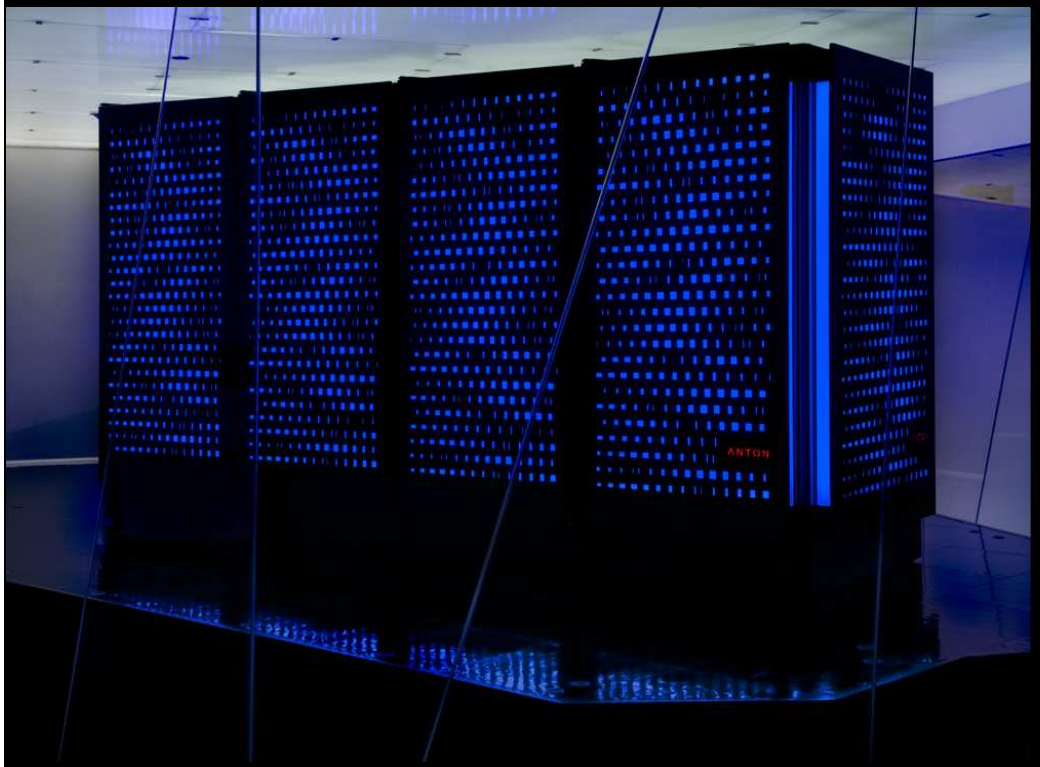
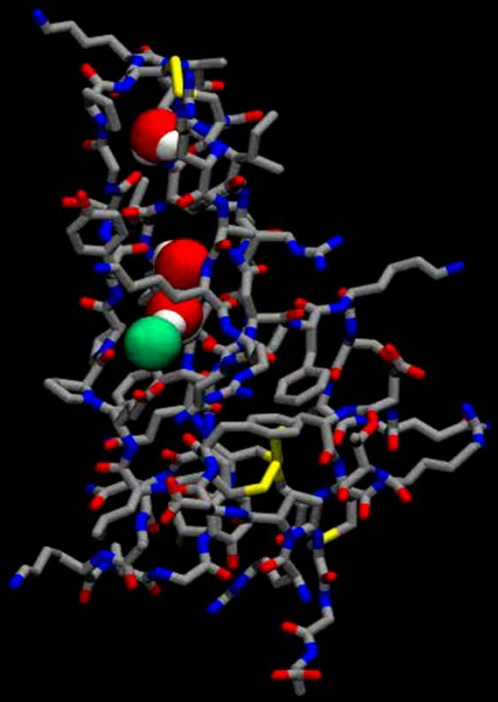
Fault-Tolerant Parallel Analysis of Millisecond-scale Molecular Dynamics Trajectories

Tiankai Tu

D. E. Shaw Research

Anton: A Special-Purpose Parallel Machine for MD Simulations





“Routine” Data Analysis Tools

- Data validation: detect corrupted simulation output data via checksums
- Summary statistics: rmsd, bond length, dihedral angles, electron density map
- Event detection: ion permeation, conformational change, binding events
- Data clustering: k-mean, EM-based Procrustes analysis

Millisecond-Scale MD Trajectories

Simulation length:	1x 10^{-3} s
÷ Output interval:	100 x 10^{-12} s
Total output frames:	10 M frames

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

A Single Trajectory size is about 6 TB

Performance Bottleneck in Analysis

Millisecond-scale trajectory size :	6 TB
Local disk read bandwidth:	100 MB / s
Data access time:	16 hours
Analysis time:	O(n)
Total time:	**day(s)**

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

Support for “Routine” Data Analysis

Data validation

Summary statistics

Event detection

Data Clustering

Parallel programming model

Parallel I/O (read in particular)

Fault tolerance

Part I: How to Analyze MD Trajectories in Parallel?

Infrastructure for Data Analysis

Data validation

Summary statistics

Event detection

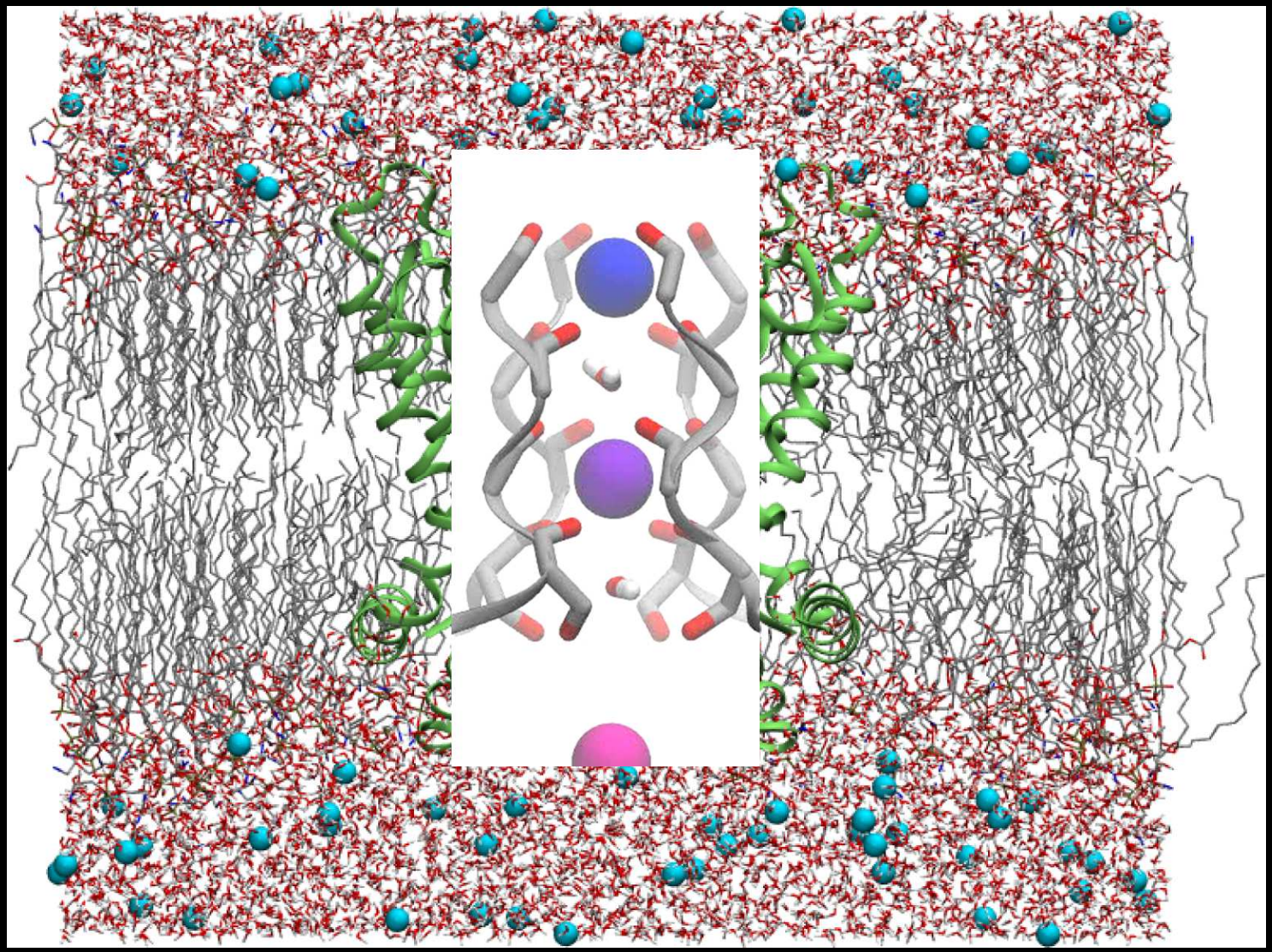
Data Clustering

Parallel programming model

Parallel I/O (read in particular)

