- 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.

- The last sentence makes little sense as well. What causes wear and tear in SSDs? How do you balance it out? Why (and what kind of) fragmentation is happening and how do you get rid of that?

  The major drawback of SSDs is **longevity**. Each time a flash memory goes through an erase operation, it leaves a bit of charge, which in turn degrades the performance (even a single bit write operation needs an erase operation to occur). So each SSD can only be used for a particular number of erase cycles, after that it just happens to be in a “zombie” state.

  To ~~confront~~ this problem, a process called wear leveling is used. Wear leveling makes sure that each block is used equally, i.e. each block has gone through “almost” similar count of erase cycles. Though we still can’t protect them from eventually wearing out but balancing out the erase operations can at least make sure we use the available memory to their full potential.

- 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, the latency of each task becomes **unpredictable**, due to unpredictable collisions from each user.

  Partitioning the SSD in a logical manner is a solution for this. To perform IO Isolation, each tenant is limited to perform IO tasks to a few parallel units of a device (**L**ogical **U**nit **N**umbers acting as UUID's to these partitions). This way, IO’s from different tenants, reaching the same device would not interfere with each other. Each tenant maintains there set of LUN's, and have exclusive control over these. And since host machine has control over FTL and data placement, so it's easy to maintain this new architecture.

But again, this solution will face problems, over the decision of allocation of LUNs to each tenant. As not all tenants perform the same type of tasks (some might be more focused towards write-operations, others more focused on low-cost read operations). To solve inter-tenant IO tasks collision, this might itself create intra-collision over less available LUNs.

- 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 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 of 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 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 the control over FTL, the host uses lightnvm.

lightnvm itself is an interface, that acts as an abstract layer of implementing various system calls, irrespective of the underlying architecture.

Complete Picture of lightnvm in working:

host machine uses lightnvm API’s (at the application level)

↓

lightnvm acts on these API calls (acts as an interface b/w application level and kernel)

↓

calls lightnvm media manager (at the kernel level)

↓

parses & redirects to NVMe Device Driver (acts as an interface between kernel and h/w)

- **DOUBTS:**

  It’s mentioned that in OC-SSD, the host doesn’t get control of wear leveling? But it’s important to have wear leveling in host control so that with the IO scheduling that they plan, they also do keep check of balancing in the device.

  I tried searching this, wasn’t able to find a concise solution to this:
  Why can’t the host get control over wear leveling?

  Does it have to do something with “different host machines (tenants) trying to implement their own logic of wear leveling”?