BAD-Check

A Story of Reclaiming Broken Promises and Rediscovering Lost Potential

John Bent
Office of the CTO, EMC
PDSW, November 2015

EMC²

BULK ASYNCRONOUS DISTRIBUTED CHECKPOINTING AND IO COAUTHORS

- Los Alamos National Lab
  - Brad Settlemyer
  - Jeremy Sauer
- EMC
  - Sorin Faibish
  - Haiyun Bao
  - Jingwang Zhang

SUMMARY OF TODAY'S TALK

Asynchronous computing is the future

- Asynchronous computing needs asynchronous checkpoint
- [It helps synchronous computing too; just less so.]

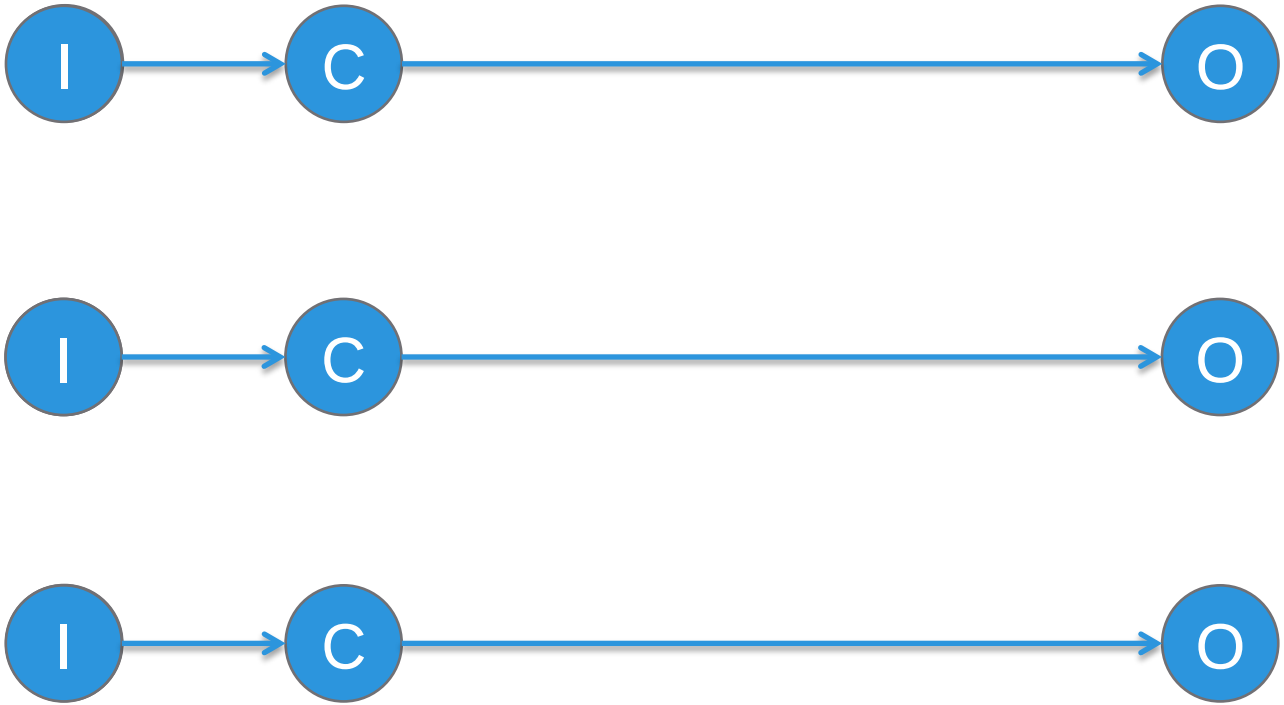
Our contributions

0. An instance of asynchronous checkpointing
   - Bulk-asynchronous distributed transactions (from DOE FF)
1. An evaluation methodology
   - Hotspots, heat, movement, communications
2. A simulated bulk synchronous application
   - 18% runtime or 40% storage reduction

