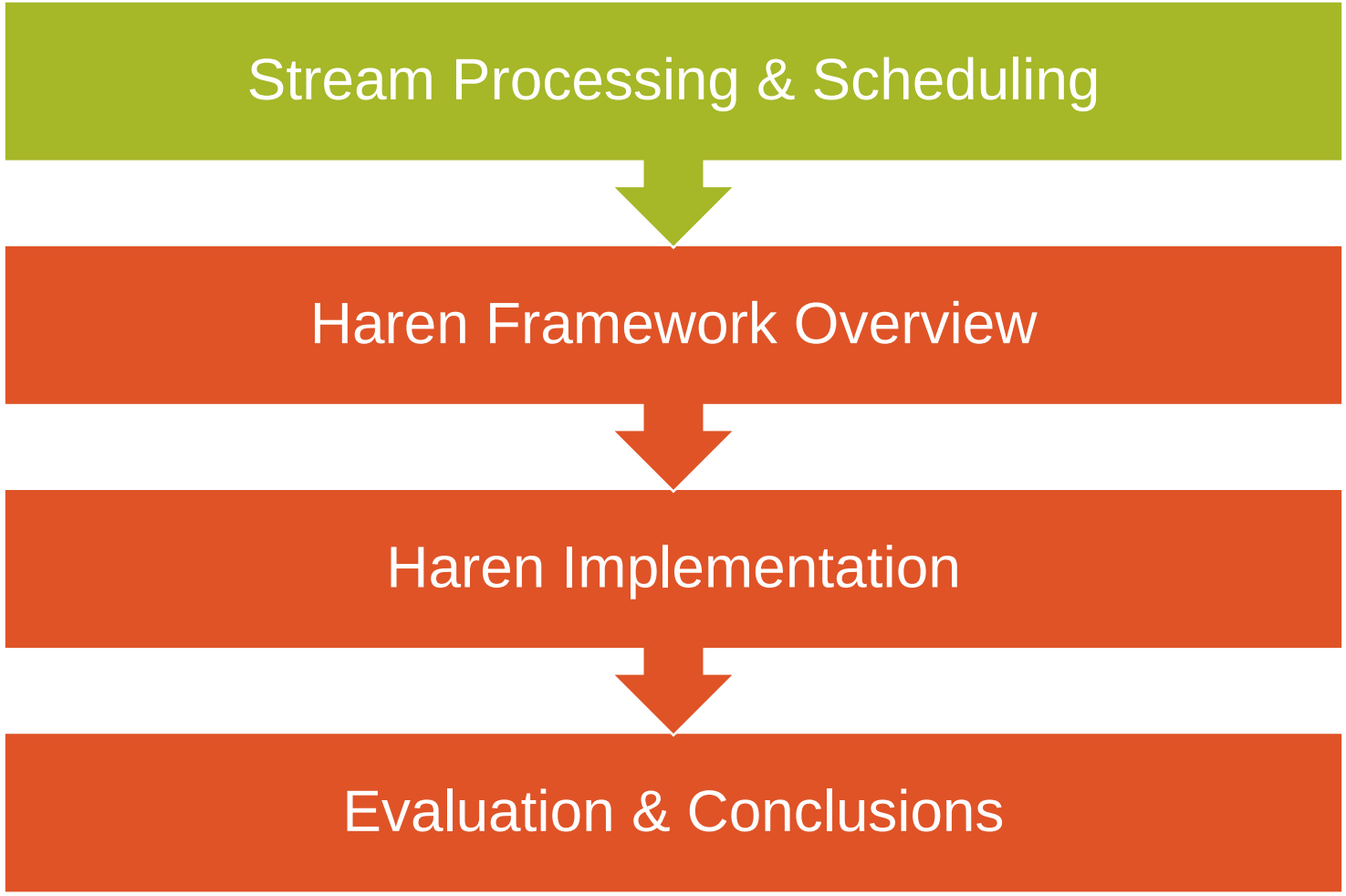
Distributed Computing and Systems
Chalmers university of technology

Haren: A Framework for Ad-Hoc Thread Scheduling Policies for Data Streaming Applications

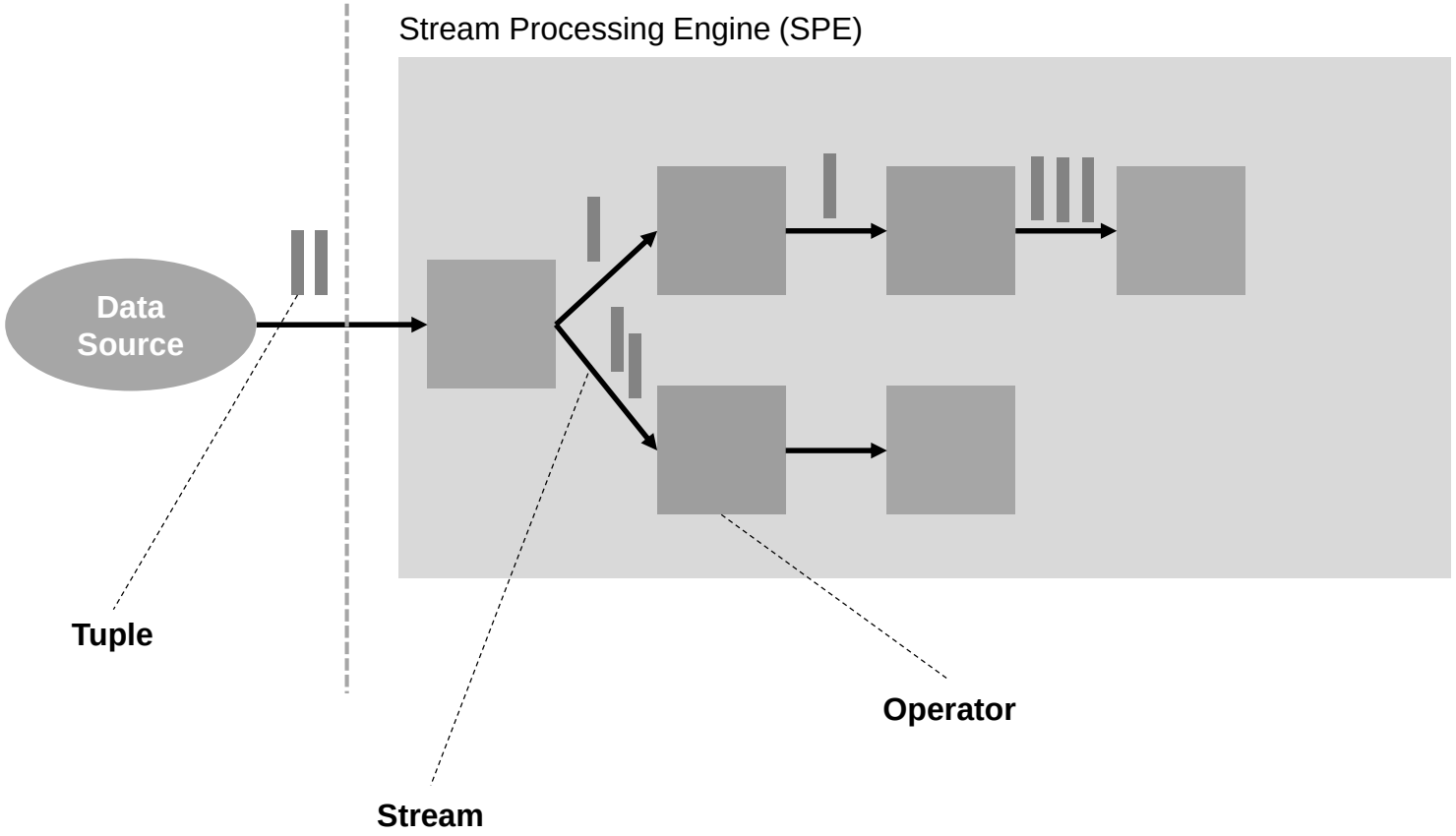
Dimitris Palyvos-Giannas, Vincenzo Gulisano, Marina Papatriantafilou

13th International Conference on Distributed and Event-Based Systems

June 24-28, 2019, Darmstadt


Stream Processing & Scheduling
Haren Framework Overview
Haren Implementation
Evaluation & Conclusions

