CWI

Centrum Wiskunde & Informatica

COMMIT/

Efficient Data Management and Statistics with Zero-Copy Integration

Jonathan Lajus & Hannes Mühleisen

SSDBM 2014, 2014-06-30

Collect data

Bottleneck, thanks David!

Statistical Toolkit

Load data

Filter, transform & aggregate data

Analyze & Plot

Data Management System

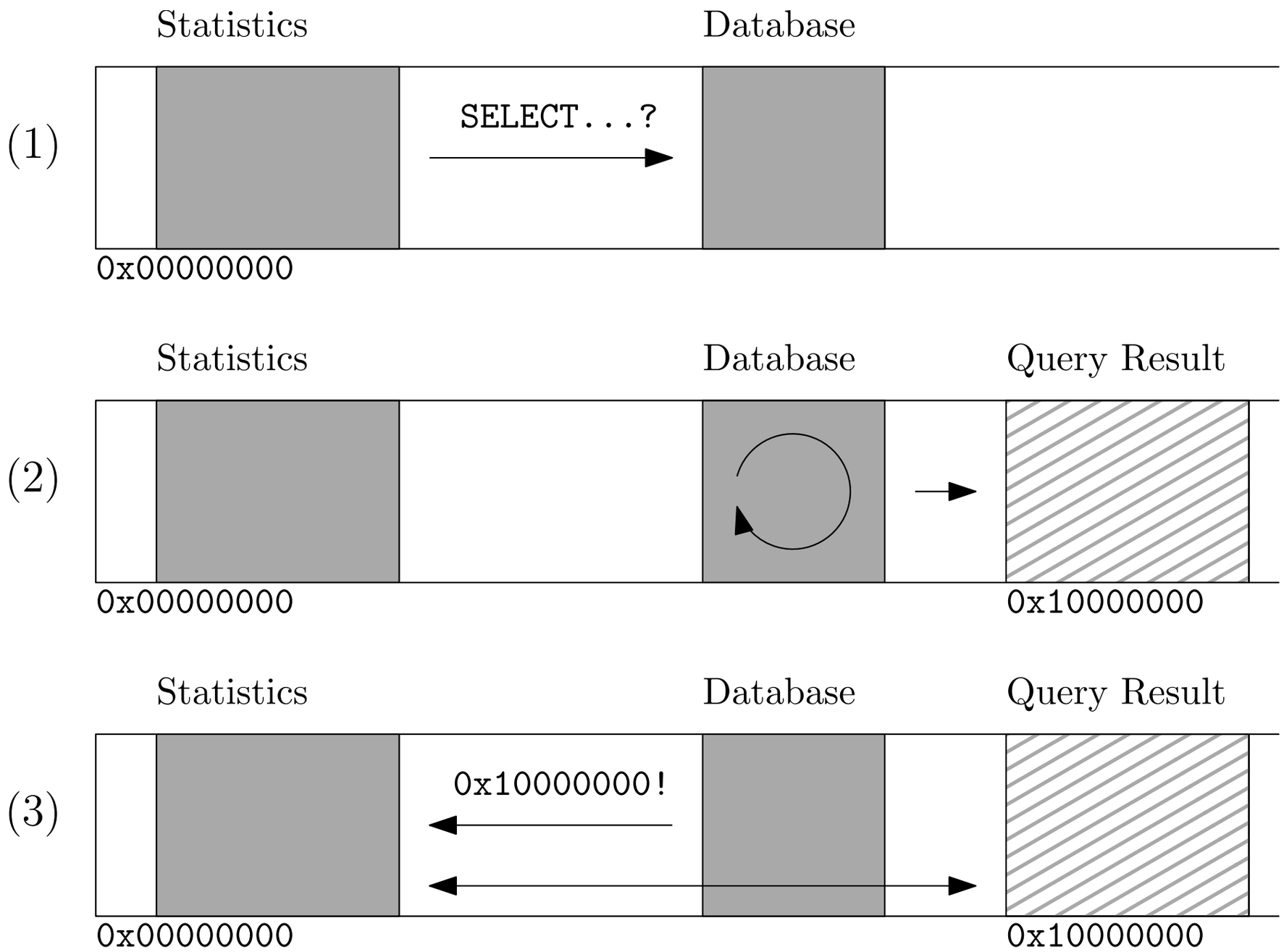
Publish paper/ Profit

Data Transfer Options

- “Socket-Style”, JDBC, ODBC, DBI, …
  - Serialization, copy, copy, copy, Deserialization
- In-process embedding
  - No sockets (hopefully), but still conversion
- Shared memory
- No transfer altogether by extending DB or stats

