DeltaFS

Parallel In-situ Data Indexing

Qing Zheng, Garth Gibson, Brad Settlemyer,
Chuck Cranor, George Amvrosiadis, Saurabh Kadekodi, Fan Guo

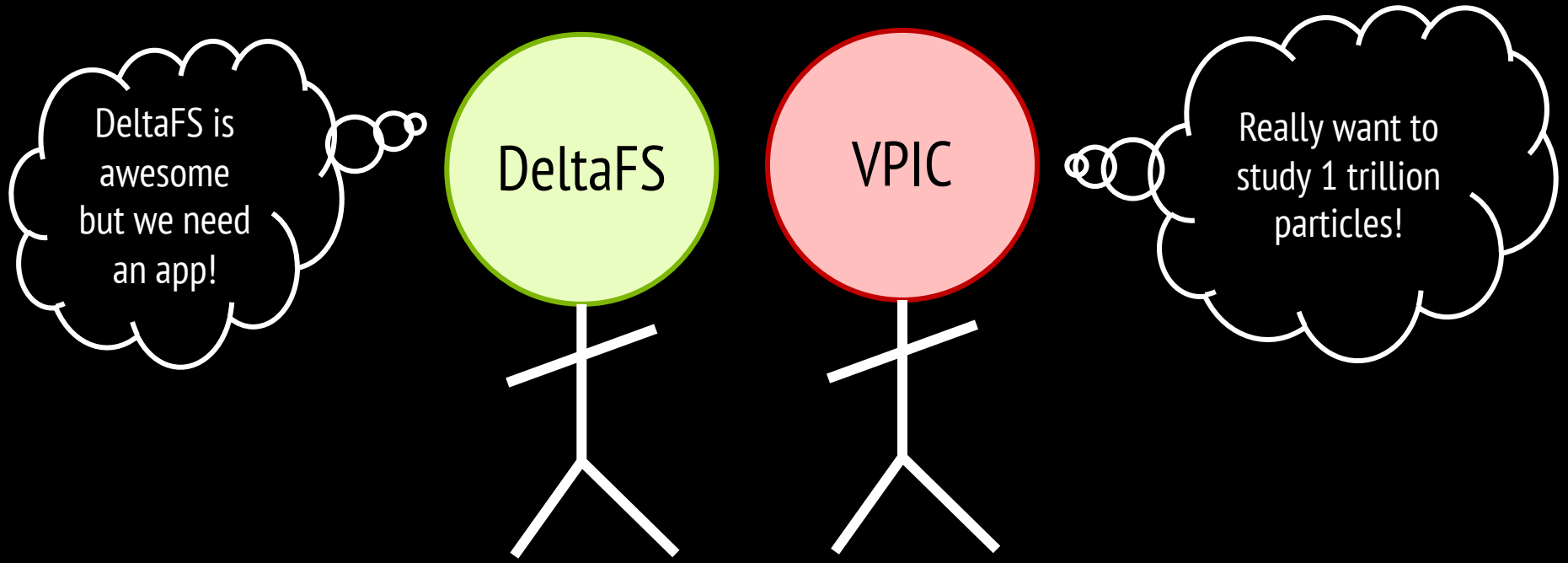
2017 USRC Annual Symposium

Carnegie Mellon University
Los Alamos National Laboratory

LA-UR-17-26515

USRC

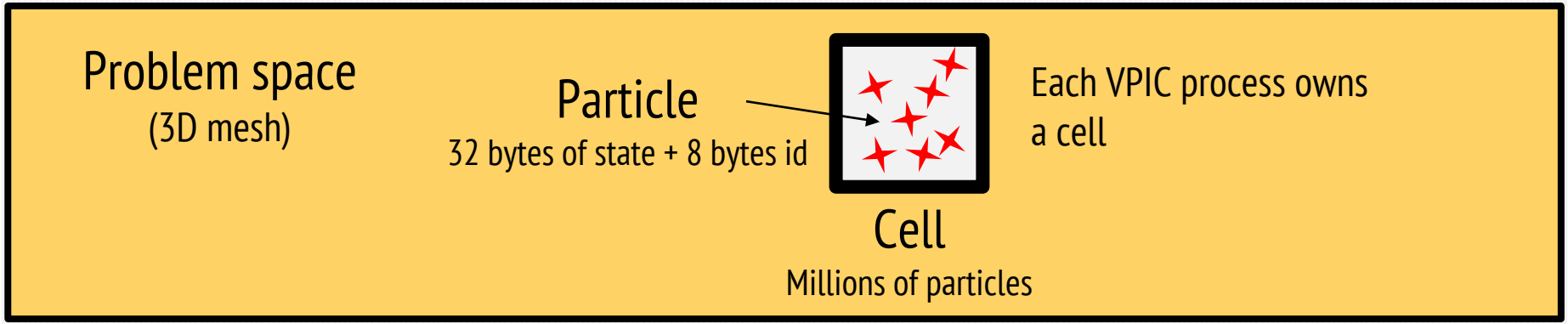
DeltaFS/VPIC "conspiracy"



