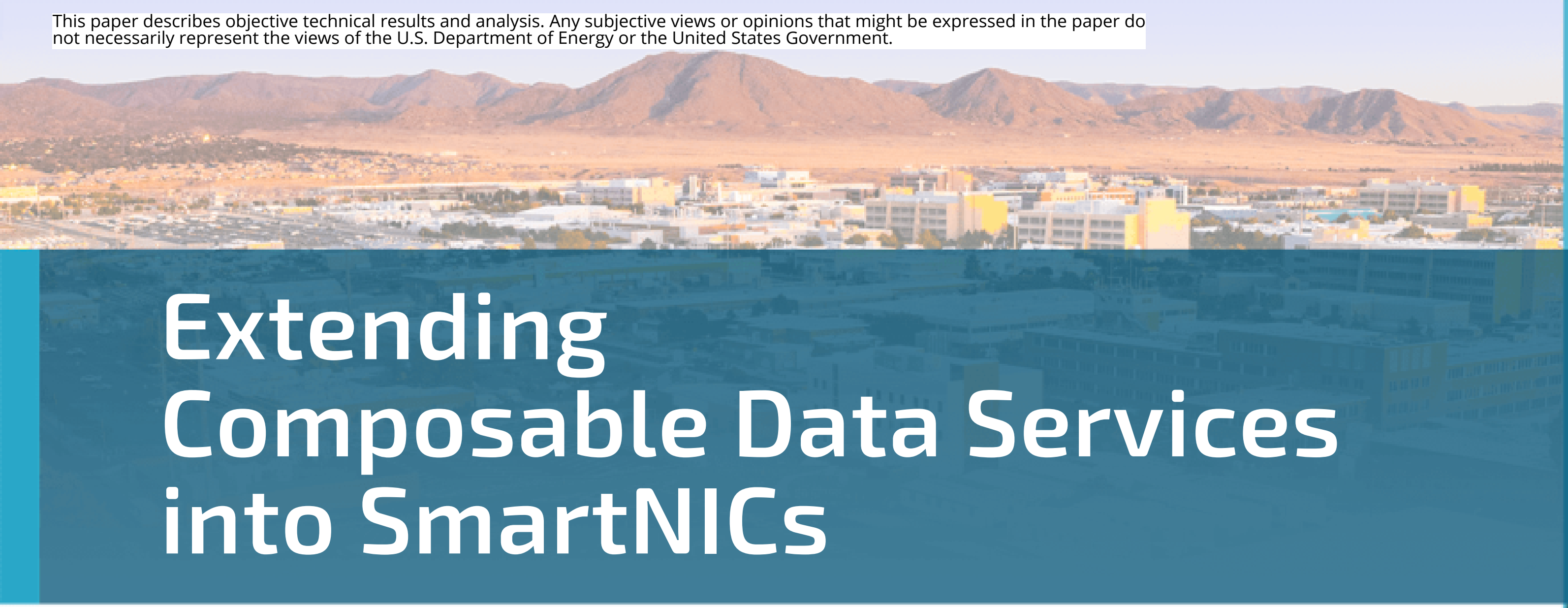
SAND2023-03752C



Extending Composable Data Services into SmartNICs

Craig Ulmer, Sandia National Labs, California

Jianshen Liu & Carlos Maltzahn, University of California Santa Cruz

Matthew L. Curry, Sandia National Labs, New Mexico

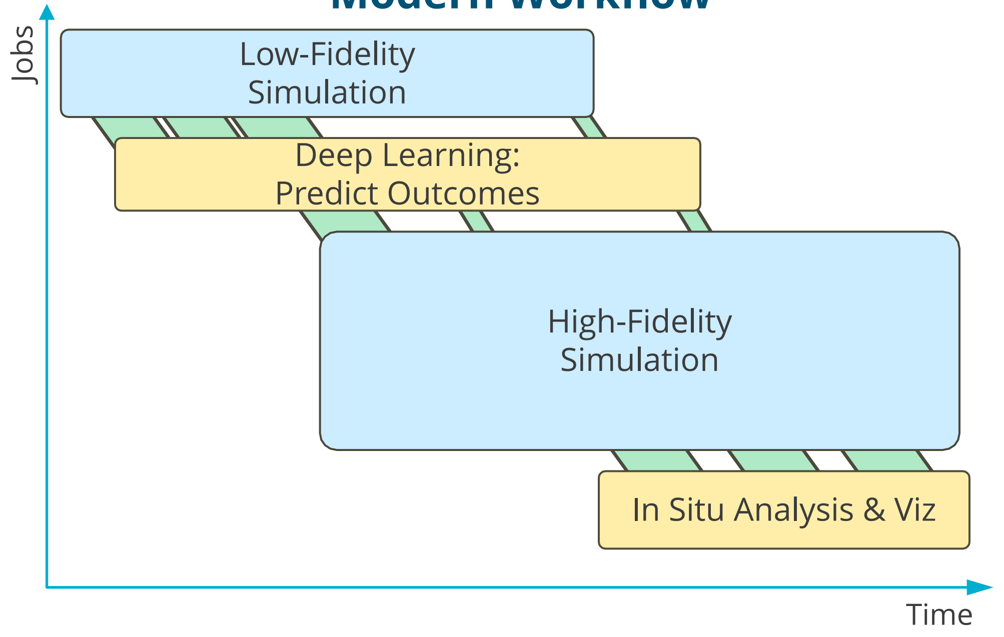
This material is based upon work supported by the U.S. Department of Energy, Office of

Proposal Number 20-023266.

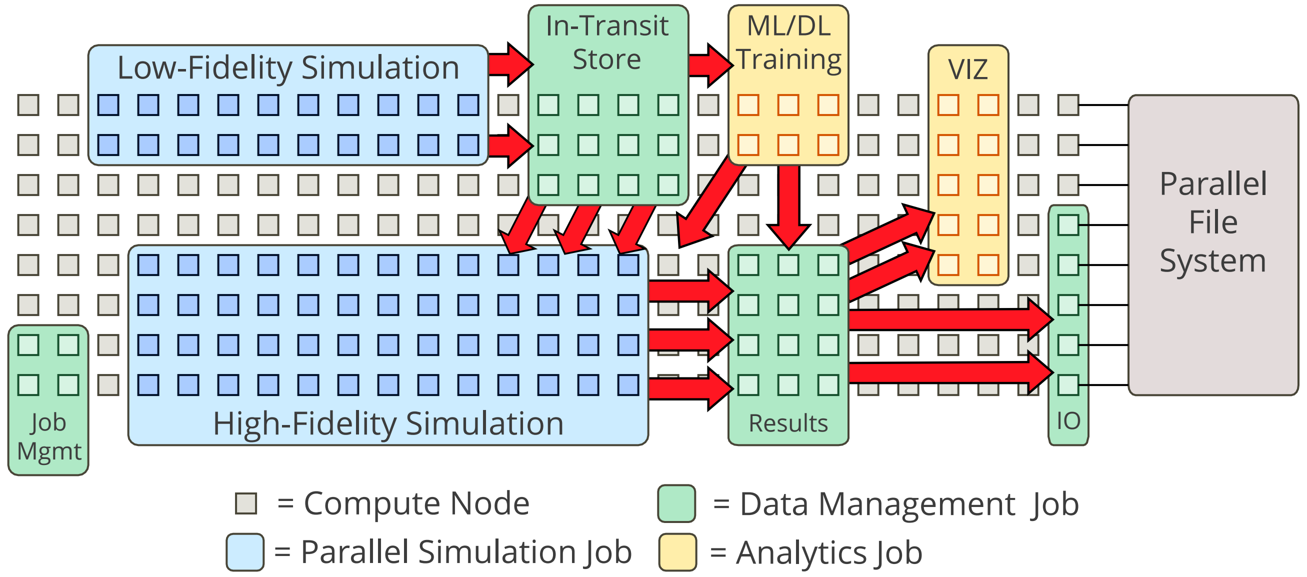
Background: High-Performance Computing Workflows

- Scientific Computing workflows involve multiple applications that run in parallel
- *Composable Data Services* responsible for moving data between applications
- ***Problem***: Data services consume compute-node resources