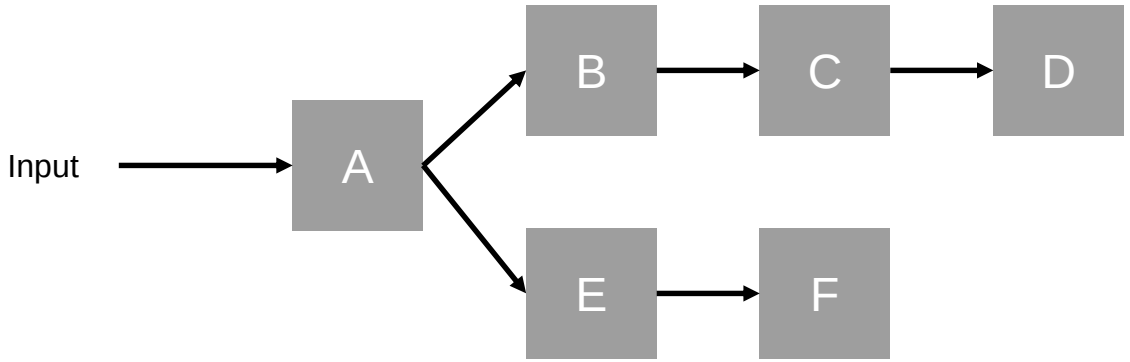
Stream Processing Basics



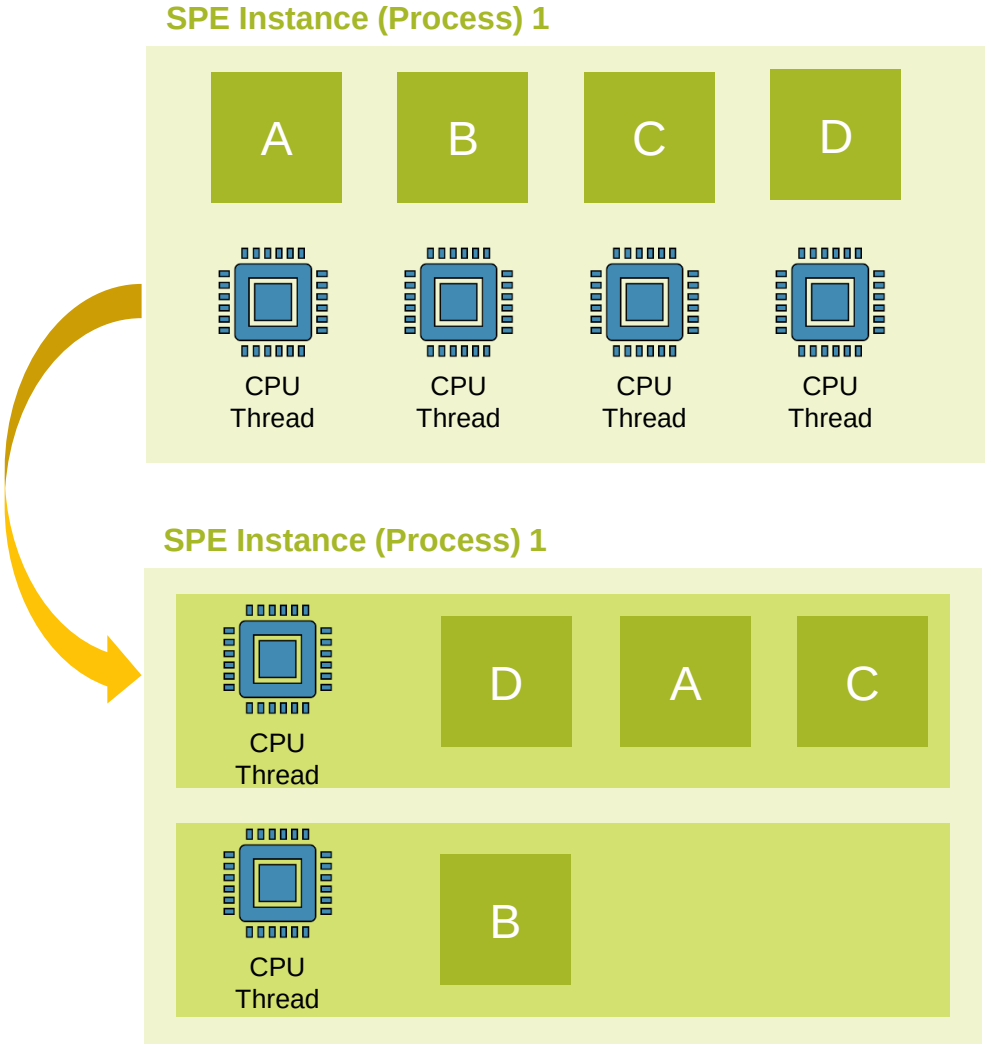
Performance Metrics

- Throughput
- Latency
- CPU Utilization
- Memory Utilization

Resource Scheduling



Thread Scheduling inside SPE Instances



- Operators are executed by **CPU threads**.
- Usually **one dedicated thread per operator**.
- What if **#CPUs < #Operators**?
- **Operating System scheduler** allocates CPU…
- …but it has **no knowledge of streaming goals**!

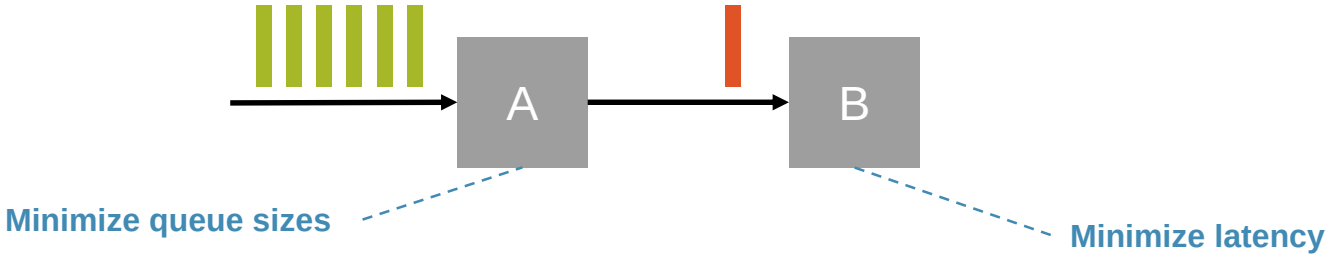
- Alternative: **Application-level thread scheduling**
- Can optimize for specific **performance goals!**
- For a (short) time interval, **two questions**:
  1. How to **assign** operators to threads (inter-thread)?
  2. What is the **priority** of operators for each thread (intra-thread)?

→ **Two scheduling functions** – almost any policy!

Custom Thread Scheduling

Which operator to execute?

It depends! Custom scheduling = **more choice**!



But there are **obstacles**…

- **Low-level programming details.**
  - Difficult to program.
  - Difficult to ensure efficiency and correctness.
- Schedulers **programmed to specific SPE.**
  - Reinventing the wheel - cannot reuse code.
  - Difficult to port scheduling policies to other SPEs.

Most SPEs avoid custom thread scheduling!

Haren: A Scheduling Framework for Streaming

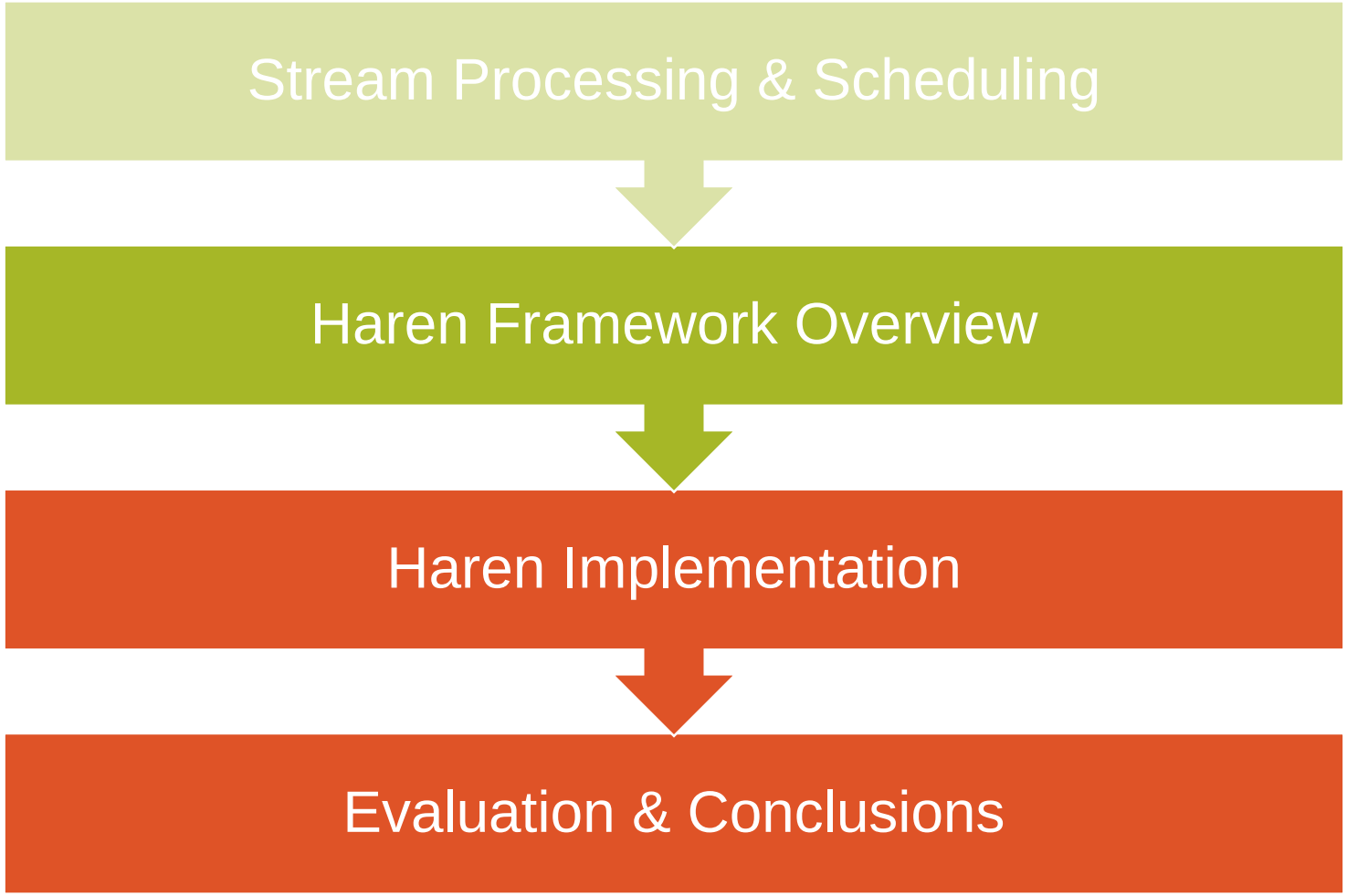
Haren **hides the complexity** of custom scheduling: User only programs **high-level scheduling logic**!

