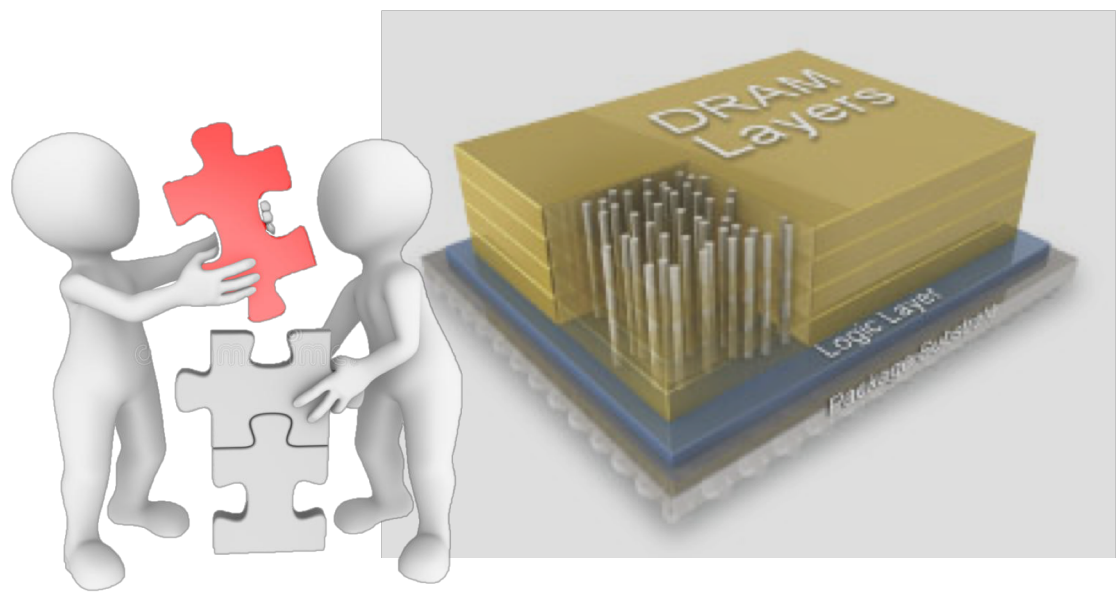
THE UNIVERSITY of EDINBURGH
informatics

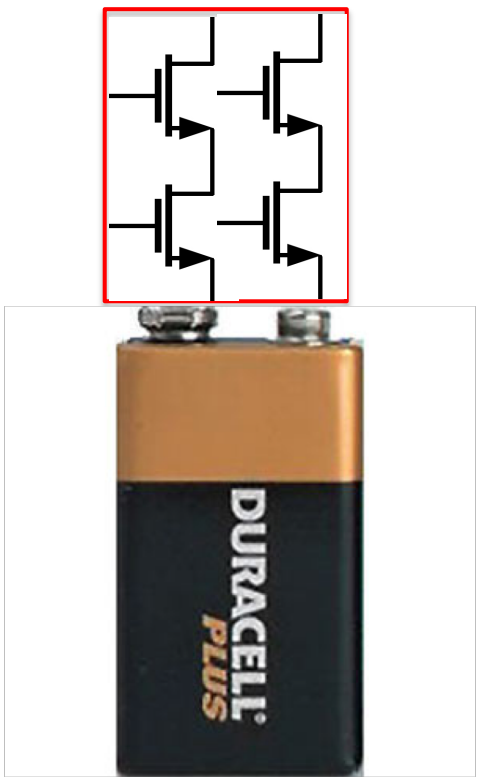
icsa | Institute for Computing Systems Architecture

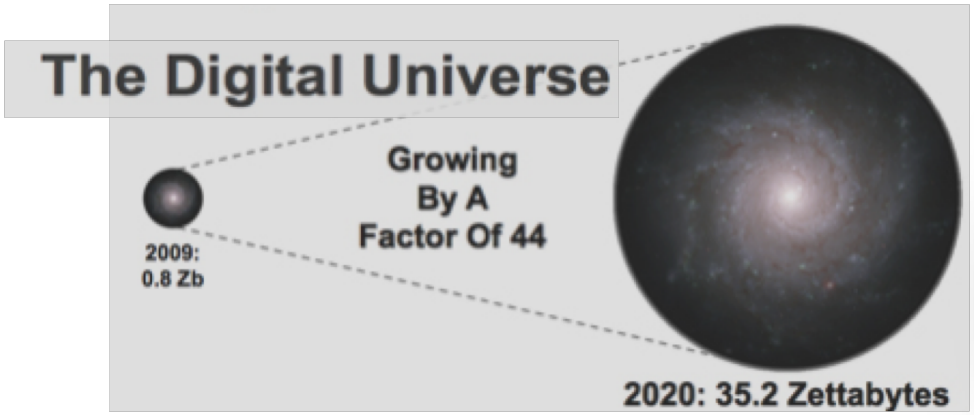
# Near-Memory Processing: It's the SW *and* HW, stupid!

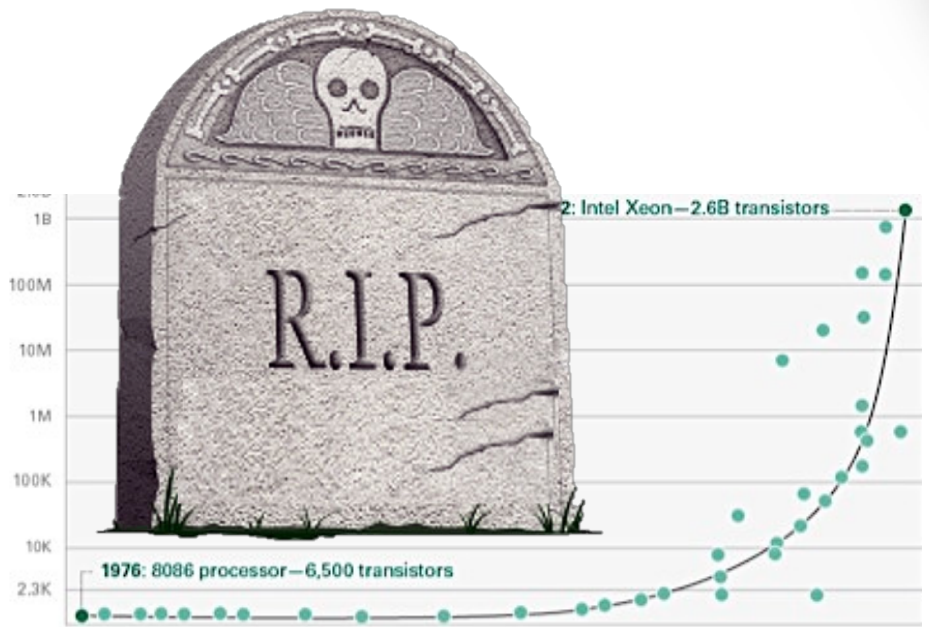
*Boris Grot*

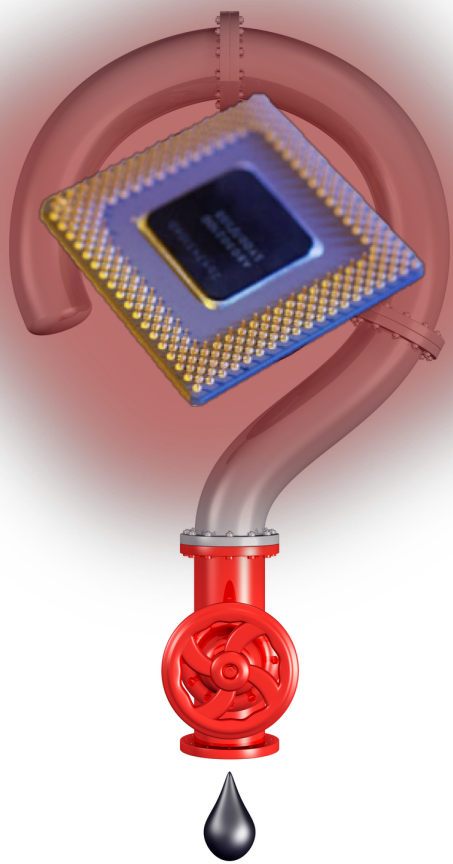


# The End is ~~Near~~ Here!



Where do we go from here?

# An **exponential** is ending…

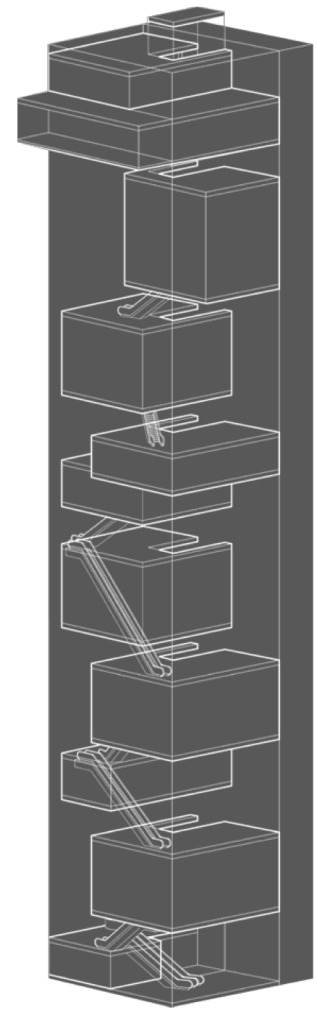
10%, 20%, .. improvement in performance of component X won't get you far

- No new transistors
- Fixed power ceiling

Emerging technologies are either incremental (e.g., Intel's Xpoint Memory) or cover niche areas (e.g., quantum)

# The Way Forward: Vertical Integration

Software/hardware co-design
for high efficiency and programmability



Is this **always** a good idea?