What's VPIC?

One of the 4 most used scientific codebases at LANL

Extremely scalable code that simulates a massive virtual universe



- Alternates between computation and I/O for every O(N) timesteps
- Each I/O phase generates a frame with the state of all particles

Need fast trajectory query over trillions of particles

I/O challenges

- ❑ Write - utilize all I/O bandwidth (large aligned sequential writes)
- ❑ Read - query latency (time-to-science)

★ current state-of-the-art

LANL/LBNL – 1T particles - NERSC Hopper

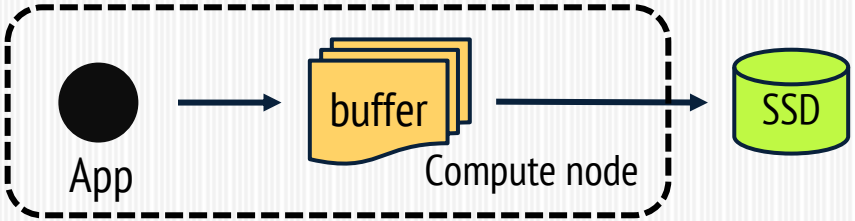
- Write: H5Part + MPI-IO + Lustre (N-1 style, 85% I/O utilization)
- Read: post-processing (via a time-consuming parallel merge-sort phase)

From N-1/post-processing to N-N/in-situ indexing

DeltaFS thinks it can do better!

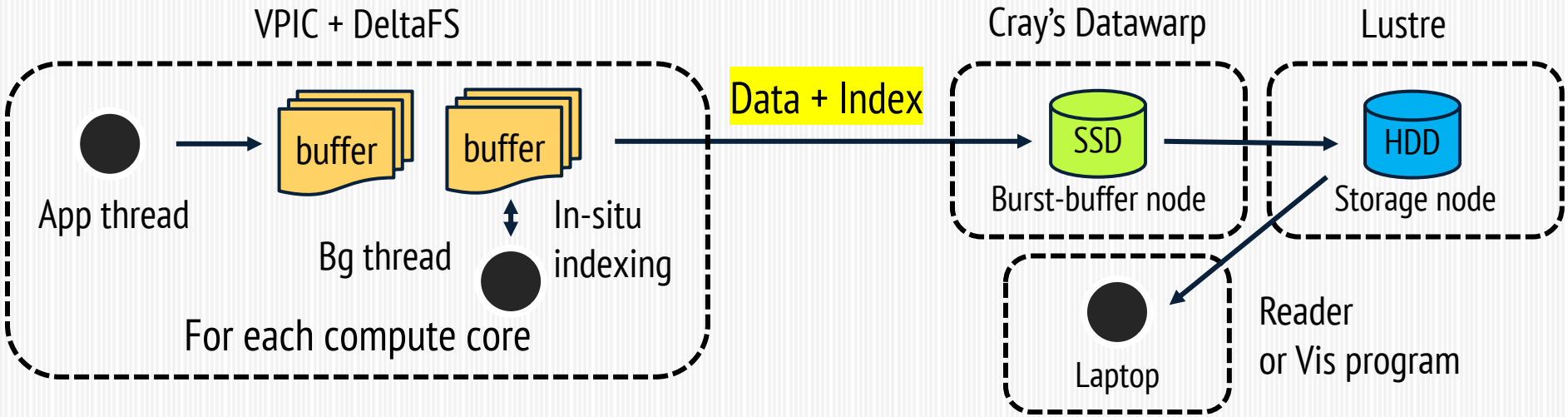
1) HDD to SSD (burst-buffer storage, more h/w bandwidth)
2) N-1 to N-N (more I/O utilization in principle, lesson learned from plfs)
3) post-processing to in-situ indexing (reduce time-to-science)

➡ Why in-situ indexing is doable?



