

BIZ & IT —

# Solid-state revolution: in-depth on how SSDs really work

SSDs use a huge grab bag of techniques to make a computer feel "snappy."

LEE HUTCHINSON - 6/4/2012, 8:30 AM



SSDs—how do they work? Not with magnets.

Way back in  
1997, when

---

**SSD Revolution**

dinosaurs roamed the earth and I was working part-time at the local Babbage's for \$4.25 an hour, I scraped together enough

spare change to purchase a 3Dfx Voodoo-based **Diamond Monster 3D** video card. The era of 3D acceleration was in its infancy and the Voodoo chipset was *the* chipset to beat. It all seems a bit silly now, but when I slapped that sucker into my aging Pentium 90 and fired up the new card's pack-in version of **MechWarrior 2**—which had texture-mapping and visual effects that the original 2D version lacked—my jaw hit the floor. I couldn't wait to speed-dial my buddy Matt and tell him that his much-faster Pentium 166 no longer brought all the polygons to the yard.

That video card was the most important PC upgrade I ever made, sparking a total change in my perception of what computers could do. I didn't think I would ever again experience something as significant as that one single upgrade—until the first time I booted up a laptop with a solid-state drive (SSD) in it. Much like that first glimpse of a texture-mapped *MechWarrior 2*, that first fast boot signaled a sea change in how I thought and felt about computers.

**The future of flash memory: tiny (and extremely tough to build)**

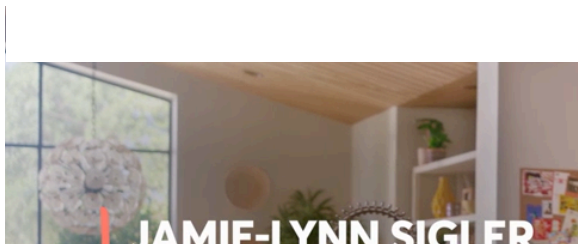
**How do you make the most of your SSD?**

**How SSDs conquered mobile devices and modern OSes**

**Solid-state revolution: in-depth on how SSDs really work**

> [View more stories](#)

---



Join Ars Technica and

## Get Our Best Tech Stories

**DELIVERED STRAIGHT TO YOUR INBOX.**

By signing up, you agree to our user agreement (including the class action waiver and arbitration provisions), our privacy policy and cookie statement, and to receive marketing and account-related emails from Ars Technica. You can unsubscribe at any time.



SIGN ME UP

The introduction of 3D graphics changed our perceptions of computing not because it made colors brighter or virtual worlds prettier—though it did those things and they are awesome—but because it made a smoothly responsive 30 and 60 frames per second gaming experience a standard. Solid-state drives have a similar effect. They're faster than spinning disk, to be sure, but their most important contribution isn't just that they *are* faster, but rather that they make the whole computer *feel* faster. They remove barriers between you and your PC, in effect thinning the glass between you and the things that you're doing with and through your computer.

Solid-state drives are odd creatures. Though they sound simple in theory, they store some surprisingly complex secrets. For instance, compare an SSD to a traditional magnetic hard drive. A modern multi-terabyte spinning hard disk plays tricks with **magnetism and quantum mechanics**, results of decades of research and billions of dollars and multiple Nobel Prizes in physics. The drives contain complex moving parts

manufactured to extremely tight tolerances, with [drive heads moving around](#) just thousandths of a millimeter above platters rotating at thousands of revolutions per minute. A modern solid-state drive performs much more quickly, but it's also a more mundane on the inside, as it's really a hard drive-shaped bundle of [NAND flash memory](#). Simple, right?

However, the controller software powering an SSD does some remarkable things, and that little hard drive-shaped bundle of memory is more correctly viewed as a computer in its own right.

Given that SSDs transform the way computers "feel," every geek should know at least a bit about how these magical devices operate. We'll give you that level of knowledge. But because this is Ars, we're also going to go a lot deeper—10,000 words deep. Here's the only primer on SSD technology you'll ever need to read.

## Varying degrees of fast

It's easy to say "SSDs make my computer fast," but understanding *why* they make your computer fast requires a look at the places inside a computer where data gets stored. These locations can collectively be referred to as the "memory hierarchy," and they are described in great detail in the classic Ars article "[Understanding CPU Caching and Performance](#)."

