

# Column Block for Parallel Learning

- Data in each block is stored in the compressed column (CSC) with each column sorted by the corresponding feature value
- => Row Orientation vs Column Orientation

Employee_ID	Job	Dept	City
1	Shipping	Operations	Toronto
2	Receiving	Operations	Toronto
3	Accounting	Finance	Boston

Data stored in rows

1	Shipping	Operations	Toronto
2	Receiving	Operations	Toronto
3	Accounting	Finance	Boston

Data stored in columns

1	Shipping	Operations	Toronto
2	Receiving	Operations	Toronto
3	Accounting	Finance	Boston

<https://www.youtube.com/watch?v=doQfVHGfFJA>

개념은 간단합니다.

보통 모든 정보를 얻기 위하여 row형태로 데이터가 저장되어있는 것을 column 기준으로 저장하는 것입니다.

Row ID	League	PLAYER	Position	Team	Games	Home Run
1	American	Aubrey, Michael	1B	BAL	31	4
2	American	Wigginton, Ty	1B/3B/DH	BAL	122	11
3	American	Roberts, Brian	2B	BAL	159	16
4	American	Turner, Justin	3B	BAL	12	0
5	American	Atkins, Garrett	3B/1B	BAL	126	9
6	American	Moeller, Chad	C	BAL	30	2
7	American	Rodriguez, Guillermo	C	BAL	7	0
8	American	Tatum, Craig	C	BAL	26	1
9	American	Wieters, Matt	C	BAL	96	9
10	American	Scott, Luke	DH/OF	BAL	28	25
11	American	Fiorentino, Jeff	OF	BAL	24	0
12	American	Jones, Adam	OF	BAL	12	1
13	American	Markakis, Nick	OF	BAL	16	1
14	American	Montanez, Luis	OF	BAL	29	1
15	American	Pie, Felix	OF	BAL	7	0
16	American	Reimold, Nolan	OF	BAL	10	1
17	American	Andino, Robert	SS	BAL	78	2
18	American	Izturis, Cesar	SS	BAL	14	2
19	American	Tejada, Miguel	SS	BAL	59	14
20	American	Bailey, Jeff	1B	BOS	26	3
21	American	Bates, Aaron	1B	BOS	5	0

1,American,"Aubrey,Michael",1B,BAL,31,4,2,American,"Wigginton,Ty",1B/3B/DH,BAL,122,11,3,American,"Roberts,Brian",2B,BAL,159,16,4,American,"Turner,Justin",3B,BAL,12,0,5,American,"Atkins,Garrett",3B/1B,BAL,126,9,6,American,"Moeller,Chad",C,BAL,30,2,7,American,"Rodriguez,Guillermo",C,BAL,7,0,8,American,"Tatum,Craig",C,BAL,26,1,9,American,"Wieters,Matt",C,BAL,96,9,10,American,"Scott,Luke",DH/OF,BAL,28,25,11,American,"Fiorentino,Jeff",OF,BAL,24,0,12,American,"Jones,Adam",OF,BAL,12,1,13,American,"Markakis,Nick",OF,BAL,16,1,14,American,"Montanez,Luis",OF,BAL,29,1,15,American,"Pie,Felix",OF,BAL,7,0,16,American,"Reimold,Nolan",OF,BAL,10,1,17,American,"Andino,Robert",SS,BAL,78,2,18,American,"Izturis,Cesar",SS,BAL,14,2,19,American,"Tejada,Miguel",SS,BAL,59,14,20,American,"Bailey,Jeff",1B,BOS,26,3,21,American,"Bates,Aaron",1B,BOS,5,0,22,American,"Youkilis",1B,BOS,5,0

<https://www.youtube.com/watch?v=8KGVFB3kVHQ>

Row 형태로 저장되어 있는 데이터를 특정 column에 대해 filtering이나 indexing을 하는 경우 전체 데이터를 순차적으로 읽어서 각 column에 어떤 정보가 있는지 찾아야 합니다.





