# BriskStream: Scaling Data Stream Processing on Shared-Memory Multicore Architectures

Shuhao Zhang[*1], Jiong He[2], Amelie Chi Zhou[3], Bingsheng He[1]

1 National University of Singapore

2 ADSC Illinois at Singapore Pte Ltd

3 Shenzhen University

*Work done while as research trainee at SAP Singapore.

# Importance of Data Stream Processing

Apache Storm

Twitter Heron

Apache Flink

…

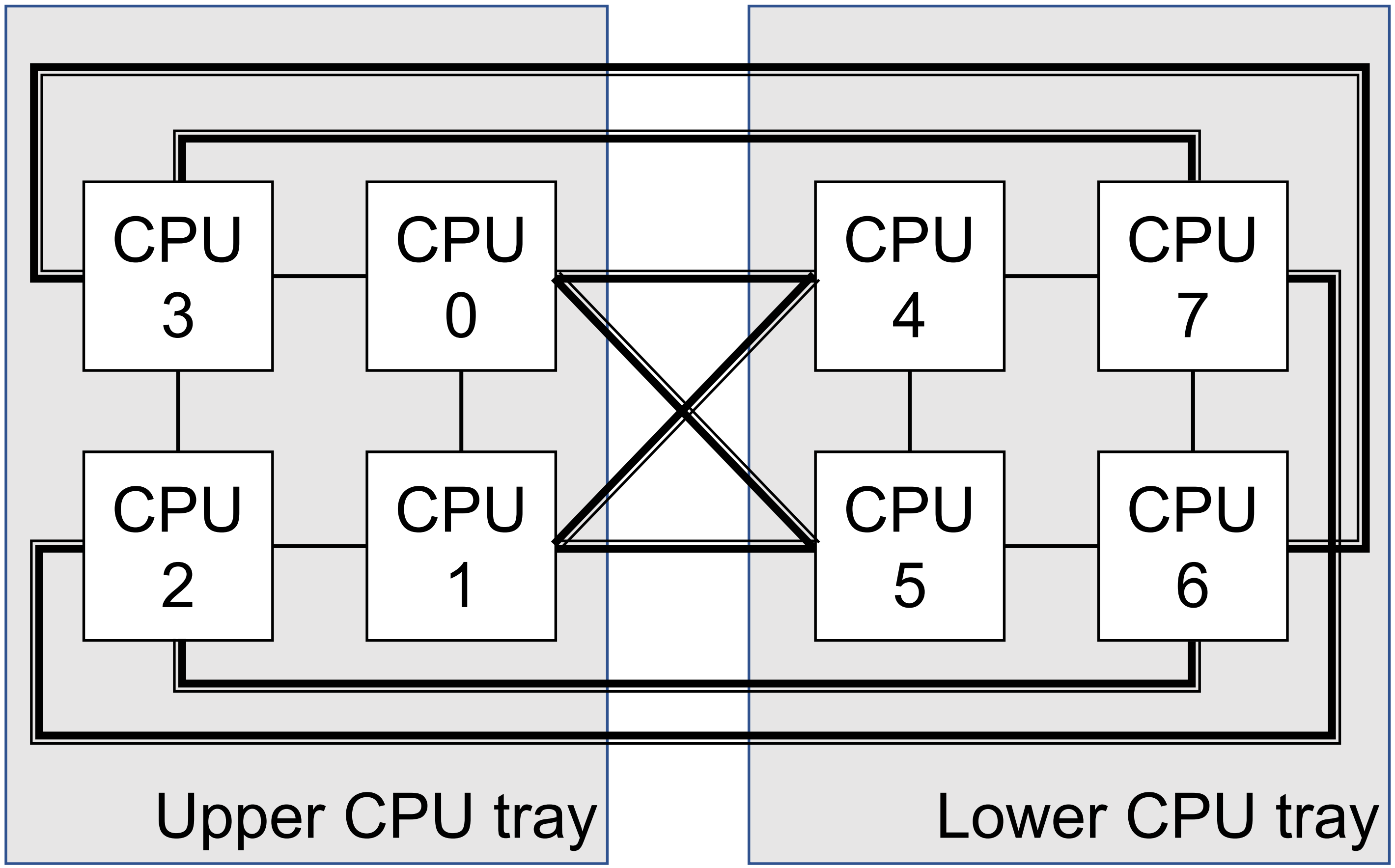
# DSPS on Modern Hardware

- **Multicore architectures are attractive platform for DSPSs.**

- **However, fully exploiting its computation power can be challenging.**

- **This work focuses on NUMA-awareness.**

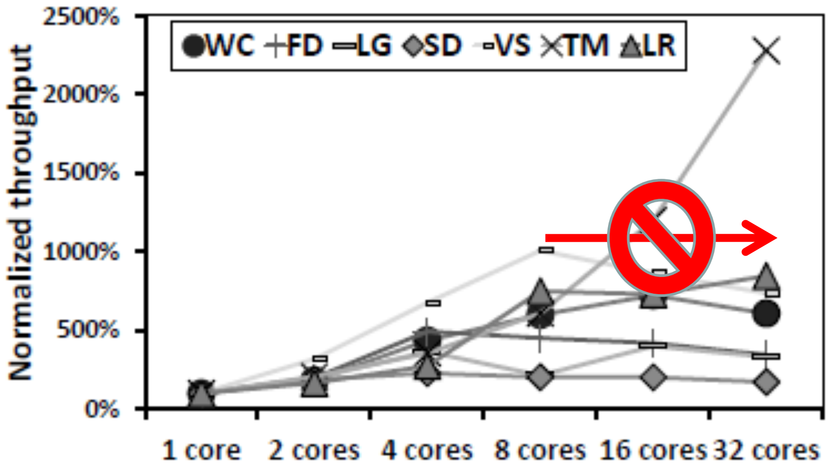
# NUMA (non- uniform-memory-access) Servers




Server A: HUAWEI KunLun 8×18 Cores (w/o HT) @1.2GHz

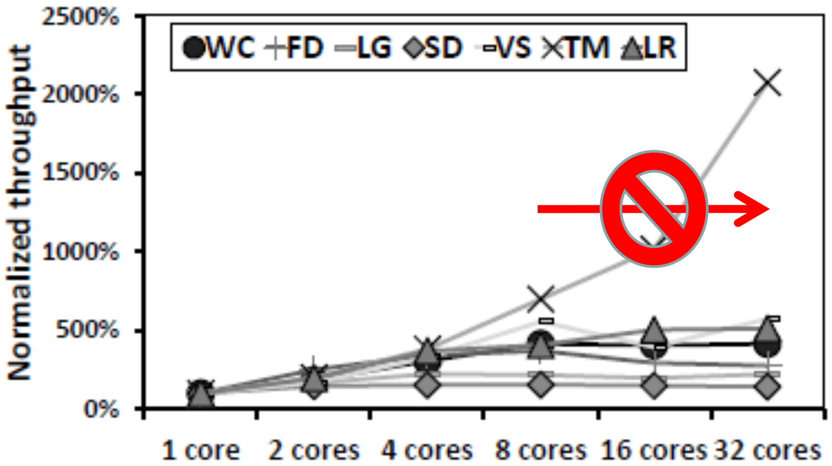
144 cores in one machine

# Scalability on varying number of cores/sockets



(2sockets)(4sockets)

(a) Storm

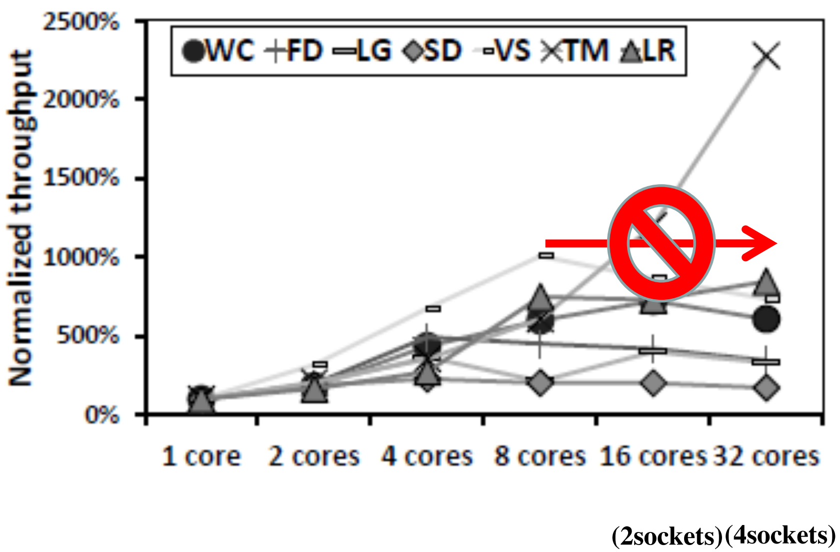


(2sockets)(4sockets)

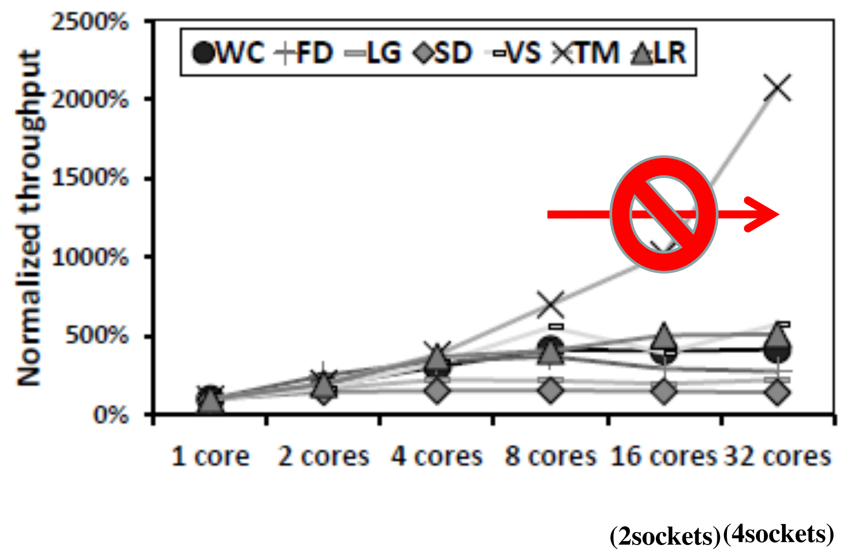
(b) Flink

Revisiting the Design of Data Stream Processing Systems on Multi-Core Processors, Zhang et al. ICDE'17

# Scalability on varying number of cores/sockets




(a) Storm

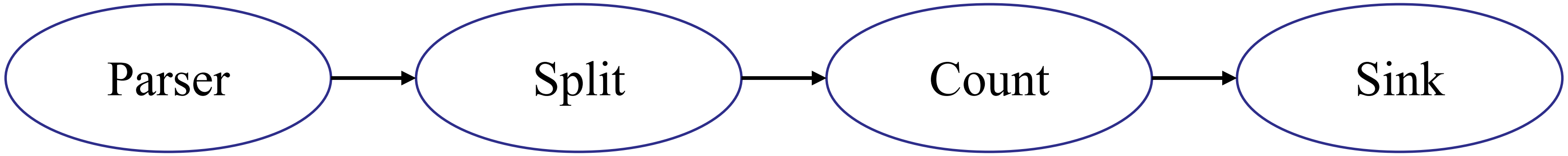



(b) Flink

Revisiting the Design of Data Stream Processing Systems on Multi-Core Processors, Zhang et al. ICDE'17

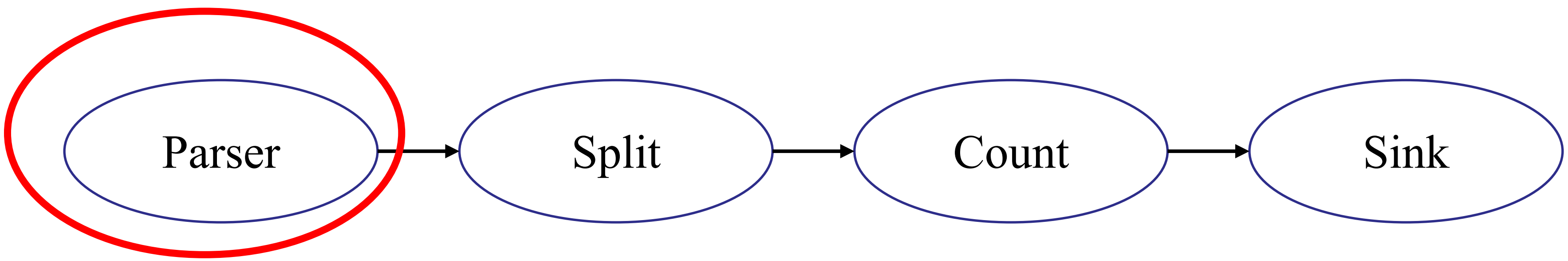
How can we maximize the throughput of a stream application on a NUMA machine (limited HW resources)?

# Stream processing

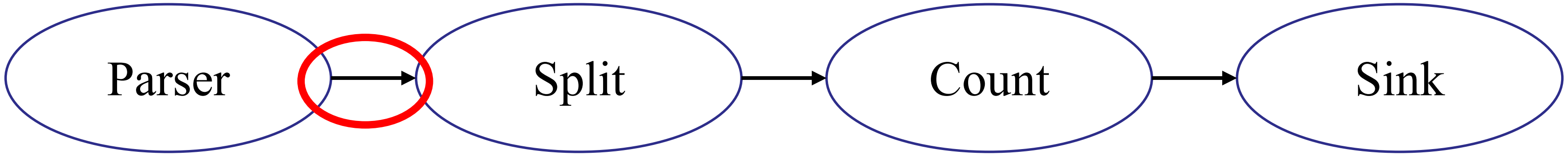



Word-count (WC) application

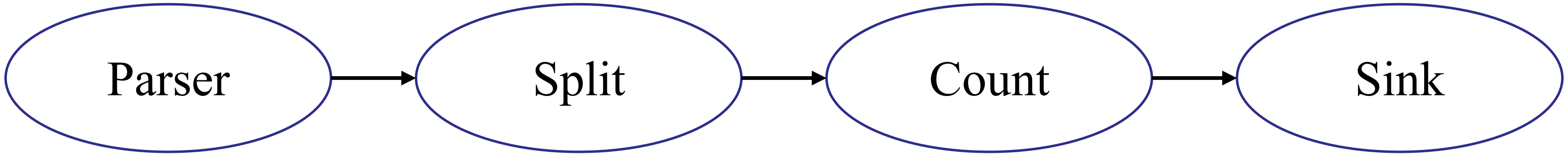
# Stream processing



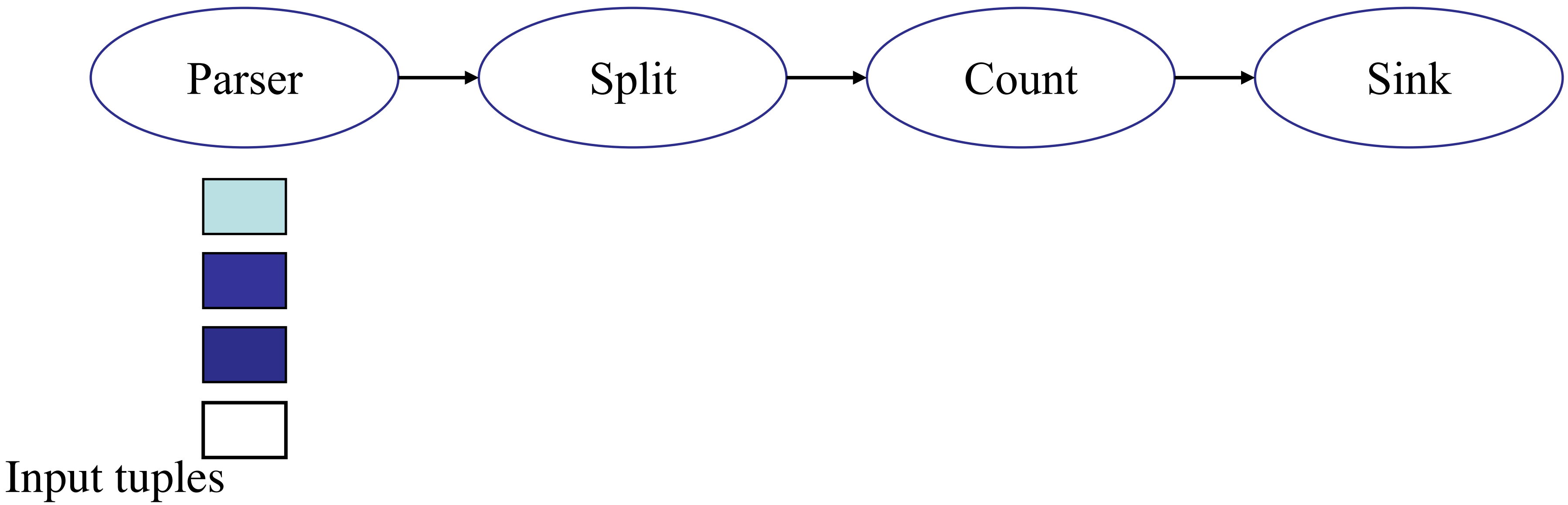
# Stream processing



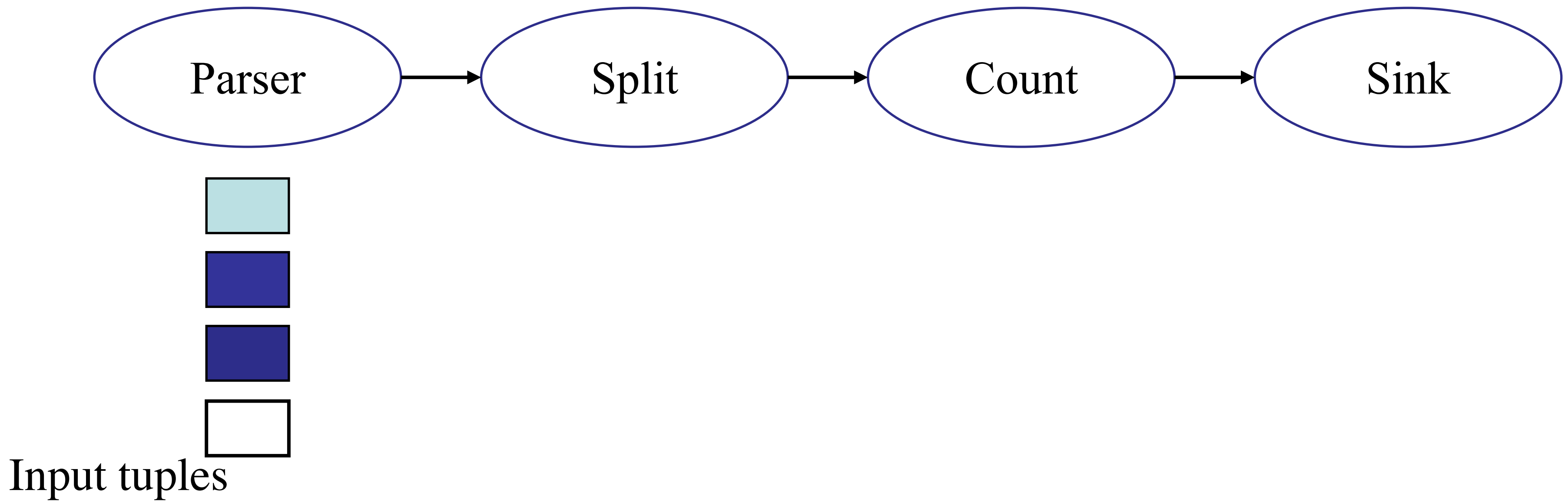
# Stream processing



# Stream processing



# Stream processing



# Problem formulation

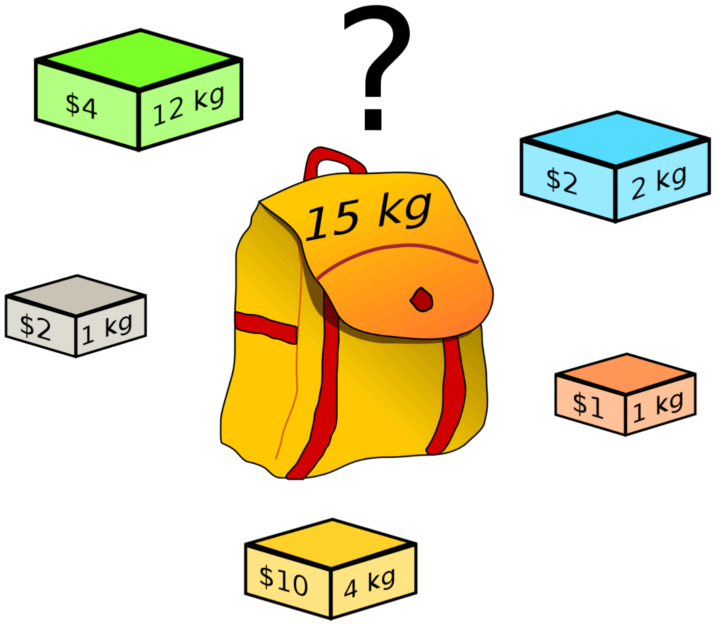
# Problem formulation

- **Each operator may be carried in multiple threads, and each thread can be allocated at any CPU socket.**

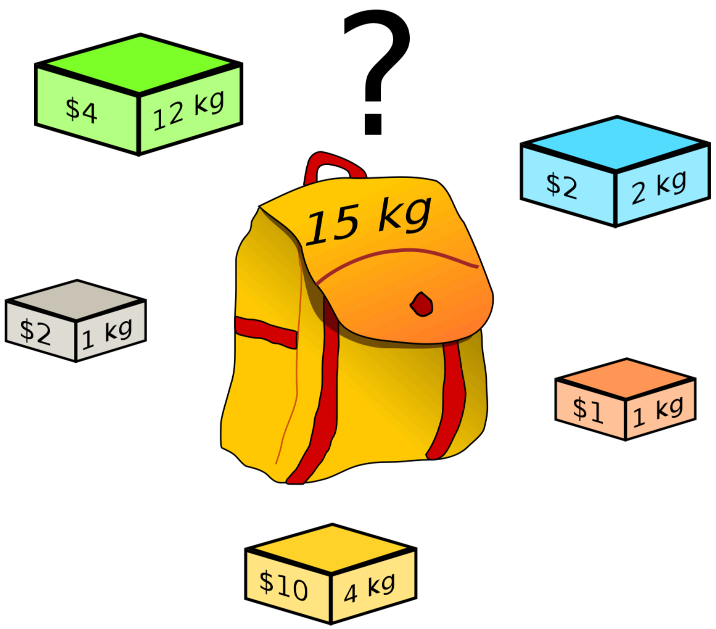
# Problem formulation

- **Each operator may be carried in multiple threads, and each thread can be allocated at any CPU socket.**
  - **Scaling optimization**: determine suitable parallelism level of each operator
  - **Placement optimization**: determine suitable placement of each thread.

