Impact of Noise on Parallel Programs

- Noise interrupts a core from working on a parallel program
- Major problem when working at large scales, e.g., peta- and exa-flop
- Questions:
  - What is the impact of noise?
  - How can we prevent noise?

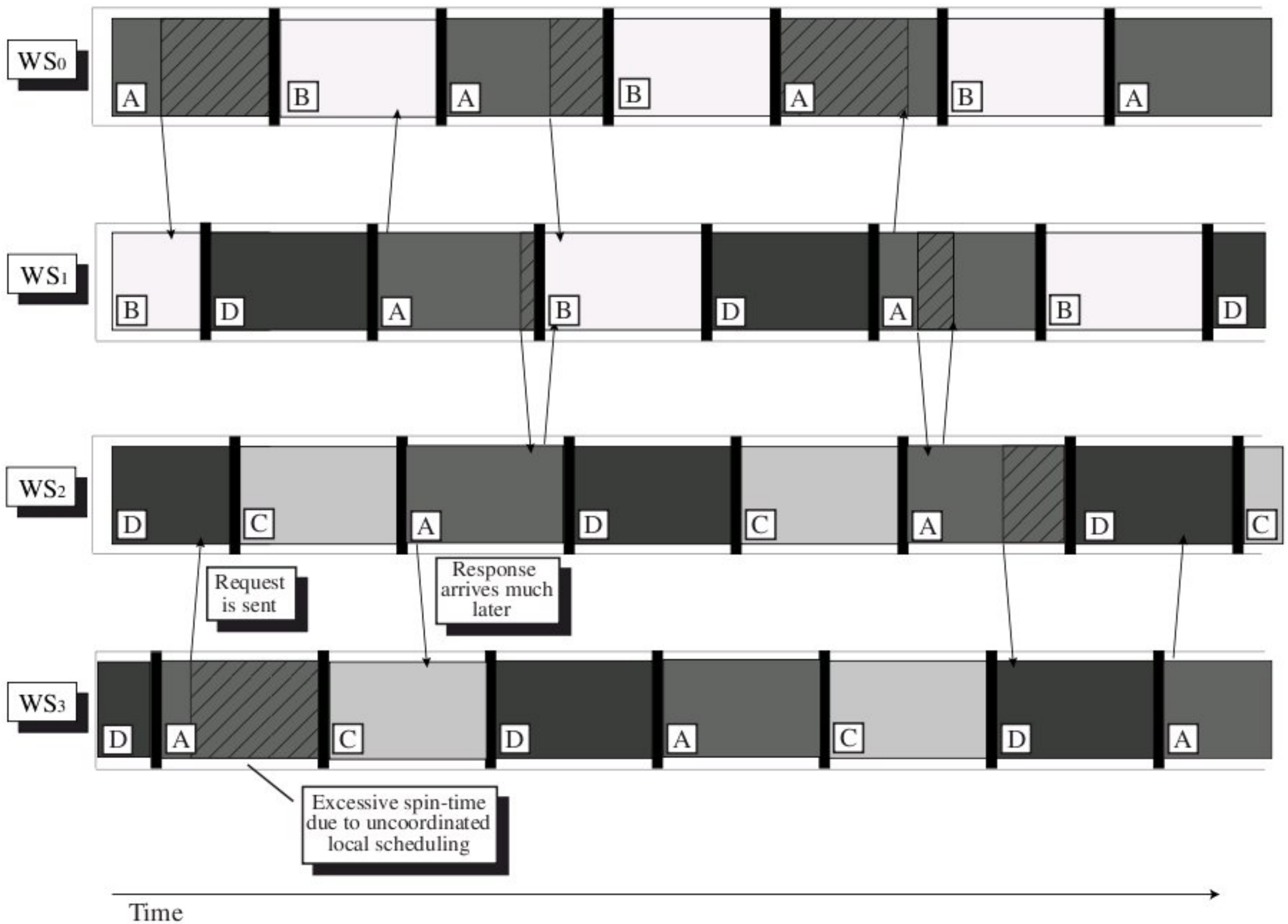
Why does noise occur?

- Some reasons:
  - Another application can be executing
  - System activities may need to occur

Noise due to other applications

- This occurs, for example, when:
  - You and your classmate both try to execute a program on the 930 cluster on the same set of nodes
    - The node is multiprogrammed
  - Generally we'd expect each program to take X times longer, if there are X users running jobs
    - But can be worse because of *uncoordinated nodes*

Uncoordinated Scheduling

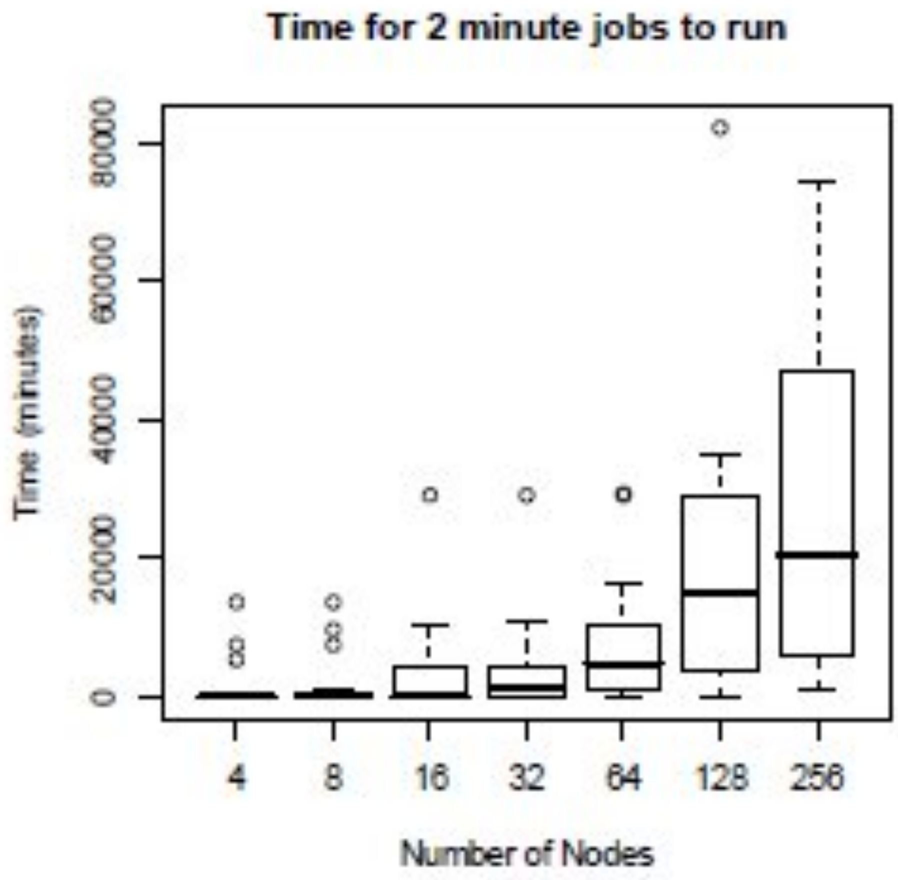


Courtesy of Andrea Arpaci-Dusseau's Ph.D thesis (1999)

Noise due to other applications

- One "solution" is: define it away
  - On large-scale systems such as national lab machines, the nodes are *space shared*
    - Means that a group of nodes is assigned solely to a user for a given amount of time
    - Scheduling these groups is done based on priority, number requested, and time requested
  - Note that space sharing has overhead---namely, potentially low utilization when application is inefficient
    - Plus wait times if the machine is busy

Batch queue waiting time example



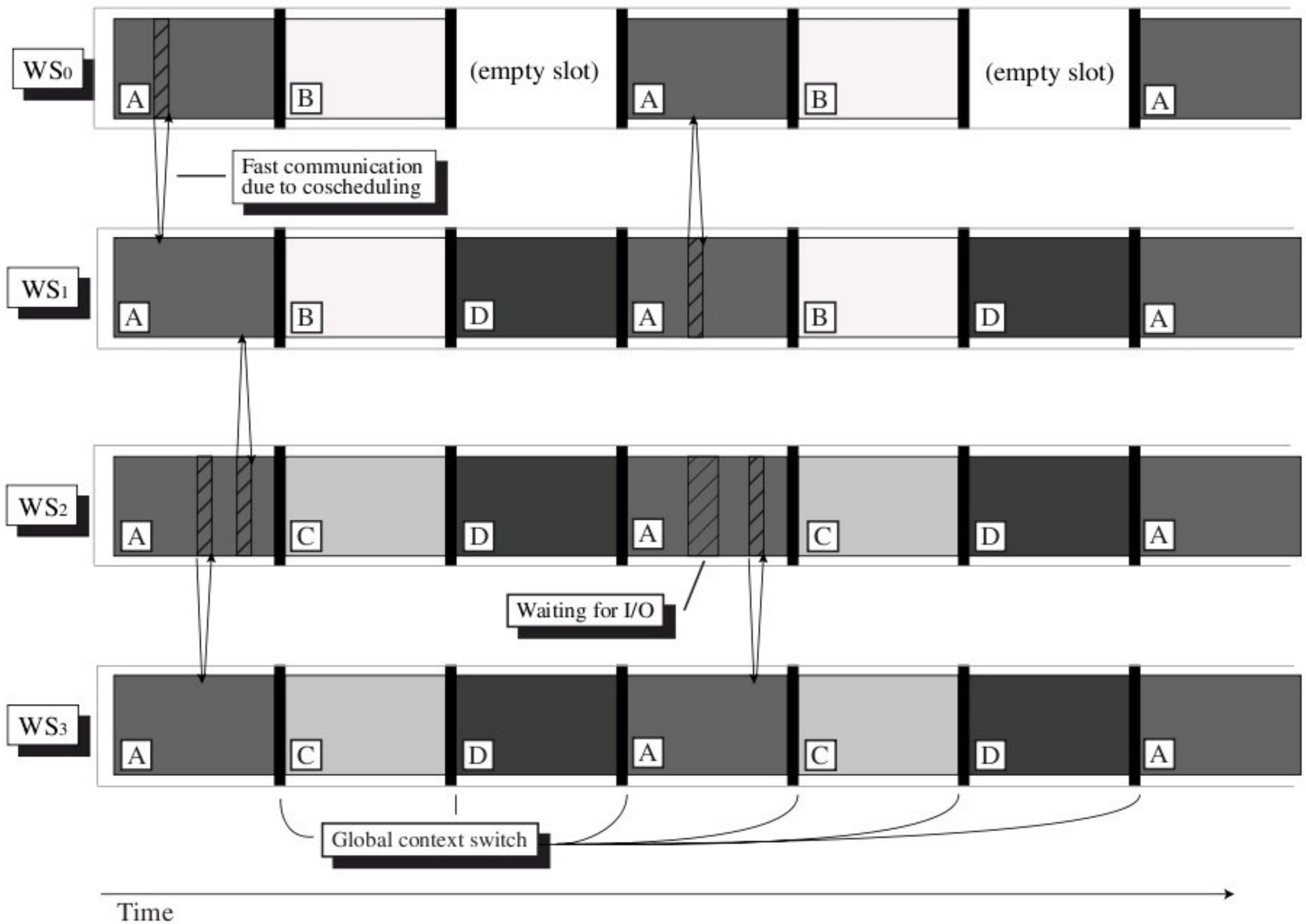
Significant wait time increase at large node counts

