Transparent Data Transformation

or

How to stop worrying about data layouts?

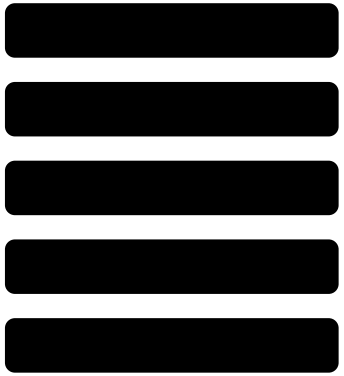
Manos Athanassoulis

mathan@bu.edu

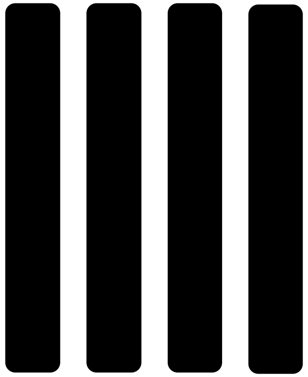
Talk at HPTS 2022

Break the *Fractured Mirrors*!

OLTP



OLAP

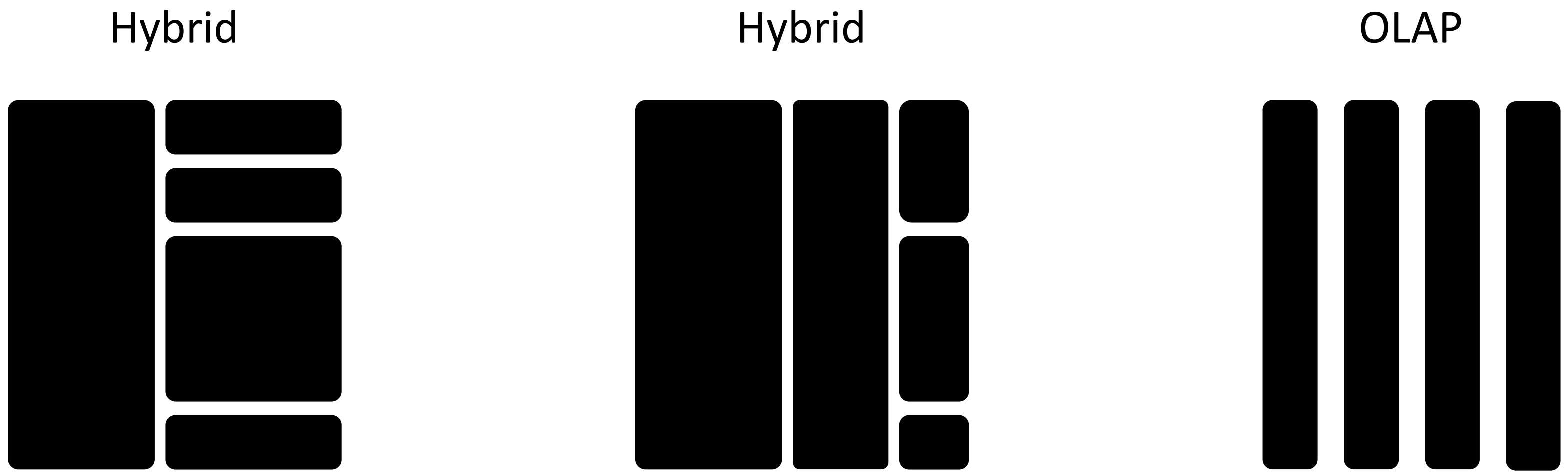


No need to have two systems!

No need to convert data!

What if we could have **the benefits of both** *without storing or maintaining two copies* of data?

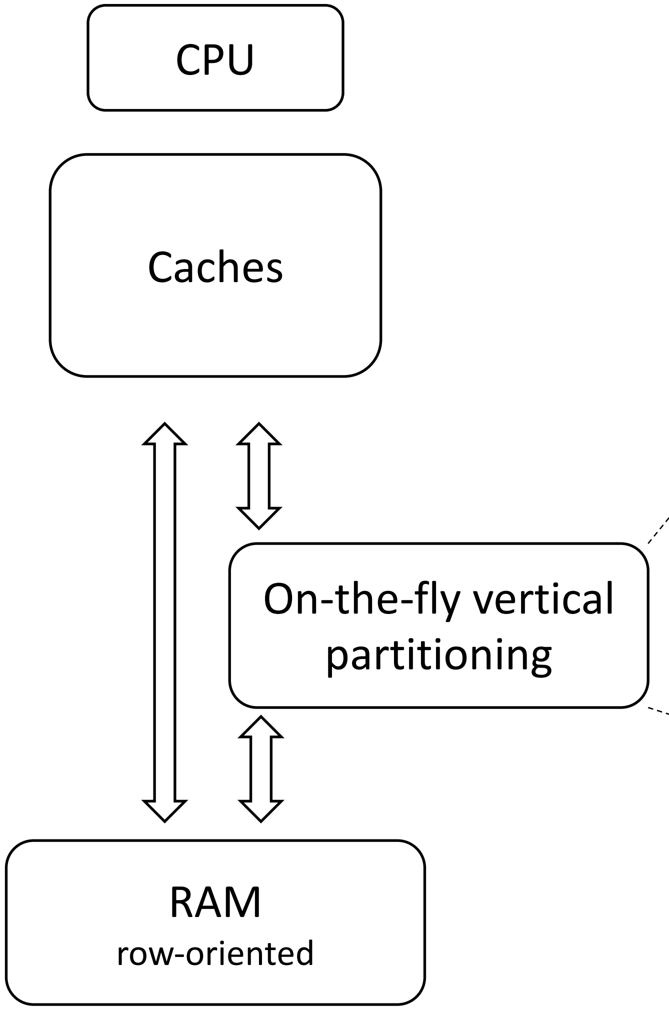
Bridge the *Archipelago* of Hybrid Layouts

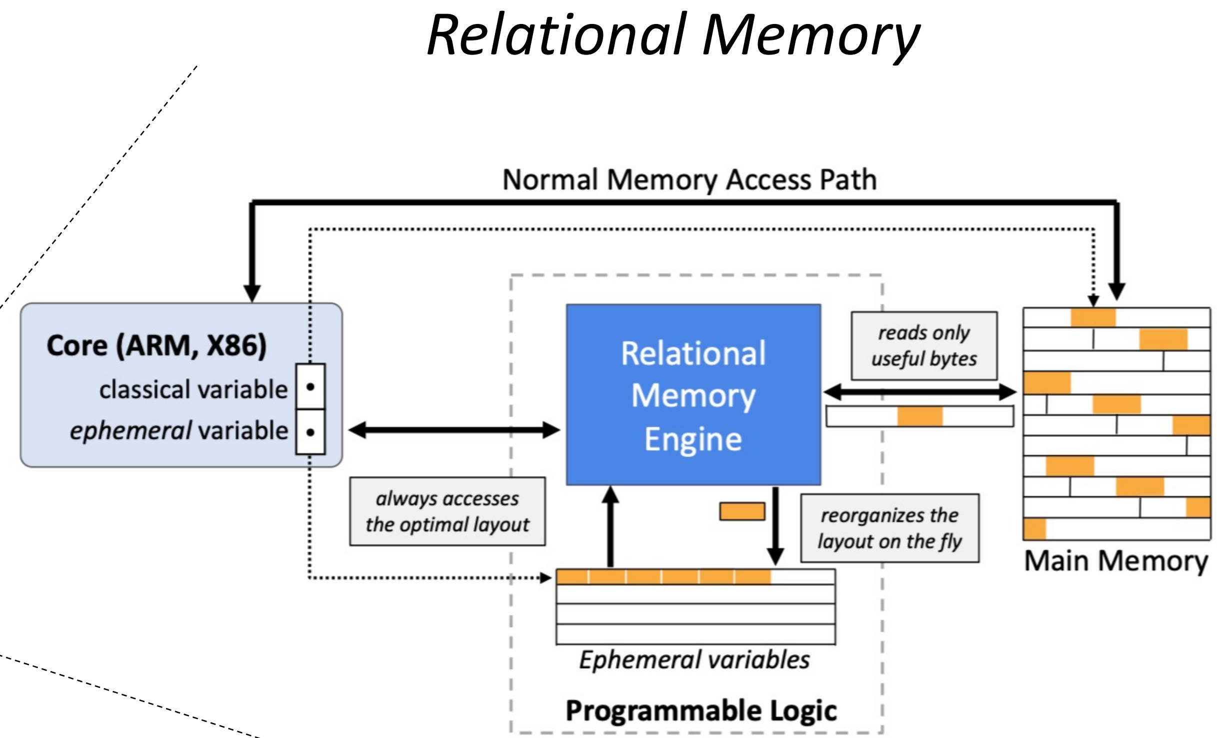


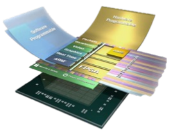
What if we could have **the benefits of any data layout?**

For free?

Our vision: **Relational Memory**







AMD XILINX UltraScale+

ENZIAN

intel AGILEX

Example

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
Eve	5	43	168	58
Frank	6	22	181	79
Greg	7	52	175	67
Henry	8	17	169	76
Iris	9	34	158	49
Jane	10	29	165	59
Kenneth	11	31	184	94
Luke	12	13	125	38

```
SELECT name, ID
       FROM table
       WHERE height>170;
```

Example

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
Eve	5	43	168	58
Frank	6	22	181	79
Greg	7	52	175	67
Henry	8	17	169	76
Iris	9	34	158	49
Jane	10	29	165	59
Kenneth	11	31	184	94
Luke	12	13	125	38

```
SELECT name, ID
    FROM table
    WHERE height>170;
```

CPU

SELECT name, ID
FROM table
WHERE height>170;

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	ID	height
Alice	1	120
Bob	2	175
Charles	3	168
David	4	179

CPU

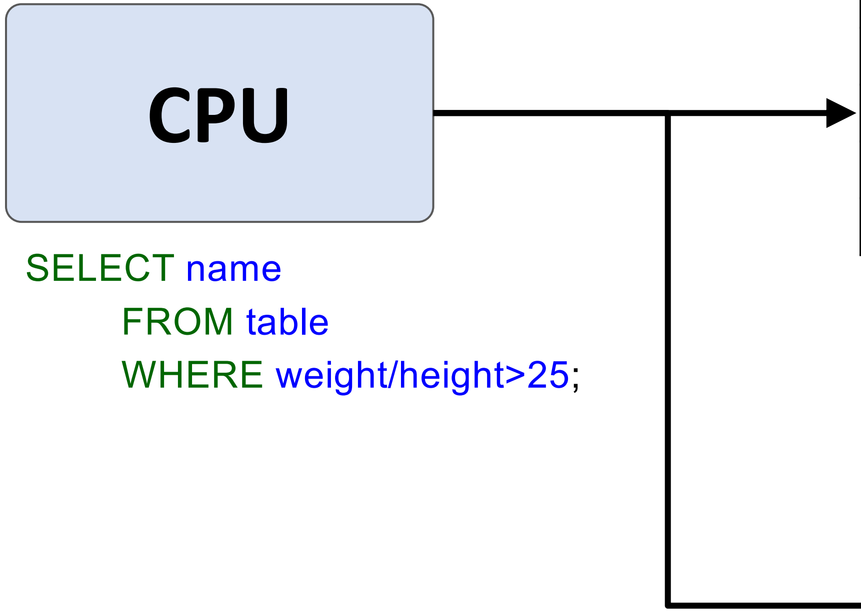
SELECT AVG (height)
FROM table;

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

height
120
175
168
179




```
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

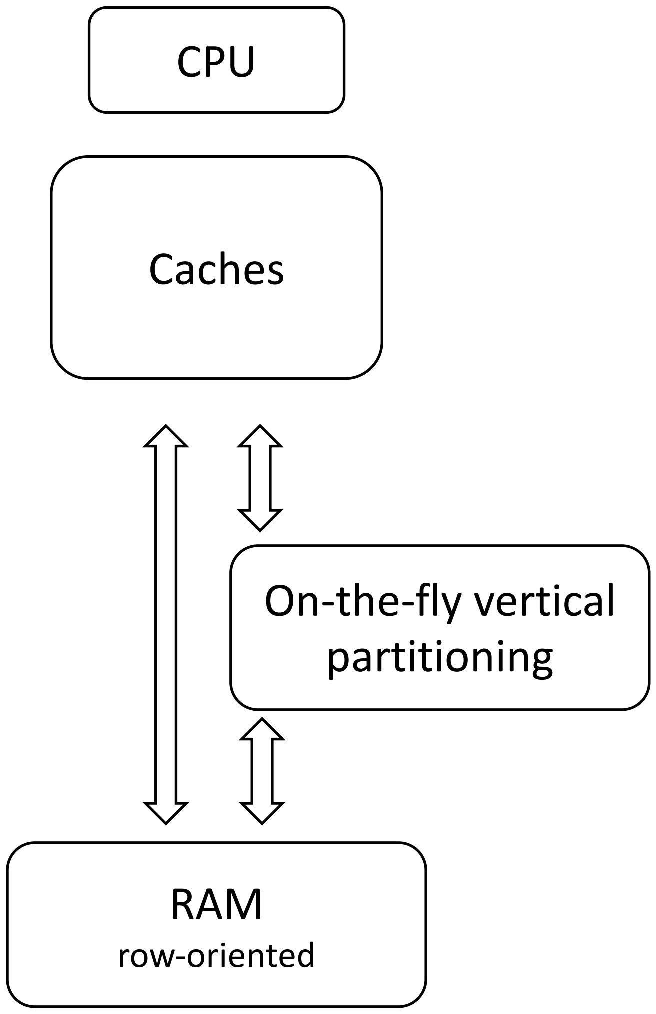
how to use a Relational Memory Engine (RME) to access the desired columns?

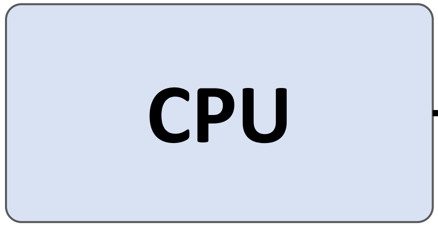
leads to normal memory accesses

variable table[ ];
ephemeral variable col_group[ ];

”fake” address that is intercepted by RME, but CPU thinks it exists

ephemeral variables






```
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 ;
};
```

***ephemeral* variable**

```
struct row the_table[ ];
struct column_group* cg1;
cg1 = configure (the_table, column_group);

for (int i = 0 ; i < cg1.length ; i++) {
      if (cg1[i].weight/cg1[i].height > 25) { output cg1[i].name; }
}
```

CPU

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

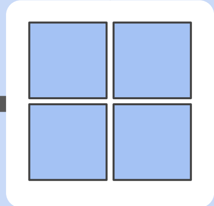
base row store

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



Programmable logic in the middle

***ephemeral* variable**

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

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 ;
};
```

