DeltaFS Indexed Massive Dir

Software-Defined Storage For Fast Query

PDSW-DISCS 2017

Qing Zheng, George Amvrosiadis, Saurabh Kadekodi, Michael Kuchnik
Chuck Cranor, Garth Gibson
Brad Settlemyer, Gary Grider, Fan Guo
Carnegie Mellon University
Los Alamos National Laboratory (LANL)

IRHPIT INSTITUTE FOR RELIABLE HIGH PERFORMANCE INFORMATION TECHNOLOGY

DeltaFS Indexed Massive Dir

Key features

1. Require no dedicated resources
2. Almost no post-processing is needed
3. Low I/O overhead

DeltaFS Indexed Massive Dir

Target workloads

1. Data-intensive HPC simulations
2. Not designed for indexing checkpoints
3. I/O bandwidth is limited

Agenda

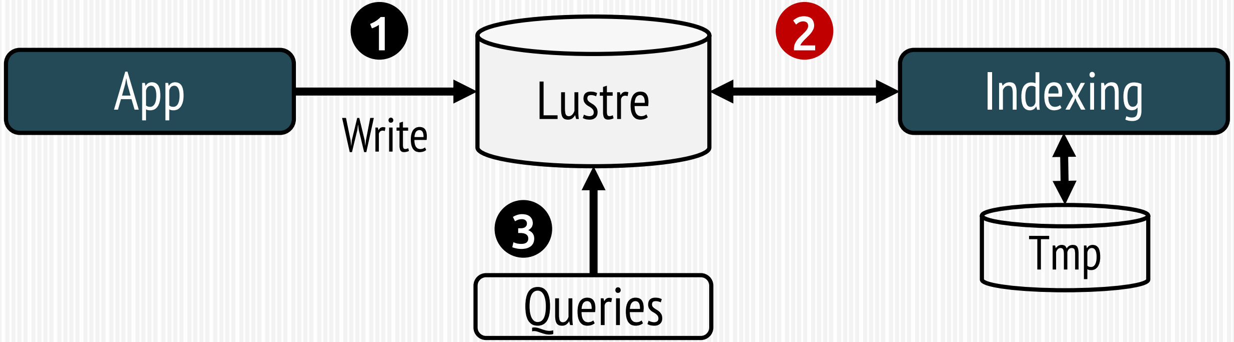
Part 1 – Motivation

Part 2 – In-situ indexing design

Part 3 – API, LANL VPIC integration

Conclusion

Existing HPC builds indexes during post-processing



Delay queries until post-processing done (5-20% simulation time)

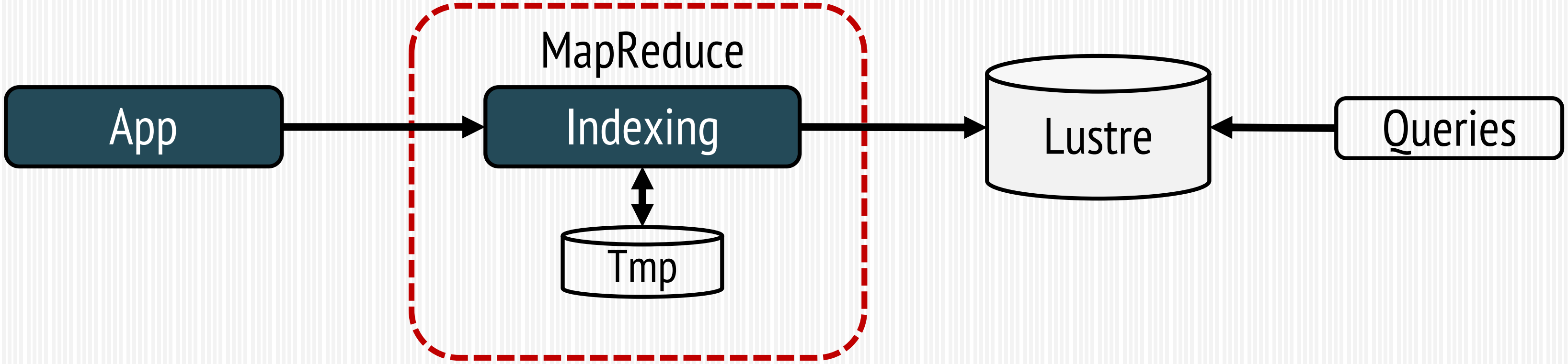
Problem faced:

The increasing time-to-science

Due to the growing gap between compute and I/O
Inefficient support on small data

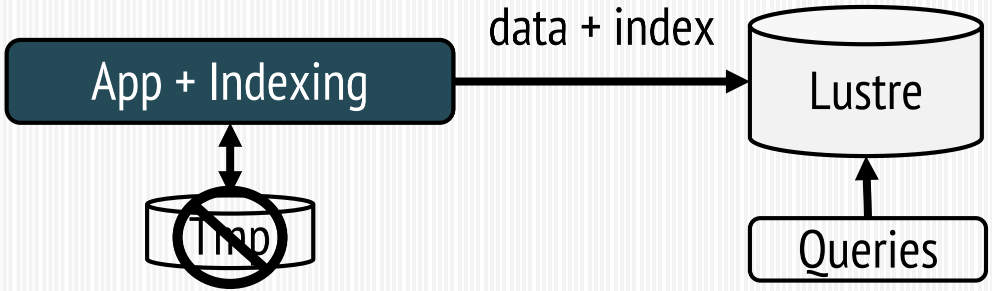
simulation start ↔ query finish

Processing data in-transit while data is written to storage



Need separate resources for sorting and indexing ✕

In-situ indexing directly on app nodes using app resources

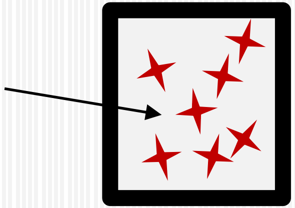


No need for a separate indexing cluster

Key idea:
Reuse storage write-back buffering and idle CPU cycles for in-situ indexing

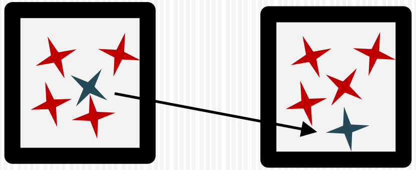
Example app: LANL VPIC

Particle
40 bytes



VPIC simulation

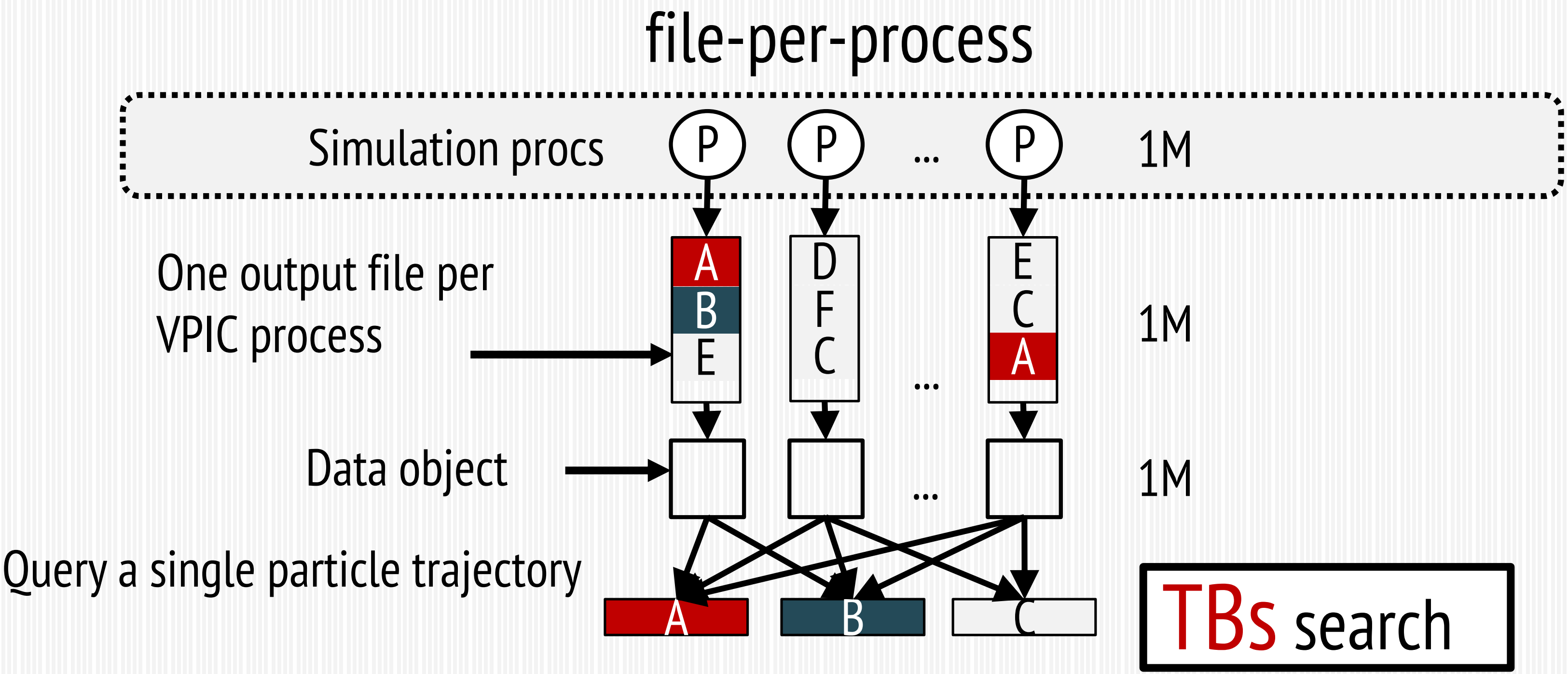
Each VPIC process simulates
millions of particles

