

Harnessing Parallelism in Multicore Clusters with the All-Pairs and Wavefront Abstractions

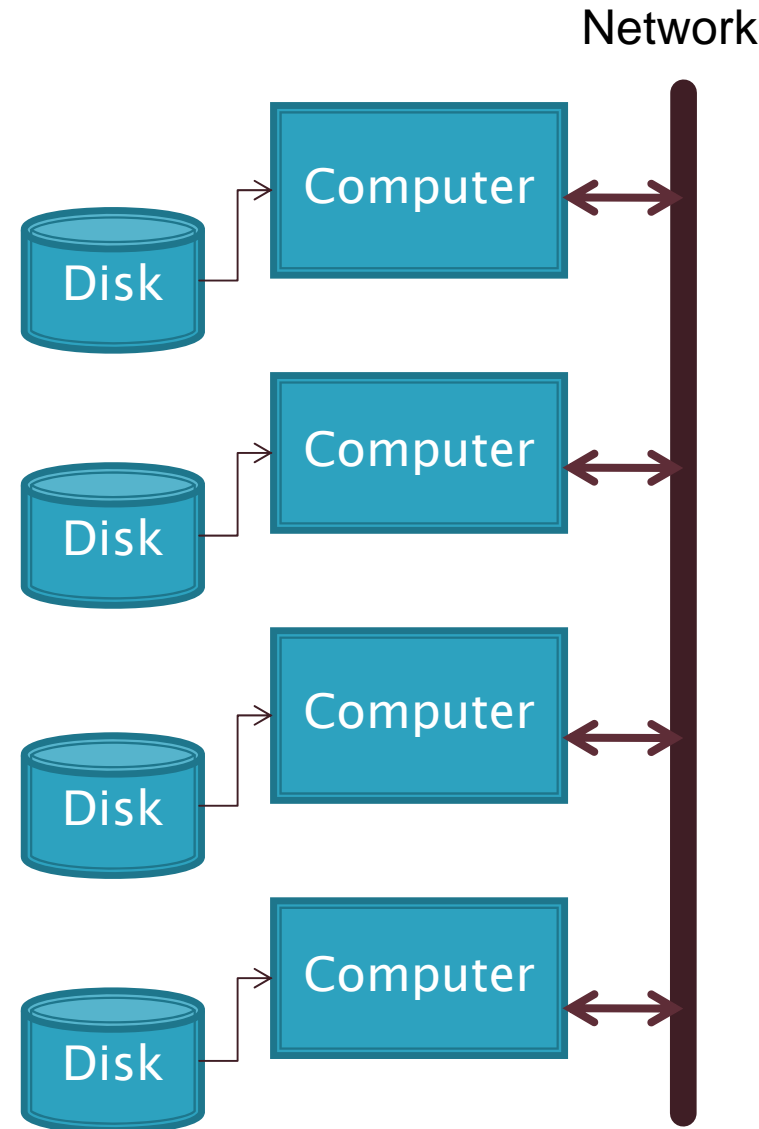
Li Yu, Christopher Moretti
Scott Emrich, Kenneth Judd*, Douglas Thain
University of Notre Dame, *Stanford University

Overview

- ▶ Distributed systems are hard to use effectively. Multicore distributed systems are even harder!
- ▶ An abstraction is a regular structure that can be efficiently scaled up to very large problem sizes.
- ▶ We have implemented two abstractions – AllPairs and Wavefront – with applications in biometrics, bioinformatics, and economics.
- ▶ Three Technical Results From Paper:
 - N-core CPU \neq N x 1-core CPUs
 - Abstractions make it easy to model performance accurately.
 - Aggressive failure detection is needed in order to scale up to large numbers of CPUs.

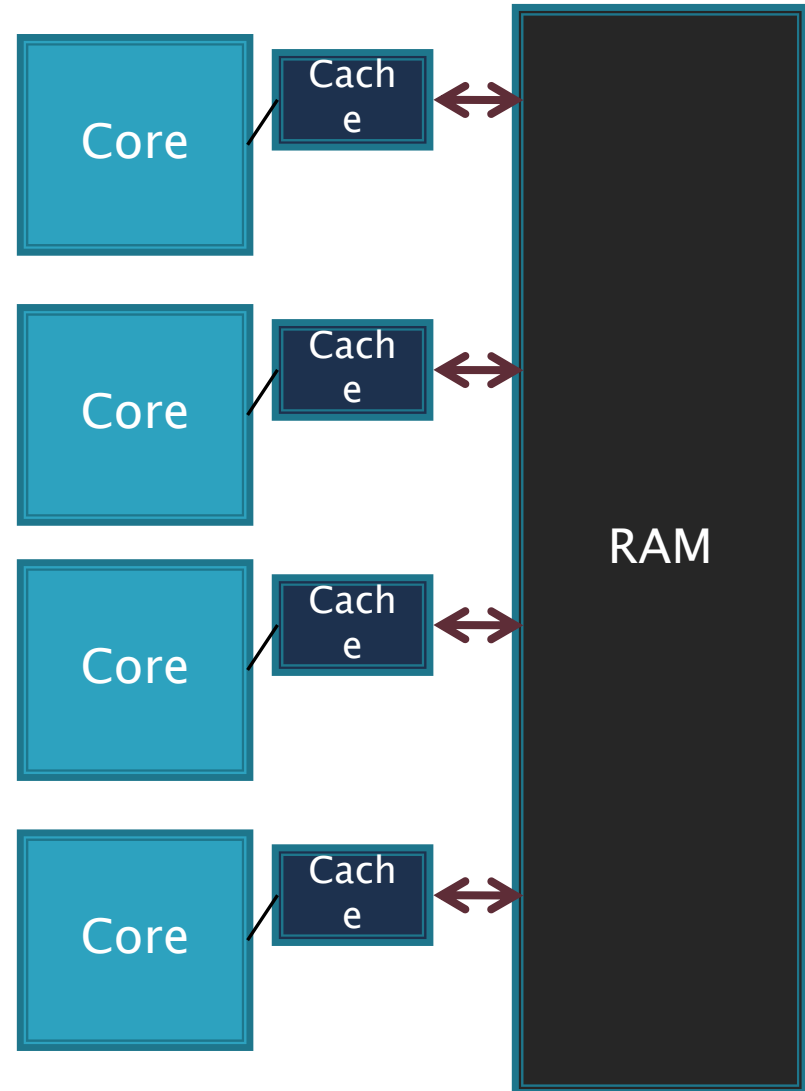
Clusters, clouds and grids are hard to use for large workloads

- ▶ How should the workload be broken up into jobs?
- ▶ How should the data be distributed to each node?
- ▶ How many nodes should be used?
- ▶ Will the network be a bottleneck?

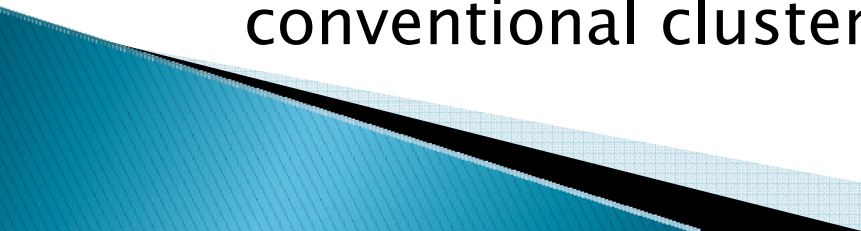


Multicore system is also hard to program

- ▶ How should work be divided among threads?
- ▶ Should we use message passing or shared memory?
- ▶ How many CPUs should be used?
- ▶ Will memory access present a bottleneck?

