LSDS

Large-Scale Distributed Systems Group

Imperial College
London

SABER Window-Based Hybrid Stream Processing for Heterogeneous Architectures

github.com/lsds/saber

Alexandros Koliousis

a.koliousis@imperial.ac.uk

Joint work with Matthias Weidlich, Raul Castro Fernandez,
Alexander L. Wolf, Paolo Costa & Peter Pietzuch

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

High-Throughput Low-Latency Analytics

Facebook Insights

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

Google Zeitgeist

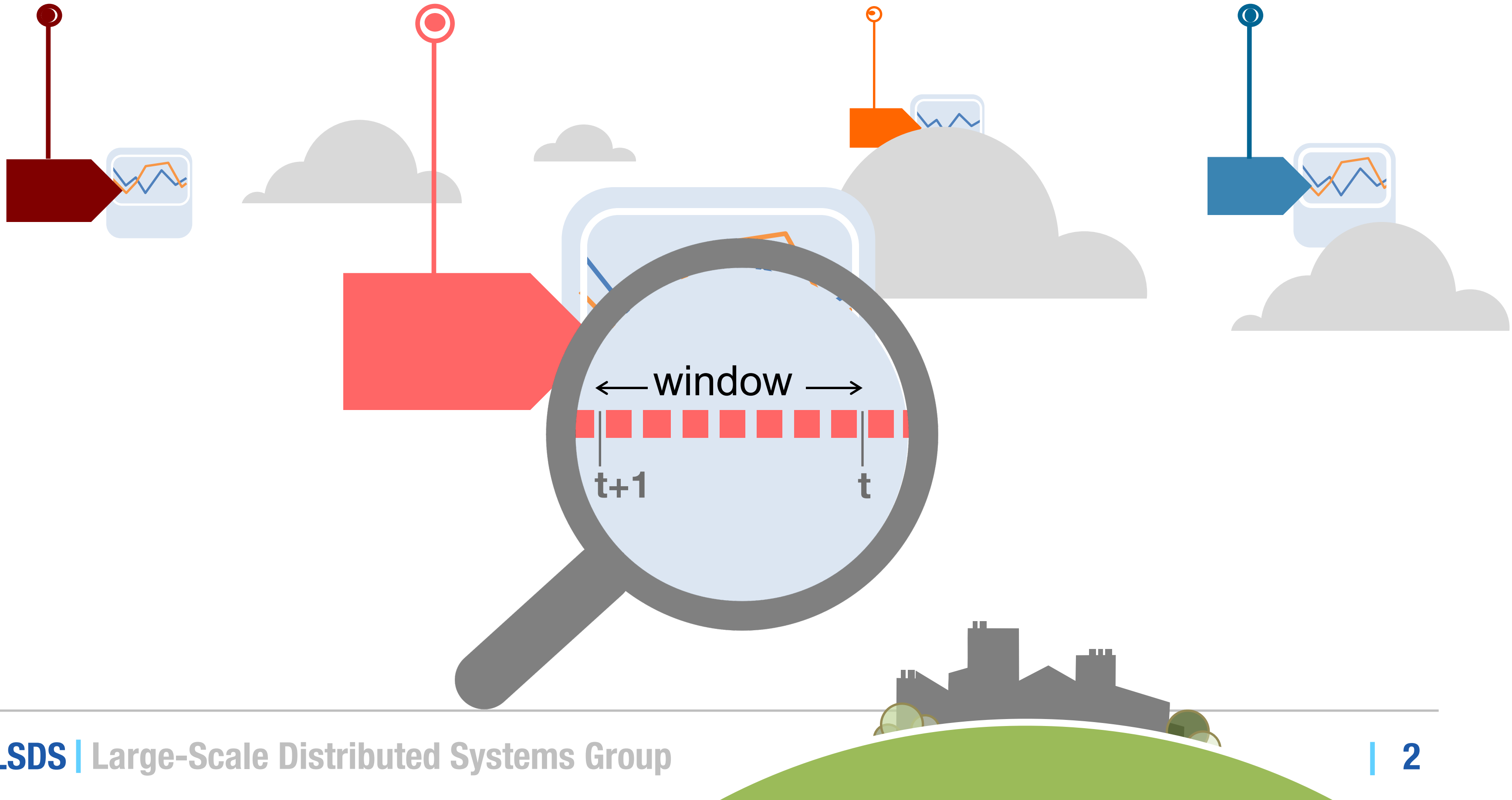
40K
user queries/s
Within ms

Feedzai

40K
card trans/s
In 25 ms

NovaSparks

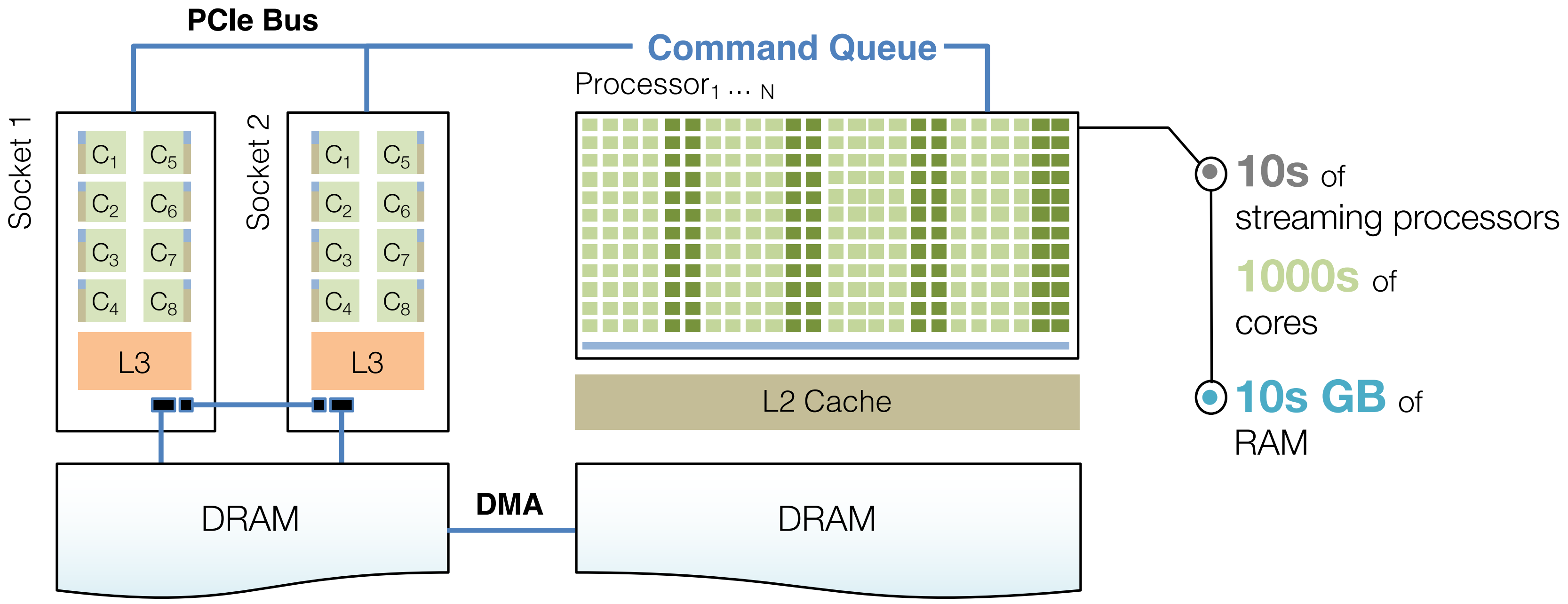
150M
stock options/s
In less than 1 ms



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

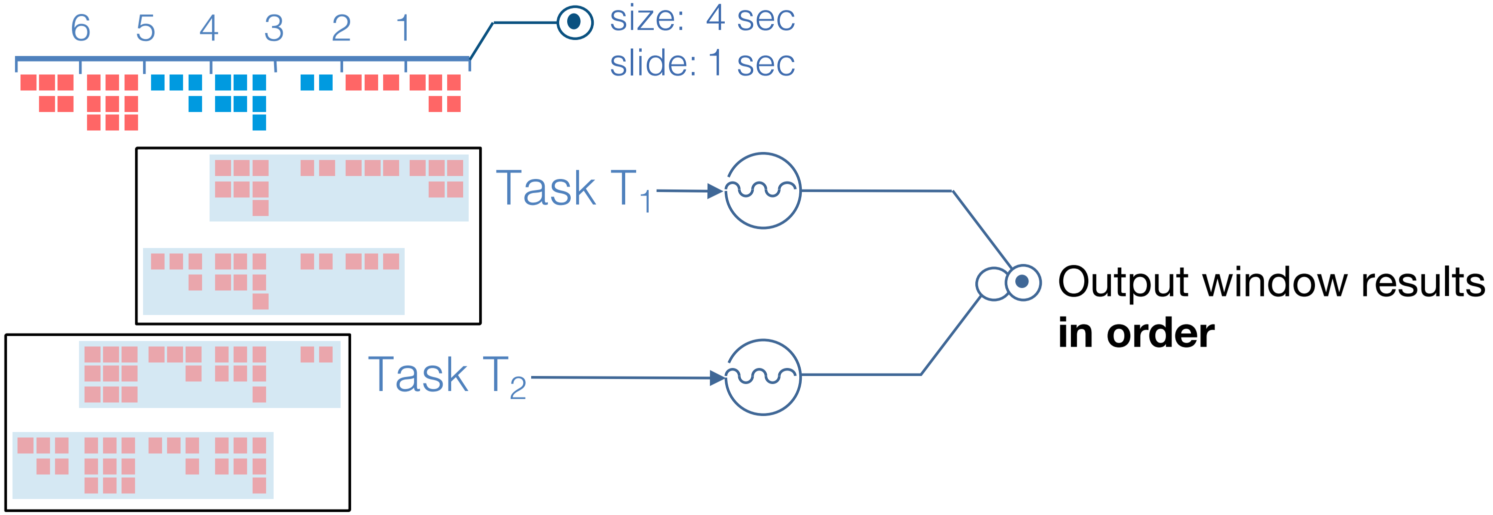
3. How to reduce GPU data movement costs?

