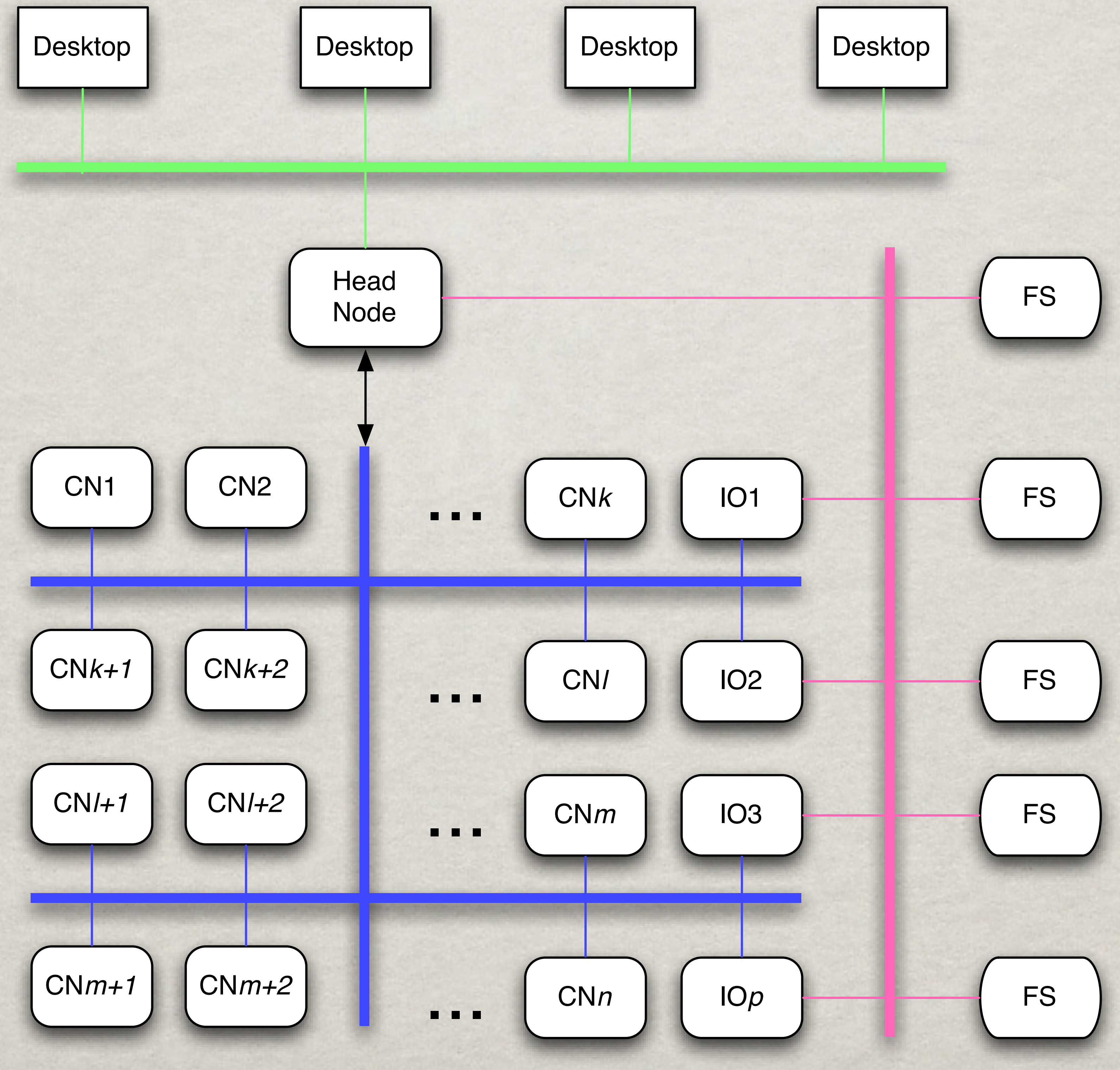
DRepl

Optimizing Access to Application Data for Analysis and Visualization

Latchesar Ionkov
Michael Lang
LANL

Carlos Maltzahn
UCSC

HPC Cluster


Desktop
Desktop
Desktop
Desktop
Head Node
FS
CN1
CN2
...
CN*k*
IO1
FS
CN*k+1*
CN*k+2*
...
CN*l*
IO2
FS
CN*l+1*
CN*l+2*
...
CN*m*
IO3
FS
CN*m+1*
CN*m+2*
...
CN*n*
IO*p*
FS

