Relational Memory:

Native In-Memory Accesses on Rows and Columns

Ju Hyoung Mun

BOSTON UNIVERSITY

Technical University of Munich

Red Hat

Relational Databases are everywhere

Column

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	d_2	e_2
a_3	b_3	c_3	d_3	e_3

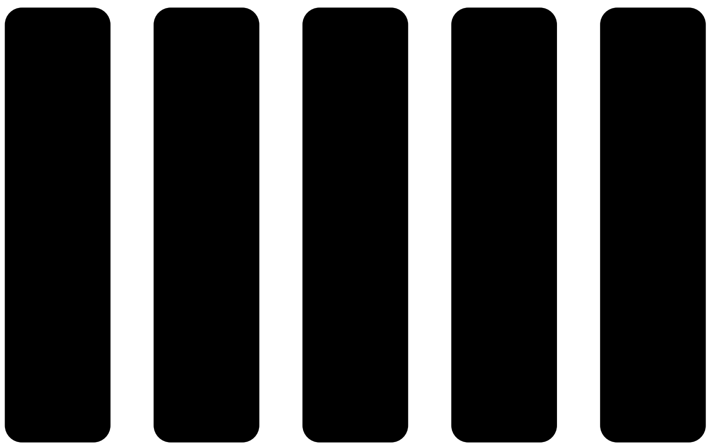
Row

Relation

Data Layouts

row-stores

column-stores

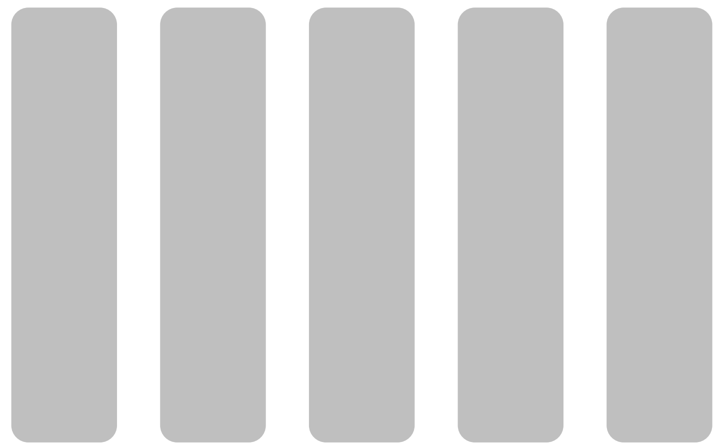


Data Layouts

row-stores

Transactional

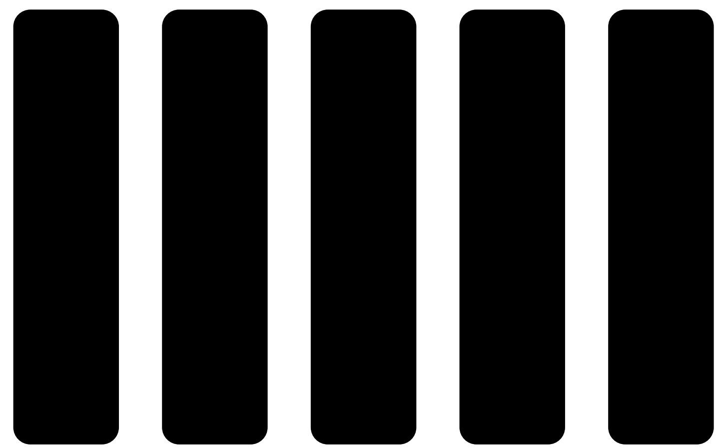
column-stores



Data Layouts

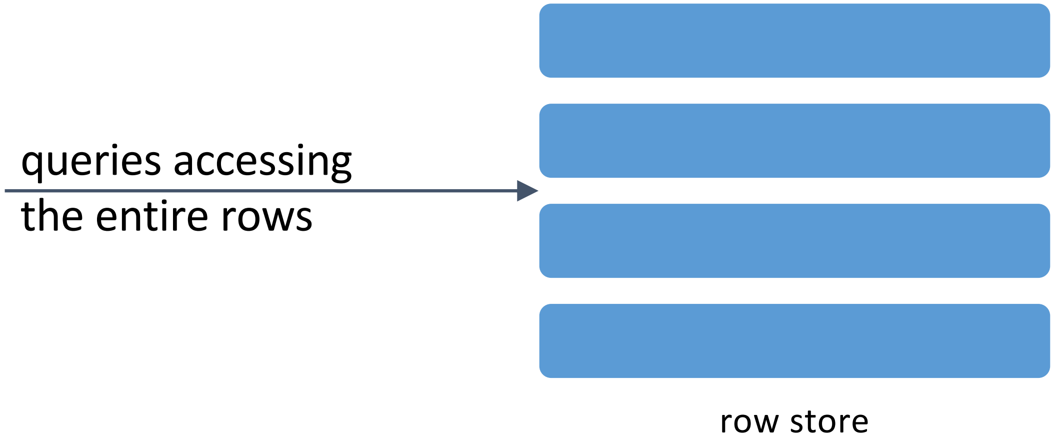
row-stores

column-stores



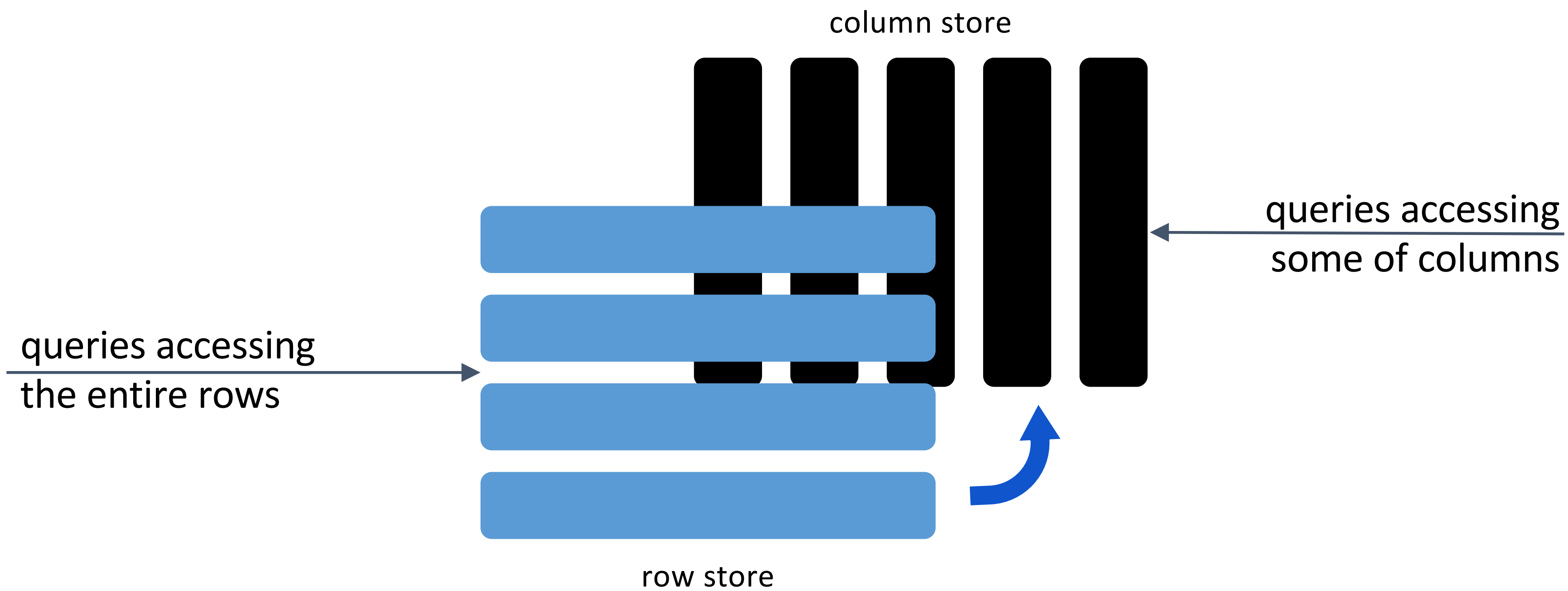
Analytical

Adaptive layout



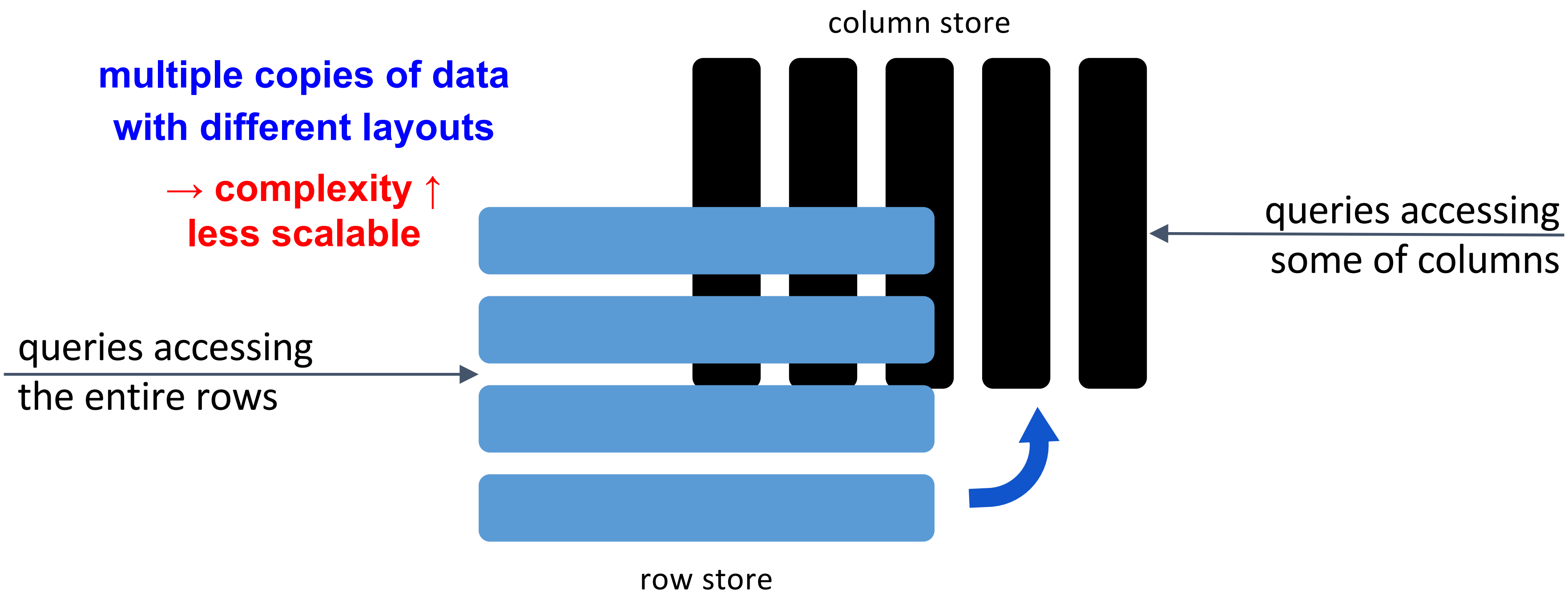
E.g., H2O (ACM SIGMOD, 2014), HyPer (IEEE ICDE , 2011), Peloton (ACM SIGMOD, 2016), OctopusDB (CIDR, 2011)

What are the disadvantages of the adaptive layout?



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

Adaptive layout

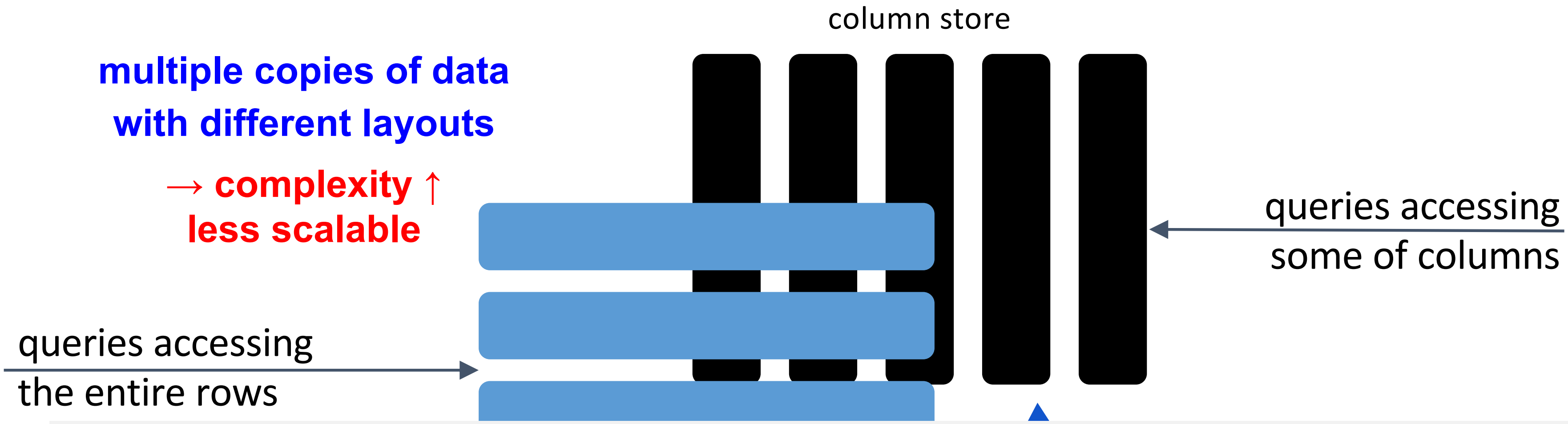


E.g., H2O (ACM SIGMOD, 2014), HyPer (IEEE ICDE , 2011), Peloton (ACM SIGMOD, 2016), OctopusDB (CIDR, 2011)

Adaptive layout

multiple copies of data with different layouts

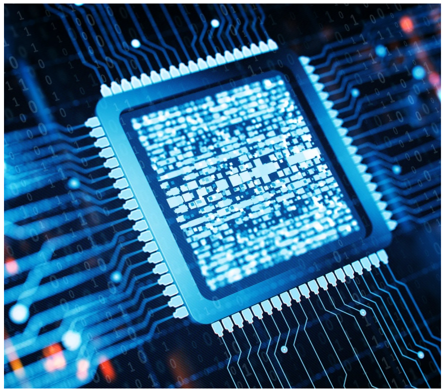
→ complexity ↑ less scalable



What if there is a shift over columns?

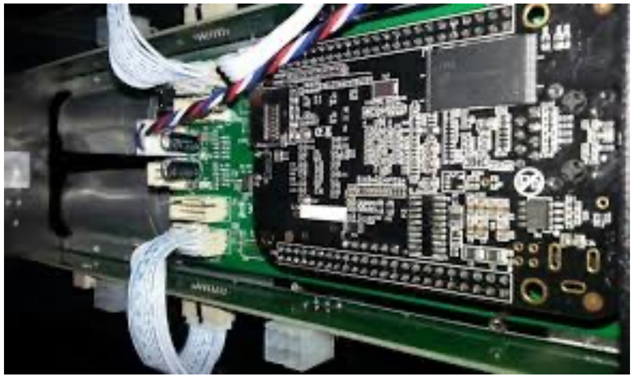
E.g., H2O (ACM SIGMOD, 2014), HyPer (IEEE ICDE , 2011), Peloton (ACM SIGMOD, 2016), OctopusDB (CIDR, 2011)

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

What is Hardware?



