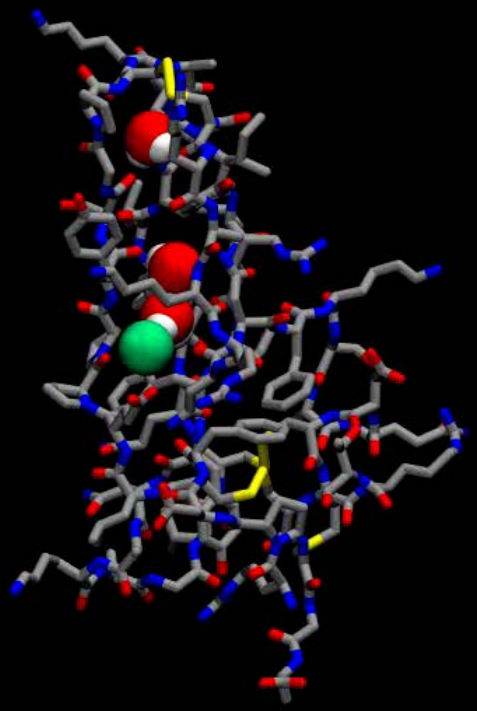
Accelerating Parallel Analysis of Scientific Simulation Data via Zazen

Tiankai Tu, Charles A. Rendleman, Patrick J. Miller, Federico Sacerdoti, Ron O. Dror, and David E. Shaw

D. E. Shaw Research

Motivation

- Goal: To model biological processes that occur on the millisecond time scale
- Approach: A specialized, massively parallel super-computer called *Anton* (2009 ACM Gordon Bell Award for Special Achievement)



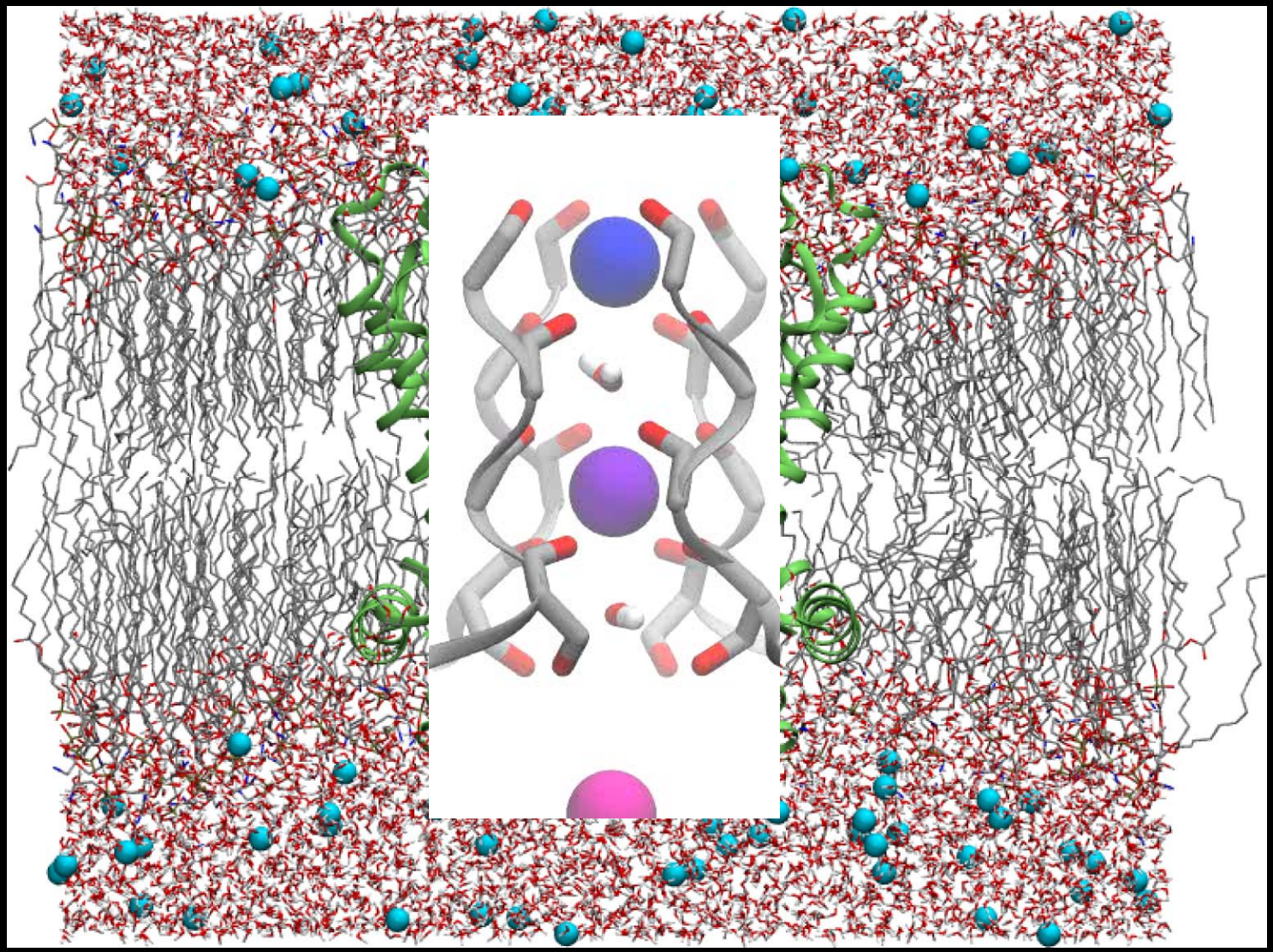
Millisecond-scale MD Trajectories

	A biomolecular system:	25 K atoms
×	Position and velocity:	24 bytes/atom
	Frame size:	0.6 MB/frame

	Simulation length:	1×10^{-3} s
÷	Output interval:	10×10^{-12} s
	Number of frames:	100 M frames

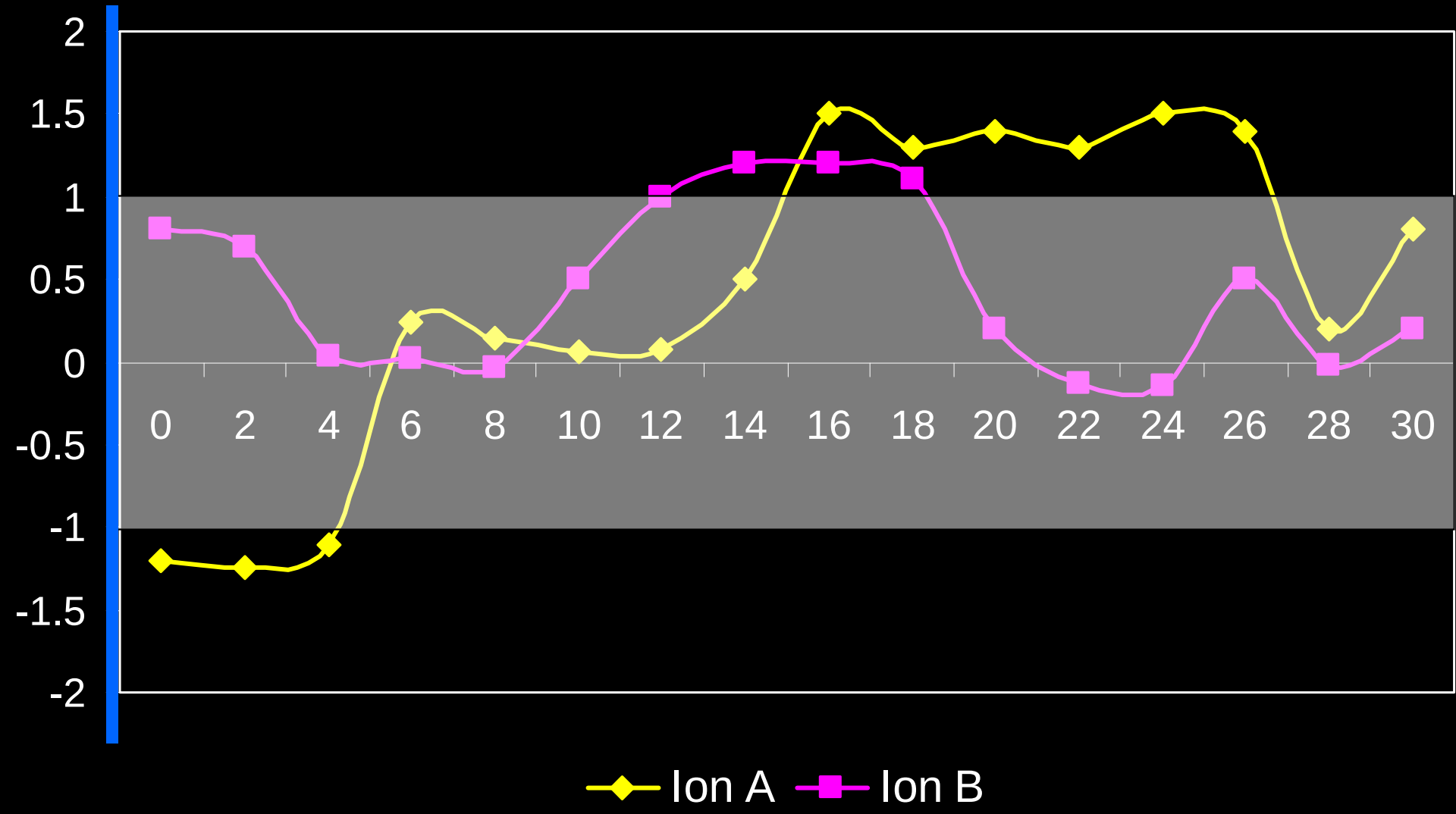
Part I: How We Analyze Simulation Data in Parallel

