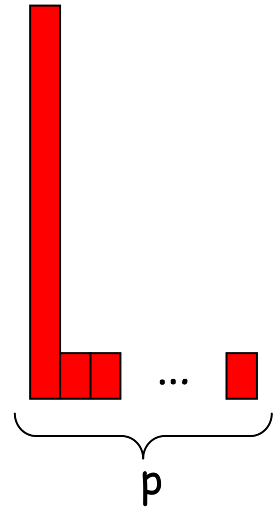


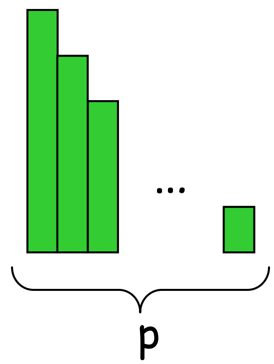




2, 3: same total amount of data
m = c, c is Base Size

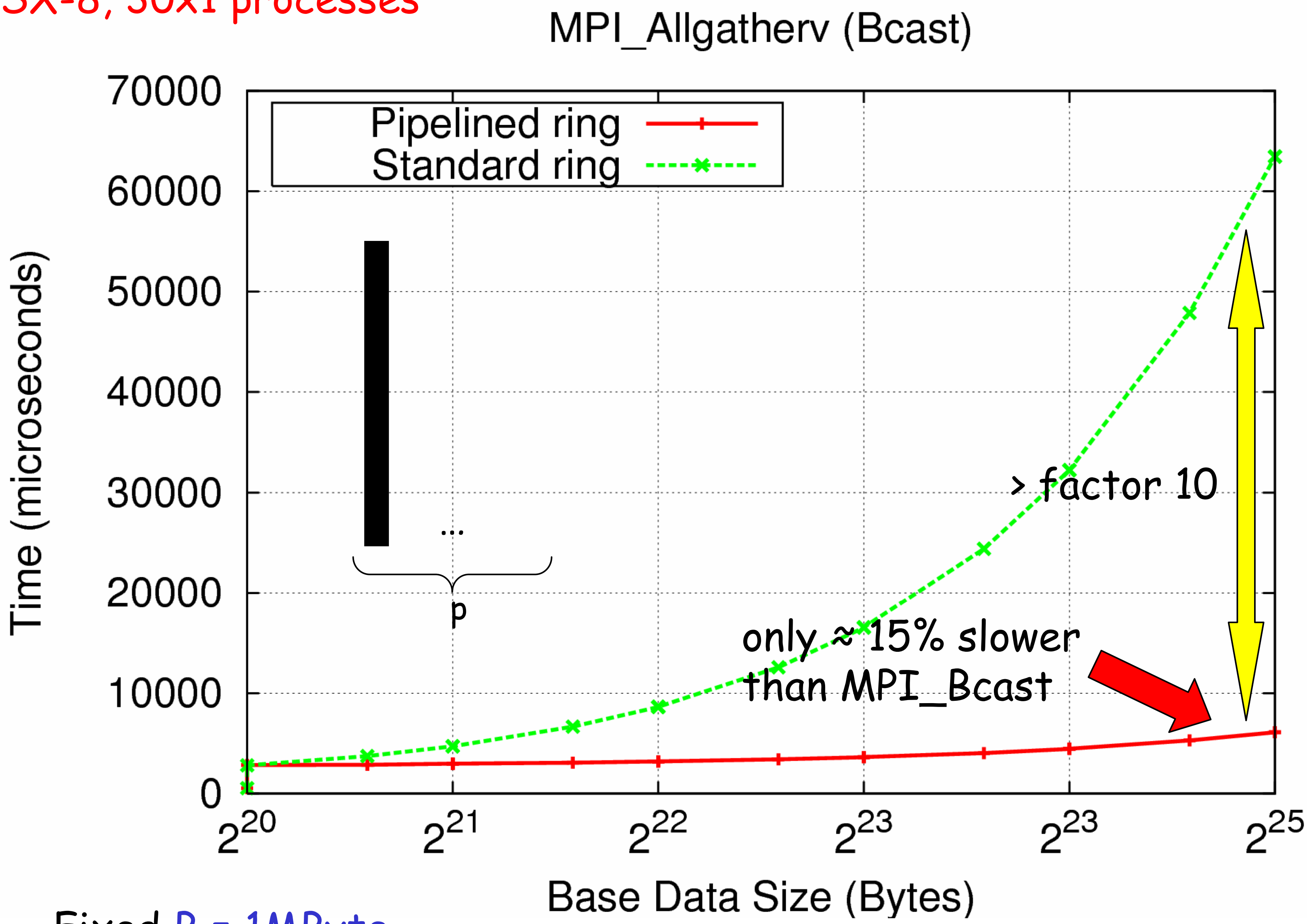



1, 4, 5, 6: same total amount of data
m = pc, c is Base Size

⇨ Comparable running times

SX-8, 30x1 processes

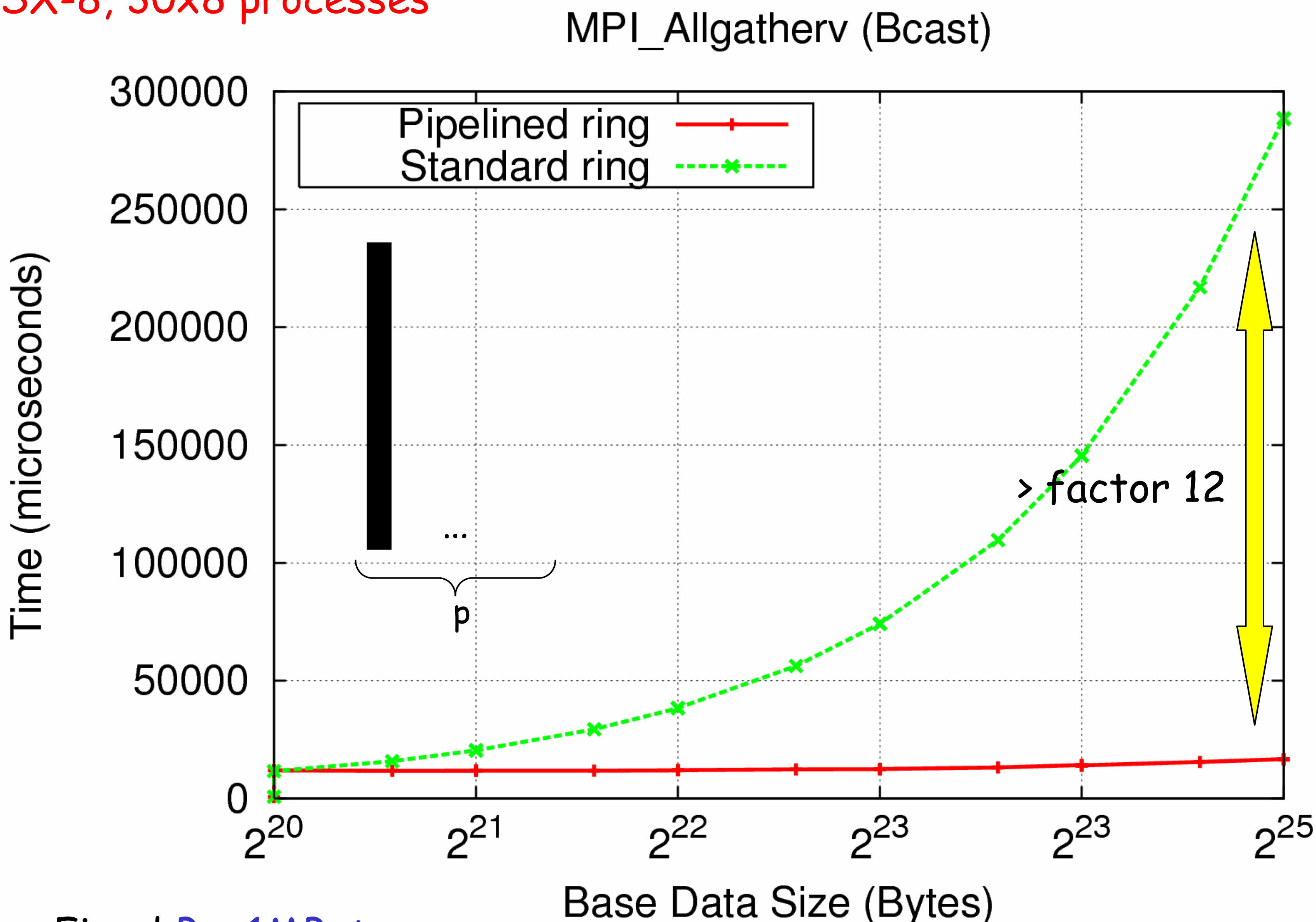



