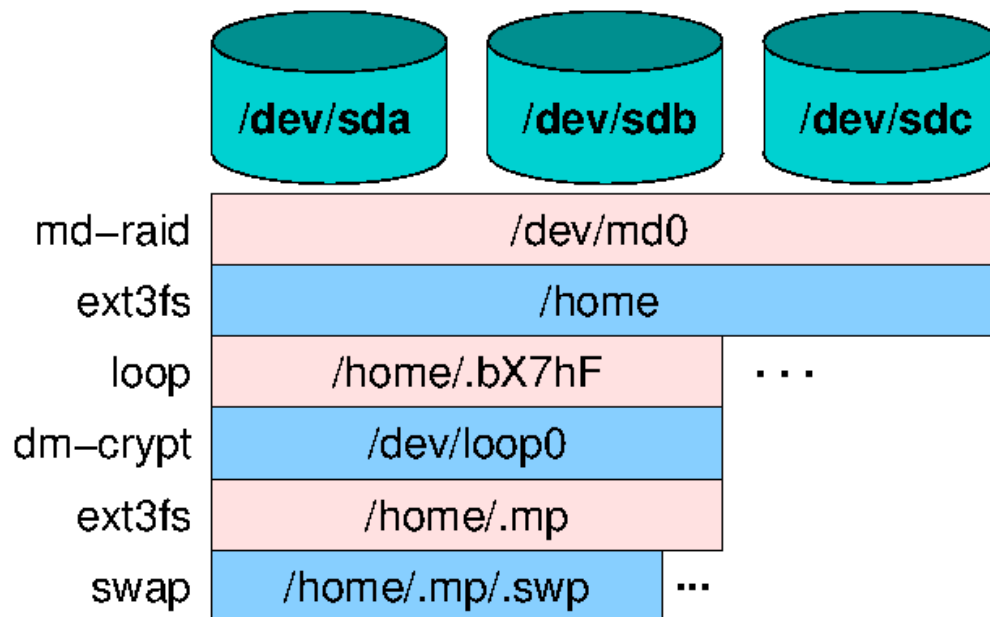


# Storage Management for Privacy and Resiliency

Chad D. Kersey

# Overview

- Creating disk image files
- Creating loopback devices
- RAID
  - What and why
  - RAID levels
  - Disclaimers
  - Using mdadm
- Encryption
  - Why privacy matters
  - About entropy
  - Choosing a Key
  - Why encrypt swap?
  - Encrypted storage in the cloud
  - Using dm-crypt
  - Setting up swap
- Demo



# Creating Disk Image Files

To create a large (100GB), empty sparse file:

```
$ dd if=/dev/zero of=MY_FILE count=0 bs=1G seek=100
```

To overwrite this with (low-quality) random bits:

```
$ shred -n 1 MY_FILE
```

To create a file full of (high-quality) random bits (this will take a very long time):

```
$ dd if=/dev/random of=MY_FILE bs=1G count=100
```

To put a filesystem on a device or file:

```
$ mkfs.ext3 MY_FILE_OR_DEVICE
```

To mount this filesystem:

```
$ sudo mount MY_FILE_OR_DEVICE mountpoint/
```

# Loopback Devices

Linux loopback devices allow files to be treated like block devices:

To list all active loopback devices ("loops"):

```
$ losetup -a
```

To create a new loopback device using image file MY\_IMG:

```
$ losetup /dev/loop0 MY_IMG
```

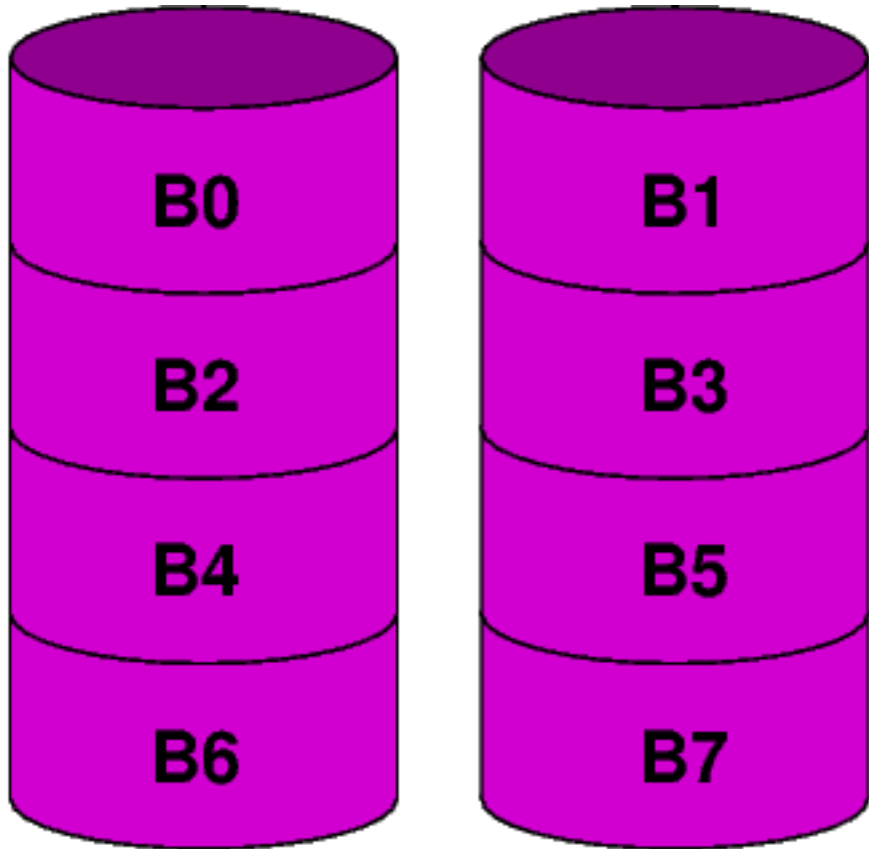
To delete a loopback device:

```
$ losetup -d /dev/loop0
```

# RAID: Overview

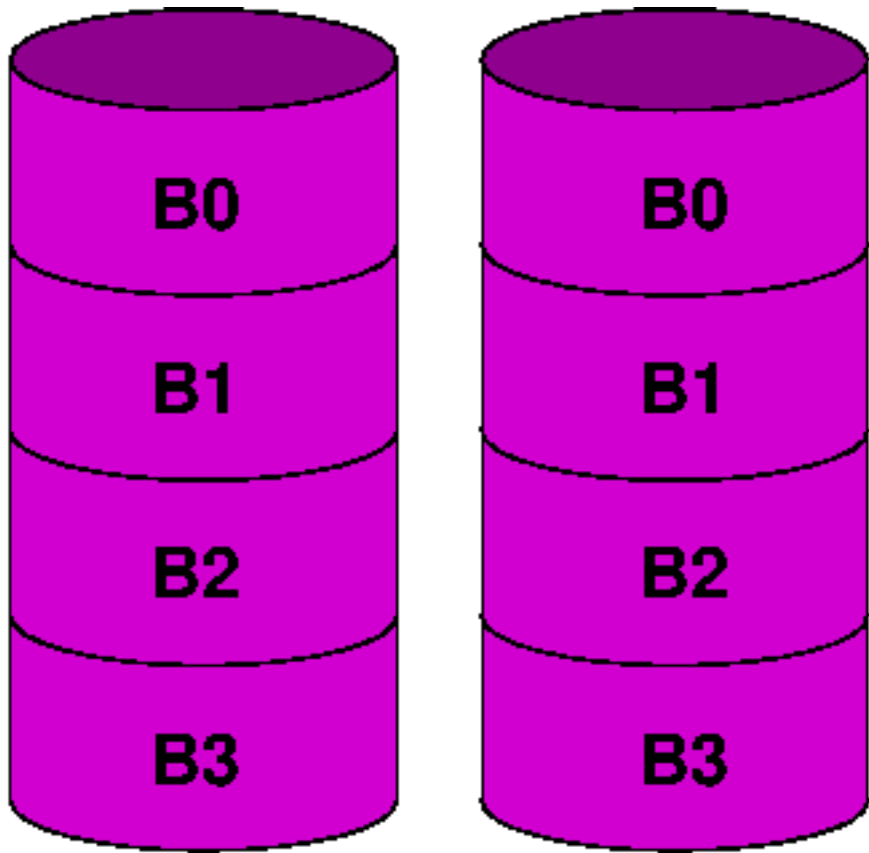
- Redundant Array of In(expensive|dependent) Disks
- Improves reliability and performance compared to JBOD or concatenation.
- Expensive hardware is available for the high-end server market and some motherboards support RAID too, but Linux provides support in software.
- The point: Giant filesystems and multiyear uptimes. Especially now that SATA's brought hotswap to the masses.
- Say "RAID Array" as much as you can.

