RTS ASSIGNMENT

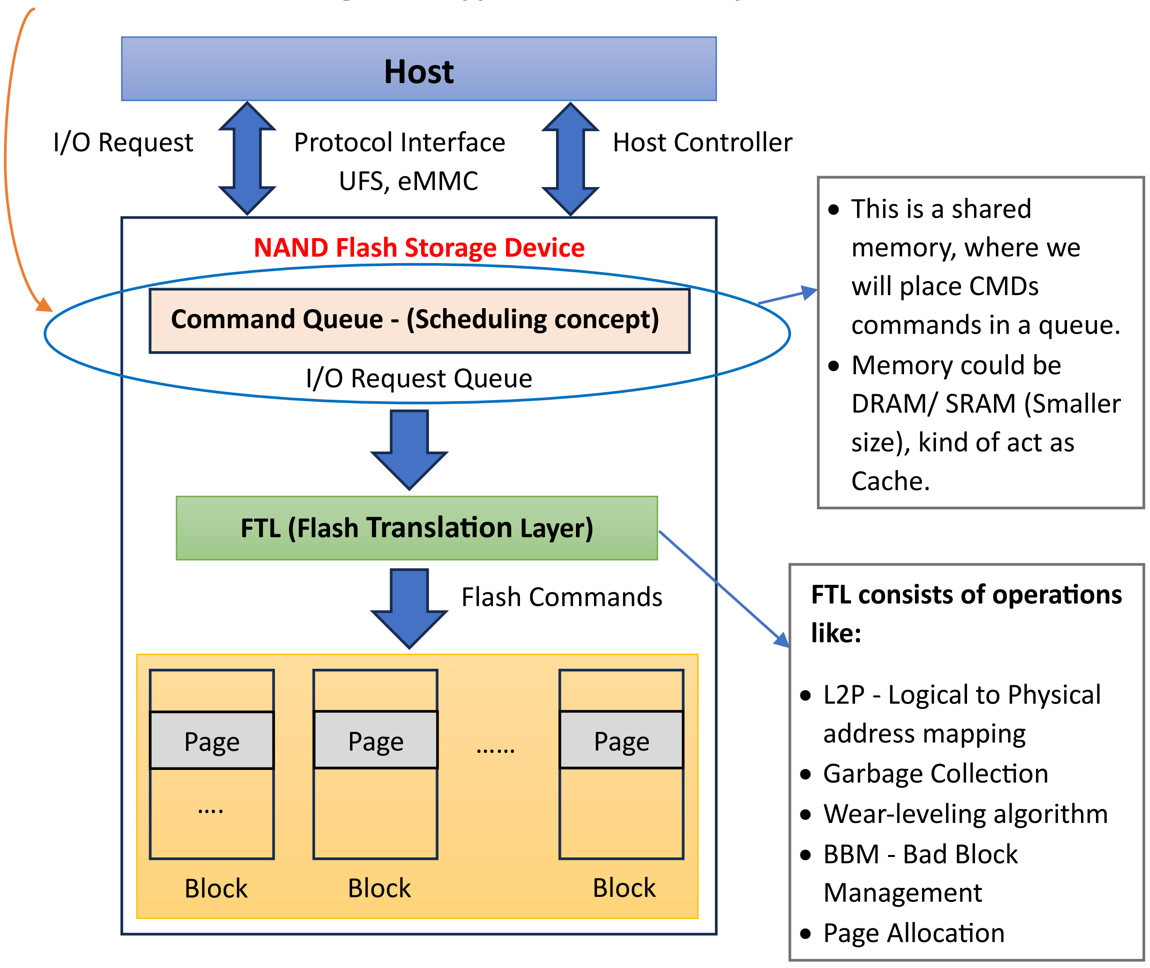
Name: CHOTHANI SAGAR VINUBHAI
BITS ID: 2023MT13010 – (2023mt13010@wilp.bits-pilani.ac.in)
Organization Mentor: Rinkal Patel (Western Digital)

Title: **Internal Command Scheduling in NAND Flash Memory Firmware for Embedded System to Improve Performance. (Real-Time Scheduling)**

Introduction:

Embedded systems are becoming increasingly sophisticated, often relying on real-time guarantees for data access and processing. NAND flash memory, despite its widespread use, presents challenges for real-time systems due to the complex nature of its write operations. Many NAND flash storage systems access flash memories by generating flash commands to process input/output (I/O) requests from a host, Because the order which flash commands are processed affects the I/O performance. Thus, the priority and queue-based scheduling concepts are more helpful to implement the firmware. This assignment explores how real-time scheduling concepts can be employed in NAND flash memory firmware to manage write and read requests effectively, ensuring timely data access and system predictability in devices like smartphones, wearables, and industrial controllers that use UFS (Universal Flash Storage), eMMC (Embedded Multimedia Controller), or similar flash storage solutions like external storage MicroSD, USB drives, etc.

