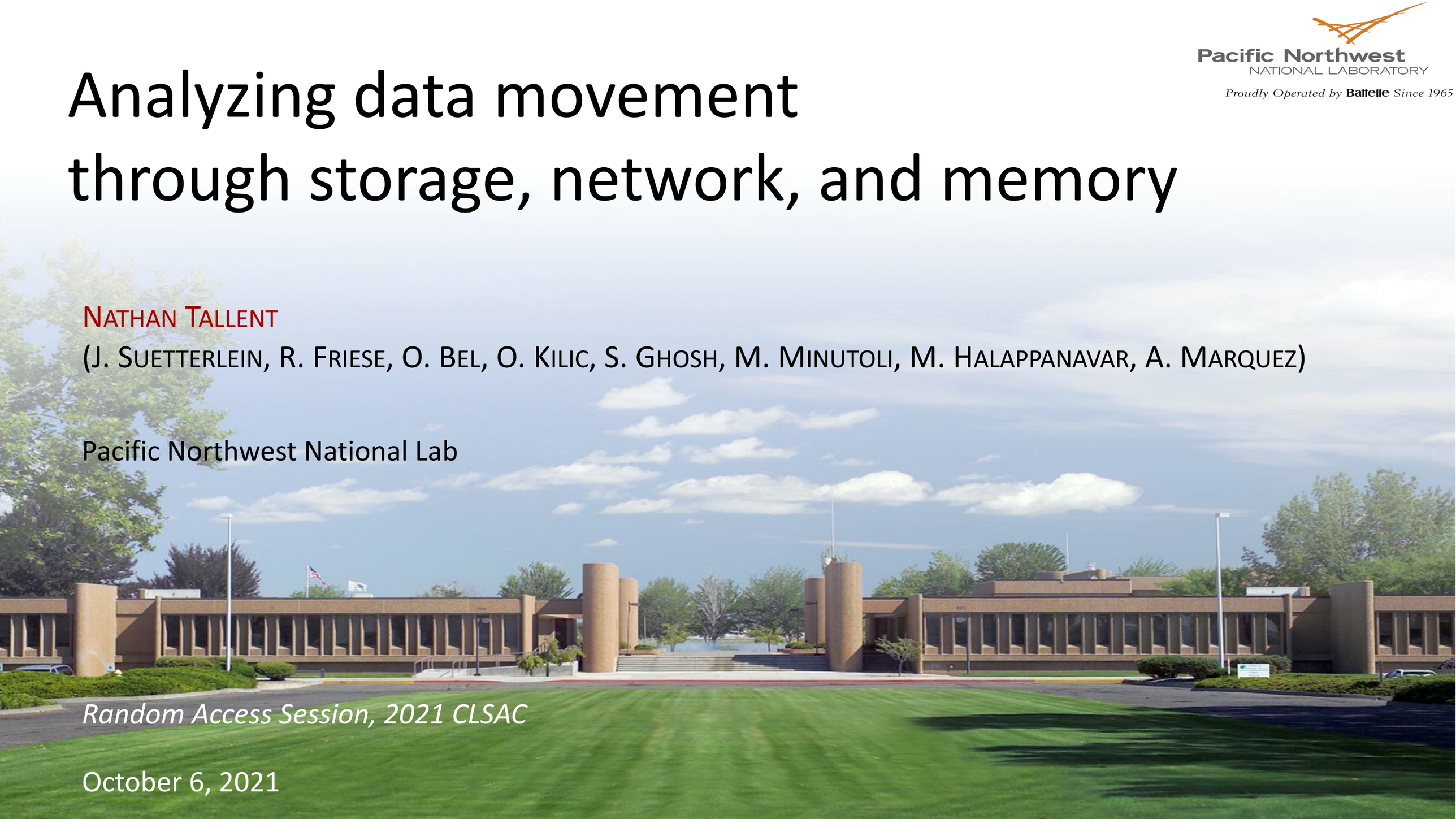
Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by **Battelle** *Since 1965*

Analyzing data movement through storage, network, and memory

NATHAN TALLENT

(J. SUETTERLEIN, R. FRIESE, O. BEL, O. KILIC, S. GHOSH, M. MINUTOLI, M. HALAPPANAVAR, A. MARQUEZ)

