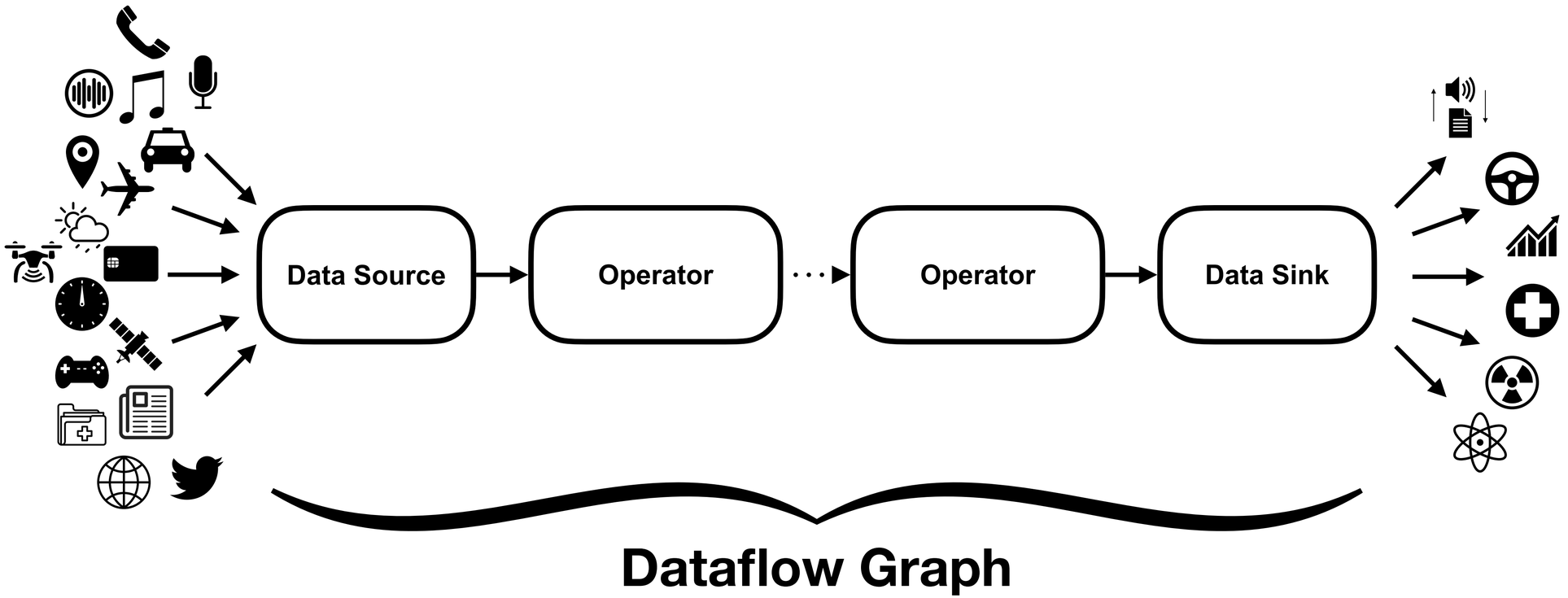
AquaLang: A Streaming Dataflow Programming Language

Klas Segeljakt[1,2] (klasseg@kth)
Seif Haridi[1,2] (haridi@kth.se)
Paris Carbone[1,2] (parisc@kth.se)

[1] **Data Systems Lab** at KTH Royal Institute of Technology
[2] RISE Research Institutes of Sweden

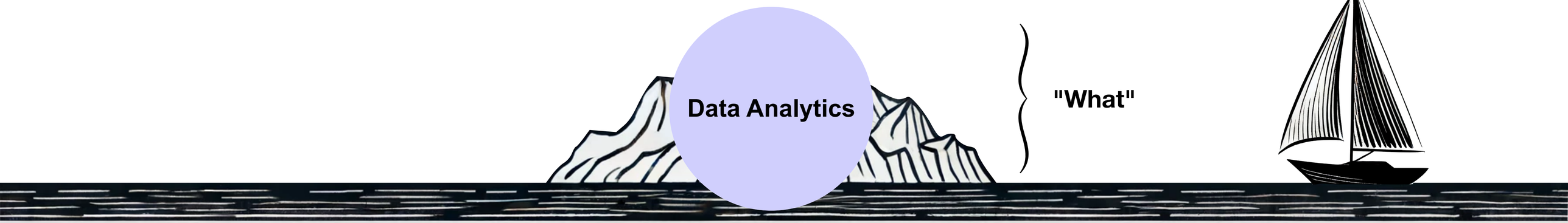
HYTRADBOI 25

What is Dataflow Programming?

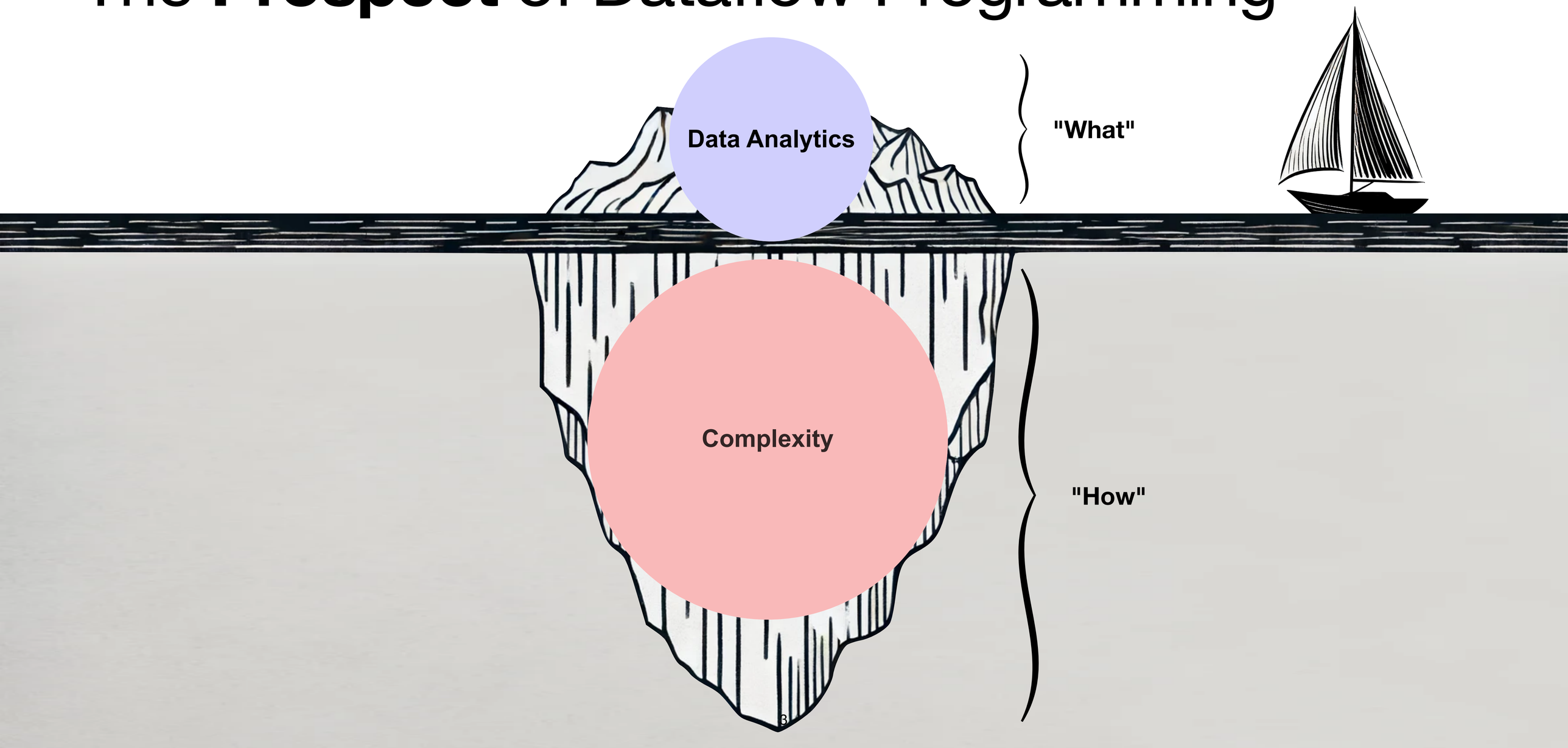


The **Prospect** of Dataflow Programming

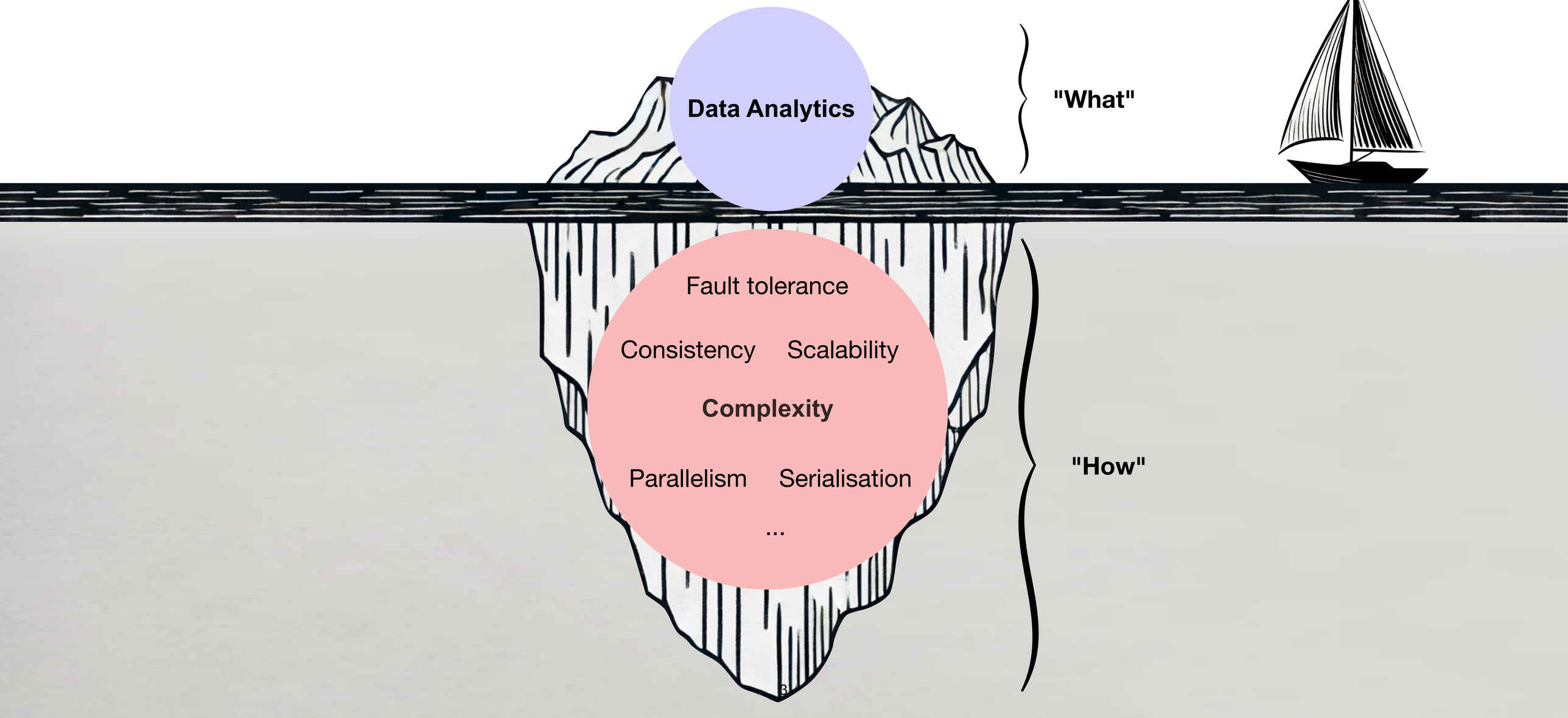
The **Prospect** of Dataflow Programming



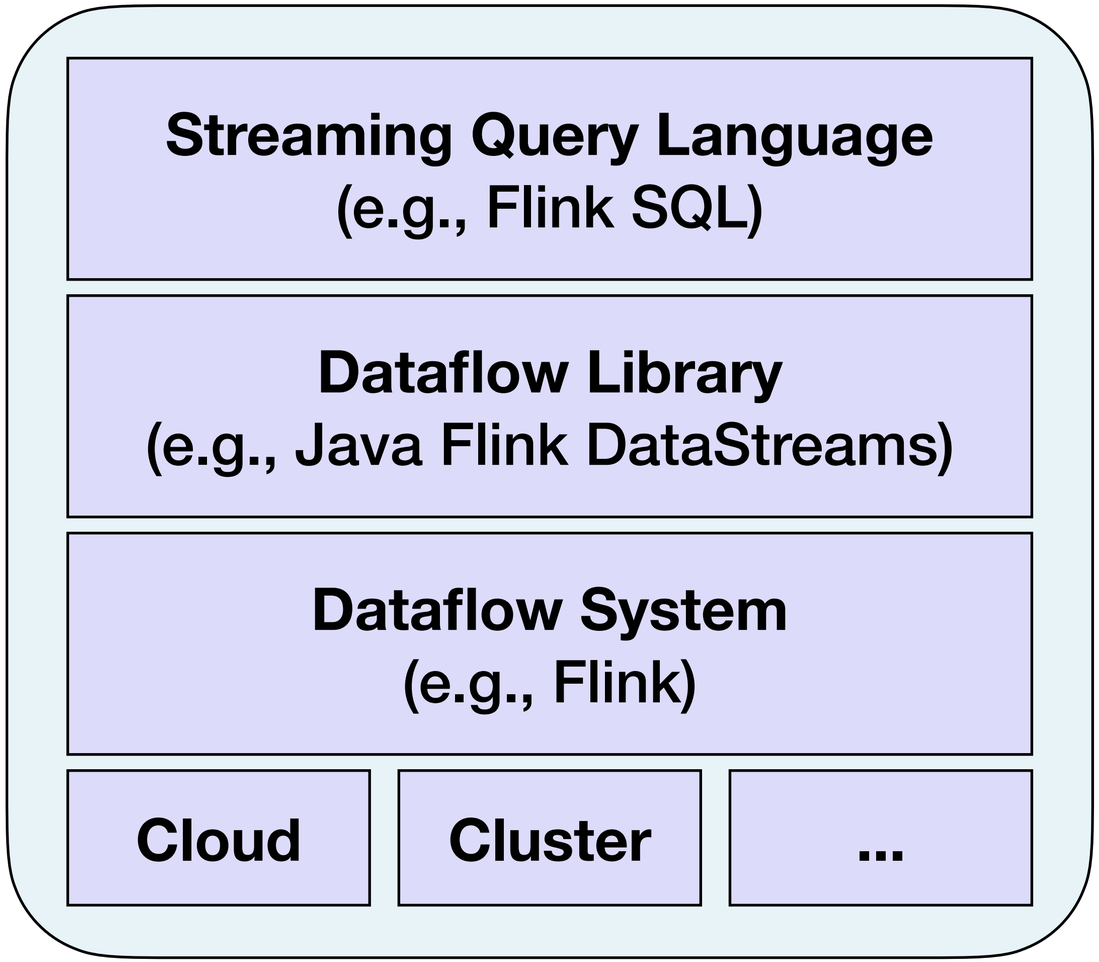
The **Prospect** of Dataflow Programming



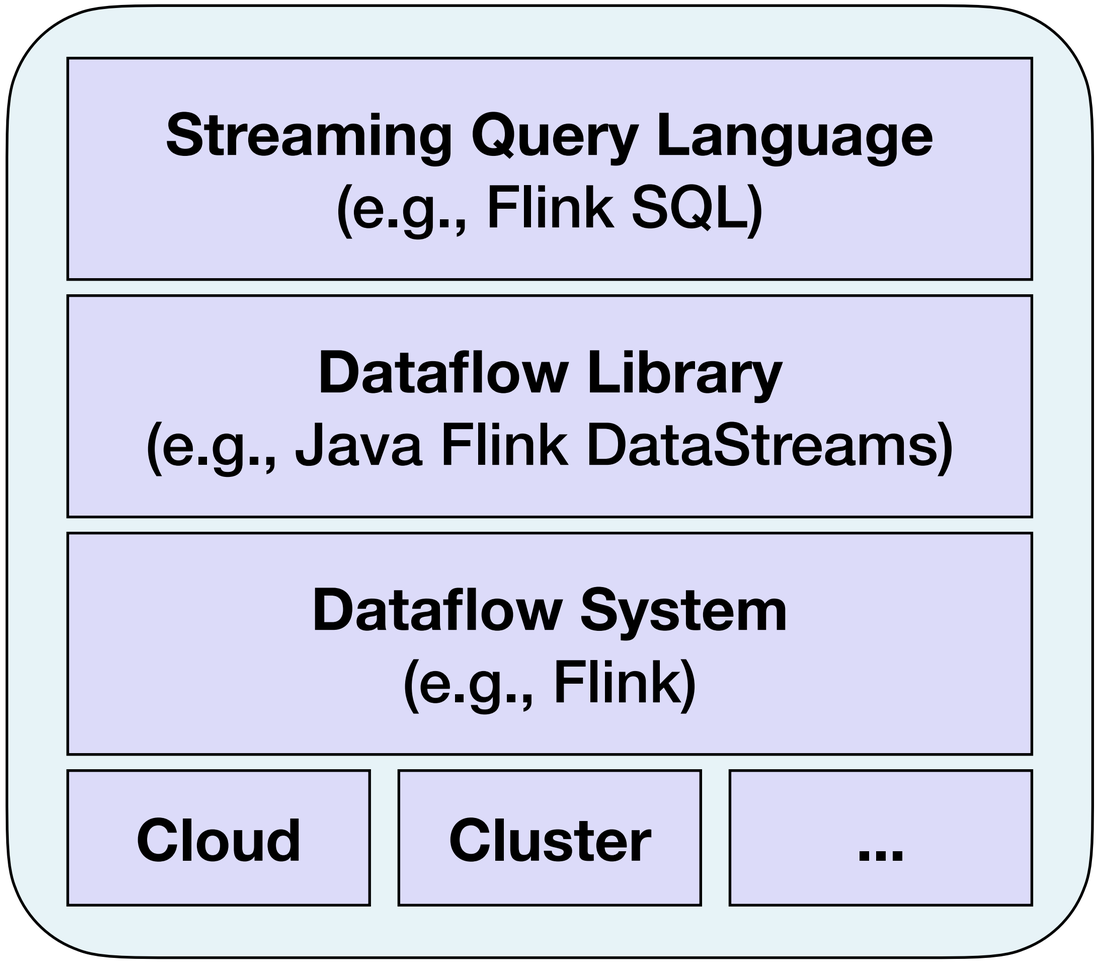
The **Prospect** of Dataflow Programming