- **Where Real-Time Scheduling can be applied in Flash memory Firmware?**


Host
I/O Request
Protocol Interface
UFS, eMMC
Host Controller
NAND Flash Storage Device
Command Queue - (Scheduling concept)
I/O Request Queue
FTL (Flash Translation Layer)
Flash Commands
Page
Page
Page
....
......
Block
Block
Block
This is a shared memory, where we will place CMDs commands in a queue.
Memory could be DRAM/ SRAM (Smaller size), kind of act as Cache.
FTL consists of operations like:
L2P - Logical to Physical address mapping
Garbage Collection
Wear-leveling algorithm
BBM - Bad Block Management
Page Allocation

(Block Diagram of Flash Memory & It's Firmware flow operation)

⇨ The Above Block Diagram represents the Flash memory Overview architectural view, where pictorial view is showing How Host can issue the operation for Flash and While doing Operation, how **Firmware** is uses Real-time scheduling concept for placing the commands in a queue to boost the performance.

- **What is the product and where Real-time scheduling is applicable to my Current Project?**

⇨ The Product is UFS (Universal Flash Storage), and the Project is Firmware Development for UFS flash memory.

⇨ Since, the customer is entire global market where we can consider Internal Memory is required for that we should develop a memory and its firmware such a way that it should fulfill all requirements and customer claims all cases.

⇨ Basically, Firmware is the portion which runs without an operating system. But, when comes for actual usage, User should expect the proper operation with respect to the Write, Read, Erase, Locking, Security, No Data Corruption, Data Retention, Endurance, etc.

⇨ When Firmware is designed, it should maintain the scheduling of Internal SCSI commands for UFS memory and flash the data into NAND.

⇨ UFS memory architecture model refers to SCSI standard. (Small Computer System Interface)

⇨ Firmware uses the Shared memory concept to process the actual commands to the NAND. Where Shared memory is SRAM (Size is 2 MB). Here, SRAM is the portion where scheduling concept is applied for processing the commands which are placed in queue.

- **Let's Understand What Real-Time Scheduling is possible in Firmware?**

Real-time scheduling involves managing tasks with strict timing constraints. Two common scheduling strategies are involved in our Firmware development to enhance the performance of Write and Read operation:

1. **Rate Monotonic Scheduling (RMS):** Assign priorities based on task periods. Shorter periods imply higher priority.
2. **Earliest Deadline First (EDF):** Prioritize tasks based on their deadlines, executing the task with the closest deadline first.

⇨ RMS applies in firmware where, multiple Write or Read operations are there based on task tags that time based on task tag value priorities will be considered and particular command will get processed to FTL layer, where using NAND handling mechanism at the end actual data will be written on NAND.

⇨ In case of EDF, to make this algorithm useful, we Firmware decides the which command to process first. For this, from the Host side we must make sure the flow of operation and send the multiple commands in a queue, where all commands will access the SRAM shared memory address location and it gets placed in a queue format.

⇨ Once the Command queue is formed, Firmware recognizes the operation flow considering the first entered command is having the higher value of data and task tag which to be programmed first and then rest all operation to be followed.

⇨ This process is determined during the design implementation. So, as a Host side we can make sure the which data to be programmed into the NAND and once we get the response, we can Read the same data in a contagious manner. Here, Start of Metablocks address always matters. Because Implementation of algorithm is in such a way that consider multiple of block address which falls under one the any metablocks, from those points it starts reading and sending the data to the Host.

- **How Utilizing of Real-Time Scheduling in NAND Flash Memory Firmware?**

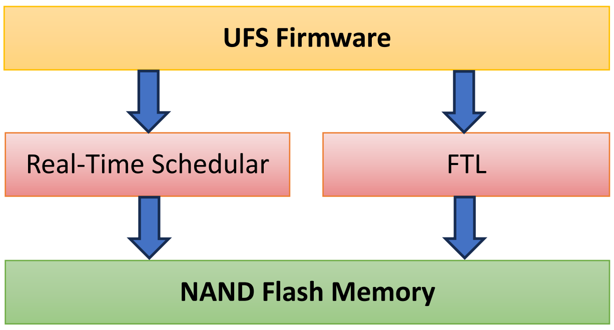
In NAND flash memory firmware, real-time scheduling ensures that critical operations meet deadlines, enhancing system responsiveness and reliability. Here's how real-time scheduling can be applied:

1. **Task Prioritization:** Assign priorities to flash memory operations based on their criticality and deadlines. For example, erase operations may have higher priority than write operations to prevent block erase latency from impacting system performance.
2. **Interrupt Handling:** Utilize interrupts to respond promptly to time-sensitive events, such as read or write completions. Prioritize interrupt service routines (ISRs) using real-time scheduling algorithms to minimize response time.

