

# P&S Modern SSDs

Understanding and Designing  
Modern NAND Flash-Based Solid-State Drives

Dr. Jisung Park

Prof. Onur Mutlu

ETH Zürich

Spring 2021

24 March 2021

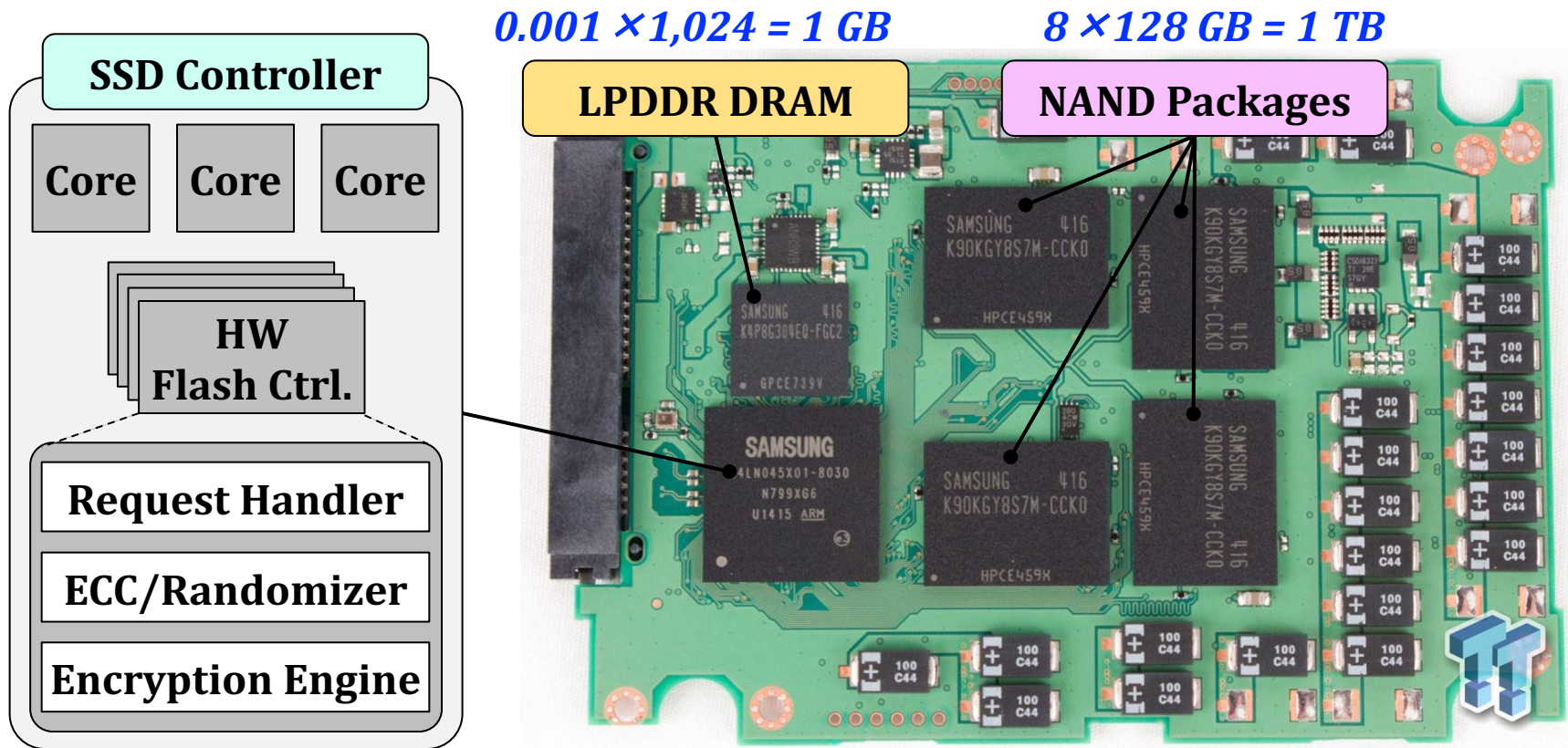
# Today's Agenda

---

