Week3 Homework
Practice 1 & 3

2022. 7. 19

Suhwan Sin

E-Mail: tlstnghks77@dankook.ac.kr

Content

- Practice 1
  - Meta operations
  - Option (fillseq, fillrandom)
  - Compare stats
  - Q1
  - Q2
  - Q3

- Practice 3
  - Batch processing
  - Conclusion

Practice 1

[A] $ ./db_bench --benchmarks="fillseq"

[B] $ ./db_bench --benchmarks="fillrandom"

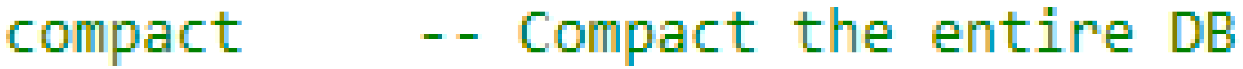
Q1. Compare throughput, latency, and stats of two benchmarks and explain why.

Q2. In benchmark A, SSTs are not written in L0. Why?

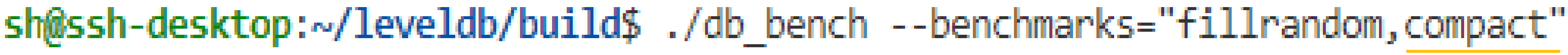
Q3. Calculate SAF (Space Amplification Factor) for each benchmark.

Meta operations

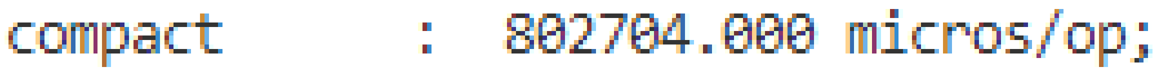
# compact



```
compact      -- Compact the entire DB
```

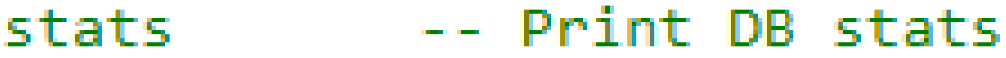


```
sh@ssh-desktop:~/leveldb/build$ ./db_bench --benchmarks="fillrandom,compact"
```

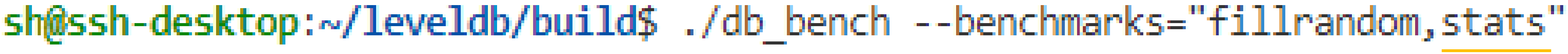


```
compact      :  802704.000 micros/op;
```

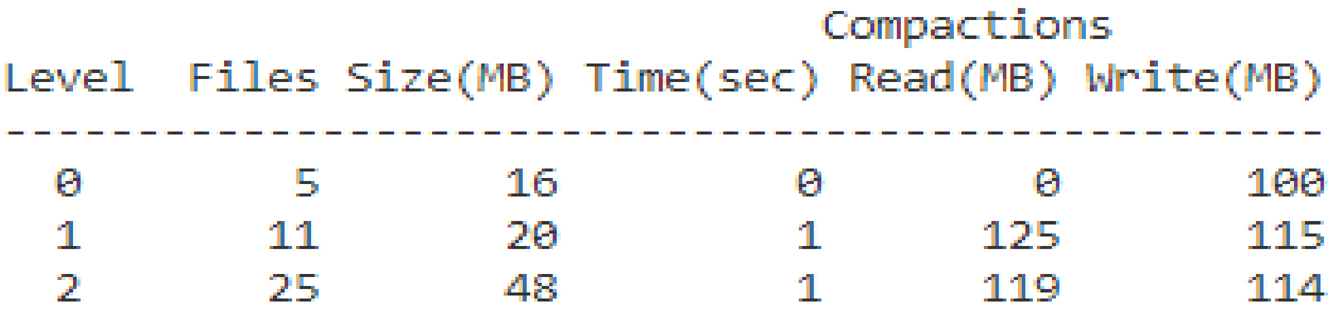
# stats



```
stats        -- Print DB stats
```



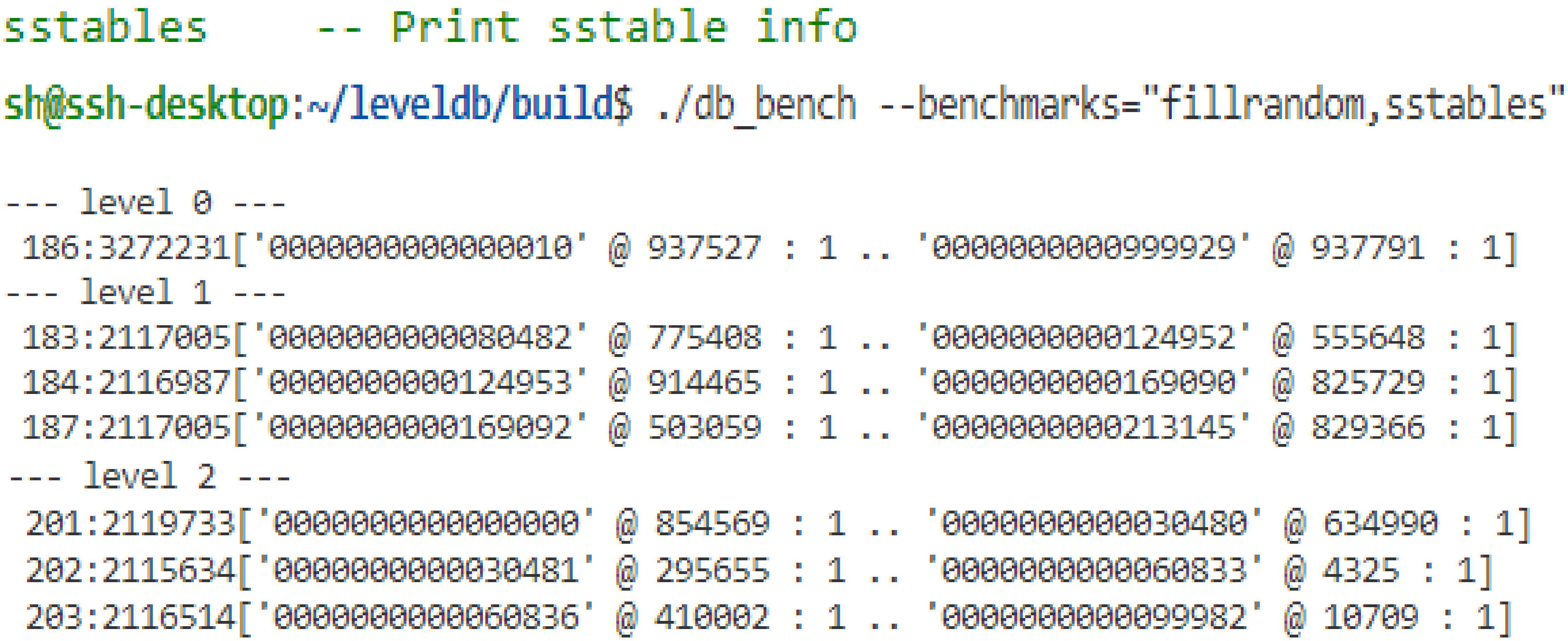
```
sh@ssh-desktop:~/leveldb/build$ ./db_bench --benchmarks="fillrandom,stats"
```



```
                              Compactions
Level  Files Size(MB) Time(sec) Read(MB) Write(MB)
--------------------------------------------------
  0        5      16         0        0      100
  1       11      20         1      125      115
  2       25      48         1      119      114
```