Taken from "Thunder" cluster at LLNL in 2006

Noise due to other applications

- Another solution is gang scheduling
  - Basic idea: a time slice is global, across the entire machine
    - Do not pre-empt for I/O
    - Guarantees that there will not be overhead due to, say, another job running on at least one node
  - Disadvantages:
    - Complex
    - Cannot utilize blocking processor if within a time slice
    - May force nodes to be completely idle (!!)

Gang Scheduling



Taken from Andrea Arpaci-Dusseau's Ph.D thesis (1999)

Noise due to other applications

- Final solution is co-scheduling (Arpaci-Dusseau ’96)
  - Basic idea: local scheduler per machine, but scheduling based on communication
    - When a processor sends a message, do not context switch for a threshold time (intuition: reply is coming shortly)
    - If beyond that threshold, then context switch
  - Disadvantages:
    - No guarantees
    - Can “game” the scheduler by communicating frequently

System Noise

- Caused by OS daemons performing system tasks
- These tasks cannot always be put off
  - Example: if the network daemon fails to take packets out of the network buffer, deadlock can occur
- Even if nodes are space shared, OS daemons are likely to be uncoordinated

System Noise

- May become a huge problem at large scales
- Analogous to the resilience problem: the more nodes you have, the worse the problem is
- Entire HPC OS kernels have been built to lessen the effects of noise
  - "Lightweight kernels"
    - Simple scheduler and memory management
    - Minimum number of daemons
- Also have been efforts to "sync up" daemons across nodes…but this is tricky

System Noise

- Might seem like a small problem in that:
  - If nodes have similar noise profiles, can't we just assume that it's extra load and is "load balanced"
    - **Not that simple because there are dependencies between nodes---can be complex**

Time Scale of Events

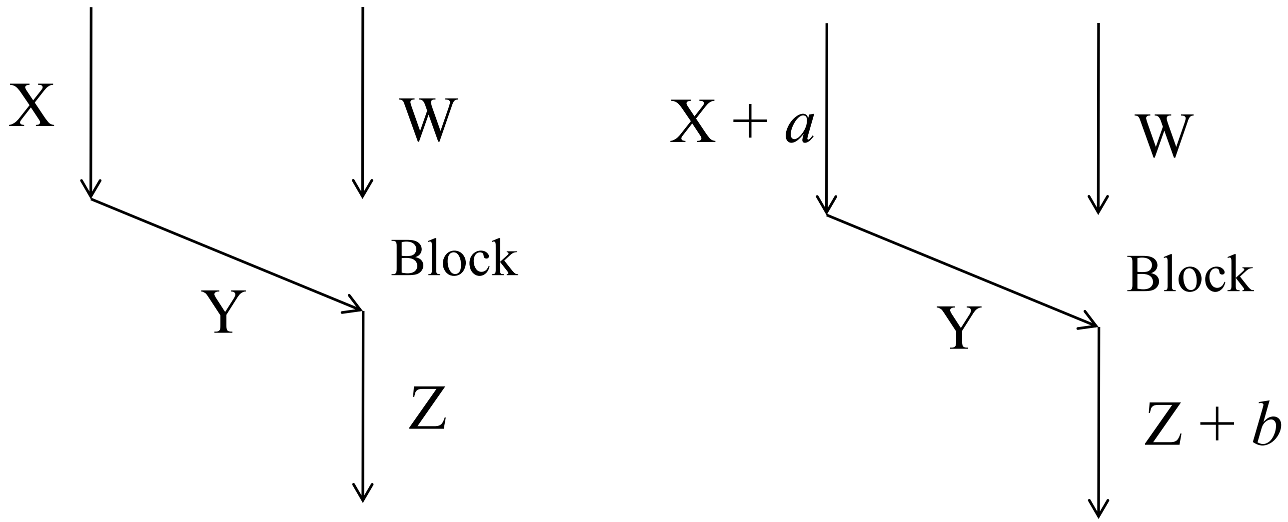
I.e., how long is a process interrupted?

- Cache miss, TLB miss: ~100 ns
- **Hardware interrupt**, PTE miss, **timer update**: ~1 us
- Page fault, swap in, **pre-emption**: ~10 ms

Noise: items from the above list that user cannot control even with careful programming (**boldfaced**)

(Events and times from Beckman 2008)

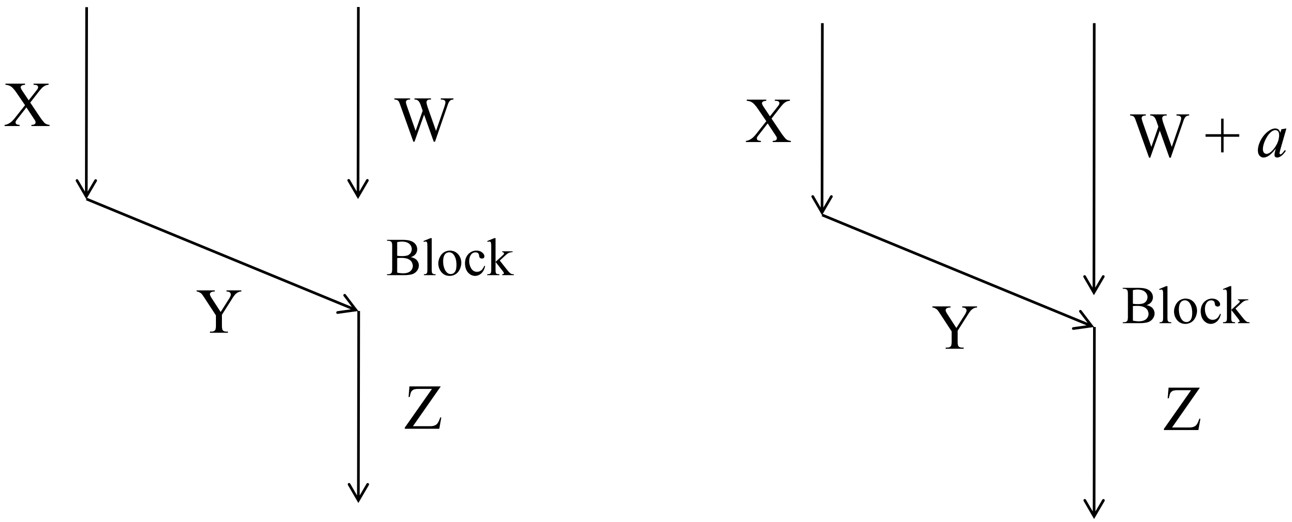
Noise Propogation



a and *b* are noise and are added to execution time

Total time with noise is X+a+Y+Z+b

Noise Absorption



a is noise yet it's absorbed---does not affect execution time

Total time with noise is X+Y+Z

Influential Paper: Petrini et al., "The Case of the Missing Supercomputer Performance" (Supercomputing 2003, best paper)

- Ran a large-scale application called SAGE on 8192 processors (a lot in 2003)
- Improved performance **by a factor of 2** by removing selected daemons

Note: all following pictures from the Petrini 2003 paper

SAGE performance (weak scaling)