# RAID Levels: RAID 0



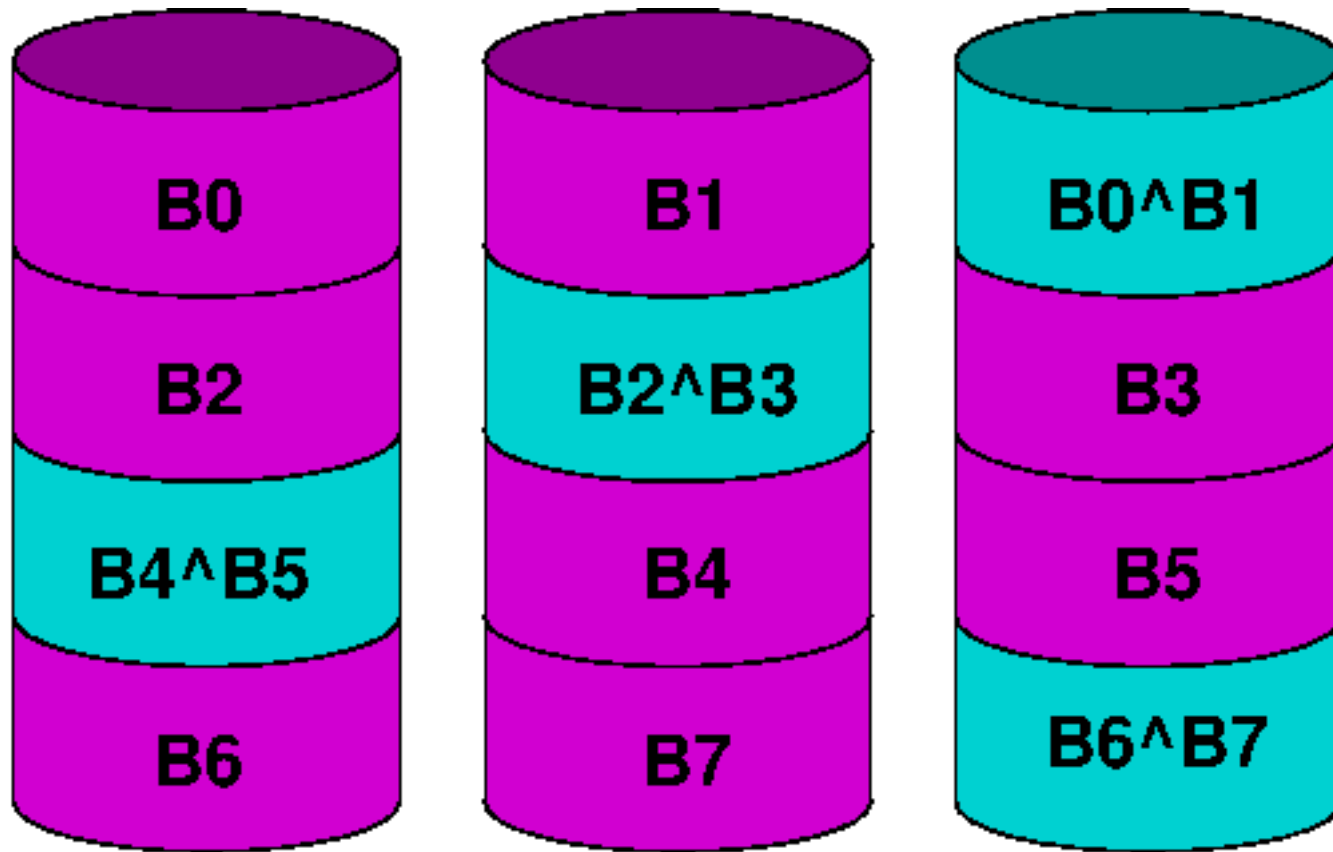
- Simple block-level striping
- No redundancy

# RAID Levels: RAID 1



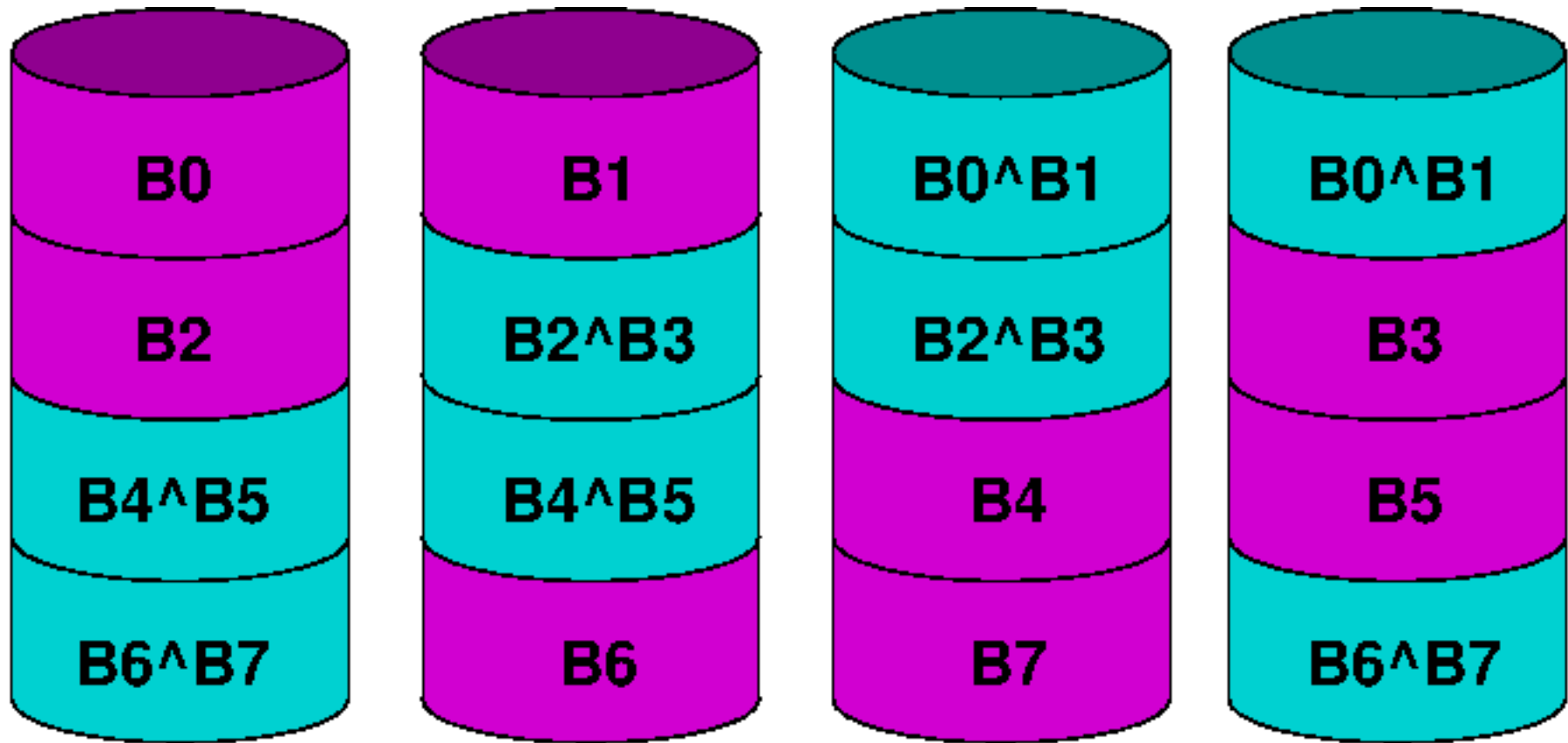
- Simple mirroring
- Each additional disk increases reliability but not space.