Row ID	League	PLAYER	Position	Team	Games	Home Run
22	American	Youkilis, Kevin	1B/3B	BOS	136	27
23	American	Hulett, Tug	2B	BOS	15	0
24	American	Pedroia, Dustin	2B	BOS	154	15
25	American	Beltre, Adrian	3B	BOS	111	8
245	American	Buck, John	C	TOR	51	2
246	American	Chavez, Raul	C	TOR	5	0
247	American	Phillips, Kyle	C	TOR	22	0
248	American	Dellucci, David	DH	TOR	33	10
249	American	Ruiz, Randy	DH	TOR	51	35
250	American	Lind, Adam			7	0
251	American	Gathright, Joey			0	0
252	American	Padilla, Jorge			0	0
253	American	Reed, Jeremy			26	0
254	American	Snider, Travis			77	9
255	American	Wells, Vernon			58	15
256	American	Bautista, Jose			13	13
257	American	Gonzalez, Alex			12	8
258	American	McDonald, John			73	4
259	National	Canizares, Barbaro			5	0
260	National	Conrad, Brooks			30	2

Row ID 1-258,  
Leagues, Teams



특히 sparse matrix나 컬럼이 class를 나타내는 경우는 동일한 값이 반복되어 단순하게 압축을 할 수 있는 장점이 있습니다.

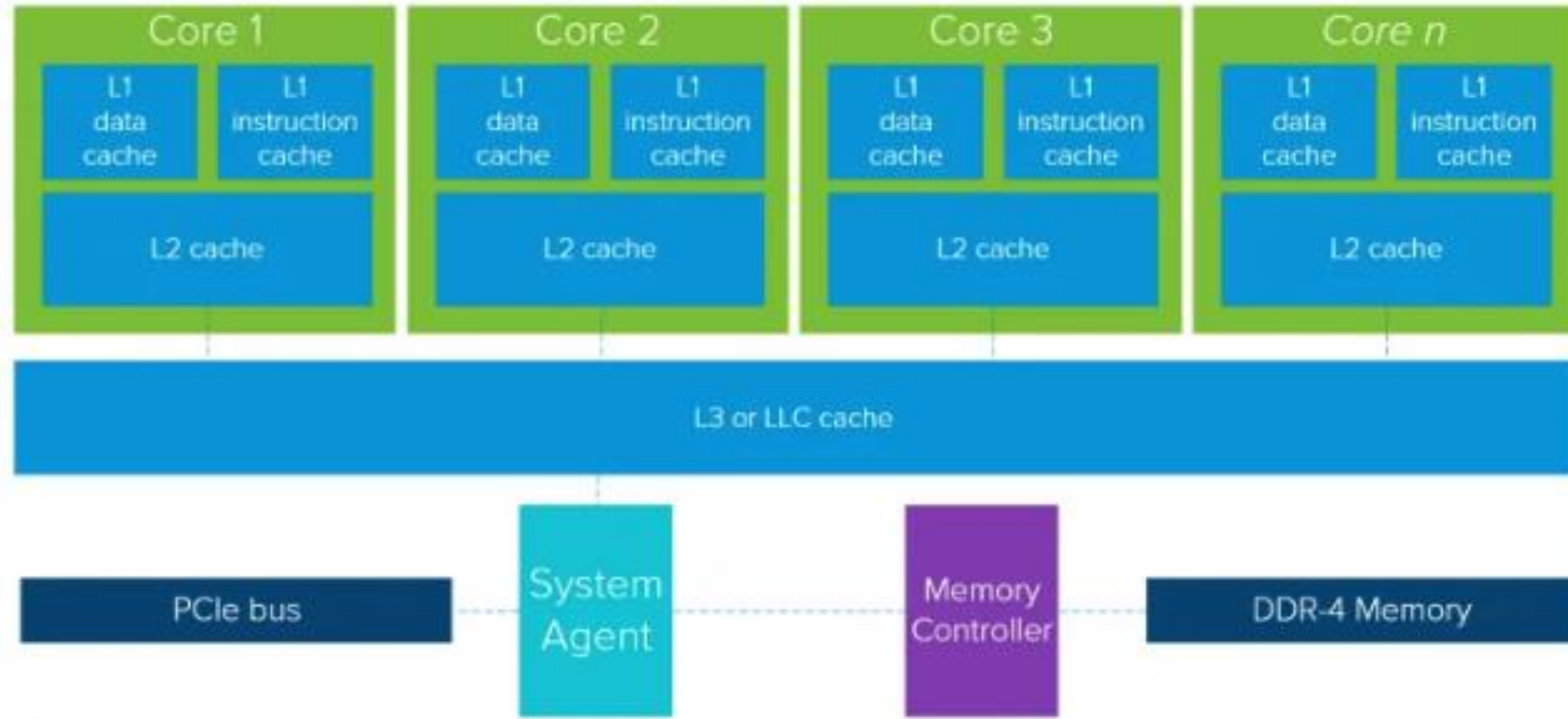
우리는 Split을 할 때 column 형태의 데이터를 사용하게 되므로  
기존 row형태의 데이터 저장보다 column형태로 데이터를 저장하는 경우 더 빠른 속도를 얻을 수 있습니다.

