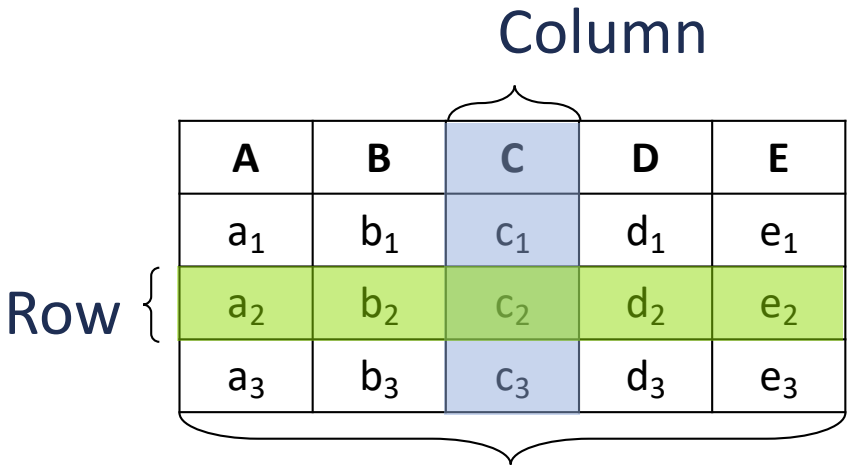
Relational Memory:

Native In-Memory Accesses on Rows and Columns

Ju Hyoung Mun

Relational Databases are everywhere



SQLite

MySQL

ORACLE DATABASE

PostgreSQL

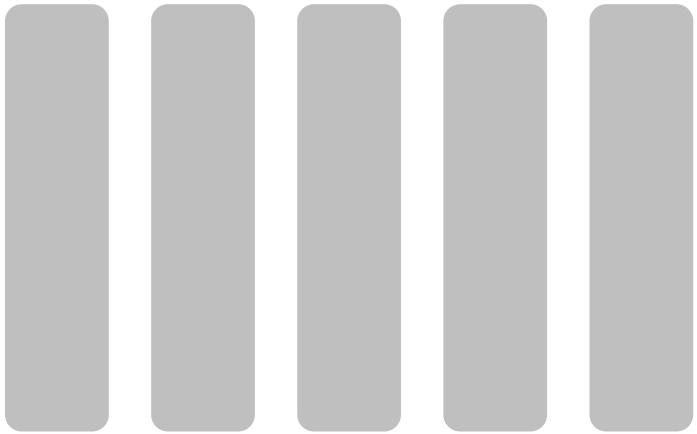
Microsoft SQL Server

Brandeis

Data Layouts

row-stores

column-stores



Transactional

Brandeis

Data Layouts

row-stores

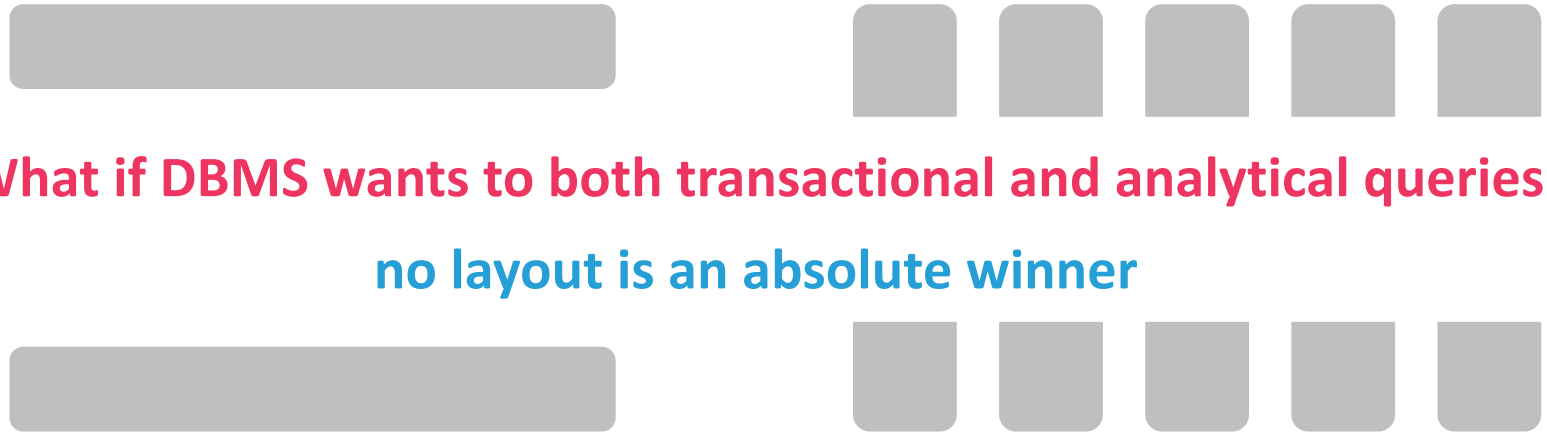
column-stores

Analytical

Data Layouts

row-stores

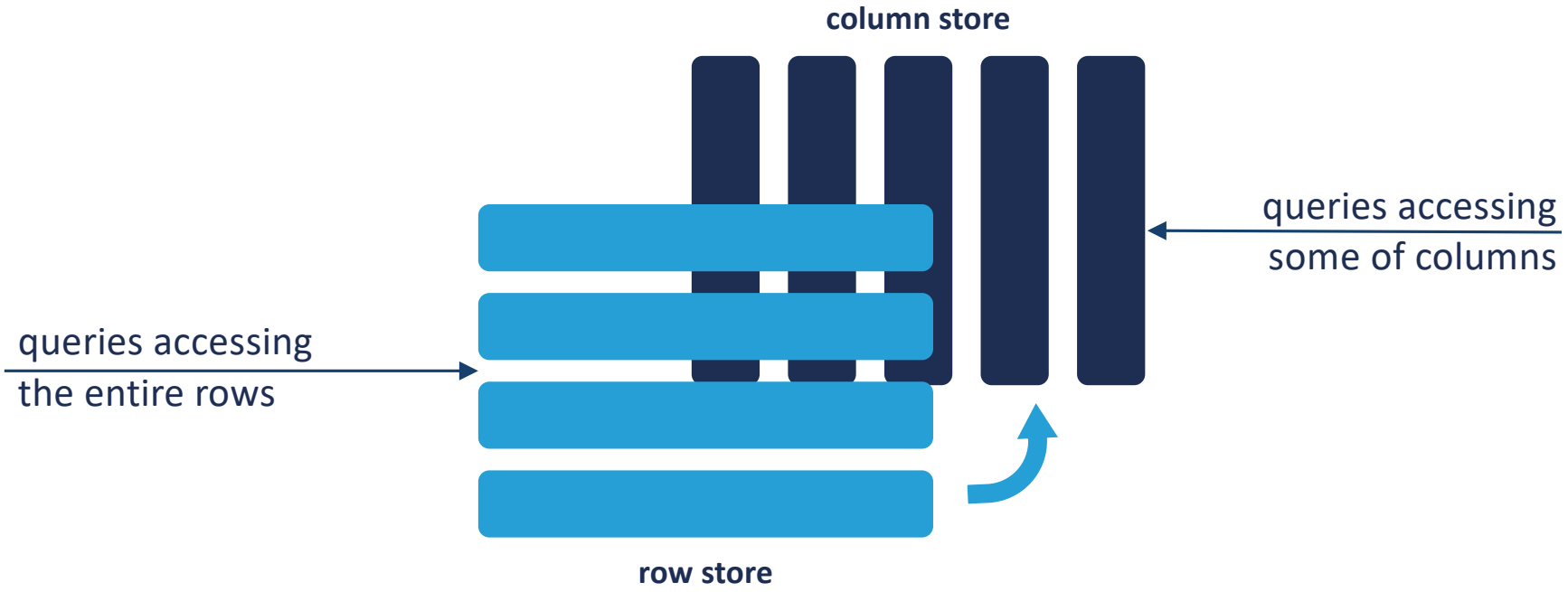
column-stores



What if DBMS wants to both transactional and analytical queries?

no layout is an absolute winner

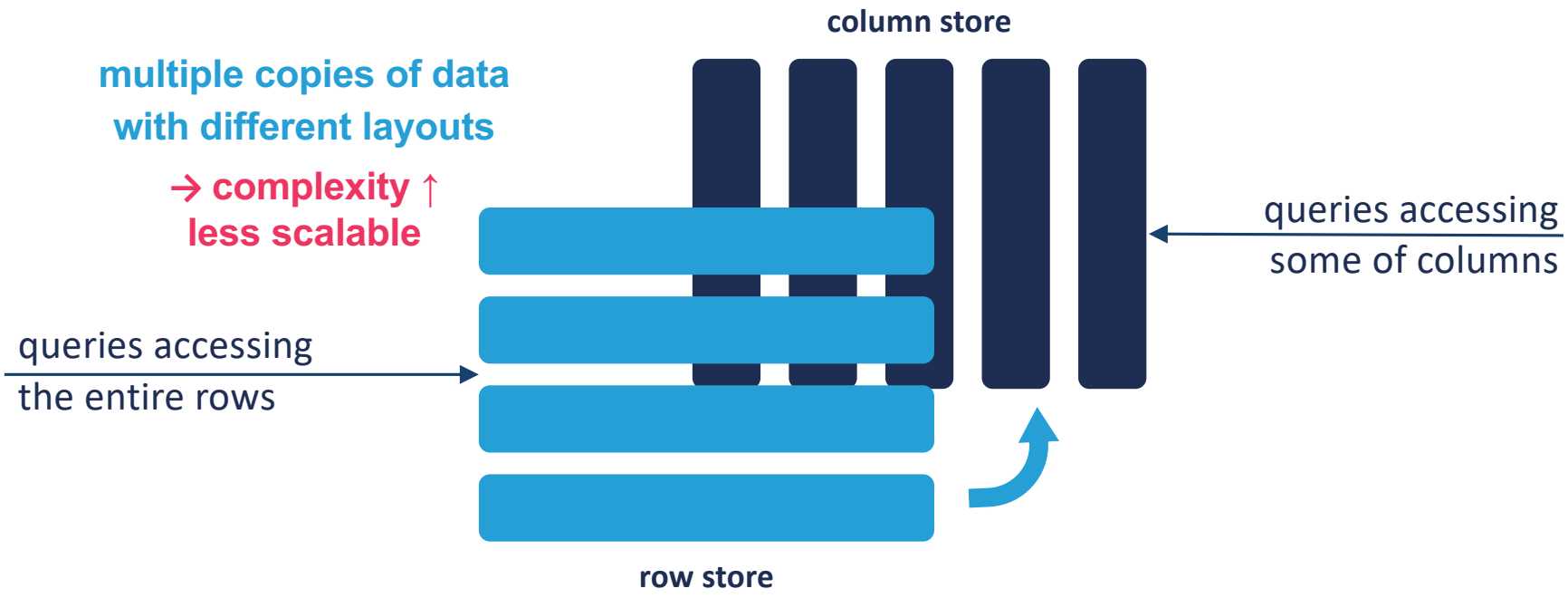
What are the disadvantages of the adaptive layout?



Brandeis

E.g., H2O (ACM SIGMOD, 2014), HyPer (IEEE ICDE , 2011), Peloton (ACM SIGMOD, 2016), OctopusDB (CIDR, 2011)

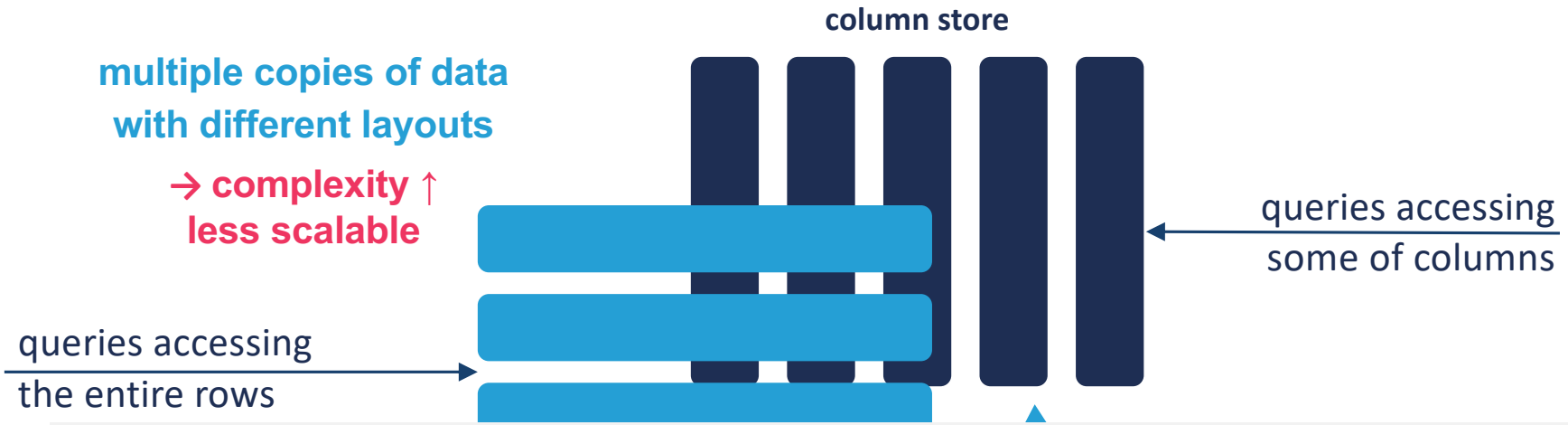
Adaptive layout



Brandeis

E.g., H2O (ACM SIGMOD, 2014), HyPer (IEEE ICDE , 2011), Peloton (ACM SIGMOD, 2016), OctopusDB (CIDR, 2011)

Adaptive layout



What if there is a shift over columns?

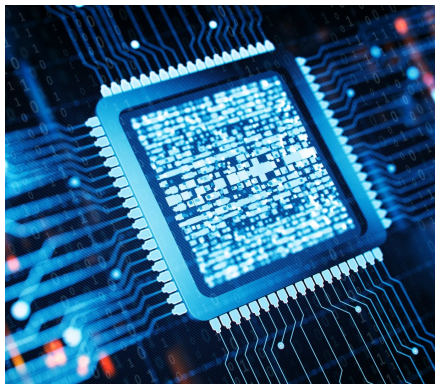
Brandeis

E.g., H2O (ACM SIGMOD, 2014), HyPer (IEEE ICDE , 2011), Peloton (ACM SIGMOD, 2016), OctopusDB (CIDR, 2011)

How can we access
only the desired columns
without storing or maintaining
multiple copies of data?

