Week 10

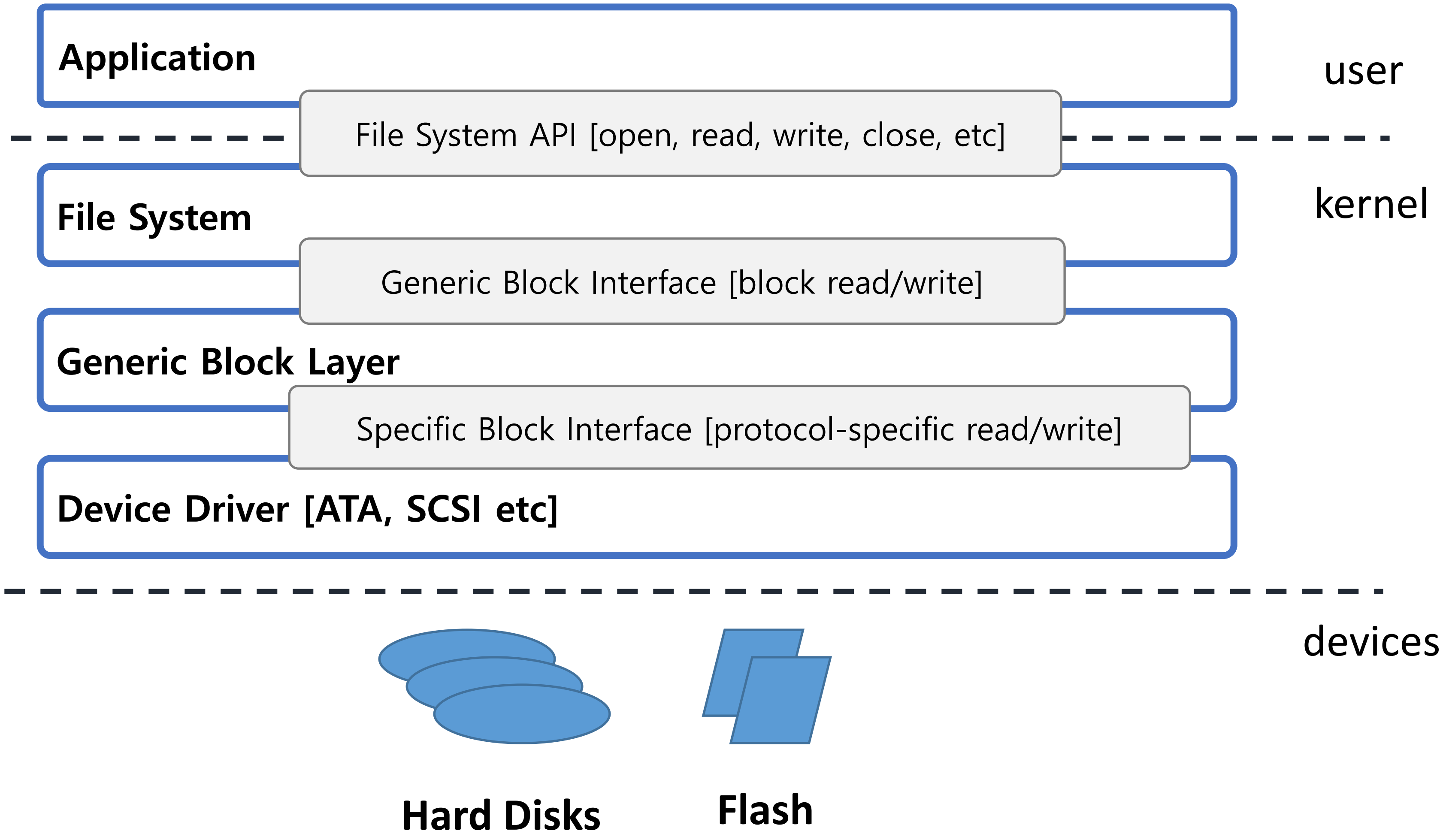
Persistent Storage: Intro to File Systems

Oana Balmau
March 9, 2023

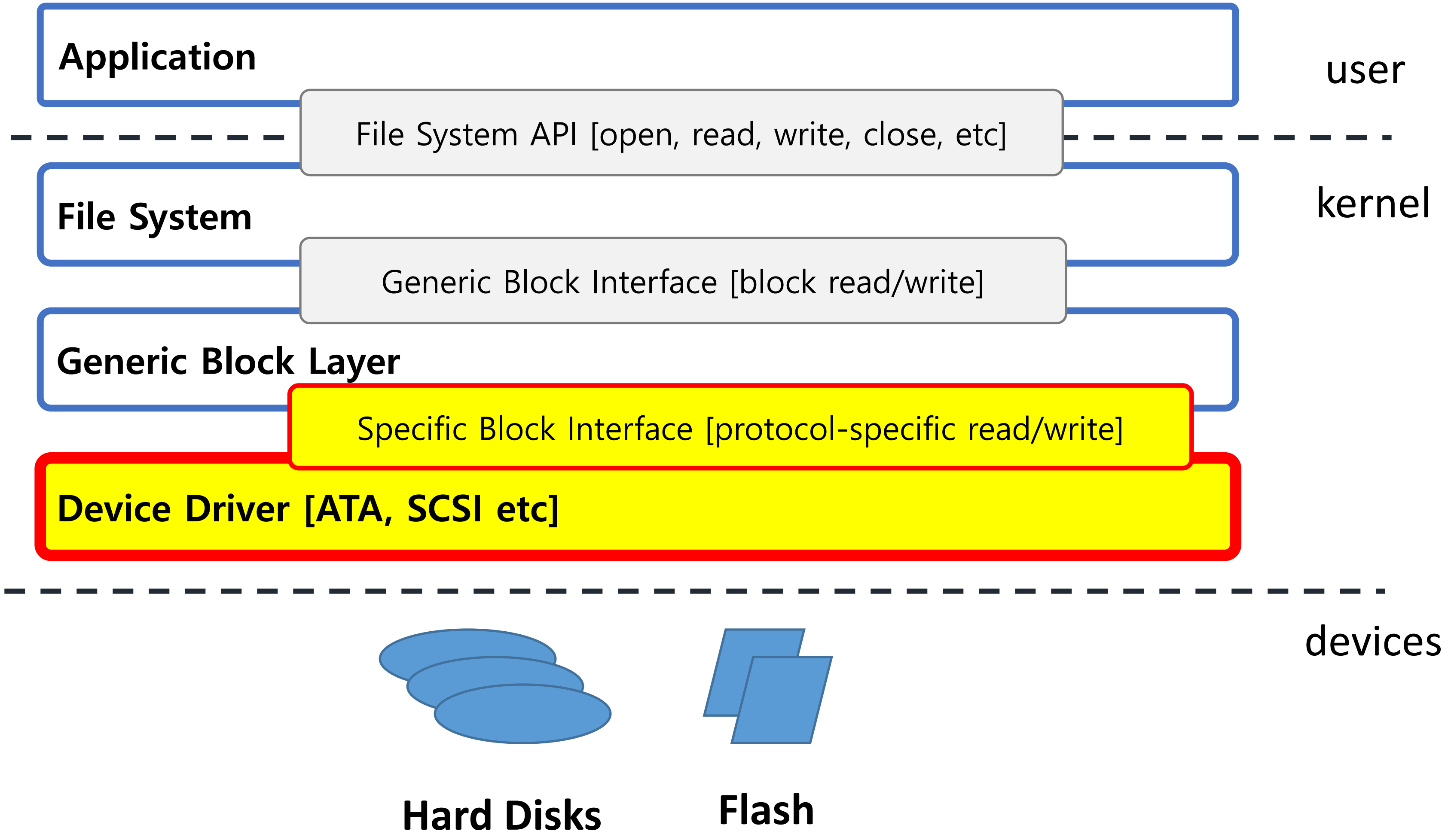
Class Admin

Week 10 **File Systems**	mar 6 No lab. Work on Assignment 2 ~~**Scheduling Assignment Due**~~	mar 7 **Intro to File Systems (1/2)** Recorded lecture. Do not come to class. Optional reading: OSTEP Chapters 36, 37, 39	mar 8 Memory Management Assignment Released	mar 9 **Intro to File Systems (2/2)** Memory Management Assignment Overview — with Jiaxuan	mar 10
Week 11 **File Systems**	mar 13 Scheduling Assignment Due ~~**Graded Exercises Due**~~ C Review: Complex structs	mar 14 **Basic File System Implementation (1/2)** Optional reading: OSTEP Chapters 40, 41, 45	mar 15	mar 16 **Basic File System Implementation (2/2)** • ~~Grades released for Scheduling Assignment~~	mar 17
Week 12 **File Systems**	mar 20 Graded Exercises Due C Review: Pointers & Memory Allocation II	mar 21 **Advanced File System Implementation (1/2)**	mar 22	mar 23 **Advanced File System Implementation (2/2)** • Grades released for Scheduling Assignment	mar 24
Week 13 **File Systems**	mar 27 C Review: Advanced debugging	mar 28 **Handling Crashes & Performance (1/2)** Optional reading: OSTEP Chapters 38, 43	mar 29	mar 30 **Handling Crashes & Performance (2/2)** • ~~Grades released for Exercises Sheet~~ • Practice Exercises Sheet: File Systems	mar 31
Week 14 **Advanced Topics**	apr 3 No lab. Work on Assignment 3 ~~**Memory Management Assignment Due**~~	apr 4 **Advanced topics: Virtualization**	apr 5	apr 6 **Advanced topics: Operating Systems Research** (Invited Speaker: TBD) Grades released for Exercises Sheet	apr 7
Week 15 **Wrap-up**	apr 10 No Lab. Prepare for end-of-semester. Memory Management Assignment Due	apr 11 End-of-semester Q&A– not recorded	apr 12	apr 13 End-of-semester Q&A — not recorded. Last class!	apr 14 Grades released for Memory Management Assignment