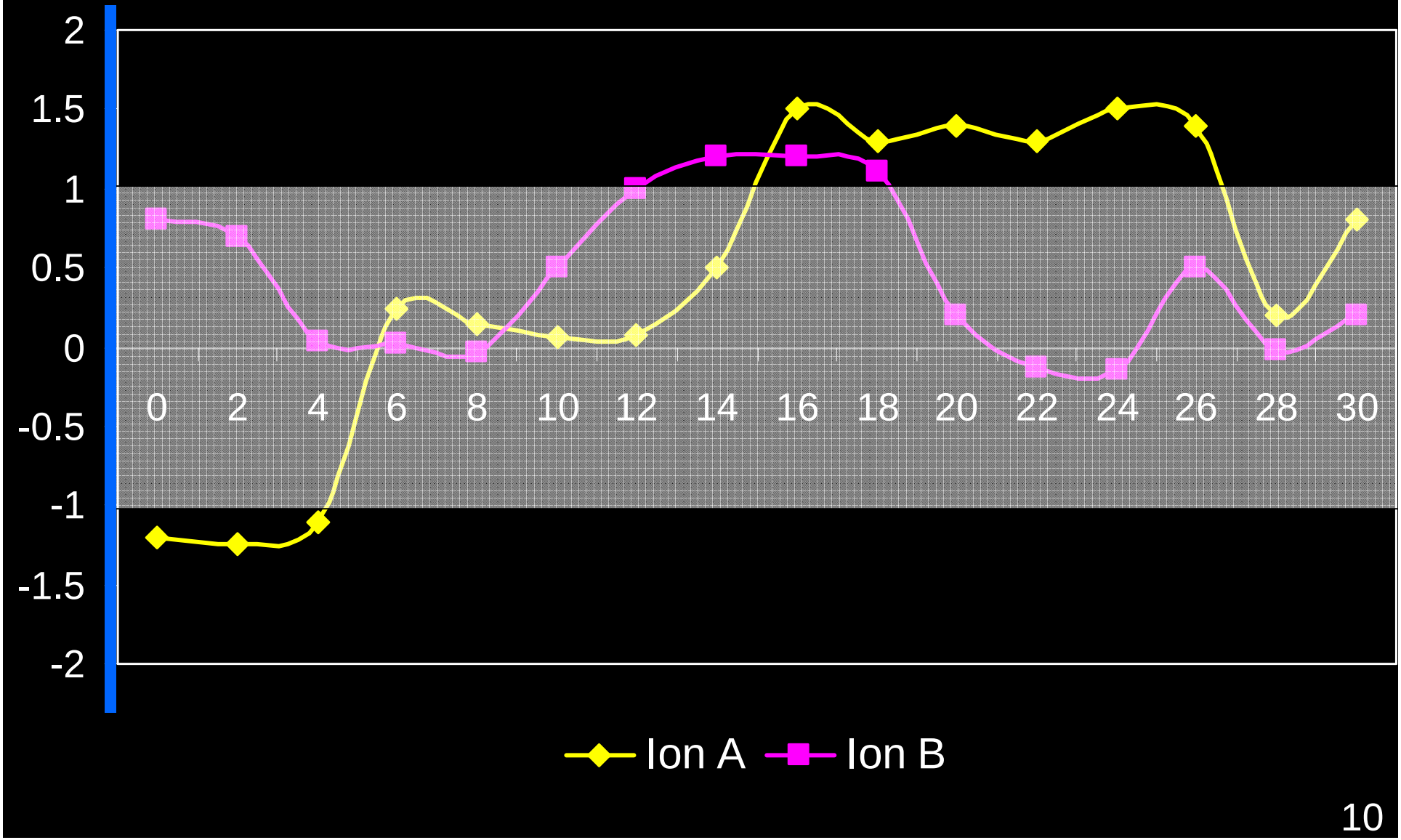
Fault tolerance

An Event Detection Example: Ion Permeation

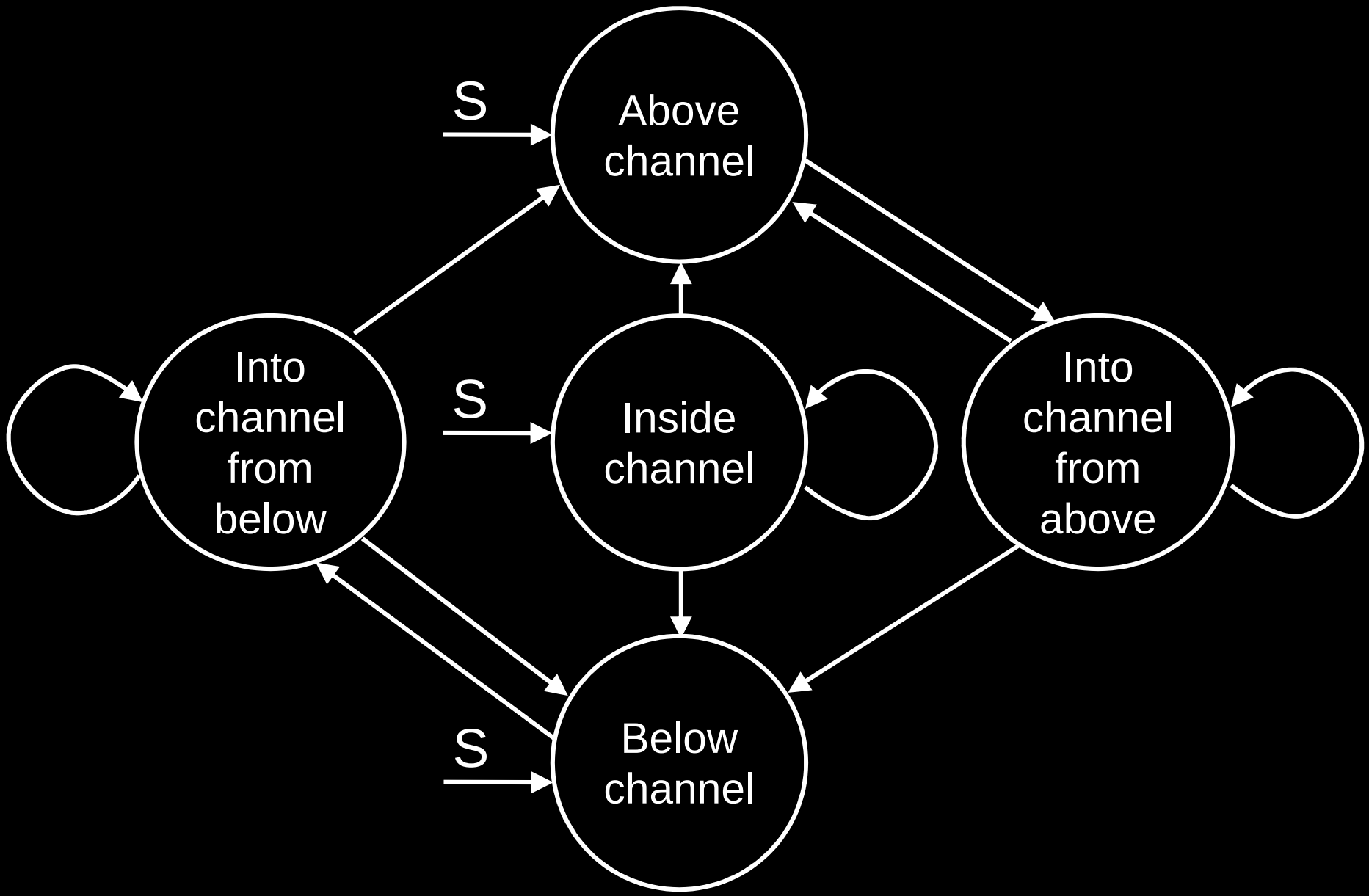


A Hypothetic Trajectory

20,000 atoms in total; two ions of interest



Ion State Transition



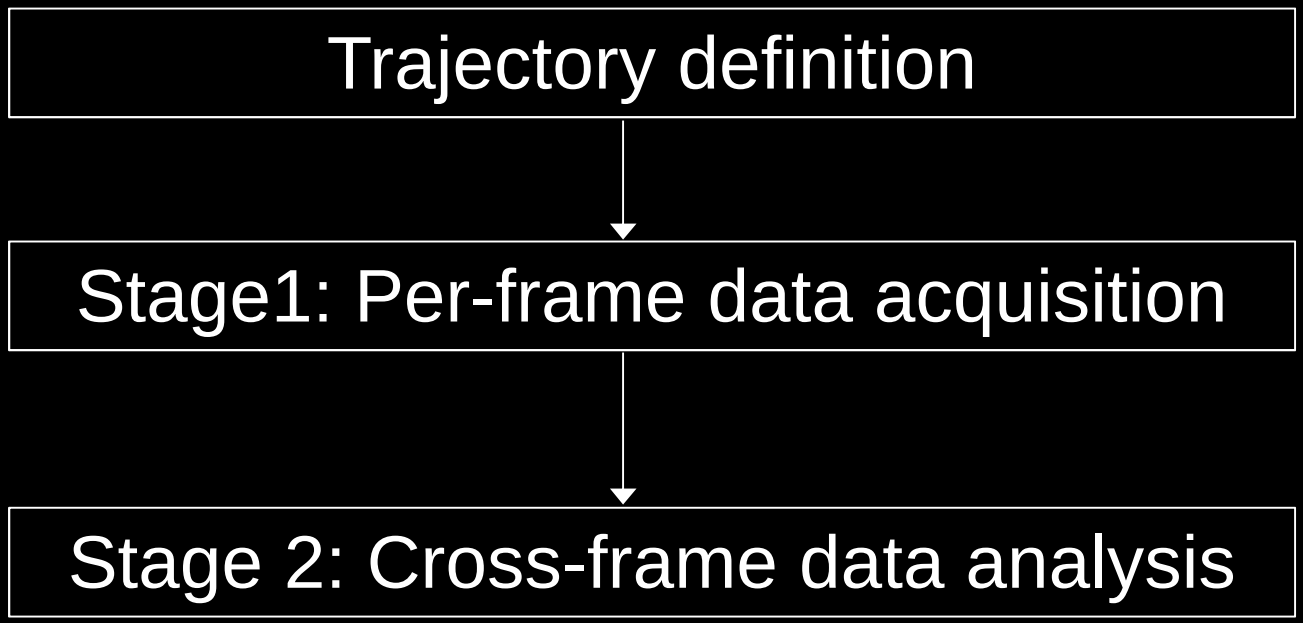
Sequential Analysis