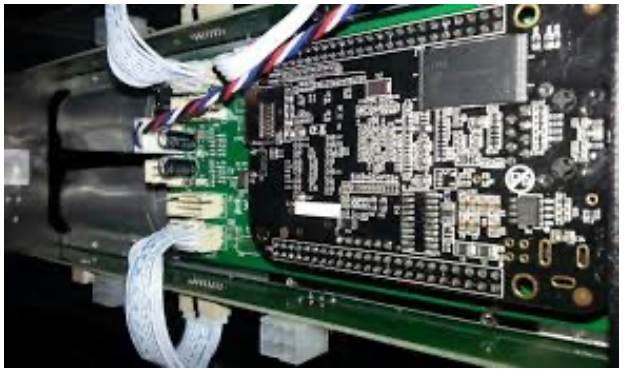
a novel hardware design
for data transformation

Relational Memory



Brandeis

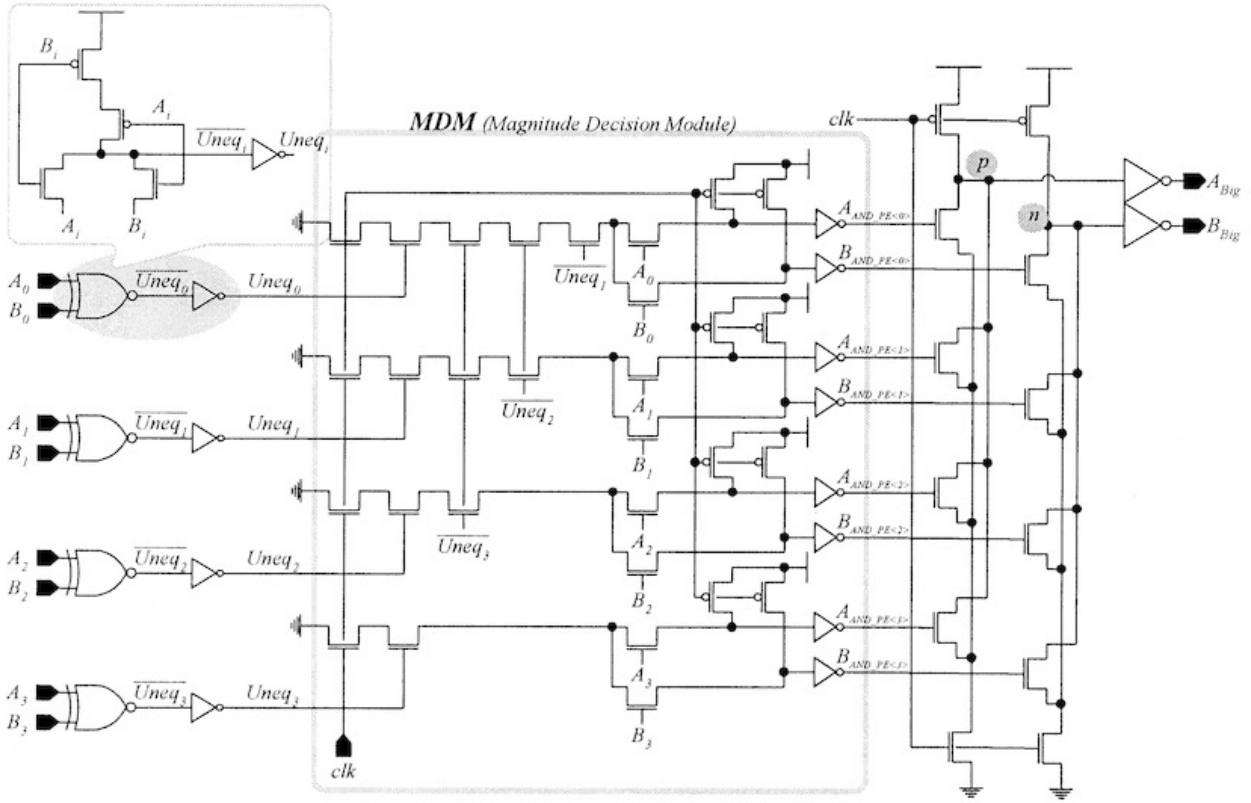

What is
Hardware?



Application-**S**pecific **I**ntegrated **C**ircuits (ASICs)

Brandeis

Integrated Circuits



Brandeis

Hardware Accelerators

Advantages

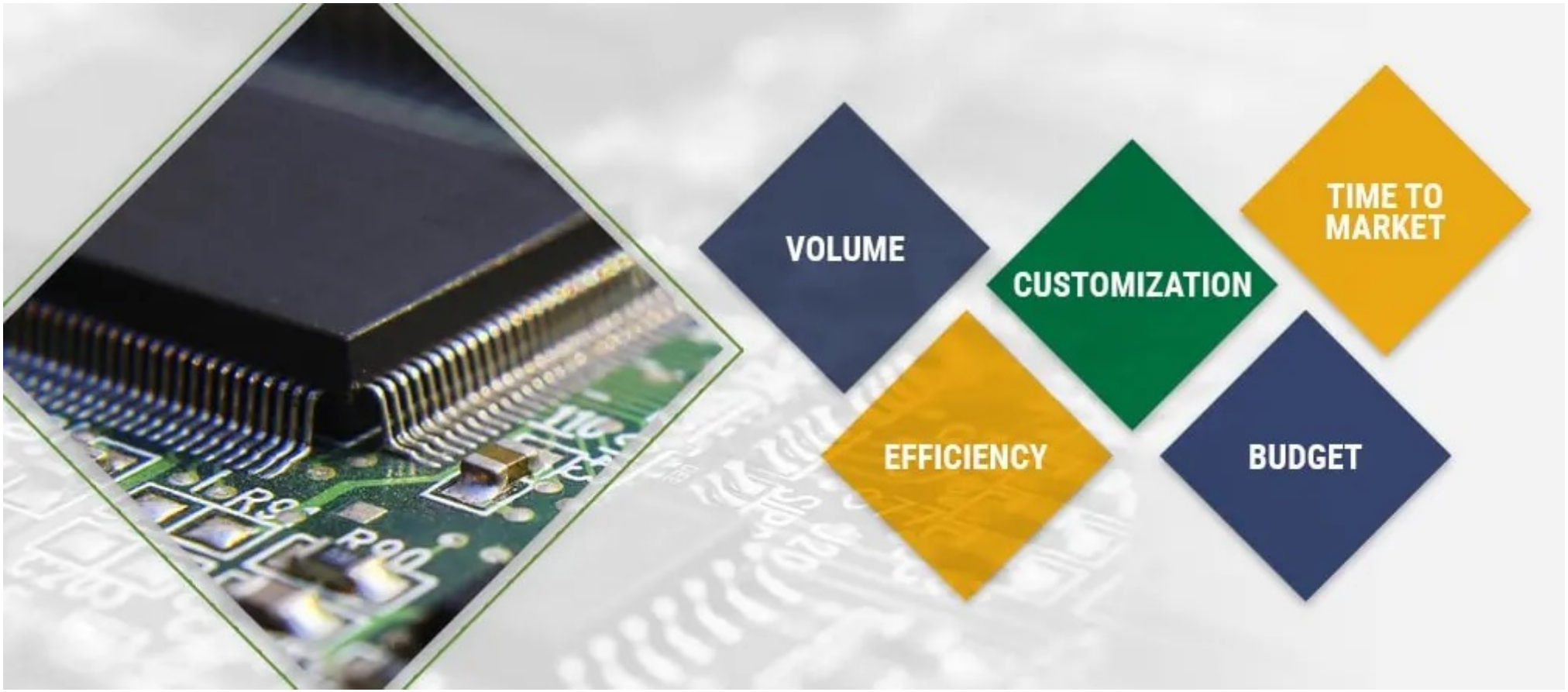
- **Speedup**
- Reduced **power consumption**
- Lower **latency**
- Increased parallelism and bandwidth
- Better utilization of **area** and functional components available on an integrated circuit

Disadvantages

- Lower **flexibility**
- **Higher costs** of functional verification and times to market

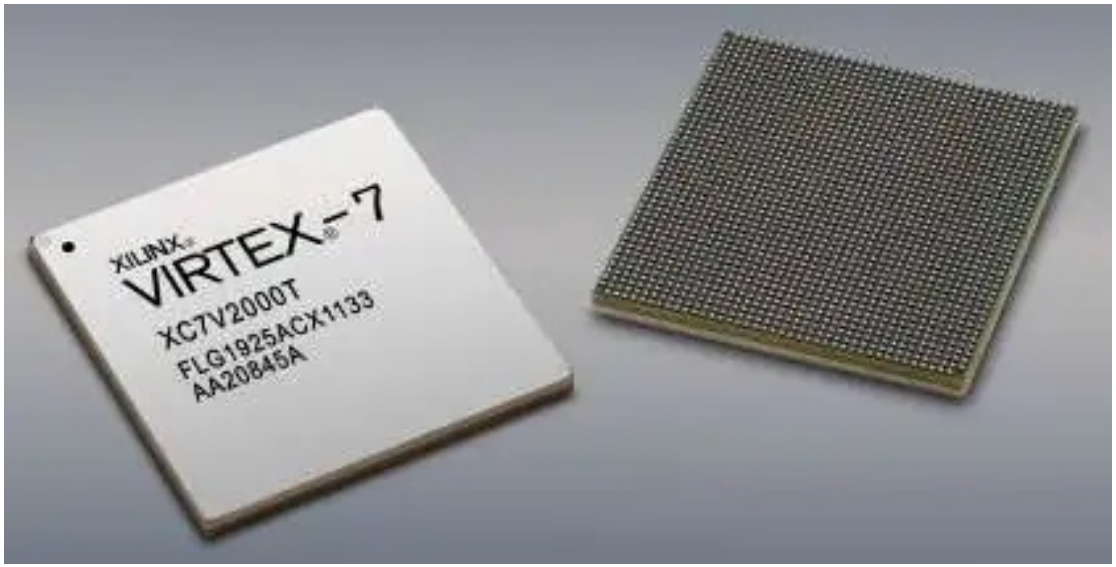
Brandeis

ASICs



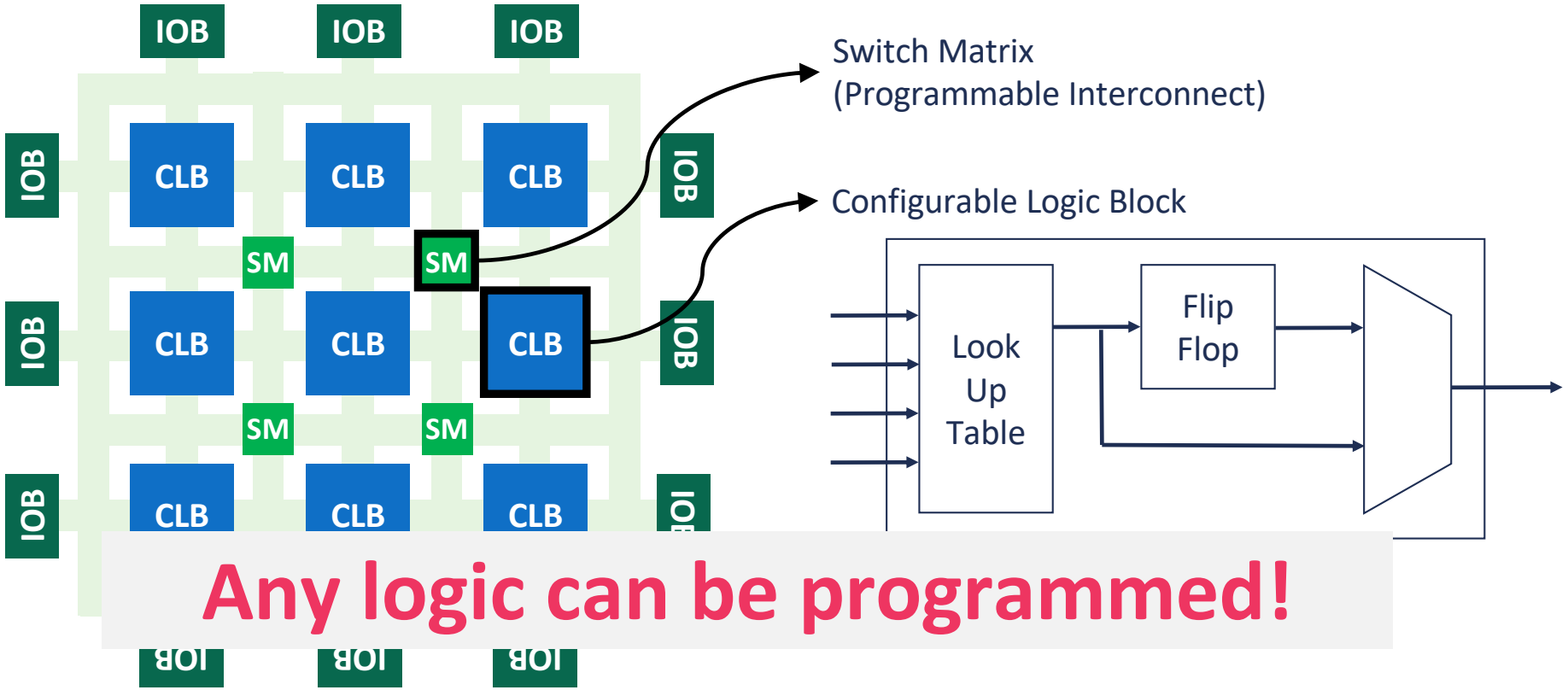
Programmable Logic

Field Programmable Gate Arrays (FPGAs)



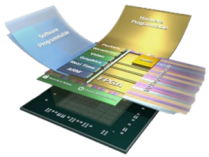
Brandeis

Architecture of Programmable Logic



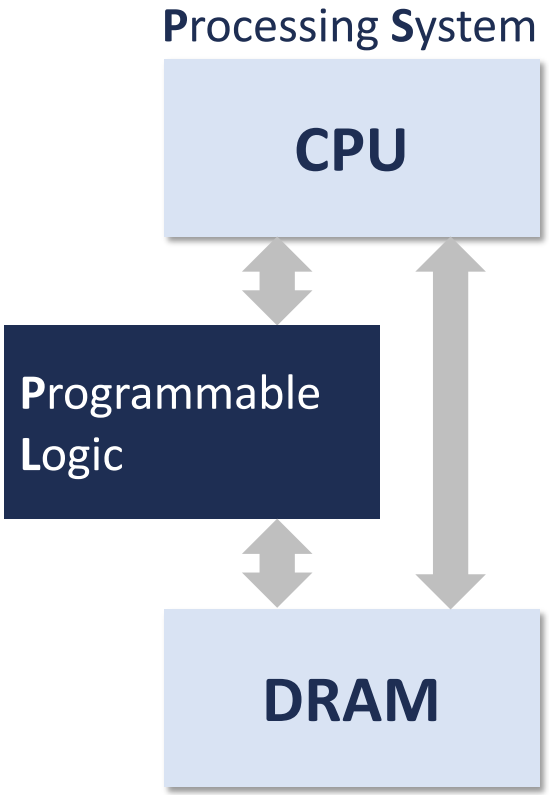
Any logic can be programmed!

Brandeis



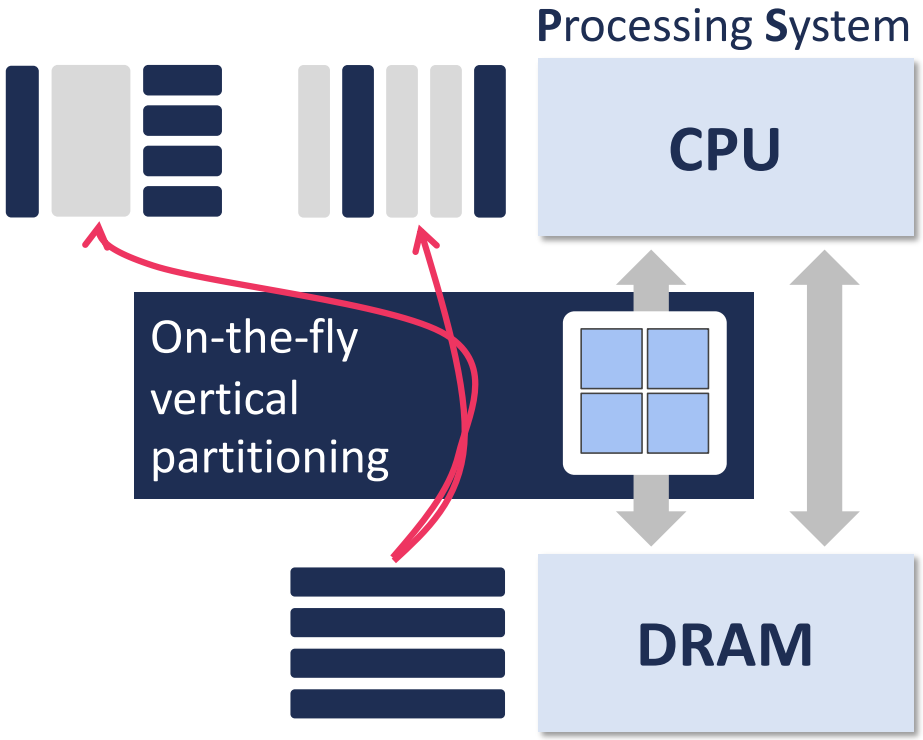
AMD XILINX UltraScale+ ENZIAN intel AGILEX

PS-PL Platforms



Brandeis

Relational Memory Engine

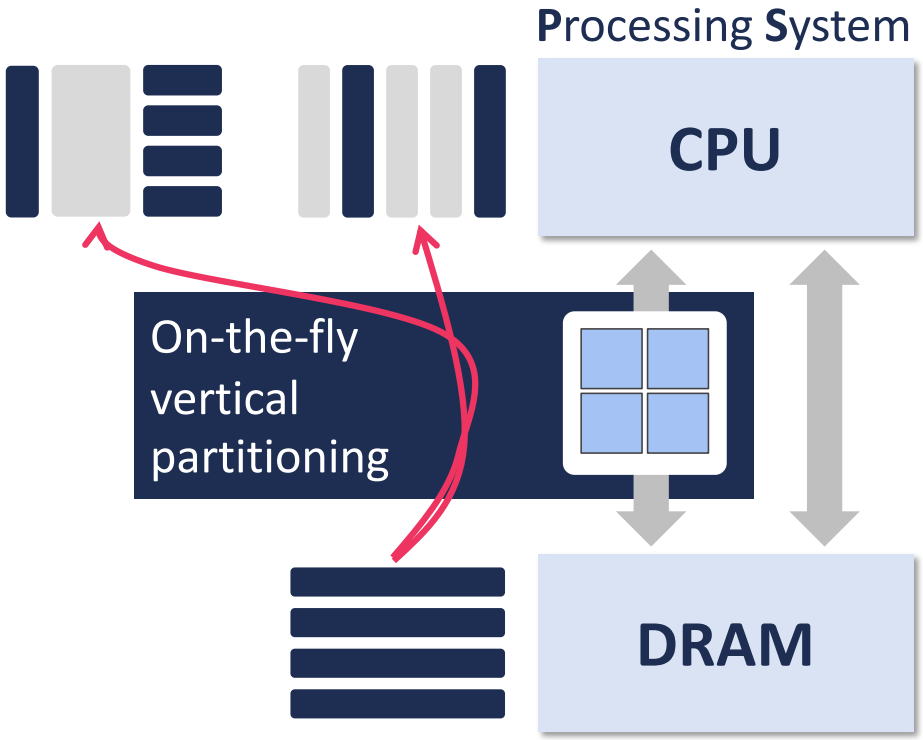


Brandeis

Q1: how to build?

