CS 471 Operating Systems

Yue Cheng

George Mason University
Fall 2017
Disk vs. Flash
Disk Overview

- I/O requires: seek, rotate, transfer

- Inherently:
  - Not parallel (only one head)
  - Slow (mechanical)
  - Poor random I/O (locality around disk head)

- Random requests each taking $\sim 10+$ ms
Flash Overview

- Hold charge in cells. No moving (mechanical) parts!
- Inherently parallel!
- No seeks!
Storage Hierarchy Overview

**Flash:**
- Smaller capacity
- Faster accesses

**HDD:**
- Larger capacity
- Way slower accesses
Disks vs. Flash: Performance

- **Throughput**
  - Disk: ~130MB/s (sequential)
  - Flash: ~400MB/s

- **Latency**
  - Disk: ~10ms (one op)
  - Flash:
    - Read: 10-50us
    - Program: 200-500us
    - Erase: 2ms
Disks vs. Flash: Performance

- **Throughput**
  - Disk: ~130MB/s (sequential)
  - Flash: ~400MB/s

- **Latency**
  - Disk: ~10ms (one op)
  - Flash:
    - Read: 10-50us
    - Program: 200-500us
    - Erase: 2ms

Types of write, more later…
Disks vs. Flash: Capacity

An obvious question is why are we talking about spinning disks at all, rather than SSDs, which have higher IOPS and are the “future” of storage. The root reason is that the cost per GB remains too high, and more importantly that the growth rates in capacity/$ between disks and SSDs are relatively close (at least for SSDs that have sufficient numbers of program-erase cycles to use in data centers), so that cost will not change enough in the coming decade. We do make extensive use of SSDs, but primarily for high performance workloads and caching, and this helps disks by shifting seeks to SSDs.

~ Eric Brewer et al.

Flash Architecture
SLC: Single-Level Cell

NAND Cell

charge
SLC: Single-Level Cell

NAND Cell

charge

1
SLC: Single-Level Cell

NAND Cell

charge

0
MLC: Multi-Level Cell

00
MLC: Multi-Level Cell
MLC: Multi-Level Cell
MLC: Multi-Level Cell

charge

NAND Cell

11
Single- vs. Multi-Level Cell

charge

SLC

charge

MLC
Single- vs. Multi-Level Cell

SLC

expensive
robust

MLC

cheap
sensitive
Wearout

- Problem: flash cells wear out after being erased too many times
  - MLC: ~10K times
  - SLC: ~100K times
- Usage strategy: ???
Wearout

- Problem: flash cells wear out after being erased too many times

- MLC: ~10K times
- SLC: ~100K times

- Usage strategy: wear leveling
  - Prevents some cells from being wornout while others still fresh
Banks

- Flash devices are divided into banks (aka. planes)
- Banks can be accessed in parallel
Banks

- Flash devices are divided into banks (aka. planes)
- Banks can be accessed in parallel

```
read
read
Bank 0
Bank 1
Bank 2
Bank 3
```
Banks

- Flash devices are divided into banks (aka. planes)

- Banks can be accessed in parallel
Flash Writes

- Writing 0’s
  - Fast, fine-grained

- Writing 1’s
  - Slow, coarse-grained
Flash Writes

- Writing 0’s
  - Fast, fine-grained
  - called “program”

- Writing 1’s
  - Slow, coarse-grained
  - called “erase”
Flash Writes

- Writing 0’s
  - Fast, fine-grained [page-level]
  - called “program”

- Writing 1’s
  - Slow, coarse-grained [block-level]
  - called “erase”
Flash Writes

- **Writing 0’s**
  - Fast, fine-grained [page-level]
  - called “*program*”

- **Writing 1’s**
  - Slow, coarse-grained [block-level]
  - called “*erase*”

- Flash can only “write” (program) into **clean** pages
  - “**clean**”: pages containing all 1’s (pages that have been erased)
  - Flash does not support in-place overwrite!
Blocks

Bank 0

Bank 1

Bank 2

Bank 3
Blocks

Each bank contains many “blocks”
Block

One block
Block

One page
All pages are clean ("programmable")
Block

program

1011 1111 1111 1111
1000 1111 1111 1111

1111 1111 1111 1111
1111 1111 1111 1111

1111 1111 1111 1111
Block

```
1011 1111 1111 1111
1000 1111 1111 1111
1111 0110 1111 1111
1111 1010 1111 1111
```

program
Block

Two pages hold data
(cannot be overwritten)
still want to write data into this page???

