Presentations from Yahoo! JAPAN

Session: Yosegi

https://github.com/yahoojapan/yosegi

Booth: AntPickax

https://antpick.ax/index.html

https://github.com/yahoojapan/AntPickax

Schema-Less Columnar Storage Format Yosegi

Open Source Summit Japan 2019

Yasunori Oto

About me

Yasunori Oto

Yahoo! JAPAN

Data Platform Division

Data Group

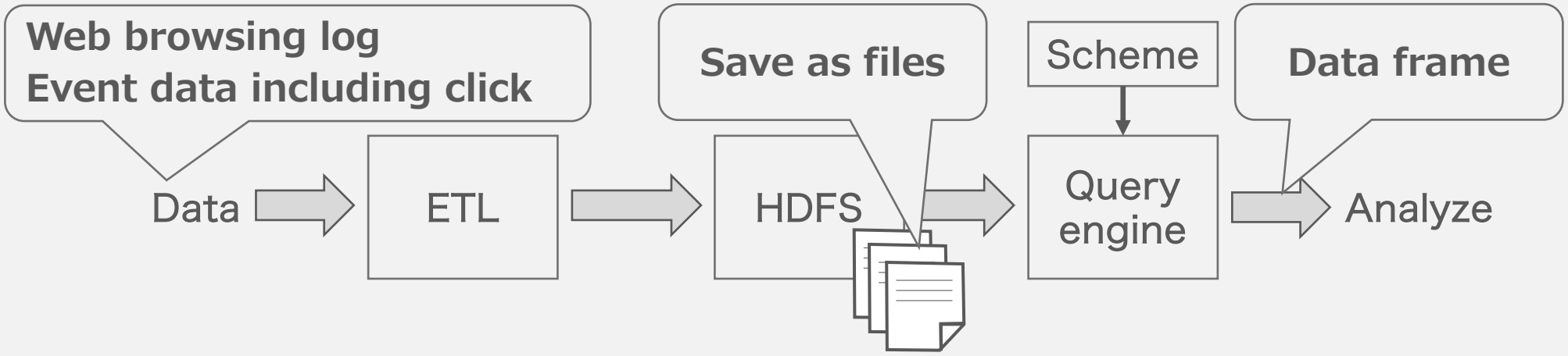
Contents

- Log collection system
- Format strategy
- Yosegi
- Performance
- Use case
- Conclusion

Log collection system

Log collection system

- Collect log data from each service
- Save data as files
- Analyze data as a data frame



Format strategy

Format strategy

Requirements

- Flexibility to changing structure of inputs
- Compact storage data
- Efficient use at the time of analysis

Format strategy

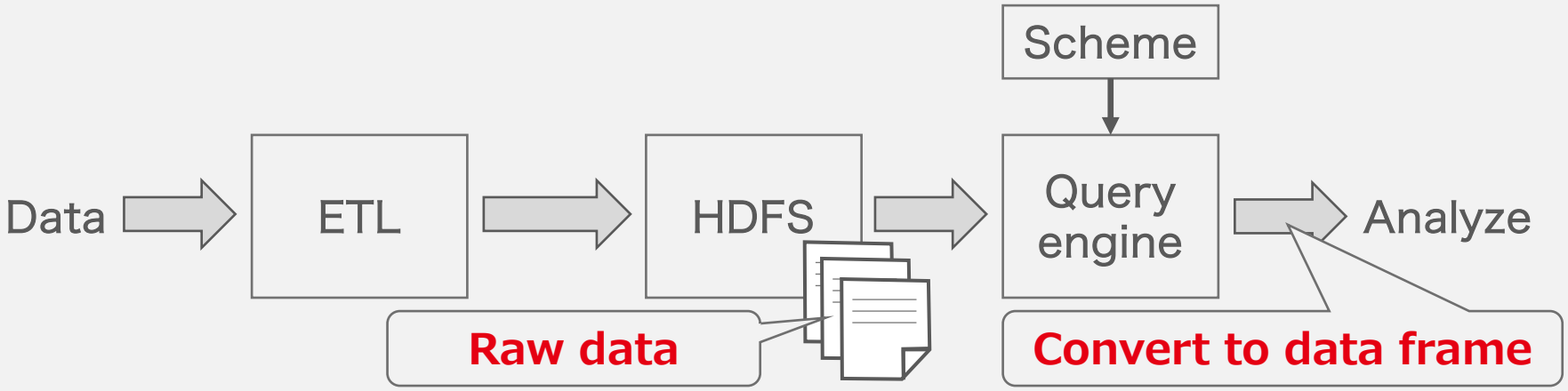
Existing methods

- Schema-on-read by raw data
- Schema-on-write by columnar format

Format strategy –Existing methods

Schema-on-read by raw data

- Storage by raw data
- Convert to data frame before each analysis



Format strategy –Existing methods

Schema-on-read by raw data

Pros

- **Changeable structure of inputs**
  without modification of the system
- **Changeable analysis schema**
  without consideration of input data

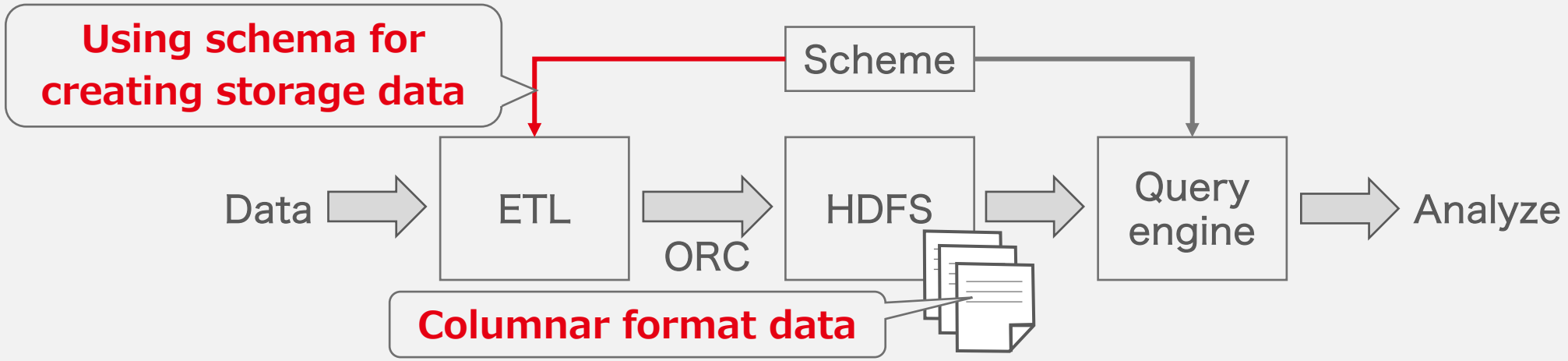
Cons

- **Not so good CPU and Memory usage and Compression ratio**

Format strategy –Existing methods

Schema-on-write by columnar format

- Need schema before storage
- Create storage data from schema



Format strategy –Existing methods

Schema-on-write by columnar format

Pros

- **Small size storage data**
- **Good CPU and Memory usage**
  at the time of analysis