**No!**

Need high volume for cost-efficiency

Need large perf/Watt gains to be worth the effort

# This Talk

Vertical integration

for

in-memory data analytics





ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

FORTH-ICS

# Data Analytics Takes Center Stage

User data grows exponentially

- Need to monetize data


In-memory data operators

- Poor locality
- Low computational requirement
- Highly parallel

# Data Analytics Takes Center Stage

## User data grows exponentially

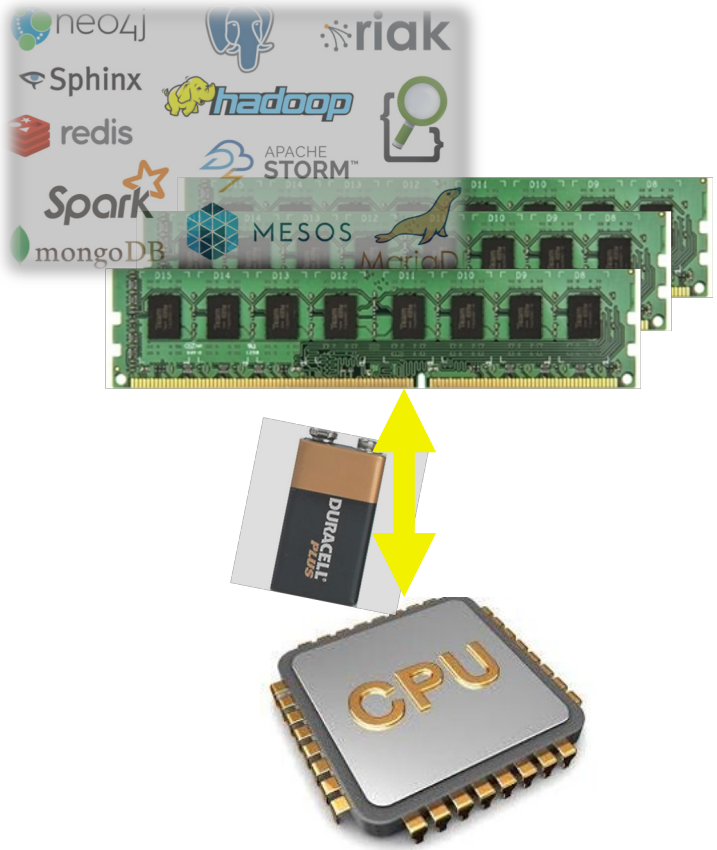
- Need to monetize data

## In-memory data operators

- Poor locality
- Low computational requirement
- Highly parallel

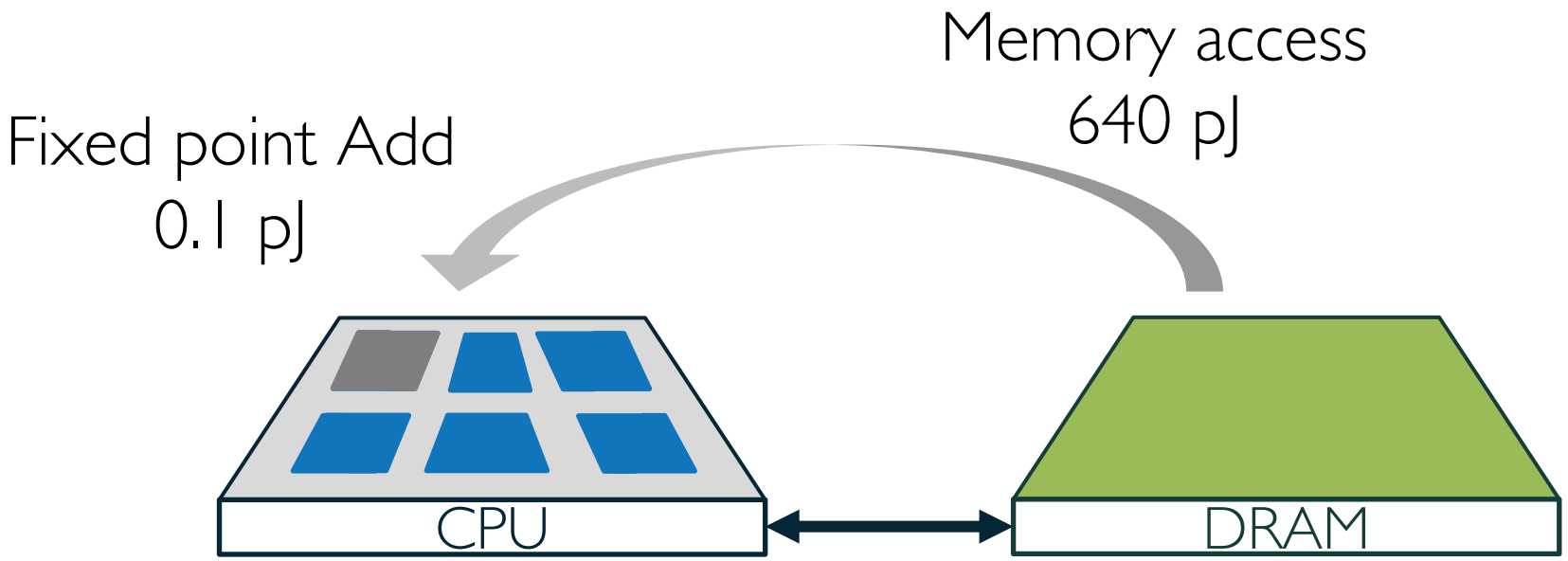
## Data movement

- High energy cost
- High BW requirement



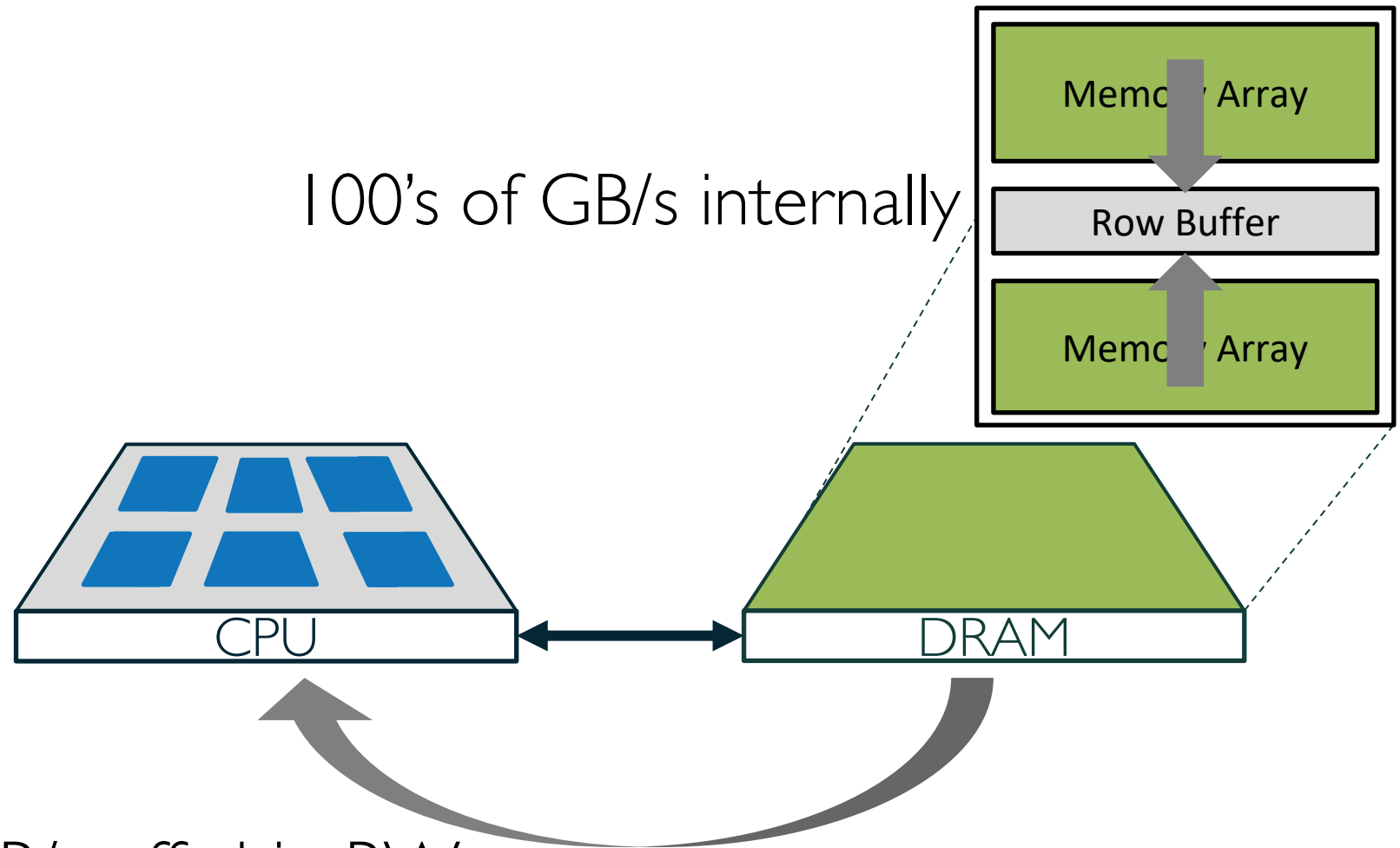

Data movement bottlenecks data analytics

# Cost of Moving Data




Data access much more expensive than arithmetic operation

# DRAM BW Bottleneck




Internal DRAM BW presents big opportunity

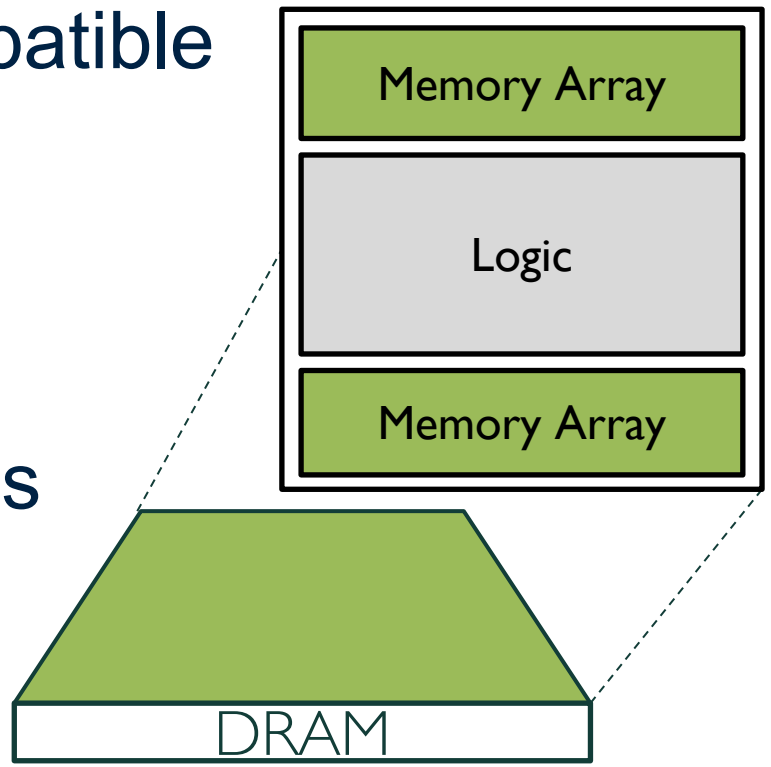
# Logic inside DRAM? Not a Good Idea

Fabrication processes not compatible

- DRAM is optimized for density
- Logic is irregular, wire-intensive

In-memory logic failed in the 90s

- DRAM is cost-sensitive

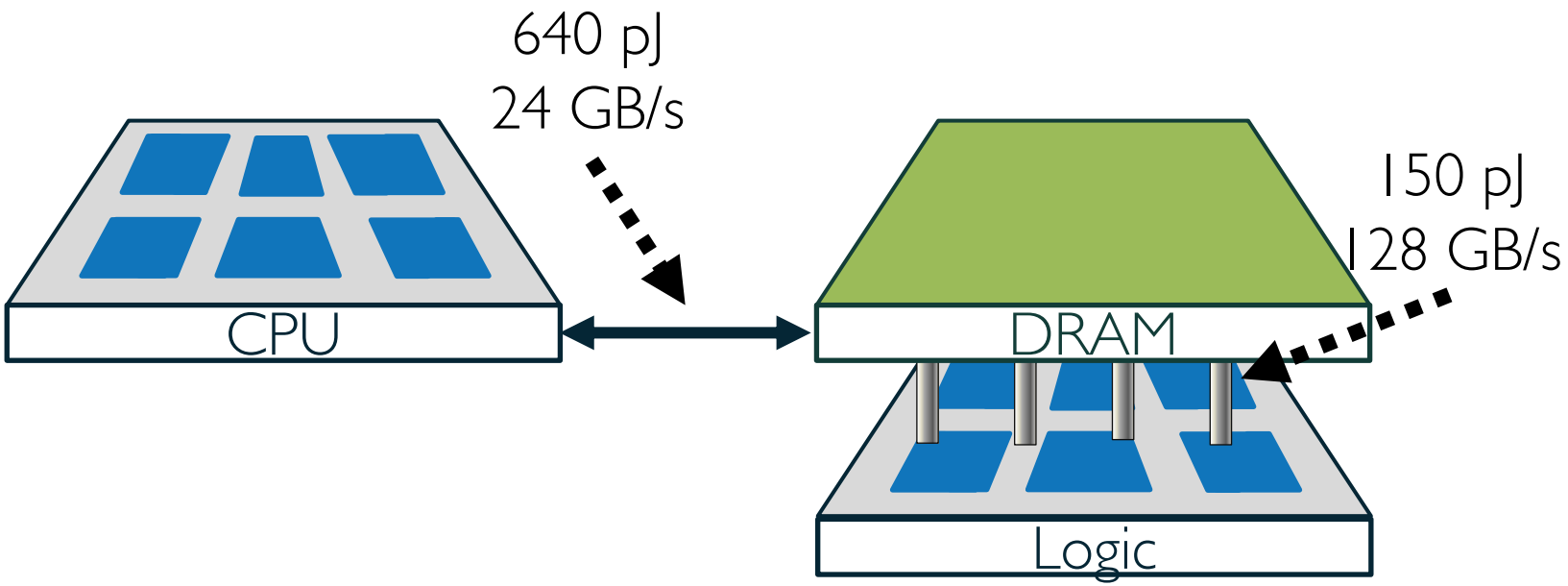



Must exploit DRAM in a non-disruptive manner

