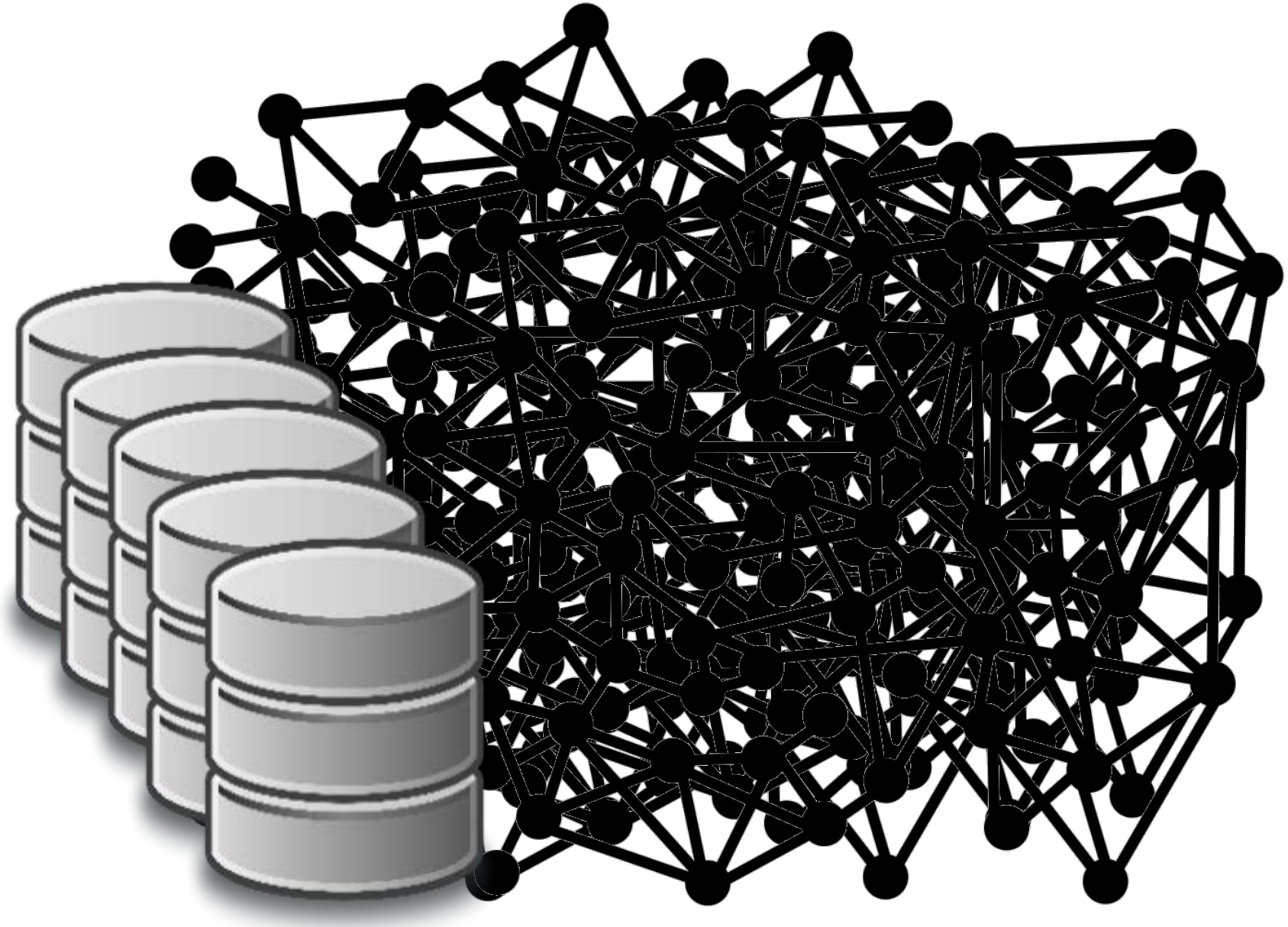
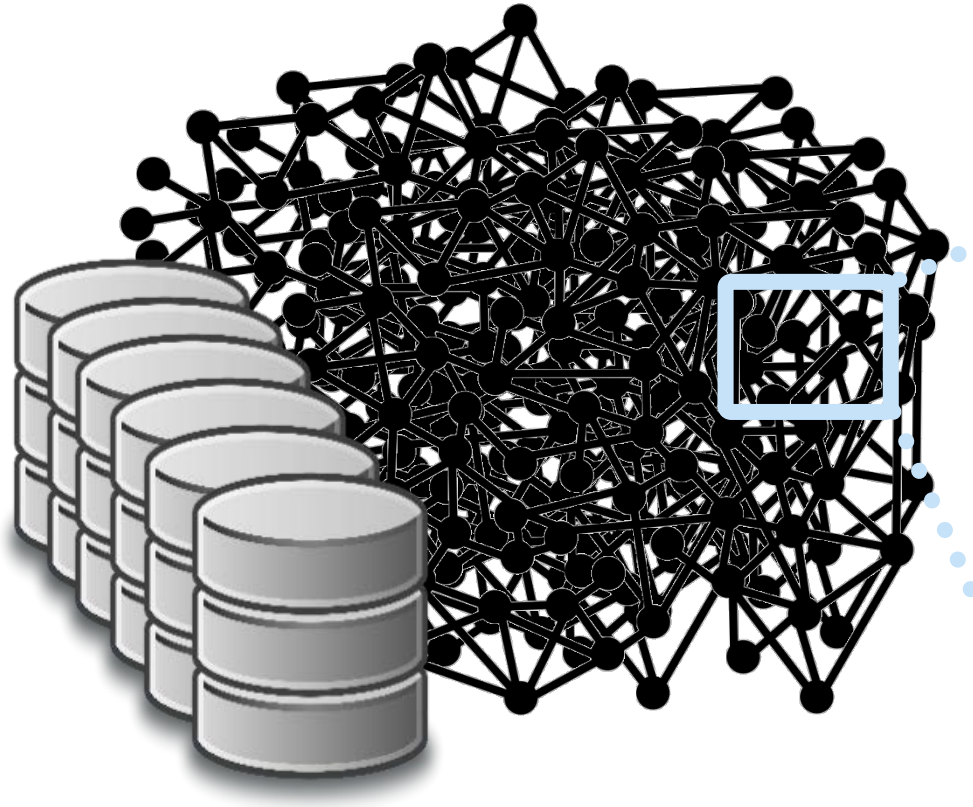
M. Besta, R. Gerstenberger, M. Fischer, M. Podstawski, N. Blach, B. Egeli,
G. Mitenkov, W. Chlapek, M. Michalewicz, H. Niewiadomski, J. Müller, T. Hoefler

The Graph Database Interface: Scaling Online Transactional and Analytical Graph Workloads to Hundreds of Thousands of Cores