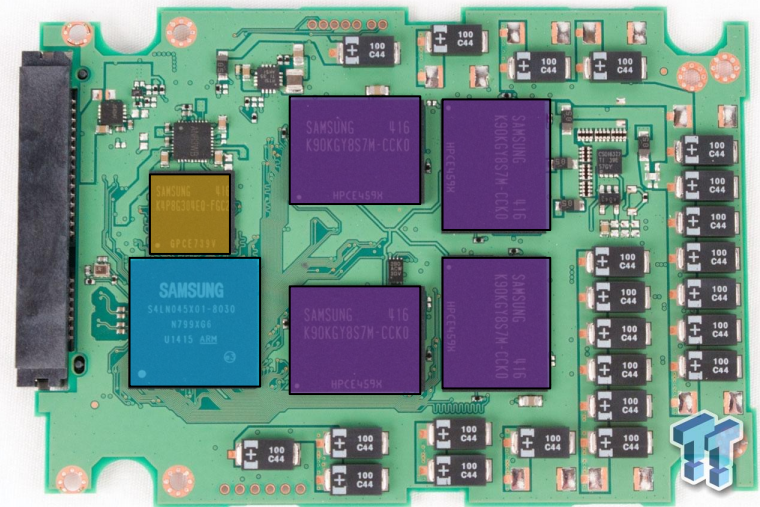
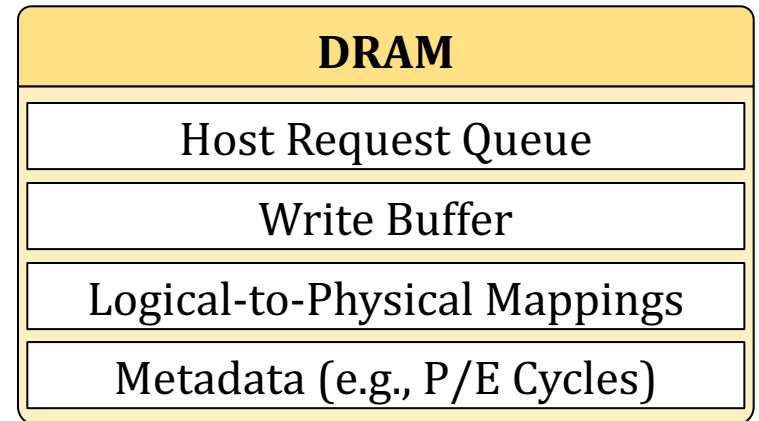
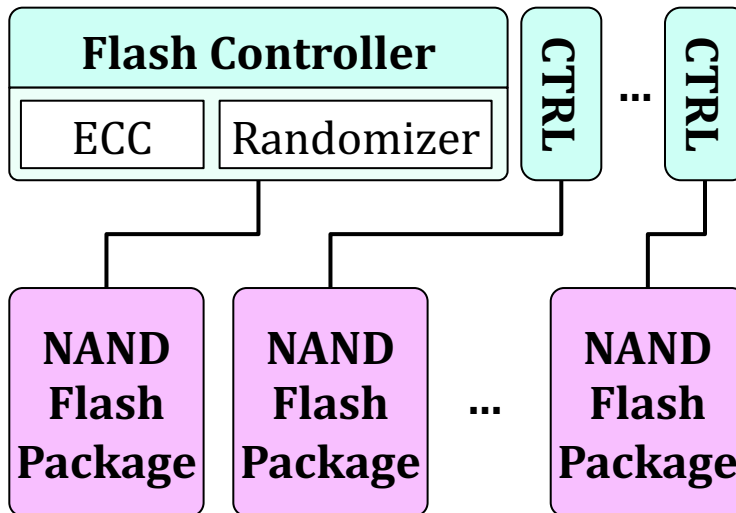
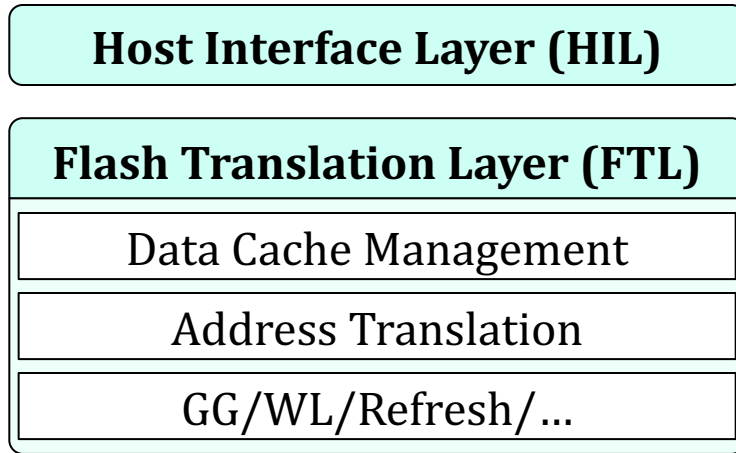
- Request Handling in Modern SSDs
- MQSim Organization