Modern Workflow



Parallel Jobs on HPC Platform



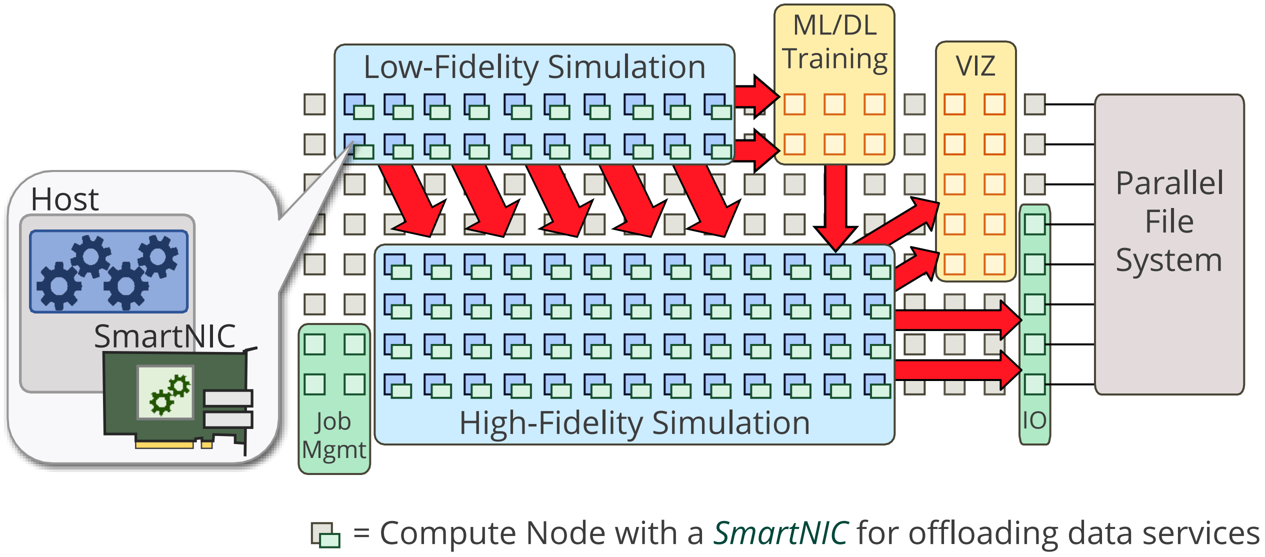
Smart Network Interface Cards (SmartNICs)

- Network vendors now offer SmartNICs with *user-programmable* resources
  - Example: NVIDIA BlueField-2 DPU
  - Embedded processors are an order of magnitude slower than hosts
  - Isolated space for caching and processing in-transit data
- Emerging HPC platforms include SmartNICs
  - How do we make an environment for hosting data services in SmartNICs?

BlueField-2 DPU

- 100Gb/s InfiniBand
- 8 Arm Cores
- 16GB DRAM
- 60GB Flash

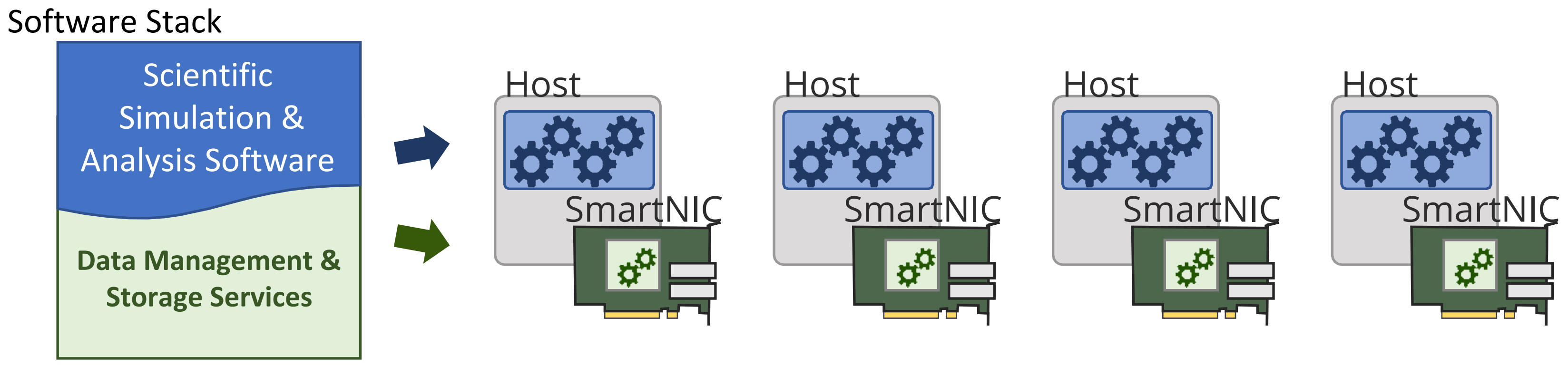
HPC Platform with SmartNICs



Creating an Environment for Data Services on SmartNICs

Create an Environment for Hosting Data Services on SmartNICs

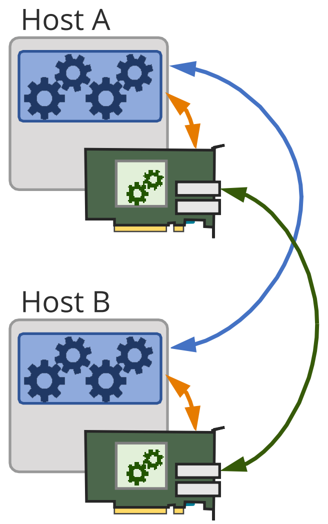
- We define five requirements (R1-R5) for creating this environment
  - Three communication, Two computation
- Existing composable data service libraries for hosts are a good starting point
  - High-level APIs: Remote Procedure Calls, Key/Value stores, Async Tasking, RDMA primitives
- Prototype environment
  - Communication via **Faodel**: C++ library with distributed-memory Key/Blob API built on RDMA
  - Computation via **Apache Arrow**: C++ library for processing in-memory tabular data


Software Stack
Scientific Simulation & Analysis Software
Data Management & Storage Services
Host
SmartNIC
Host
SmartNIC
Host
SmartNIC
Host
SmartNIC

Resolving Communication Requirements with Faodel

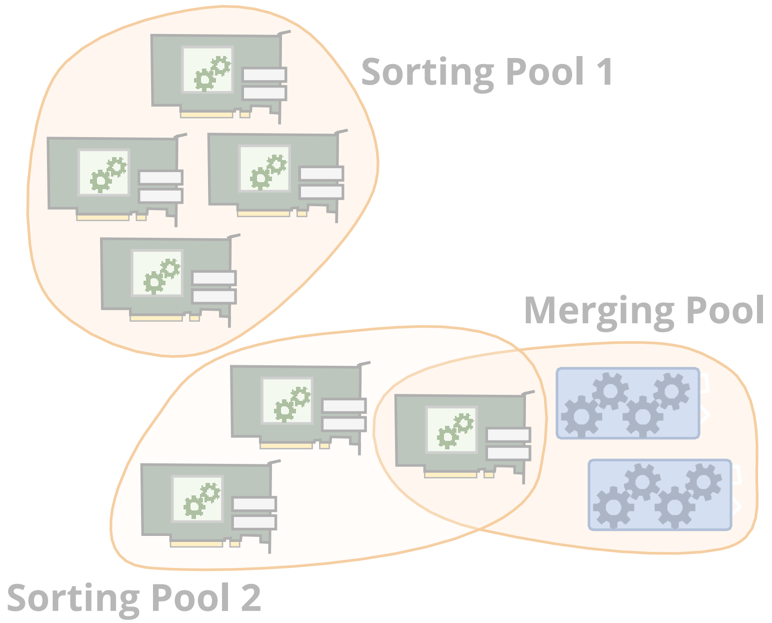
R1: Any-to-Any Transfers

- Faodel has globally accessible endpoints
- Host and SmartNICs can be endpoints
- Put/Get remote objects
- RDMA for point-to-point transfers



