10/23/2023

Join on GPUs

Ahmedur Rahman Shovon
PhD student
Department of Computer Science
Email: ashov@uic.edu
Website: arshovon.com

UNIVERSITY OF ILLINOIS CHICAGO

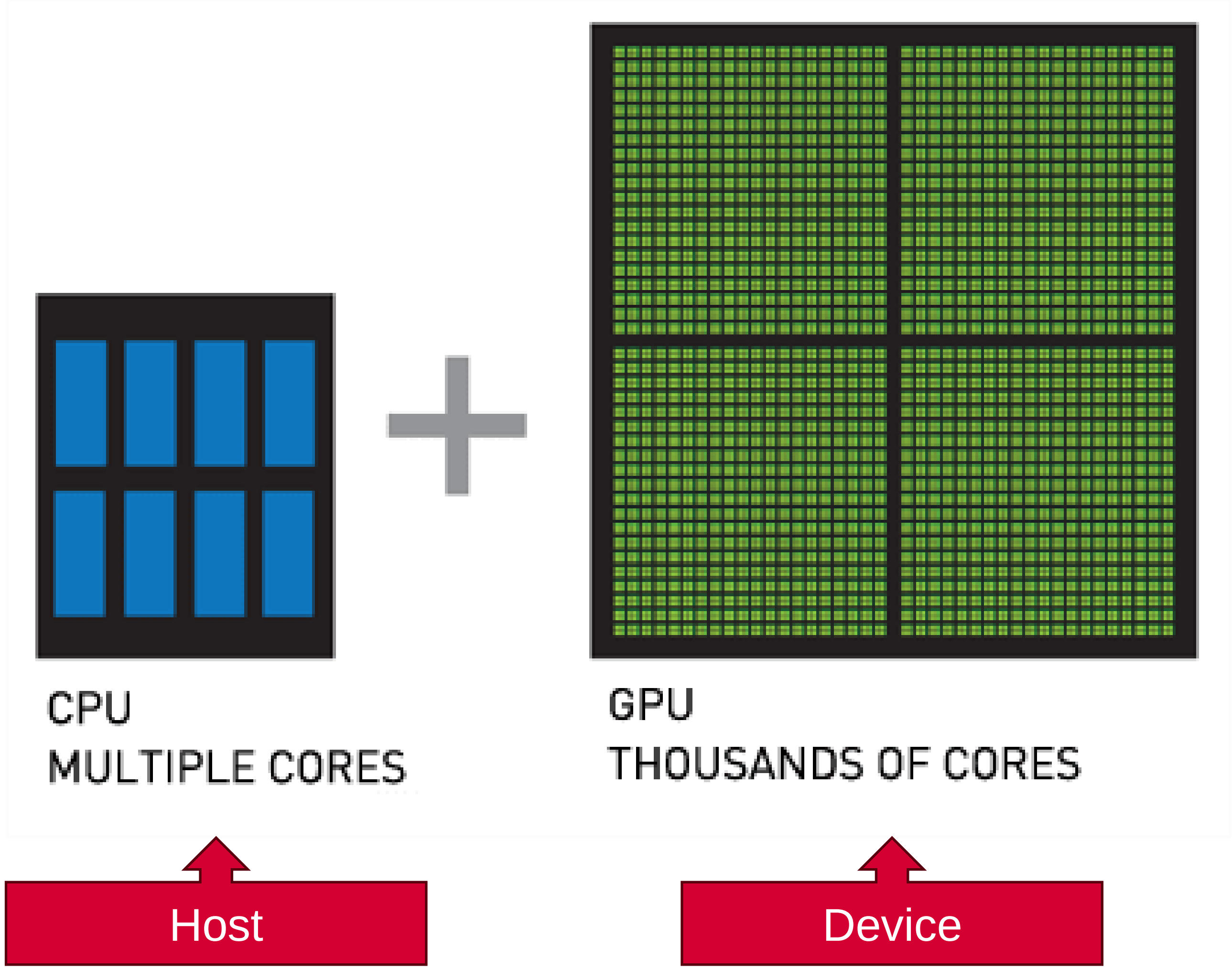
Roadmap

- Introduction to GPU
- GPGPU
- Recap to Join operation
- Parallel join
- Off-the-shelf parallel join
- Parallel hash join

Introduction to GPU

GPU

- Graphics processing unit (GPU) accelerates graphics and data processing
- Work together with CPU
- GPU is designed for parallel processing
- Use cases:
  - ✓ Graphics and video rendering
  - ✓ Gaming
  - ✓ Machine learning, AI, Deep learning
  - ✓ Scientific computing
- Major manufacturers: Nvidia, AMD, Intel



https://www.intel.com/content/www/us/en/products/docs/processors/what-is-a-gpu.html

Advancements in GPU

1970s - 1980s
Basic 2D graphics rendering

1990s
3D graphics accelerators

Early 2000s
Programmable shaders

Mid 2000s
Parallel processing on GPU

Late 2010s - Early 2020s
AI / ML acceleration

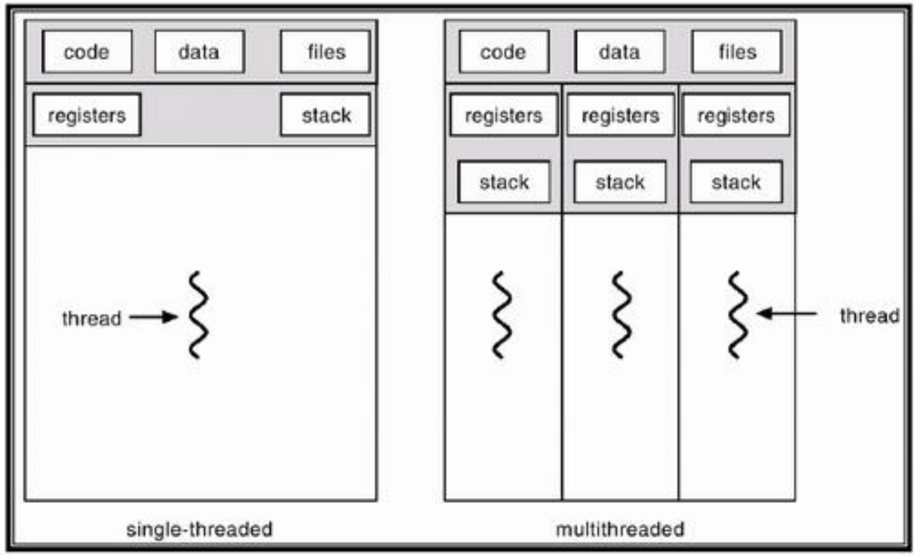
CPU vs GPU (A Sample Machine)

CPU

- 13th Gen Intel® Core™ i9-13900H
- 14 cores (6 P-cores + 8 E-cores)
- Total 20 threads
- Suitable for serial workloads
- Access to system memory (RAM)

GPU

- NVIDIA GeForce RTX 4070, 8 GB GDDR6
- 5888 CUDA cores
- Total 94,208 threads
- Suitable for parallel workloads
- Access to dedicated VRAM

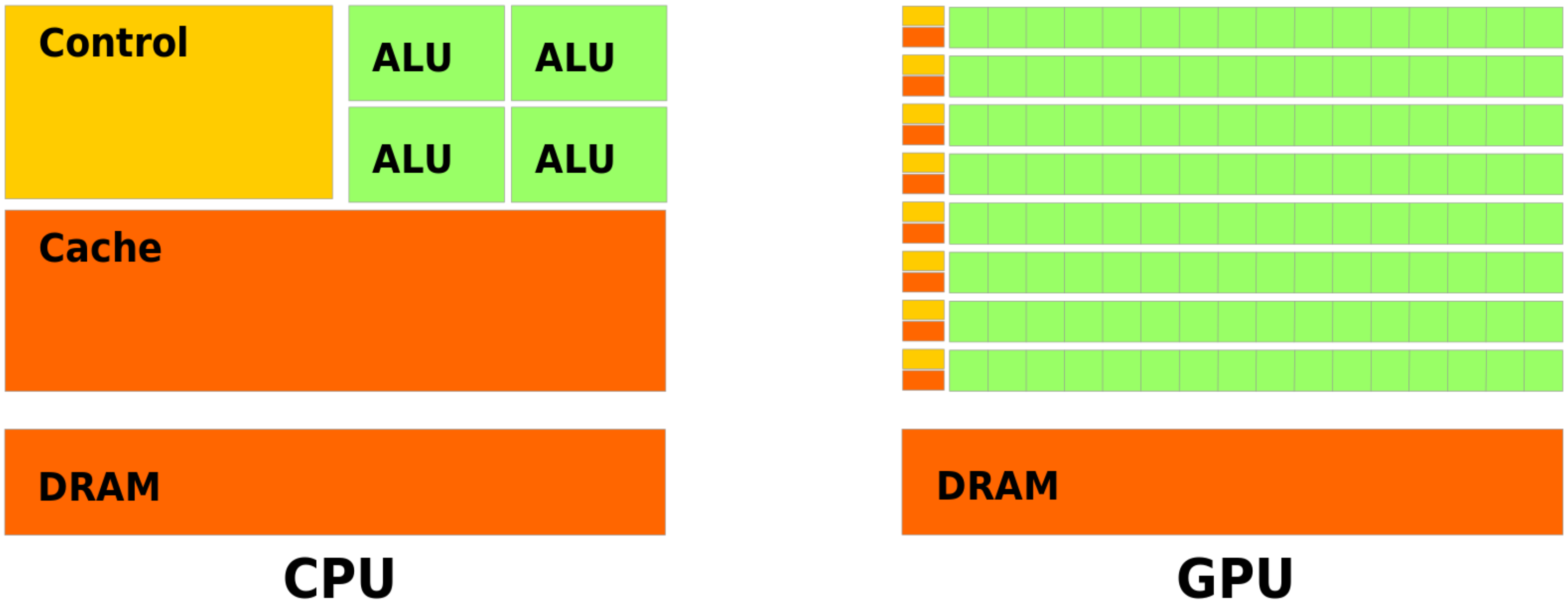


UIC

GPGPU

GPGPU

- General Purpose computing using GPU
- Influenced the scientific computing paradigms
- Offers thousands of cores
- Power efficiency TFlop per Watt



