Design Tradeoffs for SSD Performance

Agrawal, Nitin, Ted Wobber, John D. Davis, Mark Manasse, Rina Panigrahy

Microsoft Research, Silicon Valley University of Wisconsin-Madison

2008 USENIX Annual Technical Conference

2024. 07. 17

Presentation by Kim MinSeong, Wee DaYeon

kms0509@dankook.ac.kr, wida10@dankook.ac.kr

Contents

Ref: WD 2.5인치 벨로시랩터 WD3000BLFS,
https://www.javatpoint.com/ssd

1. Introduction

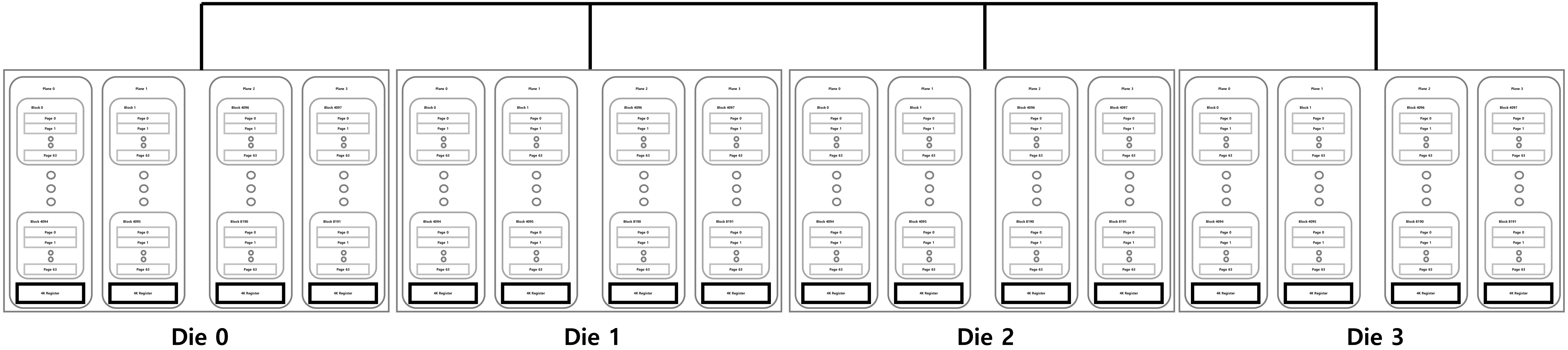
- **Hard disk drive VS Solid State Drive**

- Cheaper cost per byte
- Higher storage capacity option
- Lifetime

- Exceptional bandwidth
- Random I/O performance
- Saving power budget
- Durability

1. Introduction

- **Data placement:** How will the data be stored in which chip on SSD?
  - Provide Load-balancing
  - Consider Wear-leveling
- **Parallelism:** Flash components work in parallel for optimal performance

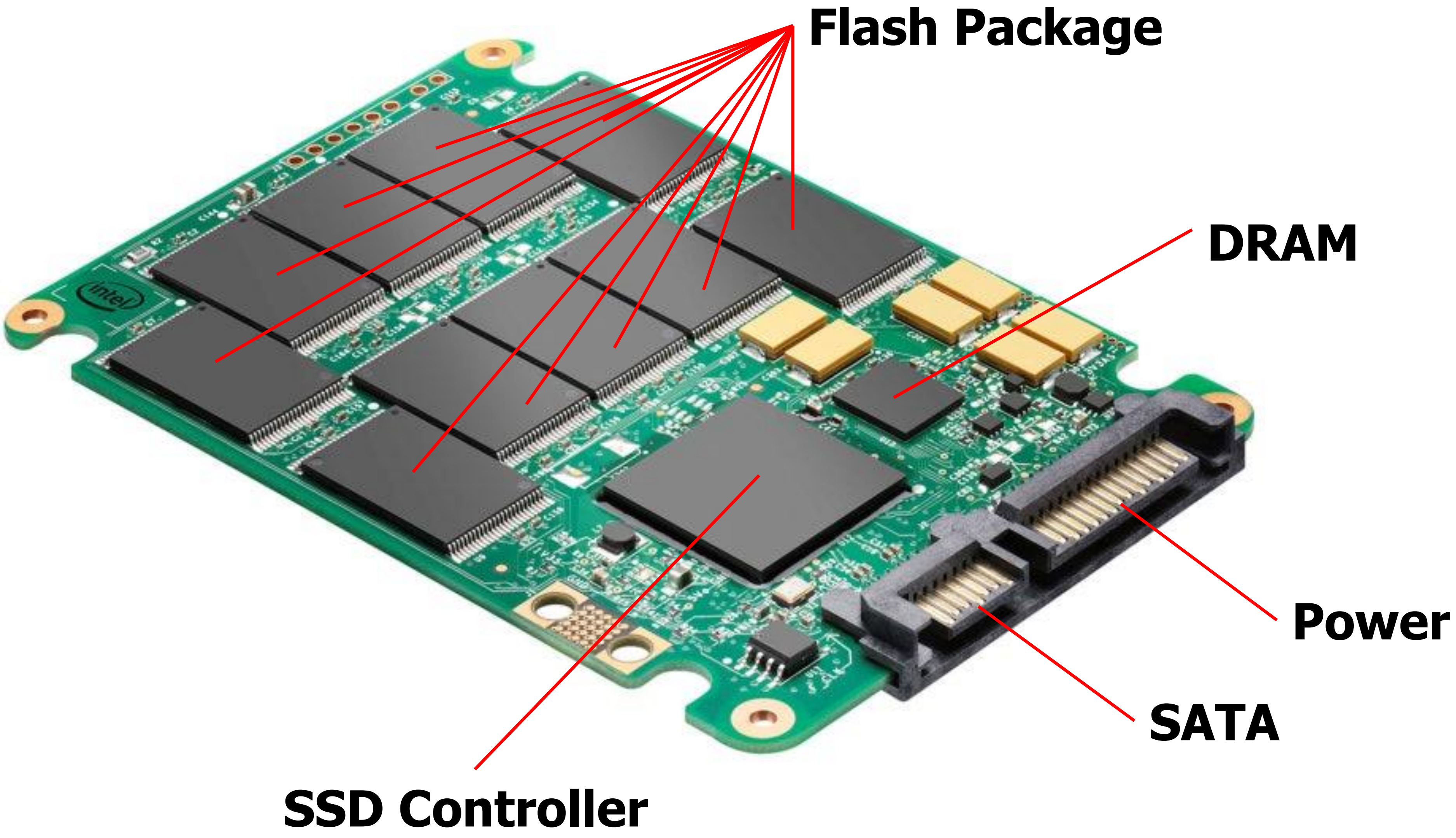


1. Introduction

- **Write ordering: Small random writes are tricky**
  - Properties of NAND flash
    - Page-Level Read/Write
    - Block-Level Erase
    - Write Amplification

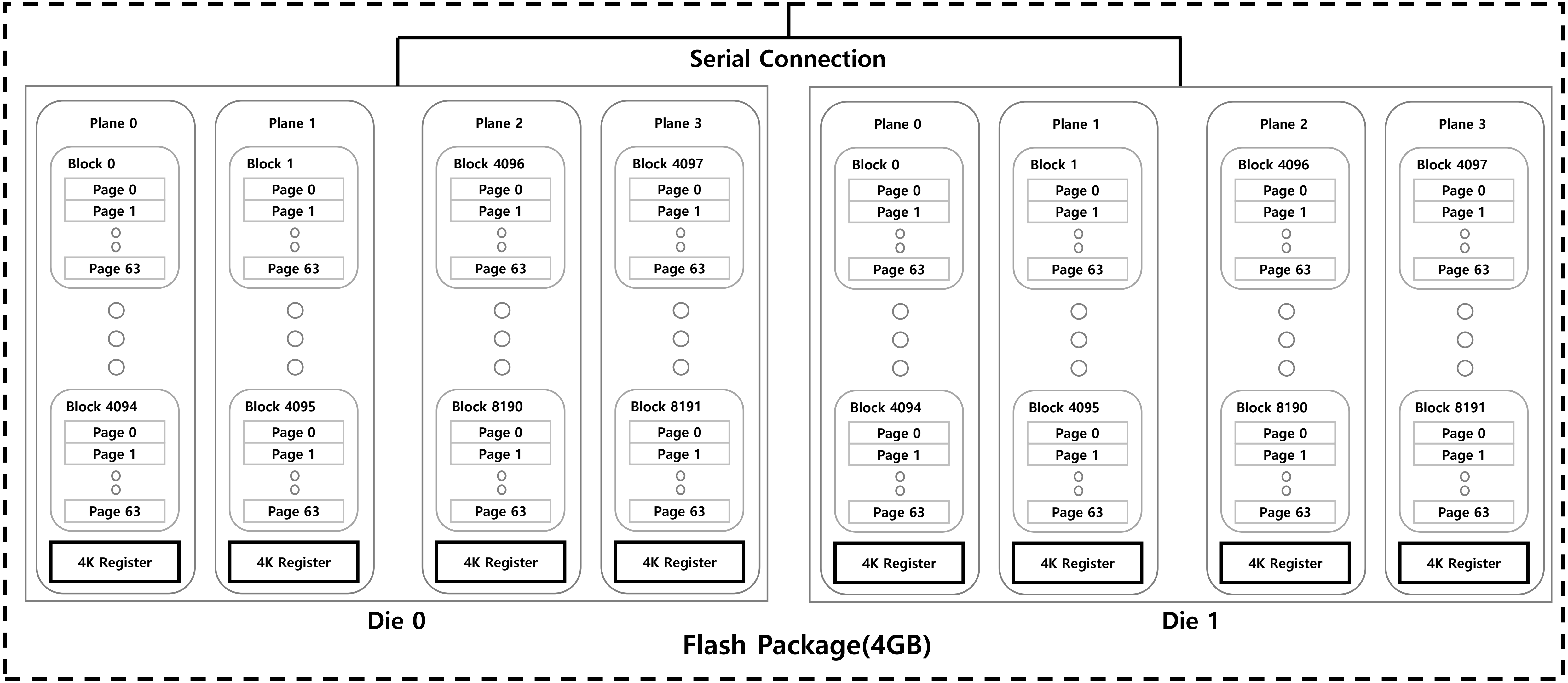
- **Workload management: Performance depends on workload**
  - Design choice
    - Sequential workload ↔ Random workload

