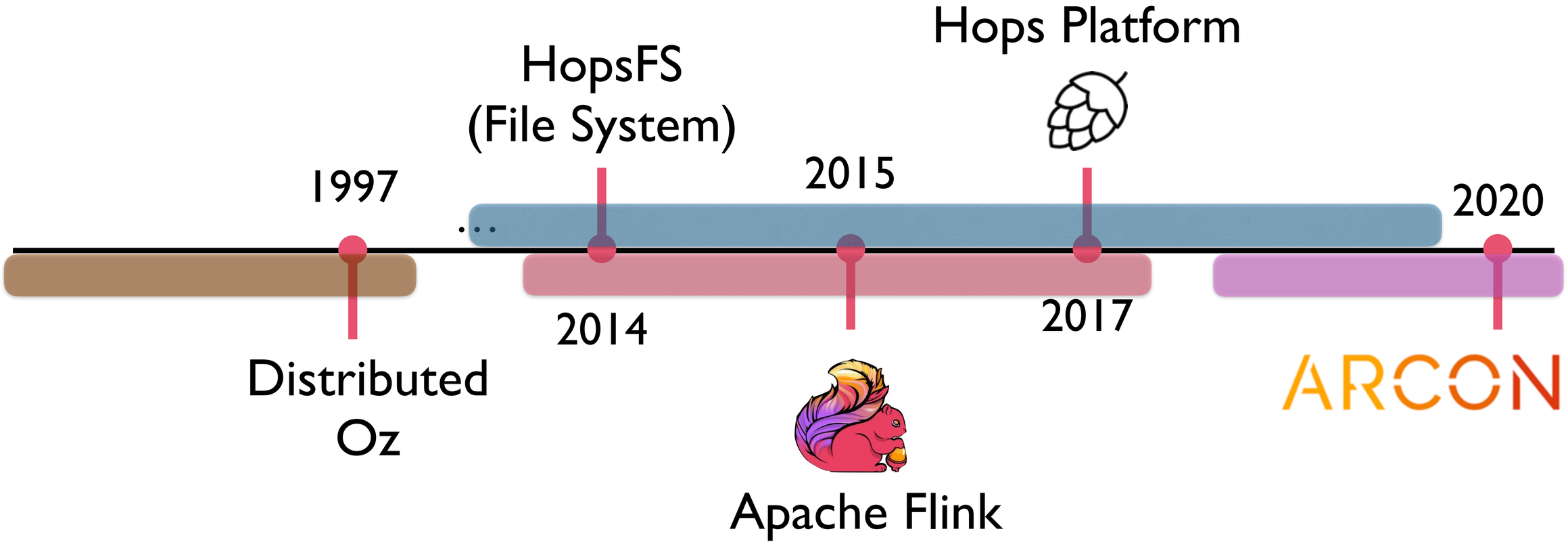
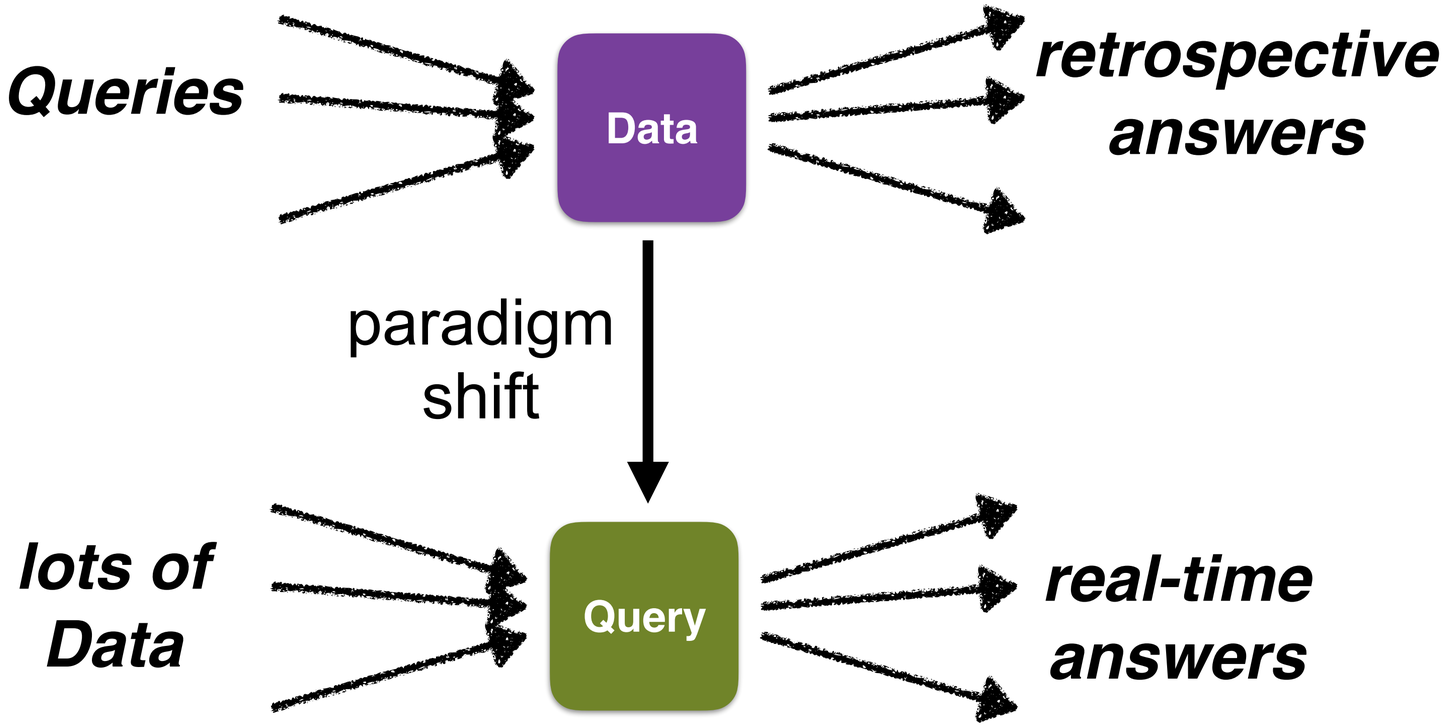
Seamless Batch and Stream Computation on Heterogeneous Hardware with Arcon

Paris Carbone
Research Institutes of Sweden

Open-Source @ DS Group



The Paradigm Shift Some Missed



The Real-Time Analytics Stack

High Level Models	Stream SQL, CEP…
Compute	Flink, Beam, Kafka-Streams, Apex, Storm
Storage	Kafka, Pub/Sub, Kinesis, Pravega…