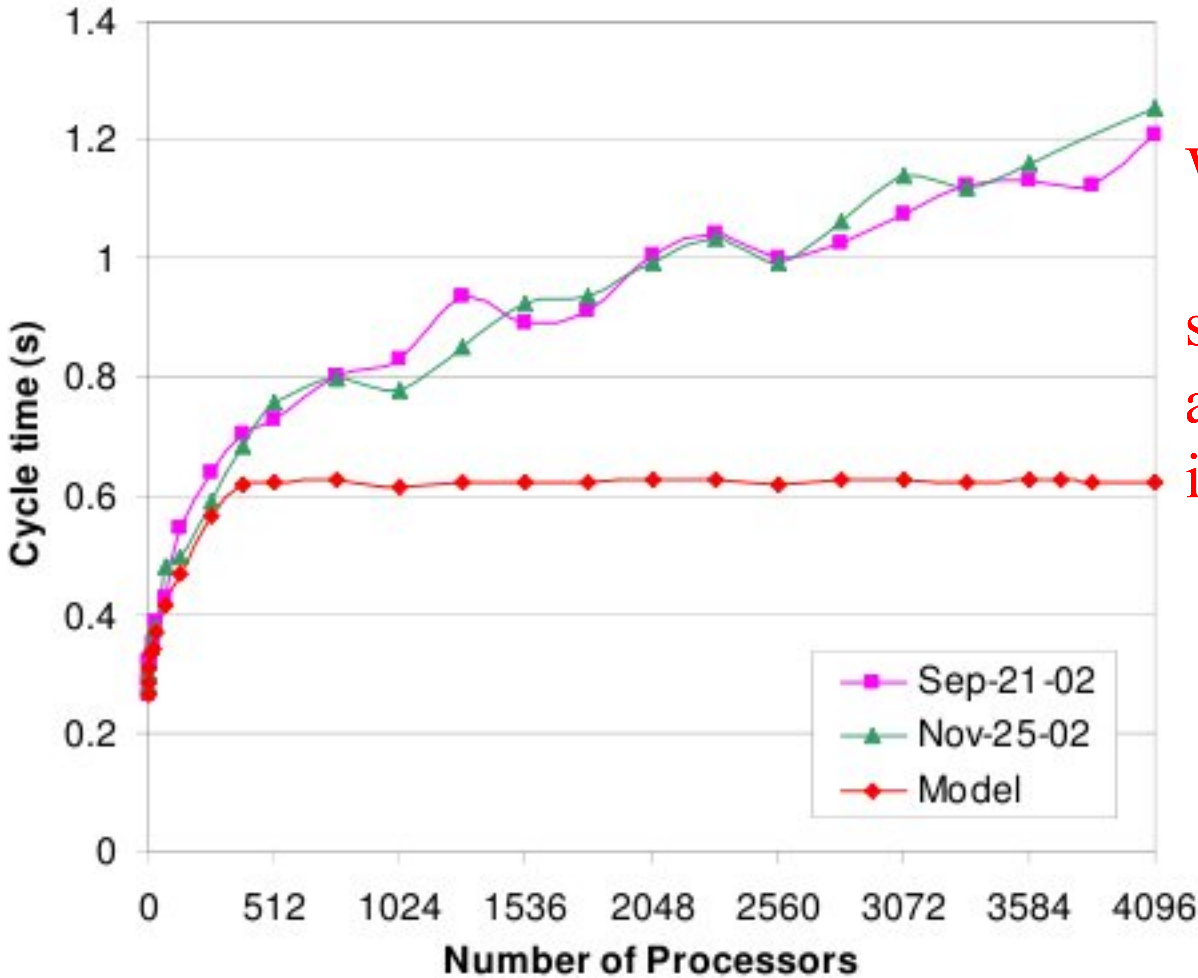
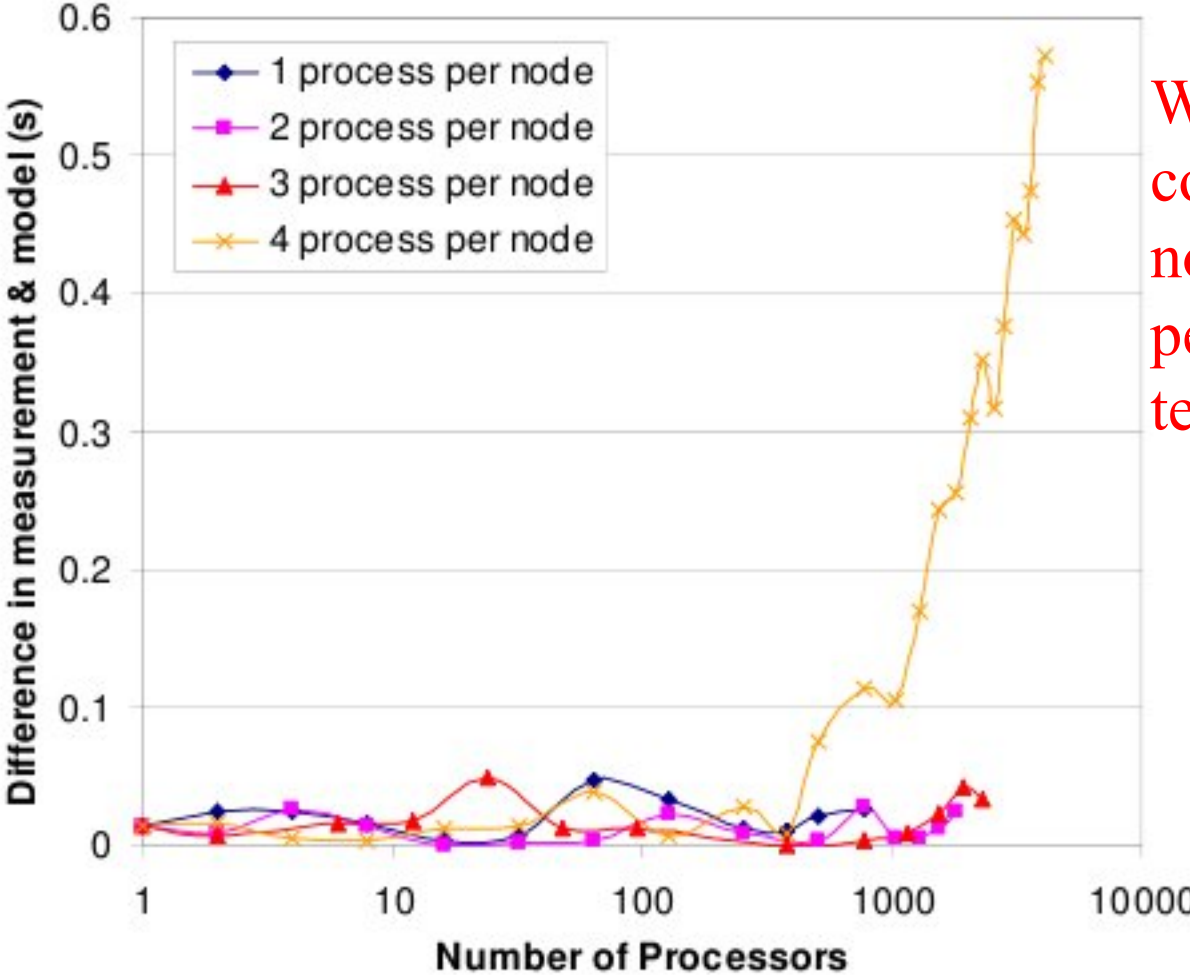


