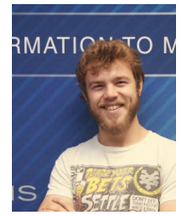
Tutorial: An Outlook to Declarative Languages for Big Steaming Data

EBDT 2020

Riccardo Tommasini, Hojjiat Jafarpour,
Sherif Sakr, and Emanuele Della Valle



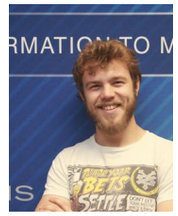
Hojjiat Jafarpour

Riccardo Tommasini

Emanuele Della Valle

Sherif Sakr

Who we are



1979
2020

Agenda

- Introduction on Stream Processing Models
- Declarative Language: Opportunities, and Design Principles
- Comparison of Prominent Streaming SQL Dialects for Big Stream Processing Systems
- Conclusion

Unbounded yet time-ordered sequence of data

Jennifer Widom

Stream Processing 101

- Data Stream Management Systems (DSMS)
  - Found origin within databased community
  - focus on continuous query answering and analytics
    - Reference Models inspired by Relational Algebra: CQL, Secret
- Complex Event Processing (CEP)
  - Found origin within software engineering community
  - focus on continuous detection of patterns
  - Reference Models inspired by regular languages: NFA, SNOOP

Stream Processing 101

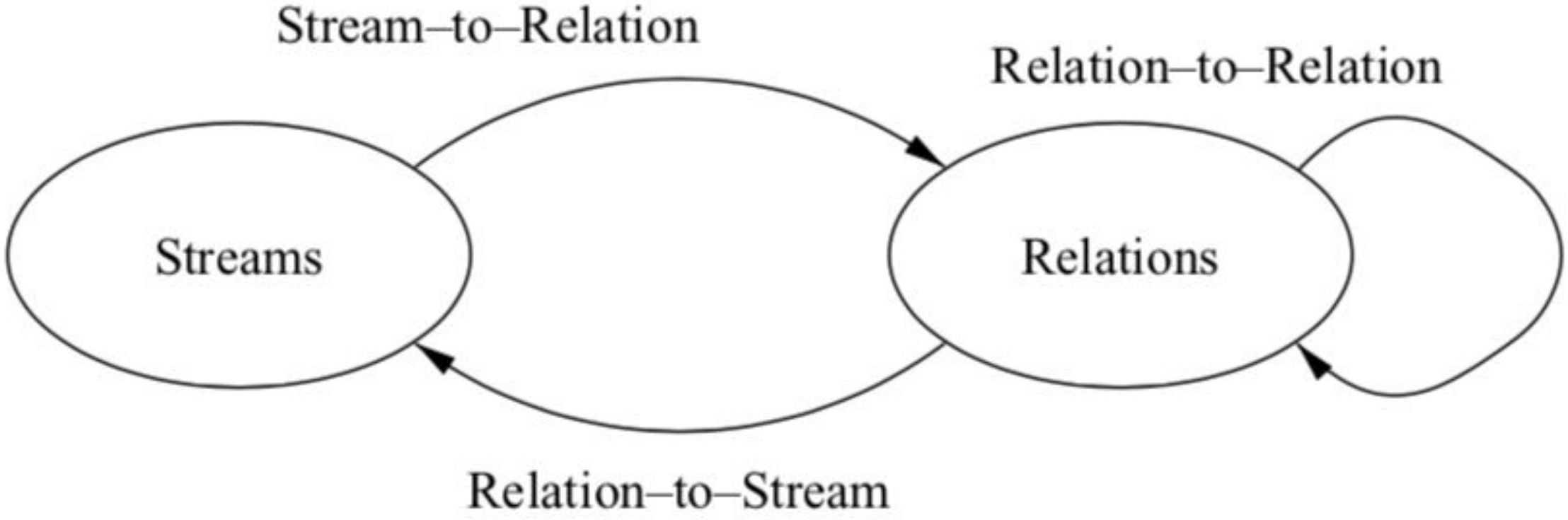
- Data Stream Management Systems (DSMS)
  - Found origin within databased community
  - focus on continuous query answering and analytics
    - Reference Models inspired by Relational Algebra: CQL, Secret
- ~~Complex Event Processing (CEP)~~
  - ~~Found origin within software engineering community~~
  - ~~focus on continuous detection of patterns~~
  - ~~Reference Models inspired by regular languages: NFA, SNOOP~~

Time Management

- **Processing Time** (consumer) implies a total order on the stream.

- **Event Time** (producer), implies a partial order on the data.

DSMS (CQL)


Stream–to–Relation
Relation–to–Relation
Streams
Relations
Relation–to–Stream

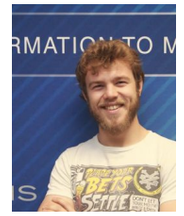
DSMS (CQL)

- Expressive Languages (SQL++)
- Windowing
  - *Canonical*: logical/physical sliding/tumbling
  - *Custom*: session, data-driven, event-driven

DSMS (CQL)

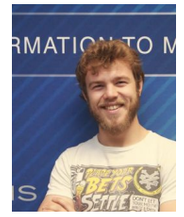
CREATE SCHEMA Stock(id string, price float, timestamp int)

SELECT avg(price)
FROM Stock#**time**(10 minutes)
OUTPUT EVERY 1 minutes
GROUP BY id



Big Stream Processing

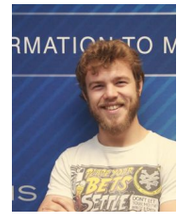
Ecosystem



Big Stream Processing

Fully-Distributed Systems

- Fault-Tolerant:
  - **At-Least-Once** or **Exactly-Once** semantics
- Scalable (millions of tuples per minute)
- Flexible Programmatic API that guides towards the creation of Direct Acyclic Graph



Declarative Languages [CQL]

KSQL,FlinkSQL, SparkSQL

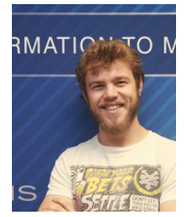
Functional API

DataFrames, KafkaStreams,
Flink Stream API

Dataflow Model

Kafka Processor API, Flink Process Function

Actor Model [Hewit et al.]



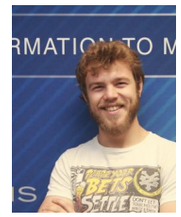
Big Stream Processing

Languages

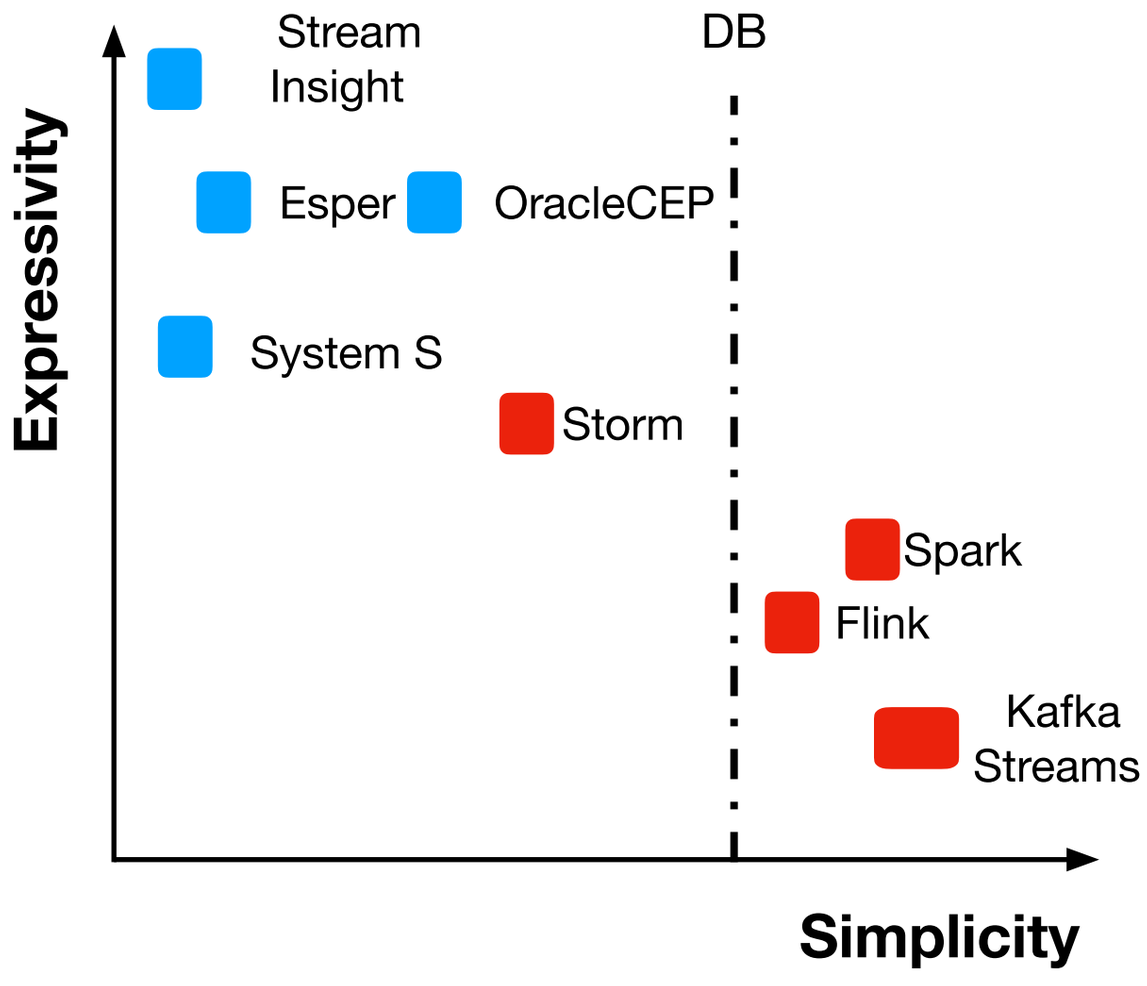
- Offer languages that are embedded in a general-purpose host language, typically Java

- Encourage developers to **explicitly** code a Direct-Acyclic Graph

- Provide relational operators, but also expose low-level details such as partitioning, timestamp extraction

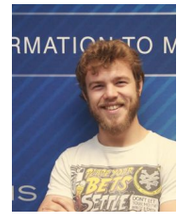


Solution Landscape (qualitative)



Big Stream Processing Solutions

Single Machine



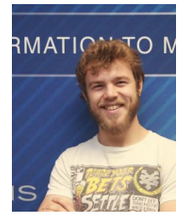
Big Stream Processing

(Issues)

- Distributions makes out-of-order handling a primary problem, and, thus solutions appears in the programmatic APIs.

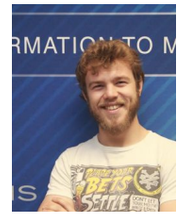
- Languages are not self-contained, thus, are **hard to isolate** clearly from the host language

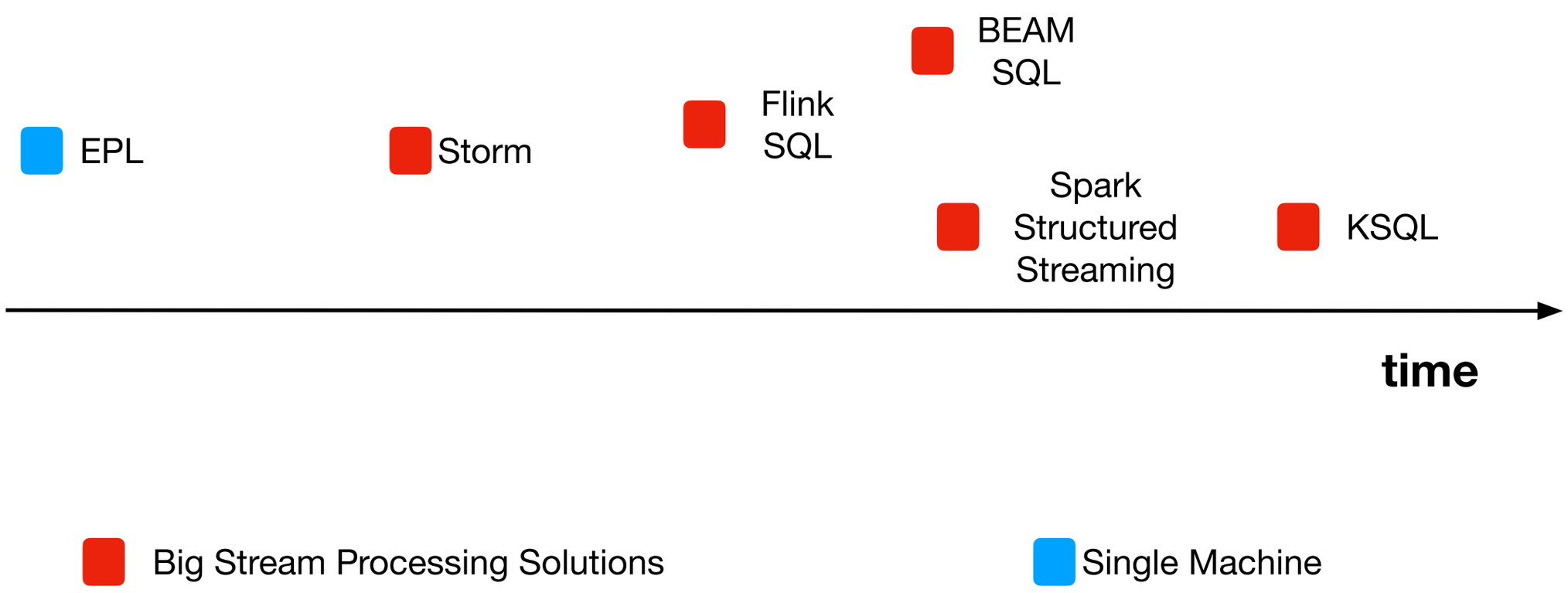
- Debugging, benchmarking, and standardisation becomes hard

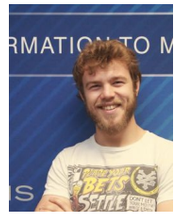


Major systems started migrating towards a fully-declarative approach ultimately evolved into SQL-like streaming DSL.