- File:Cpu-gpu.svg. (2020, October 1). Wikimedia Commons. Retrieved 18:08, April 10, 2023 from https://commons.wikimedia.org/w/index.php?title=File:Cpu-gpu.svg&oldid=477433509.
- Levinas, M. (2020, November). What is GPU computing and how is it applied today? What Is GPU Computing And How Is It Applied Today? Retrieved from https://blog.cherryservers.com/what-is-gpu-computing

GPGPU Advantages

Massive parallel processing: Scientific simulations

Efficient large dataset handling: Machine learning algorithms

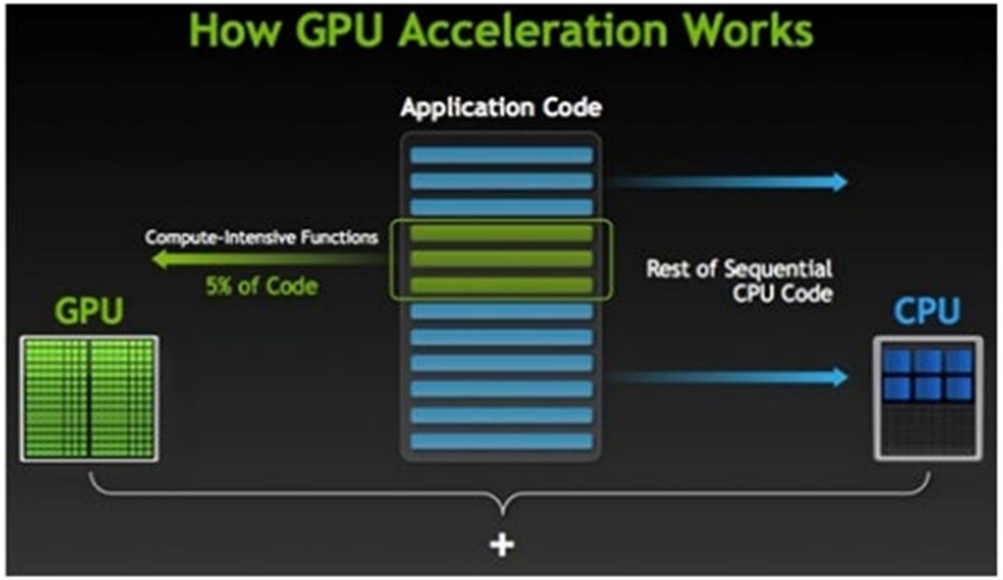
Real-time processing: Gaming and streaming

Accelerated financial modeling: Risk assessment and pricing

GPU Programming Model

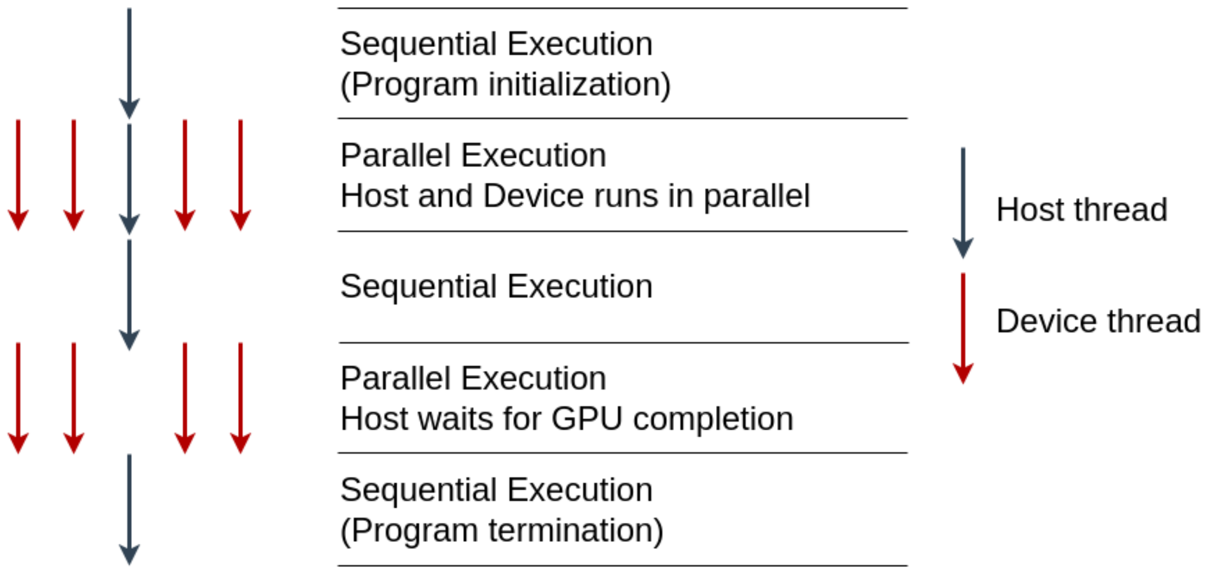
- **CUDA** - proprietary to Nvidia GPUs but most mature and established
- **HIP** - targets AMD GPUs
- **SYCL** - open standard for cross-platform portability
- **DPC++** - Intel's implementation of SYCL
- **OneAPI** - Intel's unified programming model across devices

GPU and CPU communication



CUDA Programming Model

- Globally Sequential Locally Parallel programming pattern
- Invokes parallel kernels that execute across a set of threads
- CUDA spawns the threads from a hierarchy of grid (3D) and block (3D)
- Each thread executes an instance of the kernel
- Supports C/C++, Fortan, Python



- Jason. Sanders. CUDA by example : an introduction to general-purpose GPU programming. Addison-Wesley, Upper Saddle River, NJ, 2011.
- Jianbin Fang, Chun Huang, Tao Tang, and Zheng Wang. Parallel programming models for heterogeneous many-cores: a comprehensive survey. CCF Transactions on High Performance Computing, 2(4):382–400, 2020.

GPGPU Challenges

- **Algorithm adaptation for GPU**: Sequential to parallel
- **Parallelism synchronization**: Putting barrier
- **Memory management**: Host to device and device to host data transfer
- **Portability**: Portability to different GPU devices

- Gerassimos Barlas. Chapter 6 - gpu programming. In Gerassimos Barlas, editor, Multicore and GPU Programming, pages 391–526. Morgan Kaufmann, Boston, 2015.
- J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone and J. C. Phillips, "GPU Computing," in Proceedings of the IEEE, vol. 96, no. 5, pp. 879-899, May 2008, doi: 10.1109/JPROC.2008.917757.

Recap to Join operation

Recap to Relational Data

- Relation: 2-dimensional structure
- Attribute: Represents characteristics
- Row: Represents unique record
- Join (⋈): Combines data from relations
- Projection (Π): Select specific columns

Relation

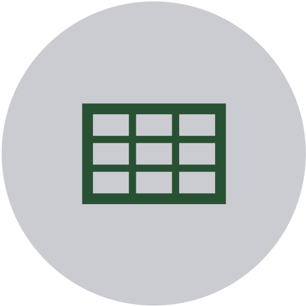
UserID	UserName	UserEmail
101	Alice	alice@example.com
102	Bob	bob@example.com

Tuple (Row)

Attribute (Column)

- Sidharth Kumar and Thomas Gilray. Distributed relational algebra at scale. In International Conference on High Performance Computing, Data, and Analytics (HiPC). IEEE, 2019.
- Sidharth Kumar and Thomas Gilray. Load-balancing parallel relational algebra. In International Conference on High Performance Computing, pages 288–308. Springer, 2020.

Why Join is Important?



COMBINE DATA FROM MULTIPLE TABLES

FIND PATTERNS IN DATA

CLEAN DATA

CREATE NEW DATA SETS

- Daniel Zinn, Haicheng Wu, Jin Wang, Molham Aref, and Sudhakar Yalamanchili. General-purpose join algorithms for large graph triangle listing on heterogeneous systems. In Proceedings of the 9th Annual Workshop on General Purpose Processing Using Graphics Processing Unit, pages 12–21, 2016.

Example of Natural Join ⋈

User

UserID	UserName	UserEmail
101	Alice	alice@example.com
102	Bob	bob@example.com
103	Eve	eve@example.com

⋈

Order

UserID	OrderTotal	Items
101	25.69	2
102	145.66	3
103	12.11	1
103	44.00	2

Example of Natural Join ⋈

User

UserID	UserName	UserEmail
101	Alice	alice@example.com
102	Bob	bob@example.com
103	Eve	eve@example.com

⋈

Order

UserID	OrderTotal	Items
101	25.69	2
102	145.66	3
103	12.11	1
103	44.00	2

User ⋈ Order

UserID	UserName	UserEmail	OrderTotal	Items
101	Alice	alice@example.com	25.69	2

Example of Natural Join ⋈

User

UserID	UserName	UserEmail
101	Alice	alice@example.com
102	Bob	bob@example.com
103	Eve	eve@example.com

⋈

Order

UserID	OrderTotal	Items
101	25.69	2
102	145.66	3
103	12.11	1
103	44.00	2

User ⋈ **Order**

UserID	UserName	UserEmail	OrderTotal	Items
101	Alice	alice@example.com	25.69	2
102	Bob	bob@example.com	145.66	3

Example of Natural Join ⋈

User

UserID	UserName	UserEmail
101	Alice	alice@example.com
102	Bob	bob@example.com
103	Eve	eve@example.com

⋈

Order

UserID	OrderTotal	Items
101	25.69	2
102	145.66	3
103	12.11	1
103	44.00	2

User ⋈ **Order**

UserID	UserName	UserEmail	OrderTotal	Items
101	Alice	alice@example.com	25.69	2
102	Bob	bob@example.com	145.66	3
103	Eve	eve@example.com	12.11	1