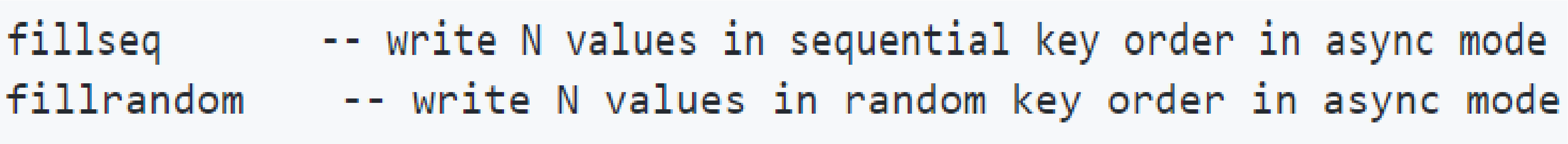
Meta operations

# sstables



```
sstables     -- Print sstable info

sh@ssh-desktop:~/leveldb/build$ ./db_bench --benchmarks="fillrandom,sstables"

--- level 0 ---
 186:3272231['0000000000000010' @ 937527 : 1 .. '0000000000999929' @ 937791 : 1]
--- level 1 ---
 183:2117005['0000000000080482' @ 775408 : 1 .. '0000000000124952' @ 555648 : 1]
 184:2116987['0000000000124953' @ 914465 : 1 .. '0000000000169090' @ 825729 : 1]
 187:2117005['0000000000169092' @ 503059 : 1 .. '0000000000213145' @ 829366 : 1]
--- level 2 ---
 201:2119733['0000000000000000' @ 854569 : 1 .. '0000000000030480' @ 634990 : 1]
 202:2115634['0000000000030481' @ 295655 : 1 .. '0000000000060833' @ 4325 : 1]
 203:2116514['0000000000060836' @ 410002 : 1 .. '0000000000099982' @ 10709 : 1]
```

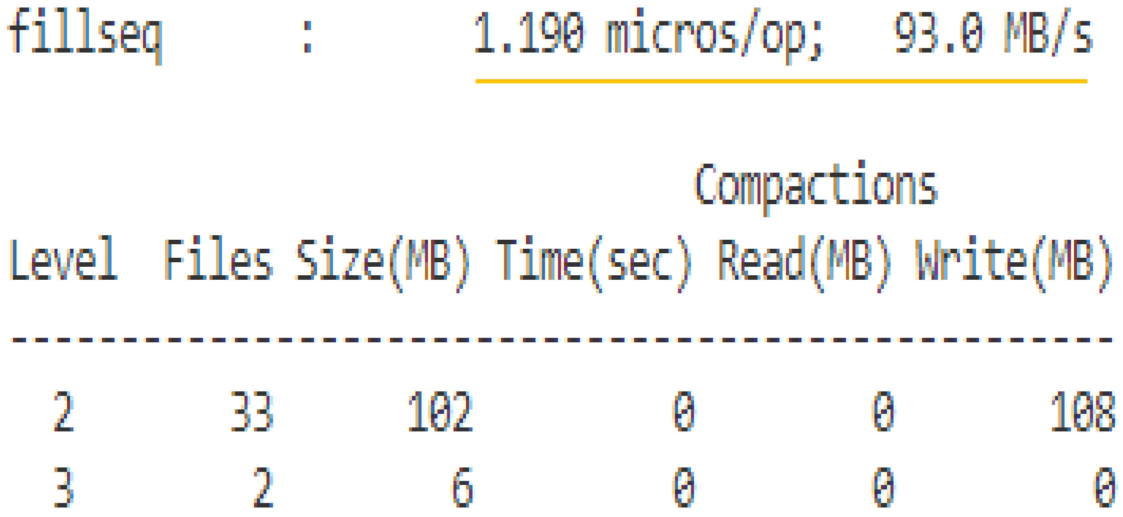
Option – “fillseq & fillrandom”



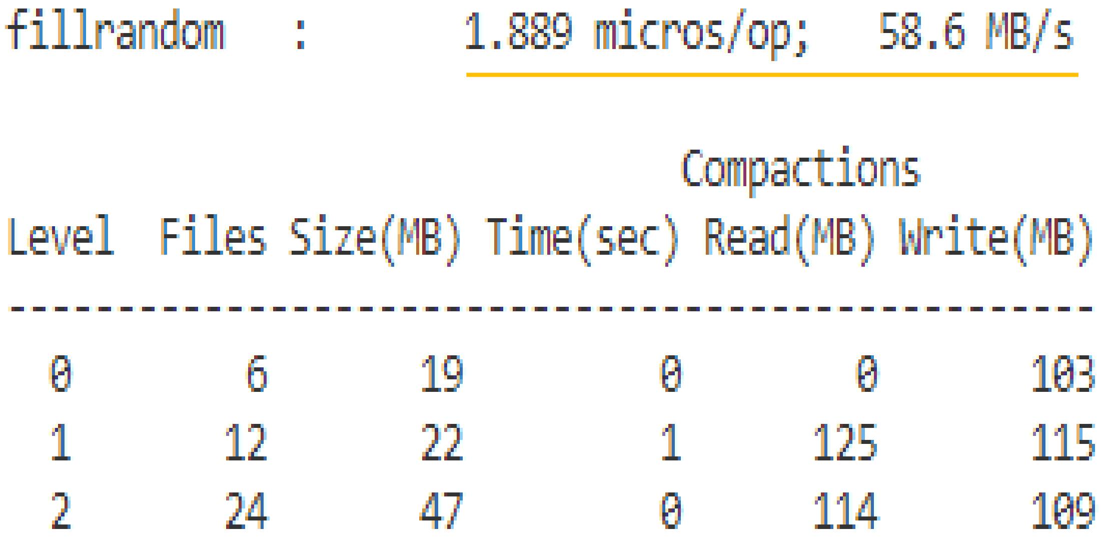
```
fillseq       -- write N values in sequential key order in async mode
fillrandom     -- write N values in random key order in async mode
```

	Key range 중복	Compaction	Example
fillseq	x	x	123 456 789
fillrandom	o	o	372 543 111

Compare Seq / Random



```
fillseq      :       1.190 micros/op;    93.0 MB/s

                               Compactions
Level  Files Size(MB) Time(sec) Read(MB) Write(MB)
--------------------------------------------------
  2       33     102         0        0       108
  3        2       6         0        0         0
```



```
fillrandom   :       1.889 micros/op;    58.6 MB/s

                               Compactions
Level  Files Size(MB) Time(sec) Read(MB) Write(MB)
--------------------------------------------------
  0        6      19         0        0       103
  1       12      22         1      125       115
  2       24      47         0      114       109
```

Q1.

Q1. Compare throughput, latency, and stats of two benchmarks and explain why.

A.

	throughput	latency
fillseq	⬆	⬆
fillrandom	⬇	⬇

Q2.

Q2. In benchmark A, SSTs are not written in L0. Why?