LIFE OF A COMPUTE JOB

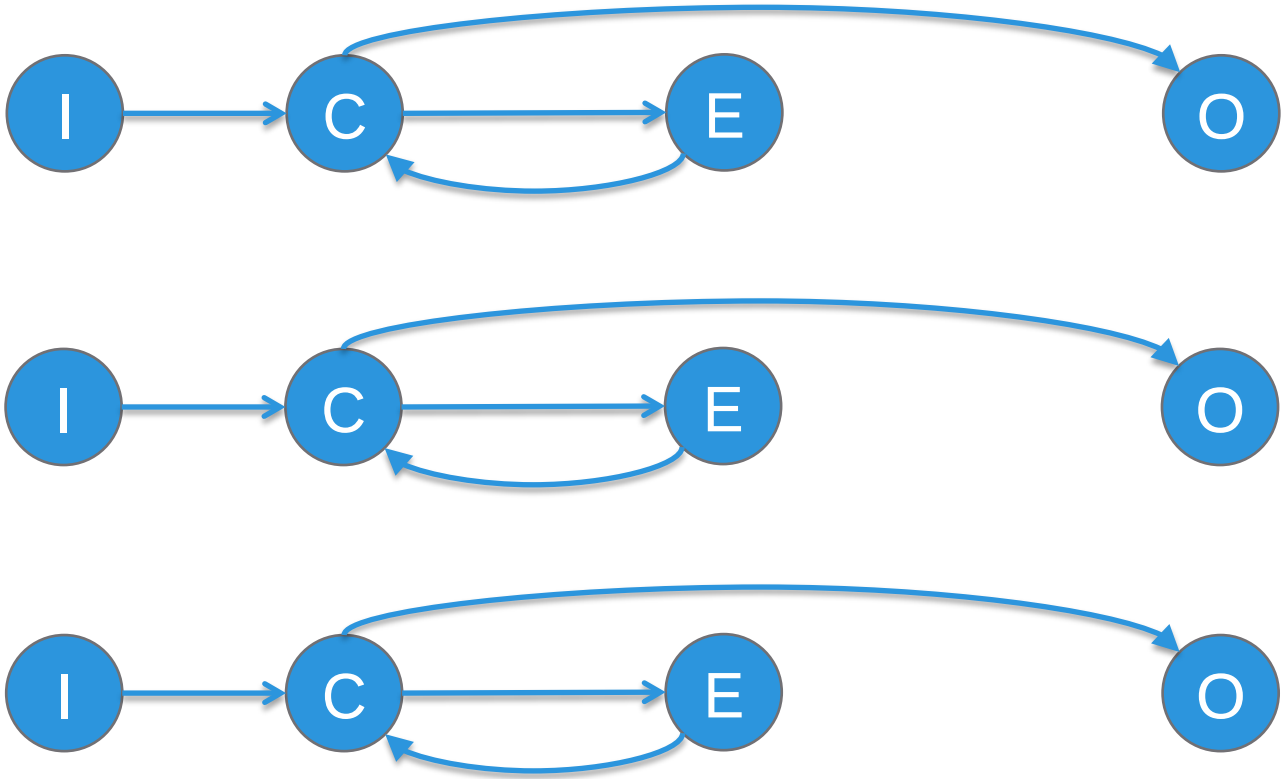
I: Input **C**: Compute **O**: Output

LIFE OF A COMPUTE JOB



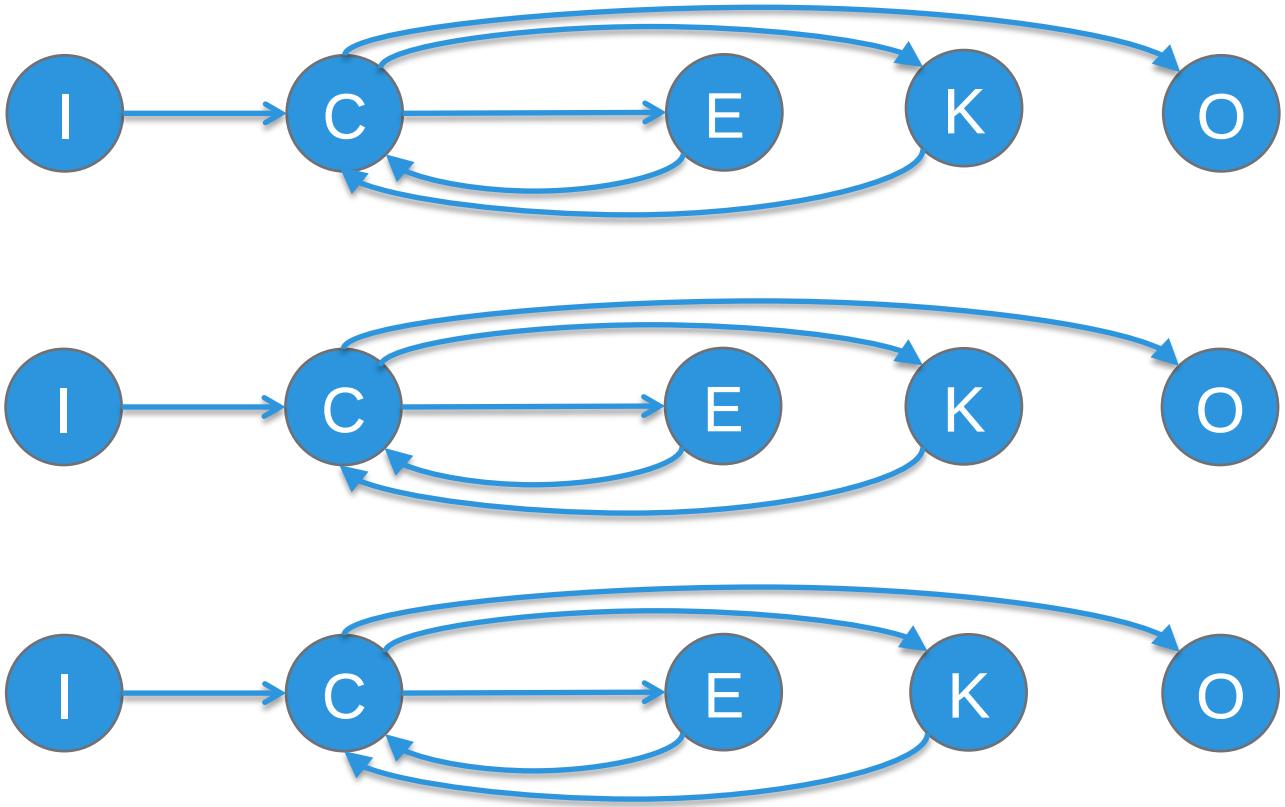
I: Input **C**: Compute **O**: Output

LIFE OF A COMPUTE JOB



I: Input **C**: Compute **E**: Exchange **O**: Output

LIFE OF A COMPUTE JOB



I: Input **C**: Compute **E**: Exchange K: Checkpoint **O**: Output

HEY! WHAT ABOUT RESTART?!?!

On the Non-Suitability of Non-Volatility

John Bent*† Brad Settlemyer†‡ Nathan DeBardeleben†‡
Sorin Faibish* Uday Gupta* Dennis Ting* Percy Tzelnic*

Abstract

For many emerging and existing architectures, NAND flash is the storage media used to fill the cost-performance gap between DRAM and spinning disk. However, while NAND flash is the best of the available options, for many workloads its specific design choices and trade-offs are not wholly suitable. One such workload is long-running scientific applications which use checkpoint-restart for failure recovery. For these workloads, HPC data centers are deploying NAND flash as a storage acceleration tier, commonly called burst buffers, to provide high levels of write bandwidth for checkpoint storage. In this paper, we compare the costs of adding reliability to such a layer versus the benefits of not doing so. We find that, even though NAND flash is non-volatile, HPC burst buffers should not be reliable when the performance overhead of adding reliability is greater than 2%.

