18-344 Recitation 8

11/07/2025

Outline

1. Logistics
2. Homeworks
3. Review
4. Lab 3

Logistics

- Homework 7 released, deadline extended: November 10 (Monday) (in 3 days)
  - Covers lectures 15-16
- Homework 8 released, due on November 17 (Monday) (in 10 days)
  - Covers lecture 17

- Lab 3 deadlines extended as well
  - Code freeze: October 21 (Friday) (in two weeks)
    - Submit (push a commit to your repo) on GitHub Classroom
  - Report: November 24 (Monday) (in two weeks + three days)
    - Submit on Gradescope
- Lab 4 will still be released on November 17
  - Code freeze: December 11
    - Submit (push a commit to your repo) on GitHub Classroom
  - Report: December 14
    - Submit on Gradescope

Homework 7 – in case you missed it

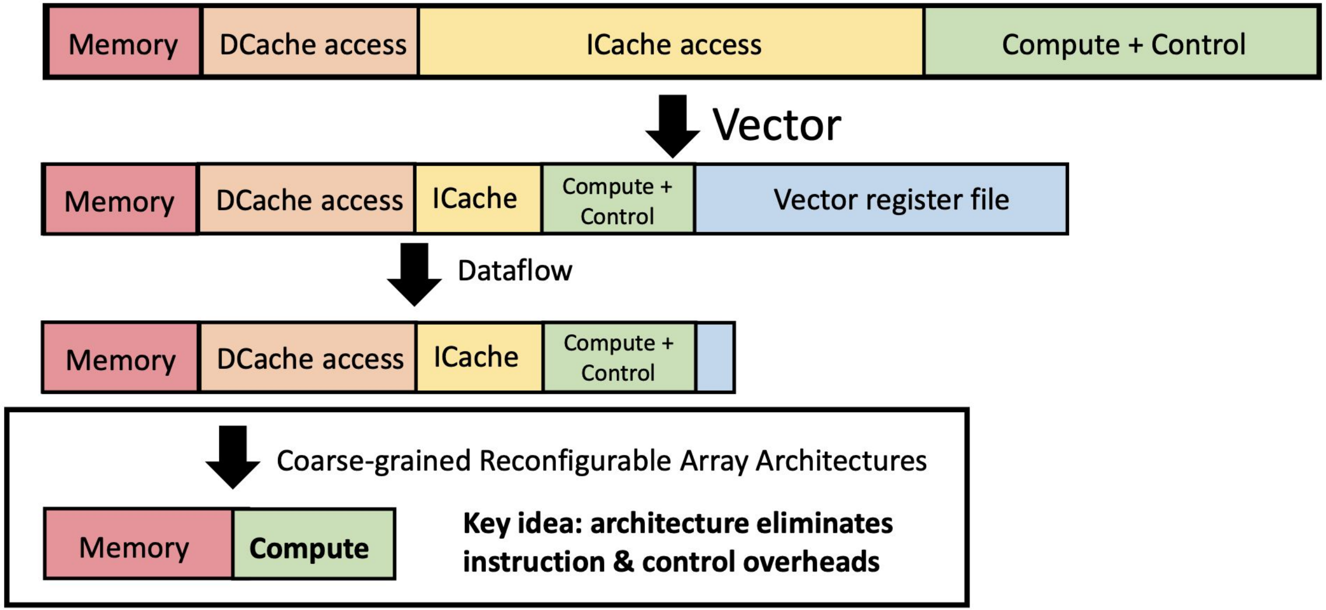
How to calculate the average ***ILP*** given a sequence of instructions?

For this assignment, we define: $$ILP = \frac{T_1}{T_\infty}$$

where T_n is the number of cycles to execute the sequence of instructions with n execution units.