- **Data Stream Processing** as a 24/7 execution paradigm

Apache Flink Foundations



Data Streams, Iterations,
Fault Tolerance, Window Aggregation

- Top-level Apache Project
- #1 stream processor (2020)
- Production-Proof

- > 400 contributors
- 100s of deployments

commercial deployments


influenced



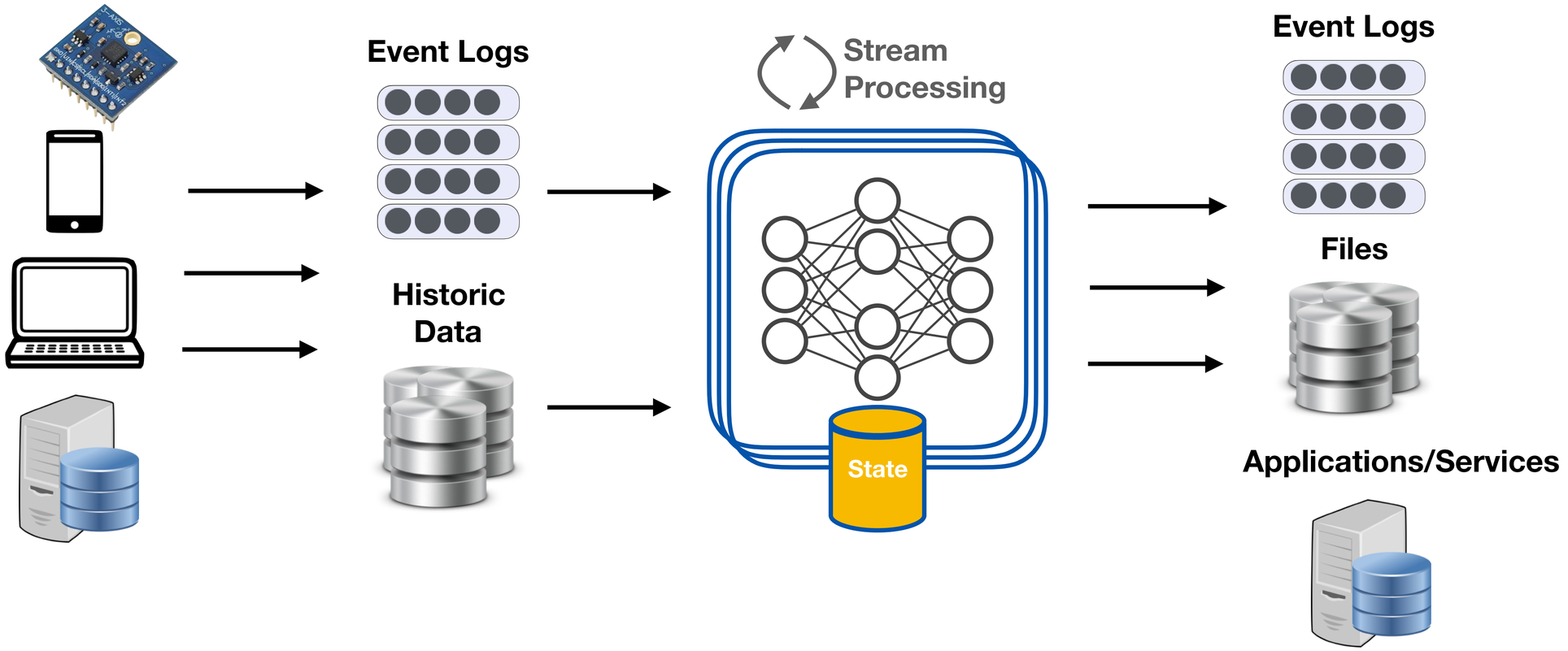






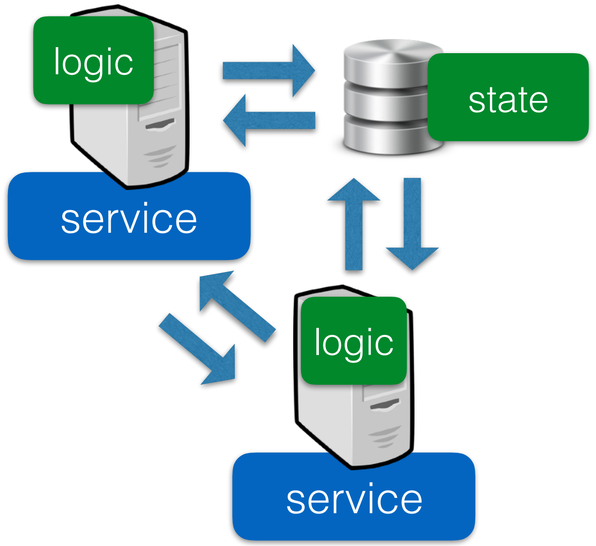



Stream Processors



Actors vs Streams

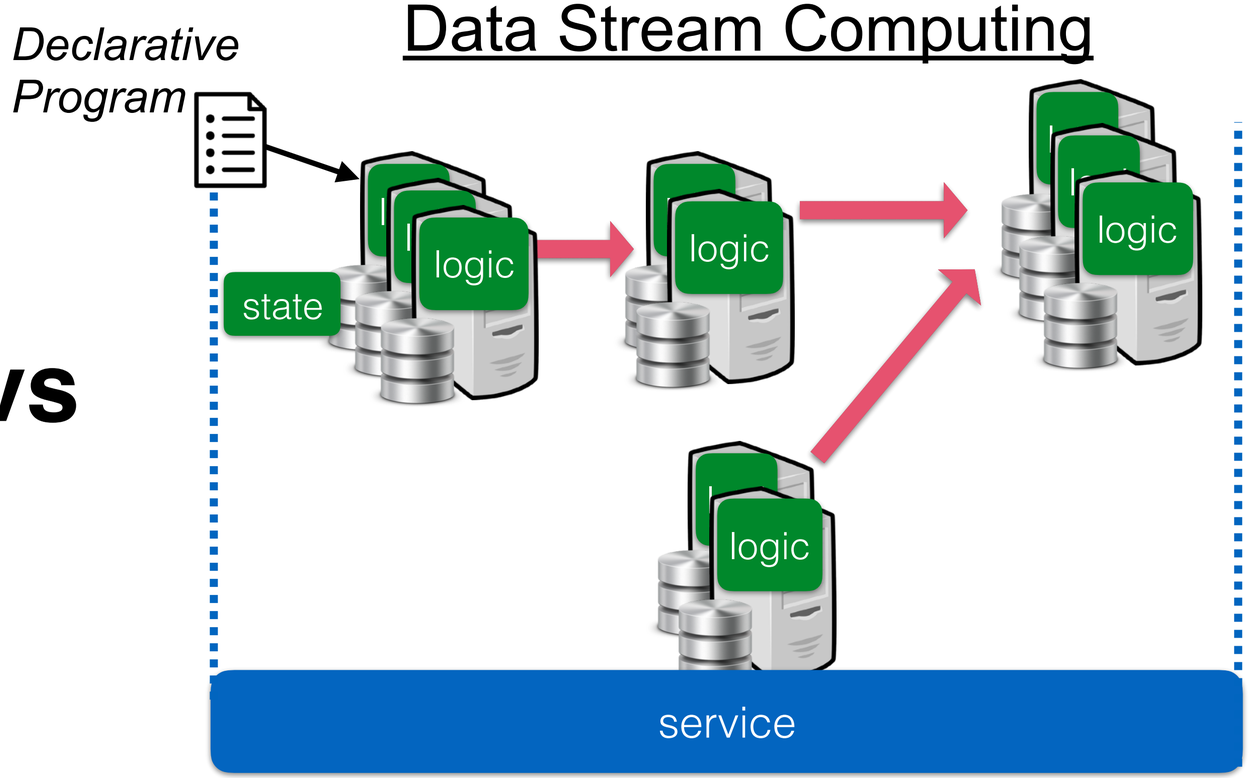
Actor Programming



- Low-Level Event-Based Programming
- Manual/External State
- Not Robust: Manual Fault Tolerance

vs

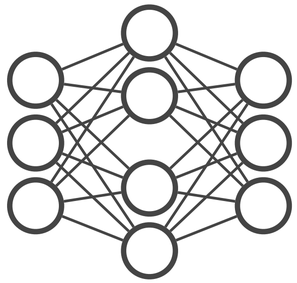
Data Stream Computing



- Declarative Programming
- State Managed by the system
- Robust: Built-in Fault Tolerance