A. Trivial move

LevelDB One kind of optimization is when the following conditions are met

- There is only one file in the layer
- Layer files and level+1, Layer files do not overlap
- Layer files and level+2, The file size of the file overlapping part of the layer does not exceed the threshold

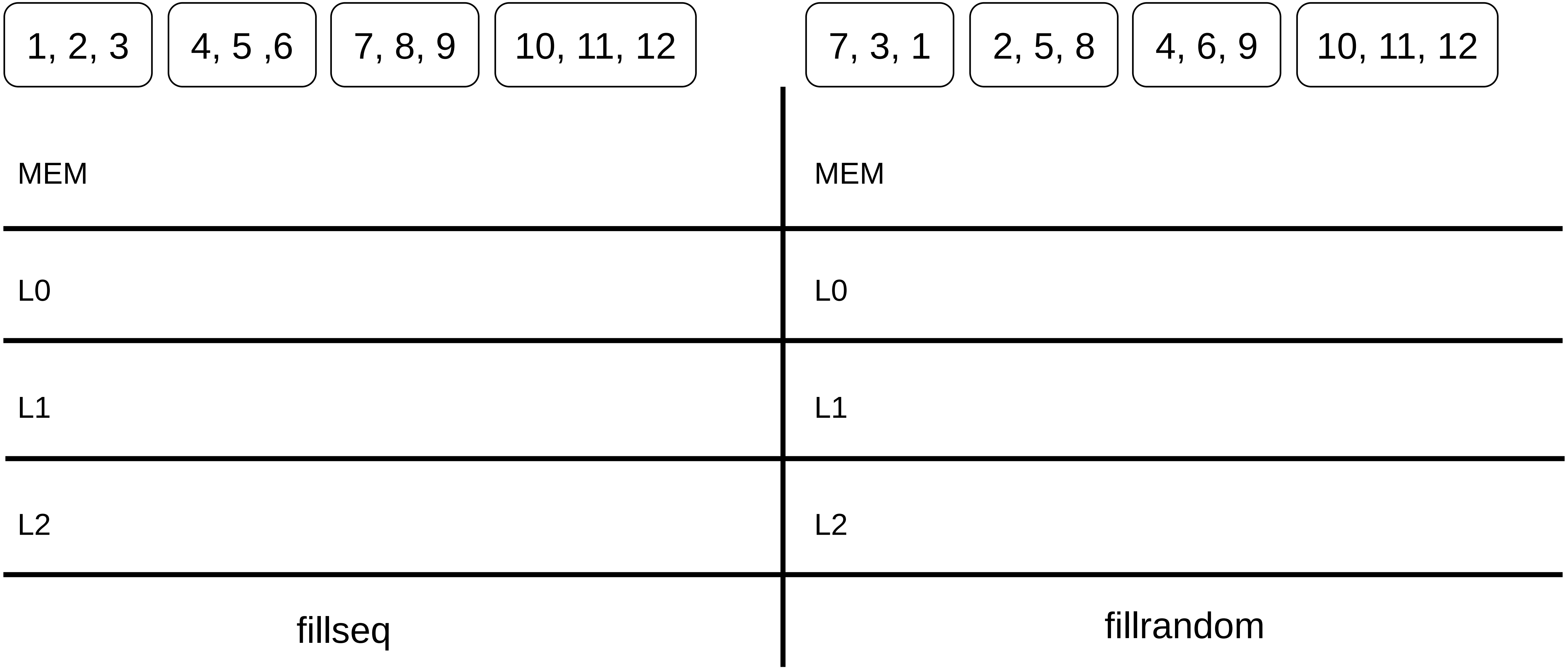
The file of the layer is moved to level+1 Delamination directly