Goal 1 **Compact interface** that allows implementing arbitrary scheduling policies.

Goal 2 Configuration of both **inter-thread** and **intra-thread** rules.

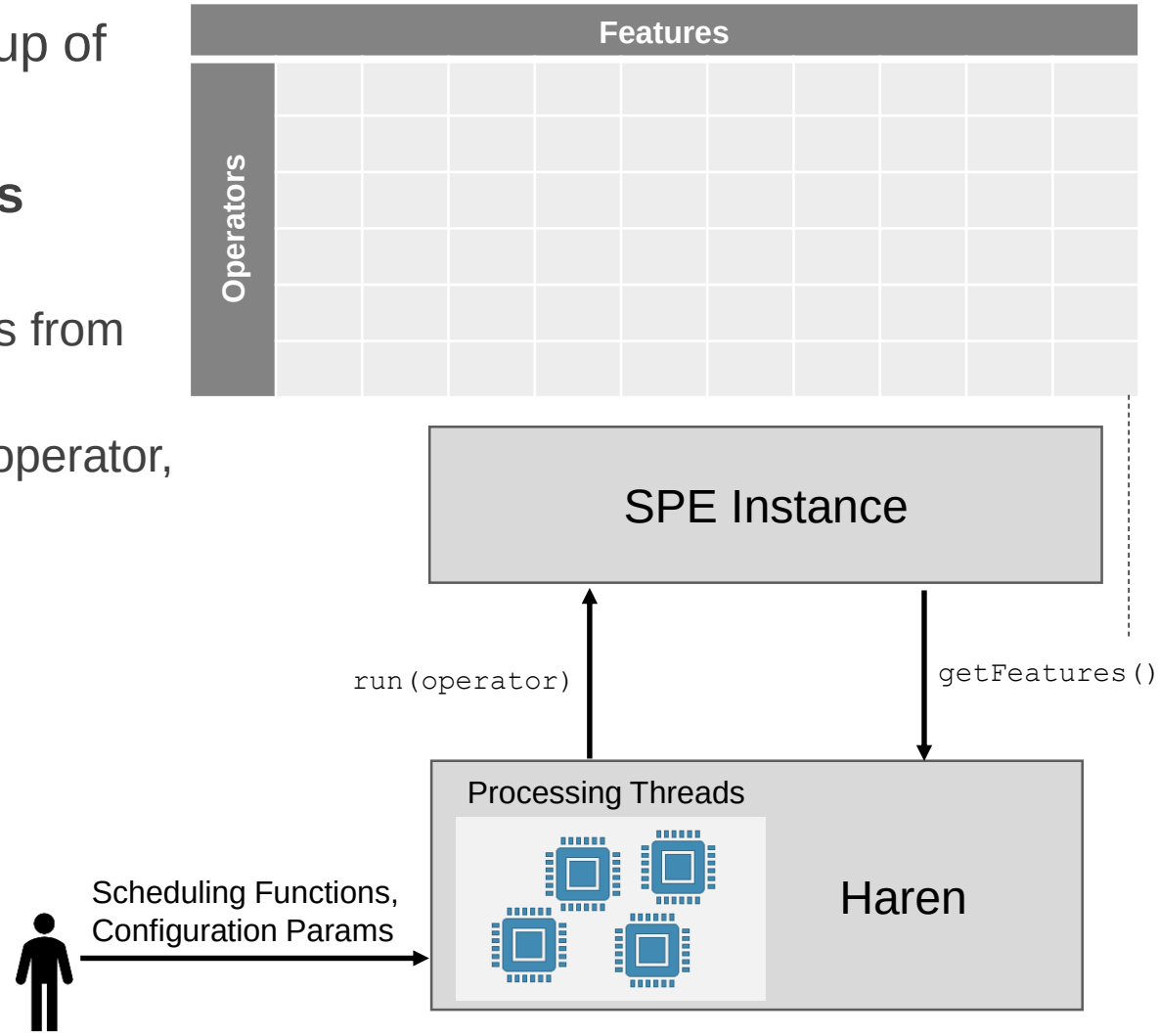
Goal 3 **Parallelization** of scheduling computation when possible.

Reusable scheduling policies in different SPEs!


Stream Processing & Scheduling
Haren Framework Overview
Haren Implementation
Evaluation & Conclusions

Haren Overview

- Orchestrates operator execution using a group of **Processing Threads (PTs)**.
- Remains SPE-agnostic by using the **features abstraction**.
  - Retrieves necessary features of the operators from the SPE through a **well-defined interface**.
  - A **feature** is any value that characterizes an operator, its streams or its tuples.
  - Example features: cost, input stream size, …
  - Maintains a **table of operator features**.
- Applies high-level **user-defined scheduling functions** to the features to take scheduling decisions.

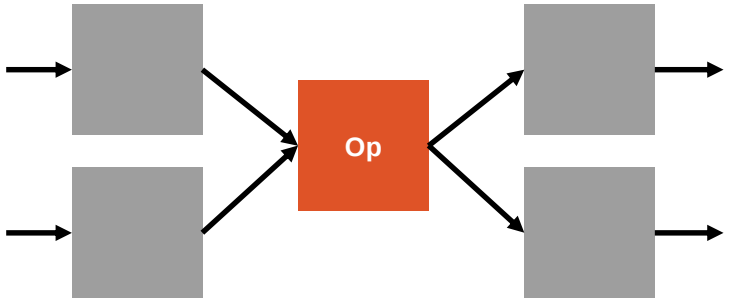


Feature Categories

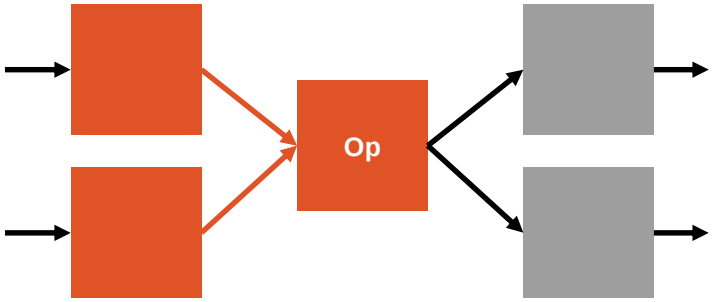
Static (e.g., operator type)
Remain constant