Review

Scalar/Vector/(Spatial) Dataflow architectures



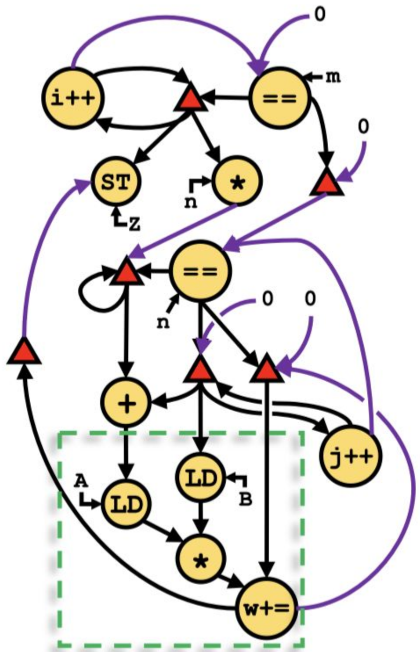
[1] Brandon Lucia. 2024. Energy-minimal Computing: Edge architectures for extreme efficiency. CMU 18-344.

Spatial dataflow for programmable acceleration

Dense matrix-vector multiplication:

Pseudocode and dataflow-graph (DFG)[1]

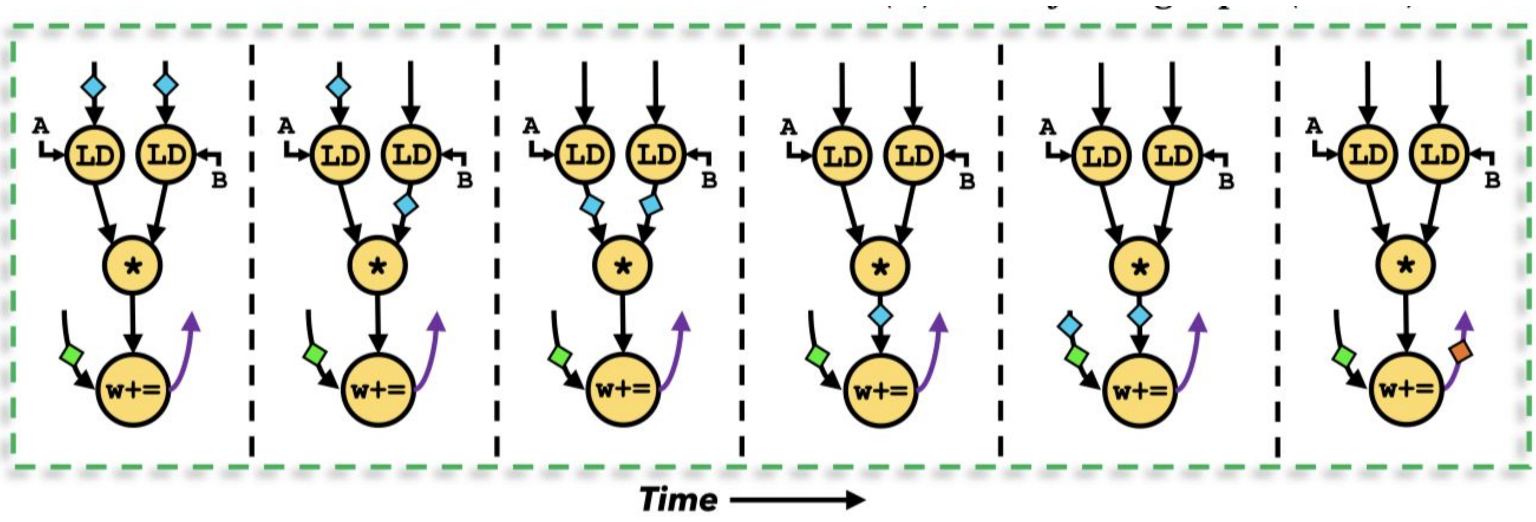
```
def dmv (A, B, Z, m, n):
  for i = 0..m:
    w = 0
    for j = 0..n:
      w += A[i,j] * B[j]
  Z[i] = w
```



[1] Nikhil Agarwal et al. 2024. The TYR Dataflow Architecture: Improving Locality by Taming Parallelism. MICRO 2024.