# Near-Memory Processing (NMP)

## 3D logic/DRAM stack

- Exposes internal BW to processing elements
- But constrains logic layer's area/power envelope




Exploit the bandwidth without data movement

# How to Best Exploit DRAM BW?

DRAM internals optimized for density

DRAM accesses must activate *rows*

- Single access activates KBs of data
- Activations dominate access latency & energy


Can't utilize internal BW with random access

- Need to maintain many open rows
- Complex bookkeeping logic

Need sequential access to utilize DRAM BW

# NMP HW-Algorithm Co-Design

Algorithms: Must have sequential access

- Even if we perform more work

Hardware: Must leverage data parallelism

- On a tight area/power budget

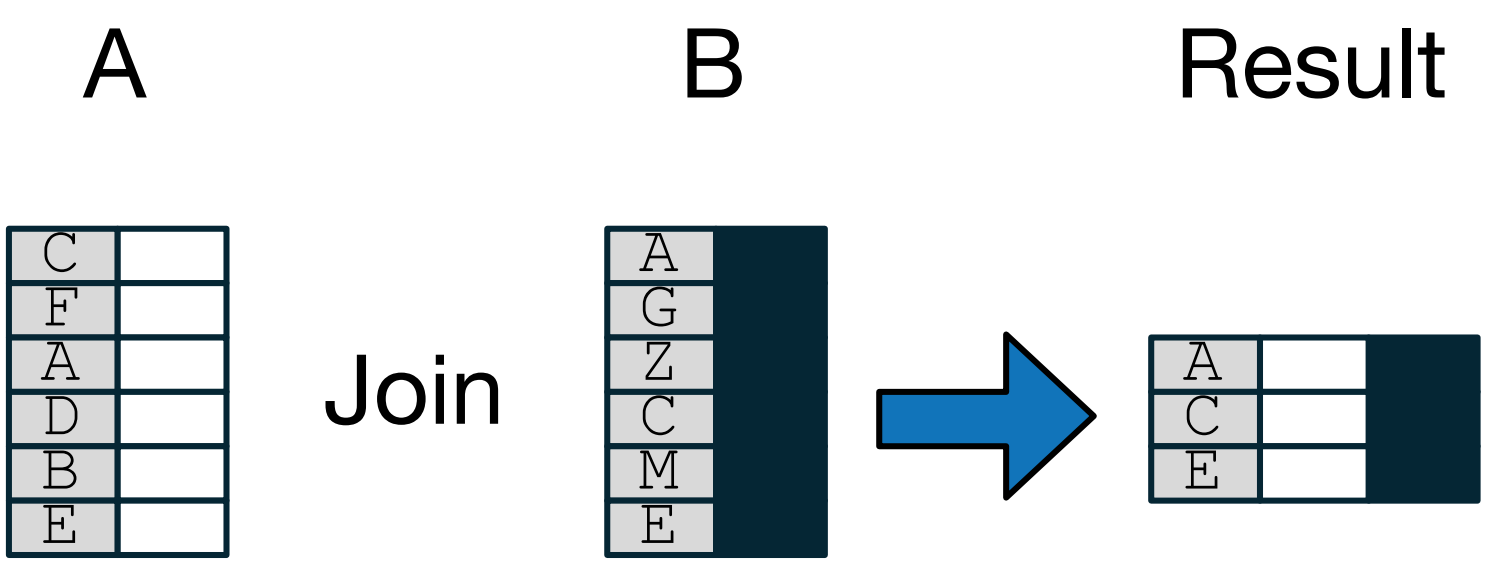
HW-algorithm co-design necessary to make best use of NMP

# Example data operator: Join

Iterates over a pair of tables to find matching keys

Major operation in data analytics

```
Q: SELECT ... FROM A, B WHERE A.Key = B.Key
```



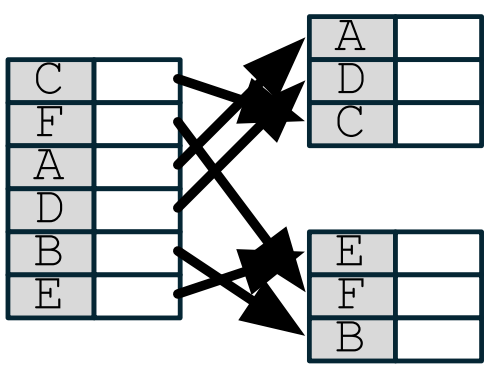
# Baseline: CPU Hash Join

Best performing algorithm in CPU-centric systems

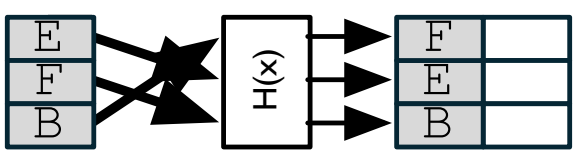
Performed in two phases: Partition & Probe

1. **Partition** generates cache sized partitions
2. **Probe** builds and queries cache resident hash tables

