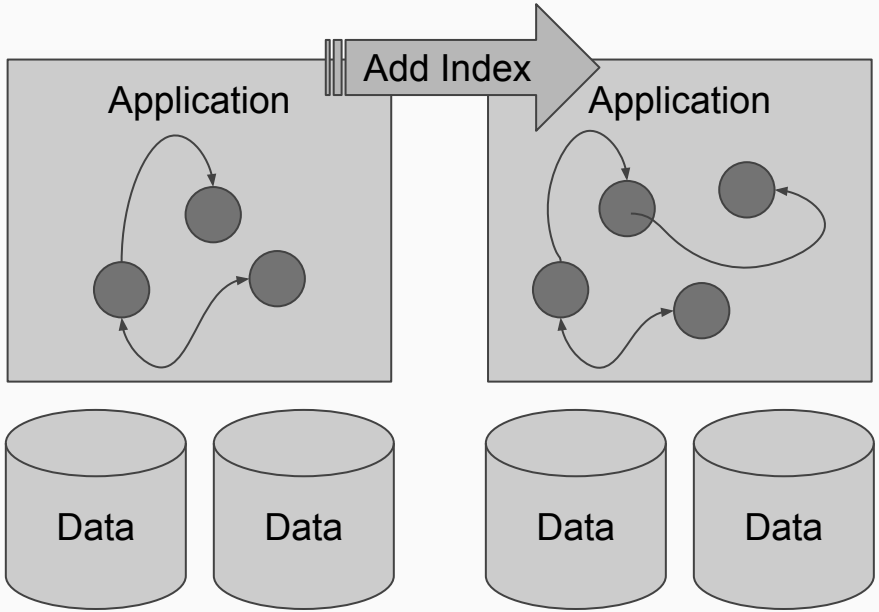
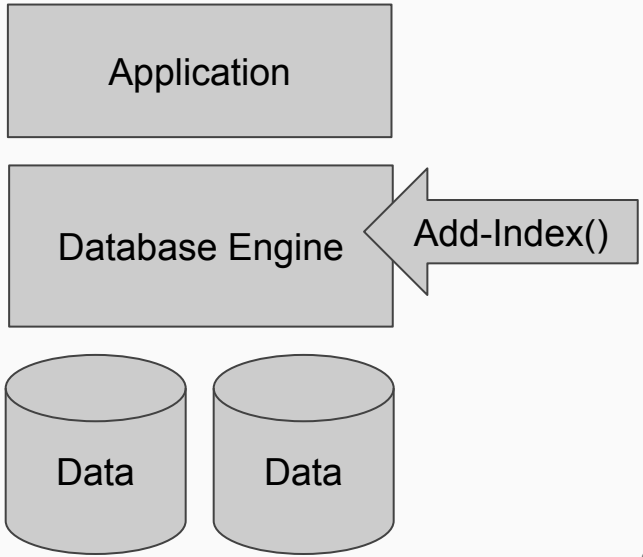
# Automatic and Transparent I/O Optimization With Storage Integrated Runtime Support


| Noah Watkins<br>Carlos Maltzahn | Zhihao Jia<br>Alex Aiken | Galen Shipman<br>Pat McCormick |
| --- | --- | --- |
| *UC Santa Cruz* | *Stanford* | *LANL* |

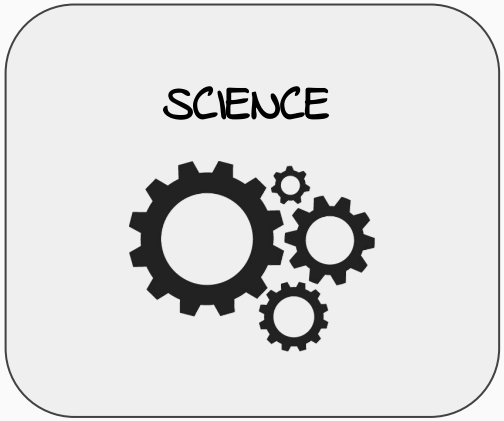
# What is this talk about?

- Convince you that storage should be interested into [HPC] application execution models

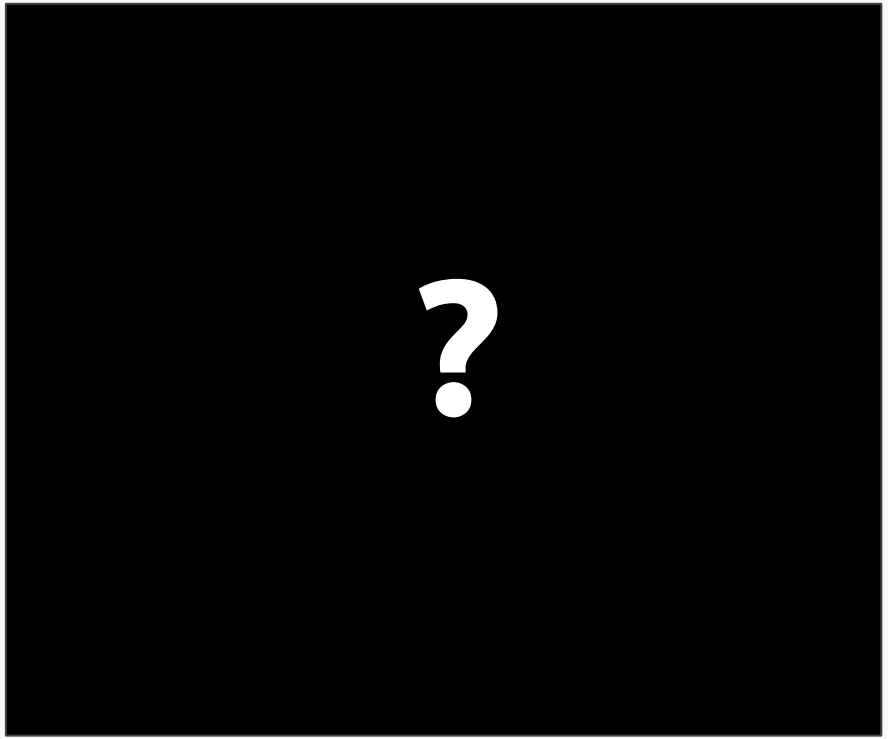


# Application and System Development

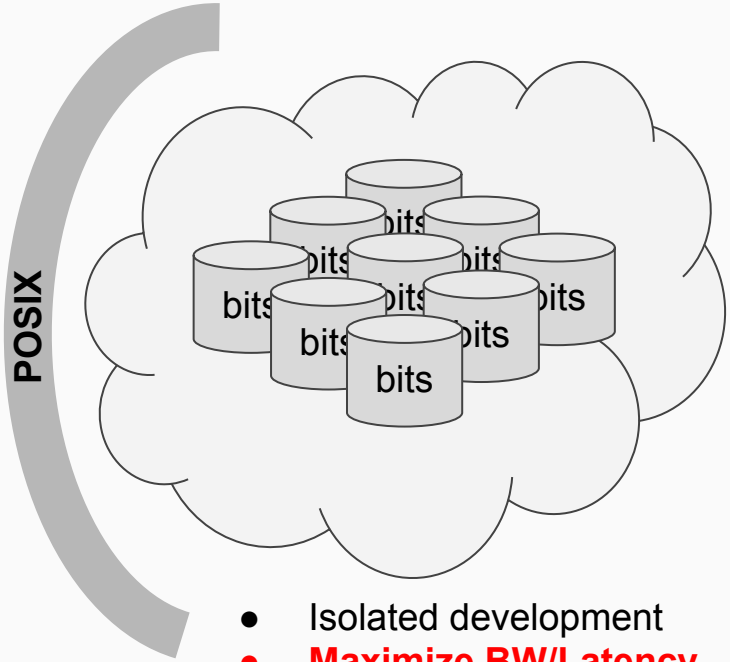



- Isolated development
- **Maximize FLOPS**
- Checkpoint / Restart



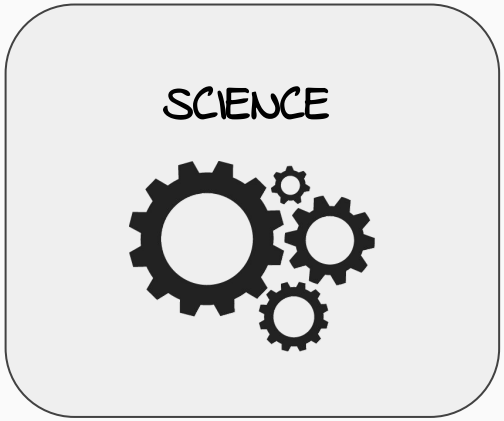
# Application and System Development

?

