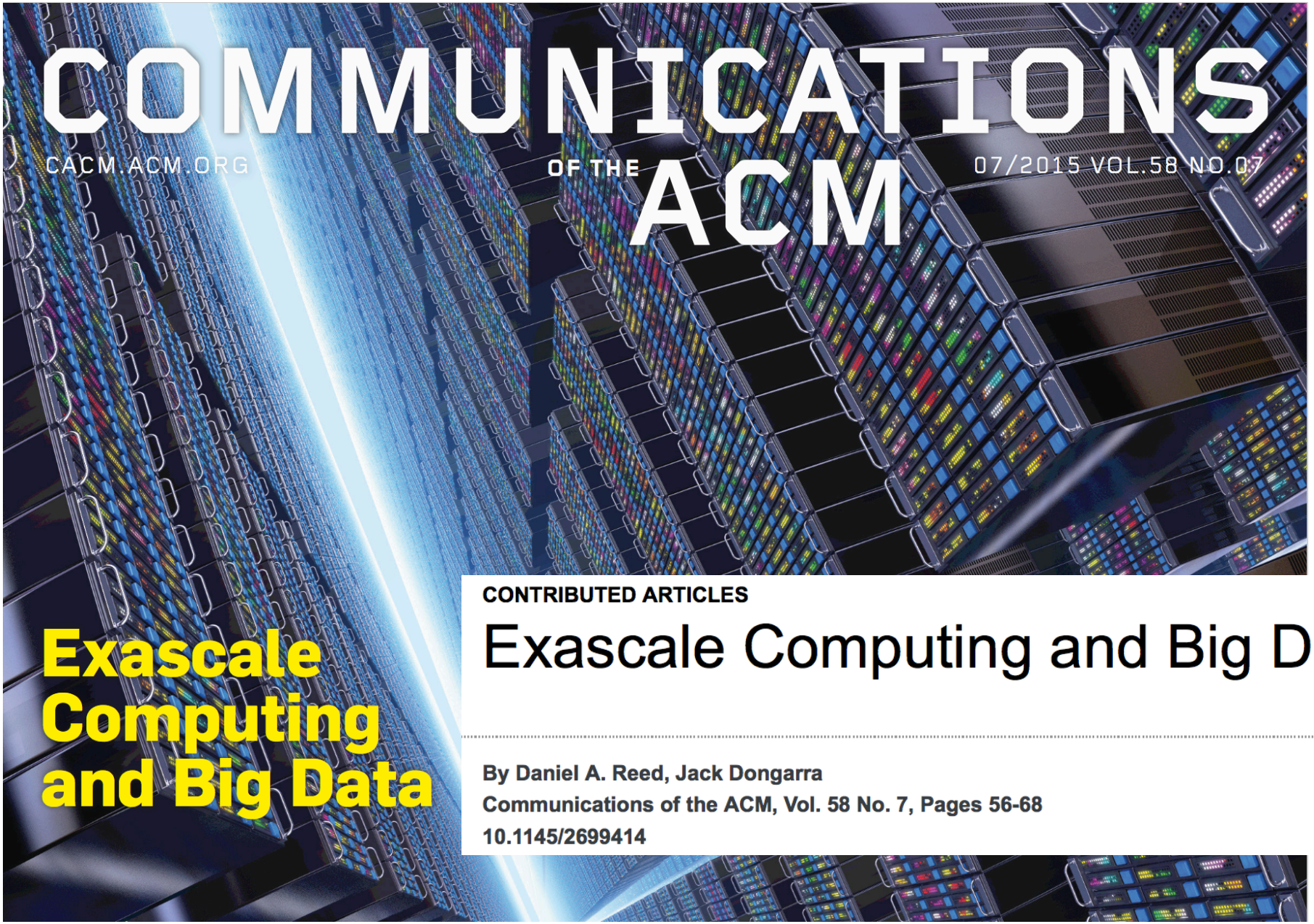
Storage-Based Convergence Between HPC and Big Data

Pierre Matri*, Alexandru Costan†, **Gabriel Antoniu**◇, Jesús Montes*, María S. Pérez*

* Universidad Politécnica de Madrid, Madrid, Spain — † INSA Rennes / IRISA, Rennes, France — ◇ Inria Rennes Bretagne-Atlantique, Rennes, France

ORAP Forum, Paris, March, 2017



CONTRIBUTED ARTICLES

Exascale Computing and Big Data

By Daniel A. Reed, Jack Dongarra

Communications of the ACM, Vol. 58 No. 7, Pages 56-68
10.1145/2699414

Big Data and Extreme Computing

High-end data analytics and HPC are both essential elements of an integrated computing research-and-development agenda;

- Big compute generates and is needed to analyze big data
- Networking and memory performance are critical to both

Programming models and tools are perhaps the biggest point of divergence between the scientific-computing and big-data ecosystems.

Credits: Jack Dongarra

National Strategic Computing Initiative (NSCI)

White House Executive Order

1. ***Accelerating delivery of a capable exascale computing system*** that integrates hardware and software capability to deliver approximately 100 times the performance of current 10 petaflop systems across a range of applications representing government needs.
2. ***Increasing coherence between the technology base used for modeling and simulation and that used for data analytic computing.***
3. Establishing, over the next 15 years, ***a viable path forward for future HPC systems even after the limits of current semiconductor technology are reached*** (the "post- Moore's Law era").
4. ***Increasing the capacity and capability of an enduring national HPC ecosystem*** by employing a holistic approach that addresses relevant factors such as networking technology, workflow, downward scaling, foundational algorithms and software, accessibility, and workforce development.
5. ***Developing an enduring public-private collaboration*** to ensure that the benefits of the research and development advances are, to the greatest extent, shared between the United States Government and industrial and academic sectors.

Big Data and Extreme Computing workshops series (BDEC)

http://www.exascale.org/bdec/

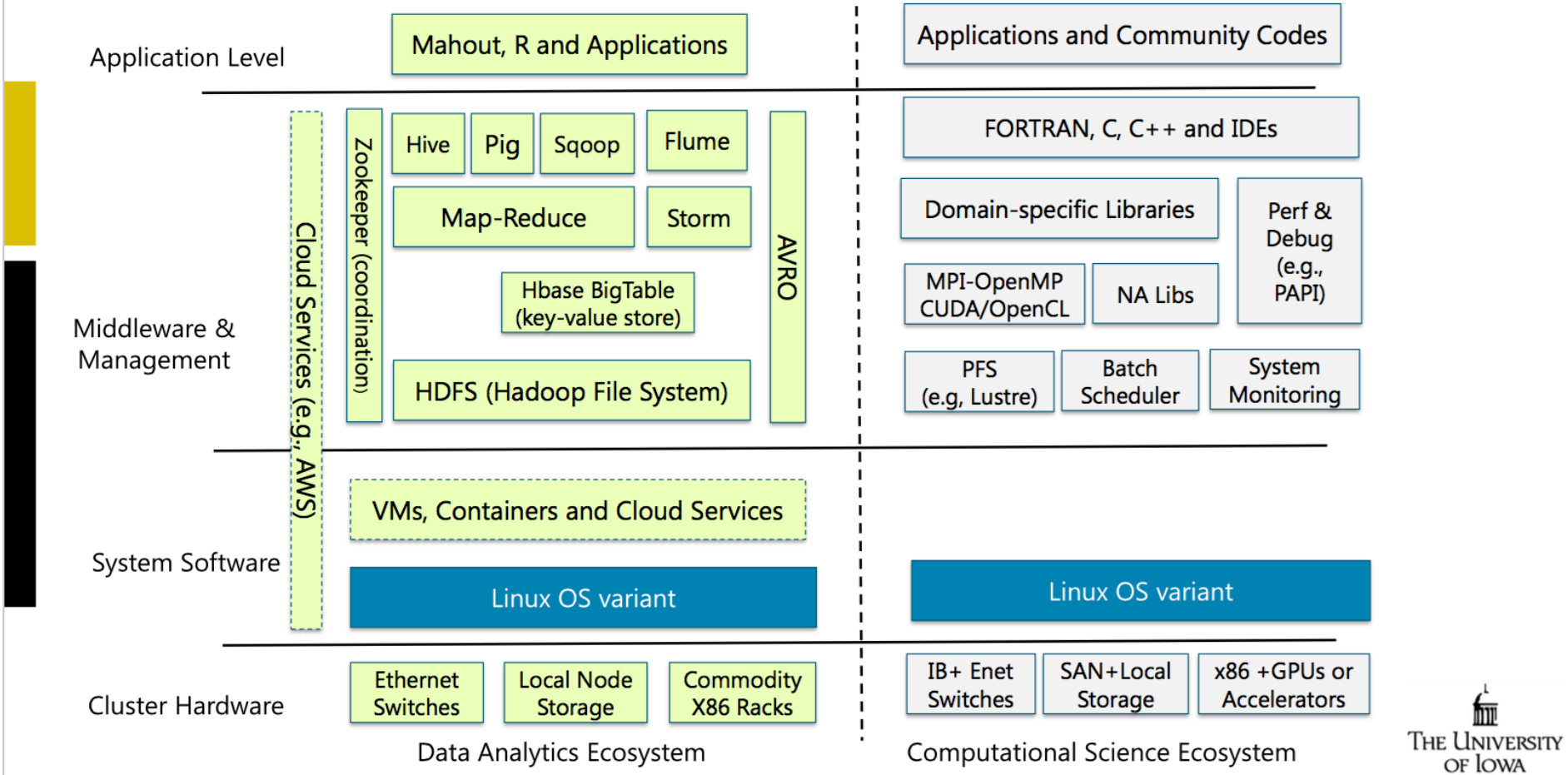
Overarching goal:

1. Create an international collaborative process focused on the co-design of software infrastructure for extreme scale science, addressing the challenges of both extreme scale computing and big data, and supporting a broad spectrum of major research domains,
2. Describe funding structures and strategies of public bodies with Exascale R&D goals worldwide
3. Establishing and maintaining a global network of expertise and funding bodies in the area of Exascale computing

BDEC Workshop, Charleston, SC, USA, April 29-May 1, 2013
BDEC Workshop, Fukuoka, Japan, February 26-28, 2014
BDEC Workshop, Barcelona, Spain, January 28-30, 2015
BDEC Workshop, Frankfurt, June 16-17, 2016

Credits: Jack Dongarra

Divergent ecosystems (Reed/Dongarra, CACM, July 2016)



Credits: Dan Reed

Stepping Stones: Towards EC/BD Convergence

Sharing the same resources

- Resource management methods need to evolve so that BD and EC can share resources

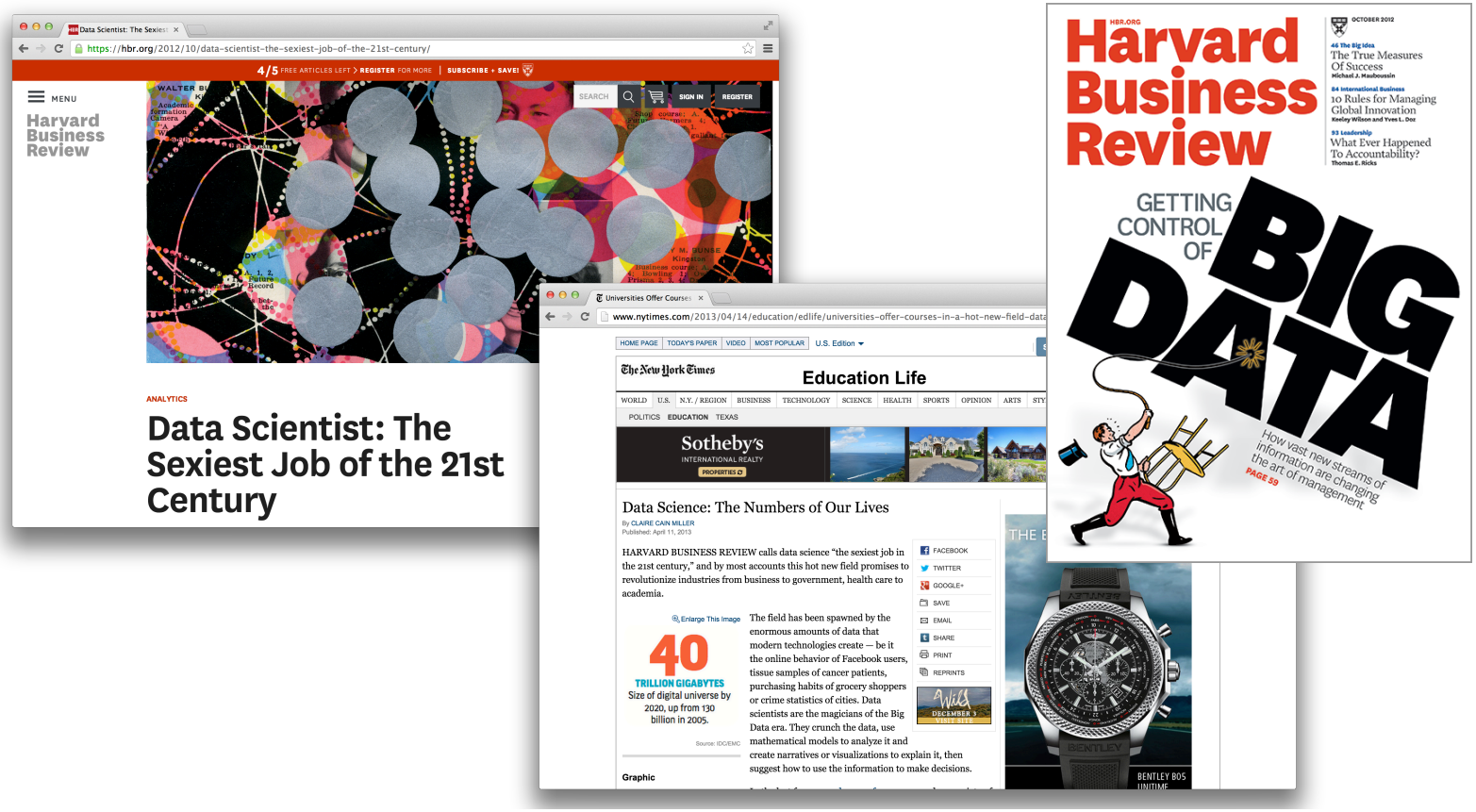
Convergence towards « stepping stones »

- Challenges and demonstrations
    - Software representing an entire system that can be used for BDEC

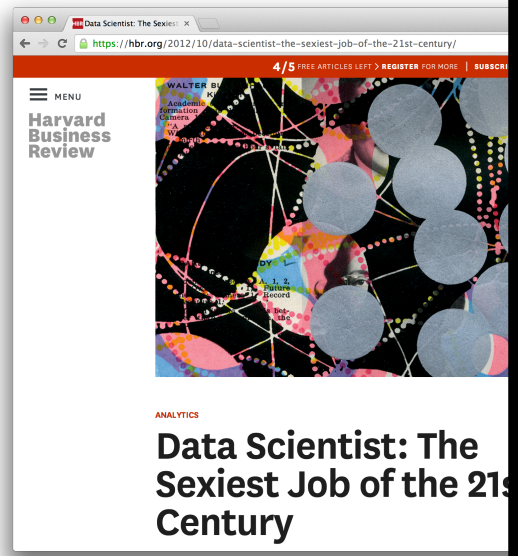
- HPC features available in the cloud (HPC)
- Cloud" features available on HPC platforms (availability, predictable response time, etc.)