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

Advantages of 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

Advantages of Hardware Accelerators

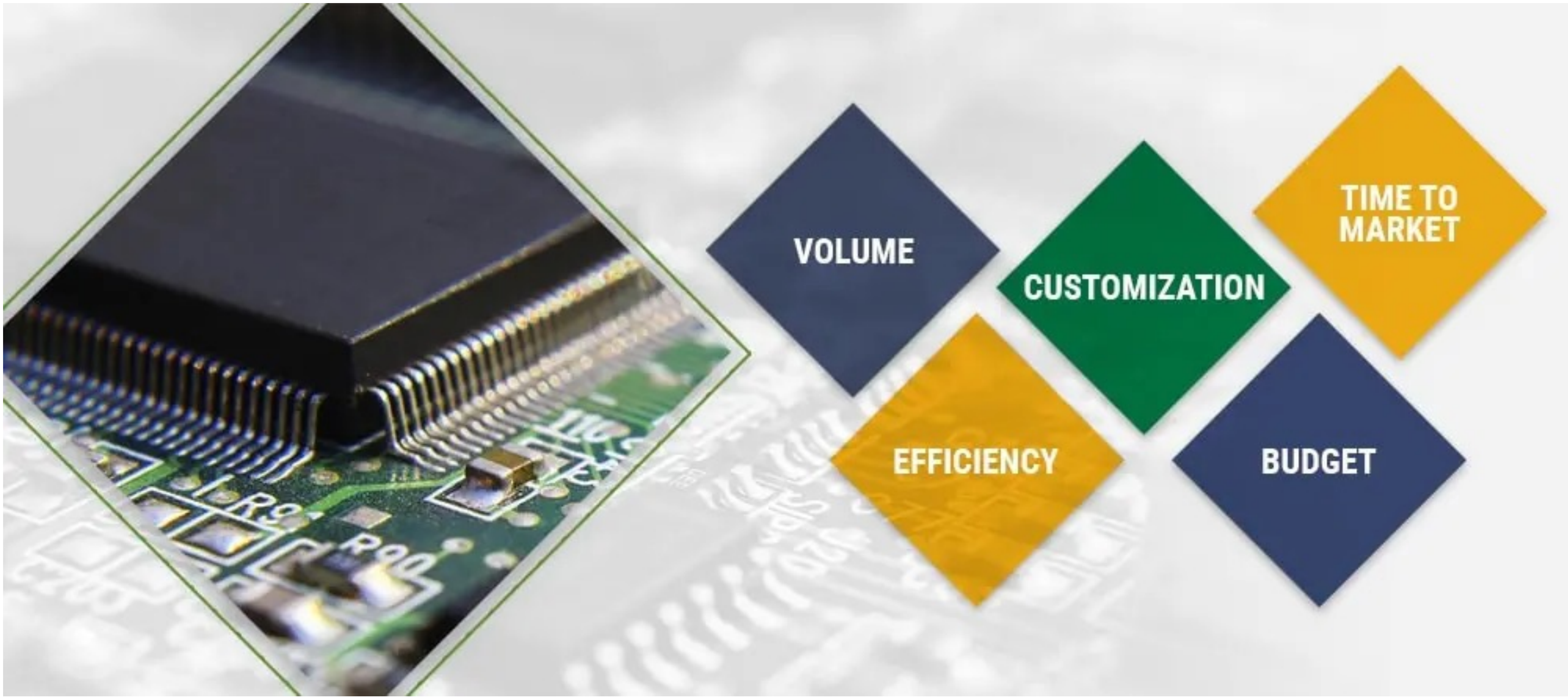
Advantages

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

Disadvantages

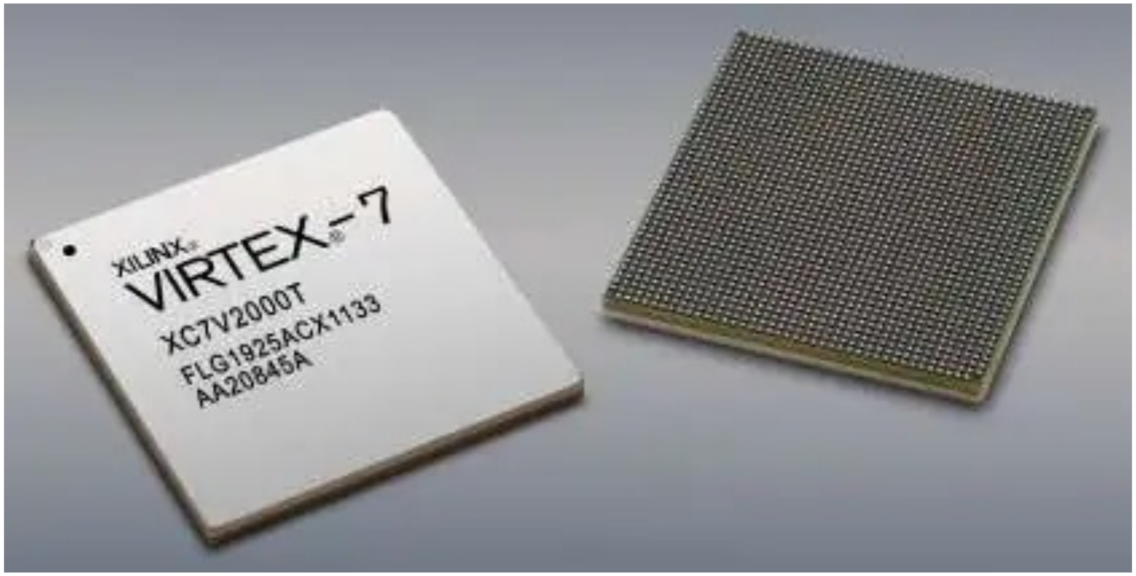
- Lower flexibility
- Higher costs of functional verification and times to market

ASICs

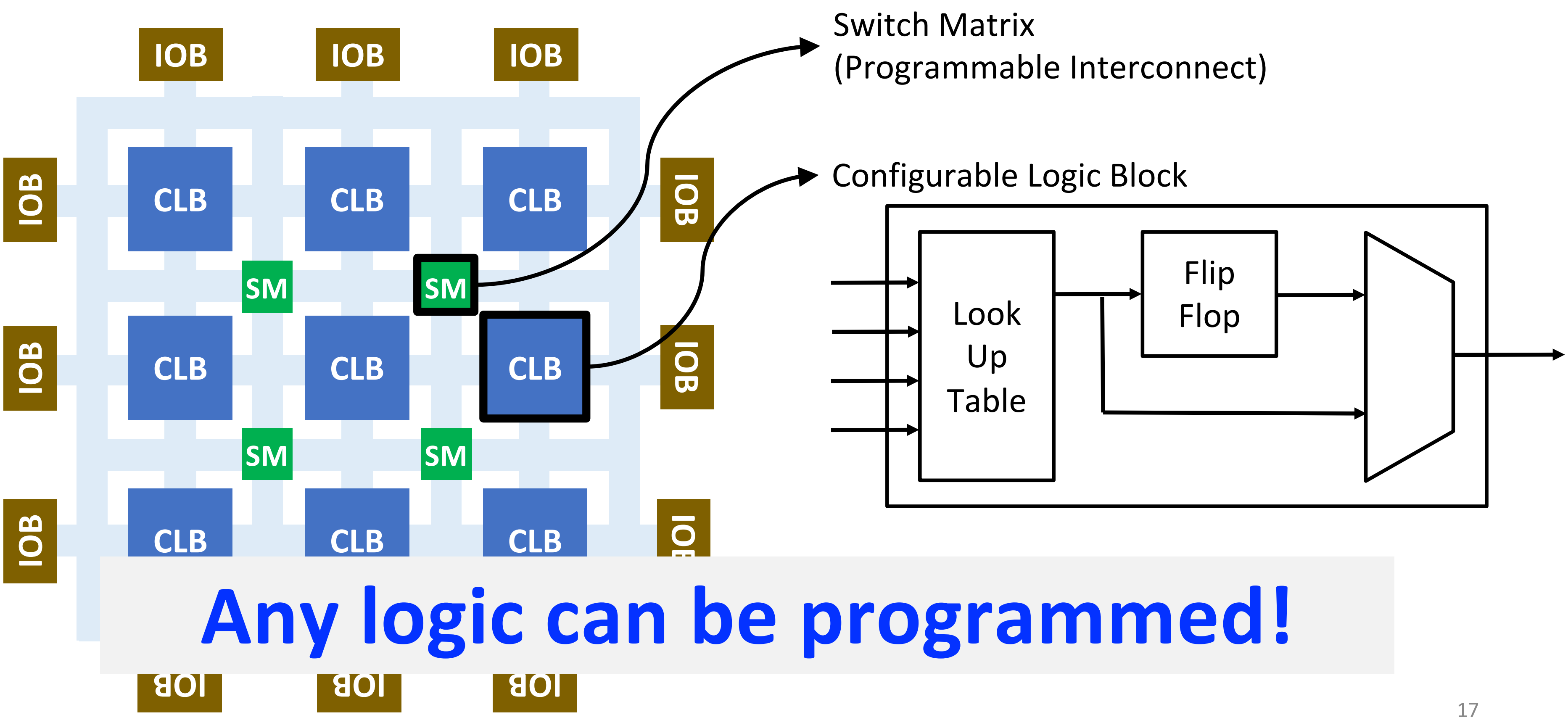


Programmable Logic

Field Programmable Gate Arrays (FPGAs)

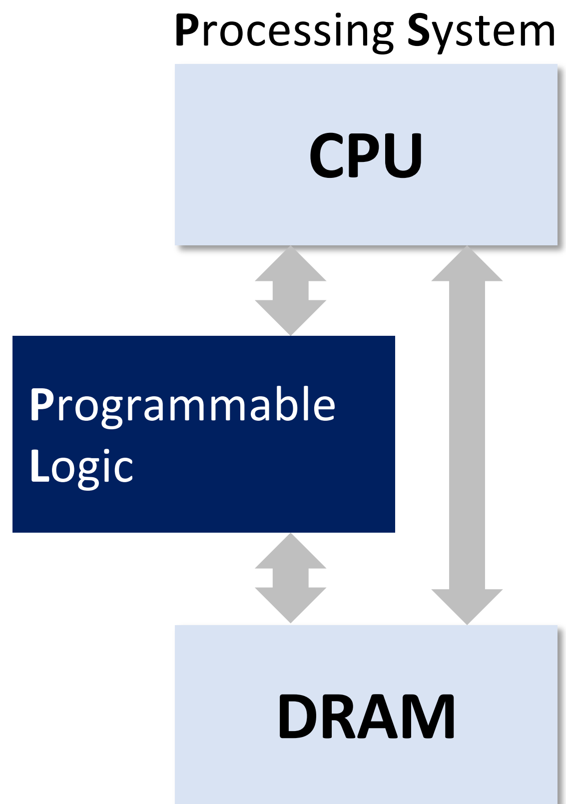


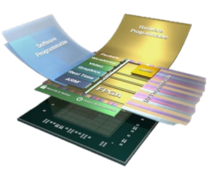
Architecture of Programmable Logic



Any logic can be programmed!

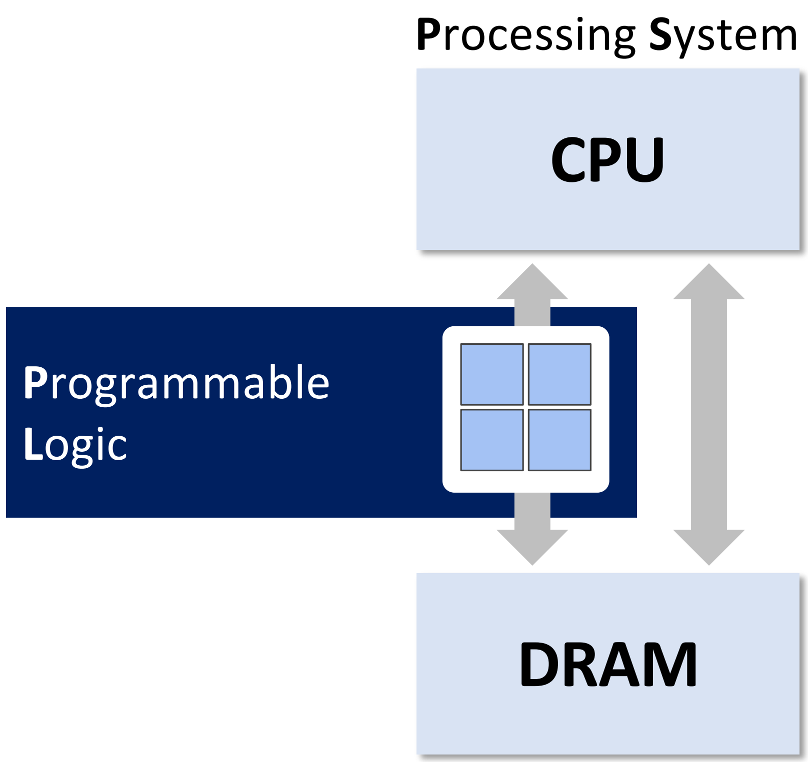
PS-PL Platforms



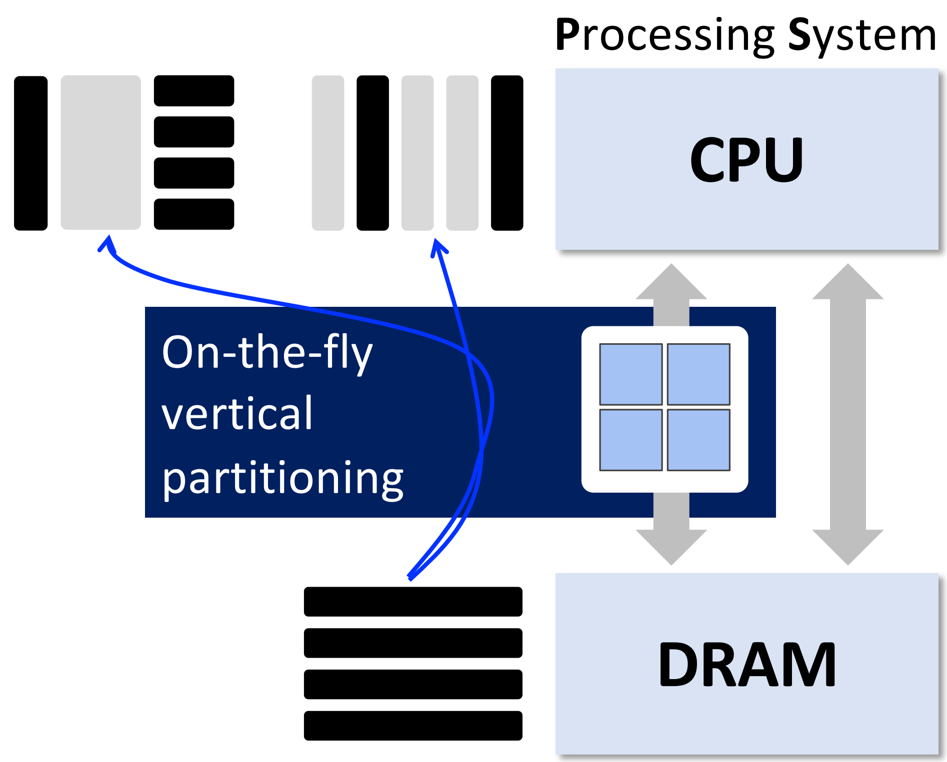


AMD XILINX UltraScale+ ENZIAN intel AGILEX

Relational **M**emory **E**ngine



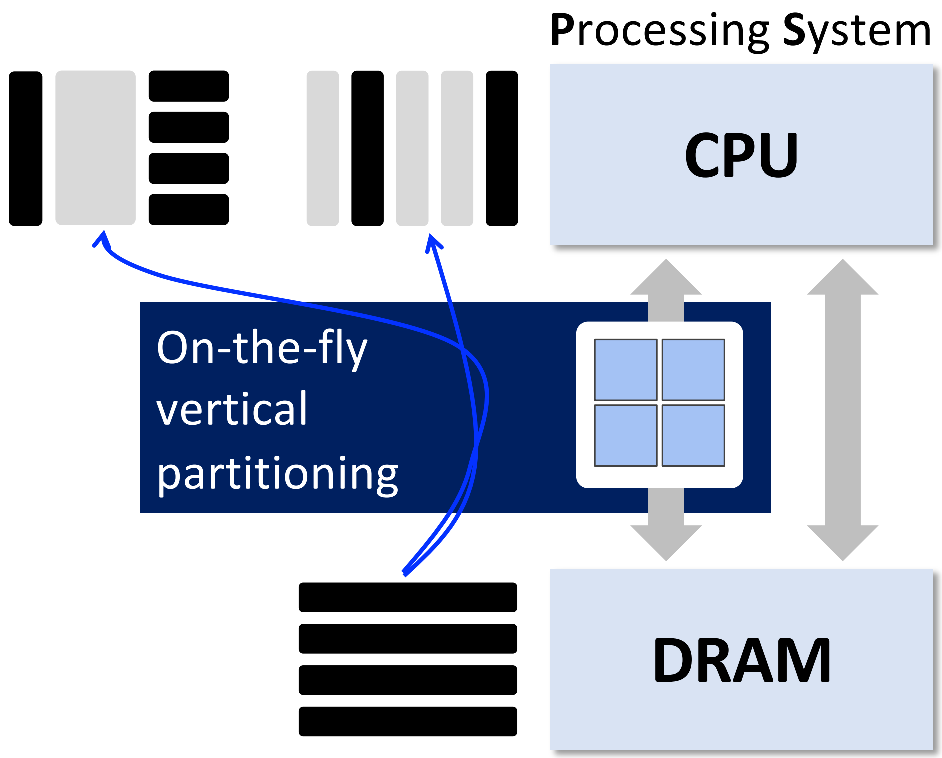
Relational **M**emory **E**ngine



Q1: how to build?

