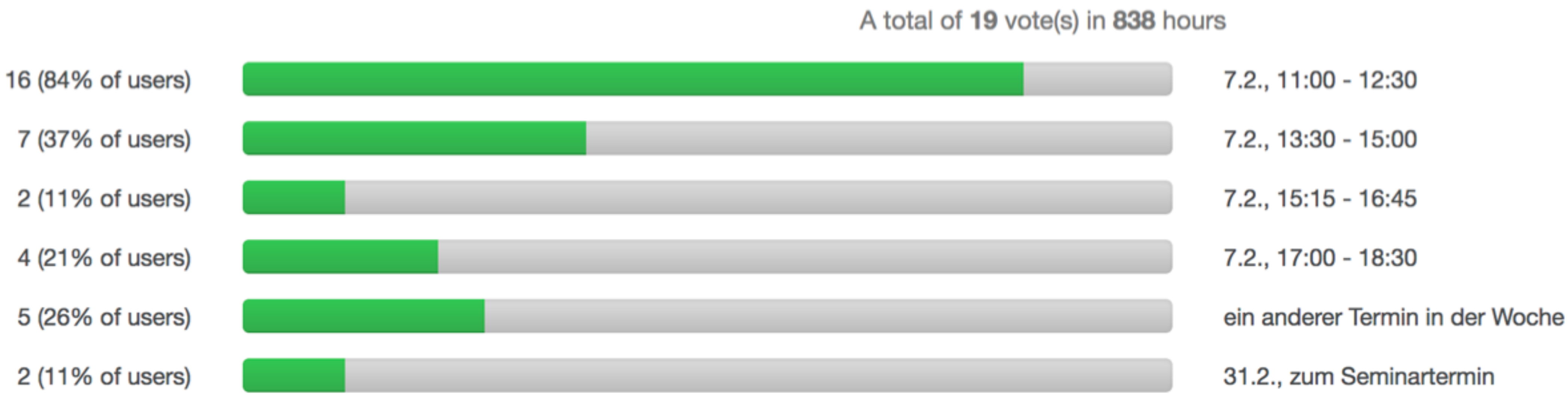
Termin Endpräsentation

- 16 / 19 können um 11:00 – 12:00, also machen wir es so



Running Order

- Networking
- Small String Optimization
- Partitioning
- Optimizer Rules
- Pruning Filters
- Subqueries
- Self-Driving
- NUMA-Optimized Join

Hyrise as a Server

10.01.2018

Lawrence, Stephan, Robert

Supervisor: Stefan Klauck

Task

```
> ./hyriseServer 5432
Server running...
```

Start Hyrise as a Server application

Use any existing Postgres client and execute queries

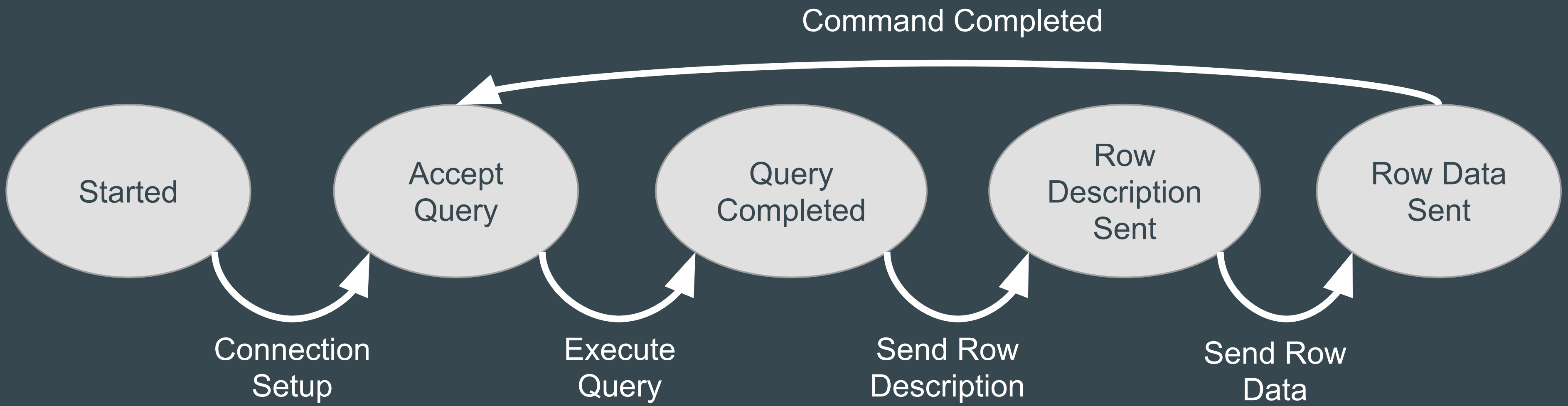
```
> psql -h 123.456.1.1 -p 5432
=> SELECT * FROM foo;
 a
---
123
(1 row)
```

Work Done so Far

- `SELECT * FROM foo;`
- Async request handling
- Basic tests

Async Request Handling

- Handle network connections concurrently from one thread
- `boost::asio::io_service` dispatches methods



Loopback: lo0

ip.addr == 127.0.0.1 and (tcp.port == 5432)

No.	Time	Source	Destination	Protocol	Length	Info
189…	338482.4244…	127.0.0.1	127.0.0.1	PGSQL	188	<T
189…	338482.4244…	127.0.0.1	127.0.0.1	TCP	56	64462 → 5432 [ACK] Seq=116 Ack=149 Win=408128 Len=0 TSval=1333342053 TSecr=1333342053
189…	338482.4245…	127.0.0.1	127.0.0.1	PGSQL	144	<D
189…	338482.4245…	127.0.0.1	127.0.0.1	TCP	56	64462 → 5432 [ACK] Seq=116 Ack=237 Win=408064 Len=0 TSval=1333342053 TSecr=1333342053
189…	338482.4246…	127.0.0.1	127.0.0.1	PGSQL	70	<C
189…	338482.4246…	127.0.0.1	127.0.0.1	TCP	56	64462 → 5432 [ACK] Seq=116 Ack=251 Win=408032 Len=0 TSval=1333342053 TSecr=1333342053
189…	338482.4246…	127.0.0.1	127.0.0.1	PGSQL	62	<Z
189…	338482.4246…	127.0.0.1	127.0.0.1	TCP	56	64462 → 5432 [ACK] Seq=116 Ack=257 Win=408032 Len=0 TSval=1333342053 TSecr=1333342053
189…	338491.3556…	127.0.0.1	127.0.0.1	PGSQL	94	>Q
189…	338491.3557…	127.0.0.1	127.0.0.1	TCP	56	5432 → 64462 [ACK] Seq=257 Ack=154 Win=408128 Len=0 TSval=1333350941 TSecr=1333350941
189…	338491.4004…	127.0.0.1	127.0.0.1	PGSQL	88	<T
189…	338491.4005…	127.0.0.1	127.0.0.1	TCP	56	64462 → 5432 [ACK] Seq=154 Ack=289 Win=408000 Len=0 TSval=1333350985 TSecr=1333350985
189…	338491.4005…	127.0.0.1	127.0.0.1	PGSQL	68	<D
189…	338491.4006…	127.0.0.1	127.0.0.1	TCP	56	64462 → 5432 [ACK] Seq=154 Ack=301 Win=408000 Len=0 TSval=1333350985 TSecr=1333350985
189…	338491.4006…	127.0.0.1	127.0.0.1	PGSQL	70	<C
189…	338491.4006…	127.0.0.1	127.0.0.1	TCP	56	64462 → 5432 [ACK] Seq=154 Ack=315 Win=407968 Len=0 TSval=1333350985 TSecr=1333350985
189…	338491.4006…	127.0.0.1	127.0.0.1	PGSQL	62	<Z

```
▶ Frame 18913: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 5432, Dst Port: 64462, Seq: 257, Ack: 154, Len: 32
▼ PostgreSQL
    Type: Row description
    Length: 31
  ▼ Field count: 1
    ▼ Column name: I_NAME
        Table OID: 0
        Column index: 0
        Type OID: 25
        Column length: -1
        Type modifier: -1
        Format: Text (0)
```

```
0000  02 00 00 00 45 02 00 54  00 00 40 00 40 06 00 00   ....E..T ..@.@...
0010  7f 00 00 01 7f 00 00 01  15 38 fb ce 43 64 d3 4e   ........ .8..Cd.N
0020  bc e3 35 93 80 18 31 d2  fe 48 00 00 01 01 08 0a   ..5...1. .H......
0030  4f 79 52 49 4f 79 52 1d  54 00 00 00 1f 00 01 49   OyRIOyR. T......I
0040  5f 4e 41 4d 45 00 00 00  00 00 00 00 00 00 00 19   _NAME... ........
0050  ff ff ff ff ff ff 00 00                            ........
```

Work to come

- Full query support
  - Currently only SELECT works
- Benchmarking
  - Send timing information back to client
  - Analyze performance (e.g. with pgbench)
- Optimizing #packets sent
- Testing (unit / end2end)
- Bug fixing
- Submit PR

Small String Optimization

Develop your own database

Benjamin Feldmann, Marcel Jankrift, Toni Stachewicz
Advisor: Jan Kossmann
Hasso Plattner Institute

Problem

- (Enterprise) data usually contains many short strings
- `std::string` uses small-string-optimization
  - msvc: 0 - 15 byte strings
  - gcc >= 5: 0 - 15 byte strings
  - clang: 0 - 22 byte strings
- Longer strings
  - Compiler allocates memory on heap
  - Additional indirection
  - msvc and clang allocate more memory than the string size

Example

- Extract of SAP ACDOCA table

MATNR	SEGMENT
P-100	MANF
P-1	MANF
P-1234	MANF

Layout of vector:

MATNR:	P	-	1	0	0										P	-	1	…
SEGMENT:	M	A	N	F											M	A	N	…

Our Solution

- Define maximum string length

- Store all strings in one vector of chars
  - Each string has maximum length

Our Solution: Example

- MATNR string length: 6
- SEGMENT string length: 4

Layout of vector:

MATNR:	P	-	1	0	0	\0	P	-	1	\0	\0	\0	P	-	1	2	3	4
SEGMENT:	M	A	N	F	M	A	N	F	M	A	N	F						

Drawbacks

- Only one (or a few) long strings
  - More memory consumption for smaller strings

ExampleColumn
ab
cd
this_is_a_very_very_very_very_very_very_very_very_very_very_very_long_string
123
B
m

ValueVector Implementation

- New class for:
  - DictionaryColumn :: _dictionary
  - ValueColumn :: _values

- ValueVector can store values in std::vector<std::string|int|float|...>
- or strings with a fixed length in std::vector<char>

ValueVector Implementation

```
template <typename T>
class ValueVector {
 public:
  explicit ValueVector(uint8_t fixed_string_length);

```

- Has std::vector functions
- Additional constructor for fixed string length

Partitioning

Felix Musmann, Jonas Chromik, Niklas Hoffmann
Winter Term 2017 / 2018
DYOD

Partitioning

= Dividing a table in multiple parts

➚ Performance gain for **queries**
partitions not matching the criteria can be pruned

➙ Manageability
➙ Availability
➙ Load-balancing

Vertical Partitioning

Name	Birthday
Neo	1970-01-01
Morpheus	1955-12-11
Trinity	1976-06-07
Smith	1970-01-01

City	Country
Oslo	Norway
Murmansk	Russia
Darwin	Australia
Yokohama	Japan

similar to database normalization, but on physical level

Horizontal Partitioning

Name	Birthday	City	Country
Neo	1970-01-01	Oslo	Norway
Morpheus	1955-12-11	Murmansk	Russia

Trinity	1976-06-07	Darwin	Australia
Smith	1970-01-01	Yokohama	Japan

take partitioning key and assign data to partition based on some criteria

Horizontal Partitioning: Partitioning Criteria

Problem: How to determine which tuple resides in which partition?