What exists today



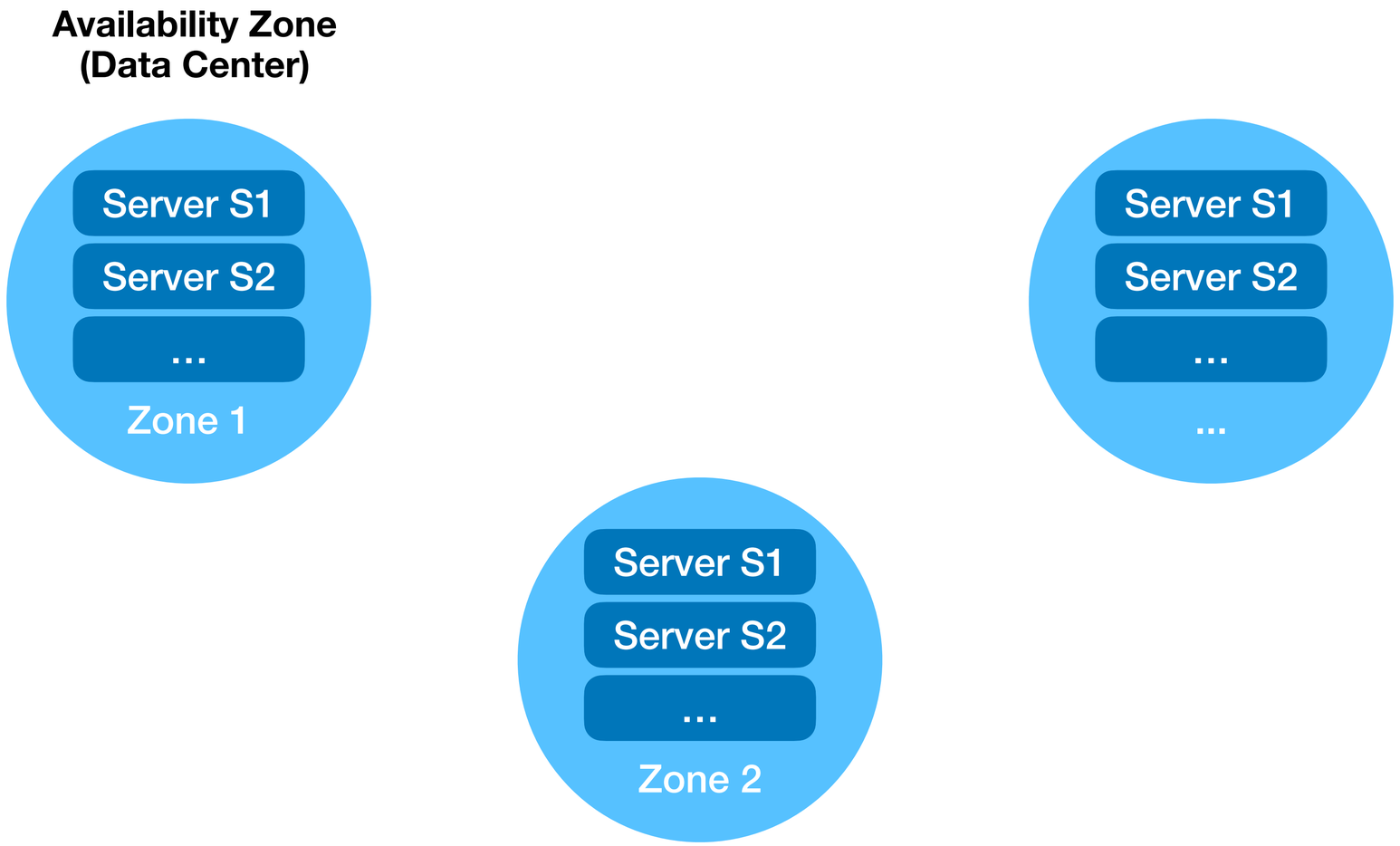
What exists today



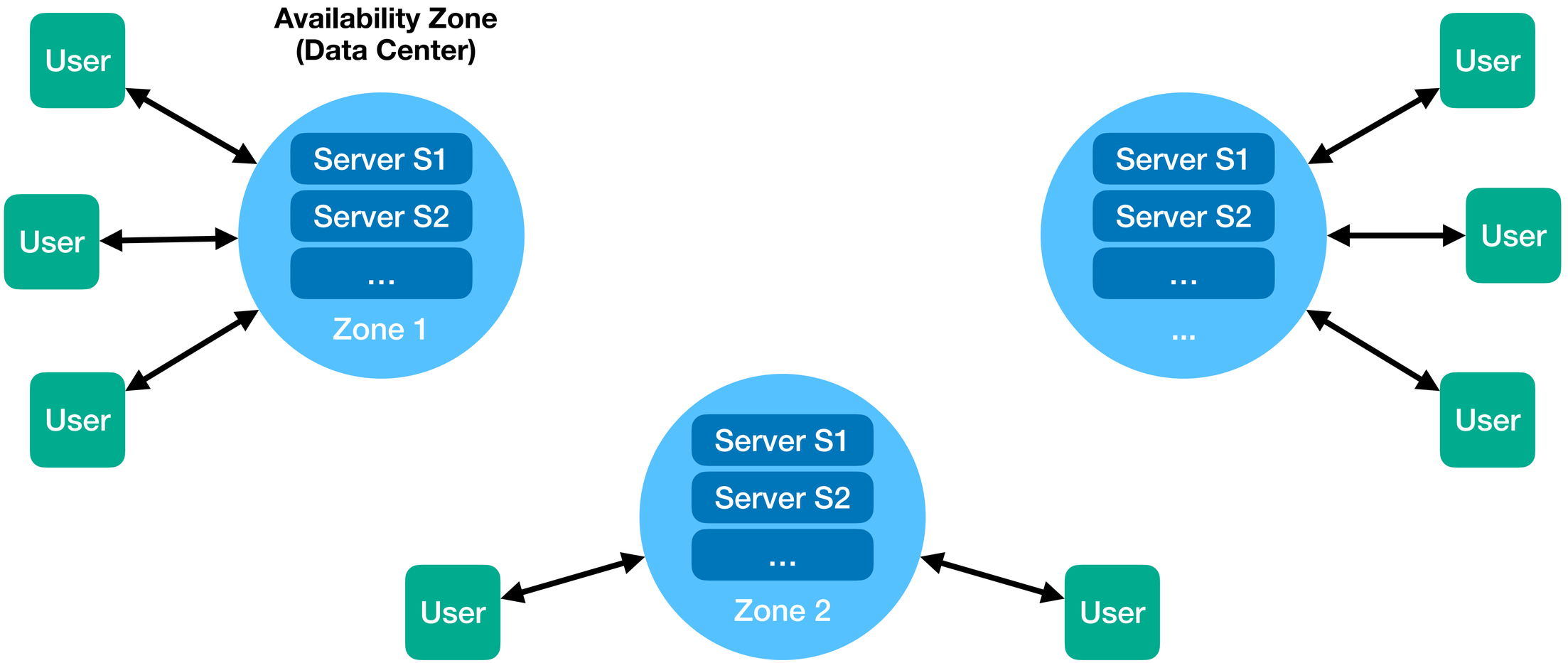
Why AquaLang?

