Imperial College London

LSDS
Large-Scale Data & Systems Group

Hardware-efficient Stream Processing

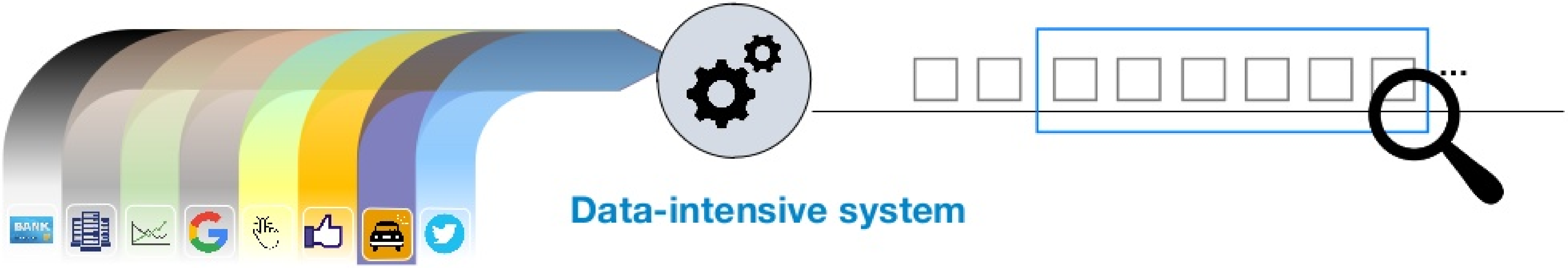
Georgios Theodorakis, Panagiotis Garefalakis, Alexandros Koliousis, Holger Pirk, Peter Pietzuch

Large-Scale Data & Systems (LSDS) Group
Imperial College London

http://lsds.doc.ic.ac.uk
<grt17@imperial.ac.uk>

[Flink Forward Berlin 2018 - Tuesday, September 4, 2018]

It's a Streaming Problem!



Data-intensive system

High-throughput Processing

Low-latency results

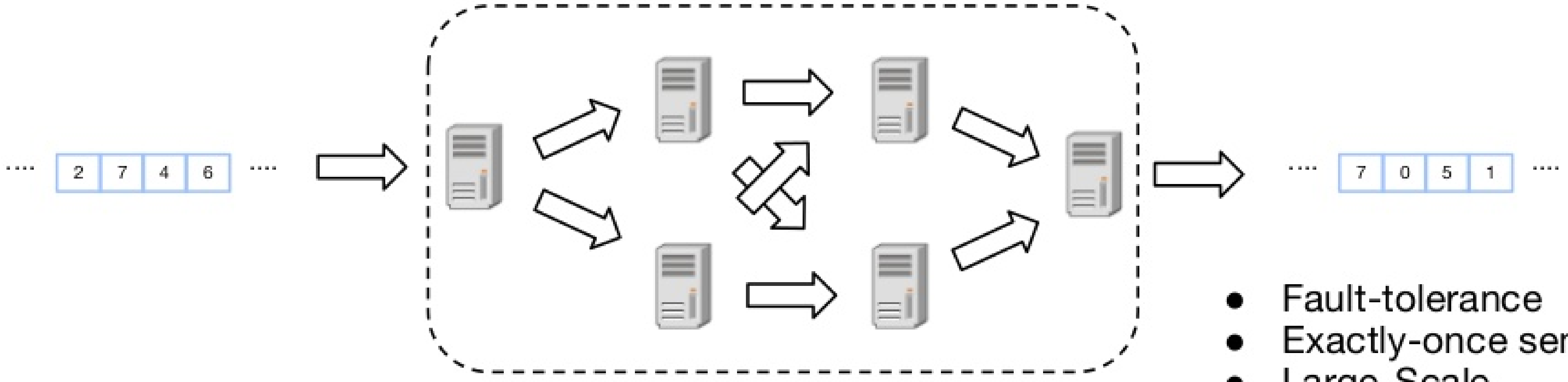
Data-Intensive Applications

- Real-time analytics applications
- Personalised web services
- Network Monitoring
- Training **machine learning** models with large volumes of data

High-throughput & Result Freshness Matter!

Stream Processing

Distributed Streaming Engines



- Fault-tolerance
- Exactly-once semantics
- Large-Scale
- High-level API
- Connectors

☛ Exploit the **aggregated throughput** of many nodes

Nobody (yet!) ever got fired for using a Hadoop cluster

[HotCDP'12]