Amortise data movement delays with deep pipelining

How to Parallelise Window Computation?

Problem: Window semantics affect system throughput and latency

– Pick task size based on window size?

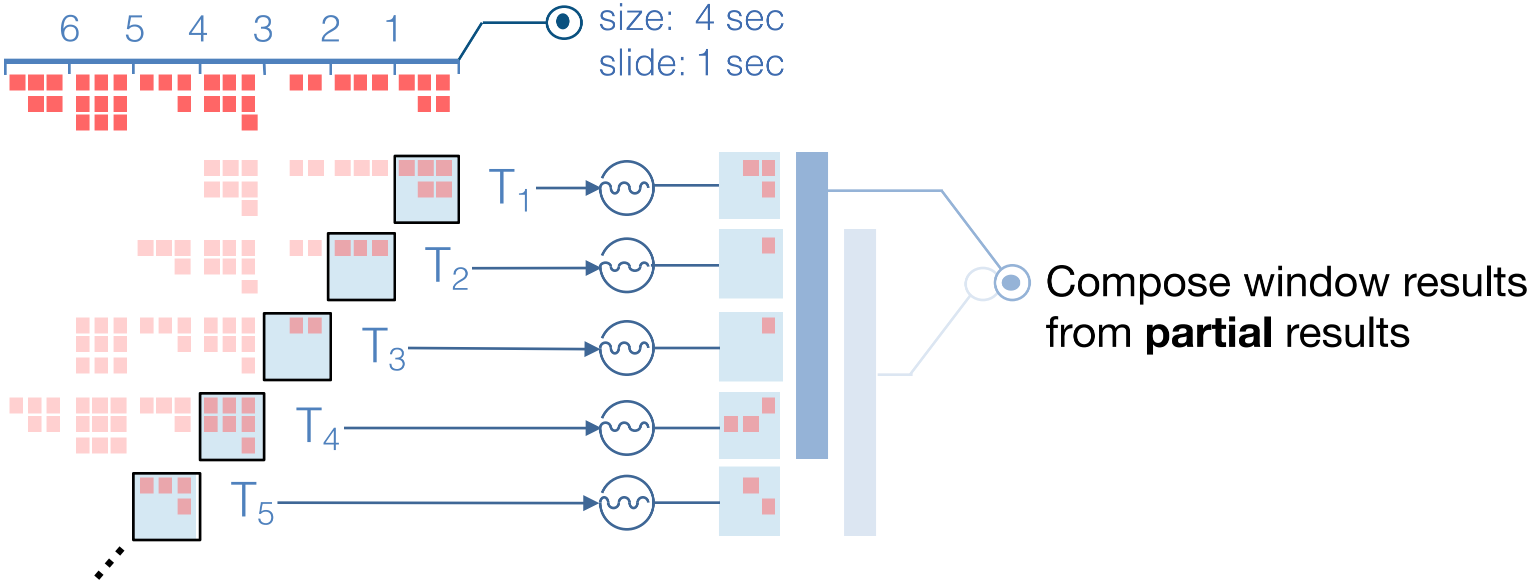


Window-based parallelism results in **redundant** computation

How to Parallelise Window Computation?