Example of Natural Join ⋈

User (Outer Relation)

UserID	UserName	UserEmail
101	Alice	alice@example.com
102	Bob	bob@example.com
103	Eve	eve@example.com

⋈

Order (Inner Relation)

UserID	OrderTotal	Items
101	25.69	2
102	145.66	3
103	12.11	1
103	44.00	2

User ⋈ Order

UserID	UserName	UserEmail	OrderTotal	Items
101	Alice	alice@example.com	25.69	2
102	Bob	bob@example.com	145.66	3
103	Eve	eve@example.com	12.11	1
103	Eve	eve@example.com	44.00	2

Example of Natural Join ⋈

User (Outer Relation)

UserID	UserName	UserEmail
101	Alice	alice@example.com
102	Bob	bob@example.com
103	Eve	eve@example.com

⋈

Order (Inner Relation)

UserID	OrderTotal	Items
101	25.69	2
102	145.66	3
103	12.11	1
103	44.00	2

User ⋈ Order

UserID	UserName	UserEmail	OrderTotal	Items
101	Alice	alice@example.com	25.69	2
102	Bob	bob@example.com	145.66	3
103	Eve	eve@example.com	12.11	1
103	Eve	eve@example.com	44.00	2

Duplicates on Join Result

2
2

User ⋈ Order

UserID	UserName	UserEmail	OrderTotal	Items
101	Alice	alice@example.com	25.69	2
102	Bob	bob@example.com	145.66	3
103	Eve	eve@example.com	12.11	1
103	Eve	eve@example.com	44.00	2

$\Pi_{(UserName, UserEmail)}(User \bowtie Order)$

UserName	UserEmail
Alice	alice@example.com
Bob	bob@example.com
Eve	eve@example.com
Eve	eve@example.com

Parallel Join

How can we do join in parallel?

User (Outer Relation)

UserID	UserName	UserEmail
101	Alice	alice@example.com
102	Bob	bob@example.com
103	Eve	eve@example.com

$\bowtie$

Order (Inner Relation)

UserID	OrderTotal	Items
101	25.69	2
102	145.66	3
103	12.11	1
103	44.00	2

Parallel Join ⋈

User (Outer Relation)

UserID	UserName	UserEmail
101	Alice	alice@example.com
102	Bob	bob@example.com
103	Eve	eve@example.com

Order (Inner Relation)

UserID	OrderTotal	Items
101	25.69	2
102	145.66	3
103	12.11	1
103	44.00	2

⋈

Thread 1

Thread 2

Thread 3

All executing in parallel

Parallel Join

What: Perform relational join operation simultaneously on a number of processors or machines

When: Useful when input data is enormous and the join is computationally costly

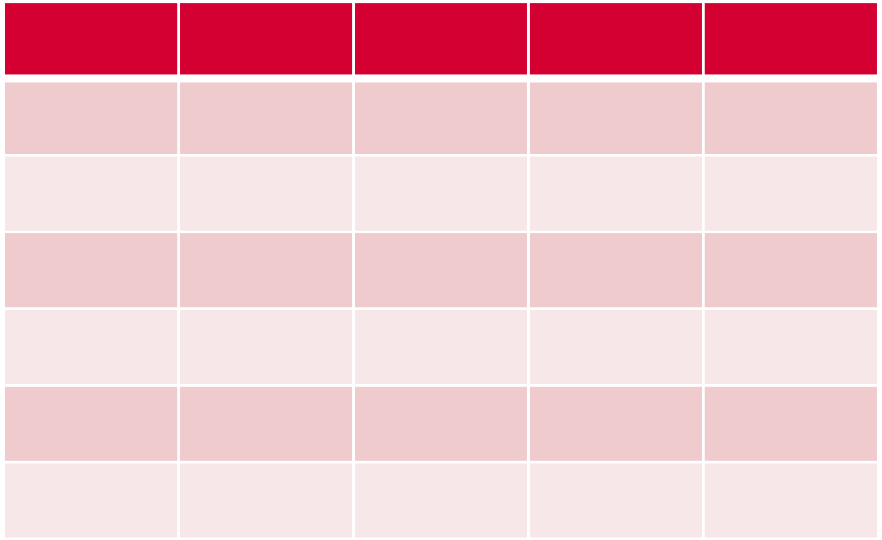
How: Divide the data into partitions and assign each partition to a different processor

- Daniel Zinn, Haicheng Wu, Jin Wang, Molham Aref, and Sudhakar Yalamanchili. General-purpose join algorithms for large graph triangle listing on heterogeneous systems. In Proceedings of the 9th Annual Workshop on General Purpose Processing Using Graphics Processing Unit, pages 12–21, 2016.

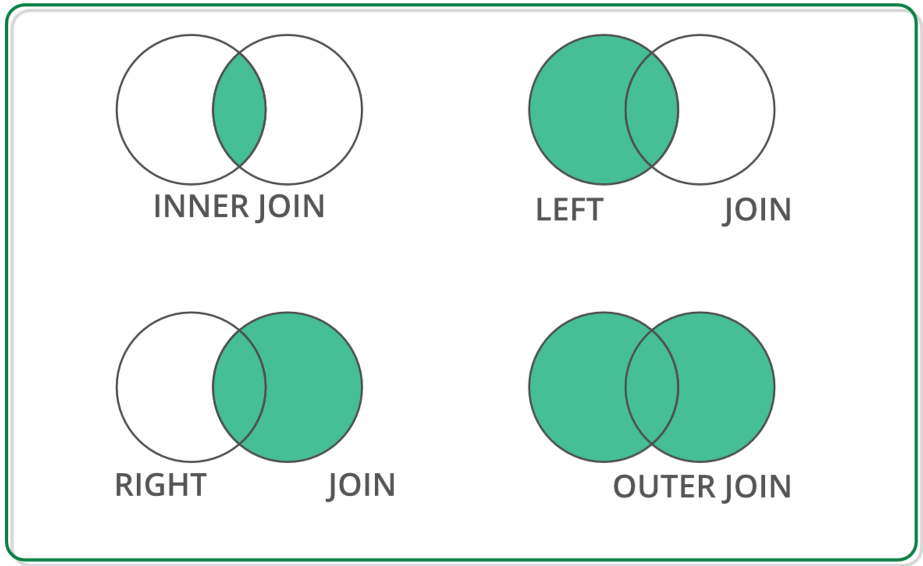
Off-the-shelf Parallel Join

Off-the-shelf Data Structure for Join Operation

DataFrame: 2D labeled tabular data structure



DataFrame has RA primitives APIs



- Reback, J., McKinney, W., Van Den Bossche, J., Augspurger, T., Cloud, P., Klein, A., ... & Seabold, S. (2020). pandas-dev/pandas: Pandas 1.0. 5. Zenodo.
- Chen, D. Y. (2017). Pandas for everyone: Python data analysis. Addison-Wesley Professional.
- Singh, R. (2020, July 1). Merging DataFrames with Pandas: Pd.merge(). Medium. Retrieved April 8, 2023, from https://medium.com/swlh/merging-dataframes-with-pandas-pd-merge-7764c7e2d46d

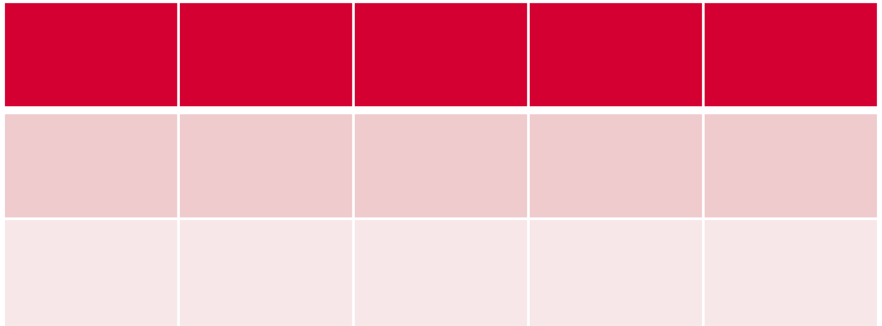
Off-the-shelf Python Libraries

pandas

RAPIDS

CPU

DataFrame: 2D labeled tabular data structure



GPU
(NVIDIA cuDF)

Both supports join operation with similar APIs

- Reback, J., McKinney, W., Van Den Bossche, J., Augspurger, T., Cloud, P., Klein, A., ... & Seabold, S. (2020). pandas-dev/pandas: Pandas 1.0. 5. Zenodo.
- Chen, D. Y. (2017). Pandas for everyone: Python data analysis. Addison-Wesley Professional.
- Green, O., Du, Z., Patel, S., Xie, Z., Liu, H., & Bader, D. A. (2021, December). Anti-Section Transitive Closure. In 2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC) (pp. 192-201). IEEE.
- Fender, A., Rees, B., & Eaton, J. RAPIDS cuGraph. In Massive Graph Analytics (pp. 483-493). Chapman and Hall/CRC.