***on-the-fly* data transformation**

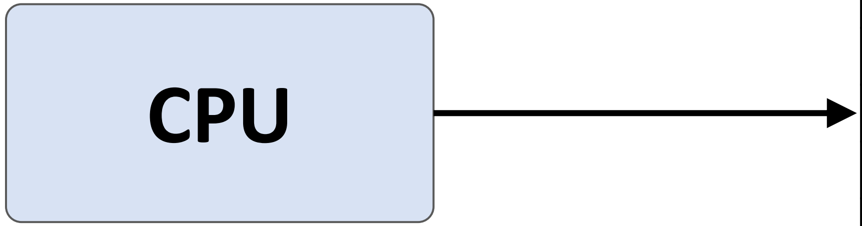
optimal layout

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

Programmable logic in the middle

***ephemeral* variable**

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

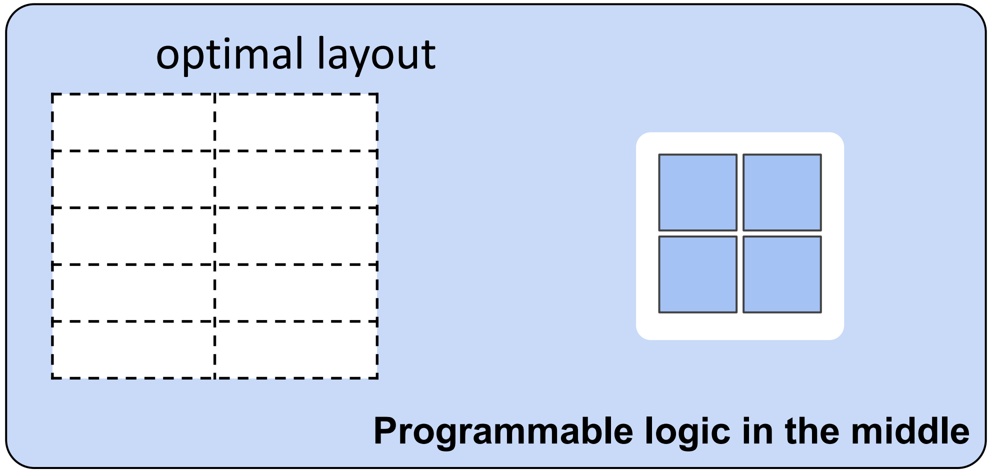


SELECT ID FROM table
WHERE age>40;

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 ;
};
```



CPU

SELECT ID FROM table
WHERE age>40;

base row store

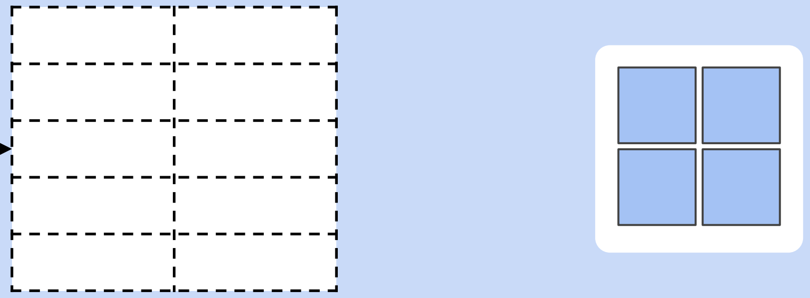
name	ID	age	height	weight
Alice	1	10	120	34
Bob	2	71	175	74
Charles	3	37	168	61
David	4	25	179	80

```
struct row {
 string name;
 int ID ;
 int age ;
 int height ;
 int weight ;
};
```

***ephemeral* variable**

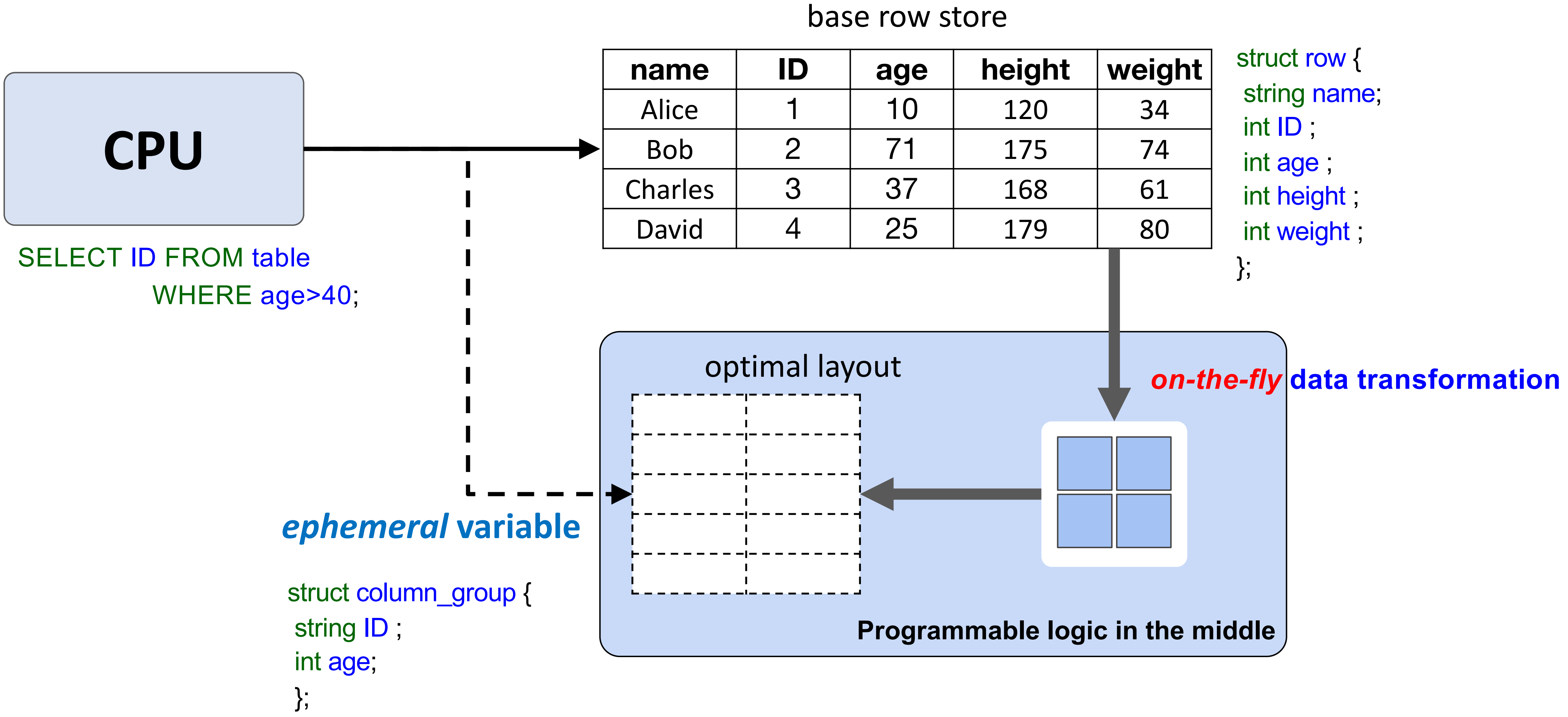
```
struct column_group {
 string ID ;
 int age;
};
```

optimal layout



Programmable logic in the middle

BOSTON UNIVERSITY



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

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 ID FROM table
        WHERE age>40;
```

on-the-fly **data transformation**

optimal layout

ID	age
1	10
2	71
3	37
4	25

Programmable logic in the middle

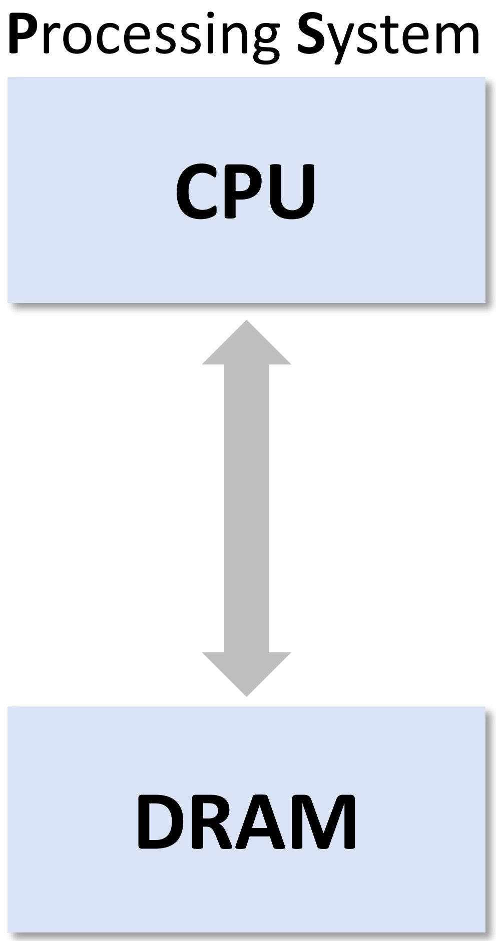
ephemeral **variable**

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

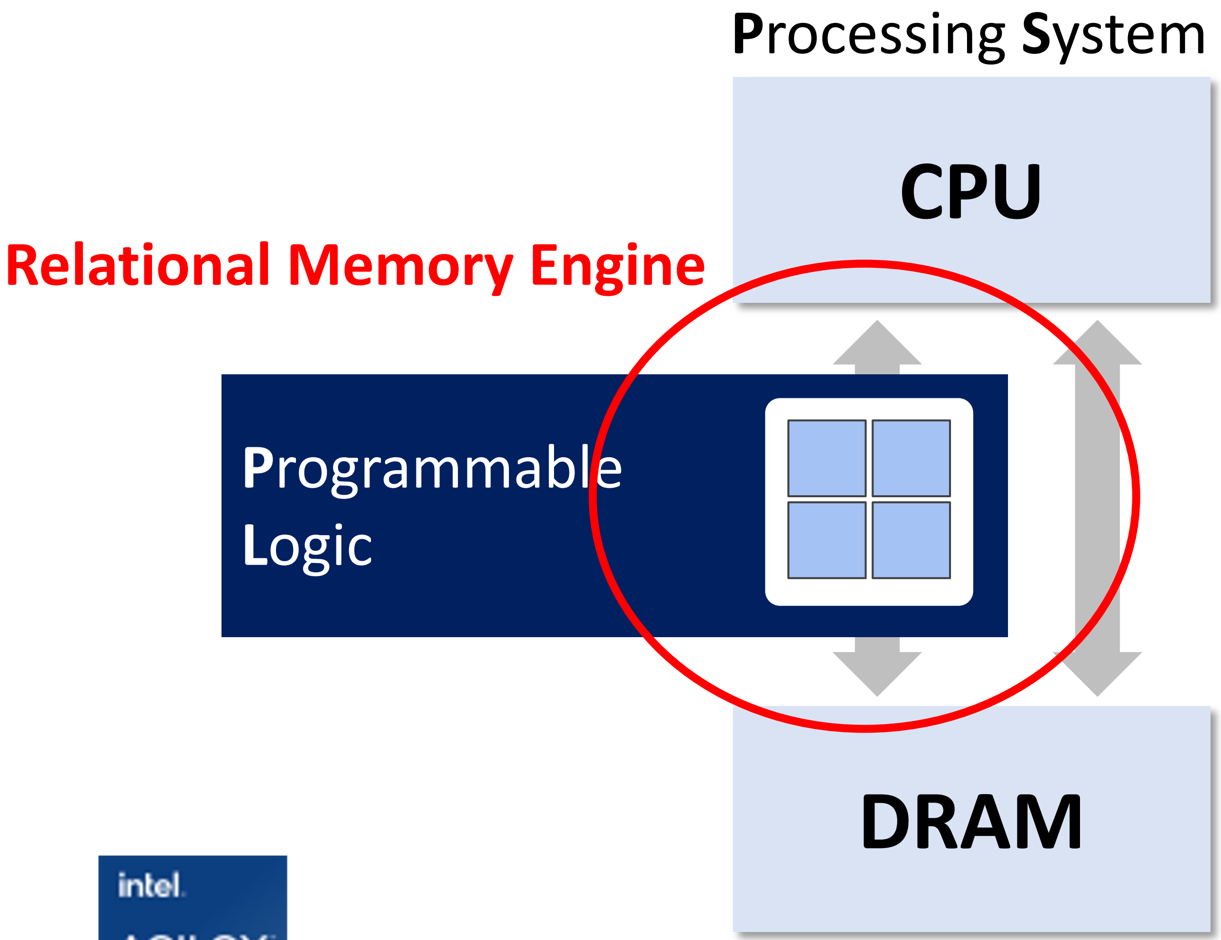
Programmable **L**ogic **I**n the **M**iddle

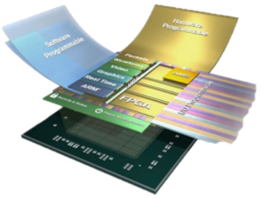
The Potential of Programmable Logic in the Middle: Cache Bleaching, RTAS 2020