3. **Wear-Leveling and Garbage Collection:** Schedule wear-leveling and garbage collection tasks judiciously to prevent excessive latency. Real-time scheduling ensures these background tasks do not interfere with time-critical operations.

- **How Real-Time Scheduling Action is applied in NAND Flash memory Firmware?**

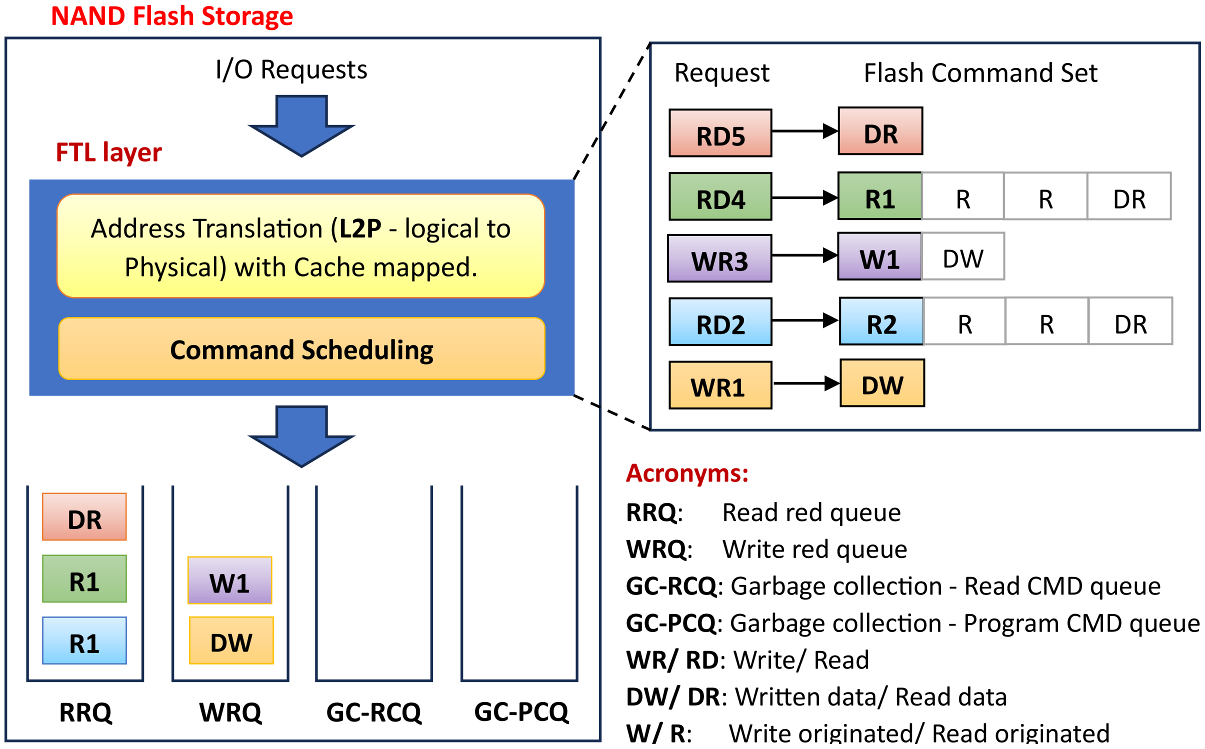
Let's understand with the simple diagram.
Below is a diagram illustrating the application of real-time scheduling in NAND flash memory firmware:



In this diagram:

- The Real-Time Scheduler prioritizes flash memory operations based on their deadlines using RMS or EDF.
- Flash Translation Layer translates high-level commands into NAND flash-specific operations, ensuring compliance with real-time scheduling requirements.
- NAND Flash Memory stores and retrieves data, with operations managed by the firmware.

