

# SSD 의 특성과 TRIM 기능의 이해, 자동 TRIM(트림) 기능의 작동 여부 확인과 설정 방법

자동 TRIM 기능의 작동 여부 확인/설정 프로그램과 배치 파일

현재 윈도우에서 SSD 를 위한 **자동 TRIM(트림)** 기능을 사용 중인지 바로 확인하고, 동시에 기능을 켜고 끌 수 있는 간단한 프로그램과 배치 파일입니다. 참고로 **자동 TRIM 기능은 윈도우 7, 윈도우 8 에서만 지원(동일 커널의 서버 윈도우 포함)**하기 때문에 그 이전의 윈도우에서는 사용할 수 없습니다.

배치 파일과, 실행 프로그램 두 가지로 준비해보았습니다. 취향껏 받으세요. [프로그램은 윈도우 7 에 기본 포함된 닷넷 3.0 버전을 기준으로 제작하여 윈도우 7 에서는 곧바로 사용 가능합니다. 윈도우 8 은 닷넷의 설치 여부에 따라 닷넷을 먼저 설치해야 할 수도 있습니다. 그럼 그냥 배치 파일로 쓰세요.]



자동 TRIM 기능의 작동 여부 확인 및 설정 방법

윈도우 8 과 윈도우 7 에서는 아래의 명령을 통해 자동 TRIM 여부를 확인할 수 있습니다.

**fsutil behavior query DisableDeleteNotify**

**DisableDeleteNotify = 0 : TRIM 켜짐(활성)**

**DisableDeleteNotify = 1 : TRIM 꺼짐(비활성)**

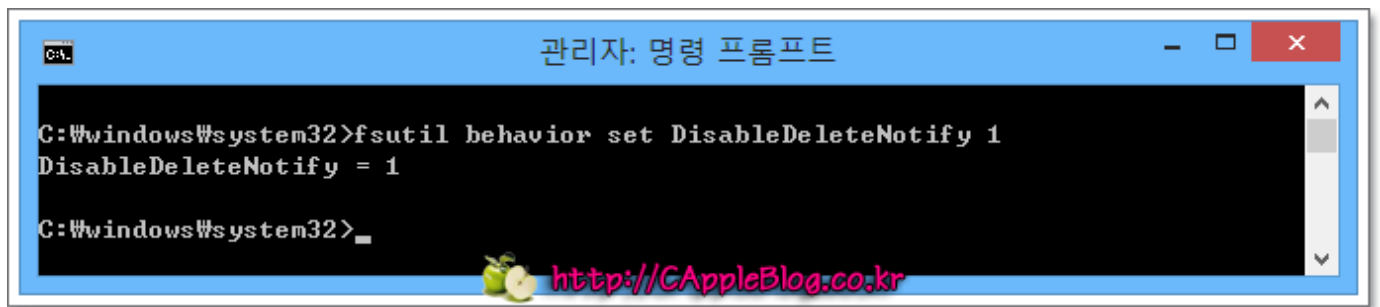
```
C:\Windows\system32>fsutil behavior query DisableDeleteNotify
DisableDeleteNotify = 0

C:\Windows\system32>
```

**DisableDeleteNotify** 의 값이 **0** 이면 자동 TRIM 기능을 사용 중인 상태이고, **1** 이면 사용 중이지 않은 상태이죠. 그리고 이러한 **DisableDeleteNotify** 값은 다시 아래의 명령을 통해 직접 설정할 수 있습니다. 이 때 해당 명령은 관리자 권한으로 실행되어야 합니다.

**fsutil behavior set DisableDeleteNotify 0**

**fsutil behavior set DisableDeleteNotify 1**



```
C:\Windows\system32>fsutil behavior set DisableDeleteNotify 1
DisableDeleteNotify = 1

C:\Windows\system32>
```

굉장히 간단하죠? 배치 파일과 프로그램은 이러한 확인 작업과 설정 작업을 자동으로 진행해주는 것입니다. 별거 없는데 프로그램까지 만들면서 괜히 있어 보일려고 많이 노력했습니다. ^^ 근데 이대로 끝내긴 아쉬우니 간단하게 TRIM 에 대해서 좀 더 이야기하는 시간을 가져보도록 하겠습니다.

TRIM 이란 무엇인가? - SSD 를 구성하는 NAND Flash Memory 의 특성

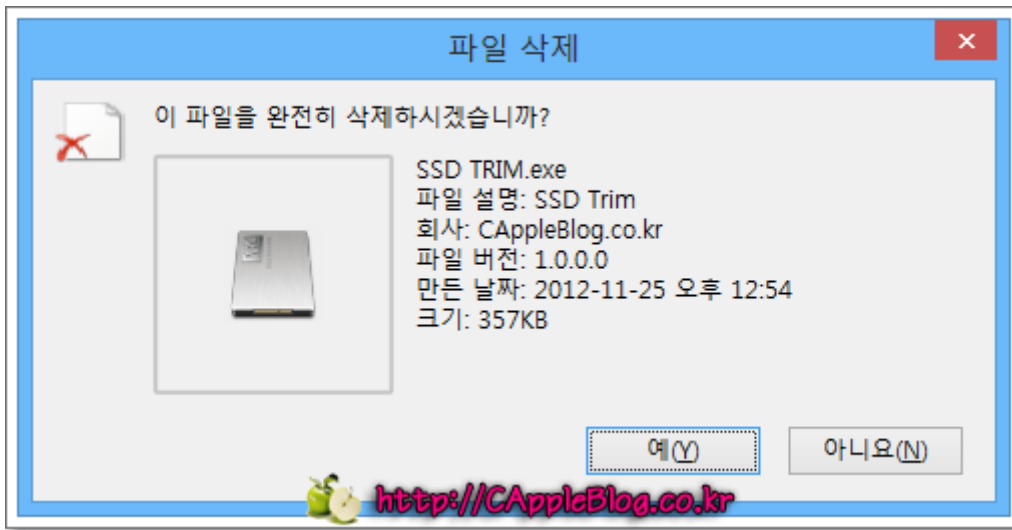
**참고!** 이 단락에서 이야기하는 디스크라는 용어는 윈도우에서의 표현을 기준으로 한 것으로 HDD, SSD 와 같은 저장 매체를 의미하는 것입니다. 의미상으로 큰 문제가 없기 때문에 그냥 디스크로 통일하도록 하겠습니다.

또한 SSD 에서의 TRIM 명령은 보통 대문자로 표기하지만 TRIM 이 약어는 아닙니다. TRIM 은 단어 그대로의 Trim 일 뿐입니다. 다만 Trim 이라는 용어가 단어 자체이기도 하고 Trim 함수 등 많이 쓰이기 때문에 이를 구별하기 위해 일반적으로 대문자로 표기하는 것입니다. 개인적으로 별 의미는 없어 보이지만, 글에서도 이를 따르도록 하겠습니다.

그렇다면 TRIM 이란 무엇일까요? 아주 간단하게 표현하자면 **TRIM 이란 윈도우에서 파일을 지우면 SSD 에서도 실제로 파일을 지우는 기능**입니다. 일단 먼저 윈도우에서 파일의 삭제란 무엇인지 이야기해보도록 하겠습니다.

## 1. 운영체제의 파일 삭제 구조

지난 [파일의 완전 삭제란 - 제로필과 DoD 5220.22- 와이핑](#) 글에서 이야기한 것과 같이 운영체제(윈도우)에서 파일을 삭제하더라도 실제로 디스크에서는 파일이 삭제되지 않습니다. 한 마디로 디스크엔 삭제된 파일이 고스란히 남아 있는 것이죠.



나는 윈도우에서는 삭제되었지만 디스크에서는 실제로 삭제된 것이 아니라네.

운영체제가 이러한 방식을 취하는 이유는 디스크에서 파일을 실제로 삭제하게 되면 파일을 기록하는 것만큼 (Write) 삭제 작업도(Erase) 오래 걸리게 되기 때문입니다. 성능상 좋지 않죠.

결정적으로 HDD 는 특성상 굳이 디스크에서 데이터를 실제로 지울 필요가 없었습니다. 왜냐하면 HDD 는 **덮어쓰기(Overwrite)**가 가능하기 때문에 이전의 데이터를 지울 필요없이 해당 데이터 위에 곧바로 다른 데이터를 쓰는 것이 가능합니다. 즉, HDD 는 어떠한 데이터를 쓰기 위해 해당 섹터가 비어있든 비어있지 않든 아무런 상관이 없다는 것이죠. Write 나 Overwrite 나 성능의 차이도 없습니다.