Partition

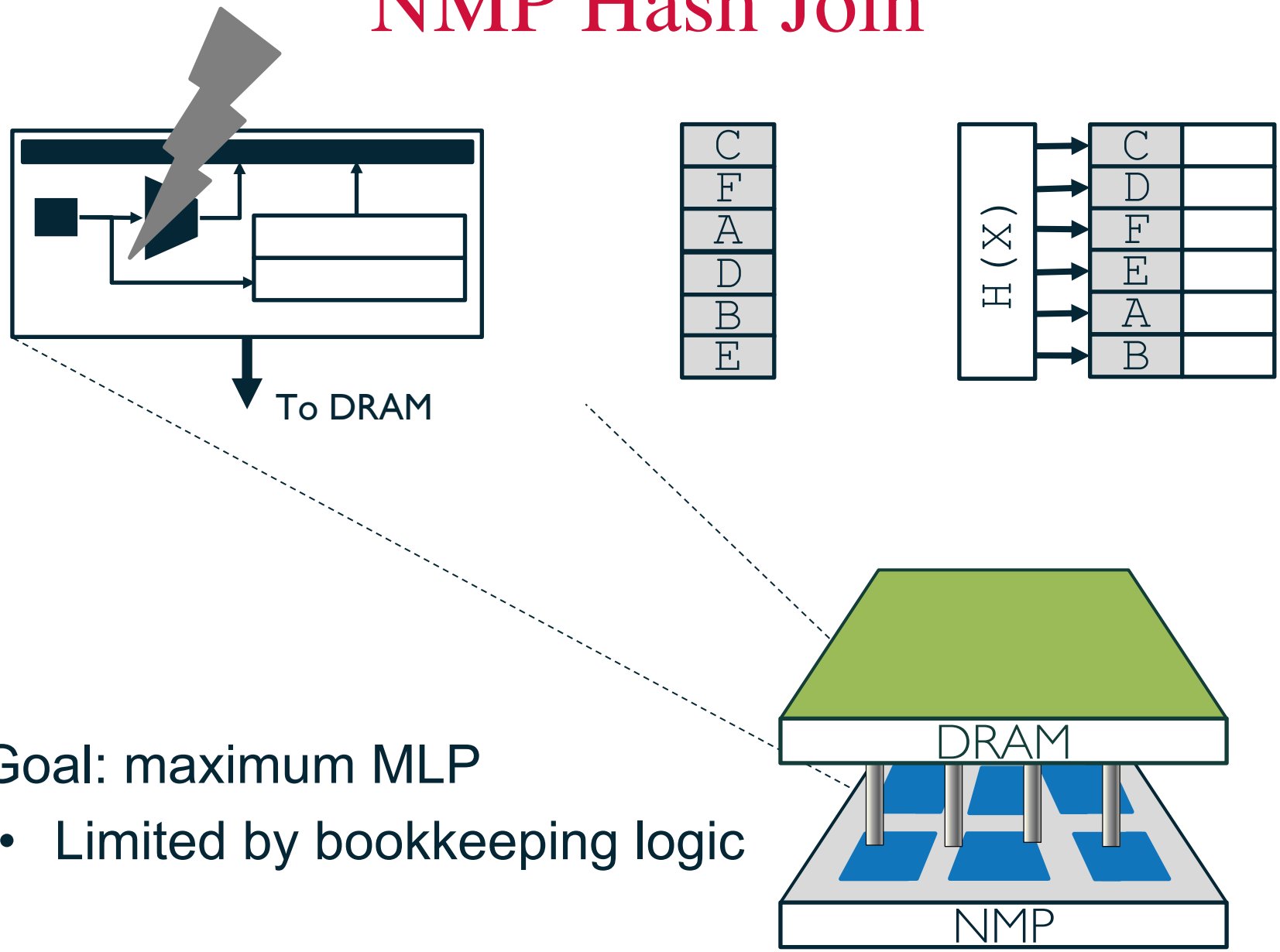



Probe




Optimized for random accesses to cached data

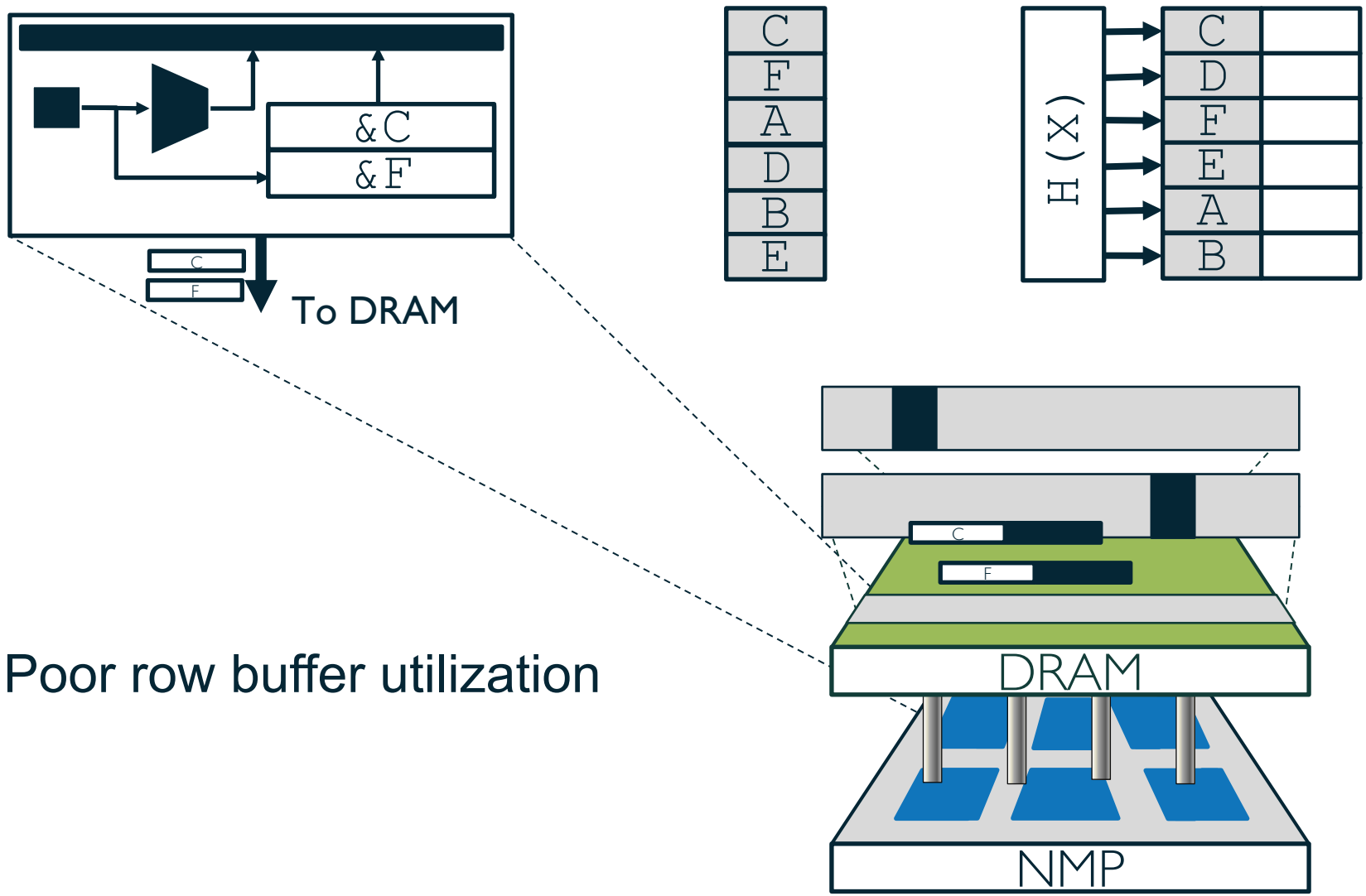
# NMP Hash Join




Goal: maximum MLP

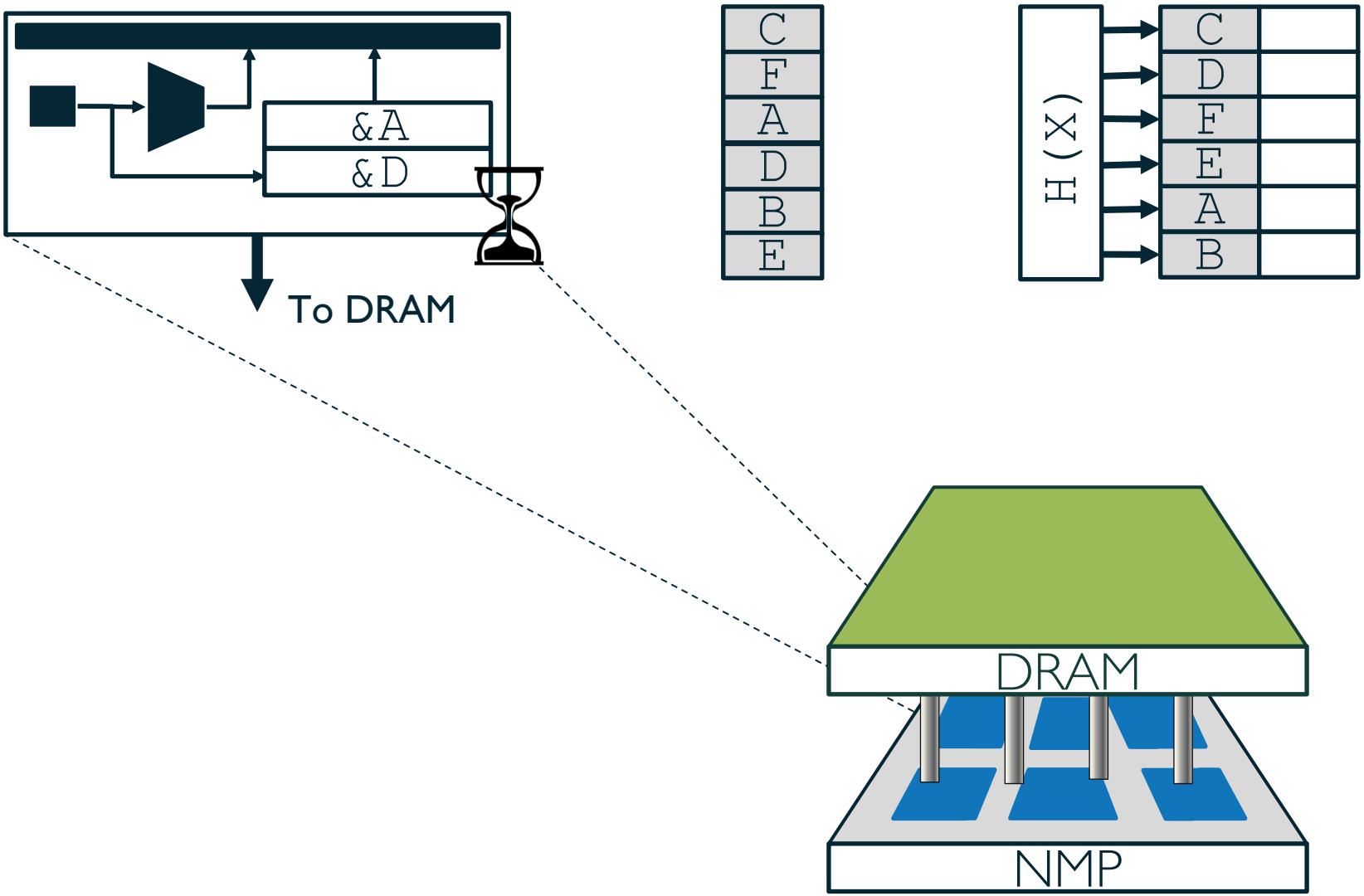
- Limited by bookkeeping logic

# NMP Hash Join




Poor row buffer utilization

# NMP Hash Join



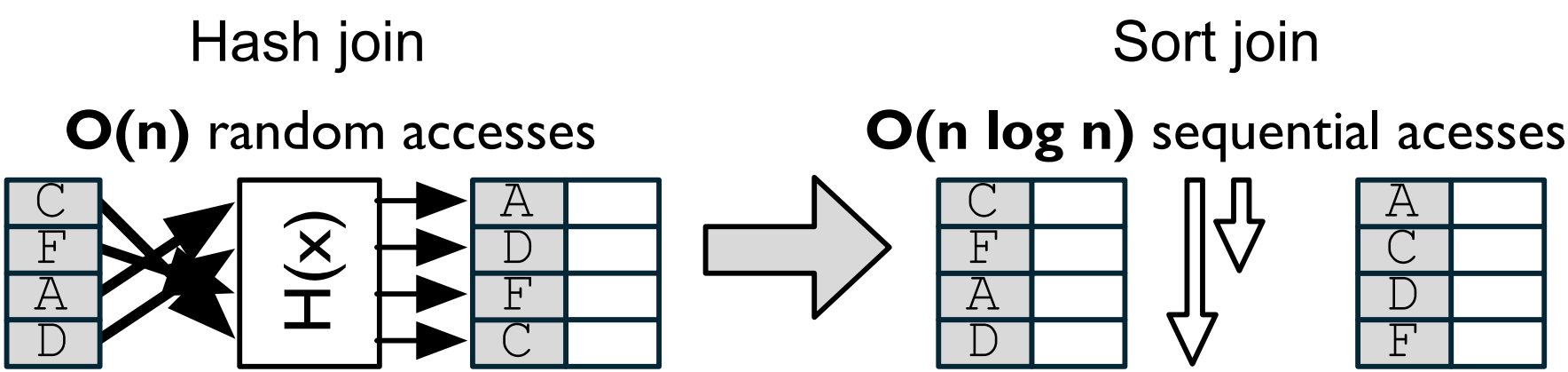

Random accesses are inefficient and under-utilize internal BW

# Eliminate Random Access?

Insight: use **Sort Join**

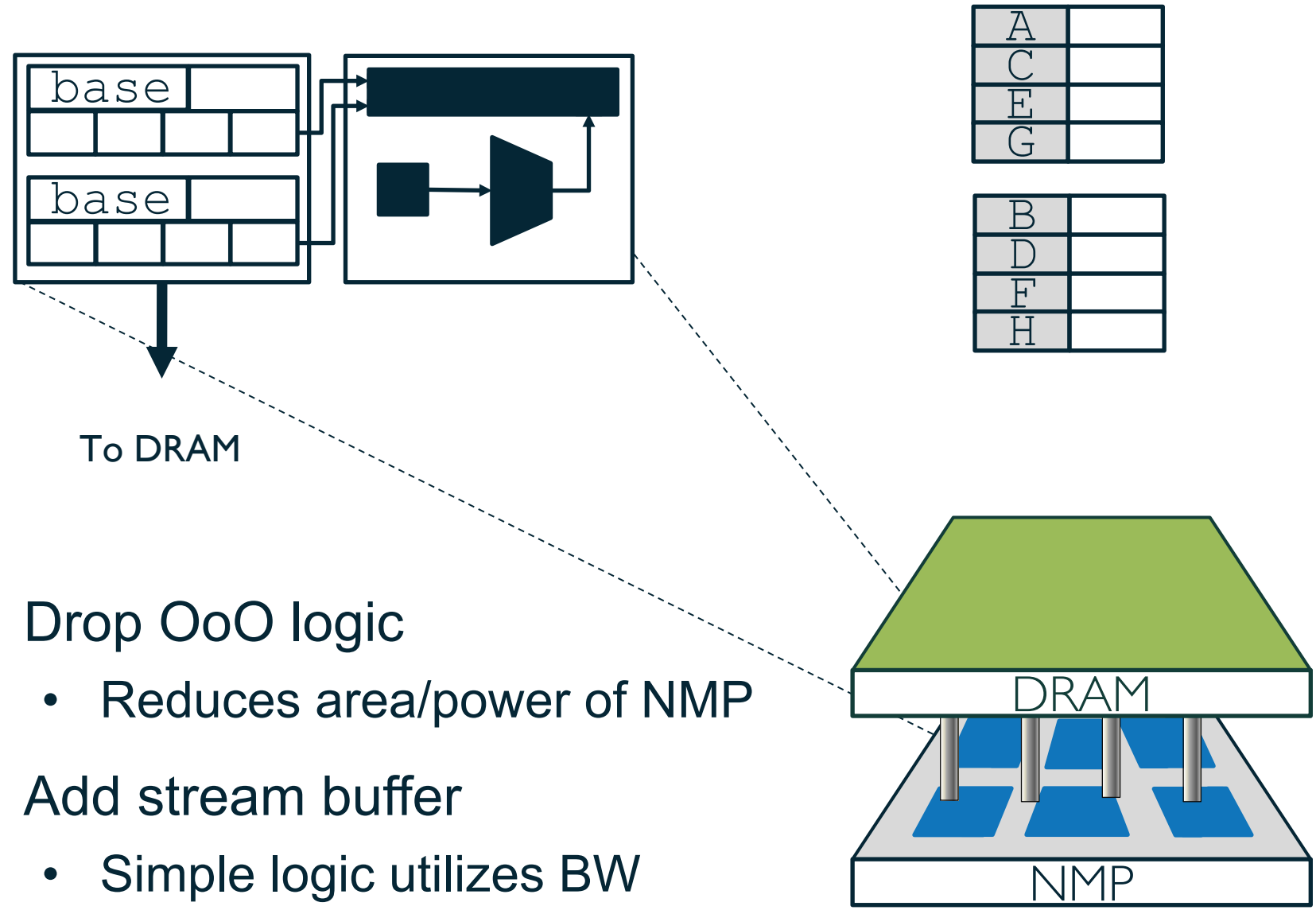
- Performs mostly sequential accesses
- But has higher algorithmic complexity