Adoption of SQL-like interface







One SQL to Rule Them All: An Efficient and Syntactically Idiomatic Approach to Management of Streams and Tables

An Industrial Paper

Edmon Begoli
Oak Ridge National Laboratory / Apache Calcite
Oak Ridge, Tennessee, USA
begoli@apache.org

Tyler Akidau
Google Inc. / Apache Beam
Seattle, WA, USA
takidau@apache.org

Fabian Hueske
Ververica / Apache Flink
Berlin, Germany
fhueske@apache.org

Julian Hyde
Looker Inc. / Apache Calcite
San Francisco, California, USA
jhyde@apache.org

Kathryn Knight
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
knightke@ornl.gov

Kenneth Knowles
Google Inc. / Apache Beam
Seattle, WA, USA
kenn@apache.org

ABSTRACT

Real-time data analysis and management are increasingly critical for today's businesses. SQL is the de facto *lingua franca* for these endeavors, yet support for robust streaming analysis and management with SQL remains limited. Many approaches restrict semantics to a reduced subset of features and/or require a suite of non-standard constructs. Additionally, use of event timestamps to provide native support for analyzing events according to when they actually occurred is not pervasive, and often comes with important limitations.

We present a three-part proposal for integrating robust streaming into the SQL standard, namely: (1) time-varying relations as a foundation for classical tables as well as stream-

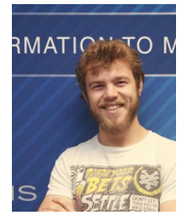
CCS CONCEPTS

• **Information systems** → **Stream management**; **Query languages**;

KEYWORDS

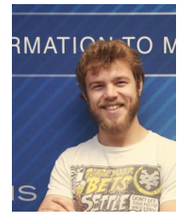
stream processing, data management, query processing

ACM Reference Format:
Edmon Begoli, Tyler Akidau, Fabian Hueske, Julian Hyde, Kathryn Knight, and Kenneth Knowles. 2019. One SQL to Rule Them All: An Efficient and Syntactically Idiomatic Approach to Management of Streams and Tables: An Industrial Paper. In *2019 International Conference on Management of Data (SIGMOD '19), June 30–July 5*



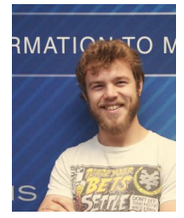
Agenda

- Introduction on Stream Processing Models [done]
- Declarative Language: Opportunities, and Design Principles
- Comparison of Prominent Streaming SQL Dialects for Big Stream Processing Systems
- Conclusion



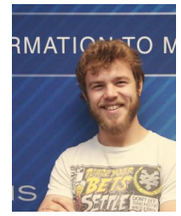
“The key idea of declarative programming is that a **program** is a **theory** in some suitable **logic,** and the **computation** is **deduction** from the **theory**”

–J.W. Lloyd



Advantages

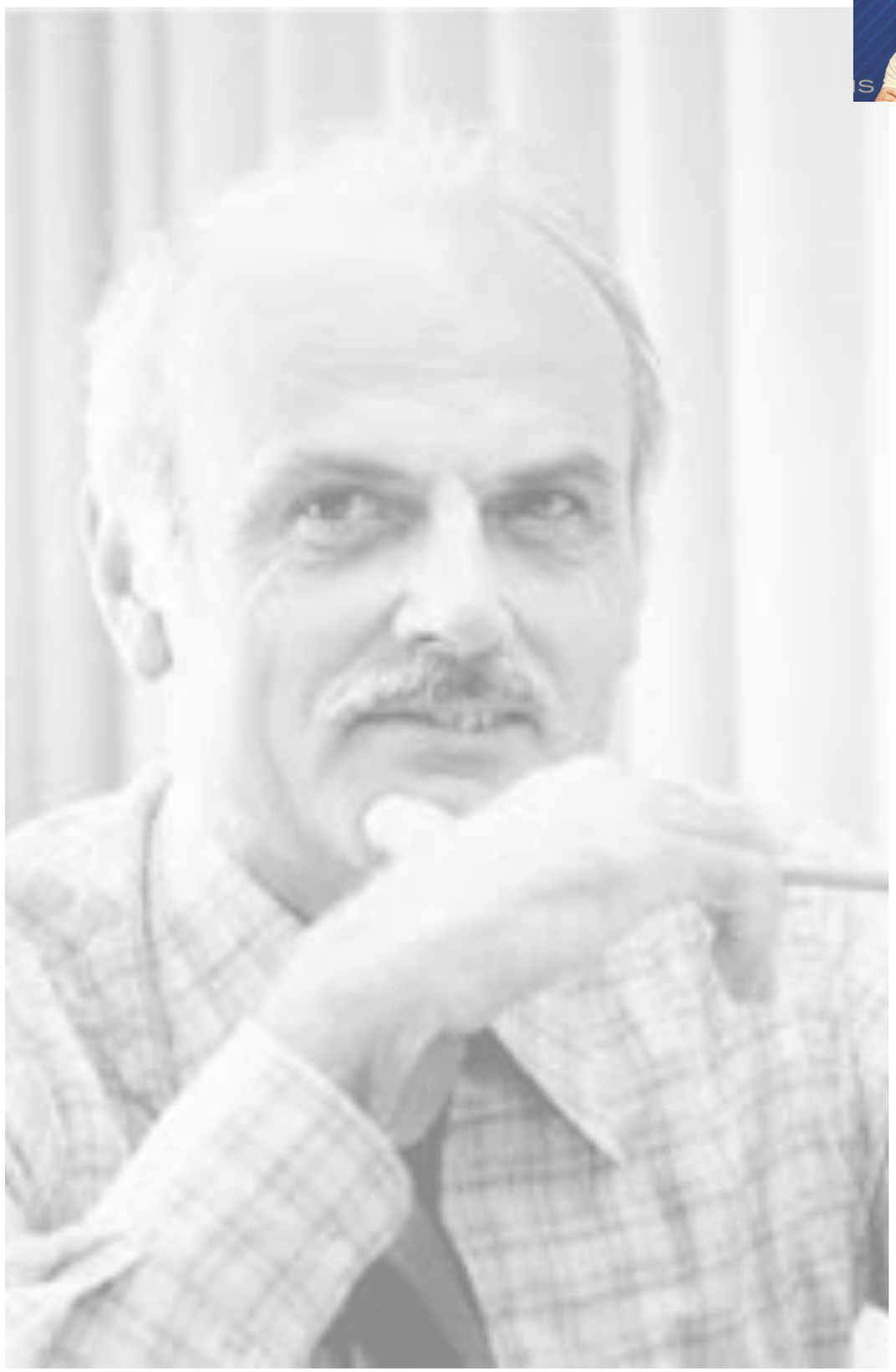
- Decouple interpretation and execution (e.g. parallelism)
- Allows optimisation relying on the formal semantics
- **IDEALLY PORTABLE (well-defined semantics)**

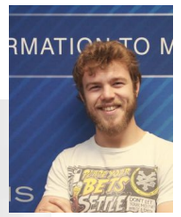


How to design a good language?

Minimality

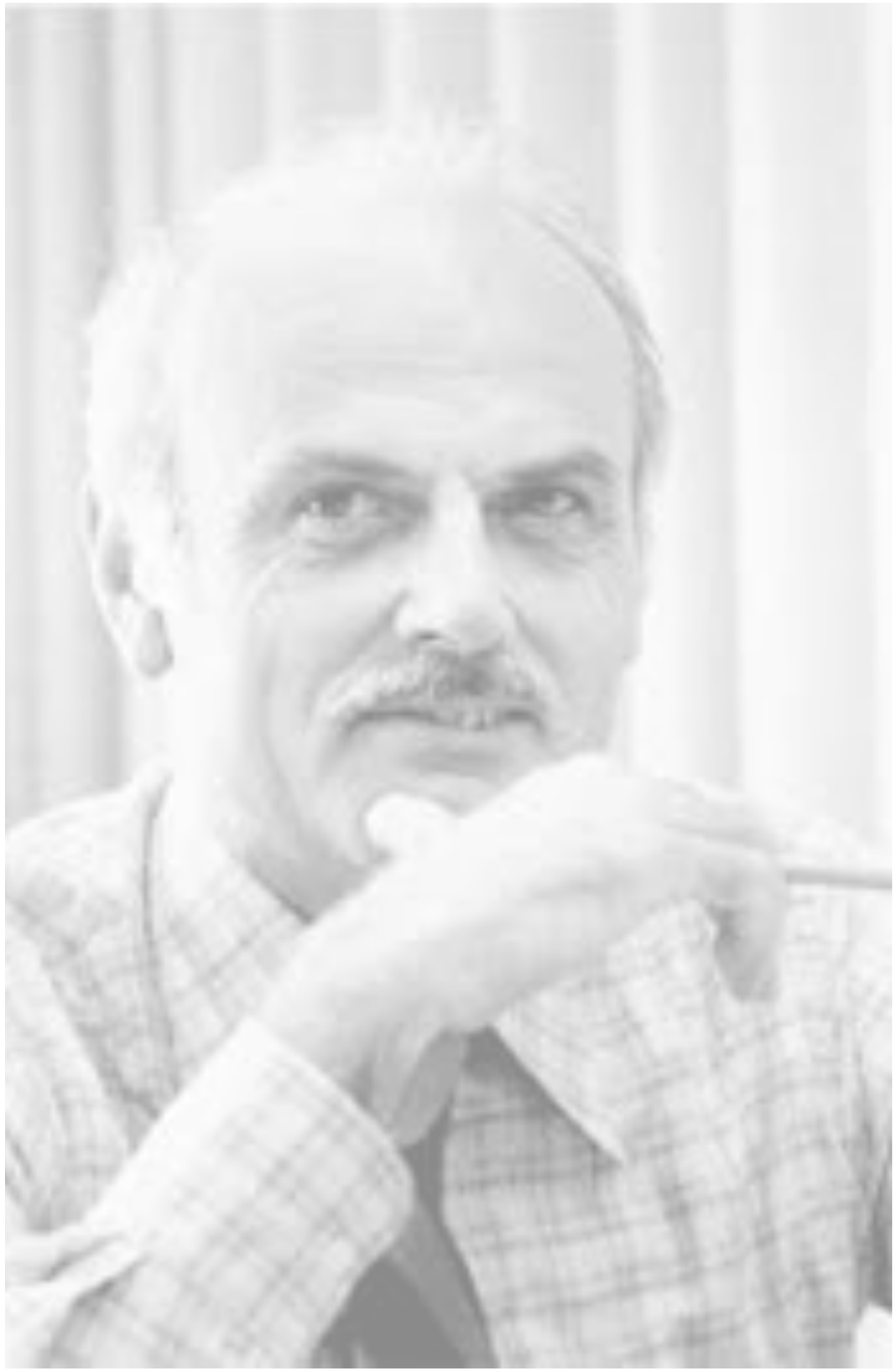
a language should provide only a **small set** of needed **language constructs** so that the same meaning **cannot** be **expressed** by **different** language constructs;