Pacific Northwest National Lab

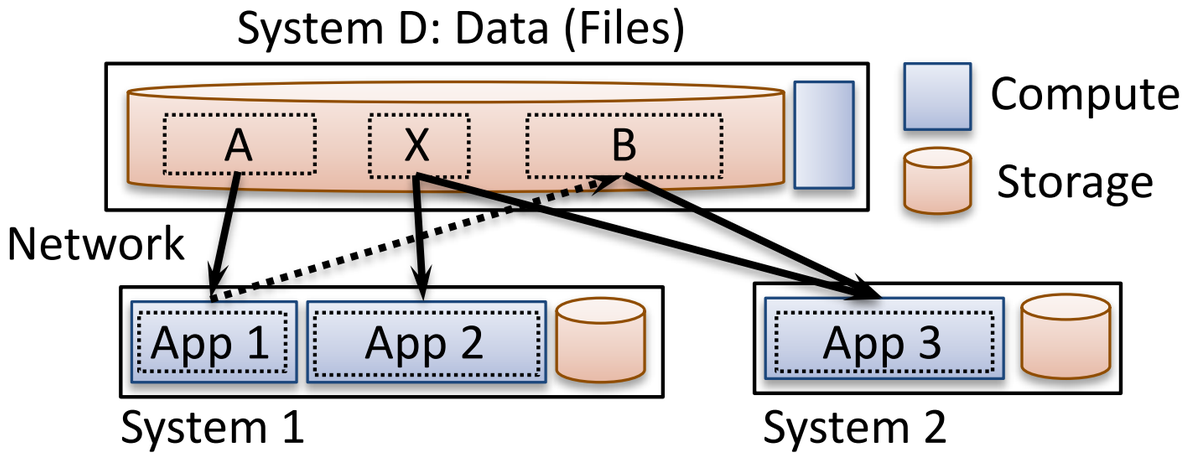


Random Access Session, 2021 CLSAC

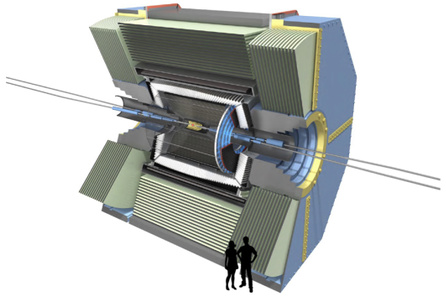
October 6, 2021

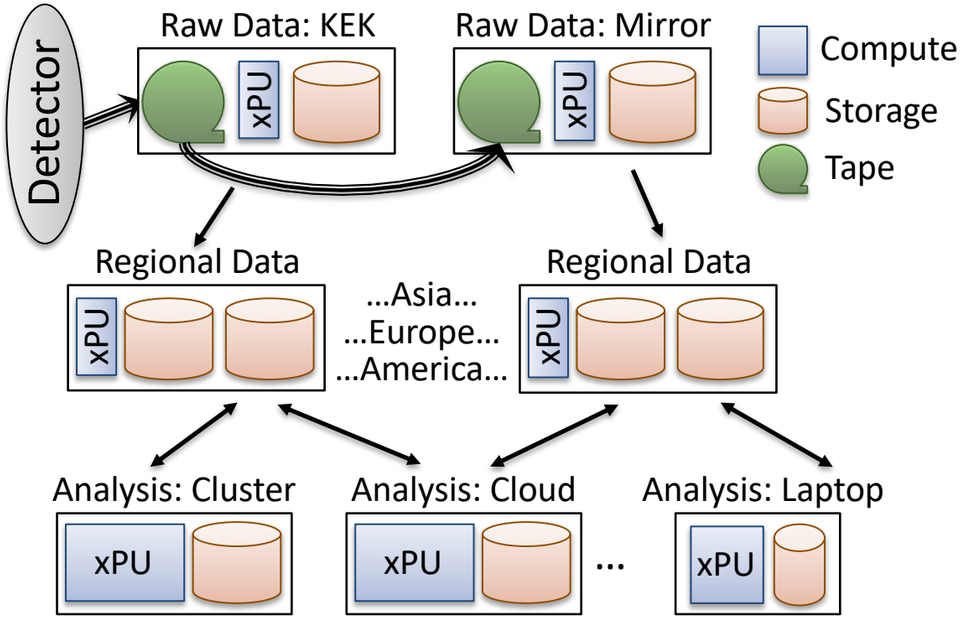
Scientific Exploration Is Increasingly Distributed & Data-intensive