Processing System



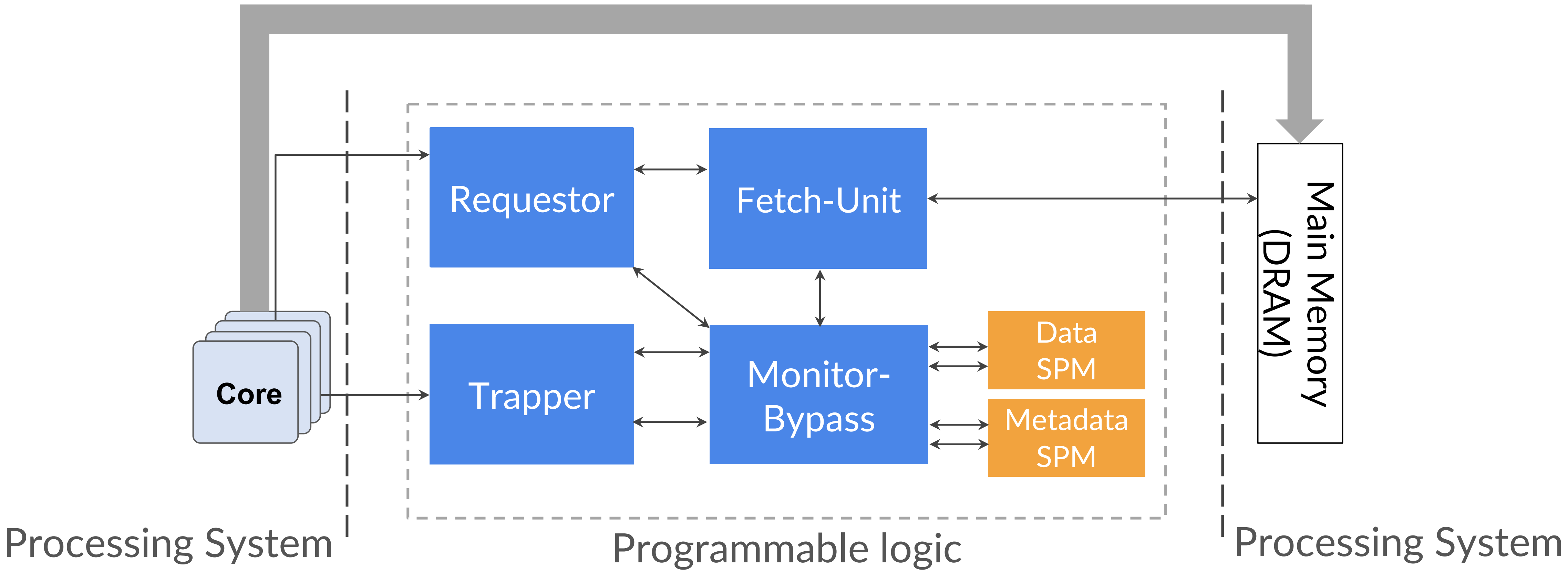
PS-PL Platforms



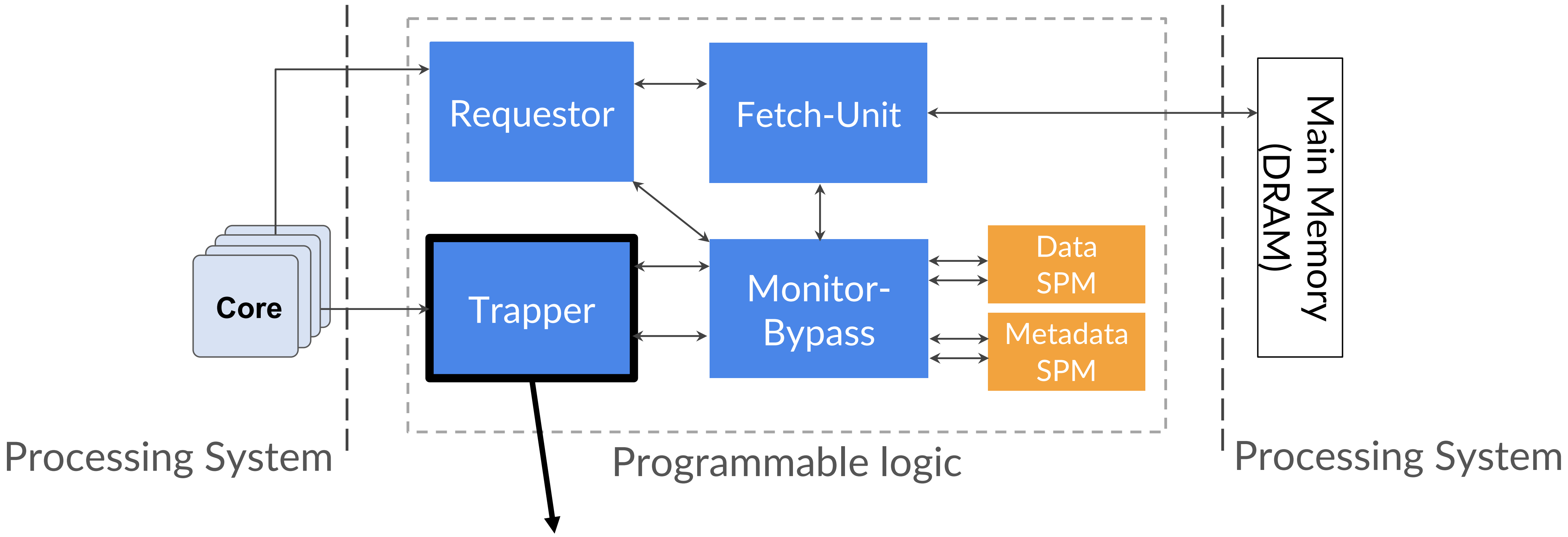


AMD XILINX UltraScale+ ENZIAN intel AGILEX

Relational Memory Engine

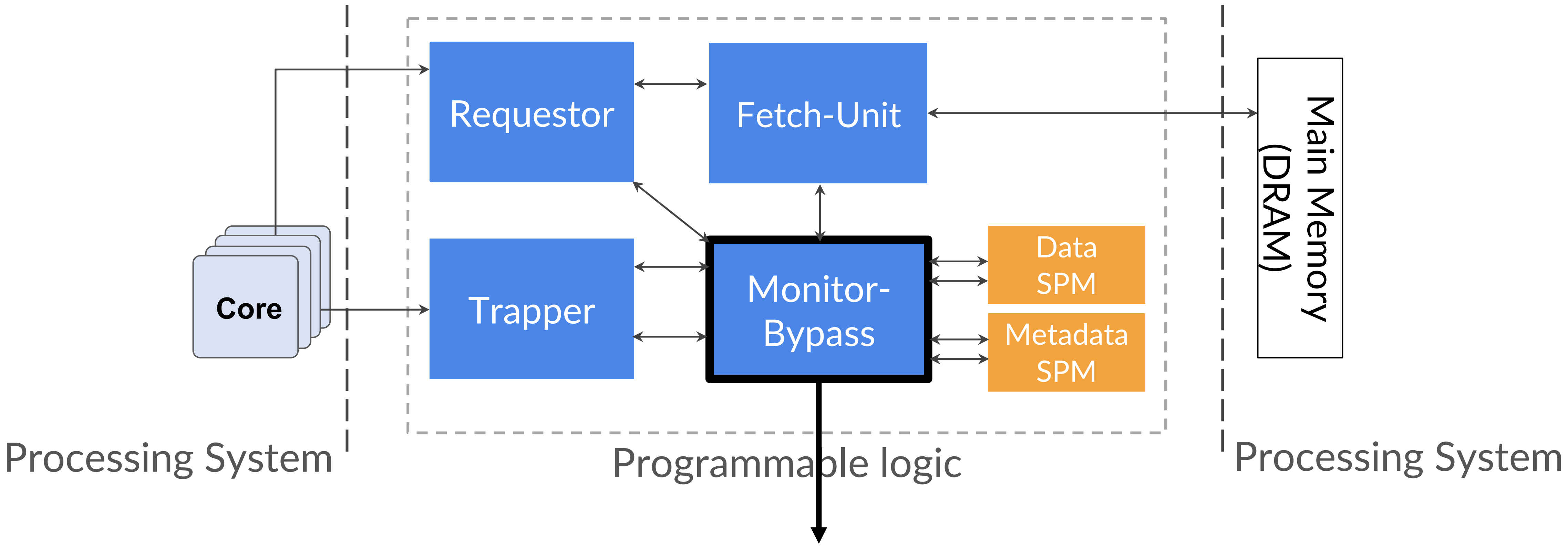


Relational Memory Engine



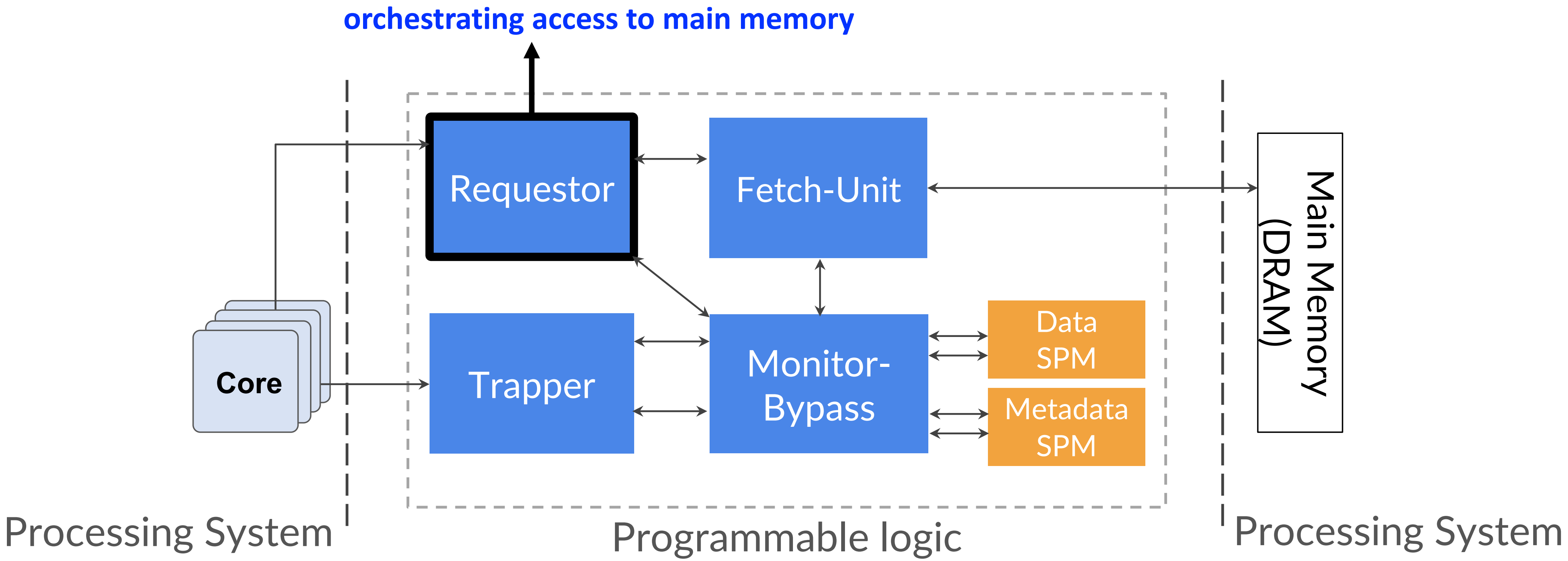
The interface between RME and the CPUs

Relational Memory Engine

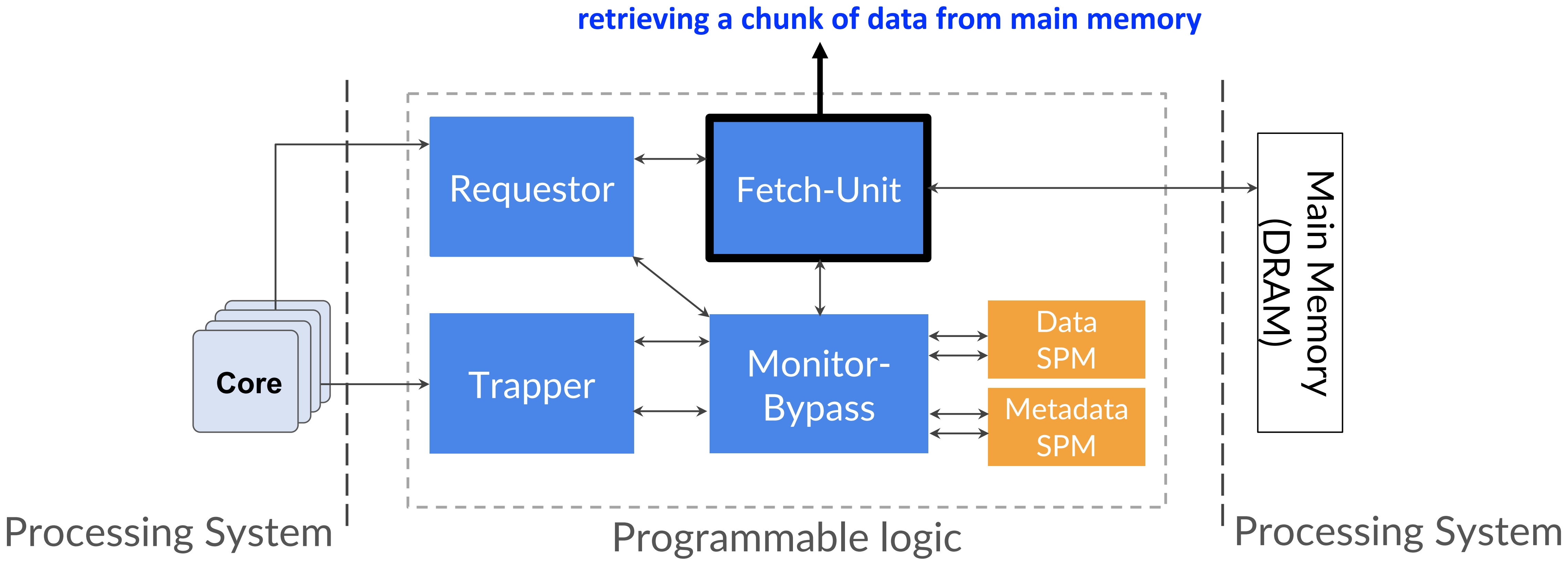


Monitoring the completion and availability status of each reorganized data

Relational Memory Engine