Or Flink & 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**

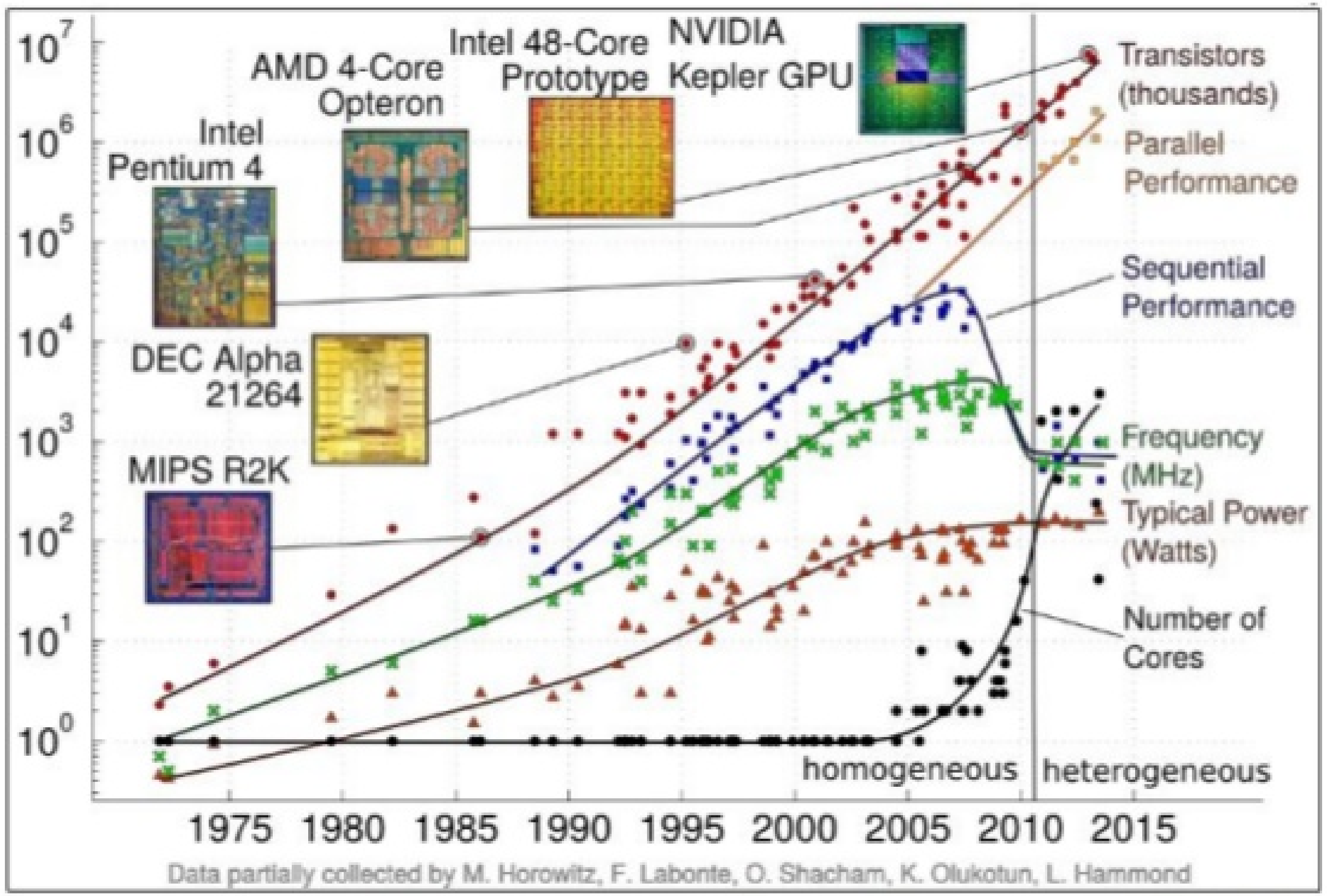
⟹ The size of the workloads has changed, but so has the size/price of **memory**!

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**

Servers are becoming increasingly Heterogeneous



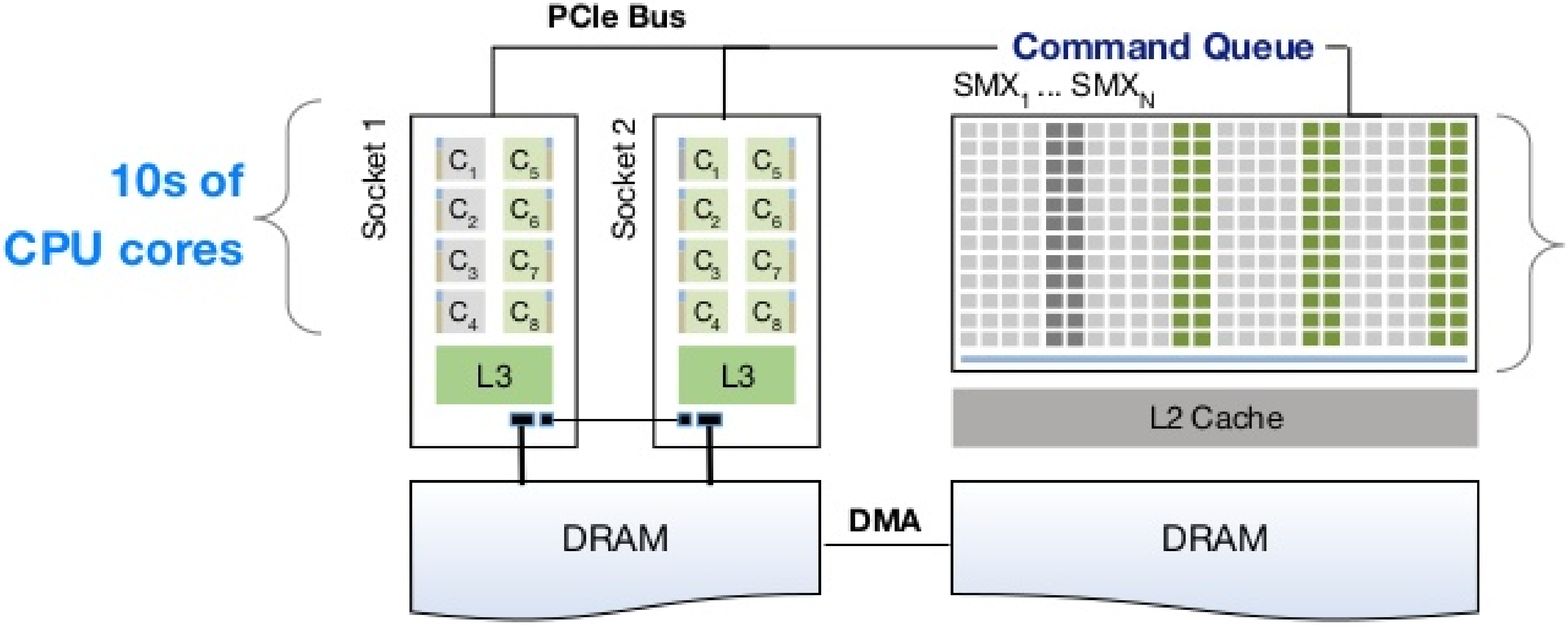
Slide courtesy of Torsten Hoefler (Systems Group, ETH Zürich)

☛ **Perfomance will only come from careful tailoring to the parallel hardware!**

Parallelism of Heterogeneous Servers

Servers have many parallel **CPU cores**

Heterogeneous servers with **GPUs** common



1000s of GPU cores

New types of **compute accelerators**

– Xeon Phi, TPUs, FPGAs, ...