Fixed B = 1MByte



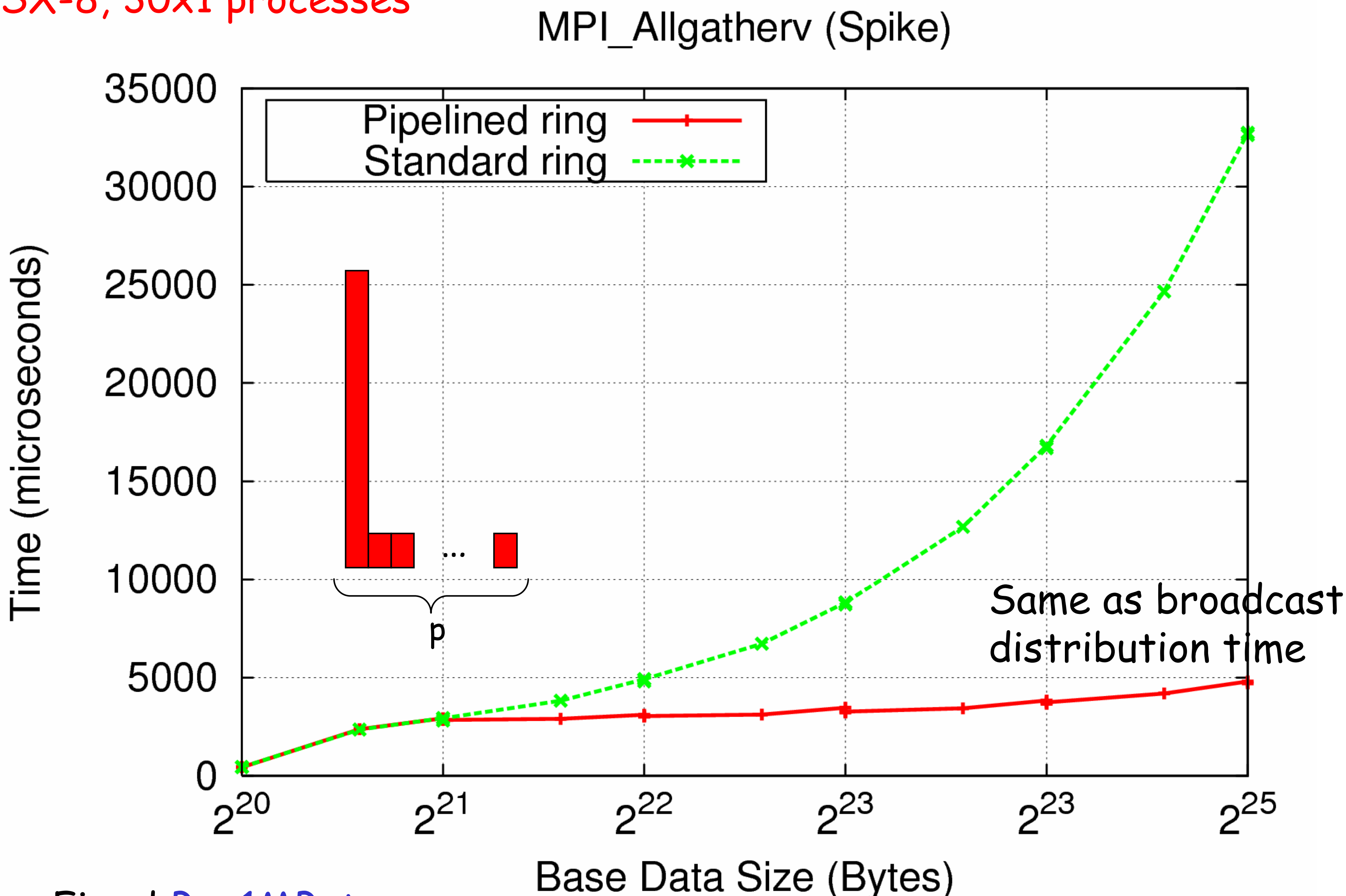

NEC

SX-8, 30x8 processes




Fixed B = 1MByte

SX-8, 30x1 processes

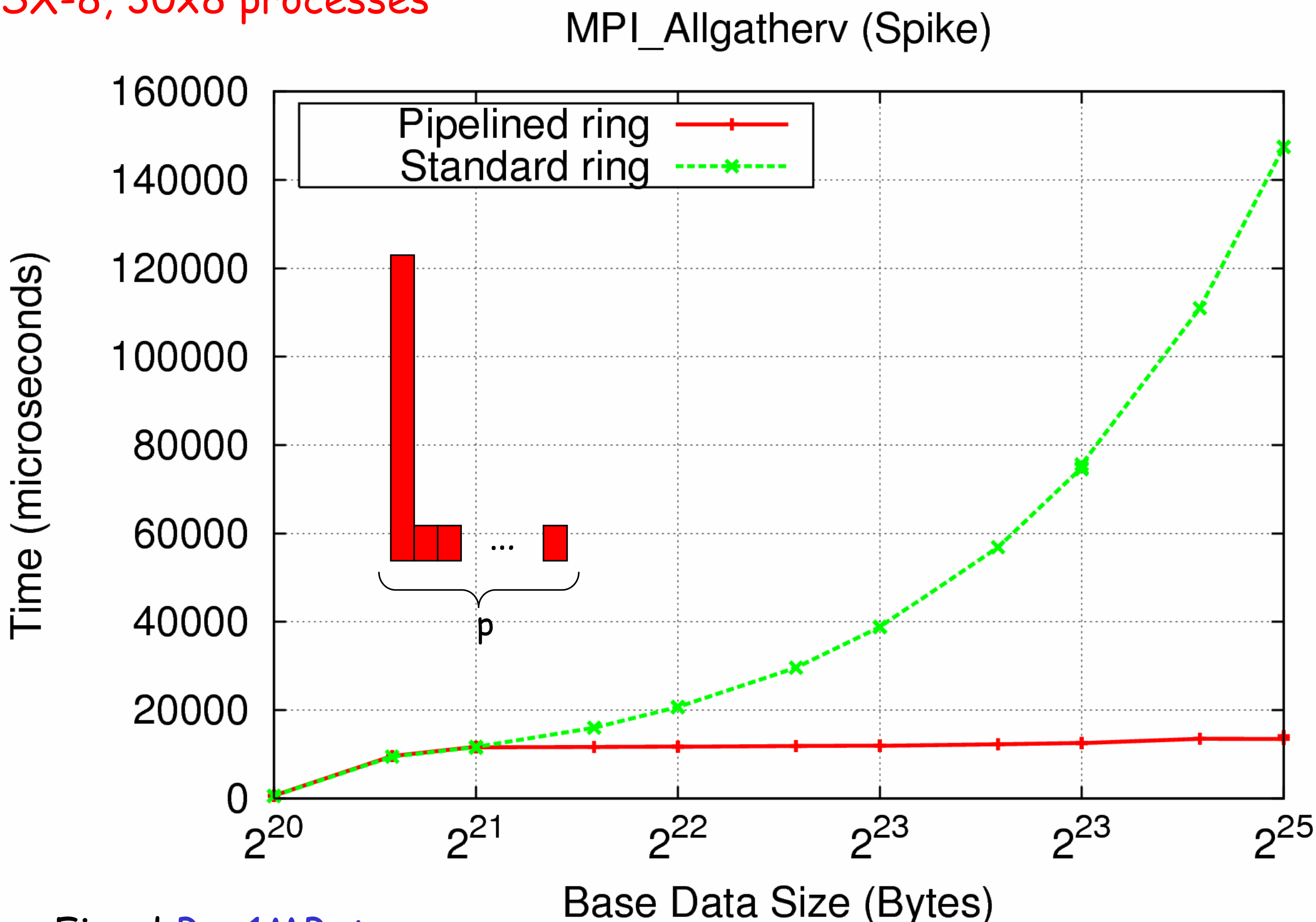



Fixed B = 1MByte



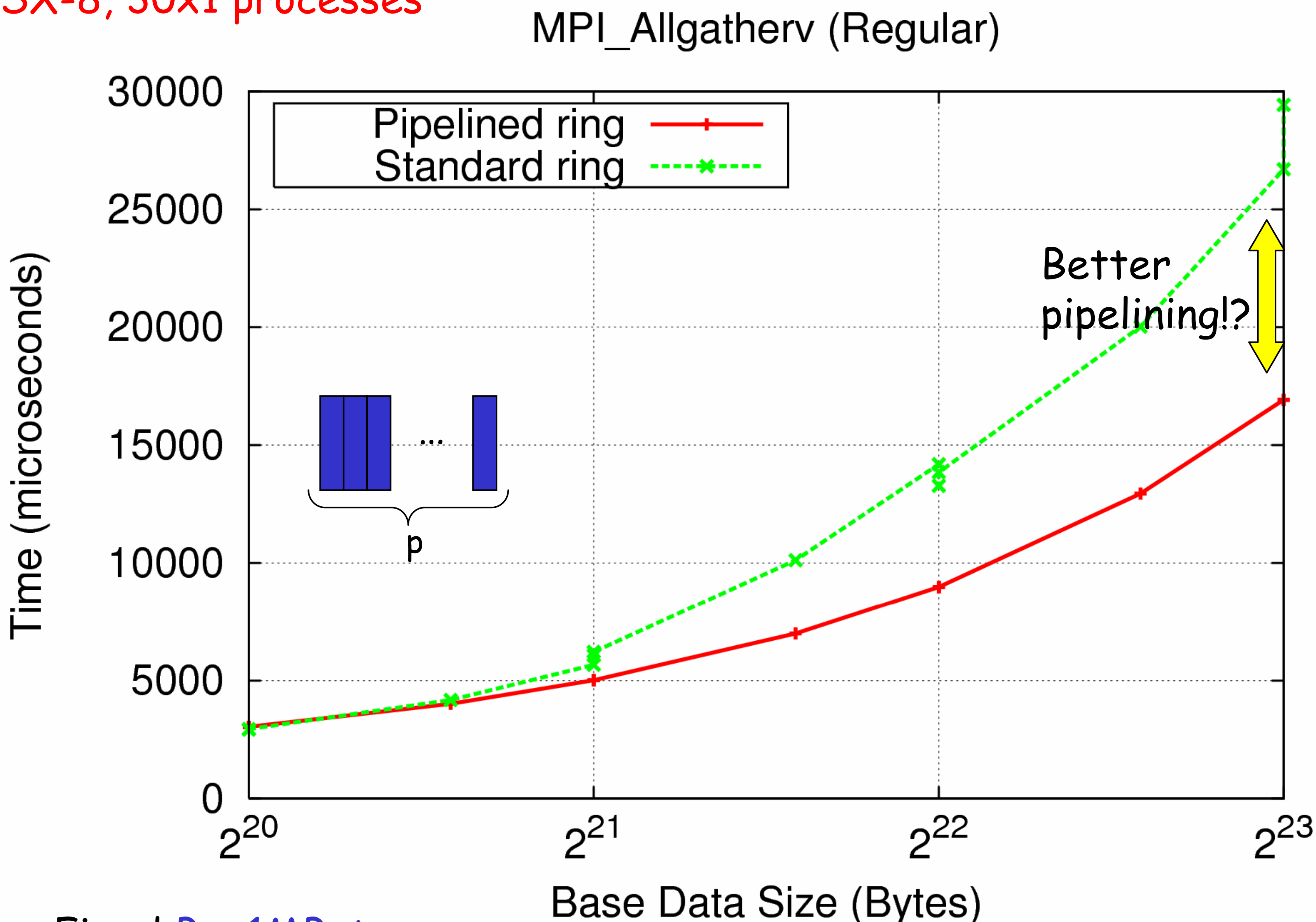

NEC

SX-8, 30x8 processes



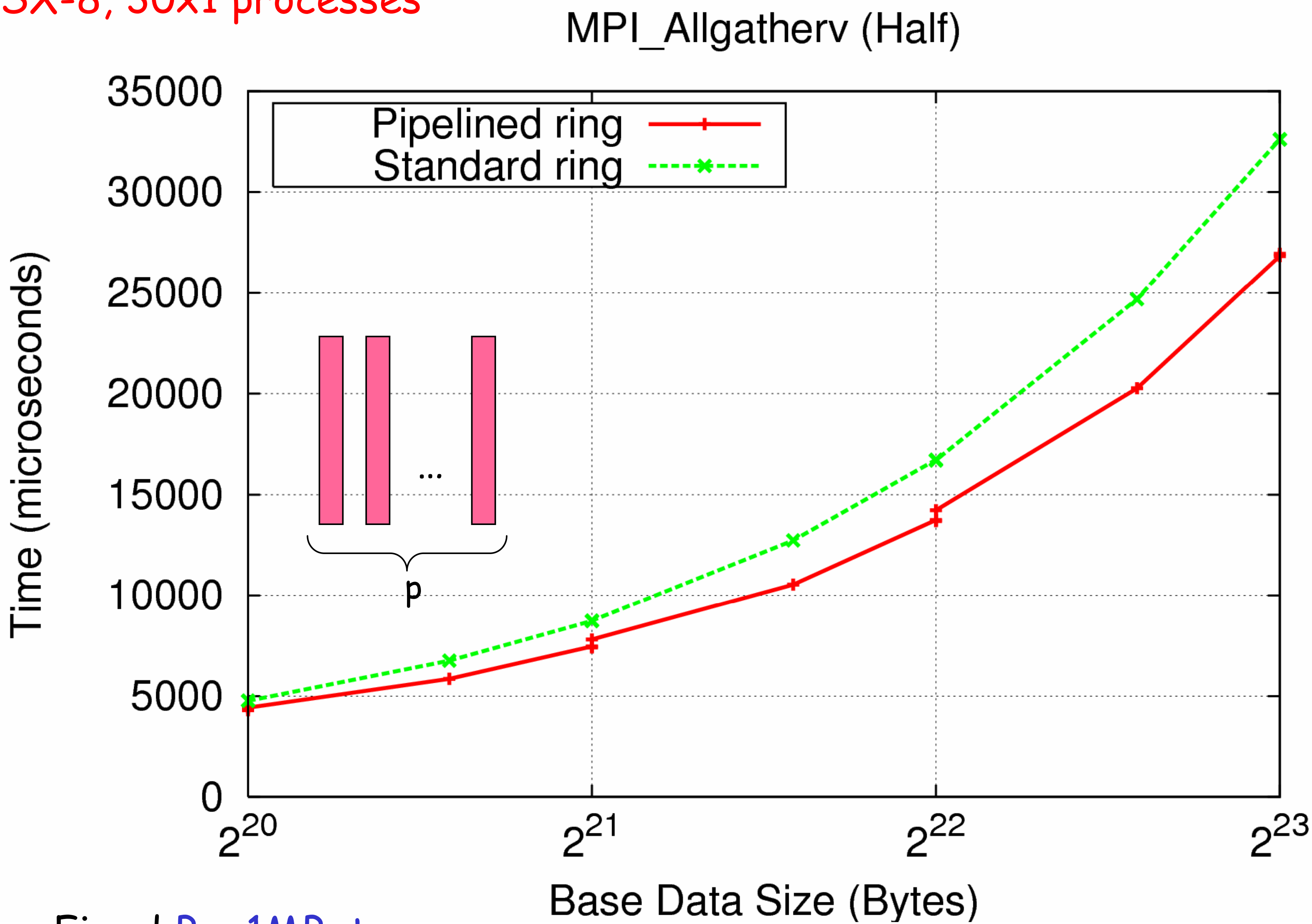

Fixed B = 1MByte

SX-8, 30x1 processes



