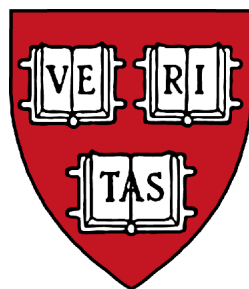


CS161: Operating Systems

Matt Welsh
mdw@eecs.harvard.edu



Lecture 18: RAID
April 19, 2007

RAID

Redundant Arrays of Inexpensive Disks

- Invented in 1986-1987 by David Patterson and Randy Katz, UC Berkeley
- Now a multibillion dollar a year industry



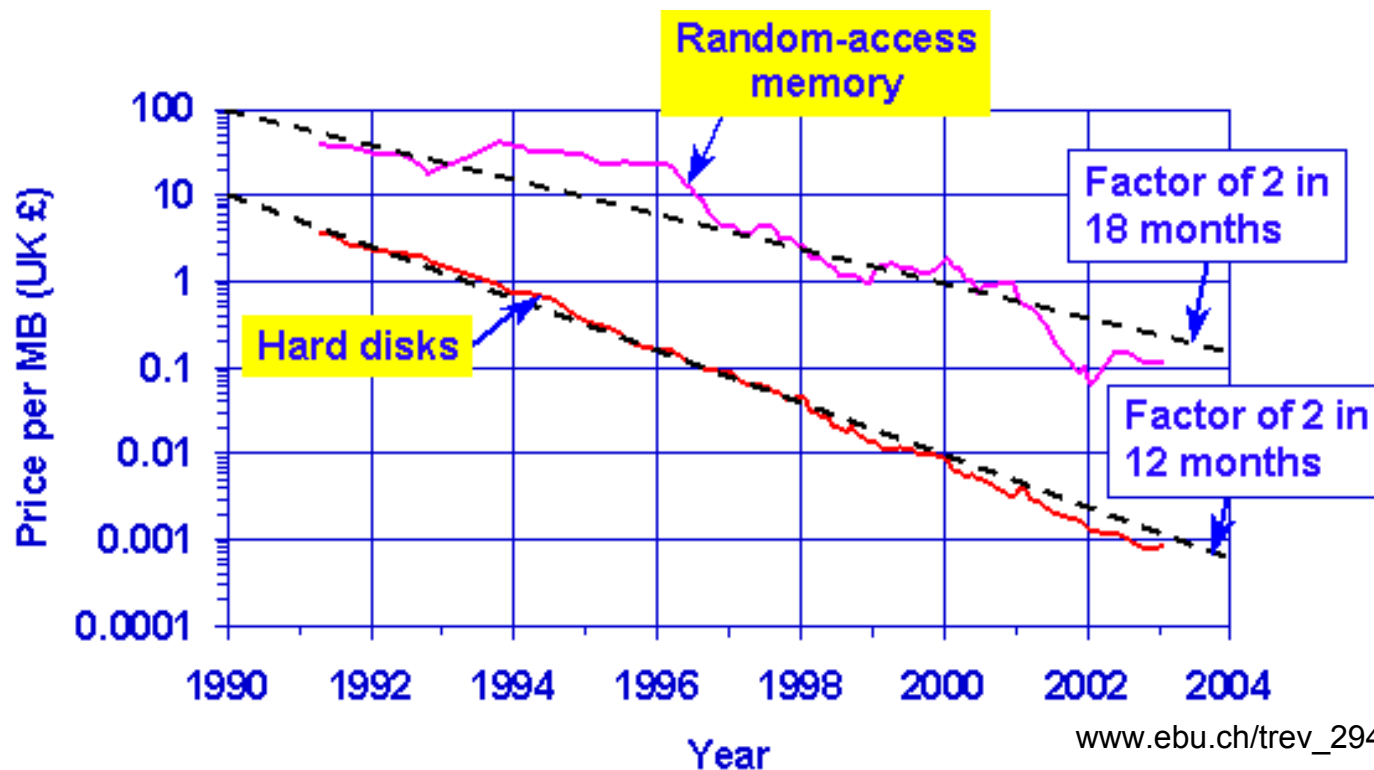
RAID Motivation

Speed of disks not matching that of other components

- Moore's Law: CPU speed doubles every 18 months
- SRAM speeds increasing by 40-100% a year
- In contrast, disk seek time only improving 7% a year
 - *Although greater density leads to improved transfer times once seek is done*