1. I/O is the slowest part of the system
2. Thus CPU underutilized
3. Memory reserved for storage write-back buffering can be reused to serve simple in-situ data indexing
4. So we can just do it!

A simplified view of our initial system design



A file-per-process storage layout similar to plfs

```
# ls /users/qingzhen/jobs/deltafs.51165/deltafs_P16384M_C1024_N32/out/particle
L-0001.dat L-0001.idx L-0002.dat L-0002.idx L-0003.dat L-0003.idx
L-0004.dat L-0004.idx L-0005.dat L-0005.idx L-0006.dat L-0006.idx
L-0007.dat L-0007.idx L-0008.dat L-0008.idx L-0009.dat L-0009.idx
...
```

There are two problems

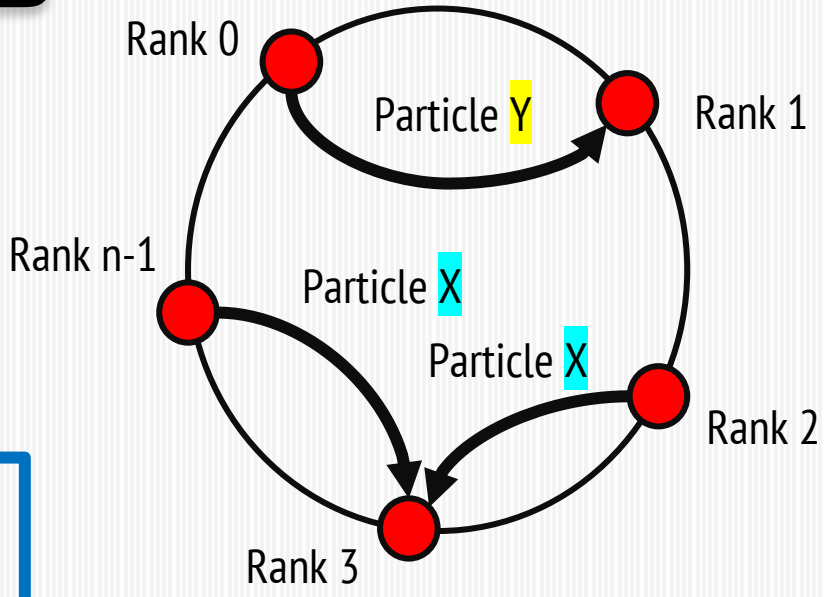
1) Need to read all .idx files to find the trajectory of any particle
2) Load imbalance: some .dat larger than others

MDHIM-style id partitioning and data shuffling

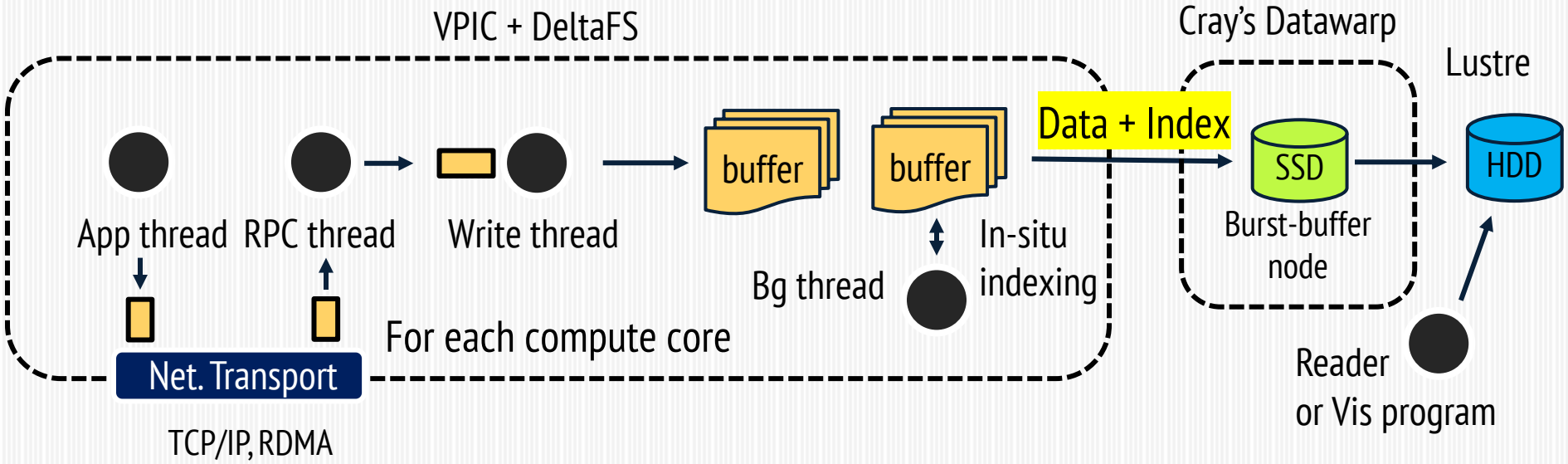
Goal: data from a single particle goes to a single core