It's an axiom of the memory hierarchy that as one walks down the tiers from top to bottom, the storage in each tier becomes larger, slower, and cheaper. The primary measure of speed we're concerned with here is *access latency*, which is the amount of time it takes for a request to traverse the wires from the CPU to that storage tier. Latency plays a tremendous role in the effective speed of a given piece of storage,

because latency is dead time; time the CPU spends waiting for a piece of data is time that the CPU isn't actively working on that piece of data.

---

Advertisement

---

---

The table below lays out the memory hierarchy:

<b>LEVEL</b>	<b>ACCESS TIME</b>	<b>TYPICAL SIZE</b>
Registers	"instantaneous"	under 1KB
Level 1 Cache	1-3 ns	64KB per core
Level 2 Cache	3-10 ns	256KB per core
Level 3 Cache	10-20 ns	2-20 MB per chip
Main Memory	30-60 ns	4-32 GB per system
Hard Disk	3,000,000-10,000,000 ns	over 1TB

---

At the very top of the hierarchy are the tiny chunks of working space inside a CPU where the CPU stores things it's actively manipulating; these are called *registers*. They are small—only a few hundred bytes total—and as far as memory goes, they have the equivalent of a Park Avenue address. They have the lowest latency of any segment of the entire memory hierarchy—the electrical paths from the parts of the CPU doing the work to the registers themselves are unfathomably tiny, never even leaving the core portion of the

CPU's die. Getting data out in and out of a register takes essentially no time at all.

Adding more registers could potentially make the CPU compute faster, and as CPU designs get more advanced they do indeed tend to gain more (or larger) registers. But simply adding registers for the sake of having more registers is costly and complicated, especially as software has to be recompiled to take advantage of the extra register space. So data that the CPU has recently manipulated but that isn't being actively fiddled with at the moment is temporarily placed one level out on the memory hierarchy, into level 1 cache. This is still pricey real estate, being a part of the CPU die, but not as pricey as the registers. In a modern CPU, getting data out of the L1 cache takes three or four cycles (typically around a nanosecond or so) compared to zero cycles for the registers. The trade-off for that slower performance is that there's a lot more space in this tier—up to 32KB of data per CPU core in an Intel Ivy Bridge i7 CPU.

Data that the CPU expects to access again shortly is kept another level out, in the level 2 cache, which is slower and larger, and which carries still more latency (typically between 7 and 20 cycles).

Modern CPUs have level 3 caches as well, which have higher latencies again, and which can be several megabytes in size.

Even further down the hierarchy is the computer's main memory, which has much higher effective latency than the CPU's on-die cache. The actual RAM chips are rated for very low latency (DDR2 DRAM, for example, is typically rated for five nanoseconds), but the components are physically distant from the CPU and the effective latency is therefore higher—usually between 40 and 80 nanoseconds—because the electrical signals from the CPU

have to travel through the motherboard's traces to reach the RAM.

At the bottom of the hierarchy sits our stalwart hard disk, the repository of all your programs, documents, pictures, and music. All roads lead here. Any time a program is executed, an MP3 is played, or any kind of data needs to be viewed or changed by you, the user, the computer calls on the disk to deliver it up.

Disks these days are large, but they are also glacially slow compared to the other tiers in the memory hierarchy, with latency a million times higher than the previous tier. While waiting for main memory to respond, the processor might have nothing to do for a few dozen cycles. While waiting for the disk to respond, it will twiddle its thumbs for *millions* of cycles.

Worse, the latency of a spinning hard disk is variable, because the medium itself is in motion. In order to start an application like, say, Google Chrome, the hard disk may have to read data from multiple locations, which means that the drive heads have to seek around for the right tracks and in some cases even wait whole milliseconds for the correct blocks to rotate underneath them to be read. When we're defining latency in terms of billionths of a second in previous tiers, suddenly having to contend with intervals thousands of times larger is a significant issue. There are many tricks that modern computers and operating systems do to lessen this latency, including trying to figure out what data might be needed next and preemptively loading that data into RAM before it's actually requested, but it's impossible to overcome all of the latency associated with spinning disks.