Trade algorithmic complexity for desirable access pattern




Utilizing internal DRAM BW compensates for increased cost

# NMP Sort Join: Sequential Accesses



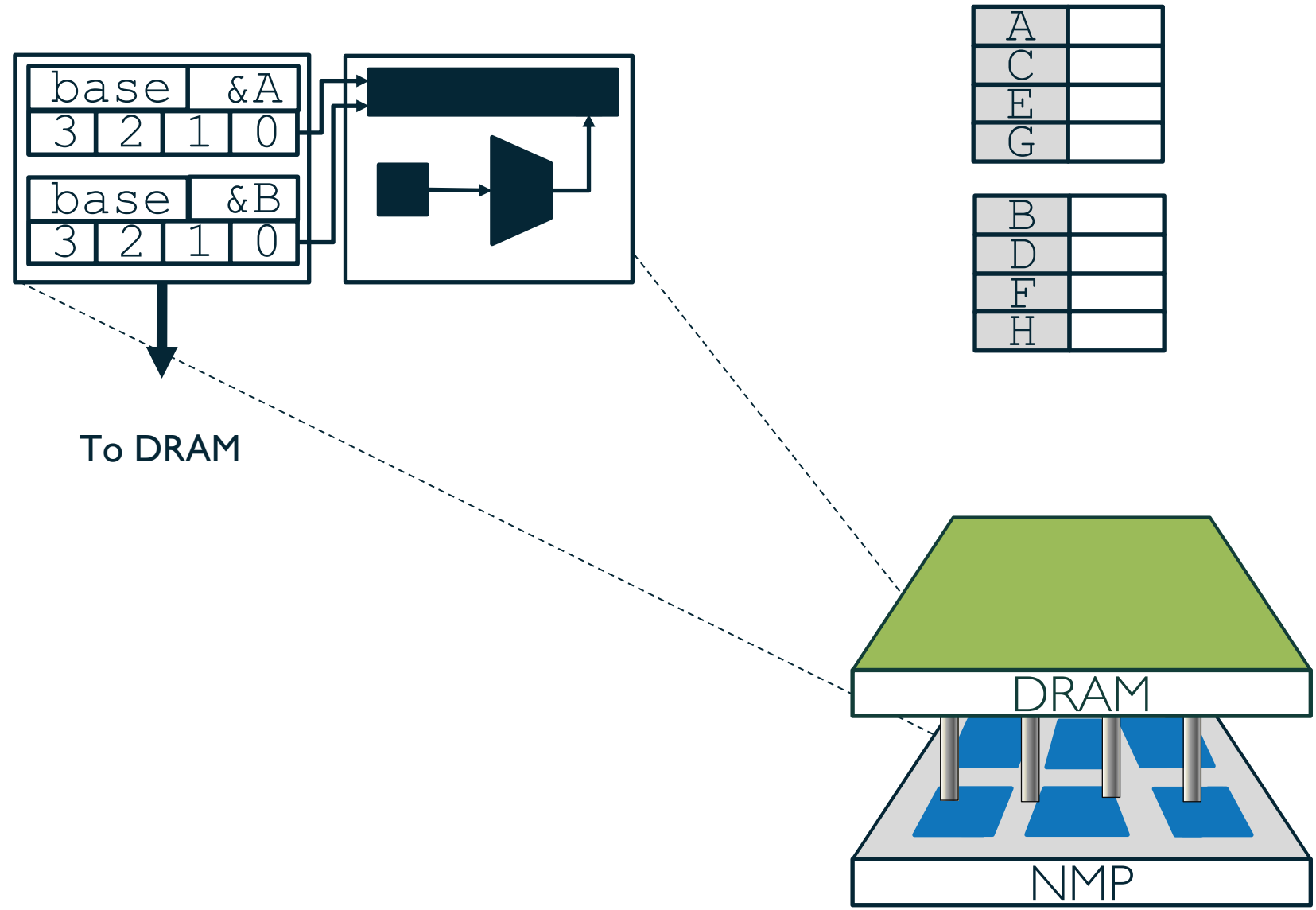

Drop OoO logic

- Reduces area/power of NMP

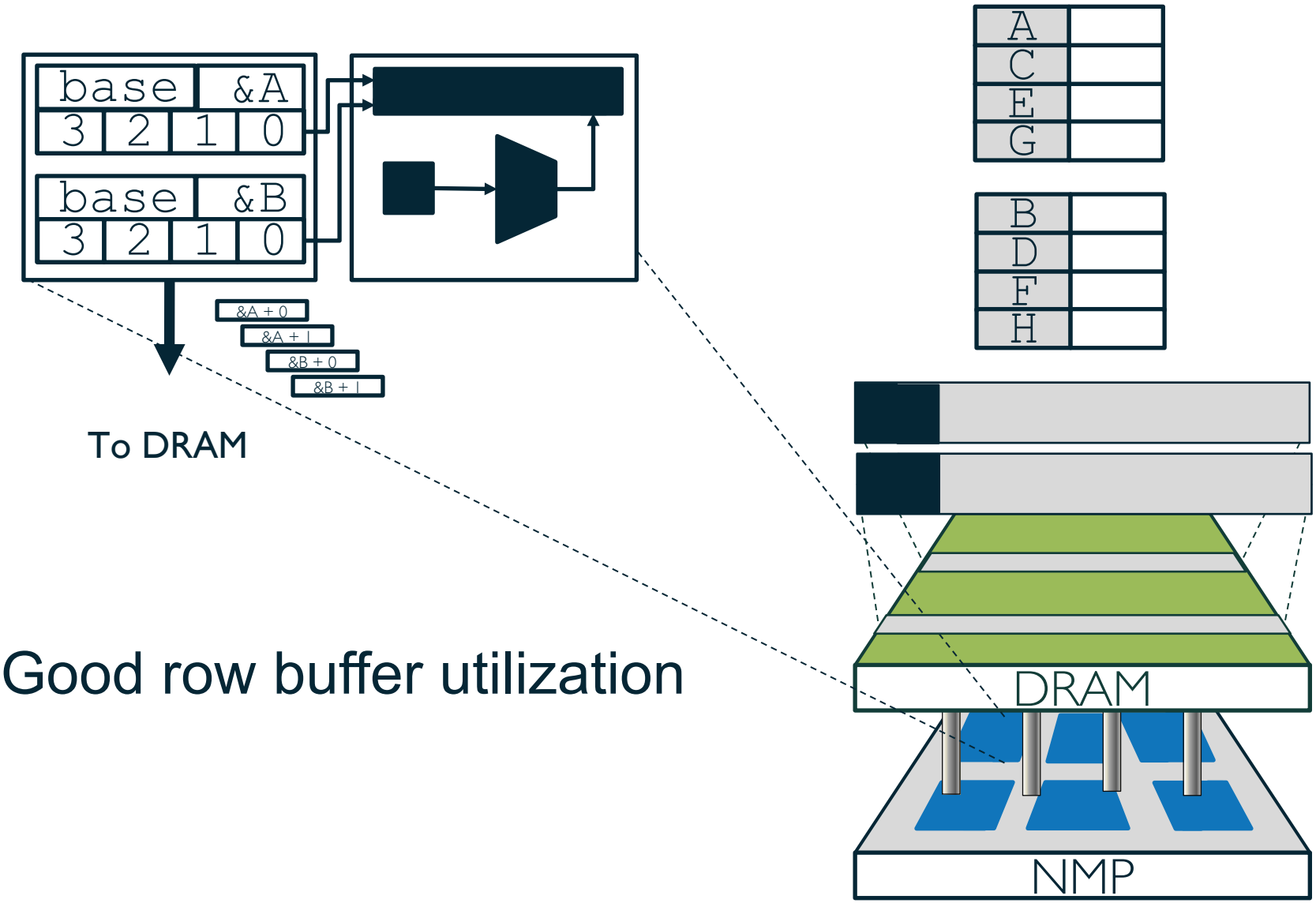
Add stream buffer

- Simple logic utilizes BW

# NMP Sort Join: Sequential Accesses

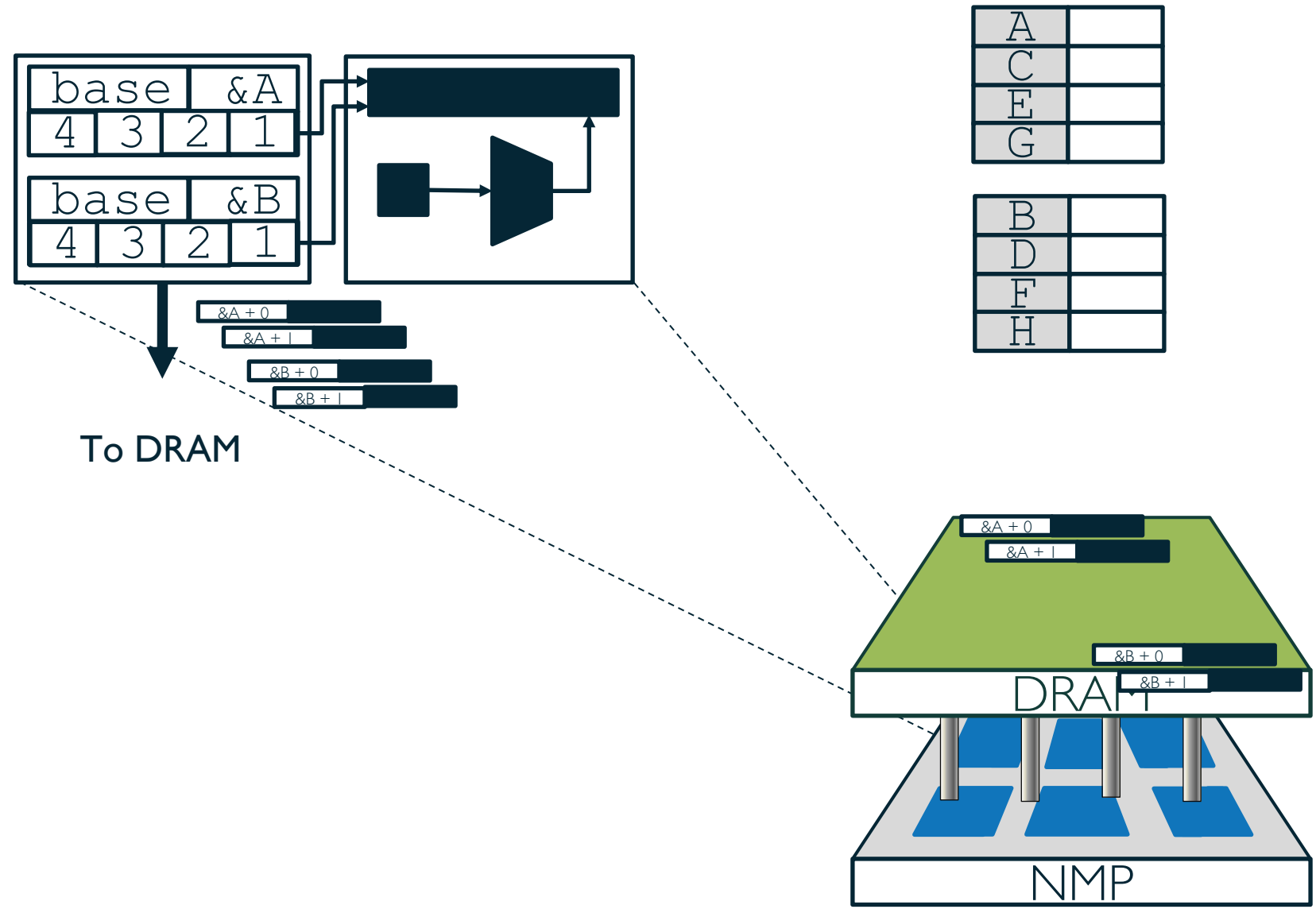


# NMP Sort Join: Sequential Accesses

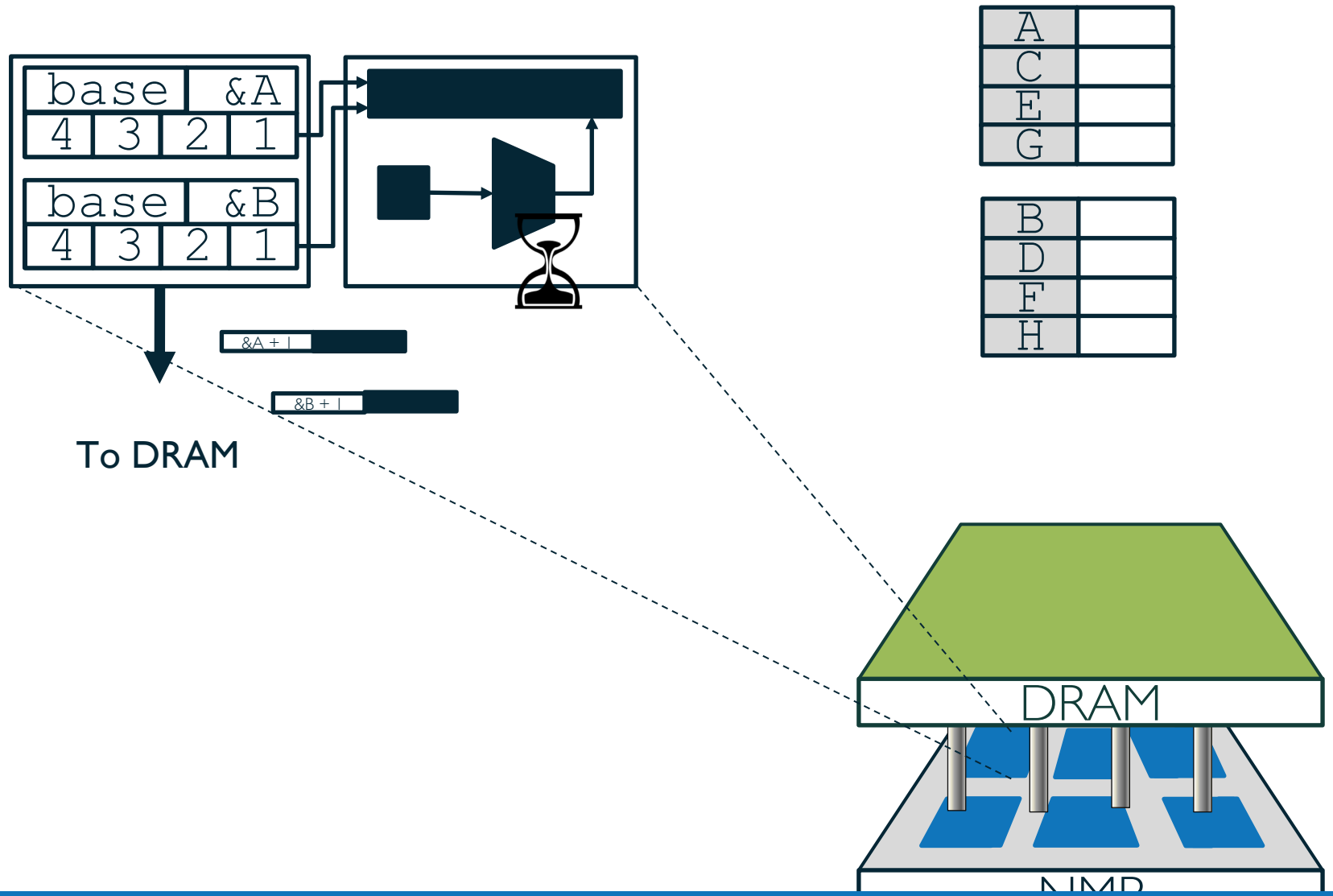



Good row buffer utilization

# NMP Sort Join: Sequential Accesses

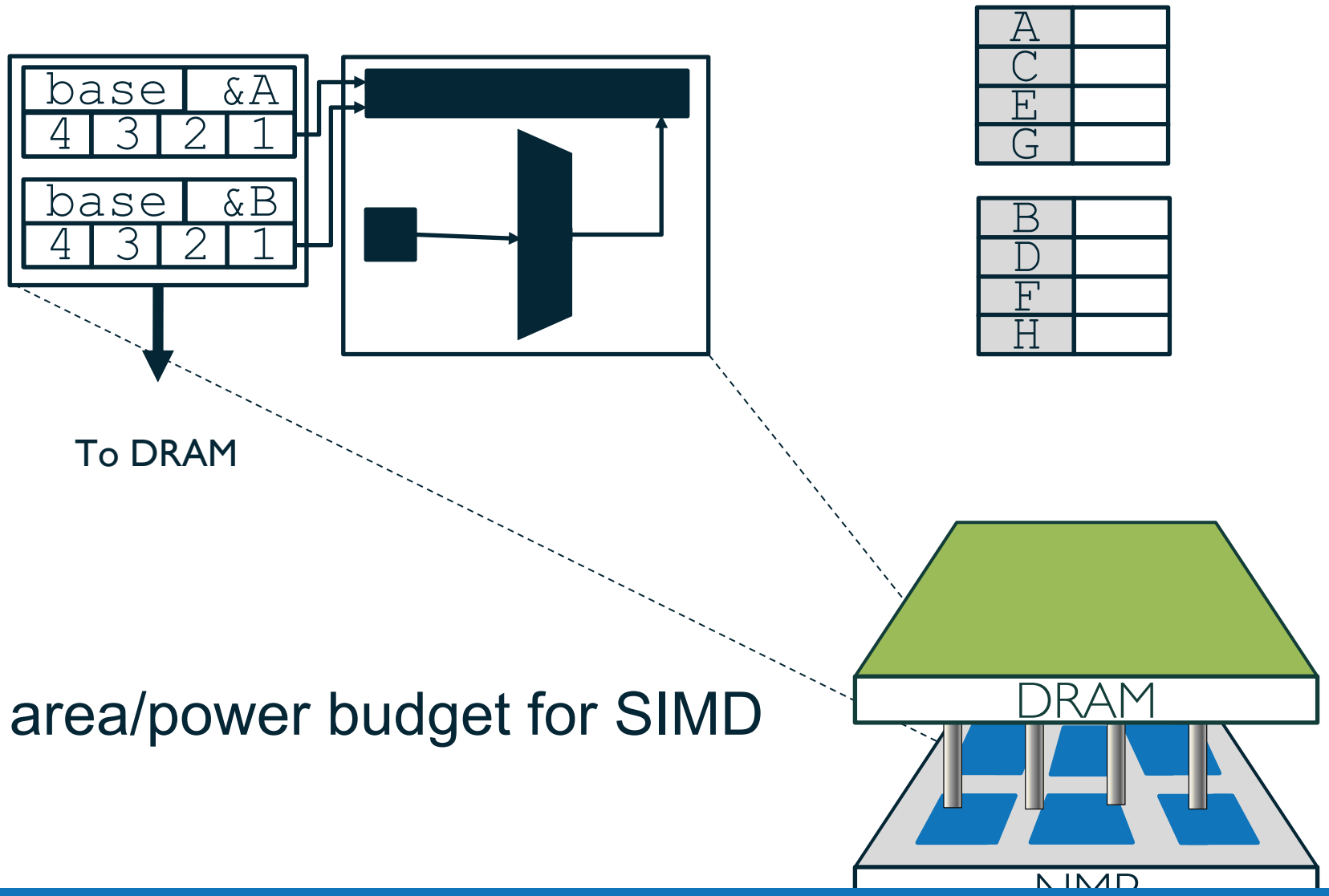


