ROYAL INSTITUTE
OF TECHNOLOGY

Arcon
Continuous and Deep Data Stream Analytics

Max Meldrum, Klas Segeljakt, Lars Kroll
Paris Carbone, Christian Schulte, Seif Haridi

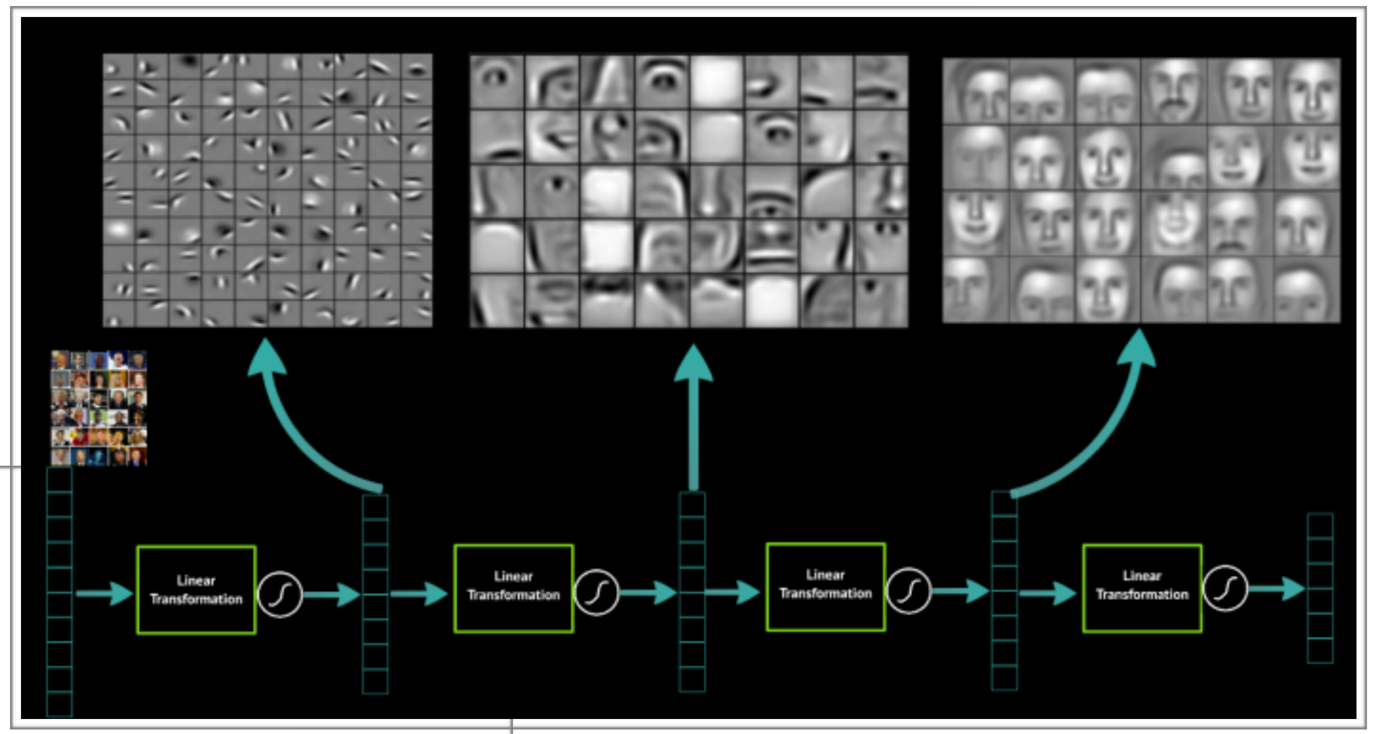
ROYAL INSTITUTE OF TECHNOLOGY

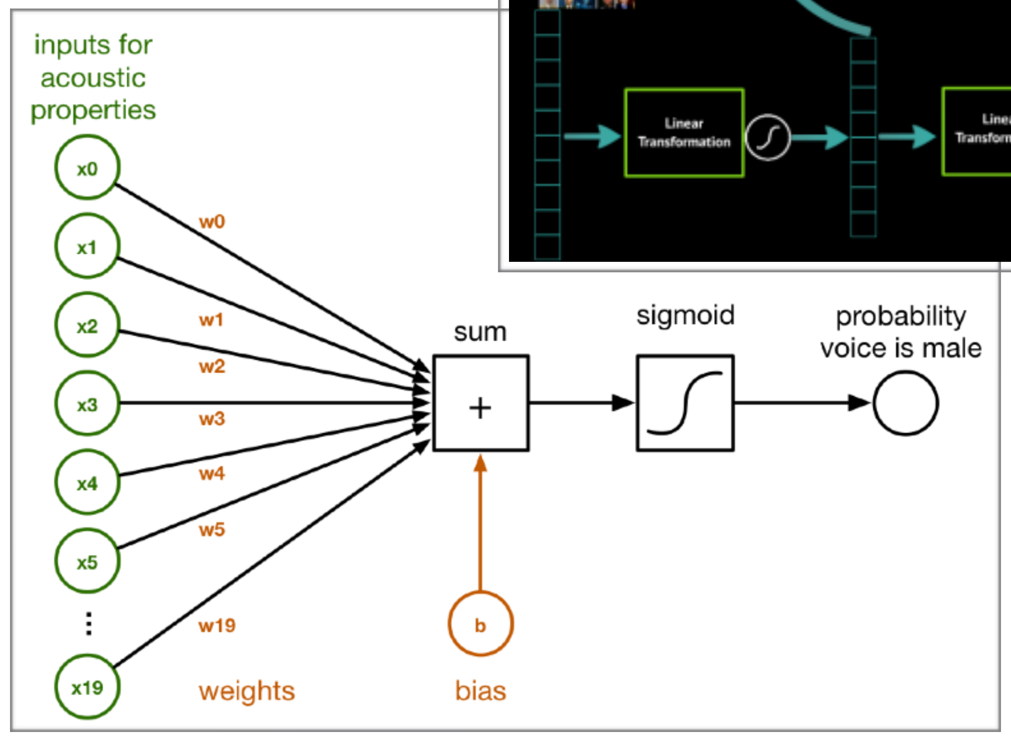
Outline

- Project Introduction
- Arc IR and Compilation Pipeline
- Demo (Frontend, IR, CodeGen, Execution)
- Conclusions and Future Work