# RAID Levels: RAID 5



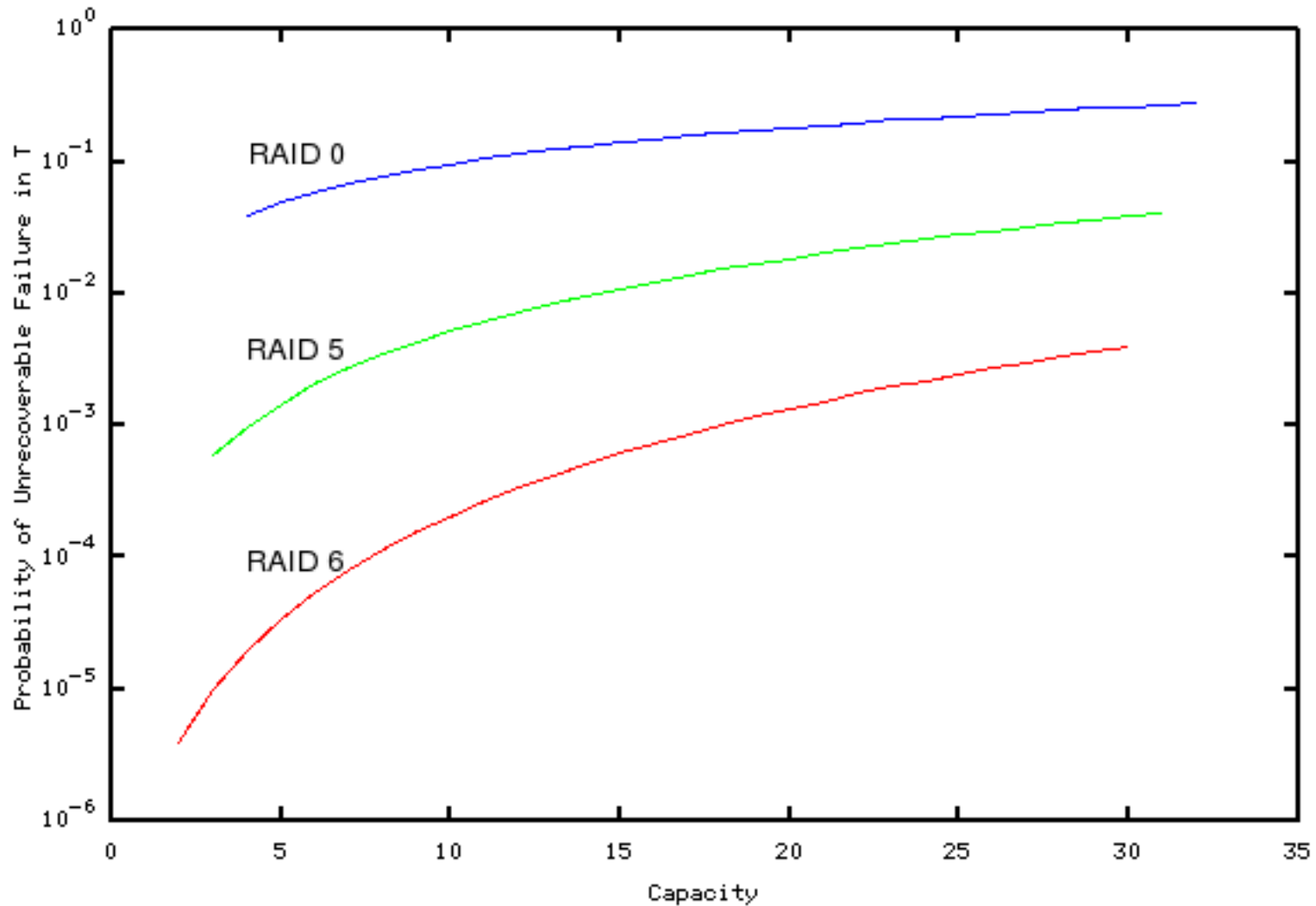
- One parity block per stripe
- Can survive any single-disk failure.

# RAID Levels: RAID 6



- Two parity blocks per stripe
- Allows any two disks to fail.
- Note: for sizes larger than 4 devices, more complicated operations than XOR are needed.

# Raid Levels: Reliability



Probability of unrecoverable failure in a period T for which a single device has a 1:100 chance of failure.