---

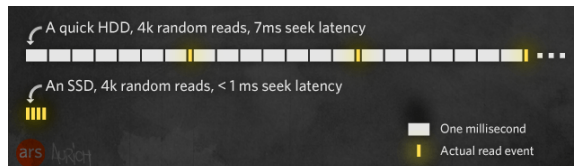
On one hand, human beings like us don't operate in terms of milli-, micro-, or nanoseconds, at least not without the aid of serious drugs. A thousandth of a second is the same to us as a billionth of a second—both are intervals so small that they might as well be identical. However, with the computer doing many millions of things per second, those tiny fractions of time add up to very real subjective delays, and it can be frustrating when you click on Microsoft Word and stare at a spinning "Please wait!" cursor for seconds at a time. Waiting on the computer while it drags something off of a slow hard disk is disruptive to workflow and can be a jarring experience, especially if you've got a rapidly derailing train of thought barreling through your head that you need to write down.

Solid-state drives provide an immediate boost to the subjective speed of the computer because they take a big chunk out of the largest amount of latency you experience. Firstly and more obviously, solid-state drives don't have moving heads and rotating platters; every block is accessible at the same speed as every other block, whether they're stored right next to each other or in different physical NAND chips. Reading and writing data to and from the solid-state drive is faster as well, so not only does the computer have to wait fewer



milliseconds for its requests to be serviced, but the solid-state drive can also effectively read and write data faster. Quicker responses (lower latency) plus faster transfer speeds (more bandwidth) mean that an SSD can move more data faster—its *throughput* is higher.

Even just halving the latency of a spinning disk (and SSDs typically do far more than that) provides an incredible subjective improvement to the computing experience. Looking at the higher tiers in the memory hierarchy, it's easy to see why. If, for example, a sudden breakthrough in RAM design decreased the effective latency to and from a system's RAM by a factor of 10x, then calls to and from RAM would drop from a best case of 60ns to 6ns. Definitely impressive, but when looked at in terms of the total delay during an I/O operation from CPU to RAM to disk, there's still so much time spent waiting for the disk that it's an insignificant change. On the other hand, cutting the disk's effective latency from 5-10 milliseconds for a random read to less than a single millisecond for a random read—because any block of an SSD is always as readable as any other block, without having to position heads and wait for the platter to spin—you've just knocked out a tremendous percentage of the total amount of time that entire "CPU to RAM to disk" operation takes. In other words, the speed increases provided by an SSD are targeted right at the longest chain in the memory hierarchy.



Latency affects throughput by letting you read more data in a smaller amount of time. Here, the spinning disk spends most of its time waiting on the platter and heads to find the right data to be read.

---

Now, a solid-state drive isn't going to always be faster than a spinning hard disk. You can define and run benchmarks which highlight a hard disk's advantages over an SSD; a synthetic benchmark that repeatedly writes and rewrites data blocks on a full SSD without giving the SSD time to perform garbage collection and cleaning can overwhelm the SSD controller's ability to manage free blocks and can lead to low observed performance, for example (we'll get into what garbage collection is and why it's important in just a bit).

But in everyday use in the real world, when performing under an organic workload, there are almost no areas where simply having an SSD doesn't make the entire computer seem much faster.

So how does an SSD actually work? Let's take a peek inside.

---

---

---

## Inside the box

Unlike spinning hard disks, which read and write data magnetically, an SSD reads and

writes to a medium called NAND flash memory. Flash memory is non-volatile, which means that it doesn't lose its contents when it loses power like the DRAM used in your computer's memory does. It's the same stuff that lives inside your smartphones, mp3 players, and little USB thumb drives, and it comes from the same assembly lines. What makes an SSD so much faster than a thumb drive is a combination of how the NAND chips are addressed, and the various caching and computing shortcuts that the SSD's built-in controller uses to read and write the data.