Motivation

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





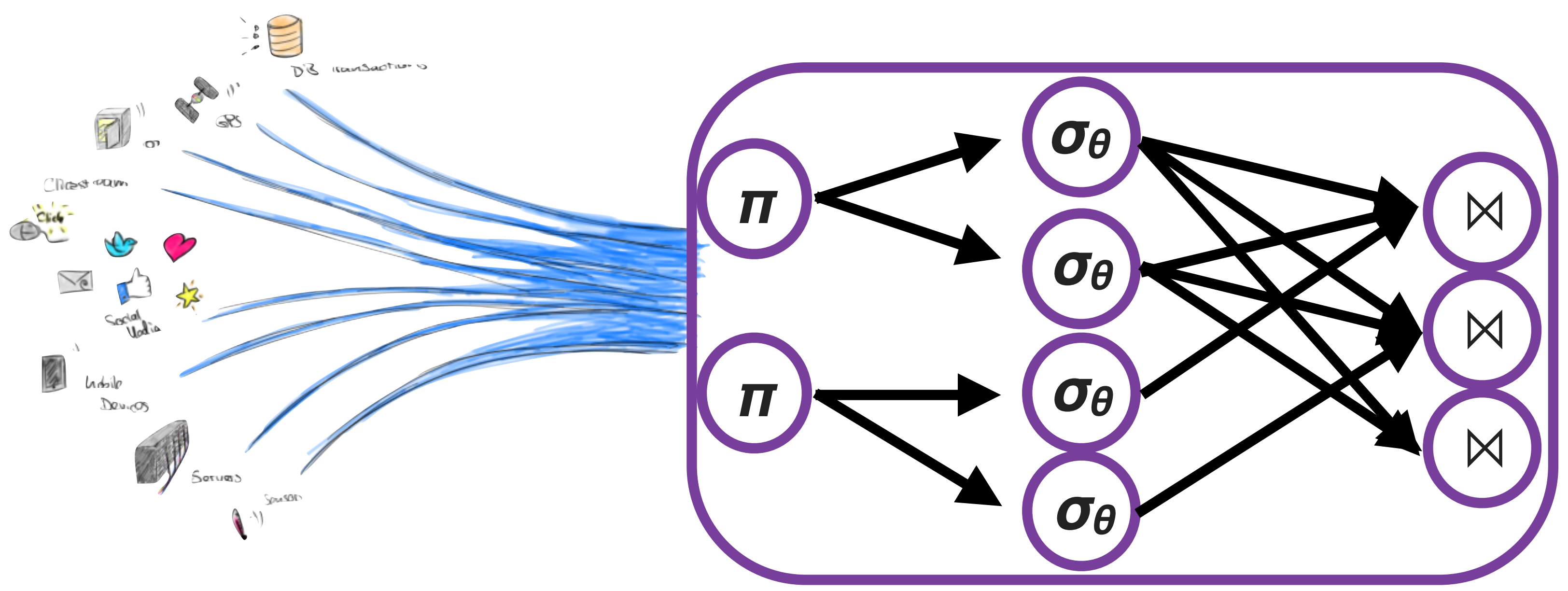
Feature

AI

Tensor

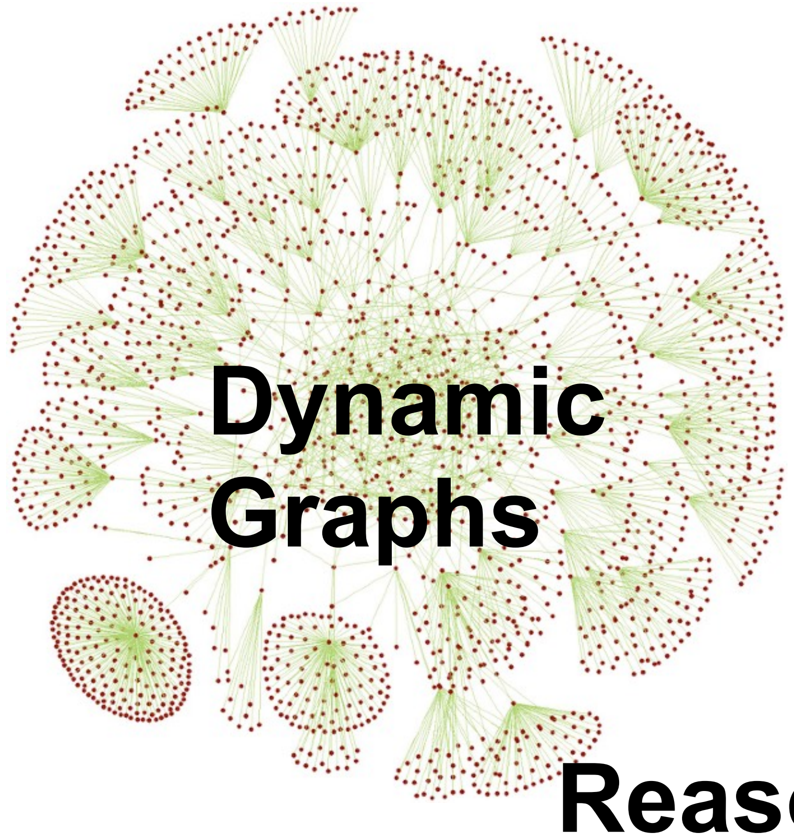
RL

Simulation



Relational Data

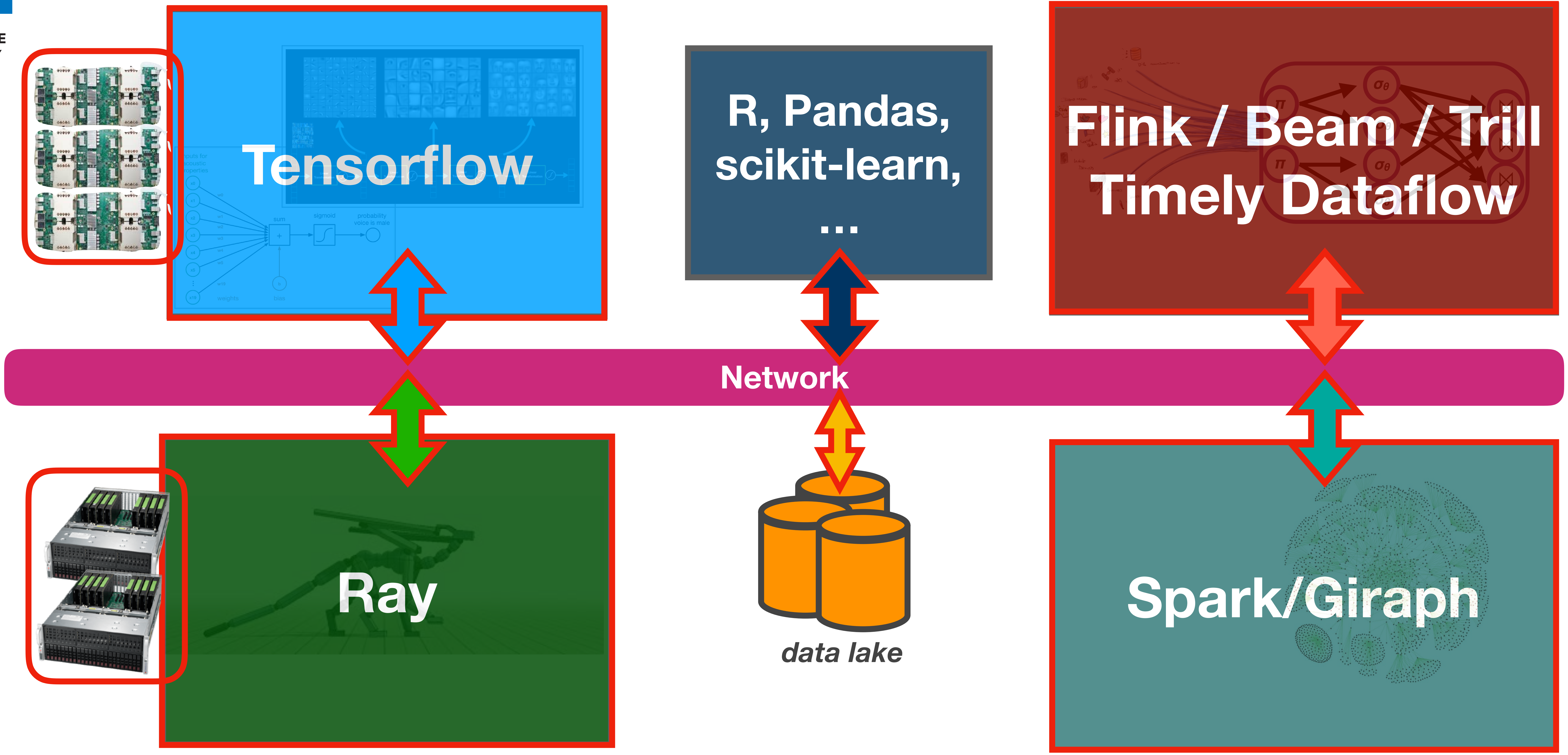
Streams



ML

Reasoning

Silos in Data Pipelines



- Impedance Mismatch *(e.g., types, guarantees, state etc.)*
- Excessive IO/ Data Movement of intermediate results
- Isolated HW Execution - <u>No cross-framework optimisation</u>