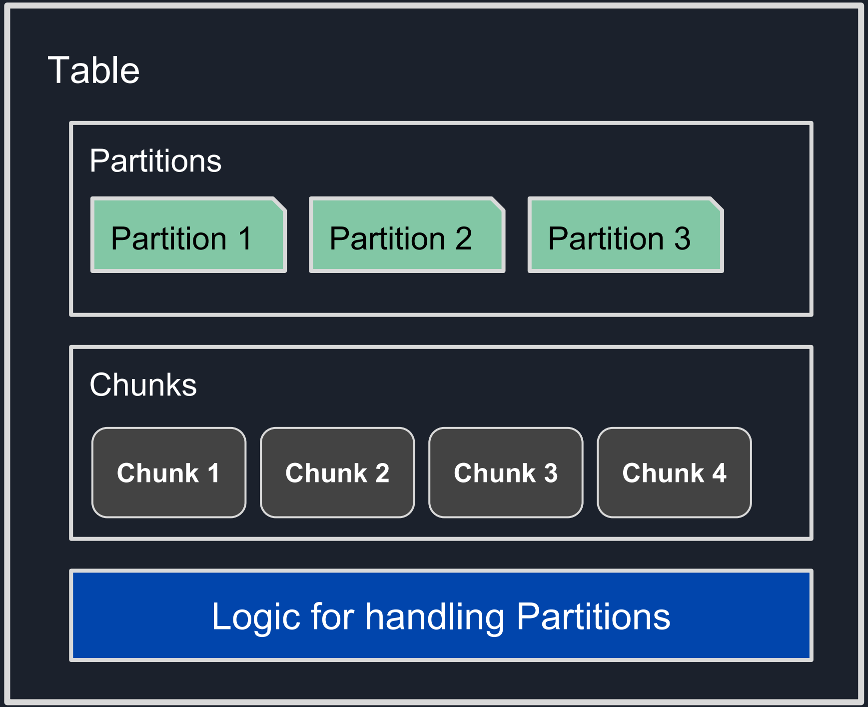
Approaches:

- Range-based
- Hash-based
- Round Robin
- Tuple matching some predicate
- Time-based
- List-based

How to build this?

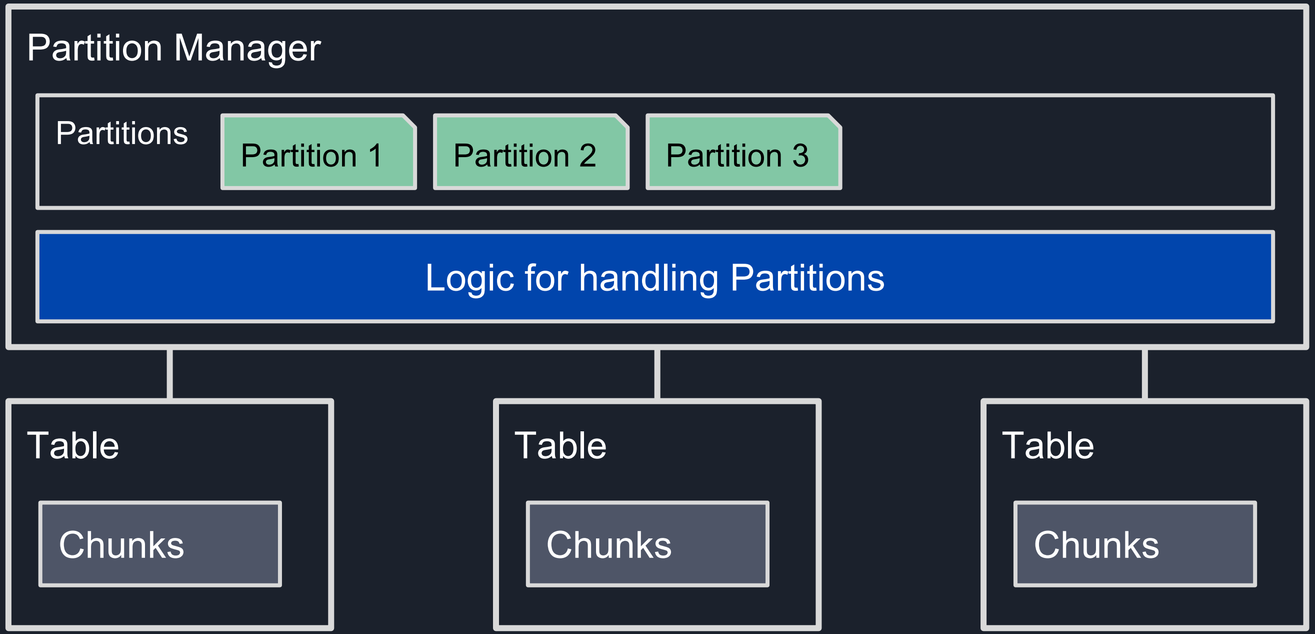
How to handle partition management?

Alternative #1



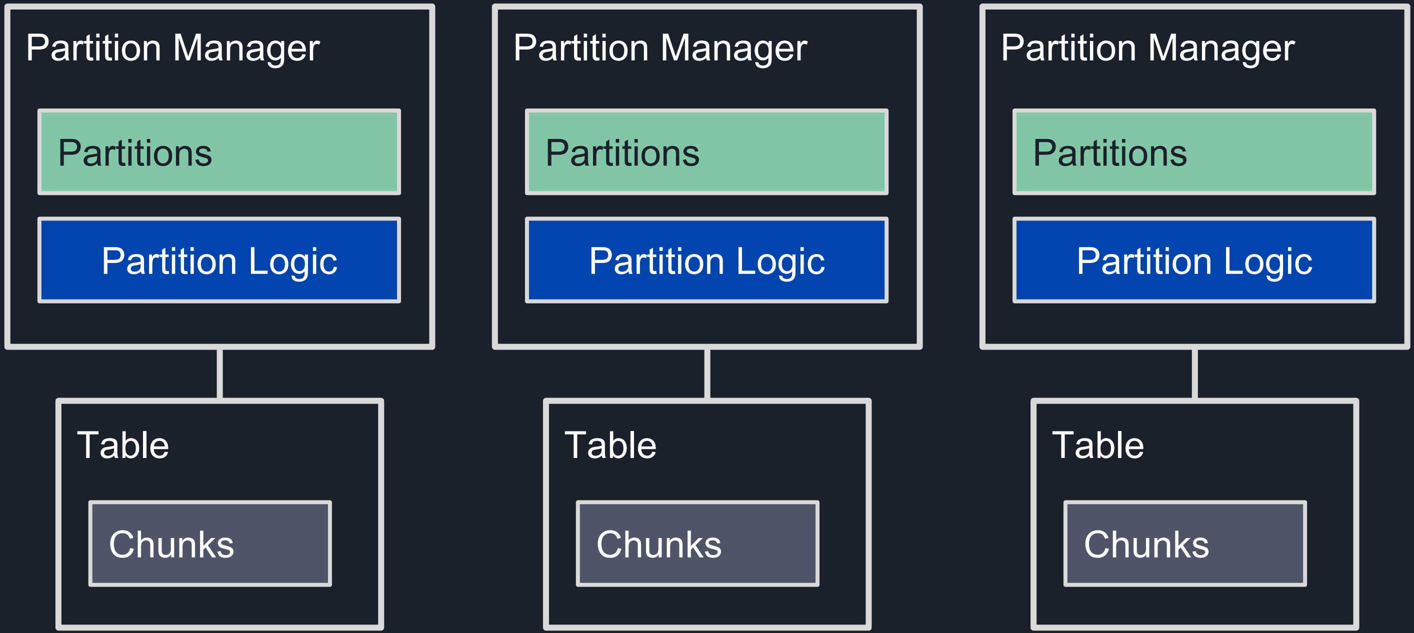
How to handle partition management?

Alternative #2



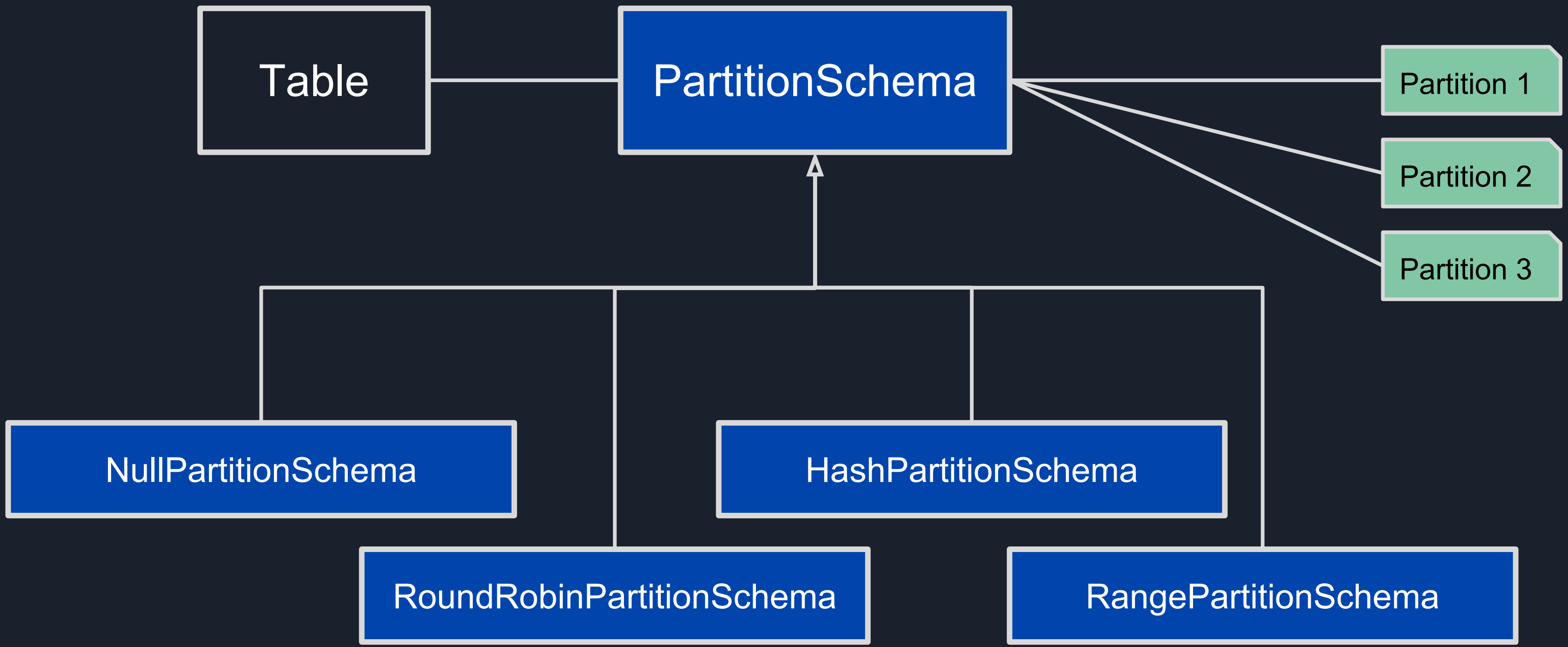
How to handle partition management?

Alternative #3

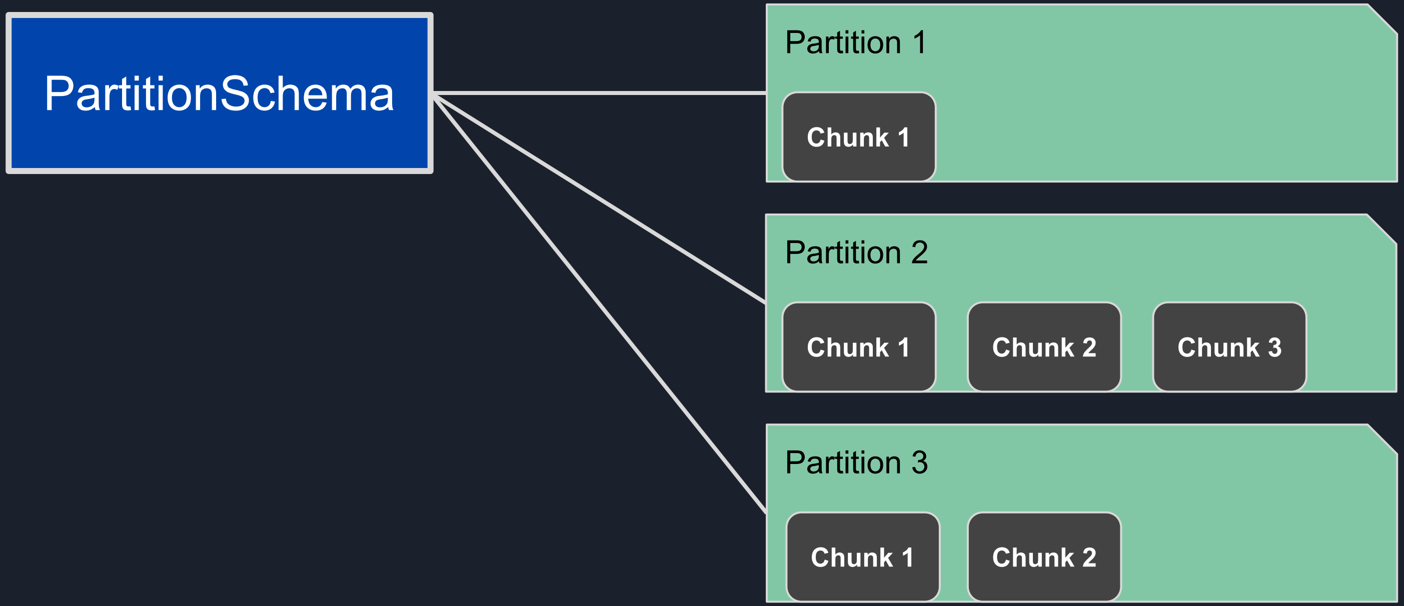


How **we** built this.

