LSDS
Large-Scale Data & Systems Group

Imperial College London

SABER The design of a hybrid stream processing system for heterogeneous servers

Alexandros Koliousis

a.koliousis@imperial.ac.uk

Joint work with Matthias Weidlich, Raul Castro Fernandez, Alexander L. Wolf, Paolo Costa, Peter Pietzuch and, more recently, George Theodorakis, Panos Garefalakis and Holger Pirk

Large-Scale Data & Systems Group
Department of Computing, Imperial College London
http://lsds.doc.ic.ac.uk

Streams of Data Everywhere

Many new data sources are now available:

Linear access patterns make data processing a streaming problem

High-Throughput Low-Latency Analytics

Facebook Insights

9GB
of page metrics/s
In less than 10 s

Google Zeitgeist

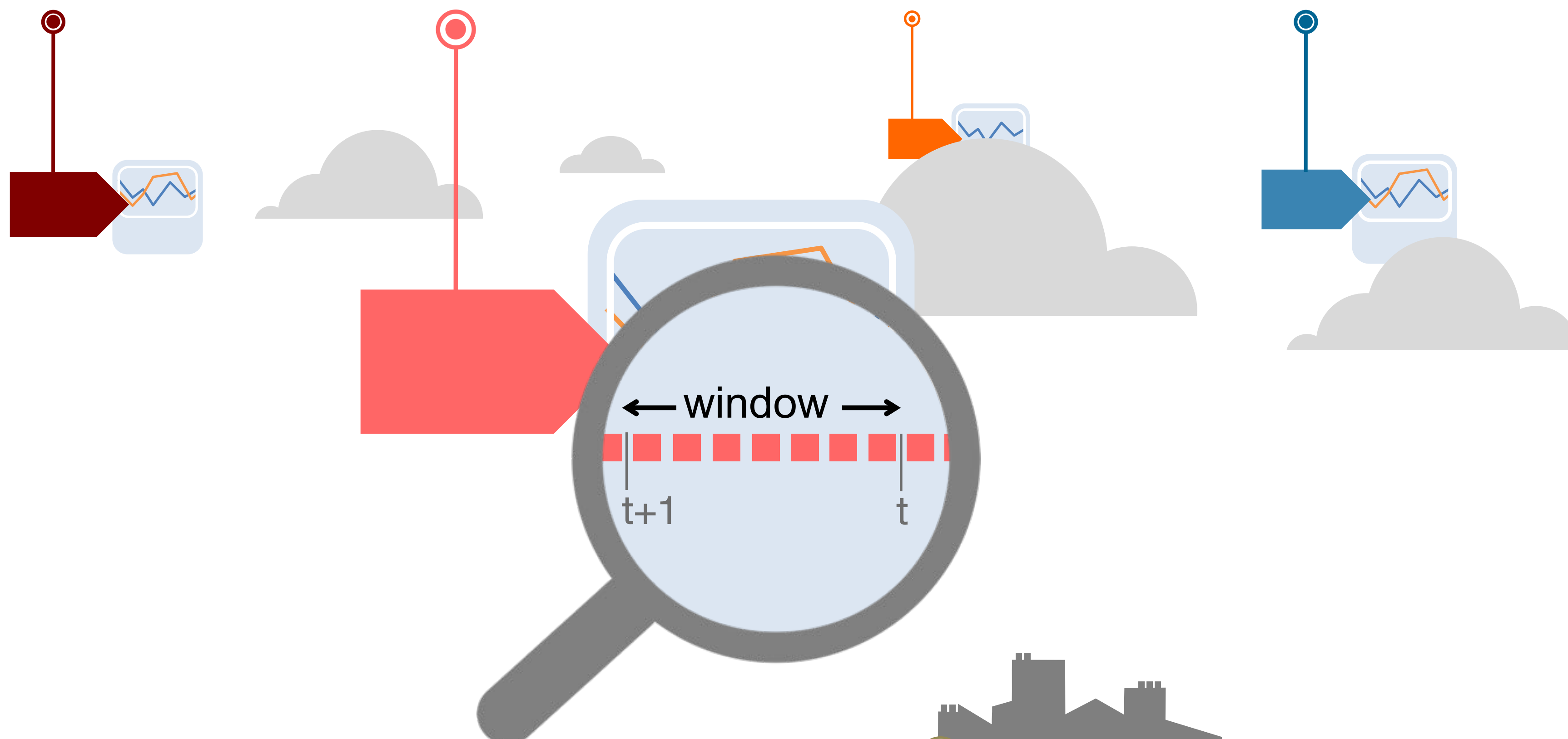
40K
user queries/s
Within ms

Feedzai

40K
card trans/s
In 25 ms

NovaSparks

150M
stock options/s
In less than 1 ms



Algorithmic Complexity Increases

T1

T2

T3

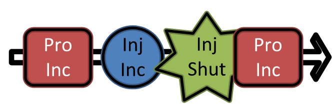
Topic-based filtering

T1(a, b, c)

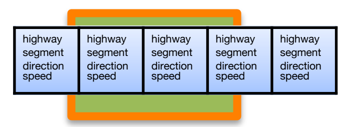
T2(c, d, e)

T3(g, i, h)

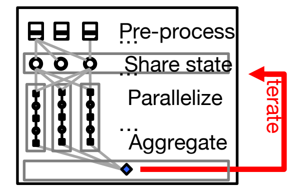
Content-based filtering



Complex pattern matching



Stream queries



Online machine learning, data mining

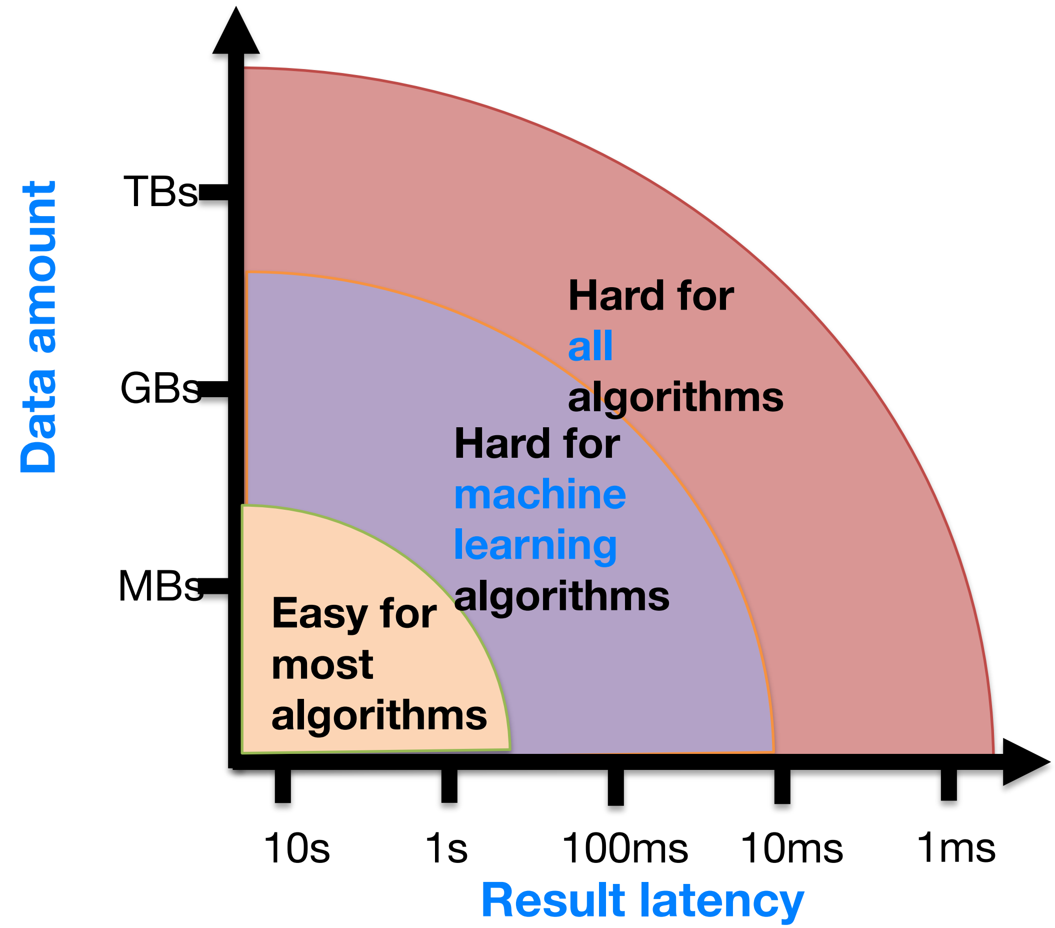
Publish/Subscribe

Complex Event Processing (CEP)

Stream processing

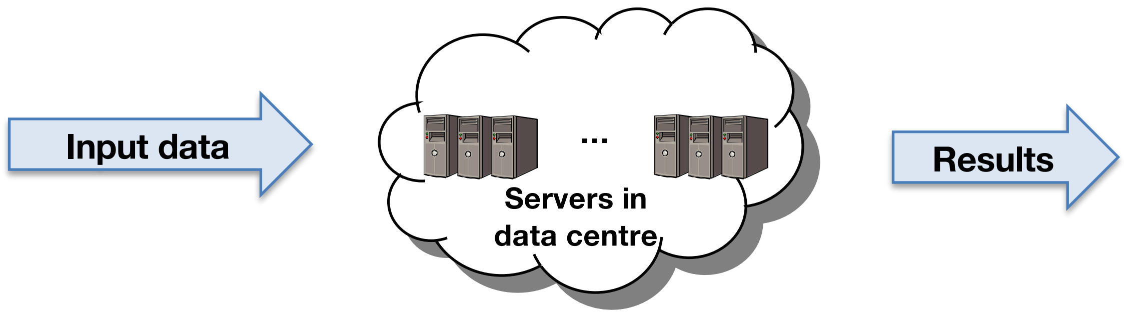
Design Space for Data-Intensive Systems

Tension between performance & algorithmic complexity



Scale Out in Data Centres

Task *vs* Data Parallelism



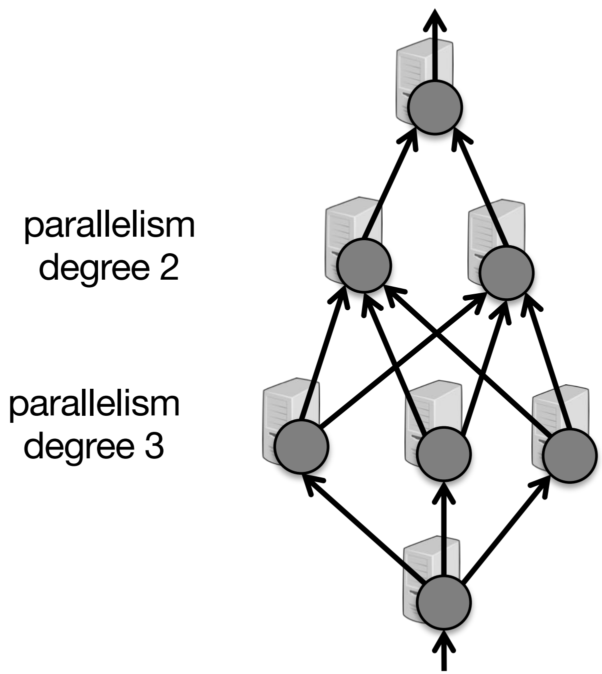
select highway, segment, direction, AVG(speed)
from Vehicles[**range** 5 seconds **slide** 1 second]
group by highway, segment, direction
having avg < 40

select highway, segment, direction, AVG(speed)
from Vehicles[**range** 5 seconds **slide** 1 second]
group by highway, segment, direction
having avg < 40

