Lecture #12 – Join Algorithms (Sorting)

Andy Pavlo // Carnegie Mellon University // Spring 2016

TODAY'S AGENDA

Background
SIMD
Parallel Sort-Merge Join
Evaluation
Hate Mail

SORT-MERGE JOIN (R⋈S)

Phase #1: Sort

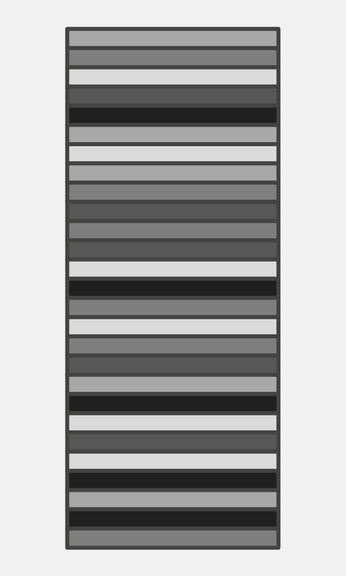
→ Sort the tuples of **R** and **S** based on the join key.

Phase #2: Merge

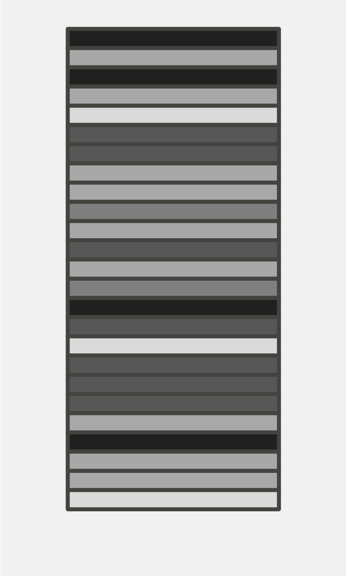
→ Scan the sorted relations and compare tuples.
→ The outer relation **R** only needs to be scanned once.

SORT-MERGE JOIN (R⋈S)

Relation R



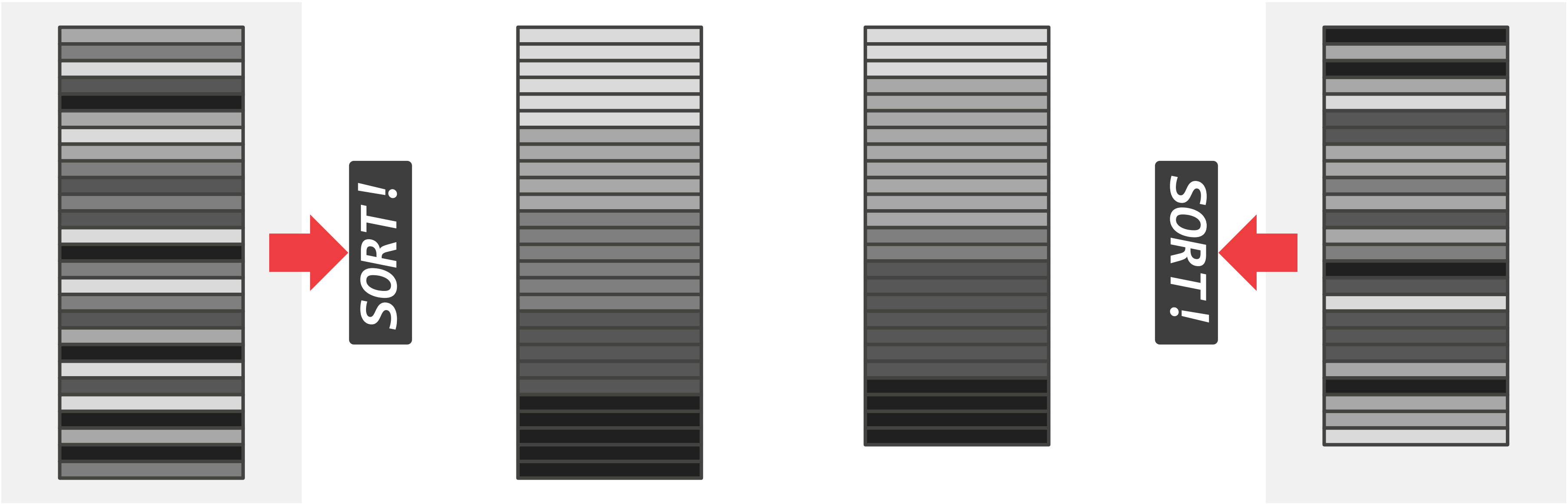
Relation S



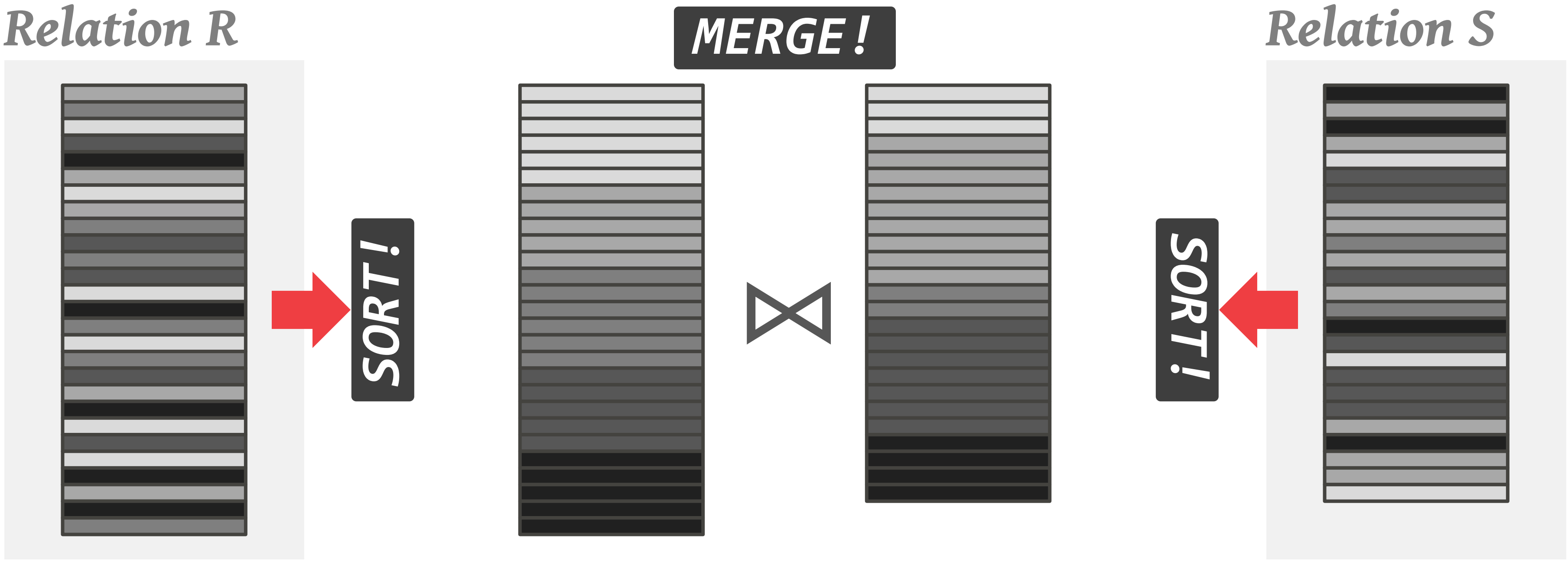
SORT-MERGE JOIN (R⋈S)

Relation R

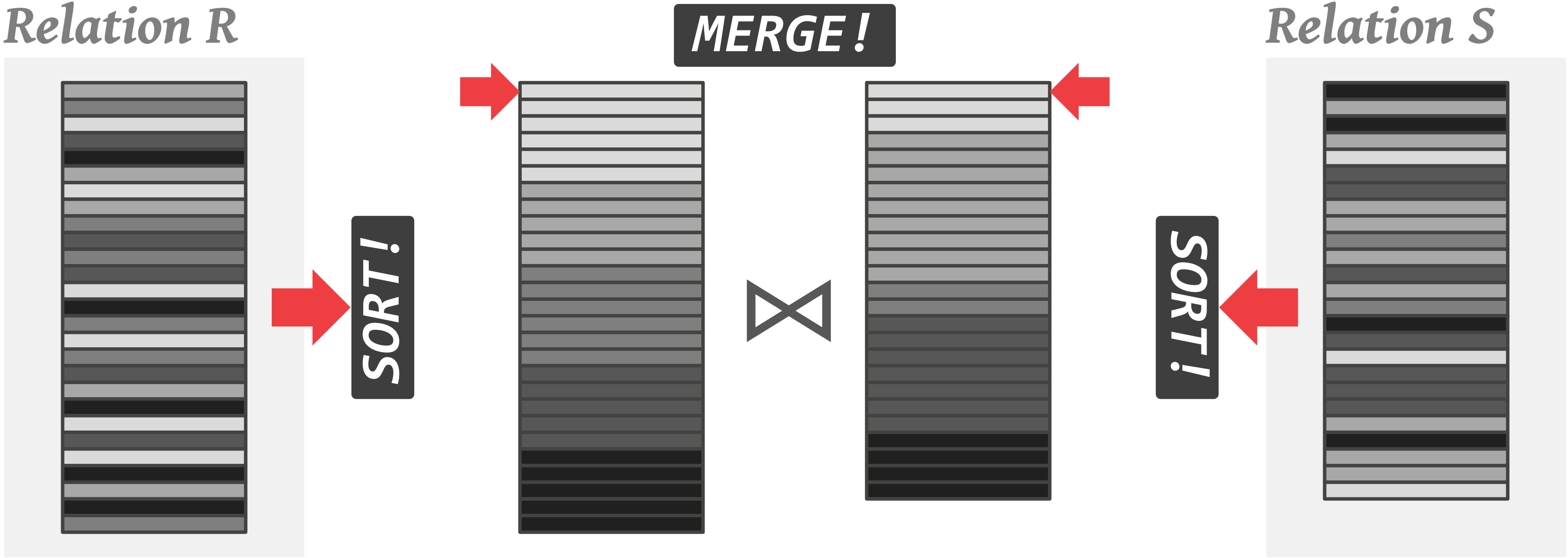
Relation S



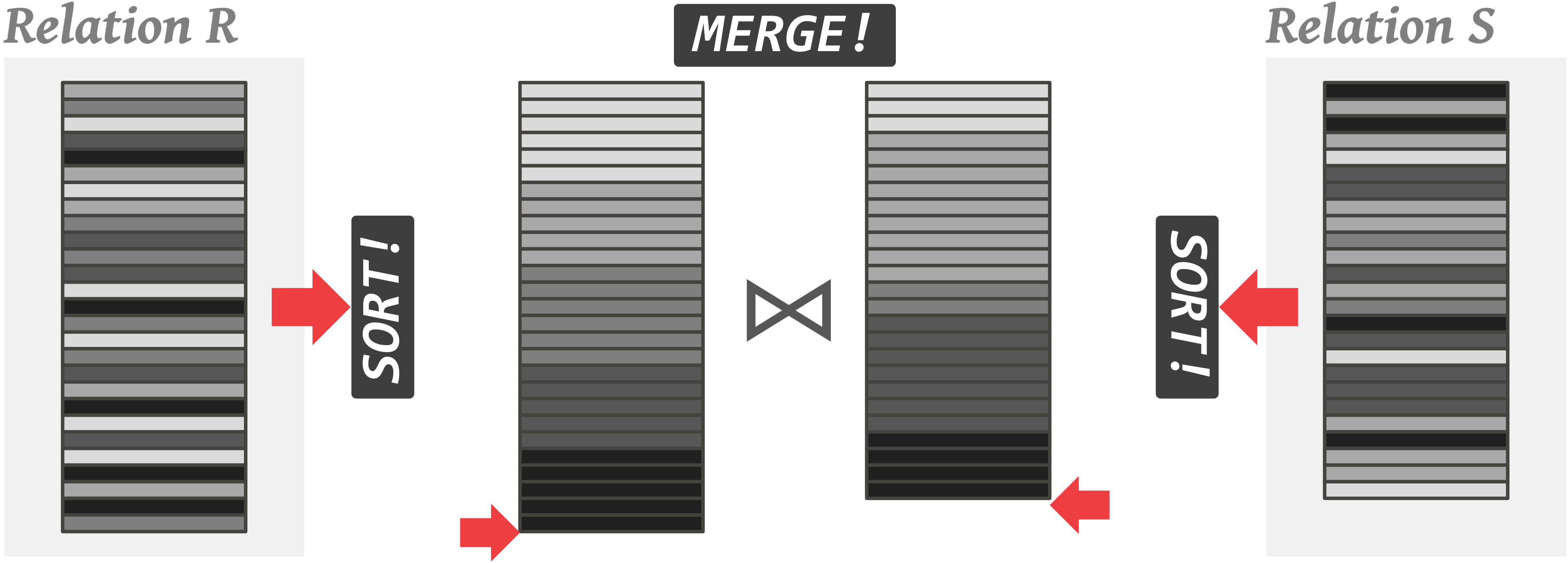
SORT-MERGE JOIN (R⋈S)



SORT-MERGE JOIN (R⋈S)



SORT-MERGE JOIN (R⋈S)



SORTING VS. HASHING

1970s – Sorting
1980s – Hashing
1990s – Both
2000s – Hashing
2010s – ???

IN-MEMORY JOINS

SORT VS. HASH REVISITED: FAST JOIN IMPLEMENTATION ON MODERN MULTI-CORE CPUS
VLDB 2009

→ Hashing is faster than Sort-Merge.
→ Sort-Merge will be faster with wider SIMD.

MASSIVELY PARALLEL SORT-MERGE JOINS IN MAIN MEMORY MULTI-CORE DATABASE SYSTEMS
VLDB 2012

→ Sort-Merge is already faster, even without SIMD.

MAIN-MEMORY HASH JOINS ON MULTI-CORE CPUS: TUNING TO THE UNDERLYING HARDWARE
ICDE 2013

→ New optimizations and results for Radix Hash Join.

Source: Cagri Balkesen

SINGLE INSTRUCTION, MULTIPLE DATA

A class of CPU instructions that allow the processor to perform the same operation on multiple data points simultaneously.

Both current AMD and Intel CPUs have ISA and microarchitecture support SIMD operations.
→ MMX, 3DNow!, SSE, SSE2, SSE3, SSE4, AVX

STREAMING SIMD EXTENSIONS (SSE)

SSE is a collection SIMD instructions that target special 128-bit SIMD registers.

These registers can be packed with four 32-bit scalars after which an operation can be performed on each of the four elements simultaneously.

First introduced by Intel in 1999.

SIMD EXAMPLE

X + Y = Z

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \dots \\ x_n + y_n \end{pmatrix}$$

SIMD EXAMPLE

X + Y = Z

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \dots \\ x_n + y_n \end{pmatrix}$$

```
for (i=0; i<n; i++) {
  Z[i] = X[i] + Y[i];
}
```

SIMD EXAMPLE

X + Y = Z

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \dots \\ x_n + y_n \end{pmatrix}$$

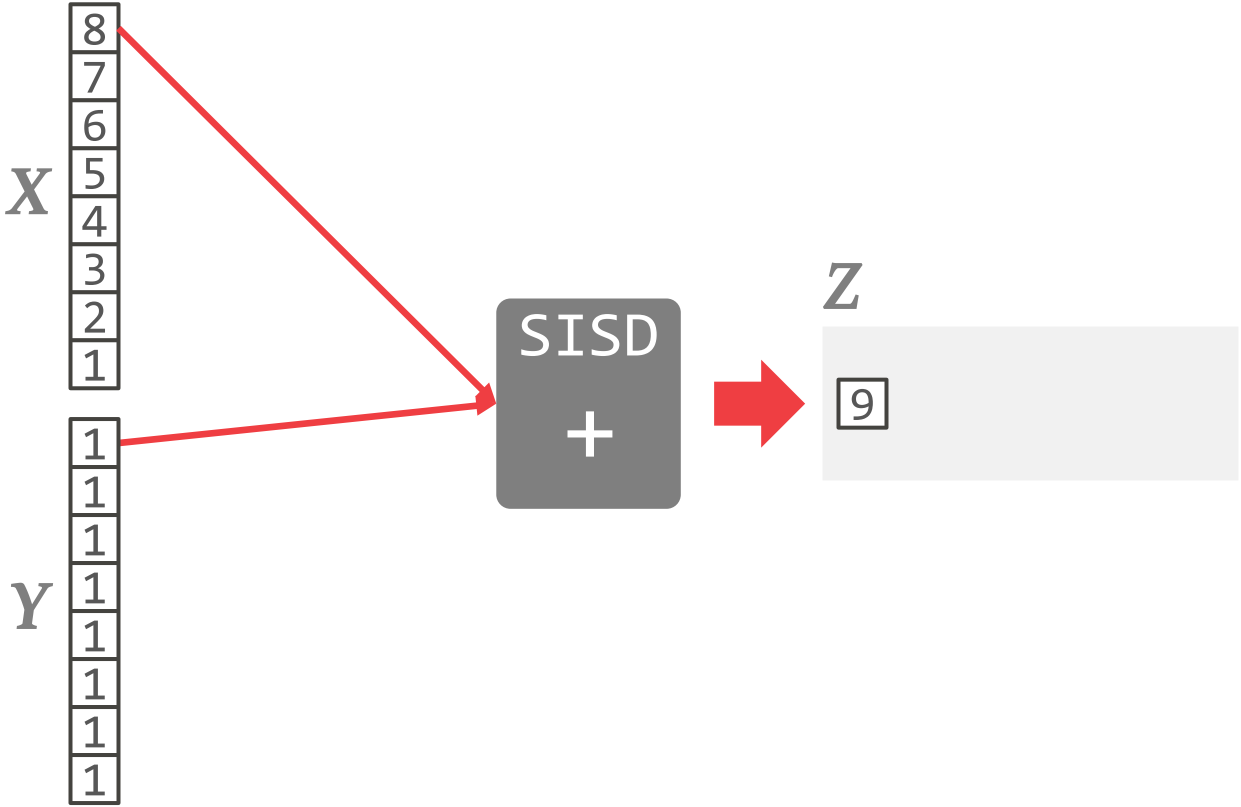
```
for (i=0; i<n; i++) {
  Z[i] = X[i] + Y[i];
}
```

SIMD EXAMPLE

X + Y = Z

$$\begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \cdots \\ x_n + y_n \end{pmatrix}$$

```
for (i=0; i<n; i++) {
  Z[i] = X[i] + Y[i];
}
```

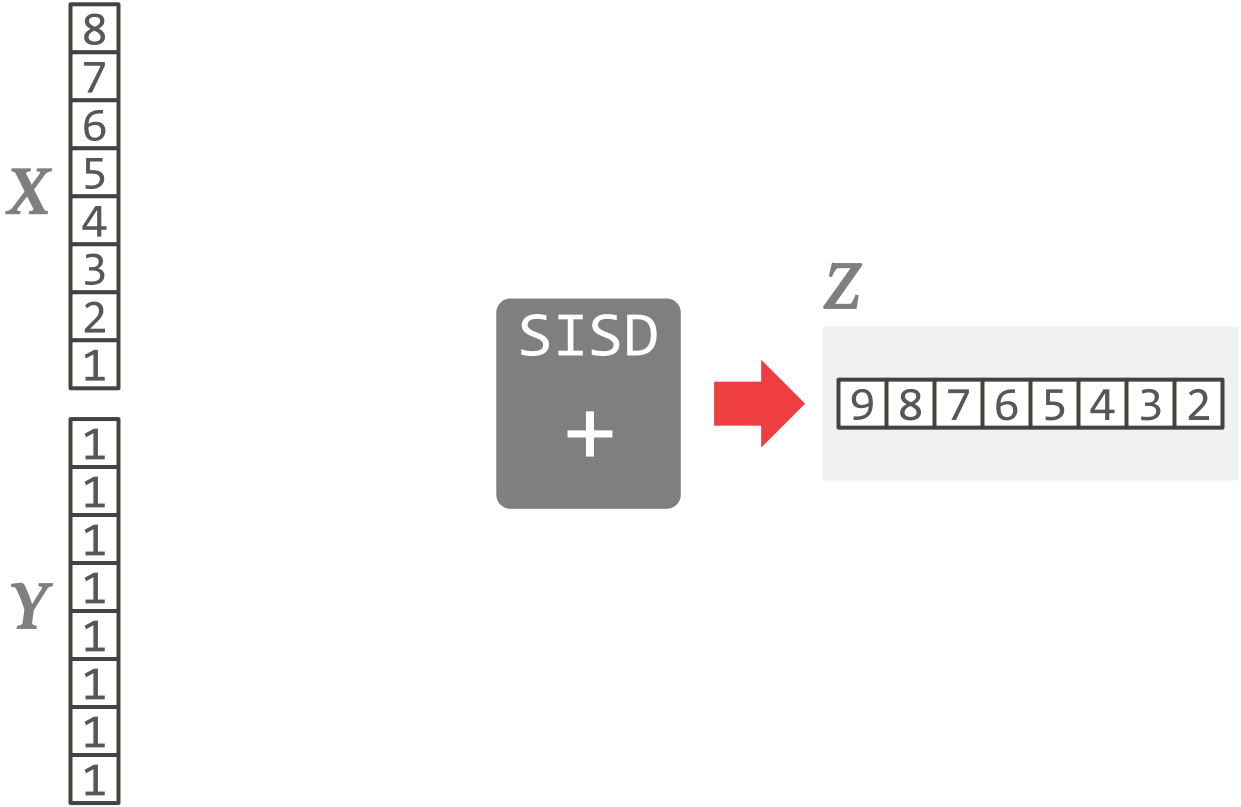


SIMD EXAMPLE

X + Y = Z

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \dots \\ x_n + y_n \end{pmatrix}$$

```
for (i=0; i<n; i++) {
  Z[i] = X[i] + Y[i];
}
```



SIMD EXAMPLE

$X + Y = Z$

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \dots \\ x_n + y_n \end{pmatrix}$$

```
for (i=0; i<n; i++) {
  Z[i] = X[i] + Y[i];
}
```

X

8
7
6
5
4
3
2
1

Y

1
1
1
1
1
1
1
1

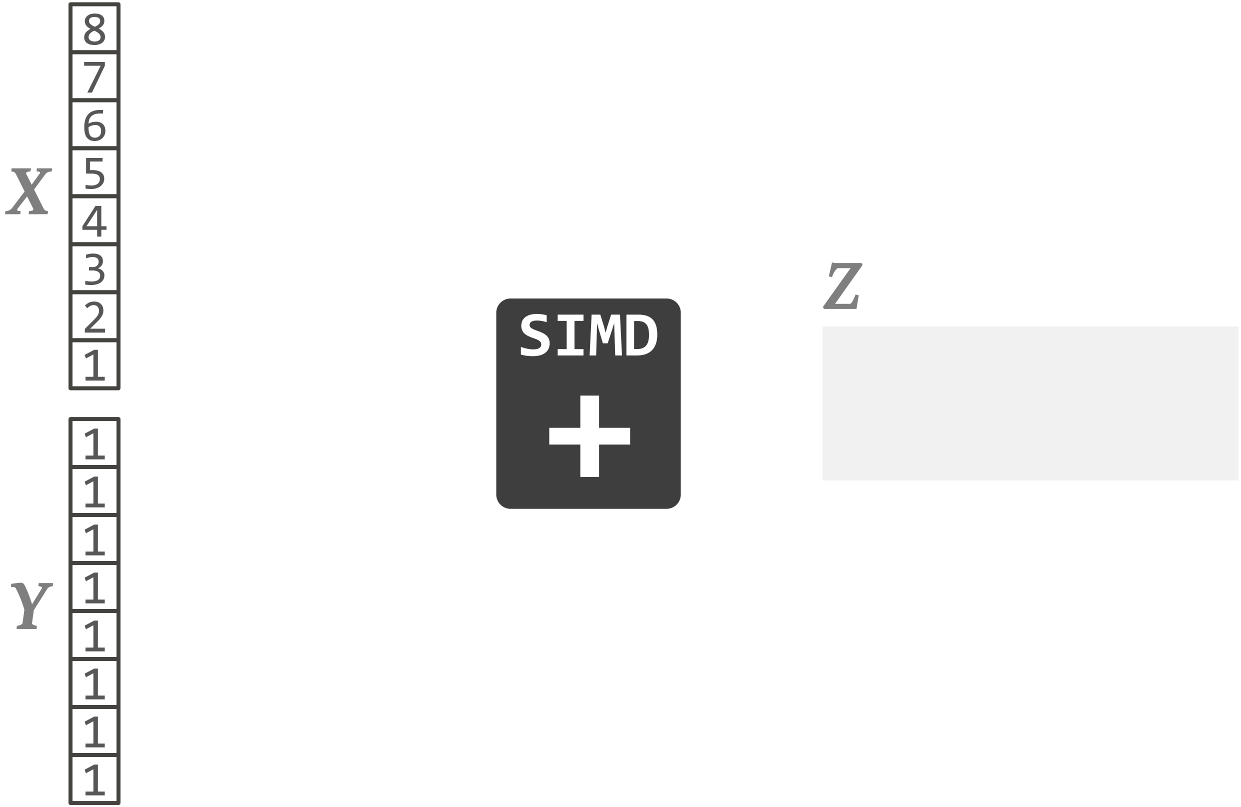
Z

SIMD EXAMPLE

X + Y = Z

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \dots \\ x_n + y_n \end{pmatrix}$$

```
for (i=0; i<n; i++) {
  Z[i] = X[i] + Y[i];
}
```

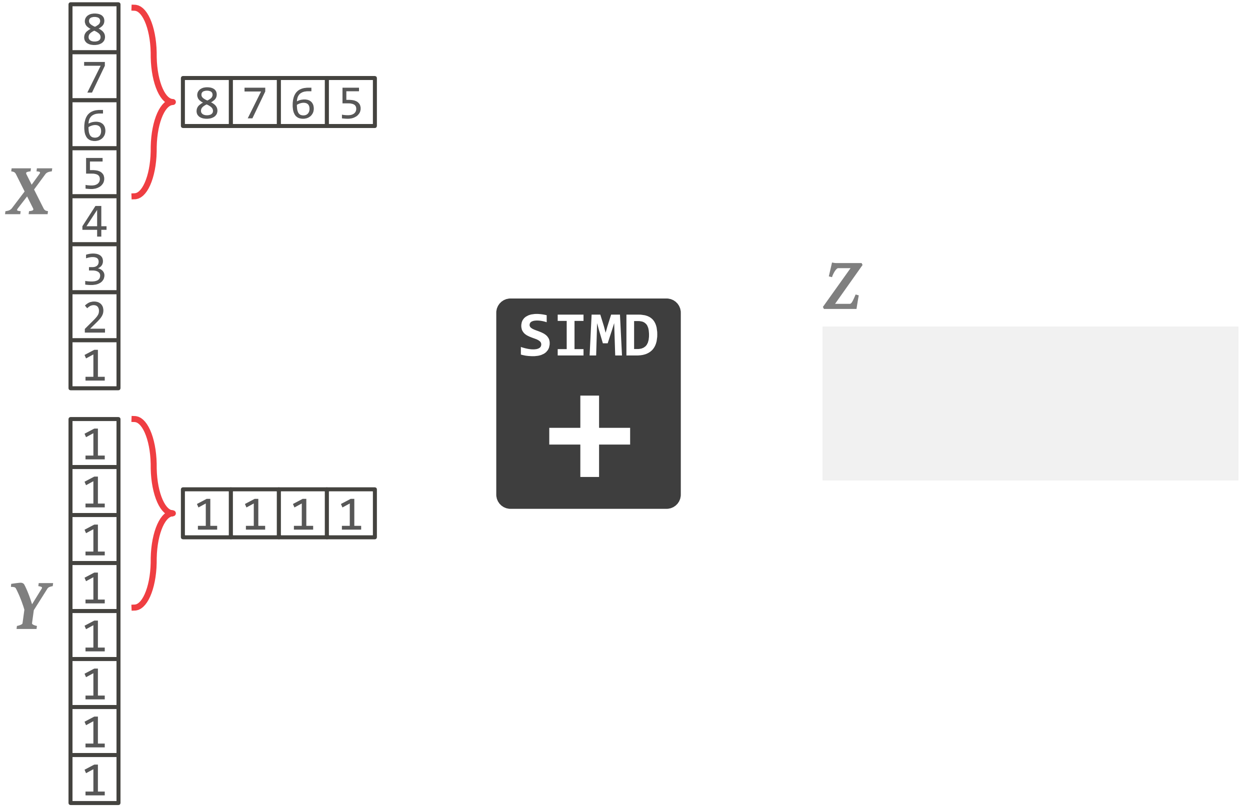


SIMD EXAMPLE

$X + Y = Z$

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \dots \\ x_n + y_n \end{pmatrix}$$

```
for (i=0; i<n; i++) {
  Z[i] = X[i] + Y[i];
}
```

