# Revisiting the Design of Data Stream Processing Systems on Multi-Core Processors

**Shuhao Zhang**[1,2], Bingsheng He[1], Daniel Dahlmeier[2], Amelie Chi Zhou[3], Thomas Heinze[2]

1 National University of Singapore

2 SAP

3 Inria

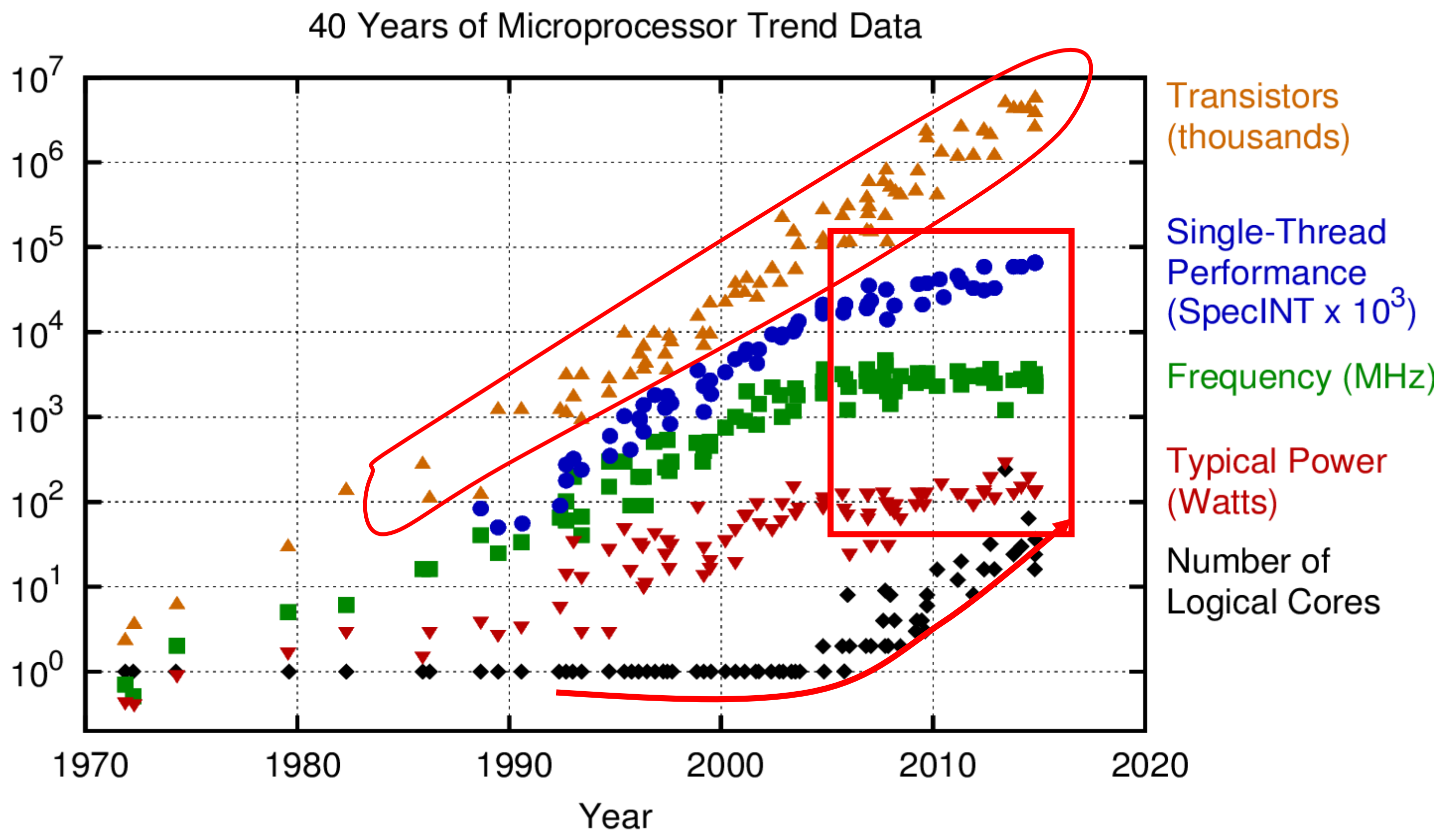
# Outline

- ❖Background.
- ❖Profiling study of DSP systems on modern multi-core processors.
- ❖A new DSP system designed for modern multi-core processors.

# Outline

- ❖ **Background**.
  - ❖ Multi-core processors
  - ❖ DSP systems
- ❖ Profiling study of DSP systems on modern multi-core processors.
- ❖ A new DSP system designed for modern multi-core processors.

# The trend of CPU development

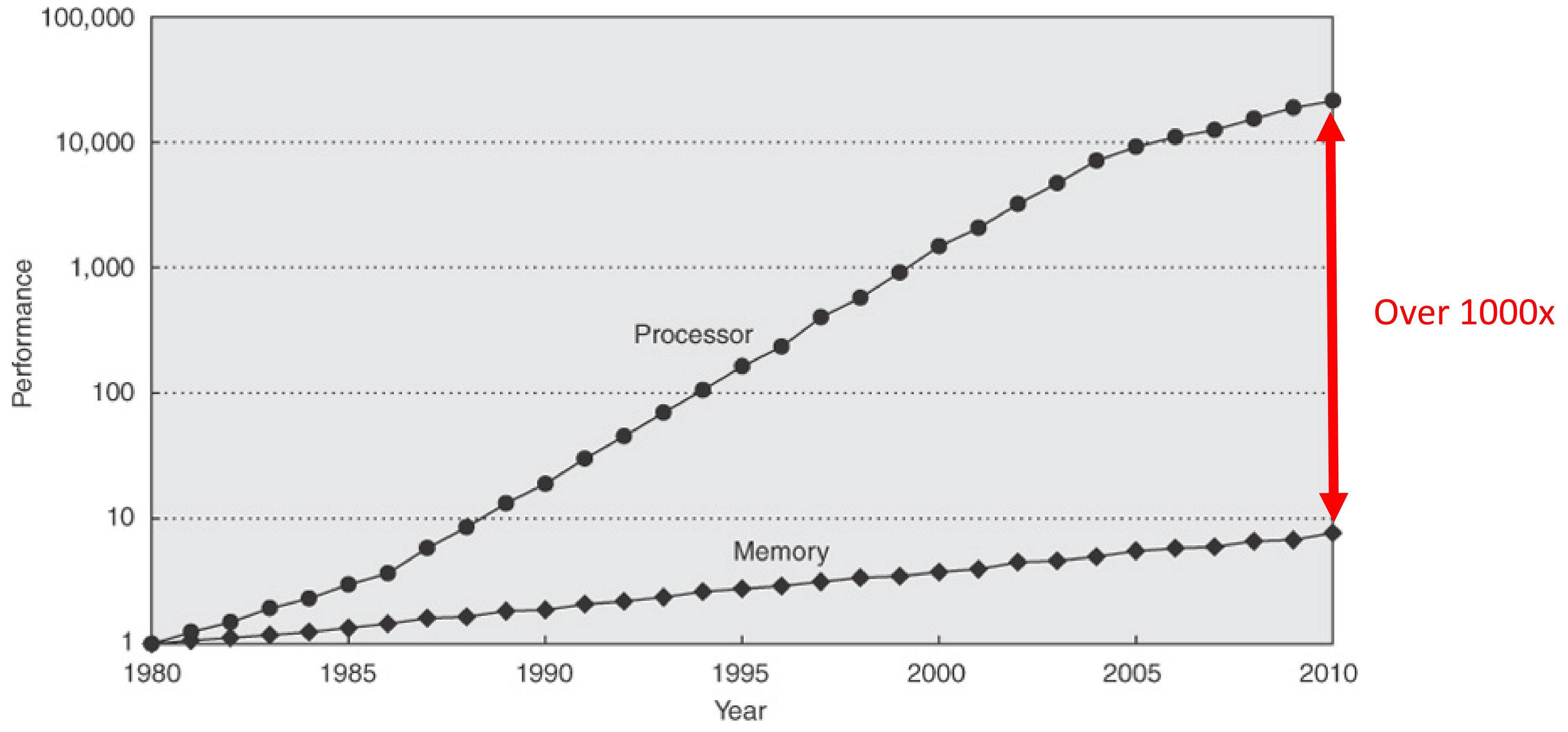


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

- Transistors is still rapidly increasing.
- Number of logical cores is also increasing fast.
- Other metrics are slowing down.

