Mark Raasveldt, Hannes Mühleisen

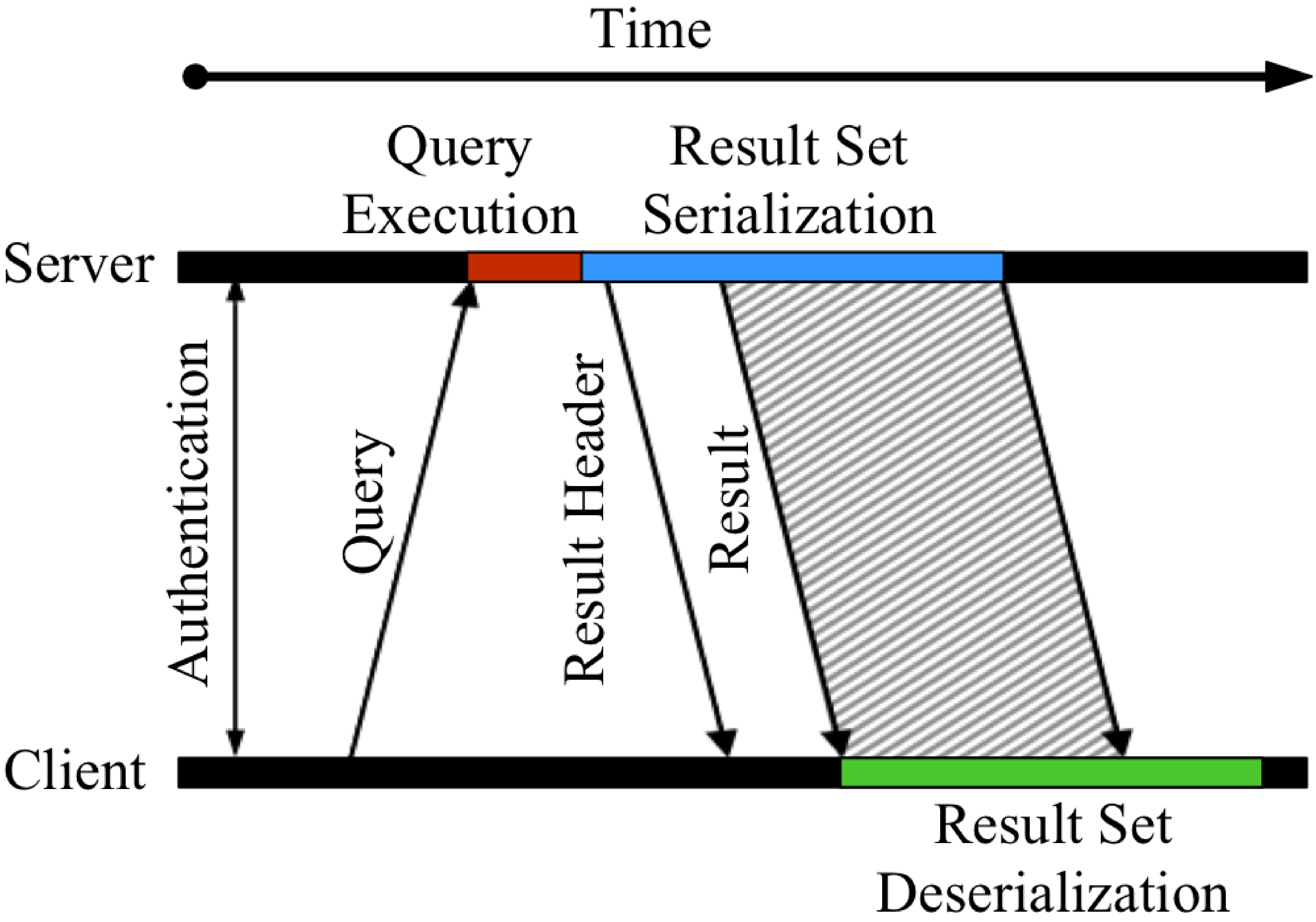
Don’t Hold My Data Hostage

A Case For Client Protocol Redesign

What is a Client Protocol anyway?

- Protocol is how a client communicates with a server
  - ODBC, JDBC, psql
- Every database that supports remote clients has a client protocol
- Using this protocol, clients can:
  - Connect to the database
  - Query it
  - Receive the query results

What is a Client Protocol anyway?