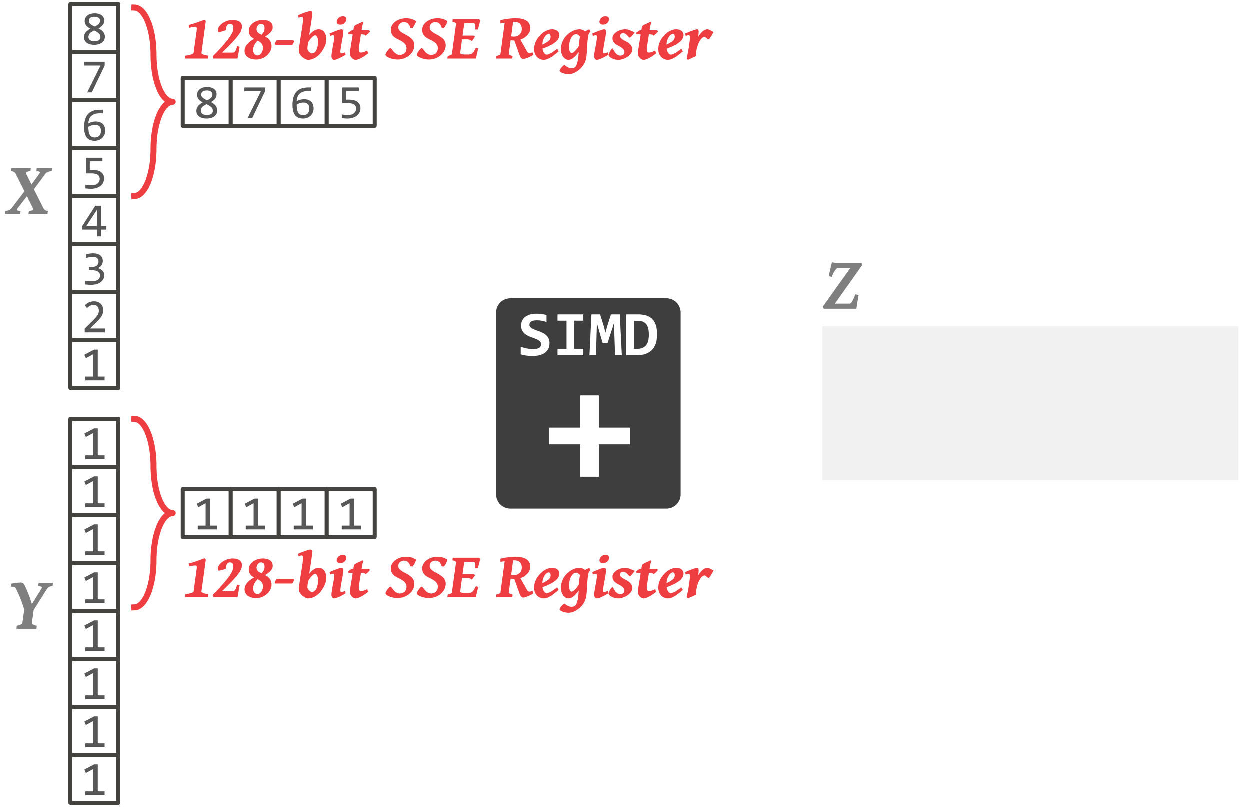


SIMD EXAMPLE

X + Y = Z

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \dots \\ x_n + y_n \end{pmatrix}$$

```
for (i=0; i<n; i++) {
  Z[i] = X[i] + Y[i];
}
```

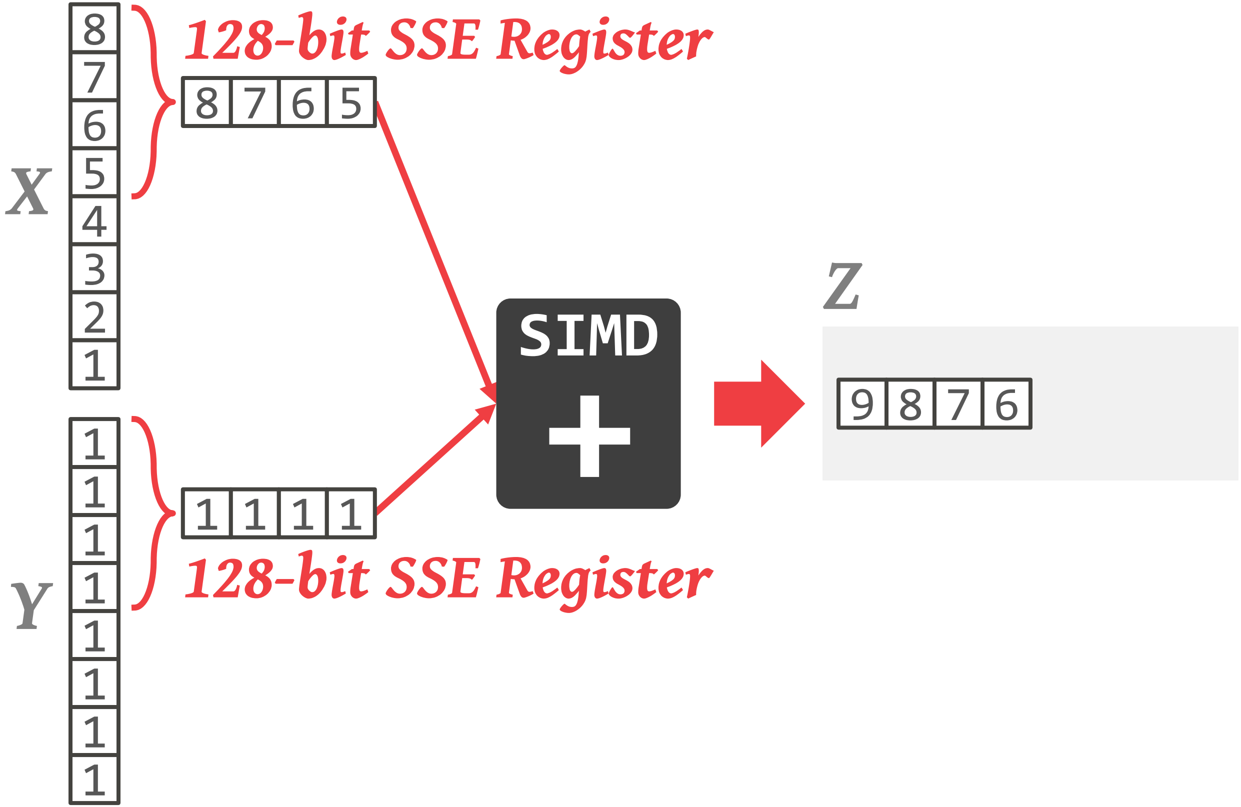


SIMD EXAMPLE

X + Y = Z

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \dots \\ x_n + y_n \end{pmatrix}$$

```
for (i=0; i<n; i++) {
  Z[i] = X[i] + Y[i];
}
```

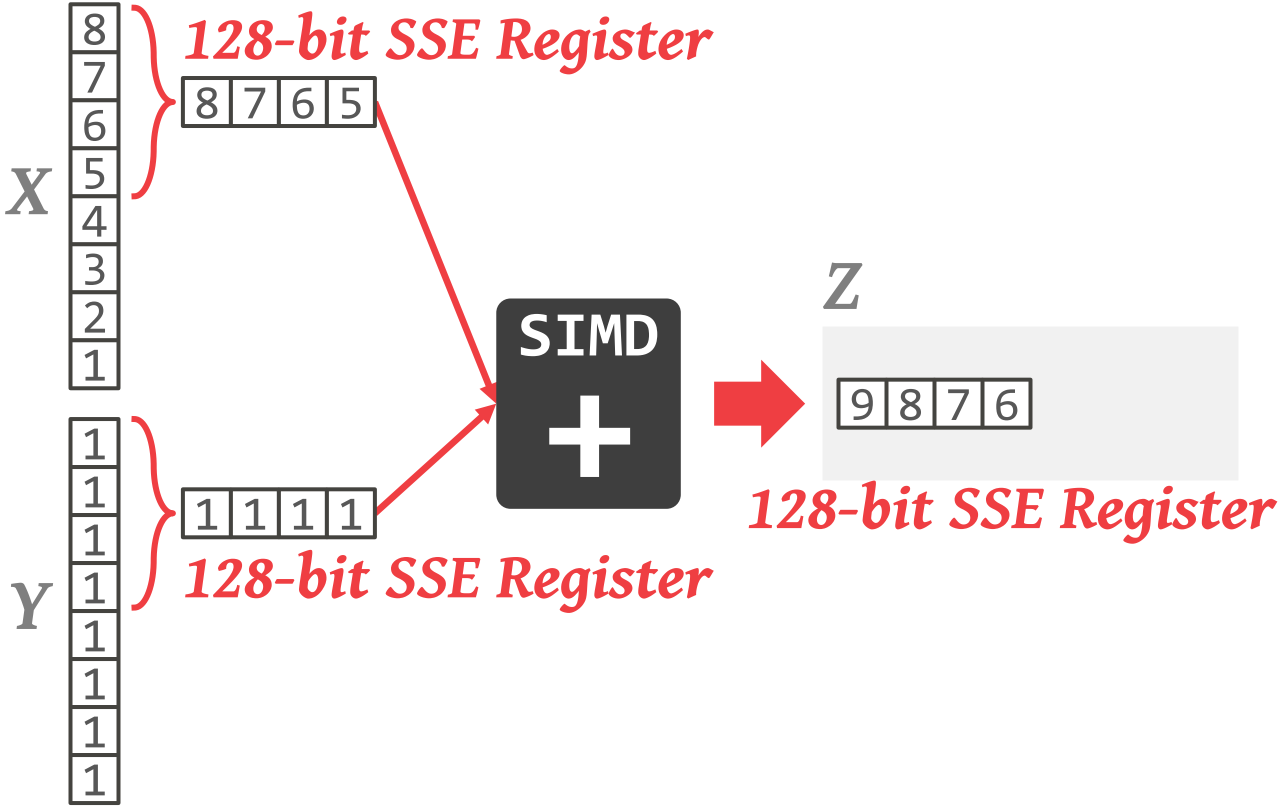


SIMD EXAMPLE

X + Y = Z

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \dots \\ x_n + y_n \end{pmatrix}$$

```
for (i=0; i<n; i++) {
  Z[i] = X[i] + Y[i];
}
```

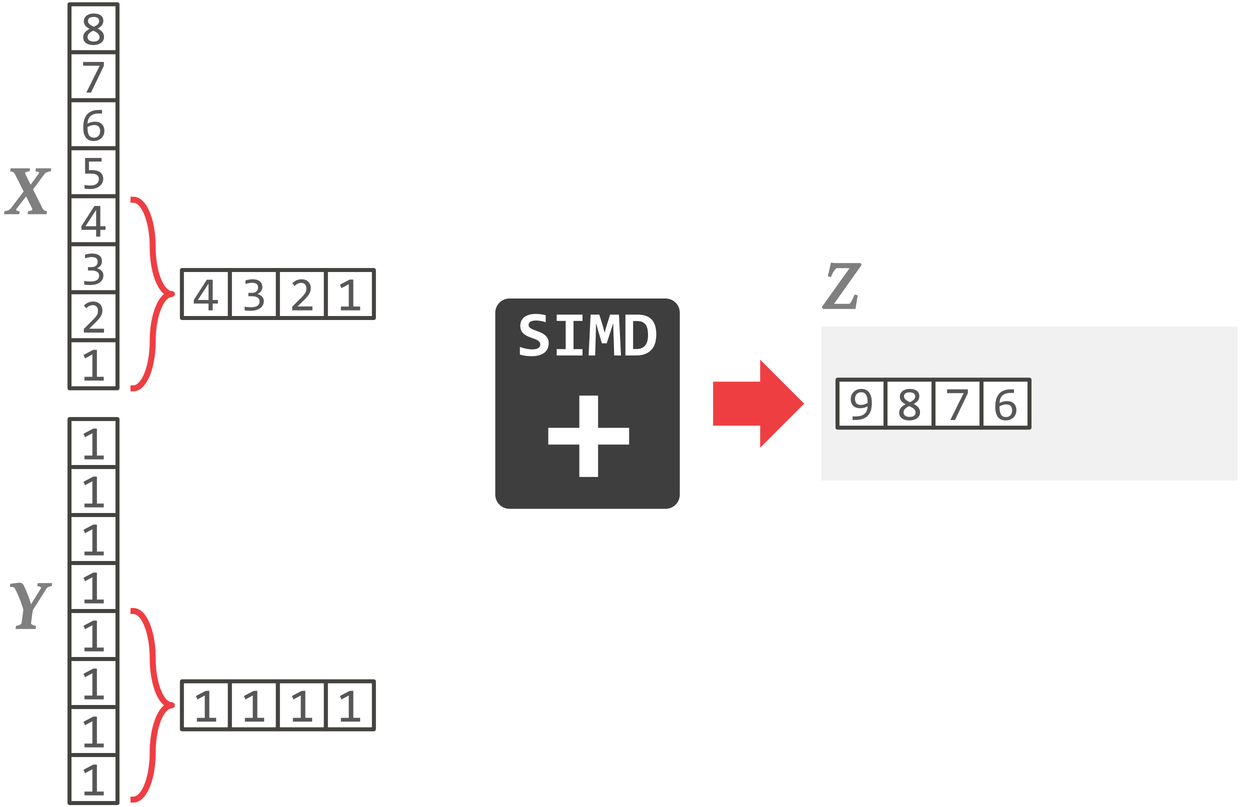


SIMD EXAMPLE

$X + Y = Z$

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \dots \\ x_n + y_n \end{pmatrix}$$

```
for (i=0; i<n; i++) {
  Z[i] = X[i] + Y[i];
}
```

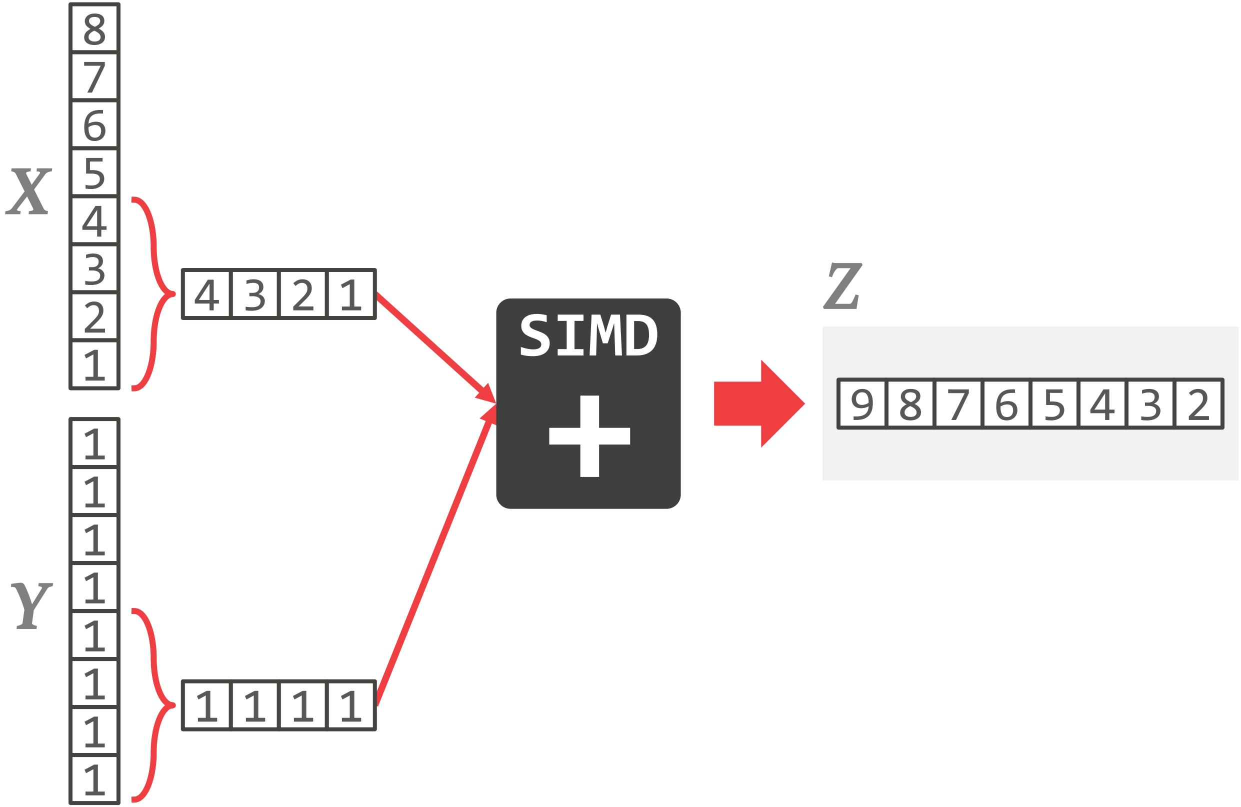


SIMD EXAMPLE

X + Y = Z

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \dots \\ x_n + y_n \end{pmatrix}$$

```
for (i=0; i<n; i++) {
  Z[i] = X[i] + Y[i];
}
```



SIMD TRADE-OFFS

Advantages:

→ Significant performance gains and resource utilization if an algorithm can be vectorized.

Disadvantages:

→ Implementing an algorithm using SIMD is still mostly a manual process.
→ SIMD may have restrictions on data alignment.
→ Gathering data into SIMD registers and scattering it to the correct locations is tricky and/or inefficient.

WHY NOT GPUS?

Moving data back and forth between DRAM and GPU is slow over PCI-E bus.

Emerging co-processors that can share CPU's memory may change this.
→ Examples: AMD's APU, Intel's Knights Landing

PARALLEL SORT-MERGE JOINS

Sorting is always the most expensive part.

Take advantage of new hardware to speed things up as much as possible.

→ Utilize as many CPU cores as possible.
→ Be mindful of NUMA boundaries.

MULTI-CORE, MAIN-MEMORY JOINS: SORT VS. HASH REVISITED
VLDB 2013

PARALLEL SORT-MERGE JOIN (R⋈S)

Phase #1: Partitioning (optional)

→ Partition **S** and assign them to workers / cores.

Phase #2: Sort

→ Sort the tuples of **R** and **S** based on the join key.

Phase #3: Merge

→ Scan the sorted relations and compare tuples.
→ The outer relation **R** only needs to be scanned once.

PARTITIONING PHASE

Divide the relations into chunks and assign them to cores.

→ Explicit vs. Implicit

Explicit: Divide only the outer relation and redistribute among the different CPU cores.

→ Can use the same radix partitioning approach we talked about last time.

SORT PHASE

Create **runs** of sorted chunks of tuples for both input relations.

It used to be that Quicksort was good enough.

But NUMA and parallel architectures require us to be more careful…

CACHE-CONSCIOUS SORTING

Level #1: In-Register Sorting

→ Sort runs that fit into CPU registers.

Level #2: In-Cache Sorting

→ Merge the output of Level #1 into runs that fit into CPU caches.

→ Repeat until sorted runs are ½ cache size.

Level #3: Out-of-Cache Sorting

→ Used when the runs of Level #2 exceed the size of caches.

CACHE-CONSCIOUS SORTING

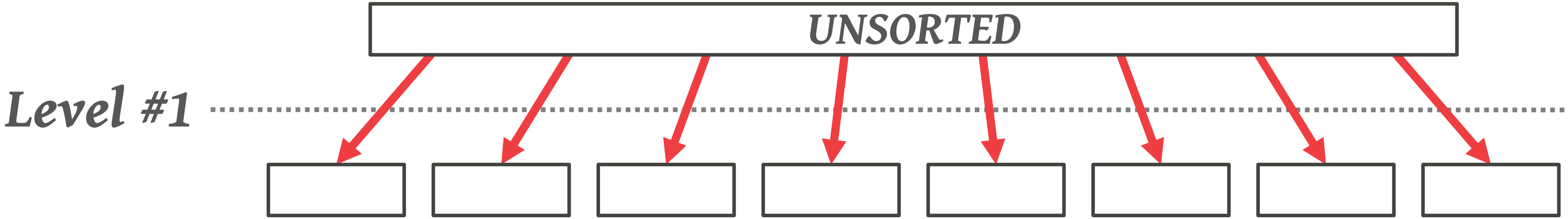
UNSORTED

CACHE-CONSCIOUS SORTING

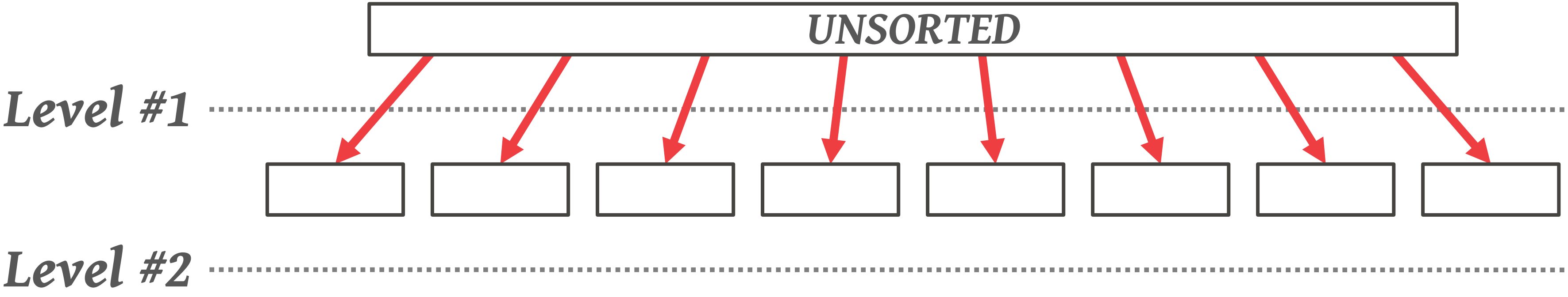
UNSORTED

Level #1

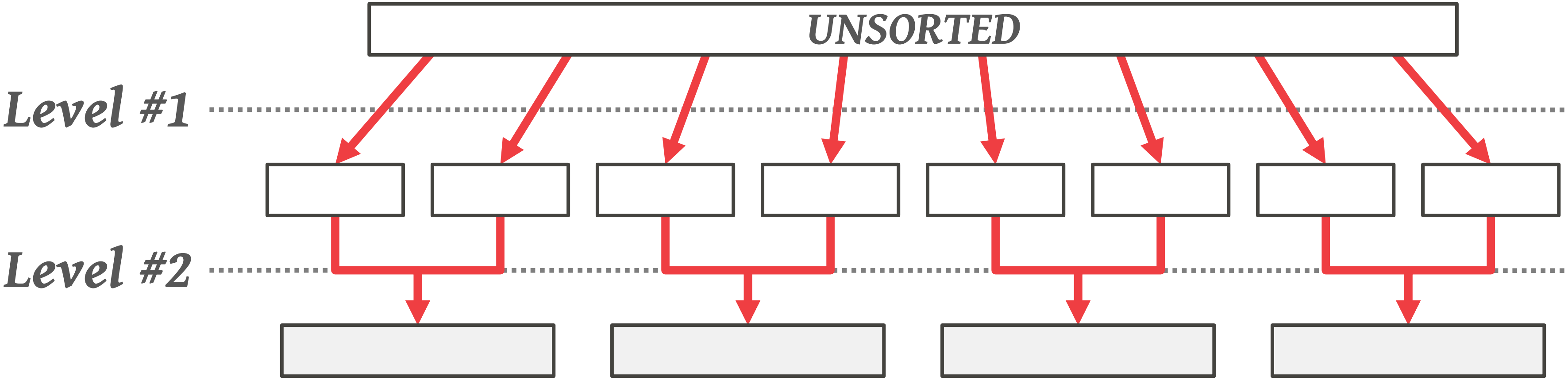
CACHE-CONSCIOUS SORTING



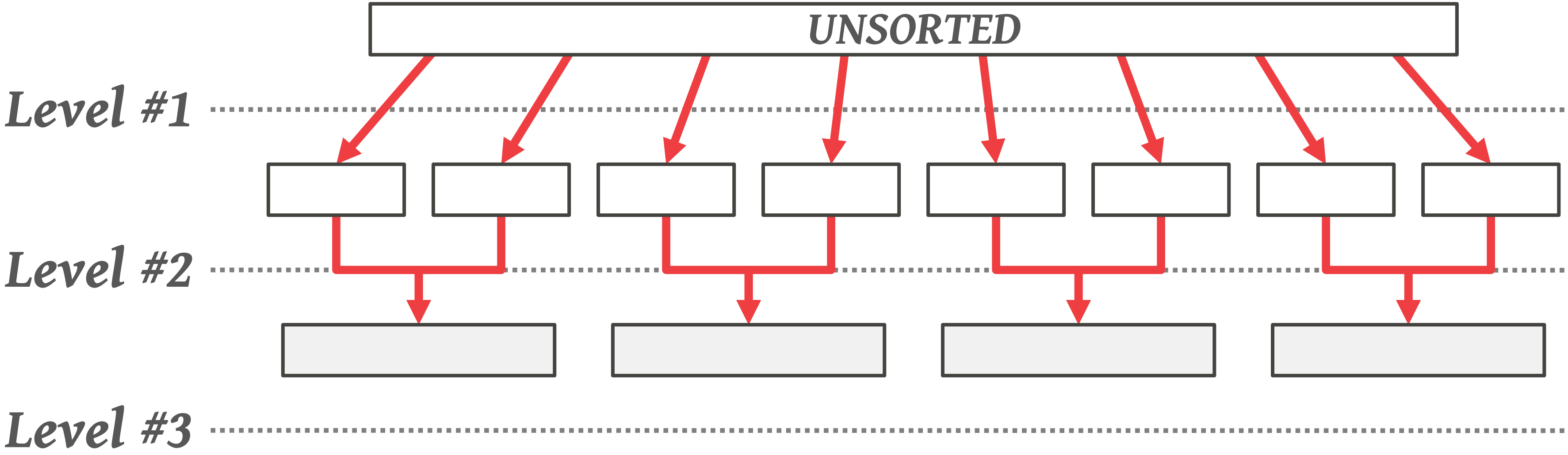
CACHE-CONSCIOUS SORTING



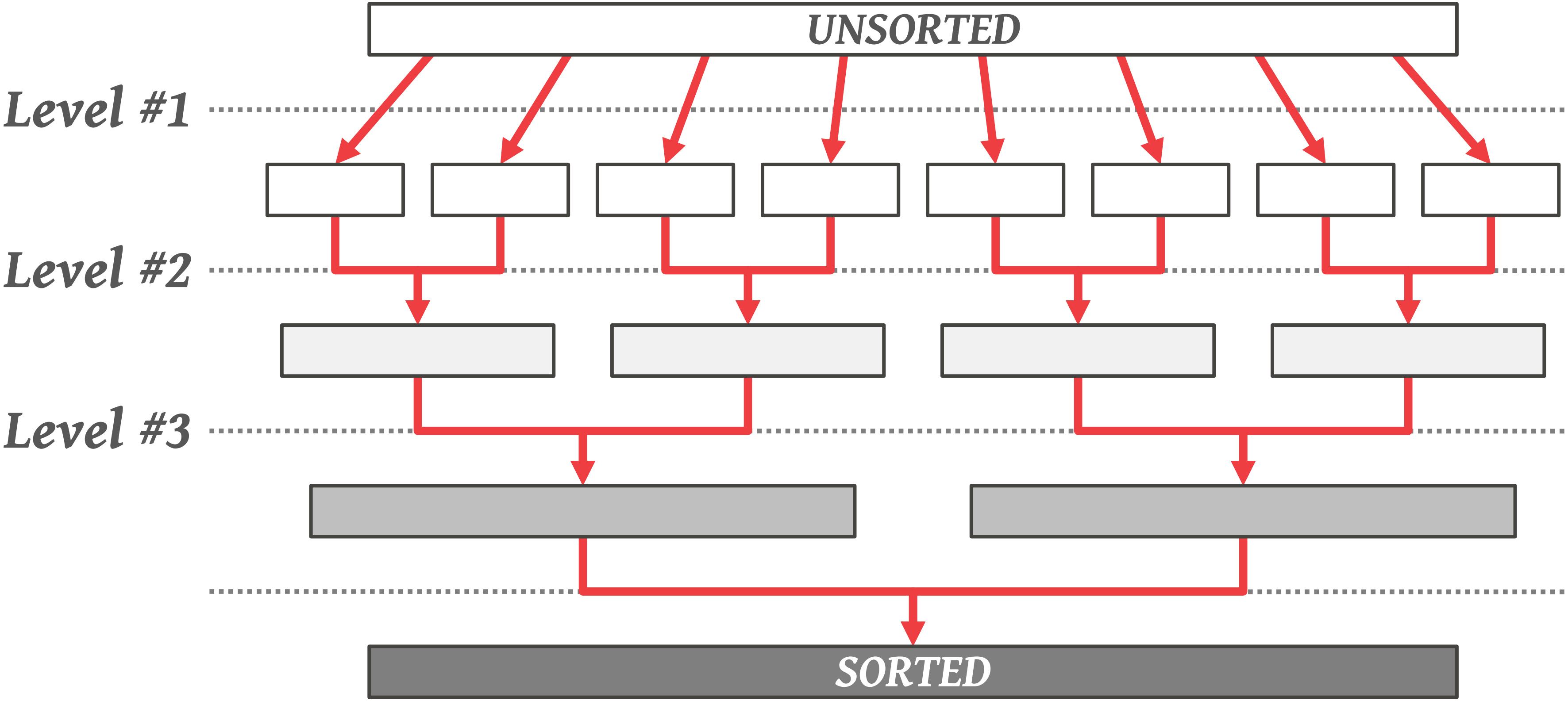
CACHE-CONSCIOUS SORTING



CACHE-CONSCIOUS SORTING



CACHE-CONSCIOUS SORTING

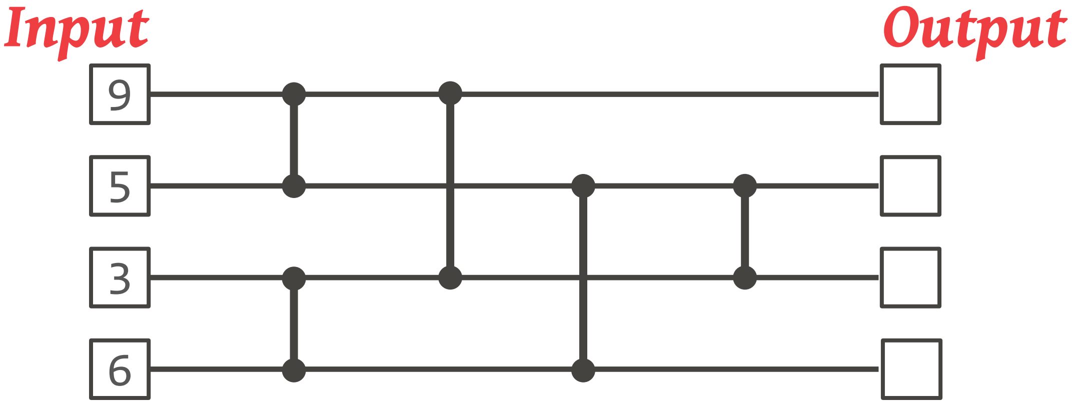


LEVEL #1 – SORTING NETWORKS

Abstract model for sorting keys.