***"A given amount of storage built out of NAND flash would take up about 60 percent less physical space than the same structure built out of NOR flash."***

Flash memory's non-volatility comes from the types of transistors used in its makeup —namely, **floating gate transistors**. Normal transistors are simple things; they're essentially just electronically controlled switches. Volatile memory, like a computer's RAM, uses a transistor coupled with a capacitor to indicate a zero or a one. The

transistor is used to transfer charge to or drain charge from the capacitor, and that charge must be refreshed every few microseconds. A floating gate transistor, on the other hand, is more than just a switch, and doesn't have a needy external capacitor to hold a charge. Rather, a floating gate transistor creates a tiny cage (called the *floating gate*), and then encourages electrons to migrate into or out of that cage using a particular kind of quantum tunneling effect. The charge those electrons represent is permanently trapped inside the cage, regardless of whether or not the computer it's in is currently drawing power or not.

It's easy to see how a floating gate transistor could form the basis for a storage medium. Each transistor can store a single bit—a "1" if the cell is uncharged, a "0" if it is charged—so just pack a load of them together and there you are. The cells are customarily organized into a grid and then wired up. There are two types of flash memory used for storage today: NOR and NAND. Both have a grid of cells; how they differ is their wiring.

## NOR flash

The simplest way to use transistors is to wire each row together and each column together, to allow each individual bit position to be read. This particular layout is used in NOR flash. The circuits connecting each row are called "word lines," while the ones for each column are "bit lines." Read operations are very simple with this arrangement: apply a voltage to each word line, and the bit lines will then show a voltage (or not) depending on whether each cell stores a 0 or a 1. This is similar to how volatile SDRAM is laid out.

To understand how this works in a bit more detail, we have to first understand floating

gate transistors. A normal transistor has three connections named, depending on the technology being used, "base, collector, and emitter," or "gate, drain, and source." Floating gate transistors use the second set of terms.

When a voltage is applied to the gate (or the base, in the other kind of transistor), a current can then flow from the source to the drain (or from the collector to the emitter). When low voltages are applied to the gate, the voltage flowing from source to drain varies in proportion to the gate voltage (so a low gate voltage causes a low flow from source to drain, a high voltage causes a high one). When the gate voltage is high enough, the proportional response stops. This allows transistors to be used both as amplifiers (a small signal applied to the gate causes a proportionately larger signal at the source) and as switches (use only a high voltage or a zero voltage at the gate, and there's either a high current or no current between source and drain). (There's a truly excellent video showing how a transistor works in greater detail [available on YouTube.](#))

---

Advertisement

---

The floating gate sits between the gate and the rest of the transistor, and it changes the behavior of the gate in an important way. The charge contained in the floating gate portion of the transistor alters the *voltage threshold* of the transistor. When the gate

voltage is above a certain value, known as  $V_{\text{read}}$ , and typically around 0.5 V, the switch will always close. When the gate voltage is below this value, the opening of the switch is determined by the floating gate.

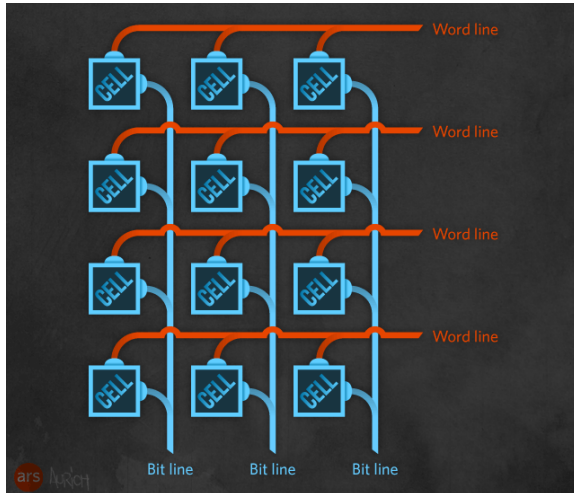
If the floating gate has no charge, then a low voltage applied to the gate can still close the switch and allow current to flow from source to drain. If the floating gate *does* store a charge, then the full  $V_{\text{read}}$  voltage needs to be applied to the gate for the current to flow. That is, whether or not the float gate contains a charge changes how much voltage must be applied to the transistor's gate in order for it to conduct or not conduct.

In the grid of cells, the word lines are connected to the transistors' gates. The bit lines are attached to their drains. The sources are connected to a third set of lines, called the source lines. As with the bit lines, all the transistors in the same column have their sources connected together.