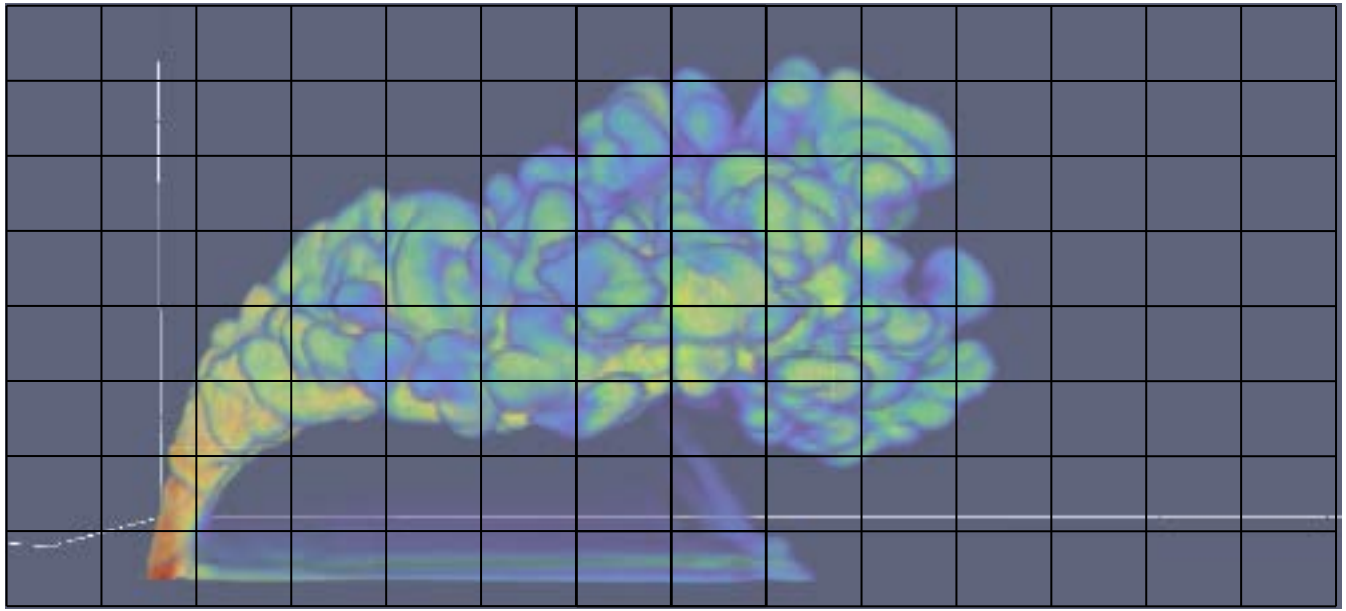
Compute Nodes	100K	–
Compute Cores	1B	–
Checkpoint Data	1EB / day	–
CN:BB Ratio	100:1	–
Burst Time	5 min	CK_{Unr}
Drain Time	60 min	D
Compute Phase	60 min	CO
MTBI	24 hours	N

Table 1: **Exascale Projections.** *Values taken from the Department of Energy Exascale Initative Steering Committee's roadmap to exascale[4] and the variables we use to represent some of them.*

faster than the same access pattern on spinning disks) [11]. Due to the non-volatile storage property of the flash memory within the storage acceleration tier, HPC burst buffers appear to fit as a part of the storage hierarchy as opposed to a part

HotStorage 2015

EMC²



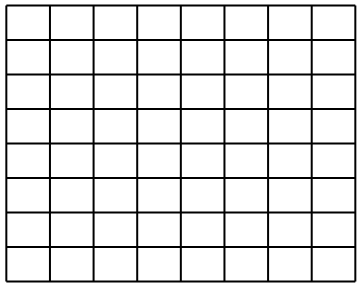
REPARTITIONING WORK IS HARD

We have taken serious measures (under synchronous io) to ensure great load-balancing. However in multiphysics scenarios (e.g. Monte Carlo radiation calculation in fire scenarios, or what is essentially lagrangian particle tracking for wind turbine blade elements) we can turn the difficult algorithm-disparity, load balancing nightmare into a benefit, by dynamically allowing those types of tasks to be performed by the fastest arriving cores and providing some overlap time for slow arriving cores to catch up. Essentially we could create the same types of imbalance we struggle so hard to circumvent now.

– LANL scientist.

MULTI-DIMENSIONAL MESHES

- Spatial meshes in 1-D, 2-D, & 3-D for finite-difference/finite-element numerics



Brick mesh (Cartesian, cylindrical, spherical)

REPARTITIONING WORK IS GETTING HARDER

U.S. DEPARTMENT OF ENERGY

Potential System Architecture Targets

System attributes	2010	"2015"		"2018"	
System peak	2 Peta	200 Petaflop/sec		1 Exaflop/sec	
Power	6 MW	15 MW		20 MW	
System memory	0.3 PB	5 PB		32-64 PB	
Node performance	125 GF	0.5 TF	7 TF	1 TF	10 TF
Node memory BW	25 GB/s	0.1 TB/sec	1 TB/sec	0.4 TB/sec	4 TB/sec
Node concurrency	12	O(100)	O(1,000)	O(1,000)	O(10,000)
System size (nodes)	18,700	50,000	5,000	1,000,000	100,000
Total Node Interconnect BW	1.5 GB/s	20 GB/sec		200 GB/sec	
MTTI	days	O(1day)		O(1 day)	

Exascale Initiative Steering Committee

Slide 11

ALTERNATIVE APPROACH: ASYNC COMPUTE

Automatic and Transparent I/O Optimization With Storage Integrated Application Runtime Support

Noah Watkins
UC Santa Cruz
jayhawk@soe.ucsc.edu

Zhihao Jia
Stanford University
zhihao@cs.stanford.edu

Galen Shipman
LANL
gshipman@lanl.gov

Carlos Maltzahn
UC Santa Cruz
carlosm@soe.ucsc.edu

Alex Aiken
Stanford University
aiken@cs.stanford.edu

Pat McCormick
LANL
pat@lanl.gov

ABSTRACT

Traditionally storage has not been part of a programming model's semantics and is added only as an I/O library interface. As a result, programming models, languages, and storage systems are limited in the optimizations they can perform for I/O operations, as the semantics of the I/O library is typically at the level of transfers of blocks of uninterpreted bits, with no accompanying knowledge of how those bits are used by the application. For many HPC applications where I/O operations for analyzing and checkpointing large data sets are a non-negligible portion of the overall execution time, such a "know nothing" I/O design has negative performance implications.

We propose an alternative design where the I/O semantics are integrated as part of the programming model, and a common data model is used throughout the entire memory and storage hierarchy enabling storage and application level co-optimizations. We demonstrate these ideas through the integration of storage services within the Legion [2] runtime and present preliminary results demonstrating the integration.

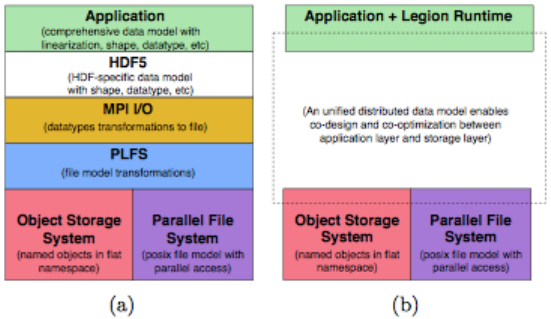


Figure 1: (a) In a contemporary I/O stack each layer uses a distinct data model. (b) Our proposed architecture uses a unified data model and run-time to enable system-wide co-design and co-optimization strategies

HotStorage 2015

SAVING DISTRIBUTED STATE IS HARD

Operating Systems — R. Stockton Gaines, Editor

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.