→ Always has fixed wiring "paths" for lists with the same number of elements.

→ Efficient to execute on modern CPUs because of limited data dependencies and no branches.

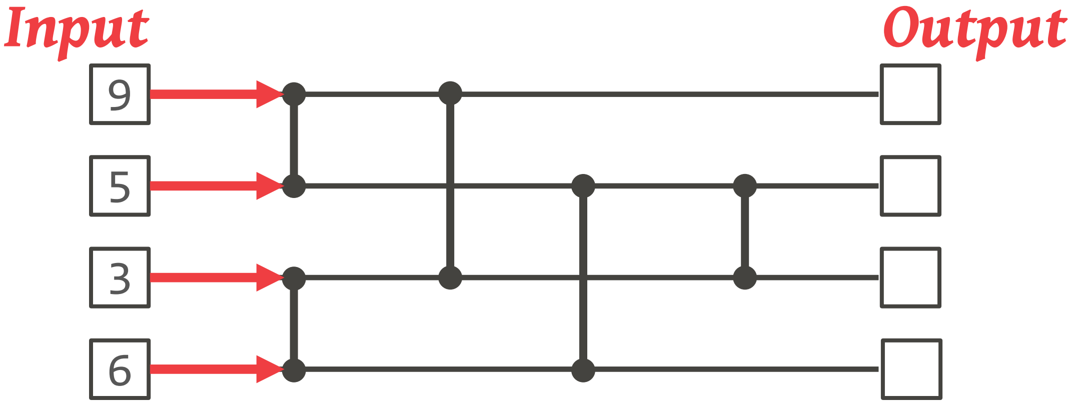


LEVEL #1 – SORTING NETWORKS

Abstract model for sorting keys.

→ Always has fixed wiring "paths" for lists with the same number of elements.

→ Efficient to execute on modern CPUs because of limited data dependencies and no branches.

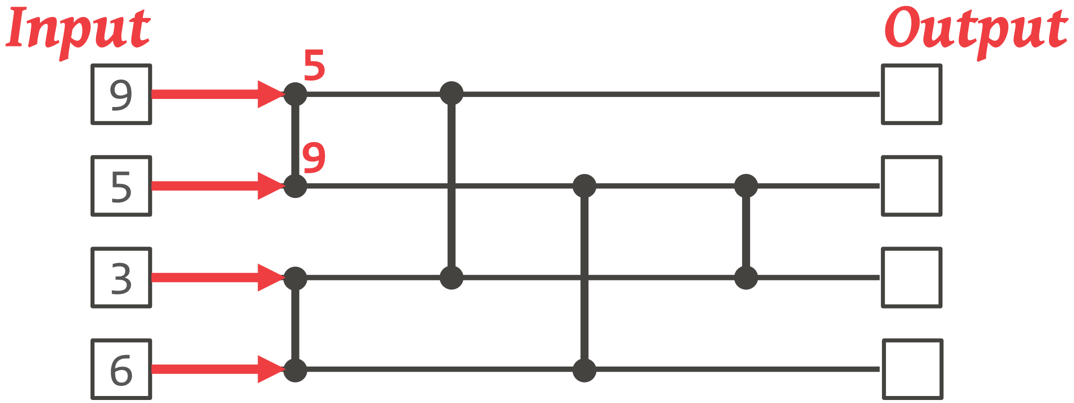


LEVEL #1 – SORTING NETWORKS

Abstract model for sorting keys.

→ Always has fixed wiring "paths" for lists with the same number of elements.

→ Efficient to execute on modern CPUs because of limited data dependencies and no branches.

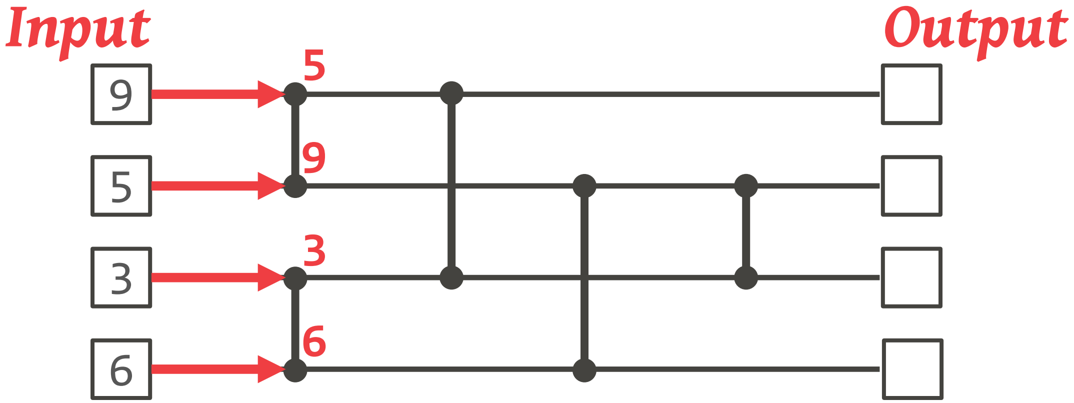


LEVEL #1 – SORTING NETWORKS

Abstract model for sorting keys.

→ Always has fixed wiring "paths" for lists with the same number of elements.

→ Efficient to execute on modern CPUs because of limited data dependencies and no branches.

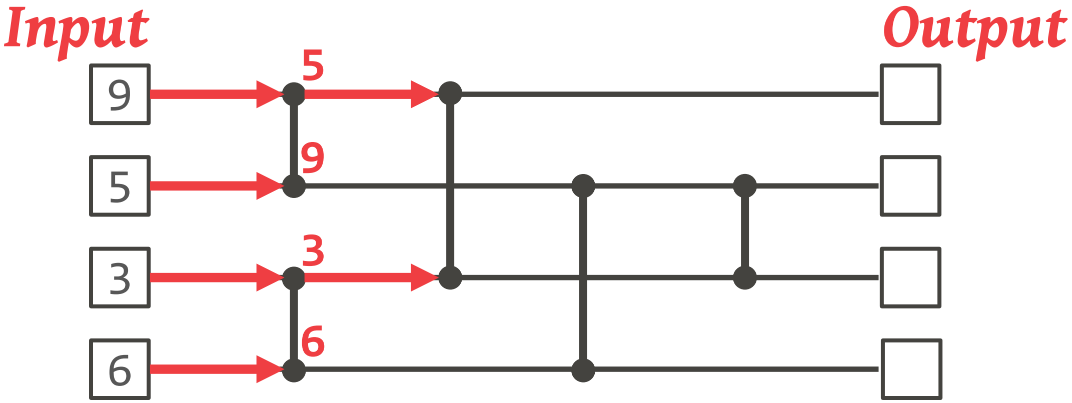


LEVEL #1 – SORTING NETWORKS

Abstract model for sorting keys.

→ Always has fixed wiring "paths" for lists with the same number of elements.

→ Efficient to execute on modern CPUs because of limited data dependencies and no branches.

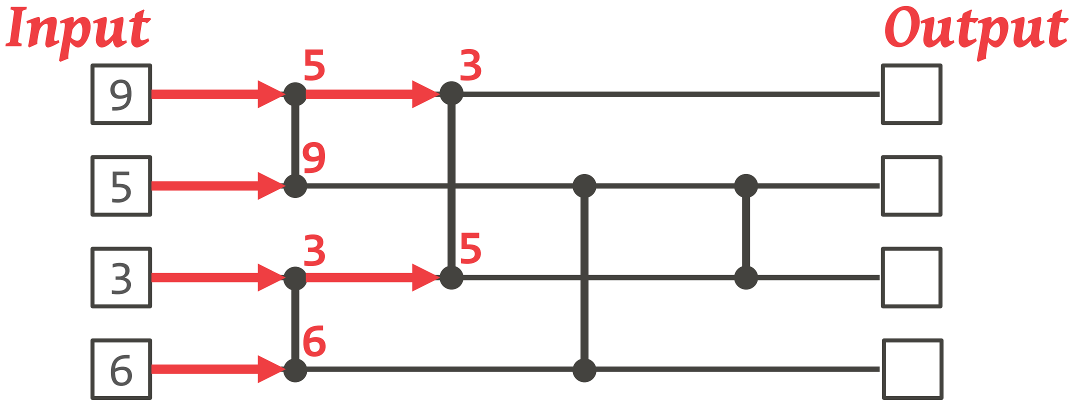


LEVEL #1 – SORTING NETWORKS

Abstract model for sorting keys.

→ Always has fixed wiring "paths" for lists with the same number of elements.

→ Efficient to execute on modern CPUs because of limited data dependencies and no branches.

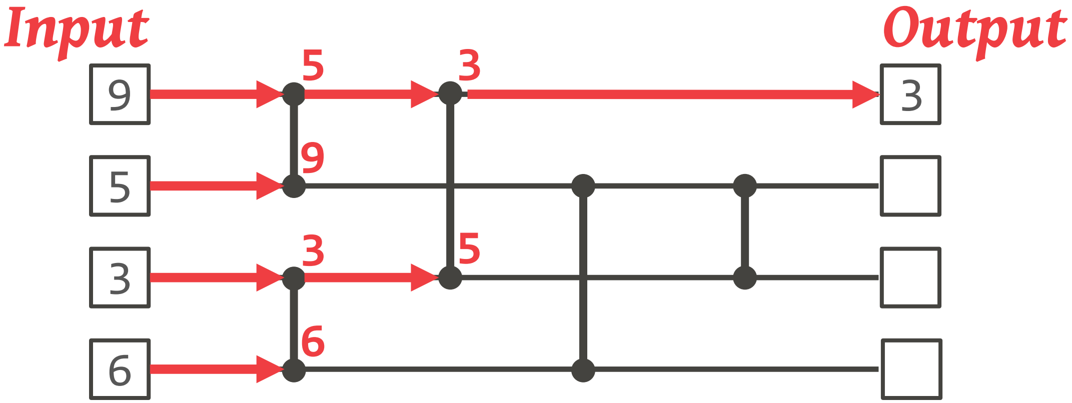


LEVEL #1 – SORTING NETWORKS

Abstract model for sorting keys.

→ Always has fixed wiring “paths” for lists with the same number of elements.

→ Efficient to execute on modern CPUs because of limited data dependencies and no branches.

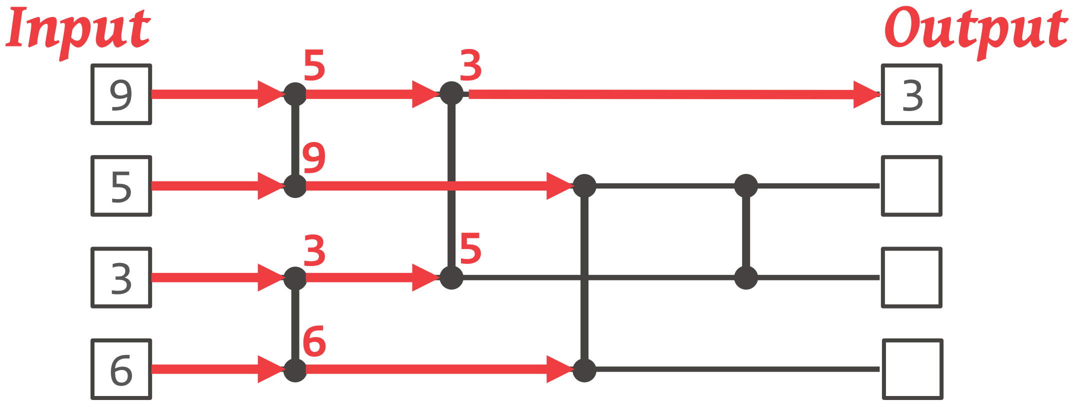


LEVEL #1 – SORTING NETWORKS

Abstract model for sorting keys.

→ Always has fixed wiring "paths" for lists with the same number of elements.

→ Efficient to execute on modern CPUs because of limited data dependencies and no branches.

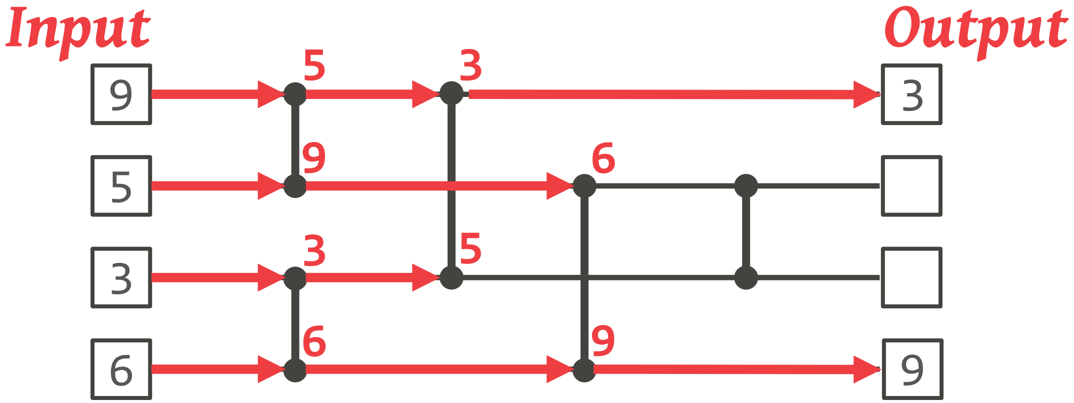


LEVEL #1 – SORTING NETWORKS

Abstract model for sorting keys.

→ Always has fixed wiring "paths" for lists with the same number of elements.

→ Efficient to execute on modern CPUs because of limited data dependencies and no branches.

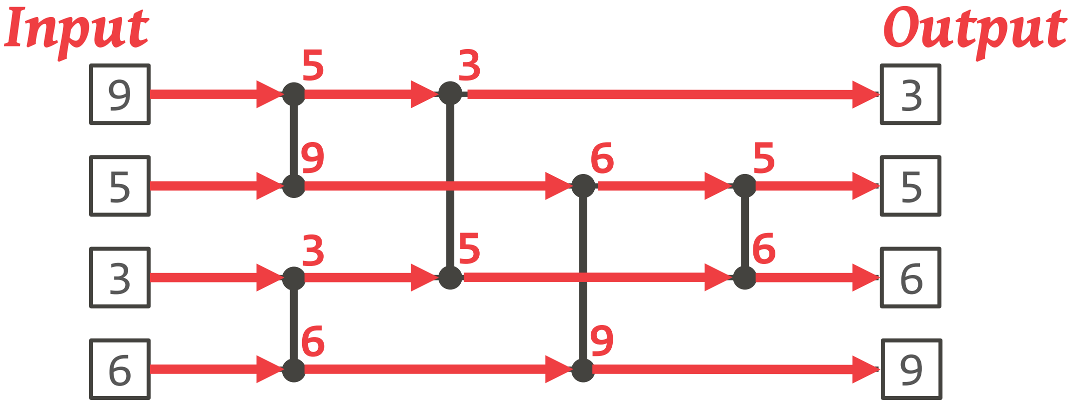


LEVEL #1 – SORTING NETWORKS

Abstract model for sorting keys.

→ Always has fixed wiring "paths" for lists with the same number of elements.

→ Efficient to execute on modern CPUs because of limited data dependencies and no branches.



LEVEL #1 – SORTING NETWORKS

12	21	4	13
9	8	6	7
1	14	3	0
5	11	15	10

LEVEL #1 – SORTING NETWORKS

12	21	4	13
9	8	6	7
1	14	3	0
5	11	15	10

Instructions:

→ 4 **LOAD**

LEVEL #1 – SORTING NETWORKS

Sort Across Registers

12	21	4	13
9	8	6	7
1	14	3	0
5	11	15	10

Instructions:

→ 4 **LOAD**

LEVEL #1 – SORTING NETWORKS

Sort Across Registers

12	21	4	13
9	8	6	7
1	14	3	0
5	11	15	10

Instructions:

→ 4 **LOAD**

LEVEL #1 – SORTING NETWORKS

Instructions:

→ 4 **LOAD**

LEVEL #1 – SORTING NETWORKS

Sort Across Registers

12	21	4	13
9	8	6	7
1	14	3	0
5	11	15	10

1	8	3	0
5	11	4	7
9	14	6	10
12	21	15	13

Instructions:

→ 4 **LOAD**

Instructions:

→ 10 **MIN/MAX**

LEVEL #1 – SORTING NETWORKS

Sort Across Registers

12	21	4	13
9	8	6	7
1	14	3	0
5	11	15	10

Transpose Registers

1	8	3	0
5	11	4	7
9	14	6	10
12	21	15	13

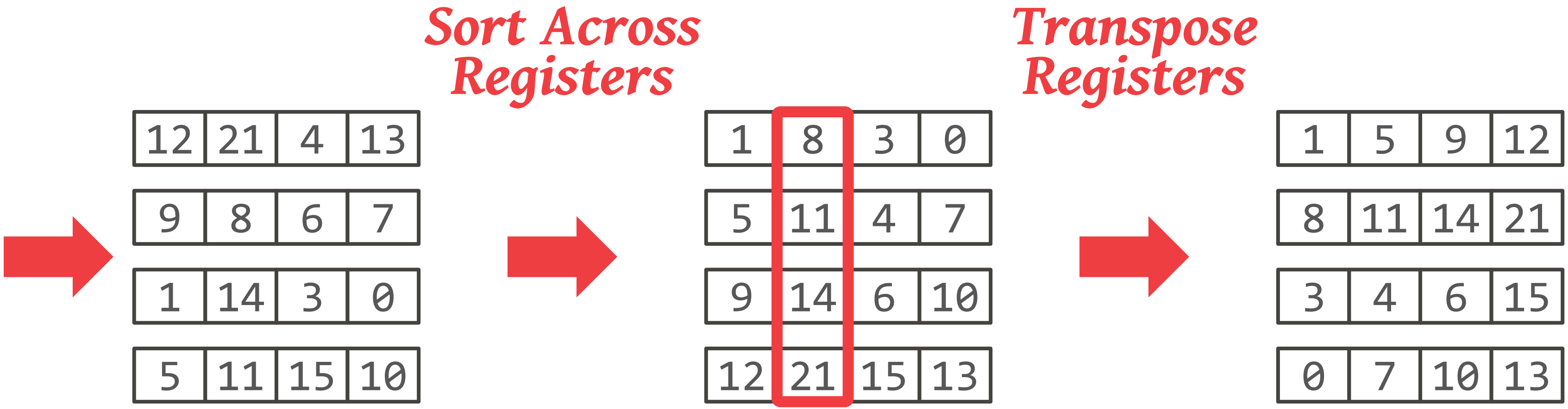
Instructions:

→ 4 **LOAD**

Instructions:

→ 10 **MIN/MAX**

LEVEL #1 – SORTING NETWORKS



Instructions:
→ 4 **LOAD**

Instructions:
→ 10 **MIN/MAX**

LEVEL #1 – SORTING NETWORKS

12	21	4	13
9	8	6	7
1	14	3	0
5	11	15	10

Sort Across Registers

1	8	3	0
5	11	4	7
9	14	6	10
12	21	15	13

Transpose Registers

1	5	9	12
8	11	14	21
3	4	6	15
0	7	10	13

Instructions:
→ 4 **LOAD**

Instructions:
→ 10 **MIN/MAX**

LEVEL #1 – SORTING NETWORKS

12	21	4	13
9	8	6	7
1	14	3	0
5	11	15	10

Instructions:
→ **4 LOAD**

Sort Across Registers

1	8	3	0
5	11	4	7
9	14	6	10
12	21	15	13

Instructions:
→ **10 MIN/MAX**

Transpose Registers

1	5	9	12
8	11	14	21
3	4	6	15
0	7	10	13

Instructions:
→ **8 SHUFFLE**
→ **4 STORE**

LEVEL #2 – BITONIC MERGE NETWORK

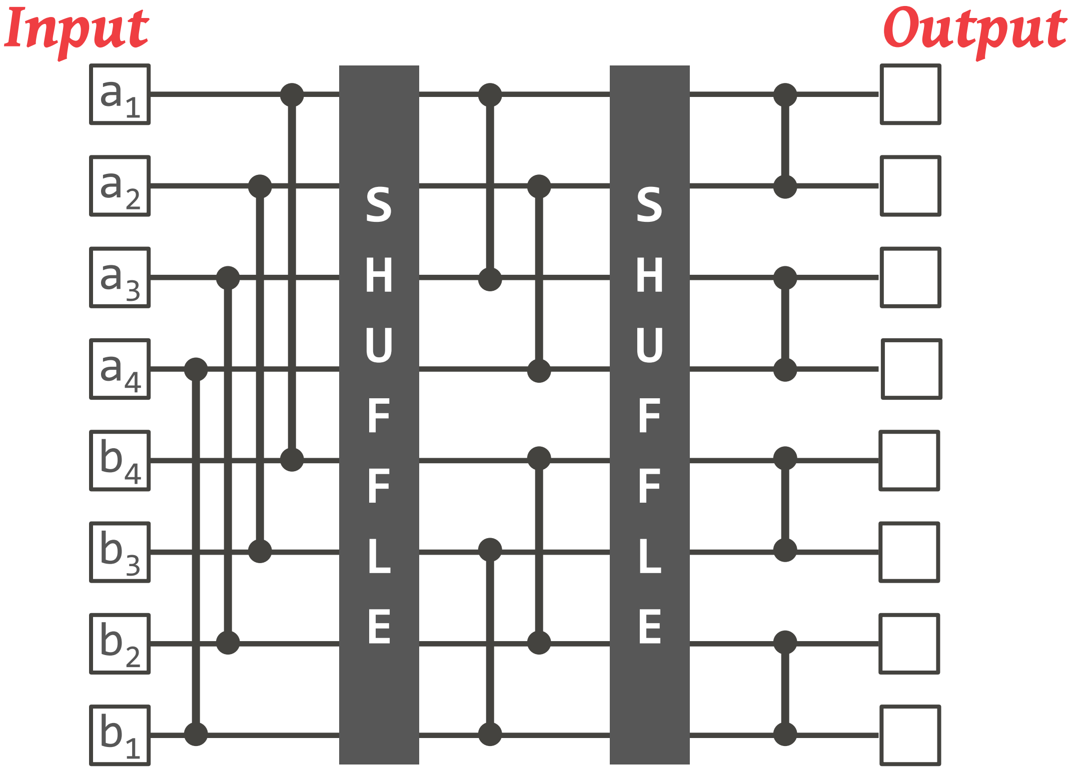
Like a Sorting Network but it can merge two locally-sorted lists into a globally-sorted list.

Can expand network to merge progressively larger lists (½ cache size).

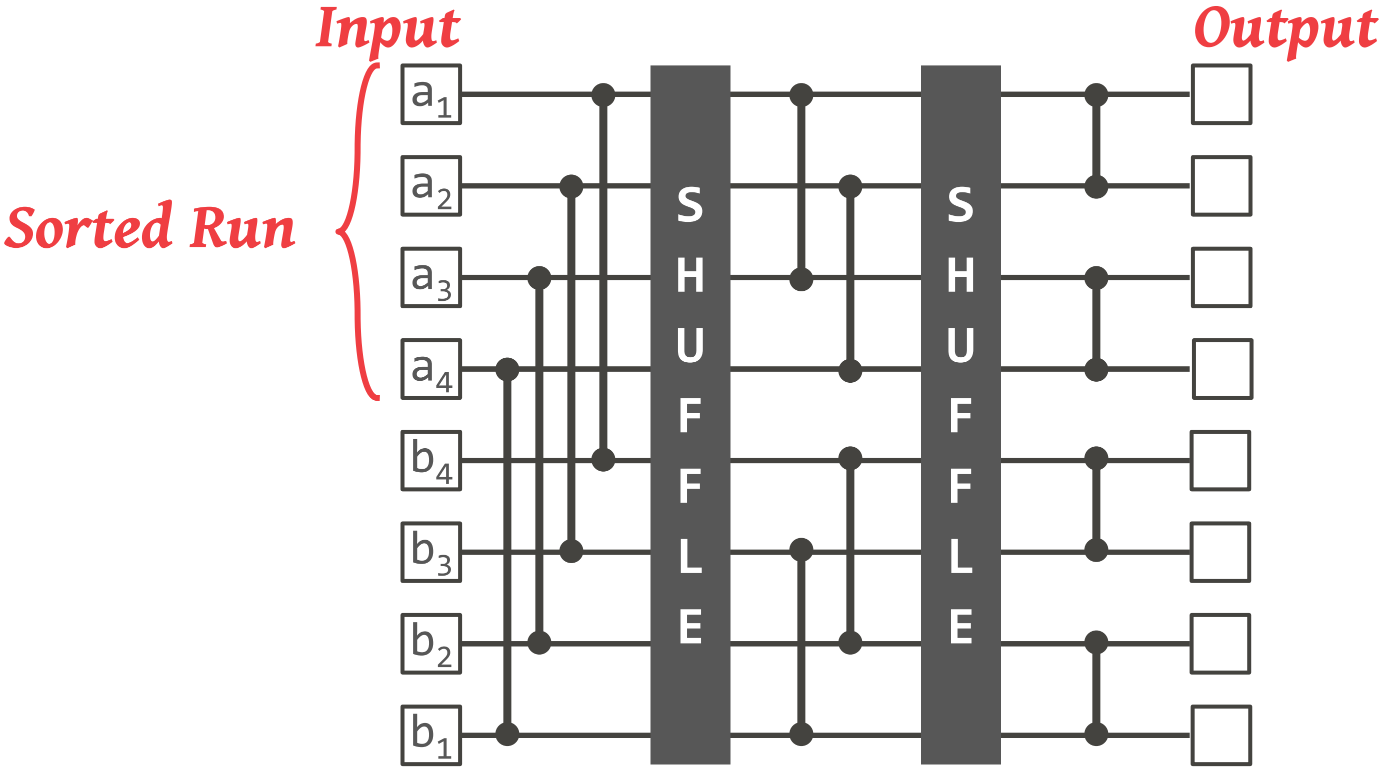
Intel's Measurements
→ 2.25–3.5x speed-up over SISD implementation.