Problem: Window semantics affect system throughput and latency

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



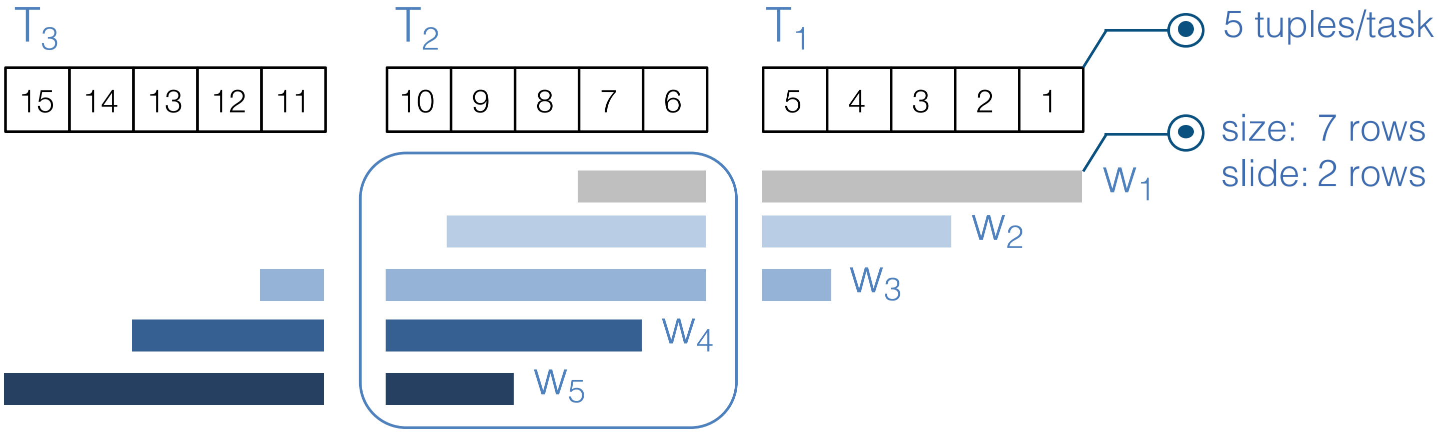
Compose window results from **partial** results

Slide-based parallelism limits GPU parallelism

SABER's Window Processing Model

Idea: Decouple task size from window size/slide

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

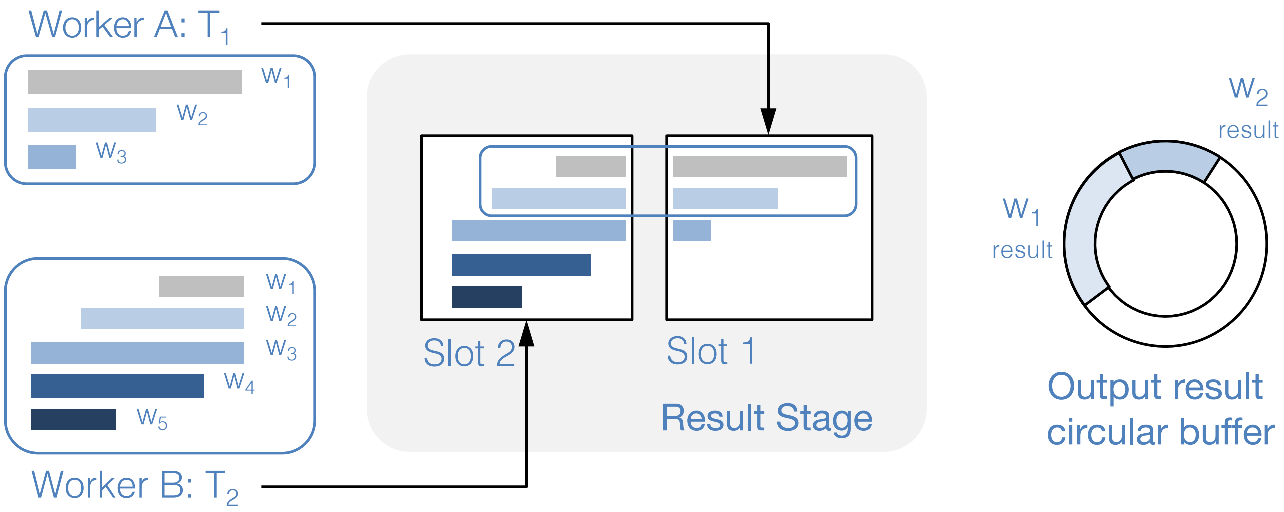


- Task contains one or more window fragments
  - E.g. closing/pending/opening windows in T_2

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, merges window fragment results and forwards complete windows downstream