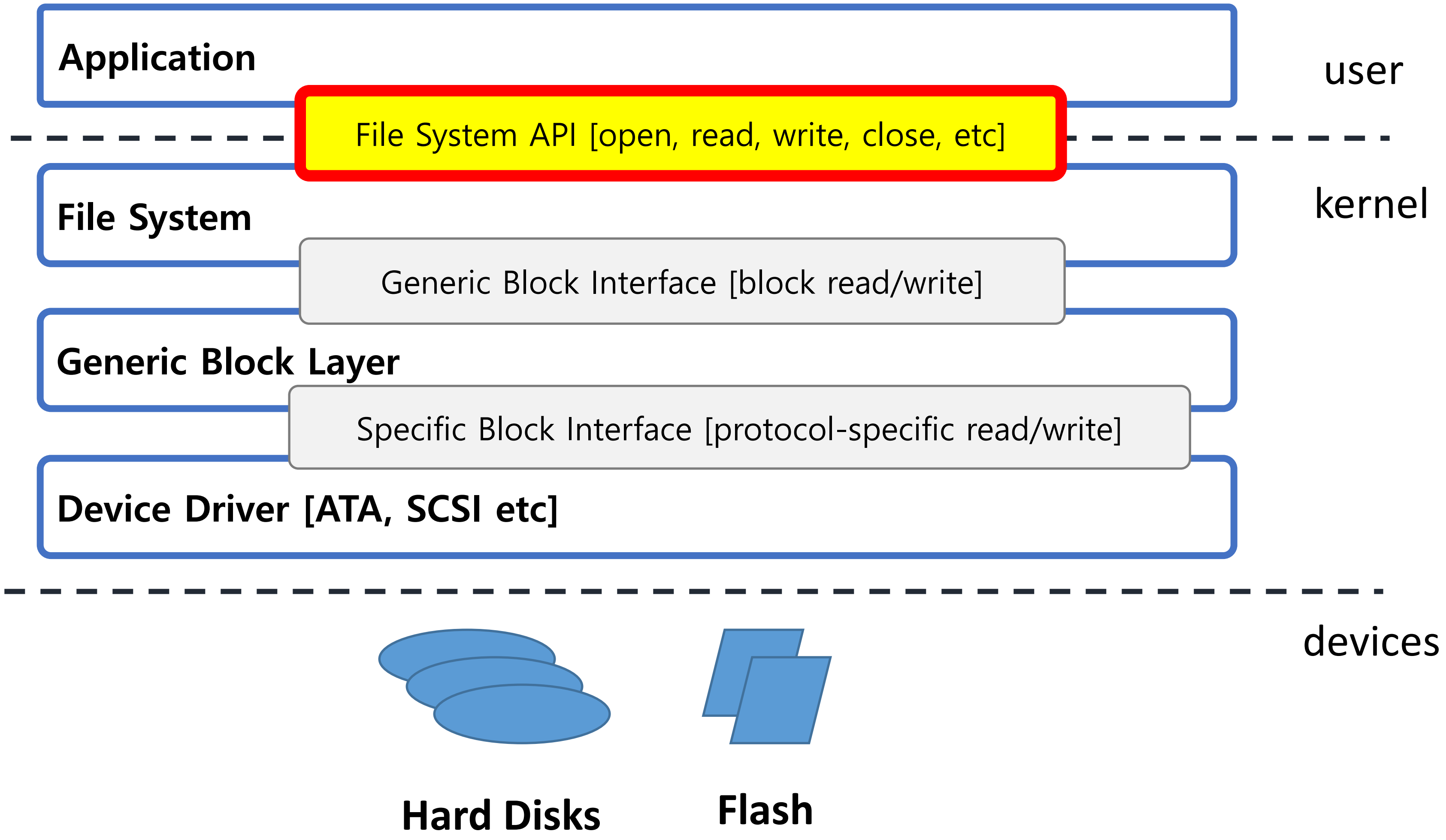
Recap: Overall Picture



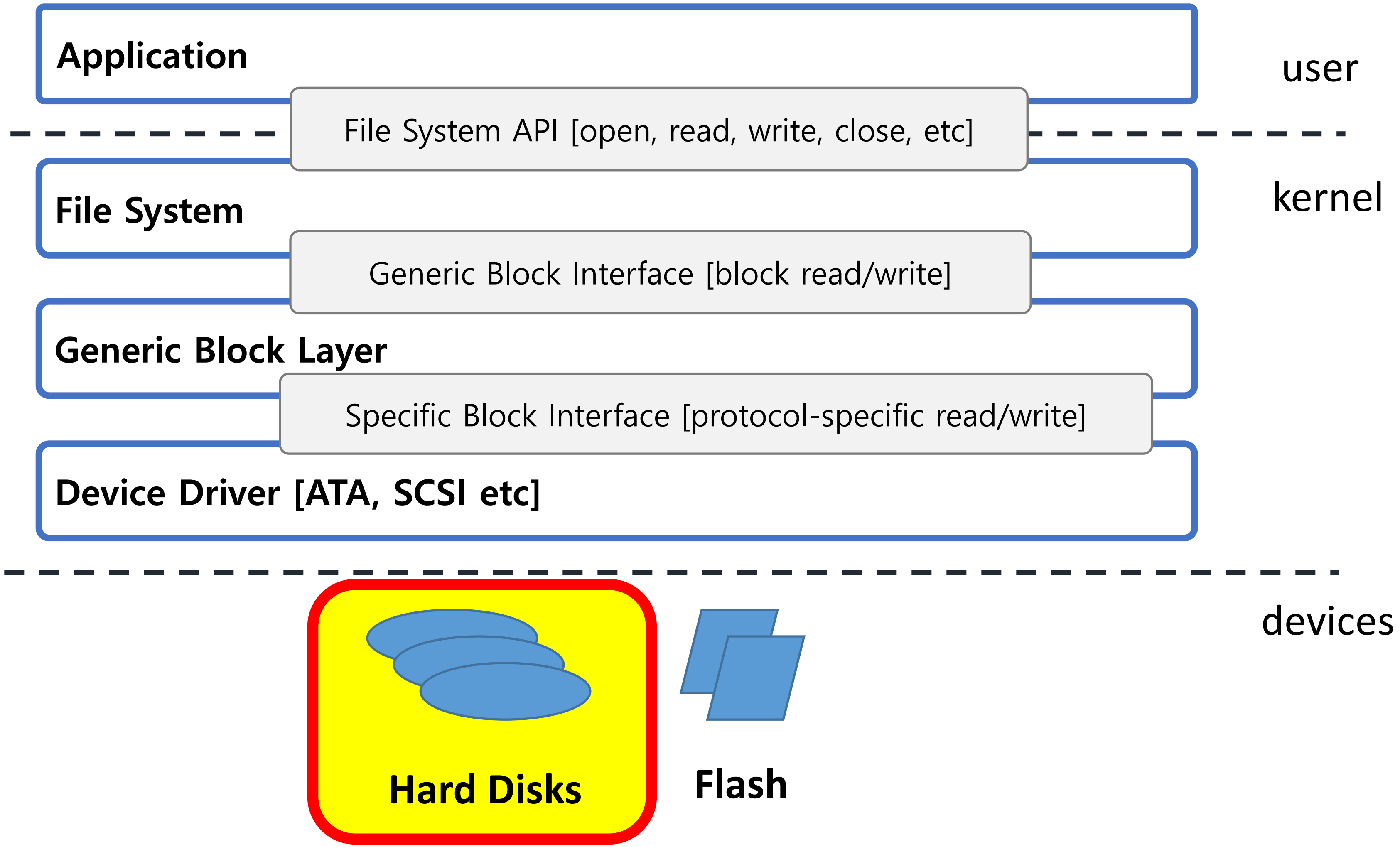
On Tuesday we talked about this part



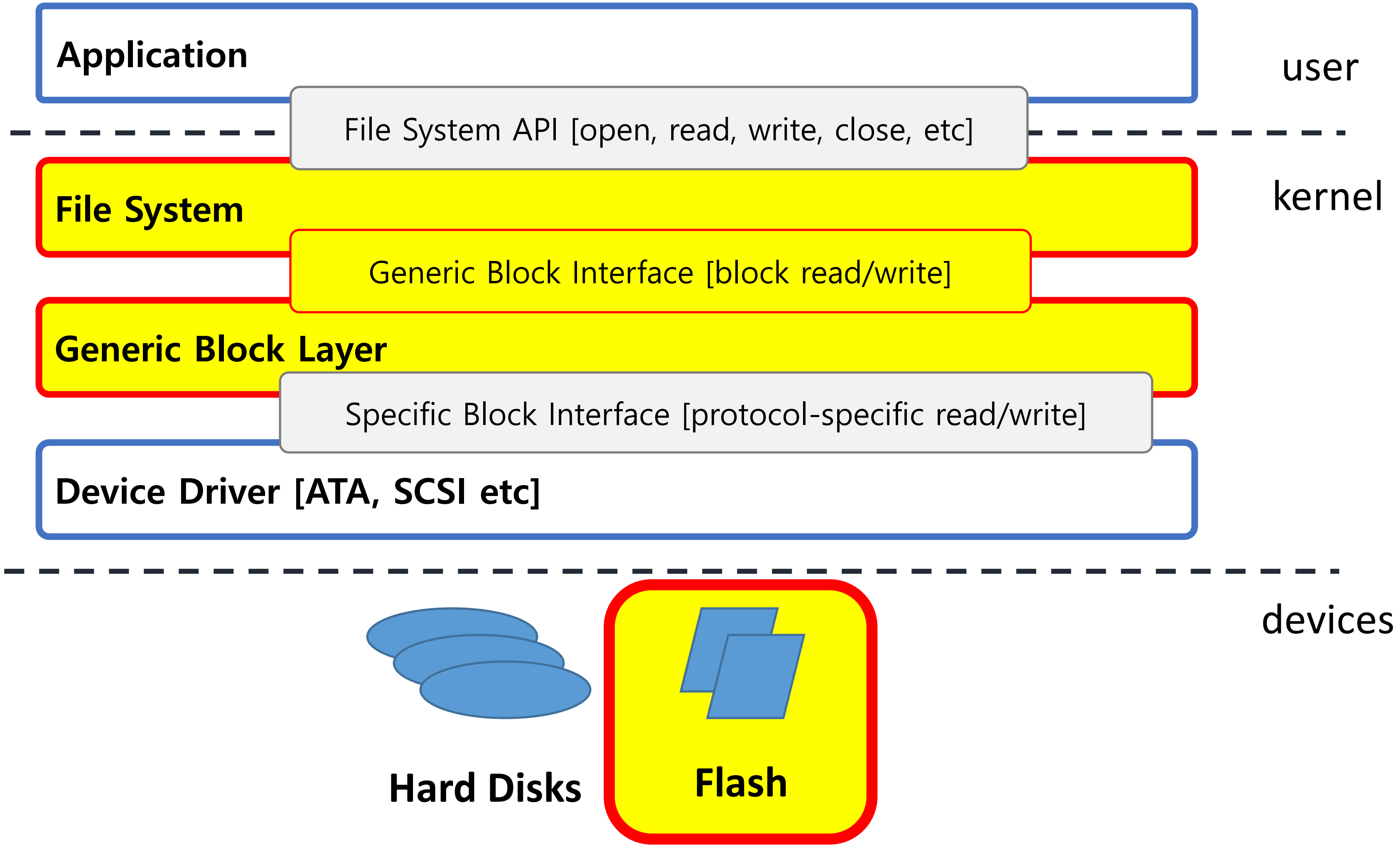
Then we talked about this part



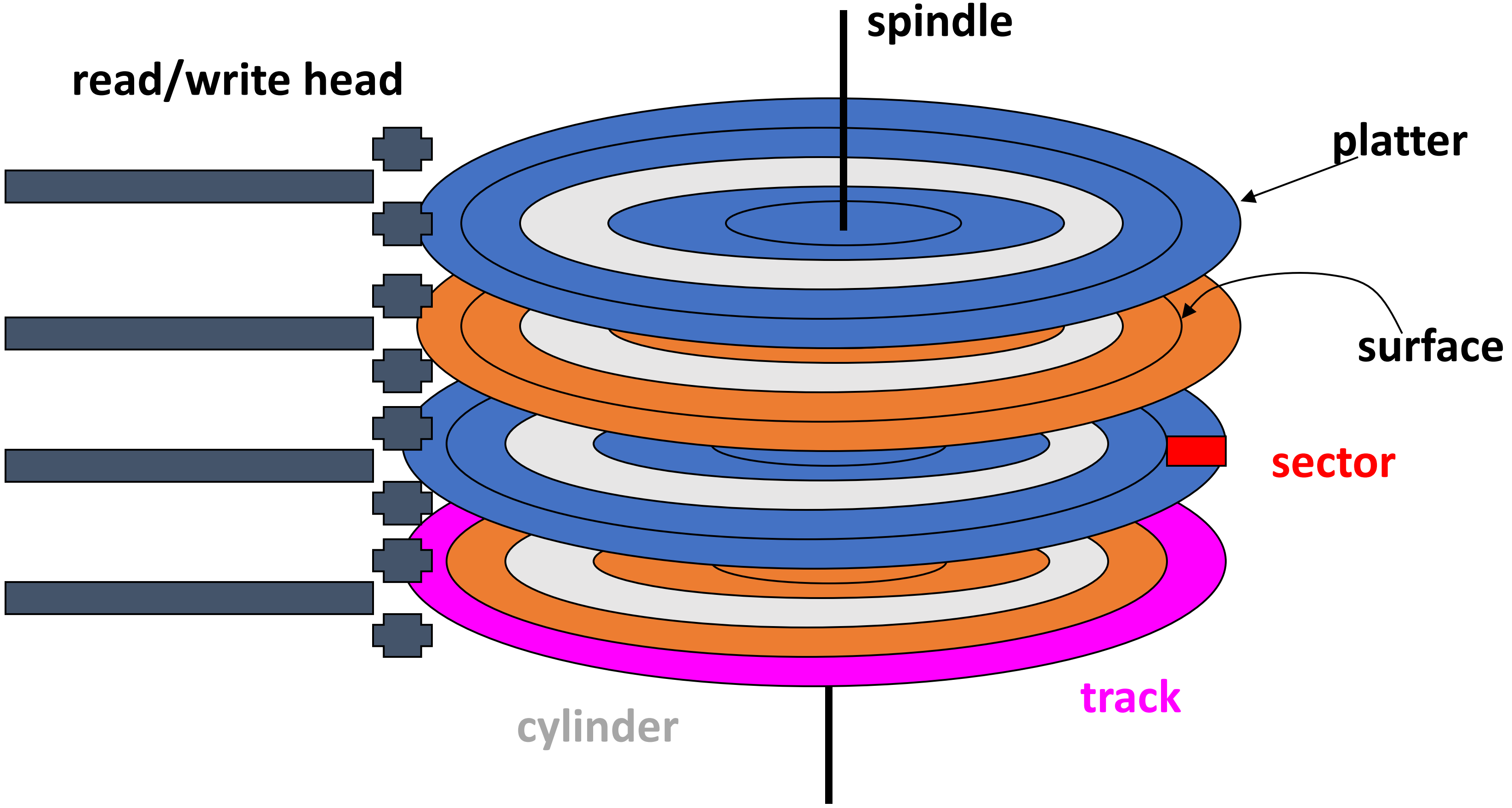
And this part. Today we continue with this part.