2. Background

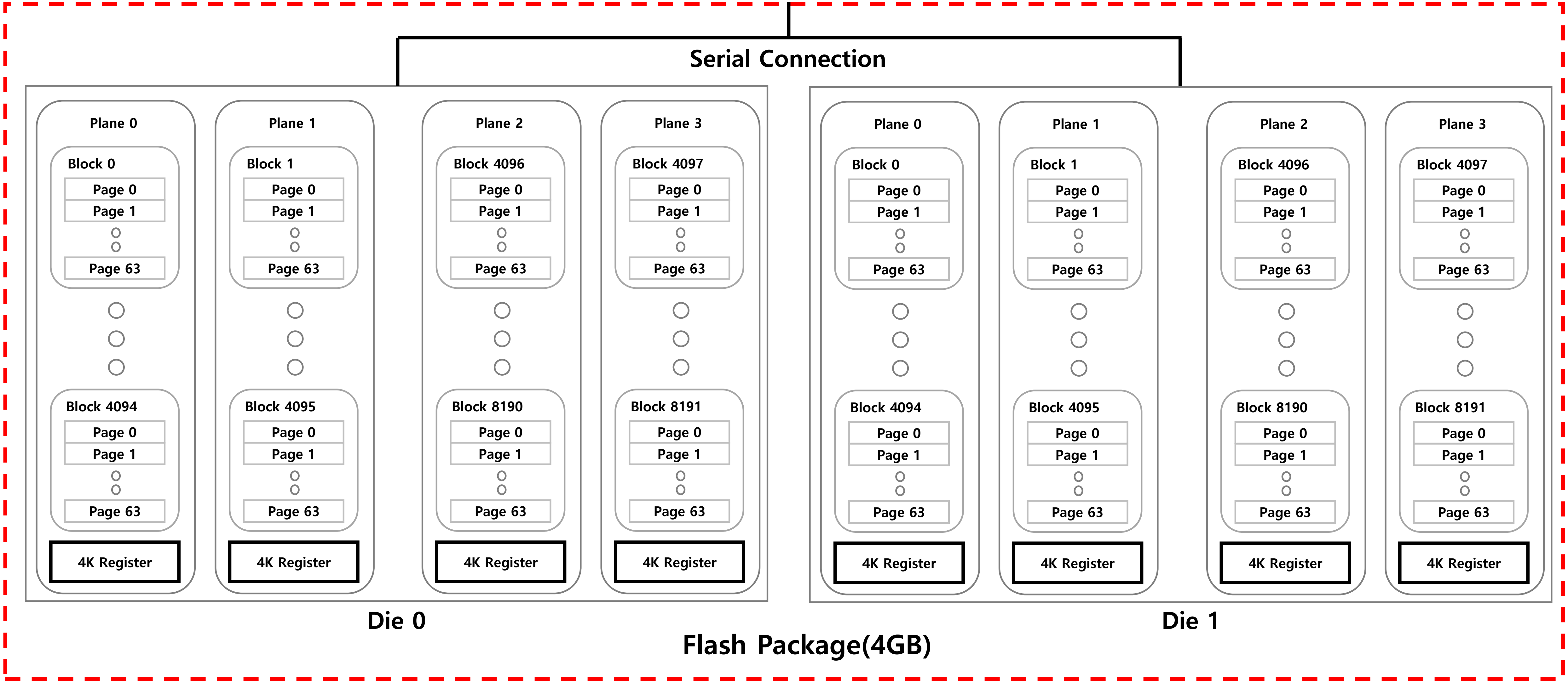


Ref: https://www.javatpoint.com/ssd

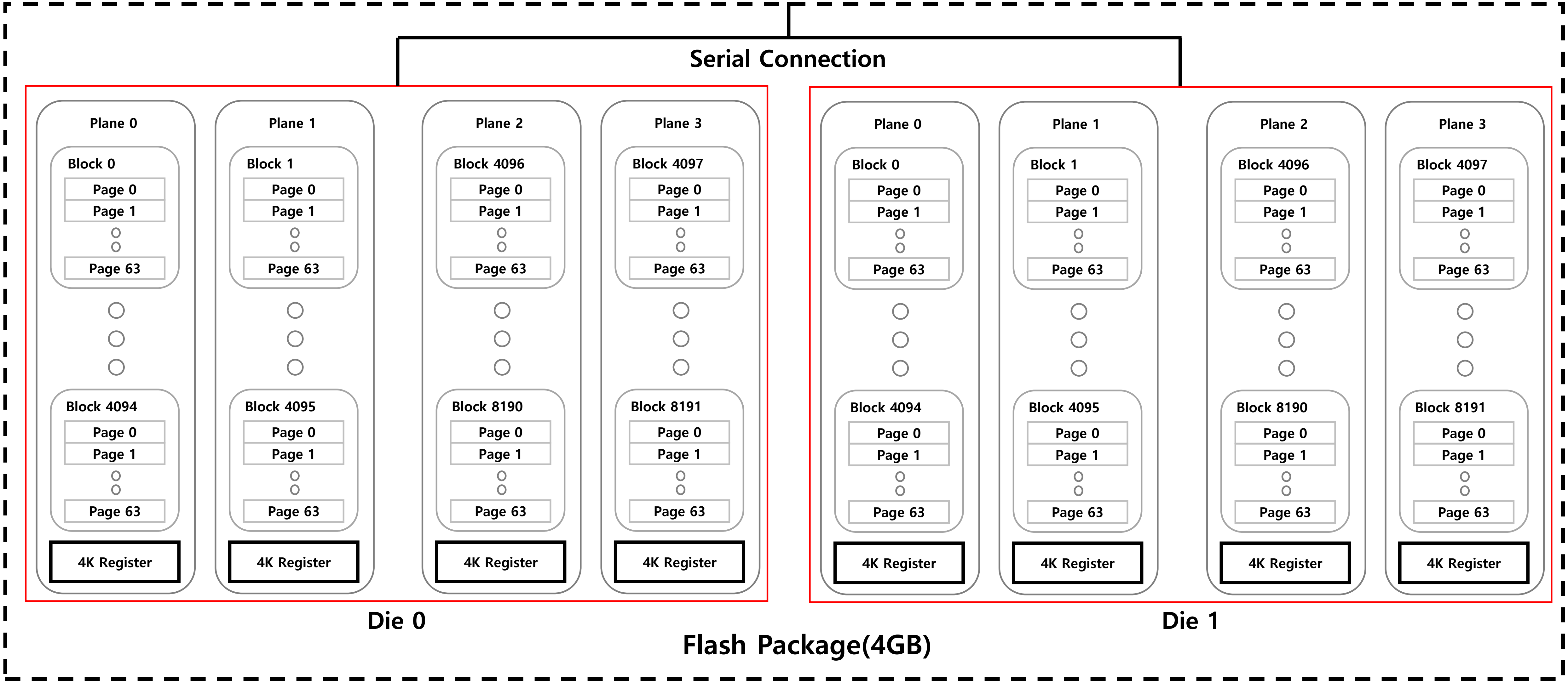
2. Background



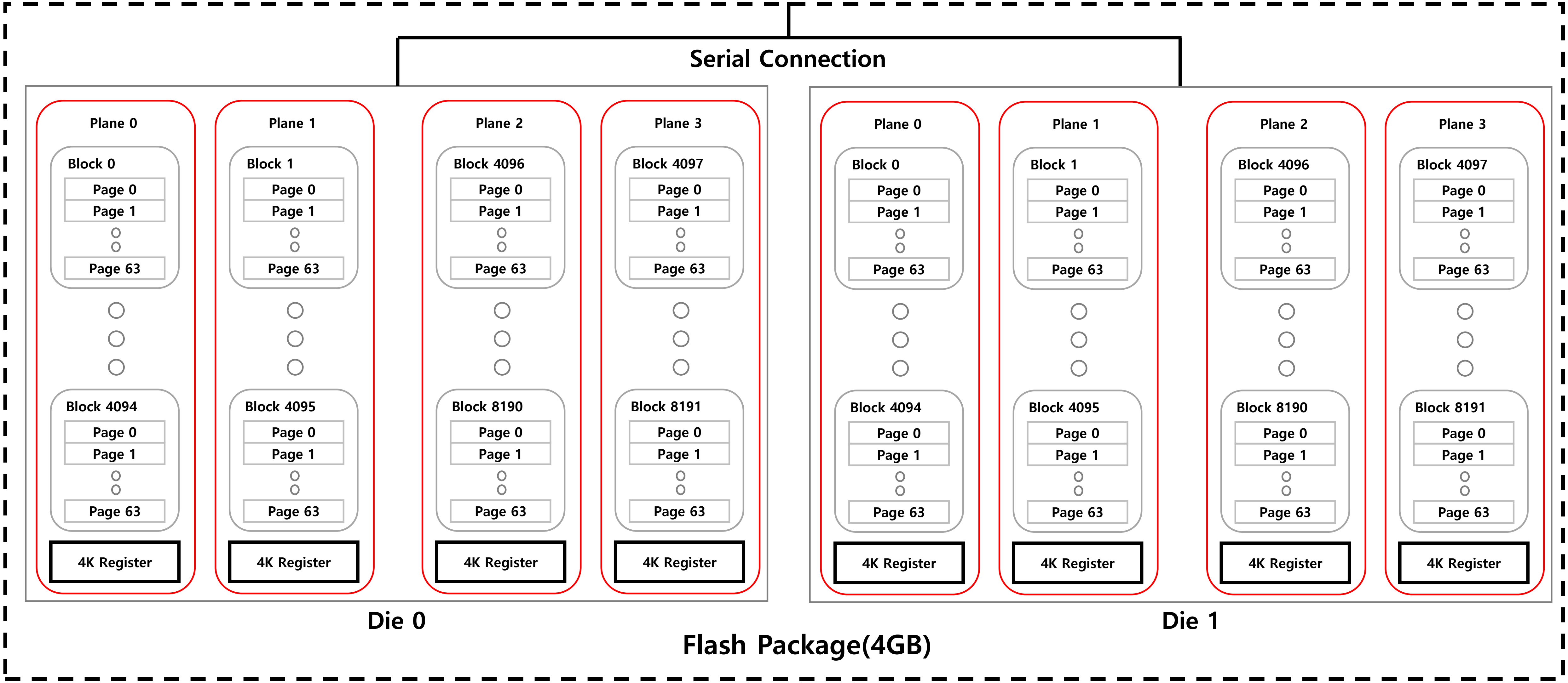
2. Background



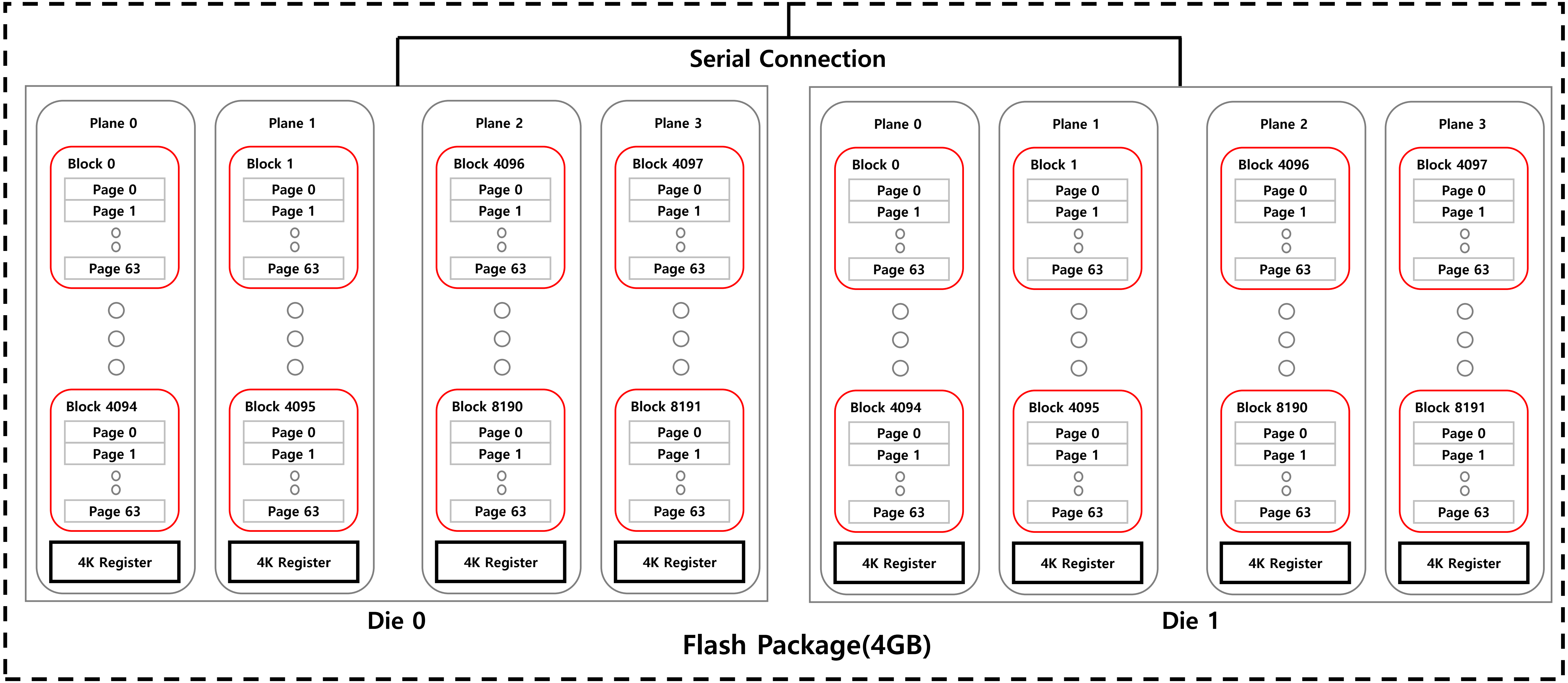
2. Background



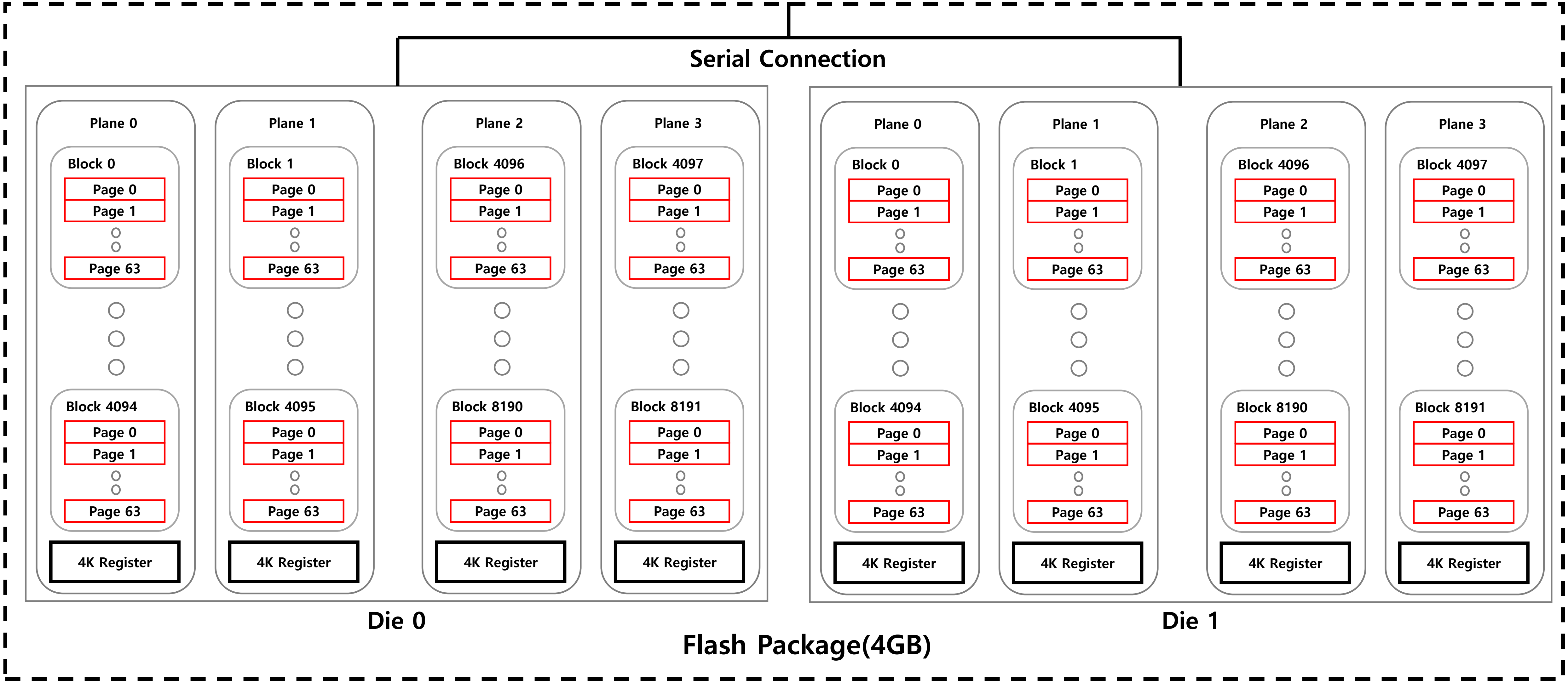
2. Background



2. Background



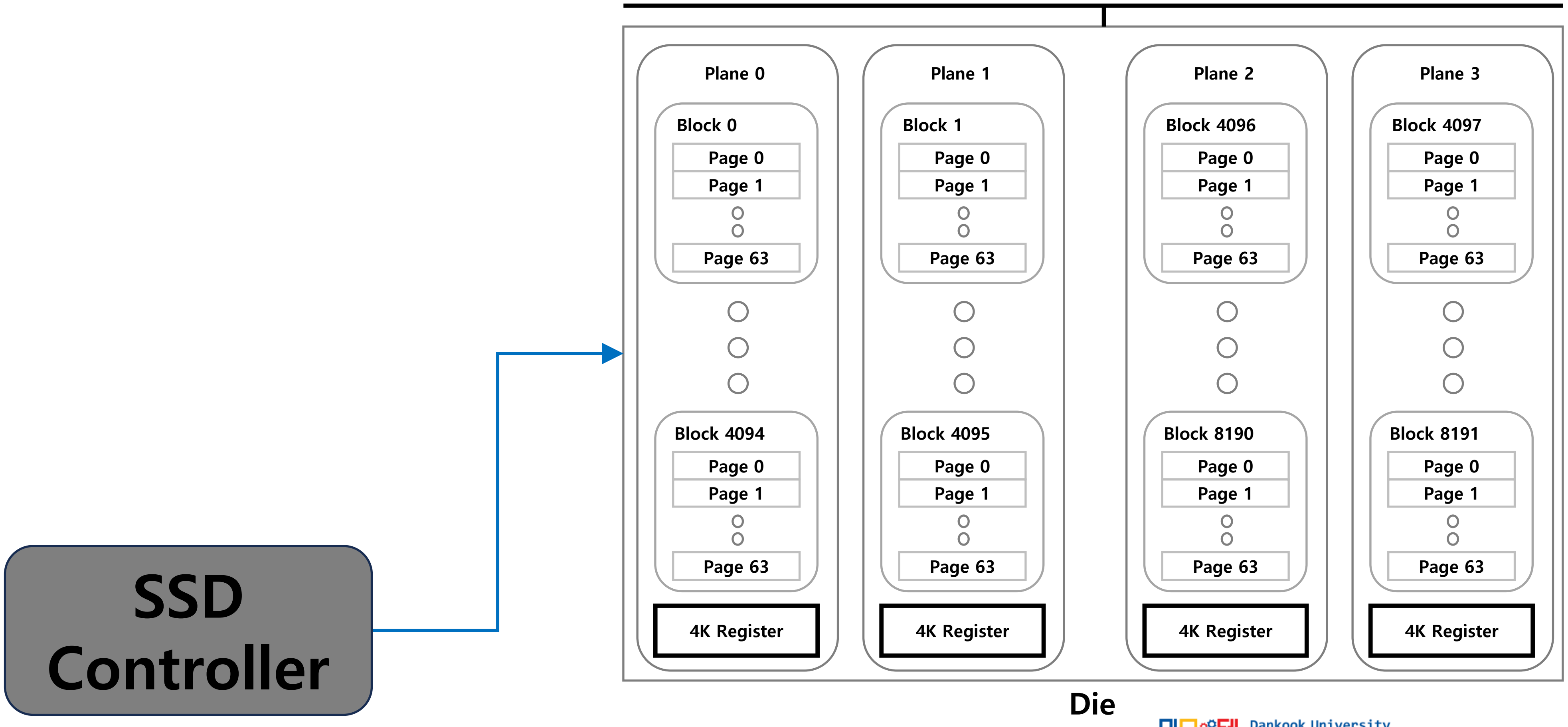
2. Background



2. Background

- Serial Read

Page Read to Register	25µs
Page Program(Write) from Register	200µs
Block Erase	1.5ms
Serial Access to Register(Data Bus)	100µs