Partitioning as a Strategy



Our Implementation: Chunks in Partitions



Our implementation: Interface

```
void create_hash_partitioning(const ColumnID column_id, const HashFunction hash_function,
                                  const size_t number_of_partitions);

void create_range_partitioning(const ColumnID column_id,
                                   const std::vector<AllTypeVariant> bounds);

void create_round_robin_partitioning(const size_t number_of_partitions);

bool is_partitioned() const;

void remove_partitioning();

std::vector<ChunkID> get_partition(PartitionID partition_id);

std::vector<PartitionID> get_partition_ids();
```

Problems to solve

```
// creates a new chunk and appends it
void create_new_chunk();

// returns the chunk with the given id
Chunk& get_chunk(ChunkID chunk_id);
const Chunk& get_chunk(ChunkID chunk_id) const;
ProxyChunk get_chunk_with_access_counting(ChunkID chunk_id);
const ProxyChunk get_chunk_with_access_counting(ChunkID chunk_id) const;

// Adds a chunk to the table. If the first chunk is empty, it is replaced.
void emplace_chunk(Chunk chunk);
```

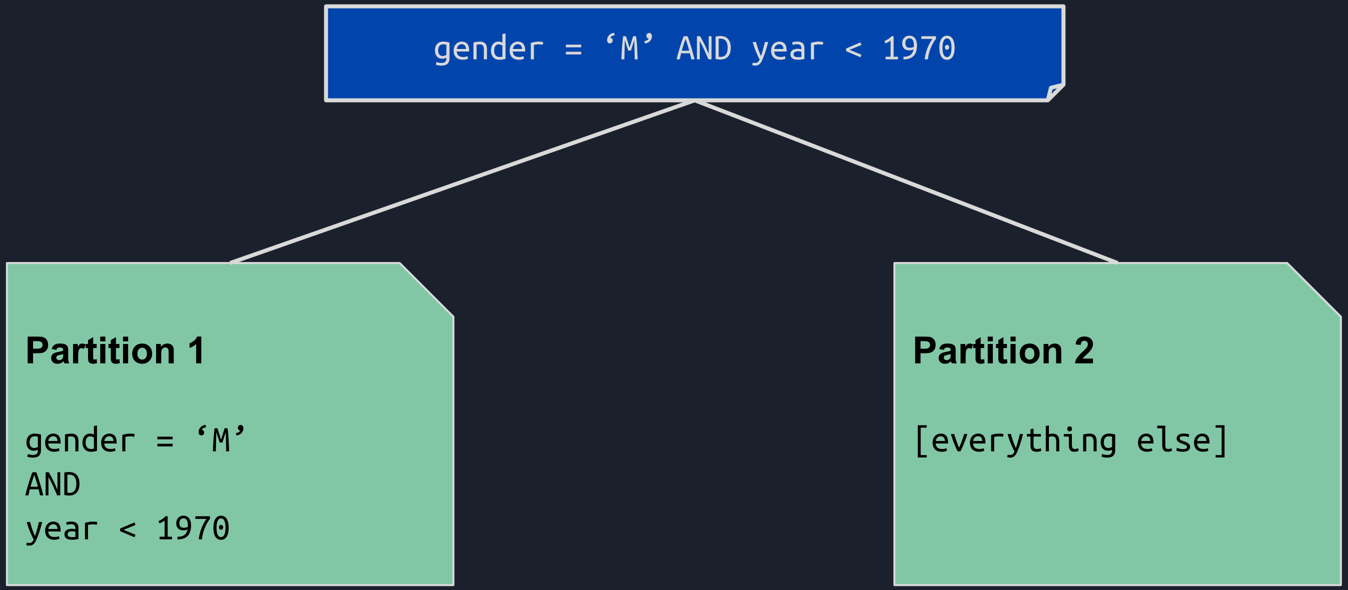
(excerpt from table.hpp)

Partitioning Criteria

We had to deal with the following questions:

- Single criterion vs. multiple criteria
- What if a tuple does not belong in any partition?

Multiple Criteria: Problem Statement



Multiple Criteria: Problem Statement

`gender = 'M' AND year < 1970`

Partition 1

```
gender = 'M'
AND
year < 1970
```

Partition 2

```
gender = 'M'
AND
NOT year < 1970
```

Partition 3

```
NOT gender = 'M'
AND
year < 1970
```

Partition 4

```
NOT gender = 'M'
AND
NOT year < 1970
```

Multiple Criteria: Approach and Solution

Each table has only one partition schema. The partition schema has to ensure consistency.

For example:

- Range partitioning uses split points.
  [20, 50] leads to 3 partitions:
  - one with values <= 20
  - one with 20 < values <= 50
  - one with values > 50

- Hash partitioning has user-defined number of partitions.
  E.g. modulo of hash value.

Further Work: Liberal Partitioning

Liberal Partitioning:

Partitioning criteria are not disjoint
→ One tuple can be in multiple partitions

If a tuple matches multiple partitions, we have to decide

1. Put it in **all** matching partitions → Deduplication problem
2. Put it in **one** matching partition → More partitions to be searched
3. Put it in a **remainder** partition → Can lead to uneven distribution

Optimizer Rules

Midterm Presentation

Develop your own database

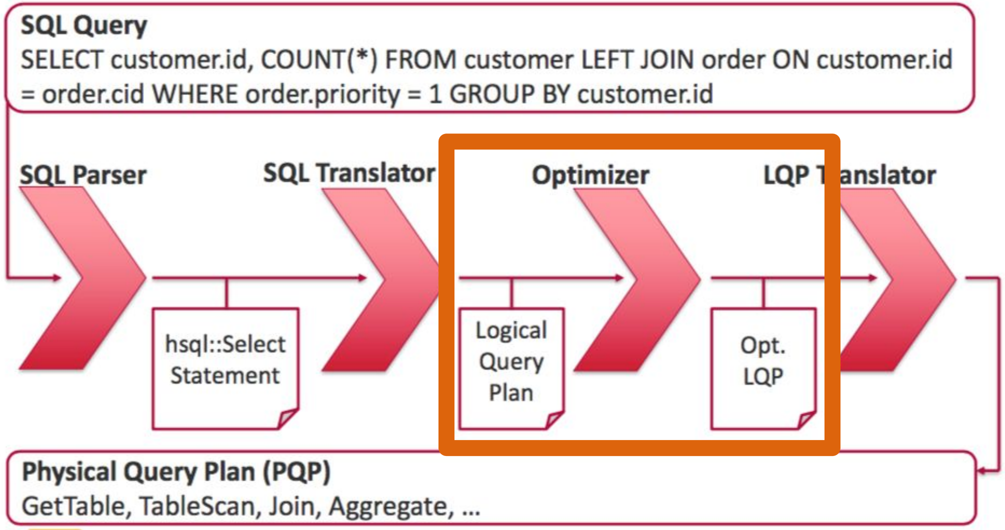
Falco Dürsch, Maxi Fischer, Tim Friedrich

2018-01-10

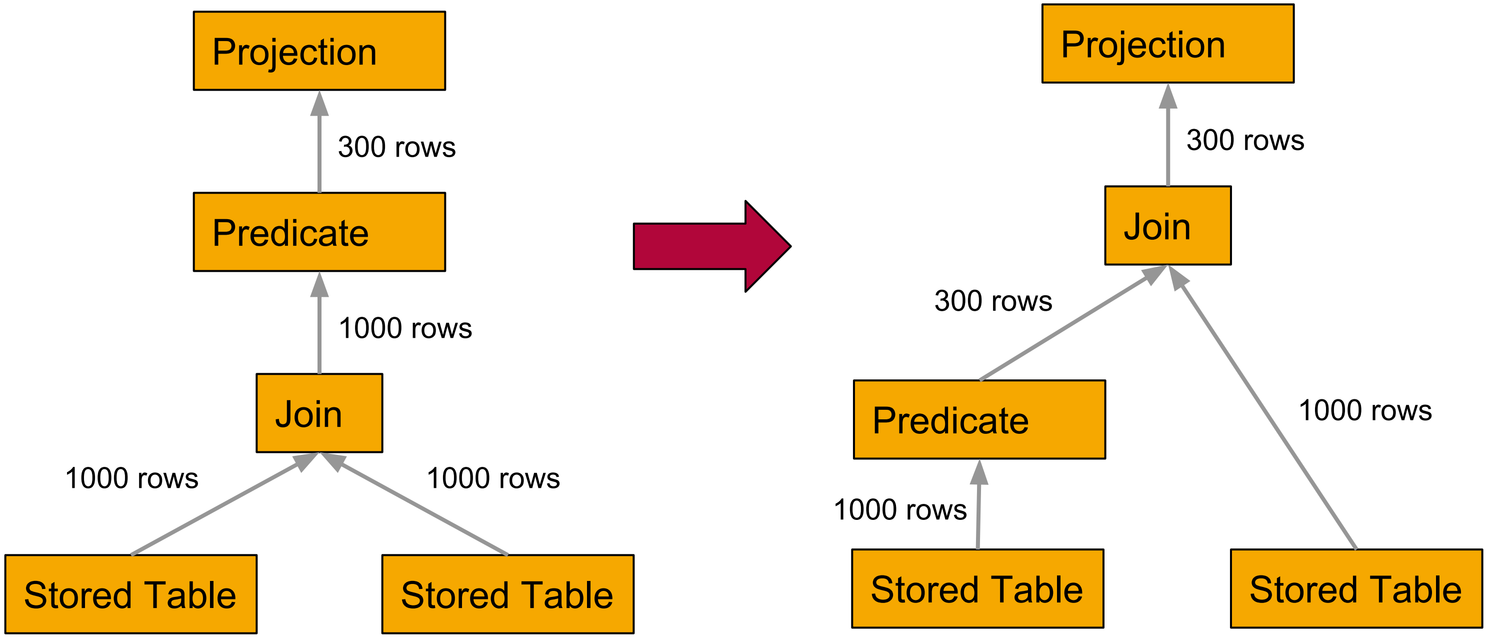
Optimizer Component

Idea: Transform a query plan to a more performant one (memory, CPU)

- Consists of a set of transformation rules
- Loops through them without any sense of ordering (no dependency management between rules)



Predicate Pushdown Explained



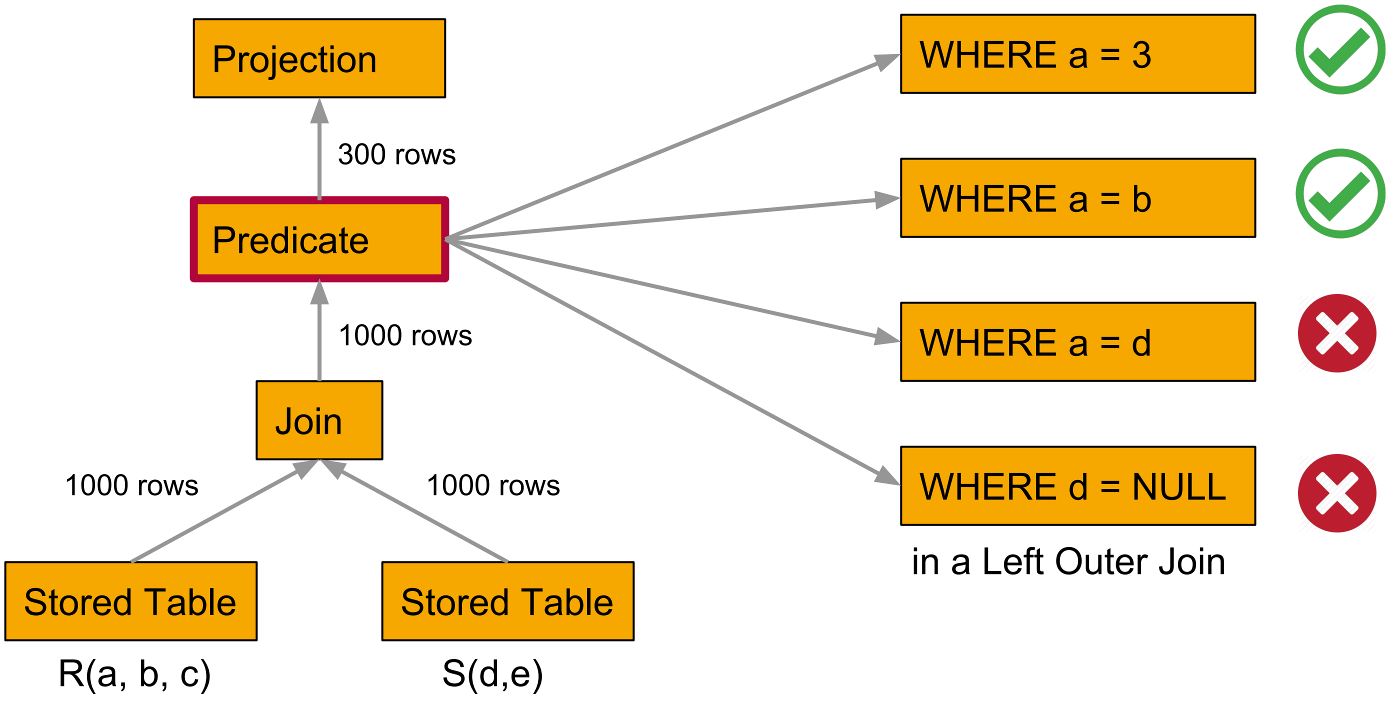
Predicate Pushdown - Node Types

Stored Table	Tables need to be loaded first
Aggregate, Limit	Predicate could change row count or refer to aggregated value
Union, Select Distinct	Not implemented
Validate, Predicate	Subject to Predicate Reordering Rule