EFFICIENT IMPLEMENTATION OF SORTING ON MULTI-CORE
VLDB 2008

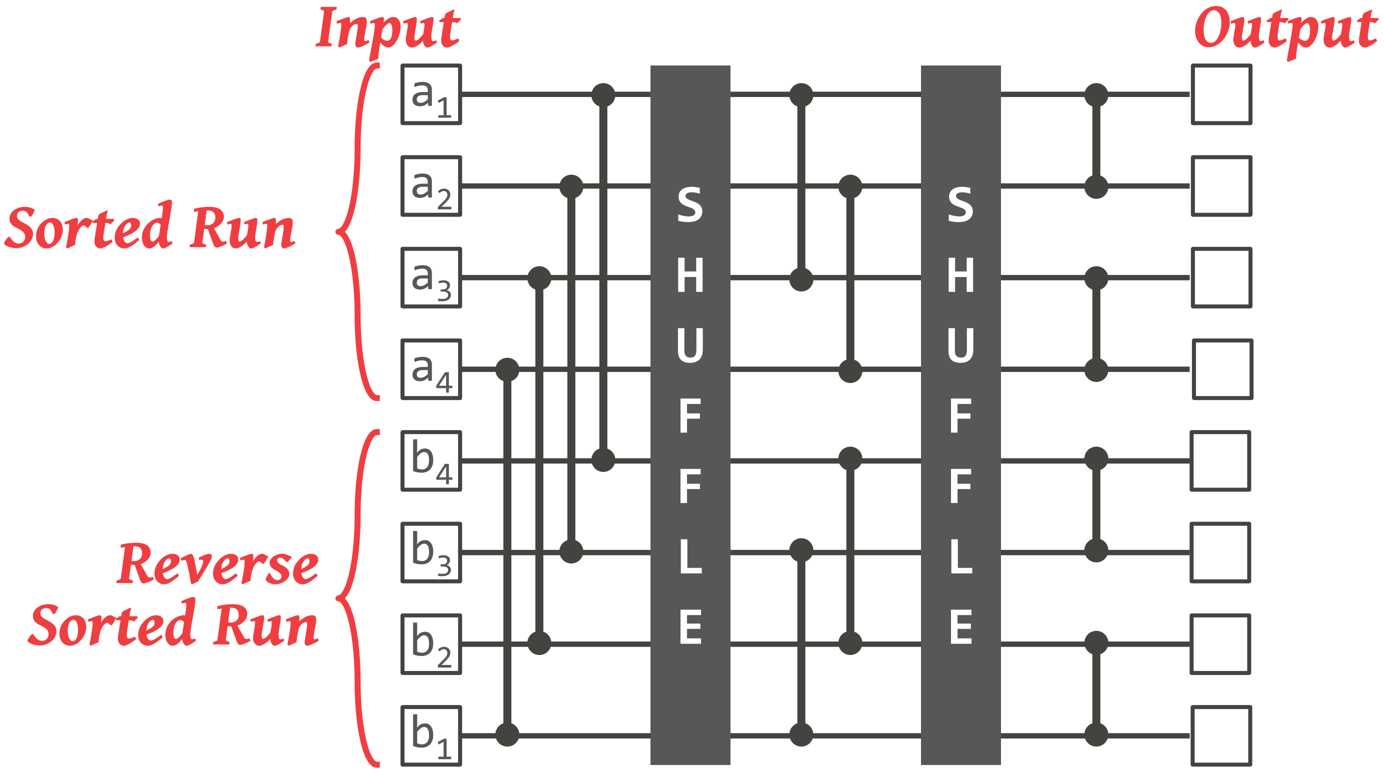
LEVEL #2 – BITONIC MERGE NETWORK



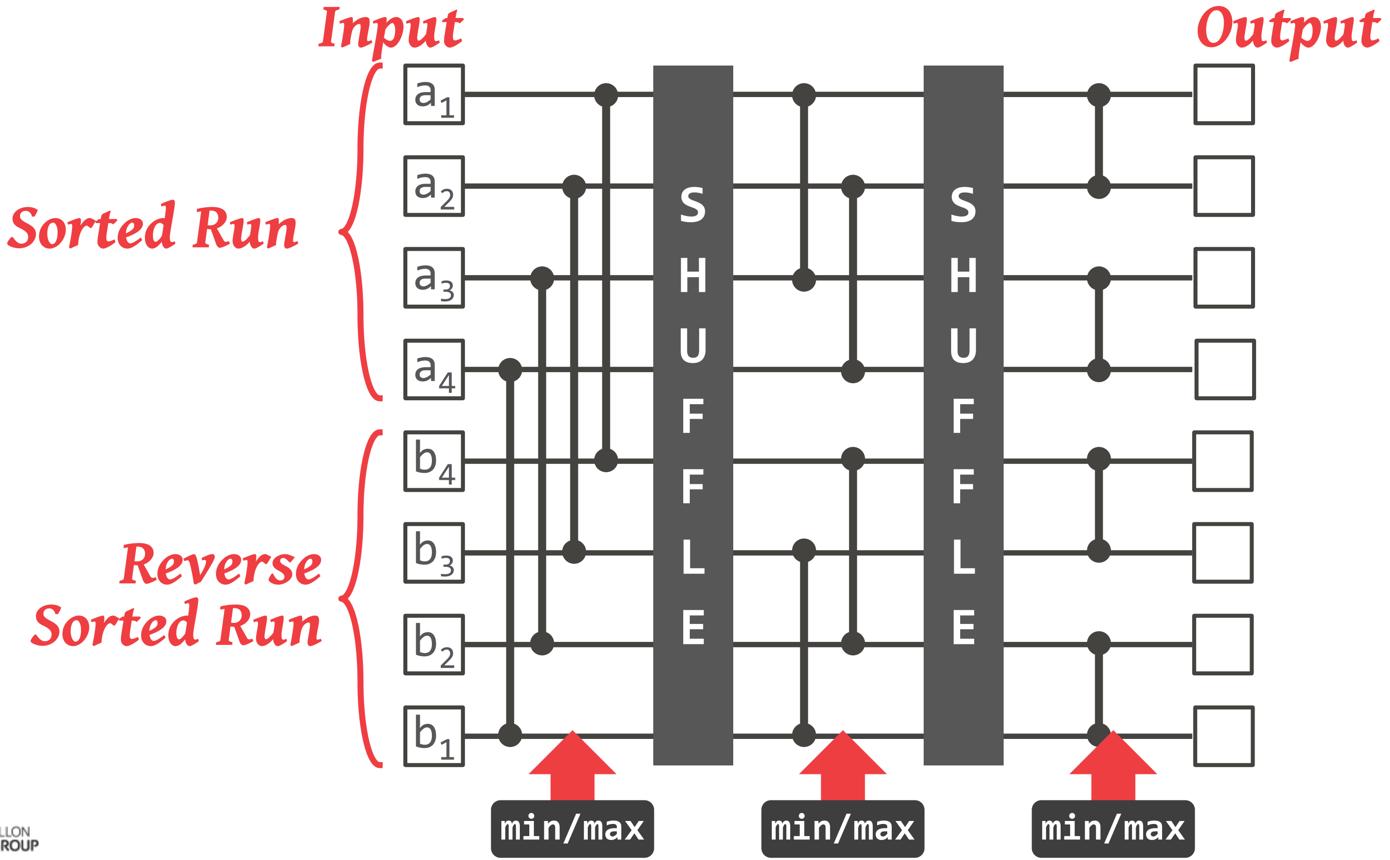
LEVEL #2 – BITONIC MERGE NETWORK



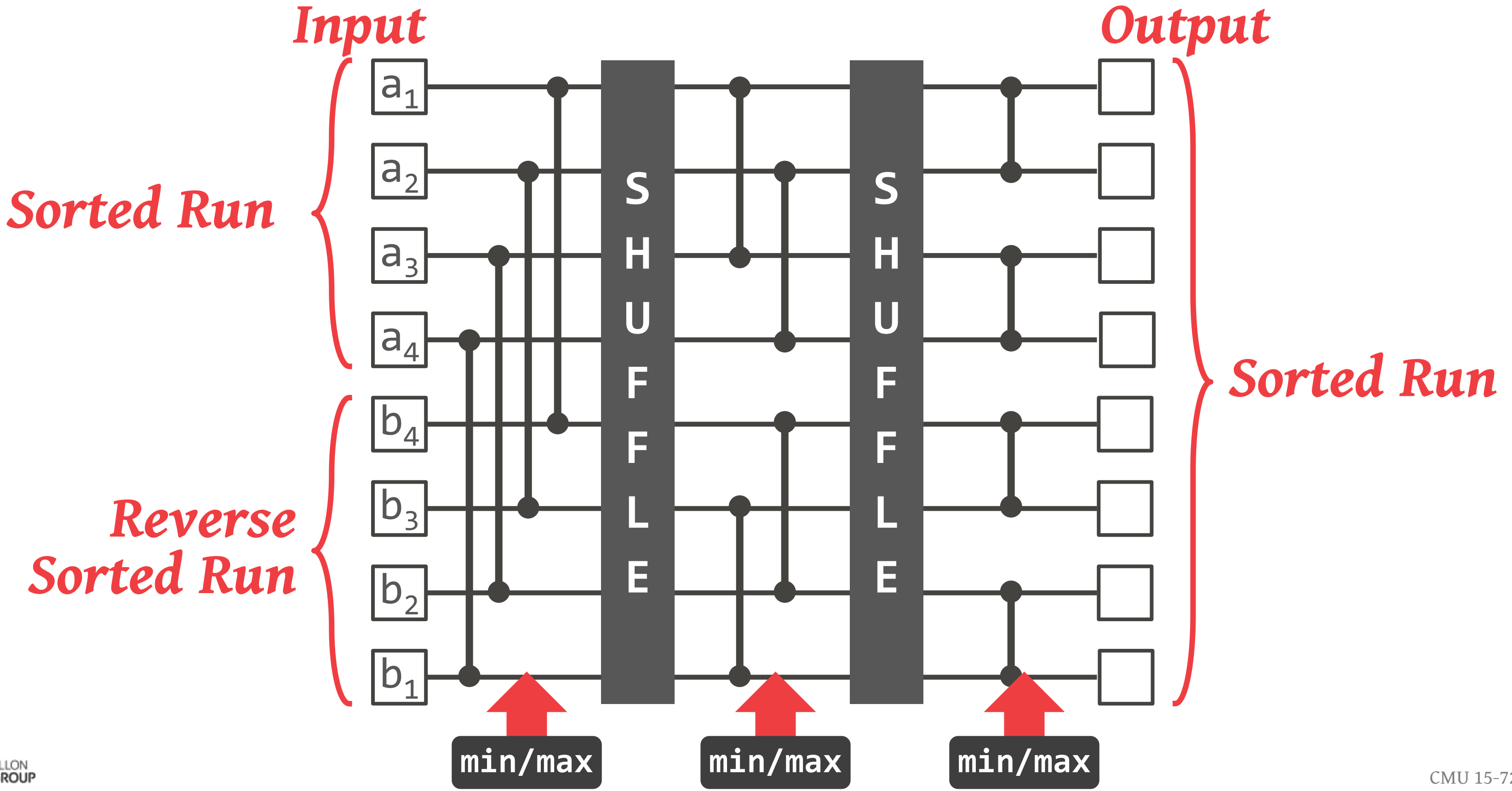
LEVEL #2 – BITONIC MERGE NETWORK



LEVEL #2 – BITONIC MERGE NETWORK



LEVEL #2 – BITONIC MERGE NETWORK



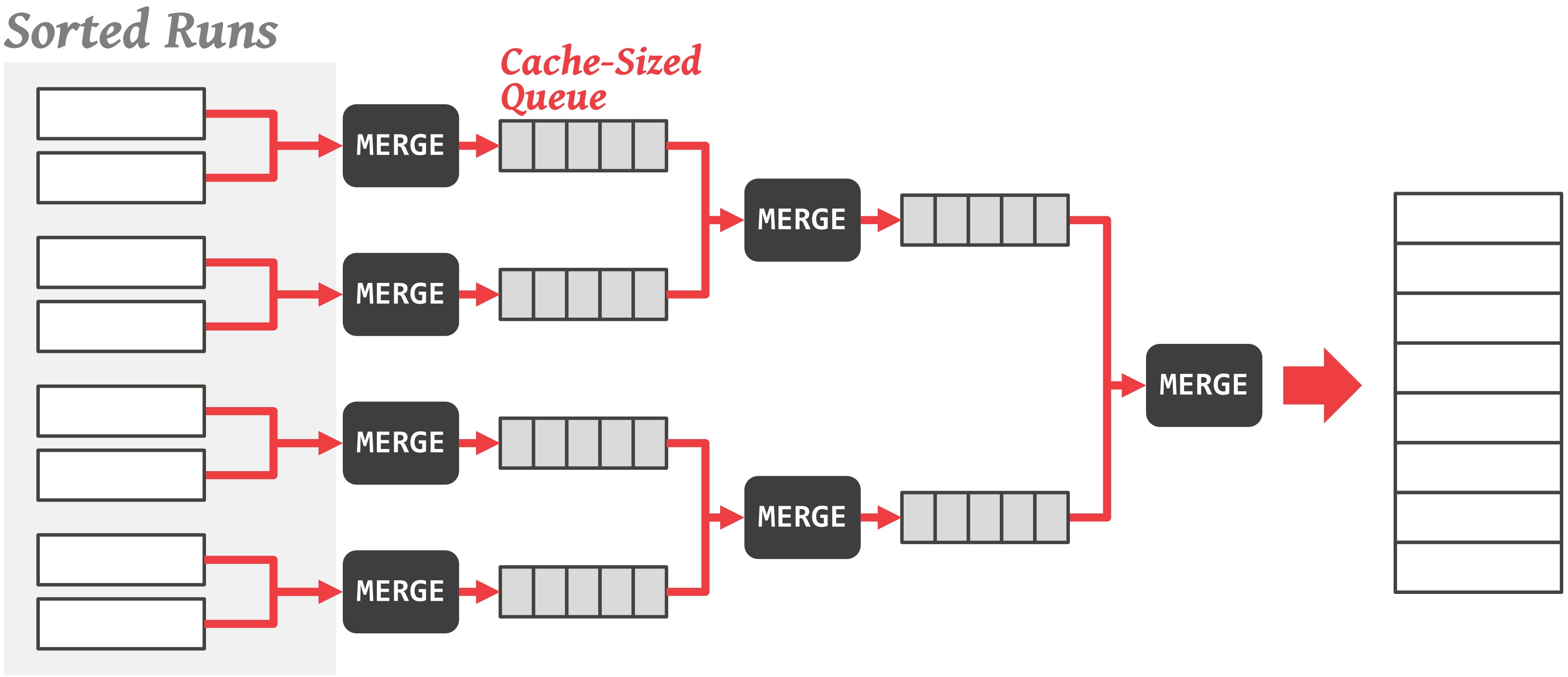
LEVEL #3 – MULTI-WAY MERGING

Use the Bitonic Merge Networks but split the process up into tasks.
→ Still one worker thread per core.
→ Link together tasks with a cache-sized FIFO queue.

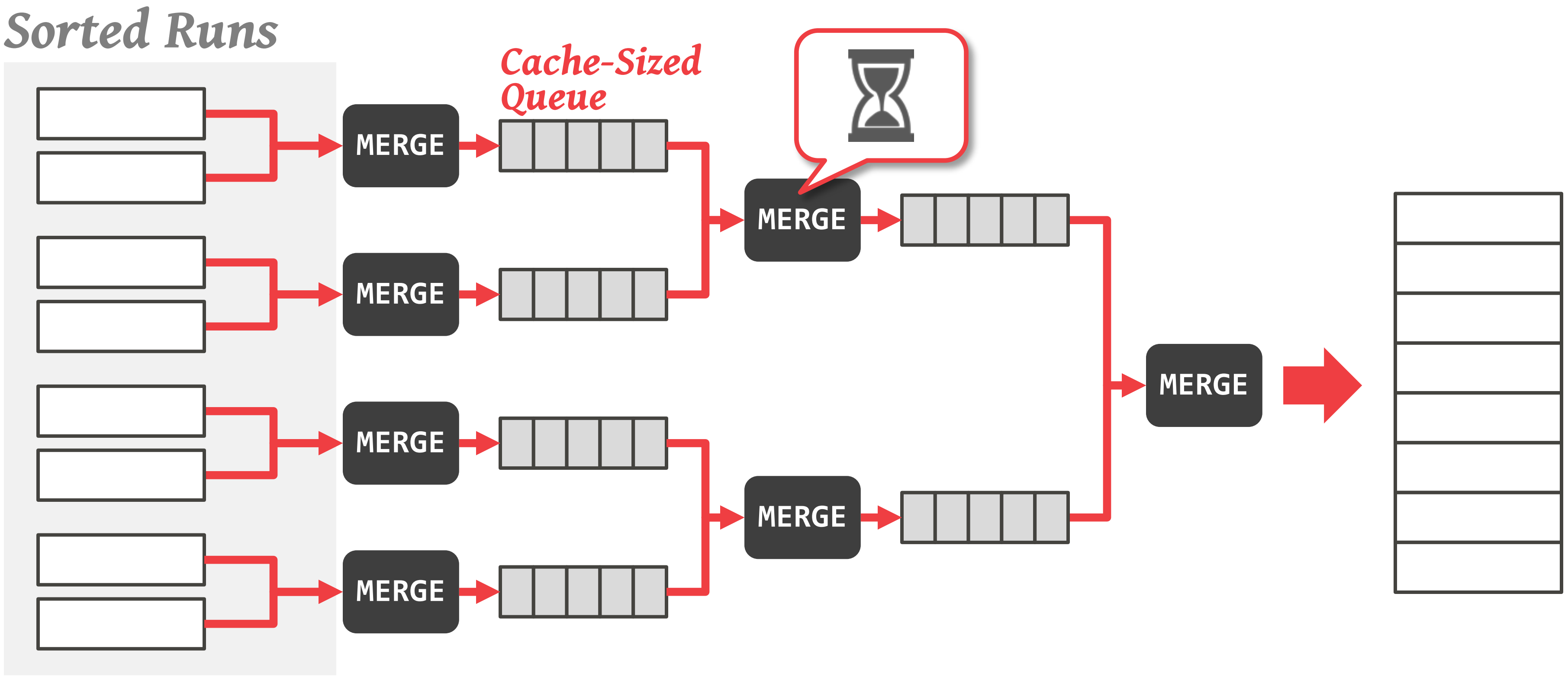
A task blocks when either its input queue is empty or its output queue is full.

Requires more CPU instructions, but brings bandwidth and compute into balance.

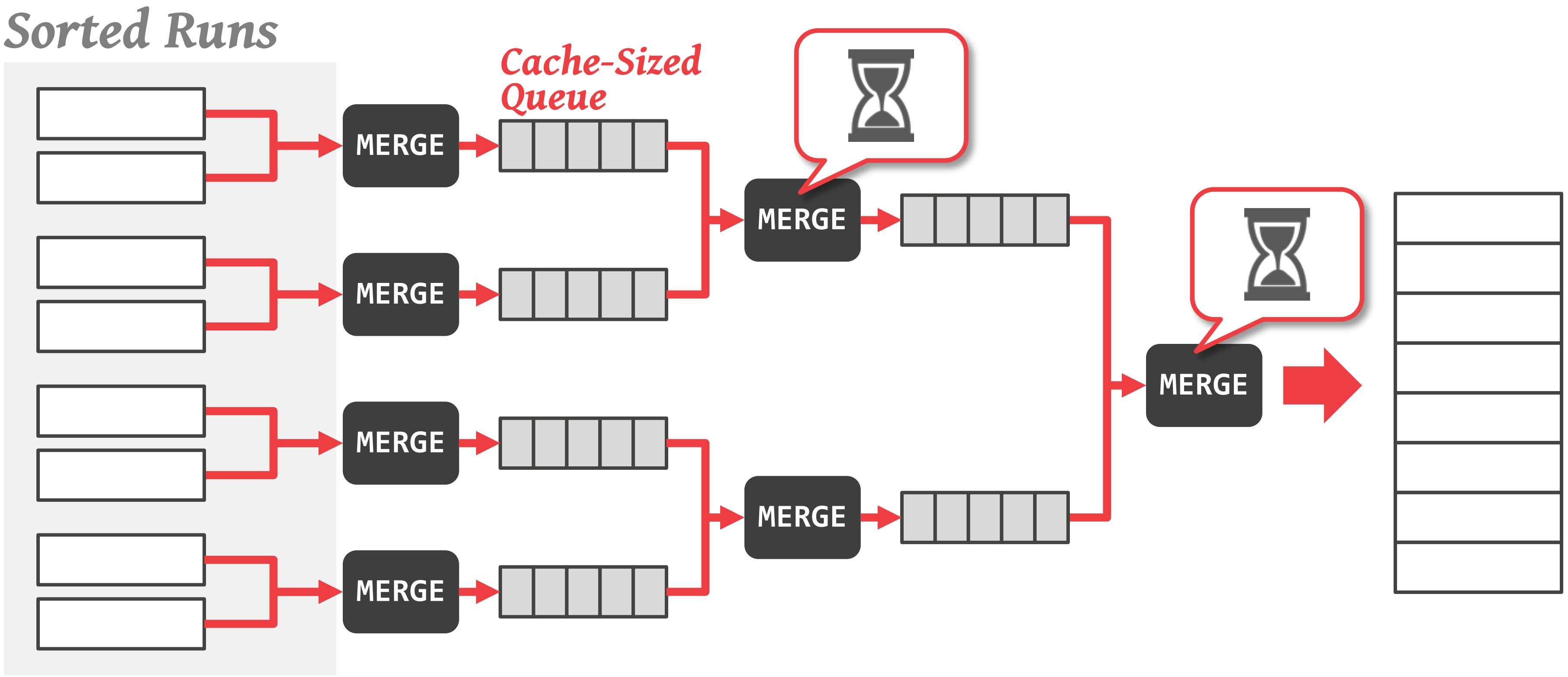
LEVEL #3 – MULTI-WAY MERGING



LEVEL #3 – MULTI-WAY MERGING



LEVEL #3 – MULTI-WAY MERGING



MERGE PHASE

Iterate through the outer table and inner table in lockstep and compare join keys.

May need to backtrack if there are duplicates.

Can be done in parallel at the different cores without synchronization if there are separate output buffers.

SORT-MERGE JOIN VARIANTS

Multi-Way Sort-Merge (**M-WAY**)

Multi-Pass Sort-Merge (**M-PASS**)

Massively Parallel Sort-Merge (**MPSM**)

MULTI-WAY SORT-MERGE

Outer Table

→ Each core sorts in parallel on local data (levels #1/#2).

→ Redistribute sorted runs across cores using the **multi-way merge** (level #3).

Inner Table

→ Same as outer table.

Merge phase is between matching pairs of chunks of outer/inner tables at each core.