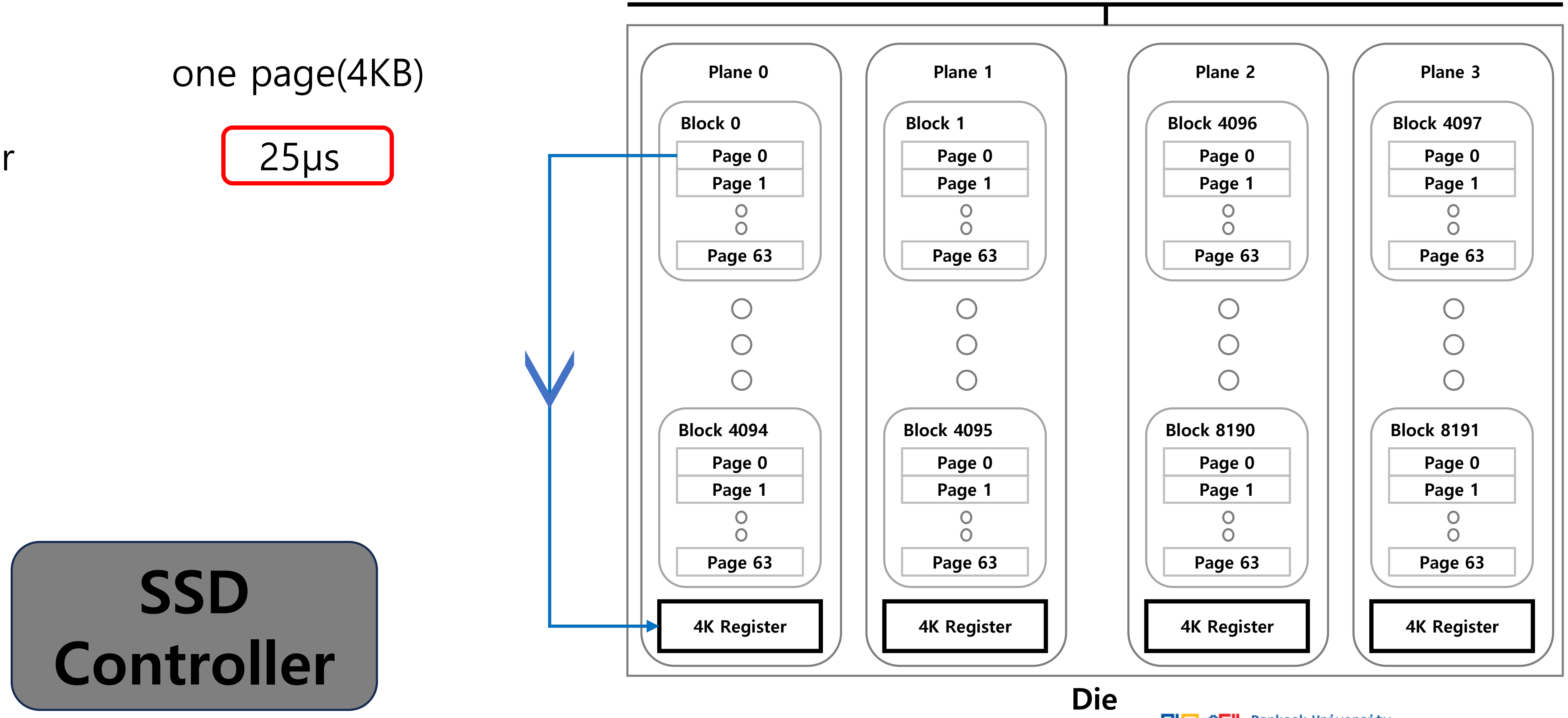
2. Background

- Serial Read

Page Read to Register	25µs
Page Program(Write) from Register	200µs
Block Erase	1.5ms
Serial Access to Register(Data Bus)	100µs

one page(4KB)

1. Page → Register 25µs



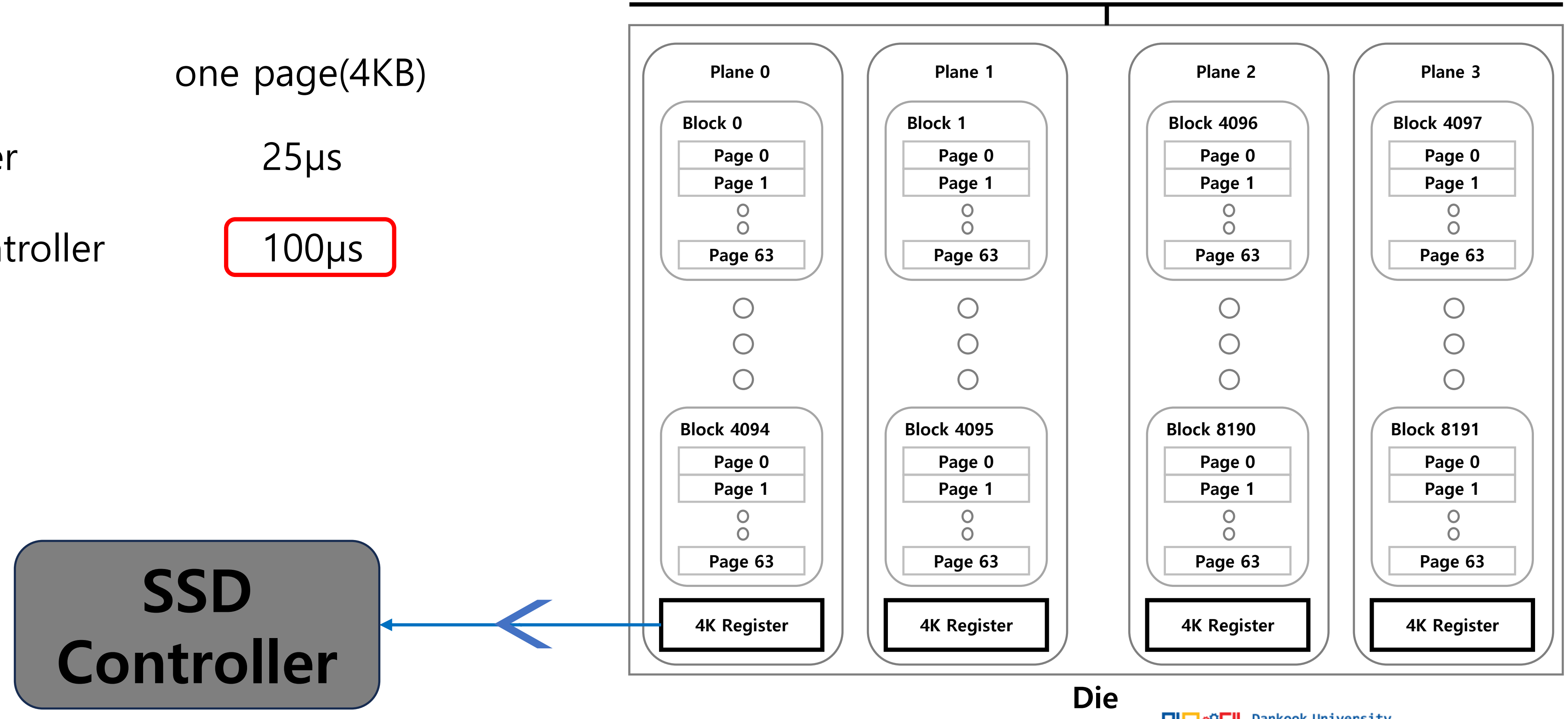
2. Background

Operation	Time
Page Read to Register	25μs
Page Program(Write) from Register	200μs
Block Erase	1.5ms
Serial Access to Register(Data Bus)	$\boxed{100μs}$

- Serial Read

one page(4KB)

1. Page → Register 25μs
2. Register → Controller $\boxed{100μs}$



2. Background

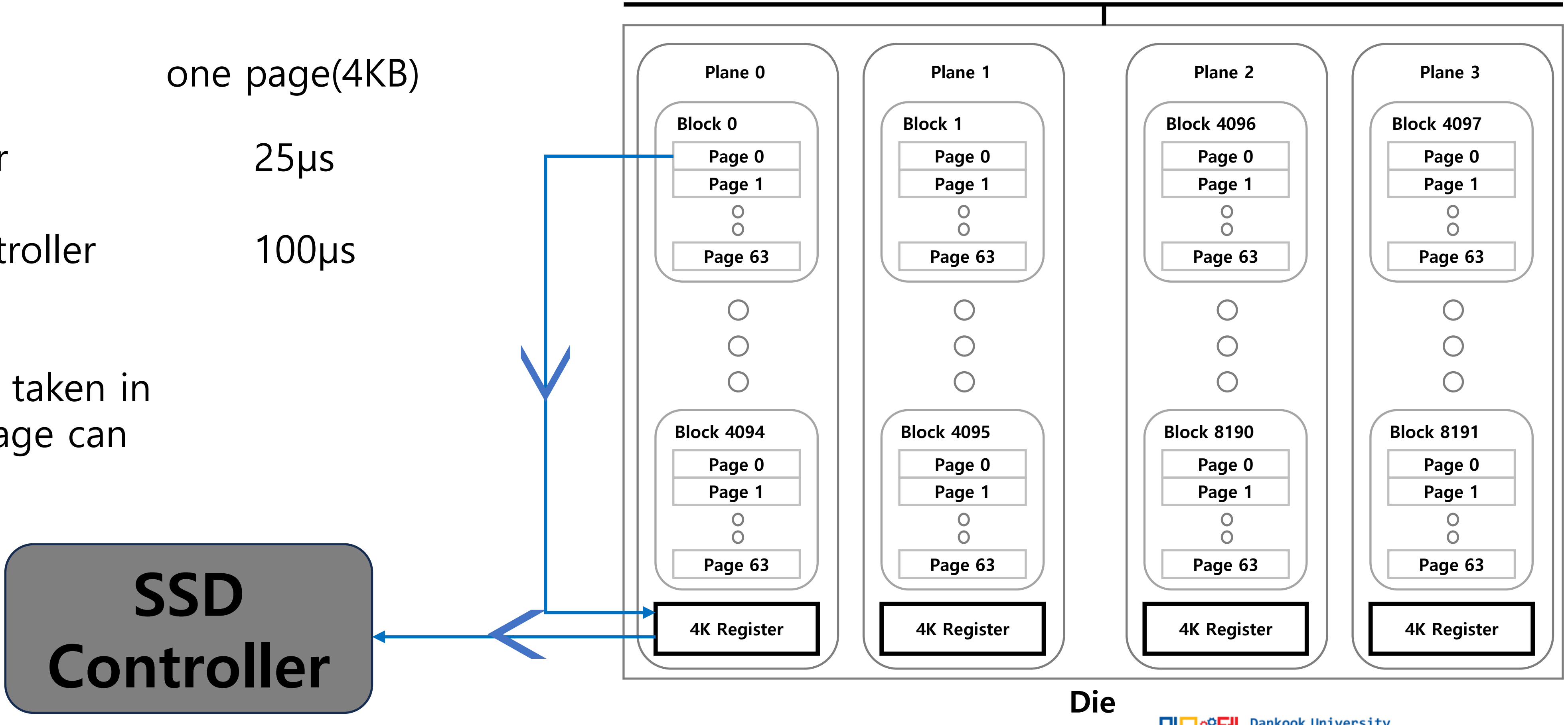
- Serial Read

Page Read to Register	25μs
Page Program(Write) from Register	200μs
Block Erase	1.5ms
Serial Access to Register(Data Bus)	100μs

one page(4KB)

1. Page → Register 25μs
2. Register → Controller 100μs

Two operations are taken in series, a flash package can produce 32MB/s



2. Background

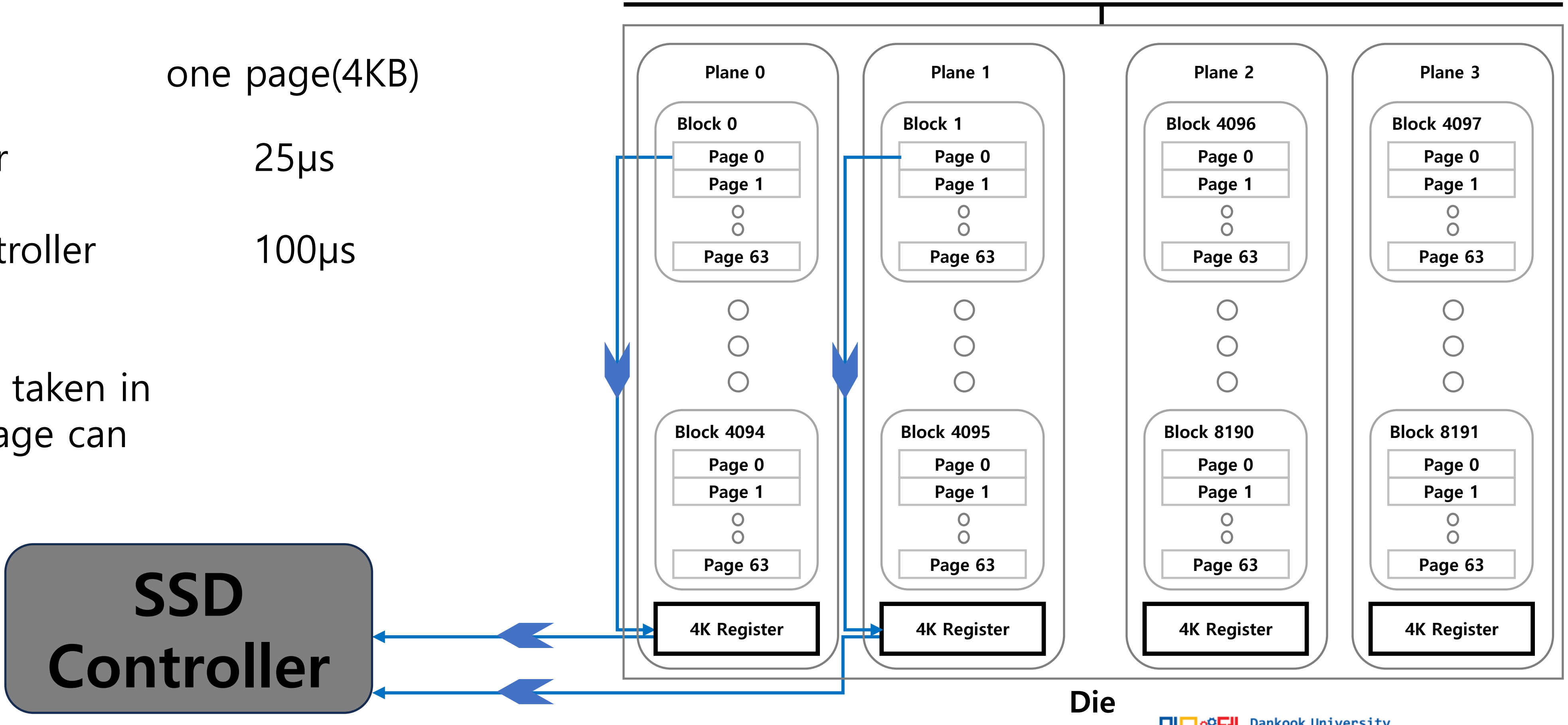
- Interleaving(Die) Serial Read

Page Read to Register	25μs
Page Program(Write) from Register	200μs
Block Erase	1.5ms
Serial Access to Register(Data Bus)	100μs

one page(4KB)