Predicate Pushdown - Node Types

Stored Table	Tables need to be loaded first
Aggregate, Limit	Predicate could change row count or refer to aggregated value
Union, Select Distinct	Not implemented
Validate, Predicate	Subject to Predicate Reordering Rule
Projection	Possible if predicate column remains unchanged (arithmetic)
Sort	Possible
Join	Possible, dependent on join type Predicate must not involve reference both join partners

Predicate Pushdown Example



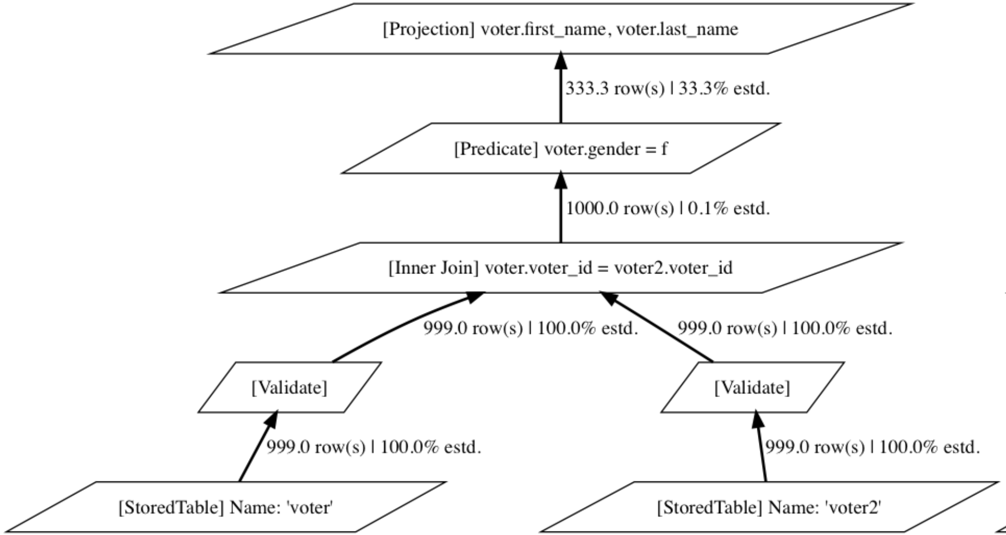
Conclusion and Next steps

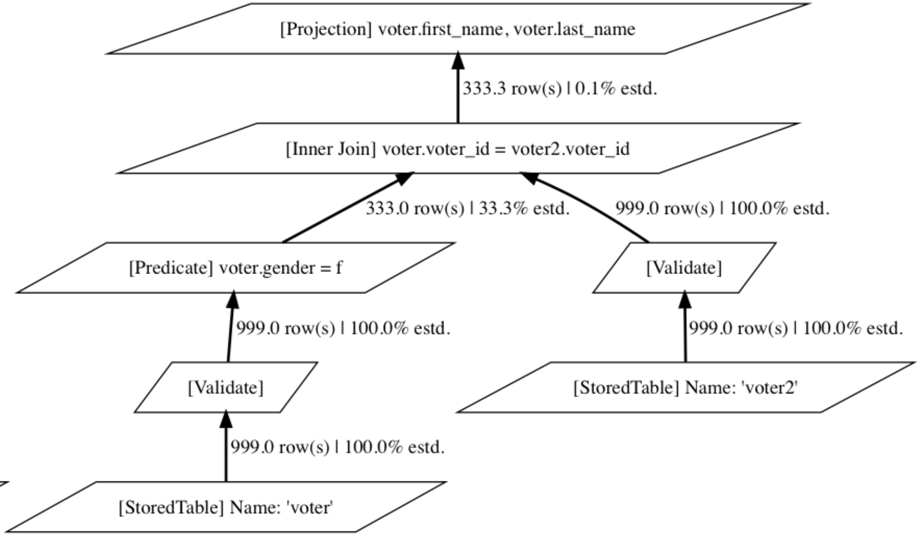
- Costs of joins can be reduced
- Precise definition required to keep LQP idempotence

- Another optimizer rule (Logical optimization, Sort Positioning Rule)
- Support more node types (Projection, Sort)
- Support TPC-H-13 query

Backup

Predicates Pushdown Example





Pruning Filters

Speed up filter operations

What is Pruning?

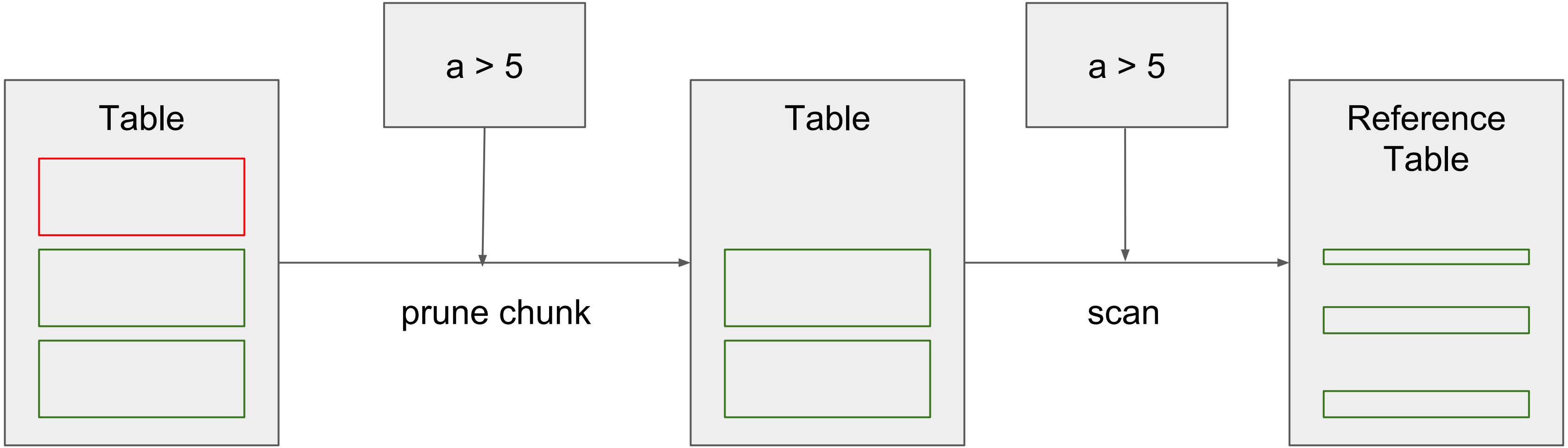
Reducing the amount of data (chunks in our case) to process when executing queries.

Improves cardinality estimation of filtering operations
(and thus helps the optimizer determine a better order of execution)

What do we do?

We will implement pruning for immutable chunks by extending compressed chunks with statistics (e.g. min, max).

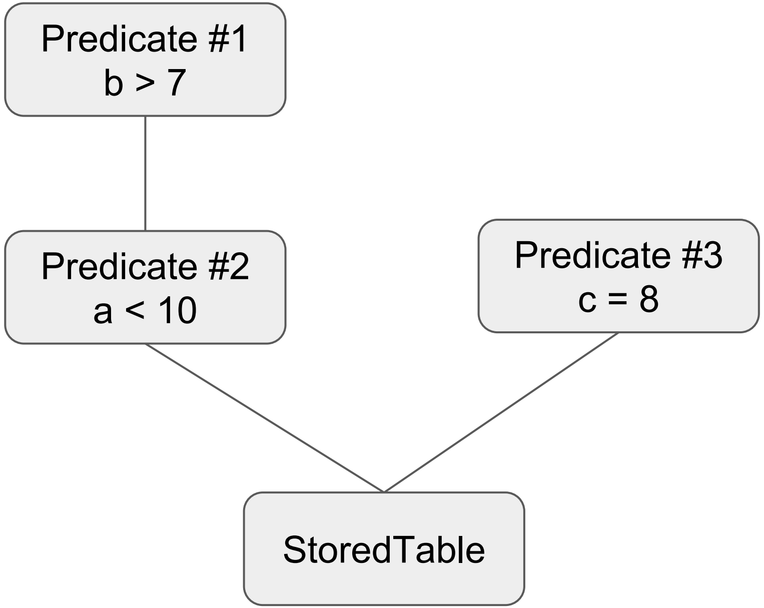
These will be used to reduce the amount of chunks that are considered in the GetTable operator.



How do we do it?

- add statistics calculation to chunk compression
- make chunk statistics available to the optimizer
- add ChunkPruningRule to the query optimizer
- create exclusion list of chunks for the StoredTable LQP Nodes
- use exclusion lists to perform less scanning
- remove prunable chunks in GetTable Operator

How does the optimizer rule work?

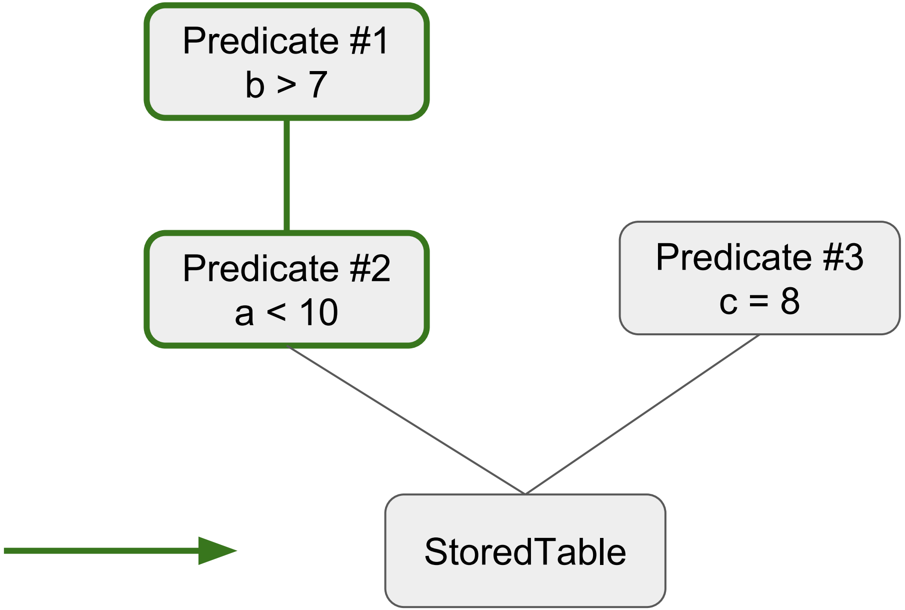


exclusion list: nullptr

All Chunks: {1, …, 10}

How does the optimizer rule work?