An MD Trajectory Analysis Example: Ion Permeation

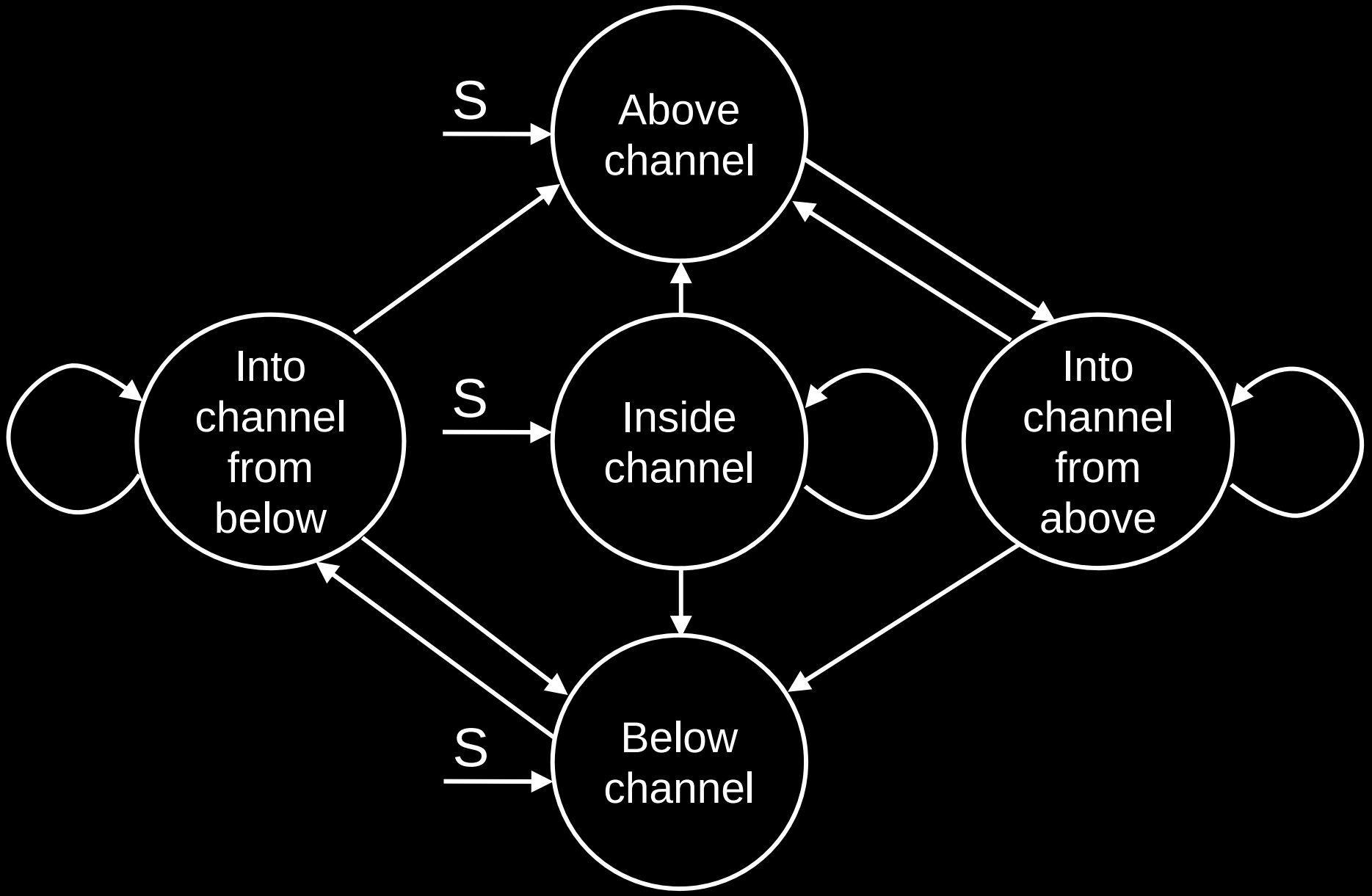


A Hypothetic Trajectory

20,000 atoms in total; two ions of interest



Ion State Transition



Typical Sequential Analysis

- Maintain a main-memory resident data structure to record states and positions
- Process frames in ascending simulated physical time order

Strong inter-frame data dependence:
Data analysis tightly coupled with data acquisition

Problems with Sequential Analysis

Millisecond-scale trajectory size : 60 TB

Local disk read bandwidth : 100 MB / s

Time to fetch data to memory : 1 week

Analysis time : Varied

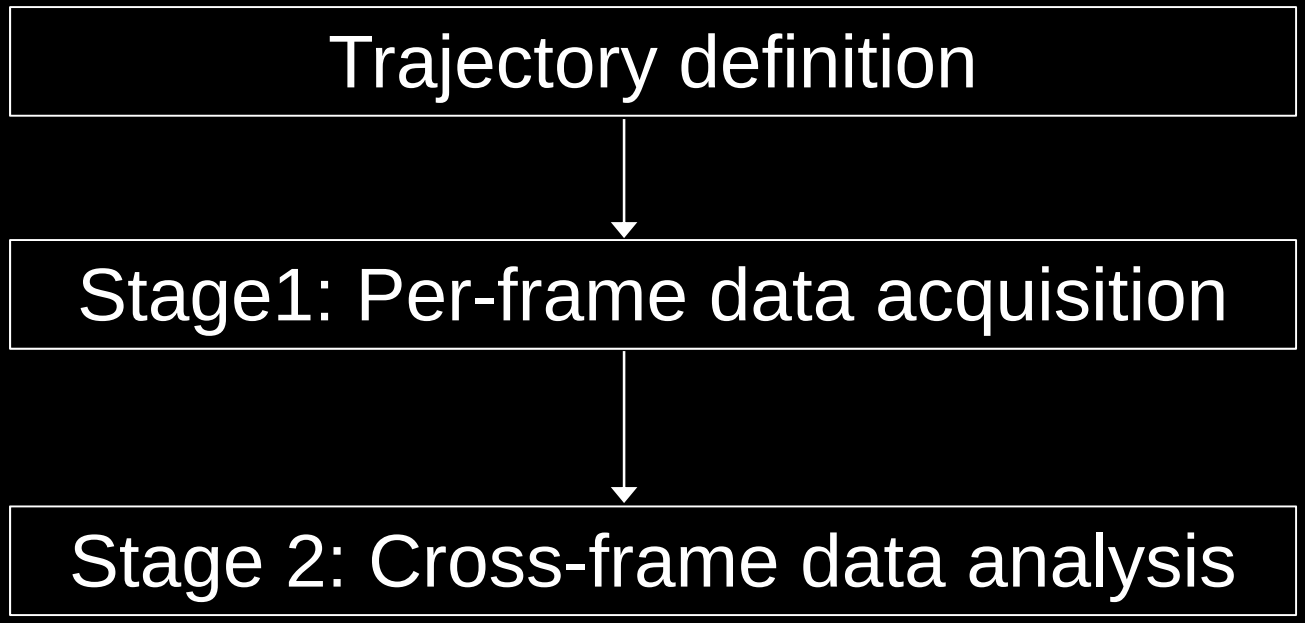
Time to perform data analysis : Weeks

Sequential analysis lack the computational, memory, and I/O capabilities!