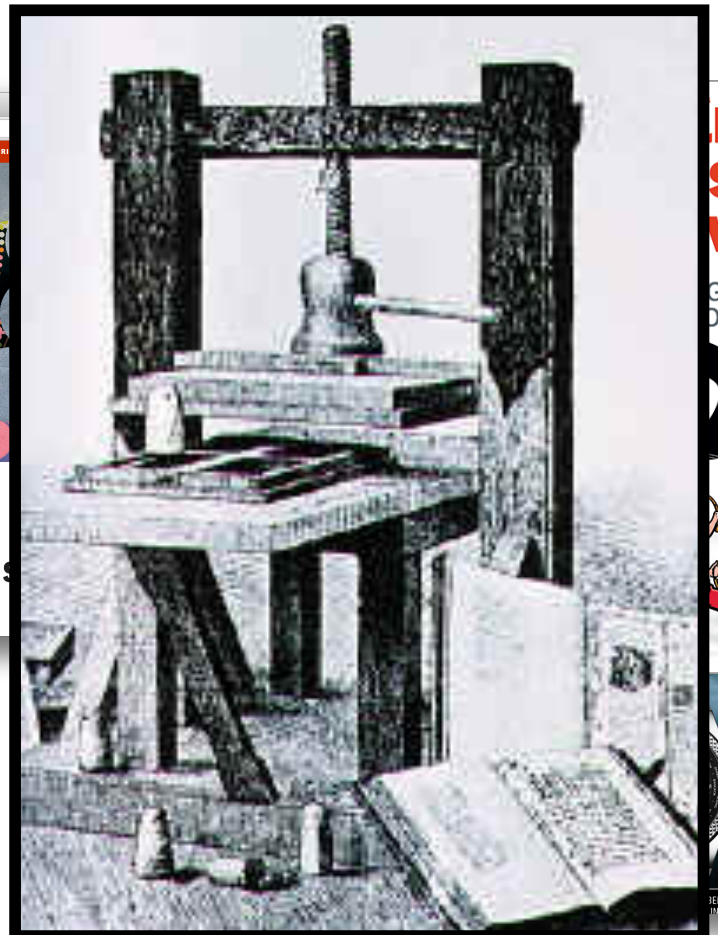
Credits: Kate Keahey

A Catalyst for Convergence: Data Science



A Catalyst for Convergence: Data Science



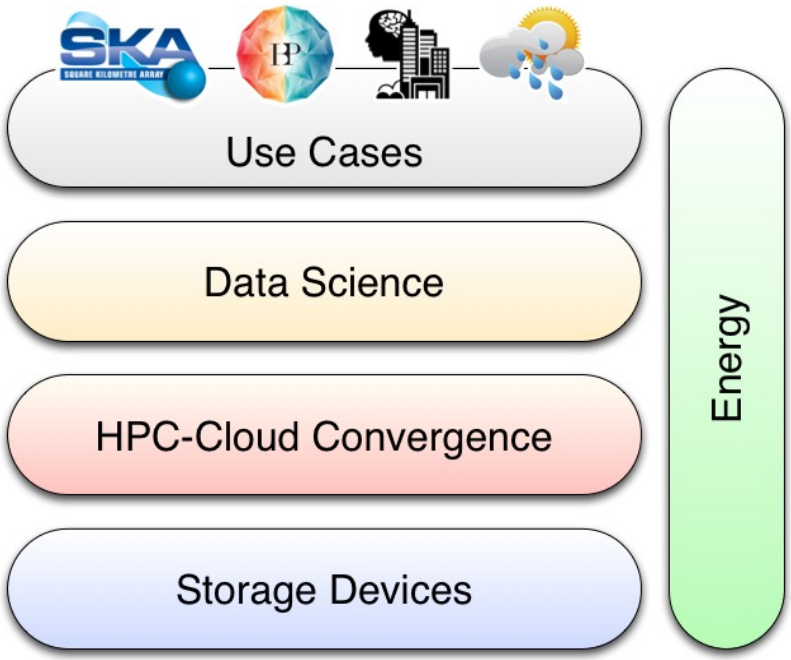


An Approach:
The BigStorage H2020 Project

Project overview

Data Science
- Modelling Big Data processing
- Energy-efficient analysis
- Data-driven decision making for Big Data applications

HPC-Cloud Convergence
- Applications
- Middleware, operating in the cloud and HPC environments
- Infrastructure for Storage and Computing



Storage Devices
- Storage acceleration
- Storage convergence
- Storage isolation

Energy
- Compression or de-duplication for storage footprint reduction
- Hints from application to storage system, enabling energy consumption reduction

BigStorage

The BigStorage Consortium

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Inria
INVENTEURS DU MONDE NUMÉRIQUE