We can read the contents of the cell by applying a low voltage to the gates, and seeing if a current flows. Specifically, the word line to which that cell is connected is energized to some voltage below  $V_{\text{read}}$ . Current will flow through the cell from the source line to its connected bit line if and only if the cell's float gate contains no charge (a 1). If the float gate contains a charge (a 0), the voltage threshold of the whole cell is too high for the small voltage to overcome. The bit line connected to that cell is then checked for current. If it has a current, it's treated as a logical 1; if it doesn't, it's a logical 0.

This arrangement of transistors is used in NOR flash. The reason it's named NOR is that it resembles a **logical NOR**—low current on the word line is NOR-ed against the charge in the float gate. If you apply  $V_{\text{read}}$  and the float gate has no charge, then the

bit line shows charge (0 NOR 0 = 1), whereas the floating gate won't conduct the low read current if it contains a charge (0 NOR 1 = 0).

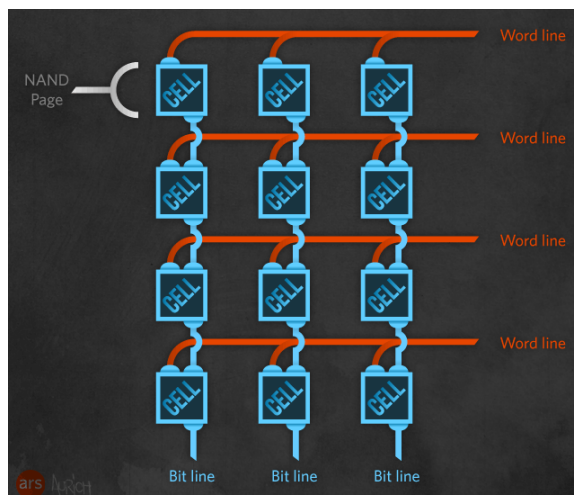


A simplified diagram showing a chunk of NOR flash, and how the cells are connected.

NOR flash has a big drawback. Though the design is *conceptually* simple, the chips themselves are quite complex because a big chunk of NOR flash is taken up with all those individual word line and bit line connections—each transistor has to have them and it results in a great deal of connectivity. In some applications where the ability to read and write in single-bit increments is a requirement, NOR flash fits the bill; for replacing hard disk drives, though, we don't need that extreme granularity of access. Hard drives aren't addressable down to the nearest byte; instead, you can only read and write whole sectors (traditionally 512 but increasingly 4096 bytes) at a time.

## NAND flash

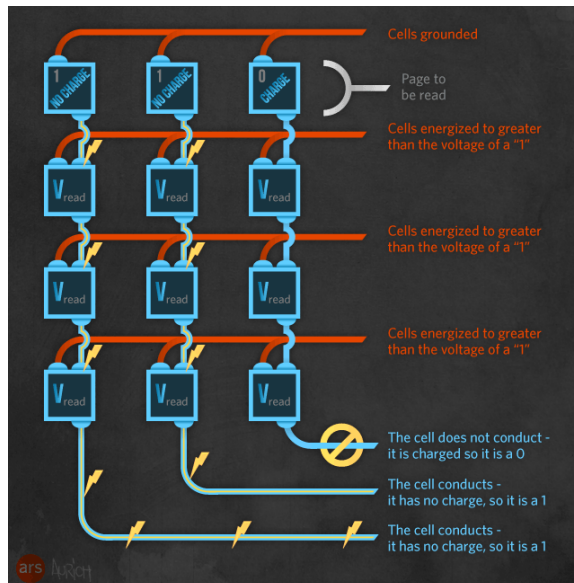
So on to a new arrangement, one that reduces all those connecting wires. Start with the same grid of transistors and the same word lines connecting the gates of each row. However, the connection of the bit lines changes. Each transistor in a column is connected in series, the drain of one feeding in to the source of the next. The first transistor in the column is connected to a regular transistor, the source select, and the source select to the source line. The last transistor in the column is then connected to another regular transistor, the bit line select, and that is connected to the bit line. The source select transistors and bit line select transistors all have their gates tied together, so they conceptually add extra rows above and below the word lines.





A simplified diagram showing a NAND flash block, made up of pages, and how the cells are connected. Note serial bit lines instead of parallel.