A Parallel Data Analysis Model

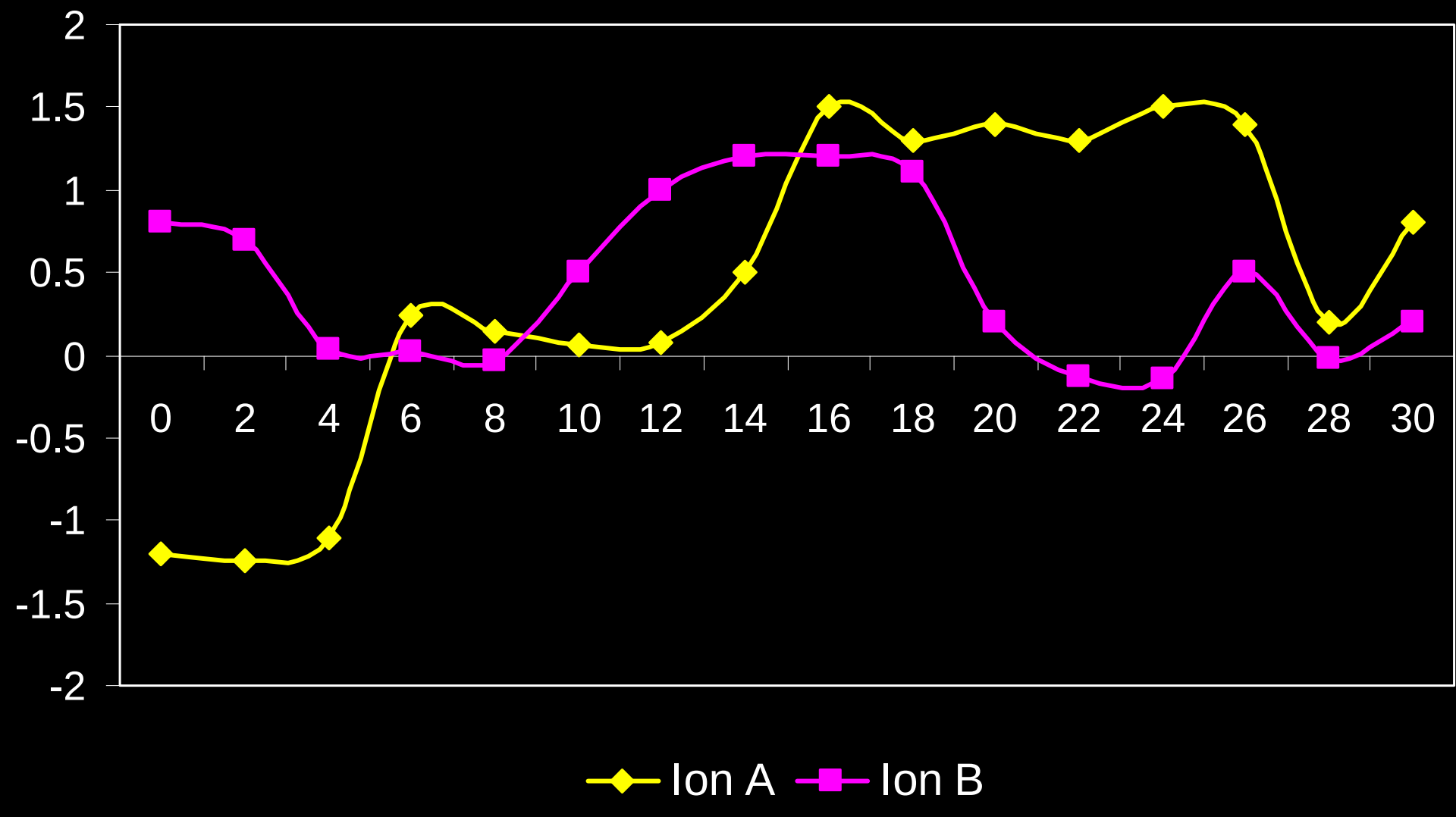
Specify which frames to be accessed

Decouple data acquisition from data analysis

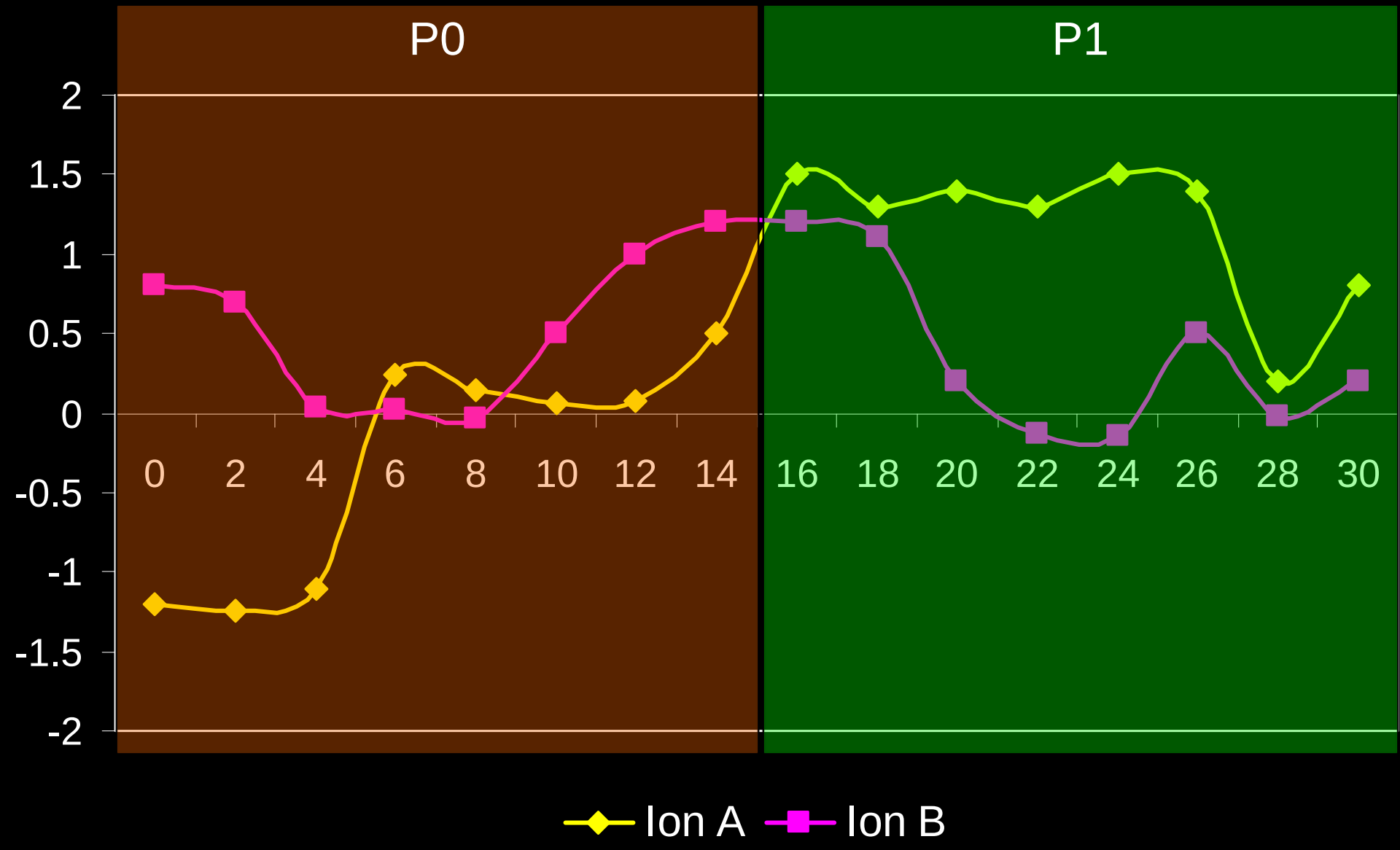


Trajectory Definition

Every other frame in the trajectory

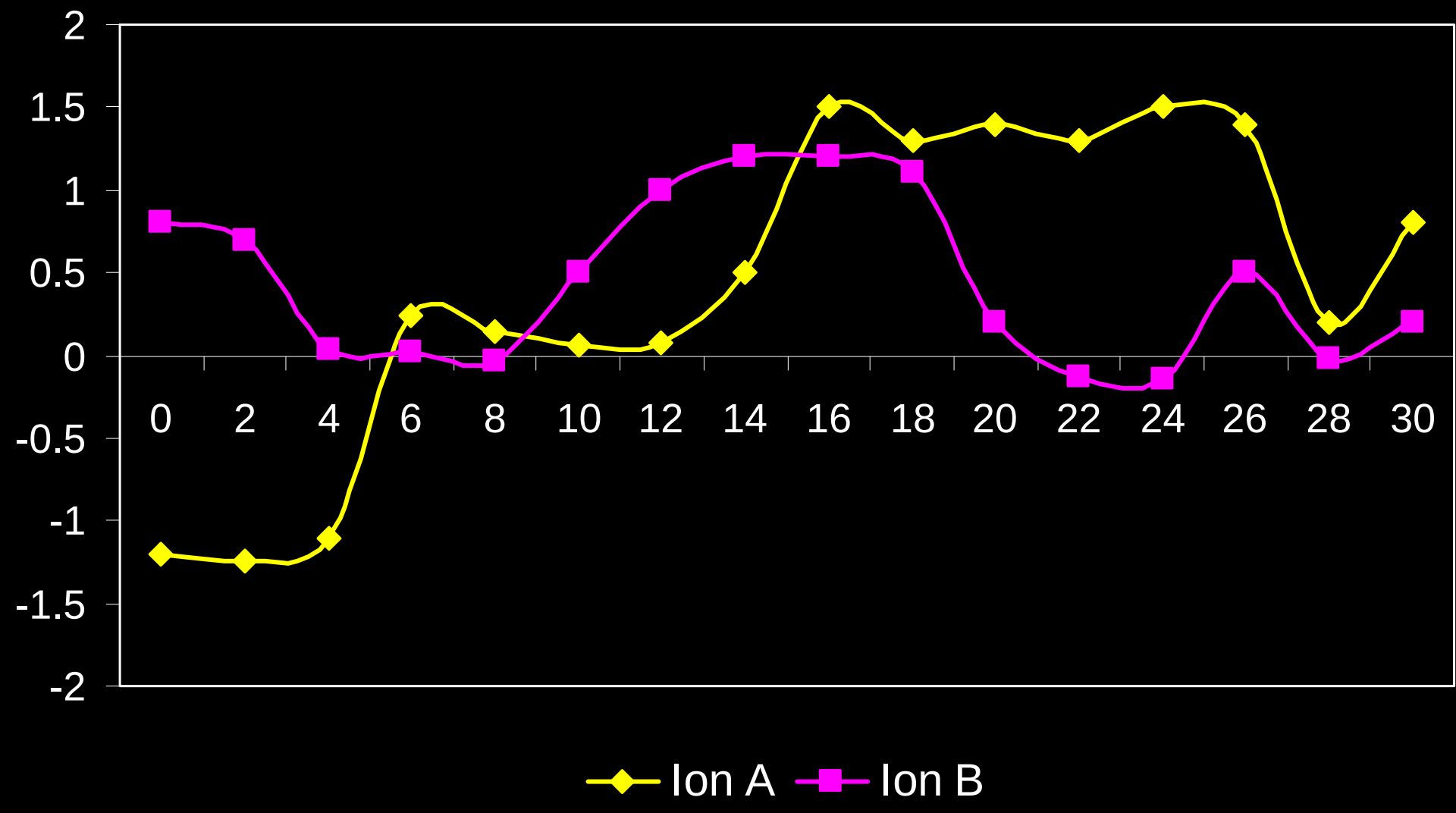


Per-frame Data Acquisition (stage 1)