Data Storage

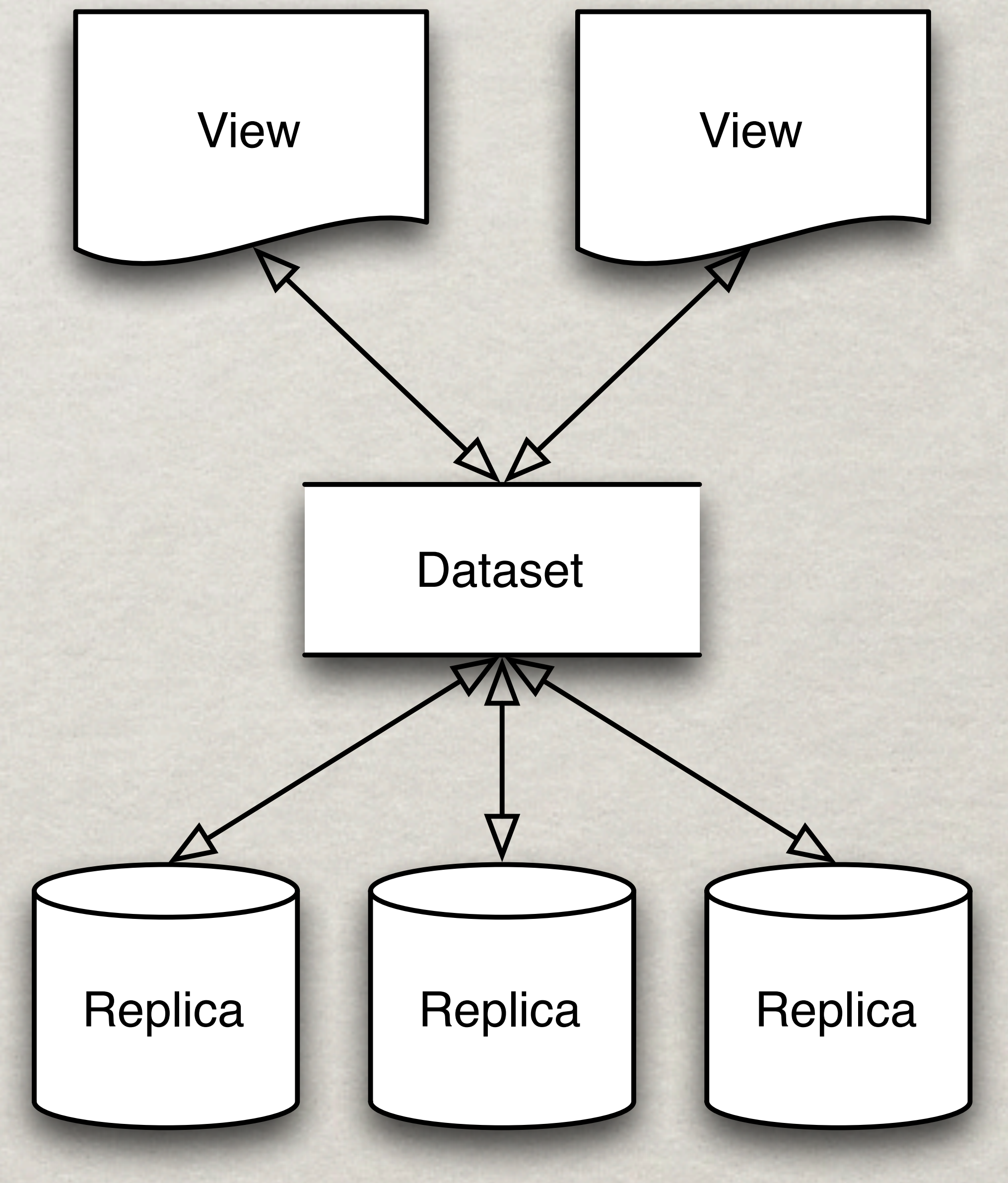
- Data stored in files
- Many applications use legacy formats
- Data is stored in format, convenient for the producer
- In-situ and in-transit data analysis slow

Objective

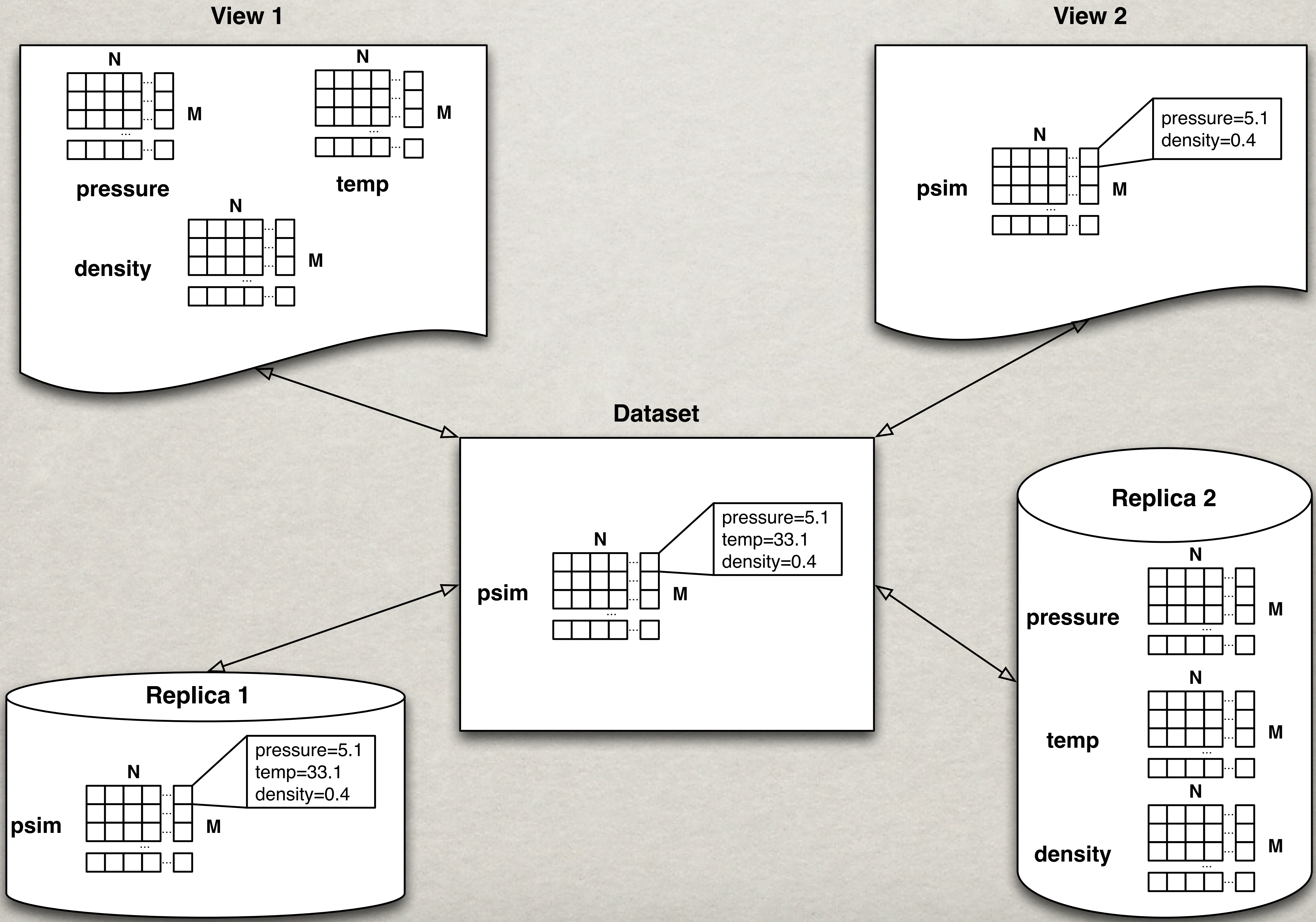
- Decouple storage data layout from application data layout(s)
- Make replicas with different data layouts
- Each application working with the data can use a layout that is optimized for it
- Allow both materialized (on-storage) and on-the-fly data layouts