# RAID Levels: Summary

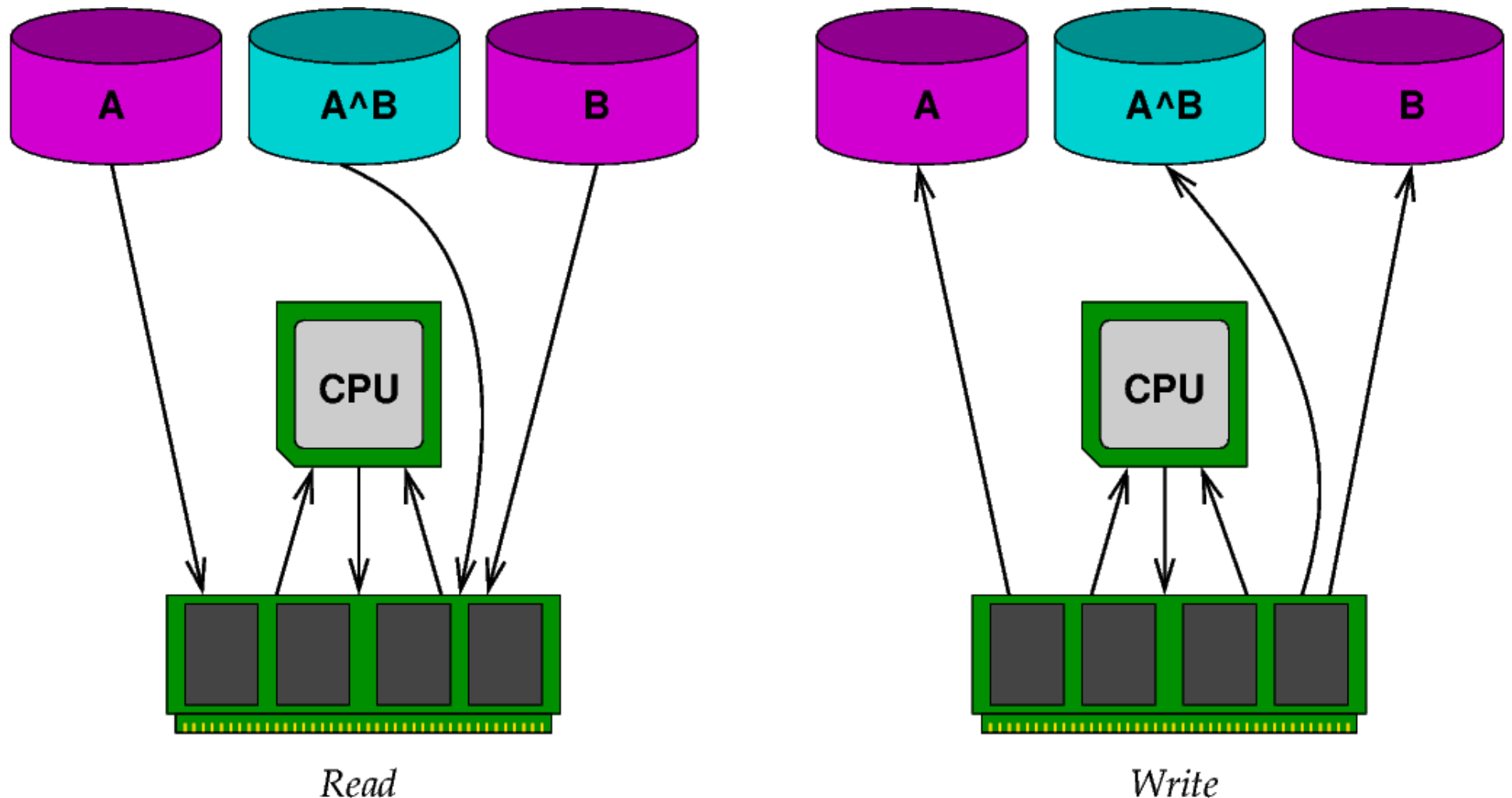
RAID Level	Min. Devices	Capacity	Max. Write BW	Max. Read BW	Probability of Unrecoverable Failure in Period for which 1 Disk Has Failure Probability $p$	Cost per Unit Capacity
RAID 0	2	$n$	$n$	$n$	$1 - (1-p)^n$	1
RAID 1	2	1	1	$n$	$p^n$	$n$
RAID 5	3	$n - 1$	$n - 1$	$n - 1$	$1 - (1-p)^n - np(1-p)^{n-1}$	$1 + 1/n$
RAID 6	4	$n - 2$	$n - 2$	$n - 2$	$1 - (1-p)^n - np(1-p)^{n-1} - (n^2-n)p^2(1-p)^{n-2}/2$	$1 + 2/n$

Simplifying assumptions:

- Disks cost much more than disk controllers.
- Disk failures are always independent (See disclaimer #3)
- Cost of power supplies, mounting hardware, etc. scales linearly with number of disks being housed.
- Bandwidth between the controllers and RAM is much greater than disk bandwidth.

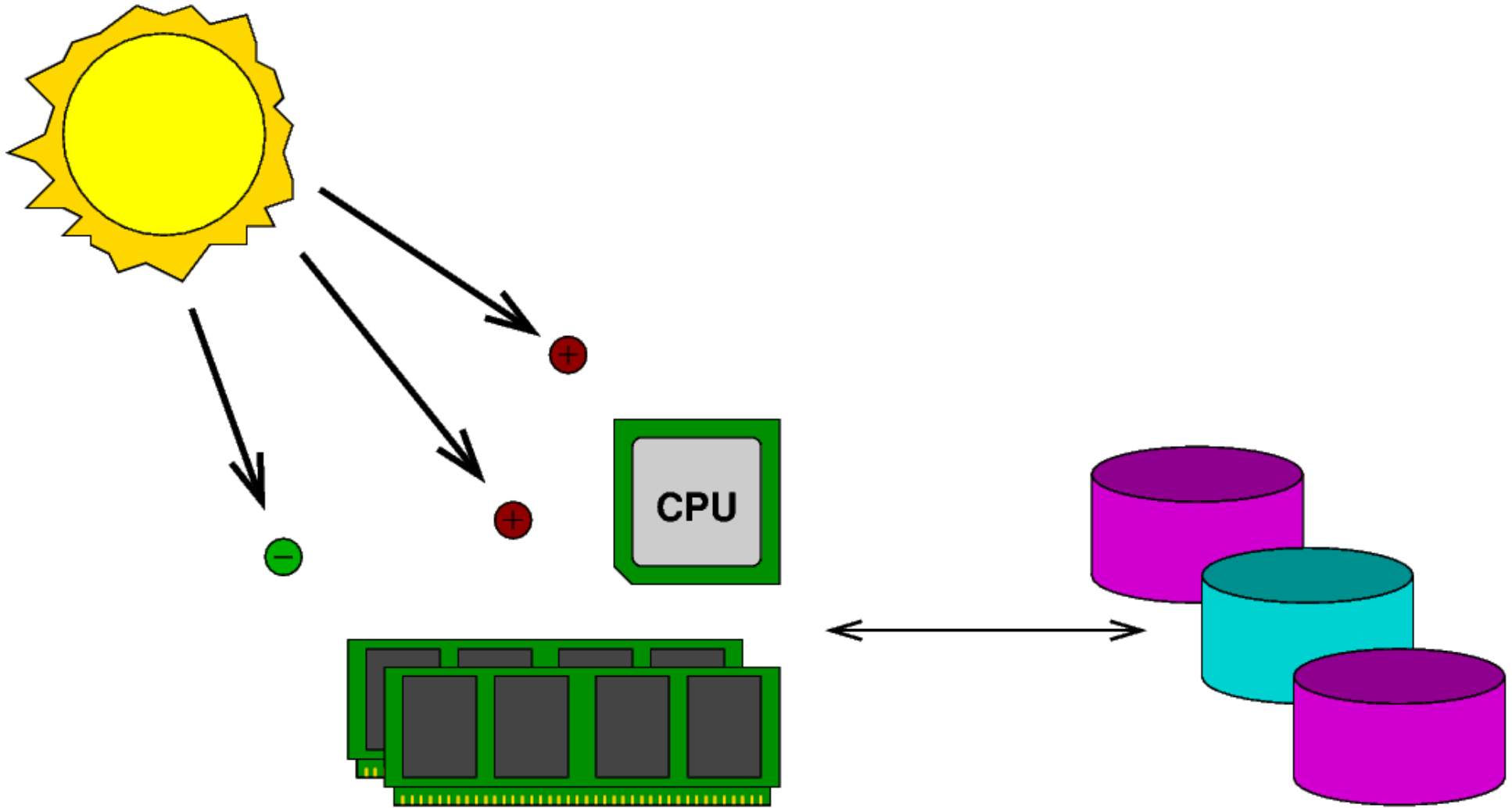
# Software RAID Disclaimer #1

- Parity Computation May be Non-Negligible
- Heavy RAID usage will increase OS CPU utilization
- Disk latency will suffer even if throughput does not