Dynamic (e.g., cost, input stream size)
Can change over time

Independent (e.g., cost)
Can only change upon execution of Op

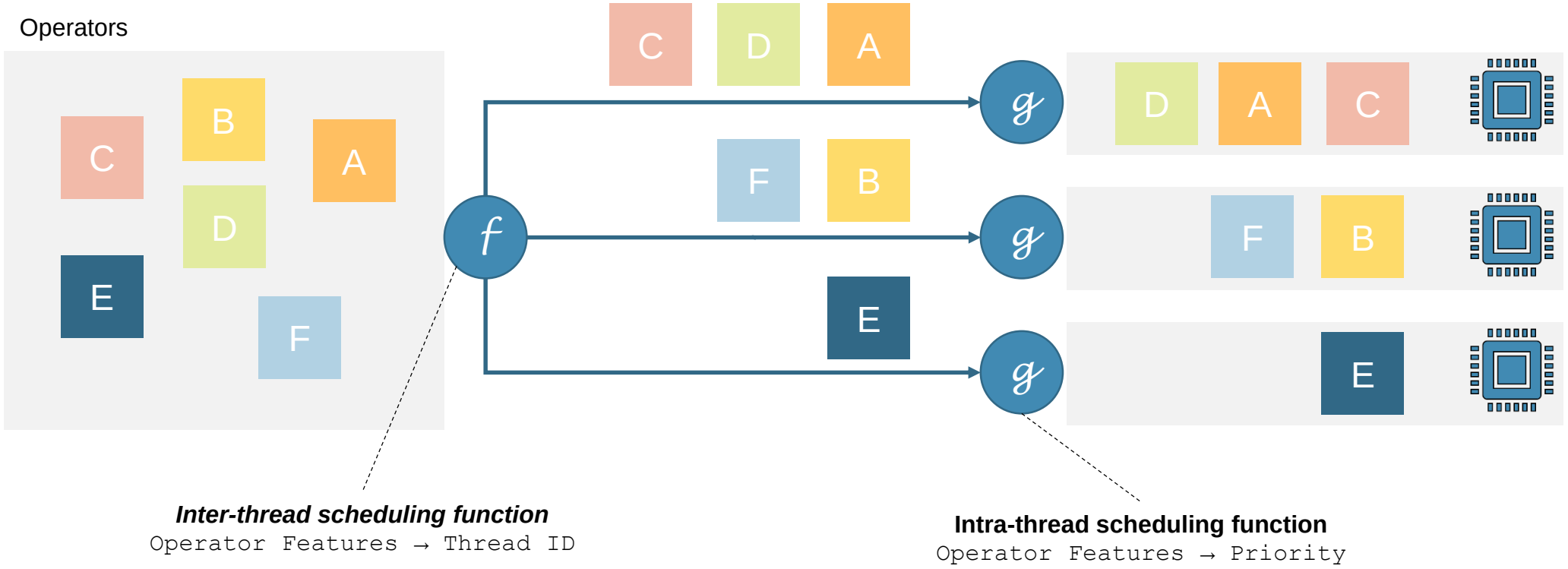


Dependent (e.g., input stream size)
Can change upon execution of some other operator ≠ Op

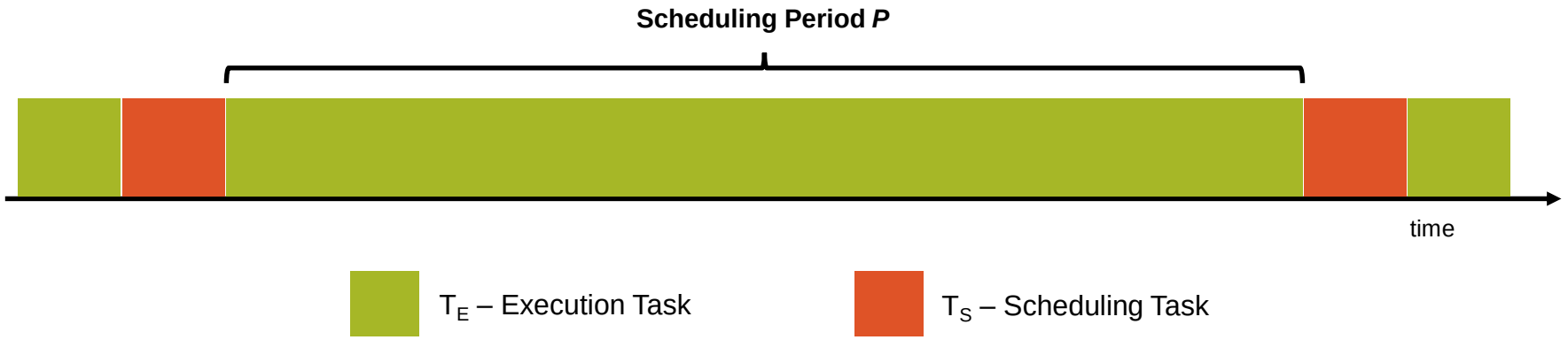


Inter & Intra Thread Scheduling Functions

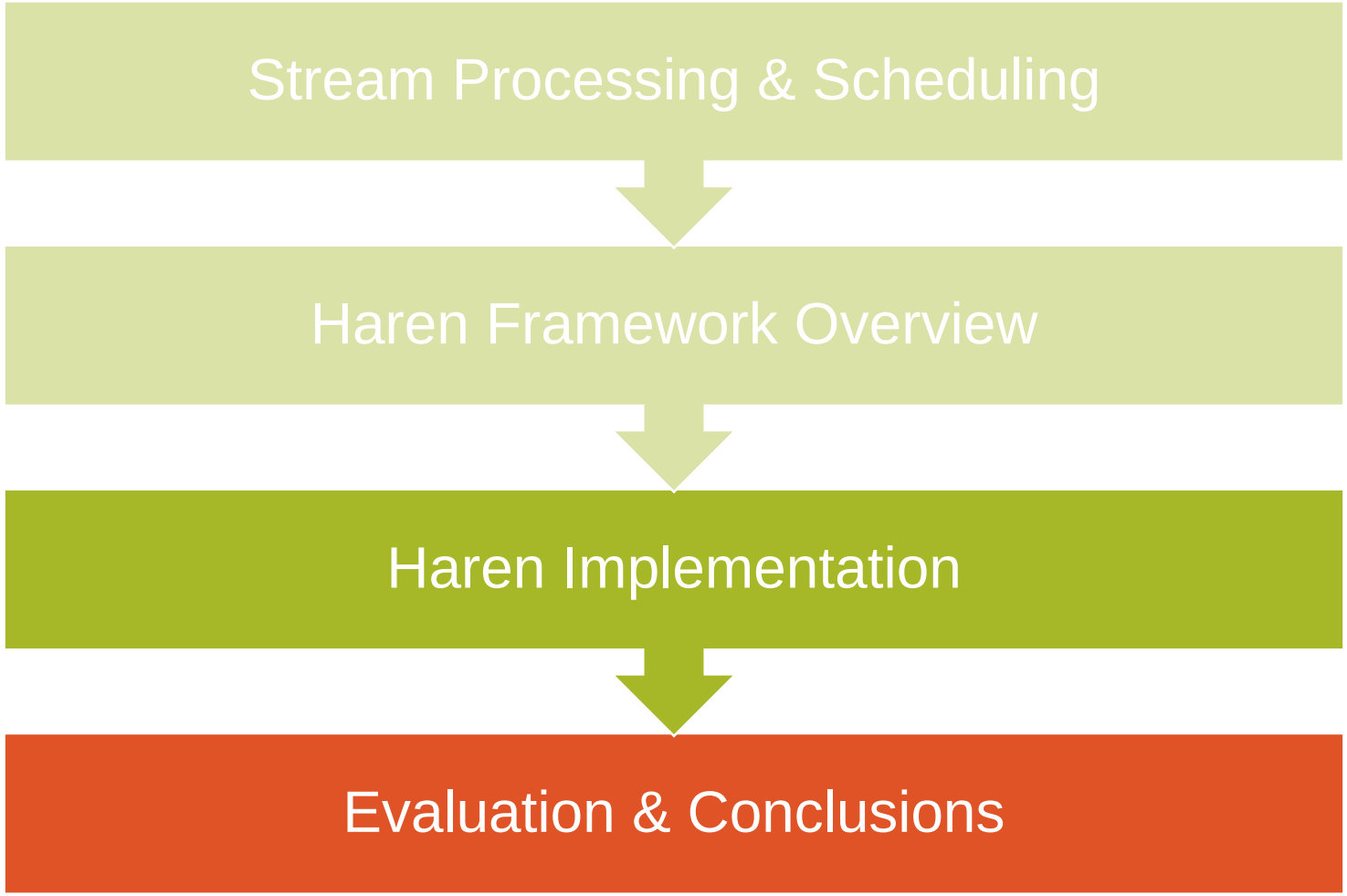
1. Inter-thread: How to **assign** operators to threads?
2. Intra-thread: How to compute the **priority** of operators in each thread?



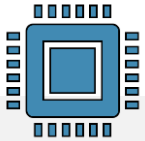
Haren Processing Thread (PT) Behavior



- PTs **execute** operators most of the time (**T_E**).
- Dynamic nature of stream processing ➞ features & priorities change over time.
- PTs periodically switch to **scheduling**, updating features and scheduling decisions (**T_S**).
- Fine-grained control over scheduling overhead by tuning the scheduling period *P*.