Two pages hold data (cannot be overwritten)
Block

```
1011 1111 1111 1111
1000 1111 1111 1111
1111 0110 1111 1111
1111 1010 1111 1111
```

erase
Block

erase
Block

Free state
(can write new data in any page)
Block

This blue page holds data
Flash vs. Disks: APIs

disk  flash

read
write
Flash vs. Disks: APIs

<table>
<thead>
<tr>
<th></th>
<th>disk</th>
<th>flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>read sector</td>
<td>read page</td>
</tr>
<tr>
<td>write</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Flash vs. Disks: APIs

<table>
<thead>
<tr>
<th></th>
<th>Disk</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Read</strong></td>
<td>read sector</td>
<td>read page</td>
</tr>
<tr>
<td><strong>Write</strong></td>
<td>write sector</td>
<td>program page</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0’s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>erase block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1’s)</td>
</tr>
</tbody>
</table>
Flash Architecture

- **Bank/plane**: 1024 to 4096 blocks
  - Banks accessed in parallel

- **Block**: 64 to 256 pages
  - Unit of erase

- **Page**: 2 to 8 KB
  - Unit of read and program
Disks vs. Flash: Performance

- **Throughput**
  - Disk: $\sim130$MB/s (sequential)
  - Flash: $\sim400$MB/s

- **Latency**
  - Disk: $\sim10$ms (one op)
  - Flash:
    - **Read**: 10-50us
    - **Program**: 200-500us
    - **Erase**: 2ms
Working with File System
Traditional File Systems

File System

Storage Device

Traditional API:
• read sector
• write sector
Traditional File Systems

File System

Storage Device

Traditional API:
• read sector
• write sector

Mismatch with flash!
Traditional APIs wrapping around Flash APIs

read(addr):
    return flash_read(addr)

write(addr, data):
    block_copy = flash_read(all pages of block)
    modify block_copy with data
    flash_erase(block of addr)
    flash_program(all pages of block_copy)
Flash Write

Memory

Flash

block 0

block 1

block 2
File system wants to write 0001
Flash Write

Memory

Flash

block 0

block 1

block 2
Flash Write

Memory

Read all pages in block

Flash

block 0

block 1

block 2
## Flash Write

![Flash Write Diagram]

<table>
<thead>
<tr>
<th>Memory</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00</td>
<td></td>
</tr>
<tr>
<td>01 10</td>
<td></td>
</tr>
<tr>
<td>00 11</td>
<td></td>
</tr>
<tr>
<td>01 11</td>
<td></td>
</tr>
<tr>
<td>11 10</td>
<td></td>
</tr>
<tr>
<td>11 11</td>
<td></td>
</tr>
<tr>
<td>block 0</td>
<td>block 1</td>
</tr>
</tbody>
</table>
**Flash Write**

Modify target page in memory

<table>
<thead>
<tr>
<th>Memory</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

block 0  block 1  block 2
Flash Write

Memory

Flash

block 0

block 1

block 2
Flash Write

Memory

Erase whole block

Flash

block 0

block 1

block 2
Flash Write

Memory

Flash

block 0

block 1

block 2
Flash Write

Program all pages in block

Memory

Flash

00 01
01 11

00 01
01 11

01 10
11 11

00 01
01 11

11 01
11 11

11 10
11 11

00 01
01 11

11 10
11 11

11 10
11 11

block 0 block 1 block 2
Flash Write

Memory

Flash

block 0  block 1  block 2
Write Amplification

- Random writes are expensive for flash!

- Writing one 4KB page may cause:
  - read, erase, and program of the whole 256KB block
Flash Translation Layer
Flash Translation Layer (FTL)

- Add an address translation layer between upper-level file system and lower-level flash
  - Translate logical device addresses to physical addresses
  - Convert in-place write into append-write
Flash Translation Layer (FTL)

Logical

Physical

block 0

block 1
Flash Translation Layer (FTL)

Write 0011

Logical

0 1 2 3 4 5 6 7

Physical

00 11 01 10 11 11 00 10
block 0

11 11 11 11 01 11 11 11
block 1
Flash Translation Layer (FTL)

Logical

Physical

Write 0011

block 0

block 1
Flash Translation Layer (FTL)

Logical

Physical

Write 0011

Change addr mapping

block 0

block 1
Flash Translation Layer (FTL)

Logical

Physical

Marked as invalid and will be garbage collected

Write 0011

Change addr mapping
SSD Architecture with FTL

SSD provides disk-like interface

FTL

SRAM: Mapping table
Flash Translation Layer (FTL)

- Usually implemented in flash device’ firmware

- Where to store mapping?
  - SRAM

- Physical pages can be in three states
  - valid, invalid, free
State Transition of Physical Pages

- Free
- Invalid
- Valid
State Transition of Physical Pages

Free  ->  Program  ->  Valid

Invalid

Diagram showing the state transition of physical pages with states Free, Invalid, and Valid, and a program transition arrow between Free and Valid.
State Transition of Physical Pages

- **Free** → **Program** → **Valid**
- **Invalid** → **Relocate (overwrite)**
State Transition of Physical Pages

- Free
- Valid
- Invalid

- Program
- Erase
- Relocate (overwrite)