Emergence of PCs starting to drive down costs of disks

- *(This is 1988 after all)*
- PC-class disks were smaller, cheaper, and only marginally slower



www.ebu.ch/trev_294-editorial.html

RAID Motivation

Basic idea: Build I/O systems as arrays of cheap disks

- Allow data to be **striped** across multiple disks
- Means you can read/write multiple disks in parallel – greatly improve performance

Problem: disks are extremely unreliable

Mean Time to Failure (MTTF)

- $MTTF(\text{disk array}) = MTTF(\text{single disk}) / \# \text{ disks}$
- Adding more disks means that failures happen more frequently..
- **An array of 100 disks with an MTTF of 30,000 hours = just under 2 weeks!**

Increasing Reliability

Idea: Replicate data across multiple disks

- When a disk fails, lost information can be regenerated from the redundant data

Simplest form: Mirroring (also called “RAID 1”)

- All data is mirrored across two disks

Advantages?

-
-

Disadvantages?

-
-

Increasing Reliability

Idea: Replicate data across multiple disks

- When a disk fails, lost information can be regenerated from the redundant data

Simplest form: Mirroring (also called “RAID 1”)

- All data is mirrored across two disks

Advantages:

- Reads are faster, since both disks can be read in parallel
- Higher reliability (of course)

Disadvantages:

- Writes are slightly slower, since OS must wait for both disks to do write
- This approach also doubles the cost of the storage system!

RAID 3

Rather than mirroring, use *parity codes*

- Given N bits $\{b_1, b_2, \dots, b_N\}$, the *parity bit* P is the bit $\{0,1\}$ that yields an even number of “1” bits in the set $\{b_1, b_2, \dots, b_N, P\}$
- Idea: If any bit in $\{b_1, b_2, \dots, b_N\}$ is lost, can use the remaining bits (plus P) to recover it.

Where to store the parity codes?

- Add an extra “check disk” that stores parity bits for the data stored on the rest of the N disks

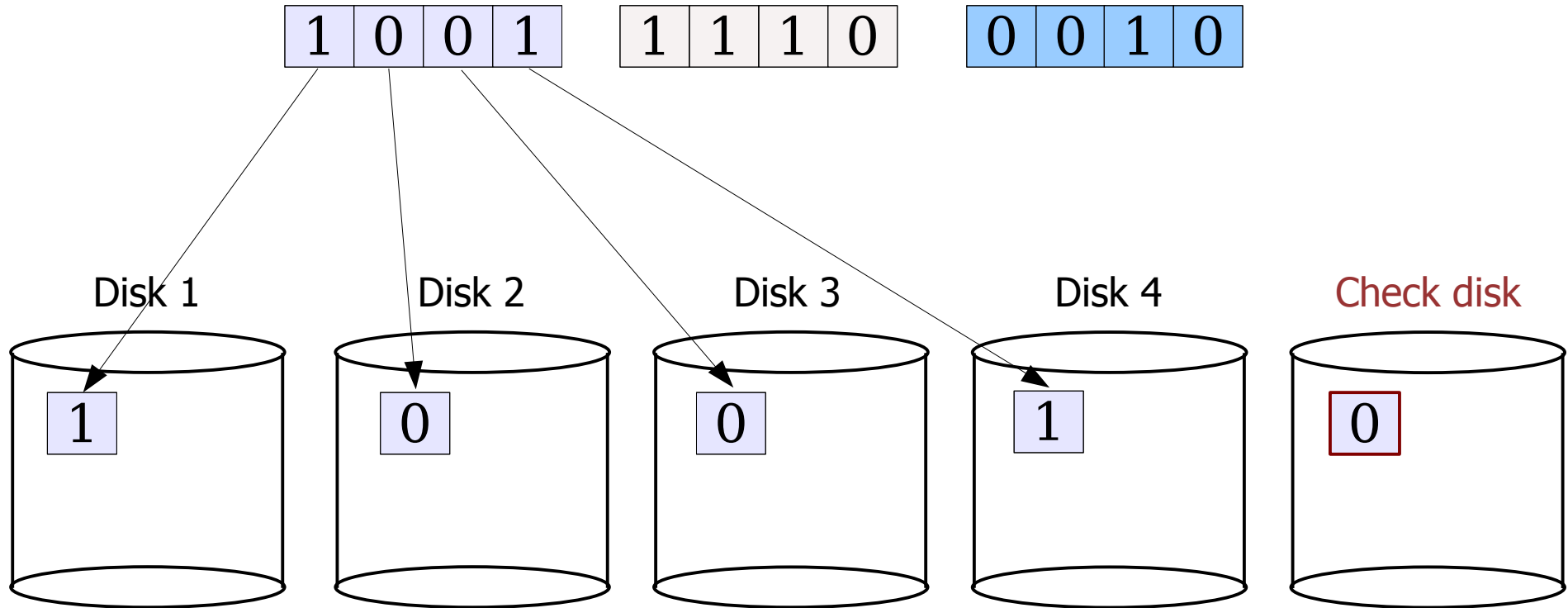
Advantages:

- If a single disk fails, can easily recompute the lost data from the parity code
- Can use one parity disk for *several* data disks (reduces cost)

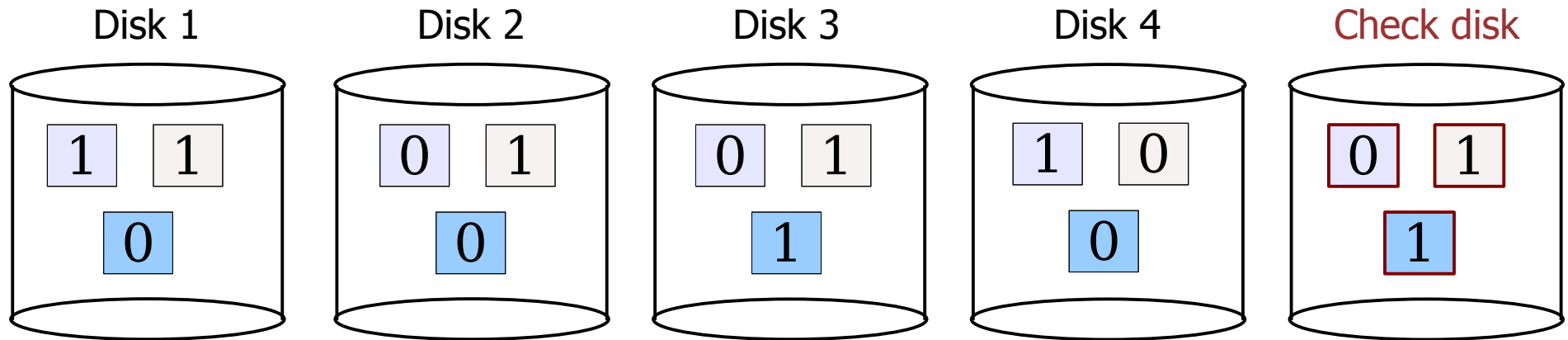
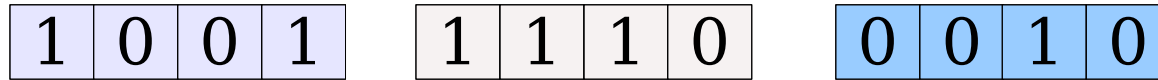
Disadvantages:

- Each write to a block must update the corresponding parity block as well

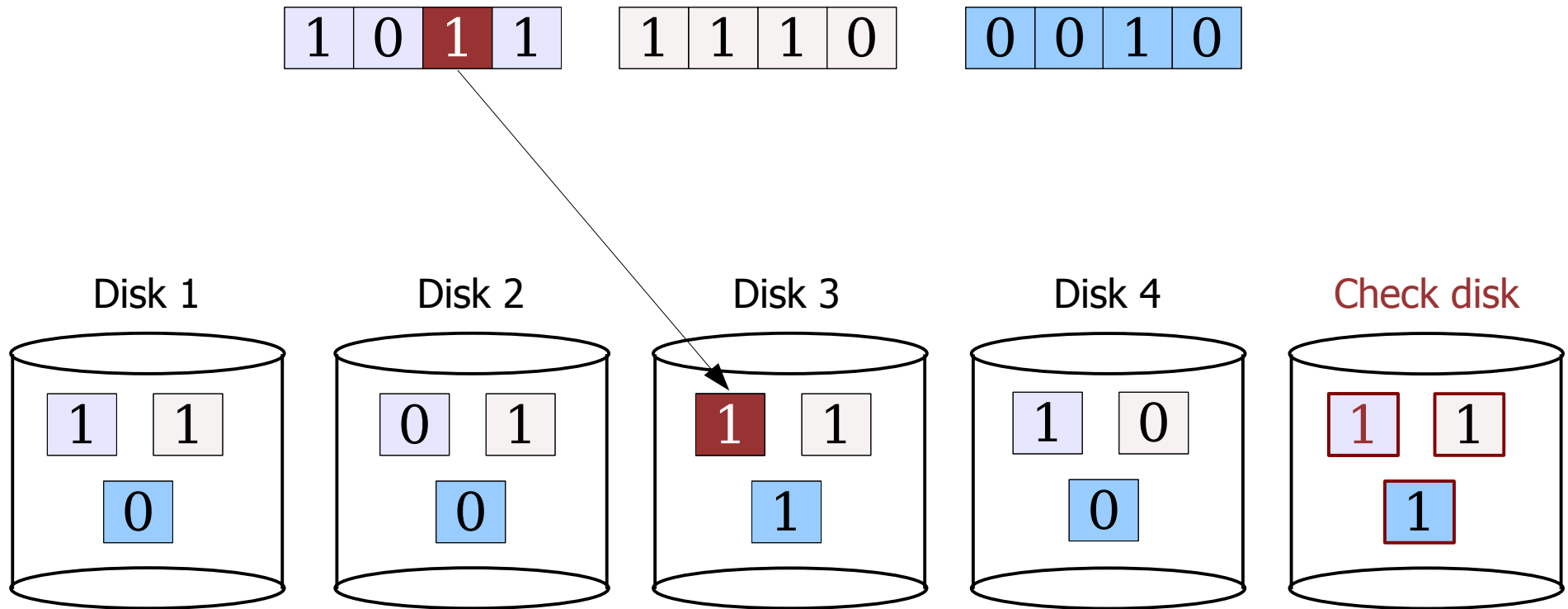
RAID 3 Example



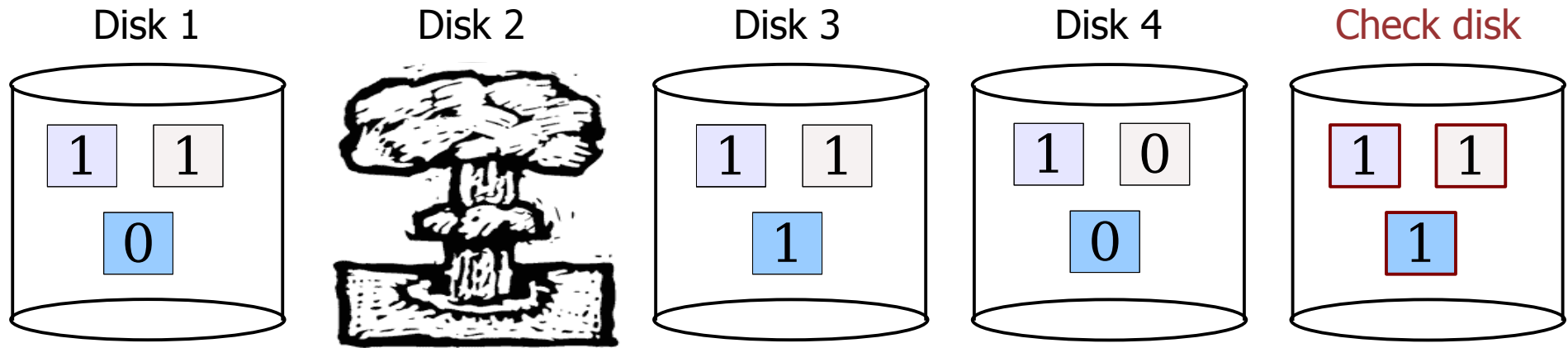
RAID 3 Example



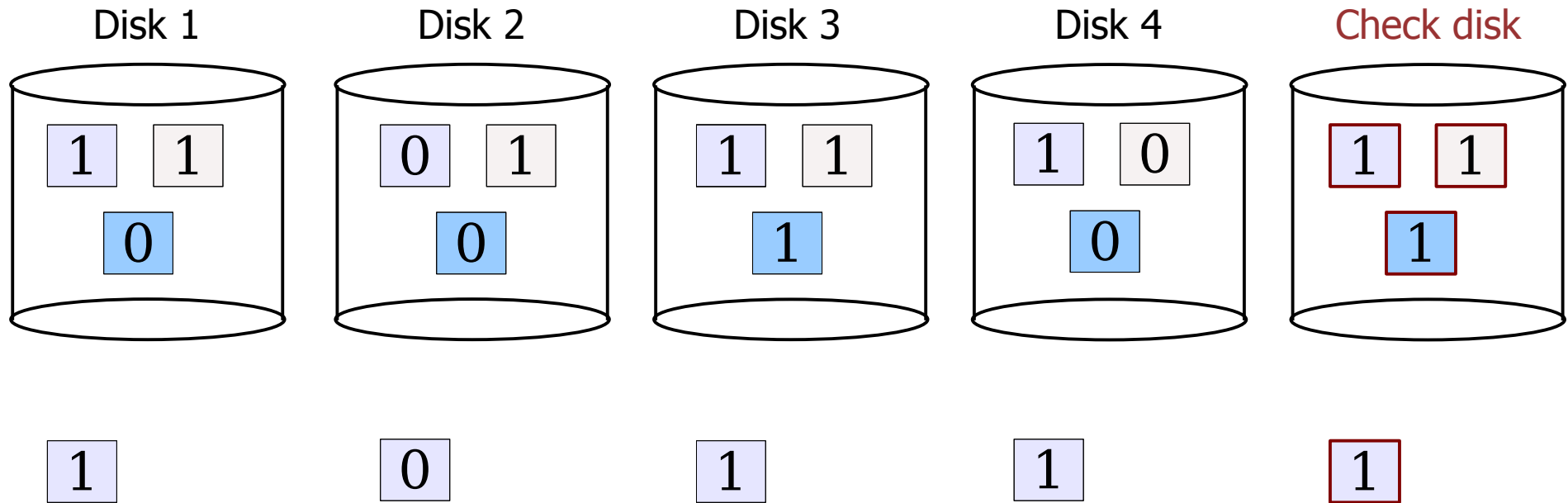
RAID 3 Example



RAID 3 Example



RAID 3 Example



1. Read back data from other disks
2. Recalculate lost data from parity code
3. Rebuild data on lost disk

RAID 3 Issues

What is the MTTF of RAID?

- Both RAID 1 and RAID 3 tolerate the failure of a single disk