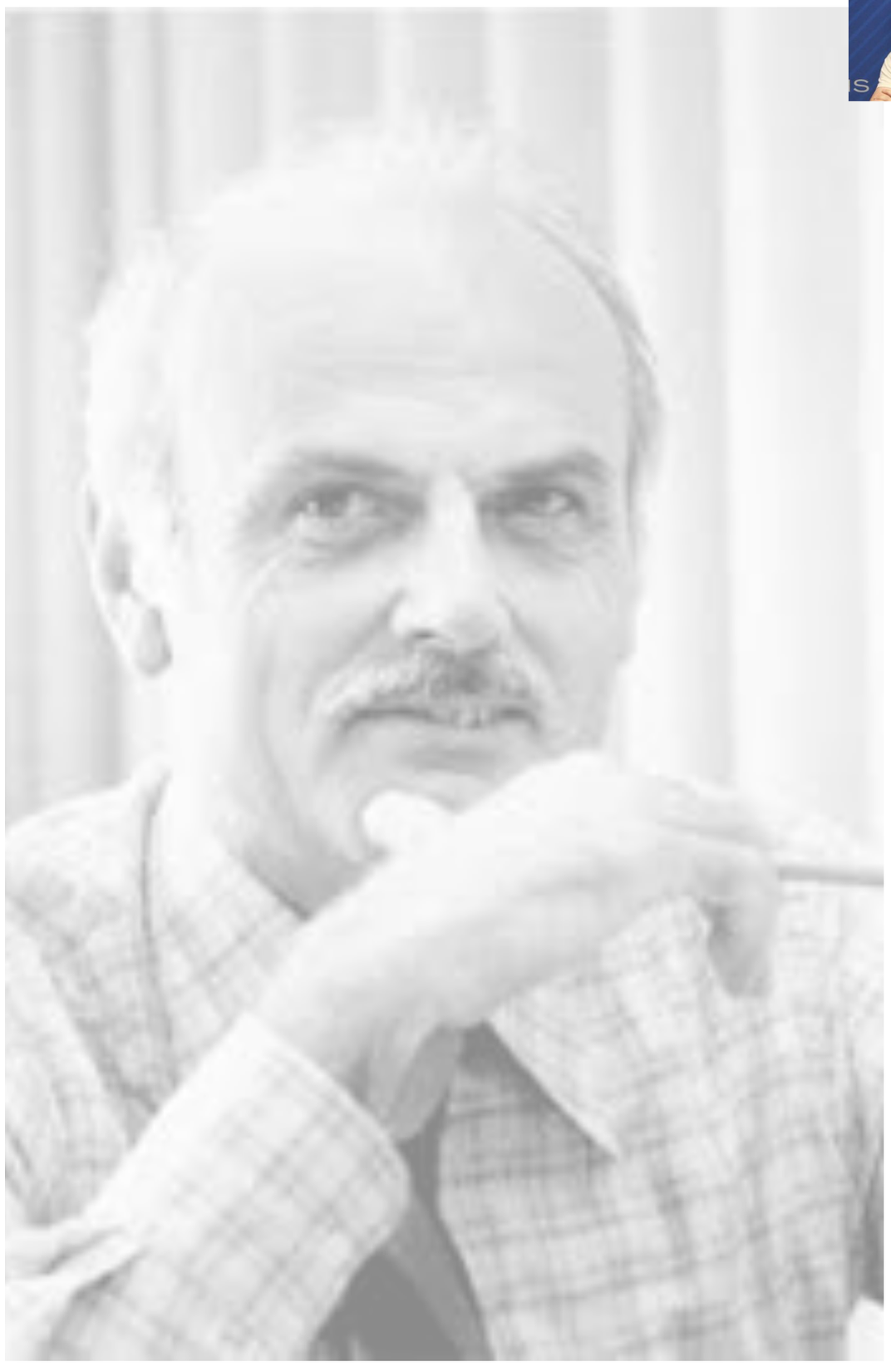
Symmetry

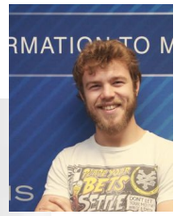
a language should **ensure** that the same language **construct** **always** expresses the same **semantics** **regardless** of the context

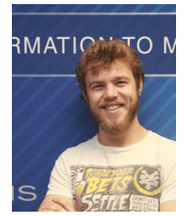


Orthogonality

a language should **guarantee** that every meaningful **combination** its **constructs** is applicable.





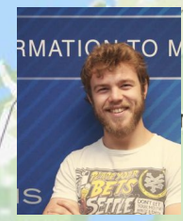


When do we need it?

- Writing the optimal solution is as hard as solving the problem (e.g. JOIN optimisation)

- We want to enhance programmer productivity by adding Domain-Specific abstraction (e.g. streams)

- We want to limit the expressiveness of the languages to ensure some nice property (e.g. decidability)



11 h 25 min
from €1,437

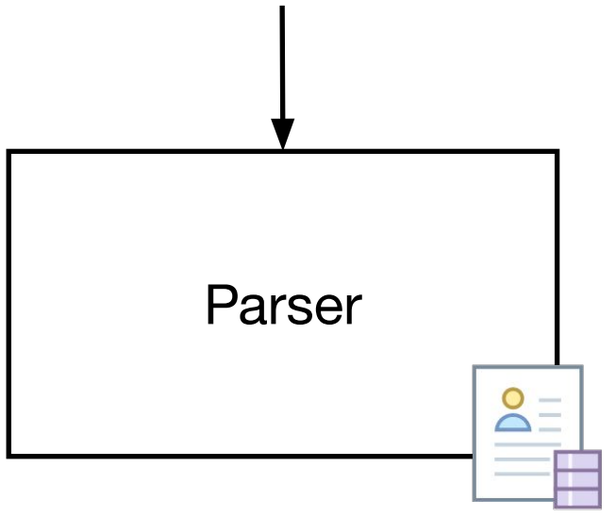
San Francisco International Airport

Darmstadt

Iceland
Sweden
Norway
United Kingdom
Poland
France
Italy
Spain
Canada
Mexico
North Atlantic Ocean
Venezuela
Colombia
Brazil
Peru
Bolivia
Algeria
Libya
Egypt
Mali
Niger
Chad
Nigeria
Sudan
DRC
Angola
Namibia

Arctic Ocean
Barents Sea
Bering Sea
Greenland Sea
Baffin Bay
Iceland
Greenland
Sweden
Belarus
Poland
United Kingdom
Ireland
Italy
Spain
Portugal
North Pacific Ocean
Canada
Northwestern Passages
Hudson Bay
AK
YT
NT
NU
BC
AB
SK
MB
ON
WA
OR
ID
MT
ND
MN
NV
WY
SD
UT
WI
MI
IA
IL
IN
OH
PA
NY
NH
MA
ME
NB
PE
NS
AZ
KS
MO
NM
OK
AR
KY
WV
DE
VA
TN
NC
TX
MS
AL
SC
LA
GA
FL
United States
Mexico
Gulf of Mexico
Cuba
Puerto Rico
Guatemala
Nicaragua
Venezuela
North Atlantic Ocean
11 h 25 min
from €1,437
San Francisco International Airport

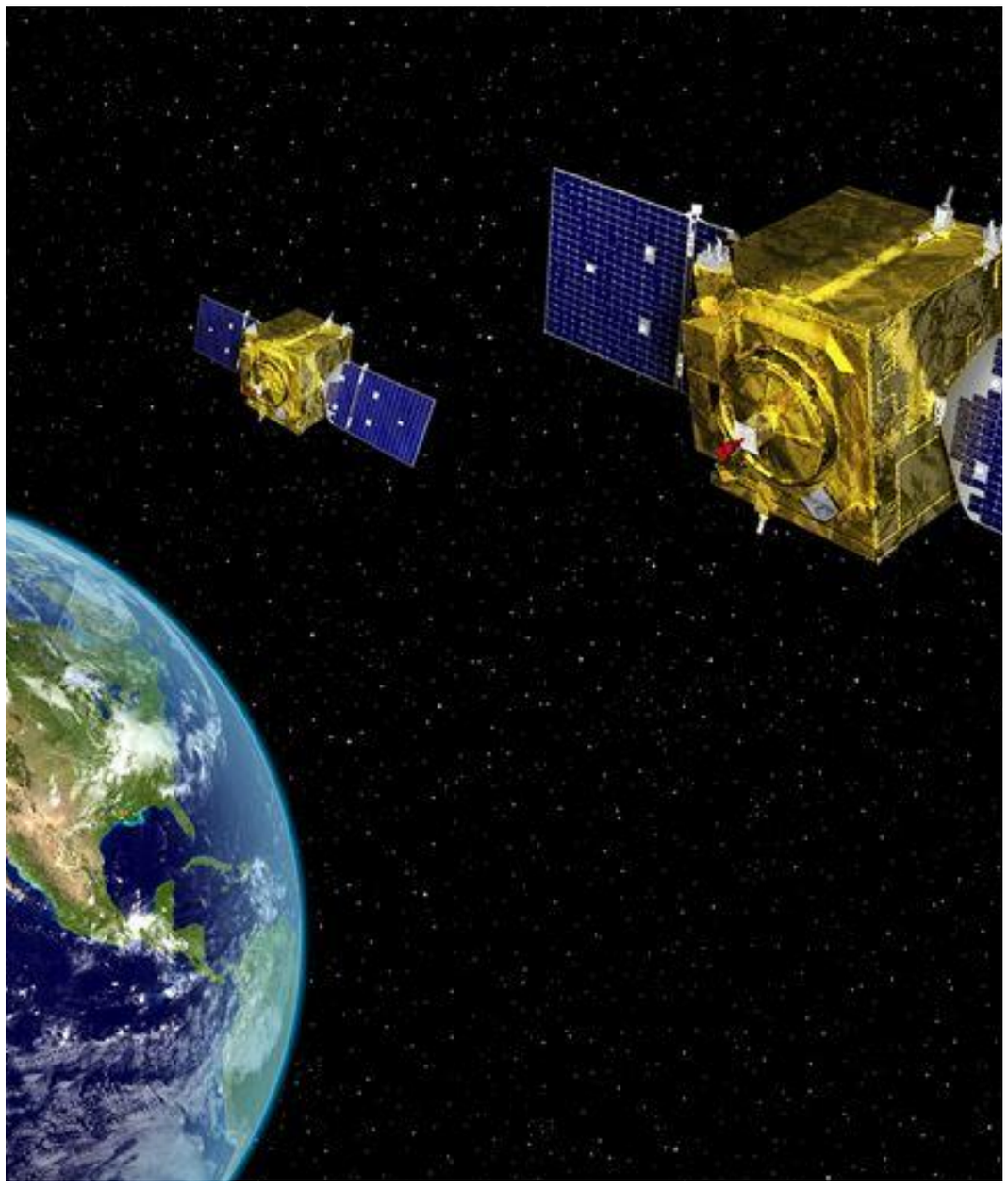
Program/Query

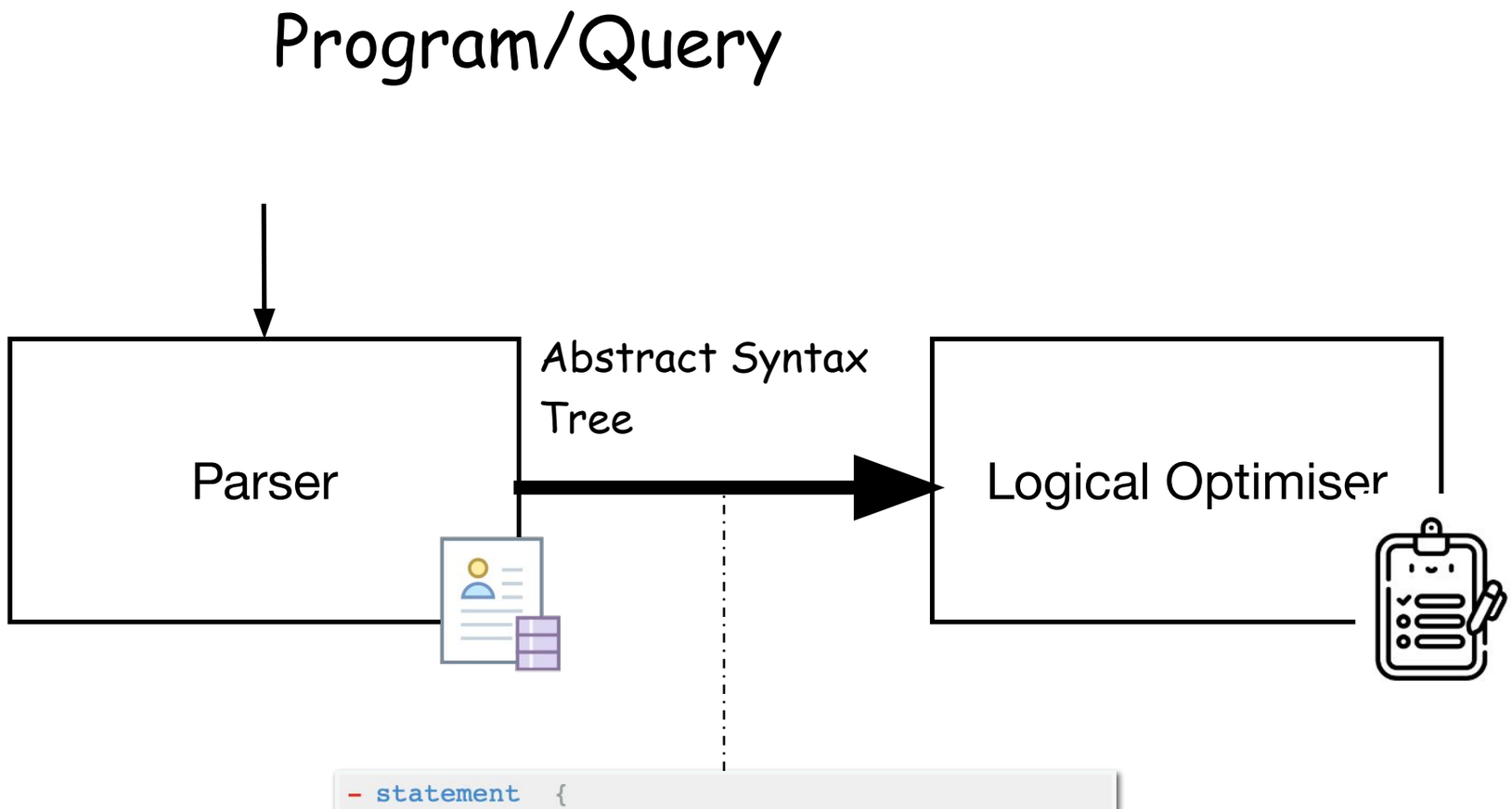



```
SELECT name

FROM (

    SELECT id, name

    FROM People) p

WHERE p.id = 1
```