Q2: how to use?

Relational **M**emory **E**ngine



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

CPU

```
SELECT NAME
    FROM table
    WHERE weight/height>25;
```

base row store

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 ;
};
```

optimal layout

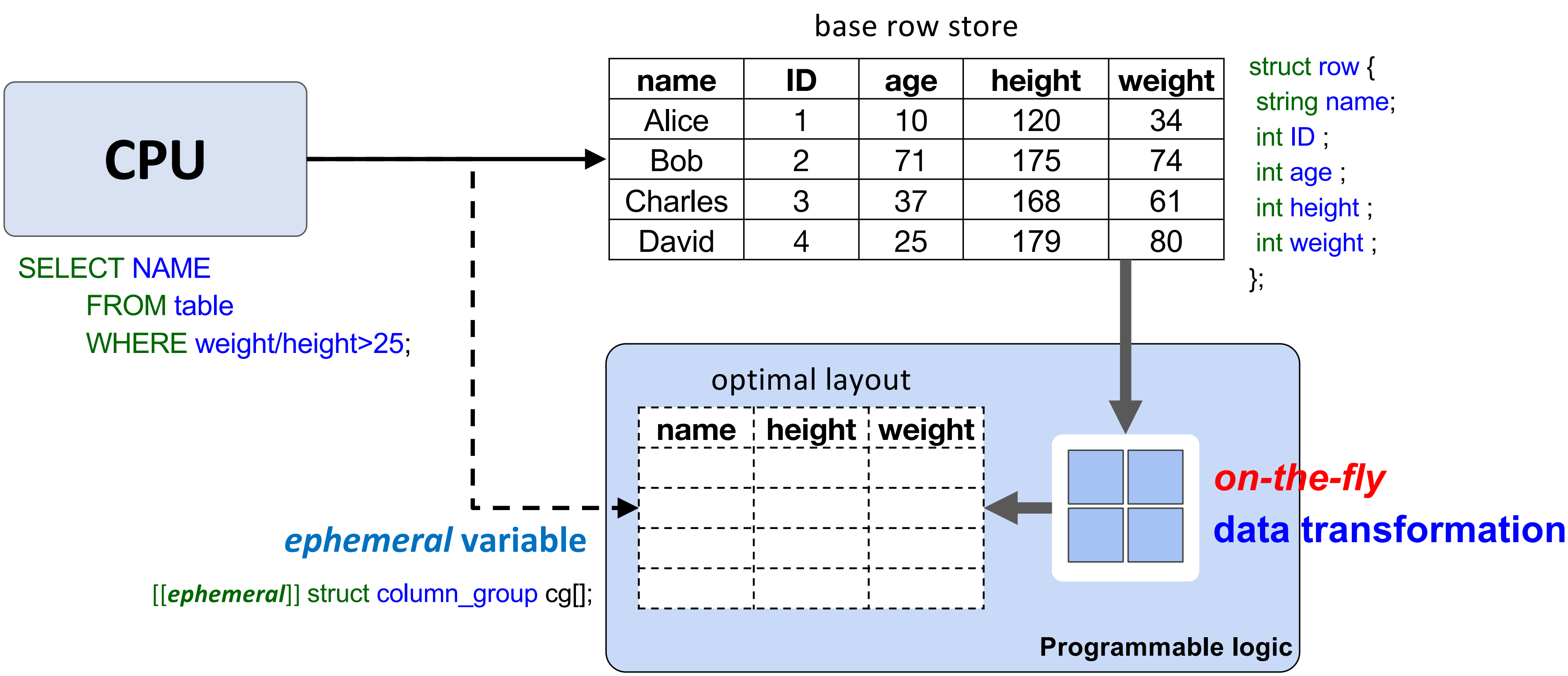
name	height	weight
Alice	120	34
Bob	175	74
Charles	168	61
David	179	80

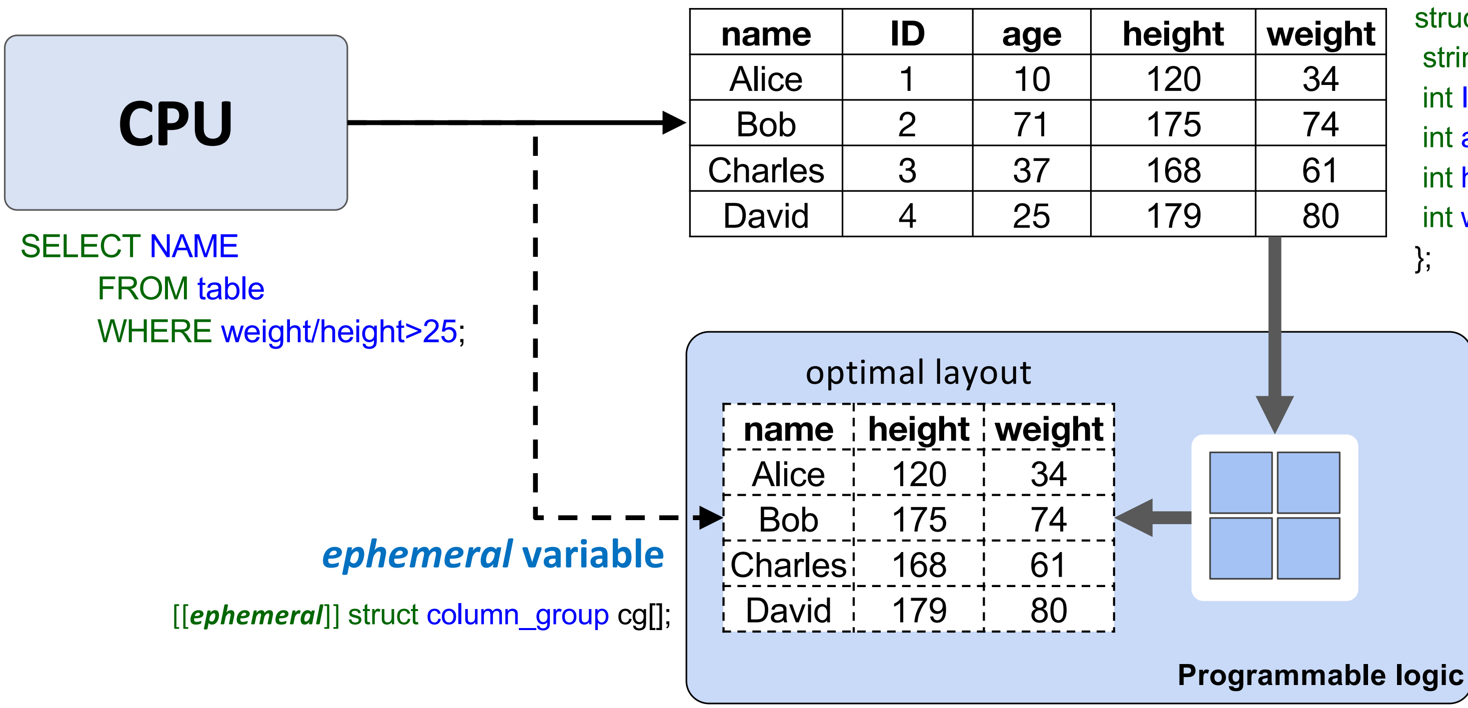
```
struct column_group {
 string NAME ;
 int height ;
 int weight ;
};
```

Not instantiated in main memory

ephemeral **variable**

```
[[ephemeral]] struct column_group cg[];
```


base row store
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 ;
};
CPU
SELECT NAME
FROM table
WHERE weight/height>25;
optimal layout
name | height | weight
on-the-fly
data transformation
ephemeral variable
[[ephemeral]] struct column_group cg[];
Programmable logic



base row store

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

***ephemeral* variable**

```
[[ephemeral]] struct column_group cg[];
```

CPU

SELECT NAME
FROM table
WHERE weight/height>25;

base row store

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

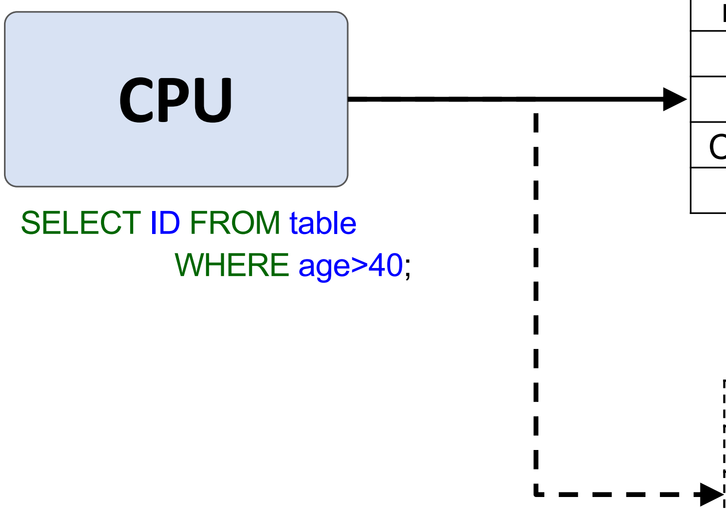
optimal layout

name	height	weight
Alice	120	34
Bob	175	74
Charles	168	61
David	179	80

```
struct column_group {
 string NAME ;
 int height ;
 int weight ;
};
```

***ephemeral* variable**

```
[[ephemeral]] struct column_group cg[];
```



SELECT ID FROM table
WHERE age>40;

base row store

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 ;
};
```

optimal layout

ID	age
1	10
2	71
3	37
4	25

```
struct column_group {
 string ID ;
 int age;
};
```

ephemeral **variable**

[[***ephemeral***]] struct column_group cg[];

✓ ***Transparent* data transformation**

✓ ***Optimal layout***

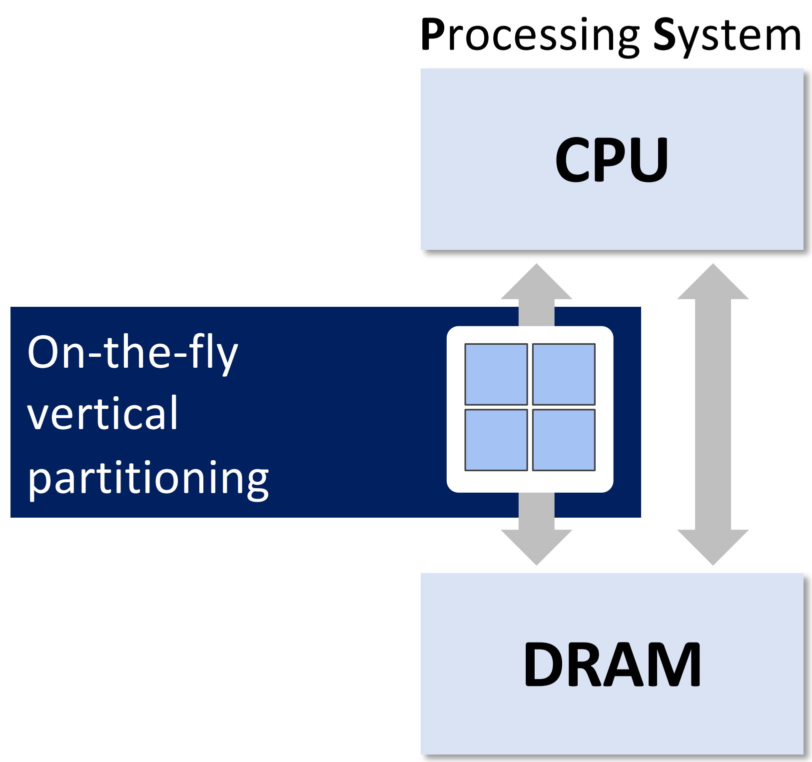
Q1: how to build?

Q2: how to use?

