Sort vs. Hash *Join* Revisited for Near-Memory Execution

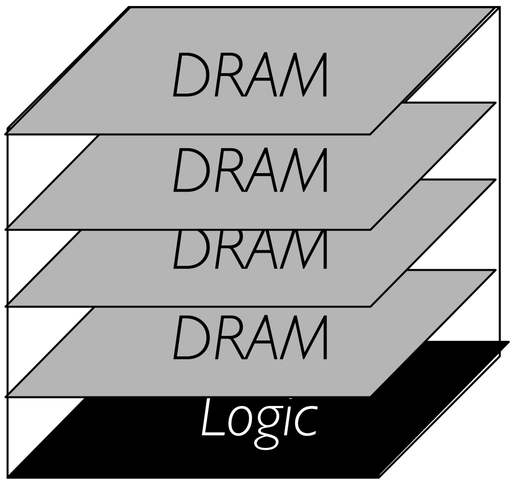
Nooshin Mirzadeh, Onur Kocberber,
Babak Falsafi, Boris Grot

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Near-Memory Processing (NMP)

Emerging technology

- Stacked memory: A logic die w/ a stack of DRAM dies
- Makes near-memory processing practical



Why NMP?

- Less data movement

→ Less energy consumption

- Leverage DRAM's massive internal BW & parallelism

→ High performance

Exploit NMP to accelerate key algorithms

Join Operation

A fundamental operation in database systems

- Main contributor to execution time in analytic DBMSs

Find the matching keys in two tables

Ongoing debate over two main algorithms:

- Hash-based: Current best for CPU execution
  - Cache-optimized
- Sort-based
  - Higher computational complexity
  - But more regular memory access patterns

Revisit sort vs. hash for near-memory execution

Near-Memory Join

Memory access patterns: Key for maximizing NMP efficiency

- Sequential access patterns best exploit DRAM characteristics

Number of accesses is only part of the story

- More sequential accesses better than fewer random accesses

Sort join trumps hash join

- Sequential access pattern + Wide NMP sort logic
  → High efficiency

Sort ~2x better than hash in perf & energy-efficiency

Outline

Overview

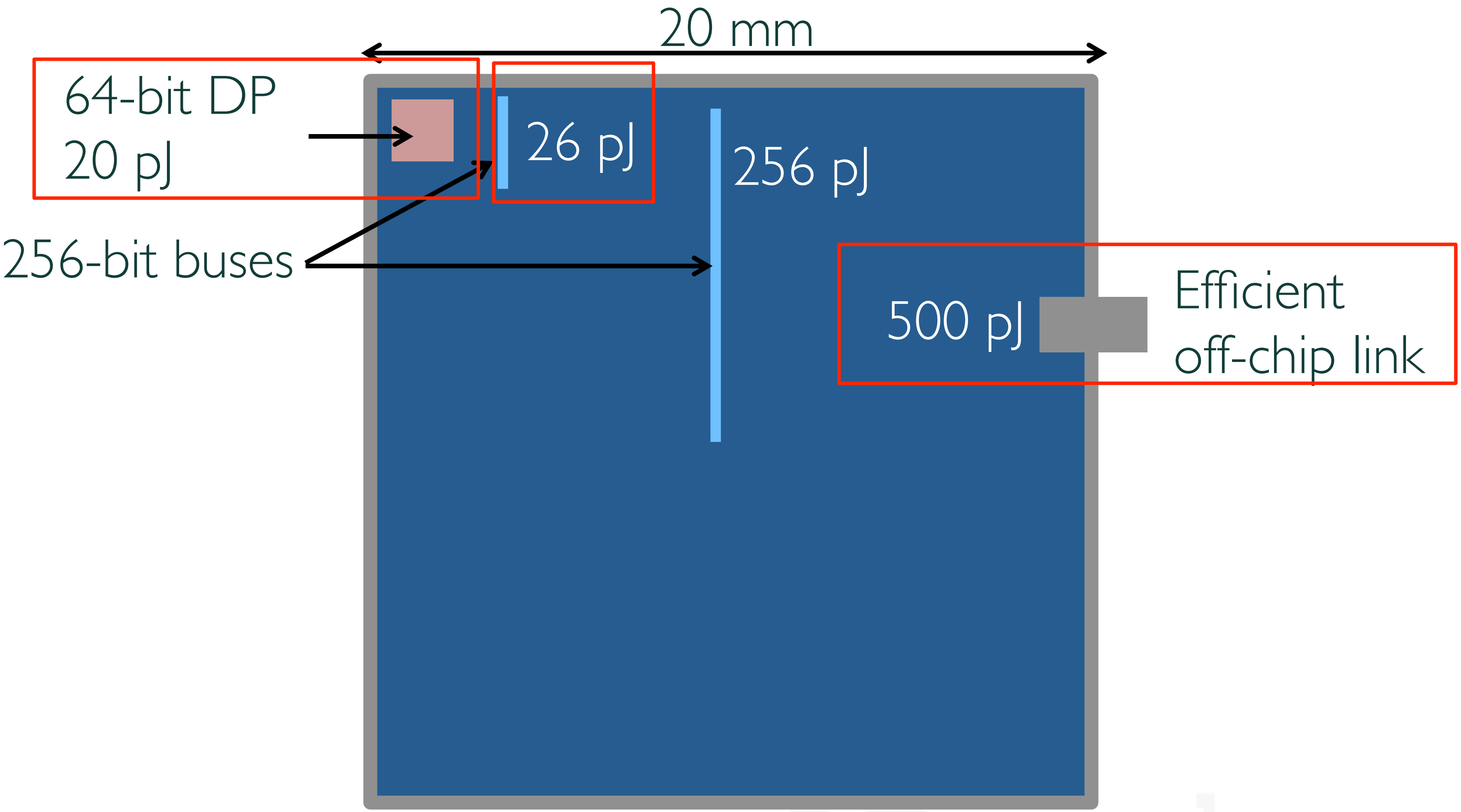
Near-memory processing (NMP)

Join operator

Evaluation

Conclusion

Data Movement and Energy



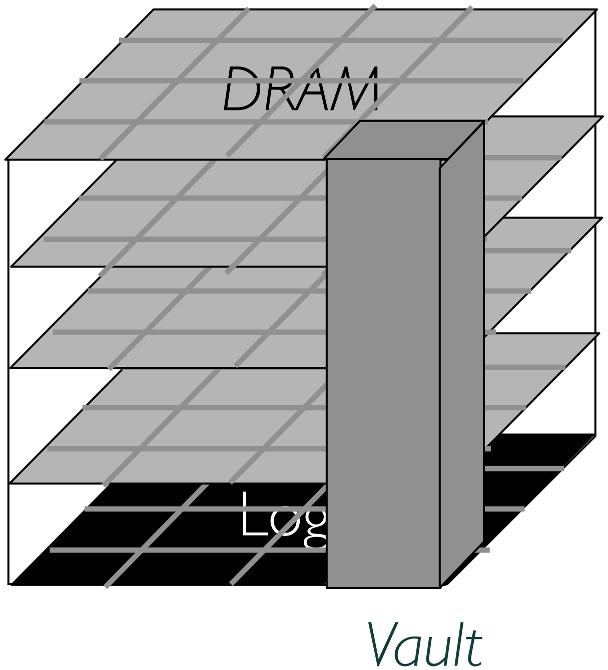
[Dally, SC'14 Panel]

Avoid unnecessary data movement through NMP

Near-Memory Processing (NMP)

Emerging technology: 3D-stacked memory

- Logic die in a stack of DRAM dies
- Through-Silicon Via (TSV)
  - Low energy consumption
- Separated vertical partitions "vaults"
  - Provide a high level of parallelism
- E.g., Micron HMC, AMD HBM