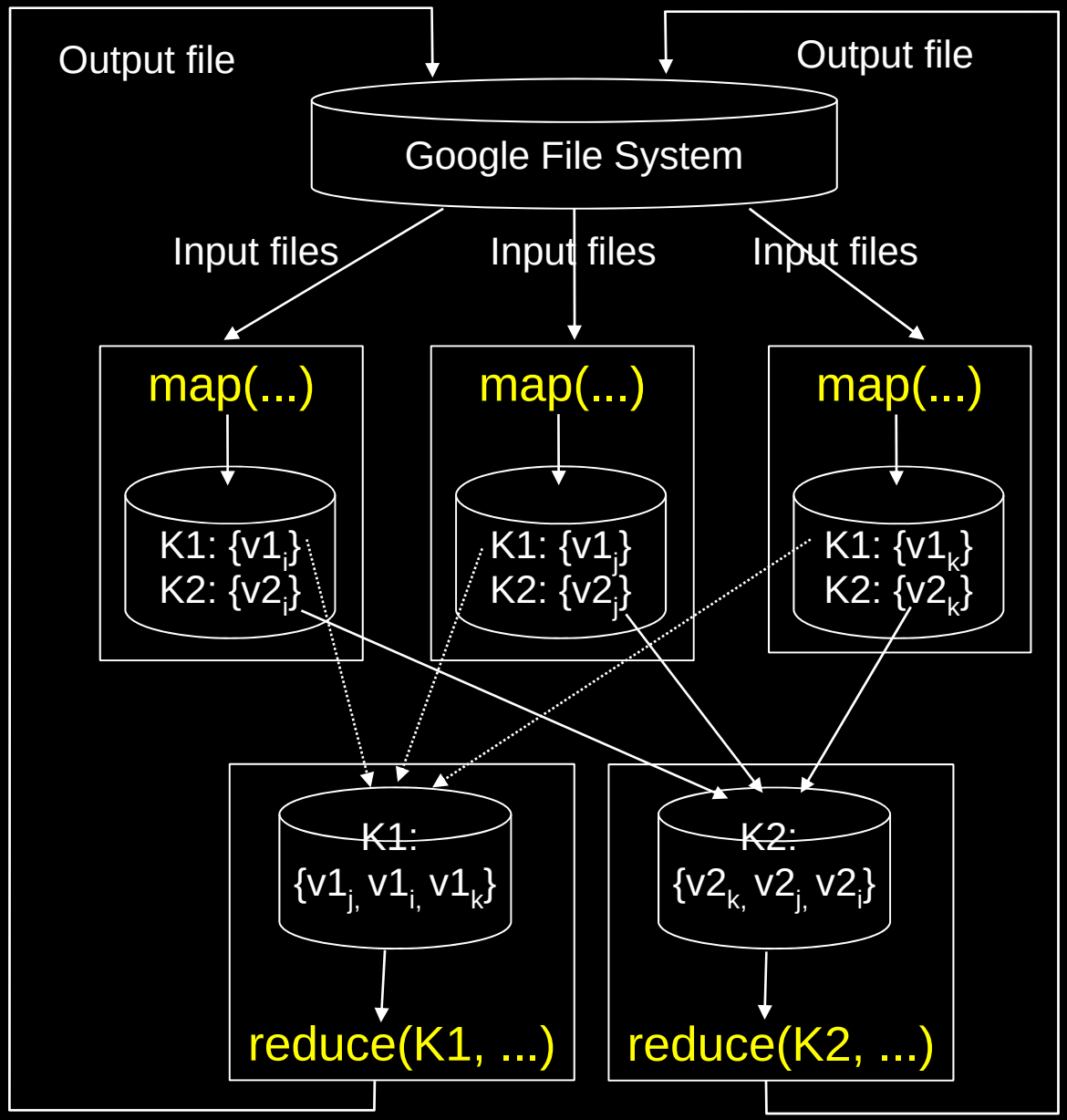


Cross-frame Data Analysis (stage 2)

Analyze ion A on P0 and ion B on P1 in parallel

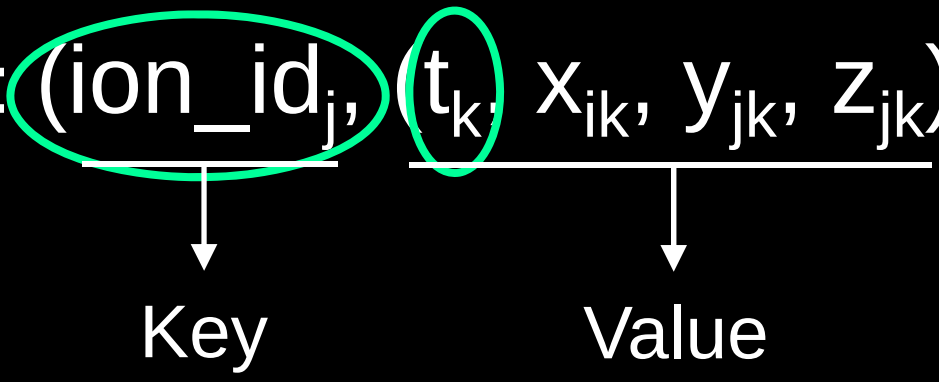


Inspiration: Google's MapReduce



Trajectory Analysis Cast Into MapReduce

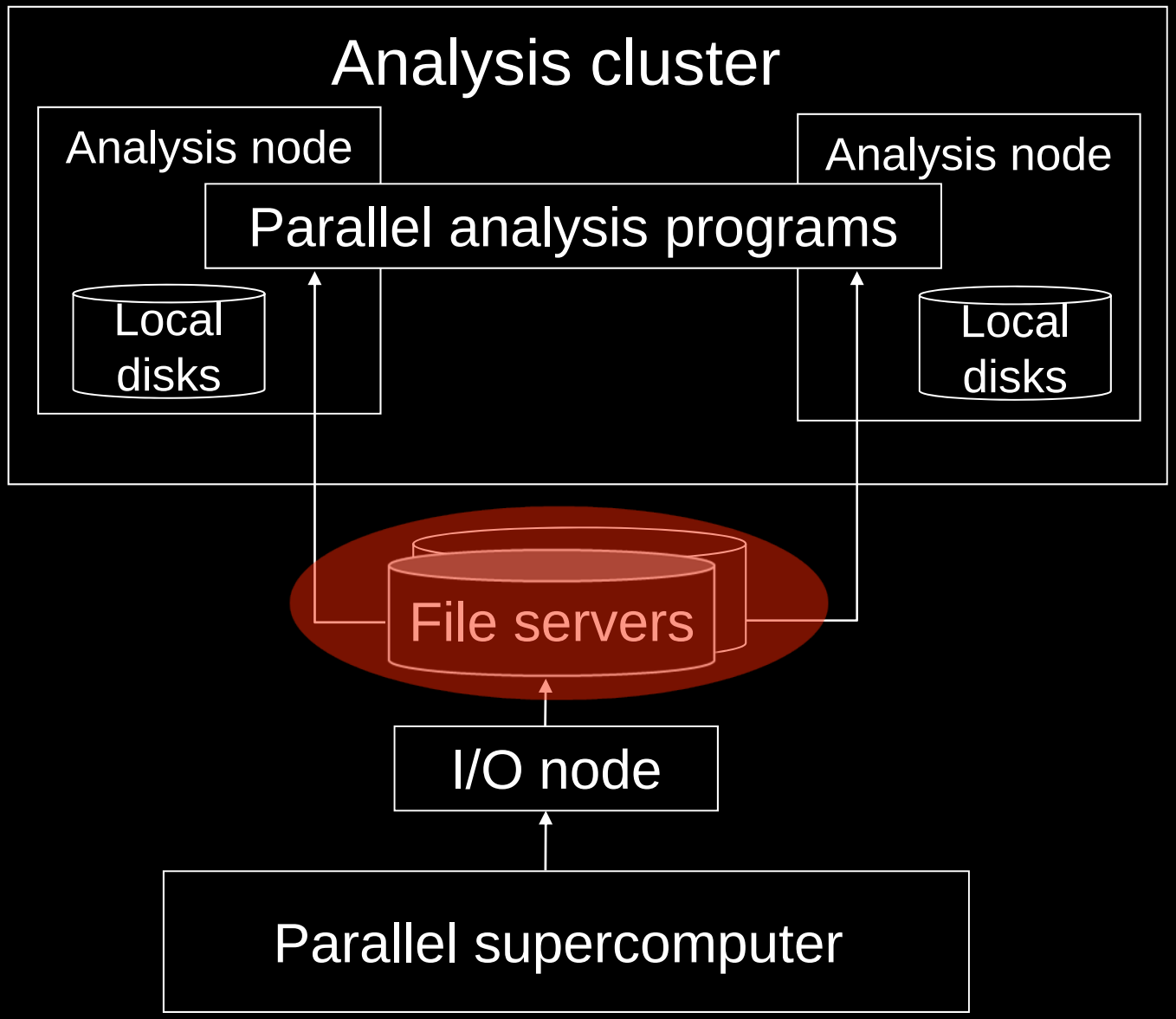
- Per-frame data acquisition (stage 1): map()
- Cross-frame data analysis (stage 2): reduce()
- Key-value pairs: connecting stage1 and stage2
  - Keys: categorical identifiers or names
  - Values: including timestamps
  - Examples: $(ion\_id_j, (t_k, x_{ik}, y_{jk}, z_{jk}))$



The HiMach Library

- A MapReduce-style API that allows users to write Python programs to analyze MD trajectories
- A parallel runtime that executes HiMach user programs in parallel on a Linux cluster automatically
- Performance results on a Linux cluster:
  - 2 orders of magnitude faster on 512 cores than on a single core

Typical Simulation–Analysis Storage Infrastructure



Part II: How We Overcome the I/O Bottleneck in Parallel Analysis

Trajectory Characteristics

- A large number of small frames
- Write once, read many
- Distinguishable by unique integer sequence numbers
- Amenable to out-of-order parallel access in the map phase