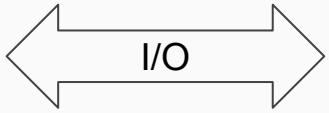


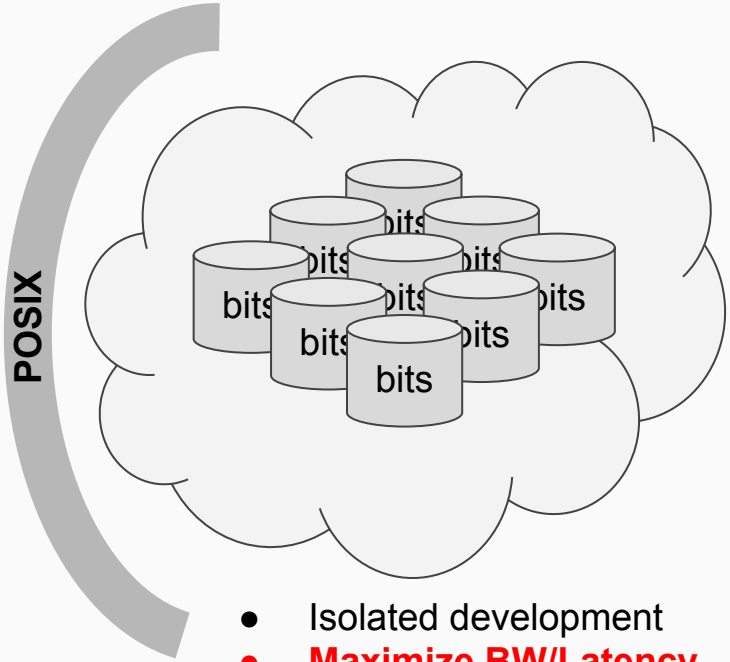

- Isolated development
- **Maximize BW/Latency**
- File system interface

# Conflict of Interest in System Development



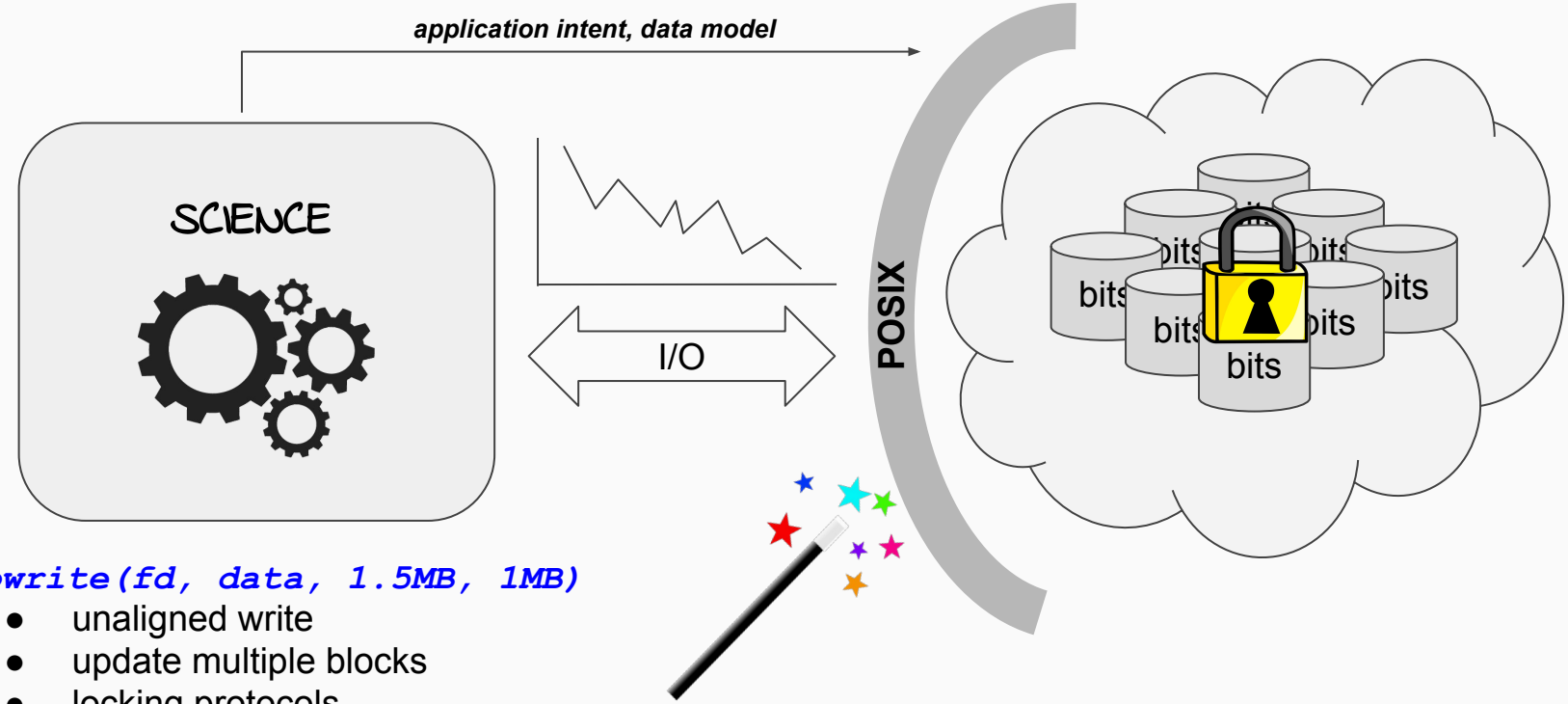

- Isolated development
- **Maximize FLOPS**
- Checkpoint / Restart




- Isolated development
- **Maximize BW/Latency**
- File system interface

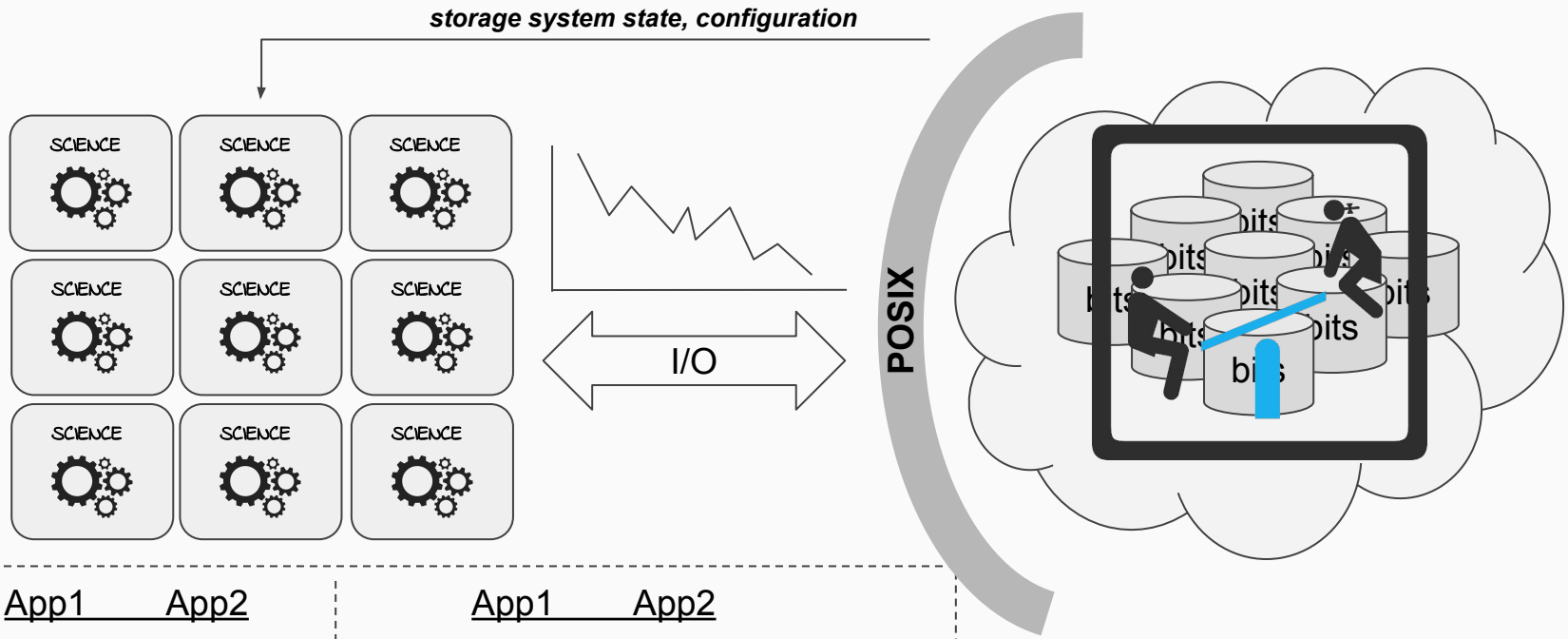
# Abstractions hide important parameters




*pwrite(fd, data, 1.5MB, 1MB)*

- unaligned write
- update multiple blocks
- locking protocols

# Inflexible applications cannot adapt

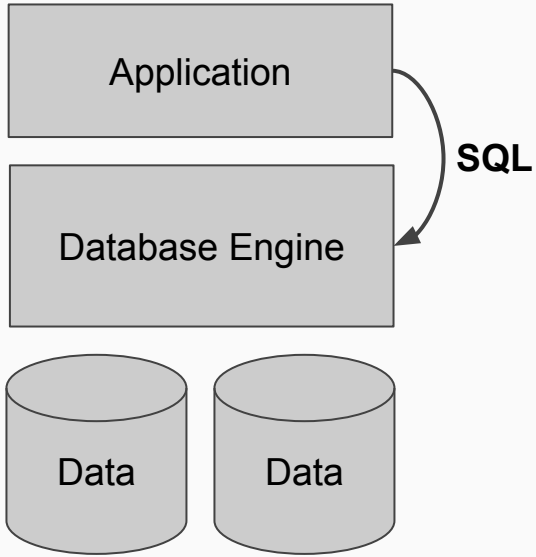


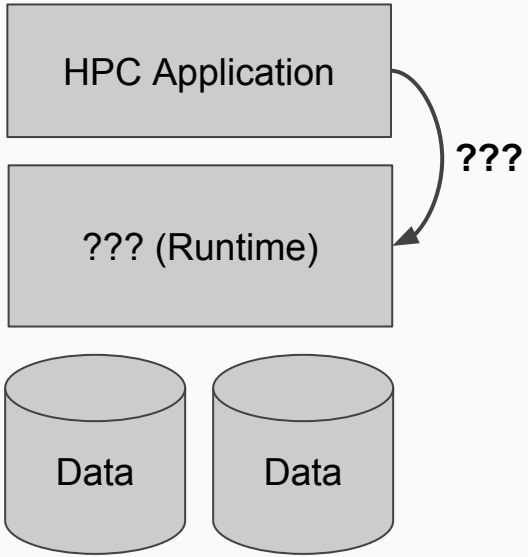

| App1 | App2 |
|---|---|
| *t0* write(...) | write(...) |
| ***contention*** | |

| App1 | App2 | |
|---|---|---|
| *t0* write(...) | | |
| *t1* | write(...) | ***blocking*** |

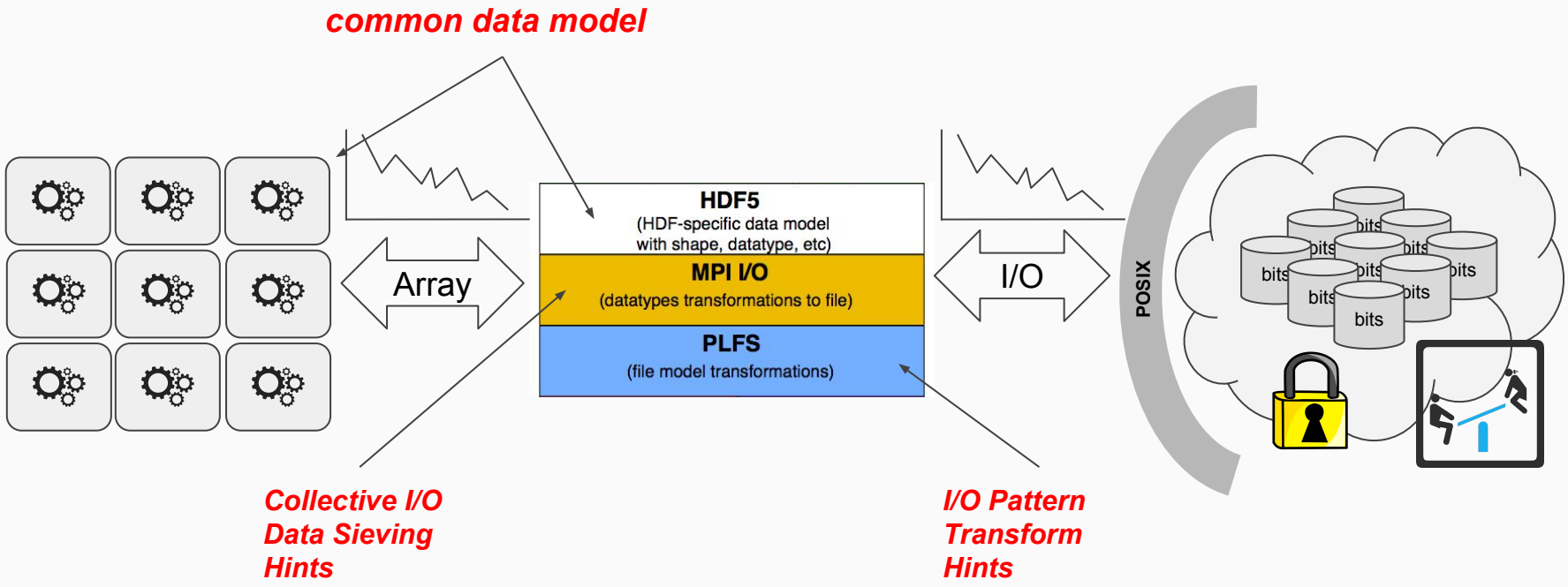
# Communicating application requirements