- **How Read Request First Scheduling Works in Firmware?**



Acronyms:

RRQ:	Read red queue
WRQ:	Write red queue
GC-RCQ:	Garbage collection - Read CMD queue
GC-PCQ:	Garbage collection - Program CMD queue
WR/ RD:	Write/ Read
DW/ DR:	Written data/ Read data
W/ R:	Write originated/ Read originated

(Flash command queue is divided into a read request queue (**RRQ**), a write request queue (**WRQ**), and **GC command queues** so that read requests are processed first.)

Description: Above diagrams represents when the flash storage usually schedules flash commands than how in Firmware read request can be processed first.

⇨ Host applications generally make synchronous read request to the storage. The application is blocked until the data is received by the read request.

⇨ In contrast, most write occurs asynchronously. In this case, application will try to finish its own request without waiting write request. so, if read requests processed quickly via CMD scheduling than host's responsivity gets increased.

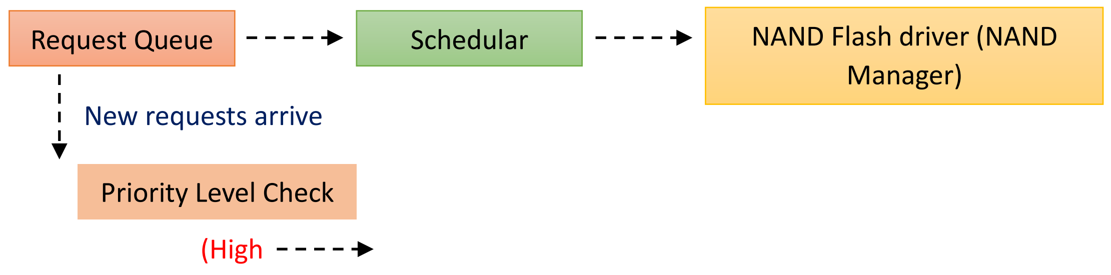
⇨ So, in Firmware CMDs are divided to **RRQ**, **WRQ** & **GC** queue to propose **Read request first (RRF)**.

- When Host send I/O request, it will be translated in FTL layer. Since, FTL can handle write and read commands, it is difficult to process read request first if flash commands are queued together.
- Consider pictorial view in the diagram, read and program commands are queued in RRQ or WRQ. If data is being read than it will be noted as DR under RRQ and for written data DW under WRQ.
- Similarly, if multiple requests are there with write(W) and Read(R), then it will be queued accordingly.
- Since, read req always makes synchronize reading, will get higher priorities, and will get placed in RRQ, which gets executed first and WRQ will have waiting period until read finished. This process is being done in Cache segment when **Logical address** comes from **Host** that time in Firmware it will be **mapped** with **Physical location** and entry will be saved in one of the internal handled **Buffer**.
- GC is to make sure, when any blocks gone bad that time written/ read data will get moved to GC-RCQ or GC-PCQ respectively.

Experience (How the Concept is relevant?)

- This concept is pretty much relevant, to make memory operation faster and at the product gives more Endurance and High-speed performance to compete the global market.
- RTS concept helps to schedule the flash commands in a priority-based scheduling. It also helps in scheduling logic like when the scheduler continuously scans the request queue. It prioritizes requests based on the following logic:
  - Deadline (means consider Command Timeout limit)
  - Priority Level (As explained in Read request first scheduling process)
  - FIFO driven policy (when commands are in Queue - SRAM(Cache) holds this operation)
- By using this Firmware implementation, we can achieve the Performance. Also, we can overcome the challenges like:
  - **Erase Before Write:** Data can only be written to pre-erased blocks. This erase operation can be slow (millisecond range) and introduces delays in write processing.
  - **Wear Leveling:** Repeated writes to the same location degrade NAND flash cells. Wear leveling algorithms distribute writes across the memory to ensure even wear, further increasing write latency.
  - **Bad Block Management:** NAND flash memory can develop bad blocks over time. The firmware needs to identify and manage these blocks, potentially requiring remapping operations and adding to write complexity.
- **Preemption:** The scheduler can preempt a lower priority write operation currently in progress if a high-priority request with a tighter deadline arrives. This ensures critical tasks meet their deadlines even if lower priority writes are ongoing.
- **Completion Handling:** Once a write request is serviced, the firmware updates its status and removes it from the queue. Any errors encountered during the write operation are reported for appropriate handling.

Simplest Pictorial view representation, Firmware Handling



Conclusion

By integrating real-time scheduling concepts into NAND flash memory firmware, storage systems can meet stringent timing requirements, ensuring timely and reliable operation. Through effective task prioritization, interrupt handling, and scheduling of background operations, firmware designers can optimize system performance and responsiveness.