Cons

- **Need schema at the time of writing**
- **Less flexibility for changing structure of input data**

Format strategy –Existing methods

Get flexibility for schema-on-write by columnar format

- Using a general purpose storage column
- Buffering temporary raw data storage before convert to columnar format

Format strategy –Existing methods

Get flexibility for schema-on-write by columnar format

- Using a general purpose storage column
- Buffering temporary raw data storage before convert to columnar format

Format strategy –Existing methods

Get flexibility for schema-on-write by columnar format

Using general purpose storage column

- **Using map type data**
- **Add key when input data structure changes**

Format strategy –Existing methods

Get flexibility for schema-on-write by columnar format

Pros

- **Able to add data without schema changing**

Cons

- **Not so good CPU and Memory usage**
  because of treating large data in this column
- **Functions for efficiency like Push Down cannot work well**

Format strategy –Summary

- **Each method has its pros and cons**
- **Select the method based on data update frequency and purpose of analysis**
- **Difficult to change after system implementation**

Existing methods

	Raw data	Columnar format
Schema-on-read	JSON, text	
Schema-on-write		ORC, Parquet

Yosegi

Yosegi –Strategy

Requirements

- Flexibility to changing structure of inputs
- Compact storage data
- Efficient use at the time of analysis

Yosegi –Strategy

Extract advantages from existing methods

Schema-on-read

- **Changeable structure of inputs**
  without modification of the system

Columnar format

- **Small size storage data**
- **Good CPU and Memory usage**
  at the time of analysis

Yosegi –Strategy

Extract advantages from existing methods

Schema-on-read by columnar format

- **Store data in columnar format**
- **No schema required at the time of writing**

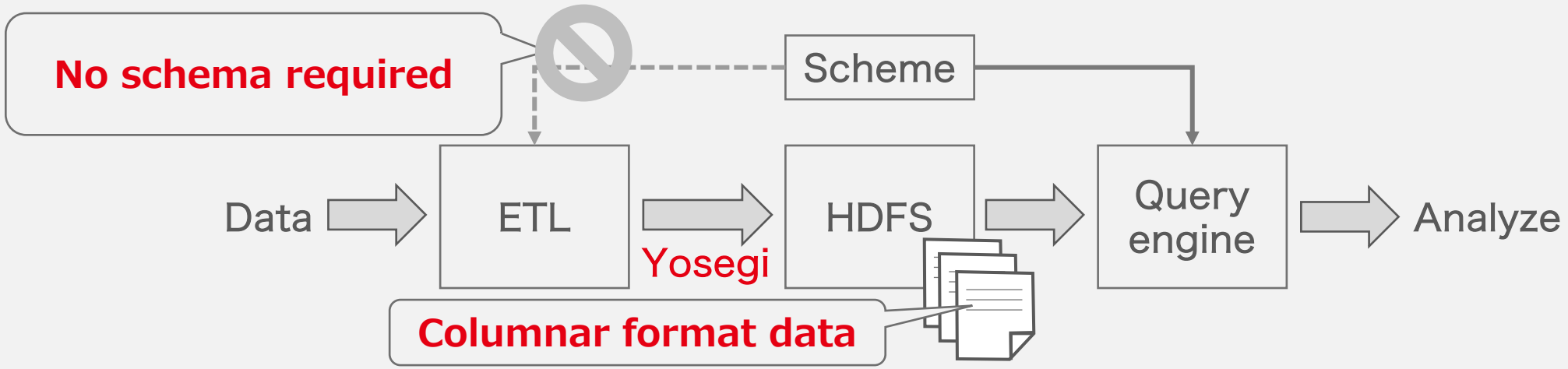
	Raw data	Columnar format
Schema-on-read	JSON, text	**Yosegi**
Schema-on-write		ORC, Parquet

Yosegi –Strategy

Extract advantages from existing methods

Schema-on-read by columnar format

- **Store data in columnar format**
- **No schema required at the time of writing**