Task parallelism:
Multiple data processing jobs

Data parallelism:
Single data processing job

Distributed Dataflow Systems



Idea:
Execute **data-parallel tasks**
on cluster nodes

Tasks organised as **dataflow graph**

Almost all big data systems do this:

- Apache Hadoop, Apache Spark, Apache Storm, Apache Flink, Google TensorFlow, ...

“Nobody Ever Got Fired for Using a Hadoop Cluster” [HotCDP’12]

Or Flink or Spark ;)

2012 study of **MapReduce** workloads

- Microsoft: median job size **< 14 GB**
- Yahoo: median job size **< 12.5 GB**
- Facebook: 90% of jobs **< 100 GB**

⇨ The size of the workloads has changed, but so has the size/price of **memory**!

Many data-intensive jobs easily fit into **memory**

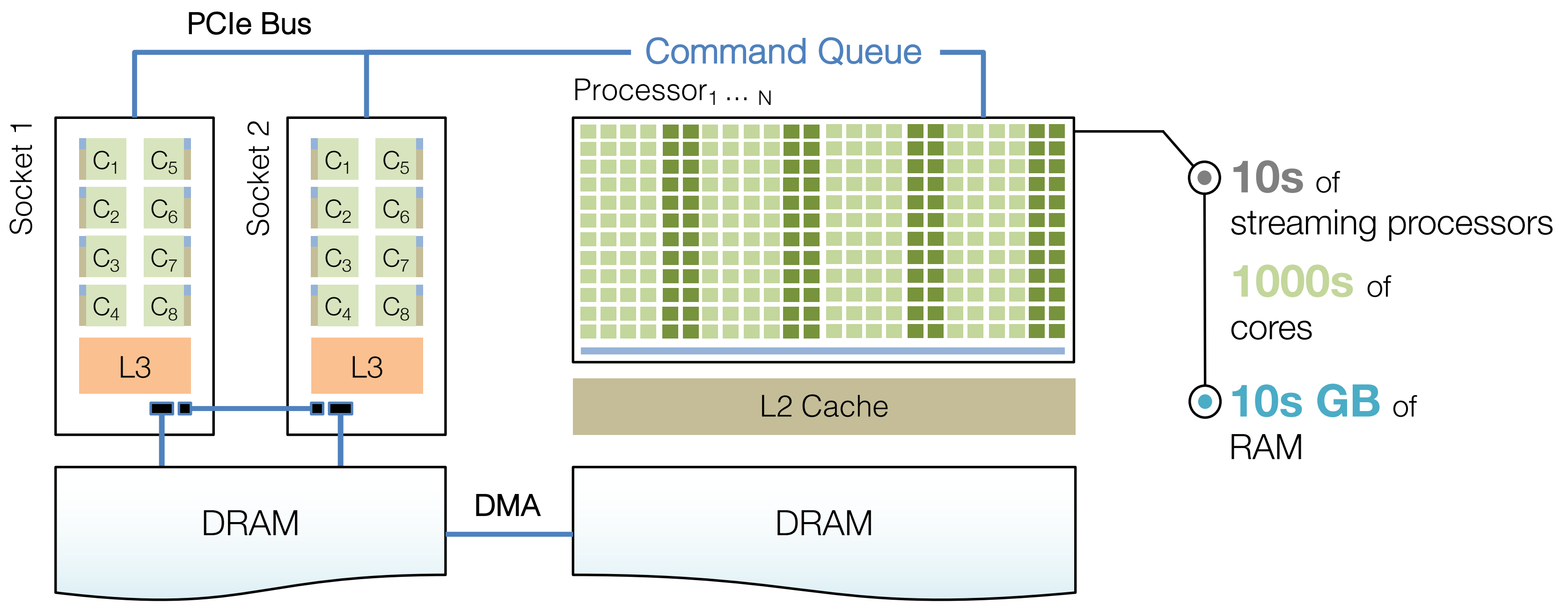
It’s **expensive** to scale-out in terms of hardware and engineering!

☛ **In many cases a single server** is cheaper/more efficient than a **cluster**

Exploit Single-Node Heterogeneous Hardware

Servers with CPUs and GPUs now common

- 10x higher linear memory access throughput
- Limited data transfer throughput



Use **both** CPU & GPU resources for stream processing

With Well-Defined High-Level Queries

CQL: SQL-based declarative language for continuous queries [Arasu *et al.*, VLDBJ'06]

Credit card fraud detection example:

- Find attempts to use same card in different regions within 5-min window

<\> CQL offers correct window semantics

Self-join

select distinct	W.cid
from	Payments [**range** 300 seconds] **as** W,
	Payments [**partition-by** 1 row] **as** L
where	W.cid = L.cid **and** W.region != L.region

Challenges & Contributions

1. How to parallelise sliding-window queries across CPU and GPU?

Decouple query semantics from system parameters

2. When to use CPU or GPU for a CQL operator?

Hybrid processing: offload tasks to both CPU and GPU

3. How to reduce GPU data movement costs?

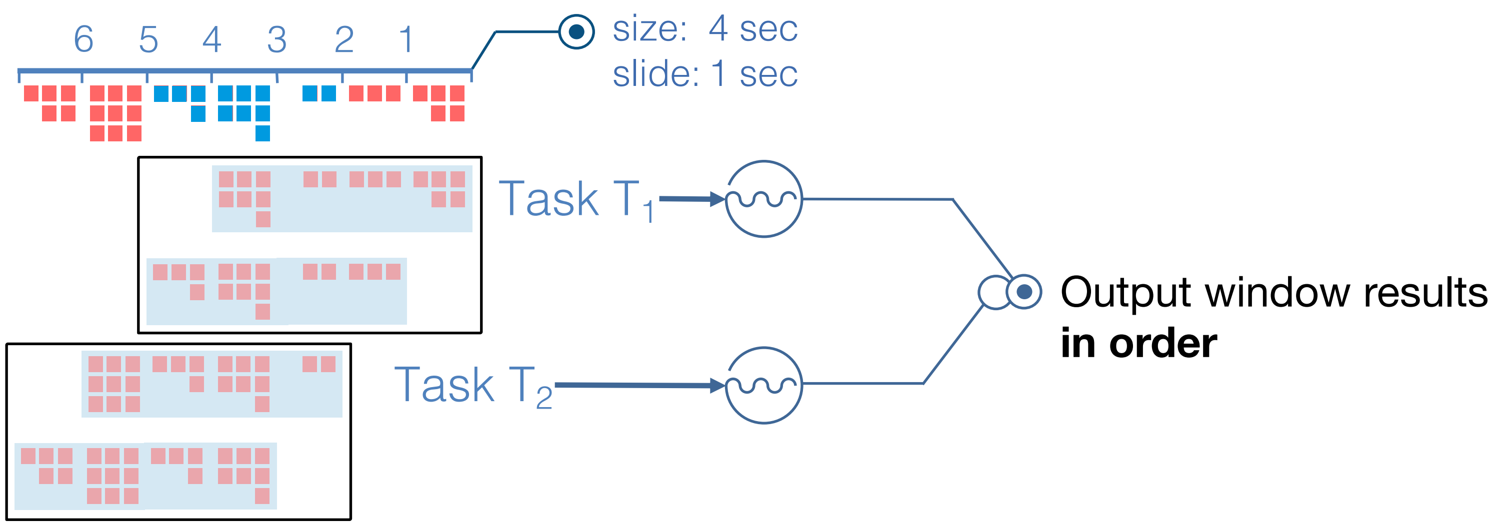
Amortise data movement delays with deep pipelining

Details omitted

How to Parallelise Window Computation?

Problem: Window semantics affect system throughput and latency

– Pick task size based on window size?

