INF5071 – Performance in Distributed Systems

Server Resources: Memory & Disks

October 1, 2010

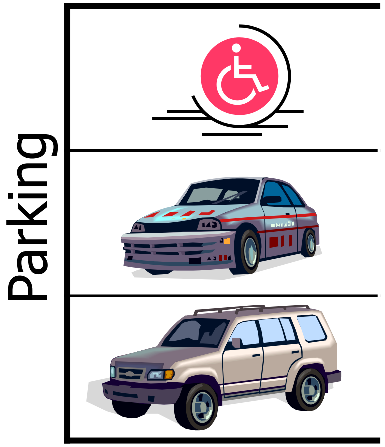
Overview

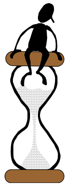
- Memory management
  - caching
  - copy free data paths
- Storage management
  - disks
  - scheduling
  - placement
  - file systems
  - multi-disk systems
  - …

Memory Management

Why look at a passive resource?

"Dying philosophers problem"

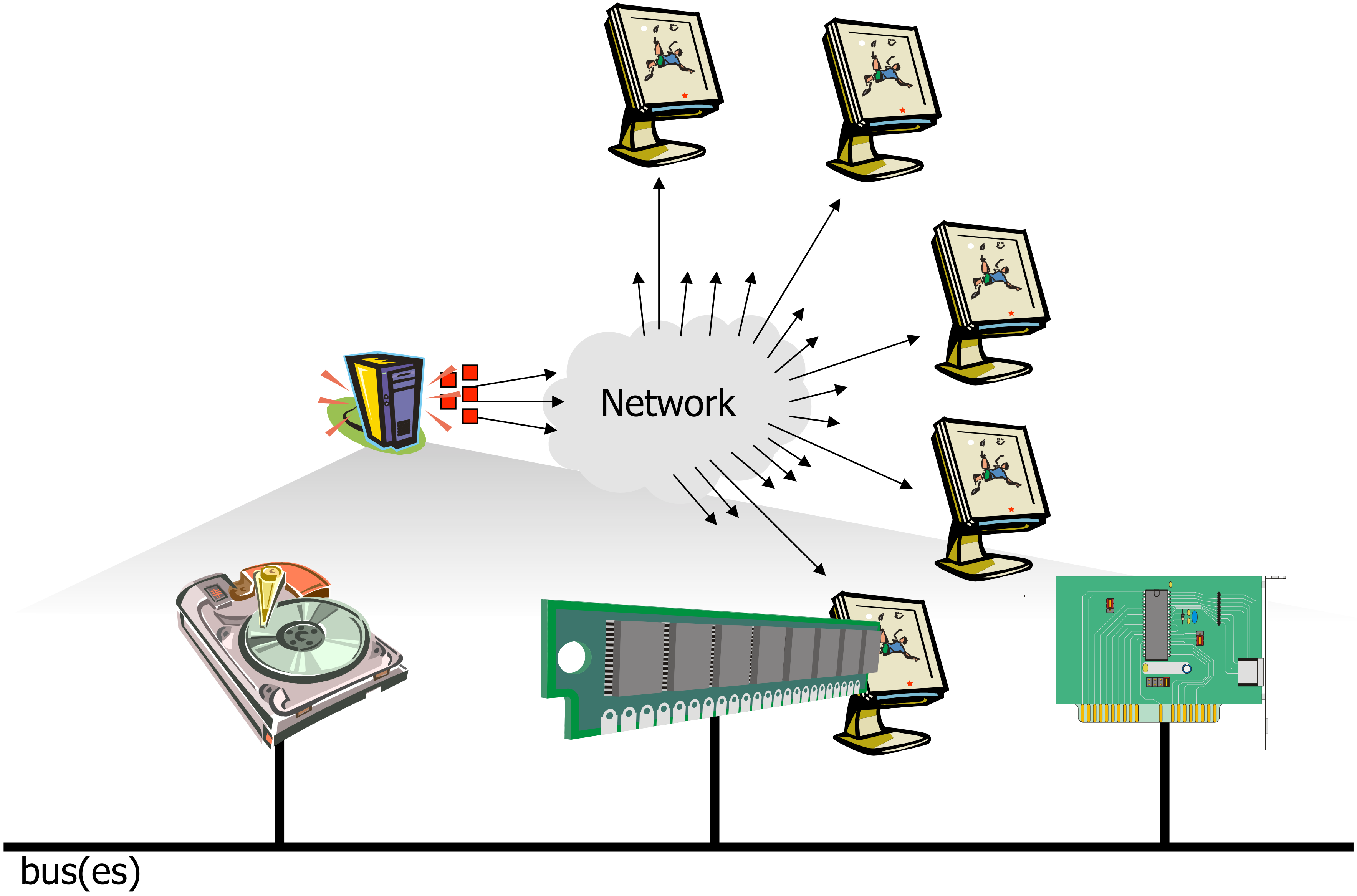




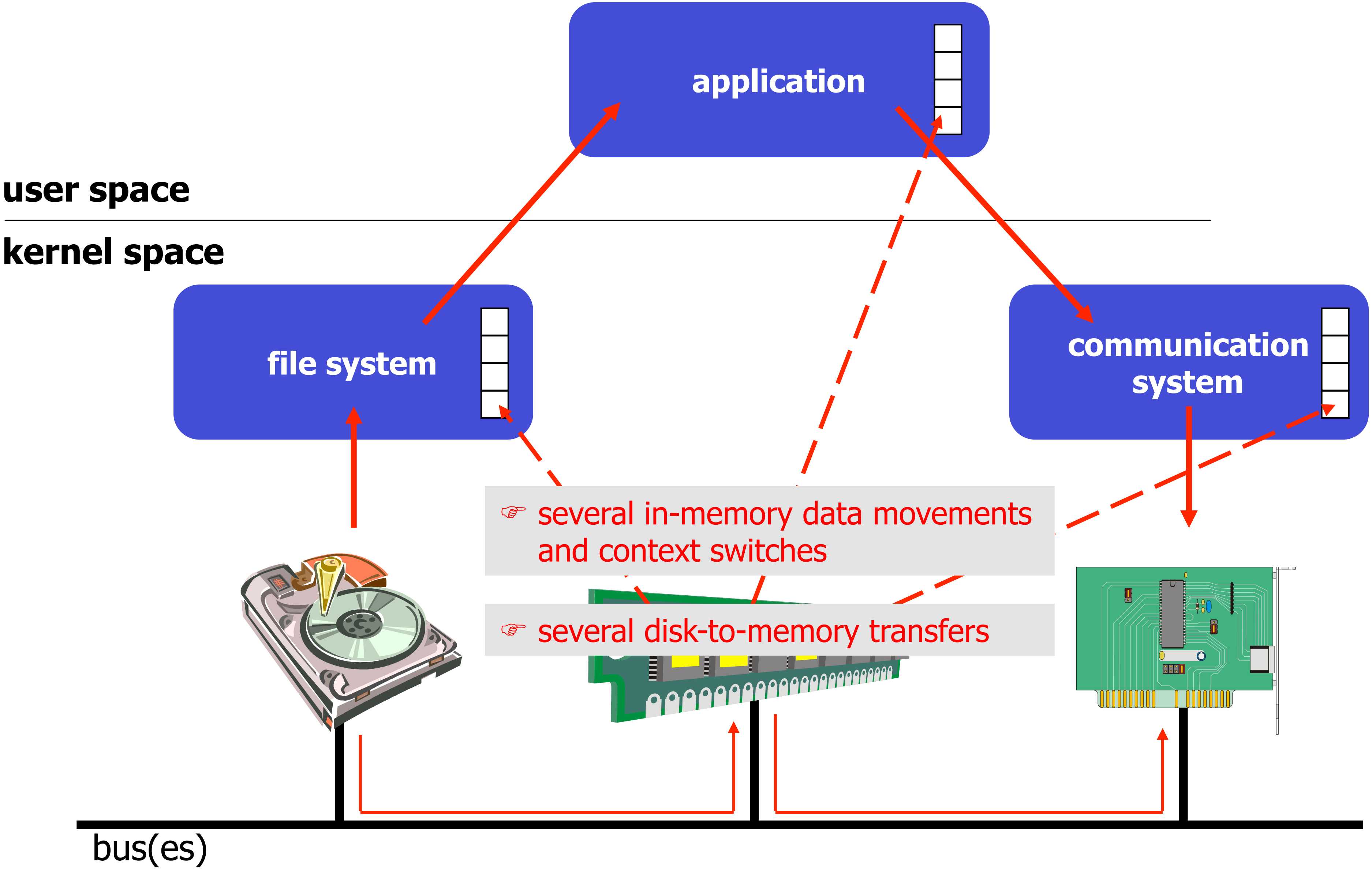
Lack of space (or bandwidth) can delay applications

- e.g., the dining philosophers would die because the spaghetti-chef could not find a parking lot

Delivery Systems



Delivery Systems



- several in-memory data movements and context switches

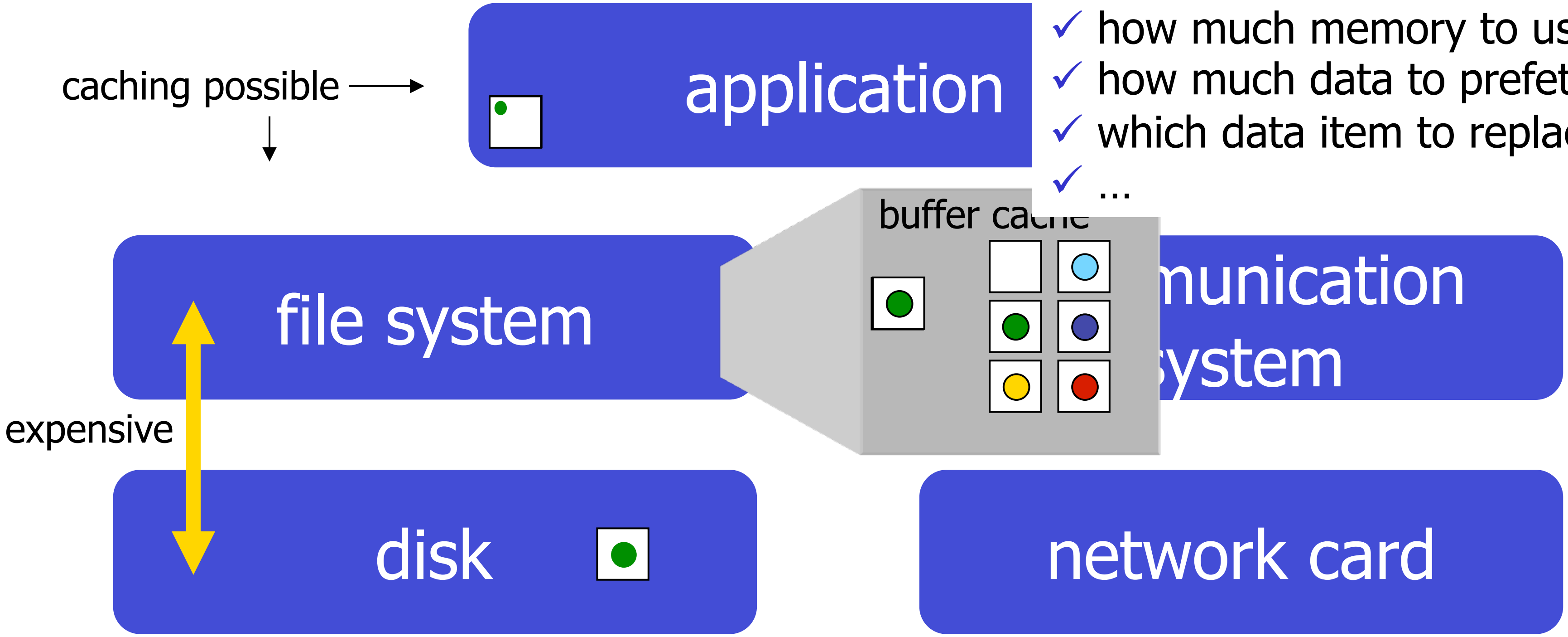
- several disk-to-memory transfers

Memory Caching

Memory Caching

How do we manage a cache?

- ✓ how much memory to use?
- ✓ how much data to prefetch?
- ✓ which data item to replace?
- ✓ …



vs.

Is Caching Useful in a High-Rate Scenario?

- High rate data may need lots of memory for caching…

Buffer vs. Rate	**160 Kbps** (e.g., MP3)	**1.4 Mbps** (e.g., uncompressed CD)	**3.5 Mbps** (e.g., average DVD video)	**100 Mbps** (e.g., uncompressed HDTV)
100 MB	85 min 20 s	9 min 31 s	3 min 49 s	8 s
1 GB	14 hr 33 min 49 s	1 hr 37 min 31 s	39 min 01 s	1 min 20 s
16 GB	133 hr 01 min 01 s	26 hr 00 min 23 s	10 hr 24 min 09 s	21 min 20 s
32 GB	266 hr 02 min 02 s	52 hr 00 min 46 s	20 hr 48 min 18 s	42 min 40 s
128 GB	1064 hr 08 min 08 s	208 hr 03 min 04 s	83 hr 13 min 12 s	2 hr 50 min 40 s

Largest **Dell** Servers in **2004**/**2008** –
and all is NOT used for caching

- Tradeoff: amount of memory, algorithms complexity, gain, …

- Cache only frequently used data – how?
  (e.g., first (small) parts of a movie, allow "top-ten" only, …)

Need For Application-Specific Algorithms?

- Most existing systems use an LRU-variant
  - keep a sorted list (most recently used at the head)
  - replace last element in list
  - insert new data elements at the head
  - if a data element is re-accessed (e.g., new client or rewind), move back to the end of the list

In this case, LRU replaces the next needed frame. So the answer is in many cases **YES…**

- Extreme example – video frame playout:

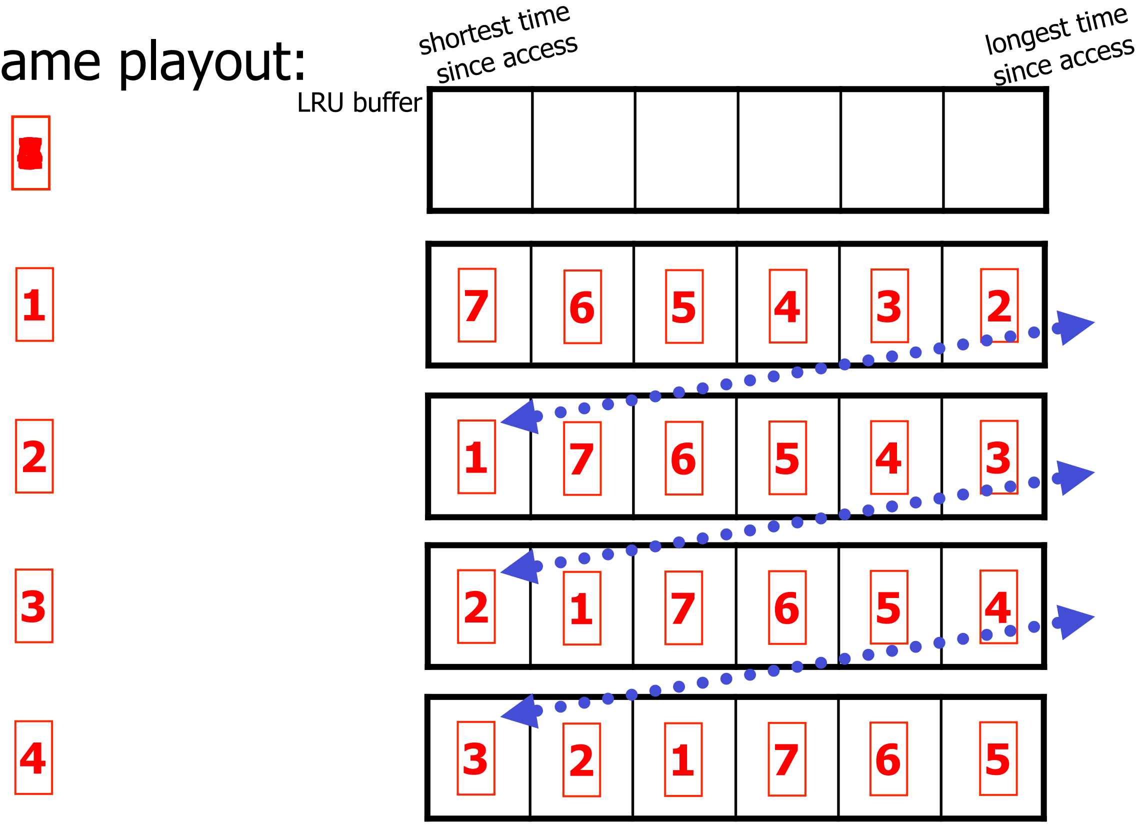
play video (7 frames):

rewind and restart playout at 1:

playout 2:

playout 3:

playout 4:



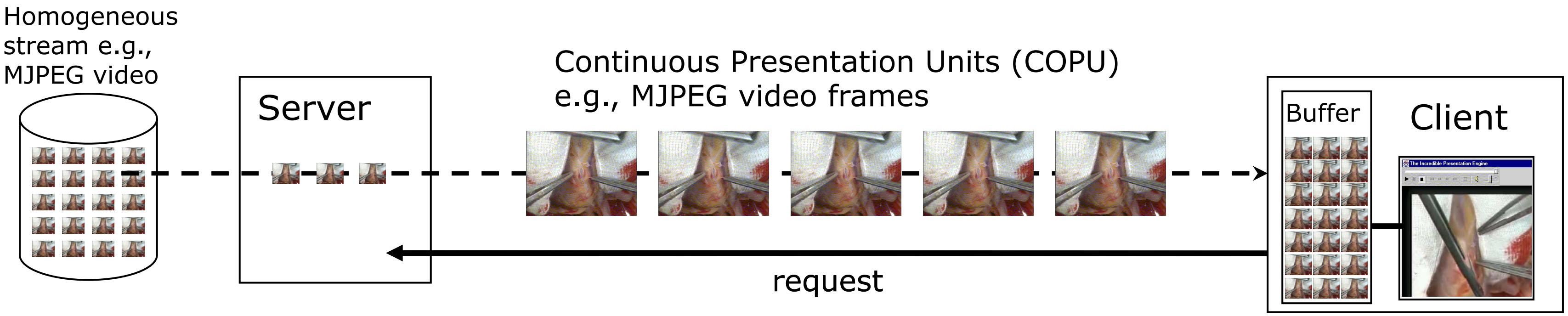
“Classification” of Mechanisms

- Block-level caching consider (possibly unrelated) set of blocks
  - each data element is viewed upon as an independent item
  - usually used in “traditional” systems
  - e.g., FIFO, LRU, LFU, CLOCK, …
  - multimedia (video) approaches:
    - *Least/Most Relevant for Presentation* (L/MRP)
    - …
- Stream-dependent caching consider (parts of) a stream object as a whole
  - related data elements are treated in the same way
  - research prototypes in multimedia systems
  - e.g.,
    - BASIC
    - DISTANCE
    - *Interval Caching* (IC)
    - *Generalized Interval Caching* (GIC)
    - Split and Merge (SAM)
    - SHR

Least/Most Relevant for Presentation (L/MRP)

[Moser et al. 95]

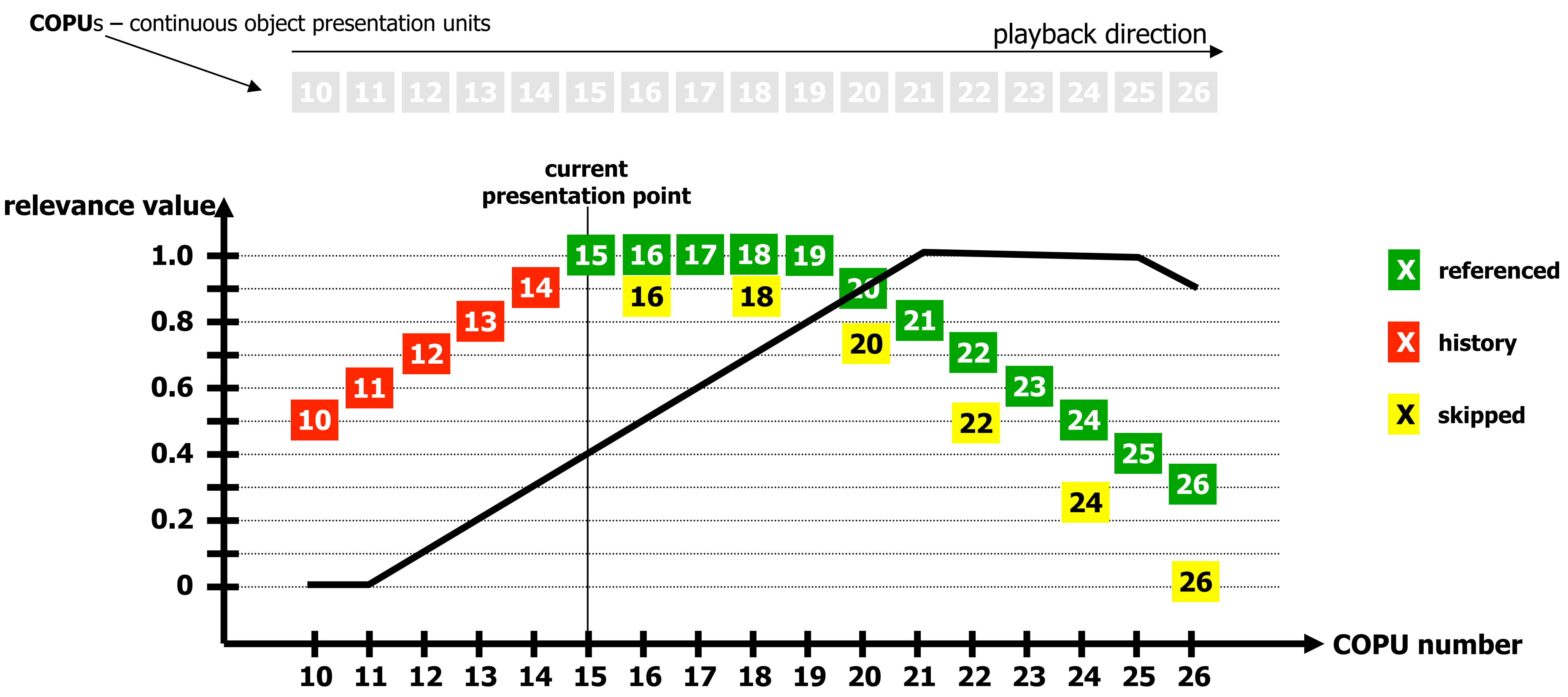
- L/MRP is a buffer management mechanism for a single interactive, continuous data stream
  - adaptable to individual multimedia applications
  - preloads units *most relevant for presentation* from disk
  - replaces units *least relevant for presentation*
  - client pull based architecture


Homogeneous stream e.g., MJPEG video
Server
Continuous Presentation Units (COPU) e.g., MJPEG video frames
Buffer
Client
request

Least/Most Relevant for Presentation (L/MRP)

[Moser et al. 95]

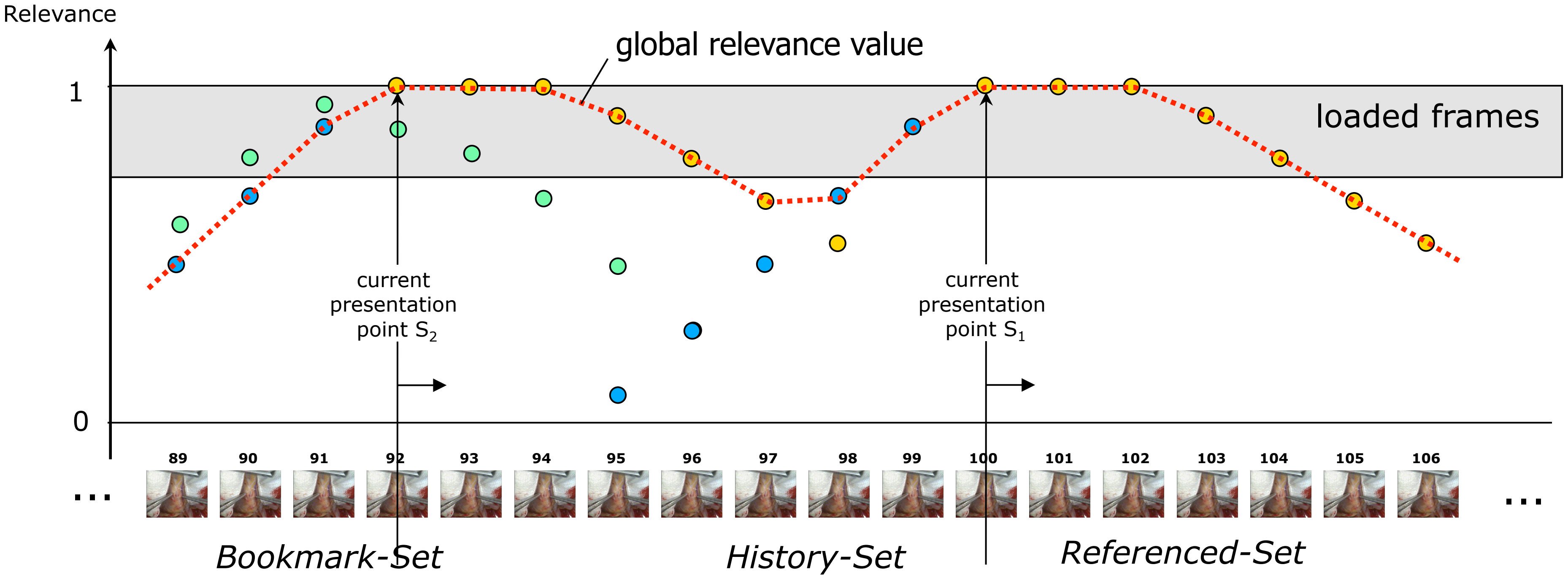
- Relevance values are calculated with respect to current playout of the multimedia stream
    - presentation point (current position in file)
    - mode / speed (forward, backward, FF, FB, jump)
    - relevance functions are configurable



Least/Most Relevant for Presentation (L/MRP)

[Moser et al. 95]

- Global relevance value
  - each COPU can have more than one relevance value
    - bookmark sets (known interaction points)
    - several viewers (clients) of the same
  - = *maximum relevance for each COPU*



Least/Most Relevant for Presentation (L/MRP)

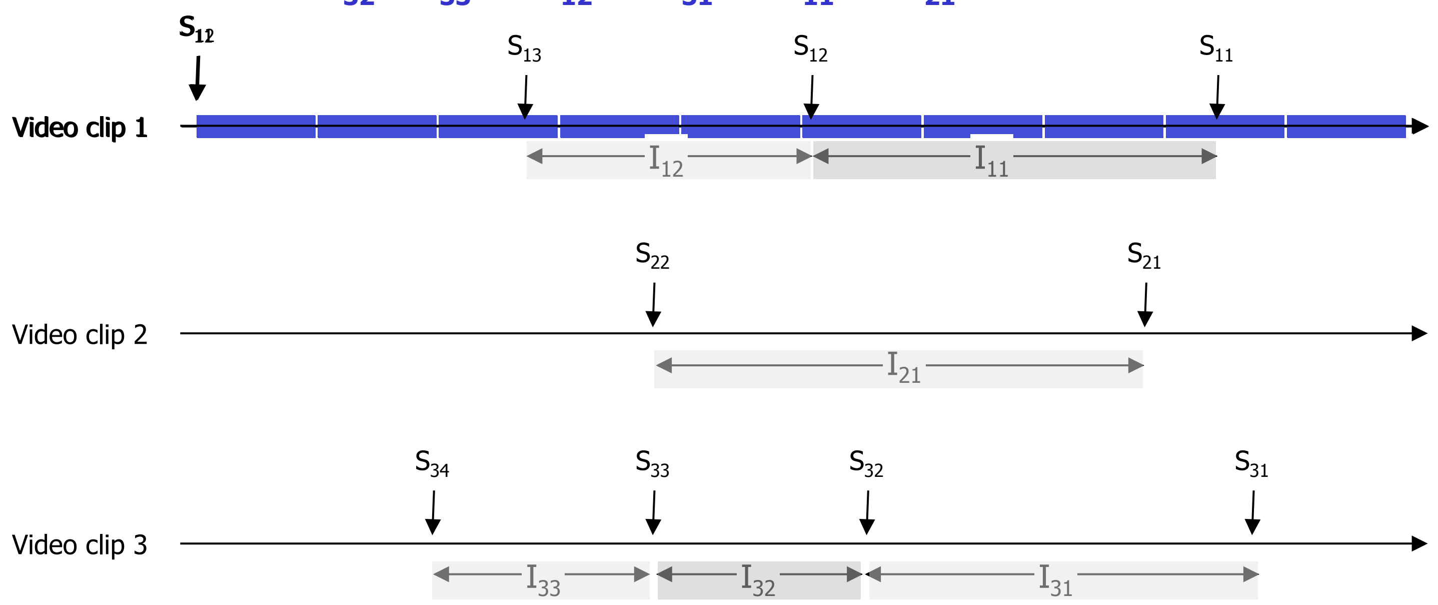
- L/MRP …
  - ☺ … gives "few" disk accesses (compared to other schemes)
  - ☺ … supports interactivity
  - ☺ … supports prefetching
  - ☹ … targeted for single streams (users)
  - ☹ … expensive (!) to execute
    (calculate relevance values for all COPUs each round)

- Variations:
  - Q-L/MRP – extends L/MRP with multiple streams and changes prefetching mechanism (reduces overhead) [Halvorsen et. al. 98]
  - MPEG-L/MRP – gives different relevance values for different MPEG frames [Boll et. all. 00]

Interval Caching (IC)

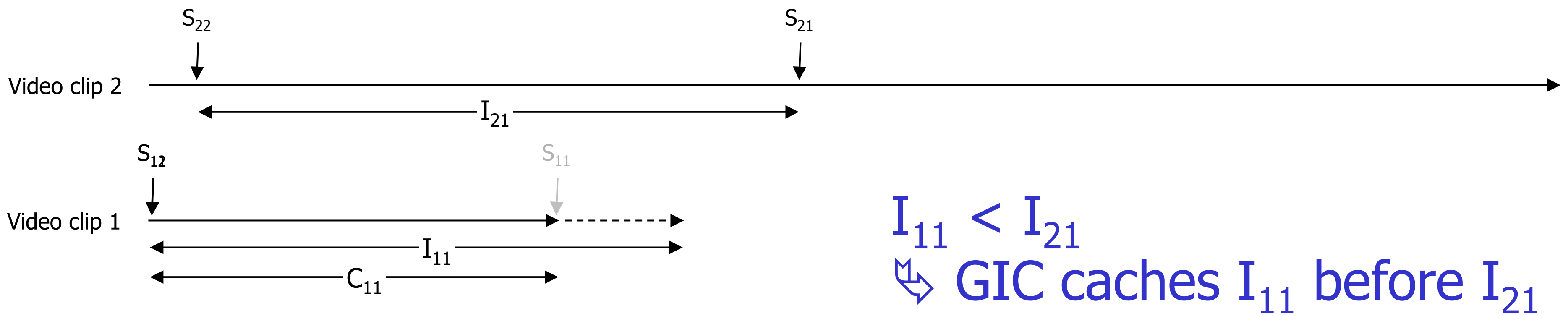
- Interval caching (IC) is a caching strategy for streaming servers
  - caches data between requests for same video stream – based on playout intervals between requests
  - *following* requests are thus served from the cache filled by *preceding* stream
  - sort intervals on length, buffer requirement is data size of interval
  - to maximize cache hit ratio (minimize disk accesses) the shortest intervals are cached first

: I_{32} I_{33} I_{12} I_{31} I_{11} I_{21}



Generalized Interval Caching (GIC)

- Interval caching (IC) does not work for short clips
  - a frequently accessed short clip will not be cached
- GIC generalizes the IC strategy
  - manages intervals for long video objects as IC
  - short intervals extend the interval definition
    - keep track of a finished stream for a while after its termination
    - define the interval for short stream as the length between the new stream and the position of the old stream if it had been a longer video object
    - the cache requirement is, however, only the real requirement
  - cache the shortest intervals as in IC



$I_{11} < I_{21}$

⇨ GIC caches I_{11} before I_{21}

Generalized Interval Caching (GIC)

- Open function:

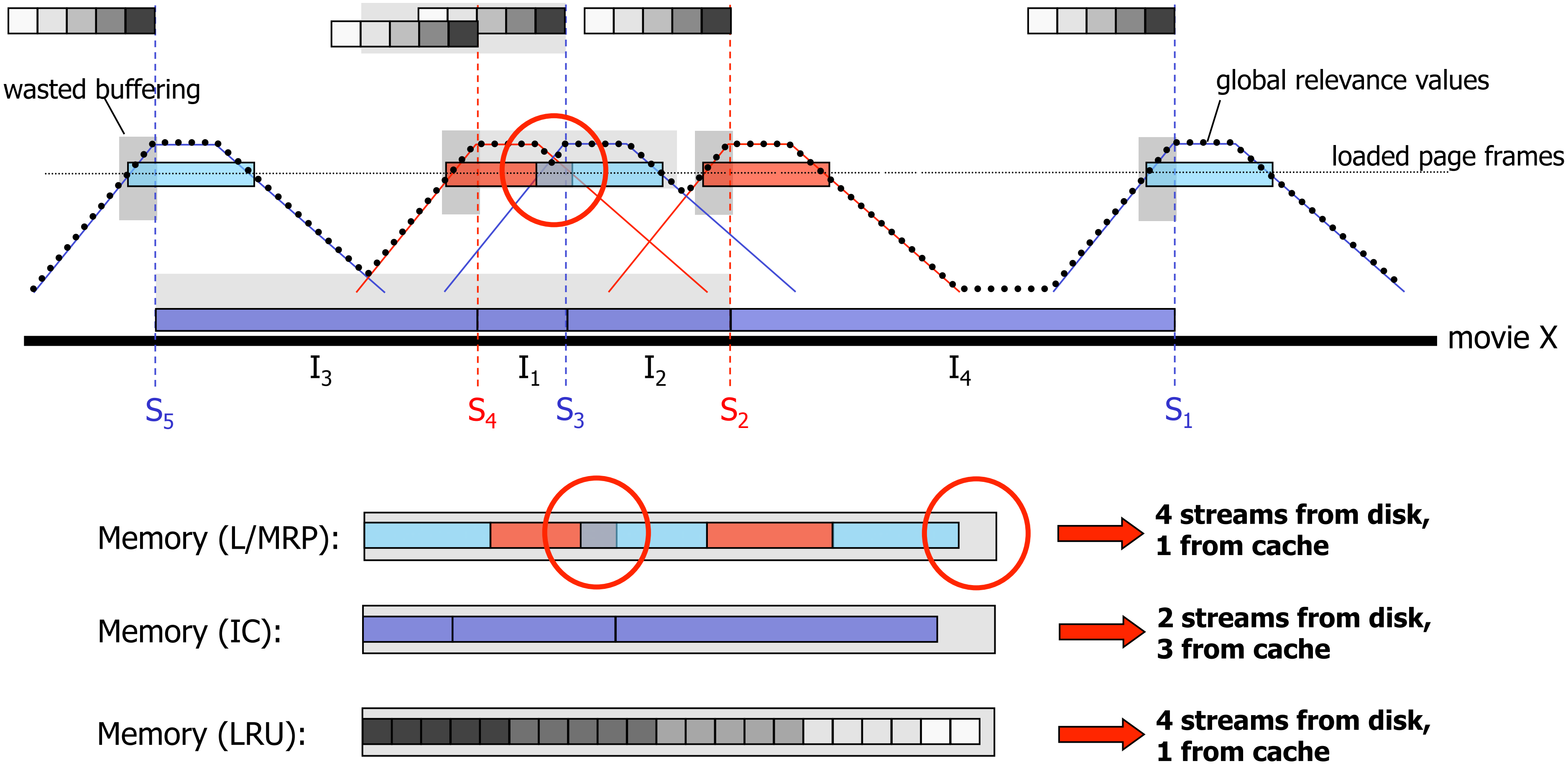
```
form if possible new interval with previous stream;
if (NO) {exit} /* don't cache */
compute interval size and cache requirement;
reorder interval list; /* smallest first */
if (not already in a cached interval) {
      if (space available) {cache interval}
      else if (larger cached intervals exist
      and sufficient memory can be released) {
            release memory from larger intervals;
            cache new interval;
      }
}
```

- Close function

```
if (not following another stream) {exit} /* not served from cache */
delete interval with preceding stream;
free memory;
if (next interval can be cached in released memory) {
      cache next interval
}
```

LRU vs. L/MRP vs. IC Caching

- What kind of caching strategy is best (VoD streaming)?
  - caching effect



Memory (L/MRP): → **4 streams from disk, 1 from cache**

Memory (IC): → **2 streams from disk, 3 from cache**

Memory (LRU): → **4 streams from disk, 1 from cache**

LRU vs. L/MRP vs. IC Caching

- What kind of caching strategy is best (VoD streaming)?
  - caching effect (IC best)
  - CPU requirement

LRU

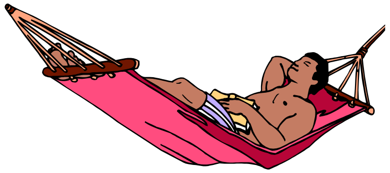
```
for each I/O request
    reorder LRU chain
```

L/MRP

```
for each I/O request
    for each COPU
        RV = 0
        for each stream
            tmp = rel ( COPU, p, mode )
            RV = max ( RV, tmp )
```

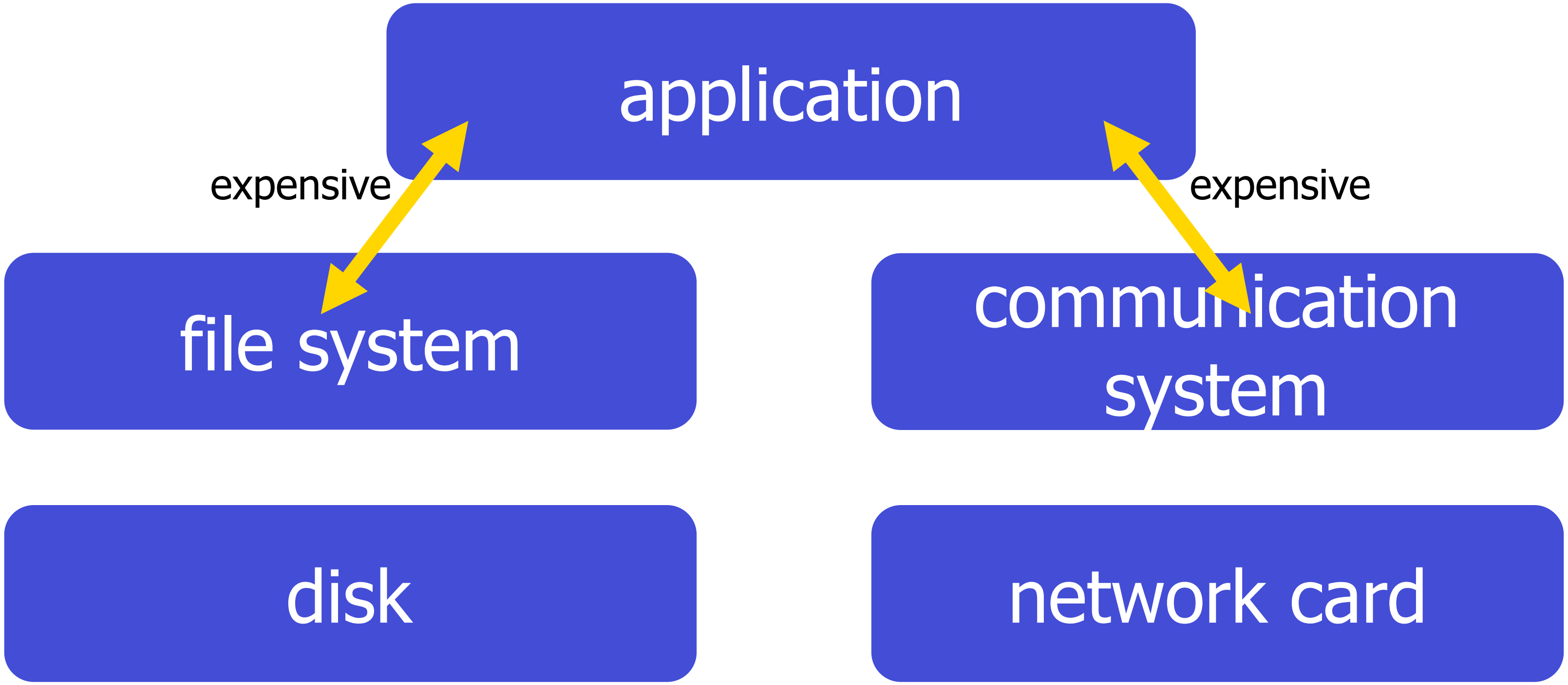
IC

```
for each block consumed
    if last part of interval
        release memory element
```

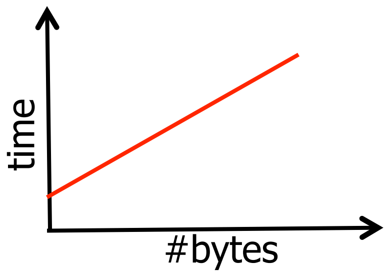


In-Memory Copy Operations

In Memory Copy Operations



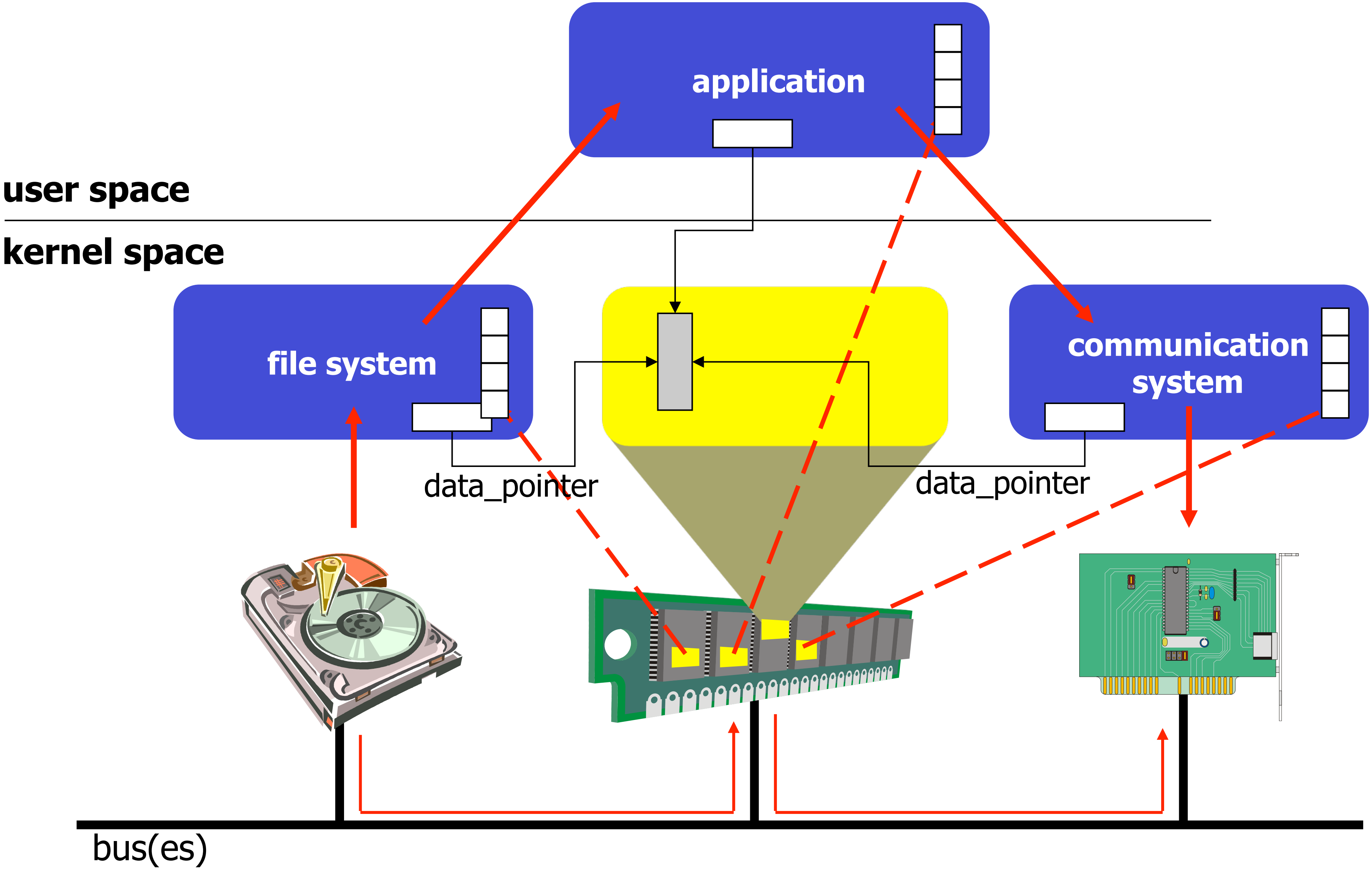
copy:



switch:

- save and restore state
- switch between user and kernel mode
- possible cache/TLB flush
- …

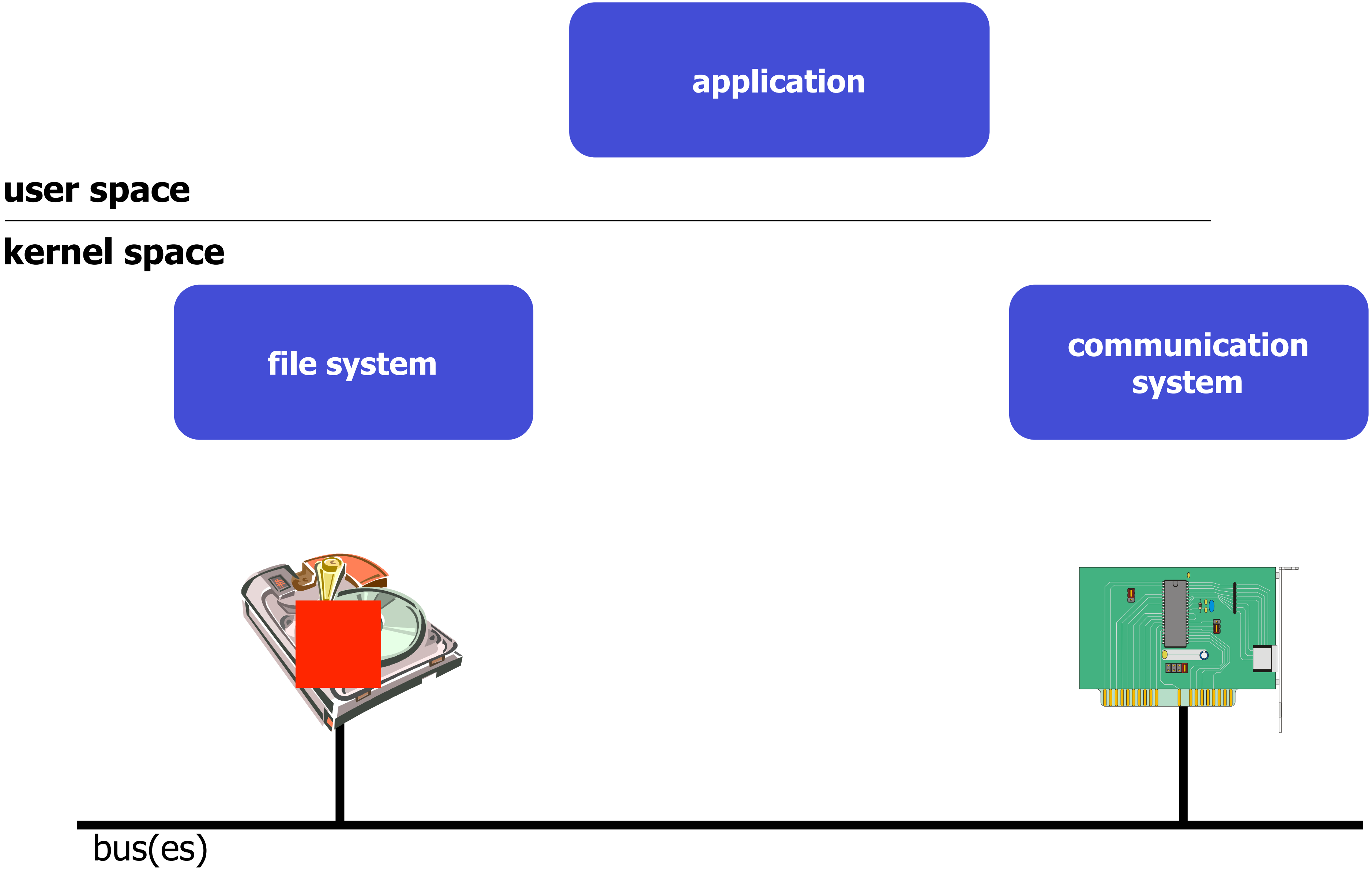
Basic Idea of Zero–Copy Data Paths



A lot of research has been performed in this area!!!!
BUT, what is the status of commodity operating systems?

Existing Linux Data Paths

Content Download



application

user space

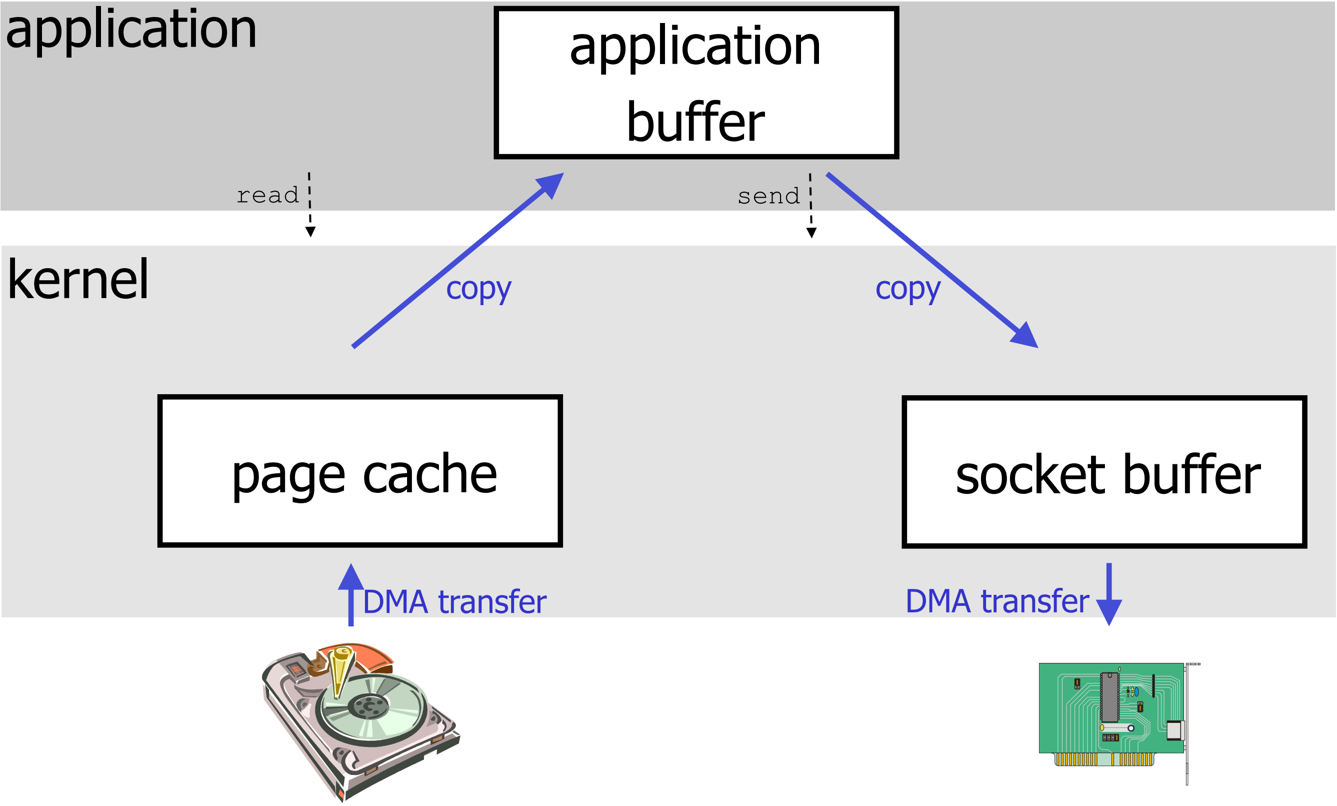
kernel space

file system

communication system

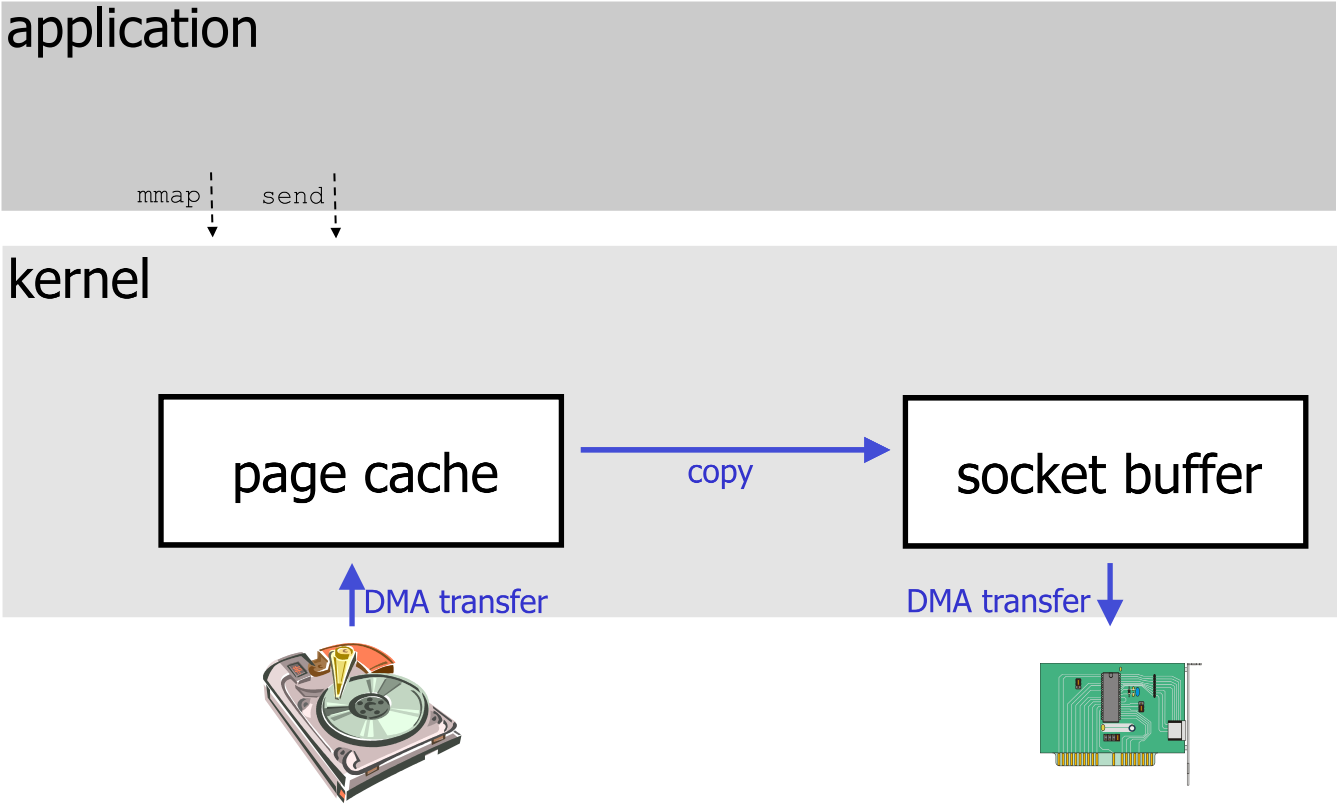
bus(es)

Content Download: `read / send`



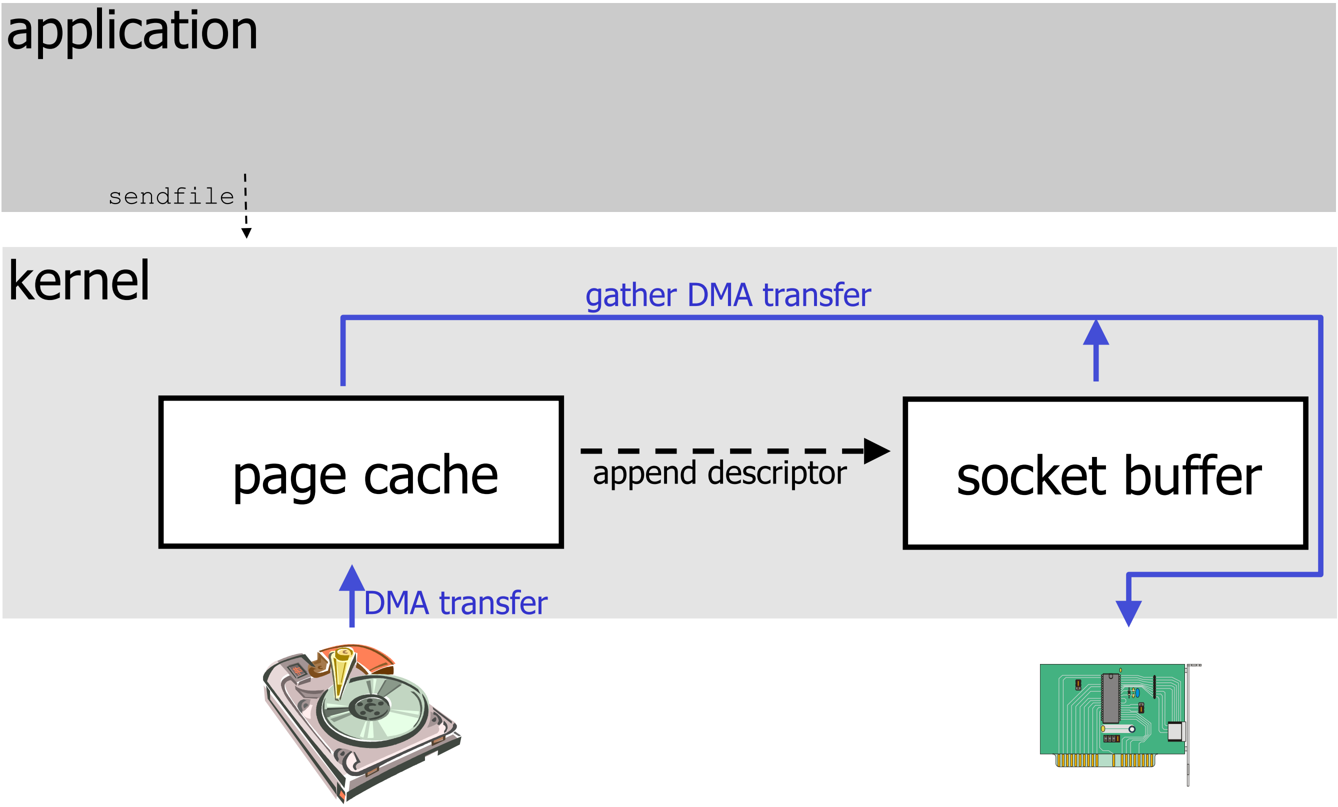
- **2*n*** copy operations
- **2*n*** system calls

Content Download: `mmap / send`



- ***n*** copy operations
- **1 + *n*** system calls

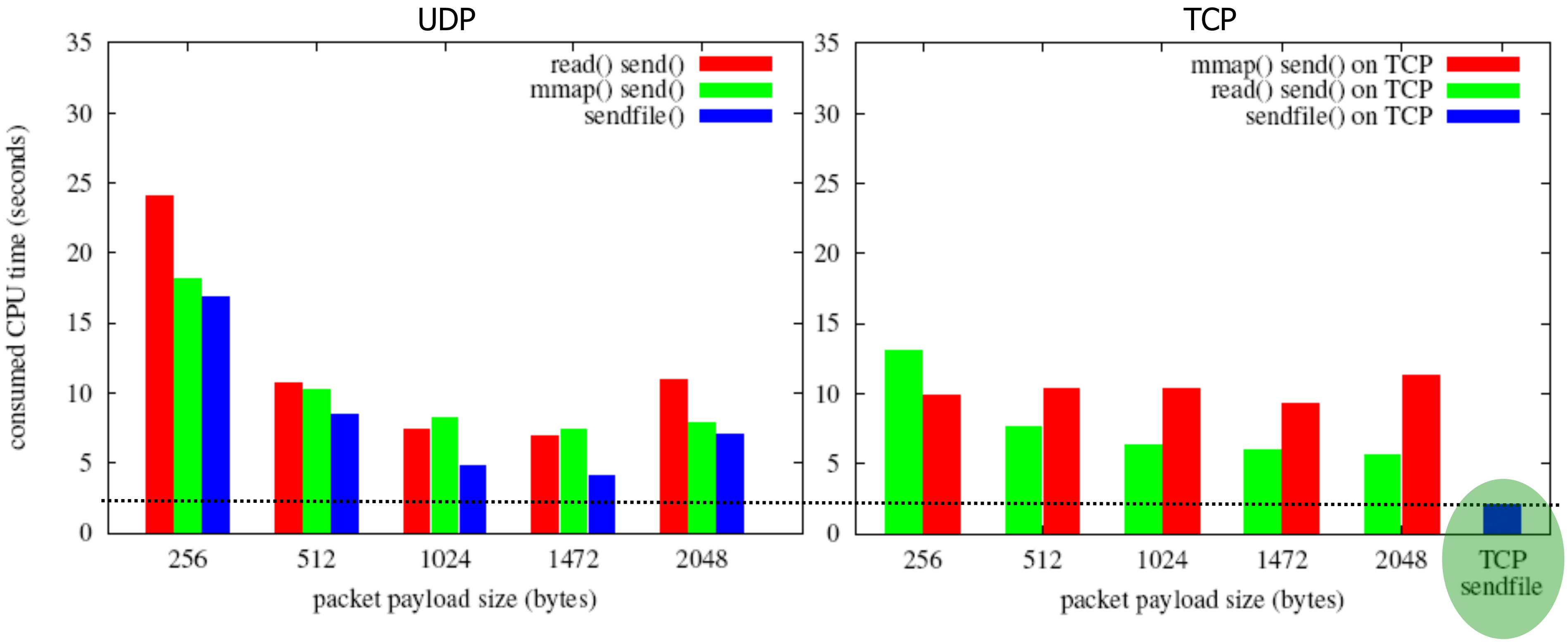
Content Download: `sendfile`



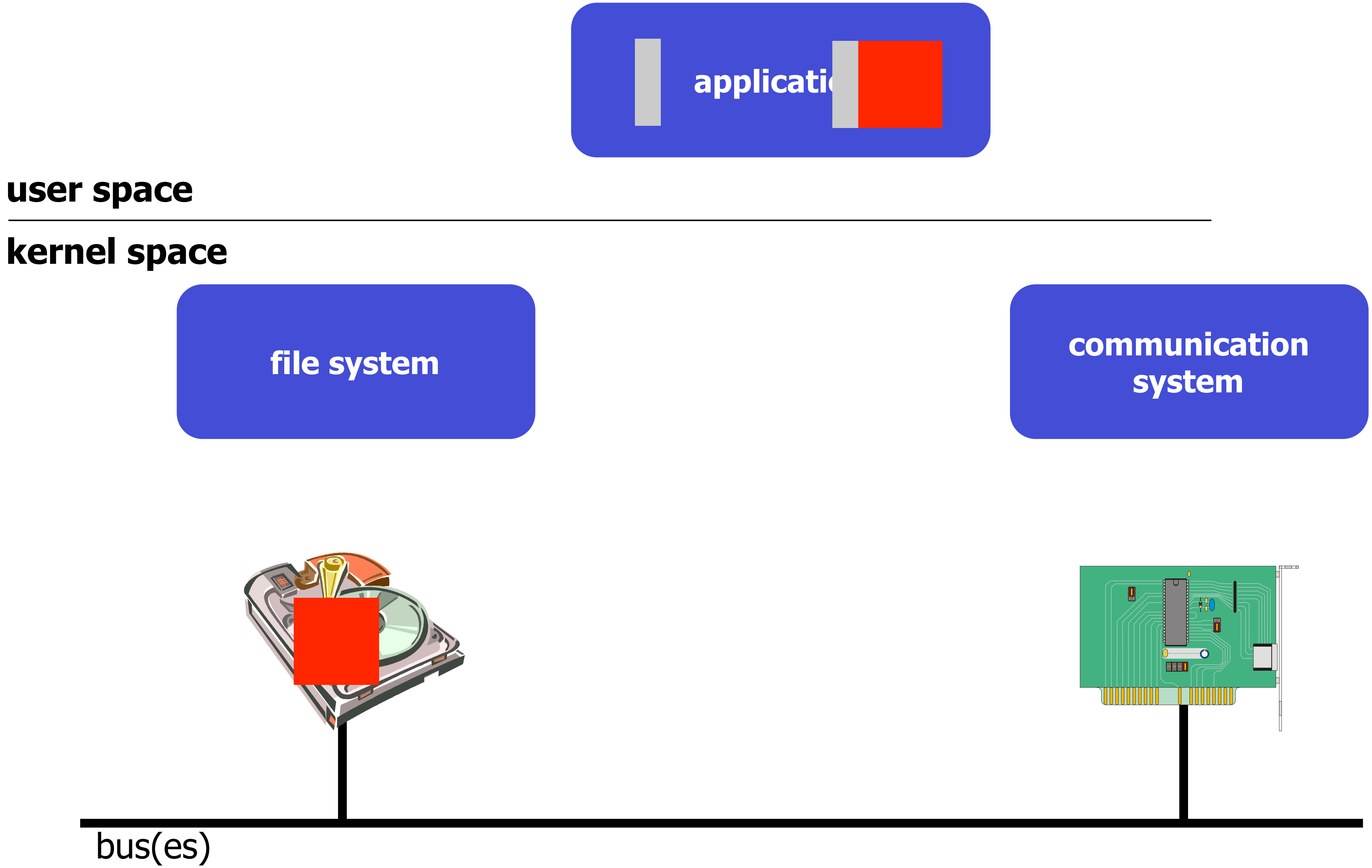
- ***0*** copy operations
- **1** system calls