Stream Processing & Scheduling
Haren Framework Overview
Haren Implementation
Evaluation & Conclusions

Execution Task T_E

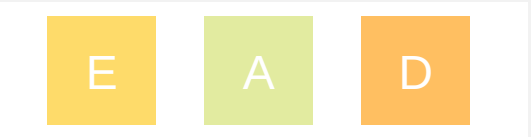
Processing Thread 

Main Loop

```
while running:
  while elapsed_time < scheduling_period:
    starting from the beginning of assigned
    pick first operator that can run
    (has input > 0 and output capacity > 0)
    if found operator that can run:
      process max b tuples
    if no operator can run:
      back-off (sleep)
   goto scheduling task T_S
```

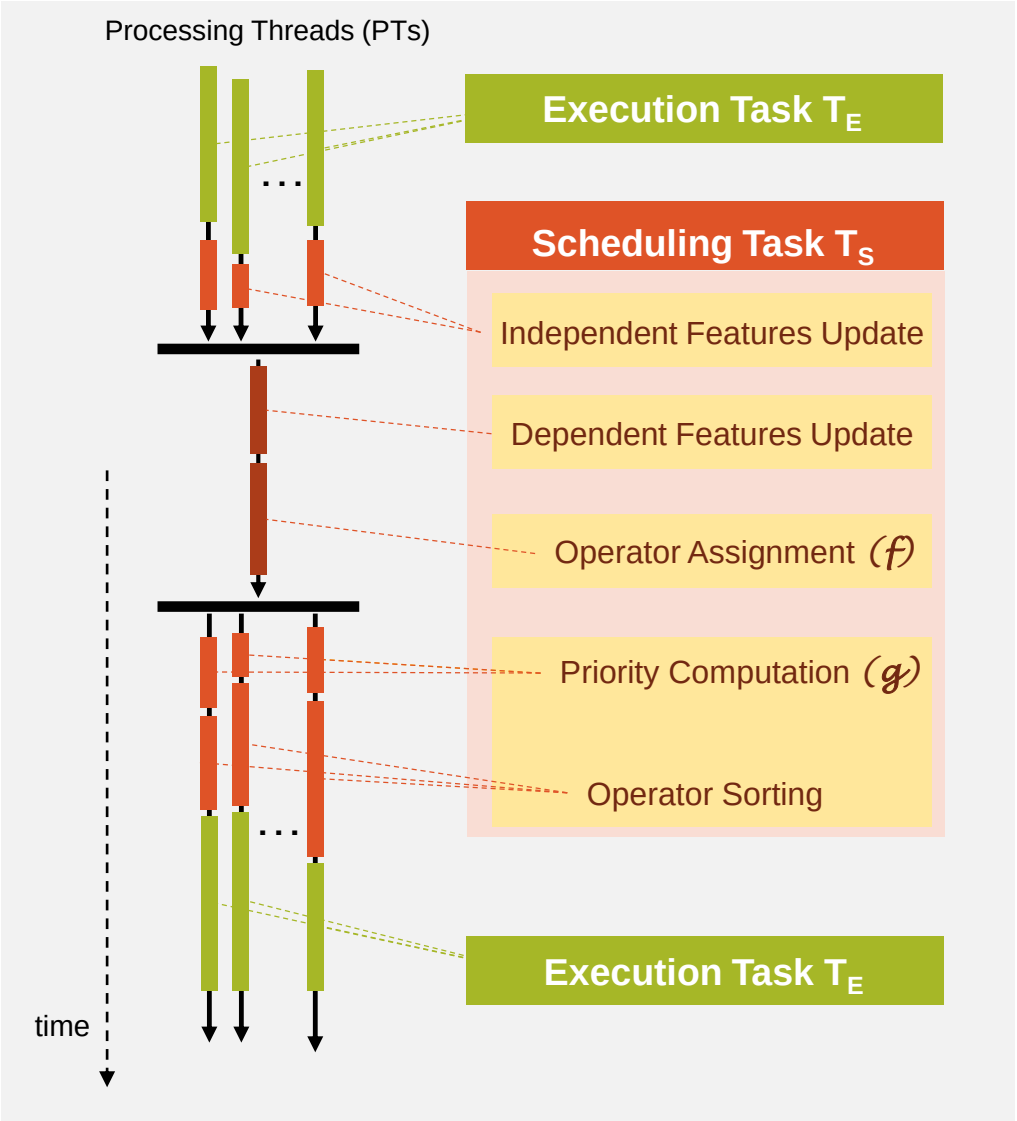
State

Assigned Array *(Computed in previous T_S)*



Scheduling Task T_S

- **Decides schedule** for the next T_E.
- Computes a **new *assigned* array** for each PT where operators sorted on priority.
- Most steps are executed in **parallel** by all PTs.
- Few **sequential** steps for PTs to synchronize and agree on scheduling decisions.



T_S: Independent Features Update

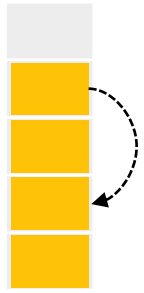
Processing Thread Concurrent

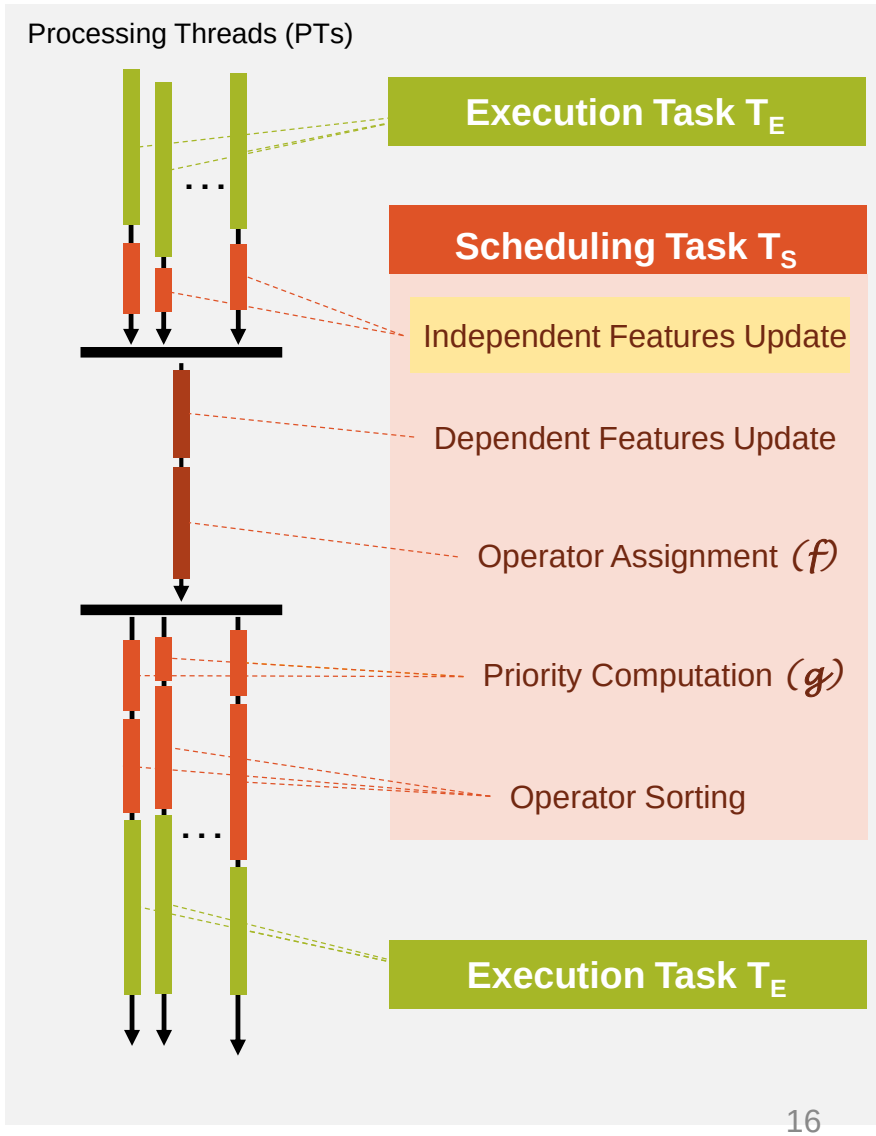
```
for op in Executed
  update independent features of op
  mark op & dependent operators
```

Feature Table

Operators		Features								
		F1	F2	F3	F4	F5	F6	F7	F8	F9
	A									
	B			PT#1						
	C			PT#2						
	D			PT#3						
	E			PT#1						

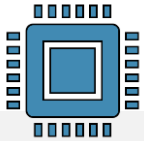
Marked Table (Bool)
(needed for next step)