- Database engines use SQL to communicate declarative requirements
- HPC applications are entirely different, and require different mechanisms



# I/O Middleware Stacks

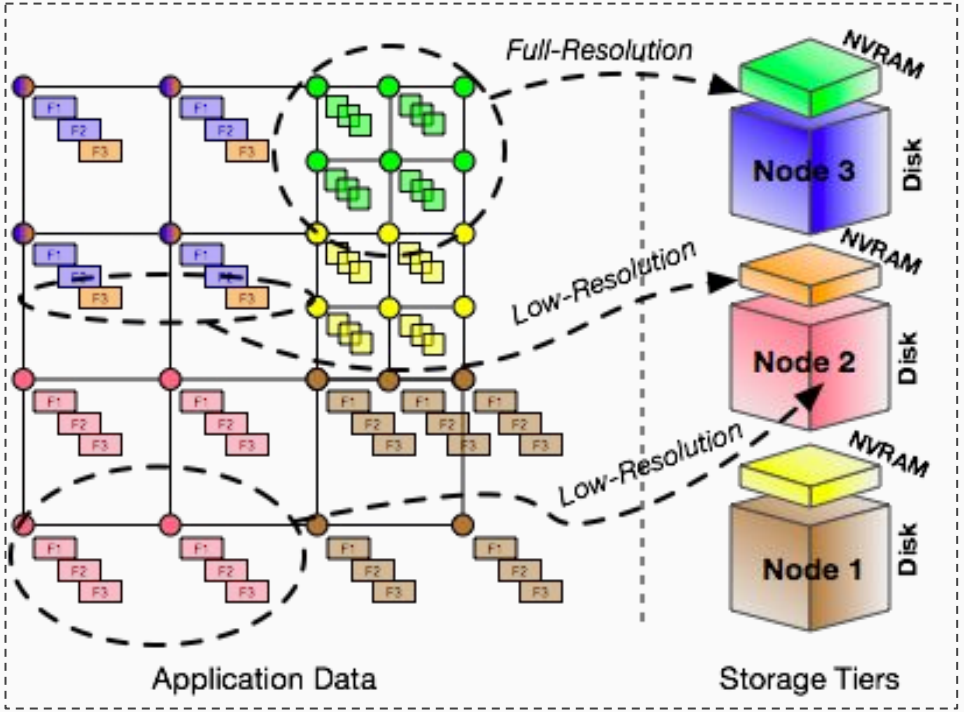


# Remainder of the talk

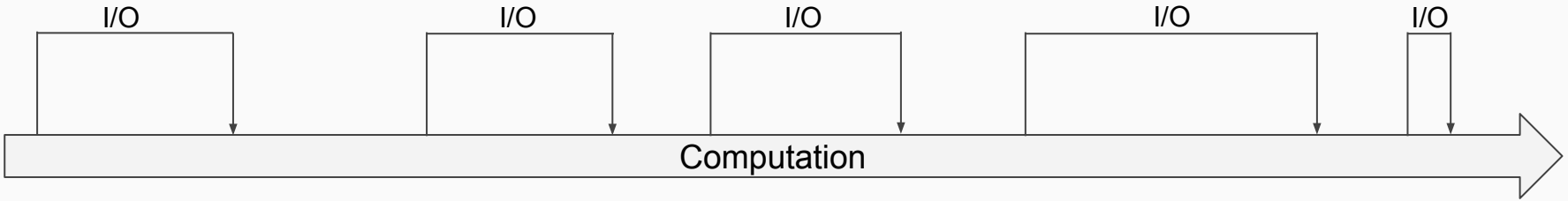
- Illustrate challenges for existing I/O stacks and application design
- Describe our work integrating storage into the Legion runtime
- Preliminary results

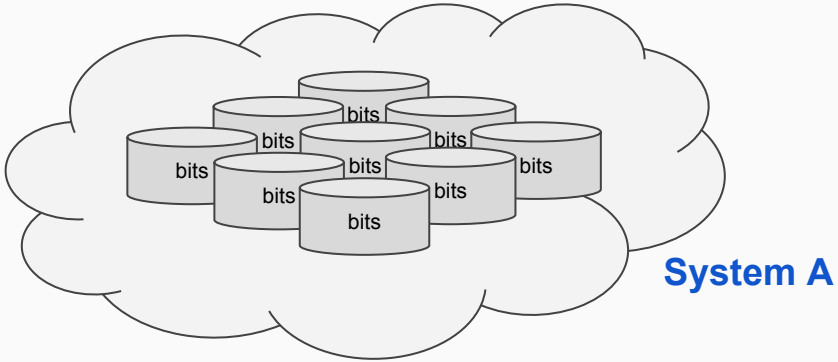
# Motivating Example

- Heterogeneous memory hierarchy
  - Multiple tiers and networks
- Adaptive mesh refinement (AMR)
  - Resolution-aware I/O
- Workflow systems and in-transit
  - Data rendezvous
- Out-of-core algorithms
- **Data management challenges**
  - **Metadata**
  - **Consistency**
- **Independent I/O**