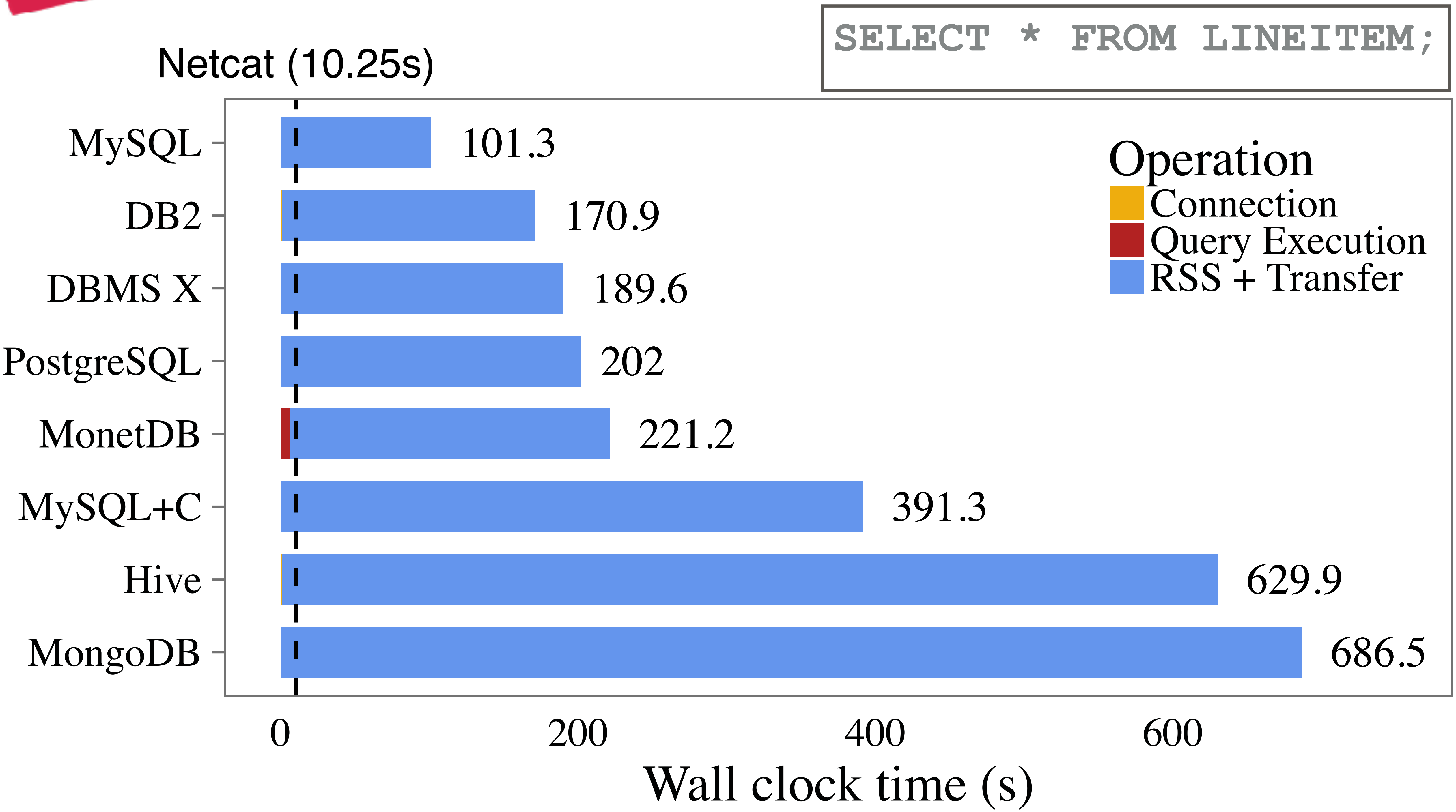


CWI

Motivation

- Problem: Current protocols were designed for exporting small amount of rows
  - Displaying results on screen
  - OLTP use cases
  - Exporting aggregates
- Exporting large amounts of data using these protocols is slow
  - Analytical tools (e.g. R/Python)
- Data export is a bottleneck when result sets are large!

Motivation



- Cost of exporting the SF10 lineitem table from TPC-H (7.2GB in CSV format) on localhost

Motivation

- We are not the first ones to notice this problem
- A lot of work on in-database processing, UDFs, etc.
- However, that work is database-specific, requires adapting of existing work flows and introduces safety issues

- Why is exporting large amounts of data from a database so inefficient?
- Can we make it more efficient?