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

Strong inter-frame data dependence:
Analysis (state transition) tightly coupled with data acquisition (positions of ions)

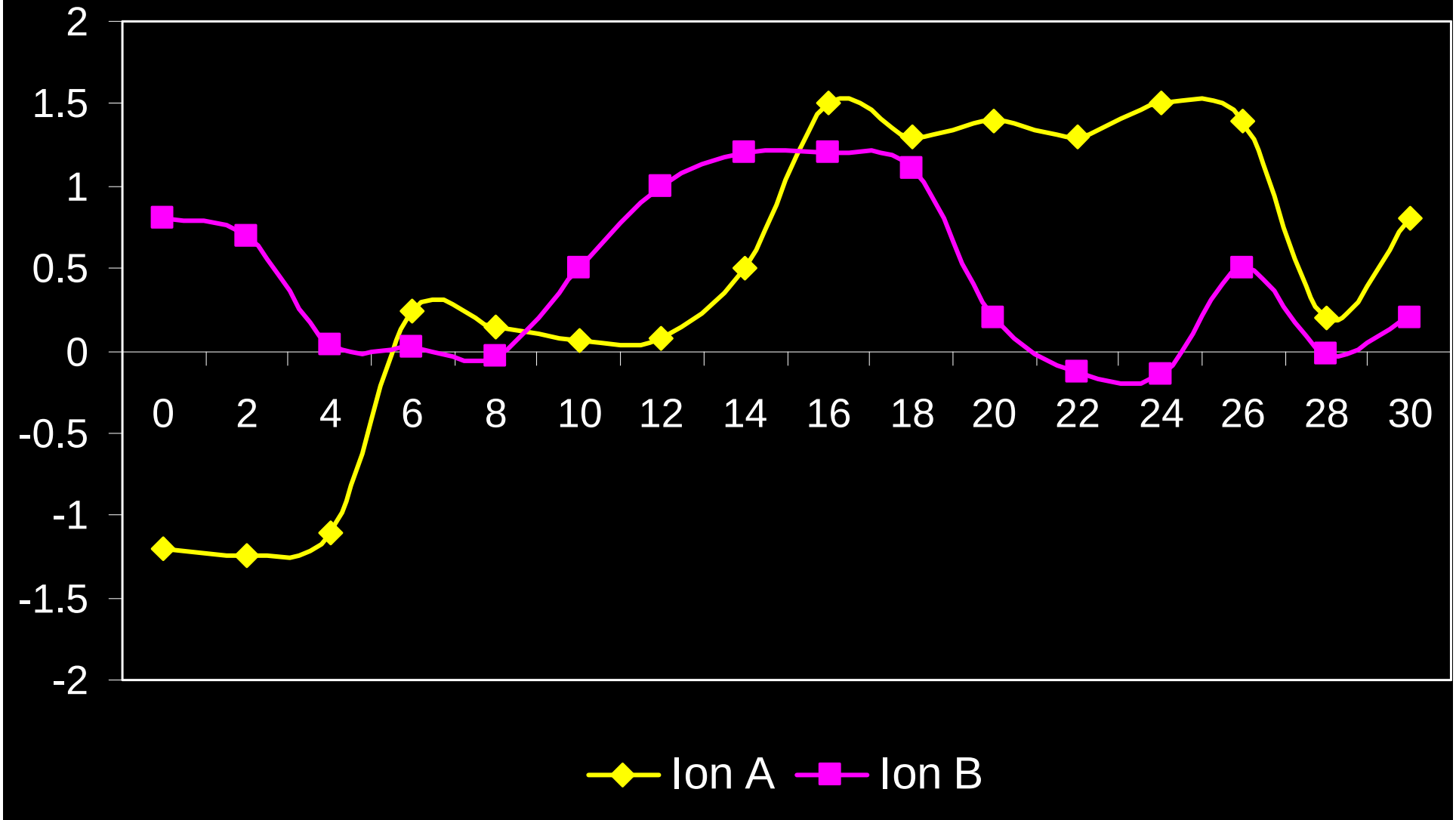
A Different View of Analysis

- Specify which frames to be accessed
- Decouple data acquisition from data analysis