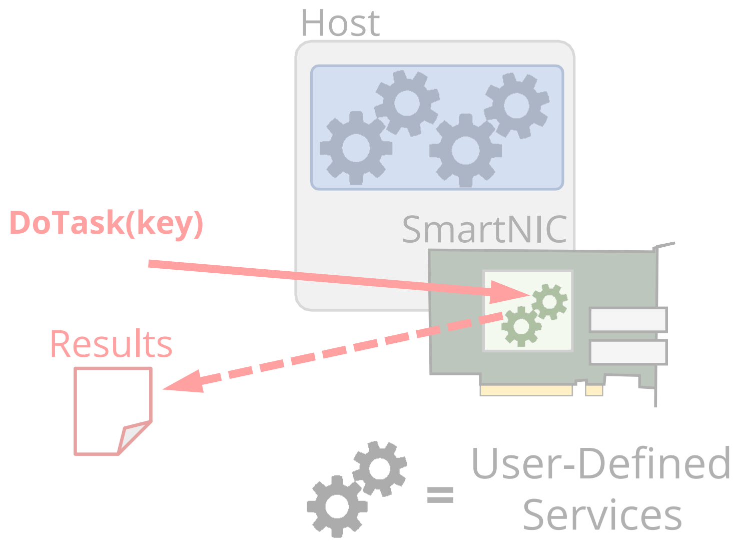
R2: Group Resources

- Faodel uses a Pool abstraction
- Pool has endpoints & distribution policy
- Work mapped to resources at run time



R3: Dispatch Computations

- Faodel primarily moves data
- Invoke remote operation on object
- Local main can also make decisions

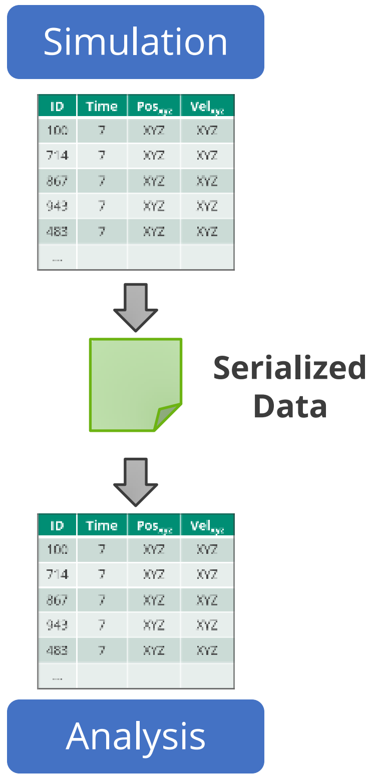


Resolving Computational Requirements with Apache Arrow

R4: Common Data Representation

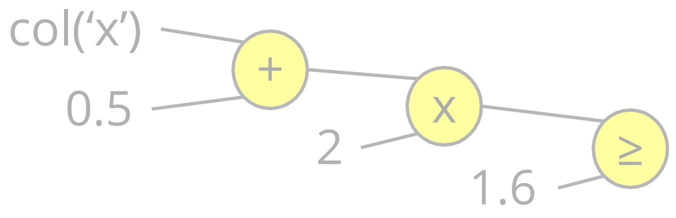
- Arrow provides robust data structures for 2D data
- Efficient in-memory storage
- Built-in functions to serialize

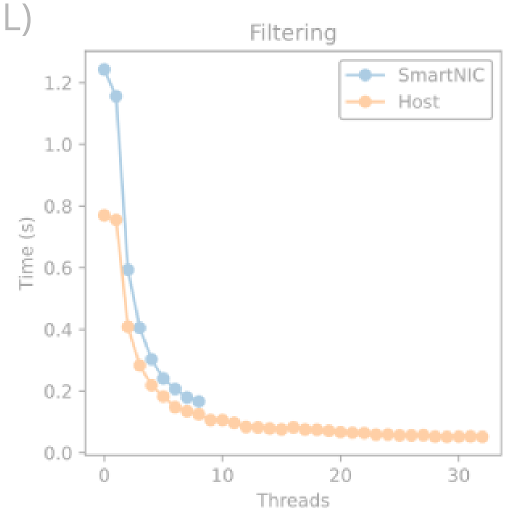
ID	Time	Pos_{xyz}	Vel_{xyz}
100	7	XYZ	XYZ
714	7	XYZ	XYZ
867	7	XYZ	XYZ
943	7	XYZ	XYZ
483	7	XYZ	XYZ
…			



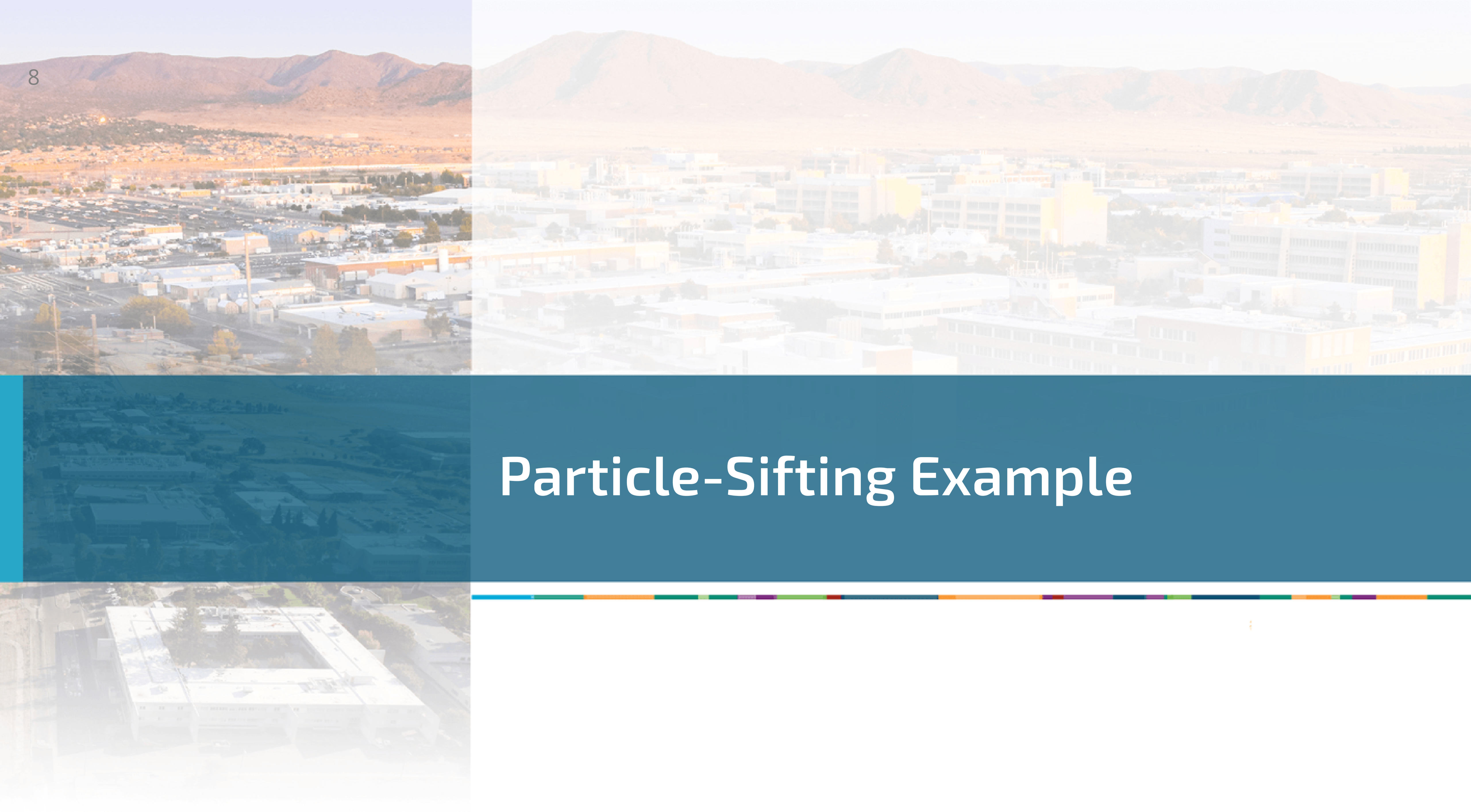
R5: Data-Parallel Computations

- Arrow includes compute functions for tables
- Target for higher-level languages (SQL)
- Thread- and SIMD-Aware