The Arcon System

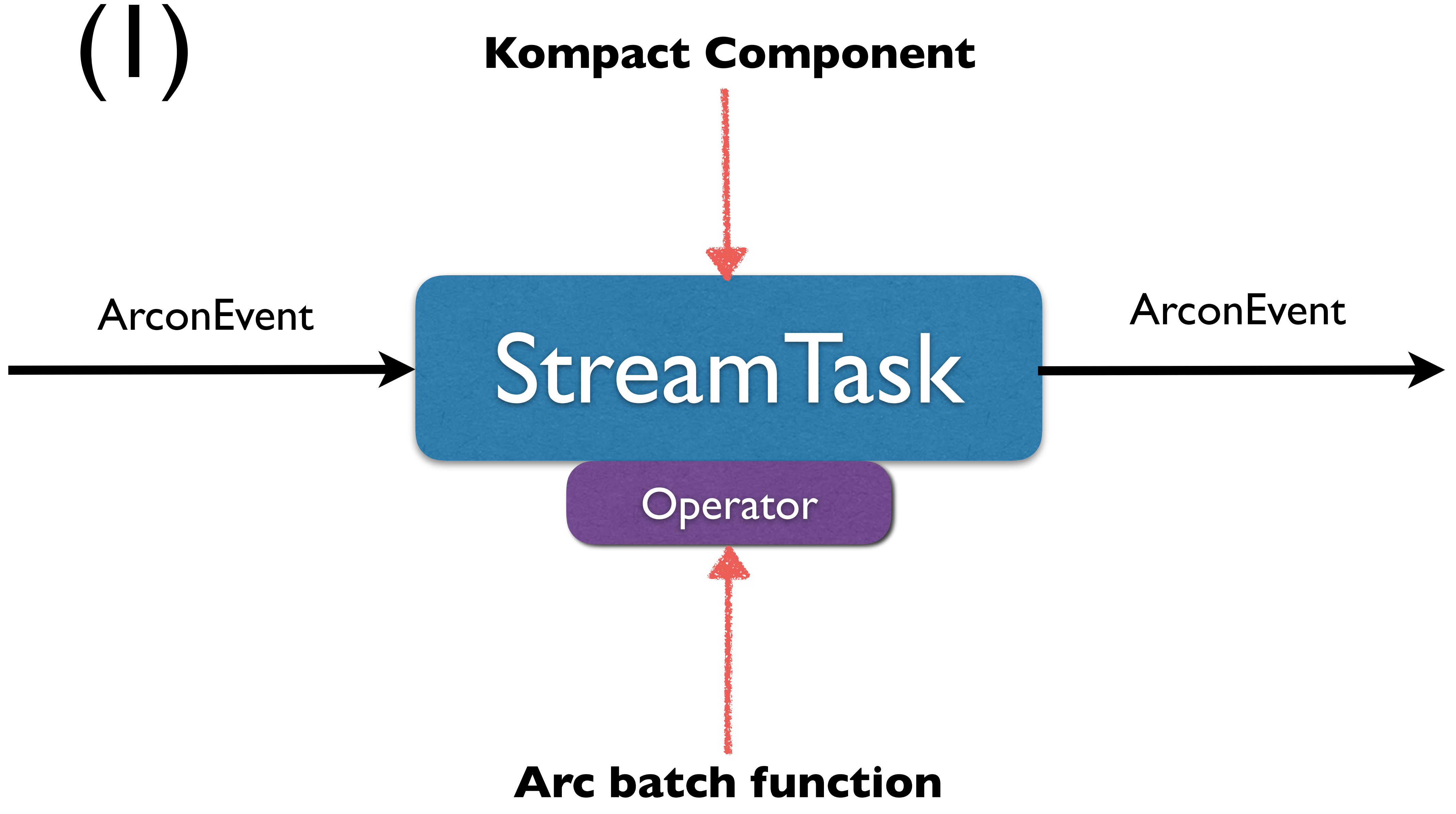
Arc IR Compiler

Arcon Runtime

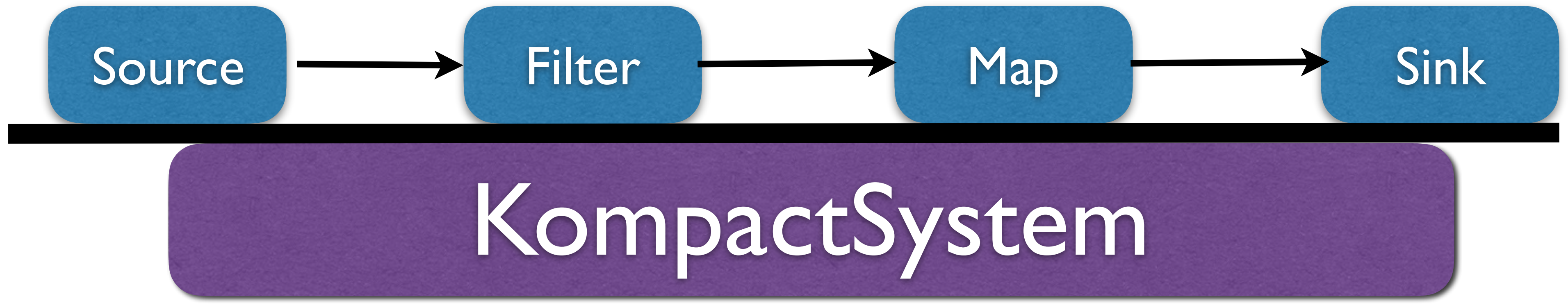
Arcon Runtime

- Rust-based distributed dataflow engine
- Building Blocks:
  - Kompact: Hybrid Concurrent Component + Actor Model
  - Arc batch backend

(1)

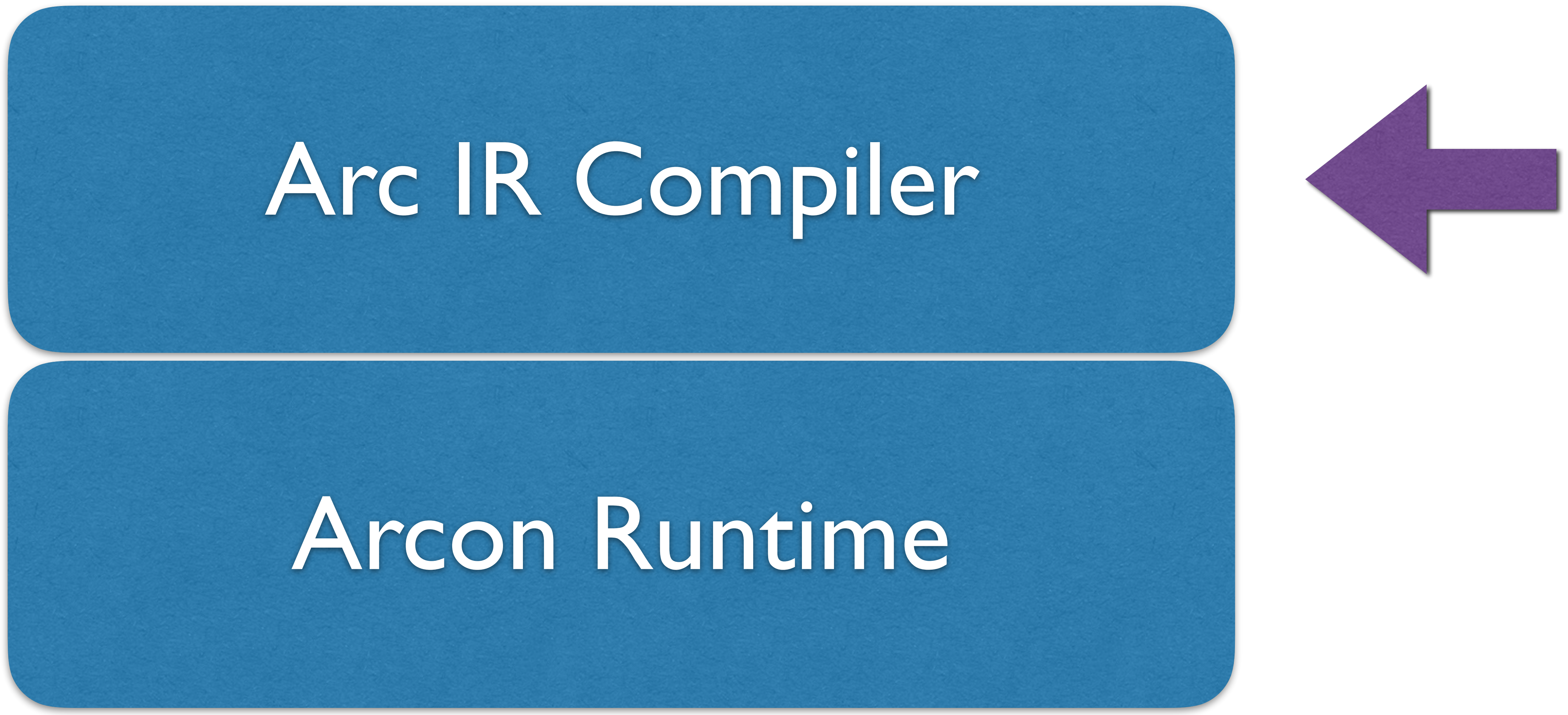


(2)