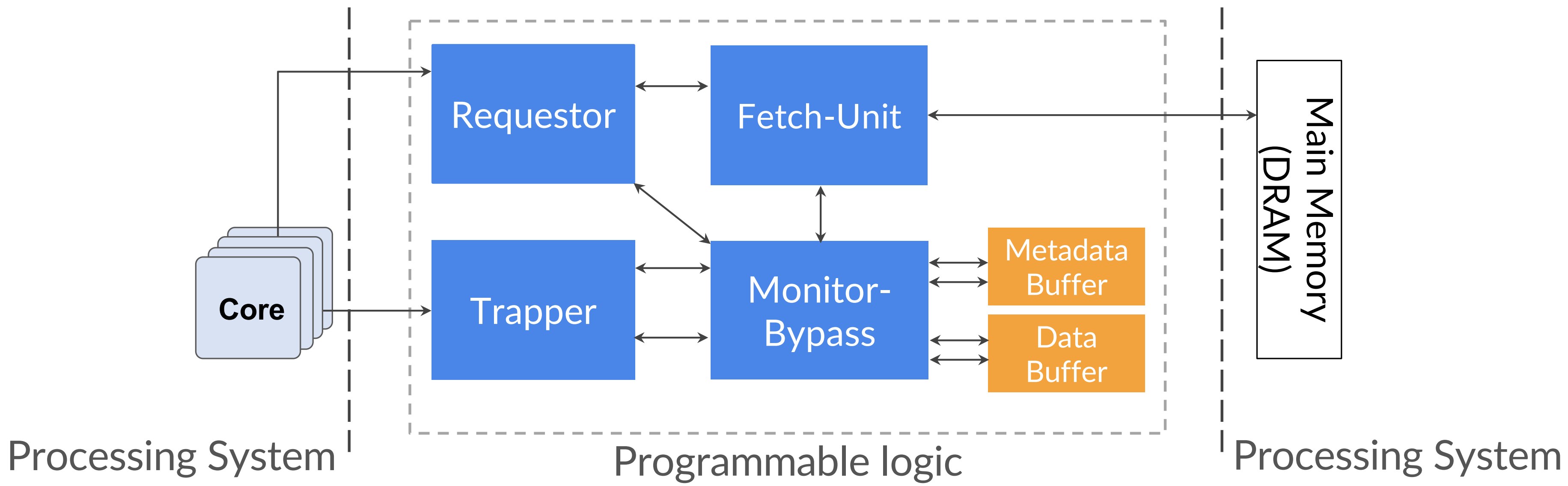
Relational

Memory

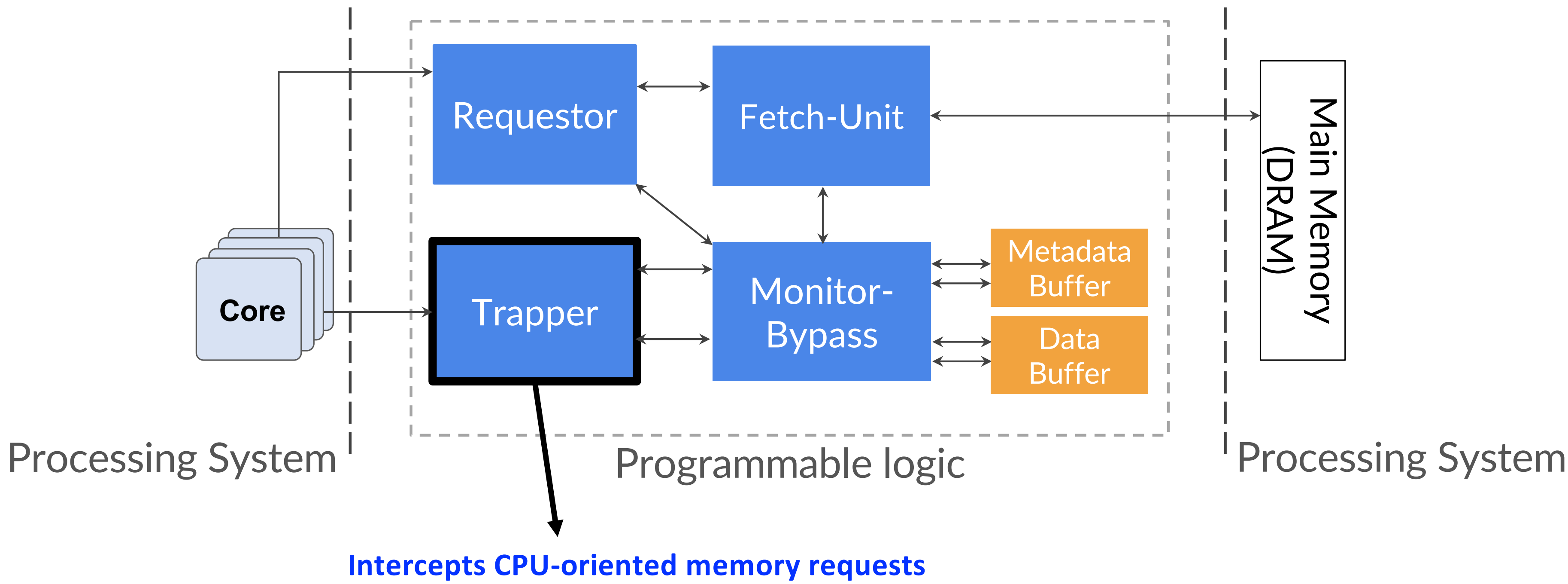
Engine



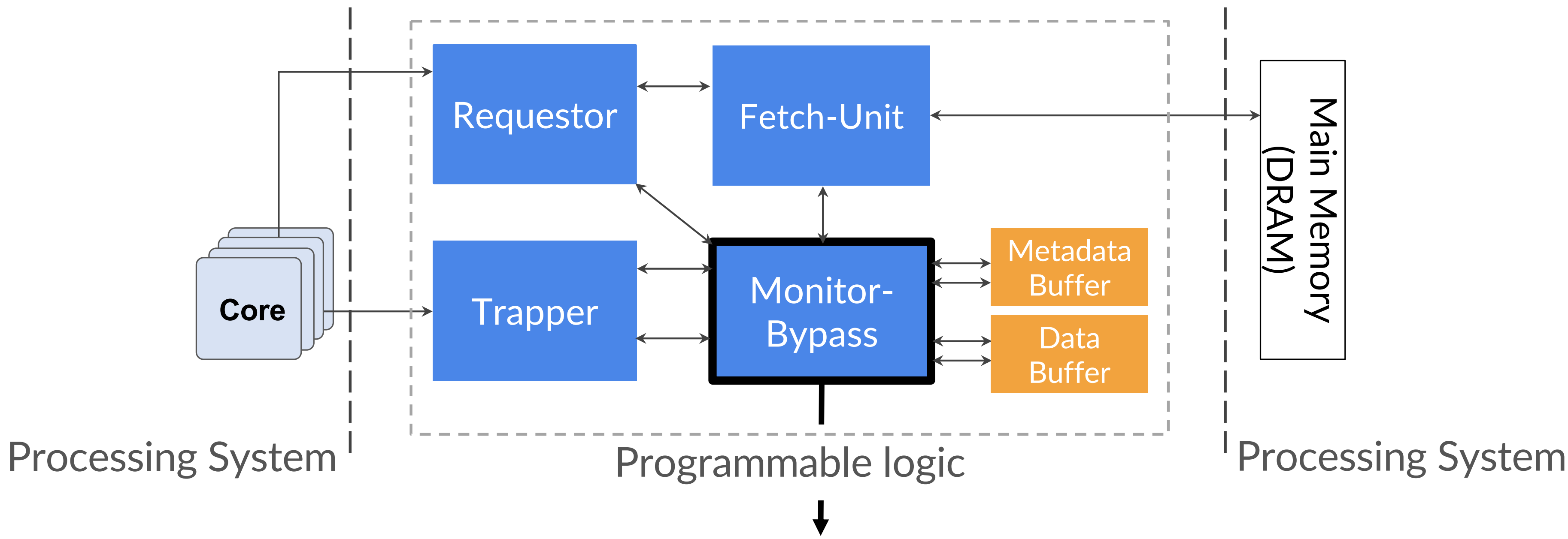
Relational Memory Engine



Relational Memory Engine

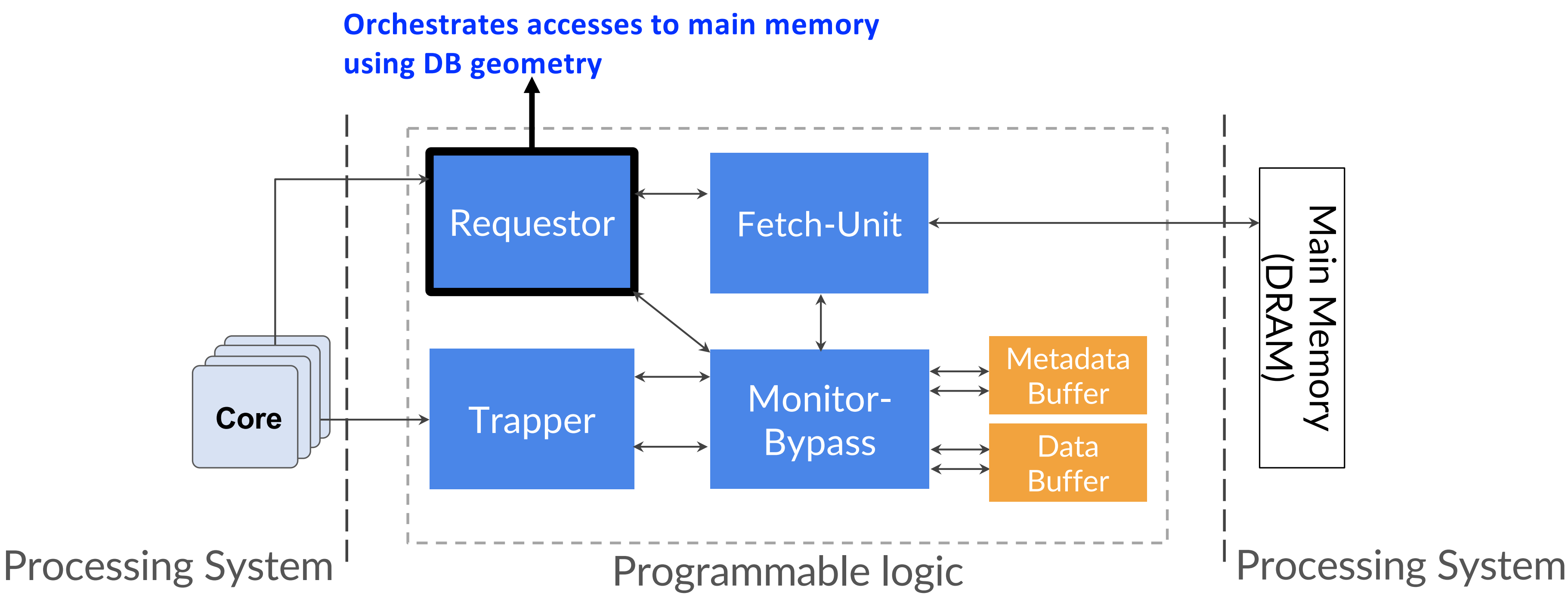


Relational Memory Engine

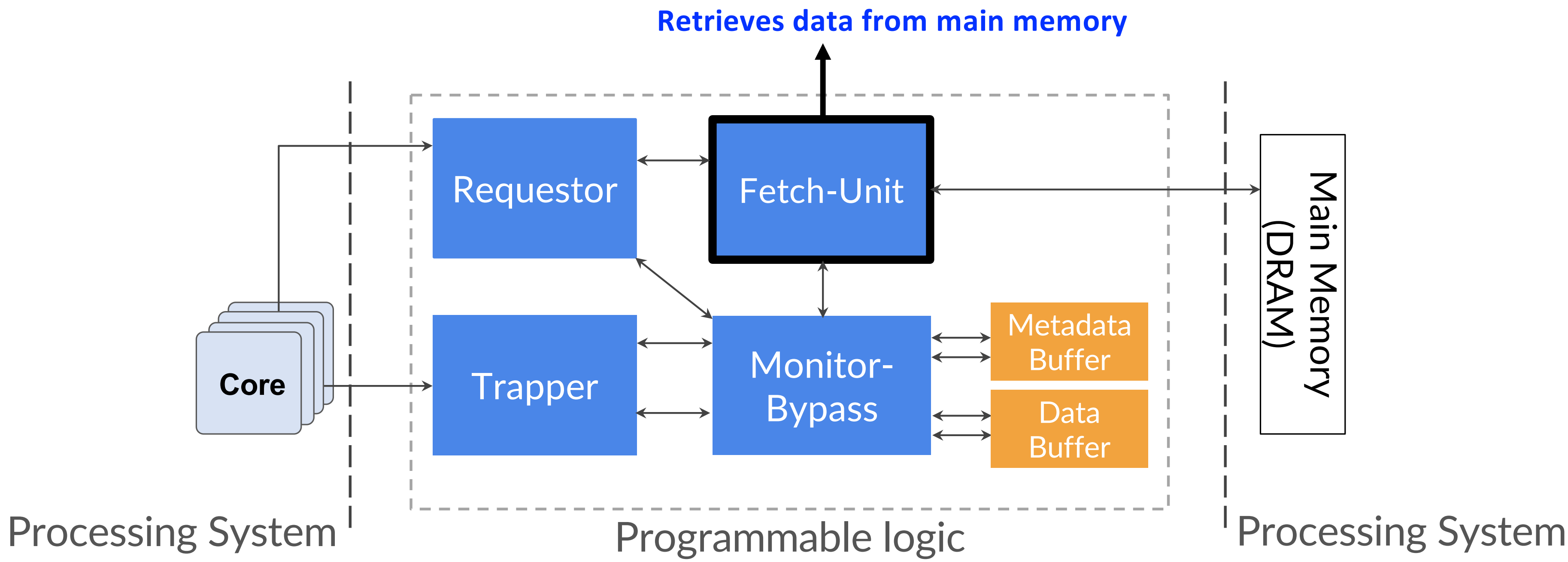


Monitors the completion of each reorganized cache line

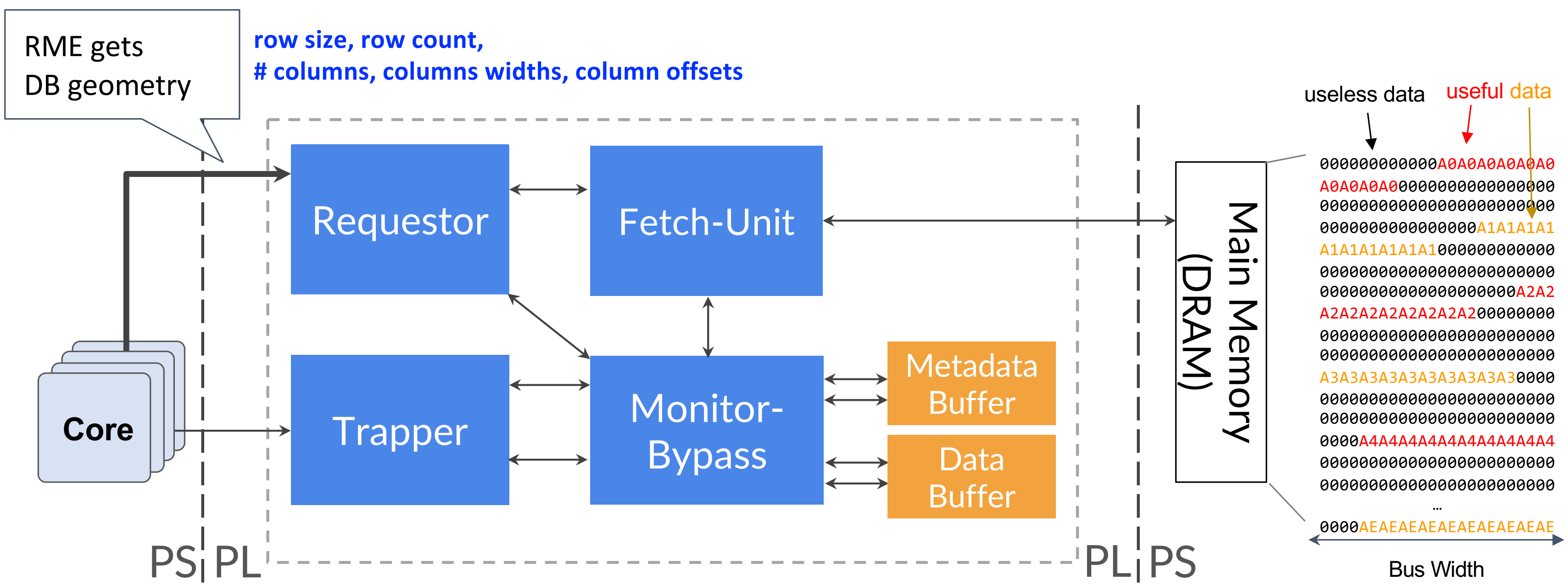
Relational Memory Engine



Relational Memory Engine



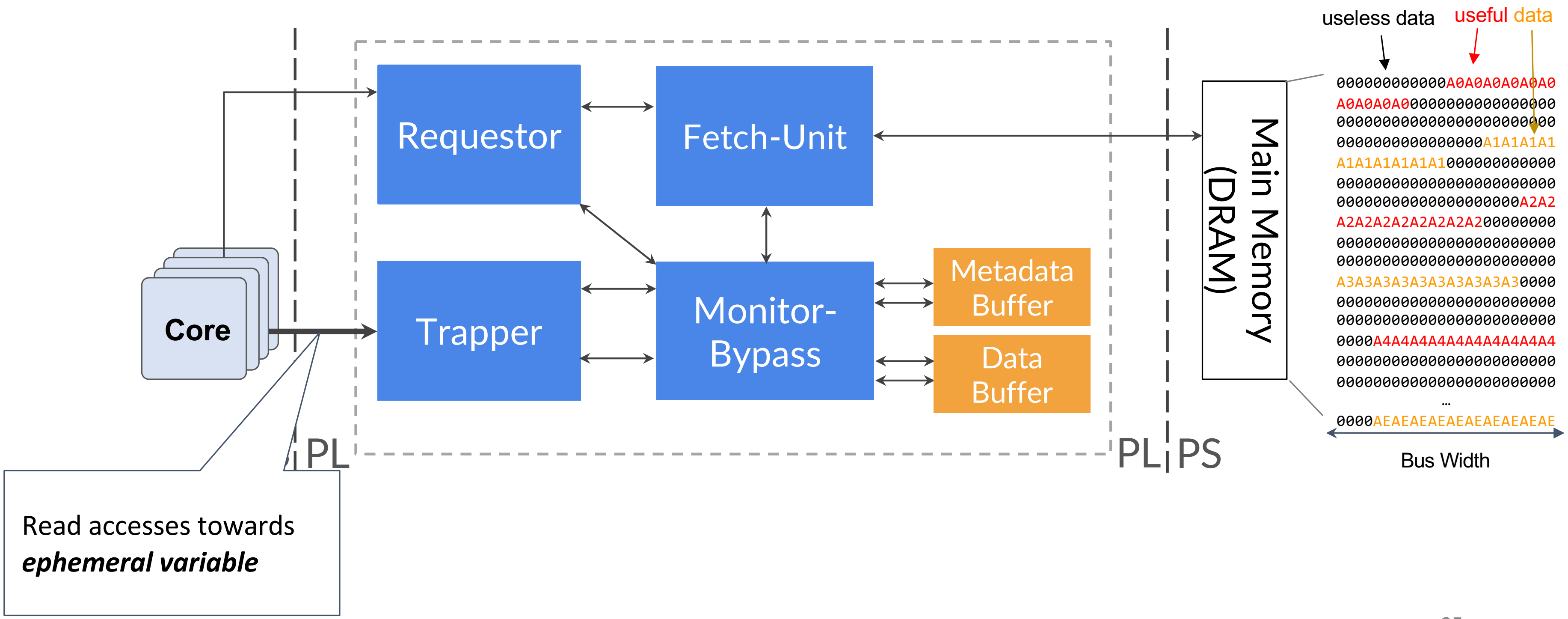
Relational Memory Engine