Graph Databases (GDBs): A Very Brief Introduction

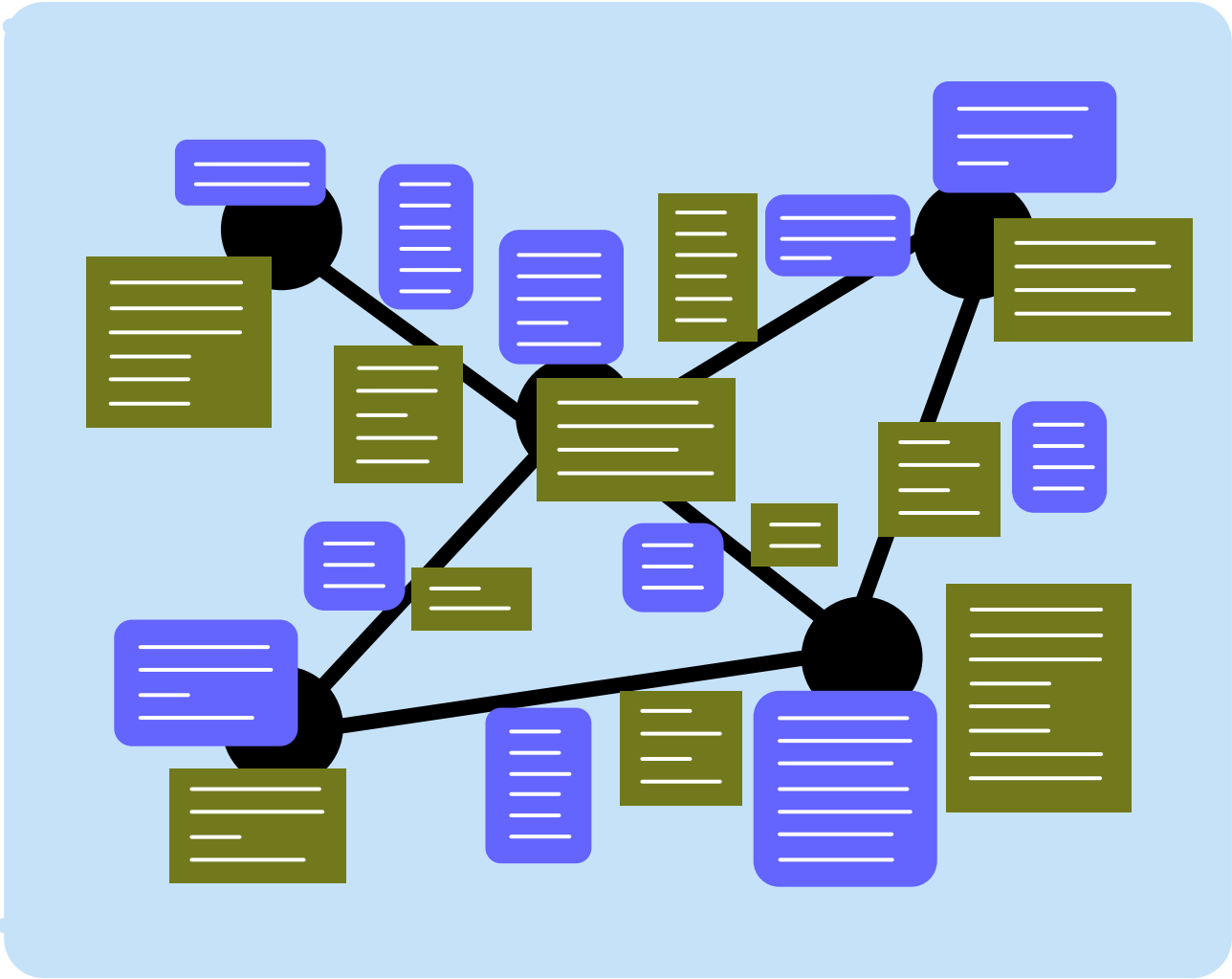


Graph Databases: The Labeled Property Graph (LPG) Data Model



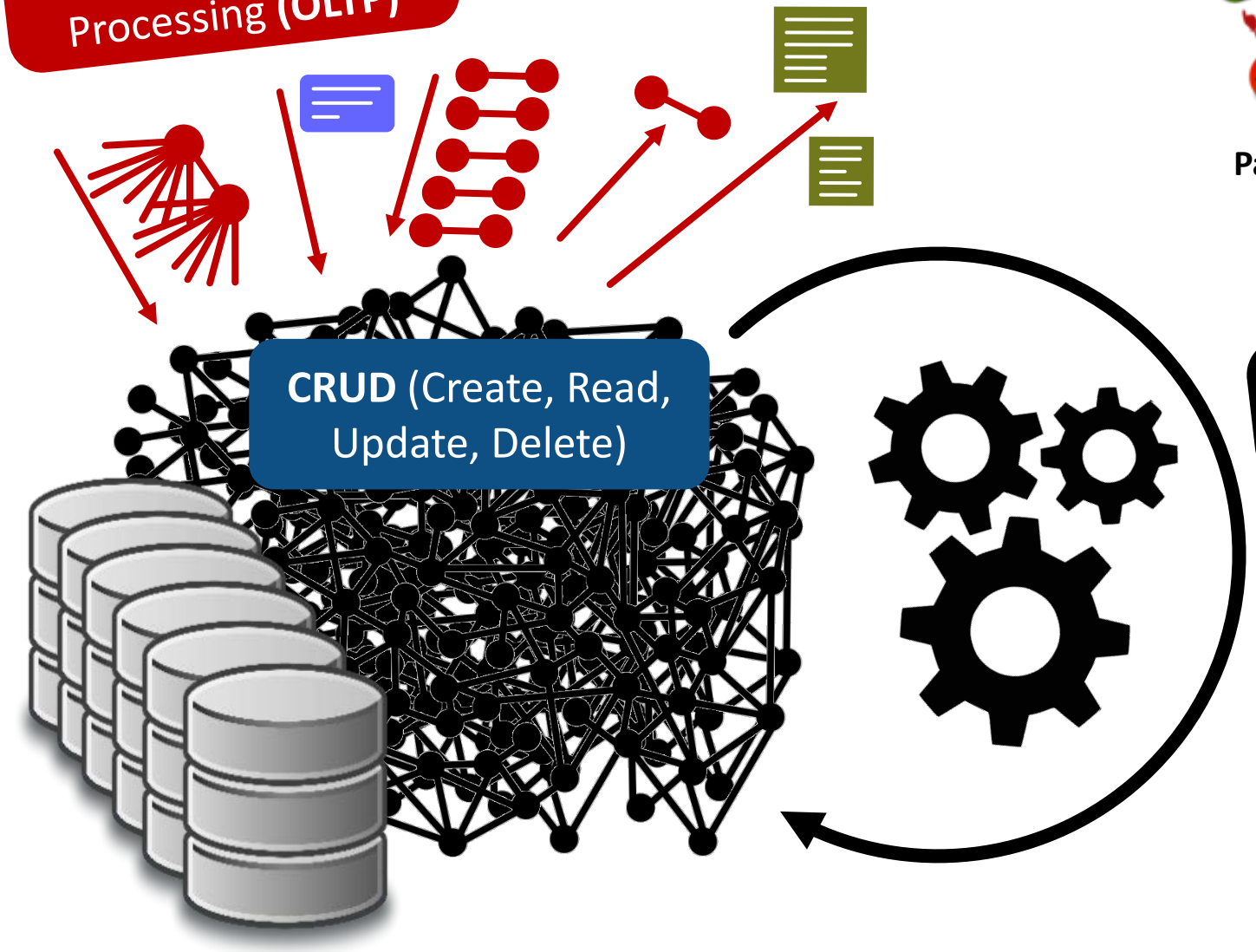
Labels

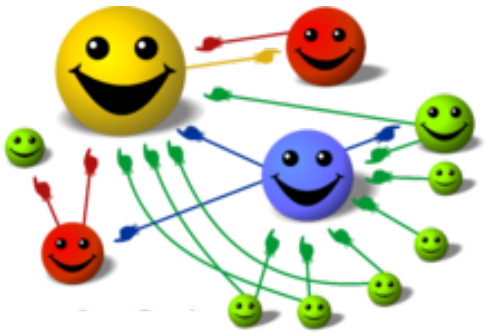
Properties



Graph Databases: Major Workloads

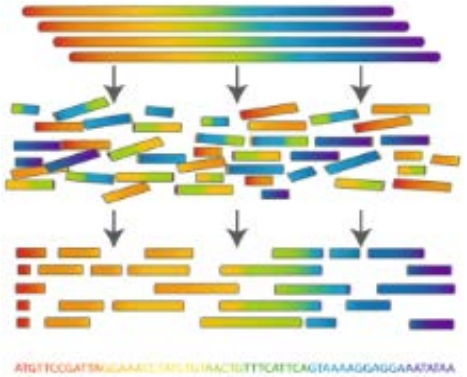
Online Transactional Processing **(OLTP)**





PageRank

Genome (BFS) traversals



Online Analytical Processing **(OLAP)**

Graph Neural Networks

Graph Databases: Where Do We Use Them?

Social sciences

Biology

Chemistry

Communication

Engineering

Medicine

Cybersecurity

Web graph analysis

Graph Databases: State of Challenges & Problems

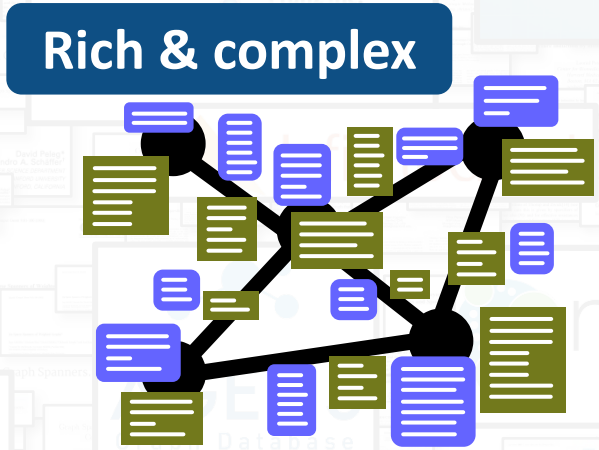
We analyzed > 300 works & dozens of systems, and realized, they all suffer from problems...

HugeGraph
Franz Inc. AllegroGraph
ANZO
Amazon Neptune
fauna
ArcadeDB
Dgraph
InfiniteGraph
ultipa
GraphDB
blazegraph
Vaticle TypeDB
graphbase.ai
AGENS Graph Database
neo4j
OrientDB
TITAN
GALAXYBASE
BangDB
VelocityDB
MEM GRAPH
KATANA GRAPH
TigerGraph
NebulaGraph
FlockDB
STARDOG
ArangoDB
TerminusDB
*Sparksee
JanusGraph

Graph Databases: State of Challenges & Problems

We analyzed > 300 works & dozens of systems, and realized, they all suffer from problems...

Data



It is hard or infeasible to process such datasets with existing systems

Our industry collaborators have 10-1000x more data than they can process with current tools

Graph Databases: State of Challenges & Problems

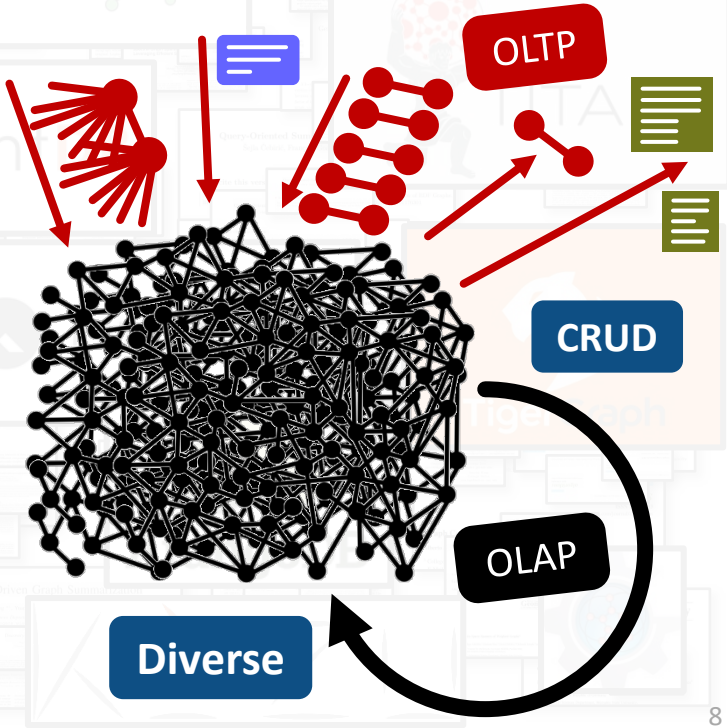
We analyzed > 300 works & dozens of systems, and realized, they all suffer from problems...

Workloads

Systems often focus on a single workload class

Workloads are hard to scale and often have high runtimes with today's systems

Largest scale published so far: MS A1 (2020), **2940 cores, 245 compute nodes;**

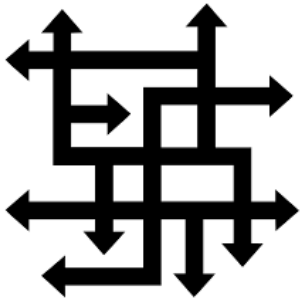


Graph Databases: State of Challenges & Problems

We analyzed > 300 works & dozens of systems, and realized, they all suffer from problems...

System design

Complex



Sharding & Replication

Indexing

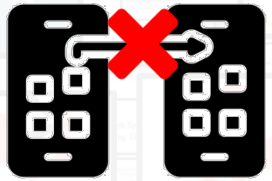
Transactions

Errors

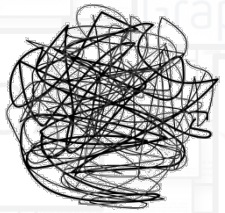
...

ACID

A - Atomicity
C - Consistency
I - Isolation
D - Durability



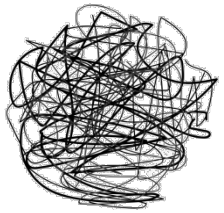
Hard to port



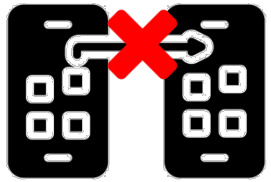
Hard to design, maintain, & extend

An example comment in a production repo of one of the top GDBs: *„We do not know the effects of OLAP when running concurrently with OLTP ones; maybe not use it in production."*

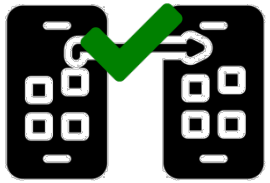
Graph Databases: State of Problems & Our Objectives



Hard to design & maintain → Programmable

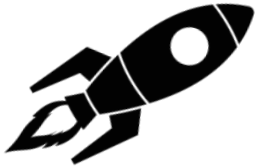


Difficult to port → Portable



Focusing on a single worklad class → Supporting all major workload classes

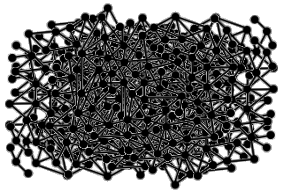
Slow → High performance



Hard to scale → Highly scalable

XS

Hard or infeasible to use with large datasets → Support for arbitrarily large datasets



XXXL

Graph Databases: State of Problems & Our Objectives

How to achieve all these objectives in a single design?

Focusing on a single worklad class

Supporting all major workload classes

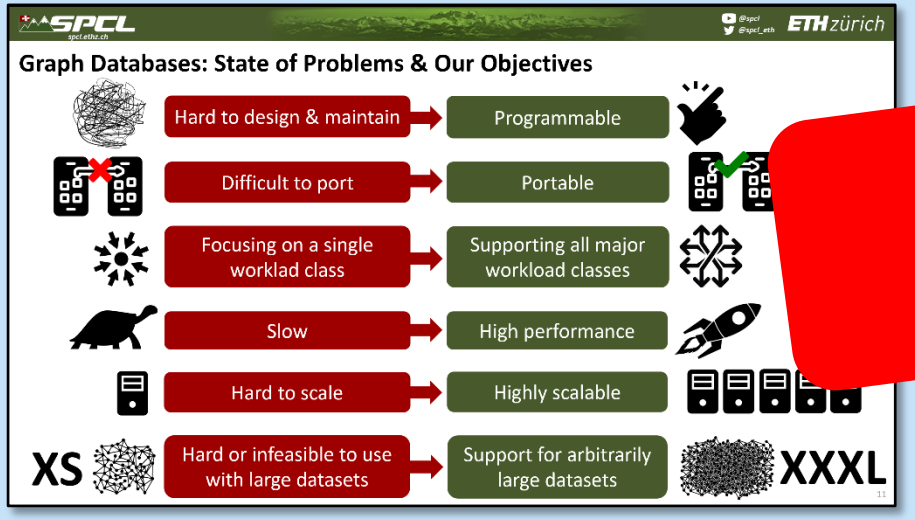
We design & implement the **Graph Database Interface (GDI)**: a paradigm for developing GDBs that ensure all these objectives

with large datasets

large datasets

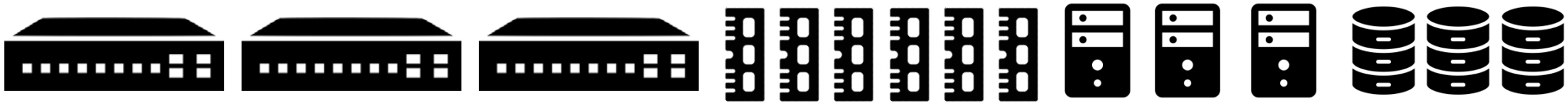
Observation: Developing a GDB Directly on Top of HW Is Not a Good Idea

Graph Database



We don't want to do this!