- ✓ Reduce the amount of .idx files for each particle from O(N) to O(1)
- ✓ Also ensure load balancing across all distributed cores

With hash-based particle id partitioning, and RPC-base communication (each rank is both a client and a server)

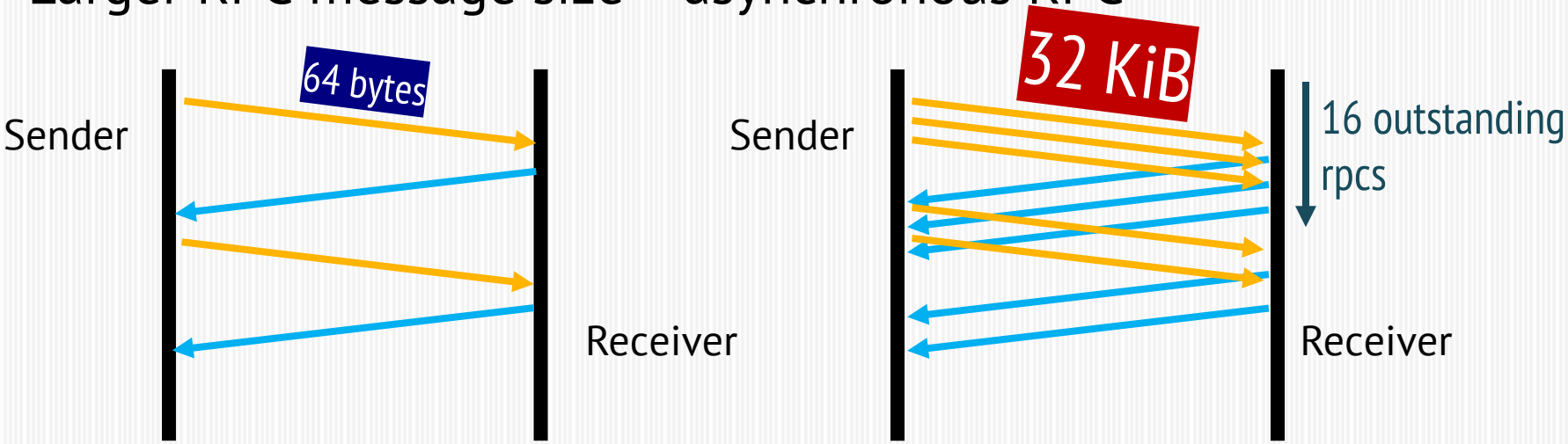


A revised design of our system



RPC buffering and flow control

Larger RPC message size + asynchronous RPC



Much higher RPC throughput, CPU utilization, and I/O utilization

Awesome performance!

	Initial code	Apr17	Jul17	Aug17
Dumping time		+64%	+35%	+26%
Storage overhead	…..	+13.5%	+8.1%	
Simulation size		16M / core		

Overhead of total simulation time

- less than 10%

Per-particle query latency

- orders of magnitude faster!

Conclusion

The rumors are true. DeltaFS is here.

