Los Alamos
NATIONAL LABORATORY
EST. 1943

Optimized Scatter/Gather for Parallel Storage

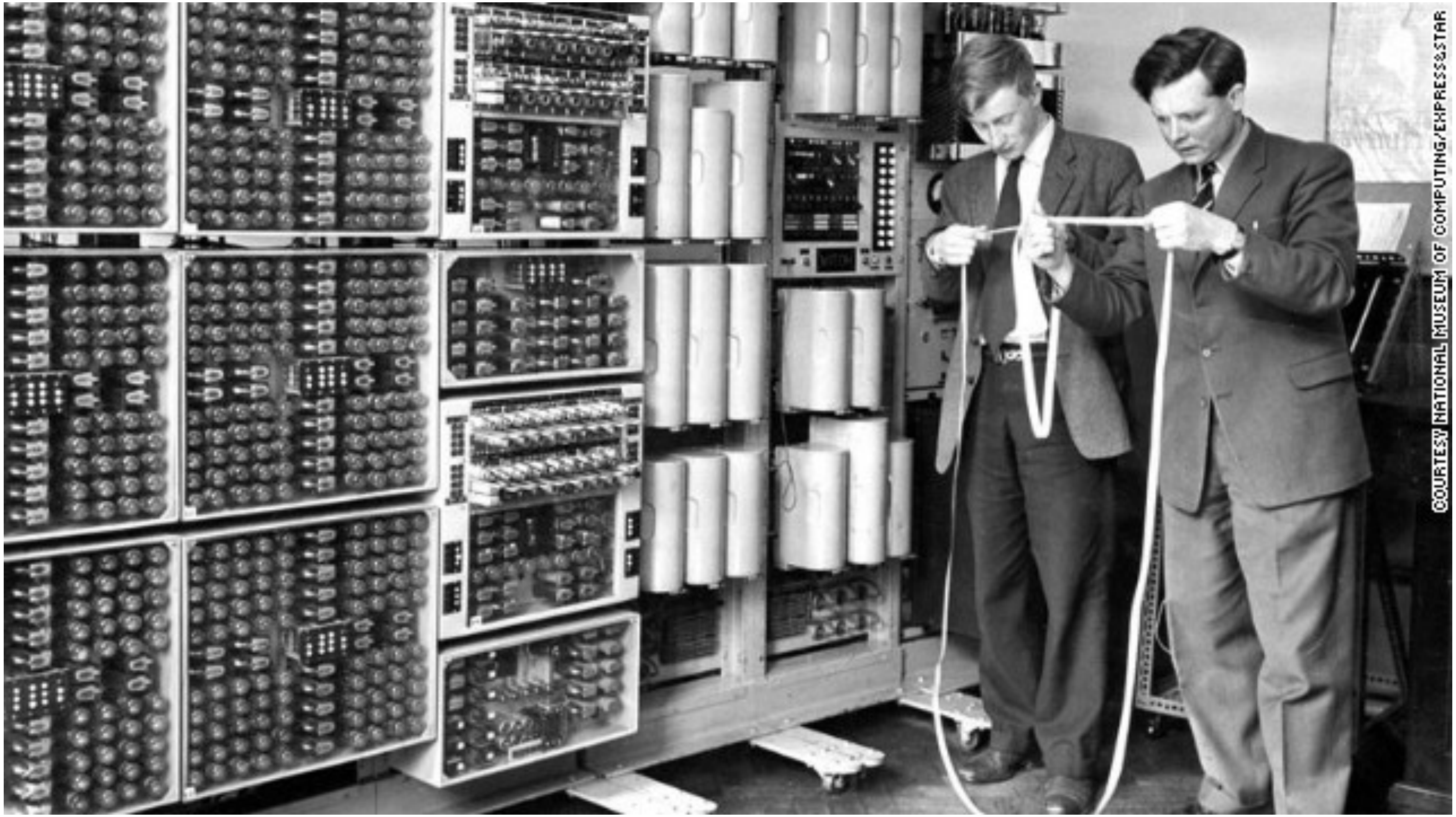
PDSW-DISCS 2017

Latchesar Ionkov
Carlos Maltzahn
Michael Lang

Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

LA-UR-17-2163

HPC Storage: Stuck in the Past



Replacing POSIX is hard

- Great interface
  - Easy to understand and use
  - Easy to implement almost correctly
  - Not scalable for shared use
  - A lot of unsettled corner cases
  - Made for programmers

- Scientists don't care about files
  - they have datasets
  - they have other things to worry about
  - best case — know how data is laid out in memory

Middleware

- Different (better?) user interface
  - HDF5
  - MPI I/O
  - ADIOS
  - ArrayQL

- Better performance
  - MPI I/O
  - PLFS
  - DeltaFS
  - GIGA+

- They all have to deal with POSIX idiosyncrasies

Complete Systems

- Huge effort
- Feature creep — even harder to finish
- Interoperability?