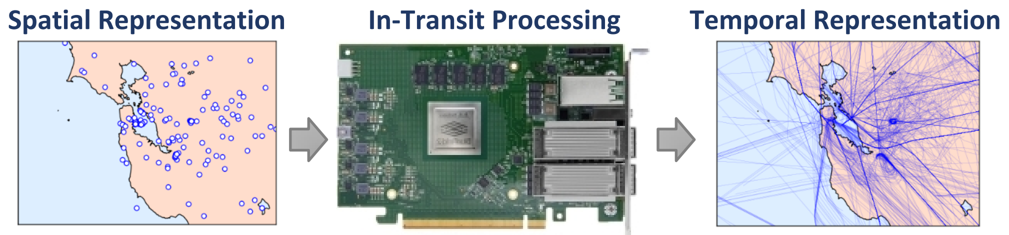
```
// 2 * (0.5 + x) >= 1.6
auto filter_expression = arrow::compute::greater_equal(
    arrow::compute::call(
        "multiply",
        {arrow::compute::literal(2),
         arrow::compute::call("add_checked", {arrow::compute::literal(0.5),
                                              arrow::compute::field_ref("x")})}),
    arrow::compute::literal(1.6));
```



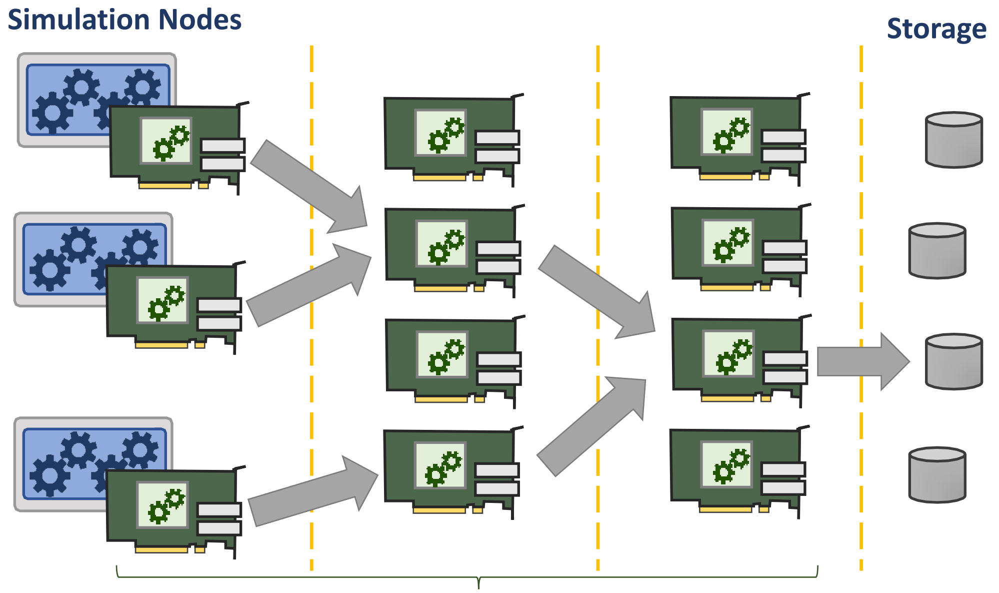
Particle-Sifting Example

Example: Reorganizing Particle Simulation Results

- Particle simulations track billions of particles
- Mismatch between producers/consumers
  - Simulations: Sorted by position and time
  - Analytics: Sorted by ID and time
- Particle sifting service
  - Periodically sample current data
  - Use distributed SmartNICs to reorganize
  - Log-structured merge (LSM) tree sorts data by ID
- Implementation
  - Faodel Pools/Keys to control data flow
  - Arrow compute to split data
- Experiments on 100-node Cluster w/ BlueField-2 DPUs



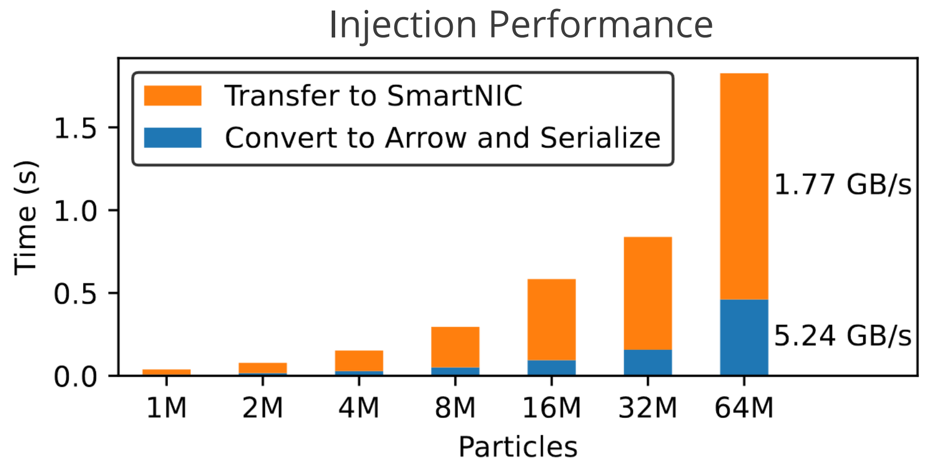
SmartNICs enable simulation results to be transformed while in transit to storage.

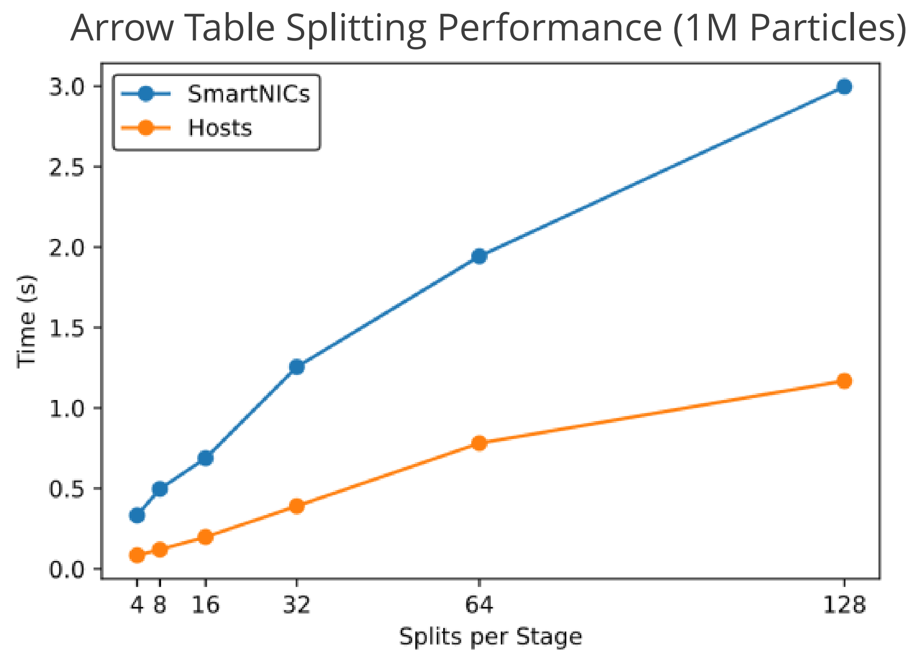


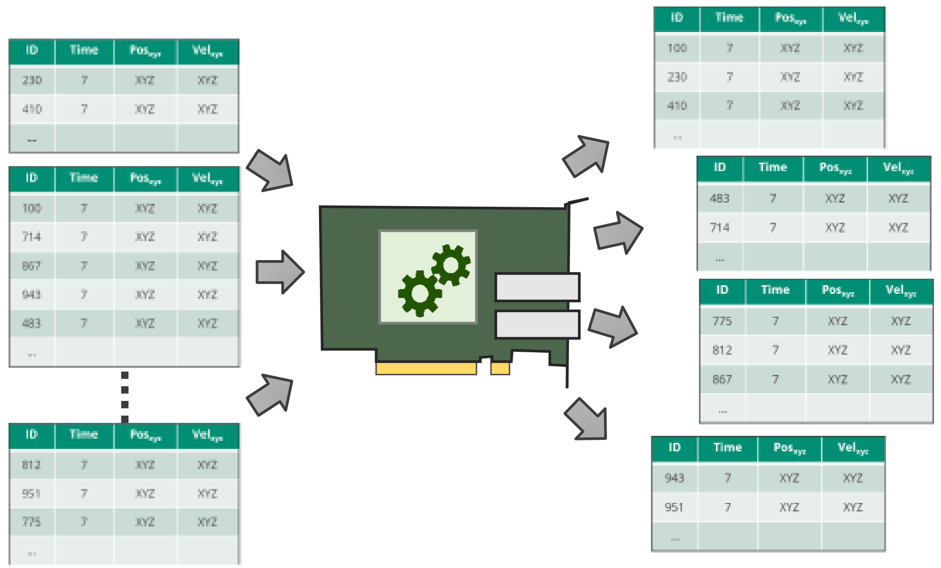
SmartNIC Particle Sifting via Log-Structured Merge Tree