데이터를 저장하고 검색하는 것을 빠르게 할 수도 있고  
대용량의 데이터가 있는 경우 column을 기준으로 정렬하여 block에 넣어놓는다면  
split이후에도 한번에 봐야할 데이터는 하나의 block에 있게 되어  
Block을 읽는 I/O도 줄일 수 있습니다.

# Cache-aware access

- For the exact greedy algorithm, we can alleviate the problem by a cache-aware prefetching algorithm
- I/O Speed : Cpu cache > memory > disk
  - 데이터의 접근 속도는 CPU Cache가 가장 빠르기에 최대한 오랫동안 cache의 내용을 활용하도록 만든다면 performance를 올릴 수 있습니다.
- 데이터를 저장할 때 Block 단위로 저장하고 split된 내용은 block 안에서 다시 읽게 만듭니다.

## CPU



<https://nielshagoort.com/2019/03/12/exploring-the-gpu-architecture/>

CPU의 cache는 core와 가장 가까이 붙어있는 L1 cache부터 L3 cache까지 존재하고 Memory Controller를 사용하여 접근할 수 있는 DDR-x Memory가 그 다음으로 빠르며 SSD라고 하더라도 Disk는 이보다 훨씬 느린 속도로 접근을 합니다.

- CPU의 Cache에 데이터가 잘 저장되고 금방 재사용하도록 알고리즘을 만든다면 오버헤드 없이 빠른 처리가 가능합니다.
- cache aware access란 cache에 어떤 데이터가 들어있는지 잘 알고 있으며 이를 다시 재활용할 수 있도록 알고리즘을 작성하는 것을 의미합니다.

# Blocks for out-of-core computation

- Block Compression

- Block형태로 데이터를 나누어 저장하고 column orientation으로 되어 있는 데이터를 압축하여 저장하는 것입니다. 대략  $2^{16}$ 개의 데이터를 하나의 block에 넣는 경우 26%~29%의 압축 효과를 볼 수 있었습니다.
- 또 하나의 key point는 읽어가면서 압축을 푸는 것입니다.
- 압축을 풀어서 저장을 하는 경우보다 적게 데이터를 읽어가면서 ( 요즘은 CPU가 빠르기에 ) 읽어가면서 압축을 풀어서 CPU내에서 연산을 할 때에는 압축이 풀린 상태로 사용하게 하는 것입니다.
- 최근 Disk의 속도가 상대적으로 느린 것을 보완하기 위하여 disk에는 압축을 하고 읽어갈 때 ( on-the-fly ) 압축을 풀어서 읽는 스토리지도 저장방식도 많이 있습니다.
- CPU를 조금 더 사용하기는 하지만 disk I/O가 훨씬 더 overhead가 크기에 충분히 의미가 있습니다.



# Blocks for out-of-core computation

- Block Sharding

- Disk가 여러 개 있는 경우 여러 개의 Disk에 Block 데이터를 나누어 놓습니다.
- 데이터를 읽어올 때에는 여러 Disk를 동시에 활용하여 읽을 수 있고 Disk I/O가 느린 것을 병렬처리를 통하여 극복하는 방법입니다.

- 우측의 그림은 RAID라는 기술입니다. 나누어 놓는 기술은 아니지만 여러 디스크를 활용하는 것에 대한 장점을 볼 수 있습니다.

왼쪽 디스크에서만 1Gigabyte의 데이터를 읽어오는 것에 비하여  
두개의 디스크를 동시에 활용하여 총 500Mbyte씩 총 1Gigabyte의  
데이터를 읽어오게 만든다면  
물리적 장치의 속도를 두배 향상시키는 효과를 얻을 수 있습니다.

## RAID 1

(2N개의 저장장치를 필요로 합니다.)

[Mirroring] - 두개의 하드디스크에 똑같은 데이터를