Yosegi –How to

Extract advantages from existing methods

Writing

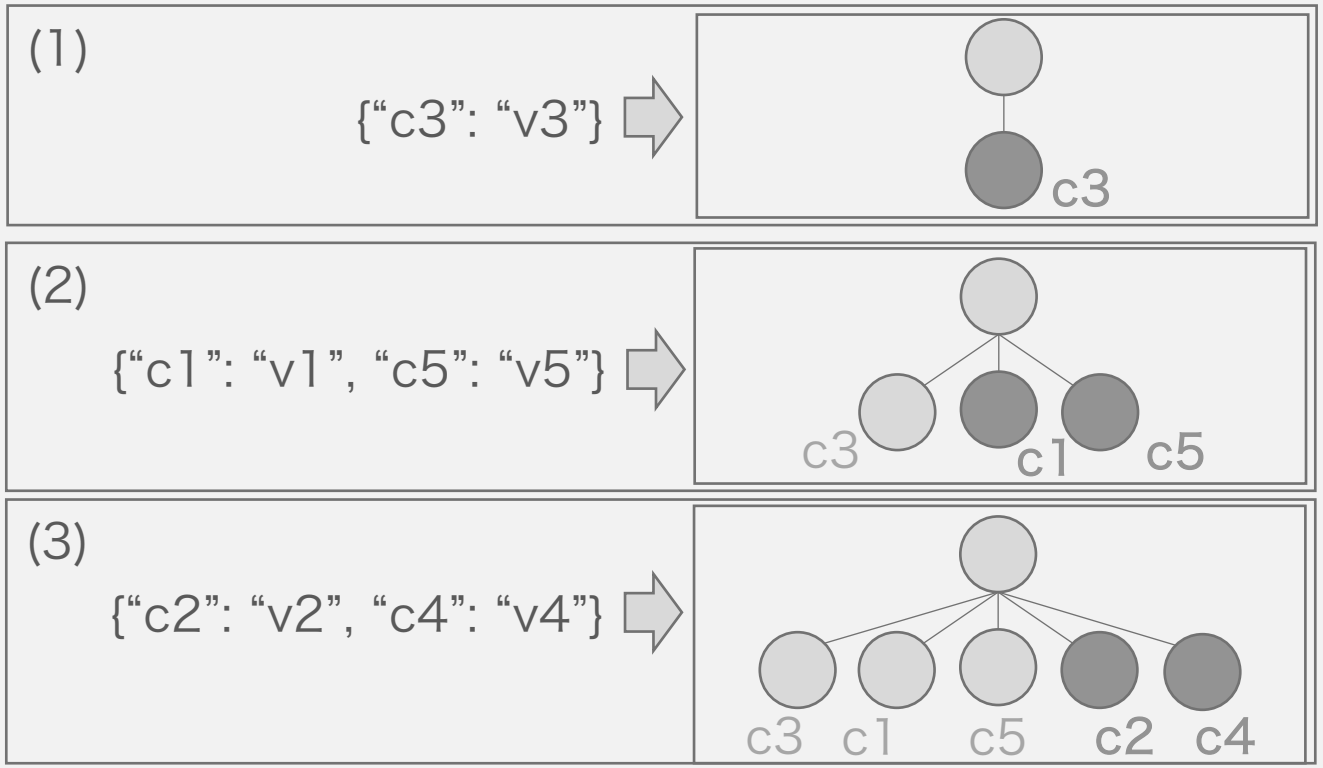
- **Create the namespace dynamically**
  as a tree structure

Reading

- **Get the node in the namespace**
  by mapping the field name in the schema

Yosegi –How to

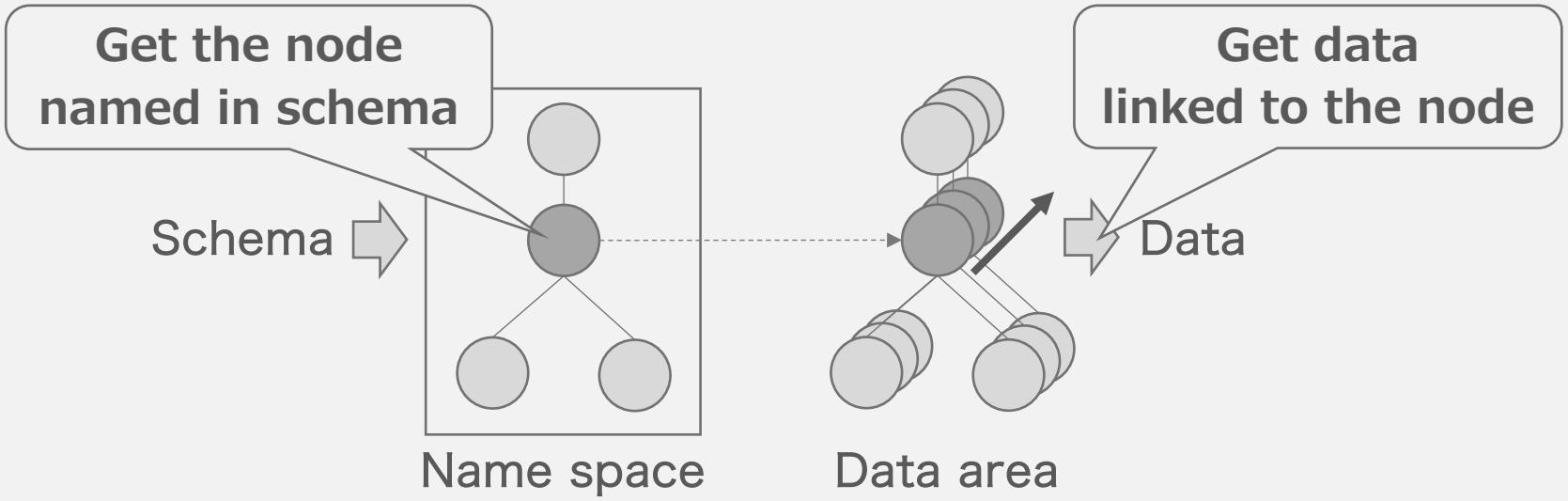
Construct the namespace dynamically as a tree structure



Yosegi –How to

Reading

Get the node in the namespace by mapping the field name in the schema



Yosegi –How to

Construct the namespace dynamically

Input data type

- Primitive
- Struct
- Array
- Union
- Map

Yosegi –How to

Construct the namespace dynamically

Primitive type

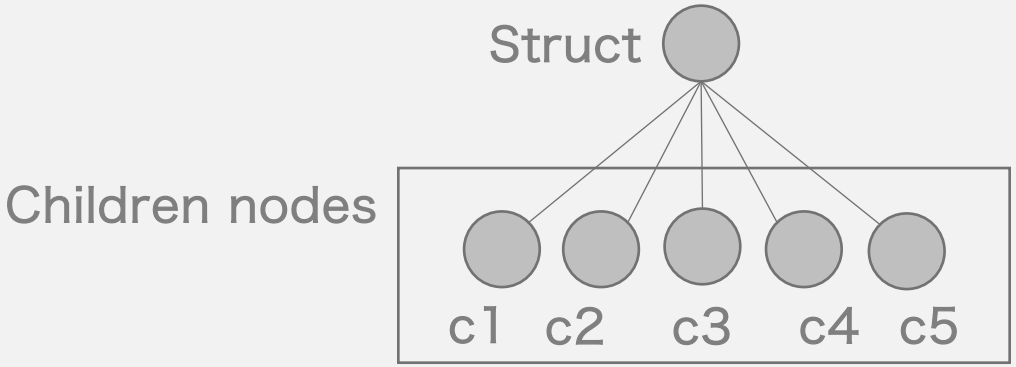
- **Has actual values such as String, Integer, Float**
- **Leaf in the namespace**