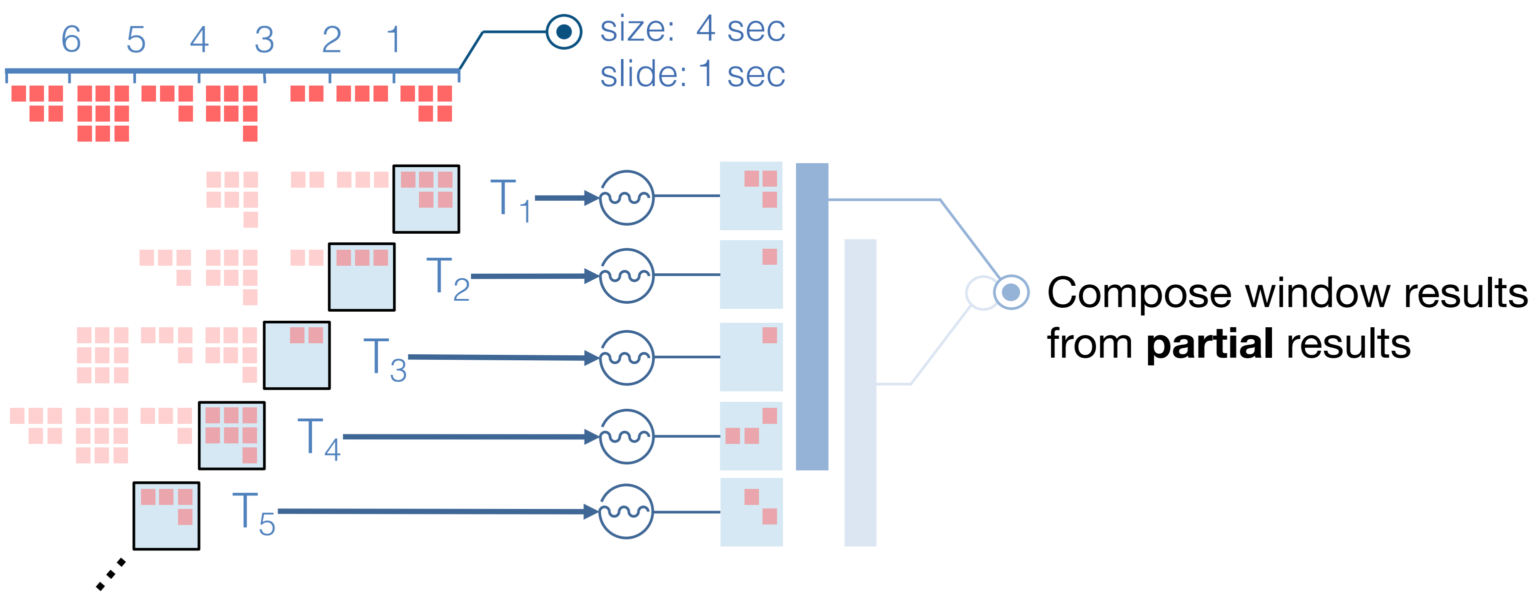


Window-based parallelism results in **redundant** computation

How to Parallelise Window Computation?

Problem: Window semantics affect system throughput and latency

– Pick task size based on window size? On window slide?

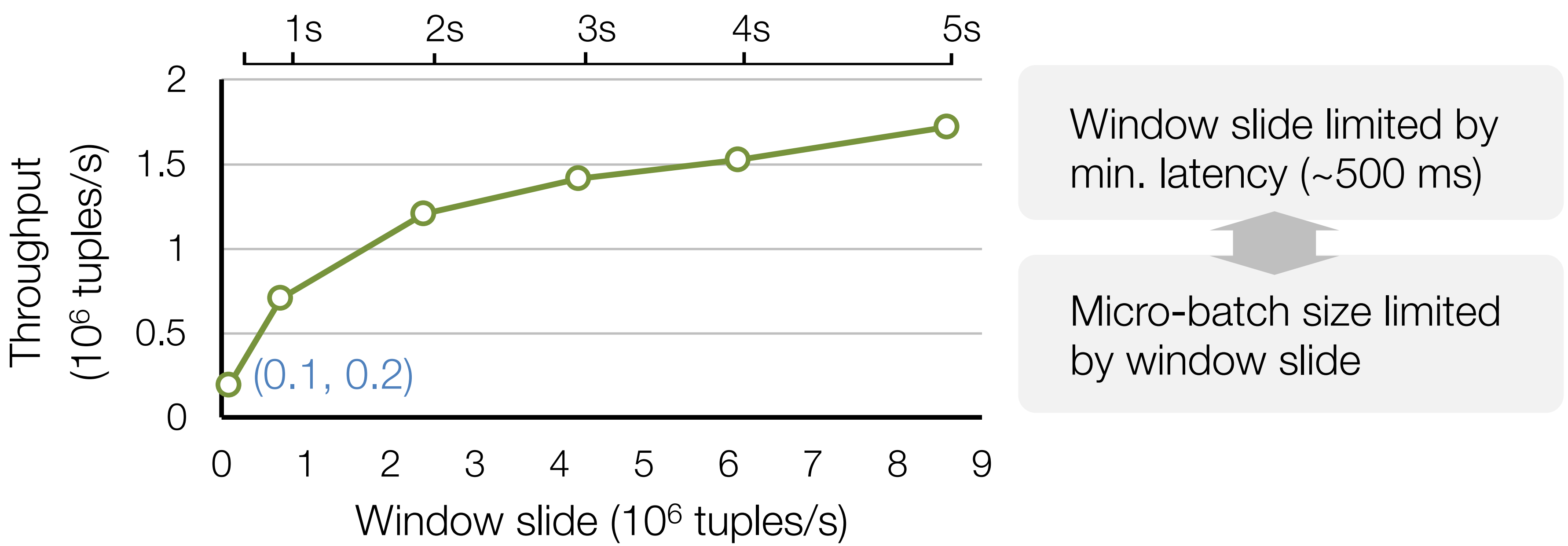


Slide-based parallelism limits GPU parallelism

How to Relate Slides to Tasks?

Avoid coupling throughput/latency of queries to window definition

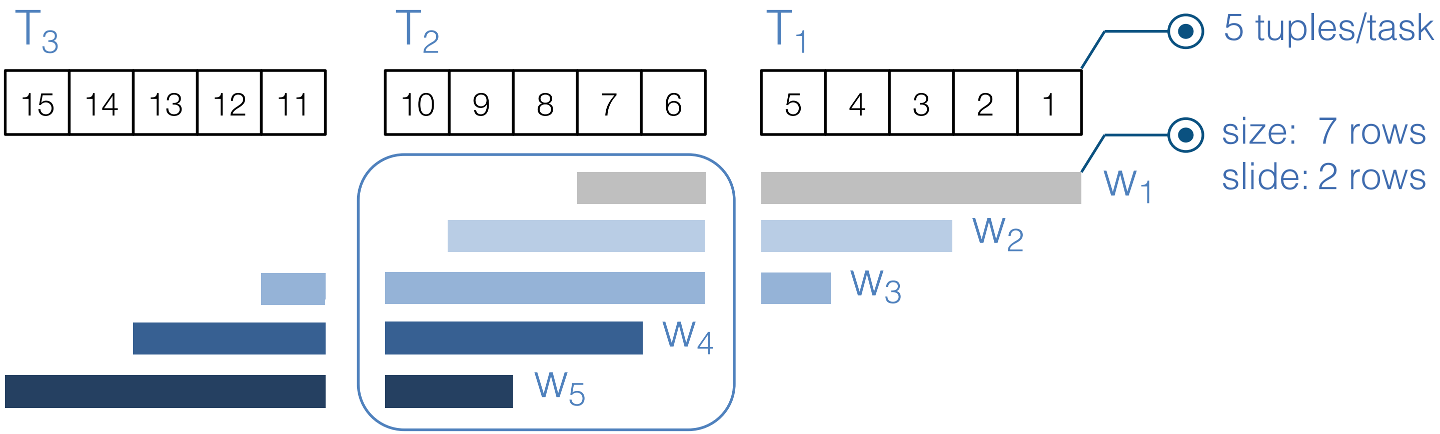
- e.g. Spark imposes lower bound on window slide:



SABER's Window Processing Model

Idea: Decouple task size from window size/slide

- Pick based on underlying hardware features
  - e.g. PCIe throughput



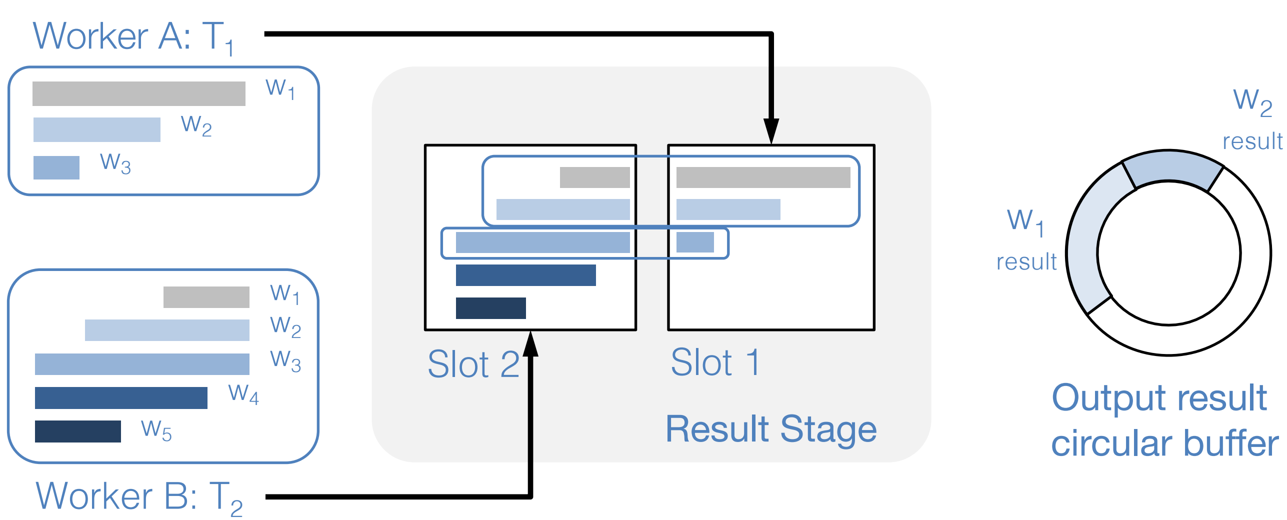
- Task contains one or more window fragments
  - E.g. closing/pending/opening windows in T_2

Merging Window Fragment Results

Idea: Decouple task size from window size/slide

– Assemble window fragment results

– Output them in correct order



Worker A **stores** T_1 **results**, and exits (nothing to forward)