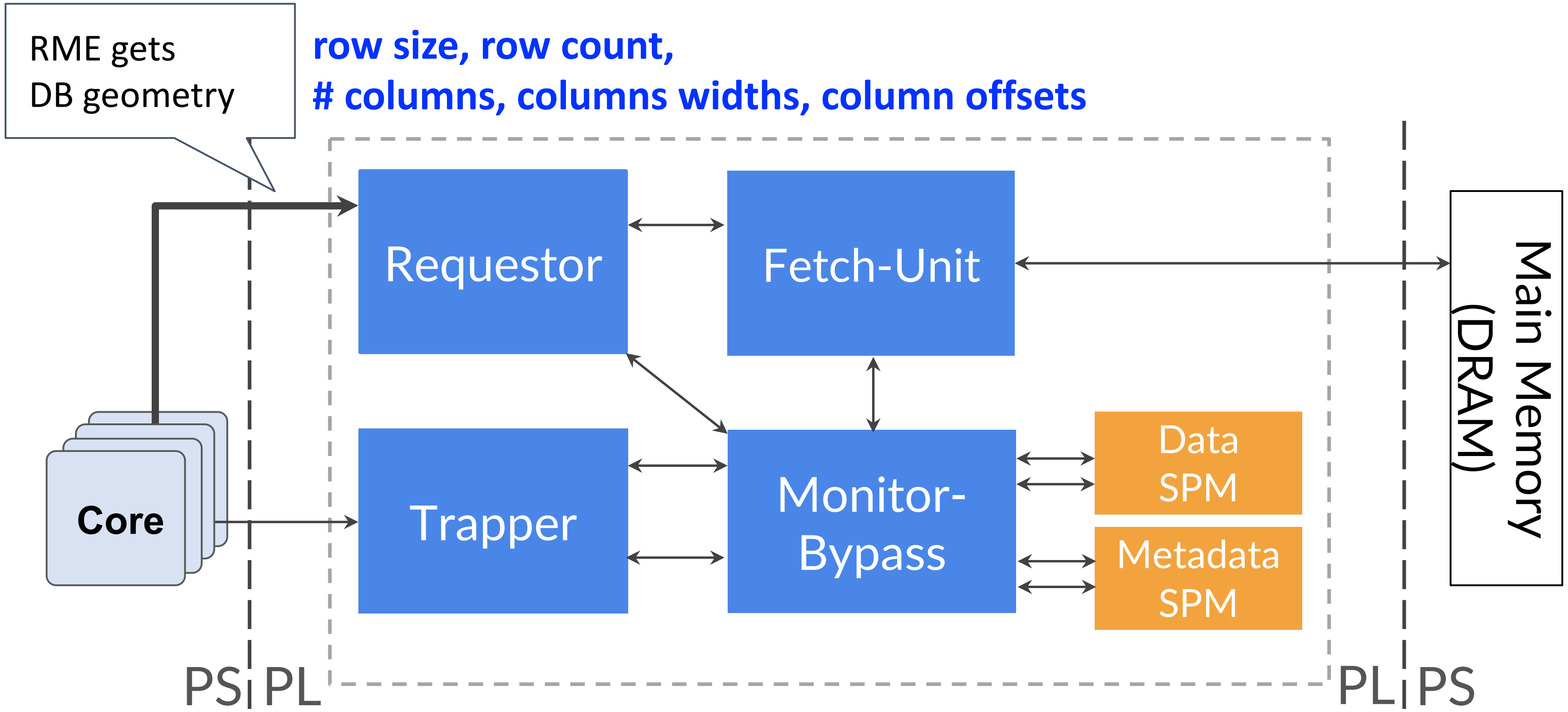
Relational Memory Engine



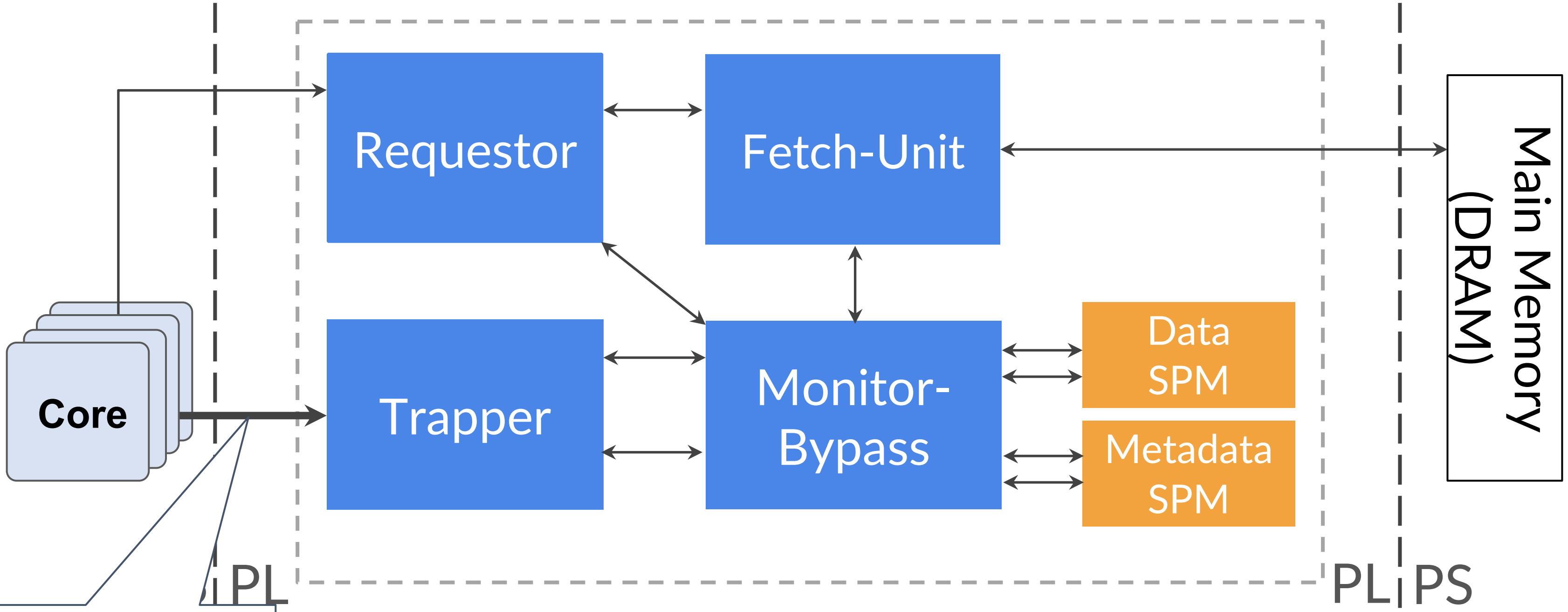
Relational Memory Engine

cg1 = configure (the_table, column_group);

row size, row count,
 # columns, columns widths, column offsets



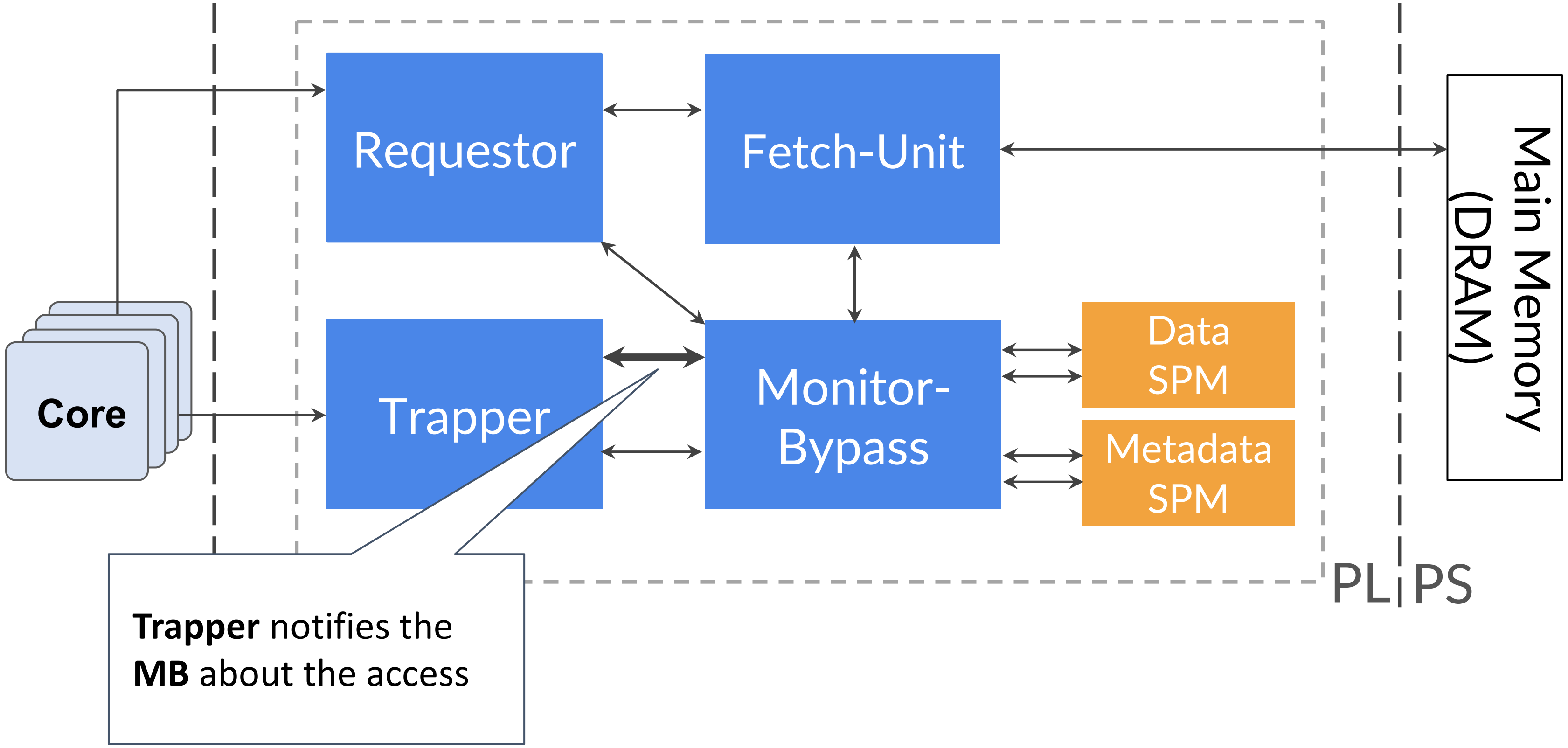
Relational Memory Engine



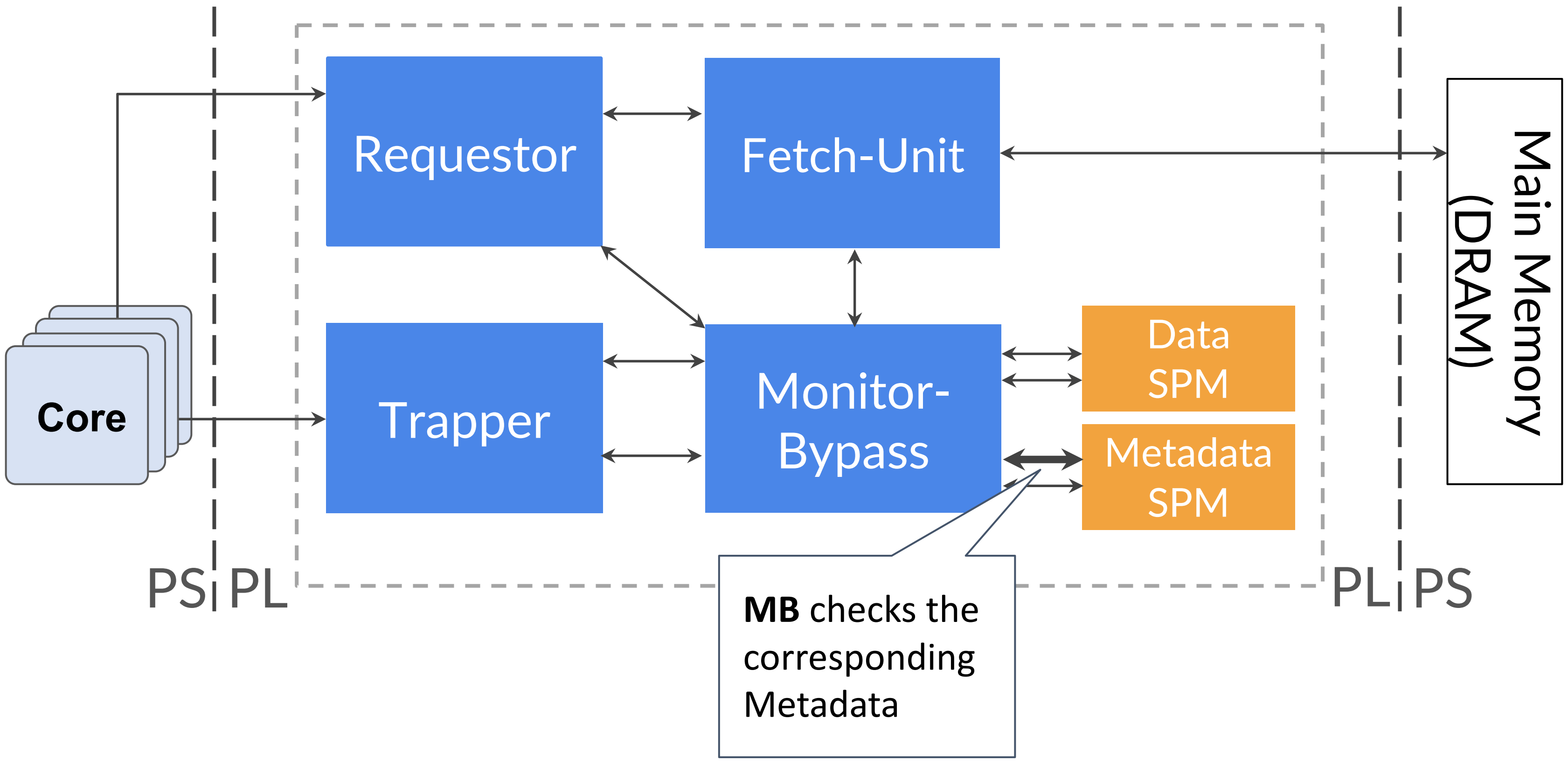
Read accesses towards ***ephemeral variable*** are intercepted by Trapper

```
if (cg1[i].weight/cg1[i].height > 25) {
    output cg1[i].name; }
```