Program Hierarchy in Flink

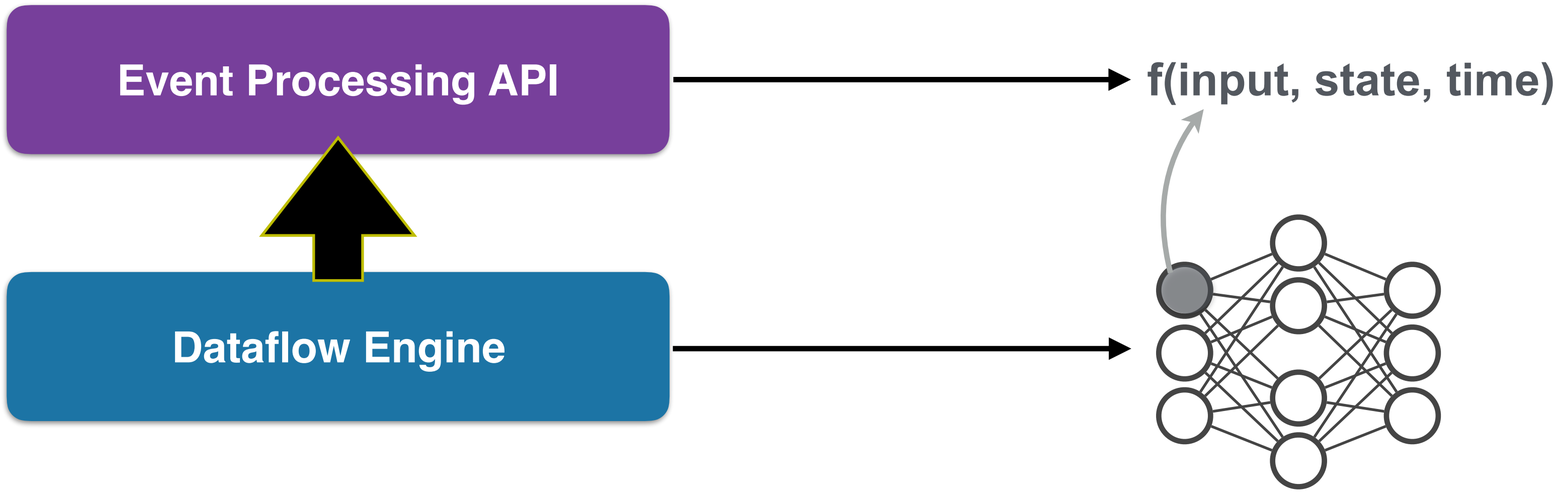
Dataflow Engine



Automates

- Fault Tolerance
- Scalability
- Monitoring/IO Management

Program Hierarchy in Flink

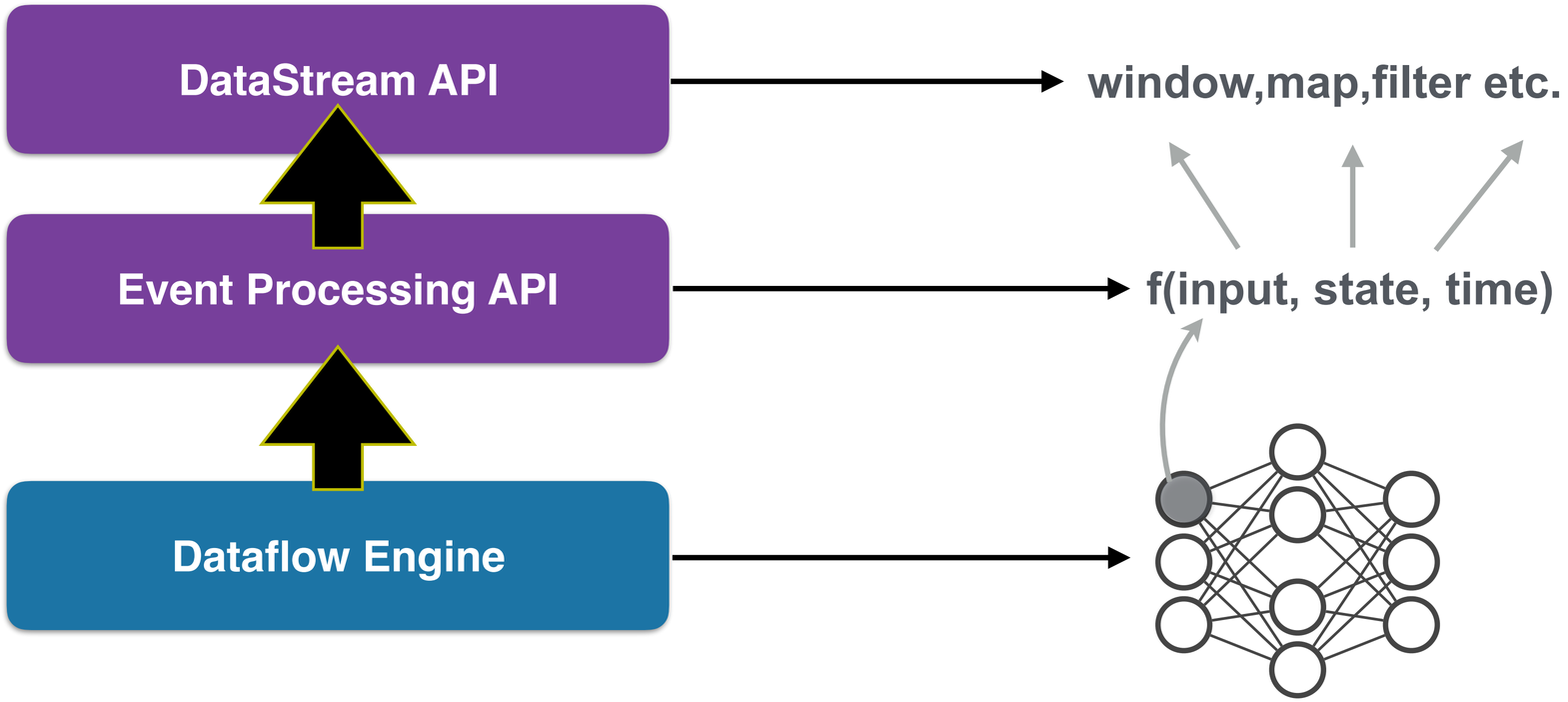


Automates

- Dynamic program state
- Operations on out-of-order streams

- Fault Tolerance
- Scalability
- Monitoring/IO Management

Program Hierarchy in Flink



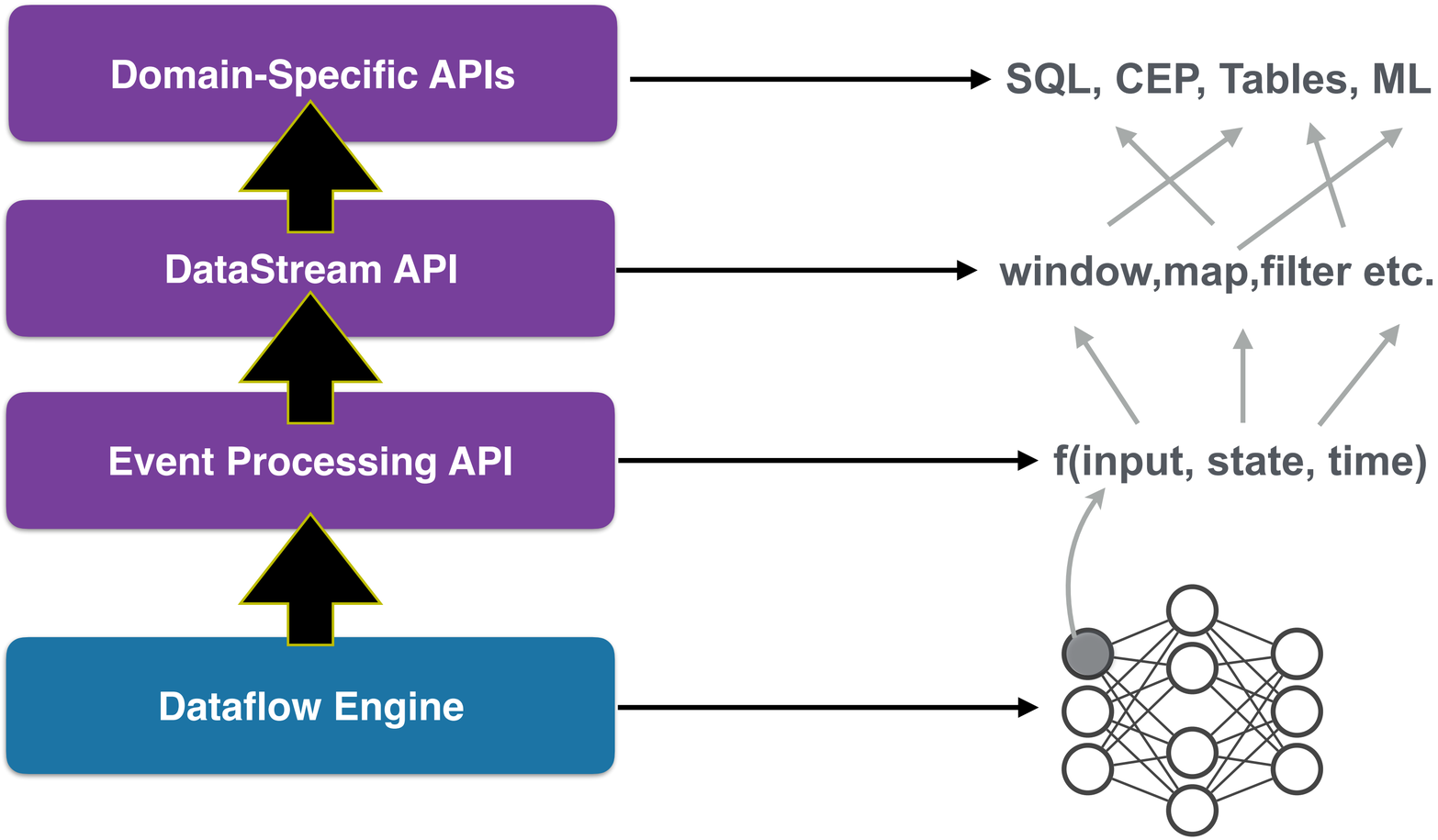
Automates

- Higher-Order Streaming Functions
- Event Windowing (sessions, time etc.)

- Dynamic program state
- Operations on out-of-order streams

- Fault Tolerance
- Scalability
- Monitoring/IO Management

Program Hierarchy in Flink



Automates

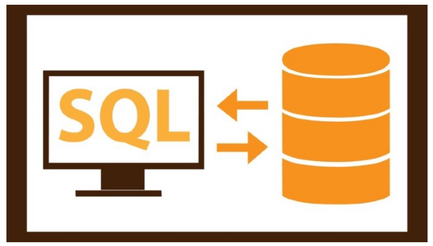
- Fully Declarative Programming
- Event Patterns, Relations etc.

- Higher-Order Streaming Functions
- Event Windowing (sessions, time etc.)

- Dynamic program state
- Operations on out-of-order streams

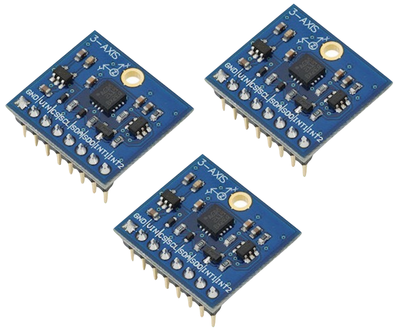
- Fault Tolerance
- Scalability
- Monitoring/IO Management

Declarative Streaming Examples



```
SELECT
  HOUR(r.rideTime) AS hourOfDay,
  AVG(f.tip) AS avgTip
FROM
  Rides r,
  Fares f
WHERE
  r.rideId = f.rideId AND
  NOT r.isStart AND
  f.payTime BETWEEN r.rideTime - INTERVAL '5' MINUTE AND r.rideTime
GROUP BY
  HOUR(r.rideTime);
```

Average Tip per Hour with Stream SQL



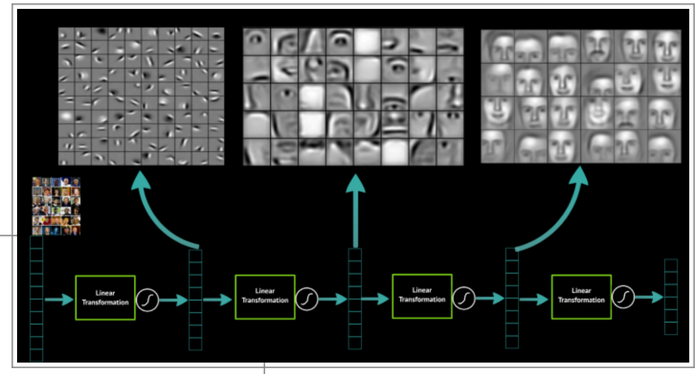
```
val completedRides = Pattern
 .begin[TaxiRide]("start").where(_.isStart)
 .next(“end").where(!_.isStart)

CEP.pattern[TaxiRide](allRides,
completedRides.within(Time.minutes(120)))
```

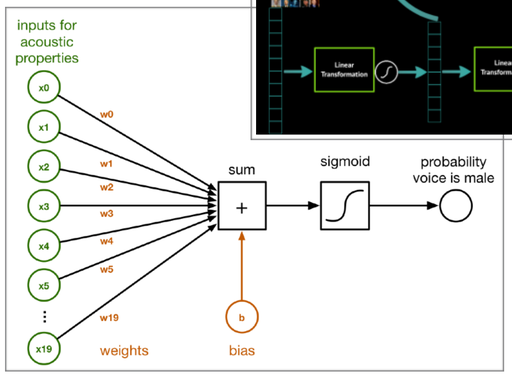
Completed Taxi Rides within 120min with Complex Event Processing

Data Pipelines Today

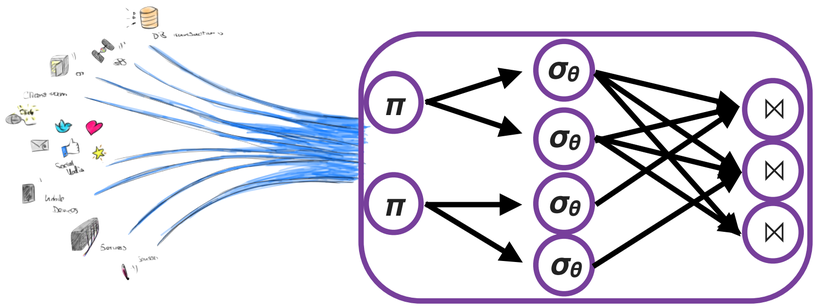
- Many Frameworks/Frontends for different needs
- (ML Training & Serving, SQL, Streams, Tensors, Graphs)



Feature Learning



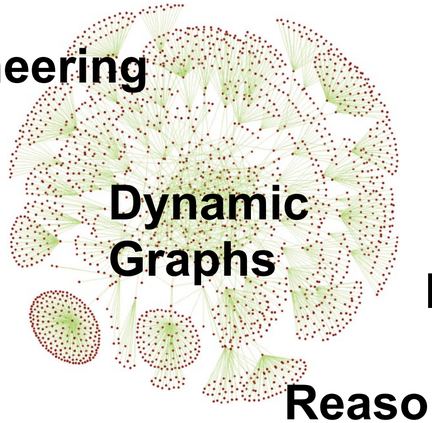
Tensor Programming



Streams

AI

Feature Engineering



ML

Model Serving

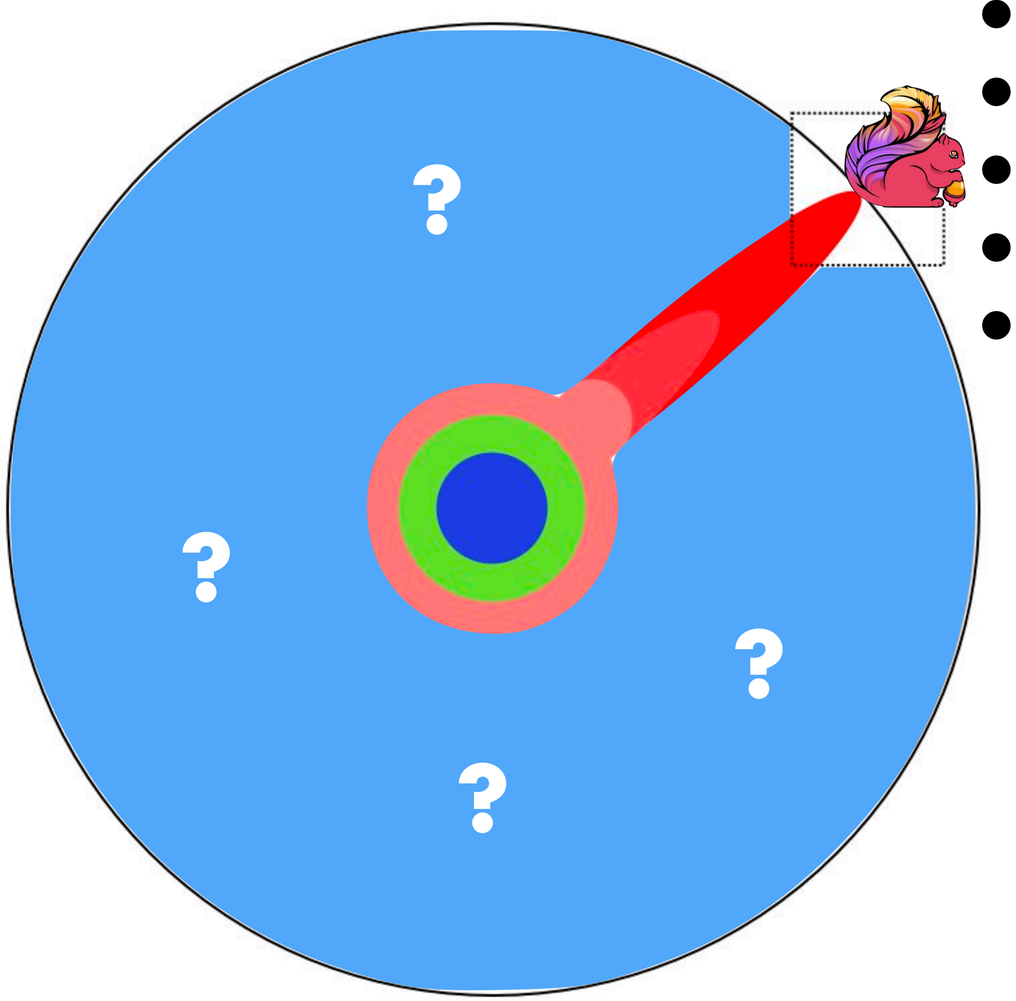
Reasoning

RL

Simulation tasks

The Bigger Picture

Data Processing



Data Streams

- scalable, fault tolerant analytics
- event-based business logic
- out-of-order computation
- dynamic relational tables (SQL)
- event pattern-matching (CEP)

but what about deeper analytics…

- tensors
- graph algorithms
- deep learning
- feature learning
- reinforcement learning
- ….