Q2: how to use?

Relational Memory Engine



Brandeis

ephemeral variable

a simple, lightweight programming abstraction

to use Relational Memory

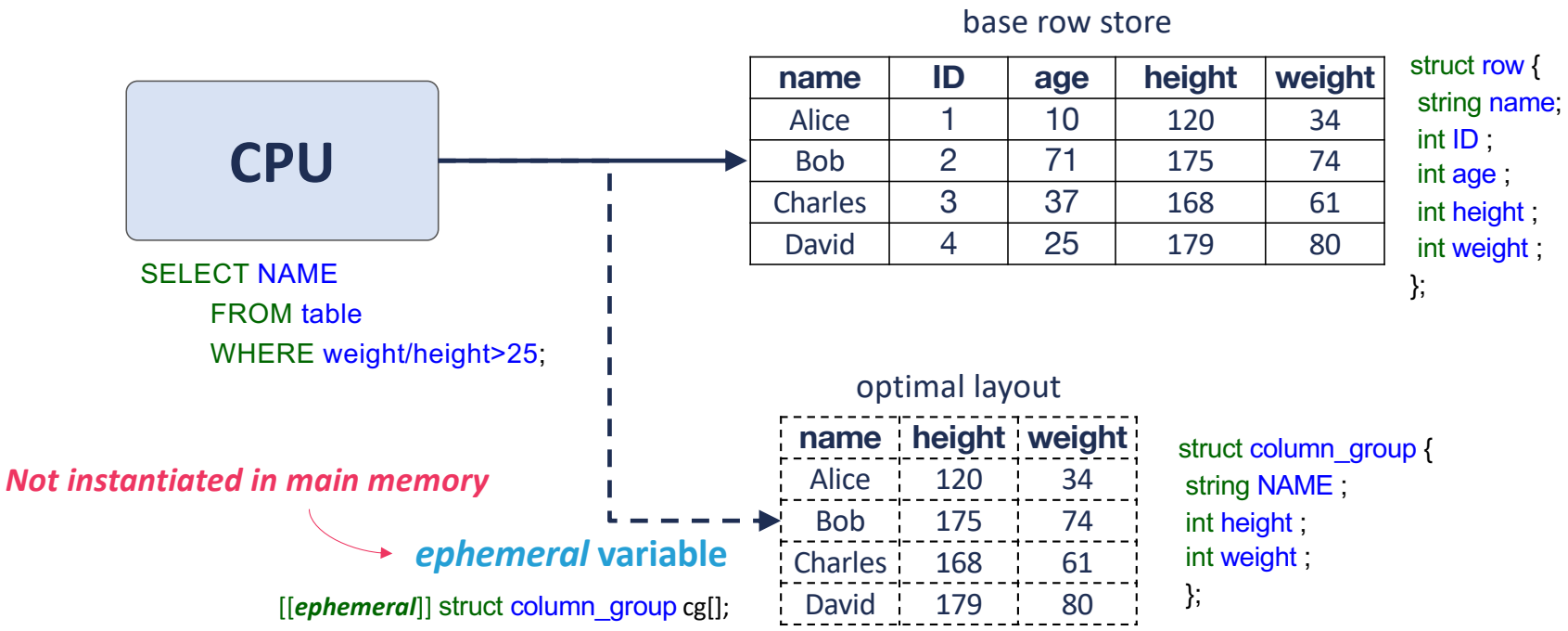
leads to normal memory accesses

```
struct row table[ ];
[[ephemeral]] struct col_group cg[ ];
```

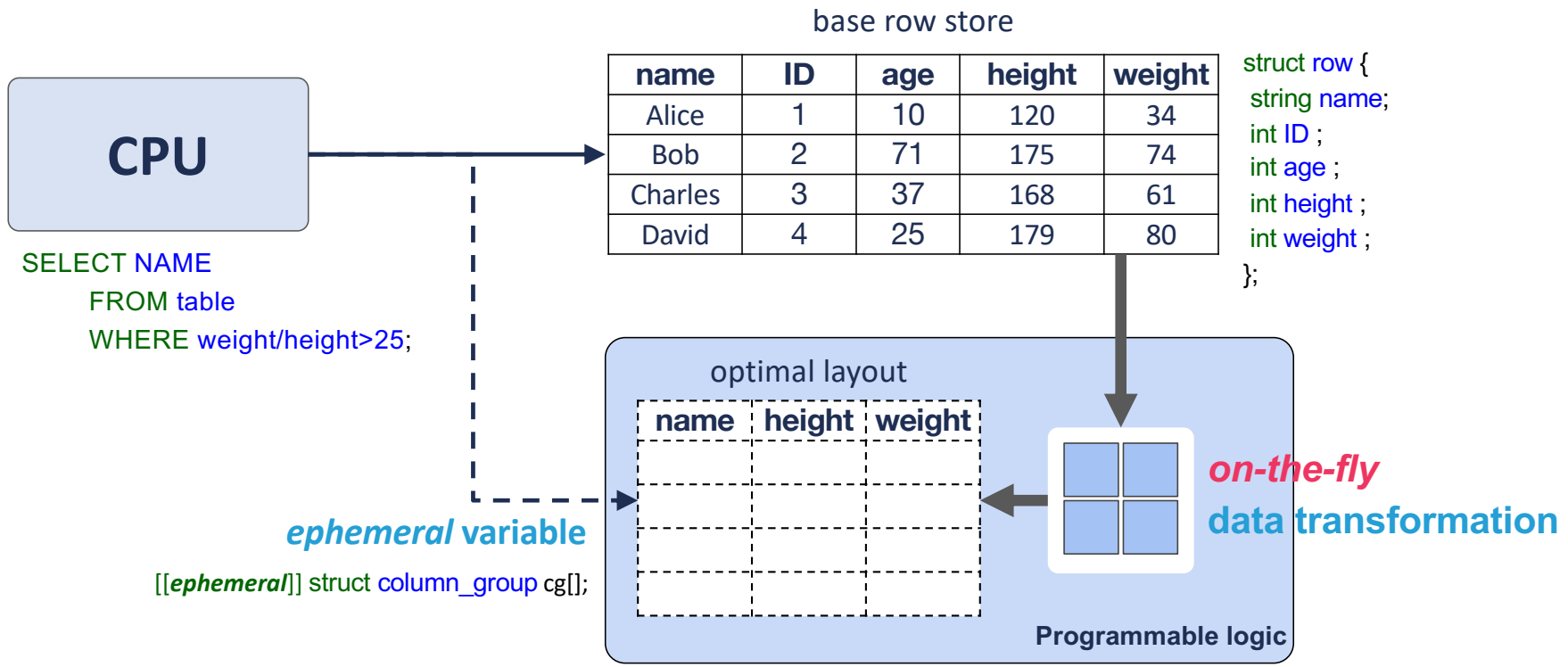
fake address that CPU thinks it exists

intercepted by RME

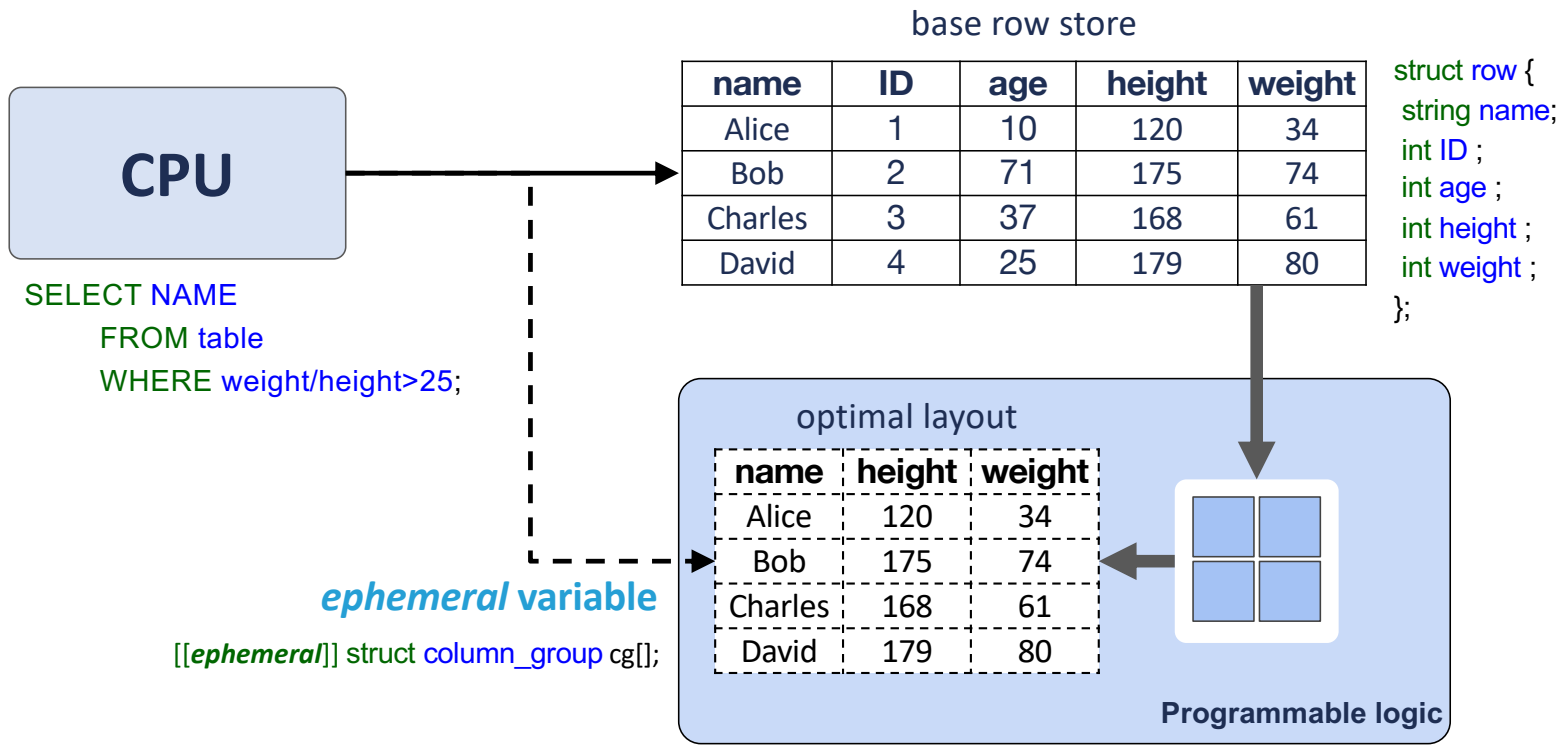
Brandeis


base row store
CPU
name ID age height weight
Alice 1 10 120 34
Bob 2 71 175 74
Charles 3 37 168 61
David 4 25 179 80
struct row {
string name;
int ID ;
int age ;
int height ;
int weight ;
};
SELECT NAME
FROM table
WHERE weight/height>25;
optimal layout
Not instantiated in main memory
ephemeral variable
[[ephemeral]] struct column_group cg[];
name height weight
Alice 120 34
Bob 175 74
Charles 168 61
David 179 80
struct column_group {
string NAME ;
int height ;
int weight ;
};

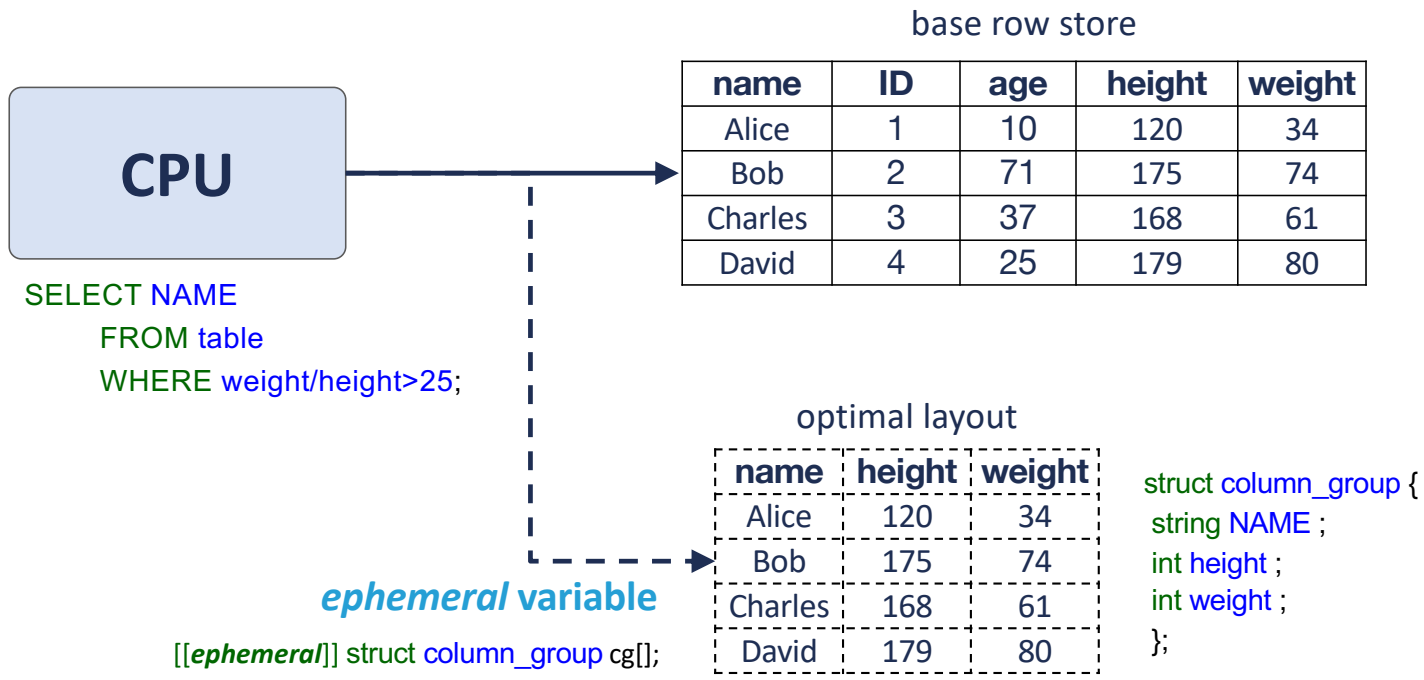
Brandeis


base row store
CPU
name
ID
age
height
weight
Alice
1
10
120
34
Bob
2
71
175
74
Charles
3
37
168
61
David
4
25
179
80
struct row {
string name;
int ID ;
int age ;
int height ;
int weight ;
};
SELECT NAME
FROM table
WHERE weight/height>25;
optimal layout
name
height
weight
ephemeral variable
[[ephemeral]] struct column_group cg[];
on-the-fly
data transformation
Programmable logic