detect predicate chain

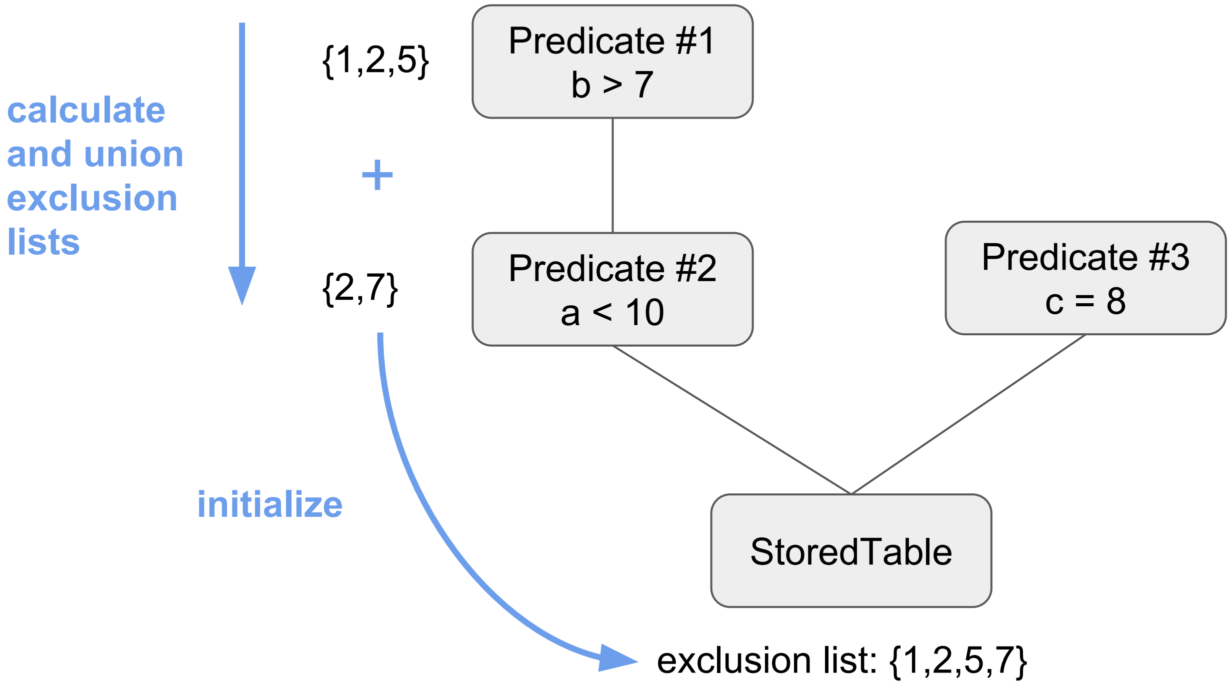


that ends in StoredTable Node

exclusion list: nullptr

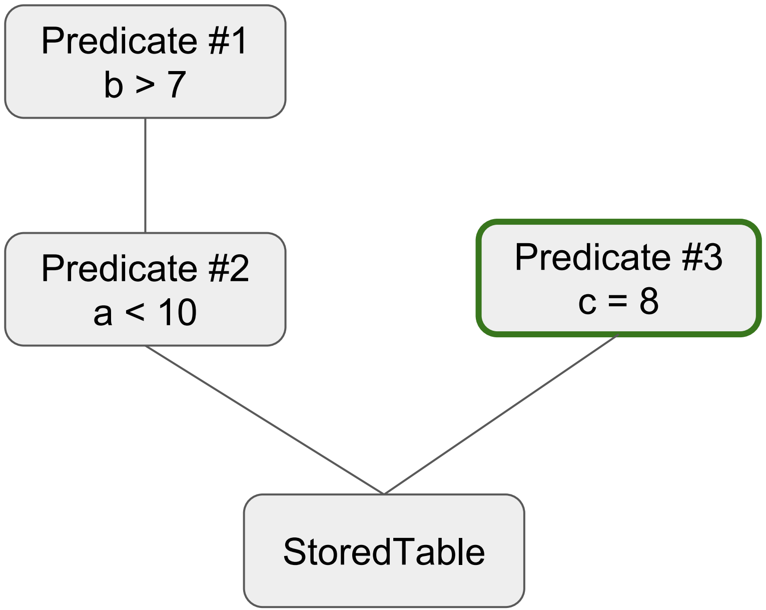
All Chunks: {1, …, 10}

How does the optimizer rule work?



All Chunks: {1, …, 10}

How does the optimizer rule work?

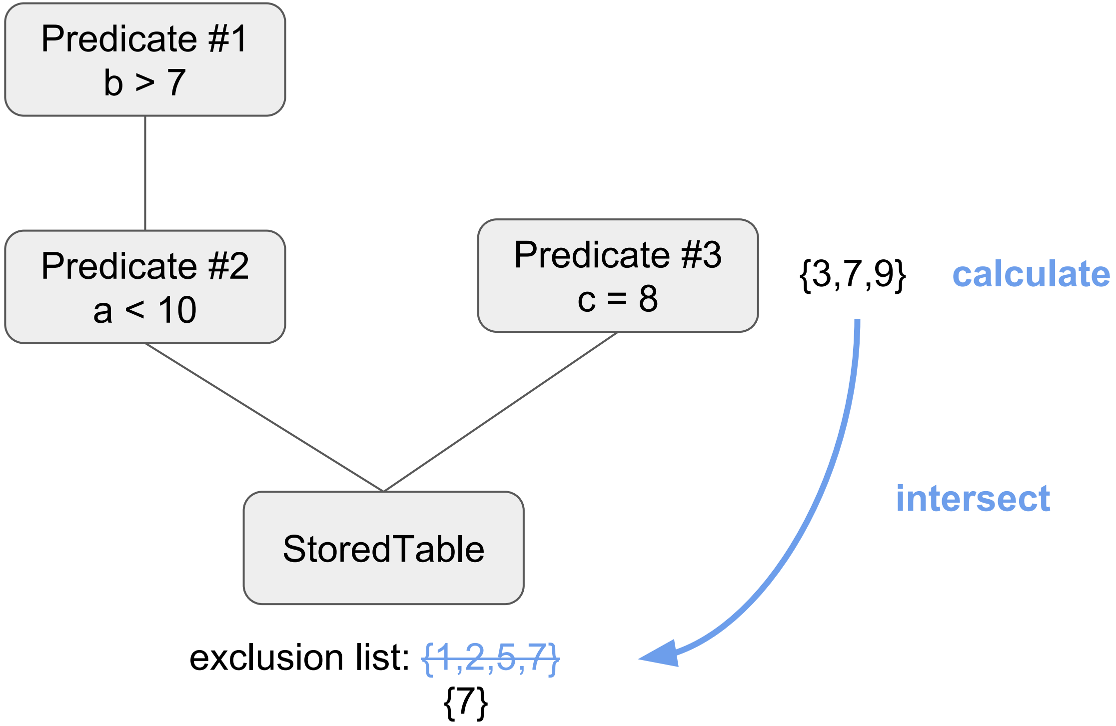


detect

exclusion list: {1,2,5,7}

All Chunks: {1, …, 10}

How does the optimizer rule work?

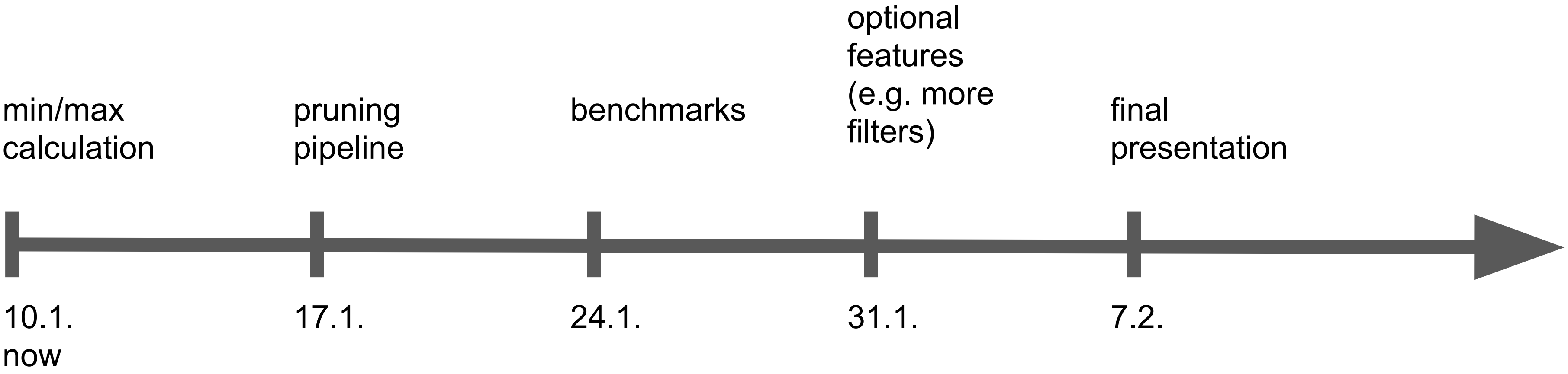


All Chunks: {1, …, 10}

Things that will get modified

- Chunk: store ChunkStatistics with std::optional
- New Class: ChunkStatistics
- compress_* methods: calculate and create ChunkStatistics
- New Optimizer Rule (as shown in previous slide)
- GetTable Operator: create temporary table without the excluded chunks
- StoredTableNode (LQP): store the exclusion list

Timeline


min/max calculation
pruning pipeline
benchmarks
optional features (e.g. more filters)
final presentation
10.1. now
17.1.
24.1.
31.1.
7.2.

Subqueries

Philipp Otto, Juliane Waack, **David Hahn**

Develop your own Database - Winter Term 2017/18

SELECT a FROM t1
WHERE a < **(SELECT MAX(b) FROM t2)**

Uncorrelated:

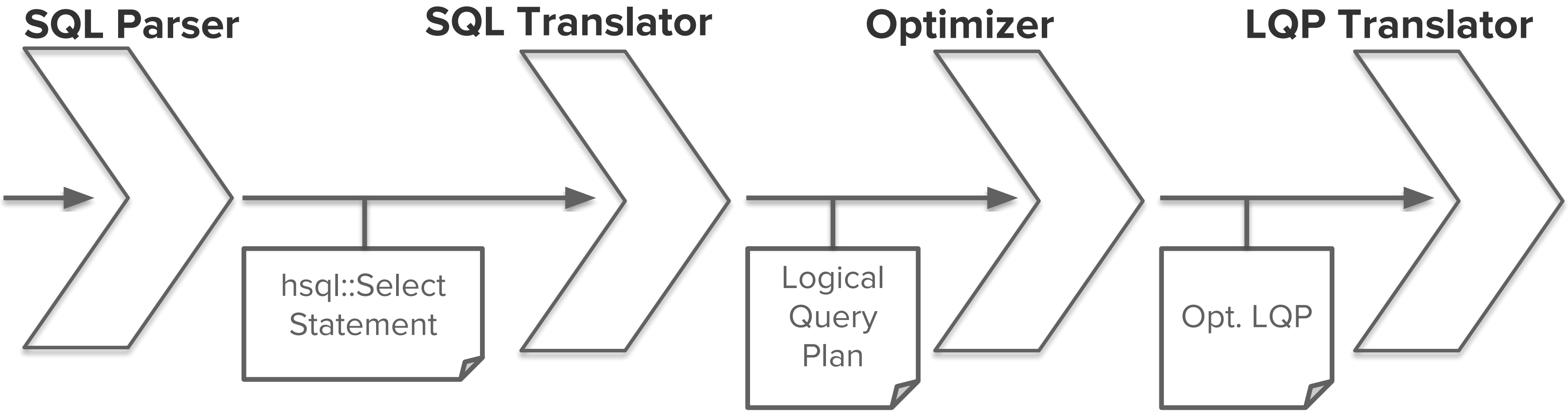
SELECT a FROM t1 WHERE a < **(SELECT MAX(b) FROM t2)**

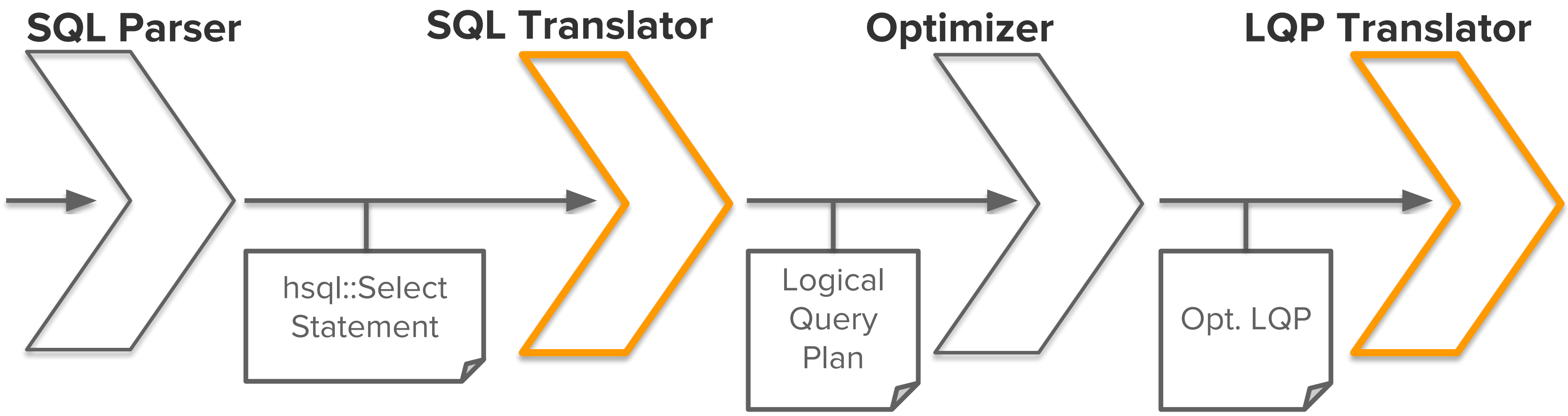
Correlated:

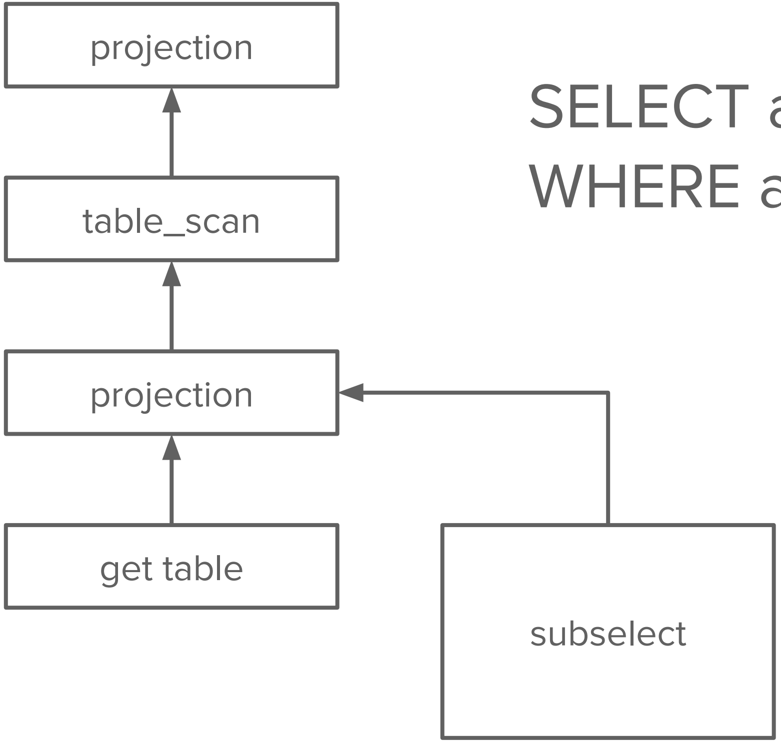
SELECT a, **(SELECT b FROM t2 WHERE b = a + 4)** FROM t1;

```
(debug)> SELECT a FROM t1
=== Columns
|       a|
|   float|
=== Chunk 0 ===
|     1.1|
|     2.2|
|     3.3|
|     4.4|
===
4 rows total (PARSE: 13 µs, COMPILE: 16 µs, EXECUTE: 350 µs (wall time))
(debug)> SELECT MIN(b) from t2
=== Columns
|  MIN(b)|
|   float|
=== Chunk 0 ===
|     1.1|
===
1 rows total (PARSE: 14 µs, COMPILE: 46 µs, EXECUTE: 2573 µs (wall time))
```

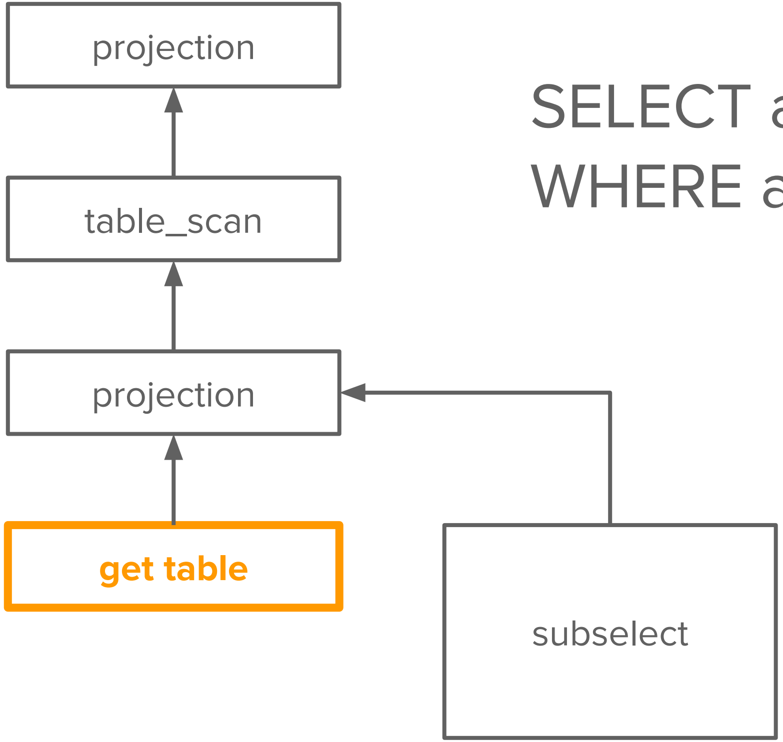
```
(debug)> SELECT a FROM t1 WHERE a > (SELECT MIN(b) from t2)
=== Columns
|       a|
|   float|
=== Chunk 0 ===
|     2.2|
|     3.3|
|     4.4|
===
3 rows total (PARSE: 14 µs, COMPILE: 16 µs, EXECUTE: 742 µs (wall time))
(debug)>
```


SQL Parser
SQL Translator
Optimizer
LQP Translator
hsql::Select Statement
Logical Query Plan
Opt. LQP


SQL Parser
SQL Translator
Optimizer
LQP Translator
hsql::Select Statement
Logical Query Plan
Opt. LQP

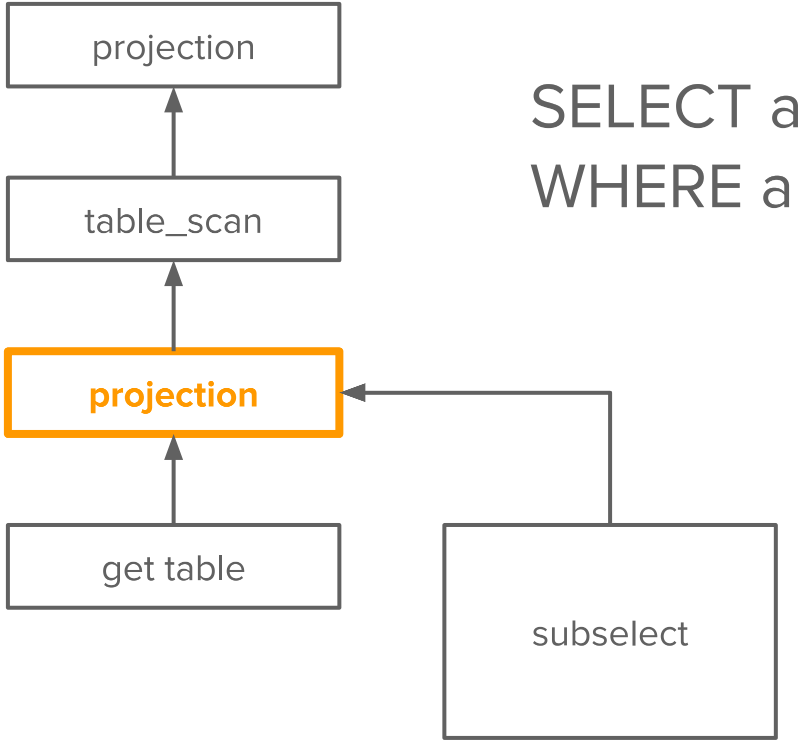


SELECT a FROM t1
WHERE a < **(SELECT MAX(b) FROM t2)**



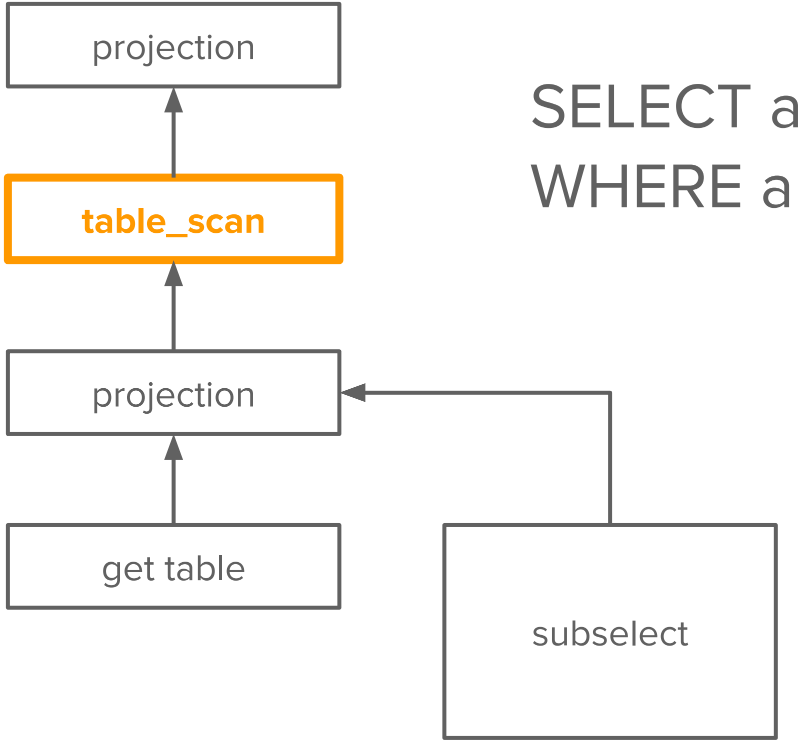
SELECT a FROM t1
WHERE a < **(SELECT MAX(b) FROM t2)**

a
5
6
8



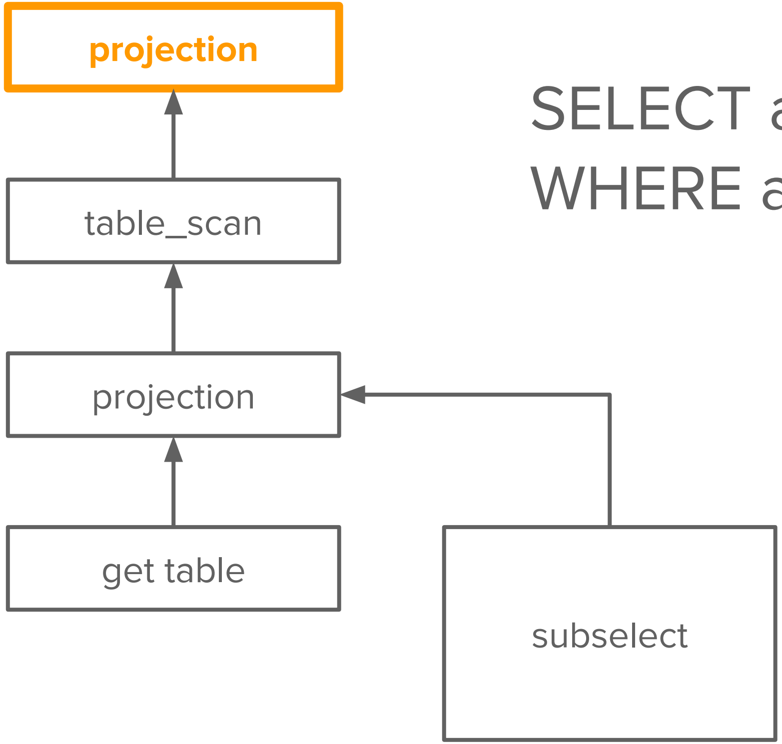
SELECT a FROM t1
WHERE a < **(SELECT MAX(b) FROM t2)**

a	MAX(b)
5	7
6	7
8	7



SELECT a FROM t1
WHERE a < **(SELECT MAX(b) FROM t2)**

a	MAX(b)
5	7
6	7



SELECT a FROM t1
WHERE a < **(SELECT MAX(b) FROM t2)**

a
5
6

Step 1

execute subquery separately for every row

Step 2

optimize uncorrelated subqueries by only executing once

Step 3

flatten correlated subqueries to JOINs if possible

Step 3

SELECT a, (SELECT b FROM t2 WHERE b = a + 4) FROM t1;

↓

SELECT a, b FROM t1 LEFT JOIN t2 ON b = a + 4;

Subqueries

Philipp Otto, Juliane Waack, **David Hahn**

Develop your own Database - Winter Term 2017/18

Self Driving Database

Adrian Holfter, Arthur Silber, Lukas Wenzel
Instructor: Jan Kossmann

Manual DB tuning is difficult

Which indexes should be created?
How should the data be partitioned?
How many threads should be used?

Large Problem Space, Inter-dependencies, Workload-specific decisions

The database knows best how to tune itself

Use Heuristics to automatically create **indices**

Example: Bank Accounts

NAME (string)	**BALANCE** (int)	**INTEREST** (float)	**LEVEL** (int)
Danni Cohdwell	144’811	0.157	3
Rosemary Picardi	236	0.226	3
Xenia Ziegler	424’675	0.239	2
Lilly Goodwin	3’645	0.538	5
...	...	...	...