Yosegi –How to

Construct the namespace dynamically

Struct type

- **Has member fields as children nodes**

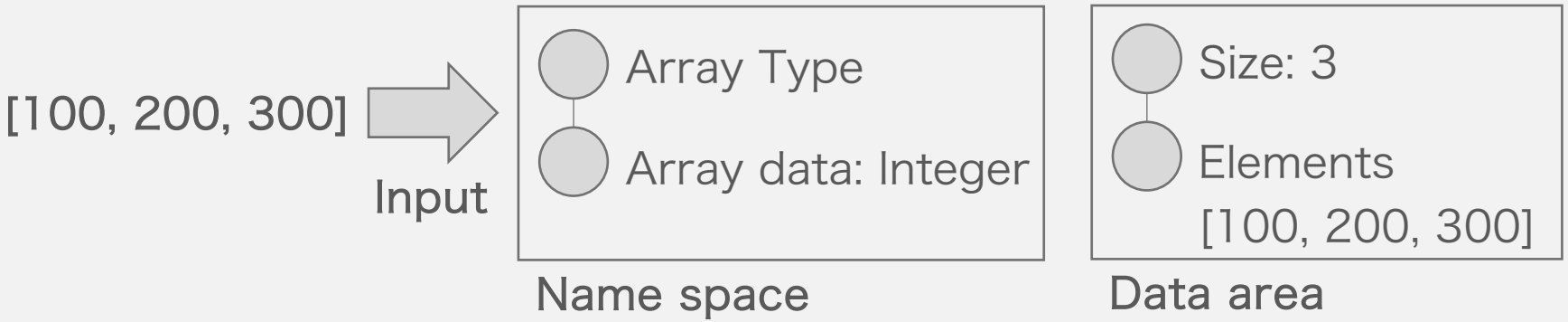


Yosegi –How to

Construct the namespace dynamically

Array type

- **Size of elements in the parent node**
- **Array elements in the child node**

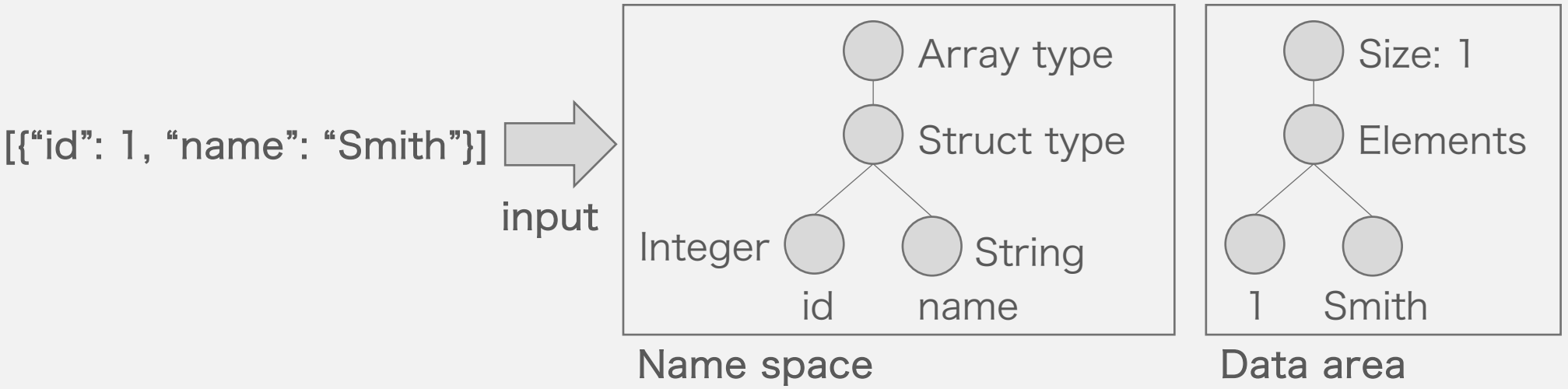


Yosegi –How to

Construct the namespace dynamically

Array type

Example: array elements of struct data



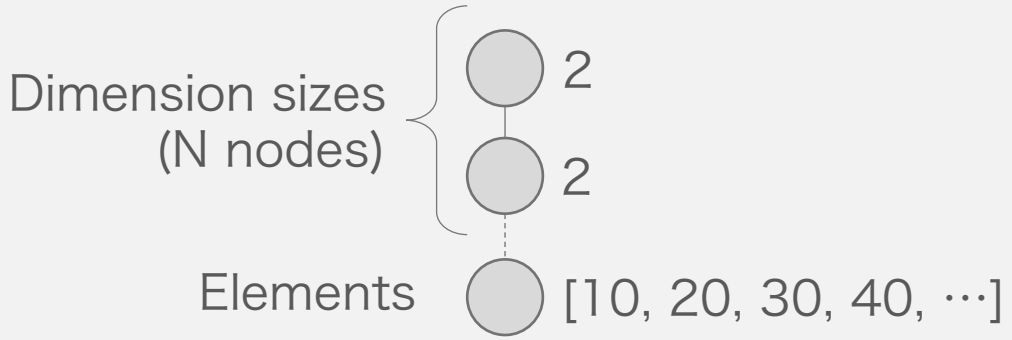
Yosegi –How to

Construct the namespace dynamically

Array type

Example: multidimensional array

- **N nodes for dimension size hang down**
- **Array data in the N+1 th node**



Yosegi –How to

Construct the namespace dynamically

Union type

- **Expressing different data type in the same node**
- **Different data types as children nodes**

