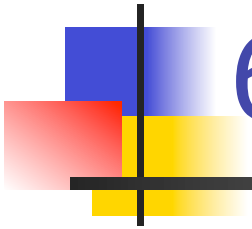


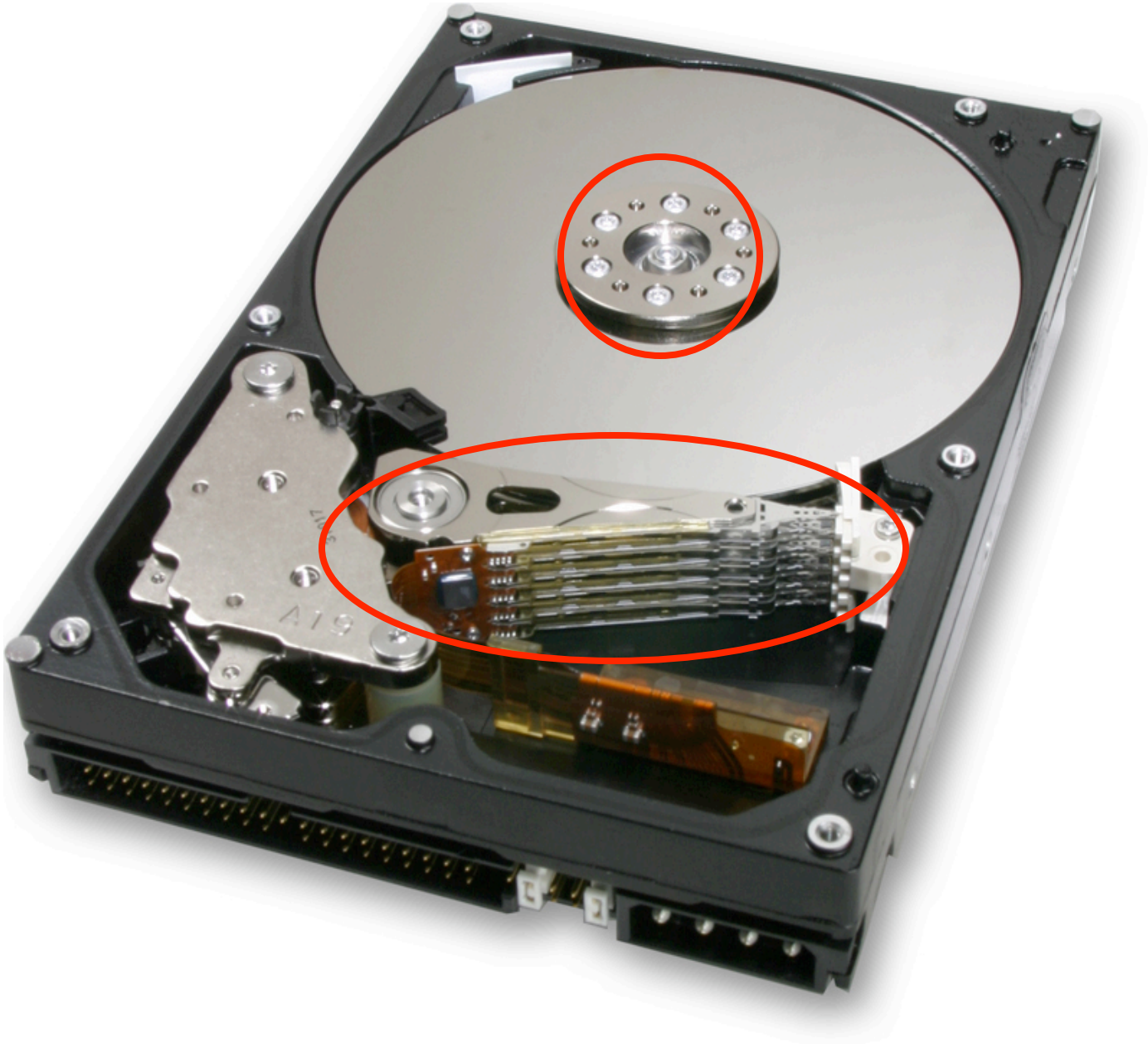
6.033 Spring 2009

Robert Morris

Lecture 8

Performance

Hitachi 7K400



16. OutOfMoney.com

OutOfMoney.com has decided it needs a real product so it is laying off most of its marketing department. To replace the marketing folks, and on the advice of a senior computer expert, OutOfMoney.com hires a crew of 16-year olds. The 16-year-olds get together and decide to design and implement a video service that serves MPEG-1 video, so that they can watch Britney Spears on their computers in living color.

```
procedure SERVICE ()
    do forever
        request ← RECEIVE_MESSAGE ()
        file ← GET_FILE_FROM_DISK (request)
        REPLY (file)
```

The disk has an average seek time of 5 milliseconds, a complete rotation takes 6 milliseconds, and its throughput is 10 megabytes per second when no seeks are required.