# Operator placement as Knapsack?

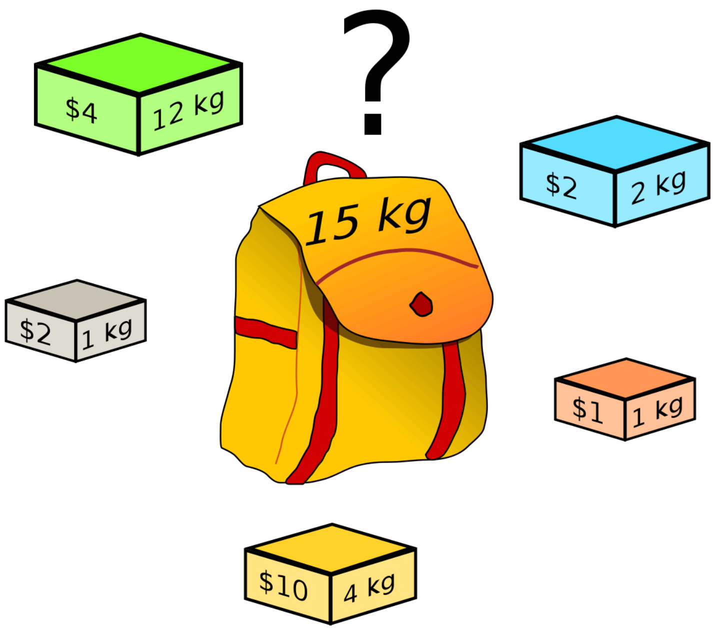


# Operator placement as Knapsack?



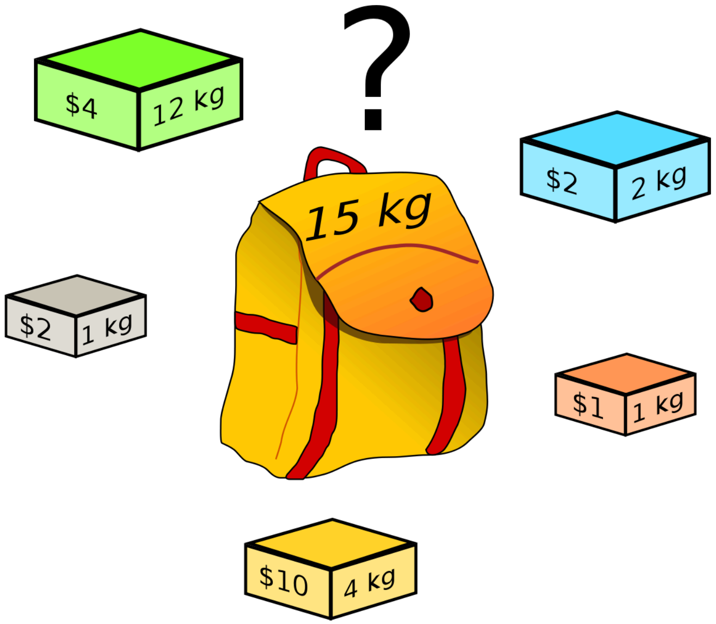

- **Bags – CPU sockets**
  - Capacity: resource availability
- **Items – operators**
  - Value: operator throughput
  - Weight: resource demand

# Operator placement as Knapsack?




- **Bags – CPU sockets**
  - Capacity: resource availability
- **Items – operators**
  - Value: operator throughput
  - Weight: resource demand

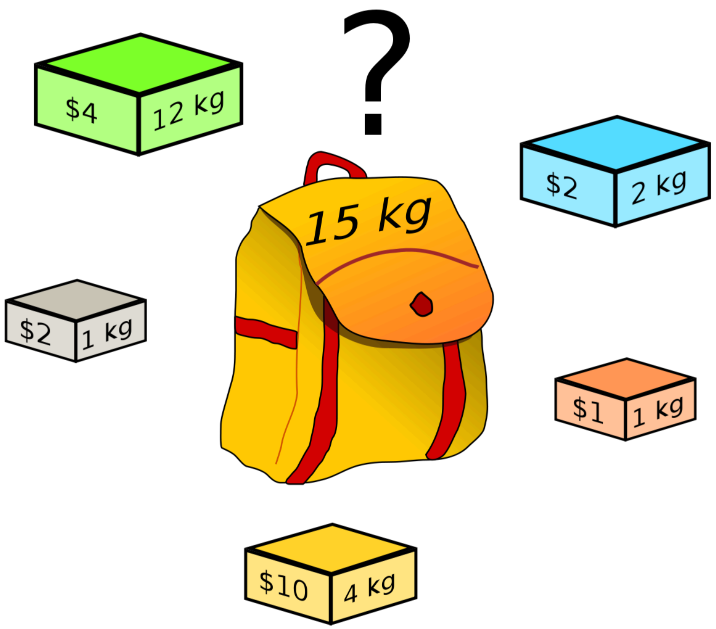
# Operator placement as Knapsack?




- **Bags – CPU sockets**
  - Capacity: resource availability
- **Items – operators**
  - Value: operator throughput
  - Weight: resource demand

Maximize value (throughput) under capacity constraint.

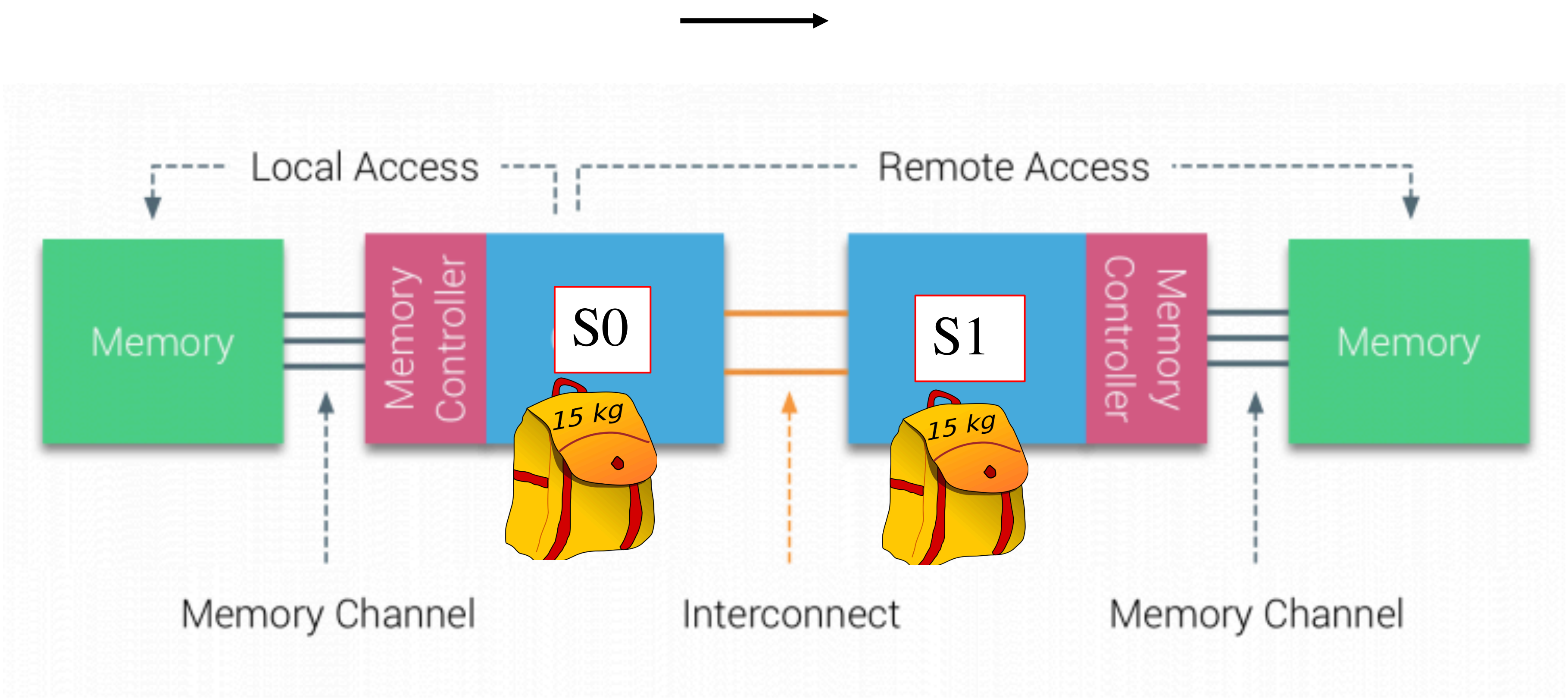
# Operator placement as Knapsack?



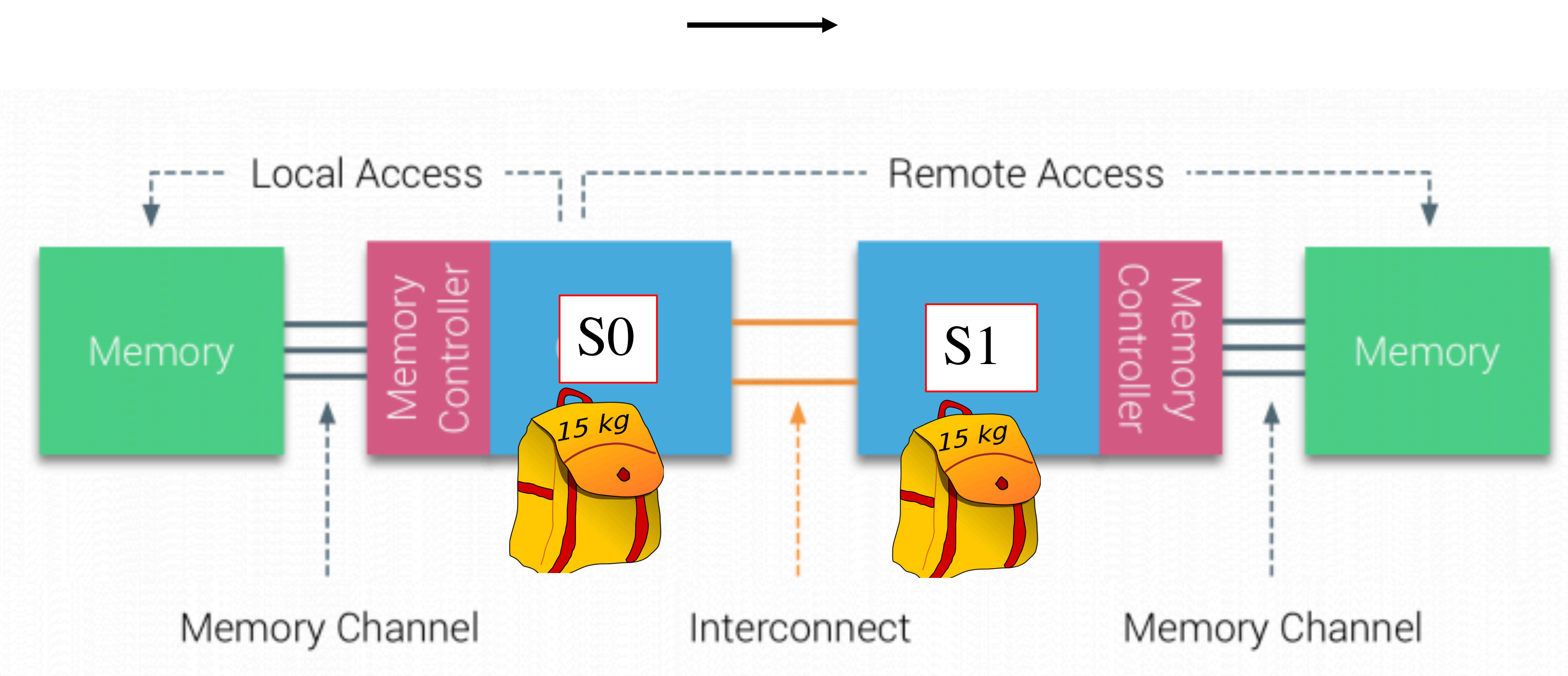

- **Bags – CPU sockets**
  - Capacity: resource availability
- **Items – operators**
  - Value: operator throughput
  - Weight: resource demand

Maximize value (throughput) under capacity constraint.

# Differences and Challenges

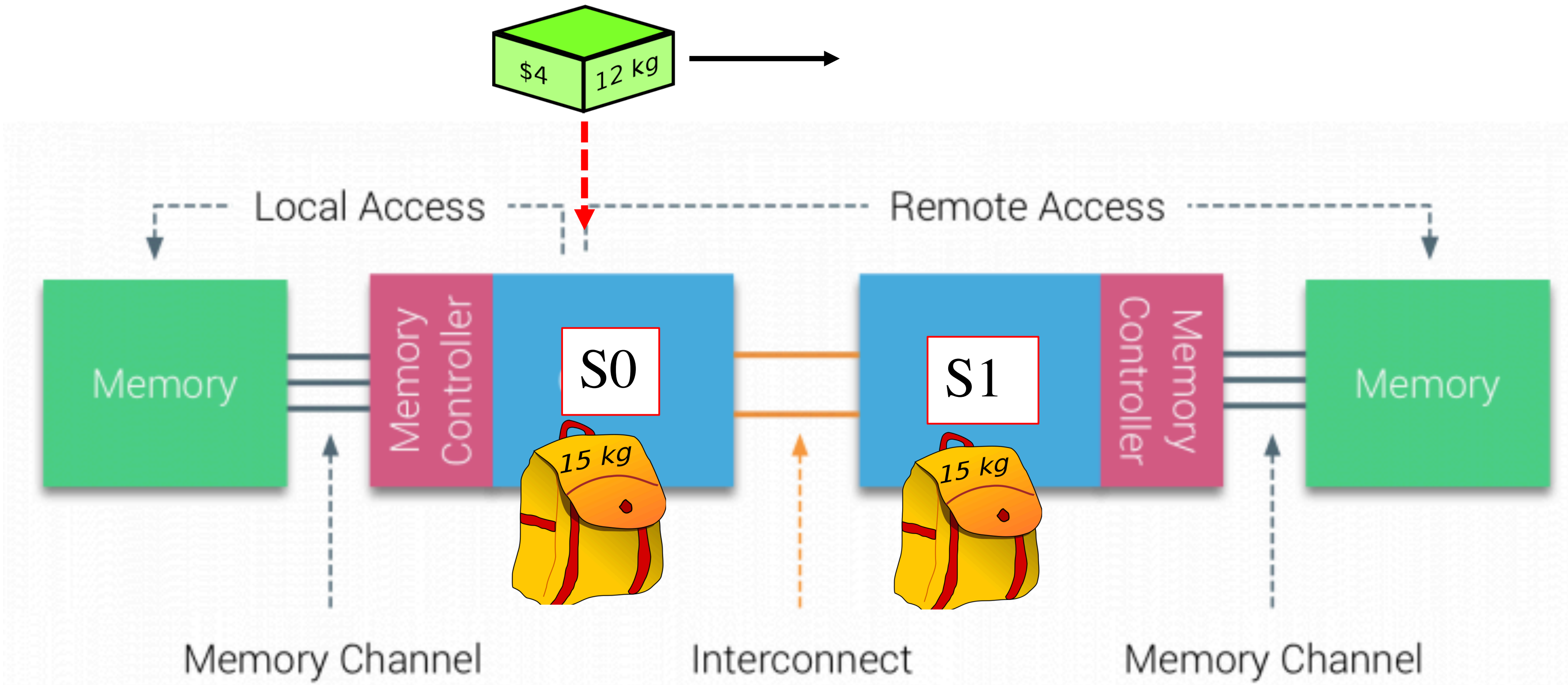


# Differences and Challenges



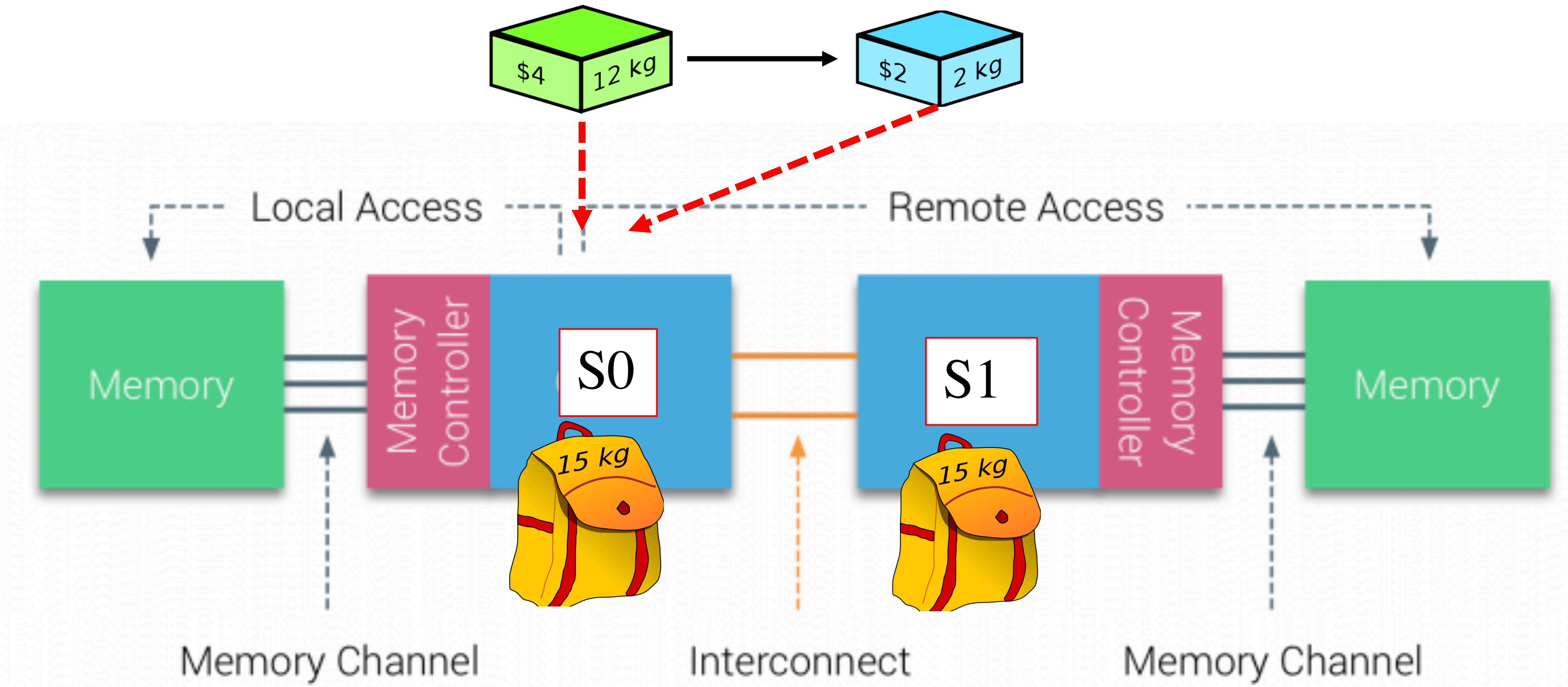

The difference is that both value and weight of each item (operator) are **variables** depending on packing (placement) decisions due to the NUMA effect.