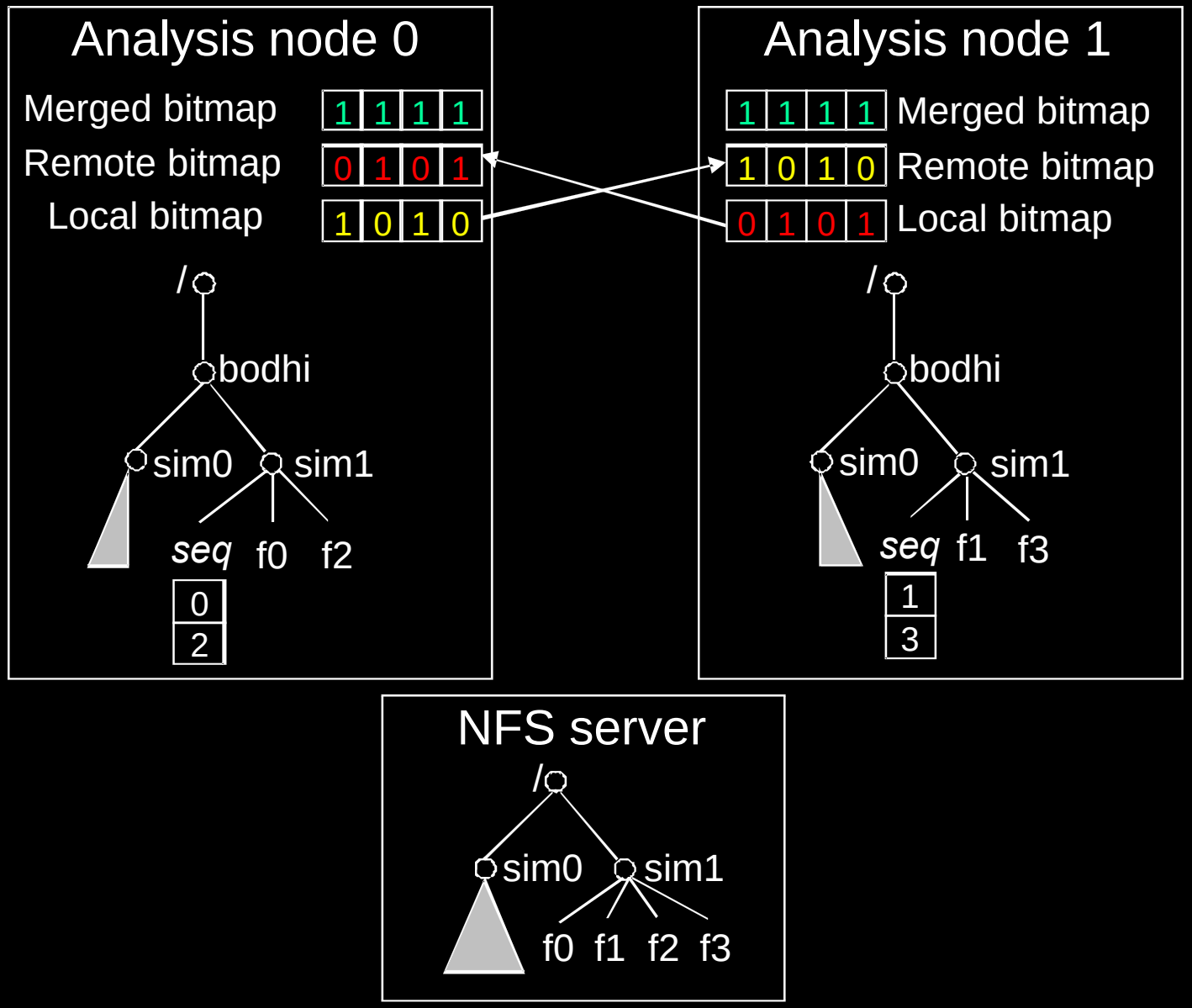
Our Main Idea

- At simulation time, actively cache frames in the local disks of the analysis nodes as the frames become available
- At analysis time, fetch data from local disk caches in parallel

Limitations

- Require large aggregate disk capacity on the analysis cluster
- Assume relatively low average simulation data output rate

An Example


Analysis node 0
Merged bitmap
1 1 1 1
Remote bitmap
0 1 0 1
Local bitmap
1 0 1 0
/
bodhi
sim0
sim1
seq
f0
f2
0
2
Analysis node 1
1 1 1 1
Merged bitmap
1 0 1 0
Remote bitmap
0 1 0 1
Local bitmap
/
bodhi
sim0
sim1
seq
f1
f3
1
3
NFS server
/
sim0
sim1
f0
f1
f2
f3

How to guarantee that each frame is read by one and only one node in the face of node failure and recovery?

The Zazen Protocol

The Zazen Protocol

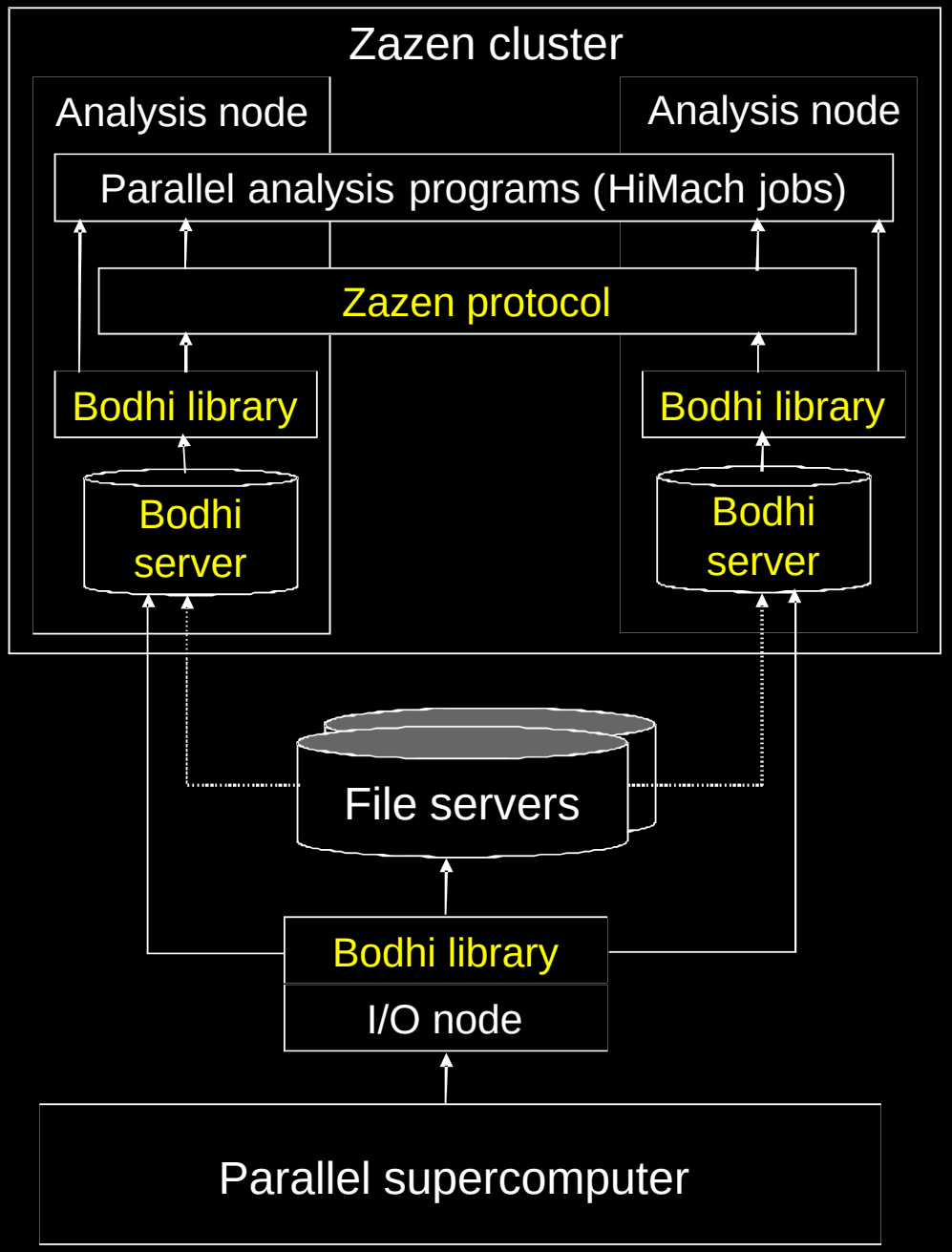
- Execute a distributed consensus protocol before performing actual disk I/O
- Assign data retrieval tasks in a location-aware manner
- Read data from local disks if the data are already cached
- Fetch missing data from file servers
- No metadata servers to keep record of who has what

The Zazen Protocol (cont'd)

- Bitmaps: a compact structure for recording the presence or non-presence of a cached copy
- All-to-all reduction algorithms: an efficient mechanism for inter-processor collective communications (used an MPI library in practice)