Design

- Definitions
  - **Dataset** -- abstract data model
  - **Views** -- how applications see the data
  - **Replicas** -- how the data is stored
- Provision of an easy way to express how data is used by the applications

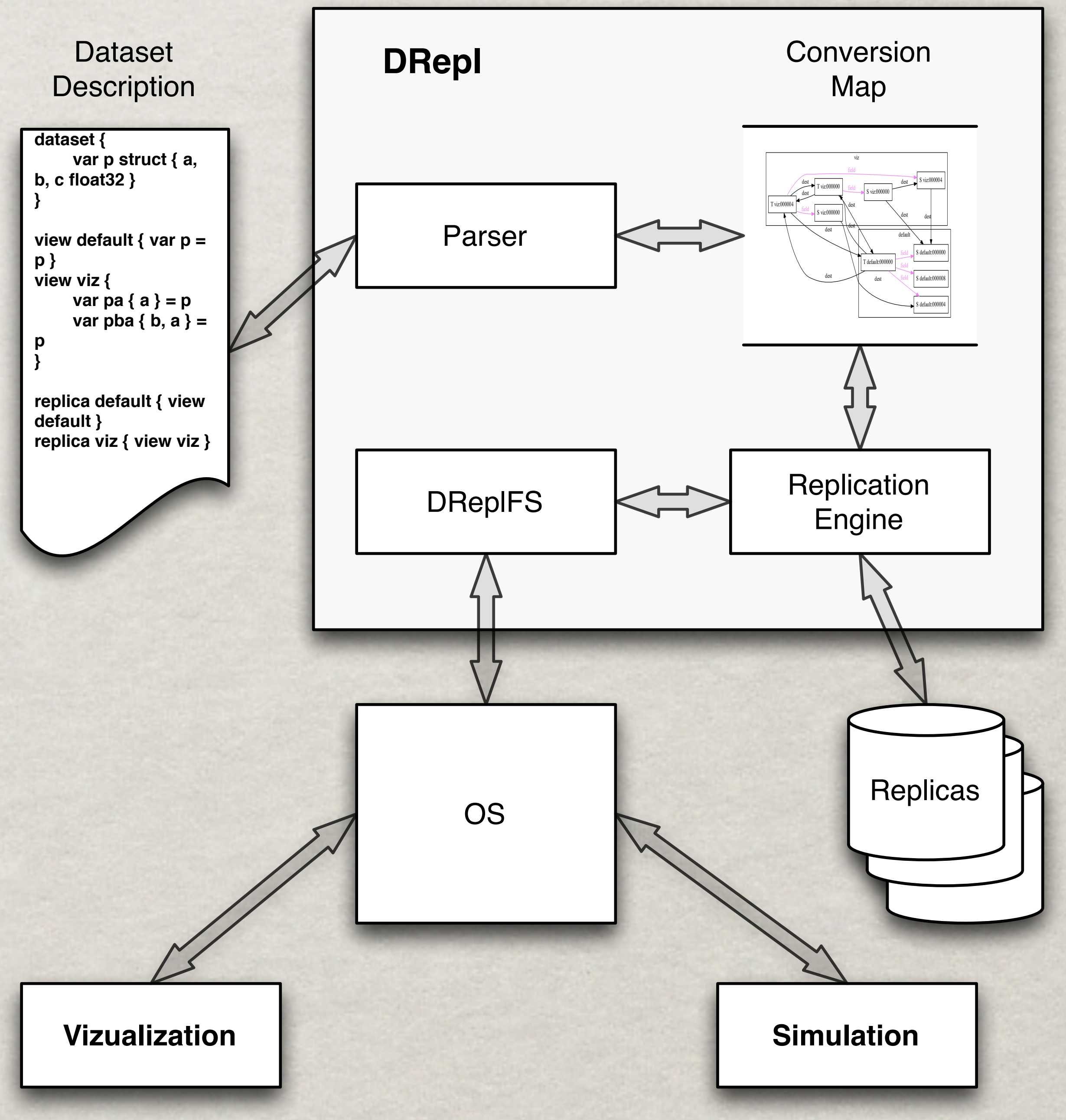


Example


View 1
N
M
pressure
N
M
temp
N
M
density
View 2
N
psim
M
pressure=5.1
density=0.4
Dataset
psim
N
M
pressure=5.1
temp=33.1
density=0.4
Replica 1
psim
N
M
pressure=5.1
temp=33.1
density=0.4
Replica 2
N
pressure
M
N
temp
M
N
density
M