Critical Limitations

Complex Analytics

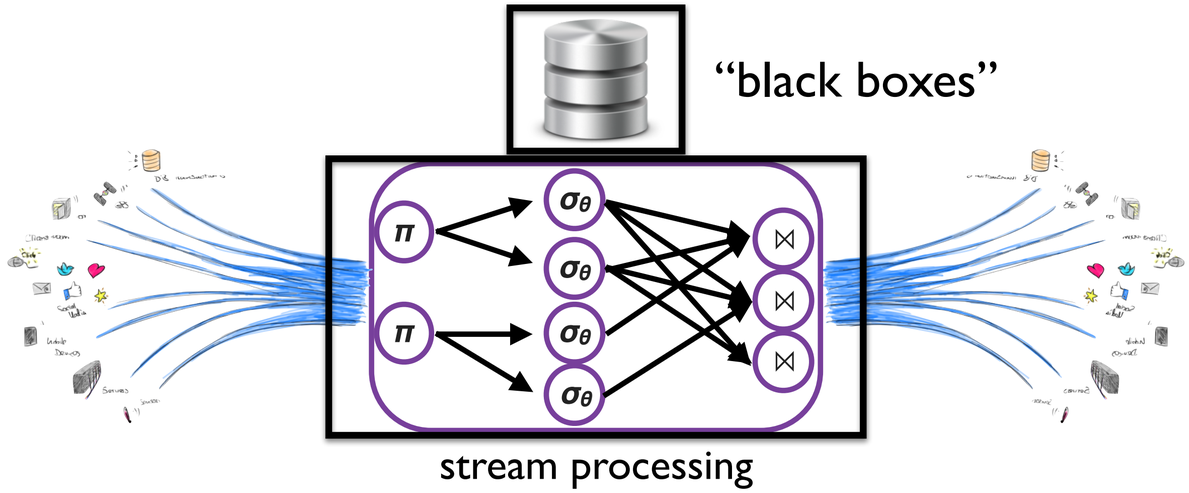
- Highly Skewed Data-Parallel KV Ops
- Beyond Data-Parallel KV ops
  - Graph Data
  - Large Matrices
- Task-Parallel Computation
- Bulk-Synchronous/Window Iterations

State Reuse

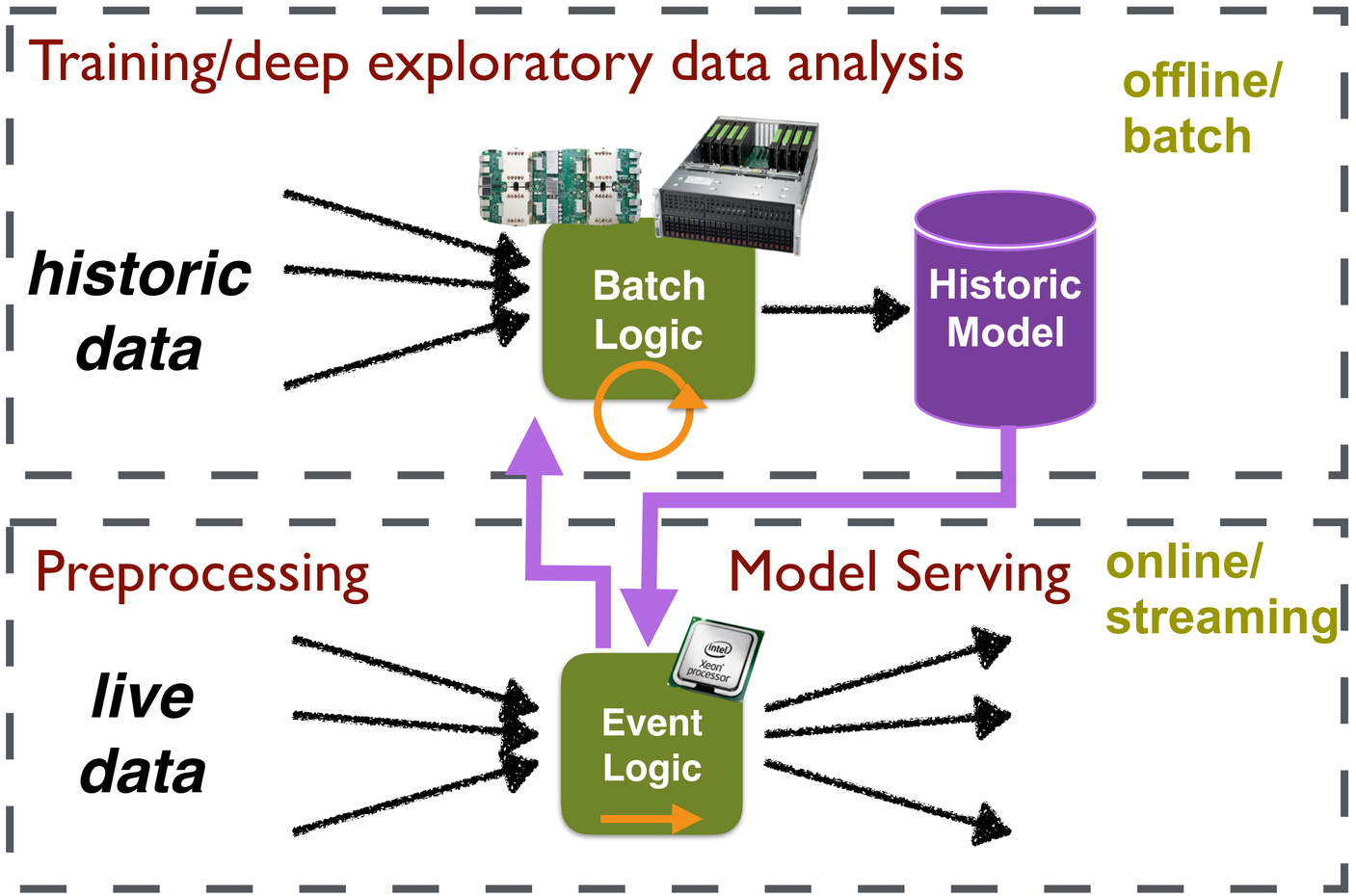
- Accessing Stream State *Externally*
  - Materialized Views (Relational)
  - Feature Matrices (ML)
  - Edge Stream State (Graphs)
- Parallelizing Stateful Ops *Internally*
  - Global Window Aggregation

Compute/HW Sharing

- Escaping the JVM
  - Code Generation for Accelerators
  - SIMD, Vectorization
  - Better Fusion Capabilities
  - Flexible, Dynamic Reconfiguration