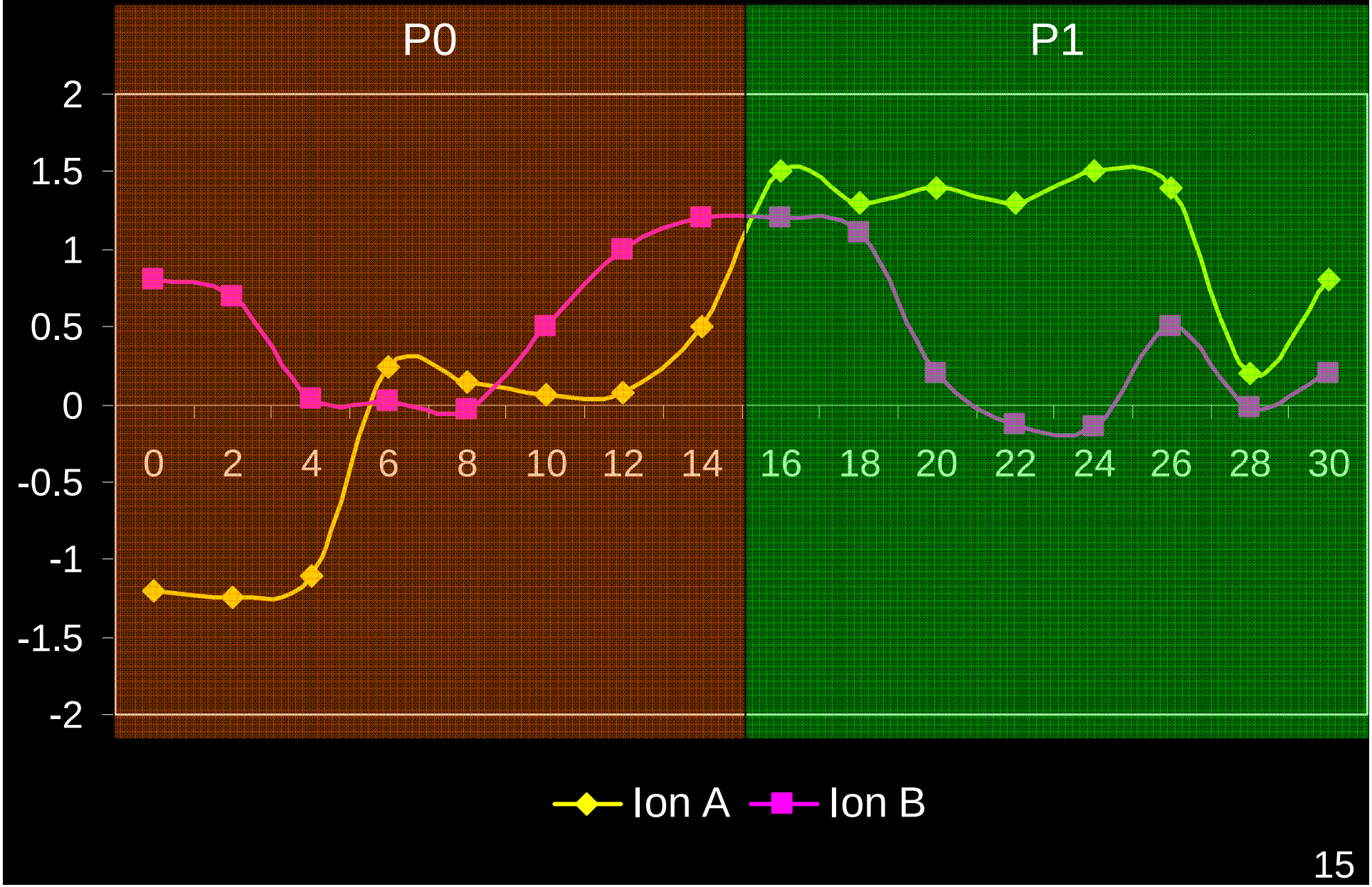


Trajectory Definition

Every other frame in the trajectory

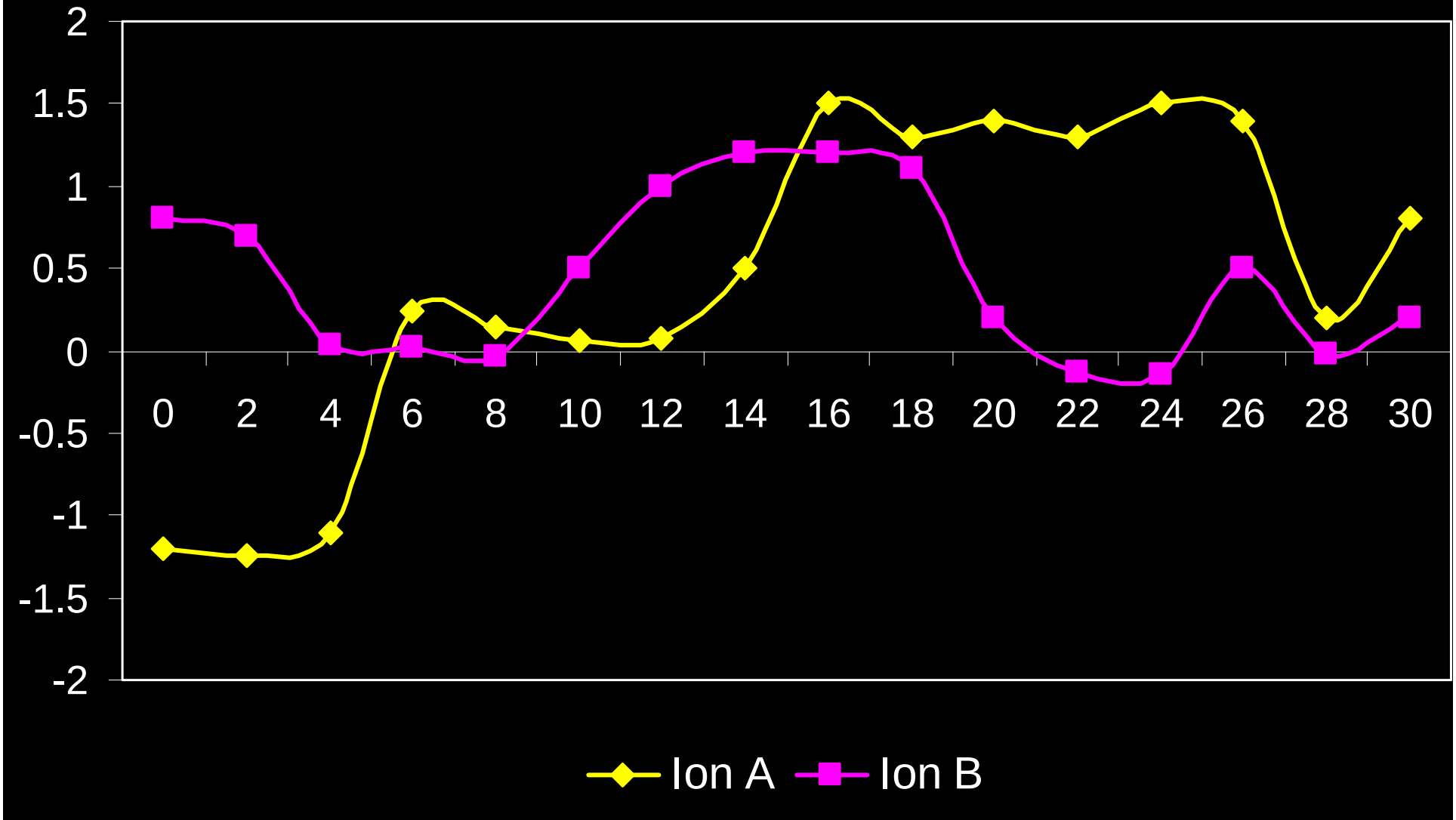


Per-frame Data Acquisition (stage 1)