The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed algorithm is given for synchronizing a system of logical clocks which can be used to totally order the events. The use of the total ordering is illustrated with a method for solving synchronization problems. The algorithm is then specialized for synchronizing physical clocks, and a bound is derived on how far out of synchrony the clocks can become.

Key Words and Phrases: distributed systems, computer networks, clock synchronization, multiprocess systems

CR Categories: 4.32, 5.29

Fig. 1.

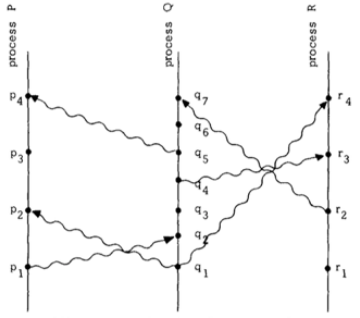
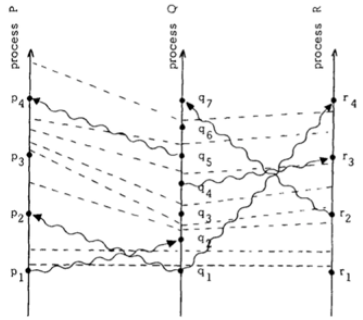


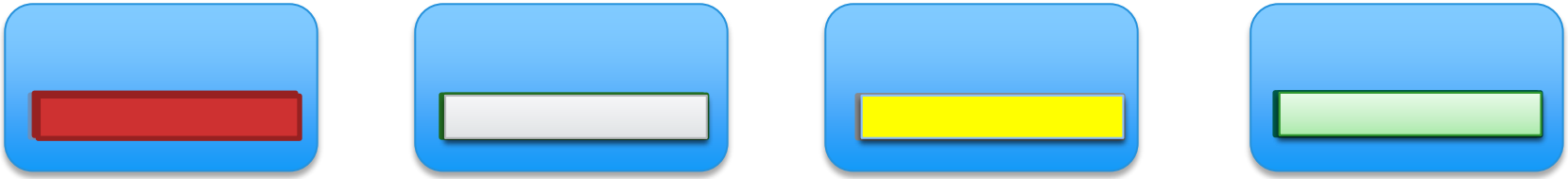
Fig. 2.