Worker B merges window fragment results and forwards complete windows downstream

Operator Implementations / API

Fragment function, f_f

Processes window fragments

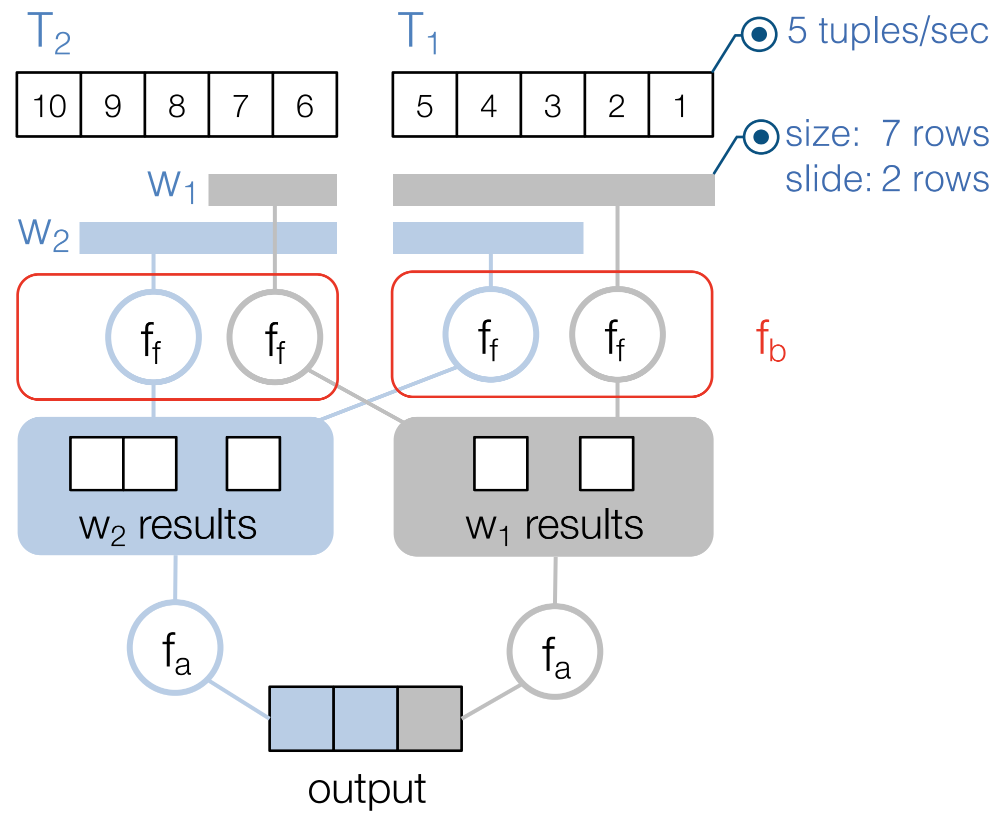
Assembly function, f_a

Merges partial window results

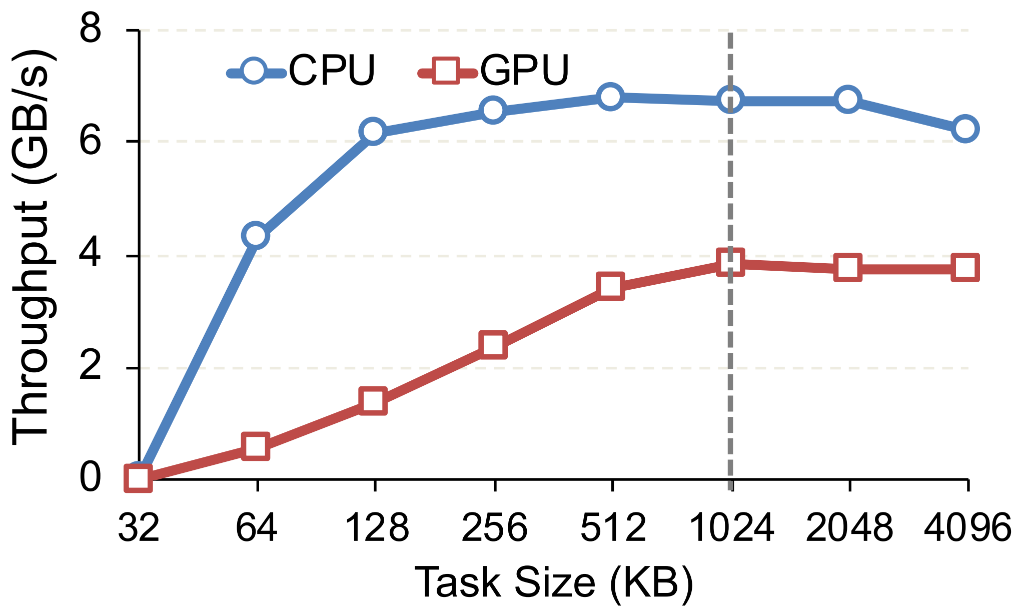
Batch function, f_b

Composes fragment functions within a task

Allows incremental processing

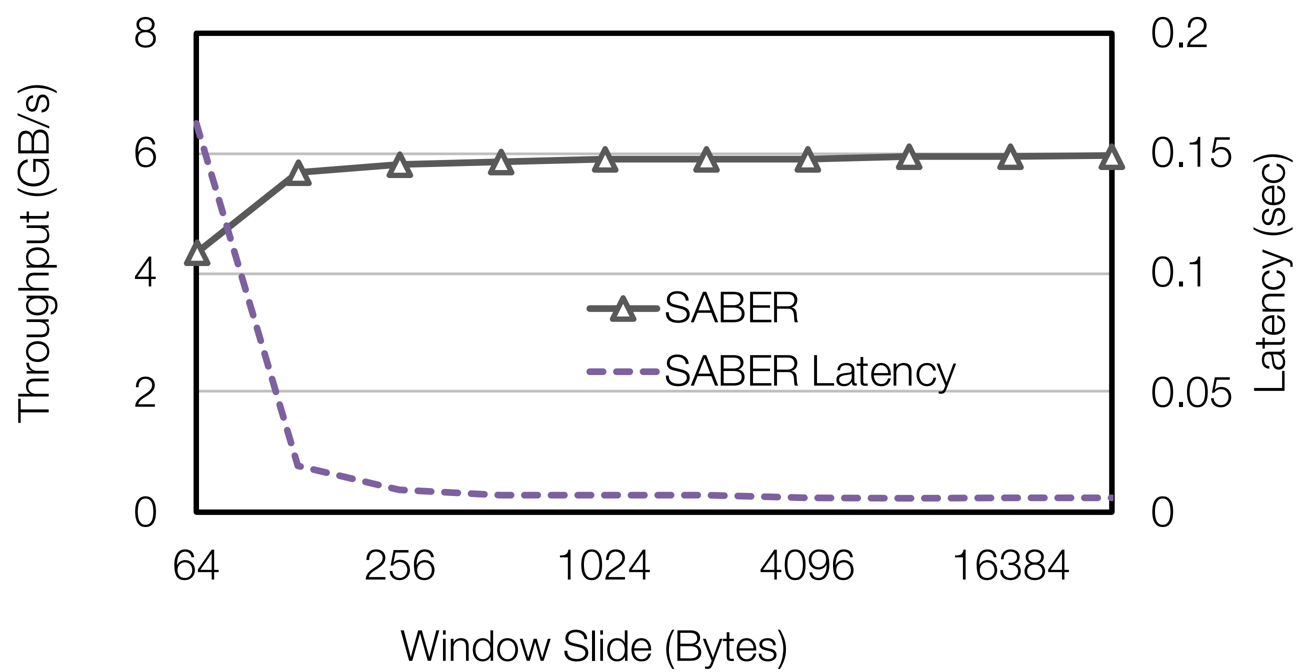


How to Pick the Task Size?



How Does Window Slide Affect Performance?

Performance of window-based queries remains predictable



Aggregation$_{avg}$ [rows 1024, slide x]

Challenges & Contributions

1. How to parallelise sliding-window queries across CPU and GPU?

Decouple query semantics from system parameters

2. When to use CPU or GPU for a CQL operator?

Hybrid processing: offload tasks to both CPU and GPU

3. How to reduce GPU data movement costs?

Amortise data movement delays with deep pipelining

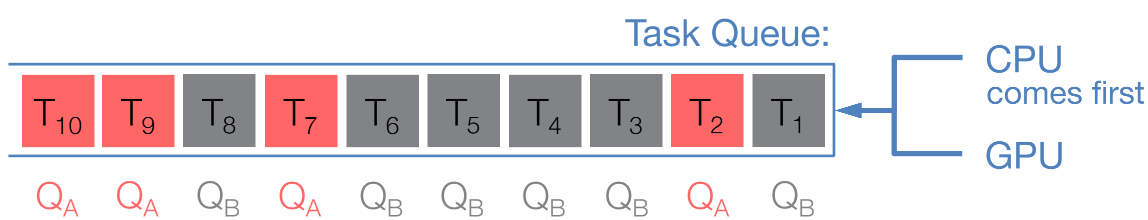
SABER's Hybrid Stream Processing Model

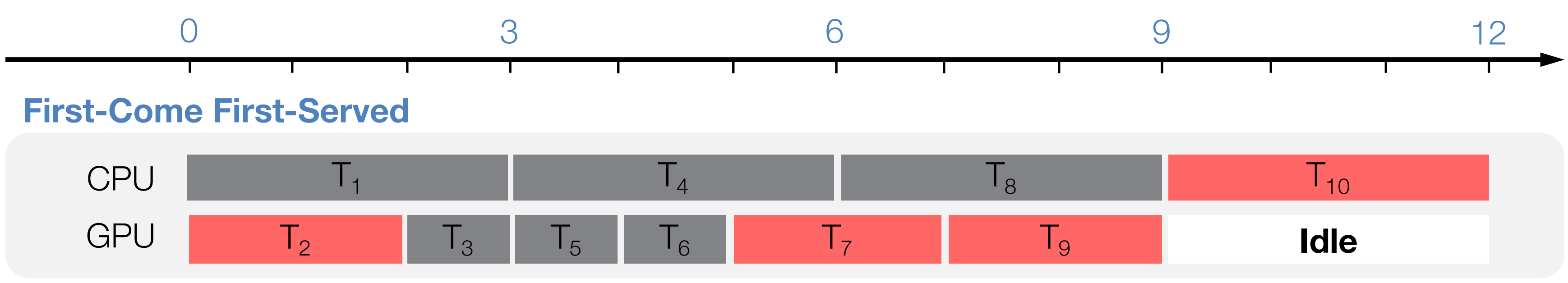
Idea: Enable tasks to run on both processors

– Scheduler assigns tasks to idle processors

Past behavior:

	CPU	GPU
Q_A	3 ms	**2 ms**
Q_B	3 ms	**1 ms**





FCFS **ignores** effectiveness of processor for given task

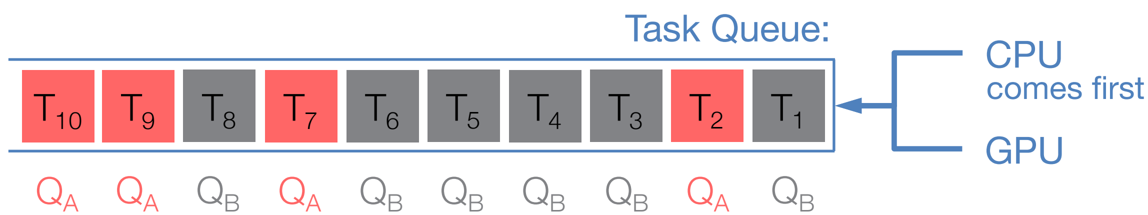
Heterogeneous Look-Ahead Scheduler (HLS)

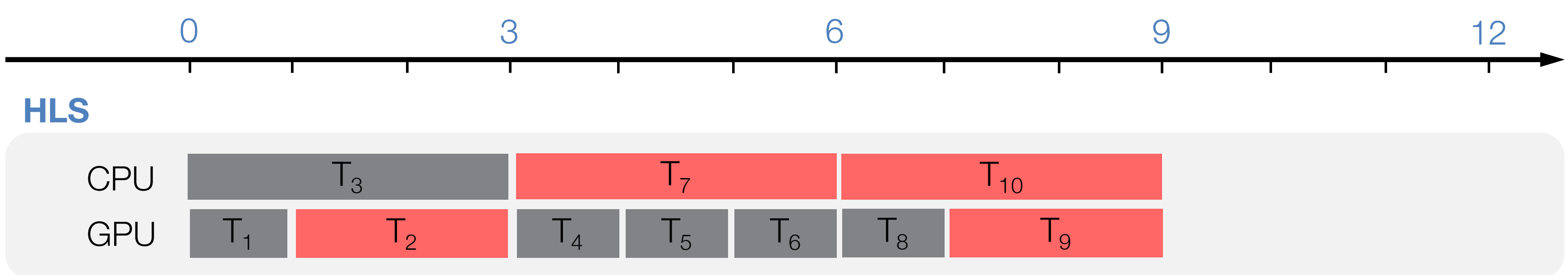
Idea: Idle processor skips tasks that could be executed faster by another processor

- Decision based on observed query task throughput

Past behavior:

	CPU	GPU
Q_A	3 ms	**2 ms**
Q_B	3 ms	**1 ms**

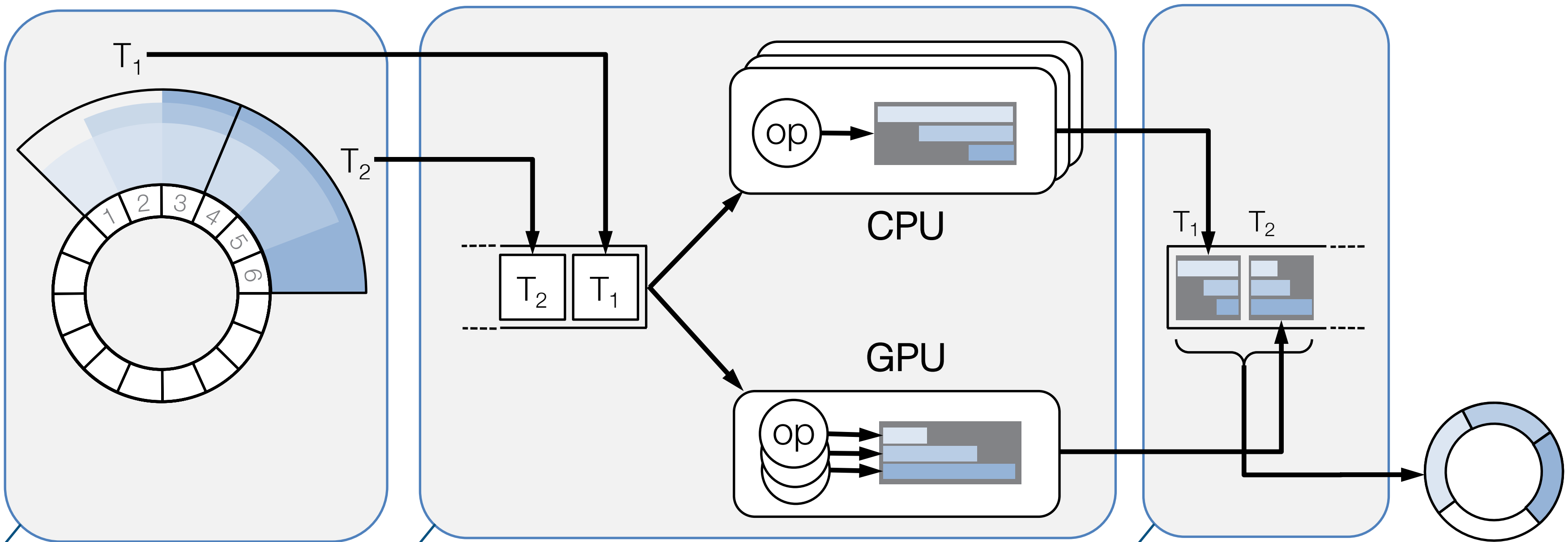




HLS **fully** utilises processors

The SABER Architecture

Java
15K LOC
C & OpenCL
4K LOC



Dispatching stage

Dispatch
fixed-size tasks

Scheduling & execution stage

Dequeue tasks
based on HLS

Result stage

Merge & forward partial
window results

Evaluation: Set-up & Workloads

PCIe 3.0 x16

10 Gbps
NIC

Intel Xeon
2.6 GHz
16 cores **64GB RAM**
Ubuntu Linux 14.04

NVIDIA Quadro
K5200
2,304 cores **8GB RAM**
NVIDIA driver 346.47

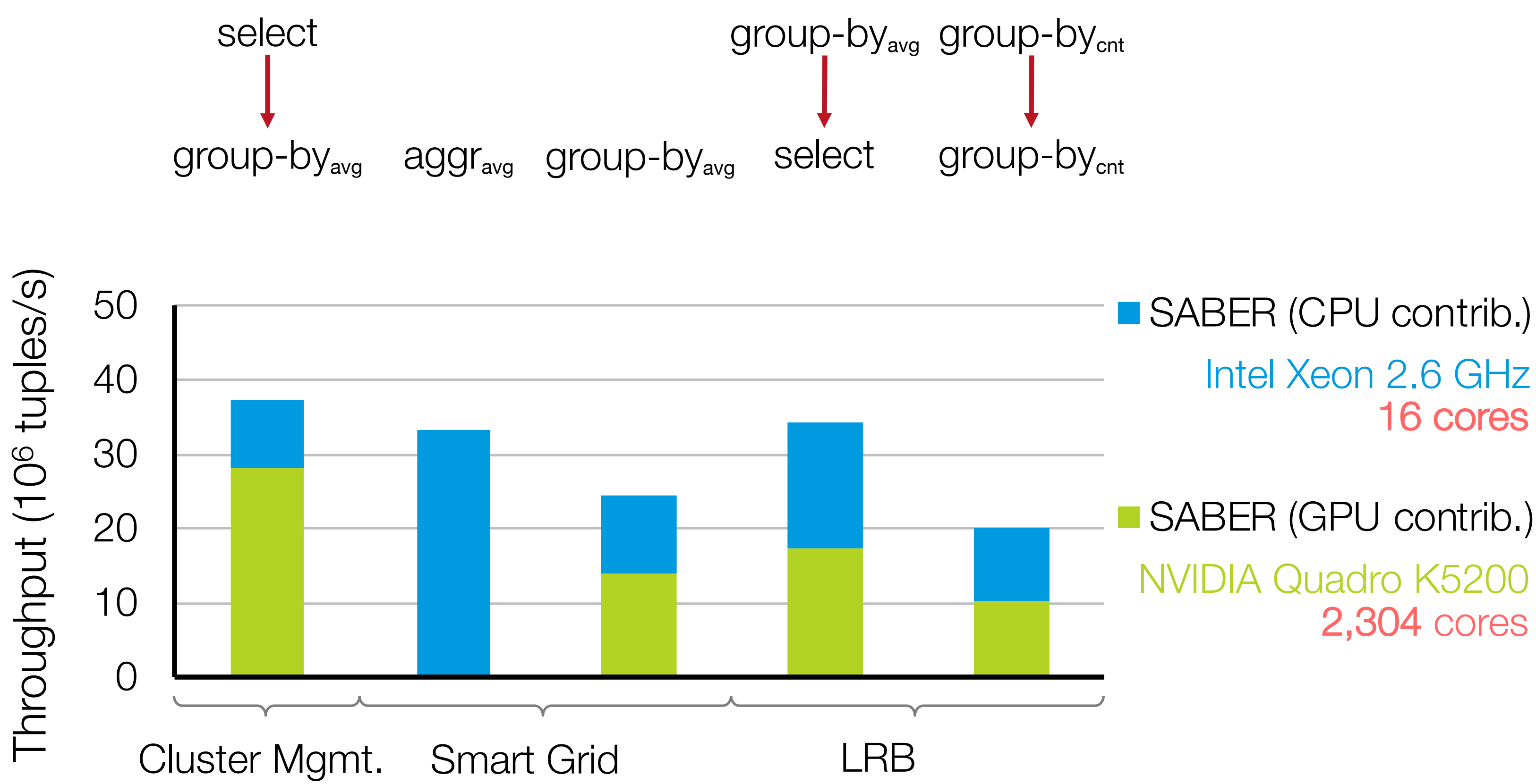
Google Cluster Data
144M jobs events from Google infrastructure