T_S: Dependent Features Update

Thread t*



Sequential

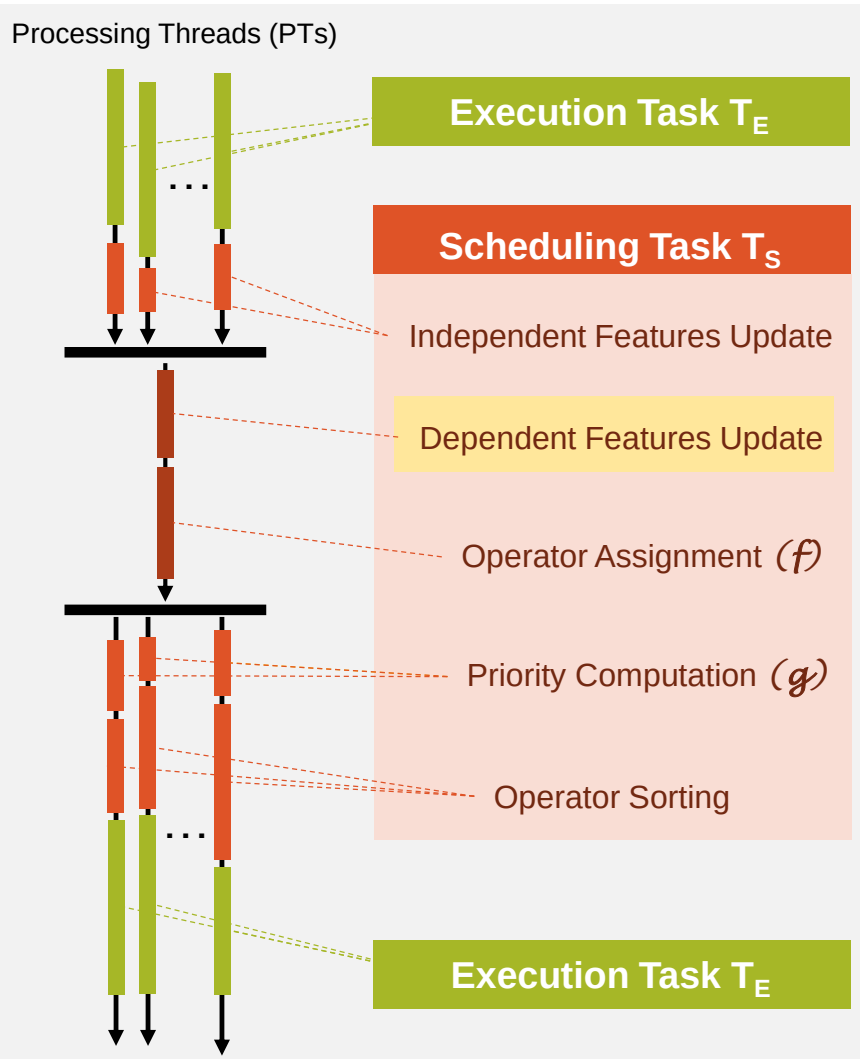
```
for op in All_Operators
  if op is marked
    update dependent features of op
```

Haren **only** updates features that:
- Have (potentially) changed.
- Are used by the scheduling functions f, g.

Feature Table

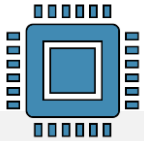
Operators	Features	F1	F2	F3	F4	F5	F6	F7	F8	F9
	A									
	B									
	C									
	D									
	E									

Marked Table



T_S: Operator Assignment

Thread t*



Sequential

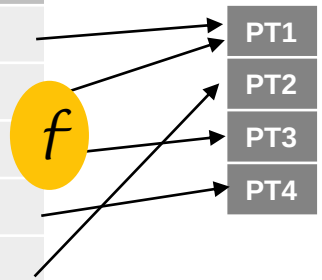
```
for op in All_Operators
  threadID = f(op)
  append op to assigned[threadID]
```

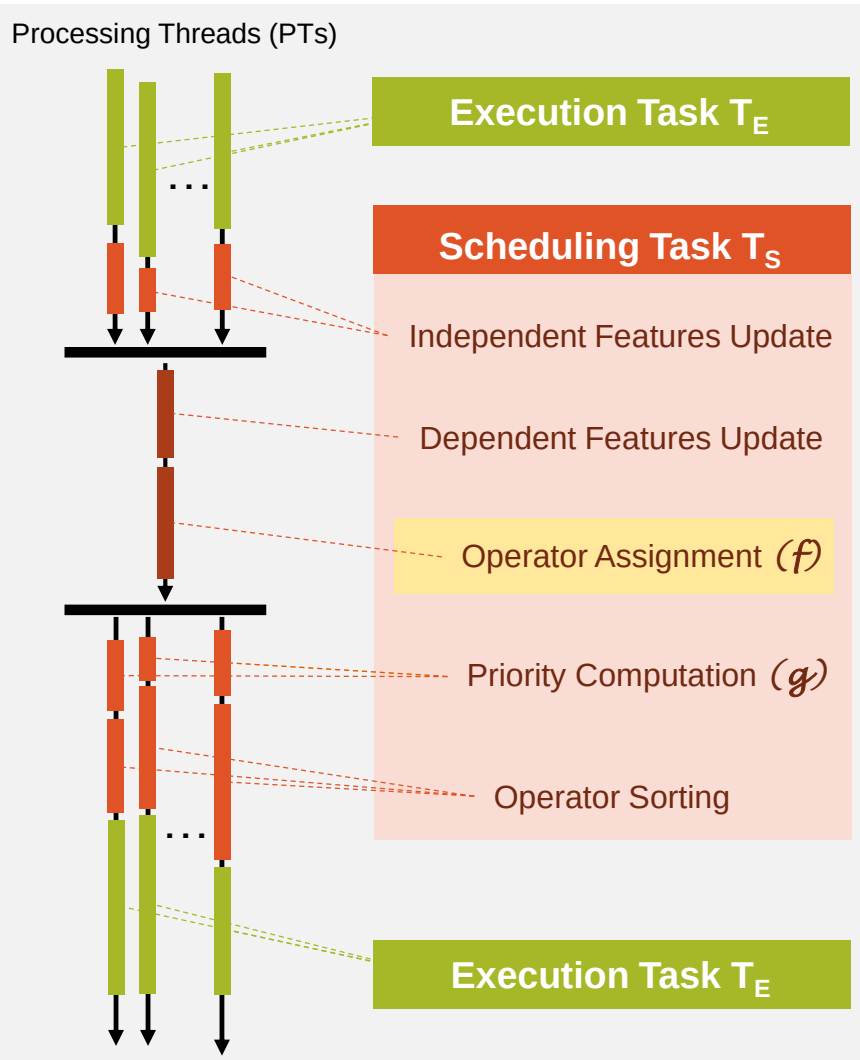
Feature Table

Operators		F1	F2	F3	F4	F5	F6	F7	F8	F9
	A									
	B			■	■	■	■		■	
	C			■	■	■	■		■	
	D			■	■	■	■		■	
	E			■	■	■	■		■	