☛ **Exploit previously unseen levels of parallelism**

SABER: Hybrid Stream Processing Engine [SIGMOD'16]

SABER: Hybrid Processing for CPUs & GPUs in a single node

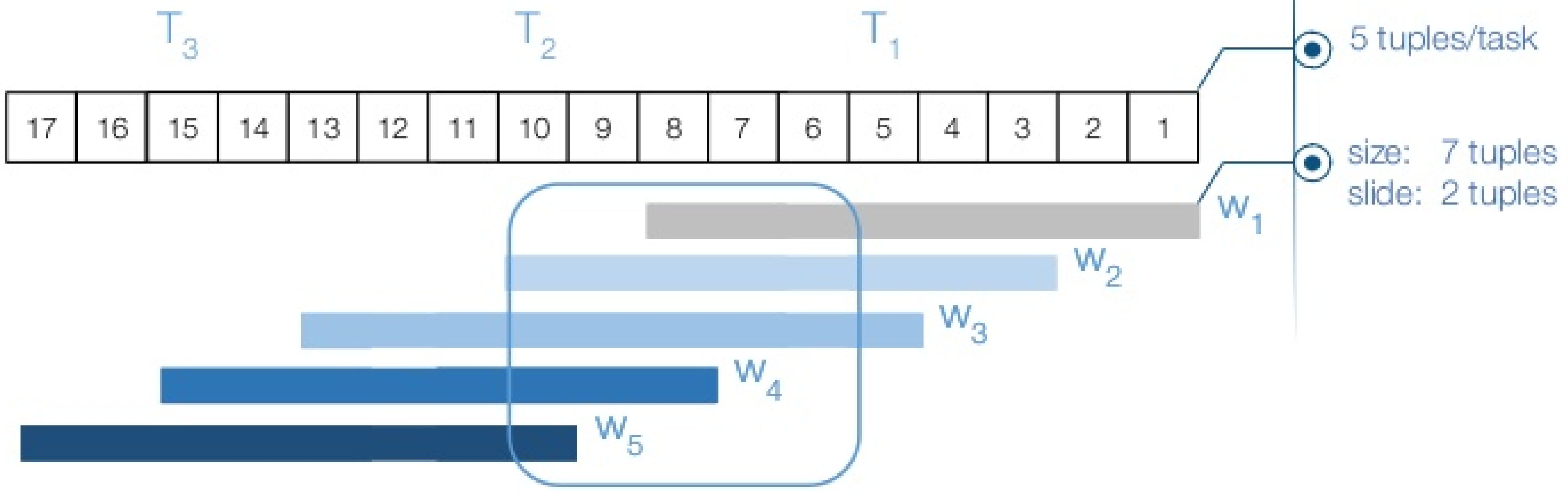
How to get single node performance:

(1) Parallelise computation to fit hardware capabilities
(2) Fully utilise all heterogeneous processors independently of workload

SABER: Hybrid Stream Processing Engine [SIGMOD'16]

(1) Parallelise computation to fit hardware capabilities

- **Decouple hardware/system parameters** from **processing semantics**
- Task contains one or more **window fragments**
- Parallelise using **task size** that is **best for hardware**

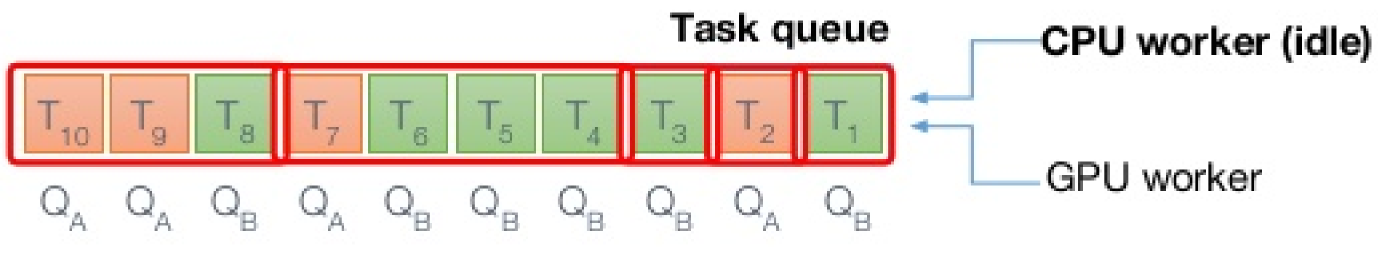


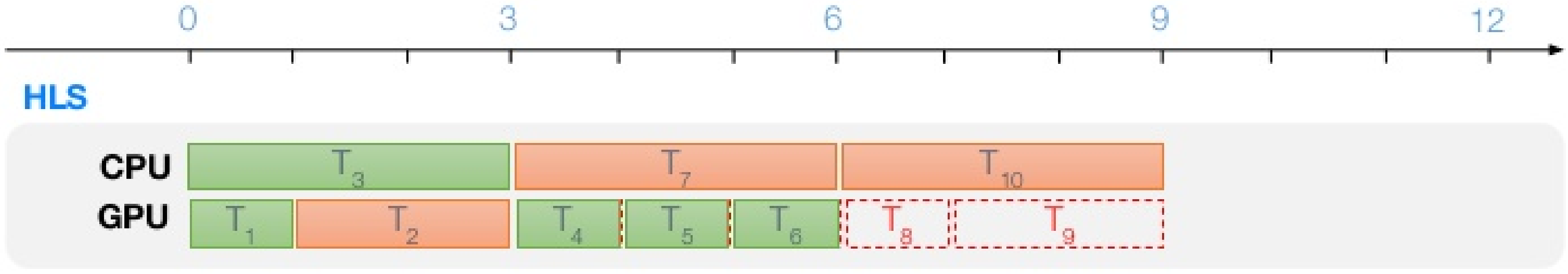
SABER: Hybrid Stream Processing Engine [SIGMOD'16]