MULTI-CORE, MAIN-MEMORY JOINS: SORT VS. HASH REVISITED
VLDB 2013

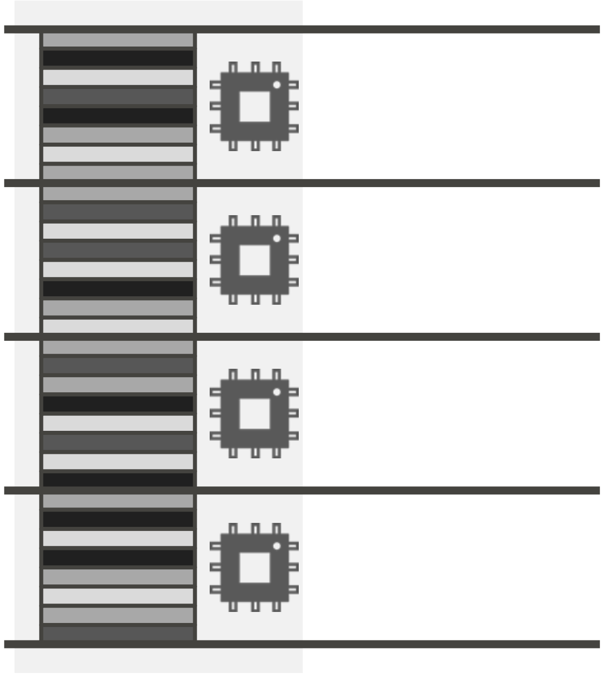
MULTI-WAY SORT-MERGE

MULTI-WAY SORT-MERGE

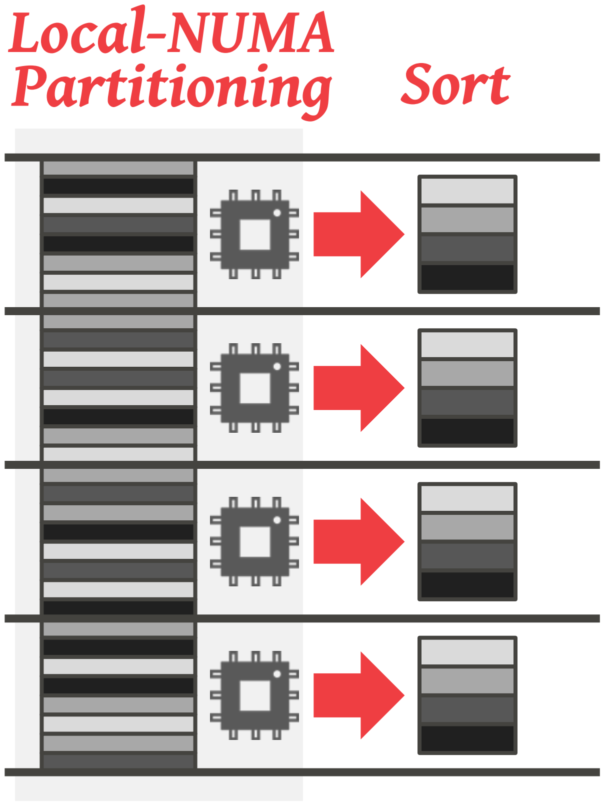
Local-NUMA Partitioning

MULTI-WAY SORT-MERGE

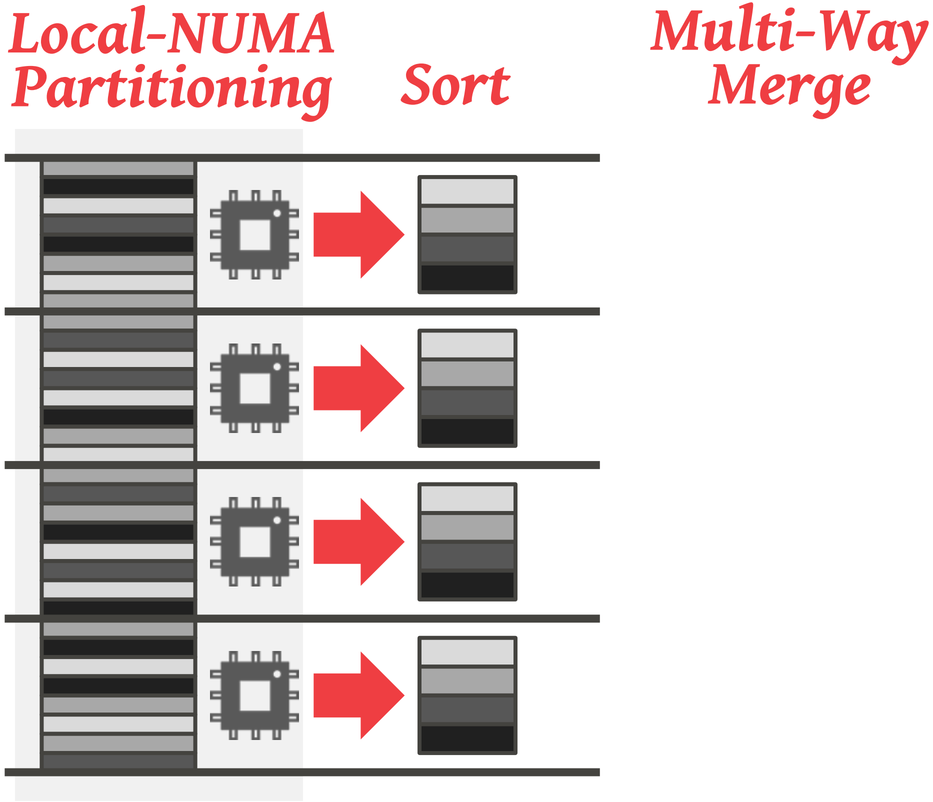
Local-NUMA Partitioning



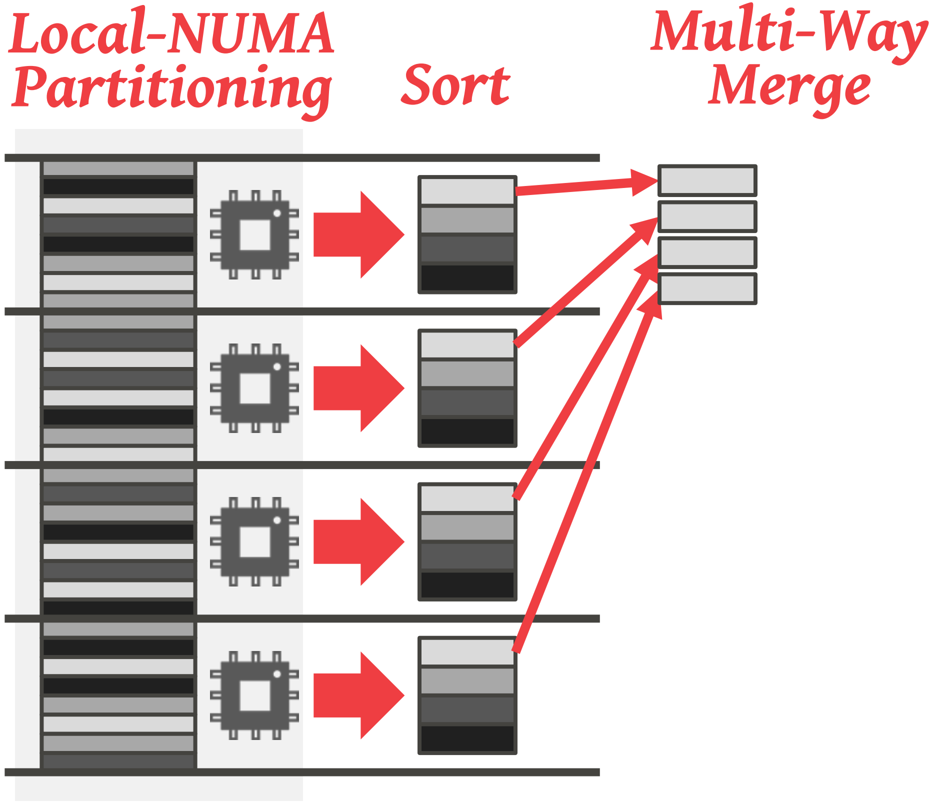
MULTI-WAY SORT-MERGE



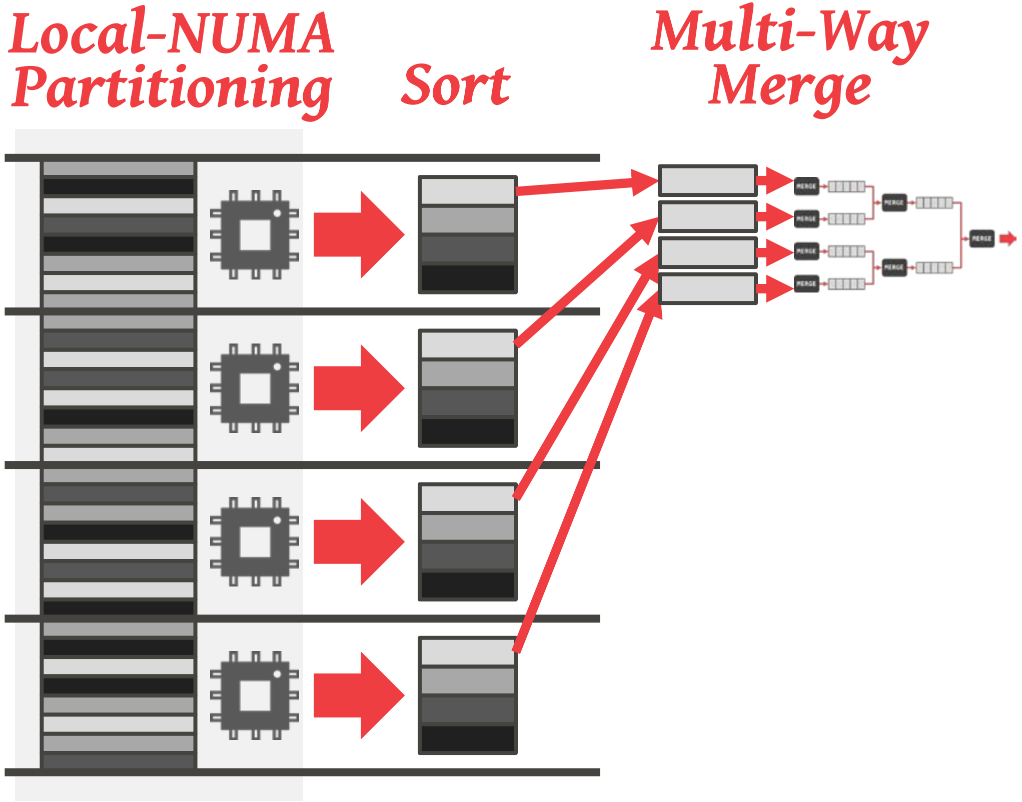
MULTI-WAY SORT-MERGE



MULTI-WAY SORT-MERGE

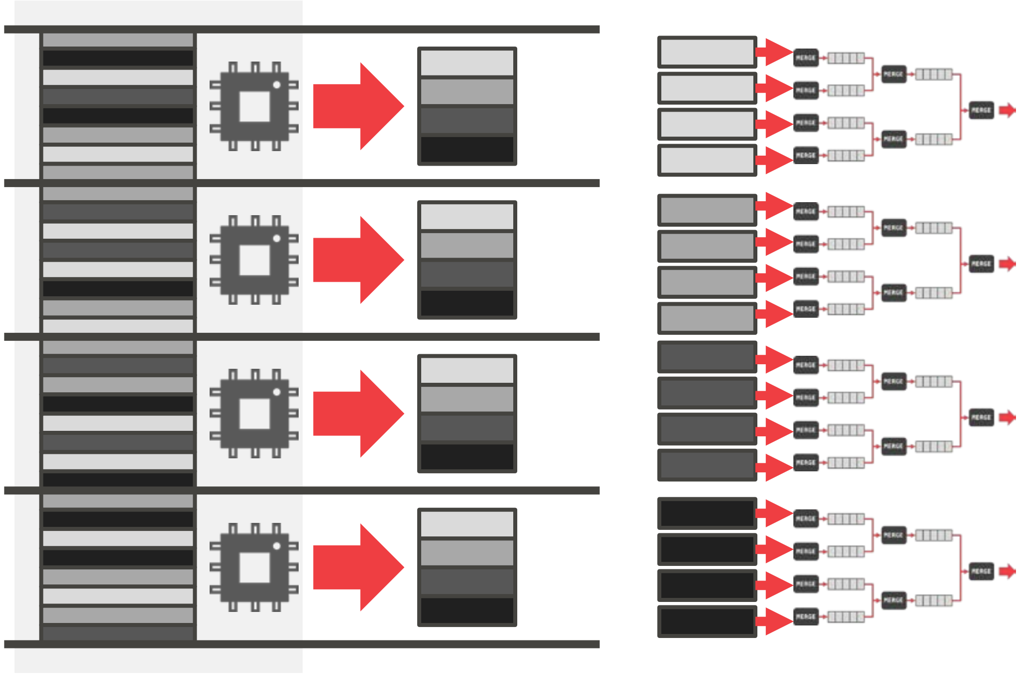


MULTI-WAY SORT-MERGE

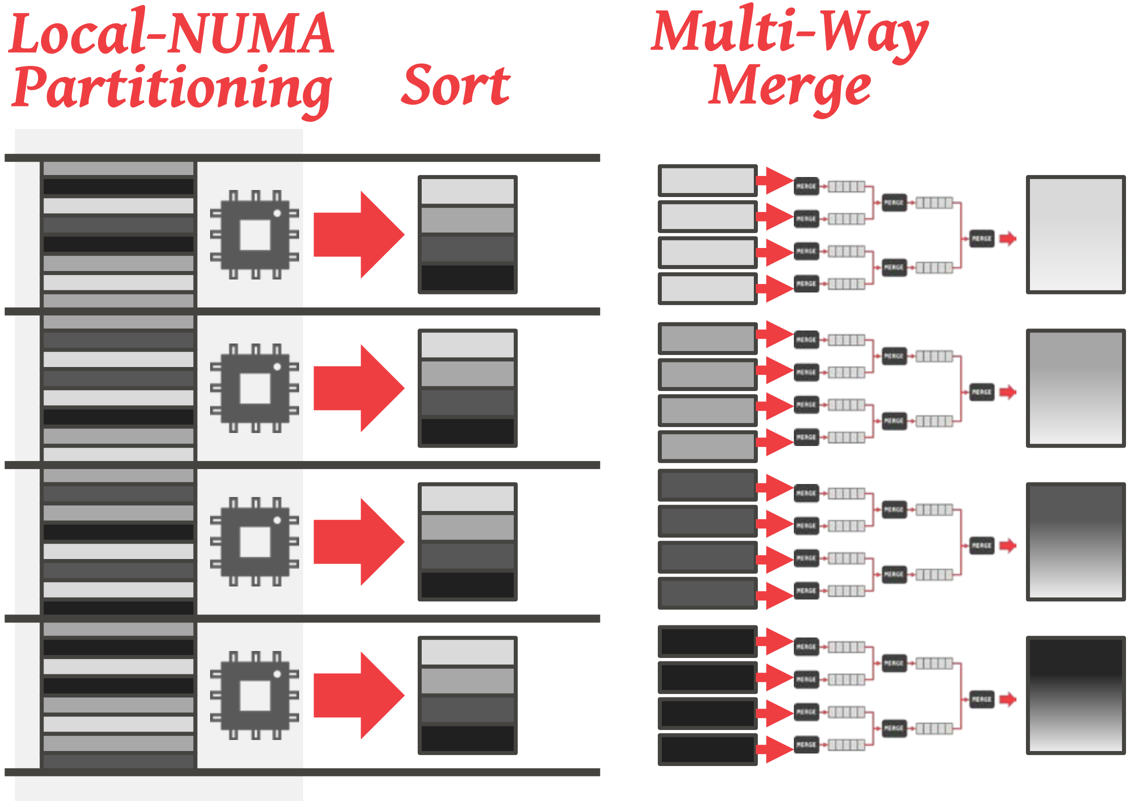


MULTI-WAY SORT-MERGE

Local-NUMA Partitioning ***Sort*** ***Multi-Way Merge***



MULTI-WAY SORT-MERGE

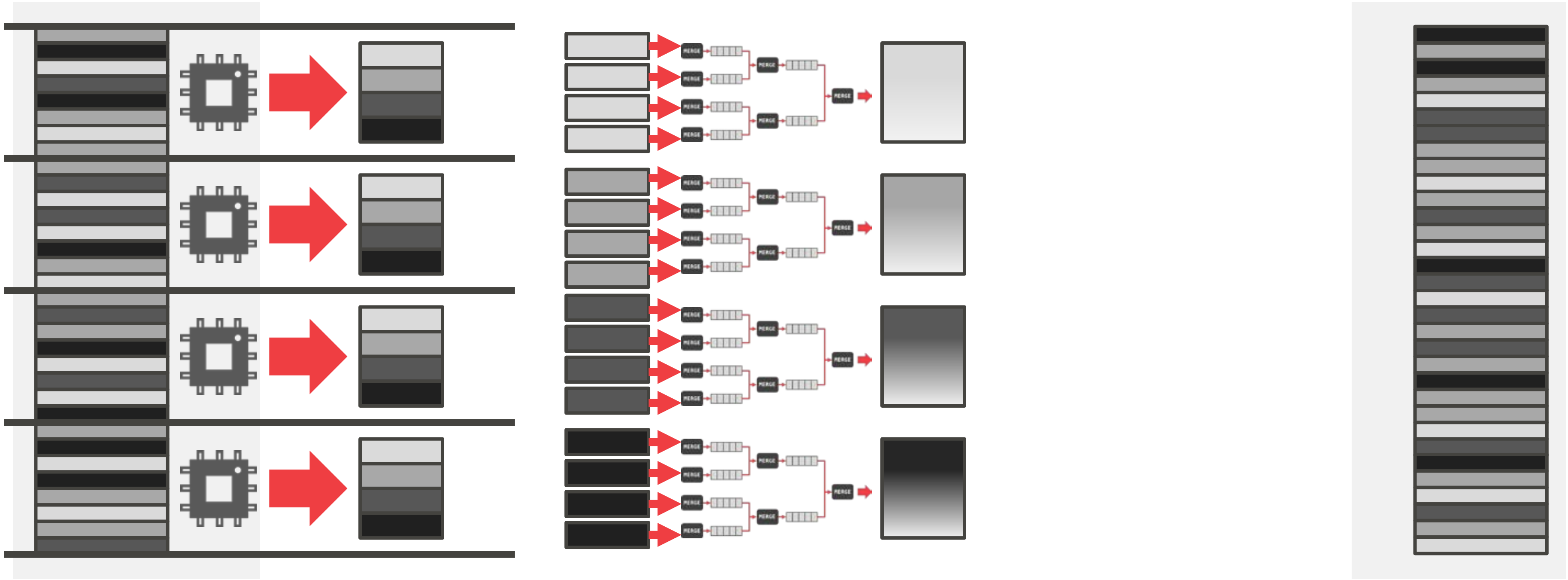


MULTI-WAY SORT-MERGE

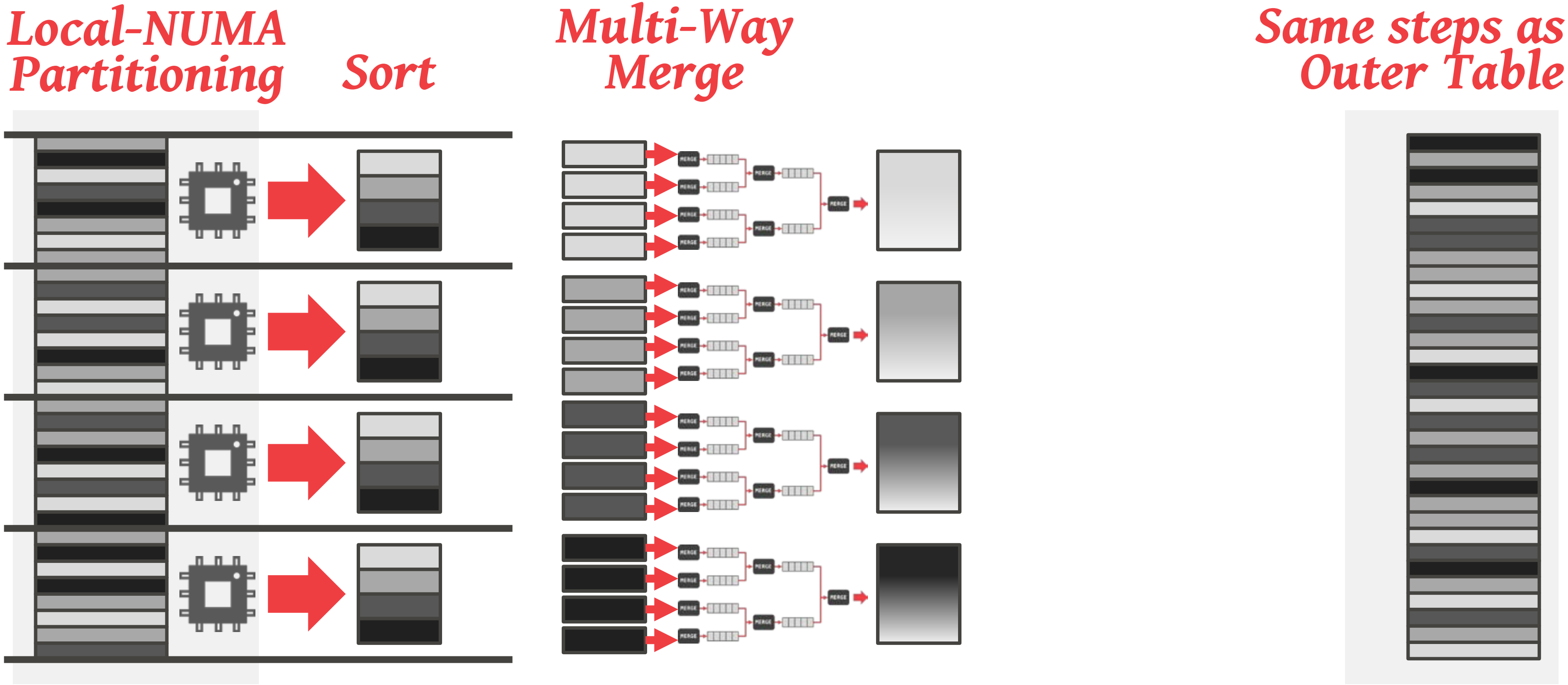
Local-NUMA Partitioning

Sort

Multi-Way Merge



MULTI-WAY SORT-MERGE



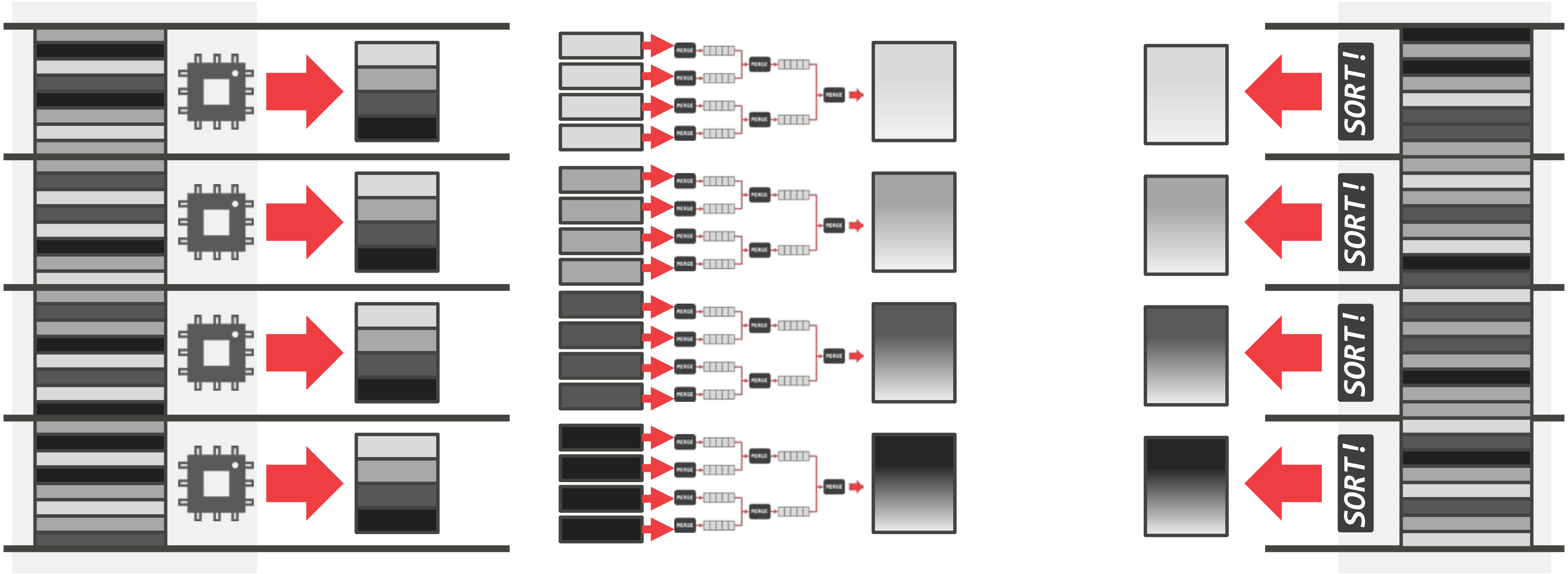
MULTI-WAY SORT-MERGE

Local-NUMA Partitioning

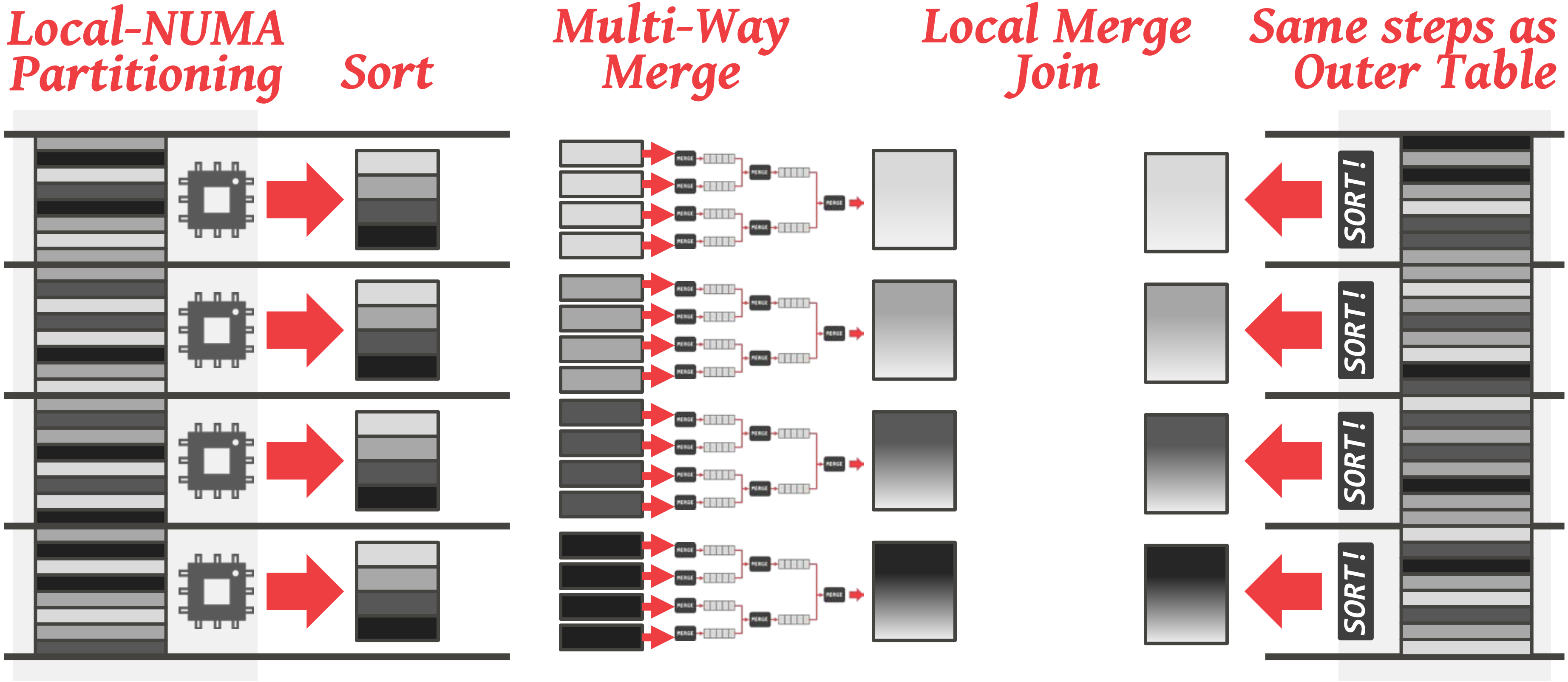
Sort

Multi-Way Merge

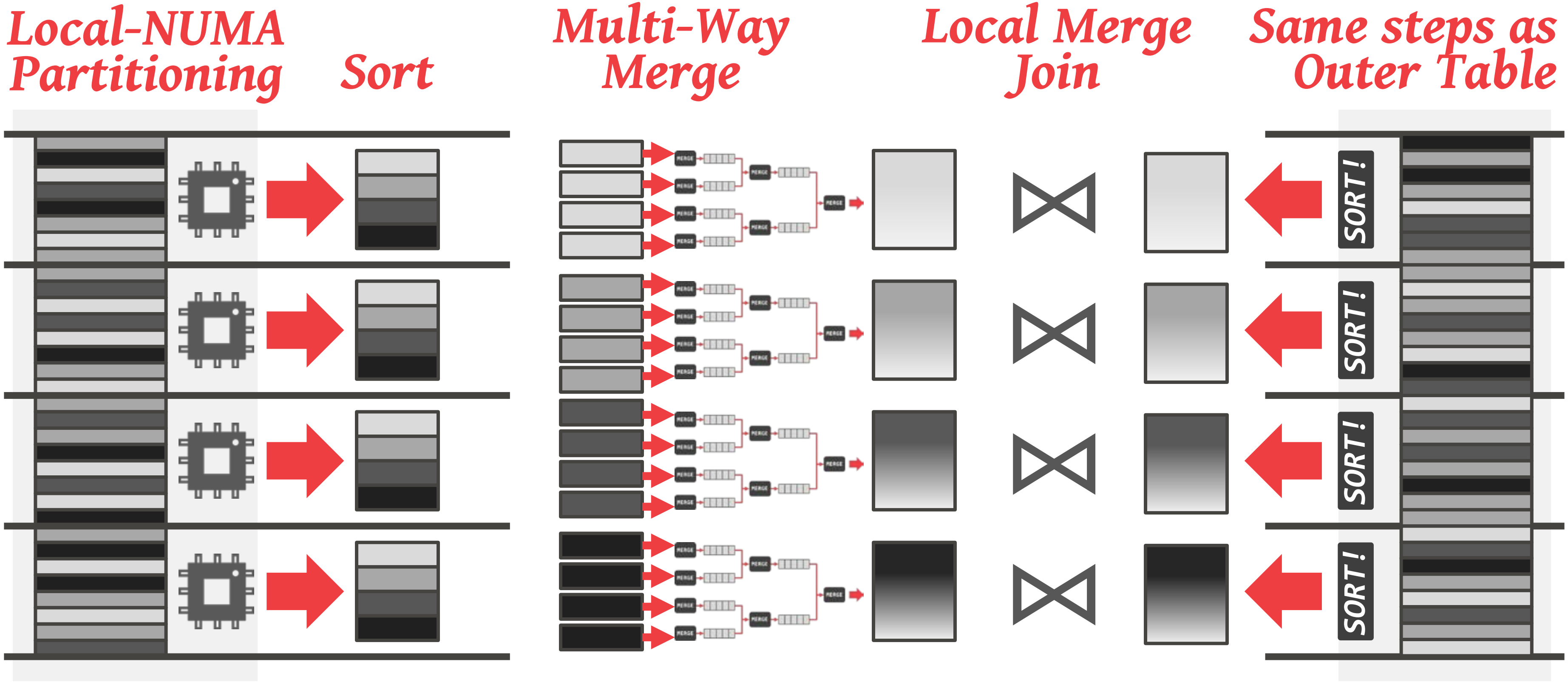
Same steps as Outer Table



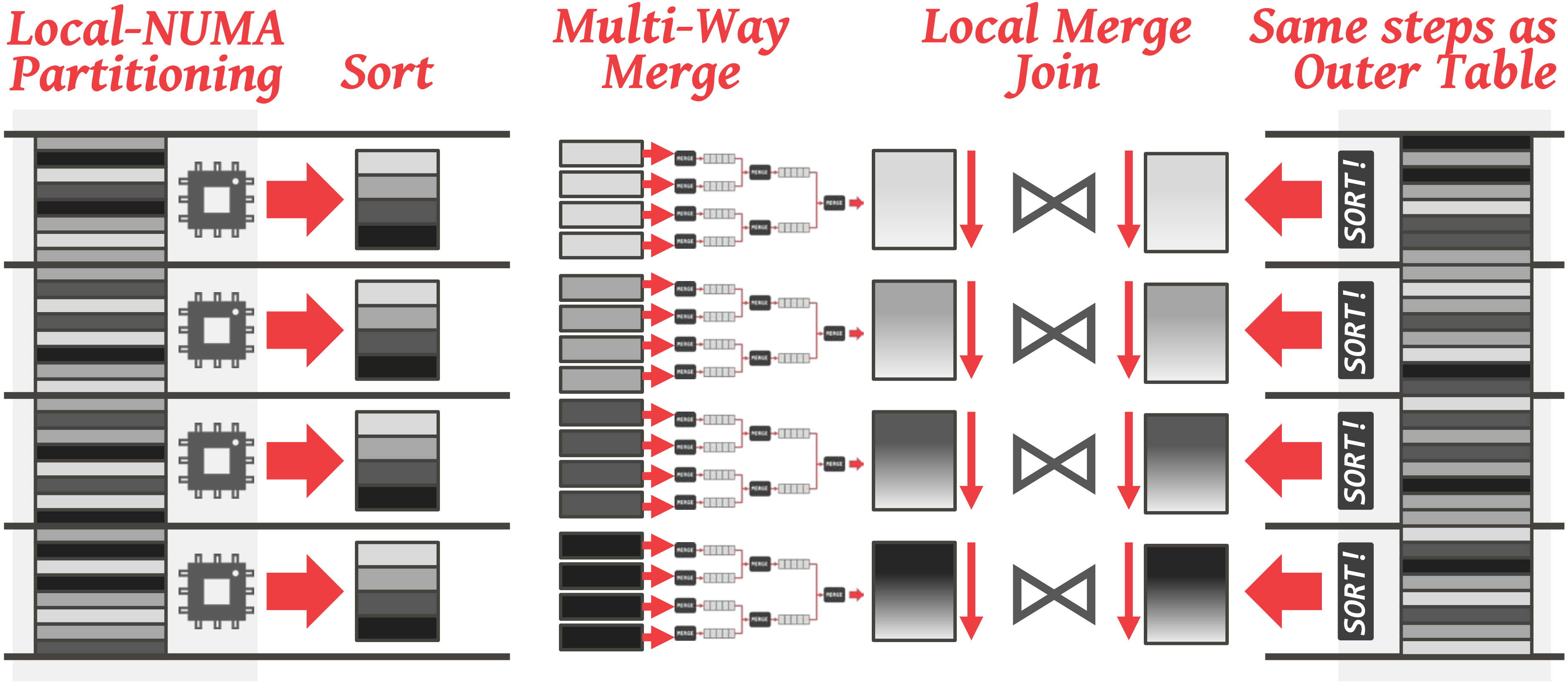
MULTI-WAY SORT-MERGE



MULTI-WAY SORT-MERGE



MULTI-WAY SORT-MERGE



MULTI-PASS SORT-MERGE

Outer Table

→ Same level #1/#2 sorting as M-WAY.
→ But instead of redistributing, it uses a **multi-pass naïve merge** on sorted runs.