DRepl

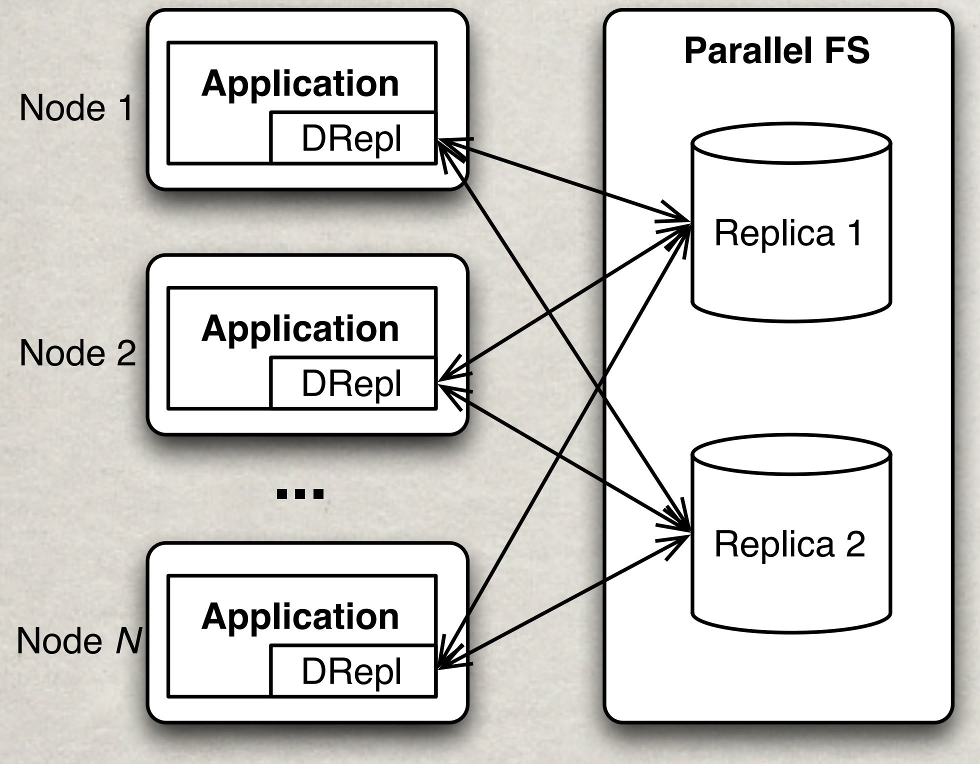
- Dataset Language
- Parser
- Replication Engine
- File Server



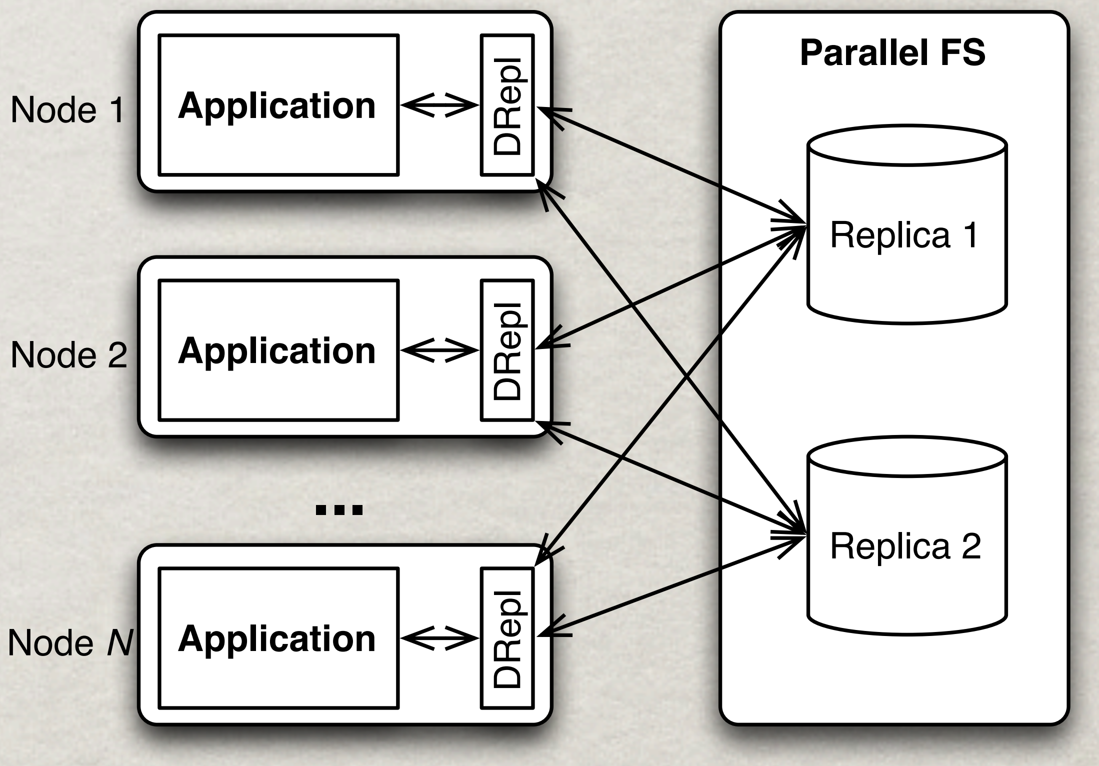
```
dataset {
        var p struct { a,
b, c float32 }
}

view default { var p =
p }
view viz {
        var pa { a } = p
        var pba { b, a } =
p
}

replica default { view
default }
replica viz { view viz }
```