Content Download: Results

- Tested transfer of 1 GB file on Linux 2.6
- Both UDP and TCP



Streaming



applicati

user space

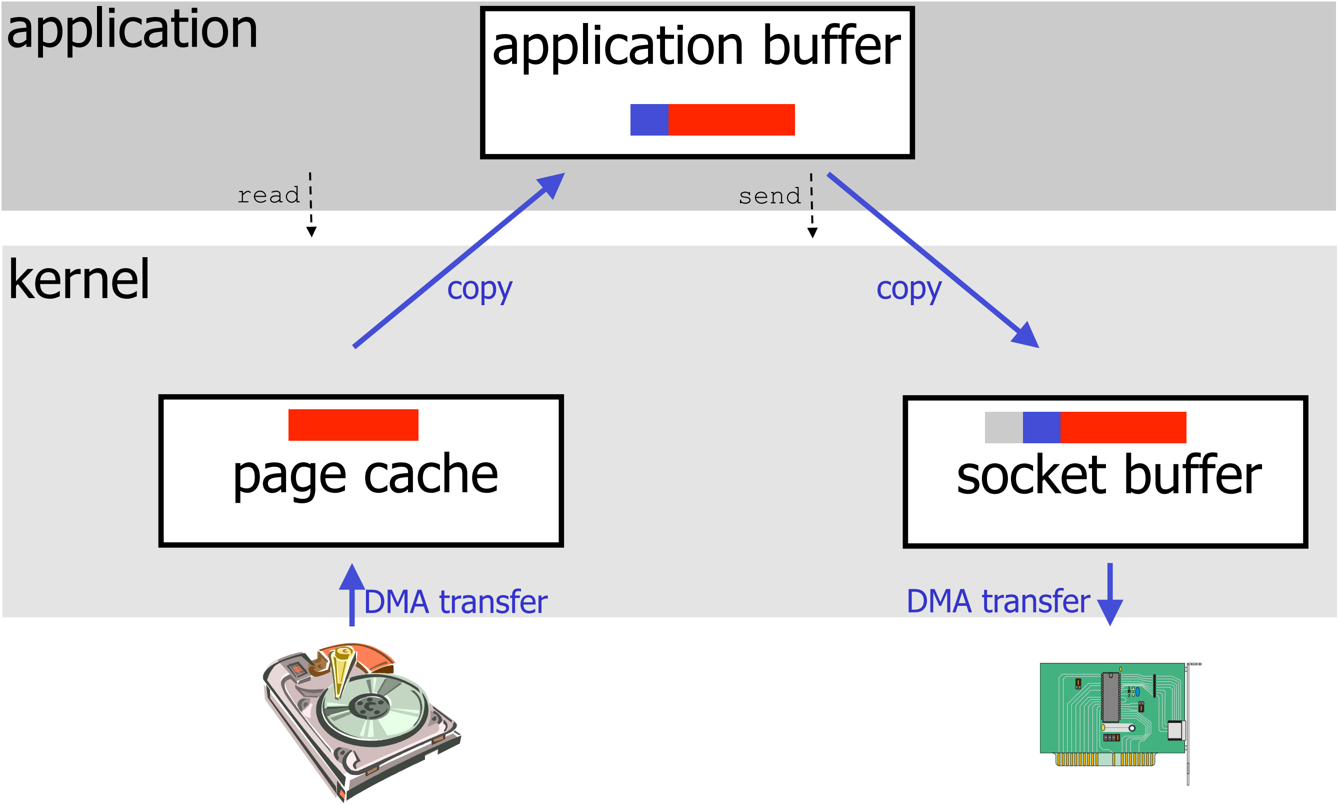
kernel space

file system

communication system

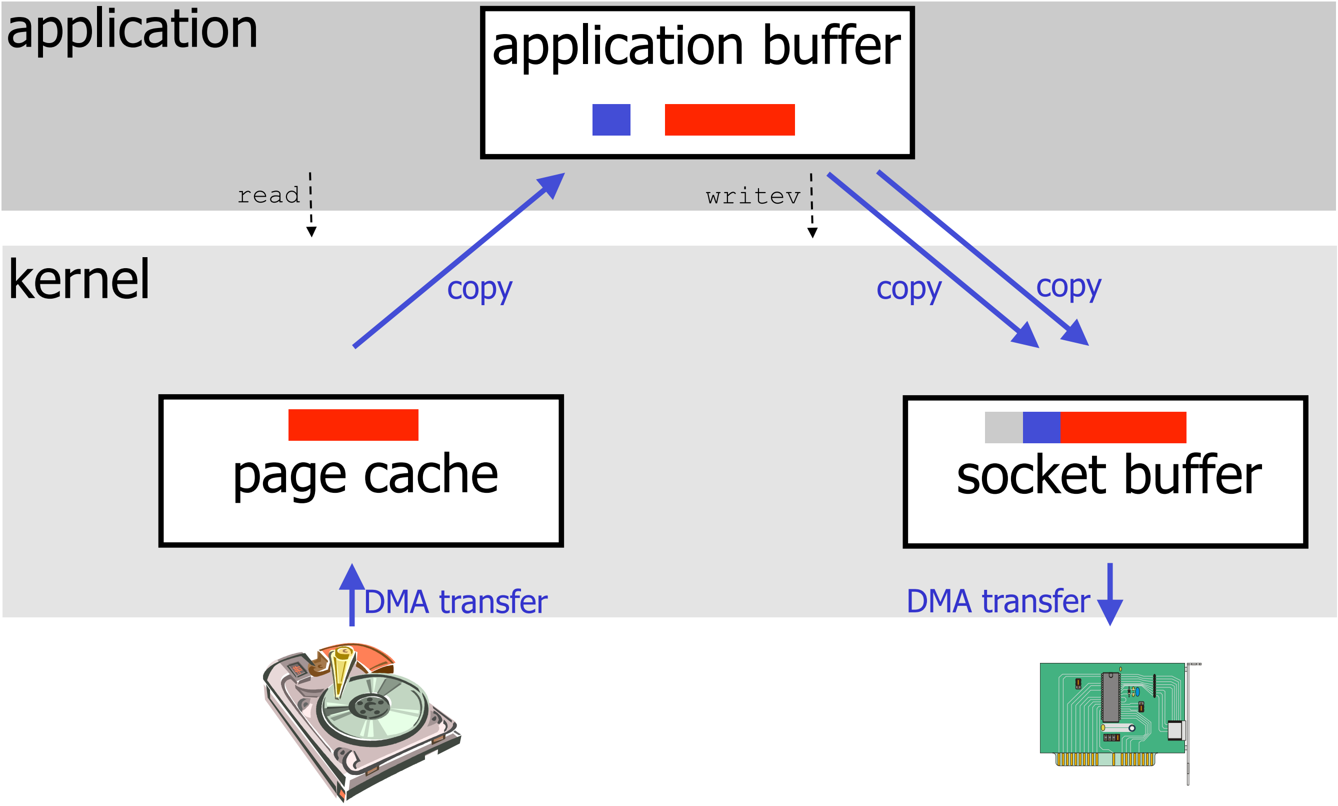
bus(es)

Streaming: `read / send`



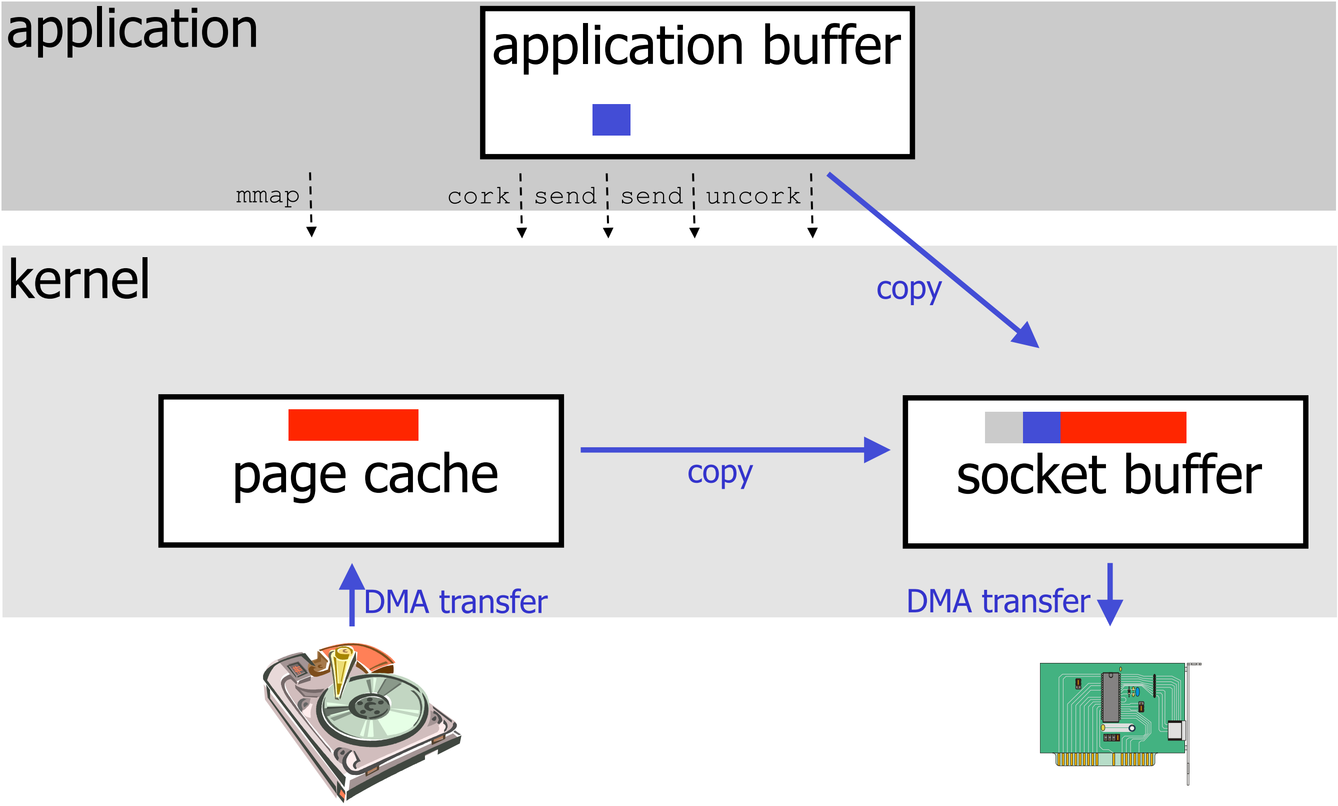
- **2*n*** copy operations
- **2*n*** system calls

Streaming: `read` / `writev`



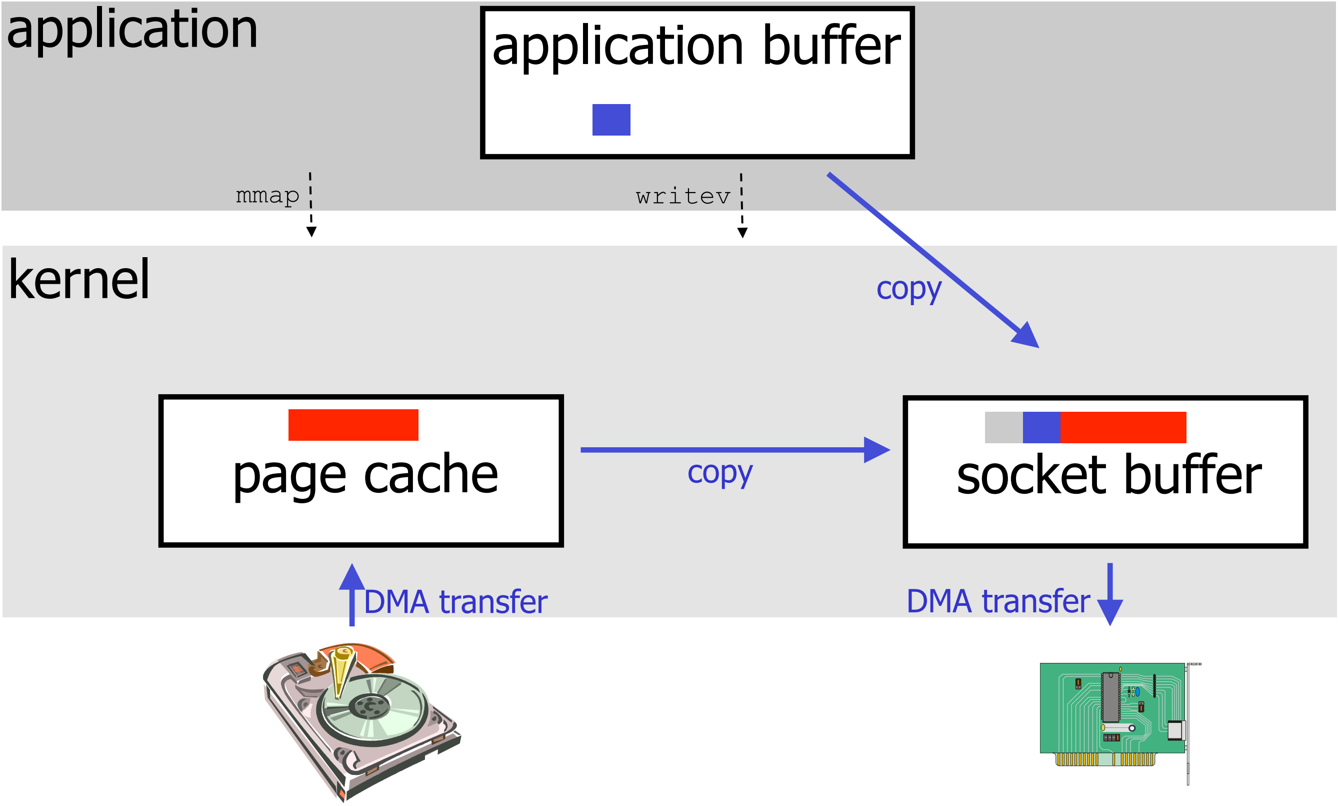
- **3*n*** copy operations ← Previous solution: one less copy per packet
- **2*n*** system calls

Streaming: `mmap` / `send`



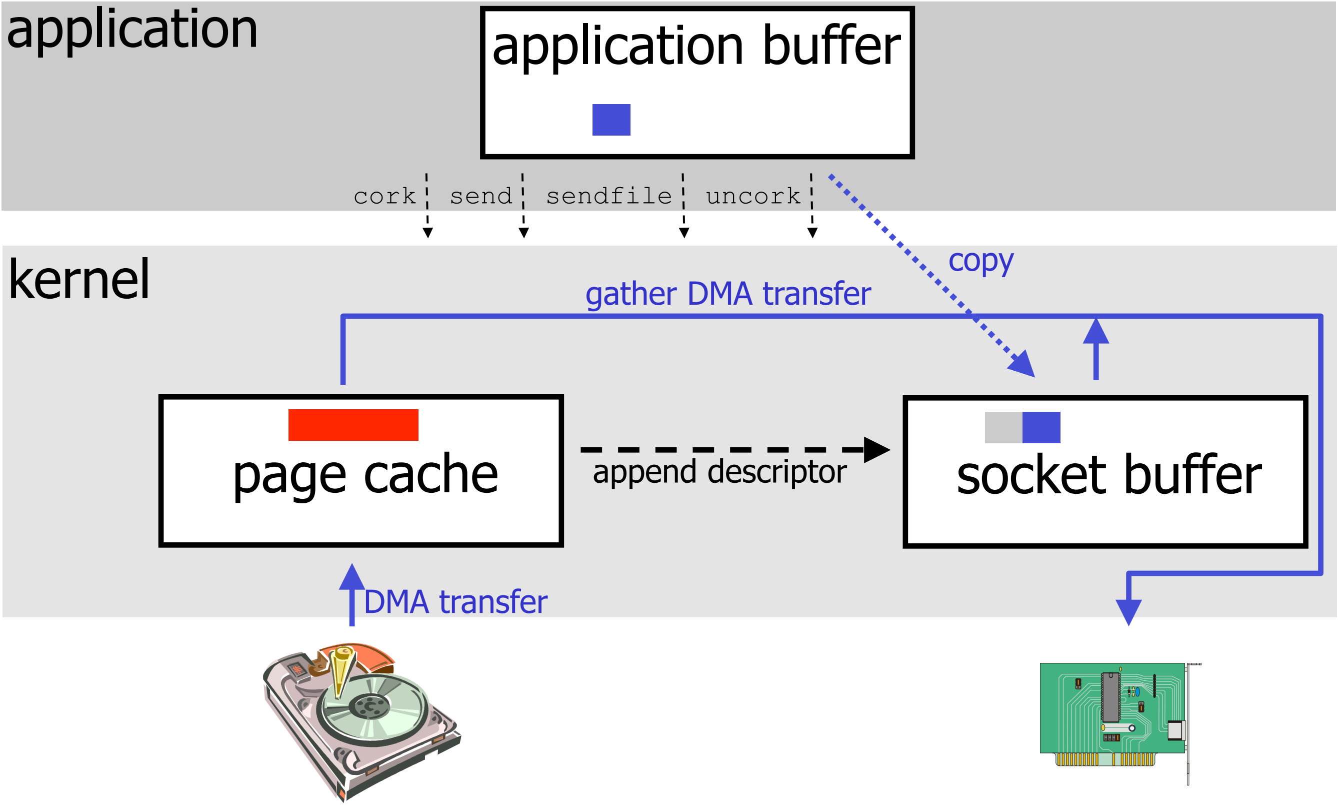
- **2*n*** copy operations (but different costs of header and data)
- **1 + 4*n*** system calls

Streaming: `mmap` / `writev`



- **2*n*** copy operations
- **1 + *n*** system calls← Previous solution: three more calls per packet

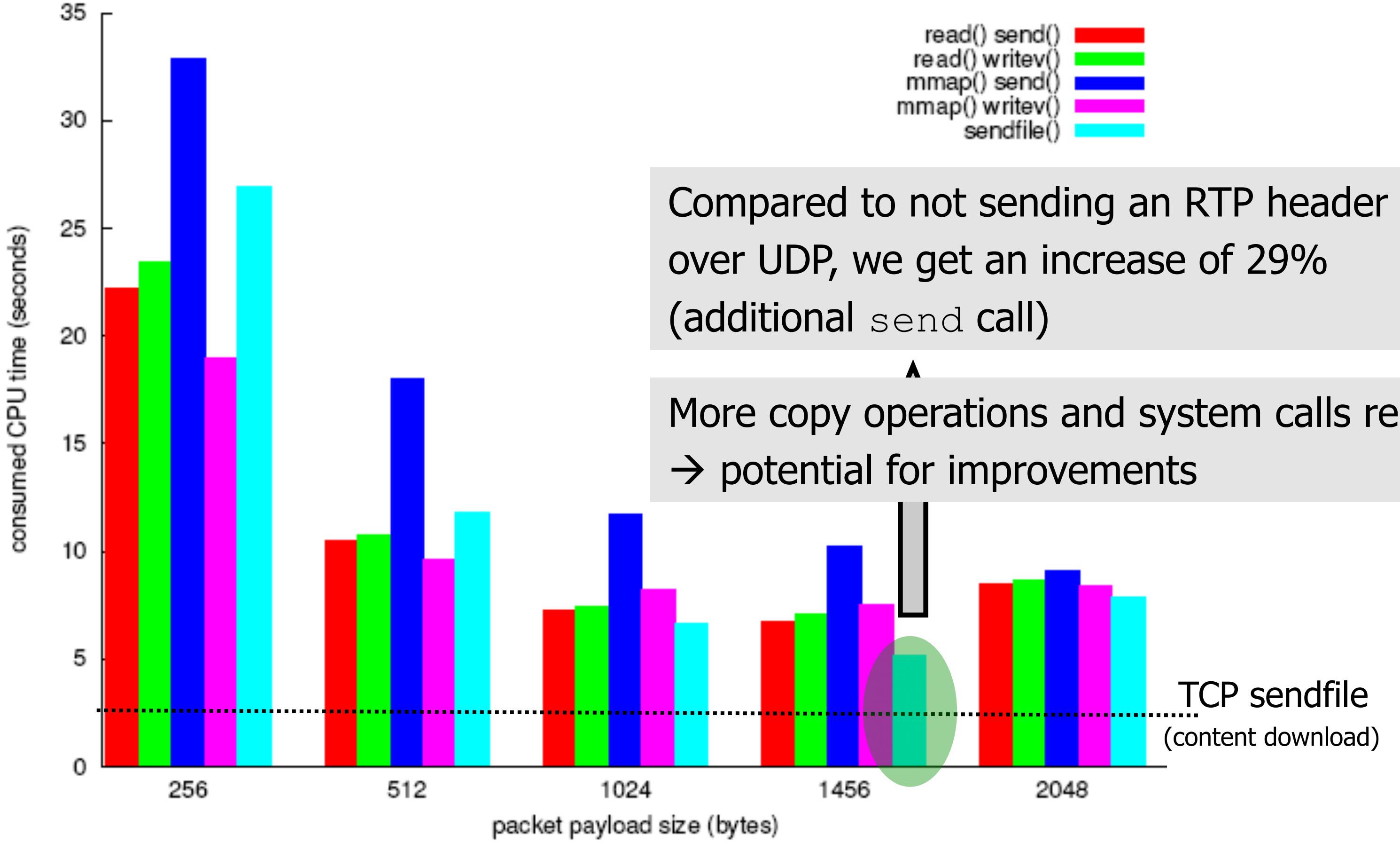
Streaming: `sendfile`



- ***n*** copy operations
- **4*n*** system calls

Streaming: Results

- Tested streaming of 1 GB file on Linux 2.6
- RTP over UDP

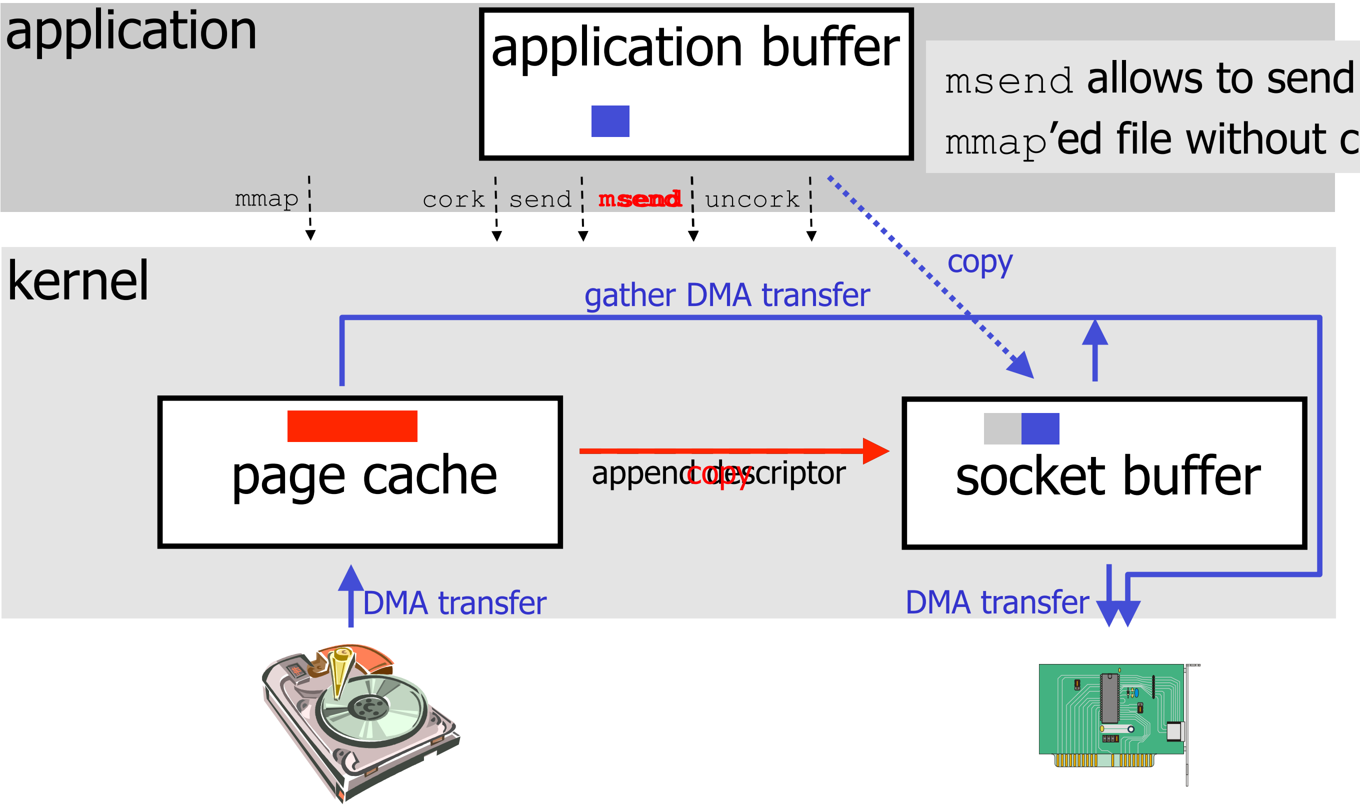


Compared to not sending an RTP header over UDP, we get an increase of 29% (additional `send` call)

More copy operations and system calls required → potential for improvements

Enhanced Streaming Data Paths

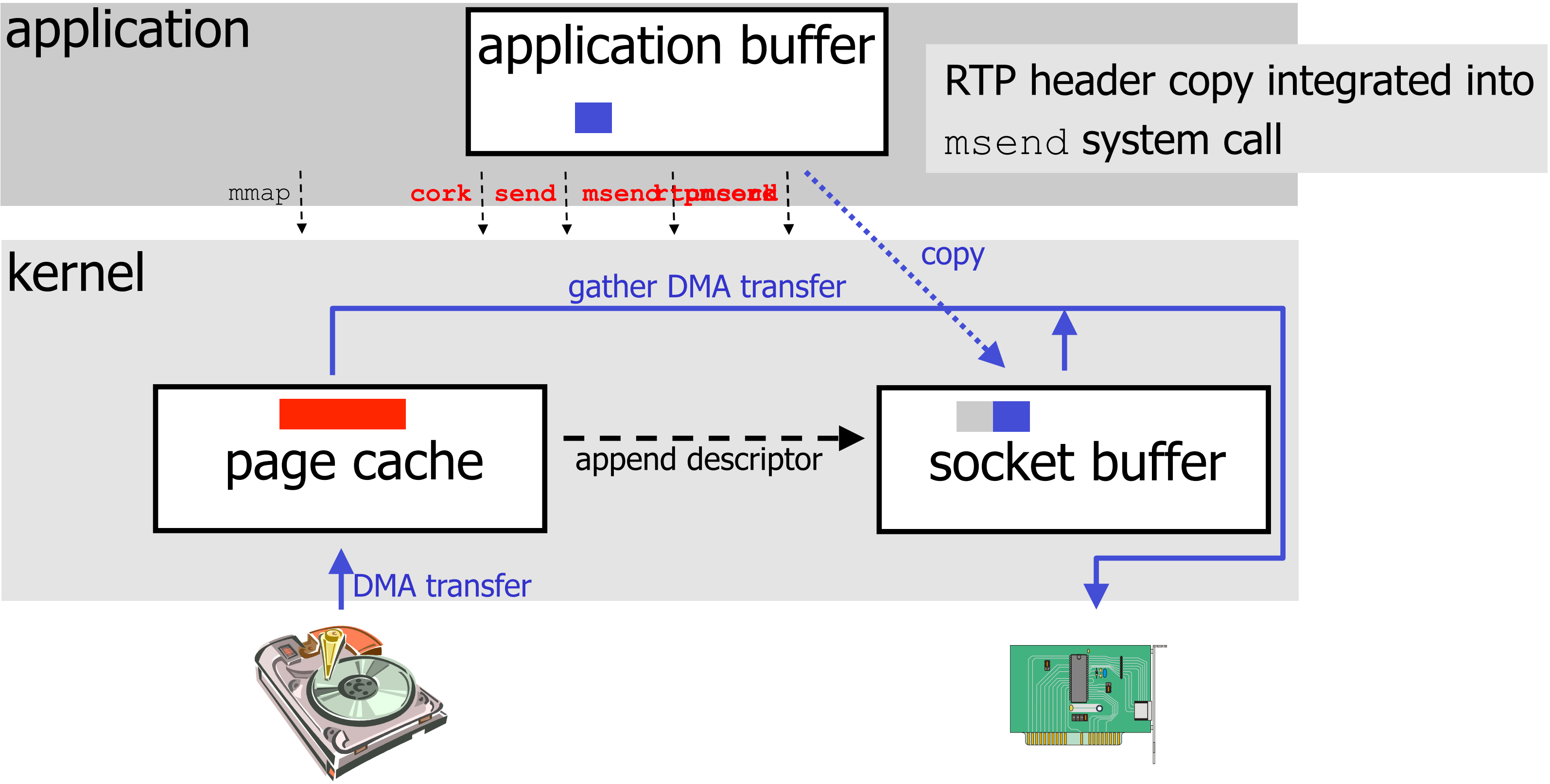
Enhanced Streaming: `mmap` / `msend`



`msend` allows to send data from an `mmap`’ed file without copy

- ➢ ***n*** copy operations ← Previous solution: one more copy per packet
- ➢ ***1 + 4n*** system calls

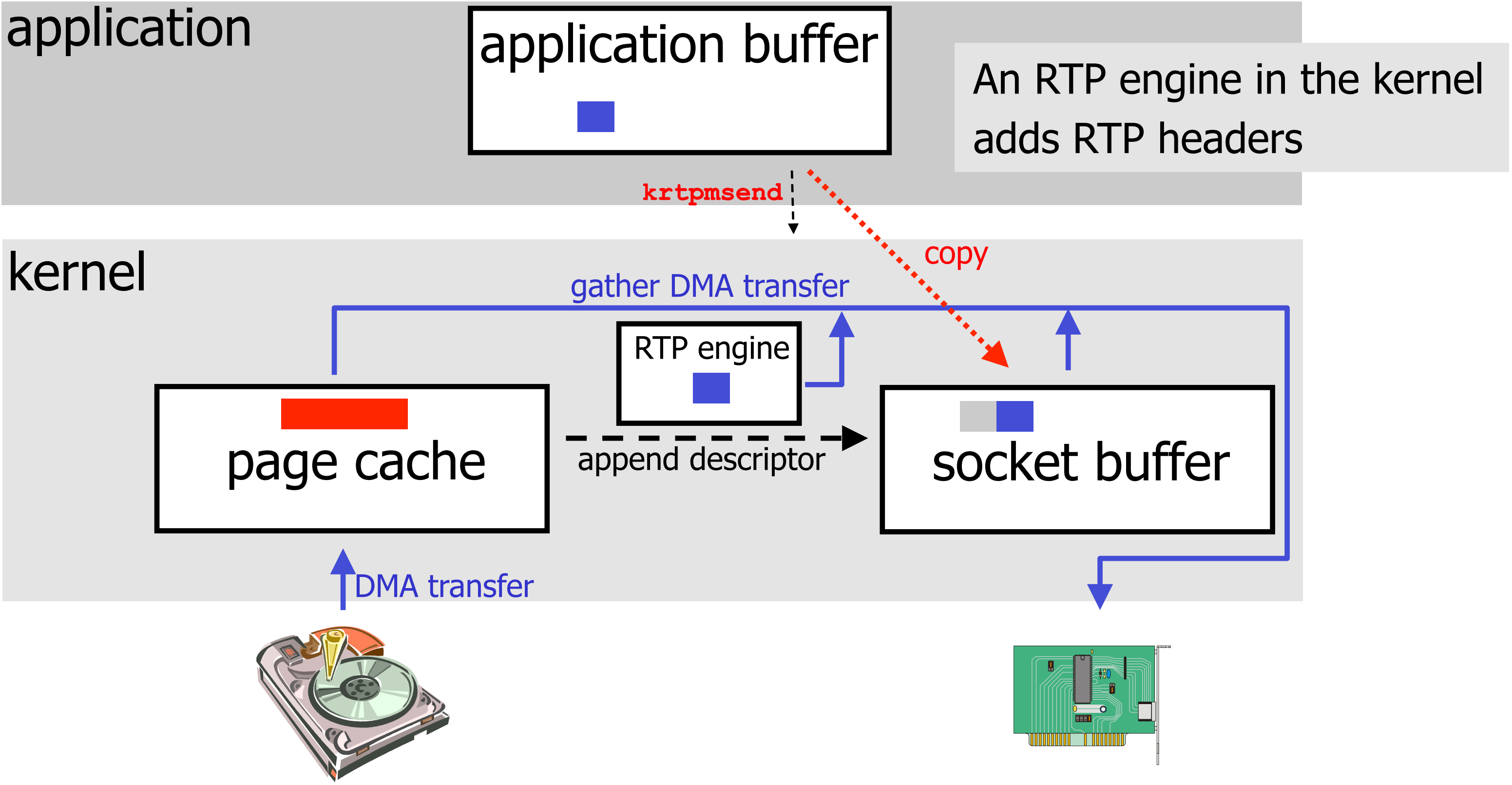
Enhanced Streaming: `mmap` / `rtpmsend`



RTP header copy integrated into `msend` system call

- ***n*** copy operations
- ***1 + n*** system calls ← previous solution: three more calls per packet