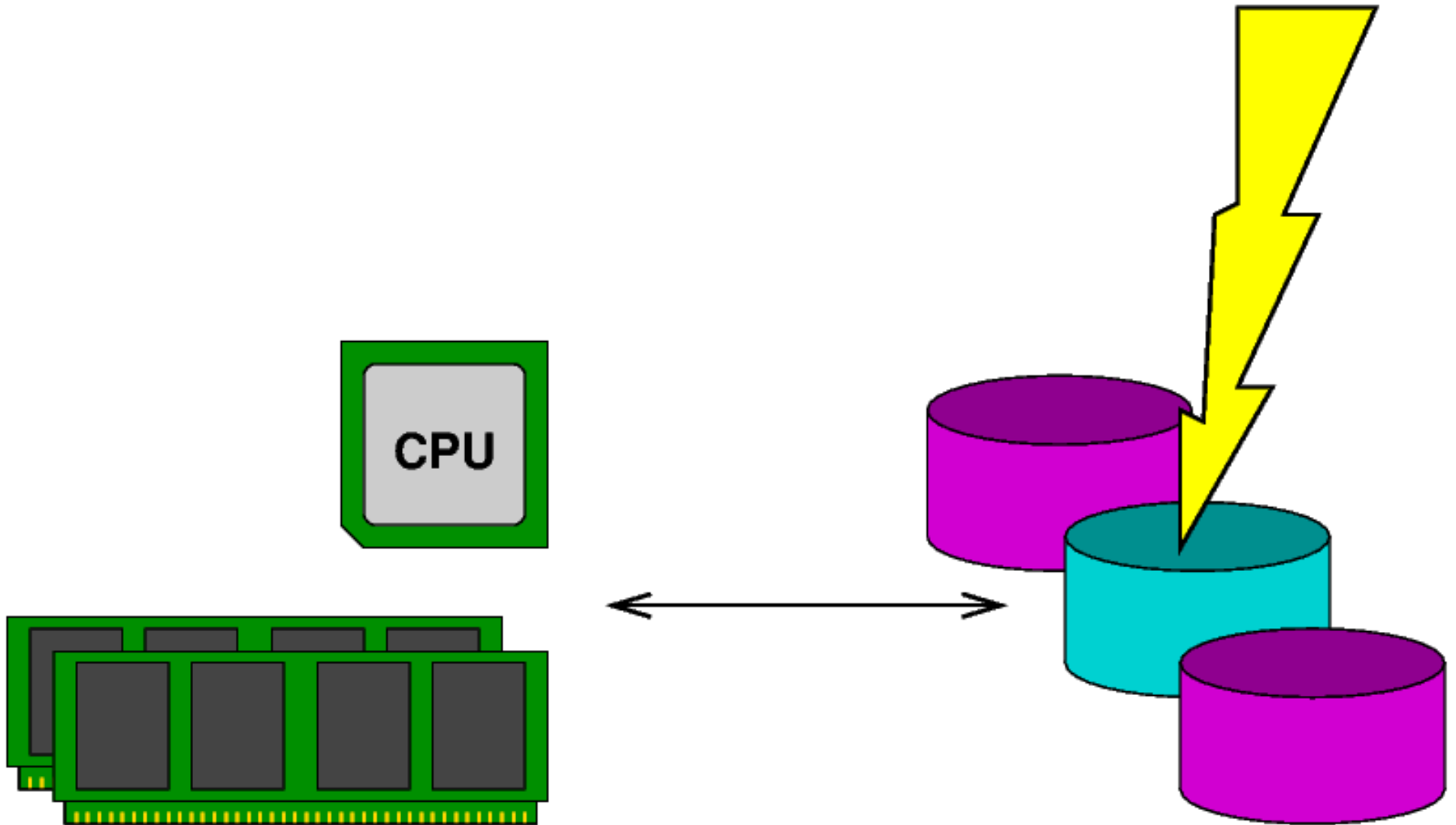
# Software RAID Disclaimer #2

- Blocks are copied additional times to recompute parity info.
- Data corruption more likely, especially with non-ECC RAM.



# Software RAID Disclaimer #3

- While more failure-proof, RAID is not disaster-proof.
- If the disks all overheat, your data is probably toast.



# Software RAID Disclaimer #4

- Disks with the same advertised capacity are not exactly the same size in blocks.
- Even disks with the *same model number* can be different by a few thousands of 512B blocks.
- Remember this warning, and leave some padding when creating partitions for your array.

# Using mdadm

First, make sure you have the mdadm package installed.

To create a RAID:

```
$ mdadm --create md0 --chunk=64 --level=6 \  
    --metadata=1.2 --raid-devices=6 DEV0 DEV1 DEV2...
```

To shut down a running RAID:

```
$ mdadm --stop /dev/md/md0
```

To restart a RAID:

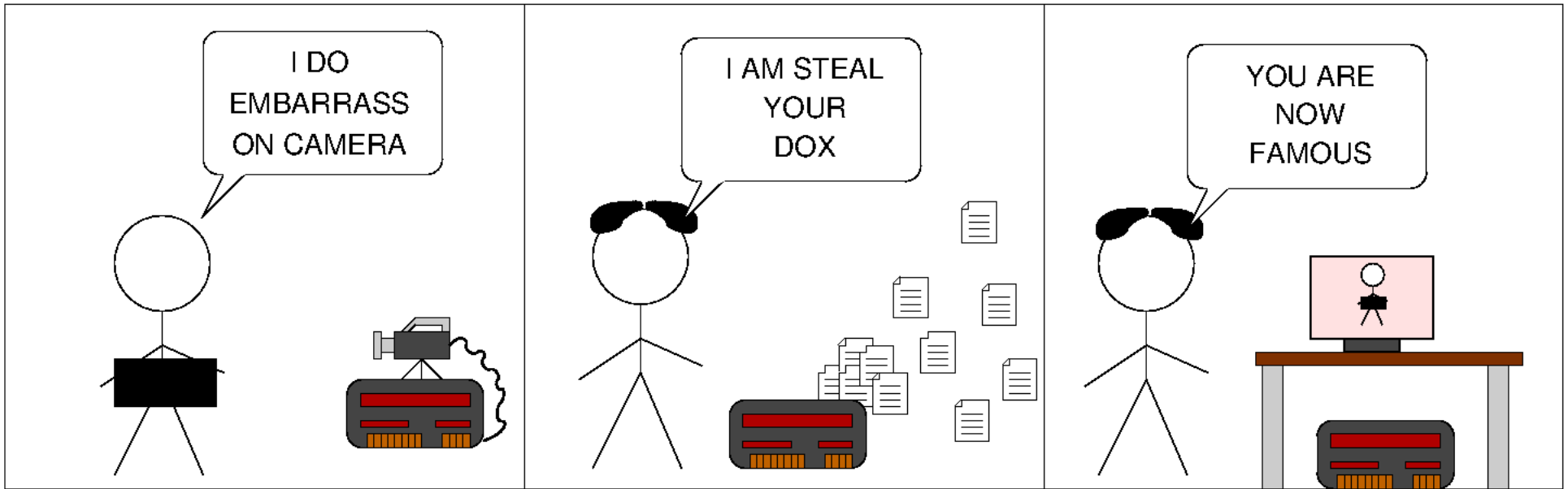
```
$ mdadm --assemble md0 DEV0 DEV1 DEV2...
```

Make a note of the parameters of the RAID! If your superblock is accidentally overwritten, you can recreate it as long as you have the parameters. If you use the defaults, note them too; the defaults are subject to change!

# Why Privacy Matters #1: Identity Theft



# Why Privacy Matters #2: Unintentional Internet Celebrity



- Good luck with your career as a public servant after this happens.

# Why Privacy Matters #3: Legal Issues

- Are you sure your hard drive contains no evidence of anyone breaking any law for any reason? Or anything that could possibly be construed as such?
- Ever seen a TV ad for a law office? Those books are lists of things someone could get in trouble for if the proper authorities were alerted.

# Entropy

- Quantifies "information" or "randomness".
- Can be measured in bits.
- 1 bit of entropy corresponds to 1 fair coin flip.