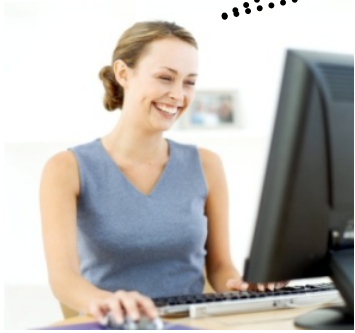


Solution: Abstractions

- ▶ An **abstraction** is a declarative framework that joins together sequential processes and data structures into a regularly structured parallel graph.
 - ▶ Our implementations distributed very large workloads on multicore CPUs, clusters, and clusters of multicore CPUs.
 - ▶ Not a general-purpose language, but a highly scalable implementation of a specific pattern.
 - ▶ (We first introduced the AllPairs abstraction for conventional clusters at IPDPS 2008.)
- 

Using Abstractions

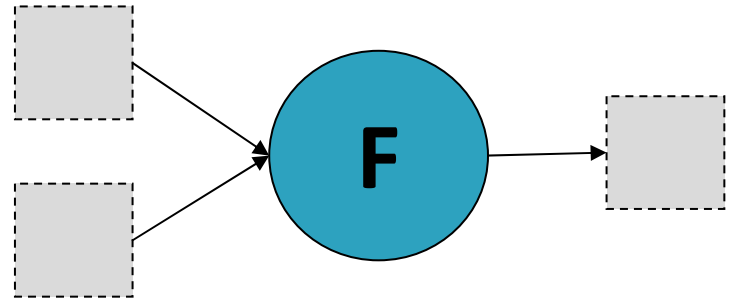
Here is my function: $F(x,y)$
Here is a folder of files: set S



set S of files



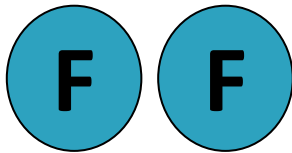
binary function F



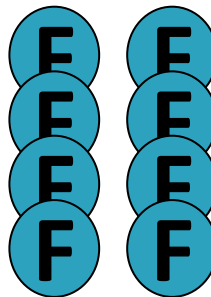
1CPU



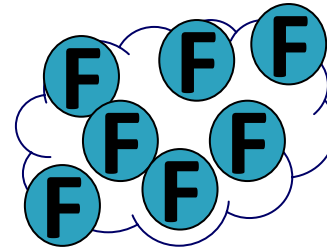
Multicore



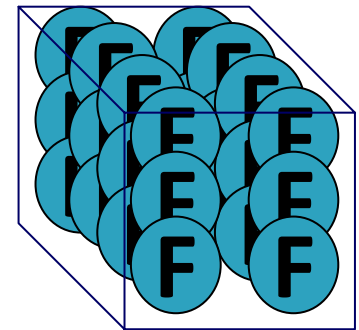
Cluster



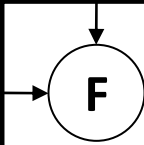
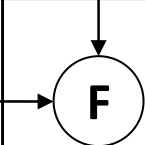
Grid



Supercomputer



AllPairs(A[i], B[j], F(a,b))

	A0	A1	A2	A3
B0		0.56	0.73	0.12
B1	0.14	0.19	0.33	0.75
B2	0.27	0.55	1.00	0.67
B3	0.12	0.84		1.00

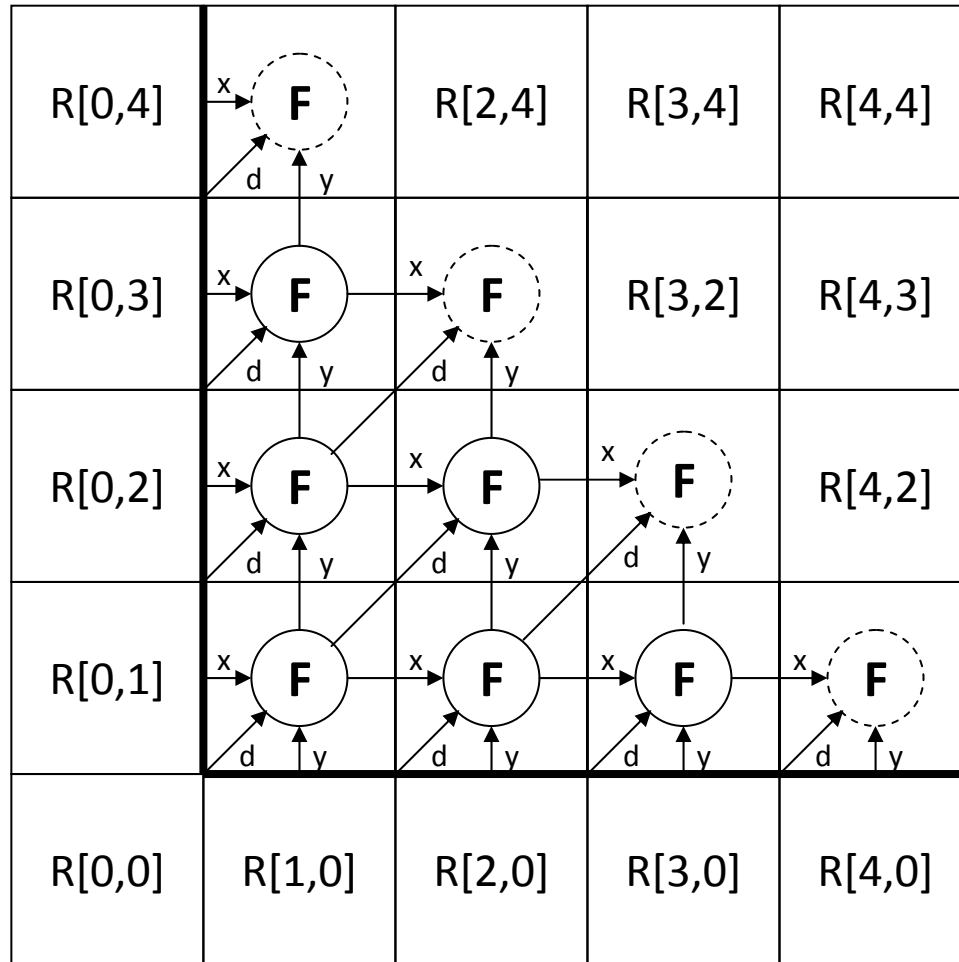
AllPairs for Biometrics

- ▶ Compare 4000 irises images of 1 MB to each other using a custom function.
- ▶ The comparison function converts each image into a iris code, and then computes the Hamming distance between the two.
- ▶ Each comparison takes about 1 second.
- ▶ CPU time under sequential computation:

185 DAYS! → 30 hours

With 200 machines

Wavefront($R[x,0]$, $R[0,y]$, $F(x,y,d)$)