그래서 운영체제는 파일을 삭제하면 파일의 메타 데이터만 삭제하여 파일 시스템과 실제 파일과의 연결만 끊을 뿐(invalid), 굳이 디스크에서까지 실제로 파일을 삭제하지는 않고 그냥 남겨두는 것입니다. 앞으로 디스크를 사용하다보면 다른 파일들에 의해 덮어쓰기가 이루어져 기존의 파일들은 자연스럽게 사라지게 될 테니까요. 즉, 운영체제의 기본적인 파일 삭제 구조는 HDD 의 Overwrite 를 염두한 방식이라고 할 수 있습니다.

## 2. 플래시 메모리를 사용한 SSD 의 등장



## 플래시 메모리를 사용한 SSD 의 등장

세월은 흘러 HDD 외에 **SSD(Solid State Drive)** 라는 전혀 새로운 방식의 디스크가 등장합니다. 바로 **플래시 메모리**를 사용한 디스크가 등장을 한 것이죠. SSD 는 플래시 메모리를 매우 효율적으로 구성하여 성능을 끌어 올렸고, 우리에게 신세계를 체험할 수 있는 기회를 주었습니다.

그런데 한 가지 큰 문제가 발생합니다. 바로 지금까지 기본적으로 사용되던 운영체제에서 파일을 삭제하더라도 디스크에서는 실제로 파일을 삭제하지 않는 운영체제의 기본 파일 삭제 방식이 SSD 에선 문제가 되었습니다. 그렇다면 이것이 왜 문제일까요? 일단 그 전에 SSD 가 사용하는 플래시 메모리라는 저장 매체의 특성에 대해서 알아보도록 하겠습니다.

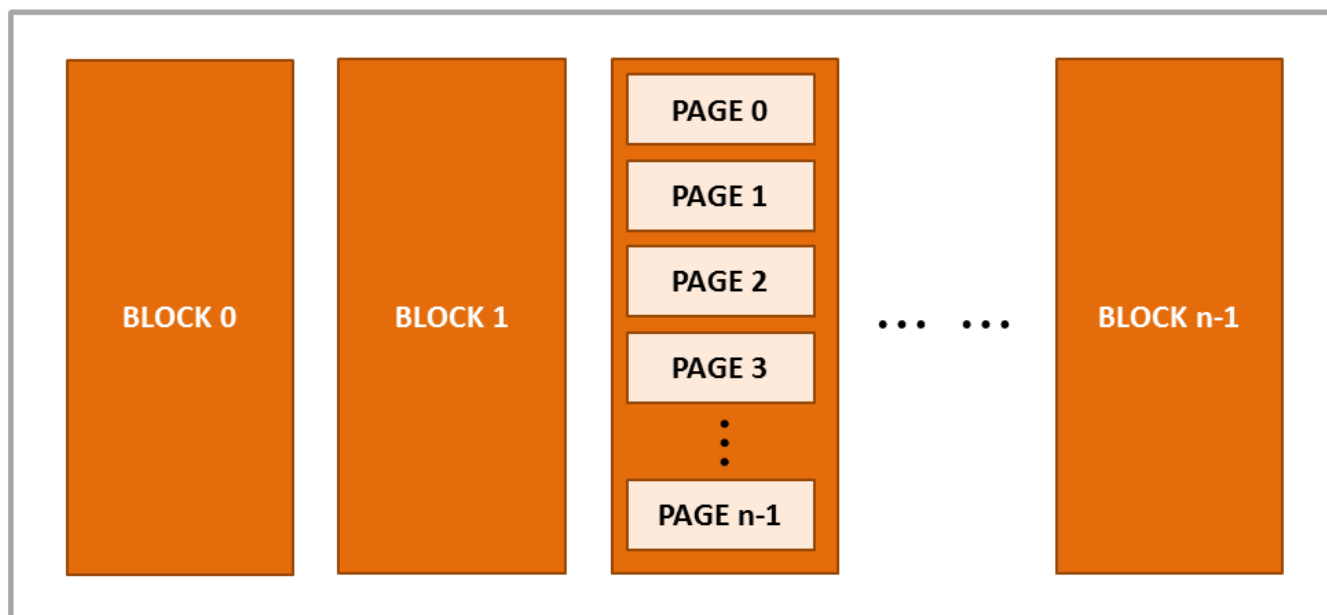
### A. SSD = NAND Flash Memory

SSD 는 플래시 메모리라는 반도체 메모리를 기반으로 하는 저장 매체입니다. 그 중에서도 NAND 방식의 플래시 메모리를 사용하죠. 아무튼 이러한 플래시 메모리는 기계적 구동 장치 없이 오직 전기적인 신호로만 데이터를 읽고 쓰기 때문에, 기계적인 구동부를 가진 HDD 보다 접근(Access) 속도에서 큰 이점을 가지고 있으며, 읽기(Read)와 쓰기(Write) 속도도 그리 나쁜 편은 아닙니다. 특히나 접근 속도는 HDD 와는 비교를 불허할 정도로 빠르다고 할 수 있죠. 그래서 SSD 는 이러한 플래시 메모리를 사용하고, 플래시 메모리를 여러 개의 채널로 연결하여(쉽게 내부적으로 레이드를 구성하여) 성능을 지금의 수준까지 끌어 올렸습니다.

뭐 간단하게 이야기하면 그렇습니다. 이걸 이번 글에선 중요한 것이 아니기 때문에 간략하게 이쯤에서 접고, 이어서 NAND 플래시 메모리의 저장 구조와 특성에 대해서 이야기하도록 하겠습니다.

### B. NAND 플래시 메모리의 저장 구조와 특성

NAND 플래시 메모리는 **페이지(Page)**와 **블록(Block)**이라는 구조로 이루어져 있습니다. 쉽게 하나의 블록 안에 페이지들이 모여 있고, 다시 하나의 플래시 메모리 칩 안에 이러한 블록들이 모여있는 구조입니다. 이러한 NAND 플래시 메모리의 구조를 추상적으로 표현하면 아래와 같습니다.



NAND 플래시 메모리는 종류에 따라 다르지만 보통 **페이지는 4KB, 블록은 512KB**의 크기로 이루어져 있습니다. 이제 이러한 구조 속에서 데이터를 읽고, 쓰는 데에는 페이지 단위로 처리하고, 지우는 데는 블록 단위로 처리를 하게 됩니다. [이는 NAND 플래시 메모리라는 저장 매체의 구조적 특성상 어쩔 수 없는 부분입니다. 그냥 그렇다고 이해하고 넘어 가겠습니다.]

**읽기/쓰기 : 페이지 단위**

**지우기 : 블록 단위**

즉, 읽고 쓰는 것은 4KB의 페이지 단위로 이루어지고, 지우는 것은 그보다 큰 512KB의 블록 단위로 이루어지는 것이죠. 문제는 NAND 플래시 메모리는 HDD와 달리 **덮어쓰기(Overwrite)**가 불가능합니다. 오직 지워져서 비어있는(Empty) 페이지에만 데이터를 기록할 수 있습니다. 즉, 비어있지 않은(기존에 데이터가 남아 있는) 페이지에 데이터를 기록하기 위해선 반드시 먼저 해당 페이지의 데이터를 지워서(Erase) 비운 후 기록해야만 합니다. HDD에선 간단하게 Overwrite 한 번으로 끝날 작업을 플래시 메모리에선 반드시 Erase -> Write 순으로 작업을 해야만 한다는 것이죠.

그런데 말했듯이 이러한 Erase는 블록 단위로만 가능합니다. 이러한 구조로 인해 필연적으로 하나의 굉장히 난해한 문제가 발생하게 됩니다. 바로 **특정 페이지를 재기록하기 위해선 해당 페이지가 포함된 블록 전체를 먼저 지워야만** 한다는 겁니다.