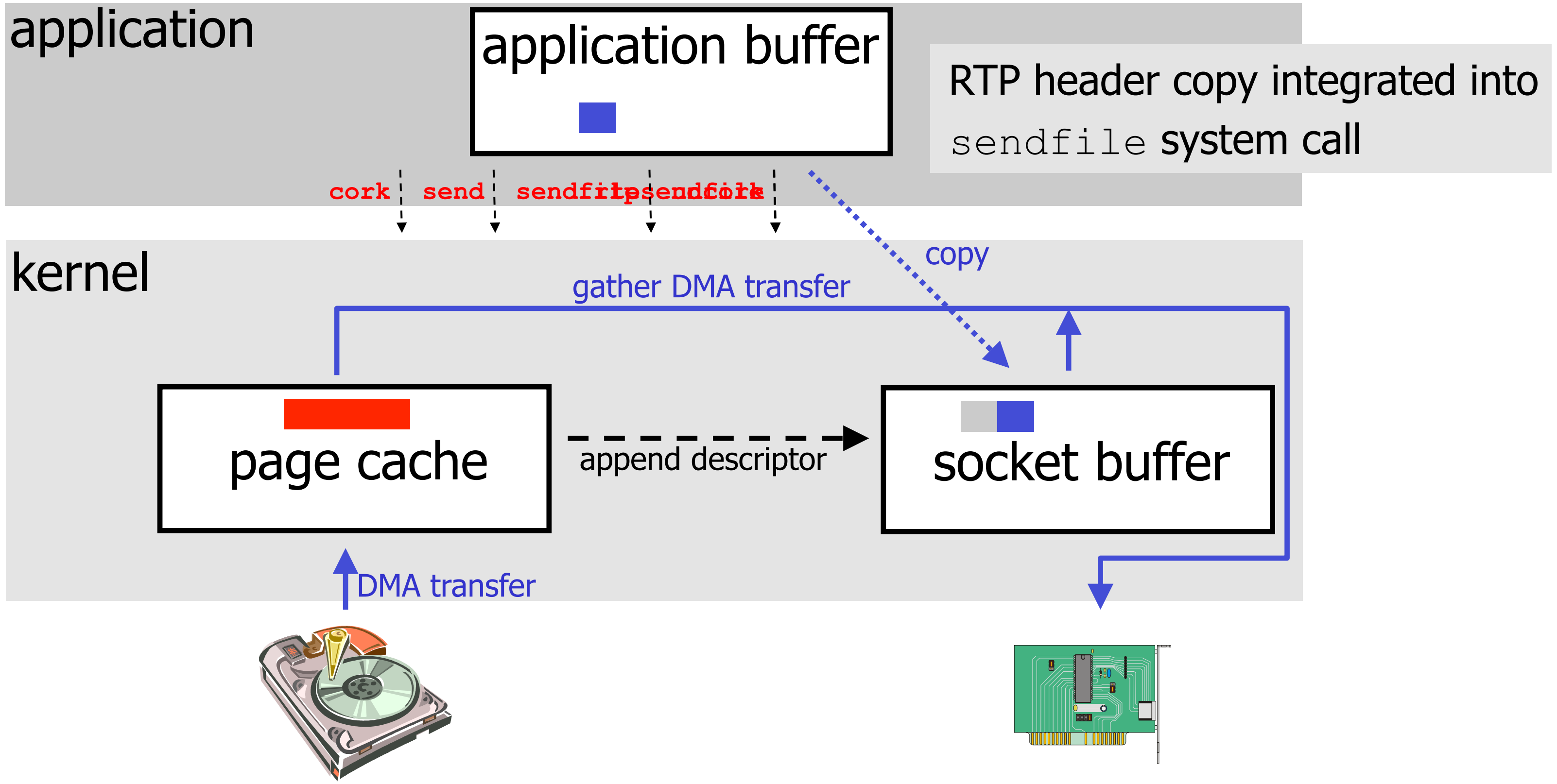
Enhanced Streaming: `mmap` / `krtpmsend`



An RTP engine in the kernel adds RTP headers

- ***0*** copy operations ← previous solution: one more copy per packet
- ***1*** system call ← previous solution: one more call per packet

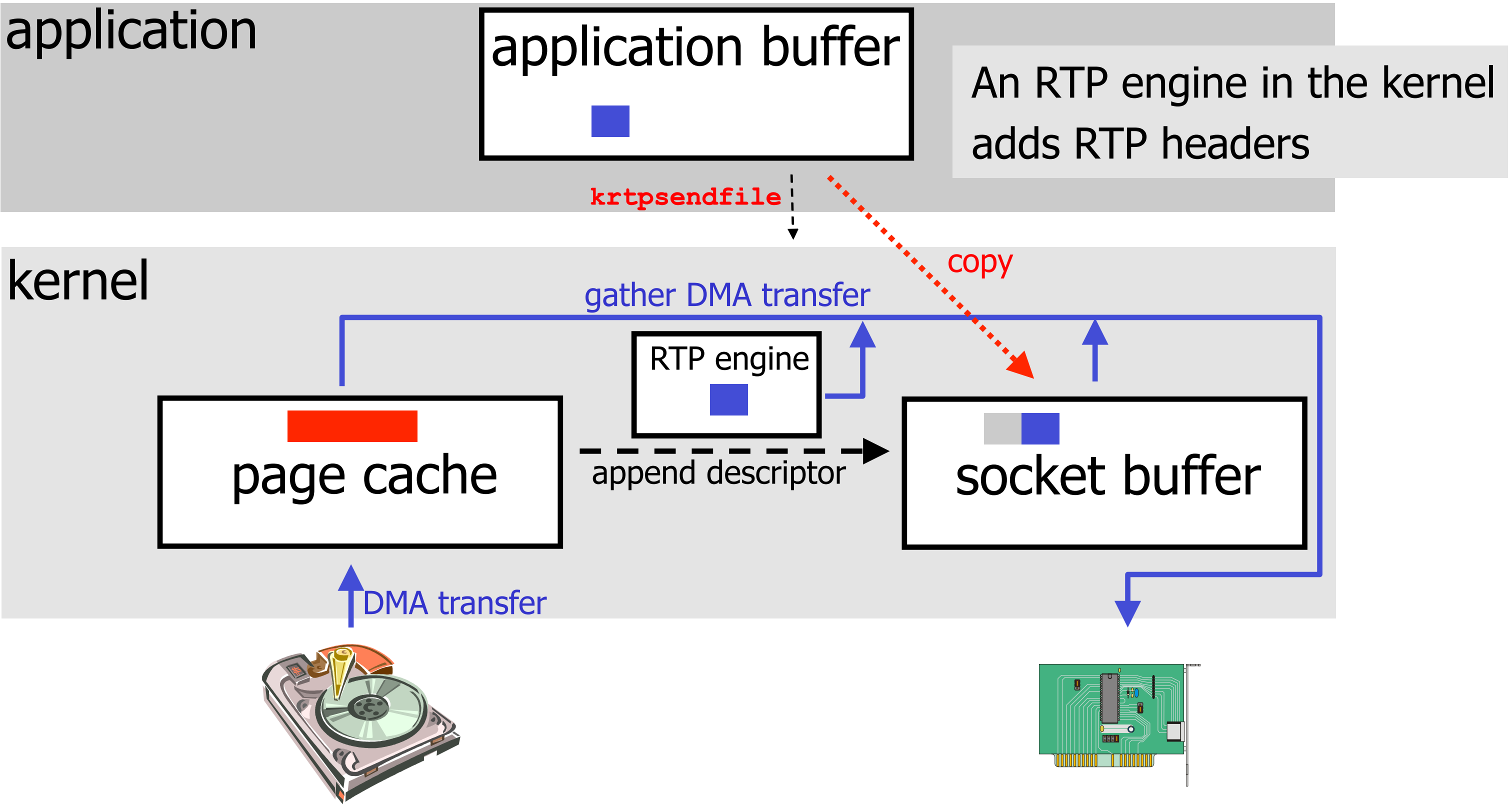
Enhanced Streaming: `rtpsendfile`



RTP header copy integrated into `sendfile` system call

- ➢ ***n*** copy operations
- ➢ ***n*** system calls ← existing solution: three more calls per packet

Enhanced Streaming: `krtpsendfile`

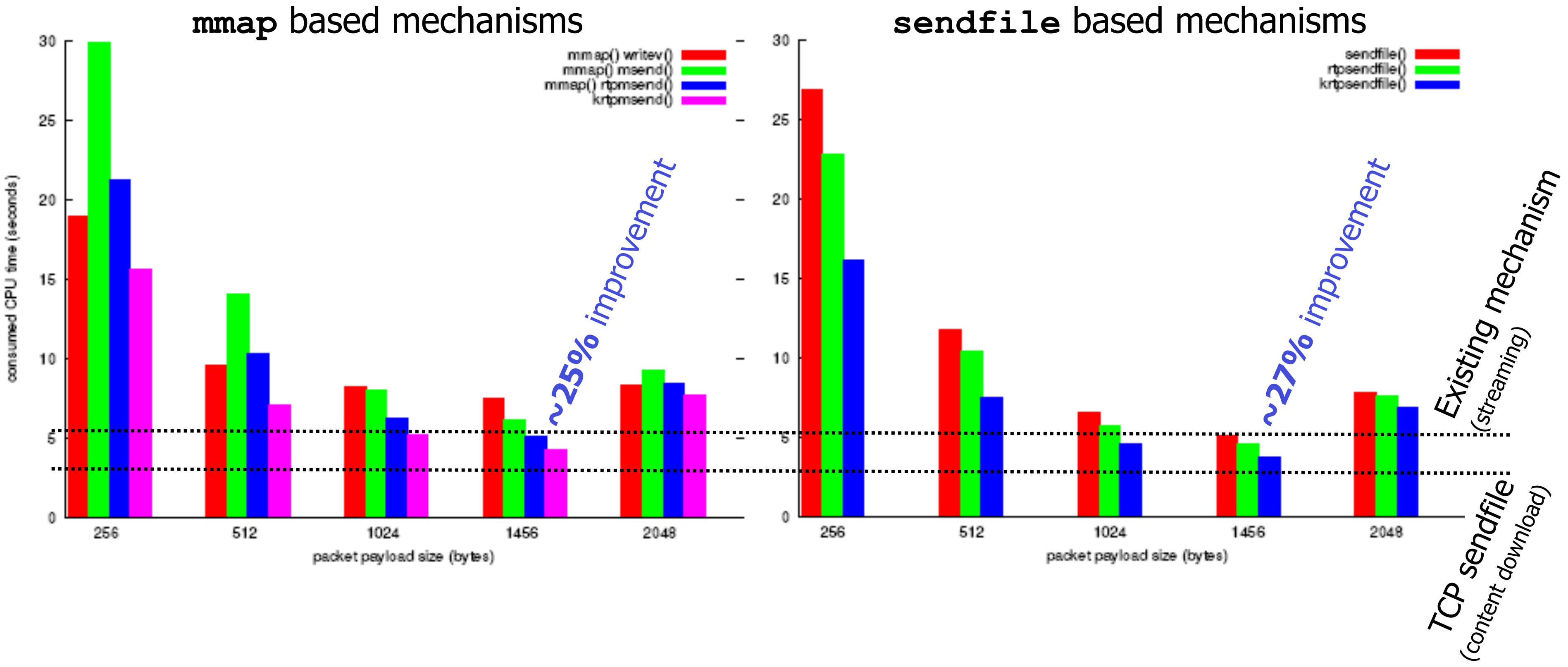


An RTP engine in the kernel adds RTP headers

- ***0*** copy operations ← previous solution: one more copy per packet
- ***1*** system call ← previous solution: one more call per packet

Enhanced Streaming: Results

- Tested streaming of 1 GB file on Linux 2.6
- RTP over UDP



`mmap` based mechanisms

`sendfile` based mechanisms

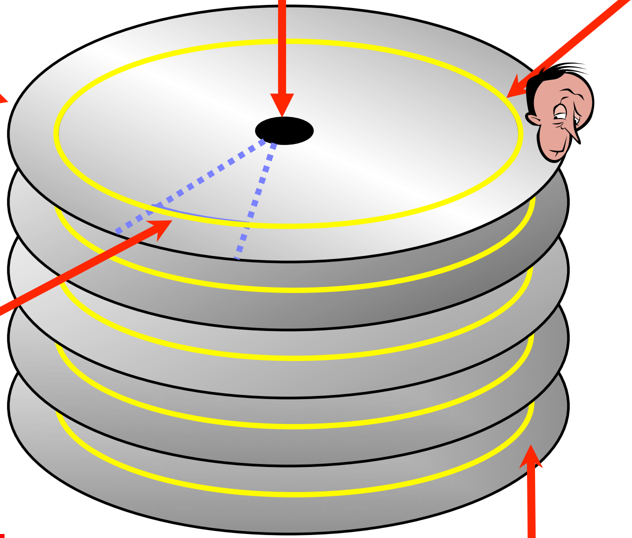
Storage: Disks

Disks

- Two resources of importance
  - storage space
  - I/O bandwidth

- Several approaches to manage data on disks:
  - specific disk scheduling and appropriate buffers
  - optimize data placement
  - replication / striping
  - prefetching
  - combinations of the above

Mechanics of Disks



Platters

circular platters covered with magnetic material to provide nonvolatile storage of bits

Spindle

of which the platters rotate around

Tracks

concentric circles on a single platter

Disk heads

read or alter the magnetism (bits) passing under it. The heads are attached to an arm enabling it to move across the platter surface

Sectors

segment of the *track* circle – usually each contains 512 bytes – separated by non-magnetic gaps. The gaps are often used to identify beginning of a sector

Cylinders

corresponding tracks on the different platters are said to form a cylinder

Disk Specifications

- Some existing (Seagate) disks today:

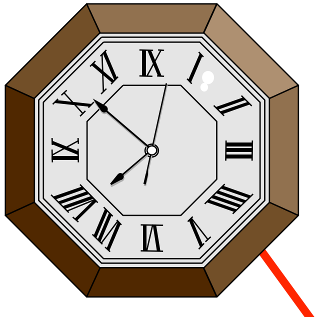
	Barracuda 180	*Cheetah 36*	*Cheetah X15*
Capacity (GB)	181.6	36.4	73.4
Spindle speed (RPM)	7200	10.000	15.000
#cylinders	24.247	9.772	18.479
average seek time (ms)	7.4	5.7	3.6
min (track-to-track) seek (ms)	0.8	0.6	0.2
max (full stroke) seek (ms)	16	12	7
average latency	4.17	3	2
internal transfer rate (Mbps)	282 – 508	520 – 682	609 – 891
disk buffer cache	16 MB	4 MB	8 MB

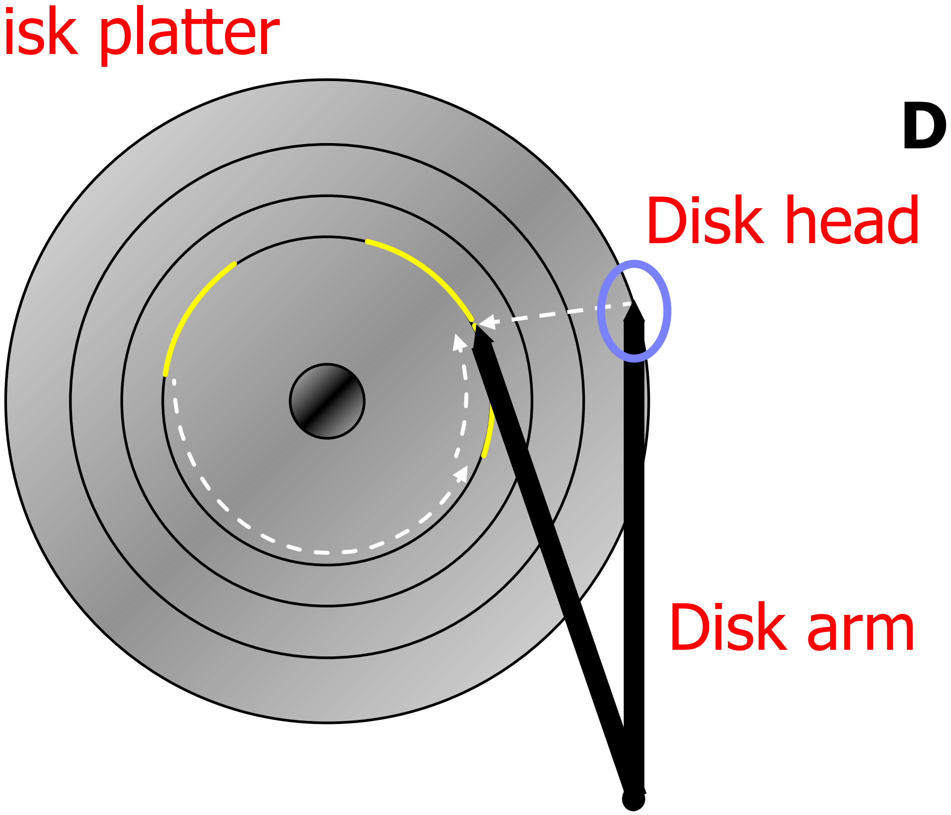
Disk Access Time

- How do we retrieve data from disk?
  - position head over the cylinder (track) on which the block (consisting of one or more sectors) are located
  - read or write the data block as the sectors move under the head when the platters rotate

- The time between the moment issuing a disk request and the time the block is resident in memory is called *disk latency* or *disk access time*

Disk Access Time

I want block X →  → block x in memory



Disk access time =

Seek time

+ Rotational delay

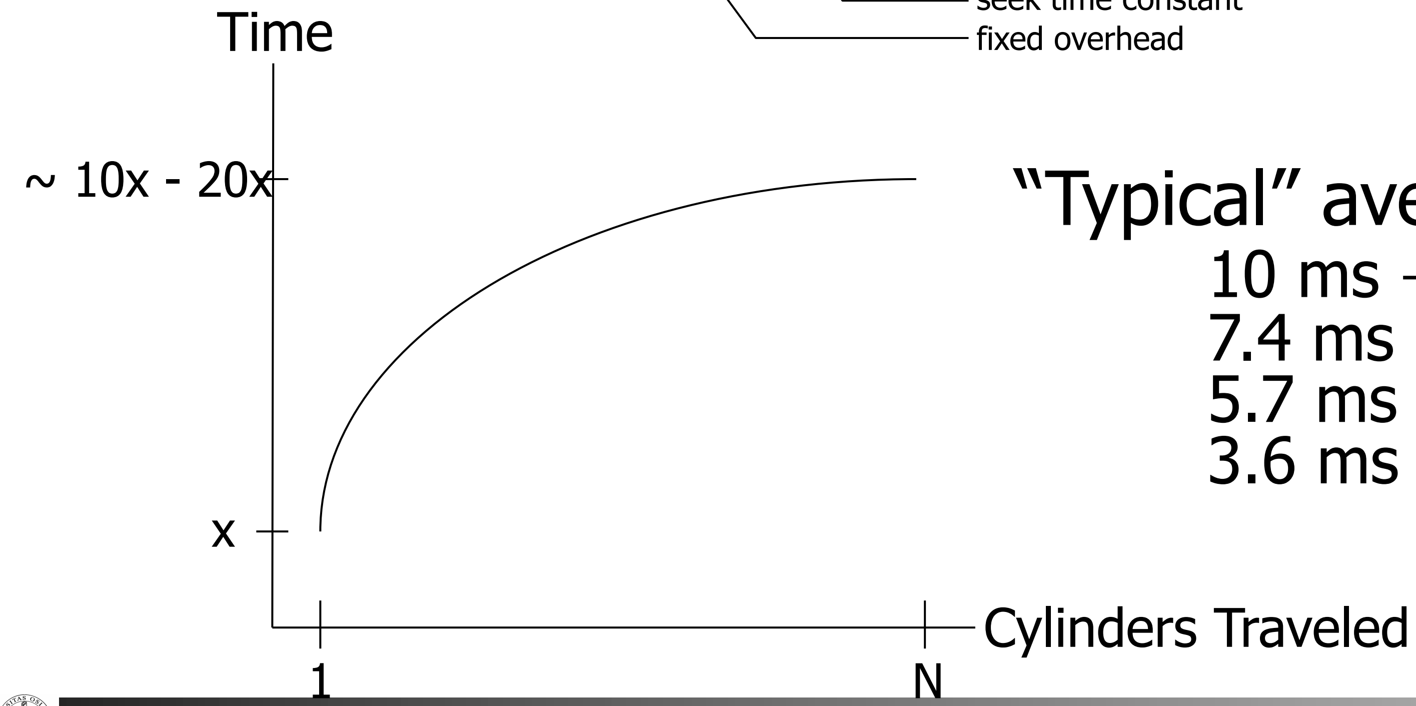
+ Transfer time

+ Other delays

Disk Access Time: Seek Time

- Seek time is the time to position the head
  - the heads require a minimum amount of time to start and stop moving the head
  - some time is used for actually moving the head – roughly proportional to the number of cylinders traveled
  - Time to move head: $\alpha + \beta\sqrt{n}$

number of tracks
seek time constant
fixed overhead

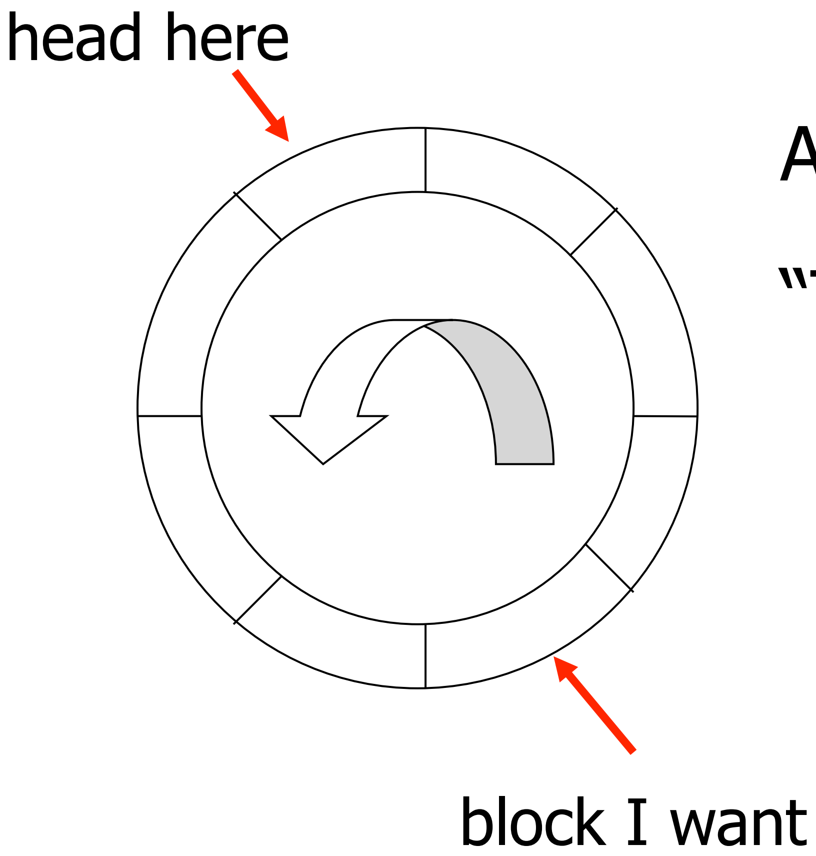


"Typical" average:

10 ms → 40 ms
7.4 ms (Barracuda 180)
5.7 ms (Cheetah 36)
3.6 ms (Cheetah X15)

Disk Access Time: Rotational Delay

- Time for the disk platters to rotate so the first of the required sectors are under the disk head



Average delay is 1/2 revolution

"Typical" average:

8.33 ms	(3.600 RPM)
5.56 ms	(5.400 RPM)
4.17 ms	(7.200 RPM)
3.00 ms	(10.000 RPM)
2.00 ms	(15.000 RPM)

Disk Access Time: Transfer Time

- Time for data to be read by the disk head, i.e., time it takes the sectors of the requested block to rotate under the head
- Transfer rate = $\frac{\text{amount of data per track}}{\text{time per rotation}}$
- Transfer time = amount of data to read / transfer rate

- Example – *Barracuda 180:*
  406 KB per track x 7.200 RPM ≈ 47.58 MB/s
- Example – *Cheetah X15:*
  316 KB per track x 15.000 RPM ≈ 77.15 MB/s

Note:
one might achieve these transfer rates reading continuously on disk, but time must be added for seeks, etc.

- Transfer time is dependent on data density and rotation speed
- If we have to change track, time must also be added for moving the head

Disk Access Time: Other Delays

- There are several other factors which might introduce additional delays:
  - CPU time to issue and process I/O
  - contention for controller
  - contention for bus
  - contention for memory
  - verifying block correctness with checksums (retransmissions)
  - **waiting in scheduling queue**
  - ...

- Typical values: “0”
  (maybe except from waiting in the queue)

Disk Throughput

- How much data can we retrieve per second?
- $$\text{Throughput} = \frac{\text{data size}}{\text{transfer time (including all)}}$$
- Example:

  for each operation we have

- average seek	- average rotational delay
- transfer time	- no gaps, etc.

  - Cheetah X15 (max 77.15 MB/s)
    4 KB blocks → 0.71 MB/s
    64 KB blocks → 11.42 MB/s

  - Barracuda 180 (max 47.58 MB/s)
    4 KB blocks → 0.35 MB/s
    64 KB blocks → 5.53 MB/s

Block Size

- The block size may have large effects on performance
- Example:
  assume random block placement on disk and sequential file access
  - doubling block size will halve the number of disk accesses
    - each access take some more time to transfer the data, but the total transfer time is the same (i.e., more data per request)
    - halve the seek times
    - halve rotational delays are omitted
  - e.g., when increasing block size from 2 KB to 4 KB (no gaps,...) for *Cheetah X15* typically an average of:
    - ☺ 3.6 ms is *saved* for seek time
    - ☺ 2 ms is *saved* in rotational delays
    - ☹ 0.026 ms is *added* per transfer time

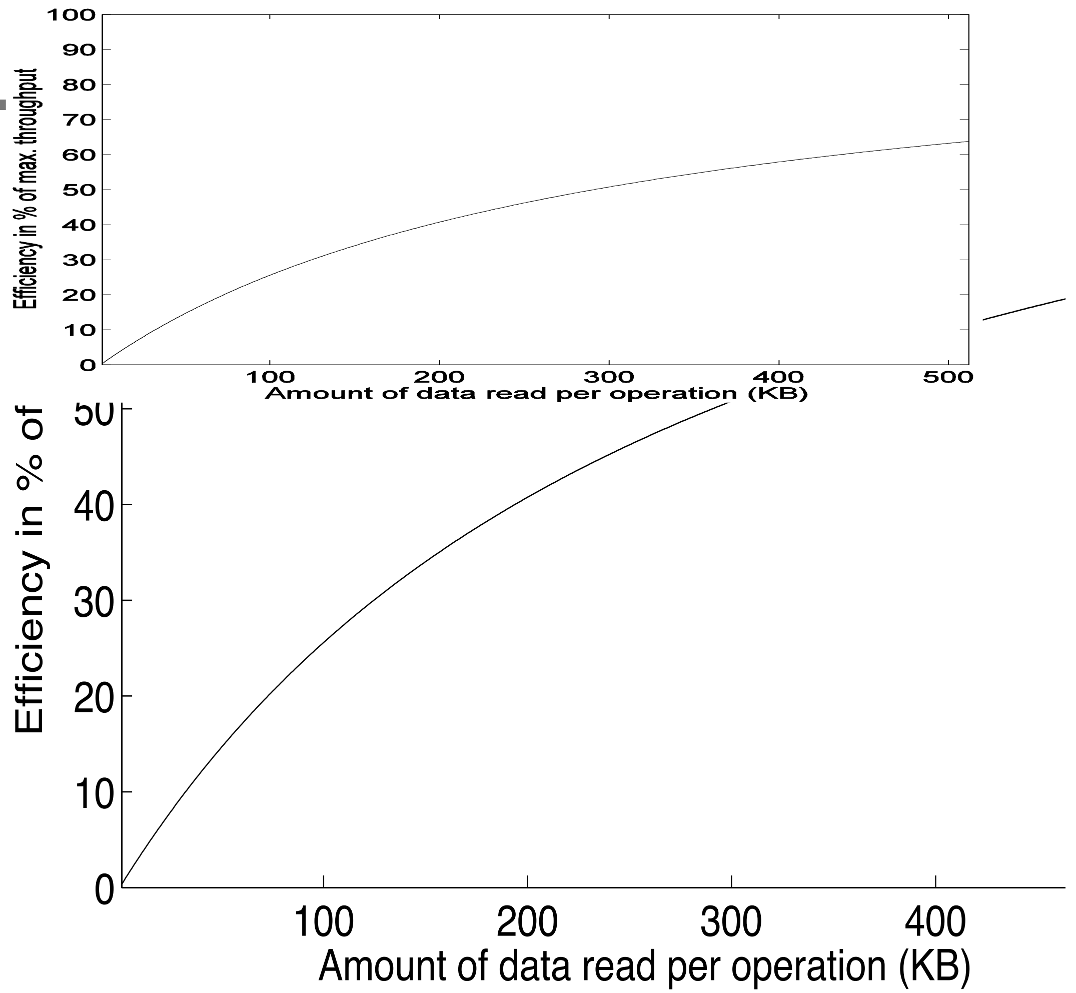
    } saving a total of 5.6 ms when reading 4 KB (49,8 %)
  - increasing from 2 KB to 64 KB saves ~96,4 % when reading 64 KB

Block Size

- Thus, increasing block size can increase performance by reducing seek times and rotational delays
  (figure shows calculation on some older device)

- But, blocks spanning several tracks still introduce latencies…

- … and a large block size is not always best
  - small data elements may occupy only a fraction of the block (fragmentation)



- Which block size to use therefore depends on data size and data reference patterns

- The trend, however, is to use large block sizes as new technologies appear with increased performance – at least in high data rate systems

Writing and Modifying Blocks

- A write operation is analogous to read operations
  - must add time for block allocation
  - a write operation may has to be *verified* – must wait another rotation and then read the block to see if it is the block we wanted to write
  - Total write time ≈ read time (+ time for one rotation)

- Cannot modify a block directly:
  - read block into main memory
  - modify the block
  - write new content back to disk
  - (verify the write operation)
  - Total modify time ≈ read time + time to modify + write time

Disk Controllers

- To manage the different parts of the disk, we use a *disk controller*, which is a small processor capable of:
  - controlling the actuator moving the head to the desired track
  - selecting which platter and surface to use
  - knowing when right sector is under the head
  - transferring data between main memory and disk

- New controllers acts like small computers themselves
  - both disk and controller now has an own buffer reducing disk access time
  - data on damaged disk blocks/sectors are just moved to spare room at the disk – the system above (OS) does not know this, i.e., a block may lie elsewhere than the OS thinks

Efficient Secondary Storage Usage

- Must take into account the use of secondary storage
  - there are large access time gaps, i.e., a disk access will probably dominate the total execution time
  - there may be huge performance improvements if we reduce the number of disk accesses
  - a “slow” algorithm with few disk accesses will probably outperform a “fast” algorithm with many disk accesses

- **Several ways to optimize .....**

– block size	- 4 KB
– file management / data placement	- various
– disk scheduling	- SCAN derivate
– multiple disks	- a specific RAID level
– prefetching	- read-ahead prefetching
– memory caching /replacement algorithms	- LRU variant
– …	

Disk Scheduling

Disk Scheduling – I

- **Seek time is the dominant factor of total disk I/O time**
- Let operating system or disk controller choose which request to serve next depending on the head's current position and requested block's position on disk (disk scheduling)
- Note that disk scheduling ≠ CPU scheduling
  - a mechanical device – hard to determine (accurate) access times
  - disk accesses cannot be preempted – runs until it finishes
  - disk I/O often the main performance bottleneck
- General goals
  - short response time
  - high overall throughput
  - fairness (equal probability for all blocks to be accessed in the same time)
- Tradeoff: seek and rotational delay vs. maximum response time

Disk Scheduling – II

- Several traditional (performance oriented) algorithms
  - First-Come-First-Serve (FCFS)
  - Shortest Seek Time First (SSTF)
  - SCAN (and variations)
  - Look (and variations)
  - …