1. Page → Register 25μs
2. Register → Controller 100μs

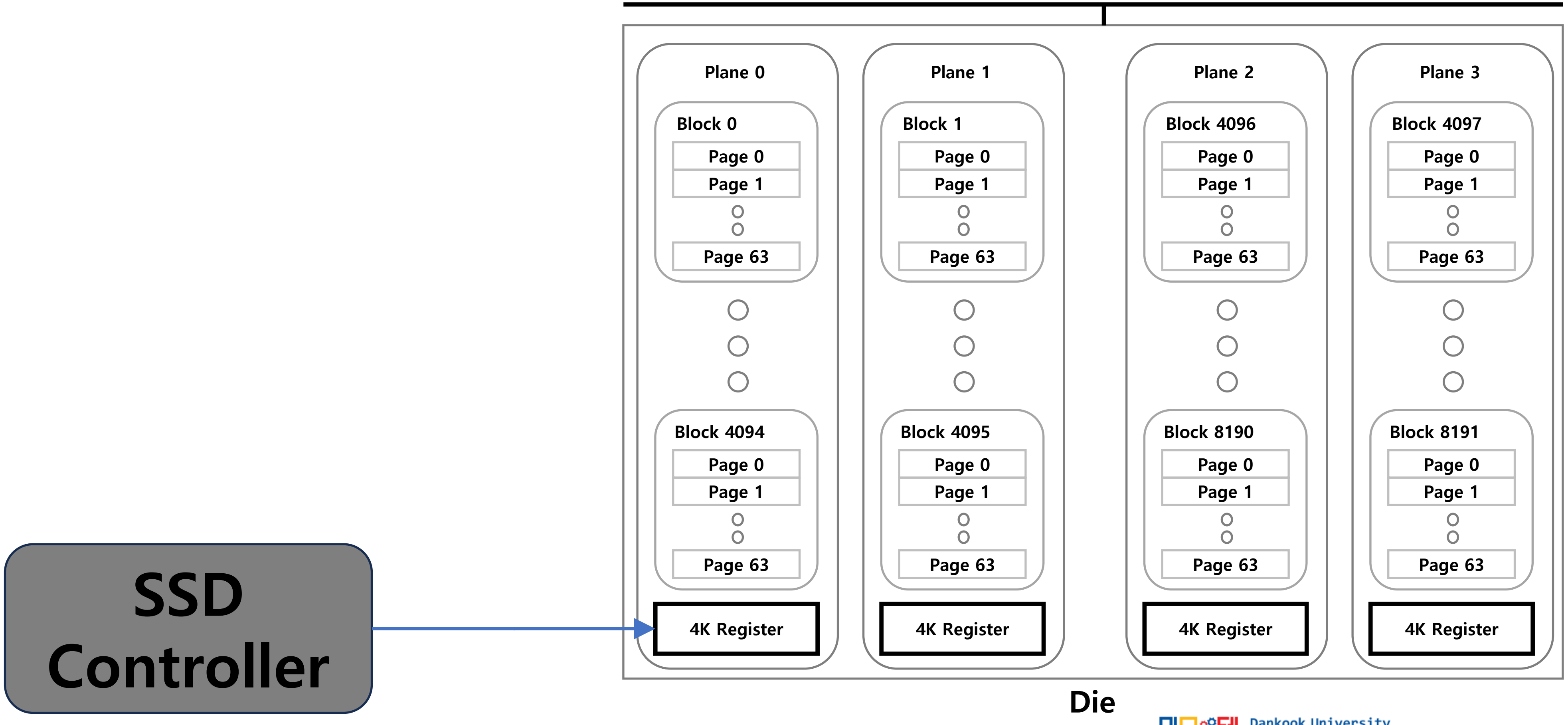
Two operations are taken in series, a flash package can produce 40MB/s



2. Background

- Serial Write

Page Read to Register	25μs
Page Program(Write) from Register	200μs
Block Erase	1.5ms
Serial Access to Register(Data Bus)	100μs



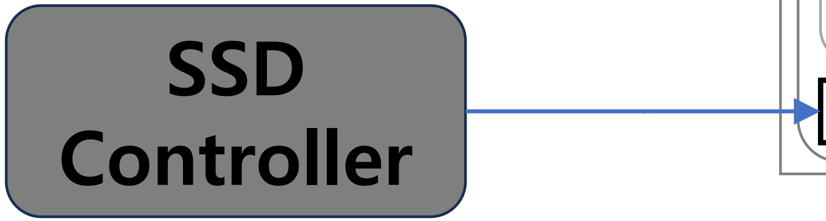
2. Background

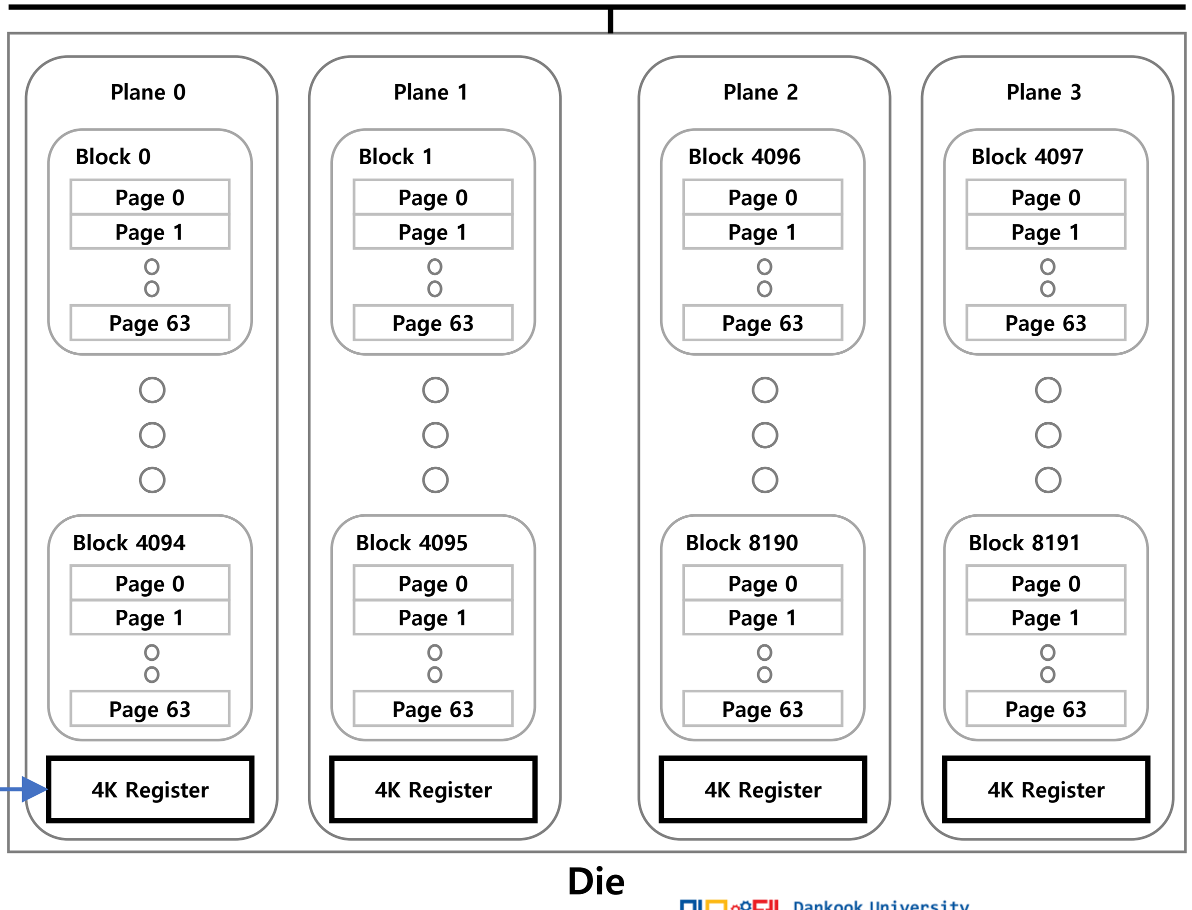
- Serial Write

Page Read to Register	25µs
Page Program(Write) from Register	200µs
Block Erase	1.5ms
Serial Access to Register(Data Bus)	100µs

one page(4KB)

1. Controller → Register 100µs





2. Background

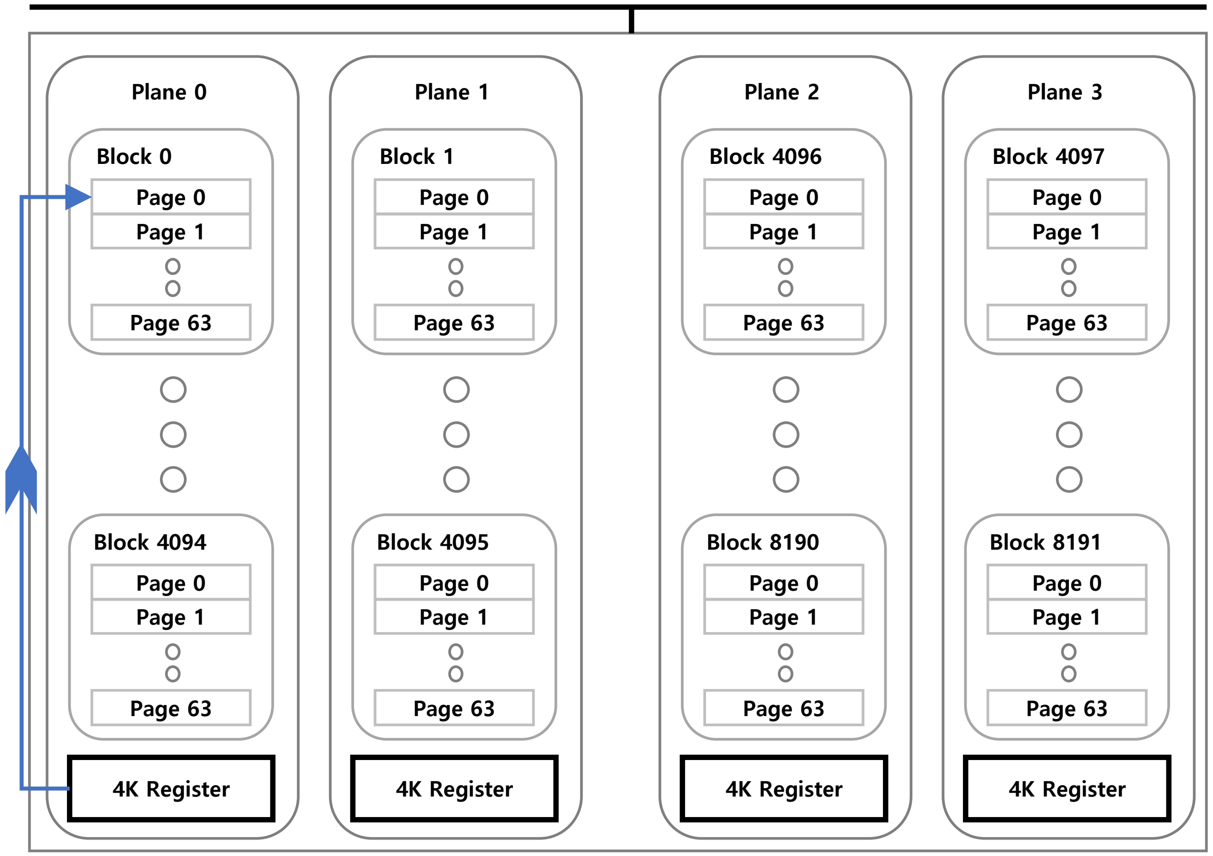
Page Read to Register	25μs
Page Program(Write) from Register	200μs
Block Erase	1.5ms
Serial Access to Register(Data Bus)	100μs

- Serial Write

one page(4KB)

1. Controller → Register 100μs
2. Register → Page 200μs

SSD Controller



Die

2. Background

Operation	Time
Page Read to Register	25μs
Page Program(Write) from Register	200μs
Block Erase	1.5ms
Serial Access to Register(Data Bus)	100μs

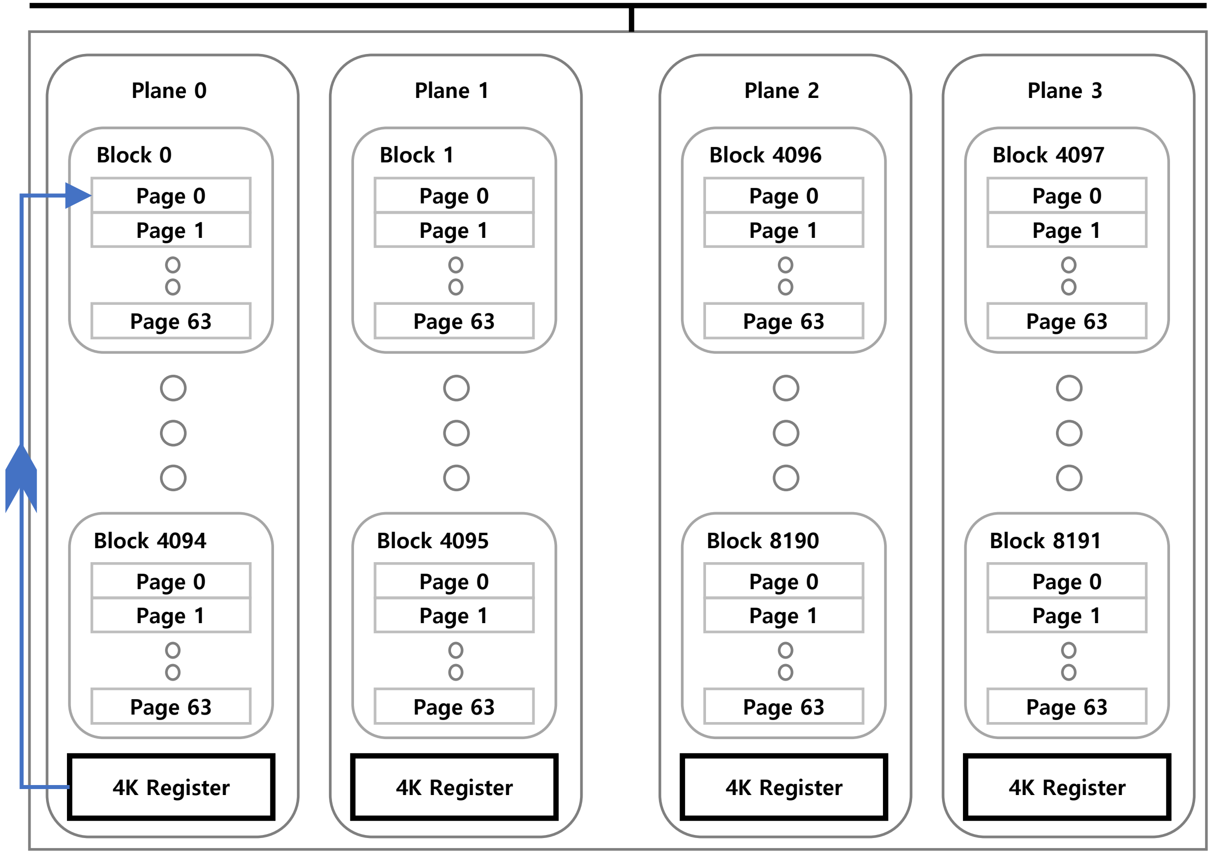
- Serial Write

one page(4KB)