Next Weeks' Lectures



Recap: Disk Terminology



Disk Access Time

Component	Time
Head Selection	nanoseconds
Seek Time	**3-12 milliseconds**
Rotational Latency	**2-7 milliseconds**
Transfer Time	microseconds
Controller Overhead	< 1 millisecond

Disk Access Time

Seek time dominates

Component	Time
Head Selection	nanoseconds
Seek Time	**3-12 milliseconds**
Rotational Latency	**2-7 milliseconds**
Transfer Time	microseconds
Controller Overhead	< 1 millisecond

Note: Disk access time >> memory access time (nanoseconds)

Let’s Practice! Disk Access Time

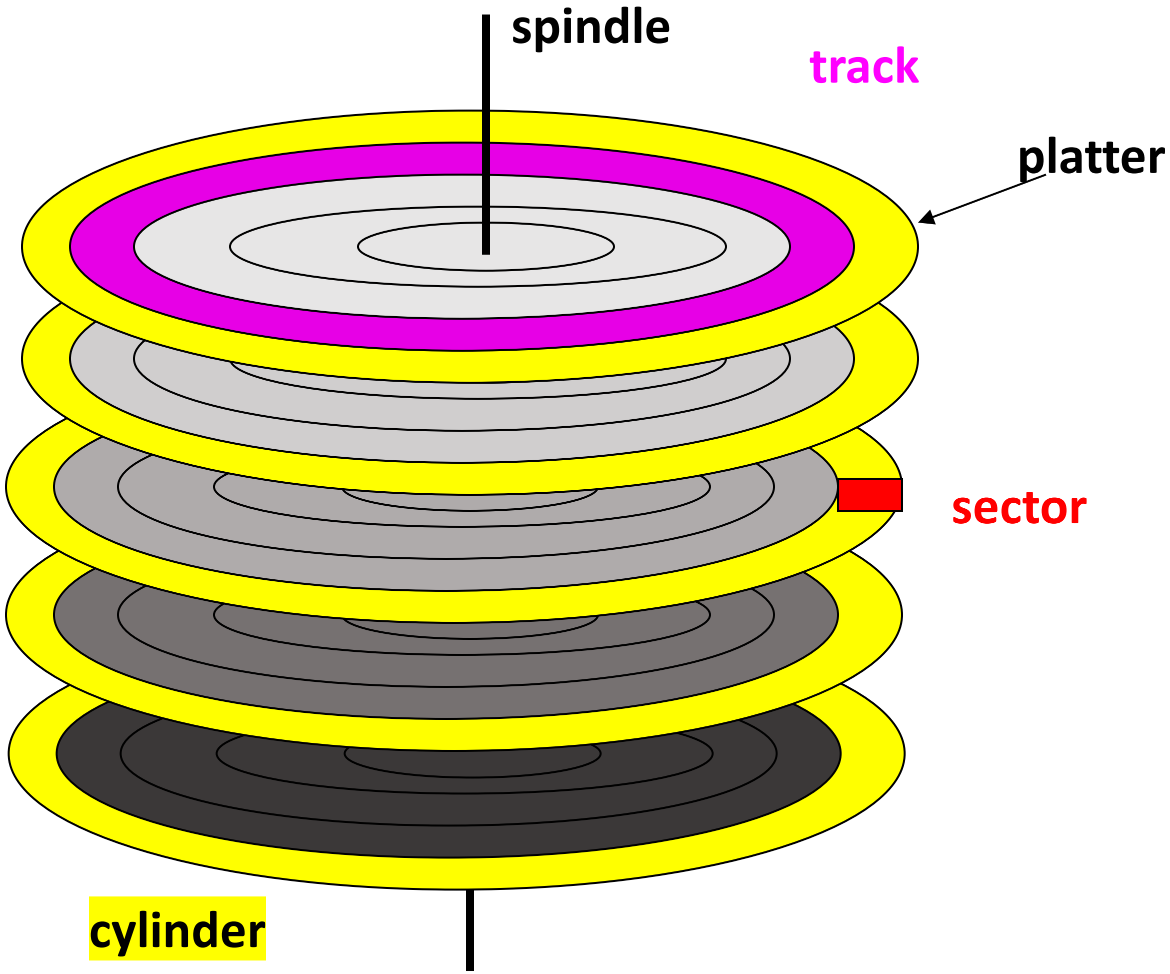
Consider a disk with a sector size of 512 bytes, 1,000 tracks per surface, 100 sectors per track, 5 double-sided platters and a block size of 2,048 bytes. Suppose that the average seek time is 10ms, the average rotational delay is 5 ms, and the transfer rate is 200 MB per second. Suppose that a file containing 1,000,000 records of 100 bytes each is to be stored on such a disk and that no record is allowed to span two blocks.

A. How many blocks are required to store the entire file?

B. If the file is arranged sequentially on disk, how many cylinders are needed?

C. What is the time required to read the file sequentially?

Let’s Practice! Disk Access Time



sector size of 512 bytes, 1,000 tracks per surface, 100 sectors per track, 5 double-sided platters and a block size of 2,048 bytes.

Let’s Practice! Disk Access Time

Consider a disk with a sector size of 512 bytes, 1,000 tracks per surface, 100 sectors per track, 5 double-sided platters and a block size of 2,048 bytes. Suppose that the average seek time is 10ms, the average rotational delay is 5 ms, and the transfer rate is 200 MB per second. Suppose that a file containing 1,000,000 records of 100 bytes each is to be stored on such a disk and that no record is allowed to span two blocks.

A. How many blocks are required to store the entire file?

Let’s Practice! Disk Access Time

Consider a disk with a sector size of 512 bytes, 1,000 tracks per surface, 100 sectors per track, 5 double-sided platters and a block size of 2,048 bytes. Suppose that the average seek time is 10ms, the average rotational delay is 5 ms, and the transfer rate is 200 MB per second. Suppose that a file containing 1,000,000 records of 100 bytes each is to be stored on such a disk and that **no record is allowed to span two blocks.**

A. How many blocks are required to store the entire file?

→ How many records fit in a block?

20 records (2048B can fully fit 20 records of 100B each)

→ 1,000,000 records are stored in 1,000,000 / 20 = **50,000 blocks**

Let’s Practice! Disk Access Time

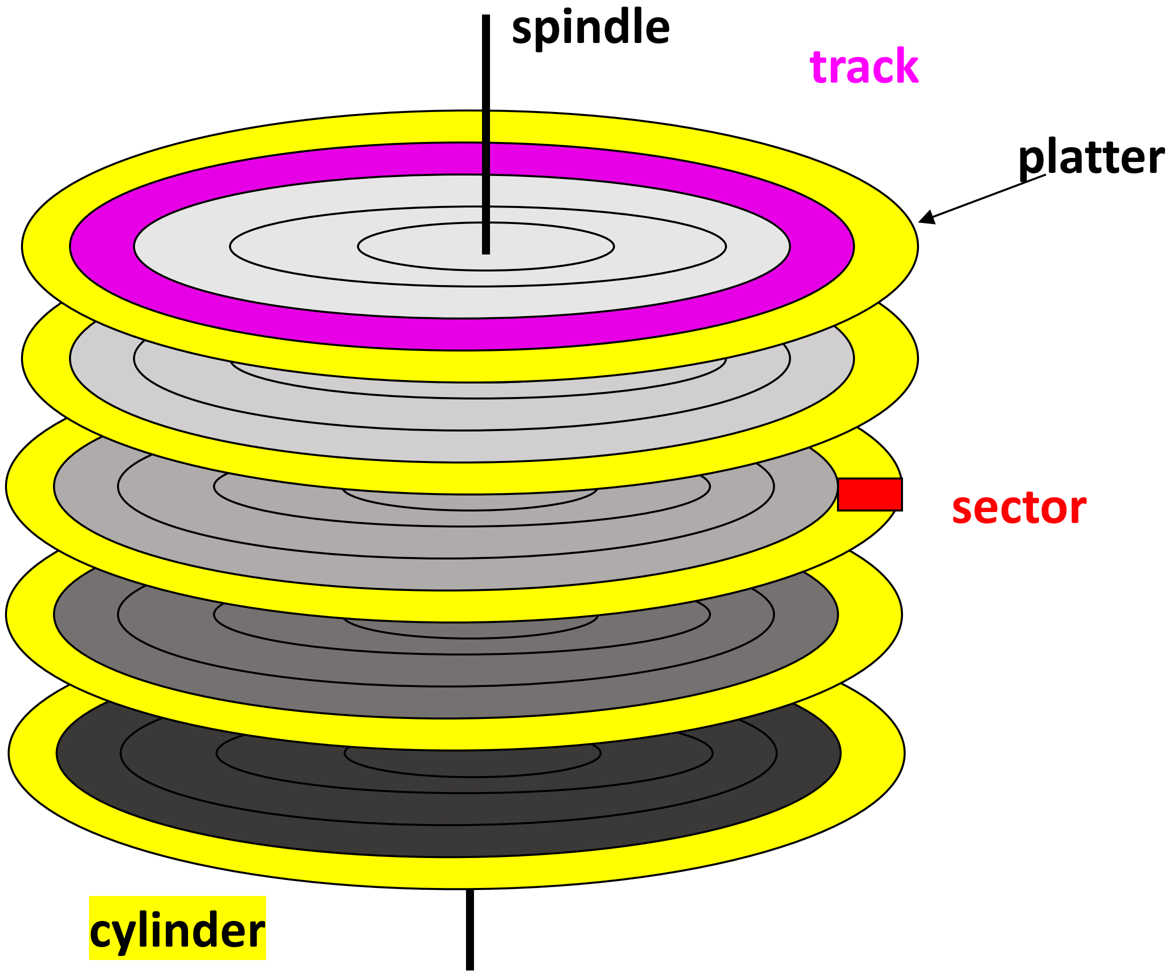
Consider a disk with a sector size of 512 bytes, 1,000 tracks per surface, 100 sectors per track, 5 double-sided platters and a block size of 2,048 bytes. Suppose that the average seek time is 10ms, the average rotational delay is 5 ms, and the transfer rate is 200 MB per second. Suppose that a file containing 100,000 records of 100 bytes each is to be stored on such a disk and that no record is allowed to span two blocks.

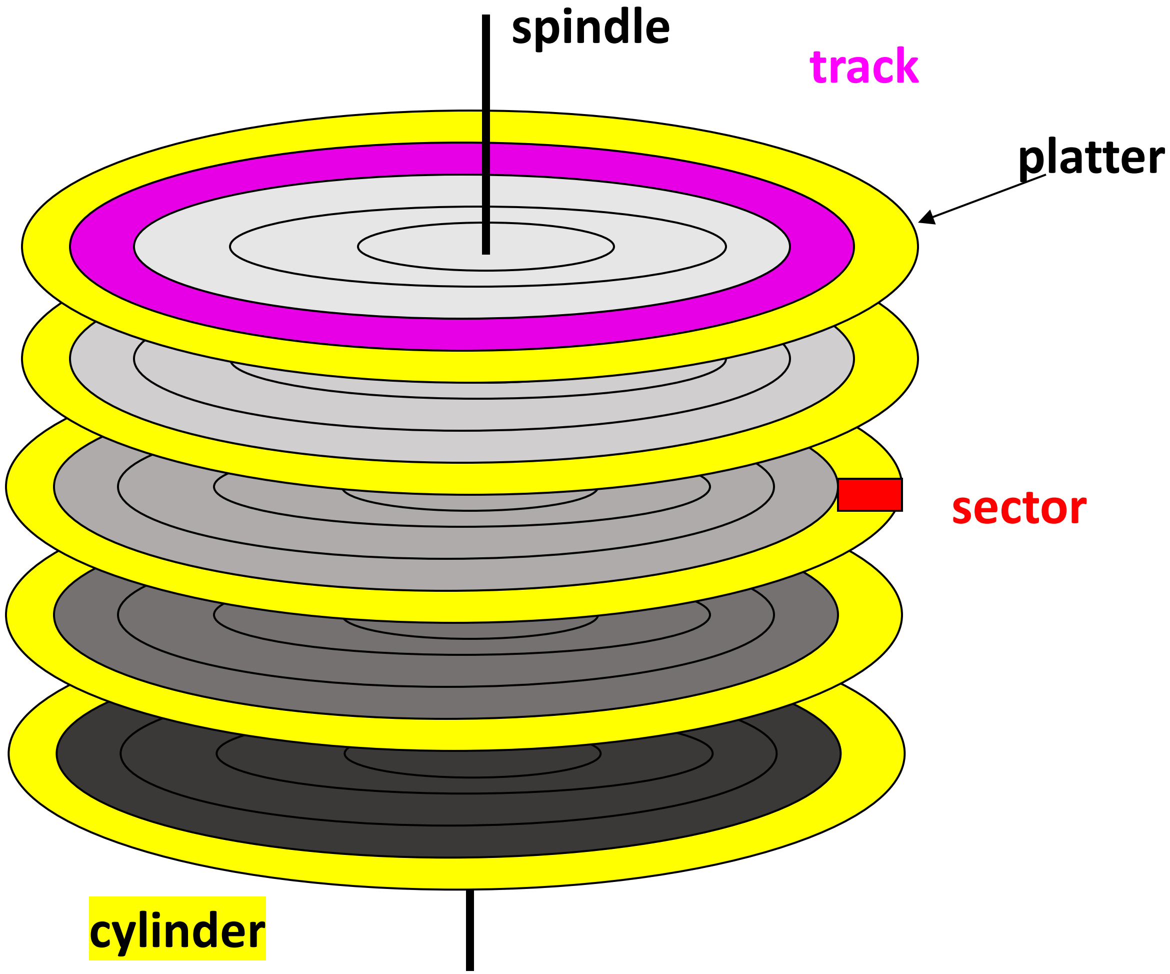
A. How many blocks are required to store the entire file? → **50,000 blocks**

B. If the file is arranged sequentially on disk, how many cylinders are needed?

2.B. If the file is arranged **sequentially on disk**, how many cylinders are needed?