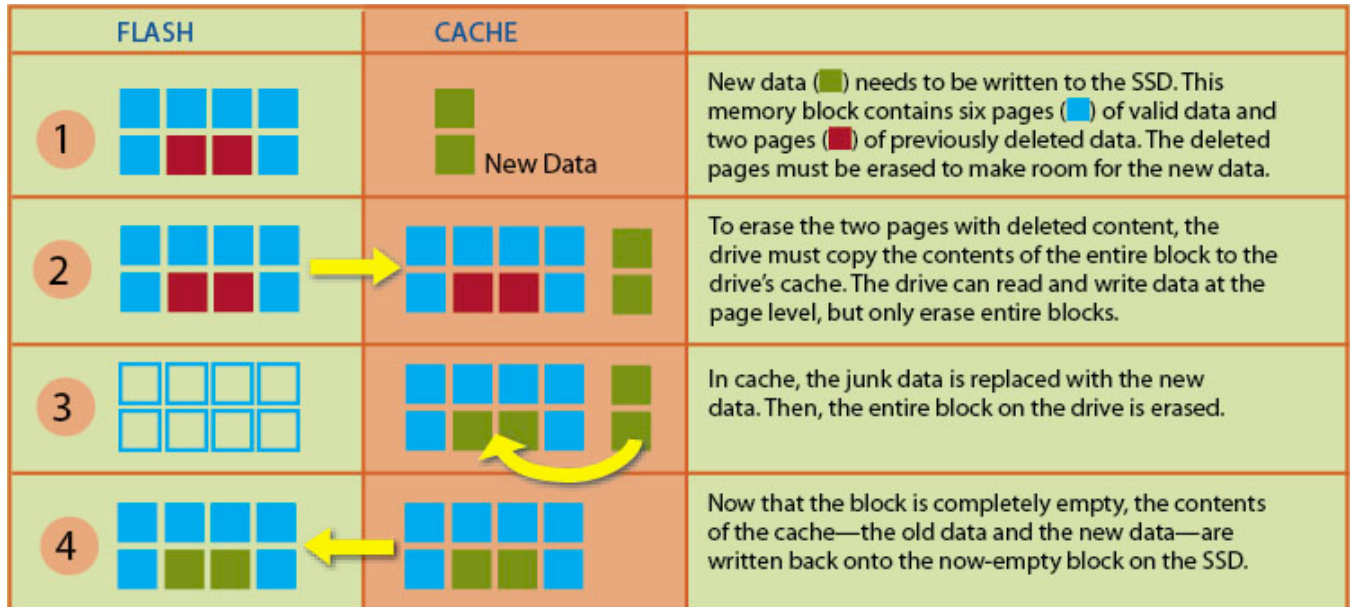
**비어있는 페이지 → 즉시 데이터 기록 가능**

**비어 있지 않은 페이지 → 기존 데이터를 지운 후 데이터 기록 가능, 하지만 지우기는 블록 단위로만 가능.**

### 3. 플래시 메모리의 쓰기/지우기 특성 문제

이러한 구조적인 문제로 인해 플래시 메모리에서는 비어있지 않은 페이지에 데이터를 기록하기 위해 해당 블록의 나머지 전혀 상관없는 다른 데이터들까지 함께 지워야만 하는 문제가 발생하게 됩니다. 새로운 데이터

기록하자고 다른 데이터를 읽을 순 없겠죠. 그래서 SSD 에서는 먼저 해당 블록의 전체 데이터를 SSD 내부의 캐쉬(DRAM)로 옮긴 후 블록을 비우고, 캐쉬에서 기존의 데이터와 새로운 데이터를 결합합니다. 그리고 최종적으로 이렇게 캐쉬에서 결합한 데이터를 다시 블록 전체에 기록해주는 과정을 거치게 됩니다. 이를 그림으로 간략하게 표시하면 아래와 같습니다.



이미 데이터가 있는 비어있지 않은 페이지에 새로운 데이터를 기록하는 과정 (네모 상자 하나 = 페이지)

출처 - [\[MAXIMUMPC\] White Paper: The TRIM Command](#)

즉, 플래시 메모리를 사용하는 SSD 에선 4KB 의 데이터를 기록하더라도 기록하려는 위치의 페이지가 비어 있지 않다면 512KB 에 달하는 블록 전체의 데이터를 옮긴 후, 비우고, 결합하여 다시 쓰는 작업을 진행해야만 하는 문제가 있습니다. 참고로 이것은 비단 SSD 뿐만 아니라 플래시 메모리를 사용하는 모든 저장 매체에서 공통적으로 가지게 되는 문제입니다.

그런데 이렇게 생각할 수도 있습니다. SSD 는 HDD 와 비교할 수 없을 만큼 빠른 성능을 가지고 있고, 그렇다면 이렇게 Overwrite 가 불가능하여 Erase -> Write 작업을 진행할 수 밖에 없다고 할 지라도(몇 단계의 과정을 더 거치더라도), SSD 의 그 빠른 성능으로 상쇄가 되어 별 차이가 없지 않을까? 라고 말이죠. 분명 처음 접한 SSD 의 그 엄청난 속도를 생각하면 충분히 그런 생각을 가질 수도 있을 법 합니다.

근데 그랬으면 좋겠지만 플래시 메모리의 또 다른 특성 중에 하나가 바로 접근(Acess), 읽기(Read), 쓰기(Write) 속도는 빠르지만, 상대적으로 지우기(Erase) 속도가 많이 느리다는 겁니다. 읽기 쓰기 속도에 비해 지우기 속도가 얼마나 느린지는 아래의 표를 참고하시길 바랍니다.

동작	속도	매개변수
Read page	~20 us	(chip #, block #, page #)
Write page	~200 us	(chip #, block #, page #)
Erase block	~2 ms	(chip #, block #)

Copy-back  
page

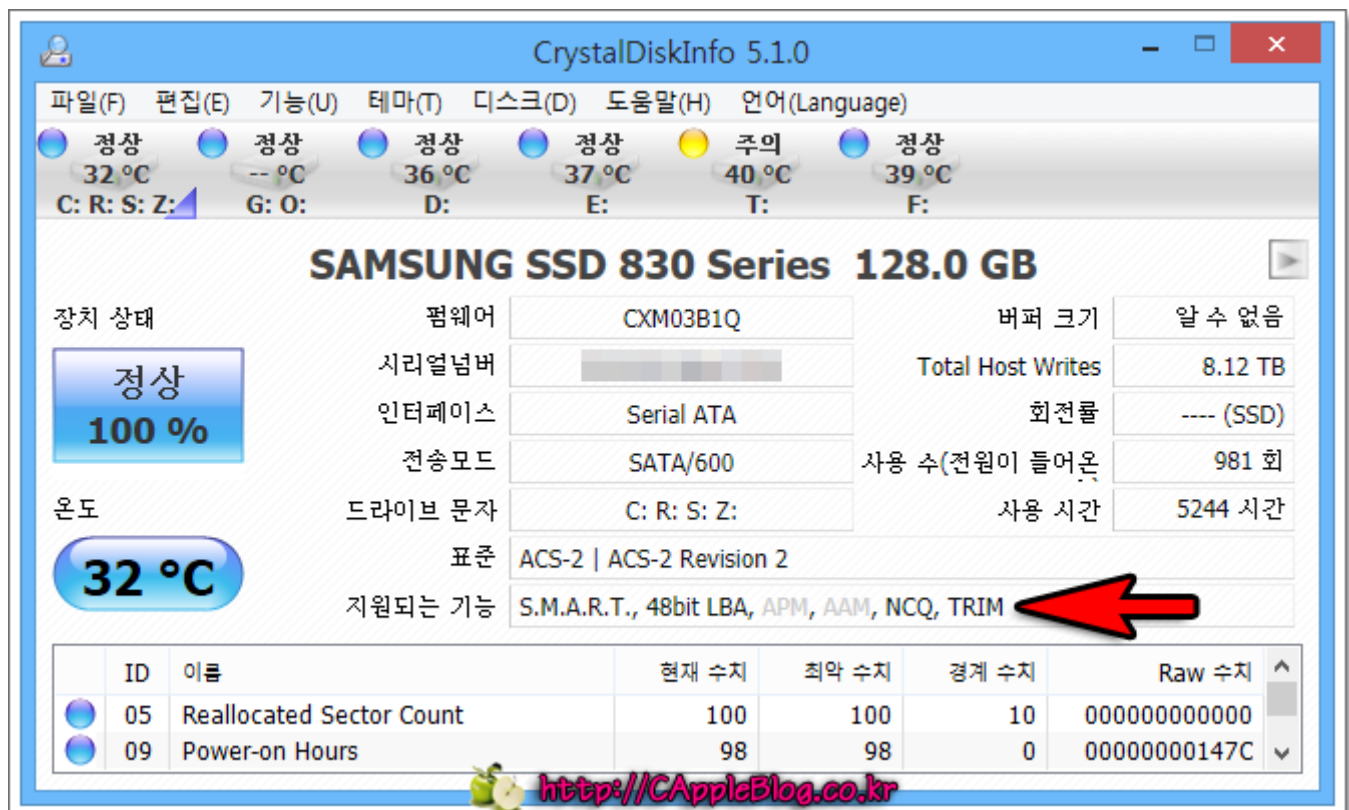
~200  
us

(chip #, target block #, target page #, source block #, source page #)

출처 - [\[FORENSIC-PROOF\] SSD 포렌식 : 데이터 복구 \(SSD Forensics : Data Recovery\)](#)

표의 데이터를 기준으로 쓰기는 0.2 밀리초 수준인데 반해, 지우기는 무려 2 밀리초 수준으로, 간단하게 생각 하더라도 쓰기에 비해 무려 10 배 가까이 느린 것을 알 수 있습니다. 즉, 이러한 지우기가 느린 플래시 메모리의 특성상 비어있는 페이지에 데이터를 기록하는 작업에 비해, 비어있지 않은 페이지에 데이터를 기록하는 작업은 무려 10 배 이상 느려질 수 있다는 이야기입니다. SSD 가 아무리 빠르더라도 갑자기 쓰기 속도가 10 배 정도로 느려지면 그건 좀 그렇죠. 과거 SSD 들의 프리징 문제도 이러한 느린 Erase 작업의 영향으로 발생하는 경우가 많았습니다.