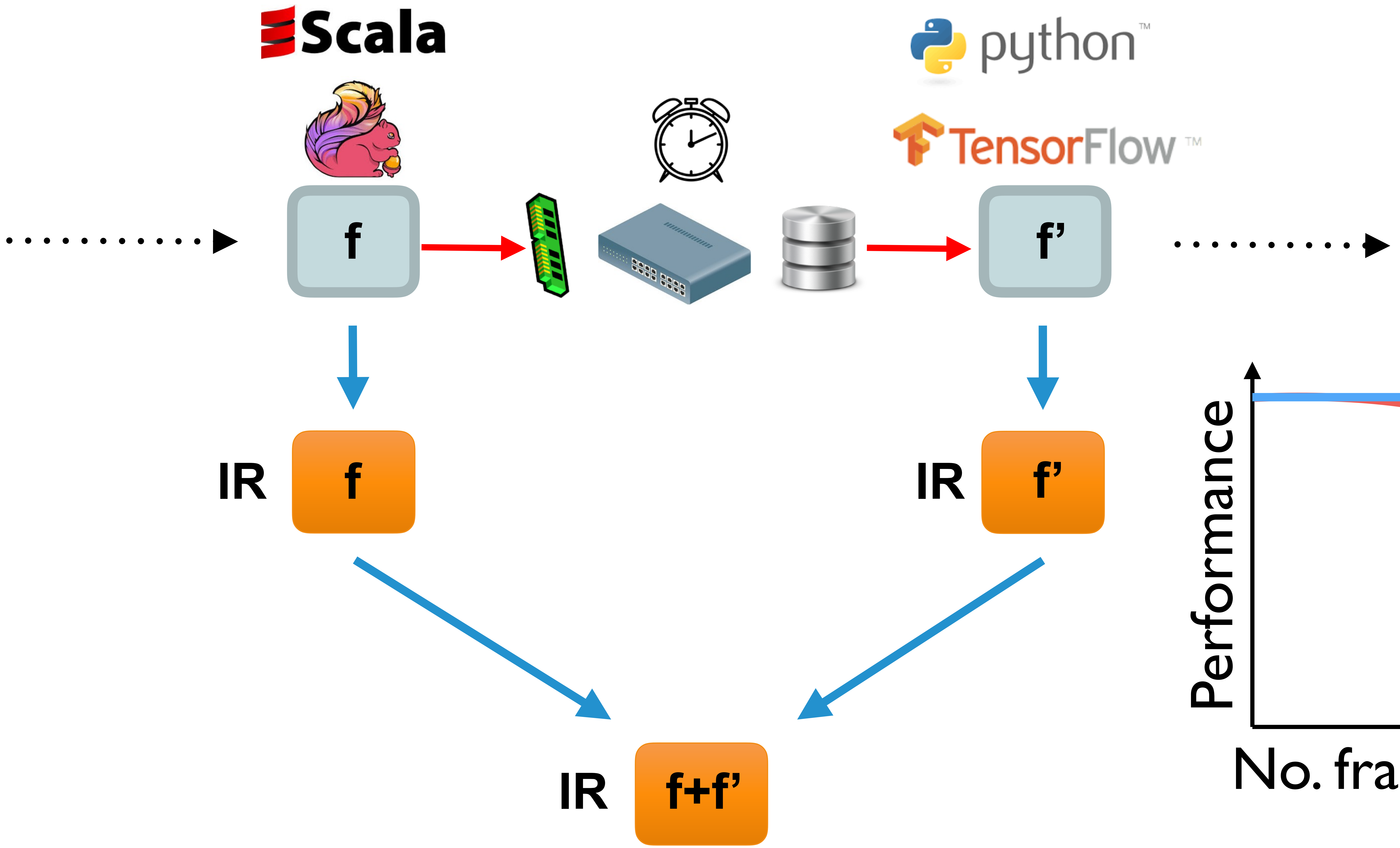


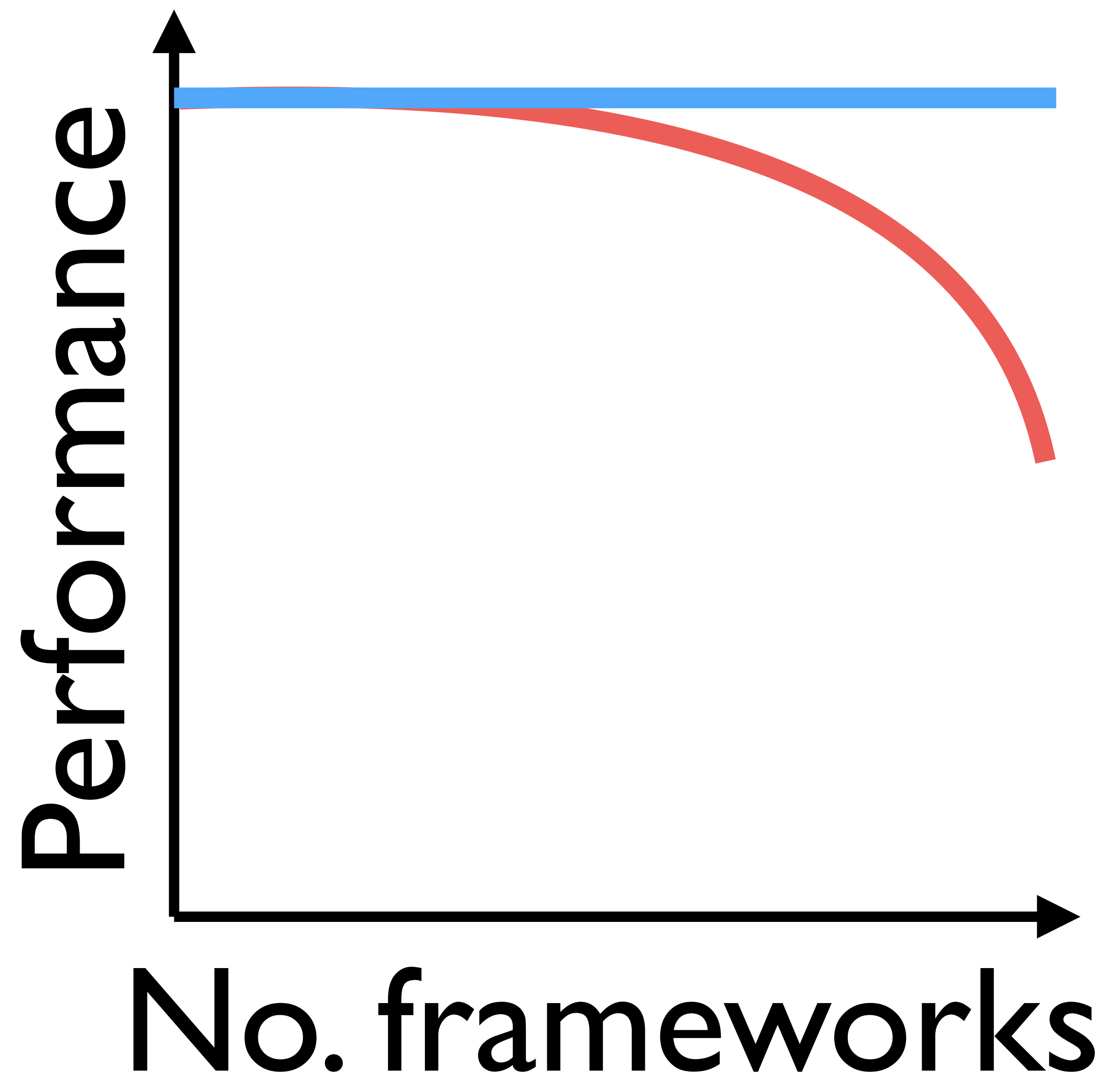
Network, Timers, and Scheduling.