All files are 1 gigabyte (roughly a half hour of MPEG-1 video). The file system in which the files are stored has no cache and it allocates data for a file in 8 kilobyte chunks. It pays no attention to file layout when allocating a chunk; as a result disk blocks of the same file can be all over the disk. A 1 gigabyte file contains 131,072 eight kilobyte blocks.

Q 16.1. Assuming that the disk is the main bottleneck, how long does the service take to serve a file?

```
cache [1073741824]                          // 1 gigabyte cache

procedure SERVICE ()
        do forever
                request ← RECEIVE_MESSAGE ()
                file ← LOOK_IN_CACHE (request)
                if file = NULL then
                        file ← GET_FILE_FROM_DISK (request)
                        ADD_TO_CACHE (request, file)
                REPLY (file)
```

The procedure LOOK_IN_CACHE checks whether the file specified in the request is present in the cache and returns it if present. The procedure ADD_TO_CACHE copies a file to the cache.

Q 16.2. Mark tests the code by asking once for every video stored. Assuming that the disk is the main bottleneck (serving a file from the cache takes 0 milliseconds), what now is the average time for the service to serve a file?

Seconds after they launch the Web site, OutOfMoney's support organization (also staffed by 16-year-olds) receives e-mail from unhappy users saying that the service is not responding to their requests. The support department measures the load on the service CPU and also the service disk. They observe that the CPU load is low and the disk load is high.

Q 16.3. What is the most likely reason for this observation?

A. The cache is too large

B. The hit ratio for the cache is low

C. The hit ratio for the cache is high

D. The CPU is not fast enough

The support department beeps Mark, who runs to his brother Ben for help. Ben suggests using the example thread package of chapter 5. Mark augments the code to use the thread package and after the system boots, it start 100 threads, each running SERVICE:

for *i* **from** 1 **to** 100 **do** CREATE_THREAD (SERVICE)

In addition, he modifies RECEIVE_MESSAGE and GET_FILE_FROM_DISK to release the processor by calling YIELD when waiting for a new message to arrive or waiting for the disk to complete a disk read. In no other place does Mark's code release the processor. The implementation of the thread package is non-preemptive.

```
cache [4 x 1073741824]                    // The 4 gigabyte cache, shared by all threads.

procedure SERVICE ()
        do forever
                request ← RECEIVE_MESSAGE ()
                file ← NULL
                for k from 1 to 131072 do
                        block ← LOOK_IN_CACHE (request, k)
                        if block = NULL then
                                block ← GET_BLOCK_FROM_DISK (request, k)
                                ADD_TO_CACHE (request, block, k)
                        file ← file + block              // + concatenates strings
                REPLY (file)
```

Mark loads up the service with one video. He retrieves the video successfully. Happy with this result, Mark sends many requests for the single video in parallel to the service. He observes no disk activity.

Q 16.4. Based on the information so far, what is the most likely explanation why Mark observes no disk activity?

Happy with the progress, Mark makes the service ready for running in production mode. He is worried that he may have to modify the code to deal with concurrency—his past experience has suggested to him that he needs an education so he is reading chapter 5. He considers protecting ADD_TO_CACHE with a lock:

```
lock instance cachelock                          // A lock for the cache

procedure SERVICE ()
        do forever
                request ← RECEIVE_MESSAGE ()
                file ← NULL
                for k from 1 to 131072 do
                        block ← LOOK_IN_CACHE (request, k)
                        if block = NULL then
                                block ← GET_BLOCK_FROM_DISK (request, k)
                                ACQUIRE (cachelock)                    // use the lock
                                ADD_TO_CACHE (request, block, k)
                                RELEASE (cachelock)                    // here, too
                        file ← file + block
                REPLY (file)
```

Q 16.5. Ben argues that these modifications are not useful. Is Ben right?

Q 16.6. Mark observes a hit-ratio of 90% for blocks in the cache. Assuming that the disk is the main bottleneck (serving blocks from the cache takes 0 milliseconds), what is the average time for service to serve a single movie?

Q 16.7. Mark loads a new Britney Spears video onto the service, and observes operation as the first users start to view it. It is so popular that no users are viewing any other video. Mark sees that the first batch of viewers all start watching the video at about the same time. He observes that the service threads all read block 0 at about the same time, then all read block 1 at about the same time, etc. For this workload what is a good cache replacement policy?

A. Least-recently used

B. Most-recently used

C. First-in, first-out

D. Last-in, first-out

E. The replacement policy doesn't matter for this workload