Zero-Copy

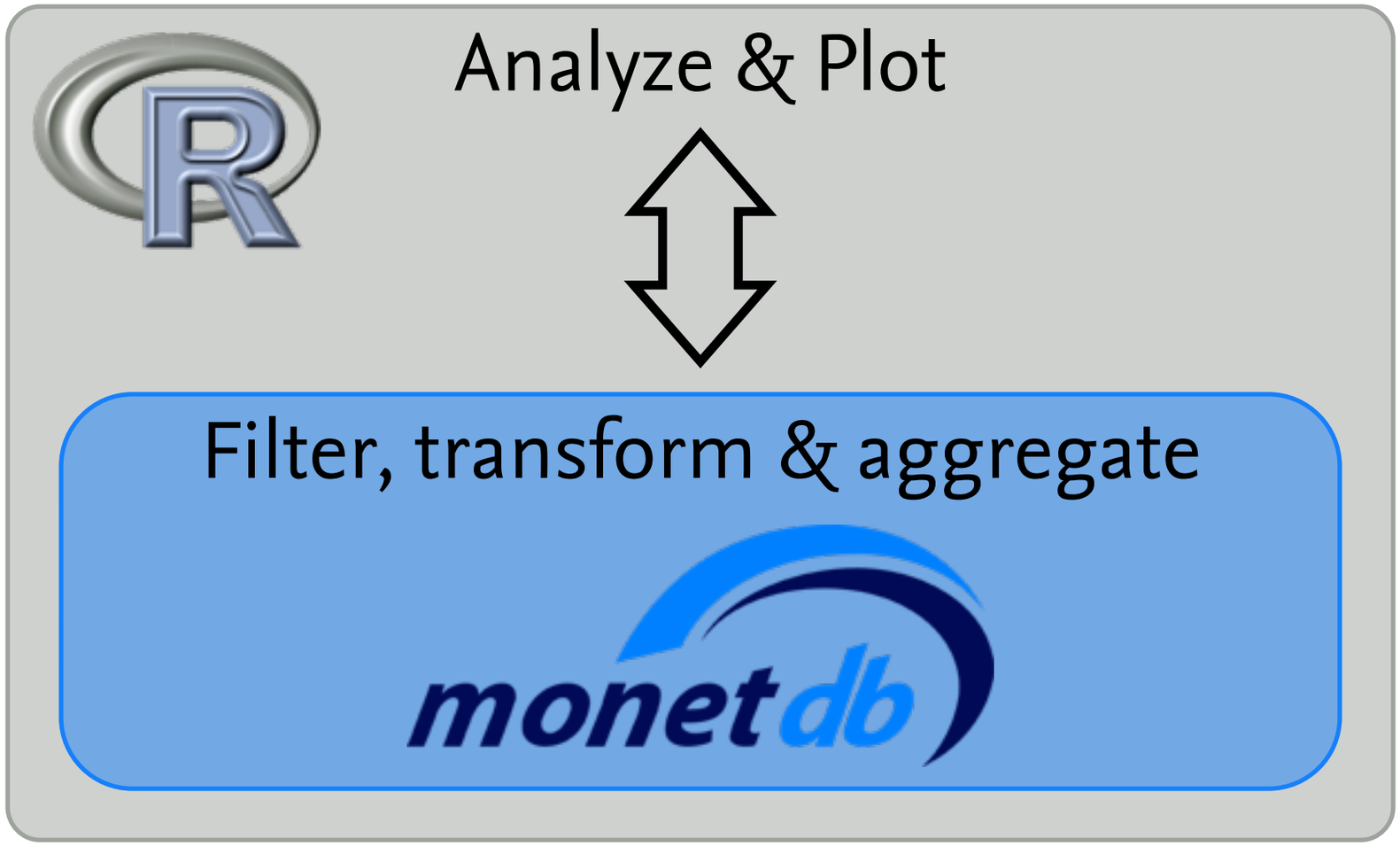
- In the end, C-arrays of native types are everywhere
  - Hardware does not like Java objects (usually)
- Hmm…
- In-process data sharing by passing pointers
  - If both systems are based on C arrays, we could get away with metadata management