#### 4. TRIM 명령의 등장



TRIM 명령을 지원하는 SSD 의 모습

처음에 이야기했듯이 운영체제는 기본적으로 파일을 삭제하면 디스크에서 해당 파일을 실제로 지우는 것이 아니라, 그냥 파일의 메타 데이터를 삭제하여 파일 시스템과의 연결만 끊고(invalid) 디스크에는 파일을 그대로 남겨 놓습니다. 즉, 디스크는 사용할 수록 이러한 실제로는 사용되지 않는 쓰레기 데이터가 점점 늘어나 그만큼 공간을 차지하고 있게 됩니다.

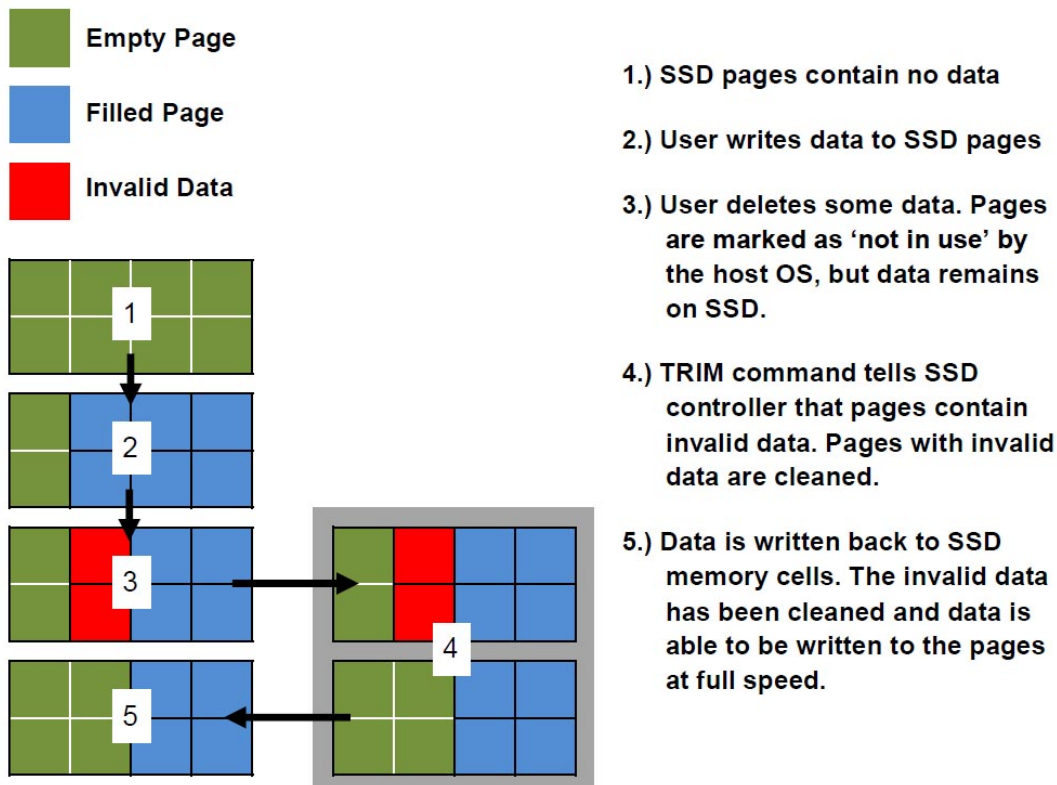
이것이 HDD 시대에는 아무런 문제가 되질 않았습니다. Overwrite 하면 끝이니깐요. 하지만 지금까지 알아본 것과 같이 플래시 메모리를 사용하는 SSD 에서는 이렇게 쌓인 쓰레기 데이터들은(단순히 OS 차원에서만



invalid 된 데이터들은) 추후에 SSD 의 전체적인 쓰기 성능에 큰 영향을 미치게 됩니다. 이러한 데이터들이 담긴 페이지가 많다는 것은 쓰기 작업에서 해당 데이터들을 만날 가능성이 높아진다는 것이고, 이는 SSD 의 입장에서 보자면 Write 작업에 앞서 Erase 작업을 먼저 진행해야할 경우가 그만큼 많아진다는 것을 의미하게 됩니다. 말했지만 Erase 는 느립니다. 그리고 복잡하죠.

그래서 SSD 를 구매한 직후에는 모든 페이지와 블록이 깨끗하게 비워져 있어 엄청난 쓰기 속도를 보여주지만, 이후 사용할 수록 점점 Erase -> Write 작업의 증가로 인해 SSD 의 전체적인 쓰기 성능이 큰 폭으로 하락하는 문제가 발생하게 됩니다. 이는 바꿔 말하면 SSD 의 쓰기 성능은 얼마나 많은 페이지와 블록이 쓰레기 데이터 없이 올바르게 비워져 있느냐에 달려 있다고 할 수도 있습니다.

이러한 문제를 해결하기 위해 SSD 에서는 **TRIM(트림)** 이라는 명령이 새롭게 등장하게 됩니다. TRIM 명령은 운영체제에서 삭제된 쓰레기 파일들을 파악하여 SSD 상에서도 실제로 해당 파일들을 지우는 기능입니다. 쉽게 생각하자면 TRIM 명령은 어차피 나중에 기록할 때 Erase 될 꺼 그냥 미리 적절히 Erase 시키는 거라고 생각하시면 되고, 처음에 이야기한 것과 같이 진짜 단순하게 생각하면 윈도우에서 파일을 지우면 SSD 에서도 실제로 파일을 지우는 기능이라고 생각하면 되는 것입니다.



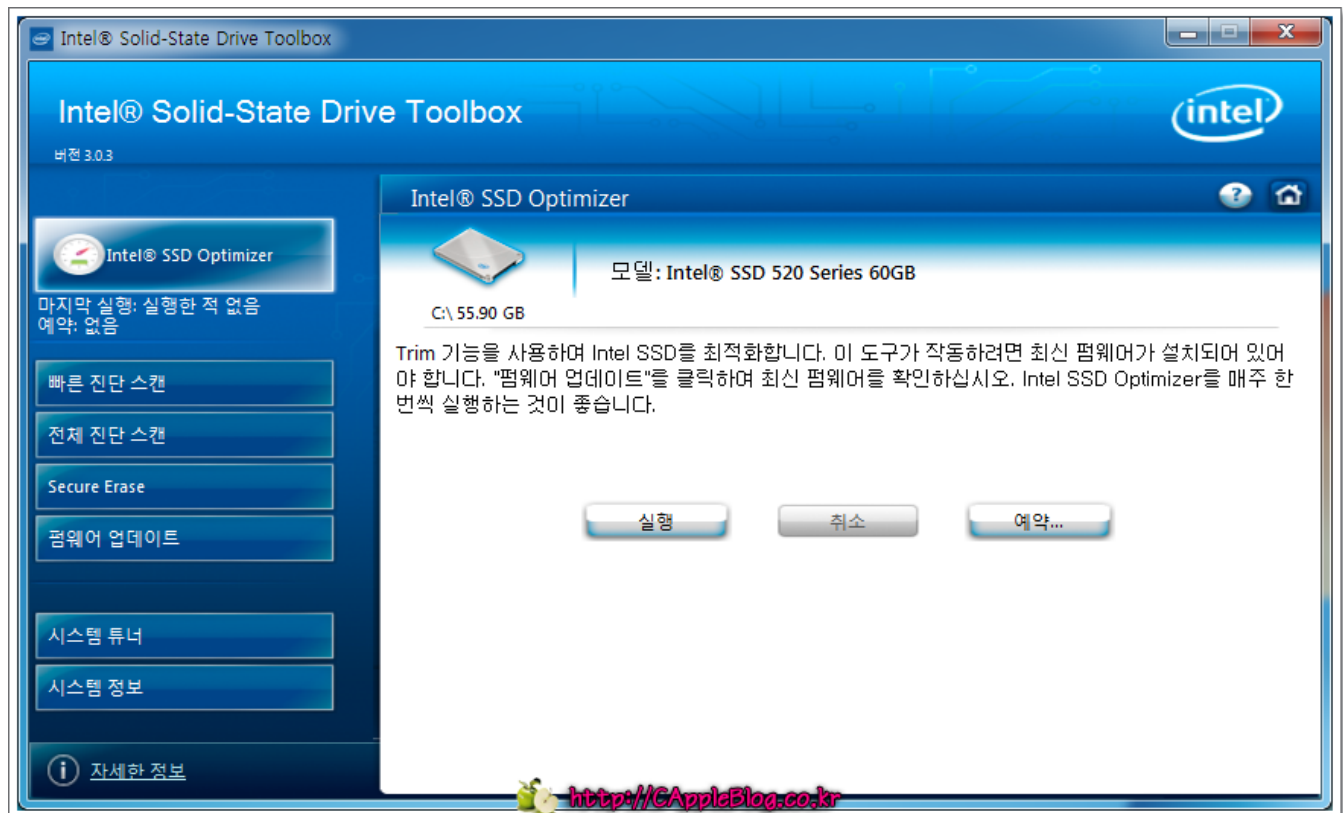
이제 이러한 SSD 의 TRIM 명령이 보다 효율적으로 운영될 수 있도록 운영체제와 SSD 가 서로 손을 잡고 협동하게 됩니다. 이러한 배경으로 윈도우 7 부터 등장한 것이 앞서 알아 본 DisableDeleteNotify 옵션이며, 이는 윈도우에서 해당 파일이 지워지면 SSD 에게 해당 파일은 이제 쓰레기니 네가 알아서 지워라~ 라고 실시간으로 알려주는 것을 의미합니다. [왜 직접 지우지 않고 알려주기만 하는지는 뒤에서]