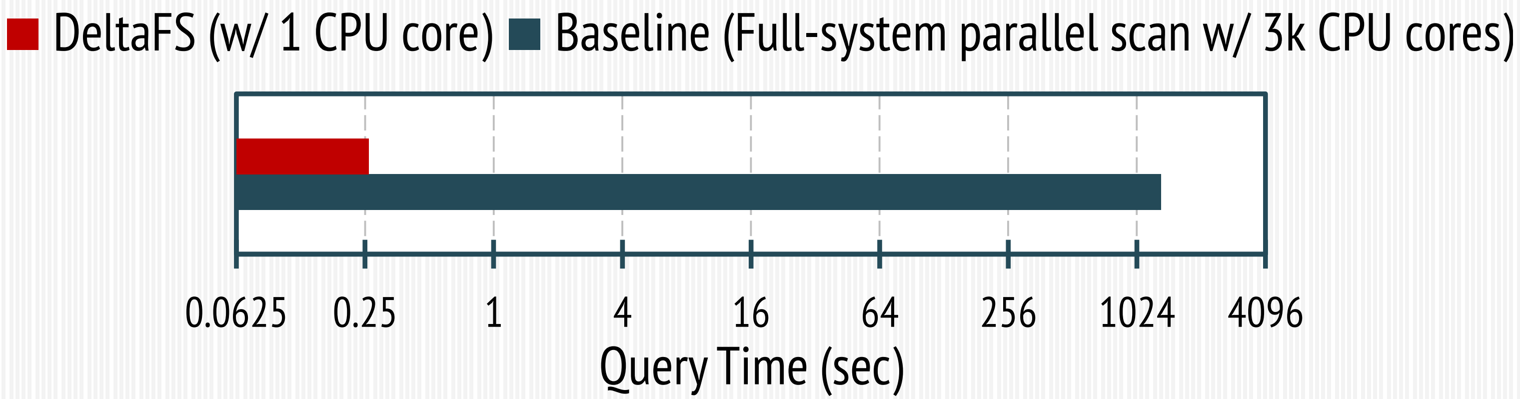


Particles move across
processes during a simulation

Small random writes
After simulation: high-selective queries

TBs I/O per trajectory fetch





Time for reading a single particle trajectory
(10TB, 48 billion particles)

5,000x faster than baseline with DeltaFS in-situ indexing

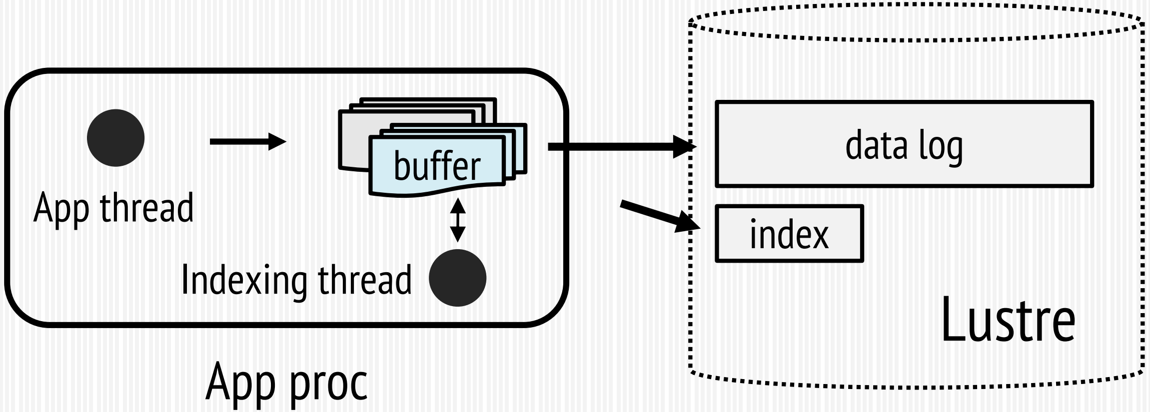
Part II

System design:

Light-weight in-situ indexing

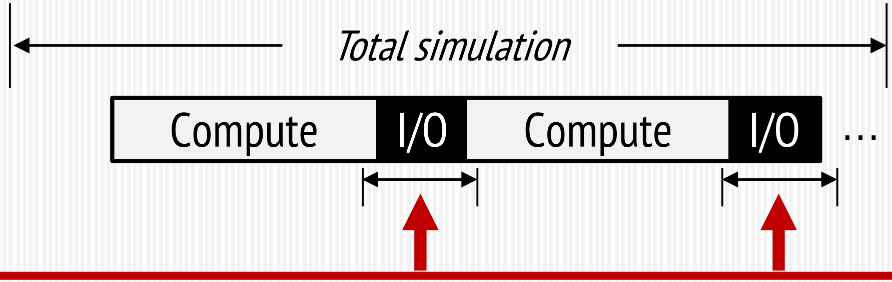
1. Tiny mem footprint
2. Zero write amplification
3. No read back

Resource-efficient indexing by log-structured I/O



Tiny mem footprint, full storage b/w util.

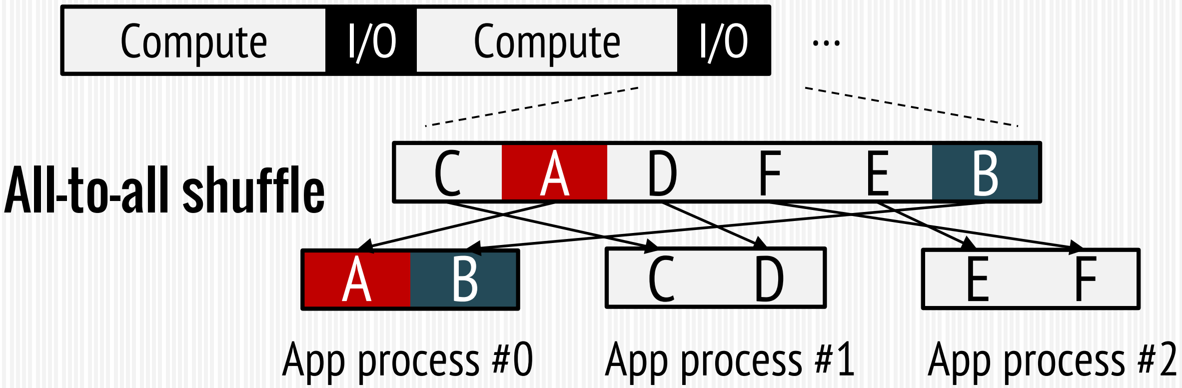
LSM-Trees compacts all the time, but we can't afford it



Must aim for low I/O overhead at 10%-20%

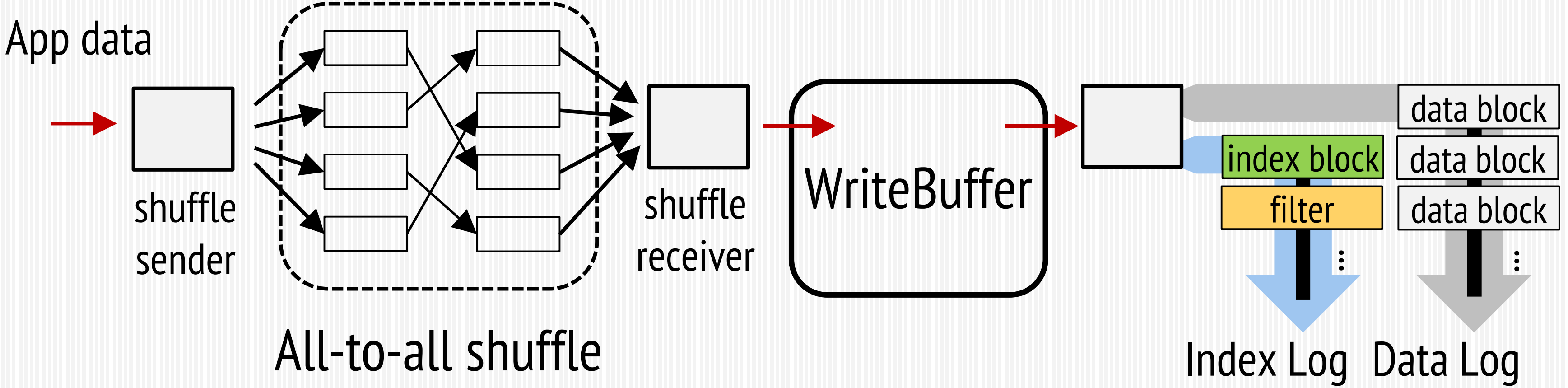
Compaction easily causes 1000% I/O overhead by reading/writing previously written data