# Recap: Modern SSD Architecture

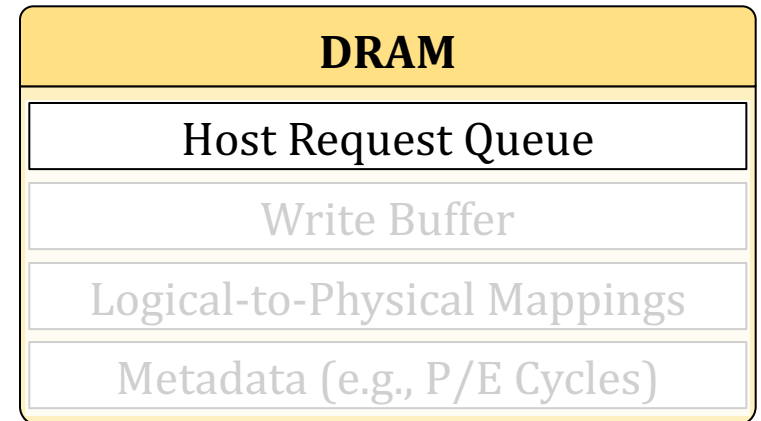
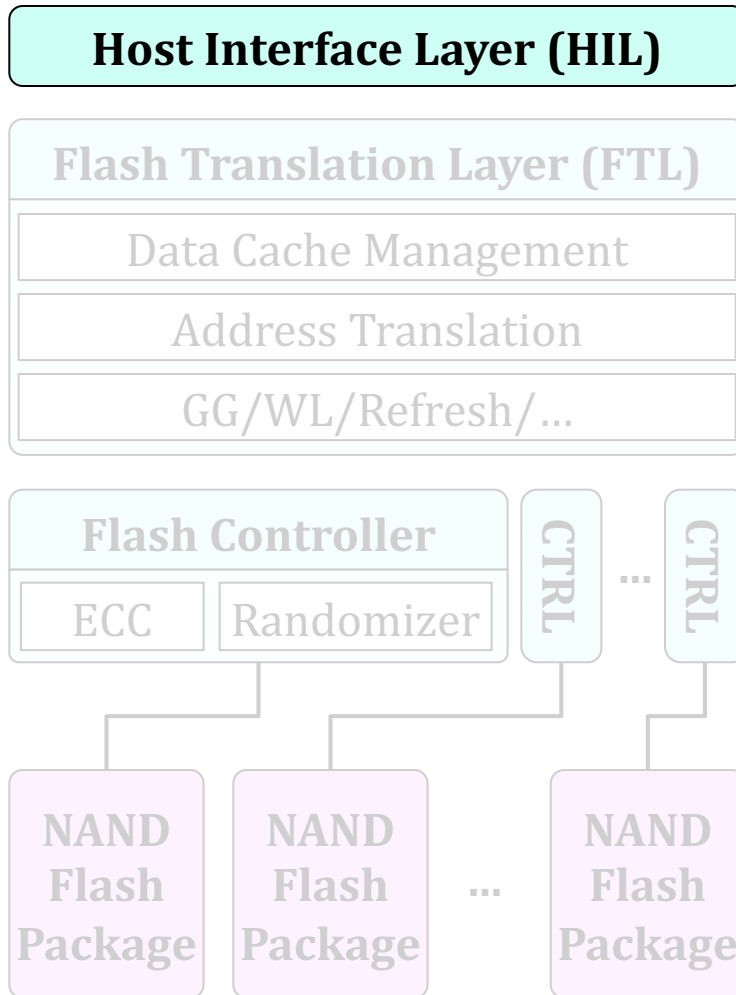
- A modern SSD is a **complicated system** that consists of **multiple cores, HW controllers, DRAM, and NAND flash memory chips**



# Another Overview

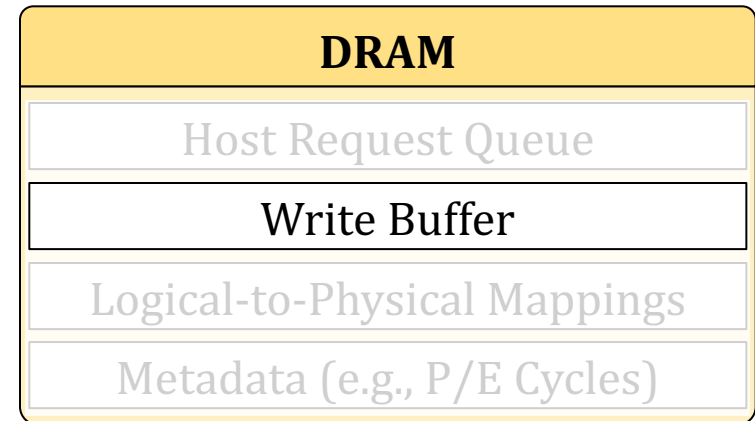
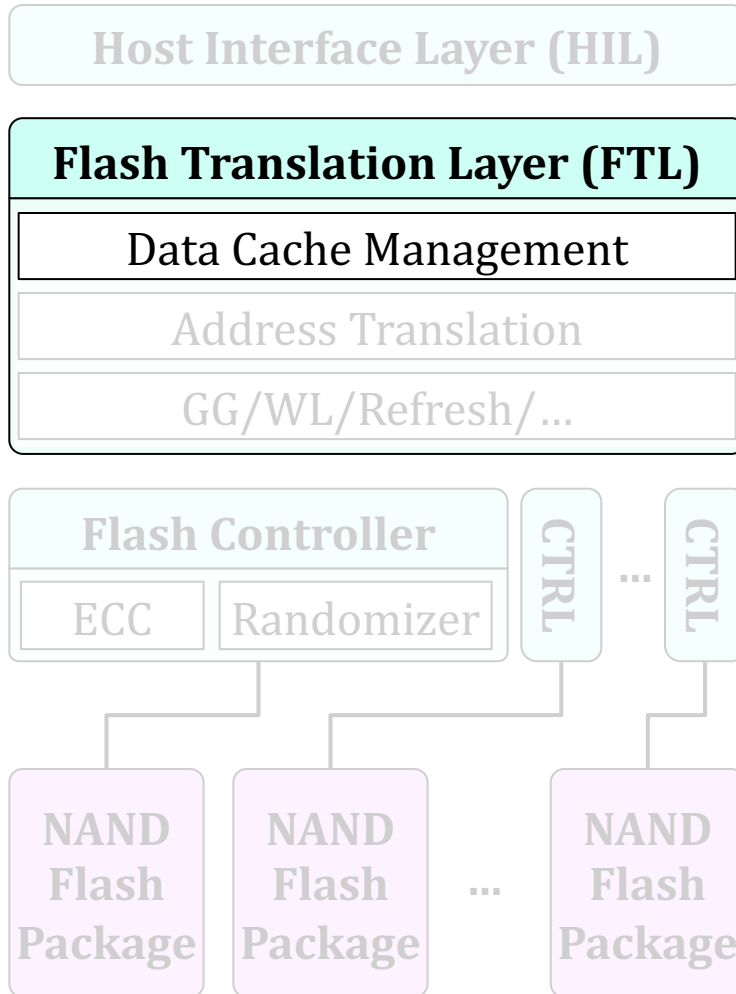