POLITÉCNICA

Seagate

DKRZ
DEUTSCHES KLIMARECHENZENTRUM

FORTH
Foundation for Research & Technology, Hellas

cea

FUJITSU

ca

IBM

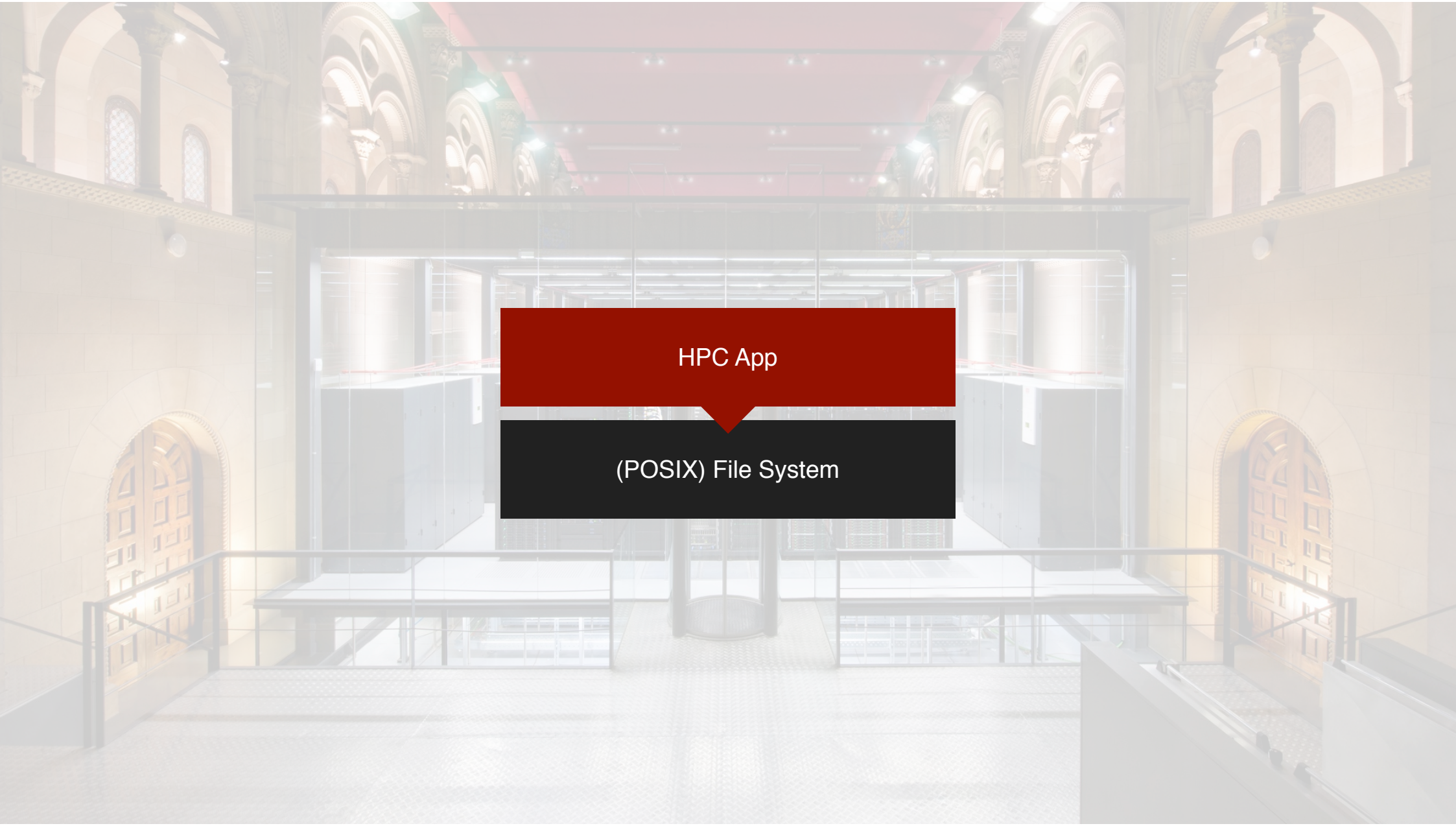
BULL

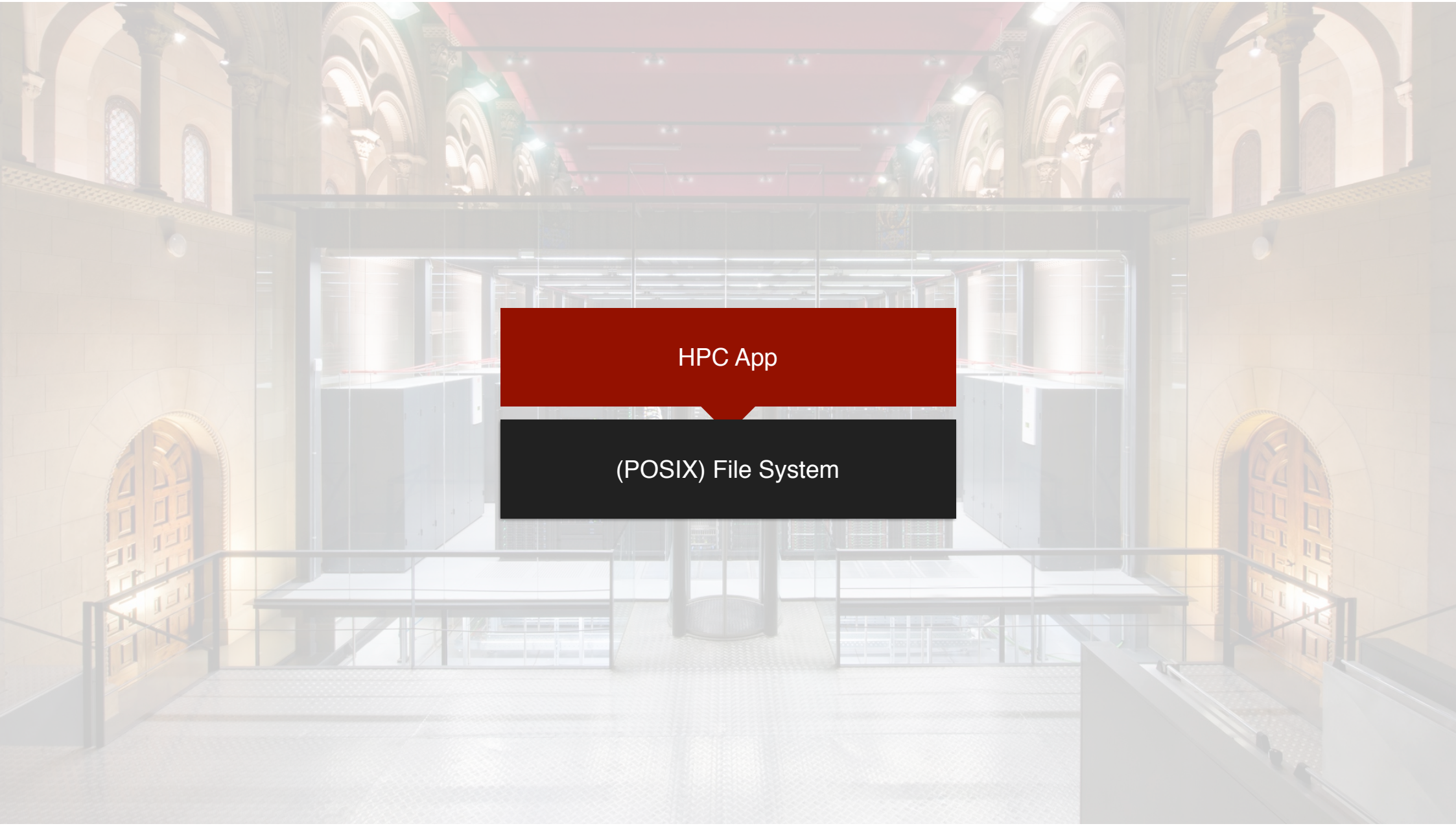
INFORMÁTICA El Corte Inglés

BigStorage

One question

HPC App


HPC App
(POSIX) File System


HPC App
(POSIX) File System

Supports

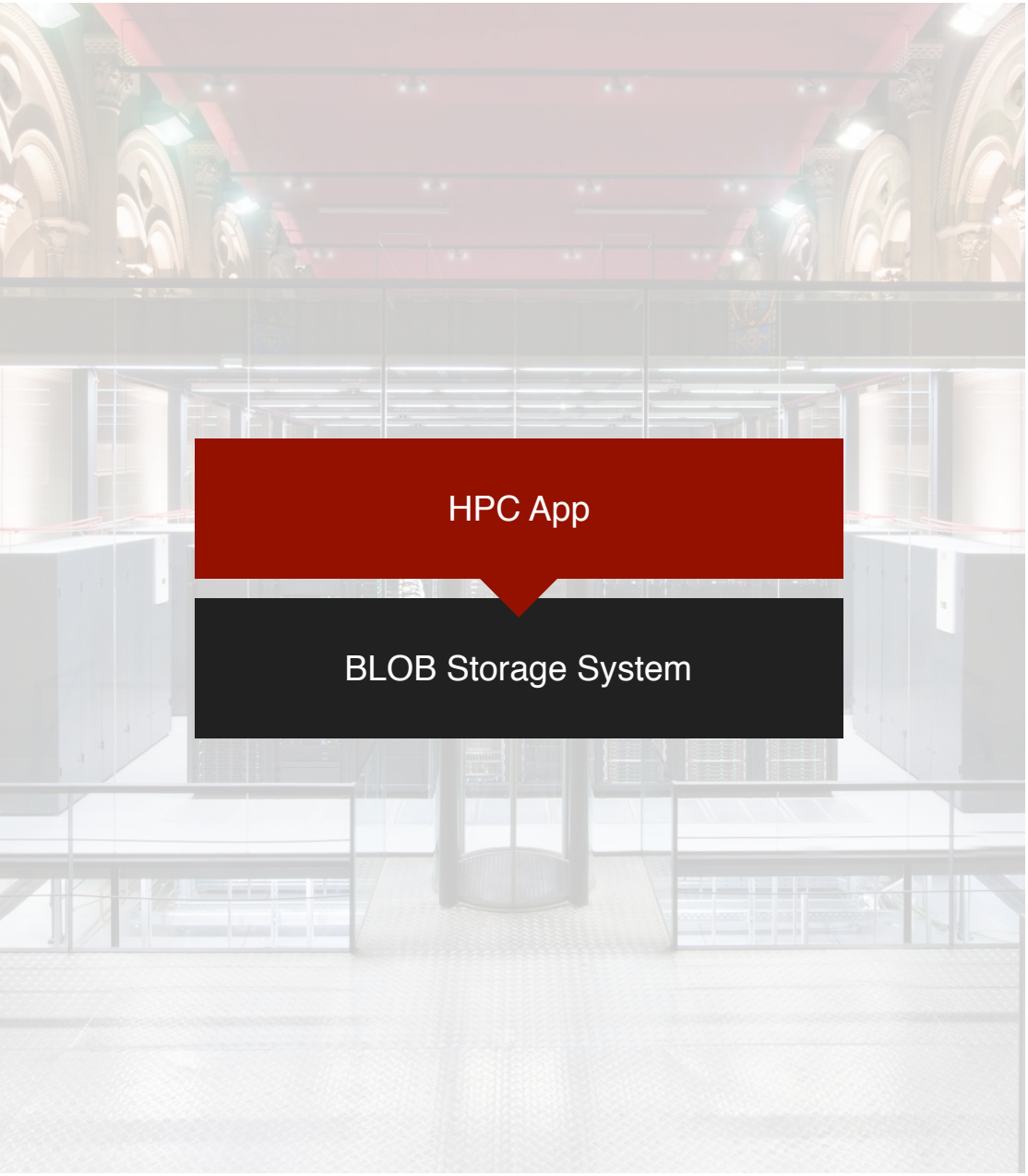
- folder / file hierarchies
- permissions
- random reads and writes to files
- atomic file renaming
- multi-user protection

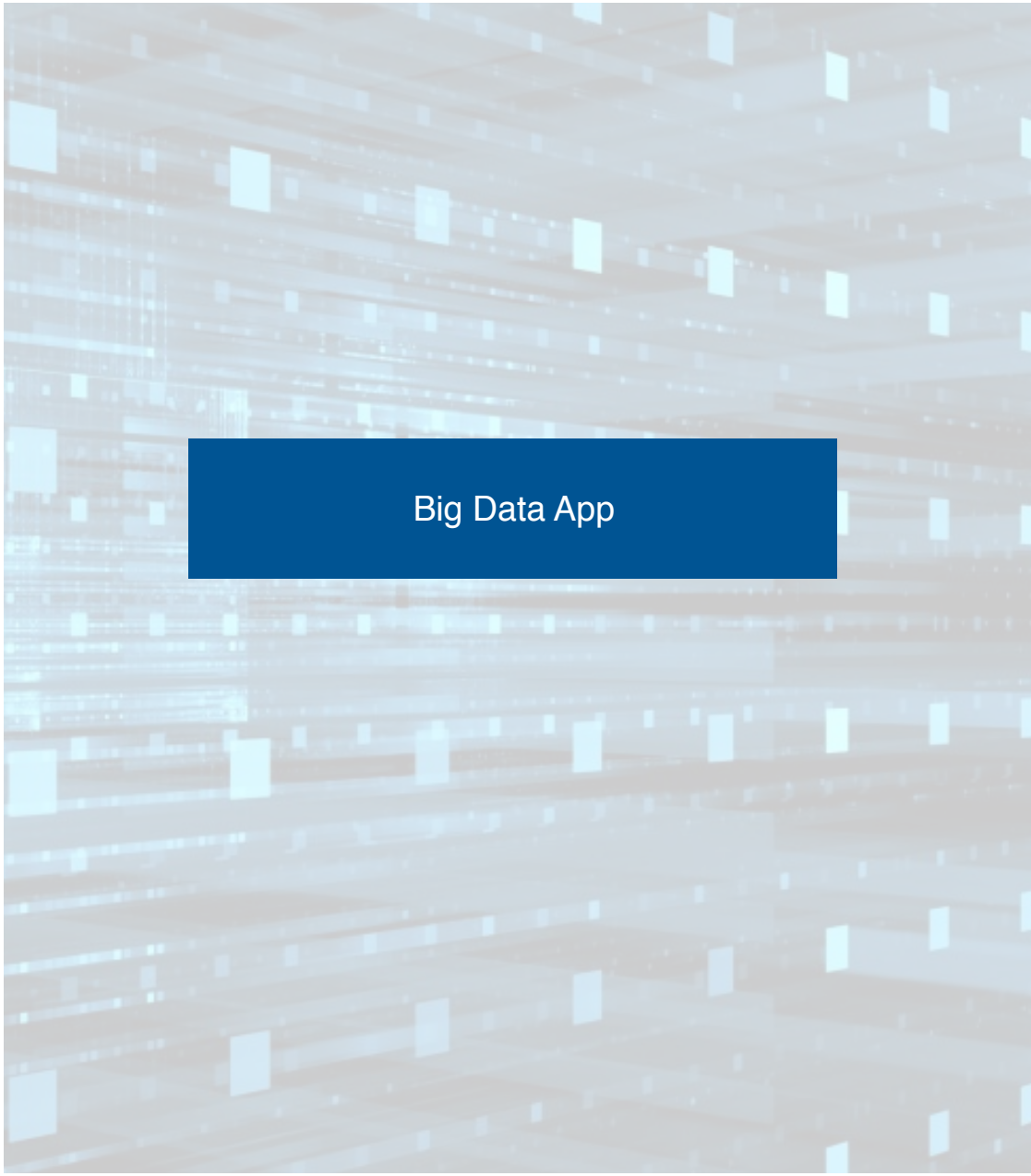
Supports random reads and writes to files - BLOBs

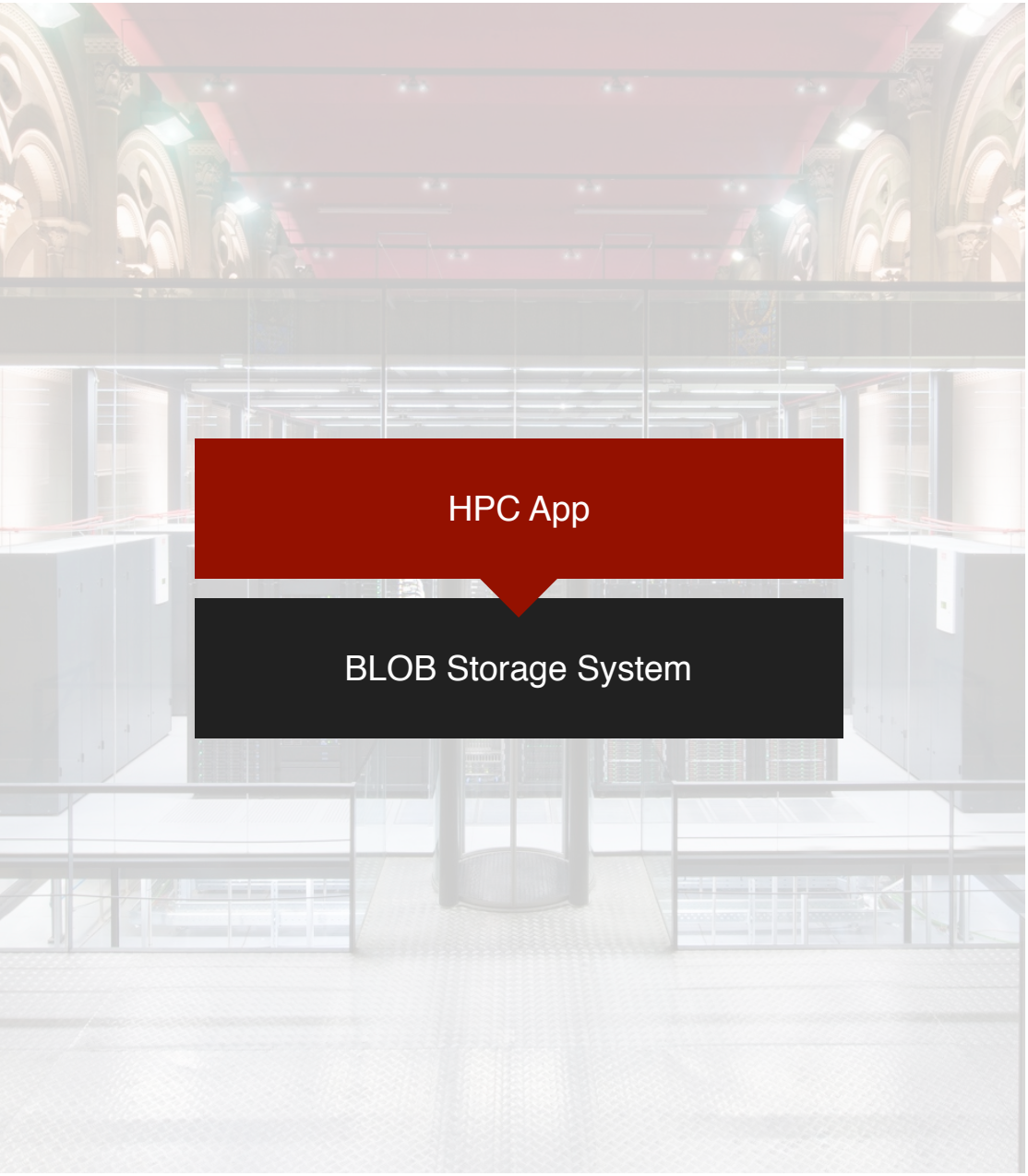
(Binary Large OBjects)