Need to place 50,000 blocks (computed in 2.A).

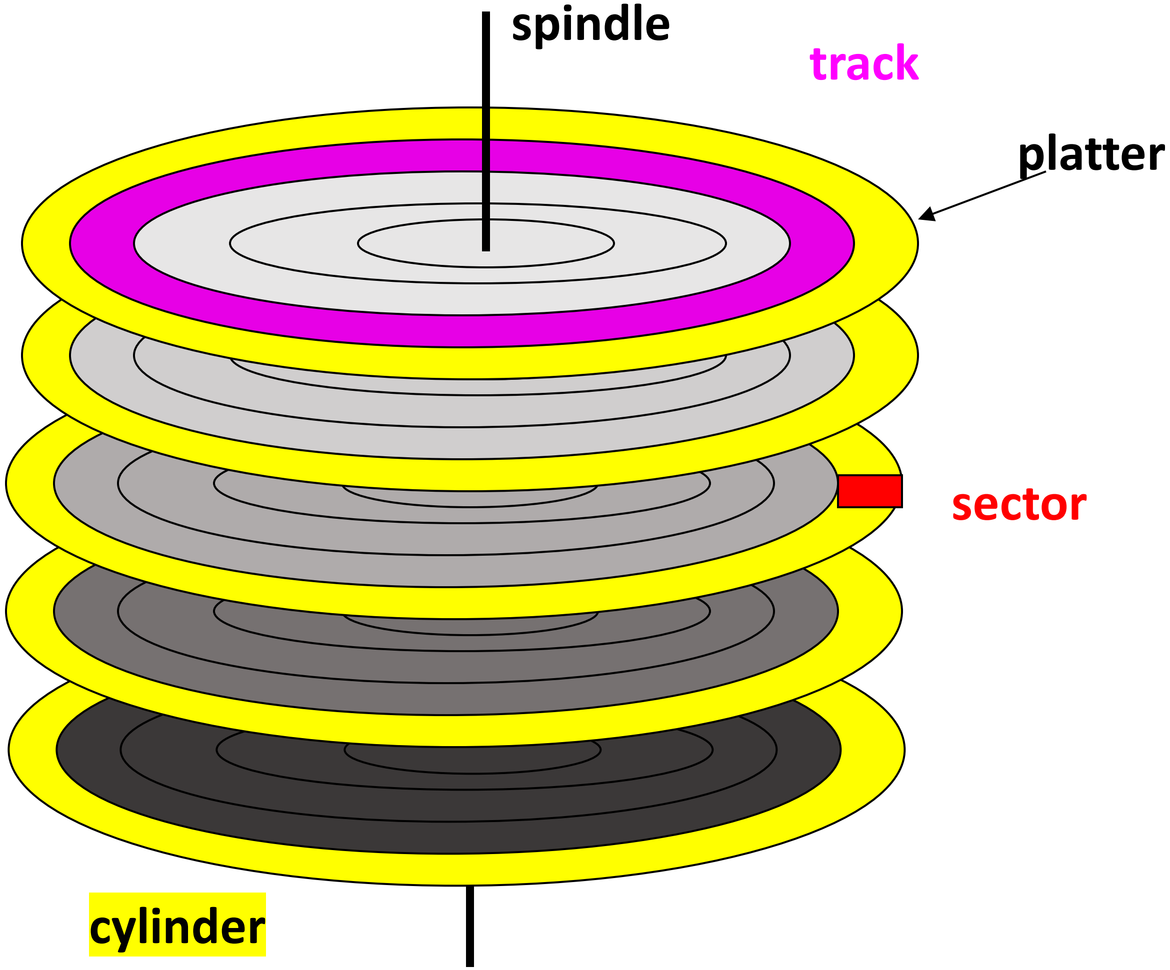




2.B. If the file is arranged **sequentially on disk**, how many cylinders are needed?

Need to place 50,000 blocks (computed in 2.A).

→ **How de we arrange blocks for sequential access?**



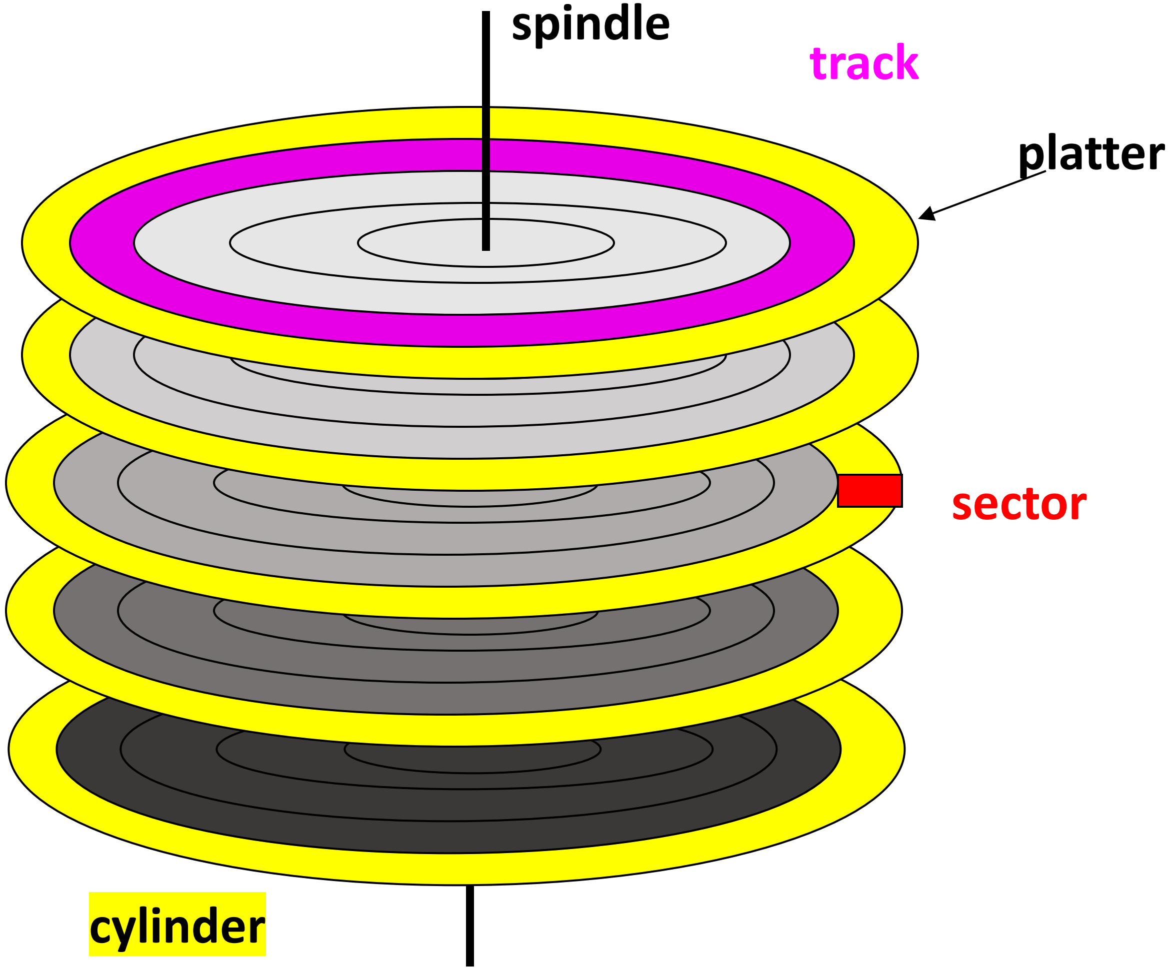
2.B. If the file is arranged **sequentially on disk**, how many cylinders are needed?

Need to place 50,000 blocks (computed in 2.A).

→ **How de we arrange blocks for sequential access?**

Next block concept:

- blocks on same track, followed by
- blocks on same cylinder, followed by
- blocks on adjacent cylinder



2.B. If the file is arranged **sequentially on disk**, how many cylinders are needed?

Need to place 50,000 blocks (computed in 2.A).

→ **How de we arrange blocks for sequential access?**

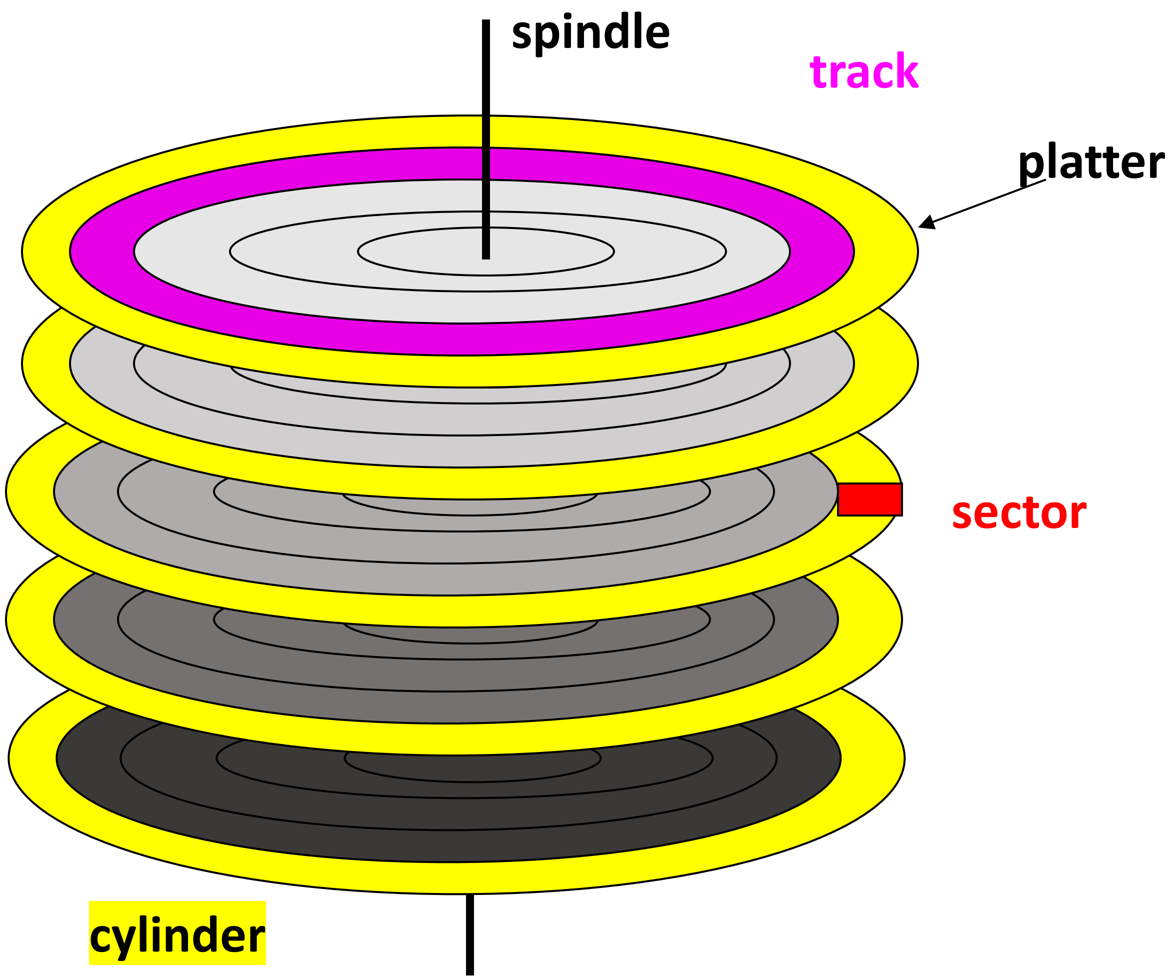
Next block concept:

- blocks on same track, followed by
- blocks on same cylinder, followed by
- blocks on adjacent cylinder

→ **How many blocks can we place in a track?**

→ **How many blocks can we place in a cylinder?**

→ **How many cylinders do we need?**



2.B. If the file is arranged **sequentially on disk**, how many cylinders are needed?