Configurations

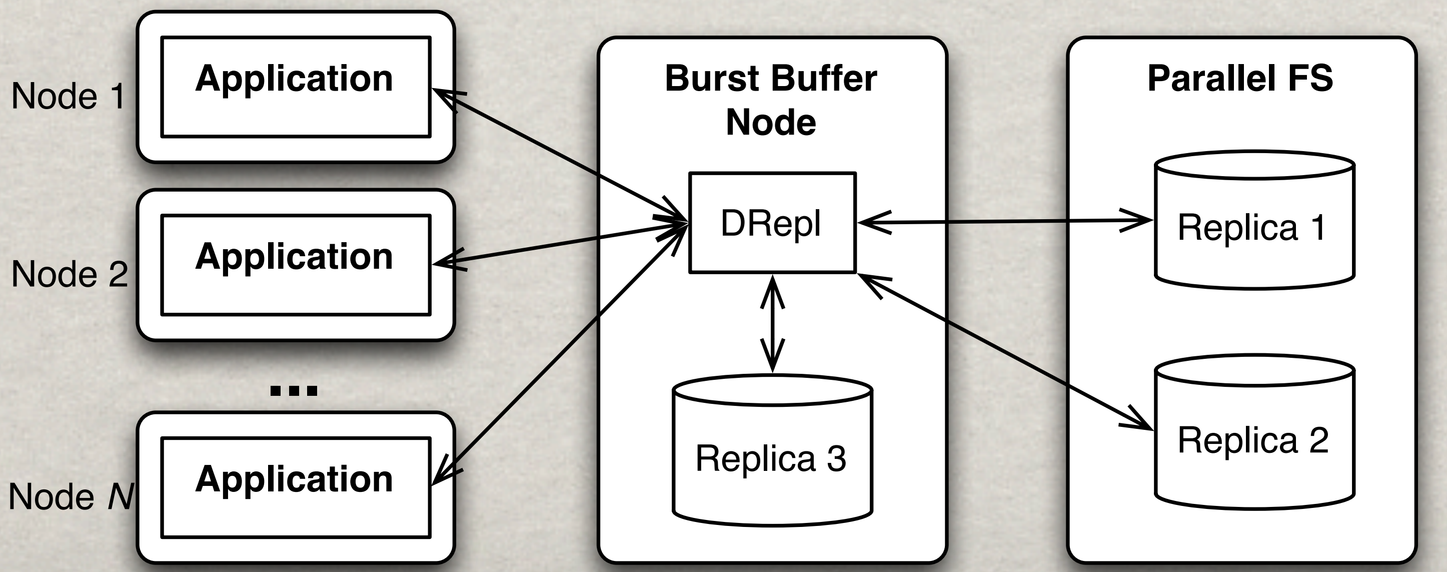
Embedded



Separate



Burst Buffer



Dataset Language

- Syntax Similar to C, C++, Java
- Dataset
  - define data types (structs, arrays)
  - define named data of the types
- View(s)
  - define substructs and subarrays
  - define named data based on the dataset data
- Replica(s)

Dataset Language

- Primary types - **`int8, int16, int32, int64, float32, float64, stringN`**

- Structs

```
struct {
    a, b, c  float64
}
```

- Multidimensional arrays

```
[50,40,21] Point
```

- Custom types

```
type int64 Point
```

- Arithmetic expressions in the subarray definitions

a[i*3, j + 2] = aa[j, i - 1]

- Support for different array orders -- row-major, row-minor, in future Hilbert and z-order

Language Example

```
dataset {
  const N = 500

  type Data struct {
    a, b, c float32
  }

  var data [N]Data
}

view array-of-structs {
  var ds = data
}

view struct-of-arrays {
  var a[i]{a} = data[i]
  var b[i]{b} = data[i]
  var c[i]{c} = data[i]
}
```

```
view ab rowmajor {
  var ab[i]{a,b} = data[i]
}

replica array-of-structs {
  view array-of-structs
}

replica struct-of-arrays {
  view struct-of-arrays
}

replica other {
  view array-of-structs
  view ab
}
```

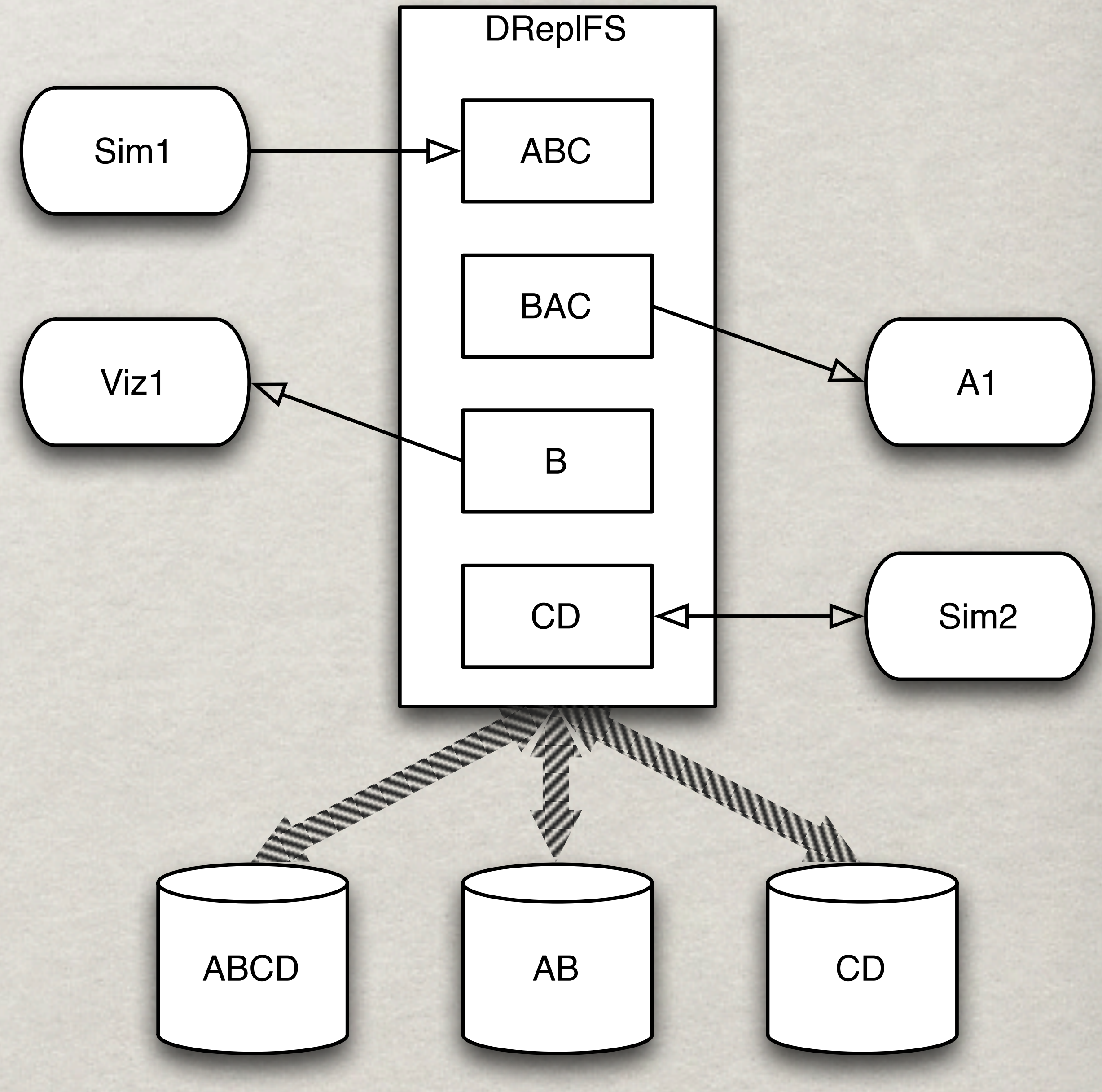
Subarray Examples

```
dataset {
  const N = 500
  const M = 200

  var data [N, M]float32
}

view v {
  // flip dimensions
  var flip[i,j] = data[j,i]

  // middle row
  var mr[i] = data[N/2, i]

  // each third element
  var te[i, j] = data[i*3, j*3]
}
```