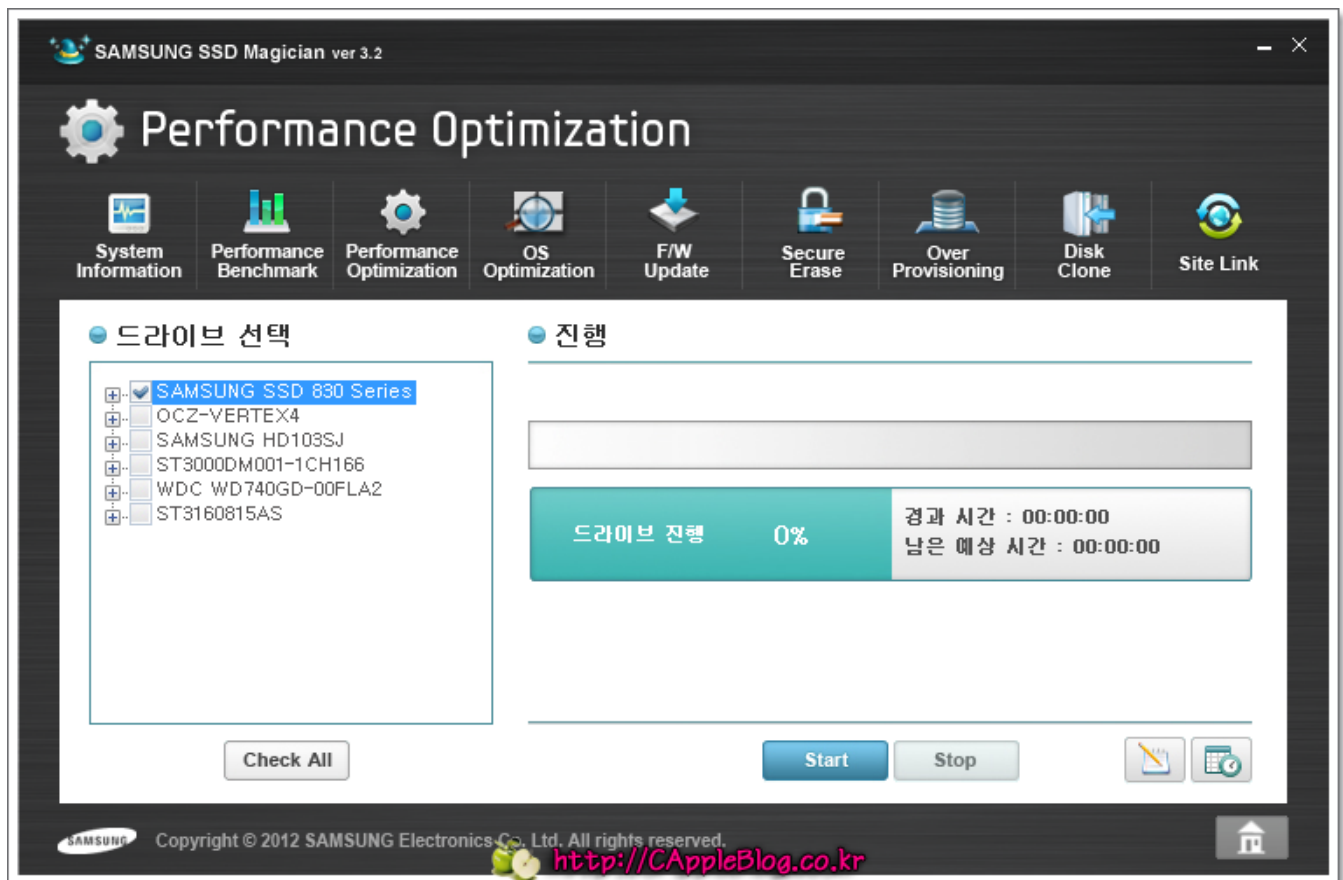
이제 운영체제로부터 이러한 정보를 받은 SSD 는 해당 파일이 담긴 페이지를 필요없는 쓰레기 페이지로 마크한 후, 적절한 알고리즘을 통해 적절한 시기에 해당 파일을 실제로 지워 페이지를 Empty 상태로 만듭니다. 이것이 바로 자동 TRIM 기능입니다. 즉, 자동 TRIM 기능은 운영체제와 SSD 양쪽에서 모두 동시에 지원을 해야



하며, 운영체제가 삭제된 파일을 알려주고 SSD 가 알아서 처리하는 구조로 작동하게 됩니다.

하지만 윈도우 비스타나 윈도우 XP 와 같은 이전의 운영체제들에는 이렇게 삭제된 파일을 SSD 에게 알려주는 기능이 없습니다. 그래서 이러한 윈도우들에선 자동 TRIM 기능을 사용할 수 없는 것이죠. 때문에 해당 윈도우 사용자들은 SSD 제조사에서 제공하는 도구를 통해 사용자가 직접 주기적으로 TRIM 작업을 진행해야 합니다. 이것이 SSD 툴박스에서 제공하는 수동 TRIM 기능이며(또는 최적화(Optimizer) 라고도 표현), 해당 기능은 도구에서 직접 삭제된 파일들을 파악하고 이를 통해 직접 TRIM 기능을 수행하는 것입니다.





## SSD 제조사에서 제공하는 툴박스의 수동 TRIM 기능

한 가지 알아야 할 것이 SSD 에서는 Zero Filling(0 으로 기록) 이 Erase 가 아닐 수 있다는 겁니다. HDD 에서는 섹터 전체를 0 으로 기록하는 제로필이 곧 섹터를 깨끗하게 지우는 것이었지만(Wipe), SSD 는 페이지를 0 으로 기록하는 것이 꼭 소자를 비우는 것을(Empty, 결론적으로 Erase) 의미하는 것은 아니라는 것입니다.

플래시 메모리는 사용된 소자에 따라 Empty 값이 모두 다릅니다. 즉, 플래시 메모리에선 0 이 빈 값일 수도 있고, 1 이 빈 값일 수도 있고, 2 나 3 이 빈 값일 수도 있다는 것이죠.(MLC, TLC 는 소자 하나로 2비트 3비트를 처리하죠.) 그래서 각각 제조사 전용의 Wiper (TRIM, Optimizer) 툴을 제공하는 것이고, 이러한 Wiper 툴은 자사의 제품에 맞는 올바른 값으로 페이지와 블록을 채워 비우게 됩니다.

지금까지 HDD 에서 통용되던 지우기의 개념은 제로필이었고 윈도우도 그렇게 작업을 했습니다. [파일 완전 삭제 프로그램들도 제로필, 포맷도 제로필, 너도 나도 제로필, 우리 모두 제로필로 위아더월드!] 그런데 이러한 제로필이 SSD 입장에서는 Erase 가 아닌 그냥 또 다른 Write 작업이 되어 버릴 수 있는 겁니다. 그래서 운영체제는 직접 데이터를 지우지 않고, SSD 에게 알아서 지우라고만 알려주는 것입니다.

## 5. TRIM 명령의 작동 방식

이러한 TRIM 의 작동 방식에는 크게 두 가지를 생각할 수 있습니다.

1. 운영체제에서 파일을 삭제하였다고 알려주면 SSD 에서도 즉시 해당 파일을 삭제.
2. 운영체제에서 파일을 삭제하였다고 알려주면 SSD 자체적으로 표시해 두었다가 적절한 순간에 삭제.