Wavefront for Bioinformatics

- ▶ Computing a complete alignment between two very large DNA sequences.
- ▶ Compute a 100 by 100 matrix, where each cell is itself a large dynamic programming problem.
- ▶ CPU time under sequential computation:

13 DAYS! → 8.3 hours

With 80 machines

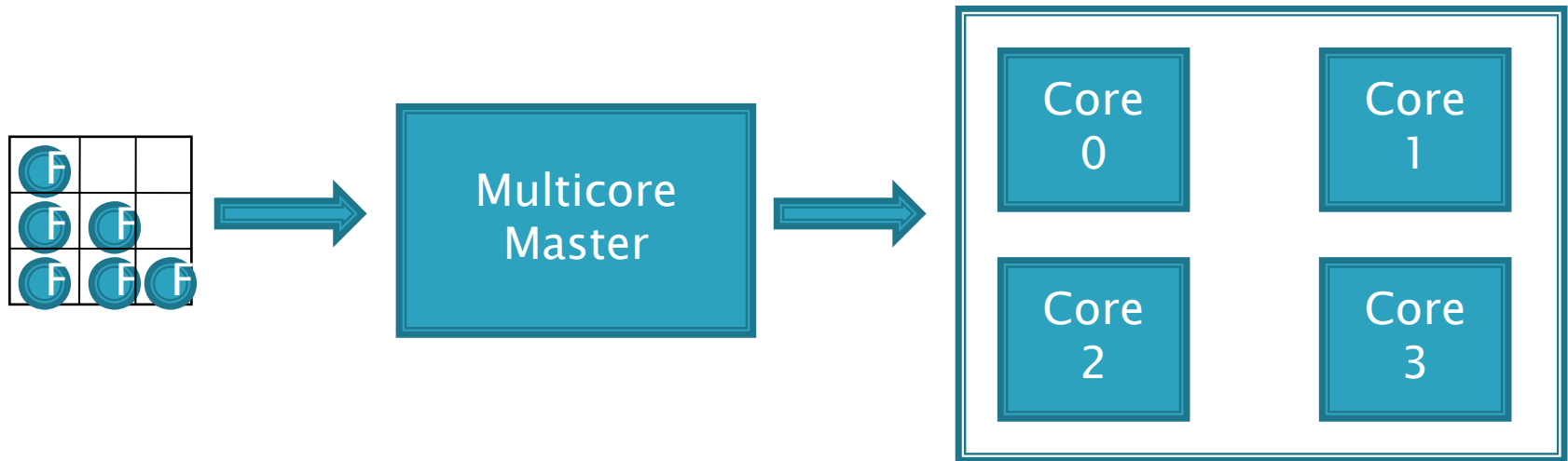
Wavefront for Economics

- ▶ A game theory problem in which we compute all possible actions of two competing companies over time.
- ▶ A 500 by 500 matrix where each cell requires 7.6 seconds to compute the Nash equilibrium.
- ▶ CPU time under sequential computation:

22 DAYS! → 2.9 hours

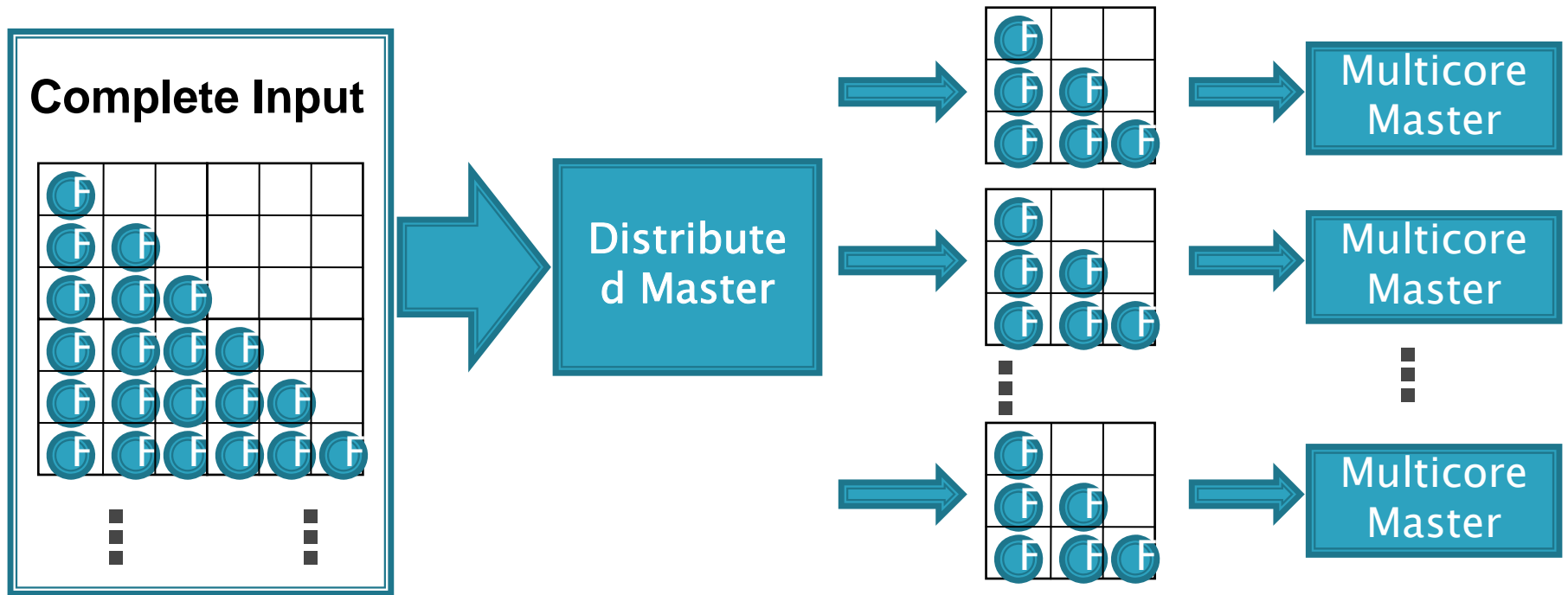
With 180 machines

Architecture – Multicore Master



- ▶ Responsibilities of the Multicore Master:
 - Divide work for each core.
 - Use multiple cores simultaneously.
 - Manage the file system cache.

Architecture – Distributed Master



- ▶ Responsibilities of the Distributed Master:
 - Divide entire problem into smaller pieces.
 - Decide how many nodes to use.
 - Recover from failed jobs efficiently.