Hardware access layer (vendor specific)



GDI Key Idea

Bring the **best practices of MPI and its fundamental design feature** into the **landscape of graph databases...**

...Break down the complexity of GDBs with the **separation between the hardware-agnostic interface (API) and the hardware-specific implementation**

GDI Key Idea: Bring the Fundamental MPI Design Feature for Graph Databases

Fundamental performance-critical building blocks for communication in a parallel environment

Portable and **productive** API for developing scalable parallel applications

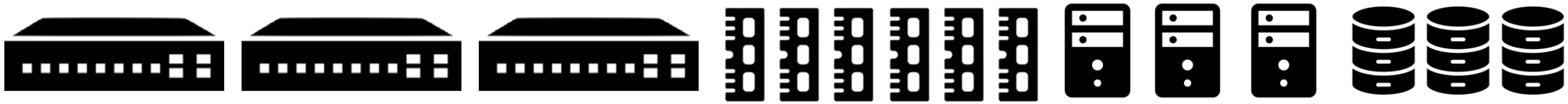
MPI Standard (API)

HW-agnostic

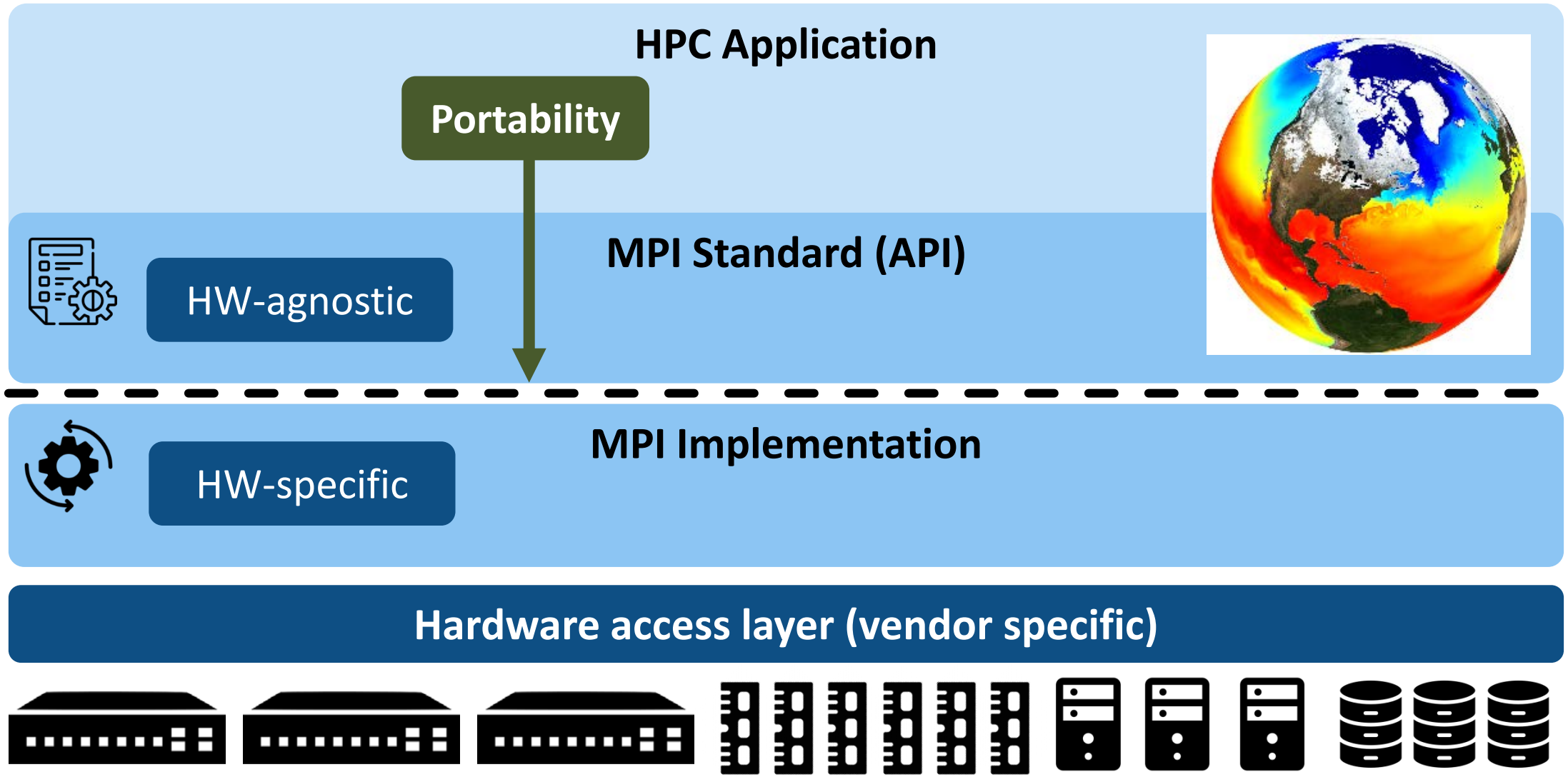
MPI Implementation

HW-specific

Hardware access layer (vendor specific)



GDI Key Idea: Bring the Fundamental MPI Design Feature for Graph Databases



GDI Key Idea: Bring the Fundamental MPI Design Feature for Graph Databases

Fundamental performance-critical building blocks for graph data layout & graph transactions in a parallel environment

Portable and **productive** API for developing scalable graph databases

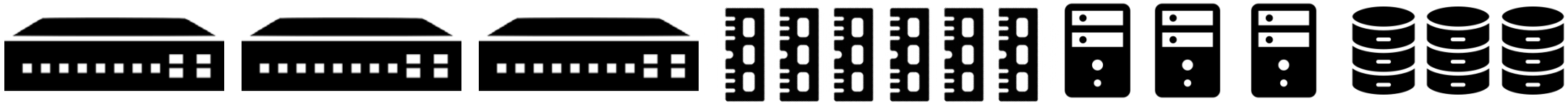
GDI Standard (API)

HW-agnostic

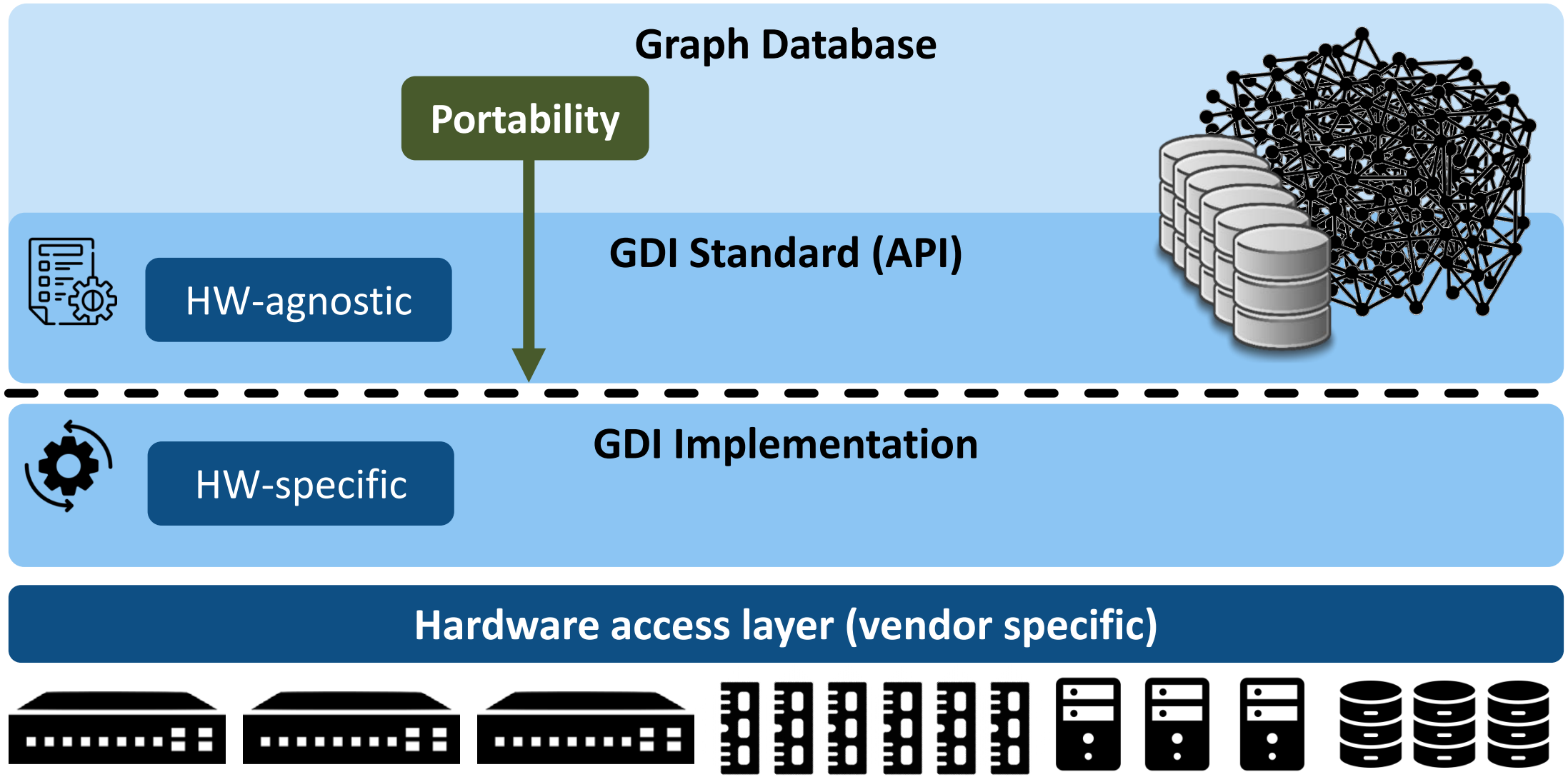
GDI Implementation

HW-specific

Hardware access layer (vendor specific)



GDI Key Idea: Bring the Fundamental MPI Design Feature for Graph Databases

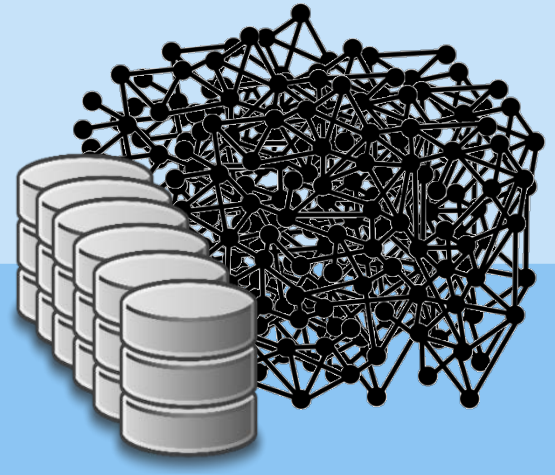


Graph Databases: State of Problems & Our Objectives

Hard to design & maintain	Programmable
Difficult to port	Portable
Focusing on a single worklad class	Supporting all major workload classes
Slow	High performance
Hard to scale	Highly scalable
Hard or infeasible to use with large datasets	Support for arbitrarily large datasets

XS XXXL

Graph Database



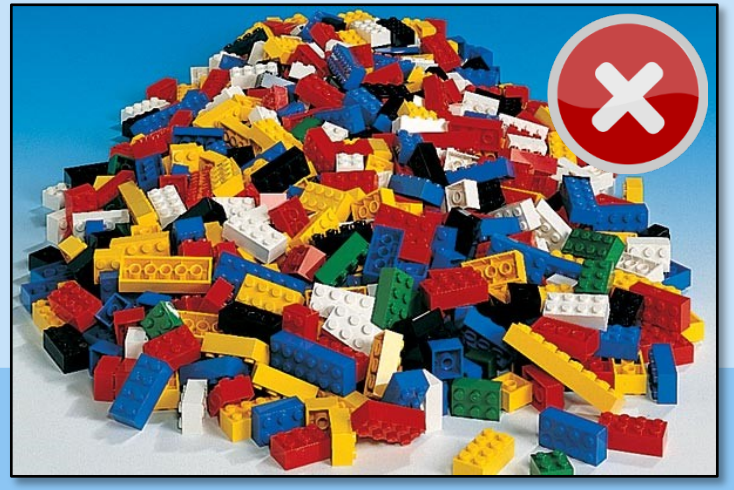
GDI Standard (API)

HW-agnostic

GDI Implementation

HW-specific

Graph Database



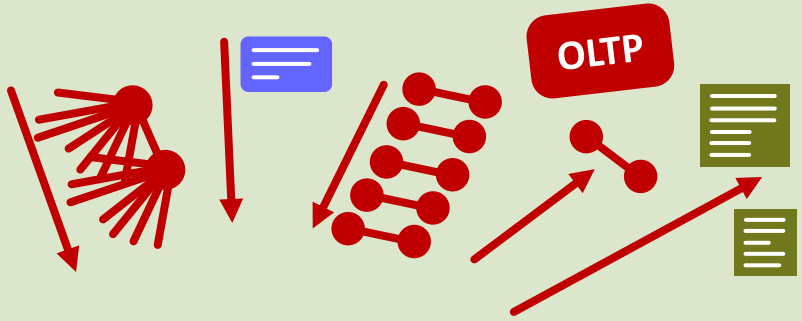
GDI Standard (API)