# Request Handling: Write



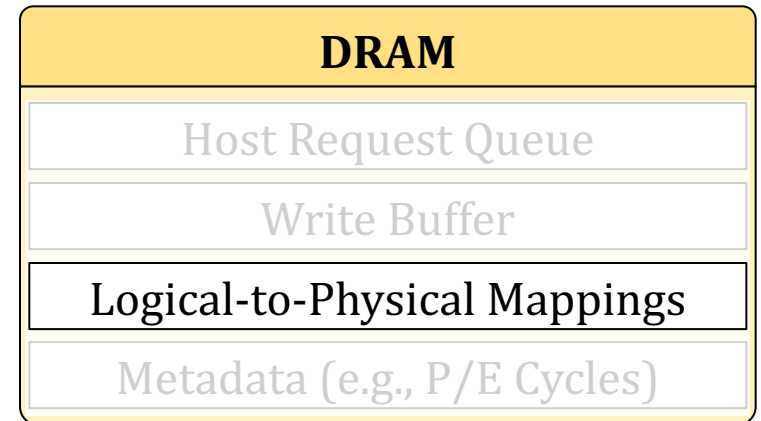
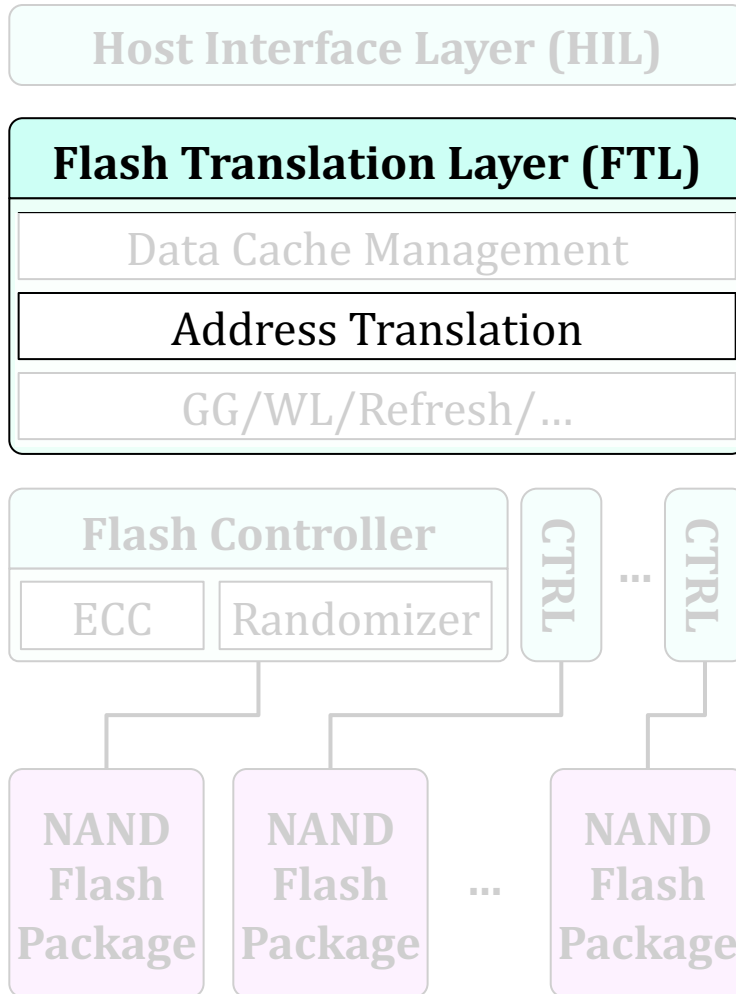
- Communication with the host operating system (receives & returns requests)
  - Via a certain interface (SATA or NVMe)
- A host I/O request includes
  - Request direction (read or write)
  - Offset (start sector address)
  - Size (number of sectors)
  - Typically aligned by 4 KiB