Parsing

- Obtaining the Declarative Program/Query
- Verify it is is syntactically valid
- Creating an AST

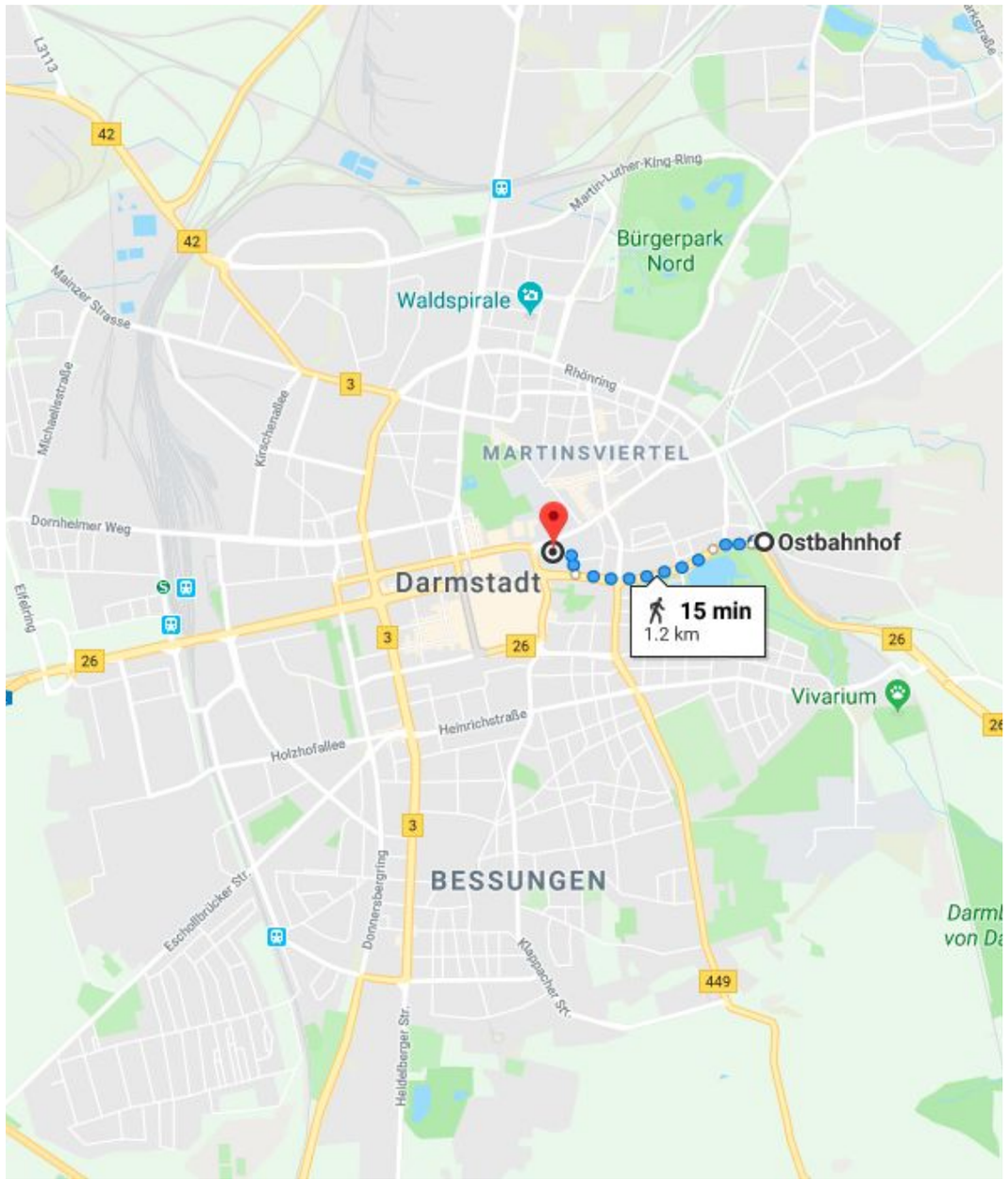


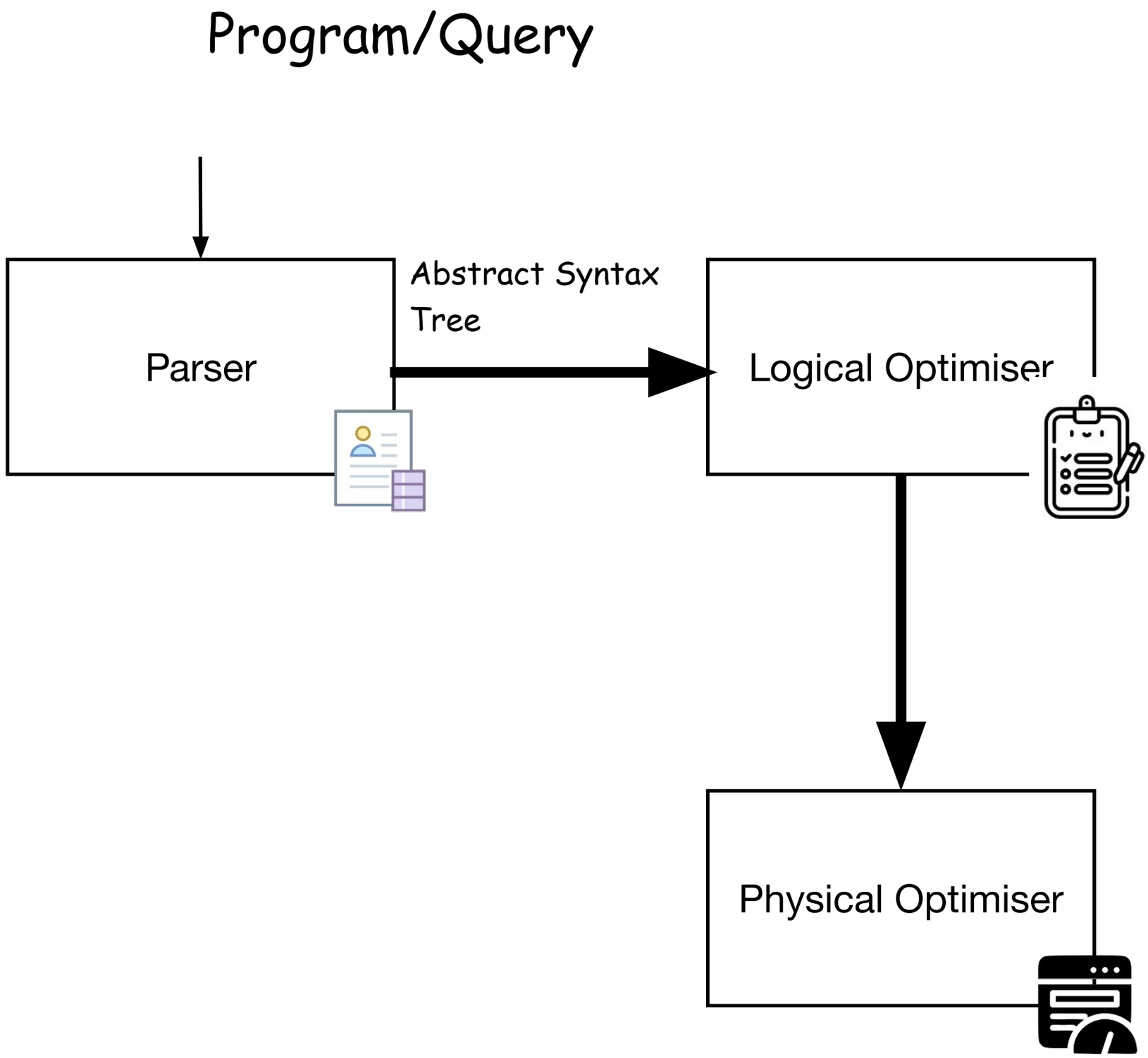



```
- statement  {
    type: "statement"
    variant: "list"
  - statement:  [
    - statement  {
        type: "statement"
        variant: "select"
      - result:  [
          + identifier {type, variant, name}
          + identifier {type, variant, name, ali
        ]
      + from: identifier {type, variant, name}
      - where:  [
          + expression {type, format, variant, o
        ]
      - order:  [
          + expression {type, variant, expressio
```

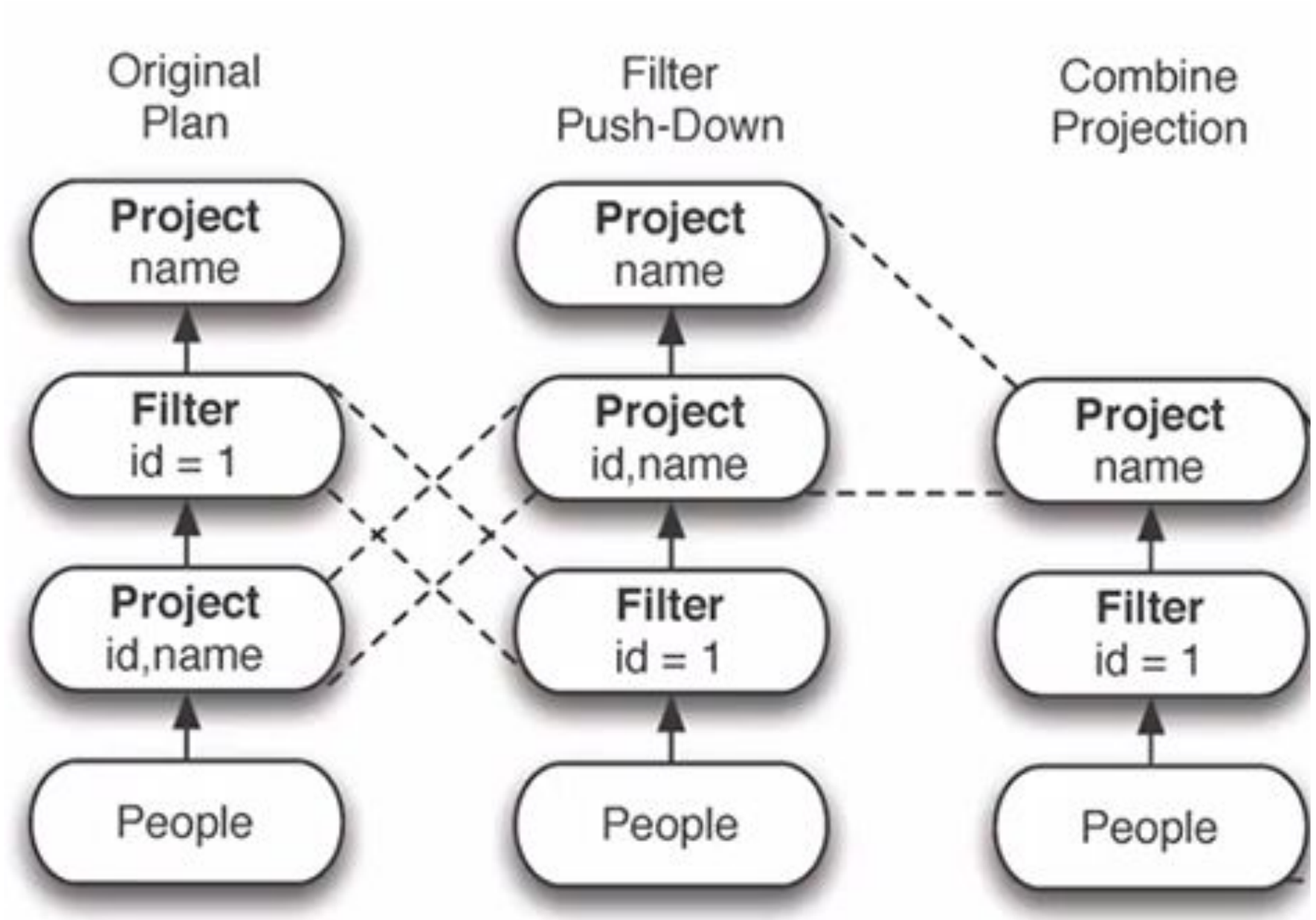
Logical Planning

- Obtaining the AST of the program/query
- Verify all the preconditions hold
- Apply optimisations
- Errors: statistics not updated, wrong decision
- Generates logical plan