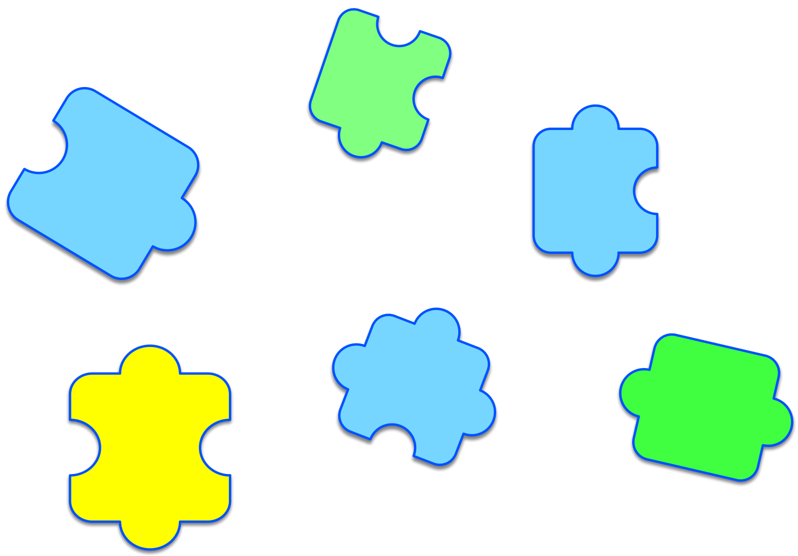
Interfaces are important

- Simple
- Not too extendable, not too many knobs
- Too much freedom is bad, the designer should make the right choices

- ASGARD tries to be the best interface for something specific
  - right level of description of data
  - for distributed environment
  - so data can be efficiently gathered from pieces scattered across many nodes
  - language and library independent

Fragments

- Describe part of the dataset
- Contiguous
- Can be materialized in memory, or stored on disk



Blocks

- Fragments consist of blocks
- Describe contiguous region of the fragment
- Can be connected to Blocks in other fragments
- Each Blocks has:
  - offset
  - size
  - list of Blocks

- Three types of Blocks

SBlock

- “Simple” Block
- Properties
  - offset
  - size
  - list of Blocks (connections, same type and size)

- Examples:
  - double -> SBlock of size 8
  - uint32_t -> SBlock of size 4

TBlock

- “sTruct” Block
- Groups other Blocks (of different sizes)
- Properties
  - offset
  - size
  - list of Blocks (fields)
  - list of Blocks (connections, same type)
- Offsets of the field Blocks relative to the start of the TBlock
- Can have holes

ABlock

- "Array" Block
- Groups Blocks of the same type and size
- Properties:
  - offset
  - dimension sizes
  - element order (row-major, column-major, etc.)
  - element Block
  - list of Destinations (connections)
- Destination
  - Block
  - (a_i, b_i, c_i, d_i, idx_i) for each dimension