State of the Art Protocols

- We reverse engineered how different databases transfer the following table “on the wire”
  - Source code/documentation
  - Decompilation of JDBC Drivers
  - Wireshark

INT32	VARCHAR10
100,000,000	OK
NULL	DPFKG

State of the Art Protocols

PostgreSQL

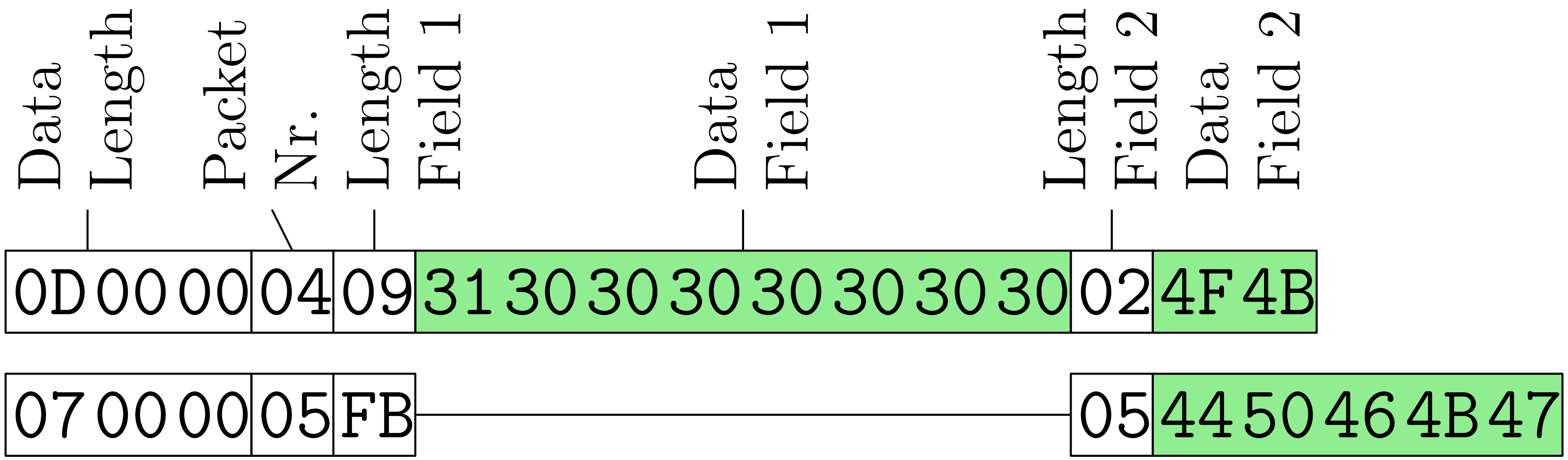
Message Type	Total Length	Field Count	Length Field 1	Data Field 1	Length Field 2	Data Field 2
44	00 00 00 10	00 02	00 00 00 04	05 F5 E1 00	00 00 00 02	4F 4B
44	00 00 00 0F	00 02	FF FF FF FF		00 00 00 05	44 50 46 4B 47

- Significant per-row overhead
- Used by many other systems:
  - Redshift, HyPer, Greenplum and Vertica