Example Application

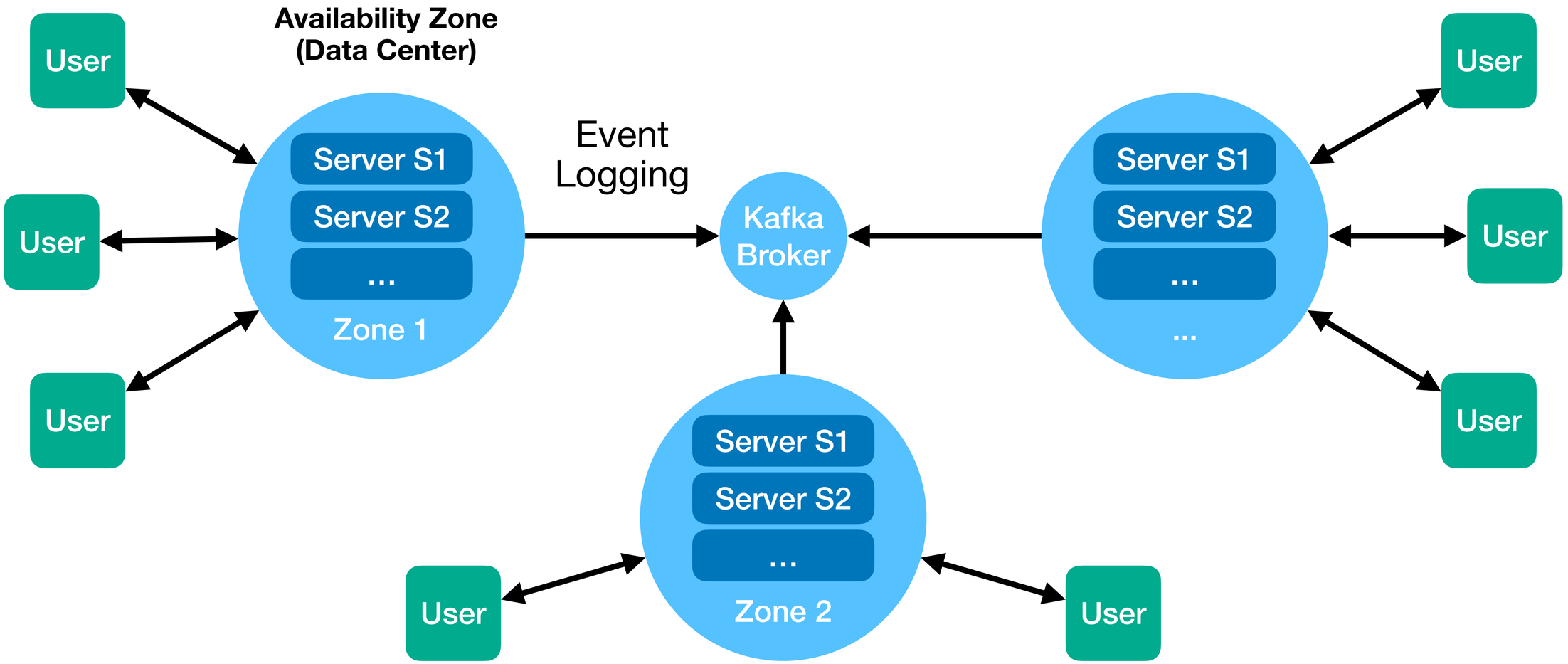
Example Application



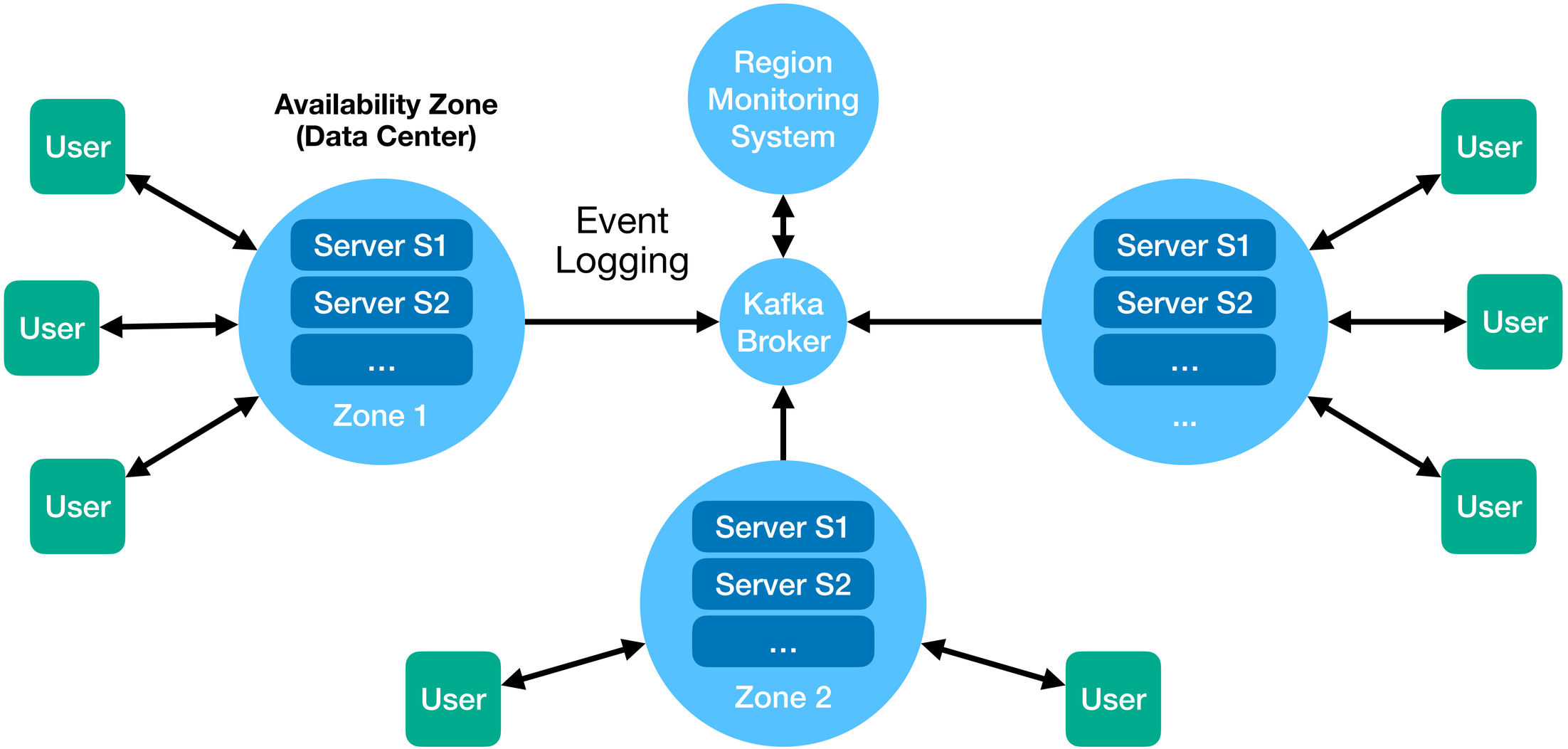
Example Application



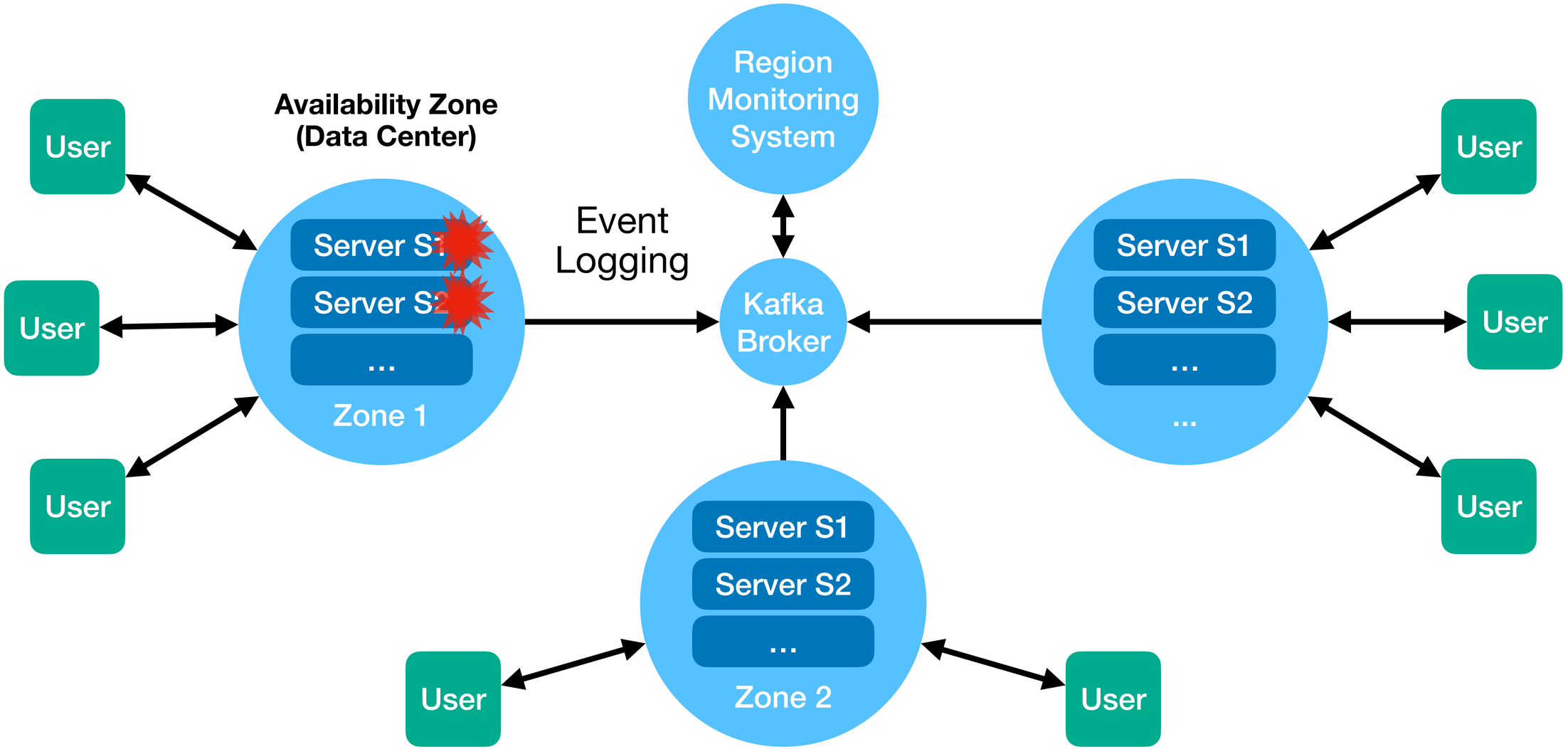
Example Application



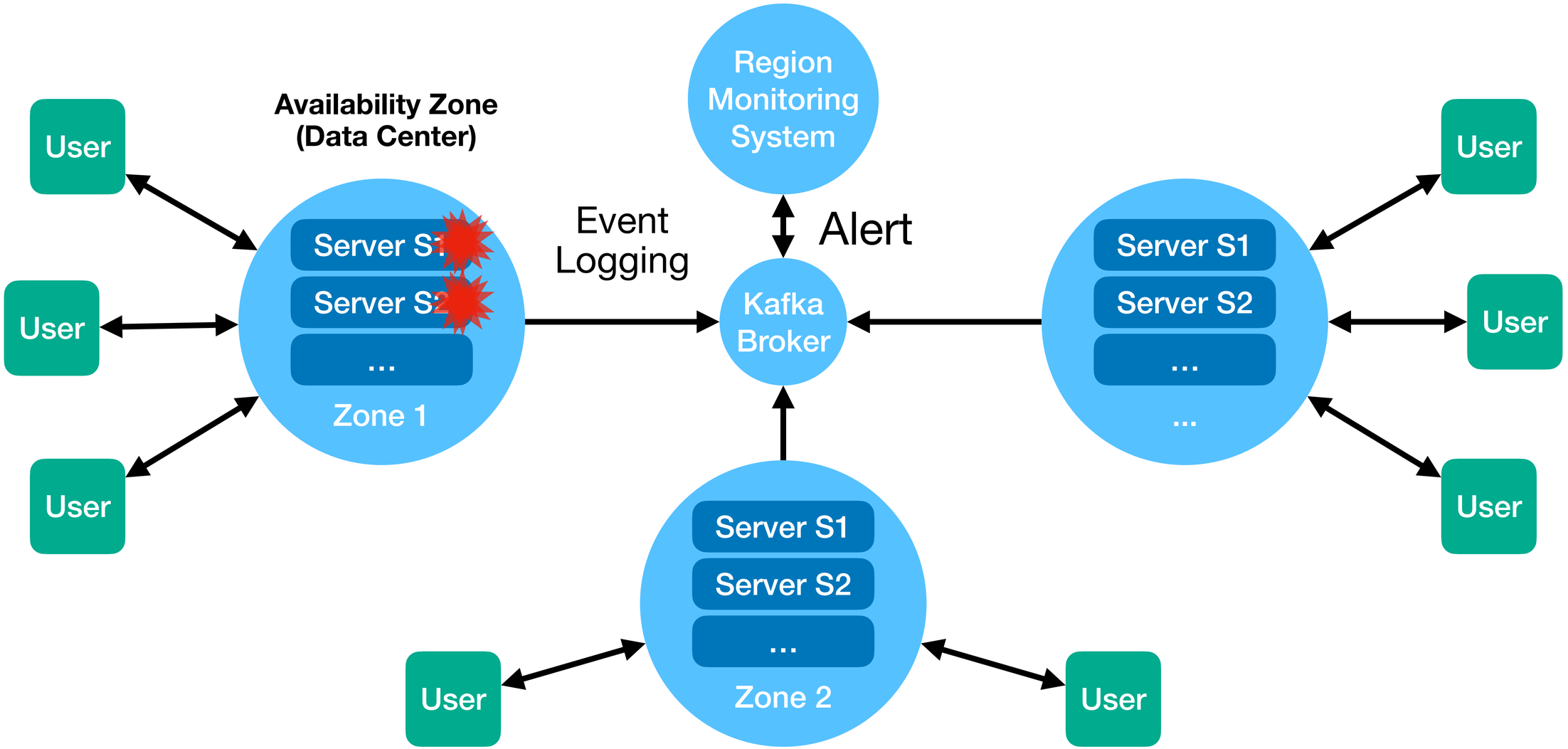
Example Application



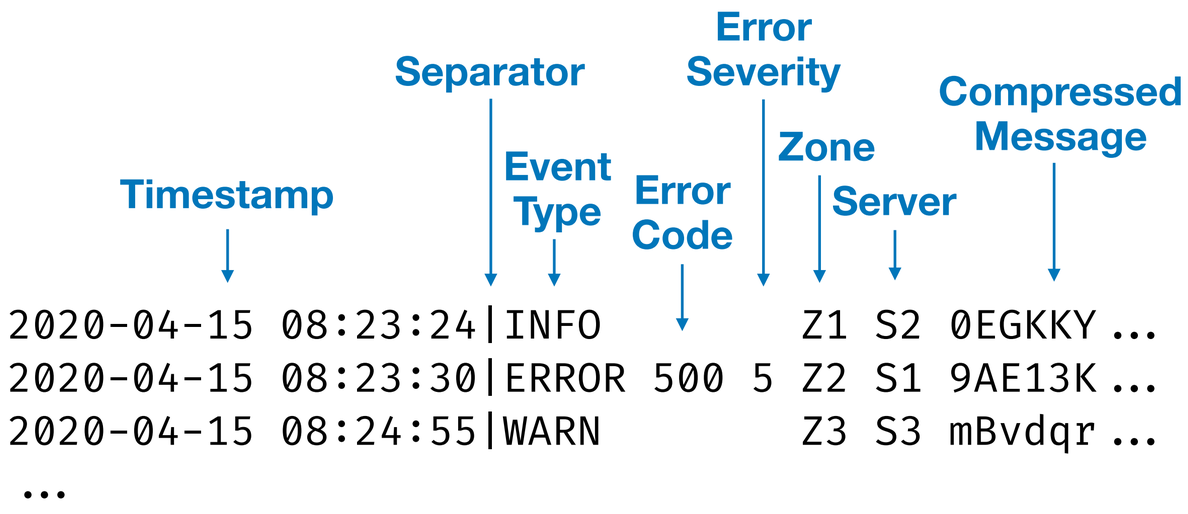
Example Application



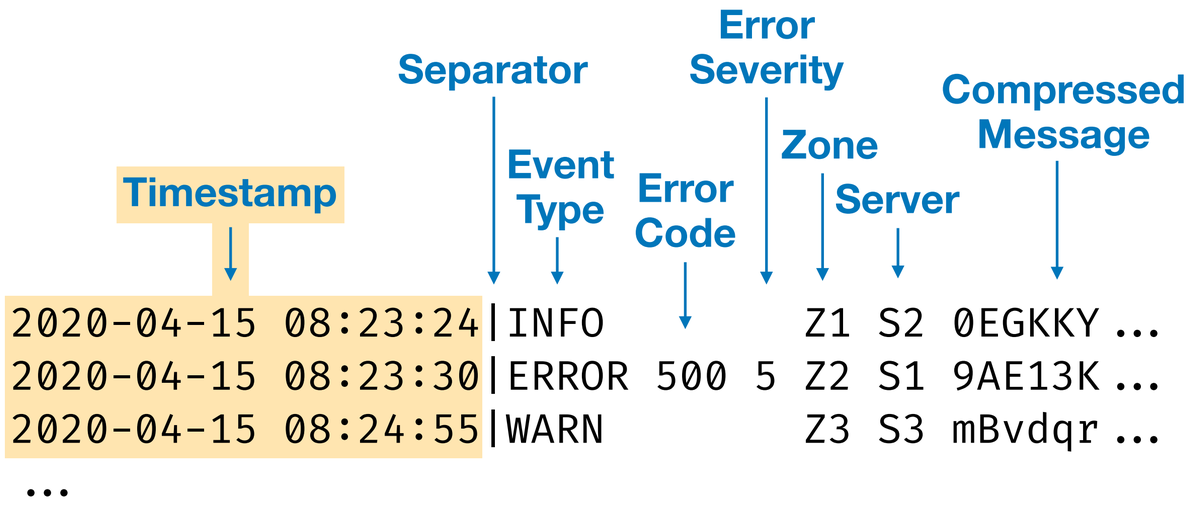
Example Application



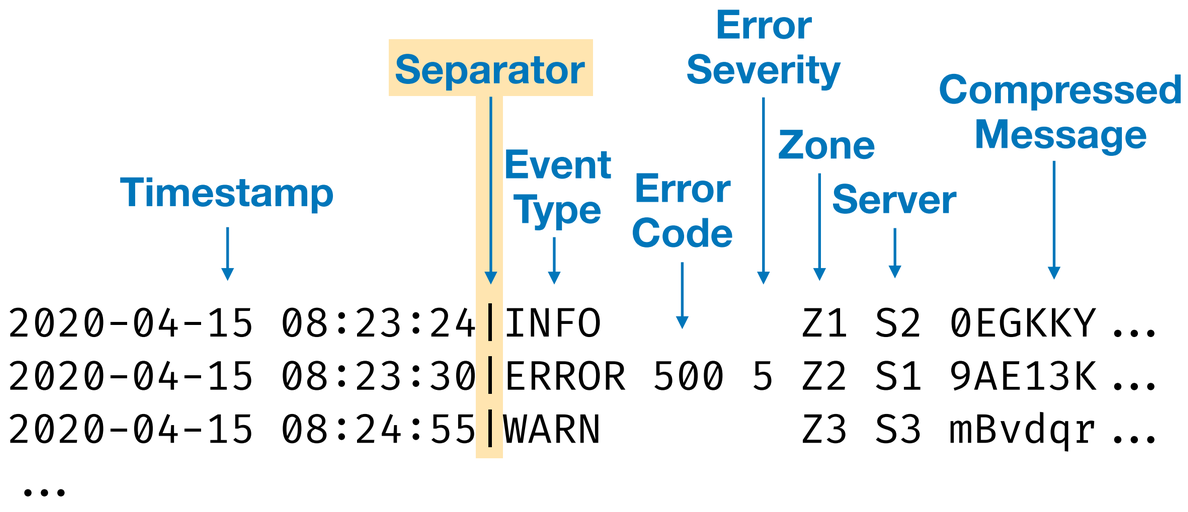
Example Application - Log Events



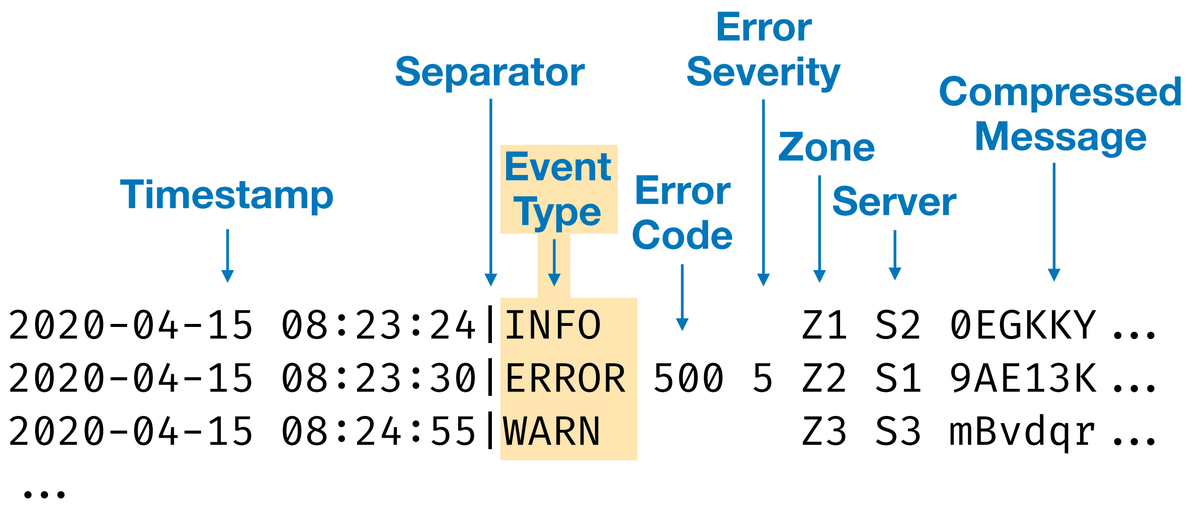
Example Application - Log Events



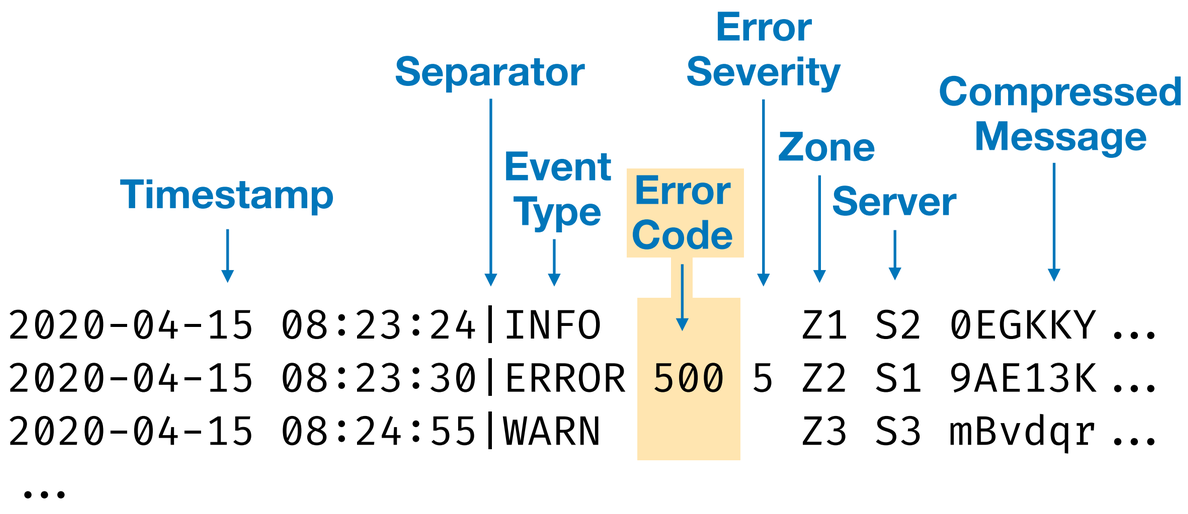
Example Application - Log Events



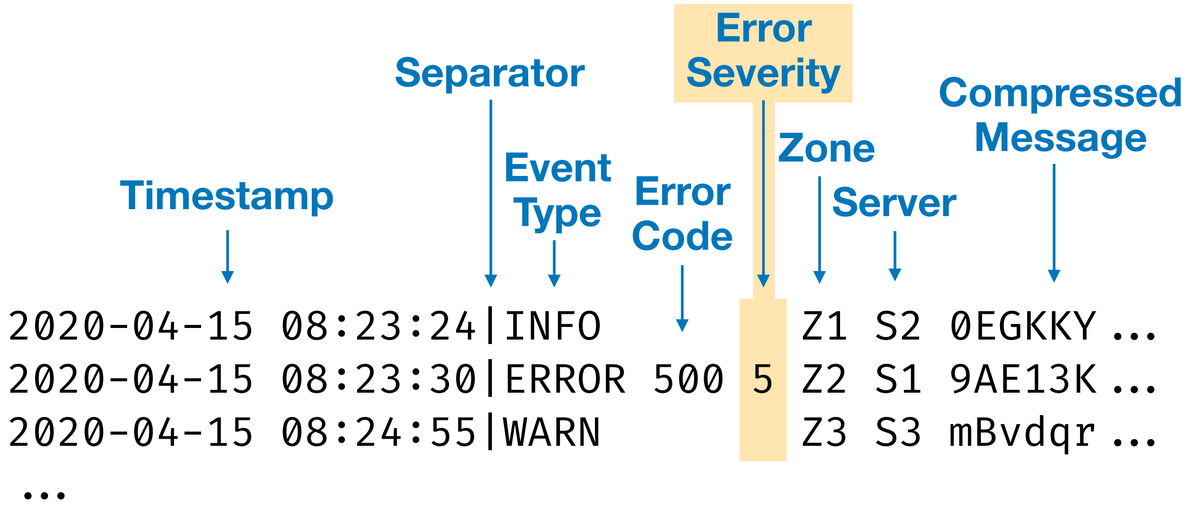
Example Application - Log Events



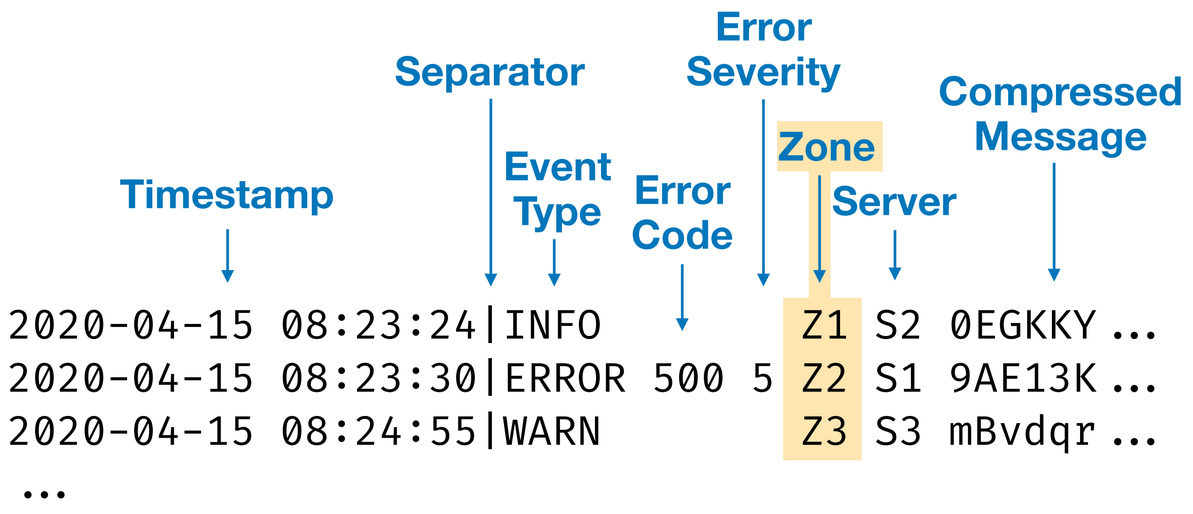
Example Application - Log Events



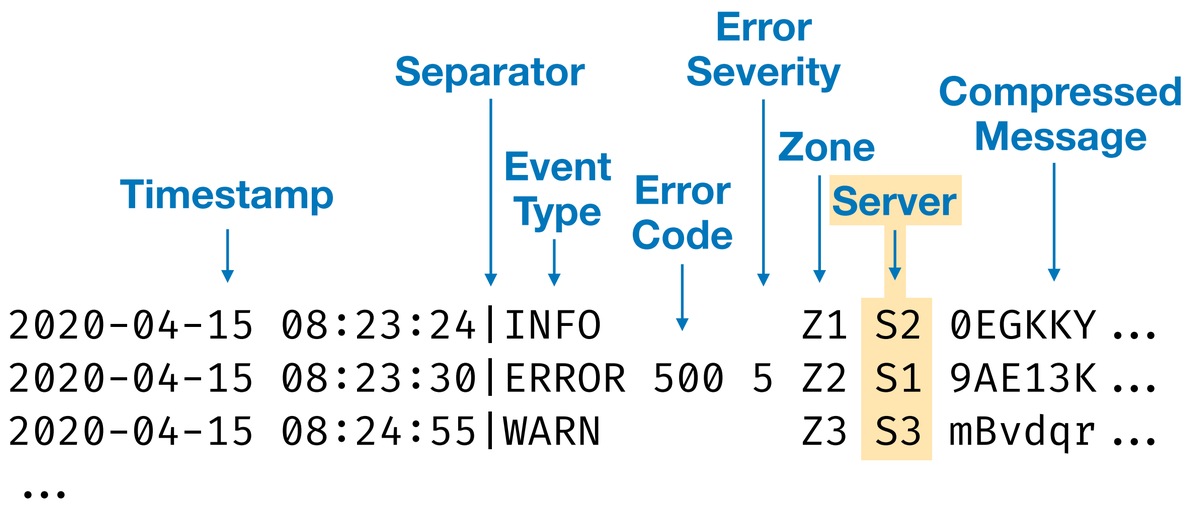
Example Application - Log Events



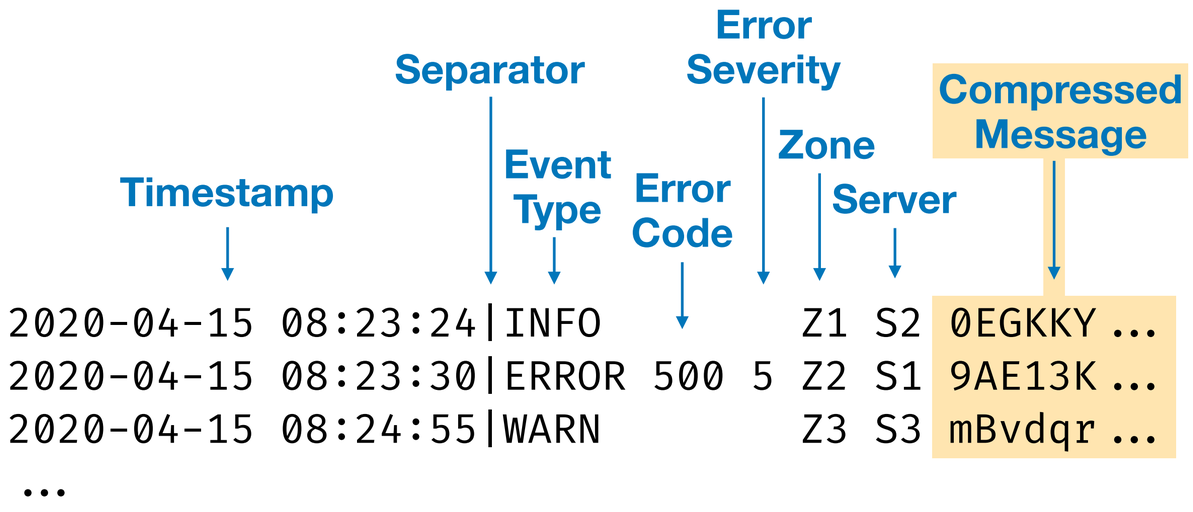
Example Application - Log Events



Example Application - Log Events



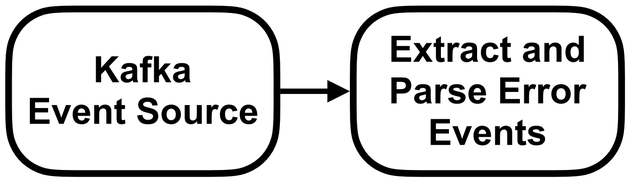
Example Application - Log Events



Example Application - Dataflow

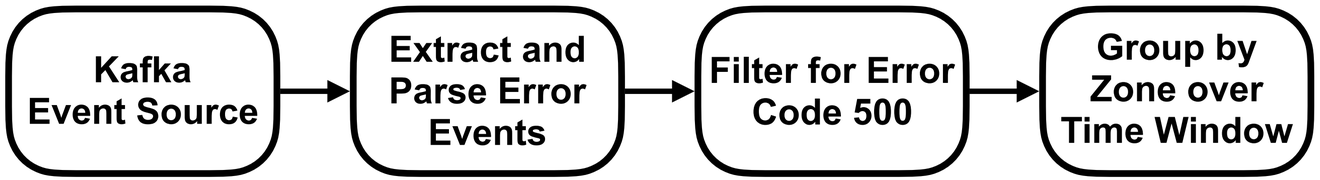
Example Application - Dataflow

Example Application - Dataflow



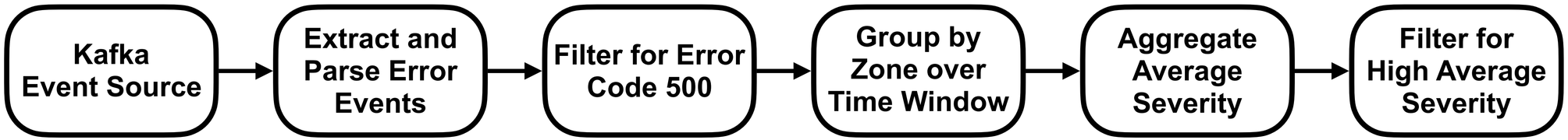
Example Application - Dataflow

Example Application - Dataflow



Example Application - Dataflow

Example Application - Dataflow



Example Application - Dataflow

Example Application - Flink SQL

Example Application - Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
  WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );
```

Example Application - Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
  WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );
```

Example Application - Flink SQL

```
1  > CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
2    WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );
```

Example Application - Flink SQL

```
1  > CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
2    WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );
3
4  > CREATE TABLE Alerts(zone STRING, avg_severity INT, window_start TIMESTAMP(3), window_end TIMESTAMP(3))
5    WITH ('connector'='kafka', 'topic'='alerts', 'format'='json', ... );
```

Example Application - Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
  WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, window_start TIMESTAMP(3), window_end TIMESTAMP(3))
  WITH ('connector'='kafka', 'topic'='alerts', 'format'='json', ... );
```

Example Application - Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
  WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, window_start TIMESTAMP(3), window_end TIMESTAMP(3))
  WITH ('connector'='kafka', 'topic'='alerts', 'format'='json', ... );

> CREATE VIEW ErrorEvents AS
SELECT ts,
       CAST(SPLIT_INDEX(line, ' ', 1) AS INT) AS code,
       CAST(SPLIT_INDEX(line, ' ', 2) AS INT) AS severity,
       SPLIT_INDEX(line, ' ', 3) AS zone,
       SPLIT_INDEX(line, ' ', 4) AS server,
       SPLIT_INDEX(line, ' ', 5) AS message
FROM RawEvents
WHERE SPLIT_INDEX(line, ' ', 0) = 'ERROR';
```

Example Application - Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
  WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, window_start TIMESTAMP(3), window_end TIMESTAMP(3))
  WITH ('connector'='kafka', 'topic'='alerts', 'format'='json', ... );

> CREATE VIEW ErrorEvents AS
SELECT ts,
       CAST(SPLIT_INDEX(line, ' ', 1) AS INT) AS code,
       CAST(SPLIT_INDEX(line, ' ', 2) AS INT) AS severity,
       SPLIT_INDEX(line, ' ', 3) AS zone,
       SPLIT_INDEX(line, ' ', 4) AS server,
       SPLIT_INDEX(line, ' ', 5) AS message
FROM RawEvents
WHERE SPLIT_INDEX(line, ' ', 0) = 'ERROR';
```

Example Application - Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
  WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, window_start TIMESTAMP(3), window_end TIMESTAMP(3))
  WITH ('connector'='kafka', 'topic'='alerts', 'format'='json', ... );

> CREATE VIEW ErrorEvents AS
SELECT ts,
       CAST(SPLIT_INDEX(line, ' ', 1) AS INT) AS code,
       CAST(SPLIT_INDEX(line, ' ', 2) AS INT) AS severity,
       SPLIT_INDEX(line, ' ', 3) AS zone,
       SPLIT_INDEX(line, ' ', 4) AS server,
       SPLIT_INDEX(line, ' ', 5) AS message
FROM RawEvents
WHERE SPLIT_INDEX(line, ' ', 0) = 'ERROR';
```

Example Application - Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
  WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, window_start TIMESTAMP(3), window_end TIMESTAMP(3))
  WITH ('connector'='kafka', 'topic'='alerts', 'format'='json', ... );

> CREATE VIEW ErrorEvents AS
SELECT ts,
       CAST(SPLIT_INDEX(line, ' ', 1) AS INT) AS code,
       CAST(SPLIT_INDEX(line, ' ', 2) AS INT) AS severity,
       SPLIT_INDEX(line, ' ', 3) AS zone,
       SPLIT_INDEX(line, ' ', 4) AS server,
       SPLIT_INDEX(line, ' ', 5) AS message
FROM RawEvents
WHERE SPLIT_INDEX(line, ' ', 0) = 'ERROR';

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end,
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
              INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Example Application - Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
  WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, window_start TIMESTAMP(3), window_end TIMESTAMP(3))
  WITH ('connector'='kafka', 'topic'='alerts', 'format'='json', ... );

> CREATE VIEW ErrorEvents AS
SELECT ts,
       CAST(SPLIT_INDEX(line, ' ', 1) AS INT) AS code,
       CAST(SPLIT_INDEX(line, ' ', 2) AS INT) AS severity,
       SPLIT_INDEX(line, ' ', 3) AS zone,
       SPLIT_INDEX(line, ' ', 4) AS server,
       SPLIT_INDEX(line, ' ', 5) AS message
FROM RawEvents
WHERE SPLIT_INDEX(line, ' ', 0) = 'ERROR';

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end,
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Example Application - Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
  WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, window_start TIMESTAMP(3), window_end TIMESTAMP(3))
  WITH ('connector'='kafka', 'topic'='alerts', 'format'='json', ... );

> CREATE VIEW ErrorEvents AS
SELECT ts,
       CAST(SPLIT_INDEX(line, ' ', 1) AS INT) AS code,
       CAST(SPLIT_INDEX(line, ' ', 2) AS INT) AS severity,
       SPLIT_INDEX(line, ' ', 3) AS zone,
       SPLIT_INDEX(line, ' ', 4) AS server,
       SPLIT_INDEX(line, ' ', 5) AS message
FROM RawEvents
WHERE SPLIT_INDEX(line, ' ', 0) = 'ERROR';

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end,
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Example Application - Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
  WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, window_start TIMESTAMP(3), window_end TIMESTAMP(3))
  WITH ('connector'='kafka', 'topic'='alerts', 'format'='json', ... );

> CREATE VIEW ErrorEvents AS
SELECT ts,
       CAST(SPLIT_INDEX(line, ' ', 1) AS INT) AS code,
       CAST(SPLIT_INDEX(line, ' ', 2) AS INT) AS severity,
       SPLIT_INDEX(line, ' ', 3) AS zone,
       SPLIT_INDEX(line, ' ', 4) AS server,
       SPLIT_INDEX(line, ' ', 5) AS message
FROM RawEvents
WHERE SPLIT_INDEX(line, ' ', 0) = 'ERROR';

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end,
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Example Application - Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
  WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, window_start TIMESTAMP(3), window_end TIMESTAMP(3))
  WITH ('connector'='kafka', 'topic'='alerts', 'format'='json', ... );

> CREATE VIEW ErrorEvents AS
SELECT ts,
       CAST(SPLIT_INDEX(line, ' ', 1) AS INT) AS code,
       CAST(SPLIT_INDEX(line, ' ', 2) AS INT) AS severity,
       SPLIT_INDEX(line, ' ', 3) AS zone,
       SPLIT_INDEX(line, ' ', 4) AS server,
       SPLIT_INDEX(line, ' ', 5) AS message
FROM RawEvents
WHERE SPLIT_INDEX(line, ' ', 0) = 'ERROR';

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end,
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Example Application - Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
  WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, window_start TIMESTAMP(3), window_end TIMESTAMP(3))
  WITH ('connector'='kafka', 'topic'='alerts', 'format'='json', ... );

> CREATE VIEW ErrorEvents AS
SELECT ts,
       CAST(SPLIT_INDEX(line, ' ', 1) AS INT) AS code,
       CAST(SPLIT_INDEX(line, ' ', 2) AS INT) AS severity,
       SPLIT_INDEX(line, ' ', 3) AS zone,
       SPLIT_INDEX(line, ' ', 4) AS server,
       SPLIT_INDEX(line, ' ', 5) AS message
FROM RawEvents
WHERE SPLIT_INDEX(line, ' ', 0) = 'ERROR';

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end,
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Example Application - Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
  WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, window_start TIMESTAMP(3), window_end TIMESTAMP(3))
  WITH ('connector'='kafka', 'topic'='alerts', 'format'='json', ... );

> CREATE VIEW ErrorEvents AS
SELECT ts,
       CAST(SPLIT_INDEX(line, ' ', 1) AS INT) AS code,
       CAST(SPLIT_INDEX(line, ' ', 2) AS INT) AS severity,
       SPLIT_INDEX(line, ' ', 3) AS zone,