# Request Handling: Write



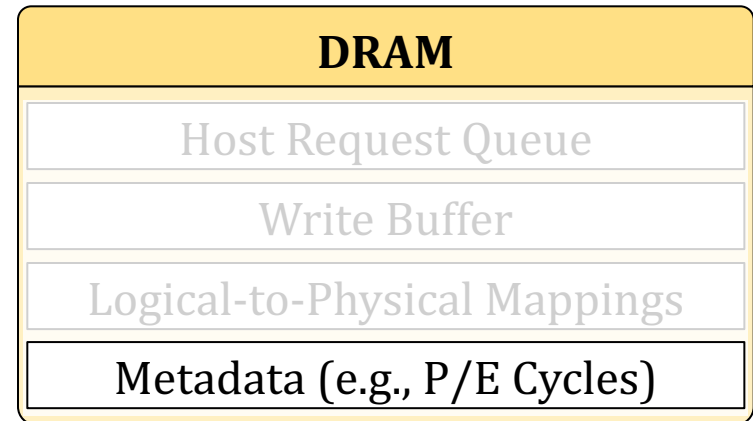
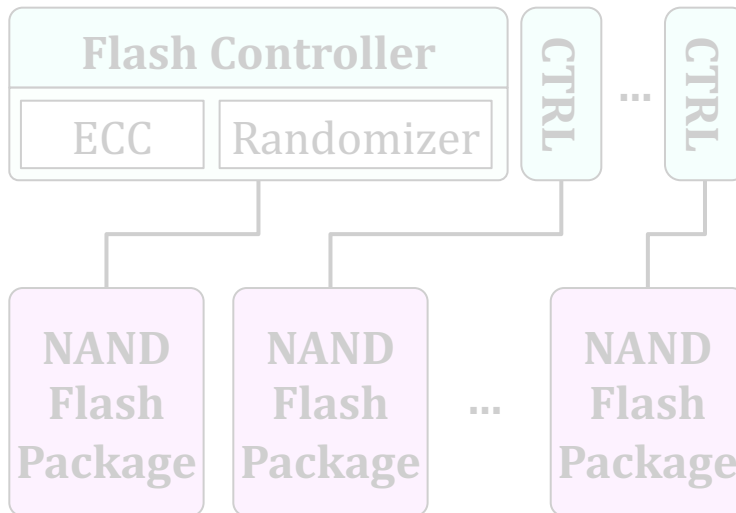
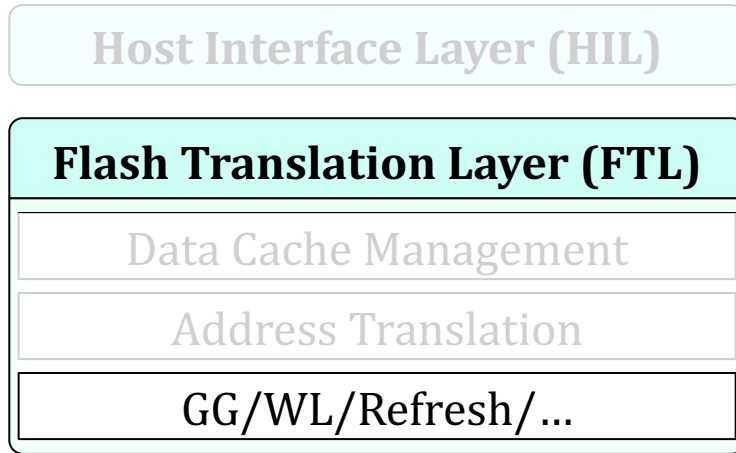
- Buffering data to write (read from NAND flash memory)
  - ❑ Essential to reducing write latency
  - ❑ Enables flexible I/O scheduling
  - ❑ Helpful for improving lifetime (not so likely)
- Limited size (e.g., tens of MBs)
  - ❑ Needs to ensure data integrity even under sudden power-off
  - ❑ Most DRAM capacity is used for L2P mappings

# Request Handling: Write



- Core functionality for out-of-place writes
  - To hide the erase-before-write property
- Needs to maintain L2P mappings
  - Logical Page Address (LPA)  
→ Physical Page Address (PPA)
- Mapping granularity: 4 KiB
  - 4 Bytes for 4 KiB → 0.1% of SSD capacity

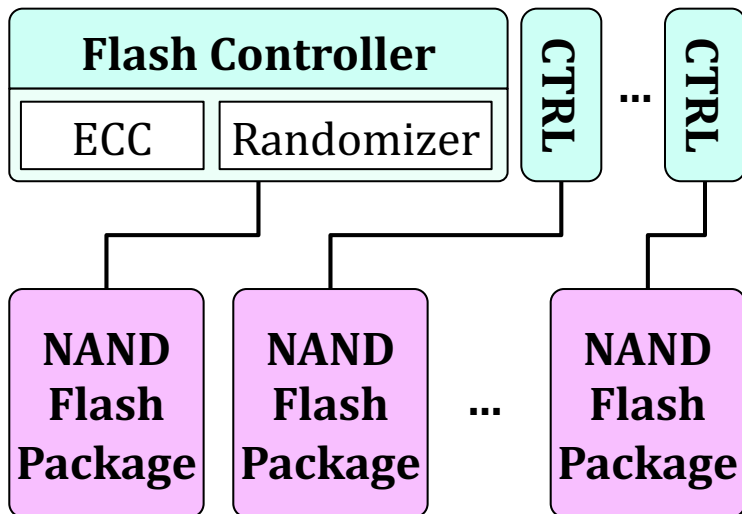
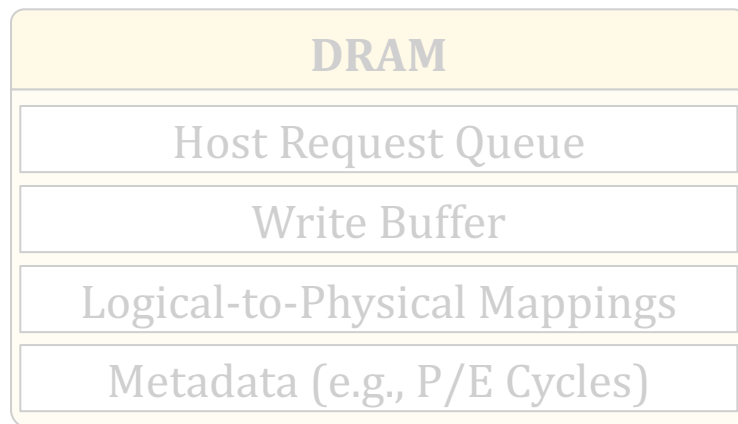
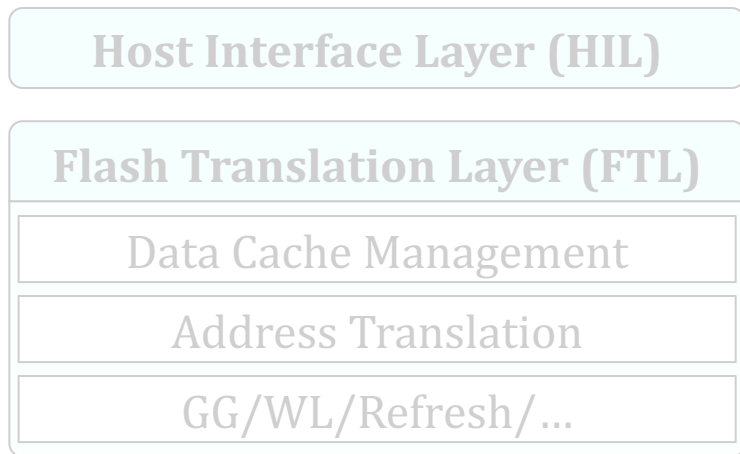
# Request Handling: Write