INT32	VARCHAR10
100,000,000	OK
NULL	DPFKG

State of the Art Protocols

MySQL



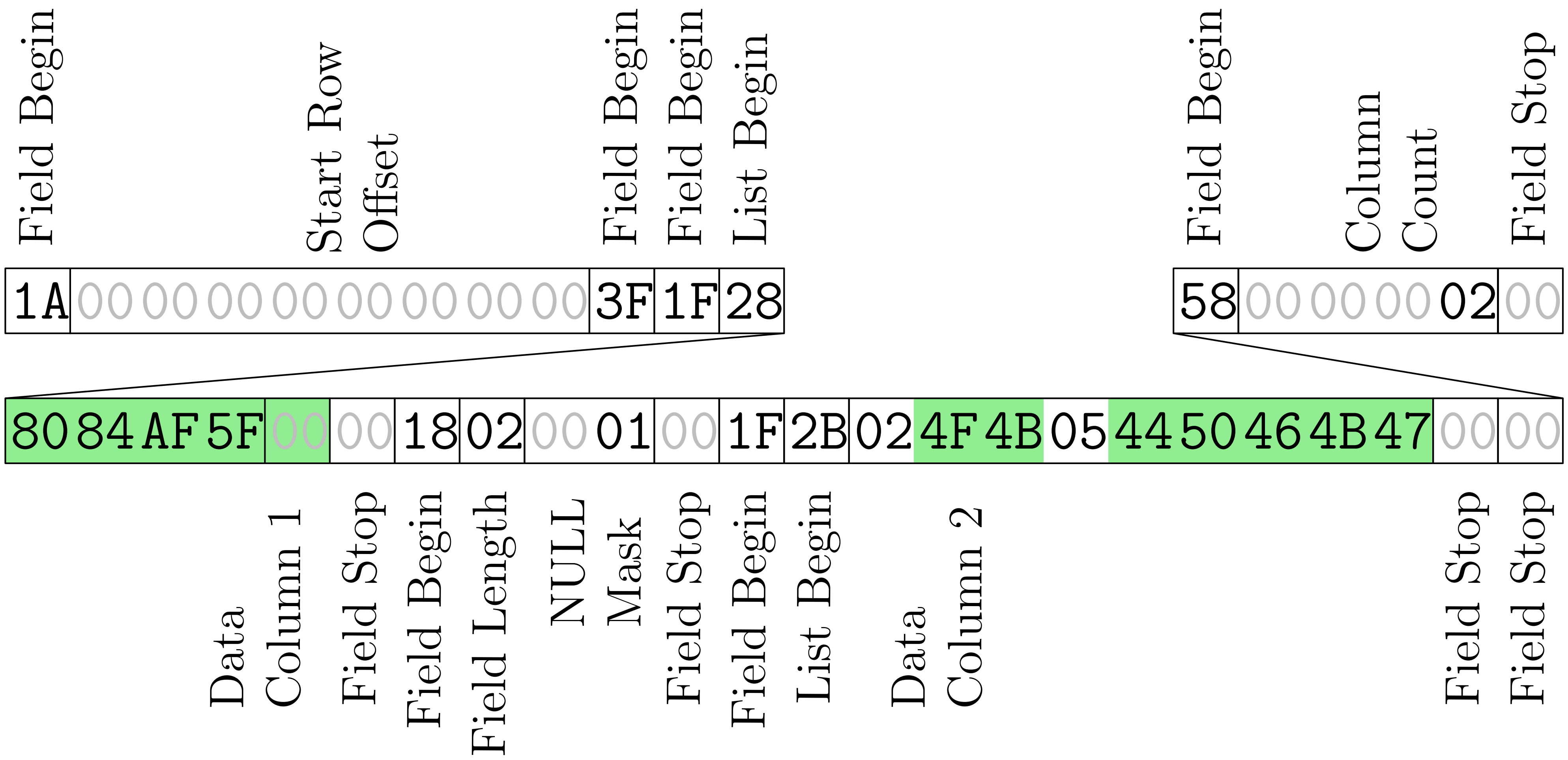
```
0D 00 00 04 09 31 30 30 30 30 30 30 30 02 4F 4B
07 00 00 05 FB                         05 44 50 46 4B 47
```

- ASCII protocol
- Every value has a length field
- Supports compression with GZIP

INT32	VARCHAR10
100,000,000	OK
NULL	DPFKG

State of the Art Protocols

Hive



Field Begin | Start Row Offset | Field Begin | Field Begin | List Begin

1A 00 00 00 00 00 00 00 00 3F 1F 28

Field Begin | Column Count | Field Stop

58 00 00 00 02 00

80 84 AF 5F 00 00 18 02 00 01 00 1F 2B 02 4F 4B 05 44 50 46 4B 47 00 00

Data Column 1 | Field Stop | Field Begin | Field Length | NULL Mask | Field Stop | Field Begin | List Begin | Data Column 2 | Field Stop | Field Stop

- Columnar protocol
- Uses generic Thrift serialisation library
- One byte per value in NULL mask
- Also used by SparkSQL