Implementation

- The Bodhi library
- The Bodhi server
- The Zazen protocol



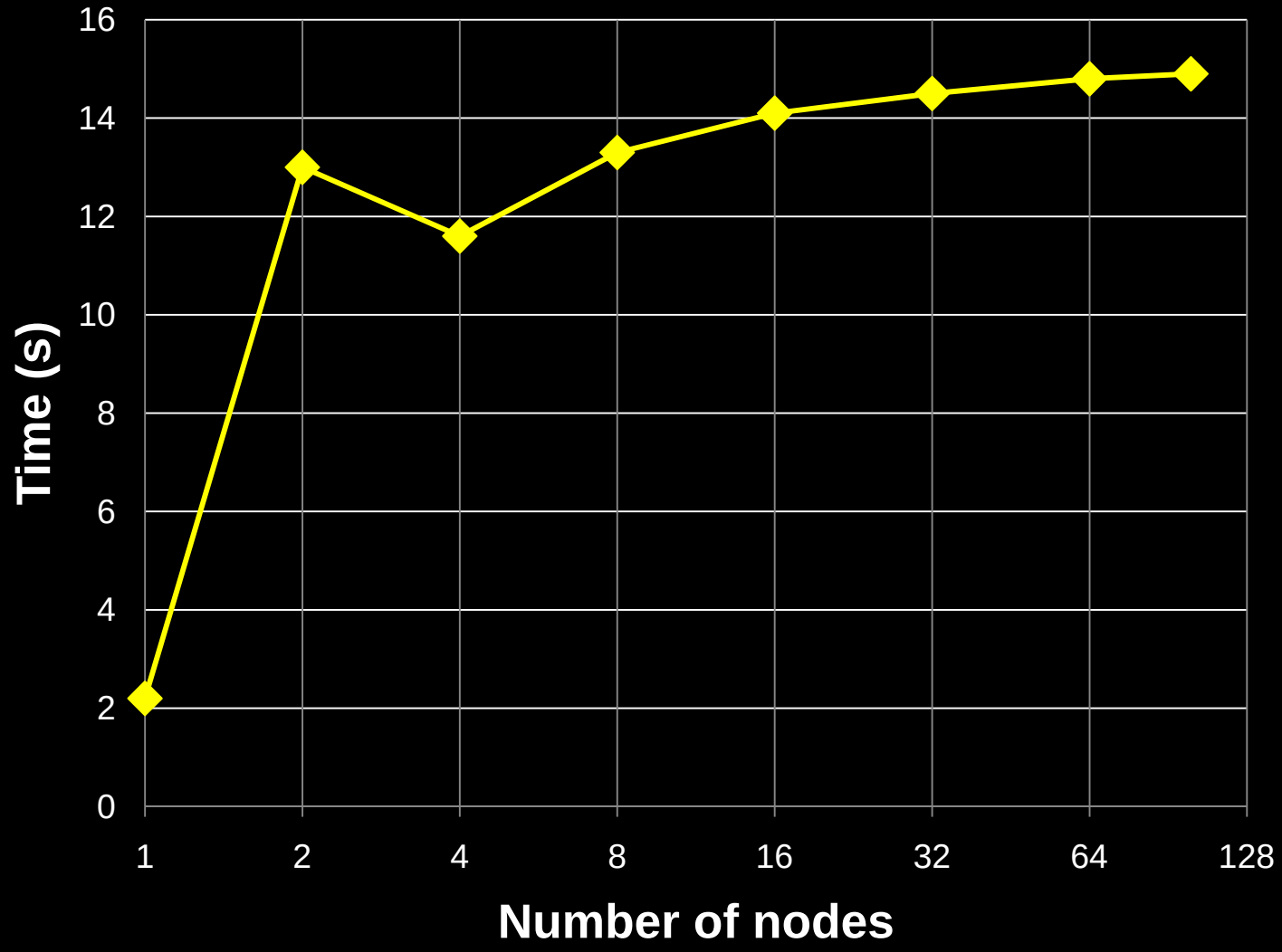
Performance Evaluation

Experiment Setup

- A Linux cluster with 100 nodes
- Two Intel Xeon 2.33 GHz quad-core processors per node
- Four 500 GB 7200-RPM SATA disks organized in RAID 0 per node
- 16 GB physical memory per node
- CentOS 4.6 with a Linux kernel of 2.6.26
- Nodes connected to a Gigabit Ethernet core switch
- Common accesses to NFS directories exported by a number of enterprise storage servers

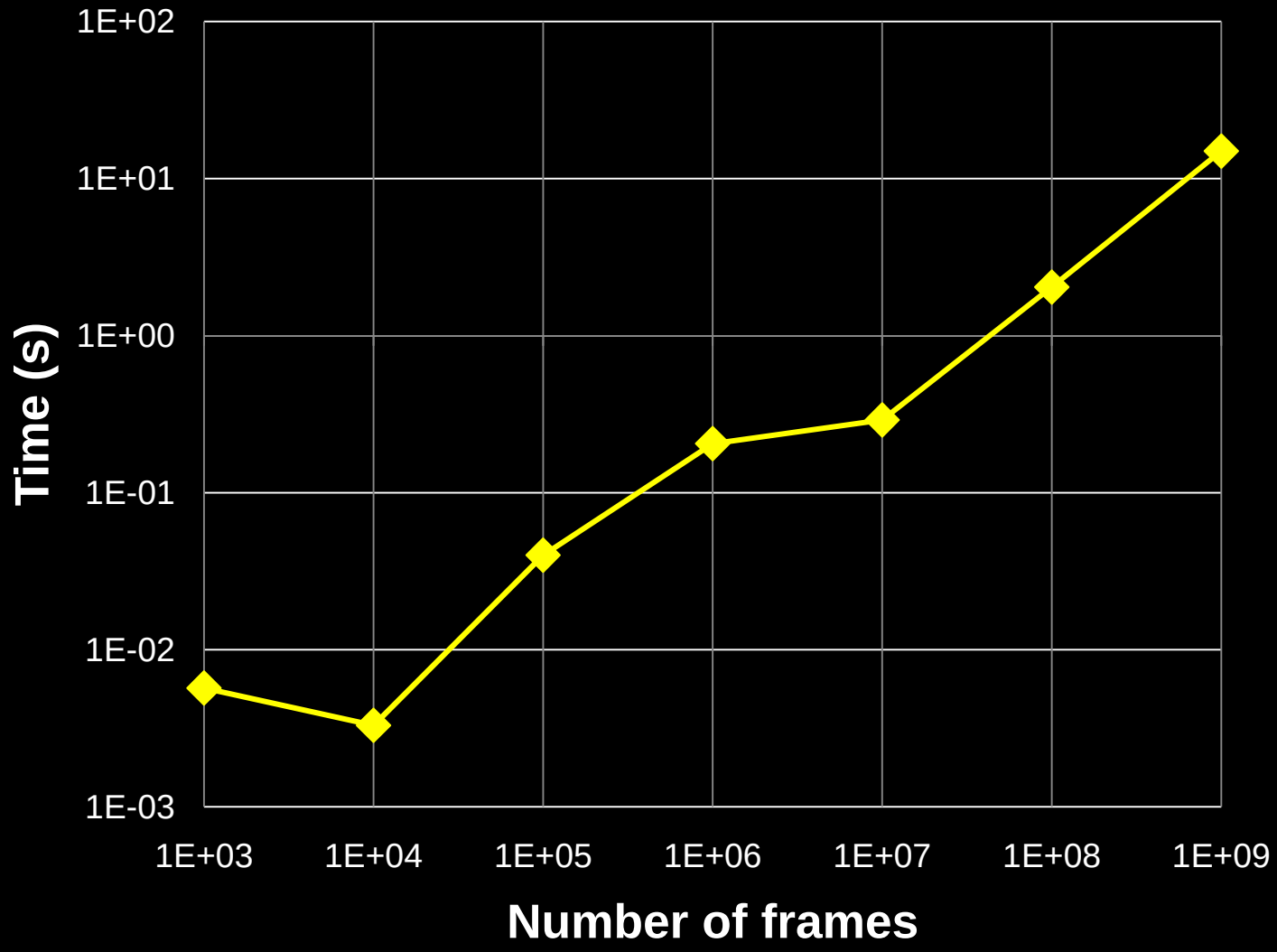
Fixed-Problem-Size Scalability

Execution time of the Zazen protocol to assign the I/O tasks of reading 1 billion frames



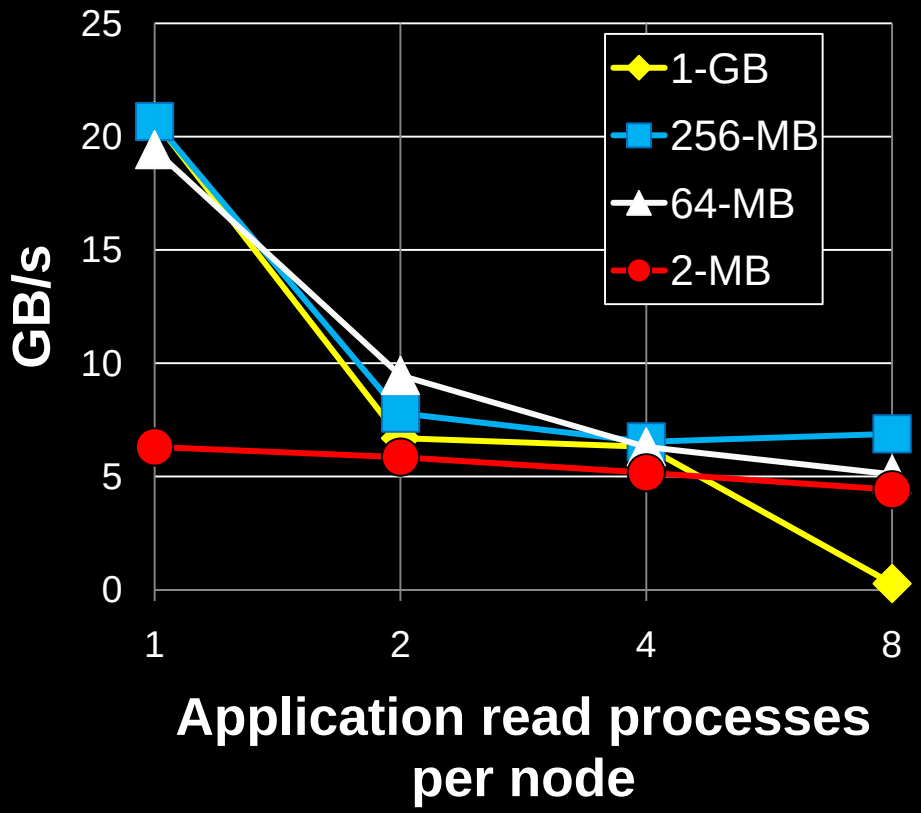
Fixed-Cluster-Size Scalability

Execution time of the Zazen protocol on 100 nodes assigning different number of frames