Related Work

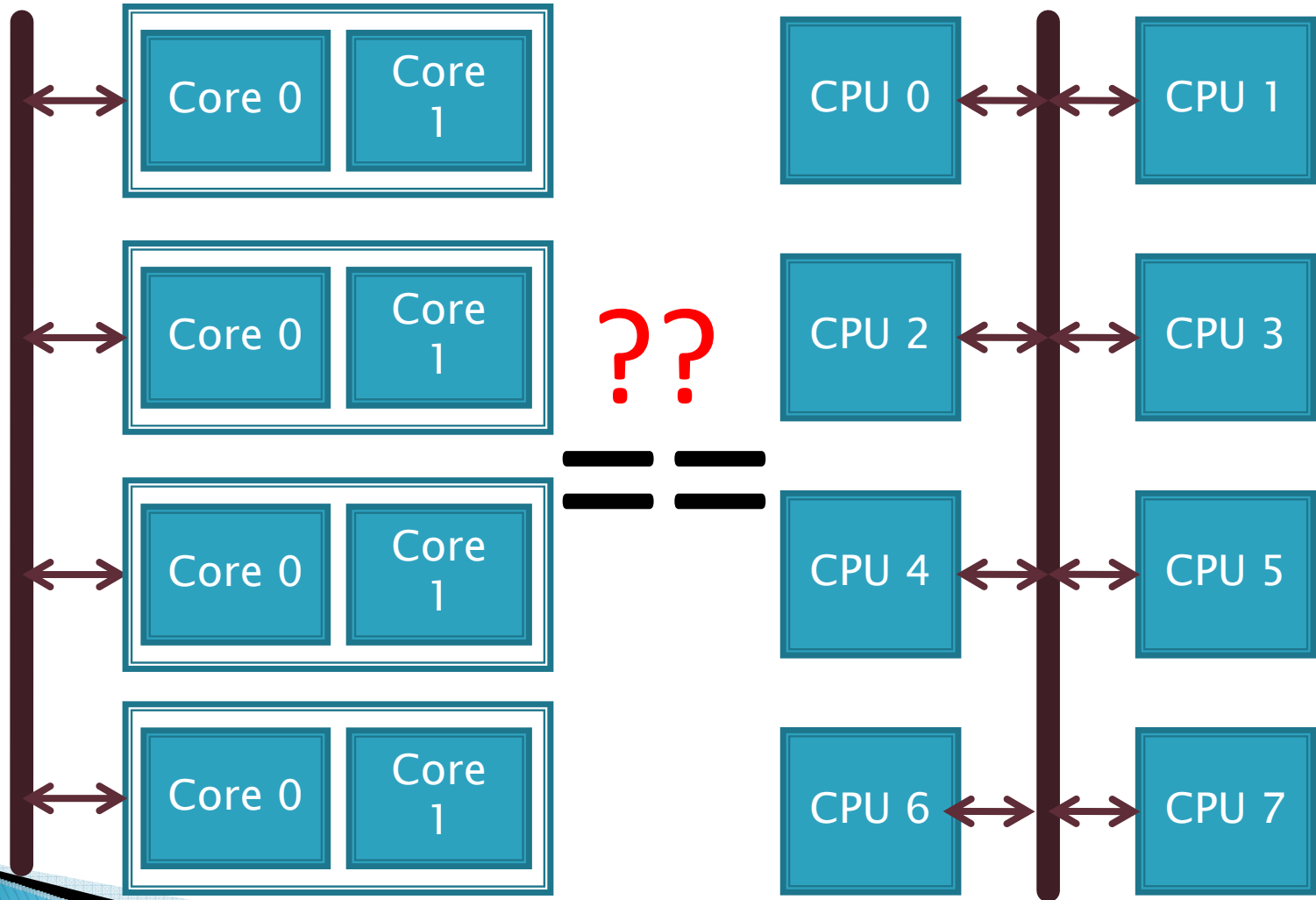
- ▶ **Parallel Languages: MPI, OpenMP**
 - General purpose languages can express arbitrary problems, but require experts to scale up to large sizes, and are not easily made fault tolerant.
- ▶ **Workflow Languages: Dagman, Swift, Taverna**
 - Designed and optimized to run irregular graphs of programs that deal with plain files.
Abstractions solve very specific flexibly divisible problems with seamless multicore integration.
- ▶ **Other Kinds of Abstractions: Map-Reduce**
 - Each abstraction solves a different category of problems. Is there a “menu” of 10 abstractions?

Three Technical Results:

- ▶ An N-core CPU should not be treated as $N \times 1$ -core CPUs.
- ▶ Abstractions make it easy to model performance accurately.
- ▶ Aggressive failure detection is needed in order to scale up to large numbers of CPUs.

(See paper for more results.)

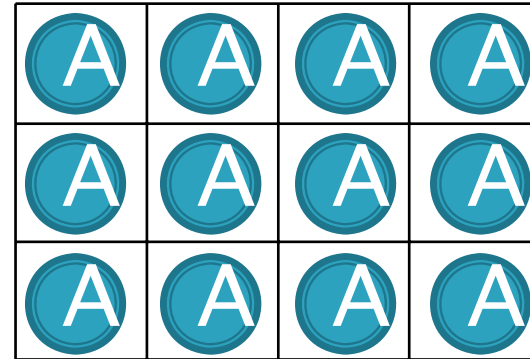
Can we treat one multicore node as multiple single core nodes?