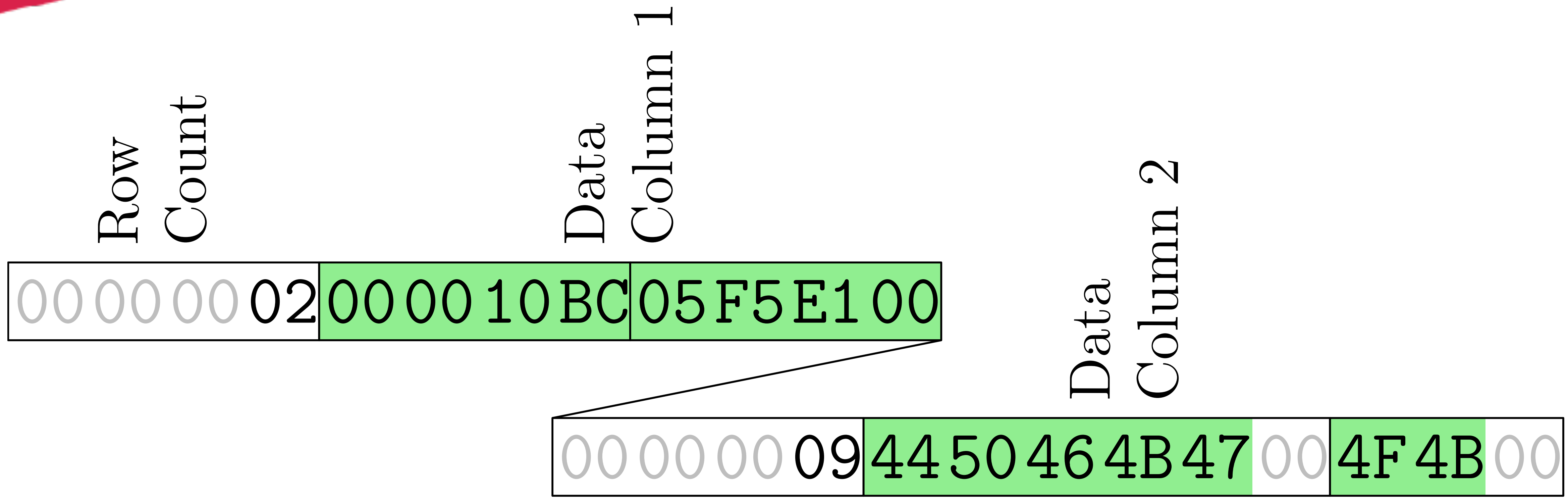
INT32	VARCHAR10
100,000,000	OK
NULL	DPFKG

Protocol Design Space

- Implemented prototype in PostgreSQL and MonetDB

- Serialisation Format (ASCII, Custom Binary, Generic)
  - **Custom Binary**
- Row Major or Column Major
  - **Column-Major (but chunked)**
- Data Compression Methods
  - **No compression local, stream compression on remote**
- Null Handling
  - **Close to native formats**

Proposed Protocol in MonetDB



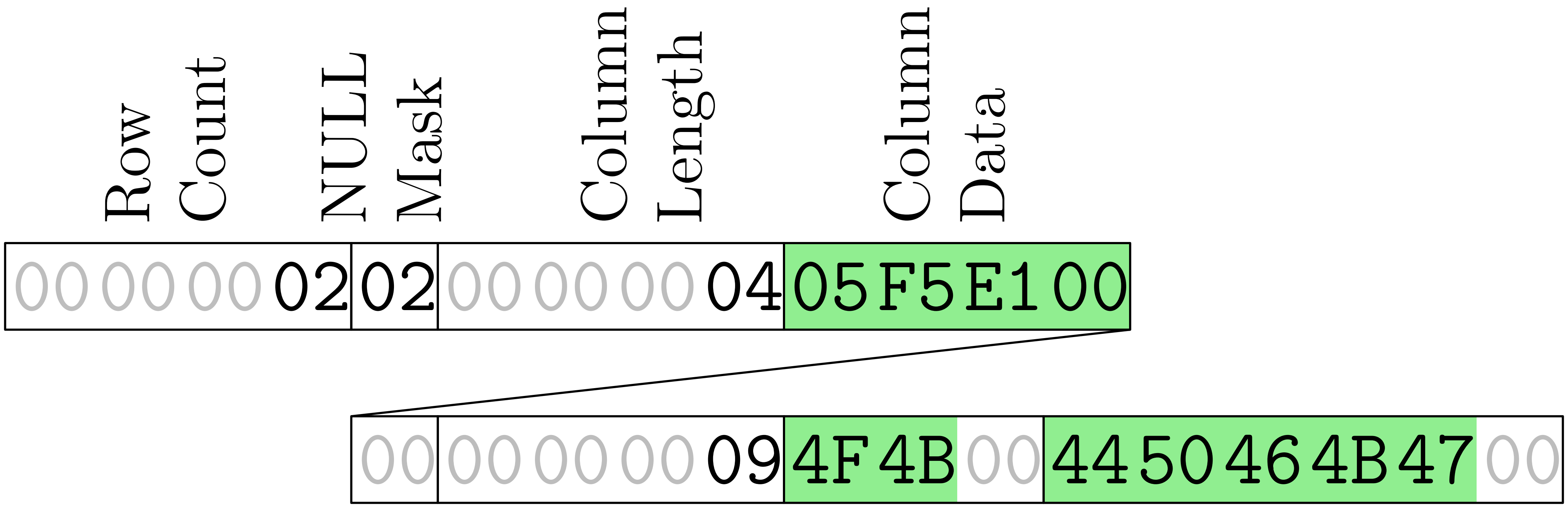
00 00 00 00 02 | 00 00 10 BC | 05 F5 E1 00