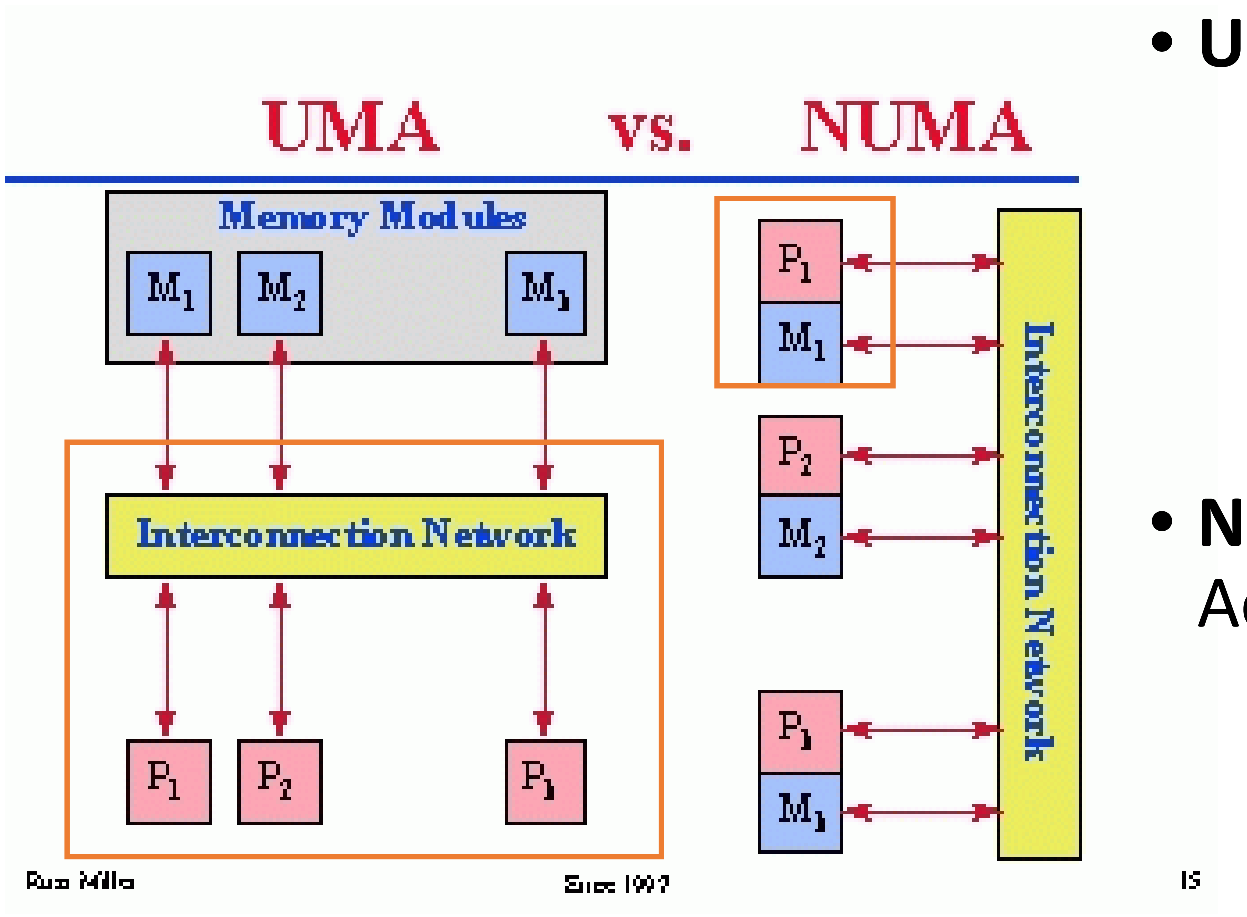
We are now at the multi-core era.

# The speed gap between memory and CPU is widening



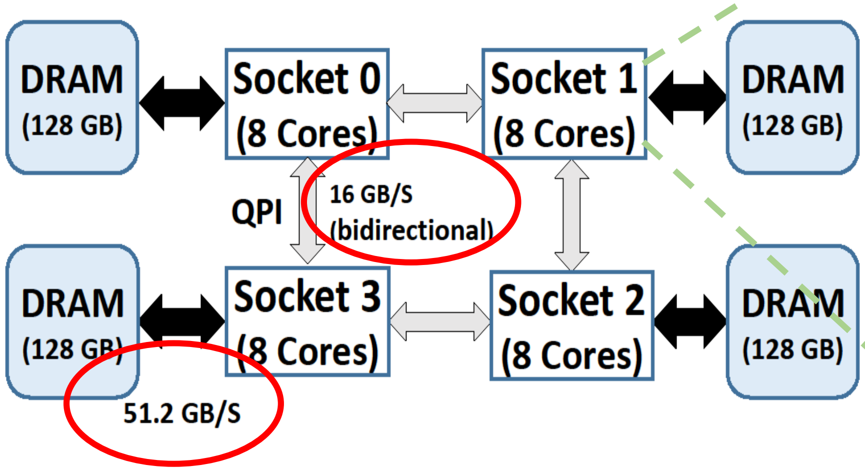
© 2007 Elsevier, Inc. All rights reserved.

# From UMA to NUMA

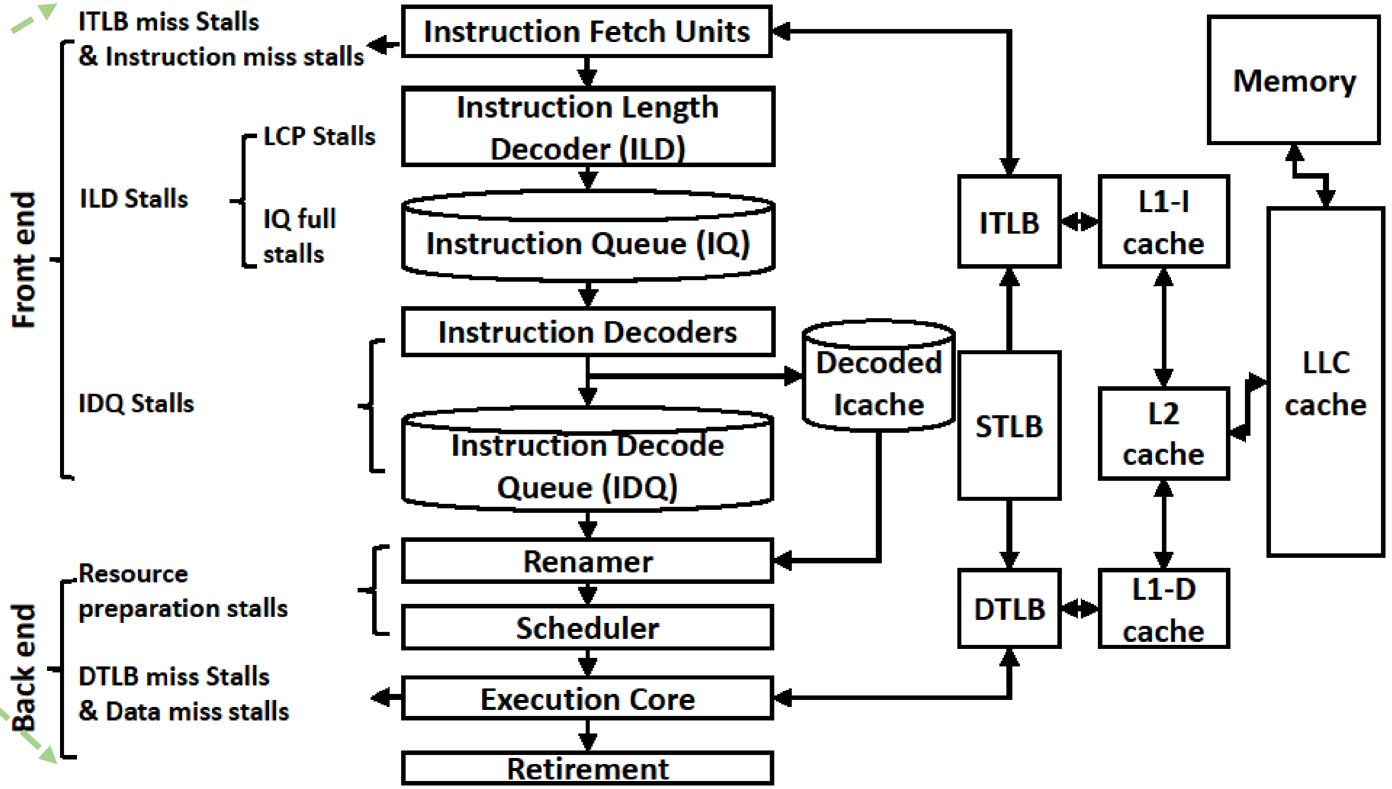


- **UMA** (Uniform Memory Access)**:**
  - access time to a memory location is independent of which processor makes the request or which memory chip contains the transferred data.
  - **Issues: power and scalability**
- **NUMA**(Non-Uniform Memory Access)**:**
  - a processor can access its own local memory faster than non-local memory.
  - Main-stream architectures

