Scalable Communication Protocols for Dynamic Sparse Data Exchange

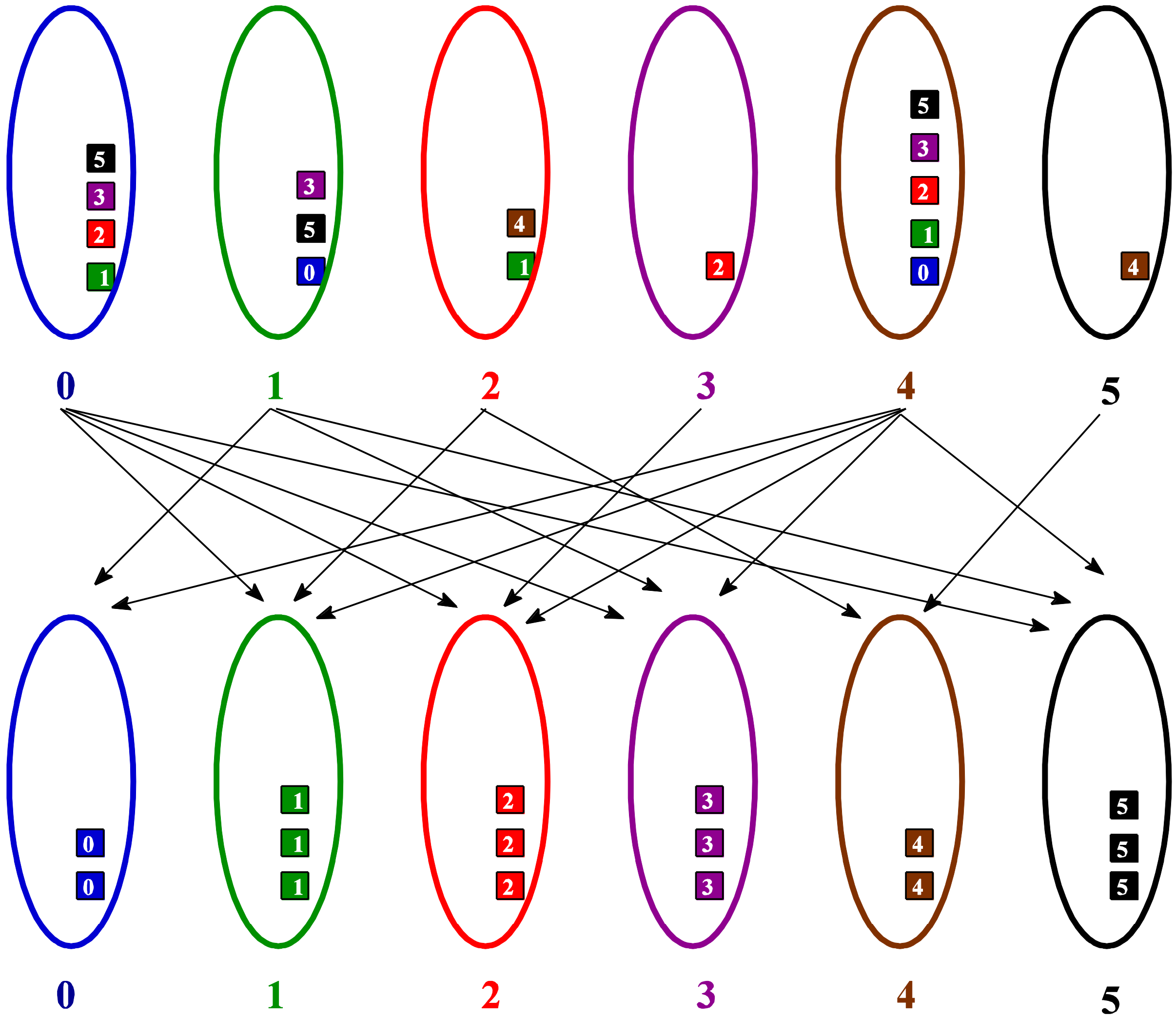
Torsten Hoefler, Christian Siebert, Andrew Lumsdaine

PPoPP 2010, Bangalore, India

The Sparse Data Exchange Problem

- Defines a generic communication problem
  - Assume a set of P processes
  - Each process communicates with a small set of other processes (called neighbors)
- How do we define “sparse”?
  - The maximum number of neighbors (k) is $\mathcal{O}(\log P)$
- Dynamic vs. Static SDE
  - Static: neighbors can be determined off-line
    - e.g., sparse matrix vector product
  - Dynamic: neighbors change during computation
    - e.g., parallel BFS

Dynamic Sparse Data Exchange (DSDE)



Our Contribution

- Analyze well-known algorithms for DSDE:
  - Personalized Exchange (MPI_Alltoall)
  - Personalized Census (MPI_Reduce_scatter)
  - Remote Summation (MPI_Accumulate)
  - Focus on large-scale systems (large P)
    - Metadata exchange easily dominates runtime!

- Propose a new, asymptotically optimal algorithm
  - Uses nonblocking collective semantics (MPI_Ibarrier)
  - Can take advantage of hardware support
  - Introduces a new way of thinking about synchronization

Preliminaries

- Distributed Consensus
  - All processes agree on a single value
  - Lower bound: broadcast $T_{BC}(P)$

$$\log_2(P) \cdot o \leq T_{BC}(P) \leq \log_2(P) \cdot (L + 2o) = \Theta(\log P)$$

- Personalized Census
  - All processes agree on a different value for each process
  - Each process sends a contribution for each other proc.

$$T_{RS}(P) \geq G(P-1) + (L + 2o - G) \cdot \lceil \log_2 P \rceil = \Theta(P)$$

- Personalized Exchange
  - All processes send different values to all other processes

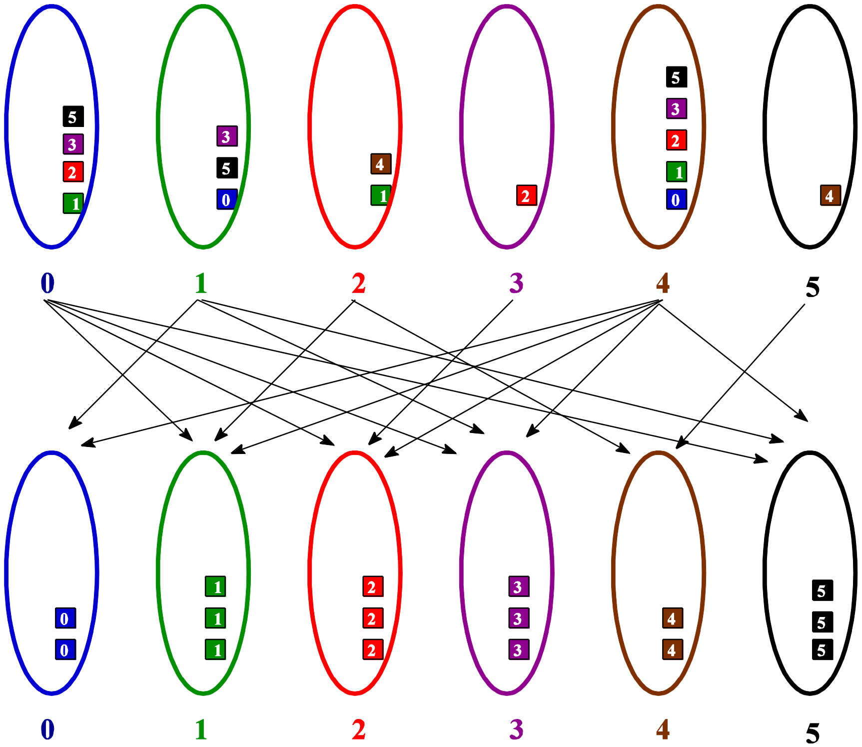
$$T_{PE}(P) \geq T_{RS}(P) = G(P-1) + (L + 2o - G) \cdot \lceil \log_2 P \rceil = \Theta(P)$$

Dynamic Sparse Data Exchange (DSDE)