- Domain science uses *workflows*:
  - Loose composition of different apps/tasks
  - Potentially different programming models
  - Data sources are distributed
  - Distributed I/O: productive method for task composition/communication



- Belle II Monte Carlo Sim: Big Data + Big Compute
  - Many logically independent tasks
  - Read intensive: inputs >> outputs
  - Simple data consistency: no files in read/write mode
  - Non-streaming accesses
  - Some inputs may be common





TAZeR: Transparent Asynchronous Zero-copy Remote I/O

Suetterlein, Friese, Tallent, and Schram, "TAZeR: Hiding the cost of remote I/O in distributed scientific workflows" BigData 2019

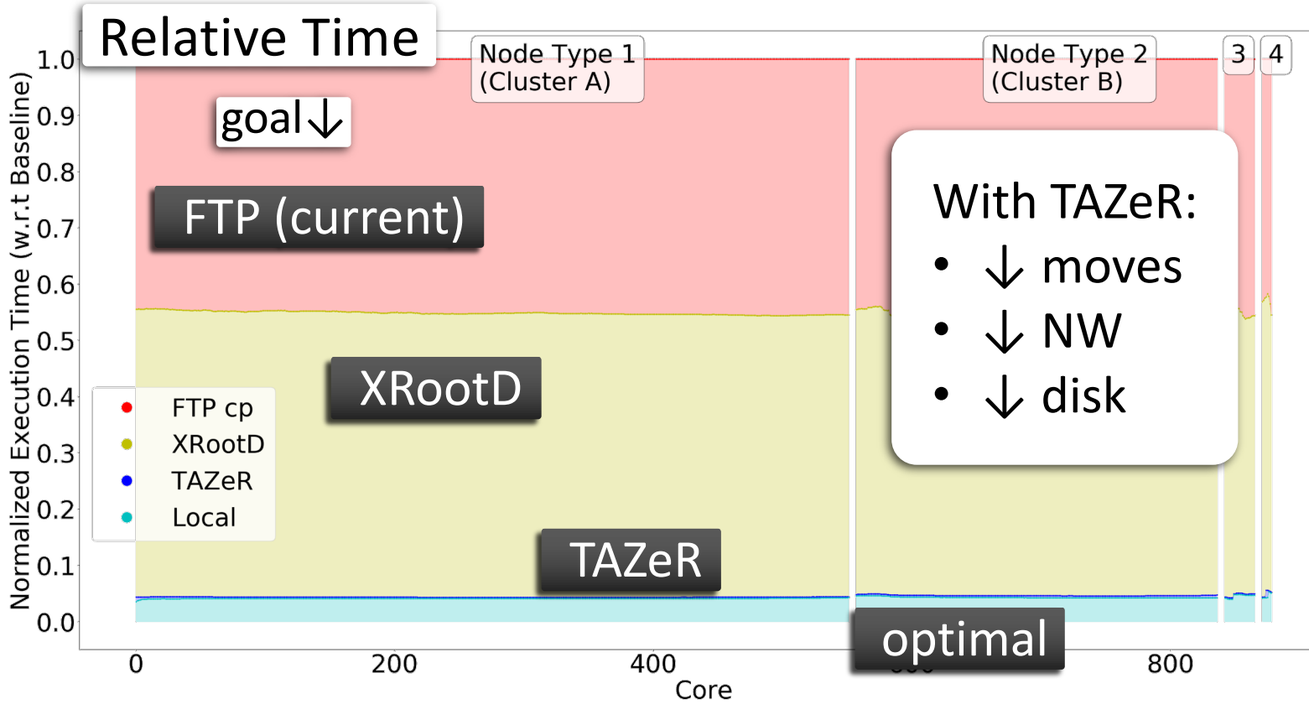
PNNL

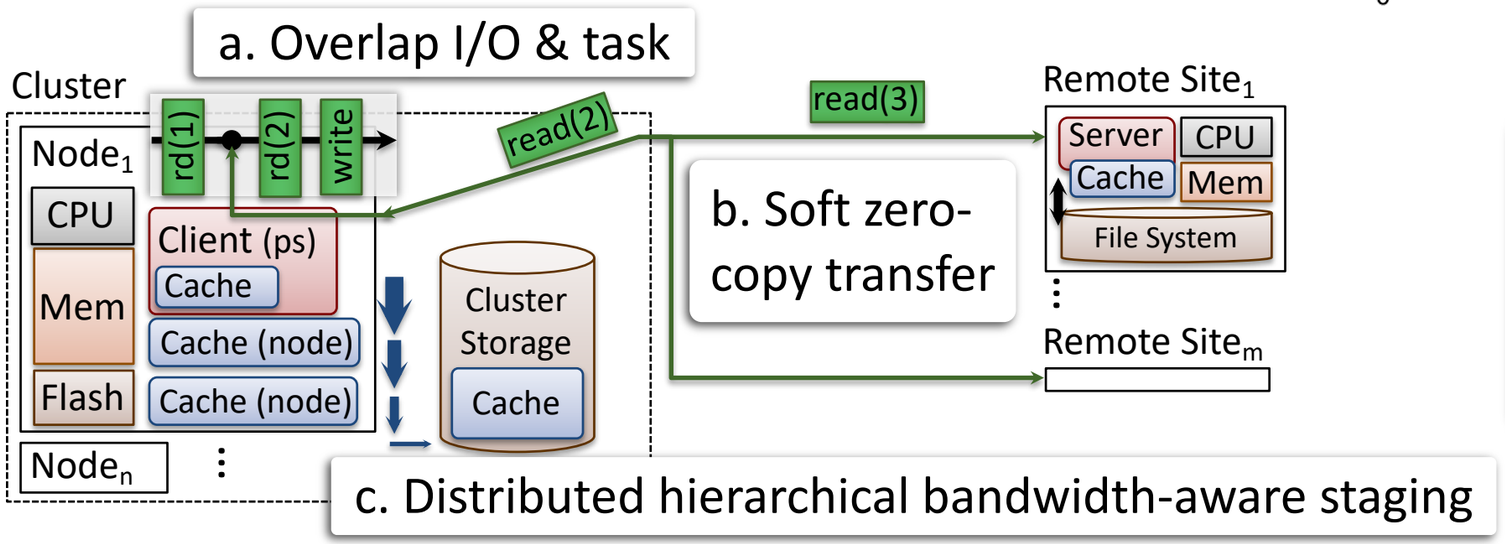
How to reuse/move data among tasks?

Belle II Monte Carlo (HEP)
- majority of Belle II compute
- 48 Gb/s over 2x 1 Gb/s WAN
- complex access pattern

TAZeR: intelligent data runtime

- Capture dynamic reuse
- ↓ access times
- ↓ data movement
- ↓ random access cost





- TAZeR: 22×/12× faster than FTP/XRootD
  - within 7% of optimal
- Workload with high reuse (4K tasks, 5 BG sets, 100 events)

BigFlowSim: Predict performance impact of I/O parameters