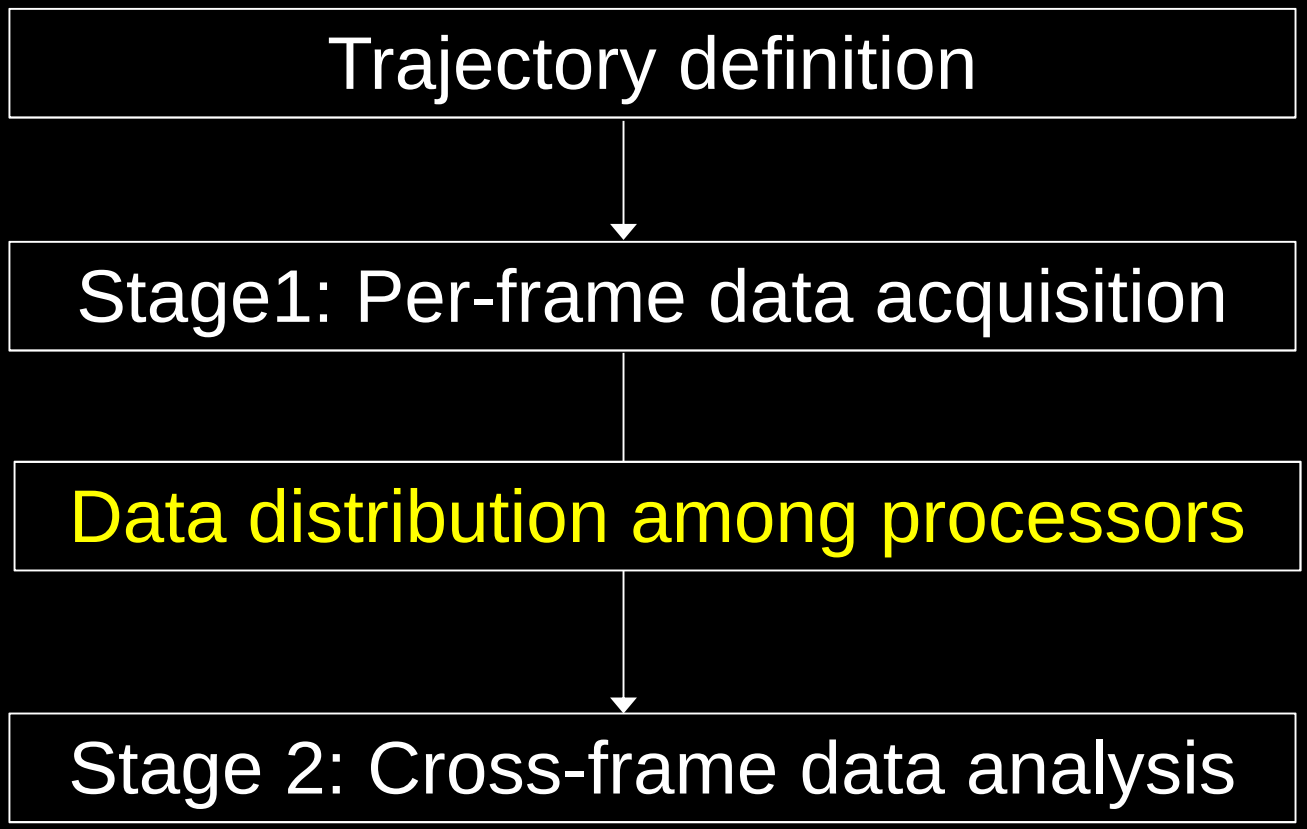


Cross-frame Data Analysis (stage 2)

Analyze ion A on P0 and ion B on P1

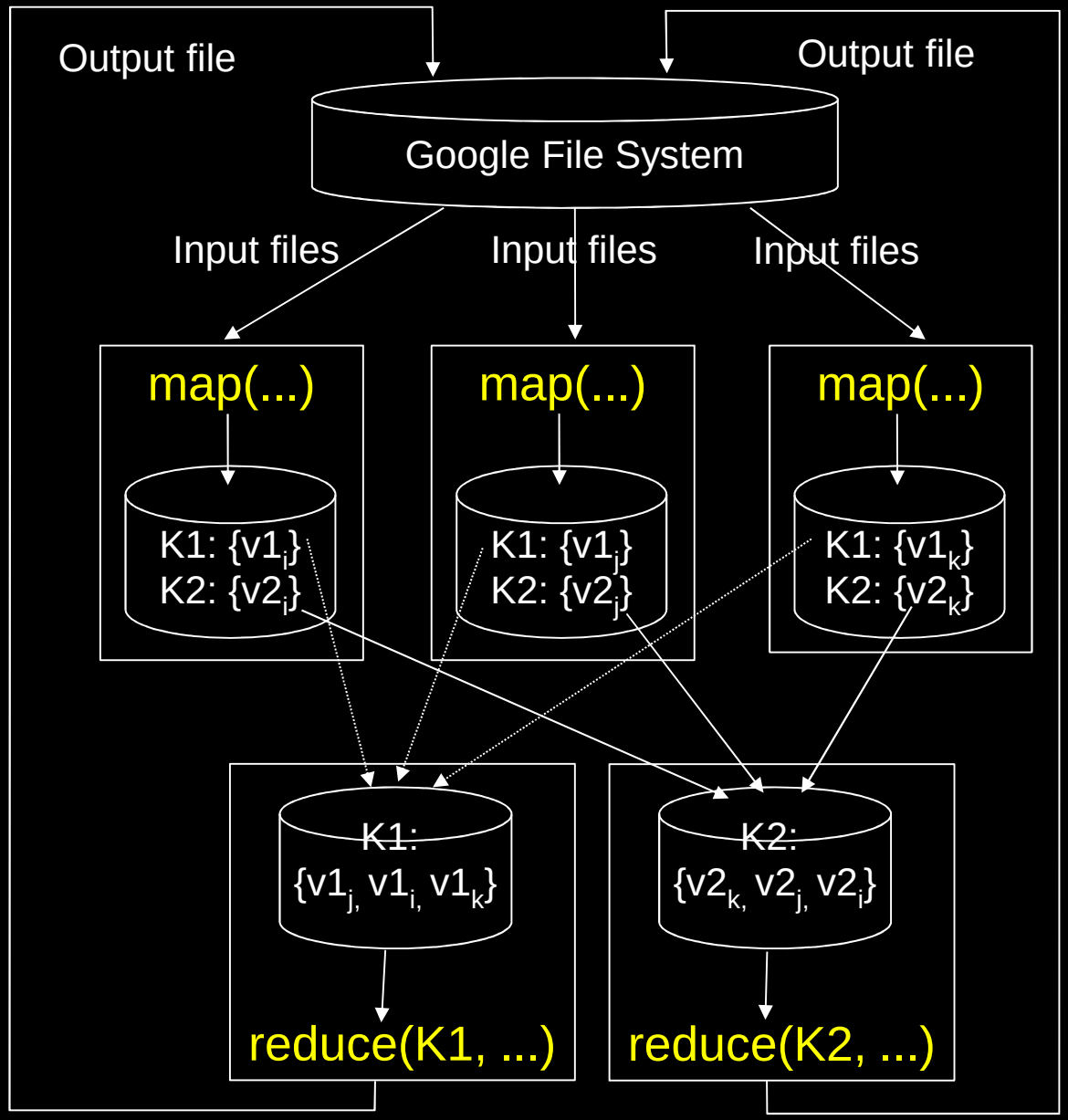


A Parallel Programmer's Perspective



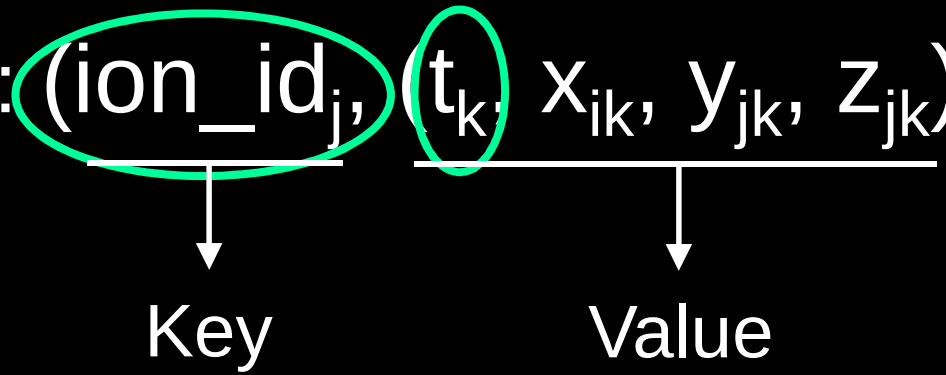
How to implement the model without writing explicit parallel code?