CPU (Pandas) and GPU (cuDF)

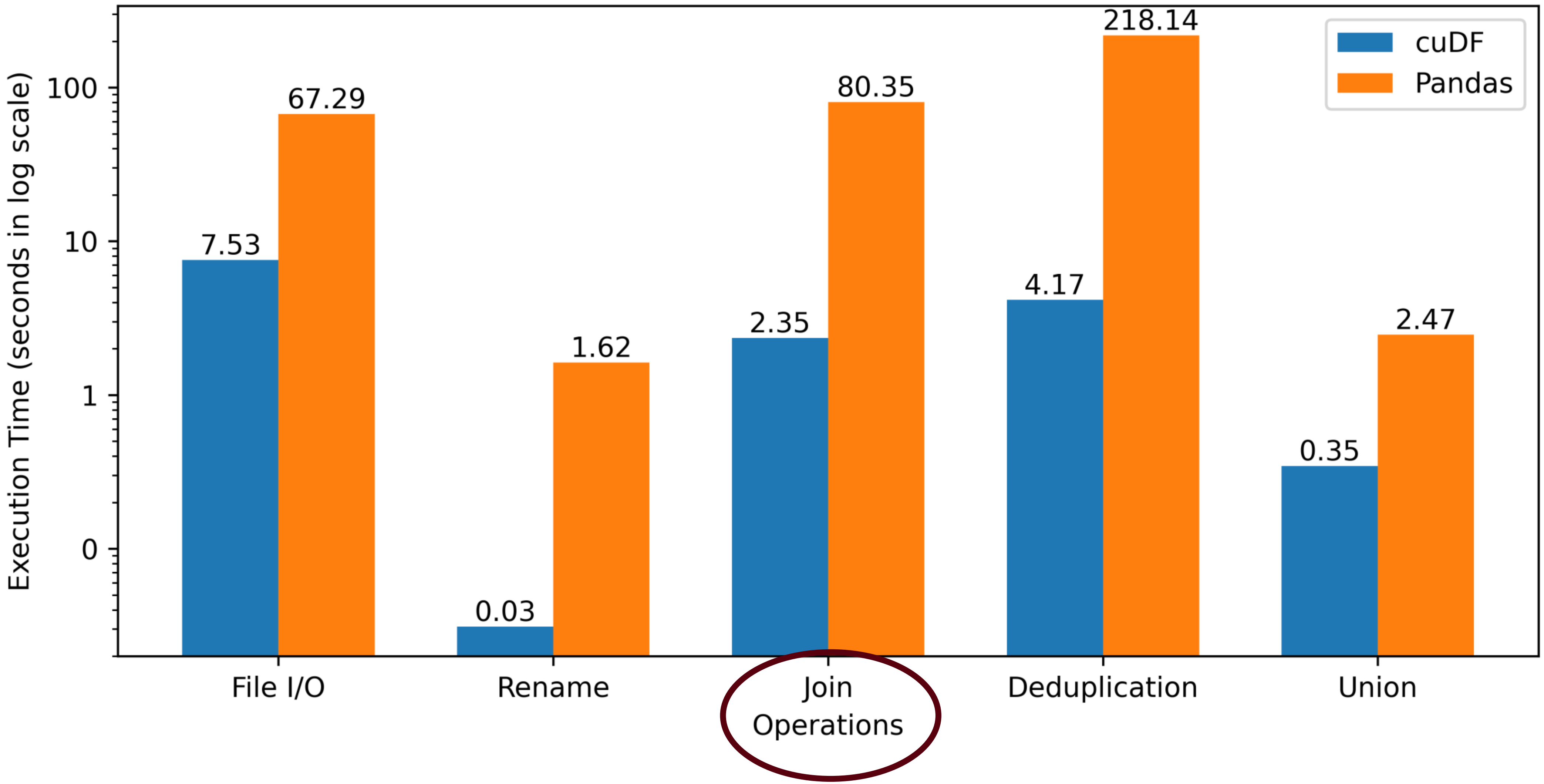
```
import pandas as pd      ← CPU Environment
import cudf              ← GPU Environment

def get_read_csv(filename, method='cudf', n):
    column_names = ['column 1', 'column 2']
    if method == 'df':
        return pd.read_csv(filename, sep='\t', header=None,
                           names=column_names, nrows=n)
    return cudf.read_csv(filename, sep='\t', header=None,
                         names=column_names, nrows=n)

def get_join(relation_1, relation_2):
    column_names = ['column 1', 'column 2']
    return relation_1.merge(relation_2, on=column_names[0],
                            how="inner",
                            suffixes=('_relation_1', '_relation_2'))
```

- https://github.com/harp-lab/GPUJoin/edit/main/IA3_2022/basic_operations.py

Performance Improvement of using GPU



- https://github.com/harp-lab/GPUJoin/edit/main/IA3_2022/basic_operations.py

Parallel Hash Join

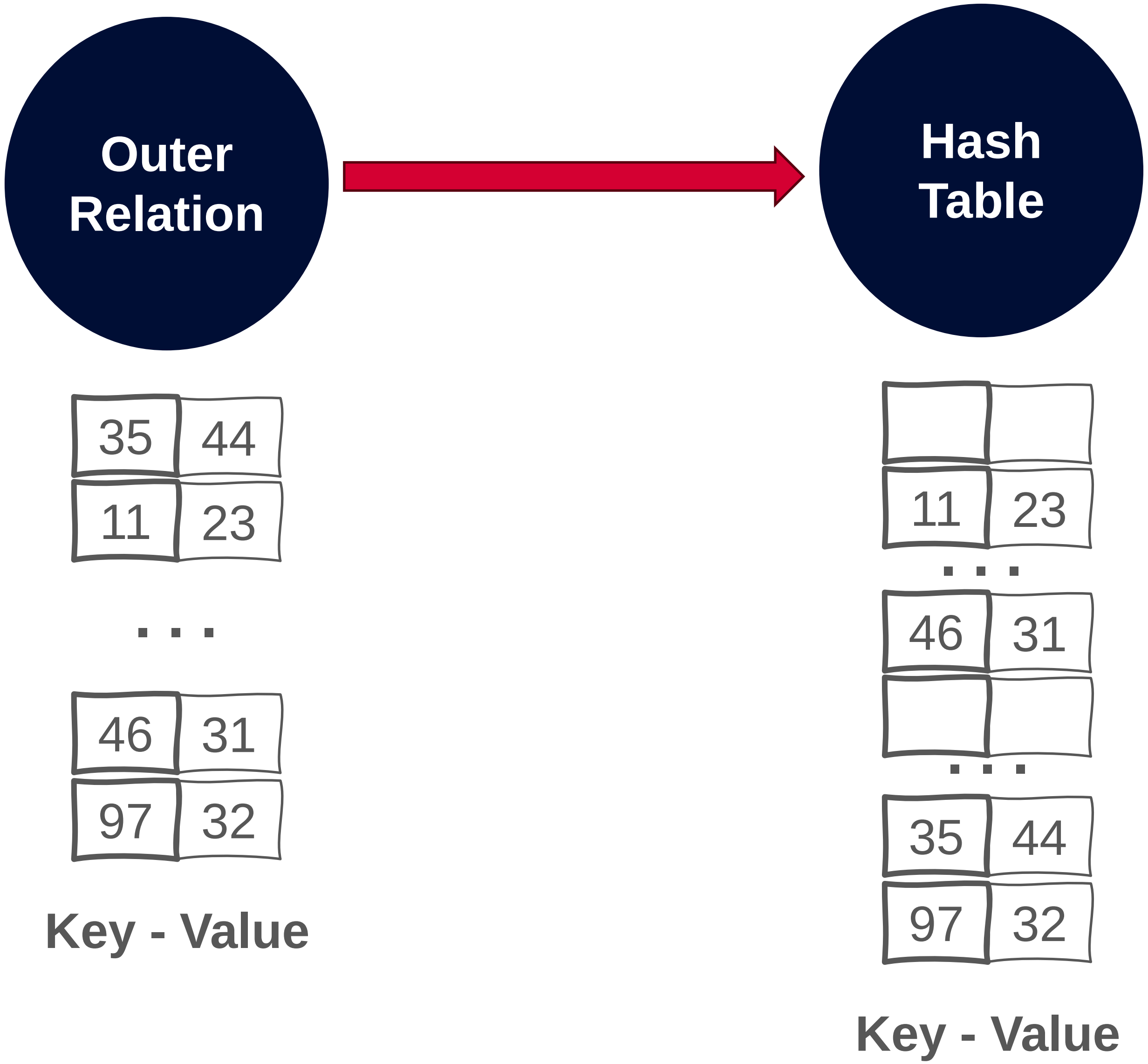
Parallel Join: Algorithms

Popular algorithms	
Sort-Merge Join (SMJ)	Hash Join (HJ)

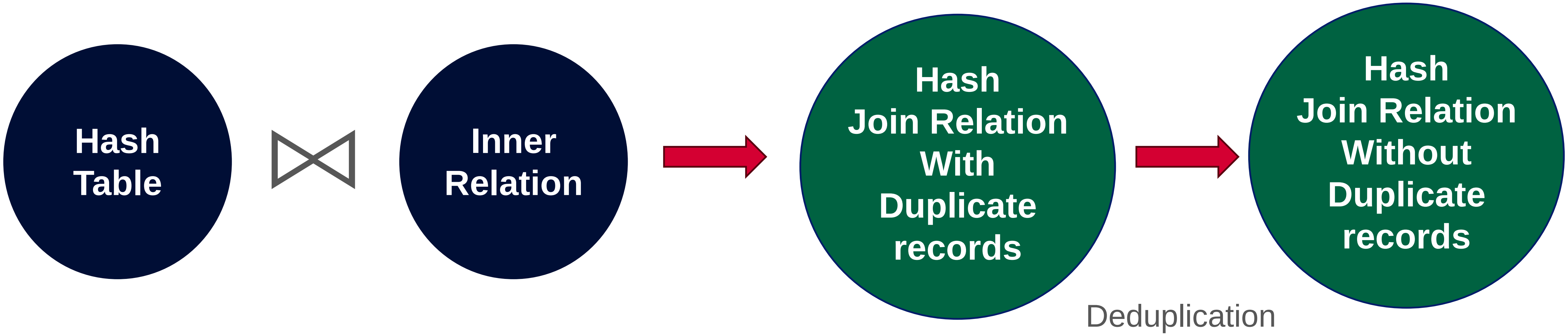
SMJ is suitable for small to medium-sized tables
HJ is suitable for large tables

- Chengxin Guo, Hong Chen, Feng Zhang, and Cuiping Li. Parallel hybrid join algorithm on gpu. 2019IEEE 21st International Conference on High Performance Computing and Communications; IEEE17th International Conference on Smart City; IEEE 5th International Conference on Data Science andSystems (HPCC/SmartCity/DSS), pages 1572–1579, 2019.
- Hongzhi Wang, Ning Li, Zheng ke Wang, and Jianing Li. Gpu-based efficient join algorithms on hadoop.The Journal of Supercomputing, 77:292 – 321, 2020.