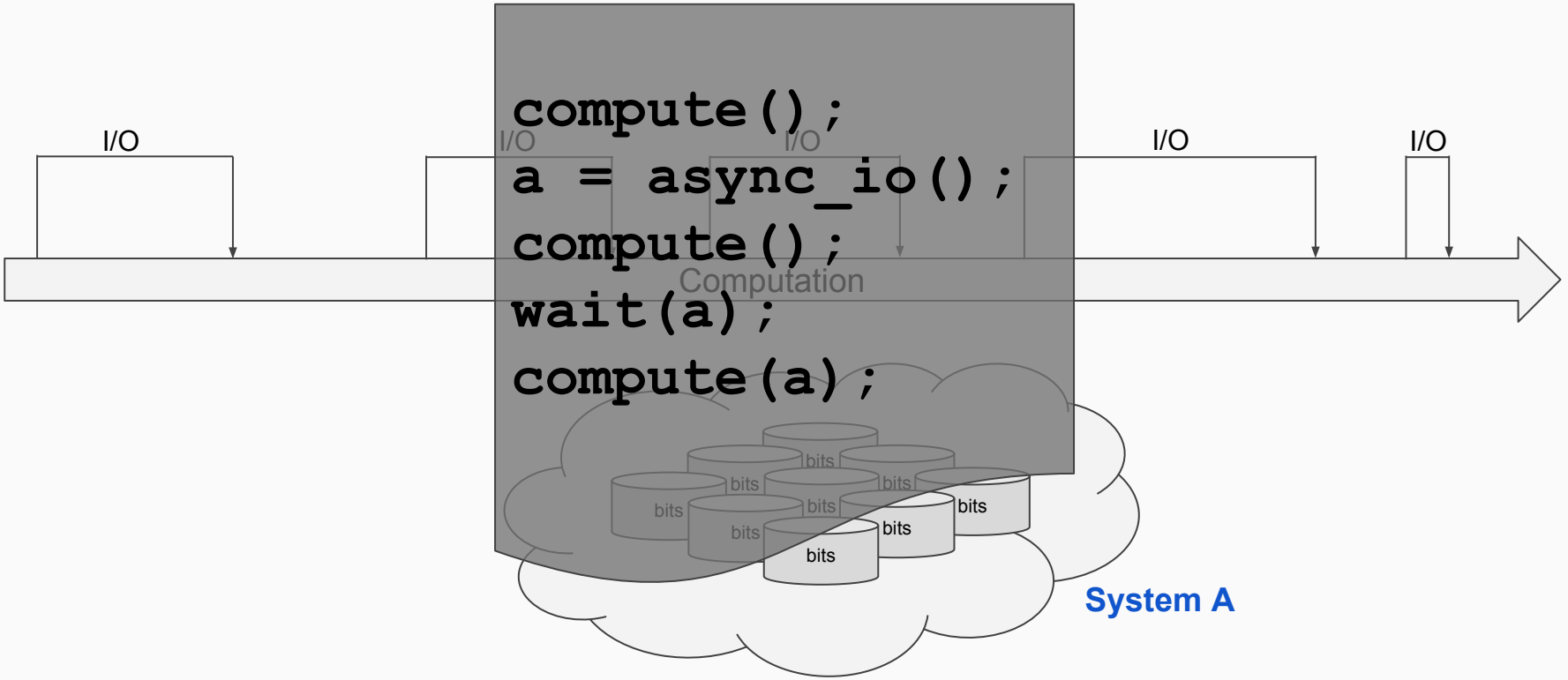


# Motivating Example: Independent I/O

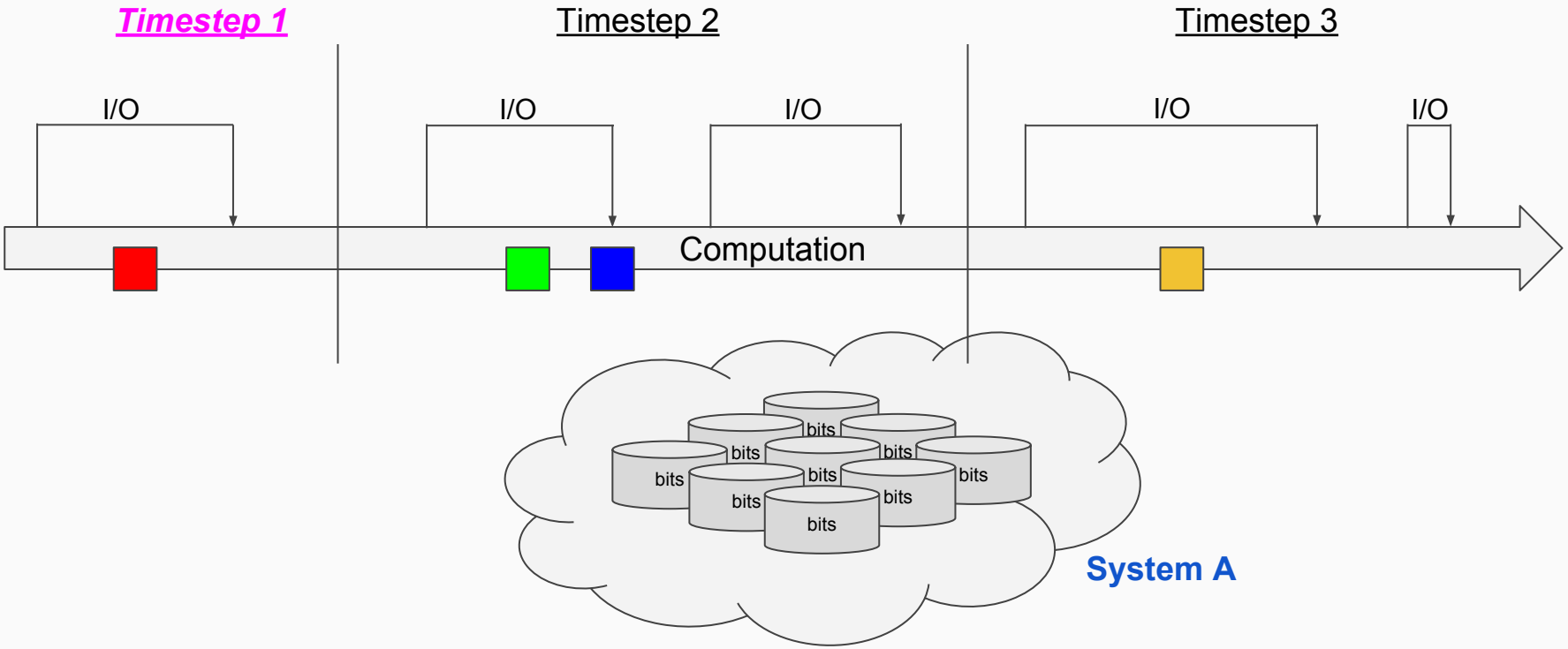


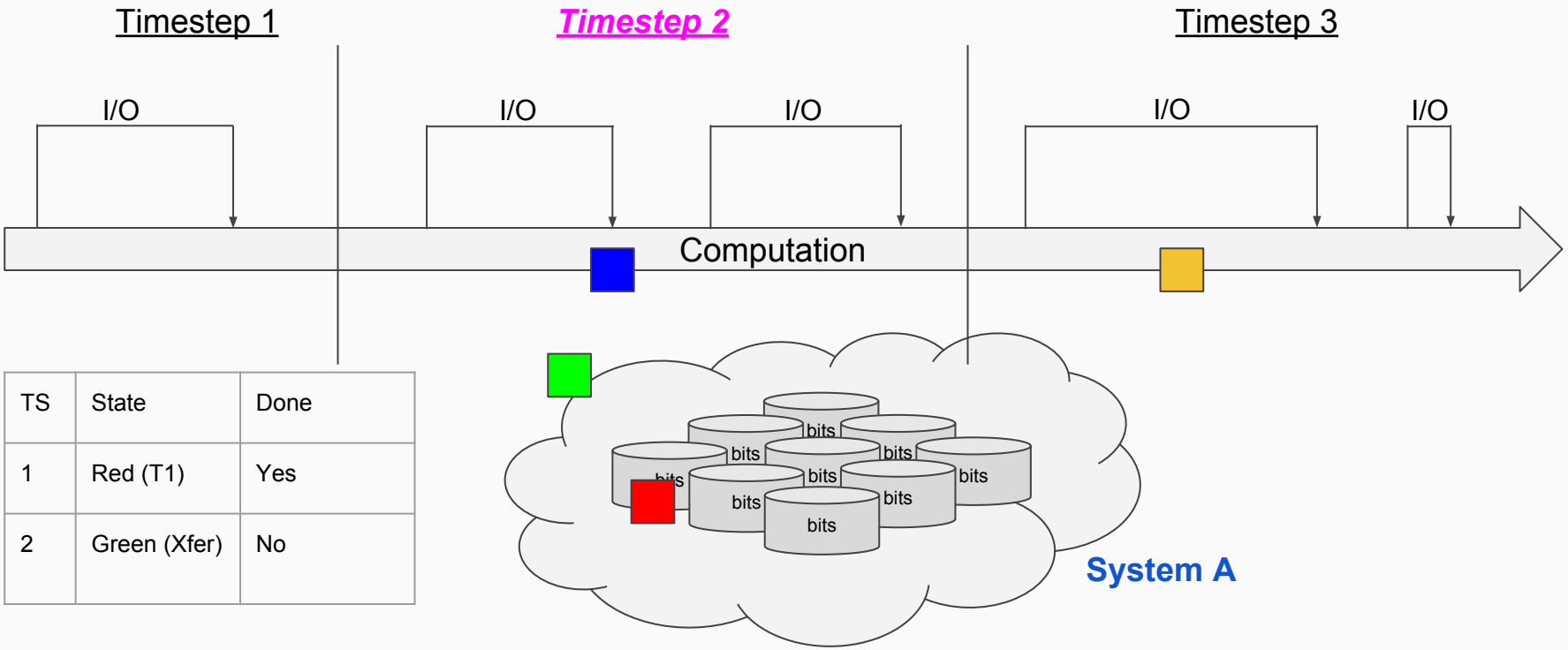
# Motivating Example: Independent I/O



# Independent I/O: Consistency Challenges

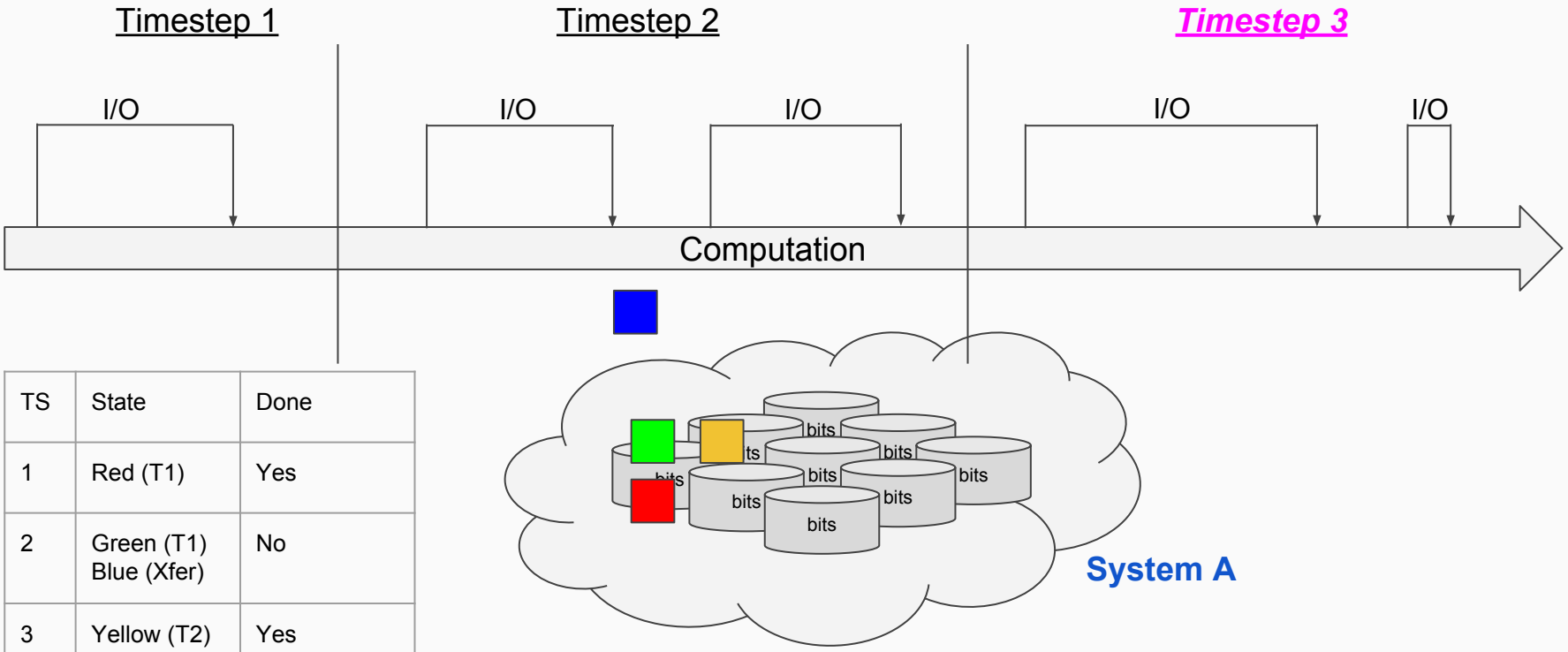


# Independent I/O: Consistency Challenges



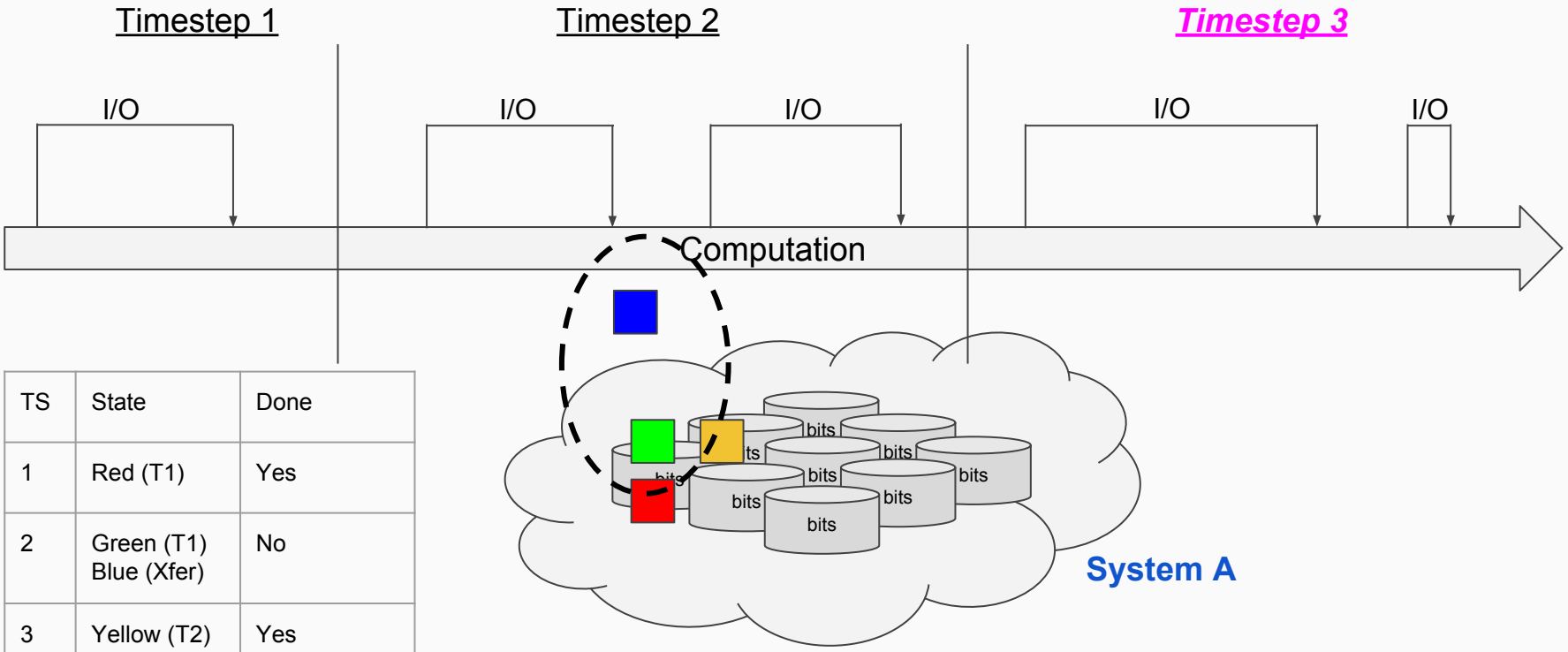

| TS | State | Done |
|---|---|---|
| 1 | Red (T1) | Yes |
| 2 | Green (Xfer) | No |