Q2. Trivial move

- **Fillseq**
  - L2, L2, L2, L2 … (Compaction) L3

- **Fillrandom**
  - L2, L1, L0, L0 (Compaction) … L0

Q2. Trivial move



1, 2, 3

4, 5 ,6

7, 8, 9

10, 11, 12

7, 3, 1

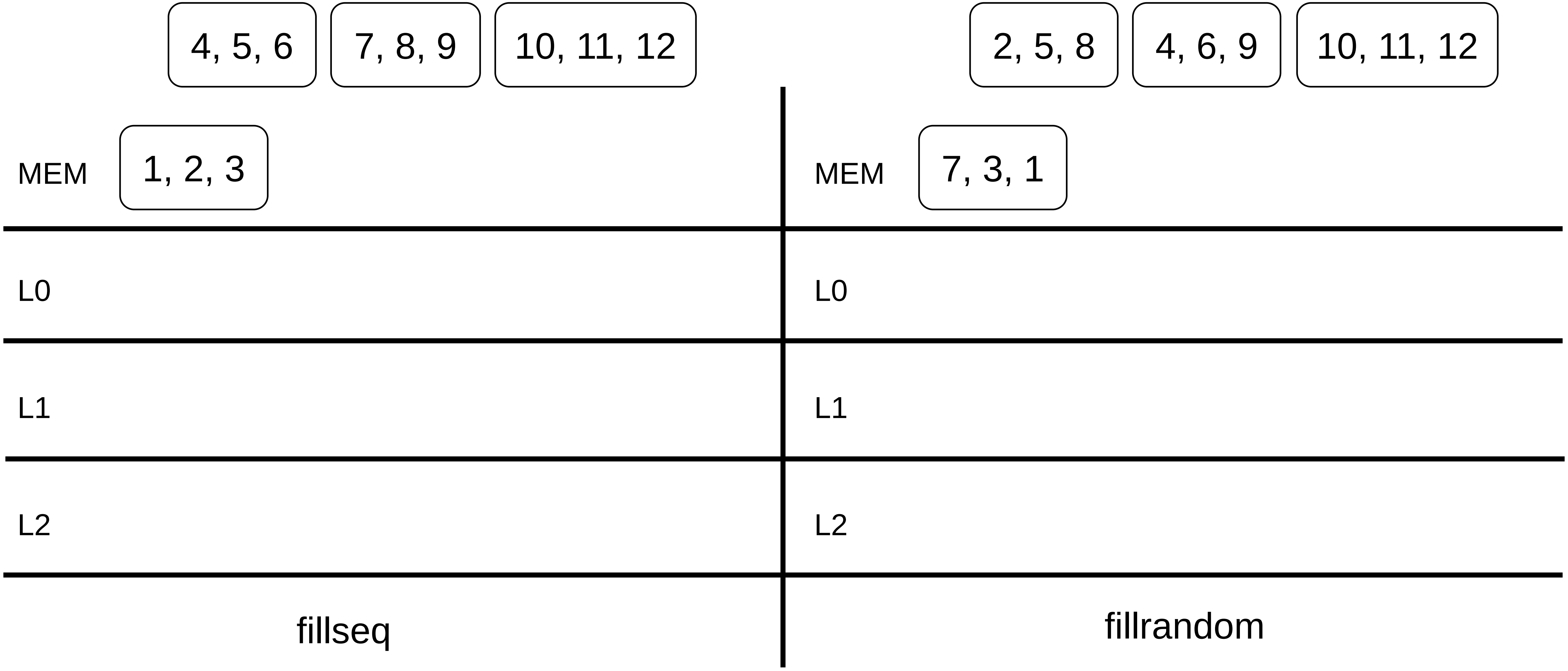
2, 5, 8

4, 6, 9

10, 11, 12

	fillseq	fillrandom
MEM		
L0		
L1		
L2		

Q2. Trivial move



4, 5, 6

7, 8, 9

10, 11, 12

MEM

1, 2, 3

L0

L1

L2

fillseq

2, 5, 8

4, 6, 9

10, 11, 12

MEM

7, 3, 1

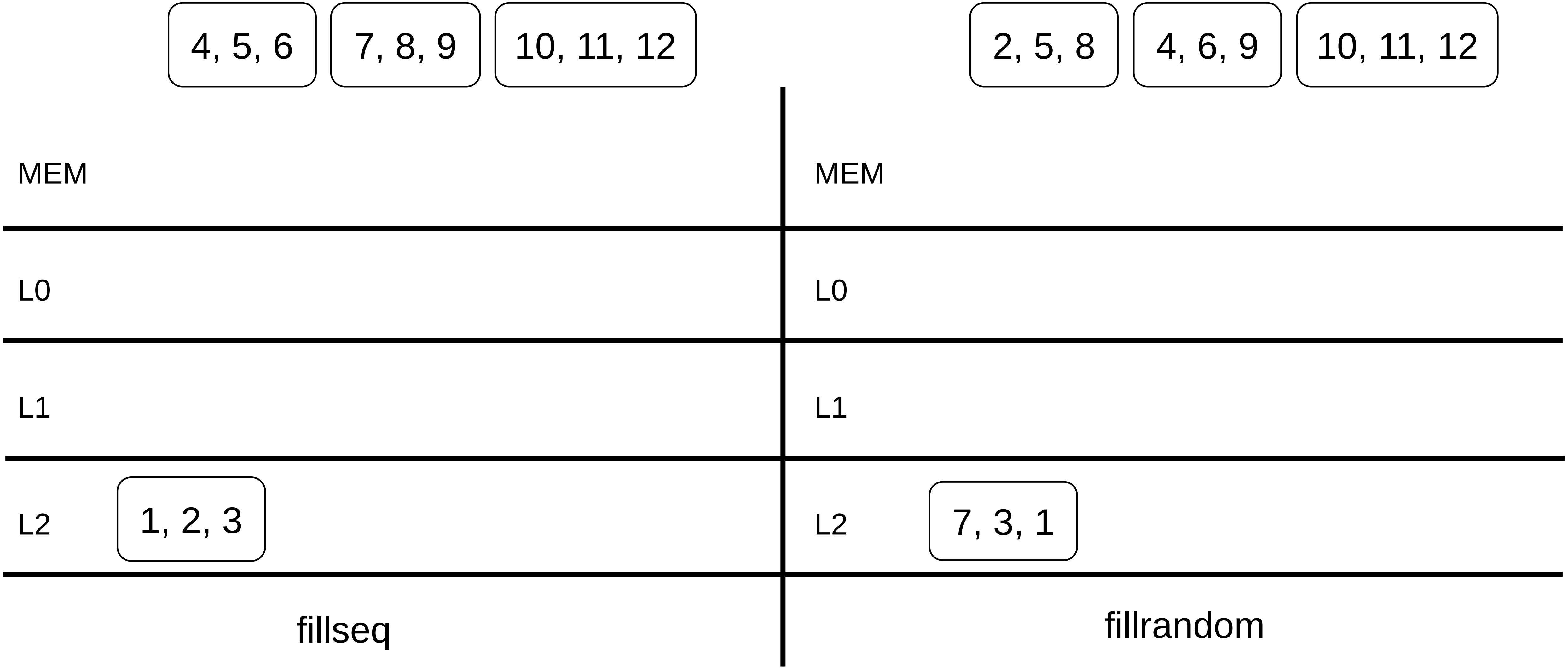
L0

L1

L2

fillrandom

Q2. Trivial move



4, 5, 6