# Differences and Challenges



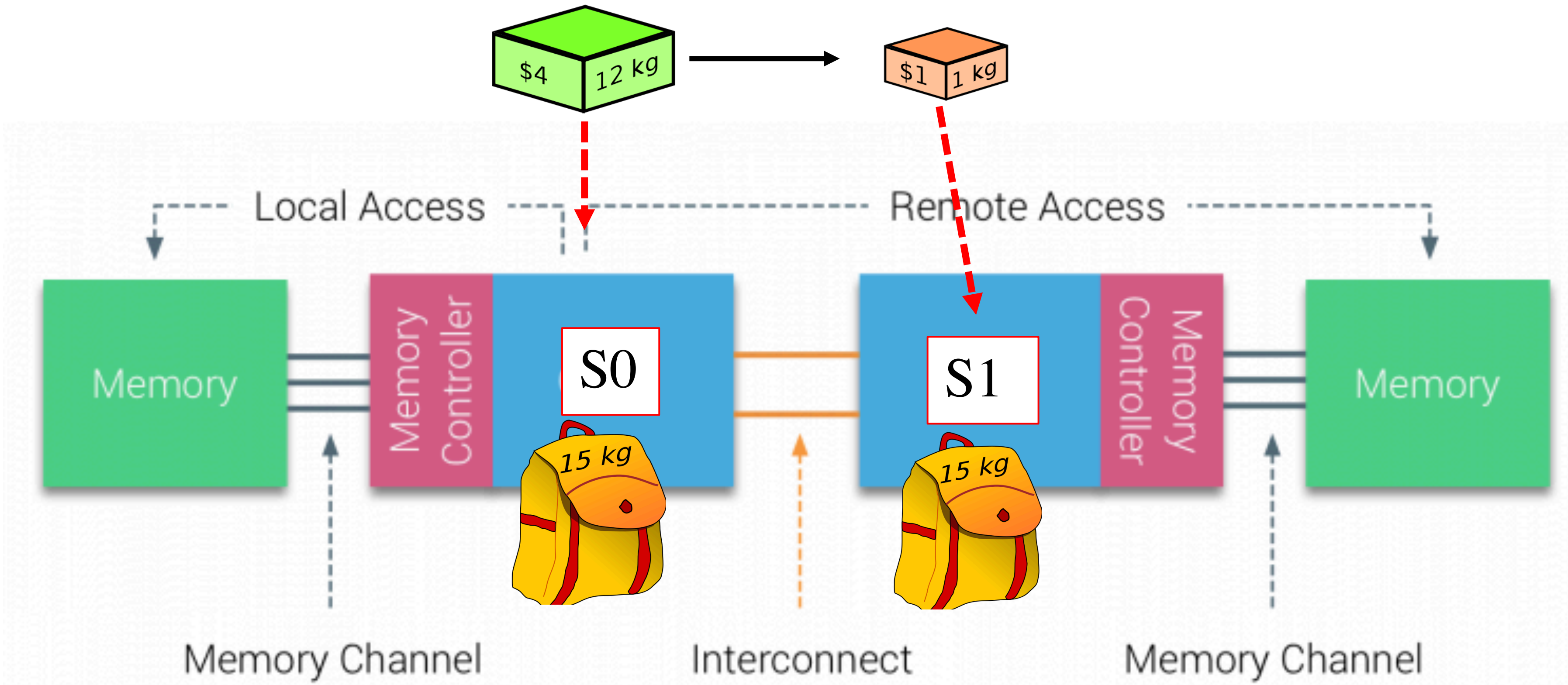

The difference is that both value and weight of each item (operator) are **variables** depending on packing (placement) decisions due to the NUMA effect.

# Differences and Challenges



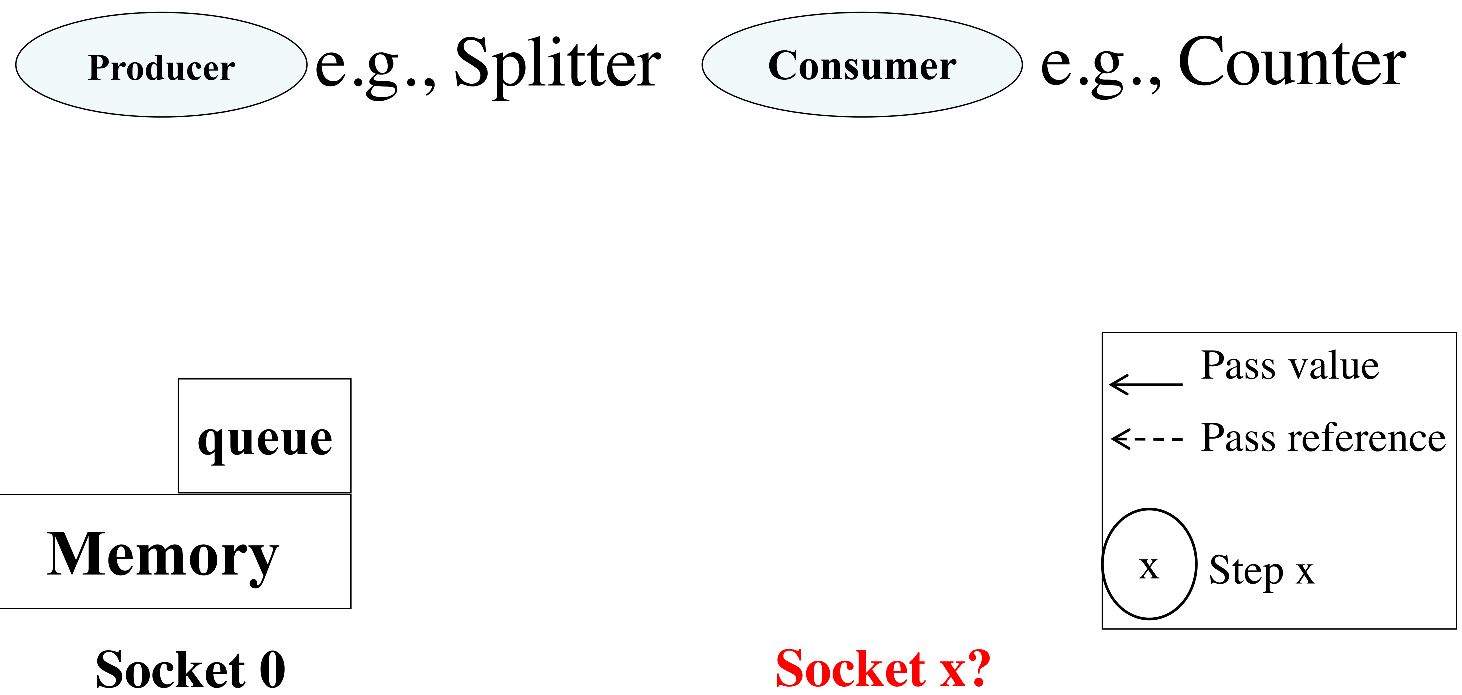

The difference is that both value and weight of each item (operator) are **variables** depending on packing (placement) decisions due to the NUMA effect.

# Differences and Challenges

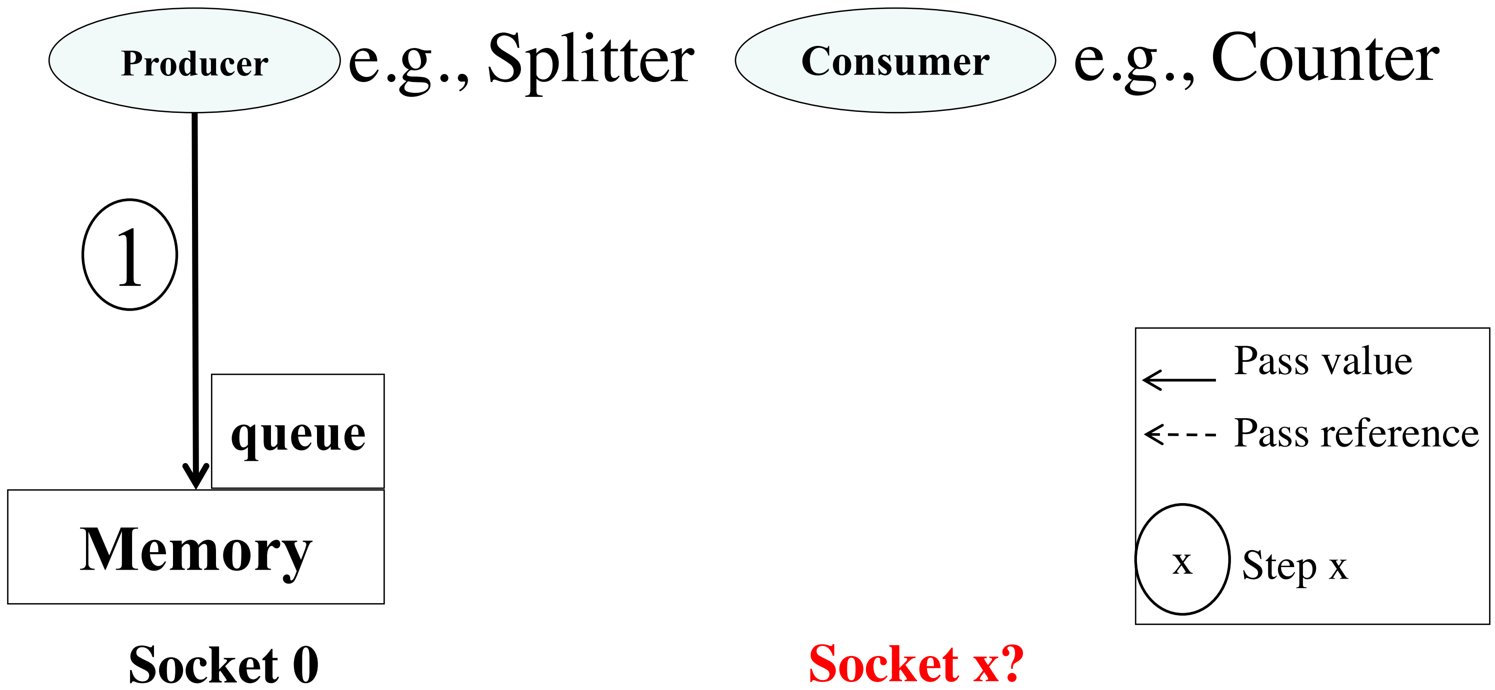



The difference is that both value and weight of each item (operator) are **variables** depending on packing (placement) decisions due to the NUMA effect.

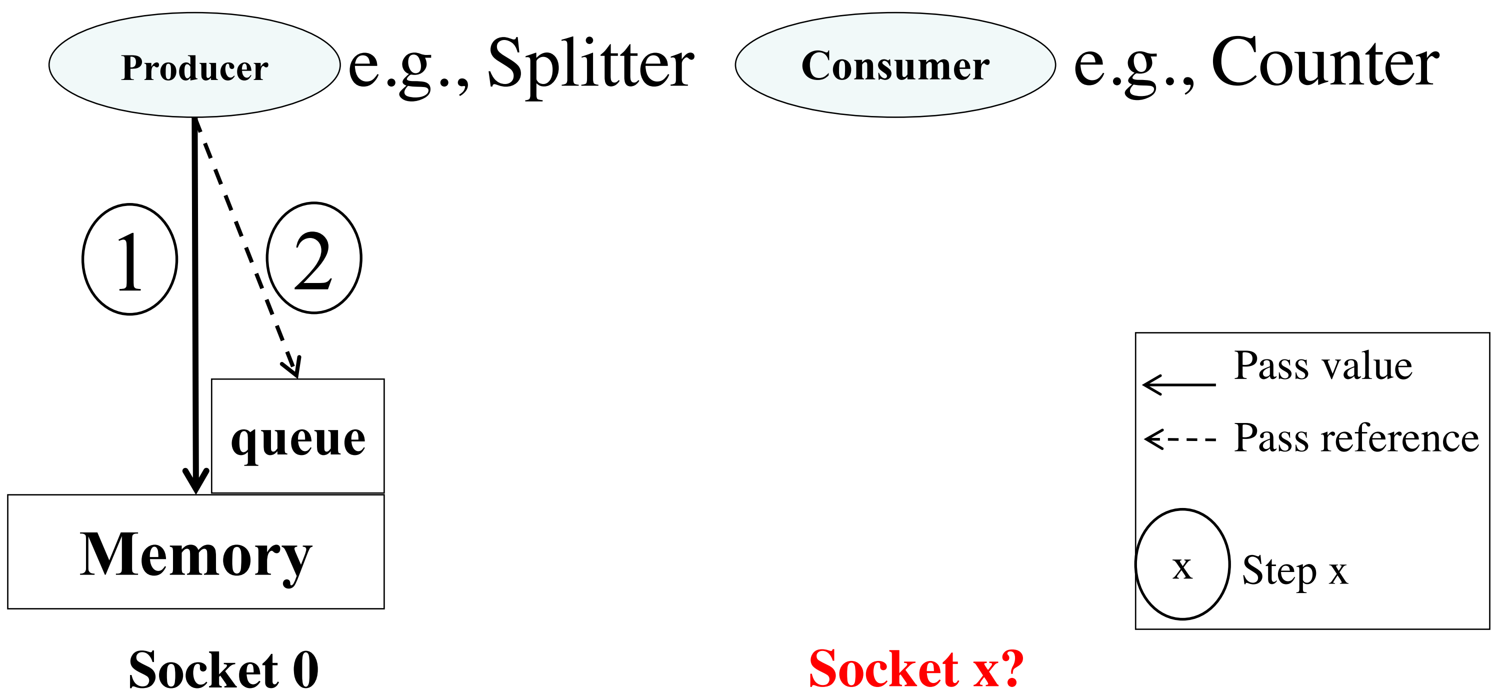
# Zoom into the current system design



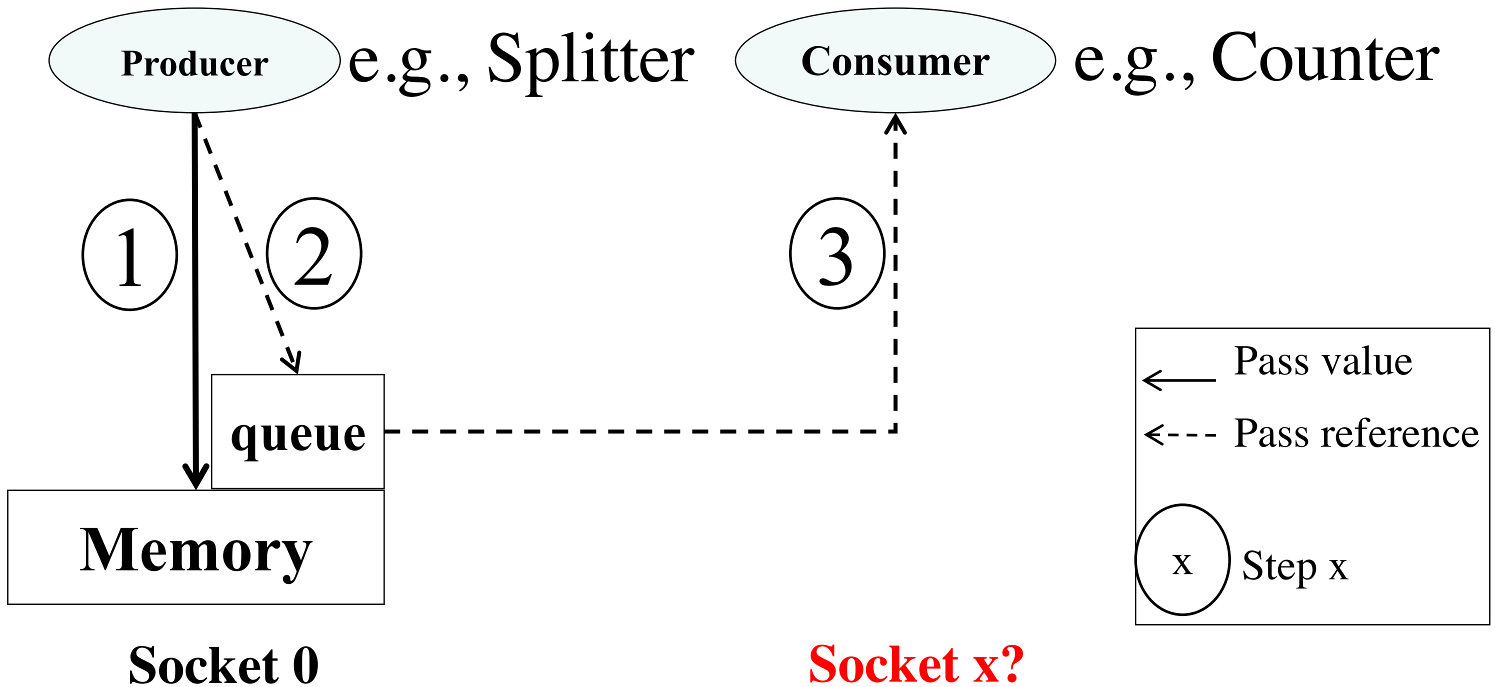
# Zoom into the current system design



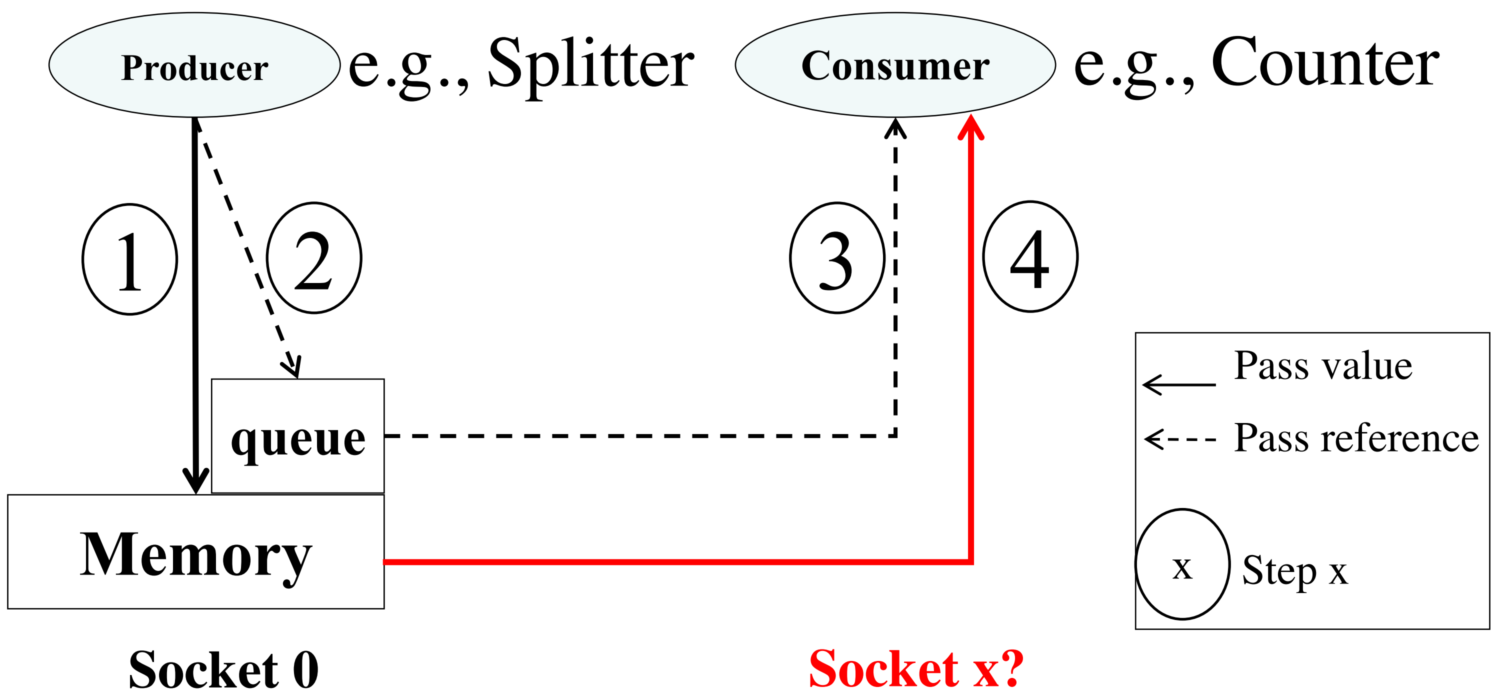
# Zoom into the current system design



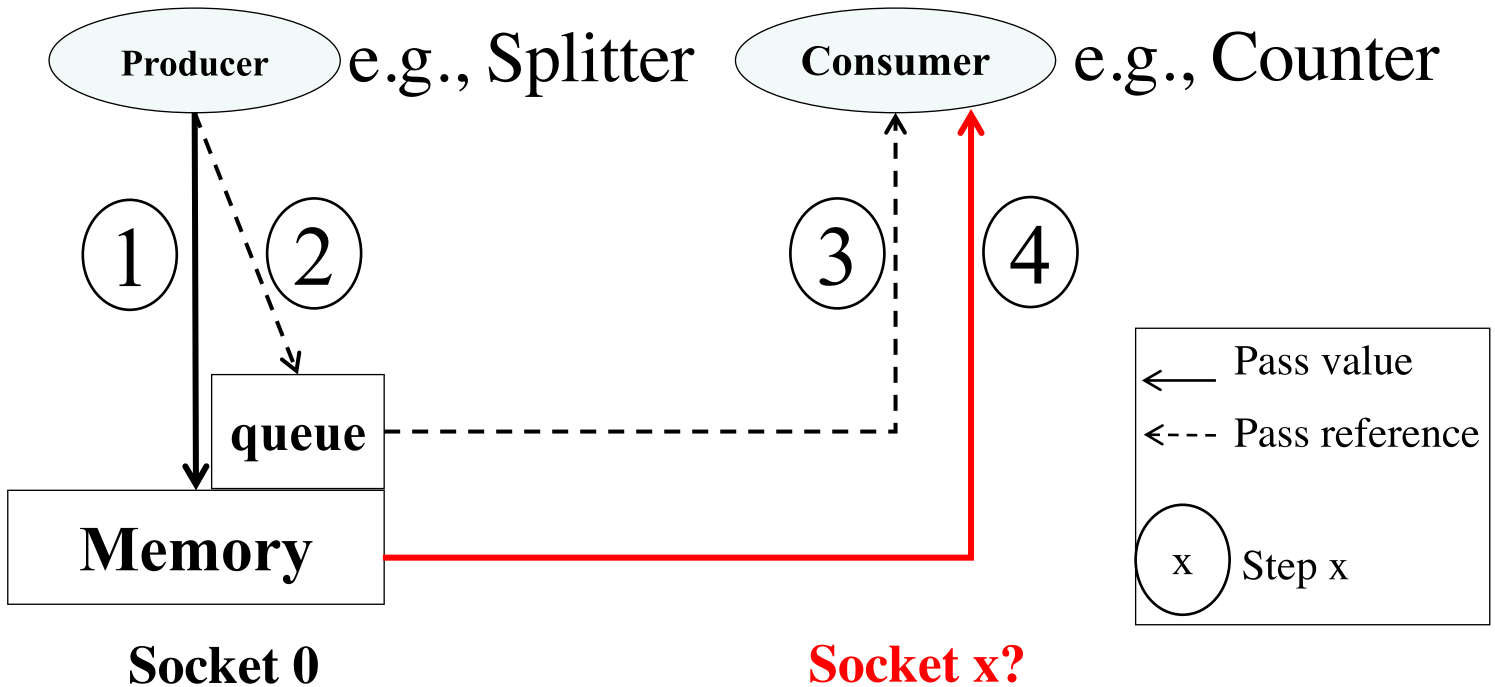
# Zoom into the current system design



# Zoom into the current system design



# Zoom into the current system design




- **Relative location affects the processing behavior of consumer.**

# Key Idea

- **Construct a performance model that is able to estimate each operator's processing behavior under different execution plans.**
  - Based on Rate-based Optimization Framework (RBO model-SIGMOD'02).
  - Extend it to capture the NUMA effect.