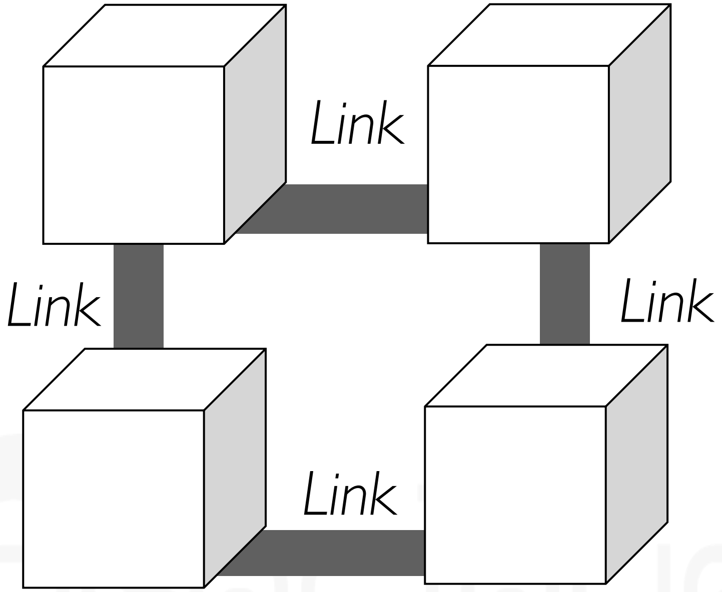


Computation performed next to memory

NMP Realities

Stacked memory: Limited capacity per chip (≤ 8GB)

- High capacity requires multiple chips
- Large datasets require chip-to-chip communication



NMP does not eliminate all data movement!

NMP: Key Aspects

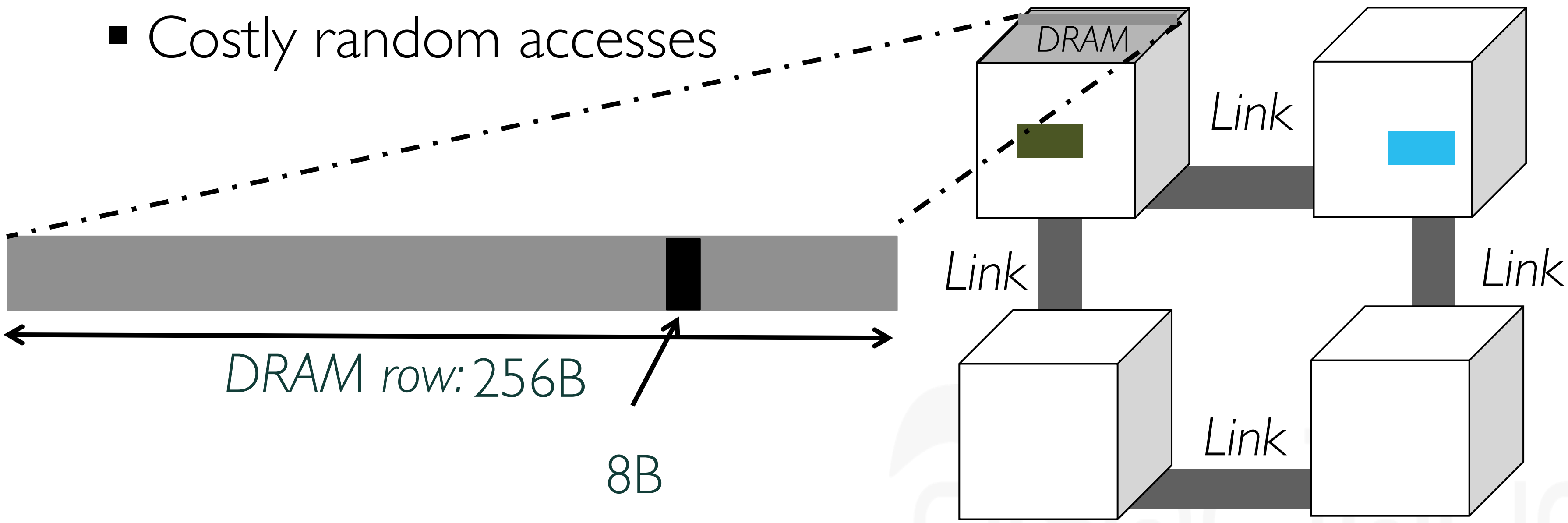
ecocloud an EPFL research center

Chip-to-chip accesses consume more energy per bit

- At least ~2x more than intra-vault accesses
- Must minimize chip-to-chip accesses for efficiency

DRAM implies wide interface and destructive accesses

- Costly random accesses



NMP algorithms must consider access pattern & locality

Outline

Overview

Near-memory processing (NMP)

Join operator

Evaluation

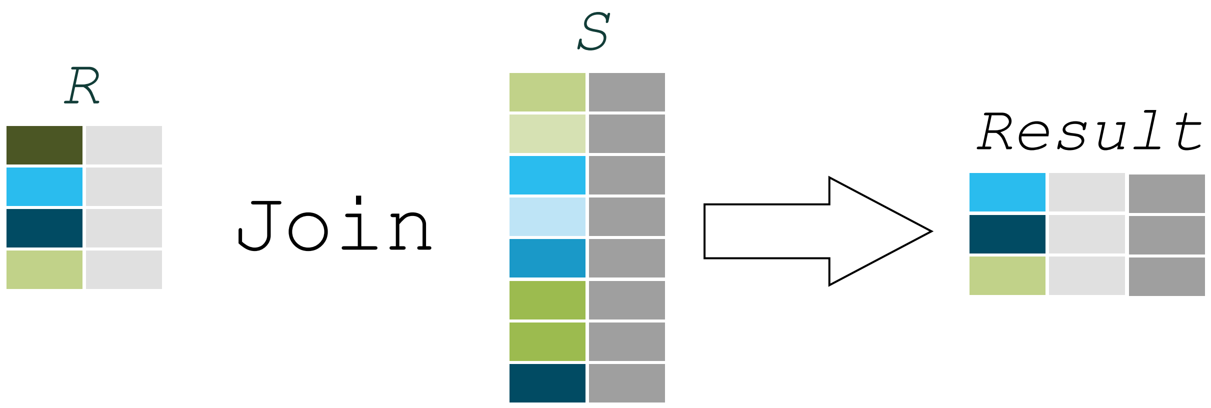
Conclusion

What is a Join?

Iterates over a pair of tables

Finds the matching keys in two tables

```
Q: SELECT ... FROM R, S WHERE R.Key = S.Key
```



Hash vs. Sort *Join*

Hash-based algorithms: build and probe a hash table

Algorithm: Radix-Hash Join [Manegold et al., 2002]

- ✓ Lower computational complexity: *O(n)*
- ✗ Random memory accesses

Sort-based algorithms: sort and merge the two tables

Algorithm: Partitioned Massively Parallel Sort-Merge (P-MPSM)