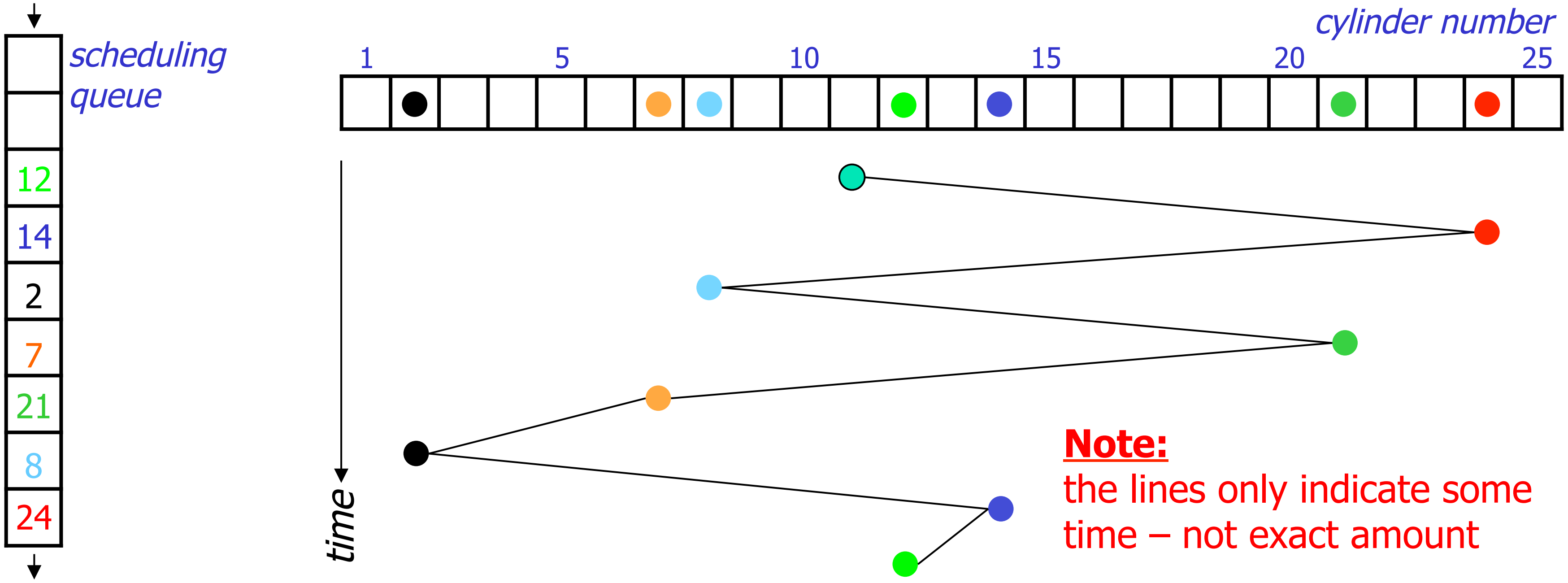
First–Come–First–Serve (FCFS)

FCFS serves the first arriving request first:

- Long seeks
- "Short" average response time

incoming requests (in order of arrival):

12 14 2 7 21 8 24



Note:
the lines only indicate some time – not exact amount

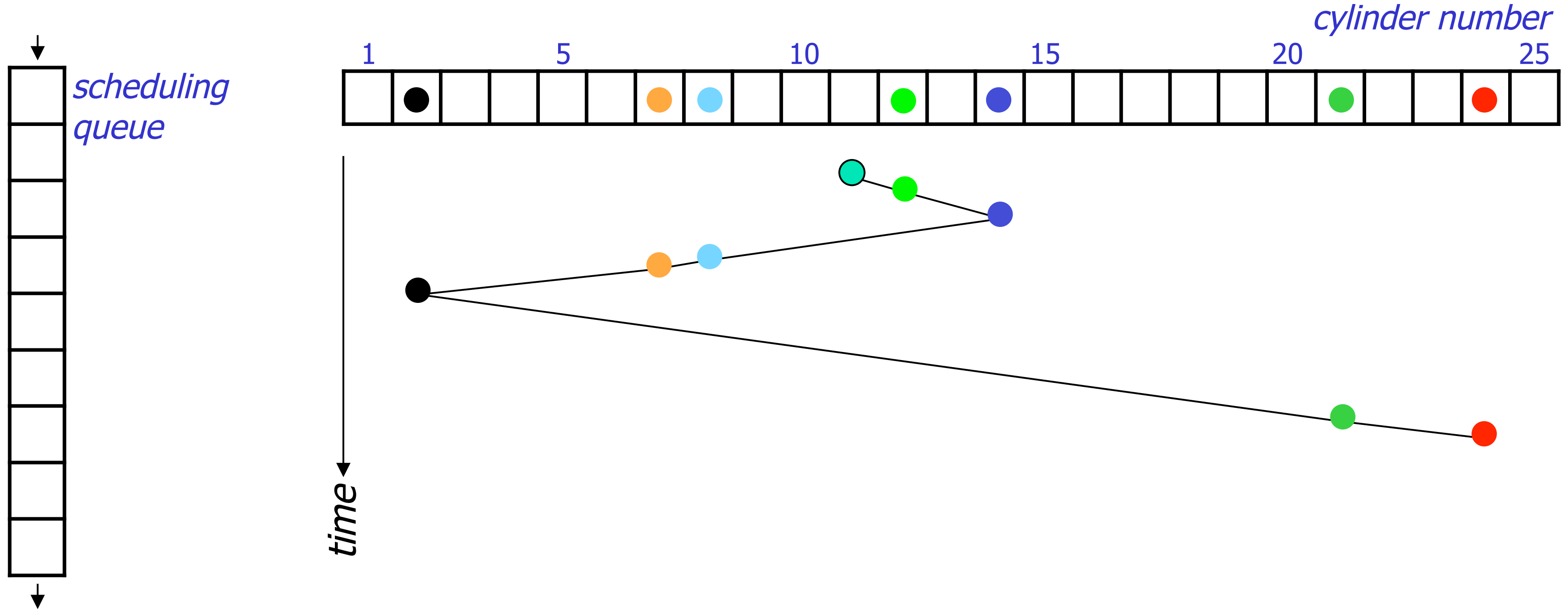
Shortest Seek Time First (SSTF)

SSTF serves closest request first:

- short seek times
- longer maximum seek times – **may even lead to starvation**

incoming requests (in order of arrival):

12 14 2 7 21 8 24



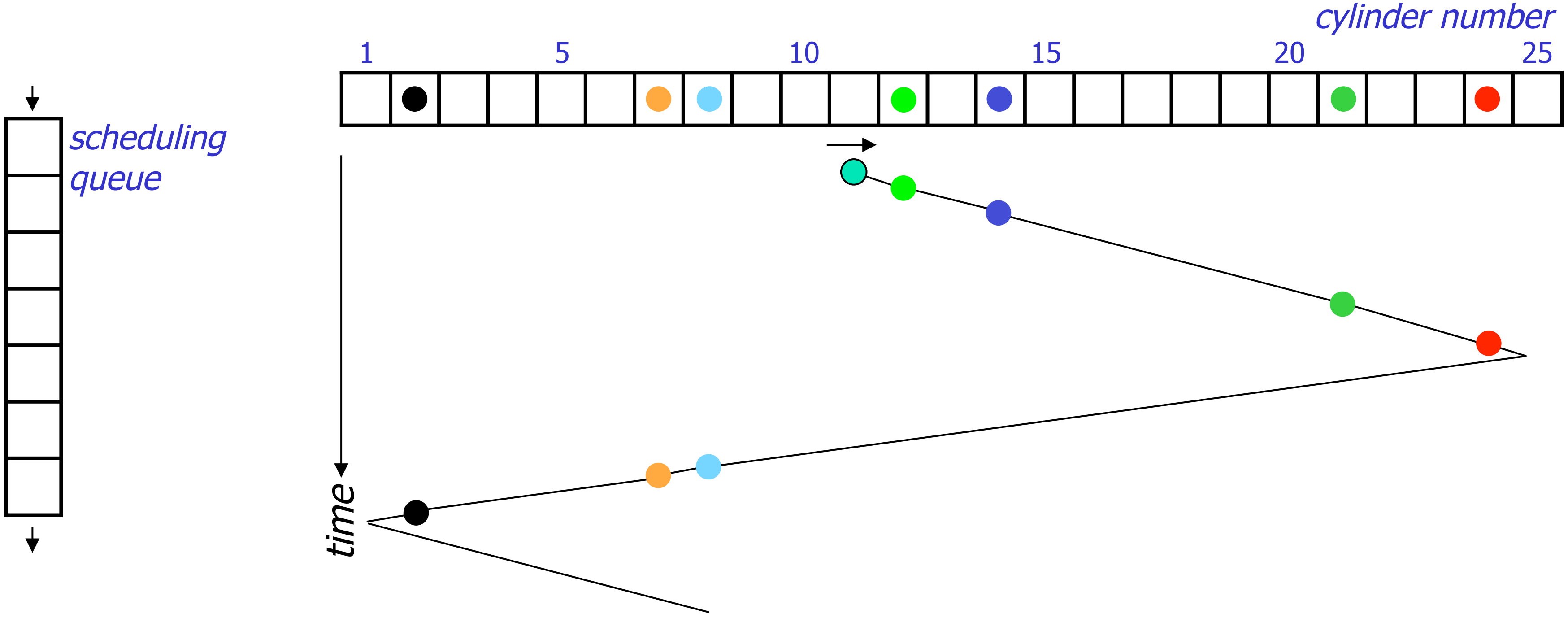
SCAN

SCAN (elevator) moves head edge to edge and serves requests on the way:

- bi-directional
- compromise between response time and seek time optimizations

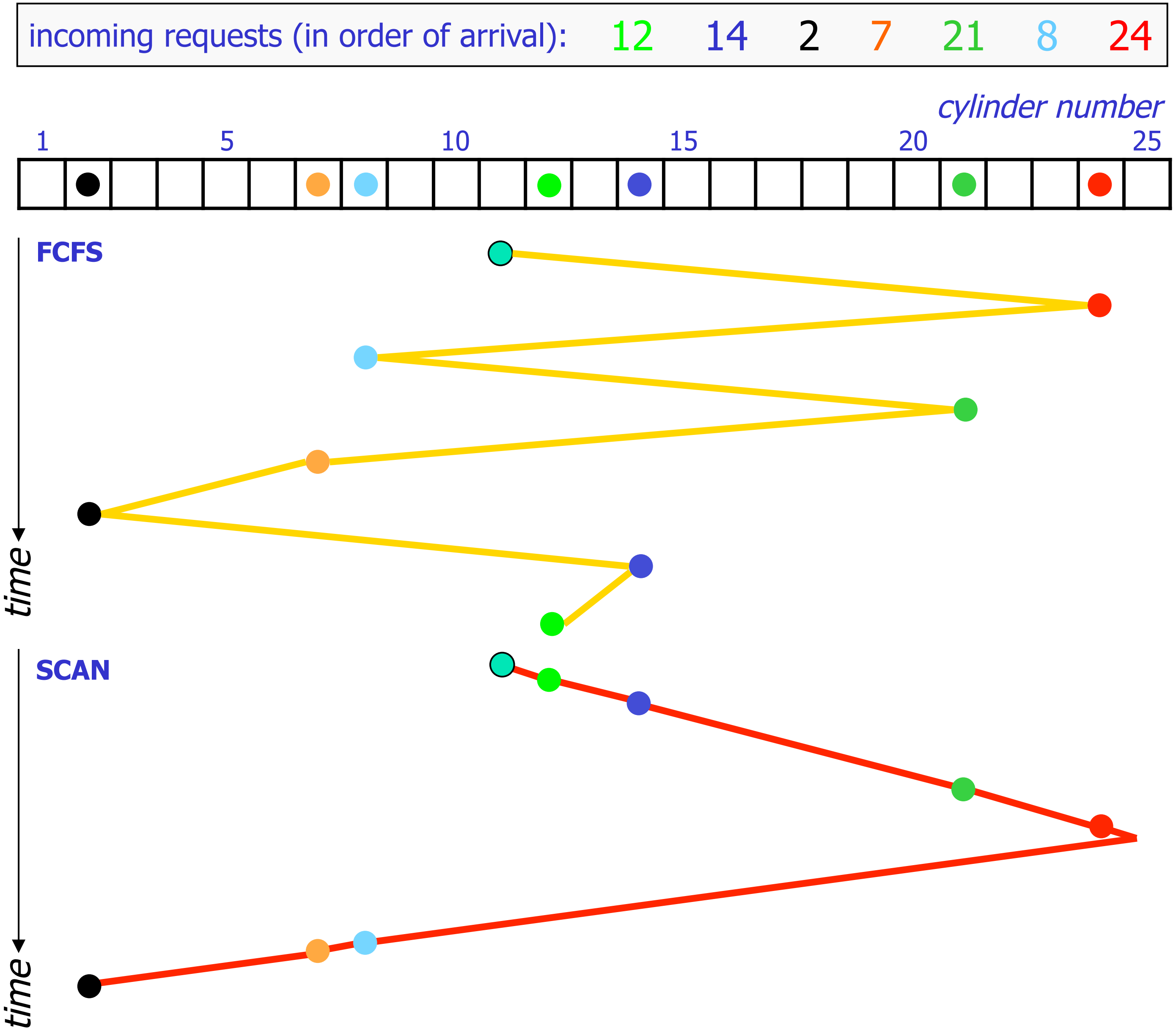
incoming requests (in order of arrival):

12 14 2 7 21 8 24



SCAN vs. FCFS

- Disk scheduling makes a difference!
- In this case, we see that SCAN requires much less head movement compared to FCFS (37 vs. 75 tracks/cylinders)



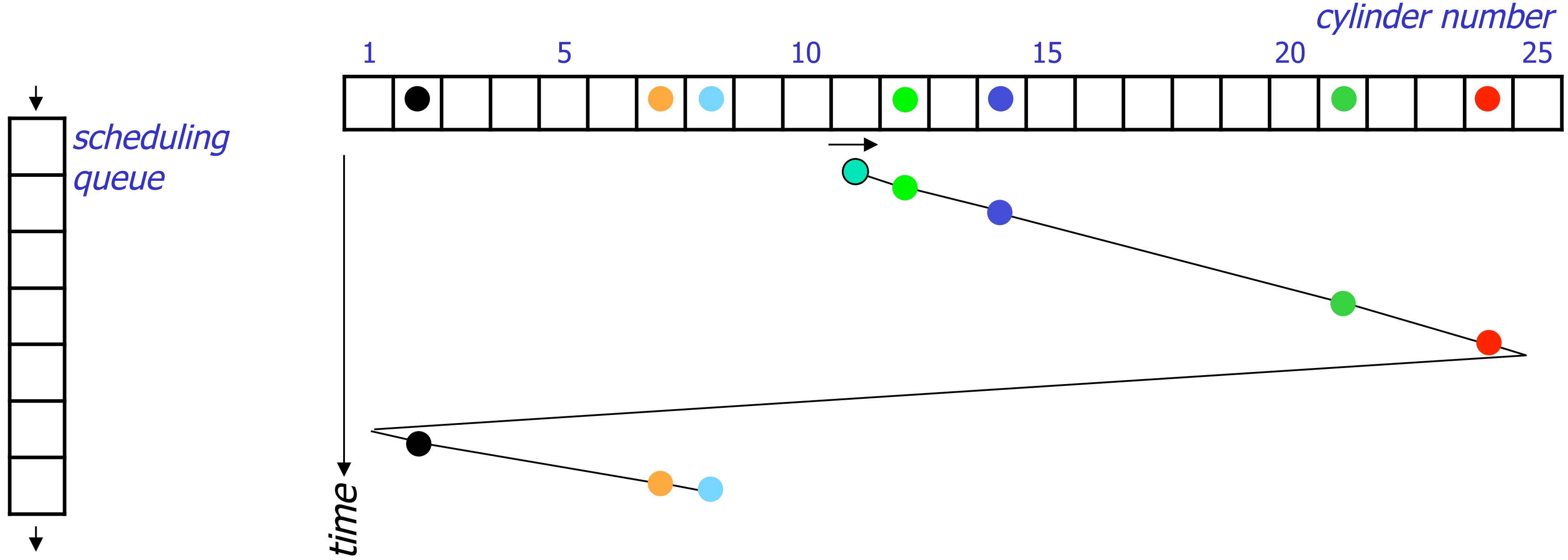
C–SCAN

Circular-SCAN moves head from edge to edge

- serves requests on one way – uni-directional
- improves response time (fairness)

incoming requests (in order of arrival):

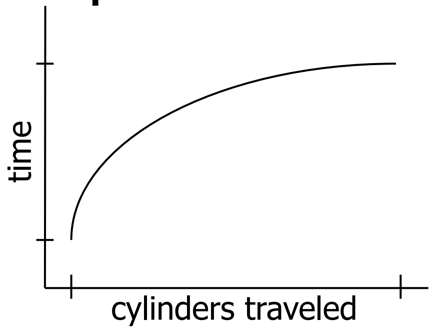
12 14 2 7 21 8 24

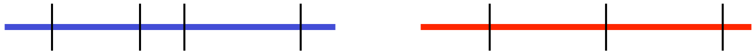
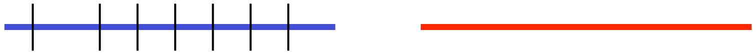


SCAN vs. C–SCAN

- Why is C-SCAN in average better in reality than SCAN when both service the same number of requests in two passes?
  - modern disks must accelerate (speed up and down) when seeking
  - head movement formula: $\alpha + \beta\sqrt{n}$

number of cylinders
seek time constant
fixed overhead



SCAN	C-SCAN
bi-directional 	uni-directional 
requests: n avg. dist: 2x total cost: $\underline{n \times \sqrt{2x}} = n \times \sqrt{2} \times \sqrt{x}$	requests: n avg. dist: x total cost: $\underline{\sqrt{n \times x}} + \underline{n \times \sqrt{x}} = (\sqrt{n} + n) \times \sqrt{x}$

if n is large:

$$n\sqrt{2} > n + \sqrt{n}$$

$$\Updownarrow$$

$$2 \times n^2 > n + 2n\sqrt{n} + n^2$$

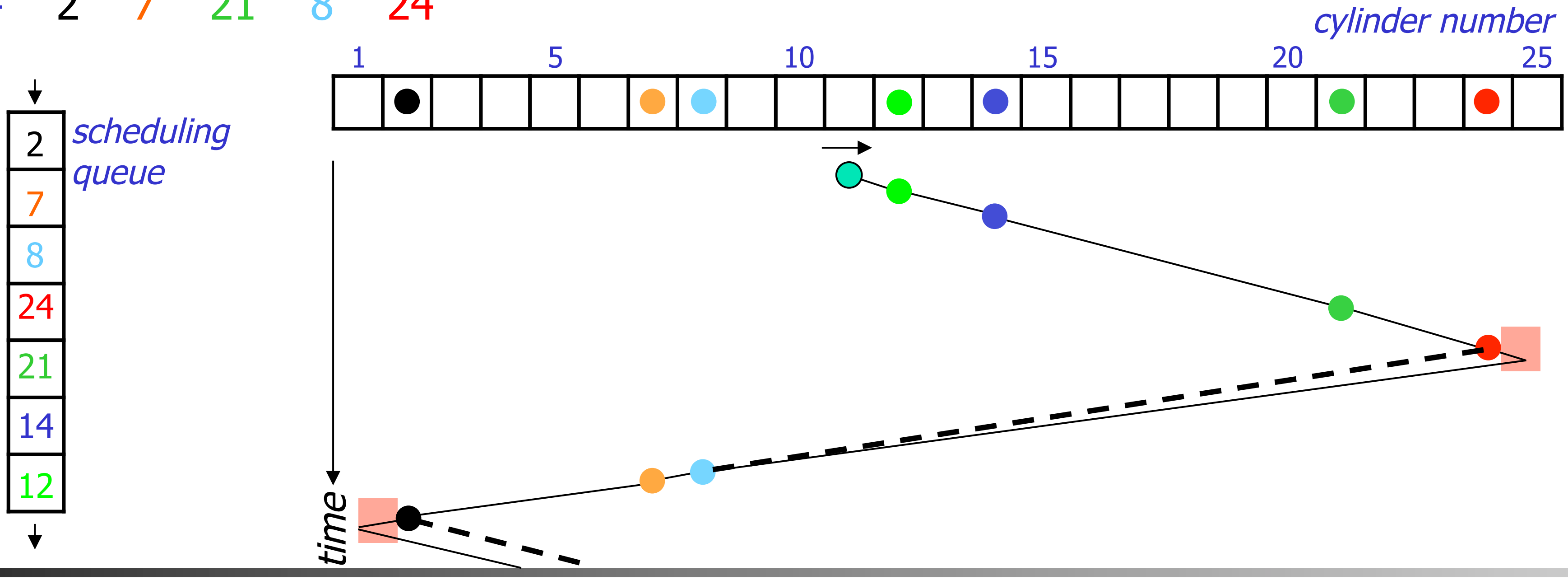
LOOK and C–LOOK

LOOK (C-LOOK) is a variation of SCAN (C-SCAN):

- same schedule as SCAN
- does not run to the edges
- stops and returns at outer- and innermost request
- increased efficiency
- SCAN vs. LOOK example:

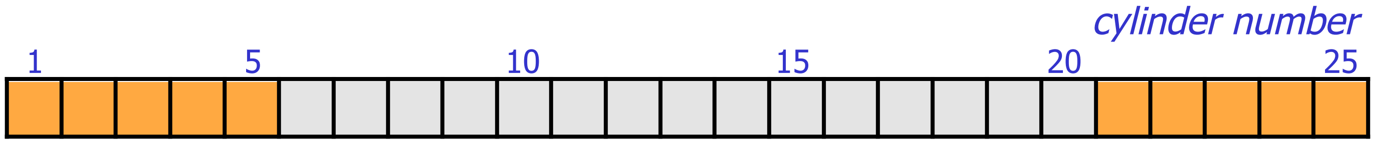
incoming requests (in order of arrival):

12 14 2 7 21 8 24



V–SCAN(R)

- V-SCAN(R) combines SCAN (or LOOK) and SSTF
  - define an **R**-sized unidirectional SCAN window, i.e., C-SCAN, and use SSTF outside the window
  - Example: V-SCAN(0.6)
    - makes a C-SCAN window over 60 % of the cylinders
    - uses SSTF for requests outside the window



  - V-SCAN(0.0) equivalent with SSTF
  - V-SCAN(1.0) equivalent with C-SCAN
  - V-SCAN(0.2) is supposed to be an appropriate configuration

END of This WEEK!!

NOW is the time to stop!!!!

… and start next time!

What About Time-Dependent Media?

- Suitability of classical algorithms
  - minimal disk arm movement (short seek times)
  - but, no provision of time or deadlines
  - ⇨ generally not suitable

- For example, a media server requires
  - support for both periodic and aperiodic
    - never miss deadline due to aperiodic requests
    - aperiodic requests must not starve
  - support multiple streams
  - buffer space and efficiency tradeoff?

Real-Time Disk Scheduling

- Traditional algorithms have no provision of time or deadlines

⇨ Real–time algorithms targeted for real–time applications with deadlines

- Several proposed algorithms
  - earliest deadline first (EDF)
  - SCAN-EDF
  - shortest seek and earliest deadline by ordering/value (SSEDO / SSEDV)
  - priority SCAN (PSCAN)
  - ...

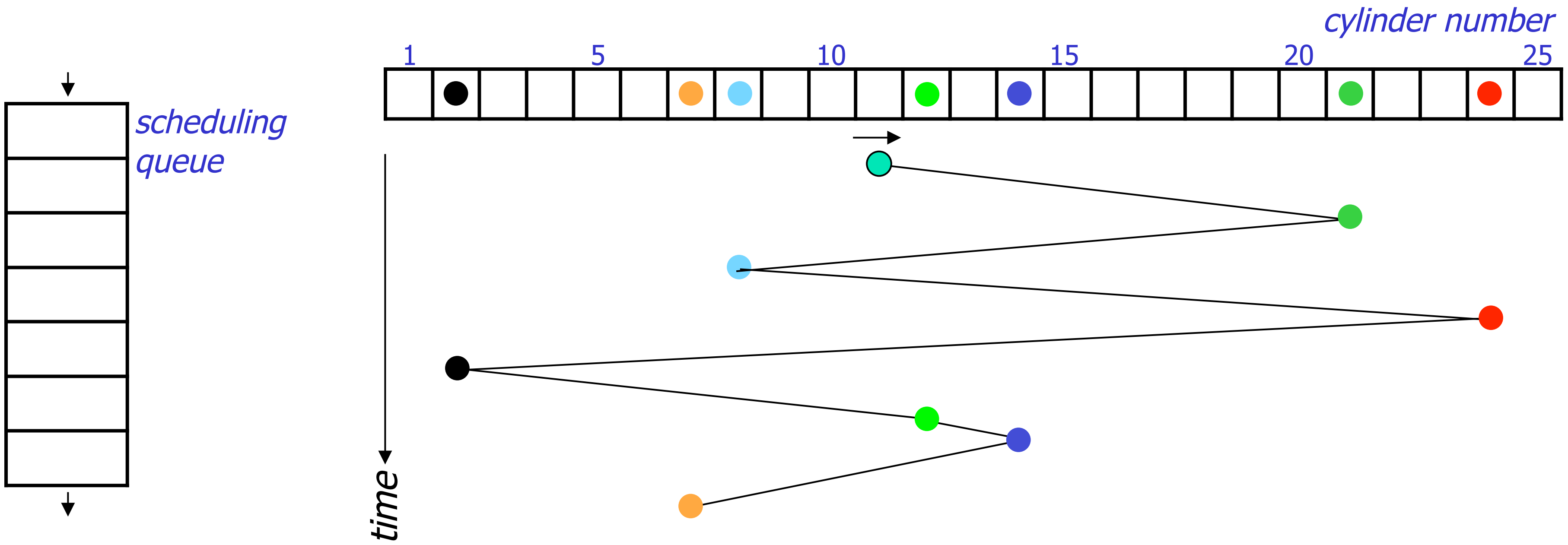
Earliest Deadline First (EDF)

EDF serves the request with nearest deadline first

- non-preemptive (i.e., an arriving request with a shorter deadline must wait)
- excessive seeks → poor throughput

incoming requests (`<block, deadline>`, in order of arrival):

12,5 14,6 2,4 7,7 21,1 8,2 24,3



SCAN–EDF

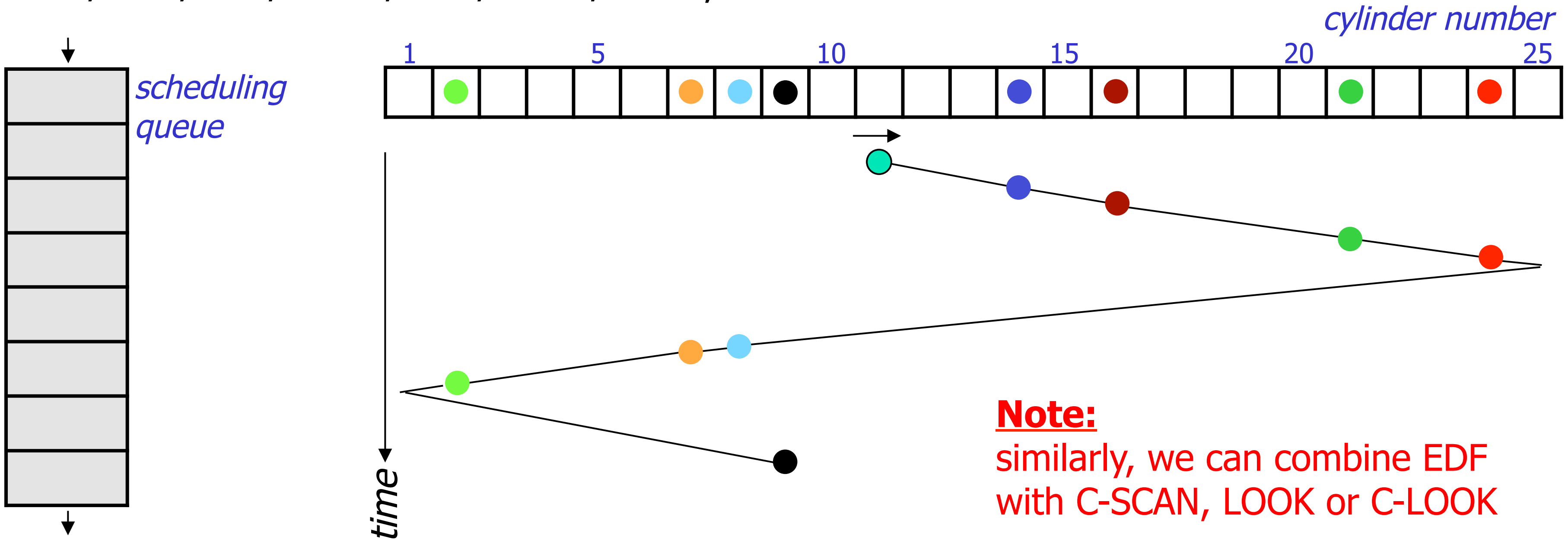
SCAN-EDF combines SCAN and EDF:

- the real-time aspects of EDF
- seek optimizations of SCAN
- especially useful if the end of the period is the deadline (some equal deadlines)

- algorithm:
  - – serve requests with earlier deadline first (EDF)
  - – sort requests with same deadline after track location (SCAN)

incoming requests (`<block, deadline>`, in order of arrival):

2,3 14,1 9,3 7,2 21,1 8,2 24,2 16,1

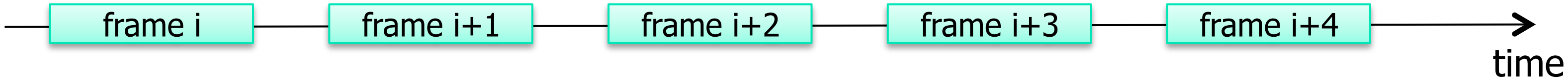


Note:
similarly, we can combine EDF with C-SCAN, LOOK or C-LOOK

Stream Oriented Disk Scheduling

- Streams often have soft deadlines and tolerate some slack due to buffering, i.e., pure real-time scheduling is inefficient and unnecessary

➡ Stream oriented algorithms targeted for streaming continuous media data requiring periodic access, e.g., a frame every 40ms:



- Several algorithms proposed:
  - group sweep scheduling (GSS)
  - mixed disk scheduling strategy
  - contiguous media file system (CMFS)
  - lottery scheduling
  - stride scheduling
  - batched SCAN (BSCAN)
  - greedy-but-safe EDF (GS_EDF)
  - bubble up
  - …

Group Sweep Scheduling (GSS)

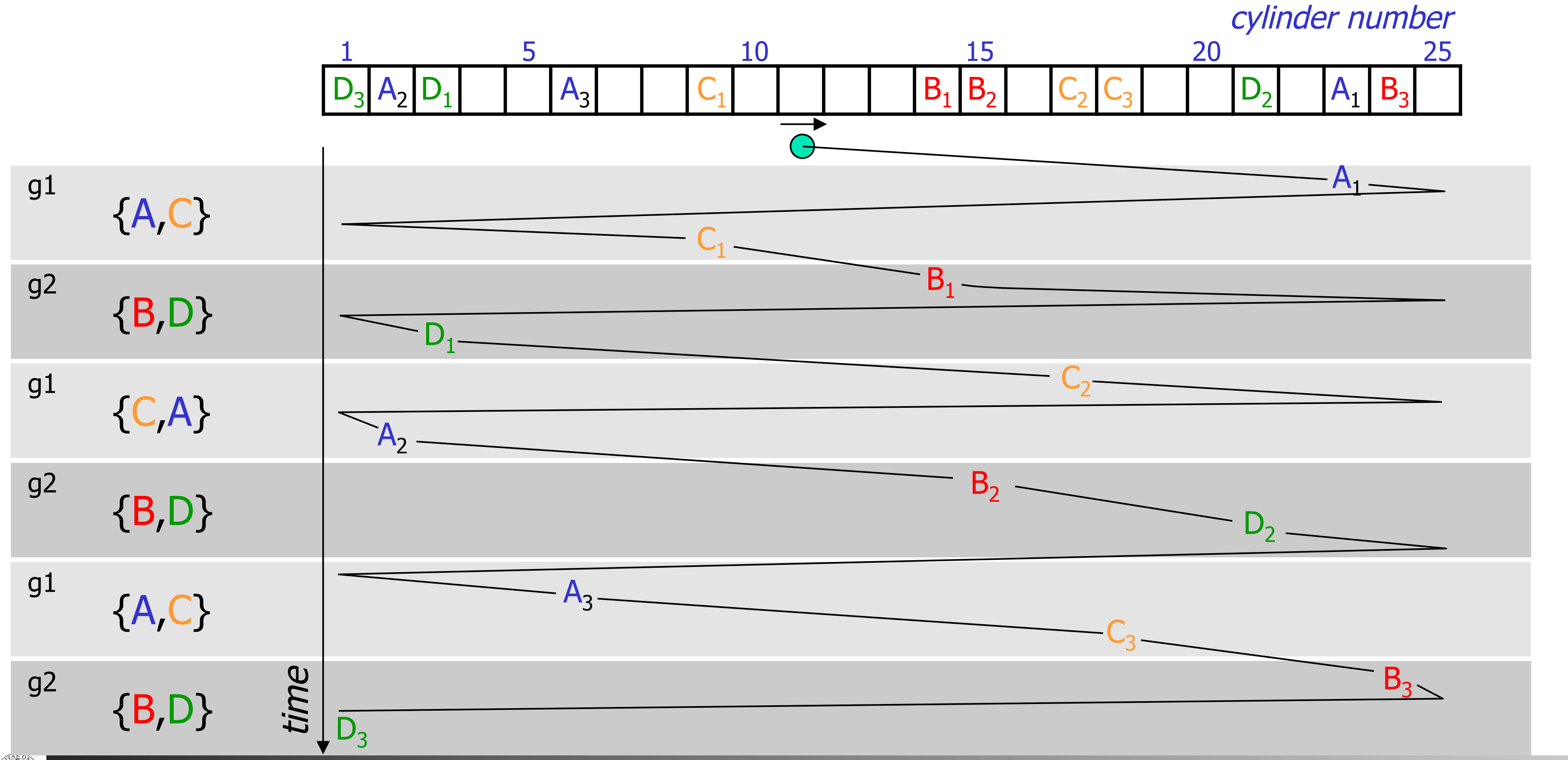
GSS combines Round-Robin (RR) and SCAN

- requests are serviced in rounds (cycles)
- principle:
  - divide S active streams into G groups
  - service the G groups in RR order
  - service each stream in a group in C-SCAN order
  - playout can start at the end of the group
- special cases:
  - G = S: RR scheduling
  - G = 1: SCAN scheduling
- tradeoff between buffer space and disk arm movement
  - try different values for G giving minimum buffer requirement while remaining efficient enough to reach the deadlines
  - a large G → smaller groups, more arm movements
  - a small G → larger groups, less arm movements