(1)
Statistics
Database
SELECT...?
0x00000000
(2)
Statistics
Database
Query Result
0x00000000
0x10000000
(3)
Statistics
Database
Query Result
0x10000000!
0x00000000
0x10000000

Challenges

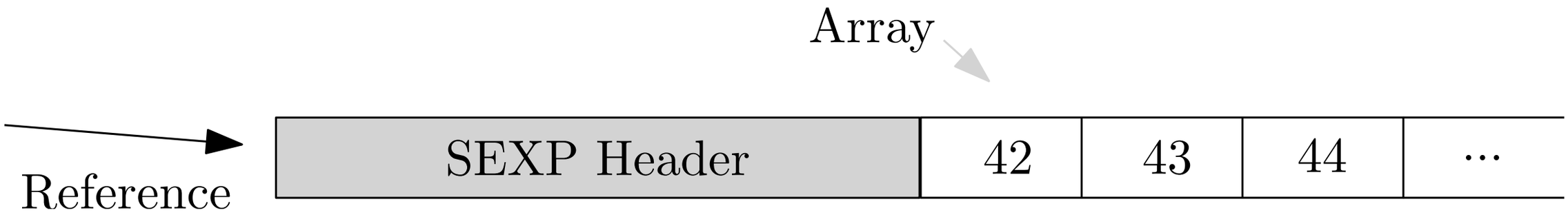
- Compilation & Linking
  - symbol name clashes are likely
- Read/Write synchronization
- Memory management (who calls free())
- NA/NULL value encoding?
- Complex Objects