Recurring Issues



Framework/Library Silos

↓

Fragmented Codebases/Runtimes

↓

Unshared Hardware

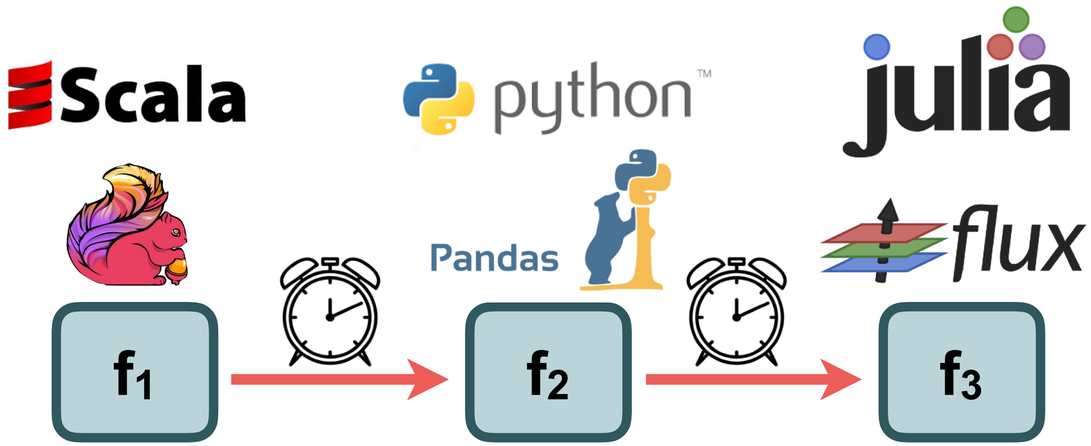
↓

Over-materialization of results

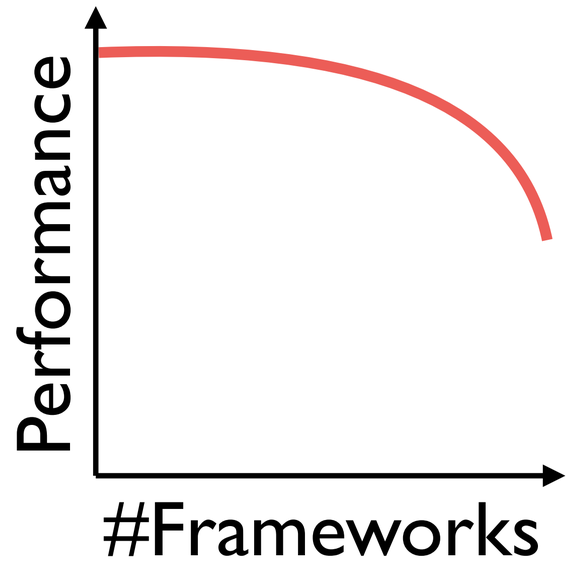
↓

Ridiculously Unoptimal Pipelines

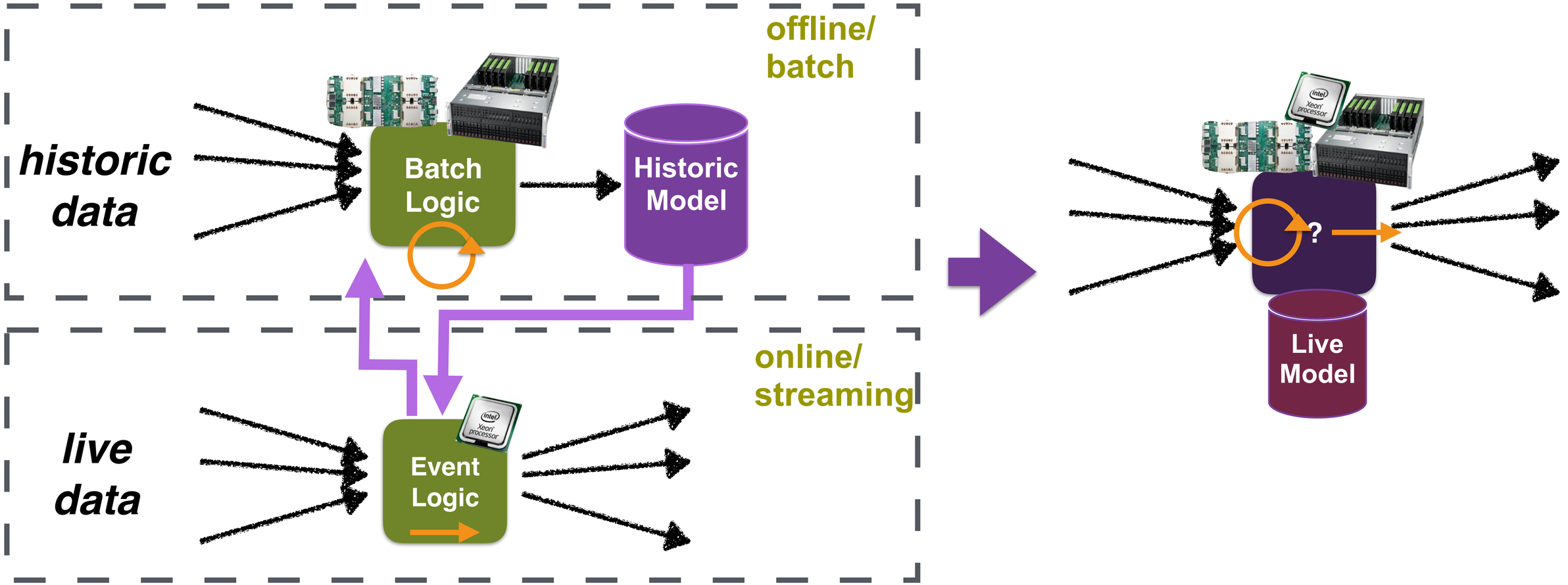
The Problem



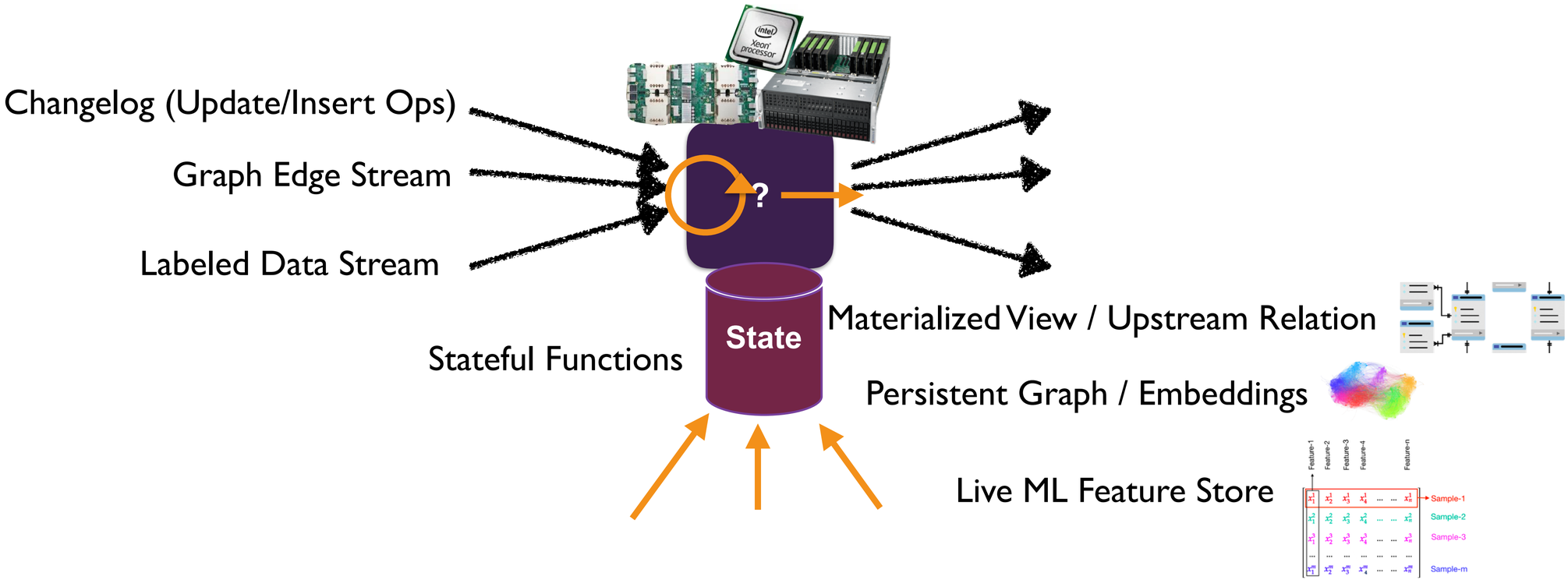
- No cross-optimisation is possible, e.g. resource sharing
- Data movement costs (⟶)