# An overview of one of the state-of-the-art NUMA server

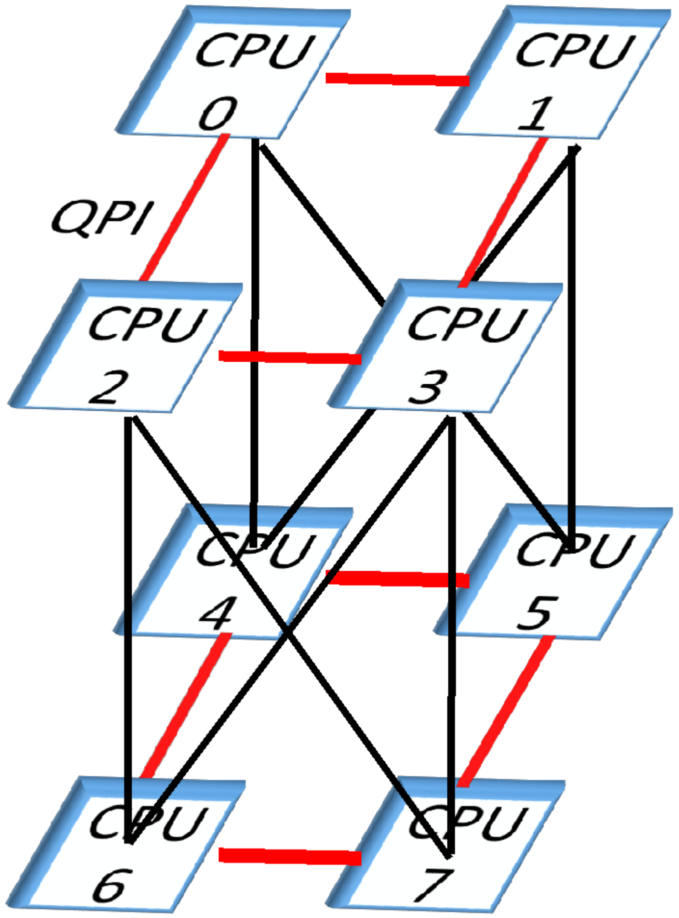


NUMA topology

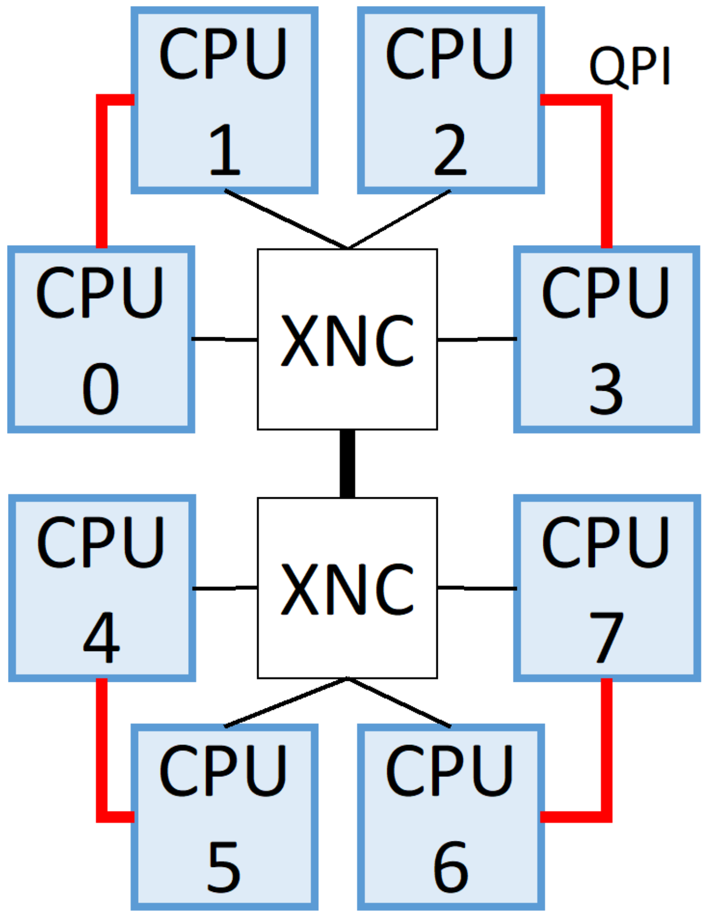


The overview of the execution components of the processor

# NUMA servers



**Glue-less** NUMA architecture from HUAWEI (288 cores and 8 TB of memory)



**Glue-assist** NUMA architecture from HP (128 cores and 2 TB of memory)

# Importance of data stream processing

❖Data stream processing (DSP) has attracted much attention for *real-time* analysis applications, e.g., stock market monitoring and IoT (Internet of Things).

❖Many DSP systems have been proposed recently.

Apache Storm

Twitter Heron

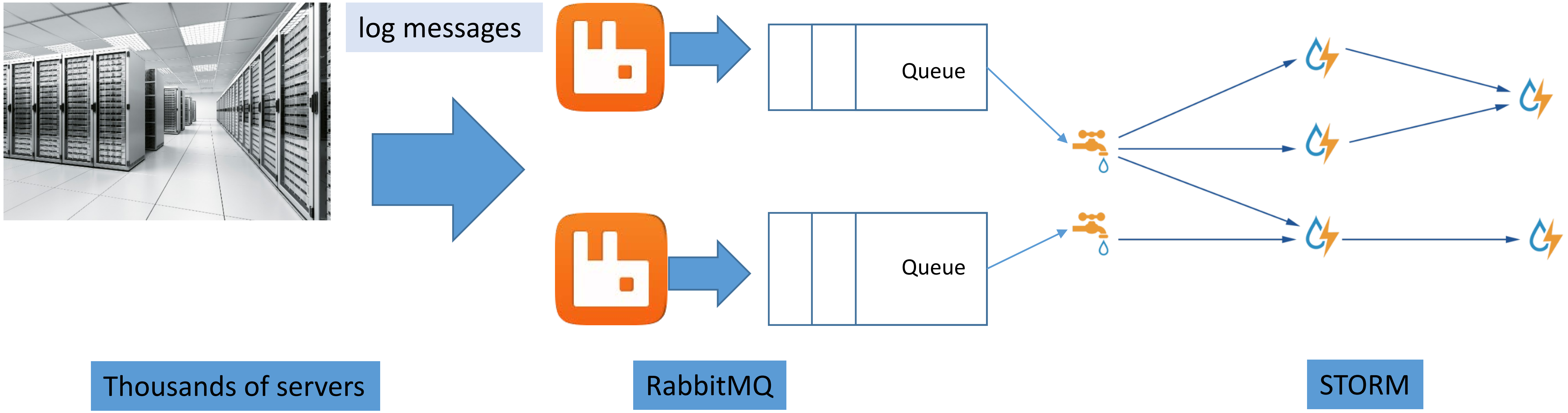
Apache Flink

…

# Some representative stream processing applications/areas

- ❖Complex event processing.
  - ❖Identify composite (complex) event consists of a series of primitive events in real-time manner. For example, stock market monitoring.
- ❖Real-time Machine Learning.
  - ❖Integrate Apache Storm and R to perform real-time data mining task. For example, make real-time decision on which ad to display [http://events.linuxfoundation.org/sites/events/files/slides/StormR.pdf].
- ❖Internet of Things (IoT).

# A concert example: NaviSite, Event Log Monitoring



# Outline

- ❖Background.
- ❖**Profiling study of DSP systems on modern multi-core processors.**
    - ❖**Published work in ICDE'2017**
- ❖A new DSP system designed for modern multi-core processors.

## Common designs of recent DSP systems

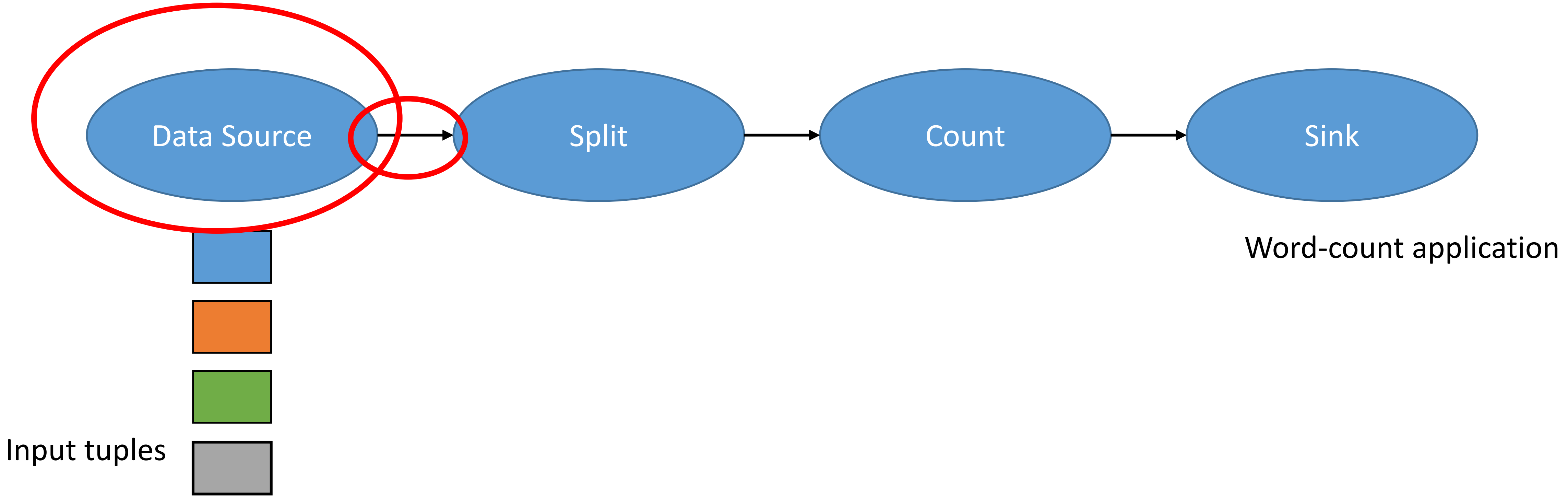
❖ Existing systems mainly focus on scaling out using a cluster of commodity machines.

❖ Three common design aspects

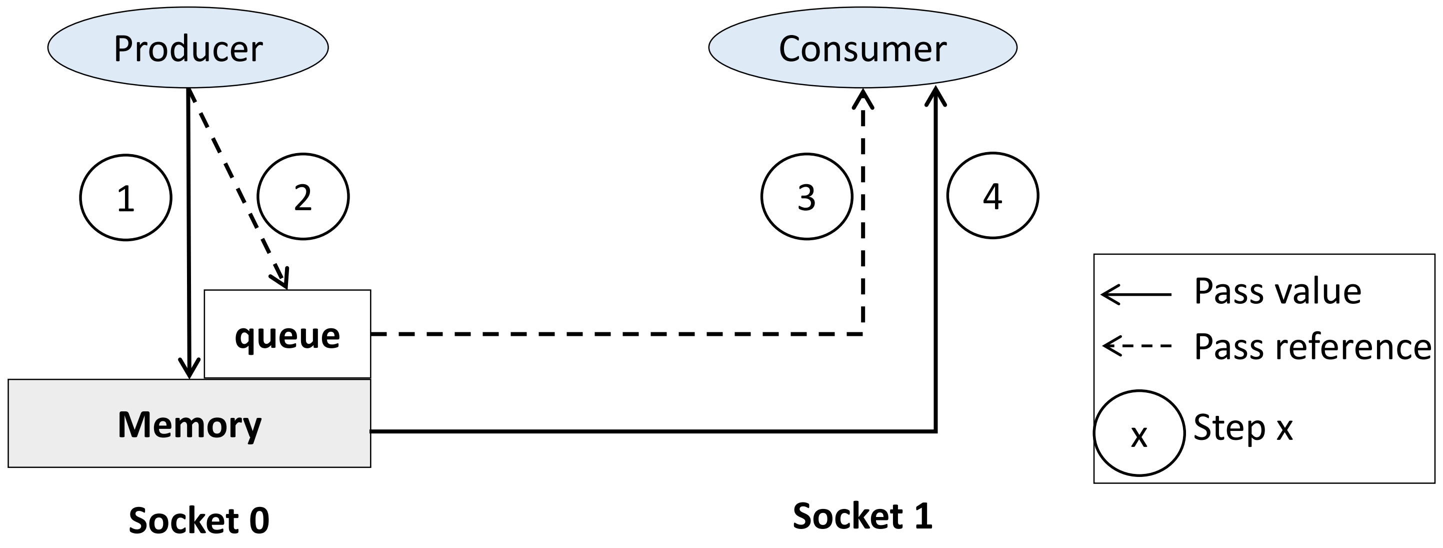
a) Pipelined processing with message passing
b) On-demand data parallelism
c) JVM based implementation

❖ In this paper, we study the three design aspects on scale-up architectures (i.e., multi-socket multi-core architectures).