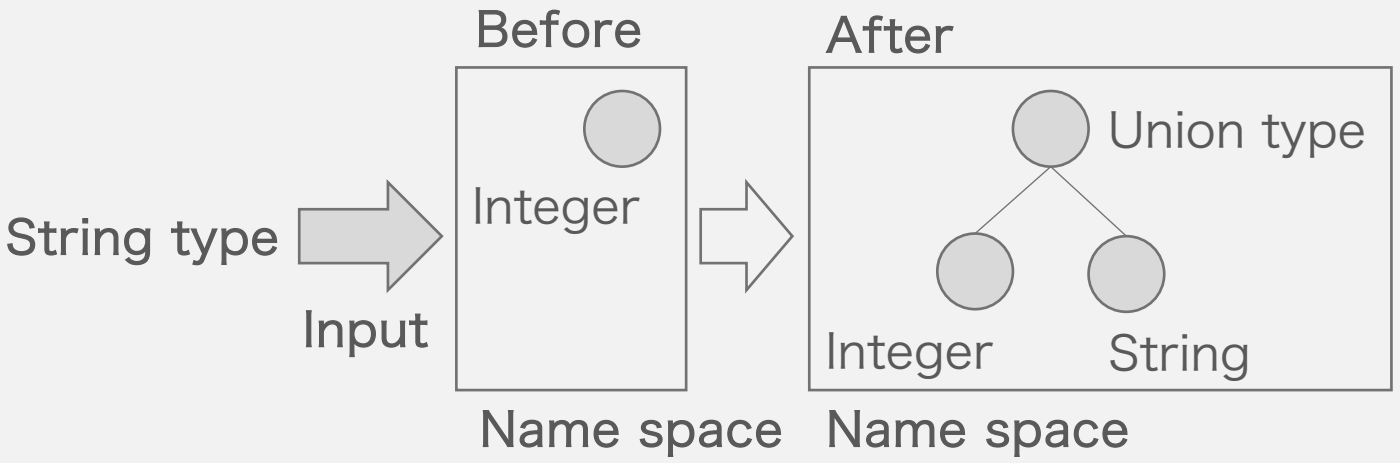
Yosegi –How to

Construct the namespace dynamically

Union type: writing

Create a union type

when the different data type is entered

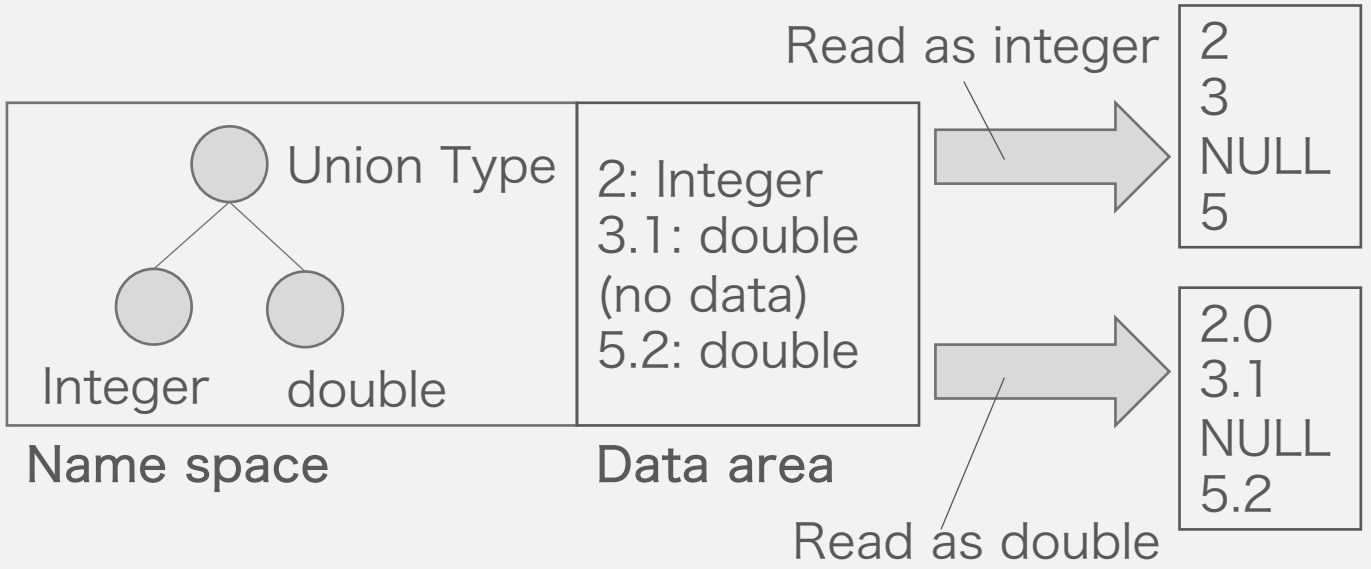


Yosegi –How to

Construct the namespace dynamically

Union type: reading

Cast data according to the schema



Yosegi –How to

Construct the namespace dynamically

Map type: using fields

Writing

- **Expand Map type data to field**

Reading

- **Refer to Map type data whose field name is as key**

Yosegi –How to

Construct the namespace dynamically

Map type: using fields

Advantage

- **Expand only the data used for analysis into memory**
- **Functions for efficiency like Push Down can work**
- **Increase compression rate**

Limitation

- **Key data type limits to String**

Yosegi –How to

Construct the namespace dynamically

Advanced feature

Expanding nested data at the time of reading

- **Expand Array type data**
- **Flatten Struct type data**

Yosegi –How to

(1)

timestamp	total_price	items		
		id	price	number
1516773762	1000	I_001	300	1
		I_005	350	2
1516773765	200	I_002	50	4
1516773770	500	I_002	50	1
		I_005	350	1
		I_009	50	2

(2)

timestamp	total_price	items		
		id	price	number
1516773762	1000	I_001	300	1
1516773762	1000	I_005	350	2
1516773765	200	I_002	50	4
1516773770	500	I_002	50	1
1516773770	500	I_005	350	1
1516773770	500	I_009	50	2

(3)

timestamp	total_price	items .id	items .price	items .number
1516773762	1000	I_001	300	1
1516773762	1000	I_005	350	2
1516773765	200	I_002	50	4
1516773770	500	I_002	50	1
1516773770	500	I_005	350	1
1516773770	500	I_009	50	2

Expanding nested data

Original

```
CREATE EXTERNAL TABLE `base`(
  `timestamp` string,
  `total_price` int,
  `items` array<struct<id:string,price:int,number:int>>
)
```

Expand Array type data

```
CREATE EXTERNAL TABLE `base_expand`(
  `timestamp` string,
  `total_price` int,
  `items` struct<id:string,price:int,number:int>
) ...
TBLPROPERTIES (
  'yosegi.expand' = '{"base": {"node":"items","link_name":"items"}}'
);
```

Flatten Struct type data

```
CREATE EXTERNAL TABLE
`yoto.test_expand_flatten`(
  `timestamp` string,
  `total_price` int,
  `items.id` string,
  `items.price` int,
  `items.number` int
)...

TBLPROPERTIES (  'yosegi.expand' = '{
  "base": {"node":"items", "link_name":"items"}}'
  "yosegi.flatten" = '[
    {"link_name": "timestamp", "nodes": ["timestamp"] },
    {"link_name": "total_price", "nodes": ["total_price"] },
    {"link_name": "items.id", "nodes": ["items", "id"] },
    {"link_name": "items.price", "nodes": ["items", "price"] },
    {"link_name": "items.number", "nodes": ["items", "number"] }
  ]');
```