       SPLIT_INDEX(line, ' ', 4) AS server,
       SPLIT_INDEX(line, ' ', 5) AS message
FROM RawEvents
WHERE SPLIT_INDEX(line, ' ', 0) = 'ERROR';

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end,
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Example Application - Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, WATERMARK FOR ts AS ts - INTERVAL '5' SECONDS)
  WITH ('connector'='kafka', 'topic'='eventlog', 'format'='csv', 'csv.field-delimiter'='|', ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, window_start TIMESTAMP(3), window_end TIMESTAMP(3))
  WITH ('connector'='kafka', 'topic'='alerts', 'format'='json', ... );

> CREATE VIEW ErrorEvents AS
SELECT ts,
       CAST(SPLIT_INDEX(line, ' ', 1) AS INT) AS code,
       CAST(SPLIT_INDEX(line, ' ', 2) AS INT) AS severity,
       SPLIT_INDEX(line, ' ', 3) AS zone,
       SPLIT_INDEX(line, ' ', 4) AS server,
       SPLIT_INDEX(line, ' ', 5) AS message
FROM RawEvents
WHERE SPLIT_INDEX(line, ' ', 0) = 'ERROR';

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end,
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

How to isolate the parsing code for reuse and testing?

Example Application - UDFs

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, ... )
  WITH ( ... );

```

Flink Java API (Maven project)

```java
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<code INT, severity INT,
                                zone STRING, server STRING,
                                message STRING>"))
public class Parse extends TableFunction<Row> {
    public void eval(String line) {
        String[] parts = line.split(" ");
        String eventType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Example Application - UDFs

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, ... )
  WITH ( ... );

```

Flink Java API (Maven project)

```
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<code INT, severity INT,
                                zone STRING, server STRING,
                                message STRING>"))
public class Parse extends TableFunction<Row> {
    public void eval(String line) {
        String[] parts = line.split(" ");
        String eventType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Example Application - UDFs

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, ... )
  WITH ( ... );

```

Flink Java API (Maven project)

```
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<code INT, severity INT,
                                zone STRING, server STRING,
                                message STRING>"))
public class Parse extends TableFunction<Row> {
    public void eval(String line) {
        String[] parts = line.split(" ");
        String eventType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Example Application - UDFs

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, ... )
  WITH ( ... );

```

Flink Java API (Maven project)

```
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<code INT, severity INT,
                            zone STRING, server STRING,
                            message STRING>"))
public class Parse extends TableFunction<Row> {
    public void eval(String line) {
        String[] parts = line.split(" ");
        String eventType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Example Application - UDFs

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, ... )
  WITH ( ... );

> ADD JAR 'file:///UDFs/target/UDFs-1.0.jar';
> CREATE FUNCTION Parse AS 'com.example.Parse' LANGUAGE JAVA;
```

Flink Java API (Maven project)

```
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<code INT, severity INT,
                                zone STRING, server STRING,
                                message STRING>"))
public class Parse extends TableFunction<Row> {
    public void eval(String line) {
        String[] parts = line.split(" ");
        String eventType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Example Application - UDFs

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, ... )
  WITH ( ... );

> ADD JAR 'file:///UDFs/target/UDFs-1.0.jar';
> CREATE FUNCTION Parse AS 'com.example.Parse' LANGUAGE JAVA;

> CREATE VIEW ErrorEvents AS
SELECT ts, parsed.*
FROM RawEvents, LATERAL TABLE(Parse(line)) AS parsed;

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Flink Java API (Maven project)

```
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<code INT, severity INT,
                                zone STRING, server STRING,
                                message STRING>"))
public class Parse extends TableFunction<Row> {
    public void eval(String line) {
        String[] parts = line.split(" ");
        String eventType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Example Application - Problems?

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, ... )
  WITH ( ... );

> ADD JAR 'file:///UDFs/target/UDFs-1.0.jar';
> CREATE FUNCTION Parse AS 'com.example.Parse' LANGUAGE JAVA;

> CREATE VIEW ErrorEvents AS
SELECT ts, parsed.*
FROM RawEvents, LATERAL TABLE(Parse(line)) AS parsed;

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Flink Java API (Maven project)

```
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<code INT, severity INT,
                                zone STRING, server STRING,
                                message STRING>"))
public class Parse extends TableFunction<Row> {
    public void eval(String line) {
        String[] parts = line.split(" ");
        String eventType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Example Application - Problems?

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(z                    INT, ... )
  WITH ( ... );

> ADD JAR 'file:///UDFs/target/UDFs-1.0.jar';
> CREATE FUNCTION Parse AS 'com.example.Parse' LANGUAGE JAVA;

> CREATE VIEW ErrorEvents AS
SELECT ts, parsed.*
FROM RawEvents, LATERAL TABLE(Parse(line)) AS parsed;

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Problem:
Separate compilation

Flink Java API (Maven project)

```
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<code INT, severity INT,
                                zone STRING, server STRING,
                                message STRING>"))
public class Parse extends TableFunction<Row> {
    public void eval(String line) {
        String[] parts = line.split(" ");
        String eventType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Example Application - Problems?

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, ... )
  WITH ( ... );

> ADD JAR 'file:///UDFs/target/UDFs-1.0.jar';
> CREATE FUNCTION Parse AS 'com.example.Parse' LANGUAGE JAVA;

> CREATE VIEW ErrorEvents AS
SELECT ts, parsed.*
FROM RawEvents, LATERAL TABLE(Parse(line)) AS parsed;

> I
SEL                                S avg_severity,
                                   end
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Problem:
Cannot pushdown predicate into UDF

Flink Java API (Maven project)

```
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<code INT, severity INT,
                                zone STRING, server STRING,
                                message STRING>"))
public class Parse extends TableFunction<Row> {
    public void eval(String line) {
        String[] parts = line.split(" ");
        String eventType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Example Application - Problems?

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, ... )
  WITH ( ... );

> ADD JAR 'file:///UDFs/target/UDFs-1.0.jar';
> CREATE FUNCTION Parse AS 'com.example.Parse' LANGUAGE JAVA;

> CREATE VIEW ErrorEvents AS
SELECT ts, parsed.*
FROM RawEvents, LATERAL TABLE(Parse(line)) AS parsed;

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Flink Java API (Maven project)

```
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<code INT, severity INT,
                                zone STRING, server STRING,
                                message STRING>"))
public class Parse extends TableFunction<Row> {
    public void eval(String line) {
        String[] parts = line.split(" ");
        String eventType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Problem:
Cannot pushdown projection into UDF

Example Application - Problems?

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, ... )
  WITH ( ... );

> ADD JAR 'file:///UDFs/target/UDFs-1.0.jar';
> CREATE FUNCTION Parse AS 'com.example.Parse' LANGUAGE JAVA;

> CREATE VIEW ErrorEvents AS
SELECT ts, parsed.*
FROM RawEvents, LATERAL TABLE(Parse(line)) AS parsed;

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Flink Java API (Maven project)

```
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<severity INT, code INT,
                                zone STRING, server STRING,
                                message STRING>"))
public class Parse extends TableFunction<Row> {
    al(String line) {
        arts = line.split(" ");
        ntType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Problem:
Impedance mismatch

Example Application - Problems?