## Design aspect 1: Pipelined processing with message passing

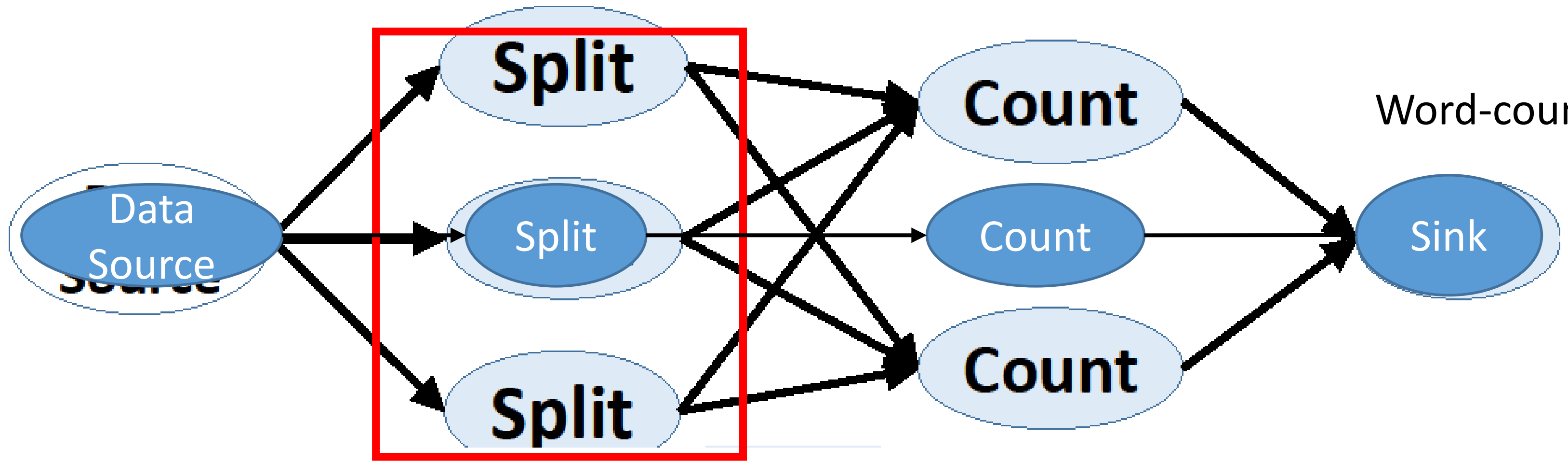


# Design aspect 1: Pipelined processing with message passing



## Design aspect 2: On-demand data parallelism

❖Modern DSP systems such as Storm and Flink are also designed to support data parallelism.



Word-count application

## Design aspect 3: JVM based implementation

❖Many DSP systems are implemented with JVM-based programming languages (i.e., Closure, Java, and Scala).

❖We examine three aspects of JVM runtime.

a) Data reference
b) Method table
c) Garbage collection (GC)

❖Since previous studies demonstrate considerate runtime overhead of those components, we study their performance impact on DSP systems.

## Goals & Contributions

❖ Goals

a) Identify the common design aspects of modern DSP systems and to understand how those designs interact with modern processors
b) Develop some hardware and software approaches to resolving the bottleneck

❖ Contributions

a) We design a micro-benchmark covering a wide range of applications.
b) We conduct detailed profiling study on two state-of-the-art DSP systems.
c) We design optimization techniques to solve the identified performance issues.

## Benchmark design

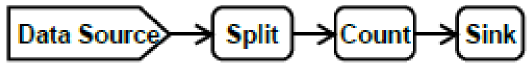
- ❖There has been no standard benchmark for DSP systems, especially on scale-up architectures.
- ❖We design our benchmark according to the four criteria proposed by Jim Gray [1].
  a) Relevance
  b) Portability
  c) Scalability
  d) Simplicity

1: J. Gray, Benchmark Handbook: For Database and Transaction Processing Systems. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992.

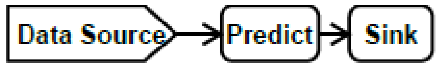
# Benchmark

d) Simplicity: Choose the applications with simplicity in mind

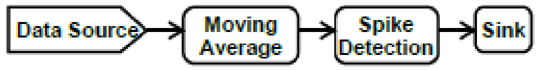
dth consumption



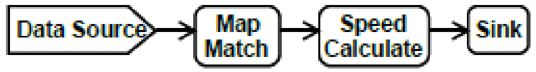
(a) Stateful Word Count (WC)



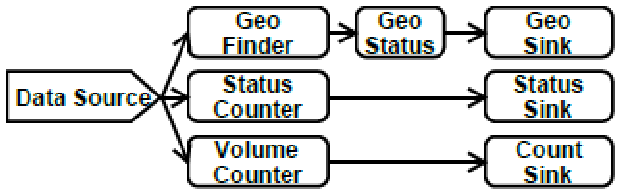
(b) Fraud Detection (FD)



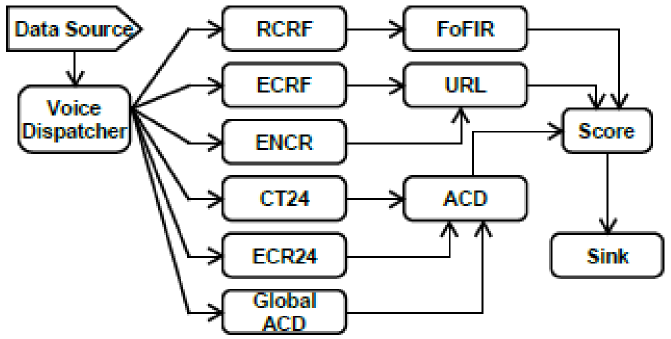
(c) Spike Detection (SD)



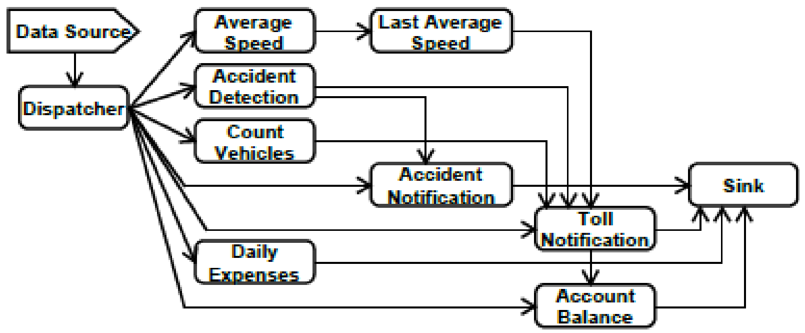
(d) Traffic Monitoring (TM)



(e) Log Processing (LG)



(f) Spam Detection in VoIP (VS)



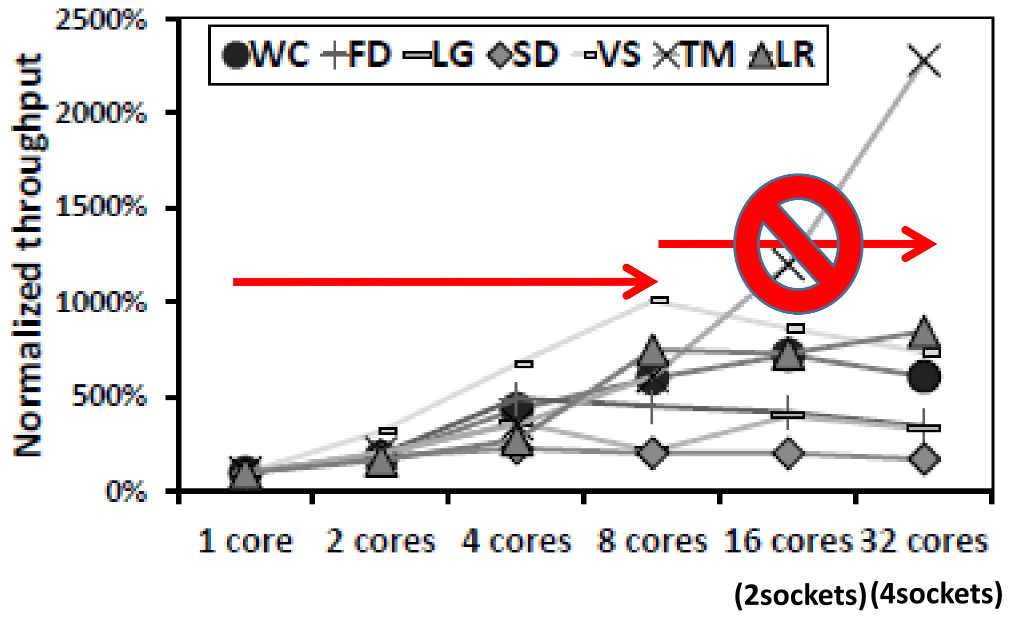
(g) Linear Road (LR)

https://github.com/shzhang1/ProfilingStudy

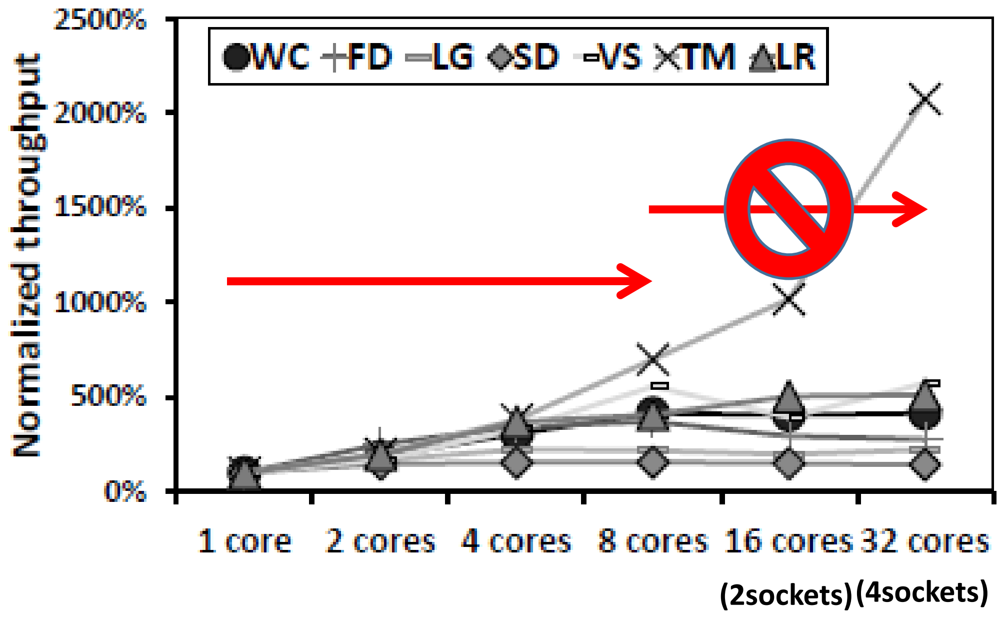
# Profiling Outline

- ❖Scalability on varying number of CPU cores
- ❖Execution time breakdown of overall execution
- ❖Study the impact of common designs