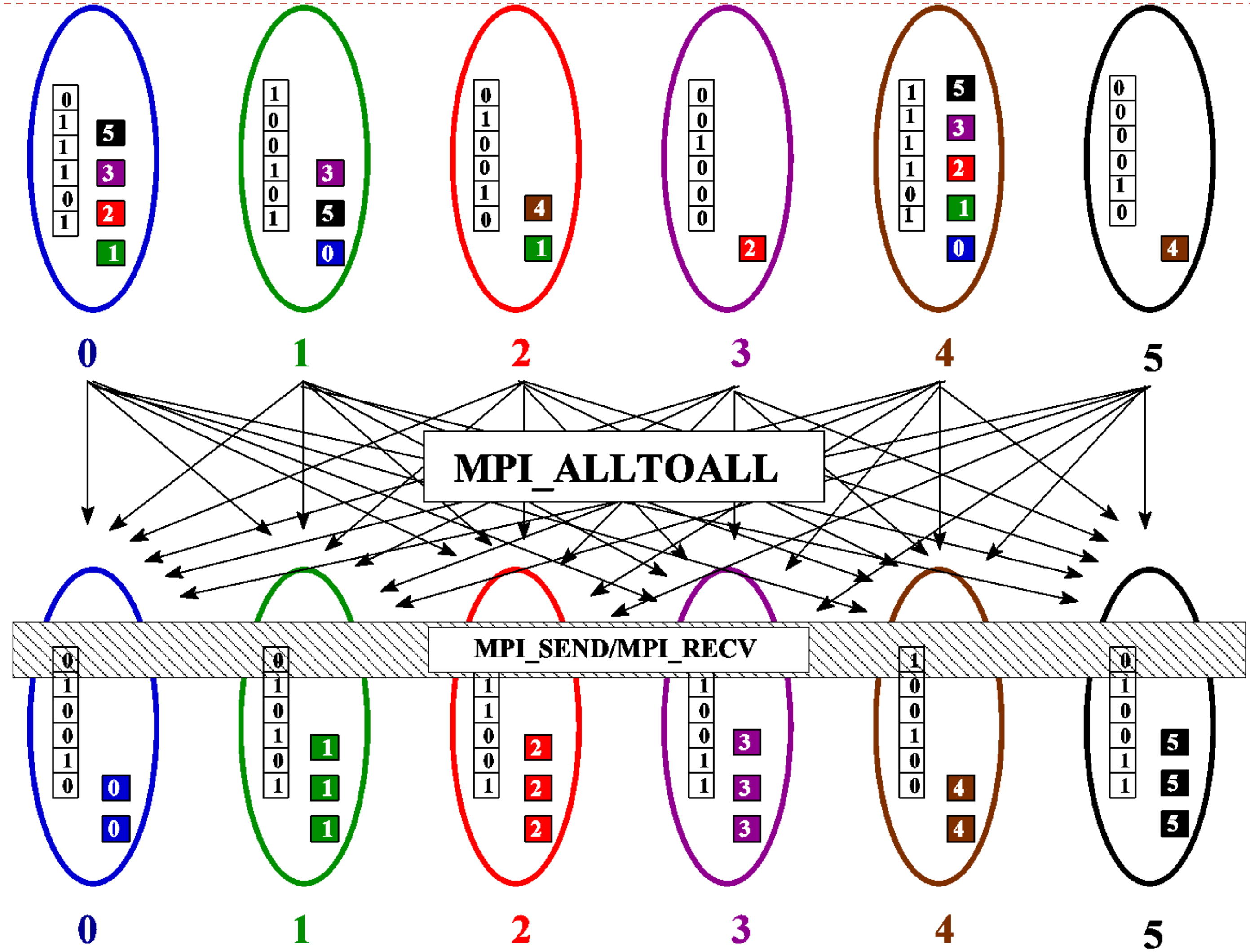
- Main Problem: metadata
  - Determine who wants to send how much data to me (I must post receive and reserve memory)

  OR:

  - Use MPI semantics:
    - Unknown sender
      - MPI_ANY_SOURCE
    - Unknown message size
      - MPI_PROBE
    - Reduces problem to counting the number of neighbors
    - Allow faster implementation!

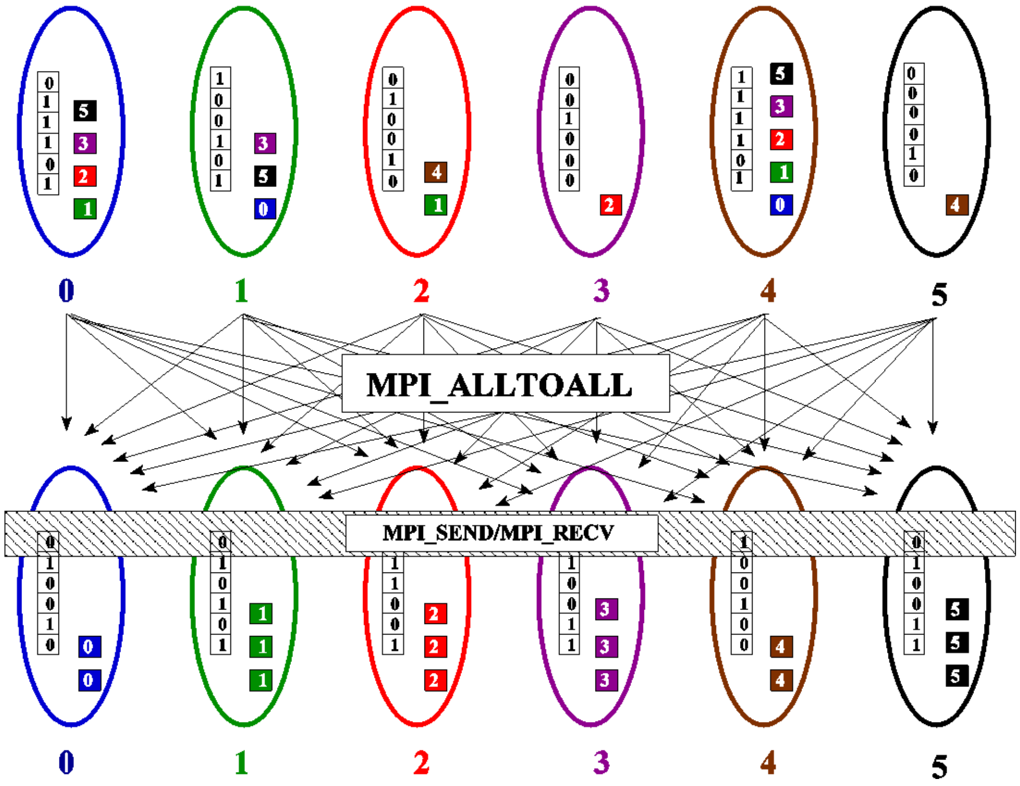


Protocol PEX (Personalized Exchange)