Revisiting Compute Systems



Example Applications



The Arcon Vision

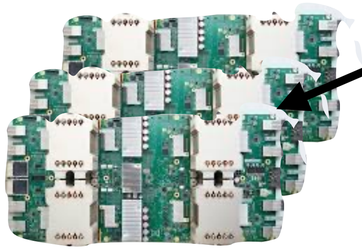
Unified Declarative Programming

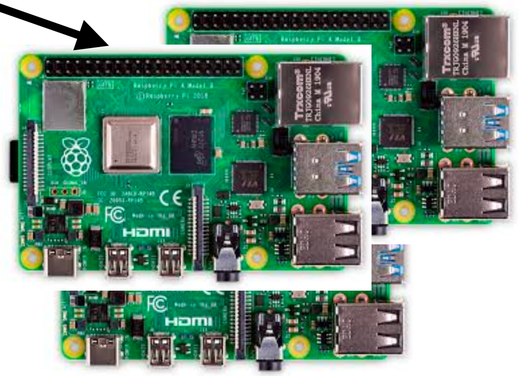
Tensors DataFrames DataStreams Graphs

Cross-Compile
Optimise²
and Generate Code

Arcon

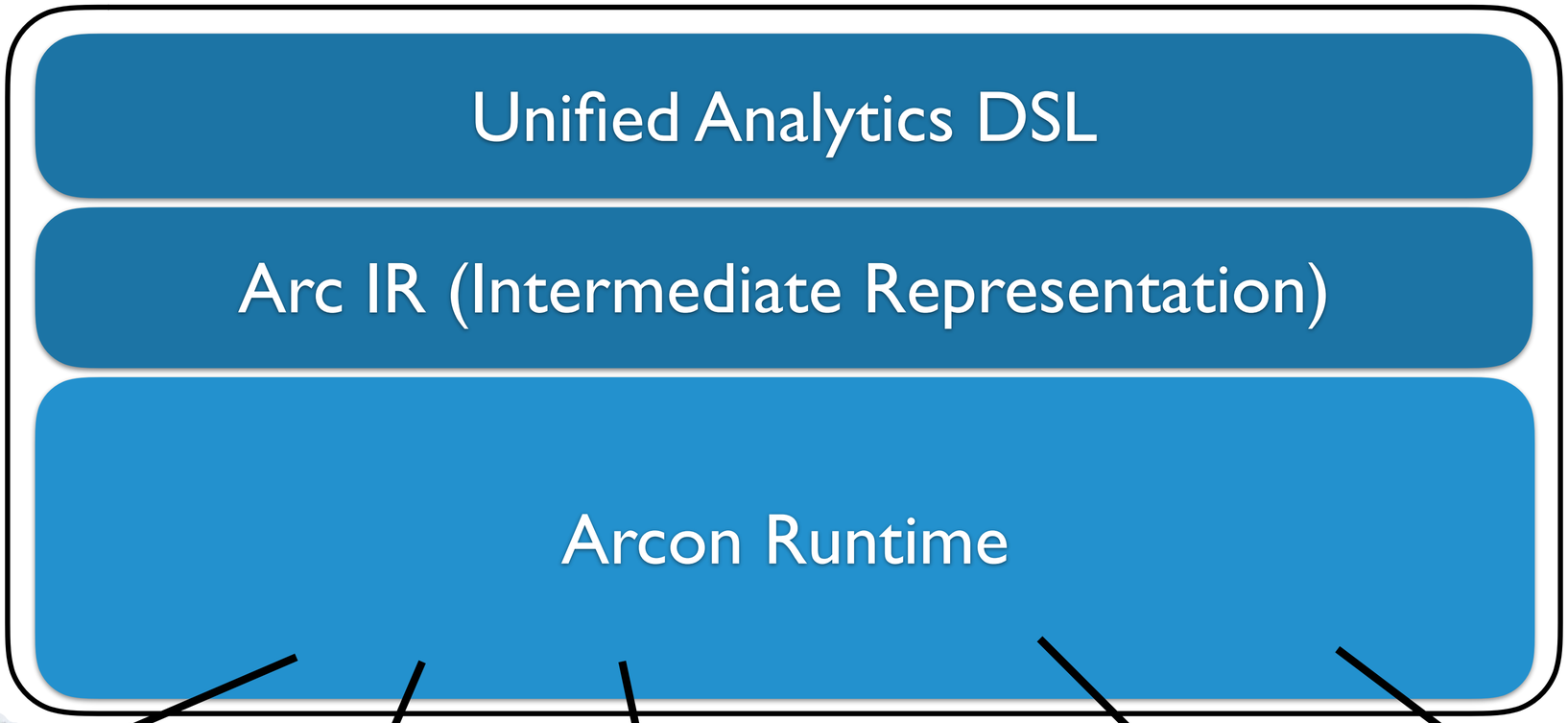
3.5mil eur
SSF funding

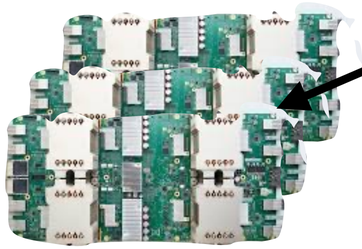


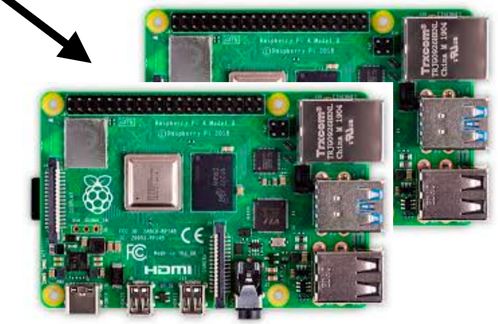


Shared Native Execution

The Arcon Architecture

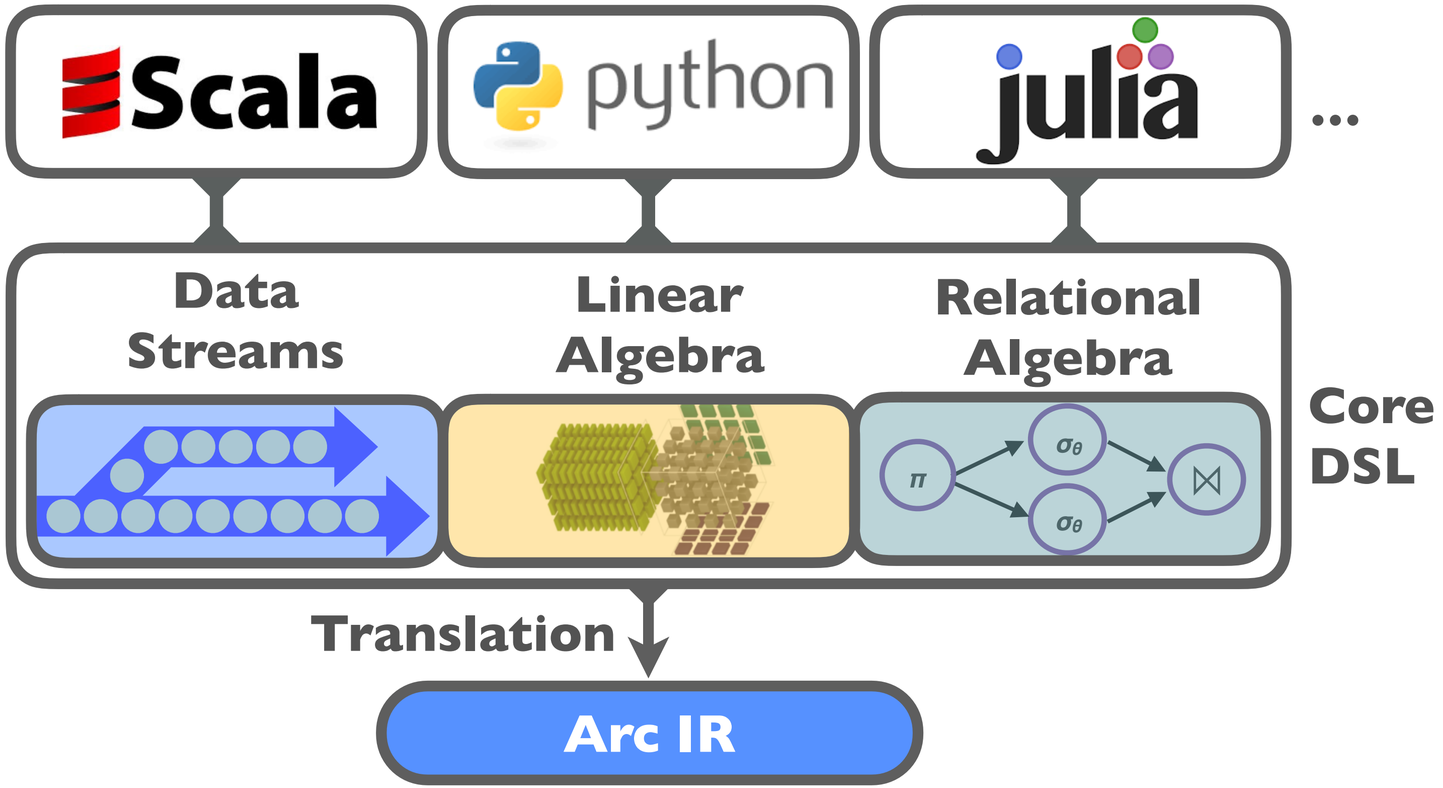






Unified Analytics DSL

- Host language-agnostic core
- Compositional
- First-class citizen support for:
  - *streams, tensors, relations*



DSL Philosophy

Comprehensions (e.g., DataSets + Matrices)

```
1  @lib def vectorizeComment(c: Comment) = { /* UDF */ }
2  @lib def vectorizeUser(u: User) = { /* UDF */ }                    ❶
3
4  optimize {                                                         ❷
5    // Join "Comments" and "Users" and vectorize the result
6    val features = for {
7      c <- Comments // DataBag[Comment]
8      u <- Users    // DataBag[User]
9      if u.user_id == c.user_id
10   } yield vectorizeComment(c) ++ vectorizeUser(u)
11   // Convert the DataBag "features" into matrix "X"
12   val X = Matrix(features)                                         ❸
13   // Filter rows that have values > 10 in the third column
14   val M = X.forRows(row => row(2) > 10)                            ❹
15   // Calculate the mean for each column
16   val means = M.forCols(col => mean(col))                          ❺
17   // Deviation of each cell of "M" to the cell's column mean
18   val U = M - Matrix.fill(M.nRows, M.nCols)((i,j) => means(j))
19   // Compute the covariance matrix
20   val C = 1 / (U.nRows - 1) * U ** U.t                             ❻
21 }
```

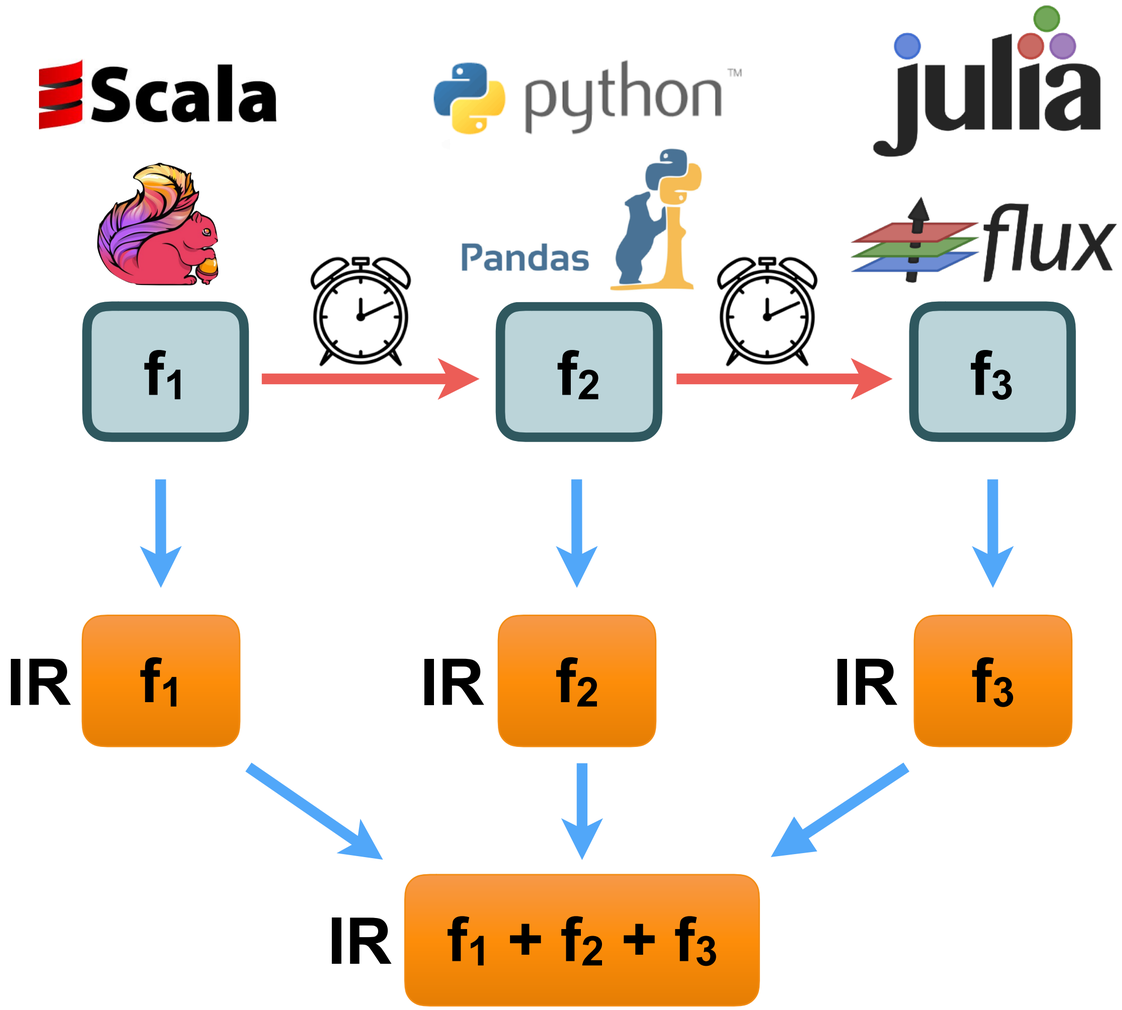
Example taken from Lara DSL

DSL Extensions

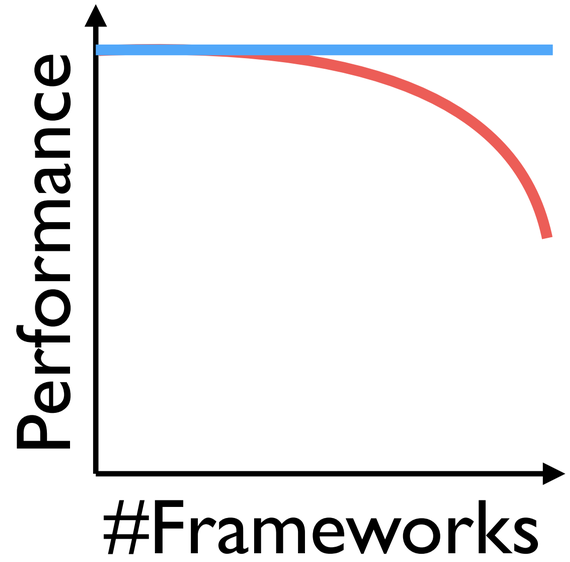
Comprehensions for Data Streams

```
  for {
    w <- features.window(length=10, stride=60) //
Window[Feature] with 10 sec length, 1 min stride
  } yield {
    // Convert the Window "w" into matrix "X"
    val X = Matrix(w)
    // Filter rows that have values > 10 in the third column
    val M = X.forRows(row => row(2) > 10)
    // Calculate the mean for each column
    val means = M.forCols(col => mean(col))
    // Deviation of each cell of "M" to the cell's column mean
    val U = M - Matrix.fill(M.nRows, M.nCols)((i,j) =>
means(j))
    // Compute the covariance matrix
    val C = 1 / (U.nRows - 1) * U ** U.t
    // ...
  }
}
```

IR Intuition



- No cross-optimisation is possible, e.g. resource sharing
- Data movement costs (⟶)



Arcon Compiler Pipeline

Arcon

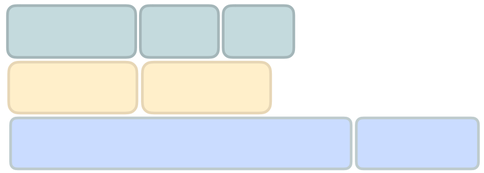
Unified Analytics DSL

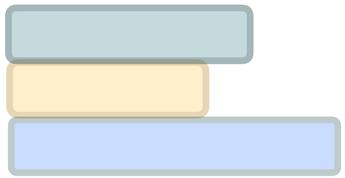
Arc (High Level IR)