(2) Fully utilise all heterogeneous processors independently of workload

- **Hybrid processing model** to achieve aggregate CPU/GPU throughput
- Fully utilise hardware parallelism with **Heterogeneous Lookahead Scheduling (HLS)**
- Hide data movements costs: pipeline the transfer between CPU & GPU

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

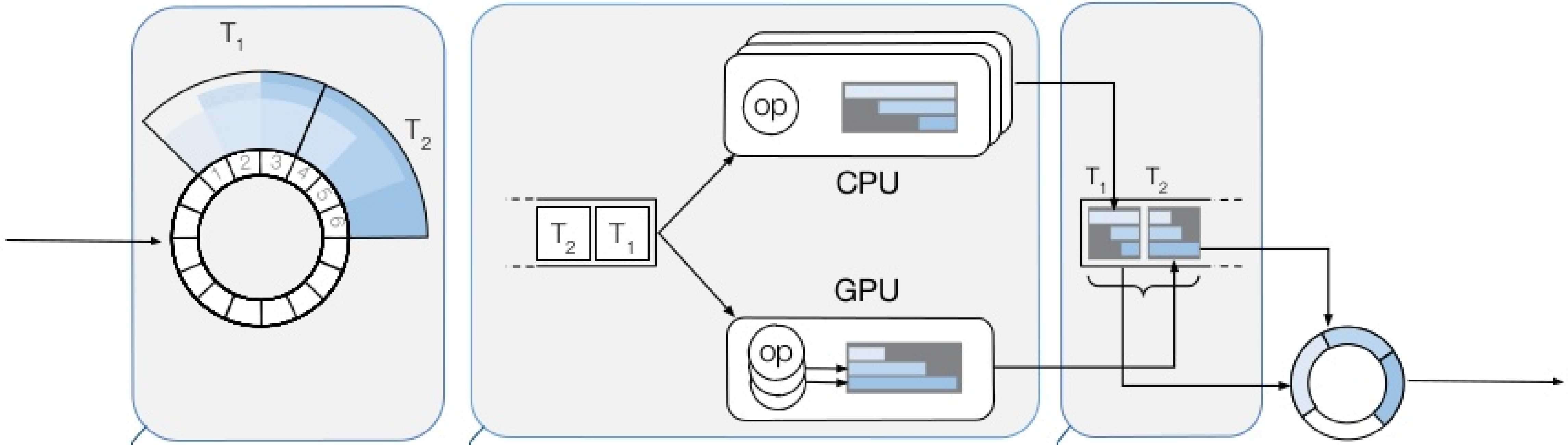




SABER: Hybrid Stream Processing Engine [SIGMOD'16]

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

Single multicore server or multi-node cluster?

Can an efficient stream processing engine on a single server compete with popular distributed stream processing systems?

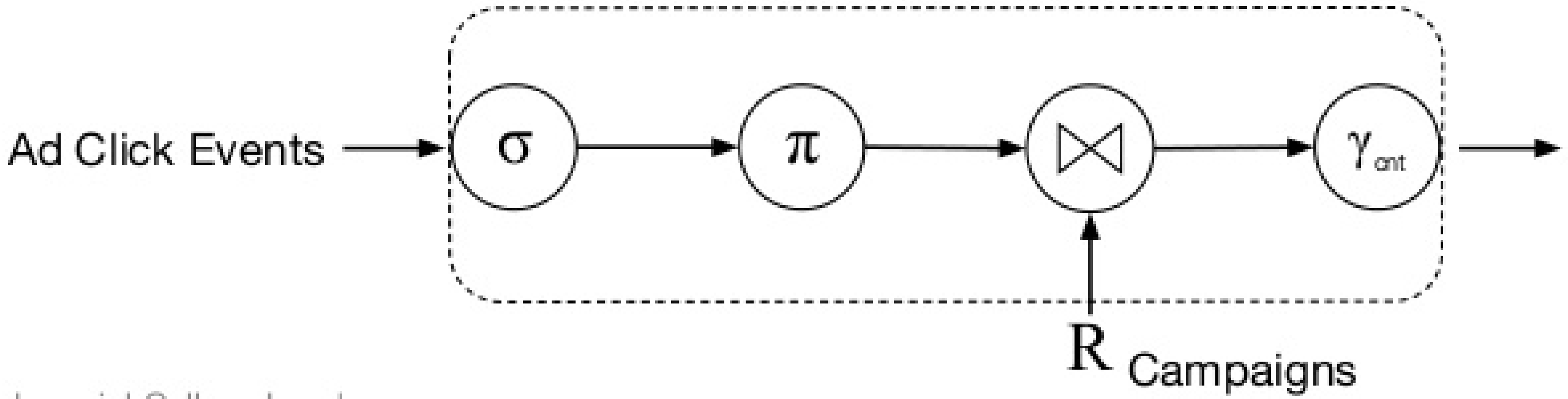
Yahoo! Streaming Benchmark

Advertisement Application: how many times a campaign has been seen in a tumbling window

Key limitations (also reported by Apache Flink, Apache Apex, Differential Dataflow):

- fails to capture the rich semantics of sliding window computation
- not computational intensive

Evaluating systems in **industry** (Databricks, Data Artisans)
& **academia** (Drizzle [SOSP'17], Spark Streaming [SIGMOD'18], Chi [VLDB'18])