Inner Table

→ Same as outer table.

Merge phase is between matching pairs of chunks of outer table and inner table.

MULTI-CORE, MAIN-MEMORY JOINS: SORT VS. HASH REVISITED
VLDB 2013

MASSIVELY PARALLEL SORT-MERGE

Outer Table

→ Range-partition outer table and redistribute to cores.
→ Each core sorts in parallel on their partitions.

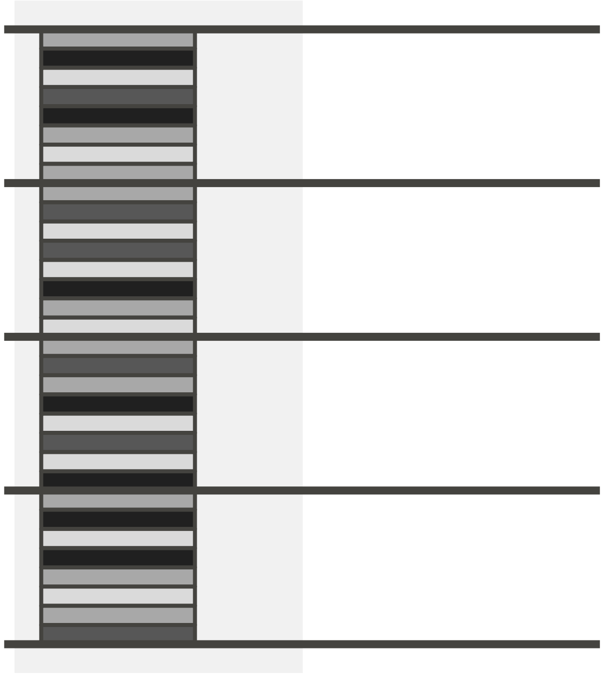
Inner Table

→ Not redistributed like outer table.
→ Each core sorts its local data.

Merge phase is between entire sorted run of outer table and a segment of inner table.

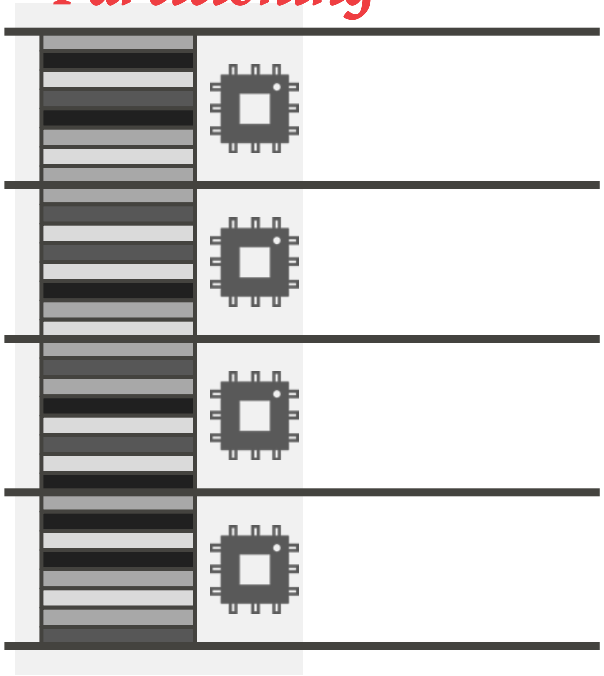
MASSIVELY PARALLEL SORT-MERGE JOINS IN MAIN MEMORY MULTI-CORE DATABASE SYSTEMS
VLDB 2012