Proof-of-Concept

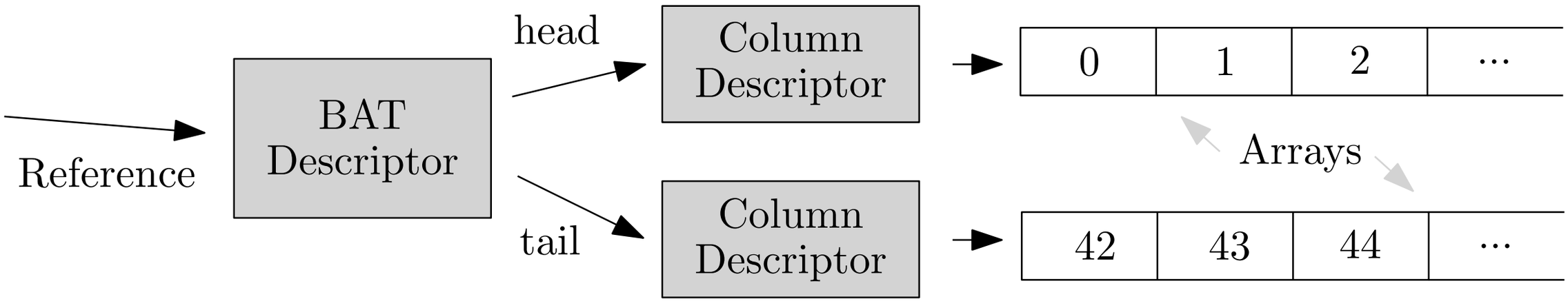


https://github.com/lajus/monetinr

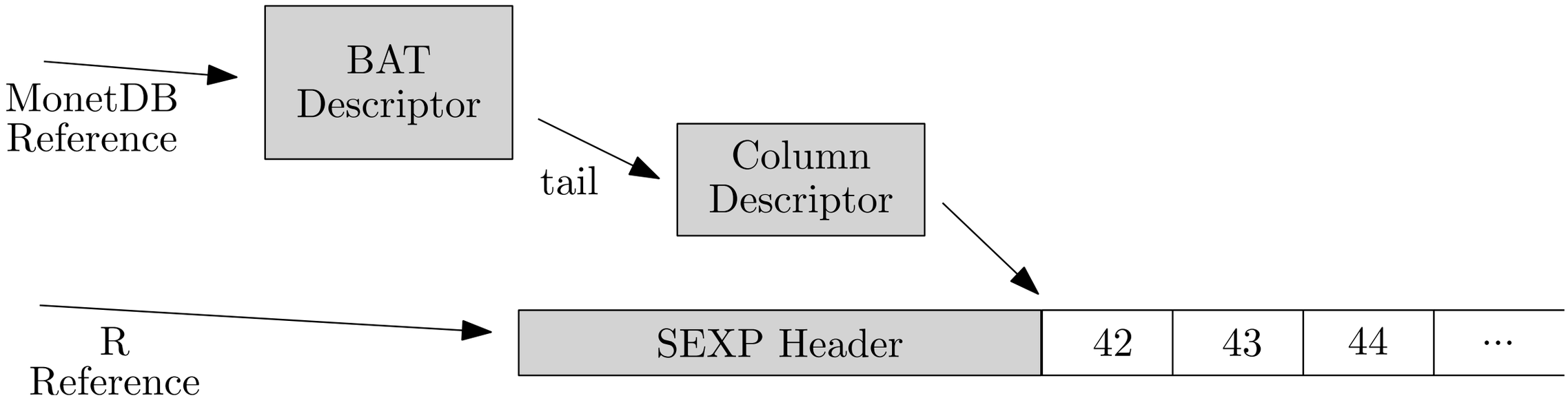
R SEXP



MonetDB BAT



Dress-up



+ Garbage Collection Fun

Experiments

- `data.table`, high-performance R data access
- `MonetDB.R`, DBI/socket-based DB access
    - Equvalent systems, different connection
- `RSQLite`, embedded SQL database
    - Still needs conversion

Predictions

- Dedicated data management systems should be better at data management than pimped stats tools
- In-process integration should make a big difference once result sets get large
- Column store performance gain should be visible

Setup

- Typical data management tasks
  - Selection & Projection
  - Aggregation
  - Joins
- 10MB, 100MB, 1GB, 10GB datasets
- Desktop-class machine, 16 GB RAM, 3.4 Ghz i7, Fedora Linux

1% Rows Selected

10% Rows Selected

50% Rows Selected

10min

1min

10s

1s

100ms

10ms

Execution Time (log)