BLOB Storage System

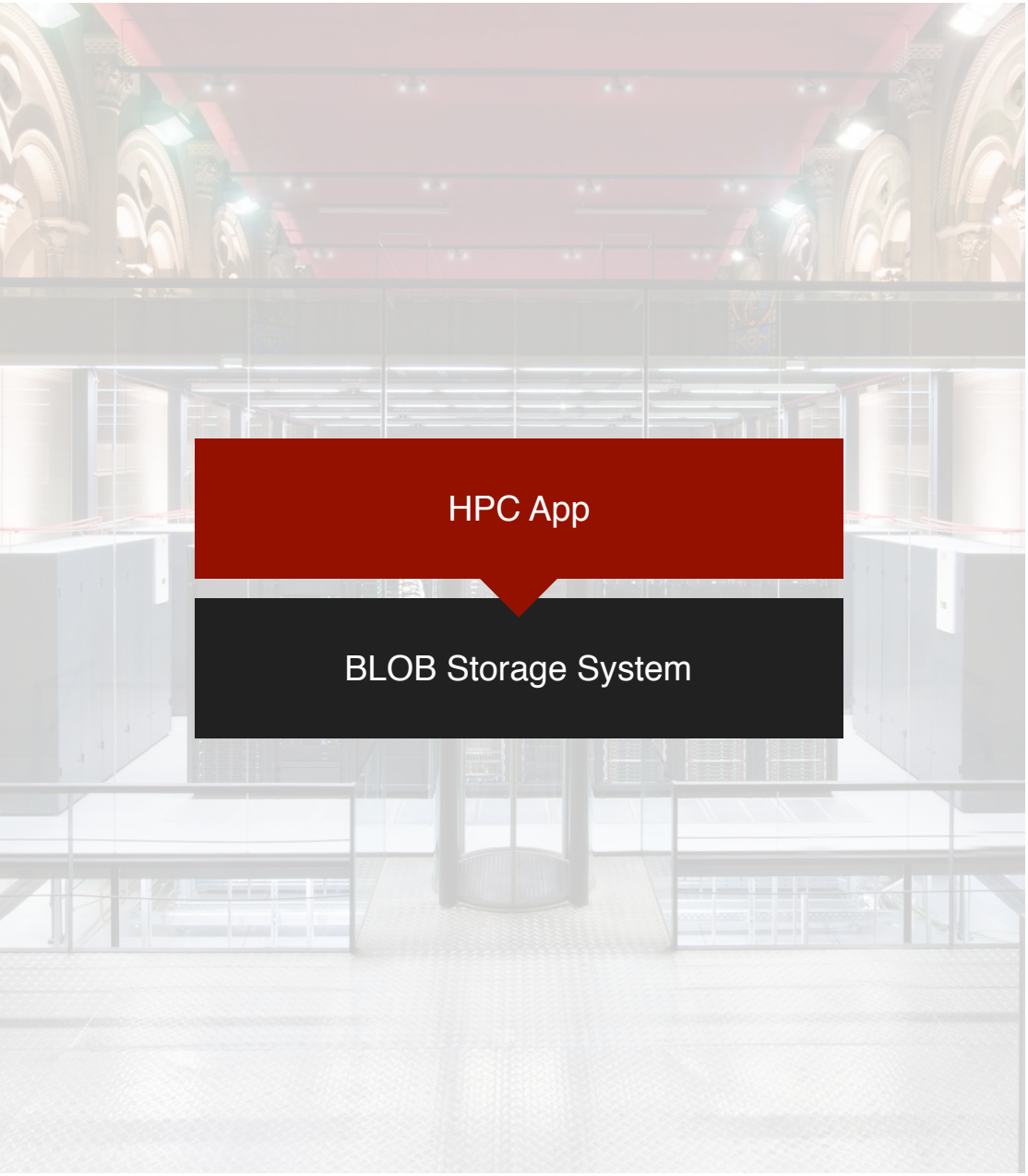

HPC App
BLOB Storage System

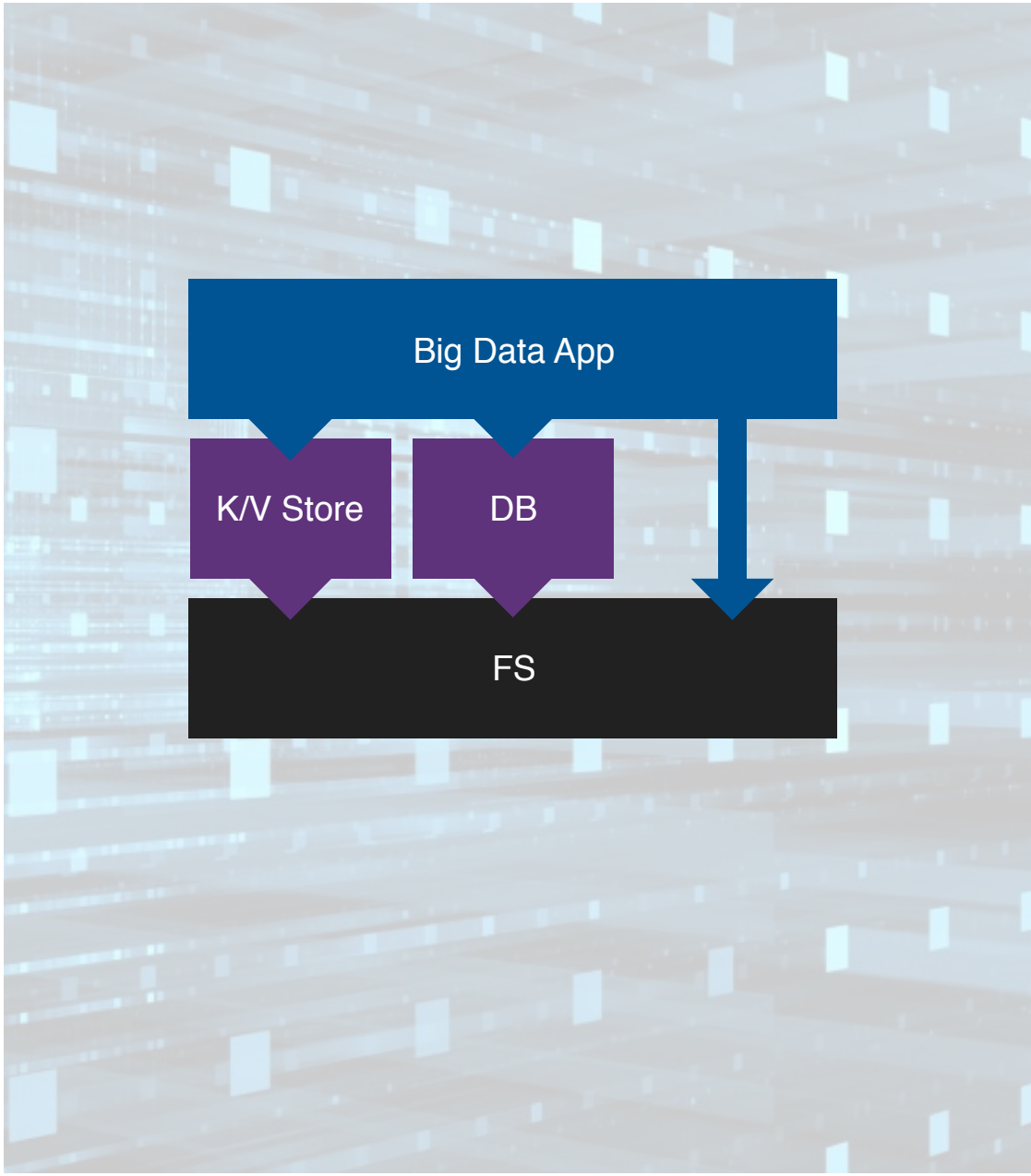

HPC App
BLOB Storage System

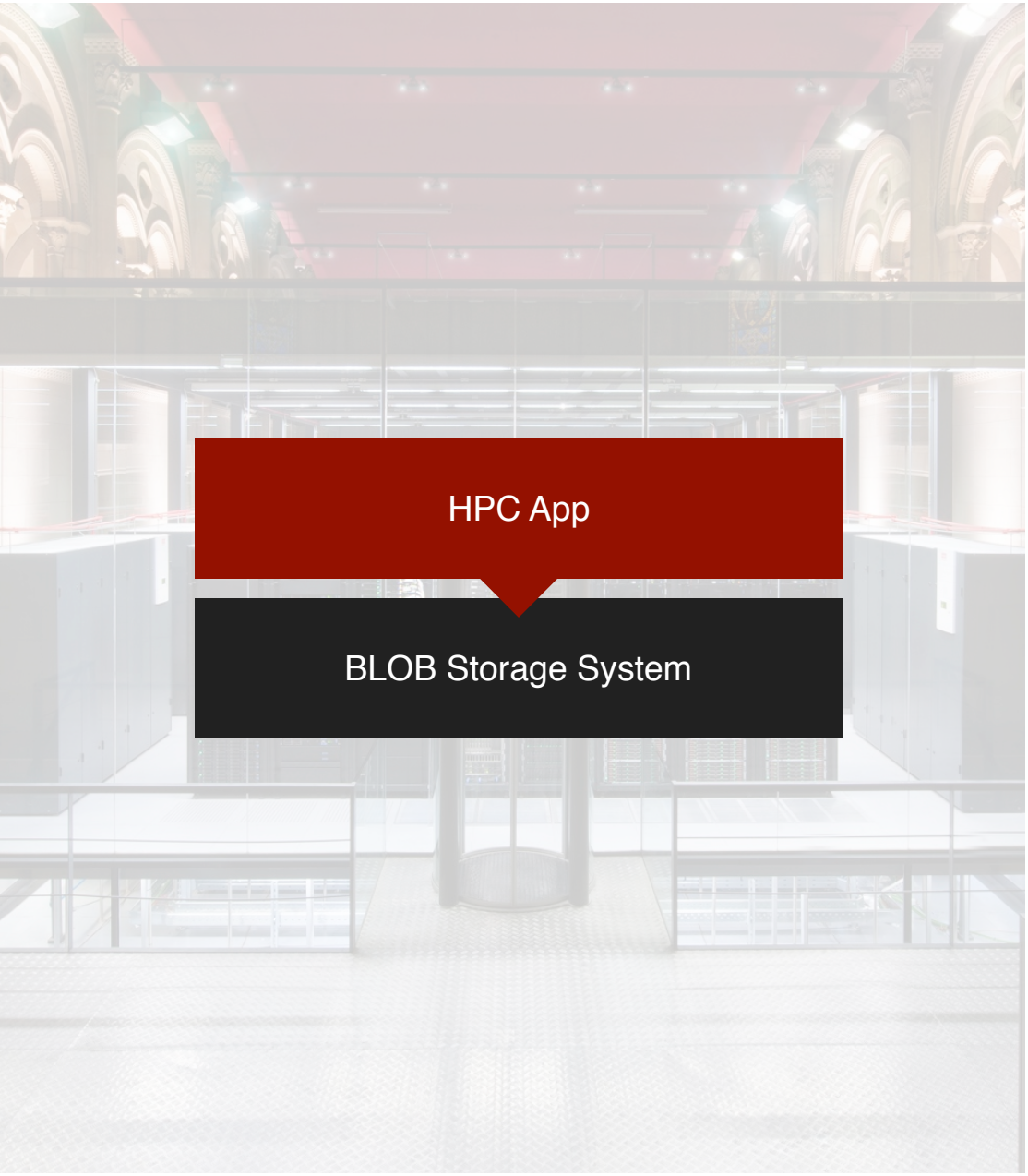

Big Data App

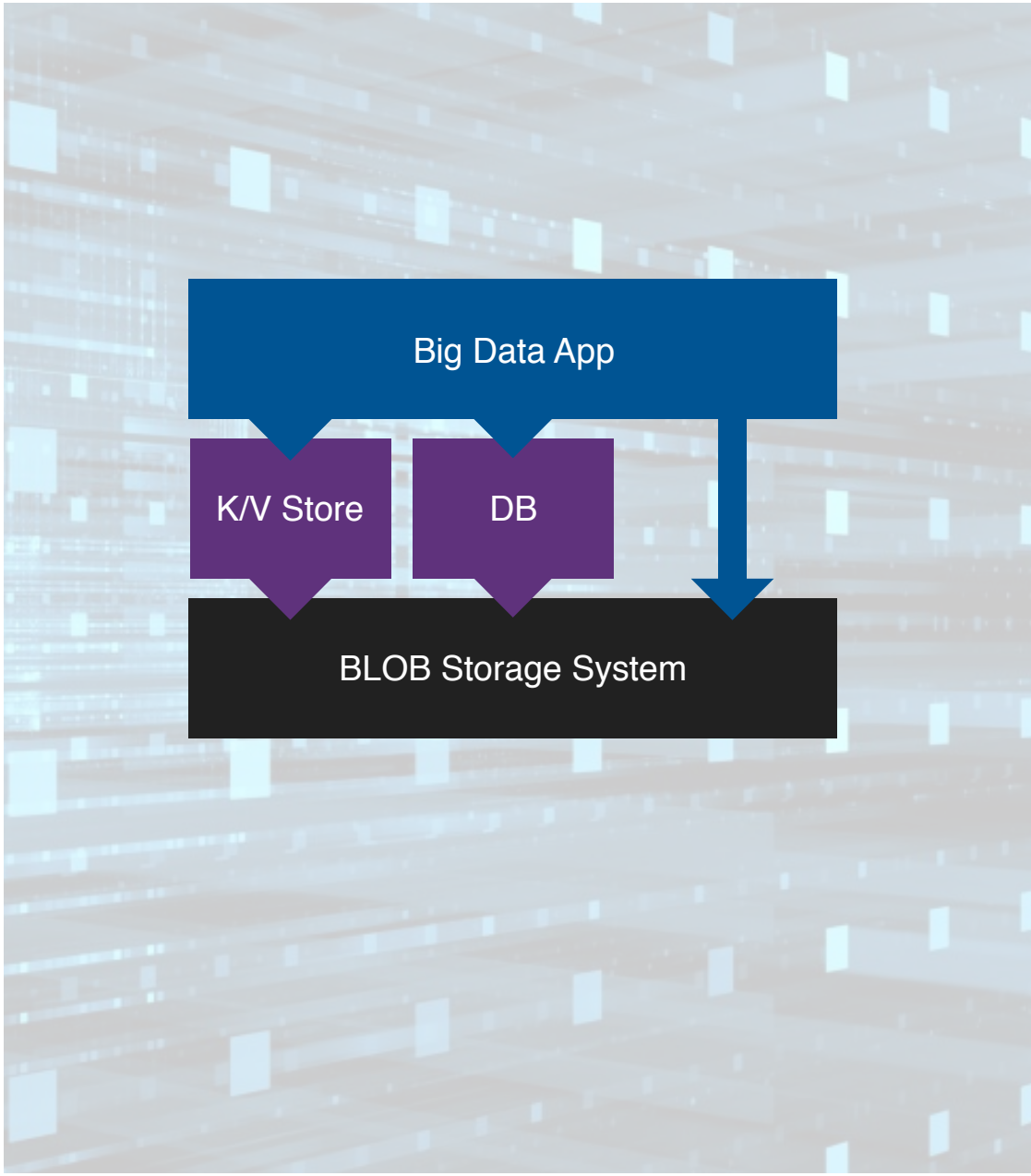

HPC App
BLOB Storage System


Big Data App
K/V Store
DB
FS


HPC App
BLOB Storage System


Big Data App
K/V Store
DB
FS


HPC App
BLOB Storage System


Big Data App
K/V Store
DB
BLOB Storage System


Big Data App
HPC App
K/V Store
DB
Converged BLOB Storage System

One use-case

One problem…

A scientific monitoring service, monitoring the ALICE CERN LHC experiment:

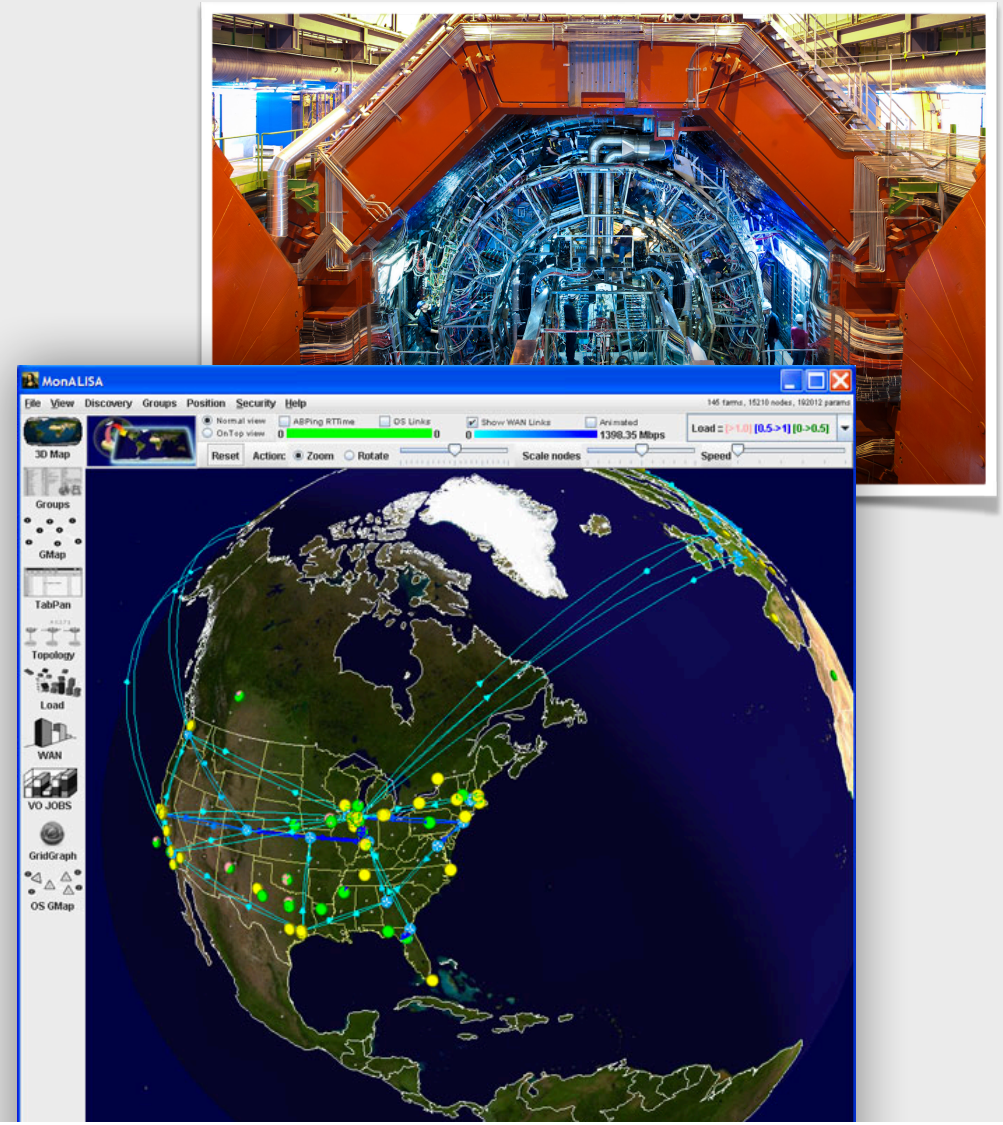
- Ingests events at a rate of up to 13 GB/s,
- Produces more than 10^9 data files per year

Computes 35.000+ aggregates in real-time

Current lock-based platform does not scale

…multiple requirements

- Multi-object write synchronization support
- Atomic, lock-free writes
- High-performance reads
- Horizontal scalability



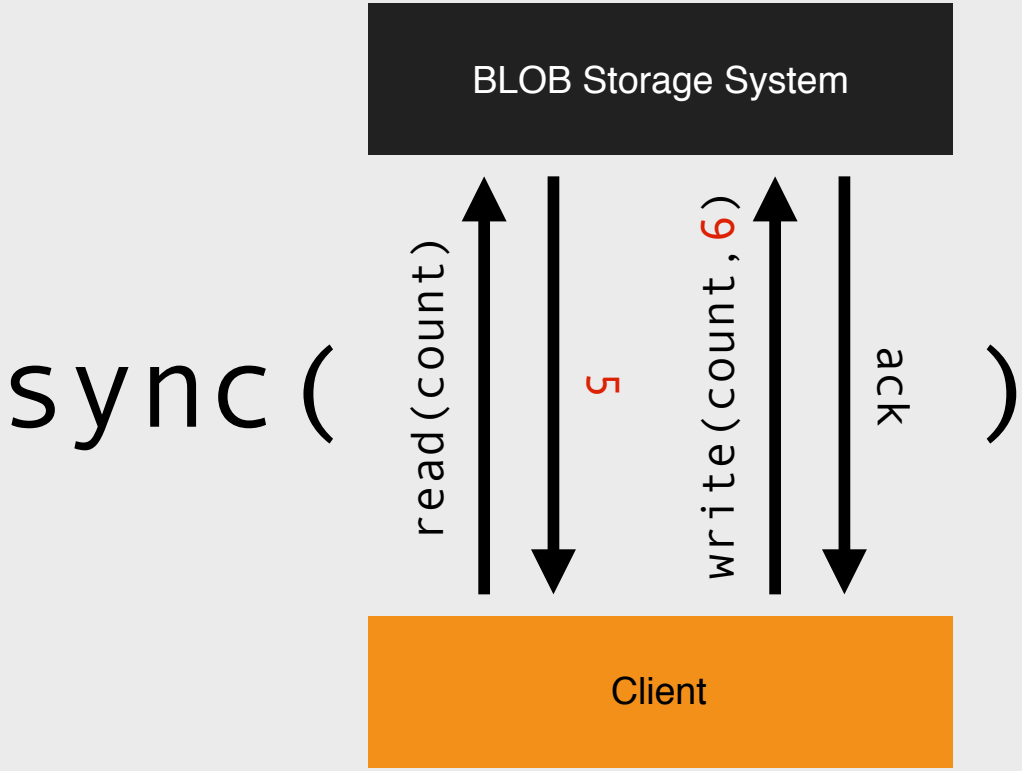
Why is write synchronization needed?

Aggregate computation is a 2-step operation:

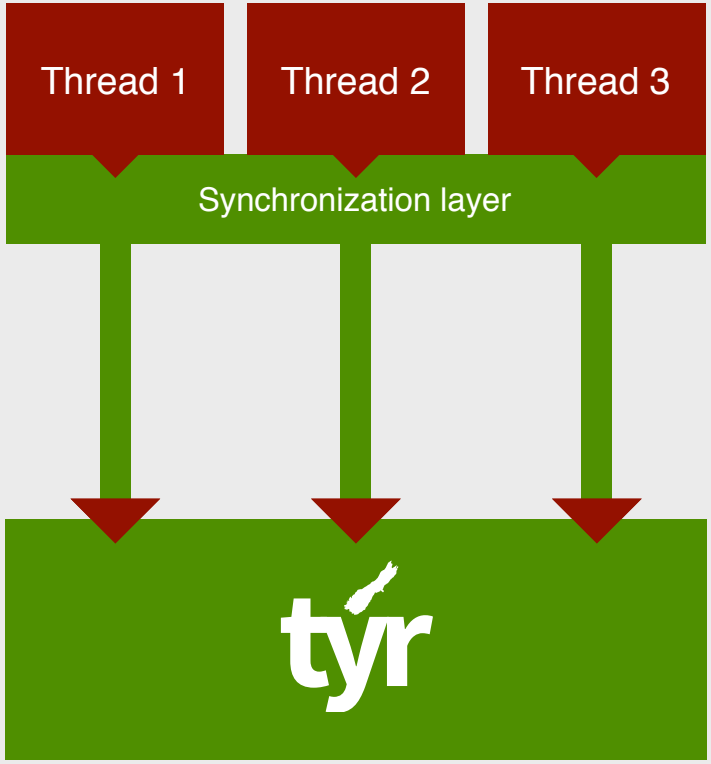
1. Read current value remotely from storage
2. Write the updated value remotely to storage

Aggregate update needs to be atomic.

Also, adding a new data to persistent storage and updating the related aggregates needs to be performed atomically as well.



At which level to handle concurrency management?



At the application level?

Enables fine-grained synchronization (app knowledge)
…but significantly complexities application design,
and typically only guarantees isolation.

At a middleware level?

Eases application design…
…but has a performance cost (zero knowledge),
and usually also only guarantees isolation.

At a storage level?

Also eases application design,
better performance than middleware (storage knowledge),
and may offer additional consistency guarantees.

Aren’t existing transactional object stores enough?

Not quite.

Existing transactional systems typically only ensure consistency of *writes*

In most current systems, *reads* are performed atomically only because objects are small enough to be located on a single server, *i.e.*

- Records for database systems
- Values for Key-Value stores

Yet, for large objects, reads spanning multiple chunks should always return a consistent view

Týr transactional design

Týr internally maps all writes to transactions

- Multi-chunk, and even multi-blob operations are processed with a serializable order
- Ensures that all chunk replicas are consistent

Týr uses a high-performance, sequentially-consistent transaction chain algorithm: WARP [1].

[1] R. Escriva et al. – Warp: Lightweight Multi-Key Transactions for Key-Value Stores

Týr is alive!

Fully implemented as a prototype with ~22.000 lines of C

Lock-free, queue-free, asynchronous design.

Leveraging well-known technologies:

- Google LevelDB [1] for node-local persistent storage,
- Google FlatBuffers [2] for message serialization,
- UDT [3] as network transfer protocol.

[1] http://leveldb.org/
[2] https://google.github.io/flatbuffers
[3] http://udt.sourceforge.net/

Týr evaluation with MonALISA

MonALISA data collection was re-implemented atop Týr, and evaluated using real data

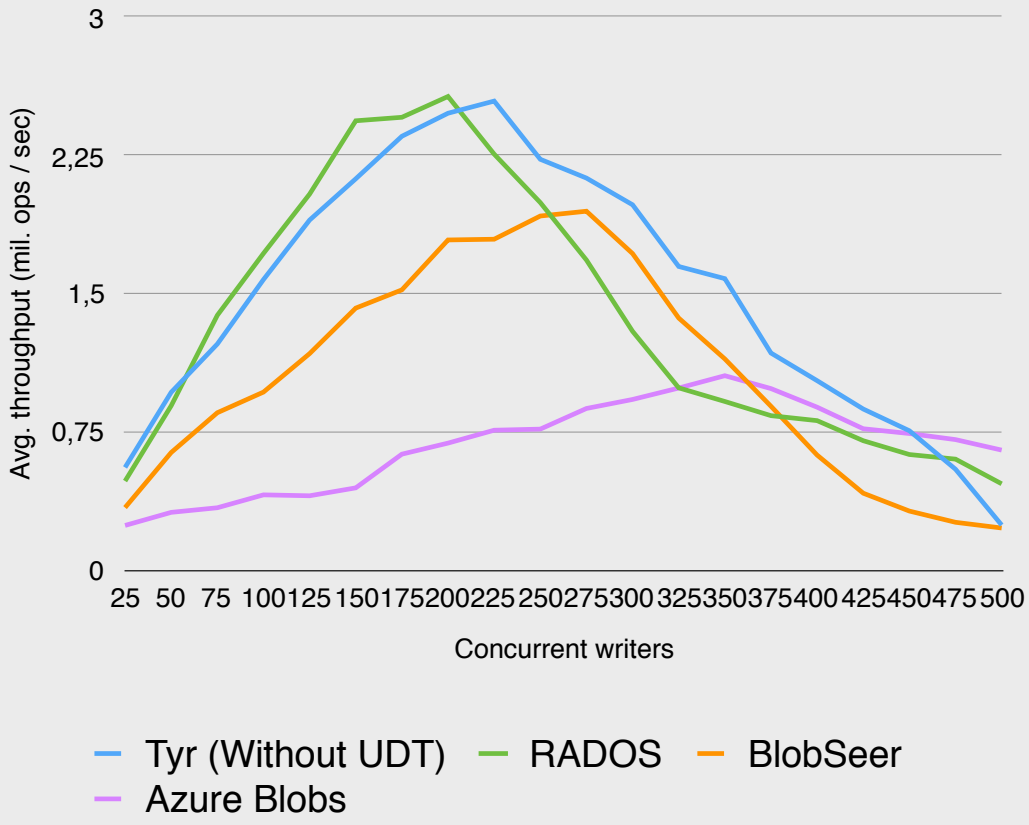
Týr was compared to other state-of-the-art, blob-based storage systems:

- RADOS / librados (Ceph)
- Azure Storage Blobs (Microsoft)
- BlobSeer (Inria)

Experiments run on the Microsoft Azure cloud, up to 256 nodes

3 x replication factor for all systems

Non-synchronized write performance:
Evaluating the cost of the transaction protocol