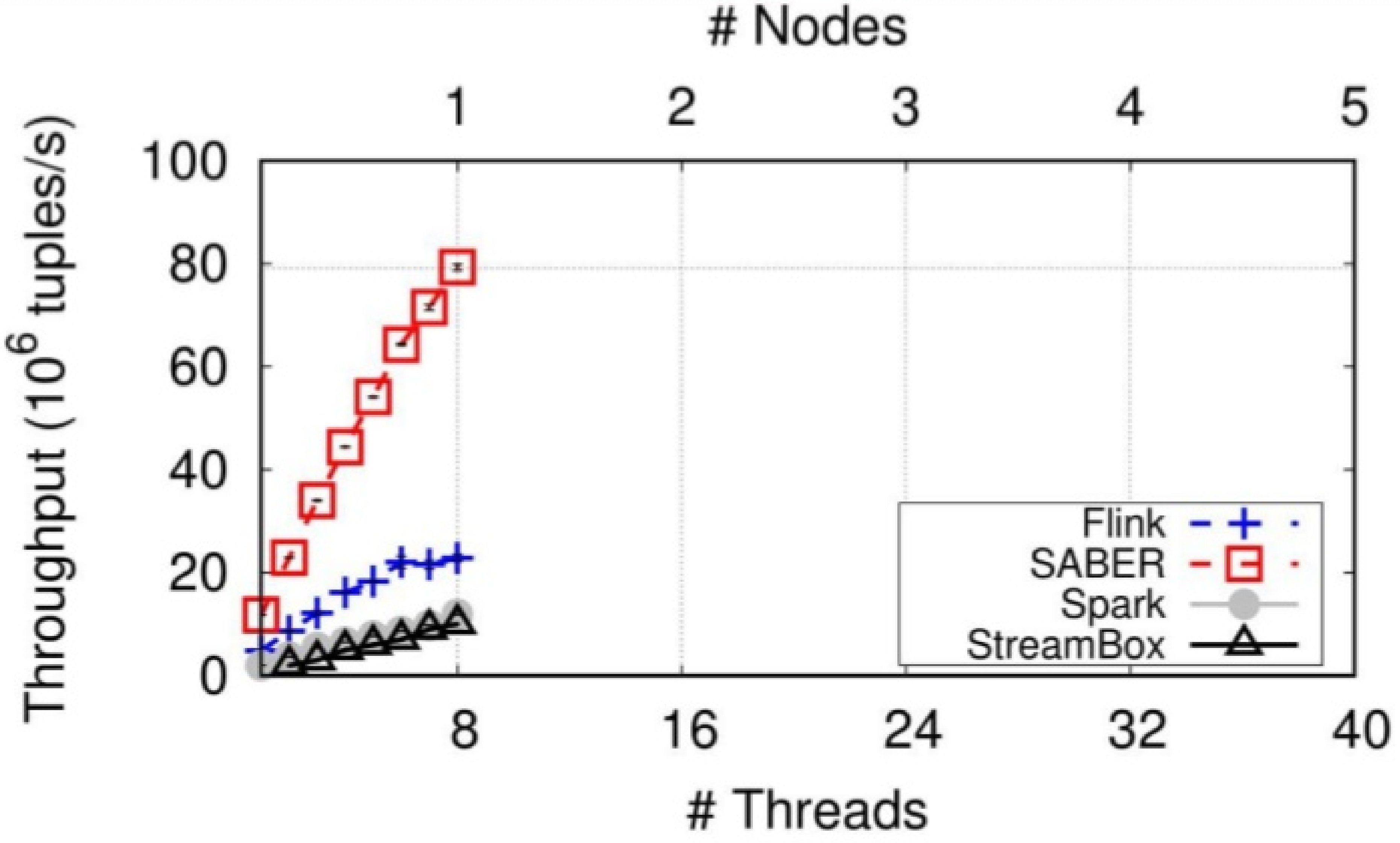
Yahoo! Streaming Benchmark: Systems

- **Apache Flink (1.3.2)**
- **Apache Spark Streaming (2.4.0)**
- **SABER**: without GPU support
- **StreamBox**: a single-server implementation that emphasises out-of-order processing of data