Friese, Mutlu, Tallent, Suetterlein, Strube. “Effectively using remote I/O for work composition in distributed workflows” BigData ’20

PNNL

- Workload Generator/Simulator for distributed workflows and remote I/O
- Study effects of data optimizations
  - (1) Reduce total data transferred over network
  - (2) Hide cost of network transfers
  - (3) Reduce inter-task footprints
  - (4) Reduce intra-task footprints

Parameters

Runtime
- Baseline: copy in/out
- Optimal: pre-staged

Data Intensity

Change I/O per work
- Data reducers
- Work aggregation

Data (re)Uses

Tasks reuse data for write-once read-many

Models & measurements order parameter impact by effects of footprints & movement

1 I/O intensity ≈ I/O Aware Locality (task ensembles) 1 3 ≥ Data (re)use 1 4

Code ≈ Access Pattern (density) 1 2

Scheduling & TAZeR

2 3 Access Pattern (ordered) ≈ Access Pattern (random)

I/O Aware Locality (no chunking) ≈ I/O Aware Locality (chunking) 3

With TAZeR

Capture reuse
↓ access time & moves
↓ random access cost

I/O Locality

Task ensembles, I/O chunking for temporal locality
- I/O-aware access/schedules
- In-situ computation

Access Pattern
- Data layout for I/O spatial locality
- Regular accesses for prefetching