Figure 1: Expected and measured SAGE performance

Weak scaling: time should stay constant as #processors increases

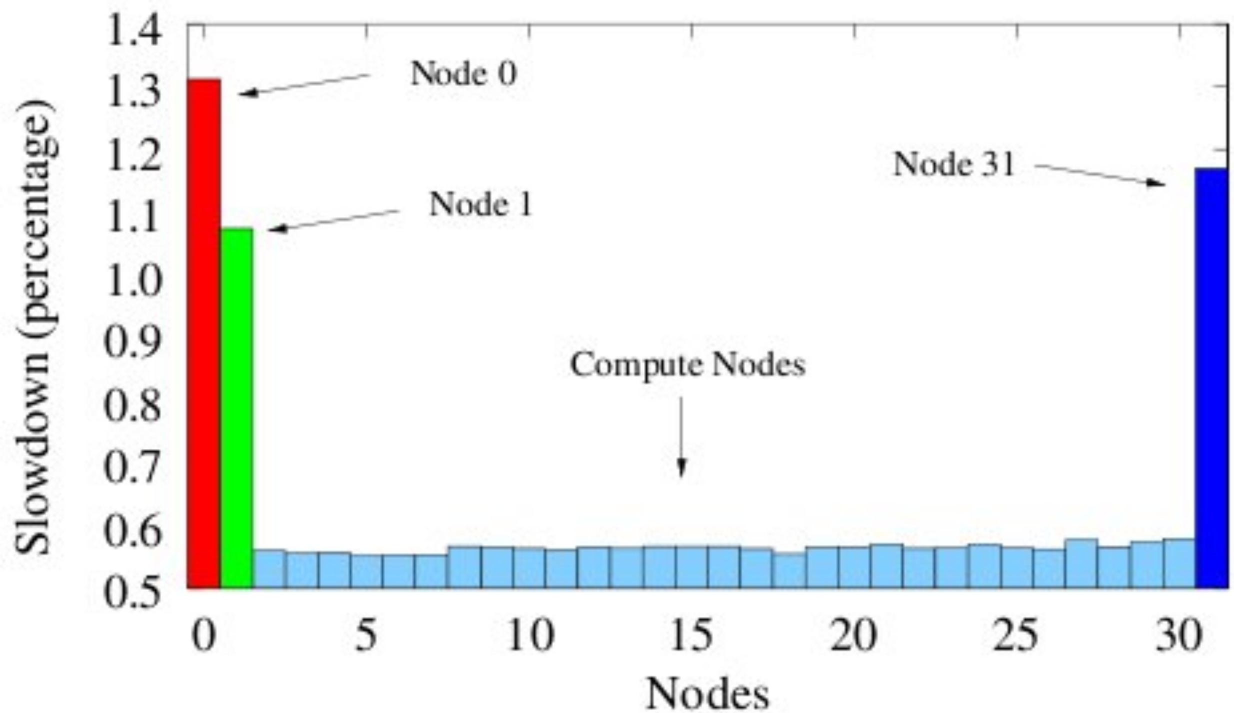
SAGE performance



Want to use all cores on the node, but performance is terrible

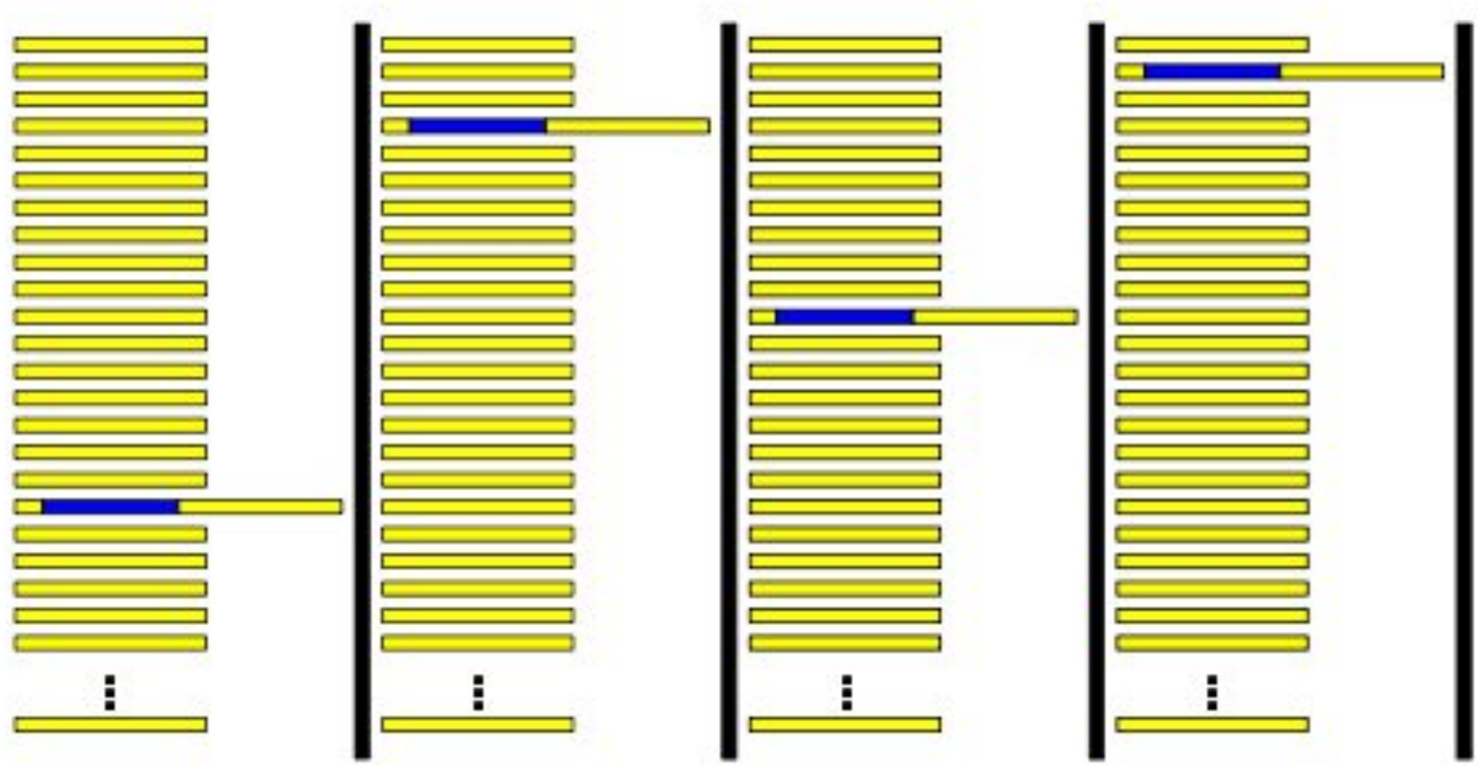
Note: improving Allreduce performance (51% of total time) did not help

Noise Pattern on ASCI Q (groups of 32 nodes)



Daemons on: node 0 (filesystem), node 1 (cluster manager), node 31 (resource manager)

Uncoordinated Noise

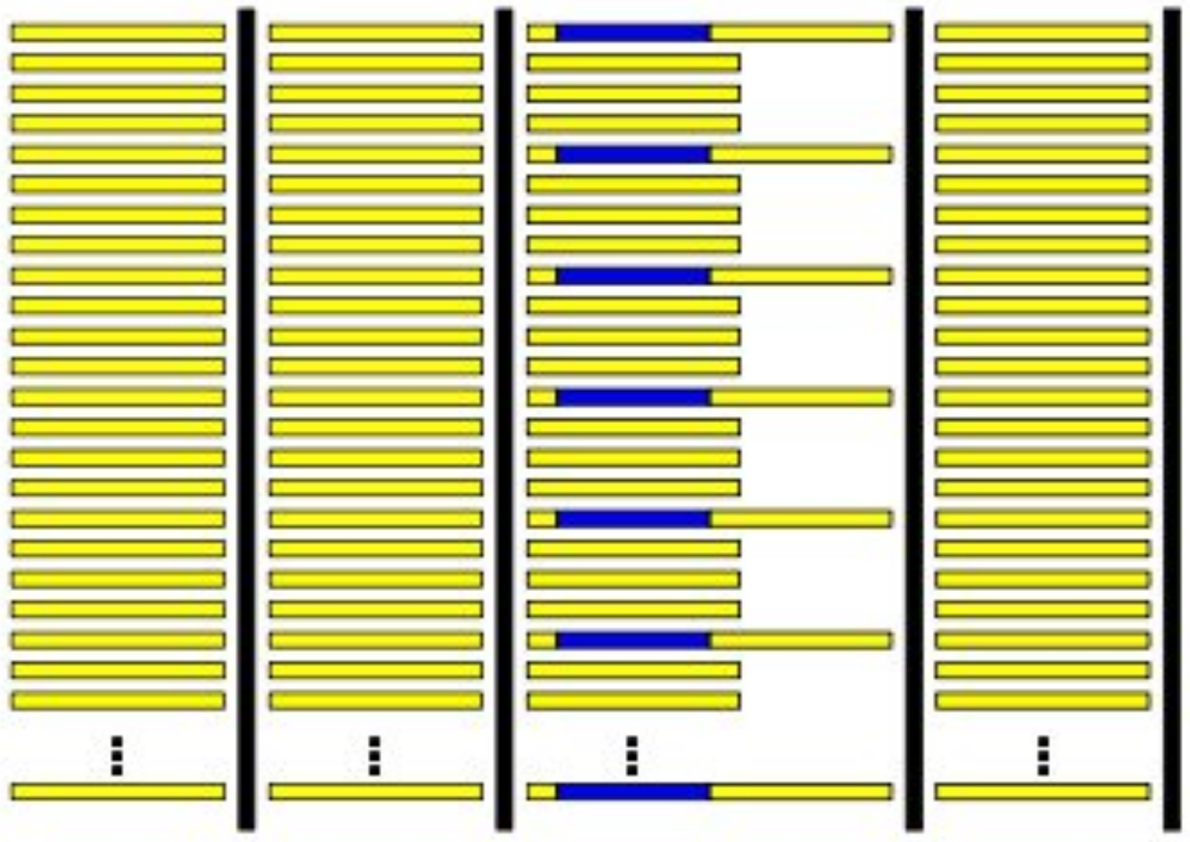


Black bars are barriers; yellow bars are computation, blue bars are noise

Takeaway message:

without coordination, fine-grain synchronization can be devastating

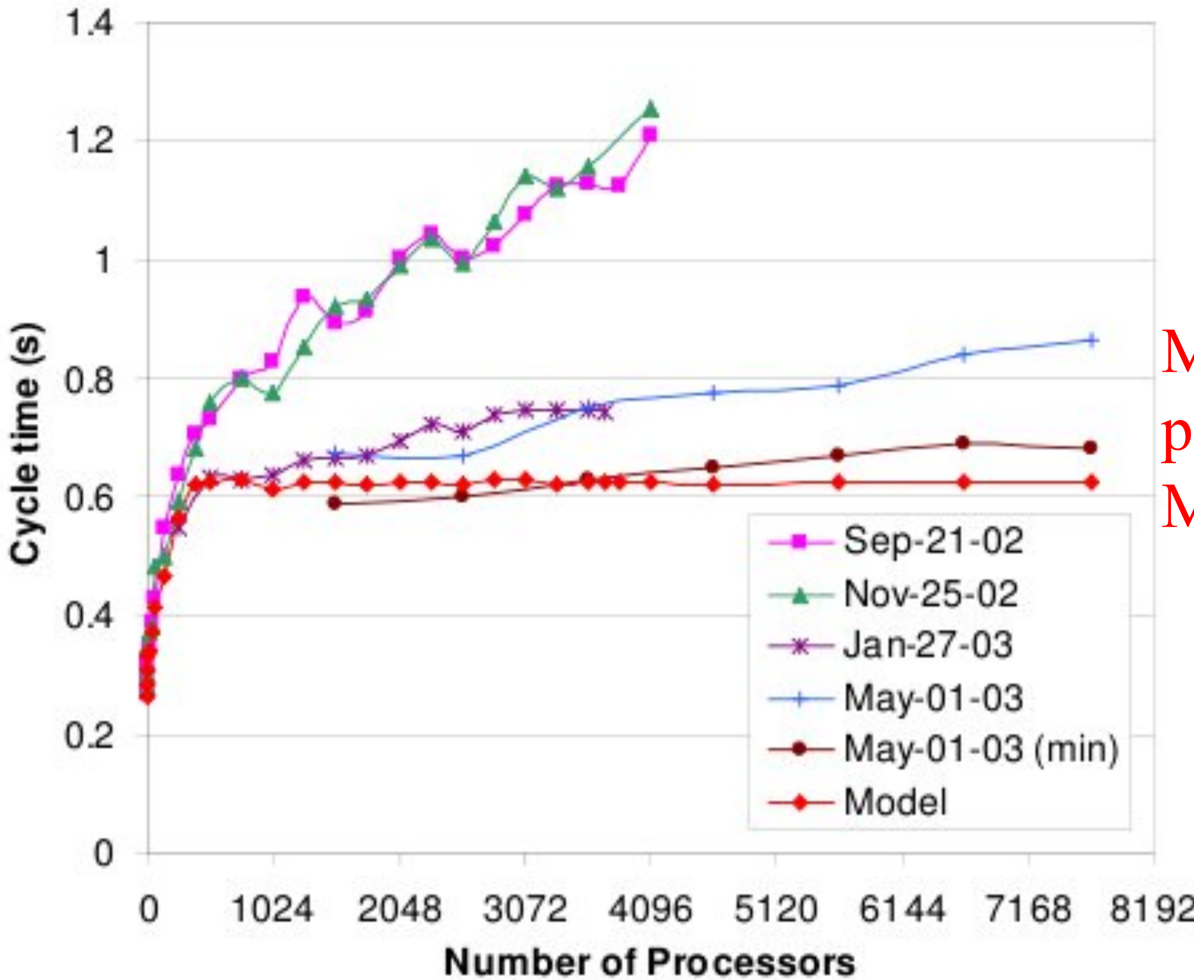
Co-scheduled Noise



(b) Coscheduled noise

All noise occurs in same timestep---much better than uncoordinated
Unfortunately, it’s not possible to do this in general in the OS

SAGE performance, part 2



Much improved performance in May

Figure 17: SAGE performance: expected and measured after noise removal

Summary of Missing Supercomputer Performance

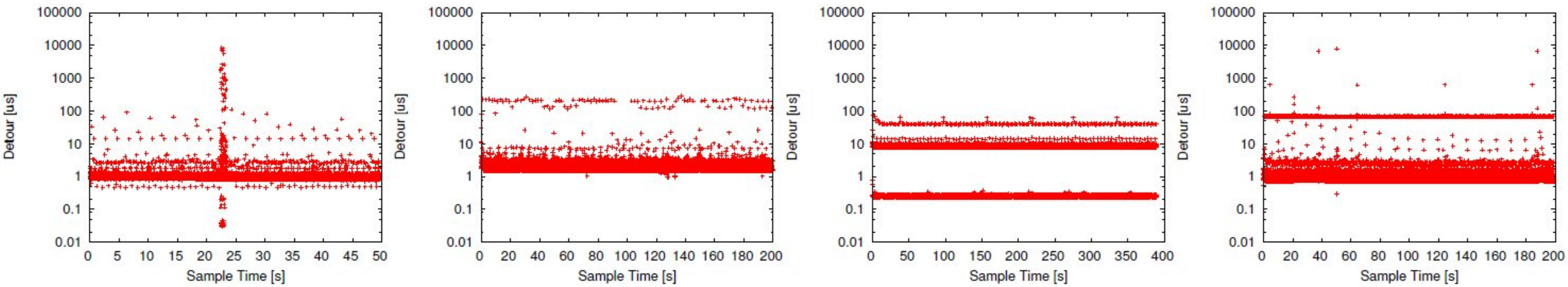
- Fine-grain synchronizing applications will be affected by high frequency, low duration noise
  - But not as much by low frequency, high duration noise
- Coarse-grain synchronizing applications will be relatively unaffected by high frequency, low duration noise (will [naturally] approach the co-scheduled picture)

Paper: Characterizing the Influence of System Noise on Large-Scale Applications by Simulation, by Hoefler et al.

- Main idea
  - use a LogGPS (variant of LogP) simulator to estimate program completion time under a variety of different hardware assumptions

Note: all following pictures from the Hoefler paper

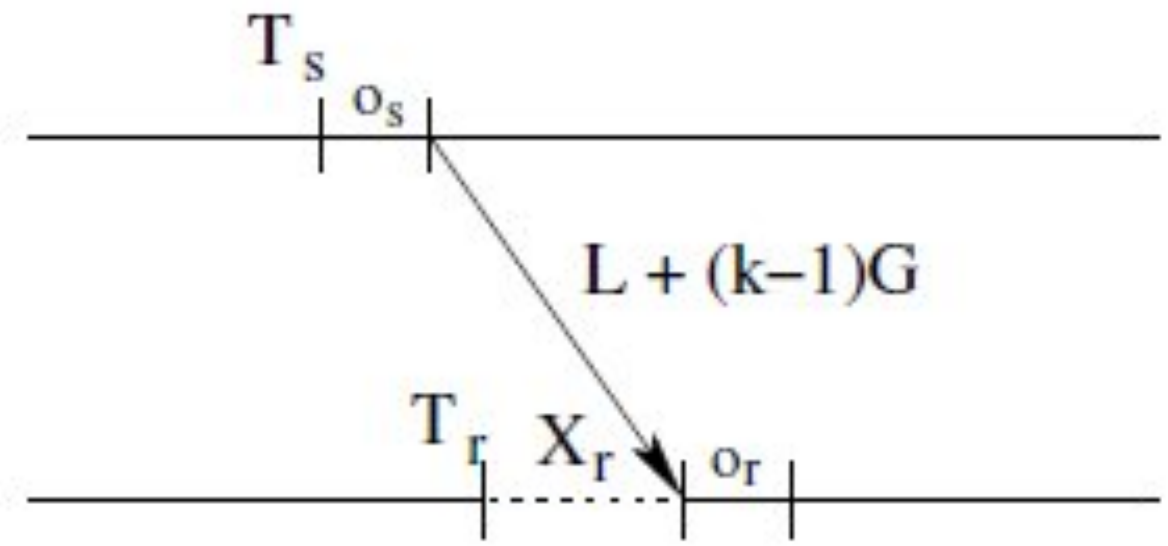
Measured Noise Patterns on Modern Supercomputer Nodes



(a) CHiC (Linux) (b) SGI Altix (c) ZeptoOS (d) Jaguar

All systems (other than BlueGene with CNK [not shown]) show noise
Noise in ZeptoOS is well-balanced, however

Blocking Communication Diagram

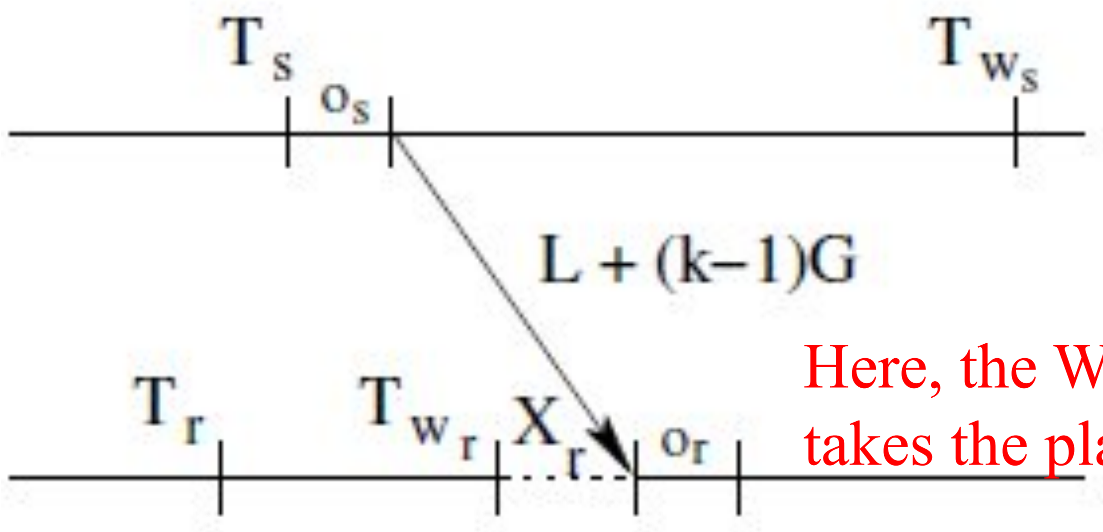


Blocking can occur on sender or receiver

---will almost surely occur on one, unless "perfectly timed" arrivals

---this picture shows "late sender", in terminology of the paper

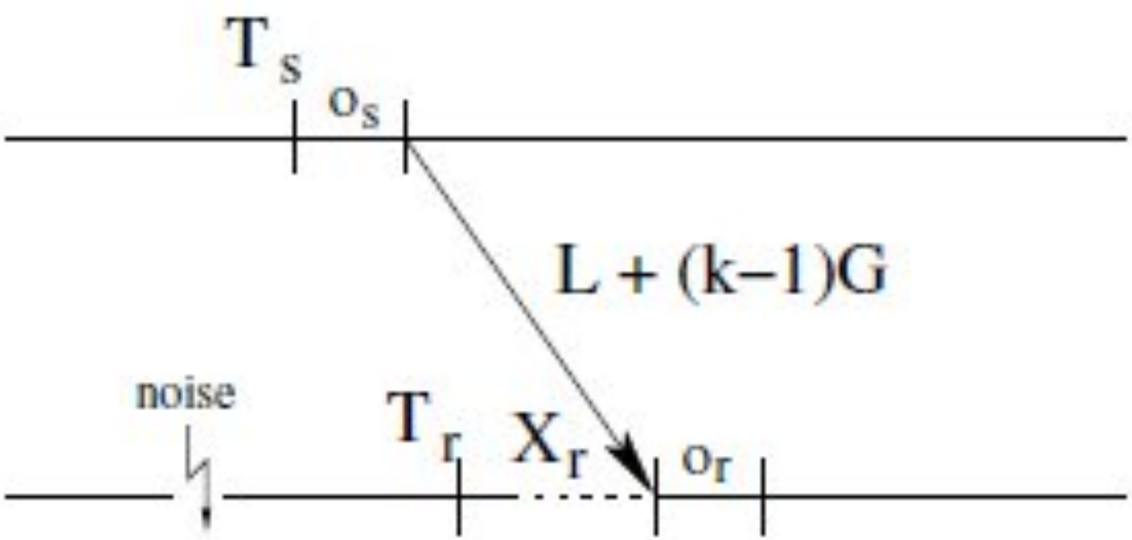
Nonblocking Communication Diagram



Here, the Wait at receiver takes the place of the Receive

Blocking can again occur on sender or receiver
---but less likely on receiver because receive is posted earlier
---this picture shows “waited for too early”, in terminology of the paper