- Generally, any uniformly distributed random process with  $n$  possible outcomes generates  $\log_2 n$  bits of entropy:

Roll a 6-sided die	2.585 bits
Random digit	3.322 bits
Random letter	4.700 bits
Random printable char:	6.570 bits
Random Diceware word:	12.925 bits

# Choosing Passphrases

- Linux uses 256-bit AES by default; passphrases with up to 256 bits of entropy are worthwhile to use.
- That's:
  - 20 diceware words
  - 40 of any character on the keyboard
  - 64 hex digits
- That's also hard to remember.
- Tip: if you must write it down or use LUKS, memorize at least 50 bits worth of it. (Think of this portion as a reasonably strong password; 17.8 years at 1 attempt/microsecond)
  - 8 printable characters (7G):2mY')
  - 4 diceware words (model-waco-ewe-pump)

# Choosing Passphrases:

## Printable Characters

- An example containing just over 256 bits worth of entropy:

iul/)ISoNj-G3\*jyR^0\d-CU?xB^g>MR-&EDuiO4%J]

- This was produced with the following C code, with input redirected from /dev/random:

```
char c;
while (chars_left && ((c = getc(stdin)) != EOF)) {
    if (isprint(c)) { putc(c, stdout); chars_left--; }
}
```

□

- Depending on your locale, there are at least 95 printable chars (6.57 bits per char).

# Choosing Passphrases: Diceware

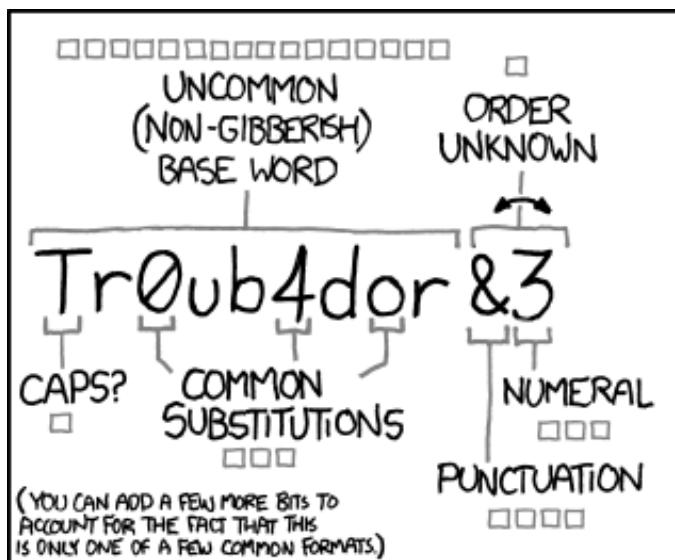
- Easier to remember.
- Require more space when written down.
- Download the list from [www.diceware.com](http://www.diceware.com) and cut off the PGP signature. To generate a passphrase:

```
$ shuf --random-source=/dev/random -n 1 \  
  diceware.wordlist.asc  
once for each word.
```

- An example containing 256 bits of entropy:

```
rubric-fuji-63-how-don-ha-purge-bolo-tried-oo-  
62-71-jewish-dish-shish-belie-gem-argive-gladdy-abel
```

# Randall Gets It



~28 BITS OF ENTROPY

$2^{28} = 3 \text{ DAYS AT } 1000 \text{ GUESSES/SEC}$

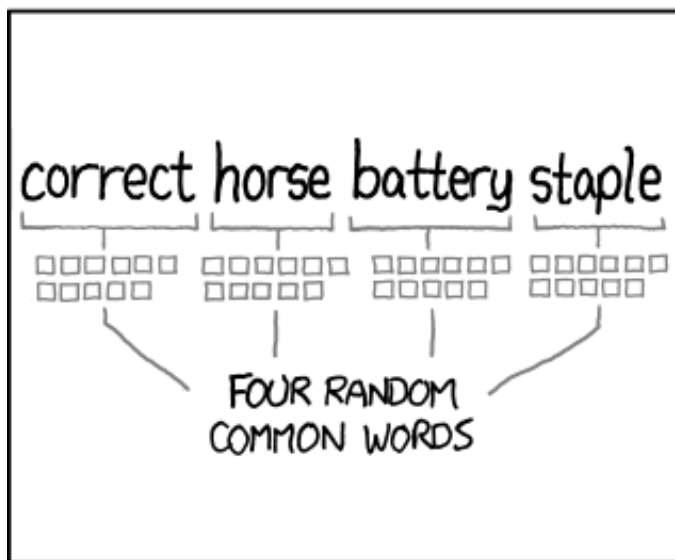
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)

DIFFICULTY TO GUESS:  
**EASY**

WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE 0s WAS A ZERO?

AND THERE WAS SOME SYMBOL...

DIFFICULTY TO REMEMBER:  
**HARD**



~44 BITS OF ENTROPY

$2^{44} = 550 \text{ YEARS AT } 1000 \text{ GUESSES/SEC}$

DIFFICULTY TO GUESS:  
**HARD**

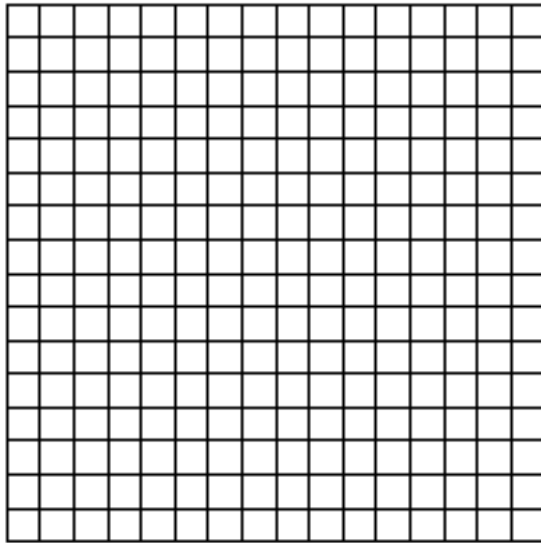
THAT'S A BATTERY STAPLE.

CORRECT!

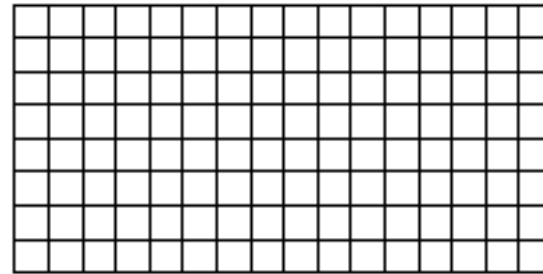
DIFFICULTY TO REMEMBER:  
YOU'VE ALREADY MEMORIZED IT

THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

# Why Encrypt Swap?



RAM: 1M (4k Pages)

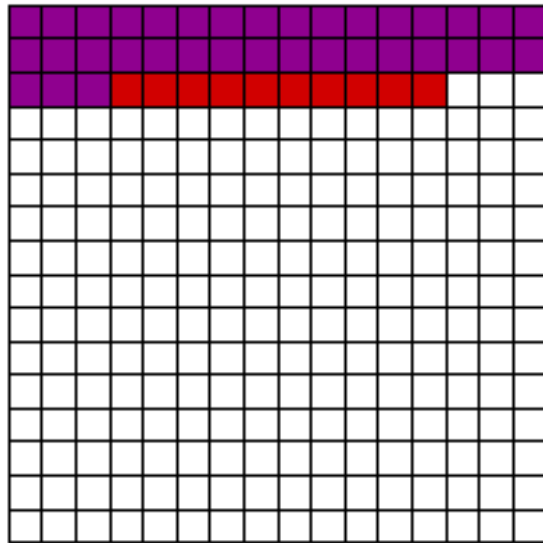


Swap: 512k (4k Pages)

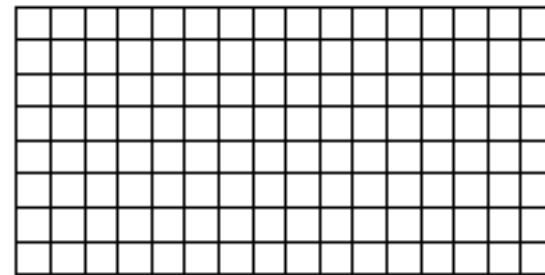
- Unallocated
- OS Data
- Benign User Data
- Sensitive User Data

# Why Encrypt Swap?

Boot the machine and start working with sensitive data.