# Key Idea

- **Construct a performance model that is able to estimate each operator's processing behavior under different execution plans.**
  - Based on Rate-based Optimization Framework (RBO model-SIGMOD'02).
  - Extend it to capture the NUMA effect.

Introduce remote-memory-access (RMA) overhead factor to the original RBO model.

# Key Idea

# Key Idea

- **Our model estimates (varying) processing behavior of each operator accurately but the problem is not yet solved.**
- **Branch and Bound technique to rescue.**
  - Bounding function: operators remaining scheduled are free to collocate with all of its producers.
  - Heuristics: to further reduce searching space.

# Key Idea

- **Our model estimates (varying) processing behavior of each operator accurately but the problem is not yet solved.**
- **Branch and Bound technique to rescue.**
  - Bounding function: operators remaining scheduled are free to collocate with all of its producers.
  - Heuristics: to further reduce searching space.

# Key Idea

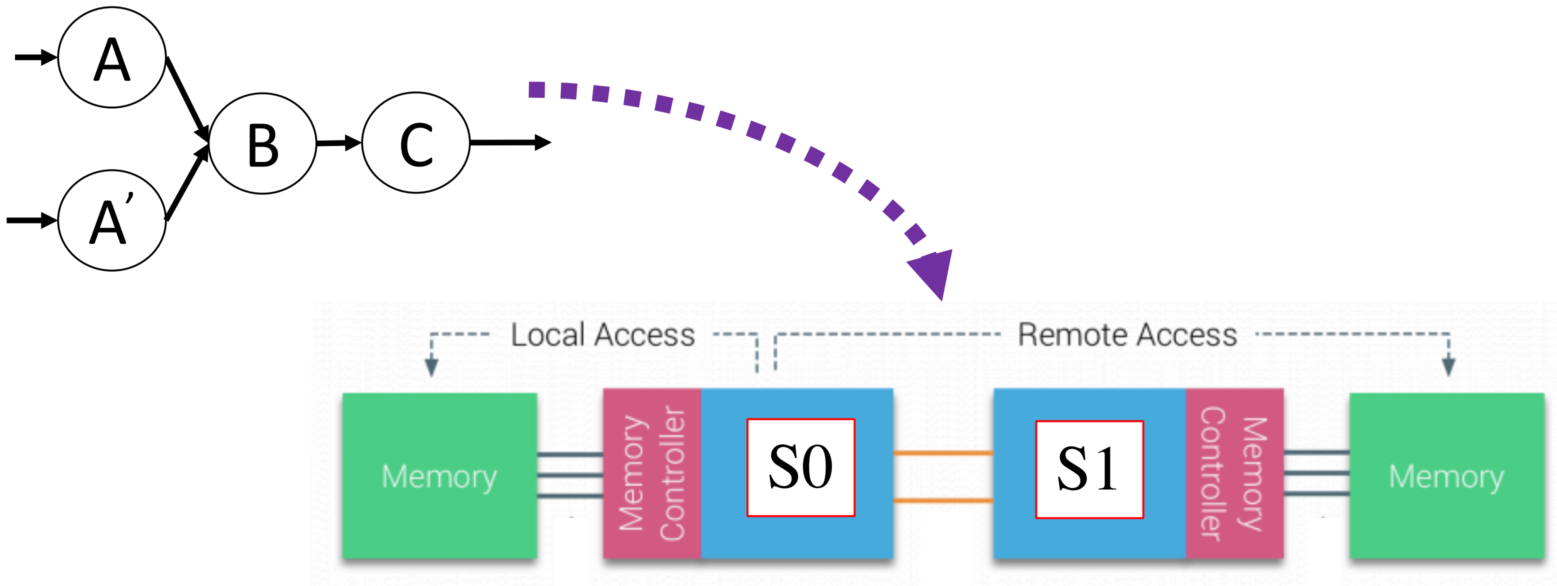
- **Our model estimates (varying) processing behavior of each operator accurately but the problem is not yet solved.**
- **Branch and Bound technique to rescue.**
  - Bounding function: operators remaining scheduled are free to collocate with all of its producers.
  - Heuristics: to further reduce searching space.

# Key Idea

- **Our model estimates (varying) processing behavior of each operator accurately but the problem is not yet solved.**
- **Branch and Bound technique to rescue.**
  - Bounding function: operators remaining scheduled are free to collocate with all of its producers.
  - Heuristics: to further reduce searching space.

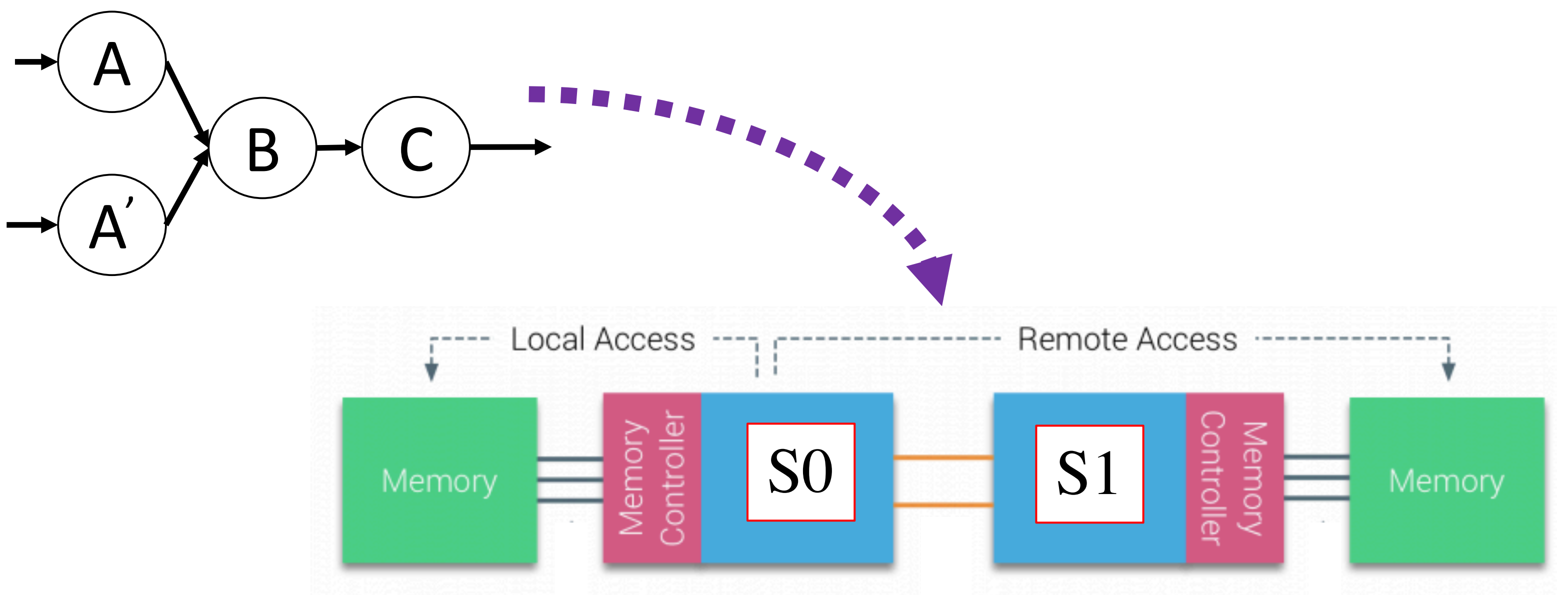
# Placement example

| Solution | |
|---|---|
| A | S0 |
| A' | S0 |
| B | S1 |
| C | S1 |



# Placement example

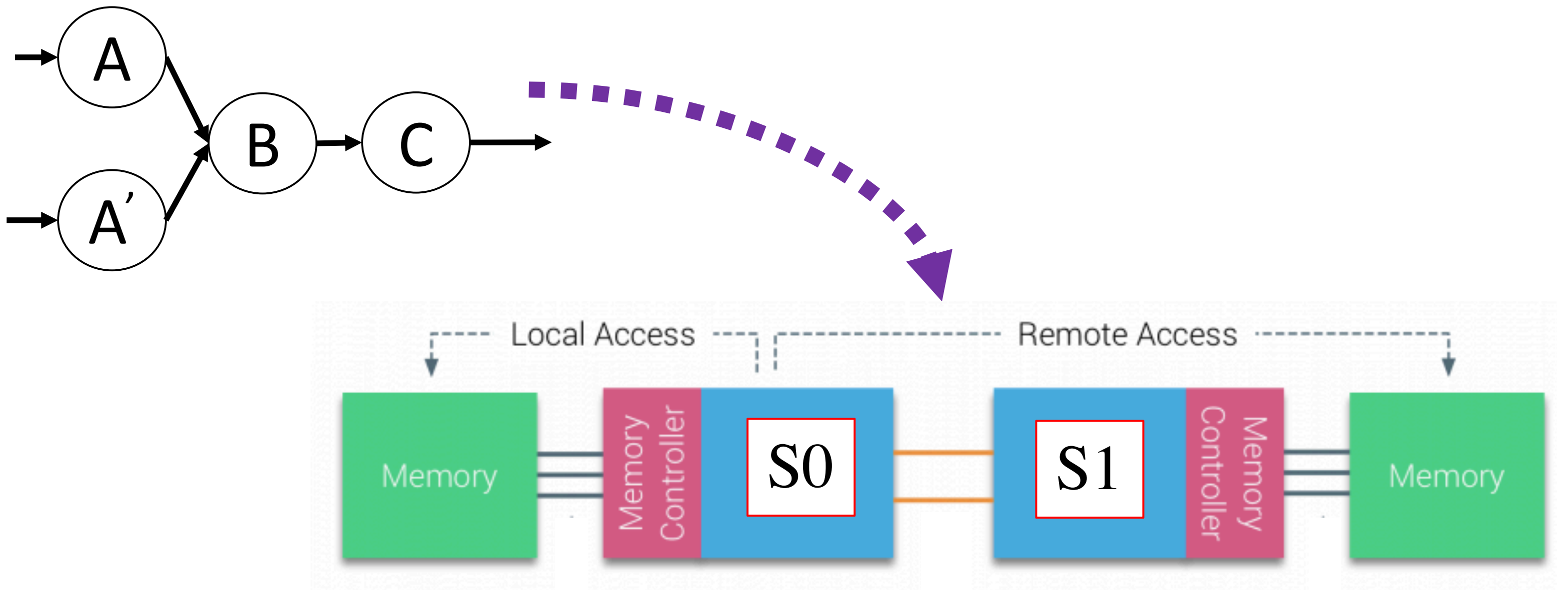
| Solution | |
|---|---|
| A | S0 |
| A’ | S0 |
| B | S1 |
| C | S1 |




- **Allocate four operators into two sockets (2^4).**
- **Three operators cannot be allocated at the same socket due to resource constraint.**

# Placement example

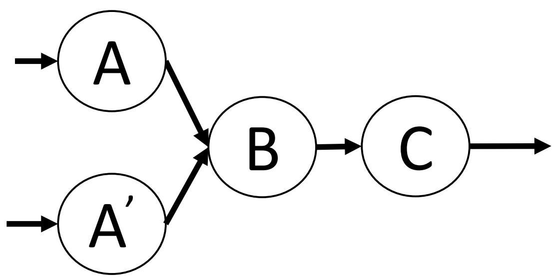
| Solution | |
|---|---|
| A | S0 |
| A’ | S0 |
| B | S1 |
| C | S1 |

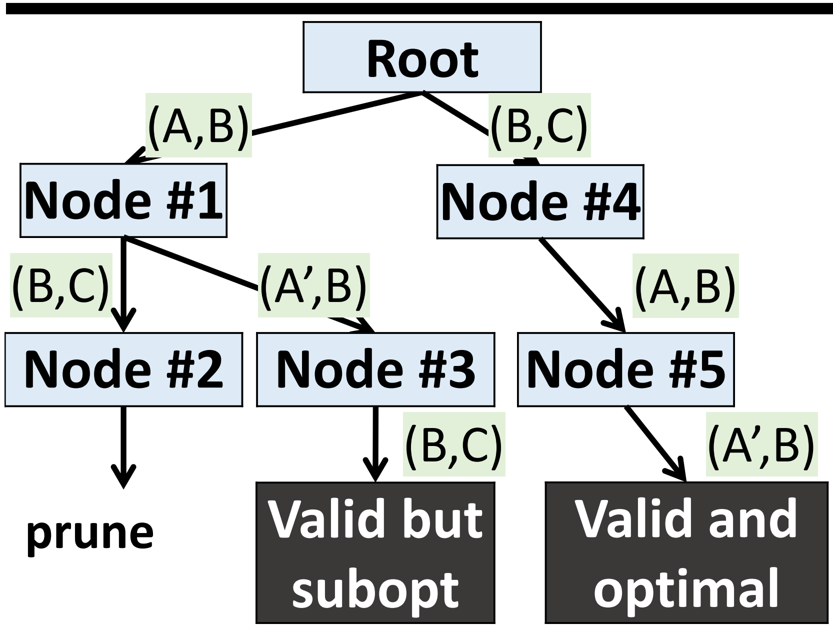



- **Allocate four operators into two sockets (2^4).**
- **Three operators cannot be allocated at the same socket due to resource constraint.**

# Placement example

| Solution | |
|---|---|
| A | S0 |
| A’ | S0 |
| B | S1 |
| C | S1 |




| Root | | | | |
|---|---|---|---|---|
| | allocation | Decisions | | |
| | | (A,B) | (A’,B) | (B,C) |
| A | - | - | - | - |
| A’ | - | - | - | - |
| B | - | ✓ | ✓ | - |
| C | - | - | - | ✓ |

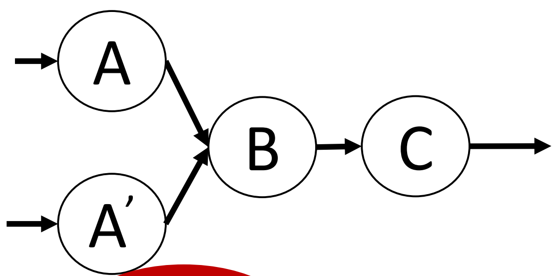
| Node #1 | | | | |
|---|---|---|---|---|
| | allocation | Decisions | | |
| | | (A,B) | (A’,B) | (B,C) |
| A | S0 | - | - | - |
| A’ | - | - | - | - |
| B | S0 | ✓ | ✓ | - |
| C | - | - | - | ✓ |

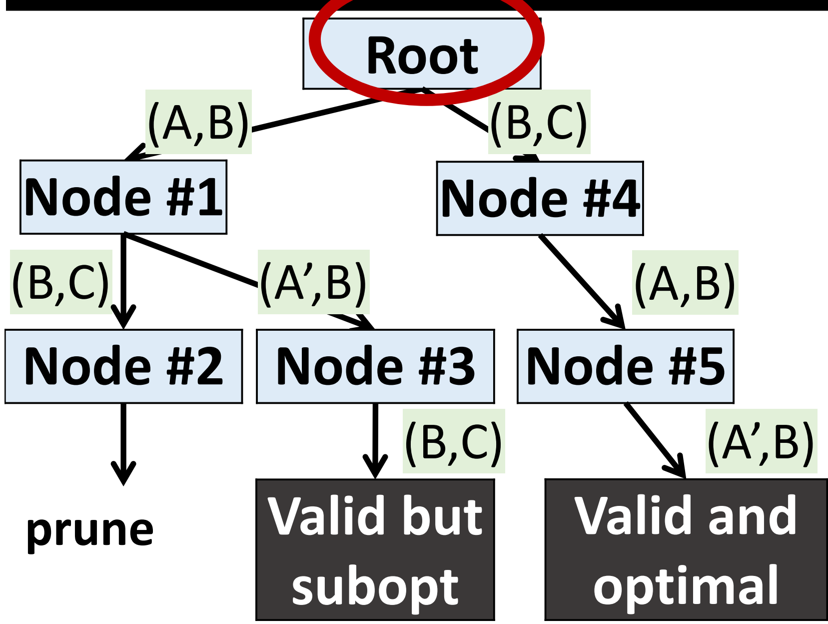
| Node #2 | | | | |
|---|---|---|---|---|
| | allocation | Decisions | | |
| | | (A,B) | (A’,B) | (B,C) |
| A | S0 | - | - | - |
| A’ | - | - | - | - |
| B | S0 | ✓ | ✓ | - |
| C | S1 | - | - | ✗ |

NUS National University of Singapore

# Placement example

| Solution | |
|---|---|
| A | S0 |
| A' | S0 |
| B | S1 |
| C | S1 |




**Root**

| | allocation | Decisions (A,B) | (A',B) | (B,C) |
|---|---|---|---|---|
| A | - | - | - | - |
| A' | - | - | - | - |
| B | - | ✓ | ✓ | - |
| C | - | - | - | ✓ |

**Node #1**

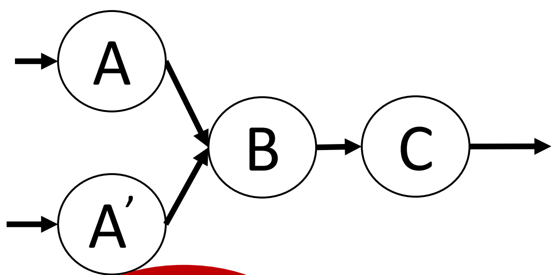
| | allocation | Decisions (A,B) | (A',B) | (B,C) |
|---|---|---|---|---|
| A | S0 | - | - | - |
| A' | - | - | - | - |
| B | S0 | ✓ | ✓ | - |
| C | - | - | - | ✓ |

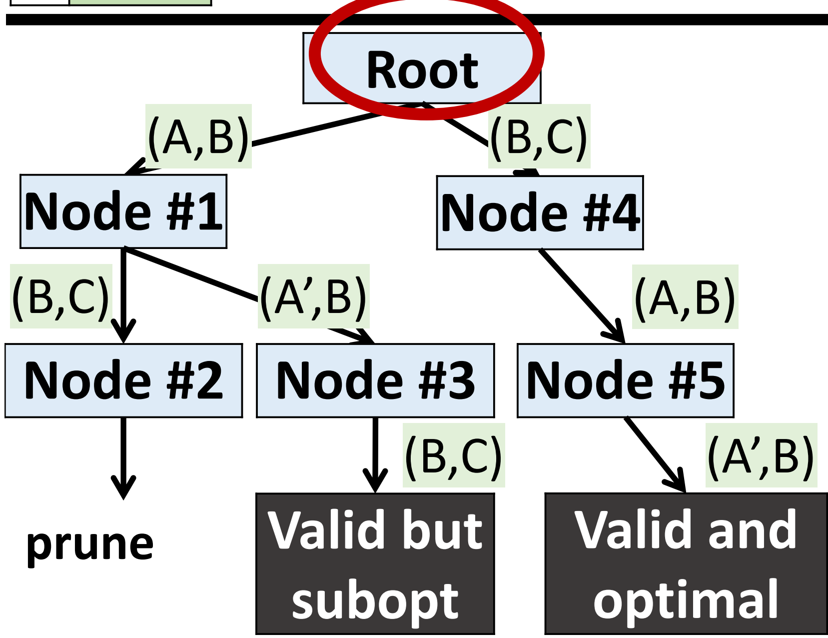
**Node #2**

| | allocation | Decisions (A,B) | (A',B) | (B,C) |
|---|---|---|---|---|
| A | S0 | - | - | - |
| A' | - | - | - | - |
| B | S0 | ✓ | ✓ | - |
| C | S1 | - | - | ✗ |