- Garbage collection (GC)
  - Reclaims free pages
  - Selects a victim block → copies all valid pages → erase the victim block
- Wear-leveling (WL)
  - Evenly distributes P/E cycles across NAND flash blocks
  - Hot/cold swapping
- Data refresh
  - Refresh pages with long retention ages

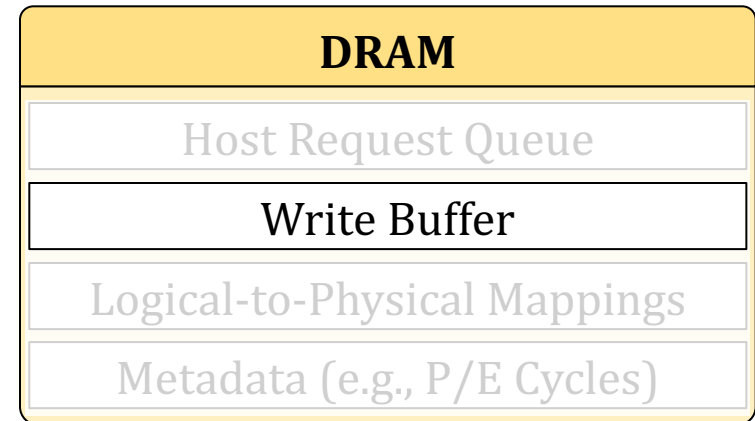
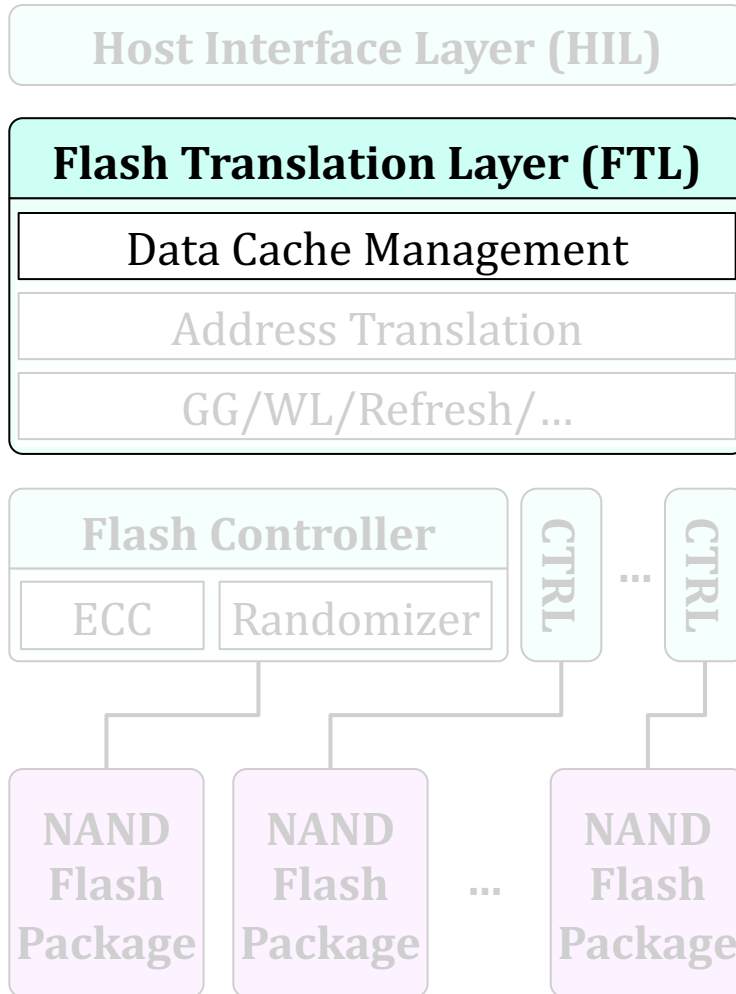