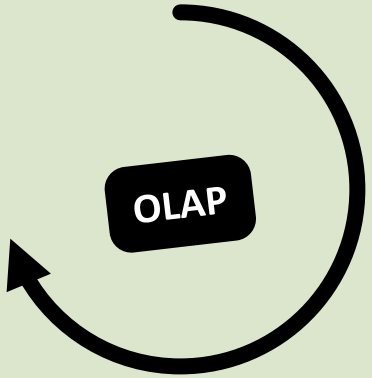
Main focus: **data** & its **operations**

- **Graph data**
- **Graph transactions**
- **Graph indexes**
- **...**

Idea: OLAP Queries as Collective Transactions

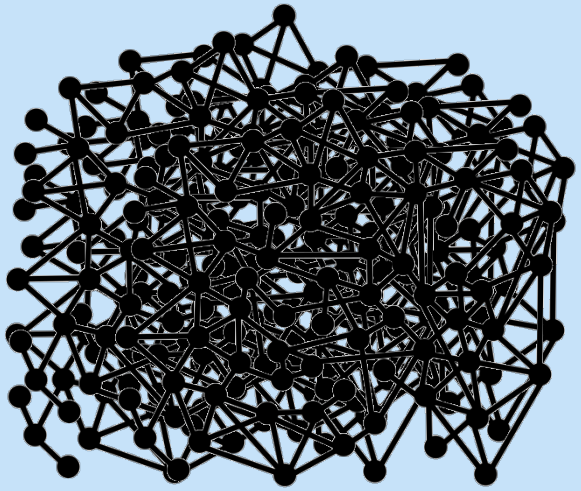
Graph transactions





Access, modify

Graph data



Vertices

Labels | Properties

Edges

Labels | Properties

Idea: OLAP Queries as Collective Transactions

Collective semantics borrowed from the MPI

Graph database user A

```
...
GDI_StartCollectiveTransaction()
  ... // Triangle Counting
GDI_CloseCollectiveTransaction()
GDI_StartLocalTransaction()
  GDI_UpdateVertexProperty(v_ID, ...)
  GDI_AddVertex(...)
  GDI_AddEdge(v_ID, ...)
  ...
GDI_CloseLocalTransaction()
GDI_StartCollectiveTransaction()
  ... // BFS traversal
GDI_CloseCollectiveTransaction()
...
```

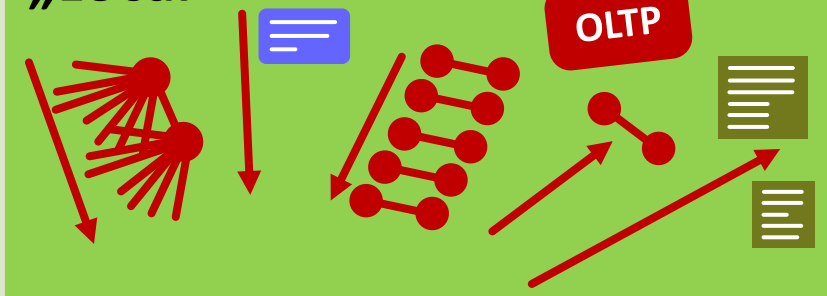
Users B, C, D, ...

When exiting, each process must have finished its participation in a given graph collective transaction (**blocking**)

After exiting, a process can poll for completion (**non-blocking**)

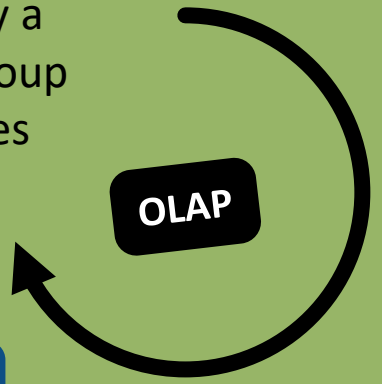
Graph transactions

„Local"



Collective

Executed by a dedicated group of processes



Idea: OLAP Queries as Collective Transactions

Collective semantics borrowed from the MPI

Improving programmability:

Semantics of parallel OLTP and OLAP queries are now **well-defined**

Graph database user A	Users B, C, D, ...	
...		
GDI_StartCollectiveTransaction()		
... // Triangle Counting		
GDI_CloseCollectiveTransaction()		
GDI_StartLocalTransaction()		
GDI_UpdateVertexProperty(v_ID, ...)		
GDI_AddVertex(...)		
GDI_AddEdge(v_ID, ...)	...	...
...		
GDI_CloseLocalTransaction()		
GDI_StartCollectiveTransaction()		
... // BFS traversal		
GDI_CloseCollectiveTransaction()		
...		

An example comment in a production repo of one of the top GDBs: *„We do not know the effects of OLAP when running concurrently with OLTP ones; maybe not use it in production."*

All routines & other design ideas are in...

MPI: A Message-Passing Interface Standard

Version 4.1

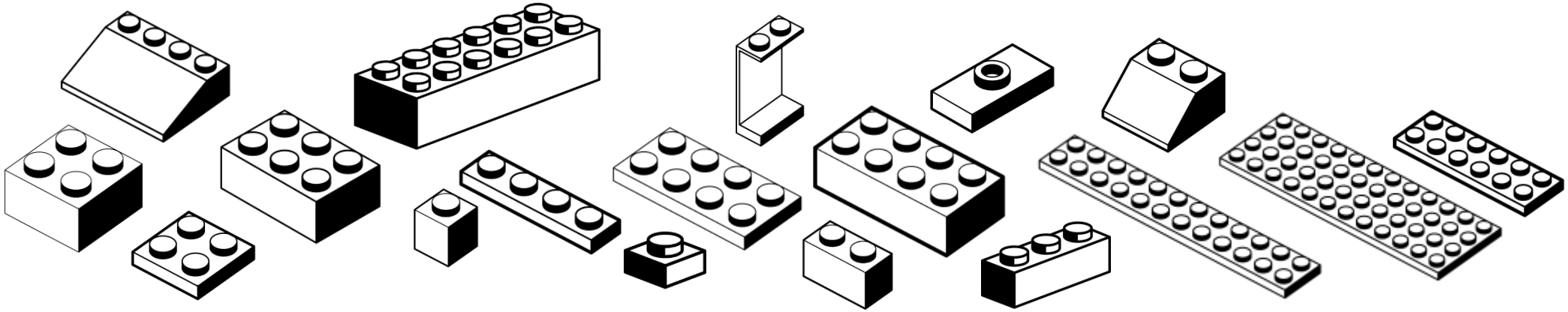
Message Passing Interface Forum

GDI: A Graph Database Interface Standard

105 routines

Maciej Besta*†, Robert Gerstenberger*†, Nils Blach, Marc Fischer‡, Torsten Hoefler†

ETH Zurich



Graph Databases: State of Problems & Our Objectives

Hard to design & maintain → Programmable

Resolved by **GDI Standard (API)**

Portable

Focusing on a single worklad class → Supporting all major workload classes

Slow → High performance

Hard to scale → Highly scalable

XS

Hard or infeasible to use with large datasets → Supporting arbitrarily large datasets

XXXL

Graph Database

GDI Standard (API)

HW-agnostic

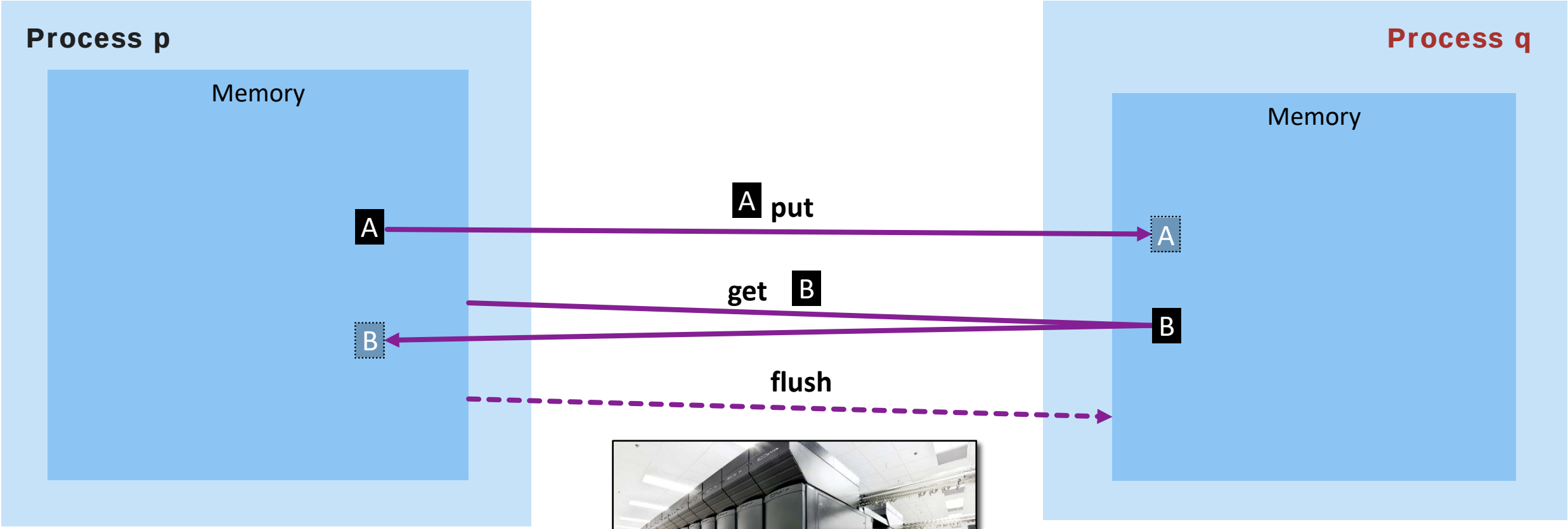
HW-specific

GDI Implementation
GDI-RMA (GDA)