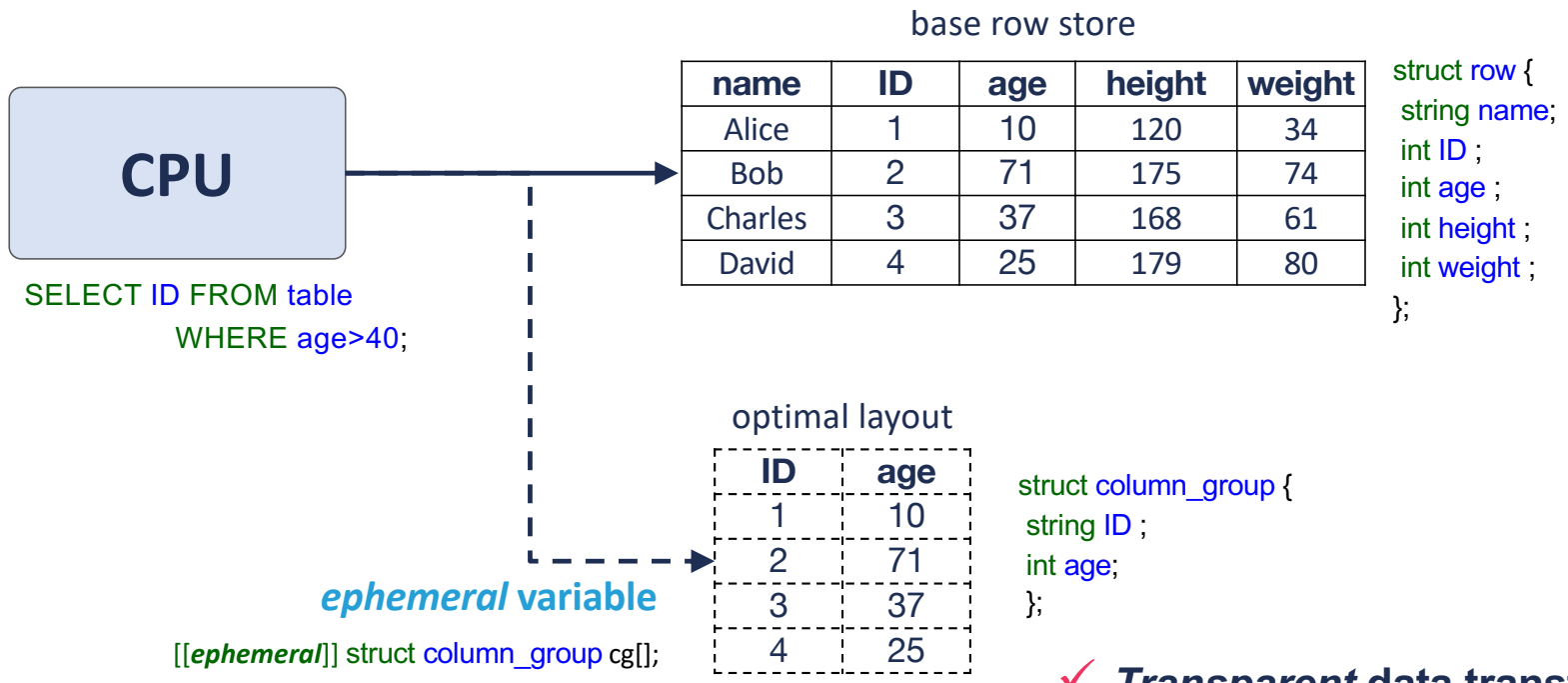
Brandeis


base row store
CPU
name ID age height weight
Alice 1 10 120 34
Bob 2 71 175 74
Charles 3 37 168 61
David 4 25 179 80
struct row {
string name;
int ID ;
int age ;
int height ;
int weight ;
};
SELECT NAME
FROM table
WHERE weight/height>25;
optimal layout
name height weight
Alice 120 34
Bob 175 74
Charles 168 61
David 179 80
Programmable logic
ephemeral variable
[[ephemeral]] struct column_group cg[];

Brandeis


base row store
CPU
name ID age height weight
Alice 1 10 120 34
Bob 2 71 175 74
Charles 3 37 168 61
David 4 25 179 80
SELECT NAME
FROM table
WHERE weight/height>25;
optimal layout
name height weight
Alice 120 34
Bob 175 74
Charles 168 61
David 179 80
ephemeral variable
[[ephemeral]] struct column_group cg[];
struct column_group {
string NAME ;
int height ;
int weight ;
};

Brandeis



✓ ***Transparent*** **data transformation**

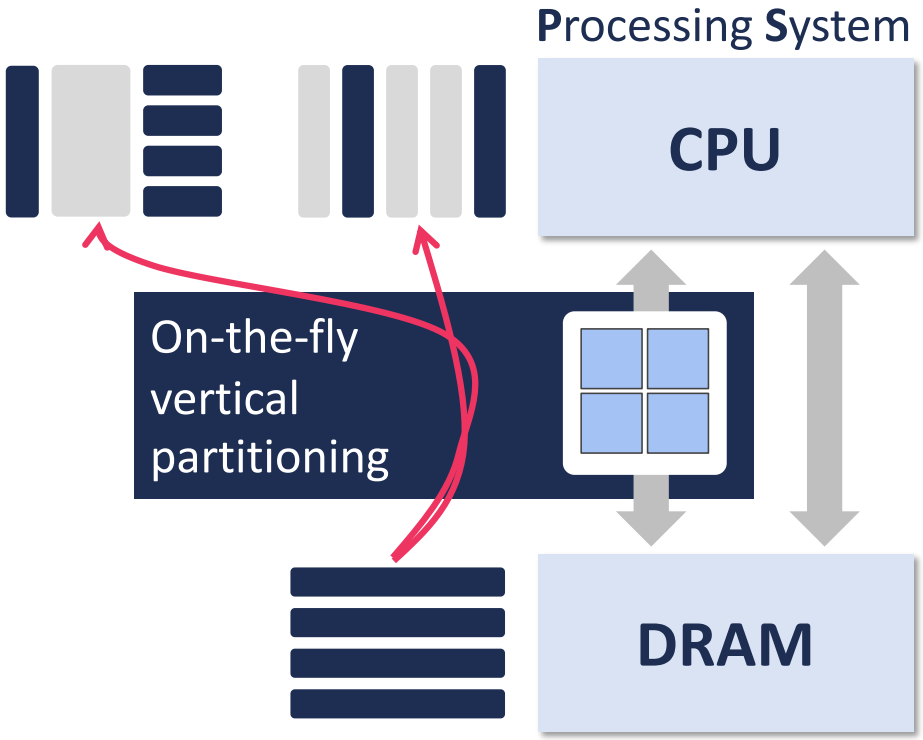
✓ ***Optimal layout***

Brandeis

Q1: how to build?

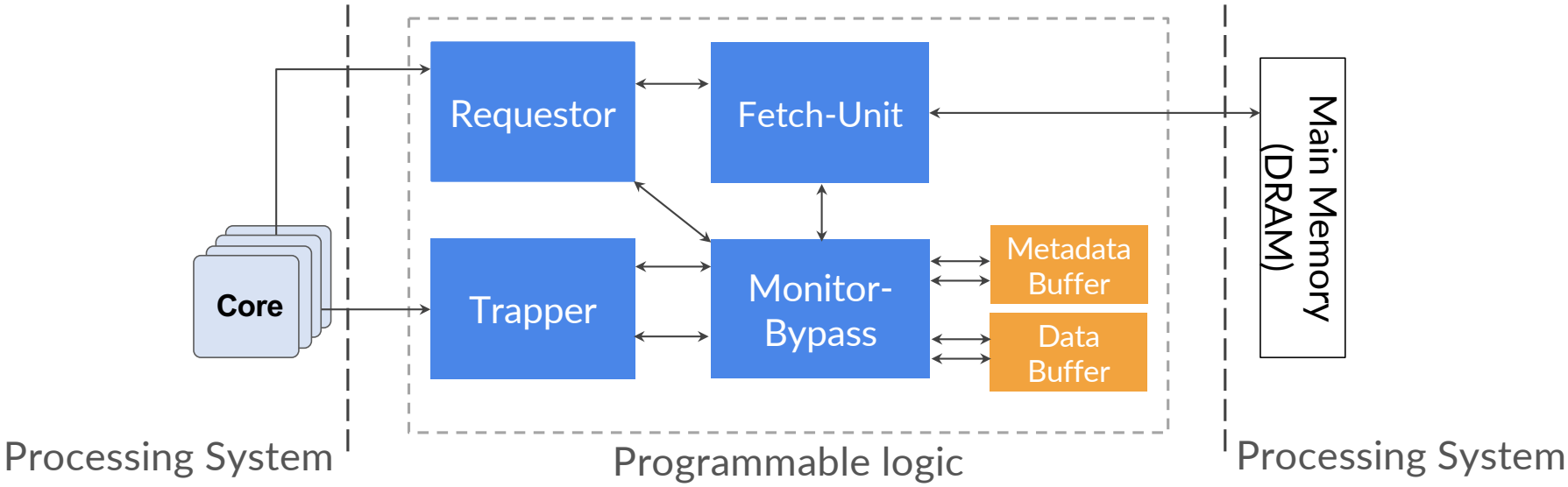
Q2: how to use?