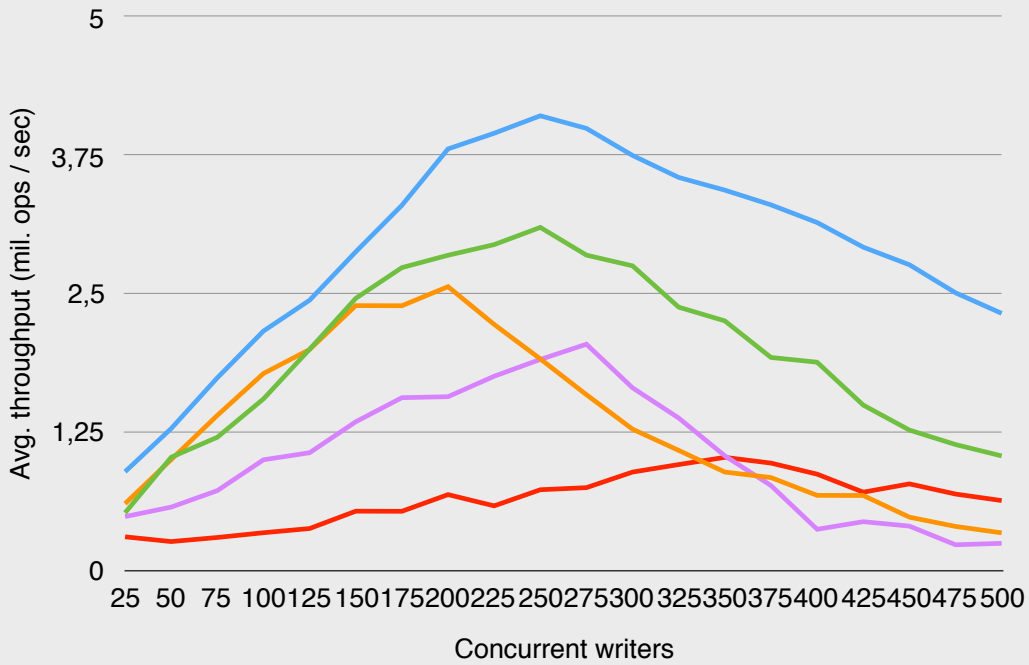
MonALISA write performance, varying the number of concurrent writers

In that experiment, no system other than Týr is synchronized

- 32 nodes,
- Average of 50 x 10-minute runs

Týr shows performance similar to RADOS, despite its significantly stronger consistency guarantees

Synchronized write performance:
Evaluating transactional write performance



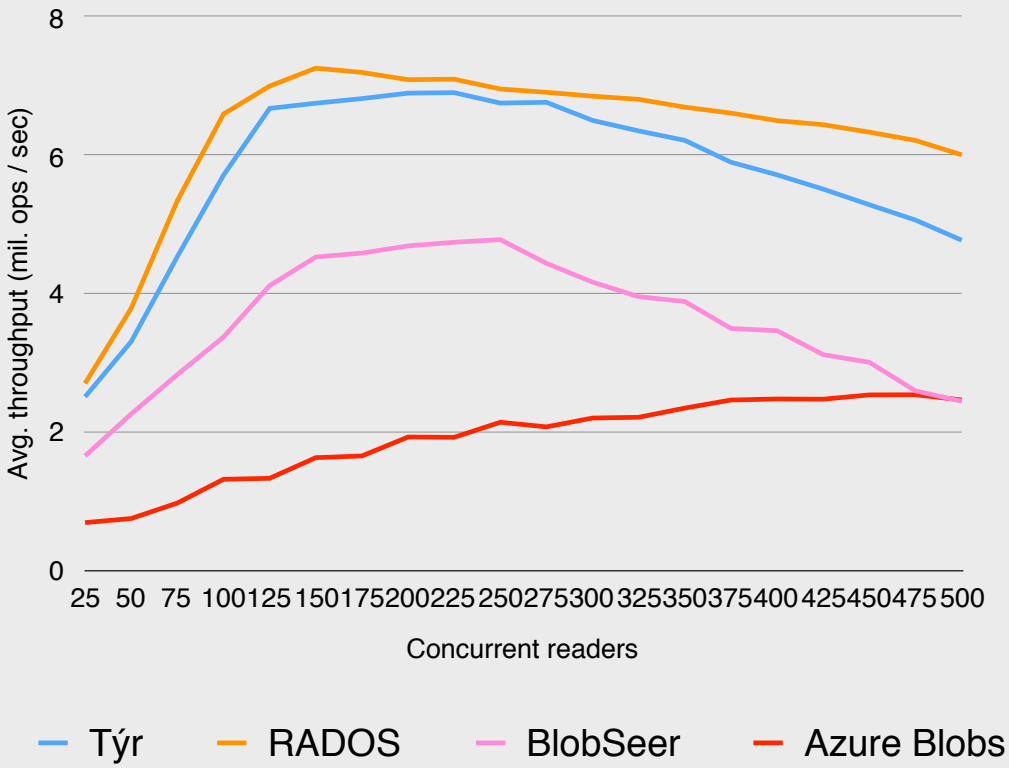
- Týr (Atomic operations)
- RADOS (Synchronized)
- Azure Blobs (Synchronized)
- Tyr (Read-Update-Write)
- BlobSeer (Synchronized)

We add fine-grained, application-level, lock-based synchronization to Týr competitors

Performance of Týr competitors decrease due to the synchronization cost

Clear advantage of Atomic operations over Read-Update-Write aggregate updates

Read performance

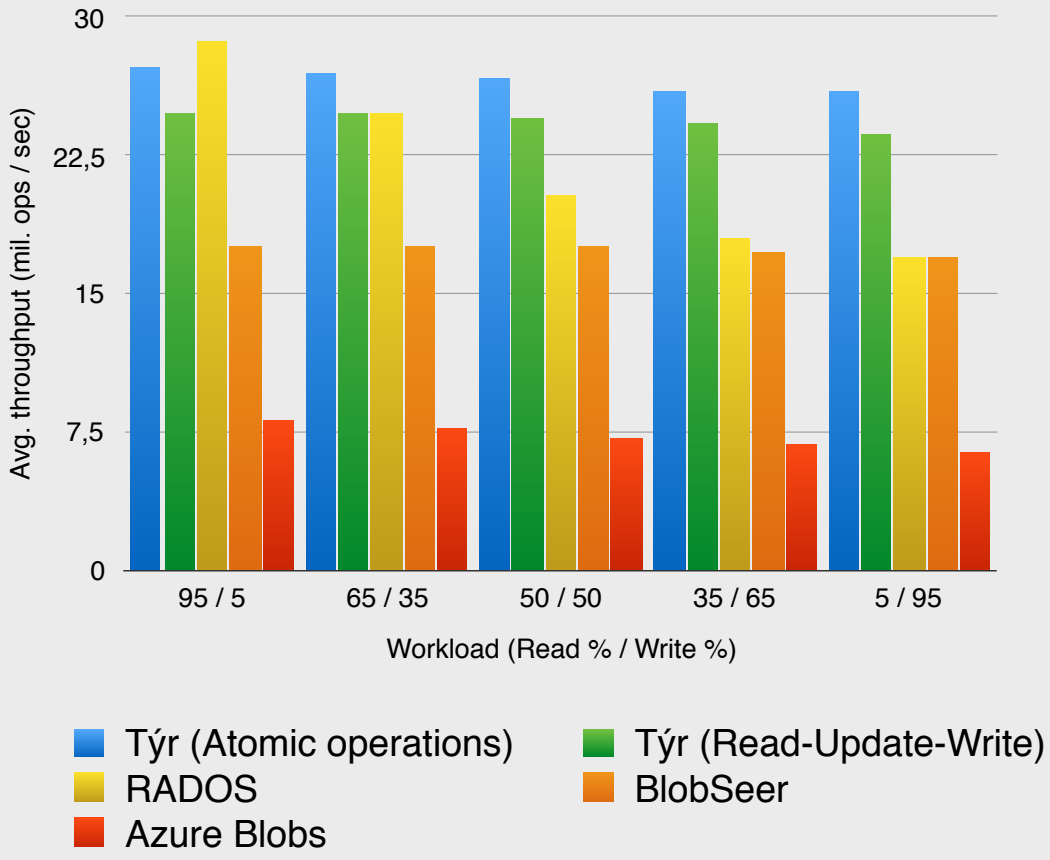


We simulate MonALISA reads, varying the number of concurrent readers

Slightly lower performance than RADOS, but offers read consistency guarantees

Týr lightweight read protocol allows it to outperform BlobSeer and Azure Storage

Combined Read-Write performance / Reader-Writer Isolation

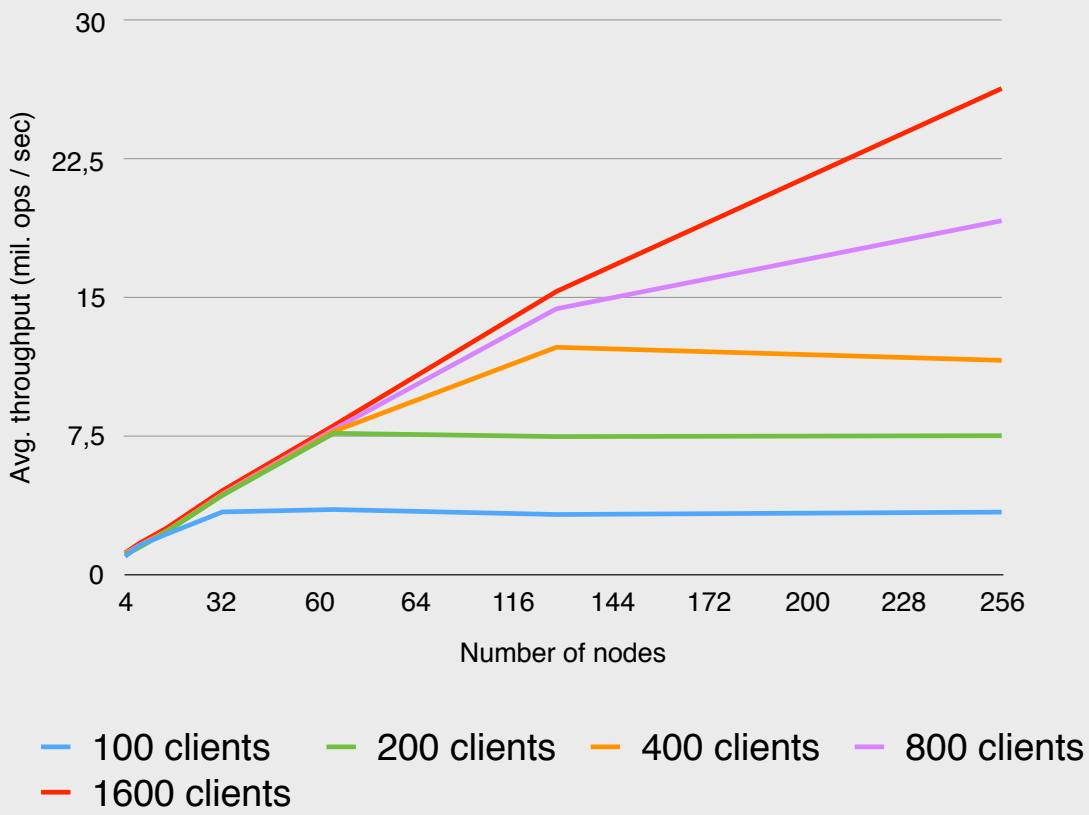


We vary the ratio of read to writes, and measure the read throughput

Týr and BlobSeer show little interference between readers and writers

RADOS and Azure Storage, in contrast, show degraded read performance as the ratio of writers to readers increase.

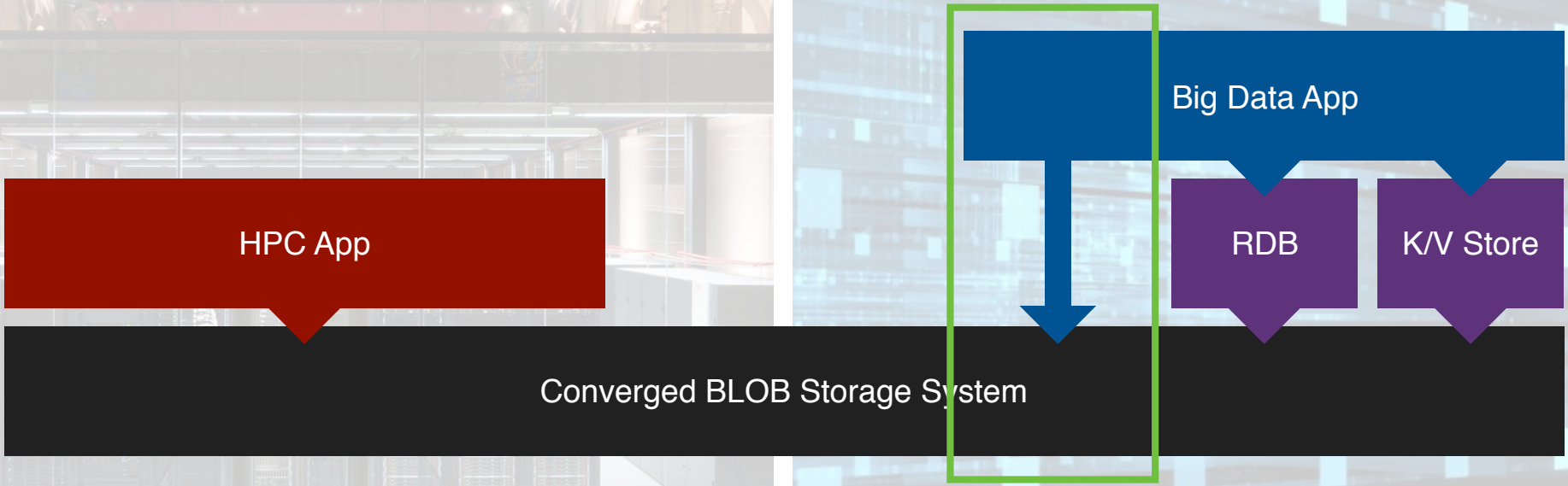
Horizontal scalability



We measure the average read / write throughput of Týr under a typical 65% read / 35% write MonALISA workload

Týr scales almost linearly with the number of nodes in the cluster

And Now?



