CWI

Mark Raasveldt, Hannes Mühleisen

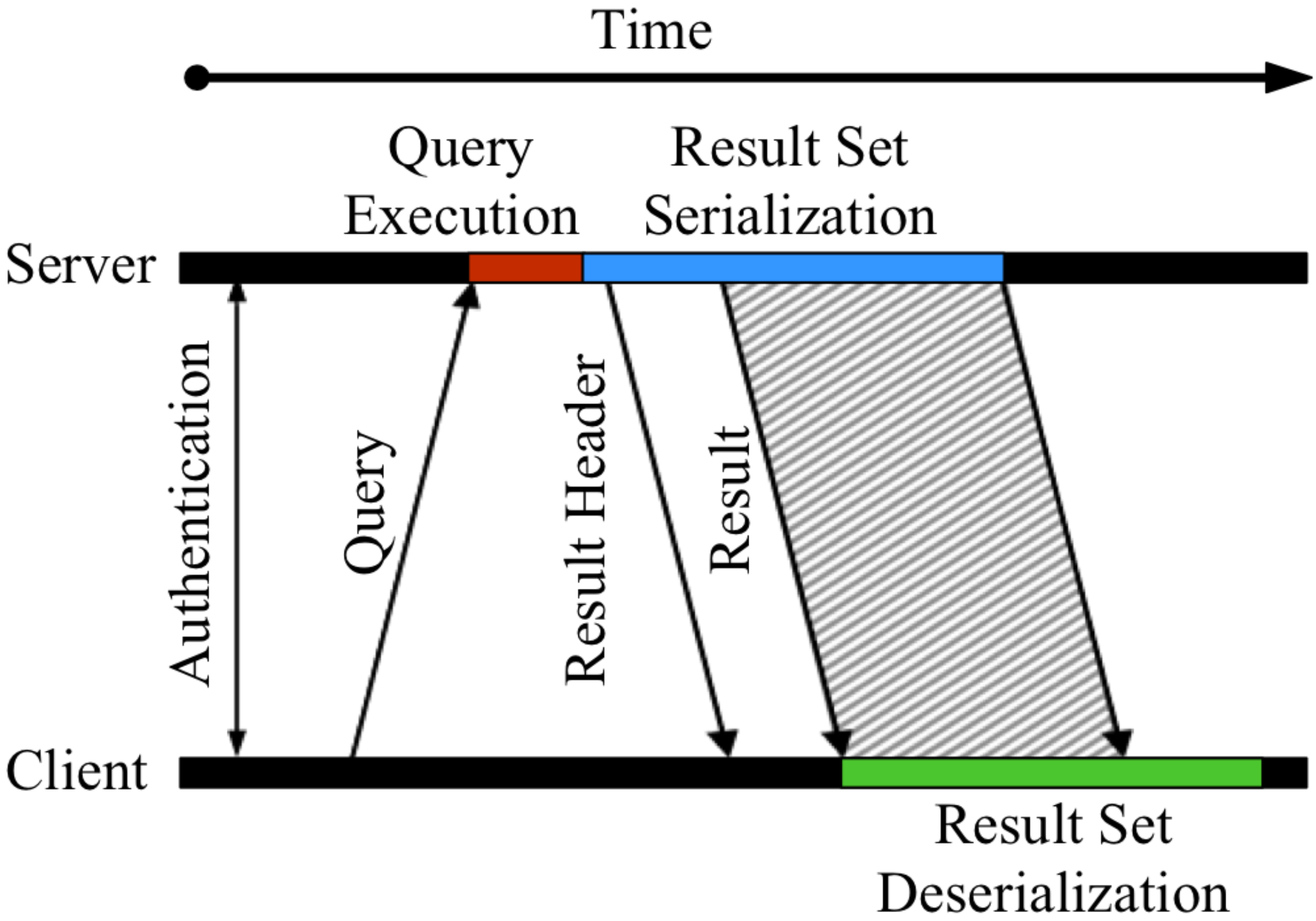
Don’t Hold My Data Hostage

A Case For Client Protocol Redesign

What is a Client Protocol anyway?

- Every database that supports remote clients has a client protocol
- Using this protocol, clients can query the database
- In response to a query, the server computes a result
- Then the result is transferred back to the client

What is a Client Protocol anyway?