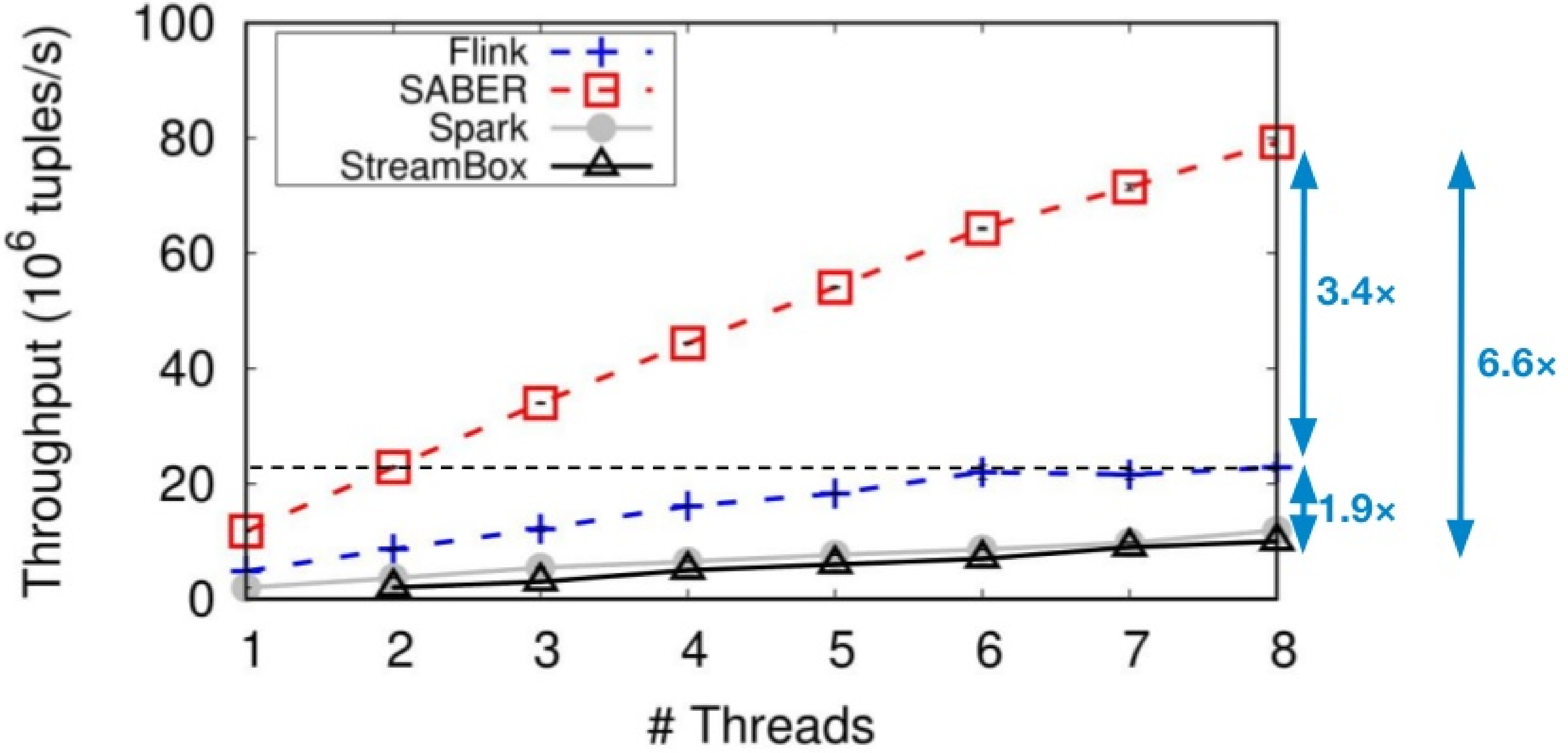
Yahoo! Streaming Benchmark: Setup

- **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**

Comparing throughput in a single node

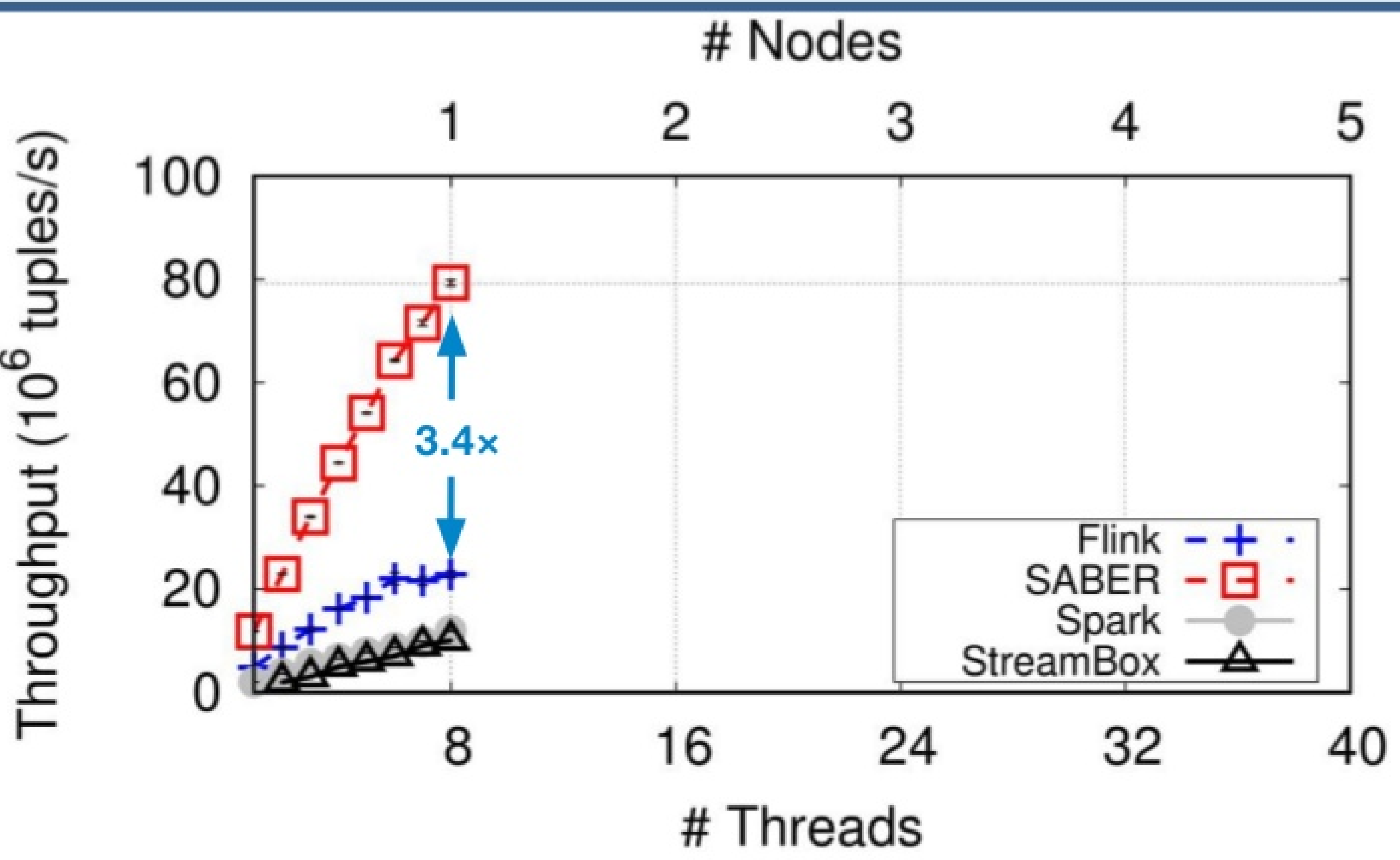


Reducing serialization & keeping data in LLC!