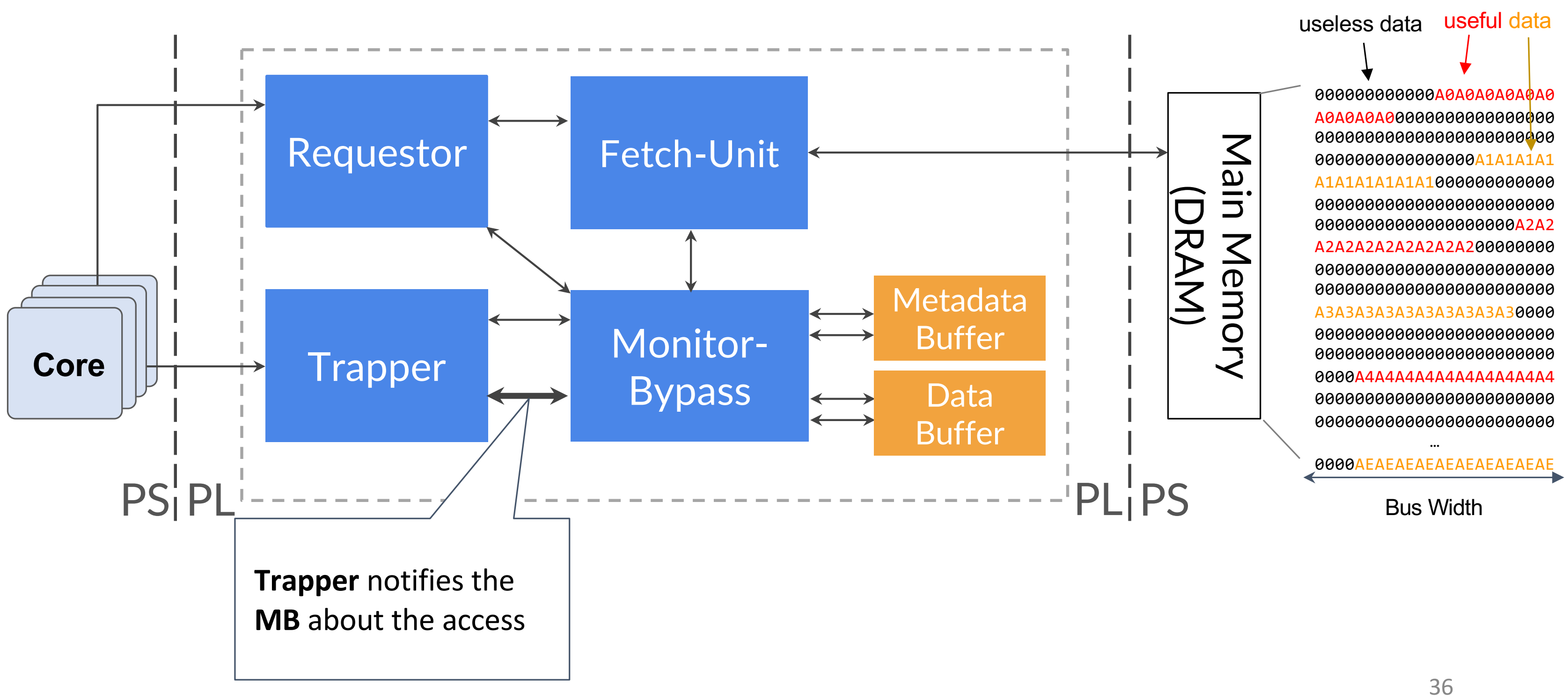
DiSC lab

BOSTON UNIVERSITY

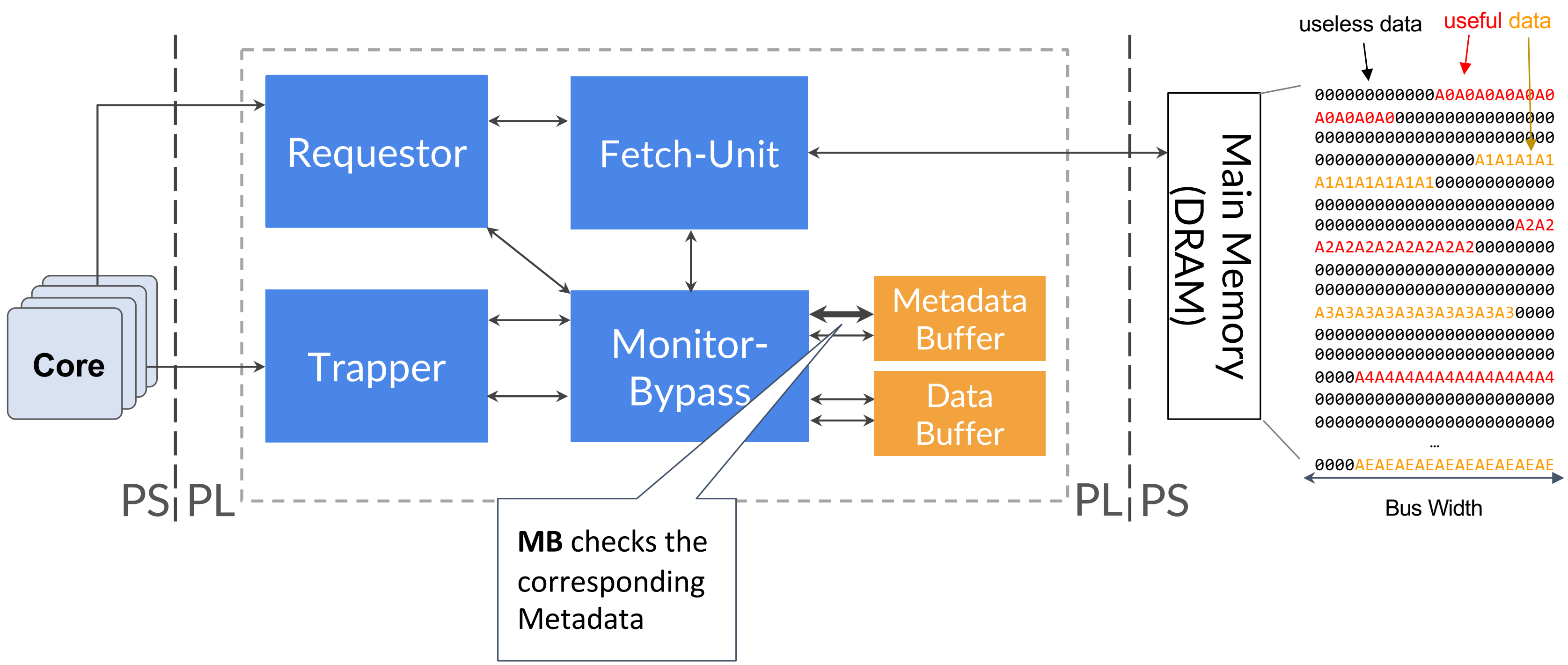
Relational Memory Engine



Relational Memory Engine

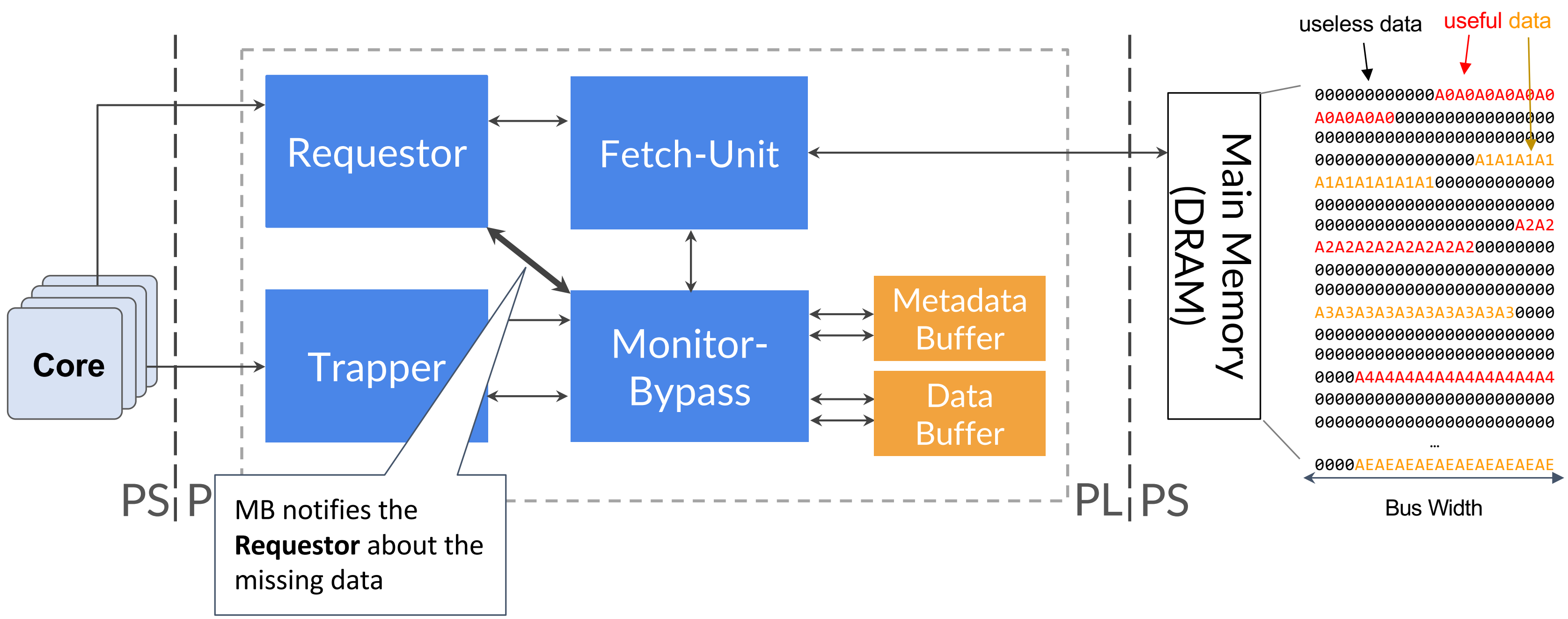


Relational Memory Engine



Relational Memory Engine

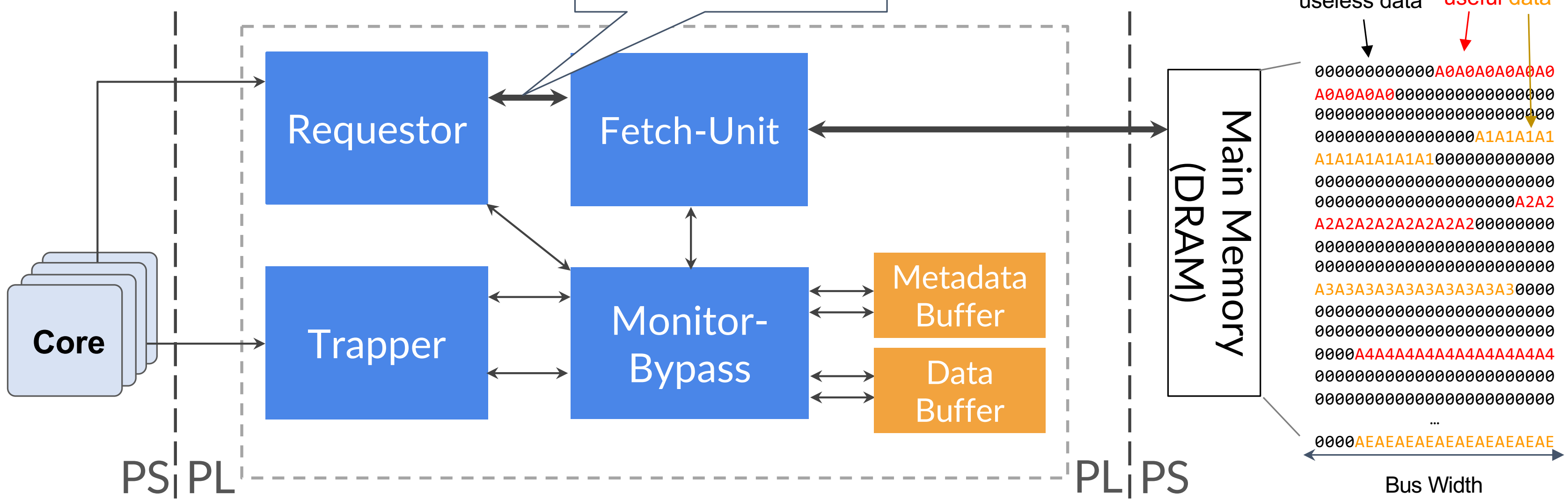
When the data is not in Data Buffer



Relational Memory Engine

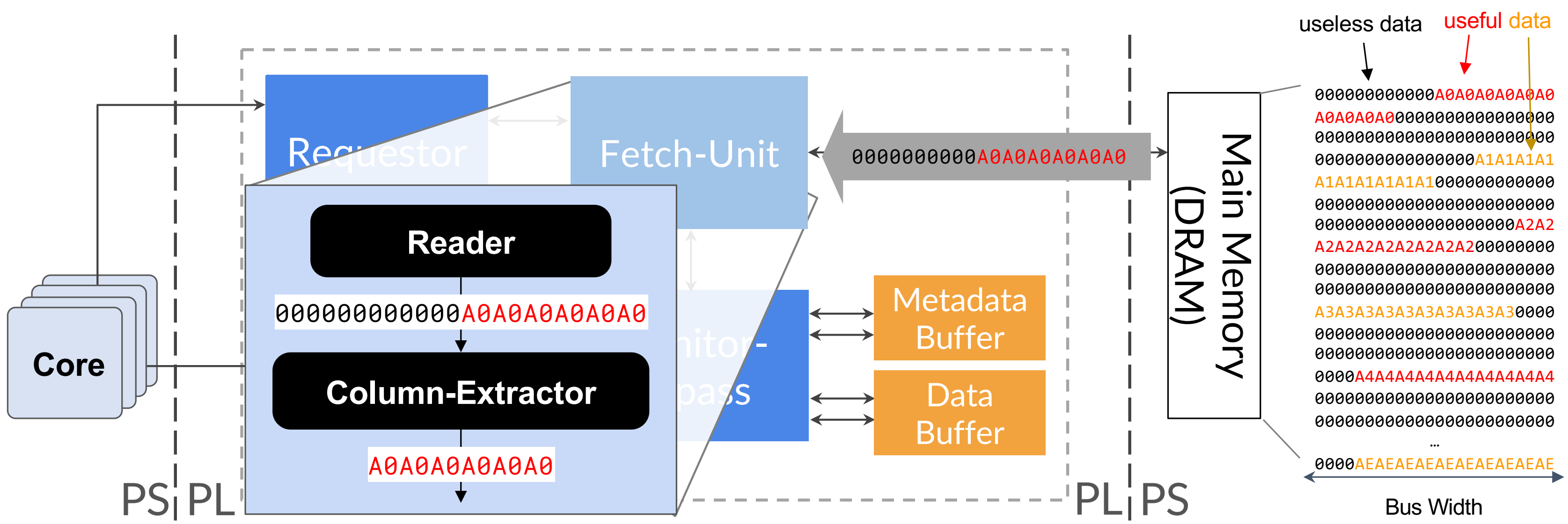
When the data is not in Data Buffer

Requestor programs the **Fetch-Unit** and it fires the read request toward the **DRAM**