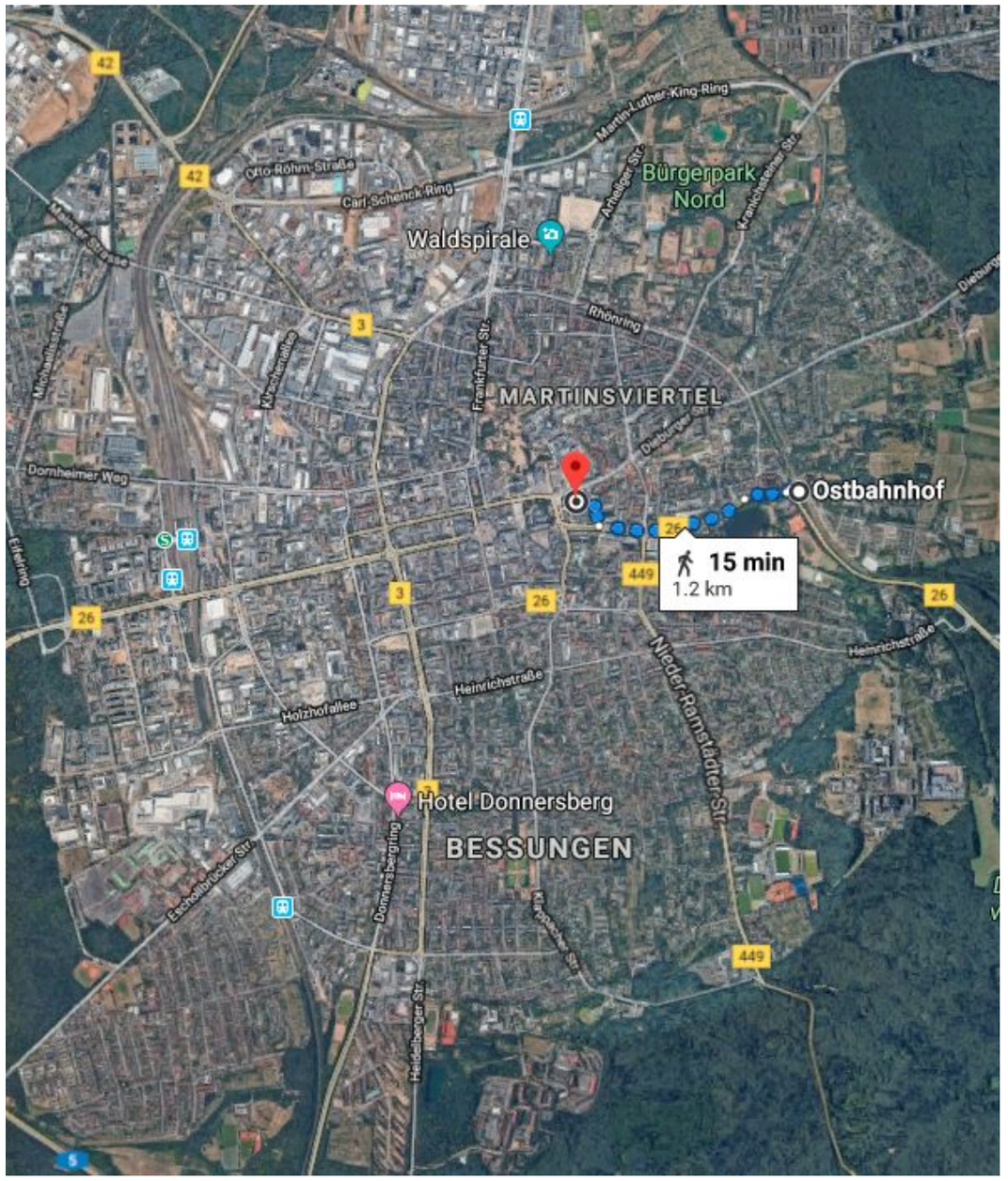

Program/Query
Parser
Abstract Syntax
Tree
Logical Optimiser
Physical Optimiser

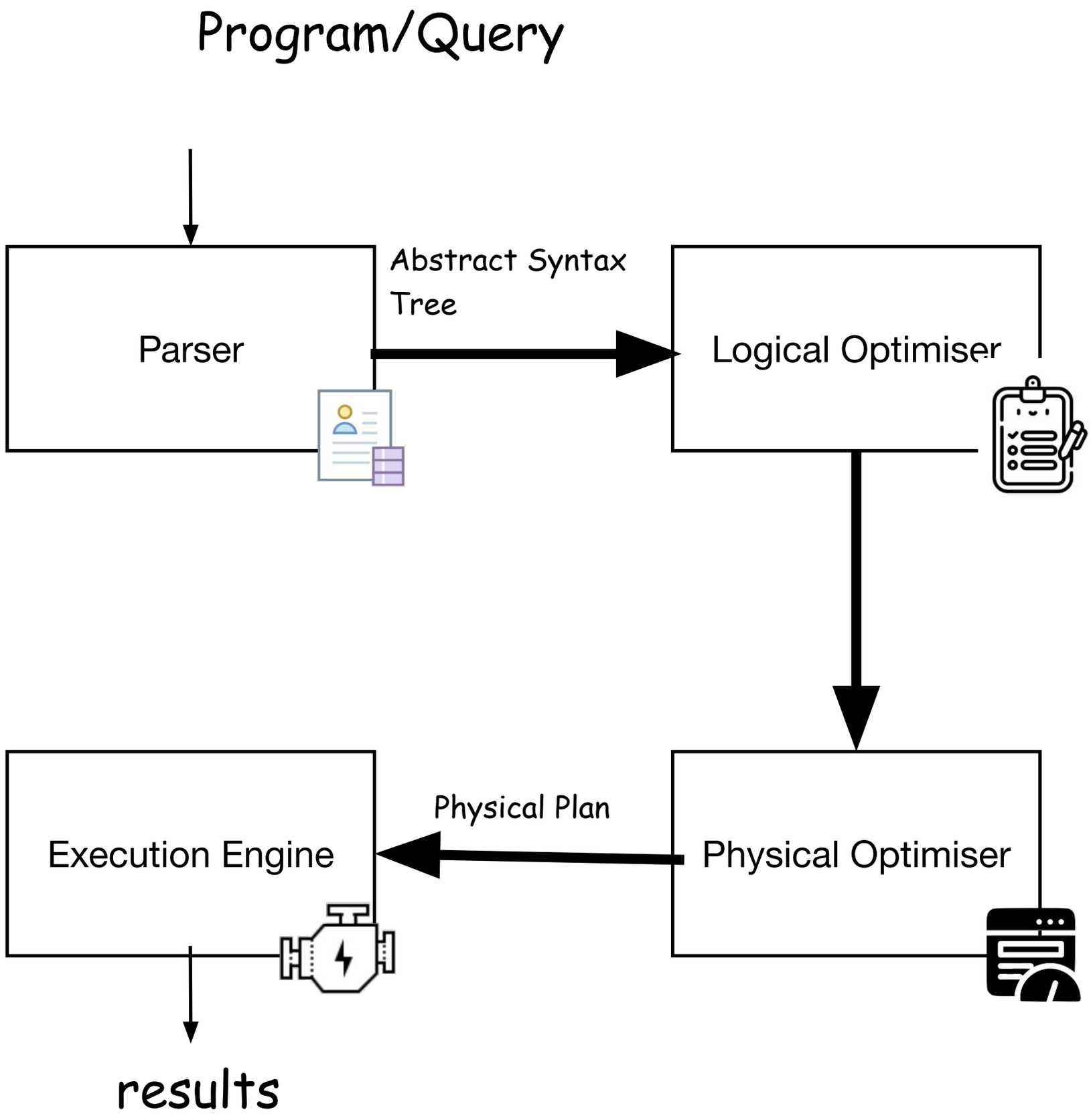
Example


Original Plan
Filter Push-Down
Combine Projection
Project name
Filter id = 1
Project id,name
People
Project name
Project id,name
Filter id = 1
People
Project name
Filter id = 1
People

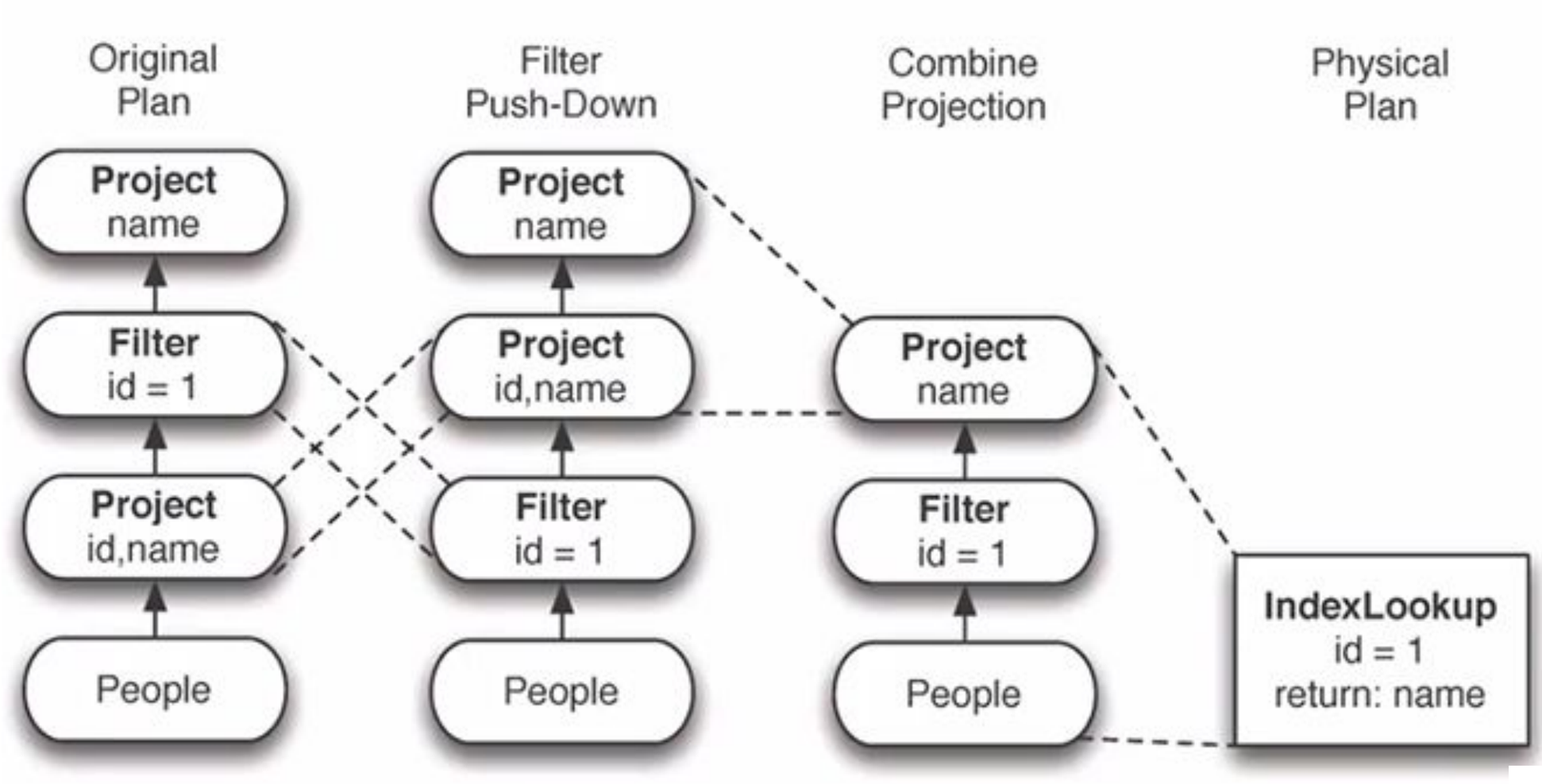
Physical Planning

- Obtaining the logical plan of the program/query
- Verify all the preconditions
- Errors: table not exists
- Generates physical plan




Program/Query
Parser
Abstract Syntax Tree
Logical Optimiser
Physical Optimiser
Physical Plan
Execution Engine
results

Example

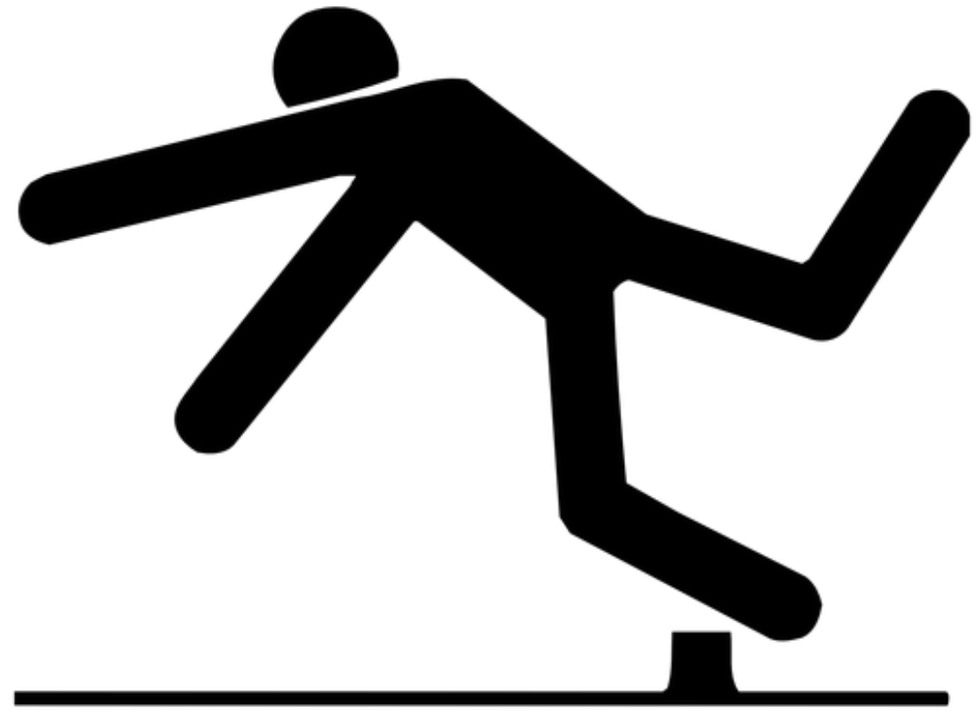

Original Plan
Filter Push-Down
Combine Projection
Physical Plan
Project name
Filter id = 1
Project id,name
People
Project id,name
Filter id = 1
People
Project name
Filter id = 1
People
IndexLookup id = 1 return: name

Executing

- Obtain physical plan of the query
- Load it for execution
- Run!