# Placement example

| Solution | |
|---|---|
| A | S0 |
| A’ | S0 |
| B | S1 |
| C | S1 |



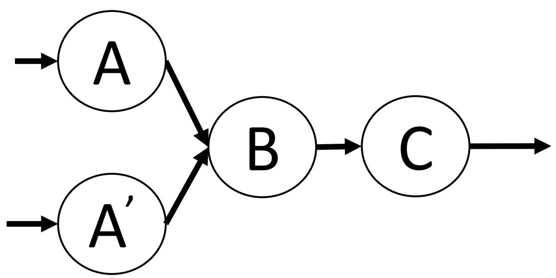

| Root | | | | |
|---|---|---|---|---|
| | allocation | Decisions | | |
| | | (A,B) | (A’,B) | (B,C) |
| A | - | - | - | - |
| A’ | - | - | - | - |
| B | - | ✓ | ✓ | - |
| C | - | - | - | ✓ |

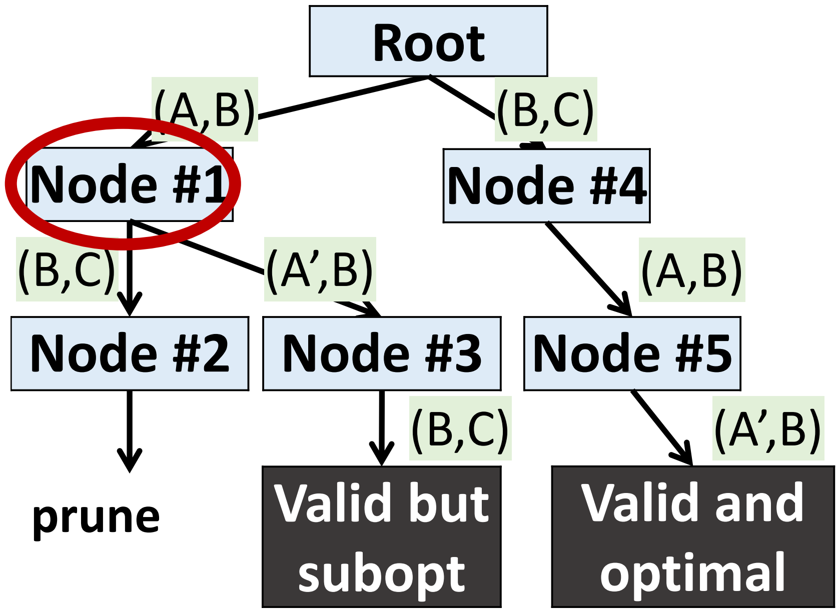
| Node #1 | | | | |
|---|---|---|---|---|
| | allocation | Decisions | | |
| | | (A,B) | (A’,B) | (B,C) |
| A | S0 | - | - | - |
| A’ | - | - | - | - |
| B | S0 | ✓ | ✓ | - |
| C | - | - | - | ✓ |

| Node #2 | | | | |
|---|---|---|---|---|
| | allocation | Decisions | | |
| | | (A,B) | (A’,B) | (B,C) |
| A | S0 | - | - | - |
| A’ | - | - | - | - |
| B | S0 | ✓ | ✓ | - |
| C | S1 | - | - | ✕ |

# Placement example

| Solution | |
|---|---|
| A | S0 |
| A' | S0 |
| B | S1 |
| C | S1 |




**Root**

| | allocation | Decisions (A,B) | (A',B) | (B,C) |
|---|---|---|---|---|
| A | - | - | - | - |
| A' | - | - | - | - |
| B | - | ✓ | ✓ | - |
| C | - | - | - | ✓ |

**Node #1**

| | allocation | Decisions (A,B) | (A',B) | (B,C) |
|---|---|---|---|---|
| A | S0 | - | - | - |
| A' | - | - | - | - |
| B | S0 | ✓ | ✓ | - |
| C | - | - | - | ✓ |

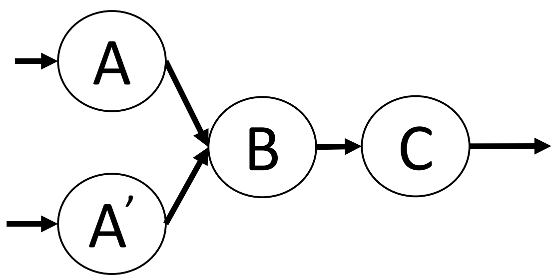
**Node #2**

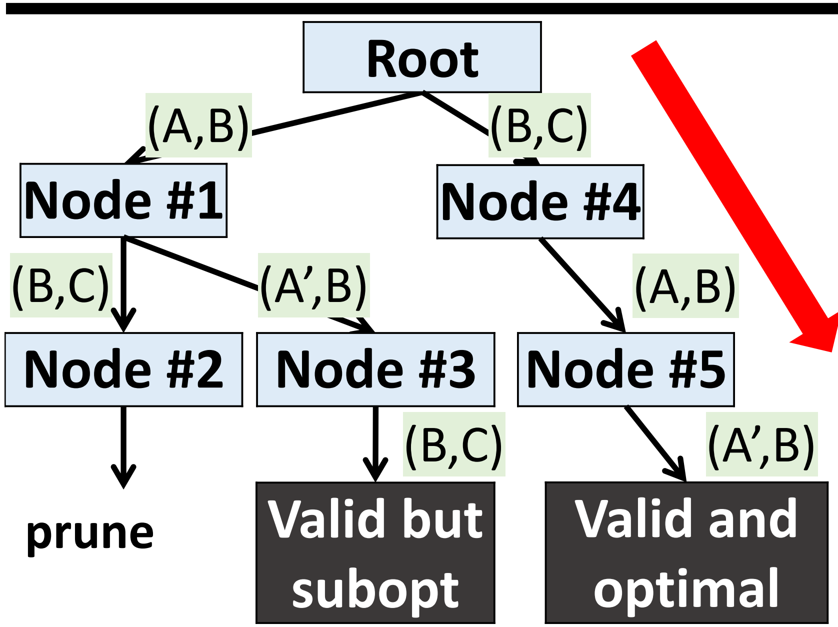
| | allocation | Decisions (A,B) | (A',B) | (B,C) |
|---|---|---|---|---|
| A | S0 | - | - | - |
| A' | - | - | - | - |
| B | S0 | ✓ | ✓ | - |
| C | S1 | - | - | ✗ |

NUS National University of Singapore

# Placement example

| Solution | |
|---|---|
| A | S0 |
| A’ | S0 |
| B | S1 |
| C | S1 |




**Root**

| | allocation | Decisions (A,B) | (A’,B) | (B,C) |
|---|---|---|---|---|
| A | - | - | - | - |
| A’ | - | - | - | - |
| B | - | ✓ | ✓ | - |
| C | - | - | - | ✓ |

**Node #1**

| | allocation | Decisions (A,B) | (A’,B) | (B,C) |
|---|---|---|---|---|
| A | S0 | - | - | - |
| A’ | - | - | - | - |
| B | S0 | ✓ | ✓ | - |
| C | - | - | - | ✓ |

**Node #2**

| | allocation | Decisions (A,B) | (A’,B) | (B,C) |
|---|---|---|---|---|
| A | S0 | - | - | - |
| A’ | - | - | - | - |
| B | S0 | ✓ | ✓ | - |
| C | S1 | - | - | ✗ |

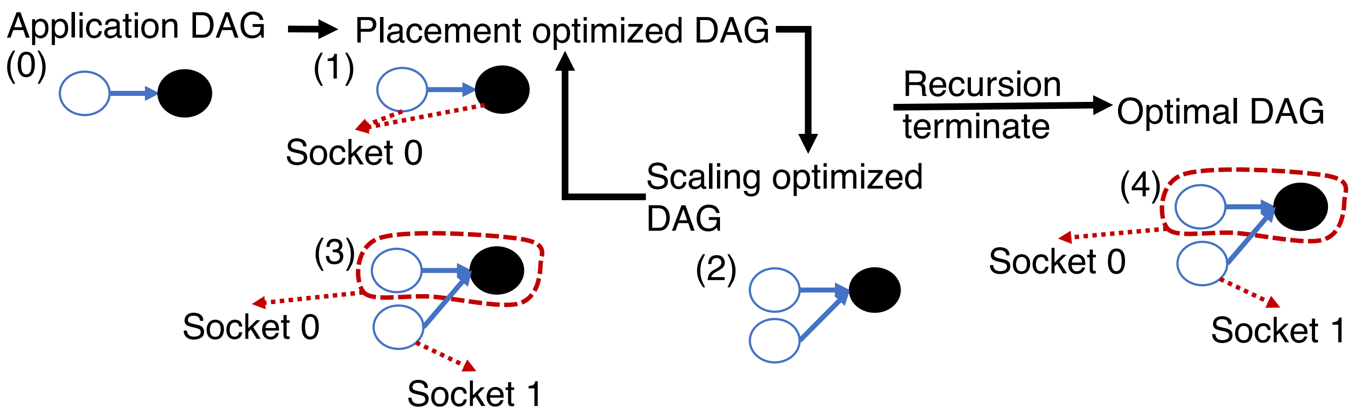
# RLAS Overview

# RLAS Overview

The key idea to optimize streaming application is to remove bottlenecks.

# RLAS Overview

## RLAS optimization
## = Repeat {Scaling Optimization + Placement Optimization}.




The key idea to optimize streaming application is to remove bottlenecks.

# BriskStream Implementation

- **Optimized for shared-memory multicore architecture.**
  - Reduced instruction footprint.
  - Improved communication efficiency.
- **Integrated RLAS optimization framework.**
  - Performance model.
    - **Profiling frameworks.**
  - Optimization Algorithms.

# Experimental Evaluation

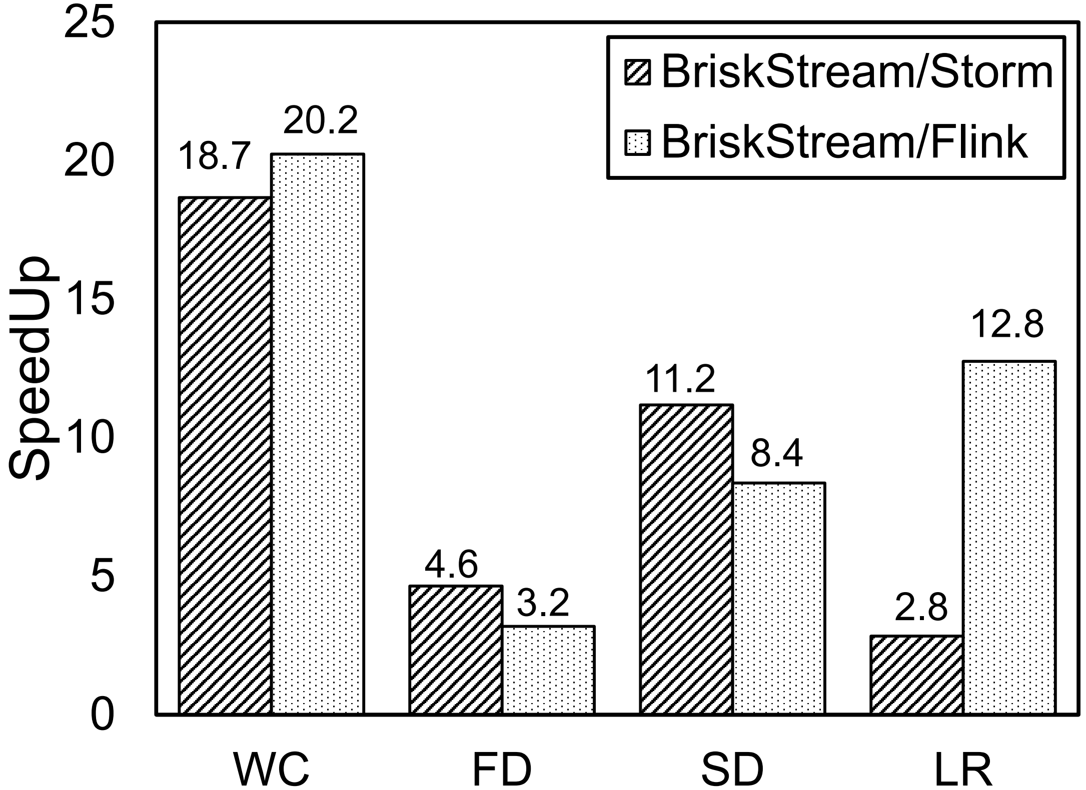
**Applications:**

WC: word-count; FD: fault-detection; SD: spike-detection; LR: linear road benchmark

# Experimental Evaluation

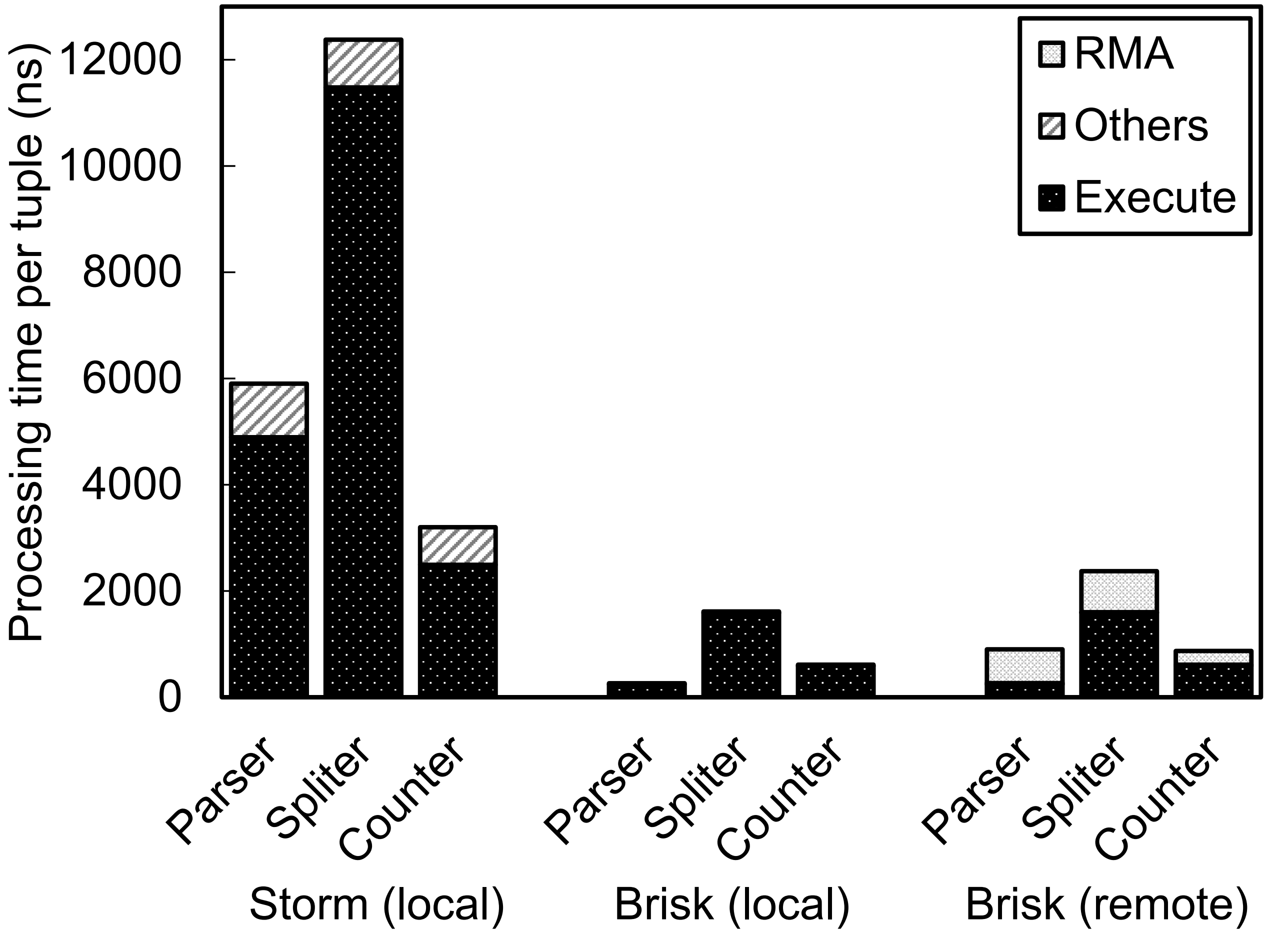
**Applications:**

WC: word-count; FD: fault-detection; SD: spike-detection; LR: linear road benchmark

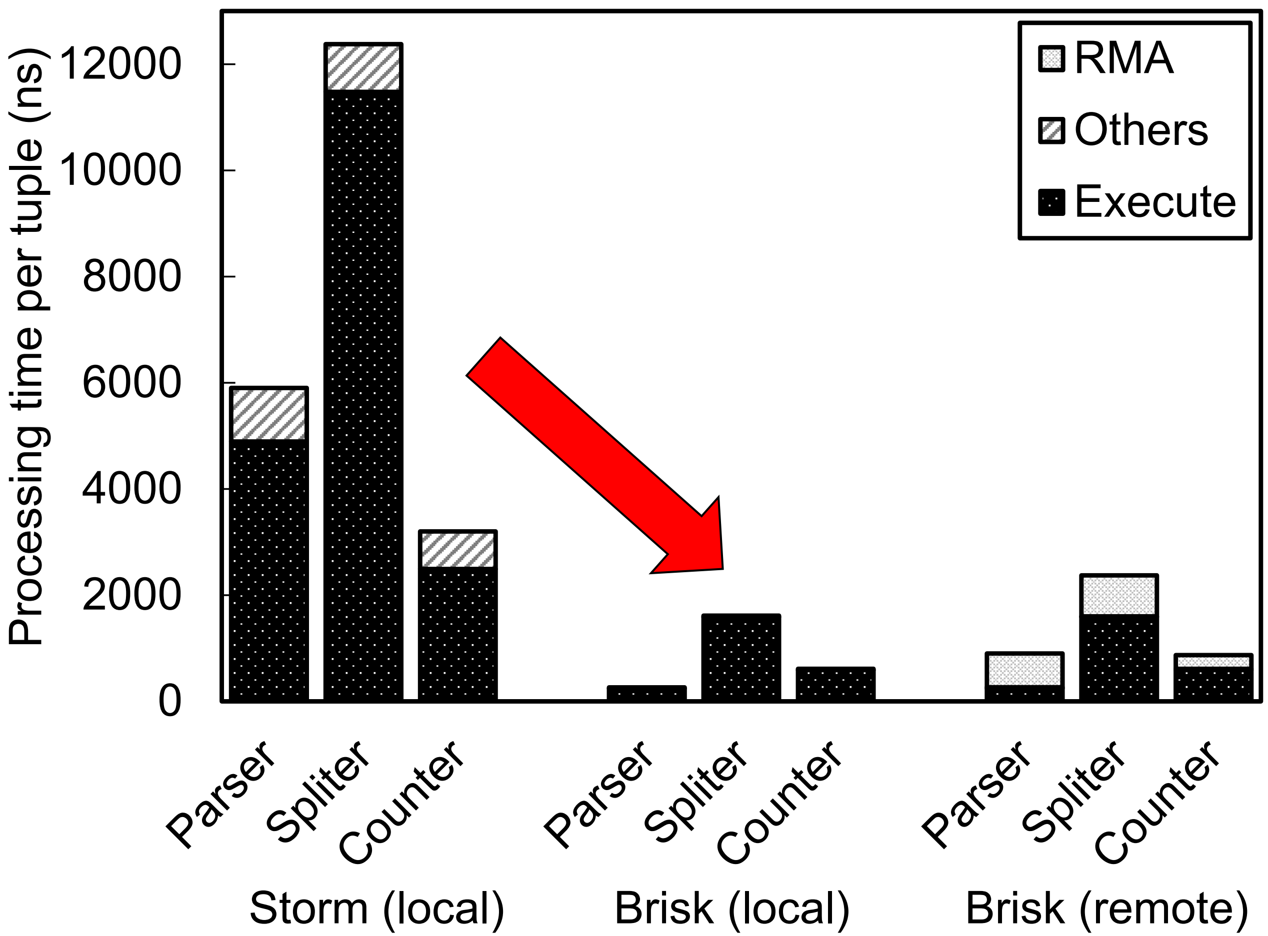



- Much higher throughput

# Execution Time Breakdown

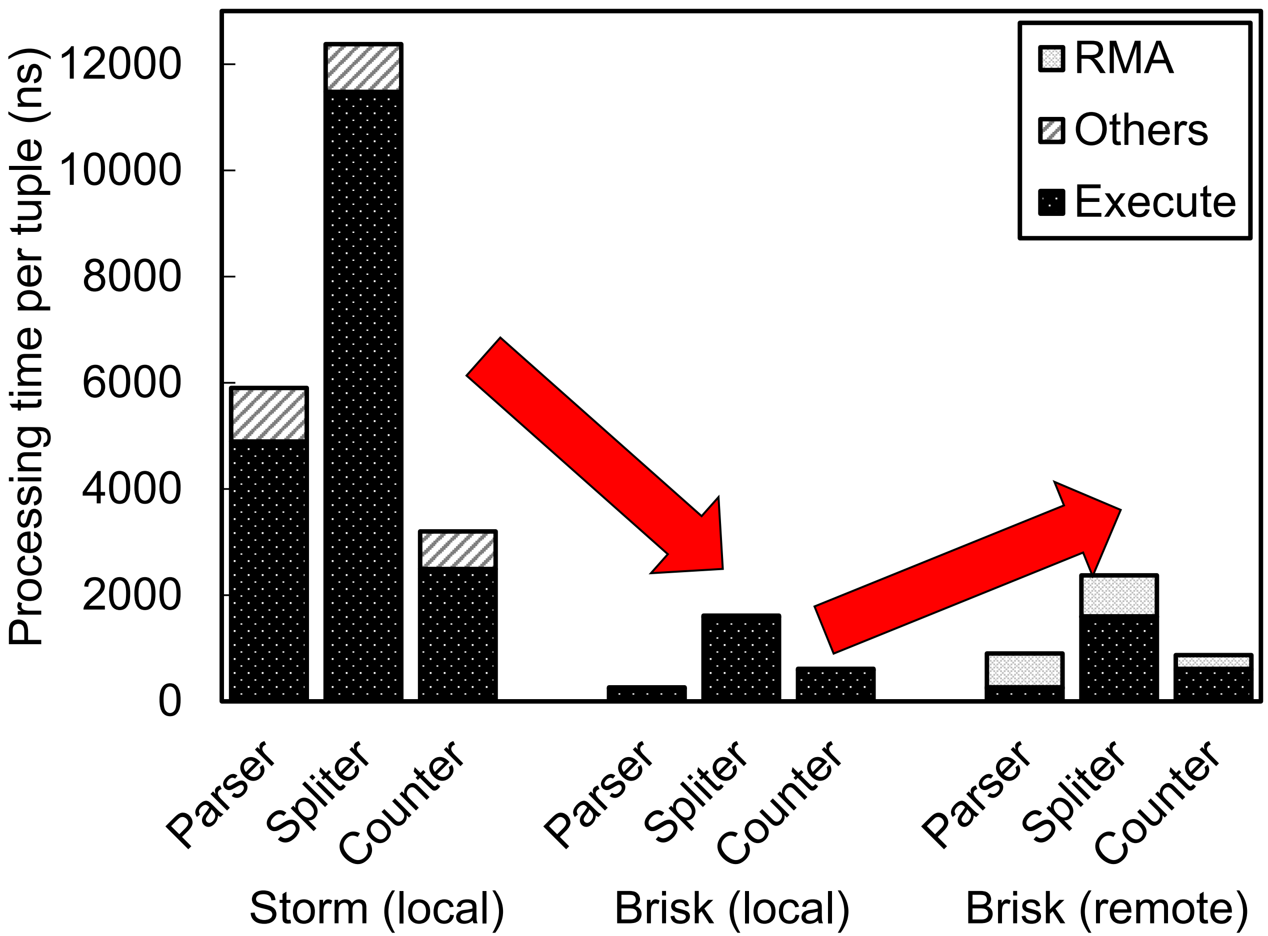


# Execution Time Breakdown




(1) "Execute" is significantly reduced to only 5 ~ 24% of that of Storm.