00 00 00 00 09 | 44 50 46 4B 47 00 | 4F 4B 00

- Columnar, 1MB chunks prefixed with row count
- Missing values stored as special values in domain
- Variable-length columns prefixed with their length

INT32	VARCHAR10
100,000,000	OK
NULL	DPFKG

Proposed Protocol in PostgreSQL



- Missing values (PostgreSQL)
- NULL bitmask for each column, 1 bit per value
- Only add mask if column has missing values
- Columns with mask have a column-length as well

INT32	VARCHAR10
100,000,000	OK
NULL	DPFKG

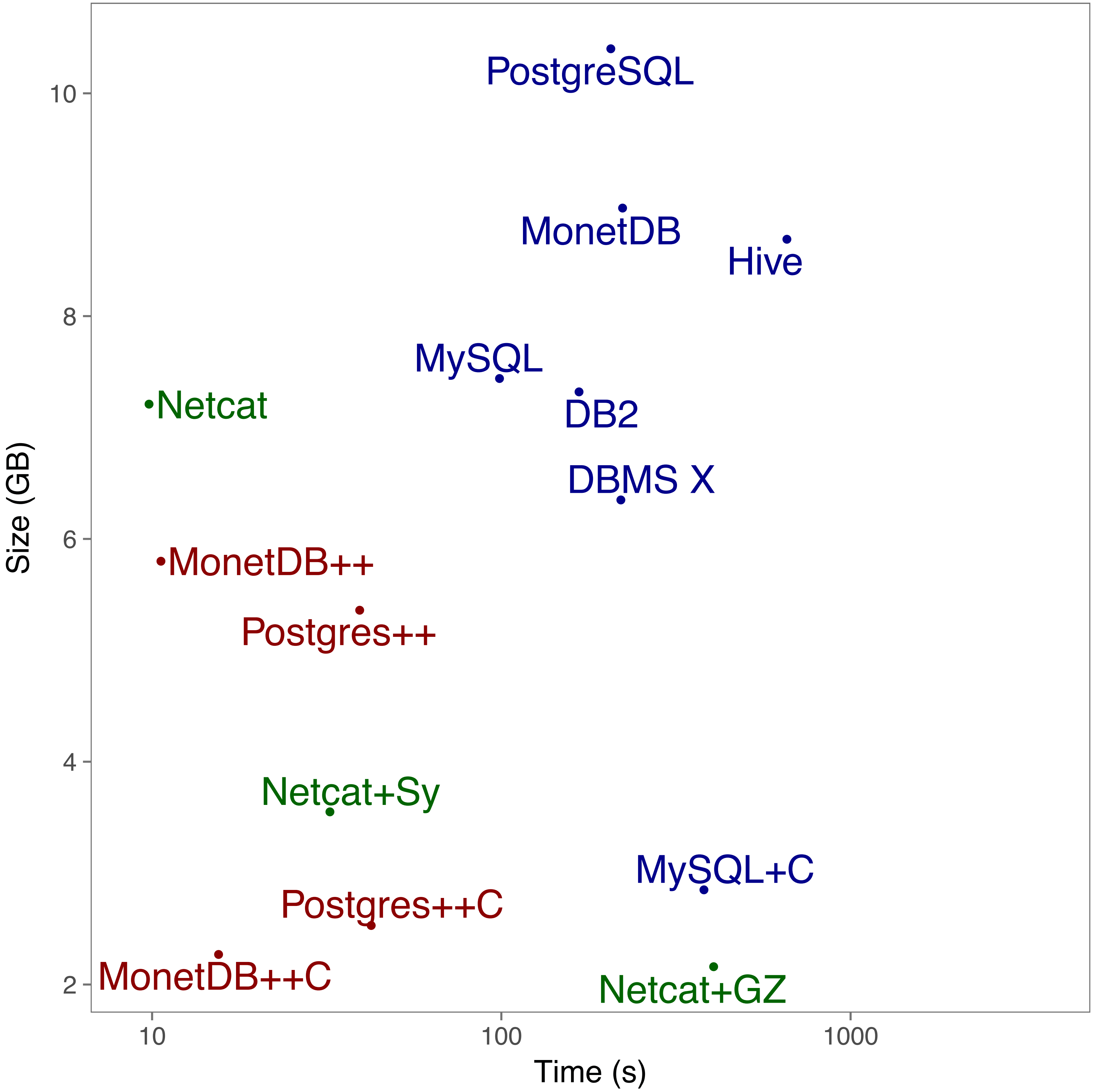
Benchmarks

- Three different network configurations
  - **Localhost:** No network restrictions
  - **LAN:** 1000 Mb/s throughput, 0.3ms latency
  - **WAN:** 100 Mb/s throughput, 25ms latency