In-situ indexing by aggressive data partitioning



Bound the number of data needed per query per timestep

In-situ indexing as a file system lib component



No dedicated cluster needed

Part III

Programming interface:

Indexed Massive Directory (IMD)

In-situ indexing keyed on filenames

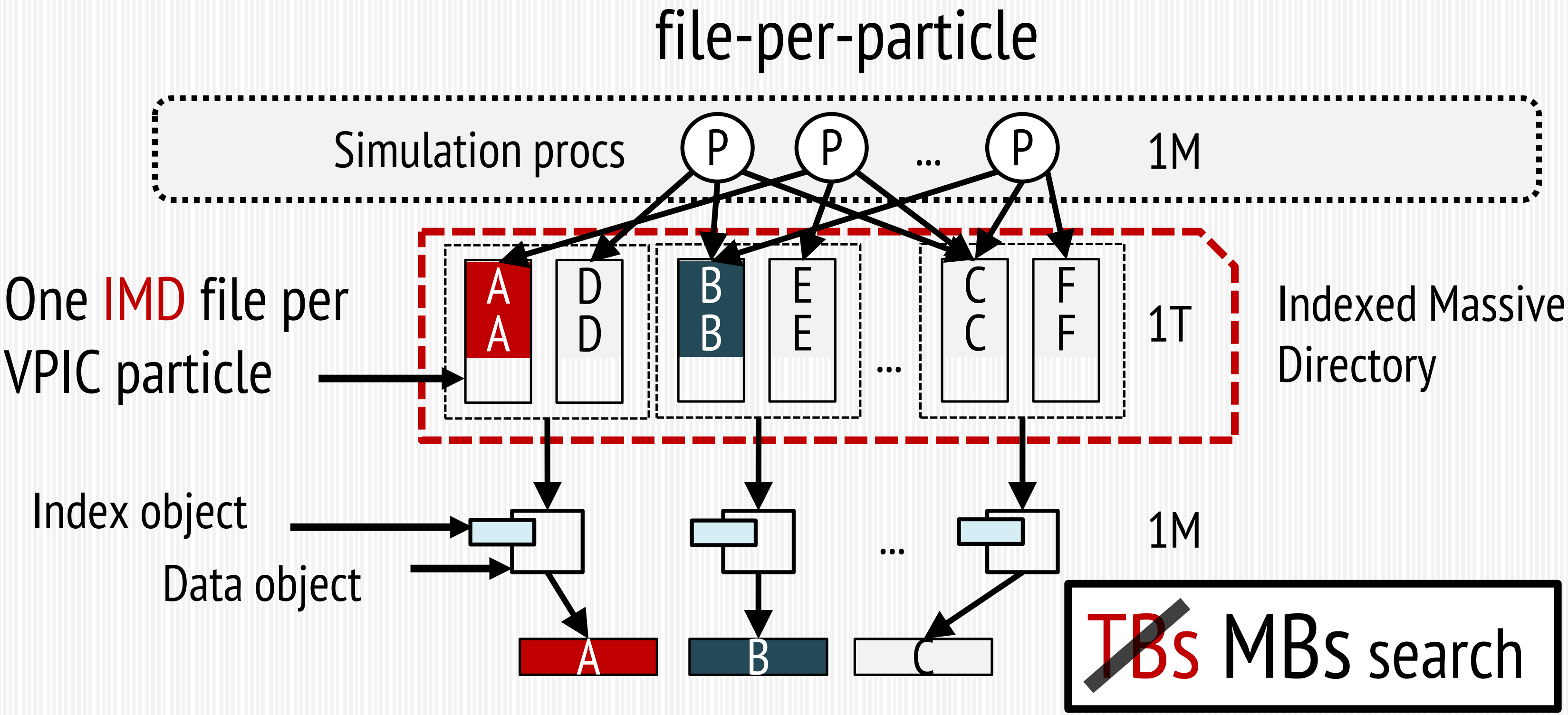
```
mkdir(“./particles”, DELTAFS_IMD)
```