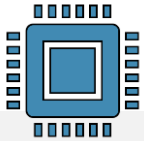
Features

Assigned
Operators per PT



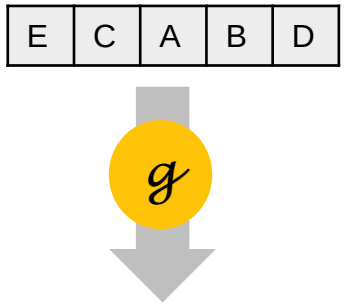


T_S: Priority Computation & Sorting

Processing Thread  Concurrent

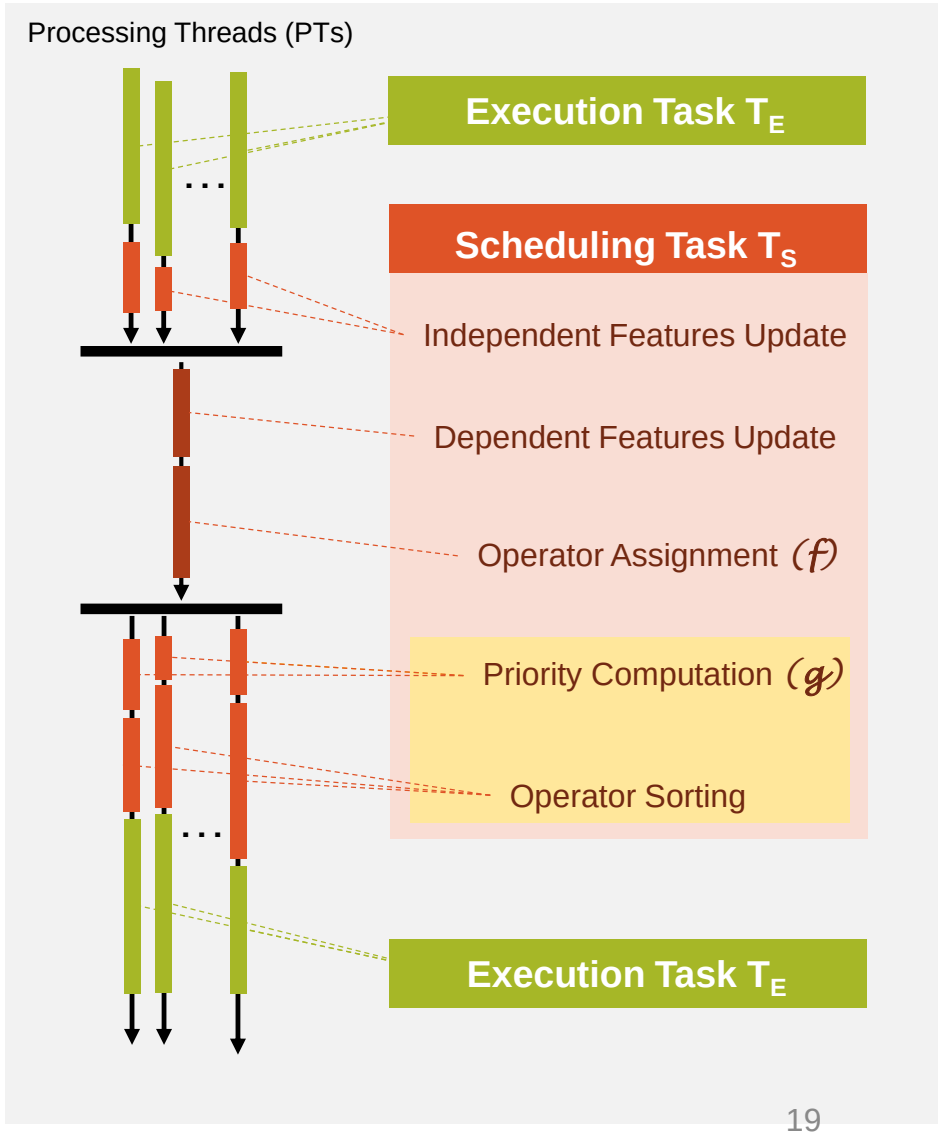
```
for op in assigned
  priority[op] = g(op)
sort assigned on priority
```

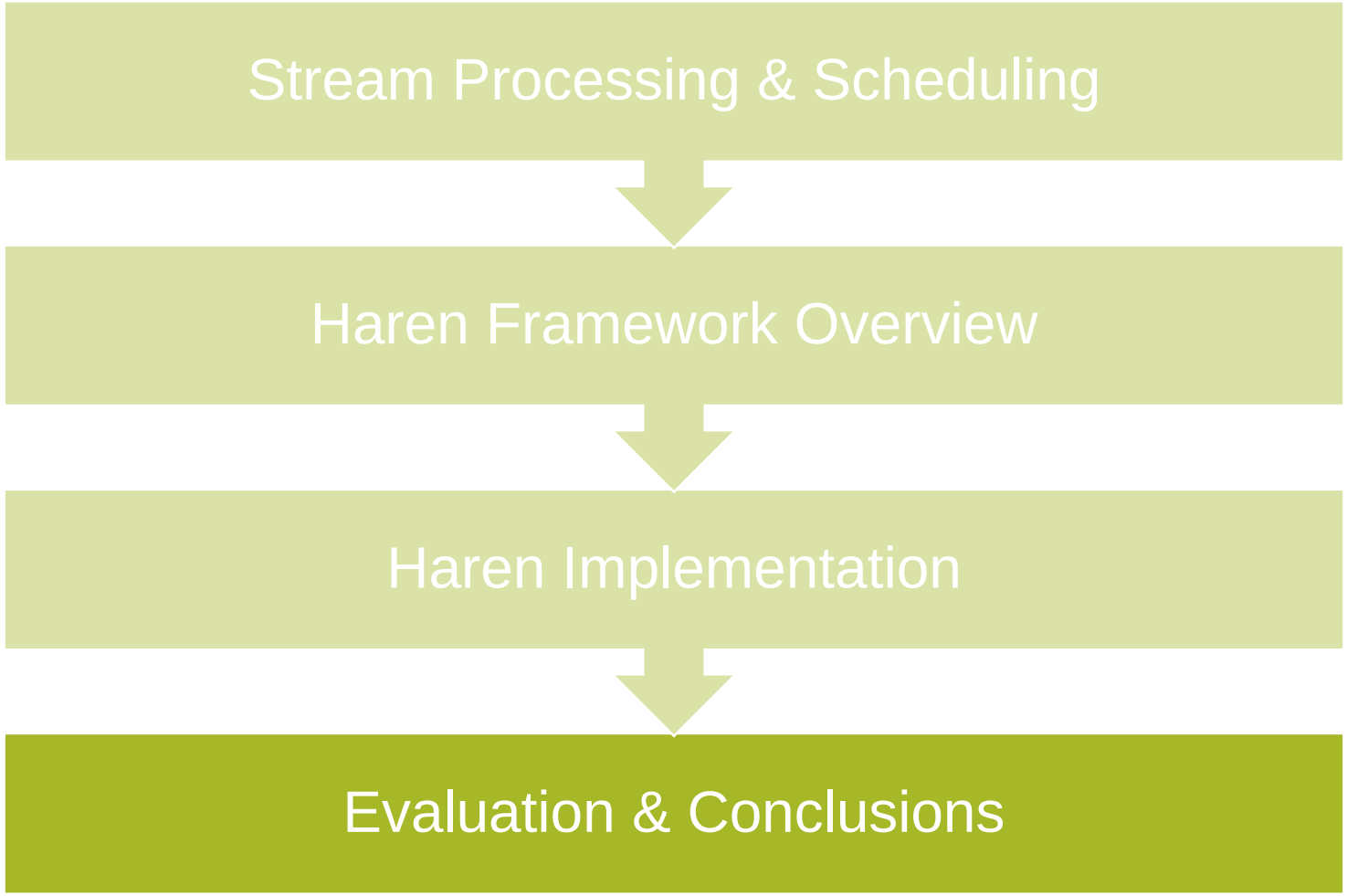
Assigned (after previous step)



Assigned (after priority computation)

Assigned (after sort)




Stream Processing & Scheduling
Haren Framework Overview
Haren Implementation
Evaluation & Conclusions

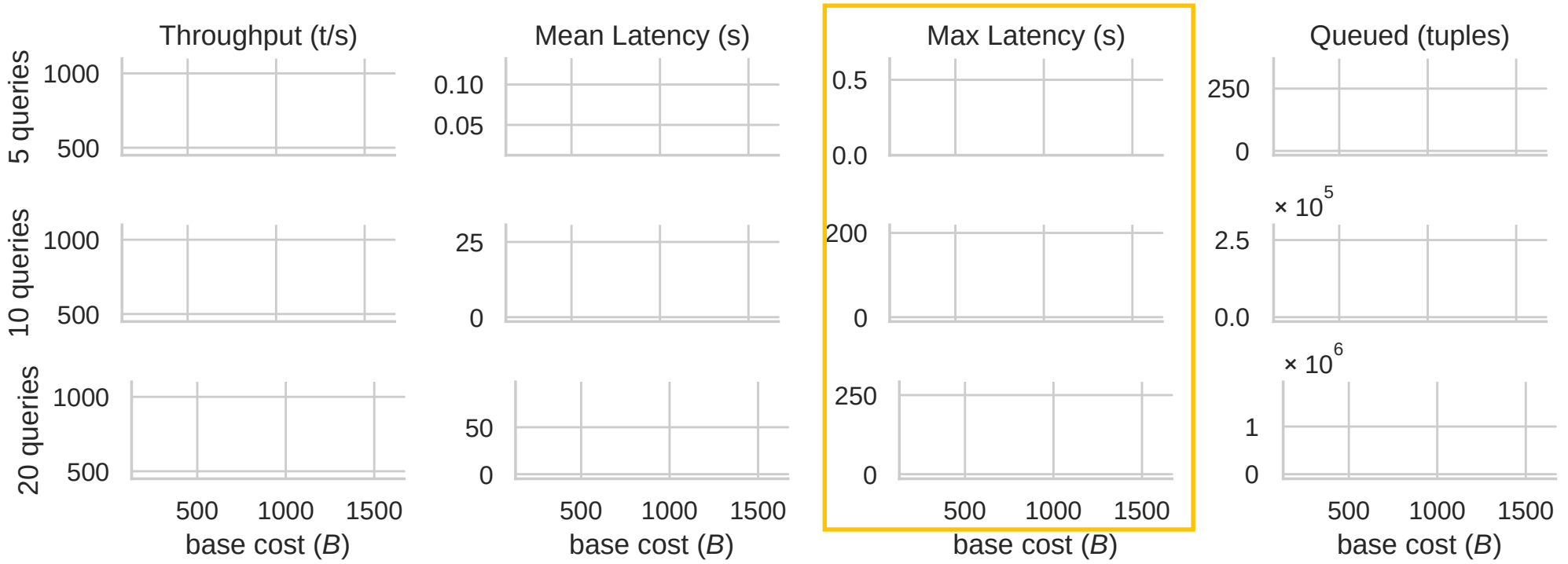
Evaluation

1. **Performance comparison** of dedicated threads (OS) vs Haren policies.
2. **Scheduling overhead** evaluation.
3. **Multi-Class** scheduling.