- As long as a second disk does not die while we are repairing the first failure, we are in good shape!
- So, what is the probability of a second disk failure?
- $P(2^{\text{nd}} \text{ failure}) = \text{MTTR} / (\text{MTTF of one disk} / \# \text{ disks} - 1)$
 - *This can be derived from independent and exponential failure rates*
 - *See Patterson RAID paper for details*
- 10 disks, MTTF (disk) = 1000 days, MTTR = 1 day
 - $P(2^{\text{nd}} \text{ failure}) = 1 \text{ day} / (1000 / 9) = 0.009$

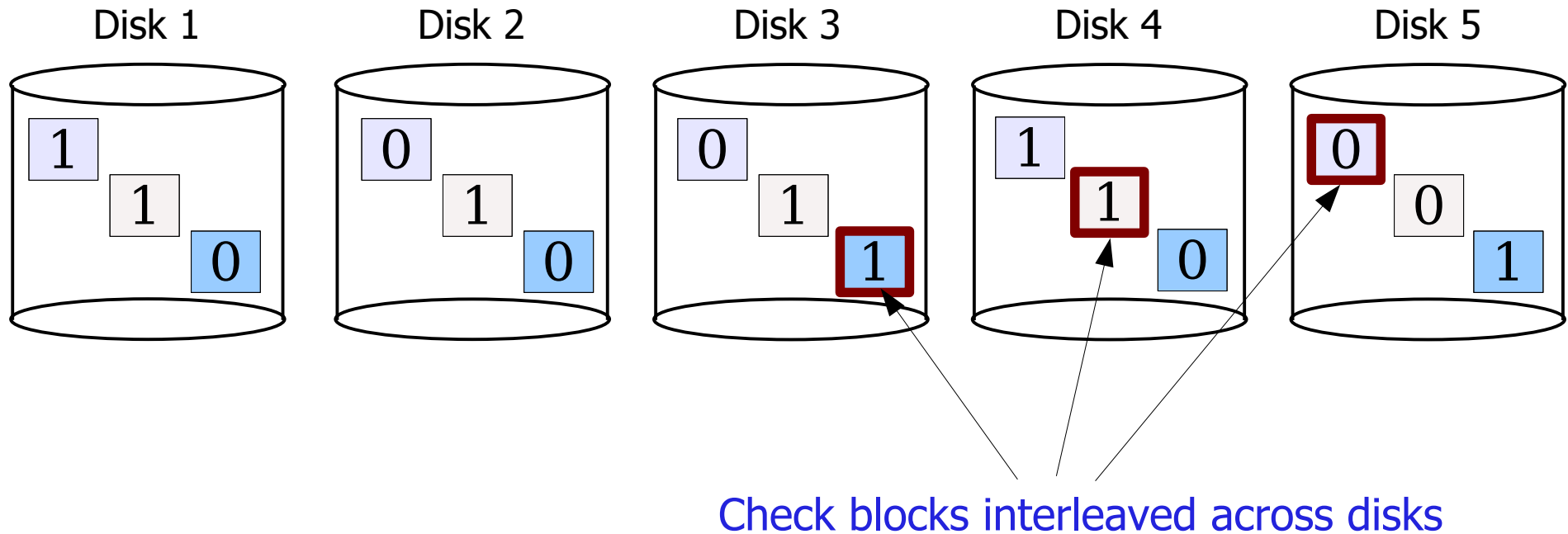
What is the performance of RAID 3?