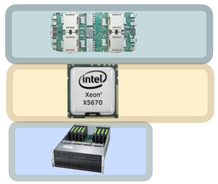
Logical Dataflow IR

Physical Dataflow IR

Binaries







Arc IR

- A minimal yet feature-complete set of read/write-only types and expressions

```
program     ::= { declaration } lambda
declaration ::= macro id ( { id , } ) = expr ;
              | type id = type ; // Type alias
              | fn id | { type , } | ( type ) = lambda ;
lambda      ::= | { id : type , } | expr
type        ::= id | valueType | builderType | struct type
valueType   ::= Unit | bool | i8 | i16 |...
              | Simd [ type ]
              | Vec [ type ]
              | Dict [ type , type ]
              | Stream [ type ]
builderType ::= Appender [ type ]
              | Merger [ type , binop ]
              | StreamAppender [ type ]
              | Windower [ type , type ]
              |...
struct type ::= { { type , } }
expr        ::= opExpr | letExpr
opExpr      ::= ( expr )
              | id
              | literal
              | type ( expr ) // Type cast
              | for ( iterator , expr , lambda )
              | merge ( expr , expr )
              | result ( expr )
              | if ( expr , expr , expr )
              | cudf [ id , type ] ( { expr , } )
              | drain ( expr , expr )
              | builderConstr
              | opExpr binop opExpr
              |...
```

```
letExpr     ::= let id : type = opExpr ; expr
binop       ::= + | - | * | / |...
              | id
literal     ::= scalarLiteral
              | [ { expr , } ] // Vec literal
              | { { expr , } } // Struct literal
              | () // Unit literal
iterator    ::= expr | iter ( expr , expr , expr )
              | next ( expr )
              | keyby ( expr , lambda )
              |...
builderConstr ::= Appender [ type ]
              | Merger [ type , binop ]
              | StreamAppender [ type ]
              | Windower [ type , type ] ( lambda , lambda,
                  lambda )
              |...
```

Read More

[Paper] Arc: An IR for Batch and Stream Programming @ DBPL19

[Code] https://github.com/cda-group/arc

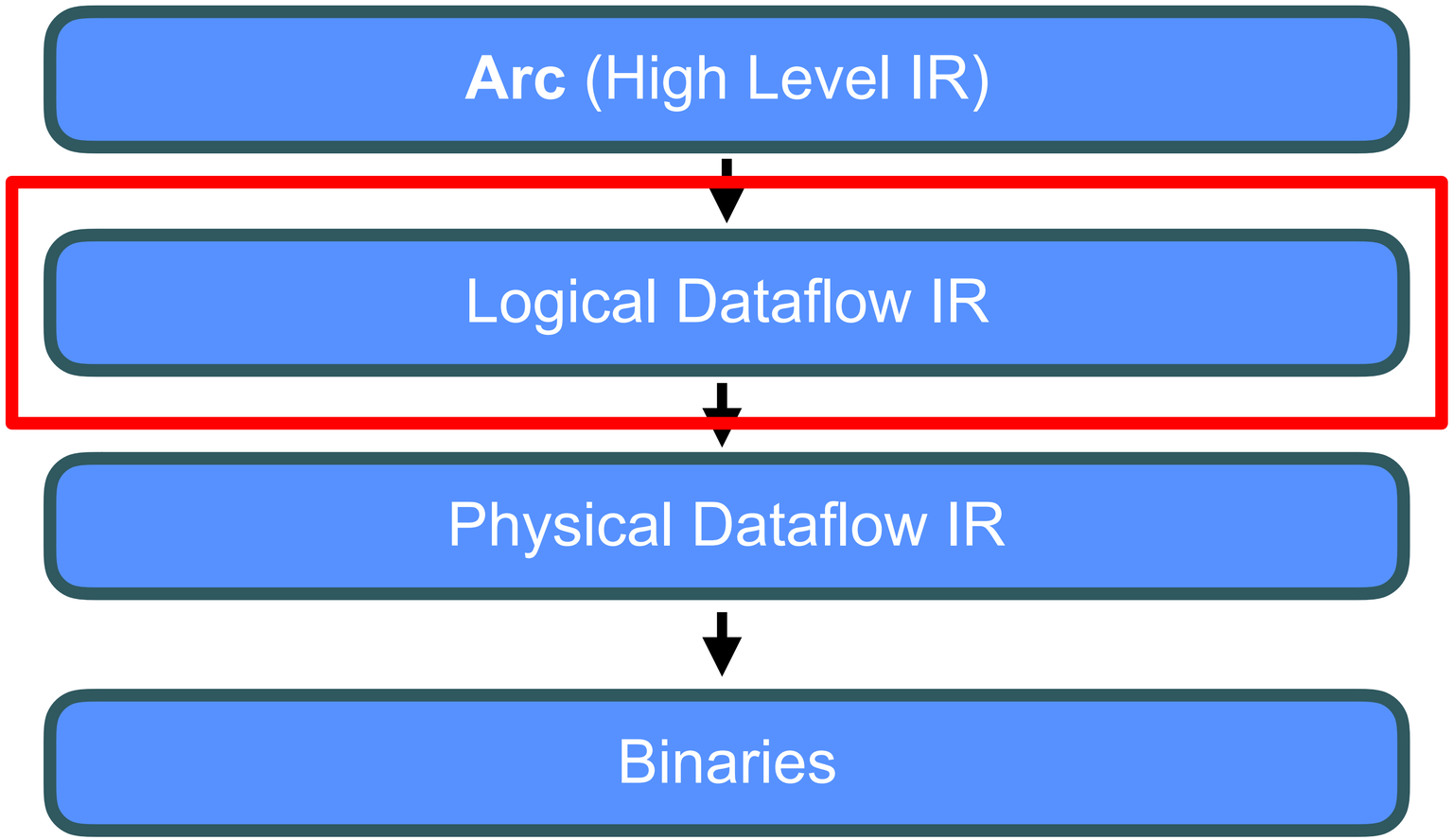
RI.
SE

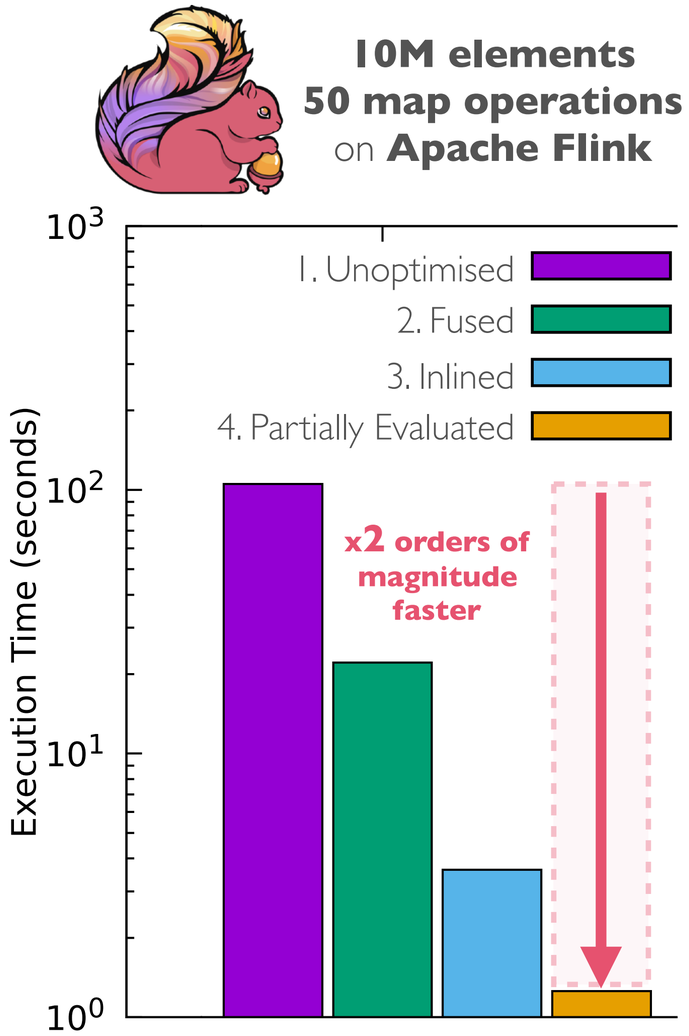
Arc Optimisations

- Arc supports **both** compiler and dataflow optimisations. (Compiler toolchain in MLIR)
  - **Compiler**: Loop unrolling, partial evaluation,
  - **Dataflow**: Operator fusion, fi ssion, reordering, specialisation, ...

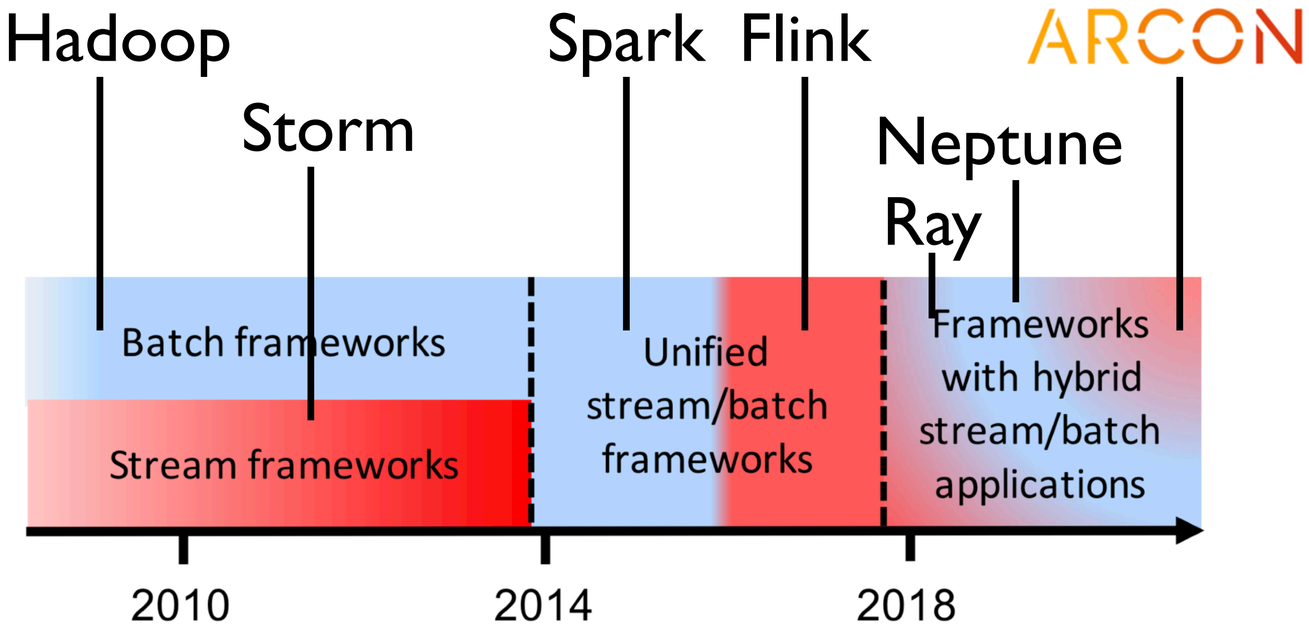
Unlocking Speed

Arc can boost even existing frameworks





Next-Gen is Hybrid

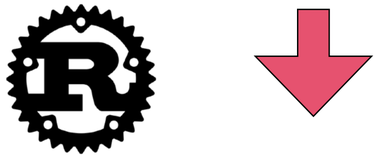


Neptune: Scheduling Suspendable Tasks for Unified Stream/Batch Applications SOCC 2019