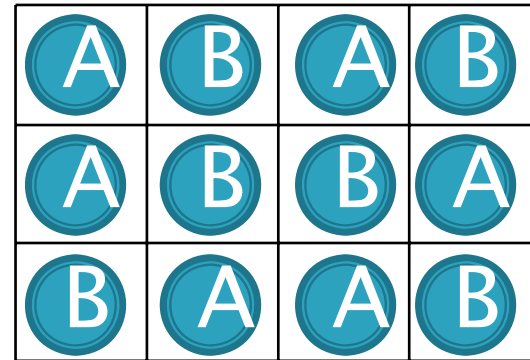
Three Options for Multicore

Block Width=2

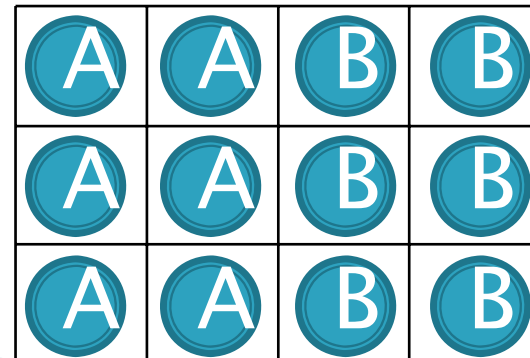
- ▶ Single Core



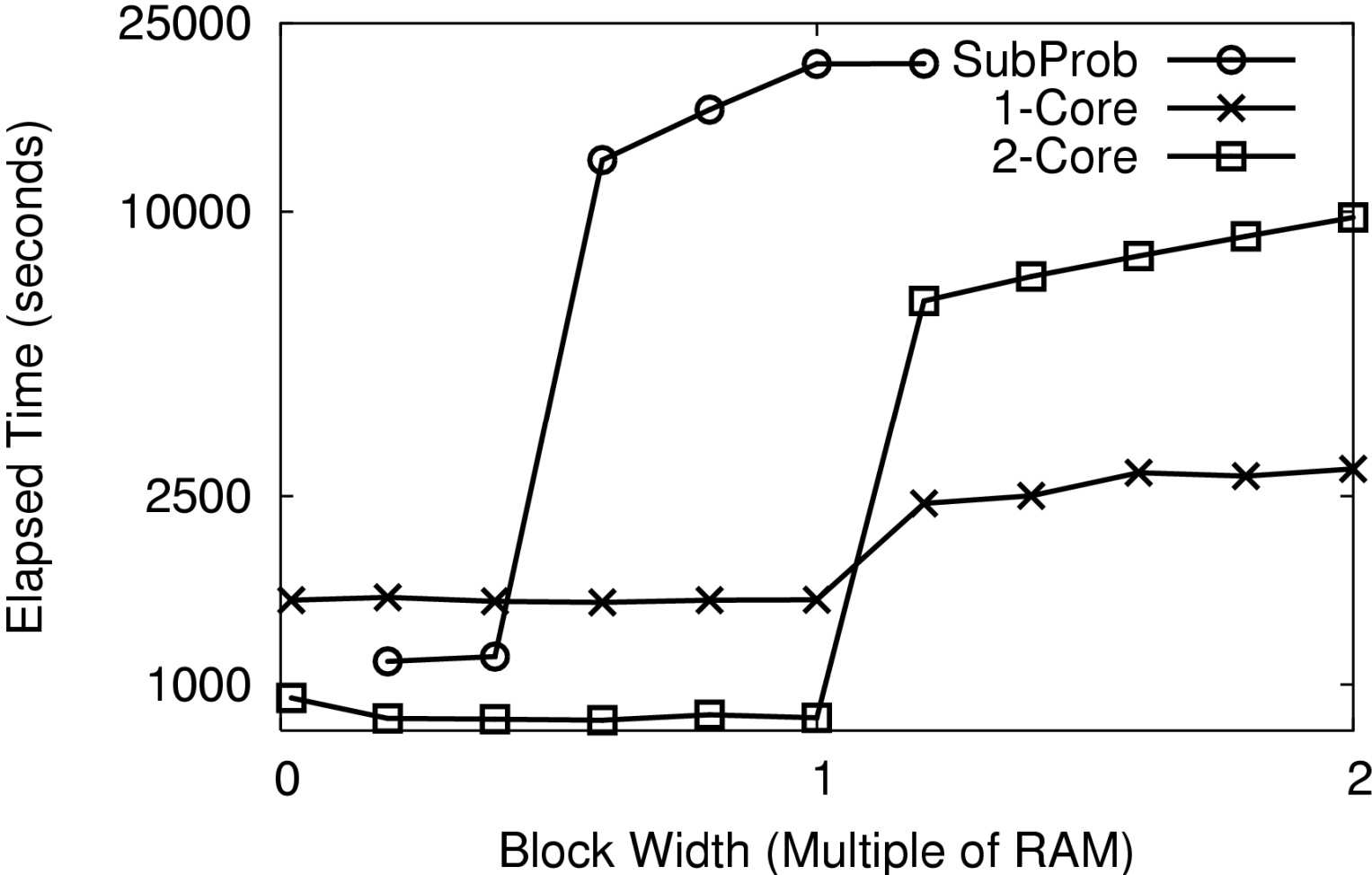
- ▶ Dual Core



- ▶ Sub-problem



Multicore vs Sub-problems



Conclusion:

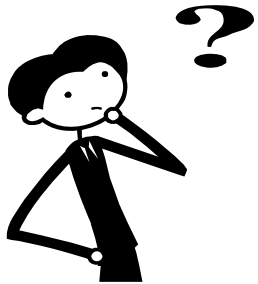
Multicore nodes need to be explicitly considered in the design of the system.

Modeling Performance



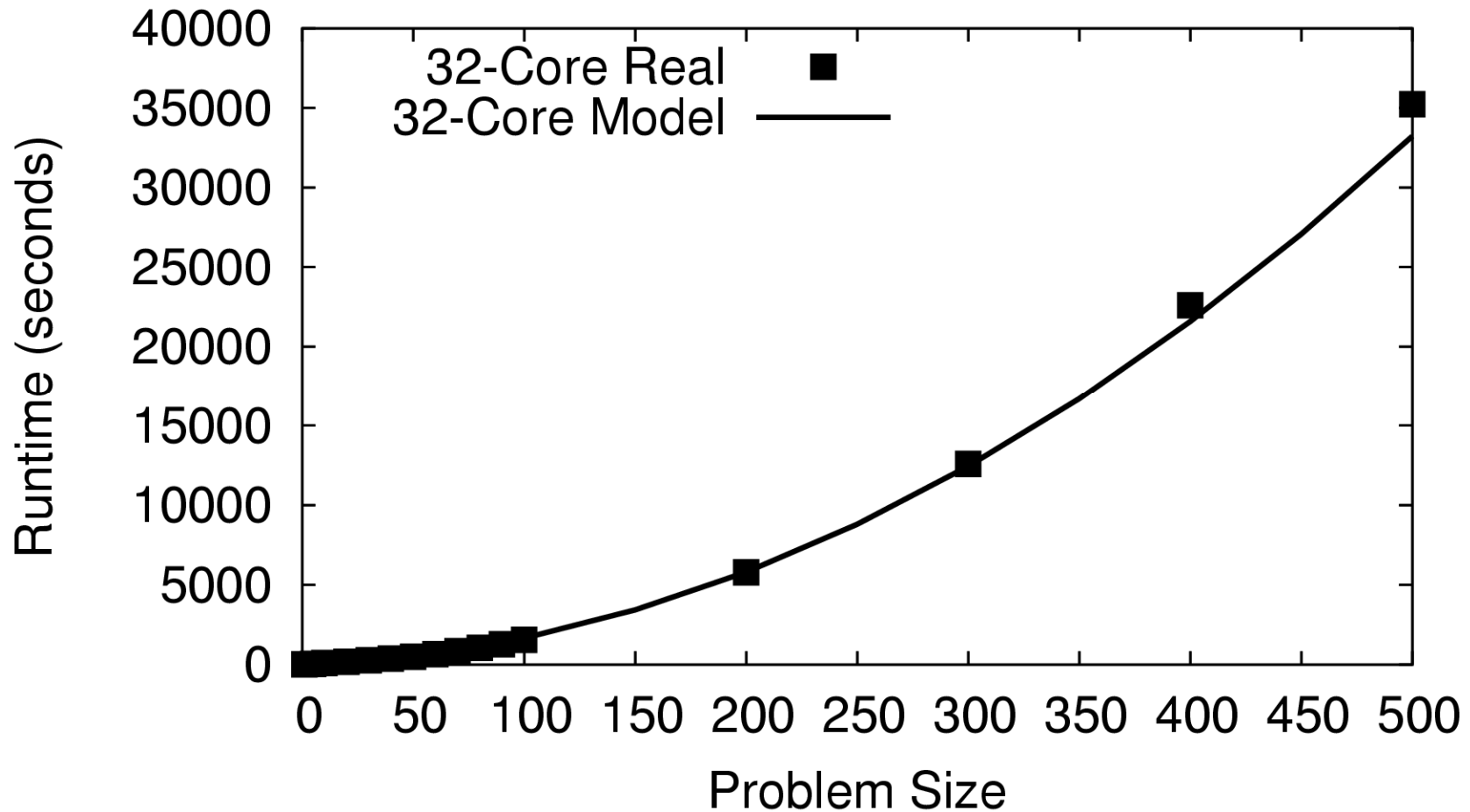
Local Mode:

Low dispatch latency.
Limited throughput.