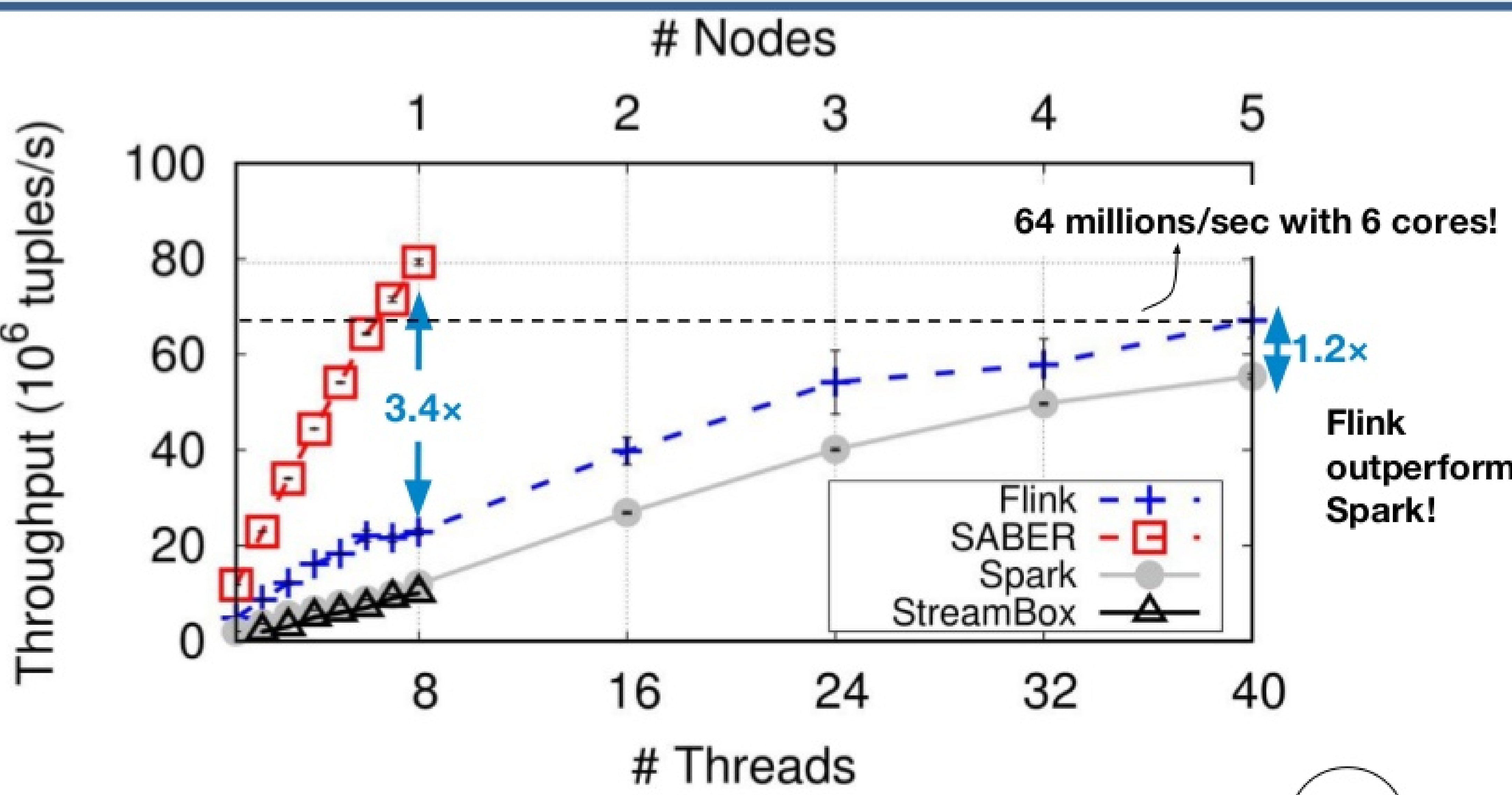


☛ **Keep Data in LLC!**

Single-Server vs Cluster Deployment

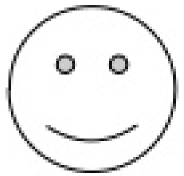


Single-Server vs Cluster Deployment



Flink outperforms Spark!

☛ **Do we need Distributed Stream Processing?**



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

What is the COST of distribution? [HOTOS'15]

Let's set our baseline with a single-core implementation:

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

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

What is the COST of distribution? [HOTOS'15]

Let's set our baseline with a single-core implementation:

	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

More than 3× speedup!

☛ Where does CPU time go?

“You can have a second computer once you’ve shown you know how to use the first one.” – Paul Barham

Design Hardware-Conscious Streaming Systems!

1. highly efficient streaming operators
2. just-in-time generation of platform-specific code for different architectures (CPUs, FPGAs, GPUs)
3. compilation-based techniques to deal with memory stalls (data in CPU registers)
4. hardware-oblivious primitives that optimise computation based on windows

Design Hardware-Conscious Streaming Systems!