Need to place 50,000 blocks (computed in 2.A).

→ **How de we arrange blocks for sequential access?**

Next block concept:

- blocks on same track, followed by
- blocks on same cylinder, followed by
- blocks on adjacent cylinder

→ **How many blocks can we place in a track?**

Sector size: 512B, 100 sectors per track

→ **25 blocks per track**

→ **How many blocks can we place in a cylinder?**

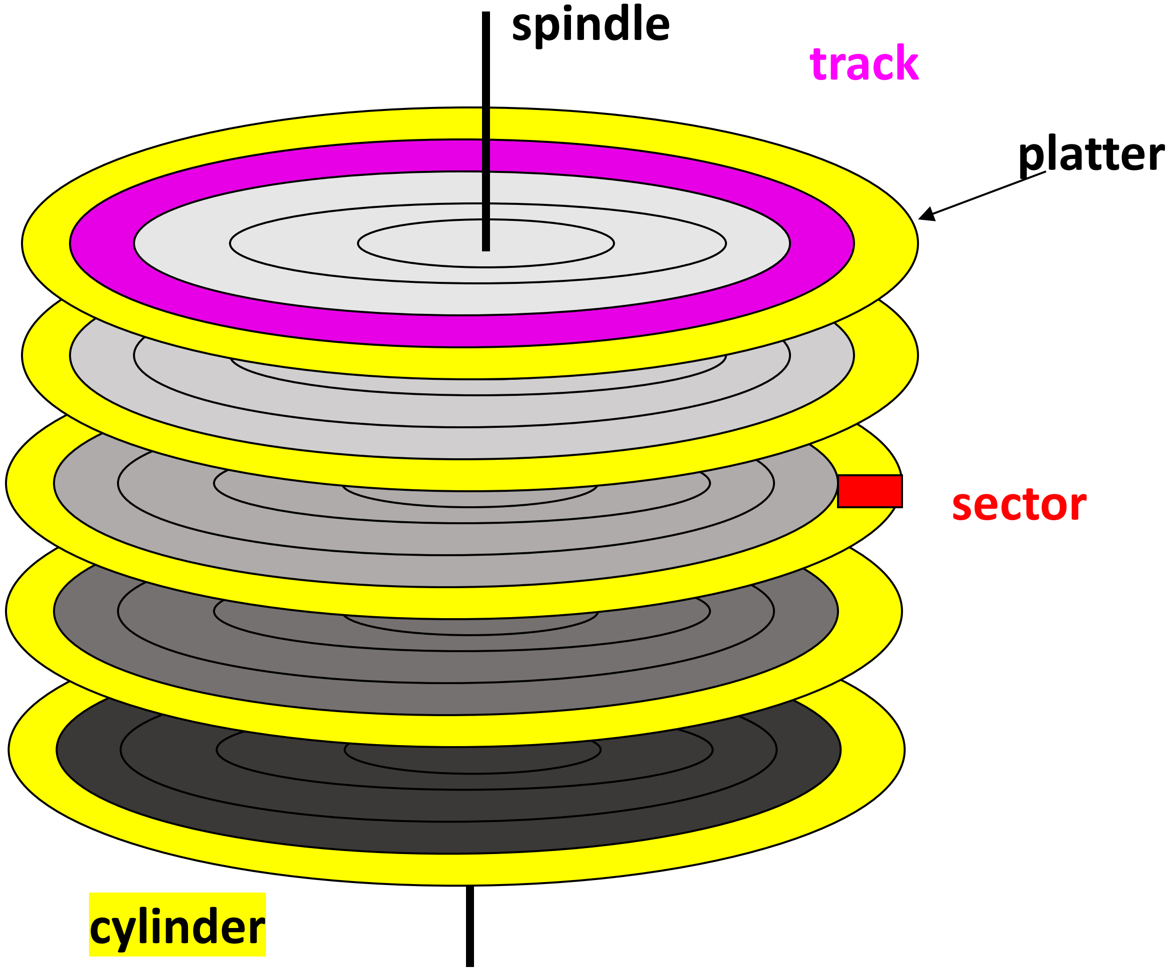
25 x 2 x 5 = **250 blocks per cylinder**

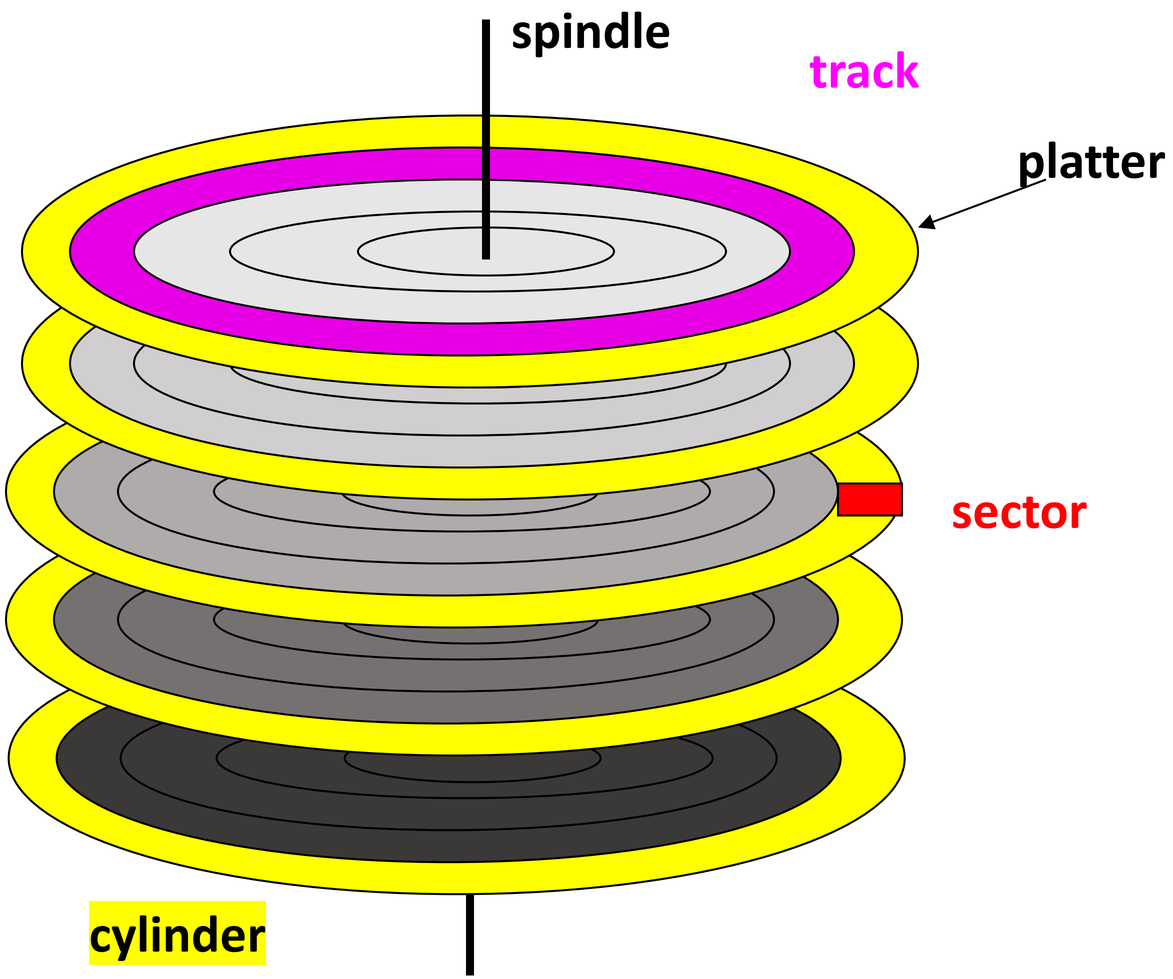
→ **How many cylinders do we need?**

50,000 / 250 = **200 cylinders**

2.C. What is the time required to read the file sequentially?

Need to read 50,000 blocks; each block has size 2048B.





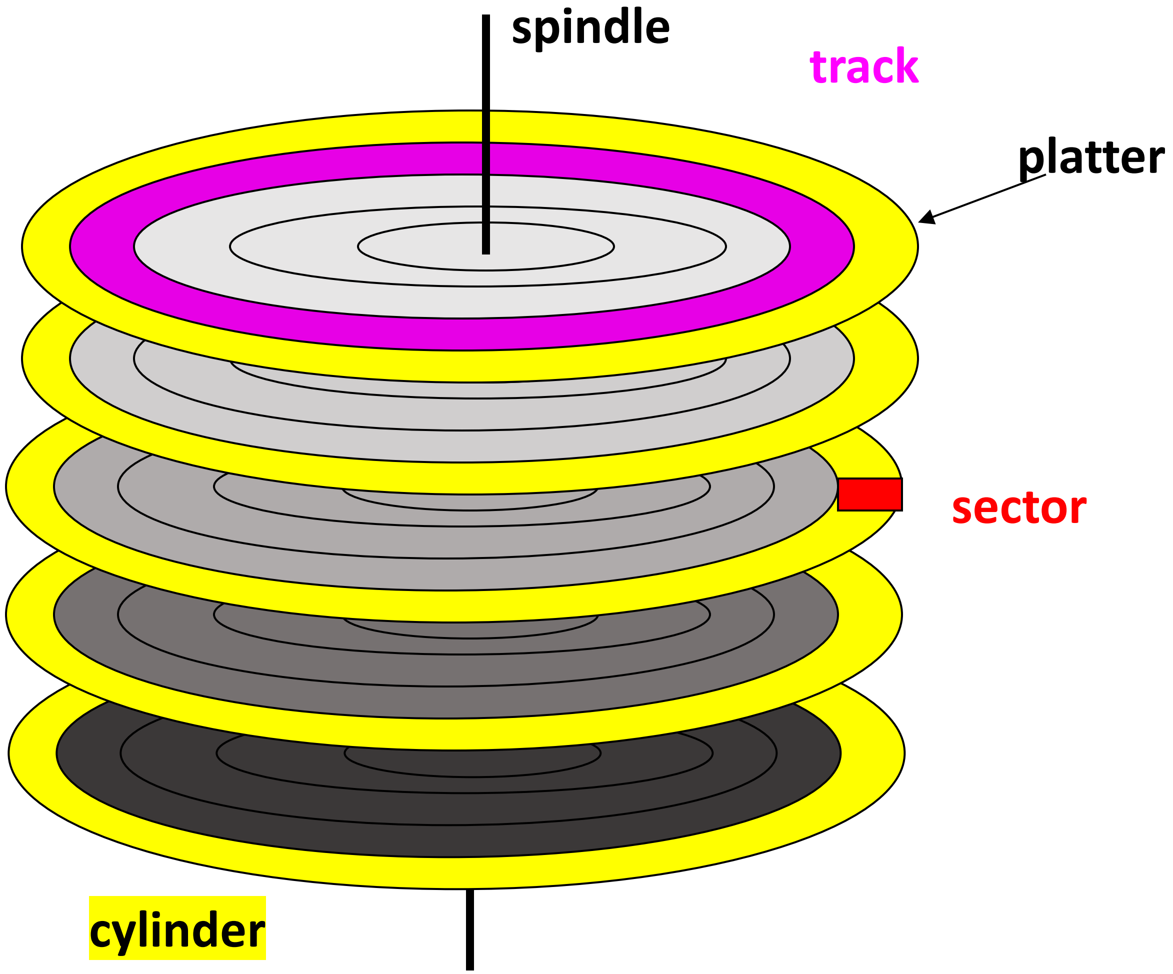
2.C. What is the time required to read the file sequentially?

Need to read 50,000 blocks; each block has size 2048B.

Seek time = 10ms

Avg rotational delay = 5ms

Transfer rate = 200 MB / second.



2.C. What is the time required to read the file sequentially?

Need to read 50,000 blocks; each block has size 2048B.