Runtime Errors

- Input not compliant to the expected one
- Table dropped while running
- Network fail (fixable)
- Node fail (fixable)



Agenda

- Introduction on Stream Processing Models [done]
- Declarative Language: Opportunities and Design Principles [done]
- Comparison of Prominent Streaming SQL Dialects for Big Stream Processing Systems
- Conclusion

Our Focus

- Prominent Big Stream Processing Engines that offer a declarative SQL-like interface.
  - Flink,
  - Spark Structured Streaming, and
  - KSQL-DB

KSQL-DB

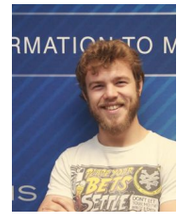
- Available since Kafka 1.9/2 (or confluent platform 5)
- builds directly on top of KStreams Library
- **Relevant Concepts:** simplicity is the key, record vs changelog stream, KTable and KStreams on top of (compacted) commit logs

Spark Structured Streaming

- Available since Spark 2.0, it extends Dataframe and Datasets to Streaming Datasets

- SQL-like programming interface that relies on Catalyst for optimization

- **Relevant Concepts**: StreamingDataFrame, registered queries, windows as group-by function, sink in ”Complete”, “Append”, or “Update “ mode



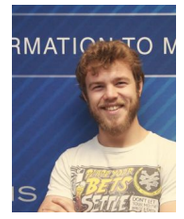
Flink SQL

- Available since version 1.3, it builds on Flink Table API (LINQ-style API)

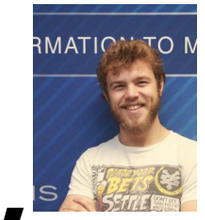
- Uses Apache Calcite of parsing, interpreting and planning, while execution relies on FLINK Runtime.

- **Relevant concepts**: windows as group-by function, temporal tables, triggering, and match-recognize (not covered today), queryable state

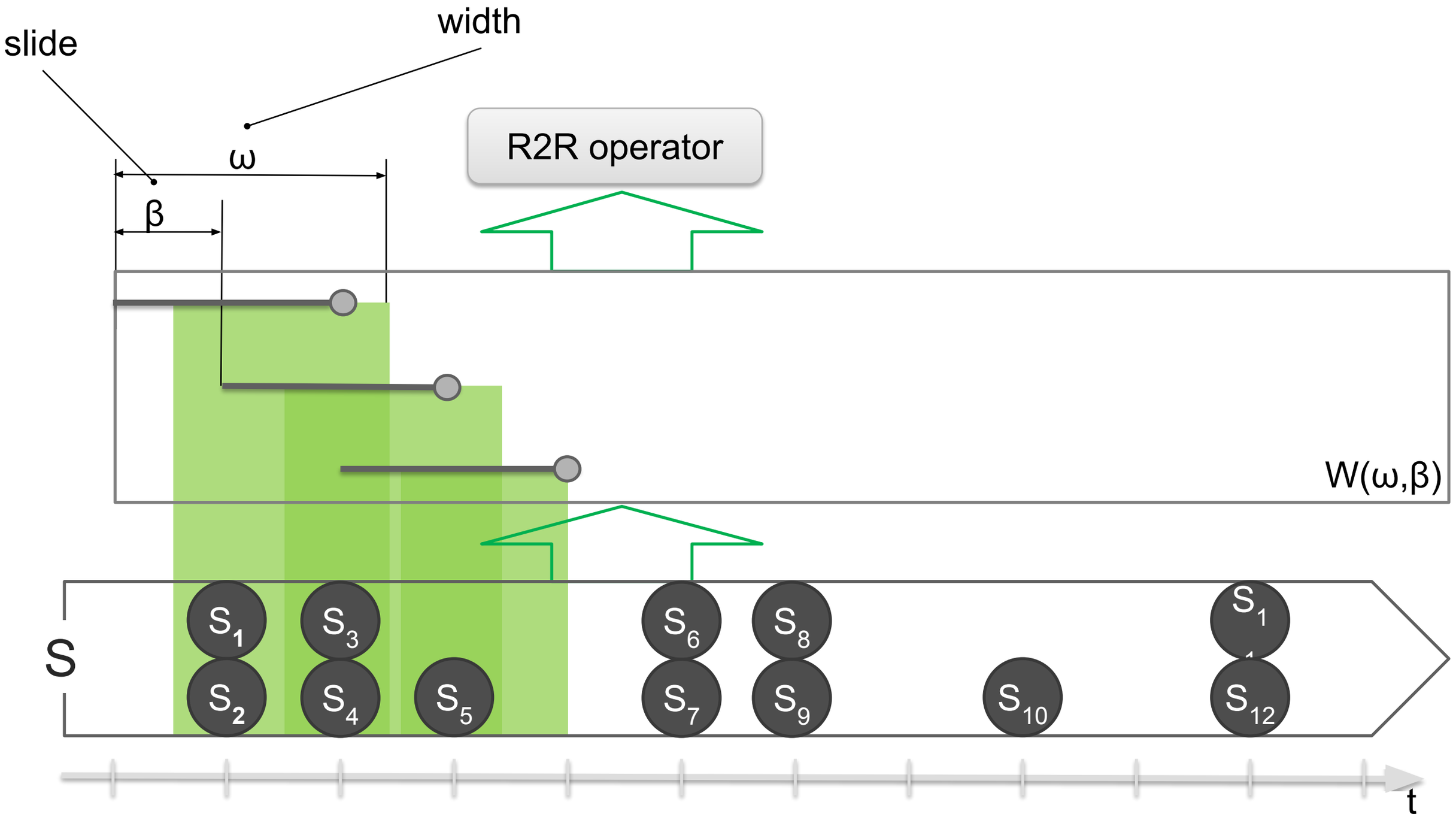
Time-Window Operators & Aggregates



- Sliding Window
- Tumbling Window
- Session Window
- Aggregations: COUNT, SUM, AVG, MEAN, MAX, MIX



Sliding/Hopping Window



KSQL Hopping Window

DDL Extension

Aggregate

Window From Function

```
CREATE TABLE analysis AS
SELECT nation, COUNT(*)
FROM pageviews
WINDOW HOPPING (SIZE 30 SECONDS, ADVANCE BY 10
SECONDS)
GROUP BY nation;
```

Results

SELECT * **FROM** analysis

1561375069212 | Page_66 : Window{start=1561375050000 end=-} | Page_66 | 1
1561375069311 | Page_11 : Window{start=1561375050000 end=-} | Page_11 | 1
1561375073332 | Page_33 : Window{start=1561375050000 end=-} | Page_33 | 1
1561375077242 | Page_32 : Window{start=1561375050000 end=-} | Page_32 | 1
1561375080706 | Page_55 : Window{start=1561375080000 end=-} | Page_55 | 1
1561375082825 | Page_34 : Window{start=1561375080000 end=-} | Page_34 | 1
1561375085084 | Page_56 : Window{start=1561375080000 end=-} | Page_56 | 1
1561375086275 | Page_85 : Window{start=1561375080000 end=-} | Page_85 | 1
1561375086905 | Page_20 : Window{start=1561375080000 end=-} | Page_20 | 1
1561375094475 | Page_27 : Window{start=1561375080000 end=-} | Page_27 | 1

Results

```
SELECT TIMESTAMPTOSTRING(ROWTIME, 'yyyy-MM-dd HH:mm:ss'),
TIMESTAMPTOSTRING(TS, 'yyyy-MM-dd HH:mm:ss'), CLUB_STATUS,
CHANNEL, CNT, TOTS
FROM RATINGS_1M
WHERE CLUB_STATUS='platinum' AND CHANNEL='iOS';
```

```
2020-03-14 06:53:43 | 2020-03-14 06:53:00 | platinum | iOS | 13 | 35
2020-03-14 06:54:56 | 2020-03-14 06:54:00 | platinum | iOS | 10 | 28
2020-03-14 06:55:58 | 2020-03-14 06:55:00 | platinum | iOS | 13 | 31
2020-03-14 06:56:57 | 2020-03-14 06:56:00 | platinum | iOS | 12 | 29
2020-03-14 06:57:55 | 2020-03-14 06:57:00 | platinum | iOS | 6 | 11
2020-03-14 06:58:28 | 2020-03-14 06:58:00 | platinum | iOS | 5 | 12
2020-03-14 06:58:40 | 2020-03-14 06:58:00 | platinum | iOS | 6 | 13
2020-03-14 06:58:55 | 2020-03-14 06:58:00 | platinum | iOS | 7 | 15
2020-03-14 06:58:57 | 2020-03-14 06:58:00 | platinum | iOS | 8 | 16
2020-03-14 06:59:08 | 2020-03-14 06:59:00 | platinum | iOS | 1 | 1
2020-03-14 06:59:09 | 2020-03-14 06:59:00 | platinum | iOS | 2 | 4
```

In the past
there is one result per window

In real-time
there is one result per rating

Flink SQL Hopping Window

Aggregate

Window helper functions

Group By Function

SELECT nation, **COUNT**(*) , HOP_START(..) HOP_END(...)
FROM pageviews
GROUP BY **HOP**(rowtime, **INTERVAL** 1H, **INTERVAL** 1M),
nation

Results

Check Code at http://streaminglangs.io/

1> (Egypt,2019-06-24 11:38:00.0,2019-06-24 11:38:01.0,1)
1> (Egypt,2019-06-24 11:39:00.0,2019-06-24 11:39:01.0,1)
1> (Egypt,2019-06-24 11:40:00.0,2019-06-24 11:40:01.0,1)
1> (Egypt,2019-06-24 11:41:00.0,2019-06-24 11:41:01.0,1)
2> (Italy,2019-06-24 11:42:00.0,2019-06-24 11:42:01.0,1)
2> (Italy,2019-06-24 11:43:00.0,2019-06-24 11:43:01.0,1)
2> (Italy,2019-06-24 11:44:00.0,2019-06-24 11:44:01.0,1)
2> (Italy,2019-06-24 11:45:00.0,2019-06-24 11:45:01.0,1)
2> (Italy,2019-06-24 11:46:00.0,2019-06-24 11:46:01.0,1)
2> (Italy,2019-06-24 11:47:00.0,2019-06-24 11:47:01.0,1)
2> (Italy,2019-06-24 11:48:00.0,2019-06-24 11:48:01.0,1)
3> (Estonia,2019-06-24 11:49:00.0,2019-06-24 11:49:01.0,1)

....

Spark Structured Streaming Hopping Window

Window operator

Aggregate

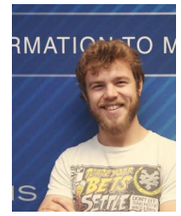
```
val df = pageviews
.groupBy(
window($"timestamp", "1 hour", "1 minute”), $"nation"), count()
```