The Arcon System



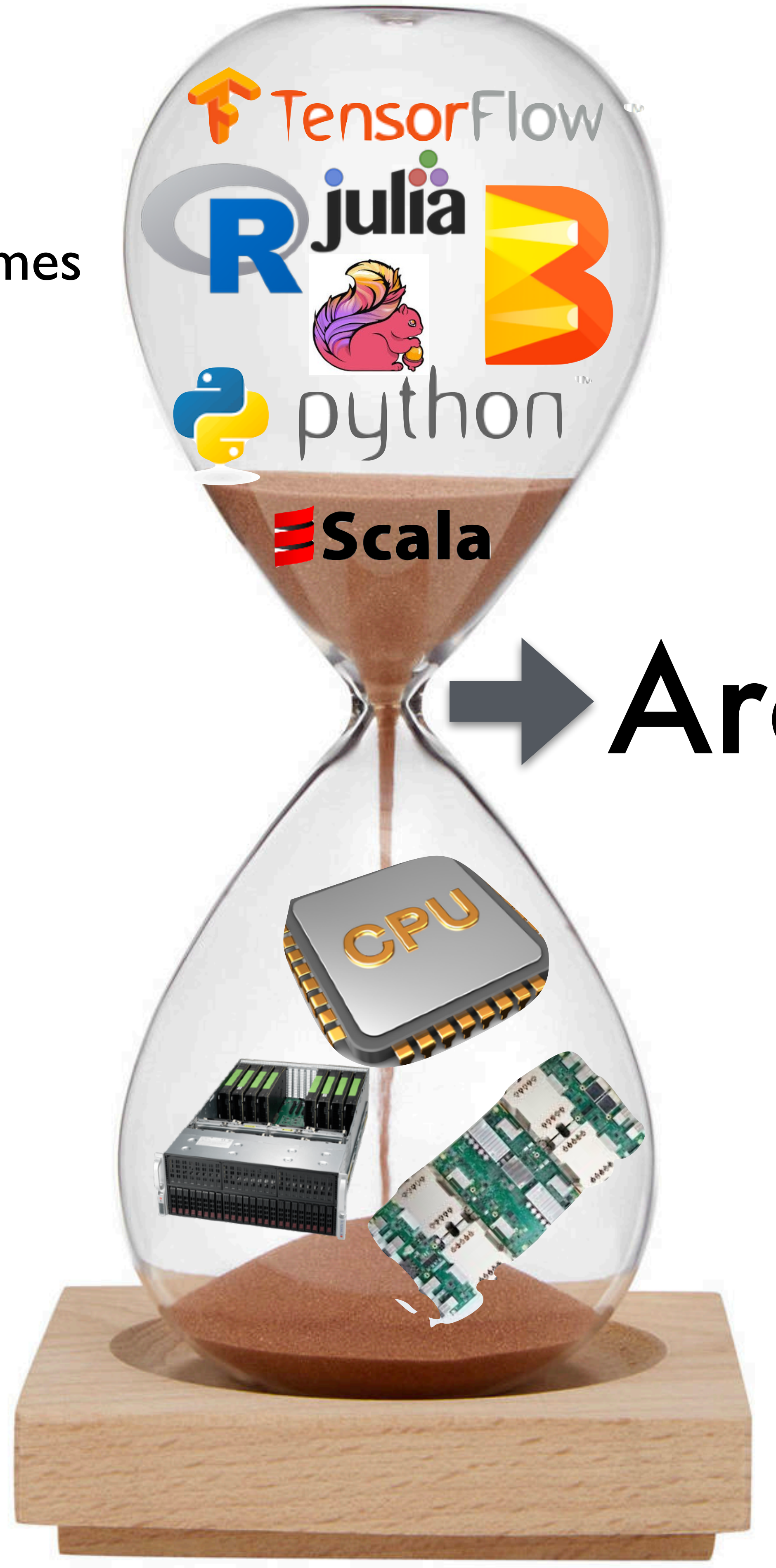
Intuition


Scala
python
TensorFlow
f
f’
IR f
IR f’
IR f+f’


Performance
No. frameworks

Arc IR

- Streams
- Tables/Data Frames
- Vectors
- Tensors
- ….



Arc:

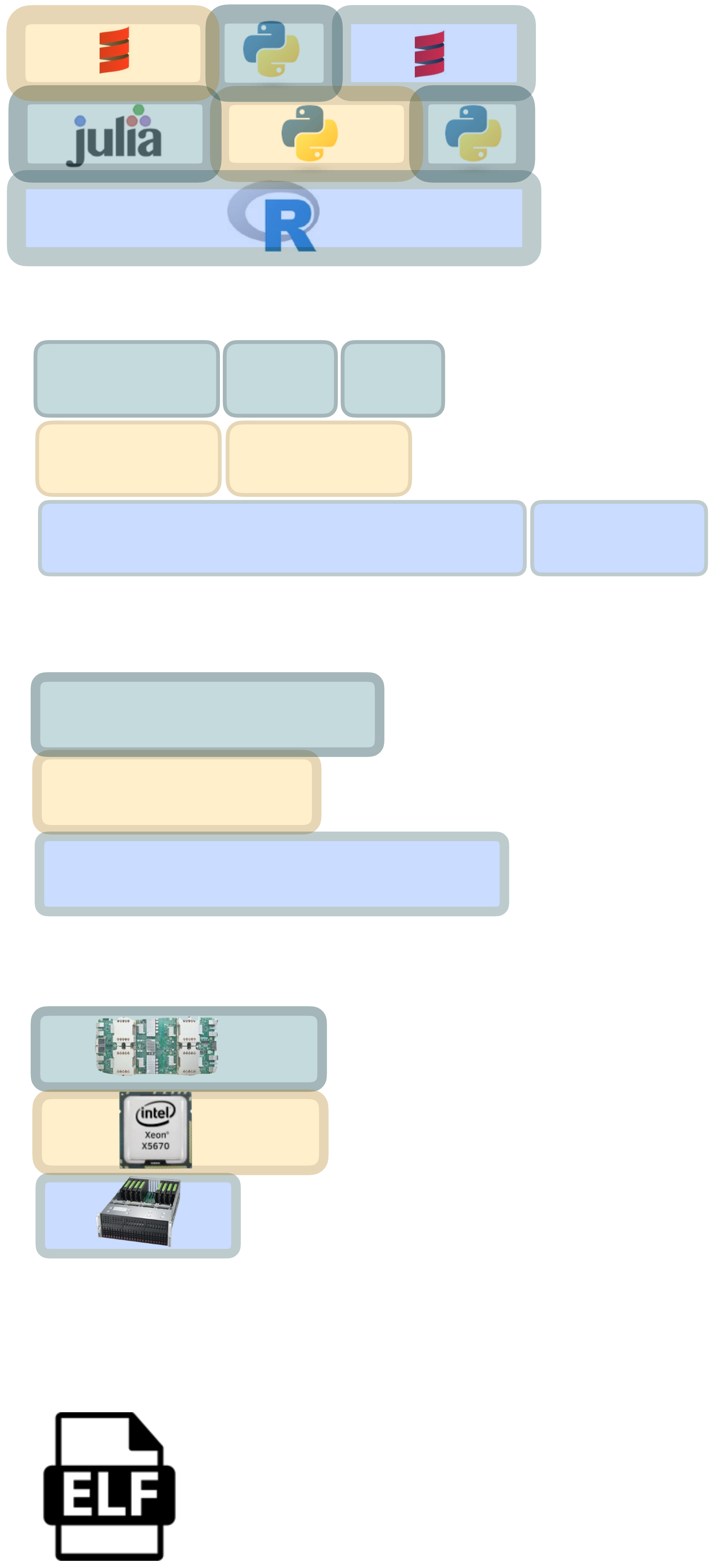
- **Support both batch and streaming abstractions**
- **Sources/Sinks/Operators**
- **User-defined Windows**
- **Out of Order Processing**

- Multicore CPUs
- GPUs
- TPUs
- FPGAs
- …..

Arcon Compiler Pipeline

Arcon

- Frontends
- **Arc** (High Level IR)
- Logical Dataflow IR
- Physical Dataflow IR
- Binaries



How does Arc work?