SmartGrid Measurements
974M plug measurements from houses

Linear Road Benchmark
11M car positions and speed on highway

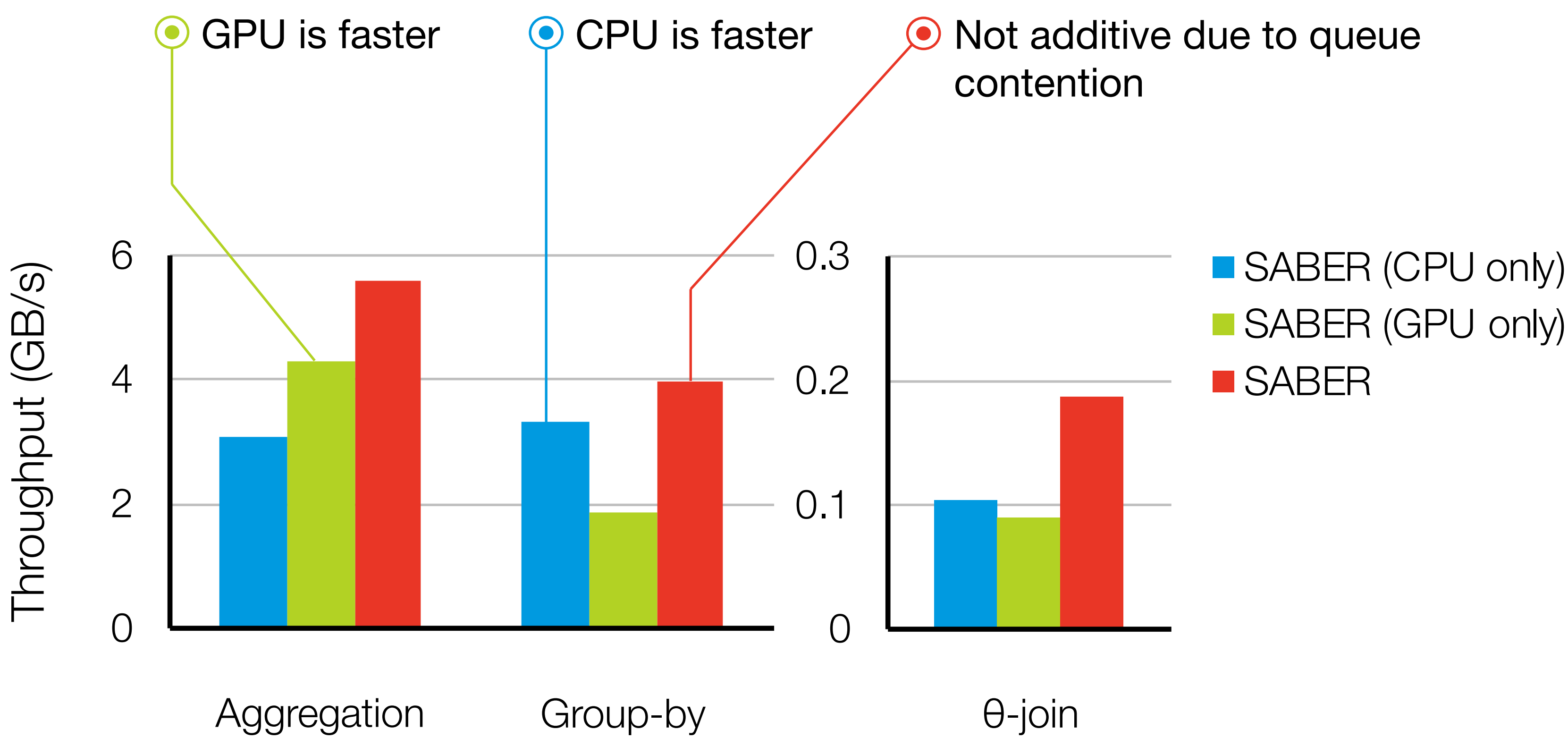
Is Hybrid Stream Processing Effective?

Different queries result in different CPU:GPU processing **split** that is hard to predict offline



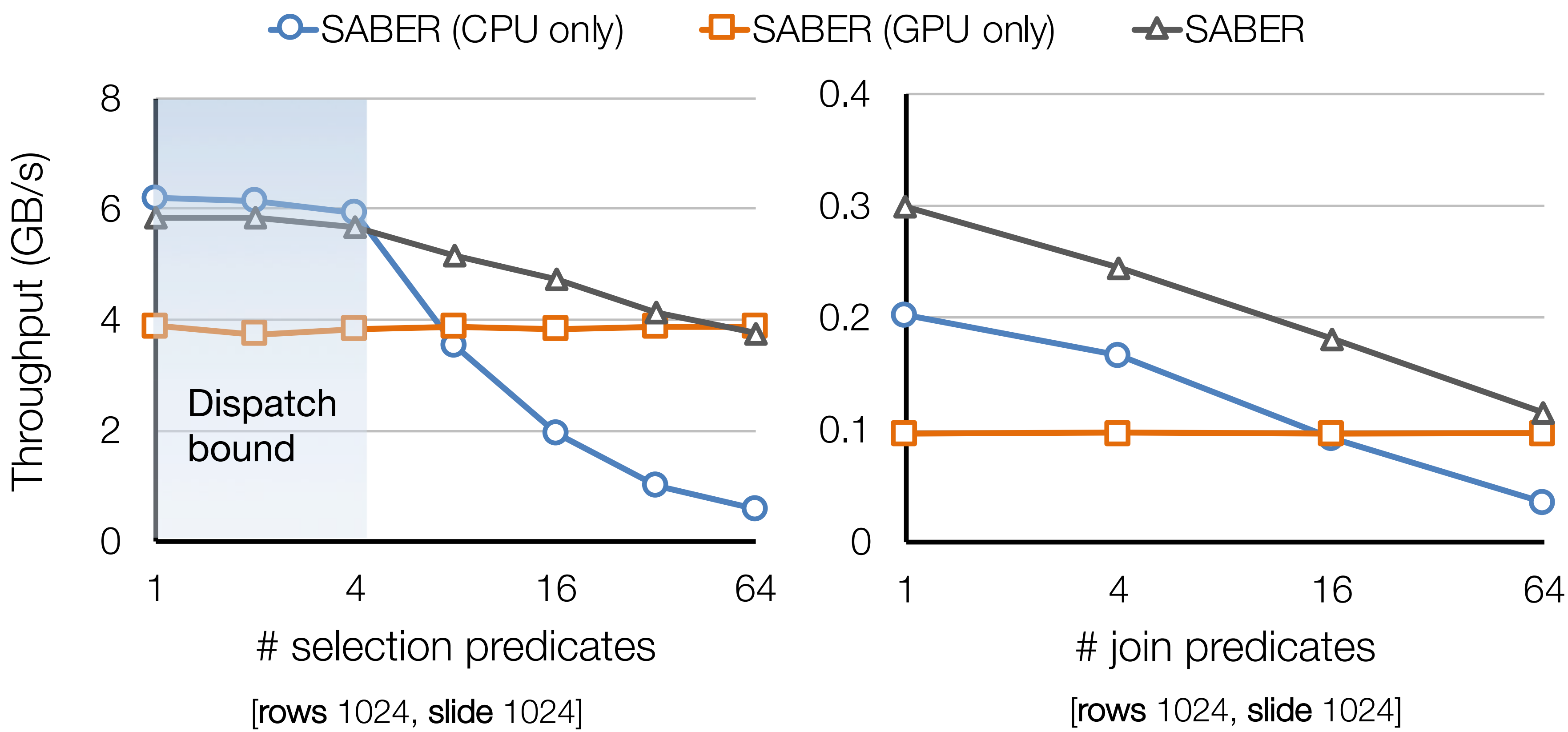
Is Hybrid Stream Processing Effective?

Aggregate throughput of CPU and GPU **always higher** than its counterparts



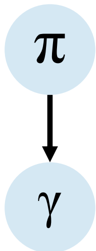
What is the CPU/GPU Trade-Off?

Hybrid processing model benefits from GPU ability to process complex predicates fast



Is Heterogeneous Look-Ahead Scheduling Effective?

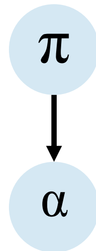
W_1



[rows 1024, slide 512]

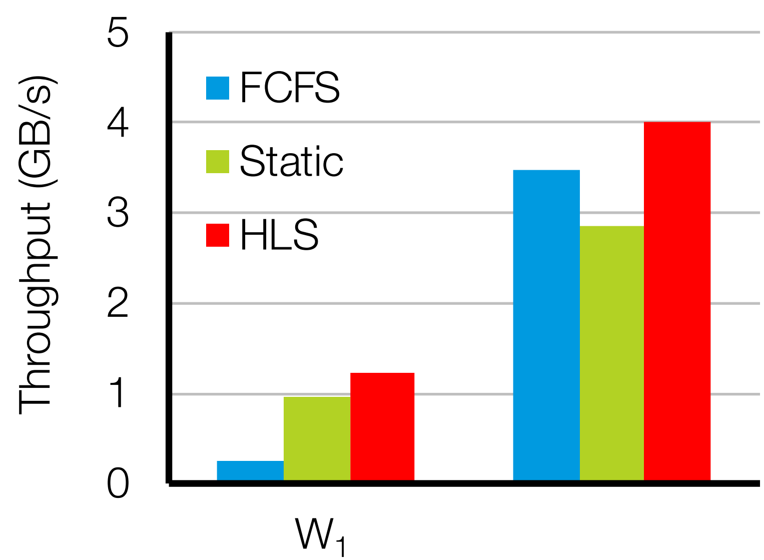
	CPU	GPU
π		5x
γ	6x	

W_2



[rows 1024, slide 1024]