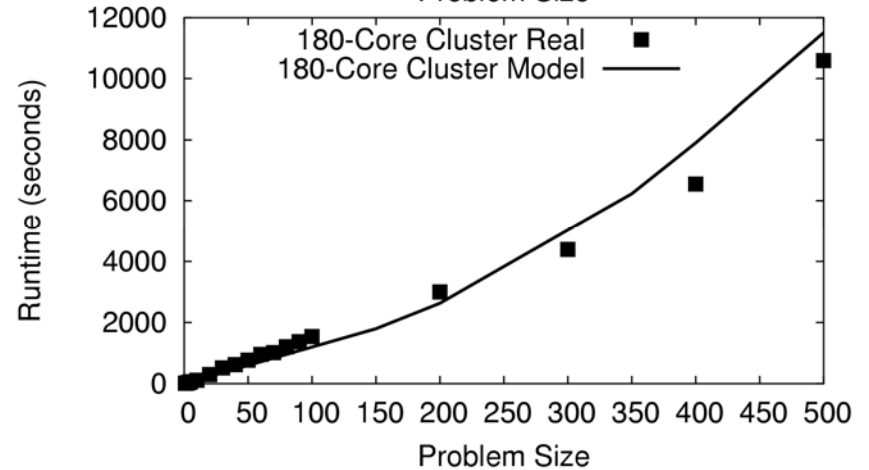
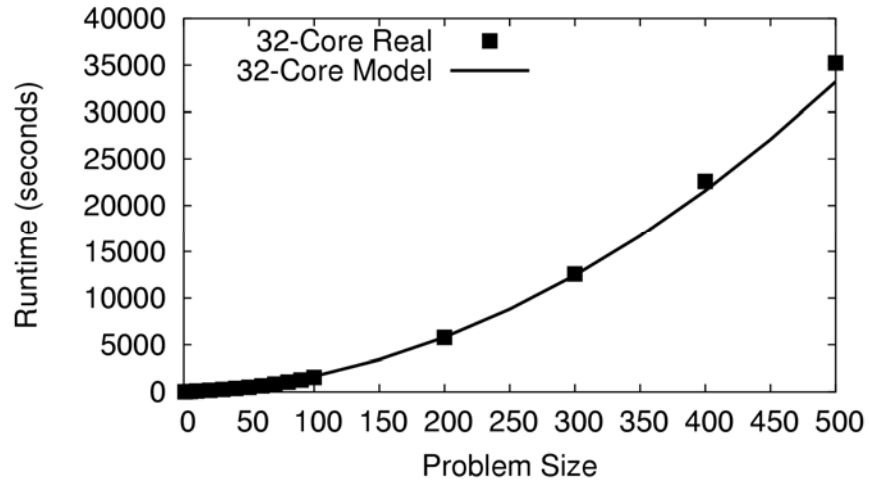
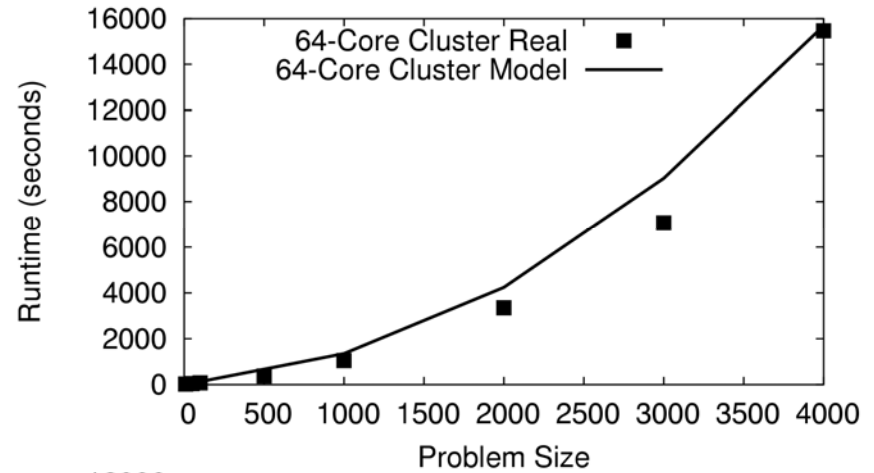
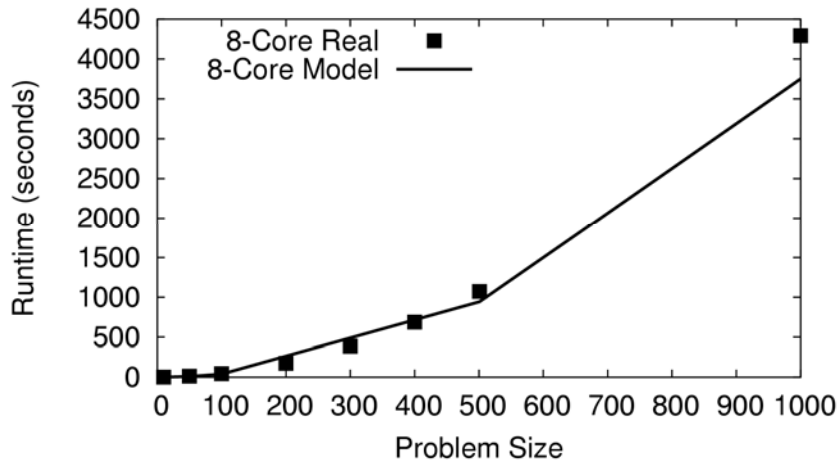


Cluster Mode:
High dispatch latency.
High throughput.