# Independent I/O: Consistency Challenges



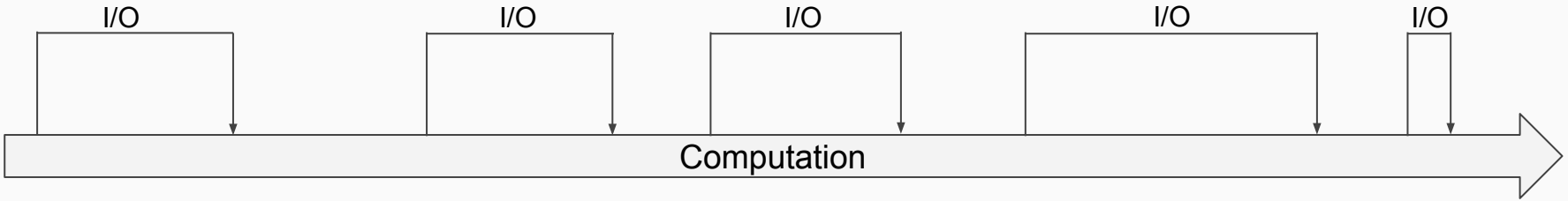

| TS | State | Done |
|---|---|---|
| 1 | Red (T1) | Yes |
| 2 | Green (T1)<br>Blue (Xfer) | No |
| 3 | Yellow (T2) | Yes |

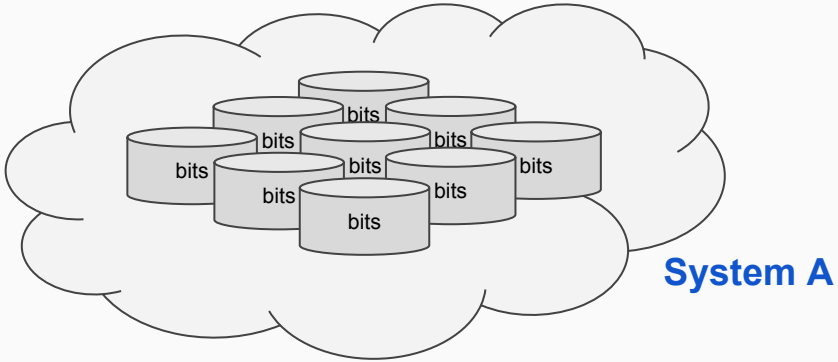
# Independent I/O: Consistency Challenges



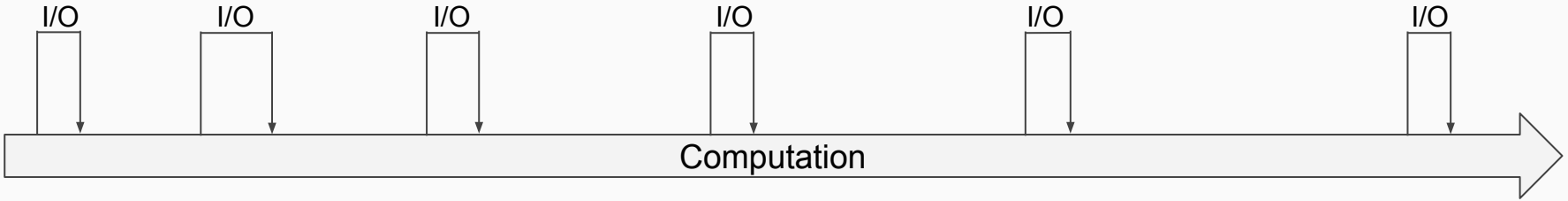

| TS | State | Done |
|---|---|---|
| 1 | Red (T1) | Yes |
| 2 | Green (T1)<br>Blue (Xfer) | No |
| 3 | Yellow (T2) | Yes |