Dataflow in action

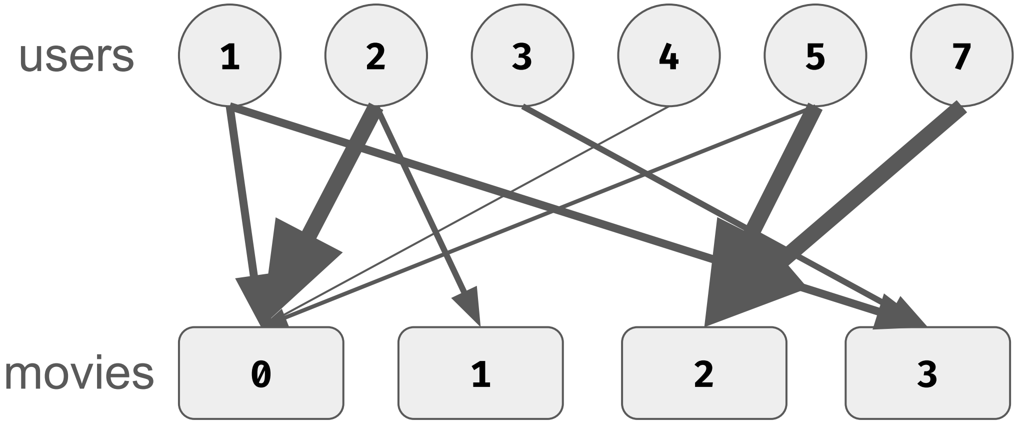
Dense matrix-vector multiplication's inner loop: a running example[1]



[1] Nikhil Agarwal et al. 2024. The TYR Dataflow Architecture: Improving Locality by Taming Parallelism. MICRO 2024.

Sparse problems

Machine learning, networking, and many more problems all involve ***sparse data***.



users

	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7
m_0	0	4	7	0	1	2	0	0
m_1	0	0	3	0	0	0	0	0
m_2	0	0	0	0	0	8	0	9
m_3	0	5	0	3	0	0	0	0

movies

Compressed Sparse Columns/Rows – **CSC/CSR**

Matrices/tensors in sparse problems have ***high memory footprint*** yet ***low density***.
=> Convert and store data in more space-efficient formats (e.g. **CSC** and **CSR**).

How to convert a matrix/edge list to **CSR**?

users

movies	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7
m_0		4	7		1	2		
m_1			3					
m_2						8		9
m_3		5		3				

Edge list (may ***not*** always be sorted)
(also called COO, coordinate list):

```
2 → 0 : 7
2 → 1 : 3
1 → 0 : 4
1 → 3 : 5
3 → 3 : 3
4 → 0 : 1
7 → 2 : 9
5 → 0 : 2
5 → 2 : 8
```

EL to **CSR**: 1/ Sort **EL** by rows (`user`)

```
user    →  movie  :   weight

2       →    0    :      7
2       →    1    :      3

1       →    0    :      4
1       →    3    :      5

3       →    3    :      3

4       →    0    :      1

7       →    2    :      9

5       →    0    :      2
5       →    2    :      8
```

EL to **CS<u>R</u>**: 1/ Sort **EL** by <u>rows (`user`)</u>

user	→	movie	:	weight
2	→	0	:	7
2	→	1	:	3
1	→	0	:	4
1	→	3	:	5
3	→	3	:	3
4	→	0	:	1
7	→	2	:	9
5	→	0	:	2
5	→	2	:	8

EL to **CSR**: 1/4 Sort **EL** by rows (`user`)

```
user   →  movie  :  weight

1      →    0    :     4
1      →    3    :     5

2      →    0    :     7
2      →    1    :     3

3      →    3    :     3

4      →    0    :     1

5      →    0    :     2
5      →    2    :     8

7      →    2    :     9
```

EL to **CSR**: 2/4 Count neighbors and build offset array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Neighbor count for each `user`

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7
0	2	2	1	1	2	0	1

Start from **0**

0

Offset Array (**OA**)