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, ... )
  WITH ( ... );

> ADD JAR 'file:///UDFs/target/UDFs-1.0.jar';
> CREATE FUNCTION Parse AS 'com.example.Parse' LANGUAGE JAVA;

> CREATE VIEW ErrorEvents AS
SELECT ts, parsed.*
FROM RawEvents, LATERAL TABLE(Parse(line)) AS parsed;

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' M
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Problem:
Undefined behaviour

Flink Java API (Maven project)

```
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<code INT, severity INT,
                                zone STRING, server STRING,
                                message STRING>"))
public class Parse extends TableFunction<Row> {
    public void eval(String line) {
        String[] parts = line.split(" ");
        String eventType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            System.out.println("Parsed line: " + line);
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Example Application - Problems?

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, ... )
  WITH ( ... );

> ADD JAR 'file:///UDFs/target/UDFs-1.0.jar';
> CREATE FUNCTION Parse AS 'com.example.Parse' LANGUAGE JAVA;

> CREATE VIEW ErrorEvents AS
SELECT ts, parsed.*
FROM RawEvents, LATERAL TABLE(Parse(line)) AS parsed;

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' M
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Problem:
Undefined behaviour

Flink Java API (Maven project)

```
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<code INT, severity INT,
                                zone STRING, server STRING,
                                message STRING>"))
public class Parse extends TableFunction<Row> {
    public void eval(String line) {
        String[] parts = line.split(" ");
        String eventType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            System.out.println("Parsed line: " + line);
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Example Application - Problems?

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, ... )
  WITH ( ... );

> ADD JAR 'file:///UDFs/target/UDFs-1.0.jar';
> CREATE FUNCTION Parse AS 'com.example.Parse' LANGUAGE JAVA;

> CREATE VIEW ErrorEvents AS
SELECT ts, parsed.*
FROM RawEvents, LATERAL TABLE(Parse(line)) AS parsed;

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Flink Java API (Maven project)

```
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<code INT, severity INT,
                                zone STRING, server STRING,
                                message STRING>"))
public class Parse extends TableFunction<Row> {
    public void eval(String line) {
        String[] parts = line.split(" ");
        String eventType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Example Application - Problems?

Flink SQL

```
> CREATE TABLE RawEvents(ts TIMESTAMP(3), line STRING, ... )
  WITH ( ... );

> CREATE TABLE Alerts(zone STRING, avg_severity INT, ... )
  WITH ( ... );

> ADD JAR 'file:///UDFs/target/UDFs-1.0.jar';
> CREATE FUNCTION Parse AS 'com.example.Parse' LANGUAGE JAVA;

> CREATE VIEW ErrorEvents AS
SELECT ts, parsed.*
FROM RawEvents, LATERAL TABLE(Parse(line)) AS parsed;

> INSERT INTO Alerts
SELECT zone, AVG(severity) AS avg_severity,
       window_start, window_end
FROM TABLE(HOP(TABLE ErrorEvents, DESCRIPTOR(ts),
               INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
WHERE code = 500
GROUP BY window_start, window_end, zone
HAVING AVG(severity) > 2;
```

Flink Java API (Maven project)

```
package com.example;
import org.apache.flink.*;

@FunctionHint(
    input = @DataTypeHint("STRING"),
    output = @DataTypeHint("ROW<code INT, severity INT,
                                zone STRING, server STRING,
                                message STRING>"))
public class Parse extends TableFunction<Row> {
    public void eval(String line) {
        String[] parts = line.split(" ");
        String eventType = parts[0];
        if (eventType.equals("ERROR")) {
            int code = Integer.parseInt(parts[1]);
            int severity = Integer.parseInt(parts[2]);
            String zone = parts[3];
            String server = parts[4];
            String message = parts[5];
            collect(Row.of(code,severity,zone,server,message));
        }
    }
}
```

Problem:
Close coupling

Other options?

Other options?

Flink DataStream API

```
1  StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
2
3  env.addSource(new FlinkKafkaConsumer<>("eventlog", ...))
4    .assignTimestampAndWatermarks( ... )
5    .flatMap(new Parse())
6    .filter(event -> event.getCode() == 500)
7    .map(event -> Tuple2.of(event.getZone(), event.getSeverity()))
8    .keyBy(tuple -> tuple.f0)
9    .window(SlidingEventTimeWindows.of(Time.minutes(10), Time.minutes(1)))
10   .apply((zone, window, errors, out) -> {
11       double avgSeverity = errors.map(error -> error.severity).average();
12       out.collect(Tuple4.of(window.getStart(), window.getEnd(), zone, avgSeverity));
13   })
14   .map(tuple -> tuple.f0 + "," + tuple.f1 + "," + tuple.f2 + "," + tuple.f3)
15   .addSink(new FlinkKafkaProducer<>("alerts", ...));
16
17 env.execute();
```

Other options?

Flink DataStream API

```
1  StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
2
3  env.addSource(new FlinkKafkaConsumer<>("eventlog", ...))
4    .assignTimestampAndWatermarks( ... )
5    .flatMap(new Parse())
6    .filter(event -> event.getCode() == 500)
7    .map(event -> Tuple2.of(event.getZone(), event.getSeverity()))
8    .keyBy(tuple -> tuple.f0)
9    .window(SlidingEventTimeWindows.of(Time.minutes(10), Time.minutes(1)))
10   .apply((zone, window, errors, out) -> {
11       double avgSeverity = errors.map(error -> error.severity).average();
12       out.collect(Tuple4.of(window.getStart(), window.getEnd(), zone, avgSeverity));
13   })
14   .map(tuple -> tuple.f0 + "," + tuple.f1 + "," + tuple.f2 + "," + tuple.f3)
15   .addSink(new FlinkKafkaProducer<>("alerts", ...));
16
17 env.execute();
```

Other options?

Flink DataStream API

```
1  StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
2
3  env.addSource(new FlinkKafkaConsumer<>("eventlog", ...))
4    .assignTimestampAndWatermarks( ... )
5    .flatMap(new Parse())
6    .filter(event -> event.getCode() == 500)
7    .map(event -> Tuple2.of(event.getZone(), event.getSeverity()))
8    .keyBy(tuple -> tuple.f0)
9    .window(SlidingEventTimeWindows.of(Time.minutes(10), Time.minutes(1)))
10   .apply((zone, window, errors, out) -> {
11       double avgSeverity = errors.map(error -> error.severity).average();
12       out.collect(Tuple4.of(window.getStart(), window.getEnd(), zone, avgSeverity));
13   })
14   .map(tuple -> tuple.f0 + "," + tuple.f1 + "," + tuple.f2 + "," + tuple.f3)
15   .addSink(new FlinkKafkaProducer<>("alerts", ...));
16
17 env.execute();
```

Other options?

Flink DataStream API

```
1  StreamExecutionEnvironment env = StreamExecutionEnvironment.
2
3  env.addSource(new FlinkKafkaConsumer<>("eventlog", ...))
4    .assignTimestampAndWatermarks( ... )
5    .flatMap(new Parse())
6    .filter(event -> event.getCode() == 500)
7    .map(event -> Tuple2.of(event.getZone(), event.getSeverity()))
8    .keyBy(tuple -> tuple.f0)
9    .window(SlidingEventTimeWindows.of(Time.minutes(10), Time.minutes(1)))
10   .apply((zone, window, events, out) -> {
11       double avgSeverity = events.map(event -> event.severity).average();
12       out.collect(Tuple4.of(window.getStart(), window.getEnd(), zone, avgSeverity));
13   })
14   .map(tuple -> tuple.f0 + "," + tuple.f1 + "," + tuple.f2 + "," + tuple.f3)
15   .addSink(new FlinkKafkaProducer<>("alerts", ...));
16
17 env.execute();
```

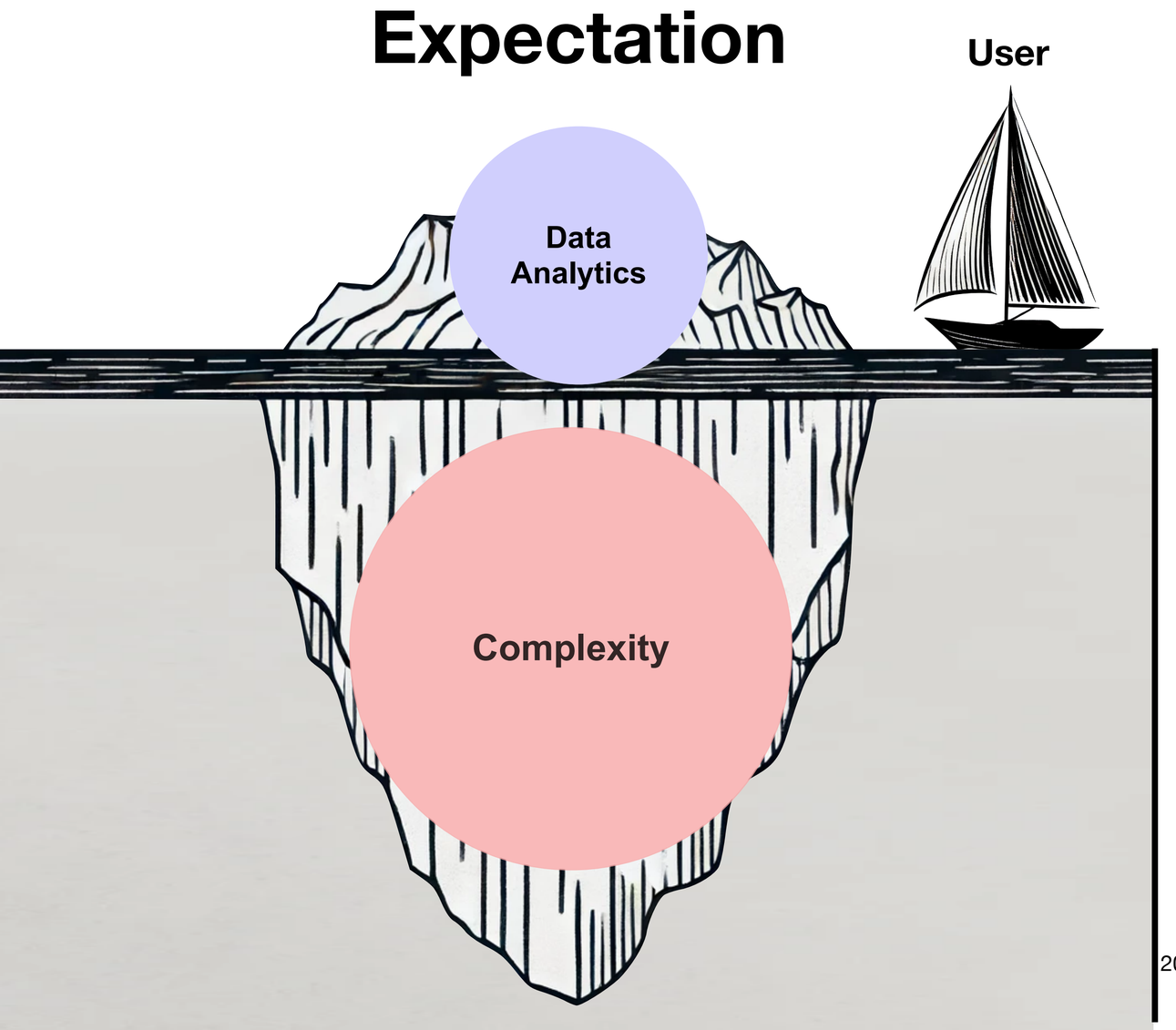
Runtime Error: The return type of function 'org.example.Main$$Lambda$263/0x00000008002e3040@36074e47' could not be determined automatically, due to **type erasure.** You can give type information hints by using the **returns(...) method** on the result of the transformation call.

Other options?

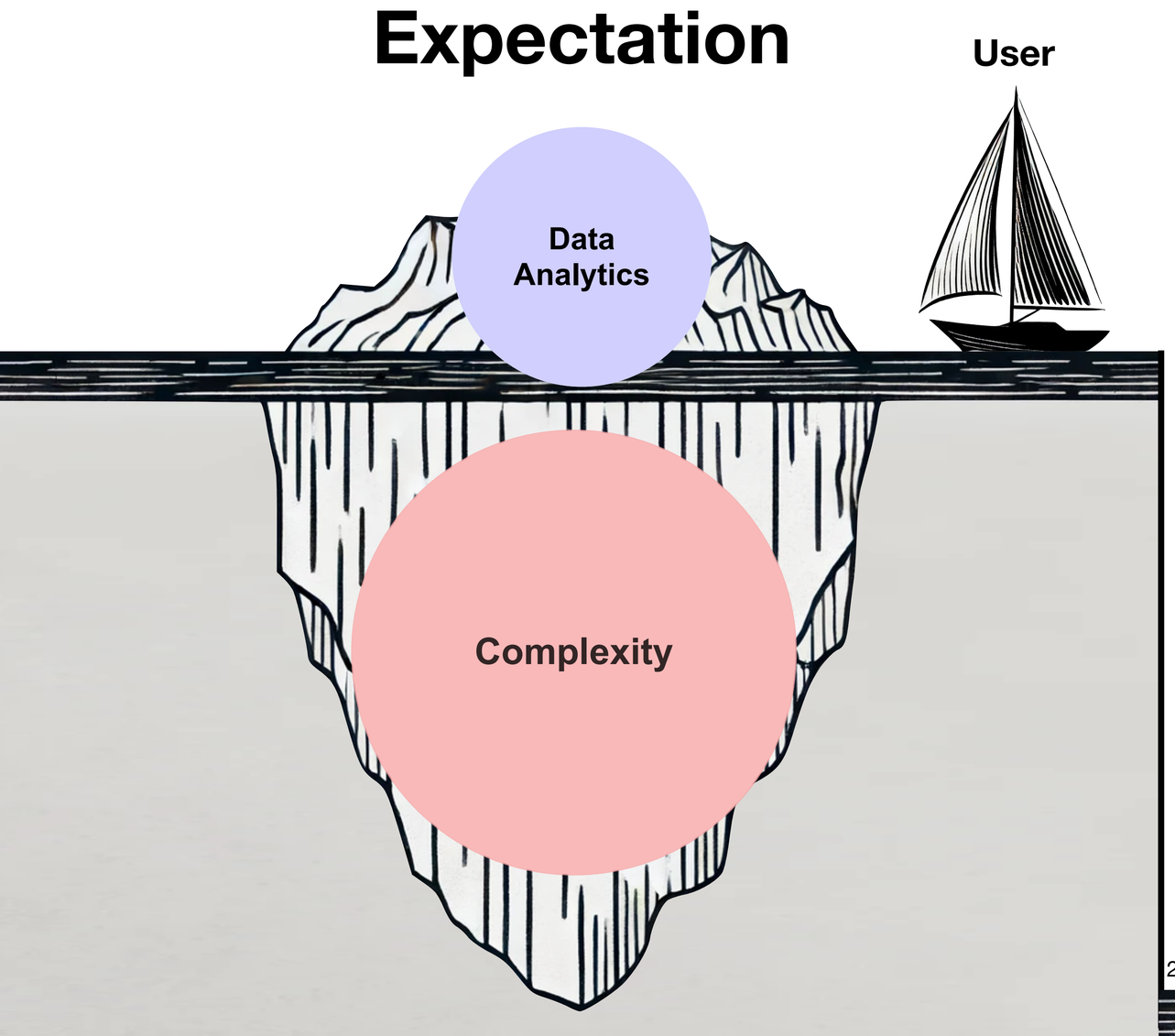
Flink DataStream API

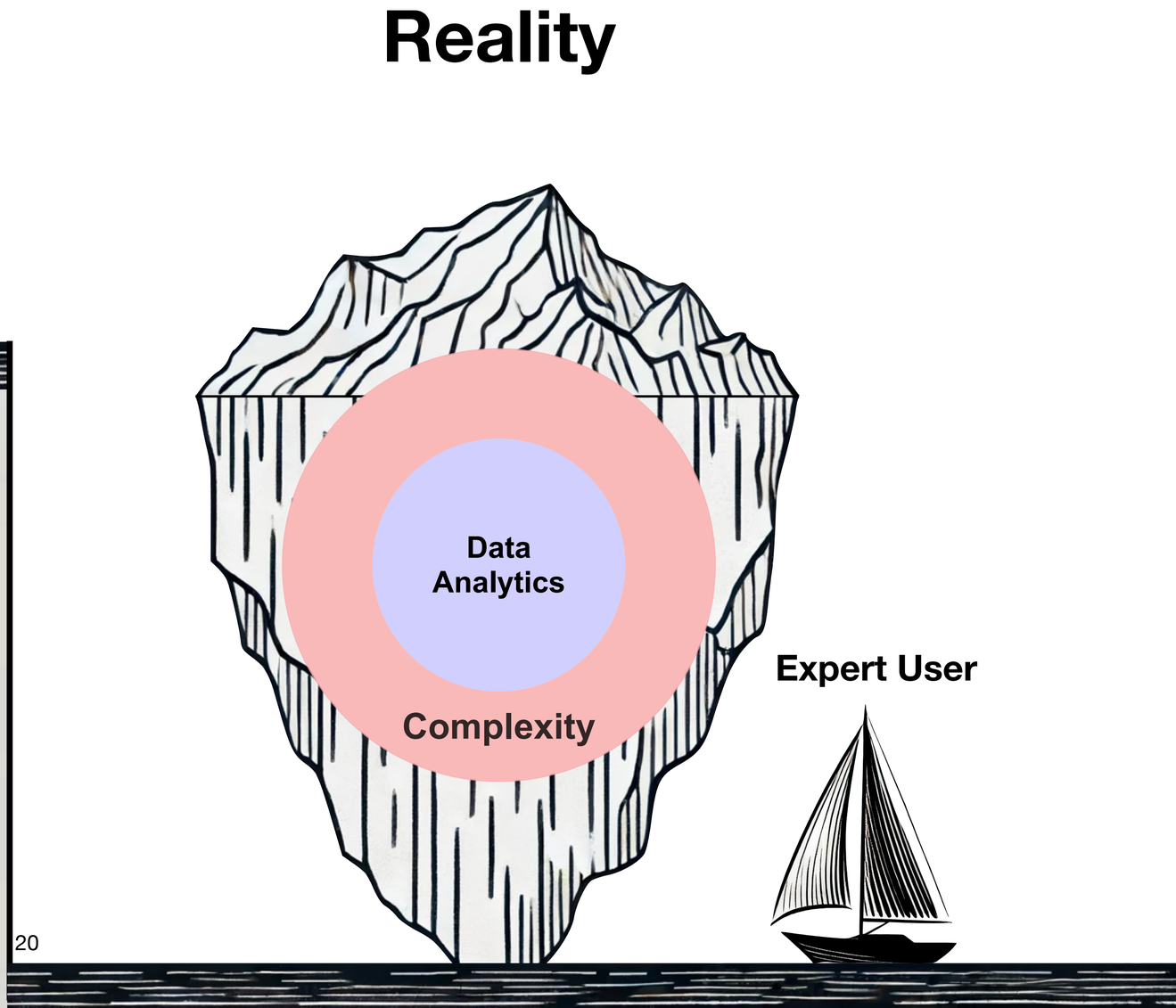
```
1  StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
2
3  env.addSource(new FlinkKafkaConsumer<>("eventlog", ...))
4    .assignTimestampAndWatermarks( ... )
5    .flatMap(new Parse())
6    .filter(event -> event.getCode() == 500)
7    .map(event -> Tuple2.of(event.getZone(), event.getSeverity()))
8    .returns(TypeInformation.of(new TypeHint<Tuple2<String, Integer>>() {}))
9    .keyBy(tuple -> tuple.f0)
10   .window(SlidingEventTimeWindows.of(Time.minutes(10), Time.minutes(1)))
11   .apply((zone, window, events, out) -> {
12       double avgSeverity = events.map(event -> event.severity).average();
13       out.collect(Tuple4.of(window.getStart(), window.getEnd(), zone, avgSeverity));
14   })
15   .map(tuple -> tuple.f0 + "," + tuple.f1 + "," + tuple.f2 + "," + tuple.f3)
16   .addSink(new FlinkKafkaProducer<>("alerts", ...));
17
18 env.execute();
```

The **Reality** of Dataflow Systems

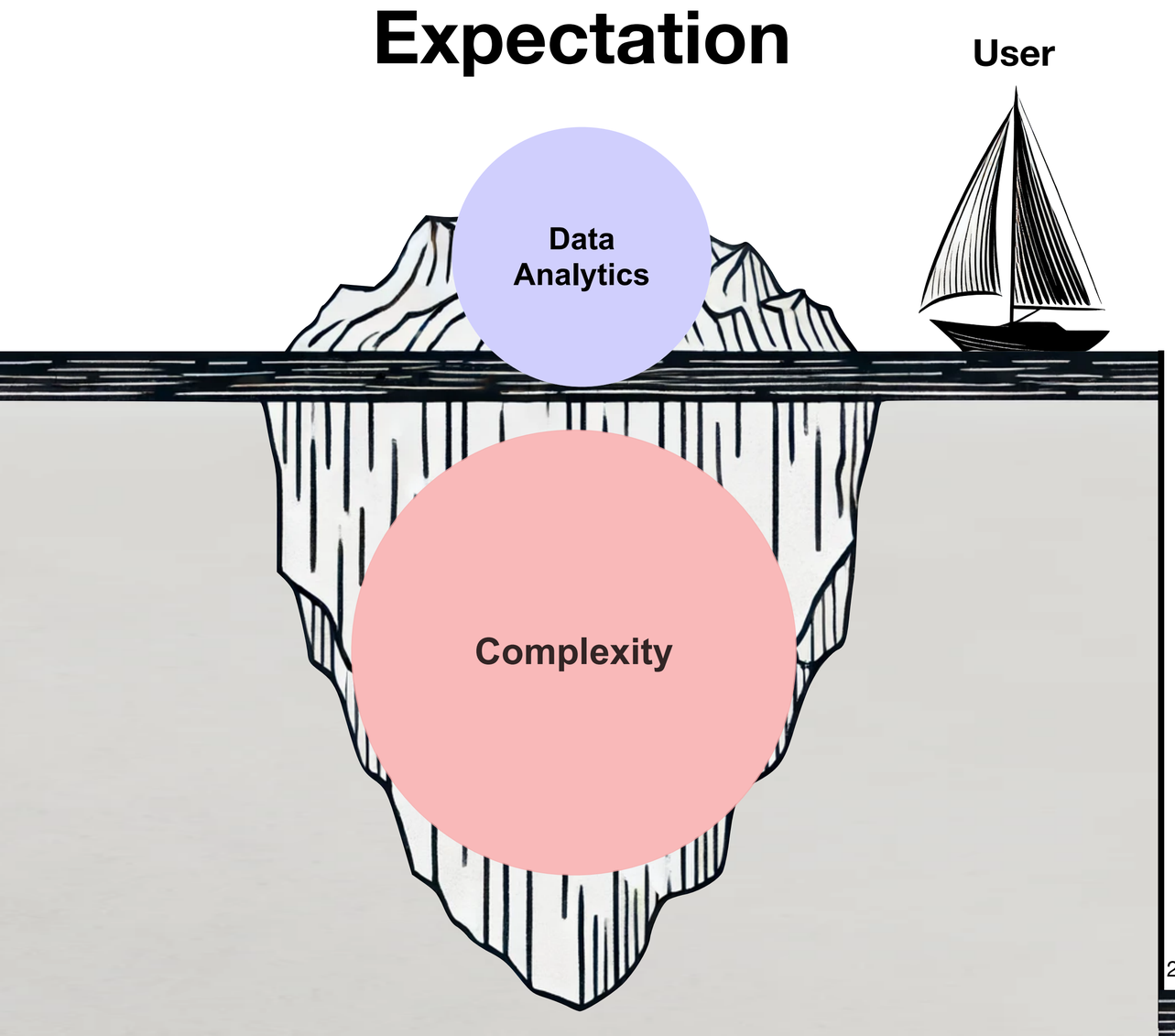


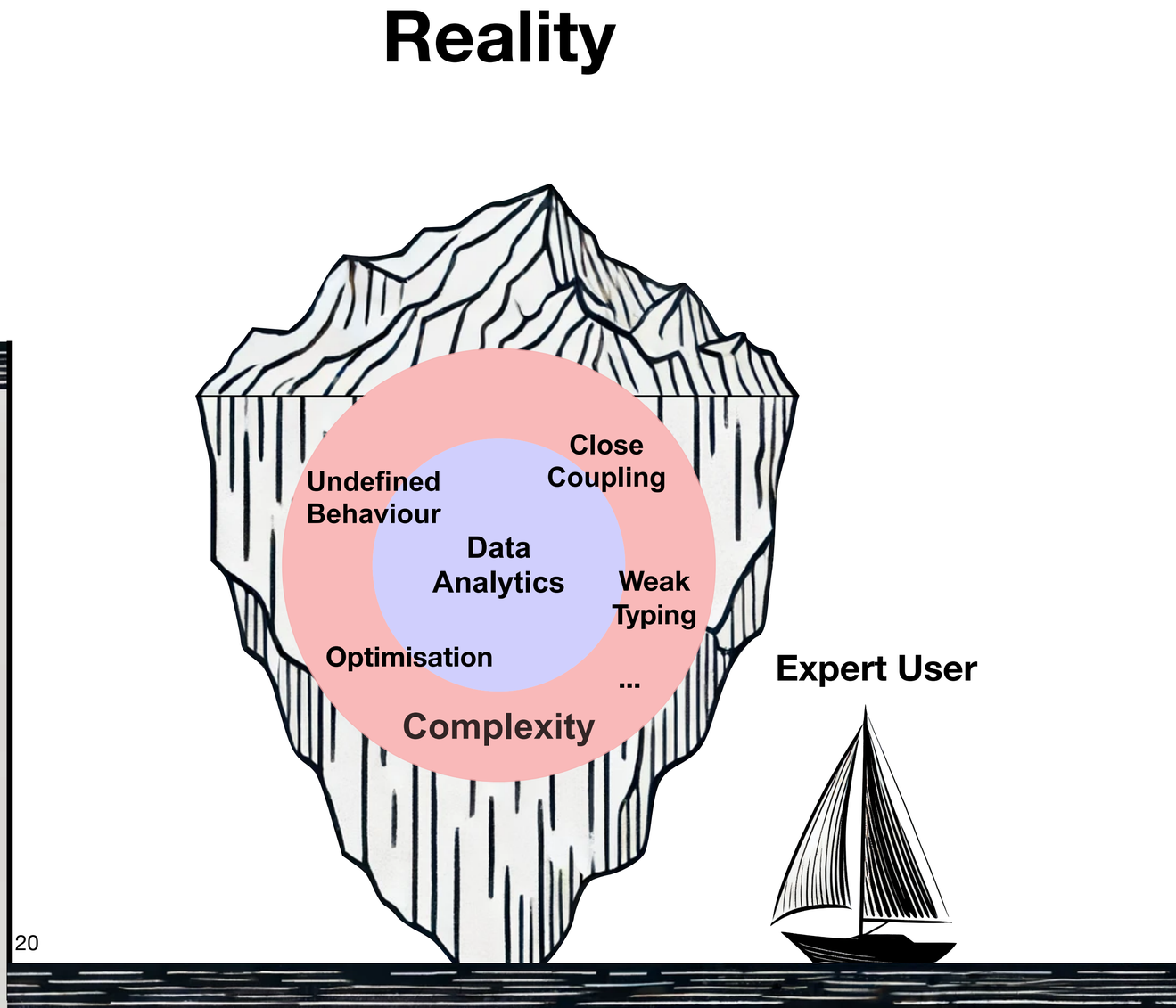
The **Reality** of Dataflow Systems





The **Reality** of Dataflow Systems





Solutions?

Solutions?

Extend Flink SQL with support for User Defined Functions

Solutions?

Extend Flink SQL with support for User Defined Functions

Create a new language


Streaming
Query Languages
AquaLang
General Purpose
Programming Languages
+ Safety, Conciseness, Optimisability
+ and more?
+ UDFs

AquaLang: Dataflow Programming

AquaLang: Dataflow Programming

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
```