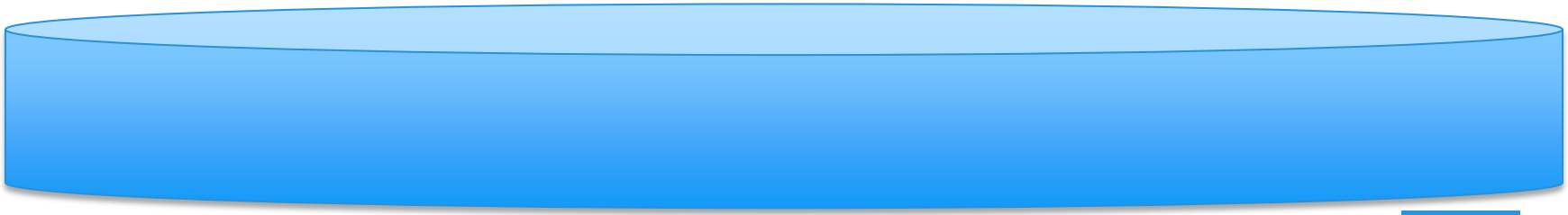


EMC²

SYNCHRONOUS STATE CAPTURE

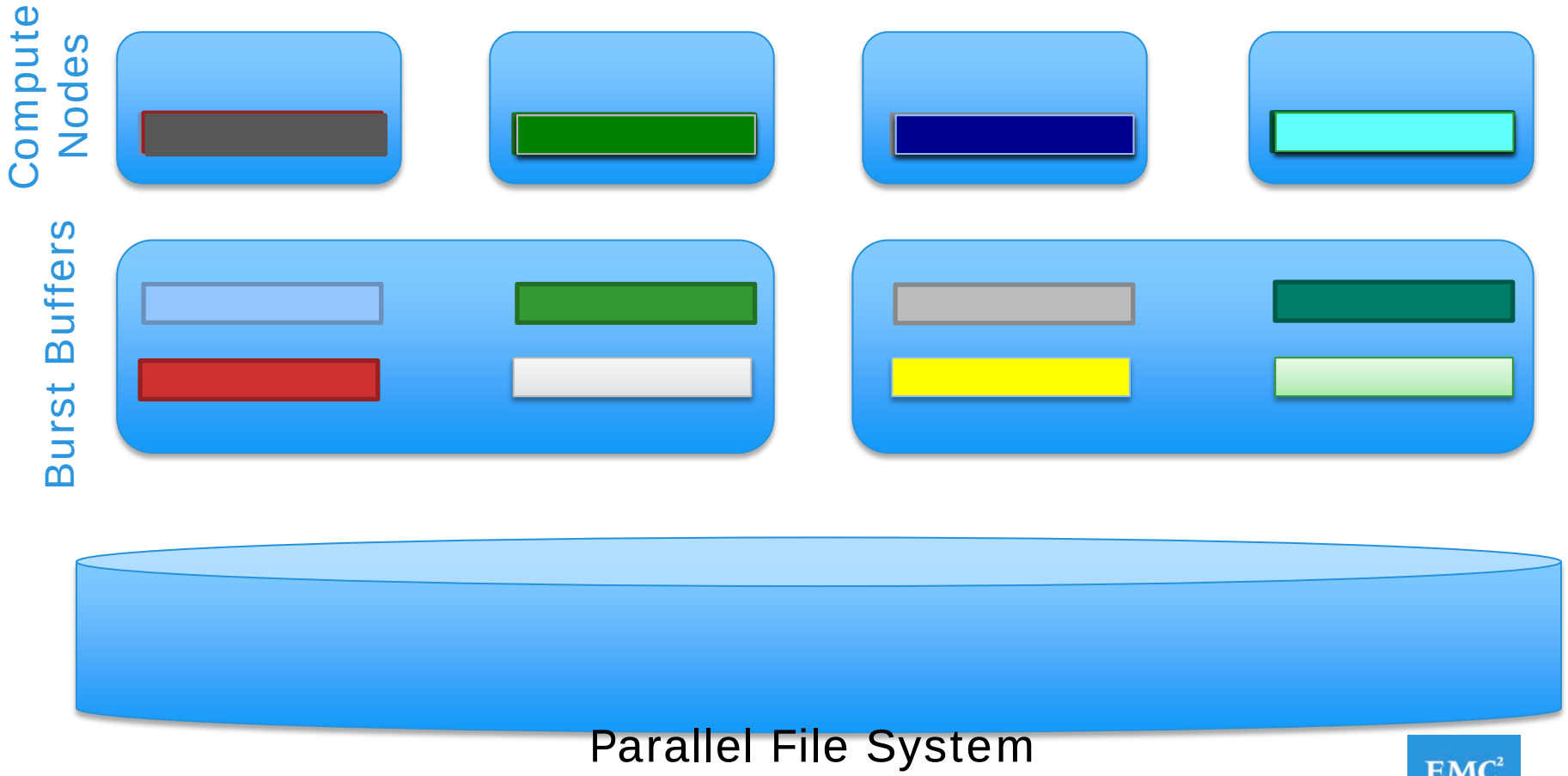
Compute Nodes





Parallel File System

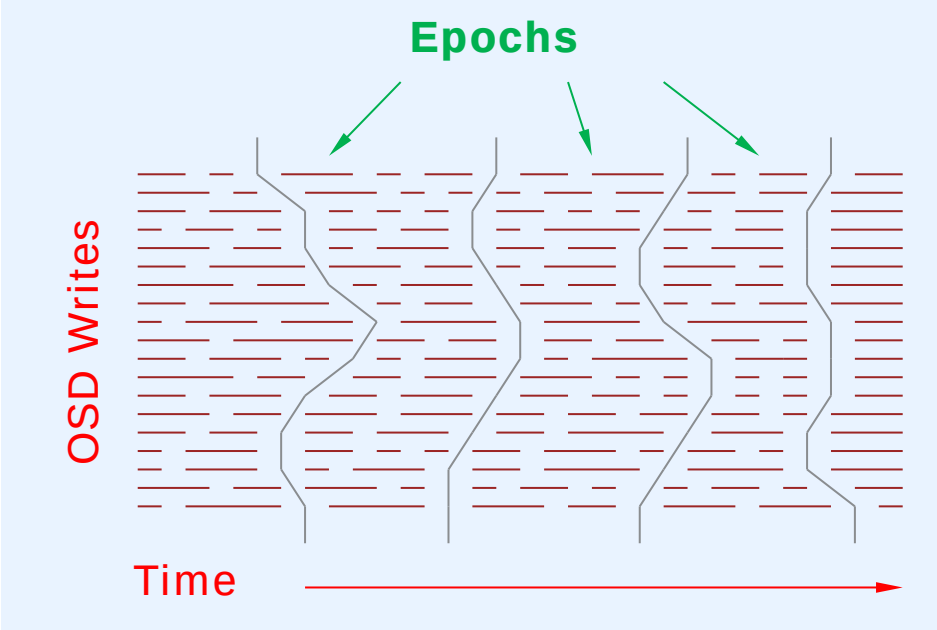
SYNCHRONOUS STATE CAPTURE WITH B.B.



ASYNC COMPUTE NEEDS ASYNC CHECKPOINT

- BAD Check is one such instance
  - Formerly known as IOD from Intel-EMC-HDF FastForward
- Move synchronization from app to storage
- Transactions
  - Capture synchronous views of data at asynchronous moments of wall-clock
- Reference counting
  - Storage ensures consistent and complete transaction