Spark Structured Streaming Hopping Window

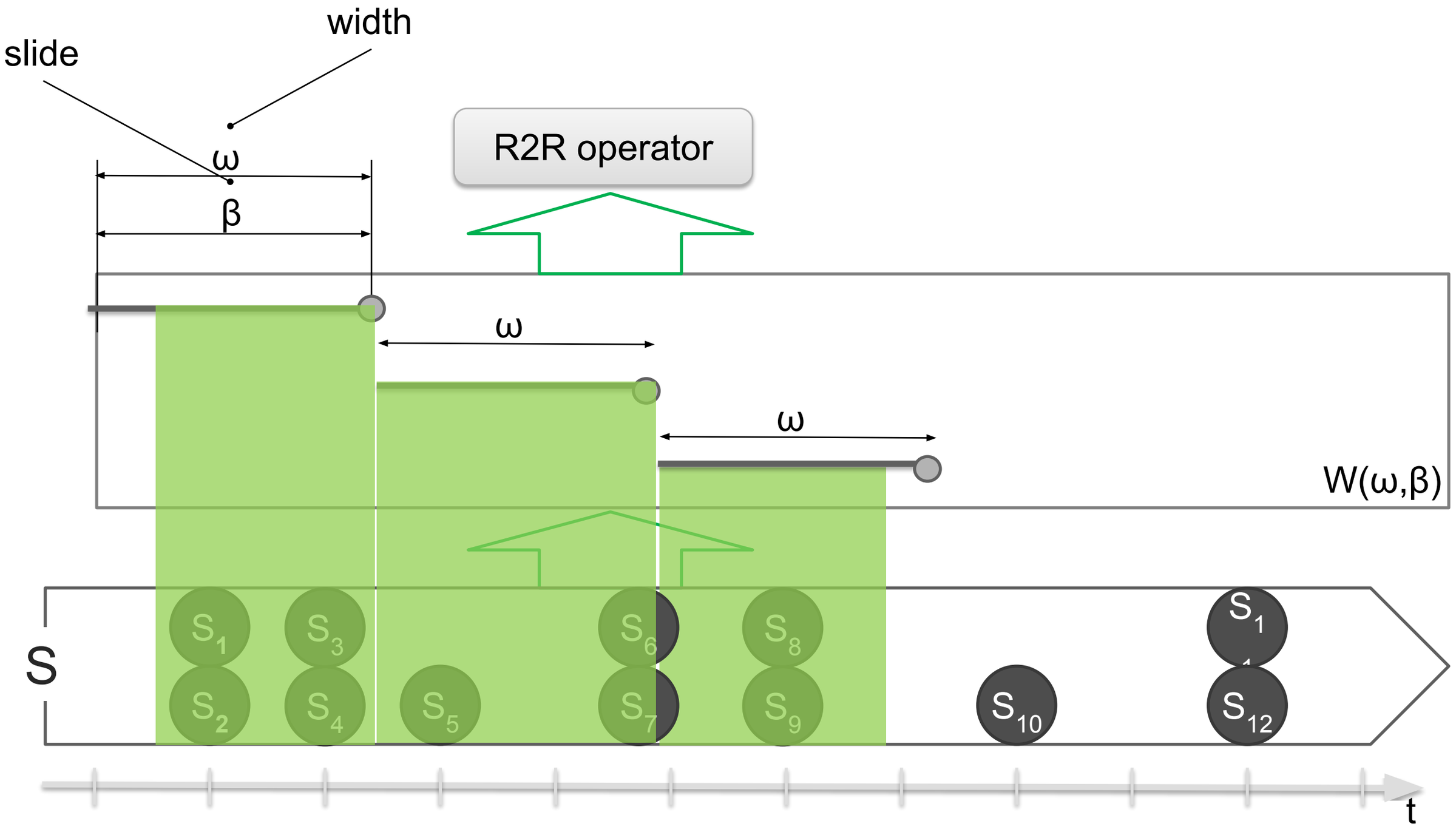
Window operator

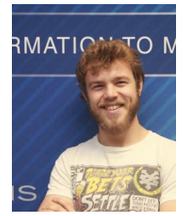
```
1 val df = pageviews.groupBy(
2     window($"timestamp", "1 hour", "1 minute"), $"nation")
3 .count()
4
```

Aggregate

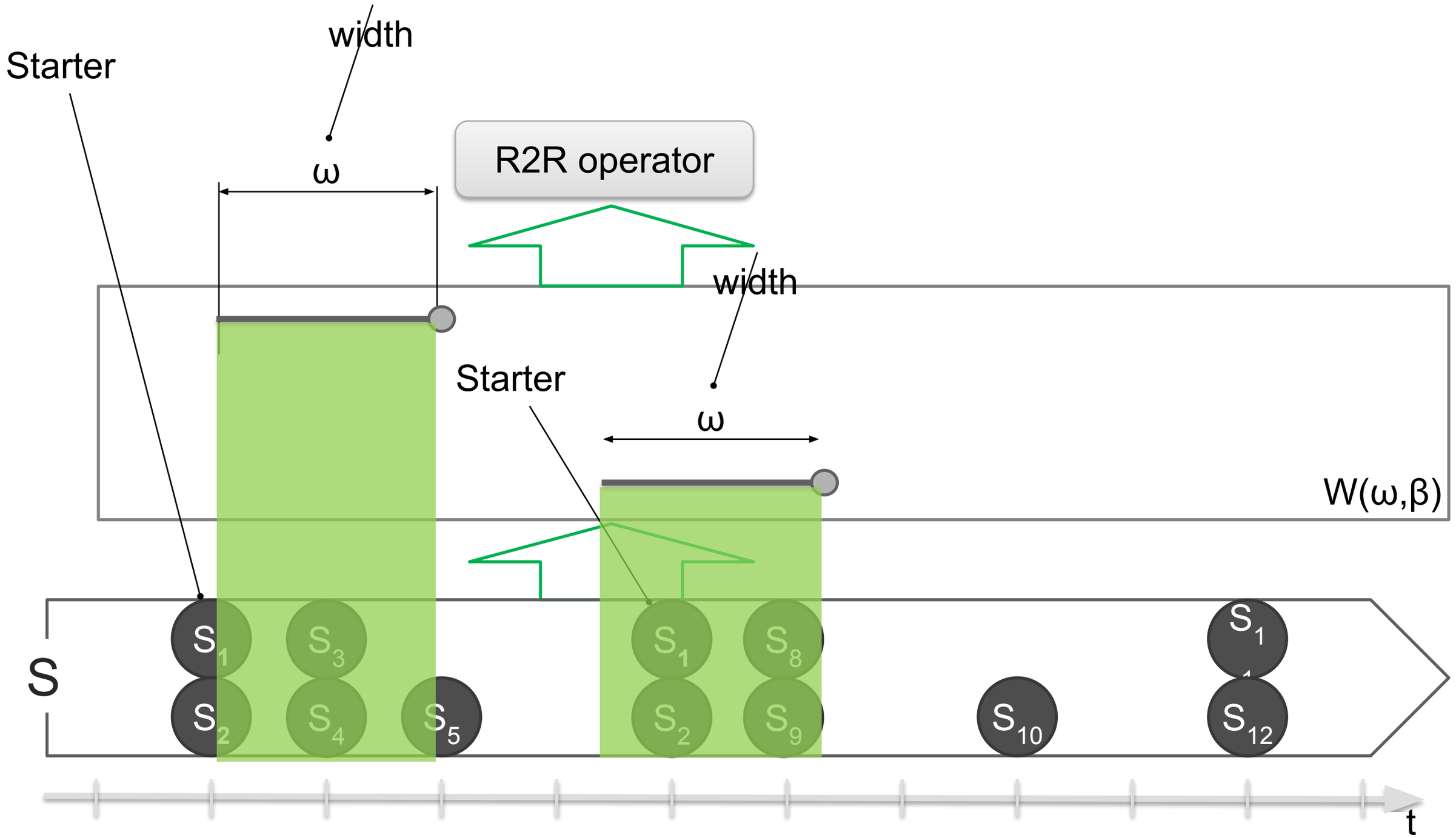


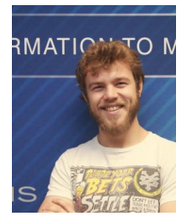
Tumbling Window





Session Window





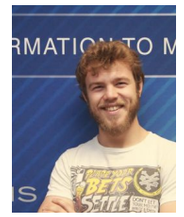
KSQL Session

DDL Extension

Aggregate

```
CREATE TABLE analysis AS
SELECT nation, COUNT (*),
TIMESTAMPTOSTRING(windowstart(), 'yyyy-MM-dd
HH:mm:ss') AS window_start_ts,
TIMESTAMPTOSTRING(windowend(),   'yyyy-MM-dd
HH:mm:ss') AS window_end_ts
FROM pageviews WINDOW SESSION (1 MINUTE)
GROUP BY nation;
```

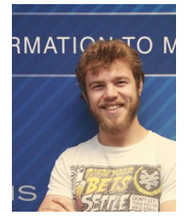
Window From Function



Results

Check Code at http://streaminglangs.io/

Page_82 | 2019-06-24 11:47:45 | 2019-06-24 11:47:45 | 1
Page_73 | 2019-06-24 11:47:46 | 2019-06-24 11:47:46 | 1
Page_16 | 2019-06-24 11:47:49 | 2019-06-24 11:47:49 | 1
Page_54 | 2019-06-24 11:47:25 | 2019-06-24 11:47:53 | 2
Page_68 | 2019-06-24 11:47:55 | 2019-06-24 11:47:55 | 1
Page_25 | 2019-06-24 11:47:40 | 2019-06-24 11:47:58 | 2
Page_17 | 2019-06-24 11:47:59 | 2019-06-24 11:47:59 | 1
Page_92 | 2019-06-24 11:48:02 | 2019-06-24 11:48:02 | 1
Page_83 | 2019-06-24 11:48:05 | 2019-06-24 11:48:05 | 1
Page_37 | 2019-06-24 11:48:06 | 2019-06-24 11:48:06 | 1
Page_86 | 2019-06-24 11:48:07 | 2019-06-24 11:48:07 | 1



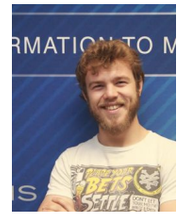
Flink SQL Session

Aggregate

Custom Window Helper Functions

Group By Function

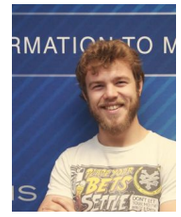
```
SELECT nation, count(*),
SESSION_START(...), SESSION_ROWTIME(...)
FROM pageviews
GROUP BY SESSION(rowtime, INTERVAL 1M), nation
```



Results

Check Code at http://streaminglangs.io/

3> (Estonia,1,2019-06-24 11:52:55.538,2019-06-24 11:52:56.538,2019-06-24 11:52:56.537)
2> (Italy,1,2019-06-24 11:52:56.132,2019-06-24 11:52:57.132,2019-06-24 11:52:57.131)
1> (Egypt,1,2019-06-24 11:52:56.633,2019-06-24 11:52:57.633,2019-06-24 11:52:57.632)
3> (Estonia,1,2019-06-24 11:52:57.136,2019-06-24 11:52:58.136,2019-06-24 11:52:58.135)
2> (Italy,1,2019-06-24 11:52:57.64,2019-06-24 11:52:58.64,2019-06-24 11:52:58.639)
1> (Egypt,1,2019-06-24 11:52:58.141,2019-06-24 11:52:59.141,2019-06-24 11:52:59.14)
3> (Estonia,1,2019-06-24 11:52:58.643,2019-06-24 11:52:59.643,2019-06-24 11:52:59.642)
2> (Italy,1,2019-06-24 11:52:59.147,2019-06-24 11:53:00.147,2019-06-24 11:53:00.146)
1> (Egypt,1,2019-06-24 11:52:59.648,2019-06-24 11:53:00.648,2019-06-24 11:53:00.647)
3> (Estonia,1,2019-06-24 11:53:00.152,2019-06-24 11:53:01.152,2019-06-24 11:53:01.151)
2> (Italy,1,2019-06-24 11:53:00.653,2019-06-24 11:53:01.653,2019-06-24 11:53:01.652)
1> (Egypt,1,2019-06-24 11:53:01.158,2019-06-24 11:53:02.158,2019-06-24 11:53:02.157)



Recap

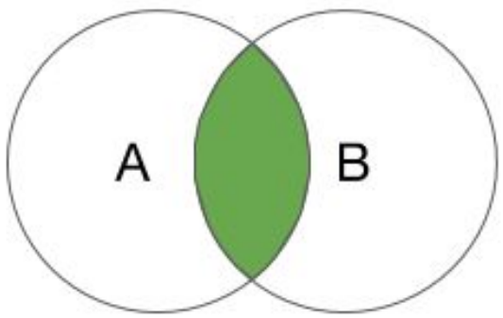
	Landmark	Tumble	Hop	Session	Aggregates
KSQL	implicit	✓	✓	✓	Standard SQL
Flink SQL	implicit	✓	✓	✓	Standard SQL
SSS	implicit	✓	✓	X	Standard SQL

Table 1: Time-Based Window Operators and aggregates across different systems.