# Scalability on varying number of cores/sockets



(a) Storm



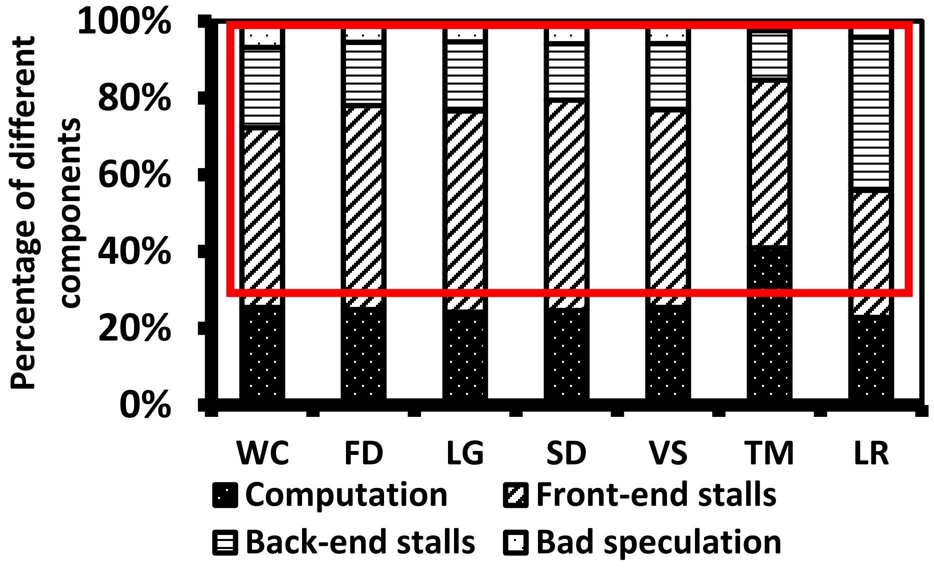
(b) Flink

Scale poorly on multiple sockets → Overhead > Additional resource benefits

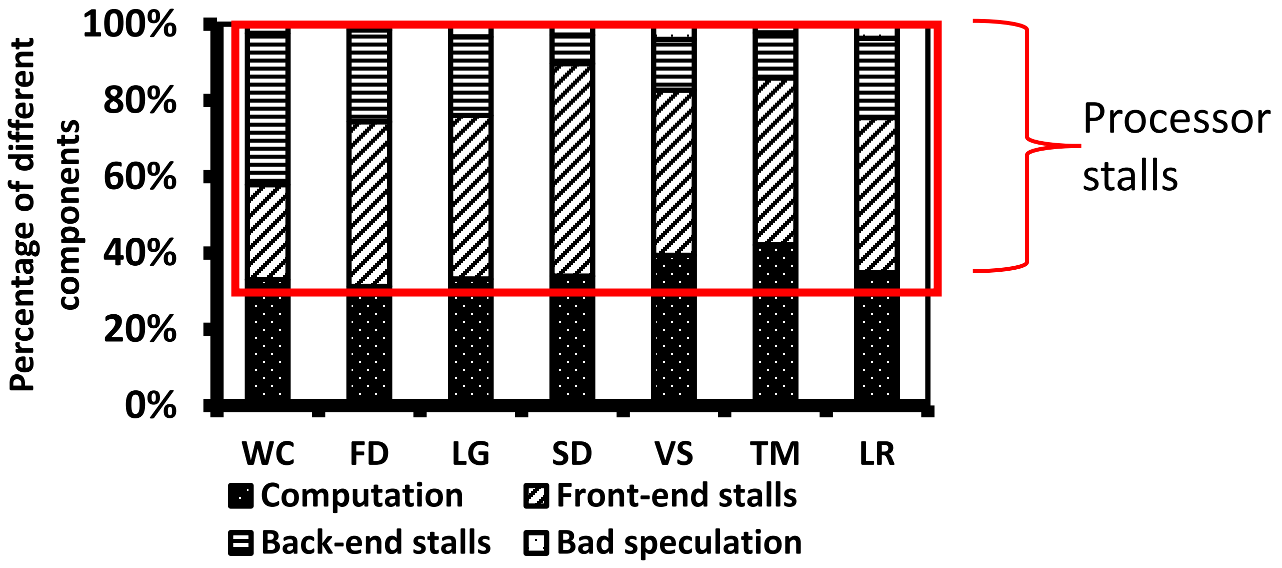
## Profiling Outline

- ❖Scalability on varying number of CPU cores
- ❖Execution time breakdown of overall execution
- ❖Study the impact of common designs

# Are there any problems when running on a single socket?



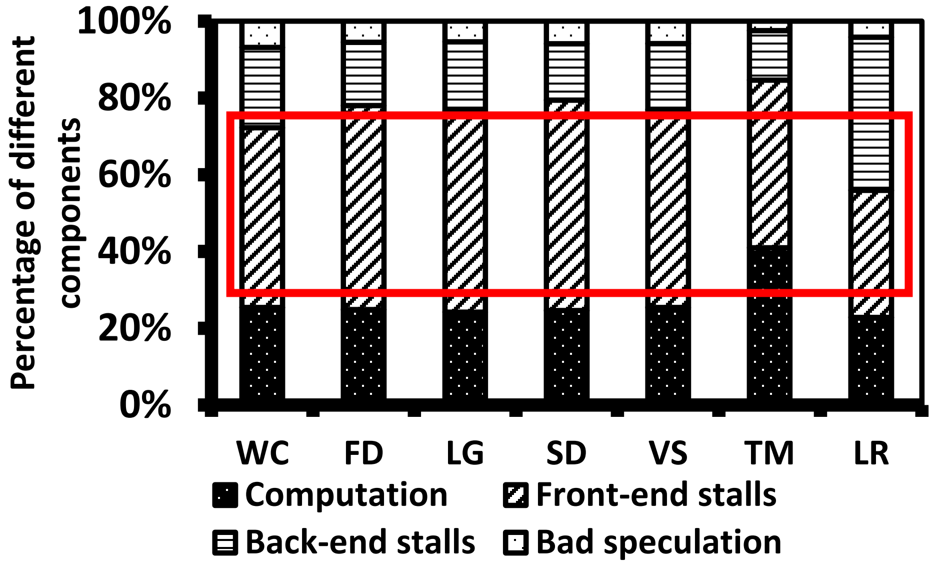
(a) Storm



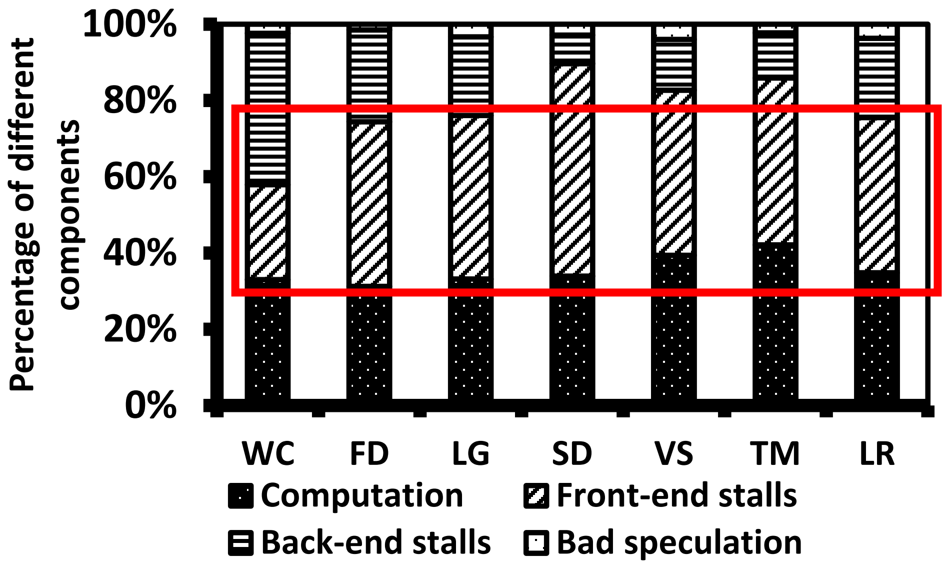
(b) Flink

70% of the execution times are spent in processor stalls.

## Are there any problems when running on a single socket?



(a) Storm



(b) Flink

Front-end stalls is a major bottleneck.