Relational Memory Engine



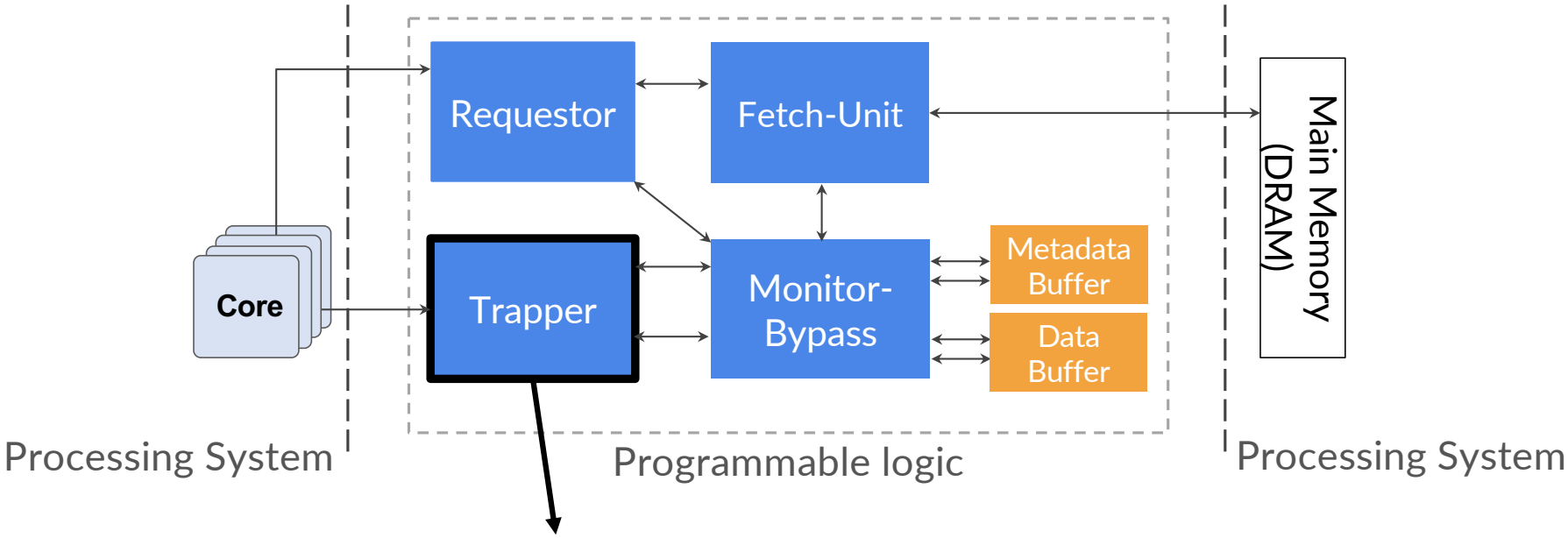
Brandeis

Relational Memory Engine



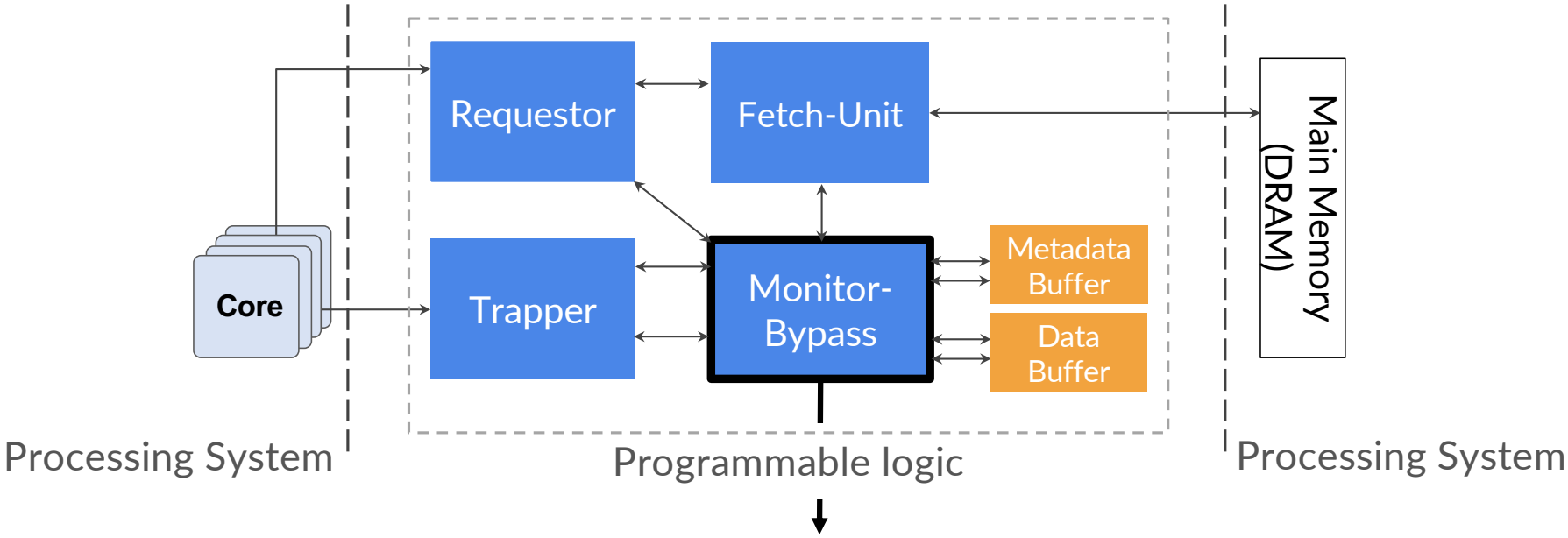
Brandeis

Relational Memory Engine



Brandeis

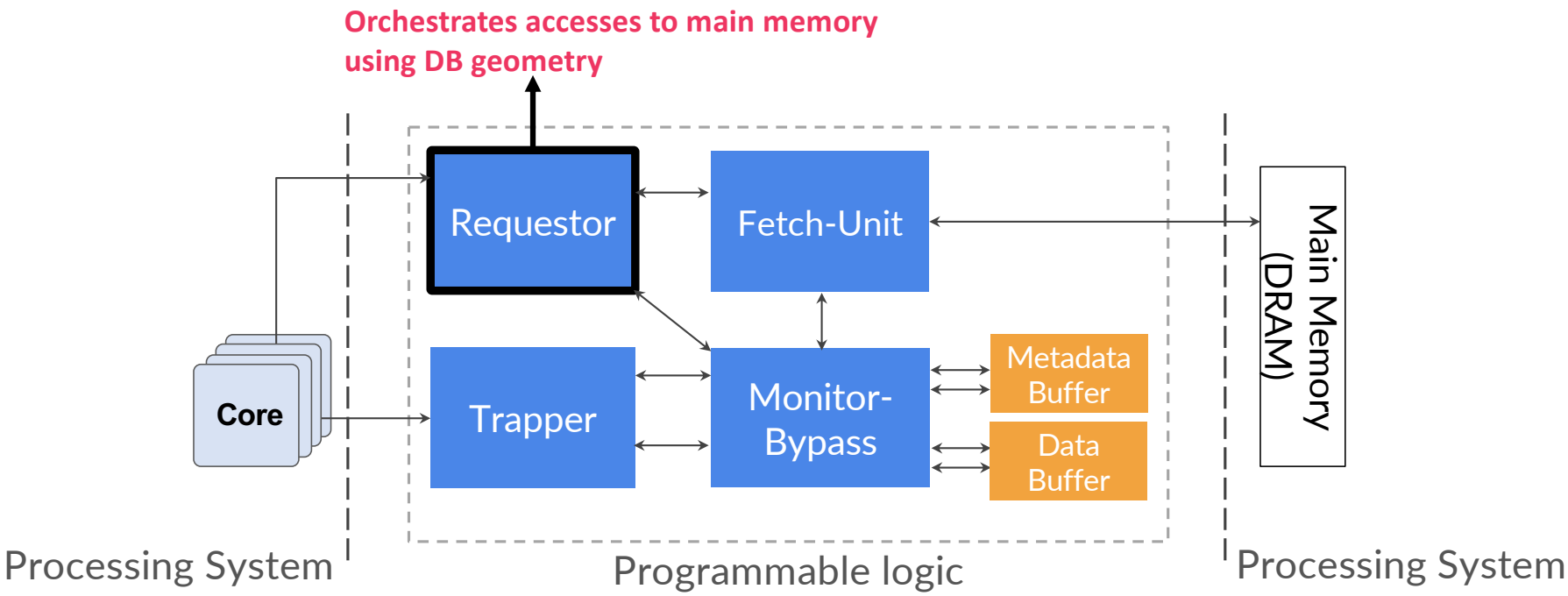
Relational Memory Engine



Monitors the completion of each reorganized cache line

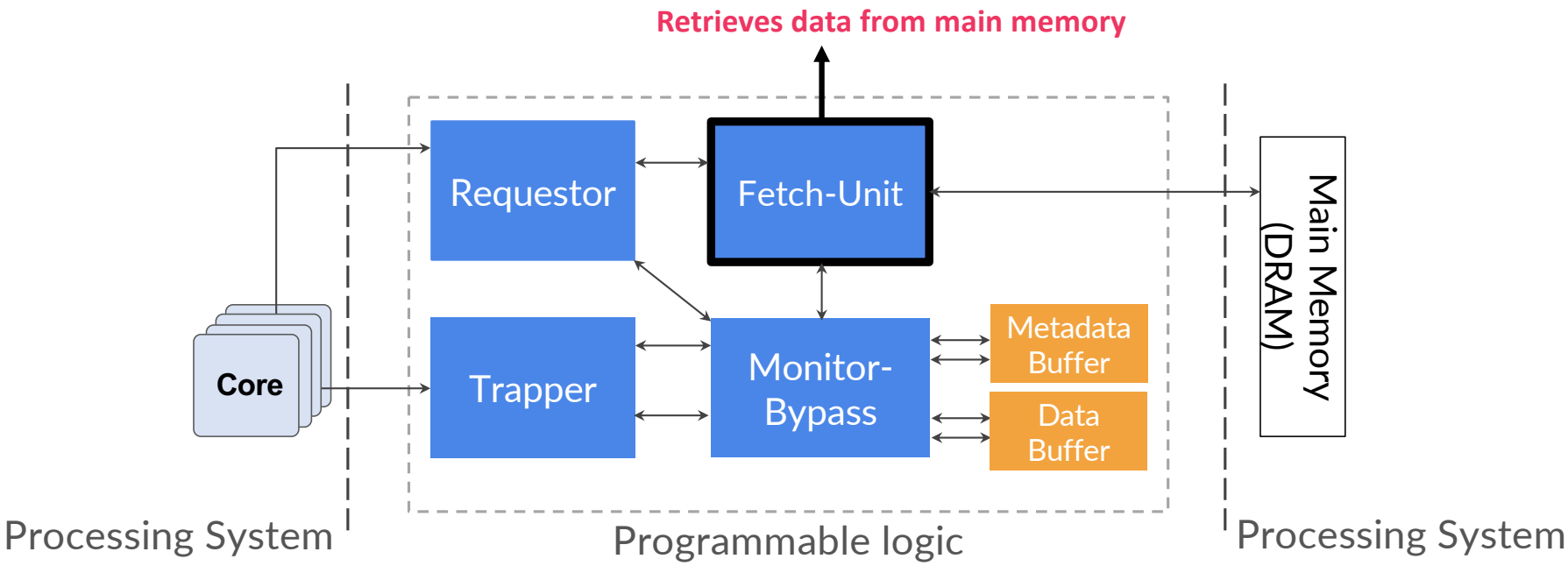
Brandeis

Relational Memory Engine



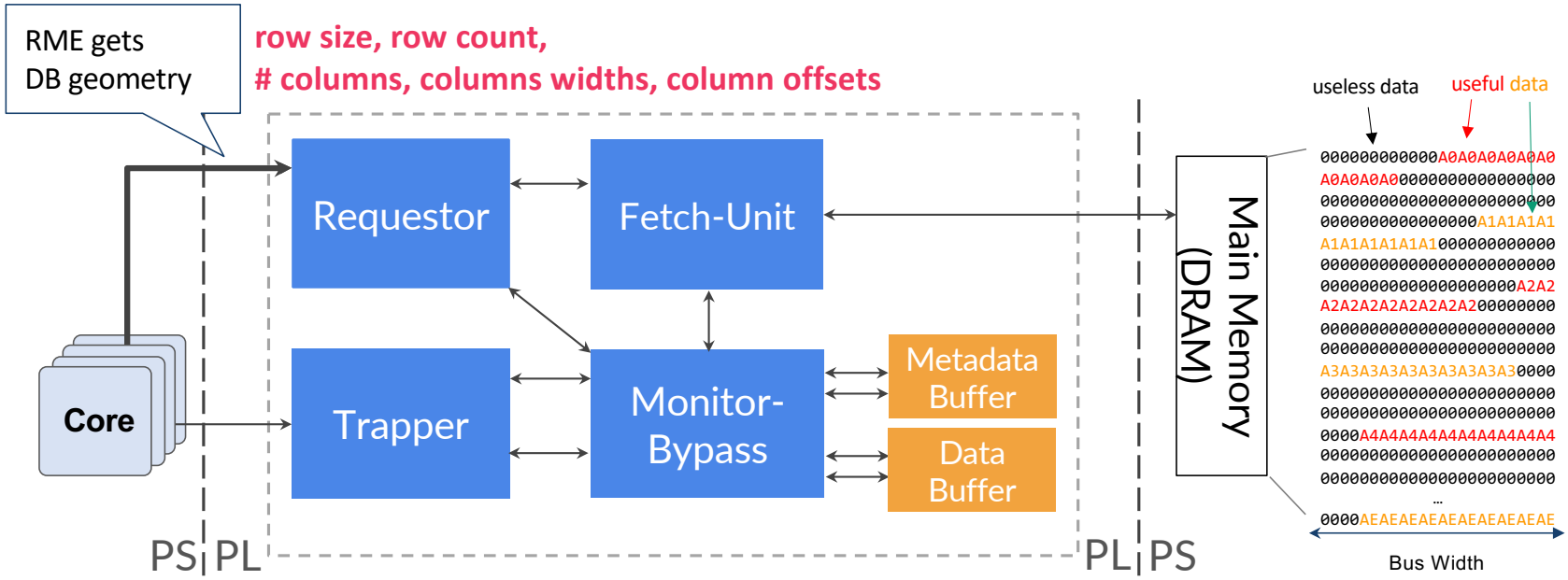
Brandeis

Relational Memory Engine



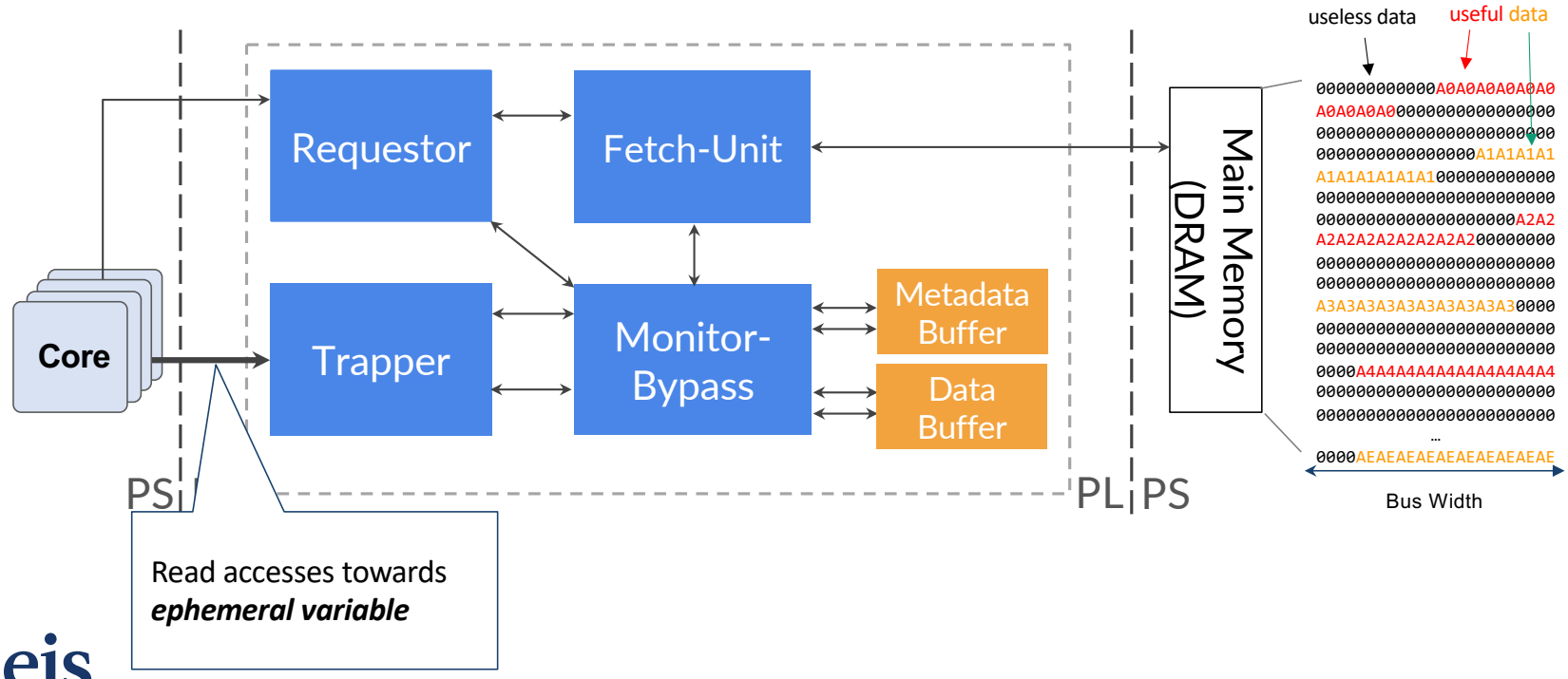
Brandeis

Relational Memory Engine



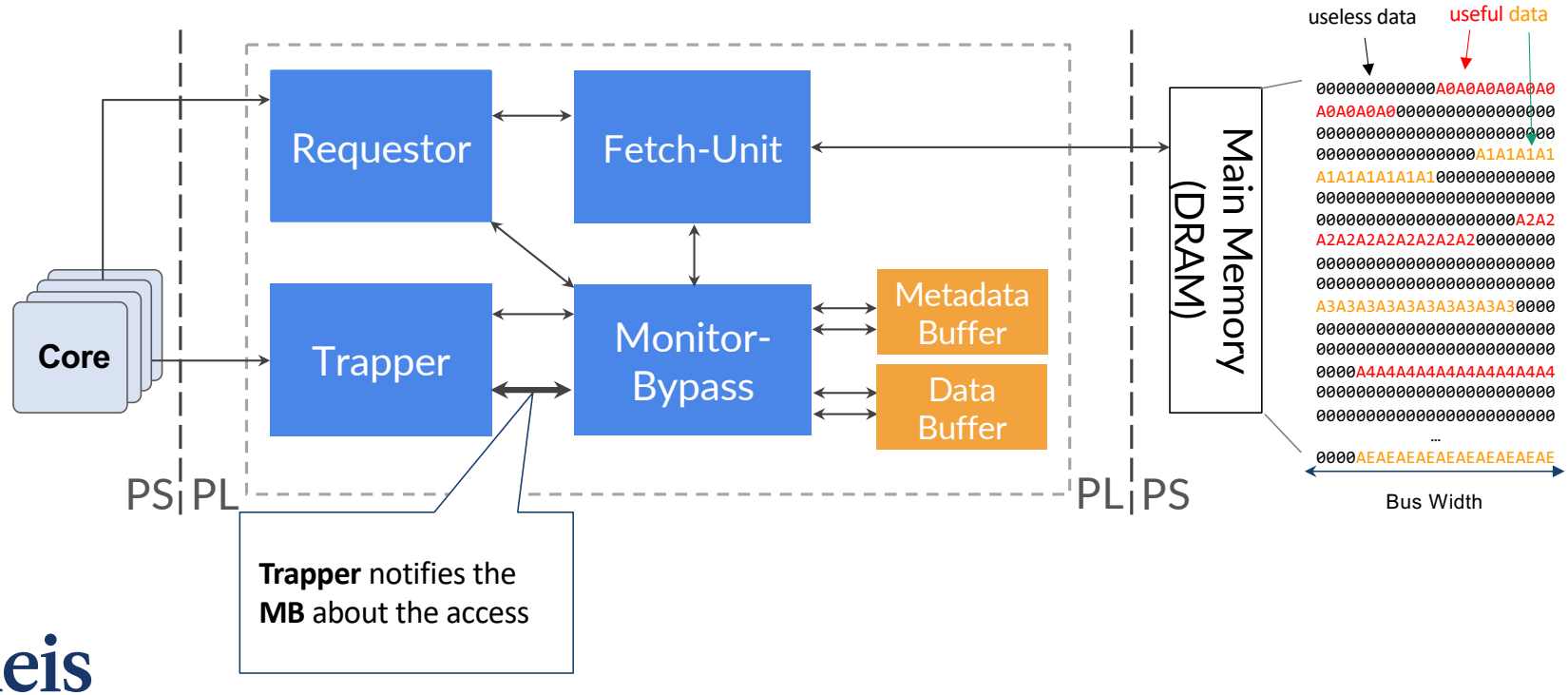
Brandeis

Relational Memory Engine



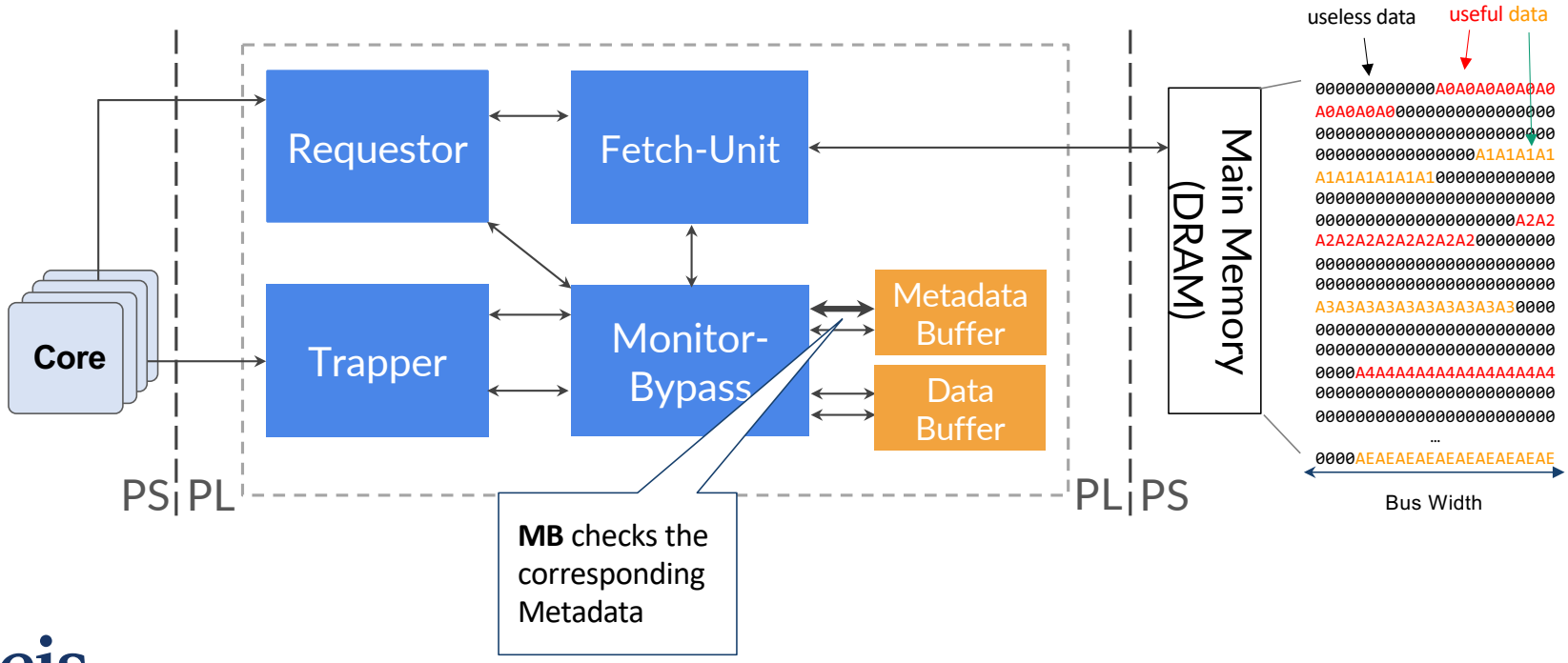
Brandeis

Relational Memory Engine



Brandeis

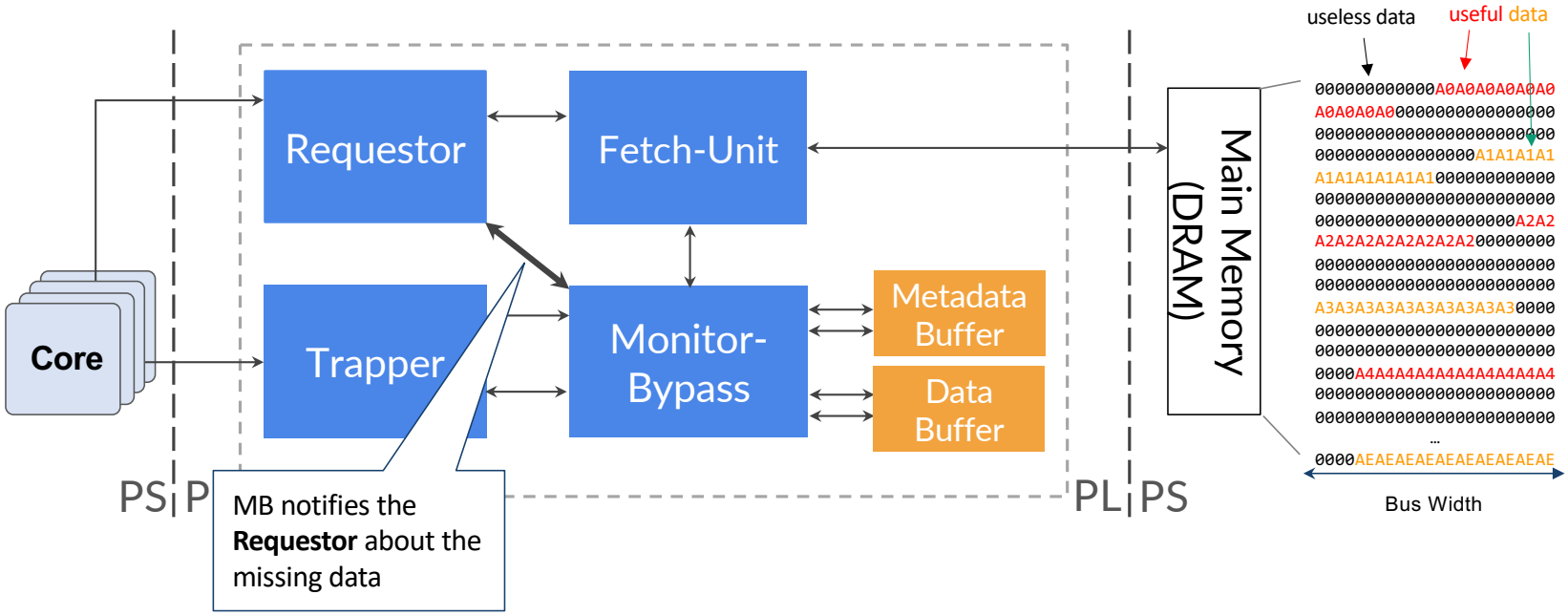
Relational Memory Engine



Brandeis

Relational Memory Engine

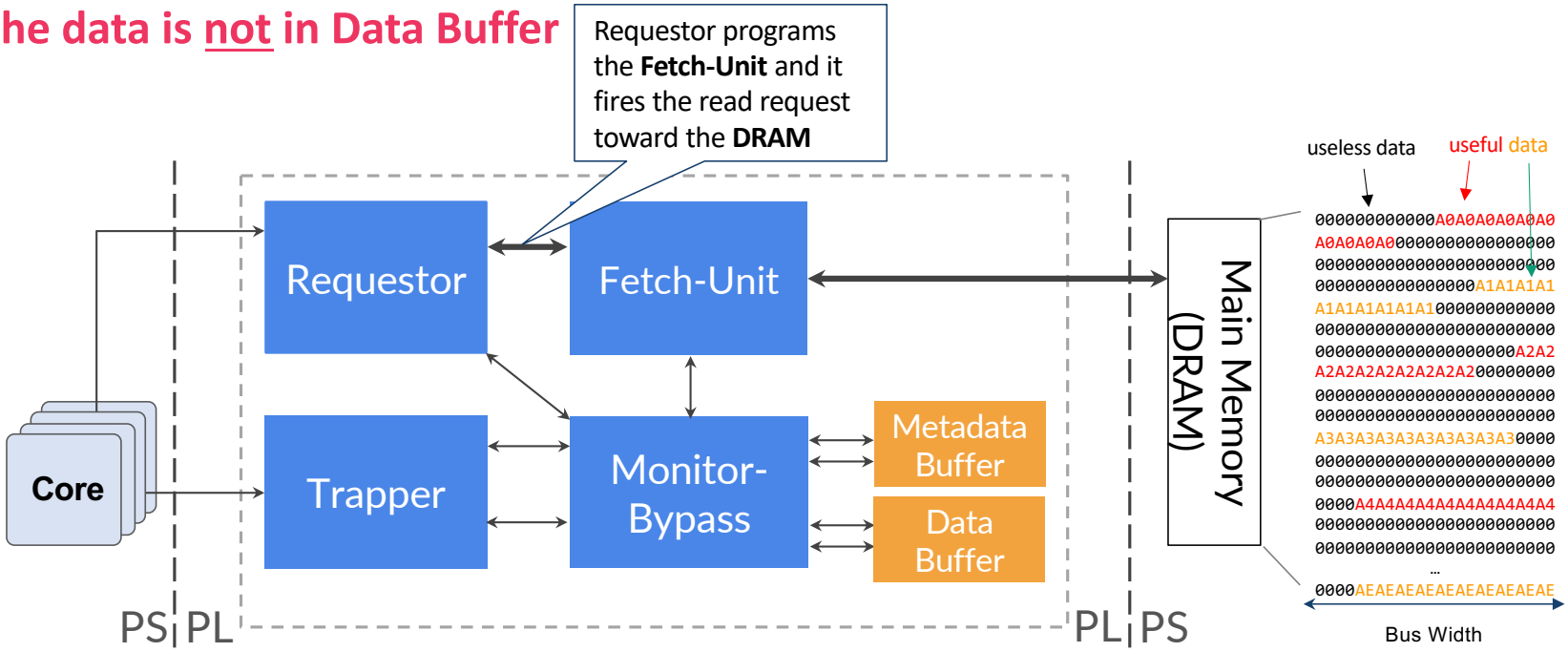
When the data is not in Data Buffer