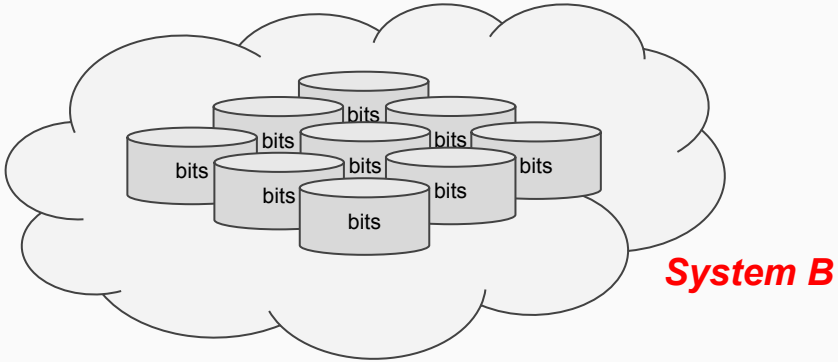
# Independent I/O: Portability



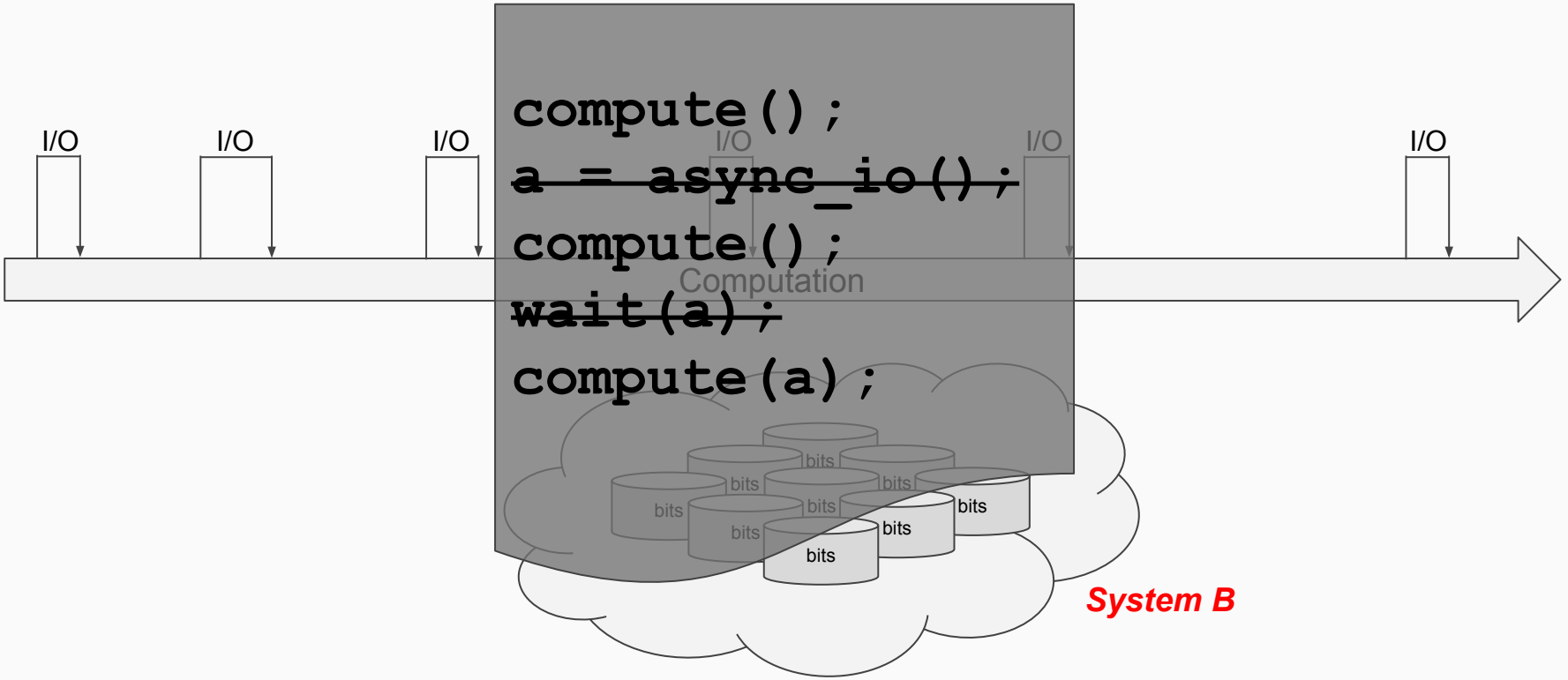



# Independent I/O: Portability

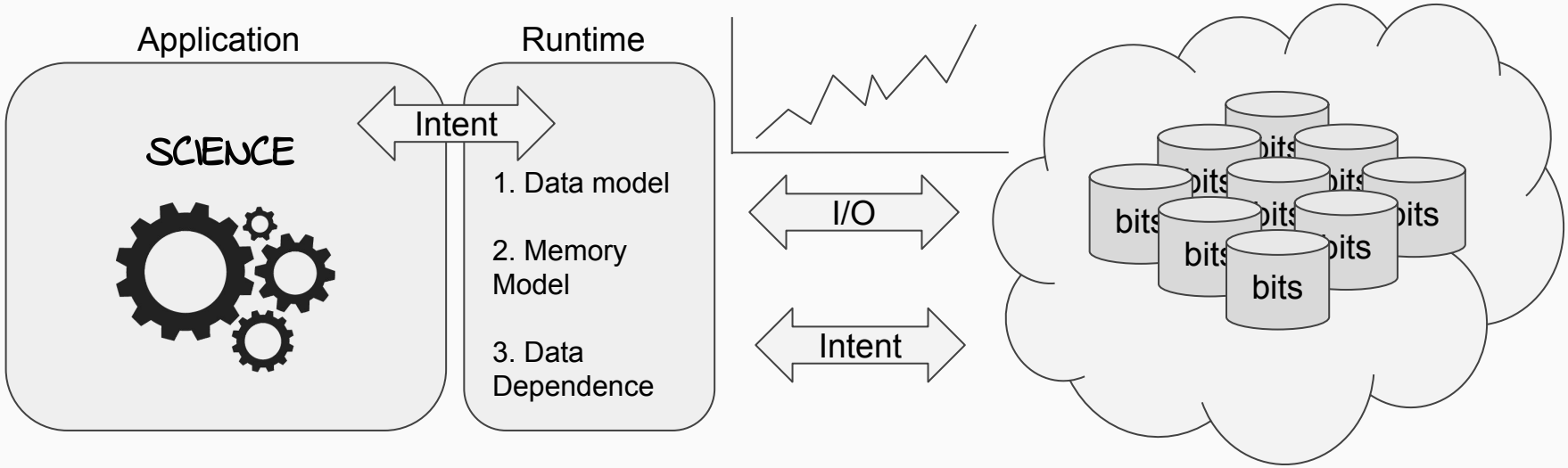


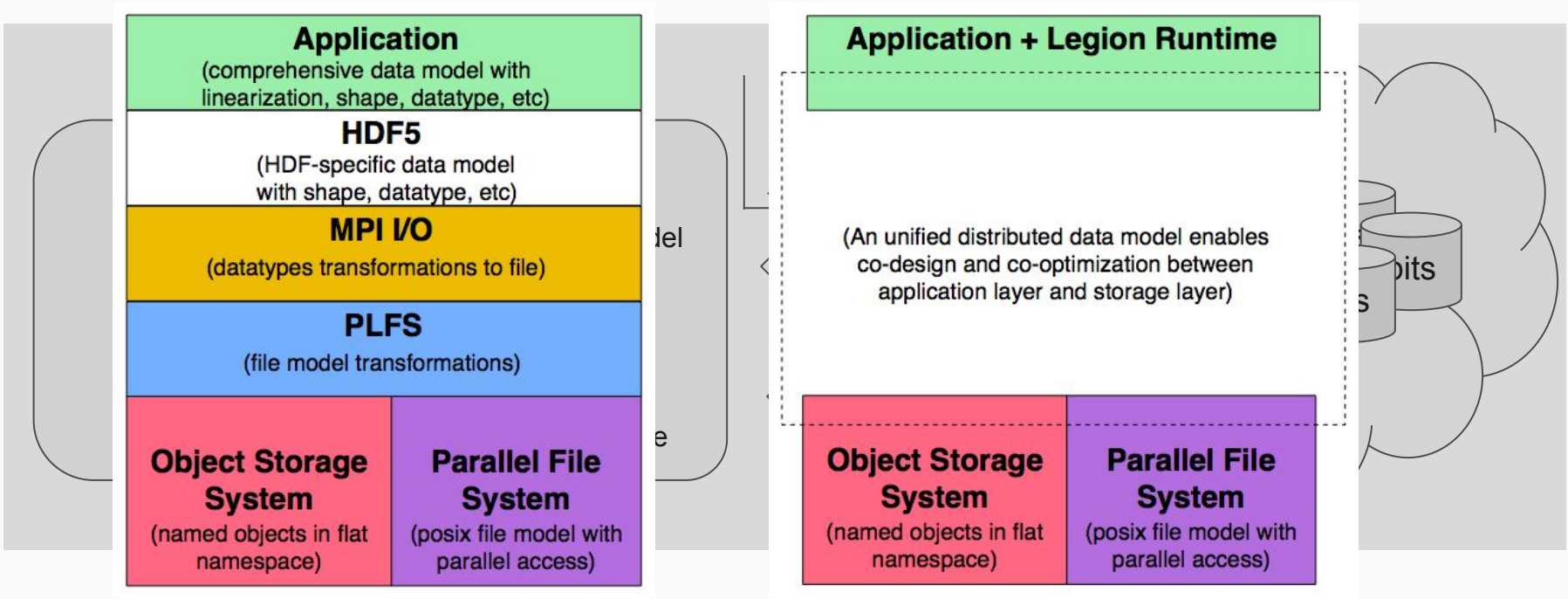
# Independent I/O: Portability



# Melding I/O and Application Semantics

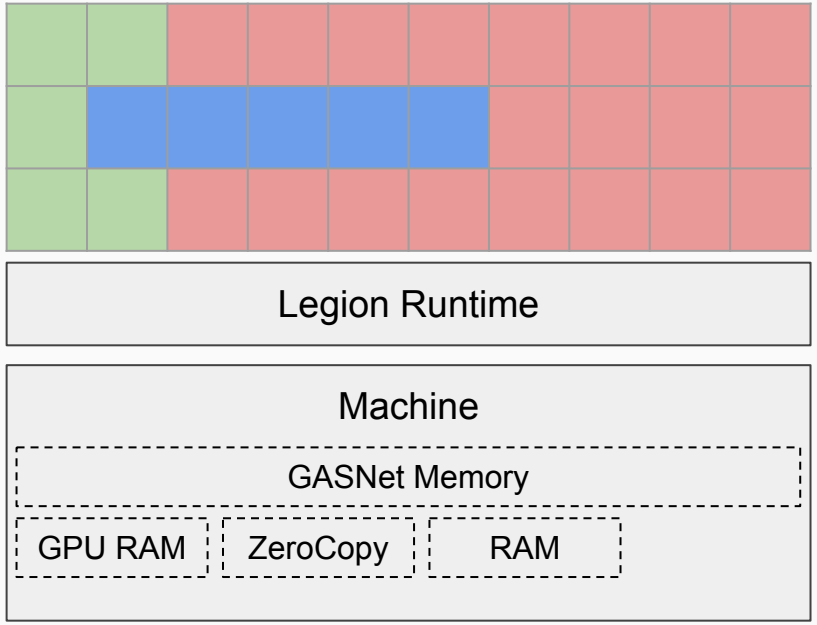


# Melding I/O and Application Semantics



# Legion Programming Model and Runtime

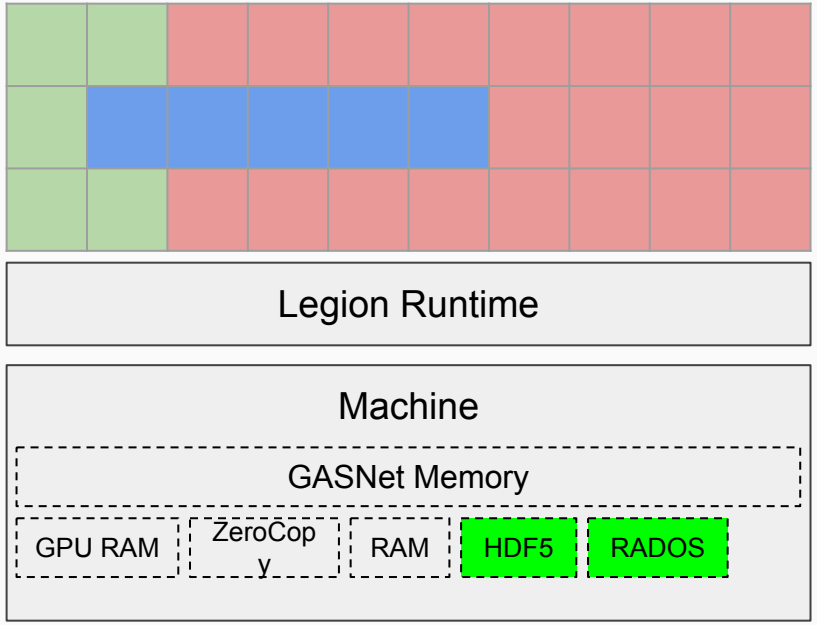
- Prototype built in Legion
  - Parallel, data-centric, task-based
- *Logical Region* Data Model
  - Do not commit to physical layout
- Memory hierarchy
  - Unified model across memory types
- Data dependencies extracted from application
  - Managed by runtime
  - Optimizations



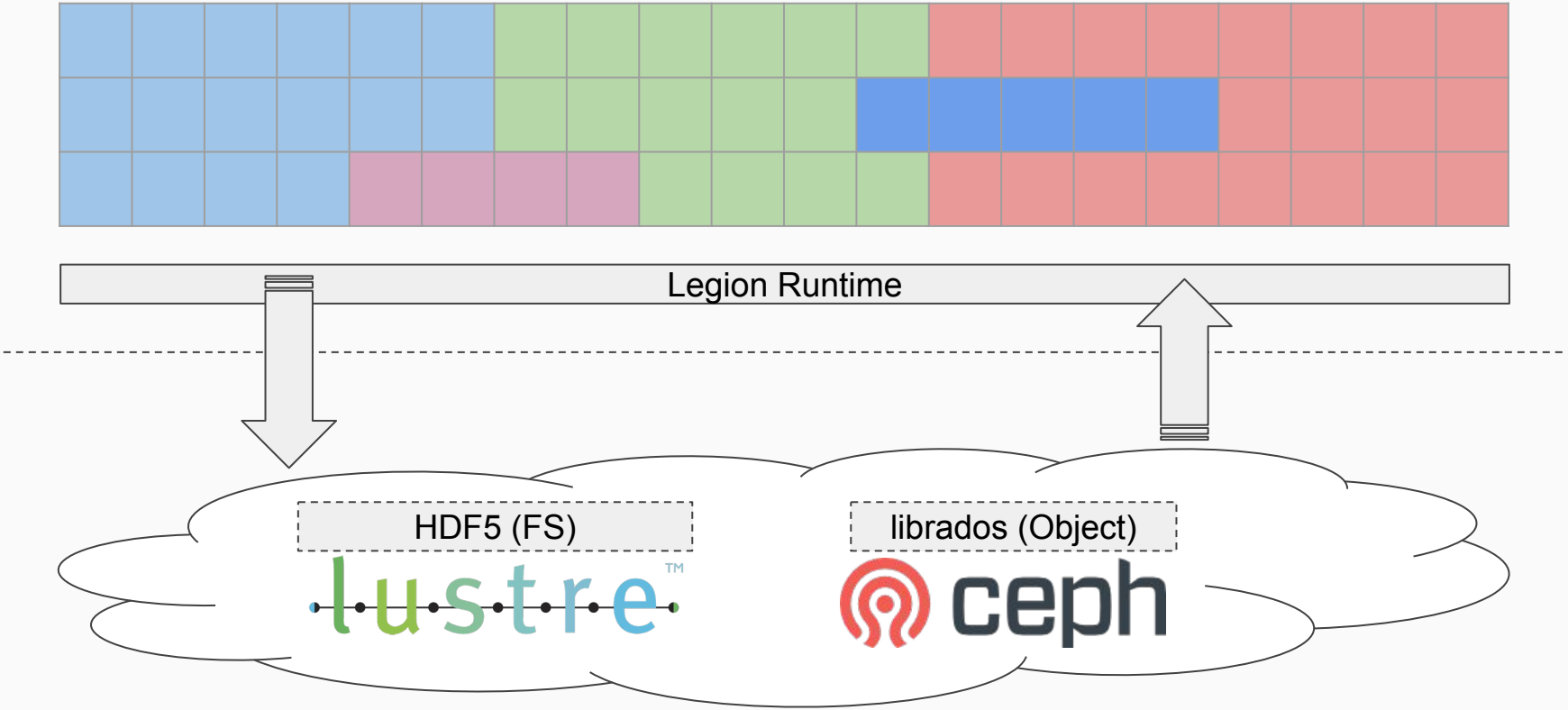

***"Legion: Expressing Locality and Independence with Logical Regions"***,
*Michael Bauer, Sean Treichler, Elliott Slaughter, Alex Aiken, SC 12*