MASSIVELY PARALLEL SORT-MERGE



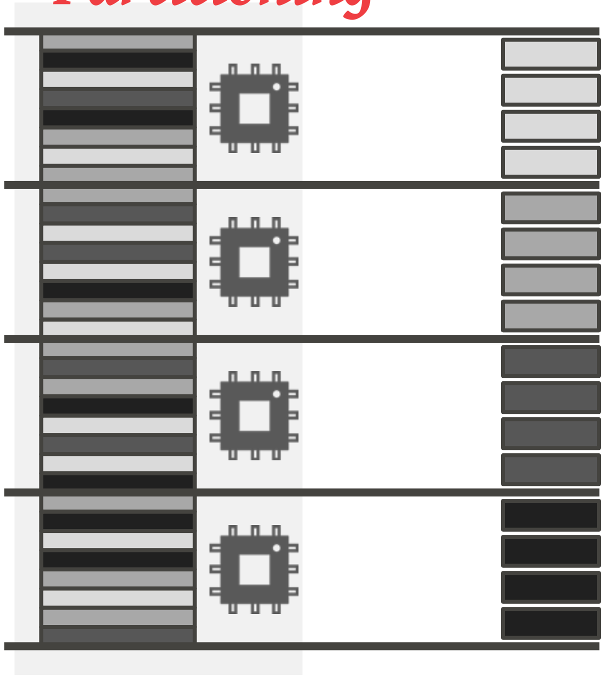
MASSIVELY PARALLEL SORT-MERGE

Cross-NUMA Partitioning

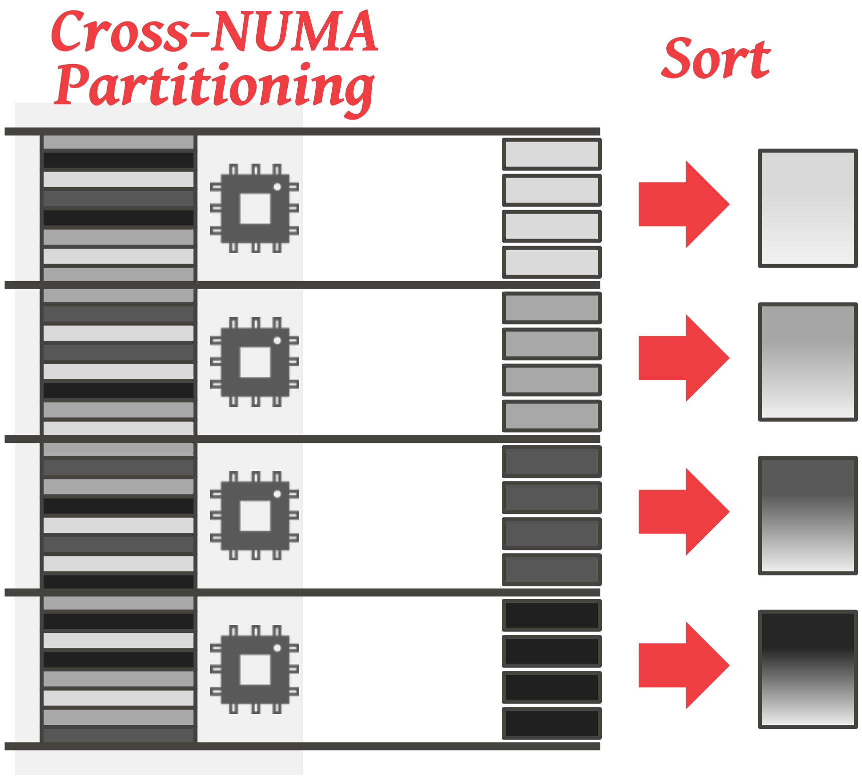


MASSIVELY PARALLEL SORT-MERGE

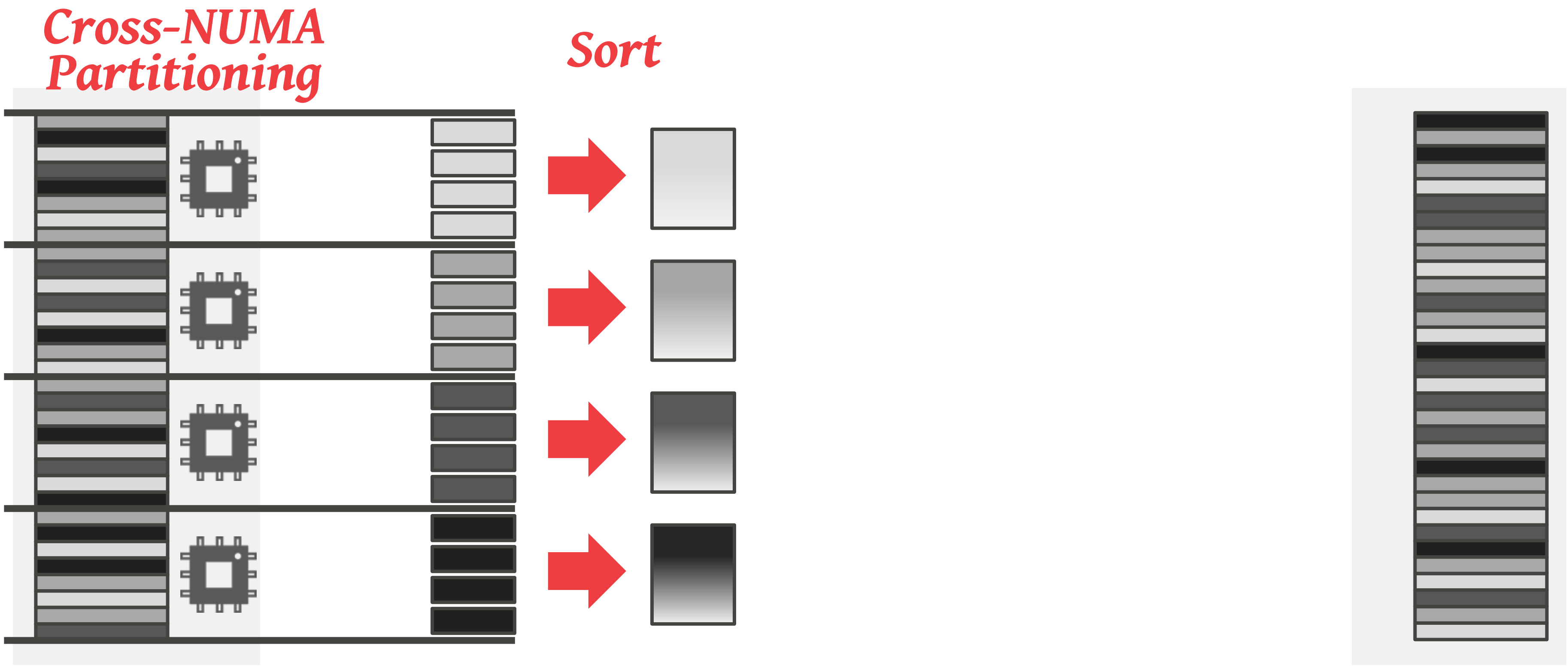
Cross-NUMA Partitioning



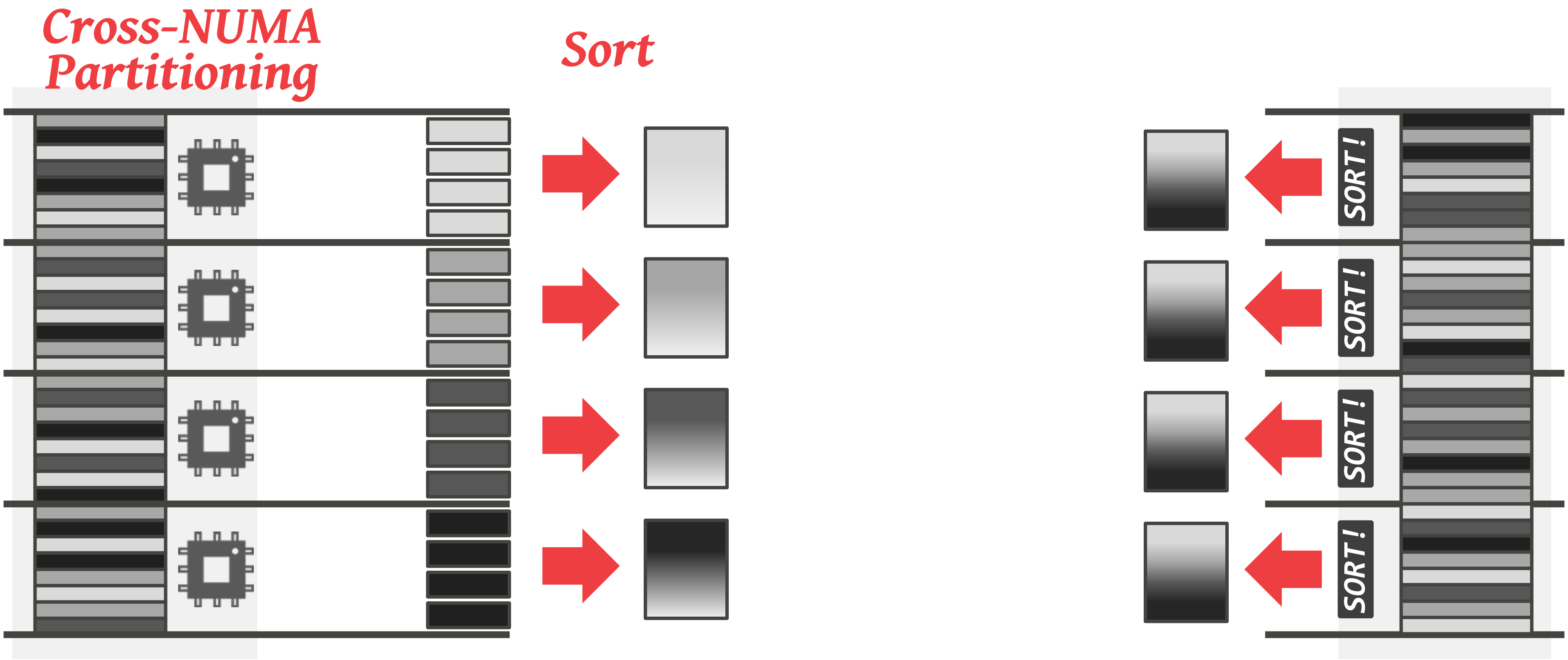
MASSIVELY PARALLEL SORT-MERGE



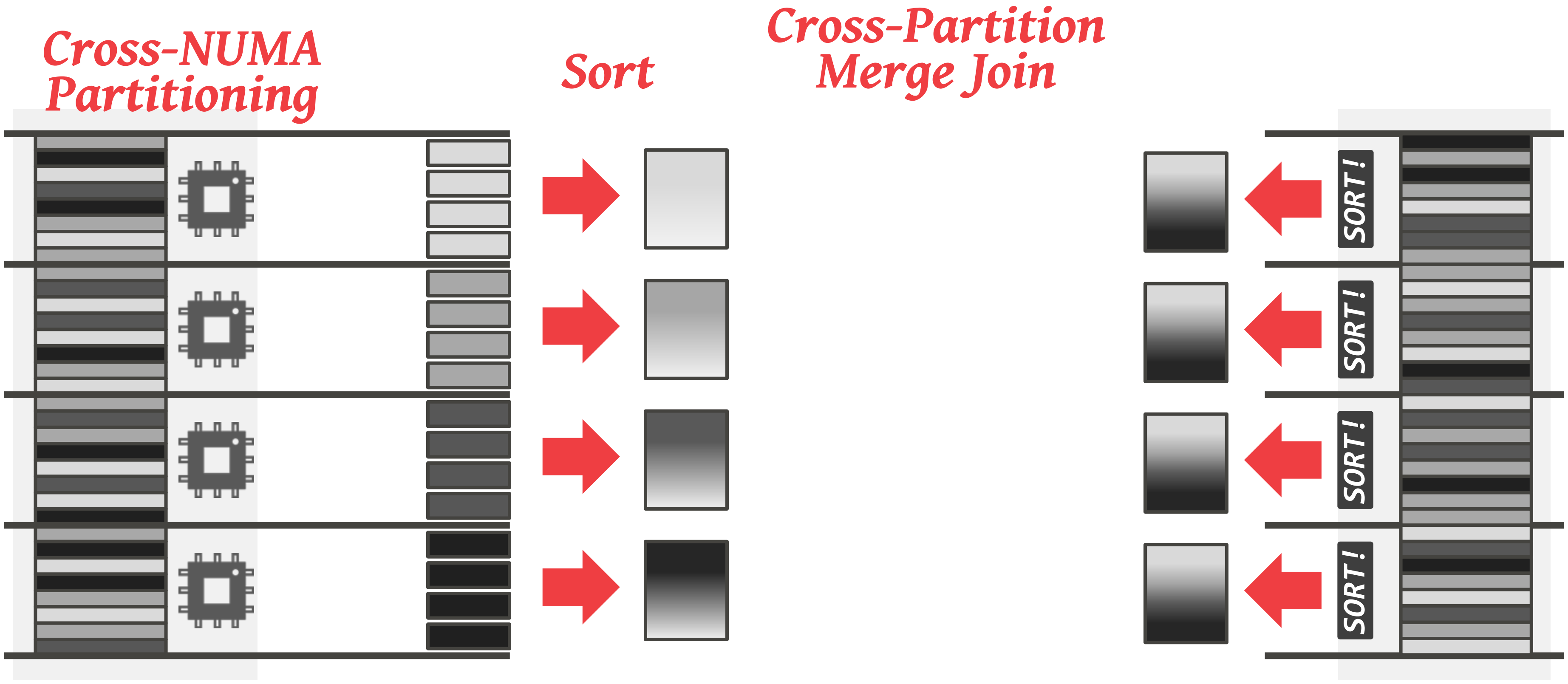
MASSIVELY PARALLEL SORT-MERGE



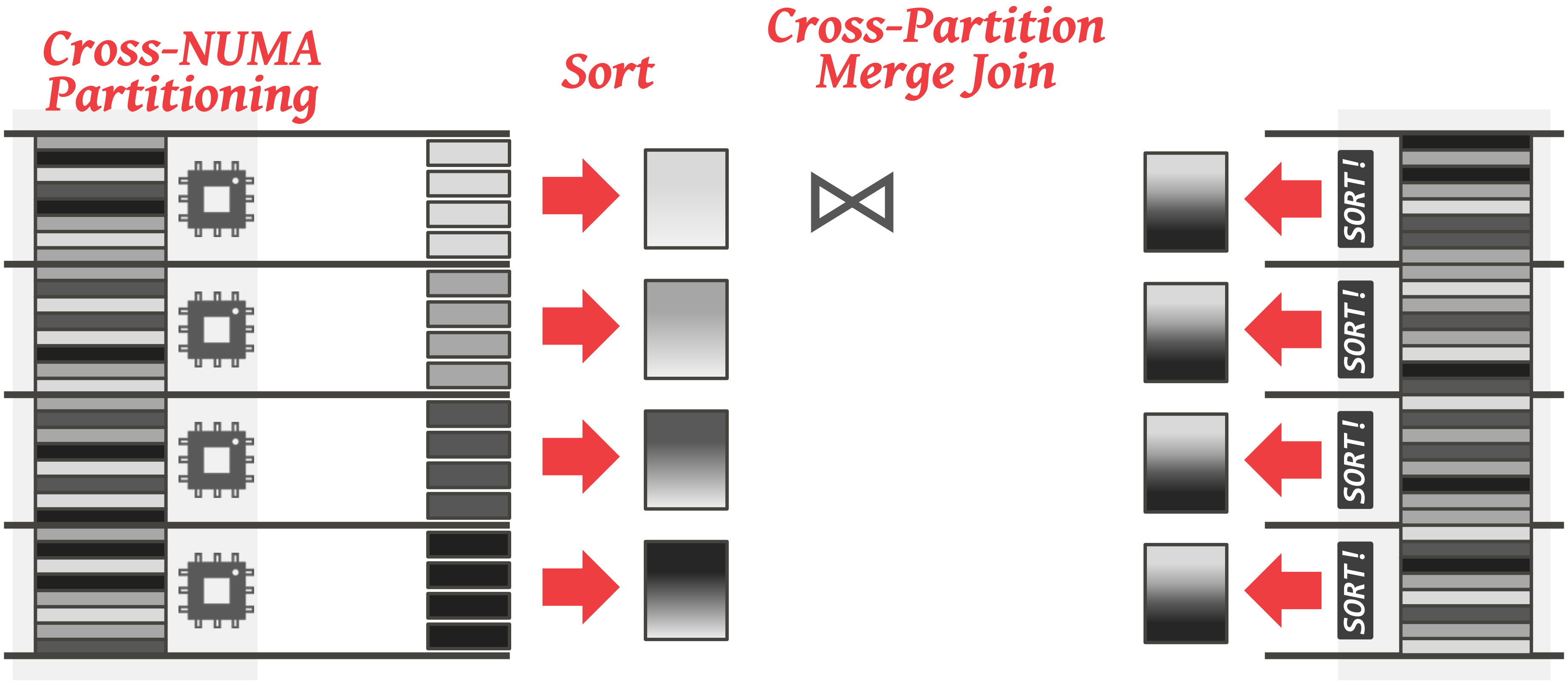
MASSIVELY PARALLEL SORT-MERGE



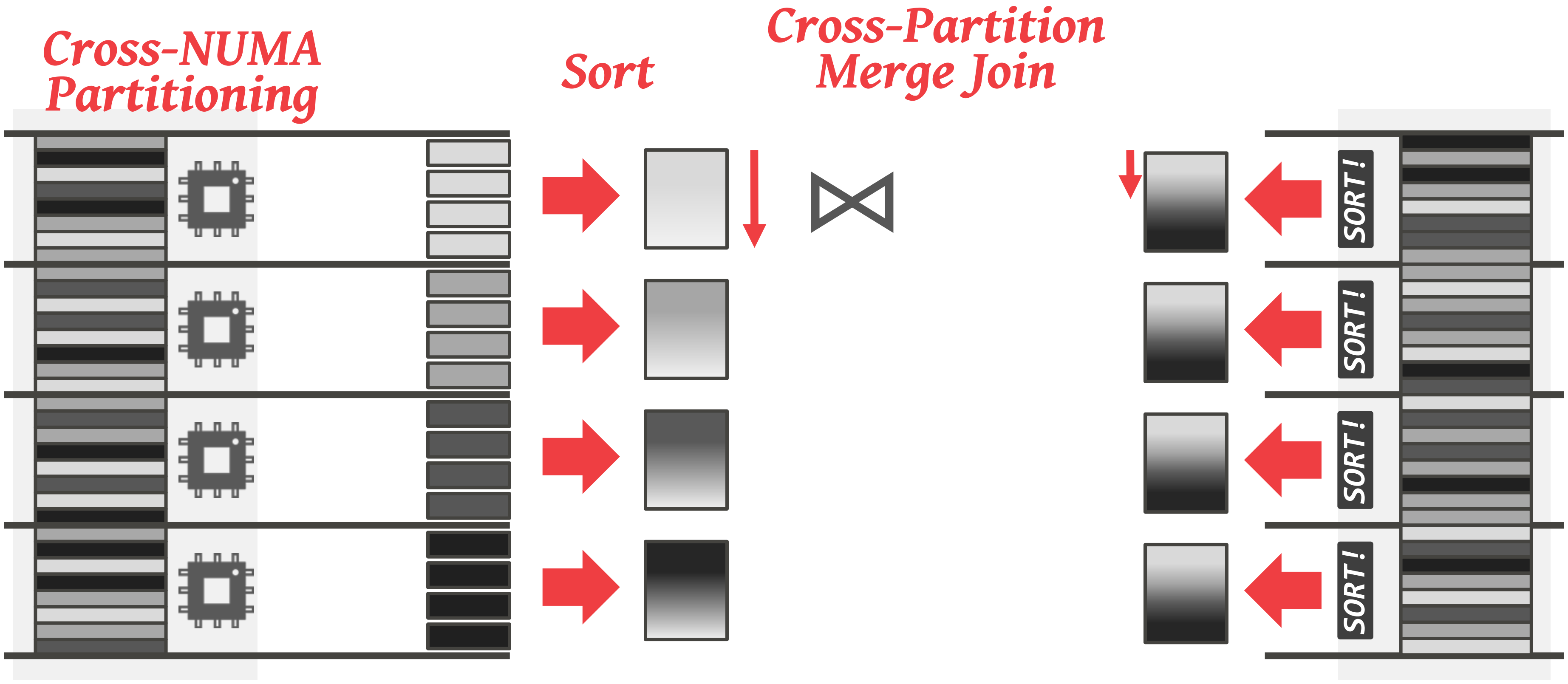
MASSIVELY PARALLEL SORT-MERGE



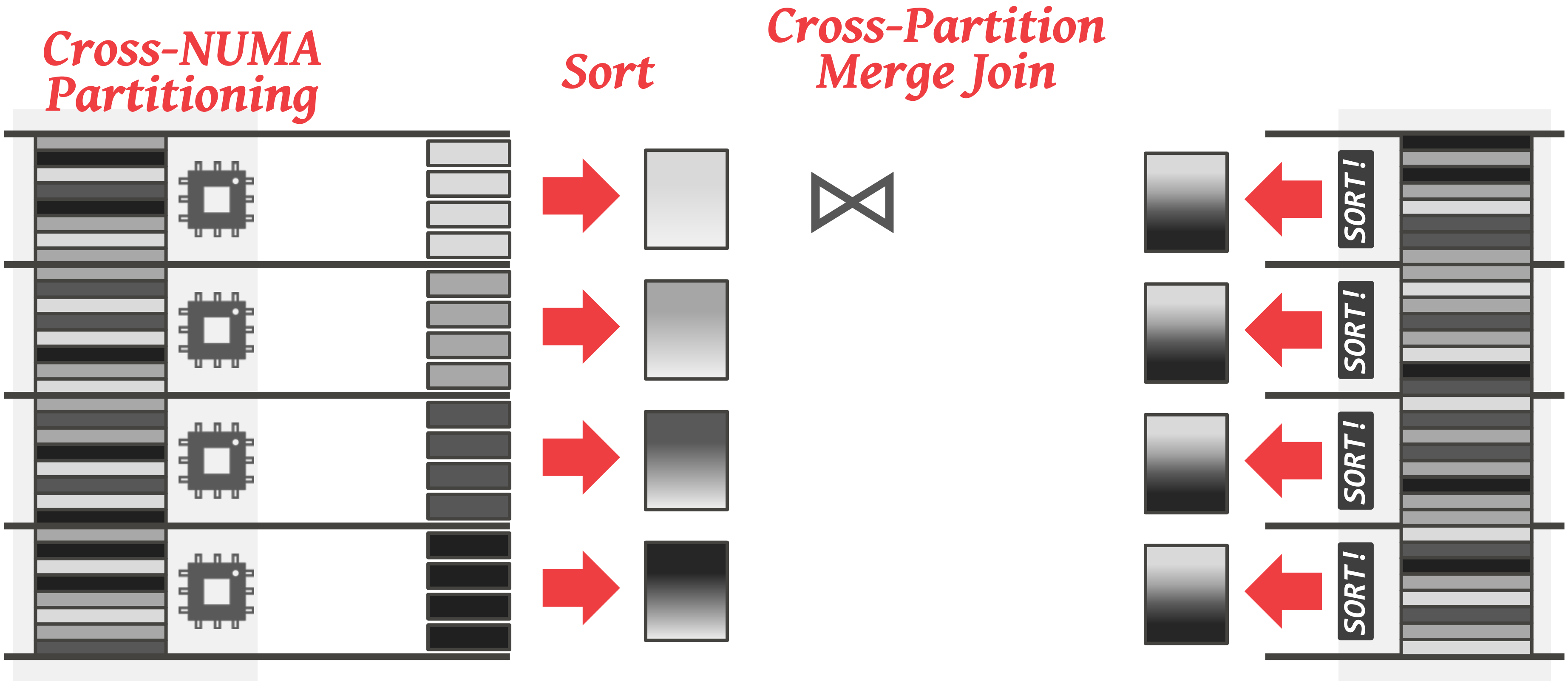
MASSIVELY PARALLEL SORT-MERGE



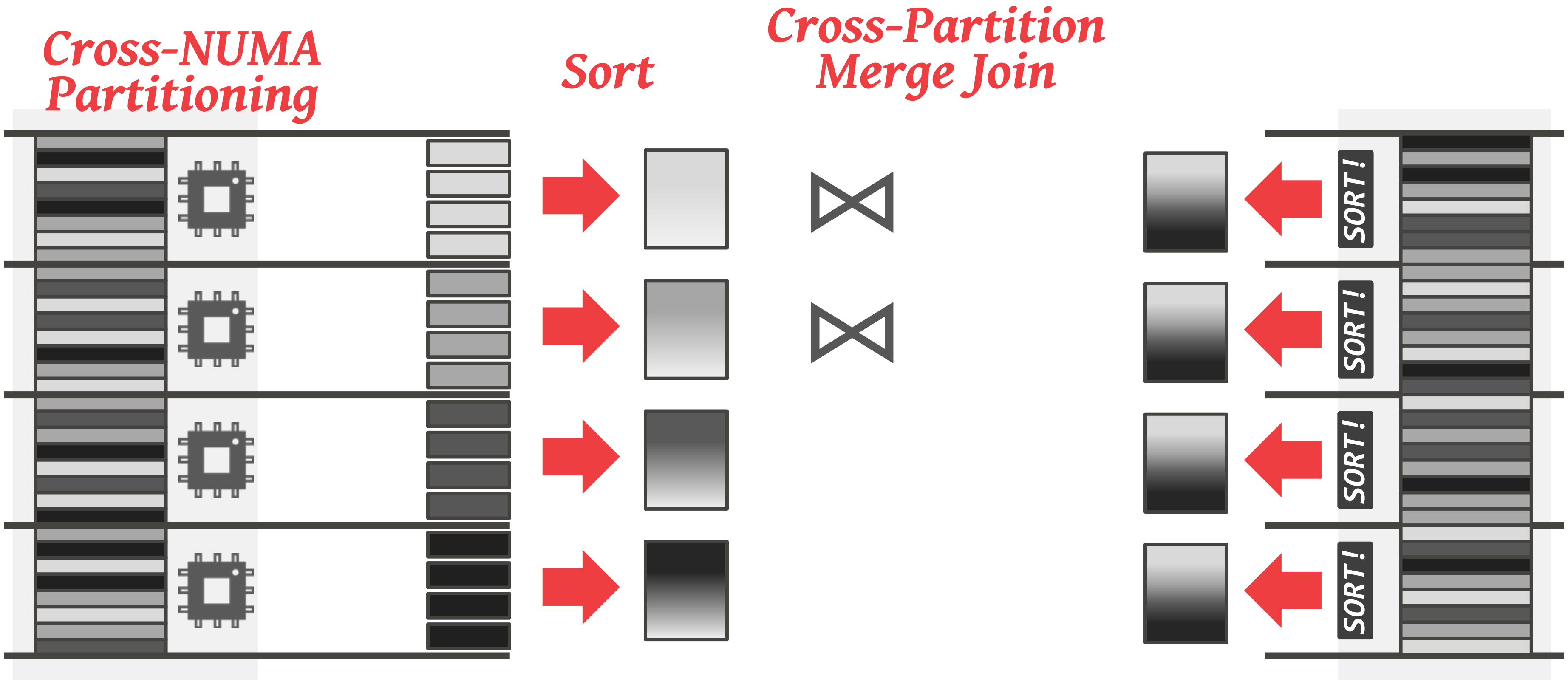
MASSIVELY PARALLEL SORT-MERGE



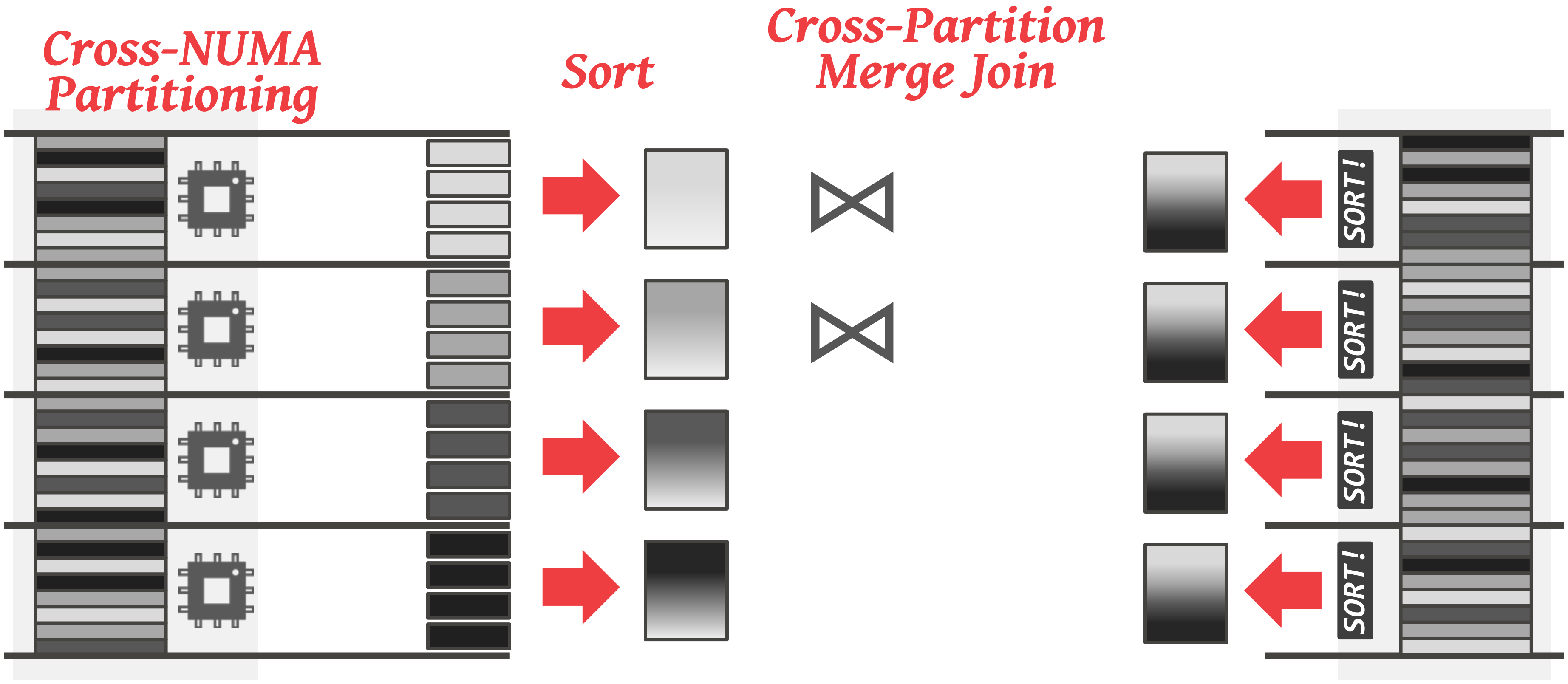
MASSIVELY PARALLEL SORT-MERGE



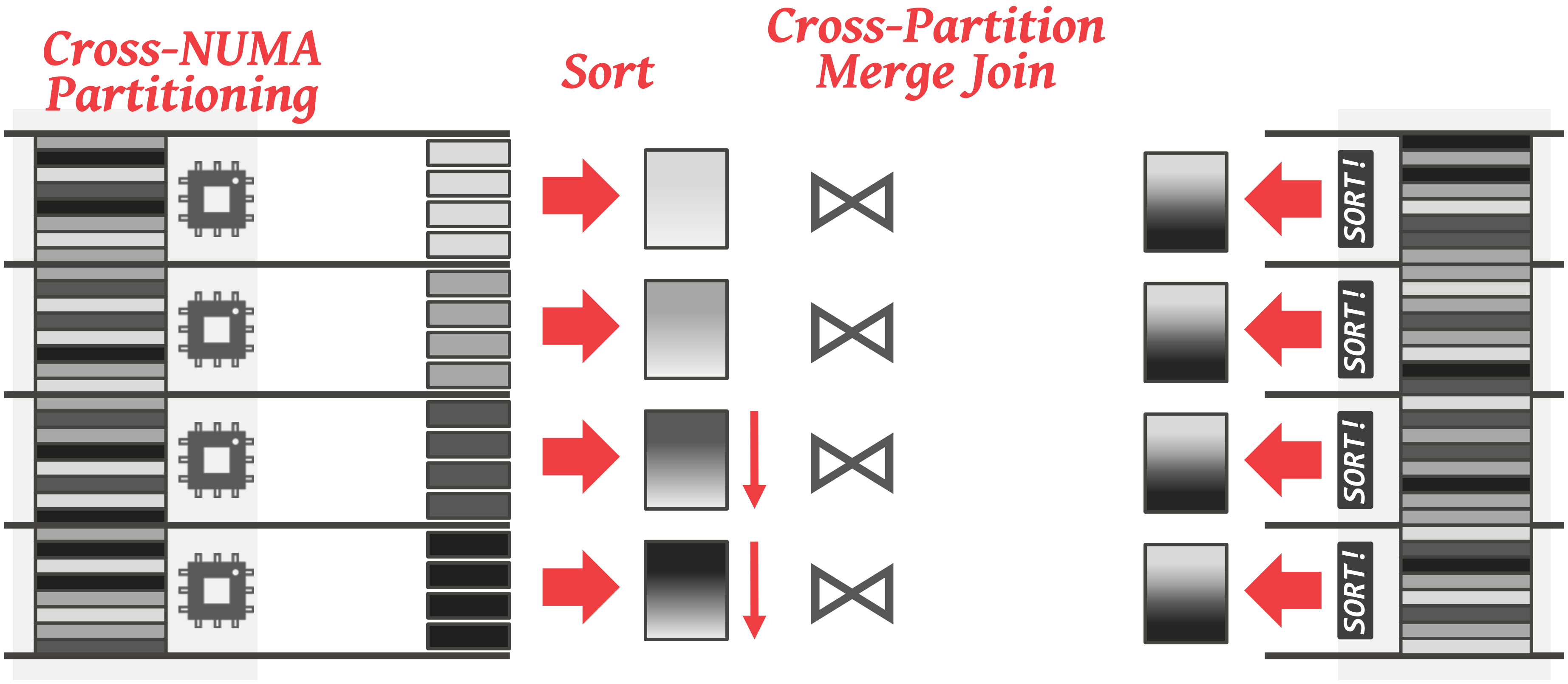
MASSIVELY PARALLEL SORT-MERGE



MASSIVELY PARALLEL SORT-MERGE



MASSIVELY PARALLEL SORT-MERGE



HYPER's RULES FOR PARALLELIZATION

Rule #1: No random writes to non-local memory

→ Chunk the data, redistribute, and then each core sorts/works on local data.

Rule #2: Only perform sequential reads on non-local memory

→ This allows the hardware prefetcher to hide remote access latency.

Rule #3: No core should ever wait for another

→ Avoid fine-grained latching or sync barriers.

Source: Martina-Cezara Albutiu

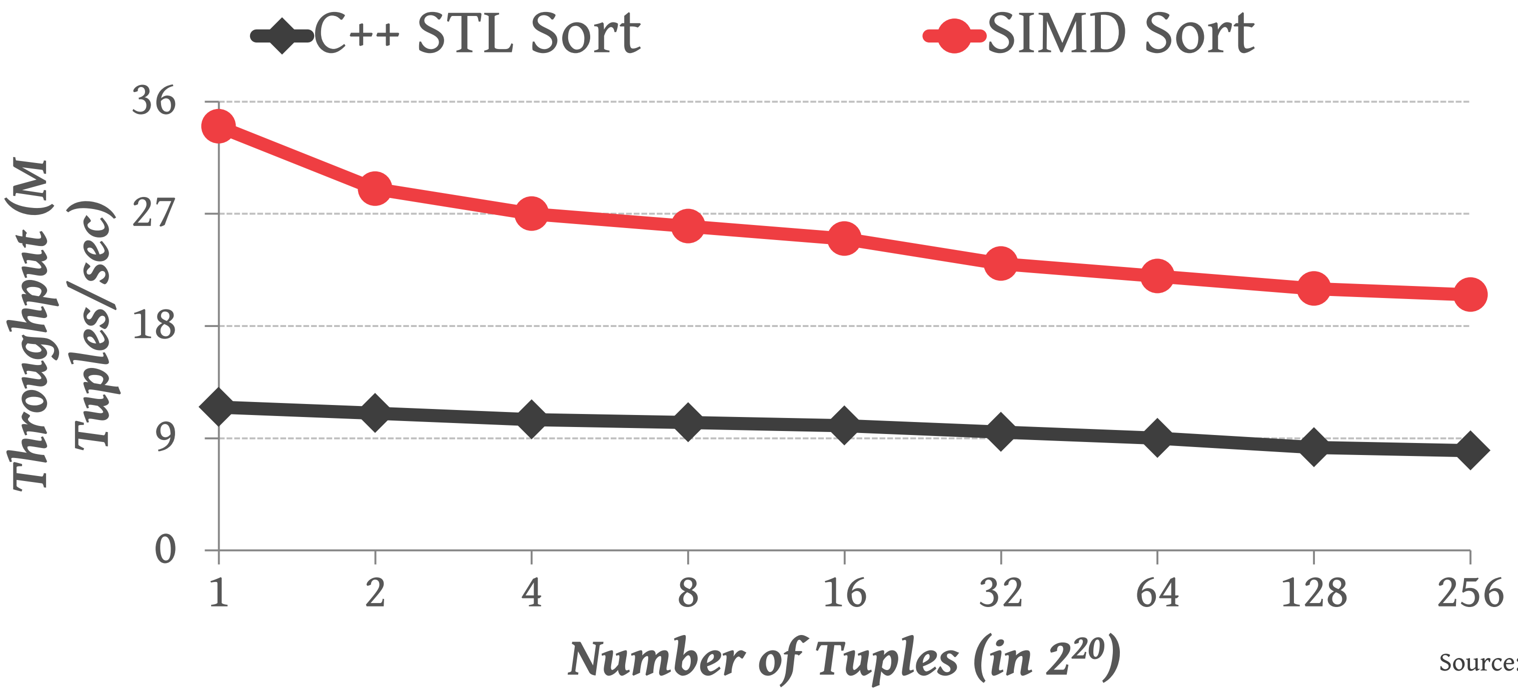
EVALUATION

Compare the different join algorithms using a synthetic data set.
→ **Sort-Merge:** M-WAY, M-PASS, MPSM
→ **Hash:** Radix Partitioning

Hardware:
→ 4 Socket Intel Xeon E4640 @ 2.4GHz
→ 8 Cores with 2 Threads Per Core
→ 512 GB of DRAM

RAW SORTING PERFORMANCE

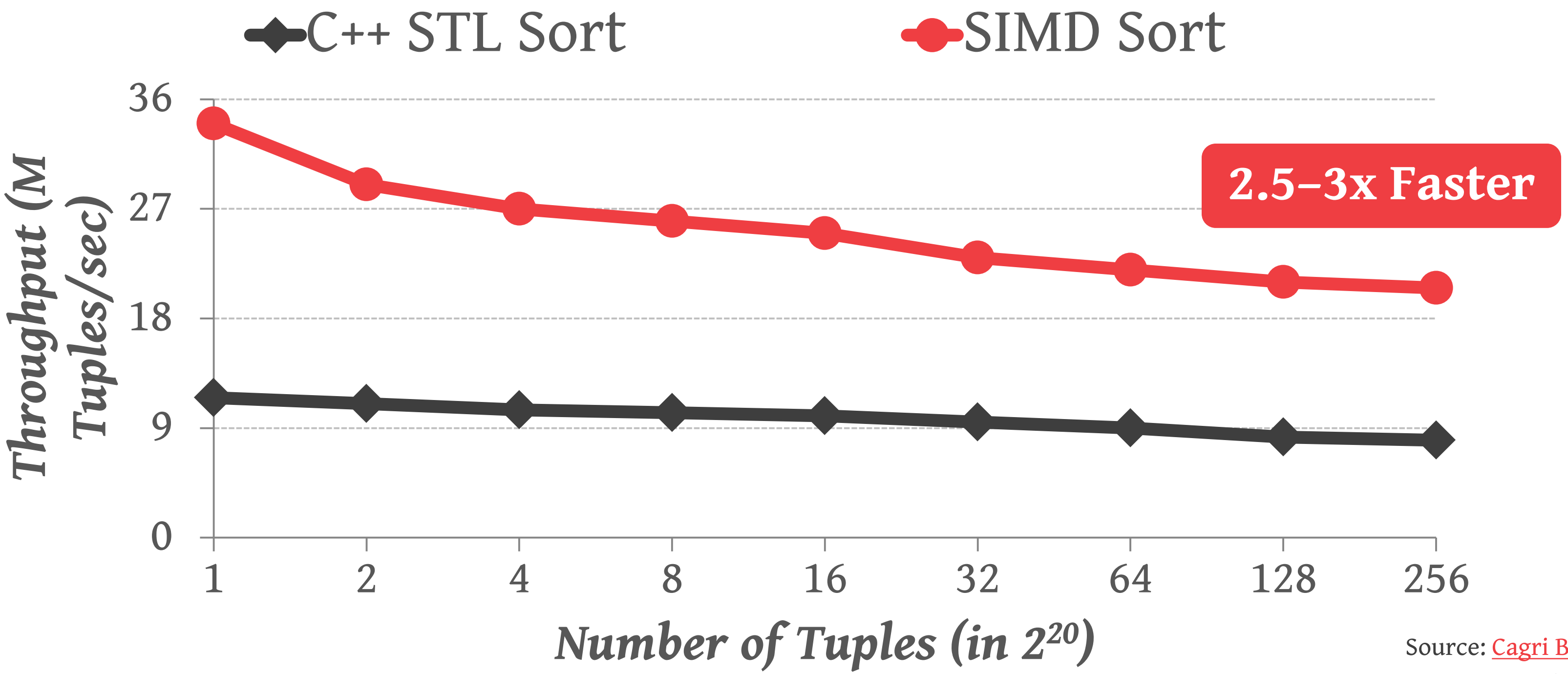
Single-threaded sorting performance



Source: Cagri Balkesen

RAW SORTING PERFORMANCE

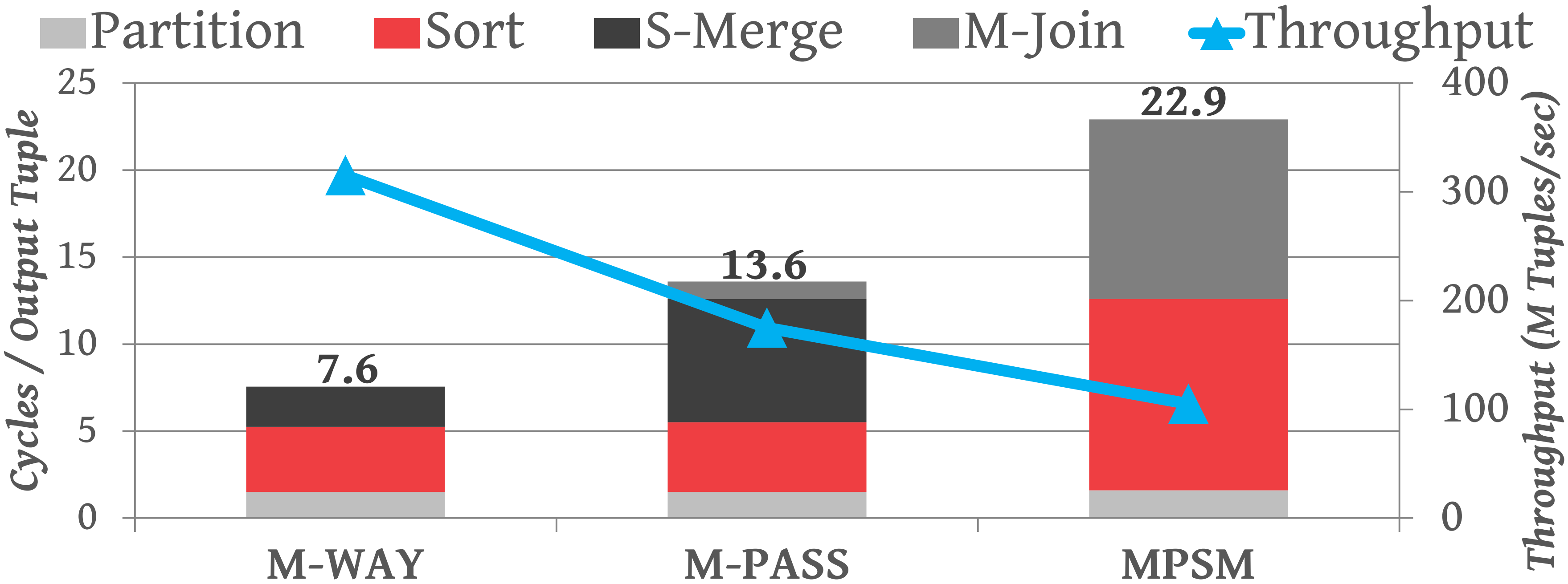
Single-threaded sorting performance



Source: Cagri Balkesen

COMPARISON OF SORT-MERGE JOINS

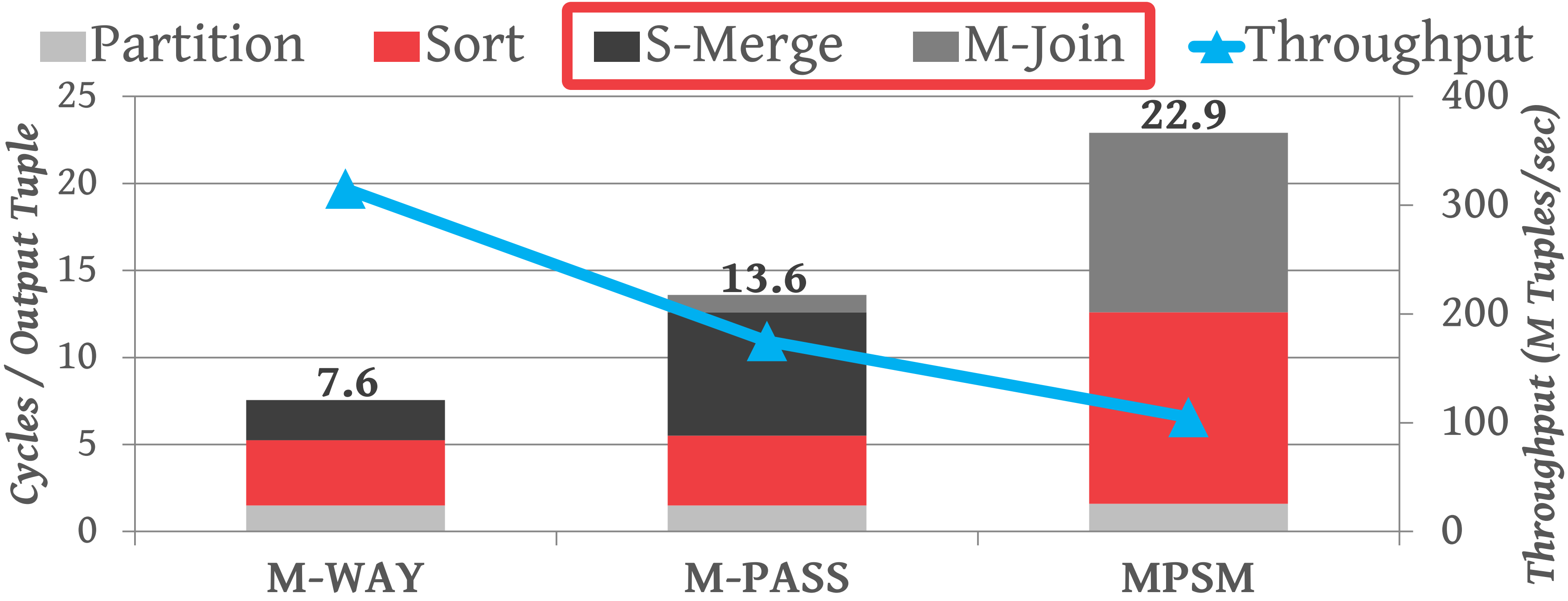
Workload: 1.6B ⋈ 128M (8-byte tuples)



Source: Cagri Balkesen

COMPARISON OF SORT-MERGE JOINS

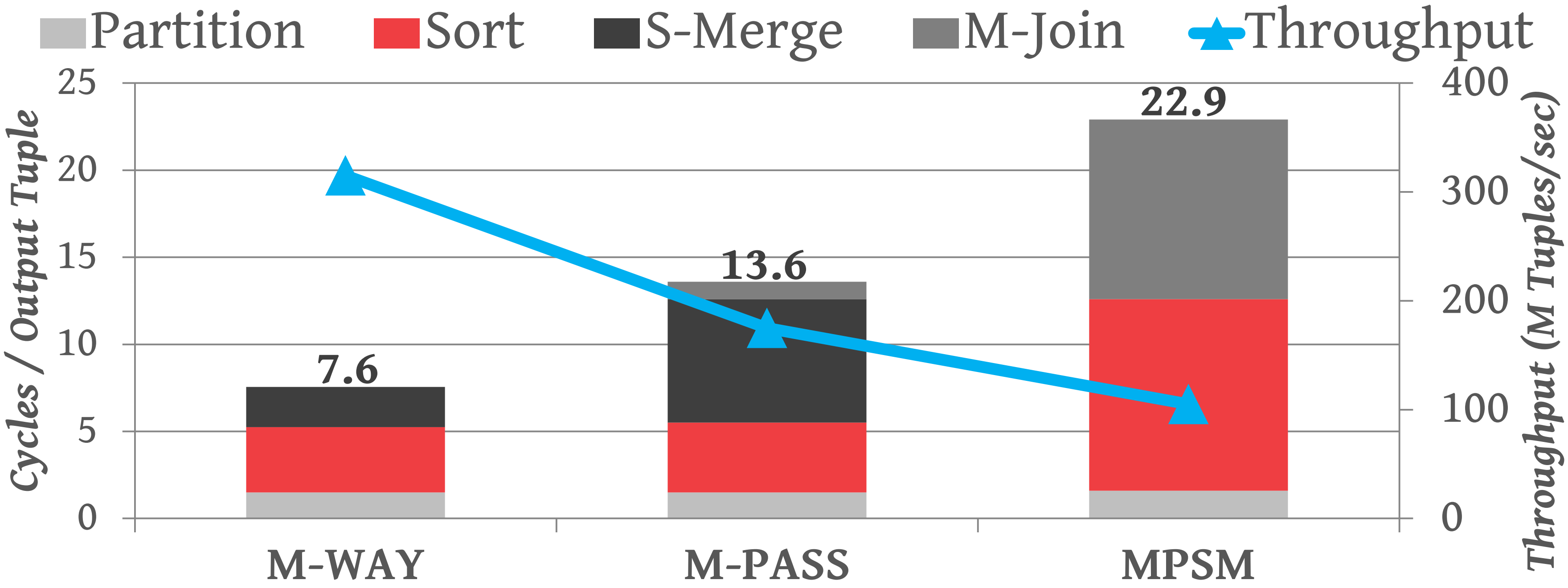
Workload: 1.6B ⋈ 128M (8-byte tuples)



Source: [Cagri Balkesen]