- Well, the check disk must be updated each time there is a write
- Problem: The check disk is then a performance bottleneck
 - *Only a single read/write can be done at once on the whole system!*

RAID 5

Another approach: Interleaved check blocks (“RAID 5”)

- Rotate the assignment of data blocks and check blocks across disks
- Avoids the bottleneck of a single disk for storing check data
- Allows multiple reads/writes to occur in parallel (since different disks affected)

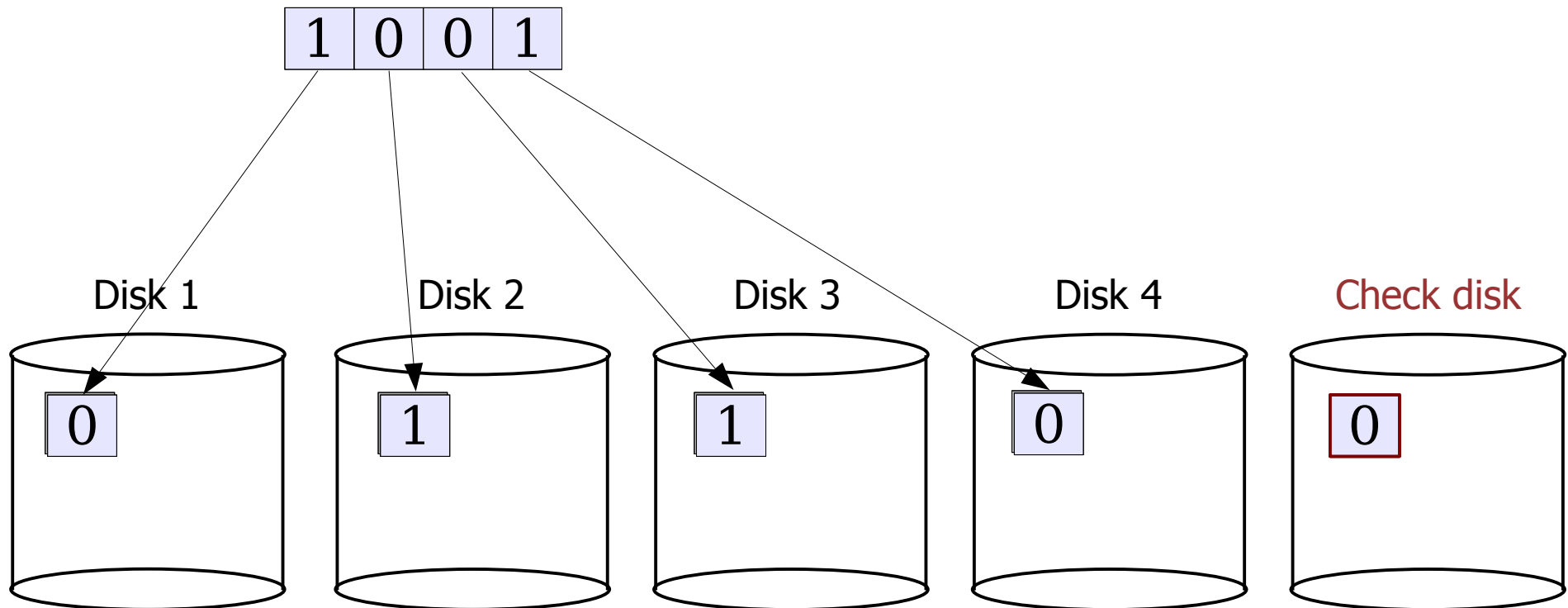


Atomic Write Failure

Many applications perform “update in place”