Yosegi –How to

Expanding nested data

Usage

- Mapping the field in the schema with the node in namespace

Advantage

- Get the nested data in expanded form
- Functions at query engine for efficiency like Push Down and vectorization can work

Performance

Performance

Evaluation method

- Execute Hive query on data in HDFS
- Compare Yosegi and existing columnar formats

Query execution node

- 32 core CPU
- 128 GB memory
- Physical server

Performance

Performance evaluation

- Basic performance evaluation
- Comparison using the data from Yahoo! JAPAN

Performance

Basic performance evaluation

Data

TPC-H scale 1000 GB

Performance

Basic performance evaluation

Formats for comparison (compress mode: ZIP)

- Yosegi
- ORC
- Parquet

Performance

Basic performance evaluation

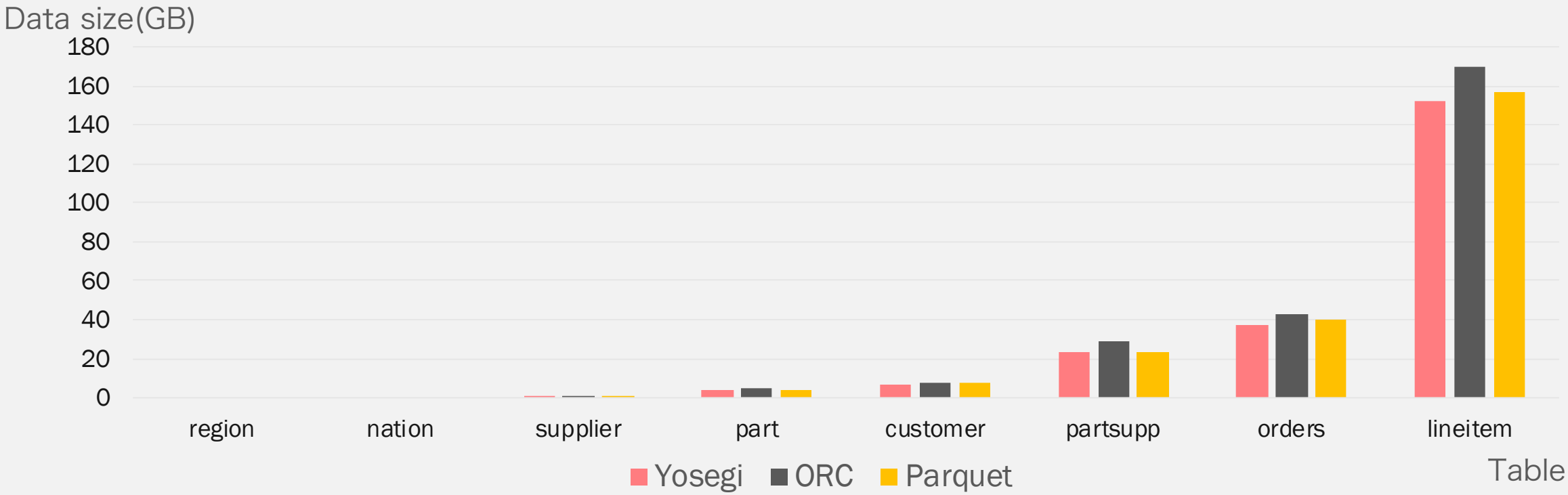
Metrics for comparation

- Data size for storage
- CPU time for reading
- CPU time for writing