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[ErrorEvent] = {
    val parts = line.split(" ");
    val eventType = parts[0];
    if eventType == "ERROR" {
        val code = parts[1].parse();
        val severity = parts[2].parse();
        val zone = parts[3];
        val server = parts[4];
        val message = parts[5];
        Some(ErrorEvent(code, severity, zone, server, message))
    } else {
        None
    }
}
```

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[ErrorEvent] = { … }

Stream[RawEvent]::source(Reader::kafka("eventlog", …), Format::csv('|'), _.time)
    .flatMap(event ⇒ parse(event.line))
    .filter(error ⇒ error.code == 500)
    .keyBy(error ⇒ error.zone)
    .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
        val avg_severity = errors.map(error ⇒ error.severity).average();
        record(zone, avg_severity, win.start, win.end)
    })
    .filter(data ⇒ data.avg_severity > 2)
    .sink(Writer::kafka("alerts", …), Format::json())
    .run(Backend::flink(…));
```

AquaLang: Dataflow Programming

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, se
3
4  def parse(line: String): Option[ErrorEvent] = { … }
5
6  Stream[RawEvent]::source(Reader::kafka("eventlog", …), Format::csv('|'), _.time)
7      .flatMap(event ⇒ parse(event.line))
8      .filter(error ⇒ error.code == 500)
9      .keyBy(error ⇒ error.zone)
10     .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
11         val avg_severity = errors.map(error ⇒ error.severity).average();
12         record(zone, avg_severity, win.start, win.end)
13     })
14     .filter(data ⇒ data.avg_severity > 2)
15     .sink(Writer::kafka("alerts", …), Format::json())
16     .run(Backend::flink(…));
```

No undefined behaviour:
UDFs are verified to be side-effect free.

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[ErrorEvent] = { … }

Stream[RawEvent]::source(Reader::kafka("eventlog", …), Format::csv('|'), _.time)
    .flatMap(event ⇒ parse(event.line))
    .filter(error ⇒ error.code == 500)
    .keyBy(error ⇒ error.zone)
    .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
        val avg_severity = errors.map(error ⇒ error.severity).average();
        record(zone, avg_severity, win.start, win.end)
    })
    .filter(data ⇒ data.avg_severity > 2)
    .sink(Writer::kafka("alerts", …), Format::json())
    .run(Backend::flink(…));
```

No close coupling:
Dataflow graphs can be run on different dataflow systems

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[Error] = { … }

Stream[RawEvent]::source(…)
    .flatMap(event ⇒ parse(event.line))
    .filter(error ⇒ error.code == 500)
    .keyBy(error ⇒ error.zone)
    .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
        val avg_severity = errors.map(error ⇒ error.severity).average();
        record(zone, avg_severity, win.start, win.end)
    })
    .filter(data ⇒ data.avg_severity > 2)
    .sink(…).run(…);
```

AquaLang: Dataflow Programming

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  def parse(line: String): Option[Error] = { … }
5
6  Stream[RawEvent]::source(…)
7      .flatMap(event ⇒ parse(event.line))
8      .filter(error ⇒ error.code == 500)
9      .keyBy(error ⇒ error.zone)
10     .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
11         val avg_severity = errors.map(error ⇒ error.severity).average();
12         record(zone, avg_severity, win.start, win.end)
13     })
14     .filter(data ⇒ data.avg_severity > 2)
15     .sink(…).run(…);
```

Functional Syntax

AquaLang: Dataflow Programming

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  def parse(line: String): Option[Error] = { … }
5
6  Stream[RawEvent]::source(…)
7      .flatMap(event ⇒ parse(event.line))
8      .filter(error ⇒ error.code == 500)
9      .keyBy(error ⇒ error.zone)
10     .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
11         val avg_severity = errors.map(error ⇒ error.severity).average();
12         record(zone, avg_severity, win.start, win.end)
13     })
14     .filter(data ⇒ data.avg_severity > 2)
15     .sink(…).run(…);
16
17 from event in Stream[RawEvent]::source(…)
18 from error in parse(event.line)
19 where error.code == 500
20 group error.zone
21     over win = Window::sliding(1m, 10m)
22     compute avg_severity = average of error.severity
23 where avg_severity > 2
24 select zone, avg_severity, win.start, win.end
25 into sink(…).run(…);
```

Functional Syntax

Relational Syntax

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[Error] = { … }

Stream[RawEvent]::source(…)
    .flatMap(event ⇒ parse(event.line))
    .filter(error ⇒ error.code == 500)
    .keyBy(error ⇒ error.zone)
    .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
        val avg_severity = errors.map(error ⇒ error.severity).average();
        record(zone, avg_severity, win.start, win.end)
    })
    .filter(data ⇒ data.avg_severity > 2)
    .sink(…).run(…);

from event in Stream[RawEvent]::source(…)
from error in parse(event.line)
where error.code == 500
group error.zone
    over win = Window::sliding(1m, 10m)
    compute avg_severity = average of error.severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink(…).run(…);
```

Functional Syntax

Semantically Equivalent

Relational Syntax

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[Error] = { … }

Stream[RawEvent]::source(…)
    .flatMap(event ⇒ parse(event.line))
    .filter(error ⇒ error.code == 500)
    .keyBy(error ⇒ error.zone)
    .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
        val avg_severity = errors.map(error ⇒ error.severity).average();
        record(zone, avg_severity, win.start, win.end)
    })
    .filter(data ⇒ data.avg_severity > 2)
    .sink(…).run(…);

from event in Stream[RawEvent]::source(…)
from error in parse(event.line)
where error.code == 500
group error.zone
    over win = Window::sliding(1m, 10m)
    compute avg_severity = average of error.severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink(…).run(…);
```

Functional Syntax

Semantically Equivalent

Relational Syntax

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[Error] = { … }

Stream[RawEvent]::source(…)
    .flatMap(event ⇒ parse(event.line))
    .filter(error ⇒ error.code == 500)
    .keyBy(error ⇒ error.zone)
    .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
        val avg_severity = errors.map(error ⇒ error.severity).average();
        record(zone, avg_severity, win.start, win.end)
    })
    .filter(data ⇒ data.avg_severity > 2)
    .sink(…).run(…);

from event in Stream[RawEvent]::source(…)
from error in parse(event.line)
where error.code == 500
group error.zone
    over win = Window::sliding(1m, 10m)
    compute avg_severity = average of error.severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink(…).run(…);
```

Functional Syntax

Semantically Equivalent

Relational Syntax

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[Error] = { … }

Stream[RawEvent]::source(…)
    .flatMap(event ⇒ parse(event.line))
    .filter(error ⇒ error.code == 500)
    .keyBy(error ⇒ error.zone)
    .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
        val avg_severity = errors.map(error ⇒ error.severity).average();
        record(zone, avg_severity, win.start, win.end)
    })
    .filter(data ⇒ data.avg_severity > 2)
    .sink(…).run(…);

from event in Stream[RawEvent]::source(…)
from error in parse(event.line)
where error.code == 500
group error.zone
    over win = Window::sliding(1m, 10m)
    compute avg_severity = average of error.severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink(…).run(…);
```

Functional Syntax

Semantically Equivalent

Relational Syntax

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[Error] = { … }

Stream[RawEvent]::source(…)
    .flatMap(event ⇒ parse(event.line))
    .filter(error ⇒ error.code == 500)
    .keyBy(error ⇒ error.zone)
    .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
        val avg_severity = errors.map(error ⇒ error.severity).average();
        record(zone, avg_severity, win.start, win.end)
    })
    .filter(data ⇒ data.avg_severity > 2)
    .sink(…).run(…);

from event in Stream[RawEvent]::source(…)
from error in parse(event.line)
where error.code == 500
group error.zone
    over win = Window::sliding(1m, 10m)
    compute avg_severity = average of error.severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink(…).run(…);
```

Functional Syntax

Semantically Equivalent

Relational Syntax

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[Error] = { … }

Stream[RawEvent]::source(…)
    .flatMap(event ⇒ parse(event.line))
    .filter(error ⇒ error.code == 500)
    .keyBy(error ⇒ error.zone)
    .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
        val avg_severity = errors.map(error ⇒ error.severity).average();
        record(zone, avg_severity, win.start, win.end)
    })
    .filter(data ⇒ data.avg_severity > 2)
    .sink(…).run(…);

from event in Stream[RawEvent]::source(…)
from error in parse(event.line)
where error.code == 500
group error.zone
    over win = Window::sliding(1m, 10m)
    compute avg_severity = average of error.severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink(…).run(…);
```

Functional Syntax

Semantically Equivalent

Relational Syntax

AquaLang: Dataflow Programming

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  def parse(line: String): Option[Error] = { … }
5
6  Stream[RawEvent]::source(…)
7      .flatMap(event ⇒ parse(event.line))
8      .filter(error ⇒ error.code == 500)
9      .keyBy(error ⇒ error.zone)
10     .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
11         val avg_severity = errors.map(error ⇒ error.severity).average();
12         record(zone, avg_severity, win.start, win.end)
13     })
14     .filter(data ⇒ data.avg_severity > 2)
15     .sink(…).run(…);
16
17 from event in Stream[RawEvent]::source(…)
18 from error in parse(event.line)
19 where error.code == 500
20 group error.zone
21     over win = Window::sliding(1m, 10m)
22     compute avg_severity = average of error.severity
23 where avg_severity > 2
24 select zone, avg_severity, win.start, win.end
25 into sink(…).run(…);
```