Relational Memory Engine

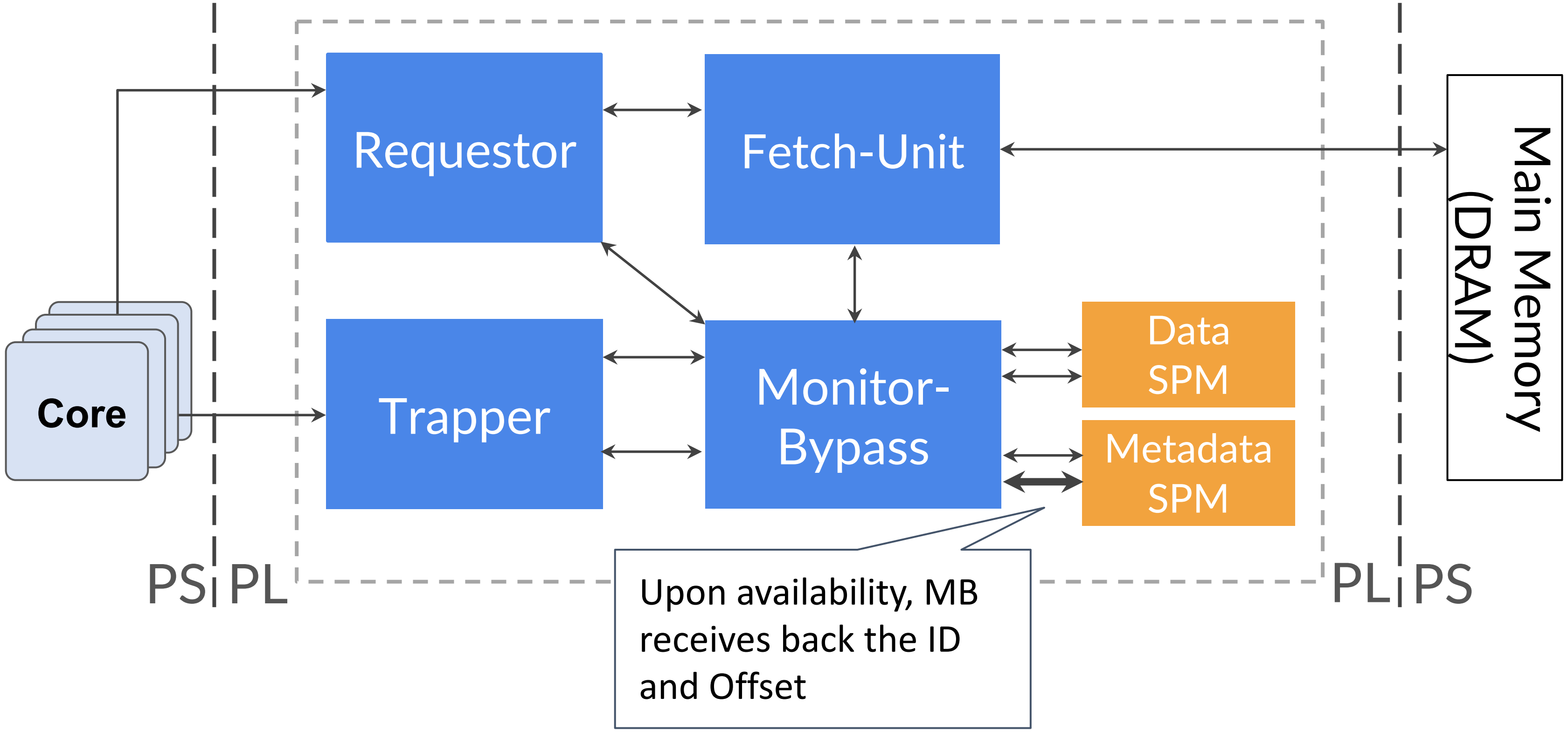


Relational Memory Engine



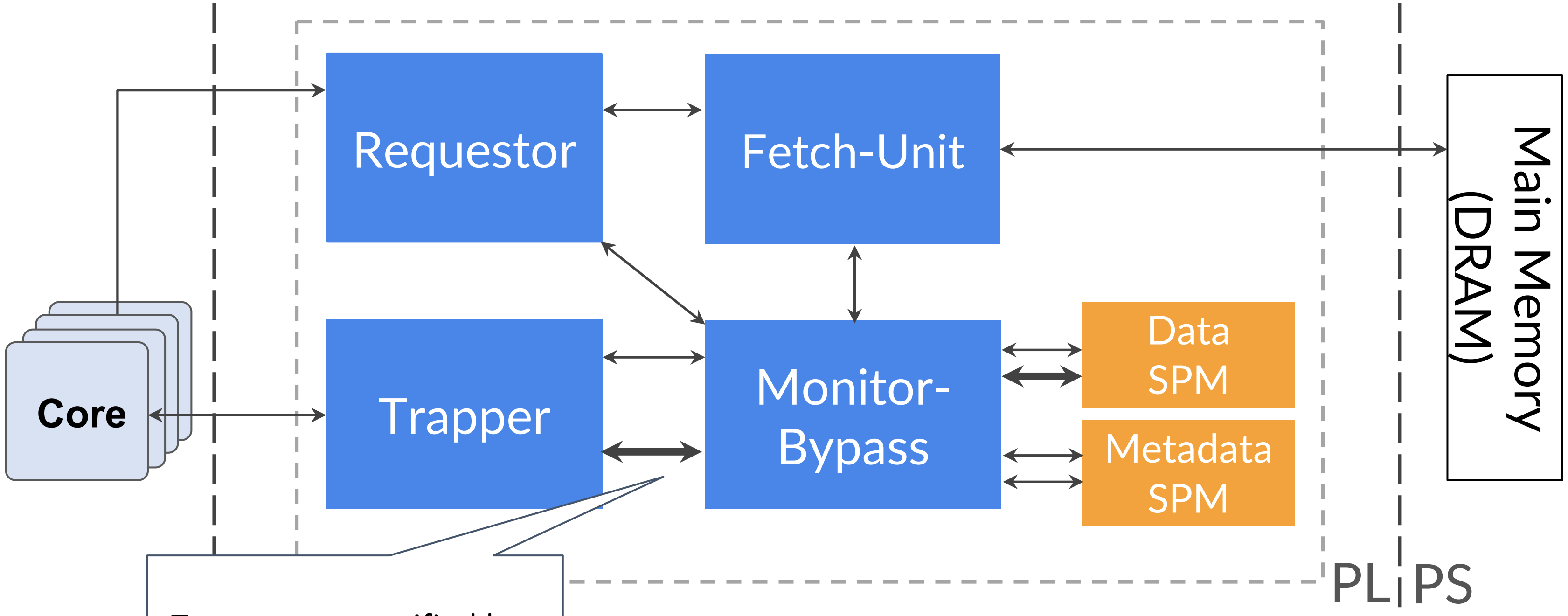
Relational Memory Engine

When the data is already in Data SPM



Relational Memory Engine

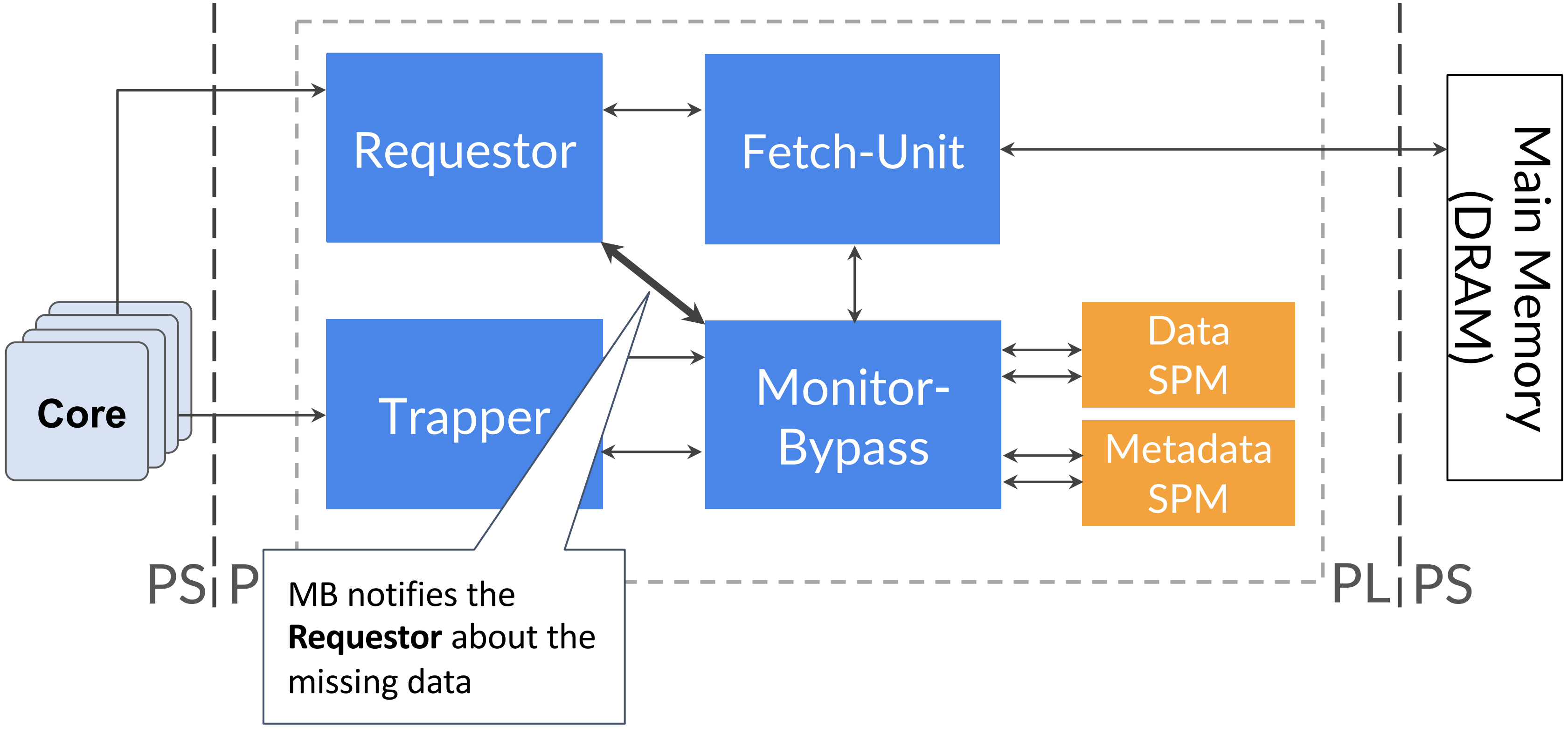
When the data is already in Data SPM



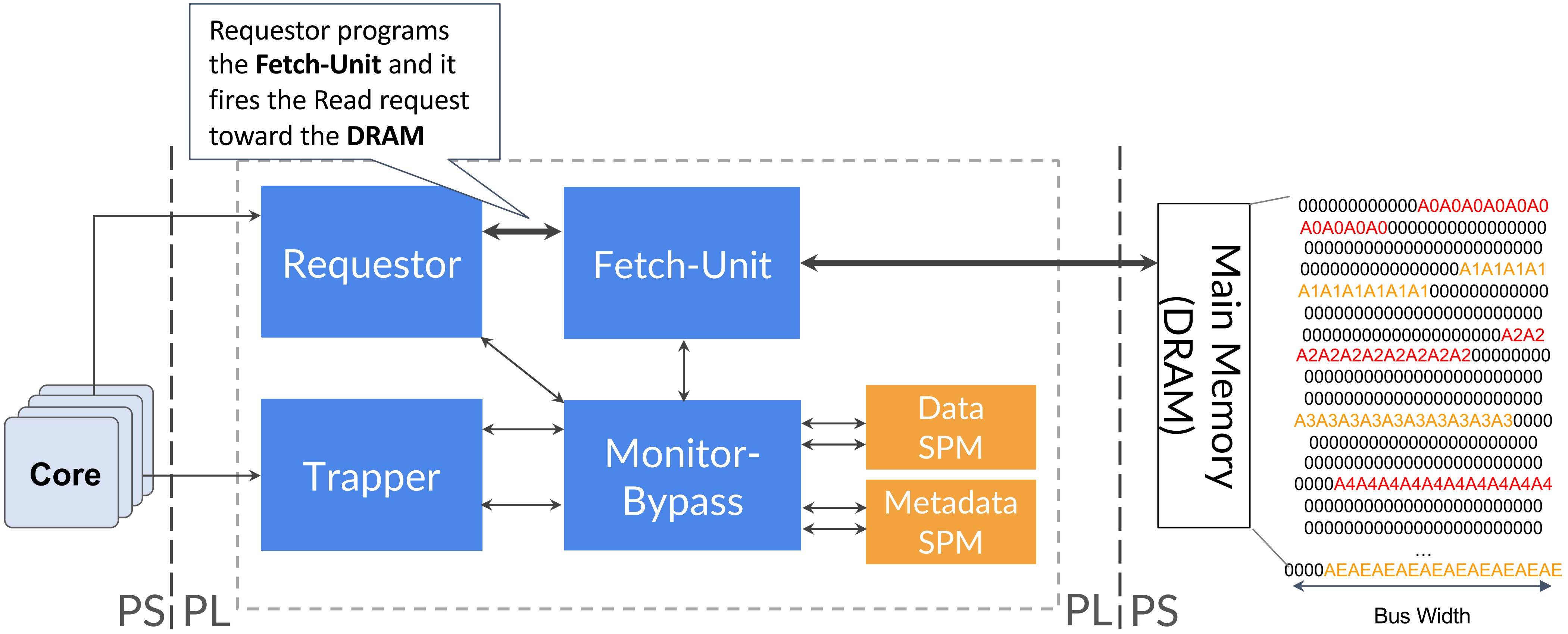
Trapper gets notified by MB and fetches the Data from Data SPM Location