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: $(ion\_id_j, (t_k, x_{ik}, y_{jk}, z_{jk}))$



Implementation: HiMach

- A MapReduce-style library that allows users to write Python programs to analyze MD trajectory
- An MPI-based parallel runtime that executes HiMach programs in parallel on a Linux cluster
- Performance: two orders of magnitude faster on 512 cores than on a single core

Part II: How to Overcome the I/O Bottleneck in Data Analysis?

Infrastructure for Data Analysis

Data validation

Summary statistics

Event detection

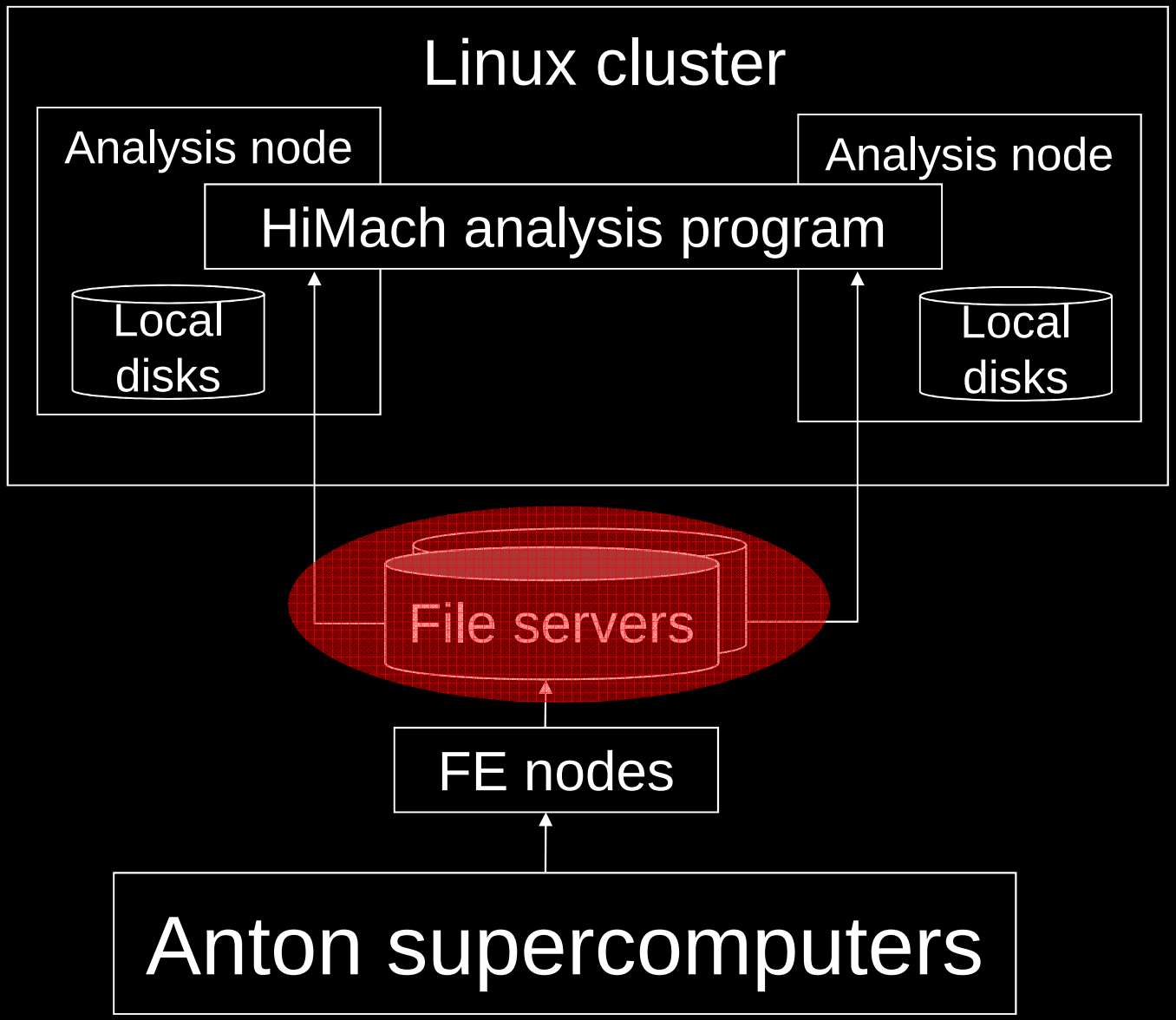
Data Clustering

Parallel programming model

Parallel I/O (read in particular)