To read an individual bit within the grid, the bit line select and source select transistors are both turned on. Then, all the word lines that you're *not* reading have  $V_{\text{read}}$  applied to them, forcing the transistors to conduct regardless of whether they have a charge in their floating gates. Only the word line of the bit that you're interested in has the lower voltage applied to it. If that bit has a stored charge (and hence is a logical 0), the transistor will remain open, and no current will flow through the chain of transistors. If it has no stored charge (and hence is a logical 1), the transistor will close, and a current will flow. The presence or absence of a current on the bit line is detected and treated as a 1 or 0, respectively. All the bit lines will deliver a signal simultaneously, in parallel.



· [BIZ & IT](#) [TECH](#) [SCIENCE](#) [POLICY](#) [CARS](#)

· [GAMING & CULTURE](#) [STORE](#) [FORUMS](#)

SUBSCRIBE

SIGN

IN

A typical flash memory grid has 32 to 256 columns (and hence 32 to 256 bit lines) and

4,096 to 65,536 rows (and hence 4,096 to 65,536 word lines). The total grid is called a *block*, and each row is called a *page* (in practice, the pages are a little larger still, to include error-correction and bookkeeping data).

This design is called NAND flash. Again, this is a reference to its similarity to a kind of logic gate, a **NAND logic gate**, meaning "not and" or "negated and."

Connecting the columns of flash cells to each other in series eliminates a tremendous amount of structure versus NOR flash; a given amount of storage built out of NAND flash would take up about 60 percent less physical space than the same structure built out of NOR flash.

But there's a downside, and that's the lost ability to read and write each bit individually. NAND flash can only read and write data one page—one row—at a time. Contrary to conventional wisdom, there's nothing inherent to the layout of NAND flash that prevents reading and writing from individual cells; rather, in sticking with NAND flash's design goal to be simpler and smaller than NOR flash, the standardized commands that NAND chips can accept and understand are structured such that pages are the smallest addressable unit. This saves on silicon space that would otherwise be needed to hold more instructions and the ability to hold a cell-to-page map.

Now let's put all this memory to use as storage and see how it works.

---

## Reading pages, erasing blocks

We know that a page of NAND flash is physically made up of a row of cells and is the smallest thing that can be read from or written to an SSD, so how much data are we talking about? In a modern SSD with 25 or 20 nm elements, the page size is 8192 bytes. This fits nicely with most modern file systems, which often use cluster sizes of 4096 or 8192 bytes. Since the operating system deals in terms of clusters and the disk itself deals in terms of pages, having these two units of allocation being either the same or convenient multiples of each other helps ensure that the data structures an operating system commits to disk are easily accommodated by the drive's layout.

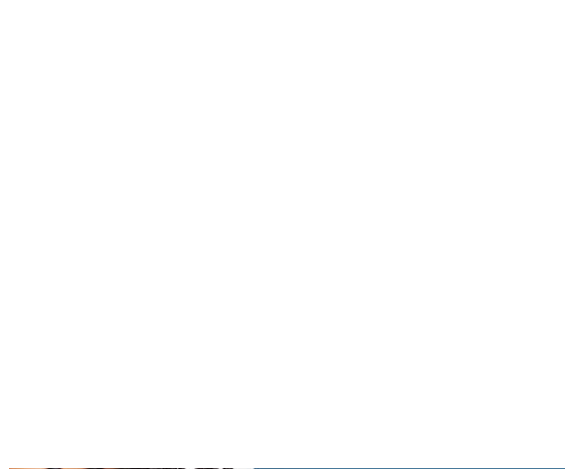
***"The SSD can get slower and slower as it ages."***

While SSDs can read and write to individual pages, they cannot *overwrite* pages. A freshly erased, blank page of NAND flash has no charges stored in any of its floating gates; it stores all 1s. 1s can be turned into 0s at the page level, but it's a one-way process (turning 0s back into 1s is a potentially dangerous operation because it uses high voltages). It's difficult to confine the effect only to the cells that need to be altered; the high voltages can cause changes to adjacent cells. This can be prevented with tunneling inhibition—you apply a very large amount of voltage to all the surrounding cells so that their electrons don't tunnel away along with the targeted cells—but this results in no small amount of stress on the cells being erased. Consequently, in order to avoid corruption and damage, SSDs are only erased in increments of entire blocks, since energy-intensive tunneling inhibition isn't necessary when you're whacking a whole bunch of cells at the same time. (There's a Mafia joke in here somewhere, I'm sure of it.)