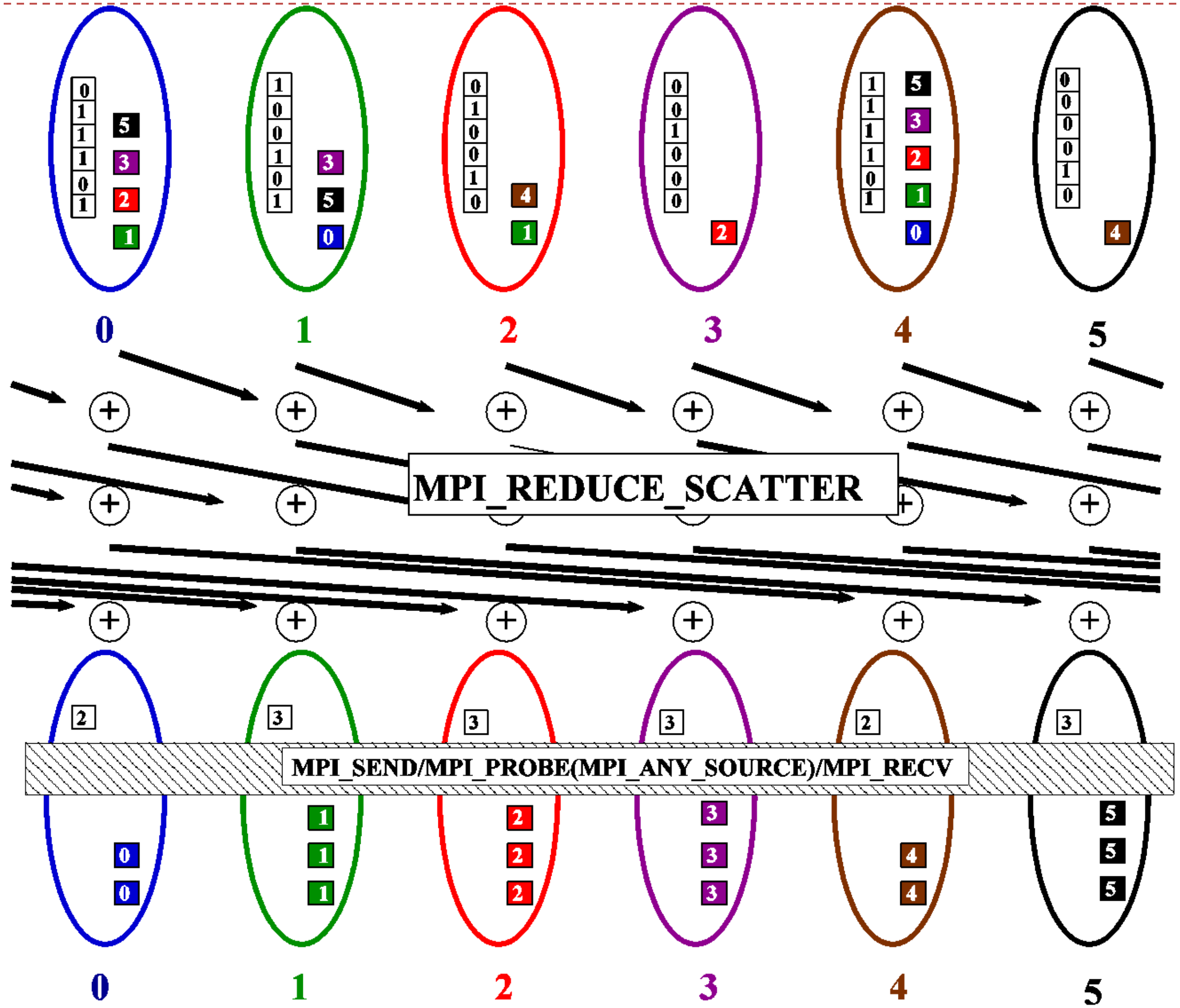


Protocol PEX (Personalized Exchange)

- Bases on Personalized Exchange ($\Theta(P)$)
  - Processes exchange metadata (sizes) about neighborhoods with all-to-all
  - Processes post receives afterwards
  - Most intuitive but least performance and scalability!

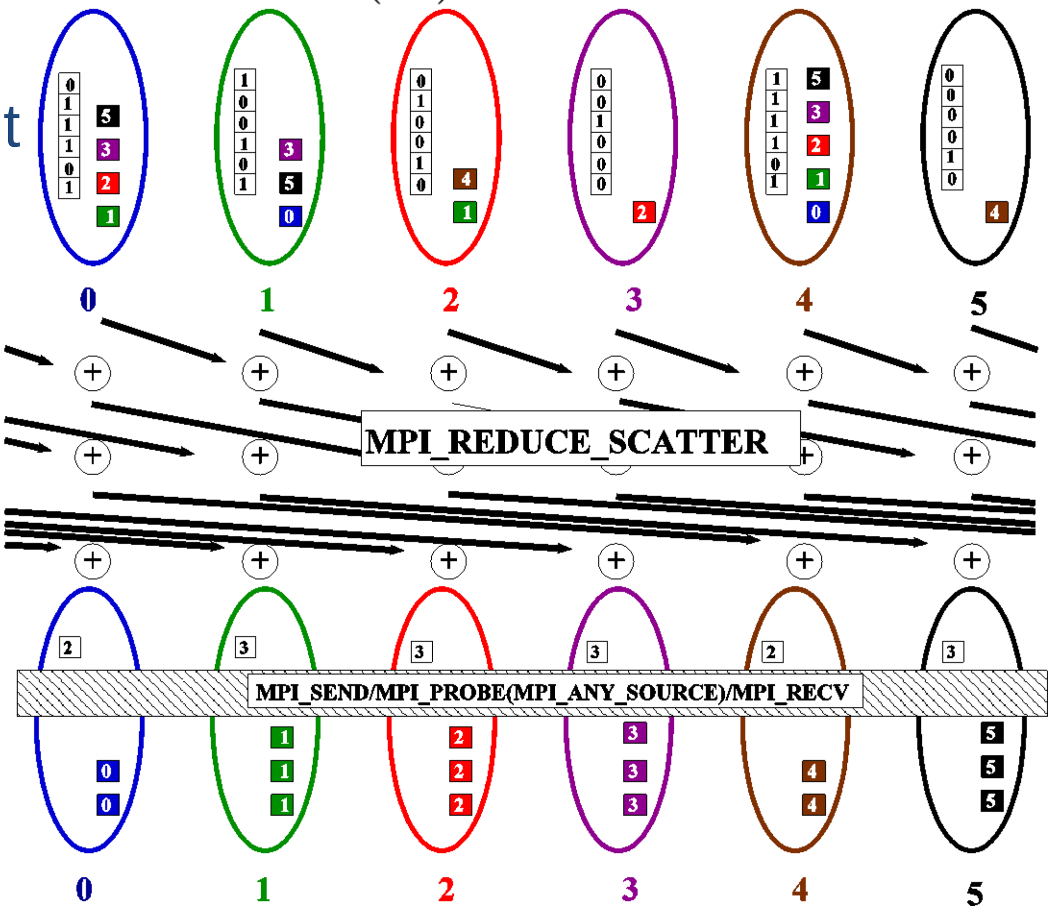


Protocol PCX (Personalized Census)

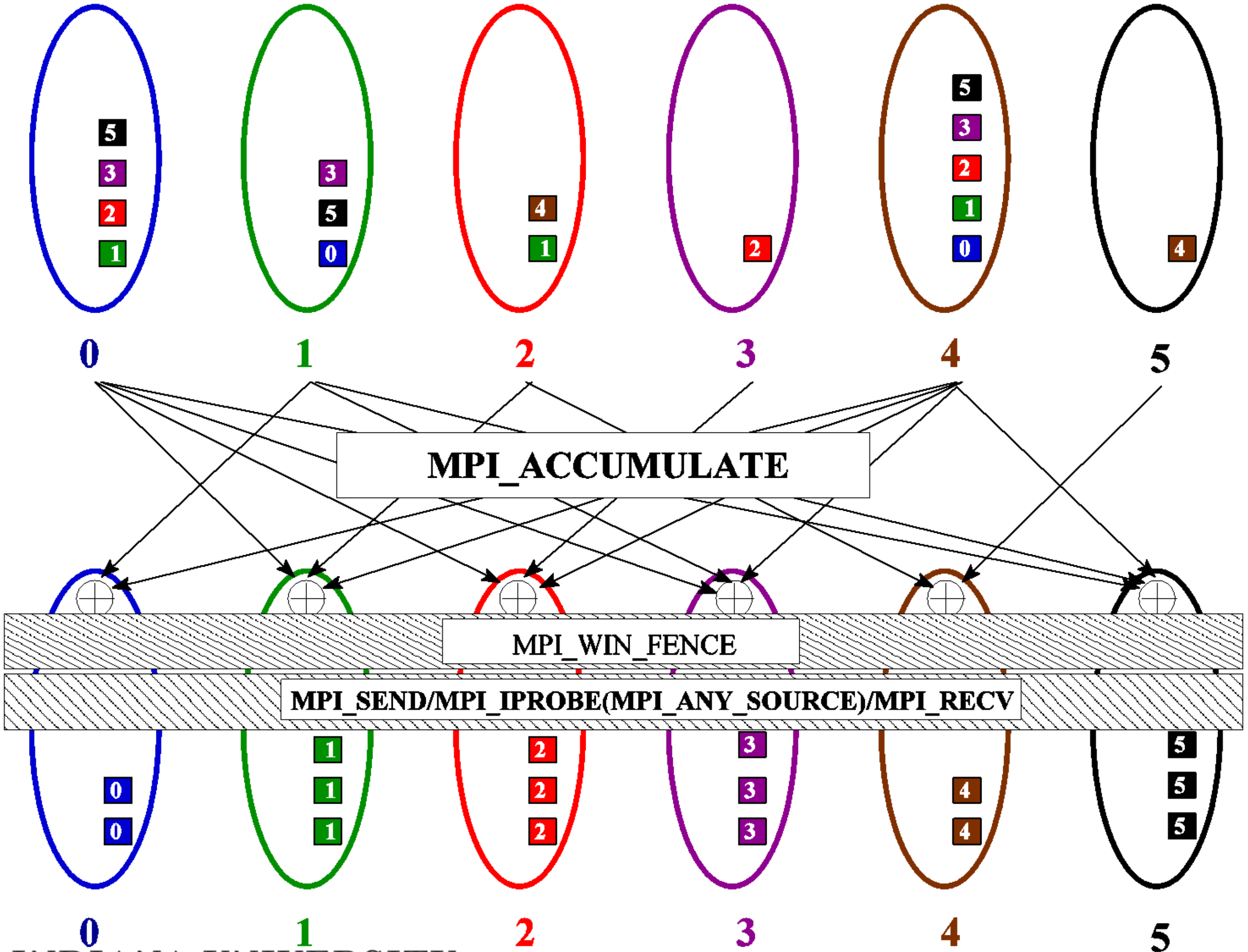


Protocol PCX (Personalized Census)

- Bases on Personalized Census ($\Theta(P)$)
  - Processes exchange metadata (counts) about neighborhoods with reduce_scatter
  - Receivers checks with wildcard MPI_IPROBE and receives messages
  - Better than PEX but non-deterministic!