# Request Handling: Write



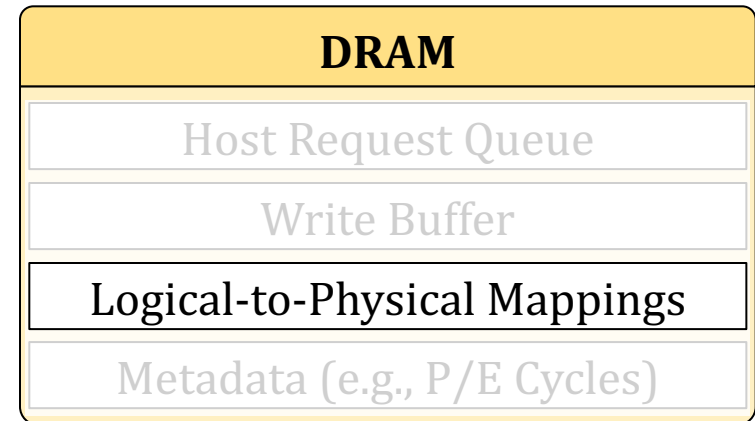
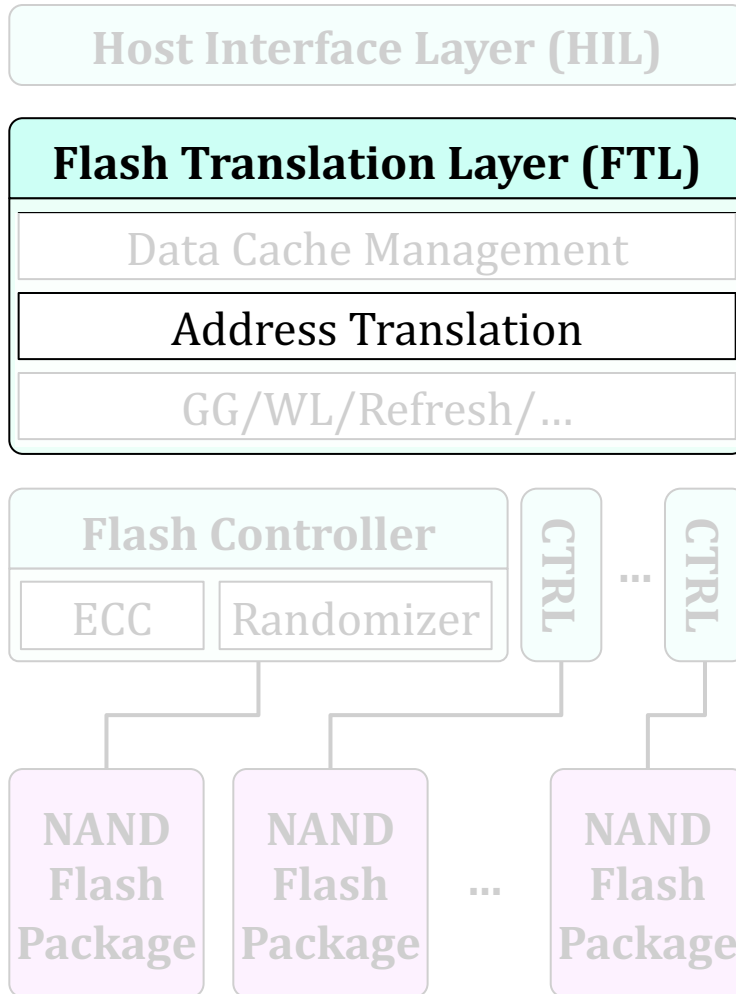
- Error-correcting codes (ECC)
  - Can detect/correct errors: e.g., 72 bits/1 KiB error-correction capability
  - Stores additional parity information together with raw data
- Randomizer
  - Scrambling data to write
  - To avoid worst-case data patterns that can lead to significant errors
- Issue NAND flash commands