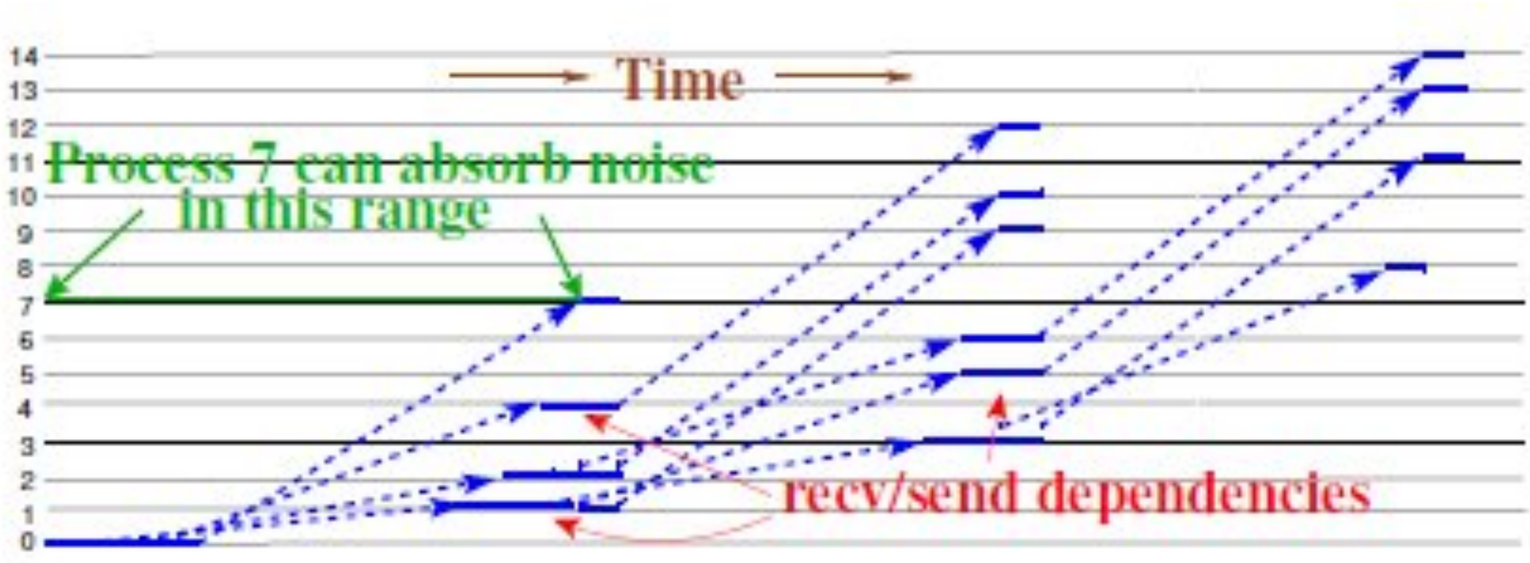
Absorption



In this figure the noise is absorbed

--note that absorption can happen on sender also if receive is invoked late enough

Modeling Broadcast is complicated



1. Multiple critical paths here (end at 8, 11, 13, 14)
2. A delay on any of these critical paths will delay Broadcast
3. If all ranks post Broadcast at same time, noise may be absorbed
4. But different amounts at different points in the tree

And this is just one possible implementation of Broadcast

Modeling Barrier is complicated

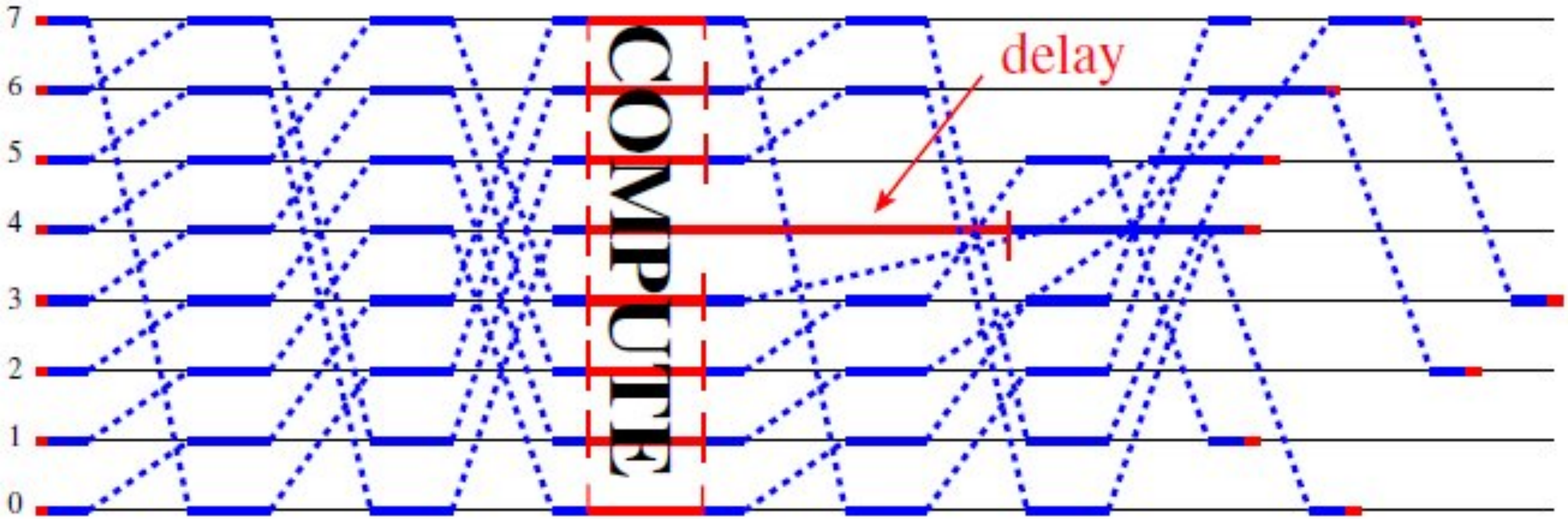


Fig. 4. LogGP diagram of two barriers with process 4 delayed ($P = 8$).

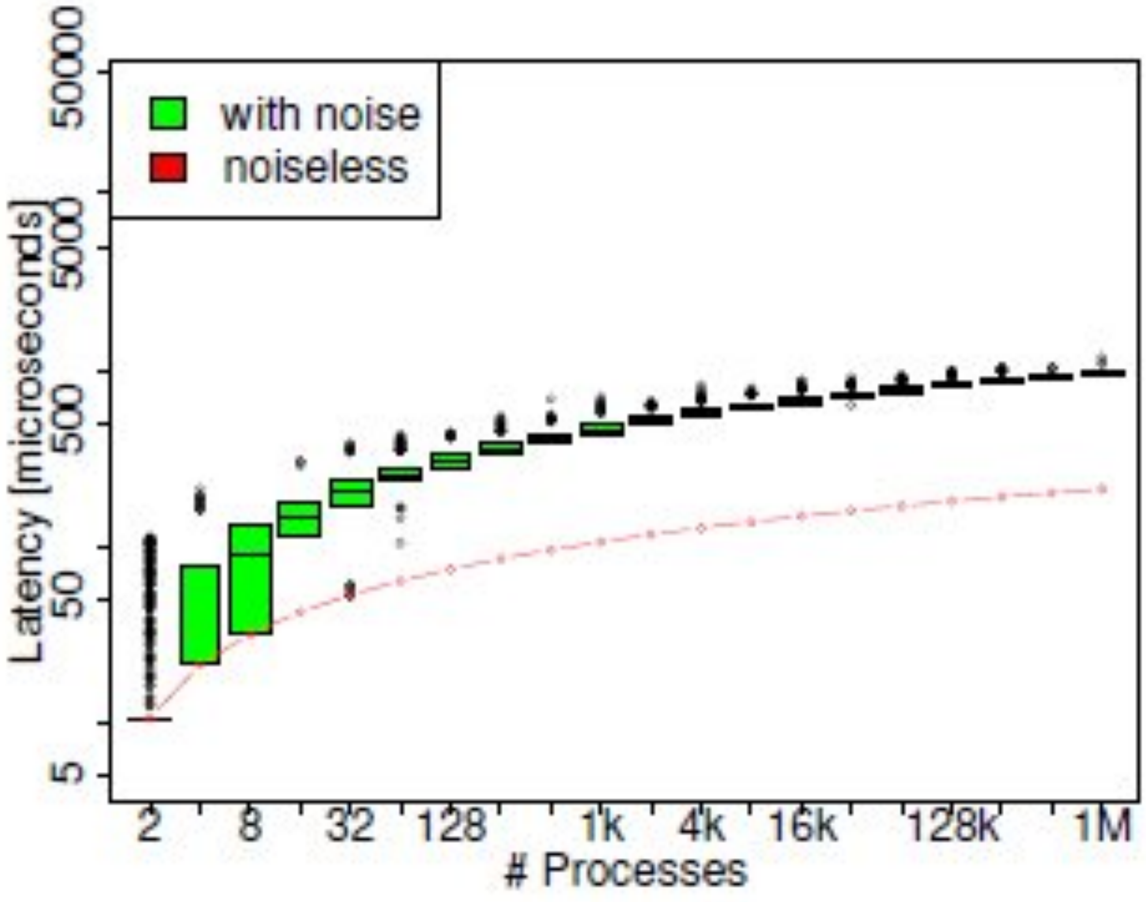
1. Dissemination barrier assumed
   A. Simpler, inferior implementations would be easier to analyze
2. The delay in rank 4 eventually resulted in largest delay on rank 3