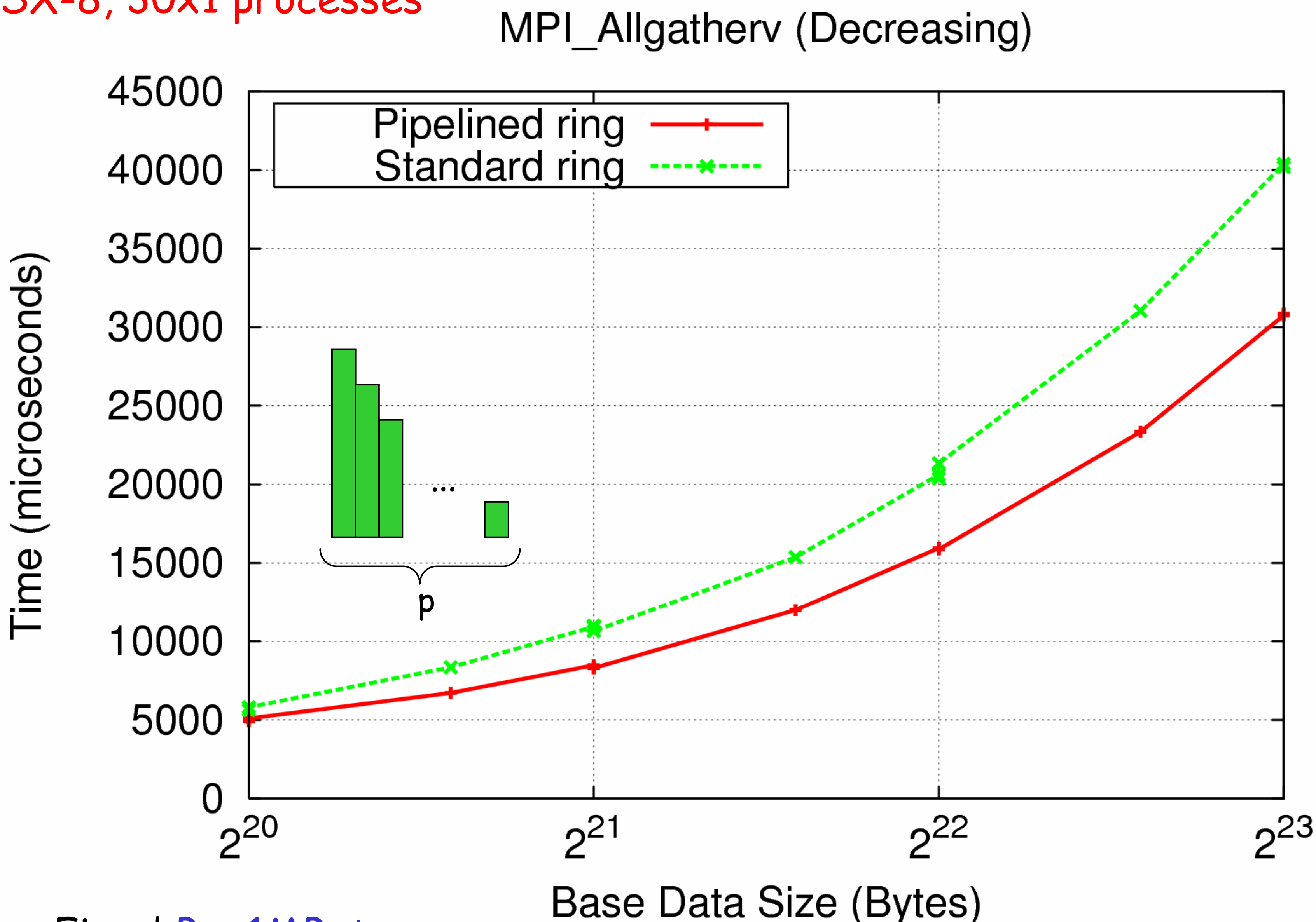

Fixed B = 1MByte

SX-8, 30x1 processes



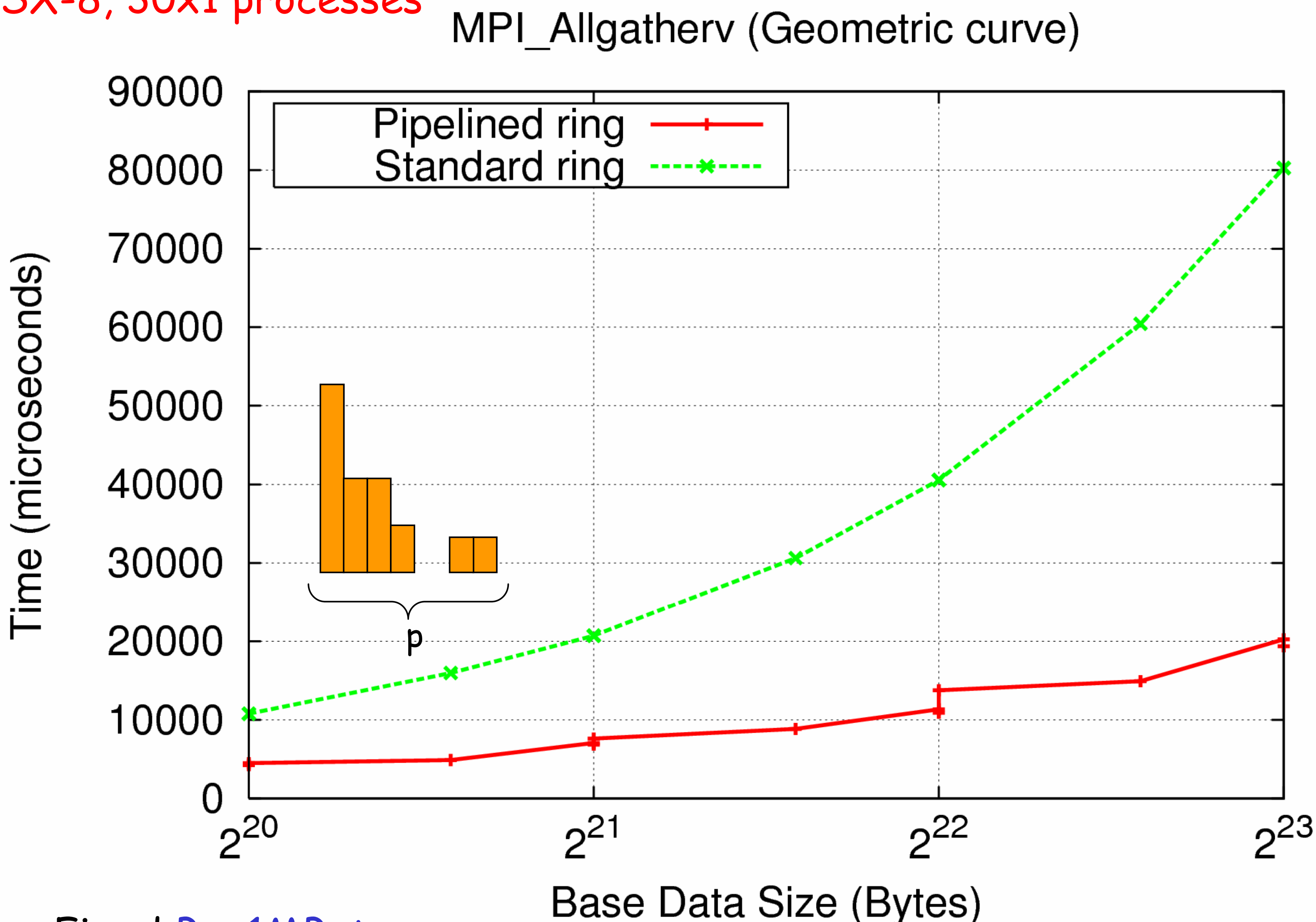

Fixed B = 1MByte

SX-8, 30x1 processes

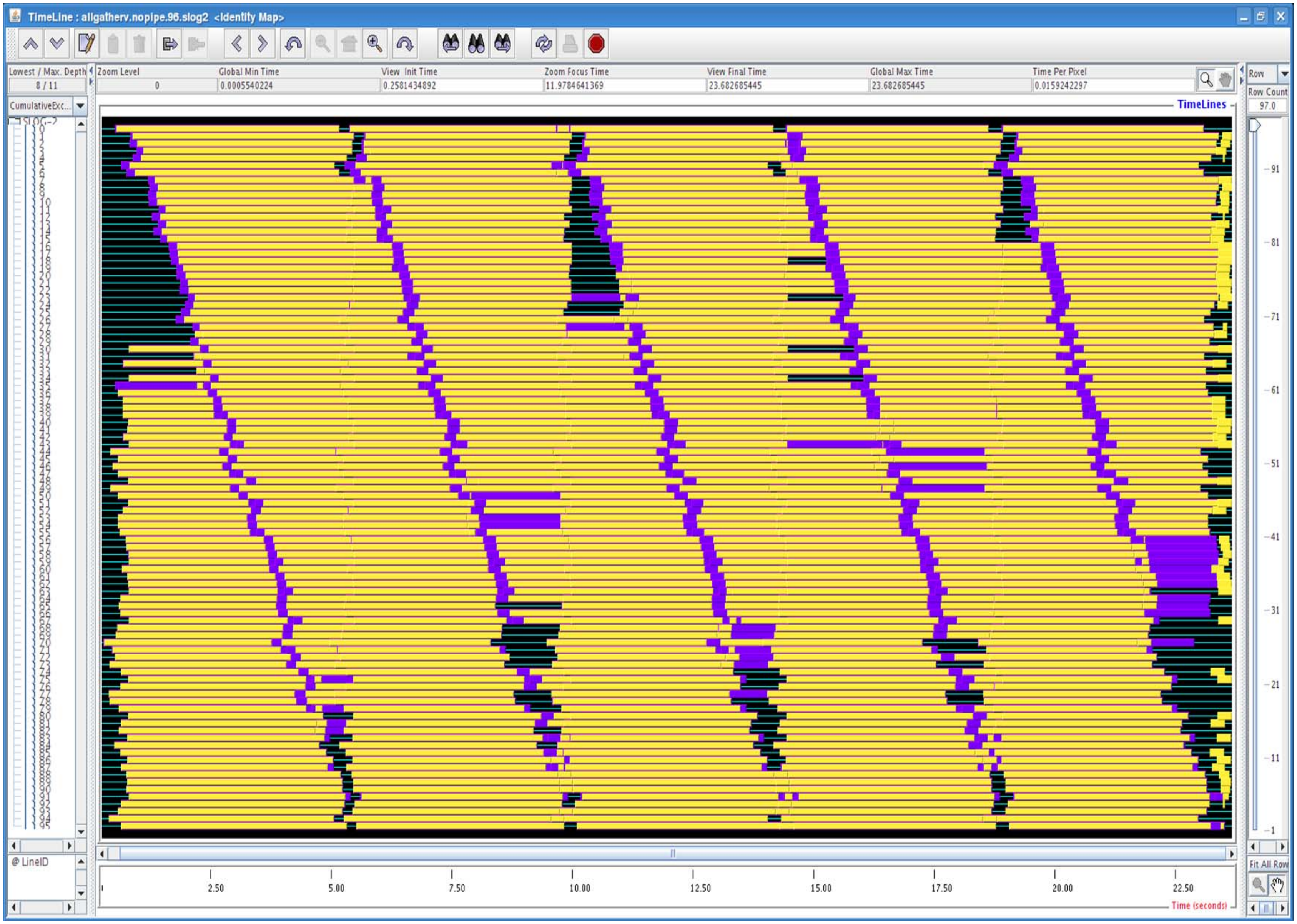



Fixed B = 1MByte

SX-8, 30x1 processes

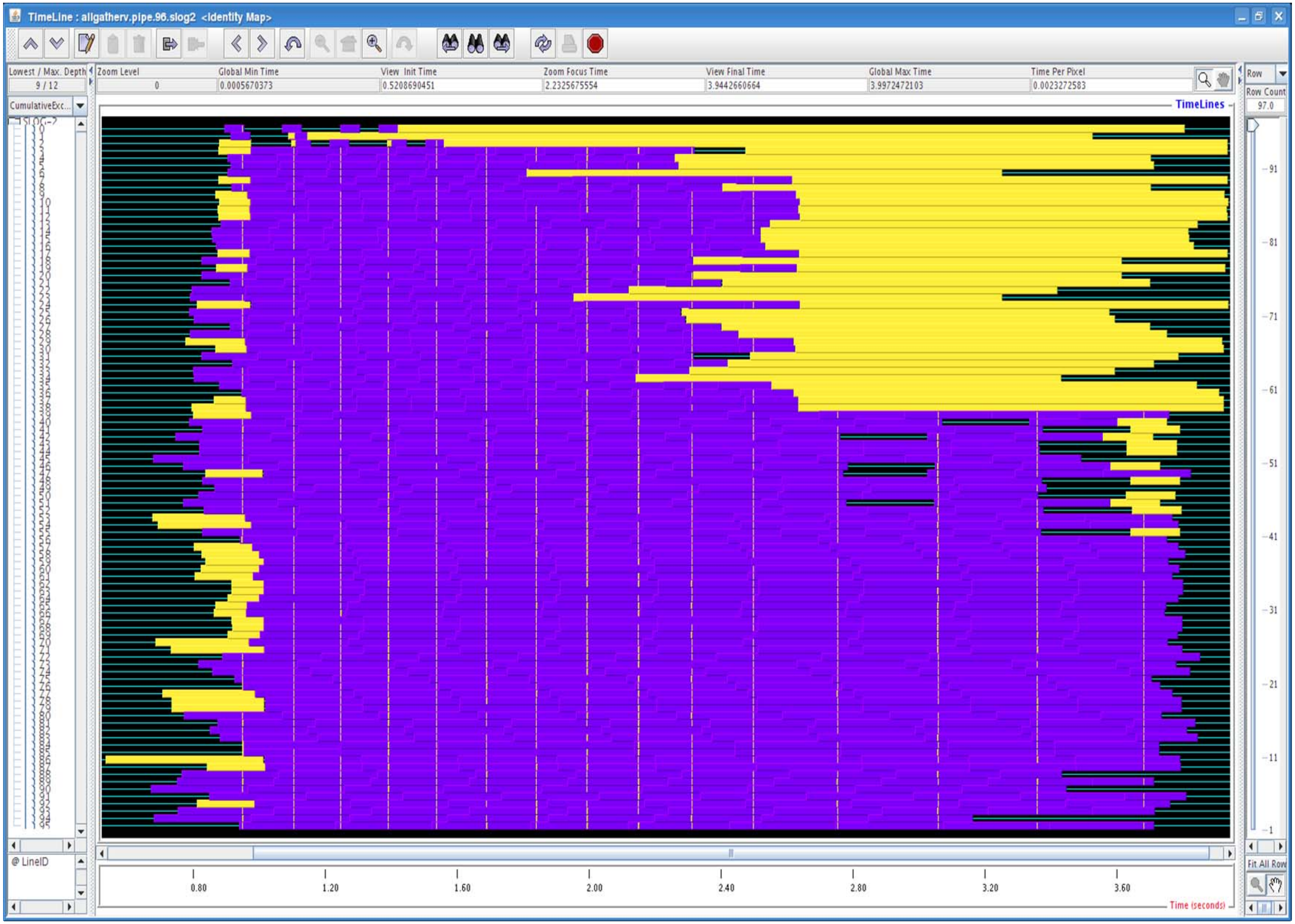



Fixed B = 1MByte


TimeLine : allgatherv.nopipe.96.slog2 <Identity Map>
Lowest / Max. Depth
8 / 11
Zoom Level
0
Global Min Time
0.0005540224
View Init Time
0.2581434892
Zoom Focus Time
11.9784641369
View Final Time
23.682685445
Global Max Time
23.682685445
Time Per Pixel
0.0159242297
Row Count
97.0
CumulativeExc...
TimeLines
2.50
5.00
7.50
10.00
12.50
15.00
17.50
20.00
22.50
Time (seconds)
Fit All Rows