1 의 경우 가장 확실하고 간단하지만, 파일을 삭제할 때마다 그만큼 딜레이가 발생하고, 그로 인해 프리징 현상 등이 발생할 수 있는 단점이 있습니다.

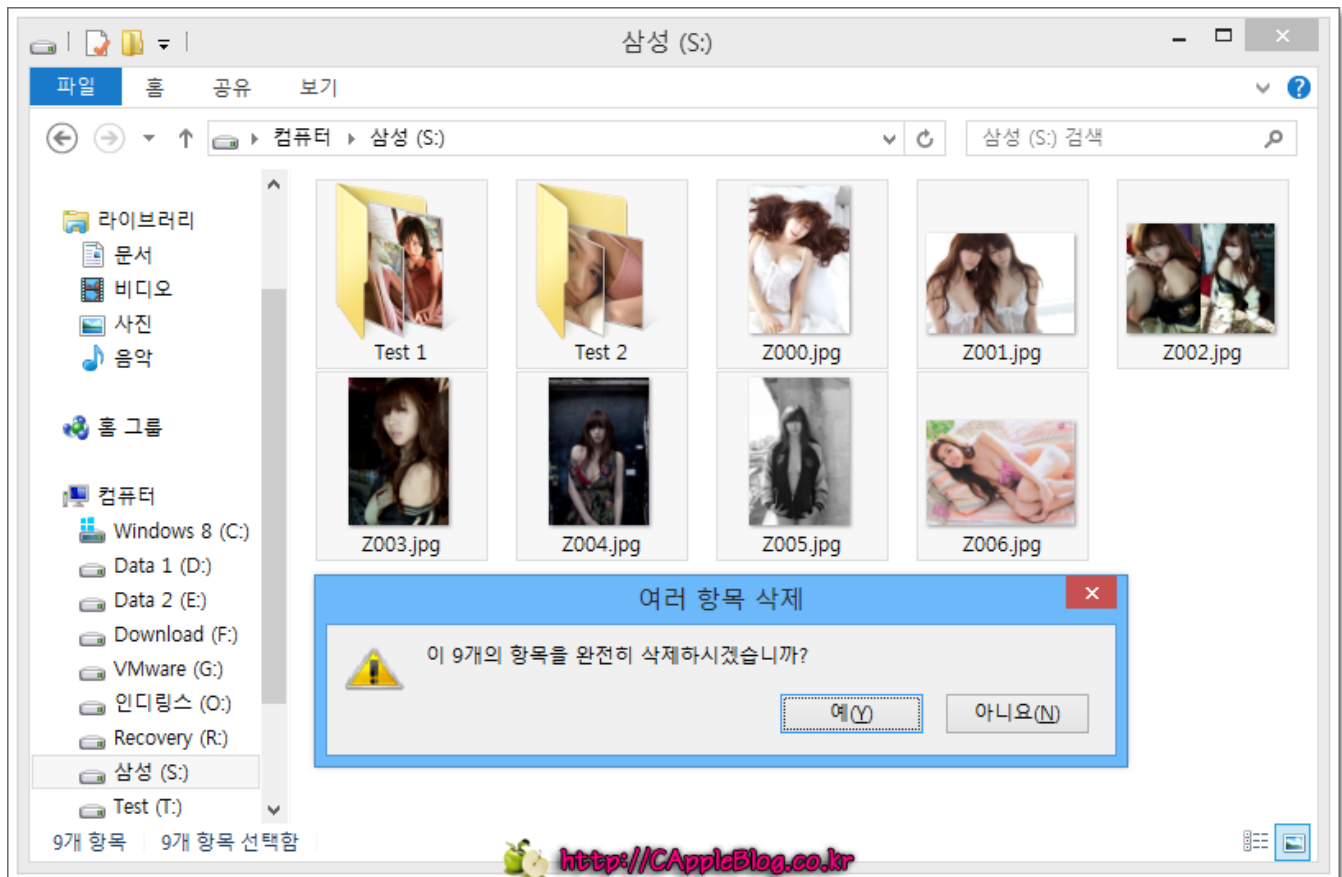
2 의 경우 해당 파일이 삭제된(invalid) 파일이라고 알려주면 해당 페이지에 표시를 하고, 나중에 SSD 가 한가할 때 적절한 타이밍을 잡아 삭제하기에, 삭제시에 딜레이가 발생하지 않는 장점이 있습니다. 하지만 이러한 페이지를 표시하고, 삭제를 위한 최적의 타이밍을 잡기 위한 알고리즘의 적절한 설계가 1 에 비해 굉장히 복잡해지는 문제가 있습니다. 그래서 2 방식의 경우 이러한 알고리즘을 얼마나 효율적으로 아름답게 잘 구성하느냐에 따라 SSD 의 전체적인 성능이 결정된다고도 할 수 있죠. [엔지니어의 피와 땀을 갈아 넣어서 완성시키죠. SSD 에는 공돌이들의 혼이 녹아 있습니다. 요즘은 외계인도 갈아 넣는 듯 하더군요.]

요즘의 SSD 컨트롤러들은 1 보다는 2 의 방식을 주로 사용하고 있으며, 차세대 S-ATA 3.1 규격에서는 Queued Trim Command 를 추가하여 S-ATA 규격 자체에서 TRIM 명령을 최적화하기 위해 준비하고 있습니다.

## 간단 실험!

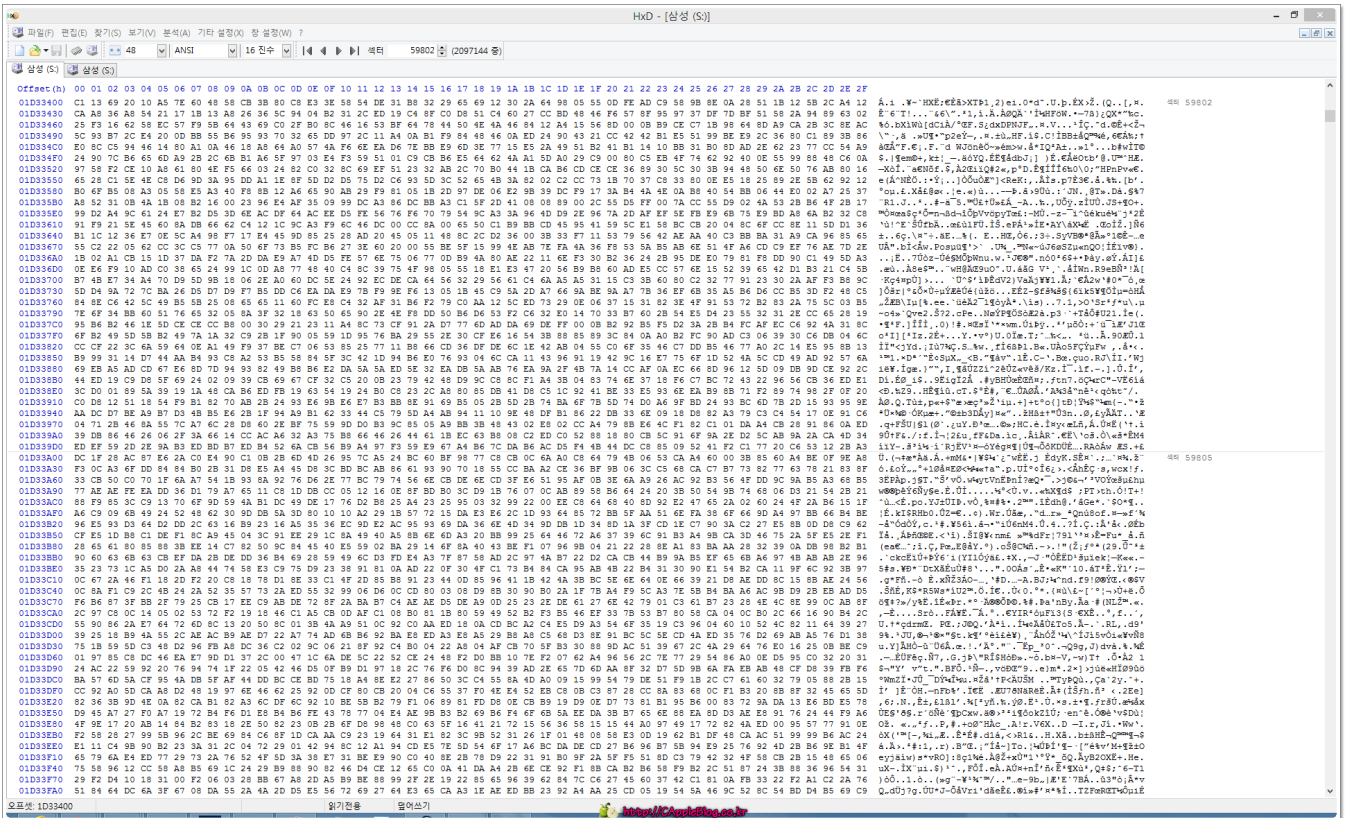
일단 실험은 삼성MCX 컨트롤러를 사용하는 삼성 830 128GB 와 인디링스 Everest 2 컨트롤러를 사용하는 OCZ Vertex4 256GB 를 통해 진행하였습니다. 실험 방법은 간단합니다. 일단 두 SSD 에 소용량의 파티션을 준비한 후 파일을 거의 가득 채우고, 이후 모든 파일을 Shift + Delete 키를 통해 곧바로 삭제하였습니다. 삭제 직후 HxD hex 에디터를 통해 SSD 에 남아 있는 데이터를 주기적으로 확인하여 TRIM 명령이 어느 시점에 실행되며 정상적으로 작동하는지 확인합니다.