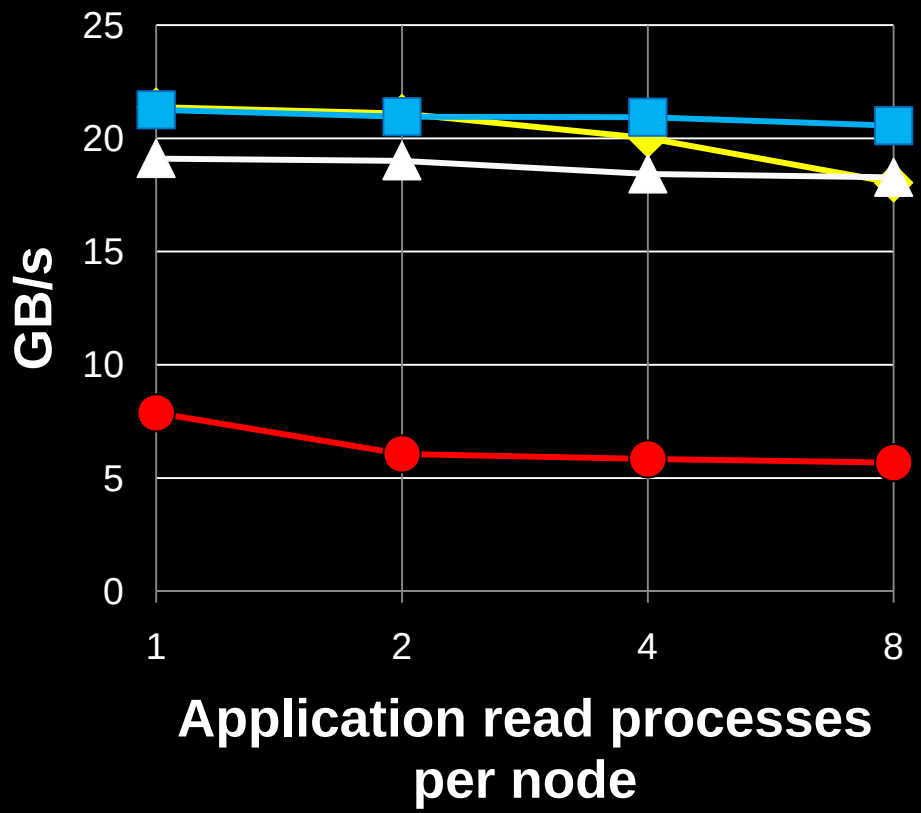


Efficiency I: Achieving Better I/O BW

One Bodhi daemon
per user process



One Bodhi daemon
per analysis node

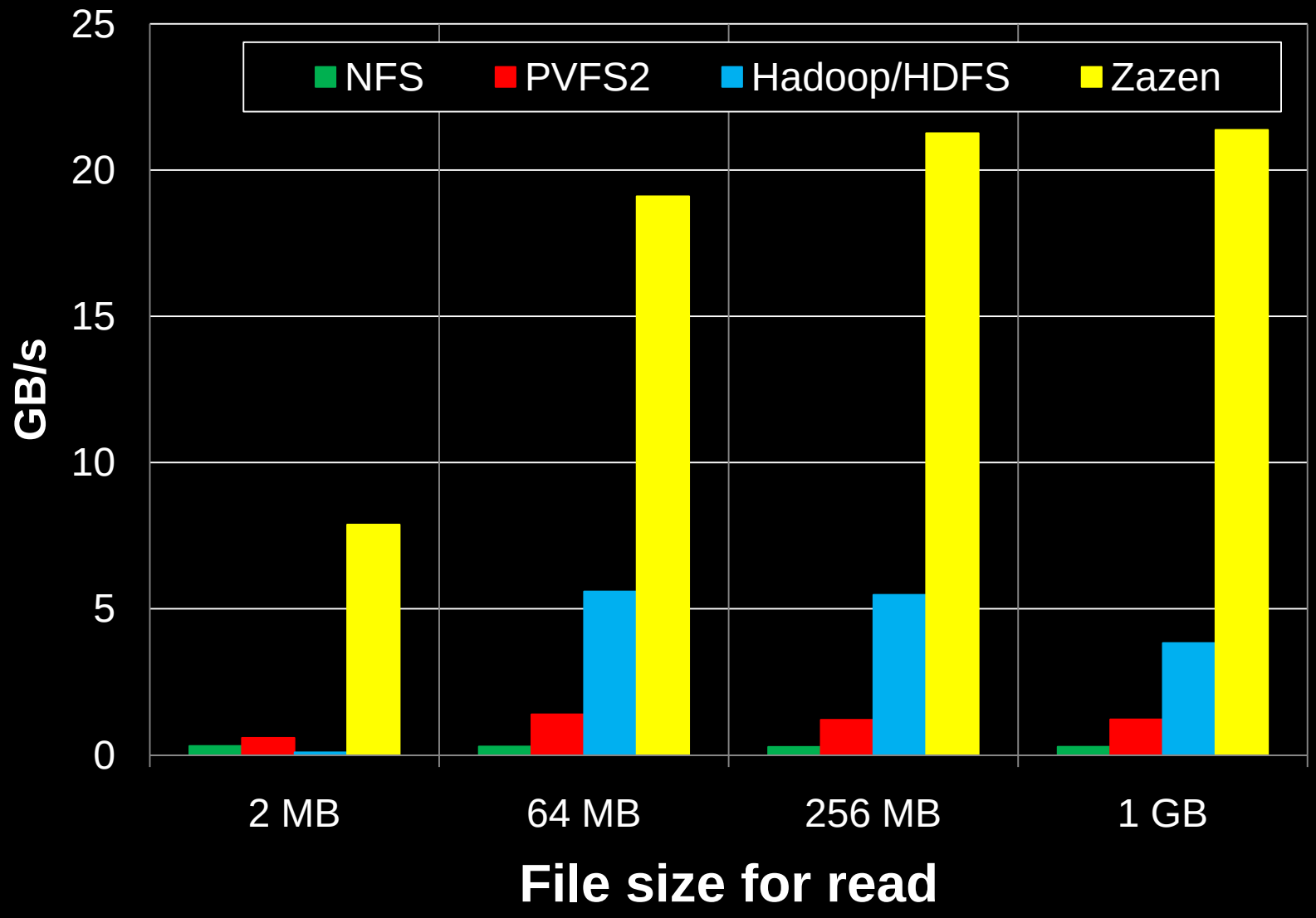


Efficiency II: Comparison w. NFS/PFS

- NFS (v3) on separate enterprise storage servers
  - Dual quad-core 2.8-GHz Opteron processors, 16 GB memory, 48 SATA disks organized in RAID 6
  - Four 1 GigE connection to the core switch of the 100-node cluster
- PVFS2 (2.8.1) on the same 100 analysis nodes
  - I/O (data) server and metadata server on all nodes
  - File I/O performed via the PVFS2 Linux kernel interface
- Hadoop/HDFS (0.19.1) on the same 100 nodes
  - Data stored via HDFS's C library interface, block sizes set to be equal to file sizes, three replications per file
  - Data accessed via a read-only Hadoop MapReduce Java program (with a number of best-effort optimizations)