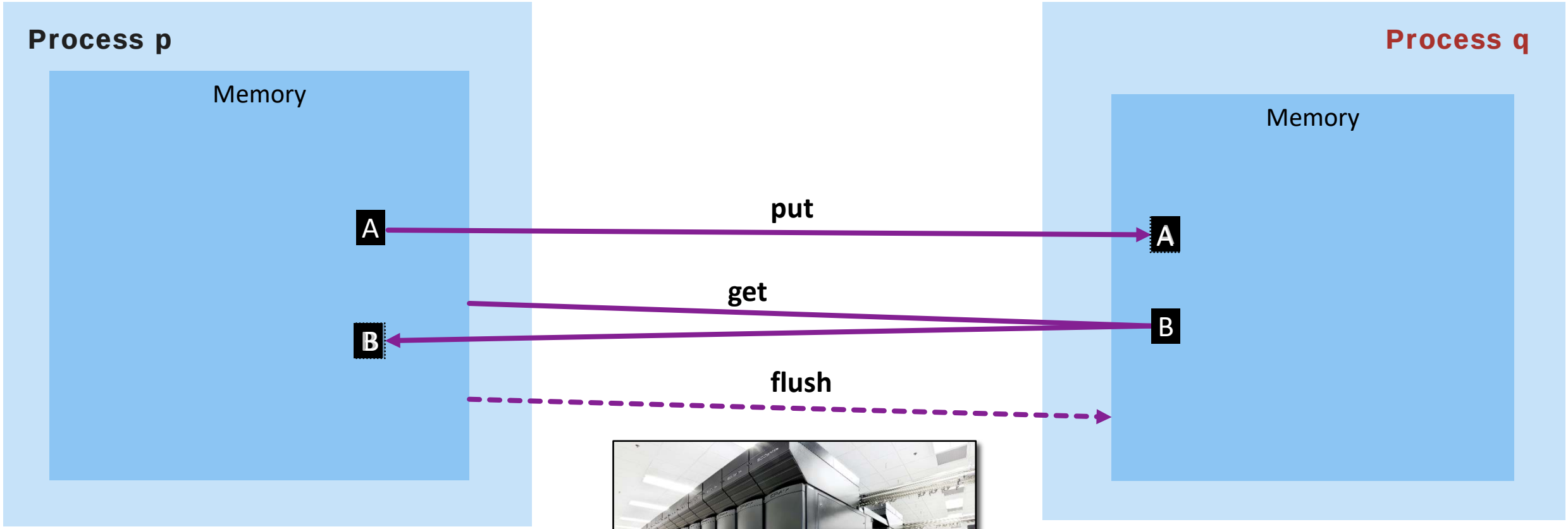
for RDMA systems

Key mechanism for high-performance:
One-Sided Non-Blocking RDMA

RDMA One-Sided for High-Performance Graph Transactions



RDMA One-Sided for High-Performance Graph Transactions

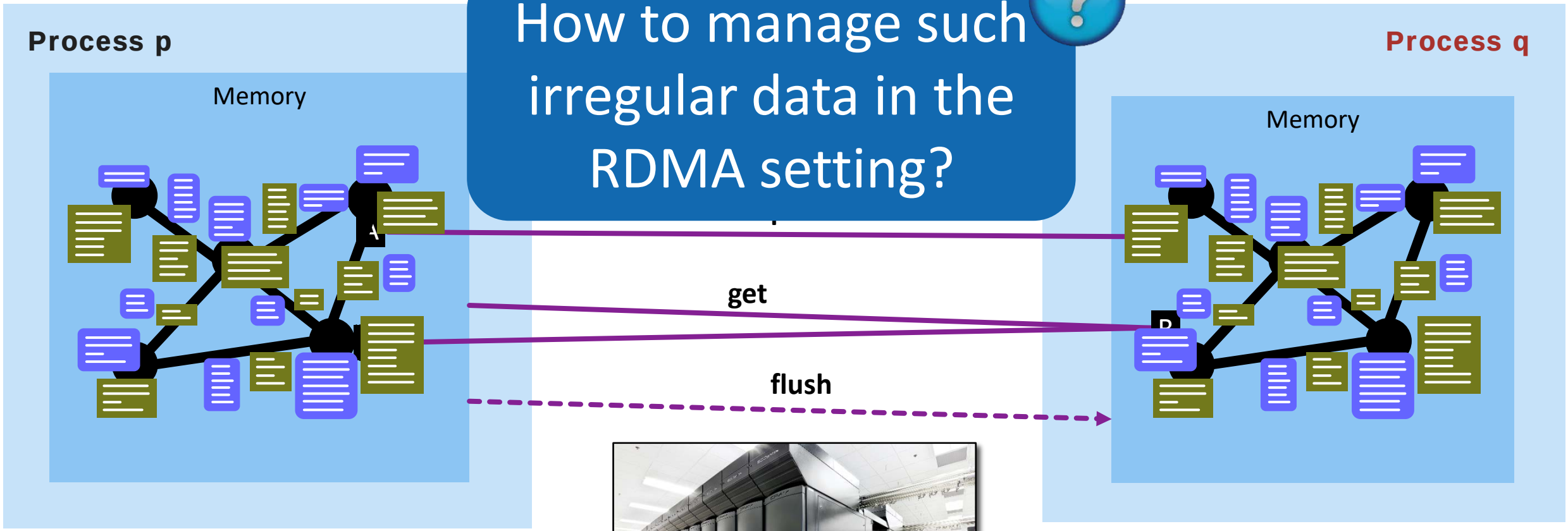


No interrupt overheads

No OS overheads

RDMA One-Sided for High-Performance Graph Transactions

How to manage such irregular data in the RDMA setting?



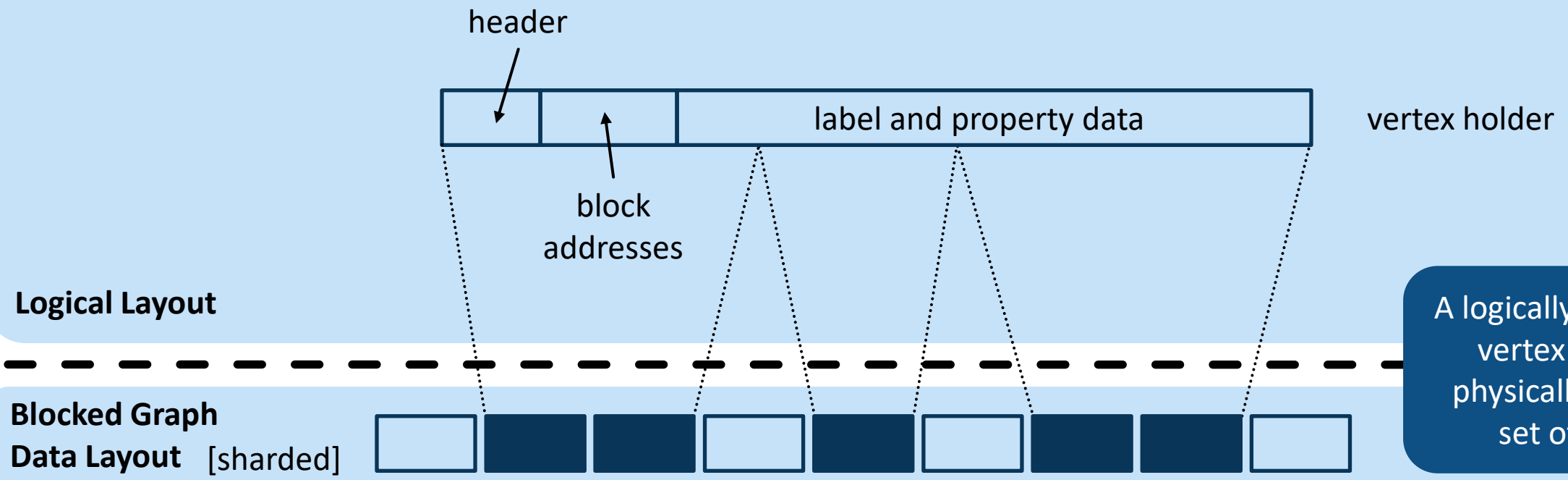
No interrupt overheads

No OS overheads

GDI-RMA: Data Layout

Logical layout is what is visible during a transaction

Logical layout is flexible in size, facilitating programmability



A logically contiguous vertex holder is physically kept as a set of blocks

Blocks are fixed in size: it facilitates efficient RDMA ops + simple memory management

Vertices can fit into a single block, so the best case for vertex fetching is a single get

Block size is configurable

GDI-RMA: Vertex Update

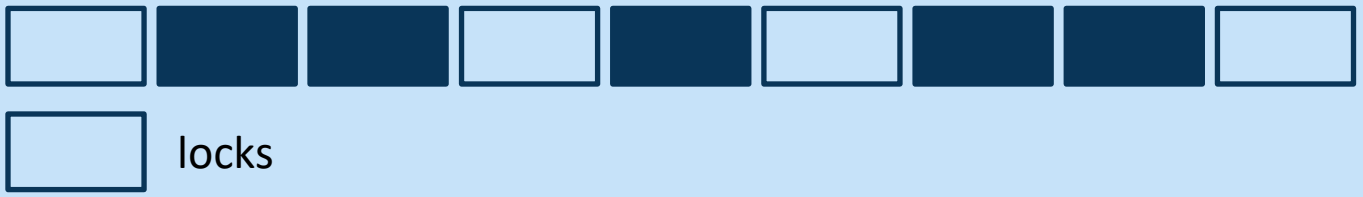
```
GDI_StartLocalTransaction()
GDI_AddPropertyToVertex(...,vH)
GDI_CloseLocalTransaction(COMMIT)
```

Create local data structures

Logical Layout

Blocked Graph Data Layout

[sharded]



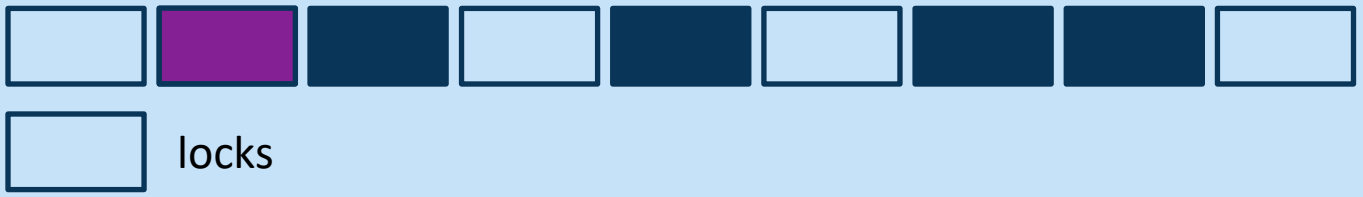
GDI-RMA: Vertex Update

GDI_StartLocalTransaction()
GDI_AddPropertyToVertex(...,vH)
GDI_CloseLocalTransaction(COMMIT)

Query an index:
find a primary block
+ acquire a read lock

Logical Layout

Blocked Graph Data Layout
[sharded]



database blocks

GDI-RMA: Vertex Update

GDI_StartLocalTransaction()
GDI_AddPropertyToVertex(...,vH)
GDI_CloseLocalTransaction(COMMIT)

Query an index:
find a primary block
+ acquire a read lock

acquire read lock

Logical Layout

Blocked Graph Data Layout
[sharded]

database blocks

GDI-RMA: Vertex Update

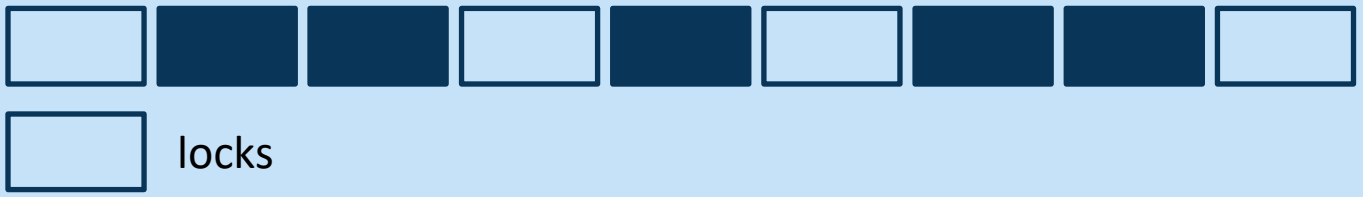
```
GDI_StartLocalTransaction()
GDI_AddPropertyToVertex(...,vH)
GDI_CloseLocalTransaction(COMMIT)
```

Fetch the block data + create the logical representation

Logical Layout

Blocked Graph Data Layout

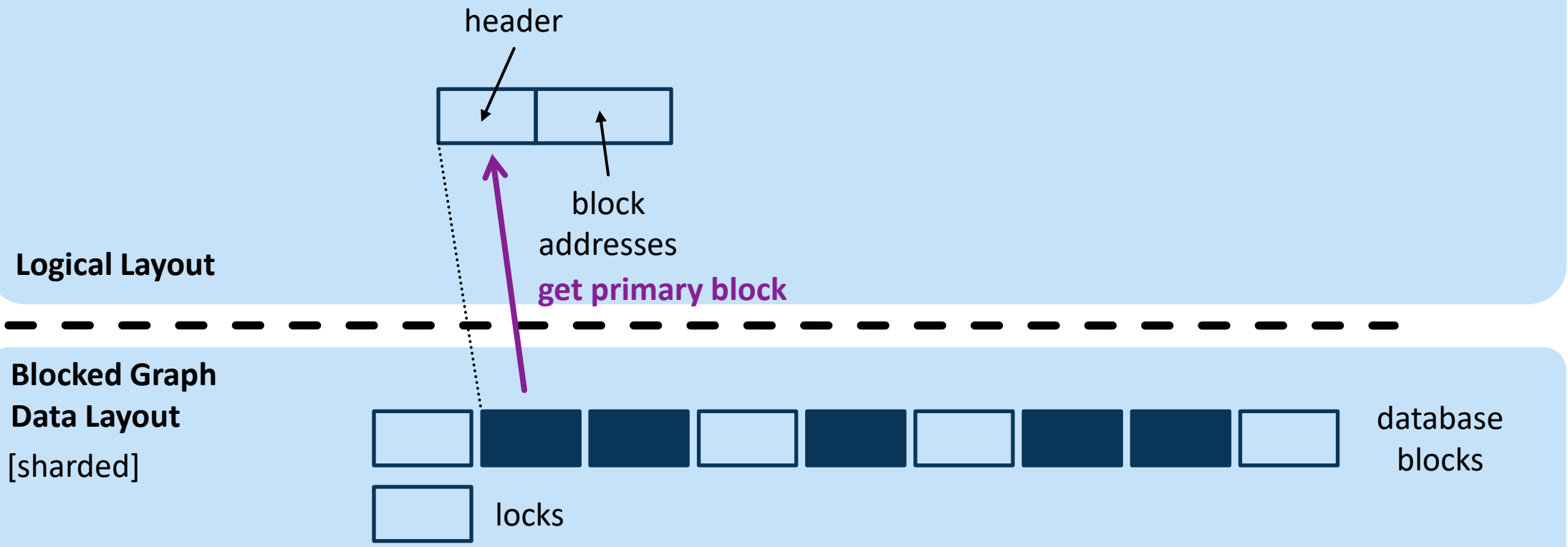
[sharded]



GDI-RMA: Vertex Update

```
GDI_StartLocalTransaction()
GDI_AddPropertyToVertex(...,vH)
GDI_CloseLocalTransaction(COMMIT)
```