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

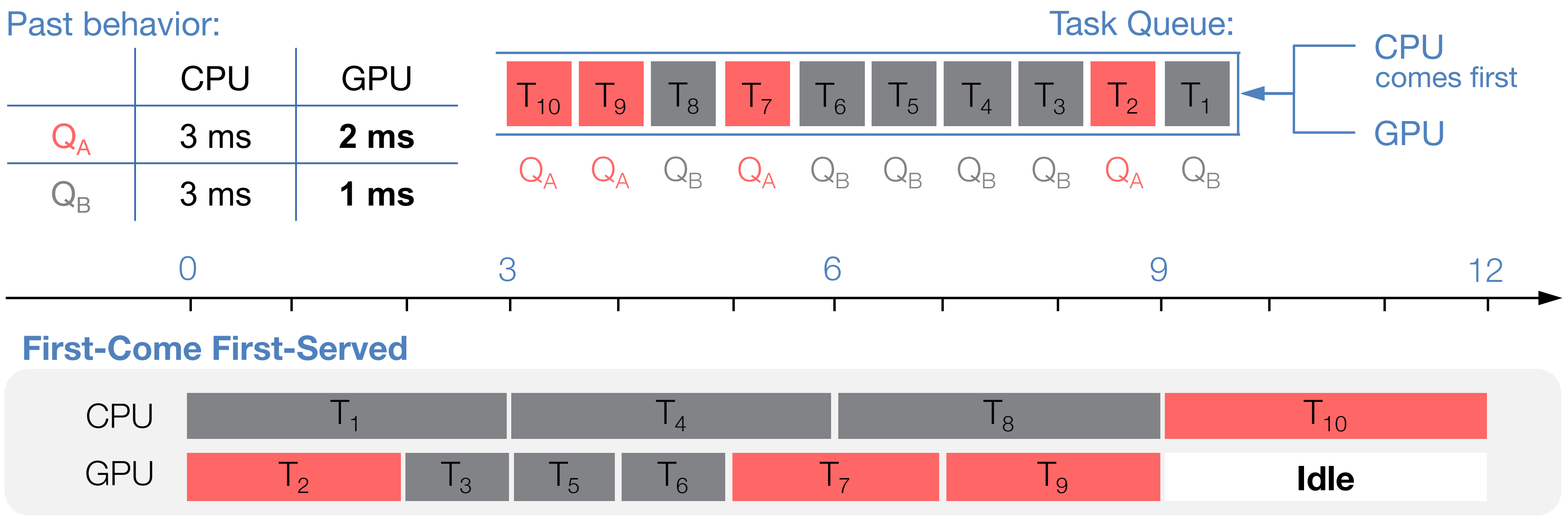
3. How to reduce GPU data movement costs?

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**

Task Queue:
T_{10} T_9 T_8 T_7 T_6 T_5 T_4 T_3 T_2 T_1
Q_A Q_A Q_B Q_A Q_B Q_B Q_B Q_B Q_A Q_B
CPU comes first
GPU
0
3
6
9
12
First-Come First-Served
CPU
T_1 T_4 T_8 T_{10}
GPU
T_2 T_3 T_5 T_6 T_7 T_9
Idle

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

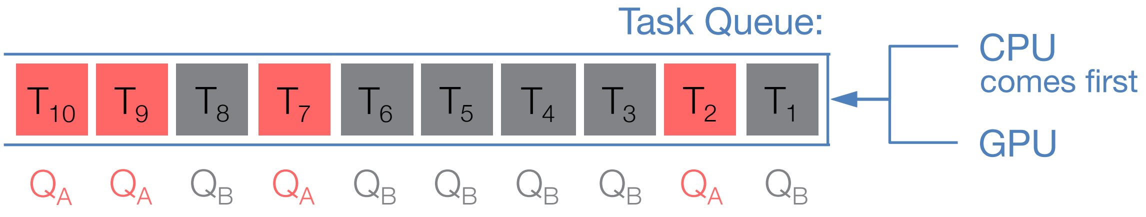
Heterogeneous Look-Ahead Scheduler (HLS)

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

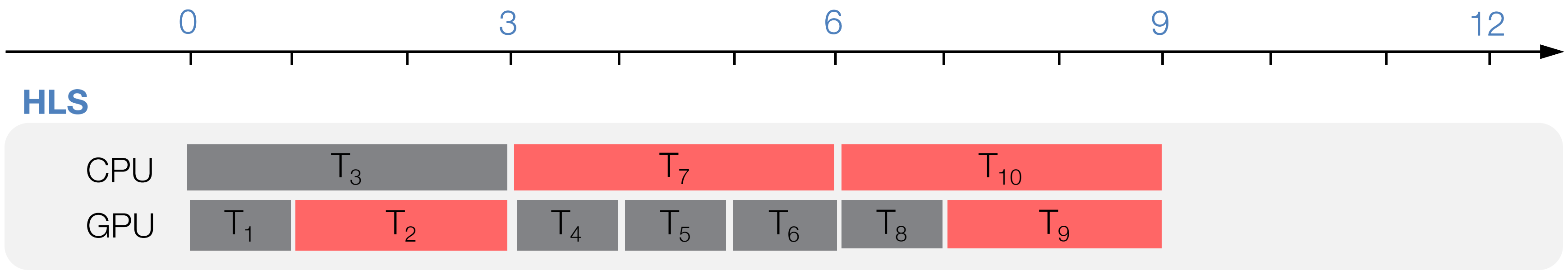
- Decision based on observed query task throughput