COMPARISON OF SORT-MERGE JOINS

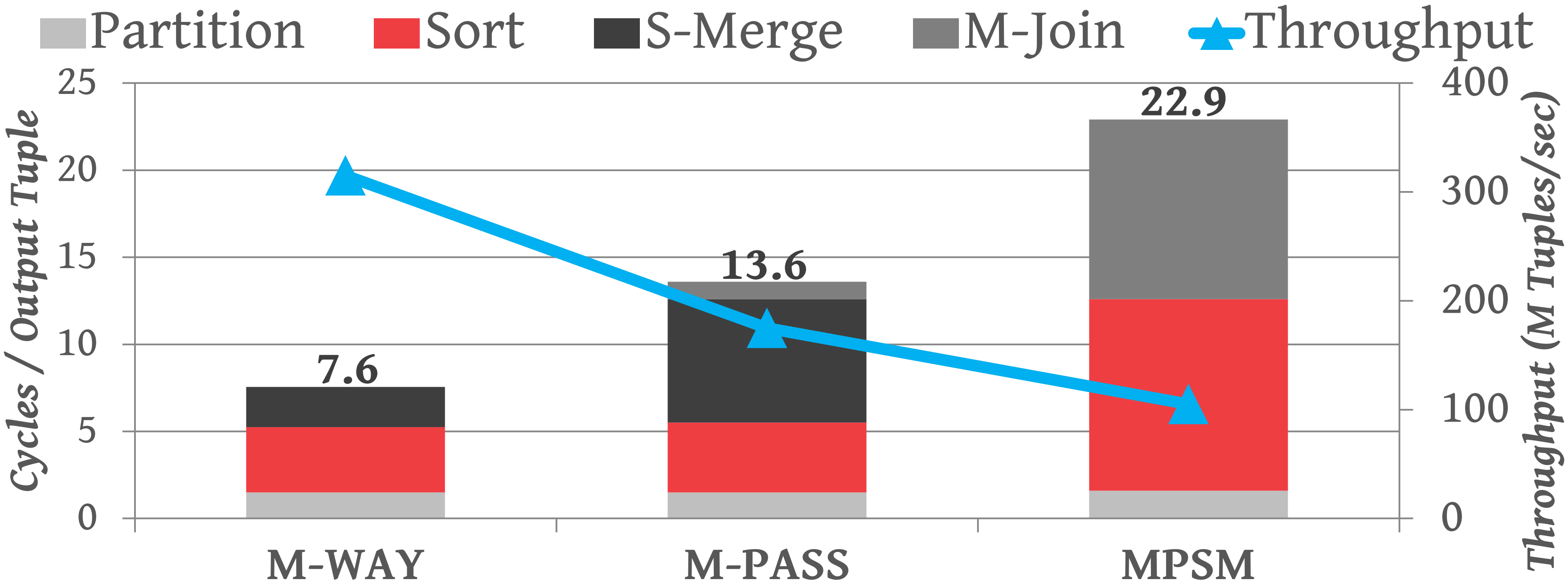
Workload: 1.6B ⋈ 128M (8-byte tuples)



Source: Cagri Balkesen

COMPARISON OF SORT-MERGE JOINS

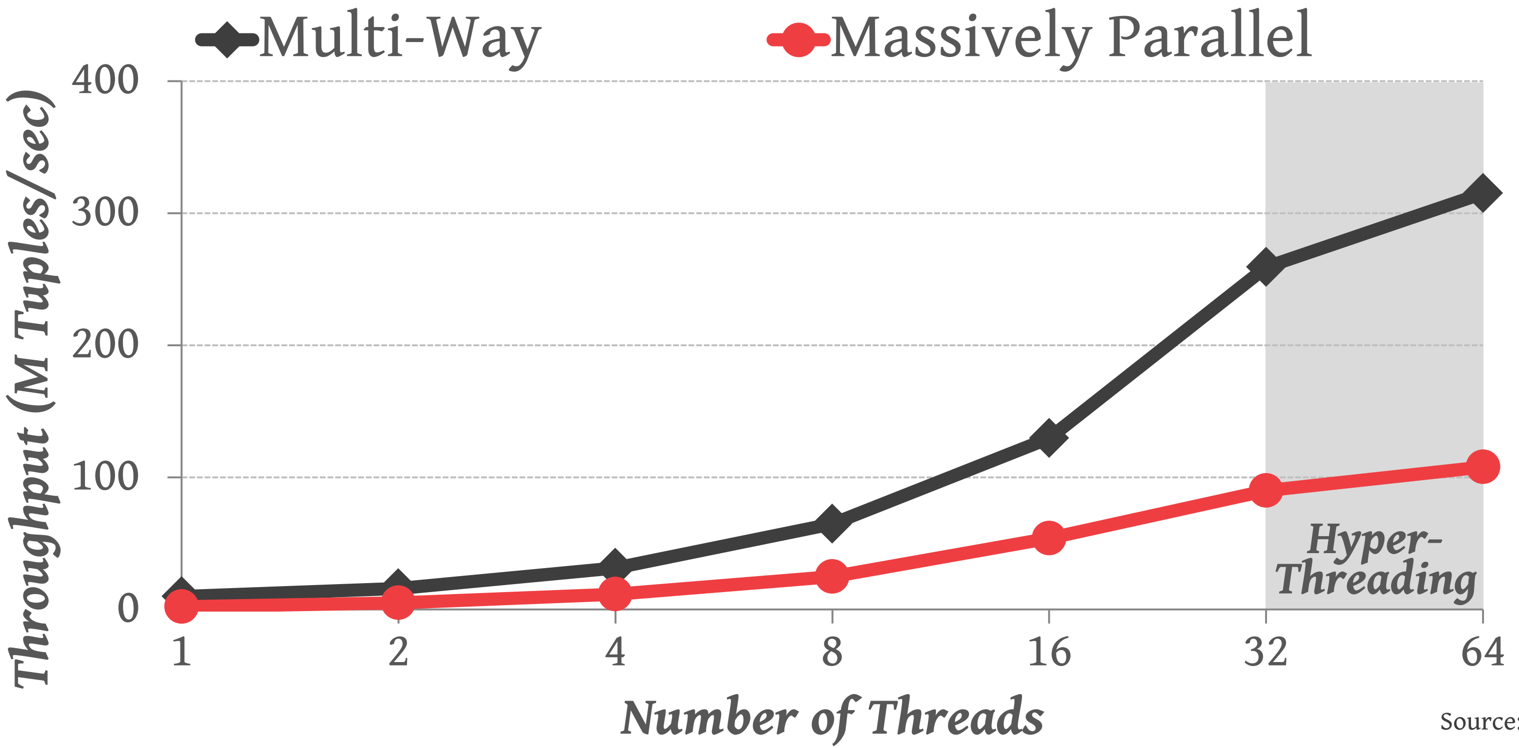
Workload: 1.6B ⋈ 128M (8-byte tuples)



Source: Cagri Balkesen

M-WAY JOIN VS. MPSM JOIN

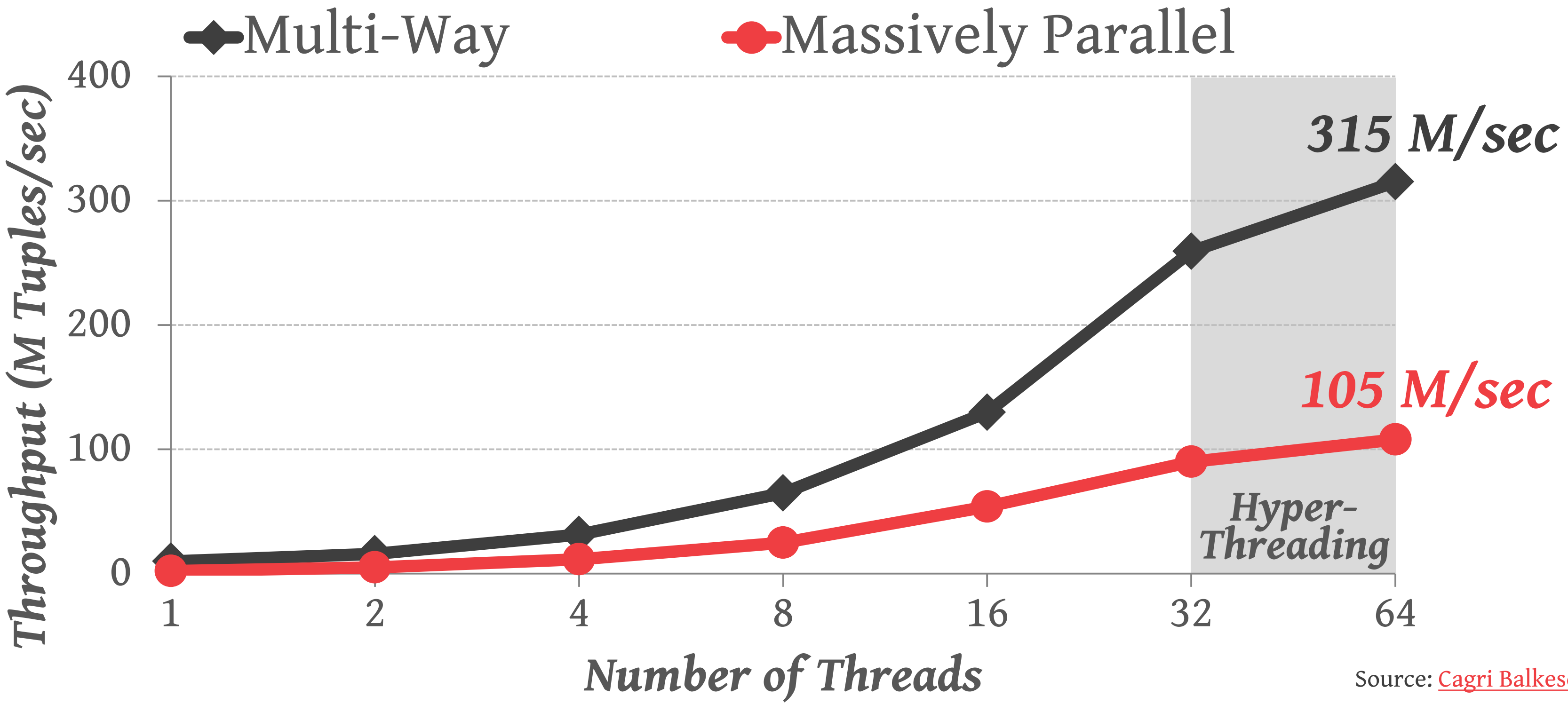
Workload: 1.6B ⋈ 128M (8-byte tuples)



Source: Cagri Balkesen

M-WAY JOIN VS. MPSM JOIN

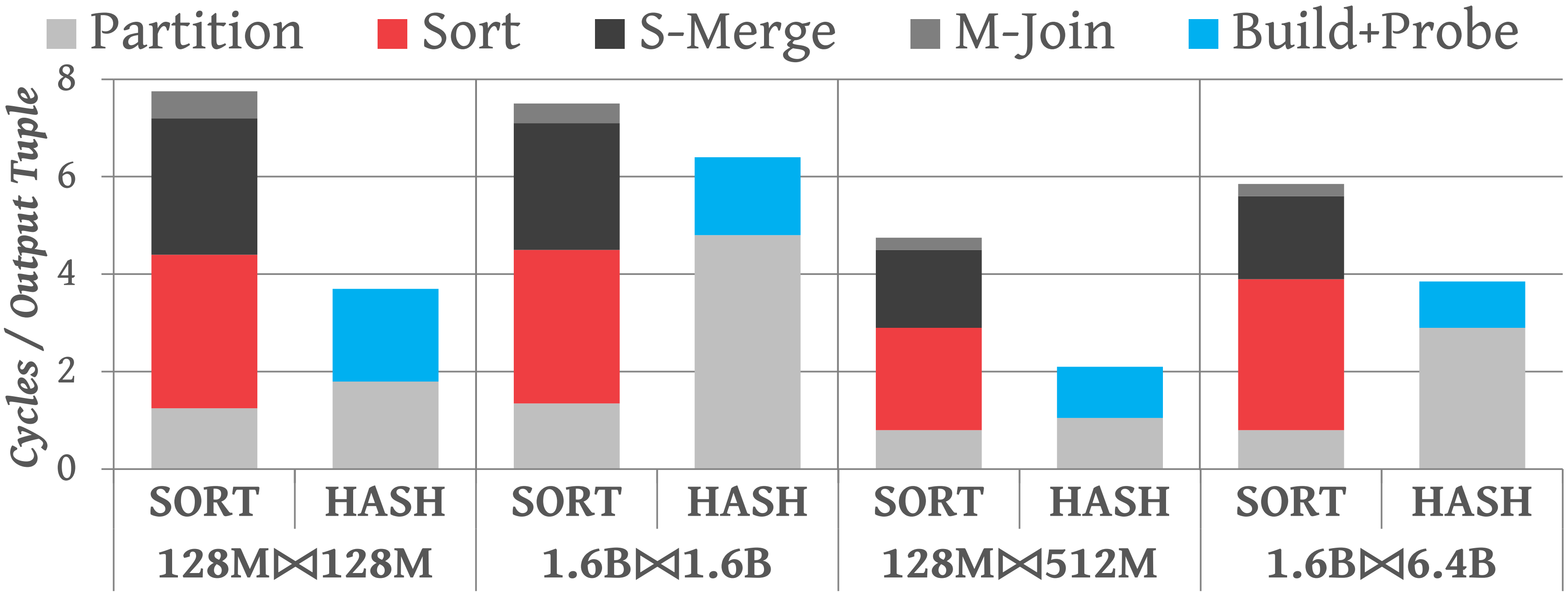
Workload: 1.6B ⋈ 128M (8-byte tuples)



Source: Cagri Balkesen

SORT-MERGE JOIN VS. HASH JOIN

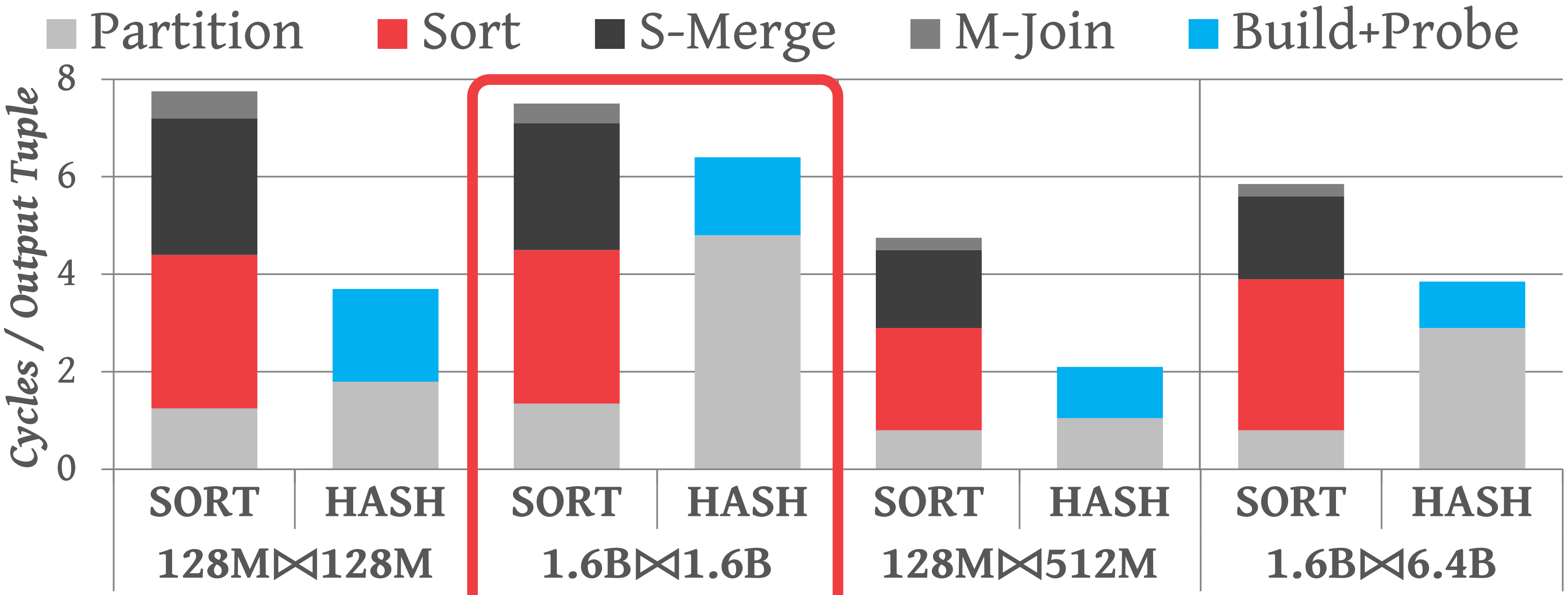
4 Socket Intel Xeon E4640 @ 2.4GHz
8 Cores with 2 Threads Per Core



Source: Cagri Balkesen

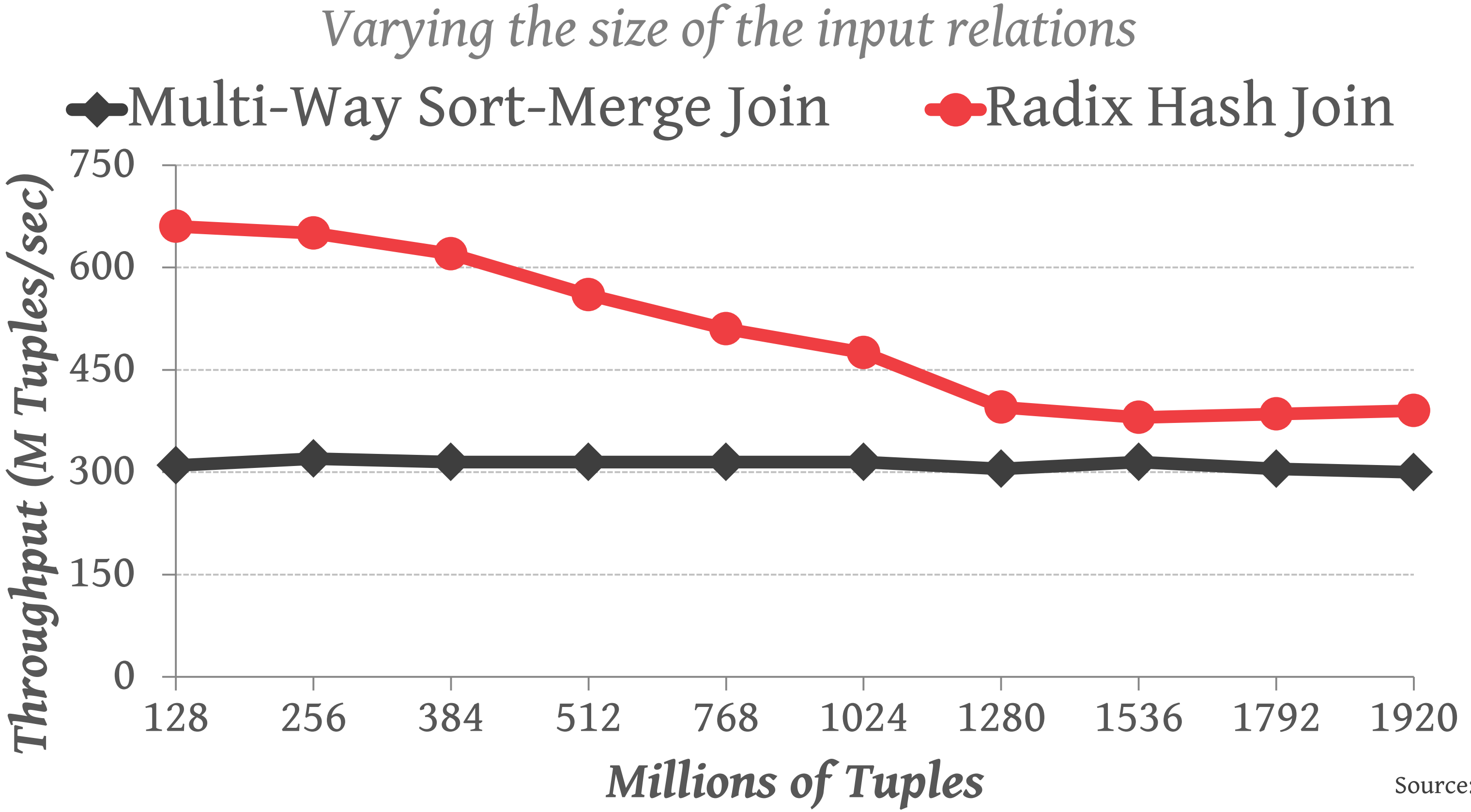
SORT-MERGE JOIN VS. HASH JOIN

4 Socket Intel Xeon E4640 @ 2.4GHz
8 Cores with 2 Threads Per Core



Source: Cagri Balkesen

SORT-MERGE JOIN VS. HASH JOIN



Source: Cagri Balkesen

PARTING THOUGHTS

Both join approaches are equally important.
Every serious OLAP DBMS supports both.

We did not consider the impact of queries where the output needs to be sorted.

HATE MAIL

Your Database Course Sucks Ass - gmail.com/Inbox – Kontact

File Edit View Message Settings Help

Print Reply Forward Trash Create To-do Ubuntu 9

Your Database Course Sucks Ass

From: [redacted]
To: "Andrew Pavlo (pavlo@cs.cmu.edu)" <pavlo@cs.cmu.edu>
Date: 02/07/16 05:43 PM

You database skills are weak. You look like a freak that hasn't showered for weeks in these videos. I feel sorry for the students that are sitting in the front row.

If I was there I would mug you in the hallway and steal your wallet.

Go fuck yourself.

[redacted]

Your Database Course Sucks Ass - gmail.com/Inbox – Kontact

File Edit View Message Settings Help

Print Reply Forward Trash Create To-do Ubuntu 9

Your Database Course Sucks Ass

From: [redacted]
To: "Andrew Pavlo
Date: 02/07/16 05:4

You database skil
these videos. I f

If I was there I

Go fuck yourself.

[redacted]

You Suck - gmail.com/Inbox – Kontact

File Edit View Message Settings Help

Print Reply Forward Trash Create To-do Ubuntu 9

You Suck

From: [redacted]
To: pavlo@cs.cmu.edu
Date: Tuesday 05:37:47 PM

```
I hate you.
I hate your course.
I hate your fake ass lessons for the street.

If I see you out in the cut I am probably going to stab you to make you bleed.
I hope you burn in hell.
```

[redacted]

Your Database Course Sucks Ass - gmail.com/Inbox – Kontact

File Edit View Message Settings Help

Print | Reply | Forward | Trash | Create To-do | Ubuntu | 9

Your Database Course Sucks Ass

From: [redacted]
To: "Andrew Pavlo
Date: 02/07/16 05:4

```
You database skil
these videos. I f

If I was there I 

Go fuck yourself.
```

[redacted]

You Suck - gmail.com/Inbox – Kontact

File Edit View Message Settings Help

Print | Reply | Forward | Trash | Create To-do | Ubuntu | 9

You Suck

From: [redacted]
To: pavlo@cs.cmu.
Date: Tuesday 05:37:

```
I hate you.
I hate your cours
I hate your fake 

If I see you out 
I hope you burn i
```

[redacted]

CMU Database Course - gmail.com/Inbox – Kontact

File Edit View Message Settings Help

Print | Reply | Forward | Trash | Create To-do | Ubuntu | 9

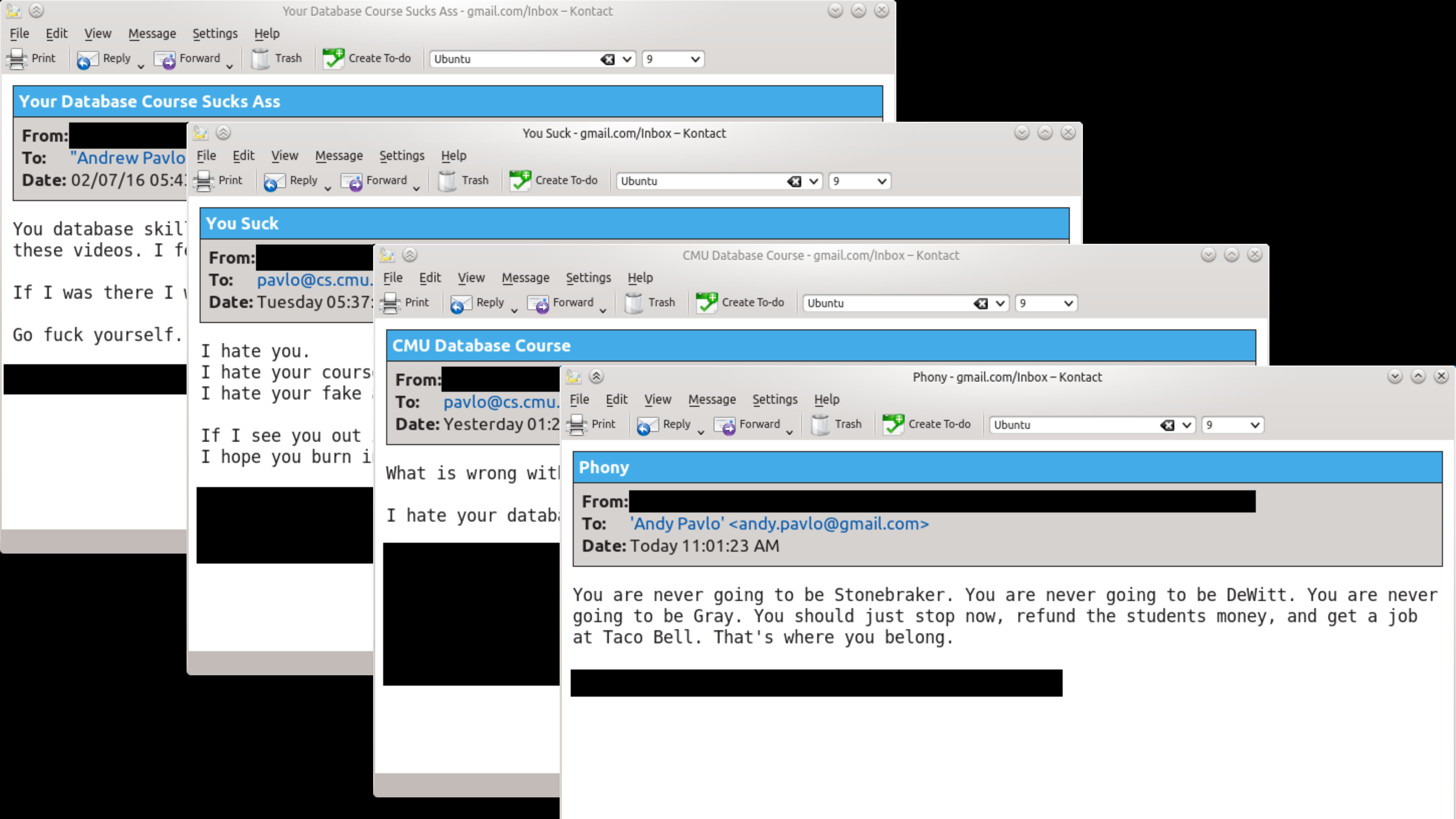
CMU Database Course

From: [redacted]
To: pavlo@cs.cmu.edu
Date: Yesterday 01:21:37 PM

```
What is wrong with your face? Why do you look like a pile of hot brown dog sluice?

I hate your database course.
```

[redacted]


Your Database Course Sucks Ass - gmail.com/Inbox – Kontact
File Edit View Message Settings Help
Print Reply Forward Trash Create To-do Ubuntu 9
Your Database Course Sucks Ass
From: [redacted]
To: "Andrew Pavlo
Date: 02/07/16 05:4
You database skil
these videos. I f
If I was there I
Go fuck yourself.
[redacted]
You Suck - gmail.com/Inbox – Kontact
File Edit View Message Settings Help
Print Reply Forward Trash Create To-do Ubuntu 9
You Suck
From: [redacted]
To: pavlo@cs.cmu.
Date: Tuesday 05:37:
I hate you.
I hate your cours
I hate your fake
If I see you out
I hope you burn i
[redacted]
CMU Database Course - gmail.com/Inbox – Kontact
File Edit View Message Settings Help
Print Reply Forward Trash Create To-do Ubuntu 9
CMU Database Course
From: [redacted]
To: pavlo@cs.cmu.
Date: Yesterday 01:2
What is wrong wit
I hate your datab
[redacted]
Phony - gmail.com/Inbox – Kontact
File Edit View Message Settings Help
Print Reply Forward Trash Create To-do Ubuntu 9
Phony
From: [redacted]
To: 'Andy Pavlo' <andy.pavlo@gmail.com>
Date: Today 11:01:23 AM
You are never going to be Stonebraker. You are never going to be DeWitt. You are never going to be Gray. You should just stop now, refund the students money, and get a job at Taco Bell. That's where you belong.
[redacted]

NEXT CLASS

Physiological Logging & Recovery