DReplFS

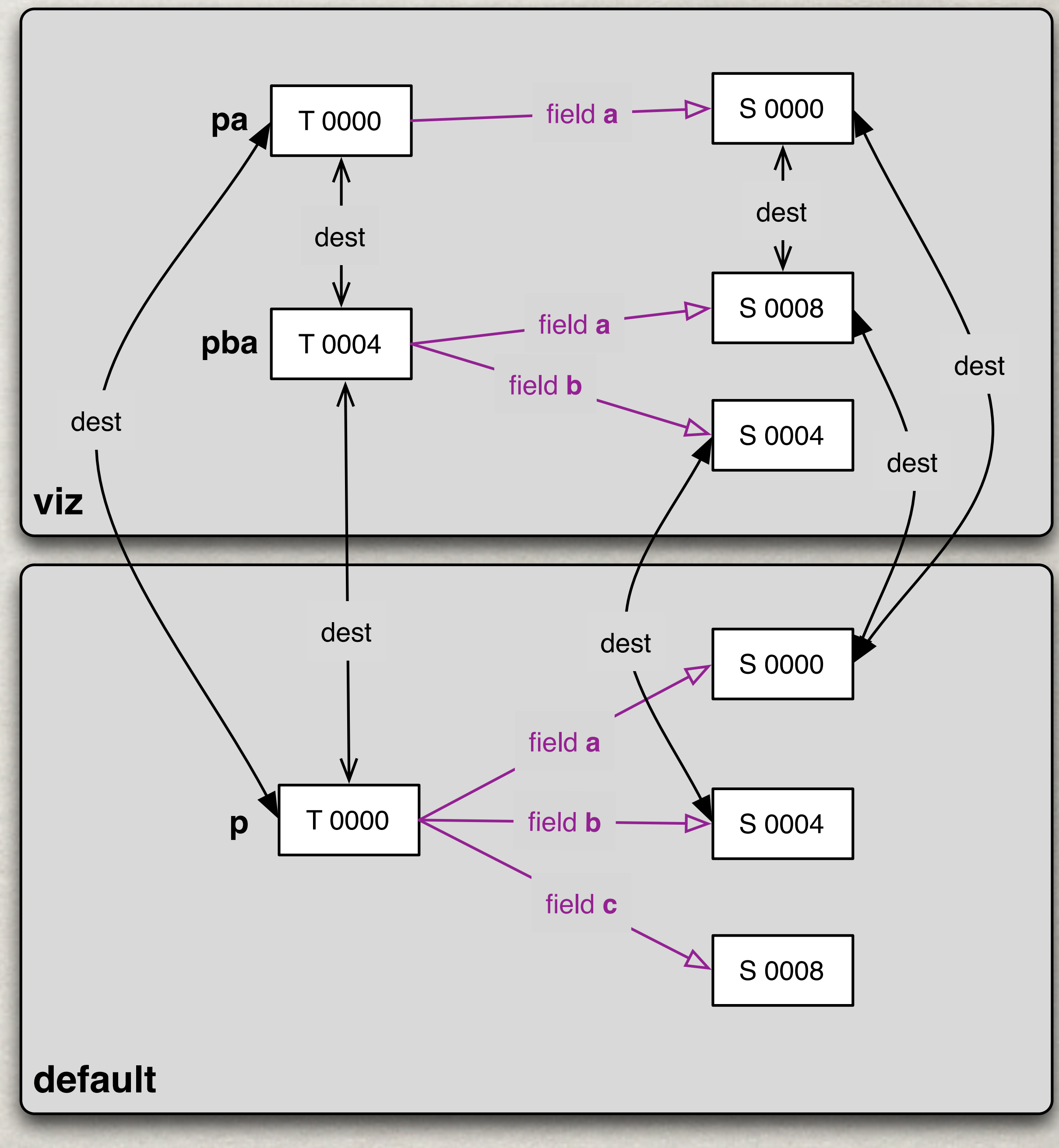
- Represent the application data formats (**views**) as virtual files

- Stored data formats (**replicas**) -- collection of replicas



Transformation Rules

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

 view default {
    var p = p
 }

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



Implementation

- DReplFS -- Parser, Replication Engine, File Server in Go
- KDreplFS -- Parser in Go, Replication Engine and File Server in the Linux kernel