Past behavior:

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



Task Queue:
T_{10} T_9 T_8 T_7 T_6 T_5 T_4 T_3 T_2 T_1
Q_A Q_A Q_B Q_A Q_B Q_B Q_B Q_B Q_A Q_B
CPU comes first
GPU

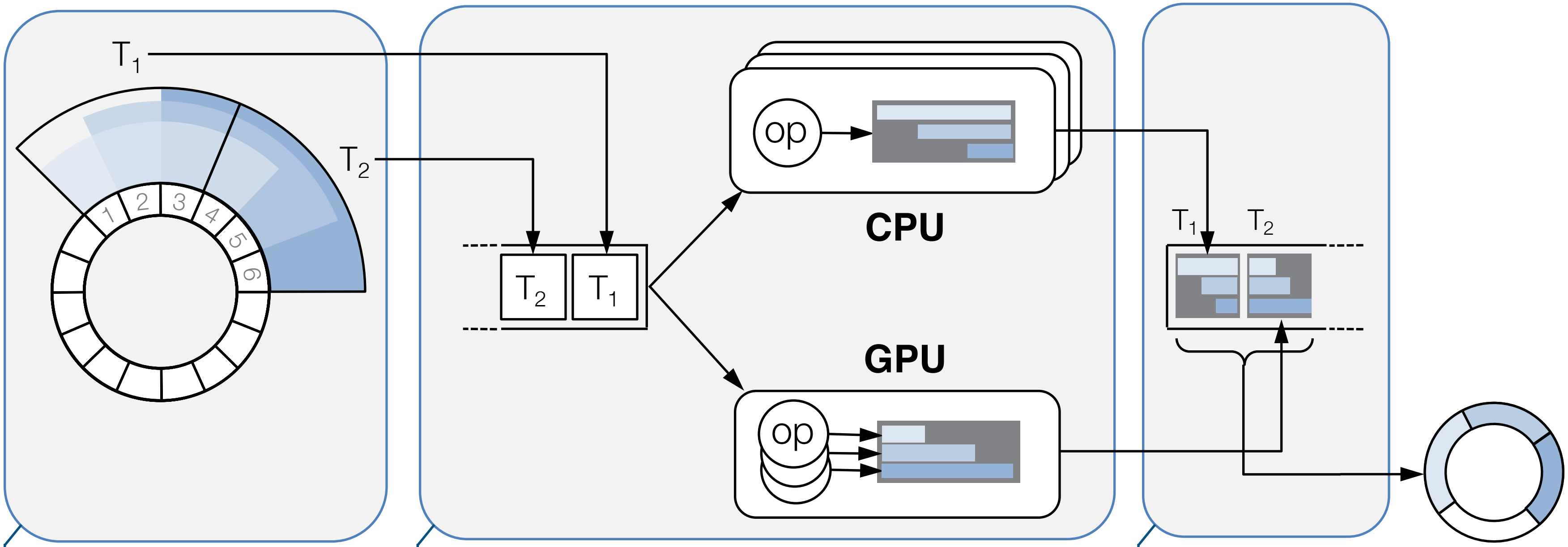


HLS **fully** utilises processors

The SABER Architecture

Java
15K LOC

C & OpenCL
4K LOC



Dispatching stage

Dispatch
fixed-size tasks

Scheduling & execution stage

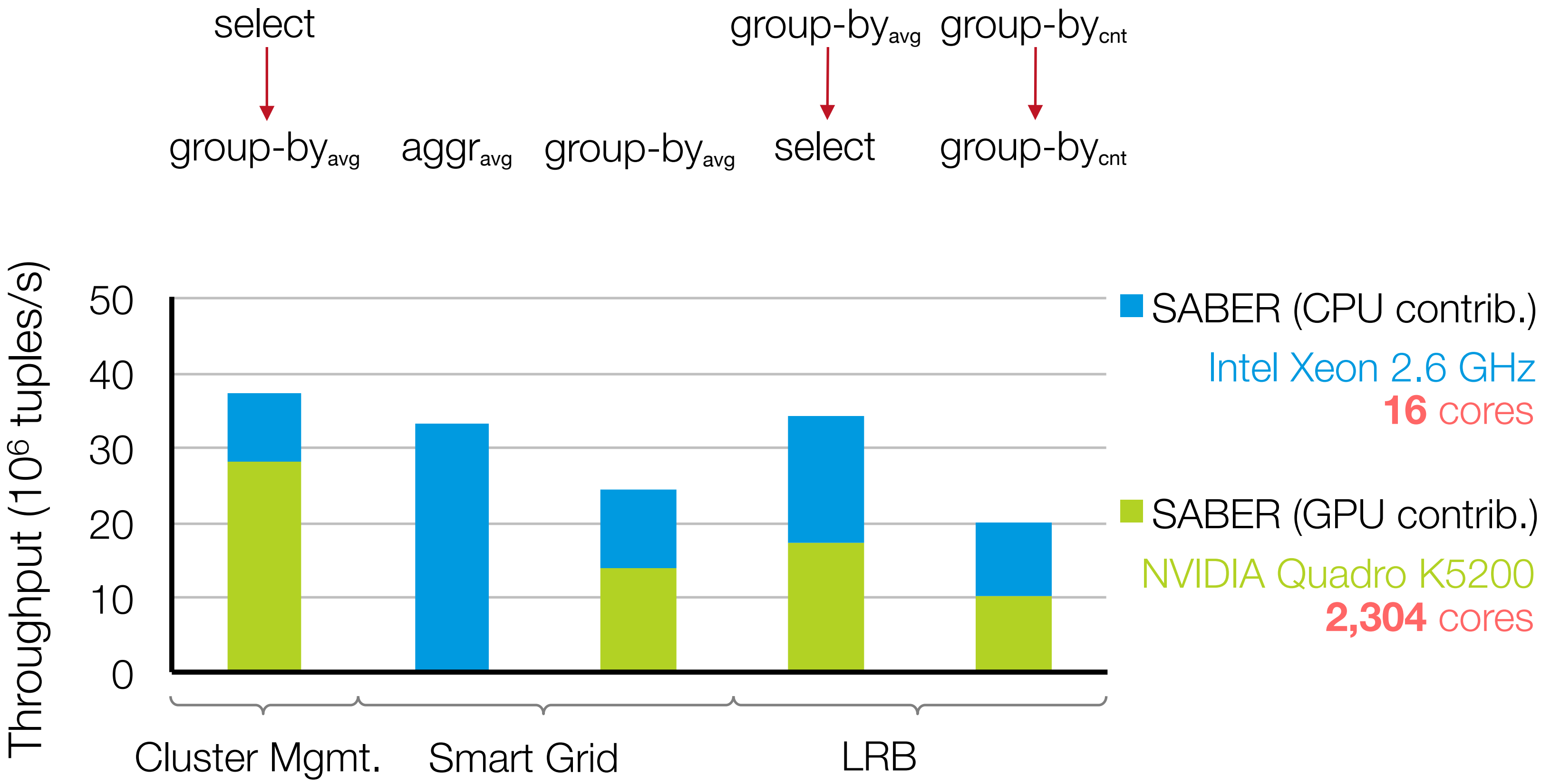
Dequeue tasks
based on HLS