	CPU	GPU
π		1.5x
α	1.5x	



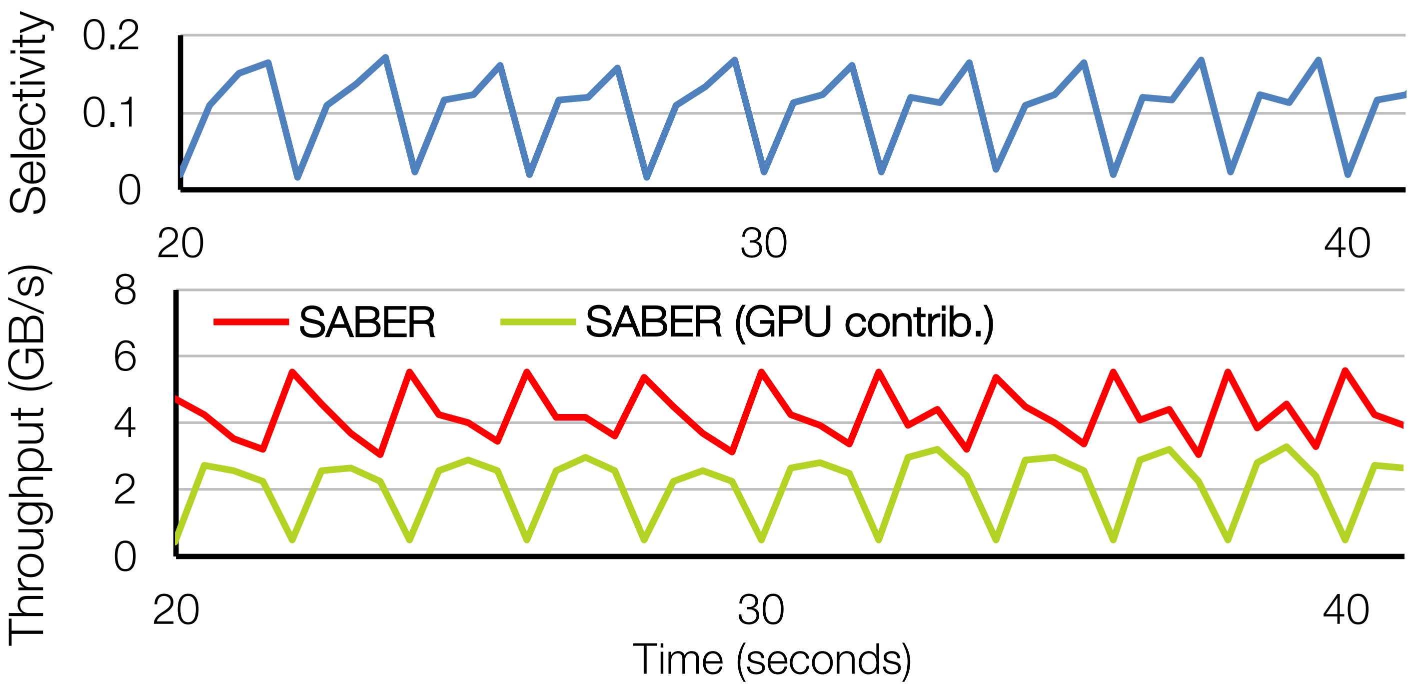
W_1 benefits from static scheduling but HLS fully utilises GPU:

– GPU also runs ~%1 of of γ tasks

Is Heterogeneous Look-Ahead Scheduling Adaptive?

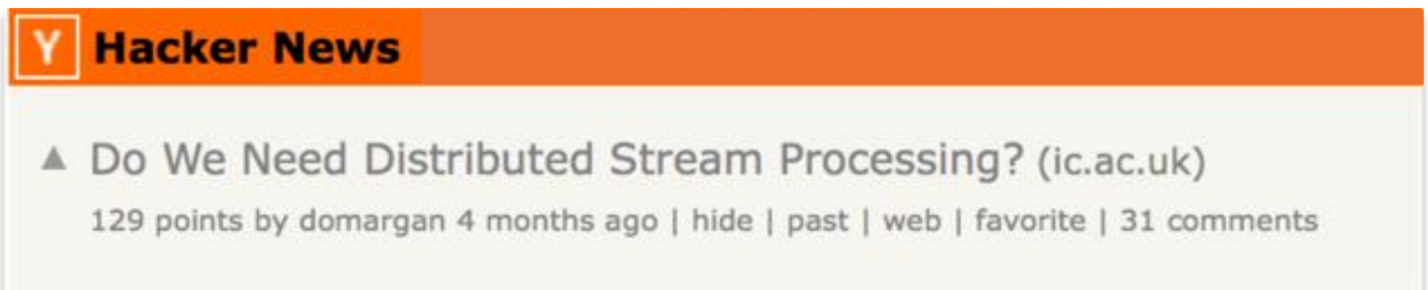
HLS periodically uses idle, non-preferred processor to run tasks to update query task throughput

Example: higher selectivity, more predicates evaluated, GPU is preferred



H/W-Oblivious Tasks, H/W-Conscious Operators

To begin with, can SABER compete with popular distributed stream processing systems?



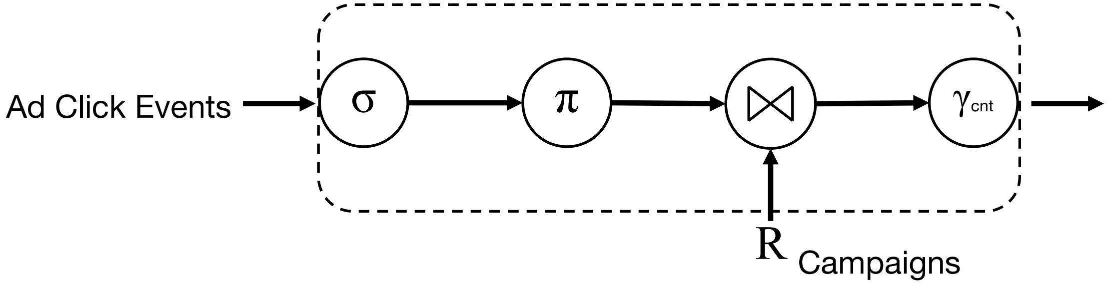
https://lsds.doc.ic.ac.uk/blog/do-we-need-distributed-stream-processing

Enter Yahoo! Stream Benchmark

An industry standard (wannabe)

Storm, Flink, Spark, Apex, Drizzle, Diff. Dataflow

Tumbling-window query, bottlenecked by factors other than computation



How many times a campaign has been seen in a tumbling window

Systems Compared

Apache Flink (1.3.2)

Apache Spark Streaming (2.4.0)