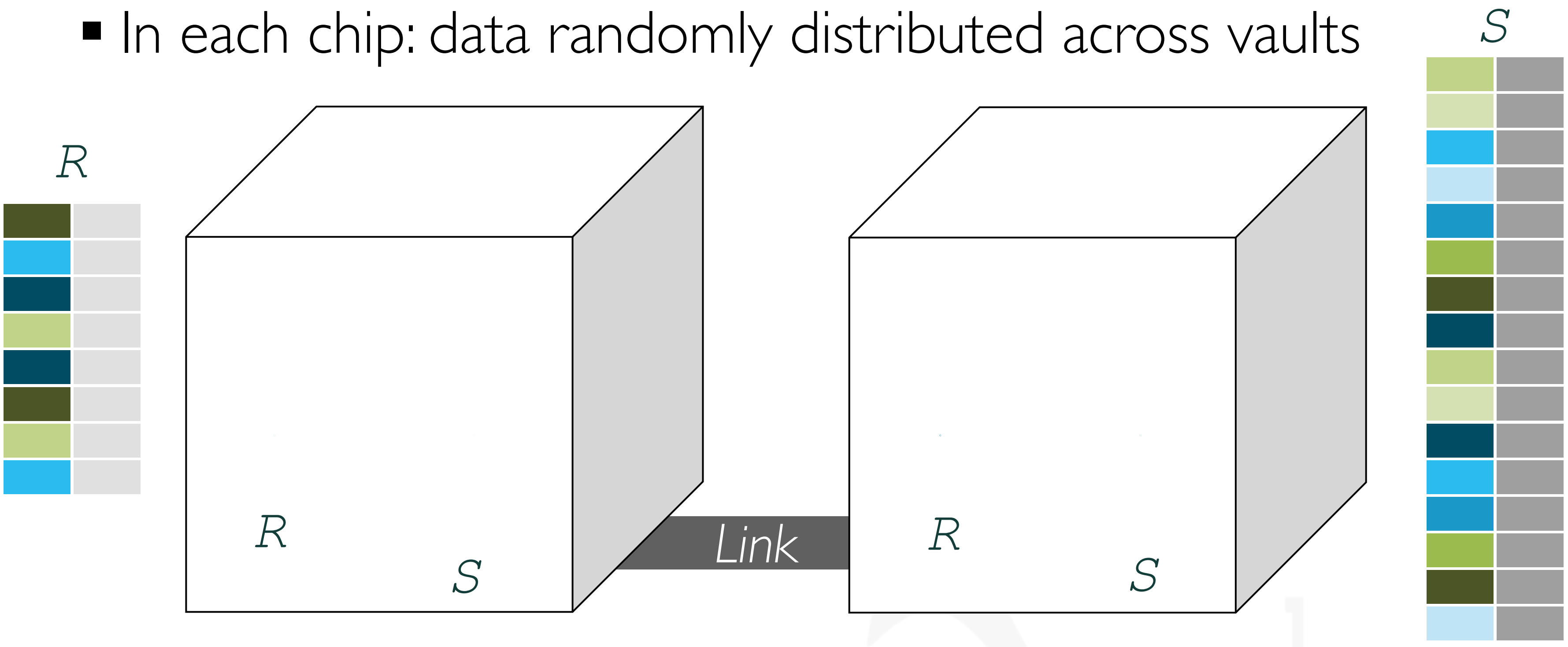
- ✗ Higher computational complexity: *O(nlogn)* [Albutiu et al., 2012]
- ✓ Sequential memory access

Computational complexity vs. memory access patterns

NMP: Data Distribution

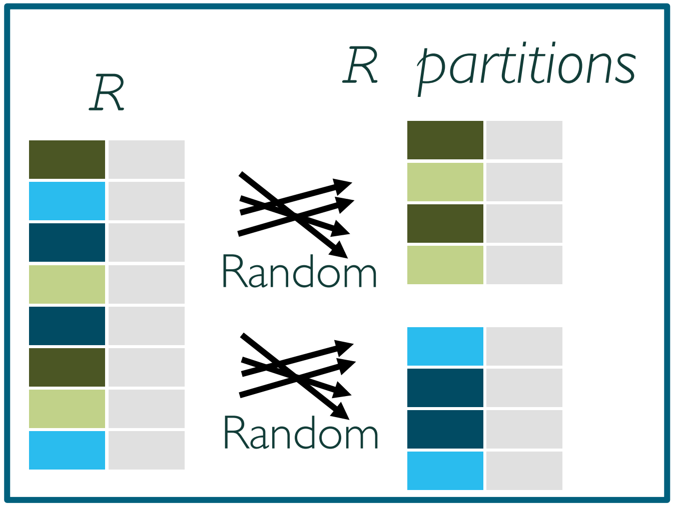
Data randomly distributed across memory chips

- Data cannot fit in one chip
- In each chip: data randomly distributed across vaults



Hash Join

1. Partitioning phase: Partition two tables based on the keys
   - CPU-centric: exploit locality in caches
   - NMP: high locality in a vault
     - # of partitions = # of vaults