Fault tolerance

Traditional I/O Infrastructure



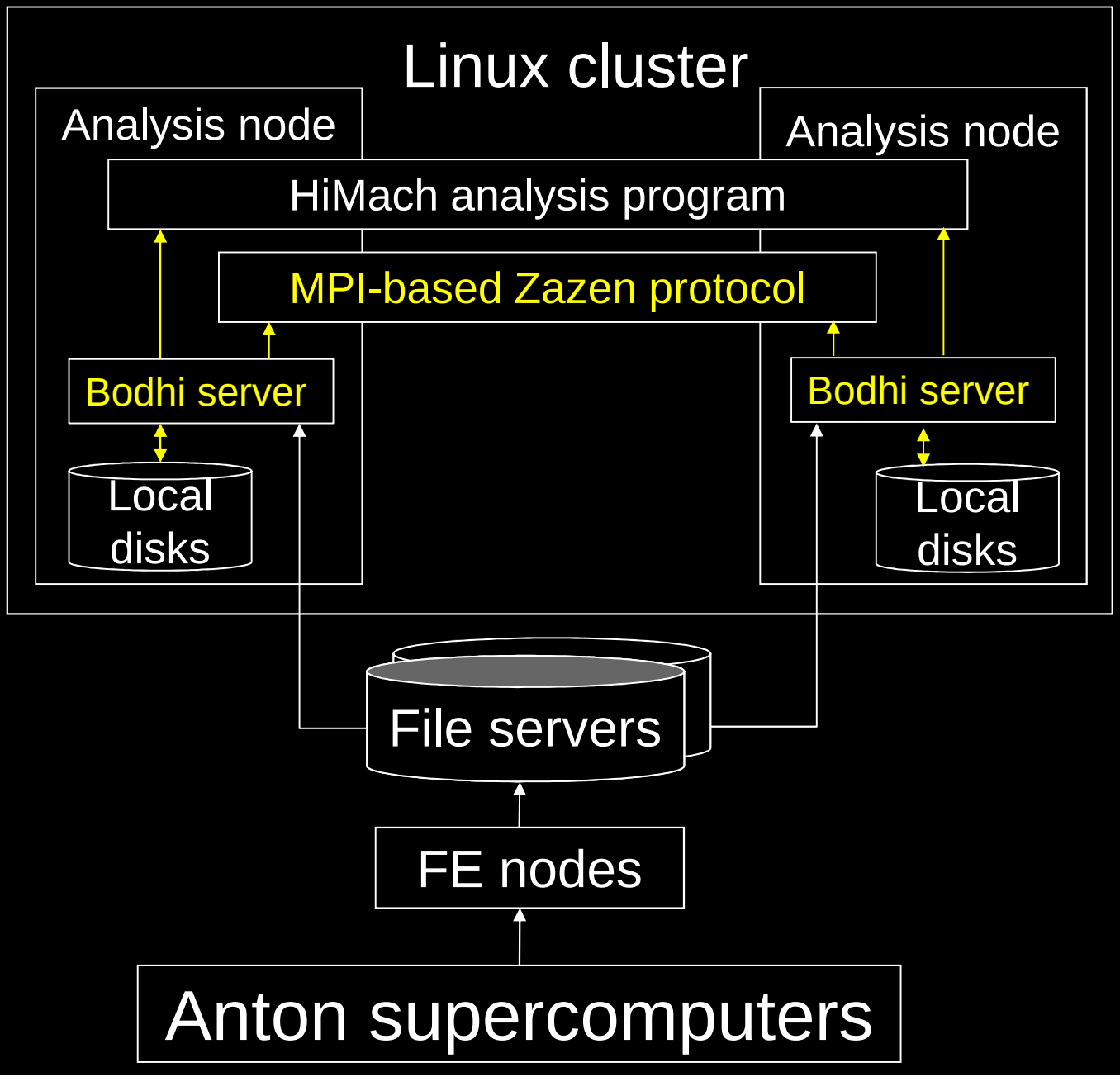
Characteristics of MD Trajectories

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

The Main Ideas

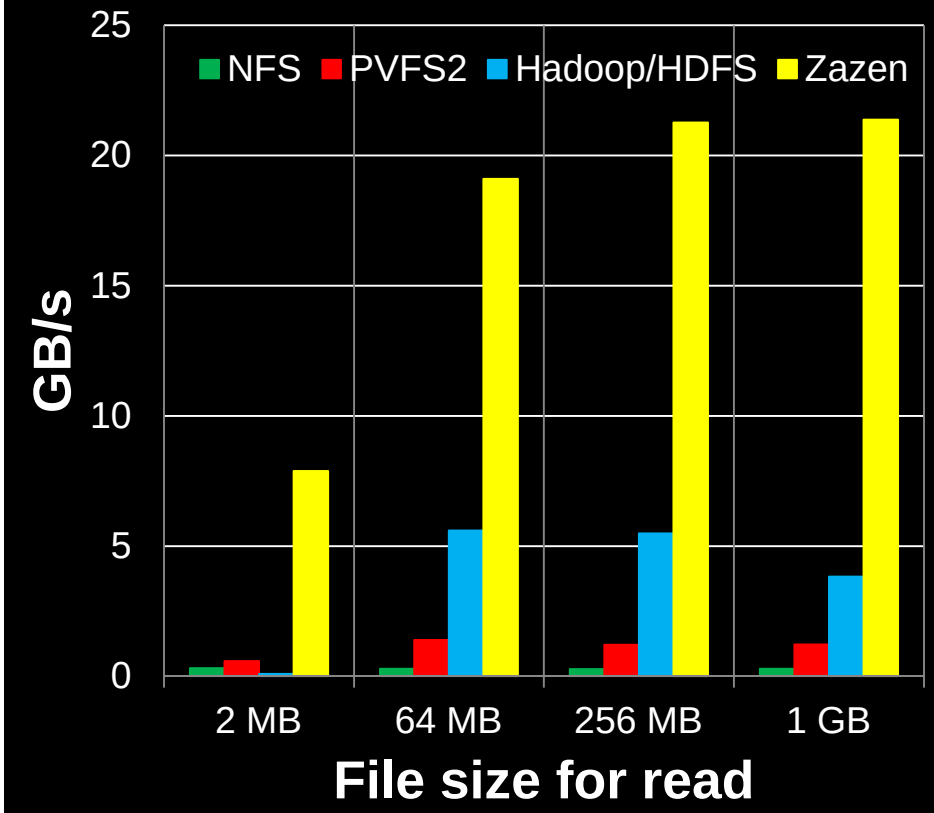
- Convert local hard drives on analysis nodes into a confederated disk cache
- Do not use a metadata server (single point of failure, performance bottleneck)
- Use MPI collective operations and bitmaps to arbitrate which nodes need to process which frames (a distributed consensus problem)