# Legion and Persistent Memory Integration

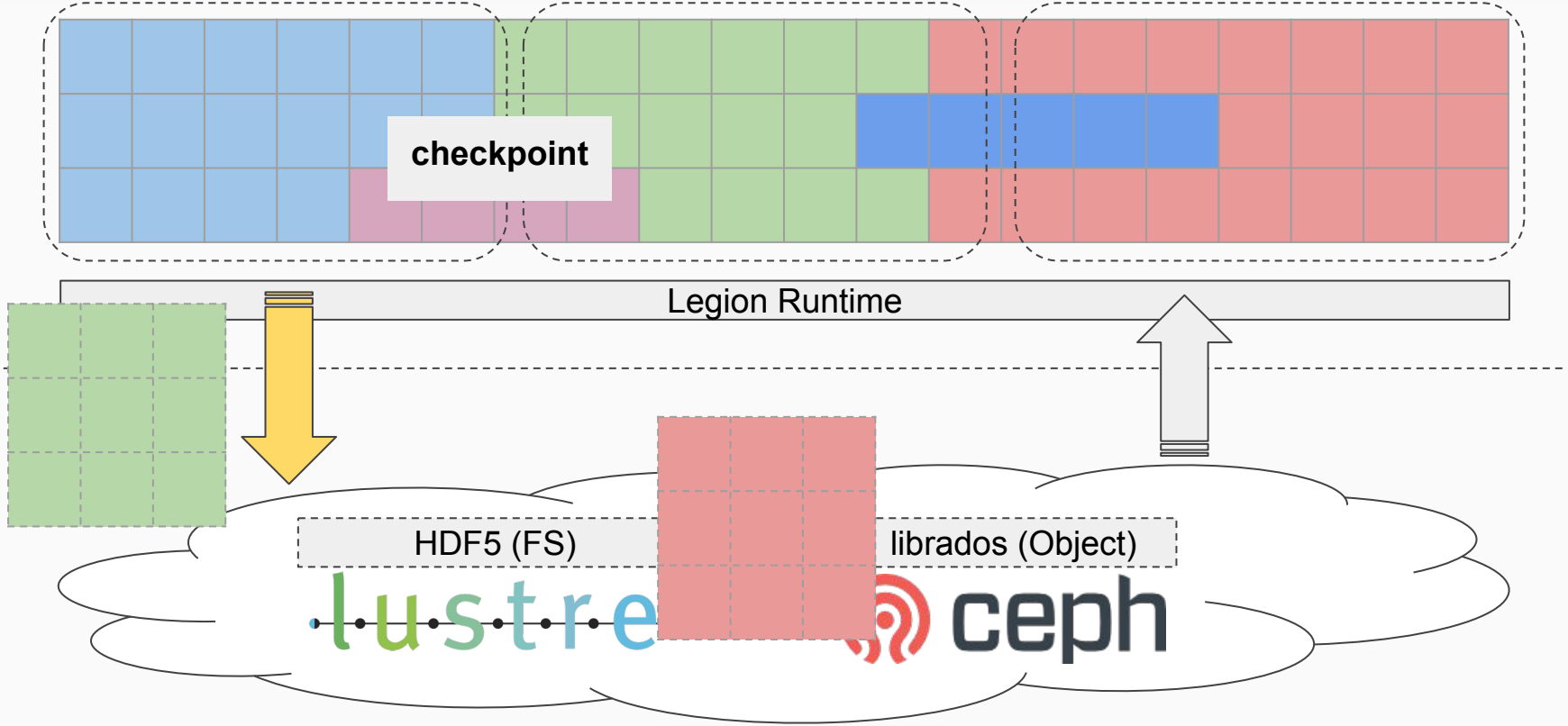
- Our work introduces persistent memory into Legion
- HDF5 and RADOS targets
- Legion tracks instances like any other memory
- Persistent is transparent to application
- Integrated with dependence tracking and coherence control