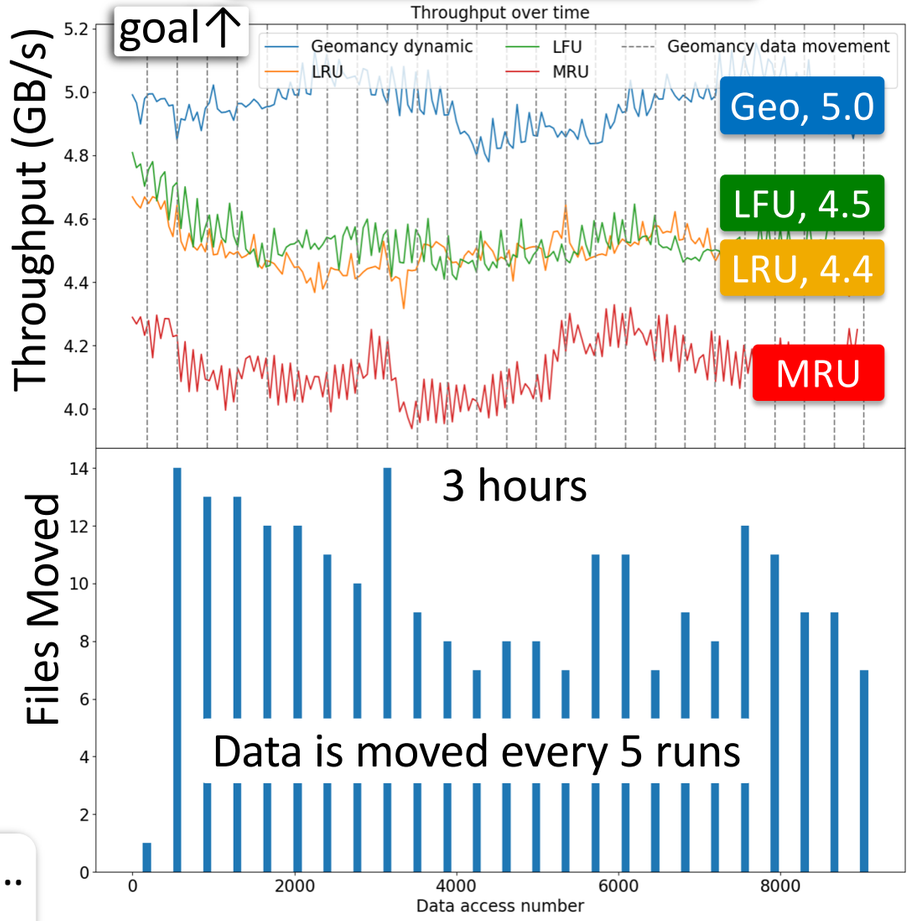
Geomancy: Adjust Data Layout To Improve Throughput

O. Bel, et al. “Geomancy: Automated Performance Enhancement through Data Layout Optimization” MSST ’20

- Data layout (storage) significantly affects performance
  - layout = map of files to storage system/devices
- Adjusting data layout can
  - tailor mapping for varying workloads
  - avoid congestion from other workloads
  - avoid failures (storage) & misconfigurations (nw link)
- Geomancy uses online learning:
  - monitor I/O activity and data layout
  - model performance (throughput)
  - predict performance during next window
  - select and realize new layout

Geomancy: 11% over LFU

Geomancy *v.* Heuristics



On Belle II MC, Geomancy learns:
- indicators of read contention
- RAID-5 has different read/write speeds

LFU: Least frequently used…
LRU: Least recently used…
MRU: Most recently used…
file to slowest

Adjusting layouts dynamically: more consistent and higher performance

Footprint Access Diagnostics Improve Distributed Graph Clustering

Kilic, Tallent, Friese. "Rapid memory footprint access diagnostics" ISPASS '20

PNNL

Analyze dynamic data locality.
Runtime overhead of 5-10%.

miniVite
(Intel SkyLake)

3 Hash Tables

0: unorder map
open hash tbl
(array + list)

1: hopscotch,
default size
closed hash tbl
(array)

2: hopscotch,
dynamic size
(vertex degree)