Example: Bank Accounts

select BALANCE from CUSTOMER where NAME = 'Danni Cohdwell'
select NAME from CUSTOMER where LEVEL = 5
select INTEREST from CUSTOMER where NAME = 'Rosemary Picardi'

NAME (string)	**BALANCE** (int)	**INTEREST** (float)	**LEVEL** (int)
Danni Cohdwell	144’811	0.157	3
Rosemary Picardi	236	0.226	3

Example: Bank Accounts

select BALANCE from CUSTOMER where NAME = 'Danni Cohdwell'
select NAME from CUSTOMER where LEVEL = 5
select INTEREST from CUSTOMER where NAME = 'Rosemary Picardi'

⇒ **Which indices should be created?**

NAME (string)	**BALANCE** (int)	**INTEREST** (float)	**LEVEL** (int)
Danni Cohdwell	144’811	0.157	3
Rosemary Picardi	236	0.226	3

Example: Bank Accounts

select BALANCE from CUSTOMER where NAME = 'Danni Cohdwell'
select NAME from CUSTOMER where LEVEL = 5
select INTEREST from CUSTOMER where NAME = 'Rosemary Picardi'

scanned 2x
unique values

NAME (string)	**BALANCE** (int)	**INTEREST** (float)	**LEVEL** (int)
Danni Cohdwell	144’811	0.157	3
Rosemary Picardi	236	0.226	3

Example: Bank Accounts

select BALANCE from CUSTOMER where NAME = 'Danni Cohdwell'
select NAME from CUSTOMER where LEVEL = 5
select INTEREST from CUSTOMER where NAME = 'Rosemary Picardi'

scanned 2x
unique values

scanned 1x
repeating values

NAME (string)	BALANCE (int)	INTEREST (float)	LEVEL (int)
Danni Cohdwell	144’811	0.157	3
Rosemary Picardi	236	0.226	3

Example: Bank Accounts

select BALANCE from CUSTOMER where NAME = 'Danni Cohdwell'
select NAME from CUSTOMER where LEVEL = 5
select INTEREST from CUSTOMER where NAME = 'Rosemary Picardi'

scanned 2x
unique values

scanned 1x
repeating values

NAME (string)	**BALANCE** (int)	**INTEREST** (float)	**LEVEL** (int)
Danni Cohdwell	144’811	0.157	3
Rosemary Picardi	236	0.226	3

Example: Bank Accounts

select BALANCE from CUSTOMER where NAME = 'Danni Cohdwell'
select NAME from CUSTOMER where LEVEL = 5
select INTEREST from CUSTOMER where NAME = 'Rosemary Picardi'

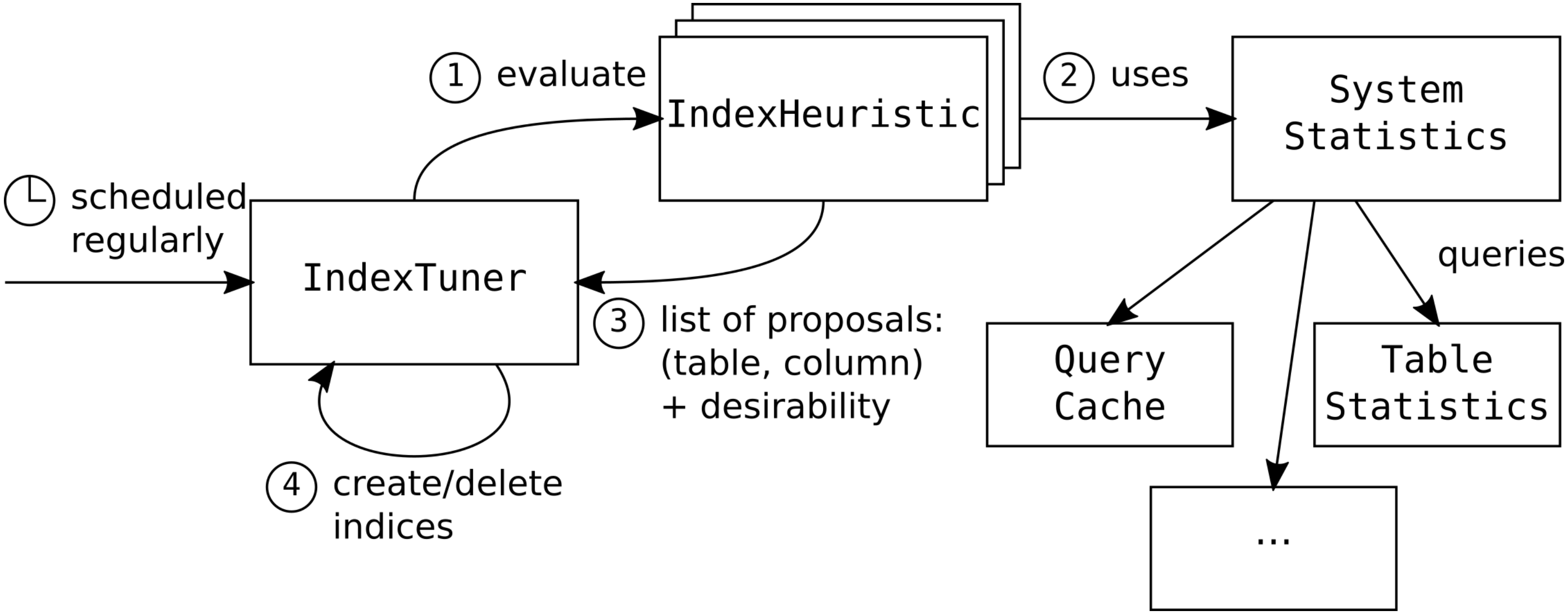
⇒ Create index on NAME and (maybe) LEVEL

scanned 2x
unique values

scanned 1x
repeating values

NAME (string)	**BALANCE** (int)	**INTEREST** (float)	**LEVEL** (int)
Danni Cohdwell	144’811	0.157	3
Rosemary Picardi	236	0.226	3

Our Architecture



Demo

```
Loading binary table...
Table loaded (10’000’000 rows in 10 chunks)
Executing queries a first time to fill up the cache...
Execute IndexTuner...

Recommended changes:
  Create index on table CUSTOMER, column NAME (desirablity 100%)
  Create index on table CUSTOMER, column LEVEL (desirablity 75%)
```

Demo

```
Executing queries a second time (with optimized indices)...
Execution times are in microseconds

SELECT BALANCE FROM CUSTOMER WHERE NAME = 'Danni Cohdwell'
     reduced to: 27.829%  (before/after: 2060 / 558)

SELECT NAME FROM CUSTOMER WHERE LEVEL = 5
     reduced to: 7.507%  (before/after: 60370 / 4810)

SELECT INTEREST FROM CUSTOMER WHERE NAME  = 'Rosemary Picardi'
     reduced to: 2.896%  (before/after: 2033 / 57)
```

Outlook

- Desirability metrics (consider value distributions, query frequencies)
- Index budgeting (creation and maintenance costs, memory footprint) → Cost/Benefit optimization
- Integration into Hyrise:
  - Generalize cache implementation specifics
  - SQL Pipeline does not yet use caching
  - IndexScan not yet used

NUMA-Optimized Join

Develop Your Own Database - WS 17/18
Mid-term Presentation

Jonas Beyer, **Julian Niedermeier, Florian Wagner**

Hasso Plattner Institut
Digital Engineering · Universität Potsdam

Existing Joins

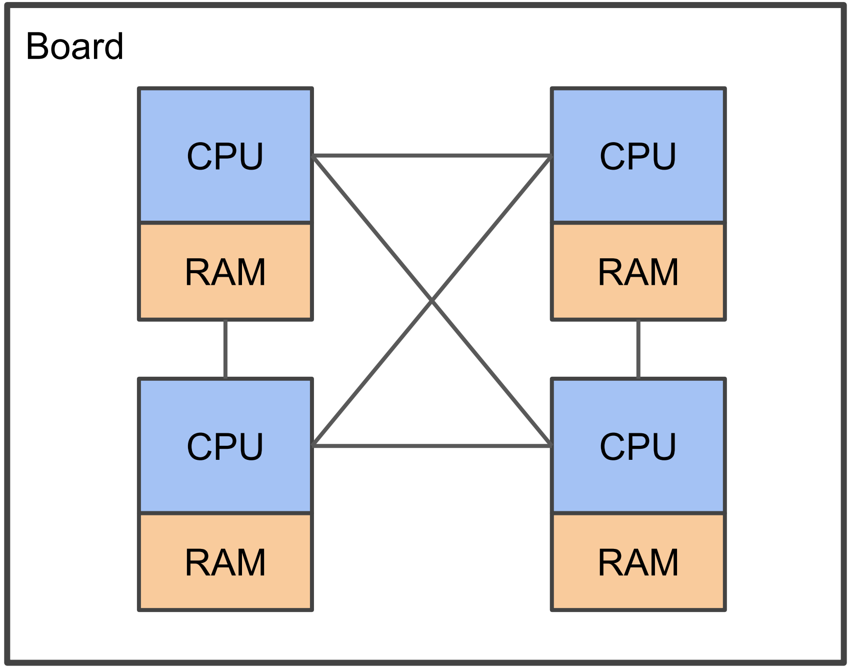
1. Hash Join
2. Sort Merge Join
3. Nested Loop Join

Existing Joins

1. Hash Join
2. Sort Merge Join
3. Nested Loop Join

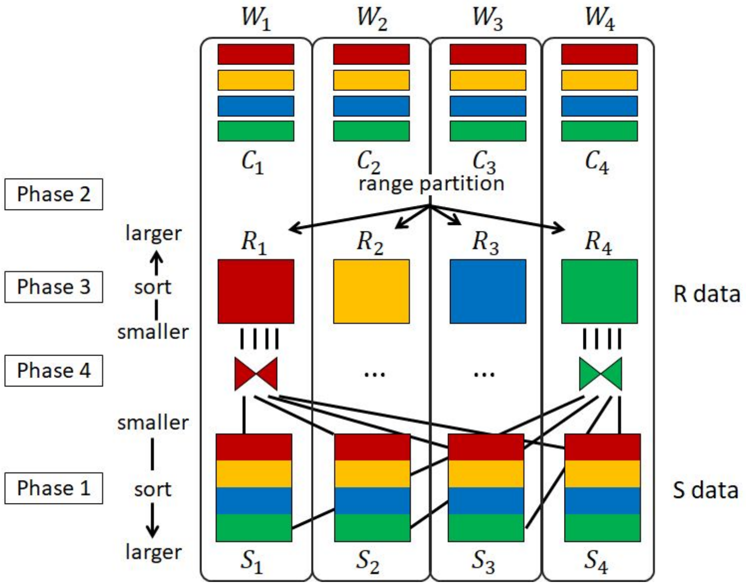
Not NUMA aware

What is NUMA? Reminder

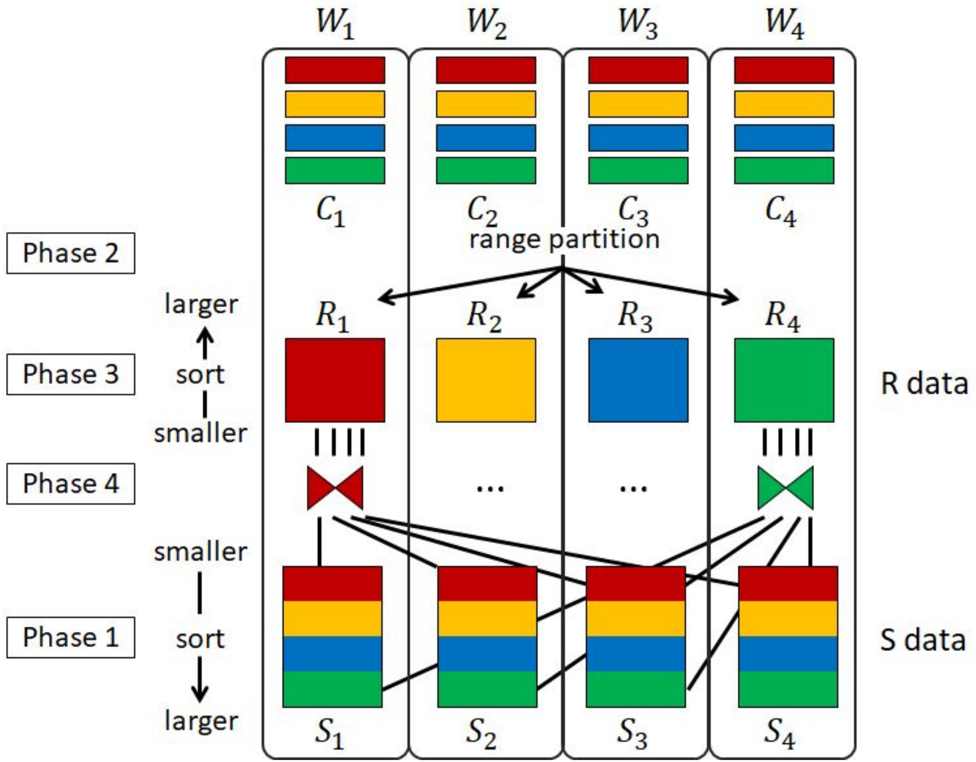


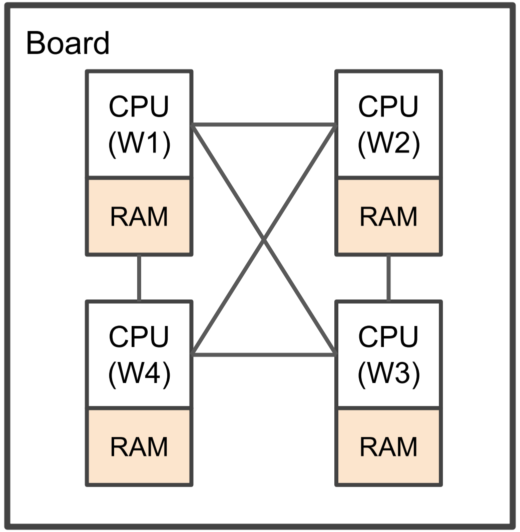
Massively Parallel Sort-Merge Join (MPSM)