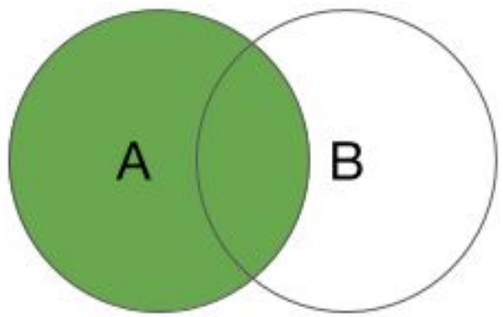
Stream-Table Joins

- Inner Joins
- Left-Outer Join
- Right-Outer Join
- Full-Outer Join

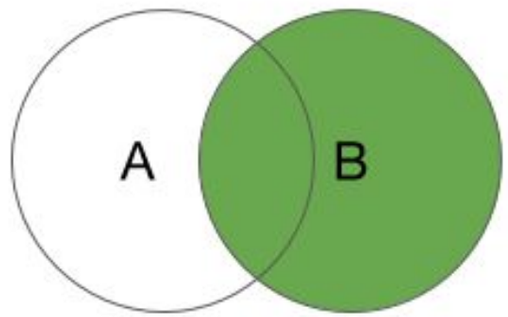
Recap on RA JOINS



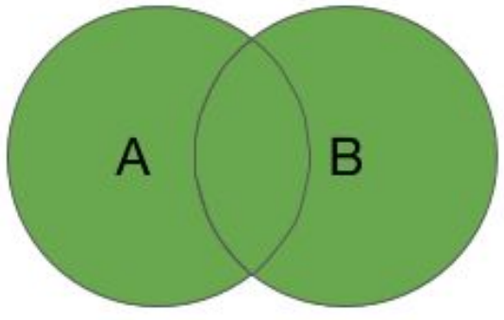
INNER JOIN



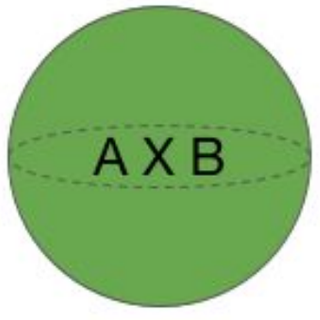
LEFT OUTER JOIN



RIGHT OUTER JOIN

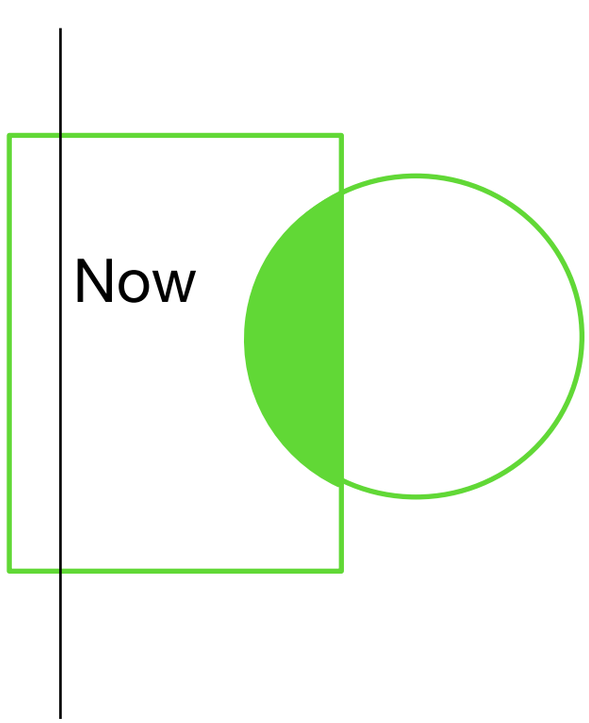


FULL OUTER JOIN

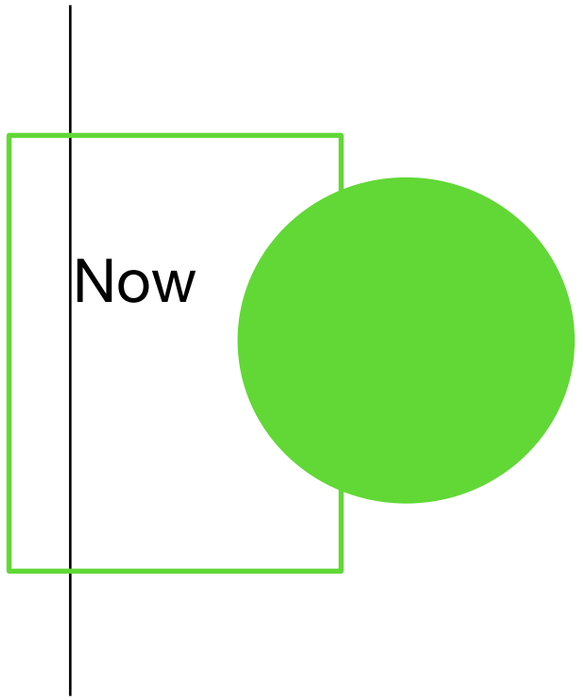


CARTESIAN (CROSS) JOIN

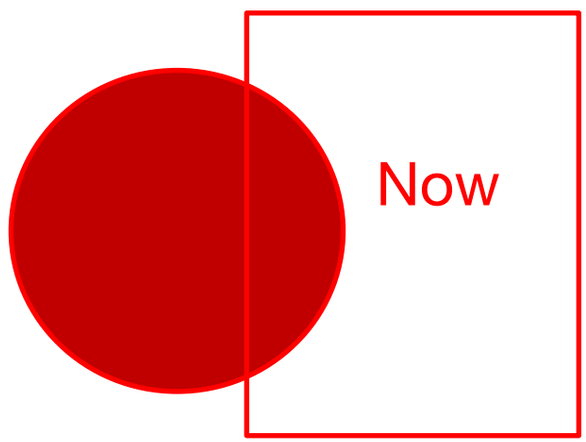
Stream-Table Joins



Inner



Right Outer



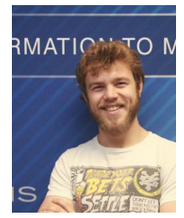
~~Left Outer~~

KSQL Left-Join

DDL Extension

Stream-Table Join

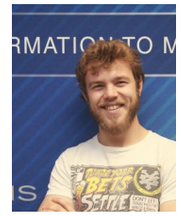
```
CREATE STREAM SENSOR_ENRICHED AS
SELECT S.SENSOR_ID, S.READING_VALUE, I.ITEM_ID
FROM SENSOR_READINGS S
  LEFT JOIN ITEMS_IN_PRODUCTION I ON
S.LINE_ID=I.LINE_ID;
```



Flink SQL LEFT-JOIN

Stream-Table Join

```
SELECT S.SENSOR_ID, S.READING_VALUE, I.ITEM_ID
FROM SENSOR_READINGS S LEFT JOIN
ITEMS_IN_PRODUCTION I ON S.LINE_ID=I.LINE_ID;
```



Results

Check Code at http://streaminglangs.io/

4> (true,0,10.12666825646483,0)
4> (true,0,10.96399203326454,0)
1> (true,2,10.87485672076606 7,2)
4> (true,0,10.268731915130621,0)
1> (true,2,10.786008348182463,2)
4> (true,1,10.360322470661394,1)
4> (true,0,10.809087822653261,0)
4> (true,1,10.238883138171406,1)
1> (true,2,10.776781799073452,2)
4> (true,1,10.528528144000497,1)
4> (true,0,10.532966430120872,0)
4> (true,1,10.449756056124912,1)
4> (true,1,10.66021657541424,1)

Spark Structured Streaming LEFT-JOIN

Table

```
val itemsInProduction = spark.read. ...
```

Stream-Table Join

```
val sensorReadings = spark.readStream. ...

val enrichedSensorReadings =
sensorReadings.join(itemsInProduction, "LINE_ID",
"left-join")
```

Spark Structured Streaming LEFT-JOIN

Table

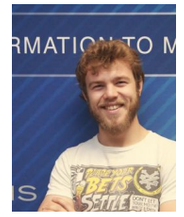
Stream-Table Join

```
1 val itemsInProduction = spark.read.  ...
2 val sensorReadings = spark.readStream.  ...
3 val enrichedSensorReadings = sensorReadings.
4         join(itemsInProduction, "LINE_ID", "left-join")
5
```

Recap

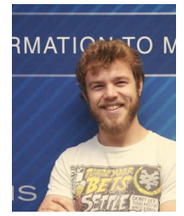
	Inner	Left Outer	Right Outer	Full Outer
KSQL	S	S	NS	NS
Flink SQL	S	S*	S*	S*
SSS	S	S	NS	NS

Table 2: Stream-Static Joins. [S]upported, [N]ot[S]upported. S*, Flink memory usage might grow indefinitely, Temporal Tables can be used to avoid it.

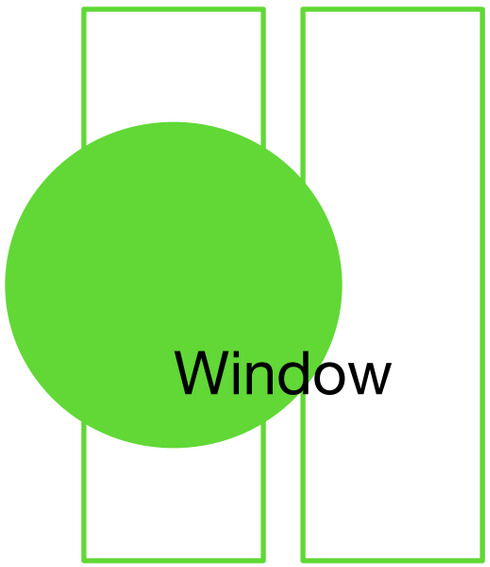


Stream-Stream Joins

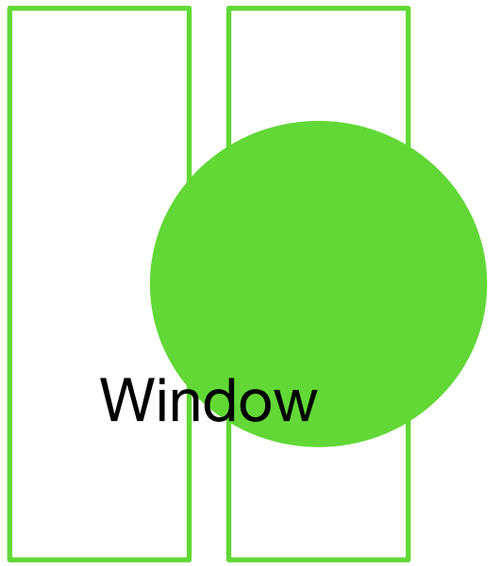
- Inner Joins
- Left-Outer Join
- Right-Outer Join
- Full-Outer Join



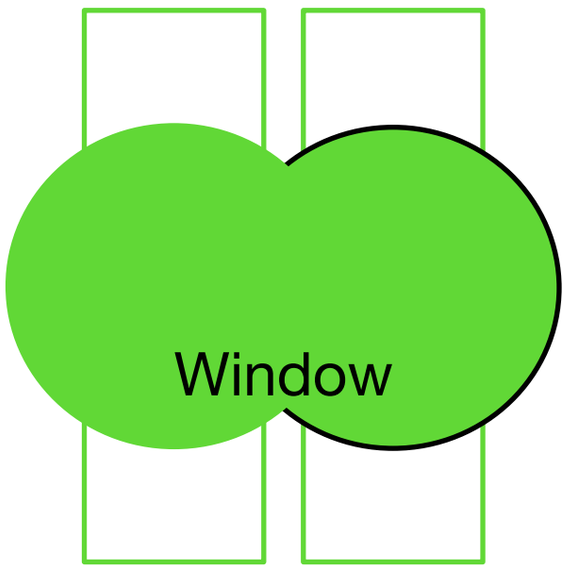
Stream-Stream Joins



Left Outer

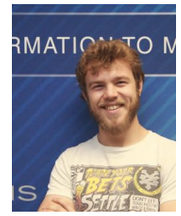


Right Outer



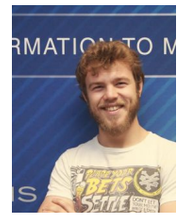
Full Outer

KSQL Inner Join



```
SELECT * FROM IMPRESSIONS, CLICKS
FROM  Impressions I INNER JOIN Clicks C
WITHIN 1 HOUR
WHERE IMPRESSION_ID = CLICK_ID
```

Flink SQL Inner Join



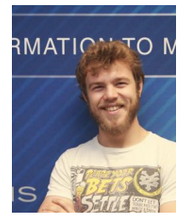
```
SELECT * FROM IMPRESSIONS, CLICKS
WHERE IMPRESSION_ID = CLICK_ID AND
CLICK_TIME BETWEEN IMPRESSION_TIME - INTERVAL
'1' HOUR AND IMPRESSION_TIME
```

Spark Structured Streaming Inner Join

```
val impressions = spark.readStream. ...

val clicks = spark.readStream. ...

// Apply watermarks on event-time columns

val imprWithWtmrk
=impressions.withWatermark("impressionTime", "2 hours")

val clicksWithWatermark = clicks.withWatermark("clickTime", "3
hours") imprWithWtmrk.join( clicksWithWatermark, expr("""
clickAdId = impressionAdId AND clickTime >= impressionTime
AND clickTime <= impressionTime + interval 1 hour"""))
```

Spark Structured Streaming Inner Join

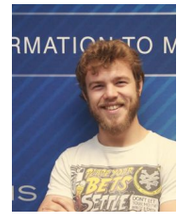
```
val impressions = spark.readStream.  ...
val clicks = spark.readStream.  ...
// Apply watermarks on event-time columns
val imprWithWtmrk =impressions.withWatermark("impressionTime", "2 hours")
val clicksWithWatermark = clicks.withWatermark("clickTime", "3 hours")
imprWithWtmrk.join( clicksWithWatermark, expr(""" clickAdId = impressionAdId AND
clickTime ⩾ impressionTime AND clickTime ⩽ impressionTime + interval 1
hour"""))
```



Recap

	Inner	Left Outer	Right Outer	Full Outer
KSQL	S, Wi	S, Wi	S, Wi	S, Wi
Flink SQL	S* + B	S + B	S + B	S + B
SSS	S	S + w on left +I	S + B on right +I	NS

Table 3: Stream-Stream Joins. [S]upported, [N]ot[S]upported, [B]etween [W]atermark, [Wi]thin, [I]nterval Constraint. S*, Flink memory usage might grow indefinitely.



Agenda

- Introduction on Stream Processing Models [done]
- Declarative Language: Opportunities, and Design Principles [done]
- Comparison of Prominent Streaming SQL Dialects for Big Stream Processing Systems [done]
- Conclusion

Conclusion

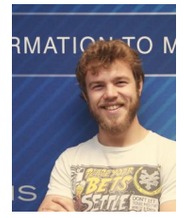
- Continuous SQL extensions are democratizing Stream Processing

- Syntactically :)

- Semantically :/ ← research!

Conclusion

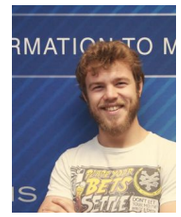
- Possible impact of the research on “Continuous SQL Semantics”
- Correctness of the results
- Portability of the workload across systems
- Continuous data integration



Conclusion

- Static Data Management is “just” a special case of Streaming Data Management

- Are Stream Processing Systems the next generation of Databases?



Test