Call path →

Metrics:	time(usec)	fp-$\overline{pf}$	footprint	fp-$\overline{pf}$%	fpΔ	fpΔ-$\overline{pf}$
171: distLouvainMethod	4.83e+07	2.50e+09	1.59e+10		4.01e-01	6.30e-02
loop at dspl.hpp-v1: 1353			0		3.98e-01	6.29e-02
loop at dspl.hpp-v1: 1353			0		3.98e-01	6.29e-02
1353: distExecuteLouvainIteration			0	1.56e+01	3.97e-01	6.18e-02
331: distBuildLocalMapCounter			0	9.07e+00	3.99e-01	3.62e-02
loop at dspl.hpp-v1: 254	3.30e+07	9.69e+08	1.11e+10	8.78e+00	3.99e-01	3.50e-02
map.insert() 281: [I] std::pair<std::__deta	2.59e+07	1.20e+09	6.93e+09	1.74e+01	3.87e-01	6.73e-02
map.find() 276: [I] std::unordered_map	3.11e+06	4.28e+07	1.82e+09	2.39e+00	3.53e-01	8.42e-03
1353: distExecuteLouvainIteration	2.76e+07	3.23e			.47e-01	6.80e-02
331: distBuildLocalMapCounter	2.29e+07	2.35e			.79e-01	5.64e-02
loop at dspl.hpp-v1: 254	2.28e+07	2.28e+09	1.99e+10	1.14e+01	4.78e-01	5.47e-02
map.insert() 281: [I] tsl::hopscotch_map<	1.71e+07	6.84e+07	1.42e+10	4.82e-01	4.82e-01	2.32e-03
map.find() 276: [I] tsl::hopscotch_map<	2.02e+06	1.78e+08	1.95e+09	9.19e+00	4.07e-01	3.74e-02
331: distBuildLocalMapCounter	1.09e+07	1.86e+09	1.69e+10	1.10e+01	7.04e-01	7.77e-02
loop at dspl.hpp-v1: 254	1.09e+07	1.85e+09	1.69e+10	1.09e+01	7.05e-01	7.72e-02
map.insert() 281: [I] tsl::hopscotch_map	6.36e+06	3.79e+08	1.18e+10	3.23e+00	8.80e-01	2.84e-02
map.find() 276: [I] tsl::hopscotch_map	1.13e+06	4.66e+06	1.24e+09	3.76e-01	3.54e-01	1.33e-03

- Hot call path by time
- Solution: new hash table
- V0→V2: high fp, low fp-$\overline{pf}$%
- V1: extra copies!