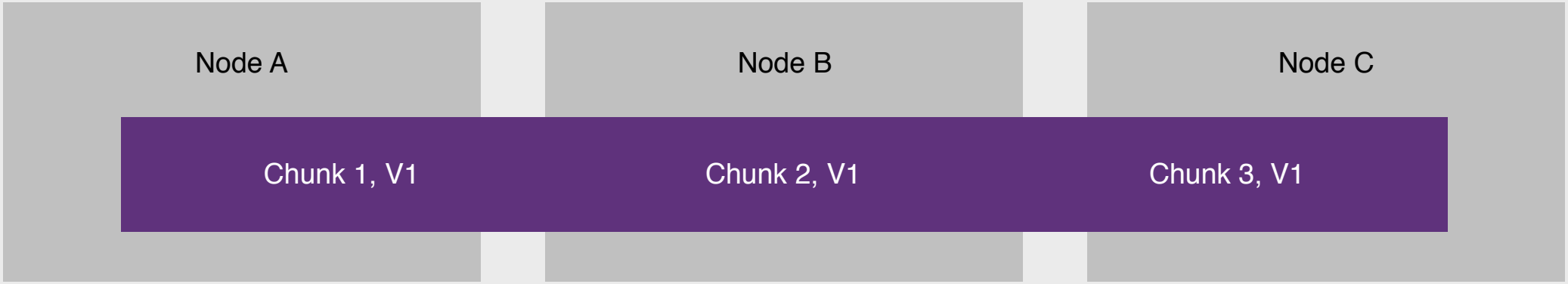
Týr for HPC applications?

Týr as a base layer for higher-level storage abstractions?

Thank you!

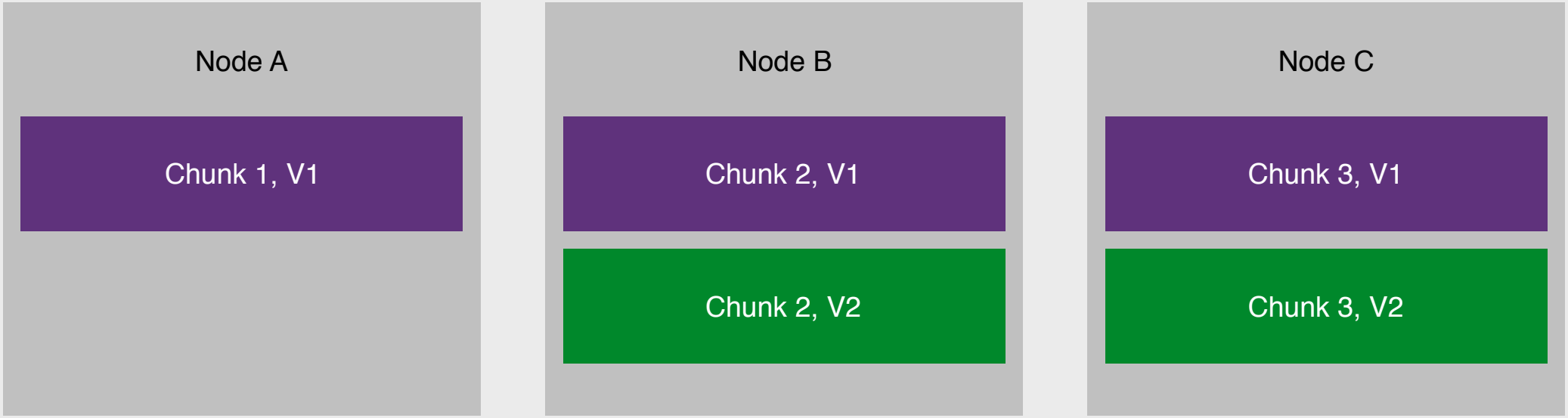
P. MATRI, A. COSTAN, G. ANTONIU, J. MONTES, M. S. PÉREZ, Tyr: Blob Storage Meets Built-In Transactions,
in : SC16 - The International Conference for High Performance Computing, Networking, Storage and Analysis 2016, Salt Lake City, November 2016.
Best Student Paper Award Finalist.