Transactions Mark Synchronous Virtual Views in Asynchronous Physical Time



[This is actually Eric Barton's picture of epochs in DAOS which are very similar to transactions in Bad Check. In fact, there is a 1:1 mapping between the two.]

Asynchronous Distributed Transactions

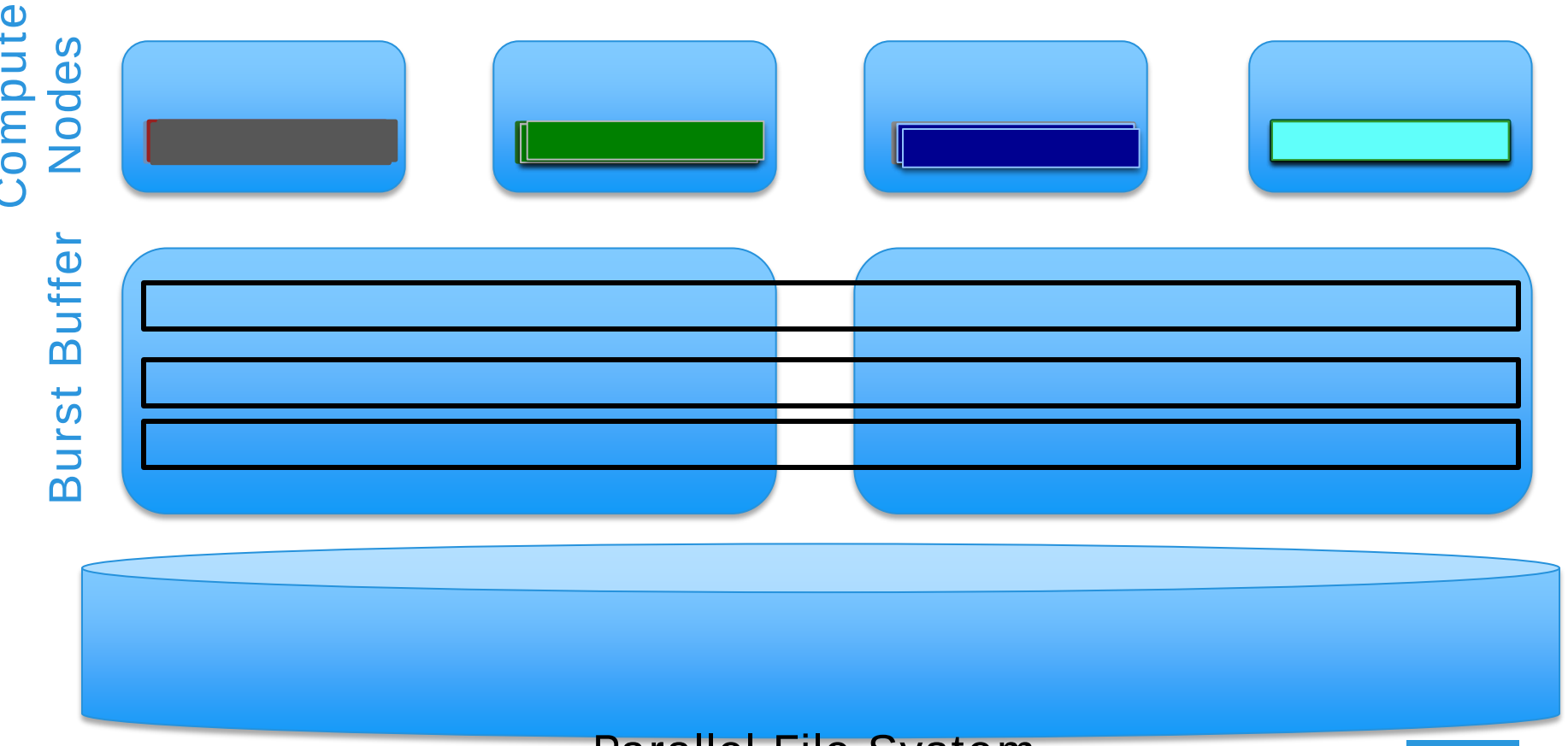
```
MPI_Init(...)
iod_init(mpicomm);
iod_container_create(...);

for( timeseries T=0; ! isdone(T); T++ ) {
    compute();
    iod_start_trans(trans=T, siblings=S);
    /* MPI_Barrier(mpicomm); */
    iod_obj_write_all(...);
    iod_end_trans(trans=T); /* asynchronous */
}
```