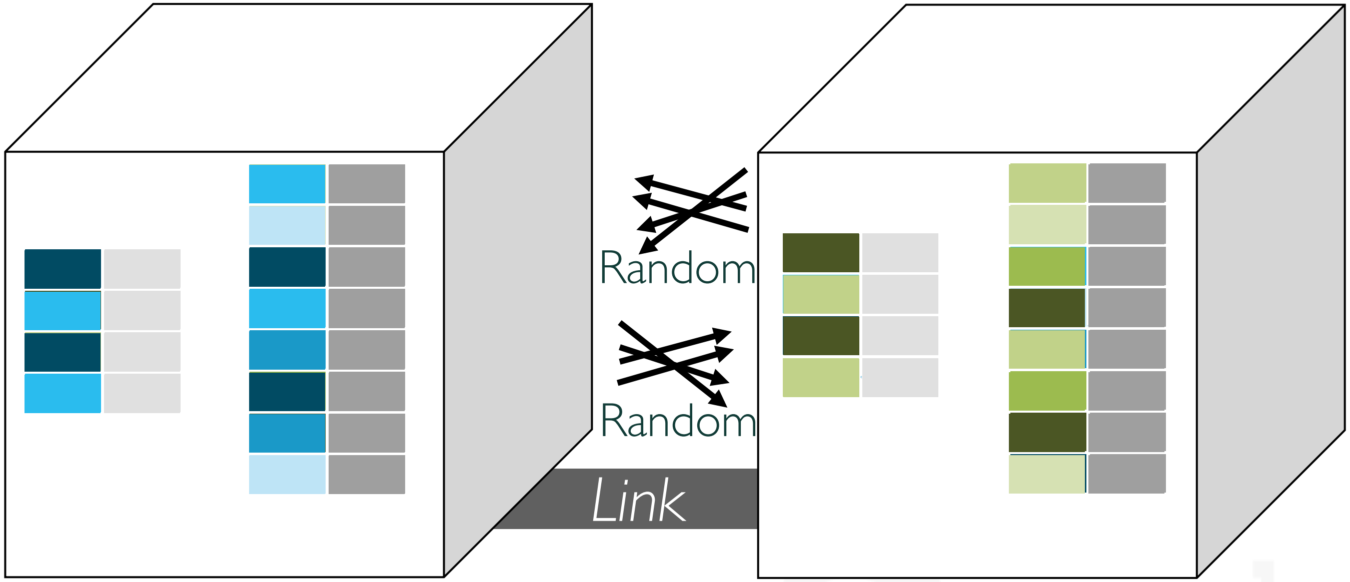
Partitioning

Partitioning

Near Memory Hash Join

1. Partitioning phase

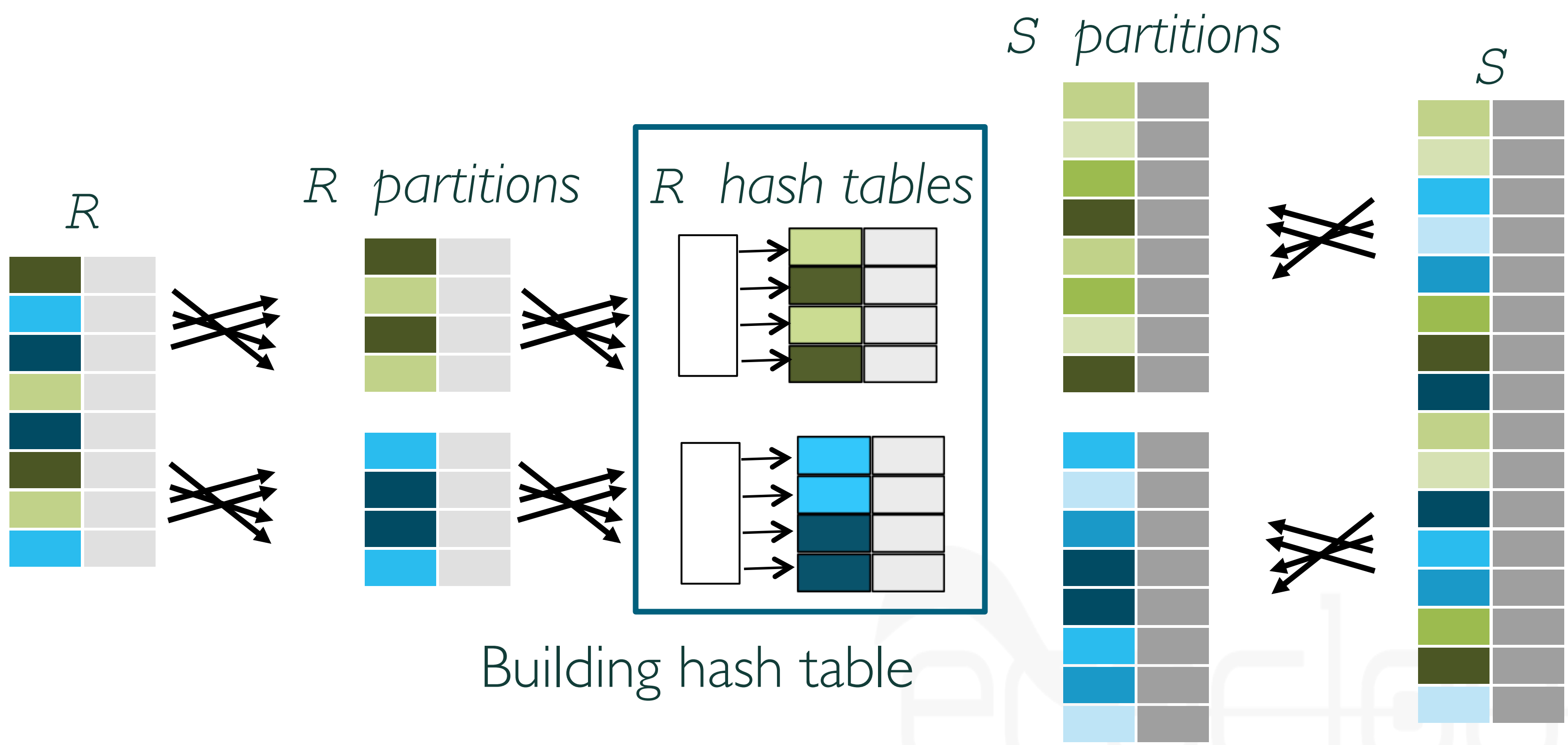
Random access patterns: both R and S
Low access locality: both R and S



Costly *random* access patterns and *low* access locality

Hash Join

2. Build phase: Build a hash table on R
   - Nearly constant look-up in the next phase



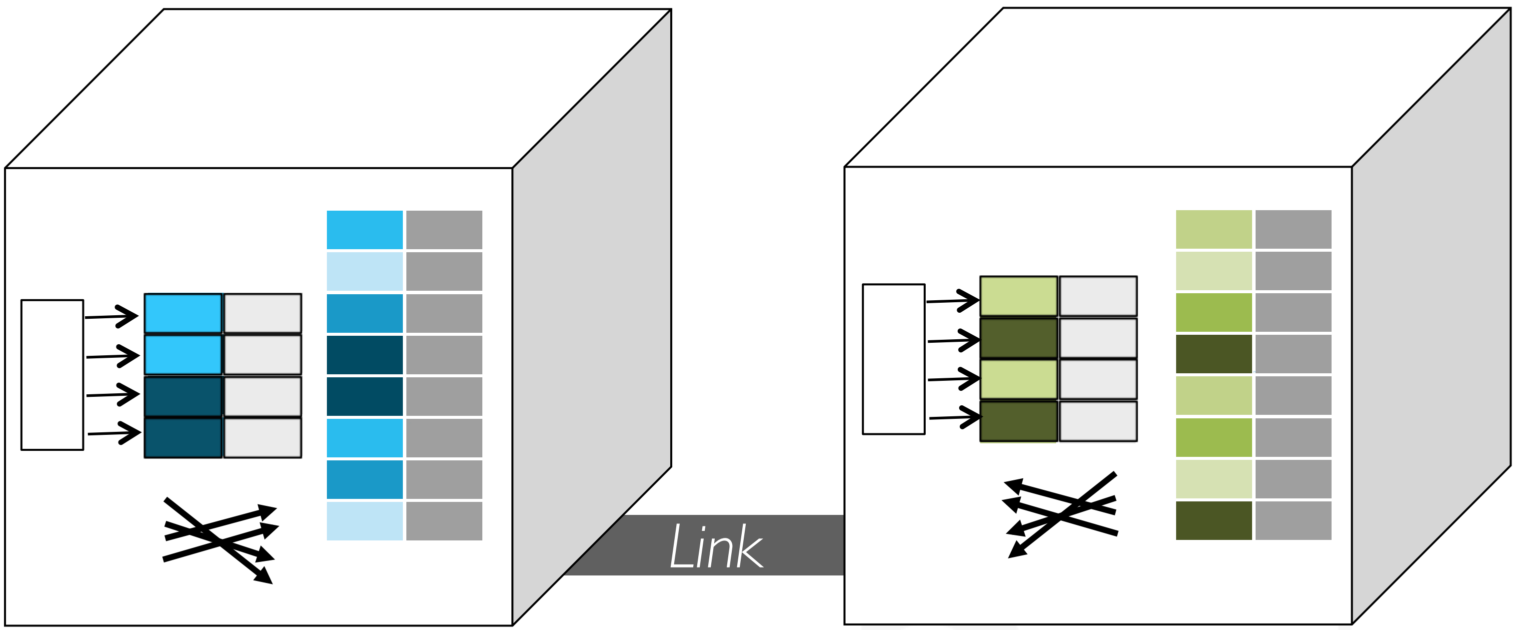
Building hash table

Near Memory Hash Join

ecocloud
an EPFL research center

2. Build phase: High locality

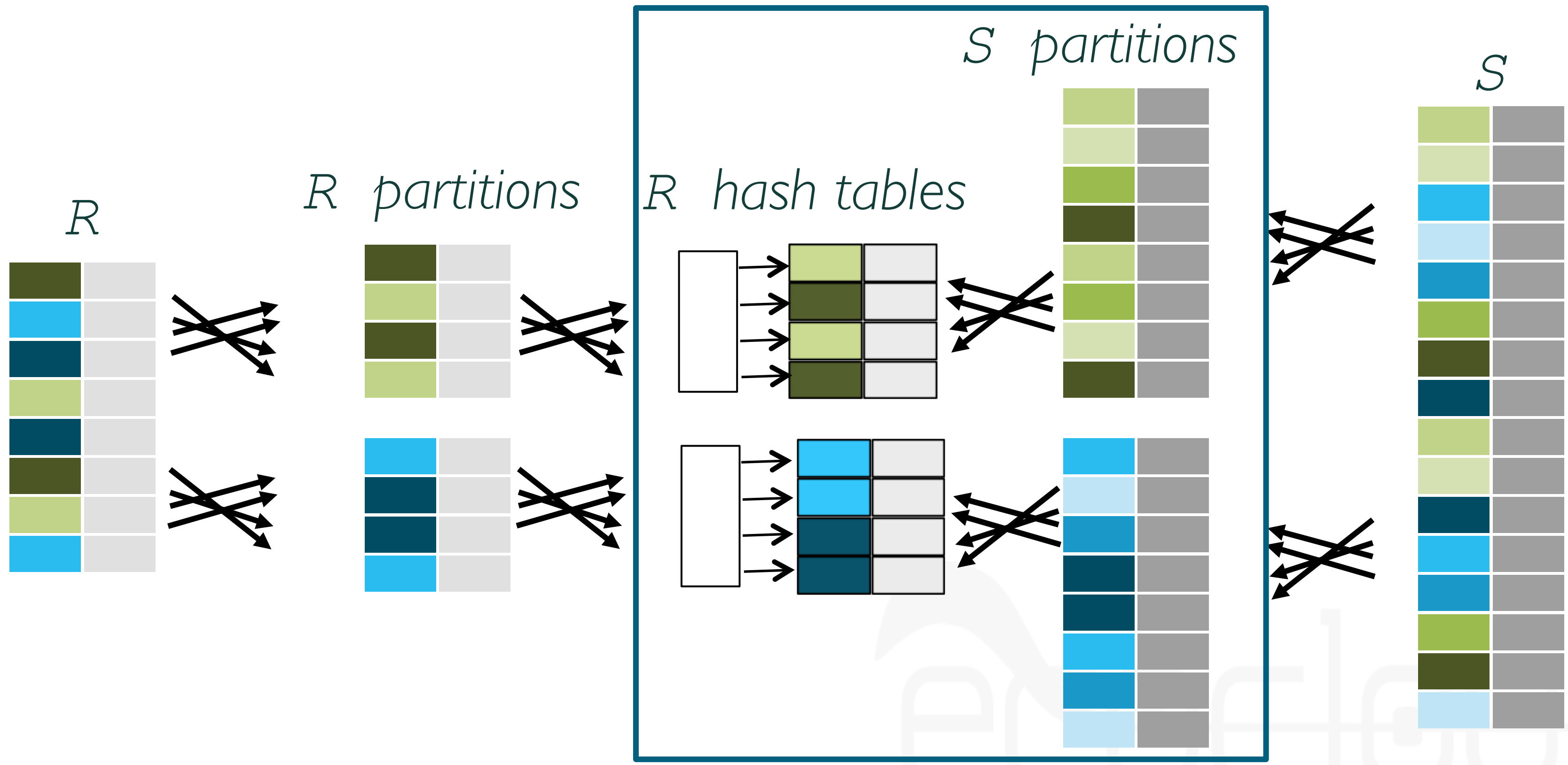
Random access pattern: building R's hash tables



