CWI

Mark Raasveldt, Hannes Mühleisen

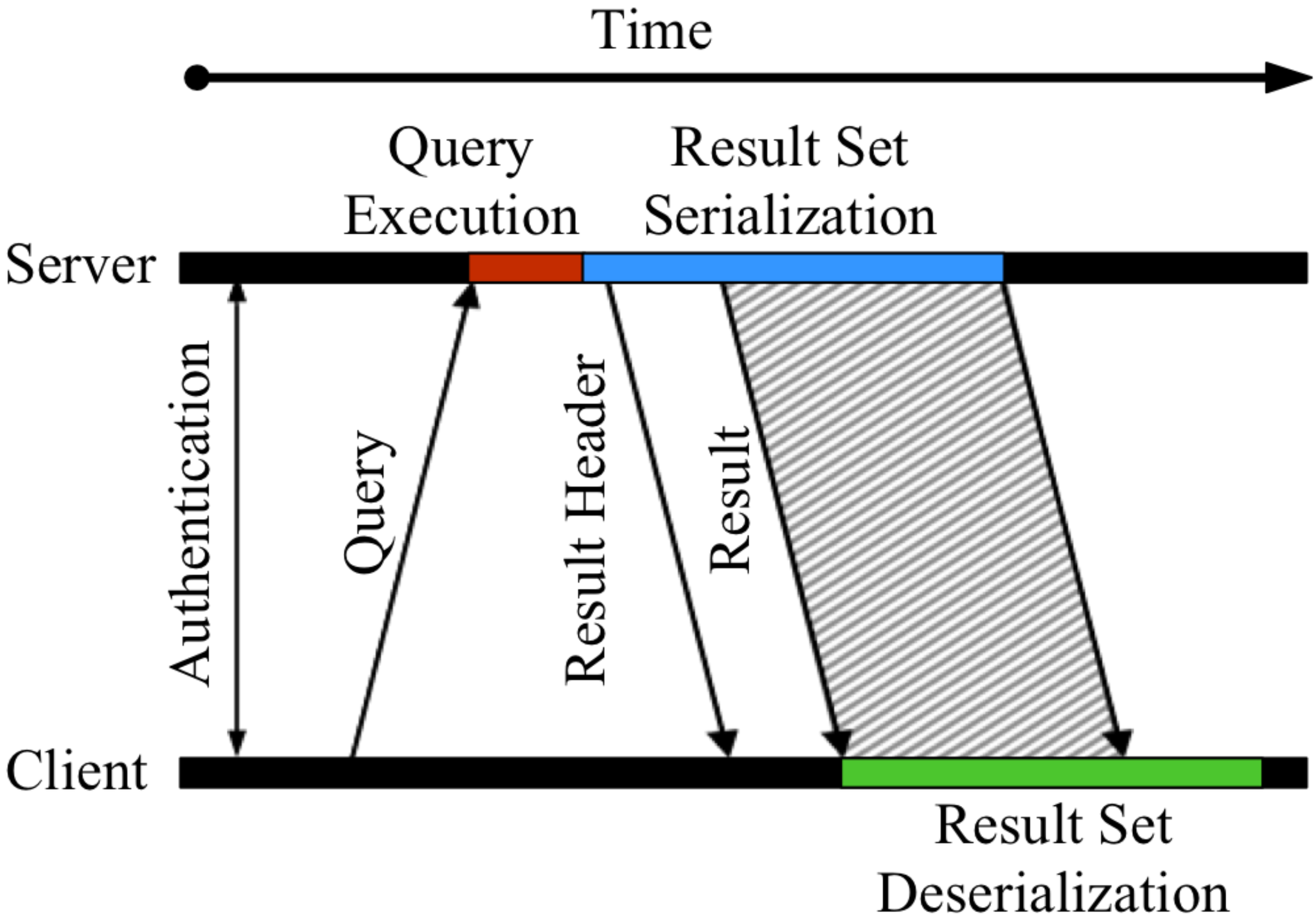
Don’t Hold My Data Hostage

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?