Relational Memory Engine

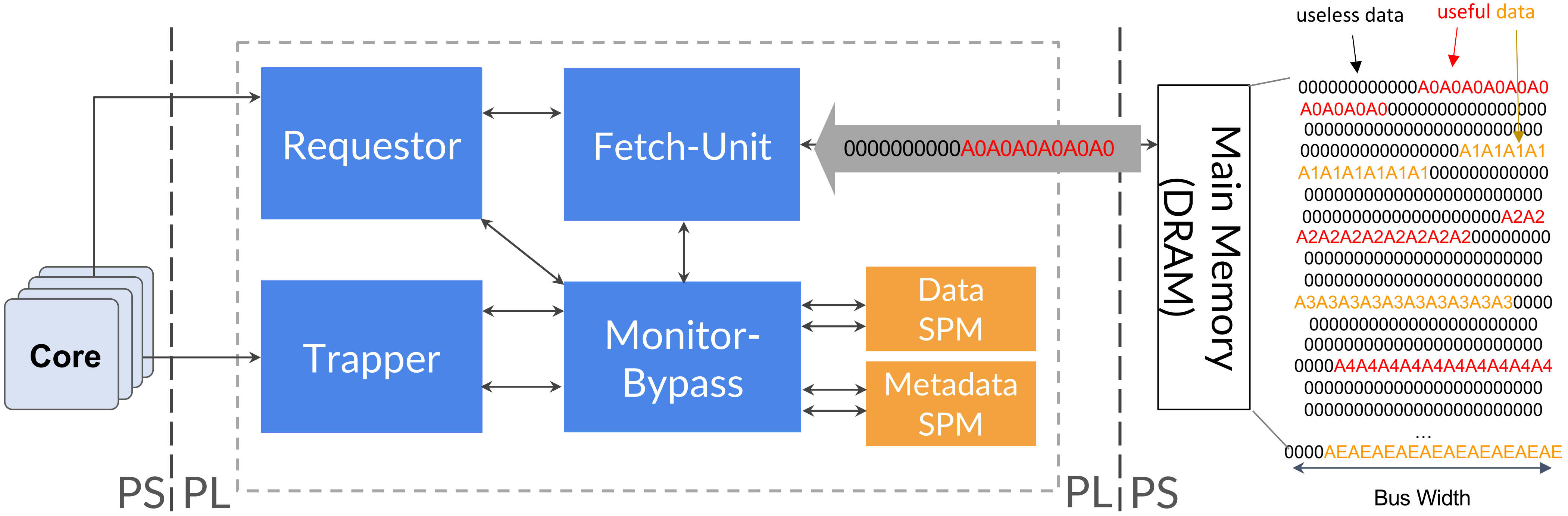
When the data is not in Data SPM



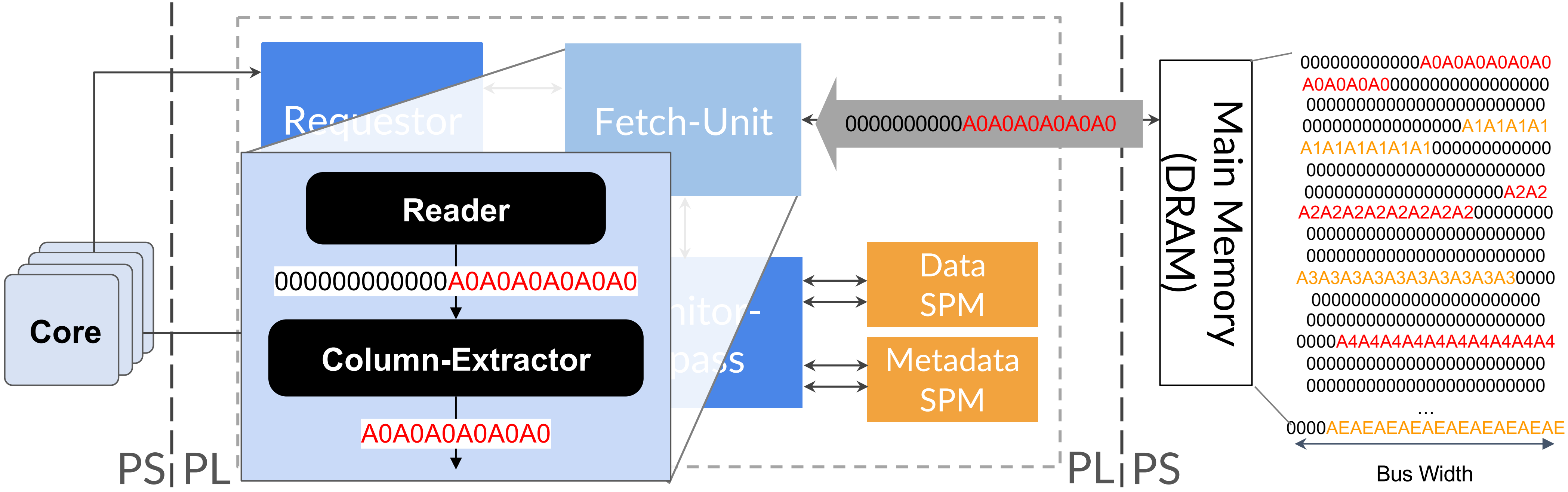
Relational Memory Engine



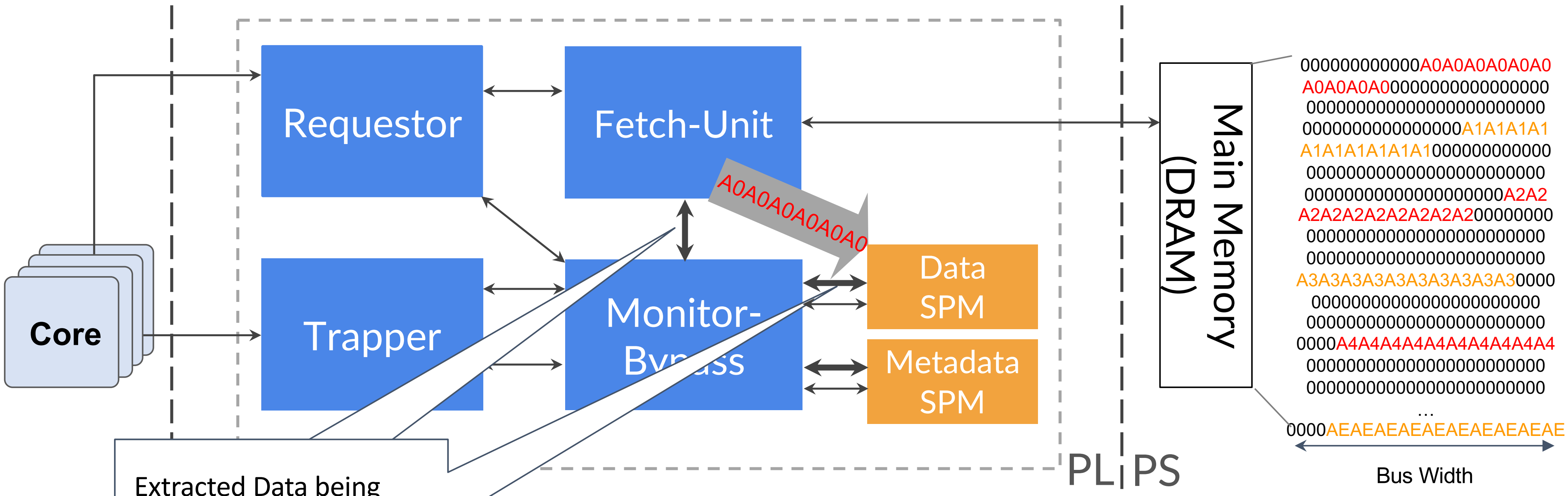
Relational Memory Engine



Relational Memory Engine

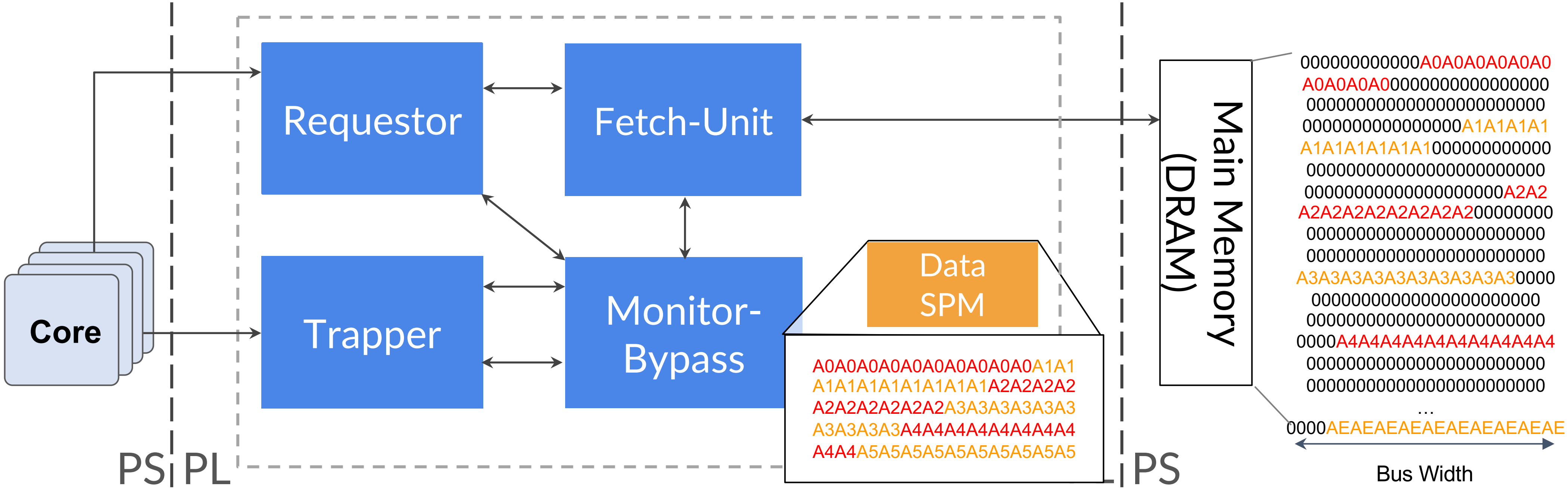


Relational Memory Engine



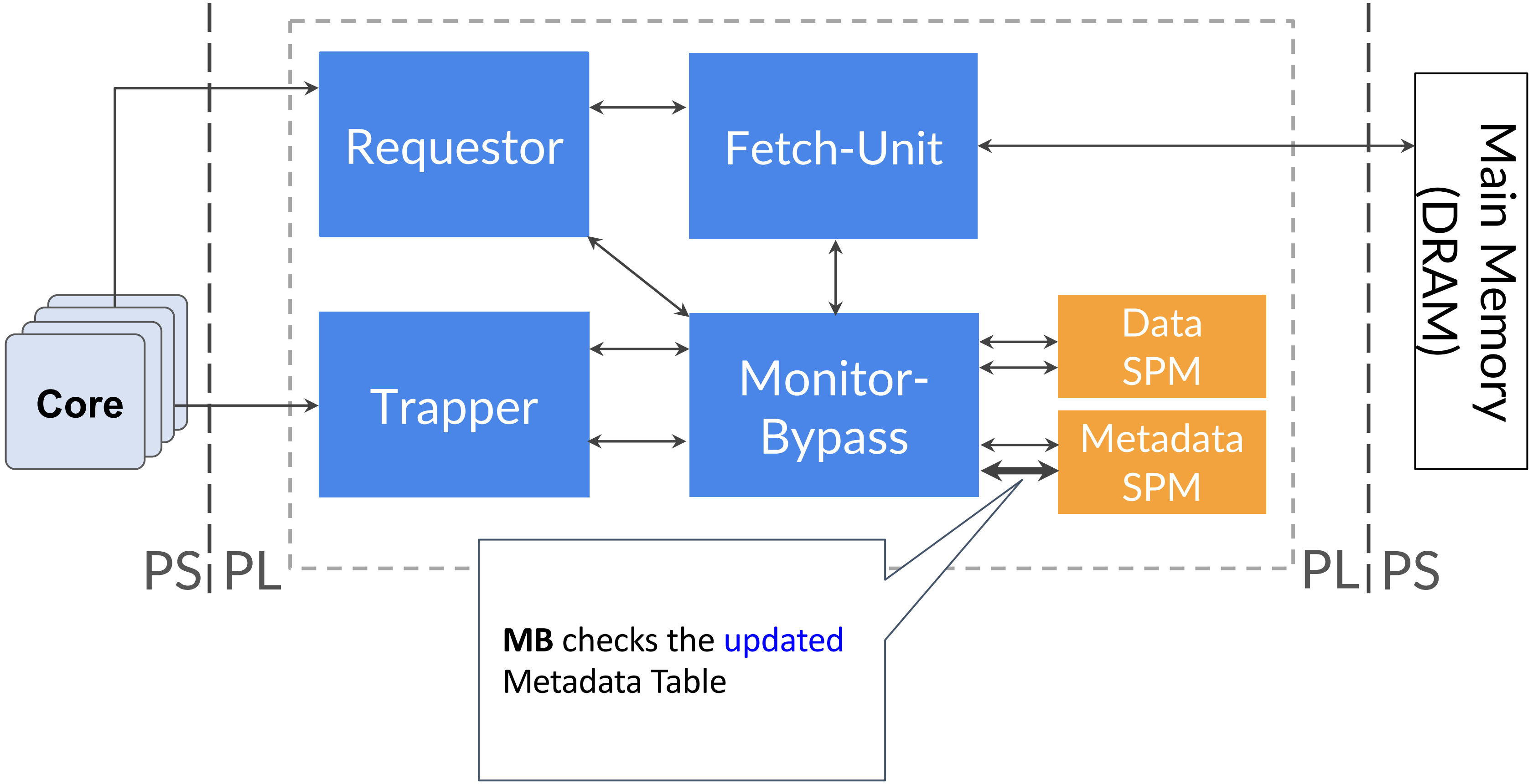
Extracted Data being sent toward the **(DATA) SPM** and **Metadate table** gets updated

Relational Memory Engine

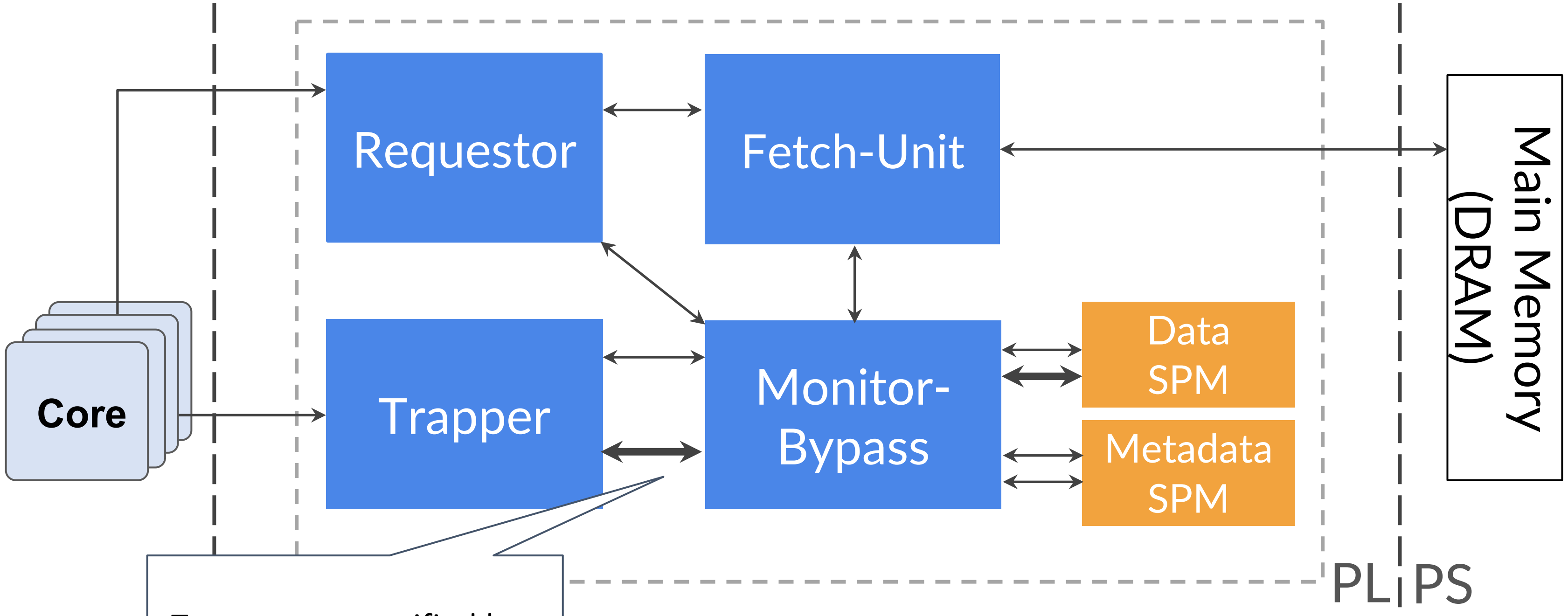


only the desired columns

Relational Memory Engine

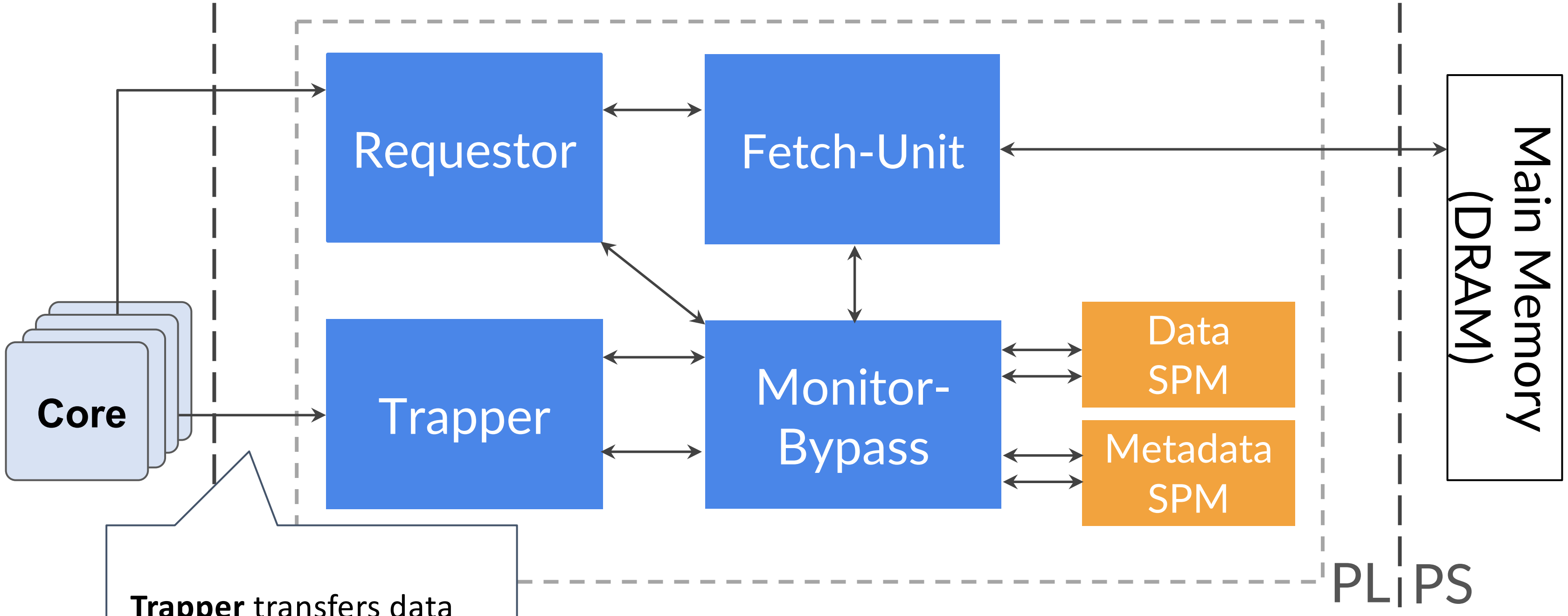


Relational Memory Engine



Trapper gets notified by MB and fetches the Data from Data SPM Location

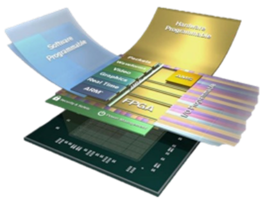
Relational Memory Engine



Trapper transfers data to pending transaction.