### 1. 데이터의 준비 및 삭제

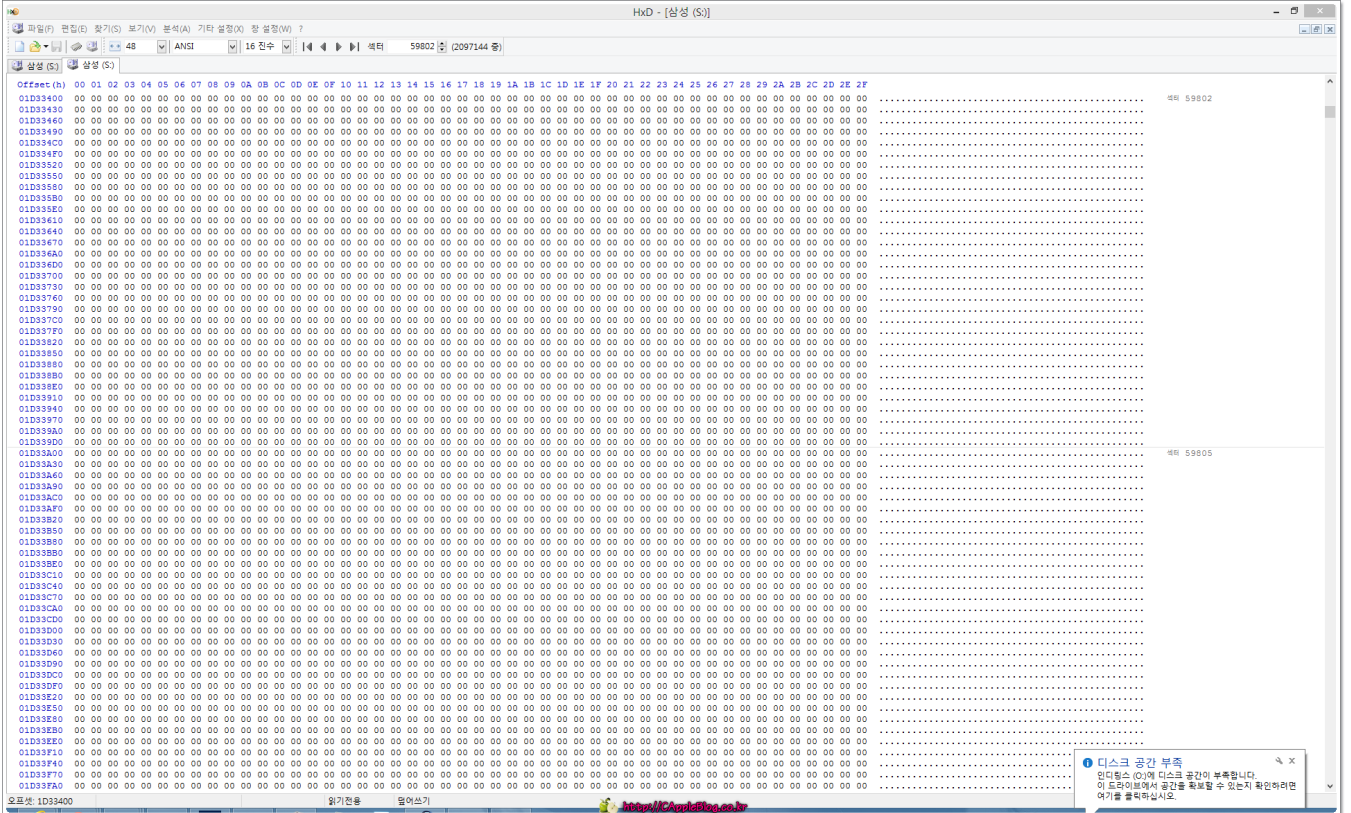


## 2. 삼성 MCX 컨트롤러 - 삼성 830 128GB

삭제 직후



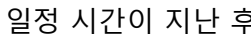
## 일정 시간이 지난 후



## 3. 인디링스 Everest2 컨트롤러 - OCZ Vertex4 256GB

### 삭제 직후





체로 몇십초 이내에 실제로 데이터가 삭제되었습니다. 인텔 제품은 표본이 없어서 어떠런지 모르겠네요. [친구 집에 샌드포스 칩을 쓴 520 이 하나 있긴 한데...]

참고로 실험을 통해 삼성 830 에서 사용된 MLC Cell 의 Empty 값은 00 이며, OCZ Vertex4 에서 사용된 MLC Cell 의 Empty 값은 FF 임을 추가로 알 수 있었습니다. 즉, 이를 통해서도 앞서 말했던 SSD 에서 사용된 플래시 메모리는 종류에 따라 Empty 값이 다르기 때문에, SSD 에서는 제로필이 꼭 페이지의 와이핑(Erase -> Empty)을 의미하지 않는다는 사실을 재차 확인할 수 있었습니다. 말했지만 오히려 제로필이 쓸데없는 Write 작업을 증가시키는 꼴이 될 수도 있습니다.

그러니까 SSD 에서는 가급적 제로필이나 통상적인 보안 삭제 도구를 통해 파일이나 공간을 삭제 하지 않는 것이 좋습니다. 자동 TRIM 기능이 켜져 있으면 알아서 자동으로 데이터가 지워지고, SSD 제조사에서 제공하는 Wiper 툴을 통해 수동 TRIM 을 진행해도 데이터가 지워집니다. 복구 업체는 좀 우울하겠네요.

## 6. 웨어 레벨링과 가비지 컬렉션 - Wear-Leveling, Garbage Collection

플래시 메모리의 셀은 재기록이 가능한 횟수가 정해져 있습니다. 즉, 플래시 메모리는 Write 작업이 일어날 수록 수명이 점점 줄어듭니다. 통상적으로 SLC 의 경우 10 만회, MLC 의 경우 1 만회, TLC 는 1 천회의 재기록이 가능하다고 알려져 있습니다. 근데 이마저도 공정의 미세화로 인해 재기록 가능 횟수가 점차 줄어들고 있는 실정입니다. 가장 저가형인 TLC 의 경우 재기록 가능 횟수가 경악할 정도로 많이 떨어지기 때문에 TLC 메모리를 사용하는 싸구려 USB 플래시 메모리들이 보통 사용한 지 얼마 안 되어 인식 불능이 되는 것도 쉽게 이해할 수 있습니다. 근데 삼성이 이번에 840 에서 대차게 TLC 를 사용했죠. 뭔가 자신이 있으니 그랬겠죠?

아무튼 그래서 SSD 는 특정 셀에 집중된 Write 를 피하기 위해 SSD 차원에서 여러 셀이 골고루 사용될 수 있도록 데이터를 분산 저장하는 **웨어 레벨링(Wear-Leveling)** 기술을 사용하게 됩니다. 간단하게 블록마다 사용된 횟수를 기록하고 그에 맞춰 덜 사용된 블록을 위주로 기록 작업을 진행하는 것입니다. 이건 워낙에 유명하니 한 번쯤은 들어보셨을거라 생각합니다. 그런데 SSD 에서는 이와 더불어 **가비지 컬렉션(Garbage Collection)**이라는 다른 기술도 함께 사용되고 있습니다.