## Profiling Outline

❖Scalability on varying number of CPU cores

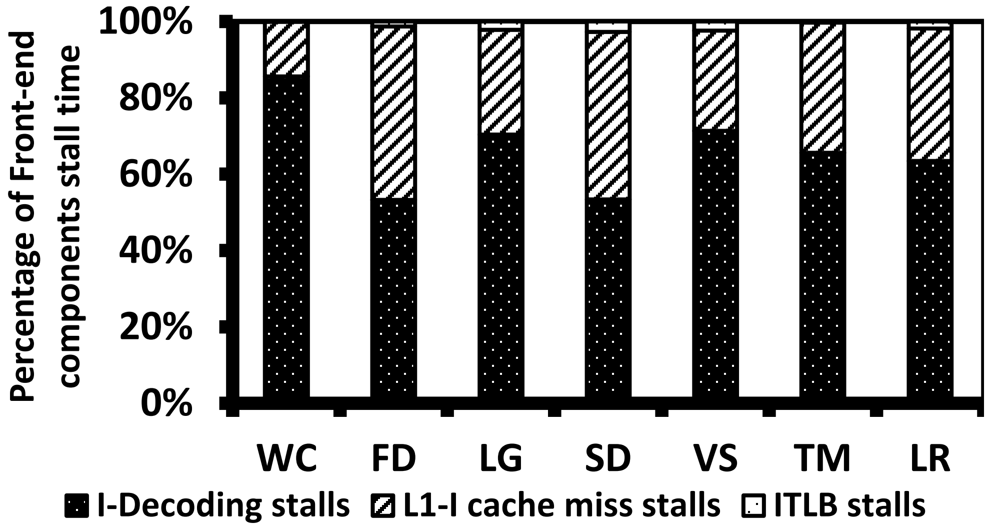
❖Execution time breakdown of overall execution

❖Study the impact of common designs

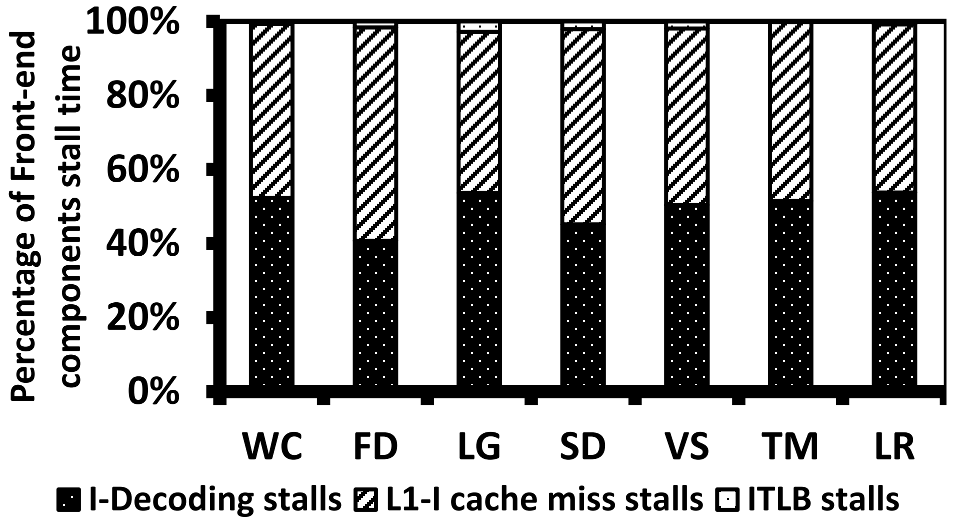
a) Pipelined processing with message passing
b) On-demand data parallelism
c) JVM based implementation

# Impact of pipelined and data parallel processing model

- The design of supporting both pipelined and data parallel processing results in a very complex massively parallel threading model.
- *This threading model poorly utilizes instruction cache.*

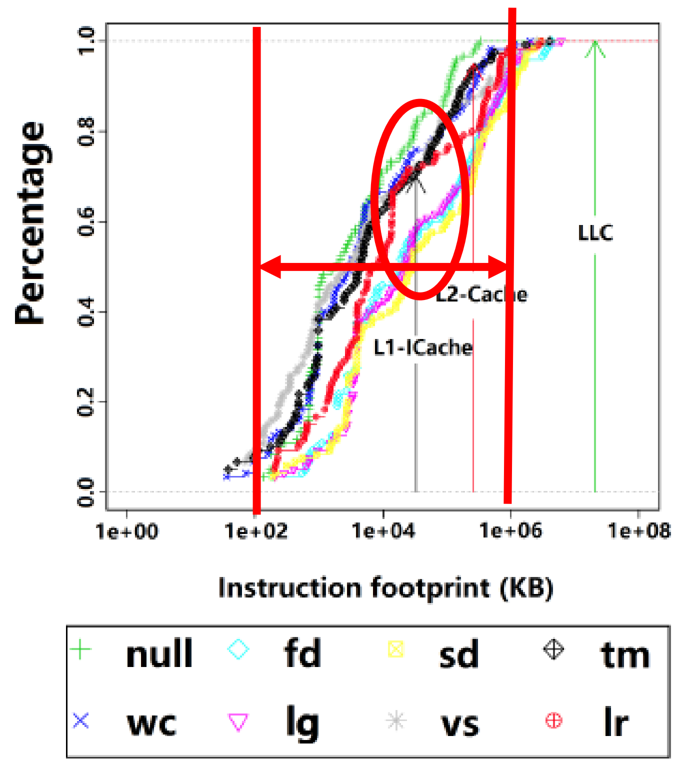


(a) Storm

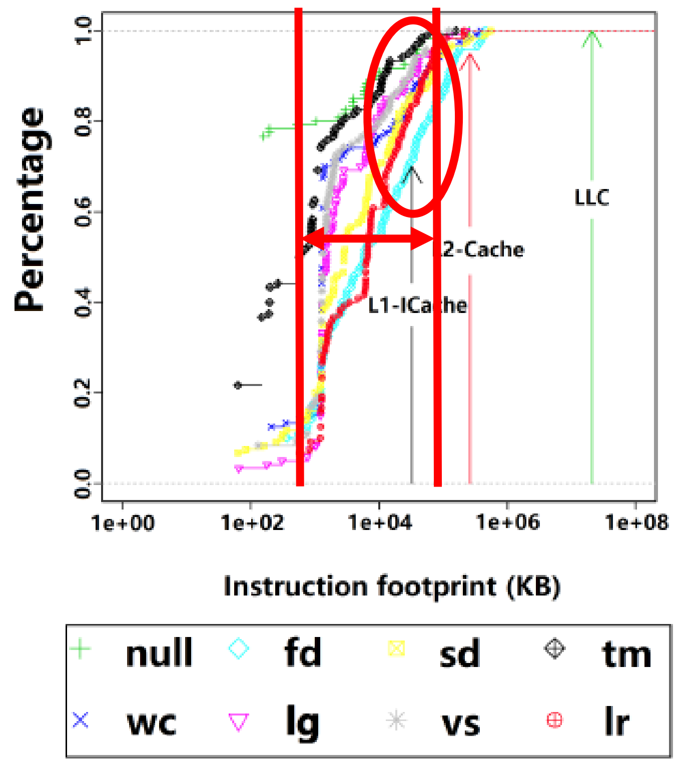


(b) Flink

# Instruction footprint between two consecutive invocations of the same function



(a) Storm



(b) Flink

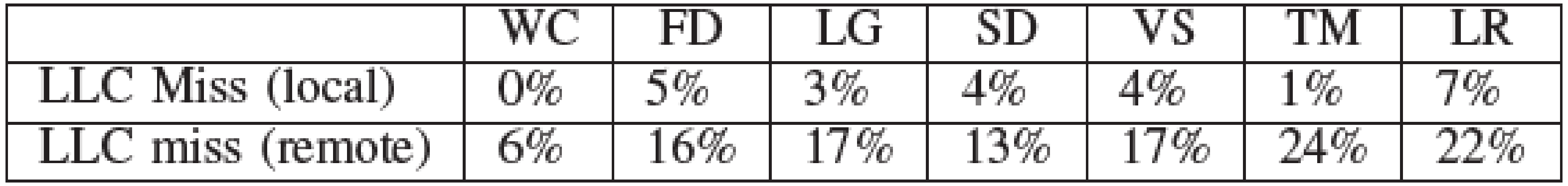
(i) Common range of their instruction footprints is between 1KB to 10MB and 1KB to 1MB

(ii) 50~70% and 20~40% of the instruction footprints are larger than the L1-ICache

## Impact of message passing and stream partitioning

- ❖Message passing between operators scheduled to different sockets comes with remote memory access (RMA).
- ❖RMA overhead prevents DSP systems from scaling well on multi-socket processors.

Table: LLC miss stalls when running Storm with four CPU sockets



| | WC | FD | LG | SD | VS | TM | LR |
|---|---|---|---|---|---|---|---|
| LLC Miss (local) | 0% | 5% | 3% | 4% | 4% | 1% | 7% |
| LLC miss (remote) | 6% | 16% | 17% | 13% | 17% | 24% | 22% |