Performance Measurements

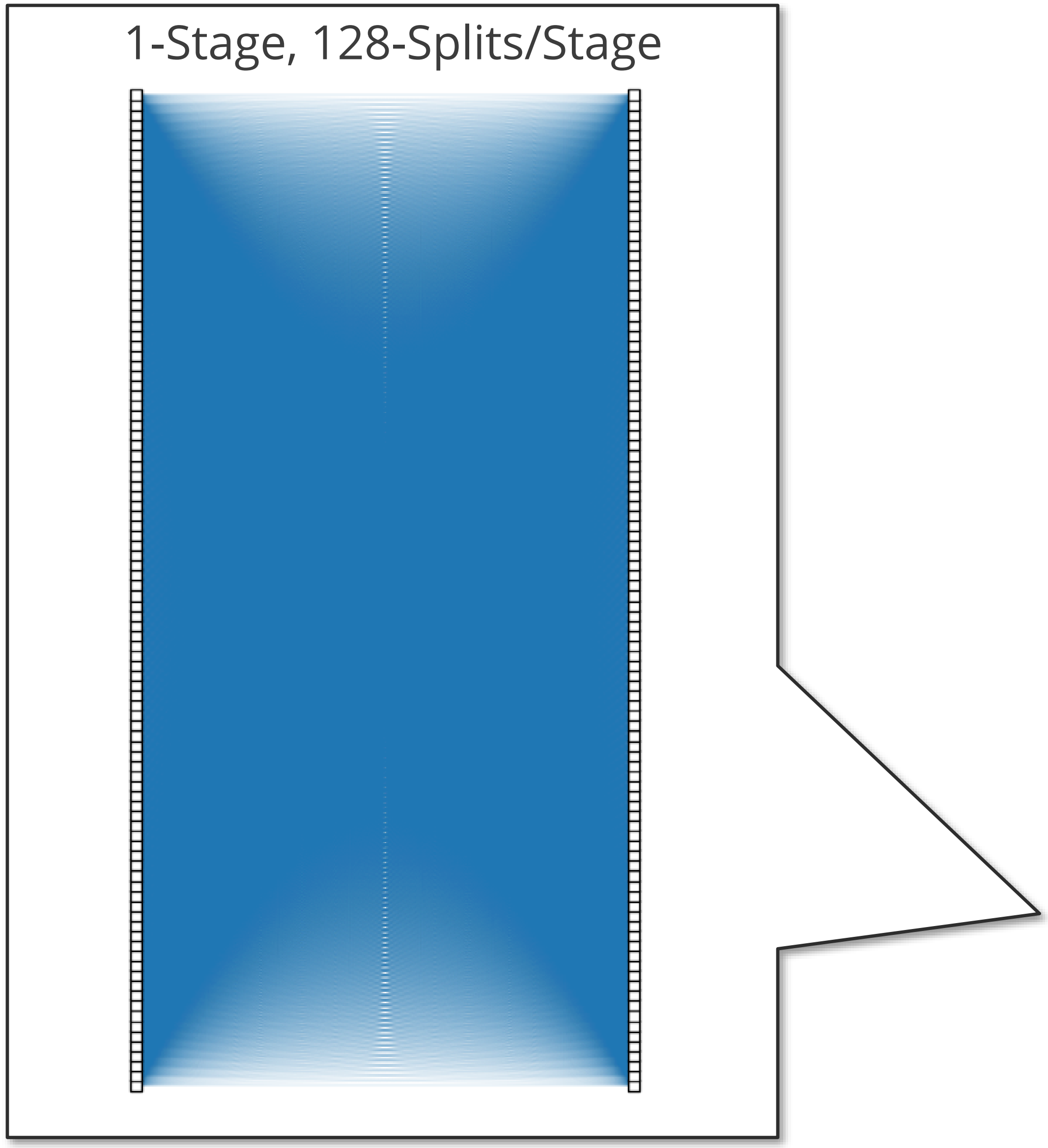
- Injection
  - Convert to Apache Arrow's serialized IPC format
  - Transfer to local SmartNIC
  - 1M–64M Particles (37MB-2.4GB), Overall: 1.32GB/s
- Splitting Tables
  - Merge incoming tables and split based on particle IDs
  - Implemented with Arrow Compute function