Group Sweep Scheduling (GSS)

GSS example: streams A, B, C and D → **g1**:{A,C} and **g2**:{B,D}

- RR group schedule
- C-SCAN block schedule within a group



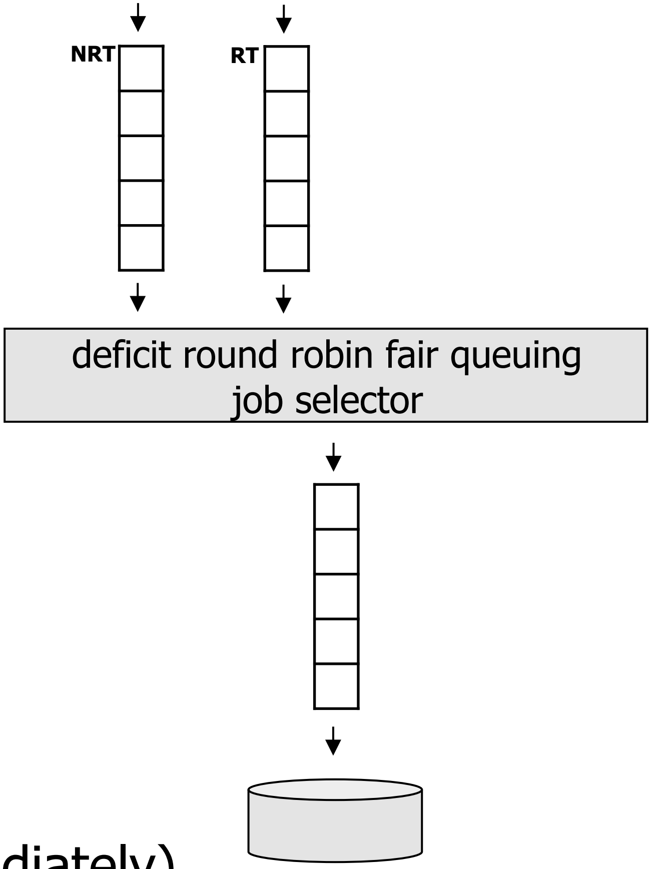
Mixed Media Oriented Disk Scheduling

- Applications may require both RT and NRT data – desirable to have all on same disk

- Several algorithms proposed:
  - Felini's disk scheduler
  - Delta L
  - Fair mixed-media scheduling (FAMISH)
  - MARS scheduler
  - Cello
  - Adaptive disk scheduler for mixed media workloads (APEX)
  - …

MARS Disk Scheduler

- Massively-parallel And Real-time Storage (MARS) scheduler supports mixed media on a single system
  - a two-level scheduling
  - round-based
  - **top-level:**
    1 NRT queue and n (1) RT queue
    (SCAN, but "future" GSS, SCAN-EDF, or…)
  - use deficit RR fair queuing to assign quantums to each queue per round – divides total bandwidth among queues
  - **bottom-level:**
    select requests from queues according to quantums, use SCAN order
  - work-conserving
    (variable round times, new round starts immediately)



Cello and APEX

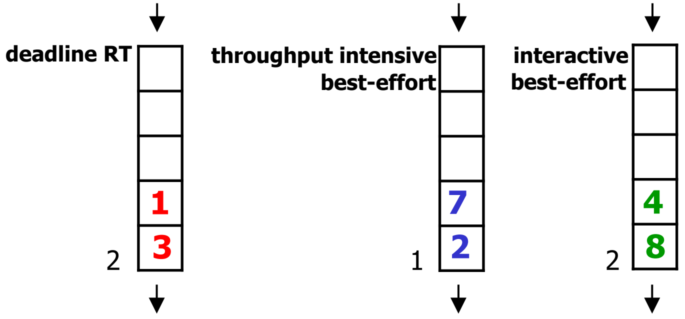
- Cello and APEX are similar to MARS, but slightly different in bandwidth allocation and work conservation

  - Cello has
    - three queues: deadline (EDF), throughput intensive best effort (FCFS), interactive best effort (FCFS)
    - *static* proportional allocation scheme for bandwidth
    - FCFS ordering of queue requests in lower-level queue
    - partially work-conserving:
      extra requests might be added at the end of the class independent scheduler, but constant rounds

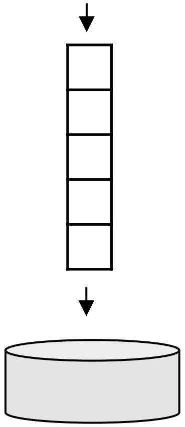
  - APEX
    - n queues
    - uses token bucket for traffic shaping (bandwidth allocation)
    - work-conserving:
      adds extra requests if possible to a batch & starts extra batch between ordinary batches

Cello

- Cello is part of the Symphony FS supporting mixed media
  - two-level scheduling
  - round-based
  - **top-level:** n (3) service classes (queues)
    - deadline (= end-of-round) real-time (EDF)
    - throughput intensive best effort (FCFS)
    - interactive best effort (FCFS)
  - divides total bandwidth among queues according to a *static* proportional allocation scheme (equal to MARS' job selector)
  - **bottom-level:** class independent scheduler (FCFS)
    - select requests from queues according to BW share
    - sort requests from each queue in SCAN order when transferred
  - partially work-conserving (extra requests might be added at the end of the class independent scheduler if space, *but* constant rounds)

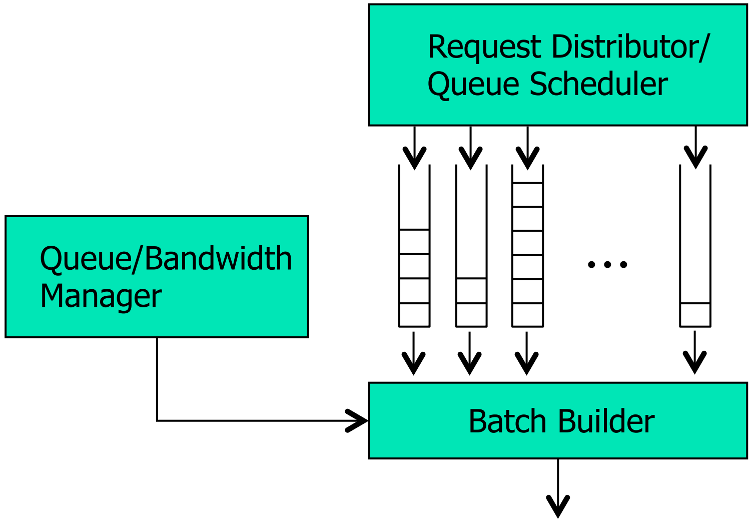


sort each queue in SCAN order when transferred



Adaptive Disk Scheduler for Mixed Media Workloads

- APEX is another mixed media scheduler
  - two-level, round-based scheduler similar to Cello and MARS
  - uses token bucket for traffic shaping (bandwidth allocation)
  - the batch builder select requests in FCFS order from the queues based on number of tokens – each queue must sort according to deadline (or another strategy)
  - work-conserving
    - adds extra requests if possible to a batch
    - starts extra batch between ordinary batches



APEX, Cello and C–LOOK Comparison

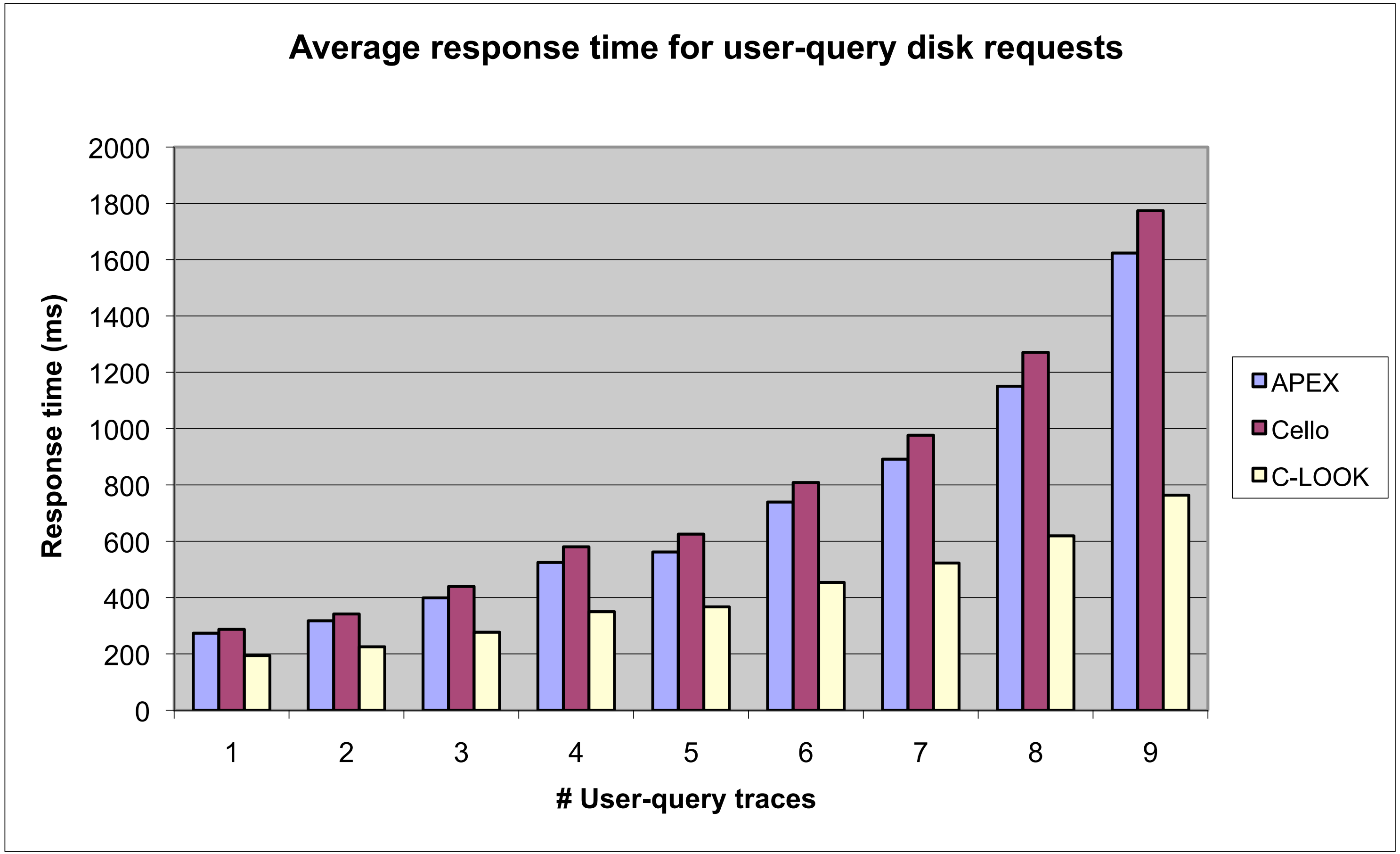
- Results from Ketil Lund (2002)

- Configuration:
  - Atlas Quantum 10K
  - data placement: random
  - round time: 1 second
  - block size: 64KB

- 6 video playbacks and up to 9 user queries
  - Video data disk requests are assigned to a real-time queue
  - User-query disk requests to a best-effort queue

  - Bandwidth is shared 50/50 between real-time and best-effort queues

APEX, Chello and C–LOOK Comparison



Deadline violations (video)

	1	2	3	4	5	6	7	8	9
APEX	0	0	0	0	0	0	0	0	0
Cello	0	0	0	0	0	0	0	0	0
C-LOOK	**0**	**18**	**90**	**288**	**404**	**811**	**1271**	**2059**	**3266**

Schedulers today (Linux)?

- NOOP
  - FCFS with request merging
- Deadline I/O
  - C-SCAN based
  - 4 queues: elevator/deadline for read/write
- Anticipatory
  - same queues as in Deadline I/O
  - delays decisions to be able to merge more requests (e.g., a streaming scenario)
- Completely Fair Scheduler (CFQ)
  - 1 queue per process (periodic access, but priode depends on load)
  - gives time slices and ordering according to priority level (real-time, best-effort, idle)
  - work-conserving

User space vs. kernel space scheduling

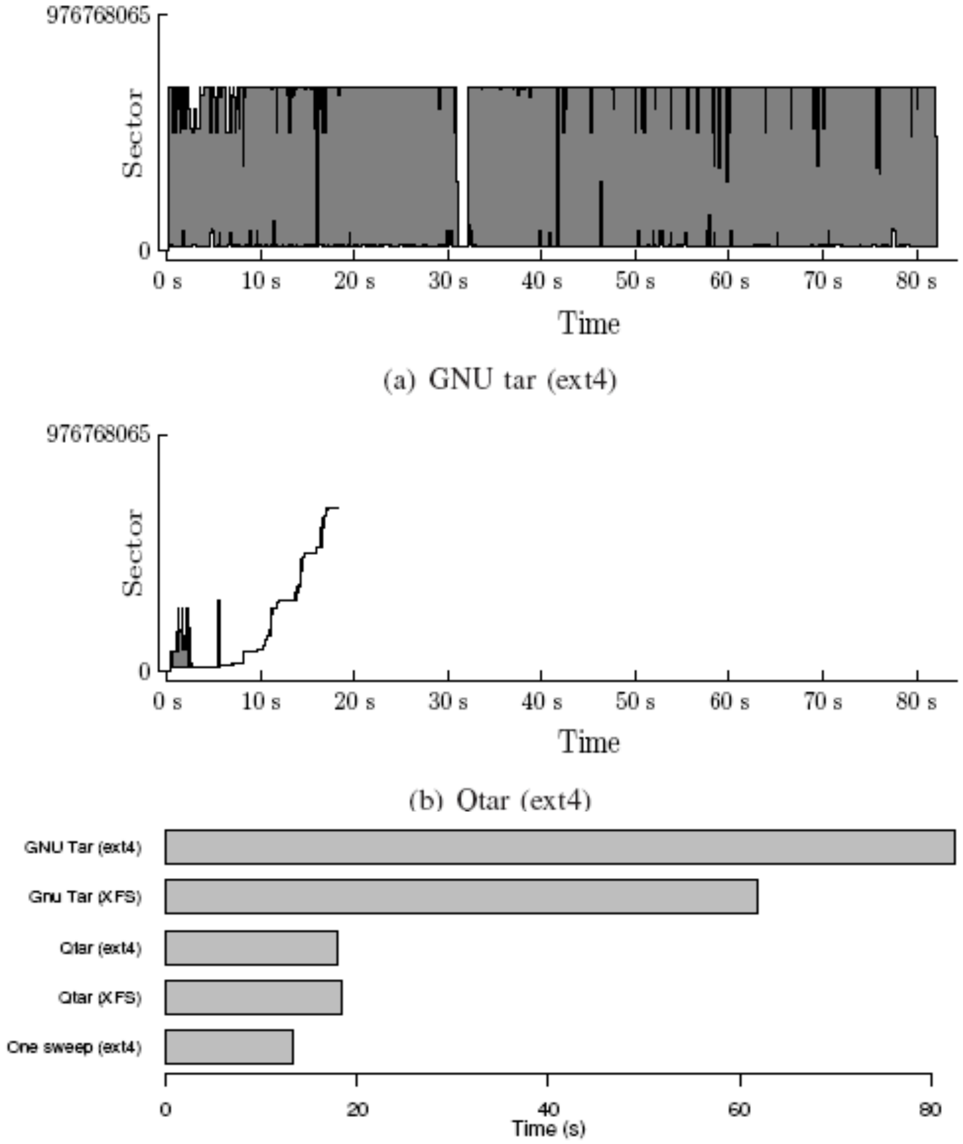
- **GNU/BSD Tar** vs. **QTAR**

User space vs. kernel space scheduling

- Do all applications fully benefit from kernel space scheduling?
- ⇘ File tree traversals
  - *processing one file after another*
  - `tar`, `zip`, …
  - recursive copy (`cp -r`)
  - search (`find`)
  - …

- Only application knows access pattern
  - use `ioctl FIEMAP` (`FIBMAP`) to retrieve extent locations
  - sort in user space
  - send I/O request according to sorted list

⇨ **GNU/BSD Tar** vs. **QTAR**



(a) GNU tar (ext4)

(b) Qtar (ext4)

Figure 3. Time to `tar` the Linux source tree (22500 files)

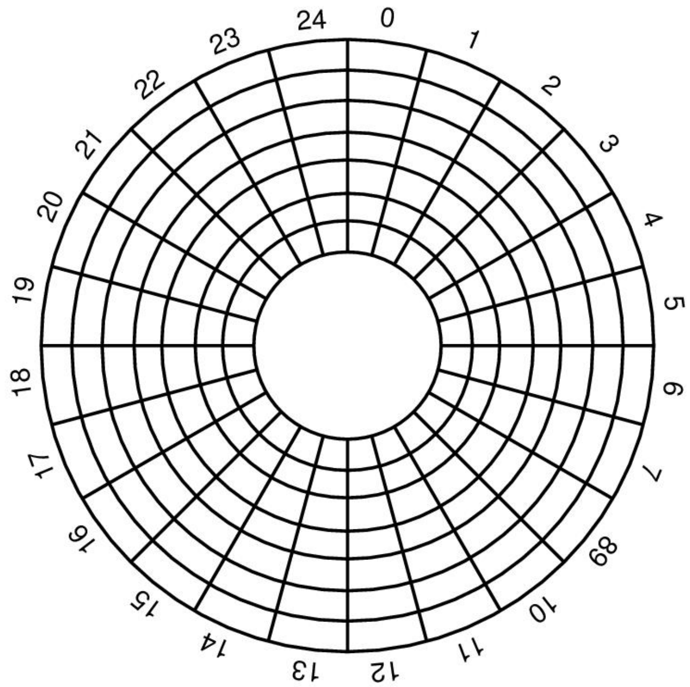
Data Placement on Disk

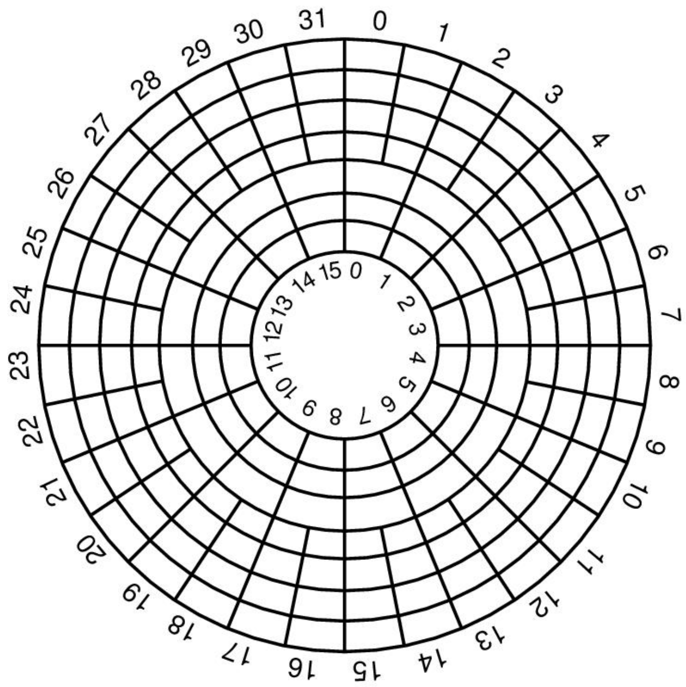
Data Placement on Disk

- Disk blocks can be assigned to files many ways, and several schemes are designed for
  - optimized latency
  - increased throughput

  ↳ access pattern dependent

Disk Layout

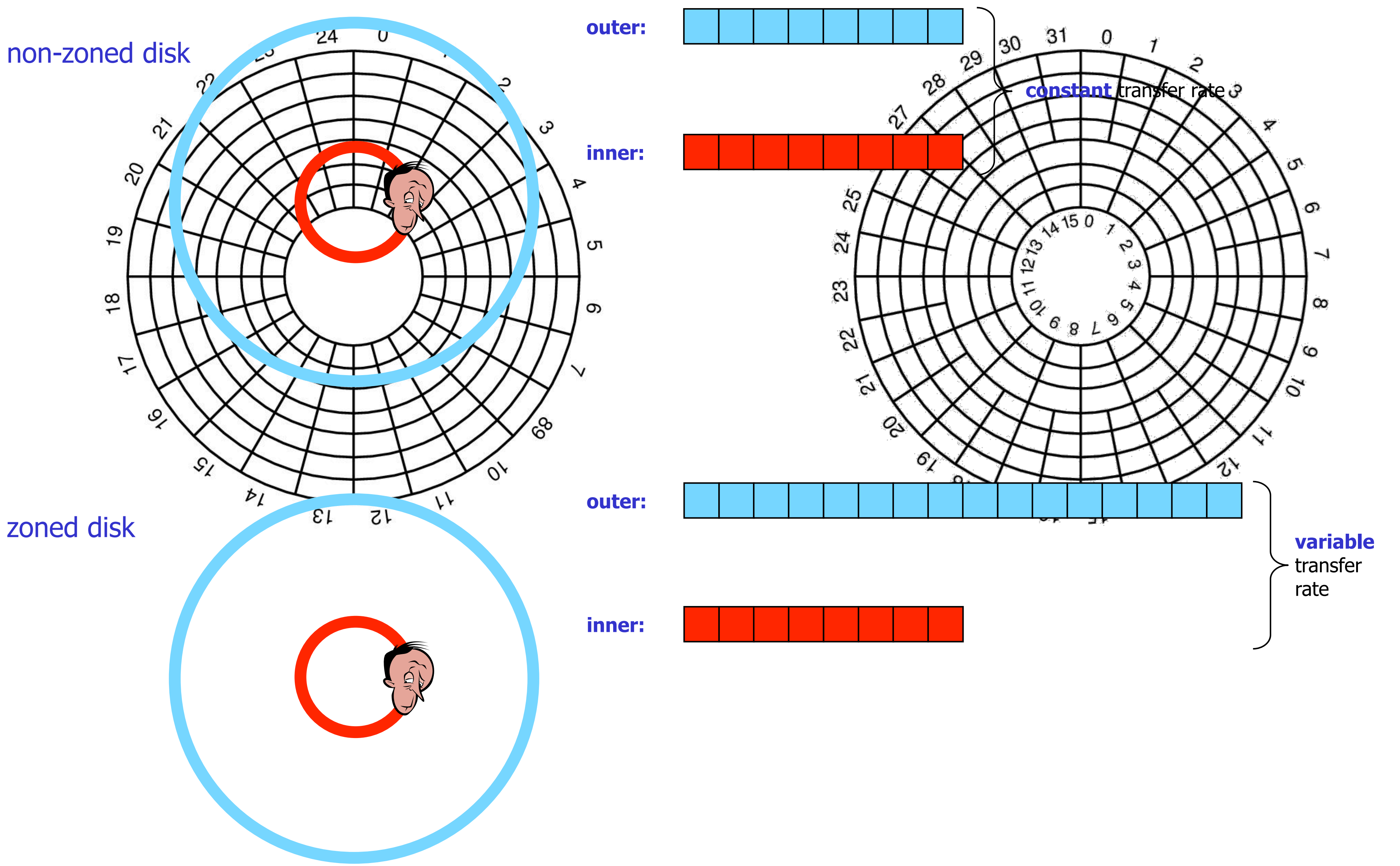




- Constant angular velocity (CAV) disks
  - equal amount of data in each track (and thus constant transfer time)
  - constant rotation speed

- Zoned CAV disks
  - zones are ranges of tracks
  - typical few zones
  - the different zones have
    - different amount of data
    - different bandwidth
    - i.e., more better on outer tracks

Disk Layout



non-zoned disk

outer:

inner:

constant transfer rate

zoned disk

outer:

inner:

variable transfer rate

Disk Layout

- Cheetah X15.3 is a zoned CAV disk:

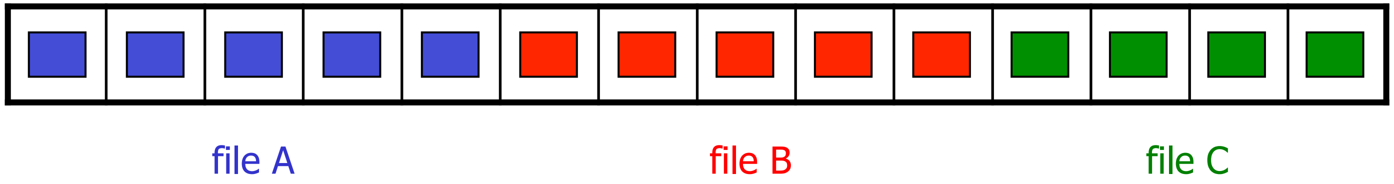
Zone	Cylinders per Zone	Sectors per Track	Zone Transfer Rate (MBps)	Sectors per Zone	Efficiency	Formatted Capacity (MB)
1	**3544**	**672**	**890,98**	19014912	77,2%	**9735,635**
2	3382	652	878,43	17604000	76,0%	9013,248
3	3079	624	835,76	15340416	76,5%	7854,293
4	2939	595	801,88	13961080	76,0%	7148,073
5	2805	576	755,29	12897792	78,1%	6603,669
6	2676	537	728,47	11474616	75,5%	5875,003
7	2554	512	687,05	10440704	76,3%	5345,641
8	2437	480	649,41	9338880	75,7%	4781,506
9	2325	466	632,47	8648960	75,5%	4428,268
10	**2342**	**438**	**596,07**	8188848	75,3%	**4192,690**

✓ Always place often used or high rate data on outermost tracks (zone 1) …!?

↳ **NO**, arm movement is often more important than transfer time

Data Placement on Disk

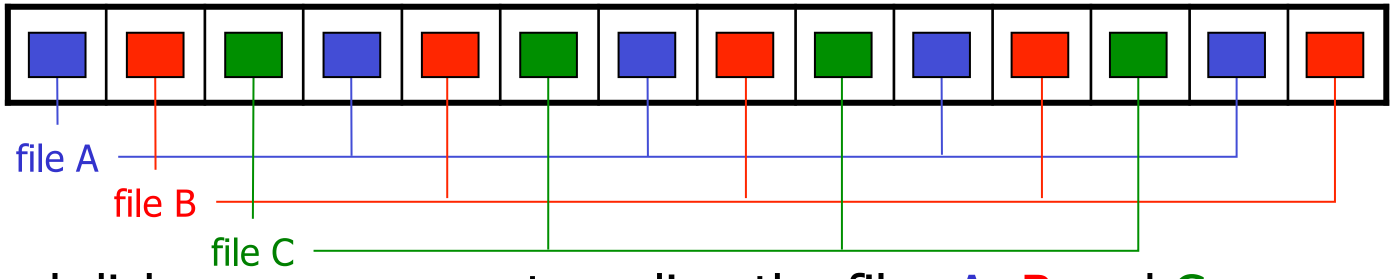
- Contiguous placement stores disk blocks contiguously on disk



  - minimal disk arm movement reading the whole file (no intra-file seeks)
  - pros/cons
    - ☺ head must not move between read operations - no seeks / rotational delays
    - ☺ can approach theoretical transfer rate
    - ☹ but usually we read other files as well (giving possible large inter-file seeks)
  - real advantage
    - do not have to pre-determine block (read operation) size
(whatever amount to read, at most track-to-track seeks are performed)
  - no inter-operation gain if we have unpredictable disk accesses

Data Placement on Disk

- Interleaved placement tries to store blocks from a file with a fixed number of other blocks in-between each block

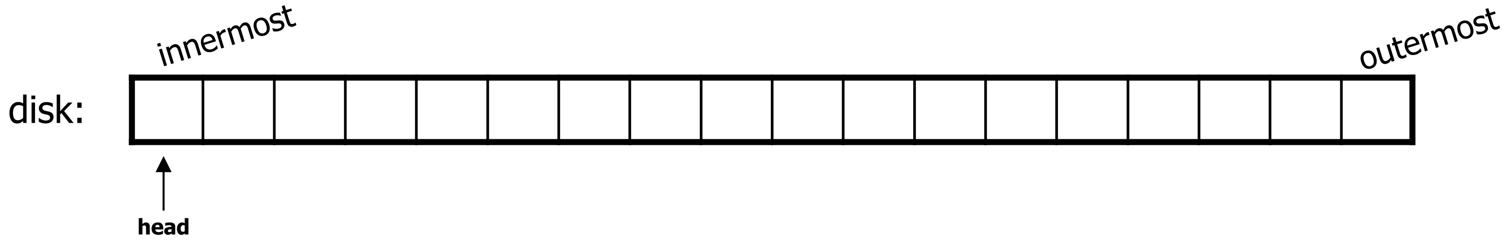


  - minimal disk arm movement reading the files A, B and C (starting at the same time)
  - fine for predictable workloads reading multiple files
  - no gain if we have unpredictable disk accesses

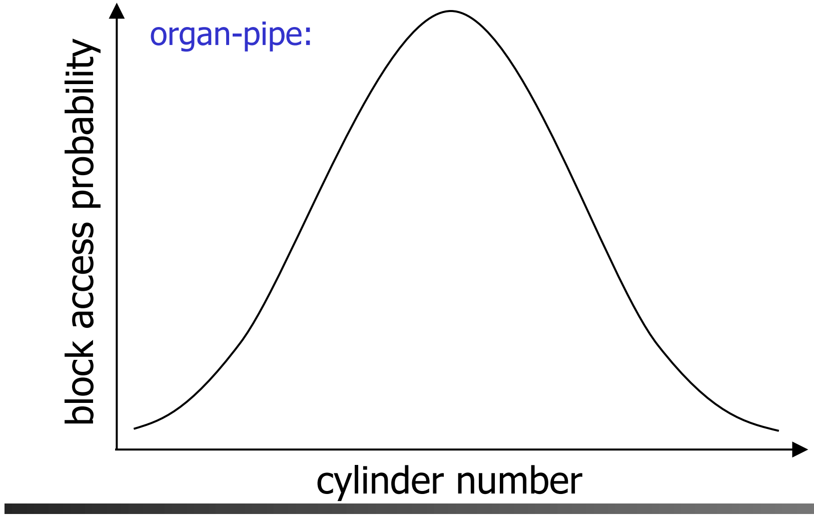
- Non-interleaved (or even random) placement can be used for highly unpredictable workloads

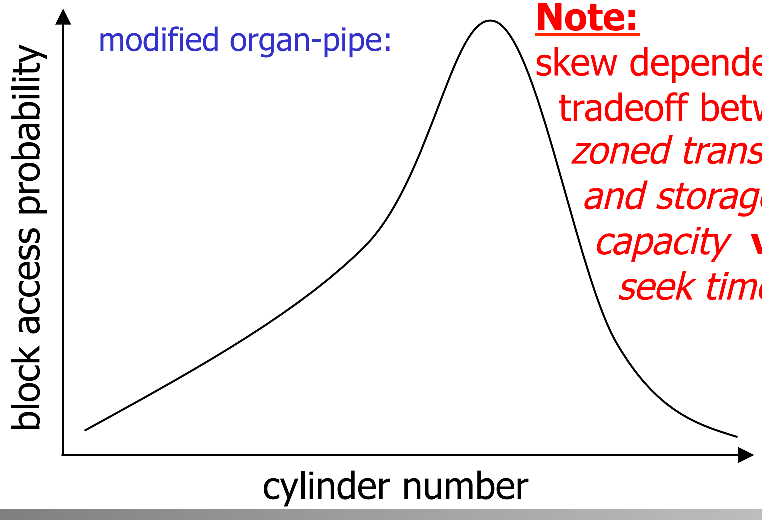
Data Placement on Disk

- Organ-pipe placement consider the 'average' disk head position
  - place most popular data where head is most often