1. Controller → Register 100μs
2. Register → Page 200μs

Two operations are taken in series, a flash package can produce 13MB/s

SSD Controller



Die

2. Background

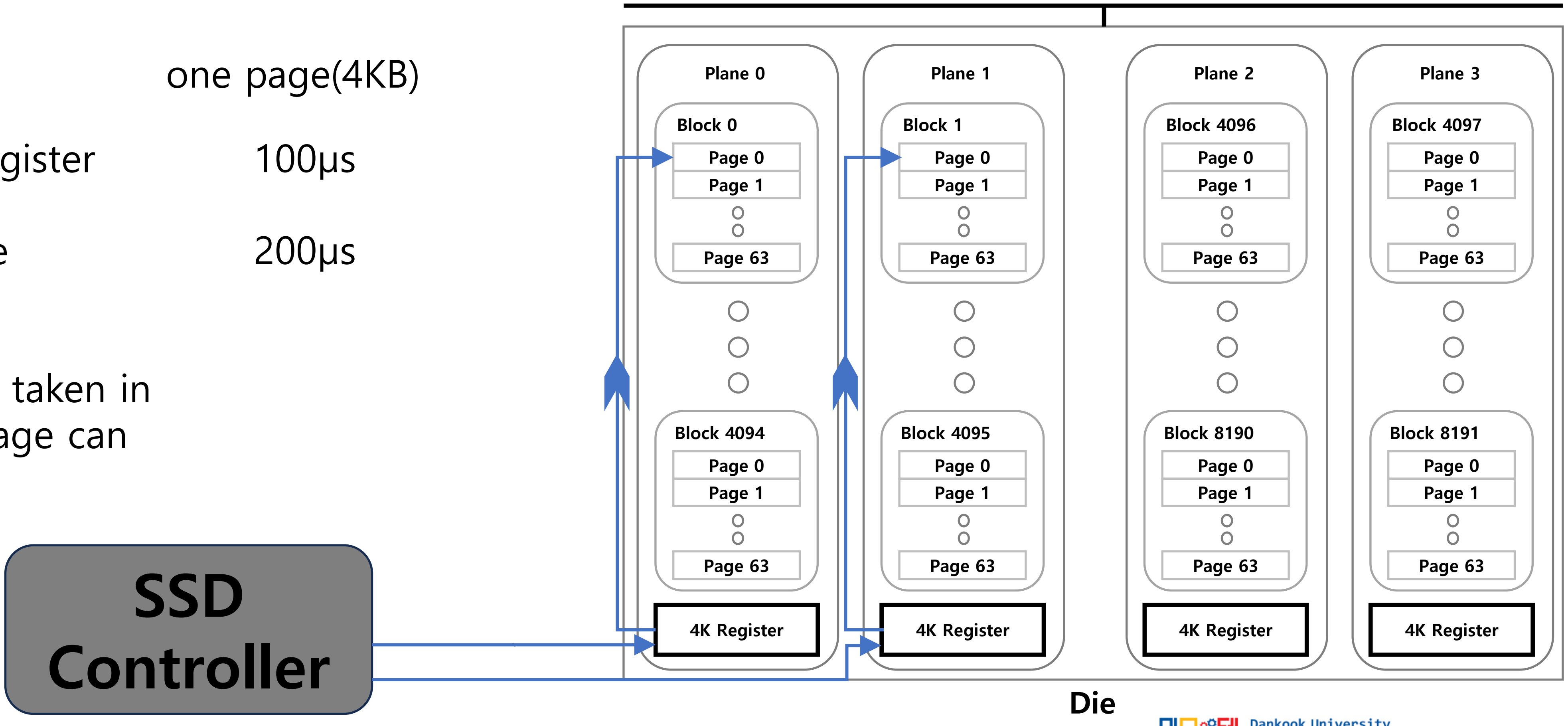
Page Read to Register	25µs
Page Program(Write) from Register	200µs
Block Erase	1.5ms
Serial Access to Register(Data Bus)	100µs

- Interleaving(Die) Serial Write

one page(4KB)

1. Controller → Register 100µs
2. Register → Page 200µs

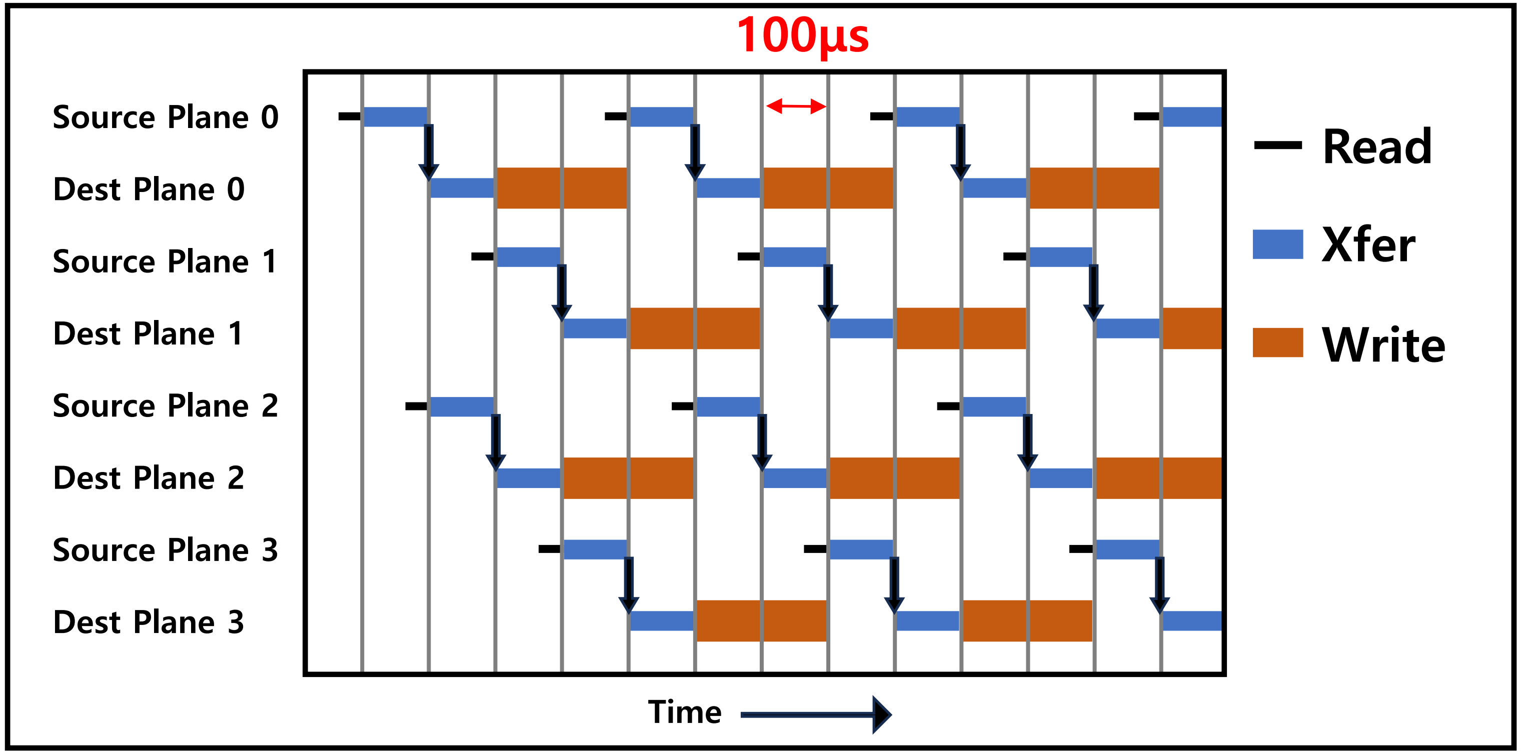
Two operations are taken in series, a flash package can produce 20MB/s



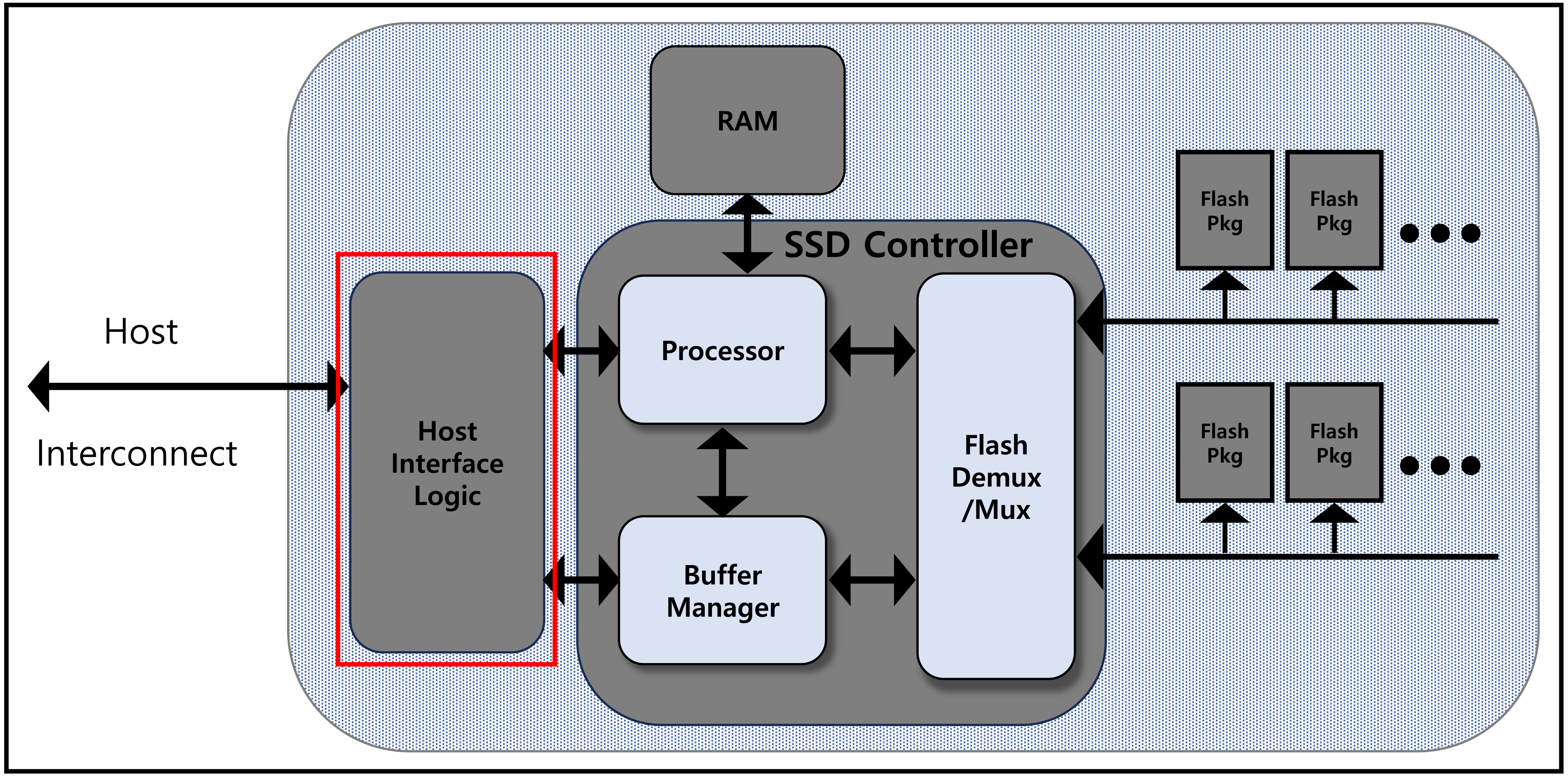
2. Background

- **Interleaving**
  - Proceed in parallel with other commands

- **Copy-Back**
  - Fast copy in same plane

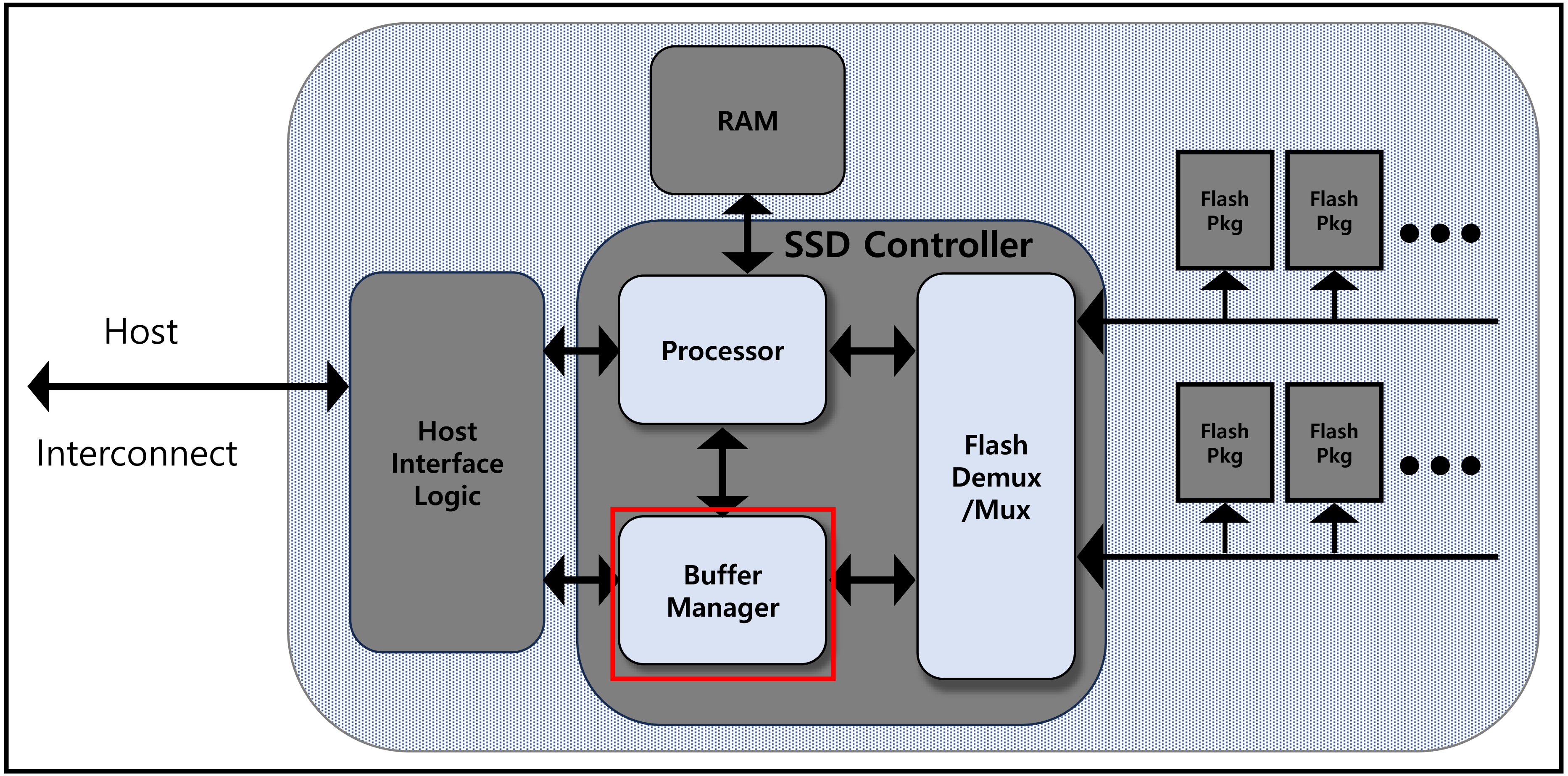


3. SSD Basics



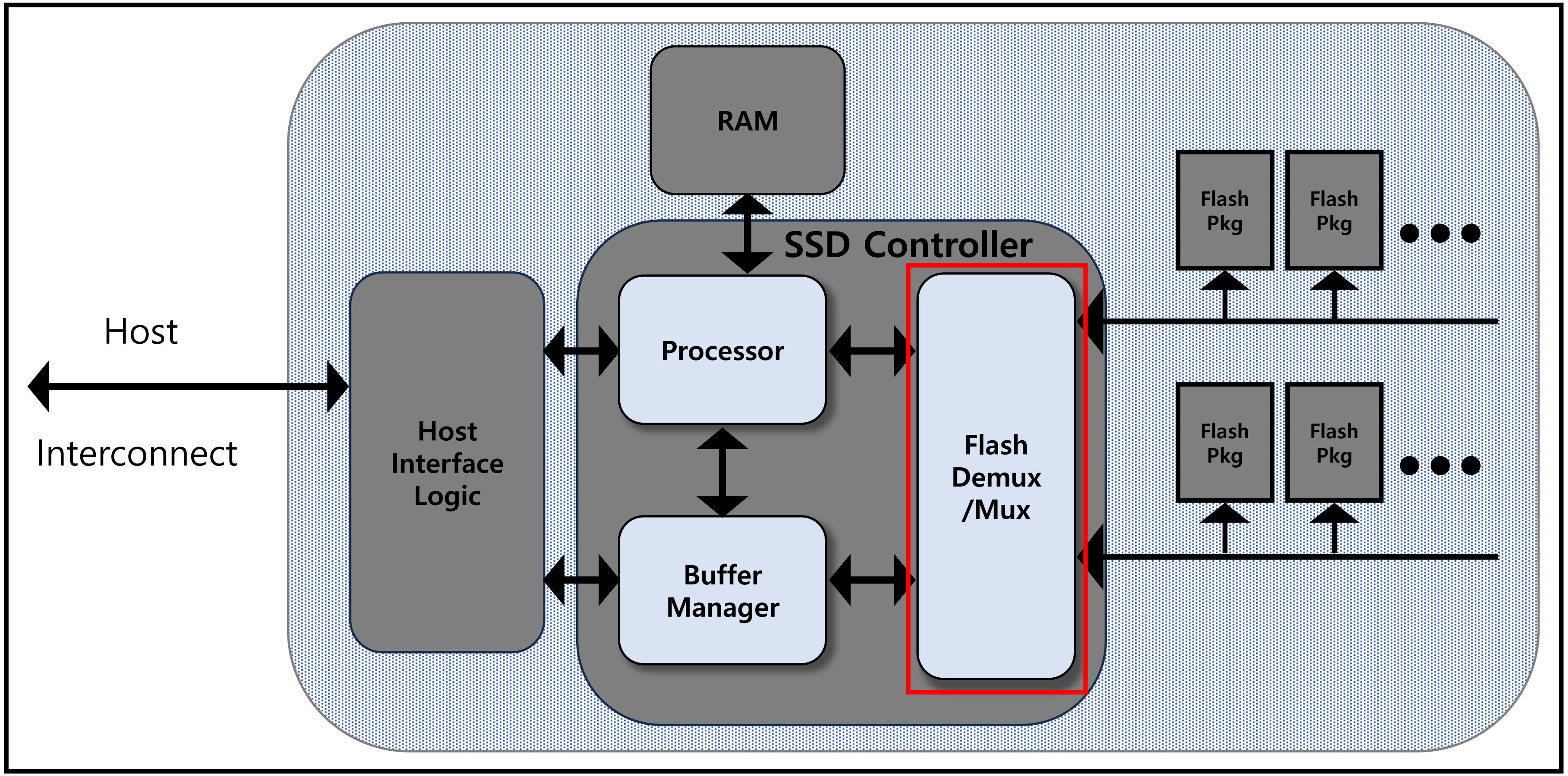
- **Physical host interface**
- **Logical disk emulation**