  - **Lineitem:** SF10, 60 million rows, 16 columns, 7.2GB in CSV format
  - 1 hour timeout

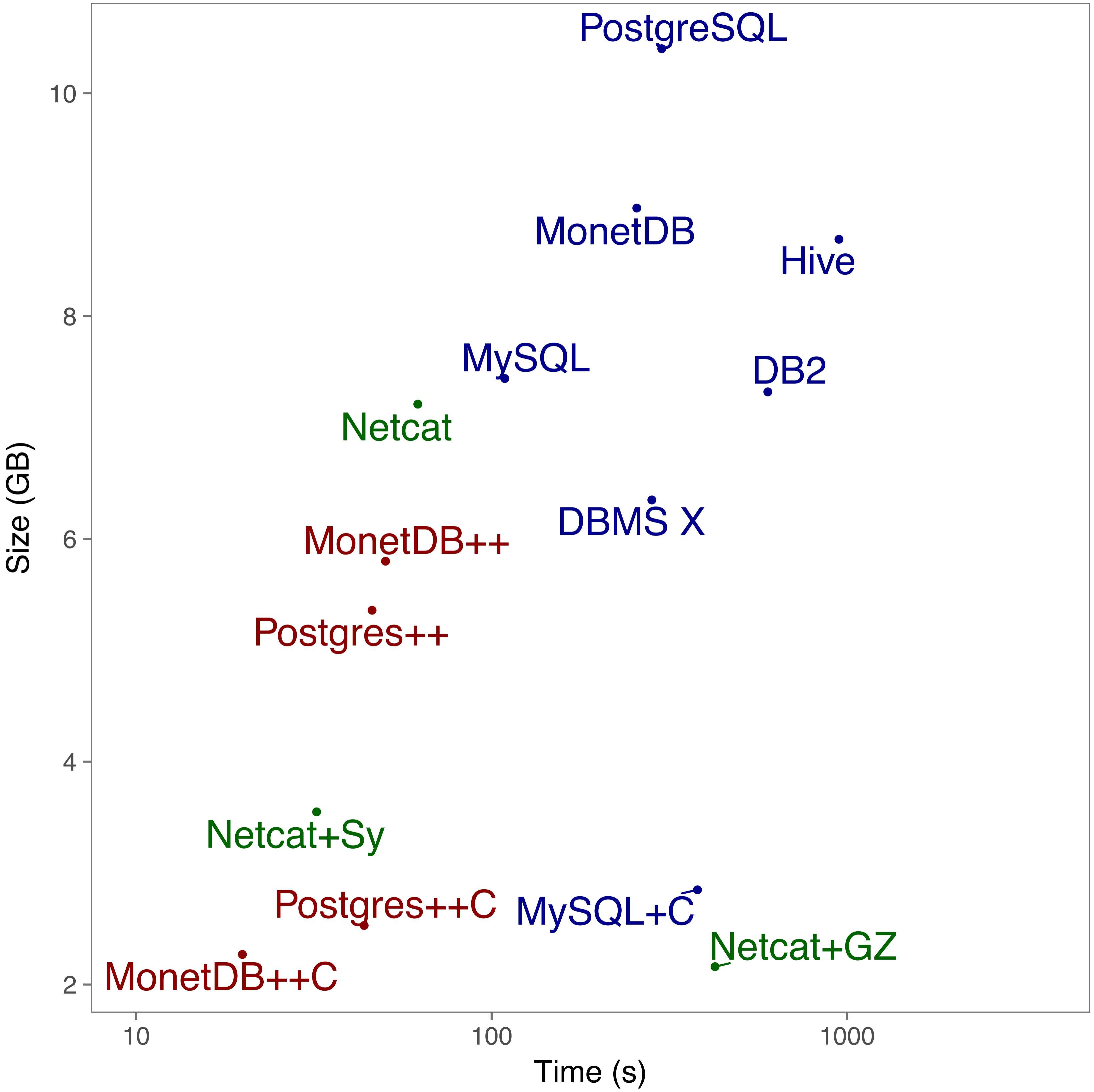
Benchmarks

Localhost (No network restrictions)