Performance

Basic performance evaluation: data size

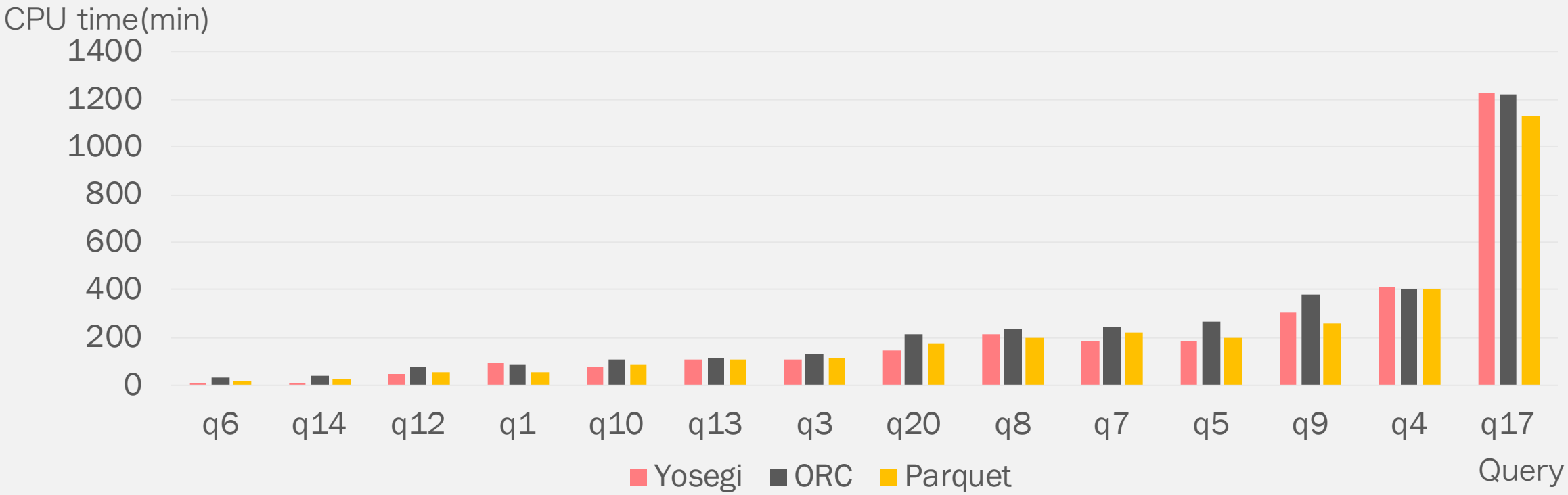
Same or smaller than existing formats



Performance

Basic performance evaluation: CPU times for reading

Same as existing formats

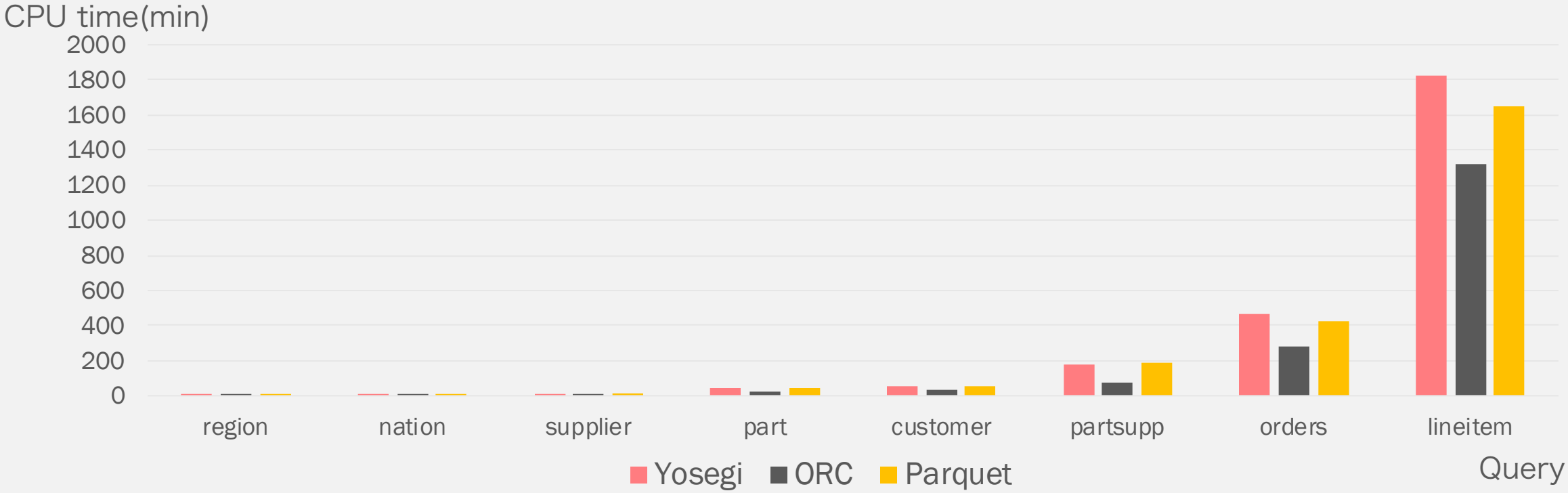


Performance

Basic performance evaluation: CPU times for writing

Same as Parquet

ORC advance seems it uses lower compression level



Performance

Comparison using the data from Yahoo! JAPAN

Data

Action log of the user who accessed the web page

contains multiple links in an array format

Performance

Comparison using the data from Yahoo! JAPAN

Formats for comparison (compress mode: ZIP)

- Yosegi
- ORC

Performance

Comparison using the data from Yahoo! JAPAN

Work load

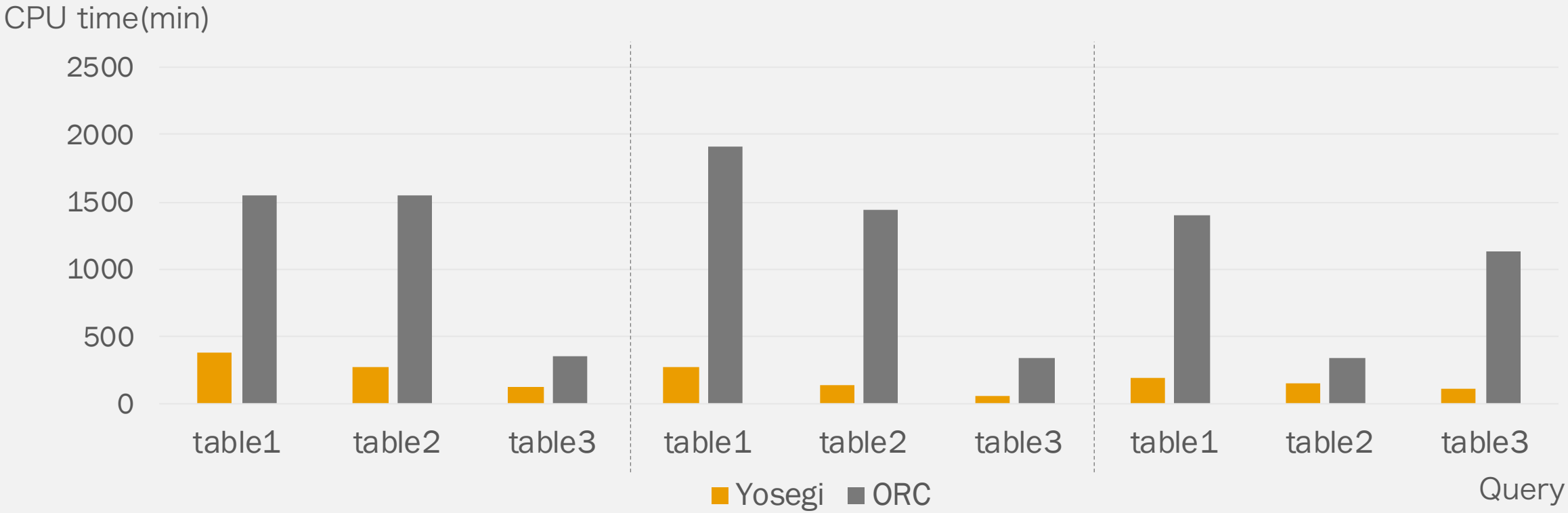
- **Three commonly used query patterns**
- **Aggregate web log data for one day**
  (using GROUP BY)
- **Use Expand & Flatten (Yosegi)**