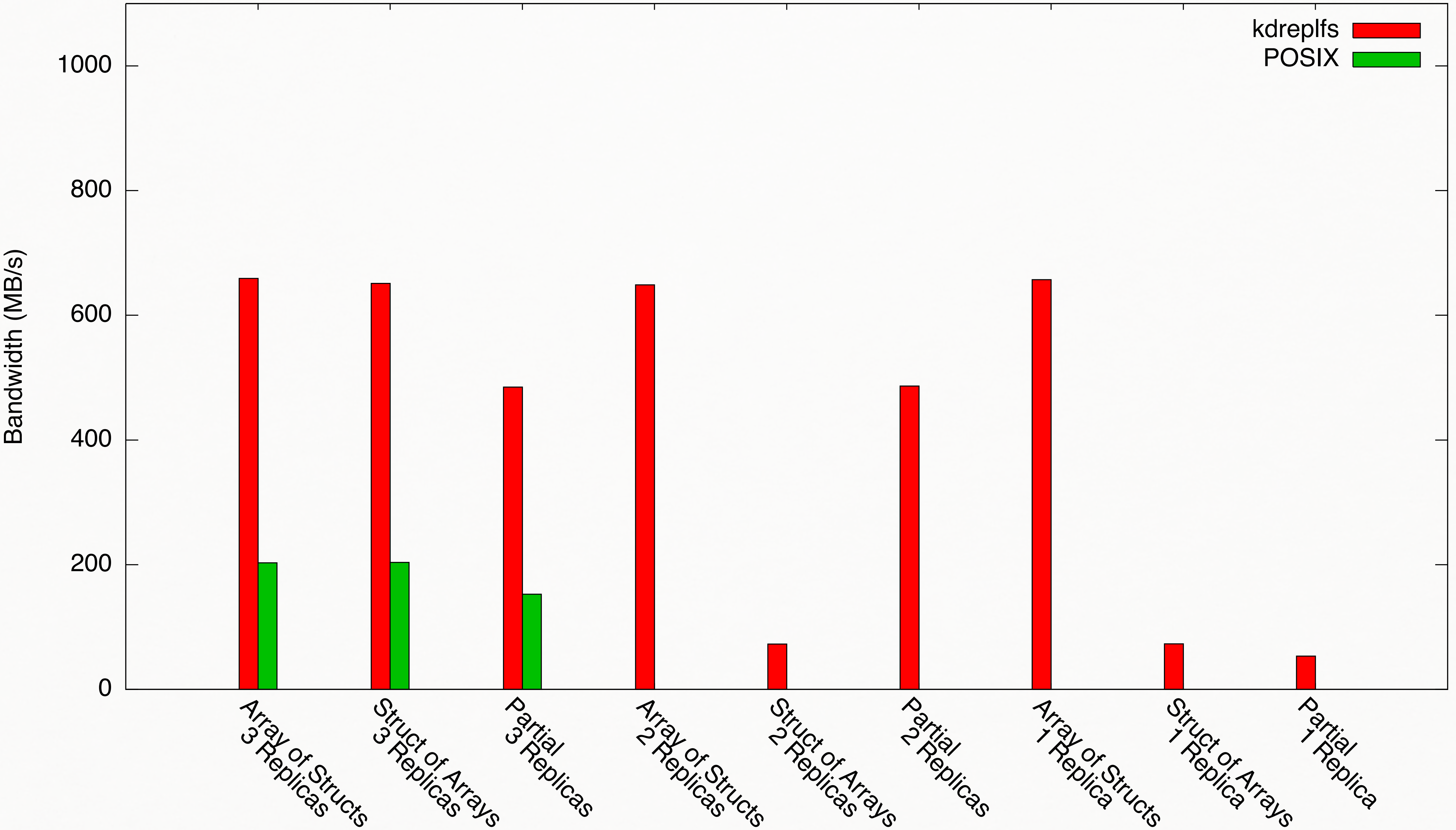
Experiments

- Dataset

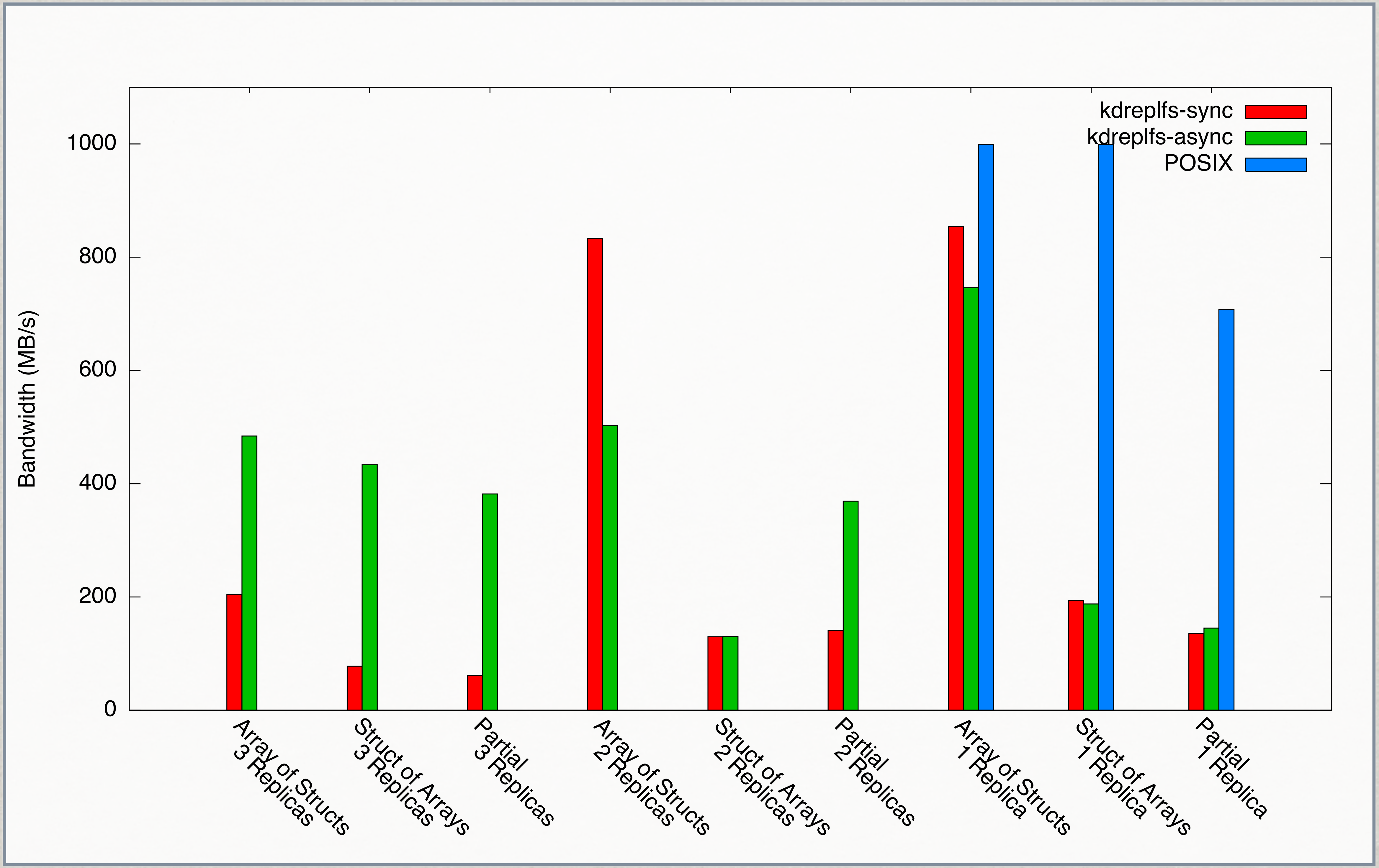
```
const N = 176160768
type Data struct {
  a, b, c float32
}
var data [N]Data
```