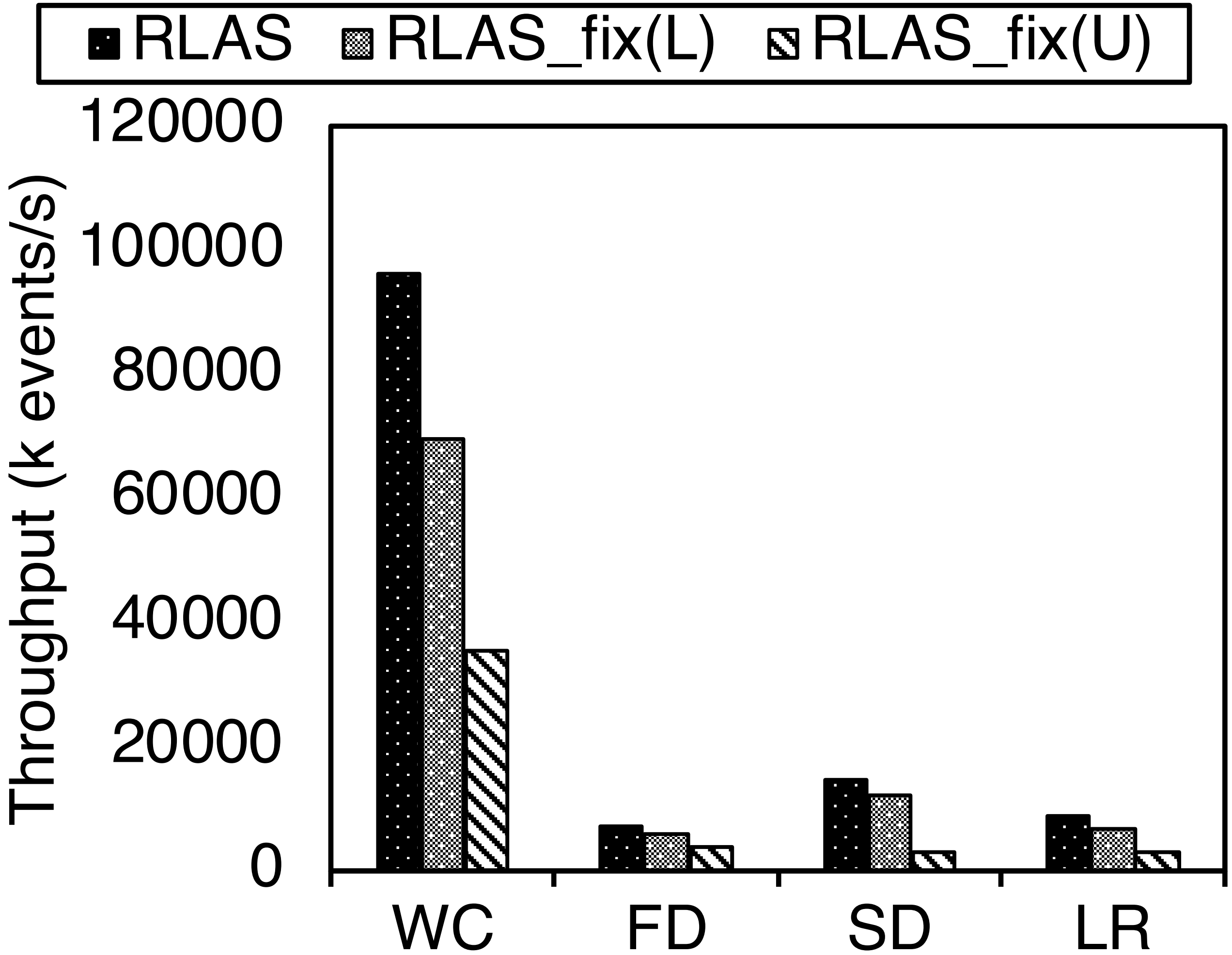
# Execution Time Breakdown



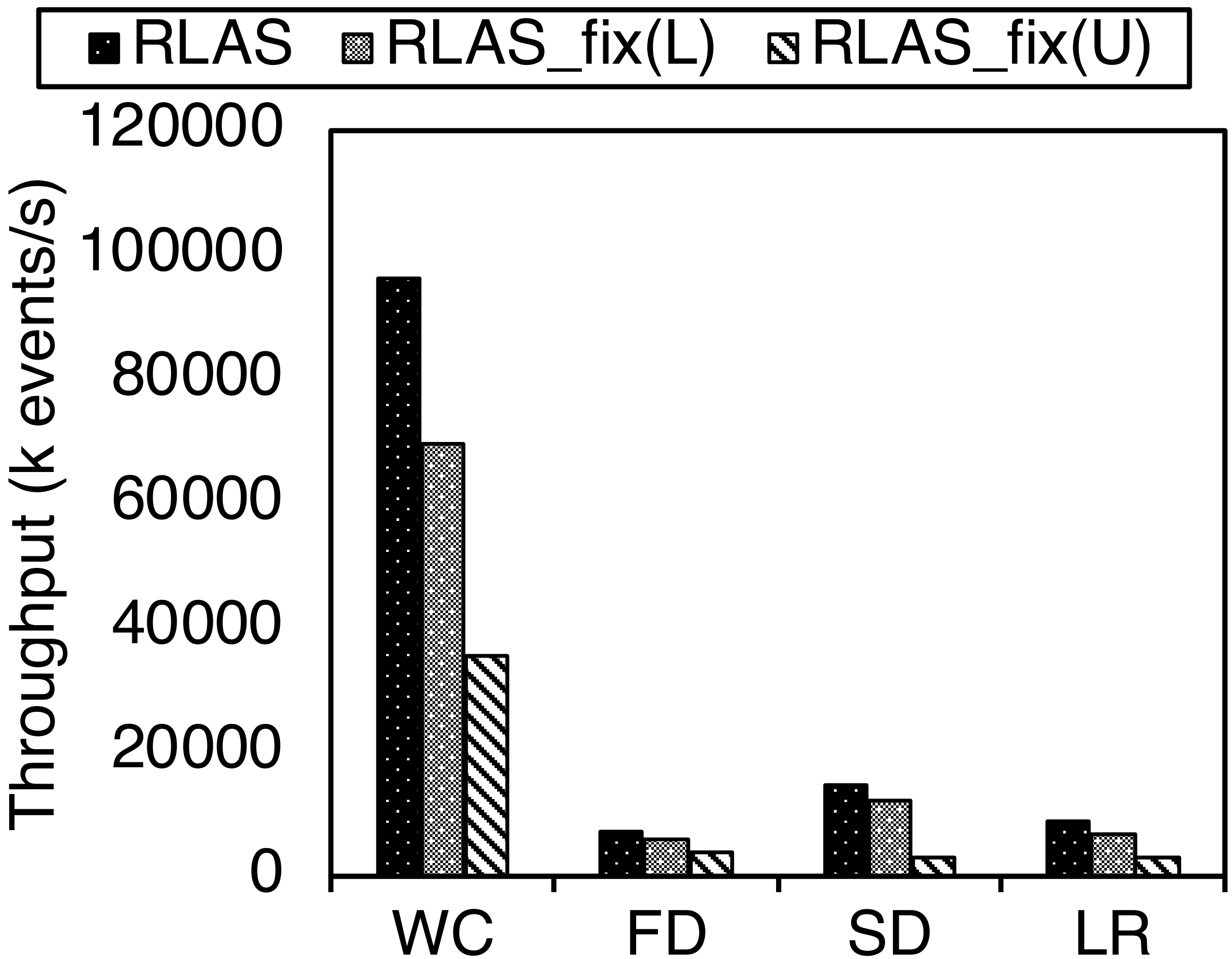

(1) "Execute" is significantly reduced to only 5 ~ 24% of that of Storm.

(2) RMA overhead becomes a critical issue to optimize.

# Importance of relative location awareness

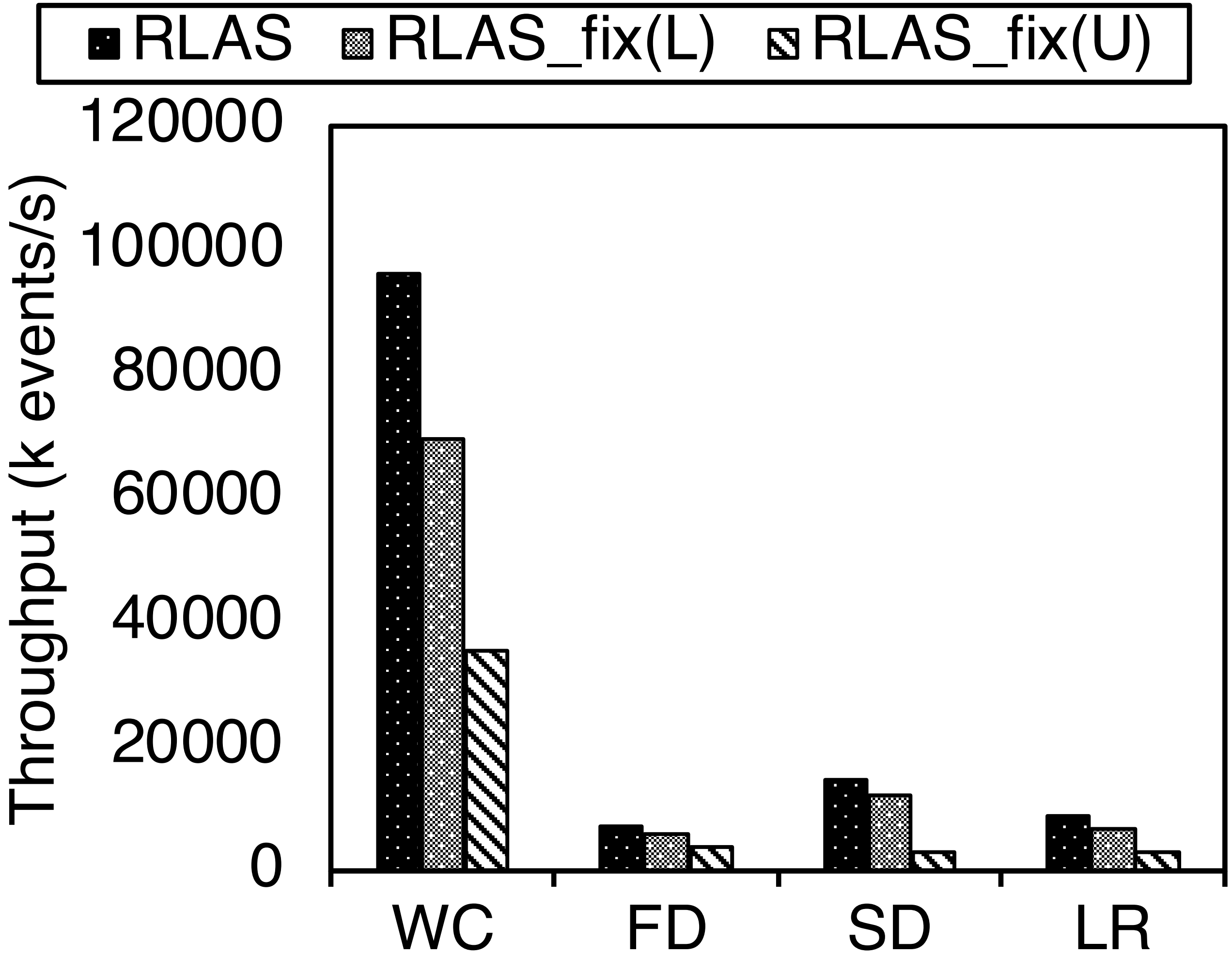


# Importance of relative location awareness



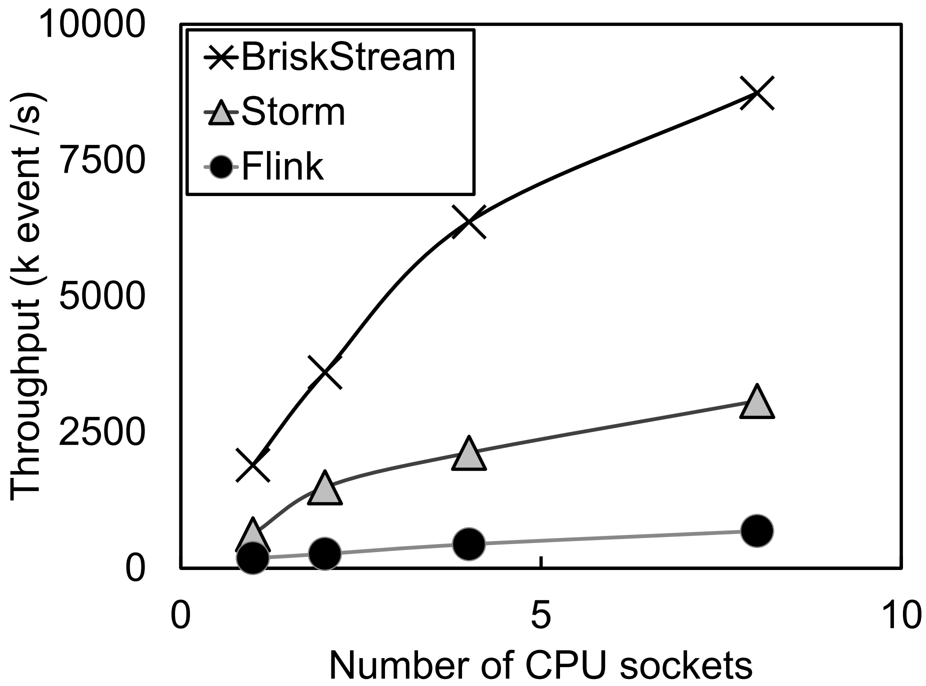

- RLAS stands for our relative-location-aware scheduling scheme.
- RLAS_fix is relative-location oblivious.
  - It assumes a fixed RMA overhead.

# Importance of relative location awareness



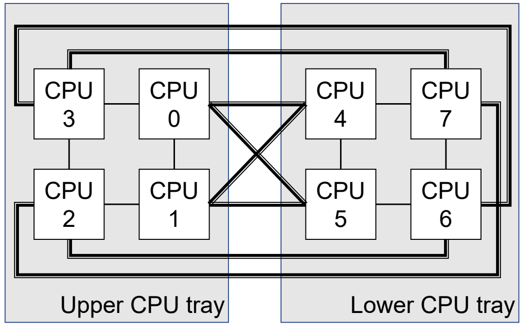

- RLAS stands for our relative-location-aware scheduling scheme.
- RLAS_fix is relative-location oblivious.
  - It assumes a fixed RMA overhead.

# Evaluation of Scalability

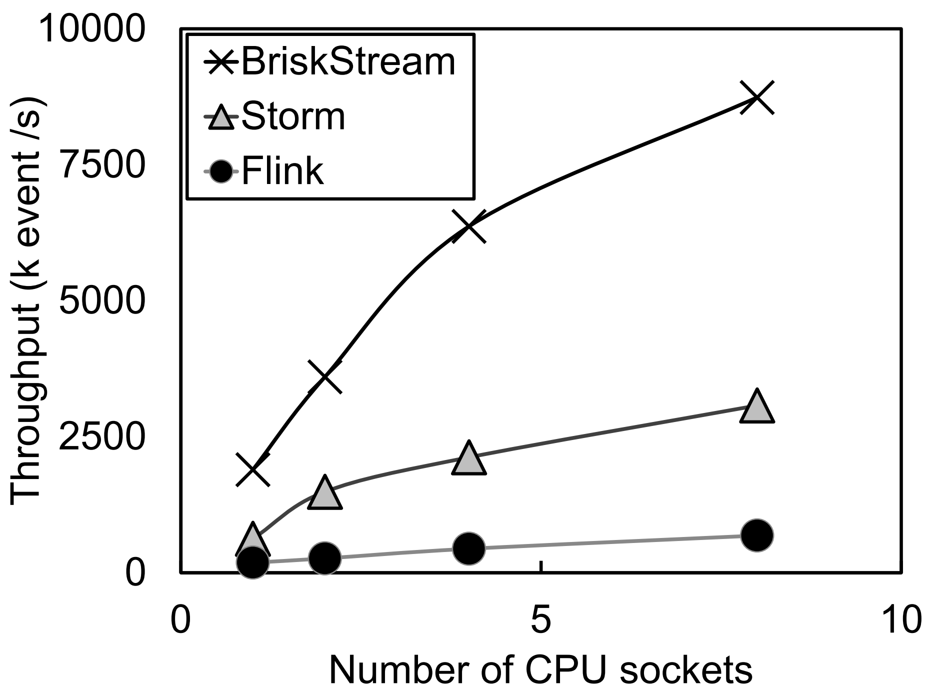



(144 cores)

Linear Road Benchmark



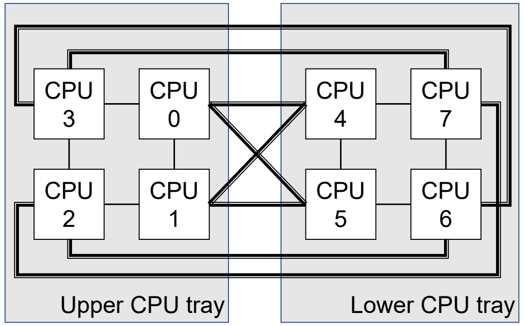
# Evaluation of Scalability



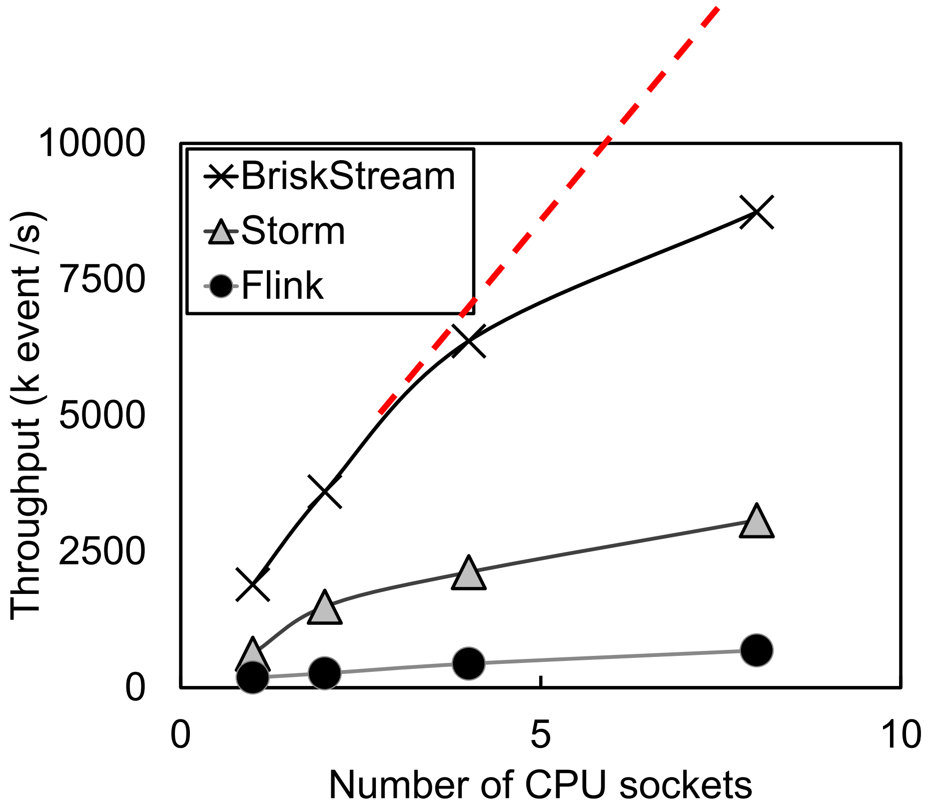

(144 cores)