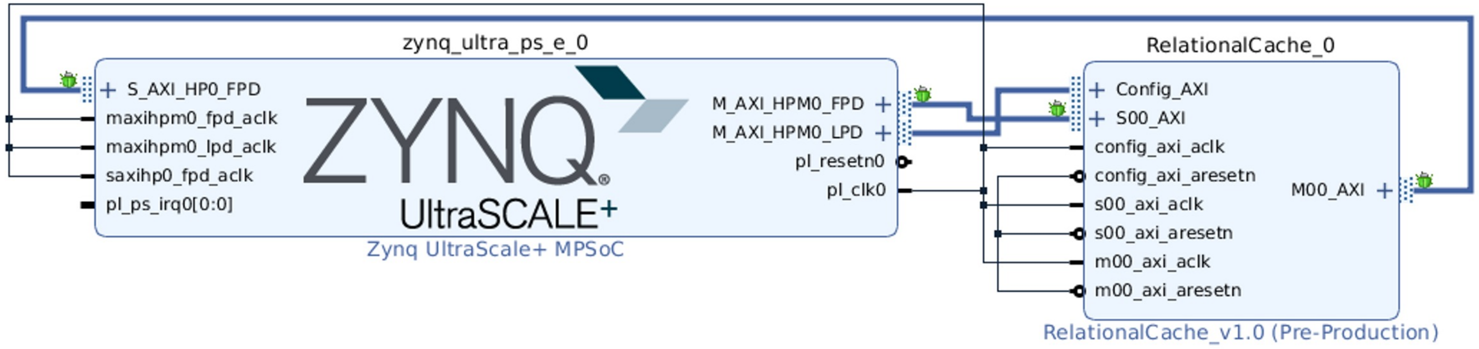
How big is the overhead of fetching the data vs. having them in Data SPM?

Evaluation



AMD XILINX
UltraScale+
ZCU102 platform

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



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

Relational Memory Benchmark

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

Q2: SELECT A1 FROM S WHERE A3 > k; → selection

Q3: SELECT A1 , A2 , ... , Ak FROM S WHERE C1, C2, ... ,Ci; → both projection & selection

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

Q5: SELECT S.A1 , R.A3 FROM S JOIN R ON S.A2 = R.A2;

→ complex queries group by & join

Approaches tested

ROW : Direct row-wise access

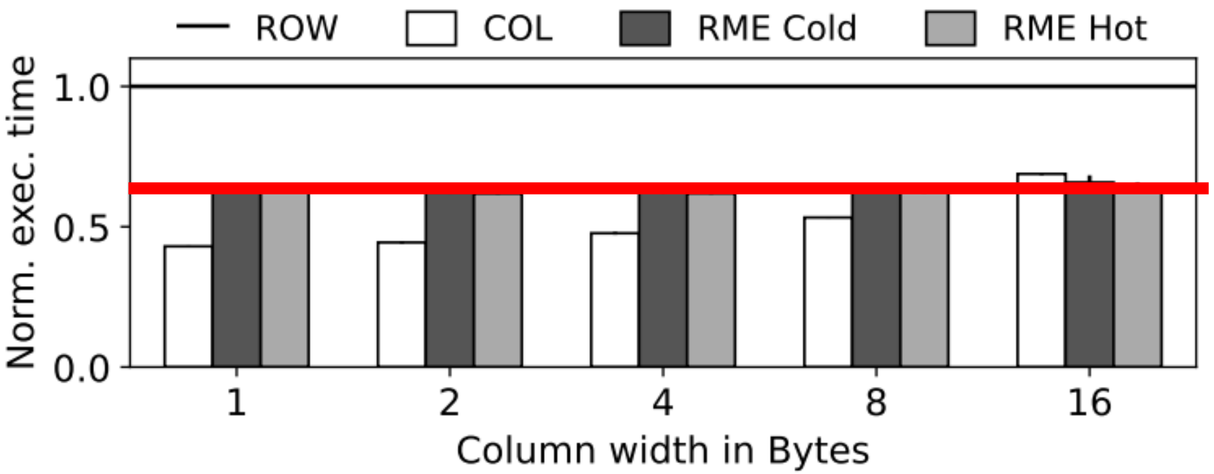
COL : Direct columnar access

RME : using Relational Memory Engine

RME Cold vs. Hot

How big is the overhead of fetching the data vs. having them in Data SPM?

Q0: SELECT avg(A1) FROM S;

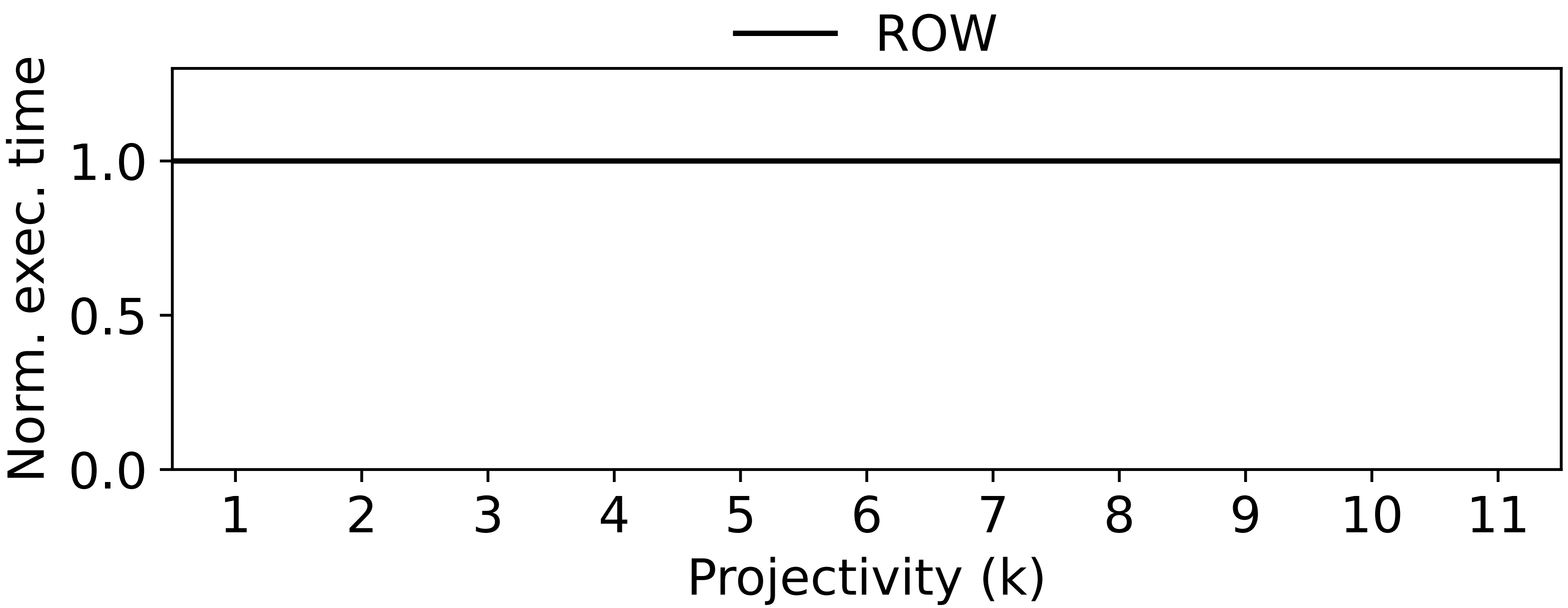


RME is comparable with directly accessing a single column!

RME Cold has virtually the same performance as RME Hot!

Queries varying Projectivity

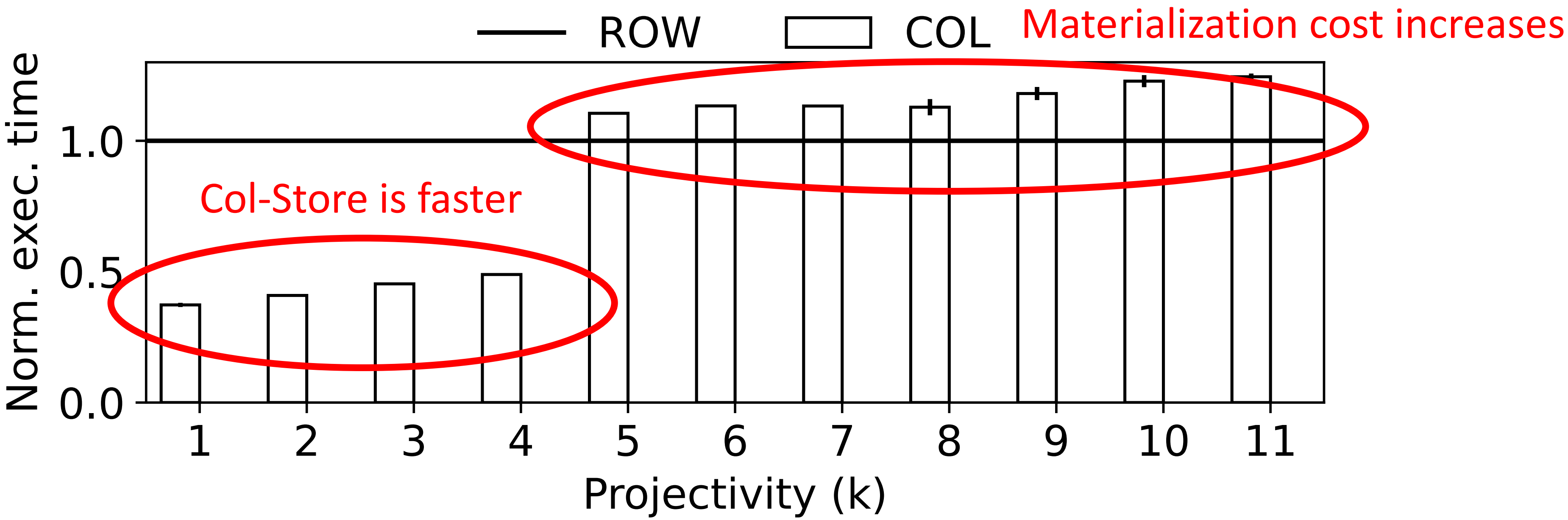
Q1: SELECT A1 , A2 , … , Ak FROM S;



Row size: 64 Bytes, Column size: 4 Bytes

Queries varying Projectivity

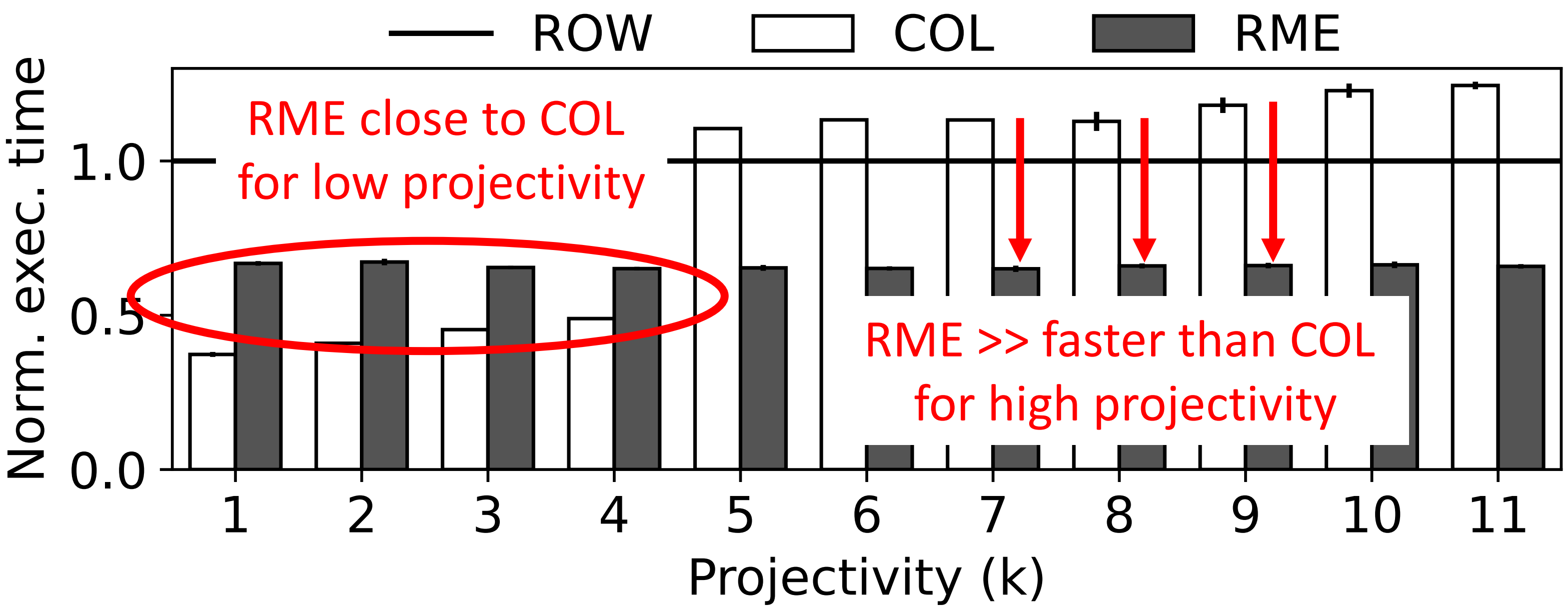
Q1: SELECT A1 , A2 , … , Ak FROM S;



Row size: 64 Bytes, Column size: 4 Bytes

Queries varying Projectivity

Q1: SELECT A1 , A2 , … , Ak FROM S;