EL to **CSR**: 2/4 Count neighbors and build offset array

```
user  →  movie  :  weight

1     →    0    :    4
1     →    3    :    5

2     →    0    :    7
2     →    1    :    3

3     →    3    :    3

4     →    0    :    1

5     →    0    :    2
5     →    2    :    8

7     →    2    :    9
```

Neighbor count for each `user`

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7
0	2	2	1	1	2	0	1

No edge starting from u_0,

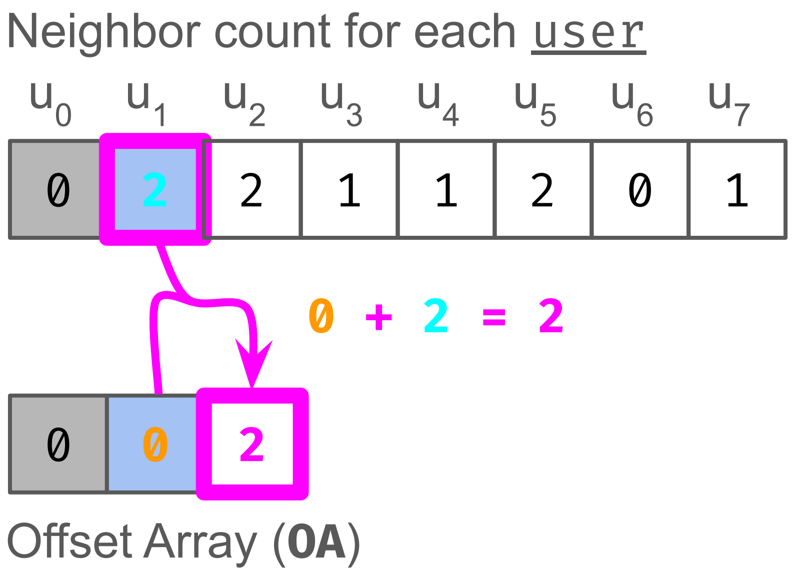
`0 + 0 = 0`

0	0

Offset Array (**OA**)

EL to **CSR**: 2/4 Count neighbors and build offset array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9



EL to **CSR**: 2/4 Count neighbors and build offset array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Neighbor count for each `user`

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7
0	2	2	1	1	2	0	1

2 + 2 = 4

0	0	2	4

Offset Array (**OA**)

EL to **CSR**: 2/4 Count neighbors and build offset array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Neighbor count for each `user`

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7
0	2	2	1	1	2	0	1

4 + 1 = 5

0	0	2	4	5

Offset Array (**OA**)

EL to **CSR**: 2/4 Count neighbors and build offset array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Neighbor count for each `user`

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7
0	2	2	1	1	2	0	1

5 + 1 = 6

0	0	2	4	5	6

Offset Array (**OA**)

EL to **CSR**: 2/4 Count neighbors and build offset array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Neighbor count for each user

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7
0	2	2	1	1	$\boxed{2}$	0	1

6 + 2 = 8

0	0	2	4	5	6	$\boxed{8}$

Offset Array (**OA**)

EL to **CSR**: 2/4 Count neighbors and build offset array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Neighbor count for each `user`

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7
0	2	2	1	1	2	0	1

No edge starting from u_6,

8 + 0 = 8

0	0	2	4	5	6	8	8

Offset Array (**OA**)

EL to **CSR**: 2/4 Count neighbors and build offset array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Neighbor count for each user

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7
0	2	2	1	1	2	0	$\boxed{1}$

8 + 1 = 9

0	0	2	4	5	6	8	8	$\boxed{9}$

Offset Array (**OA**)

EL to **CSR**: 2/4 Count neighbors and build offset array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Offset Array (**OA**)

0	0	2	4	5	6	8	8	9

The last element in **OA** always equals to **`size(edges)`**

EL to **CSR**: 3/4 Build neighbor array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Offset Array (**OA**)

0	0	2	4	5	6	8	8	9

For each non-zero-length interval in the **OA**, append its edges' destinations (**movies**) to the **NA** *in-order*.