Garefalakis, Karanasos, Pietzuch

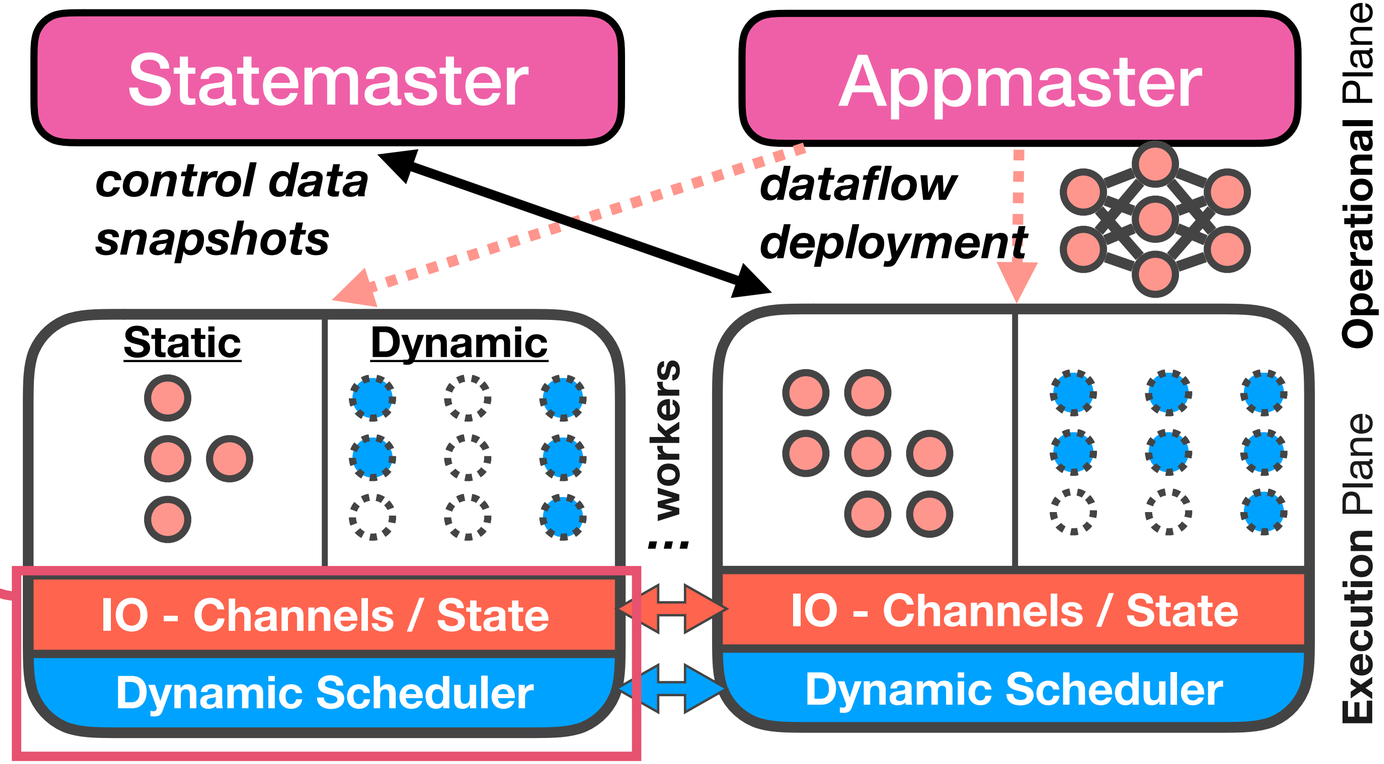
Arcon : A Runtime Capable for Unified Analytics

- performance critical libraries
- hardware native optimisations needed

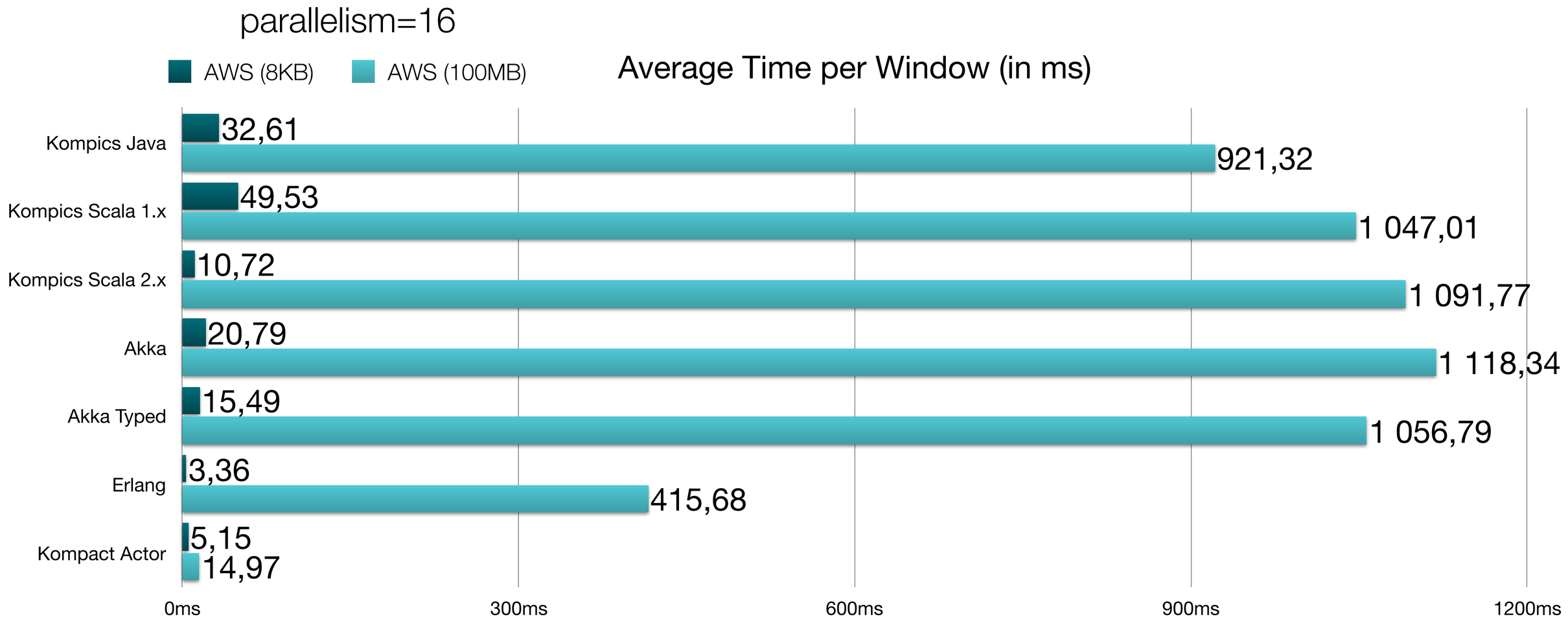


Kompact Middleware

- Message Driven Actor Framework
- Written in Rust (memory safety)
- Complex Type Checking
- High Performance

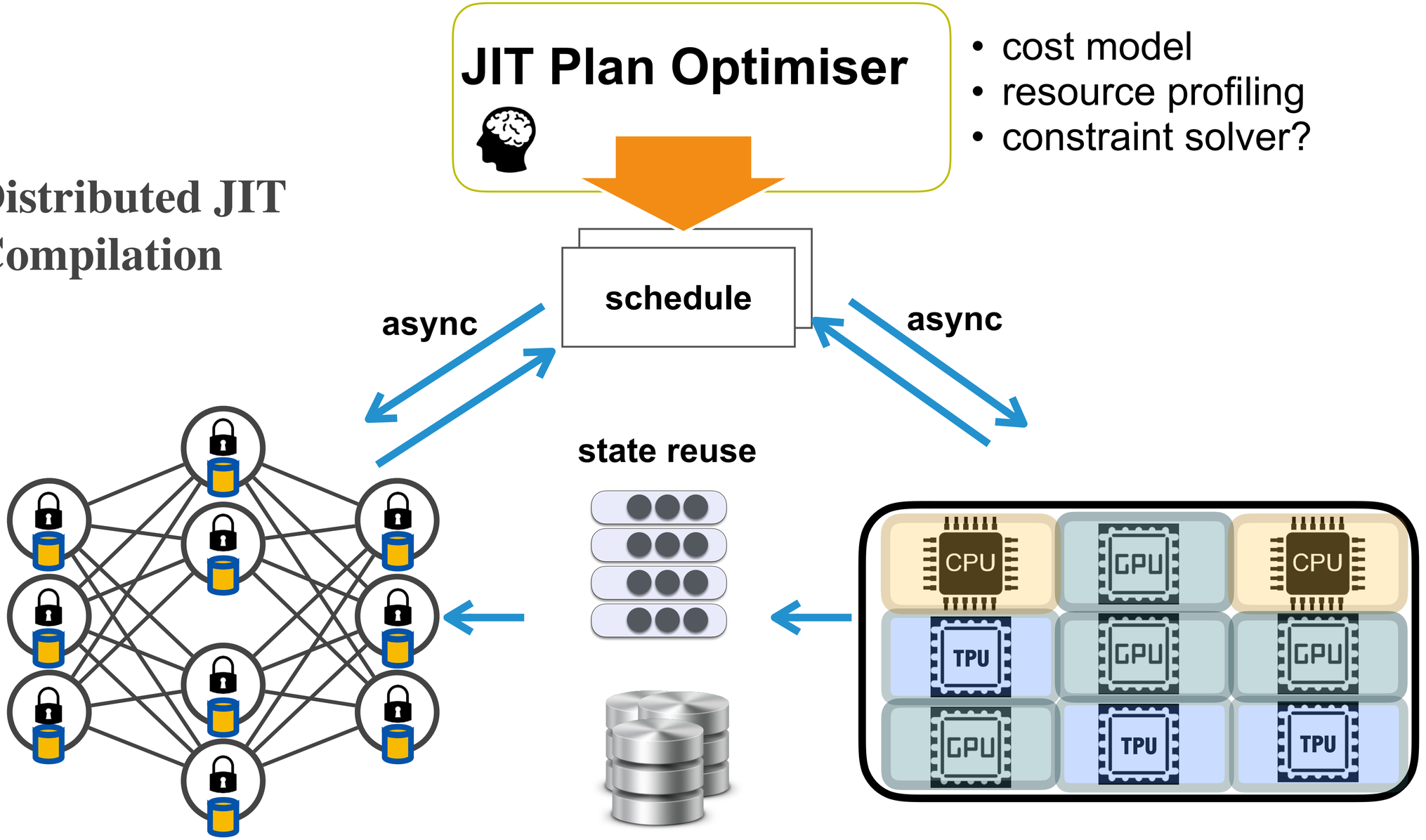


Streaming Windows Results



RI.
SE

Distributed JIT Compilation



- cost model
- resource profiling
- constraint solver?

The Team

Lars Kroll

Paris Carbone

Frej Drejhammar

Christian Schulte

Henrik Boström

Seif Haridi

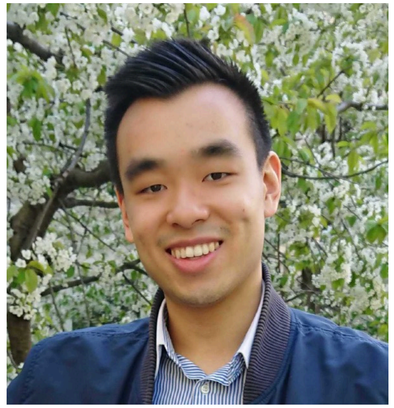
Max Meldrum

Klas Segeljakt

Adam Hasselberg

Khoa Dinh

Harald Ng



Mikolaj Robakowski

More Info

Code: https://github.com/cda-group/arc

https://github.com/cda-group/arcon

Project: https://cda-group.github.io