3. SSD Basics



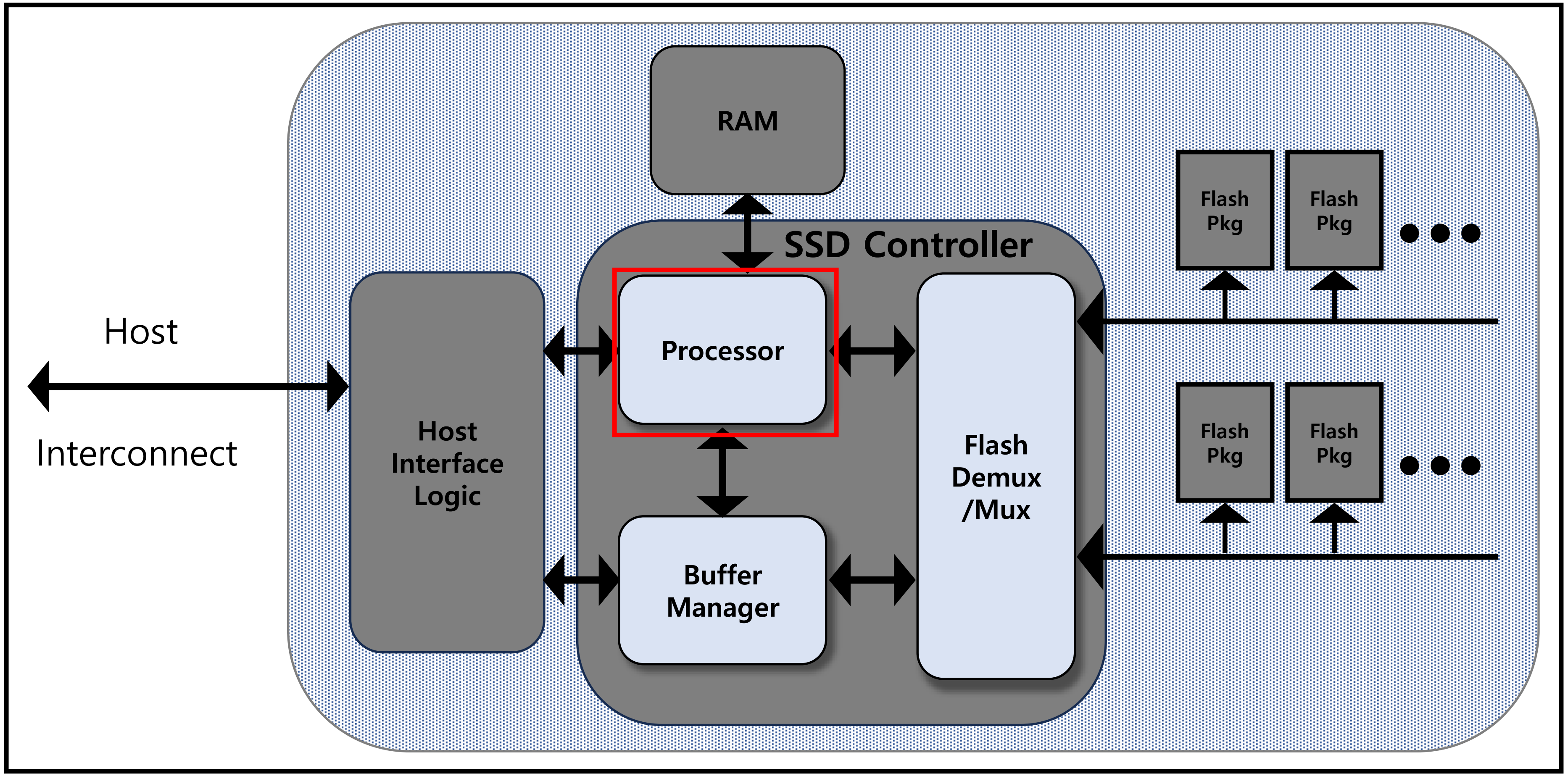
Holding requests

3. SSD Basics



Handling commands & transport

3. SSD Basics



Manage request flow & mapping

3.1 Logical Block Map

- **Allocation pool**
  - Static map
    - fixed mapping to a specific allocation pool
  - Dynamic map
    - Dynamic mapping to a specific allocation pool
  - Logical page size
    - Quarter-page(1KB) ~ Flash block(256KB)
  - Page span
    - Large size logical page may be related on different flash packages
    - Parallel accessible

3.1 Logical Block Map

- **Three constraints for variables**
  - Load balancing
    - I/O operations evenly balanced
  - Parallel access
    - Mapping to a parallel accessible structure
  - Block erasure
    - Write after Erase

3.1 Logical Block Map

- **Three constraints for variables**
  - Load balancing
    - I/O operations evenly balanced
  - Parallel access
    - Mapping to a parallel accessible structure
  - Block erasure
    - Write after Erase

Static mapping ↑ → Load balancing ↓

3.1 Logical Block Map

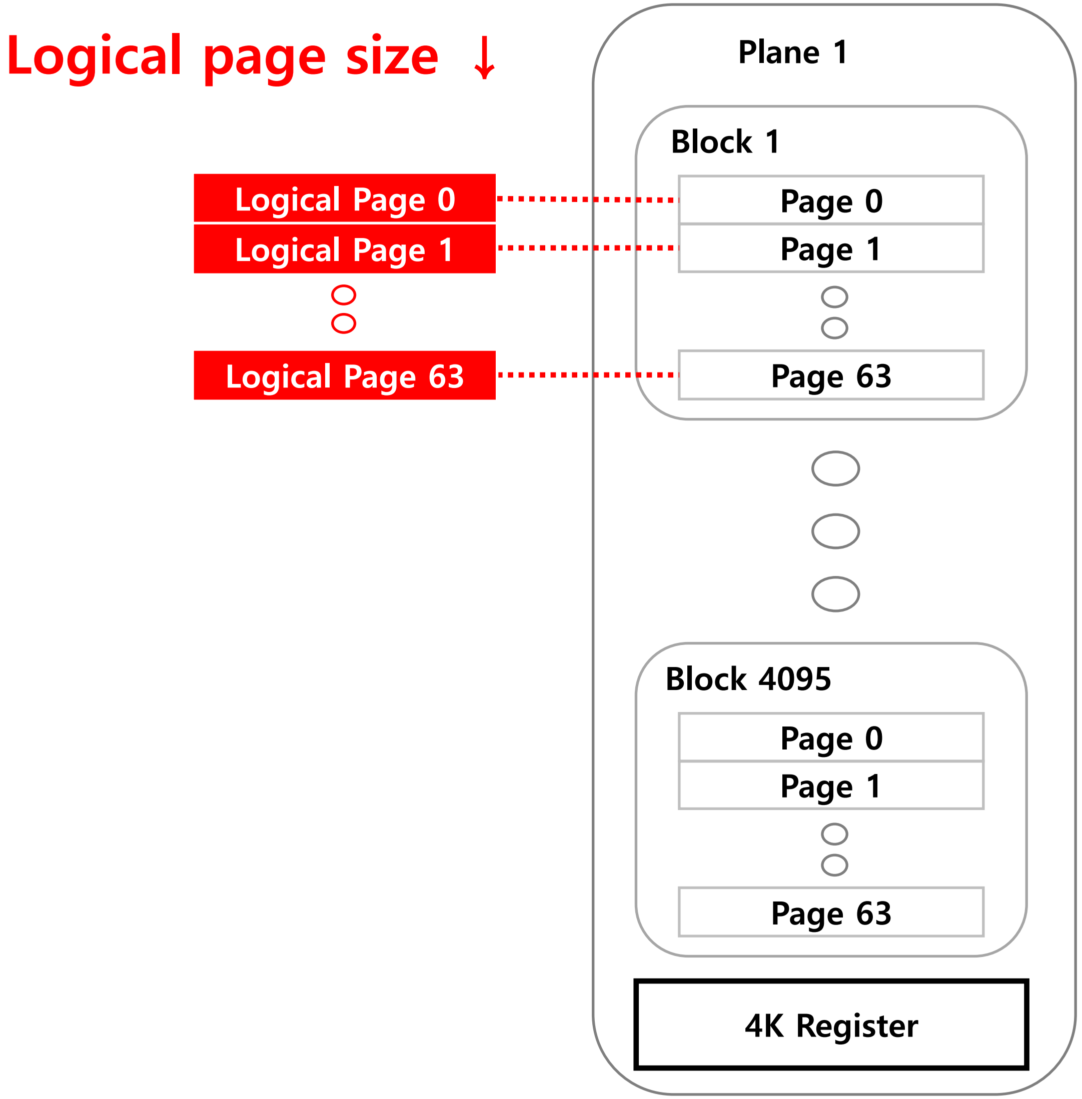
- **Three constraints for variables**
  - Load balancing
    - I/O operations evenly balanced
  - Parallel access
    - Mapping to a parallel accessible structure
  - Block erasure
    - Write after Erase

Static mapping ↑ → Load balancing ↓

Span in same block ↑ → Parallel access ↓

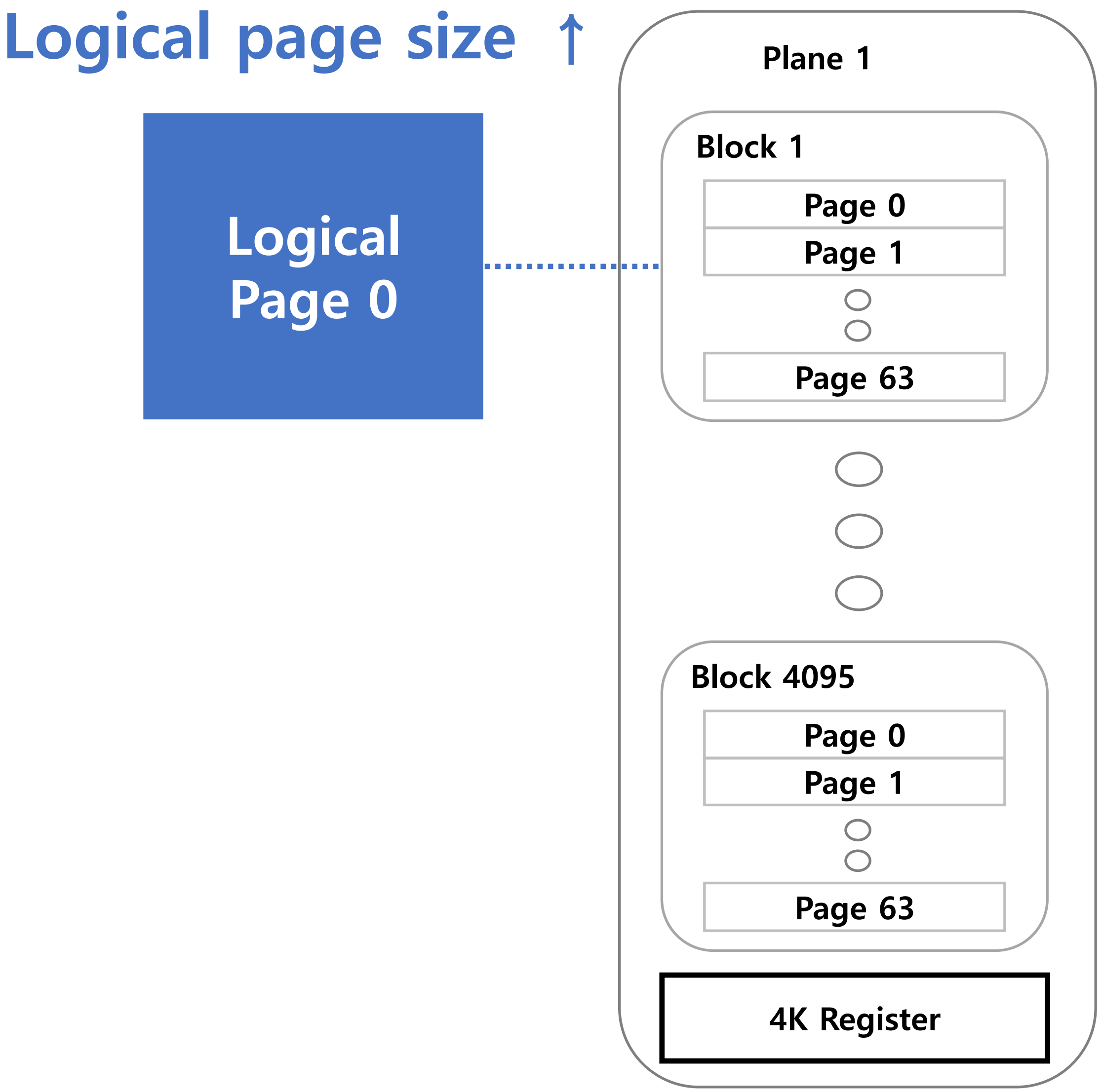
3.1 Logical Block Map

- **Three constraints for variables**
  - Load balancing
    - I/O operations evenly balanced
  - Parallel access
    - Mapping to a parallel accessible structure
  - Block erasure
    - Write after Erase



3.1 Logical Block Map

- **Three constraints for variables**
  - Load balancing
    - I/O operations evenly balanced
  - Parallel access
    - Mapping to a parallel accessible structure
  - Block erasure
    - Write after Erase