Neighbor Array (**NA**)

EL to **CSR**: 3/4 Build neighbor array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Offset Array (**OA**)

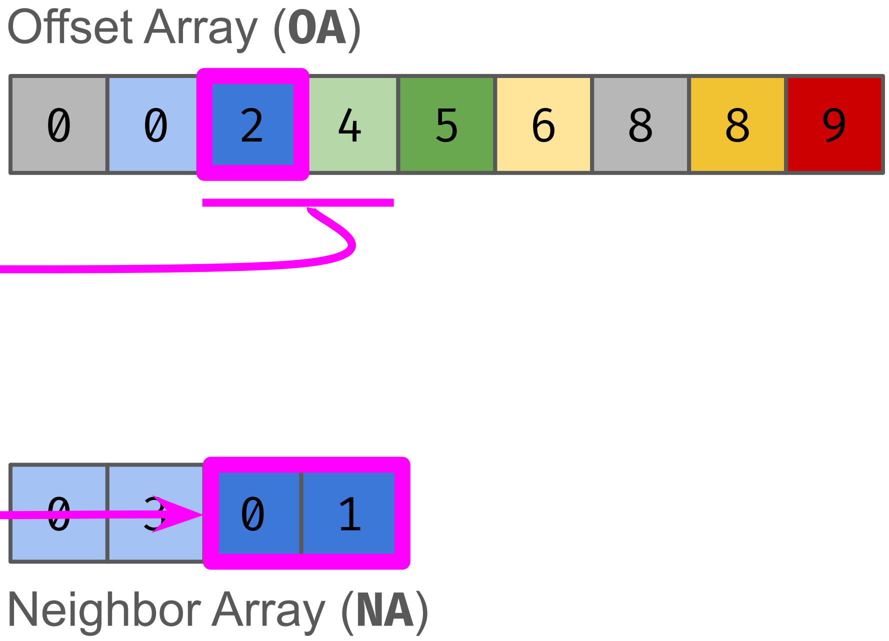
0	0	2	4	5	6	8	8	9

0	3

Neighbor Array (**NA**)

EL to **CSR**: 3/4 Build neighbor array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9



EL to **CSR**: 3/4 Build neighbor array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Offset Array (**OA**)

0	0	2	4	5	6	8	8	9

0	3	0	1	3

Neighbor Array (**NA**)

EL to **CSR**: 3/4 Build neighbor array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Offset Array (**OA**)

0	0	2	4	5	6	8	8	9

0	3	0	1	3	0

Neighbor Array (**NA**)

EL to **CSR**: 3/4 Build neighbor array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	$\boxed{0}$	:	2
5	→	$\boxed{2}$	:	8
7	→	2	:	9

Offset Array (**OA**)

0	0	2	4	5	$\boxed{6}$	8	8	9

0	3	0	1	3	0	$\boxed{0}$	$\boxed{2}$

Neighbor Array (**NA**)

EL to **CSR**: 3/4 Build neighbor array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Offset Array (**OA**)

0	0	2	4	5	6	8	8	9

0	3	0	1	3	0	0	2	2

Neighbor Array (**NA**)

EL to **CSR**: 3/4 Build neighbor array

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Offset Array (**OA**)

0	0	2	4	5	6	8	8	9

Neighbor Array (**NA**)

0	3	0	1	3	0	0	2	2

EL to **CSR**: 4/4 Build the property array (if needed)

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Offset Array (**OA**)

0	0	2	4	5	6	8	8	9

Neighbor Array (**NA**)

0	3	0	1	3	0	0	2	2

Property Array (**PA**)

4	5	7	3	3	1	2	8	9

EL to **CSR**: done!

user	→	movie	:	weight
1	→	0	:	4
1	→	3	:	5
2	→	0	:	7
2	→	1	:	3
3	→	3	:	3
4	→	0	:	1
5	→	0	:	2
5	→	2	:	8
7	→	2	:	9

Offset Array (**OA**) (`indptr`)