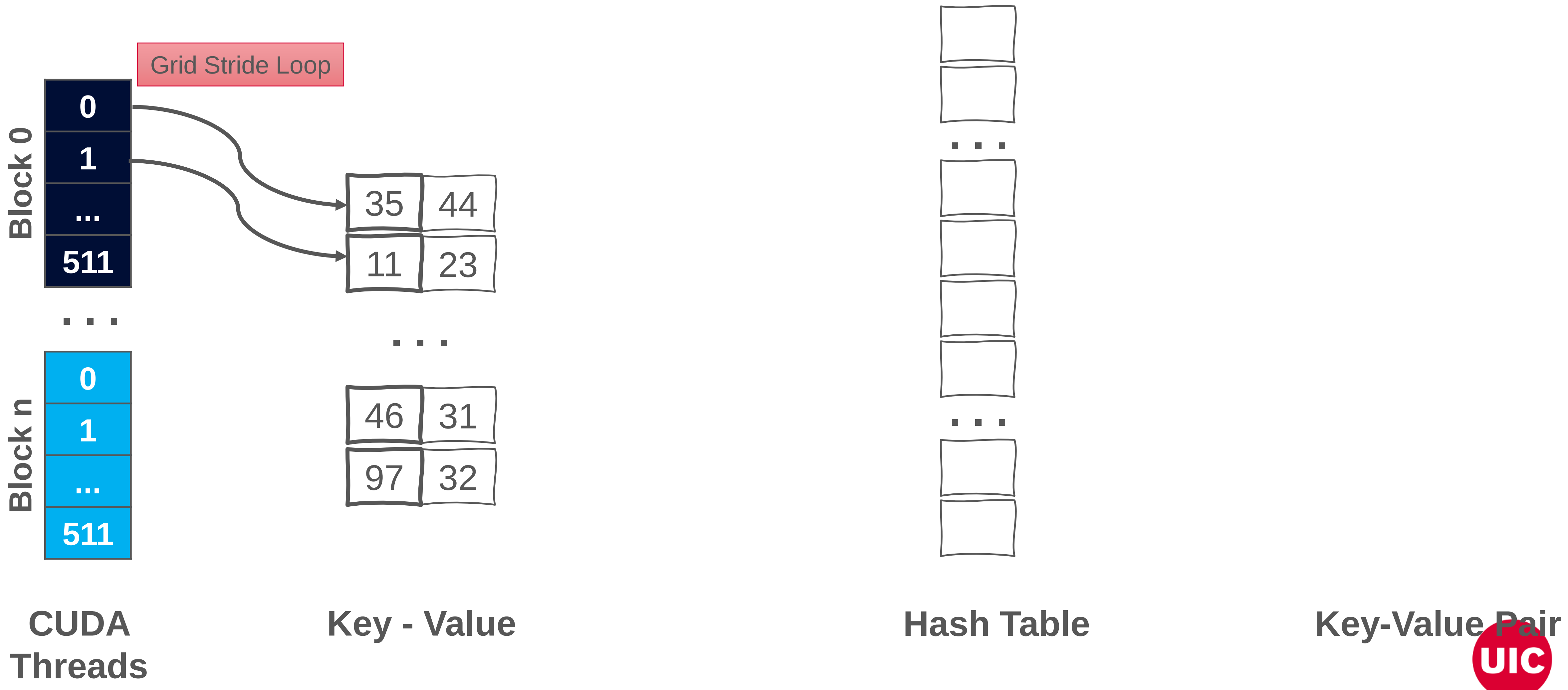
Hash Join Process



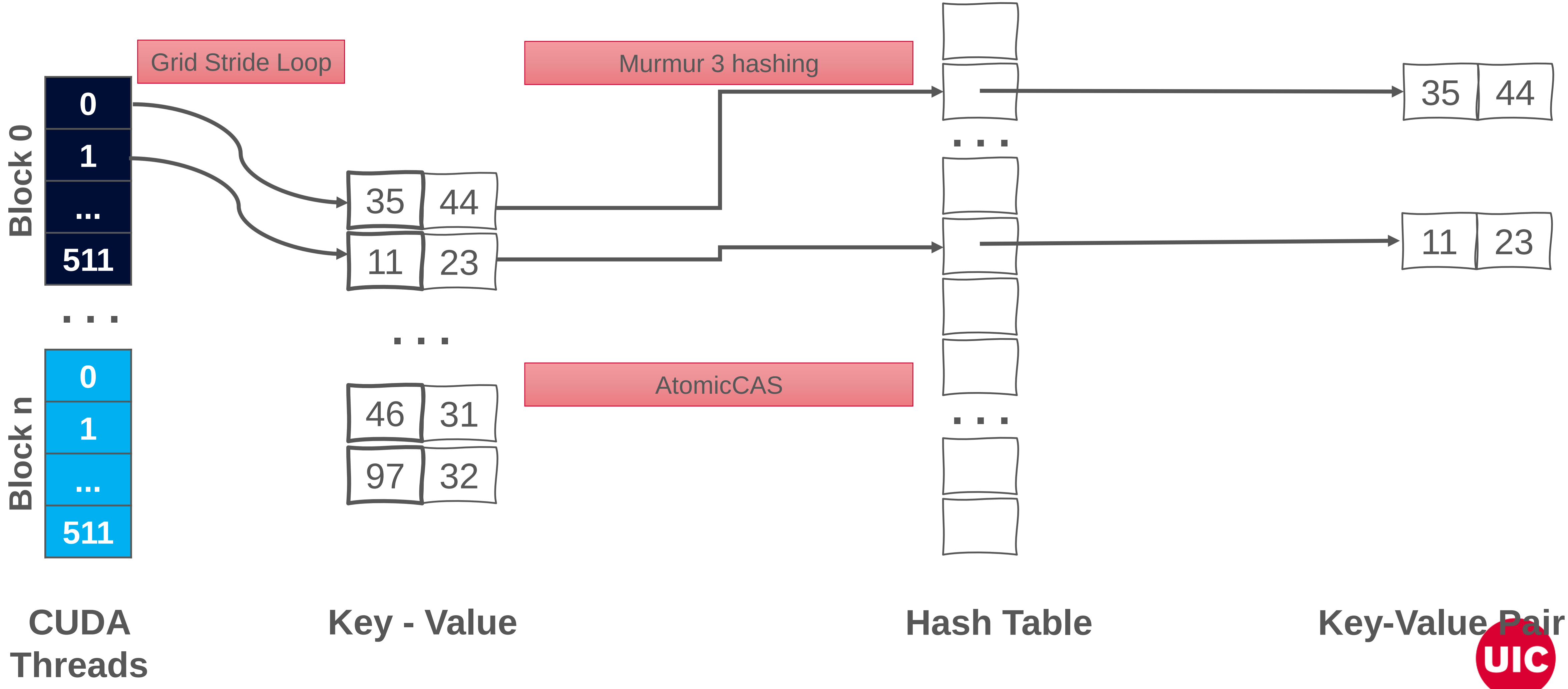
Hash Join Process



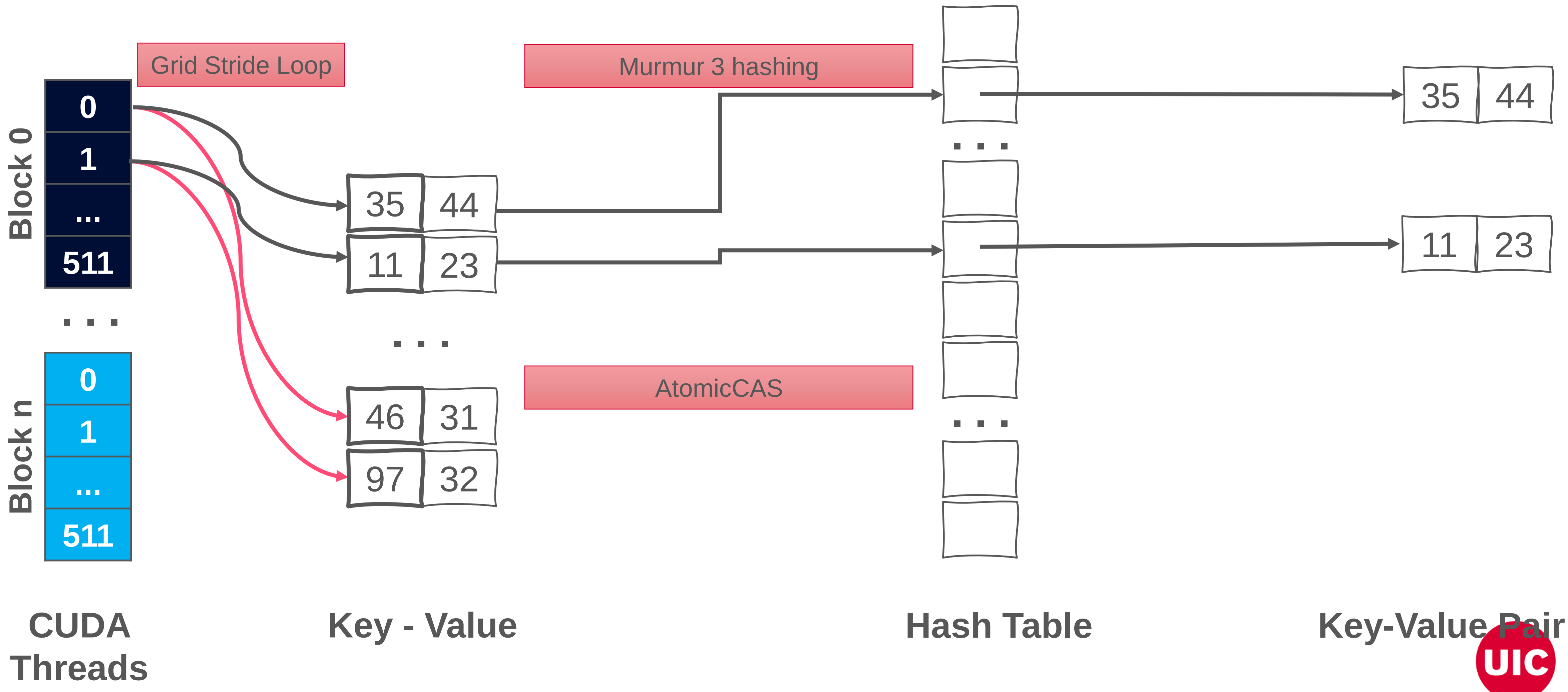
Hash Table (Open Addressing, Linear Probing)