$$y_{idx_i} = \frac{a_i x_i + b_i}{c_i x_i + d_i}$$

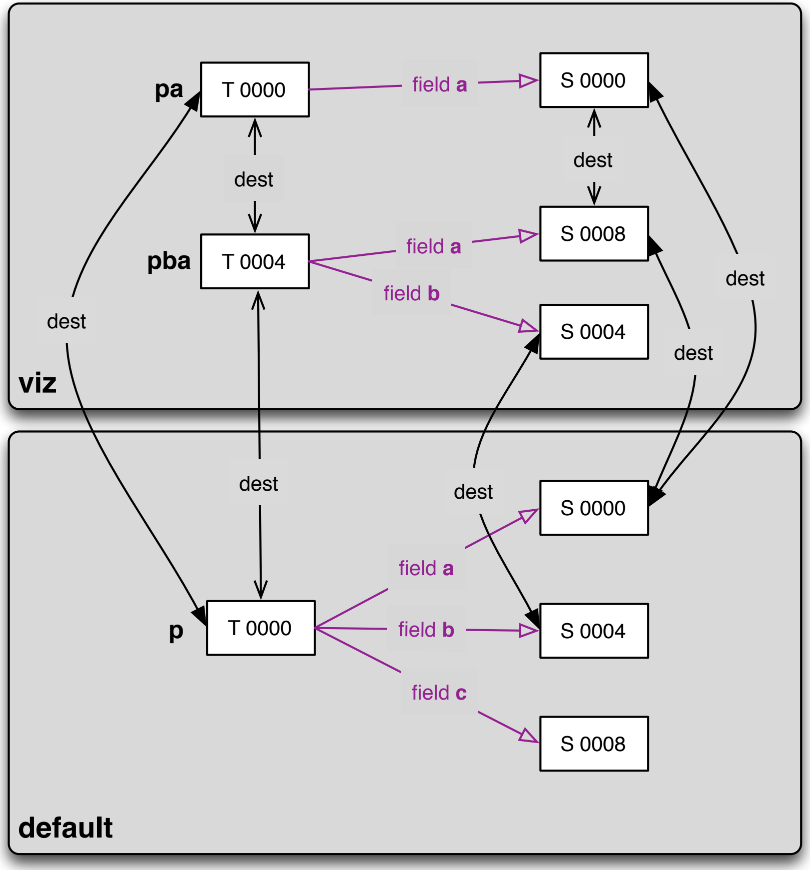
Fragment

- Fragment
  - Collection of Blocks
  - Top-level Blocks

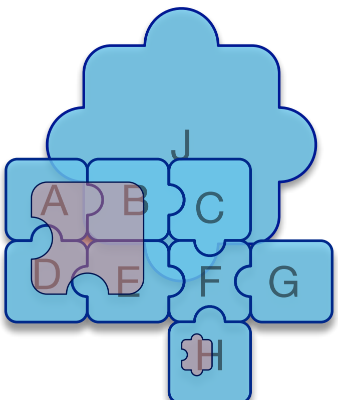
- Transformation (*src*, *dest*)
  - For each top-level block in *src*
    - SBlock — copy to each destination Block $\in$ *dest*
    - TBlock — recursively run for each field (keep offsets)
    - ABlock — for each element with index [$x_1, x_2, \ldots, x_n$]
      - calculate offset in *src*
      - for each destination Block $\in$ *dest*
        - calculate index [$y_1, y_2, \ldots, y_n$] in *dest*
        - calculate offset
        - recursively run transformation for the element Block

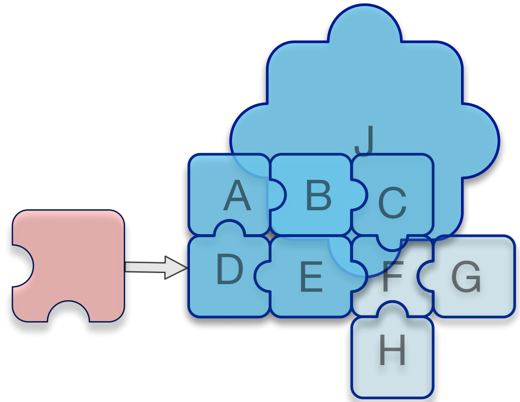
Transformation Rules

```
fragment dataset {
    var p struct {
        a, b, c float32
    }
}

fragment default {
    var p = p
}

fragment viz {
    var pa { a } = p
    var pba { b, a } = p
}
```