Modeling the Performance of Wavefront on a 32-core Machine

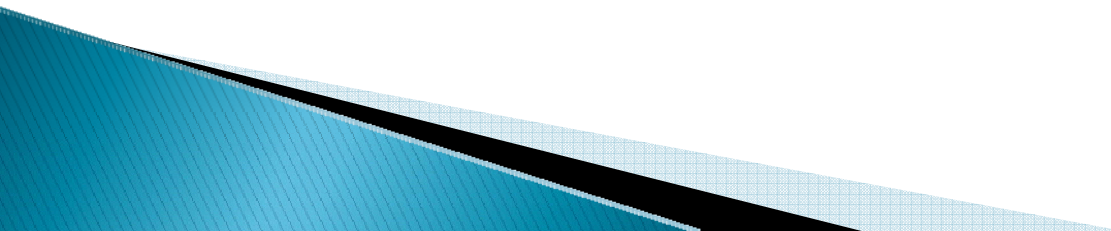


Modeling on Many Different Machines

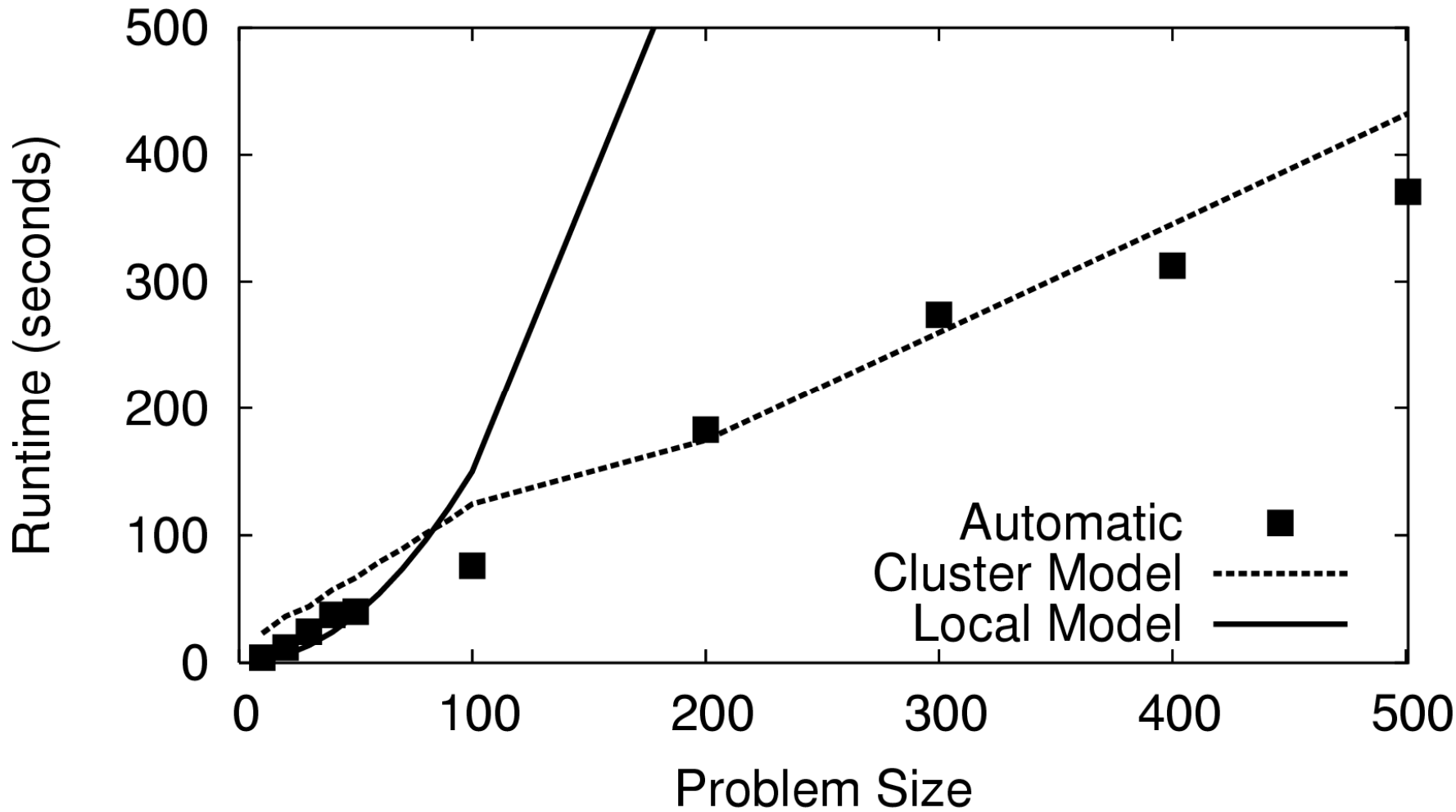


If we can model performance with reasonable accuracy...

Then we can pick the right implementation automatically at runtime!



Selecting an Implementation



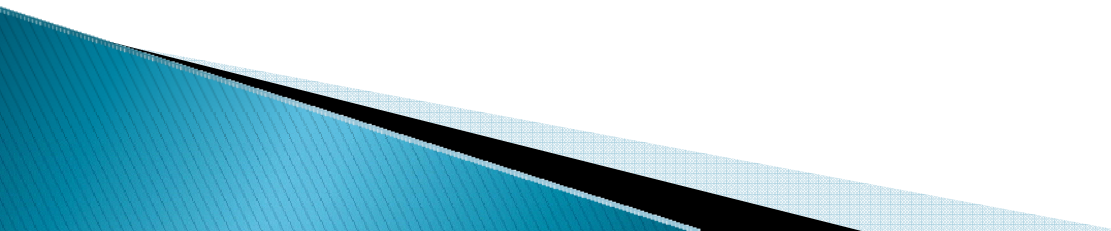
Because the abstraction is a highly restricted framework...

We know the CPU and data needs.
We can measure the performance of one function call on the system
















Thus, predicting runtime is possible.
(But not always perfect.)



















**How do we deal with failures
and slowdowns that happen in
the real world?**



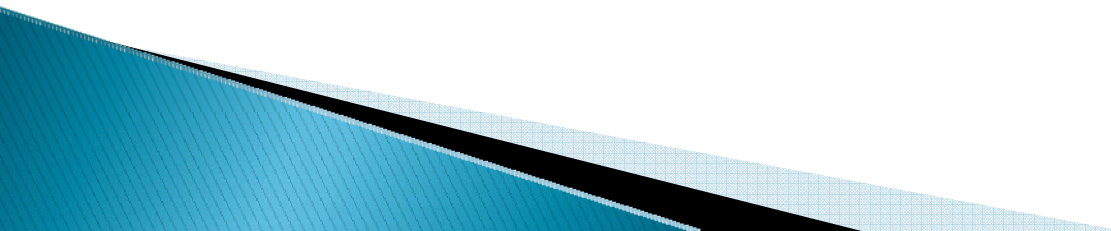
Wavefront in an Ideal World

R[0, 5]					
R[0, 4]					
R[0, 3]					
R[0, 2]					
R[0, 1]					
R[0, 0]	R[1, 0]	R[2, 0]	R[3, 0]	R[4, 0]	R[5, 0]