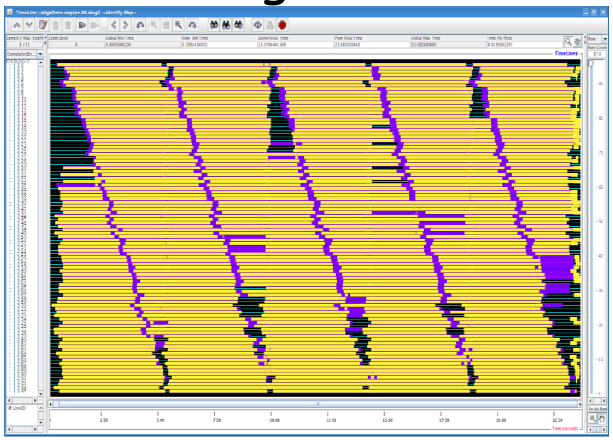


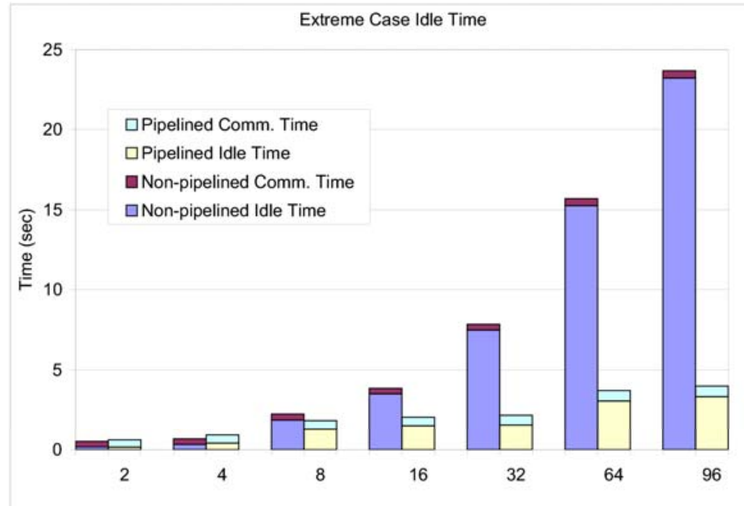

NEC


TimeLine : allgatherv.pipe.96.slog2 <Identity Map>
Lowest / Max. Depth
9 / 12
Zoom Level
0
Global Min Time
0.0005670373
View Init Time
0.5208690451
Zoom Focus Time
2.2325675554
View Final Time
3.9442660664
Global Max Time
3.9972472103
Time Per Pixel
0.0023272583
Row
Row Count
97.0
TimeLines
Time (seconds)
Fit All Row