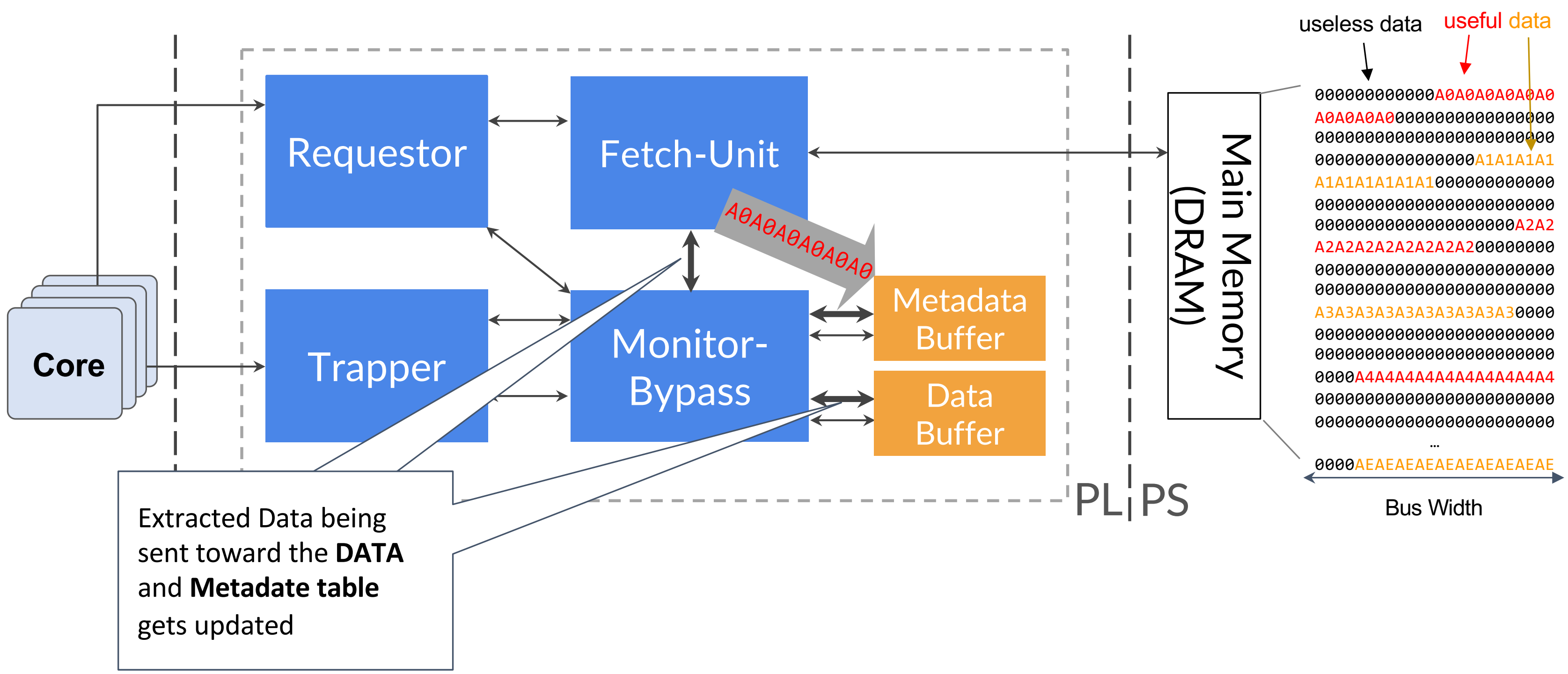
Relational Memory Engine

When the data is not in Data Buufer



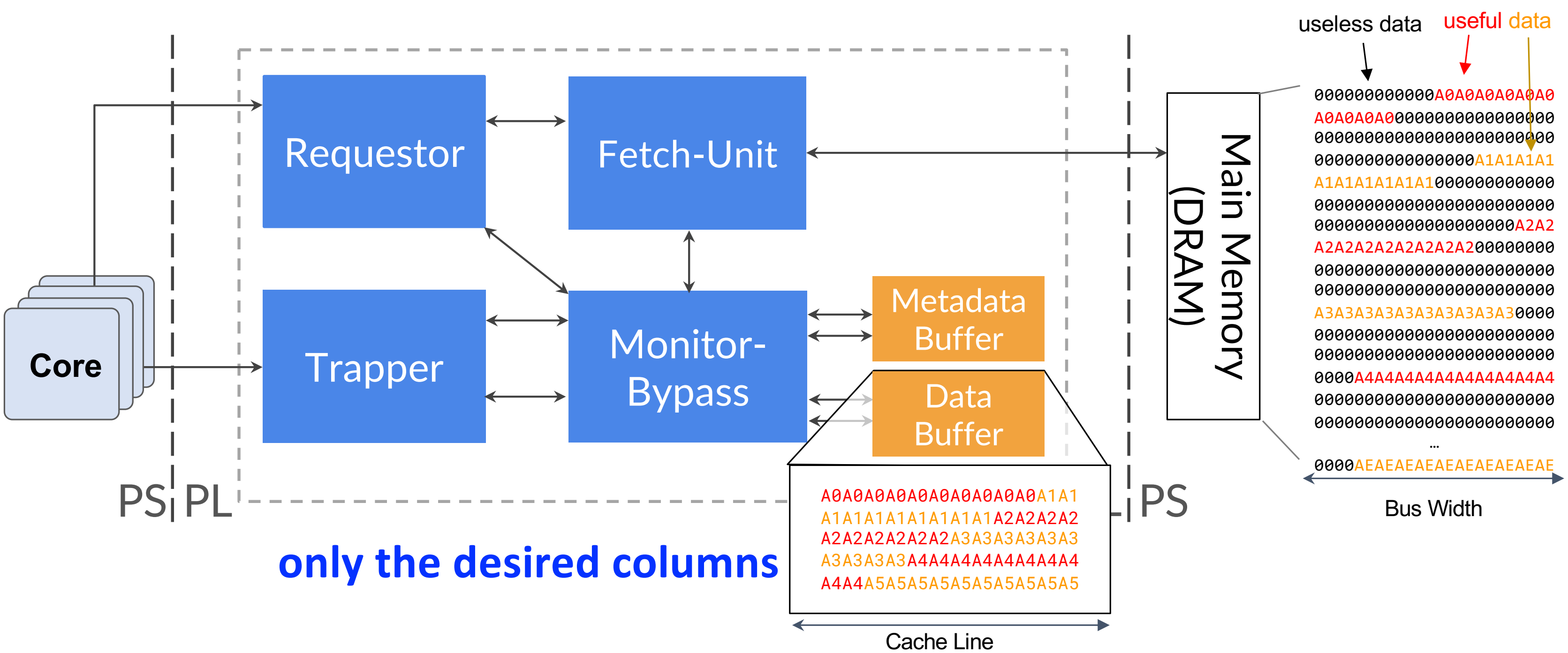
Relational Memory Engine

When the data is not in Data Buffer



Relational Memory Engine

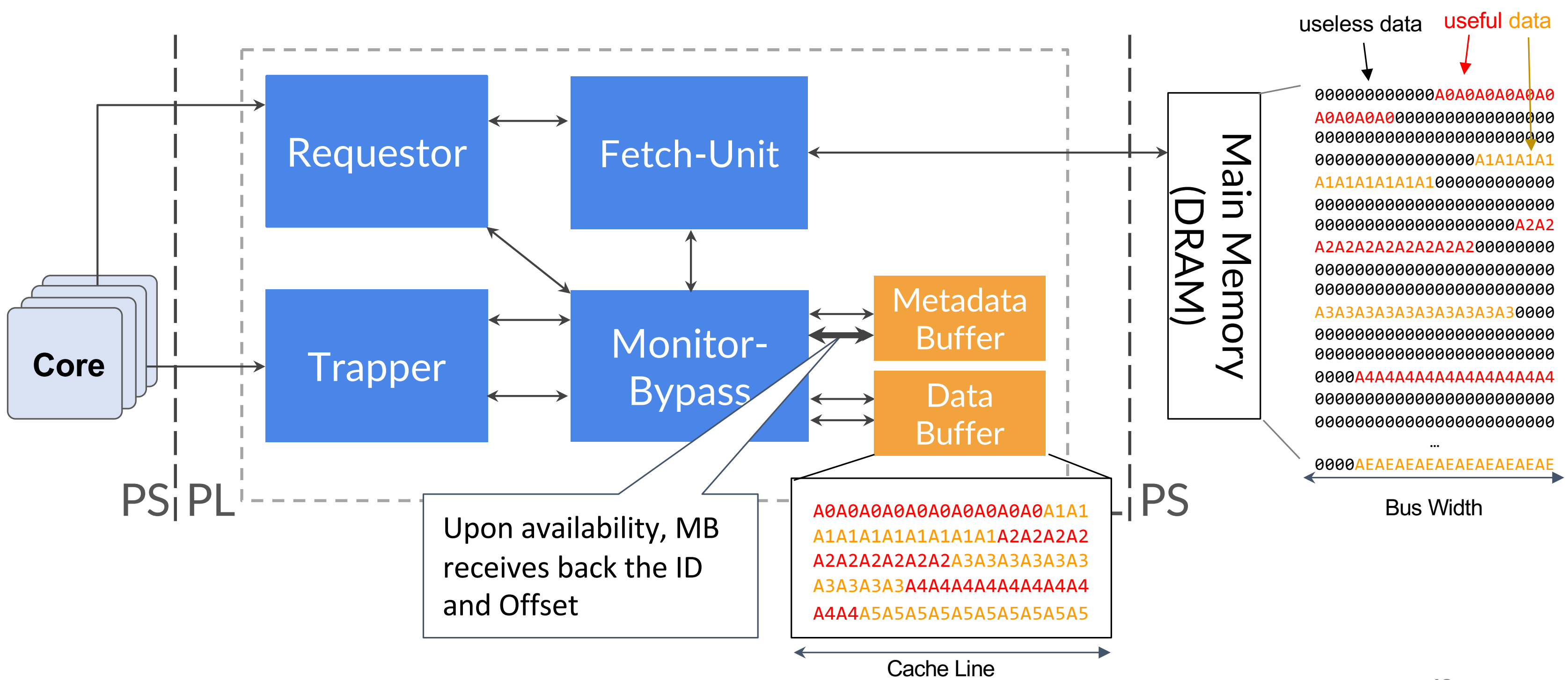
When the data is not in Data Buffer



only the desired columns

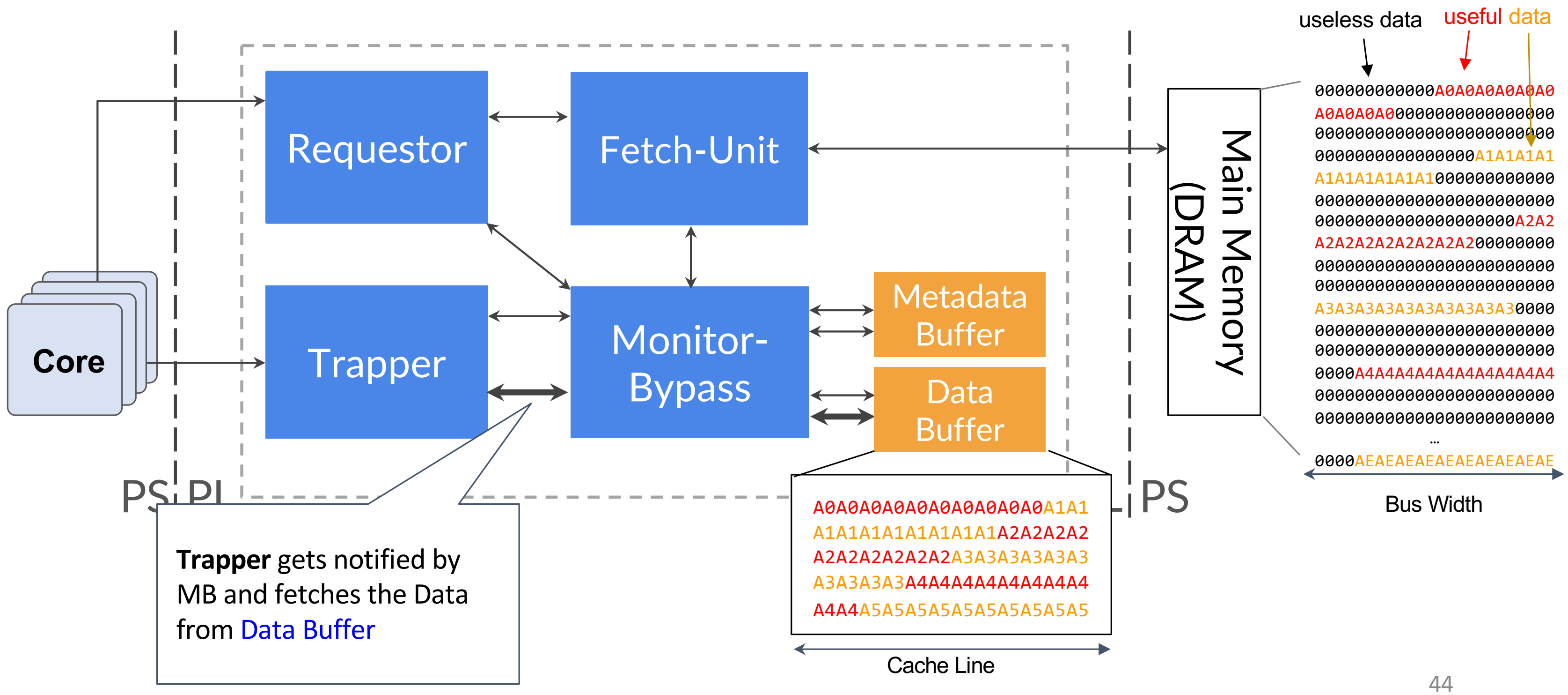
Relational Memory Engine

When the data is already in Data Buffer

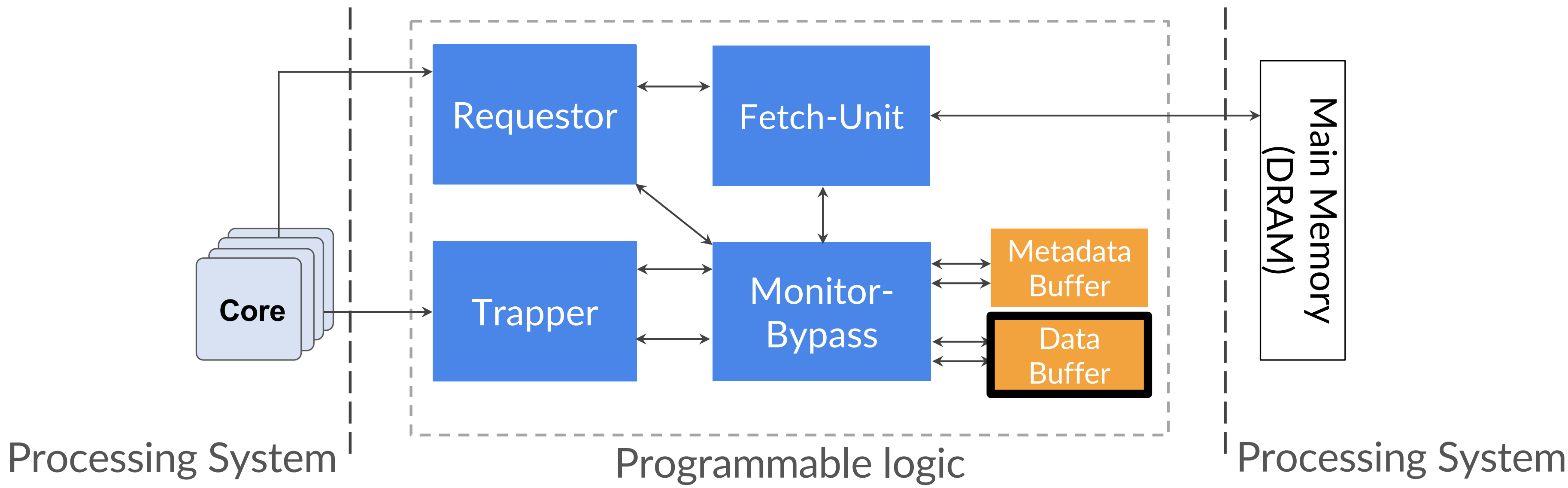


Relational Memory Engine

When the data is already in Data Buffer

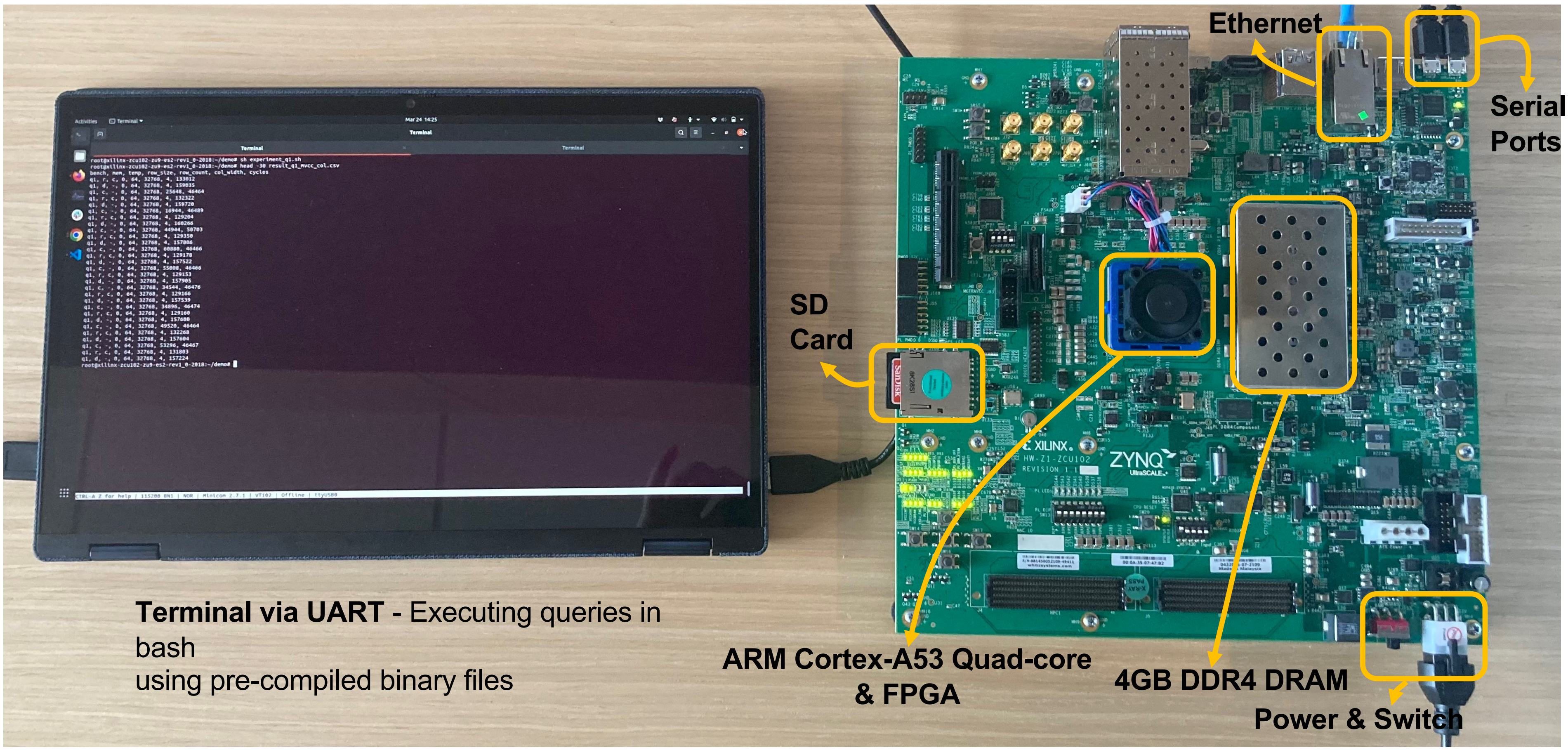


Relational Memory Engine