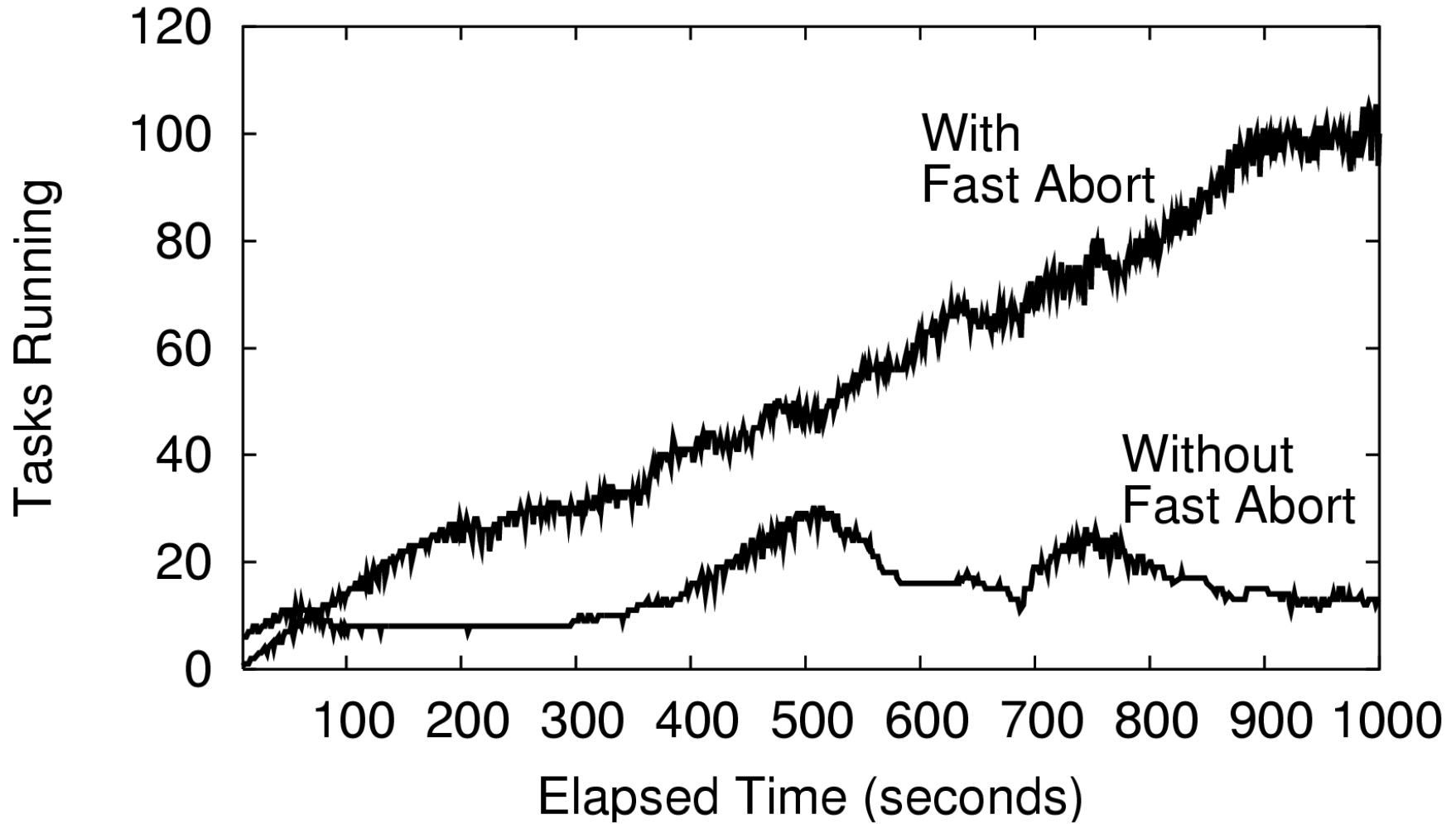
Wavefront in the **Real** World

R[0, 5]					
R[0, 4]					
R[0, 3]					
R[0, 2]					
R[0, 1]					
R[0, 0]	R[1, 0]	R[2, 0]	R[3, 0]	R[4, 0]	R[5, 0]

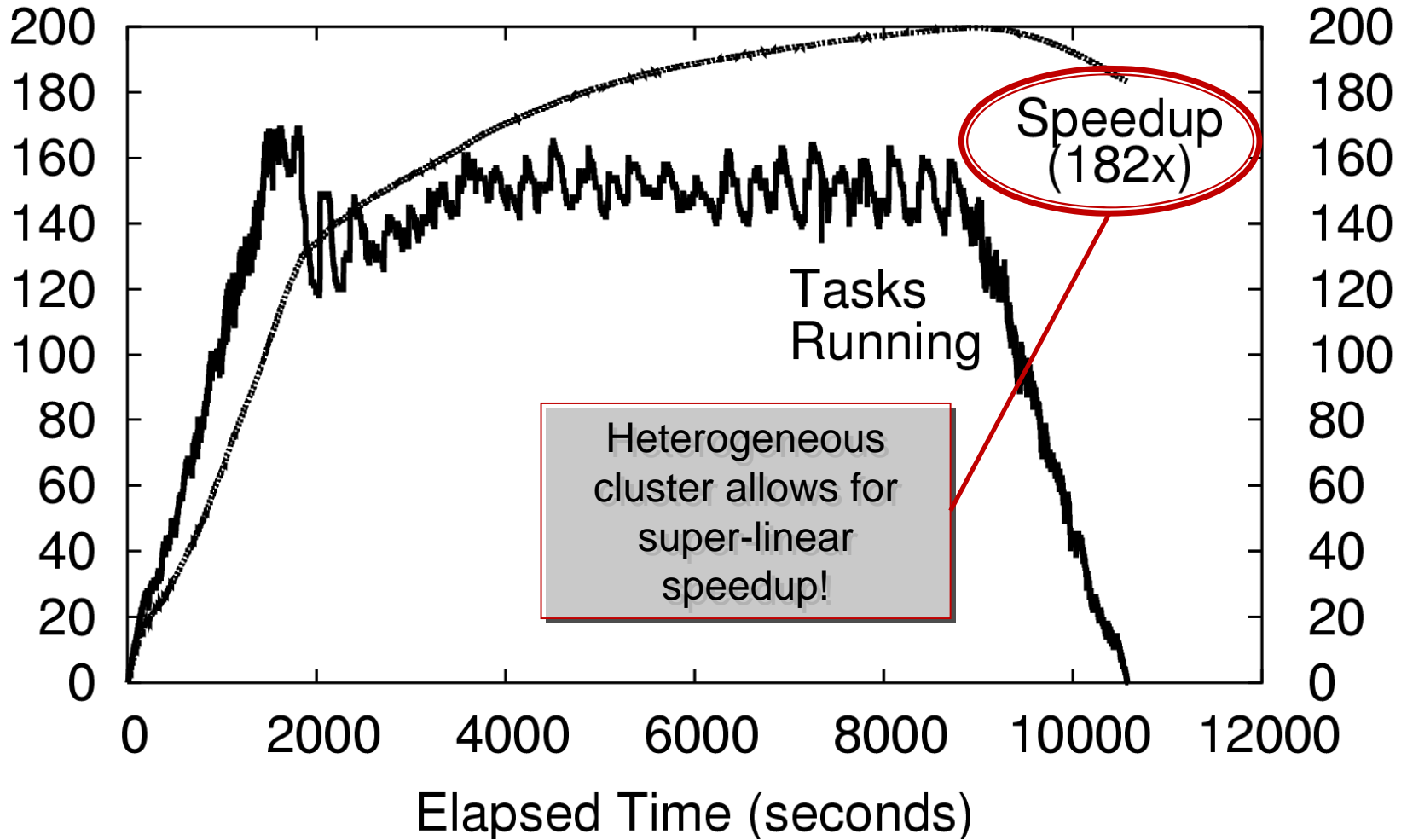
Solution: Fast Abort

- ▶ Keep statistics on function execution times.
 - ▶ When a work unit runs 10x longer than the average, abort and reschedule it.
 - ▶ Prefer to run on fast machines.
- 

Control Flow with Fast Abort



Real Economics Performance



Conclusion

- ▶ Distributed systems are hard to use effectively. Multicore distributed systems are even harder!
- ▶ An abstraction is a regular structure that can be efficiently scaled up to very large problem sizes.
- ▶ We have implemented two abstractions – AllPairs and Wavefront – with applications in biometrics, bioinformatics, and economics.
- ▶ Three Technical Results From Paper:
 - N -core CPU \neq $N \times$ 1-core CPUs
 - Abstractions make it easy to model performance accurately.
 - Aggressive failure detection is needed in order to scale up to large numbers of CPUs.

For More Information

- ▶ Li Yu
 - lyu2@nd.edu
- ▶ Douglas Thain
 - dthain@cse.nd.edu
- ▶ Cooperative Computing Lab
 - <http://www.cse.nd.edu/~ccl>