Týr version management

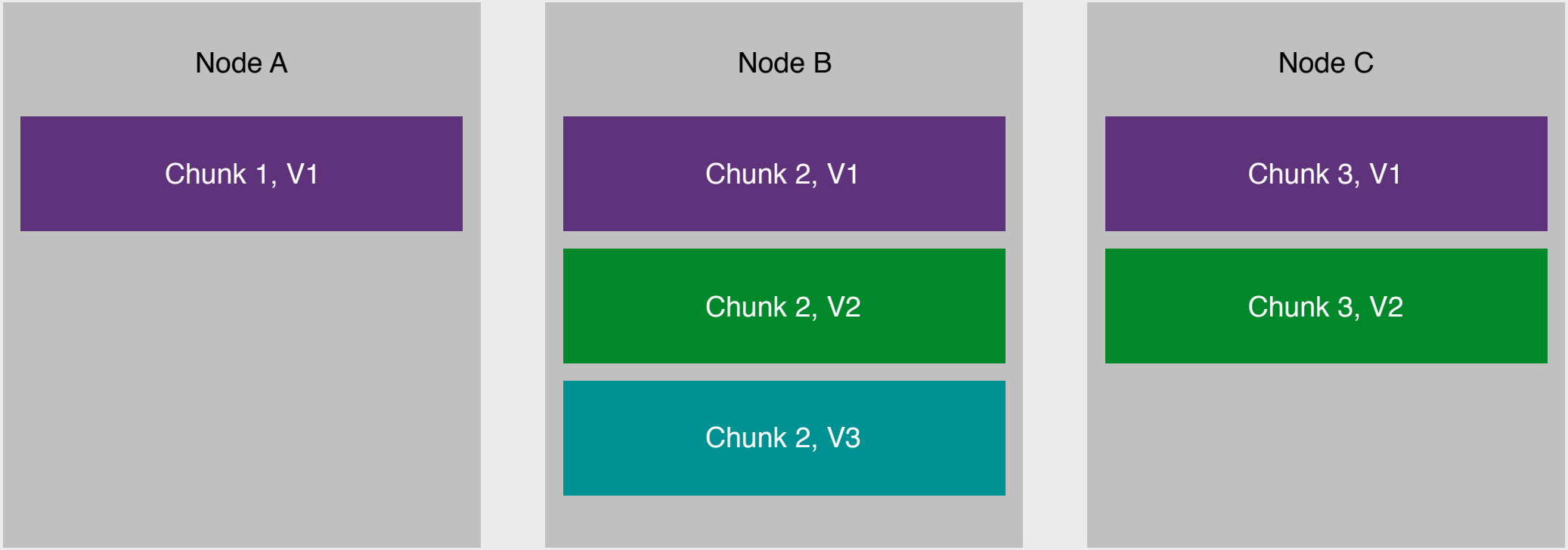


`hash({blob, chunk}) ➔ node`

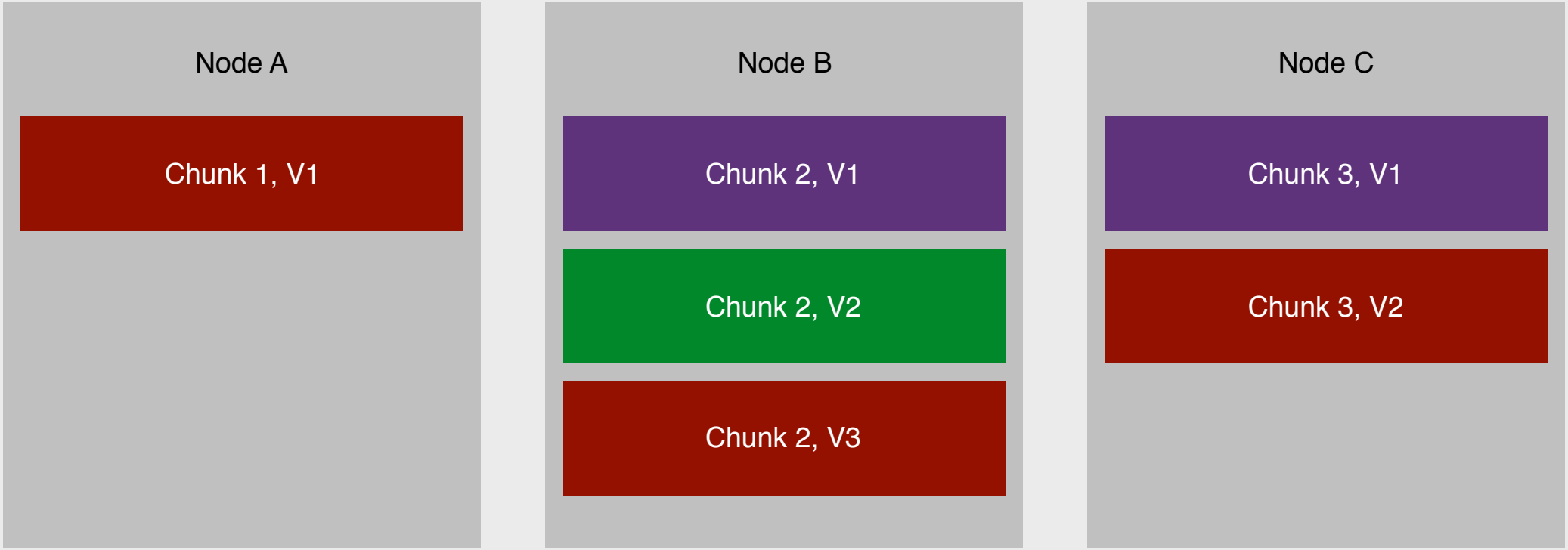
Týr version management



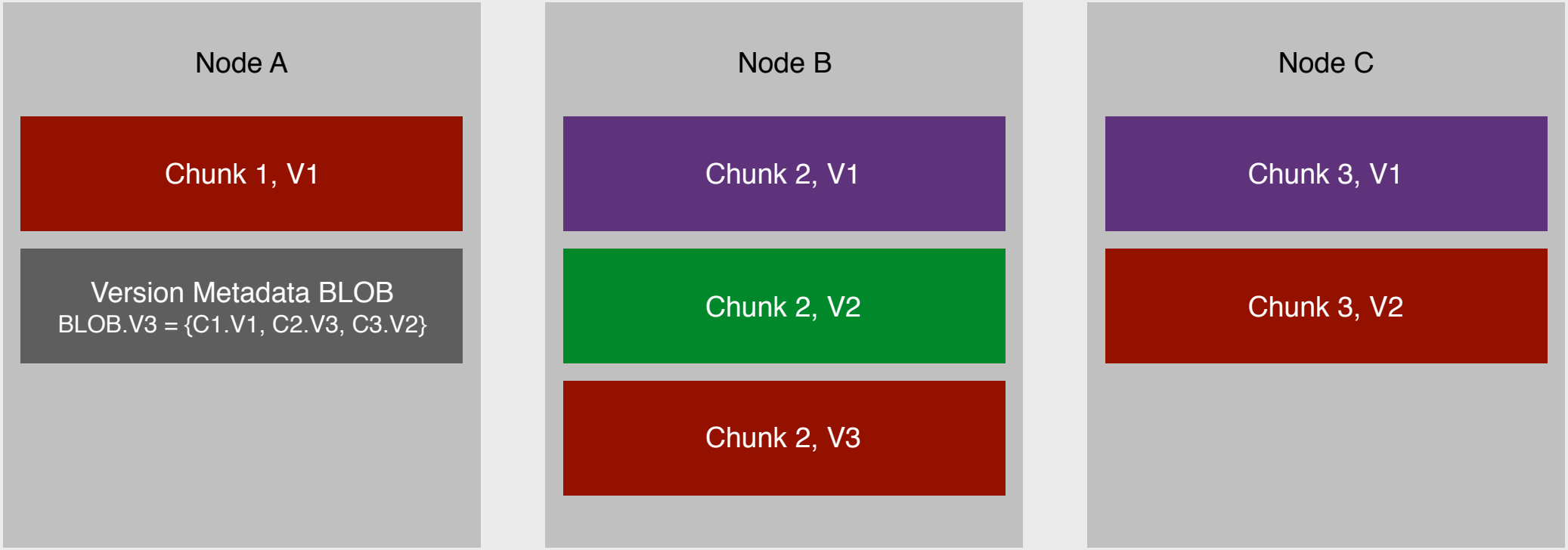
Týr version management



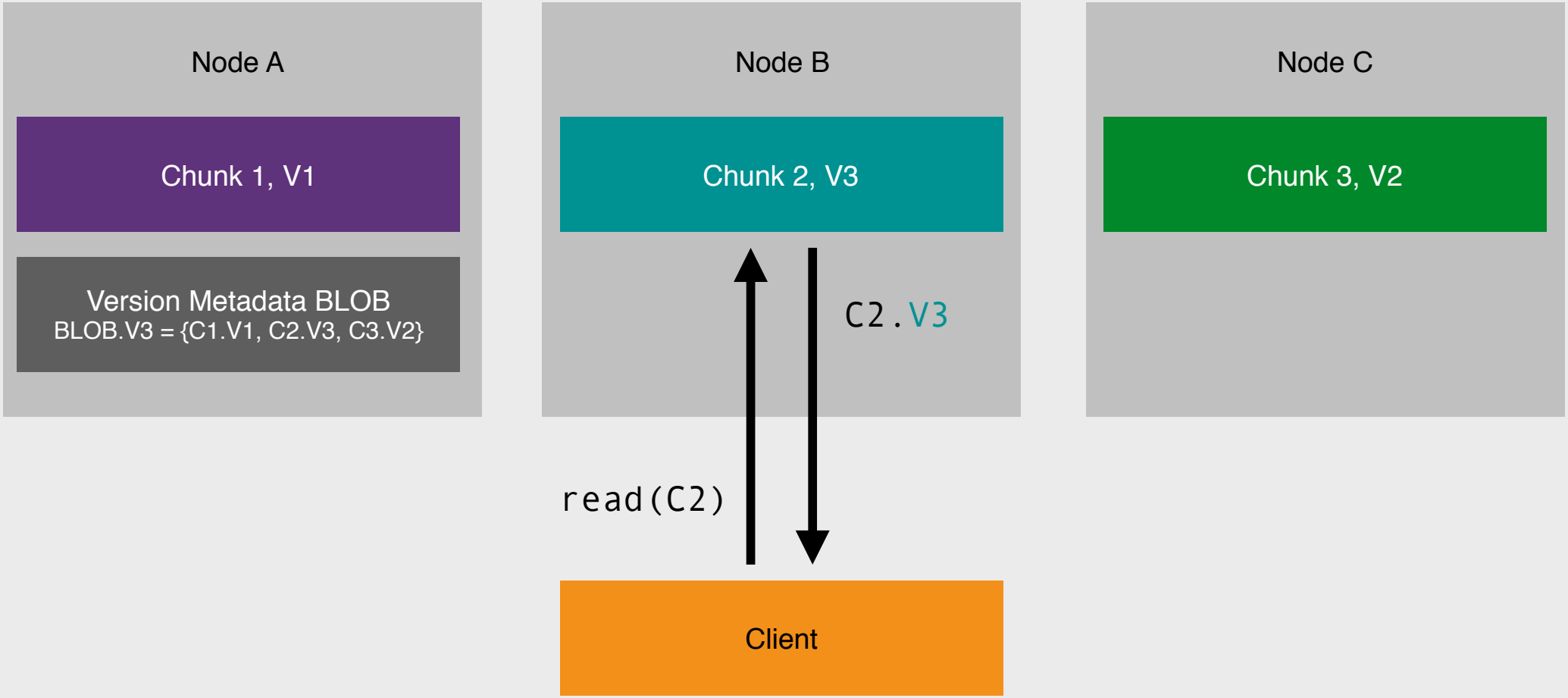
Týr version management

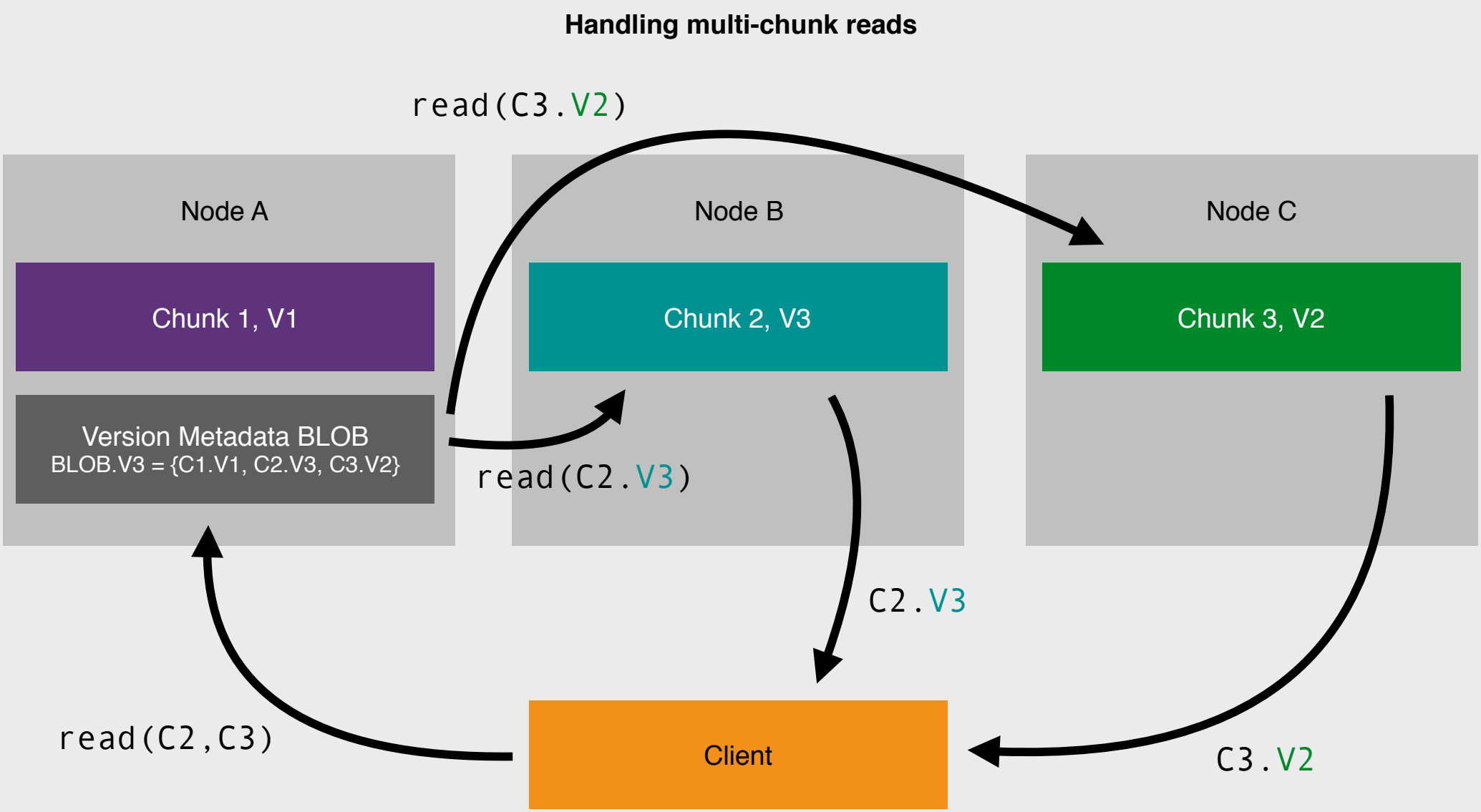


Týr version management

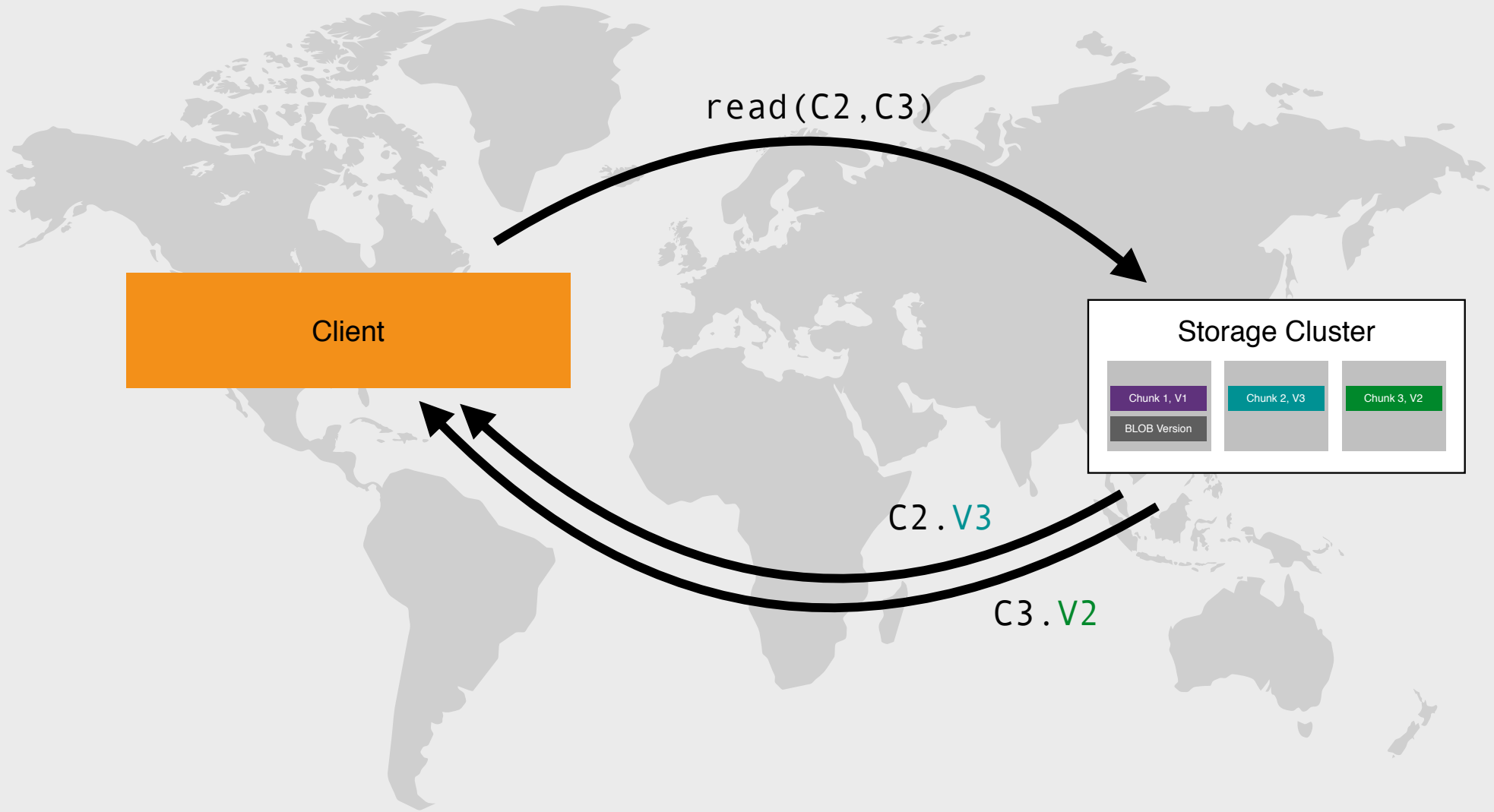


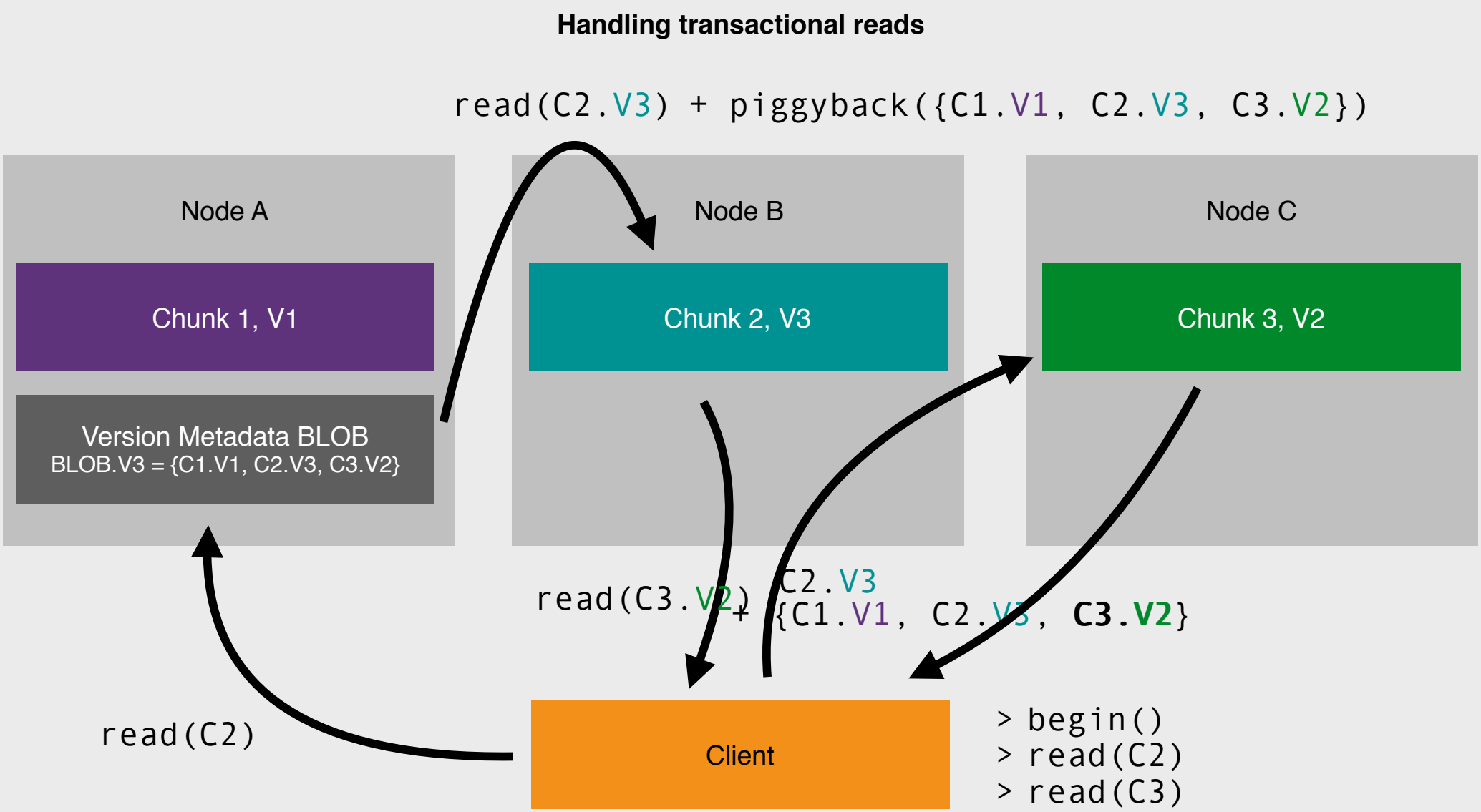
Handling single-chunk reads




Handling multi-chunk reads
read(C3.V2)
Node A
Node B
Node C
Chunk 1, V1
Chunk 2, V3
Chunk 3, V2
Version Metadata BLOB
BLOB.V3 = {C1.V1, C2.V3, C3.V2}
read(C2.V3)
C2.V3
read(C2,C3)
Client
C3.V2

Handling multi-chunk reads




Handling transactional reads
read(C2.V3) + piggyback({C1.V1, C2.V3, C3.V2})
Node A
Chunk 1, V1
Version Metadata BLOB
BLOB.V3 = {C1.V1, C2.V3, C3.V2}
Node B
Chunk 2, V3
Node C
Chunk 3, V2
read(C3.V2)
C2.V3
+ {C1.V1, C2.V3, C3.V2}
read(C2)
Client
> begin()
> read(C2)
> read(C3)

From atomic writes to atomic operations

Leveraging serializable writes, Týr generalizes atomic writes to atomic operations

Simple binary operations can be performed with a single round-trip instead of two

With MonALISA, aggregate updates can be performed atomically and efficiently