7, 8, 9

10, 11, 12

MEM

L0

L1

L2

1, 2, 3

fillseq

2, 5, 8

4, 6, 9

10, 11, 12

MEM

L0

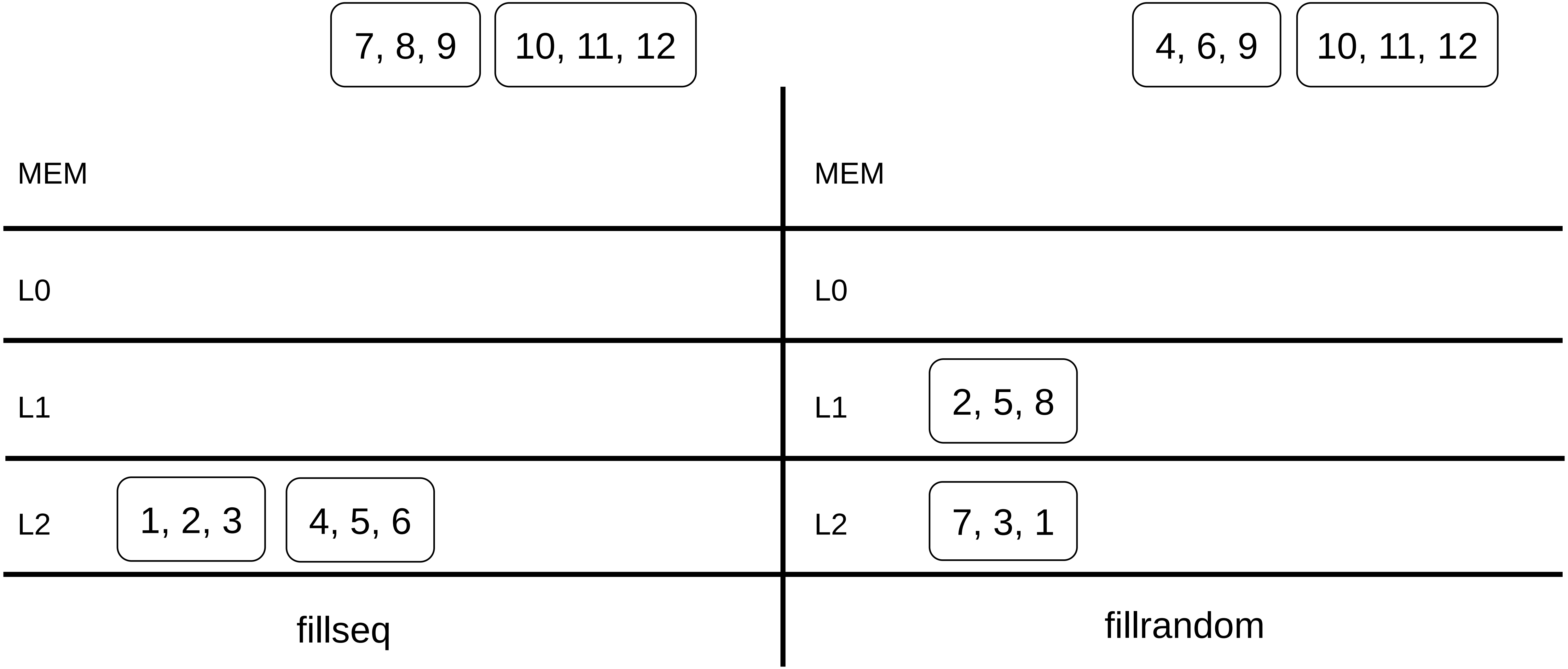
L1

L2

7, 3, 1

fillrandom

Q2. Trivial move



7, 8, 9

10, 11, 12

MEM

L0

L1

L2

1, 2, 3

4, 5, 6

fillseq

4, 6, 9

10, 11, 12

MEM

L0

L1

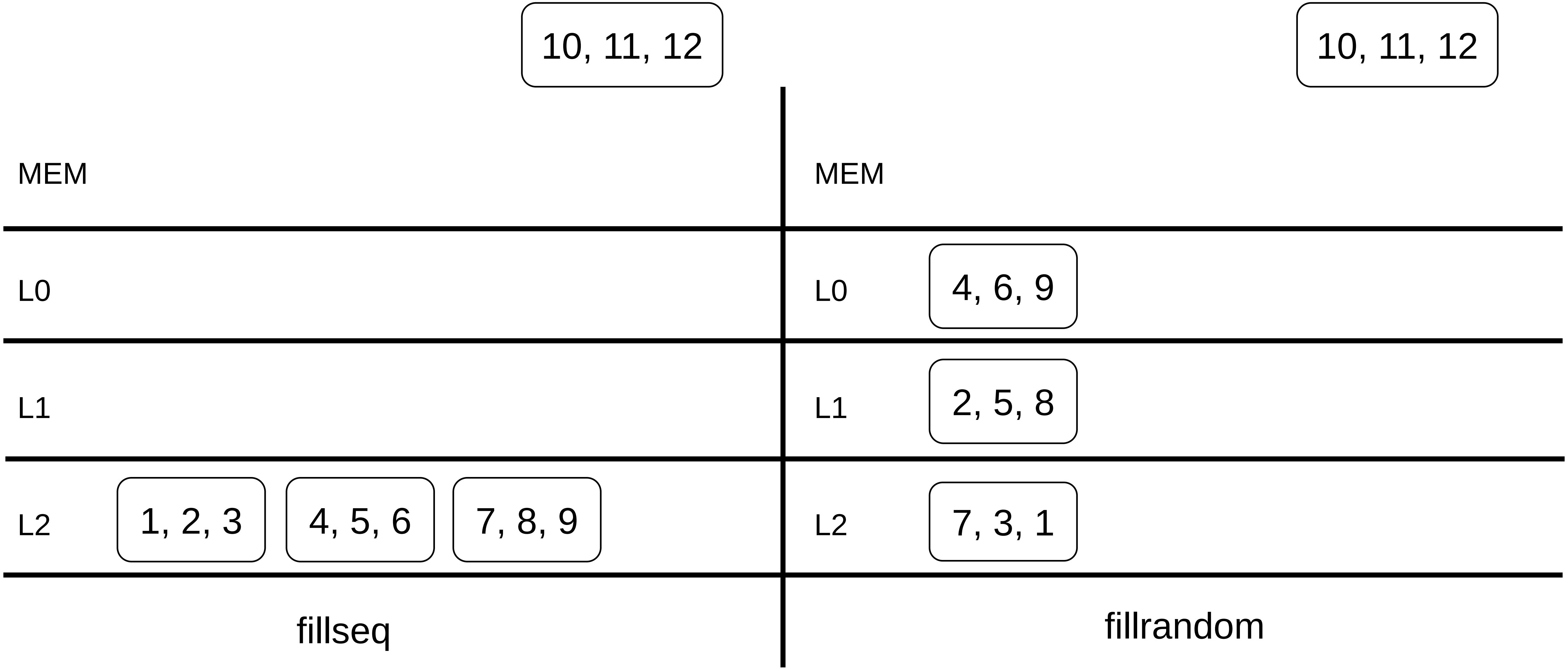
2, 5, 8

L2

7, 3, 1

fillrandom

Q2. Trivial move

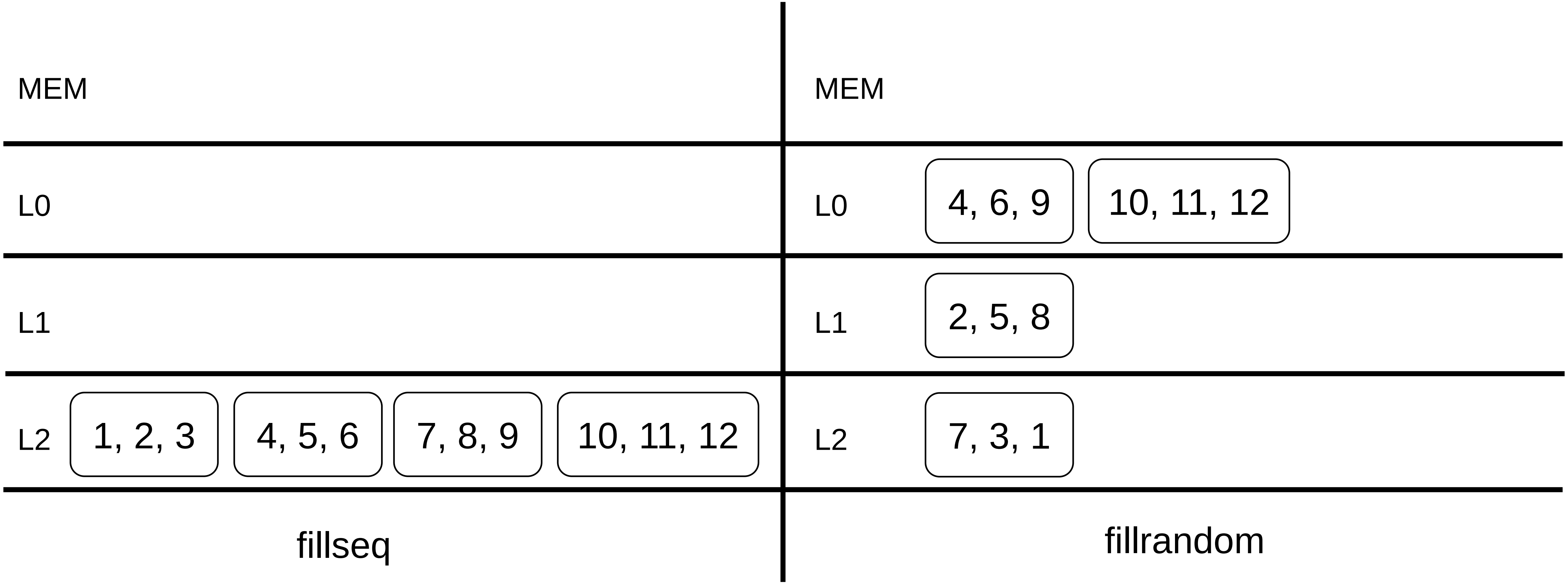


10, 11, 12

10, 11, 12

	fillseq	fillrandom
MEM		
L0		4, 6, 9
L1		2, 5, 8
L2	1, 2, 3 4, 5, 6 7, 8, 9	7, 3, 1