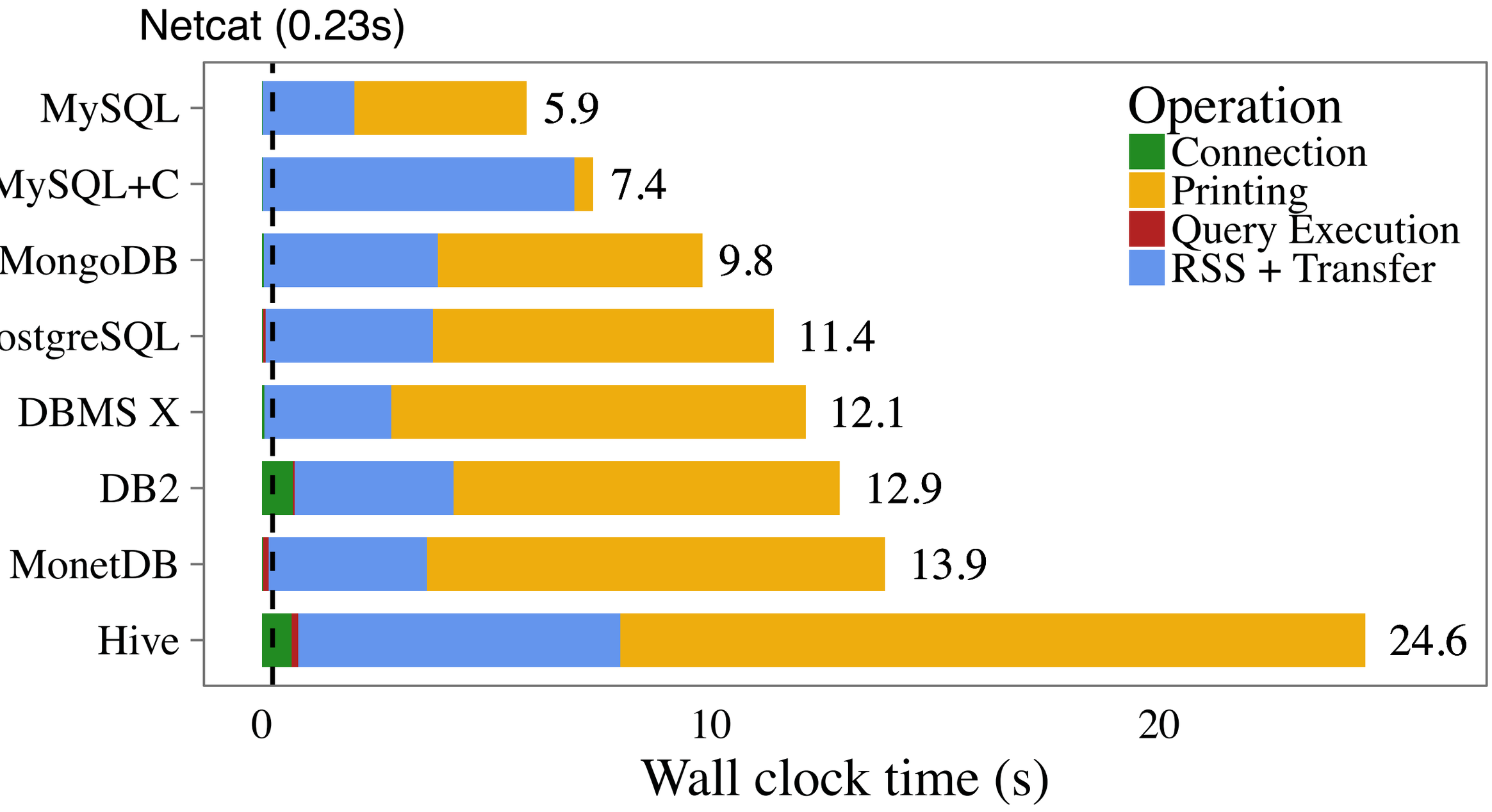
Motivation

- 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

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

Cost of Data Export

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

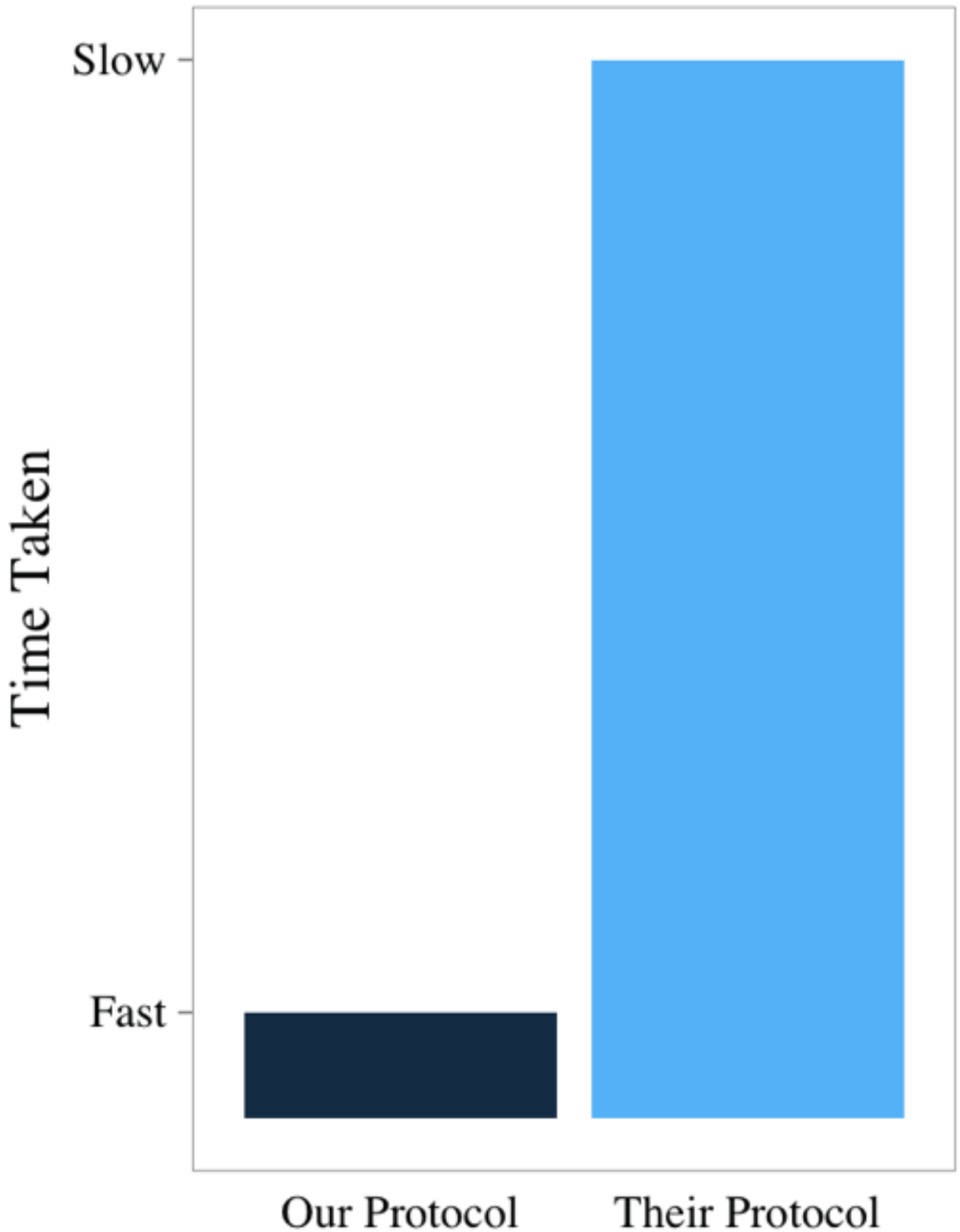
Protocol Implementation

- Main ideas
- Columnar result set format
  - Per-column overhead instead of per-row or per-value
  - Better compressibility
- Compression depending on network limitations
- Specialised column-wise compression techniques
- Avoid endianness swaps and data conversions
- Avoid per-row and per-value function calls

Benchmark Results

- We implemented our own protocol
  - In the column-store MonetDB
  - In the row-store PostgreSQL

Benchmark Results



Benchmark Results

	System	Timings (s) T_{Local}	T_{LAN}	T_{WAN}	Size
Lineitem	(Netcat)	(2.6)	(10.2)	(112.0)	(1211.3)
	(Netcat+Sy)	(5.1)	(5.5)	(52.9)	(595.8)
	(Netcat+GZ)	(69.2)	(70.7)	(69.4)	(361.1)
	MonetDB++	**1.7**	8.4	84.4	990.8
	MonetDB++C	3.3	**3.5**	**32.3**	**381.7**
	PostgreSQL++	3.6	7.7	85.1	914.9
	PostgreSQL++C	5.5	5.8	34.0	395.8
	MySQL	22.5	22.8	107.4	1279.5
	MySQL+C	75.3	84.2	86.0	482.5
	PostgreSQL	40.7	46.6	326.2	1966.2
	DB2	35.9	123.9	1451.2	1545.3
	DBMS X	32.3	46.4	691.8	1255.0
	Hive	78.3	118.7	717.0	1484.8
	MongoDB	48.2	61.3	458.2	3681.8