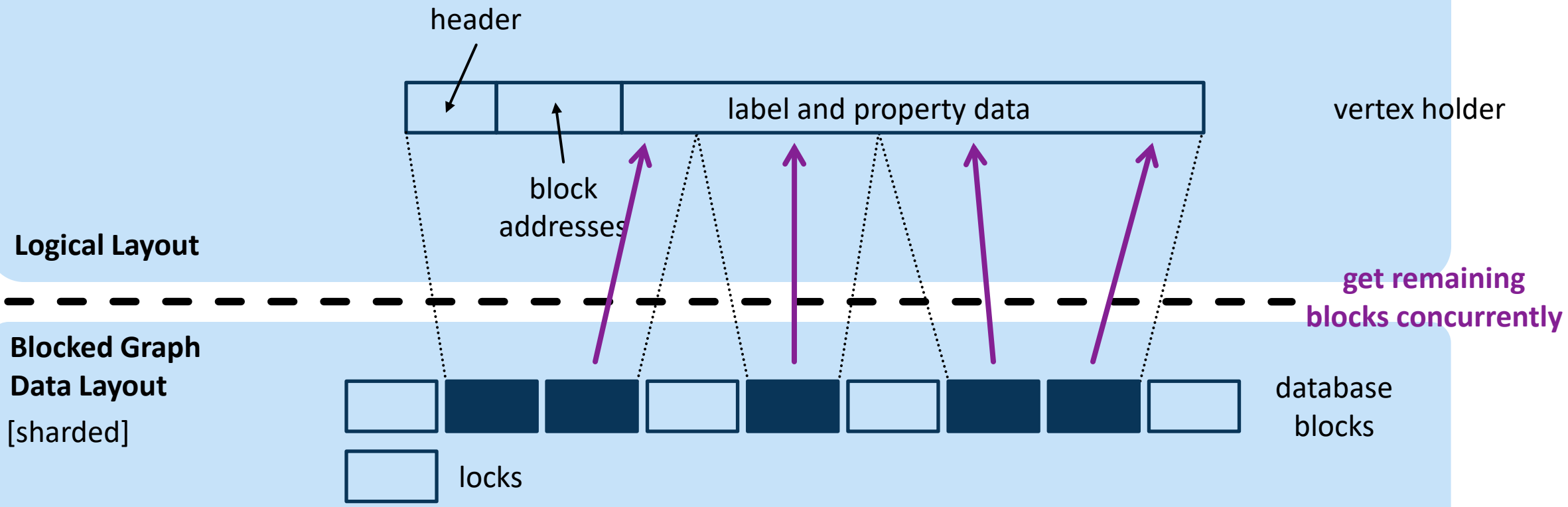
Fetch the block data + create the logical representation



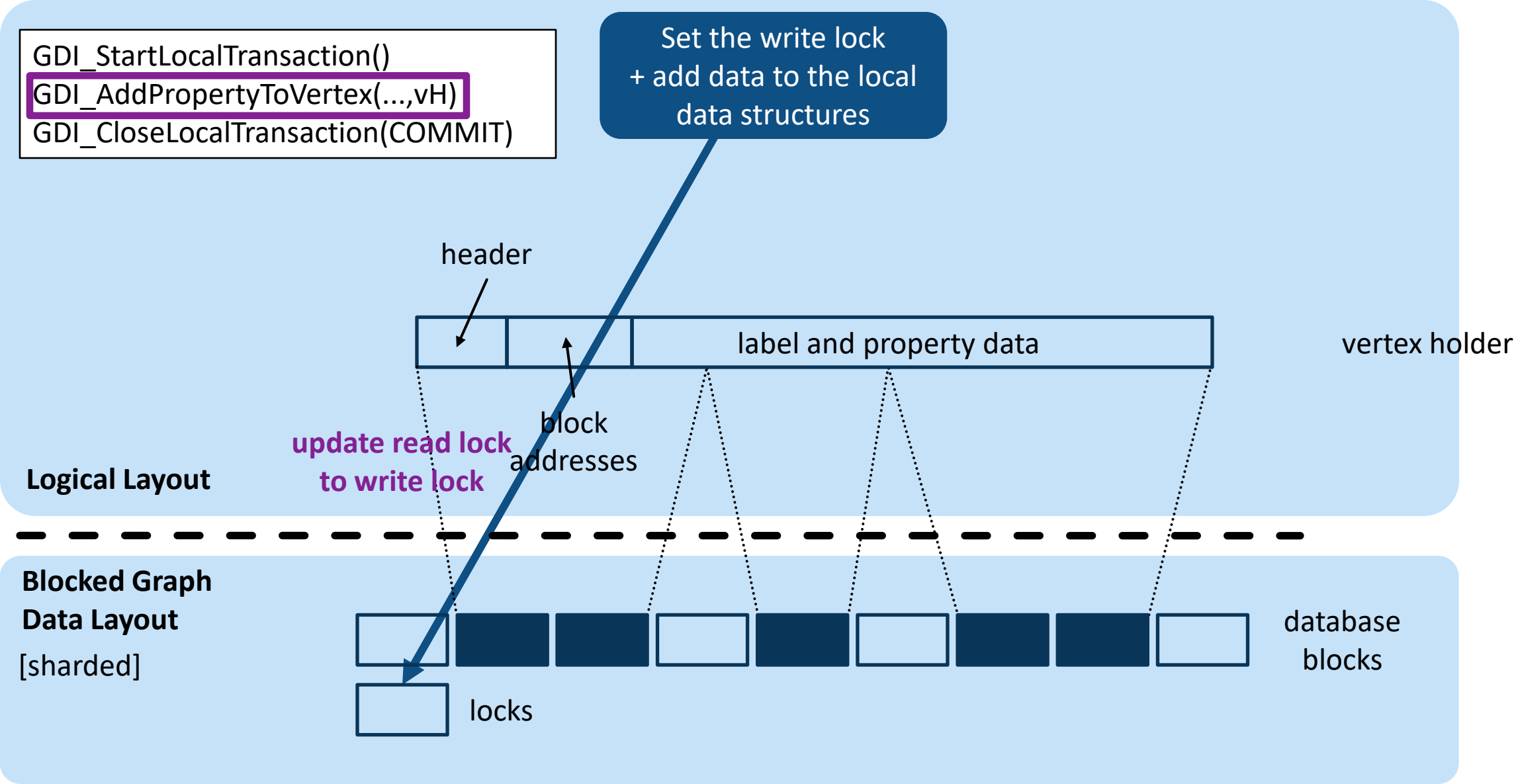
GDI-RMA: Vertex Update

```
GDI_StartLocalTransaction()
GDI_AddPropertyToVertex(...,vH)
GDI_CloseLocalTransaction(COMMIT)
```

Fetch the block data + create the logical representation



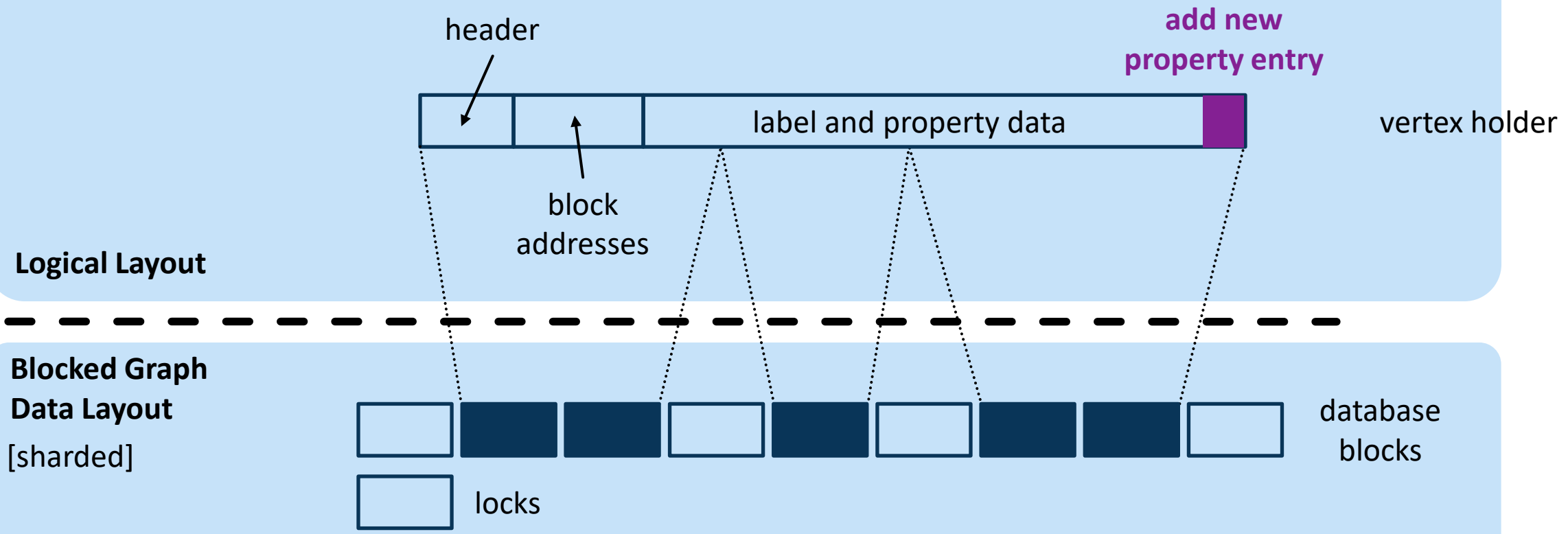
GDI-RMA: Vertex Update



GDI-RMA: Vertex Update

```
GDI_StartLocalTransaction()
GDI_AddPropertyToVertex(...,vH)
GDI_CloseLocalTransaction(COMMIT)
```

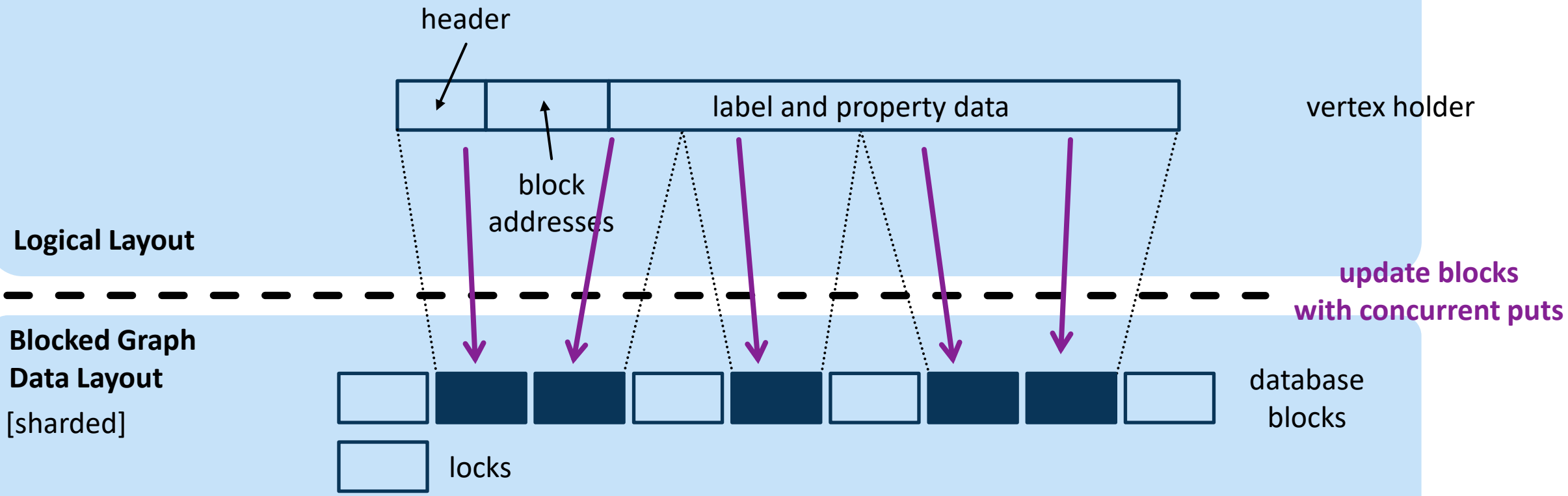
Set the write lock
+ add data to the local
data structures