Q2. Trivial move

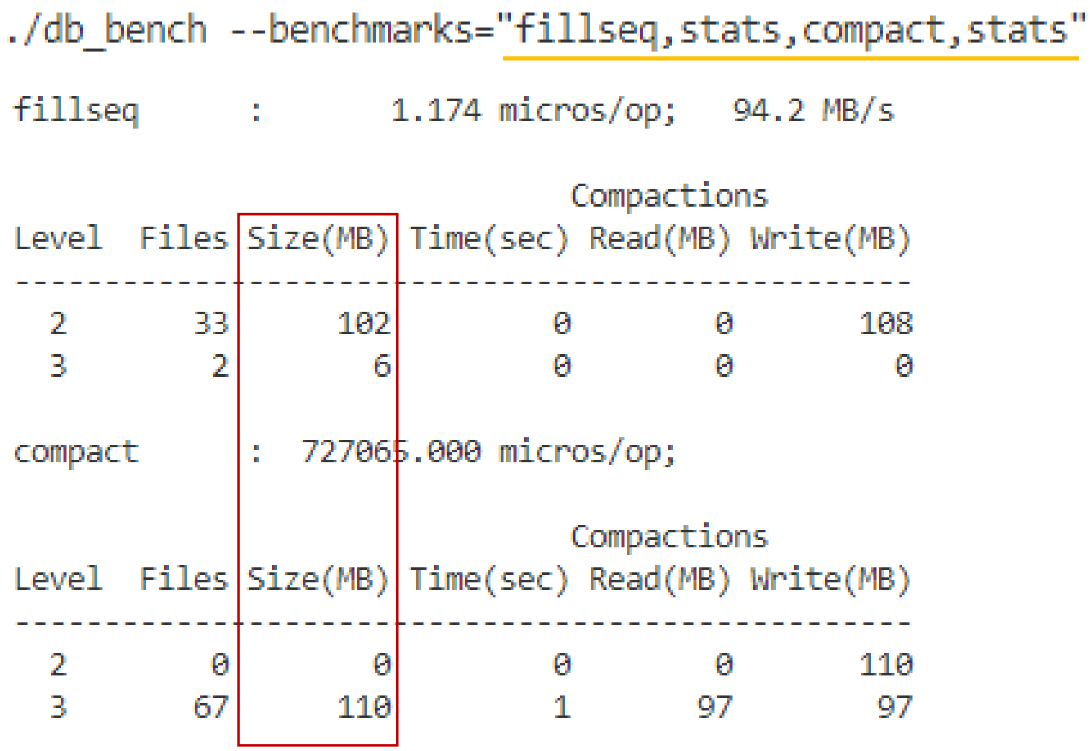


	fillseq	fillrandom
MEM		
L0		4, 6, 9 10, 11, 12
L1		2, 5, 8
L2	1, 2, 3 4, 5, 6 7, 8, 9 10, 11, 12	7, 3, 1

Q3.

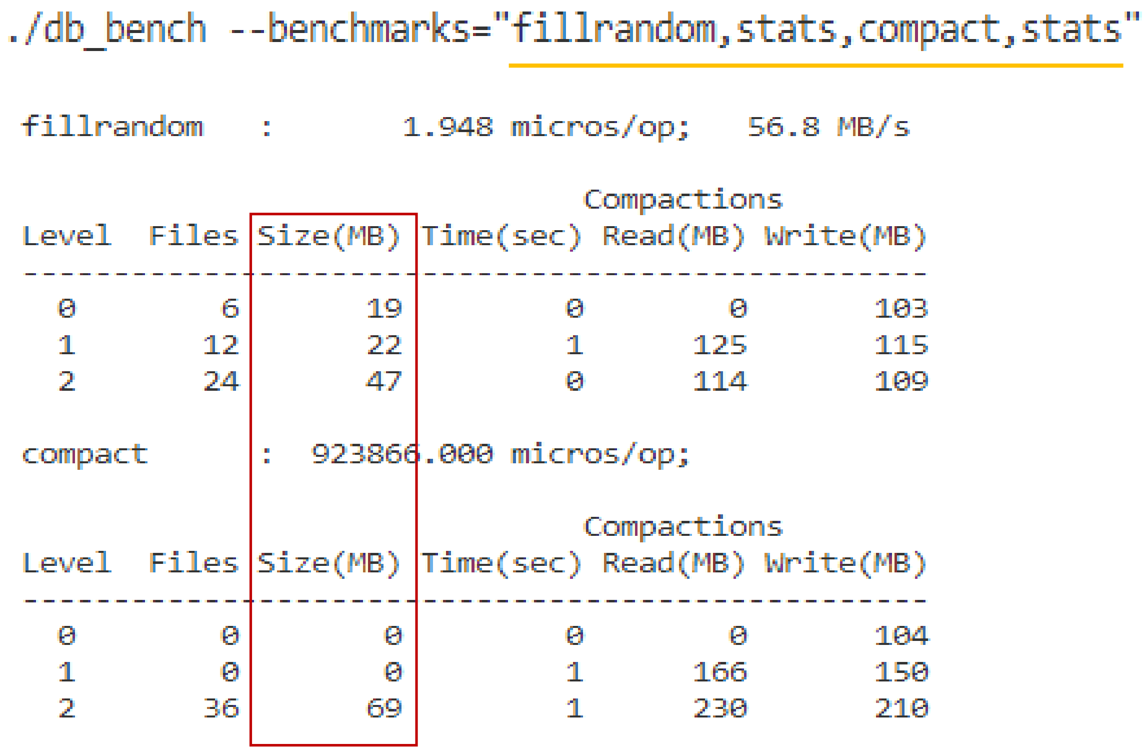
Q3. Calculate SAF (Space Amplification Factor) for each benchmark.

A. (102 + 6) / 110 = 0.98



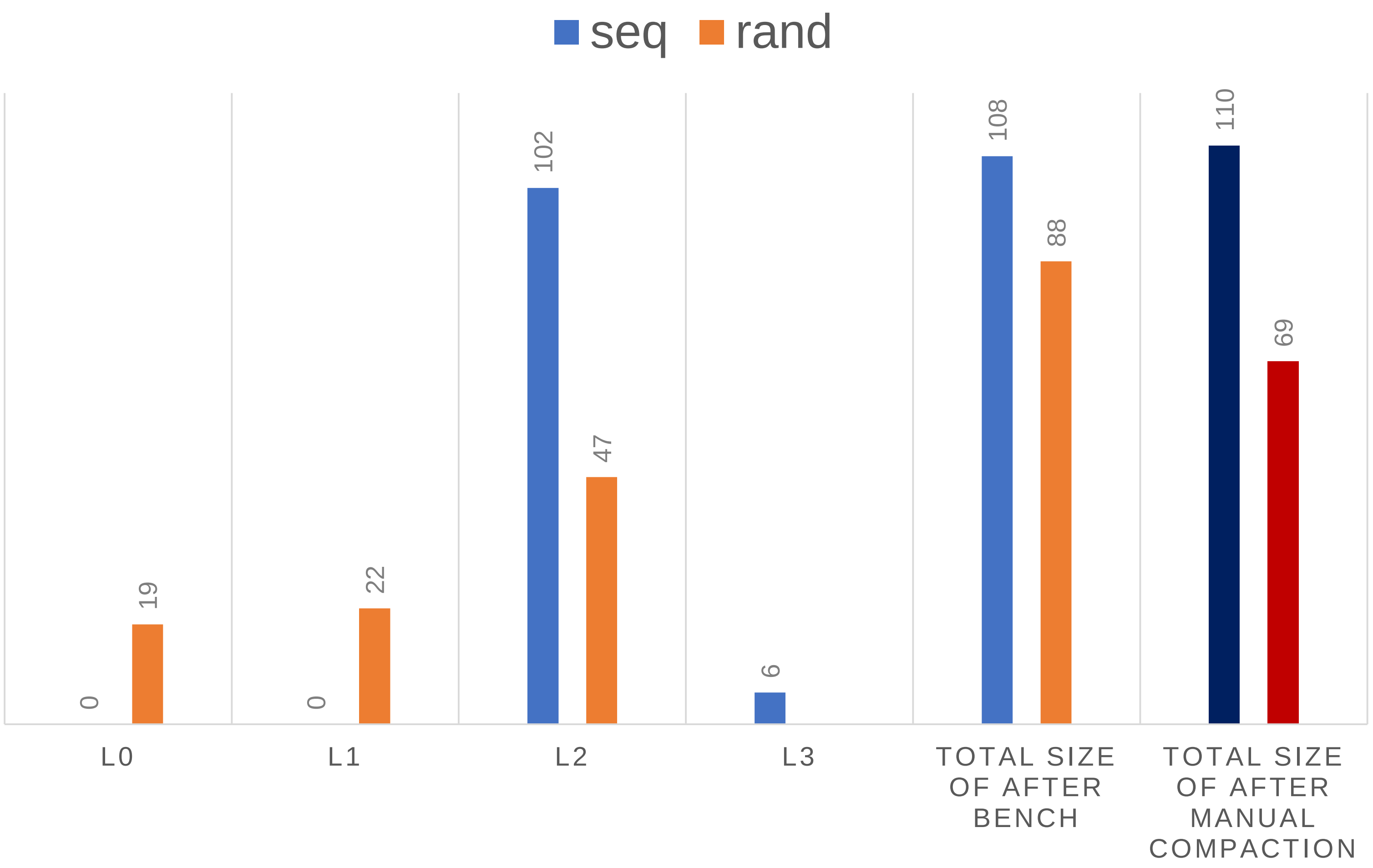
```
./db_bench --benchmarks="fillseq,stats,compact,stats"

fillseq      :       1.174 micros/op;    94.2 MB/s

                               Compactions
Level  Files Size(MB) Time(sec) Read(MB) Write(MB)
--------------------------------------------------
  2       33      102         0        0       108
  3        2        6         0        0         0

compact      :  727065.000 micros/op;

                               Compactions
Level  Files Size(MB) Time(sec) Read(MB) Write(MB)
--------------------------------------------------
  2        0        0         0        0       110
  3       67      110         1       97        97
```