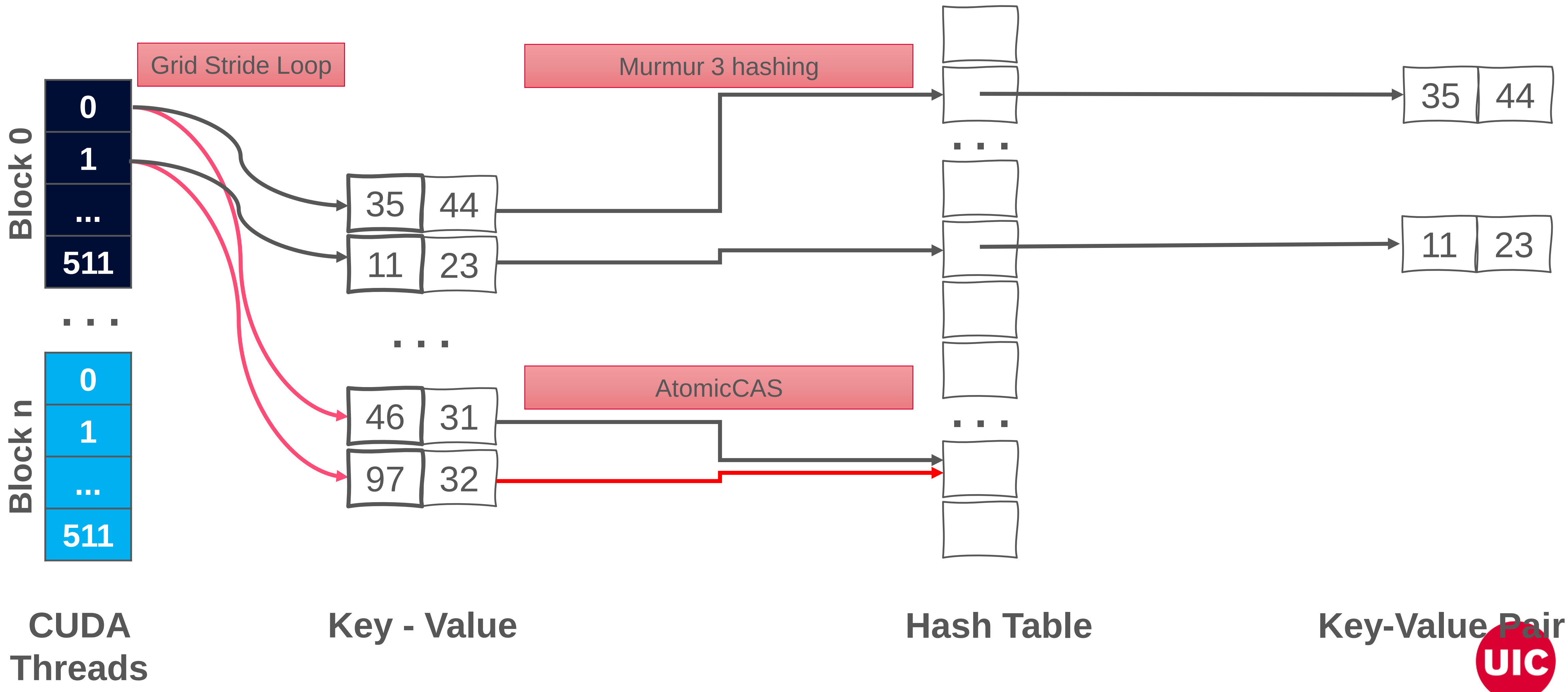
Hash Table (Open Addressing, Linear Probing)



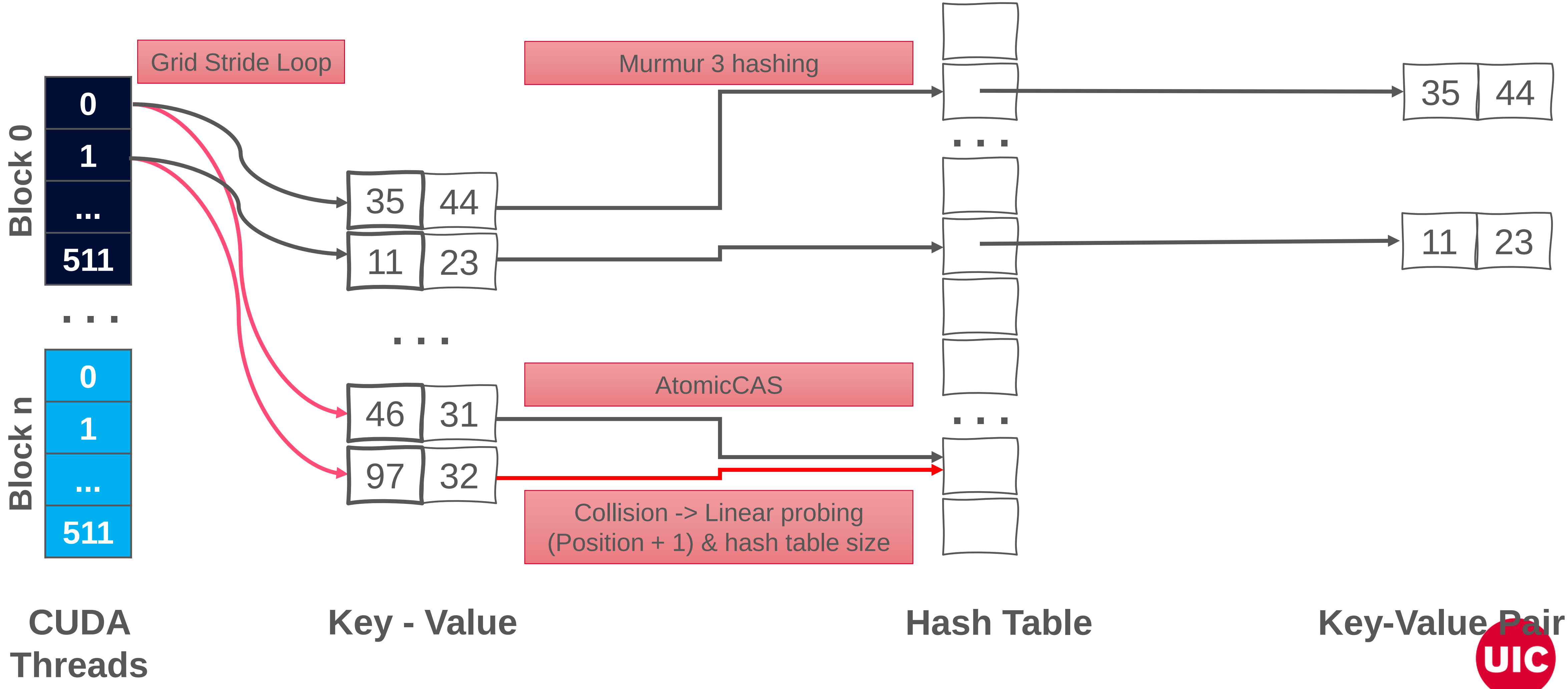
Hash Table (Open Addressing, Linear Probing)



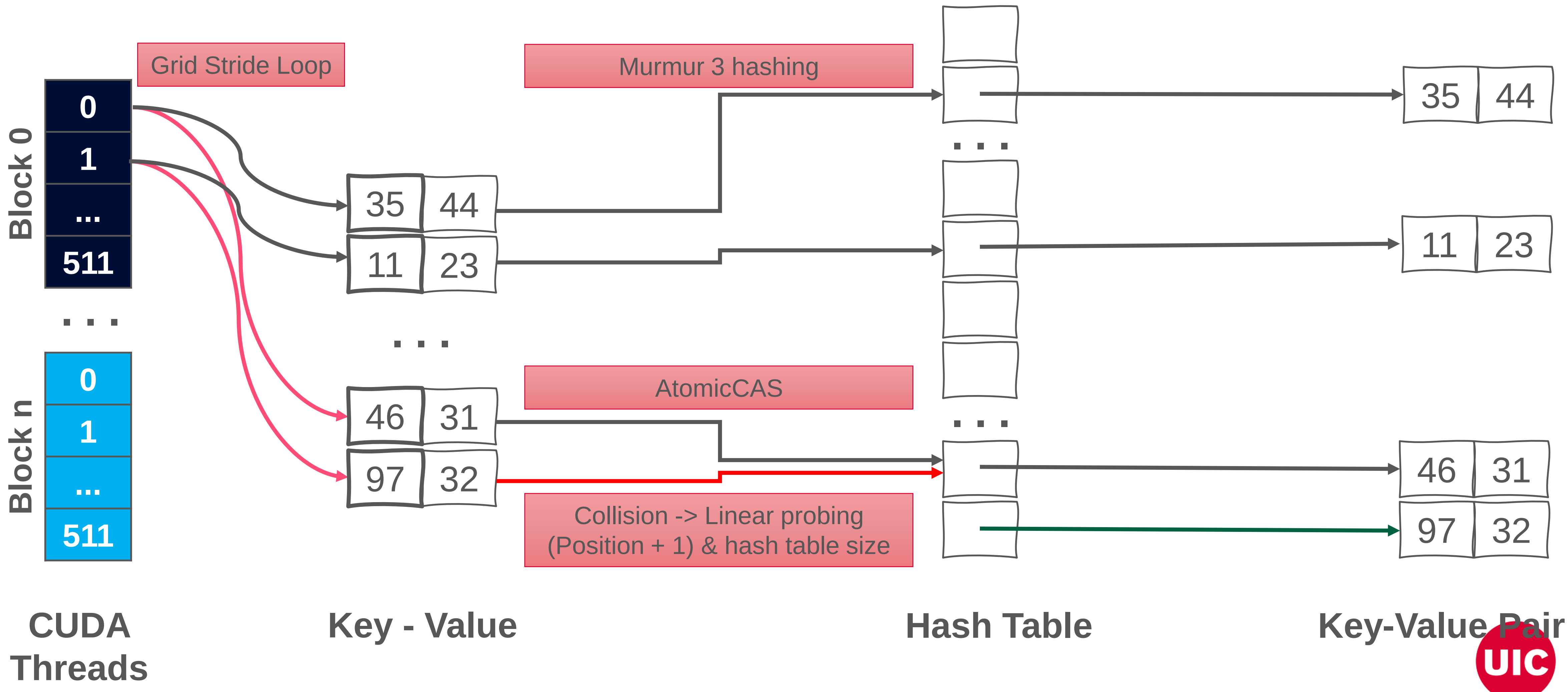
Hash Table (Open Addressing, Linear Probing)