Costly *random* access pattern

Hash Join

3. Probe phase: Probe the hash table with the other column
   - Scan S and look up the keys in R's hash table

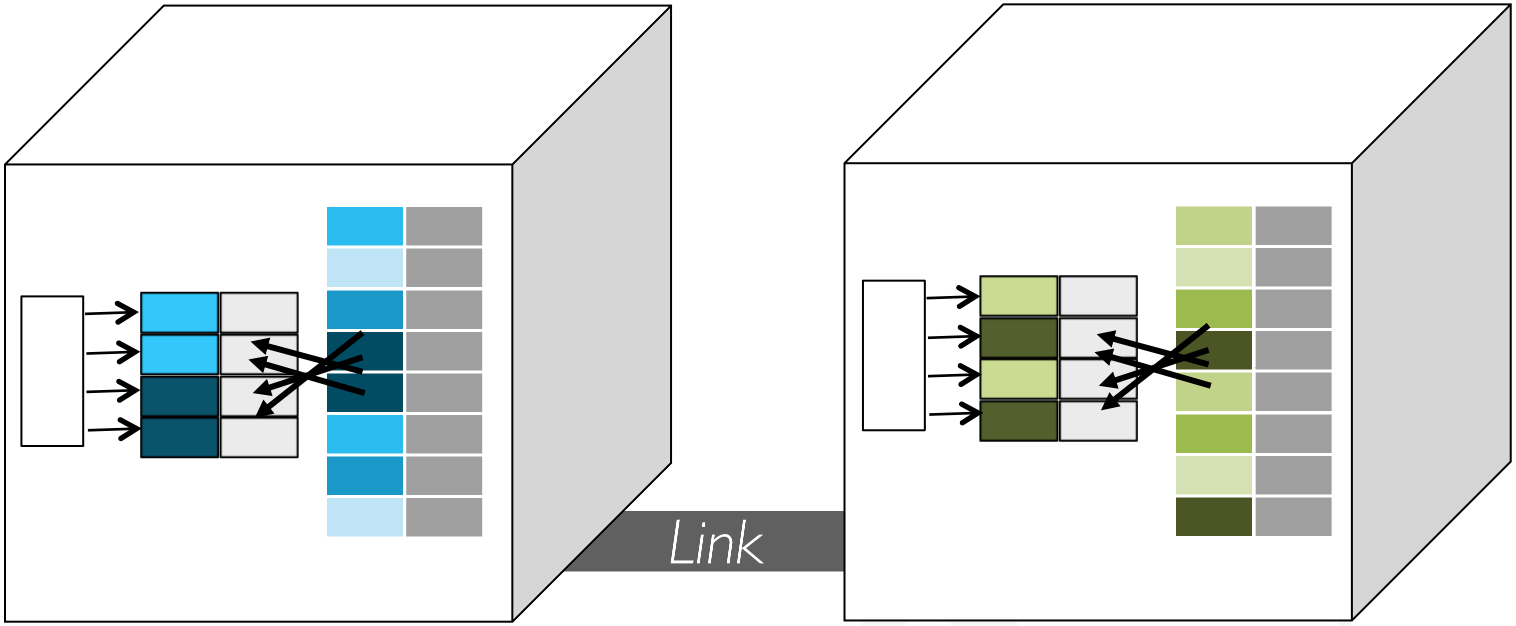


Probing the hash tables

Near Memory Hash Join

3. Probe phase: High locality

Random access pattern: R's hash table



Costly *random* access pattern

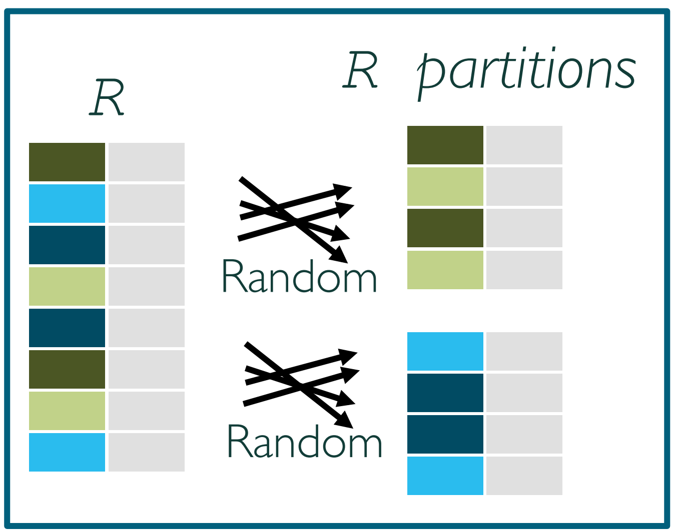
Hash Join: Summary

Phases	Hash
1. Partitioning	☹
2. Build	☹
3. Probe	☹

☹: Random access pattern (local or remote)