Brandeis

Relational Memory Engine

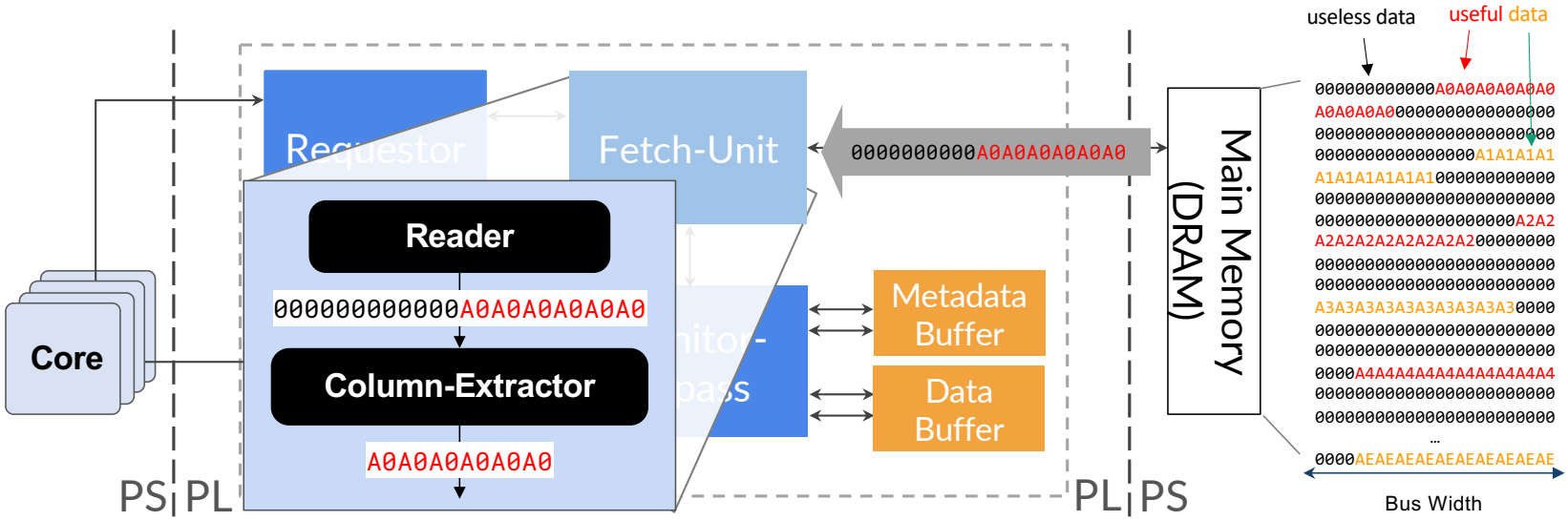
When the data is not in Data Buffer



Brandeis

Relational Memory Engine

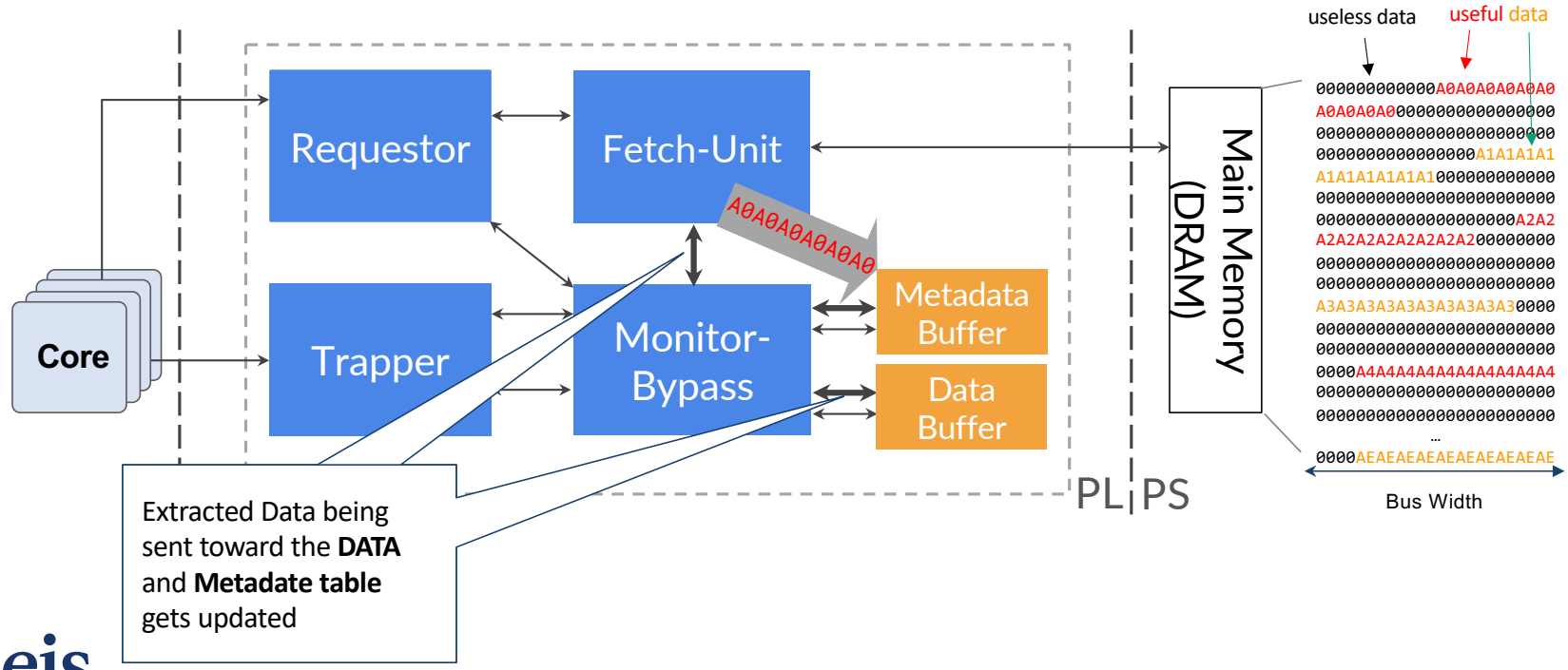
When the data is not in Data Buufer



Brandeis

Relational Memory Engine

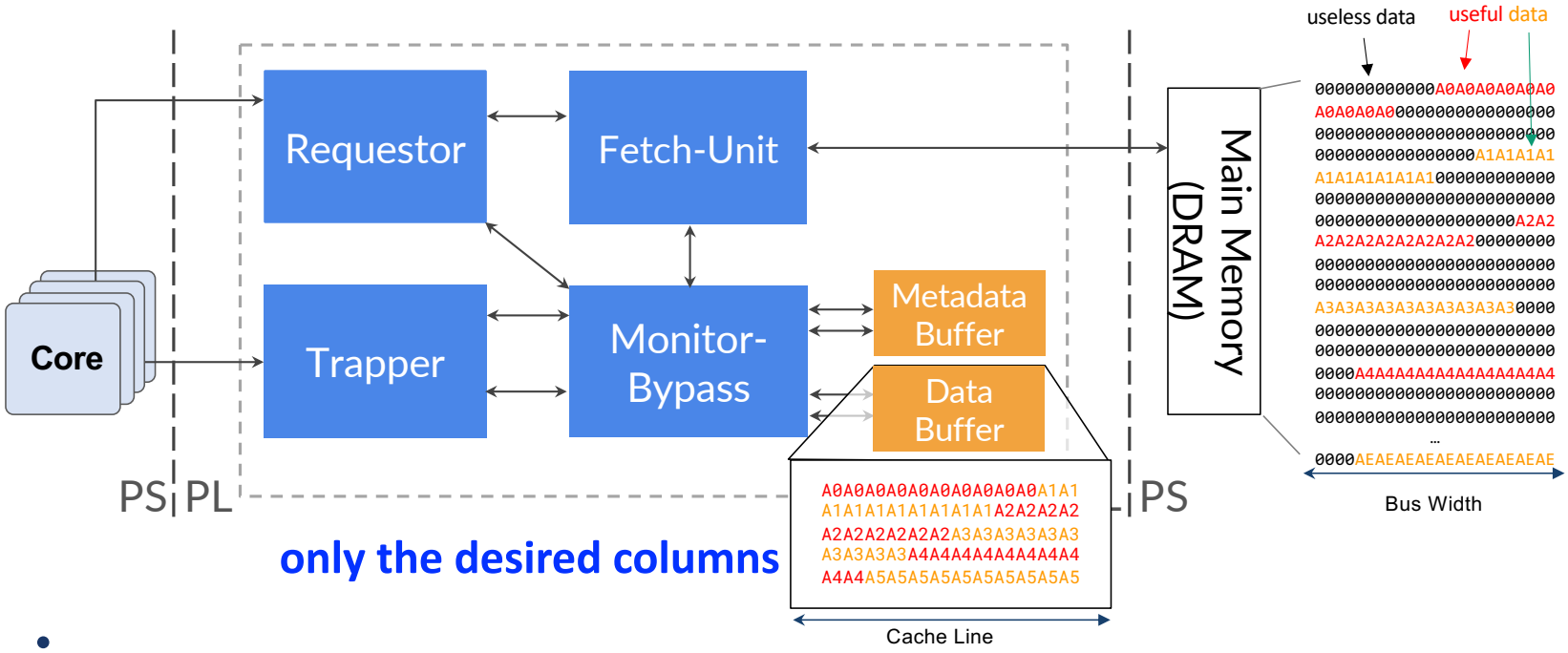
When the data is not in Data Buffer



Brandeis

Relational Memory Engine

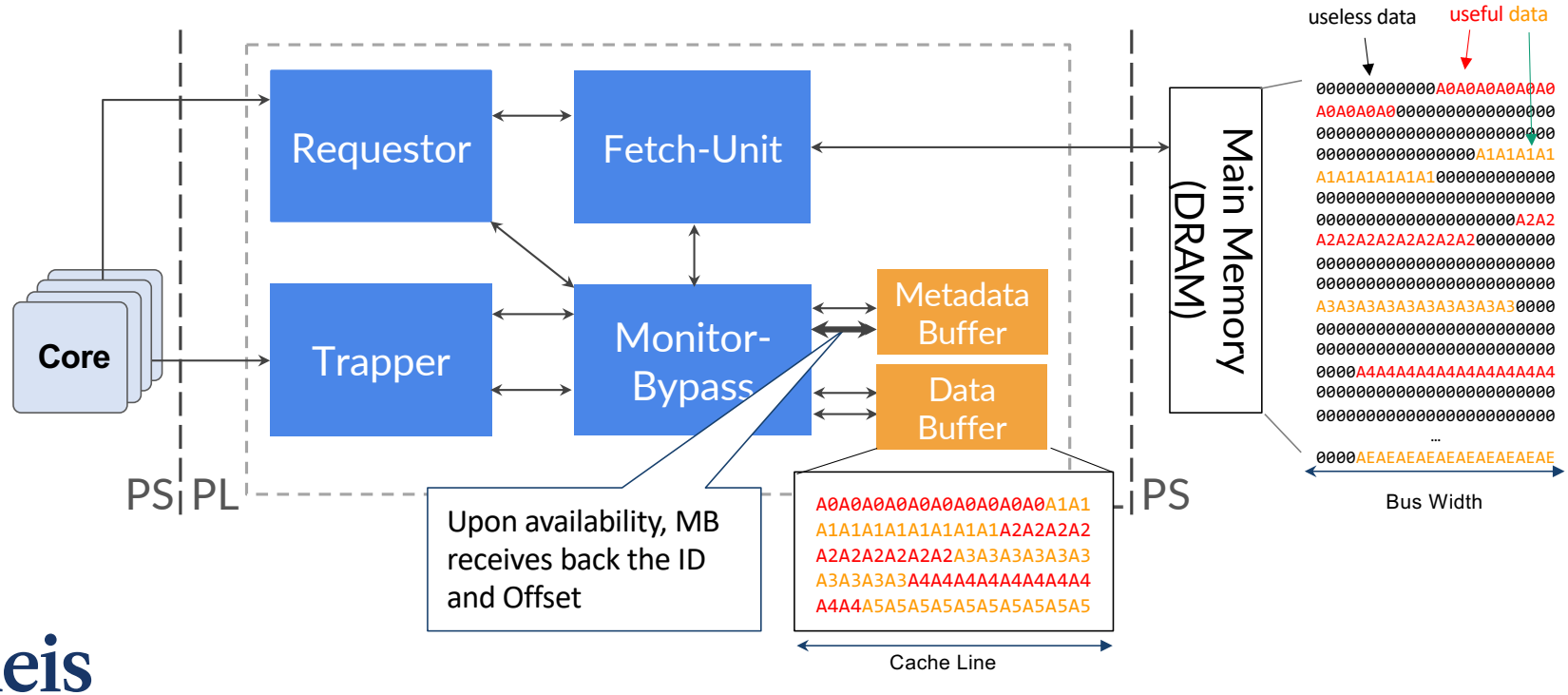
When the data is not in Data Buffer



Brandeis

Relational Memory Engine

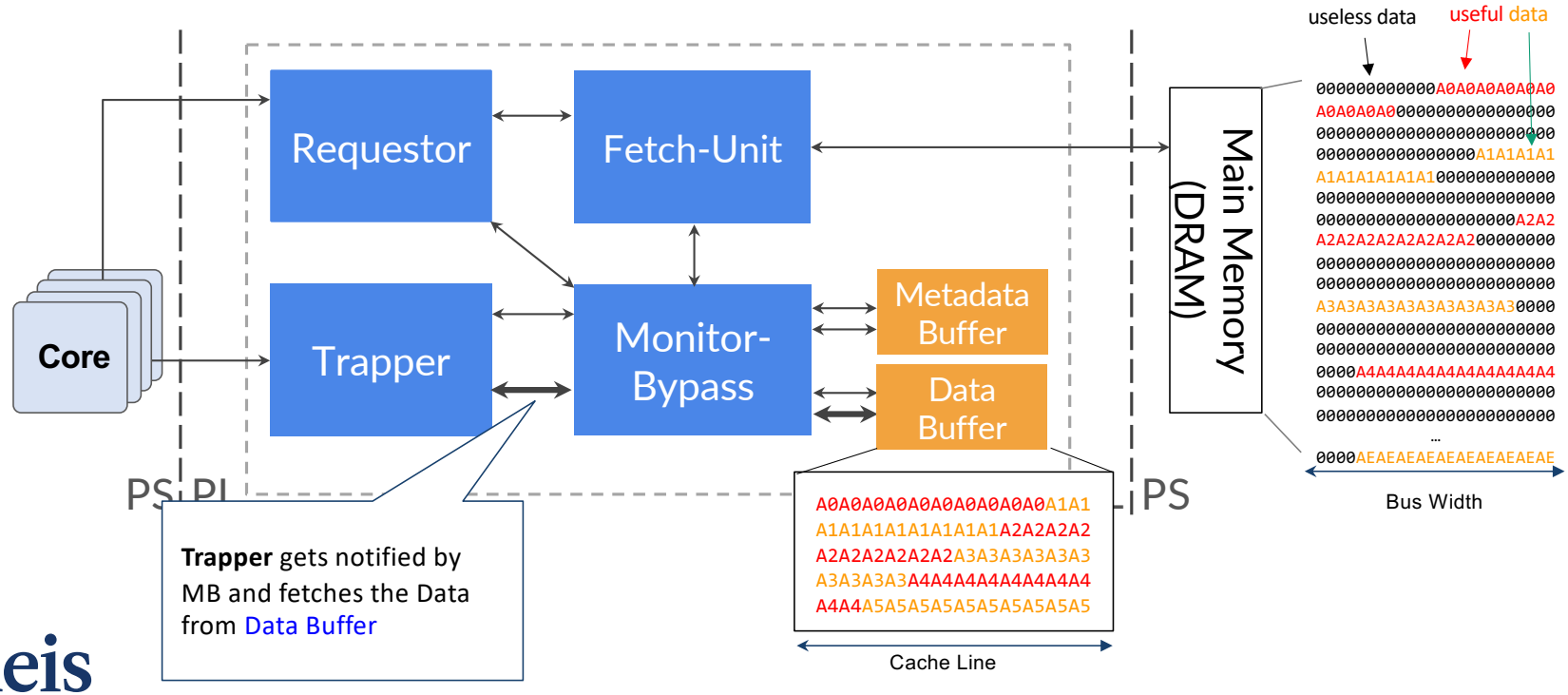
When the data is already in Data Buffer



Brandeis

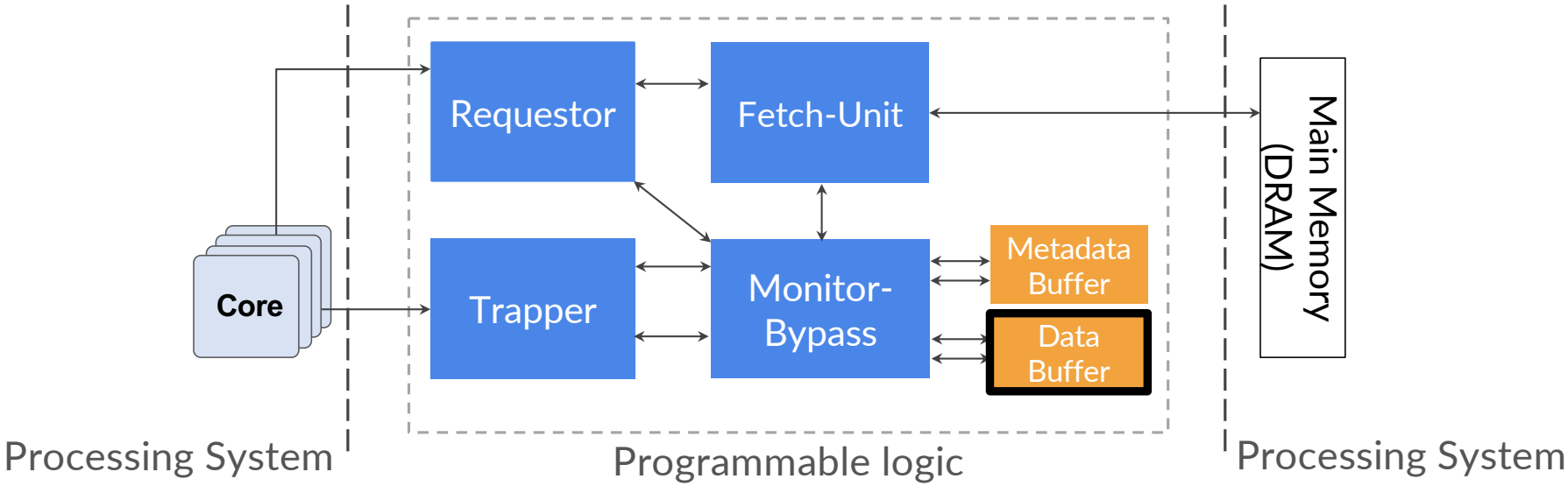
Relational Memory Engine

When the data is already in Data Buffer



Brandeis

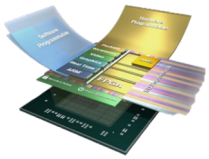
Relational Memory Engine



2MB << Data size

Brandeis

Target Platform

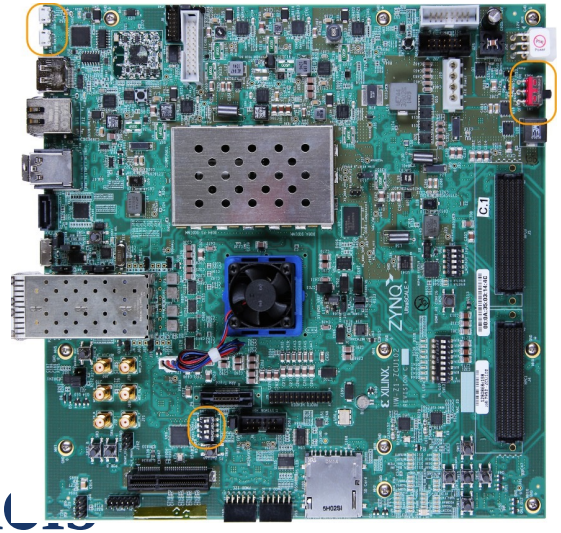


AMD XILINX

UltraScale+