Result stage

Merge & forward partial
window results

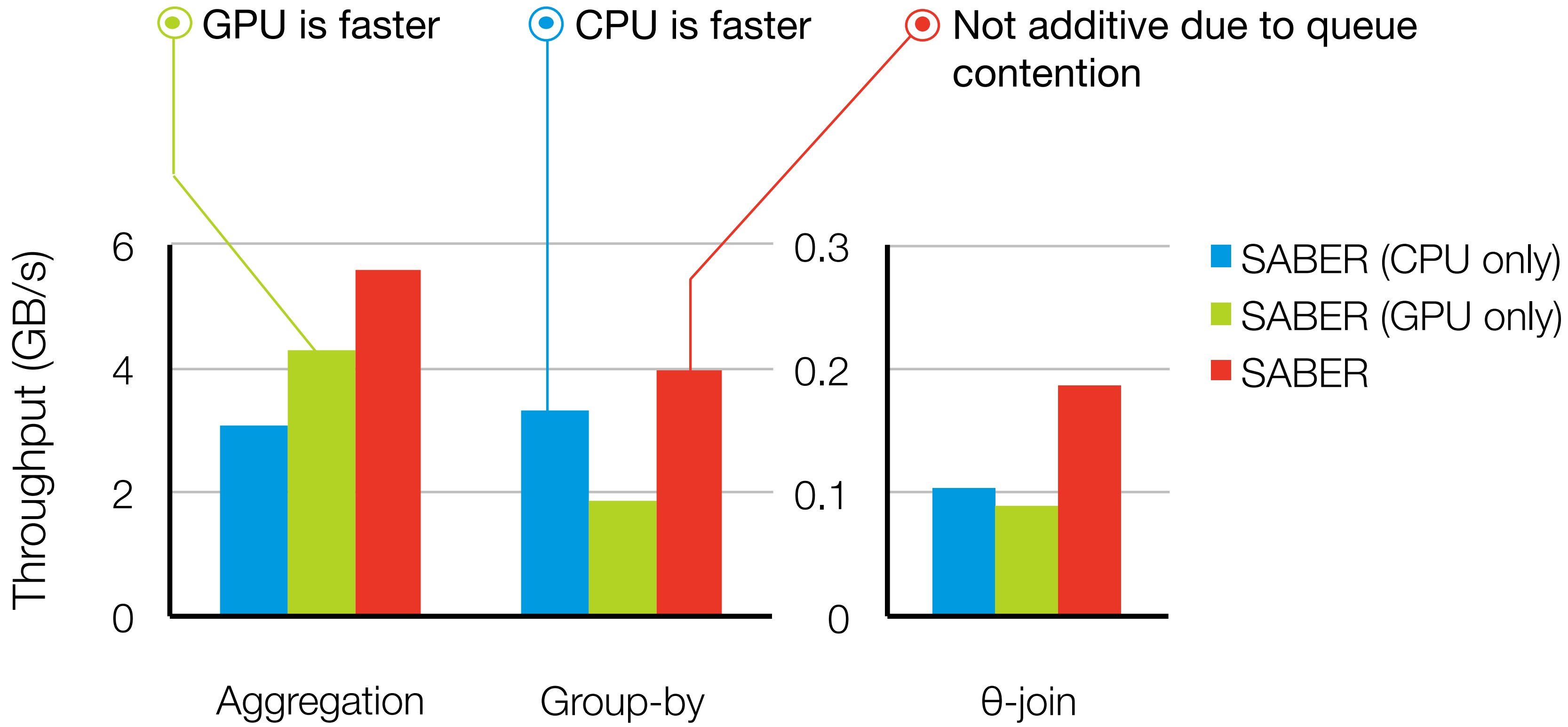
Is Hybrid Stream Processing Effective?

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



Is Hybrid Stream Processing Effective?

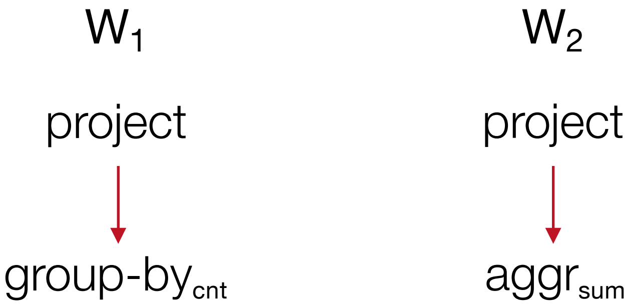
Aggregate throughput of CPU and GPU **always higher** than its counterparts


GPU is faster

CPU is faster

Not additive due to queue contention

Is Heterogeneous Look-Ahead Scheduling Effective?



W_1