Cannot treat a collective call as a black box

Basic idea of paper

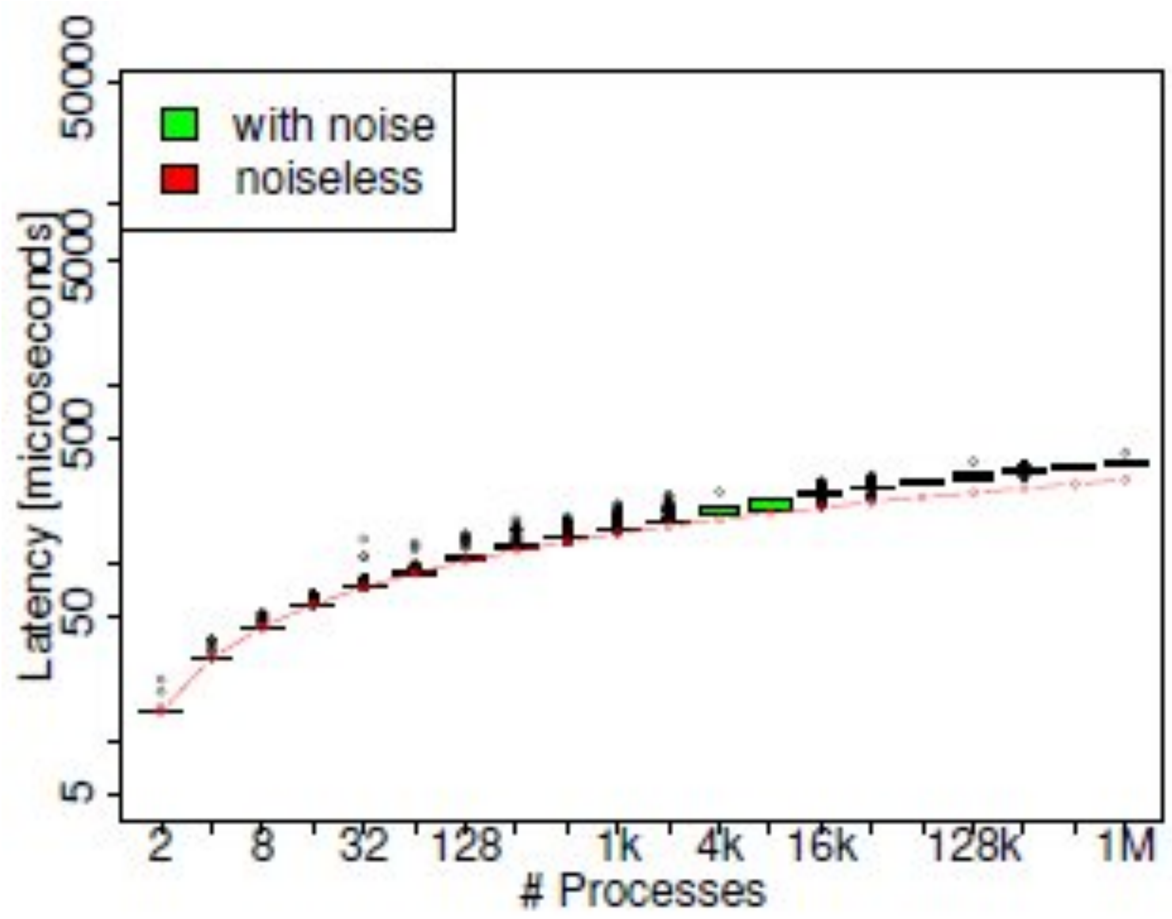
- As modeling is complicated, use a LogGPS simulator to study the effect of noise
  - The noise patterns from the real machines are fed in and used to inject noise in the experiments

Effect of noise on collectives



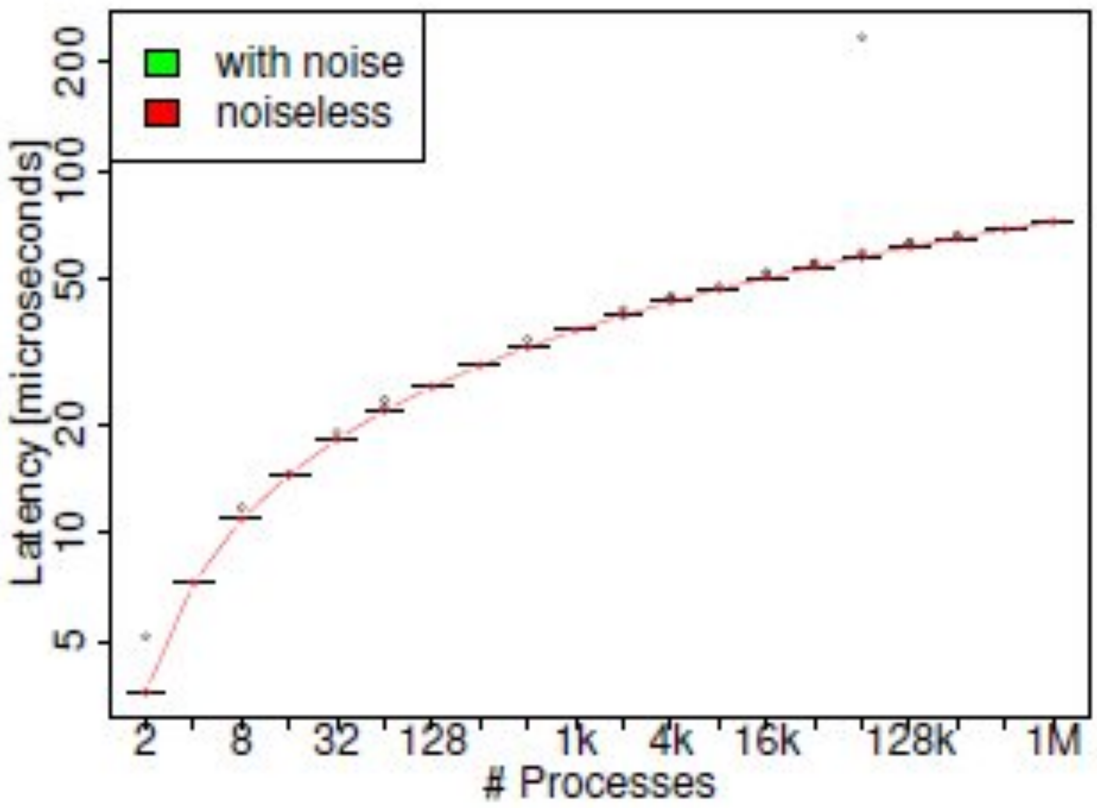
Very interesting figure: shows that the noise causes “convergence”
---but the convergence is awful (note this is a log scale on the y-axis)

Effect of noise on collectives, ZeptoOS



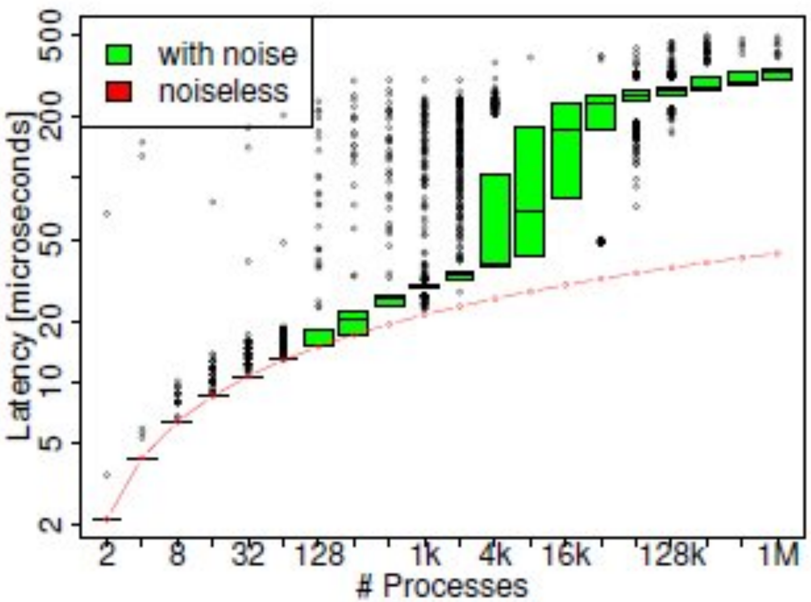
ZeptoOS balances noise well across nodes