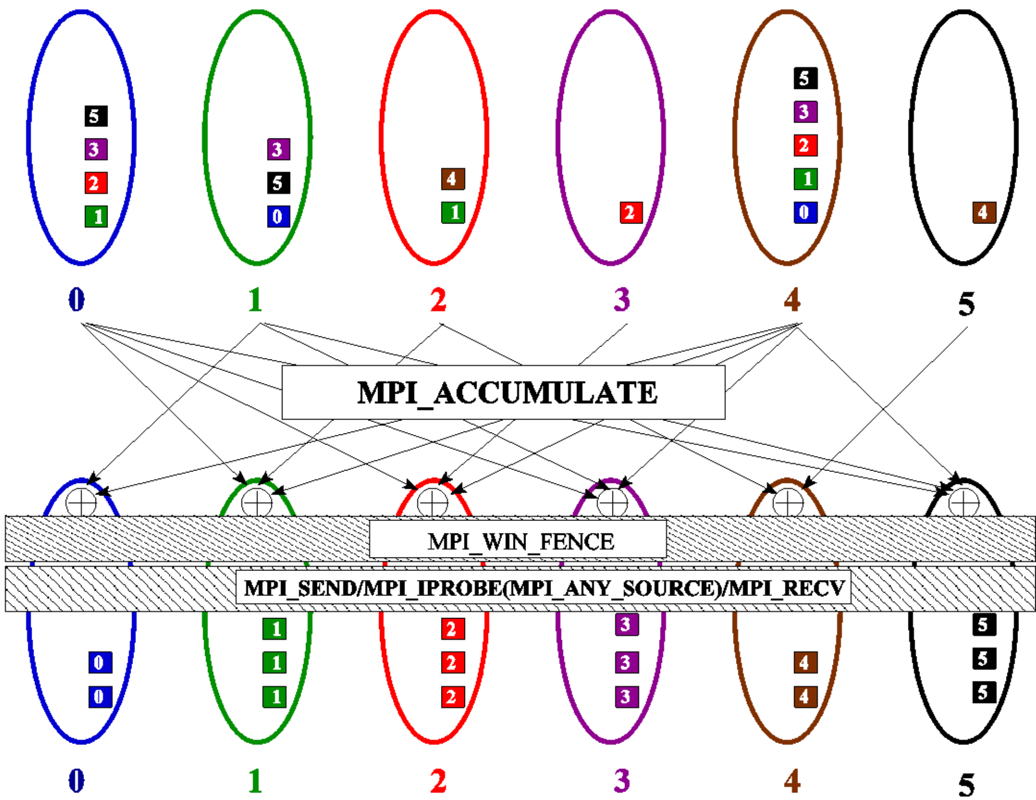


Protocol RSX (Remote Summation)



Protocol RSX (Remote Summation)

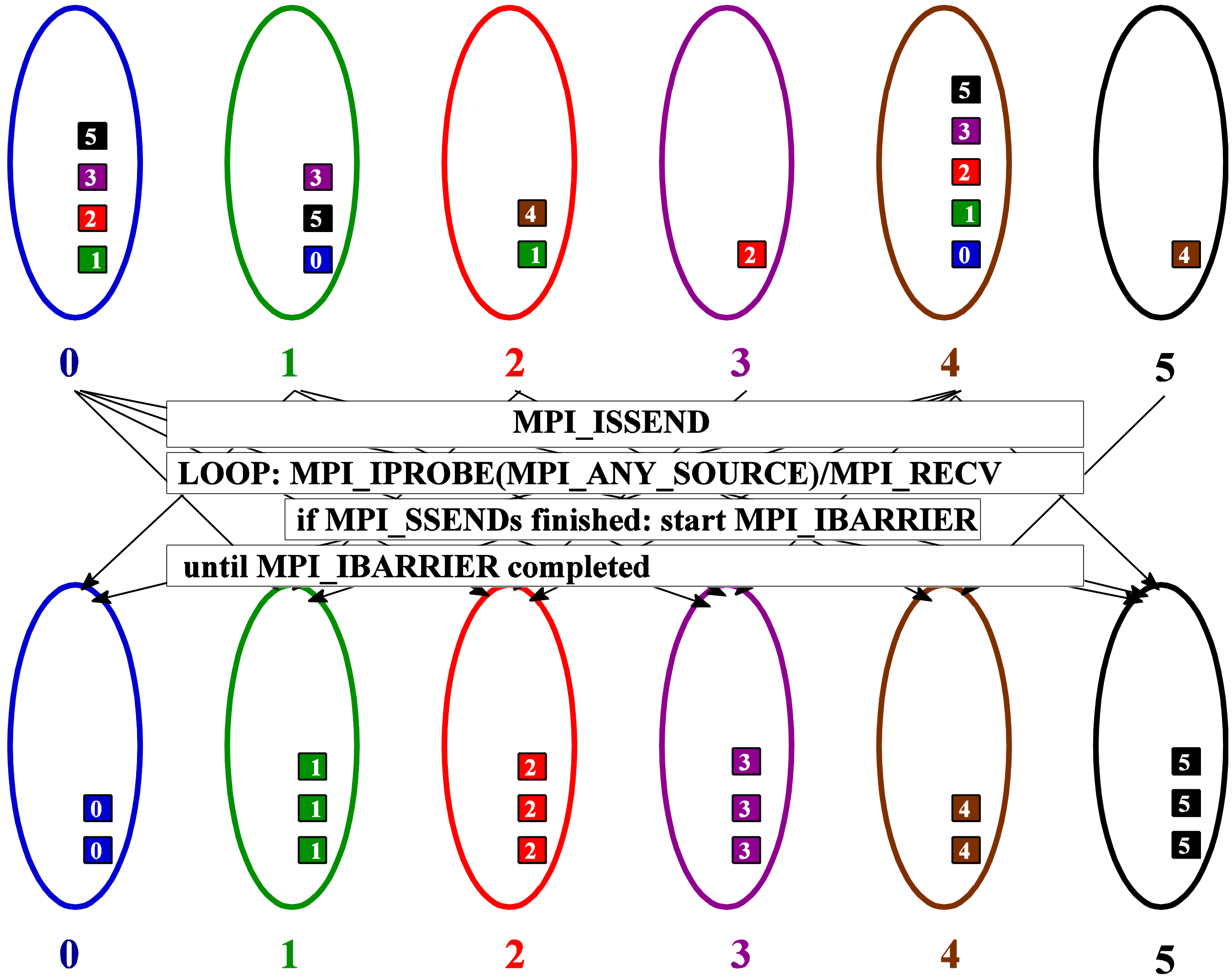
- Bases on Personalized Census (MPI_Win_fence): $\Theta(\log(P))$
  - Processes accumulate number of neighbors in receiver's memory
  - Receivers check with wildcard MPI_IPROBE and receives messages
  - Faster than PEX/PCX, non-deterministic and requires (good) RMA!



Nonblocking Collective Operations (NBC)