Incidentally, while NOR flash allows bit-wise writing, it retains the other constraints: 1s can only be overwritten with 0s, and resetting 0s back to 1s requires erasing whole blocks at a time, again due to the high voltages and risk of damaging adjacent cells if it were performed at a smaller granularity.

The fact that you can read and write in pages but only erase in blocks leads to some odd behavior when compared to traditional storage. A magnetic hard disk can always write wherever it likes and update data "in-place." Flash storage can't. It can (essentially) only write to empty, freshly erased pages.

---



---

The most obviously bad side effect of this kind of scheme is that, unless the SSD has an available erased page ready and waiting for data, it can't immediately perform a write. If it has no erased pages, it will have to find a block with unused (but not yet erased) pages, erase the *entire block*, then write the block's old contents out along with the new page.

This means that the SSD can get slower and slower as it ages. When you pull your shiny new SSD out of the box and plug it in, it's full of erased pages. As you begin copying files to it, it begins busily writing out those files in pages, very quickly, making you happy you purchased it. Hooray! SSDs don't overwrite data, though, so as you change files and delete files and copy new files in, the changed and deleted files aren't actually changed or deleted—the SSD controller leaves them right where they are and writes in the changes and the new files in fresh pages.

In a current-generation SSD with 8192-byte pages, a block can be made up of as many as 256 separate pages, meaning that to write a tiny 8KB file, the SSD must actually first copy two whole megabytes of data into cache, then erase the whole block, then re-write most or all of the entire 2 MB. It obviously takes longer to read, erase, and rewrite 2MB than to simply write 8KB, and

early generation solid-state drives developed a reputation for "getting slower" as they aged, precisely because they ran out of free pages and had to resort to these Tetris-like shenanigans when doing any writing. The SSD is constantly worrying about these things and doing everything it can to fix them; we'll get deep into how this works in just a few more sections.



Here, a block starts with two free pages and one change that must be made to a page. The change goes to a new page, and the old page is marked stale. When a new file comes in and there aren't enough free pages, the whole block is read into cache or RAM and reordered, while the controller erases the entire block, then writes the entire block back out. This is why under certain circumstances, full or old SSDs can seem slow.

## MLC, easy as 1-2-3

Floating gate transistors, with their ability to store a little bit of charge, provide the core storage mechanism for flash. But they're a little more subtle than we previously described. They don't just store *some* charge or *no* charge: they can store a variable amount of charge. And this charge can be detected, because the more charge they store, the more voltage is needed to make them switch.

---

This gives rise to another dichotomy in flash storage: MLC and SLC, which stand for "Multi-Level Cell" and "Single-Level Cell," respectively. The "single" and "multi" refer to the number of different charge levels an individual cell can store. With only a single charge level, a cell can only contain one bit; it either has a stored charge or it doesn't. But with four levels, a cell will have one charge level, and hence one voltage threshold corresponding to 11, another for 10, a third for 01, and a fourth for 00—a total of two bits of storage. With eight levels, it could store three bits.

The use of multiple threshold voltages makes reading more complex. To determine which charge level is stored, a couple of schemes are possible. The flash system can test each possible threshold voltage in turn, to see which one is enough to make the transistor turn on. Alternatively, it can measure the current coming from the cell and compare this to known reference. Both SLC flash, with one charge level per cell, and MLC flash, with several, are widely used.

SLC cells are more reliable and less complex with a commensurately lower error rate because less can go wrong. They are also faster, because it's easier to read and write a single charge value to a cell than it is to play games with voltages to read one of several, and the cells can experience more

write cycles before they go bad. This combination of greater reliability and greater cost means that today you usually find SLC solid-state drives in enterprise applications, running in large servers or even larger arrays. A common SLC SSD in the enterprise is the STEC [ZeusIOPS](#), a 3.5" form factor SSD which usually comes with a Fibre Channel or SAS bus connector and which you'd find in something like [this](#). Don't expect to be using this kind of drive at home; even if you happen to have a computer that can use SAS drives, street price on a SAS-flavored 200GB ZeusIOPS is at least \$3,000 USD. The Fibre Channel variants cost considerably more.

