The following document is in a threaded format, a continuation of “Questionnaire - OC-SSD” - following the conversations that happened before.

Order of conversation (color code):

Grey - Old Ques
White - Old Response
Yellow - New Ques
White - New Response

- In need for open channel SSD - "provisioning of location within SSD". Not exactly certain what this means.

The traditional manner of storage in SSDs is sort of using **First Fit** algorithm, i.e. finding the first available frame that fits the page. So, this might lead to **External Fragmentation**, i.e. in totality, the sum of available frames >= the storage needed.

In OC-SSD the control is in the hands of the host, so these type of problems can be “efficiently” tackled by **provisioning the location of storage,** i.e. the host machine can control the block-page where the data needs to be stored, or if a new data comes it can redirect and move the old stored data efficiently to make space for the fragmentation affected ones.

Ok. That makes sense.
However, there would also be overheads associated with this. Can you think of a few overheads that might come because now the open channel SSD is managing where the data is written, explicitly?

One of the issues with this approach will be **latency**, i.e. now the whole process of finding a suitable “**hole**” for data, might not be responsive enough. This is actually a big issue. One of the solutions can be rather than doing this “optimization” for every IO, it can be periodically strategized (for ex: finding the first fit block and after every, say 10 IO operations, the optimization of provisioning can take place).

This scenario can be somewhat related to conflicts between **FIRST FIT** vs **BEST FIT** algorithm, where OC-SSD is trying a best-fit of data storage.

- I/O isolation - how is this a problem in traditional SSDs? How is this being removed in oc SSDs?

  Traditional SSDs face a problem when multiple tenants are performing IO tasks on the same machine, as each tenant will have their separate write-read-erase tasks and definitely different scheduling of these tasks. This is why there is a need for isolating IO operations, as because of these conflicting IO tasks.

  Can you expand a little more on this? You are on the right track, but it would be good for you to write down a bit more about this. What do you mean by conflicting?

  Conflicting IO doesn't seem to be the correct terminology here.
  I'd rather replace it now with **COLLISION.** In a multi-tenant environment, the same SSD is connected with multiple tenants, thus different I/O tasks collide for the same channel (through IO PCIs or HBAs connected to SSD).

- Why is a read operation low cost? More importantly, what do you mean by cost?

  Cost = life-degrading metric of SSD

  In order to reuse a data block, HDDs simply overwrite the unneeded data blocks. But in SSDs, in order to reuse an unneeded block, the old data needs to be erased (as the whole concept of flash memory depends on storing a charge, so that needs to be cleared up before reuse). That's why "write" operations are heavy cost tasks, i.e take more time to perform an erase followed by a write.
  Moreover in SSDs, even if one page needs to written/updated, then the entire block needs to 'erased' and rewritten with updated data.

  On the other hand, read operations do not wear out SSD. Since we just read the pages, the charge remains the same, and the insulator layer (if I am correct) remains unaffected by the read operation.

  This is also one of the main reasons in SSDs, as to why read operation latencies are less than write operation.

  Do you understand the difference between the host machine providing the FTL functionality as compared to the SSD providing the FTL functionality? Can you compare and contrast the two?

  Discussed the same in the next response.

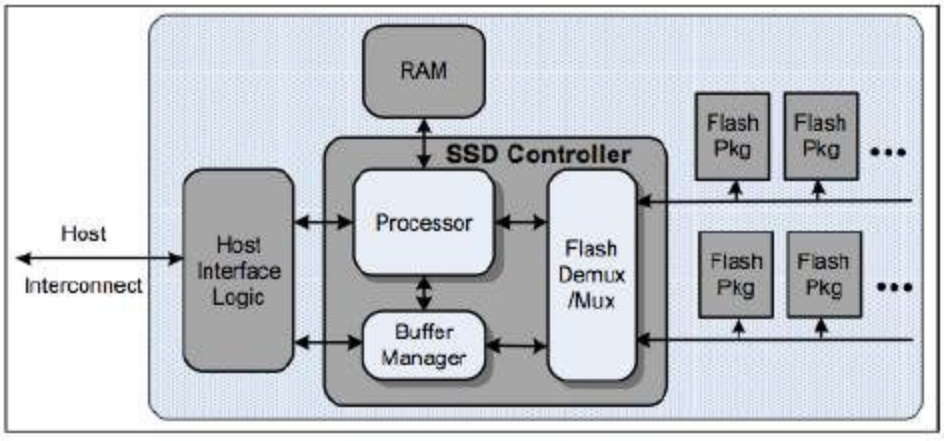
- How does host get authority over the FTL? How is that different from what happens in a traditional SSD?
  might want to get into some more detail regarding that what is lightnvm and what does it do?

  FTL is responsible for keeping a record of pages (i.e. redirecting read requests to their location, keeping a list of old/invalid pages etc) and one of the most important purposes of it is to manage wear leveling of the device. So in a way, it acts as a lookup table as well as the controller of page management tasks. In traditional SSDs, the FTL is inbuilt by the device itself, sort of firmware. The host (or the kernel, to be specific) is not able to manage the actual storage in the device.

  So one of the problems that traditional SSDs face (where FTL is embedded), is to update the actual code of it. Since it's firmware, updating or providing patches for FTL for each device is a cumbersome task. Also, if the user wants FTL to behave differently for different file systems (FAT32, exFAT), then it's a near to impossible task for the host.

  In OC-SSD, the flash transition layer is partly managed by the host machine and partly by the device. The host gets control over garbage controller, I/O scheduling and data placement, whereas error handling is controlled by the device. To get control over FTL, the host uses lightnvm.

  Where is this lookup table maintained?
  SSDs have a controller for tasks they perform. I guess lookup table is part of SSD Controller, part of Buffer Manager or somewhere closer to it's inbuilt processor. (Reference: Pic)



SSD Logic Components

Also, if the user wants FTL to behave differently for different file systems (FAT32, exFAT), then it's a near to impossible task for the host.
That is a very interesting point. Can you think of any benefits to have different FTLs for different file systems?

One of the benefits for host to have an option of maintaining different FTL for different file systems can be because of the actual difference between the file system themselves.

For example:
MAC Journaled File System vs **Windows NTFS**

MAC Journaled File System needs a lot more overhead and bookeeping than NTFS in order to 'journal' the changes brought in to the data. In case of power failure, a modified FTL layer just for MAC-Journaled-File-System would be able to help out restoring data (FTL lookup table acting as a journal itself for restoring process). Keeping the same FTL layer will be a bit overkill for NTFS systems.

This might be a bit preposterous solution for sure, considering different overheads just for doing what MAC-Journaled system seems to be doing fine on it's own.

- **DOUBTS:**

  While going through articles, I came across **Intel Optane** :
  Intel Optane Memory Tested, Makes Hard Drives Perform Like SSDs

  And in order to read more about it, this helped:
  https://youtu.be/M9jdcRfVso8

  But is this really a possibility, I tried getting in depth of how it syncs with existing RAM but wasn't able to get it much (this definitely doesn;t looks like Swap Memory).