Functional Syntax

Semantically Equivalent

Relational Syntax

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[Error] = { … }

Stream[RawEvent]::source(…)
    .flatMap(event ⇒ parse(event.line))
    .filter(error ⇒ error.code == 500)
    .keyBy(error ⇒ error.zone)
    .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
        val avg_severity = errors.map(error ⇒ error.severity).average();
        record(zone, avg_severity, win.start, win.end)
    })
    .filter(data ⇒ data.avg_severity > 2)
    .sink(…).run(…);

from event in Stream[RawEvent]::source(…)
from error in parse(event.line)
where error.code == 500
group error.zone
    over win = Window::sliding(1m, 10m)
    compute avg_severity = average of error.severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink(…).run(…);
```

Functional Syntax

Semantically Equivalent

Relational Syntax

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[Error] = { … }

Stream[RawEvent]::source(…)
    .flatMap(event ⇒ parse(event.line))
    .filter(error ⇒ error.code == 500)
    .keyBy(error ⇒ error.zone)
    .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
        val avg_severity = errors.map(error ⇒ error.severity).average();
        record(zone, avg_severity, win.start, win.end)
    })
    .filter(data ⇒ data.avg_severity > 2)
    .sink(…).run(…);

from event in Stream[RawEvent]::source(…)
from error in parse(event.line)
where error.code == 500
group error.zone
    over win = Window::sliding(1m, 10m)
    compute avg_severity = average of error.severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink(…).run(…);
```

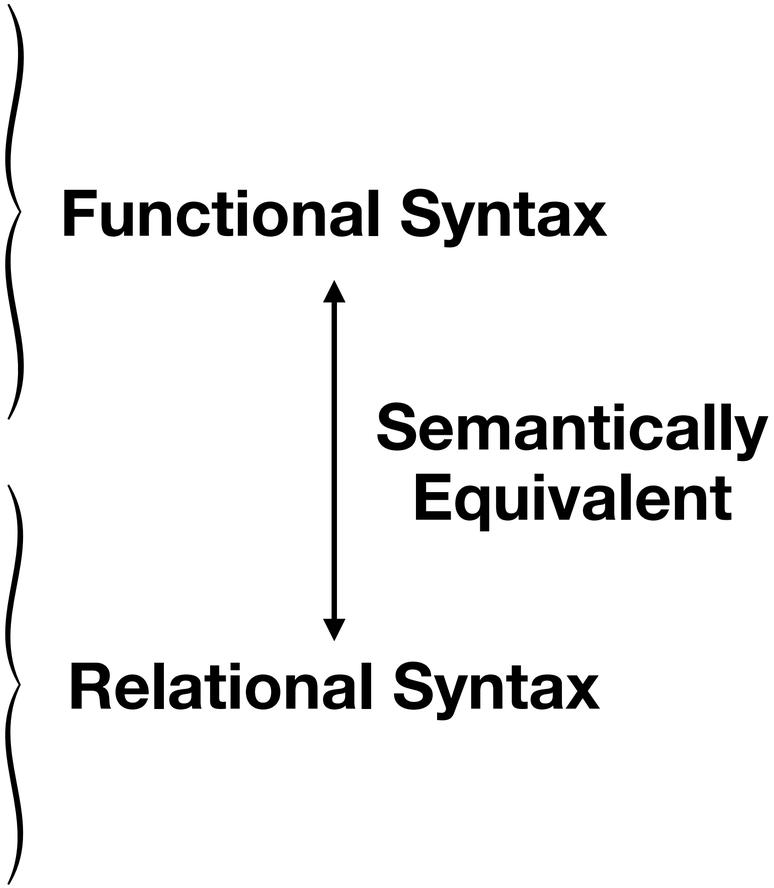
Functional Syntax

Semantically Equivalent

Relational Syntax

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[Error] = { … }

Stream[RawEvent]::source(…)
    .flatMap(event ⇒ parse(event.line))
    .filter(error ⇒ error.code == 500)
    .keyBy(error ⇒ error.zone)
    .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
        val avg_severity = errors.map(error ⇒ error.severity).average();
        record(zone, avg_severity, win.start, win.end)
    })
    .filter(data ⇒ data.avg_severity > 2)
    .sink(…).run(…);

from event in Stream[RawEvent]::source(…)
from error in parse(event.line)
where error.code == 500
group error.zone
    over win = Window::sliding(1m, 10m)
    compute avg_severity = average of error.severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink(…).run(…);
```



AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[Error] = { … }

Stream[RawEvent]::source(…)
    .flatMap(event ⇒ parse(event.line))
    .filter(error ⇒ error.code == 500)
    .keyBy(error ⇒ error.zone)
    .window(Window::sliding(1m, 10m), (errors, zone, win) ⇒ {
        val avg_severity = errors.map(error ⇒ error.severity).average();
        record(zone, avg_severity, win.start, win.end)
    })
    .filter(data ⇒ data.avg_severity > 2)
    .sink(…).run(…);

from event in Stream[RawEvent]::source(…)
from error in parse(event.line)
where error.code == 500
group error.zone
    over win = Window::sliding(1m, 10m)
    compute avg_severity = average of error.severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink(…).run(…);
```

Functional Syntax

Semantically Equivalent

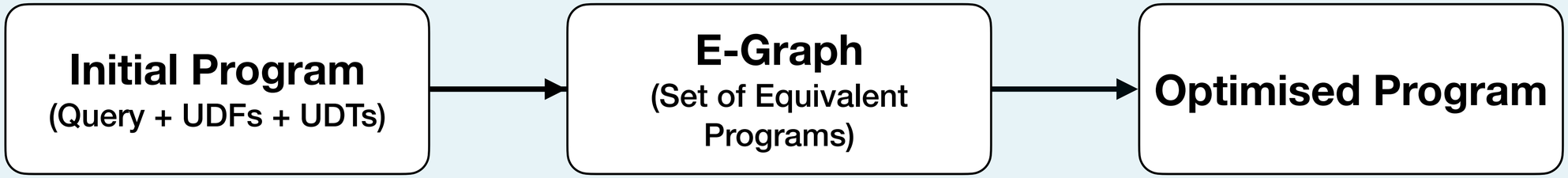
Relational Syntax

Optimisations

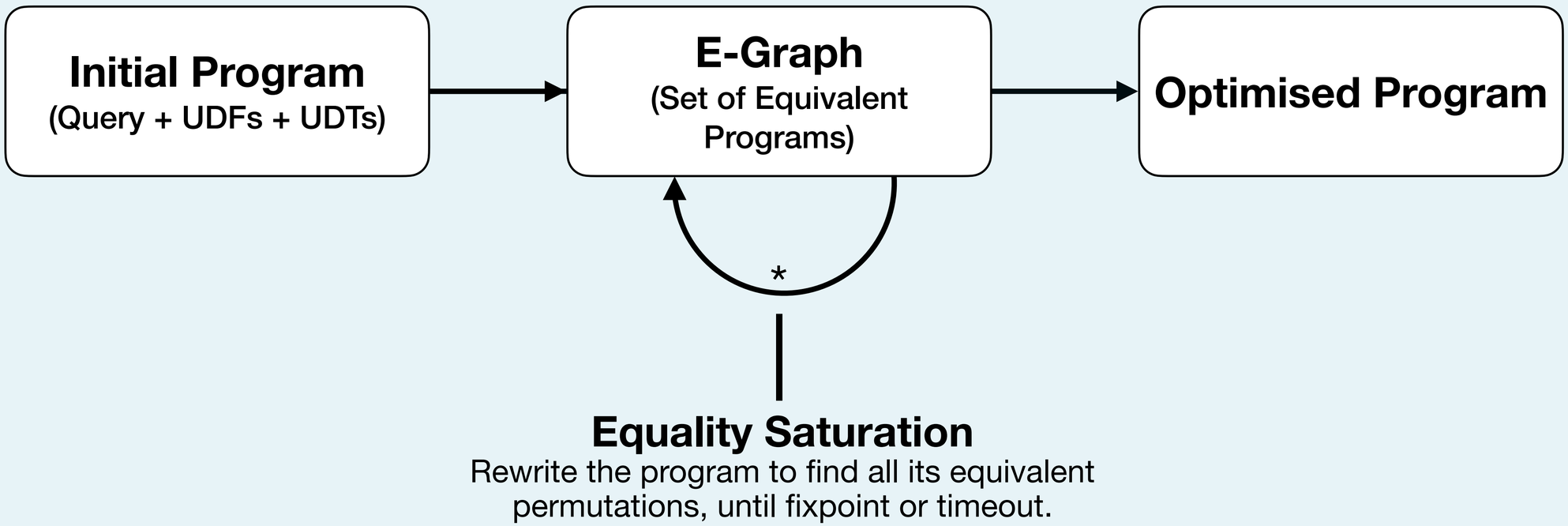
AquaLang: Optimisation via Equality Saturation

AquaLang: Optimisation via Equality Saturation

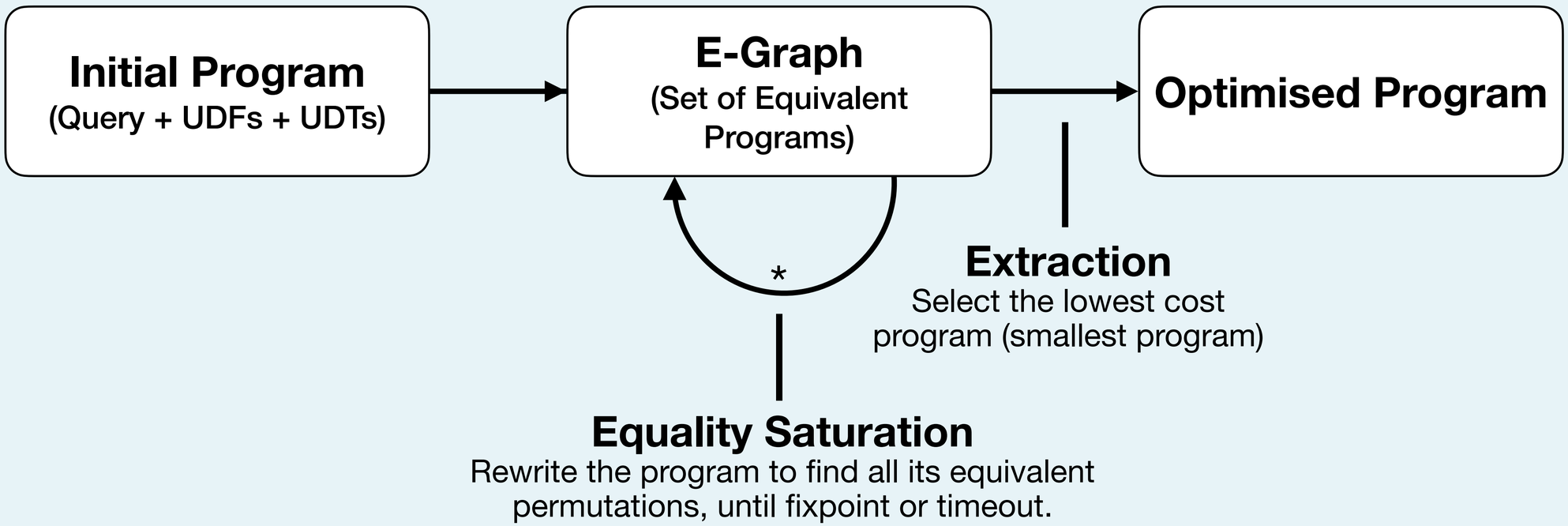
AquaLang: Optimisation via Equality Saturation



AquaLang: Optimisation via Equality Saturation



AquaLang: Optimisation via Equality Saturation



AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  def parse(line: String): Option[ErrorEvent] = { … }
5
6  from event in Stream[RawEvent]::source(…)
7  from error in parse(event.line)
8  where error.code == 500
9  group zone = error.zone
10     over win = Window::sliding(1min, 10min)
11     compute avg_severity = avg of error.severity
12 where avg_severity > 2
13 select zone, avg_severity, win.start, win.end
14 into sink( ... );
```

AquaLang: Optimisation

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def parse(line: String): Option[ErrorEvent] = { … }

from event in Stream[RawEvent]::source(…)
from error in parse(event.line)
where error.code == 500
group zone = error.zone
    over win = Window::sliding(1min, 10min)
    compute avg_severity = avg of error.severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  def parse(line: String): Option[RawError] = { … }
5
6  from event in Stream[RawEvent]::source(…)
7  from error in {
8      val parts = event.line.split(" ");
9      val eventType = parts[0];
10     if eventType == "ERROR" {
11         val code = parts[1].parse();
12         val severity = parts[2].parse();
13         val zone = parts[3];
14         val server = parts[4];
15         val message = parts[5];
16         Some(ErrorEvent(code, severity, zone, server, message))
17     } else { None }
18 }
19 where error.code == 500
20 group zone = error.zone
21     over win = Window::sliding(1min, 10min)
22     compute avg_severity = avg of error.severity
23 where avg_severity > 2
24 select zone, avg_severity, win.start, win.end
25 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  def parse(line: String): Option[RawError] = { … }
5
6  from event in Stream[RawEvent]::source(…)
7  from error in {
8      val parts = event.line.split(" ");
9      val eventType = parts[0];
10     if eventType == "ERROR" {
11         val code = parts[1].parse();
12         val severity = parts[2].parse();
13         val zone = parts[3];
14         val server = parts[4];
15         val message = parts[5];
16         Some(ErrorEvent(code, severity, zone, server, message))
17     } else { None }
18 }
19 where error.code == 500
20 group zone = error.zone
21     over win = Window::sliding(1min, 10min)
22     compute avg_severity = avg of error.severity
23 where avg_severity > 2
24 select zone, avg_severity, win.start, win.end
25 into sink( ... );
```

AquaLang: Optimisation

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

from event in Stream[RawEvent]::source(…)
from error in {
    val parts = event.line.split(" ");
    val eventType = parts[0];
    if eventType == "ERROR" {
        val code = parts[1].parse();
        val severity = parts[2].parse();
        val zone = parts[3];
        val server = parts[4];
        val message = parts[5];
        Some(ErrorEvent(code, severity, zone, server, message))
    } else { None }
}
where error.code == 500
group zone = error.zone
    over win = Window::sliding(1min, 10min)
    compute avg_severity = avg of error.severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink( ... );
```

AquaLang: Optimisation

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

from event in Stream[RawEvent]::source(…)
from error in {
    val parts = event.line.split(" ");
    val eventType = parts[0];
    if eventType == "ERROR" {
        val code = parts[1].parse();
        val severity = parts[2].parse();
        val zone = parts[3];
        val server = parts[4];
        val message = parts[5];
        Some(ErrorEvent(code, severity, zone, server, message))
    } else { None }
}
where error.code == 500
group zone = error.zone
    over win = Window::sliding(1min, 10min)
    compute avg_severity = avg of error.severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  from error in {
8      if eventType == "ERROR" {
9          val code = parts[1].parse();
10         val severity = parts[2].parse();
11         val zone = parts[3];
12         val server = parts[4];
13         val message = parts[5];
14         Some(ErrorEvent(code, severity, zone, server, message))
15     } else { None }
16 }
17 where error.code == 500
18 group zone = error.zone
19     over win = Window::sliding(1min, 10min)
20     compute avg_severity = avg of error.severity
21 where avg_severity > 2
22 select zone, avg_severity, win.start, win.end
23 into sink( ... );
```