# NMP Sort Join: Sequential Accesses




Sequential access moves bottleneck to compute

# NMP Sort Join: Compute




Use area/power budget for SIMD

General purpose SIMD keeps up with memory BW

# Partitioning Phase

Partitioning basics:

- Each partition contains *buckets of objects*
- For a given object, target bucket determined using a hash
- The order of objects within each bucket is irrelevant → buckets are *unordered*

Insight: the order in which tuples are written into a bucket in the target partition is irrelevant

# Partitioning Phase

Leverage tuple's permutability property

Turn partition's random accesses sequential

- Enable use of SIMD during partition

# Mondrian

Algorithm + hardware co-design for near-memory processing of data analytics

## NMP Algorithms

- Use sequential memory accesses
- Avoid random memory accesses
- Target partitioning and compute phases

## NMP Harware

- High memory parallelism using simple SIMD hardware
- Exploit sequential memory accesses

# Methodology

Flexus cycle accurate simulator [Wenisch'06]

Big data operators:

- Scan
- Join
- Group By
- Sort

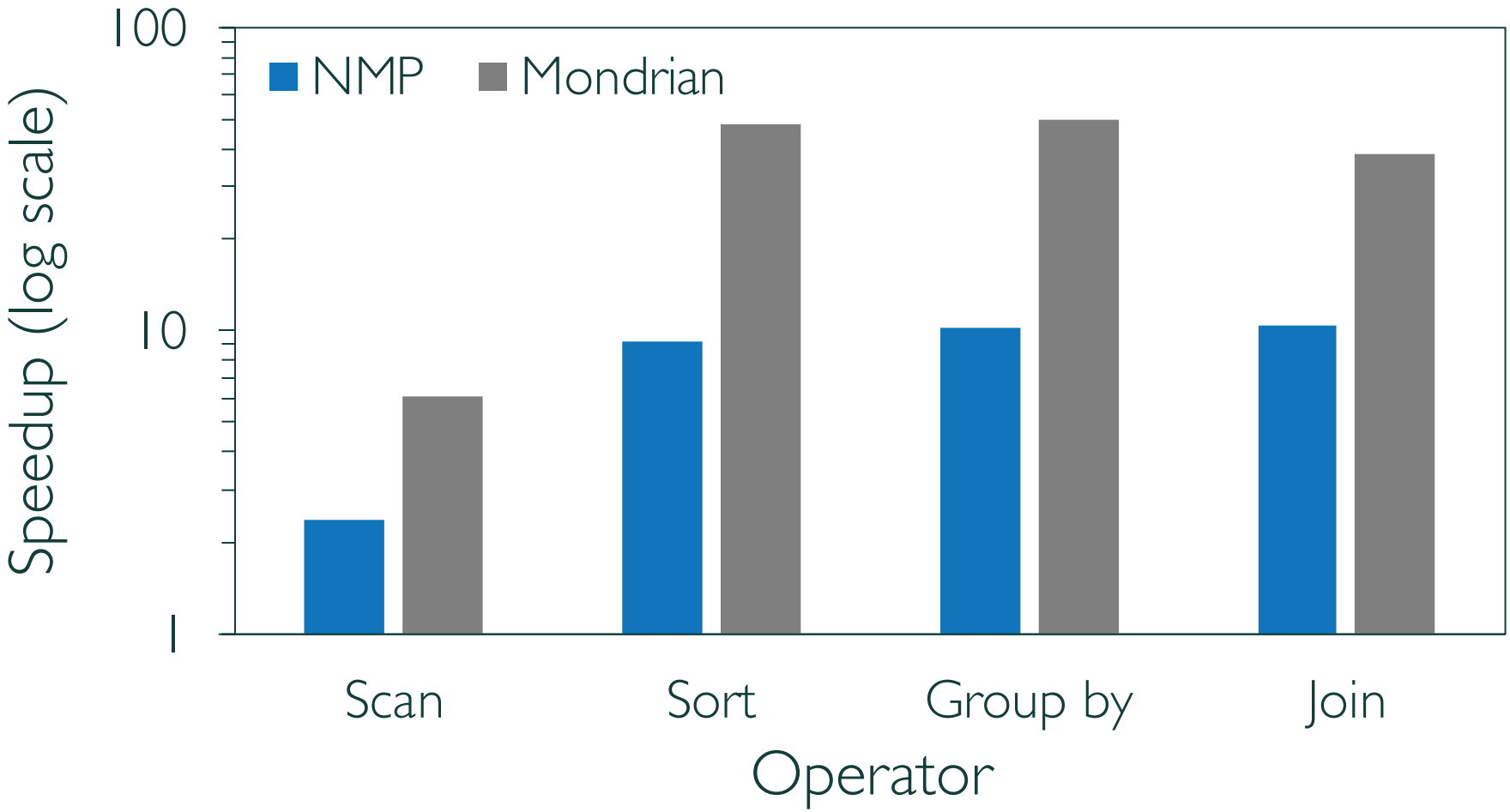
Memory subsystem:

- 4 HMC stacks
  - 20 GB/s external BW
  - 128 GB/s internal BW

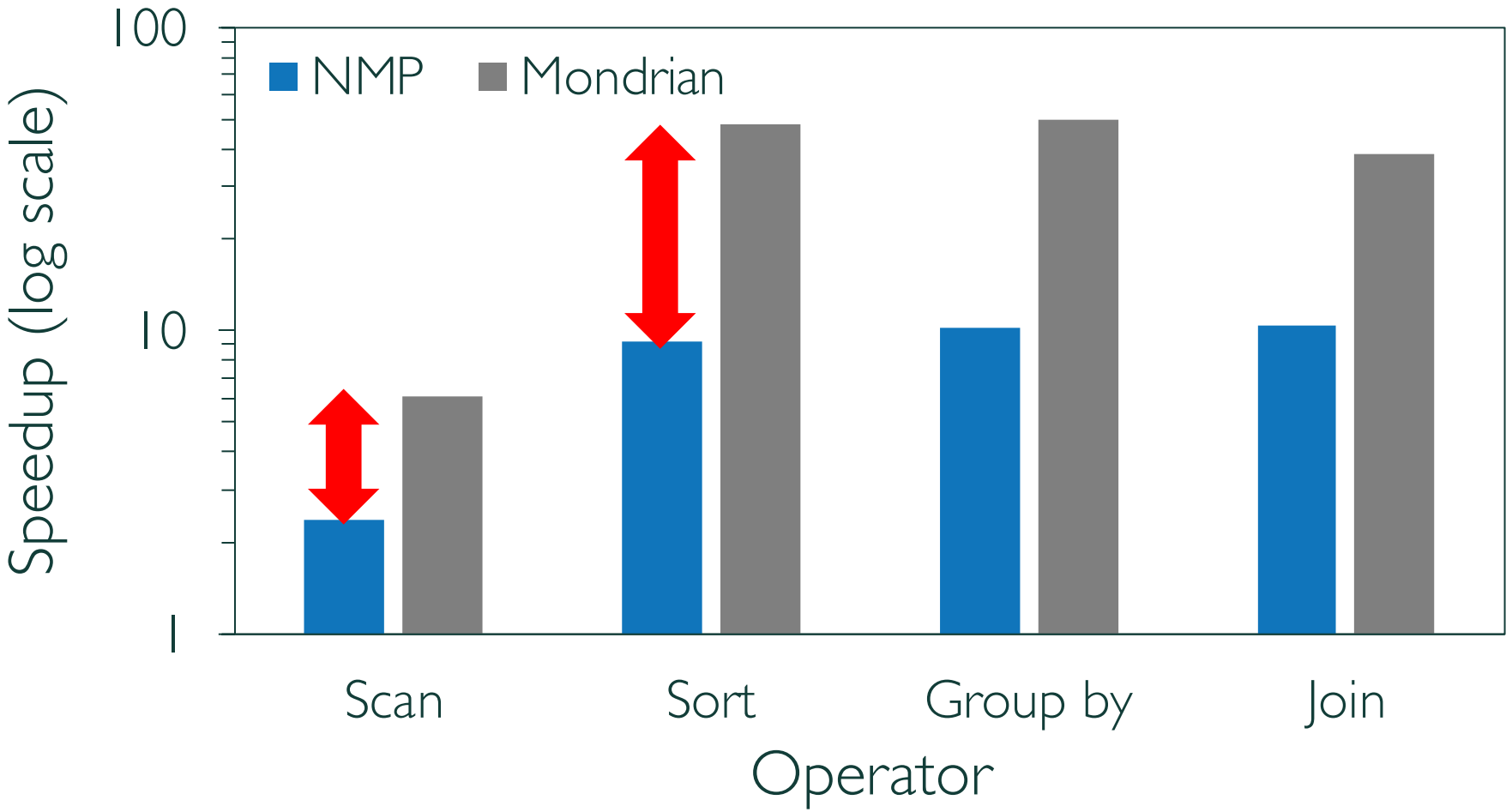
Simulated systems:

- CPU-centric: ARM Cortex-A57
  - 16 cores
  - 3-wide,128-entry ROB @ 2GHz
- NMP: Mobile ARM core
  - 16 cores per stack
  - 3-wide, 48-entry ROB @ 1GHz
- Mondrian: SIMD in-order
  - 16 cores per stack
  - 1024-bit SIMD @ 1GHz

# Evaluation: Performance

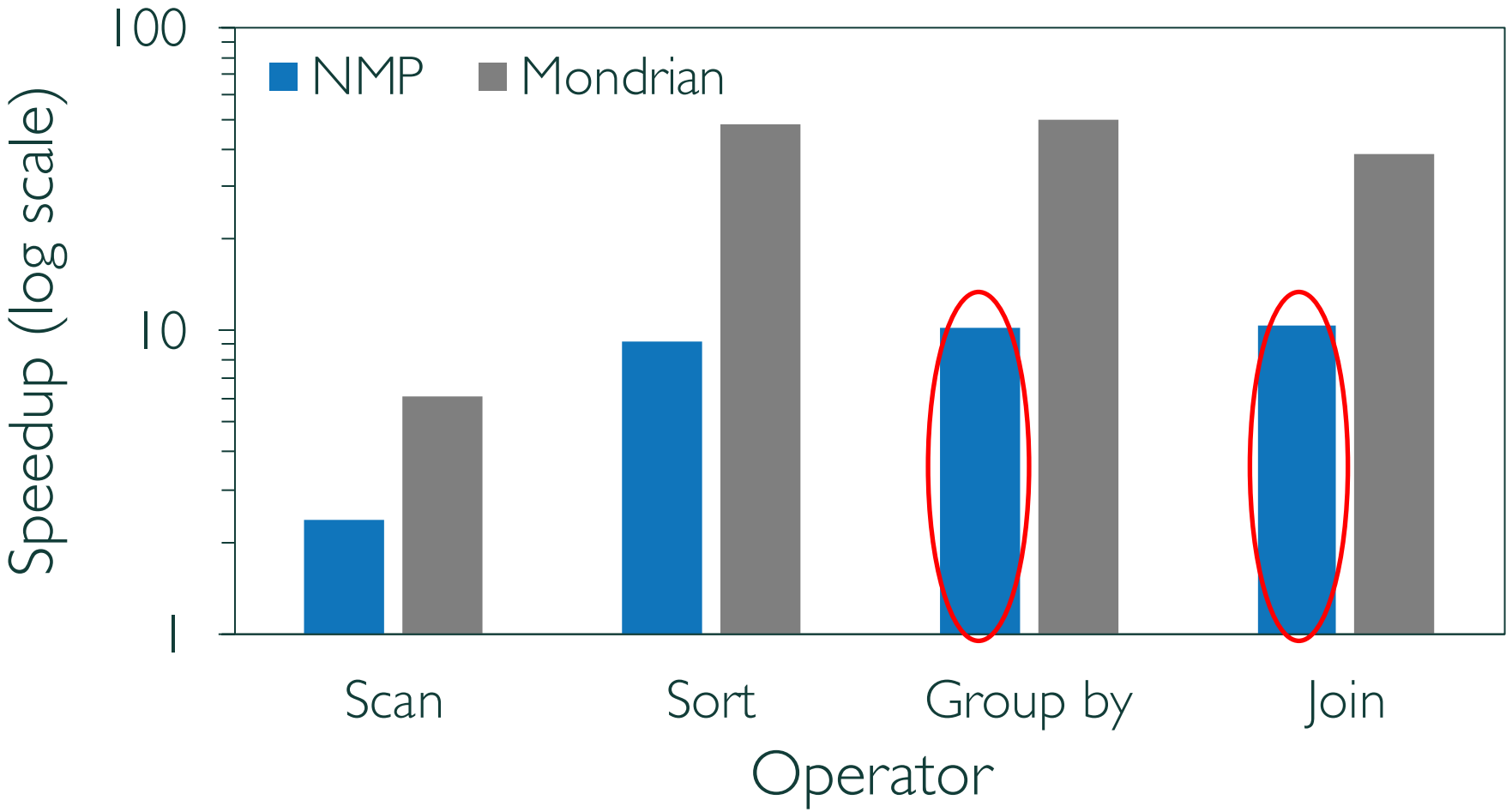


# Evaluation: Performance



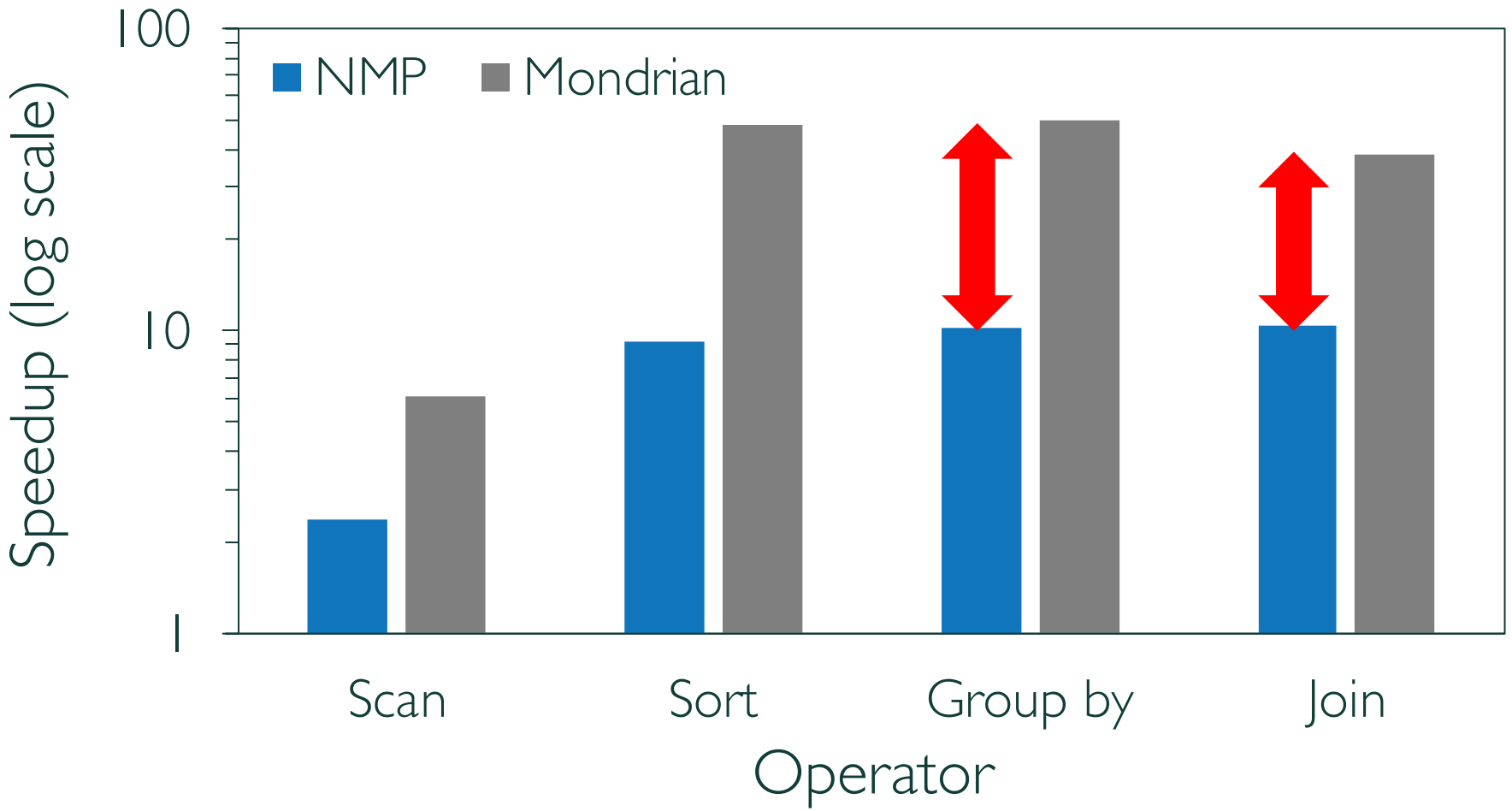

Mondrian achieves superior BW utilization

# Evaluation: Performance




NMP can't utilize memory BW with random accesses

# Evaluation: Performance




Mondrian BW utilization compensates for extra log(n) work

# Summary

End of technology scaling → must think vertical

- Software + hardware co-design

Big data analytics are a critical workload

- Large datasets, little locality → memory bottleneck!

Moving compute near memory improves performance

- But need to conform to DRAM constraints

**Mondrian** is algorithm-hardware NMP for analytics

- Adapt algorithms/HW to DRAM constraints
- Sequential rather than random memory access
- Simple hardware to exploit memory bandwidth

# Thank you!

# Questions?

*inf.ed.ac.uk/bgrot*

# Mondrian Energy Efficiency