Linear Road Benchmark

(1) Scales much better



# Evaluation of Scalability

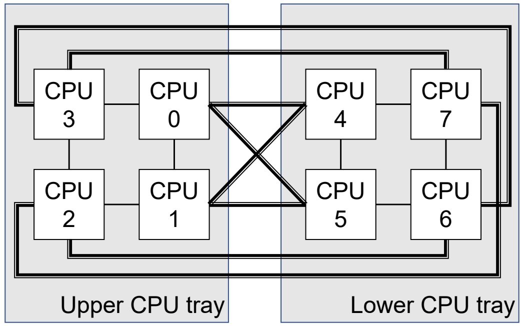



(144 cores)

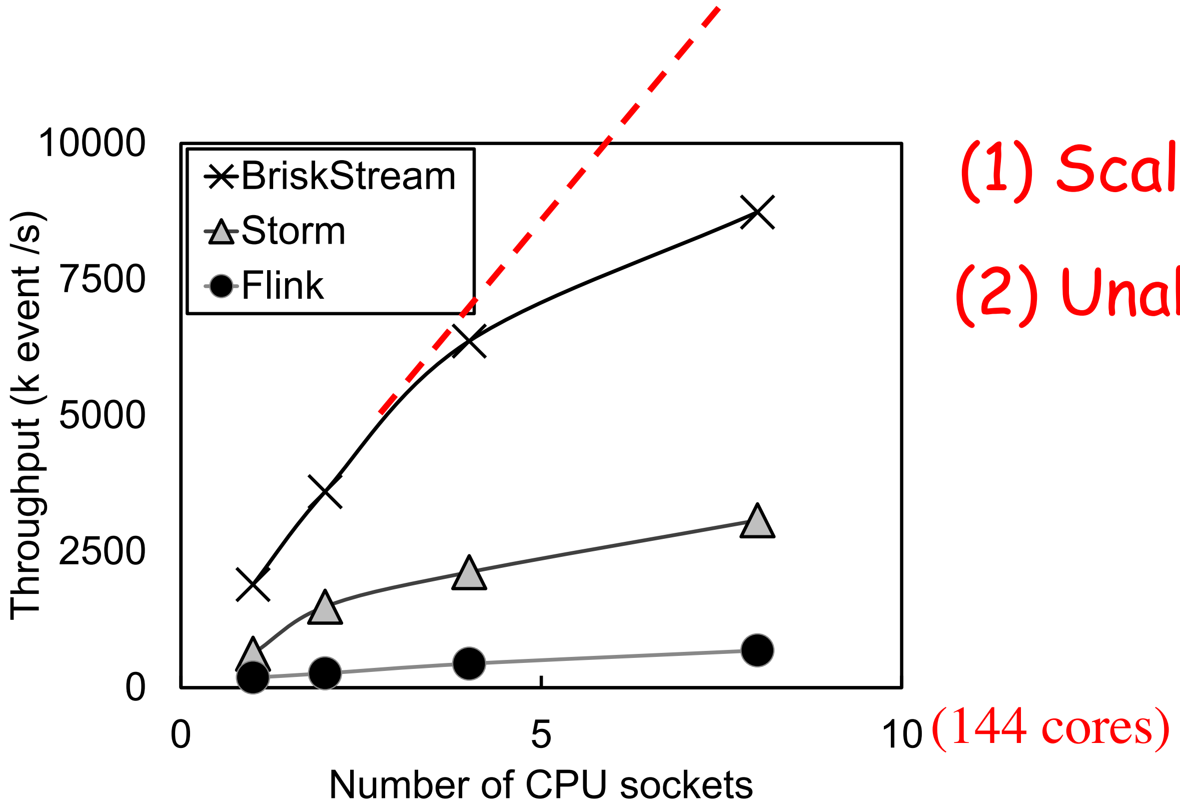
Linear Road Benchmark

(1) Scales much better

(2) Unable to linearly scaleup



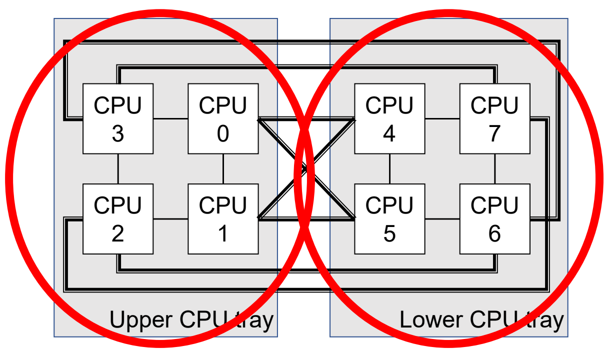
# Evaluation of Scalability




Linear Road Benchmark

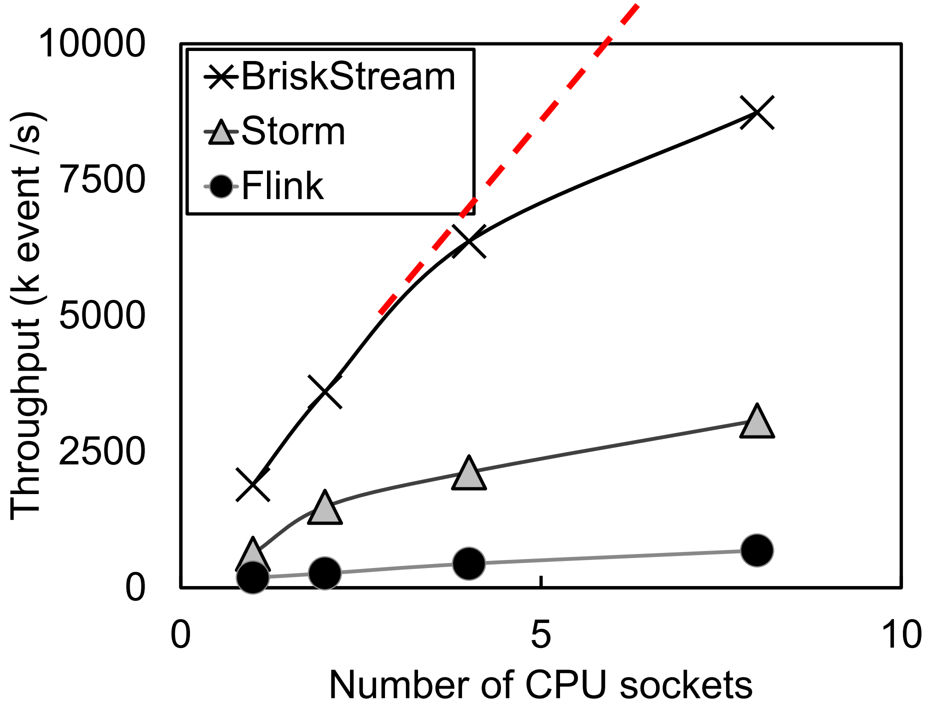
(1) Scales much better

(2) Unable to linearly scaleup




Tray 1 Tray 2

# Evaluation of Scalability



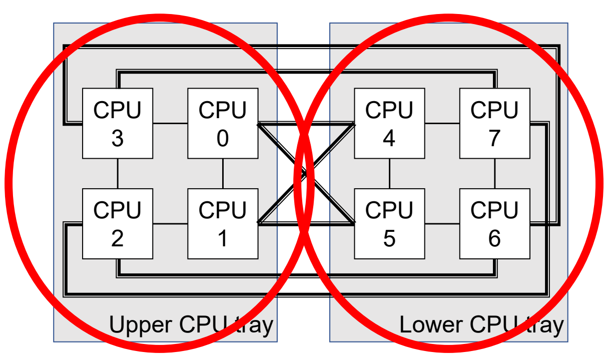

(144 cores)

Linear Road Benchmark

(1) Scales much better

(2) Unable to linearly scaleup

- Stream compression? [TerseCades, ATC'18]




Tray 1 Tray 2

# Recap

# Recap

- BriskStream scales stream computation towards hundred of cores under the NUMA effect.
- Relative-location awareness is the key to address the NUMA effect.

NUS National University of Singapore

# Recap

- BriskStream scales stream computation towards hundred of cores under the NUMA effect.
- Relative-location awareness is the key to address the NUMA effect.

# Future works

# Future works

- **Support Transactional State Management.** **[under-review]**
  - Strict state consistency guarantee.
- **Efficiently support complex query and application.** **[working on]**
  - ML, Trajectory data management, Complex Event Processing.
- **Making BriskStream distribute** **[in-plan].**
  - Elastic and Fault-tolerant.
- **Theoretical improvements** **[in-plan].**
  - To provide better/tighter bounding function of the concerned optimization problem.

# Future works

- **Support Transactional State Management. [under-review]**
  - Strict state consistency guarantee.
- **Efficiently support complex query and application. [working on]**
  - ML, Trajectory data management, Complex Event Processing.
- **Making BriskStream distribute [in-plan].**
  - Elastic and Fault-tolerant.
- **Theoretical improvements [in-plan].**
  - To provide better/tighter bounding function of the concerned optimization problem.

# Future works

- **Support Transactional State Management. [under-review]**
  - Strict state consistency guarantee.
- **Efficiently support complex query and application. [working on]**
  - ML, Trajectory data management, Complex Event Processing.
- **Making BriskStream distribute [in-plan].**
  - Elastic and Fault-tolerant.
- **Theoretical improvements [in-plan].**
  - To provide better/tighter bounding function of the concerned optimization problem.

# Future works

- **Support Transactional State Management. [under-review]**
  - Strict state consistency guarantee.
- **Efficiently support complex query and application. [working on]**
  - ML, Trajectory data management, Complex Event Processing.
- **Making BriskStream distribute [in-plan].**
  - Elastic and Fault-tolerant.
- **Theoretical improvements [in-plan].**
  - To provide better/tighter bounding function of the concerned optimization problem.

# Acknowledgement


- This work is supported by a MoE AcRF Tier 2 grant (MOE2017-T2-1-122) and an NUS startup grant in Singapore.
- Jiong He is supported by the National Research Foundation, Prime Ministers Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme.
- Chi Zhou’s work is partially supported by the National Natural Science Foundation of China under Grant 61802260 and the Guangdong Natural Science Foundation under Grant 2018A030310440.

# Q&A

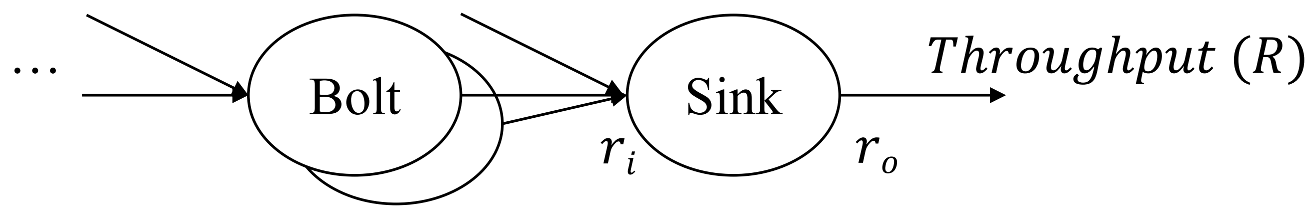
Thank you

shuhao.zhang@comp.nus.edu.sg

❖END

https://github.com/ShuhaoZhangTony/briskstream

# The Performance Model




- $r_i$ : input rate – depends on ($r_o$ of upstream operators).
- $r_o$: output rate – depends on (processing speed) and ($r_i$ of upstream operators).
- The model tries to estimate throughput ($R$), which is Sink's $r_o$.

# Estimating $r_o$ of an Operator (1)

- Output rate ($r_o$) can be estimate as #tuples processed ($N$) / time needed to process them ($t_p$).
- Consider an arbitrary observation time t,
  $N$= total aggregated input tuples arrived during t;
  $t_p = \sum_{n=1}^{N} T(p)$.
  under the assumption that $t_p \geq t$ (sufficient input).
- $T(p)$ stands for average time spend on handling each tuple.

# Estimating $r_o$ of an Operator (1)

- Output rate ($r_o$) can be estimate as #tuples processed ($N$) / time needed to process them ($t_p$).
- Consider an arbitrary observation time t,
  $N$= total aggregated input tuples arrived during t;
  $t_p = \sum_{n=1}^{\mathrm{N}} T(p)$.
  under the assumption that $t_p \geq t$ (sufficient input).
- $T(p)$ stands for average time spend on handling each tuple.

Prior works assume $T(p)$ is predefined and independent to different execution plans (p).

# Estimating $T(p)$ of an Operator

$$T(p) = T^e + T^f$$

$$T^f = \#\text{access} * \text{RMA cost per access}$$

# Estimating $T(p)$ of an Operator

$$T(p) = T^e + T^f$$

- $T^e$ :Actual function execution and emitting output tuples assuming the operator has the input data.

$$T^f = \#\text{access} * \text{RMA cost per access}$$

# Estimating $T(p)$ of an Operator

$T(p) = T^e + T^f$

- $T^e$ :Actual function execution and emitting output tuples assuming the operator has the input data.
- $T^f$ :Time required to fetch (local or remotely) the input data from its producers. Estimated as follows.

$$T^f = \#\text{access} * \text{RMA cost per access}$$

# Estimating $T(p)$ of an Operator

$$T(p) = T^e + T^f$$

- $T^e$ :Actual function execution and emitting output tuples assuming the operator has the input data.
- $T^f$ :Time required to fetch (local or remotely) the input data from its producers. Estimated as follows.

$$T^f = \#\text{access} * \text{RMA cost per access}$$

This is why $T(p)$ varies under different execution plans.

# Estimating $\overline{r_o}$ of an Operator

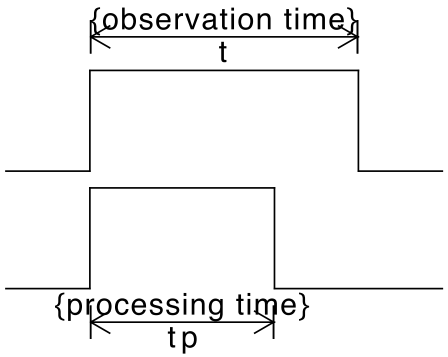
- Implict assumption: $tp \geq \mathrm{t}$
- What happen when the assumption does not hold?
  - What is the $\overline{r_o}$ in general?

# Bottleneck Operators

An operator can have only two possible conditions under a given execution plan.

# Bottleneck Operators

An operator can have only two possible conditions under a given execution plan.

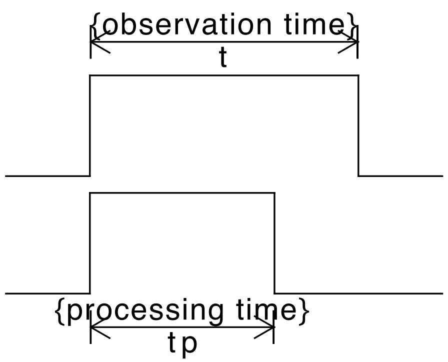



Under-supplied

Output rate is only
determined by its input rate

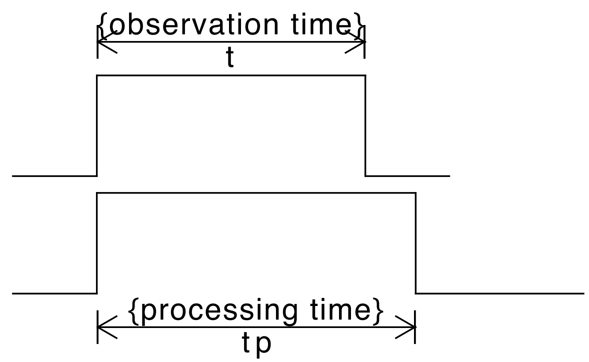
# Bottleneck Operators

An operator can have only two possible conditions under a given execution plan.




Under-supplied

Output rate is only determined by its input rate

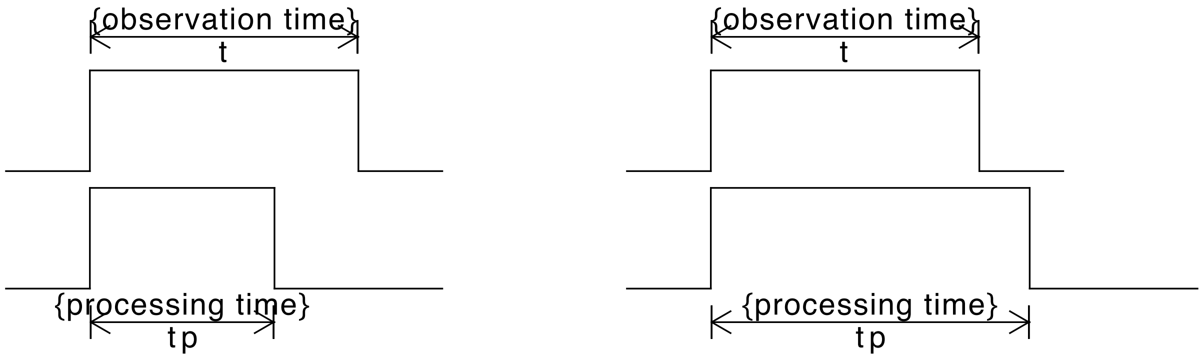



Over-supplied (or just fulfilled)

Output rate is determined by its input rate and processing speed

# Bottleneck Operators

An operator can have only two possible conditions under a given execution plan.




Under-supplied

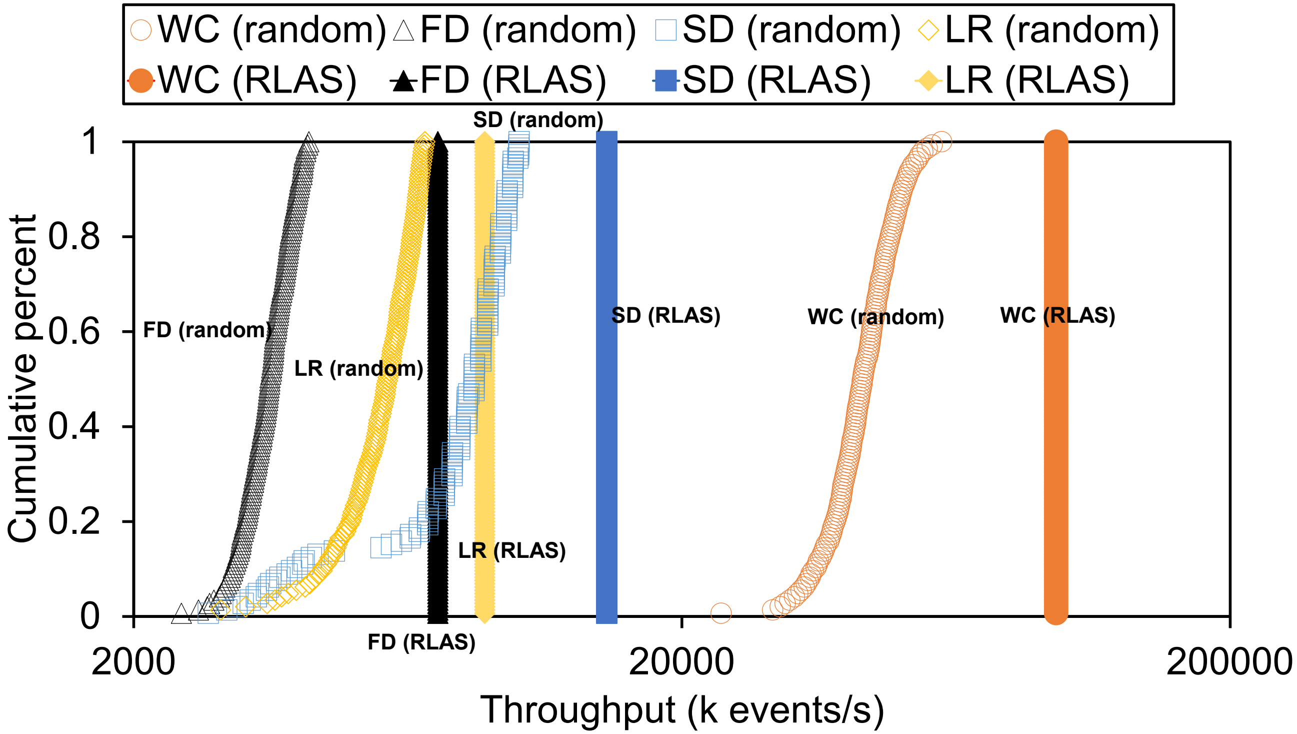
Over-supplied (or just fulfilled)

Operator being over-supplied is essentially the bottleneck.