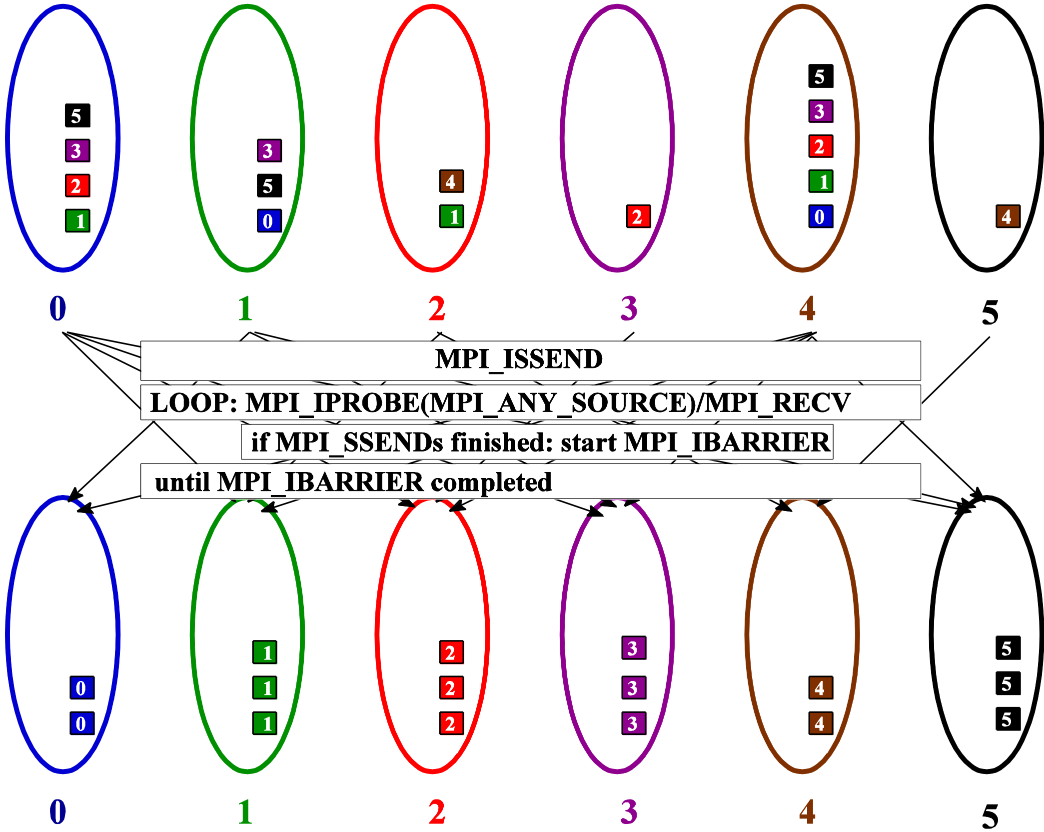
- It is as easy as it sounds: MPI_Ibarrier()
  - Decouple initiation and synchronization
    - Initiation does not synchronize
    - Completion must synchronize (in case of barrier)
  - Interesting semantic opportunities
    - Start synchronization epoch and continue
    - Possible to combine with other synchronization methods (p2p)
  - NBC accepted for MPI-3
    - Available as reference implementation (LibNBC)
      - LibNBC optimized for InfiniBand
    - Optimized on some architectures (BG/P, IB)

Protocol NBX (Nonblocking Consensus)



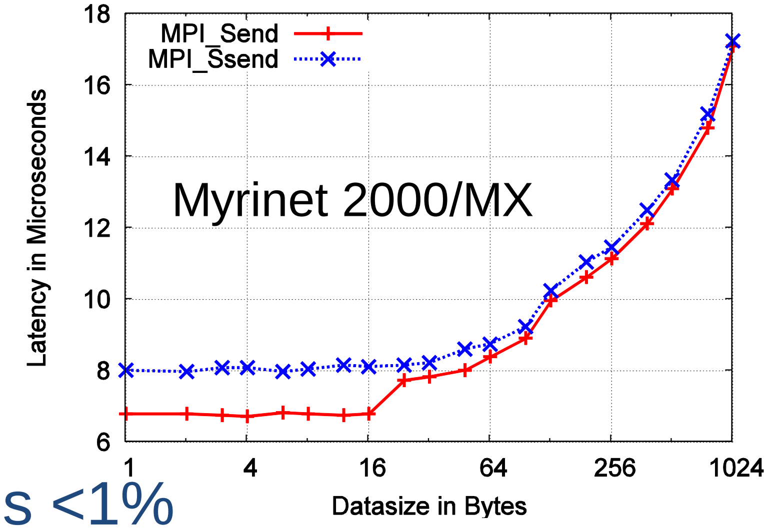
Protocol NBX (Nonblocking Consensus)

- Complexity - census (barrier): $\Theta(\log(P))$
  - Combines metadata with actual transmission
  - Point-to-point synchronization
  - Continue receiving until barrier completes
  - Processes start coll. synch. (barrier) when p2p phase ended
    - barrier = distributed marker!
  - Better than PEX, PCX, RSX!



Performance of Synchronous Send

- Worst-case: 2*L
  - Bad for small messages
  - Vanishes for large messages
- Benchmark
  - Slowdown for 1-byte messages
  - Threshold = size when overhead is <1%

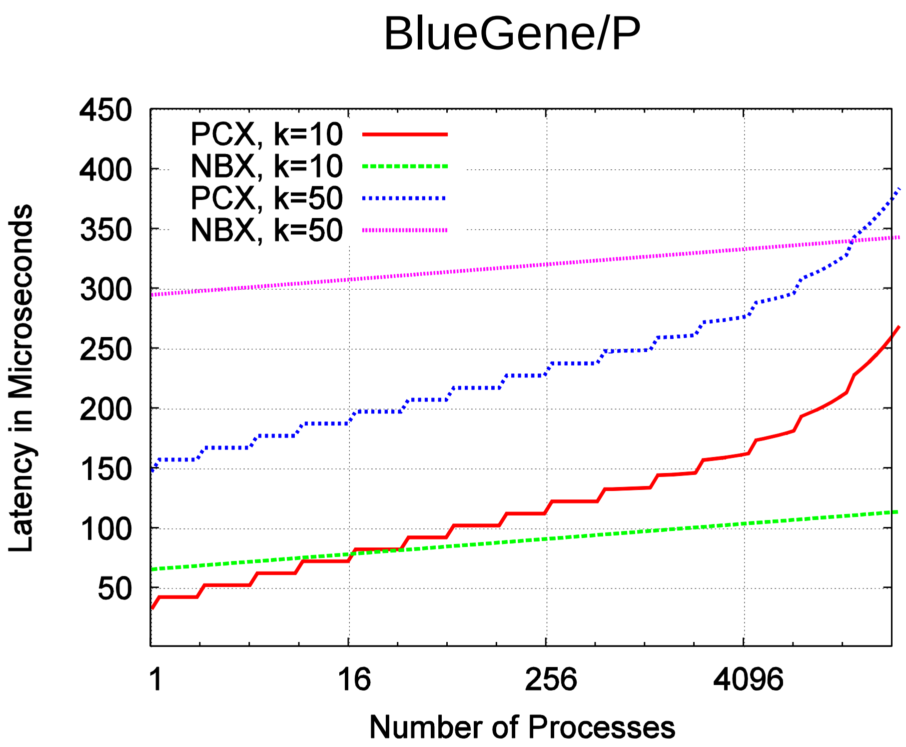


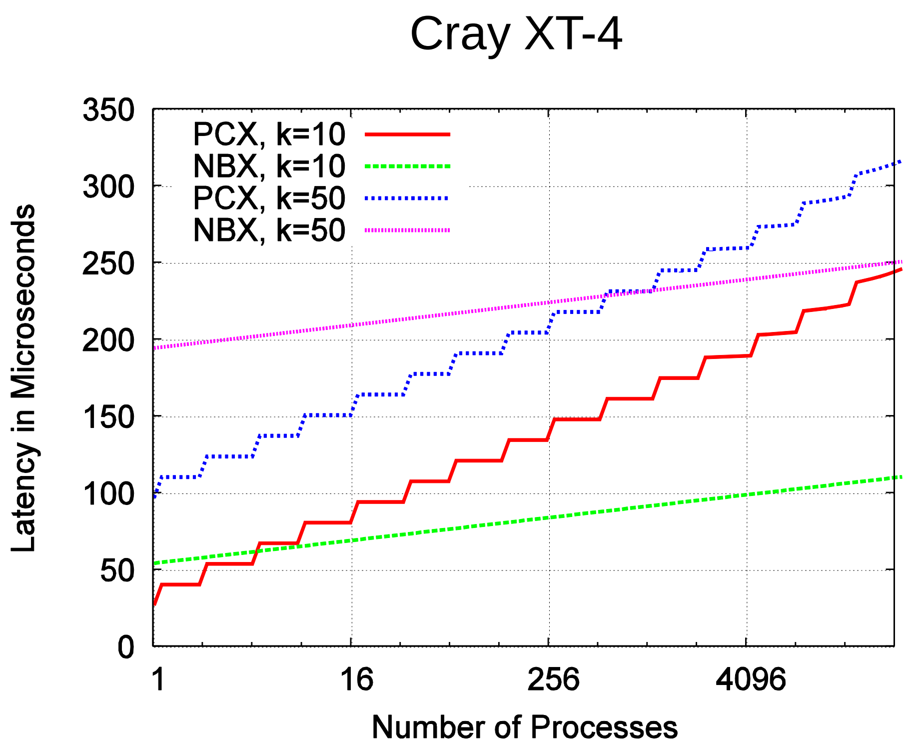
System	L (synch)	Slowdown	Threshold
Intrepid (BG/P)	5.04 us	1.17	12 kiB
Jaguar (XT-4)	25.40 us	2.57	132 kiB
Big Red (Myrinet)	8.02 us	1.13	1.5 kiB

- Very good results for BG/P and Myrinet!