Hash Table (Open Addressing, Linear Probing)



Hash Table (Open Addressing, Linear Probing)



Performing Hash Join on GPU

Static Hash Table

Key	Value

$\bowtie$

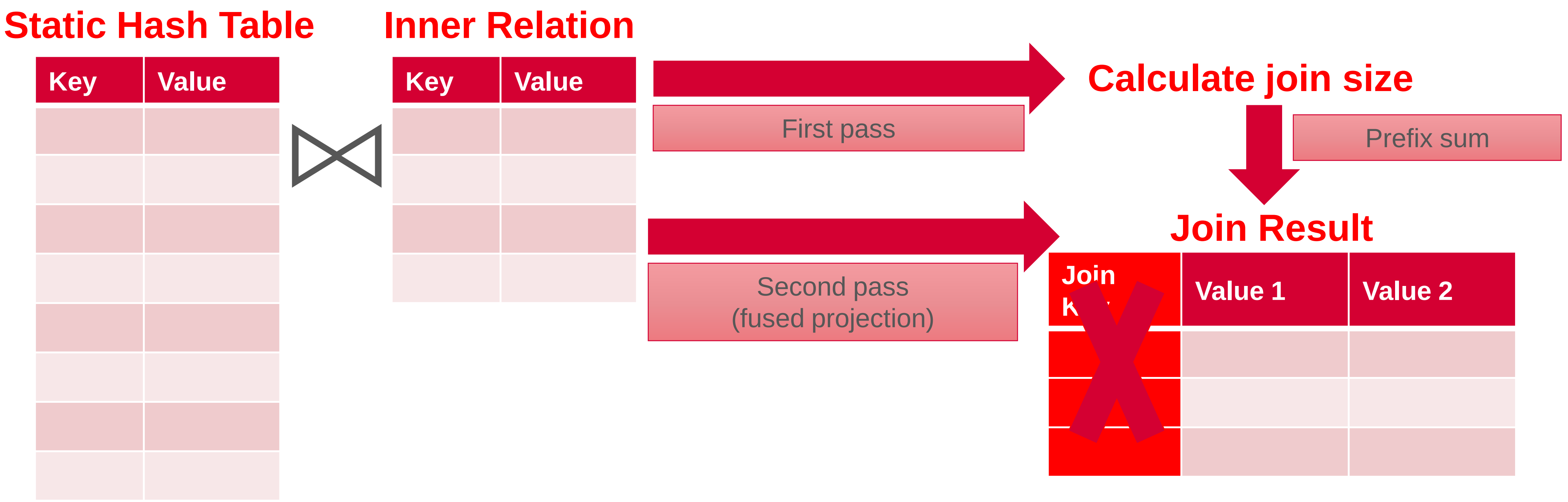
Inner Relation

Key	Value

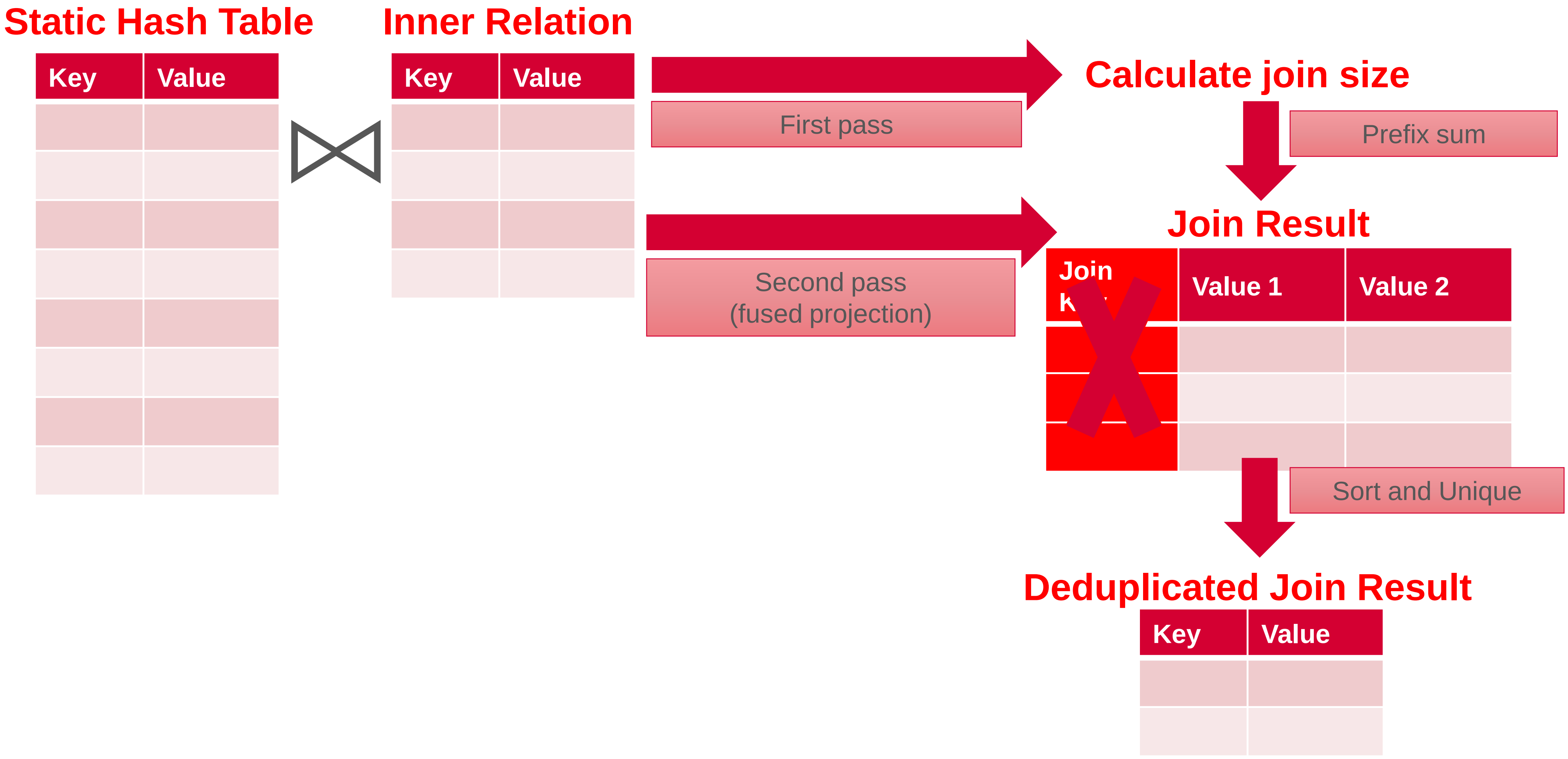
First pass

Calculate join size

Performing Hash Join on GPU



Performing Hash Join on GPU

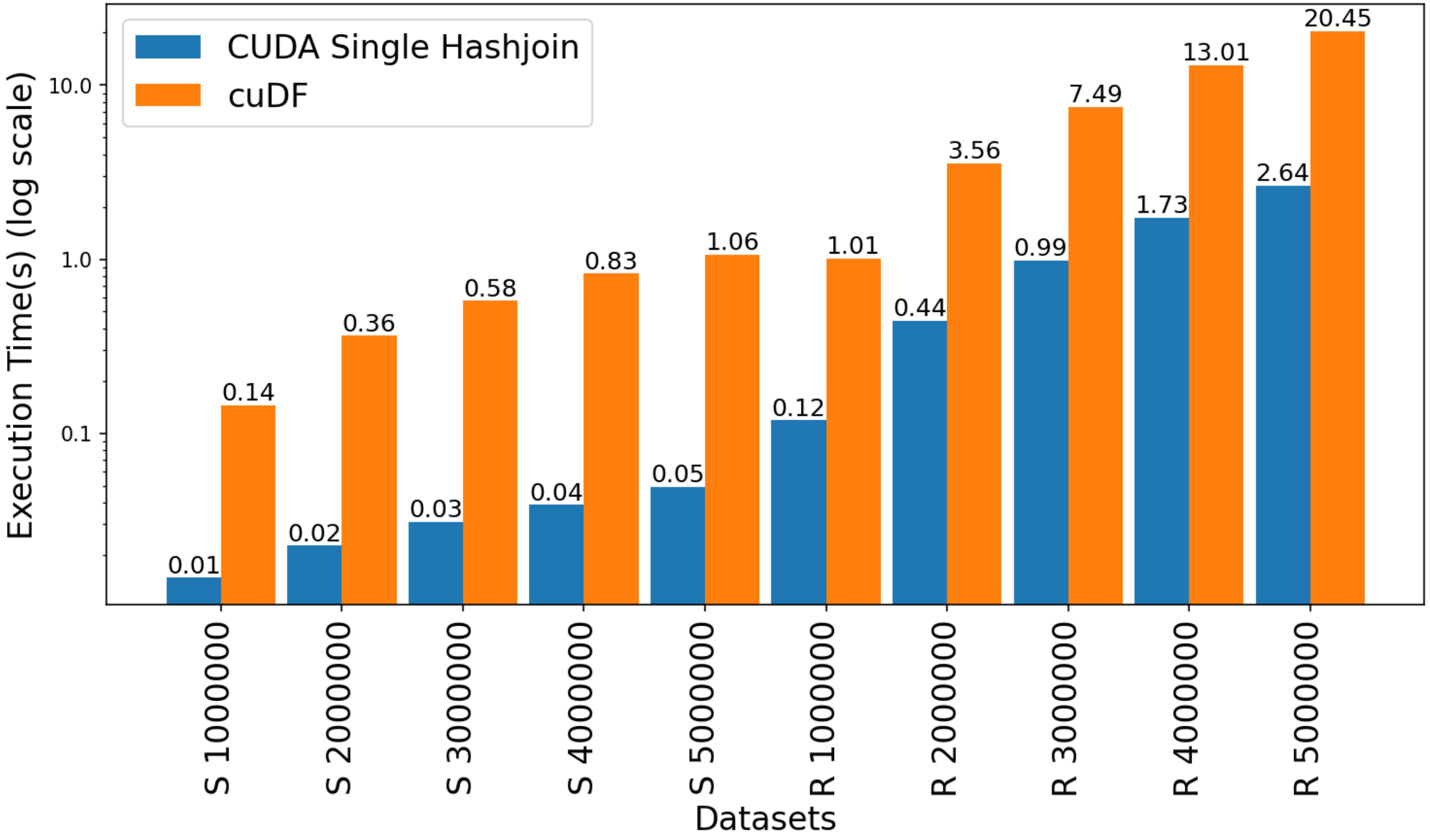


CUDA Advantages over cuDF

- Fuse operations
- Thread-block configuration
- Memory management
- Optimize data structure

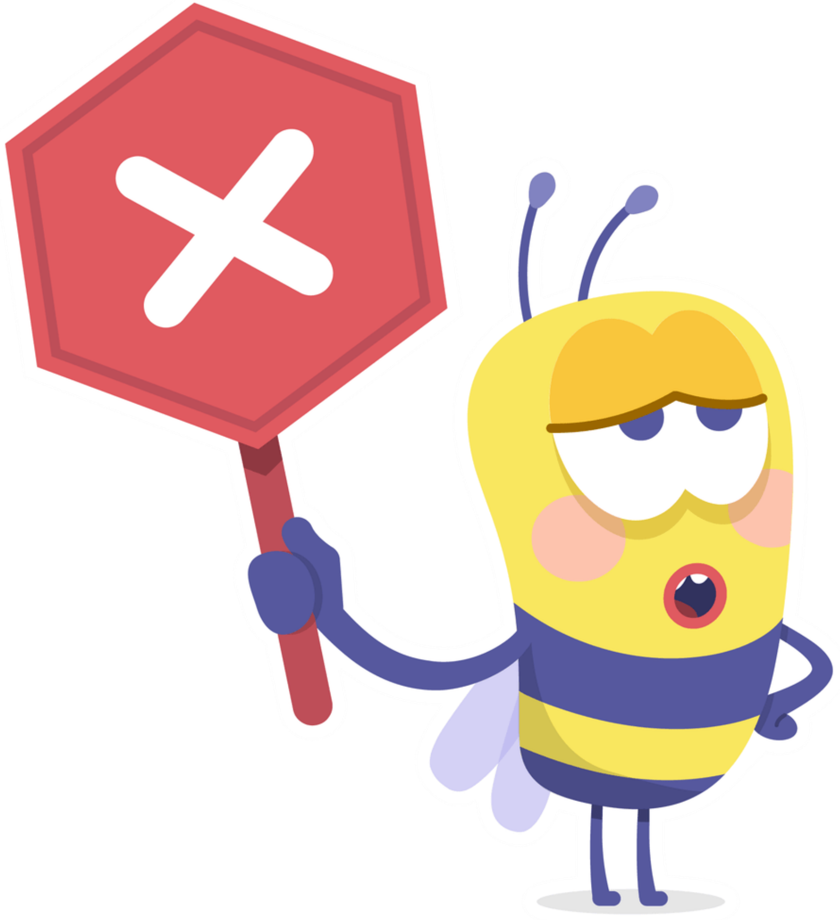
- Jason. Sanders. CUDA by example : an introduction to general-purpose GPU programming. AddisonWesley, Upper Saddle River, NJ, 2011.
- John Cheng, Max Grossman, and Ty McKercher. Professional CUDA c programming. John Wiley & Sons, 2014

Join Performance Comparison: CUDA vs cuDF



- Leadership Computing Facility, A. (2022). Argonne Leadership Computing Facility. Theta GPU Nodes. URL: https://www.alcf.anl.gov/support-center/theta-gpu-nodes