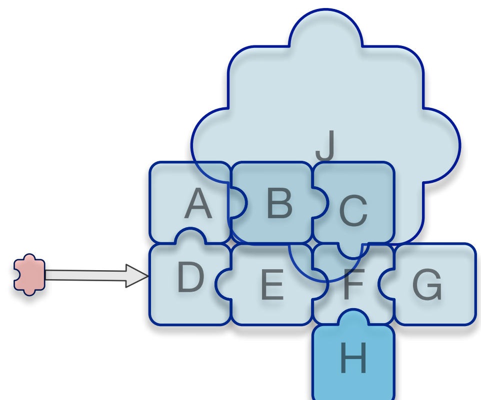


Fragment Sources



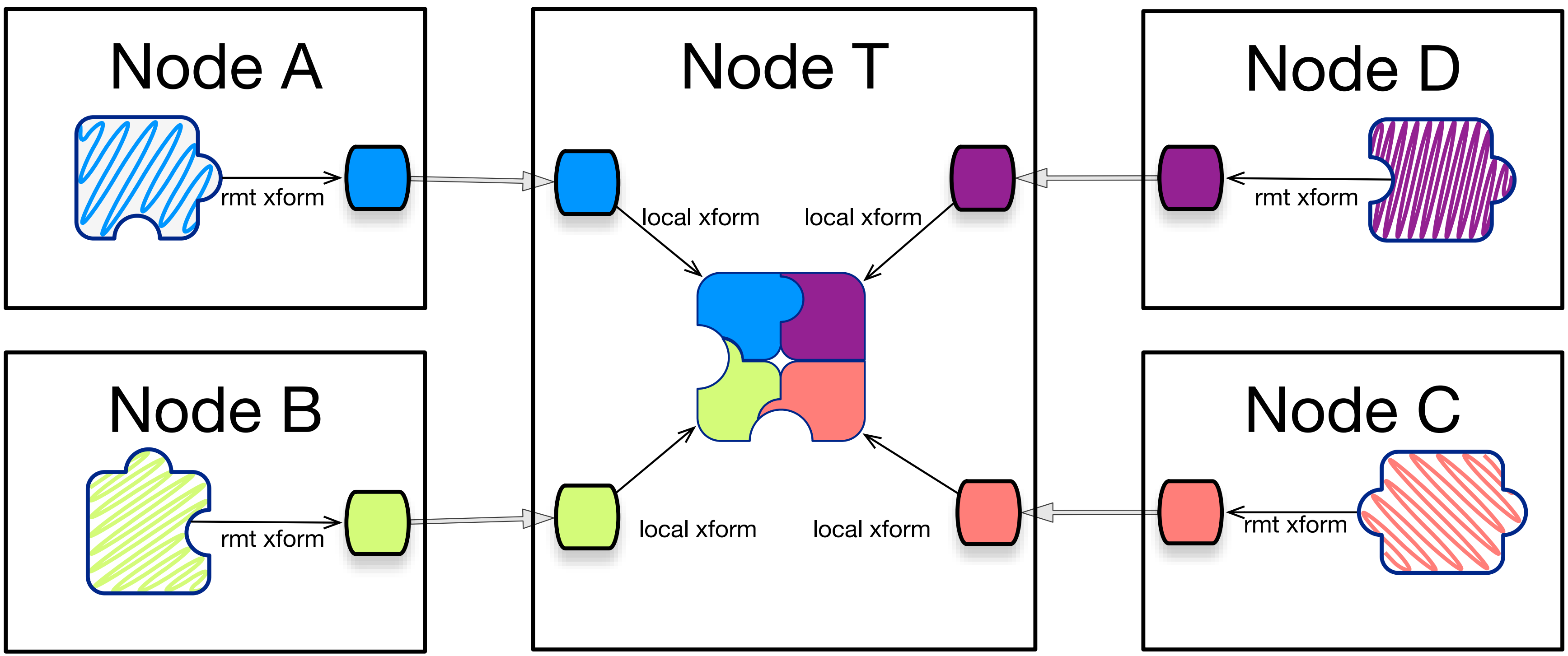


(A:0.25) (A:0.25 J:0.25)
(B:0.3) (B:0.2 J:0.2)
(D:0.4) (D:0.1 J:0.1)
(E:0.2) (E:0.3 J:0.3)

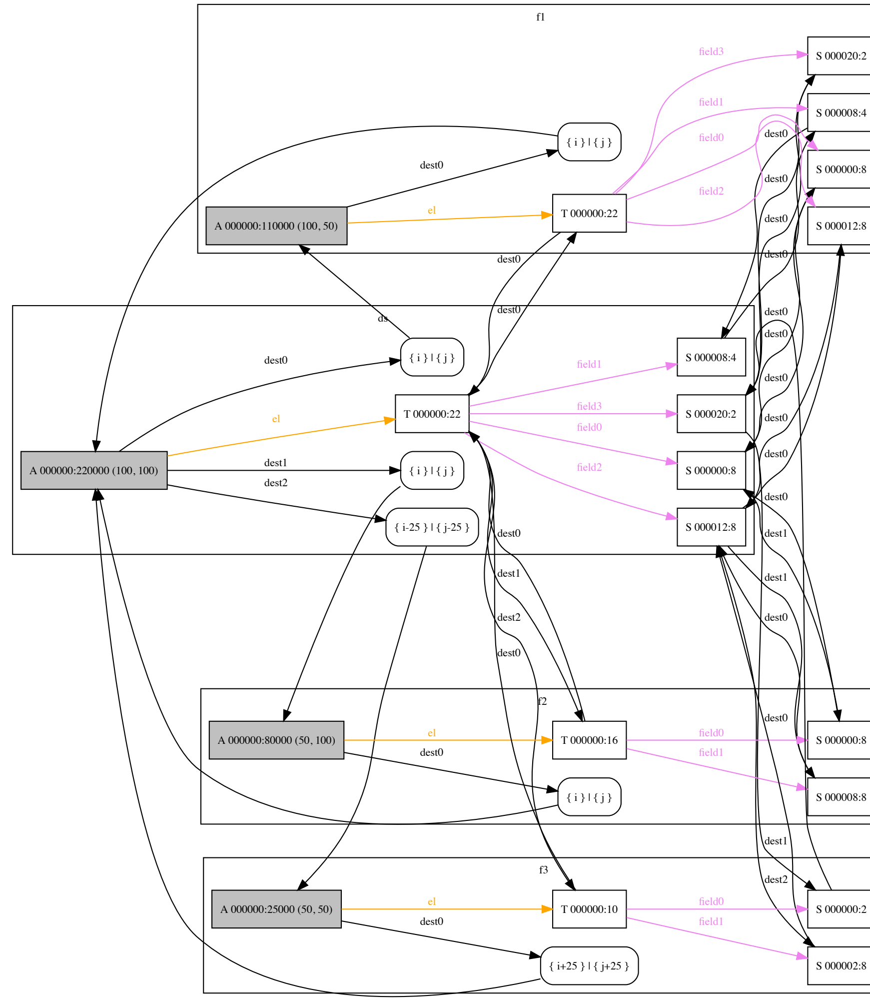


(H:1.0)