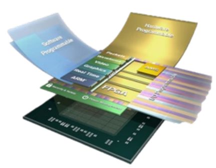


2MB << Data size

Target Platform

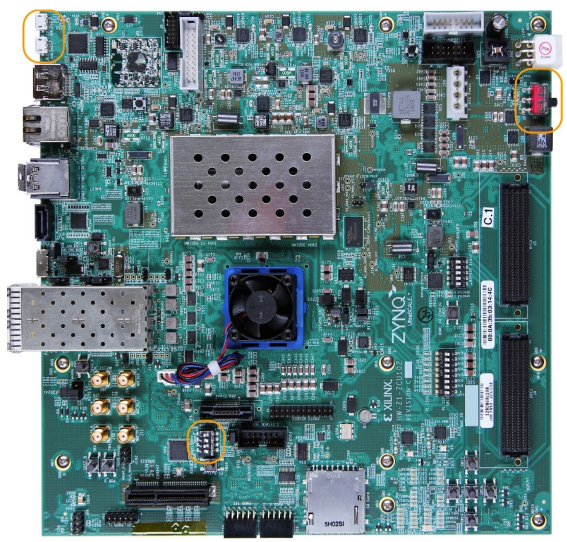


Target Platform



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

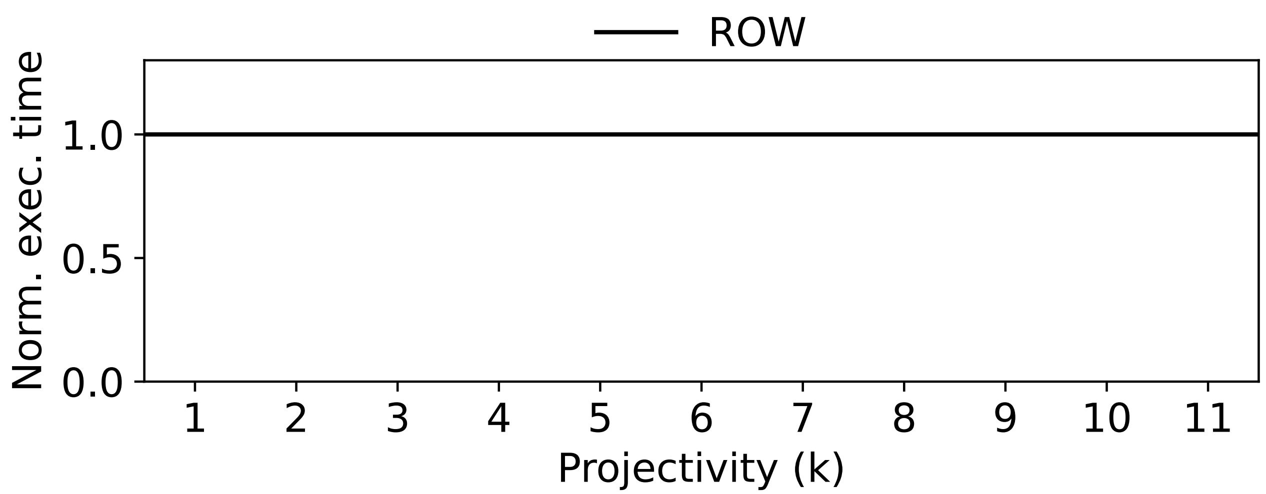
Approach tested

ROW : Direct row-wise access
COL : Direct columnar access
} **Processing System**

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

Queries Varying Projectivity

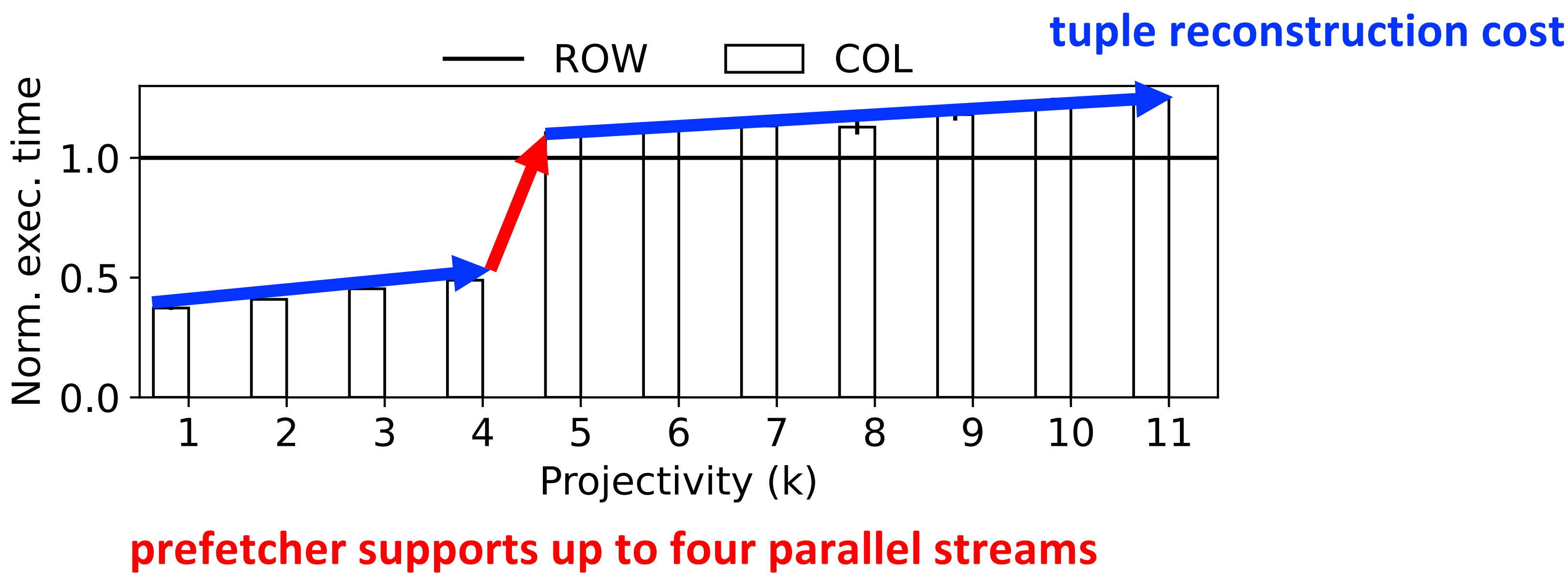
Q1: SELECT A1 , A2 , … , Ak FROM S;



Row size: 64 Bytes, Column size: 4 Bytes

Queries Varying Projectivity

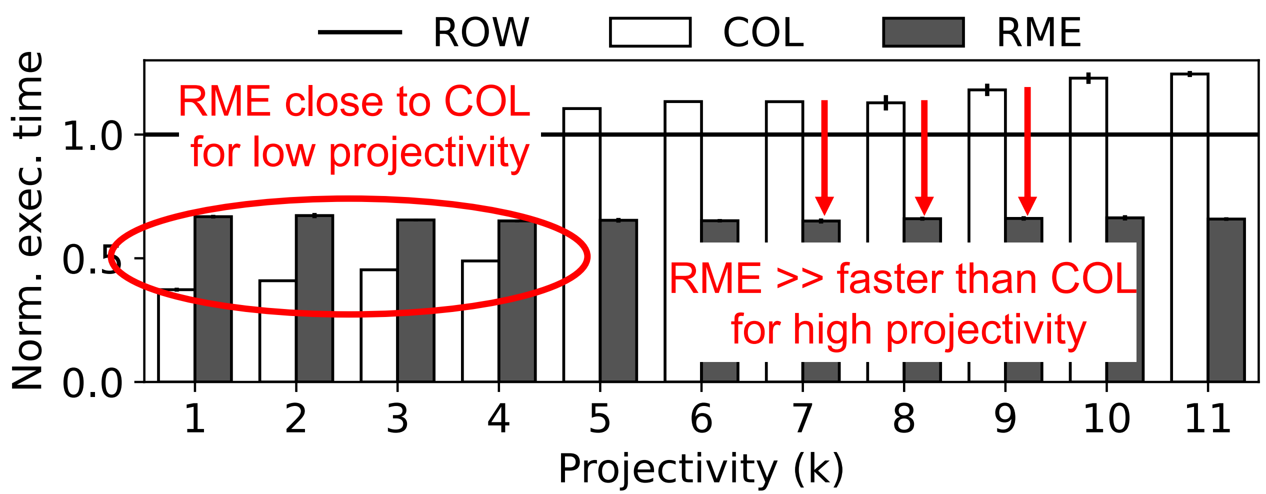
Q1: SELECT A1 , A2 , … , Ak FROM S;