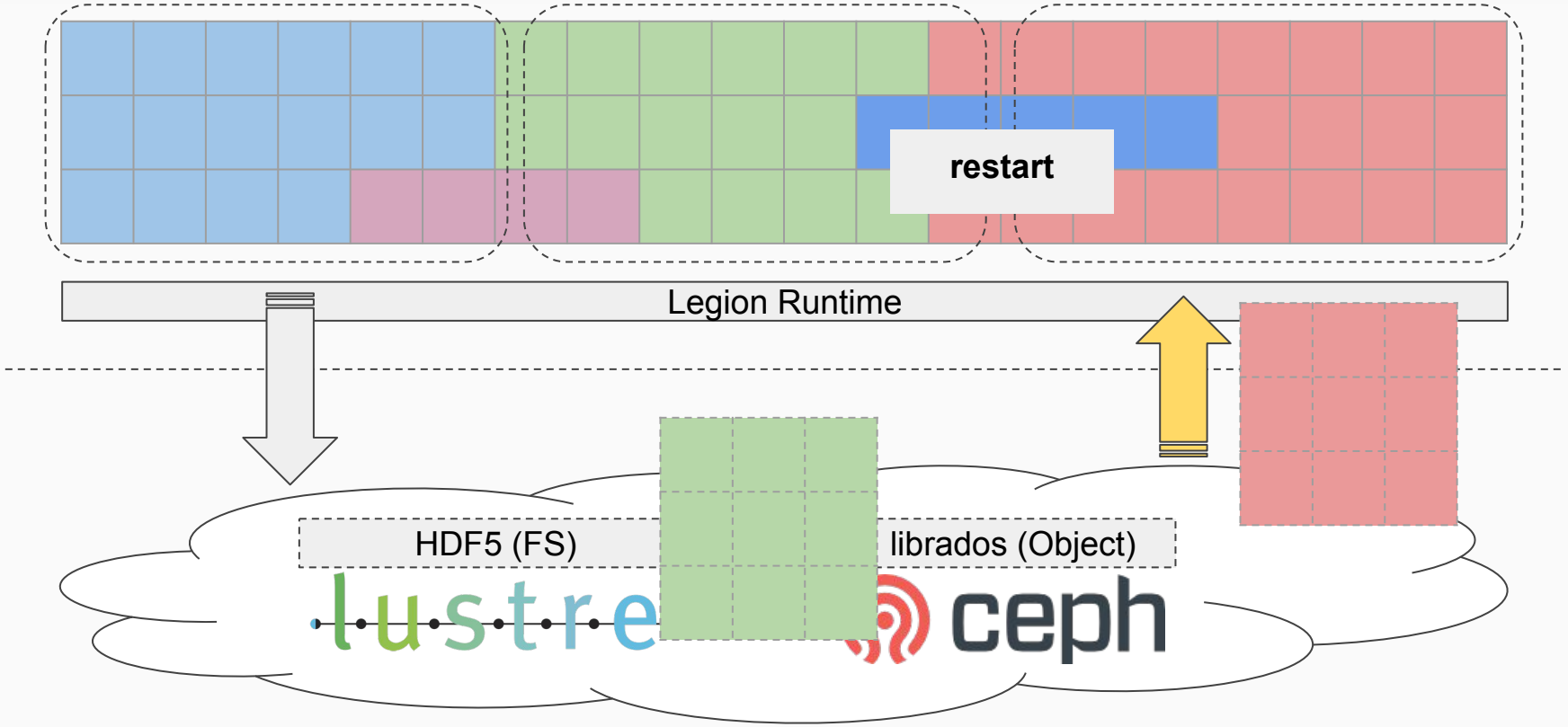
# Preliminary Results: Microbenchmark



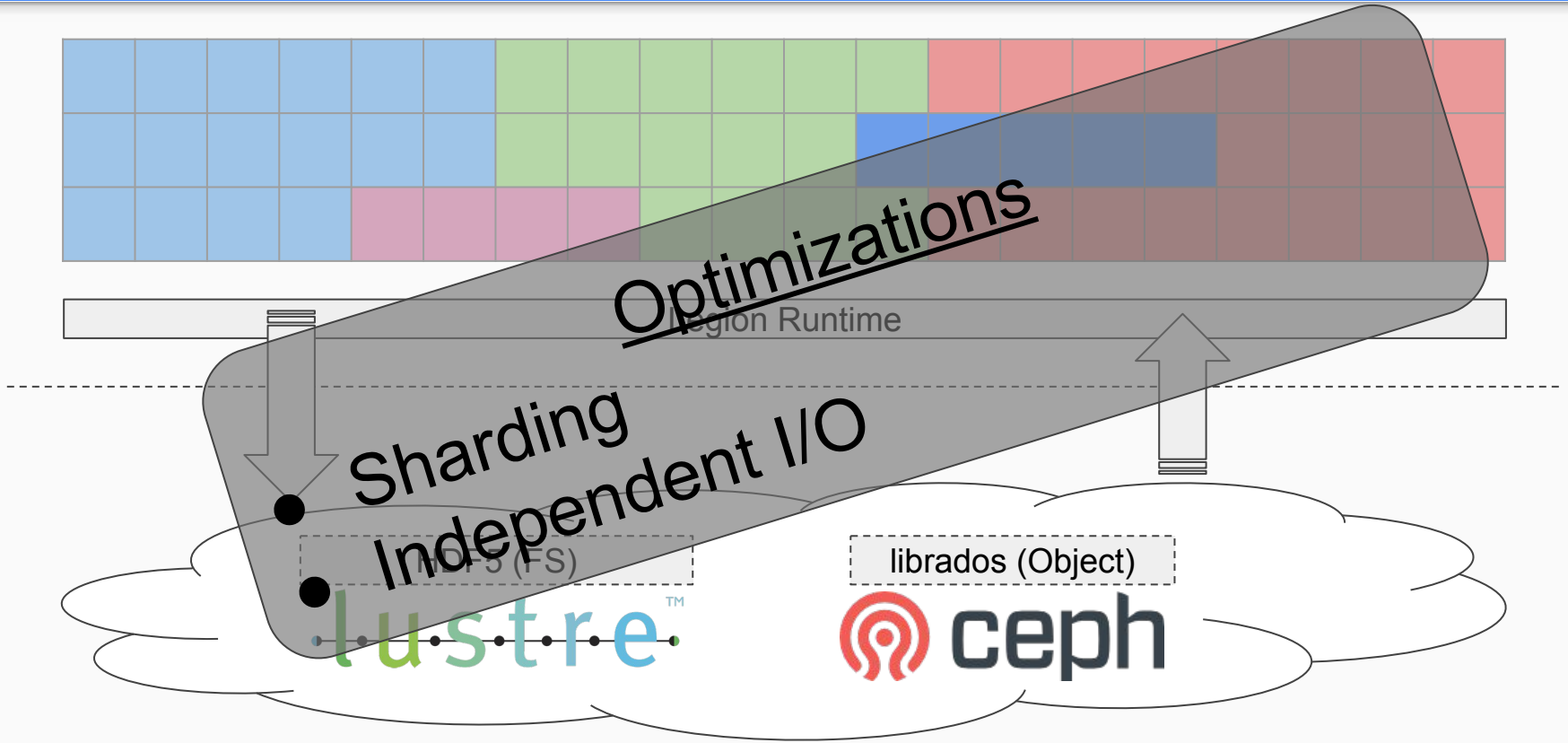
# Preliminary Results: Microbenchmark



# Preliminary Results: Microbenchmark

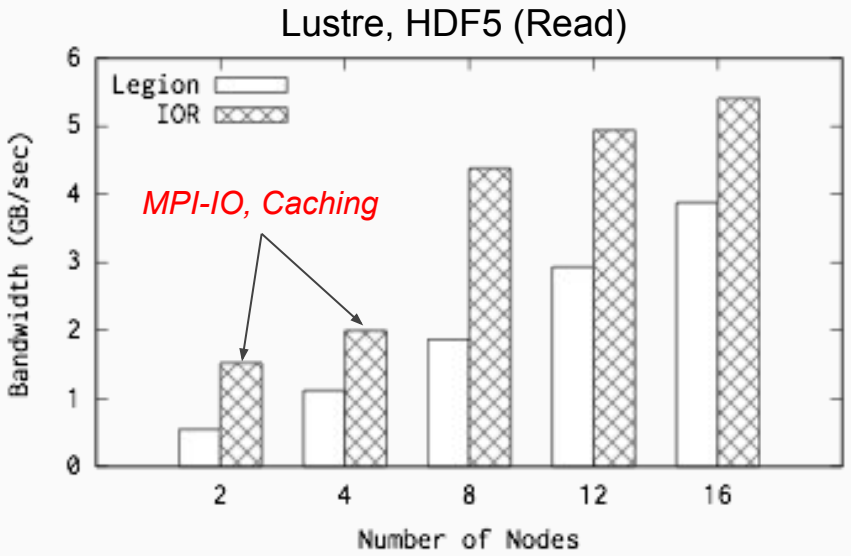


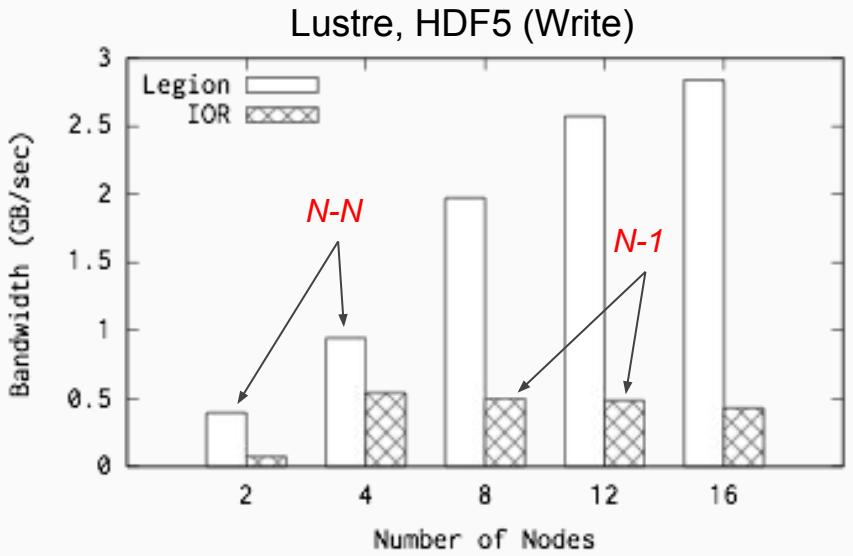
# Preliminary Results: Optimizations



# Preliminary Results: Weak Scaling

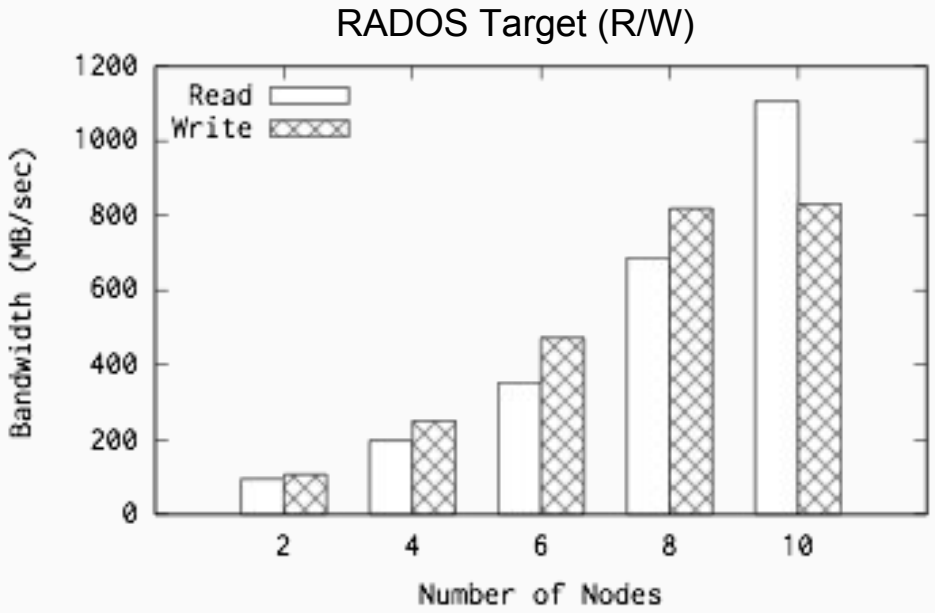
- Application state partitioned into 256 shards
- Scaled from 4 GB to 32 GB across 2 to 16 nodes
- Compared throughput against IOR, N-1, HDF5, MPI-IO



# Preliminary Results: Weak Scaling

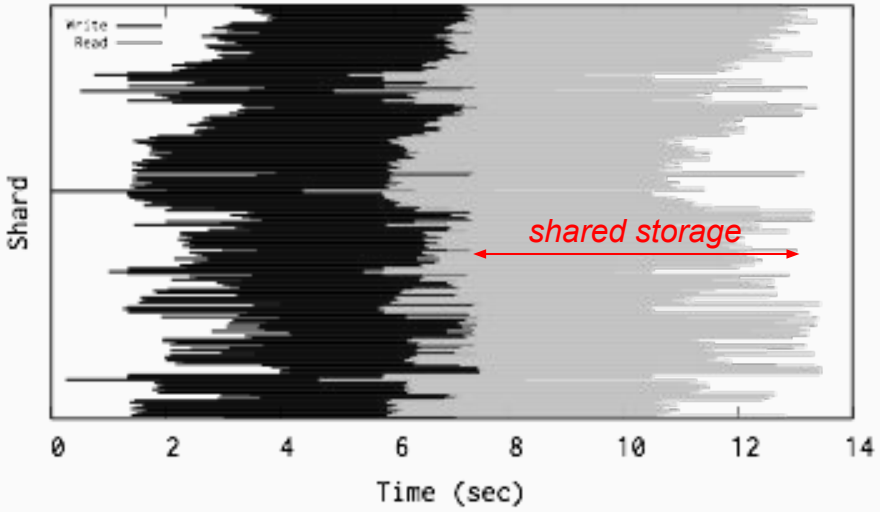
- Application state partitioned into 256 shards
- Scaled from 4 GB to 32 GB across 2 to 10 nodes
- Transparent integration with non-POSIX backends



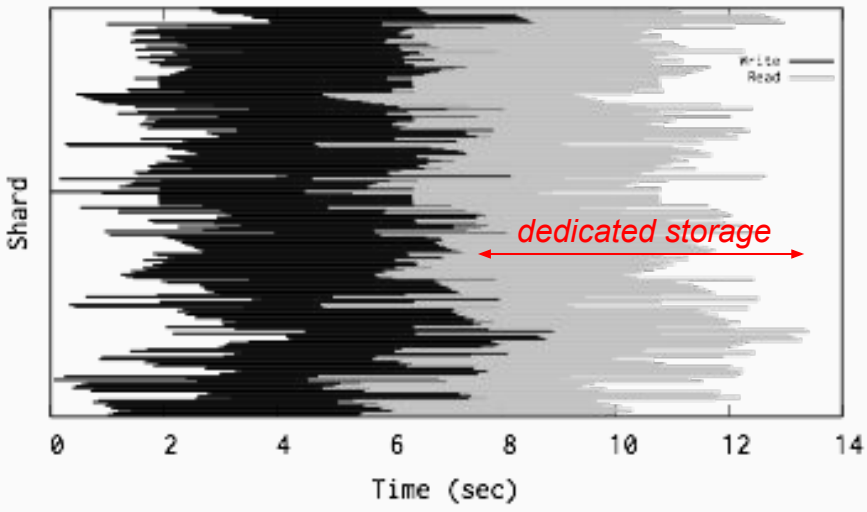
# Checkpoint without global barrier

- Application state partitioned into 256 shards
- 14 GB data set size (56 MB shards), fixed set of 12 nodes
- Tracked read-write phases for each shard

Legion, Lustre, HDF5

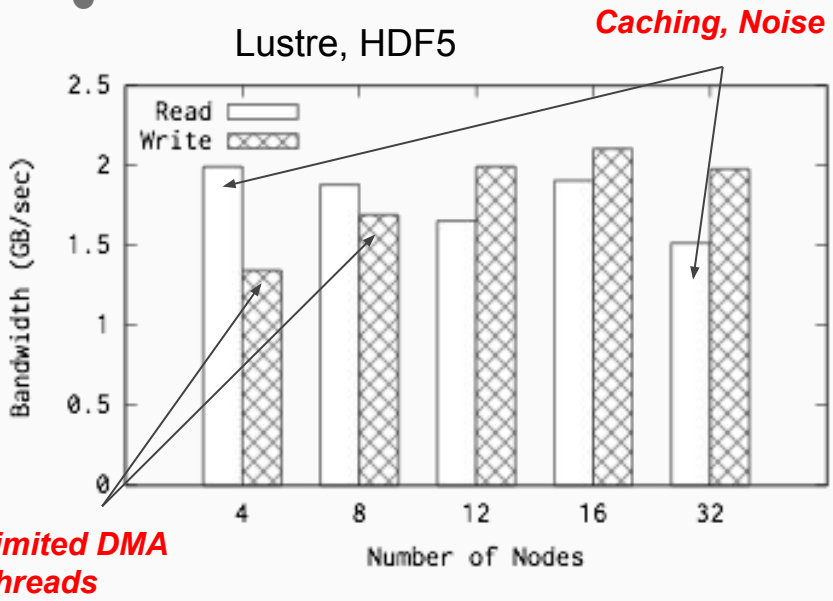


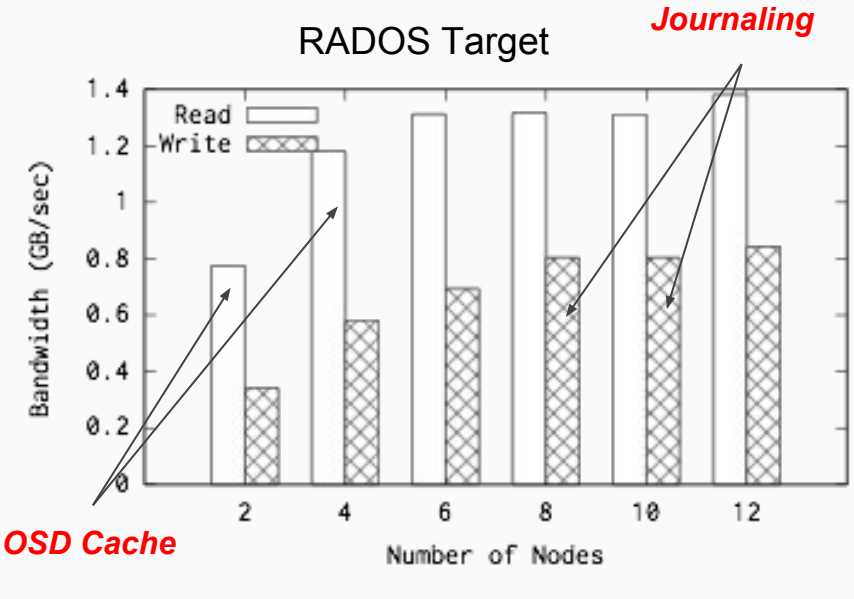

Legion, RADOS Target



# Preliminary Results: Strong Scaling

- Application state partitioned into 256 shards
- Total application state size 14 GB
- Scaled from 4 to 32 nodes (Lustre), 2 to 12 nodes (RADOS)
-



# Conclusion

- Memory hierarchies are becoming complex!
- We cannot continue to just evolve applications
- Storage should be interested into application execution models
  - Hard-coding optimizations is bad
  - Restricts flexibility and portability
- Legion runtime and programming model supports pluggable memory
- Integrate persistent storage as a memory
- Initial results show feasibility of the system design
- Enables wide range of transparent optimizations
- Questions?
  - Noah Waktins (jayhawk@soe.ucsc.edu)