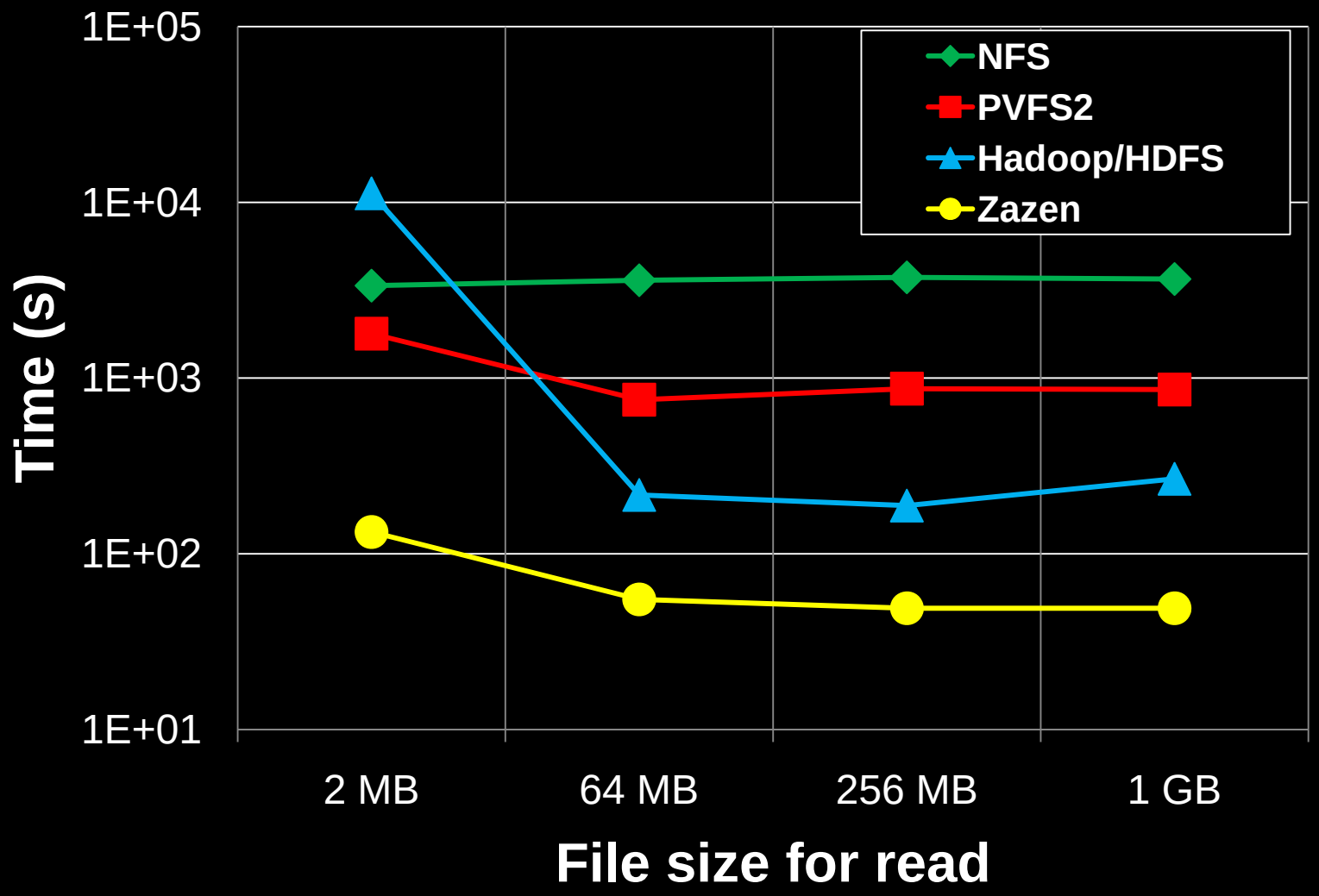
Efficiency II: Outperforming NFS/PFS

I/O bandwidth of reading files of different sizes



Efficiency II: Outperforming NFS/PFS

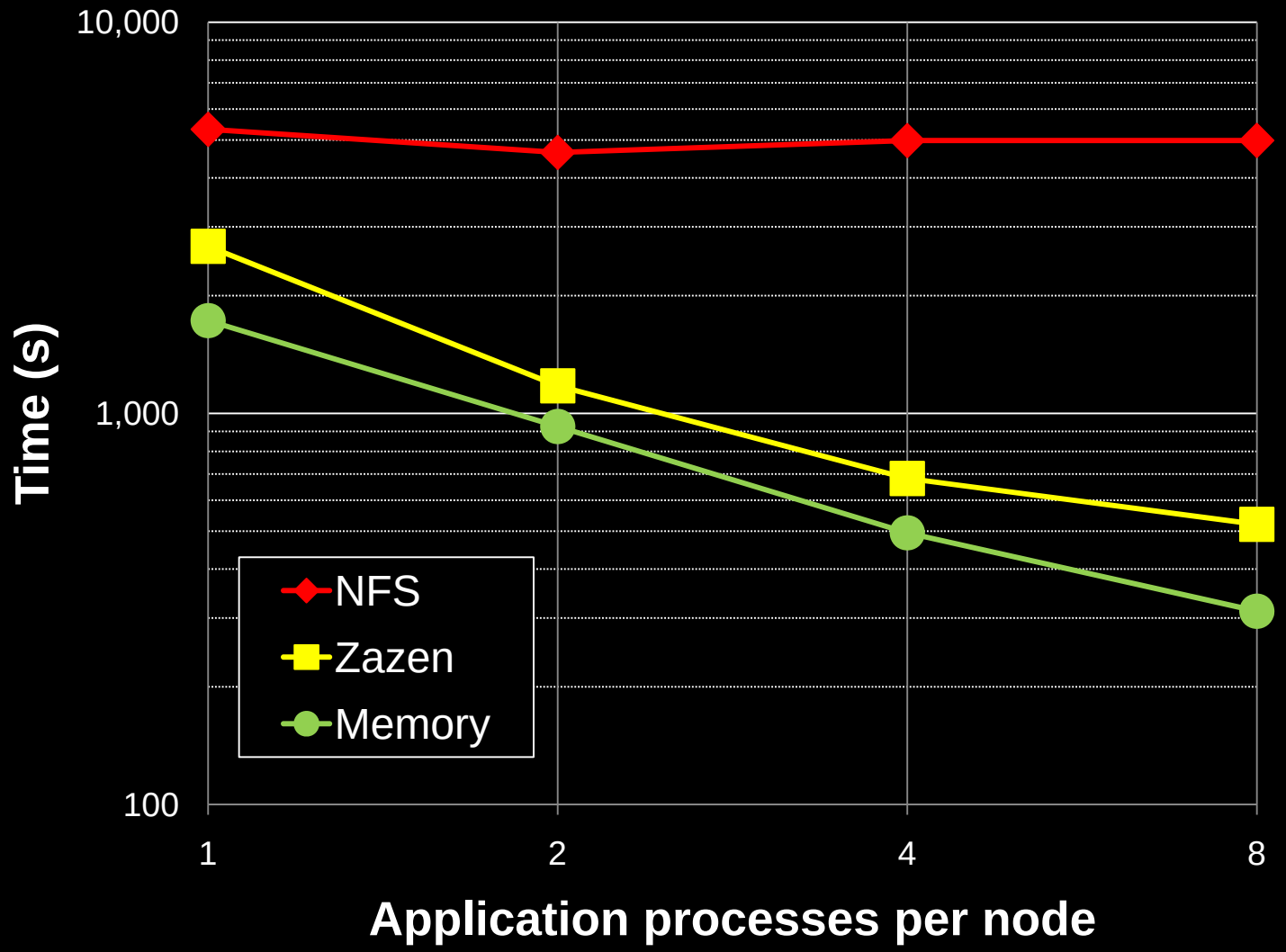
Time to read one terabyte of data



Read Perf. under Writes (1GB/s)

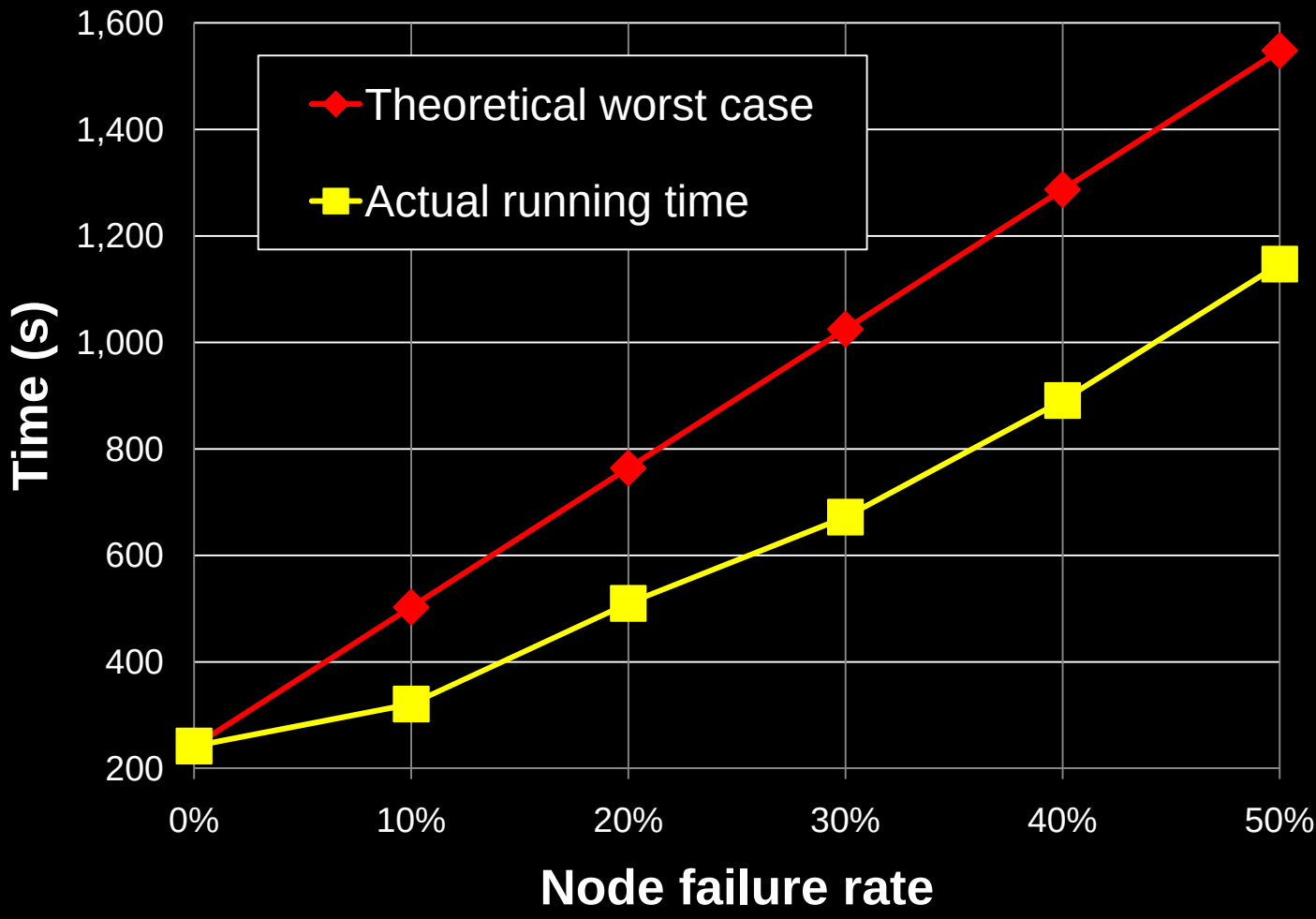
End-to-End Performance

- A HiMach analysis program called *water residence* on 100 nodes
- 2.5 million small frame files (430 KB each)



Robustness

- Worst case execution time is $T(1 + \delta (B/b))$
- The water-residence program re-executed with varying number of nodes powered off



Summary

Zazen accelerates order-independent, parallel data access by (1) actively caching simulation output, and (2) executing an efficient distributed consensus protocol.

- Simple and robust
- Scalable on a large number of nodes
- Much higher performance than NFS/PFS *
- Applicable to a certain class of time-dependent simulation datasets *