MLC drives can store more information in the same number of cells. Two bits, with four discrete voltage thresholds per floating gate, are common for today's consumer and enterprise drives, and floating gates with eight levels/three bits are in the works. This increase in storage density brings with it an increase in complexity and write cycles, which in turn brings an increase in error rates and a decrease in how long each flash cell lasts before dying. However, the cost of MLC drives is fractional compared to SLC drives, and so in the consumer space, this is what we buy to put in our computers because this is what is affordable. Practically every consumer solid-state drive sold today is MLC; there are a number of "enterprise" MLC SSDs (like the Samsung [SM825](#)) which aim to provide a higher degree of reliability and longevity than standard MLC SSDs without as much of an SLC price premium.

Wait a second, though—"before dying"? "Longevity"? That sounds... somewhat ominous. And indeed, it can be.



**LEE HUTCHINSON**

Lee is the Senior Technology Editor, and oversees story development for the gadget, culture, IT, and video sections of Ars Technica. A long-time member of the Ars OpenForum with an extensive background in enterprise storage and security, he lives in Houston.

---

Advertisement

---



## Unsolved Mysteries Of Quantum Leap With Donald P. Bellisario

Today "Quantum Leap" series creator Donald P. Bellisario joins Ars Technica to answer once and for all the lingering questions we have about his enduringly popular show. Was Dr. Sam Beckett really leaping between all those time periods and people or did he simply imagine it all? What do people in the waiting room do while Sam is in their bodies? What happens to Sam's loyal ally AI? 30 years following the series finale, answers to these mysteries and more await.



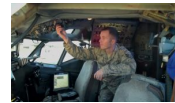
Unsolved Mysteries Of Quantum Leap With Donald P. Bellisario



Unsolved Mysteries Of Warhammer 40K With Author Dan Abnett



SITREP: F-16 replacement search a signal of F-35 fail?



Sitrep: Boeing 707

[+ More videos](#)

[← PREVIOUS STORY](#)

[NEXT STORY →](#)

## Related Stories

Intel Optane Memory: How to make revolutionary technology totally boring

Intel's 3rd-generation Xeon Scalable CPUs offer 16-bit FPU processing

Intel's Optane H20 is the latest attempt at "hybrid" laptop storage

Intel's new flash tech to bring back Turbo Memory, for real

## Top Doctor: If You Eat Avocado Every Day, This Is What Happens

GundryMD

[Learn more](#)

## These Barefoot Shoes are Leaving Neuropathy Experts Baffled

Barefoot Vitality

## People Born 1942-1971 Are Due a Large Surprise

TheWalletGuru

[Read More](#)

## MD: Building Muscle After 60 Comes Down To This 1 Thing

primenutritionsecrets.com

## 3 Toxic Foods For Dogs: The One Meat You Should Never Feed Your Dog

Dog Food Discovery

[Learn more](#)

## The Bra Designed by A 70-year-old Grandma Is Taking California By Storm!

Libiyi

[Get Offer](#)

---

# Today on Ars

*Secrets of the Octopus* takes us inside the world of these “aliens on Earth”

The fungi in our guts can make cases of Covid worse

War never changes: A Fallout fan’s spoiler-laden review of the new TV series

Why are groups of university students modifying Cadillac Lyriq EVs?

The GMO tooth microbe that is supposed to prevent cavities

Daniel Dennett, philosophical giant who championed “naturalism,” dead at 82

It’s cutting calories—not intermittent fasting—that drops weight, study suggests

CNN, record holder for shortest streaming service, wants another shot

STORE  
SUBSCRIBE  
ABOUT US  
RSS FEEDS  
VIEW MOBILE SITE

CONTACT US  
STAFF  
ADVERTISE WITH US  
REPRINTS

## NEWSLETTER SIGNUP

Join the Ars Orbital Transmission mailing list to get weekly updates delivered to your inbox. [Sign me up →](#)