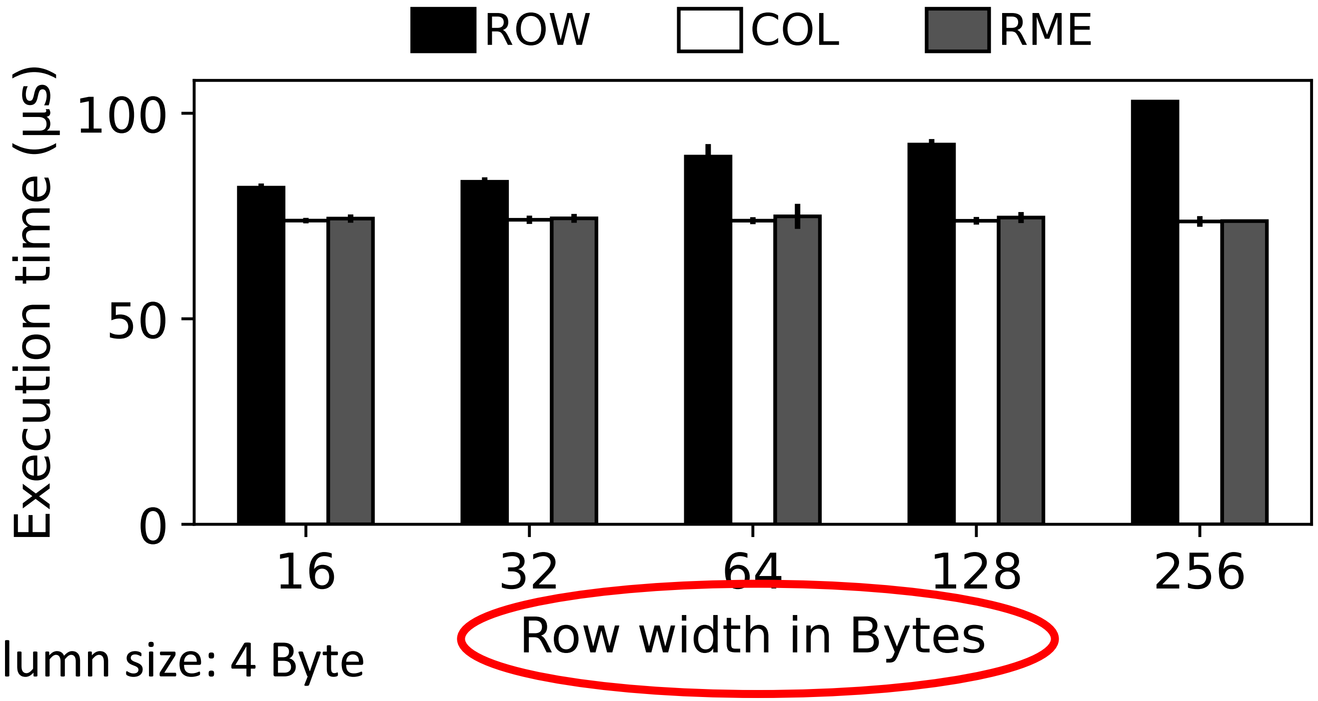
RME provides stable performance as we vary projectivity

Row size: 64 Bytes, Column size: 4 Bytes

RME is not affected by row width

Q2: SELECT A1 FROM S WHERE A3 > k;

Selectivity: 90%

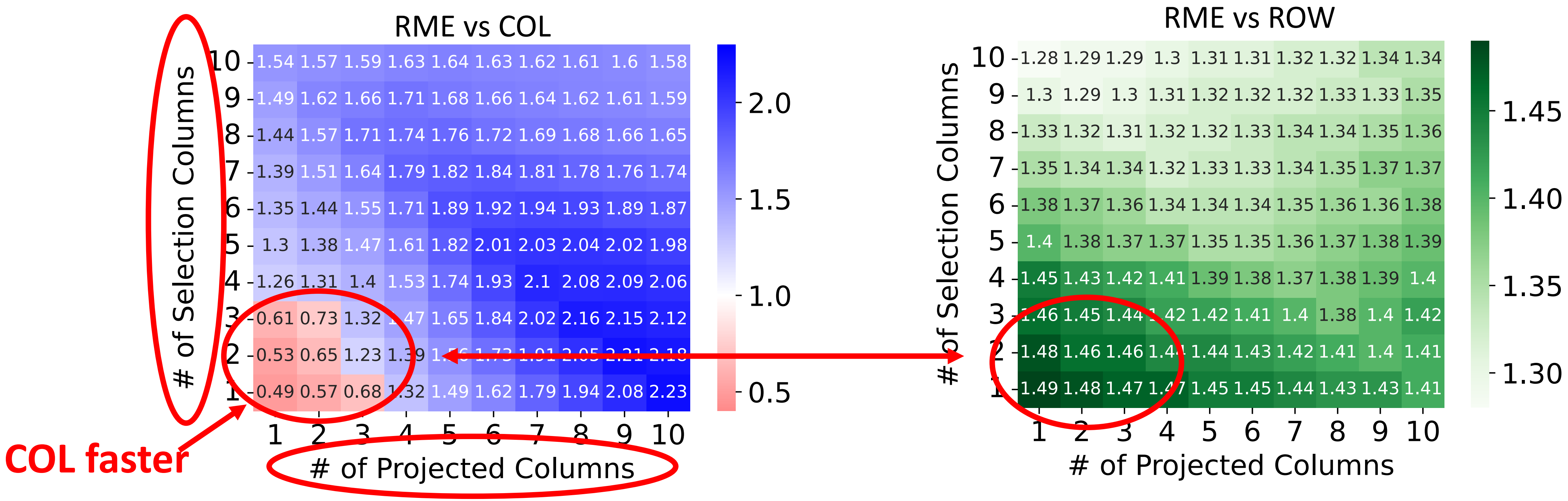


Column size: 4 Byte

Irrespectively of the row width, RME always accesses only 2 columns (like COL)

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

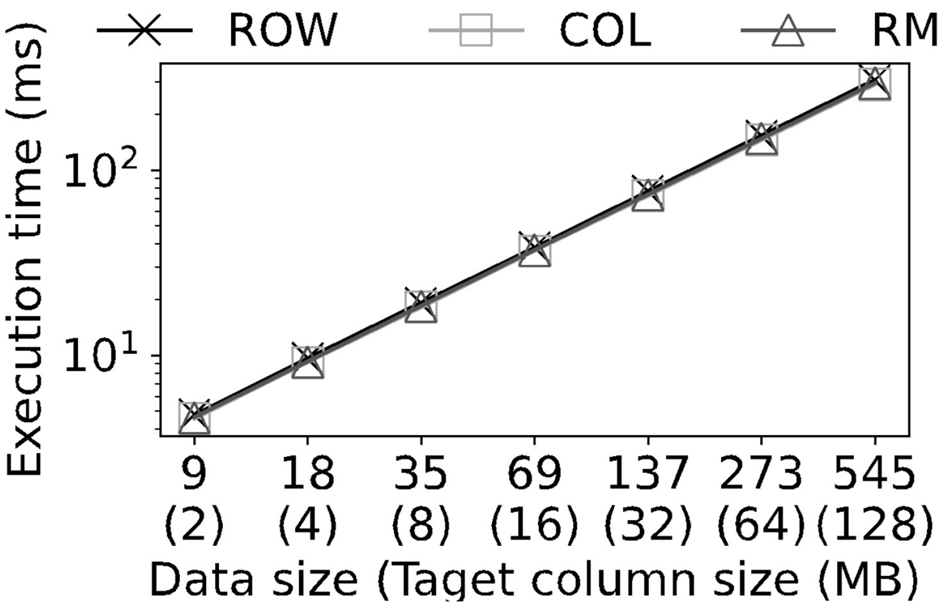


RME can be up to 2.23× faster than columnar access

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

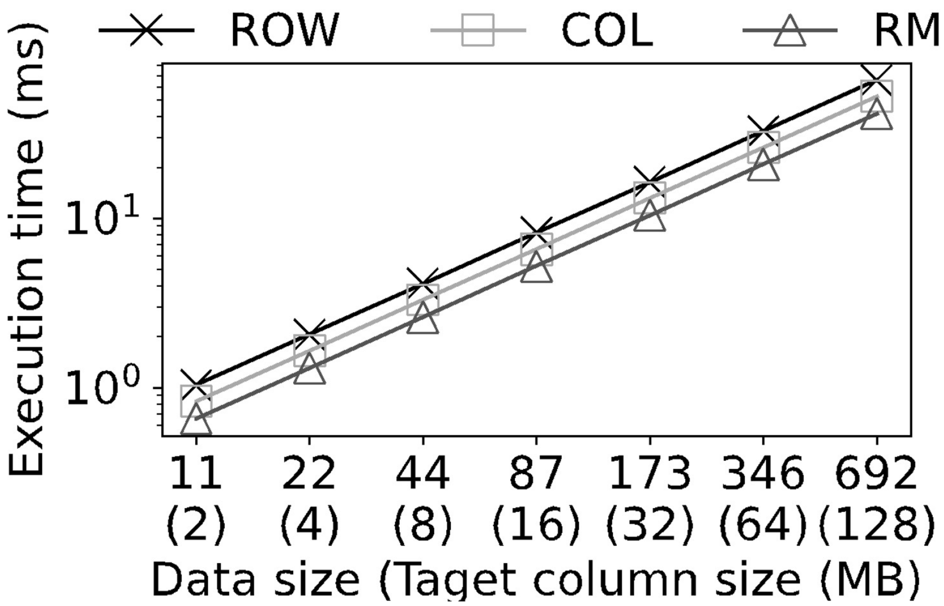
RME Scales with Data Size

TPC-H Q1 sel: 95%, proj: 24%, **CPU-bound** (sort/group by)



The CPU overhead of the query dominates the data movement cost

TPC-H Q6 sel: 15%, proj: 18%, **IO-bound**



RME benefits regardless of the data size by offering optimal layout

Updates

Can we perform updates through ephemeral variables?

No, ephemeral variables are read-only, but …

How to cater for HTAP workloads?

… RME can manage timestamps, allowing MVCC

Updates go to base row-oriented data (invalidated old/add new version of row)

In flight-queries will always read correct data (MVCC)

Updates - Example

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
Eve	5	43	168	58
Frank	6	22	181	79
Greg	7	52	175	67
Henry	8	17	169	76
Iris	9	34	158	49
Jane	10	29	165	59
Kenneth	11	31	184	94
Luke	12	13	125	38

Updates - Example

Data inserted at time t_1 and now valid

name	ID	age	height	weight	TS_{from}	TS_{to}
Alice	1	10	120	34	t_1	∞
Bob	2	71	175	74	t_1	∞
Charles	3	37	168	61	t_1	∞
David	4	25	179	80	t_1	∞
Eve	5	43	168	58	t_1	∞
Frank	6	22	181	79	t_1	∞
Greg	7	52	175	67	t_1	∞
Henry	8	17	169	76	t_2	∞
Iris	9	34	158	49	t_2	∞
Jane	10	29	165	59	t_2	∞
Kenneth	11	31	184	94	t_2	∞
Luke	12	13	125	38	t_2	∞

Data inserted at time t_2 and now valid

At t_3:

```
DELETE FROM table WHERE ID = 11;
```

Updates - Example

name	ID	age	height	weight	TS_{from}	TS_{to}
Alice	1	10	120	34	t_1	∞
Bob	2	71	175	74	t_1	∞
Charles	3	37	168	61	t_1	∞
David	4	25	179	80	t_1	∞
Eve	5	43	168	58	t_1	∞
Frank	6	22	181	79	t_1	∞
Greg	7	52	175	67	t_1	∞
Henry	8	17	169	76	t_2	∞
Iris	9	34	158	49	t_2	∞
Jane	10	29	165	59	t_2	∞
Kenneth	11	31	184	94	t_2	t_3
Luke	12	13	125	38	t_2	∞

At t_5:
UPDATE weight=82 FROM table WHERE ID = 8;

At t_3:
DELETE FROM table WHERE ID = 11;

Updates - Example

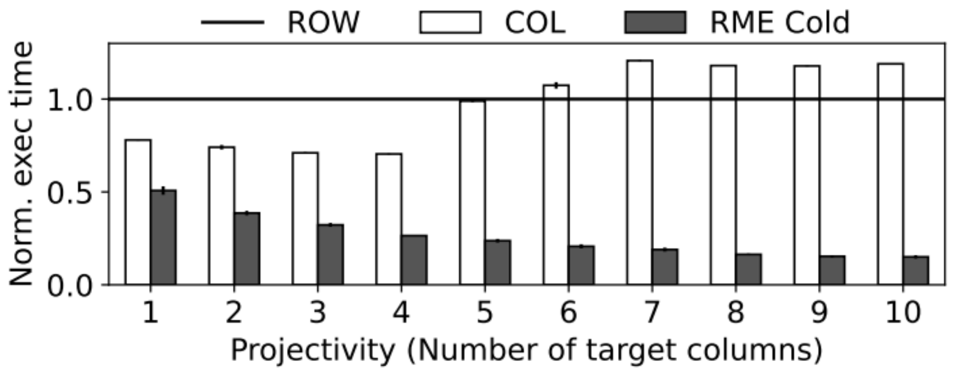
name	ID	age	height	weight	TS_{from}	TS_{to}
Alice	1	10	120	34	t_1	∞
Bob	2	71	175	74	t_1	∞
Charles	3	37	168	61	t_1	∞
David	4	25	179	80	t_1	∞
Eve	5	43	168	58	t_1	∞
Frank	6	22	181	79	t_1	∞
Greg	7	52	175	67	t_1	∞
Henry	8	17	169	76	t_2	t_5
Iris	9	34	158	49	t_2	∞
Jane	10	29	165	59	t_2	∞
Kenneth	11	31	184	94	t_2	t_3
Luke	12	13	125	38	t_2	∞
Henry	**8**	**17**	**169**	**82**	**t_5**	**∞**

At t_5:
UPDATE weight=82 FROM table WHERE ID = 8;

At t_3:
DELETE FROM table WHERE ID = 11;

Updates - Example

name	ID	age	height	weight	TS_{from}	TS_{to}
Alice	1	10	120	34	t_1	∞
Bob	2	71	175	74	t_1	∞
Charles	3	37	168	61	t_1	∞
David	4	25	179	80	t_1	∞
Eve	5	43	168	58	t_1	∞
Frank	6	22	181	79	t_1	∞
Greg	7	52	175	67	t_1	∞
Henry	8	17	169	76	t_2	t_5
Iris	9	34	158	49	t_2	∞
Jane	10	29	165	59	t_2	∞
Kenneth	11	31	184	94	t_2	t_3
Luke	12	13	125	38	t_2	∞
Henry	**8**	**17**	**169**	**82**	**t_5**	**∞**



With MVCC enabled, RME is always faster than ROW and COL!

SELECT avg(weight) FROM table;

At t_3:
cg1 = configure (table, column_group, t_3);

At t_5:
cg1 = configure (table, column_group, t_5);

RME will discard through hardware invalid rows

Implications of Relational Memory


Breaking your (fractured) mirrors …
is not bad luck anymore!

Implications of Relational Memory

Break your (fractured) mirrors!! No need to maintain multiple systems

Layout-less storage: Queries can access any layout for free!

Physical Design is simplified

Query Optimization to find the optimal plan without (layout) constraints

Query Evaluation to use the best layout for each query

Next Steps & Open Questions

Integrate with a **full-blown RDBMS**

Implement **RME within a memory controller** *(ongoing project with RedHat)*

Implement *Transparent Data Transformation* in **Smart SSD**

Transparent Data Transformation for other applications: e.g., ***slice Tensors***

Relational Memory - Summary

Relational Memory: Native In-Memory Accesses on Rows and Columns, EDBT 2023

a novel **SW/HW co-design** paradigm

every query **always access the optimal data layout**

ephemeral variables: a simple and lightweight abstraction

room for ***a lot of*** innovation






Thank you!

disc.bu.edu/relational-memory

disc.bu.edu

Red Hat CISCO NSF