LogP Comparison – PCX vs. NBX

- k=number of neighbors, assuming L(synch) = 2*L



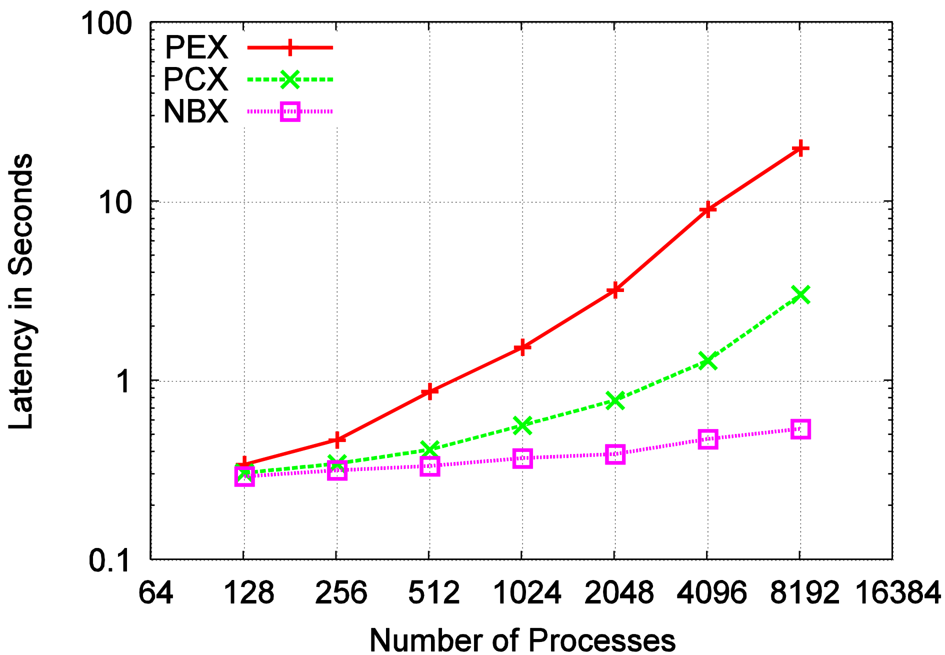


- NBX faster for few neighbors and large scale!

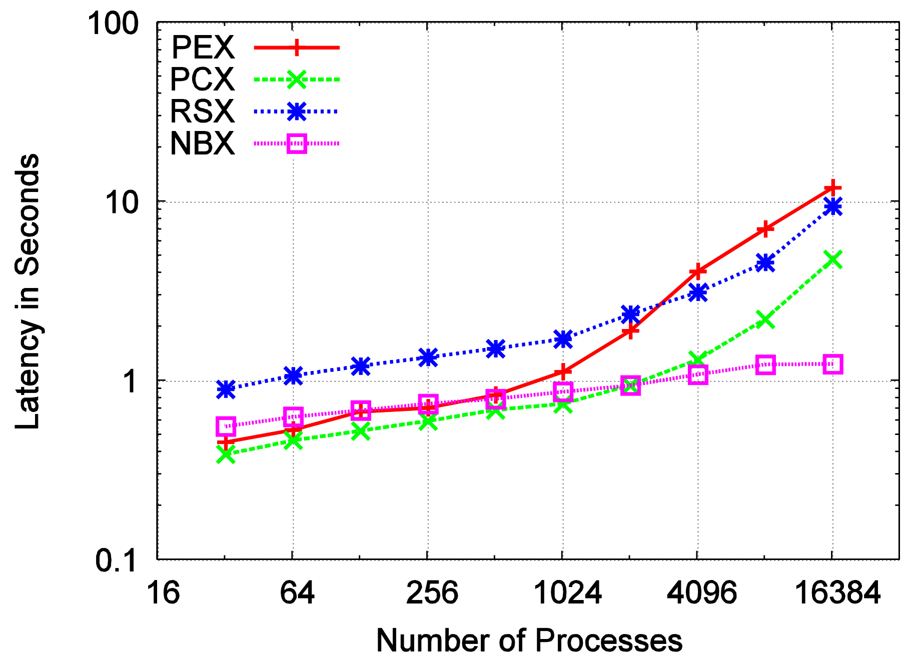
Microbenchmark

- Each process sends to 6 random neighbors

BlueGene/P



Cray XT-4

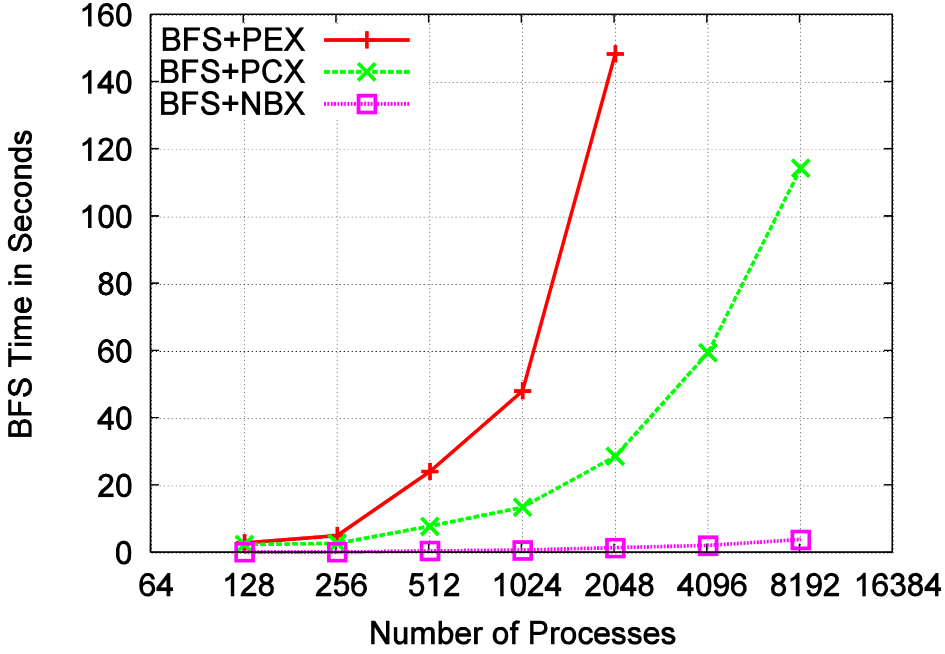


- Significant improvements at large scale!

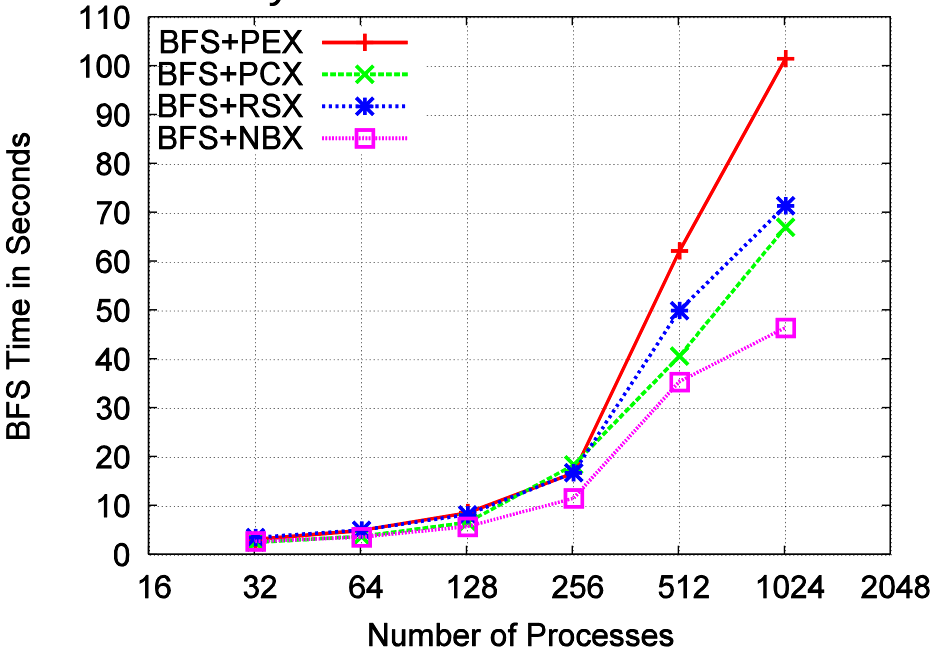
Parallel Breadth First Search

- On a clustered Erdős-Rényi graph, weak scaling
  - 6.75 million edges per node (filled 1 GiB)

BlueGene/P – with HW barrier!