Seek time = 10ms

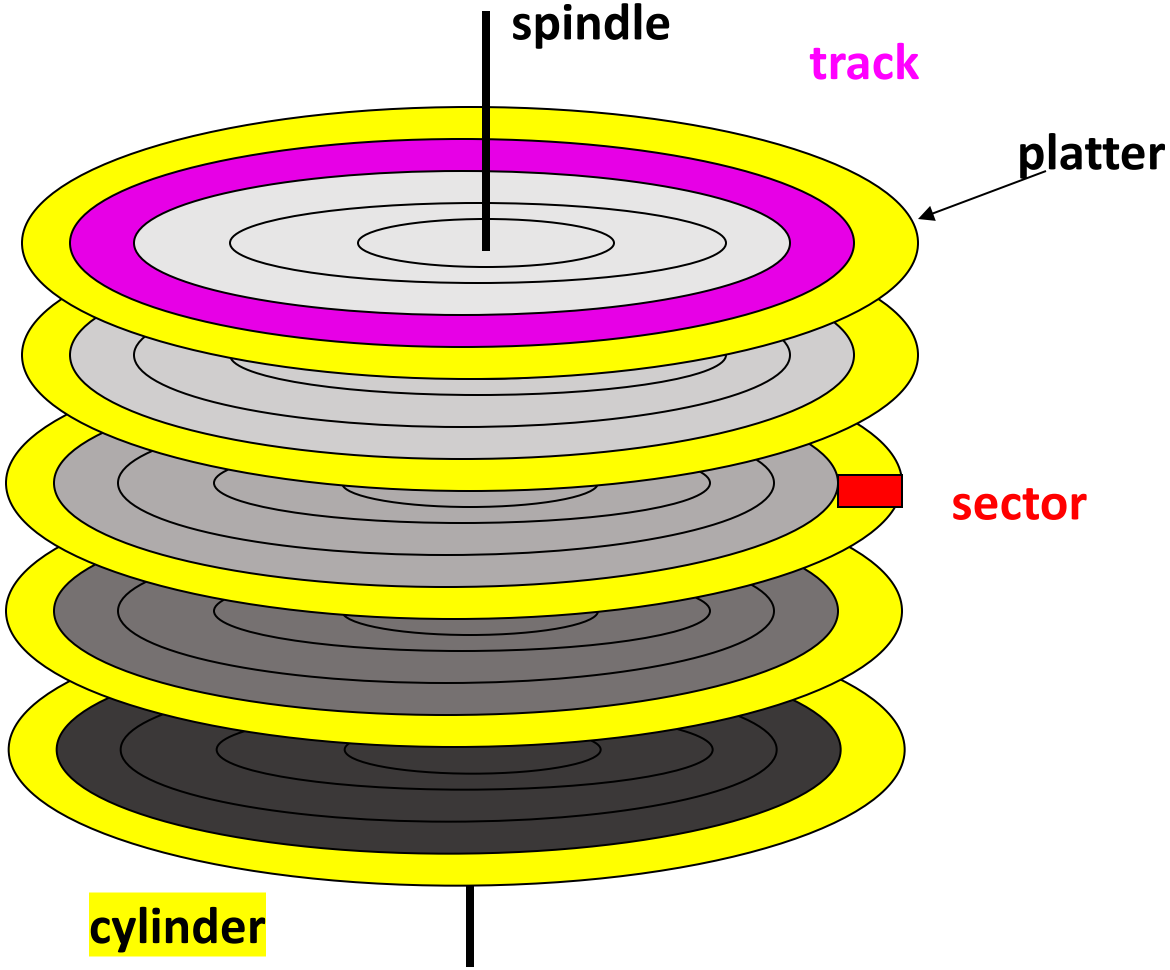
Avg rotational delay = 5ms

Transfer rate = 200 MB / second.

→ **What are the components of disk access?**

→ **How long do each of them take?**

→ **What is the total read time?**



2.C. What is the time required to read the file sequentially?

Need to read 50,000 blocks; each block has size 2048B.

Seek time = 10ms

Avg rotational delay = 5ms

Transfer rate = 200 MB / second.

→ **What are the components of disk access?**

Head selection (negligible)

Seek (one time cost because sequential access)

Rotational Delay (one time cost because sequential access)

Transfer time

Controller overhead (negligible)

→ **What is the transfer time?**

(5 * 5^4 * 2^4 * 2^11 B) / (200 * 2^20 B/s) = 0.488 s

= 488 ms

→ **What is the total read time?**

10 + 5 + 488 = **503 ms**

Optimizing disk access

Remember:

- Disk access time >> memory access time
- If we go to disk, seek time dominates

Optimize Disk Access

Rule 1:

Do not access disk, use a cache

File System Cache (Buffer Cache)

What?

- Keep recently accessed blocks in memory

Why?

- Reduce latency
- Reduce disk load

How?

- Reserve kernel memory for cache
- Cache entries: file blocks (of block size)

Read with a Cache

If in cache

- Return data from cache

If not

- Find free cache slot
- Initiate disk read
- When disk read completes, return data

Write with a Cache

Always write in cache

How does it get to disk?

- **Write-through**
- **Write-behind**

Write with a Cache

Always write in cache

How does it get to disk?

- **Write-through**
- **Write-behind**

Write-through

1. Write to cache
2. Write to disk
3. Return to user

Write with a Cache

Always write in cache

How does it get to disk?

- **Write-through**
- **Write-behind**

Write-through

1. Write to cache
2. Write to disk
3. Return to user

Write-behind

1. Write to cache
2. Return to user
3. Later: write to disk

Write-Through vs. Write-Behind

Response time:

- **Write-behind** is (much) better

Disk load:

- **Write-behind** is (much) better
- Much data overwritten before it gets to disk

Crash:

- **Write-through** is much better
- No “window of vulnerability”

Write-Through vs. Write-Behind

Response time:

- **Write-behind** is (much) better

Disk load:

- **Write-behind** is (much) better
- Much data overwritten before it gets to disk

Crash:

- **Write-through** is much better
- No “window of vulnerability”

In practice:

- Write-behind
- Periodic cache flush
- User primitive to flush data

Optimize Disk Access

Rule 2:

Do not wait for disk, read ahead

- Also called prefetching
- Only for sequential access

Read-Ahead

What?

- User request for block i of a file
- Also read block i+1 from disk

Why?

- No disk I/O on (expected) user access to block i+1

How?

- Put block i+1 in the buffer cache

→**Remember:** Pre-paging uses read-ahead if neighboring virtual pages are also neighbors in physical memory.

Read-Ahead

- **Works for sequential access**

- Most access is sequential
- In Linux it is the default

Caveat about Read-Ahead

- Does not reduce number of disk I/Os
- In fact, could increase them (if not sequential)

- In practice, very often a win
- Linux always reads one block ahead

Optimize Disk Access

Rule 3:

Minimize seeks

2 Approaches
- Clever disk allocation
- Clever scheduling

Clever Disk Allocation

Idea:

- Locate related data (same file) on same cylinder
- Allocate “related” blocks “together”

“together”

- On the same cylinder
- On a nearby cylinder

“related”

- Consecutive blocks in the same file
- Sequential access

Disk Scheduling

Idea: Reorder requests to seek as little as possible

Different disk scheduling policies:

- FCFS – First-Come-First-Served
- SSTF – Shortest-Seek-Time-First
- SCAN
- C-SCAN
- LOOK
- C-LOOK

Disk Scheduling Illustration

Initial position of the **head = cylinder 53**

Queue of requests:

98, 193, 37, 122, 14, 124, 65, 67

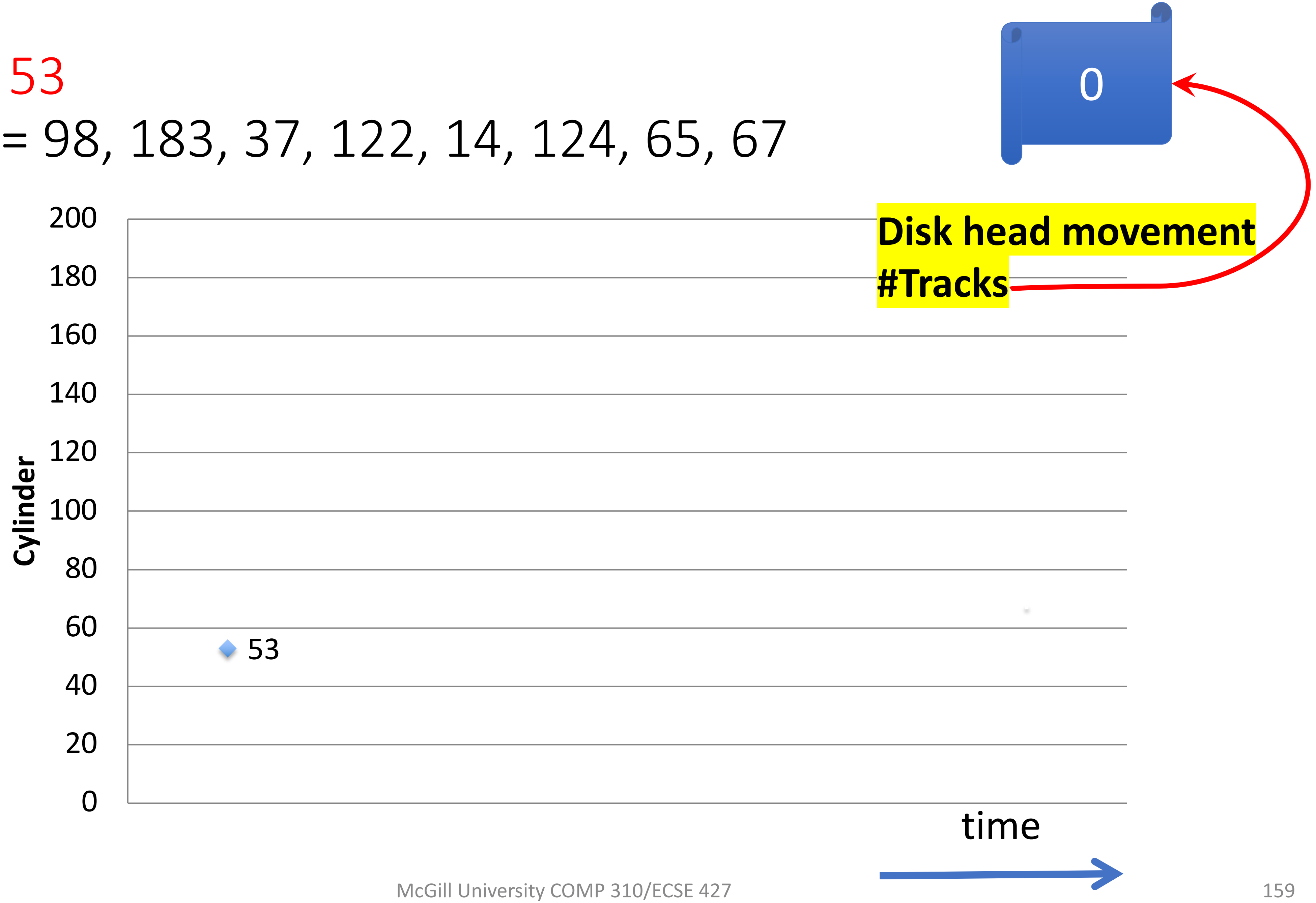
First Come, First Served (FCFS)