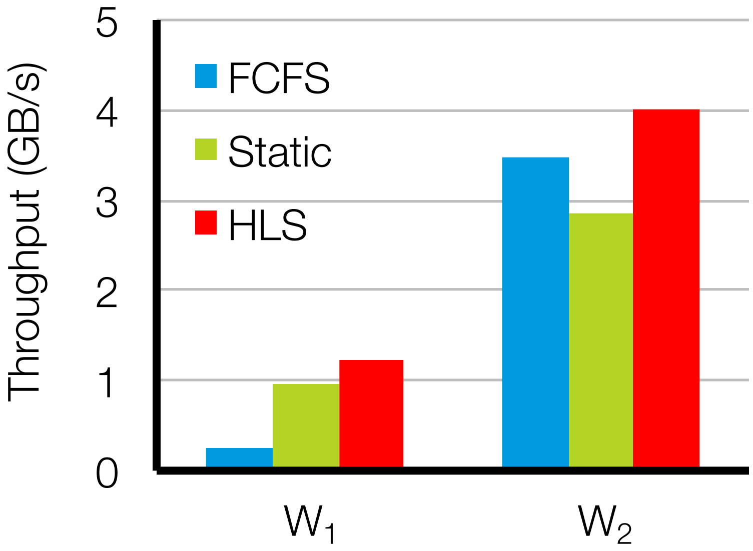
project → group-by$_{cnt}$

W_2

project → aggr$_{sum}$

	CPU	GPU
π		5x
γ	6x	

	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 group-by tasks
- W_2 benefits from FCFS but HLS better utilises GPU:
  - HLS CPU:GPU split is 1:2.5 for project and 1:0.5 for aggr

Summary

Window processing model

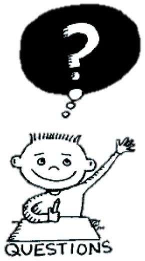
Decouples query semantics from system parameters

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