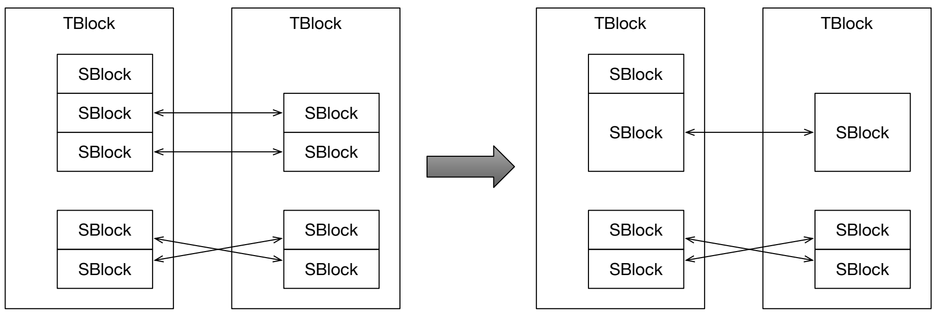
Transformation Rules



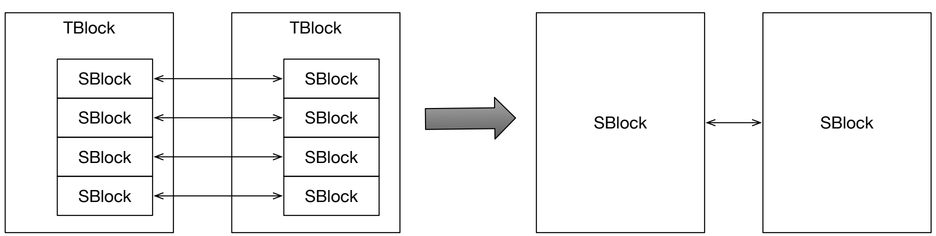
```
fragment ds {
    type P struct {
        a float64
        b float32
        c float64
        d int16
    }

    var data [100, 100] P
}

fragment f1 {
    var d1 = data
}

fragment f2 {
    var d2 { a, c } = data
}

fragment f3 {
    var d3[i, j] {d, c} =
                        data[i-25, j-25]
}
```