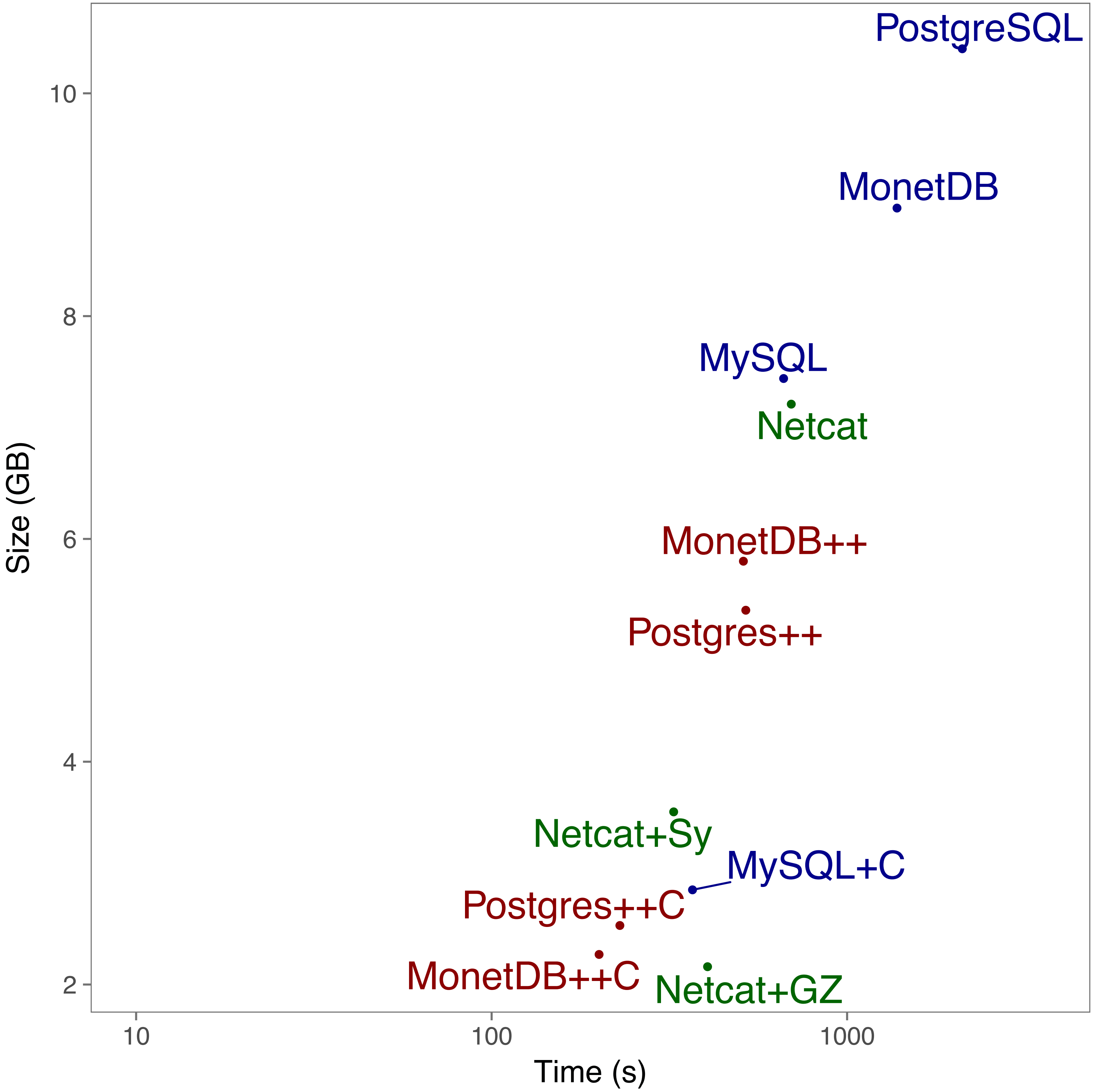
Benchmarks

LAN (1000 Mb/s throughput, 0.3ms latency)



Benchmarks

WAN (100 Mb/s throughput, 25ms latency)



Conclusions

- Exporting data from a database does not have to be so inefficient
- State of the art database protocols can be improved for this use case
- We show this by implementing prototypes in two databases (MonetDB and PostgreSQL)
  - Avoid per-row overhead, bulk transfer
  - Stay close to database native formats
  - Avoid unnecessary copying and conversion
  - Lightweight compression on remote

- MonetDB implementation is already released.
- Benchmark information: https://goo.gl/usjfyJ