Up to 24% of the total execution time are wasted due to remote memory access

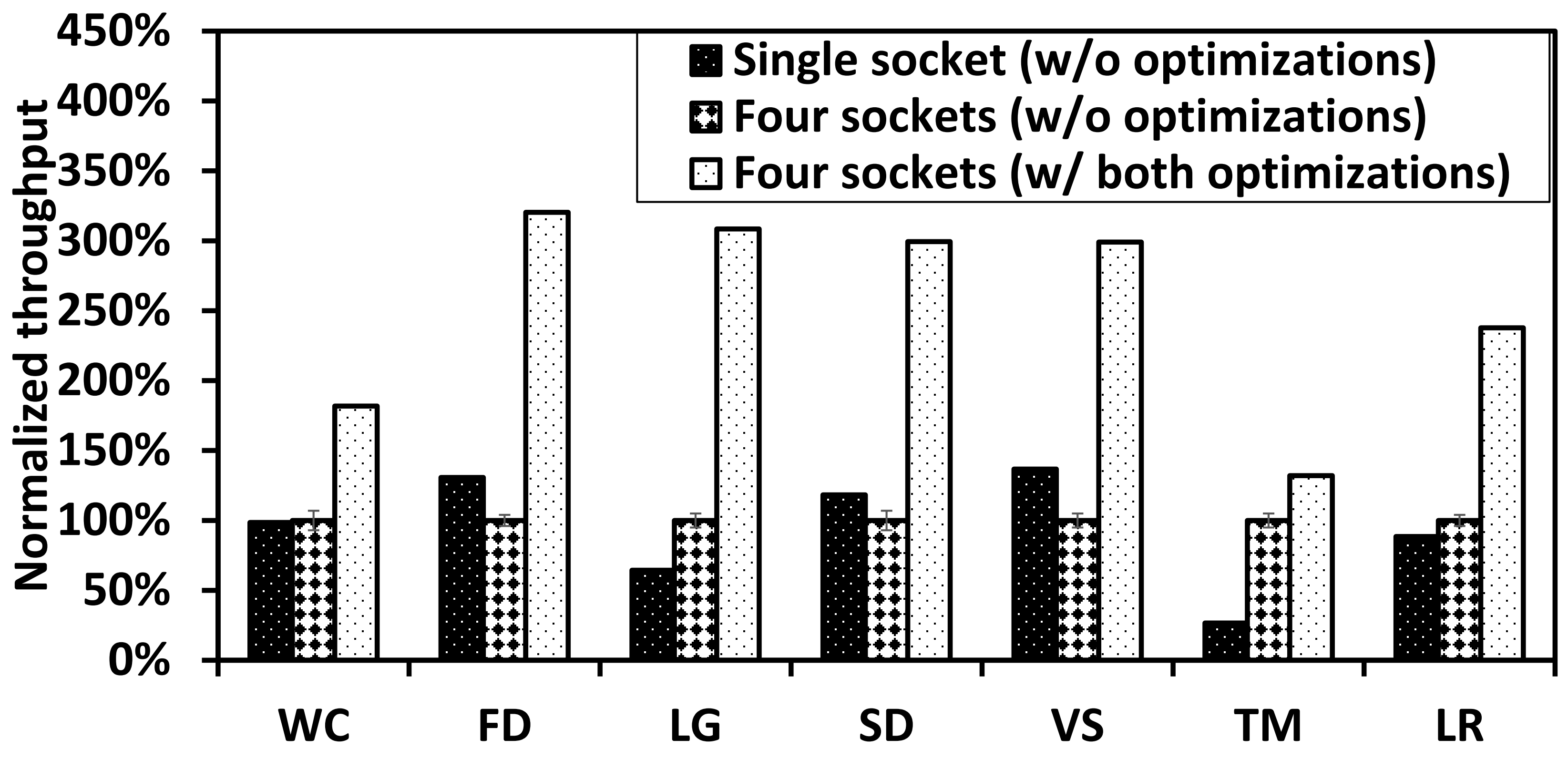
## Key Findings (recap)

- ❖Large instruction footprint between two consecutive invocations of the same function causes significant L1-Icache misses.
- ❖The current message passing design overlooks NUMA effect and results in serious performance degradation due to significant remote memory access overhead.

## Towards more efficient DSP systems

- ❖Non-blocking tuple batching
  - ❖Goal: to reduce instruction cache misses
  - ❖Batch and only batch tuples that can be batched
- ❖NUMA-aware executor placement
  - ❖Goal: to reduce remote memory accesses
  - ❖Relies on solving the min-cut graph partition problem on the executor graph

# Evaluate combining both techniques on Storm



Our optimizations can significantly improve Storm (up to 3.2x).

## Profiling study summary

- ❖There is a lack of detailed analysis on the common design aspects of modern DSP systems when running on modern multi-socket multi-core processors.
- ❖We have designed micro-benchmark, conducted extensive evaluations based on two state-of-the-art DSP systems, and identified several common performance issues.
- ❖We have designed two optimization techniques to address the found performance issues and demonstrated promising performance improvements

## Outline

- ❖Background.
- ❖Profiling study of DSP systems on modern multi-core processors.
- ❖**A new DSP system designed for modern multi-core processors.**
    - ❖**On-going work**

# BriskStream: NUMA-aware Data Stream Processing

A quick glance of my recent work

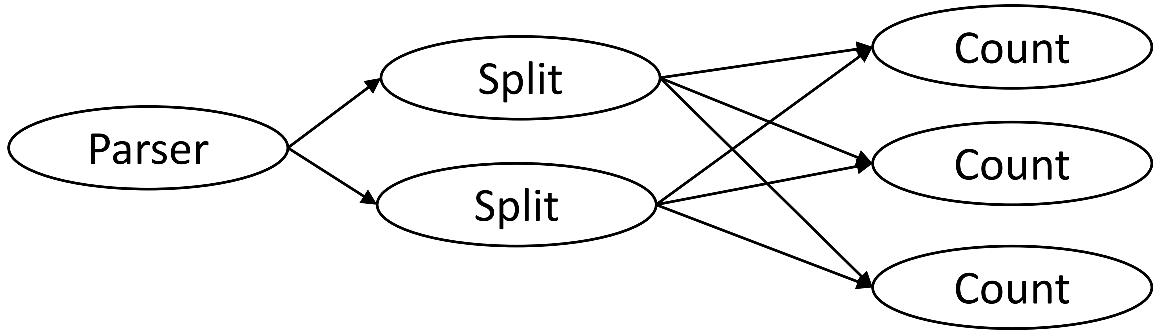
NUS
National University of Singapore

## The need of a ``scale-up” DSP system

- Applications tend to more complex, expressed as larger diameter graph structure.
  - Cross machines communication is often the performance killer, especially considering latency.
- Modern multi-core processors have demonstrated superior performance.
  - Hundreds of cores and terabytes of memory are available in a single instance.

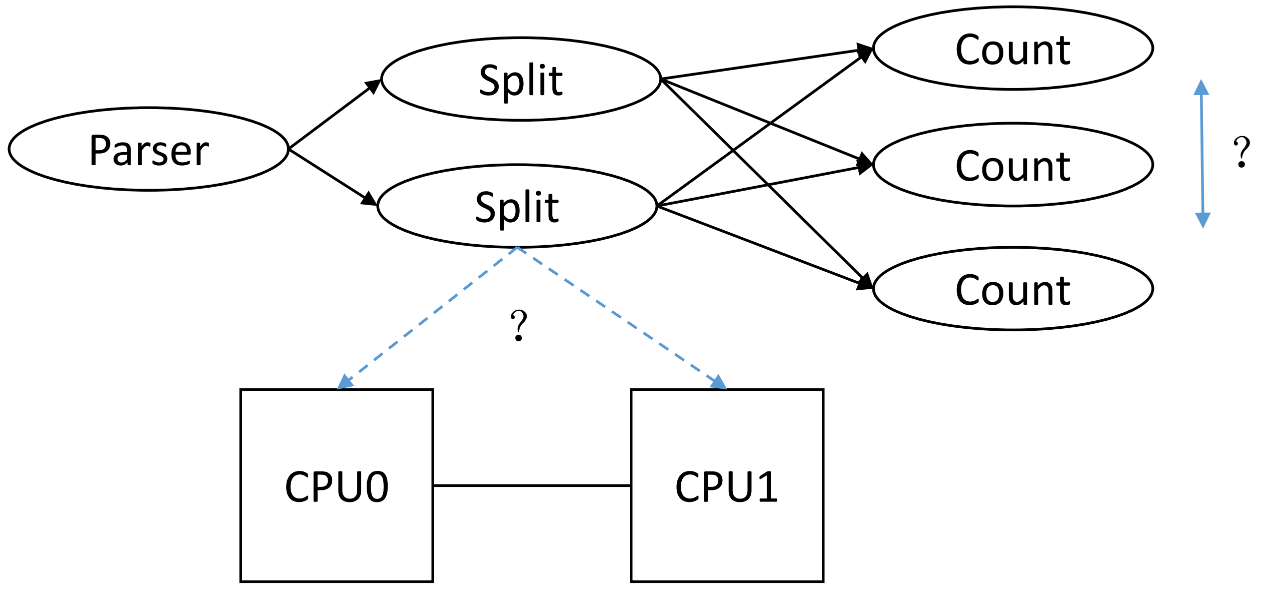
# Data Stream Processing on NUMA servers

- An application is expressed as a directed acyclic graph (DAG).
  - Pipeline execution: each vertex corresponds to one operator, which executes independently and edge indicates message passing between operators.
  - Data parallel execution: each operator may be replicated to increase throughput.