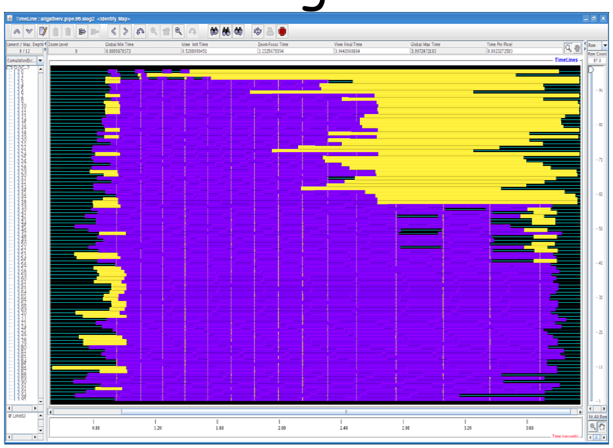
# Linux cluster, 96 processes

Linear ring





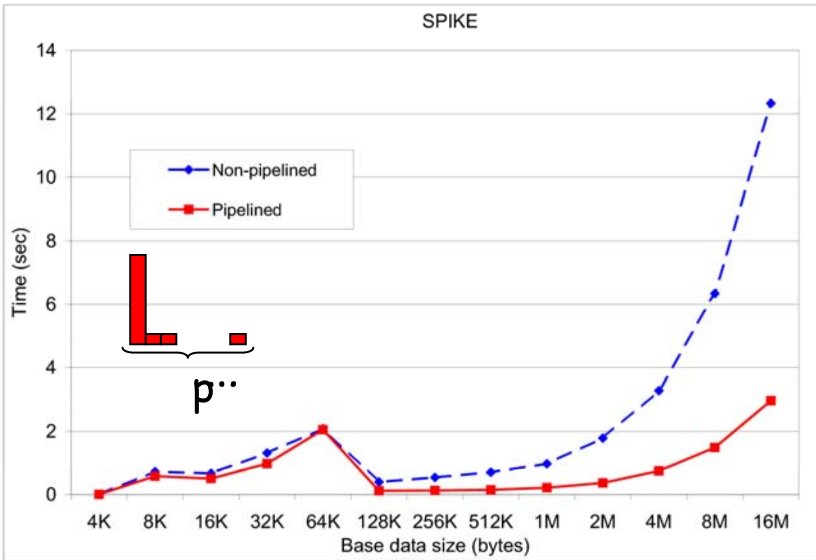

Blocked ring

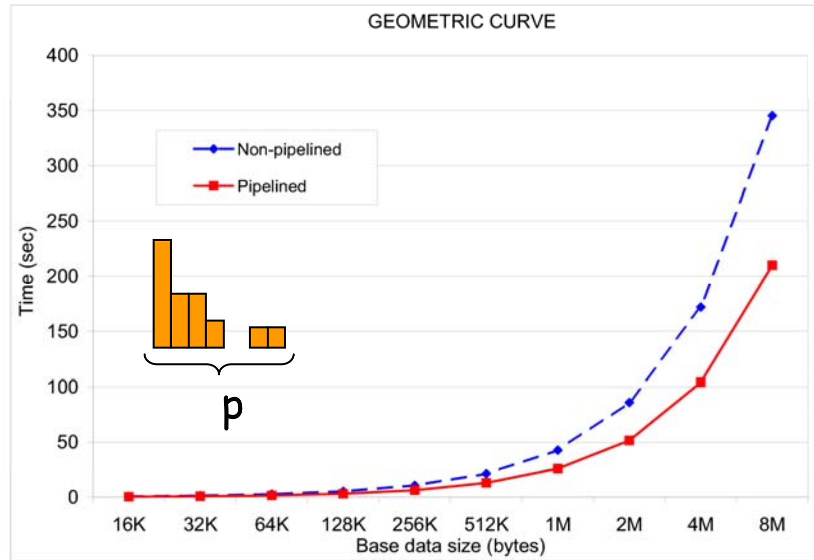


Yellow: idle time

Blue: communication time

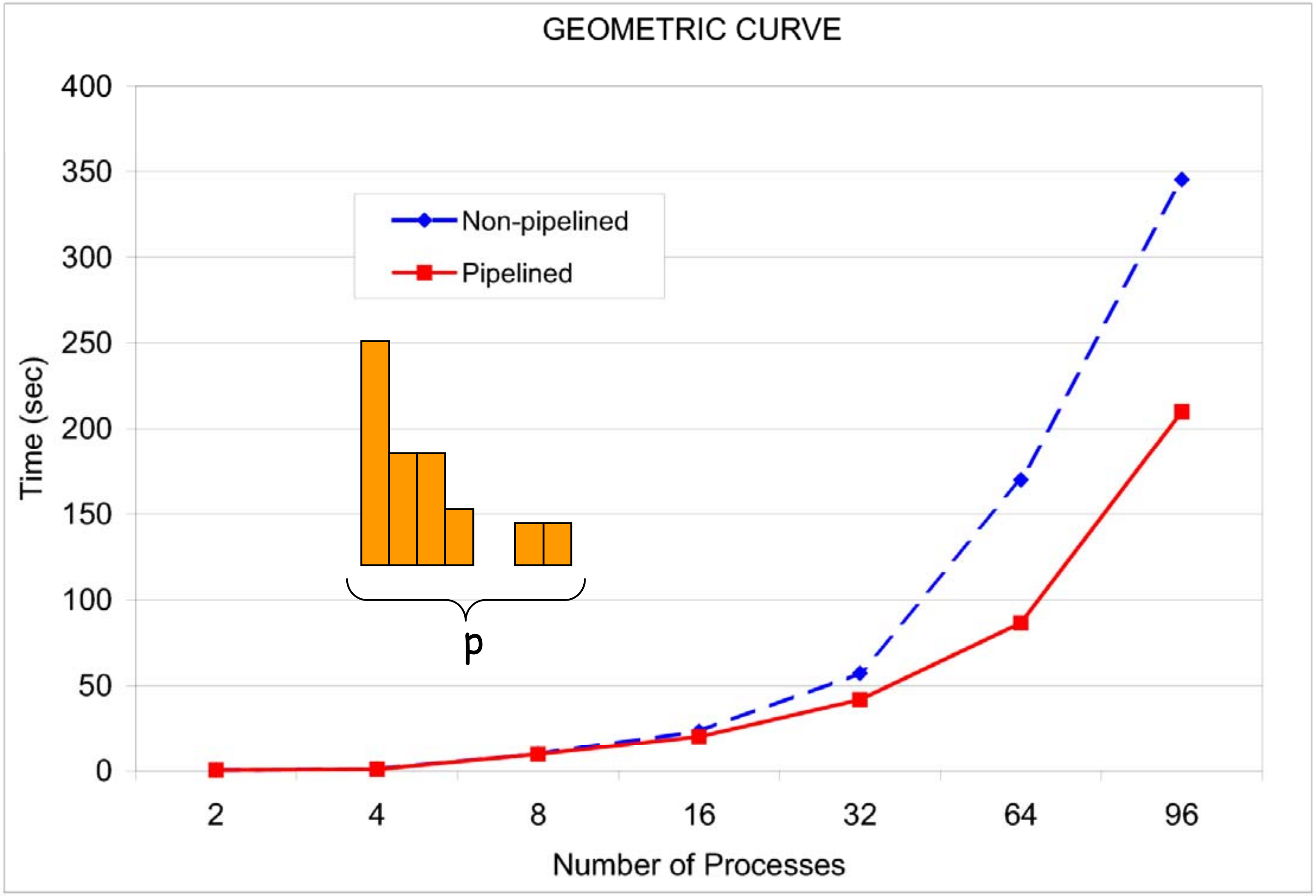
# Linux cluster, 96 processes



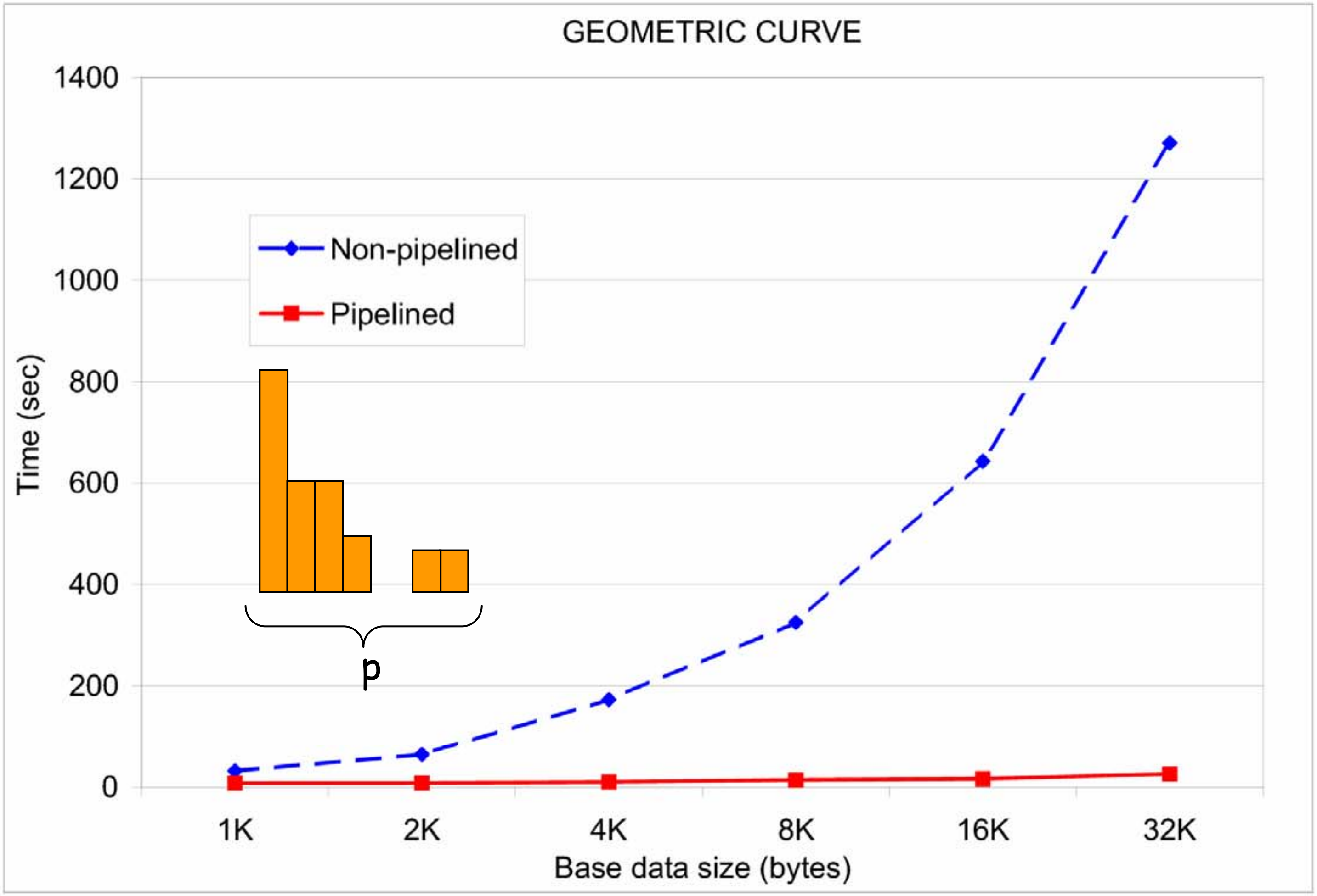

Fixed B = 32KByte

# Linux cluster, varying number of processes



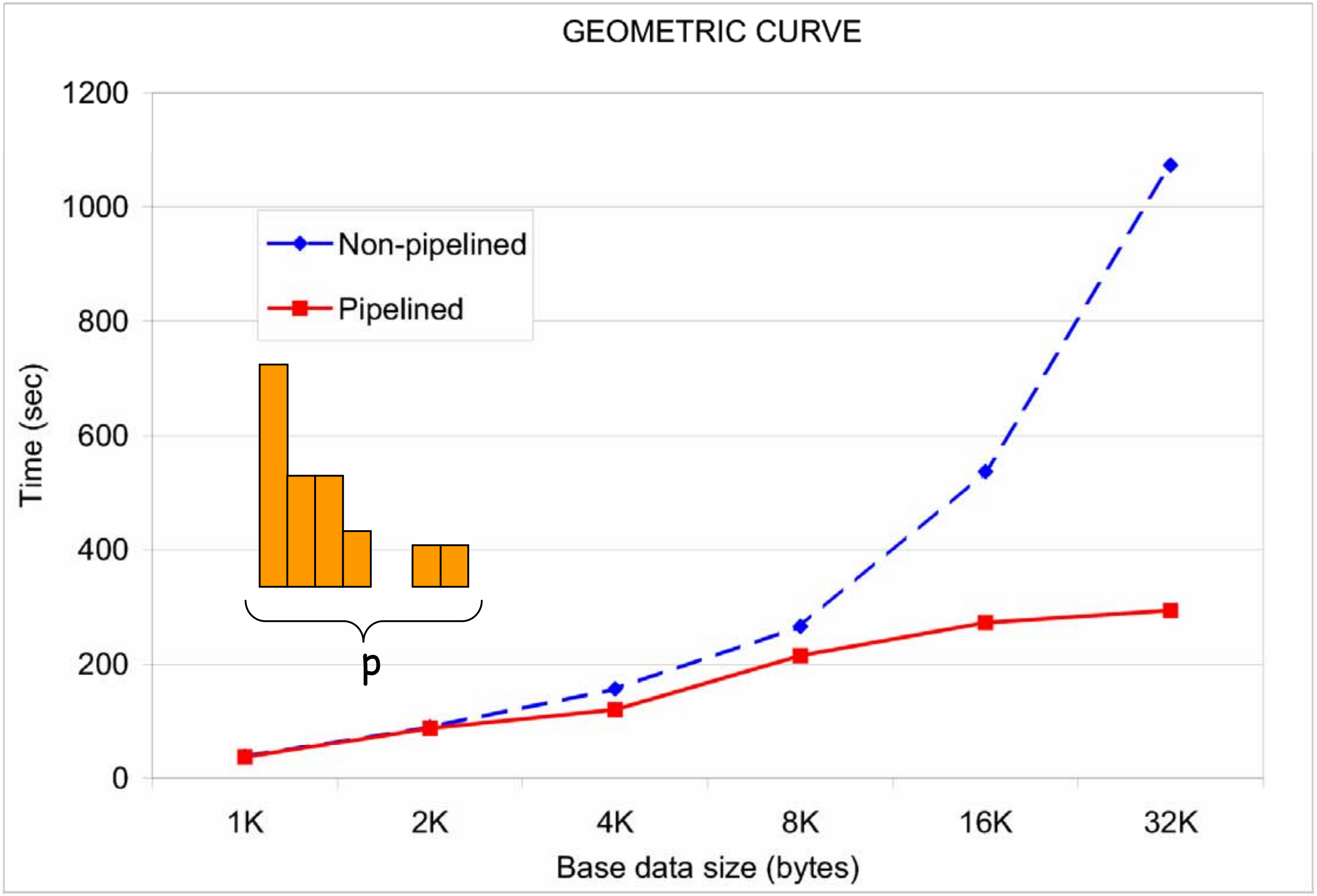

Fixed B = 32KByte

# Blue Gene/P, 4096 processes



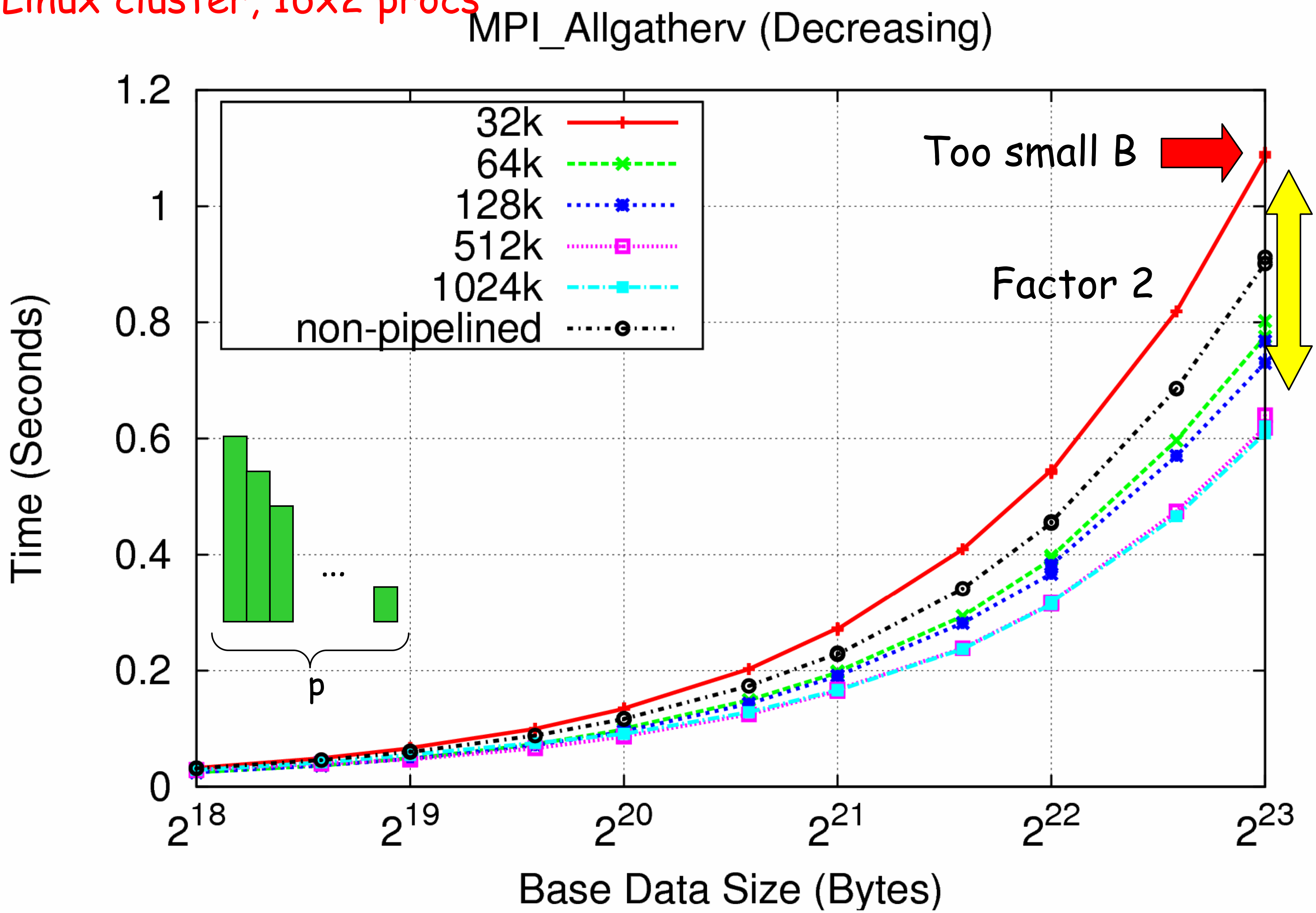