The **Weld IR*** is a subset of **Arc that supports batch computations**

- A restrictive language for describing data transformations
- **Values**: Read-only data types (e.g., `vec[T], i8..i64, bool, ...`)
- **Builders**: Write-only data types (e.g., `appender[T]`)
- Calling `result` on a **builder** returns the associated **value** type

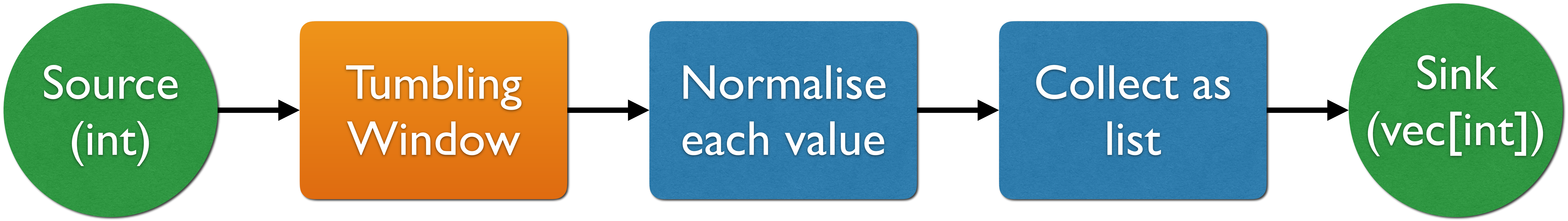
The **Arc IR**** supports both stream and batch computations

- Stream **sources** are read-only => **values** (i.e. `stream[T]`)
- Stream **sinks** are write-only => **builders** (i.e. `streamappender[T]`)
- Calling `result` on a **sink** returns a **source** and creates a **channel** between them

*Palkar, Shoumik, et al. "Weld: A common runtime for high performance data analytics." *Conference on Innovative Data Systems Research (CIDR)*. 2017.

**Kroll, Lars, et al. "Arc: an IR for batch and stream programming". In *Proceedings of the 17th ACM SIGPLAN International Symposium on Database Programming Languages*

Example: Normalisation



Normalise by dividing each element by the average

e.g. for window `[4, 2, 2, 8]`

... the average is `4`
... the output is `[1, 0.5, 0.5, 2]`

Frontend code

```
import arc_beam as beam
import arc_beam.transforms.window as window
import arc_beam.transforms.combiners as combiners
import baloo as pandas

def normalise(data):
    series = pandas.Series(data)
    avg = series.sum() / series.count()
    return series / avg

p = beam.Pipeline()

(p
 | beam.io.ReadFromText(path='input.txt').with_output_types(int)
 | beam.WindowInto(window.FixedWindows(size=5))
 | beam.CombineGlobally(normalise)
 | combiners.ToList()
 | beam.io.WriteToText(path='output.txt'))

p.run()
```

Generated Arc IR code

```
|source_0: stream[i64], sink_0: streamappender[?]|
  let operator_0 = result(for(source_0, windower[unit,appender[?],?,vec[?]](
        |ts,windows,state| { [ts/5000L], () },
        |wm,windows,state| { result(filter(windows, |ts| ts < wm), () },
        |agg| result(agg)
      ),
      |sb,se| merge(sb, se)
  ));
  for(operator_0, sink_0, |sb,se| merge(sb,
    let obj102 = (se);
    let obj105 = (result(
      for(obj102, merger[i64, +],
        |b: merger[i64, +], i: i64, e: i64|
          merge(b, e)
    )));
    let obj106 = (len(obj102));
    let obj107 = (obj105 / obj106);
    let obj108 = (result(
      for(obj102, appender[i64],
        |b: appender[i64], i: i64, e: i64|
          merge(b, e / obj107)
    )));
    obj108
  ))
```

Stream code (Beam)

Global tumbling window

Sum

Batch code (Pandas)

