1. SSD: How does the Flash Translation Level (FTL) work in SSDs?

FTL works by moving logical blocks around on the physical memory addresses in order to wear out all parts of the memory evenly. This is a standard made by Intel, and works by marking a used block as invalid for further use until all blocks have been used before resetting the counter. This is further improved by marking all blocks, not only used ones.

2. SSD: Why are sequential writes important for performance on SSDs?

Sequential writes are essential because there will be less overhead both during the write operation, as well as later when it needs to be read.

3. SSD: Discuss the effect of alignment of blocks to SSD pages.

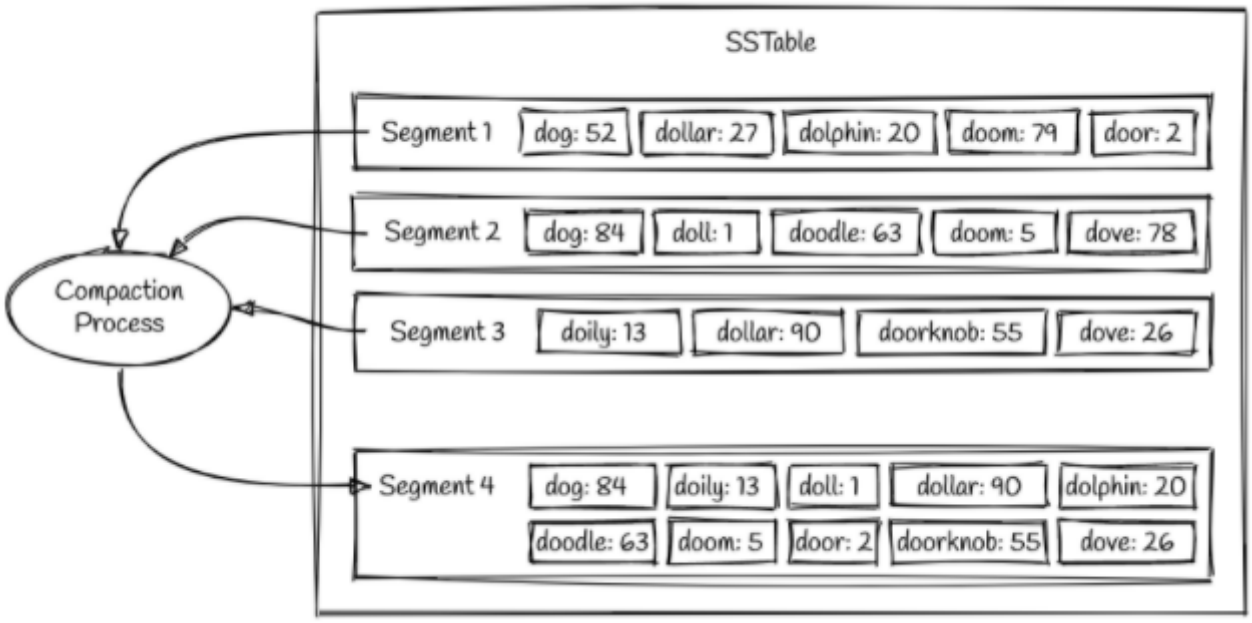
If the pages are not aligned to the blocks, there will be ocurrances of pages straddling between two blocks. Every time a page changes, it requires the entire block to be rewritten. Therefore, in the case of writing to a page straddling between blocks, both blocks will have to be rewritten. This is inefficient - both for runtime and for using more of the write cycles of the physical cells. This can be prevented by keeping every page in its own block.

4. RocksDB: Describe the layout of MemTable and SSTable of RocksDB.

RocksDB uses LSM (Log-Structured Merge-Tree) to optimize random writes to the database using Memtable and SSTable. The Memtable is a buffer that receives the writes for the database. Usually this is implemented with SkipList in RocksDB. Once this table is full, it will be flushed to the actual storage, called an SSTable (Sorted Static Table). As the name ("static") suggests, the storage is immutable. There are multiple sorted SSTables existing at once in the form of segments, and eventually, they will be compacted into one.

5. RocksDB: What happens during compaction in RocksDB?

Before compaction, there are multiple segments of sorted data, that need to be merged. Multiple SSTables will then be merged, like with merge sort, into one common SSTable through a compaction filter. During compaction, the system looks at the SSTables and only keeps the most recent update for each key, throwing away duplicate keys. It can also use Bloom filters to check if a key even exists in the dataset. An index in memory can also be created that refers to each of the sorted segments (SSTable) on disk (RocksDB uses SkipLIsts) - this helps with faster reads. All of this happens as a background process.



6. LSM-trees vs B+-trees. Give some reasons for why LSM-trees are regarded as more efficient than B+-trees for large volumes of inserts.

LSM-trees generally have faster writes than B+ trees. When this storage engine receives data, it firstly gets inserted into the memtable. When full, this gets flushed to disk as a segment into an SSTable in sorted order (hence the name "Sorted String Table"). This allows us to write data sequentially to the memtable, which is much faster as it requires less overhead.