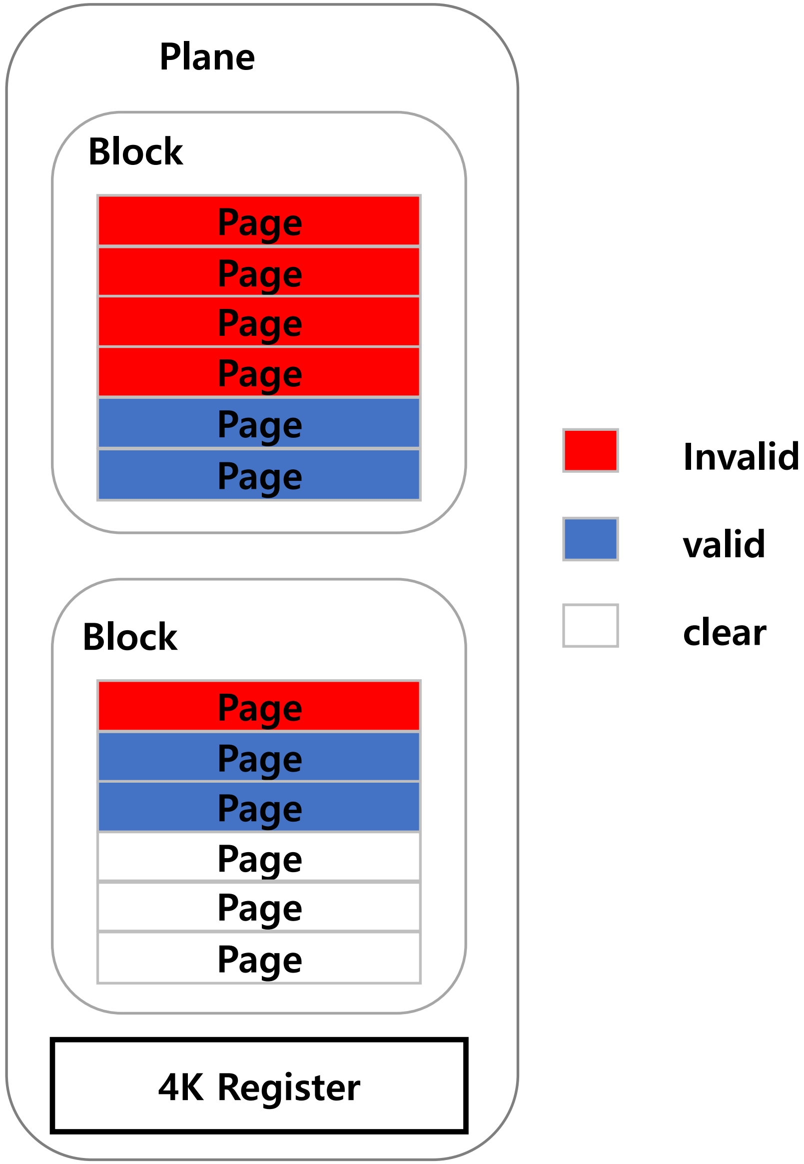


3.2 Cleaning

- **Allocation Pool**
  - Active block to assign to a write operation
  - To get Active block, we need Garbage Collection

3.2 Cleaning

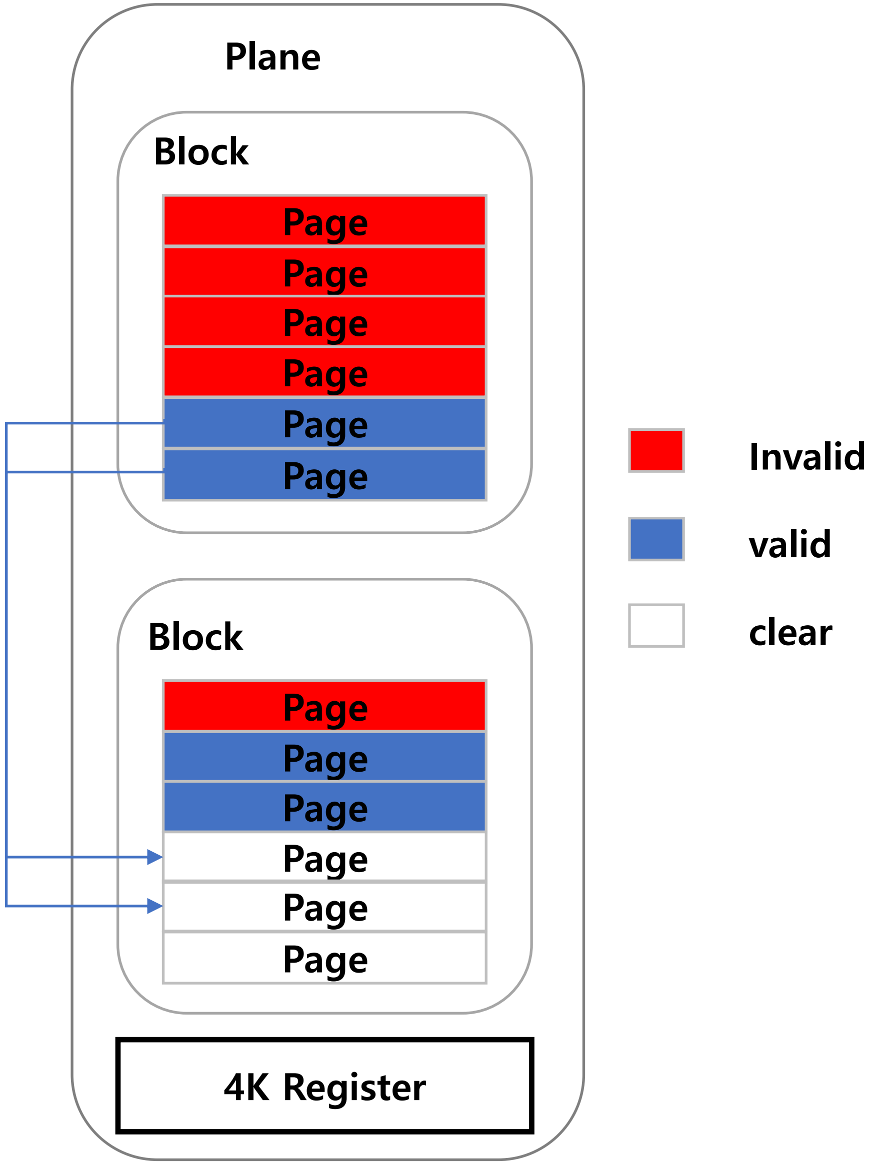
- **Allocation Pool**
  - Active block to assign to a write operation
  - To get Active block, we need Garbage Collection
- **Cleaning**



3.2 Cleaning

- **Allocation Pool**
  - Active block to assign to a write operation
  - To get Active block, we need Garbage Collection

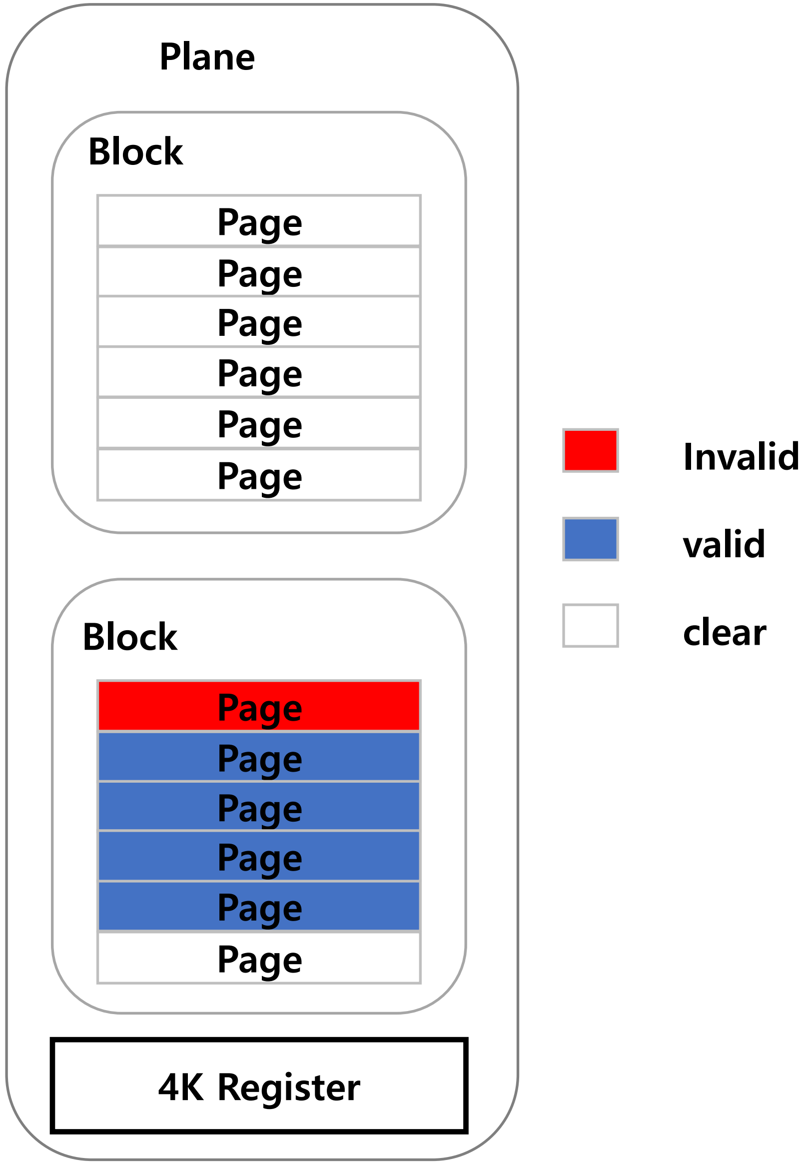
- **Cleaning**



3.2 Cleaning

- **Allocation Pool**
  - Active block to assign to a write operation
  - To get Active block, we need Garbage Collection

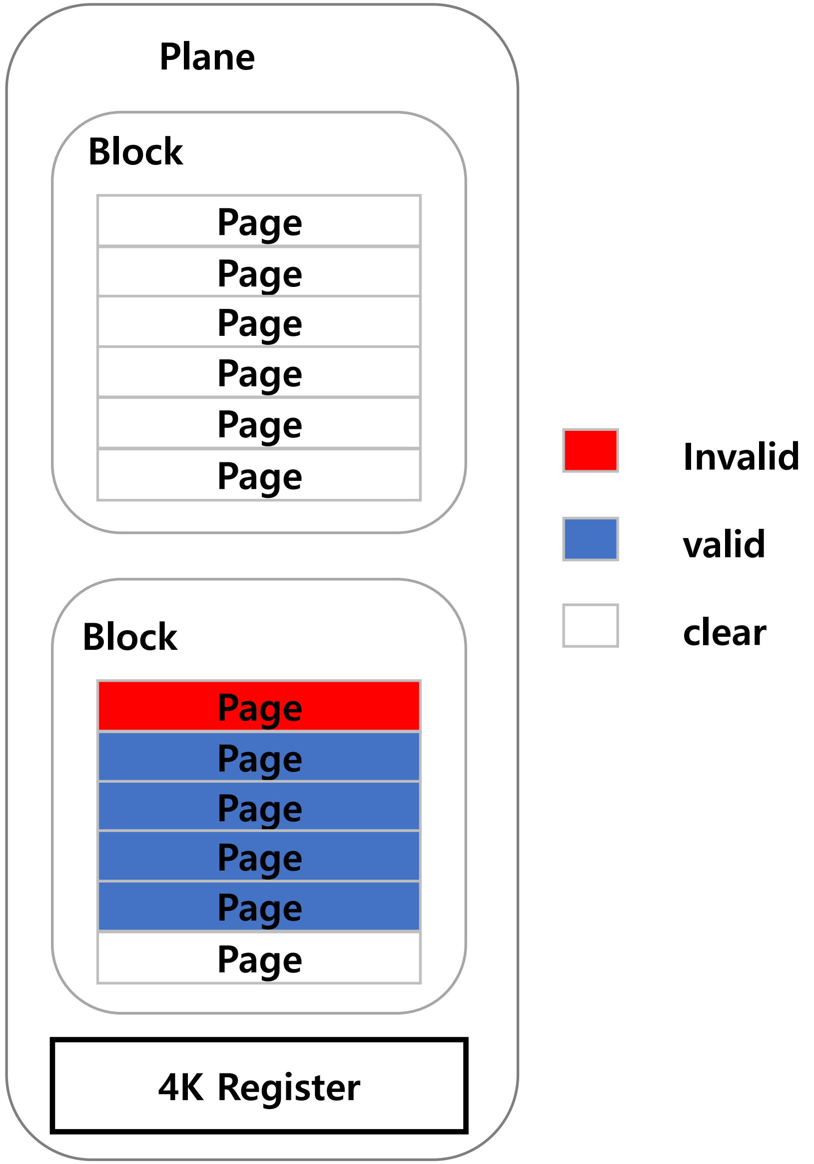
- **Cleaning**



3.2 Cleaning

- **Allocation Pool**
  - Active block to assign to a write operation
  - To get Active block, we need Garbage Collection

- **Cleaning**
  - Cleaning efficiency = $\frac{\textbf{Superseded pages}}{\textbf{Total pages}}$ (In a Block)



3.2 Cleaning

- **Overprovisioning**
  - For cleaning work, spare blocks are necessary

3.2 Cleaning

- **Overprovisioning**

  - For cleaning work, spare blocks are necessary

3.2 Cleaning

- **Overprovisioning**

  - For cleaning work, spare blocks are necessary

3.2 Cleaning

- **Overprovisioning**

  - For cleaning work, spare blocks are necessary

3.2 Cleaning

- **Overprovisioning**
  - For cleaning work, spare blocks are necessary

- **Overprovisioning size ↑, cleaning cost ↓**
  - But reduced capacity

3.2 Cleaning

- **Overprovisioning**
  - For cleaning work, spare blocks are necessary
- **Overprovisioning size ↑, cleaning cost ↓**
  - But reduced capacity
- **Allocation pool size ↑, load balancing ↑**
  - Use copy back, so make a large Allocation pool
  - But keep in intra chip operations

3.3 Parallelism and Interconnect Density

- **Parallelism**
  - Handle I/O requests on multiple flash packages

- **Techniques to obtain parallelism**
  - Parallel requests
  - Interleaving
  - Ganging
  - Background cleaning

3.3 Parallelism and Interconnect Density

- **Organizing gang of flash package**
  - ① Shared bus gang
    - Increase capacity without increasing pins
    - No increase bandwidth
  - ② Shared control gang
    - Parallel processing, increase bandwidth
    - Limit of chip enables, so all package perform the same command