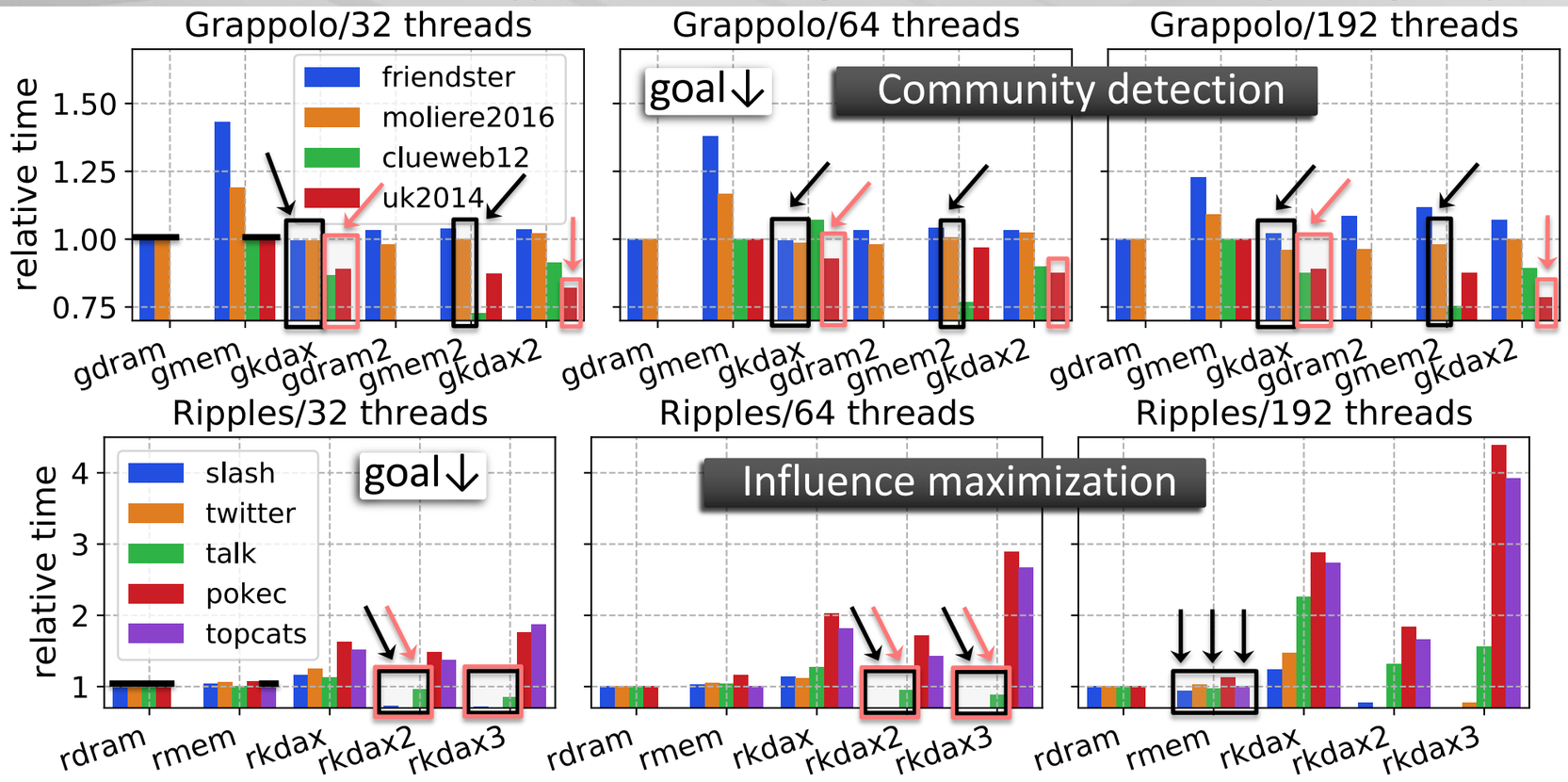
% footprint, irregular accesses

Reduce time by 67%
(26 s → 17 s → 6.4 s)

Single-Node Partitioned-Memory Graph Analytics (DRAM + Optane)

Ghosh, Tallent, Minutoli, Halappanavar, et al. “Single-Node Partitioned-Memory for Huge Graph Analytics: Cost & Performance...” SC ’21

PNNL



Apps: Different working set, partitioning, access patterns, synchronization.
Vary parallelism, variants, Optane modes.

AppDirect/Memory can equal DRAM...

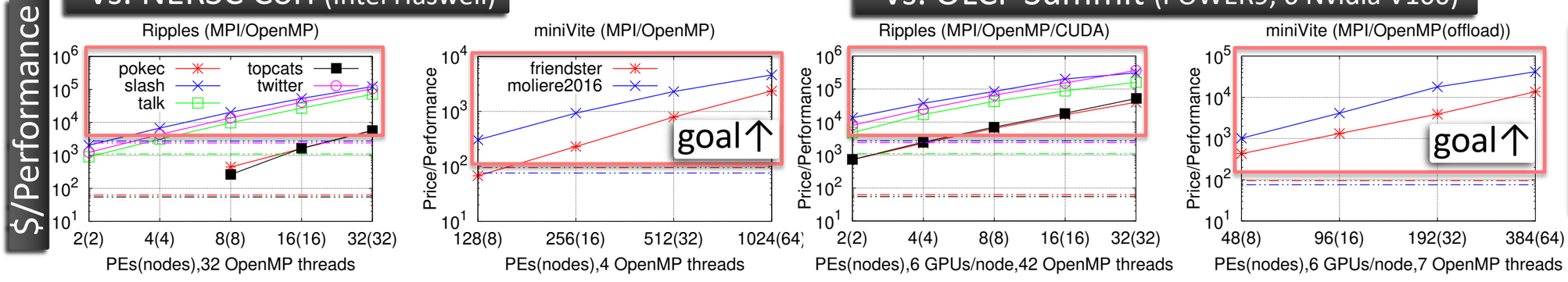
AppDirect can beat Memory...

...even with irregular access patterns, *if* most accesses are NUMA-local and Optane accesses are frequently reads

Memory v. AppDirect: L3 cache invalidations can change!
Fix with non-temporal hints.

Vs. NERSC Cori (Intel Haswell)

Vs. OLCF Summit (POWER9, 6 Nvidia V100)



Big-node often >4-10× vs. supercomputers
(Key: memory & comm for ghost vertices)