SABER (1.0), without GPU support

StreamBox: a single-server system with emphasis on out-of-order processing

Experimental Setup

6 servers (1 master and 5 slaves): 2 Intel Xeon E5-2660 v3 2.60 GHz CPUs

- 20 physical CPU cores
- **25 MB LLC**

32 GB of memory

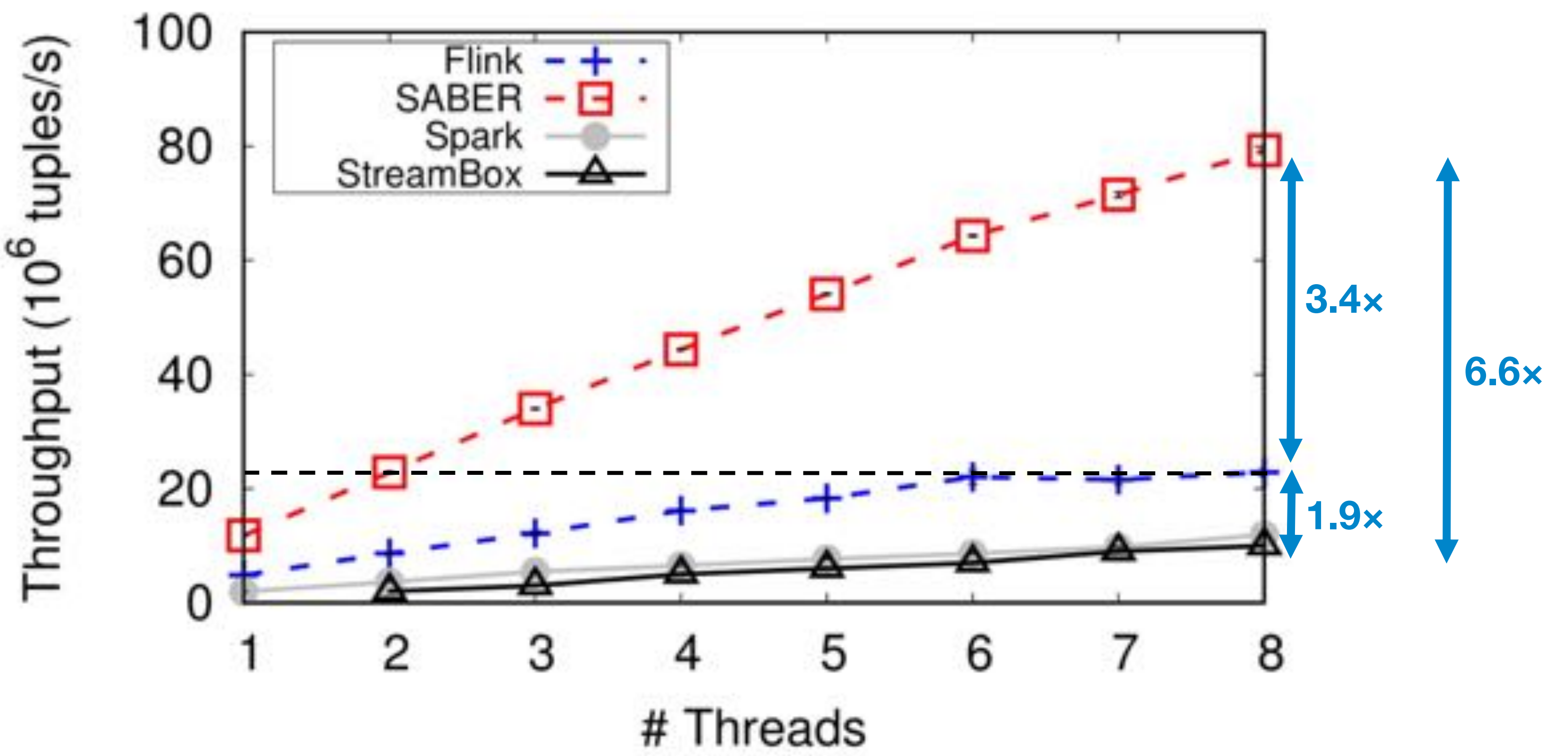
10 GigE connection between the nodes

In-memory generation

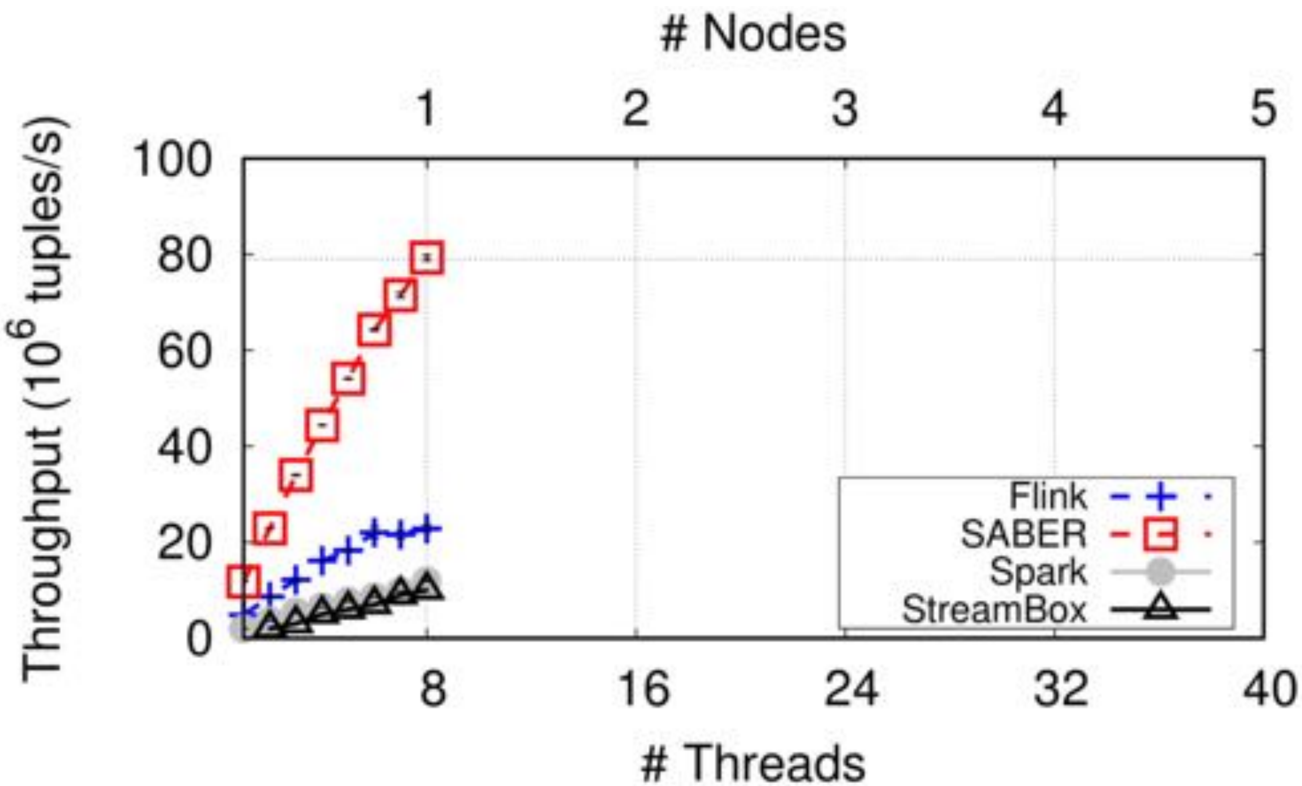
8 cores per node

On a Single Server…

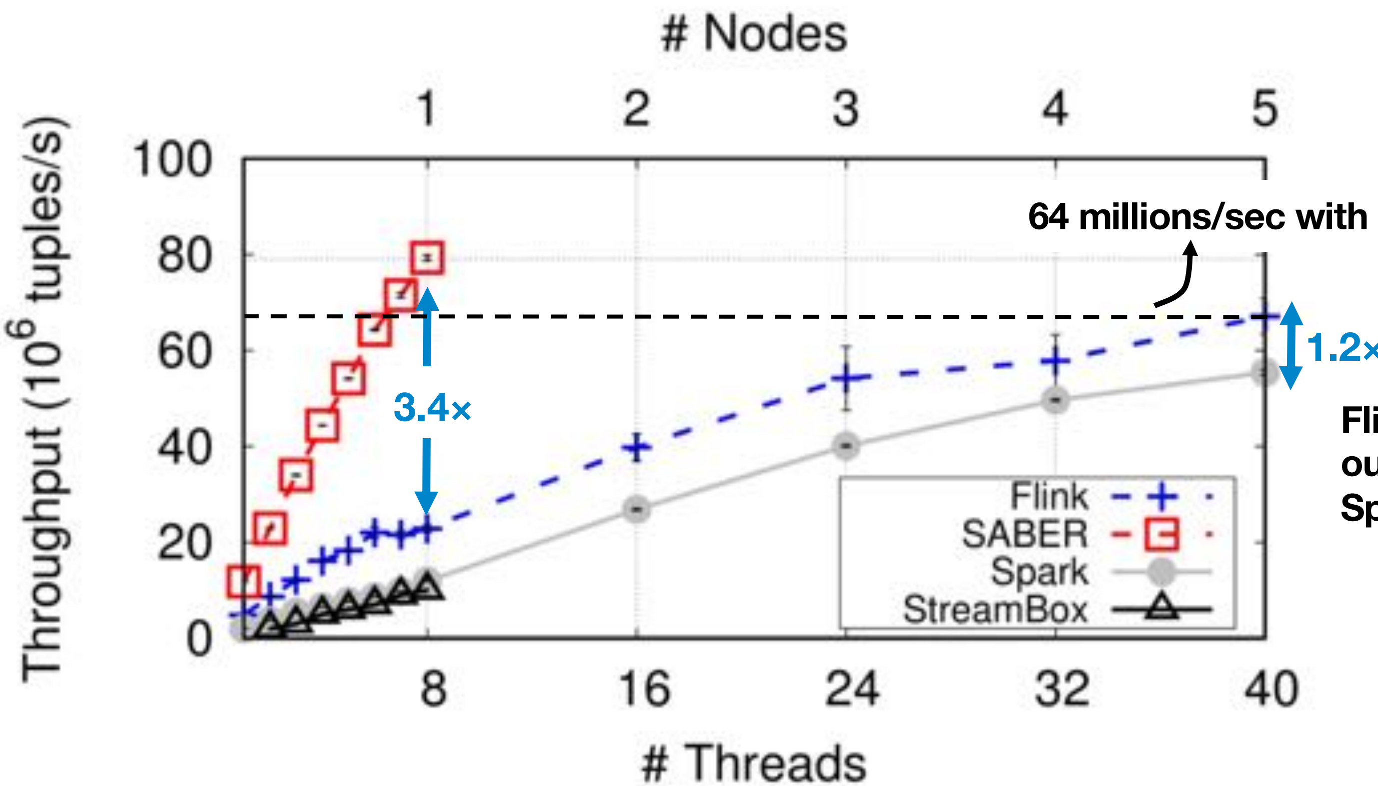
Reduced serialization costs; keeping data in LLC



On Multiple Servers…



On Multiple Servers…



Flink outperforms Spark!

COST [HotOS'15]

	Spark	Flink	SABER	Handwritten C++
Throughput (million tuples/ sec)	**2**	**4.8**	**11.8**	**39**

Do better than LLC? ⇨

Pipeline Strategy [Hyper, VLDB'11]:

- keep data in CPU registers
- as many sequential operations as possible per tuple
- maximize data locality

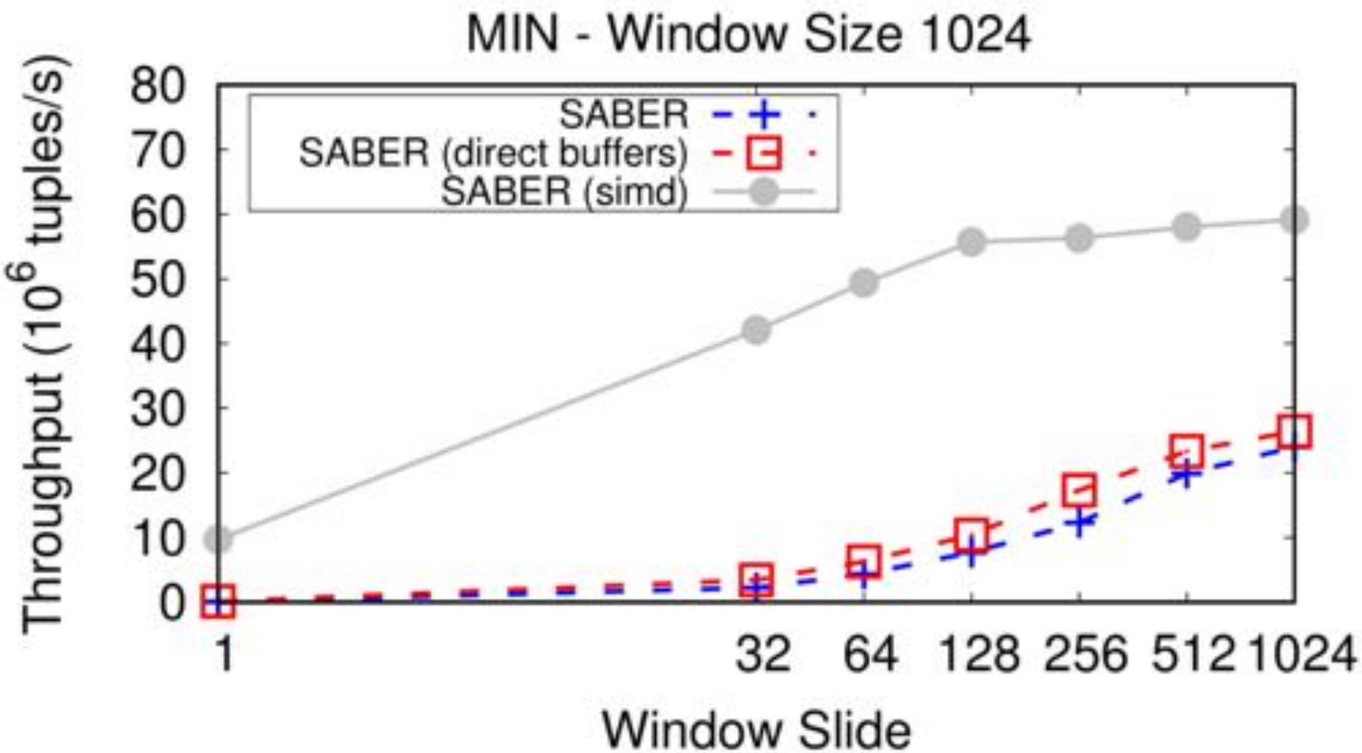
With a compiler-based approach to generate custom code based on a set of hardware-specific optimisations for any given query

H/W-Efficient Streaming Operators

Hammer Slide: Work- and CPU-efficient Streaming Window Aggregation [ADMS'18]

- **Incremental computation** for both **invertible** and **non-invertible** functions
- **Parallel processing** within a slide (**>1**) with SIMD instructions
- Bridge the gap between sliding and tumbling window computation

HammerSlide + SABER



Summary

Window processing model

Decouples query semantics from system parameters

Hybrid stream processing model

Can achieve aggregate throughput of heterogeneous processors

Hybrid Look-ahead Scheduling (HLS)

Allows use of both CPU and GPU opportunistically for arbitrary workloads

Thank you! Any Questions?

Alexandros Koliousis

github.com/lsds/saber