Implementation: Zazen

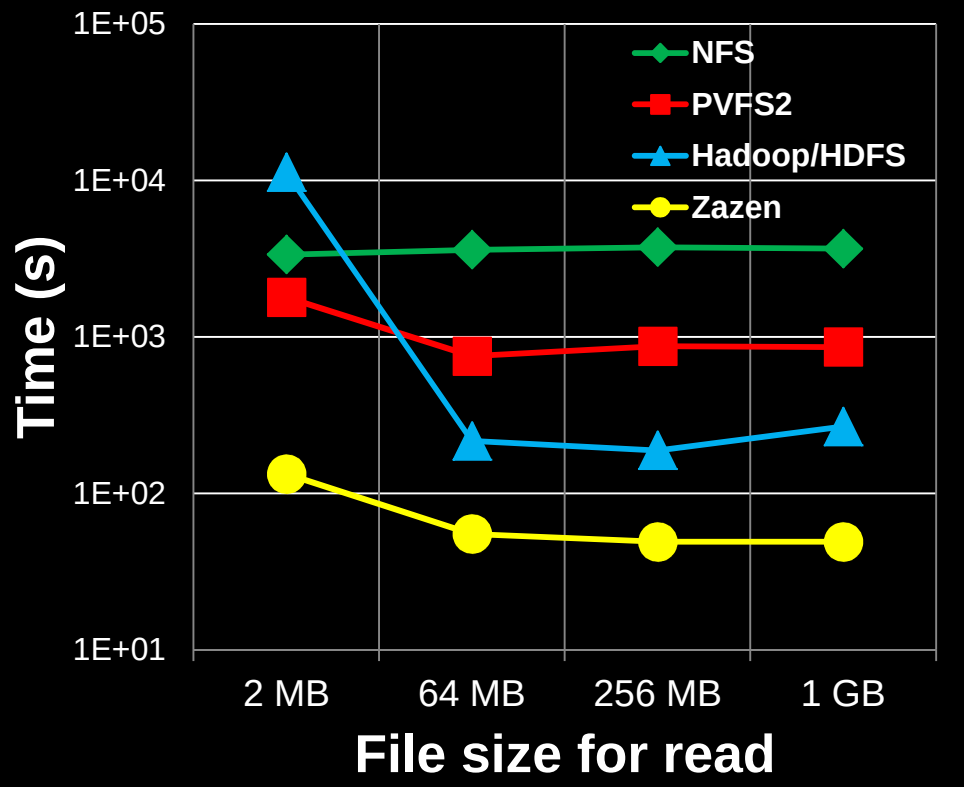


Efficiency: Outperforming NFS/PFS

I/O bandwidth of reading files of different sizes

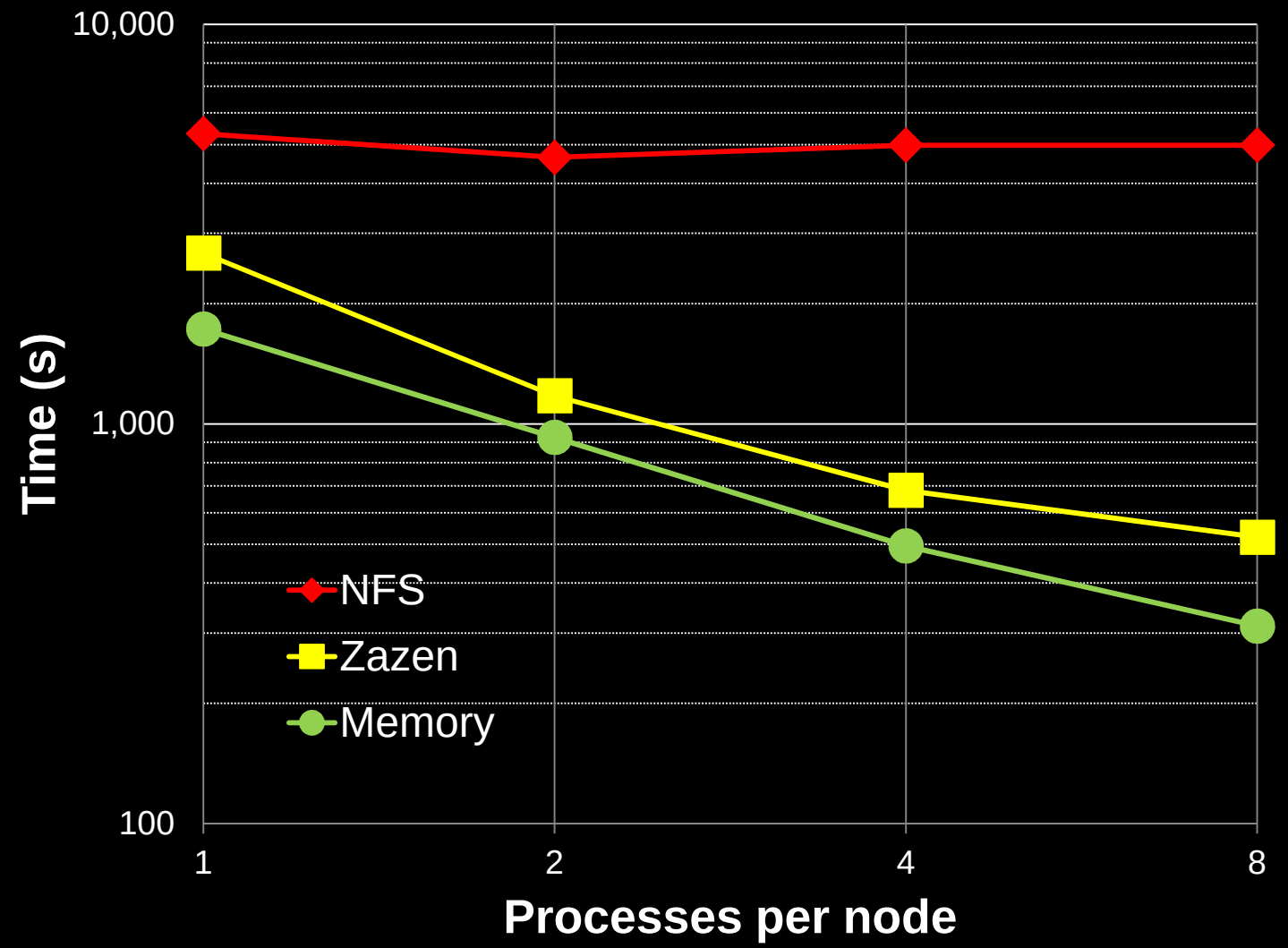


Time to read 1 terabyte files of different sizes



End-to-End Performance

- A HiMach parallel analysis program call *water-residence*
- 2.5 million small files/frames (430 KB each)



Part III: How to make analysis computation fault tolerant?

Infrastructure for Data Analysis

Data validation

Summary statistics

Event detection

Data Clustering

Parallel programming model

Parallel I/O (read in particular)

Fault tolerance

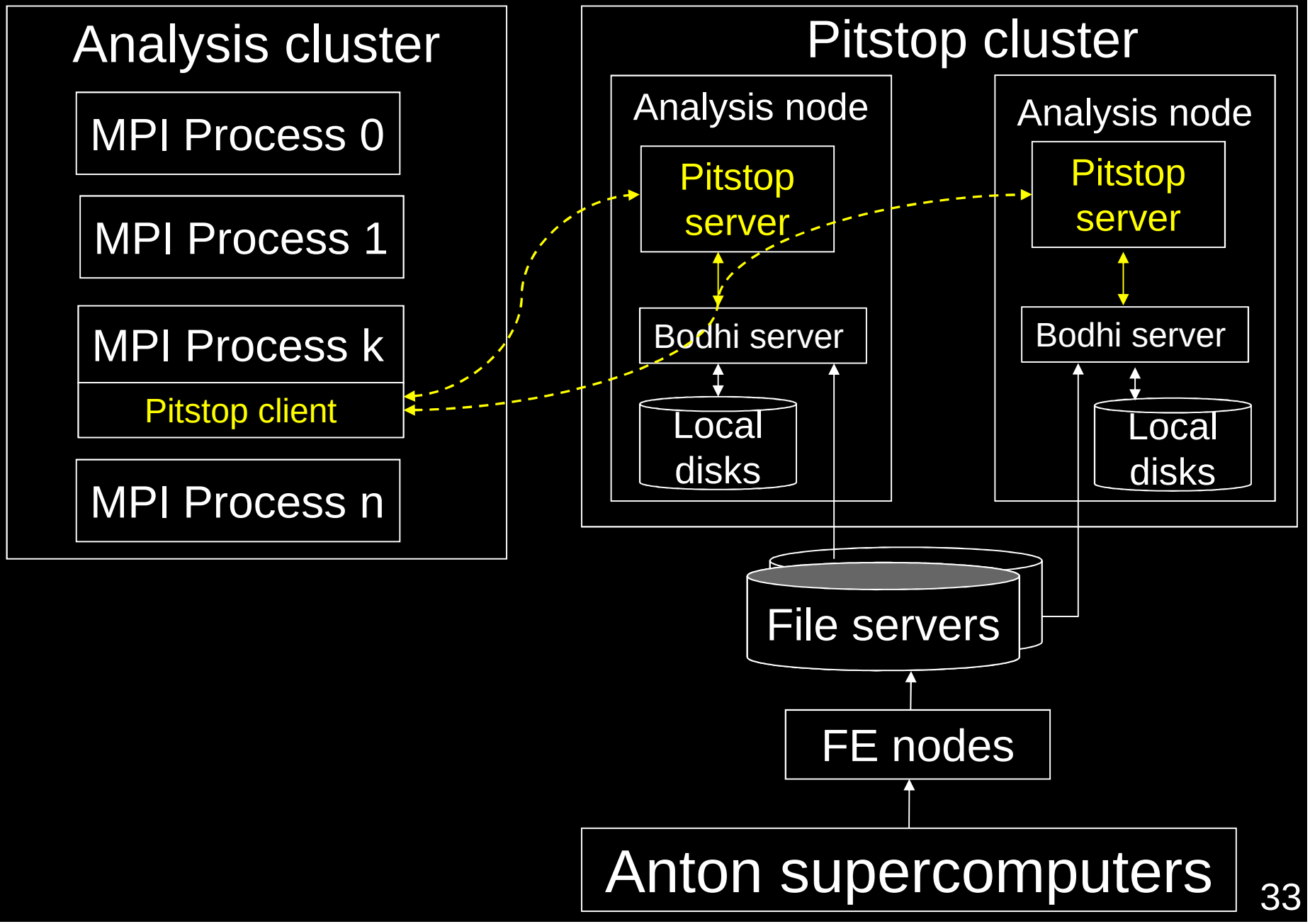
Common Types of Failures

- Network File System (NFS), local hard drives, network switches
- Interplay between MPI and NFS may cause stalling of a HiMach analysis job

The Main Ideas

- Decouple MPI from disk read operation
- Retain MPI for the compute/communication intensive operations (reduce phase)
- Use a client-server model for the map phase
- Build a distributed parallel execution engine in conjunction with a disk cache manager
- Treat servers as a peer-to-peer network
- Use consistent hashing for frame placement
- Allow application-level data re-organization

Implementation: Pitstop



Pitstop as a Fail-Fast System

- Supports molecular visualization applications
- Enables interactive data retrieval
- Executes HiMach batch analysis jobs
- Survives various node failures and NFS mount problems

Summary

- Interpretation of massive MD trajectories calls for (1) strong scientific intuition, (2) pertinent and efficient analysis algorithms, and (3) fast and scalable implementation
- The progression from HiMach to Zazen to Pitstop has improve our chemists' ability to analyze millisecond-scale MD trajectories more efficiently and effectively