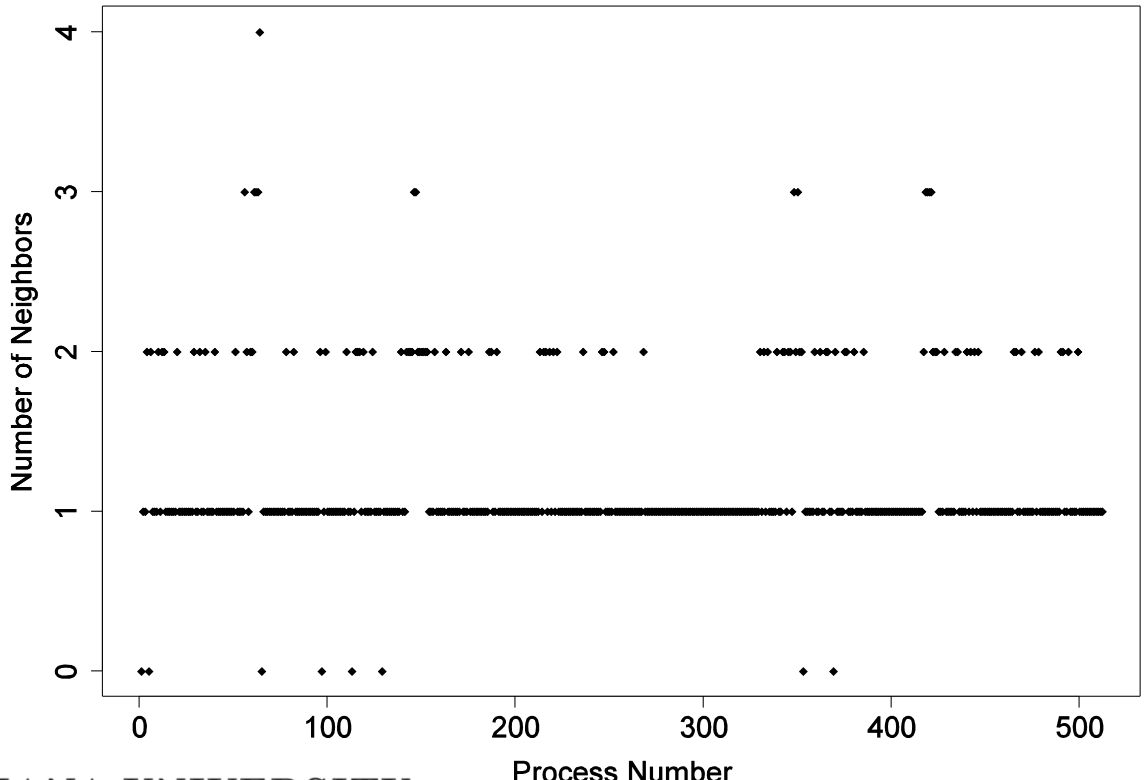
Myrinet 2000 with LibNBC



- HW barrier support is significant at large scale!

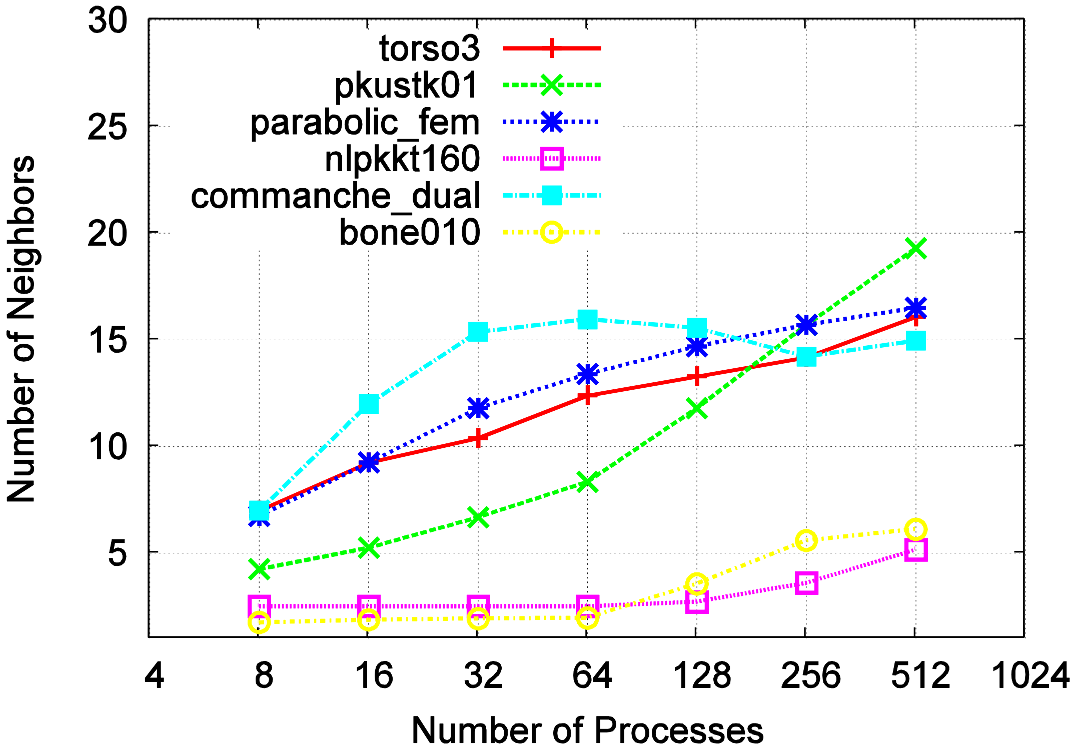
Are our assumptions for k realistic?

- Check with two applications:
  - Parallel N-body (Barnes&Hut) (512 processes)
  - Number of neighbors in rebalancing ORB step:



Are our assumptions for k realistic?

- Sparse linear algebra (CFD, FEM, …)
  - Used simple block-distribution of UFL matrices
  - Graph partitioning techniques would reduce k further!



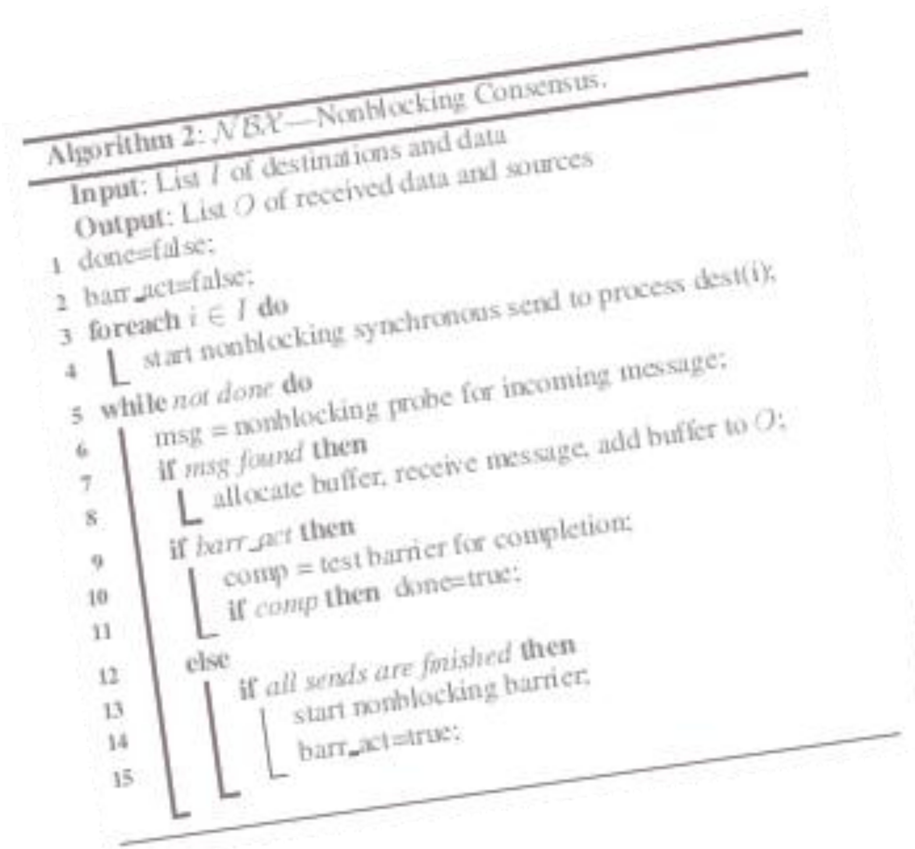
Conclusions and Future Work

- SDSE problem is important
  - Metadata exchange dominates at large scale!
- We discussed four algorithms and their complexity
  - NBX is fastest for large machines and small k
  - RCX is probably most "convenient"
- Hardware support for NBC crucial at large scale!
- Synchronous sends can be performance critical!