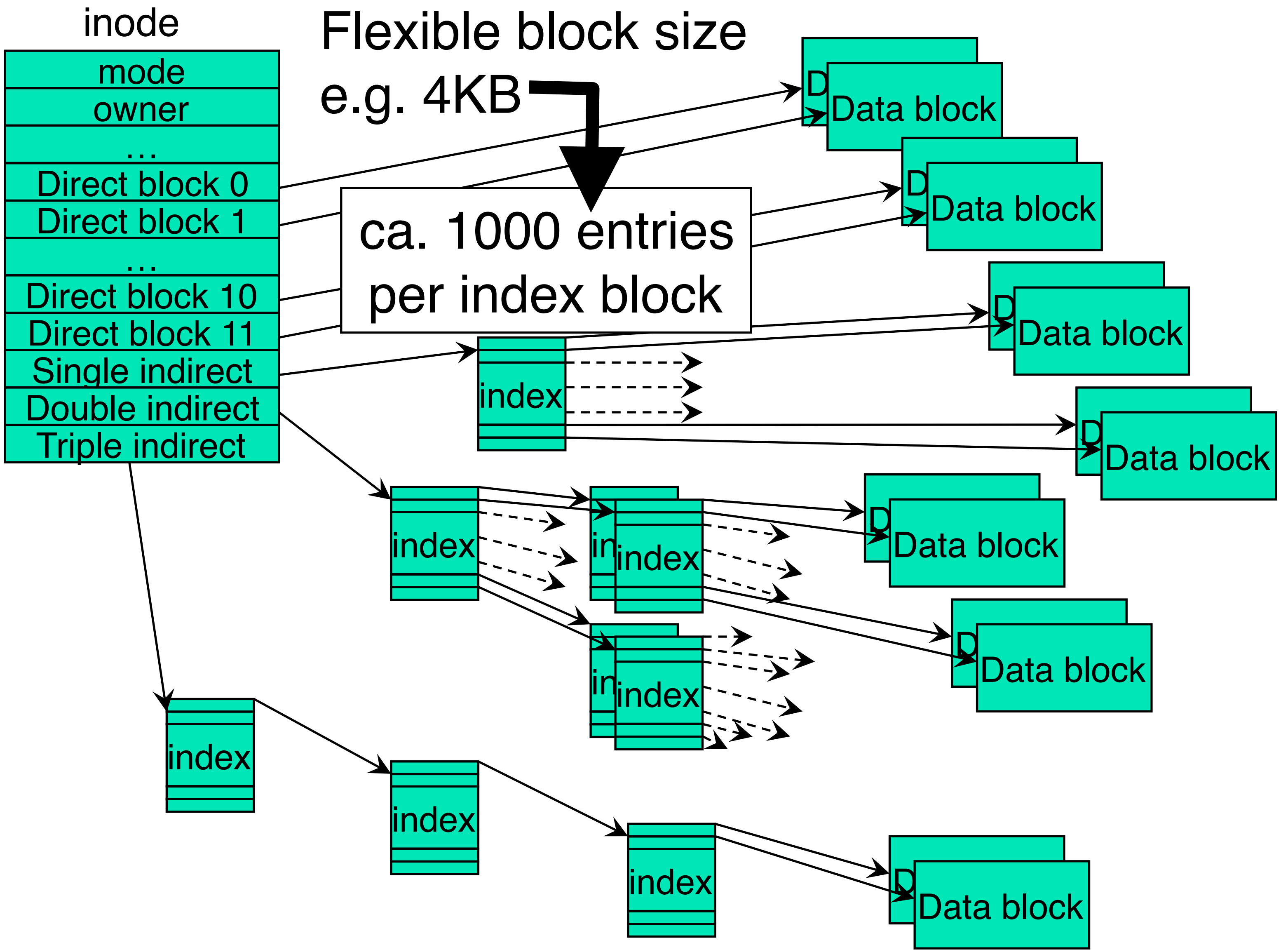
  - center of the disk is in average "closest" to the head
  - but, a bit outward for *zoned* disks (modified organ-pipe)





Note:
skew dependent on tradeoff between *zoned transfer time and storage capacity* **vs.** *seek time*

Unix/Linux Example: FFS, UFS, …



Linux Example: XFS, …, JFS, EXT4…

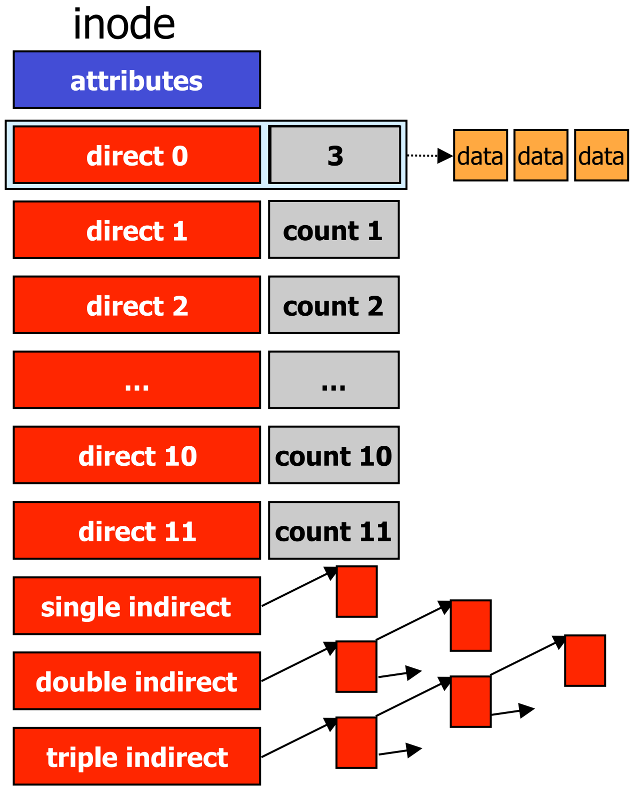
- Count-augmented address indexing in the extent sections

- Introduce a new inode structure

  – add counter field to original direct entries –

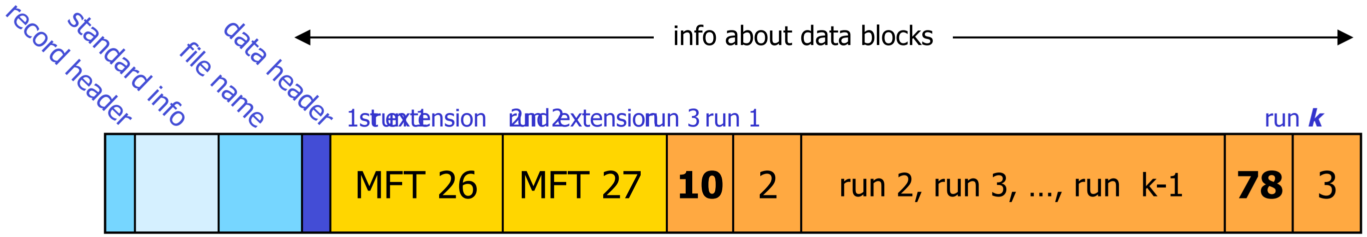
    - direct points to a disk block

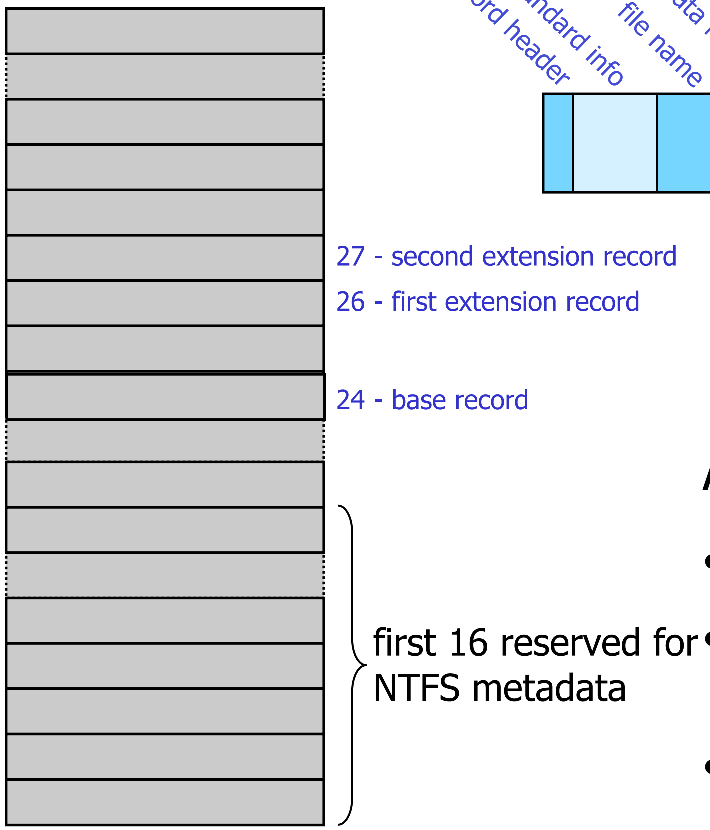
    - count indicated how many other blocks is following the first block (contiguously)



Windows Example: NTFS

- Each partition contains a master file table (MFT)
  - a linear sequence of 1 KB records
  - each record describes a directory or a file (attributes and disk addresses)





27 - second extension record
26 - first extension record
24 - base record
first 16 reserved for NTFS metadata

A file can be …

- stored within the record (immediate file, < few 100 B)
- represented by disk block addresses (which hold data): **runs** of consecutive blocks (<addr, no>, like extents)
- use several records if more runs are needed

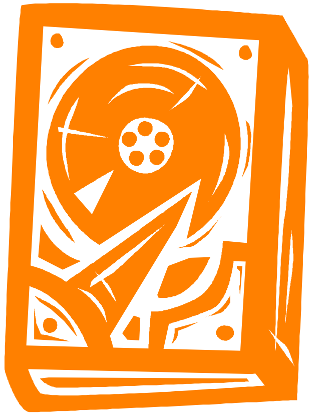
Modern Disks: Complicating Factors

Complicating Factors

- Disk used to be simple devices and disk scheduling used to be performed by OS (file system or device driver) only…

- … but, new disks are more complex
  - hide their true layout, e.g.,
    - only logical block numbers
    - different number of surfaces, cylinders, sectors, etc.

OS view

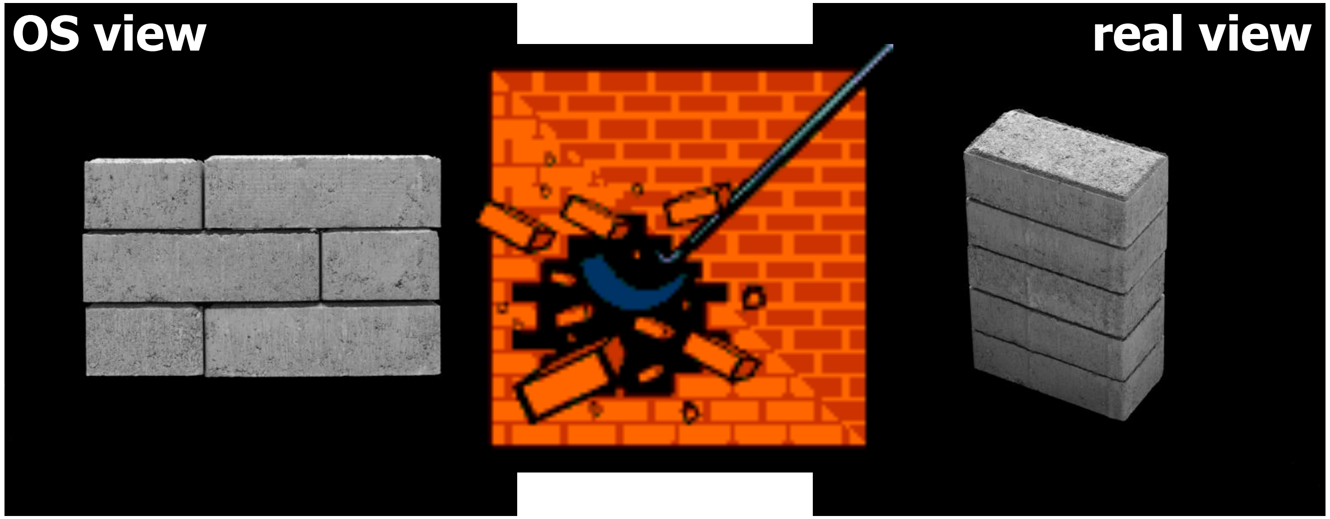


real view

Complicating Factors

- Disk used to be simple devices and disk scheduling used to be performed by OS (file system or device driver) only…

- … but, new disks are more complex
  - hide their true layout
  - transparently move blocks to spare cylinders
    - Seagate X15.3 - zone 1 - 10: (7,7,6,6,6,5,5,5,5,5)
    - e.g., due to bad disk blocks



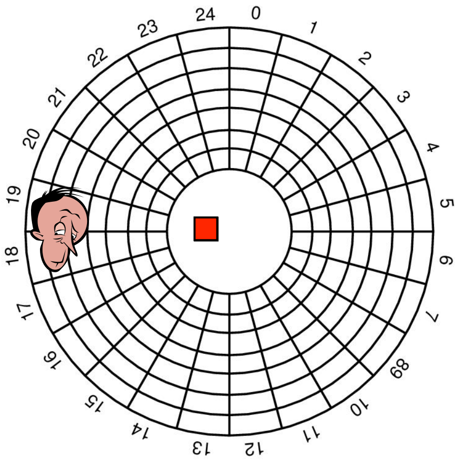
Complicating Factors

- Disk used to be simple devices and disk scheduling used to be performed by OS (file system or device driver) only…

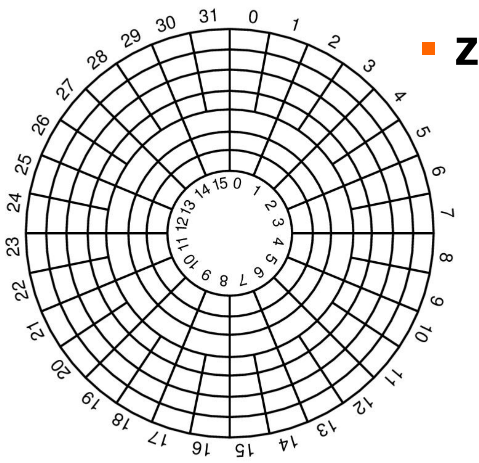
- … but, new disks are more complex
  - hide their true layout
  - transparently move blocks to spare cylinders
  - have different zones

OS view

- **Constant angular velocity (CAV) disks**
  - constant rotation speed
  - equal amount of data in each track
  - ⇨ thus, constant transfer time



real view

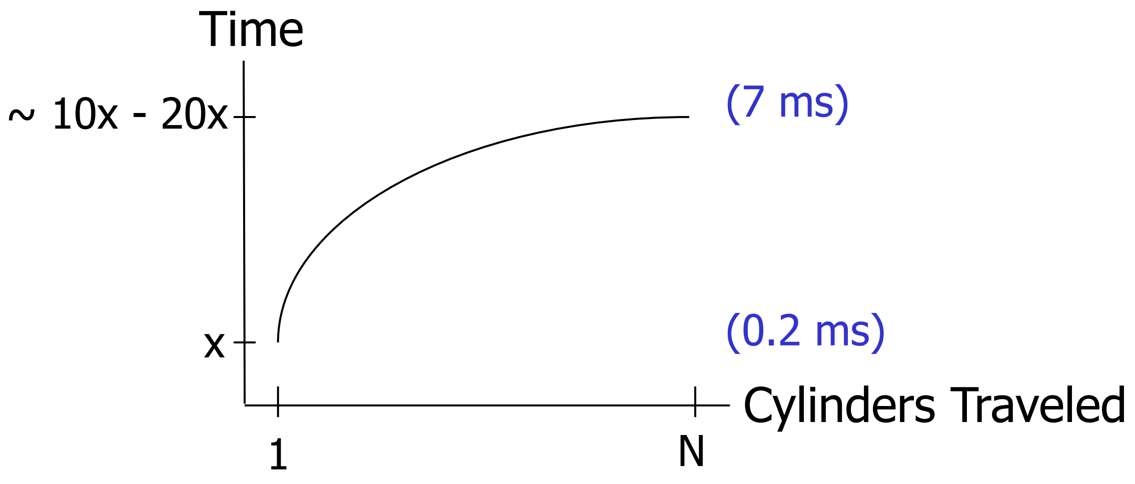


- **Zoned CAV disks**
  - constant rotation speed
  - zones are ranges of tracks
  - typical few zones
  - the different zones have different amount of data, i.e., more better on outer tracks
  - ⇨ thus, variable transfer time

Complicating Factors

- Disk used to be simple devices and disk scheduling used to be performed by OS (file system or device driver) only…

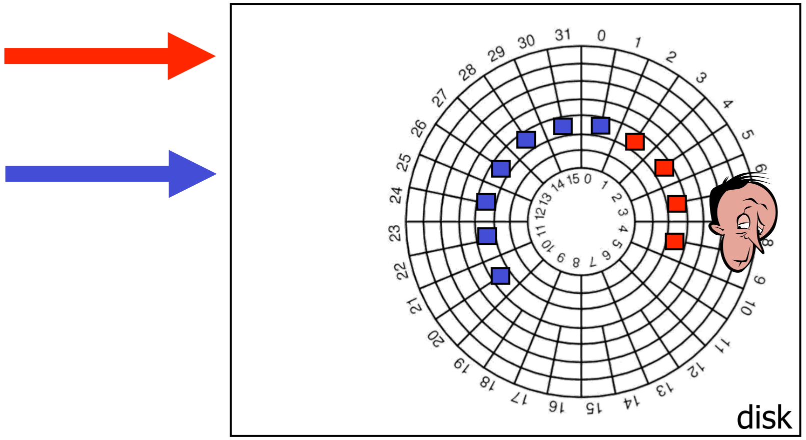
- … but, new disks are more complex
  - hide their true layout
  - transparently move blocks to spare cylinders
  - have different zones
  - head accelerates – most algorithms assume linear movement overhead

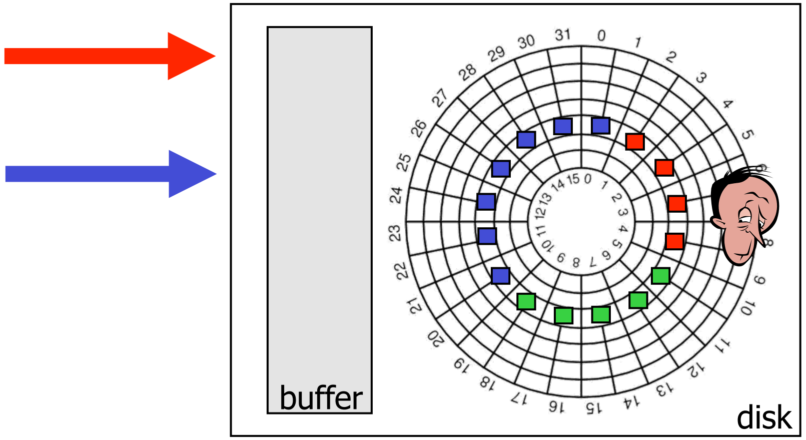


Complicating Factors

- Disk used to be simple devices and disk scheduling used to be performed by OS (file system or device driver) only…

- … but, new disks are more complex
  - hide their true layout
  - transparently move blocks to spare cylinders
  - have different zones
  - head accelerates
  - on device (and controller) buffer caches may use read-ahead prefetching

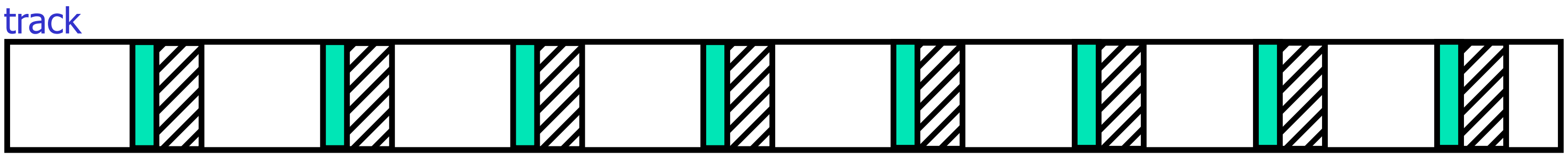




Complicating Factors

- Disk used to be simple devices and disk scheduling used to be performed by OS (file system or device driver) only…

- … but, new disks are more complex
  - hide their true layout
  - transparently move blocks to spare cylinders
  - have different zones
  - head accelerates
  - on device (and controller) buffer caches may use read-ahead prefetching
  - gaps and checksums between each sector



Complicating Factors

- Disk used to be simple devices and disk scheduling used to be performed by OS (file system or device driver) only…

- … but, new disks are more complex
  - hide their true layout
  - transparently move blocks to spare cylinders
  - have different zones
  - head accelerates
  - on device (and controller) buffer caches may use read-ahead prefetching
  - gaps and checksums between each sector

➡ "smart" with a build-in low-level scheduler (usually SCAN-derivate)

➡ we cannot fully control the device (black box)

Why Still Do Disk Related Research?

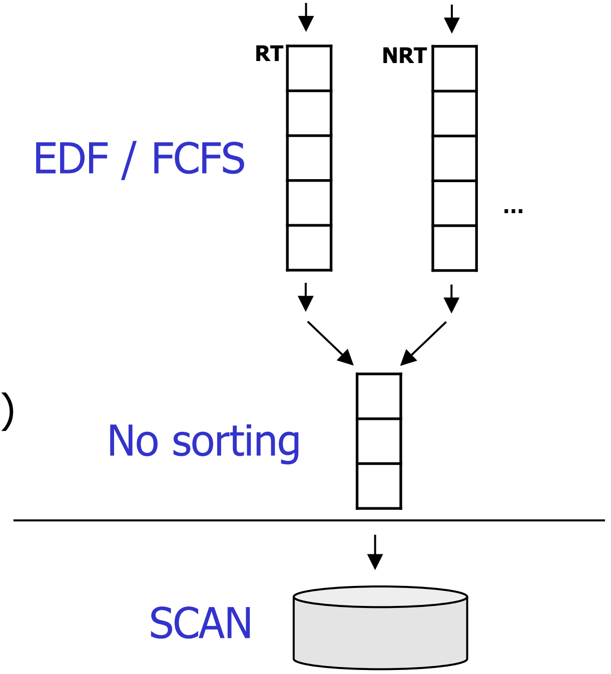
- If the disk is more or less a black box – why bother?
  - many (old) existing disks do not have the "new properties"
  - according to Seagate technical support:

    "blocks assumed contiguous by the OS probably still will be contiguous, but the whole section of blocks might be elsewhere"

    [private email from Seagate support]
  - delay sensitive requests
- But, the new disk properties should be taken into account
  - existing extent based placement is probably good
  - OS could (should?) focus on high level scheduling only

Next Generation Disk Scheduling?

- Thus, due to the complicating factors…
  (like head acceleration, disk buffer caches, hidden data layout, built-in "SCAN" scheduler,…)
  … a server scheduler can be (???):
  - hierarchical high-level software scheduler
    - several top-level queues (at least RT & NRT)
    - process queues in rounds (RR)
      - dynamic assignment of quantums
      - work-conservation with variable round length
        (full disk bandwidth utilization vs. buffer requirement)
    - only simple collection of requests according to quantums in lowest level and forwarding to disk, because ...
  - ...fixed SCAN scheduler in hardware (on disk)

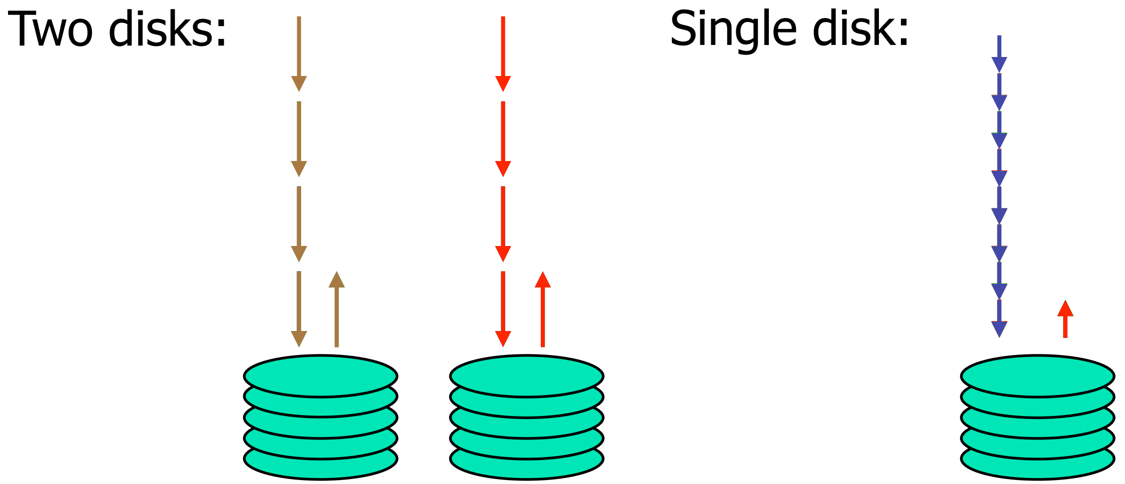


- On-device programmable processors??

Multiple Disks

Parallel Access

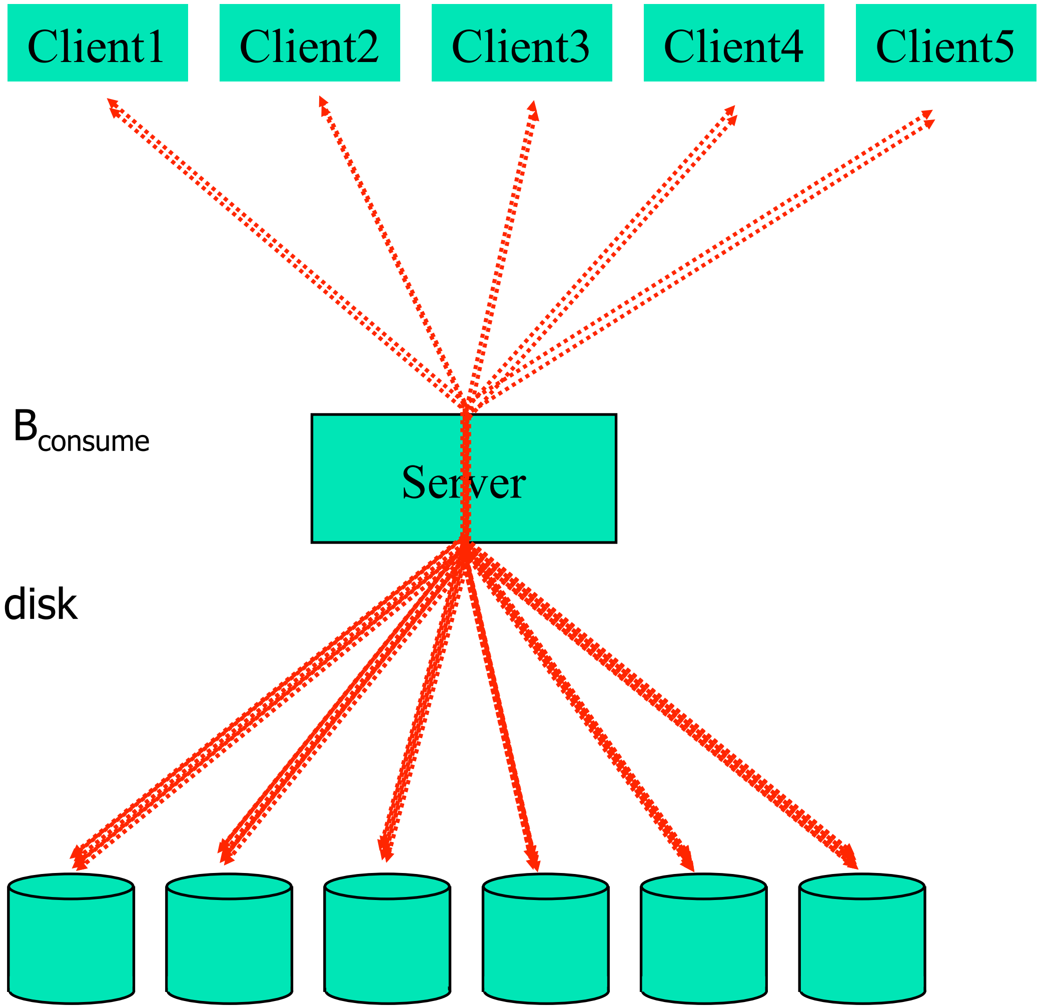
- Disk controllers and busses manage several devices
- One *can* improve total system performance by replacing one large disk with many small accessed in parallel
- Several independent heads can read simultaneously



Note:
the single disk might be faster, but as seek time and rotational delay are the dominant factors of total disk access time, the two smaller disks might operate faster together performing seeks in parallel...