Output rate is only determined by its input rate

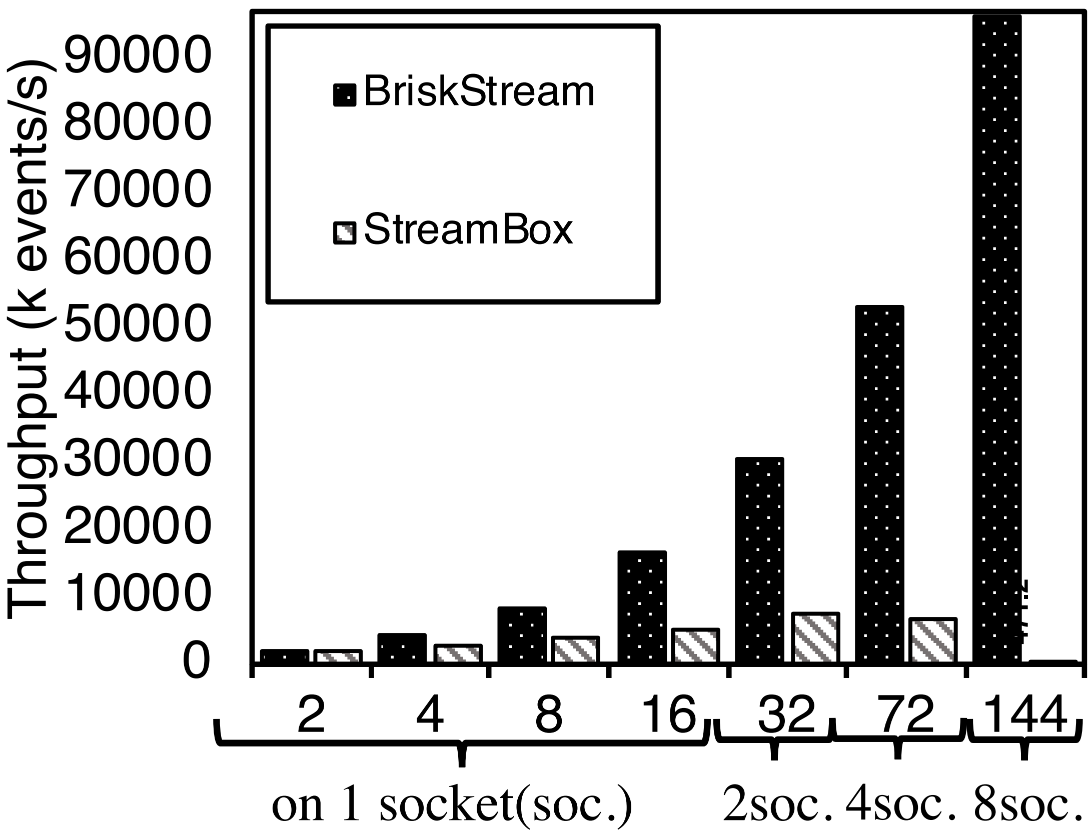
Output rate is determined by its input rate and processing speed

# Why do we need a model?

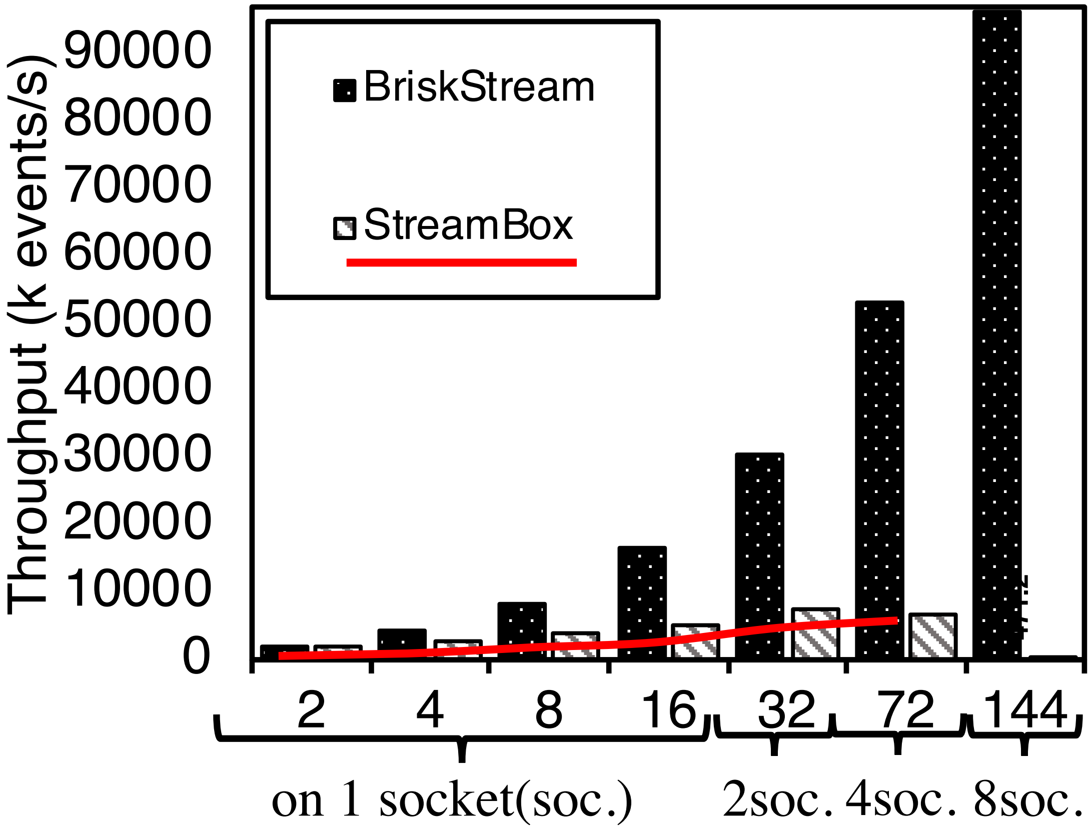



Random plans hurt the performance in a high probability due to the huge optimization space.

# How about other Single-node Optimized DSPS?



# How about other Single-node Optimized DSPS?

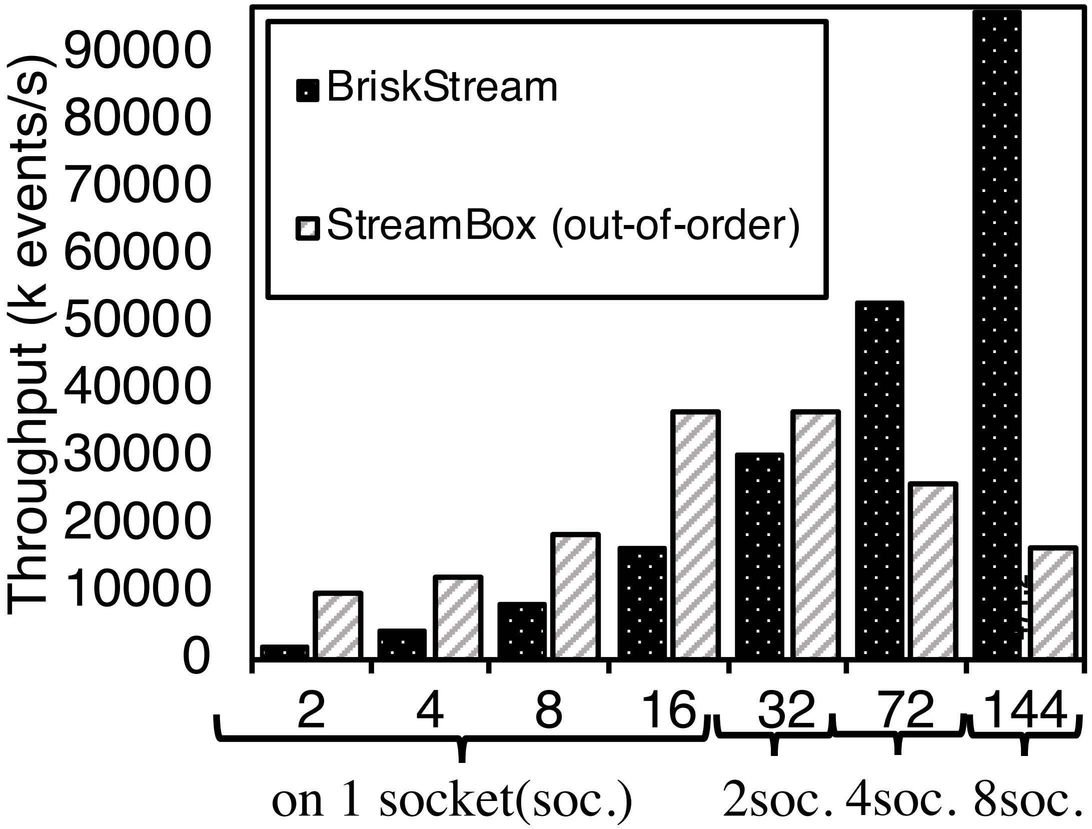



It shows a much poorer performance than BriskStream for its different design focus:

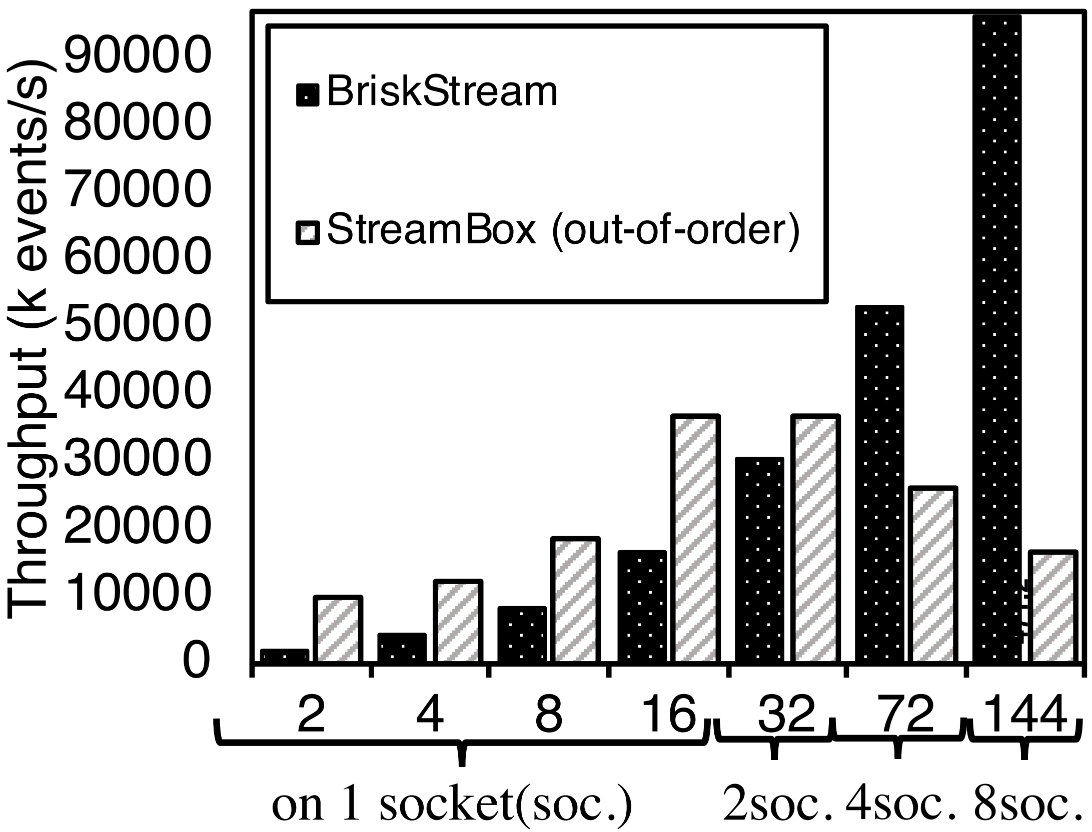
- providing ordering guarantee (bears heavy locking overhead).

NUS National University of Singapore

# How about other Single-node Optimized DSPS?

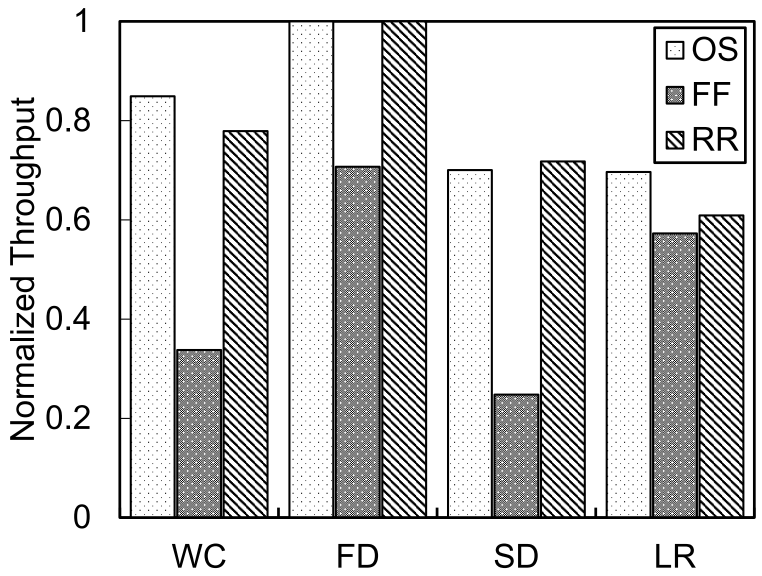


# How about other Single-node Optimized DSPS?



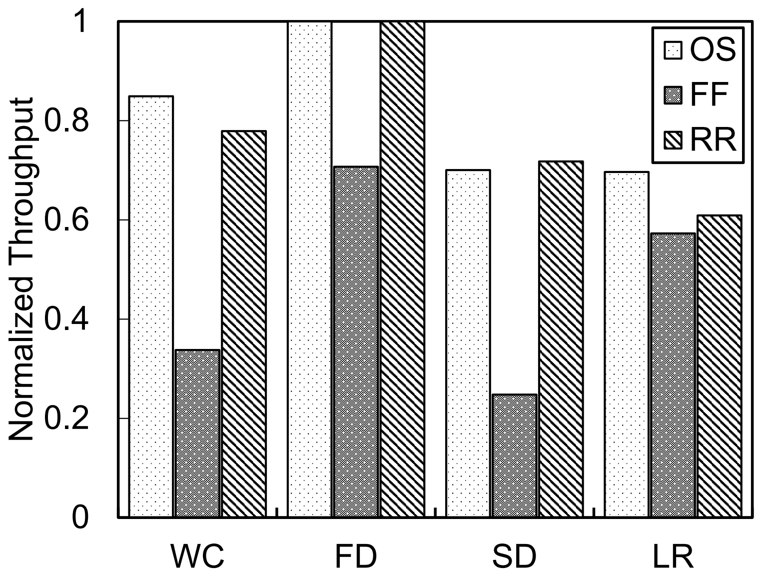

- StreamBox (w/o ordering guarantee) still leads to sub-optimality under large core count due to its greedy nature for addressing NUMA effect.
- Nevertheless, we foresee an interesting future work to employ morsel-driven execution into BriskStream.

# Why not simple heuristics? (1)




- OS: No explicit placement optimization
- FF: Greedily minimize cross-operator traffic.
- RR: Greedily balance workloads among CPU sockets.

# Why not simple heuristics? (1)




- OS: No explicit placement optimization
- FF: Greedily minimize cross-operator traffic.
- RR: Greedily balance workloads among CPU sockets.

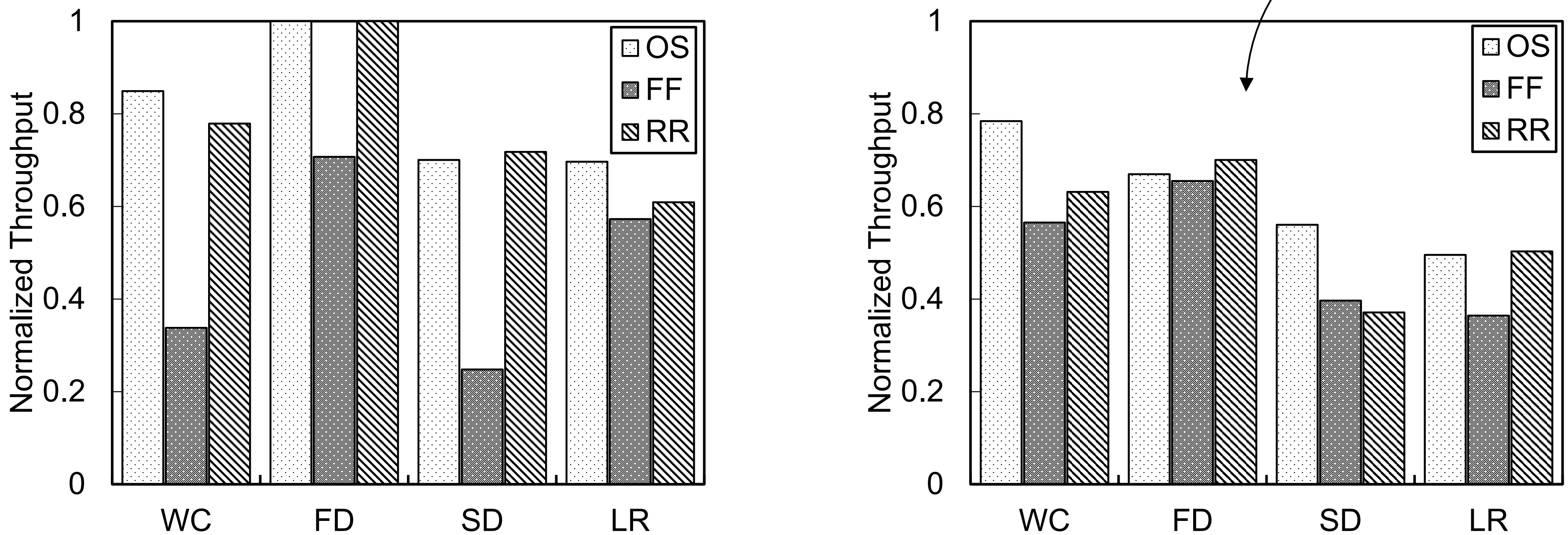
Heuristic approaches commonly used in distributed environment

- either oversubscribe a few CPU sockets (FF)
- or unnecessarily involve RMA overhead (RR).

NUS National University of Singapore

# Why not simple heuristics? (2)

Generally worse

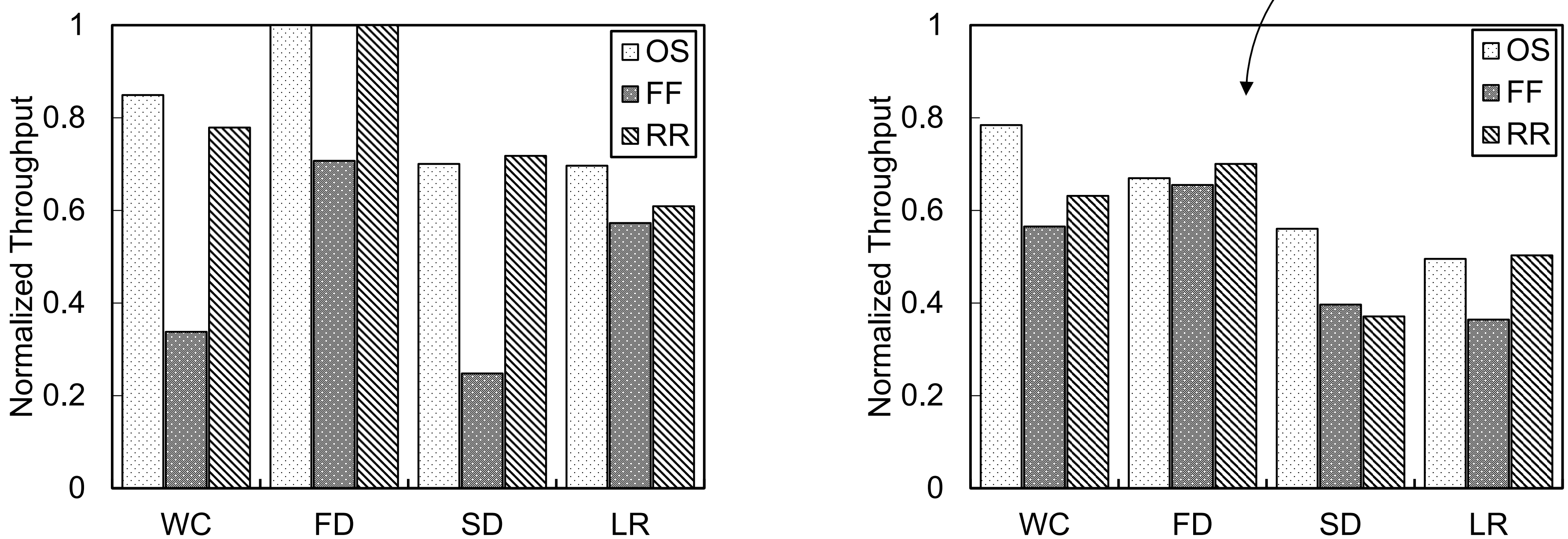



Server A

Server B

# Why not simple heuristics? (2)

Generally worse




Server A

Server B

Heuristic approach requires tedious tuning process to determine the suitable usage of HW resource.