Performance

Comparison using the data from Yahoo! JAPAN: execution time

Query execution is 2 to 10 times faster

than the existing format

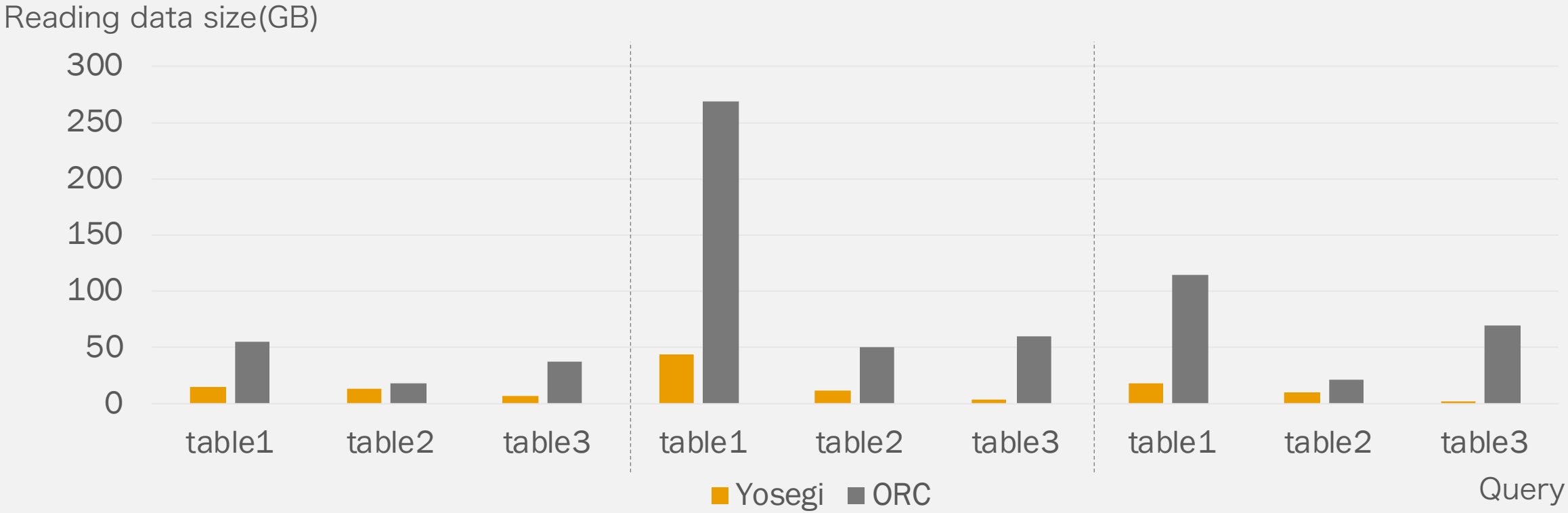


Performance

Comparison using the data from Yahoo! JAPAN: reading data size

Reading data size in Hive is widely reduced

Good efficiency of utilization



Performance

Summary

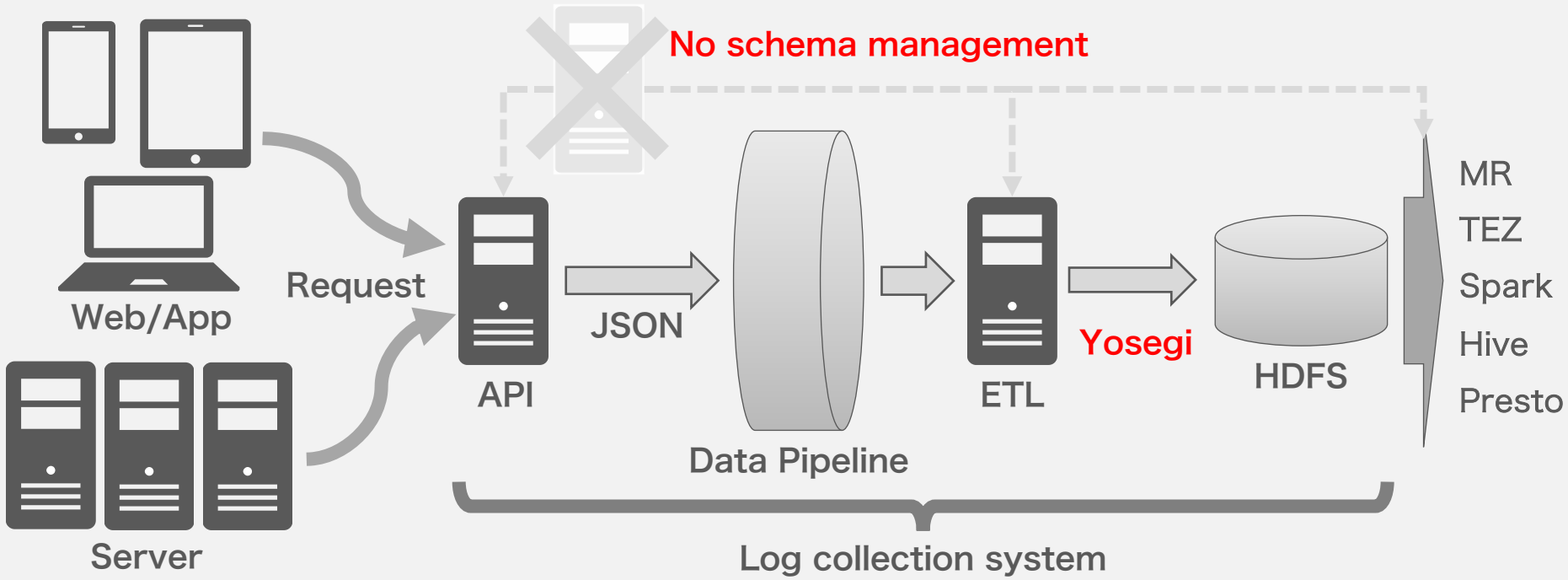
- **Basic performance is as same as existing formats**
- **Nest data reading is better than existing format**

  Using functions for efficiency
  - Push down for Array type data
  - Vectorization (multiple record processing)

Use case

Use case

Data format for log collection system of Yahoo! JAPAN



Use case

Achievement

- Flexibility for changing structure of inputs
- Compact storage data
- Efficient use at the time of analysis
- No schema management

  then, be able to create a simple system

Conclusion

Conclusion

Yosegi

- Realize schema less columnar format
  by creating the namespace dynamically

Achievement

- Flexibility for changing structure of input data
- Efficiency for usage and storage

Conclusion

Performance

- Same performance as existing formats about storage data size and utilization
- Nest data reading is better
  than the existing format

Conclusion

Use case

- The log collection system of Yahoo! JAPAN without schema management

Announce

Join us for contributor

Yosegi is a OSS

https://github.com/yahoojapan/yosegi

Please visit our GitHub and contact with us

Presentations from Yahoo! JAPAN

Session: Yosegi

https://github.com/yahoojapan/yosegi

Booth: AntPickax

https://antpick.ax/index.html

https://github.com/yahoojapan/AntPickax

Please visit our booth