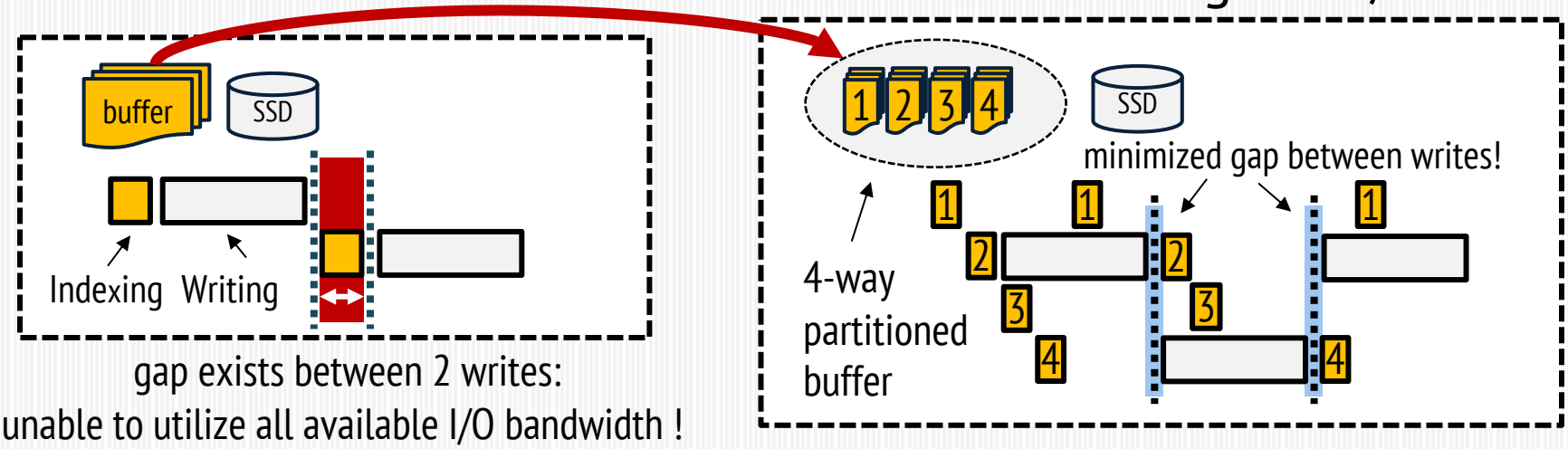
It is the new scratch file system that people keep talking about throughout the summer.

And with less than 10% overhead in writing and 1,000x+ reduction in time to science,

you are gonna love it!

Better threading structure

Partition write buffer to avoid serialized indexing and I/O

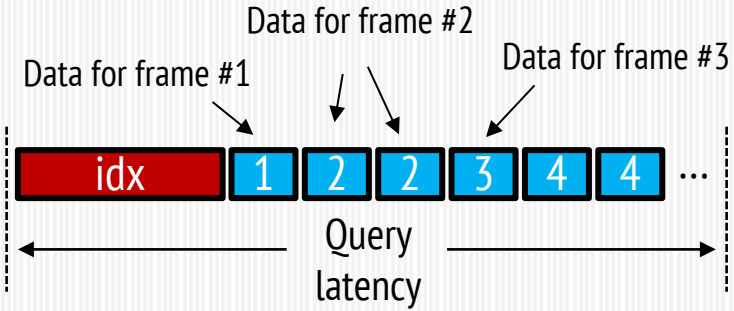


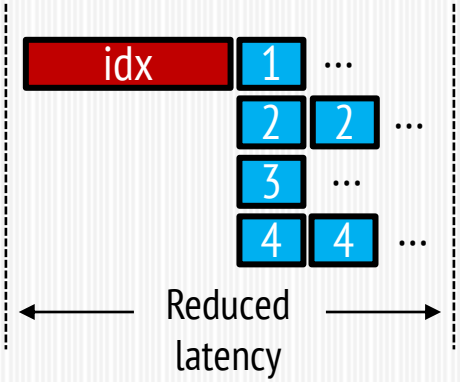
Ensure full burst-buffer bandwidth utilization

Parallel data retrieval

2 phase reading:

- **O(1)** sequential index read + **O(N)** random data fetches





Further reduce particle trajectory query latency