RAM: 1M (4k Pages)

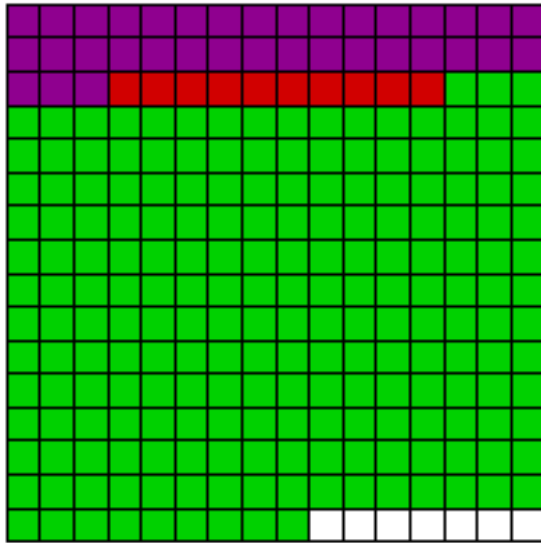


Swap: 512k (4k Pages)

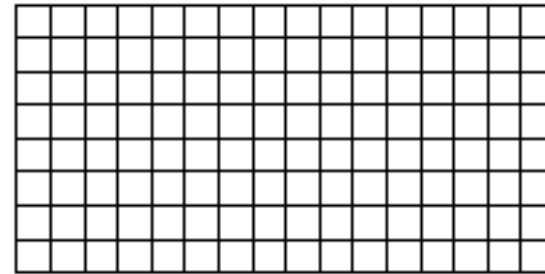
- Unallocated
- OS Data
- Benign User Data
- Sensitive User Data

# Why Encrypt Swap?

Stop using your app; load another app with large mem. footprint.



RAM: 1M (4k Pages)

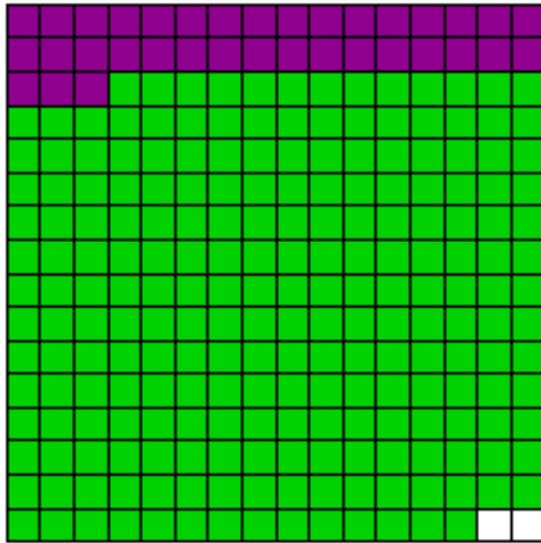


Swap: 512k (4k Pages)

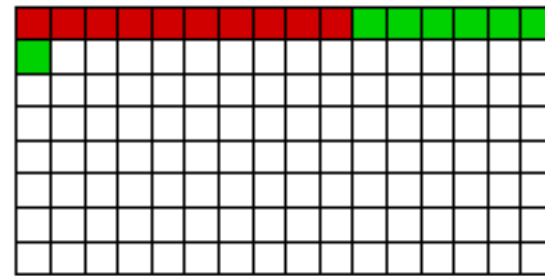
- Unallocated
- OS Data
- Benign User Data
- Sensitive User Data

# Why Encrypt Swap?

Wait.



RAM: 1M (4k Pages)

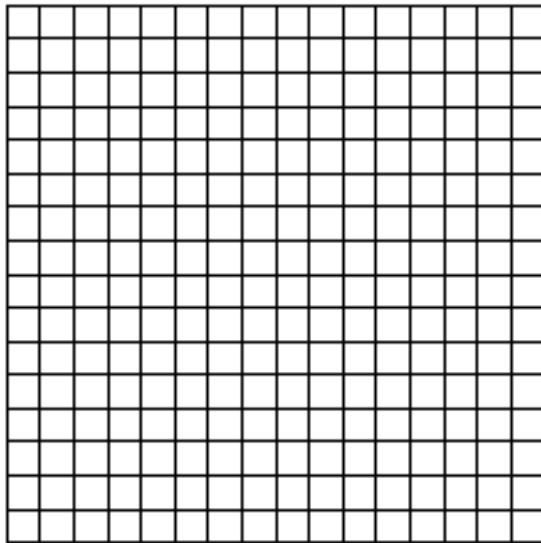


Swap: 512k (4k Pages)

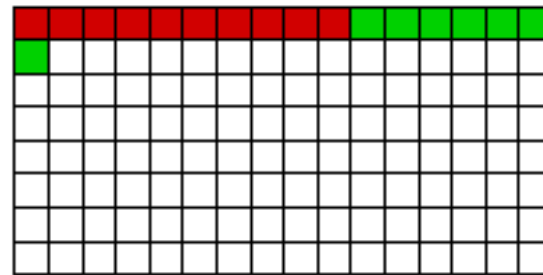
- Unallocated
- OS Data
- Benign User Data
- Sensitive User Data

# Why Encrypt Swap?

Power turned off before swapped-out pages overwritten.  
At any given time, sensitive data could be in swap.



RAM: 1M (4k Pages)



Swap: 512k (4k Pages)

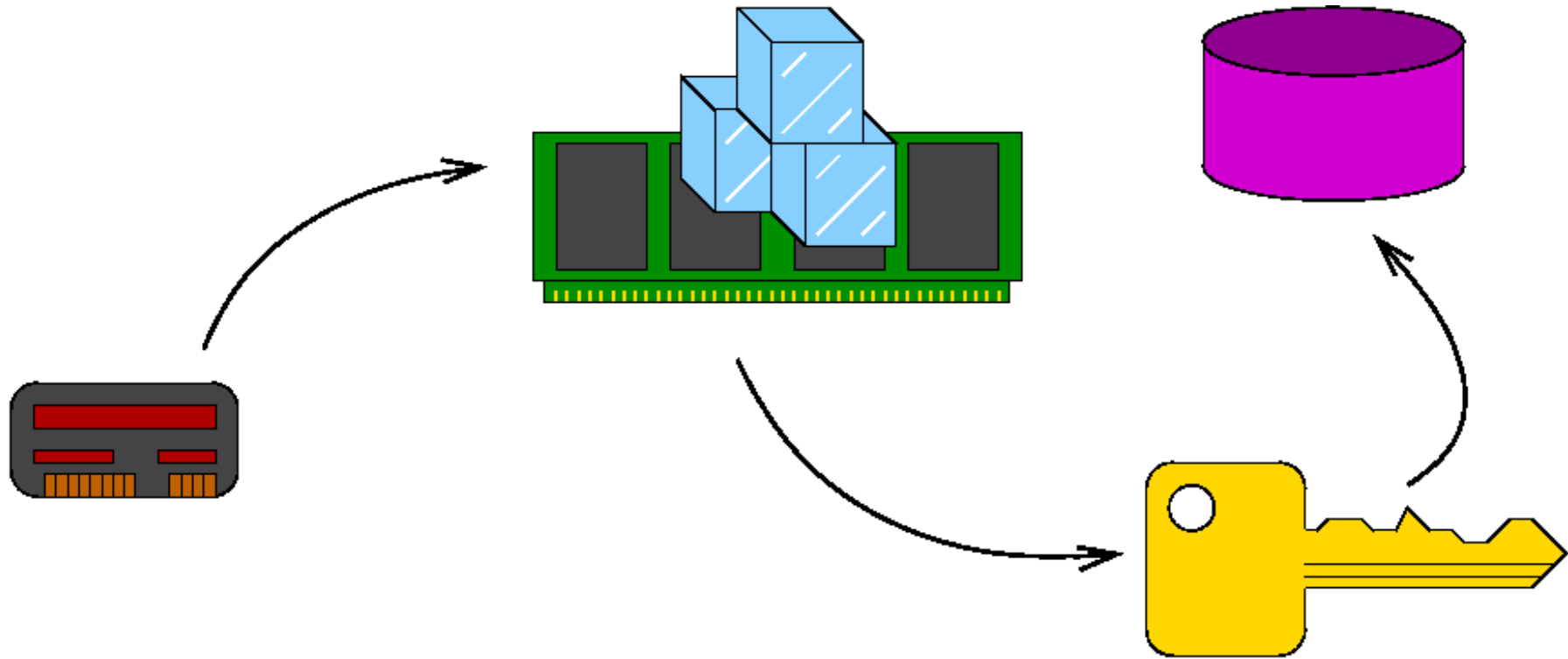
- Unallocated
- OS Data
- Benign User Data
- Sensitive User Data