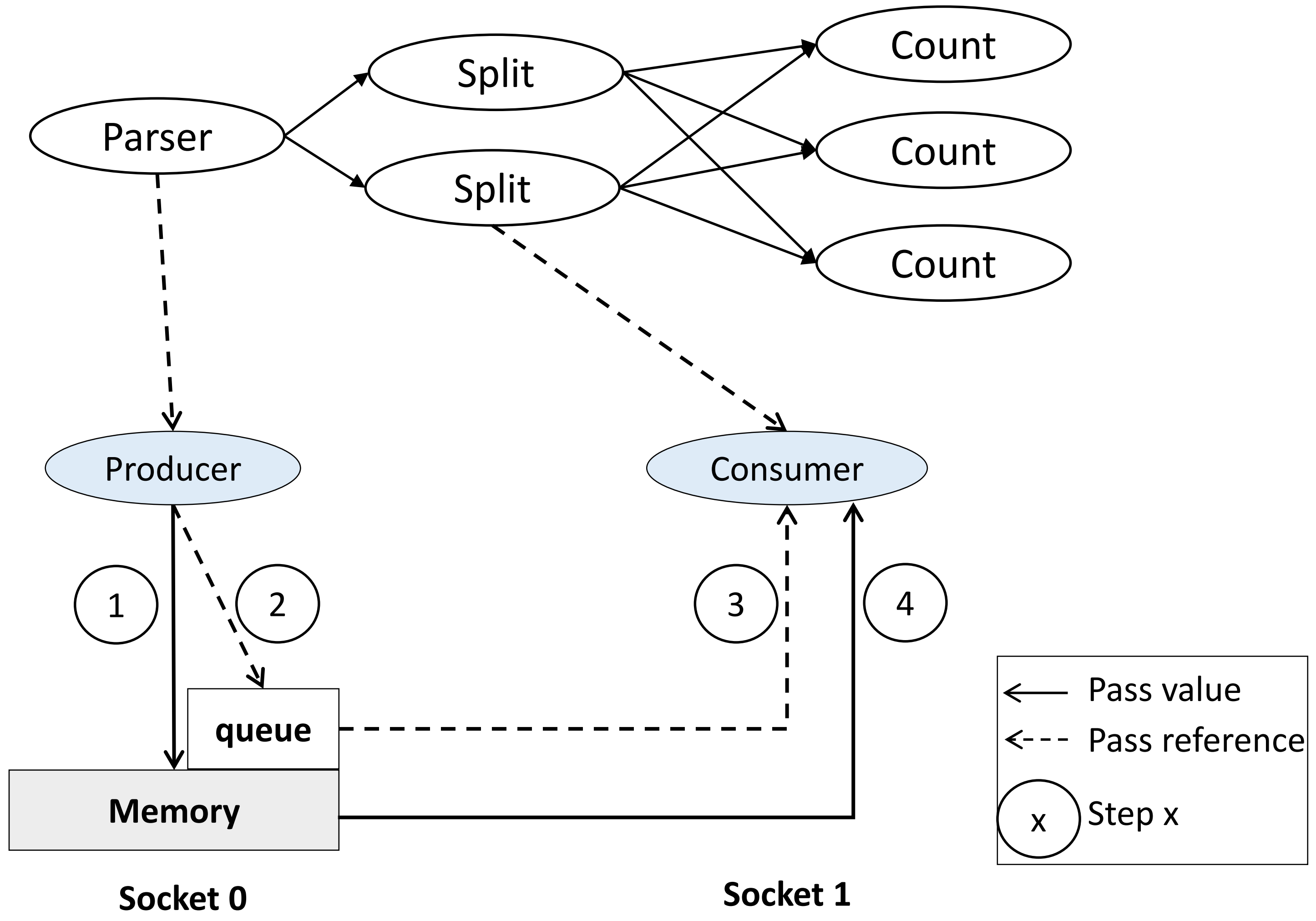


# Optimize Data Stream Processing on NUMA servers

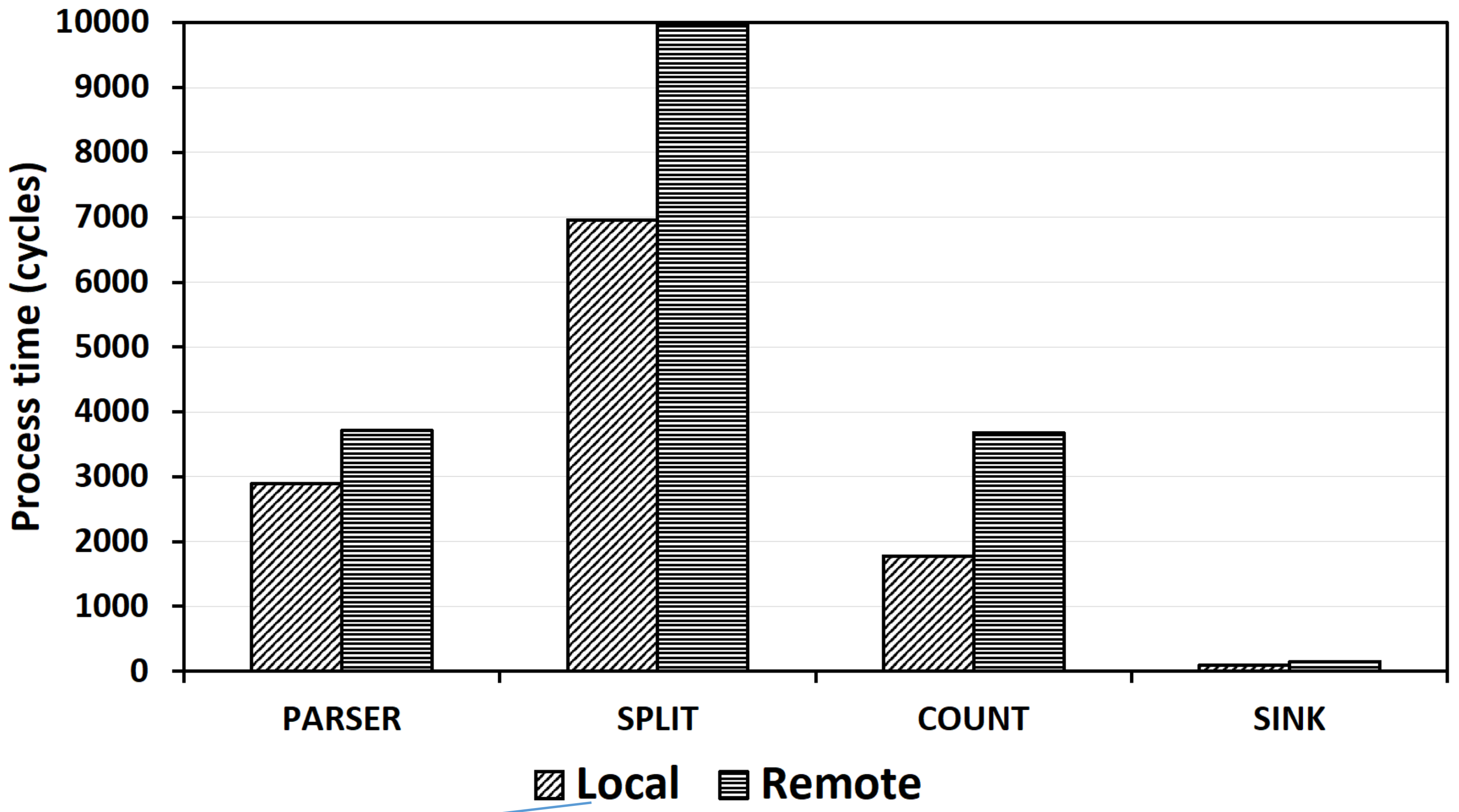
❖We need to determine both the level of operator replication and the placement of operators to optimize the execution (e.g., maximize throughput).



# Message passing



# Relative location matters



Sit together in the same CPU socket with its producer

1-hop away from its producer

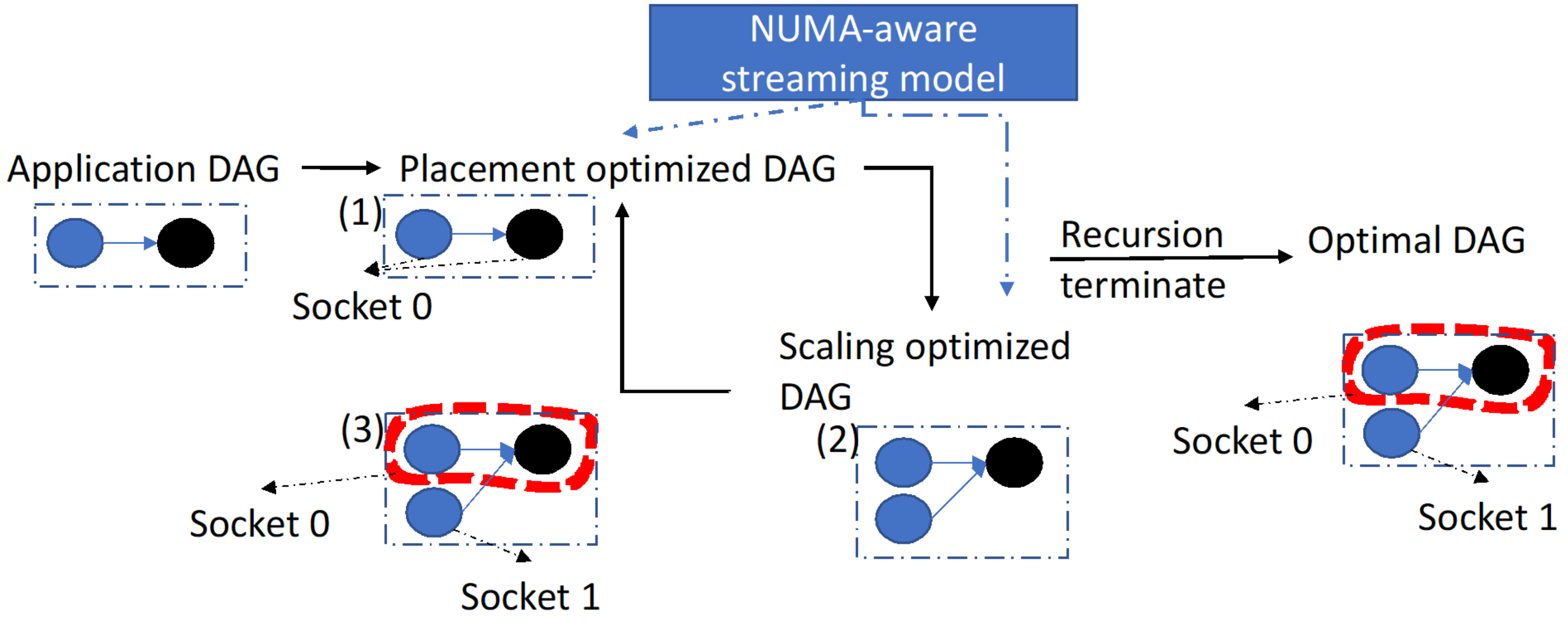
## Insufficiency of existing approaches

- NUMA is between UMA and distributed environment.
  - Cross site (i.e., cross sockets) communication cost can be significantly reduced thanks to shared-memory optimizations.

    -> traffic-minimization heuristics fall short.
  - But, it still creates differences among accessing cost to different CPU sockets.

    -> the optimization needs to be NUMA-aware to ensure optimality.
- We propose a novel NUMA-aware optimization framework for DSP systems.

# Replication and Location Aware Scheduling


NUMA-aware streaming model
Application DAG
Placement optimized DAG
(1)
Socket 0
Recursion
terminate
Optimal DAG
Scaling optimized DAG
(2)
(3)
Socket 0
Socket 1
Socket 0
Socket 1