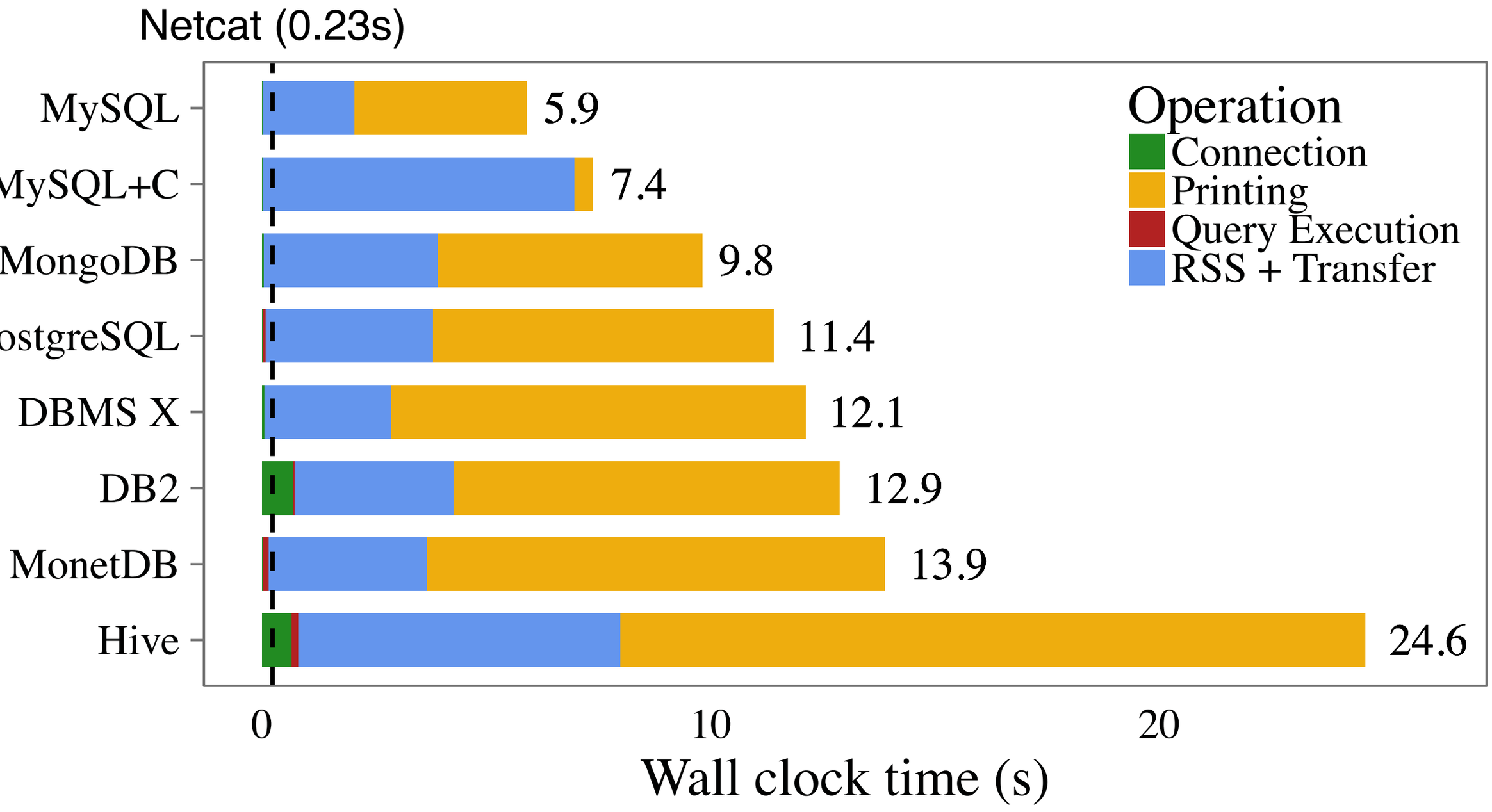
Motivation

- Traditionally, client protocols were mainly used for printing output to a console
  - Console clients (psql, mclient)
- Currently, many clients actually want to use and analyze the data
  - External analysis tools (R/Python)
  - Visualisation tools (Tableau)

Motivation

- Problem: Current protocols were designed for exporting small amount of rows
  - OLTP use cases
  - Exporting aggregations
- Exporting large amounts of data using these protocols is slow

Motivation



- Cost of exporting 1M rows of the lineitem table from TPC-H (120MB 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 and requires adapting of existing work flows

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

Cost of Data Export

- We don’t care about printing and connection costs
- What about result set (de)serialization + transfer?

System	Time (s)	Size (MB)
(Netcat)	(0.23)	(120.0)
MySQL	**2.04**	127.0
DBMS X	2.82	127.1
MonetDB	3.53	150.2
DB2	3.53	154.6
PostgreSQL	3.74	195.4
MongoDB	3.88	365.8
MySQL+C	6.95	**48.2**
Hive	7.19	148.5

Cost of Data Export

- Result Set Serialisation
  - Compression, data conversions, endianness swaps, copying data into a buffer
- Data Transfer Time
- Result Set Deserialization
  - (De)compression, data parsing, endianness swaps

State of the Art Protocols

- Why do these protocols exhibit this behaviour?
- Let’s take a look at this simple table serialised using different databases’ result set serialisation formats.