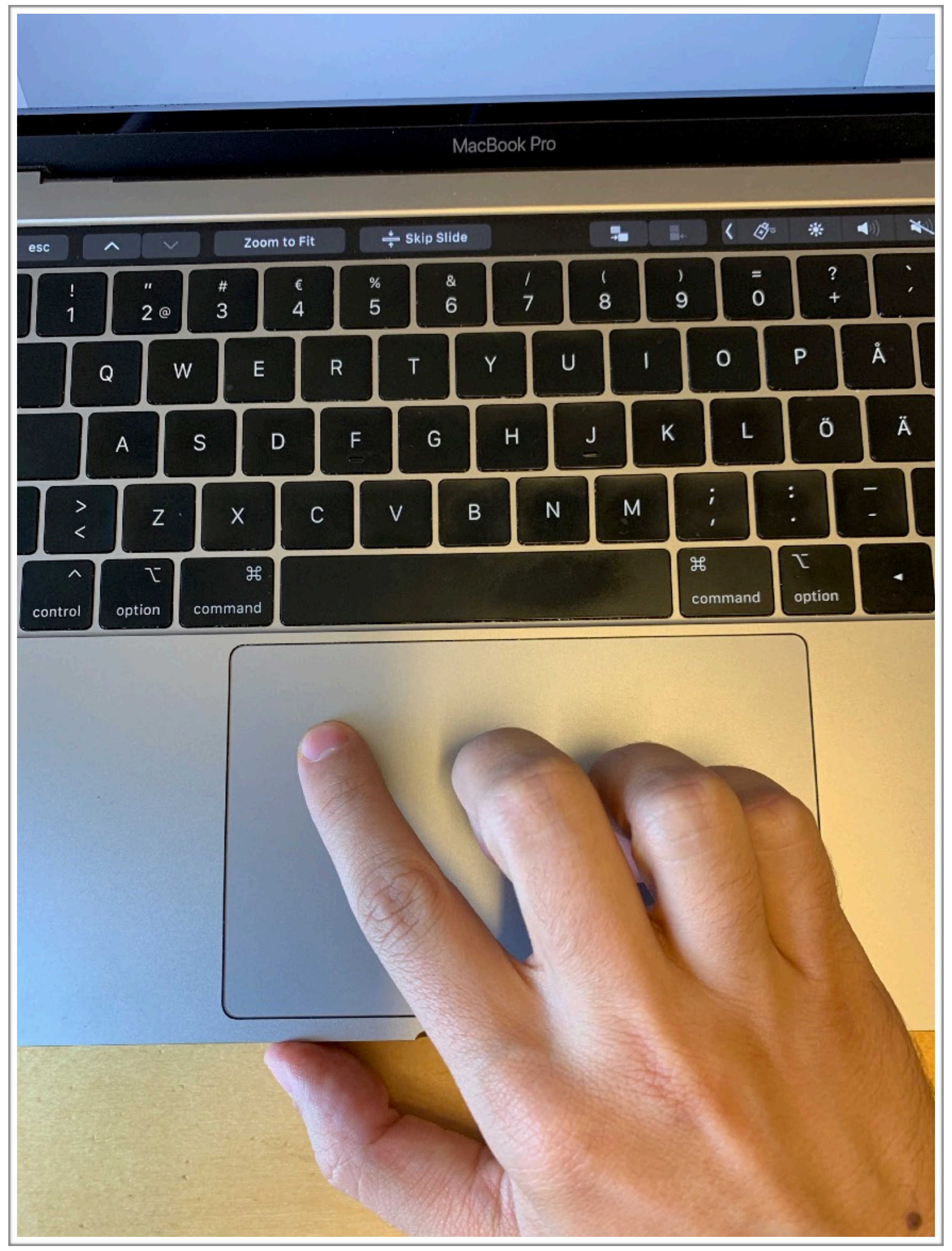
Count

Average

Normalisation

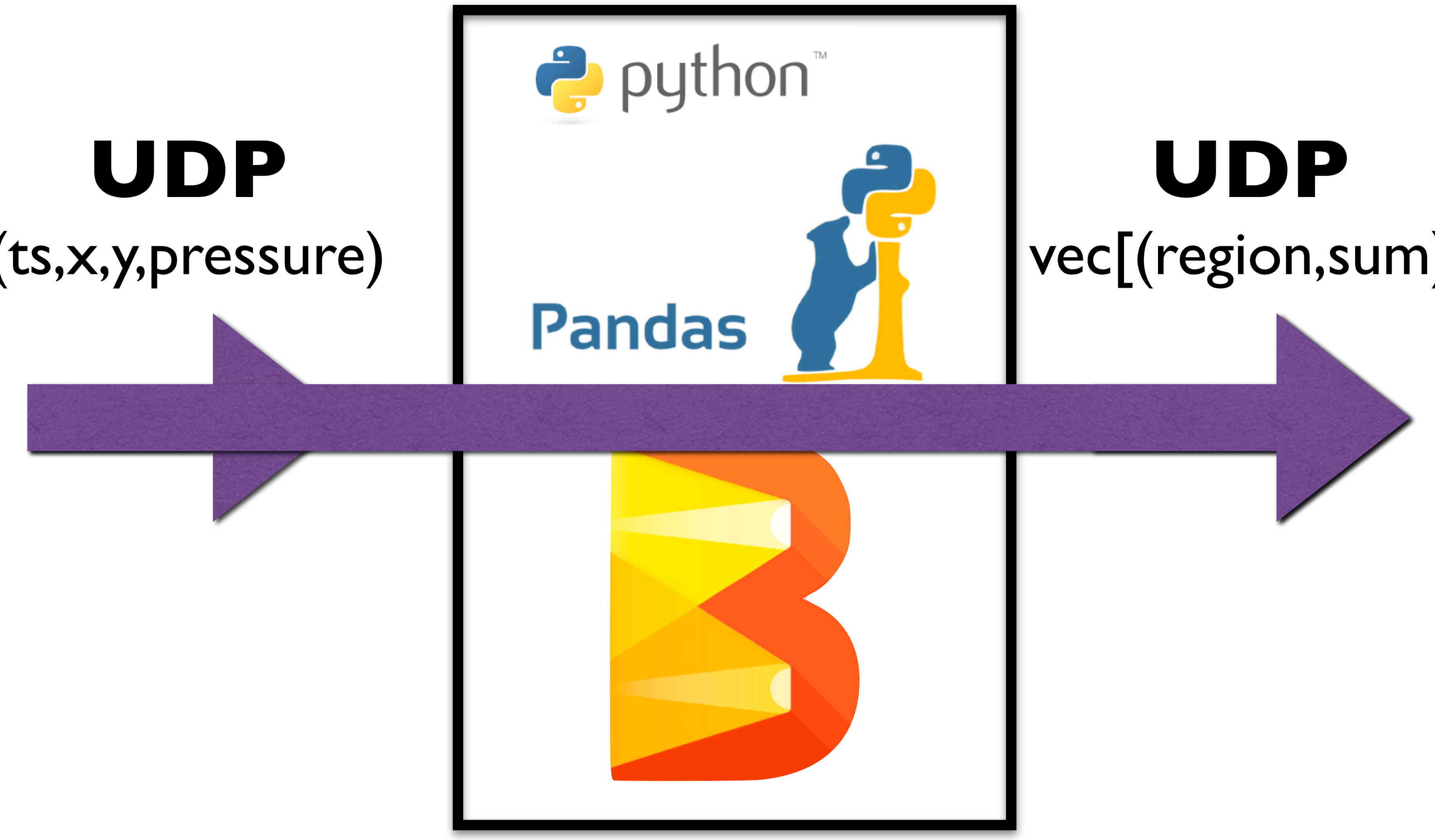
Demo: Touchpad Heatmap

The Input



The Pipeline

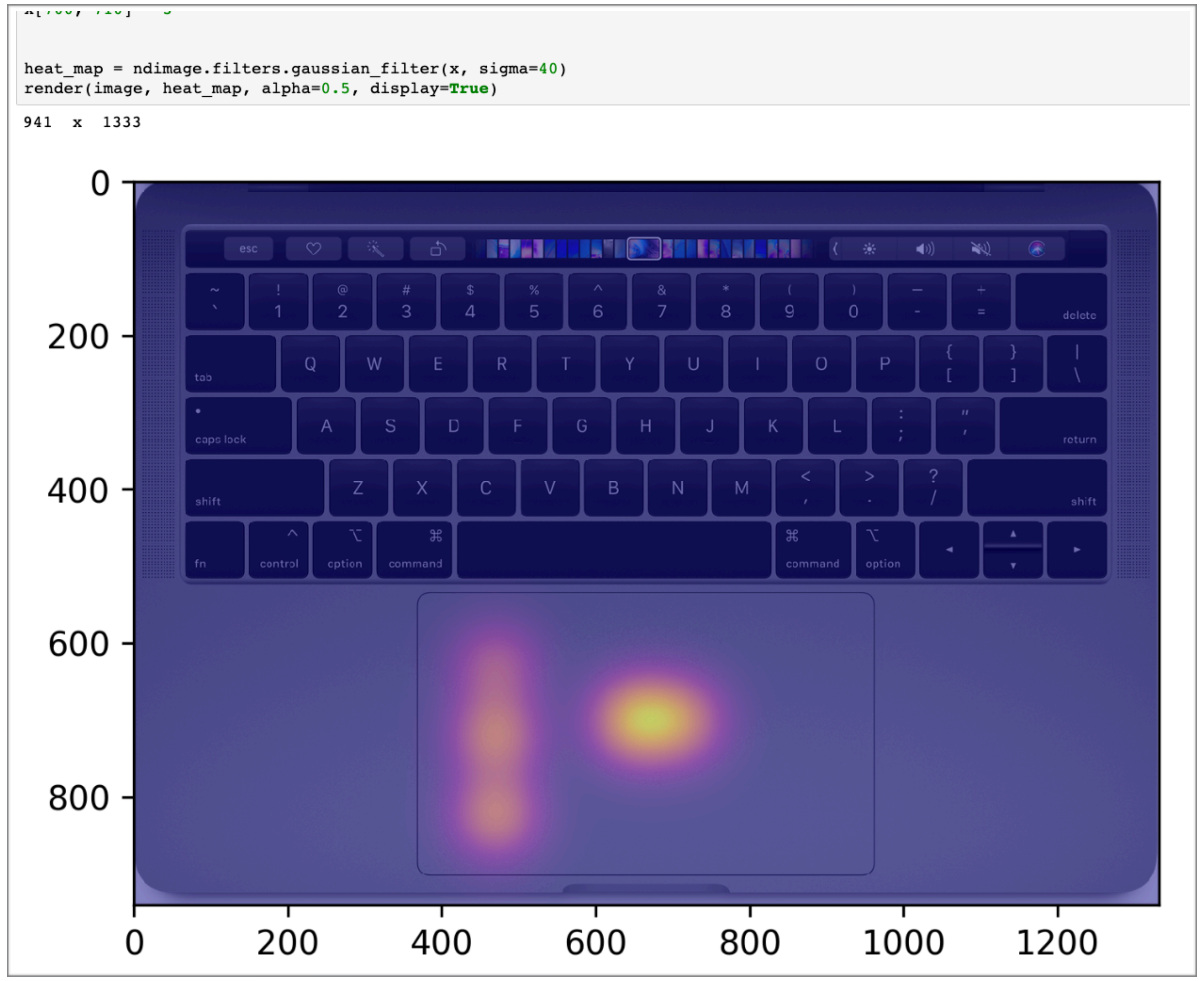
UDP
(ts,x,y,pressure)



UDP
vec[(region,sum)]

- **Touchpad is a grid of 5x3 regions**
- **Event-time window (6 seconds)**
- **Sum up pressure by region**