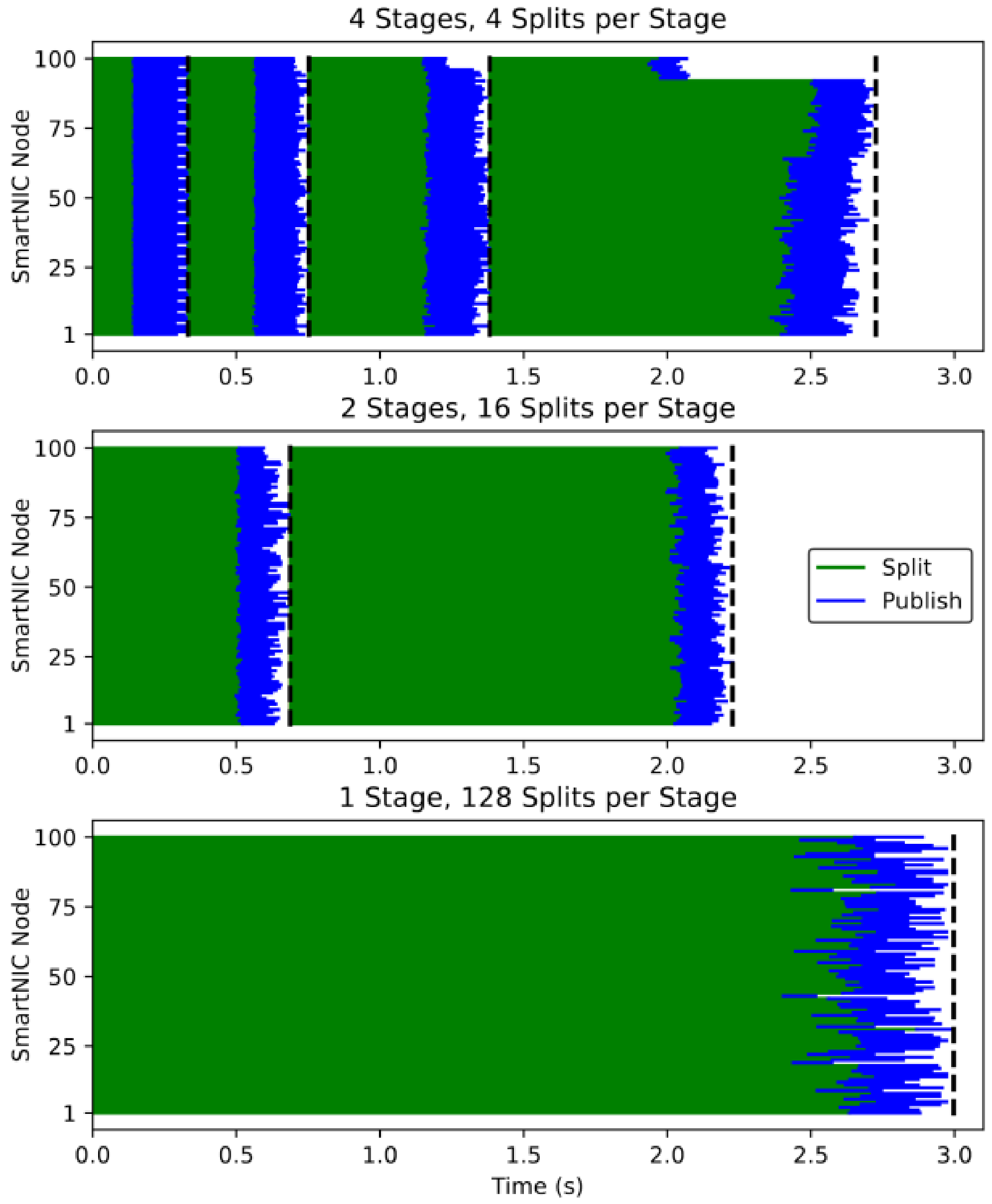




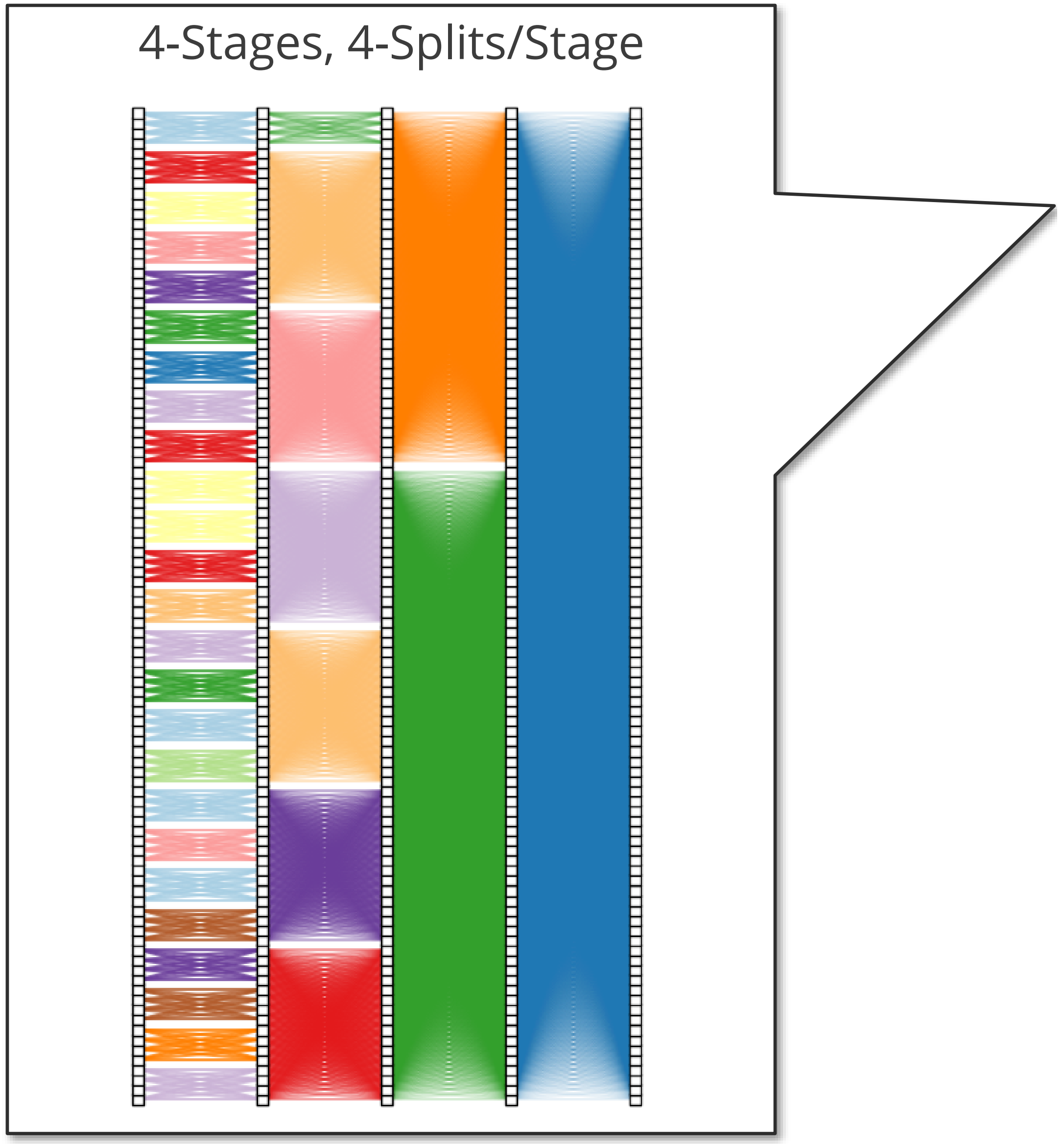


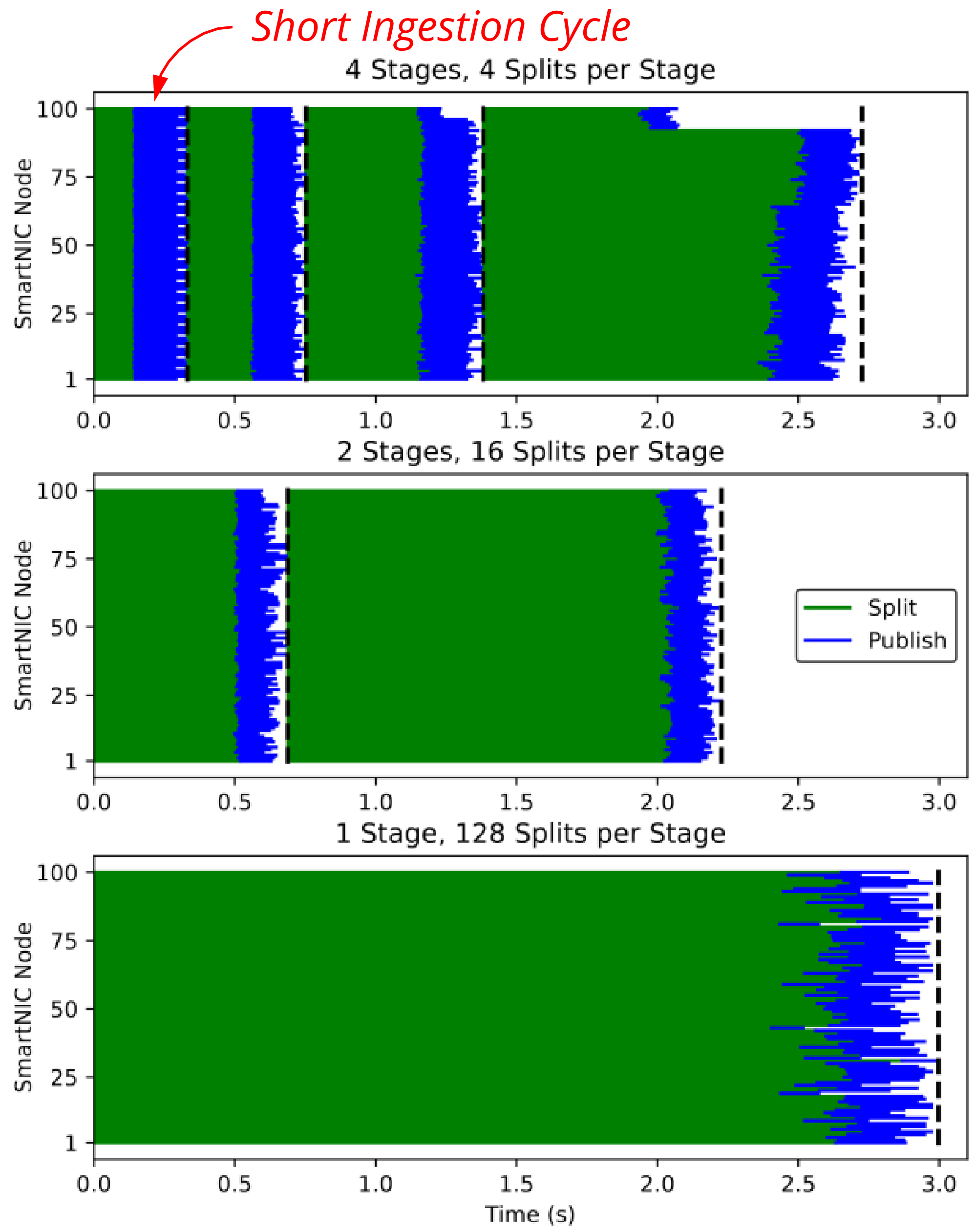
Overall Sifting Performance: 100M Particles on 100 SmartNICs