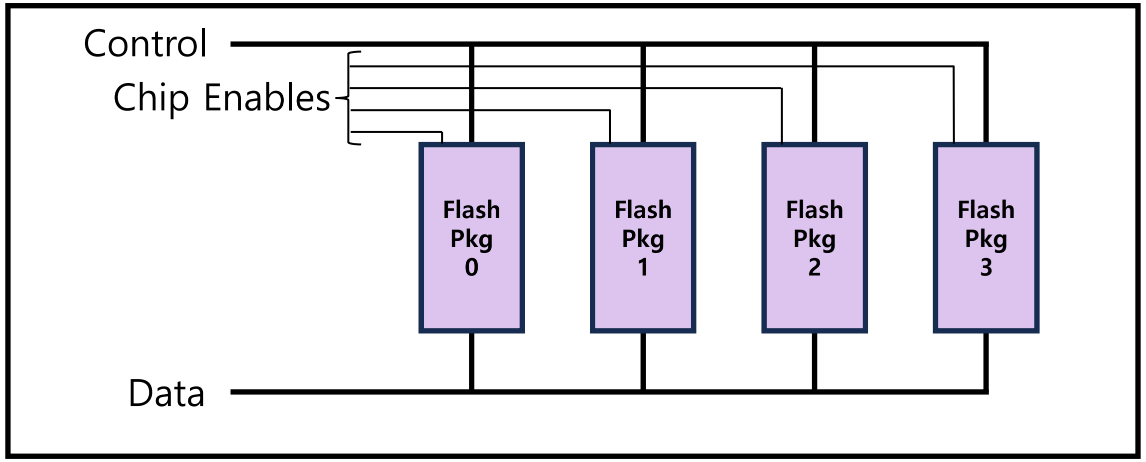


Figure 4: Shared bus gang

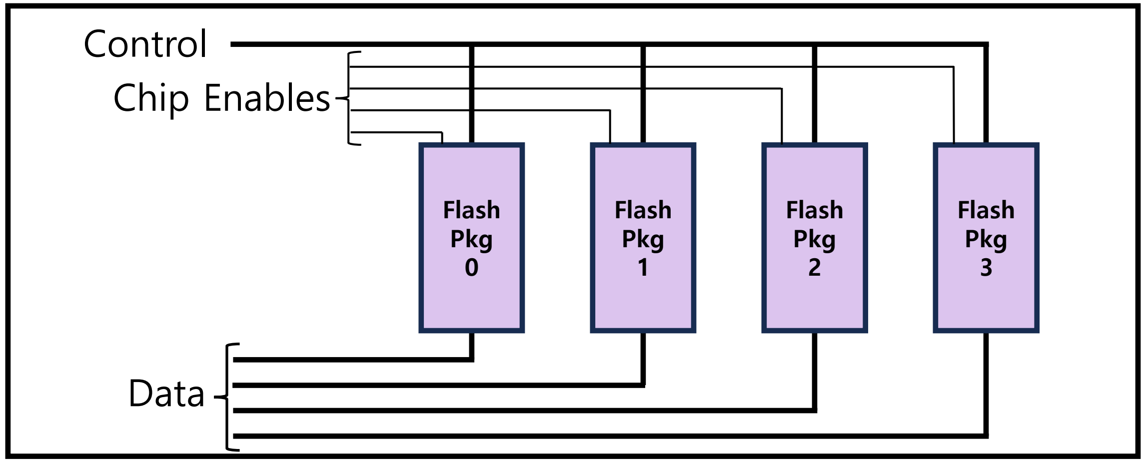


Figure 5: Shared control gang

4. Evaluation

- ## Environment

- workload

- TPC-C : Database workloads
- Exchange : Database workload with more reads than writes
- IOzone , Postmark : Standard file system benchmark

- Simulation

- Modified version of DiskSim simulator

4. Evaluation

- Pros : Parallelism ↑
- Cons : Complexity & constraints

- **Impact of interleaving**

  - TPC-C & Exchange

  Short I/O

  ⬇

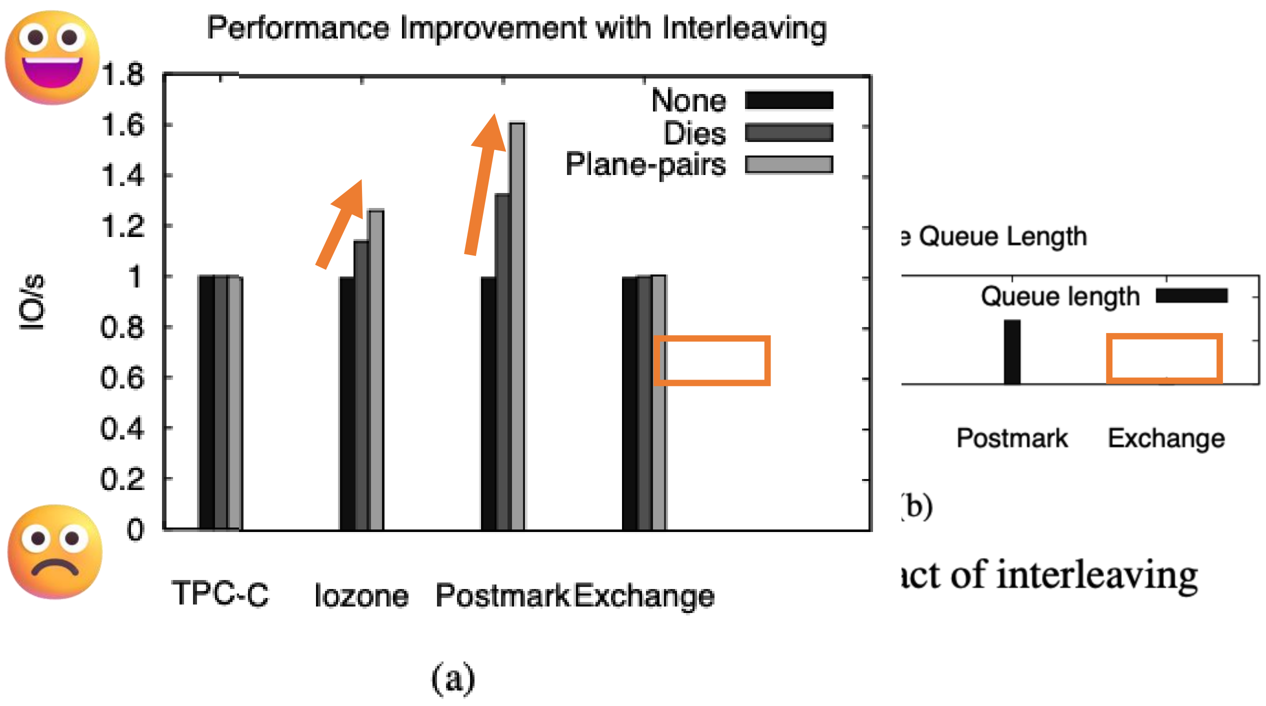
  No queuing

  ⬇

  No interleaving

  - IOzone, Postmark

  - Doubles IO/s



None : Interleaving X
Dies : Distribute through many dies
Plane-pairs : Distribute requests through plane-pairs

4. Evaluation

- **Shared-control ganging**

  - Ganging

  1. sync : all packages managed in synchrony
  2. async: separate allocation and cleaning decisions

     -> using copy-back

- Pros : Sparse wiring (단순한 배선)
- Cons : parallelism ↓

async is better than sync !!

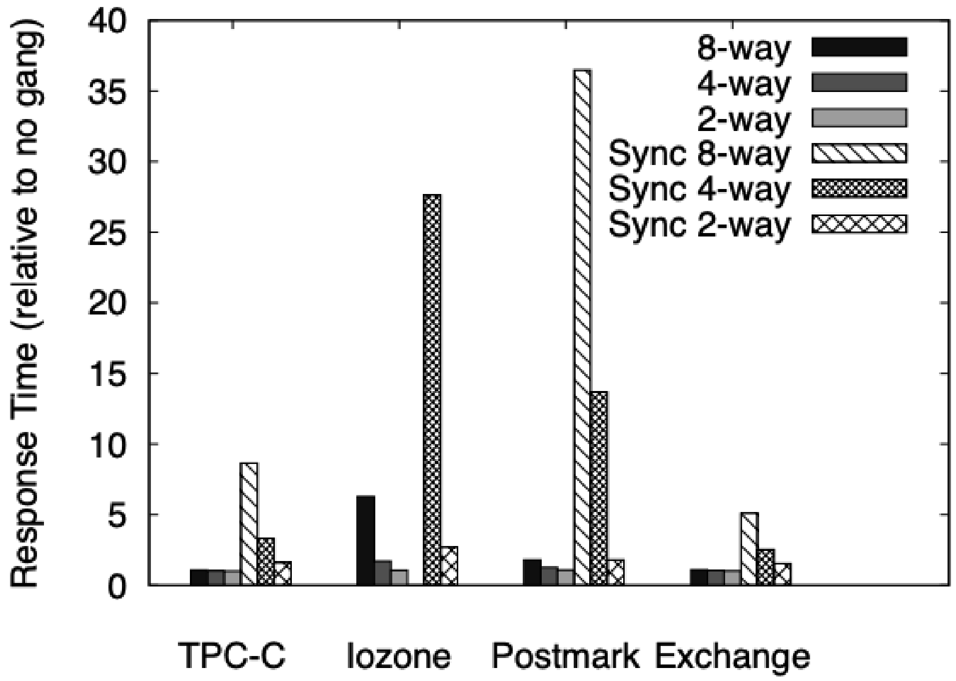


Figure 7: Shared-control ganging

4. Evaluation

- **Cleaning Threshold**
  - **- Increasing of cleaning threshold**
    - Trigger cleaning earlier
    - Pages moved frequently
    - Increasing of latency -> **OVERHEAD**

- SSD needs proper cleaning threshold

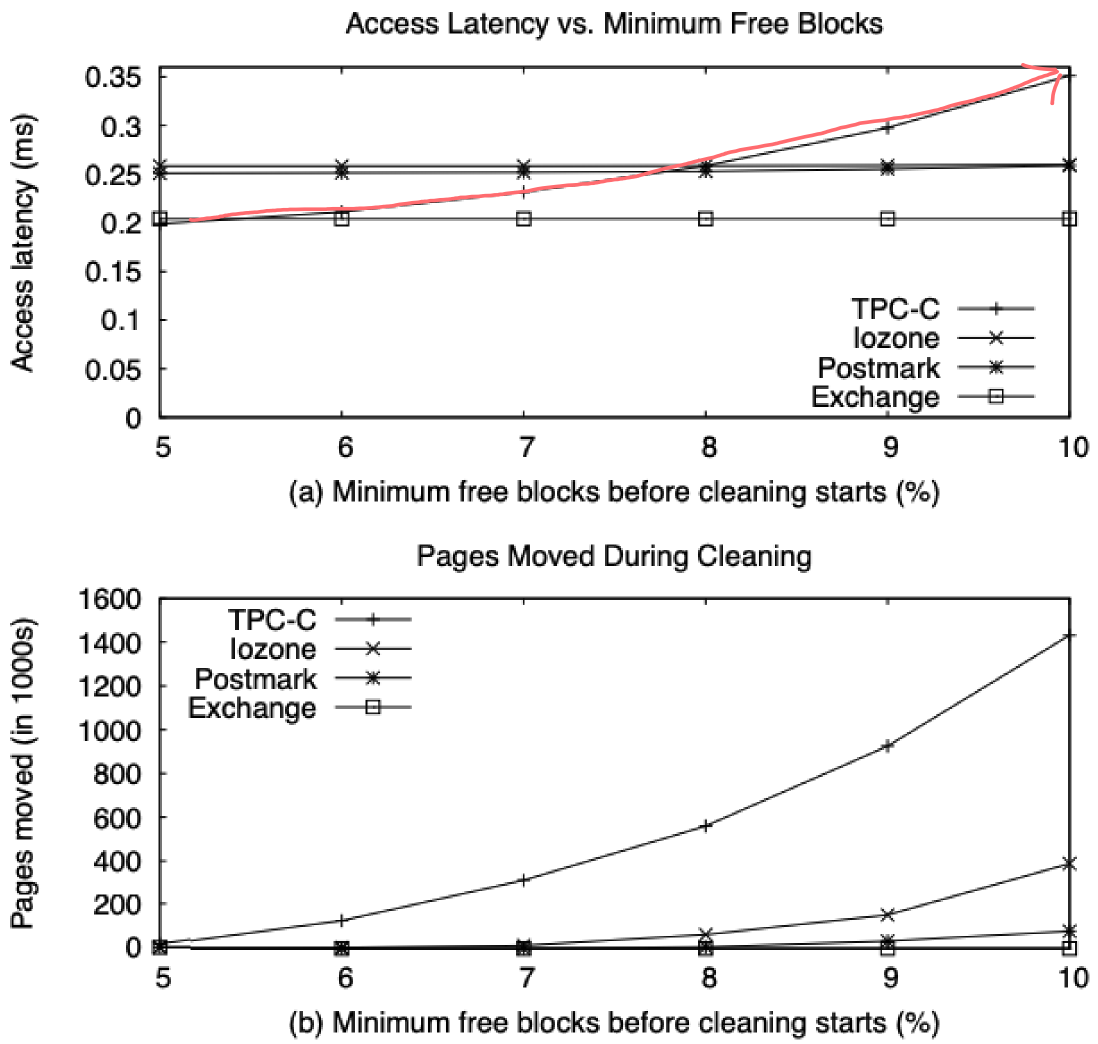
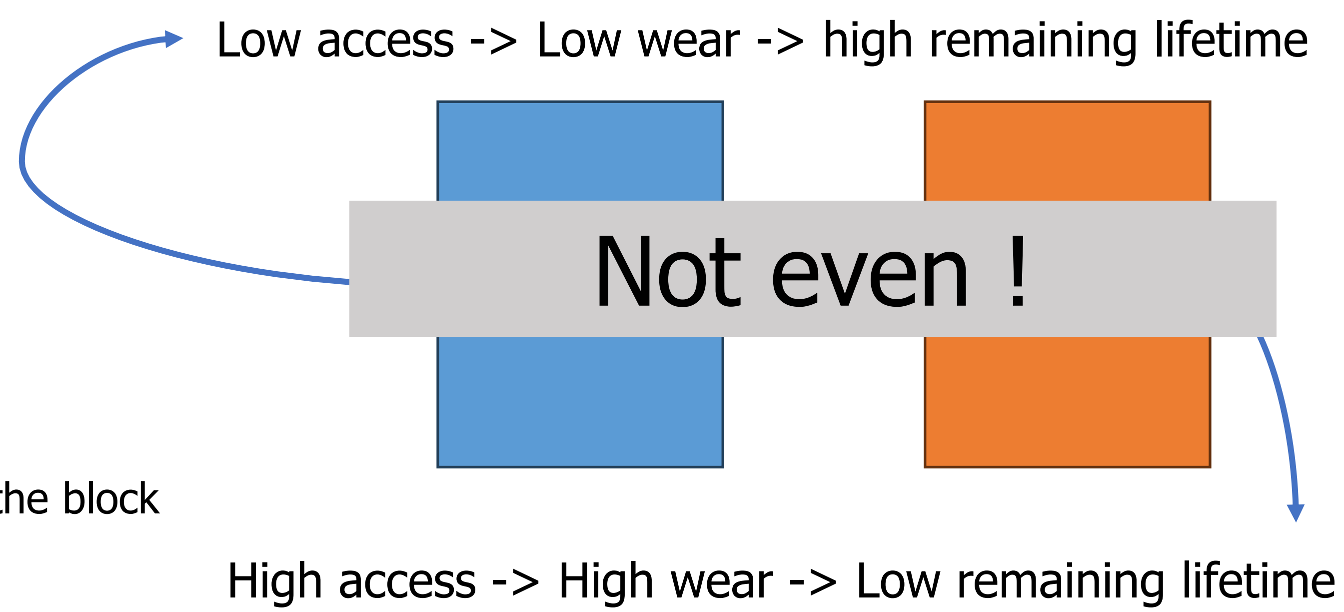


Figure 8: Impact of minimum free blocks

4. Evaluation



- **Wear leveling**

- Greedy algorithm

- Only considering the remaining lifetime of the block

- Rate-limiting algorithm

- The more remaining lifetime,
  the higher recycling probability

- Migration

- Migrates cold data to more worn blocks

	Mean Lifetime	Std.Dev.	Expired blocks
Greedy	43.82	13.47	223
+ Rate-limiting	43.82	13.42	153
+ Migration	43.34	5.71	0

Table 7: Block wear in IOzone

	< 40%	< 80%	< 100%	≥ 100%
Greedy	1442	1323	523	13096
+ Rate-limiting	1449	1341	501	13092
+ Migration	0	0	8987	7397

Table 8: Lifetime distribution with respect to mean

5. Conclusion

- SSD performance is determined by the interaction between hardware and software components and the workload.

- We have shown that plane interleaving and overprovisioning provide design benefits and demonstrated the effectiveness of the wear-leveling.

- Using a simulation technique based on traces from real hardware, we can understand the performance trade-offs for different workloads and components.

Thank you

Design Tradeoffs for SSD Performance

Agrawal, Nitin, Ted Wobber, John D. Davis, Mark Manasse, Rina Panigrahy

Microsoft Research, Silicon Valley University of Wisconsin-Madison

2008 USENIX Annual Technical Conference

Q&A

2024. 07. 17

Presentation by Kim MinSeong, Wee DaYeon

kms0509@dankook.ac.kr, wida10@dankook.ac.kr