- No **hash** calculation
- No hashmap **probing**
- Only **sequential** data access through **histogram** based **partitioning**
- No **synchronisation** during join

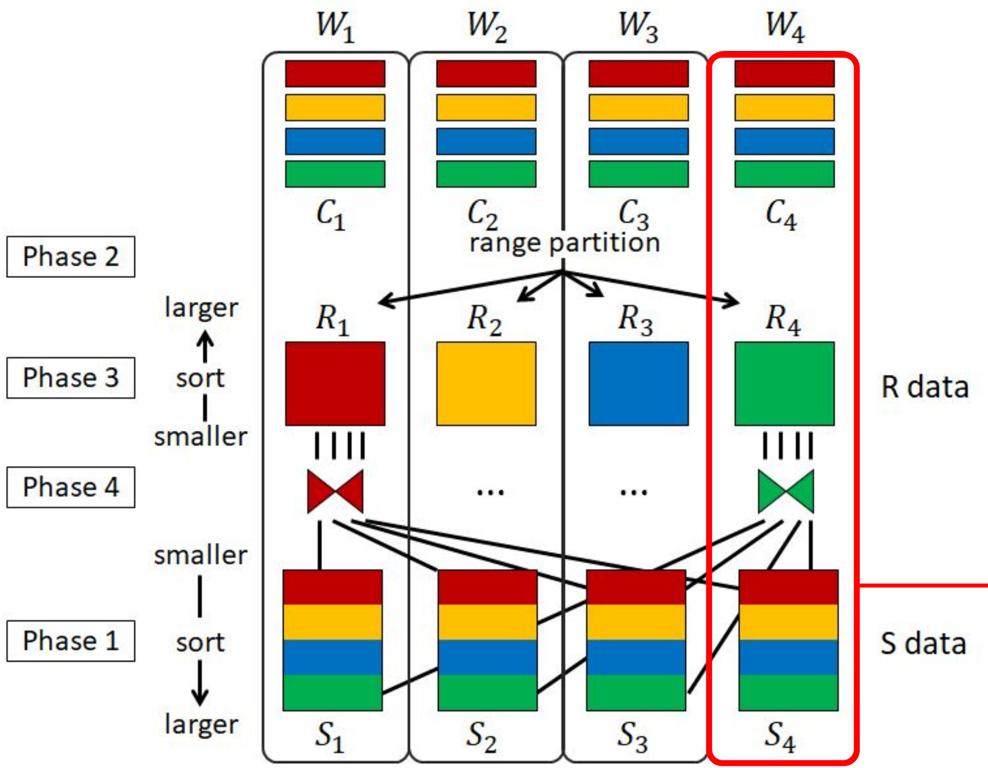


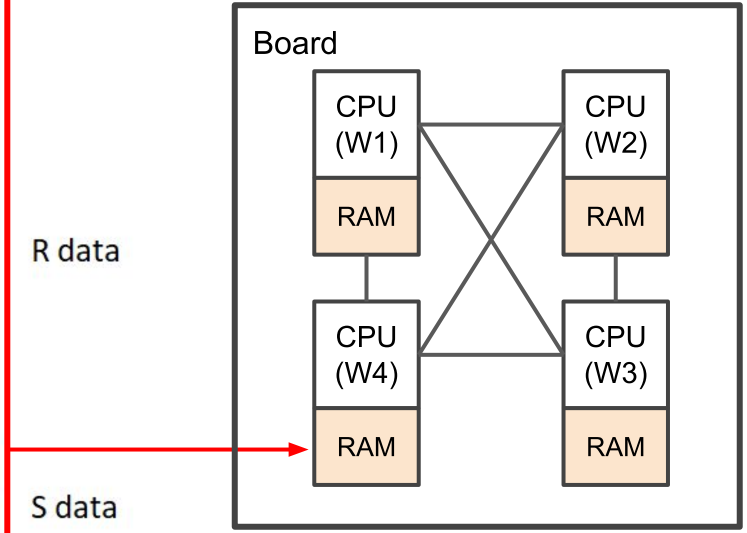
Massively Parallel Sort-Merge Join



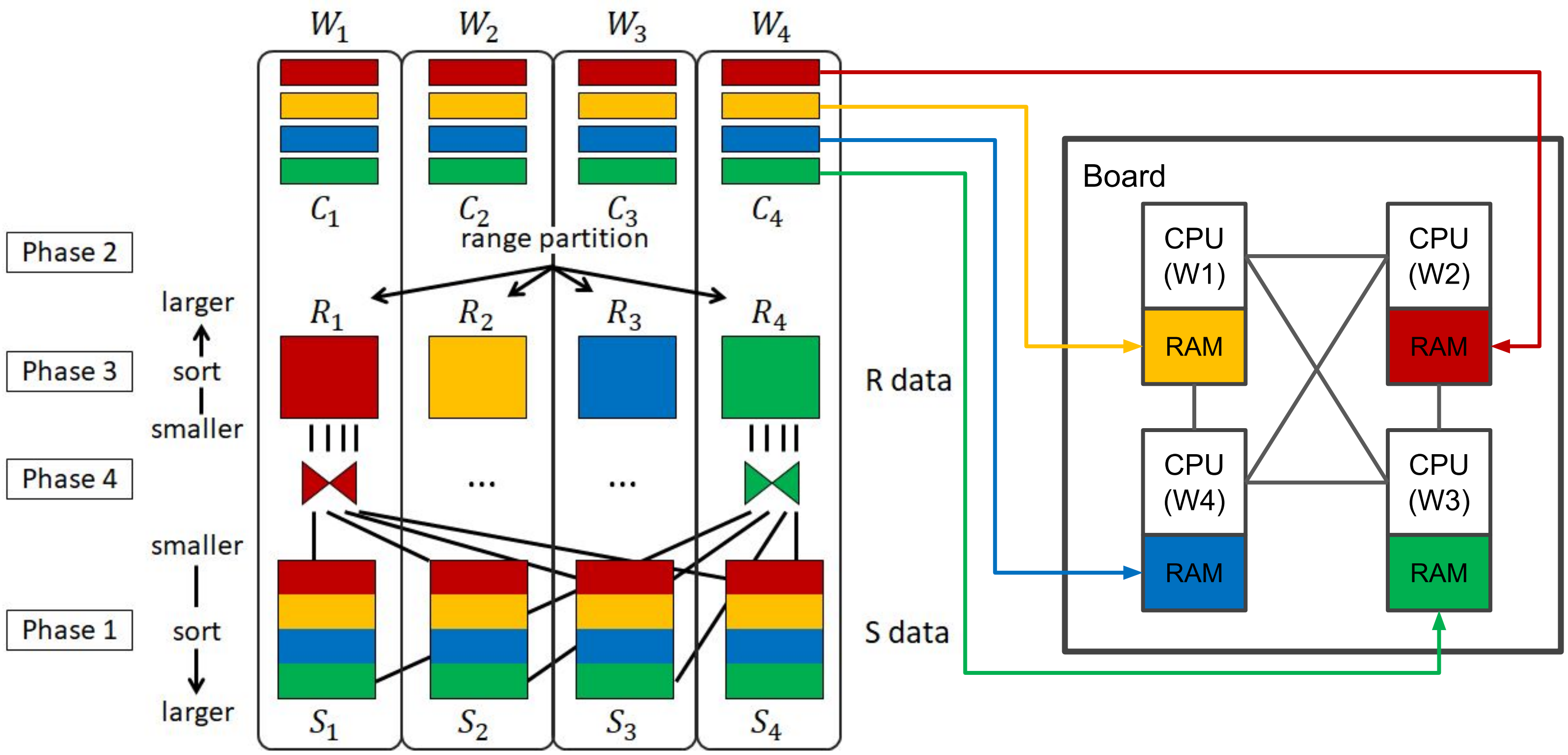


Massively Parallel Sort-Merge Join

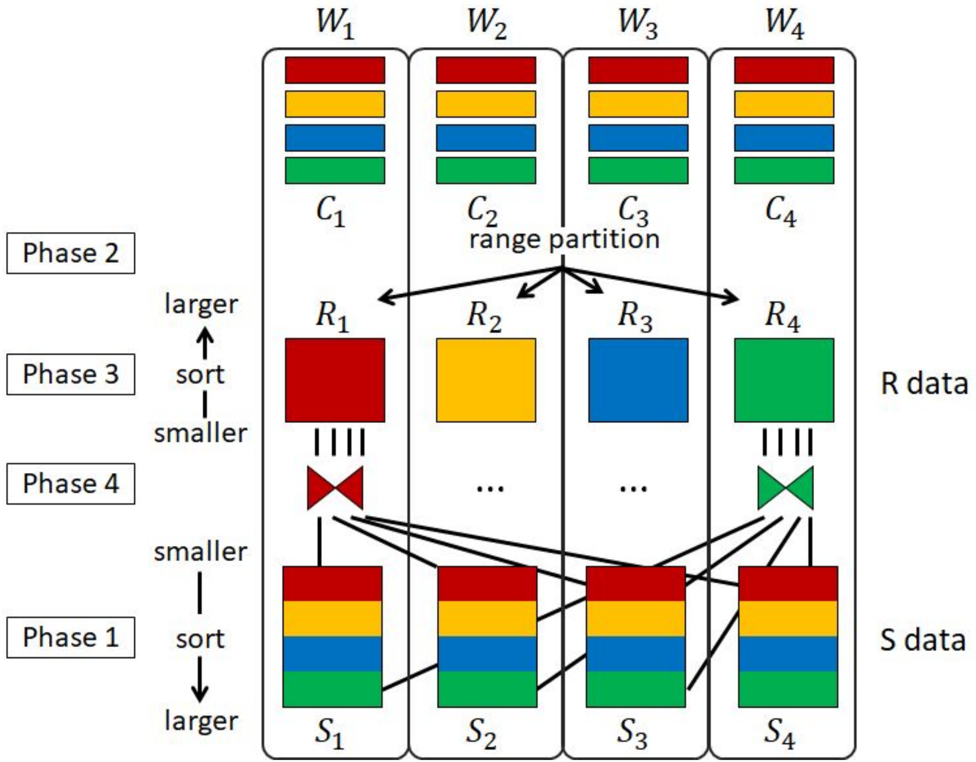


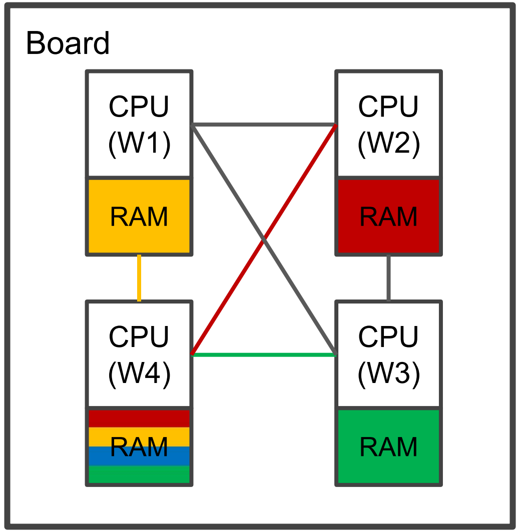


Massively Parallel Sort-Merge Join - Phase 2



Massively Parallel Sort-Merge Join - Phase 4





In Hyrise (Outlook)

- Build Index Join
- Implement Range-Partitioned MPSM Join
- Extend MPSM Join to split smartly (using histograms)
- *(Make Index Join NUMA aware)*

Benchmarks

- TPCH
  - Queries 2, 8, 9
- Join on selected columns
  - Happens often
  - Can skew range partitions
- Generated data
  - Different skew
  - Different relative sizes