How to use Indexed Massive Dir (IMD)

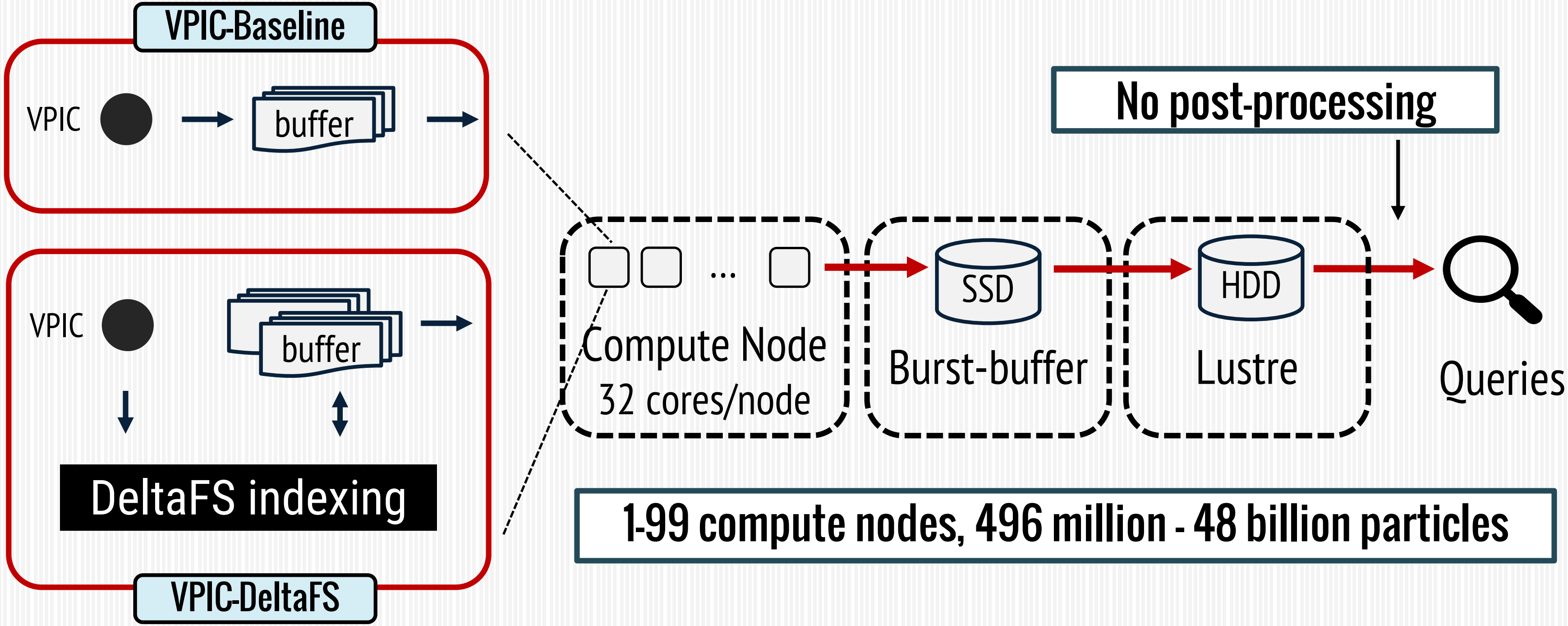
1. Data searched together go into a single IMD file
   e.g. one file for each particle
2. Create as many IMD files as you want
   e.g. 1 trillion files for 1 trillions particles

Query you data by "open-read-close"