Beats Sockets & Stats Extensions

data.table

RSQLite

MonetDB.R

Prototype

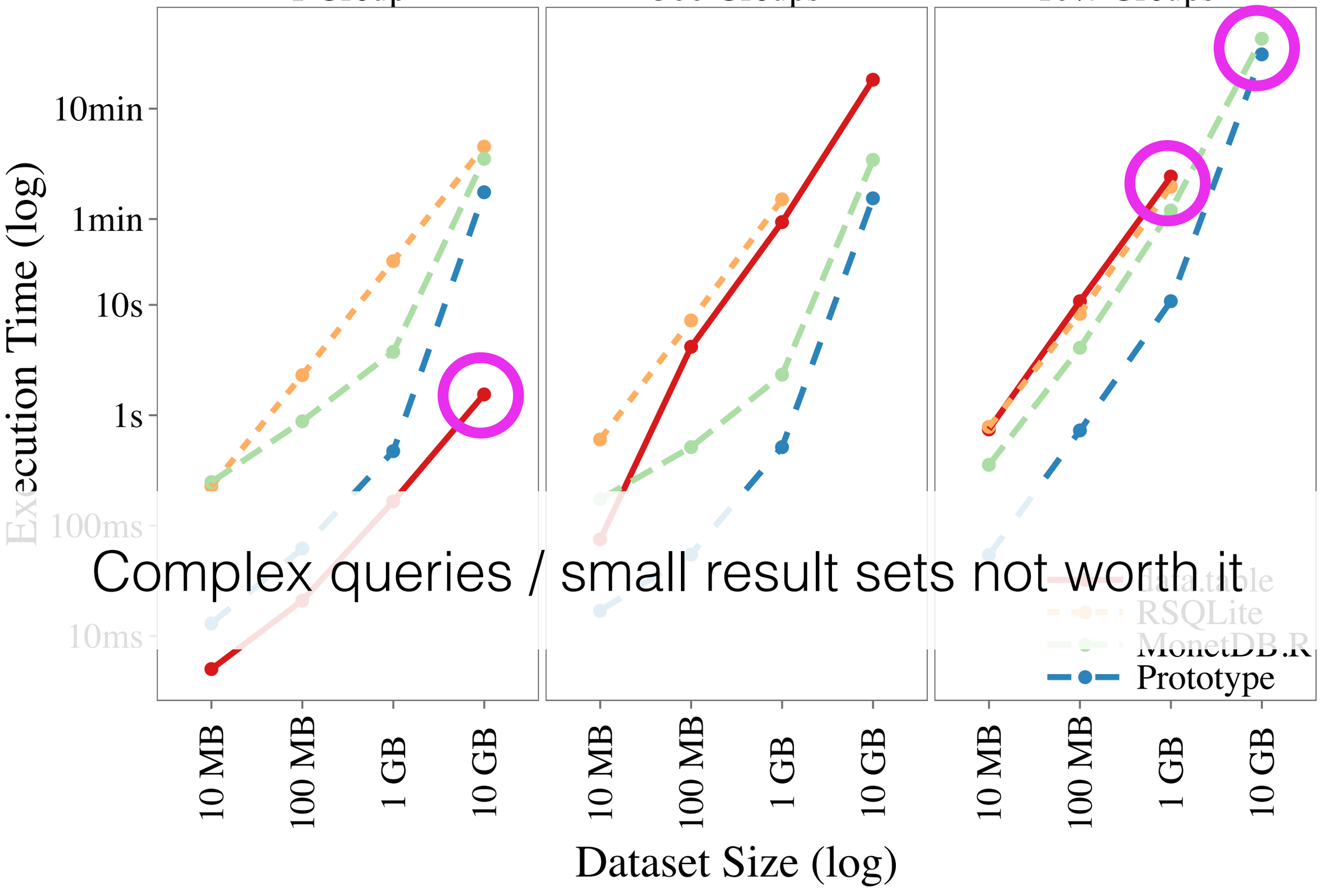
10 MB

100 MB

1 GB

10 GB

Dataset Size (log)


1 Group
500 Groups
10% Groups
Execution Time (log)
10min
1min
10s
1s
100ms
10ms
Complex queries / small result sets not worth it
data.table
RSQLite
MonetDB.R
Prototype
10 MB
100 MB
1 GB
10 GB
Dataset Size (log)

1% Join Partner Size

10% Join Partner Size

Execution Time (log)

10min
1min
10s
1s
100ms
10ms

10 MB
100 MB
1 GB
10 GB

Dataset Size (log)

data.table
RSQLite
MonetDB.R
Prototype

Conclusions

- Zero-Copy possible
- Vast performance benefits
- But
  - Read/Write access?
  - Iterative processes?
  - Optimization?