7. Regarding fault tolerance, give a description of what hardware, software and human errors may be?

Hardware problems:
Hardware problems are usually random. This includes failure of RAM, storage drives, CPUs, power delivery etc. To accommodate this, systems can add redundency components that can be used as backup while the original part gets replaced. This includes extra RAM sticks, storage disks in RAID configuration, hot swappable CPUs, extra power supplies, backup power generators etc. Recently, systems have even been built to tolerate the loss of entire machines.

Software faults:
These are harder to predict and often include edge cases caused by various bugs. This can be caused by a runaway process that uses up shared resources, a service the system depends on that slows down, or cascading failures where one fault triggers the other and so on. These faults usually cause multiple faults when they happen.

Human faults:
This is the leading cause of failure, and happens from time to time due to humans simply not paying

enough attention. It can happen from things like accidentally unplugging the wrong storage drive, ethernet cable etc. Also, making mistakes while building systems or while maintaining them. These types of errors can be minimized by taking some precautions:

- Restrict critical access, or implementing a sandbox where practice safely can be done.
- Testing everything from unit testing to full system integration tests, as well as automated testing.
- Allowing for quick and easy recovery by minimizing amount of code being pushed out at the time, and having a system for rollback.
- Analyzing and monitoring system status to be able to take action early.

8. Compare SQL and the document model. Give advantages and disadvantages of each approach. Give an example which shows the problem with many-to-many relationships in the document model, e.g., how would you model that a paper has many sections and words, and additionally it has many authors, and that each author with name and address have written many papers?

The document model:

- Pros
  - Schema flexibility
    - Schemas-on-read, not schemaless.
    - May support certain schema changes easily.
    - Advantageous if the items in the collection don't all have the same structure for some reason.
  - Suits document-like structures.
  - Can scale vertically and horizontally.
  - No relations, only key-value-stores.
- Cons
  - Bad support for joins, forcing awkward manual joins in application code.
  - Bad support for many-to-many relationships.

SQL

- Pros
  - Good support for joins.
  - Good support for many-to-many relationships.
  - Gives room for optimizations and parallel execution e.g. using indexes.
- Cons
  - Can require ORM (Object Relational Mapping) for mismatch between DB rows and application objects.
  - Have to maintain relations.
  - Can't scale horizontally because relations will be split up.

When modelling papers to authors that have written many papers (many-to-many relationship), you would make a table of authors with their values (name, address) along with an array with reference to all the IDs of papers they have written from the papers table. Vice versa for the other table. This would be a problem if you want to join the tables, and would also mean storing (worst case entire duplicate) columns inside each entry. For a many-to-many relationship, an SQL model with a junction table would be more appropriate.

9. When should you use a graph model instead of a document model? Explain why. Give an example of a typical problem that would benefit from using a graph model.

Graph models should be used when many-to-many relationships occur frequently. This is because a graph model is keyless and depends on edges as references, which can even be bidirectional. This makes for easy modeling of even complex networks. Edges can cross between any node and makes traversing easy. An example would be modeling plane routes between airports. This allows each airport to have as many or as few routes as needed, with direction specified.

10. Column compression: You have the following values for a column: 43 43 43 87 87 63 63 32 33 33 33 33 89 89 89 33

a) Create a bitmap for the values.

Column: 43 43 43 87 87 63 63 32 33 33 33 33 89 89 89 33
Cardinality: 6 (6/16)

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
43	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
87	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
63	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1
89	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

b) Create a runlength encoding for the values

column = 43: 0, 3
column = 87: 3, 2
column = 63: 5, 2
column = 32: 7, 1

column = 33: 8, 4, 3, 1
column = 89: 12, 3, 0

11. We have different binary formats / techniques for sending data across the network: MessagePack, Apache Thrift, Protocol Buffers, Avro. In case we need to do schema evolution, e.g., we add a new attribute to a Person structure: Labour union, which is a String. How is this supported by the different systems? How is forward and backward compatibility supported?

- MessagePack
  - Does not use schemas.
  - Needs to be read and written using the same version, i.e. no forward or backward compatibility.
- Apache Thrift and Protocol Buffers
  - Use schemas, and fields are identified with tag numbers.
  - Require a schema, with tools to generate code.
  - Can change name of a field in the schema, since the encoded data never refers to field names, but you cannot change a field's tag, since that would make all existing encoded data invalid.
  - Adding fields is supported as long as you give each field a new tag number. Forward compatibility supports this by ignoring unknown field tags.
  - Every new field cannot be required because that would lose its backward compatibility.
  - You can only remove a field that is optional, and can never reuse an old field tag even though the field has been deleted. This is to avoid confusion between new tags and old ones that have been deleted and should in fact be ignored.
  - Changing datatype of a field may also be possible, but more complicated.
- Avro
  - May add or remove fields that have default values.
  - Also uses schema to specify data structure.
  - One schema language for human editing, and one for machine readability.
  - Forward compatibility means having new version of schema as writer and old version of schema as reader.
  - Backward compatibility means the opposite - you can have new version of schema as reader and an old version as writer.
  - To maintain forward and backward compatibility, you can only add or remove field with default value.
  - Changing the datatype of a field is possible, provided Avro can convert the type. This is also complicated in Avro.