Look Mom, no barriers!

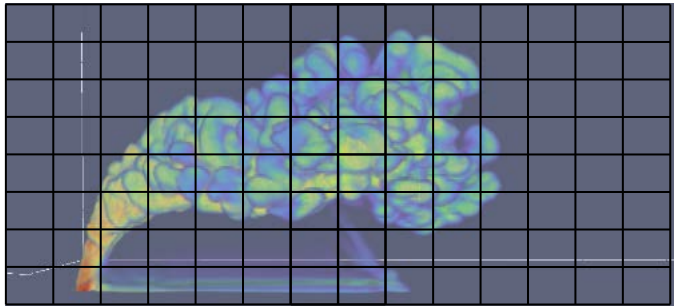
SYNCHRONOUS / ASYCHRONOUS

TYPICAL COORDINATED CHECKPOINTING

DISTRIBUTED ASYNCHRONOUS TRANSACTIONS

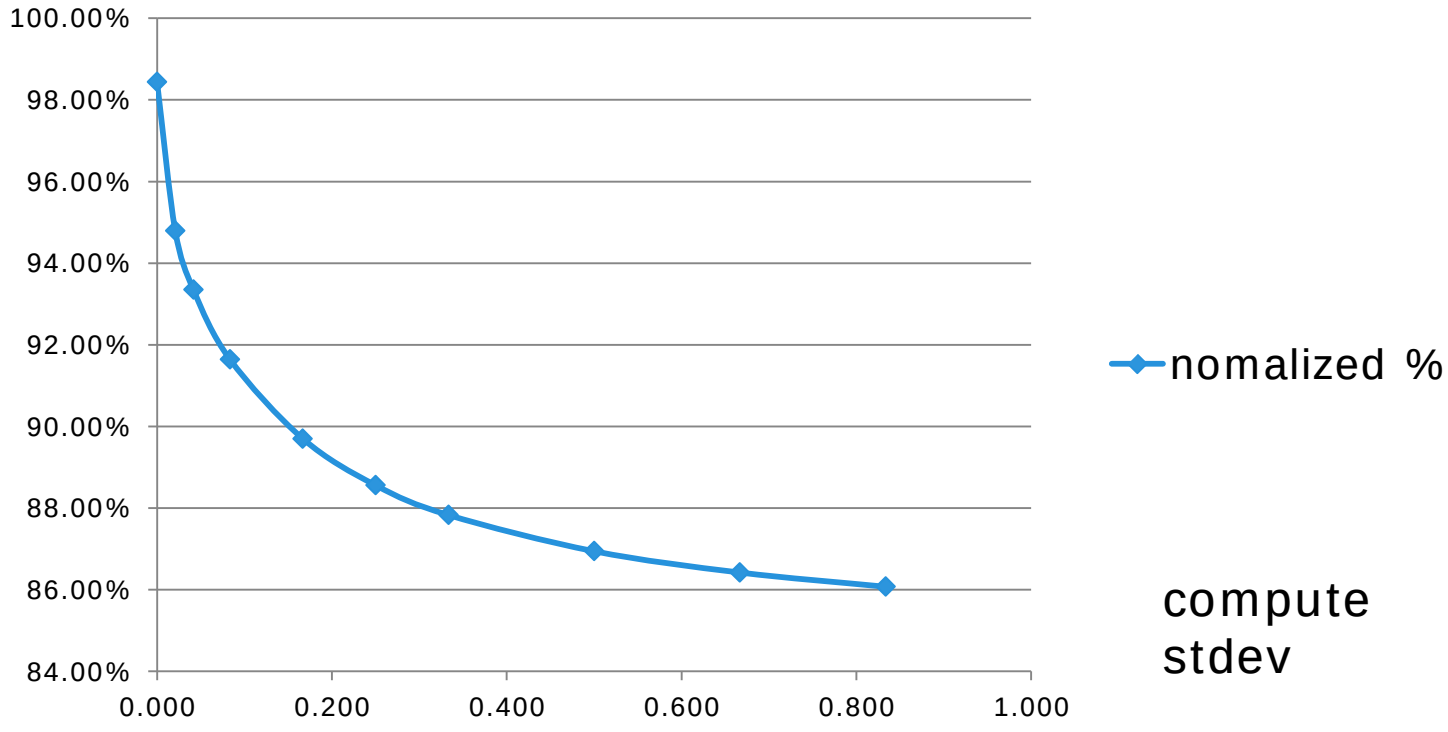


BENEFITS OF BULK-ASYNCHRONOUS



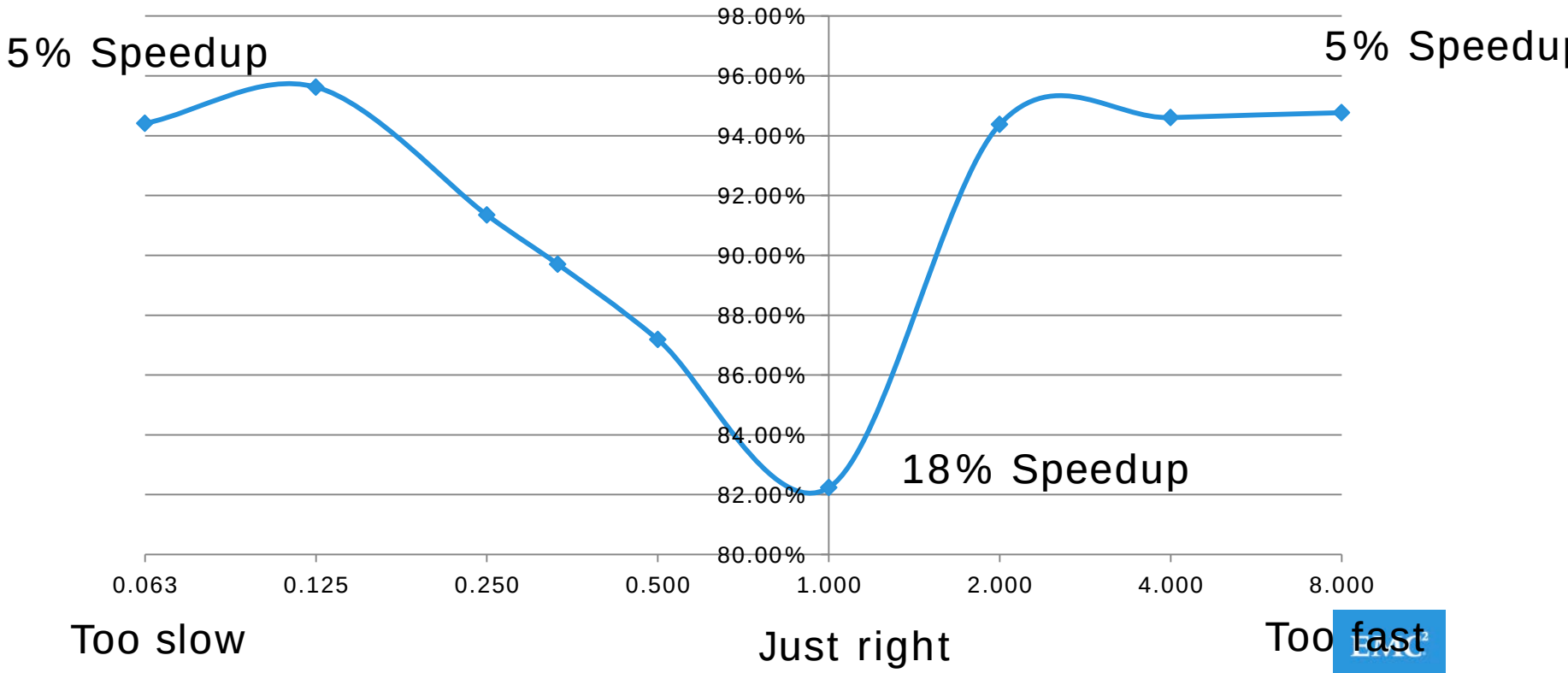
- Performance
  - Faster wall-clock runtimes
    - *When* there are hotspots
    - *When* hotspots move
    - *When* communications are not global
  - Vacate fast nodes early; prestage next job
- Cost: lower storage bandwidth requirements
- Usability
  - Versioning
  - Time-series analysis
  - Uncoordinated producer-consumer