ZCU102 platform

- CPUs : 4x ARM Cortex-A53
- L1/L2 Cache : 32+32KB I+D / 1 MB
- PS Frequency : 1.5 GHz
- PL Frequency : 100MHz



Resources	Utilization (%)
LUT	2.78
FF	0.68
DSP	0.08
BRAM	60.69

area utilization less than 3%

Relational Memory Benchmark

Q1: SELECT A1 , A2 , ... , Ak FROM S; ⇒ projection

Q2: SELECT A1 , A2 , ... , Ak FROM S WHERE C1, C2, … ,Ci; ⇒ both projection & selection

Q3: SELECT AVG (A1) FROM S WHERE A3 < k GROUP BY A2; ⇒ group by

Q4: SELECT S.A1 , R.A3 FROM S JOIN R ON S.A2 = R.A2; ⇒ join over two tables

Approach tested

ROW : Direct row-wise access

COL : Direct columnar access

} **Processing System**

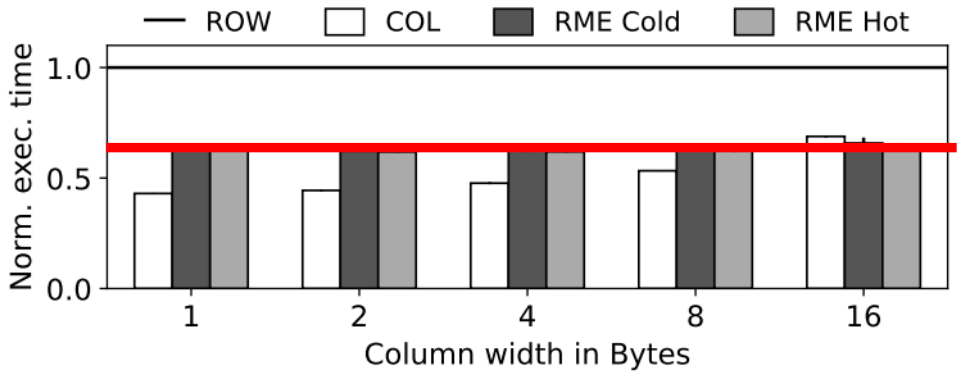
RME : using Relational Memory Engine → **Slow FPGA** (100MHz)

Brandeis

How big is the overhead of fetching the data?

RME Cold vs. Hot

SELECT A1, A3, A5 FROM S;



RME is comparable with directly accessing a single column!