- We plan to work on an self-tuning adaptive library
  - Automatic algorithm selection
- Look into large-scale applications

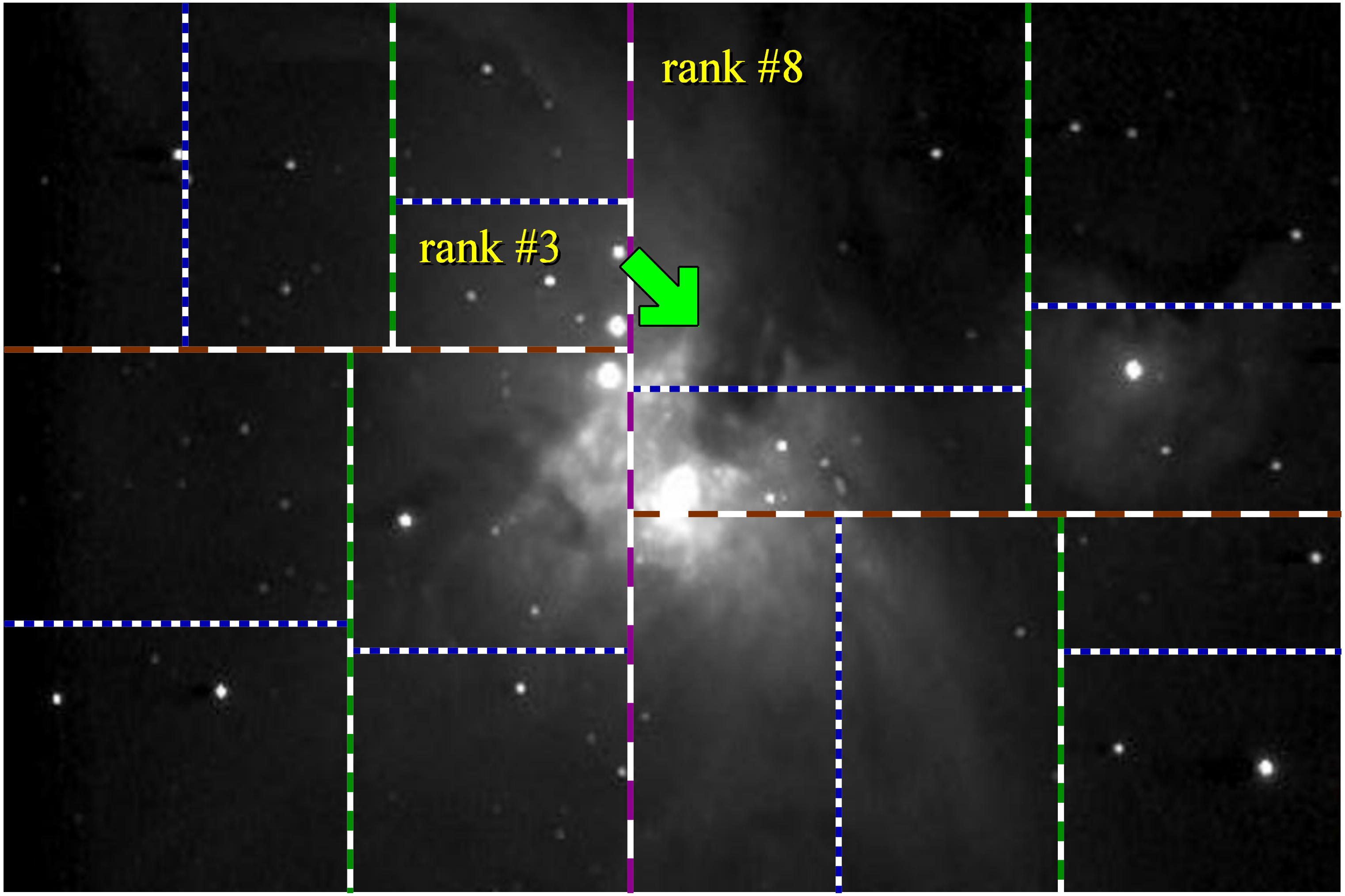
Thank you for your attention!

Questions?



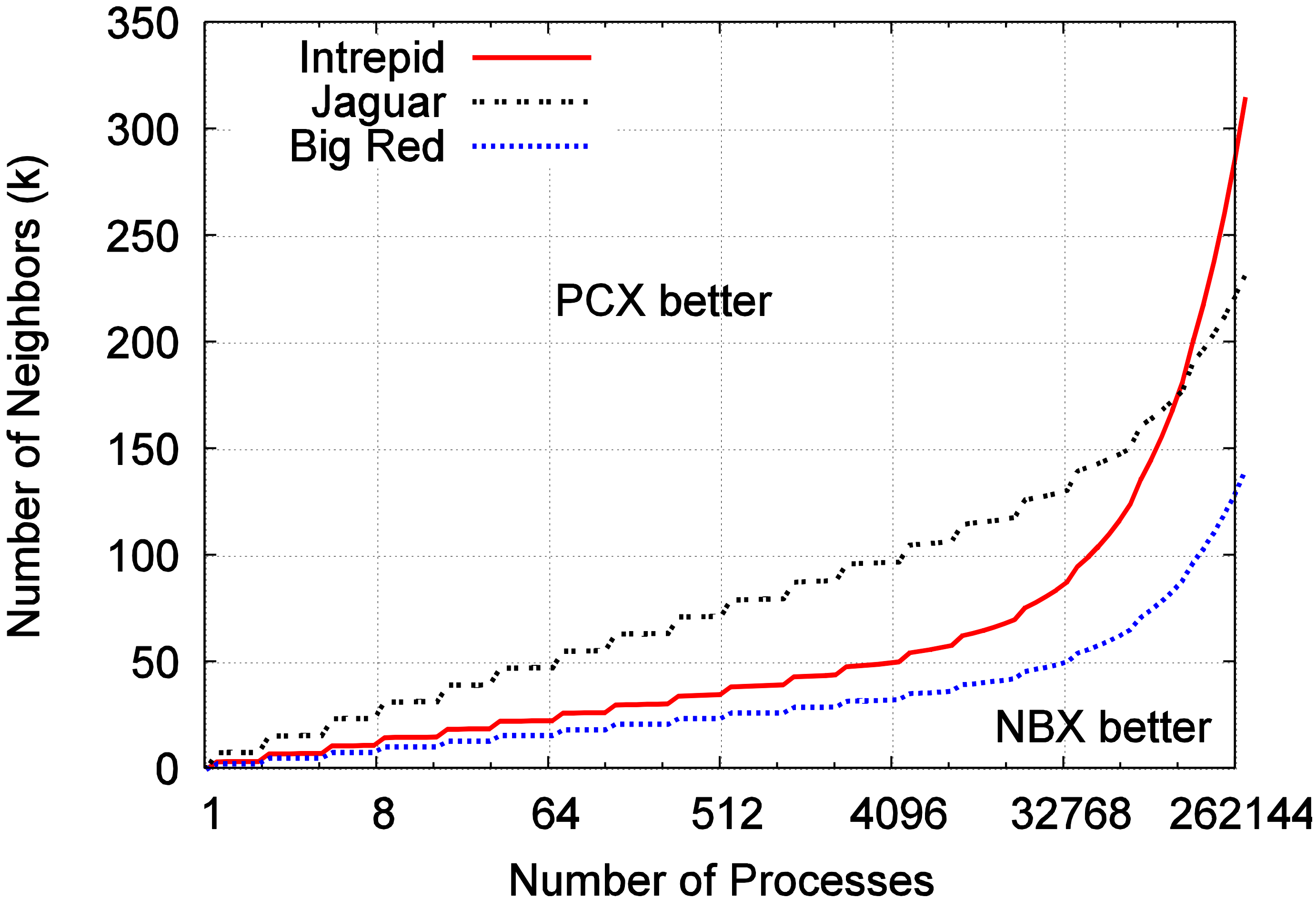
```
Algorithm 2: NBX—Nonblocking Consensus.
Input: List I of destinations and data
Output: List O of received data and sources
done=false;
barr_act=false;
foreach i ∈ I do
    start nonblocking synchronous send to process dest(i);
while not done do
    msg = nonblocking probe for incoming message;
    if msg found then
        allocate buffer, receive message, add buffer to O;
    if barr_act then
        comp = test barrier for completion;
        if comp then done=true;
    else
        if all sends are finished then
            start nonblocking barrier;
            barr_act=true;
```

Orthogonal Recursive Bisection



Influence of the Number of Neighbors

- "sparsity"-factor is important for algorithm choice!



Quick Terms and Conventions

- We use standard LogGP terms
  - L – maximum latency between any two processes
  - o – CPU send/recv overhead
  - g – time to wait between network injections
  - G – time to transmit a single byte
  - P – number of processes in the parallel job
- One single byte messages from A to B:
  - costs o on A and arrives after 2o+L on B
- We assume that o>g for simplicity
- All parallel processes start at t=0