Serve **next request** in the queue

Head = 53

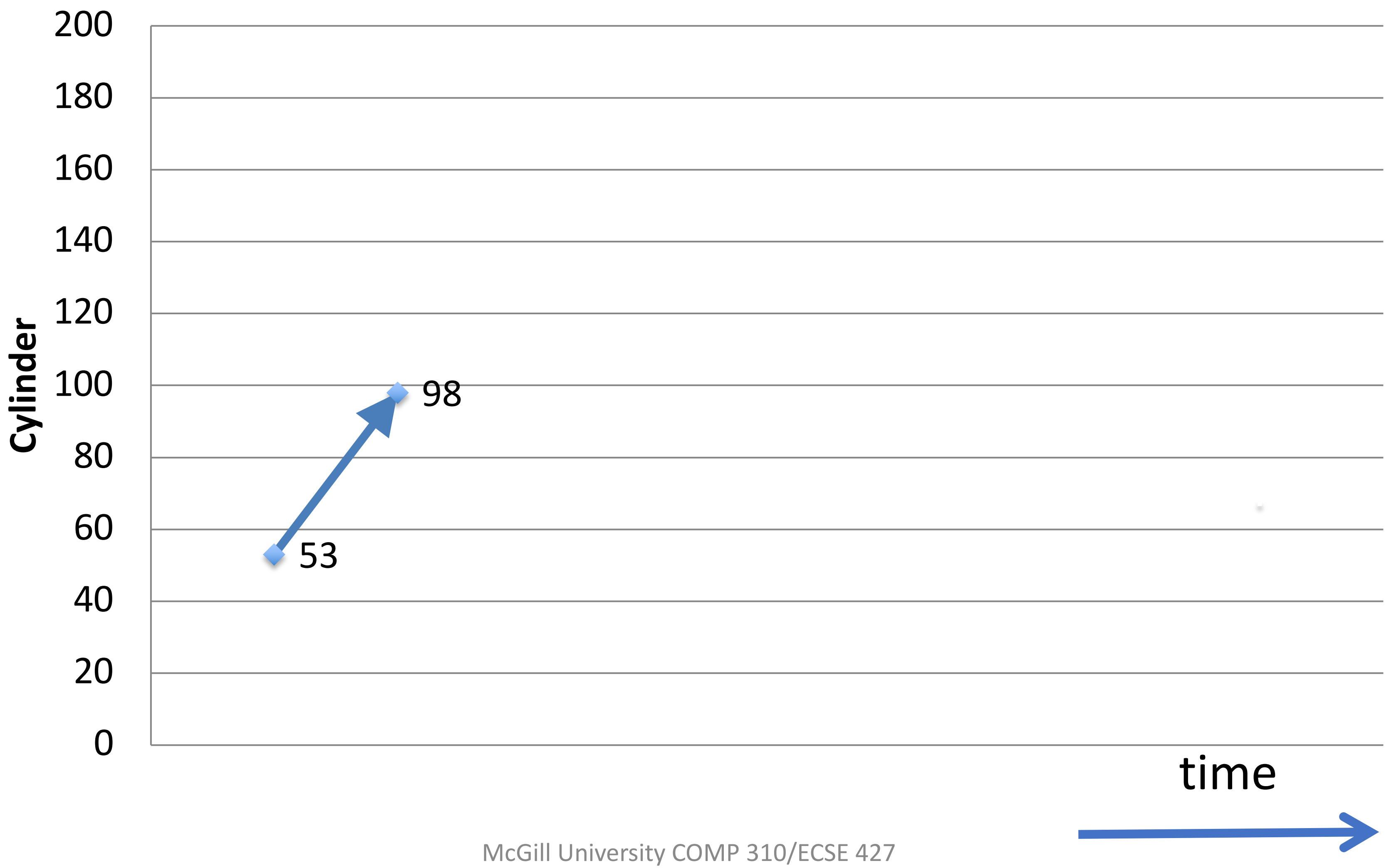
Queue = 98, 183, 37, 122, 14, 124, 65, 67