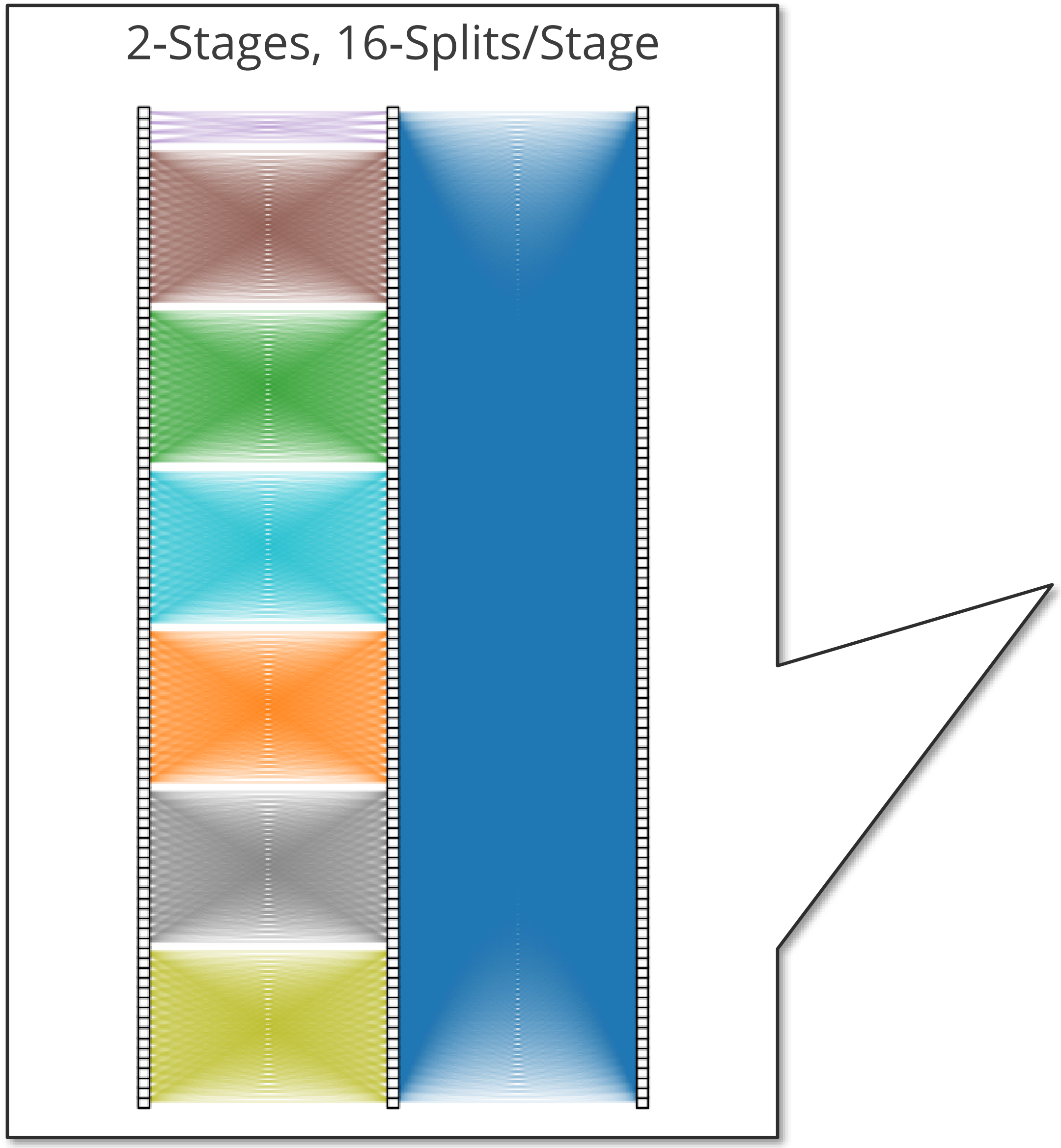


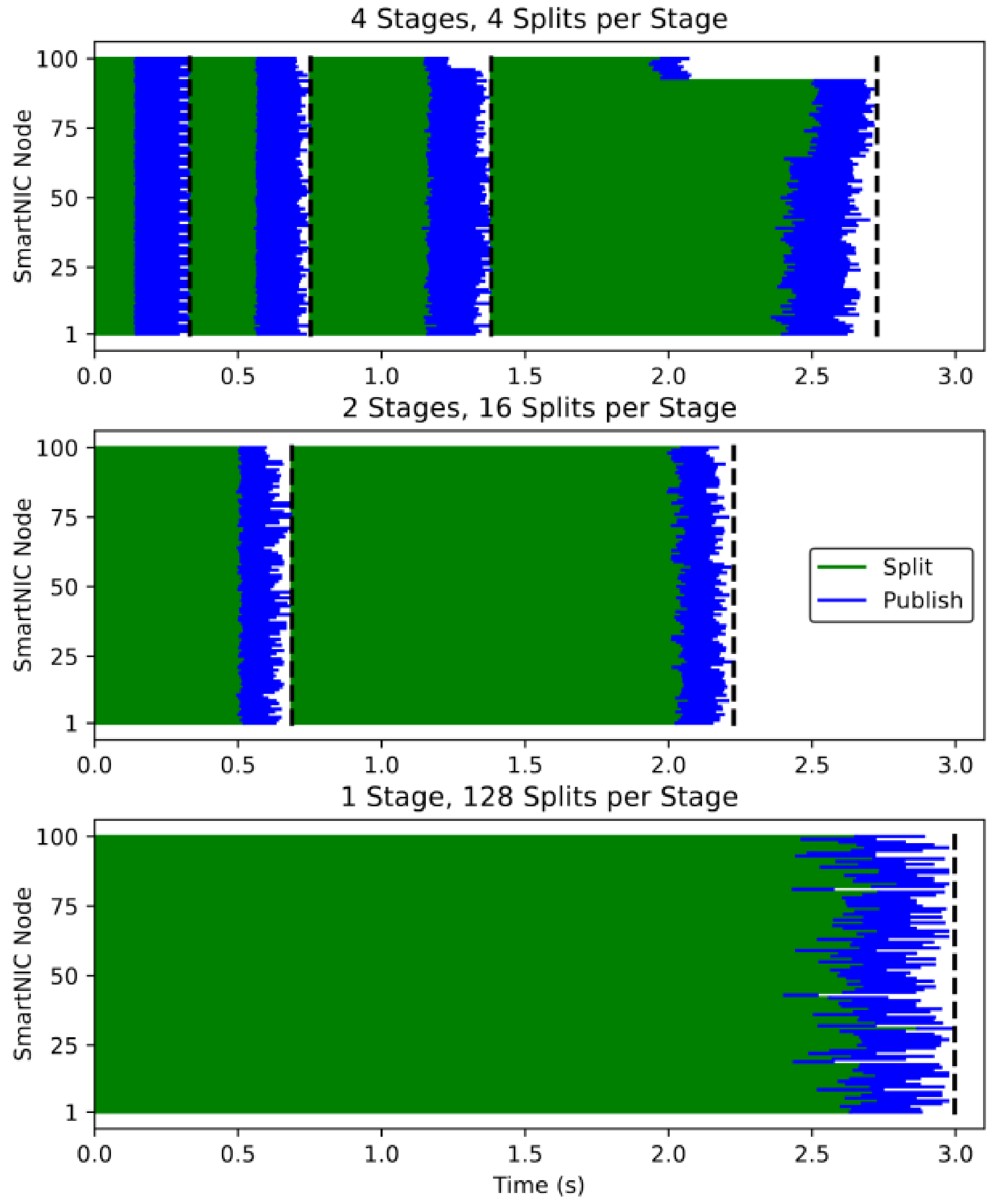
Overall Sifting Performance: 100M Particles on 100 SmartNICs





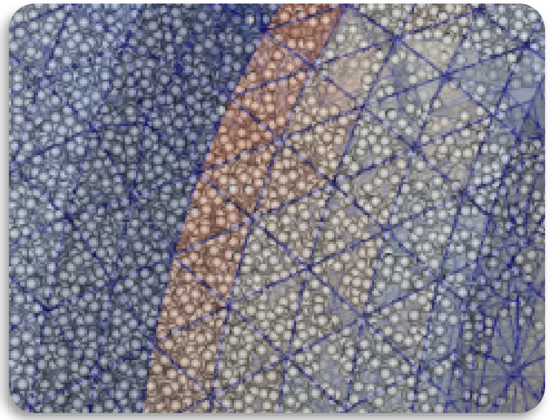
Overall Sifting Performance: 100M Particles on 100 SmartNICs