Special Thanks

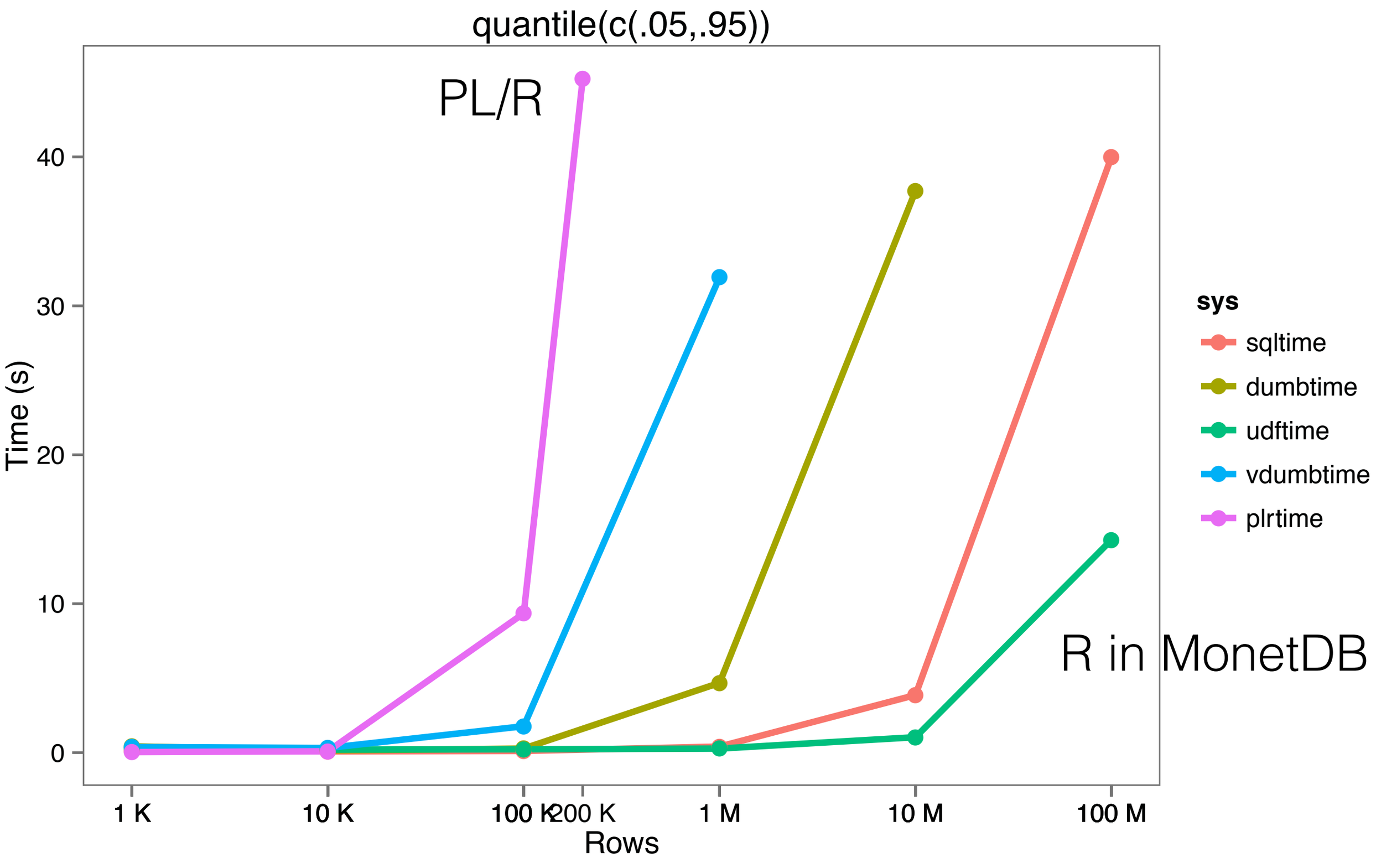
- Thomas Lumley (R)

- Sjoerd Mullender (MonetDB)

R UDFs in MonetDB

```
CREATE FUNCTION kmeans (data FLOAT,
   ncluster INTEGER) RETURNS INTEGER
LANGUAGE R { kmeans(data,ncluster)$cluster };
```

Watch the next MonetDB release…


quantile(c(.05,.95))
PL/R
R in MonetDB
Time (s)
0
10
20
30
40
1 K
10 K
100 K
200 K
1 M
10 M
100 M
Rows
sys
sqltime
dumbtime
udftime
vdumbtime
plrtime

Thank You!

Questions?