0	0	2	4	5	6	8	8	9

Neighbor Array (**NA**) (`indices`)

0	3	0	1	3	0	0	2	2

Property Array (**PA**) (`data`)

4	5	7	3	3	1	2	8	9

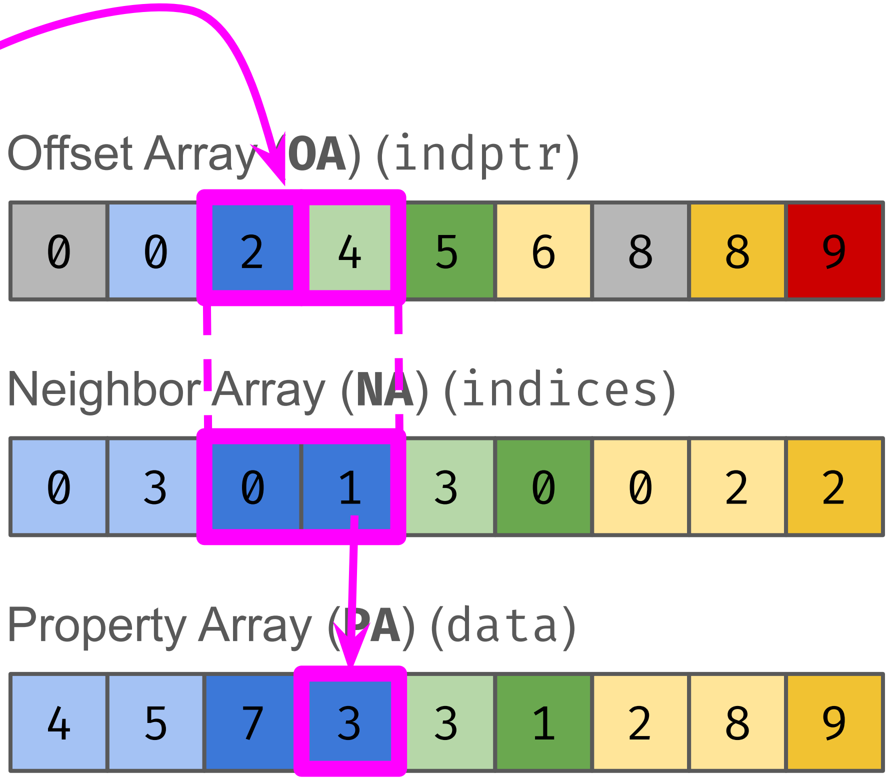
How to use the CSR?

Say, we are interested in user $\mathbf{u_2}$'s rating for movie $\mathbf{m_1}$.

First, we find the offset for user $\mathbf{u_2}$ by indexing **`OA[2] and OA[2+1]`**.

Next, we iterate through **NA** with range **`OA[2] … OA[2+1]-1`** (inclusive) and check for a match with movie $\mathbf{m_1}$'s index, **`1`**.

If match (e.g. at **`NA[i]`**), we then index **`PA[i]`** to find user $\mathbf{u_2}$'s rating.



EL (COO), **CSC**, and **CSR**

You should now be able to convert **EL** to **CSR**.
Similarly, you should be able to convert **EL** to **CSC**.

By traversing the **CSC**/**CSR** (as seen on last slide), you should be able to convert **CSC** and **CSR** to **EL** as well.

This is what you'll be implementing in the next lab!
(Without the property array.)
More to follow in upcoming weeks' recitations.

Lab 3

Lab 3 – in case you missed it

There's a little pintool that we distribute called **`vm_trace`**.

This pintool will be very useful for you to test and profile your VM implementation on real-world workloads.

It is not required that you use this pintool, however ***we highly recommend you use it***. (You'll need to *somehow* generate sufficiently complex traces for testing anyways. Why not go with the easier route?)

`vm_trace` is ***NOT*** included in the regular lab3 tar ball.
It is available separately at:
`/afs/ece.cmu.edu/class/ece344/assign/vm_trace.tar.gz`