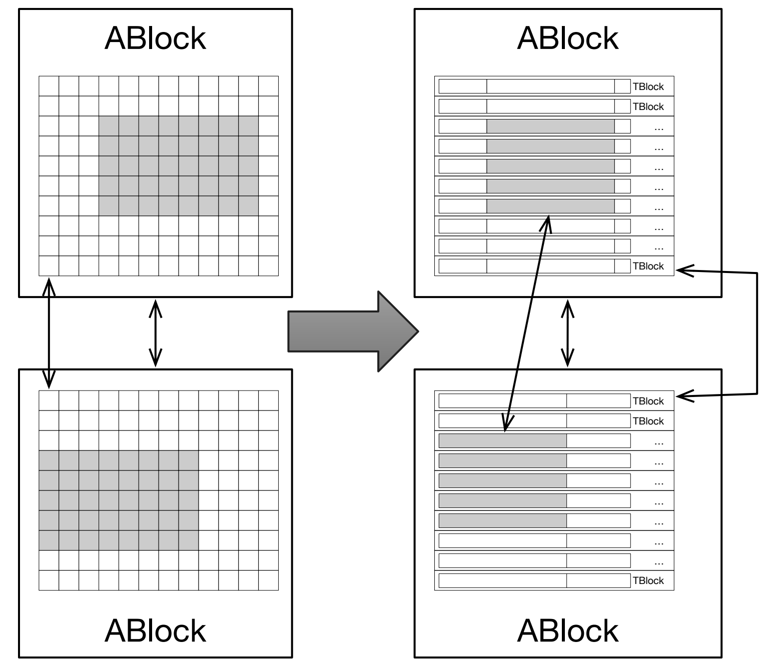
Optimizations



a. Merging neighboring fields



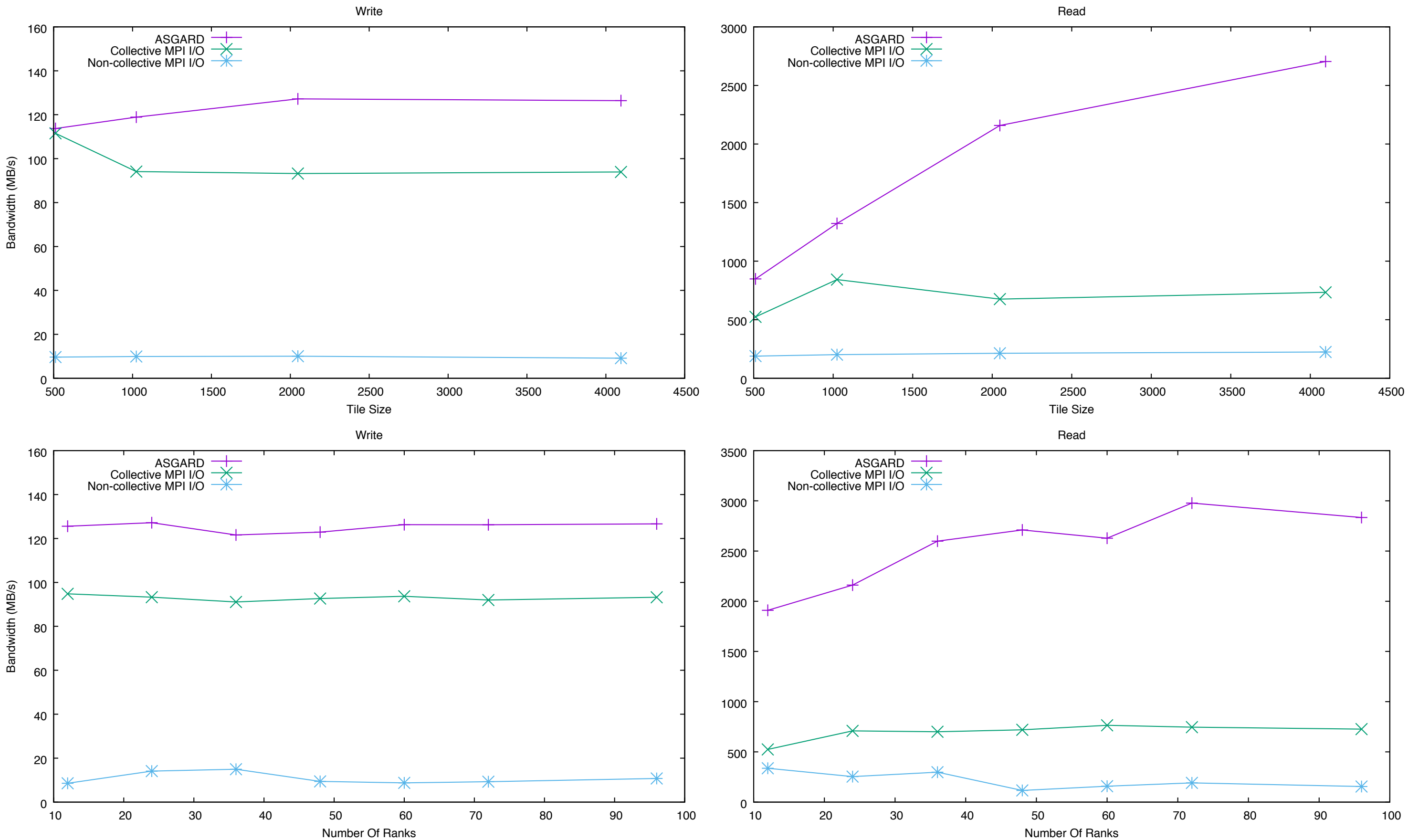
b. Replacing TBlock with a SBlock



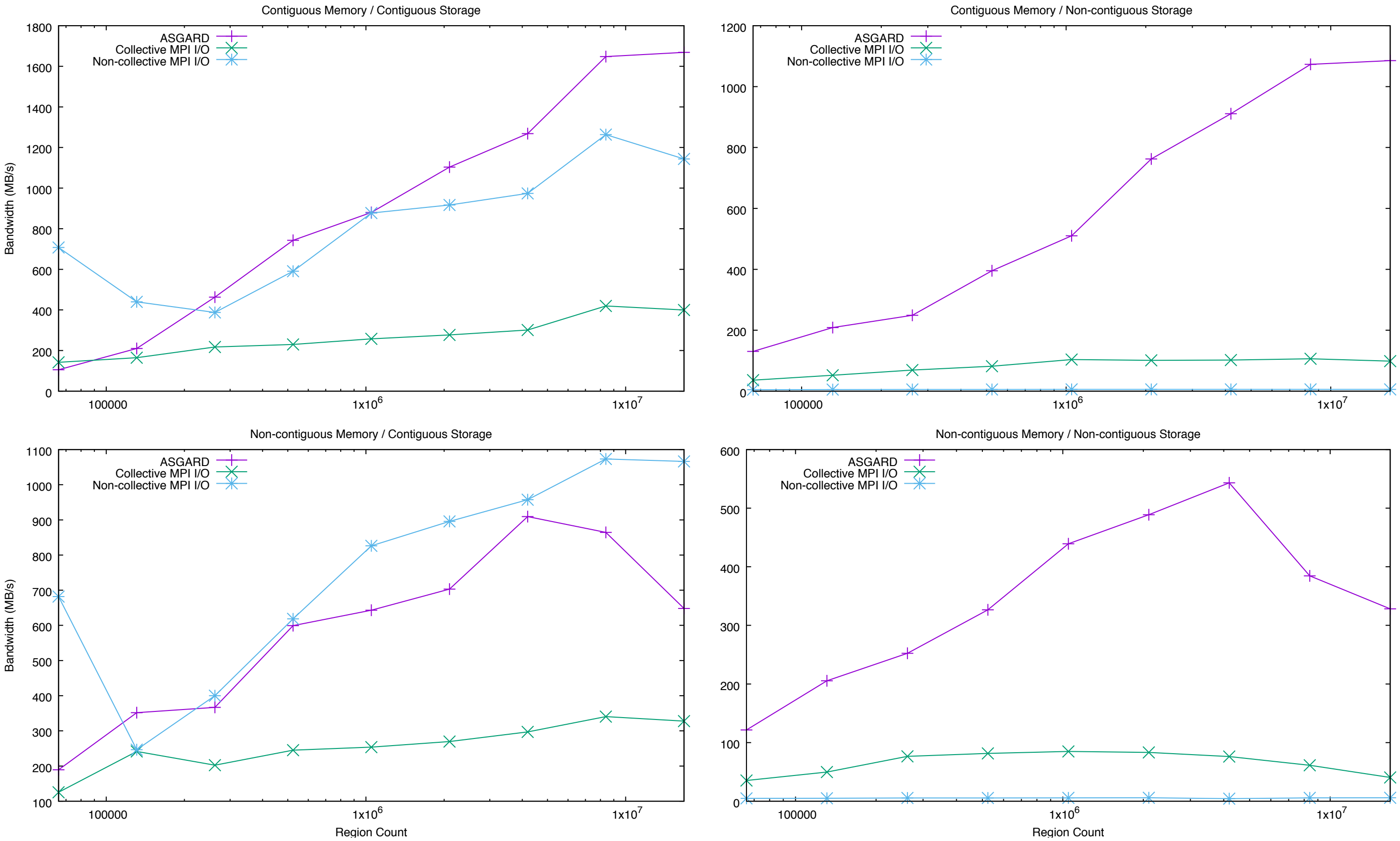
Ceph Integration

- RADOS Objects - custom object class extension
- Dataset object
  - metadata: dataset + stripe definitions
  - no data
- Stripe object
  - partial read/write using transformation rules
  - write triggers updates to secondary replicas
- Client Side
  - access unit is fragment
  - server sends back list of objects and transformation rules
  - executes local transformation rules, sends to OSD remote transformation rules (+ data)