Row size: 64 Bytes, Column size: 4 Bytes

Queries Varying Projectivity

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

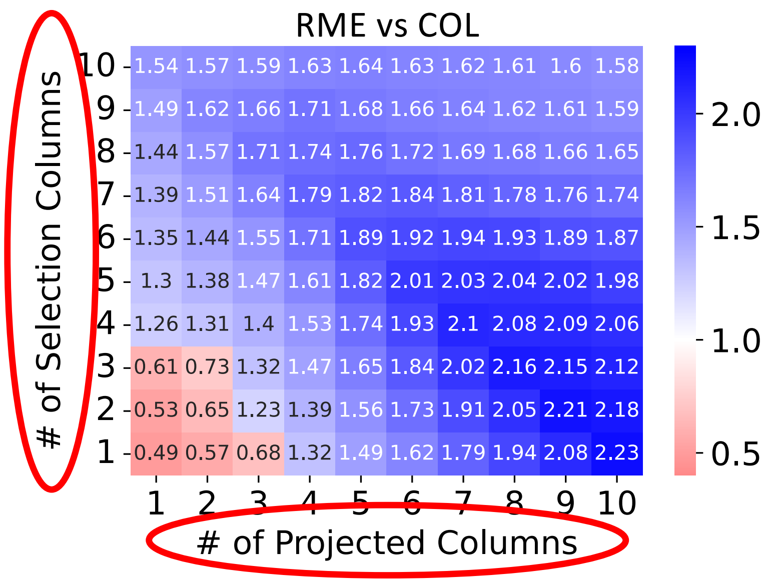


RME provides stable performance irrespectively of projectivity

Row size: 64 Bytes, Column size: 4 Bytes

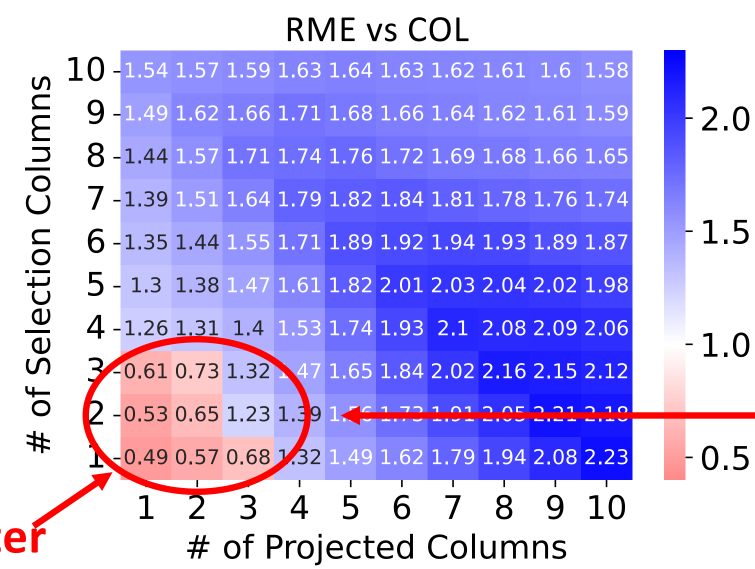
RME for Multiple Selection and Projection Attributes

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

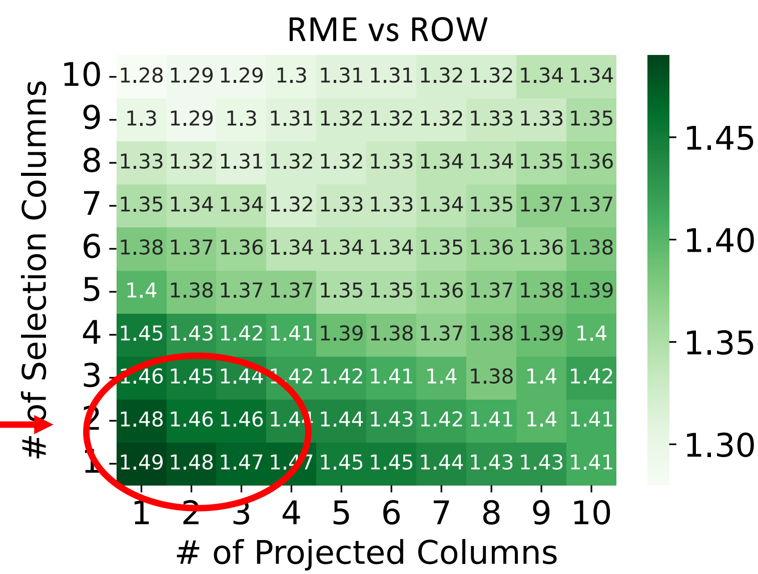


RME for Multiple Selection and Projection Attributes

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



COL faster

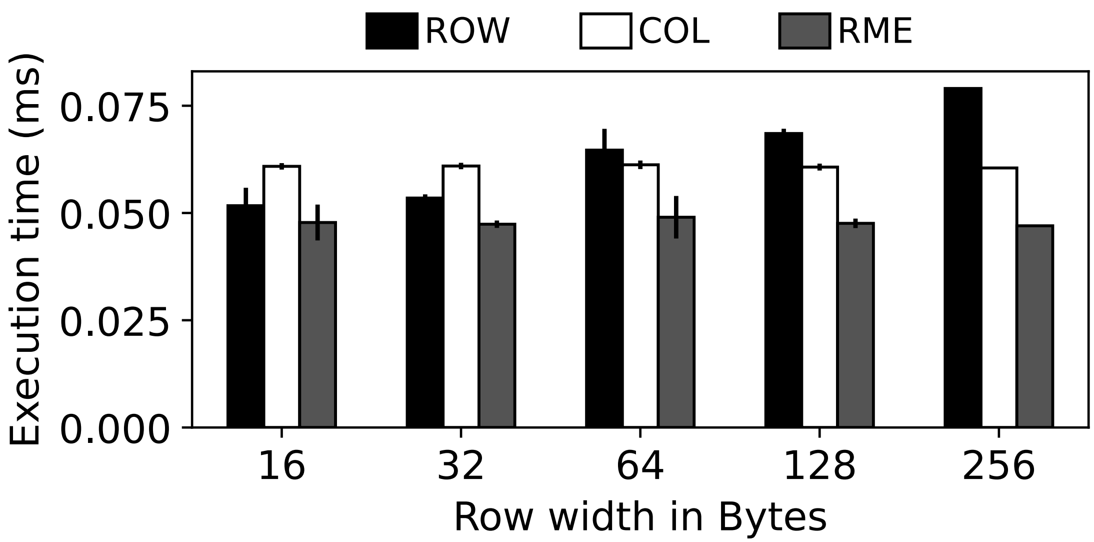


RME can be up to 2.23× faster than columnar access

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

Group by

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

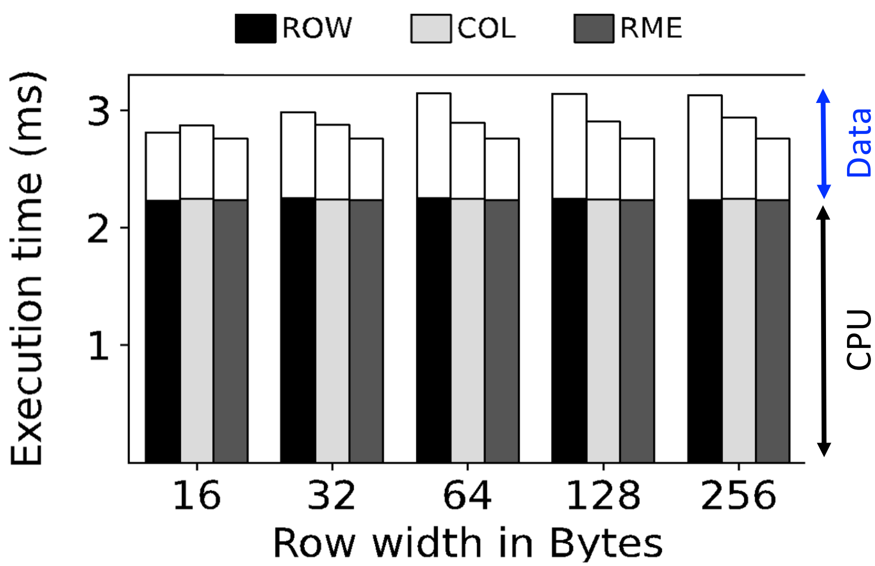


RME outperforms both ROW and COL

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: 4 Bytes

RME reduces data movement up to 41%

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

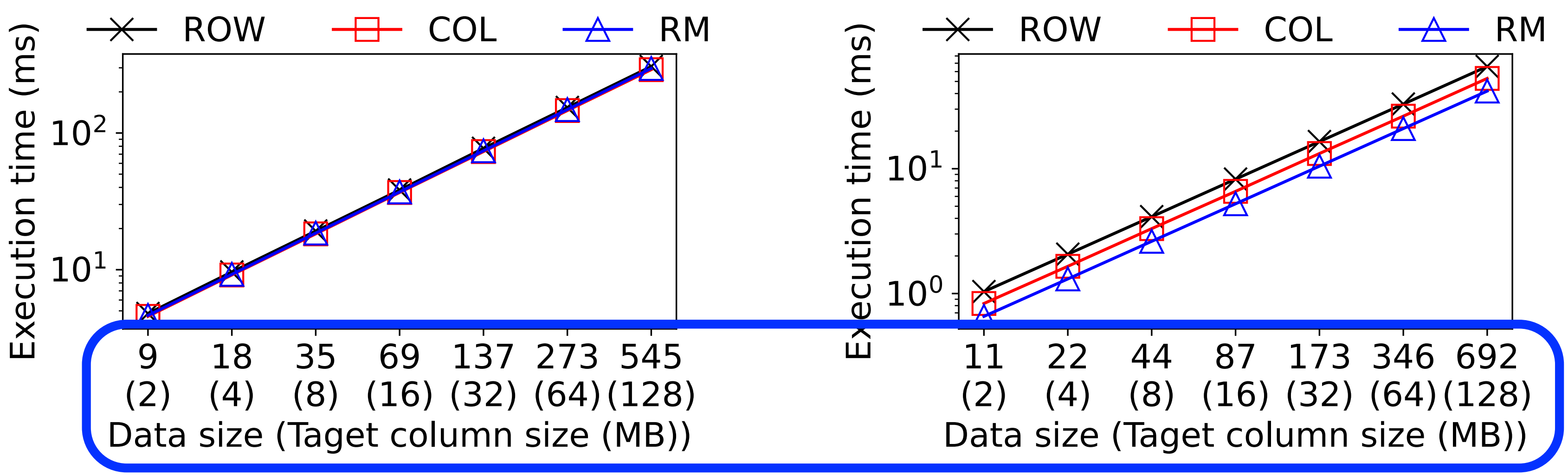
```
SELECT
   SUM(l_extendedprice*l_discount)
FROM lineitem
WHERE
   l_shipdate >= '[DATE]' and
   l_shipdate < '[DATE]' + 1 year and
   l_discount > [DISCOUNT] - 0.01 and
   l_discount < [DISCOUNT] + 0.01 and
   l_quantity < [QUANTITY];
```

Selectivity: 15%, projectivity: 18%

RME Scales with Data Size

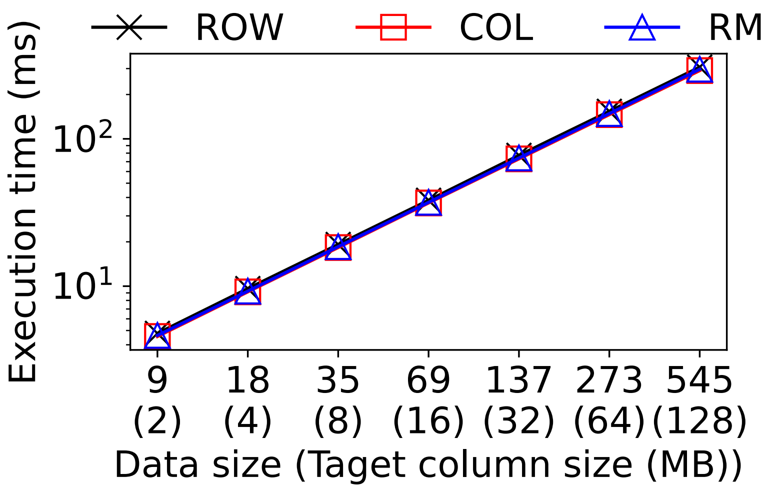
TPC-H Q1 CPU-bound (sort, group by)

TPC-H Q6 IO-bound



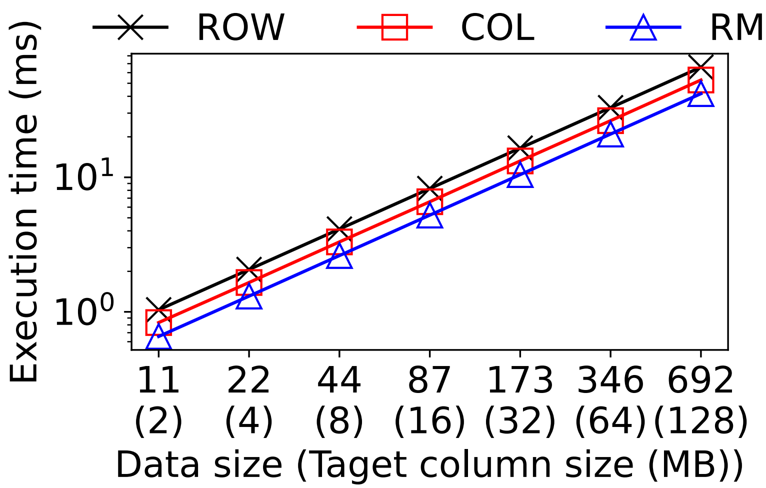
RME Scales with Data Size

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



CPU overhead dominates data movement cost

TPC-H Q6 IO-bound



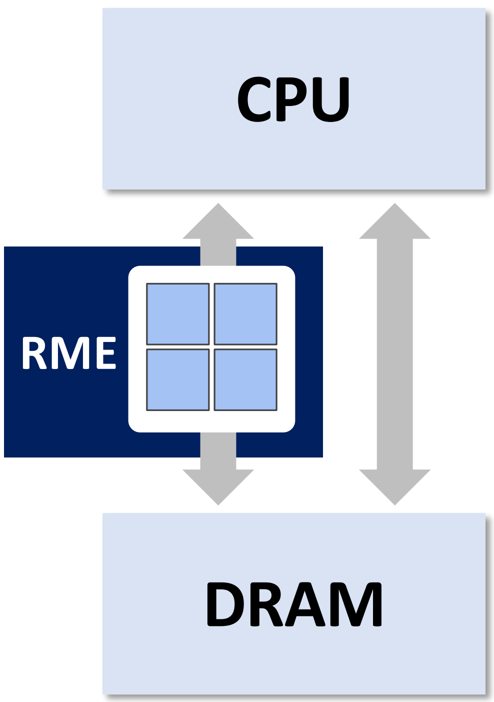
RME benefits regardless of data size

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



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

Thank you

Ju Hyoung Mun (jmun@bu.edu)