Sort Join

1. Partitioning phase: Partition R table based on the keys
   - Helps reducing merge-join time

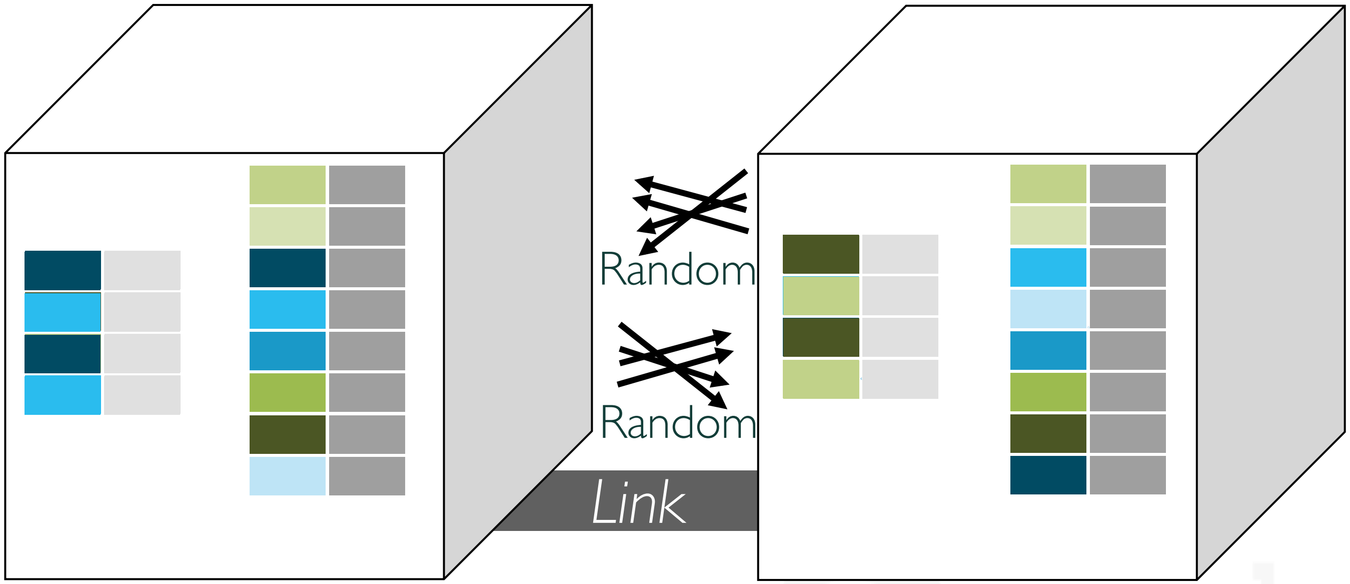


Partitioning

Near Memory Sort Join

1. Partitioning phase

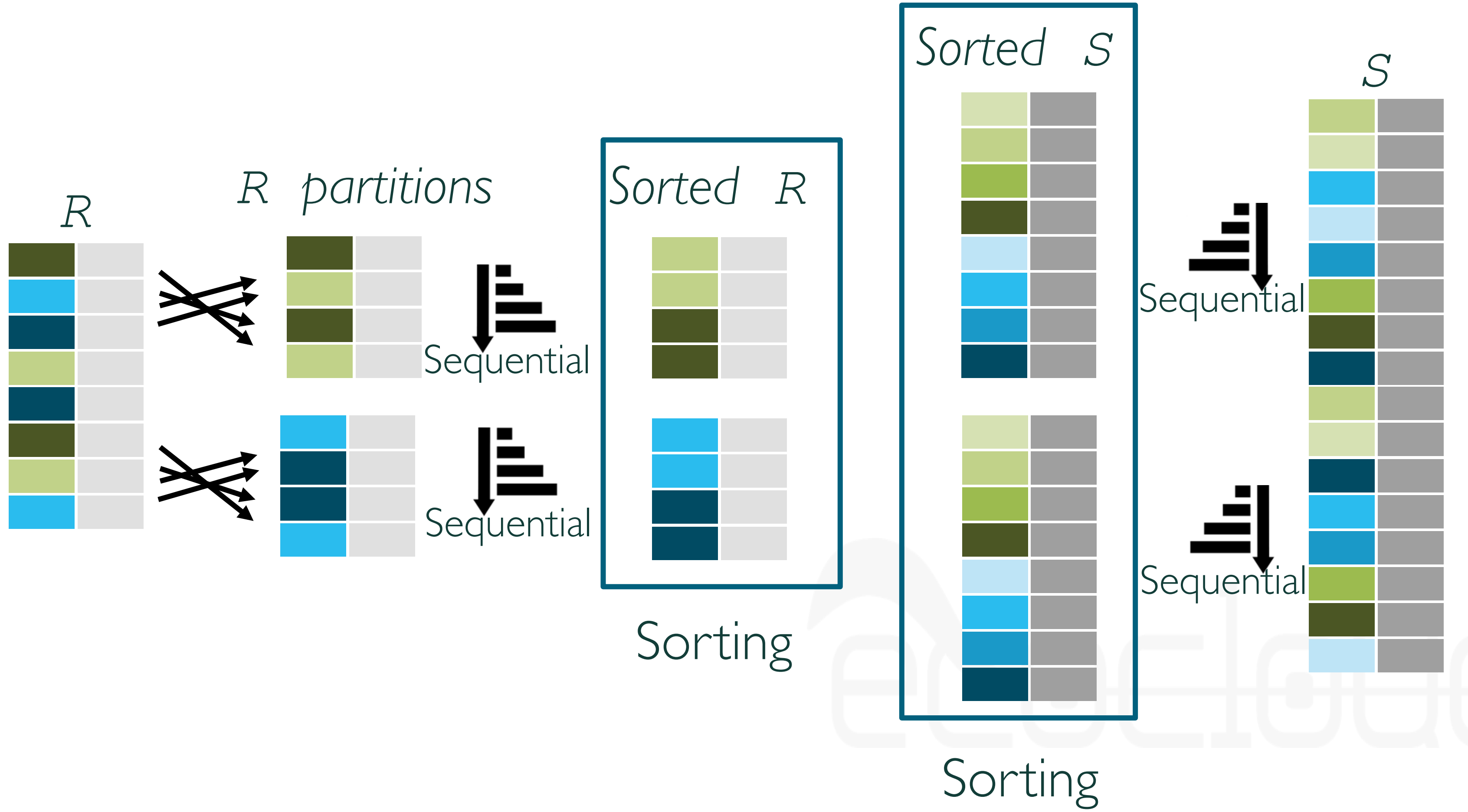
Random access pattern: only R
Low locality: only R



Costly *random* access pattern and *low* locality

Sort Join

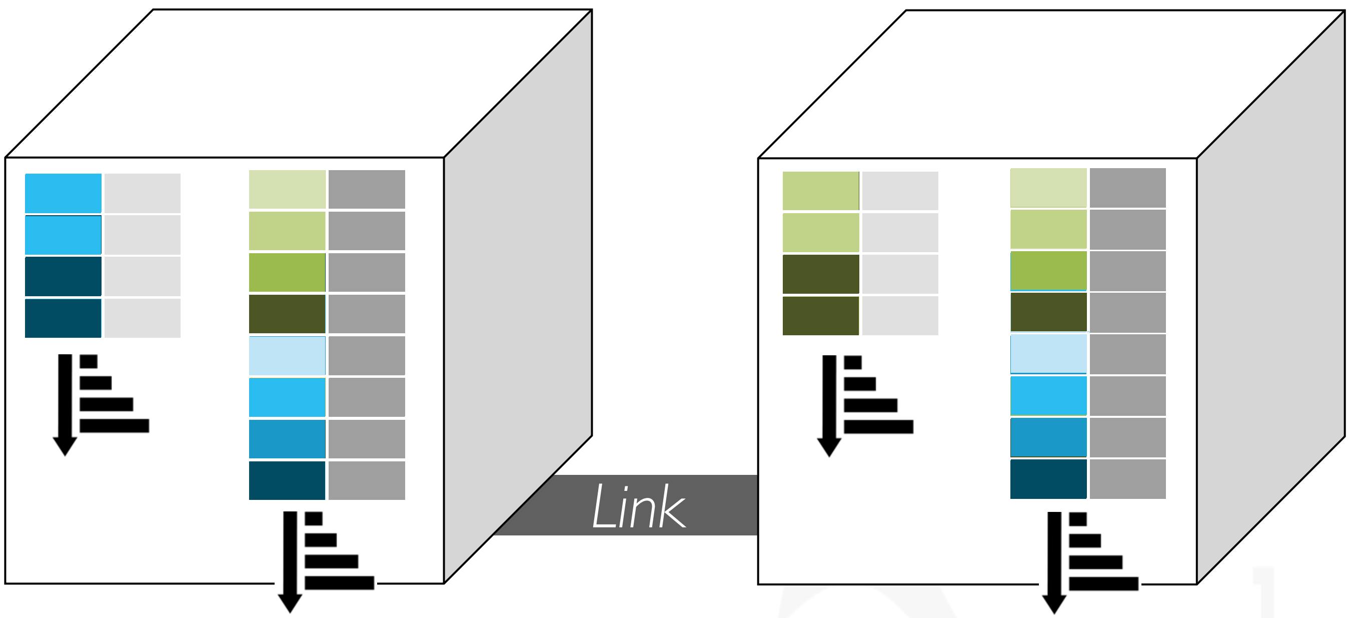
2. Sort phase: Sort both tables
    - Allows linear-time and sequential merge-join