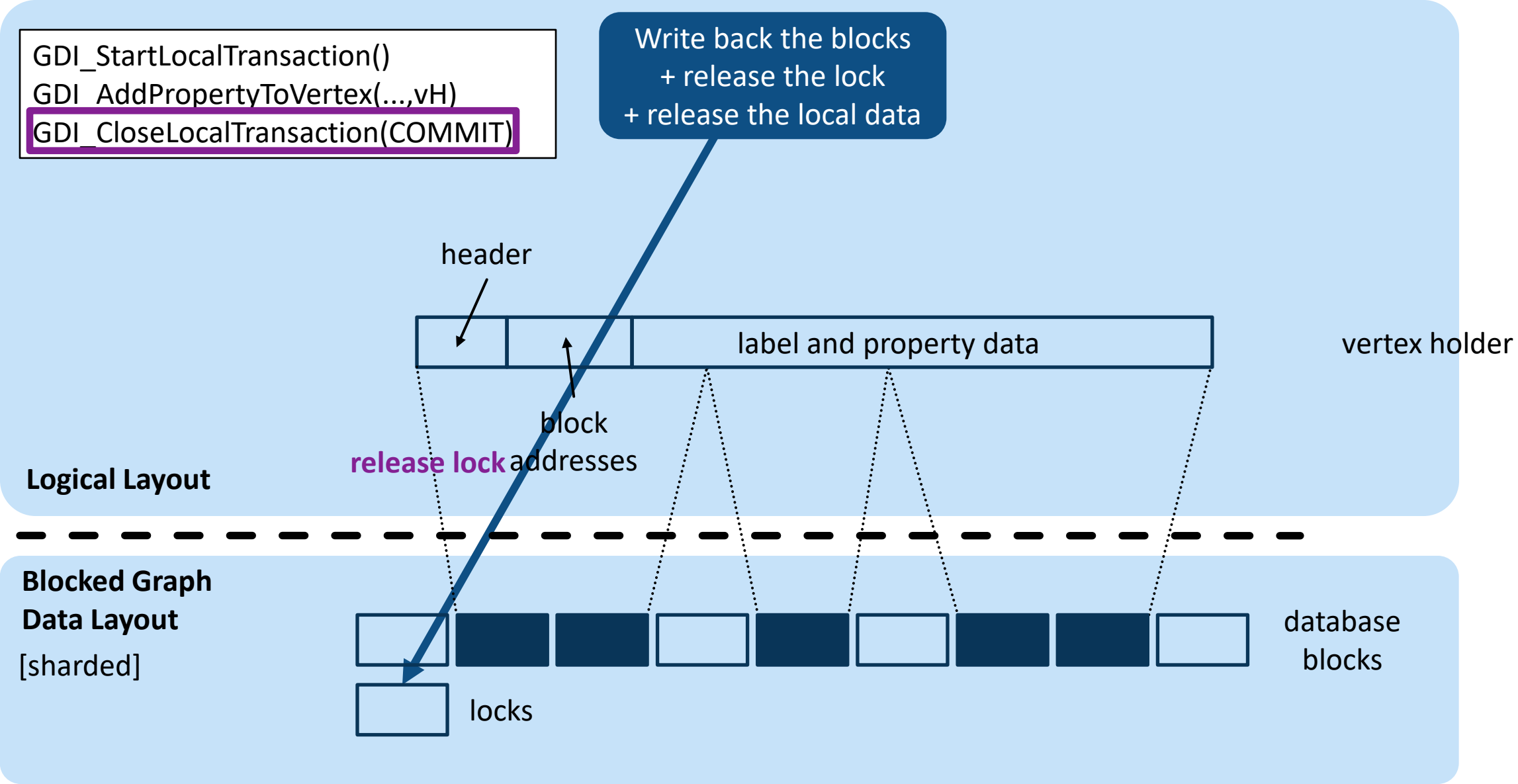
GDI-RMA: Vertex Update

```
GDI_StartLocalTransaction()
GDI_AddPropertyToVertex(...,vH)
GDI_CloseLocalTransaction(COMMIT)
```

Write back the blocks
+ release the lock
+ release the local data



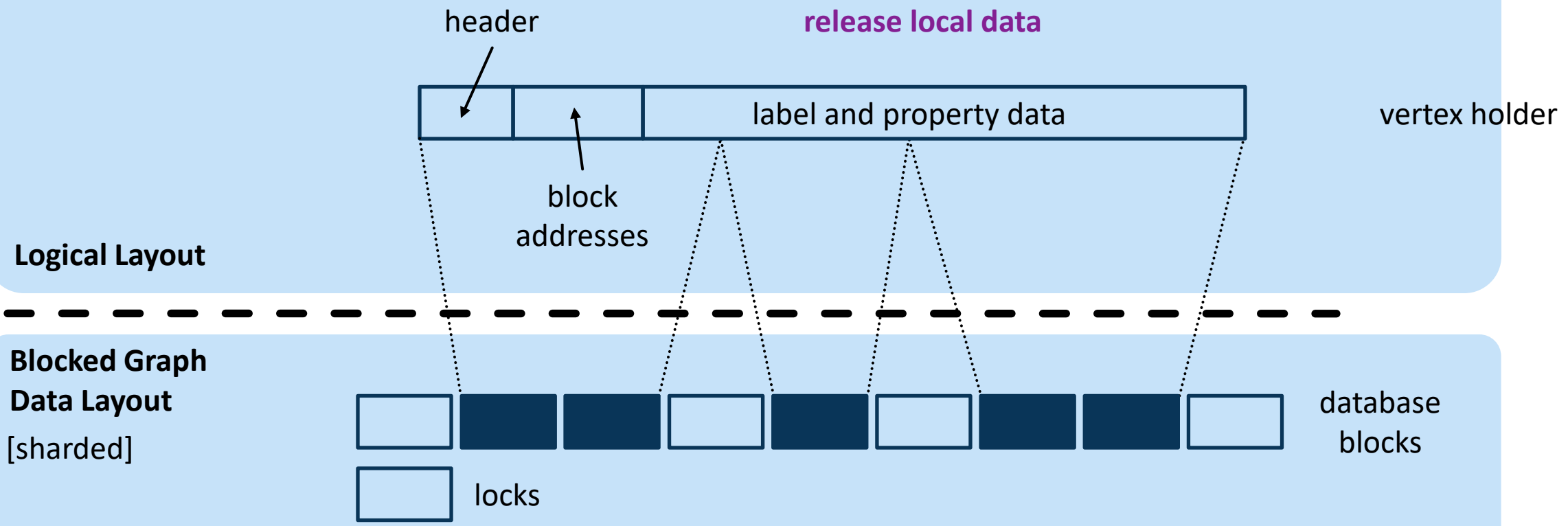
GDI-RMA: Vertex Update



GDI-RMA: Vertex Update

```
GDI_StartLocalTransaction()
GDI_AddPropertyToVertex(...,vH)
GDI_CloseLocalTransaction(COMMIT)
```

Write back the blocks
+ release the lock
+ release the local data



GDI-RMA: Vertex Update

```
GDI_StartLocalTransaction()
GDI_AddPropertyToVertex(...,vH)
GDI_CloseLocalTransaction(COMMIT)
```

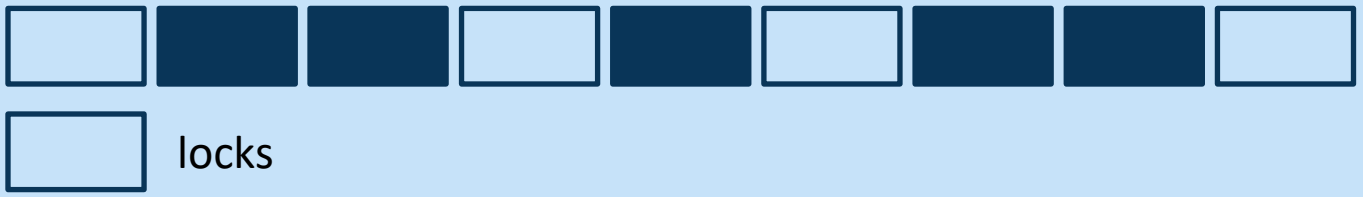
Fine-grained reader-writer synchronization

Parallel updates with non-blocking RDMA

Logical Layout

Blocked Graph Data Layout

[sharded]



GDI-RMA: Vertex Update

```
GDI_StartLocalTransaction()
GDI_AddPropertyToVertex(...,vH)
GDI_CloseLocalTransaction(COMMIT)
```

Fine-grained reader-writer synchronization

Parallel updates with non-blocking RDMA

Hundreds protocols more...

github.com/spcl/GDI-RMA

Logical Layout

Blocked Graph Data Layout

[sharded]

Nearly all routines have theoretical performance guarantees (**work, depth, communicated data volume**)

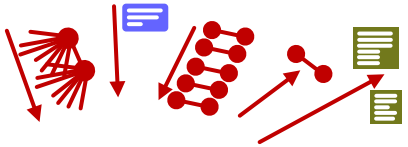
Evaluation: Used Machines & Objectives

CSCS Cray Piz Daint,
64/128 GB per compute node

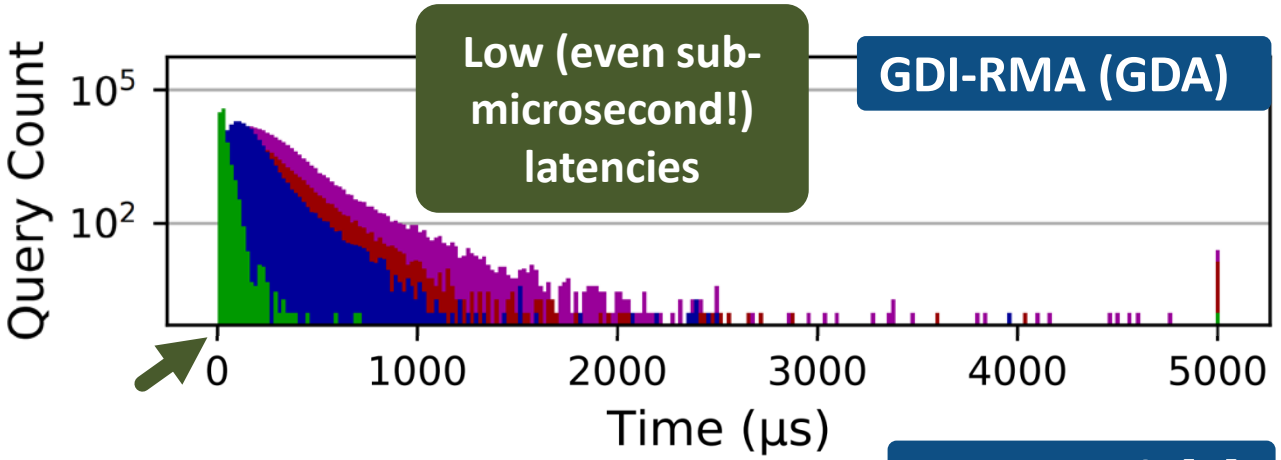
We use the <u>full scale</u> of Piz Daint for GDA:

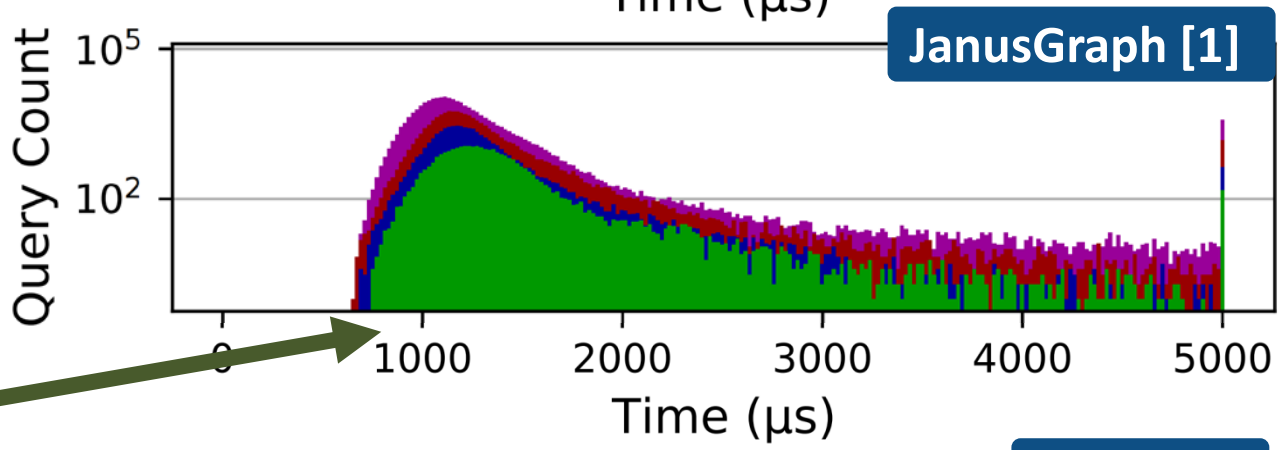
121,680 cores
7,142 servers
77.3 TB RAM

Evaluation (OLTP)