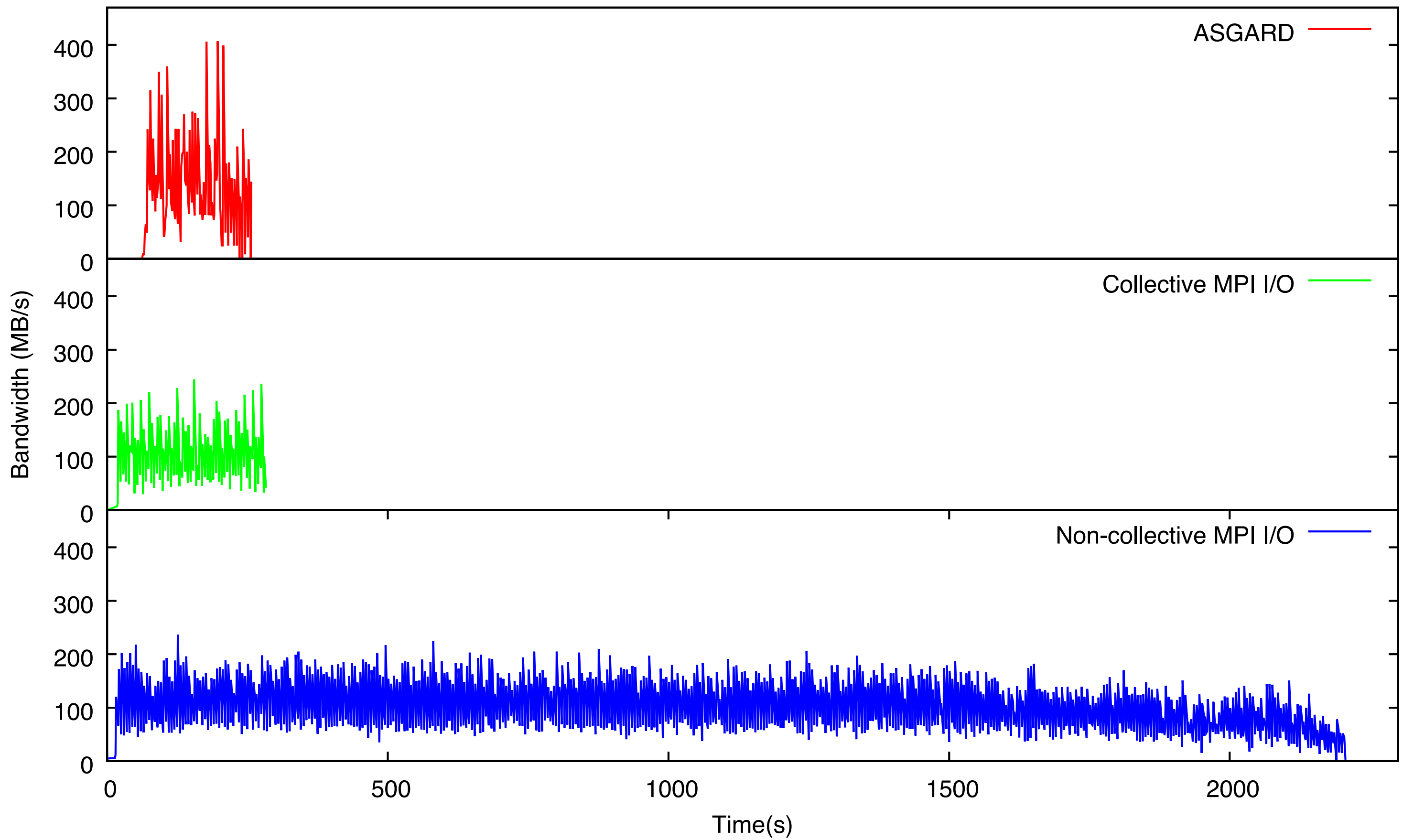
Results: MPI Tile I/O



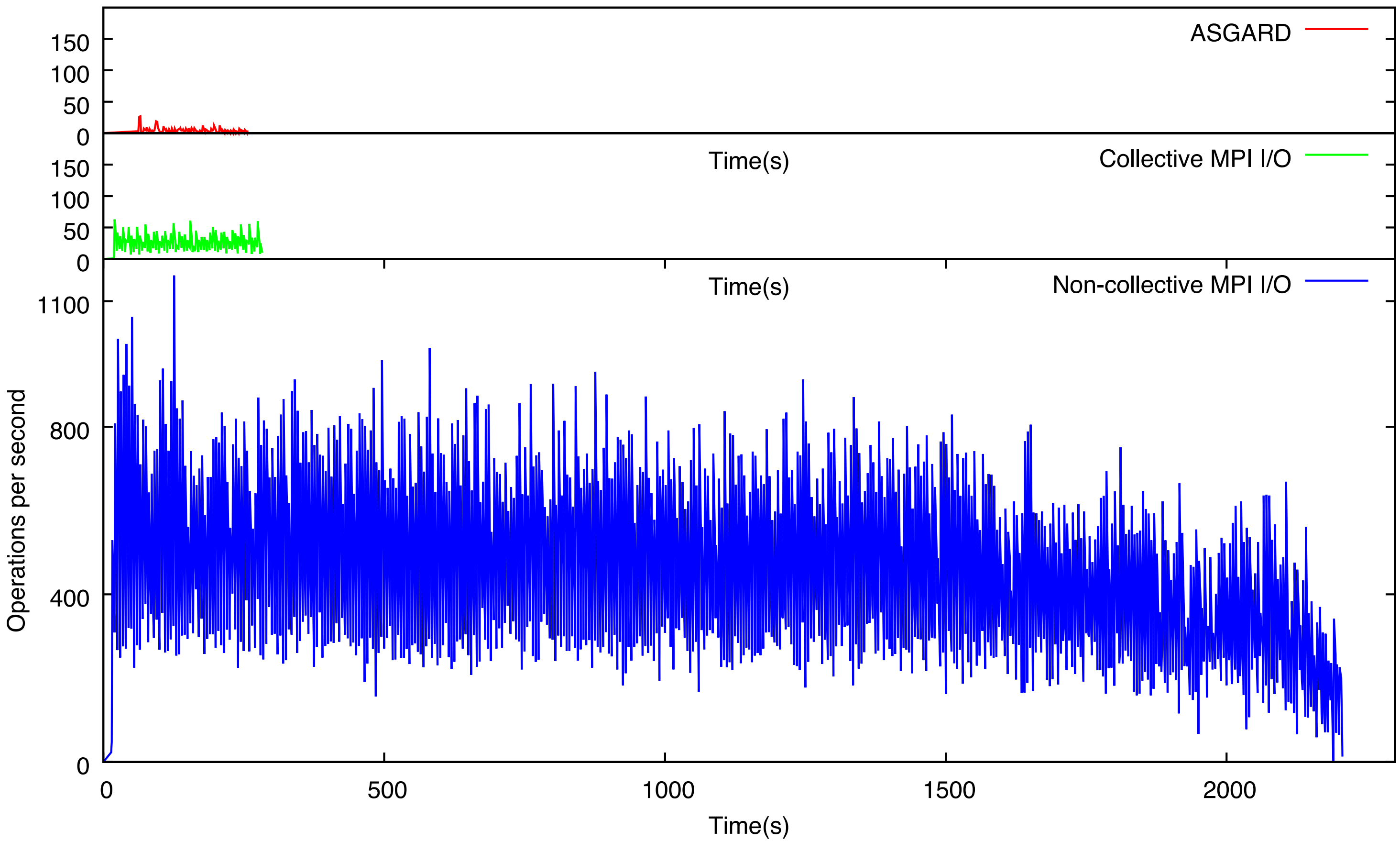
Results: HPIO Read



Ceph Bandwidth



Ceph Operations



Conclusions

- **ASGARD defines language and library independent data description**
- **Compact transformation rules**
- **Small transformation engine (3K LOC) with implementations in Go and C**
- **Easy to integrate in storage systems and libraries**

- **Questions:**
  - **is *it* the right level of data description?**
  - **does it make sense to push for general file systems?**
  - **what else did we miss?**
  - **do we need byte order (LSB, MSB) and/or primary type (IEEE 754, integer)?**