Near Memory Sort Join

2. Sort phase: High locality

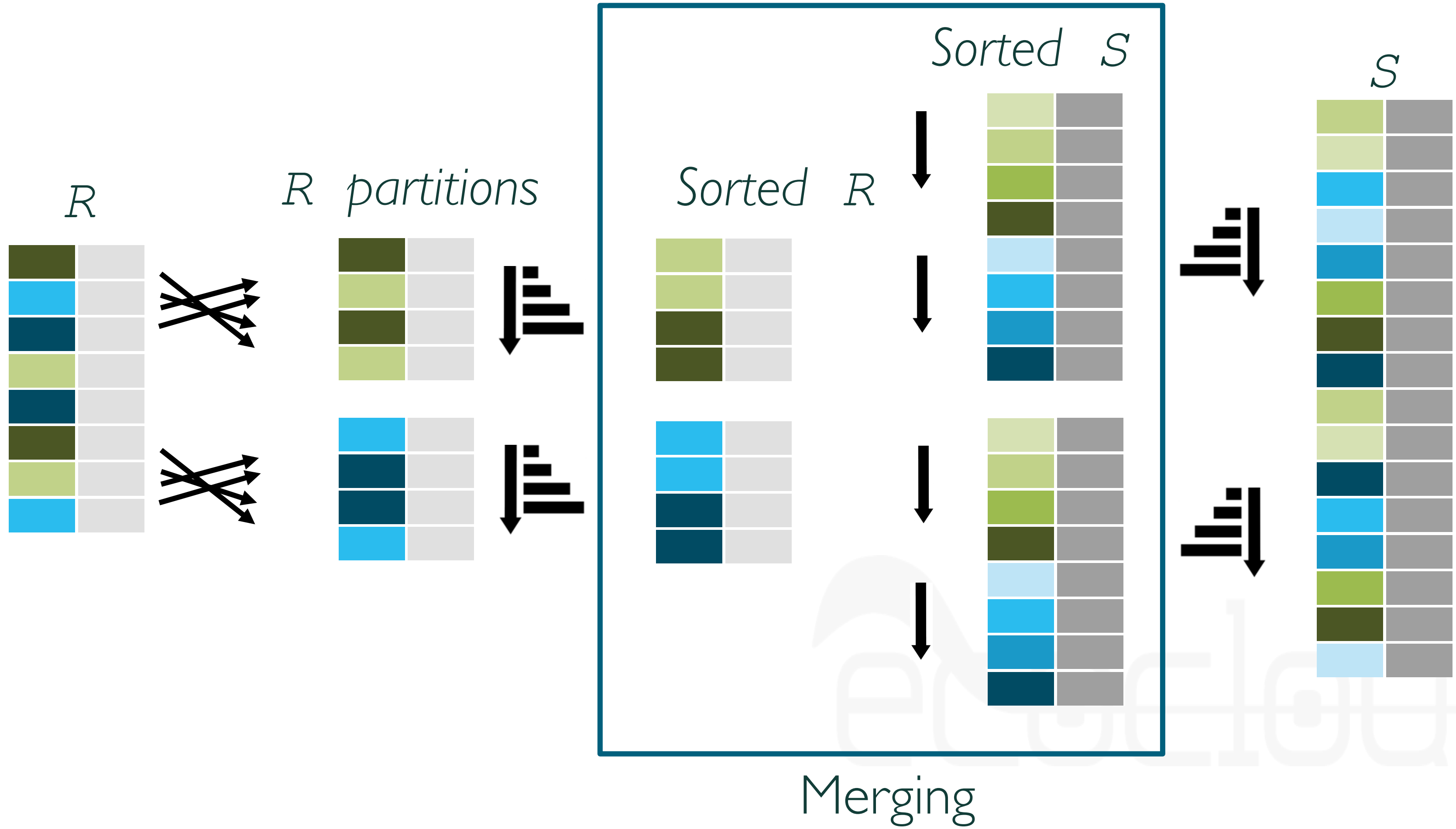
Sequential access pattern



Sequential access pattern and *high* locality

Sort Join

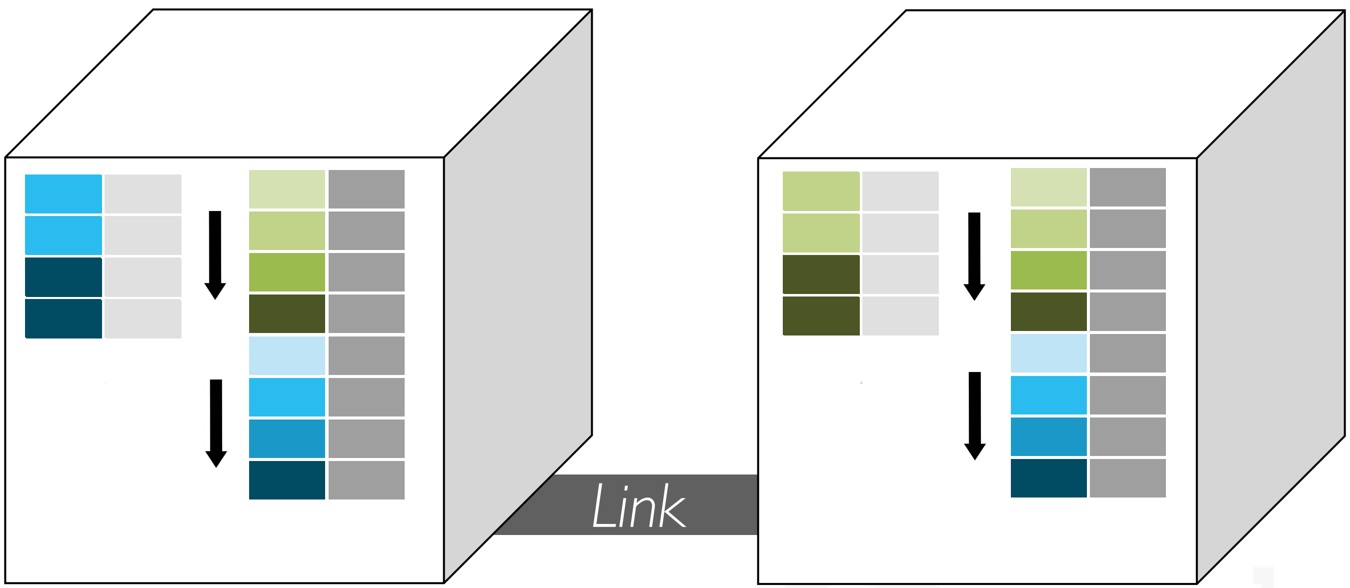
3. Merge phase: Merge-join R wtih S
   - Access data sequentially



Near Memory Sort Join

3. Merge phase: sequential access pattern
   - Stream each chunk of R and S

Low locality: only R



Low locality: one table

Hash vs. Sort Join: Summary

Phases	Hash	Sort
1. Partitioning	☹	☹
2. Build / Sort	☹	☺
3. Probe / Merge	☹	😐

☹: Random accesses (local or remote)

😐: Sequential accesses (remote)

☺: Sequential accesses (local)

Outline

- Overview
- Near-memory processing (NMP)
- Join operator
- Evaluation
- Conclusion

Methodology

First order performance and energy model:

CMP Feature
- 22nm, 16 cores

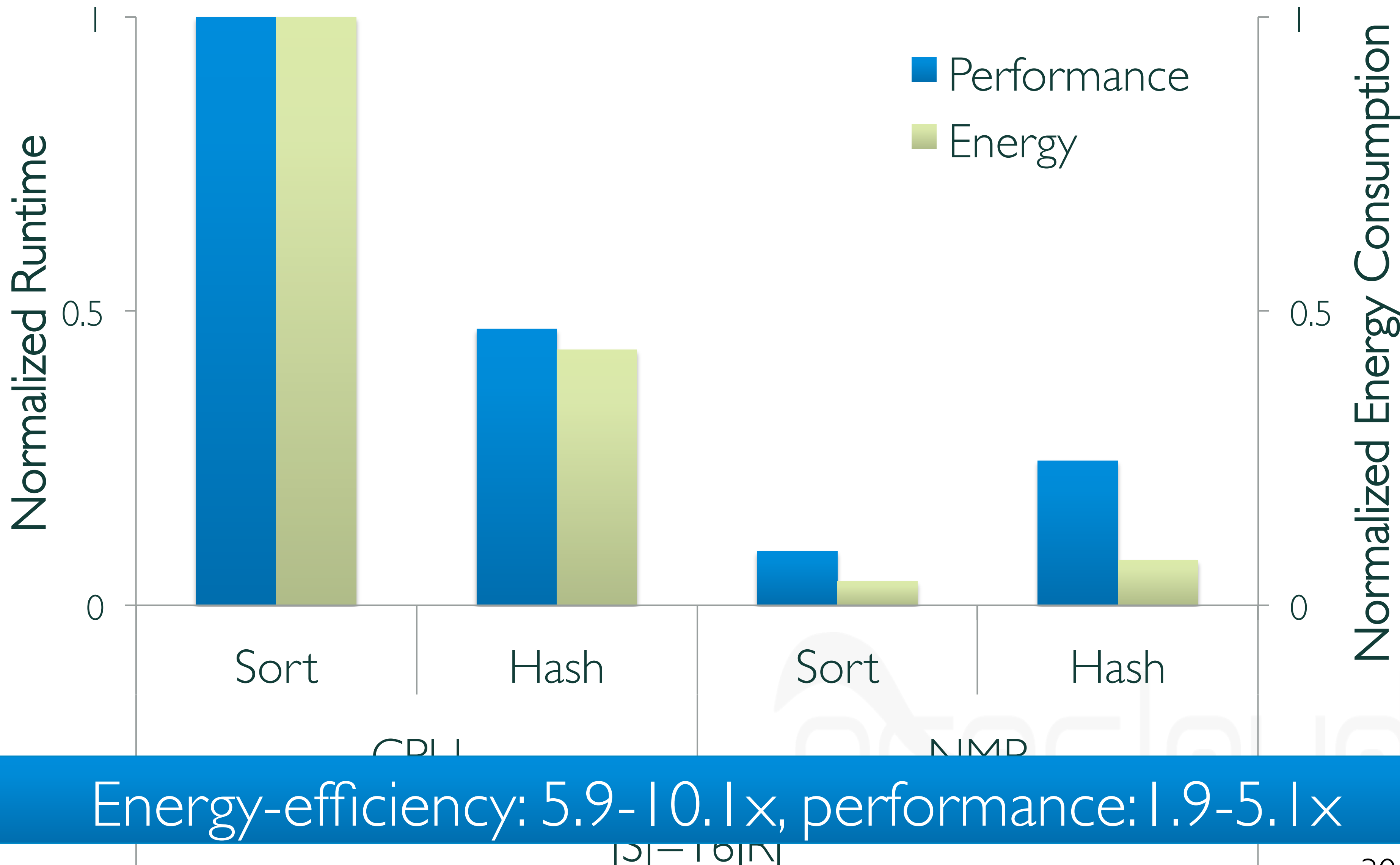
Core
- OoO, 3-wide, 2.5 GHz
- 512-bit SIMD
- 64KB L1-I/D, 64B block

LLC
- 4MB, 16-way

HMC
- 4 cubes, ring topology
- 8GB per cube
- 32 vaults per cube
- 4 links per cube

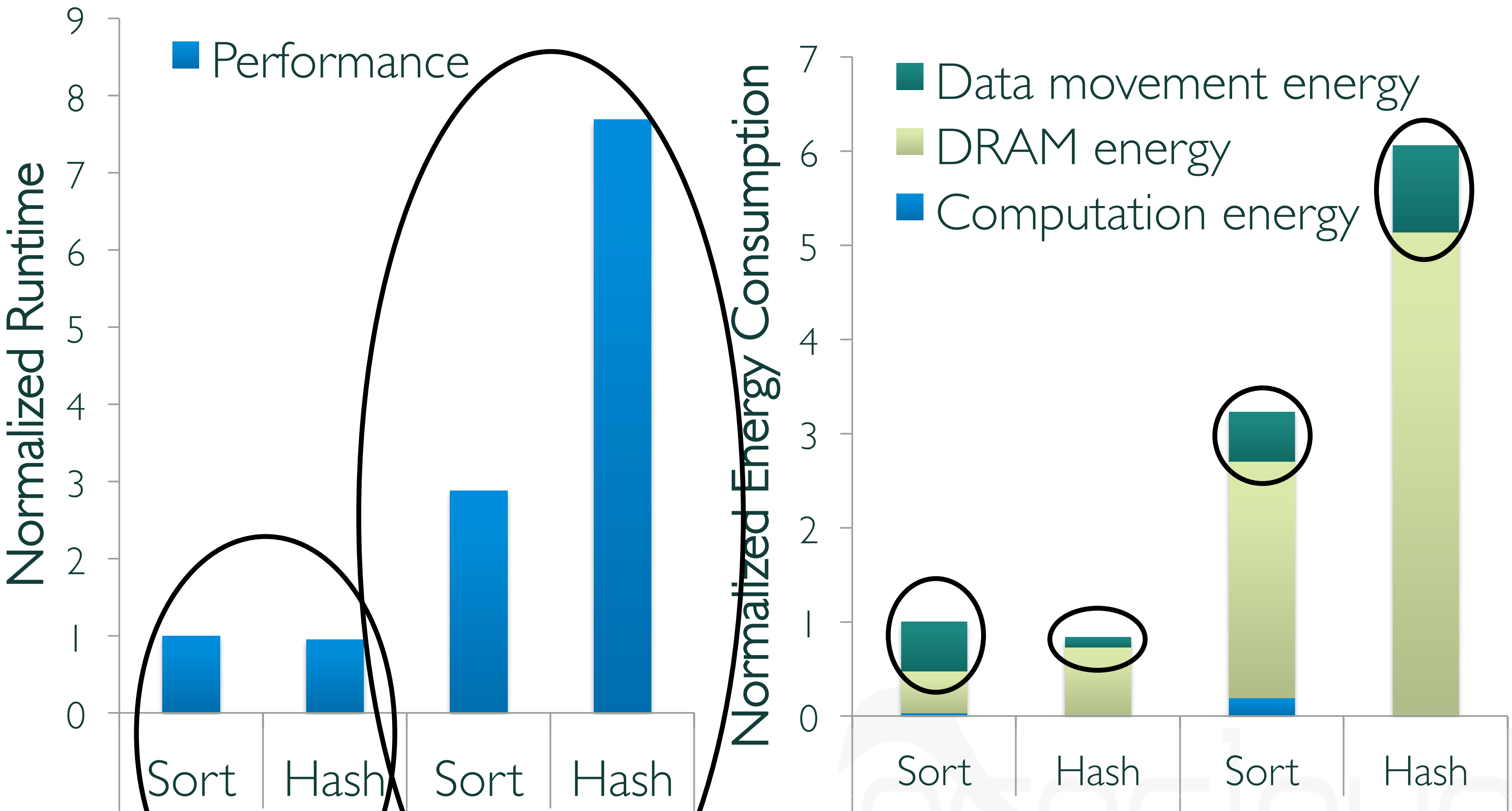
Join Logic
- 22nm logic die
- 256B SIMD
- 2D Mesh NoC

Performance and Energy



Energy-efficiency: 5.9-10.1x, performance: 1.9-5.1x

NMP: Hash vs. Sort



Sort-Join is more efficient when |S| > |R|

Conclusion

NMP improves both performance & energy efficiency

- Exploits internal DRAM bandwidth and parallelism
- Reduces data movement

NMP algorithms must consider memory access patterns

- Sequential accesses best leverage DRAM characteristics
- Intra-chip accesses minimize data movement

Locality + Sequential access patterns
→ Sort join more efficient for NMP

- Hash join still best for CPU

Thanks!

Question?