A. (19 + 22 + 47) / 69 = 1.275



```
./db_bench --benchmarks="fillrandom,stats,compact,stats"

fillrandom   :       1.948 micros/op;    56.8 MB/s

                               Compactions
Level  Files Size(MB) Time(sec) Read(MB) Write(MB)
--------------------------------------------------
  0        6       19         0        0       103
  1       12       22         1      125       115
  2       24       47         0      114       109

compact      :  923866.000 micros/op;

                               Compactions
Level  Files Size(MB) Time(sec) Read(MB) Write(MB)
--------------------------------------------------
  0        0        0         0        0       104
  1        0        0         1      166       150
  2       36       69         1      230       210
```

Q3.



	size	
level	Seq (MB)	Rand (MB)
L0	0	19
L1	0	22
L2	102	47
L3	6	
Total size of After Bench	108	88
Total size of After Manual Compaction	110	69
SAF	**0.98**	**1.275**

Practice 3

```
[A]  $ ./db_bench --benchmarks="fillrandom" --value_size=100 --num=1000000 --compression_ratio=1
[B]  $ ./db_bench --benchmarks="fillrandom" --value_size=1000 --num=114173 --compression_ratio=1
```

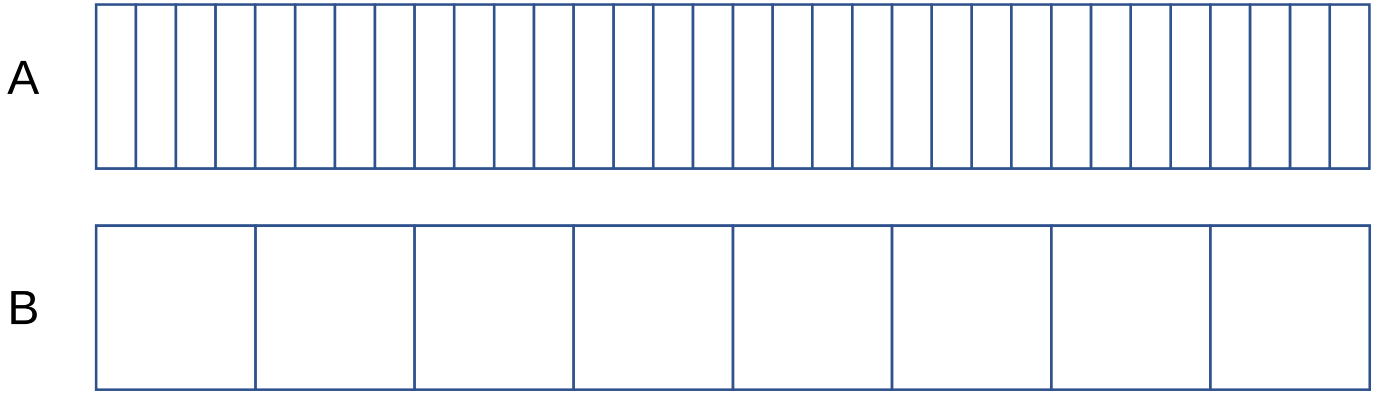
Note 1. key_size = 16 Bytes
Note 2. same total kv pairs size.
Note 3. # of B's entries = 114173 = (16+100)/(16+1000) * 1000000

Q. The size of input kv pairs is the same. But one has better throughput and latency than the other.

Explain why.

Batch processing

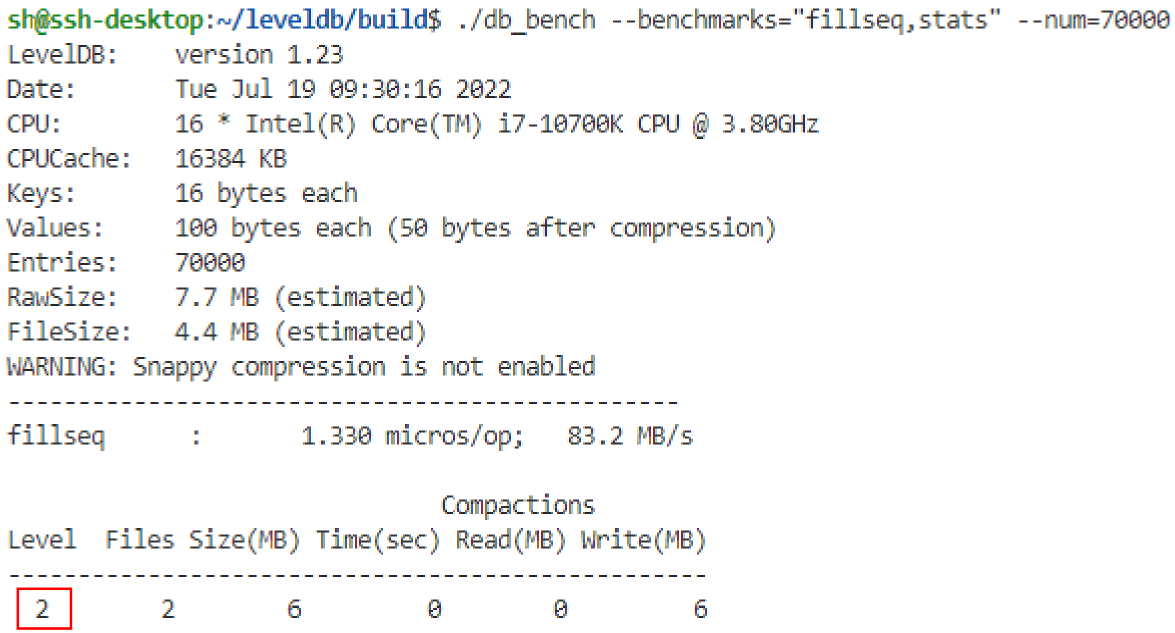
Batch Processing is a way to process the job in groups