# Request Handling: Read



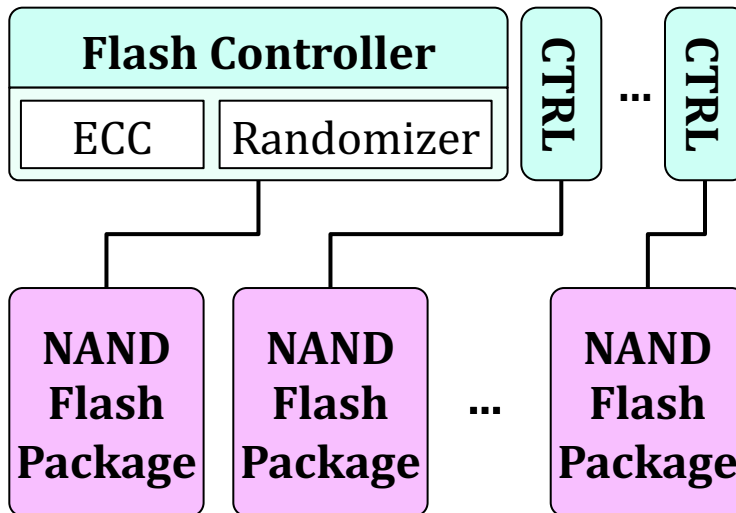
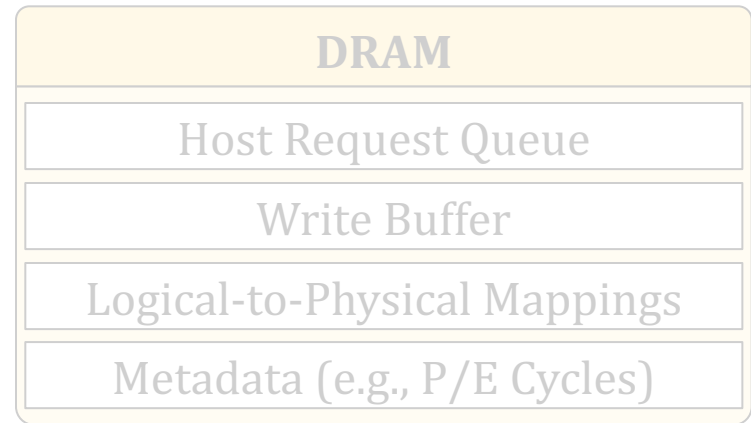
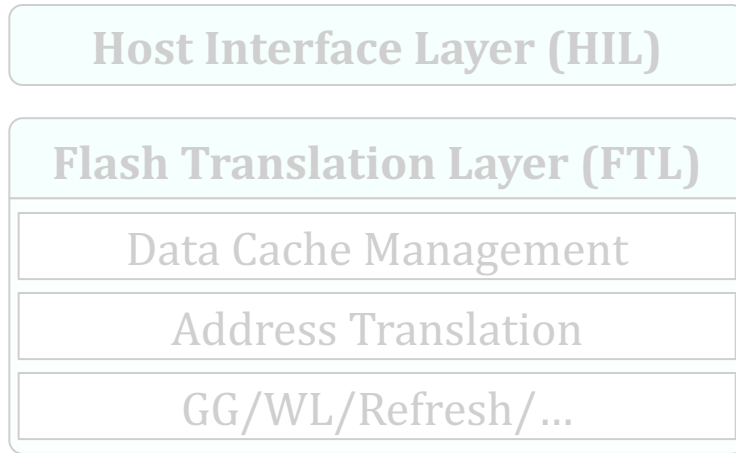
- First checks if the request data exists in the write buffer
  - If so, returns the corresponding request immediately with the data
- A host read request can be involved with several pages
  - Such a request can be returned only after all the requested data is ready

# Request Handling: Read



- Finds the PPA where the request data is stored from the L2P mapping table

# Request Handling: Read



- First reads the raw data from the flash chip
- Derandomizes the raw data
- Performs ECC decoding
- ECC decoding can fail
  - Retries reading of the page w/ adjusted  $V_{REF}$
  - Soft-decision ECC (LDPC)

# MQSim Organization

---

- Repository

- <https://github.com/CMU-SAFARI/MQSim>

- MQSim/src/

- host

Host interface models (NVMe and SATA)

- nvm\_chip

Storage memory device models (NAND flash chips)

- sim

Simulation engine

- ssd

SSD component models

- utils

- main.cpp

# MQSim Organization

---

- MQSim/src/ssd/
  - Host\_Interface\_\*.cpp
    - Host Interface Layer
  - Data\_Cache\_Manager\_\*.cpp
    - Data Cache Management
  - FTL.cpp
    - Address\_Mapping\_Unit\_\*.cpp
    - GC\_and\_WL\_Unit\_\*.cpp
    - TSU\_\*.cpp (Transaction Scheduling Unit)
  - NVM\_PHY\_\*.cpp
    - Interface w/ storage memory devices

# MQSim Organization

---

- MQSim/nvm\_chip/
  - flash\_memory/
    - Flash\_Chip.cpp
    - Plan.cpp
    - Block.cpp
    - Page.h
    - ...
  - NVM\_Chip.h

# MQSim Organization

---

- MQSim/src/sim
  - Engine.cpp
  - MQSim is an event-driven simulator
    - Events from a trace / event generator
    - A host request leads to many subevents depending on the current SSD state
      - Data cache management
      - Address translation
      - Transaction scheduling
      - NAND flash operations
    - Simulation process
      - Builds an event list (tree)
      - Collects statistics from the current state
      - Removes the first event from the list and Handles it
      - Set times to the time of this event



# First Step: Refactoring SIM Engine

---

- Go through files in `MQSim/src/sim/`
  - To figure out how classes are related to each other
  - What the simulation engine does
- Go through other files in `MQSim/src/ssd`
  - To figure out how SSD components interact with the simulation engine
- Improve coding convention
  - Mixed convention: hard to reuse the code
    - `RemoveObject` vs. `Start_simulation`  
→ `Remove_object` or `StartSimulation`
  - Suggestion: Google C++ Style Guide
    - <https://google.github.io/styleguide/cppguide.html>

# Required Material

---

- Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu,  
[“Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery,”](#)  
Invited Book Chapter in Inside Solid State Drives, 2018  
- Introduction and Section 1

# Recommended Material

---

- Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie Kim, Yixin Luo, Yaohua Wang, Nika Mansouri Ghiasi, Lois Orosa, Juan Gómez Luna, and Onur Mutlu, [“FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives,”](#) In ISCA, 2018

# Next Meetings

---

- We will provide background on NAND flash memory
- We will discuss your progress in last week
  - Please contact us whenever you have any questions

# P&S Modern SSDs

Understanding and Designing  
Modern NAND Flash-Based Solid-State Drives

Dr. Jisung Park

Prof. Onur Mutlu

ETH Zürich

Spring 2021

24 March 2021