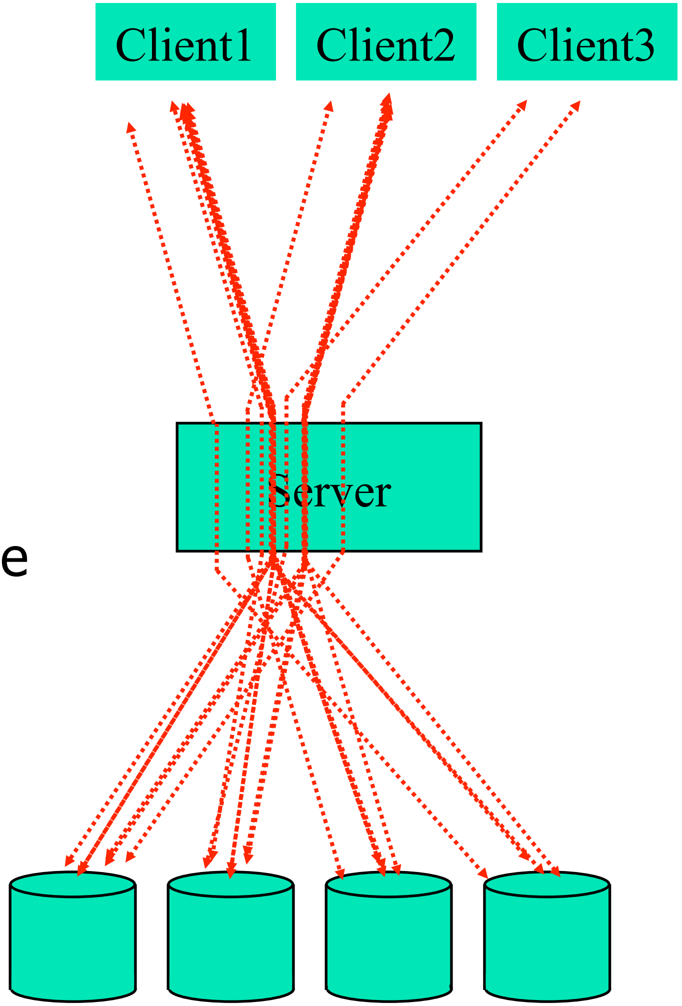
Striping

- Another reason to use multiple disks is when one disk cannot deliver requested data rate
- In such a scenario, one might use several disks for striping:
  - bandwidth disk: B_{disk}
  - required bandwidth: $B_{consume}$
  - $B_{disk} < B_{consume}$
  - read from *n* disks in parallel: $n\ B_{disk} > B_{consume}$
- Advantages
  - higher transfer rate compared to one disk
- Drawbacks
  - can’t serve multiple clients in parallel
  - positioning time increases (i.e., reduced efficiency)



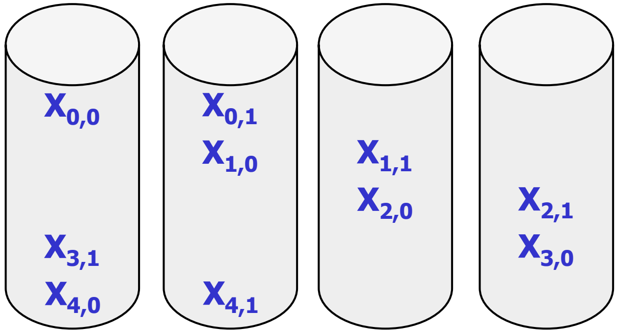
Interleaving (Compound Striping)

- Full striping usually not necessary today:
  - faster disks
  - better compression algorithms
- Interleaving lets each client be serviced by only a set of the available disks
  - make groups
  - ”stripe” data in a way such that a consecutive request arrive at next group
  - one disk group example:



Interleaving (Compound Striping)

- Divide traditional striping group into sub-groups, e.g., staggered striping



- Advantages
  - multiple clients can still be served in parallel
  - more efficient disks operations
  - potentially shorter response time

- Potential drawback/challenge
  - load balancing (all clients access same group)

Mirroring

- Multiple disks might do come in the situation where all requests are for one of the disks and the rest lie idle
- In such cases, it might make sense to have replicas of data on several disks – if we have identical disks, it is called mirroring
- Advantages
  - faster response time
  - survive crashes – fault tolerance
  - load balancing by dividing the requests for the data on the same disks equally among the mirrored disks
- Drawbacks
  - increases storage requirement
  - more expensive write operations

Redundant Array of Inexpensive Disks

- The various RAID levels define different disk organizations to achieve higher performance and more reliability
  - RAID 0 - striped disk array without fault tolerance (non-redundant)
  - RAID 1 - mirroring
  - RAID 2 - memory-style error correcting code (Hamming Code ECC)
  - RAID 3 - bit-interleaved parity
  - RAID 4 - block-interleaved parity
  - RAID 5 - block-interleaved distributed-parity
  - RAID 6 - independent data disks with two independent distributed parity schemes (P+Q redundancy)
  - RAID 10 - striped disk array (RAID level 0) whose segments are mirrored (level 1)
  - RAID 0+1 - mirrored array (RAID level 1) whose segments are RAID 0 arrays
  - RAID 03 - striped (RAID level 0) array whose segments are RAID level 3 arrays
  - RAID 50 - striped (RAID level 0) array whose segments are RAID level 5 arrays
  - RAID 53, 51, …

Replication

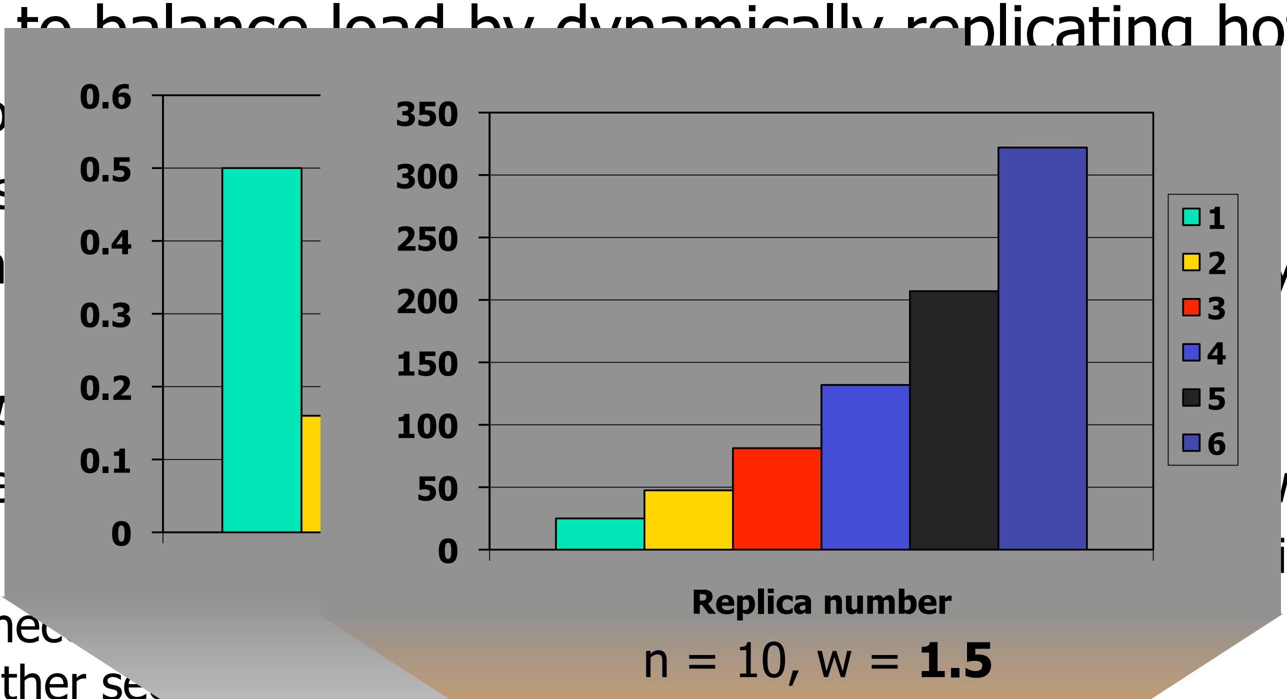
- Replication is in traditional disk array systems often used for fault tolerance (and higher performance in the new combined RAID levels)

- Replication can also be used for
  - reducing hot spots
  - increase scalability
  - higher performance
  - …
  - and, fault tolerance is often a side effect ☺

- Replication should
  - be based on observed load
  - changed dynamically as popularity changes

Dynamic Segment Replication (DSR)

- DSR tries to balance load by dynamically replicating hot data
  - also kno
  - assumes
  - predefin y examining current
  - uses *co*
  - replicate here??
    - tries ions
    - not ne (another se
    - ↳ replicates based nt i with highest p):



n = 10, w = **1.5**

$$p_i = \left(\frac{1}{r_i} - \frac{1}{r_i + 1} \right) \sum_{j=0}^{i-1} n_j w^{i-j-1}$$

number of viewers of replica j

number of replicas of segment i

factor for expected benefit for additional copy

weighting factor

this sum considers the number of future viewers for this segment

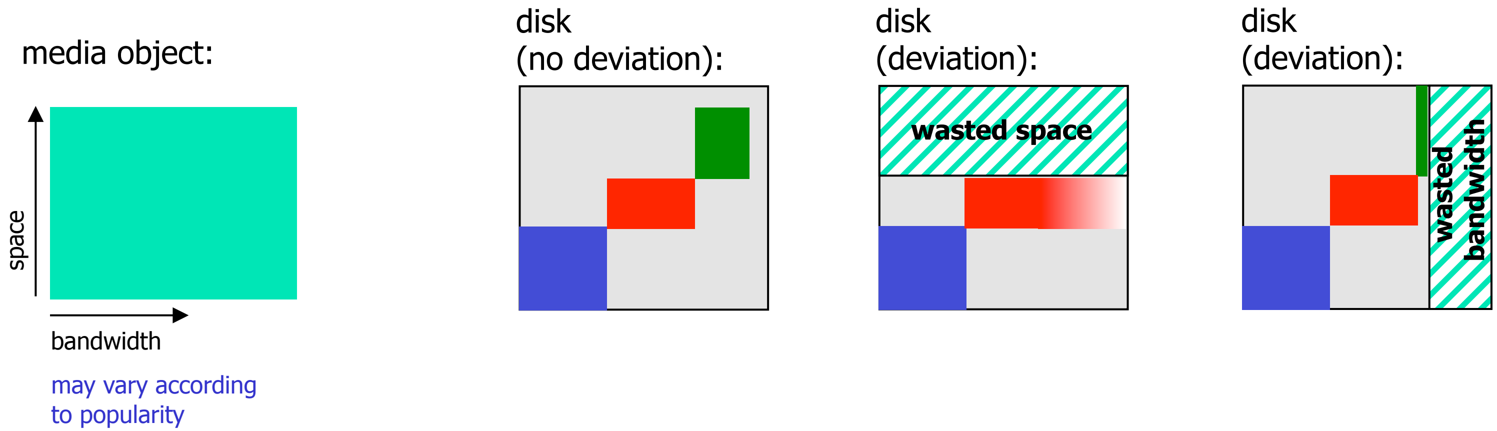
Heterogeneous Disks

File Placement

- A file might be stored on multiple disks, but how should one choose on which devices?
  - storage devices limited by both bandwidth and space
  - we have *hot* (frequently viewed) and *cold* (rarely viewed) files
  - we may have several *heterogeneous* storage devices
  - the objective of a file placement policy is to achieve maximum utilization of both bandwidth and space, and hence, efficient usage of all devices by avoiding load imbalance
    - must consider expected load and storage requirement
    - expected load may change over time

Bandwidth-to-Space Ratio (BSR)

- BSR attempts to mix hot and cold as well as large and small (multimedia) objects on heterogeneous devices
  - don't optimize placement based on throughput or space only (use both!!)



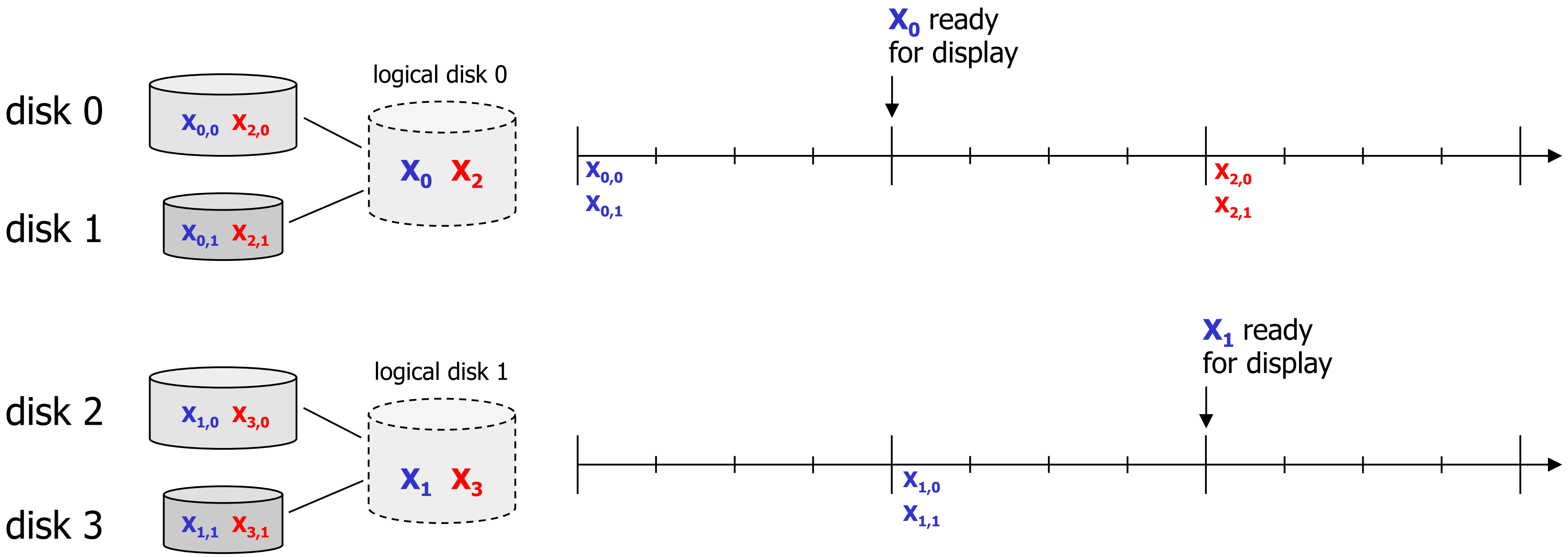
  - BSR consider both required *storage space* and *throughput* requirement (which is dependent on playout rate and popularity) to achieve a best combined device utilization

Bandwidth-to-Space Ratio (BSR)

- The BSR policy algorithm:
  - input: space and bandwidth requirements
  - phase 1:
    - find a device to place the media object according to BSR
    - if no device, or stripe of devices, can give sufficient space or bandwidth, then add replicas
  - phase 2:
    - find devices for the needed replicas
  - phase 3:
    - allocate expected load on replica devices according to BSR of the devices
  - phase 4:
    - if not enough resources are available, see if other media objects can delete replicas according to their current workload
  - all phases may be needed adding a new media object or increasing the workload – for decrease, only the *reallocation* in needed
- Popular, high data rate movies should be on high bandwidth disks

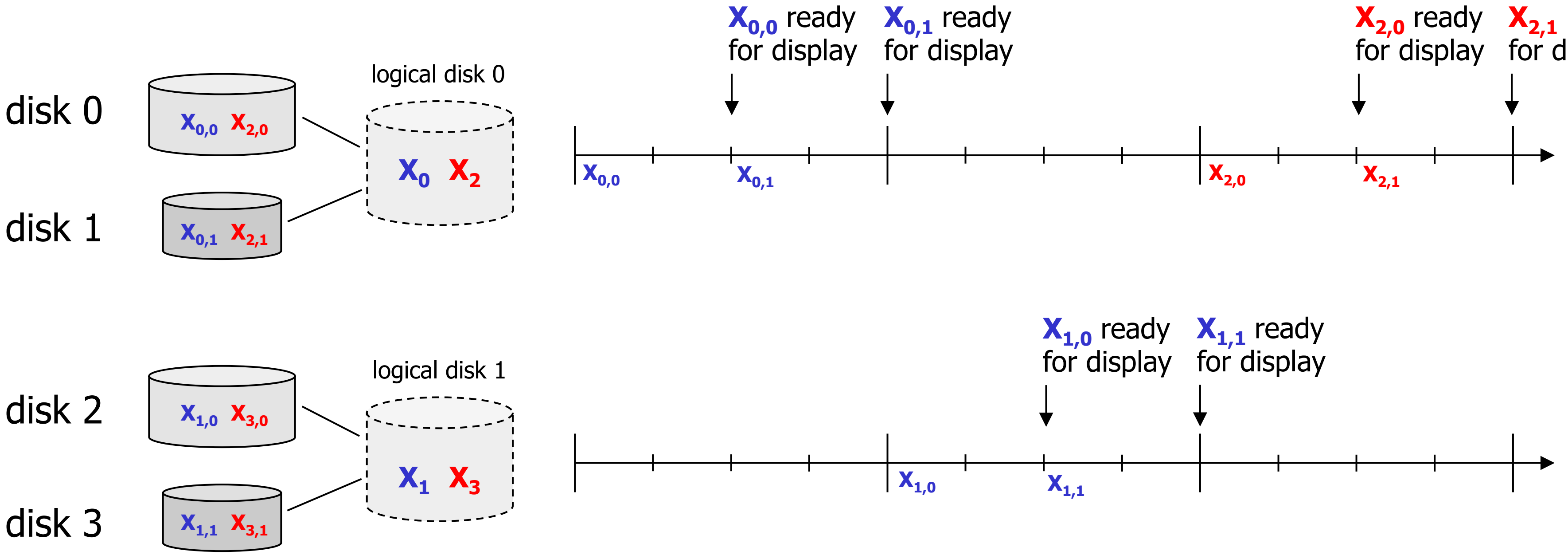
Disk Grouping

- Disk grouping is a technique to “stripe” (or fragment) data over heterogeneous disks
  - groups heterogeneous physical disks to homogeneous logical disks
  - the amount of data on each disk (fragments) is determined so that the service time (based on worst-case seeks) is equal for all physical disks in a logical disk
  - blocks for an object are placed (and read) on logical disks in a round-robin manner – all disks in a group is activated simultaneously



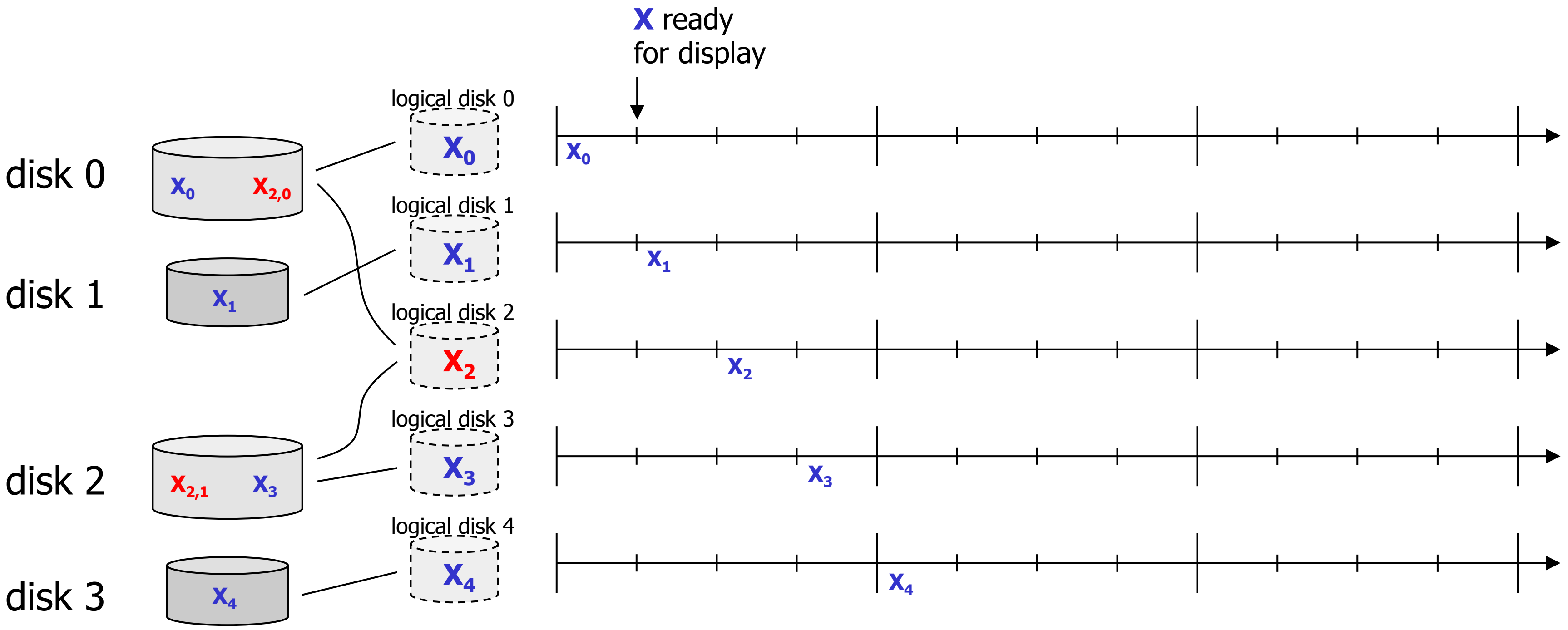
Staggered Disk Grouping

- Staggered disk grouping is a variant of disk grouping minimizing memory requirement
  - reading and playing out differently
  - not all fragments of a logical block is needed at the same time
  - first (and largest) fragment on most powerful disk, etc.
  - read sequentially (must not buffer later segments for a long time)
  - start display when largest fragment is read



Disk Merging

- Disk merging forms logical disks from capacity fragments of a physical disk
  - all logical disks are homogeneous
  - supports an arbitrary mix of heterogeneous disks (grouping needs equal groups)
  - starts by choosing how many logical disks the slowest device shall support (e.g., 1 for disk 1 and 3) and calculates the corresponding number of more powerful devices (e.g., 1.5 for disk 0 and 2 if these disks are 1.5 times better)
  - most powerful: most flexible (arbitrary mix of devices) and can be adapted to zoned disks (each zone considered as a disk)



File Systems

File Systems

- Many examples of application specific storage systems
  - integrate several subcomponents (e.g., scheduling, placement, caching, admission control, …)
  - often labeled differently: file system, file server, storage server, …
    → accessed through typical file system abstractions
  - need to address applications distinguishing features:
    - soft real-time constraints (low delay, synchronization, jitter)
    - high data volumes (storage and bandwidth)

Classification

- General file systems: "support" for all applications
  e.g.: file allocation table (FAT), windows NT file system (NTFS), extended file system (Ext2/3/4), journaling file system (JFS), Reiser, fast file system (FFS), …

- Multimedia file systems: address multimedia requirements
  - *general file systems with multimedia support* e.g.: XFS, Minorca
  - *exclusively streaming*
    e.g.: Video file server, embedded real-time file system (ERTFS), Shark, Everest, continuous media file system (CMFS), Tiger Shark
  - *several application classes*
    e.g.: Fellini, Symphony, (MARS & APEX schedulers)

- High-performance file systems: primarily for large data operations in short time
  e.g.: general parallel file system (GPFS), clustered XFS (CXFS), Frangipani, global file system (GFS), parallel portable file system (PPFS), Examplar, extensible file system (ELFS)

- "Strange file systems":
  e.g., Google FS (BigTable), OceanStore, FAST, FUSE, …

Example: Fellini Storage System

- **Fellini** (now CineBlitz)…
  - supports both *real-time* (with guarantees) and *non-real-time* by assigning resources for both classes
  - SGI (IRIX Unix), Sun (Solaris), PC (WinNT & Win95)

- Admission control
  - deterministic (worst-case) to make hard guarantees
  - services streams in rounds
  - used (and available) **disk BW** is calculated using
    - worst-case
      - seek (inner to outer)
      - rotational delay (one round)
      - settle (servicing latency) - transfer rate of inner track
    - T_{period} > total disk time = 2 x seek + $\sum$ [blocks$_i$ x (rotation delay + settle)]
  - used (and available) **buffer space** is calculated using
    - buffer requirement per stream = *2 x rate x service round*
  - a new client is admitted if enough free disk BW and buffer space (additionally Fellini checks network BW)
  - new real-time clients are admitted first

Example: Fellini Storage System

- Cache manager
  - pages are pinned (fixing) using a reference counter
  - replacement in three steps
    1. search free list
    2. search current buffer list (CBL) for the *unused* LRU file
    3. search *in-use* CBLs and assign priorities to replaceable buffers (not pinned) according to reference distance (depending on rate, direction)
       - sort (using Quicksort)
       - replace buffer with highest weight

Example: Fellini Storage System

- Storage manager
  - maintains free list with grouping contiguous blocks → store blocks contiguously
  - uses C-SCAN disk scheduling
  - striping
    - distribute the total load
    - add fault-tolerance (parity data)
  - simple flat file system

- Application interface
  - non-real-time: more or less as in other file systems, except that when opening one has an admittance check
  - real-time:
    - `begin_stream (filename, mode, flags, rate)`
    - `retrieve_stream (id, bytes)`
    - `store_stream (id, bytes)`
    - `seek_stream (id, bytes, whence)`
    - `close_stream(id)`

Discussion:

We have the Qs, you have the As!

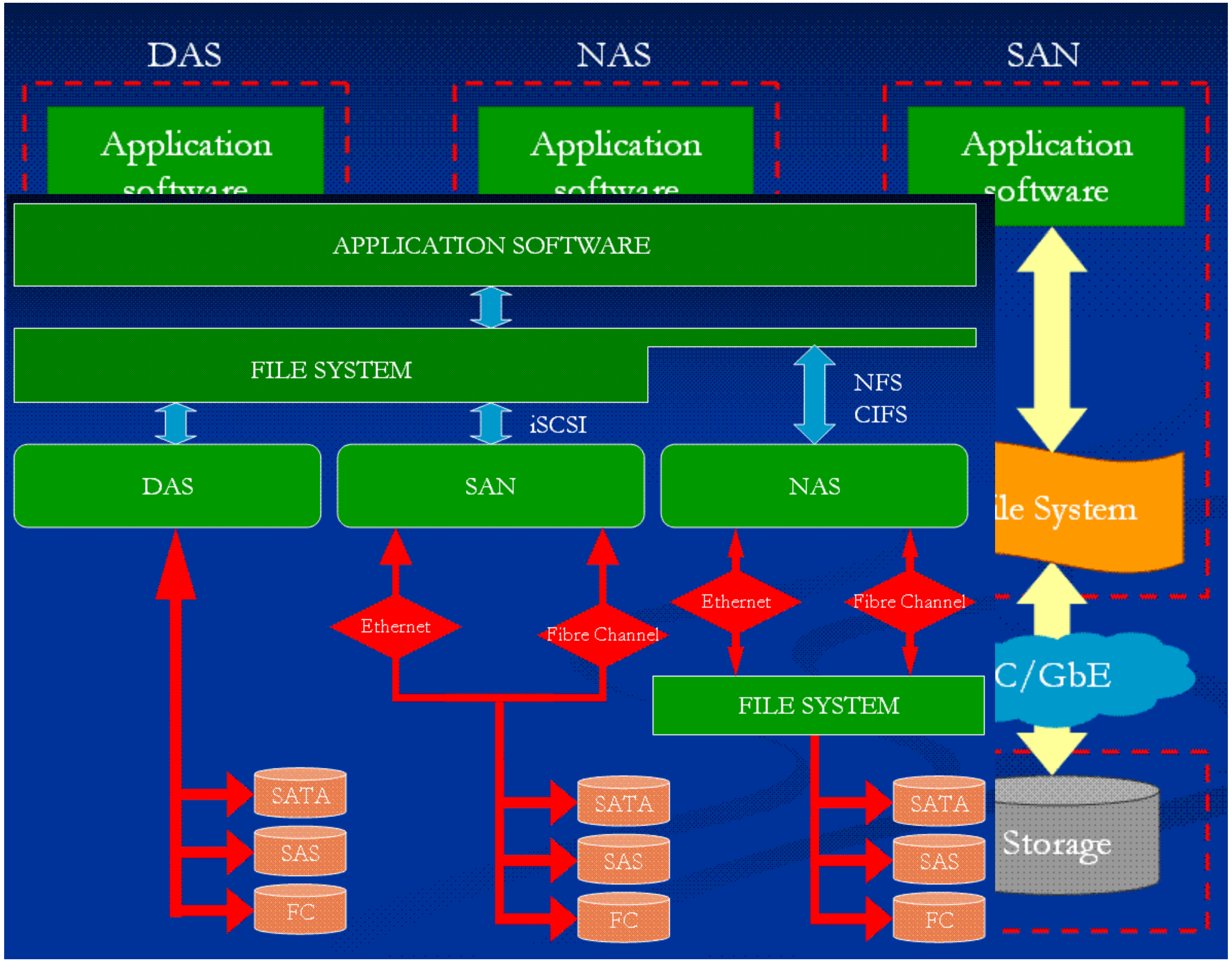
DAS vs. NAS vs. SAN??

- How will the introduction of **network attached disks** influence storage?

- Direct attached storage

- Network attached storage
  - uses some kind of file-based protocol to attach remote devices non-transparently
  - NFS, SMB, CIFS

- Storage area network
  - transparently attach remote storage devices
  - iSCSI (SCSI over TCP/IP), iFCP (SCSI over Fibre Channel), HyperSCSI (SCSI over Ethernet), ATA over Ethernet



Mechanical Disks vs. Solid State Disks???

- How will the introduction of **SSDs** influence storage?

(numbers from 2008)	**Storage capasity (GB)**	**Average seek time / latency (ms)**	**Sustained transfer rate (MBps)**	**Interface (Gbps)**
Seagate Cheetah X15.6 (3.5 inch)	450	**3.4** (track to track 0.2)	110 - **171**	SAS (3) FC (4)
Seagate Savvio 15K (2.5 inch)	73	**2.9** (track to track 0.2)	29 - **112**	SAS (3)
OCM Flash Media Core Series V2	250	< .2 - **.3**	up to **170**	SATA (3)
Intel X25-E (extreme)	64	**0.075**	**250**	SATA (3)
Intel X25-M (mainstream)	160	**0.085**	**250**	SATA (3)
Mtron SSD Pro 7500 series	128	**0.100**	**130**	SATA (1.5)
Gigabyte GC-Ramdisk	4	**0.000xxx**	**GBps**	SATA (1.5)

Evolution: New Requirements

- Architectural considerations [Prashant Shenoy et al]:
  - integrated file system support for a variety of applications
  - modernizing the multimedia file system
    - server-independent
    - self managing
    - self healing
    - networked
    - disk processors
- Trend in research towards high-performance file systems
  - usually no timeliness guarantees, but performance is maximized
  - several build on multimedia file systems (Tiger Shark → GPFS, XFS → CXFS), but have gained scalability while still supporting reservation
  - efficient support for operations like strided (non-continuous) I/O will be increasingly important (edition, interactions, scalable streaming, non-linearity)

The End:
Summary

Summary

- All resources needs to be scheduled

- Scheduling algorithms have to…
  - … be fair
  - … consider real-time requirements (if needed)
  - … provide good resource utilization
  - (… be implementable)

- Memory management is an important issue
  - caching
  - copying is expensive → copy-free data paths

Summary

- The main bottleneck is disk I/O performance due to disk mechanics: seek time and rotational delays
  (but in the future??)

- Much work has been performed to optimize disks performance

- Many algorithms trying to minimize seek overhead (most existing systems uses a SCAN derivate)

- World today more complicated
  - both different media
  - unknown disk characteristics –
    new disks are "smart", we cannot fully control the device

- Disk arrays frequently used to improve the I/O capability

- Many existing file systems with various application specific support

Some References

1. Halvorsen, P.: “Improving I/O Performance of Multimedia Servers”, Thesis for the Dr. Scient. degree at University of Oslo, Unipub forlag, ISSN 1501-7710, No. 161, Oslo, Norway, August 2001
2. Halvorsen, P., Dalseng, T.A., Griwodz, C.: Assessment of Data Path Implementations for Content Download and Streaming, Proc. of 11th Int. Conf. on Distributed Mulrimedia Systems, Banff, Canada, September 2005
3. Liu, C.L., Layland, J.W.: "*Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment*“, Journal of the Association for Computing Machinery 20, 1 (January 1973): 40-61
4. Nieh, J., Lam, M.S.: “*The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications*”, Proc. of 16th ACM Symp. on Operating System Principles (SOSP’97), St. Malo, France, October 1997, pp. 184-197
5. Plagemann, T., Goebel, V., Halvorsen, P., Anshus, O.: *"Operating System Support for Multimedia Systems"*, The Computer Communications Journal, Elsevier, Vol. 23, No. 3, February 2000, pp. 267-289
6. Solomon, D.A., Russinovich, M.E.: “Inside Microsoft Windows2000”, 3rd edition, Microsoft Press, 2000
7. Steinmetz, R., Nahrstedt, C.: “*Multimedia: Computing, Communications & Applications*”, Prentice Hall, 1995
8. Tanenbaum, A.S.: “Modern Operating Systems” (2nd ed.), Prentice Hall, 2001
9. Wolf, L.C., Burke, W., Vogt, C.: “Evaluation of a CPU Scheduling Mechanism for Multimedia Systems”, Software - Practice and Experience, Vol. 26, No. 4, 1996, pp. 375 – 398
10. Boll, S., Heinlein, C., Klas, W., Wandel, J.: “MPEG-L/MRP: Adaptive Streaming of MPEG Videos for Interactive Internet Applications”, Proceedings of the 6th International Workshop on Multimedia Information System (MIS’00), Chicago, USA, October 2000, pp. 104 - 113
11. Halvorsen, P., Goebel, V., Plagemann, T.: “Q-L/MRP: A Buffer Management Mechanism for QoS Support in a Multimedia DBMS”, Proceedings of 1998 IEEE International Workshop on Multimedia Database Management Systems (IW-MMDBMS'98), Dayton, Ohio, USA, August 1998, pp. 162 – 171
12. Moser, F., Kraiss, A., Klas, W.: “L/MRP: a Buffer Management Strategy for Interactive Continuous Data Flows in a Multimedia DBMS”, Proceedings of the 21th VLDB Conference, Zurich, Switzerland, 1995

Some References

1. Advanced Computer & Network Corporation: “RAID.edu”, http://www.raid.com/04_00.html, 2002
2. Halvorsen, P., Griwodz, C., Goebel, V., Lund, K., Plagemann, T., Walpole, J.: “Storage System Support for Continuous-Media Applications” (part 1 & 2), DSonline, Vol. 5, No. 1 & 2, January/February 2004
3. C. Martin, P.S. Narayan, B. Ozden, R. Rastogi, and A. Silberschatz, ``The Fellini Multimedia Storage System," Journal of Digital Libraries , 1997, see also http://www.bell-labs.com/project/fellini/
4. Plagemann, T., Goebel, V., Halvorsen, P., Anshus, O.: “Operating System Support for Multimedia Systems”, Computer Communications, Vol. 23, No. 3, February 2000, pp. 267-289
5. Shenoy, P., Goyal, P., Rao, S.S., Vin, H.M.: “Symphony: An Integrated Multimedia Files System”, MMCN’98, San Jose, CA, USA, 1998, pp. 124 – 138 (also as CS-TR-97-09)
6. Sitaram, D., Dan, A.: “Multimedia Servers – Applications, Environments, and Design”, Morgan Kaufmann Publishers, 2000
7. Zimmermann, R., Ghandeharizadeh, S.: “Continuous Display using Heterogeneous Disk-Subsystems”, Proceedings of the 5th ACM International Multimedia Conference, Seattle, WA, November 1997
8. Anderson, D. P., Osawa, Y., Govindan, R.:”A File System for Continuous Media”, ACM Transactions on Computer Systems, Vol. 10, No. 4, Nov. 1992, pp. 311 - 337
9. Elmasri, R. A., Navathe, S.: “Fundamentals of Database Systems”, Addison Wesley, 2000
10. Garcia-Molina, H., Ullman, J. D., Widom, J.: “Database Systems – The Complete Book”, Prentice Hall, 2002
11. Lund, K.: “Adaptive Disk Scheduling for Multimedia Database Systems”, PhD thesis, IFI/UniK, UiO
12. Plagemann, T., Goebel, V., Halvorsen, P., Anshus, O.: “Operating System Support for Multimedia Systems”, Computer Communications, Vol. 23, No. 3, February 2000, pp. 267-289
13. Seagate Technology, http://www.seagate.com