Summary and Future Work

- SmartNICs offer a new space for hosting data management services
  - Positive: Isolated space for operations near producers
  - Negative: Host processors 4x faster, Vendor-specific libraries, extra costs ($, power)
- Can build a functional environment for hosting services from existing libraries
  - Faodel offers flexible primitives for workflows
  - Apache Arrow simplified development and leveraged parallel hardware
- Future directions
  - Improving injection performance through DOCA and serialization pipelining
  - Embedding query engines in SmartNICs to support push-down queries
  - Evaluate emerging BlueField-3 hardware



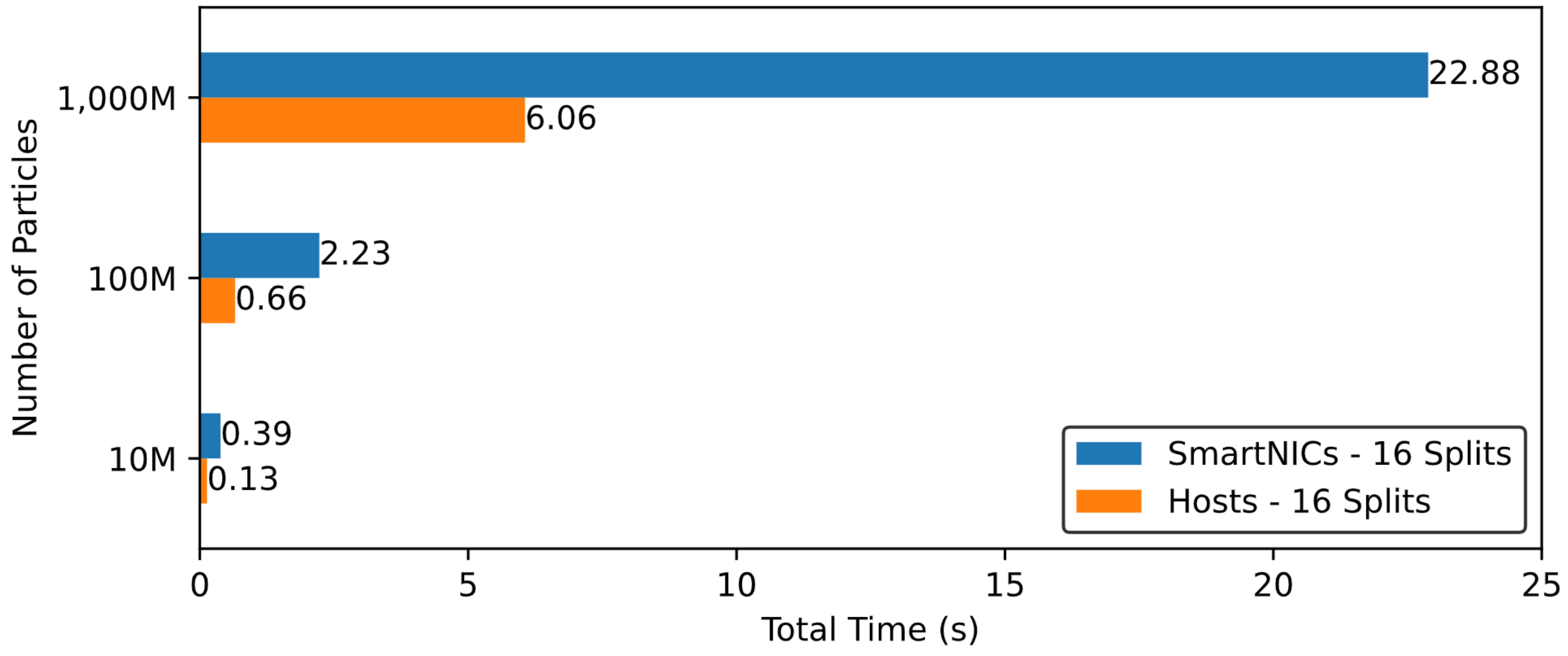
https://github.com/faodel/faodel

https://github.com/apache/arrow

Extras

SmartNIC vs Host Performance for Particle Sifting

- Scaled number of particles from 10M to 1B
    - Selected the best approach for each implementation
    - Hosts roughly 3-4x faster than SmartNICs



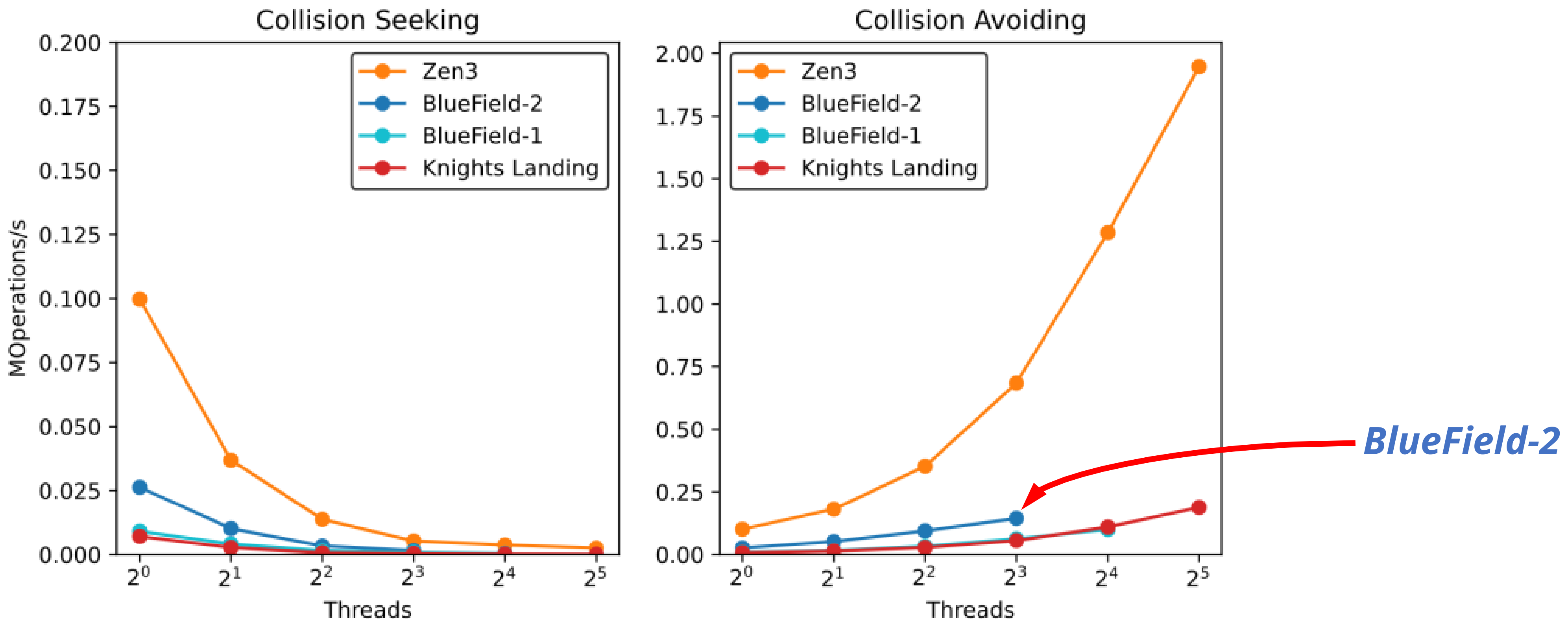
FAODEL Stress Experiments

- How are data management tasks impacted by embedded processors?
  - Stress-ng benchmark inspired us to create **faodel-stress** tool
  - Generating/sorting keys, serializing data, allocating network memory, hash maps, …
  - Compared BlueField to a variety of servers used today in HPC
- Examples Local Key/Blob store

Processor	Year	Architecture	Cores	Frequency	Memory
Zen3	2021	AMD EPYC 7543p	1x32	2.8 GHz	512 GB
BlueField-2	2021	ARMv8 A72	1x8	2.5 GHz	16 GB
BlueField-1	2018	ARMv8 A72	1x16	800 MHz	16 GB
Knights Landing (KNL)	2016	Intel Phi 7250	1x68	1.4 GHz	16+96 GB

FAODEL Stress Experiments: In-memory Key/Blob Store

- Data structure for organizing objects and scaffolding for event-driven operations
  - Perform put/get/drop operations in rapid succession
  - Use key names that either create or avoid contention



Takeaway: BF2 actually faster than some data-parallel processors.