1. **<u>highly efficient streaming operators</u>**

2. just-in-time generation of platform-specific code for different architectures (CPUs, FPGAs, GPUs)

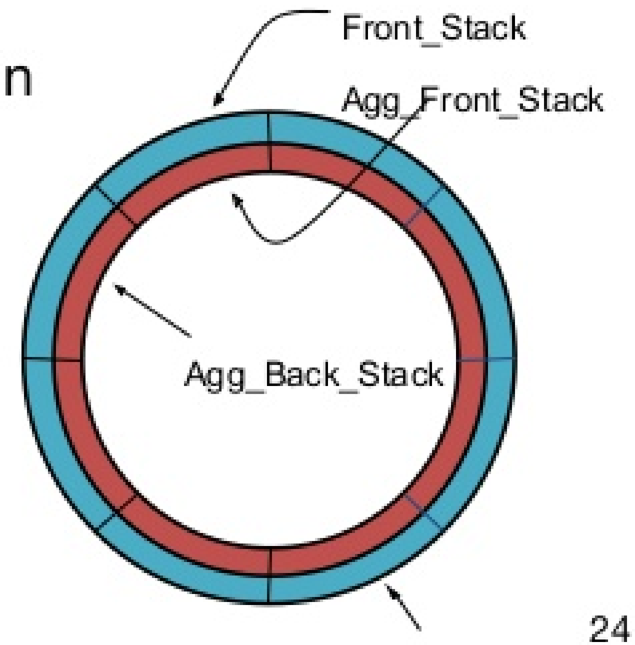
3. compilation-based techniques to deal with memory stalls (data in CPU registers)

4. hardware-oblivious primitives that optimise computation based on windows

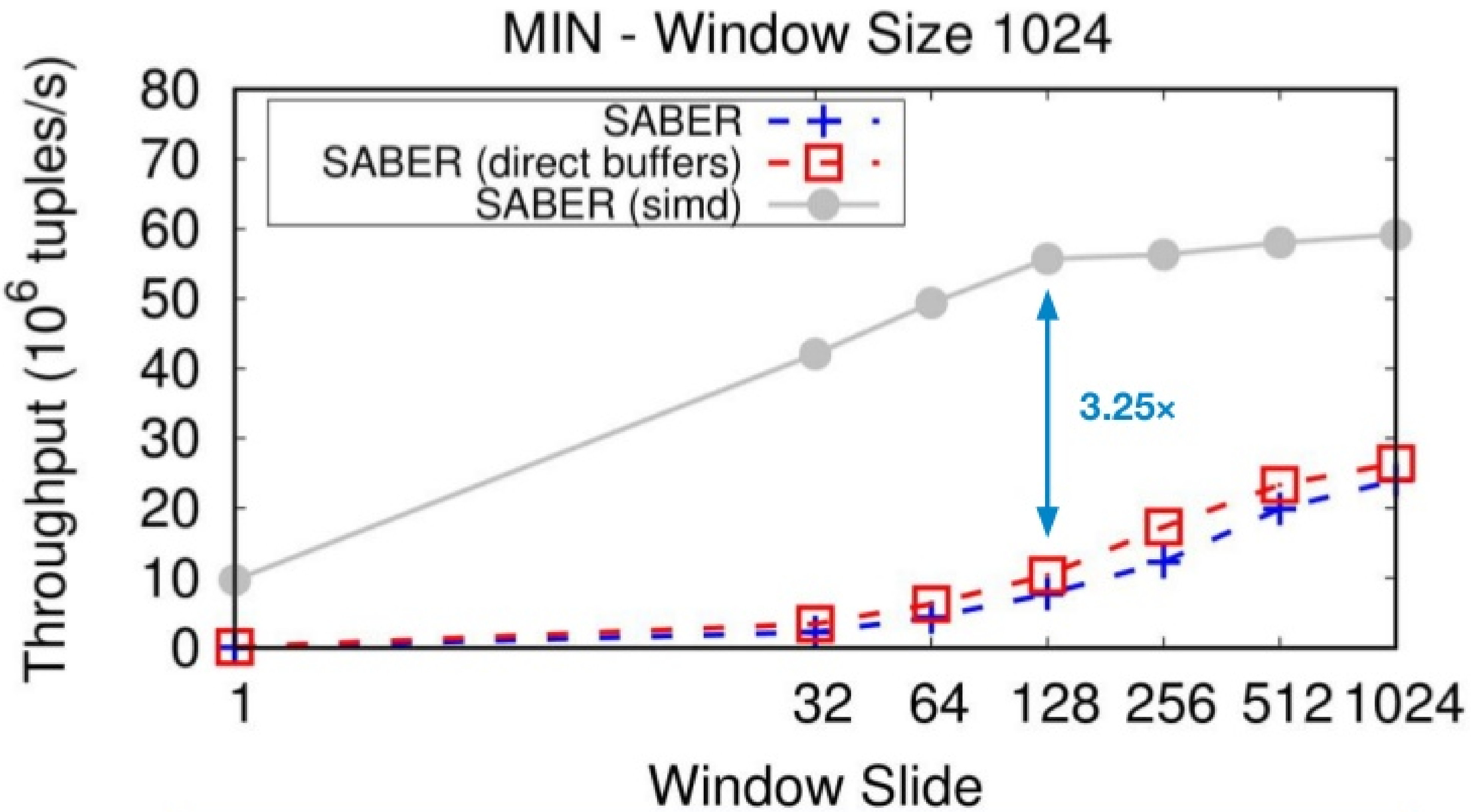
Hardware-efficient Streaming Operators

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



Hammer Slide: integration with SABER



☛ **Integrate with JVM-based streaming systems!**

Design Hardware-Conscious Streaming Systems!

1. highly efficient streaming operators ✓

2. just-in-time generation of platform-specific code for different architectures (CPUs, FPGAs, GPUs)

3. compilation-based techniques to deal with memory stalls (data in CPU registers)

4. hardware-oblivious primitives that optimise computation based on windows

Design Hardware-Conscious Streaming Systems!

1. highly efficient streaming operators ☑

Thank You!
Questions?

Our repository for Yahoo! Benchmark https://github.com/lsds/StreamBench

Georgios Theodorakis
http://lsds.doc.ic.ac.uk
<grt17@imperial.ac.uk>