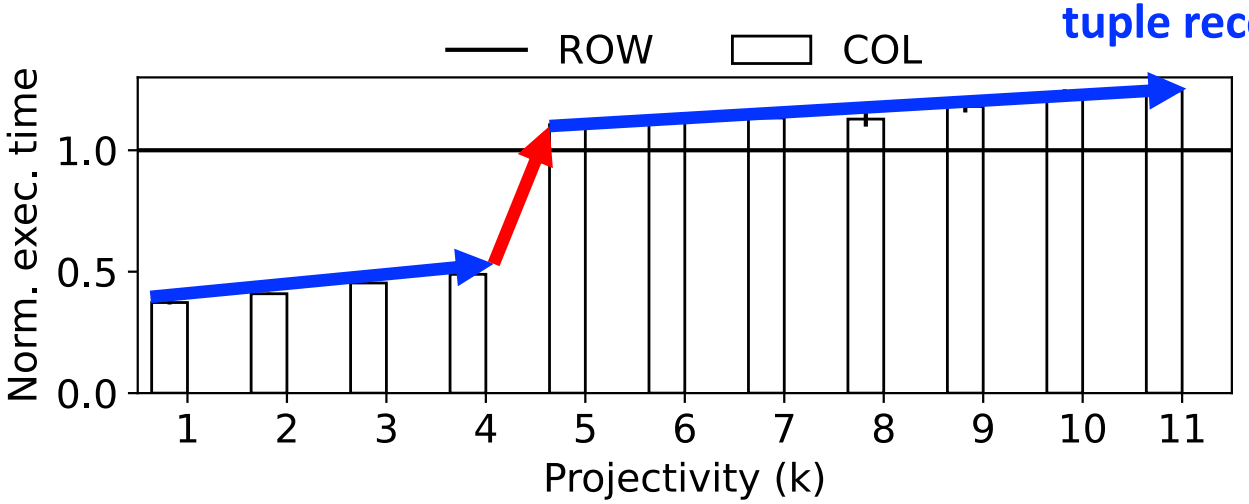
RME Cold has virtually the same performance as RME Hot!

Brandeis

Row size: 64 Bytes

Queries Varying Projectivity

Q1: SELECT A1 , A2 , … , Ak FROM S;



tuple reconstruction cost

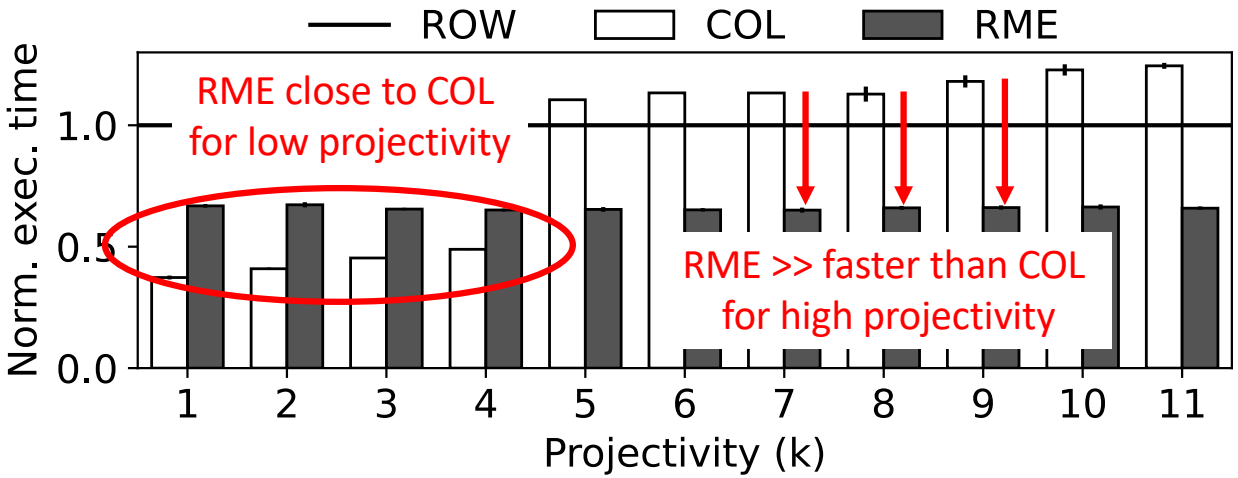
prefetcher supports up to four parallel streams

Row size: 64 Bytes, Column size: 4 Bytes

Brandeis

Queries Varying Projectivity

Q1: SELECT A1 , A2 , … , Ak FROM S;



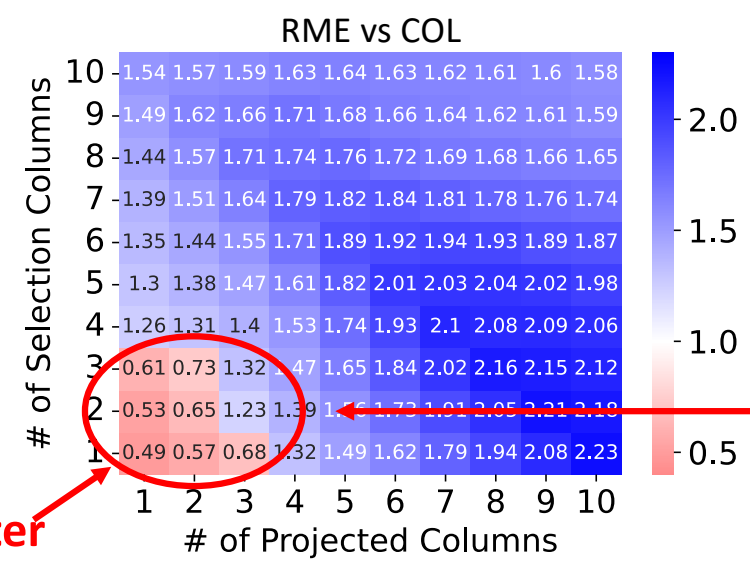
RME provides stable performance irrespectively of projectivity

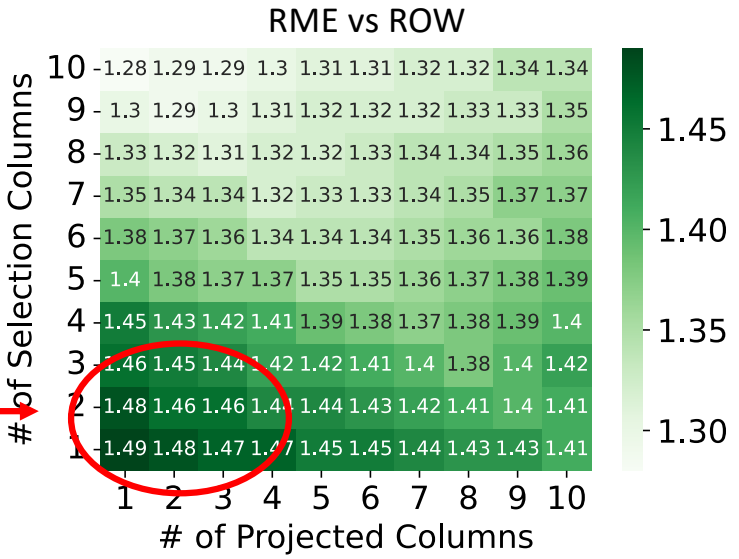
Row size: 64 Bytes, Column size: 4 Bytes

Brandeis

RME for Multiple Selection and Projection Attributes

Q3: SELECT A1 , A2 , ... , Ak FROM S WHERE C1, C2, ... ,Ci; Row size: 64 Bytes, Column size: 4 Bytes





COL faster

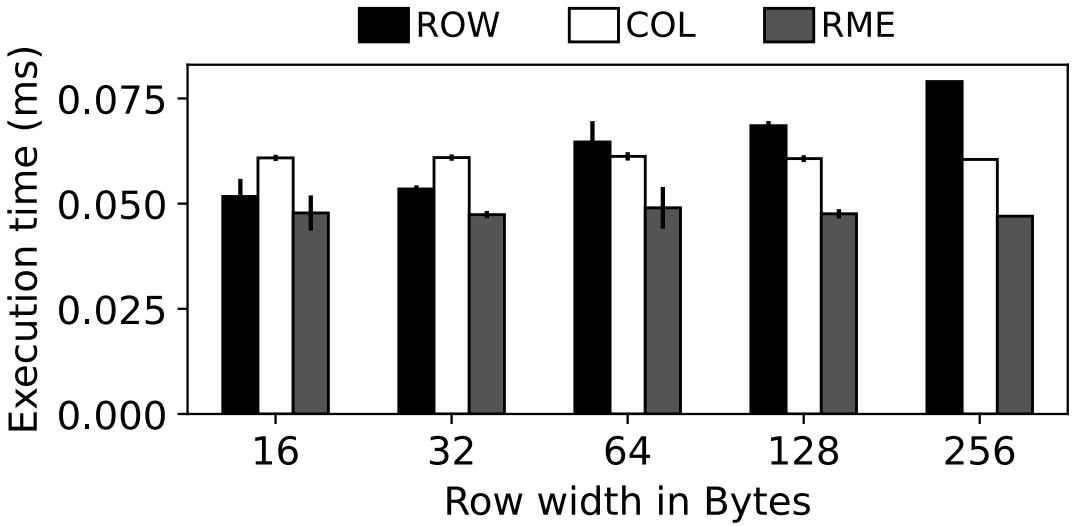
RME can be up to 2.23× faster than columnar access

RME *always* outperforms row access by being 1.3 – 1.5× faster

Brandeis

Group by

Q4: SELECT AVG (A1) FROM S WHERE A3 < k GROUP BY A2; Selectivity: 10%



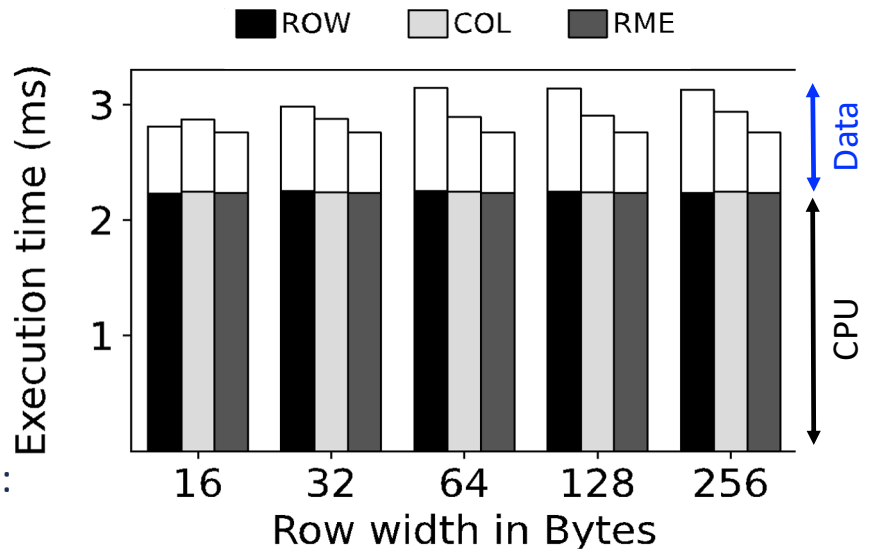
RME outperforms both ROW and COL

Brandeis

Column size: 4 Bytes

Join Over Two Tables

Q4: SELECT S.A1 , R.A3 FROM S JOIN R ON S.A2 = R.A2;



Column size:

RME reduces data movement up to 41%

Brandeis

RME Scales with Data Size

TPC-H Q1

```
SELECT l_returnflag, l_linestatus,
   SUM(l_quantity), SUM(l_extendedprice),
   SUM(l_extendedprice*(1-l_discount)),
   SUM(l_extendedprice*(1-l_discount)*(1+l_tax)),
   AVG(l_quantity), AVG(l_extendedprice),
   AVG(l_discount),
   COUNT(*)
FROM lineitem
WHERE
   l_shipdate <= '1998-12-01' - '[DELTA]' day (3)
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

Selectivity: 95%, projectivity: 24%

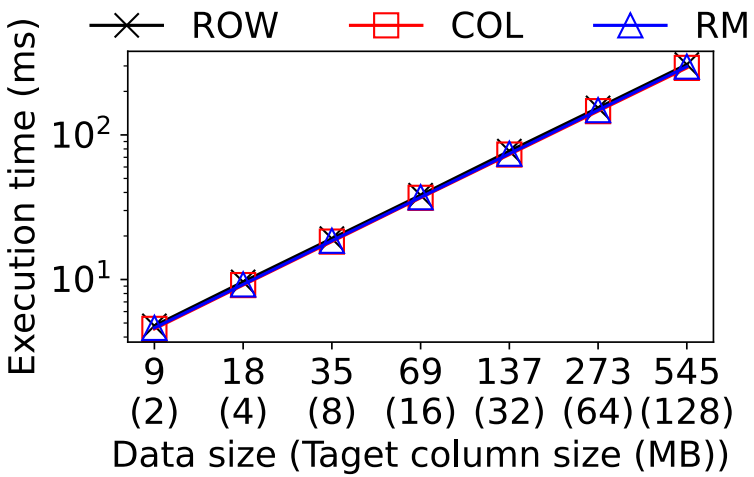
TPC-H Q6

```
SELECT
   SUM(l_extendedprice*l_discount)
FROM lineitem
WHERE
   l_shipdate >= '[DATE]' and
   l_shipdate < '[DATE]' + 1 year and
   l_discount > [DISCOUNT] - 0.01 and
   l_discount < [DISCOUNT] + 0.01 and
   l_quantity < [QUANTITY];
```

Brandeis

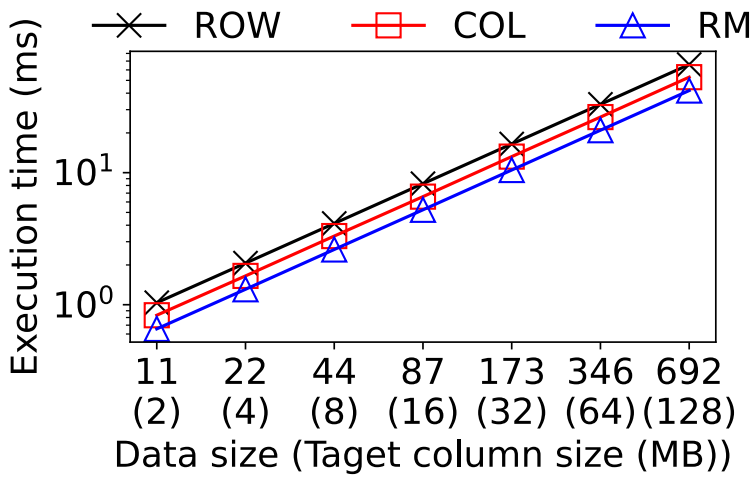
RME Scales with Data Size

TPC-H Q1 CPU-bound (sort, group by)



CPU overhead dominates data movement cost

TPC-H Q6 IO-bound

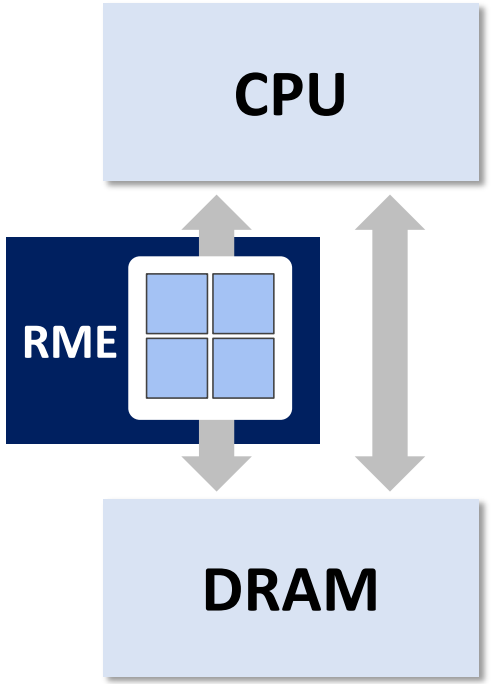


RME benefits regardless of data size

Brandeis

Summary

- **Relational Memory**
  - a novel SW/HW co-design paradigm
  - every query always has access to the optimal data layout
- ***ephemeral variables***
  - a simple and lightweight abstraction to use RM
- Relational Memory enables opportunities for innovation across the data system architecture.



Relational Fabric, ICDE '23

Brandeis

Future Work

Data Transformation for ML workloads

Matrix and tensor slicing

Integrating with Real DBMS

Exploring query optimization

DRAM Controller Augmentation

Utilizing bank interleaving and parallelism

Brandeis

Thank you

Ju Hyoung Mun (jmun@brandeis.edu)