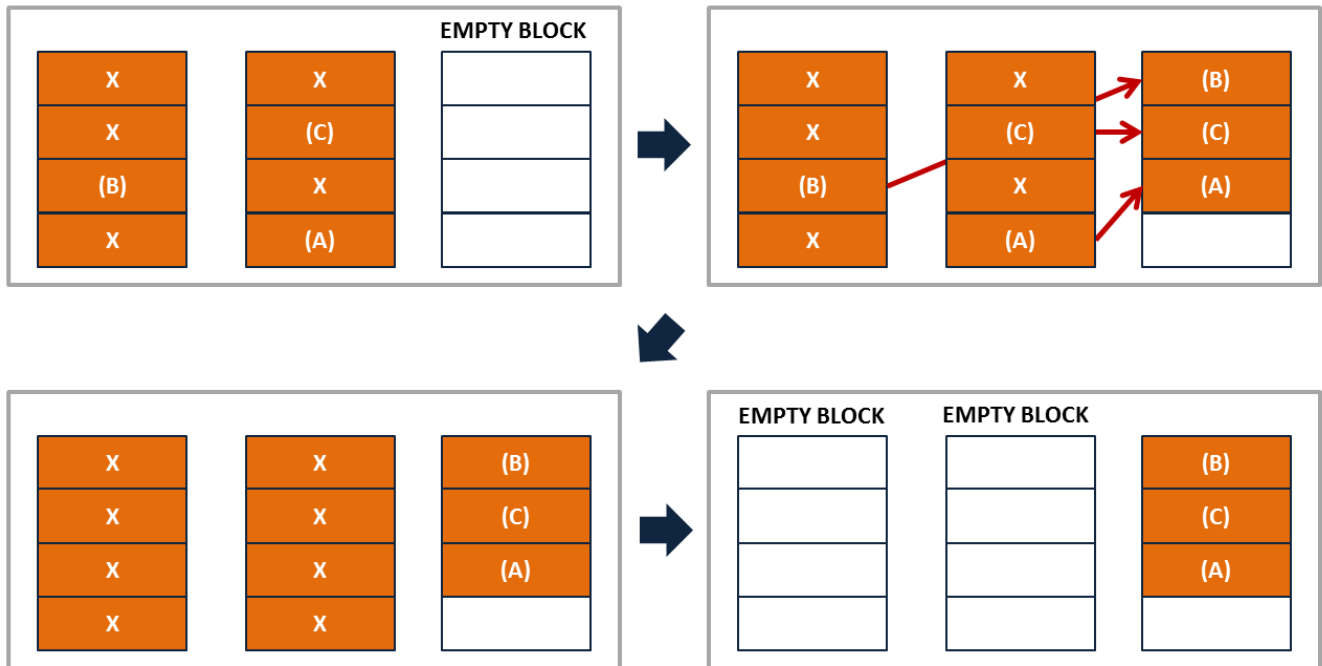
말했다시피 플래시 메모리의 셀은 재기록 가능 횟수 제한이 있기 때문에 Write 작업이 최대한 골고루 이루어져야 합니다.(모든 페이지가 골고루 사용되어야 합니다.) 그리고 지금까지 말했듯이 Erase 작업이 많이 느립니다.

그래서 취한 방법이 무엇이냐면 어떠한 페이지에 재기록 요청 작업이 오면(파일의 수정 등) 해당 요청대로 페이지의 내용을 지우고 쓰는 것이 아니라 해당 페이지는 필요 없는 데이터가 담긴 페이지라고 마크만 한 후(쓰레기 페이지) 다른 비어있는 페이지에 데이터를 기록하는 겁니다. 그리고 해당 페이지의 주소를 마크한 페이지의 주소와 바꿔치기 하는 것이죠. 이것이 바로 웨어 레벨링입니다. [SSD 는 HDD 와 달리 논리적인 섹터 주소와 실제 SSD 의 물리적인 섹터 주소가 일치하지 않습니다. 계속 자리를 바꾸니까요. 이것을 도와주는 것이 FTL(Flash Translation Layer) 입니다.]

이렇게 하면 가뜰이나 느린 Erase 작업을 최대한 회피할 수 있으며 동시에 여러 셀이 골고루 사용되도록도 할 수 있습니다. 대신 SSD 에는 점차 실재론 사용되지 않는 데이터가 담긴 쓰레기 페이지가 쌓이게 됩니다. 이런



식으로 작업을 계속 진행하다보면 어느 시점에 어떠한 블록에는 쓰레기 페이지가 다수 쌓이게 되겠죠? 즉, 유효한 페이지보다 쓰레기 페이지가 더 많은 블록이 점차 늘어나는 겁니다.(꼭 그렇지 않더라도 충분히 Erase 작업을 진행해도 좋을 만큼 쓰레기 페이지가 쌓인) 그럼 적절한 시점을 잡아 그러한 블록들을 파악하여 해당 블록들 내에서 유효한 페이지들은 한 군데로 모아 다른 블록으로 옮기고, 이 과정을 거쳐 쓰레기 페이지만 남은 블록들을 몰아서 Erase 처리를 하는 겁니다.



이것이 바로 SSD 의 가비지 컬렉션입니다. 즉, 우선 웨어 레벨링을 통해 느린 Erase 작업을 최대한 회피해서 성능을 올리고, 다시 이렇게 모아진 쓰레기 페이지들을 가비지 컬렉션을 통해 한 번에 처리하여 효율을 올리는 것이죠. 이것 역시 작업 알고리즘이 중요하며 엔지니어의 눈물로 완성되는 작업입니다. 참고로 이러한 웨어 레벨링과 가비지 컬렉션 작업은 SSD 의 펌웨어 단에서 수시로 실행이 됩니다. [운영체제가 관여하는 작업이 아니라 SSD 가 자체적으로 수행하는 작업]

그런데 생각해보면 가비지 컬렉션을 통해 생성되는 쓰레기 페이지와 TRIM 을 통해 마크된 쓰레기 페이지는 어차피 동일하게 Erase 해야 할 페이지들입니다. 그렇다면 TRIM 을 통해 마크된 페이지들을 가비지 컬렉션을 통해 생성된 쓰레기 페이지들과 함께 처리하면 어떨까요? 즉, TRIM 명령으로 발생한 쓰레기 페이지와 가비지 컬렉션 기능으로 발생한 쓰레기 페이지를 따로 처리하는 것이 아니라, TRIM 명령을 통해 생성된 쓰레기 페이지와 가비지 컬렉션의 쓰레기 페이지를 동일하게 취급하면, 이후 가비지 콜렉터가 이 둘을 한 번에 처리하게 되는 것이죠.

이번 글에서 진행한 TRIM 실험에서 SSD 에서 실제로 파일이 삭제된 시점은 바로 SSD 가 가비지 컬렉션 작업을 진행한 시점이라고 볼 수 있을 듯 합니다.

TRIM 명령을 이해하기 위한 정리는 이정도만 하면 될 듯 합니다. 그럼 이번 글을 마무리 짓도록 하죠.

## 마무리

지금까지 운영체제의 삭제 작업과 SSD 의 TRIM 명령을 기준으로 SSD 와 플래시 메모리의 특성에 대해서 알아보았습니다. 그냥 간단하게 TRIM 은 실제로 파일을 지우는 기능입니다. 라고 한 줄로 끝내면 될 것을 너무 구구절절 정리해버린 기분이 드네요. 그리고 헛갈리지 마셔야 할 것이 이 글에서 설명한 내용들은 쓰기와 관련된 것입니다. 읽기나 접근 속도는 TRIM 이 되든 말든 영향을 받지 않습니다. 즉, 읽기나 접근 속도가 느려지지는 않는다는 겁니다.

아무튼 이 글이 어쩌다가 여기까지 오게 되었는지는 모르겠지만, 이번 글에서 이야기한 플래시 메모리의 주요한 특성을 다시 세 가지로 압축하여 정리하면 아래와 같습니다.

첫 째, 접근(Access), 읽기(Read), 쓰기(Write) 속도는 빠르지만, 상대적으로 지우기(Erase) 속도가 많이 느리다.

둘 째, 덮어쓰기(Overwrite)가 불가능하다. 이미 데이터가 담긴 페이지에 어떠한 데이터를 추가로 기록하기 위해선 무조건 해당 (페이지가 포함된) 블록에 대한 지우기(Erase) 작업이 먼저 이루어져야 한다.

셋 째, 플래시 메모리의 셀은 재기록 가능 횟수가 제한되어 있다.

정리하자면 이러한 특성으로 인해 HDD 시대에는 전혀 고려하지도 않았던 문제들이 발생하였고, 이를 해결하기 위해 다양한 기술들이 SSD 내부에서 사용되고 운영체제가 이를 보조하고 있는 것입니다. 기술은 한계를 만났을 때 그것을 뛰어넘기 위해 발전하죠. 앞으로 얼마나 더 괴물 같은 기술들이 나오게 될 지, 얼마나 많은 엔지니어들의 청춘이 갈릴지 모르겠습니다. 이쯤 하죠.

최대한 쉽게 정리한다고 노력했는데도 여전히 어렵네요. 제 나름대로 이미지들도 만들고 뭐 그러려고 했는데 시간 관계상 많은 이미지들을 다른 곳의 이미지들로 사용한 게 아쉽습니다. 모쪼록 이 글이 SSD 라는 저장매체를 이해하는 데에 조금이라도 도움이 되었으면 좋겠습니다. 이상입니다. ^^