The Output



Touchpad Heatmap Demo

MouseTracker 〉 MouseTracker 〉 AppDelegate.swift 〉 No Selection

- MouseTracker
  - CocoaAsyncSocket.framework
    - Headers
      - CocoaAsyncSocket.h
      - GCDAsyncSocket.h
      - GCDAsyncUdpSocket.h
  - MouseTracker
    - AppDelegate.swift
    - Assets.xcassets
    - MainMenu.xib
    - Info.plist
    - MouseTracker.entitlements
    - NetSender.swift M
  - Products
    - MouseTracker.app

```swift
//
//  AppDelegate.swift
//  MouseTracker
//
//  Created by Lars Kroll on 2019-08-13.
//  Copyright © 2019 Lars Kroll. All rights reserved.
//

import Cocoa

@NSApplicationMain
class AppDelegate: NSObject, NSApplicationDelegate {

    @IBOutlet weak var window: NSWindow!


    var udpSender: NetSender?;

    func applicationDidFinishLaunching(_ aNotification: Notification) {
        // Insert code here to initialize your application
        print("setting up!");
        var frame = self.window?.frame
        frame?.size = NSSize(width: 505, height:323)
        self.window?.setFrame(frame!, display: true)
        udpSender = NetSender();
```

```
setting up!
Data sent: Test!
Did send data!
594886868,11.1484375,315.83984375,0.0

Data sent: 594886868,11.1484375,315.83984375,0.0

Did send data!
```

Type: Default - Swift Source
Location: Relative to Group
AppDelegate.swift
Full Path: /Users/carbone/workspace/mactouchsender/MouseTracker/AppDelegate.swift

On Demand Resource Tags

Only resources are taggable

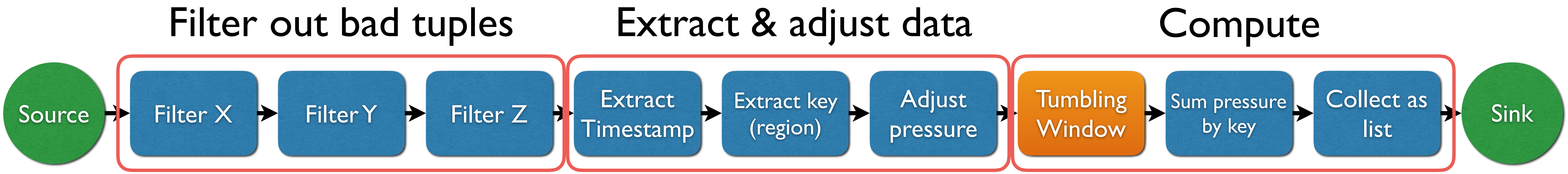
Target Membership

☑ MouseTracker

Text Settings

Text Encoding: No Explicit Encoding
Line Endings: No Explicit Line Endings
Indent Using: Spaces
Widths: 4 Tab, 4 Indent
☑ Wrap lines

Dataflow Graph



Touchpad Beam Code

```
p = beam.Pipeline()

(p
 | beam.io.ReadFromSocket(addr=touchpad, coder=beam.coders.CSVCoder())
          .with_output_types(Tuple[ts, x, y, z])
 | 'preprocess'
   >> beam.Filter(lambda e: (e[1] >= 0) & (e[1] <= width))
 |    beam.Filter(lambda e: (e[2] >= 0) & (e[2] <= height))
 |    beam.Filter(lambda e: (e[3] >= 0) & (e[3] <= max_pressure))
 | 'extract timestamp'
   >> beam.Map(lambda e: window.TimestampedValue(value=e[1:4], timestamp=e[0]))
 | 'extract key'
   >> beam.Map(lambda e: ((e[0] / grid_width, e[1] / grid_height), e[2]))
 | 'add to pressure'
   >> beam.Map(lambda e: (e[0], e[1] + epsilon))
 | 'create tumbling window'
   >> beam.WindowInto(window.FixedWindows(size=window_length))
 | 'sum up pressures'
   >> beam.CombinePerKey(lambda e: pandas.Series(e).sum())
 | 'collect window as list'
   >> combiners.ToList()
 | beam.io.WriteToSocket(addr=display, coder=beam.coders.CSVCoder()))

p.run()
```

Code-generation steps

KTH VETENSKAP OCH KONST
ROYAL INSTITUTE OF TECHNOLOGY
RI. SE

```python
from typing import Tuple
import arc_beam as beam
import arc_beam.transforms.window as wind
import arc_beam.transforms.combiners as c
import baloo as pandas

width, height = 500, 300  # 500x300 point
num_x_grids, num_y_grids = 5, 3  # 5x3 gr
grid_width, grid_height = int(width / num
epsilon = 0.1
touchpad, display = '127.0.0.1:8000', '12
ts, x, y, z = int, float, float, float
max_pressure = 1.0
window_length = 6
p = beam.Pipeline()
(p
 | beam.io.ReadFromSocket(addr=touchpad,
          .with_output_types(Tuple[ts, x,
 | 'preprocess'
    >> beam.Filter(lambda e: (e[1] ≥ 0)
 |      beam.Filter(lambda e: (e[2] ≥ 0)
 |      beam.Filter(lambda e: (e[3] ≥ 0)
 | 'extract timestamp'
    >> beam.Map(lambda e: window.Timestam
 | 'extract key'
    >> beam.Map(lambda e: (((e[0] / grid_
 | 'add to pressure'
    >> beam.Map(lambda e: (e[0], e[1] + e
 | 'create tumbling window'
    >> beam.WindowInto(window.FixedWindow
 | 'sum up pressures'
    >> beam.CombinePerKey(lambda e: panda
 | 'collect window as list'
```


```json
{
  "nodes": [
    {
      "id": "source_0",
      "kind": {
        "Source": {
          "format": "CSV",
          "kind": {
            "Socket": {
              "addr": "127.0.0.1:8000"
            }
          }
        }
      }
    },
    {
      "id": "sink_0",
      "kind": {
        "Sink": {
          "format": "CSV",
          "kind": {
            "Socket": {
              "addr": "127.0.0.1:9000"
            }
          }
        }
      }
    }
  ],
  "timestamp_extractor": 0,
  "arc_code": "|source_0: stream[{i64,
f64,f64,f64}], sink_0: streamappender[?
]|\n# preprocess\nlet operator_1 =
```

```json
{
  "id": "dfg_0",
  "target": "x86-64-unknown-linux-gnu",
  "nodes": [
   {
     "id": "source_0",
     "parallelism": 1,
     "kind": {
       "Source": {
         "source_type": {
           "Struct": {
             "id": "struct_source_0_tas
             "field_tys": [
               {
                 "Scalar": "I64"
               },
               {
                 "Scalar": "F64"
               },
               {
                 "Scalar": "F64"
               },
               {
                 "Scalar": "F64"
               }
             ]
           }
         },
         "format": "CSV",
         "channel_strategy": "Forward",
         "successors": [
           {
             "Local": {
```


```rust
#![allow(dead_code)]
#![allow(missing_docs)]
#![allow(unused_imports)]
#![allow(unused_results)]
#![allow(bare_trait_objects)]

extern crate arcon;
use arcon::macros::*;
use arcon::prelude::*;
#[macro_use]
extern crate serde;
#[arcon_decoder(,)]
#[arcon]
pub struct struct_source_0_task_0_task_1
    f0: i64,
    f1: f64,
    f2: f64,
    f3: f64,
}
impl ::std::hash::Hash for struct_source
    fn hash<H: ::std::hash::Hasher>(&sel
}
#[arcon_decoder(,)]
#[arcon]
pub struct struct_task_4_task_5_0 {
    f0: struct_task_4_task_5_1,
    f1: f64,
}
impl ::std::hash::Hash for struct_task_4
    fn hash<H: ::std::hash::Hasher>(&sel
}
#[arcon_decoder(,)]
#[arcon]
```

python™ → Arc + Metadata → Dataflow → Rust

Dataflow Graph

Initial pipeline

Fused pipeline

Operators are fused by inlining at the instruction level

ROYAL INSTITUTE OF TECHNOLOGY

Arc Optimisations

- Arc supports **both** compiler and dataflow optimisations
  - **Compiler**: Loop unrolling, partial evaluation,
  - **Dataflow**: Operator fusion, fission, reordering, specialisation, ...
- Find optimal dataflow graph through constraint model (future work)

Conclusions & Future Work

- Arc enables cross-compiling and optimising programs from diverse libraries.
- Next steps:
  - Wider support for more frontends, Tensorflow, Flink, etc.
  - Common Pipeline DSL
  - State management for dynamic task graphs
  - Runtime Optimiser and Reconfiguration

Extra slides

ROYAL INSTITUTE OF TECHNOLOGY

Optimisation example