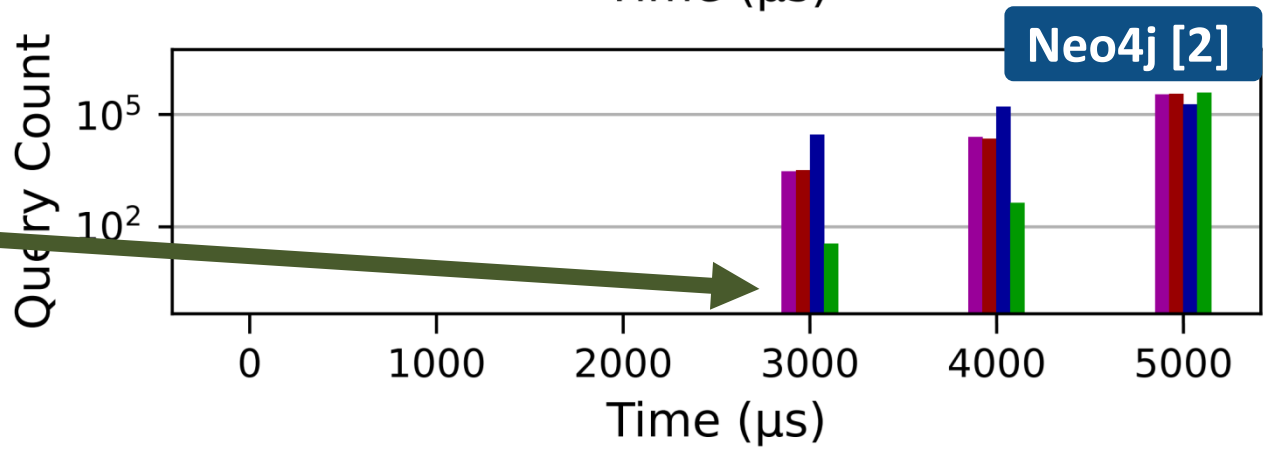


1 node | 2 nodes | 4 nodes | 8 nodes





≈1000x speedups



≈ 3000x speedups

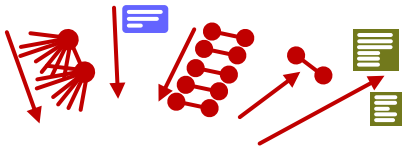
[1] Linux Foundation. 2018. JanusGraph. Available at http://janusgraph.org/
[2] I. Robinson et al.. 2015. Graph Databases (2nd ed.). O'Reilly.

Each system runs in-memory

IBM GRAKN LABS Google HORTONWORKS

#1 top GDB (DB-Engines Ranking)

Evaluation (OLTP)

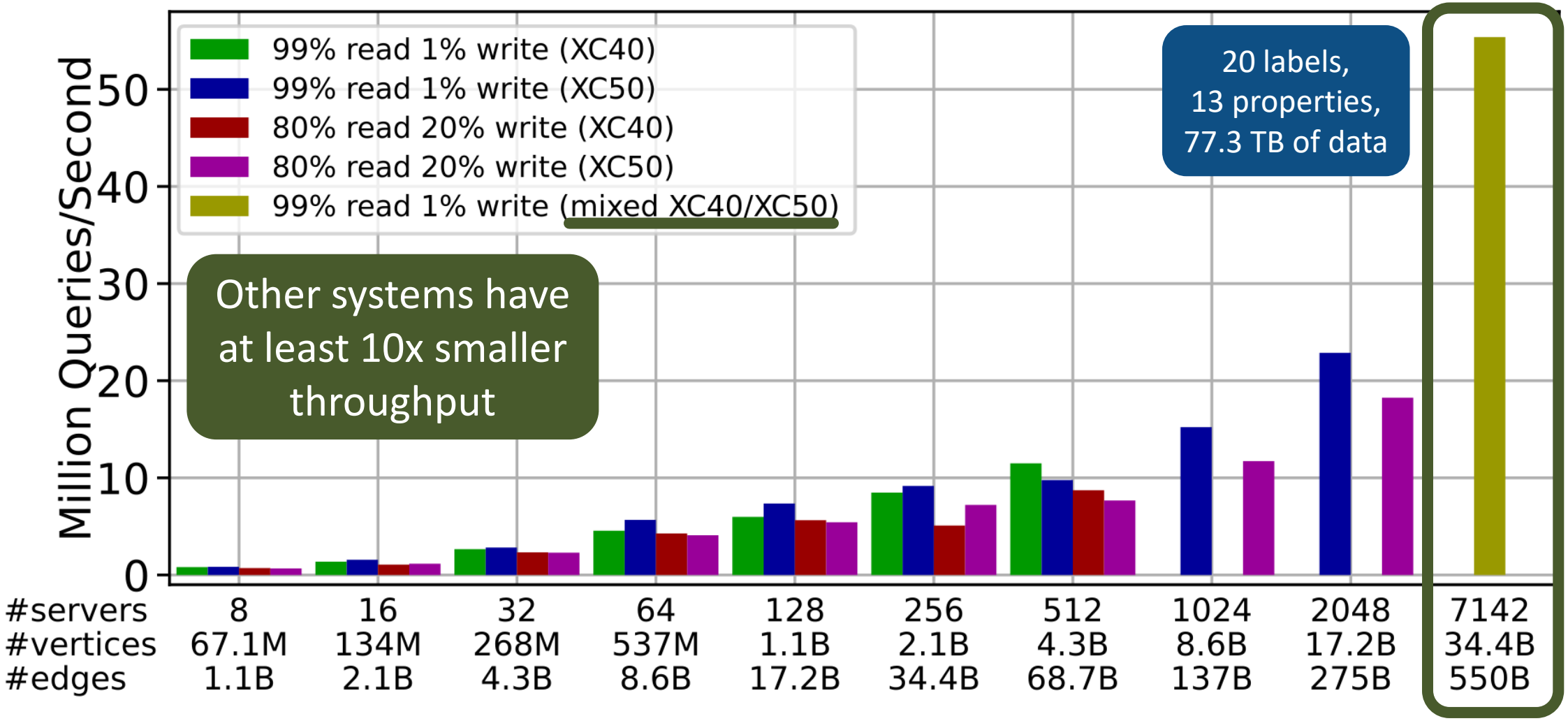


High throughput ✓

Highly scalable ✓

The **largest run ever** (OLTP) in the GDB literature, by **41x** (vs. MS A1) and **950x** (vs. Neo4j)

Our design scales to **121,680 cores** (7,142 compute nodes)



20 labels, 13 properties, 77.3 TB of data

Other systems have at least 10x smaller throughput

Evaluation (OLAP)

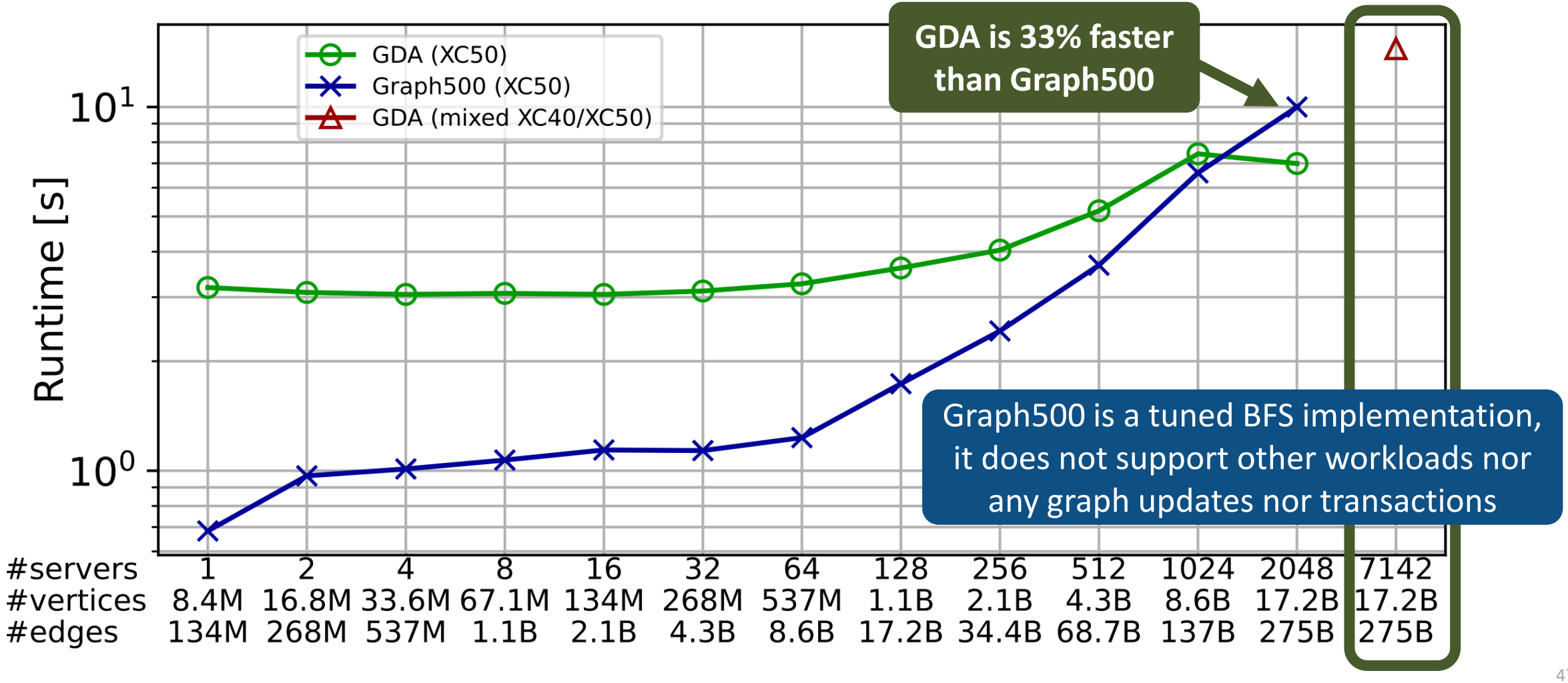
BFS

Low latency ✓

Highly scalable ✓

The **largest run ever** (OLAP) in the GDB literature, by **26x** (vs. TigerGraph) and **950x** (vs. Neo4j)

Our design scales to **121,680 cores** (7,142 compute nodes)



Graph Databases: State of Problems & Our Objectives

Hard to design & maintain → Programmable

Resolved by **GDI Standard (API)**

Portable

Focusing on a single worklad class → Supporting all major workload classes

Slow → High performance

Resolved by **GDI Implementation (GDA-RMA)**

github.com/spcl/GDI-RMA

Highly scalable

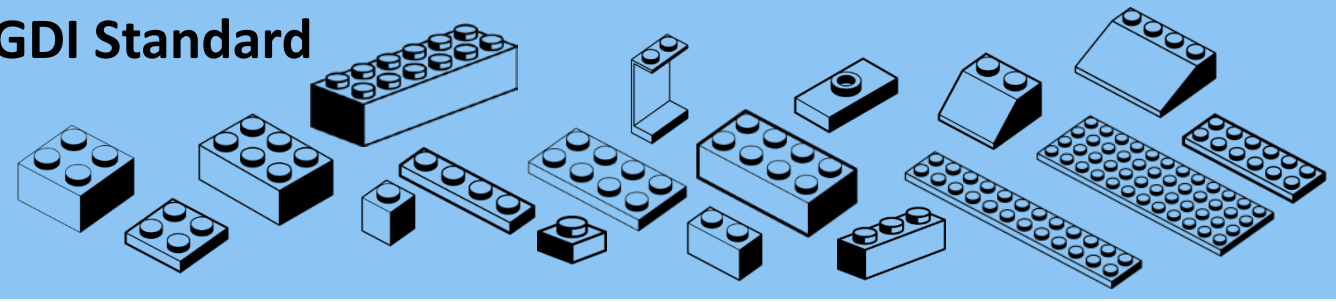
Hard or infeasible to use with large datasets → Supporting arbitrarily large datasets

XXXL

Conclusion

Graph Database Interface (GDI): **„MPI for Graph Databases"**:
it enables principled design & implementation of graph databases that are:

GDI Standard



- Programmable
- Portable
- Supporting all major workload classes

GDI Implementation for RDMA systems (GDA-RMA)

github.com/spcl/GDI-RMA

- High performance
- Highly scalable
- Supporting arbitrarily large datasets

Thank you

Want to know more?

youtube.com/@spcl

spcl.inf.ethz.ch

twitter.com/spcl_eth

github.com/spcl