AquaLang: Optimisation

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

from event in Stream[RawEvent]::source(…)
val parts = event.line.split(" ")
val eventType = parts[0]
from error in {
    if eventType == "ERROR" {
        val code = parts[1].parse();
        val severity = parts[2].parse();
        val zone = parts[3];
        val server = parts[4];
        val message = parts[5];
        Some(ErrorEvent(code, severity, zone, server, message))
    } else { None }
}
where error.code == 500
group zone = error.zone
    over win = Window::sliding(1min, 10min)
    compute avg_severity = avg of error.severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  from error in {
9      val code = parts[1].parse();
10     val severity = parts[2].parse();
11     val zone = parts[3];
12     val server = parts[4];
13     val message = parts[5];
14     Some(ErrorEvent(code, severity, zone, server, message))
15 }
16 where error.code == 500
17 group zone = error.zone
18     over win = Window::sliding(1min, 10min)
19     compute avg_severity = avg of error.severity
20 where avg_severity > 2
21 select zone, avg_severity, win.start, win.end
22 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  from error in {
9      val code = parts[1].parse();
10     val severity = parts[2].parse();
11     val zone = parts[3];
12     val server = parts[4];
13     val message = parts[5];
14     Some(ErrorEvent(code, severity, zone, server, message))
15 }
16 where error.code == 500
17 group zone = error.zone
18     over win = Window::sliding(1min, 10min)
19     compute avg_severity = avg of error.severity
20 where avg_severity > 2
21 select zone, avg_severity, win.start, win.end
22 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  val severity = parts[2].parse()
10 val zone = parts[3]
11 val server = parts[4]
12 val message = parts[5]
13 from error in {
14     Some(ErrorEvent(code, severity, zone, server, message))
15 }
16 where error.code == 500
17 group zone = error.zone
18     over win = Window::sliding(1min, 10min)
19     compute avg_severity = avg of error.severity
20 where avg_severity > 2
21 select zone, avg_severity, win.start, win.end
22 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  val severity = parts[2].parse()
10 val zone = parts[3]
11 val server = parts[4]
12 val message = parts[5]
13 from error in {
14     Some(ErrorEvent(code, severity, zone, server, message))
15 }
16 where error.code == 500
17 group zone = error.zone
18     over win = Window::sliding(1min, 10min)
19     compute avg_severity = avg of error.severity
20 where avg_severity > 2
21 select zone, avg_severity, win.start, win.end
22 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  val severity = parts[2].parse()
10 val zone = parts[3]
11 val server = parts[4]
12 val message = parts[5]
13 from error in Some(ErrorEvent(code, severity, zone, server, message))
14 where error.code == 500
15 group zone = error.zone
16     over win = Window::sliding(1min, 10min)
17     compute avg_severity = avg of error.severity
18 where avg_severity > 2
19 select zone, avg_severity, win.start, win.end
20 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  val severity = parts[2].parse()
10 val zone = parts[3]
11 val server = parts[4]
12 val message = parts[5]
13 from error in Some(ErrorEvent(code, severity, zone, server, message))
14 where error.code == 500
15 group zone = error.zone
16     over win = Window::sliding(1min, 10min)
17     compute avg_severity = avg of error.severity
18 where avg_severity > 2
19 select zone, avg_severity, win.start, win.end
20 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  val severity = parts[2].parse()
10 val zone = parts[3]
11 val server = parts[4]
12 val message = parts[5]
13 select error = ErrorEvent(code, severity, zone, server, message)
14 where error.code == 500
15 group zone = error.zone
16     over win = Window::sliding(1min, 10min)
17     compute avg_severity = avg of error.severity
18 where avg_severity > 2
19 select zone, avg_severity, win.start, win.end
20 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  val severity = parts[2].parse()
10 val zone = parts[3]
11 val server = parts[4]
12 val message = parts[5]
13 select error = ErrorEvent(code, severity, zone, server, message)
14 where error.code == 500
15 group zone = error.zone
16     over win = Window::sliding(1min, 10min)
17     compute avg_severity = avg of error.severity
18 where avg_severity > 2
19 select zone, avg_severity, win.start, win.end
20 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  val severity = parts[2].parse()
10 val zone = parts[3]
11 val server = parts[4]
12 val message = parts[5]
13 select code, severity, zone, server, message
14 where code == 500
15 group zone
16     over win = Window::sliding(1min, 10min)
17     compute avg_severity = avg of severity
18 where avg_severity > 2
19 select zone, avg_severity, win.start, win.end
20 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  val severity = parts[2].parse()
10 val zone = parts[3]
11 val server = parts[4]
12 val message = parts[5]
13 select code, severity, zone, server, message
14 where code == 500
15 group zone
16     over win = Window::sliding(1min, 10min)
17     compute avg_severity = avg of severity
18 where avg_severity > 2
19 select zone, avg_severity, win.start, win.end
20 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  val severity = parts[2].parse()
10 val zone = parts[3]
11
12
13 select code, severity, zone
14 where code == 500
15 group zone
16     over win = Window::sliding(1min, 10min)
17     compute avg_severity = avg of severity
18 where avg_severity > 2
19 select zone, avg_severity, win.start, win.end
20 into sink( ... );
```

AquaLang: Optimisation

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

from event in Stream[RawEvent]::source(…)
val parts = event.line.split(" ")
val eventType = parts[0]
where eventType == "ERROR"
val code = parts[1].parse()
val severity = parts[2].parse()
val zone = parts[3]
select code, severity, zone
where code == 500
group zone
    over win = Window::sliding(1min, 10min)
    compute avg_severity = avg of severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  val severity = parts[2].parse()
10 val zone = parts[3]
11 select code, severity, zone
12 where code == 500
13 group zone
14     over win = Window::sliding(1min, 10min)
15     compute avg_severity = avg of severity
16 where avg_severity > 2
17 select zone, avg_severity, win.start, win.end
18 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  where code == 500
10 val severity = parts[2].parse()
11 val zone = parts[3]
12 select code, severity, zone
13 group zone
14     over win = Window::sliding(1min, 10min)
15     compute avg_severity = avg of severity
16 where avg_severity > 2
17 select zone, avg_severity, win.start, win.end
18 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  where code == 500
10 val severity = parts[2].parse()
11 val zone = parts[3]
12 select code, severity, zone
13 group zone
14     over win = Window::sliding(1min, 10min)
15     compute avg_severity = avg of severity
16 where avg_severity > 2
17 select zone, avg_severity, win.start, win.end
18 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  where code == 500
10 val severity = parts[2].parse()
11 val zone = parts[3]
12 select severity, zone
13 group zone
14     over win = Window::sliding(1min, 10min)
15     compute avg_severity = avg of severity
16 where avg_severity > 2
17 select zone, avg_severity, win.start, win.end
18 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  where code == 500
10 val severity = parts[2].parse()
11 val zone = parts[3]
12 select severity, zone
13 group zone
14     over win = Window::sliding(1min, 10min)
15     compute avg_severity = avg of severity
16 where avg_severity > 2
17 select zone, avg_severity, win.start, win.end
18 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  val eventType = parts[0]
7  where eventType == "ERROR"
8  val code = parts[1].parse()
9  where code == 500
10 val severity = parts[2].parse()
11 val zone = parts[3]
12 select severity, zone
13 group zone
14     over win = Window::sliding(1min, 10min)
15     compute avg_severity = avg of severity
16 where avg_severity > 2
17 select zone, avg_severity, win.start, win.end
18 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6
7  where parts[0] == "ERROR"
8
9  where parts[1].parse() == 500
10
11
12 select severity = parts[2].parse(), zone = parts[3]
13 group zone
14     over win = Window::sliding(1min, 10min)
15     compute avg_severity = avg of severity
16 where avg_severity > 2
17 select zone, avg_severity, win.start, win.end
18 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6
7  where parts[0] == "ERROR"
8
9  where parts[1].parse() == 500
10
11
12 select severity = parts[2].parse(), zone = parts[3]
13 group zone
14     over win = Window::sliding(1min, 10min)
15     compute avg_severity = avg of severity
16 where avg_severity > 2
17 select zone, avg_severity, win.start, win.end
18 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  where parts[0] == "ERROR"
7  where parts[1].parse() == 500
8  select severity = parts[2].parse(), zone = parts[3]
9  group zone
10     over win = Window::sliding(1min, 10min)
11     compute avg_severity = avg of severity
12 where avg_severity > 2
13 select zone, avg_severity, win.start, win.end
14 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  where parts[0] == "ERROR"
7  where parts[1].parse() == 500
8  select severity = parts[2].parse(), zone = parts[3]
9  group zone
10     over win = Window::sliding(1min, 10min)
11     compute avg_severity = avg of severity
12 where avg_severity > 2
13 select zone, avg_severity, win.start, win.end
14 into sink( ... );
```

AquaLang: Optimisation

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  where parts[0] == "ERROR"
7    and parts[1].parse() == 500
8  select severity = parts[2].parse(), zone = parts[3]
9  group zone
10     over win = Window::sliding(1min, 10min)
11     compute avg_severity = avg of severity
12 where avg_severity > 2
13 select zone, avg_severity, win.start, win.end
14 into sink( ... );
```

AquaLang: Optimisation

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

from event in Stream[RawEvent]::source(…)
val parts = event.line.split(" ")
where parts[0] == "ERROR"
  and parts[1].parse() == 500
select severity = parts[2].parse(), zone = parts[3]
group zone
    over win = Window::sliding(1min, 10min)
    compute avg_severity = avg of severity
where avg_severity > 2
select zone, avg_severity, win.start, win.end
into sink( ... );
```

More features

AquaLang: Dataflow Programming

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  where parts[0] == "ERROR"
7    and parts[1].parse() == 500
8  select severity = parts[2].parse(), zone = parts[3]
9  group zone
10     over win = Window::sliding(1min, 10min)
11     compute avg_severity = avg of severity
12 where avg_severity > 2
13 select zone, avg_severity, win.start, win.end
14 into sink( ... );
```

AquaLang: Dataflow Programming

```
1  struct RawEvent(ts:Time, line:String);
2  struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);
3
4  from event in Stream[RawEvent]::source(…)
5  val parts = event.line.split(" ")
6  where parts[0] == "ERROR"
7    and parts[1].parse() == 500
8  select severity = parts[2].parse(), zone = parts[3]
9  group zone
10     over win = Window::sliding(1min, 10min)
11     compute avg_severity = avg of severity
12 where avg_severity > 2
13 select zone, avg_severity, win.start, win.end
14 into sink( ... );
```

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def queryAlerts(input: Stream[RawEvent], errorCode: u8): Stream[_] =
    from event in input
    val parts = event.line.split(" ")
    where parts[0] == "ERROR"
      and parts[1].parse() == errorCode
    select severity = parts[2].parse(), zone = parts[3]
    group zone
      over win = Window::sliding(1min, 10min)
      compute avg_severity = avg of severity
    where avg_severity > 2
    select zone, avg_severity, win.start, win.end
    into sink( ... );
```

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def queryAlerts(input: Stream[RawEvent], errorCode: u8): Stream[_] =
    from event in input
    val parts = event.line.split(" ")
    where parts[0] == "ERROR"
      and parts[1].parse() == errorCode
    select severity = parts[2].parse(), zone = parts[3]
    group zone
      over win = Window::sliding(1min, 10min)
      compute avg_severity = avg of severity
    where avg_severity > 2
    select zone, avg_severity, win.start, win.end
    into sink( ... );
```

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def queryAlerts(input: Stream[RawEvent], errorCode: u8): Stream[_] =
    from event in input
    val parts = event.line.split(" ")
    where parts[0] == "ERROR"
      and parts[1].parse() == errorCode
    select severity = parts[2].parse(), zone = parts[3]
    group zone
      over win = Window::sliding(1min, 10min)
      compute avg_severity = avg of severity
    where avg_severity > 2
    select zone, avg_severity, win.start, win.end
    into sink( ... );
```

AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def queryAlerts(input: Stream[RawEvent], errorCode: u8): Stream[_] =
    from event in input
    val parts = event.line.split(" ")
    where parts[0] == "ERROR"
      and parts[1].parse() == errorCode
    select severity = parts[2].parse(), zone = parts[3]
    group zone
      over win = Window::sliding(1min, 10min)
      compute avg_severity = avg of severity
    where avg_severity > 2
    select zone, avg_severity, win.start, win.end
    into sink( ... );
```

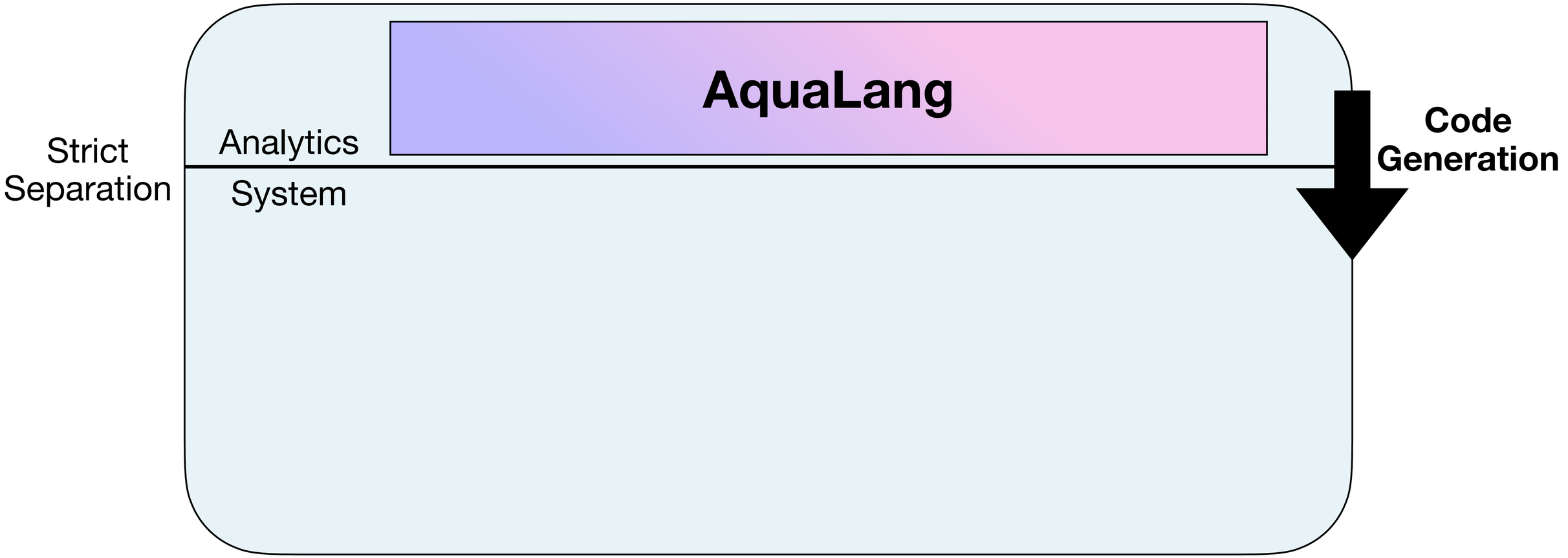
AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def queryAlerts(input: Stream[RawEvent], errorCode: u8): Stream[_] =
    from event in input
    val parts = event.line.split(" ")
    where parts[0] == "ERROR"
      and parts[1].parse() == errorCode
    select severity = parts[2].parse(), zone = parts[3]
    group zone
      over win = Window::sliding(1min, 10min)
      compute avg_severity = avg of severity
    where avg_severity > 2
    select zone, avg_severity, win.start, win.end
    into sink( ... );

from alert in queryAlerts(Stream::source(…), 401)
where alert.zone == "Z2"
into sink(…).run(…);
```

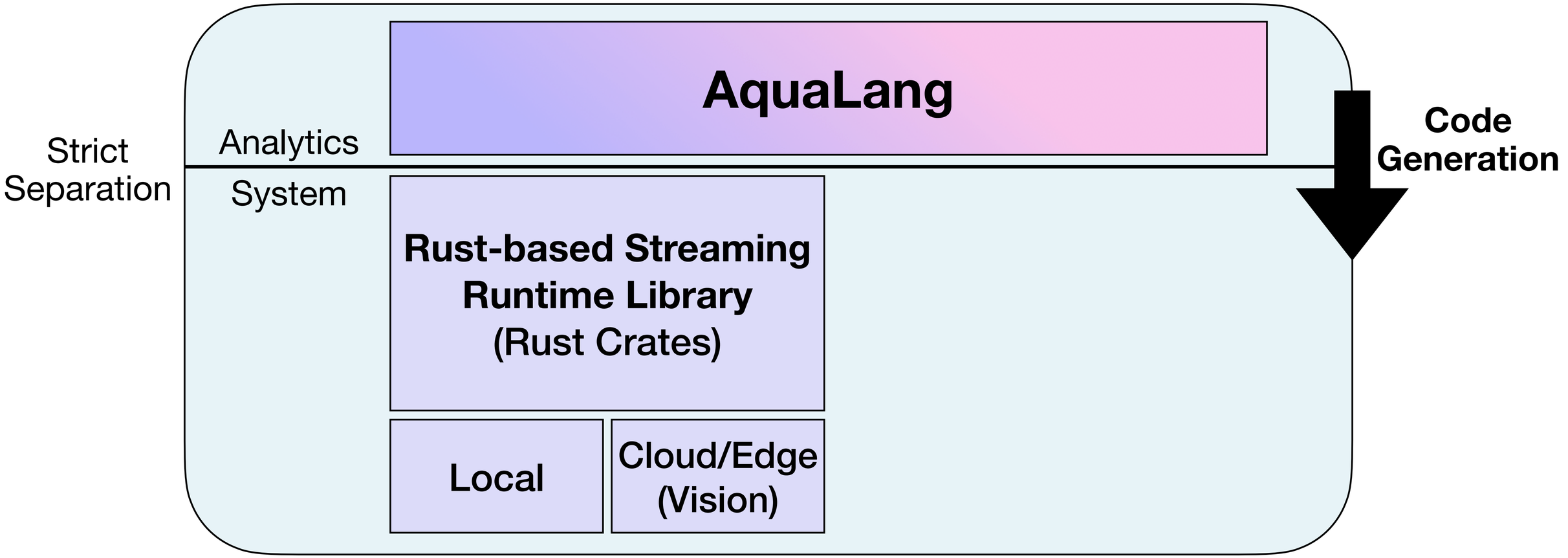
AquaLang: Dataflow Programming

```
struct RawEvent(ts:Time, line:String);
struct ErrorEvent(code:u8, severity:u8, zone:String, server:String, message:String);

def queryAlerts(input: Stream[RawEvent], errorCode: u8): Stream[_] =
    from event in input
    val parts = event.line.split(" ")
    where parts[0] == "ERROR"
      and parts[1].parse() == errorCode
    select severity = parts[2].parse(), zone = parts[3]
    group zone
      over win = Window::sliding(1min, 10min)
      compute avg_severity = avg of severity
    where avg_severity > 2
    select zone, avg_severity, win.start, win.end
    into sink( ... );

from alert in queryAlerts(Stream::source(…), 401)
where alert.zone == "Z2"
into sink(…).run(…);
```

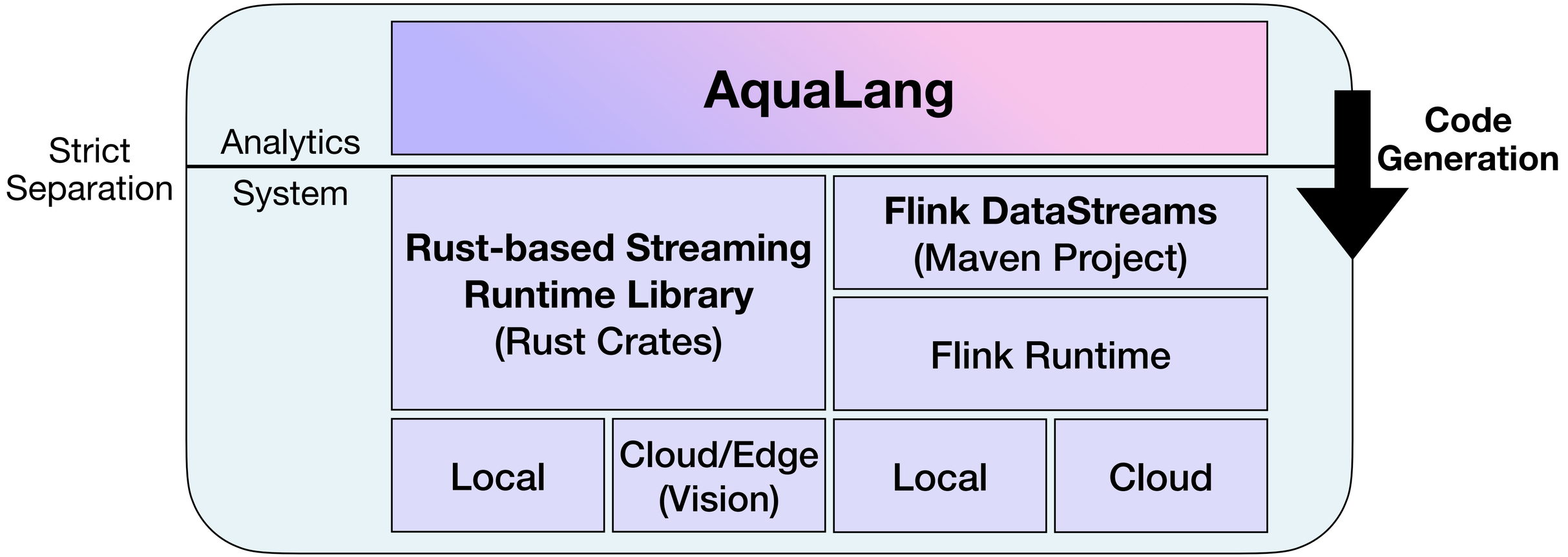
AquaLang: Code Generation



AquaLang: Code Generation



AquaLang: Code Generation



Conclusion

- **AquaLang** is a programming language that targets streaming dataflow systems
- **AquaLang** is designed to be compositional, safe, fast, and easy to use
- We are aiming to release a beta version later this year
- If you find **AquaLang** interesting, please get in touch! :)
  - **Website**: https://aqua-language.github.io/
  - **GitHub**: https://github.com/aqua-language/aqua
  - **Paper**: AquaLang - A Dataflow Programming Language (DEBS'24)