# Encrypted Storage in the Cloud

- A benefit of loopback files is that they can be synced to a remote server or service like Dropbox.
- With encryption, these files can be synced to an untrusted service.
- With md RAID an encrypted volume can be striped across several remote services.

# Encryption Disclaimer

Even if you encrypt your swap, no consumer hardware encrypts the RAM contents. These fade slowly, so there is a good chance a determined thief could dump RAM\*, get keys, and access your encrypted disk.



\*See J. Alex Halderman et al. Lest we remember: cold-boot attacks on encryption keys. Communications of the ACM. v.52 n.5. May 2009

# Using dm-crypt

To create an encrypted device:

```
$ sudo cryptsetup create crypt0 MY_DEVICE
```

To disable it:

```
$ sudo cryptsetup remove crypt0
```

Cipher can be specified with `--cipher`. Just like RAID layout, make a note of the cipher, even if it's the default.

# Using LUKS

LUKS adds to the spartan dm-crypt a header and simple key management (multiple passphrases). This:

- Simplifies the process of bringing up the encrypted device at boot
- Provides feedback when a passphrase entry attempt was unsuccessful
- Stores which cipher was given to luksFormat in a header so this does not have to be remembered.
- Makes it dead obvious that your device contains an encrypted volume.

Relevant commands include:

```
$ cryptsetup luksFormat MY_DEVICE  
$ cryptsetup luksOpen MY_DEVICE crypt0  
$ cryptsetup luksClose crypt0  
$ cryptsetup luksAddKey crypt0  
$ cryptsetup luksRemoveKey crypt0
```

# Setting up Swap Space

Swap can be held in a file or a block device (including loopback devices). Putting it in a file hurts performance somewhat, but this is a snowflake in the avalanche when you're swapping.

Format swap devices with `mkswap`:

```
$ /sbin/mkswap SWAPFILEorDEVICE
```

Enable swap on a device:

```
$ sudo swapon SWAPFILEorDEVICE
```

View current swap devices/files:

```
$ cat /proc/swaps
```

Disable swap on a device:

```
$ sudo swapoff SWAPFILEorDEVICE
```

# Swap to File

The easy, safe way to swap to a file (contiguous):

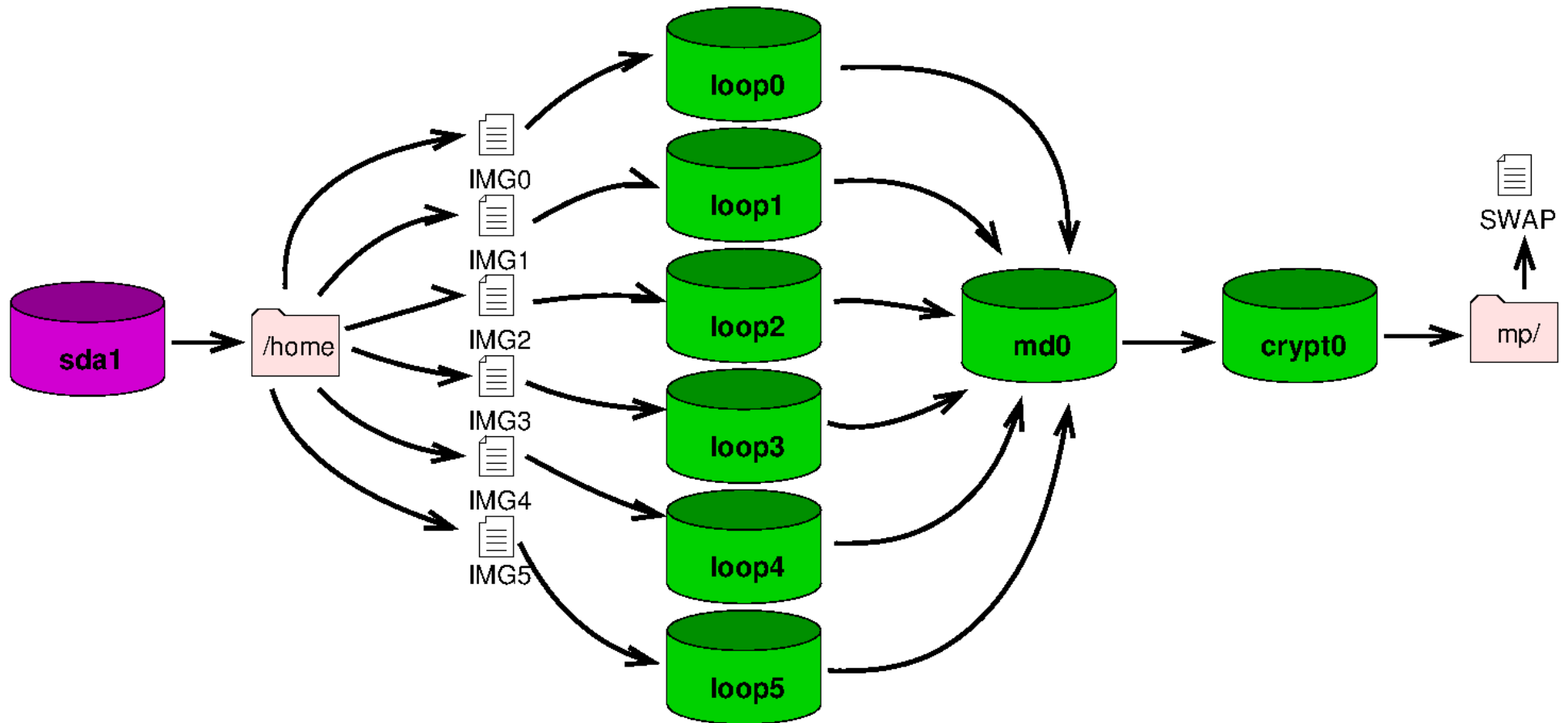
```
$ mkswap MYFILE$ sudo swapon MYFILE
```

To cheat and swap to a sparse file:

```
$ sudo losetup -a # Find unused loopback dev$ sudo losetup /dev/loop0  
MYFILE$ sudo mkswap /dev/loop0$ sudo swapon /dev/loop0
```

**WARNING:** This creates a vulnerability. Any user with write permissions to the filesystem containing MYFILE could create a condition in which the next attempt to swap will cause a kernel panic!

# Demo



*The Demo: six 20MB image files in a RAID6, encrypted, containing a 50MB swap file.*