INT32	VARCHAR10
42	DPFKG
100,000,000	OK

Table 1: Simple result set table.

State of the Art Protocols

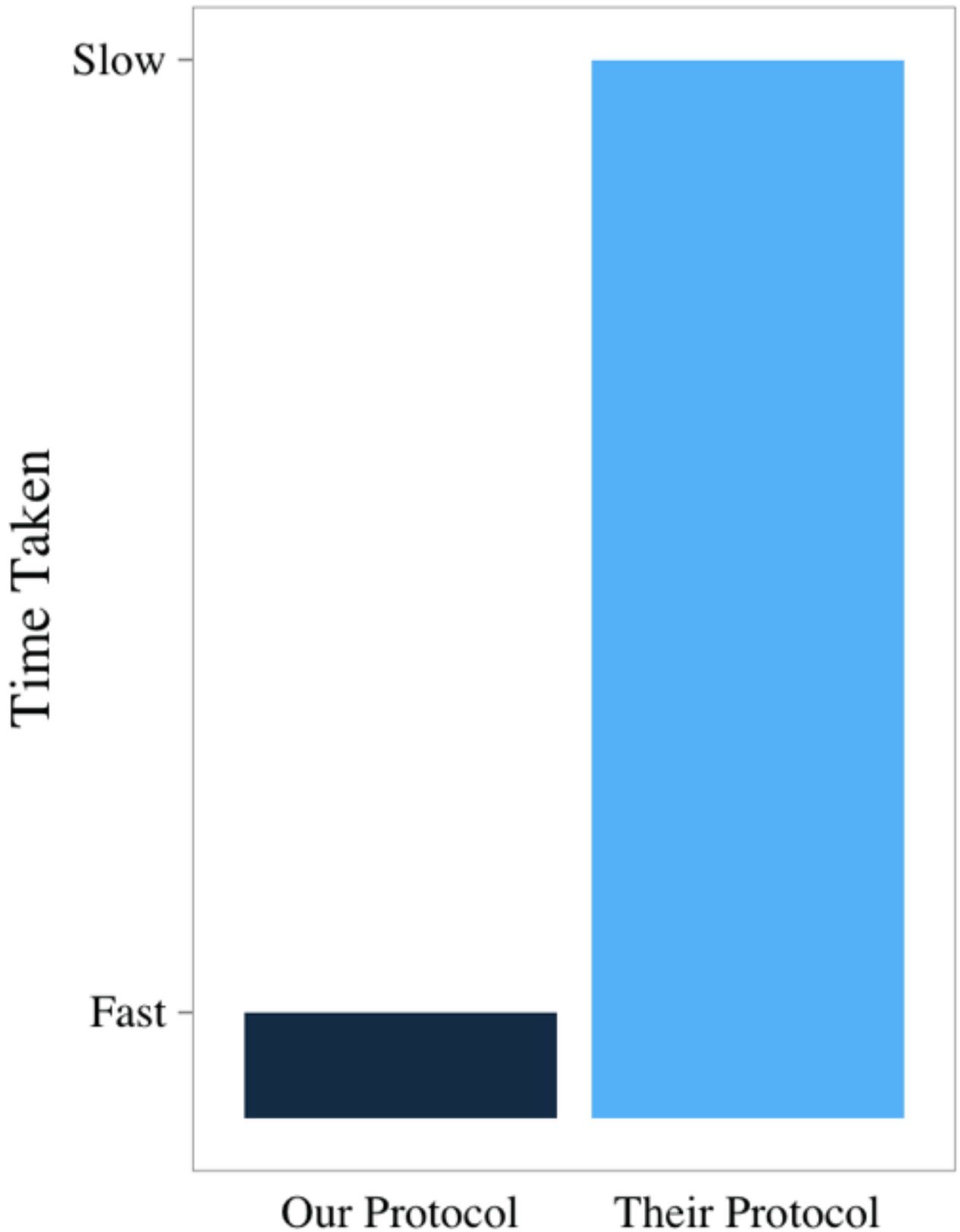
Message Type	Length	Field Count	Length Field 1	Data Field 1	Length Field 2	Data Field 2
44	00000017	0002	00000004	000010BC	00000005	4450464B47
44	00000014	0002	00000004	05F5E100	00000002	4F4B

- PostgreSQL serialisation of the previous table

Protocol Implementation

- We implemented our own protocol
  - In MonetDB
  - In PostgreSQL
- Without per-row overhead
- With efficient compression techniques

Benchmark Results



Conclusion

- Current protocols are not suited for large result set export
- This leads to large result set export being a bottleneck
- We show there is room for improvement by implementing our own protocol that is an order of magnitude faster