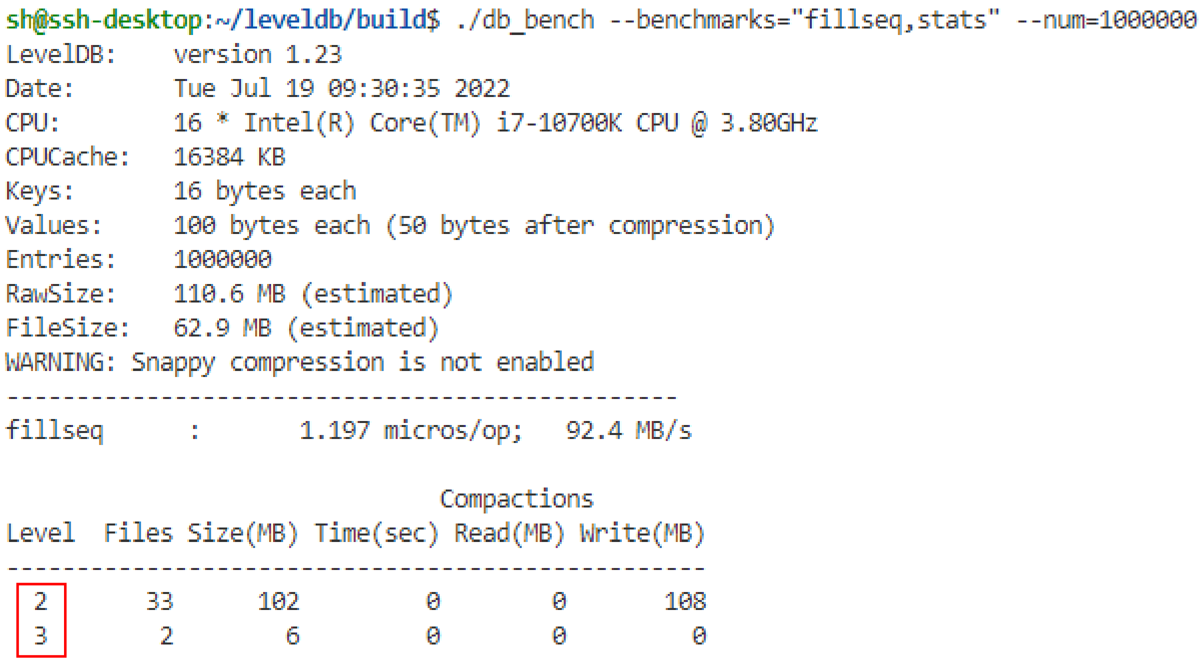
	DB size	# of entries	Entry size	Throughput	Latency
A	Same	1,000,000	116B	↓	↑
B	Same	114173	1016B	↑	↓

Appendix

- Trivial move in “fillseq”



```
sh@ssh-desktop:~/leveldb/build$ ./db_bench --benchmarks="fillseq,stats" --num=70000
LevelDB:    version 1.23
Date:       Tue Jul 19 09:30:16 2022
CPU:        16 * Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz
CPUCache:   16384 KB
Keys:       16 bytes each
Values:     100 bytes each (50 bytes after compression)
Entries:    70000
RawSize:    7.7 MB (estimated)
FileSize:   4.4 MB (estimated)
WARNING: Snappy compression is not enabled
------------------------------------------------
fillseq      :       1.330 micros/op;   83.2 MB/s

                               Compactions
Level  Files Size(MB) Time(sec) Read(MB) Write(MB)
--------------------------------------------------
  2         2        6         0        0         6
```



```
sh@ssh-desktop:~/leveldb/build$ ./db_bench --benchmarks="fillseq,stats" --num=1000000
LevelDB:    version 1.23
Date:       Tue Jul 19 09:30:35 2022
CPU:        16 * Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz
CPUCache:   16384 KB
Keys:       16 bytes each
Values:     100 bytes each (50 bytes after compression)
Entries:    1000000
RawSize:    110.6 MB (estimated)
FileSize:   62.9 MB (estimated)
WARNING: Snappy compression is not enabled
------------------------------------------------
fillseq      :       1.197 micros/op;   92.4 MB/s

                               Compactions
Level  Files Size(MB) Time(sec) Read(MB) Write(MB)
--------------------------------------------------
  2        33      102         0        0       108
  3         2        6         0        0         0
```

Appendix

- Batch processing

Advantage

- Accuracy (No human errors)
- Simplicity (No special systems)
- Efficiency (Offline feature)
- Cost savings (Automation)

Disadvantage

- Slow (relatively Stream process

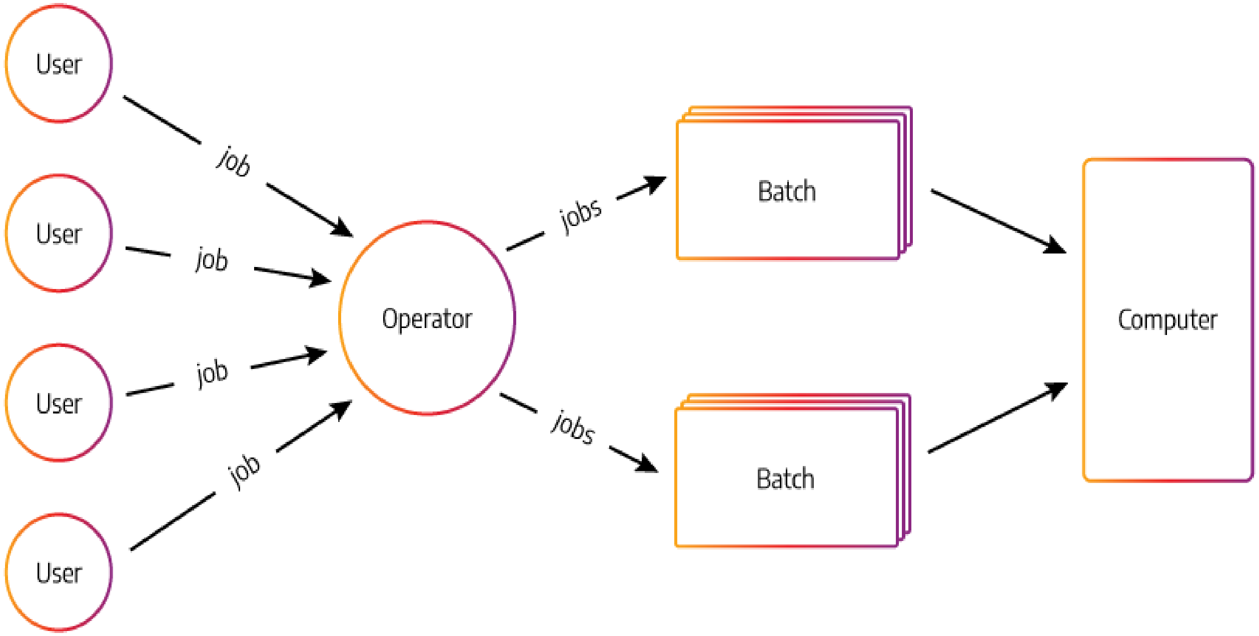


그림 출처 : https://memgraph.com/blog/batch-processing-vs-stream-processing

Question