HEAT OF HOTSPOTS

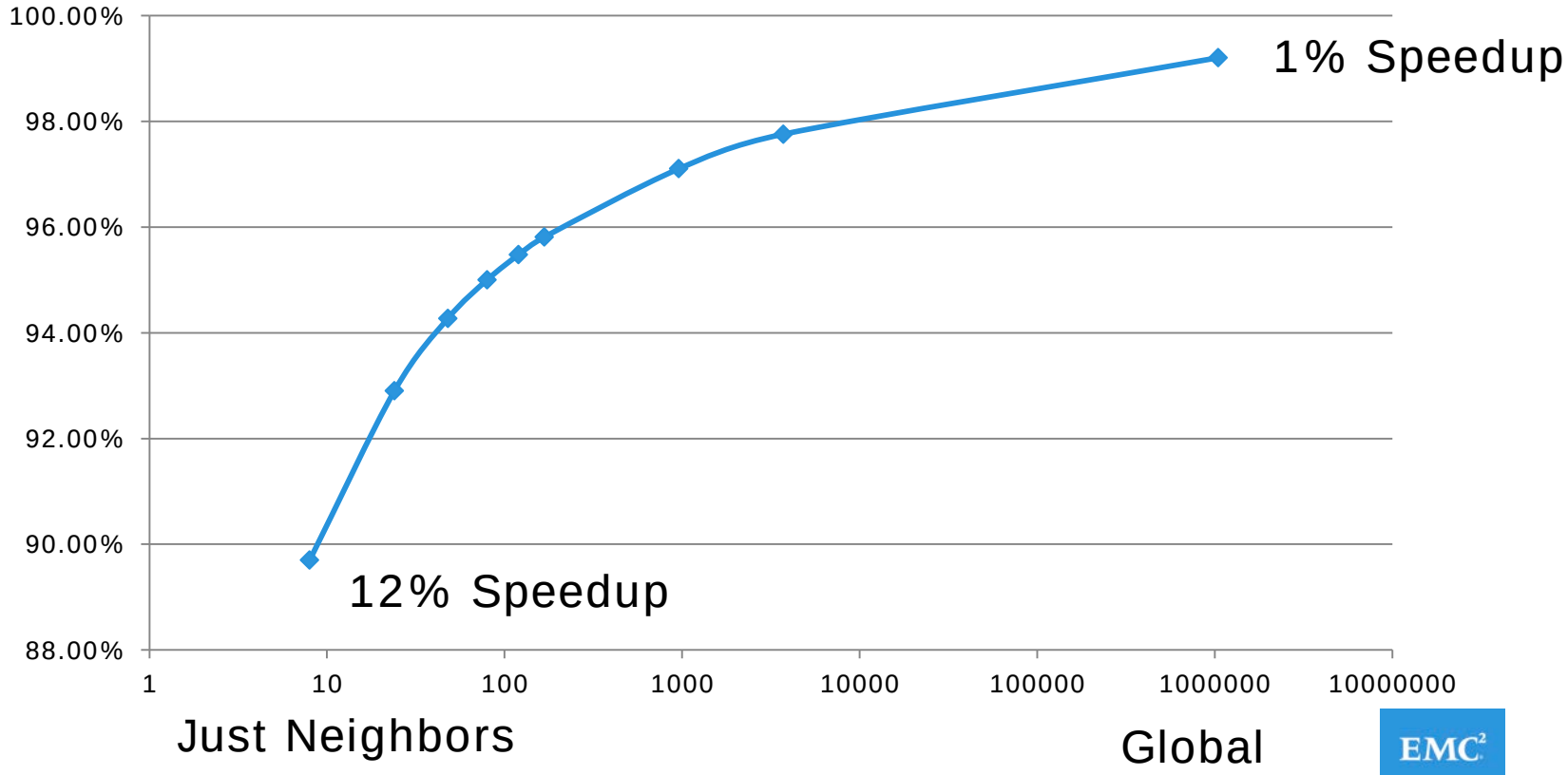


compute
stdev

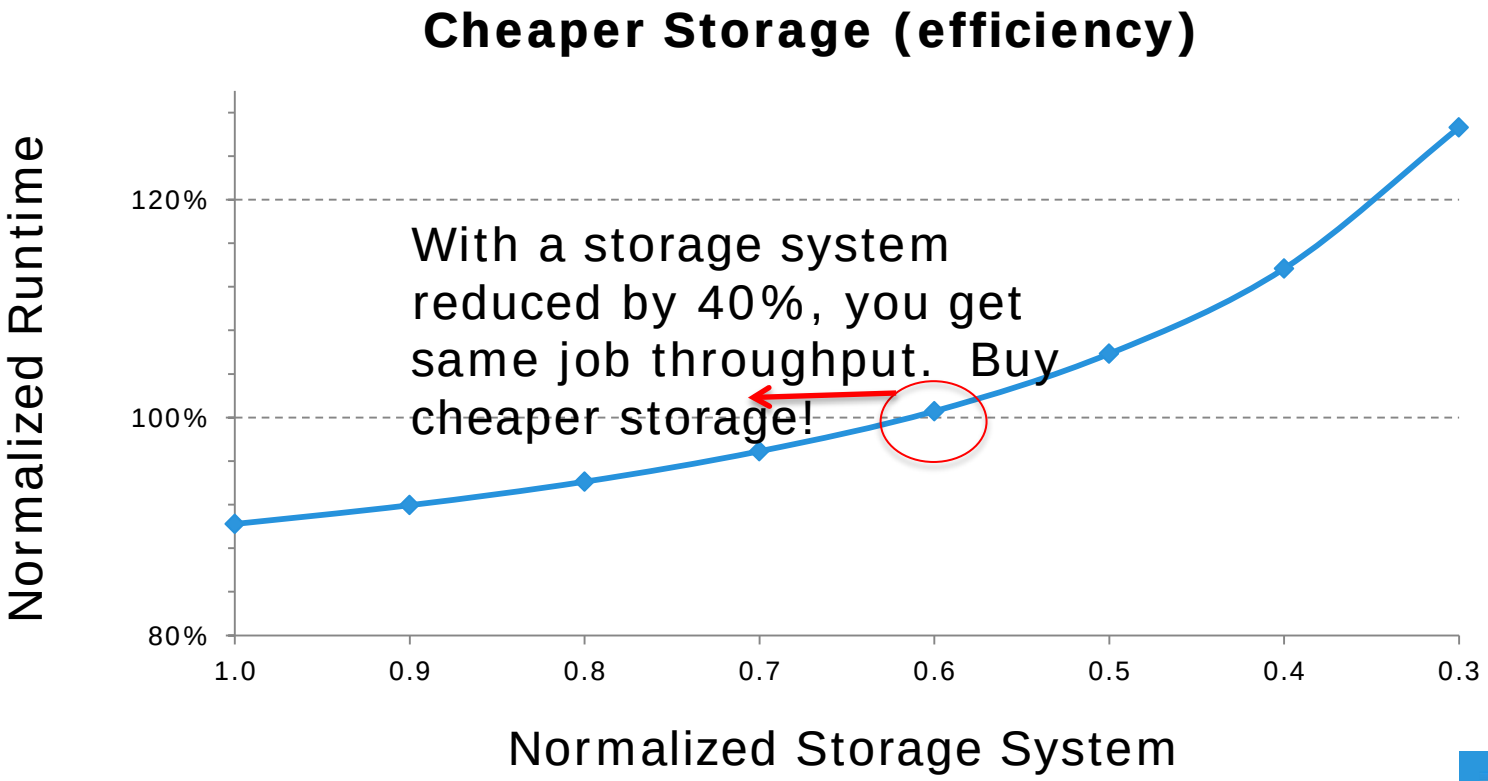
HOTSPOT MOVEMENT



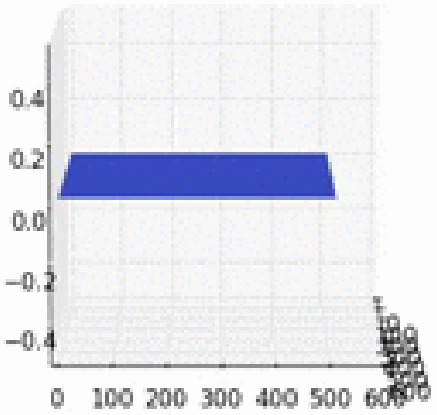
COMMUNICATIONS



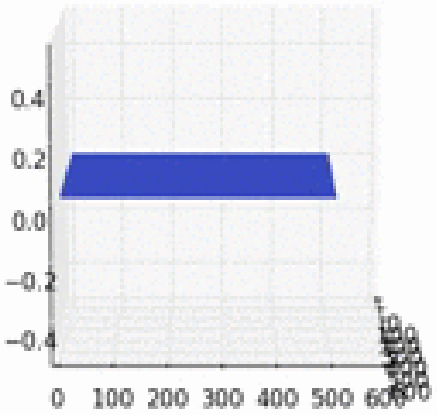
STORAGE COST REDUCTION



THROUGHPUT IMPROVEMENTS



Bulk Asynchronous



Bulk Synchronous

BAD CHECK IS GOOD . . . WHEN

1. Hot spots exists
2. Hot spots move
   - Not too slowly
   - Nor too quickly
3. Communication is not global
4. Failure detection is reliable
   - E.g. storage provides reliable transactions

Asynchrony is coming; do not be afraid.

Asynchrony is coming; do not be afraid.

john.bent@emc.com