Limitations



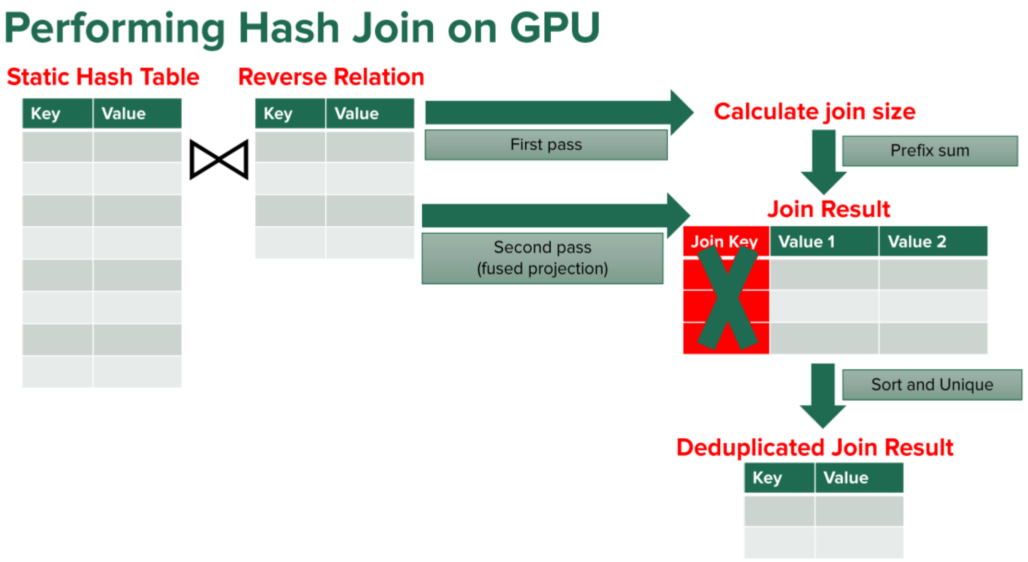
Limited to a single GPU that dictates scaling by available VRAM on the GPU

Memory overflow error for larger graphs

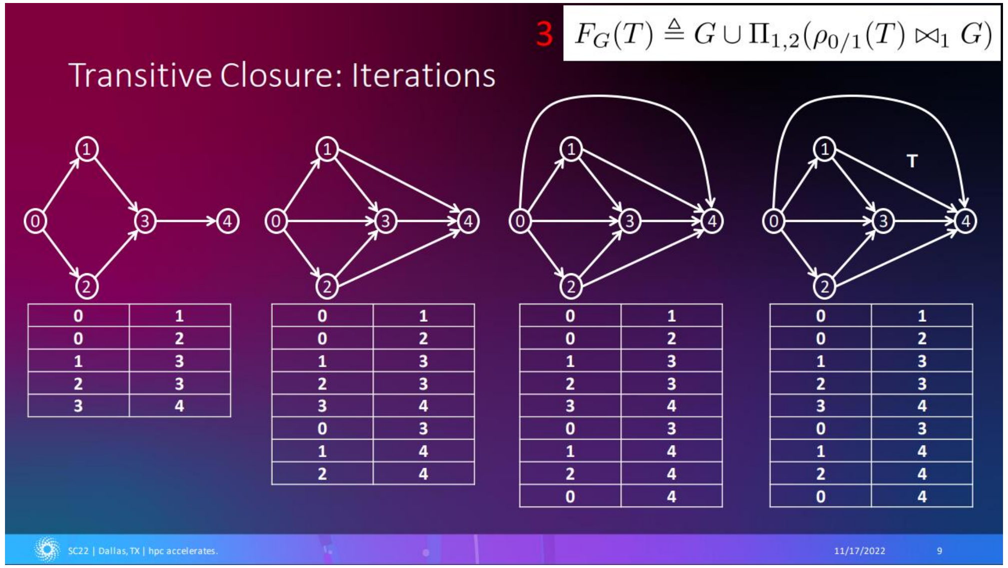
Open addressing based hash table causes memory overhead

Publications

Shovon, A. R., Gilray, T., Micinski, K., & Kumar, S. (2023). Towards iterative relational algebra on the {GPU}. In 2023 USENIX Annual Technical Conference (USENIX ATC 23) (pp. 1009-1016).



Shovon, A. R., Dyken, L. R., Green, O., Gilray, T., & Kumar, S. (2022, November). Accelerating Datalog applications with cuDF. In 2022 IEEE/ACM Workshop on Irregular Applications: Architectures and Algorithms (IA3) (pp. 41-45). IEEE.



Summary

GPU provides performance enhancement

Python based cuDF can be a head start to GPU programming

Low level CUDA has a learning curve but improves performance

Thank You

ashov@uic.edu

UIC