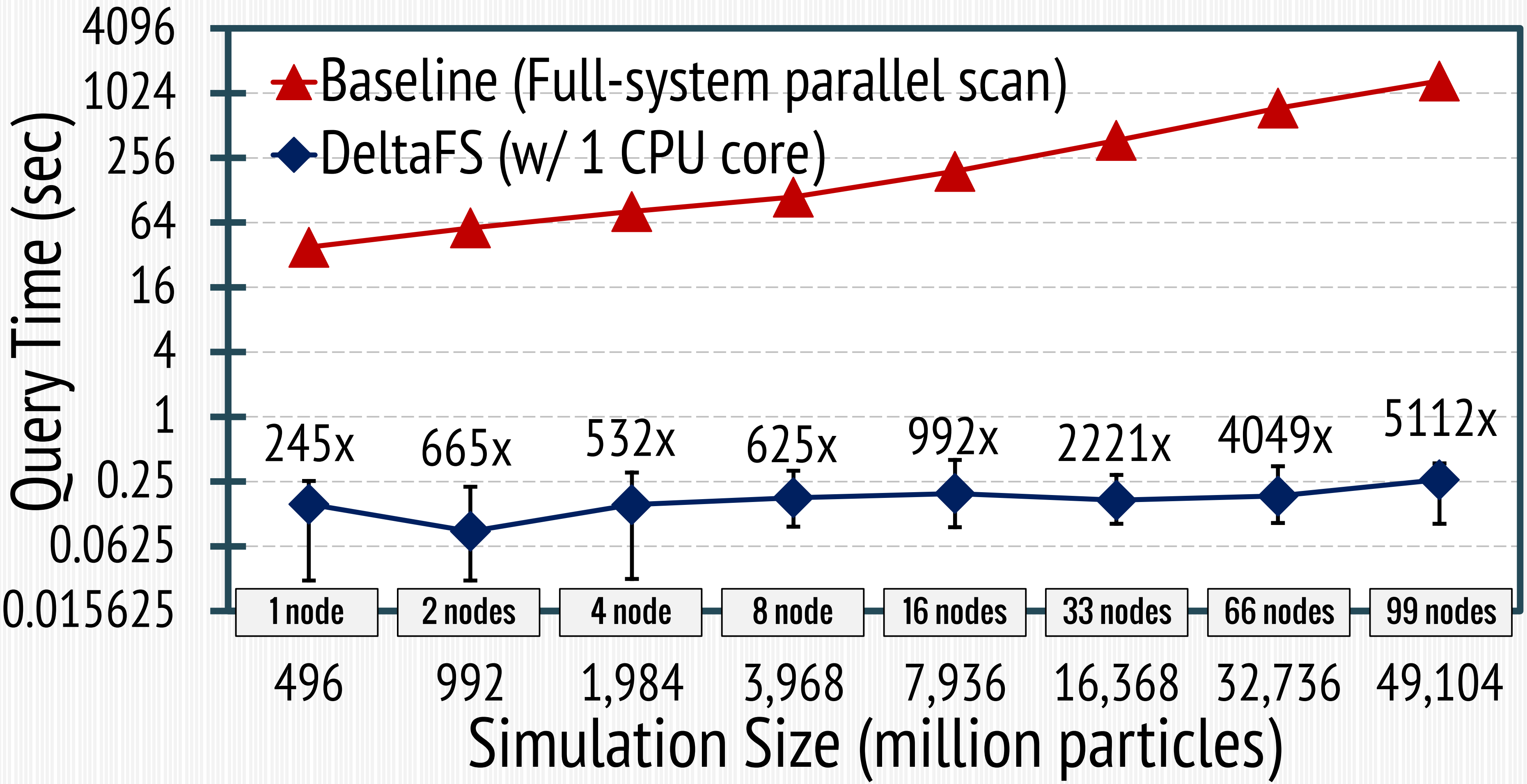
VPIC using DeltaFS IMD



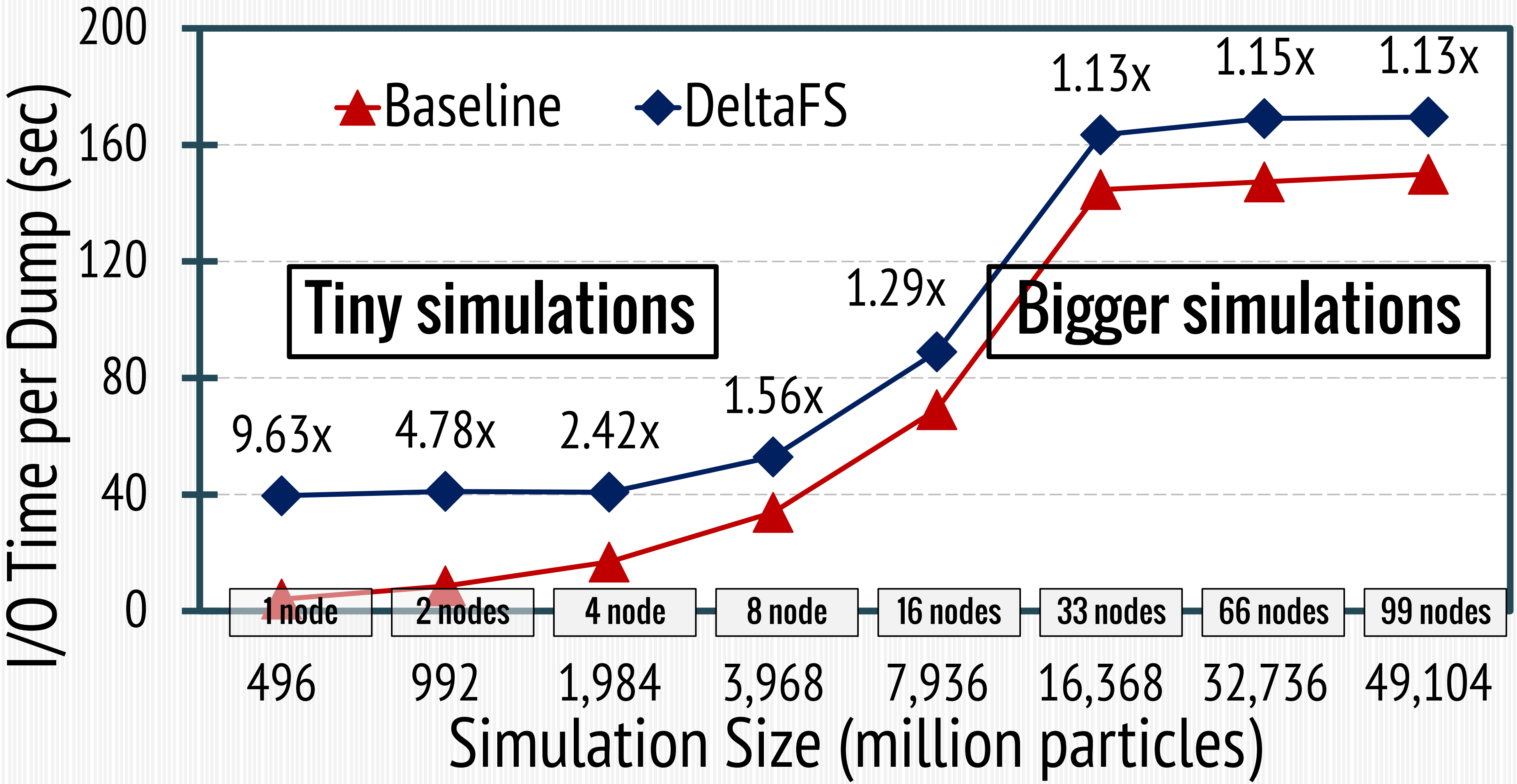
LANL Trinity Experiments



1-99 compute nodes, 496 million - 48 billion particles



Query Time (sec)
4096
1024
256
64
16
4
1
0.25
0.0625
0.015625
Baseline (Full-system parallel scan)
DeltaFS (w/ 1 CPU core)
245x
665x
532x
625x
992x
2221x
4049x
5112x
1 node
2 nodes
4 node
8 node
16 nodes
33 nodes
66 nodes
99 nodes
496
992
1,984
3,968
7,936
16,368
32,736
49,104
Simulation Size (million particles)



I/O Time per Dump (sec)
200
160
120
80
40
0
Baseline
DeltaFS
Tiny simulations
Bigger simulations
9.63x
4.78x
2.42x
1.56x
1.29x
1.13x
1.15x
1.13x
1 node
2 nodes
4 node
8 node
16 nodes
33 nodes
66 nodes
99 nodes
496
992
1,984
3,968
7,936
16,368
32,736
49,104
Simulation Size (million particles)

Conclusion

https://github.com/pdlfs/deltafs

In-situ indexing for transparent, almost-free query acceleration

no dedicated nodes, no post-processing, ~15% I/O overhead

- Indexed Massive Dir (~3% app mem, compaction-free, POSIX API)
- Powered by Mercury RPC

  https://mercury-hpc.github.io/
- DeltaFS is one of the Mochi micro-services

  https://press3.mcs.anl.gov/mochi/

Mochi