Fixed B = 64KByte

# SiCortex, 5784 processes

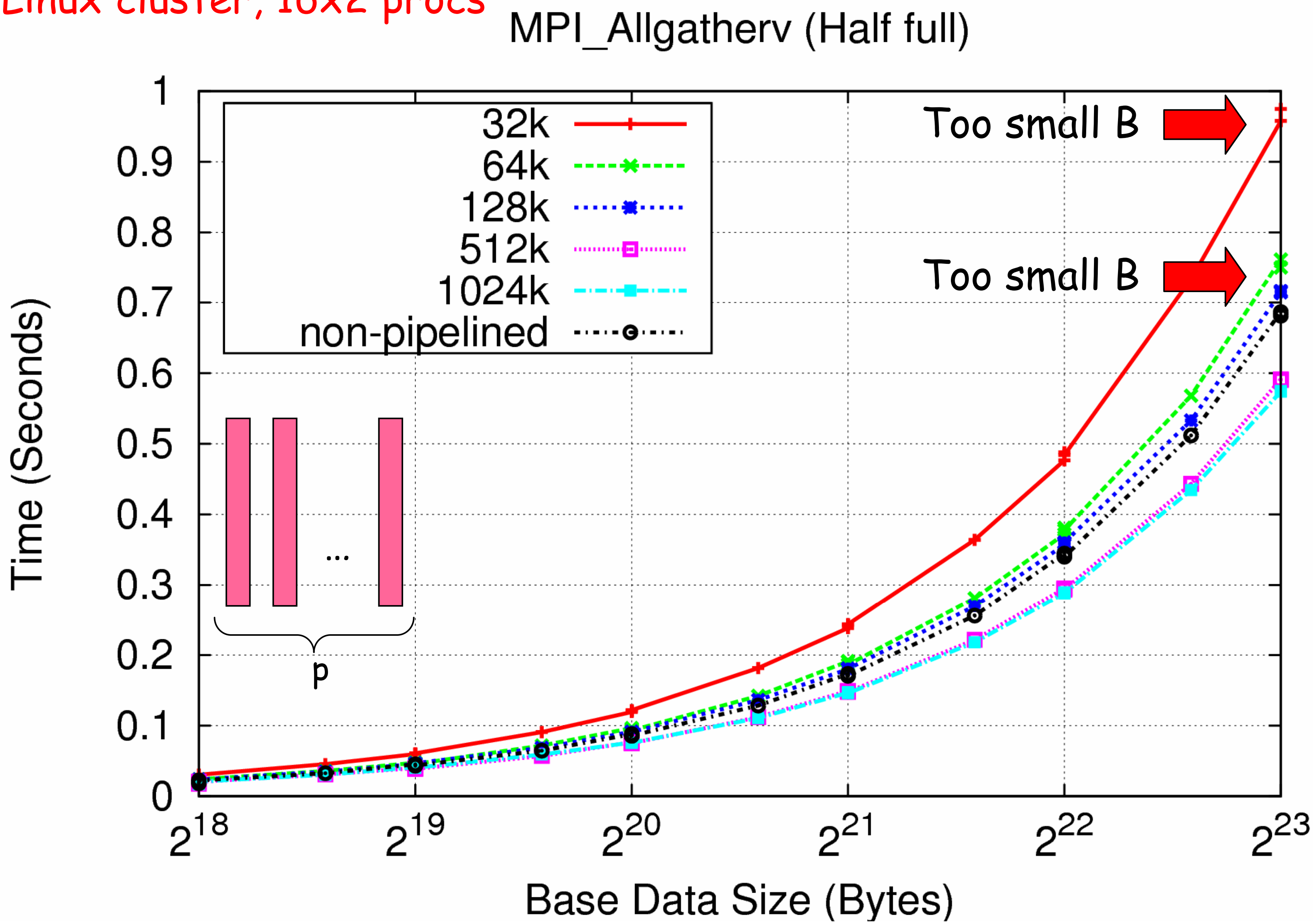



Fixed B = 1MByte

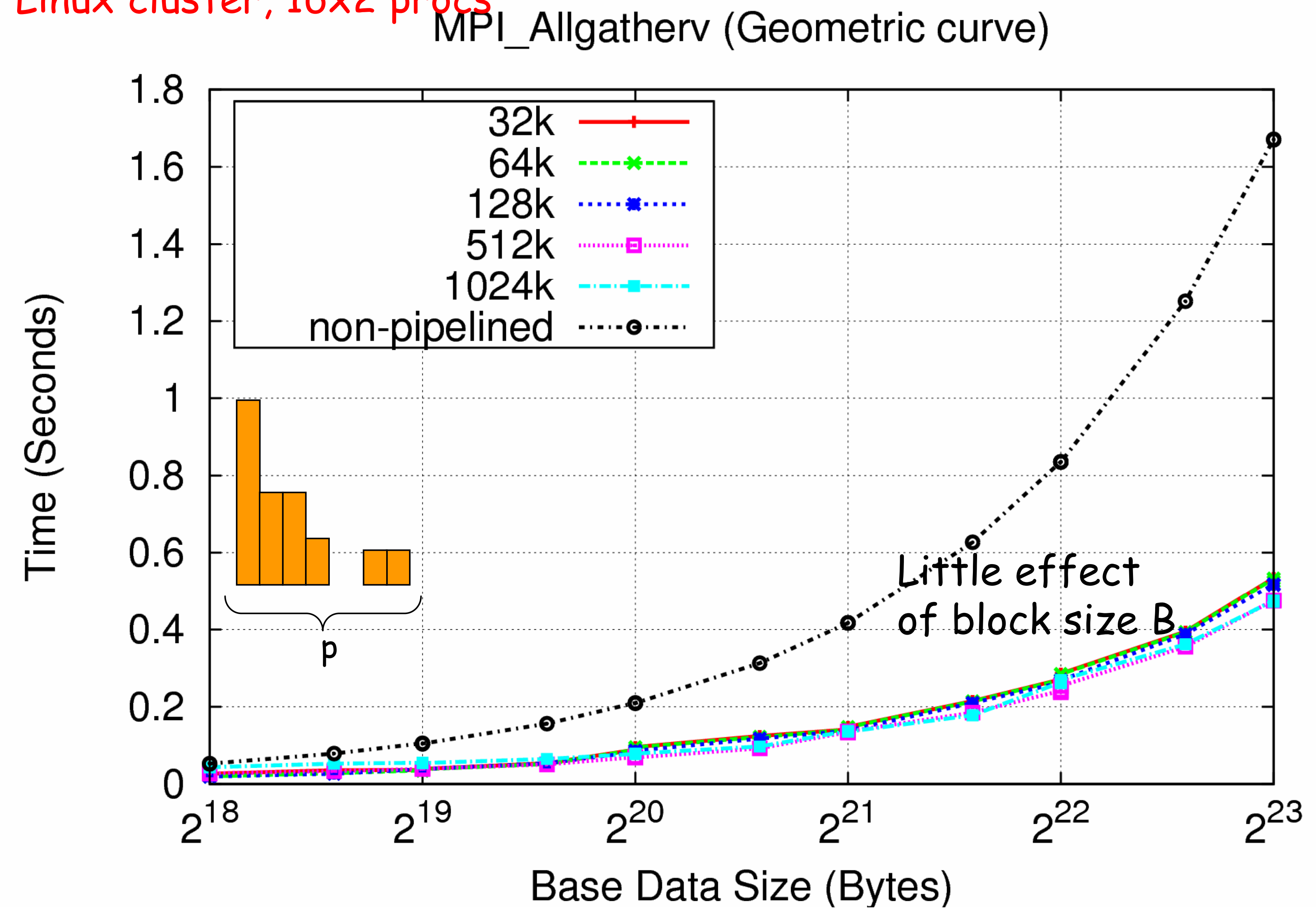
Linux cluster, 16x2 procs



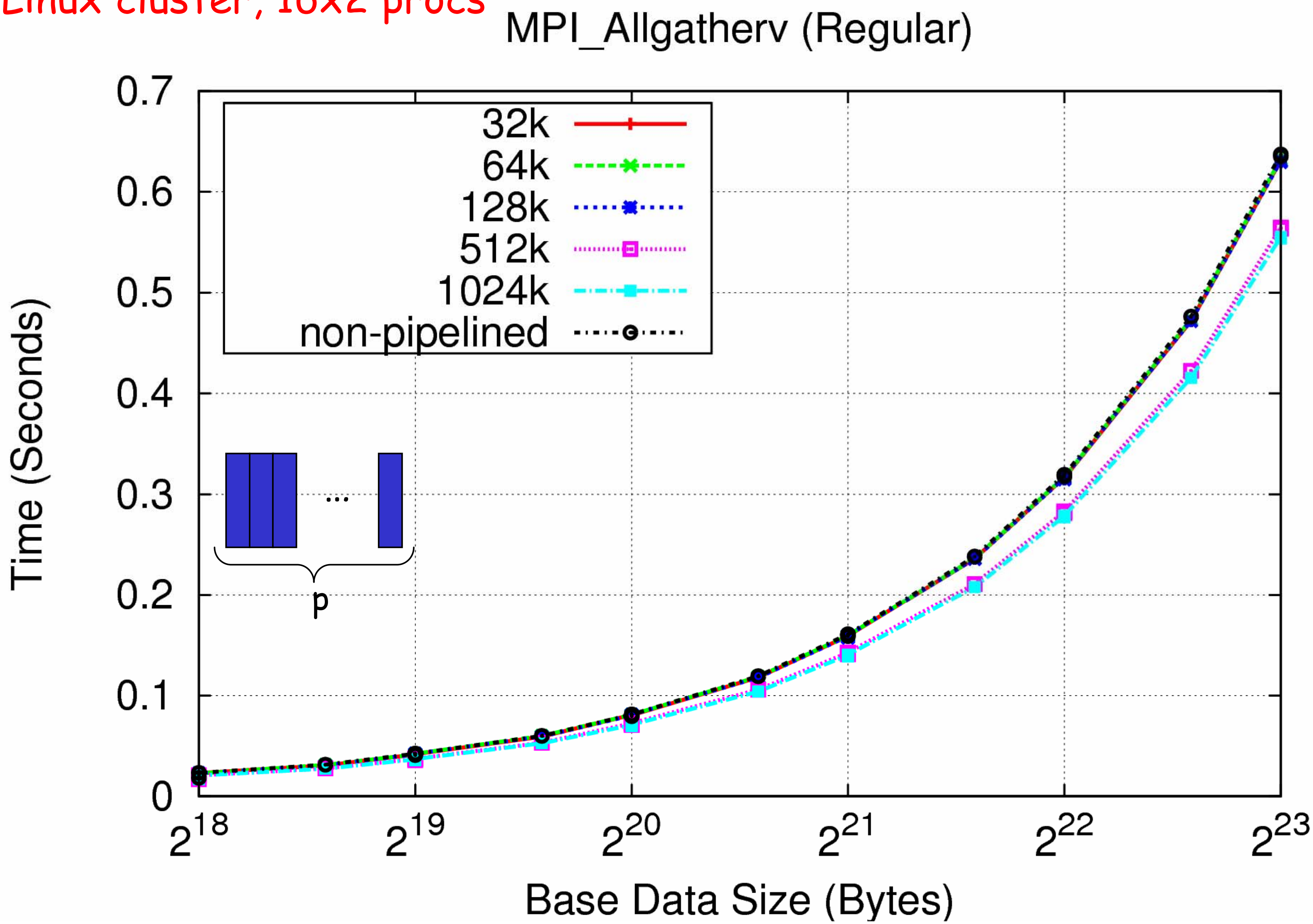
Linux cluster, 16x2 procs



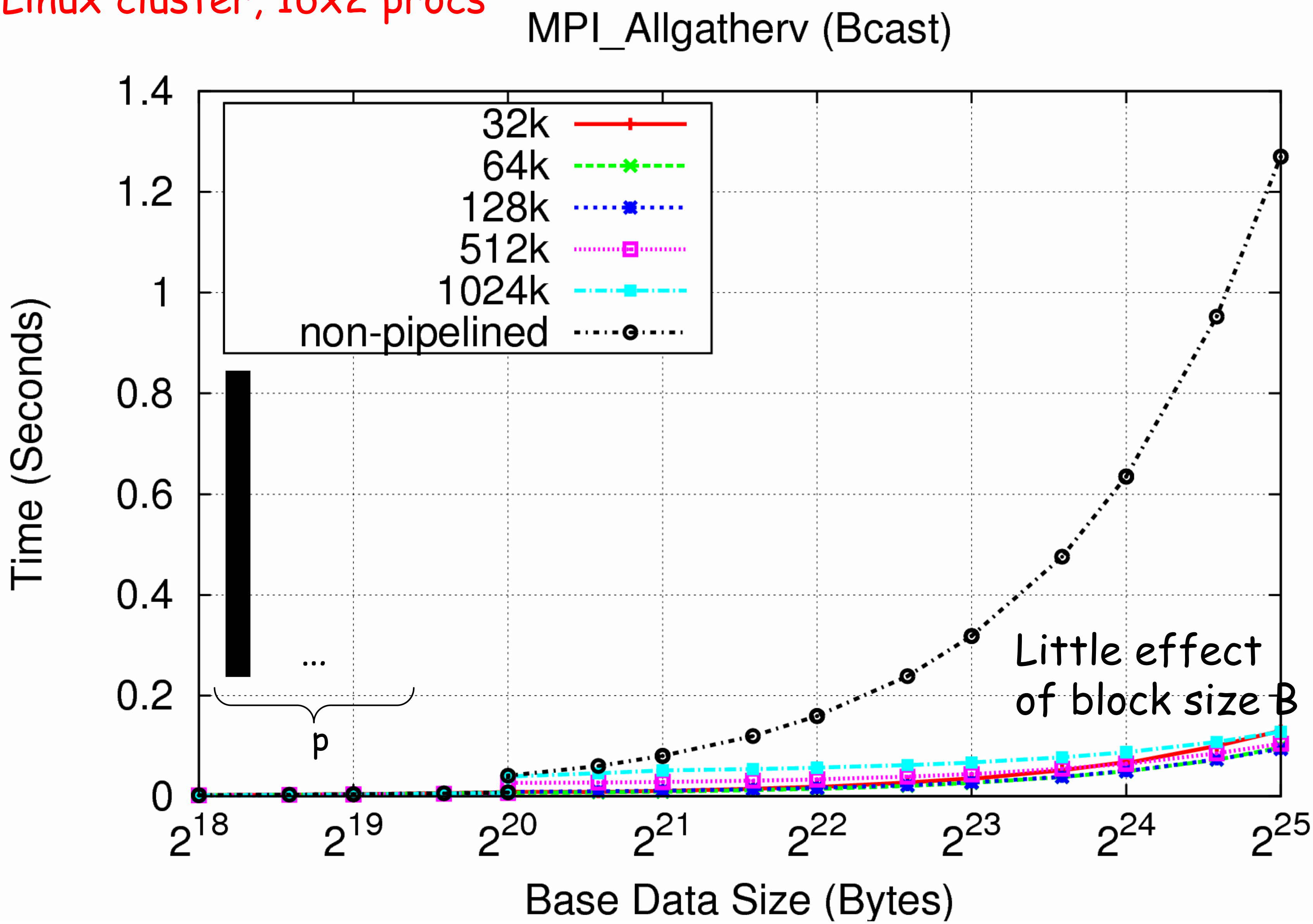
Linux cluster, 16x2 procs



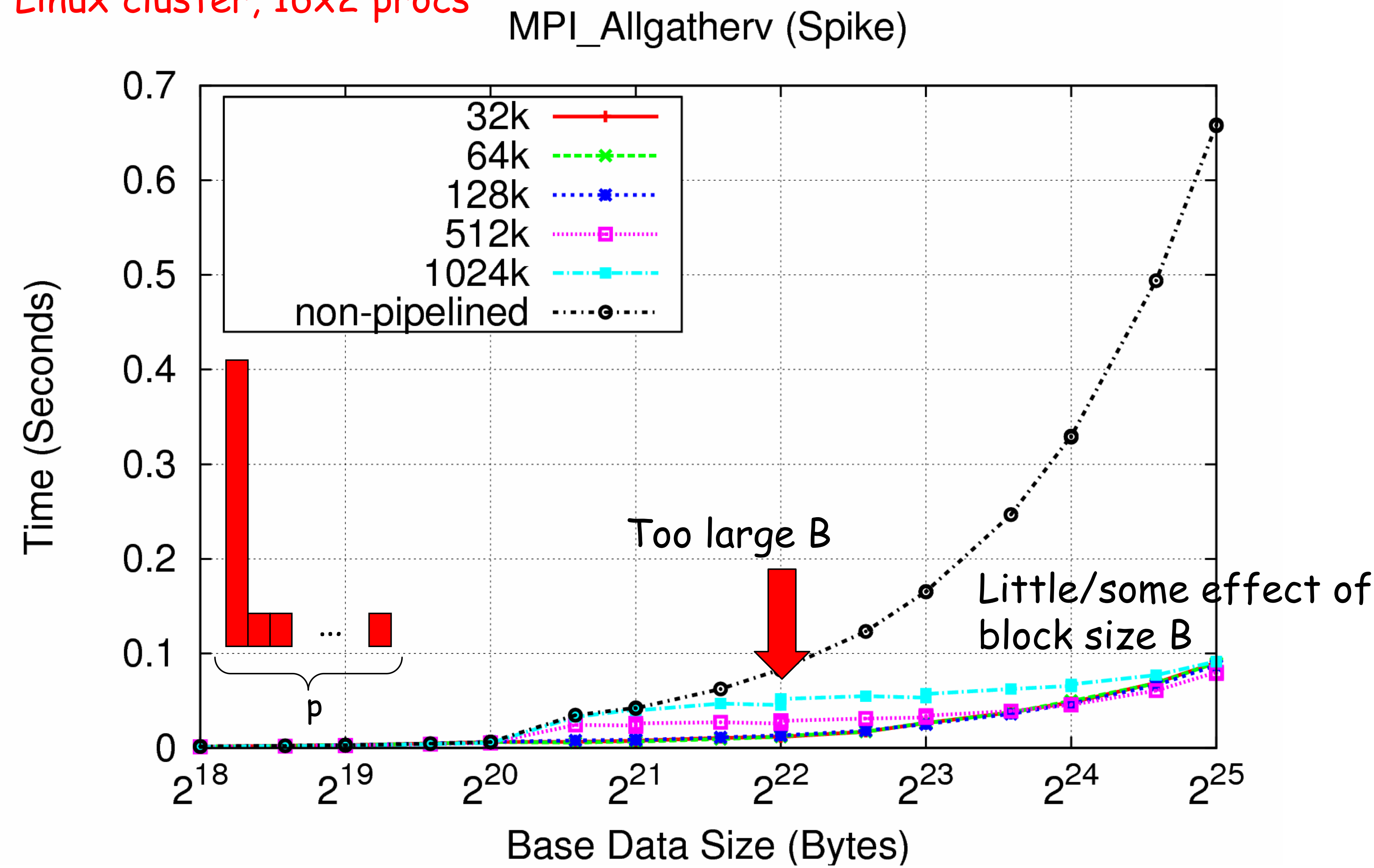
Linux cluster, 16x2 procs



Linux cluster, 16x2 procs



Linux cluster, 16x2 procs



## Summary

- Simple, blocked linear ring algorithm for MPI_Allgatherv
    - NEW? Observation not found in literature
- Large performance gains for large problems on different systems
- Good limit behavior: identical to linear ring for regular problems, similar to pipelined broadcast for extreme distributions
- Tuning of block size: dependent on data distribution, linear model inadequate, experimental work needed
- There are relationships between regular and irregular collectives (on processes and nodes) that can (sometimes) be exploited for design of new algorithms