Evaluation setup:

- Queries → **chains of operators**.
- **Varying cost** and **selectivity** for each query.
- **Varying parallelism** (#queries).
- **Odroid-XU4** devices.
  - Samsung Exynos5422 Cortex-A15 2Ghz and Cortex-A7 Octa core CPU, 2 GB RAM
  - Resource constrained → Custom scheduling even more important.
- Java Haren implementation.
  - Integrated with the lightweight Liebre SPE (https://github.com/vincenzo-gulisano/Liebre)

Evaluation 1: Performance Comparison



CPU, Memory and more in the paper…

OS | HR | FCFS | Chain — Haren

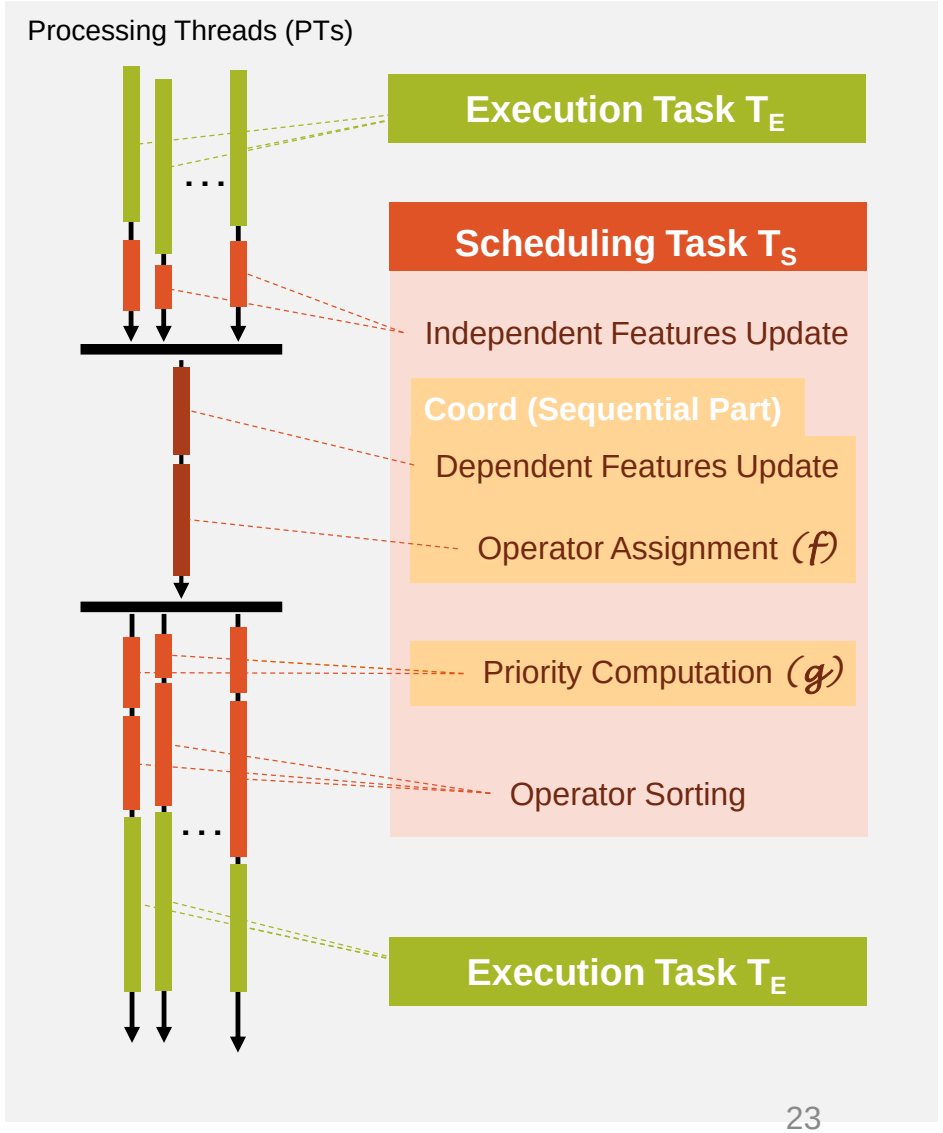
Operating System Scheduling (Dedicated Threads)
Unaware of streaming goals

Highest Rate
Optimize mean latency

First-Come-First-Serve
Optimize maximum latency

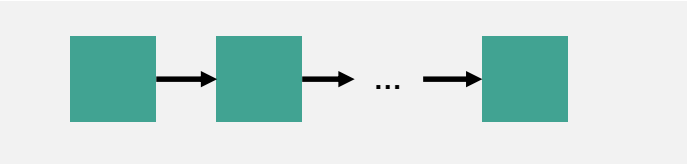
Chain
Optimize total queue sizes

Evaluation 2: Scheduling Overheads

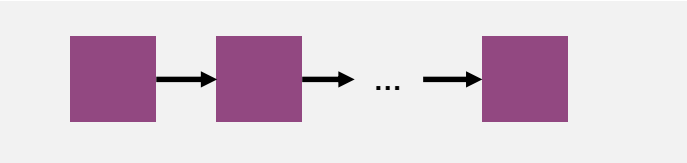


Evaluation 3: Multi-Class Scheduling

3 High Priority Queries



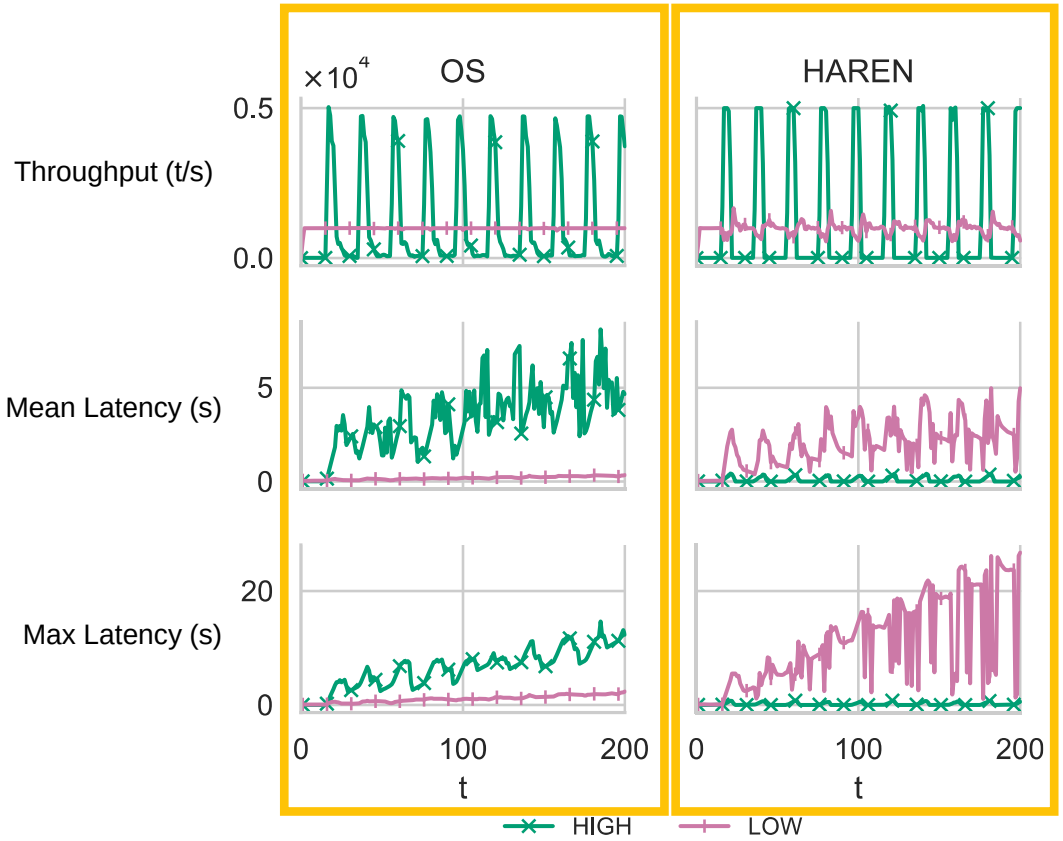
10 Low Priority Queries



Haren Scheduling Policy

1. ***Prioritize*** High Queries over Low Queries
2. Optimize ***Max Latency*** for High Queries
3. Optimize ***Mean*** **Latency** for Low queries

More graphs in the paper!



Conclusions

- **Haren** is an all-purpose **framework** for **scheduling in streaming**.
- Easy definition of **ad-hoc thread scheduling policies**.
- Expressive and efficient, **can outperform dedicated threads approach**.
- **Parallelizes** scheduling computations.

Dimitris Palyvos-Giannas
palyvos@chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY

Distributed Computing and Systems
Chalmers university of technology