- They change a file on disk by **overwriting** it with a new version

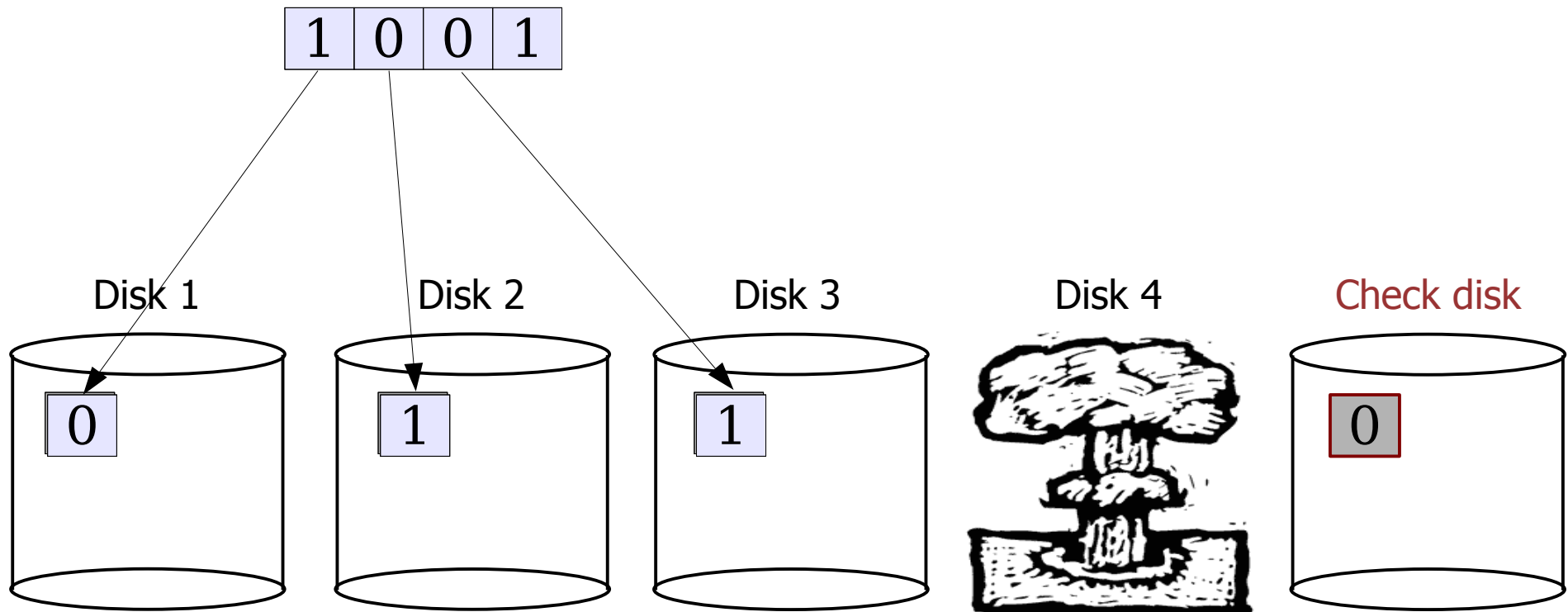
What happens with RAID?



Atomic Write Failure

But is the complete write to all disks really *atomic*?

- Generally, no!



Atomic Write Failure

But is the complete write to all disks really *atomic*?

- Generally, no!

What does this mean?

- Data can be left in an inconsistent state across the different disks!
- Really hard to recover from this.

Problem: Most applications assume the storage system has atomic write semantics.

Possible fixes?

- Use a journaling filesystem-like approach: Record changes to data objects transactionally.
 - *Requires extensive changes to filesystem sitting on top of the RAID.*
- Battery-backed write cache:
 - *RAID controller remembers all writes in a battery-backed cache*
 - *When recovery occurs, flush all writes out to the physical disks*
 - *Doesn't solve the problem in general but gives you some insurance.*