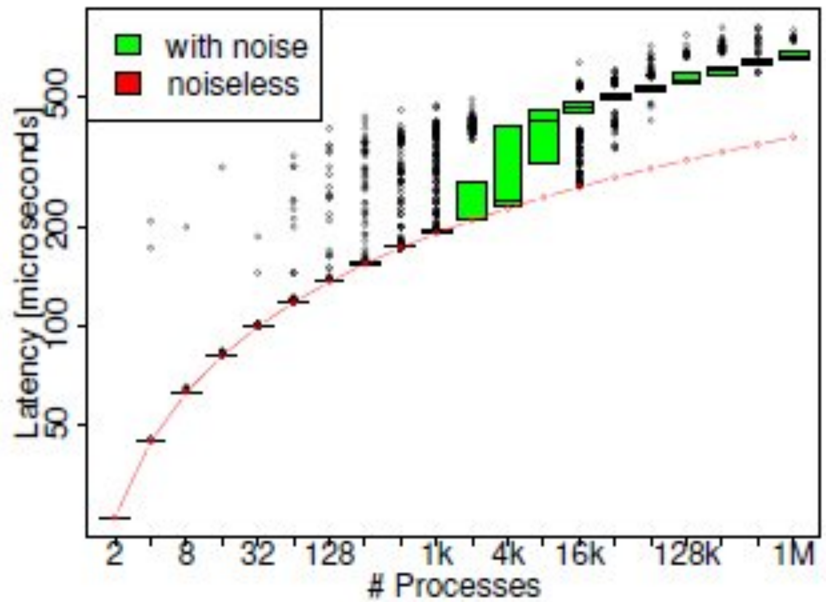
Co-scheduling (Synchronizing) Noise



Noise has a small effect; min time ~= median time ~= max time
However, outliers still exist

Effect of 10x and 0.1x network performance





Faster network (left) is worse when it comes to noise

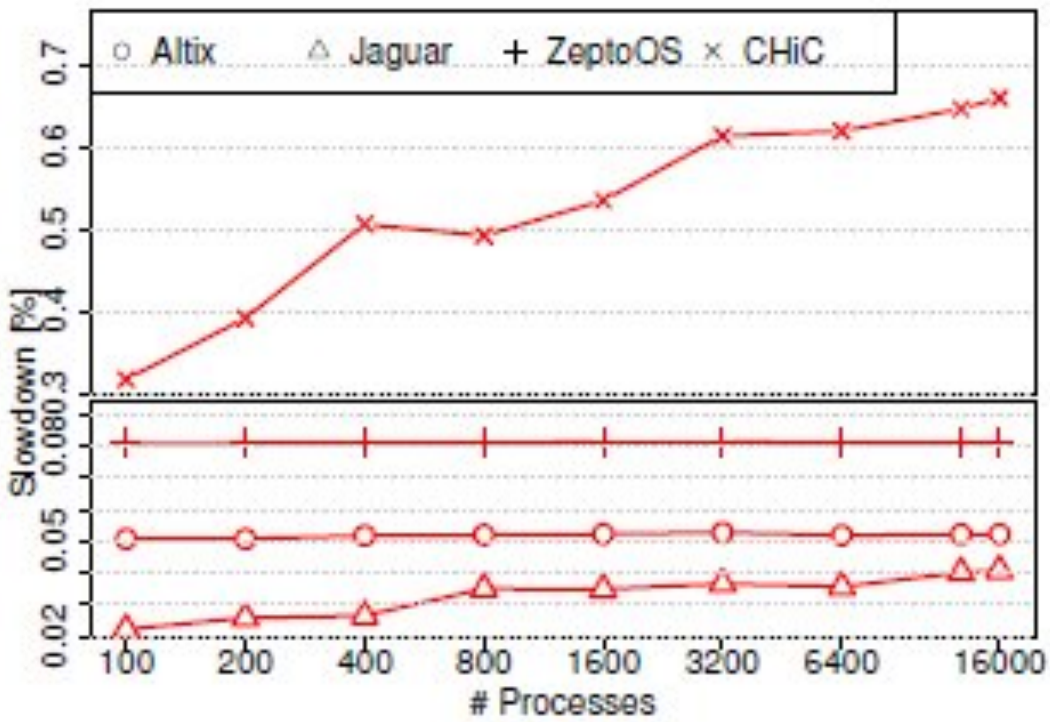
Performance at large number of processes is similar

So, why pay money for a faster network?

--called this the "noise bottleneck"

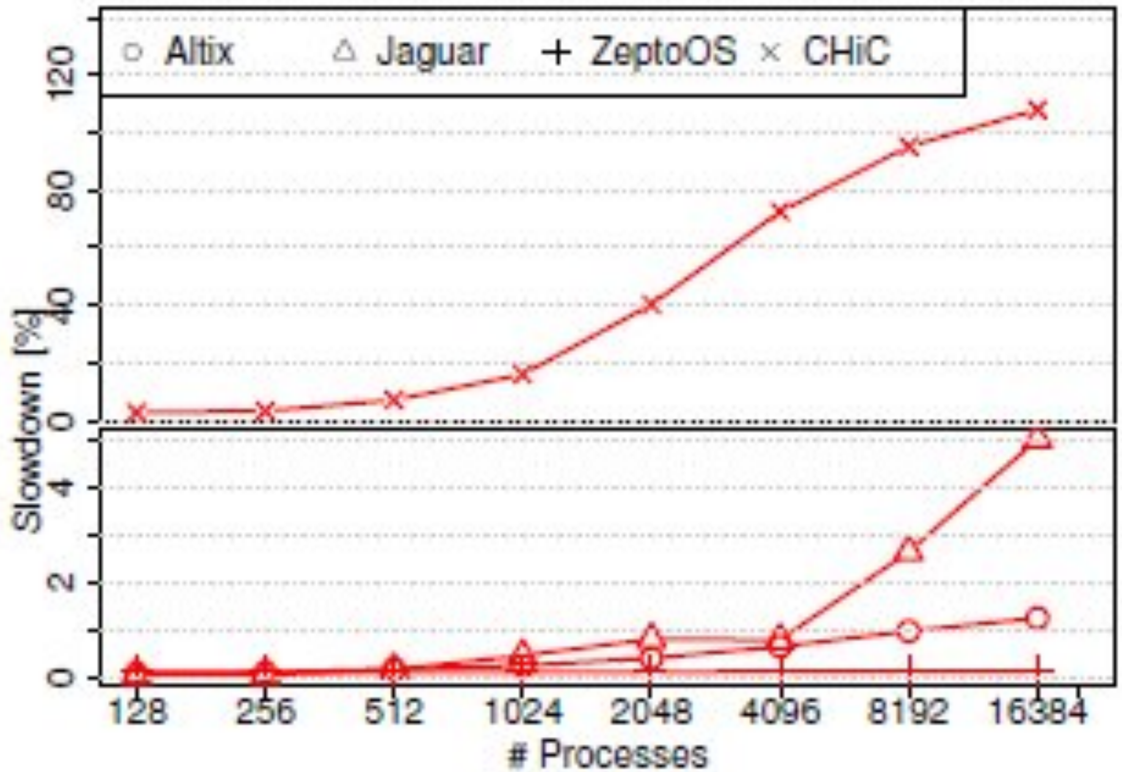
Note also: large messages reduce noise (larger transmission time)

Effect of noise on Sweep3d



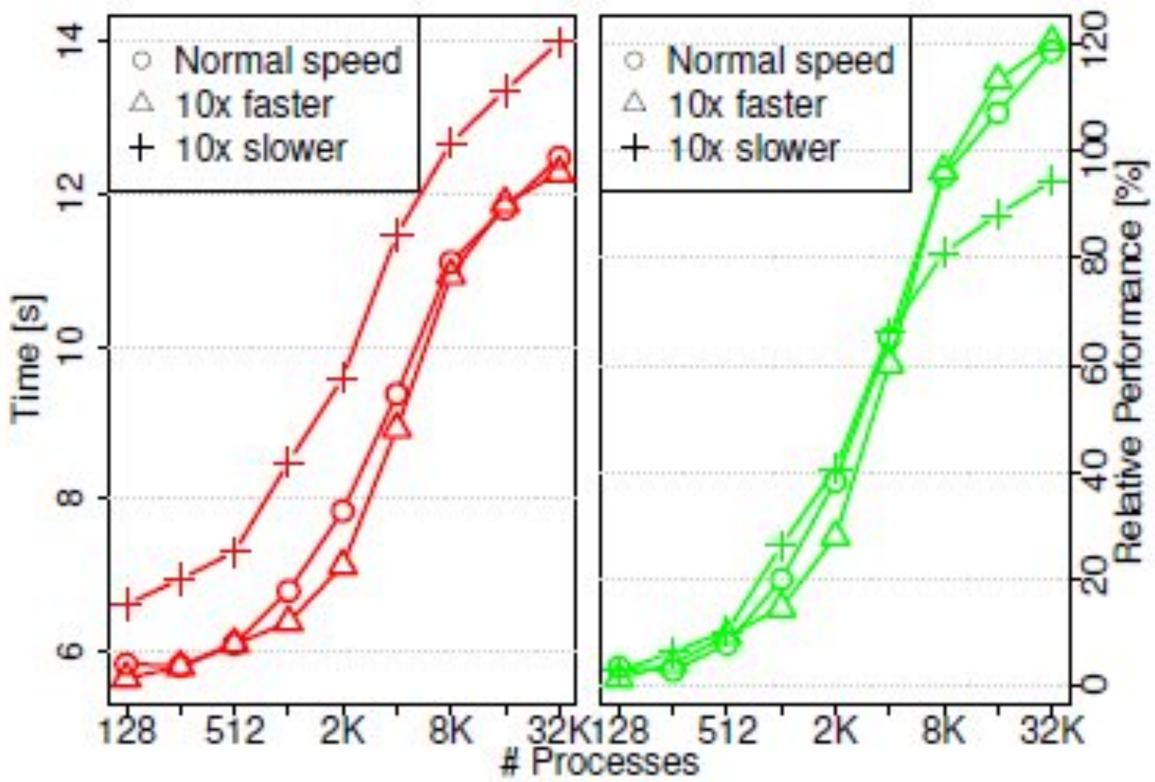
Virtually no effect

Effect of noise on POP



Can be large, depending on machine; and it's growing

Effect of network speed on applications



Similar effect to what is seen in the microbenchmarks