- Views
  - array of structs (AOS)
  - struct of arrays (SOA)
  - partial (only b)
- Replicas
  - three replicas (AOS, SOA, b)
  - two replicas (AOS, b)
  - one replica (AOS)
- Each replica on separate SSD
- File Servers
  - pass-through (POSIX)
  - kdreplfs

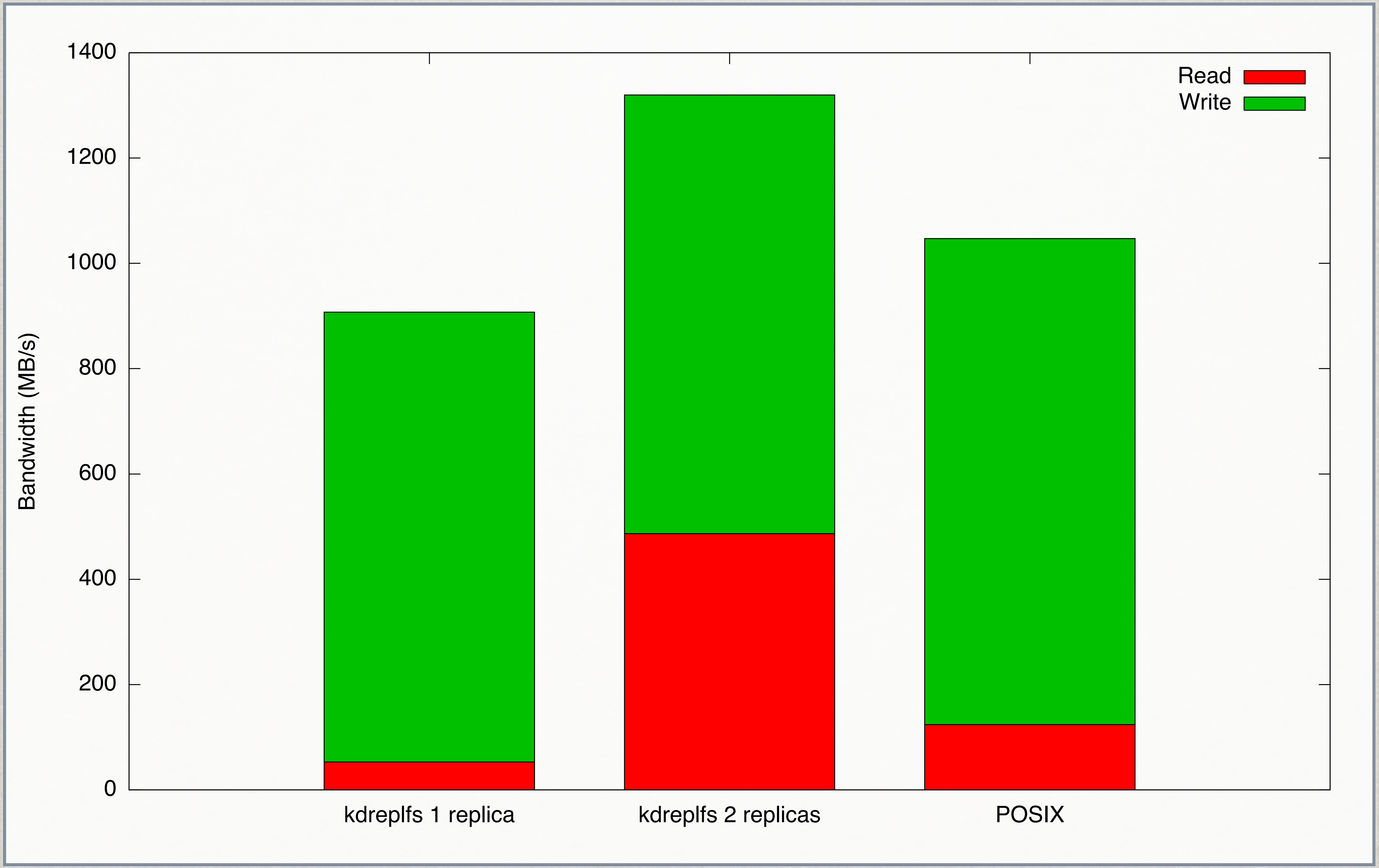
Results: Read



Results: Write



Results: Combined



Future Work

- Variable-sized arrays
- More array element orders (z-order, Hilbert)
- Optimizations
- Endianness for primary types
- Support for HDF5 replicas
- Implementation that doesn't use file servers
- Automatic generation of dataset definition from standard data formats (HDF5, NetCDF)