FCFS



Head = 53
Queue = 98, 183, 37, 122, 14, 124, 65, 67

FCFS

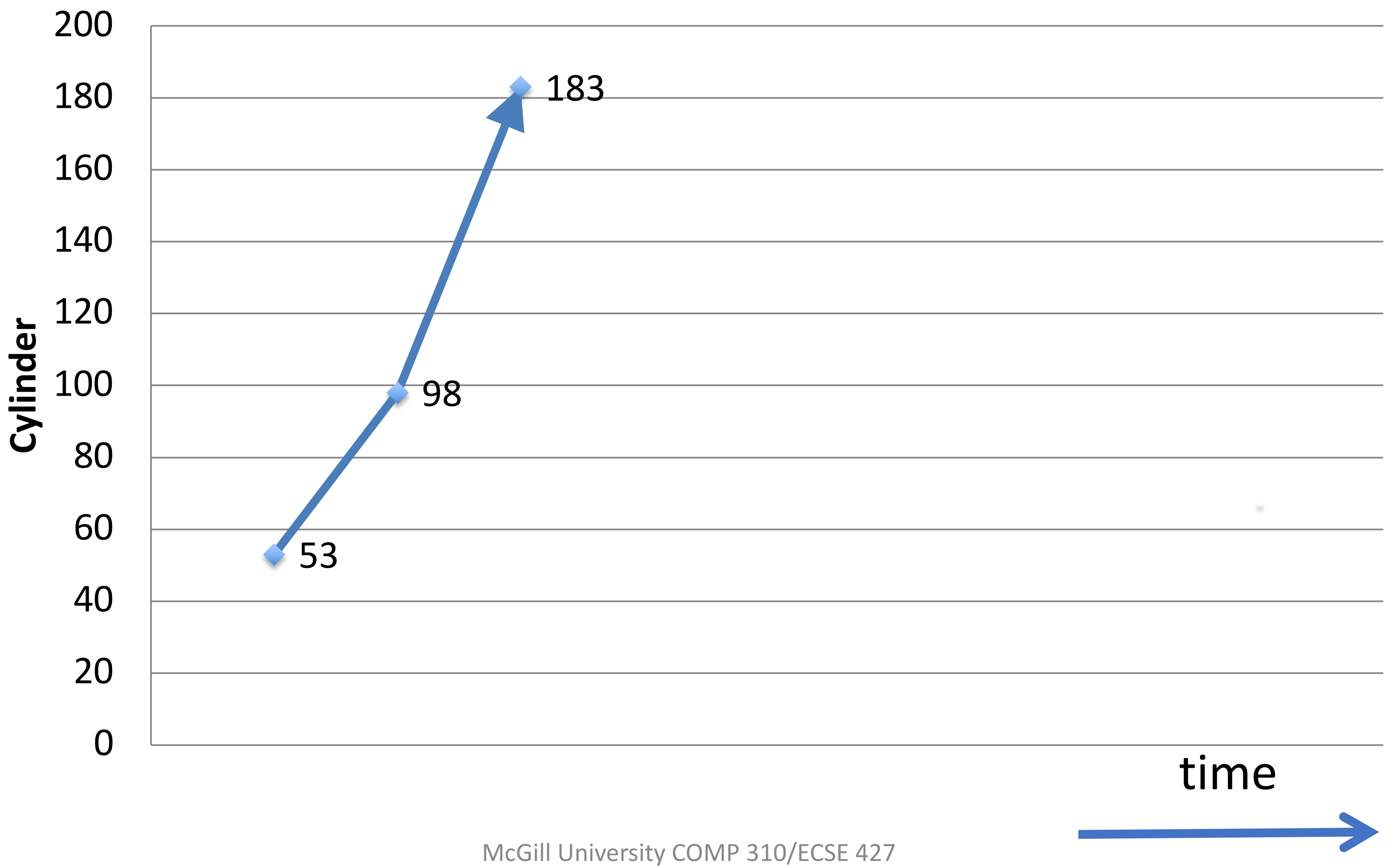


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

130

FCFS

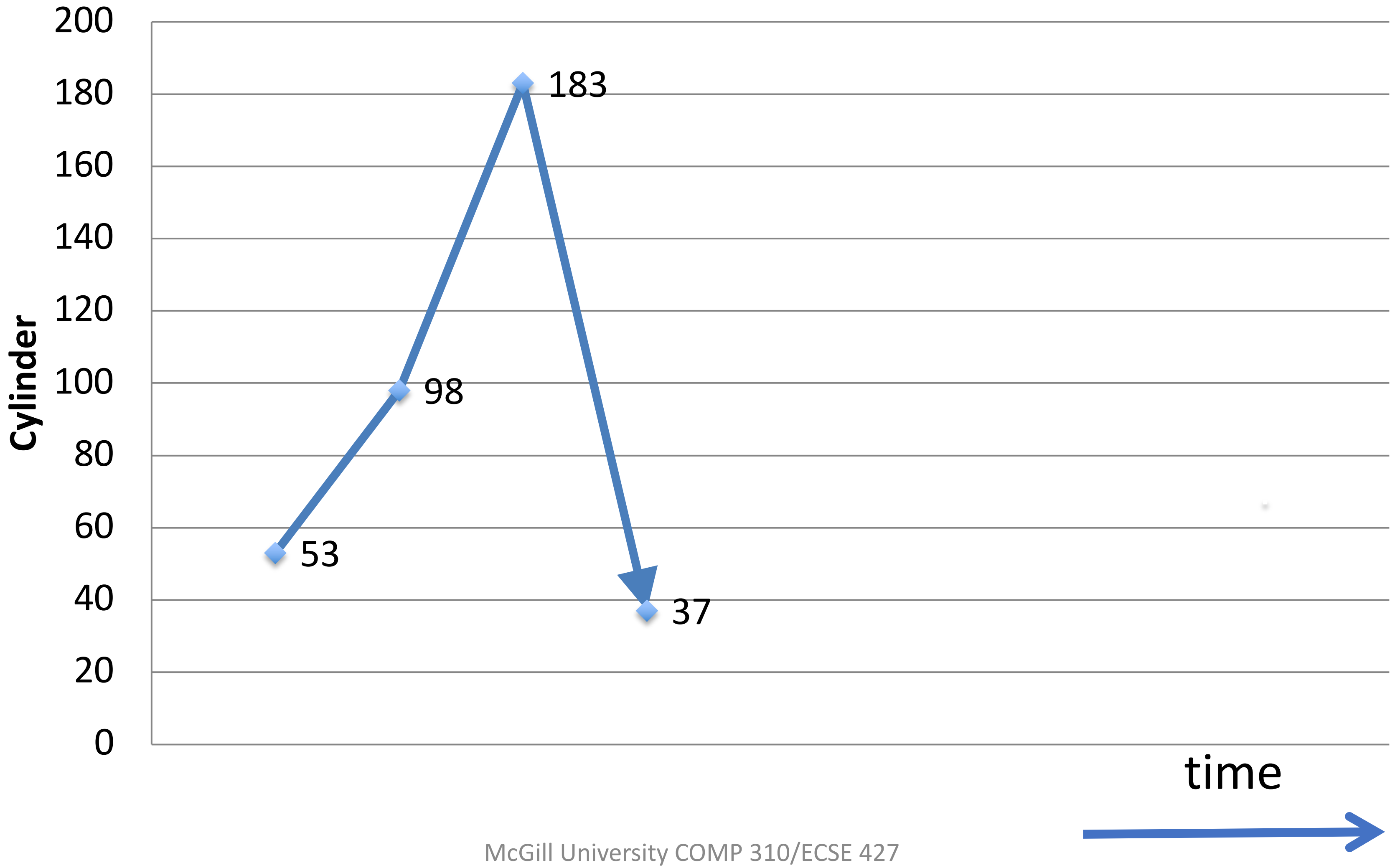


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

276

FCFS

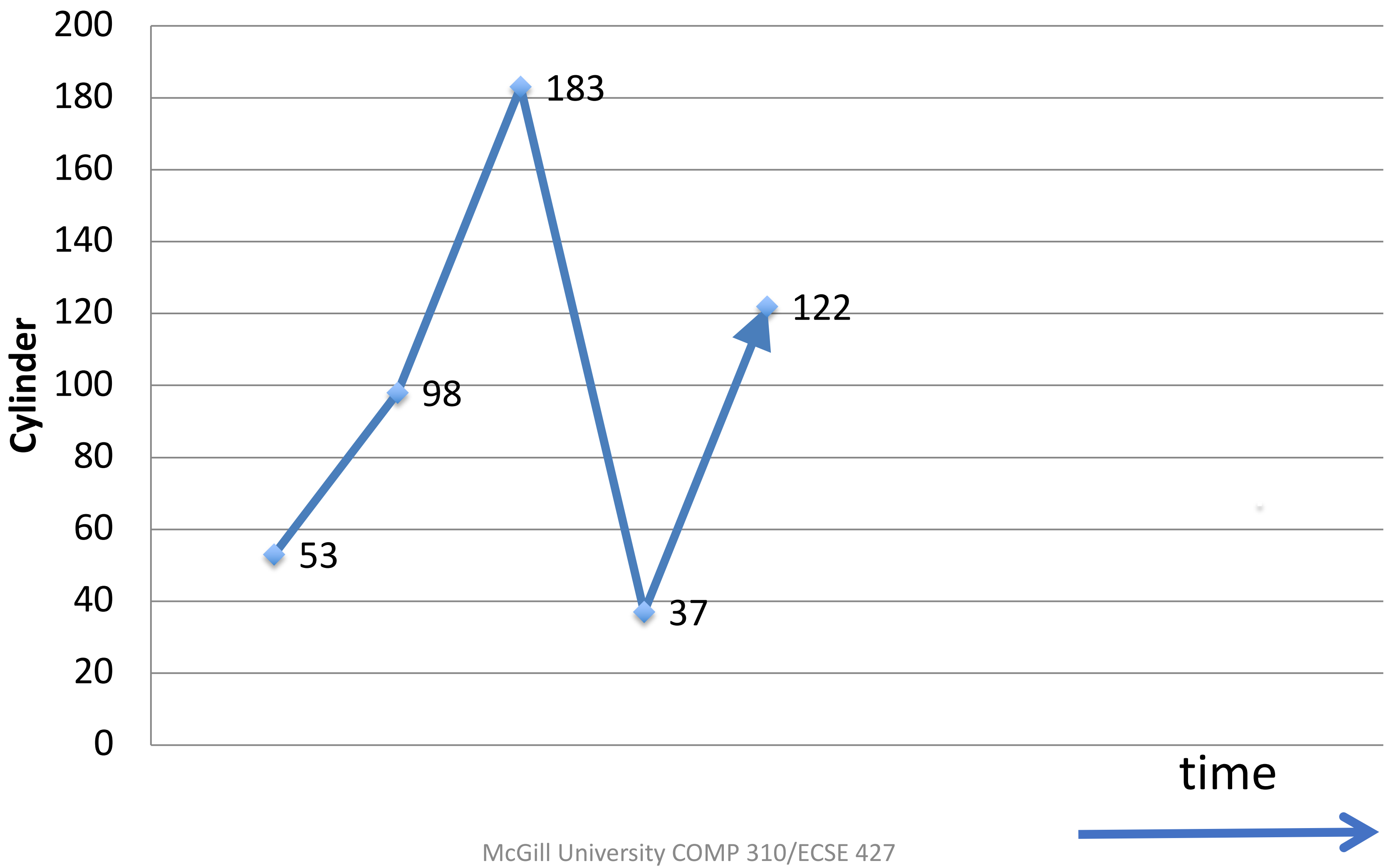


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

361

FCFS

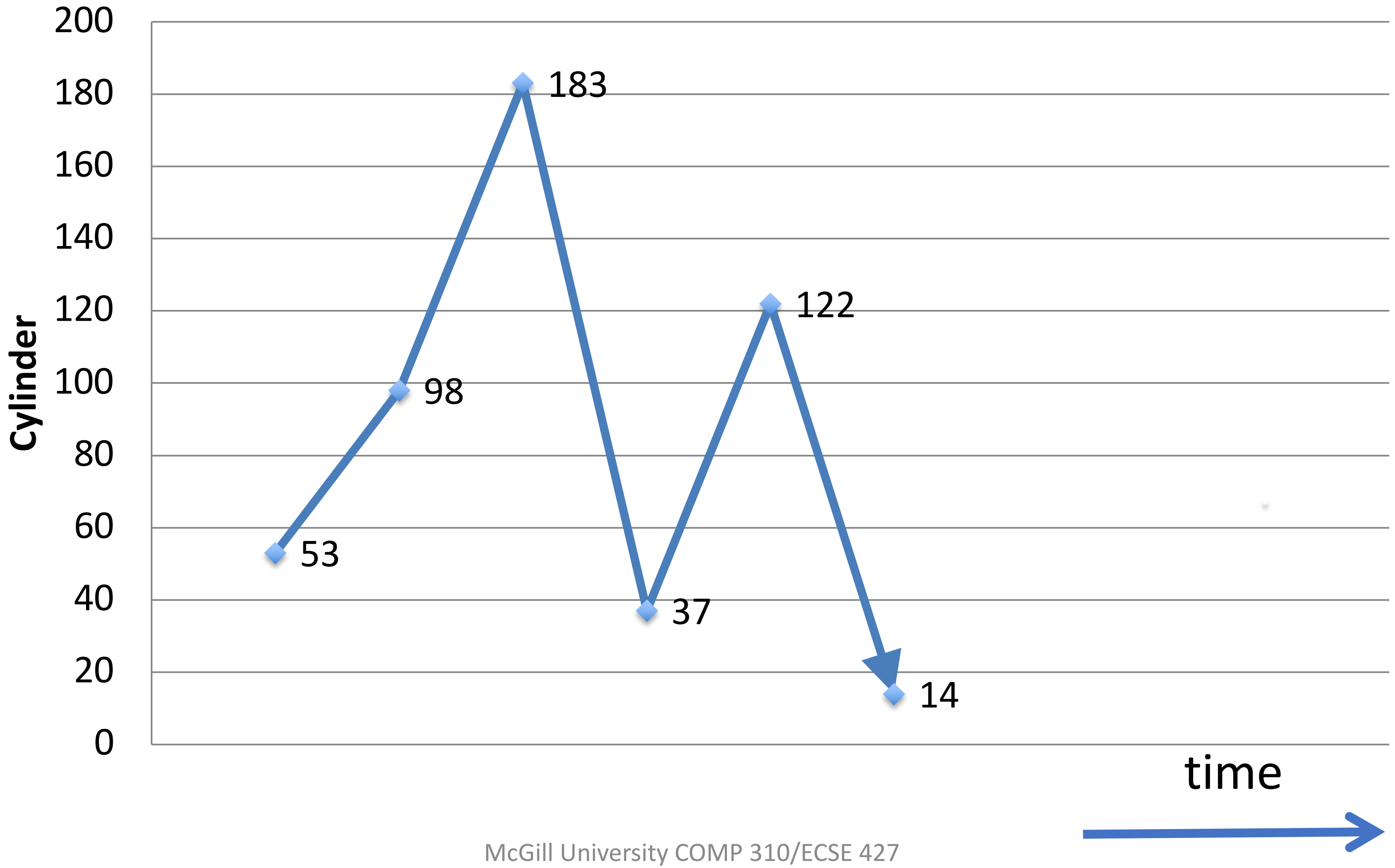


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

469

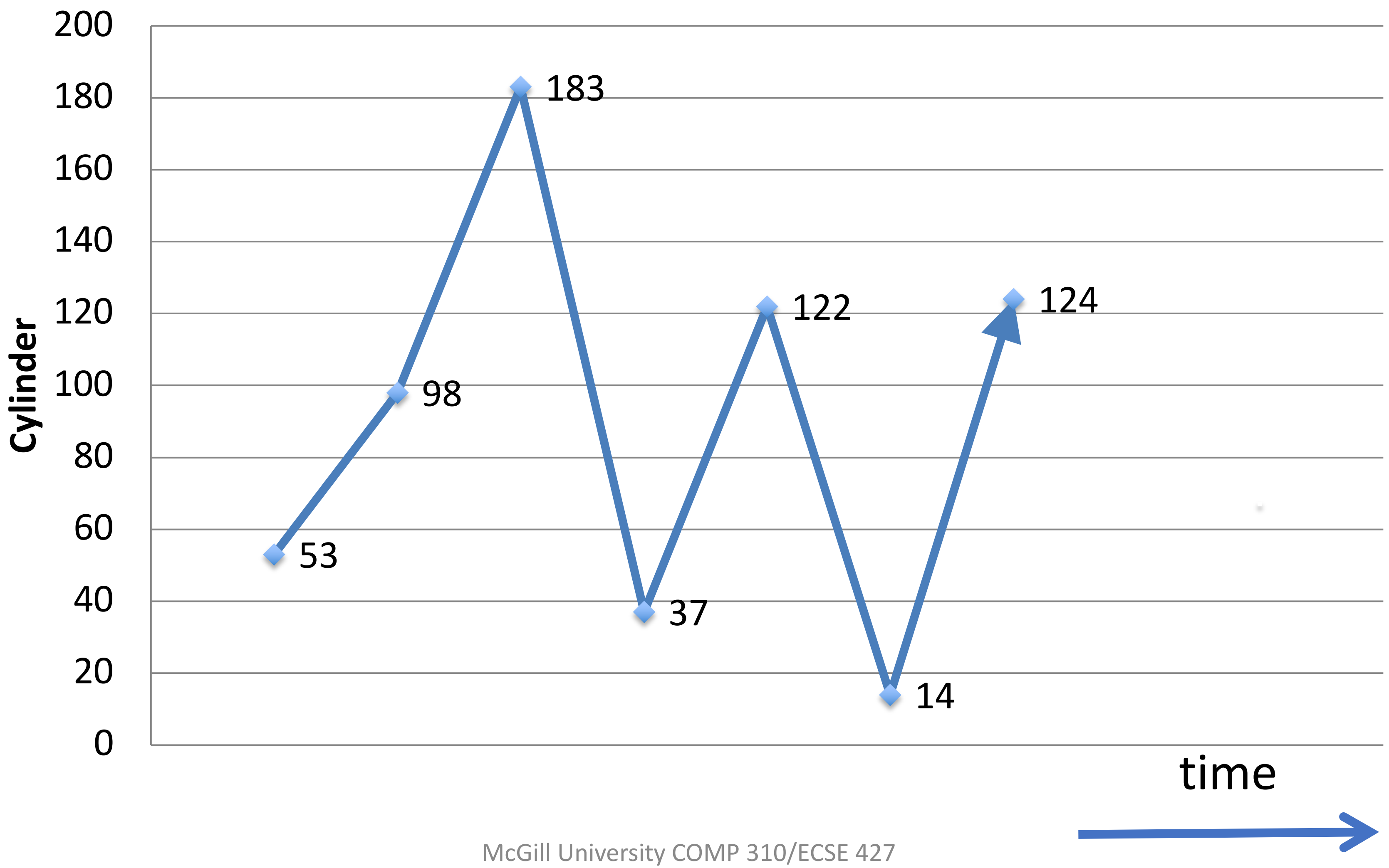
FCFS



Head = 53
Queue = 98, 183, 37, 122, 14, 124, 65, 67

579

FCFS

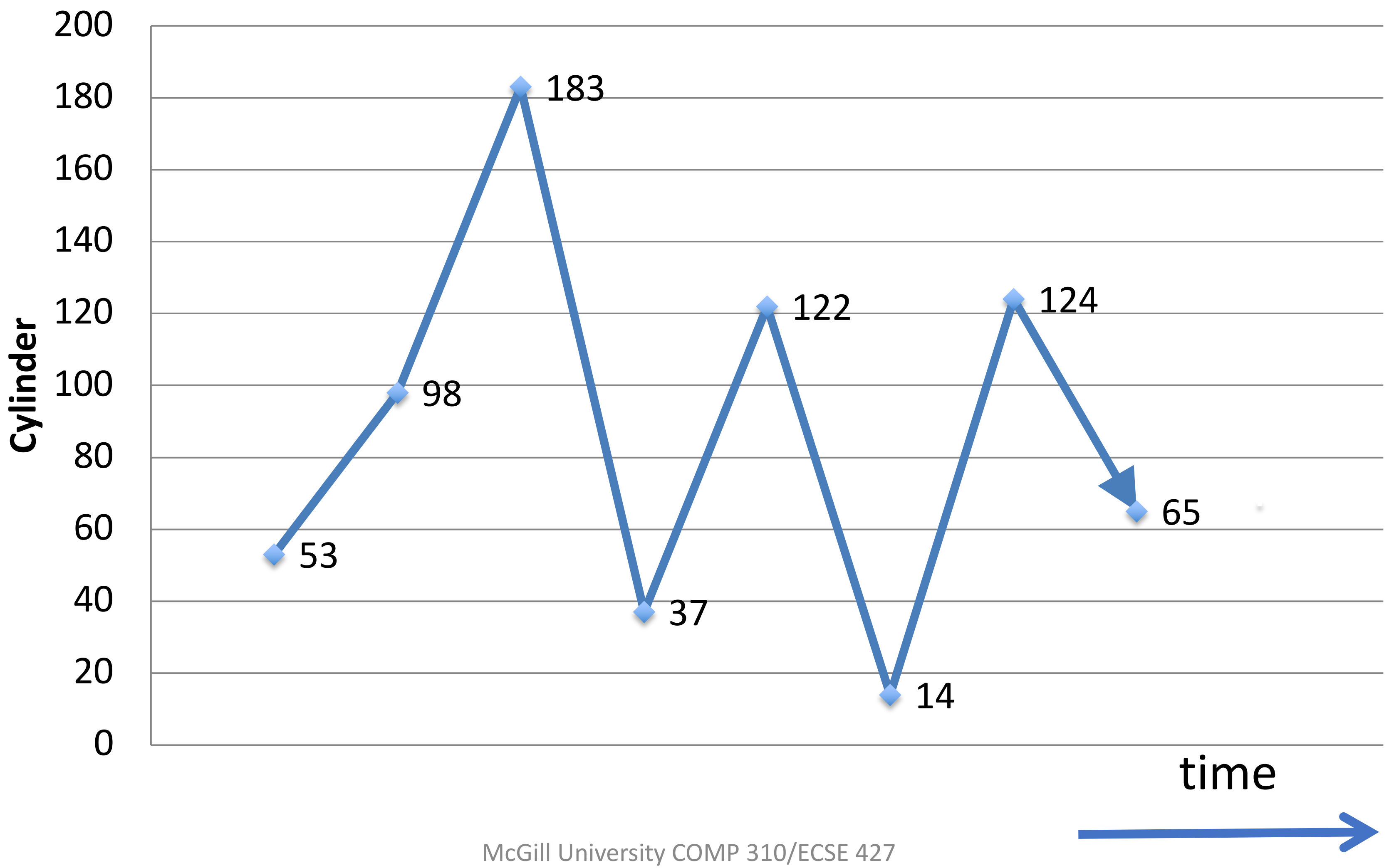


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

638

FCFS

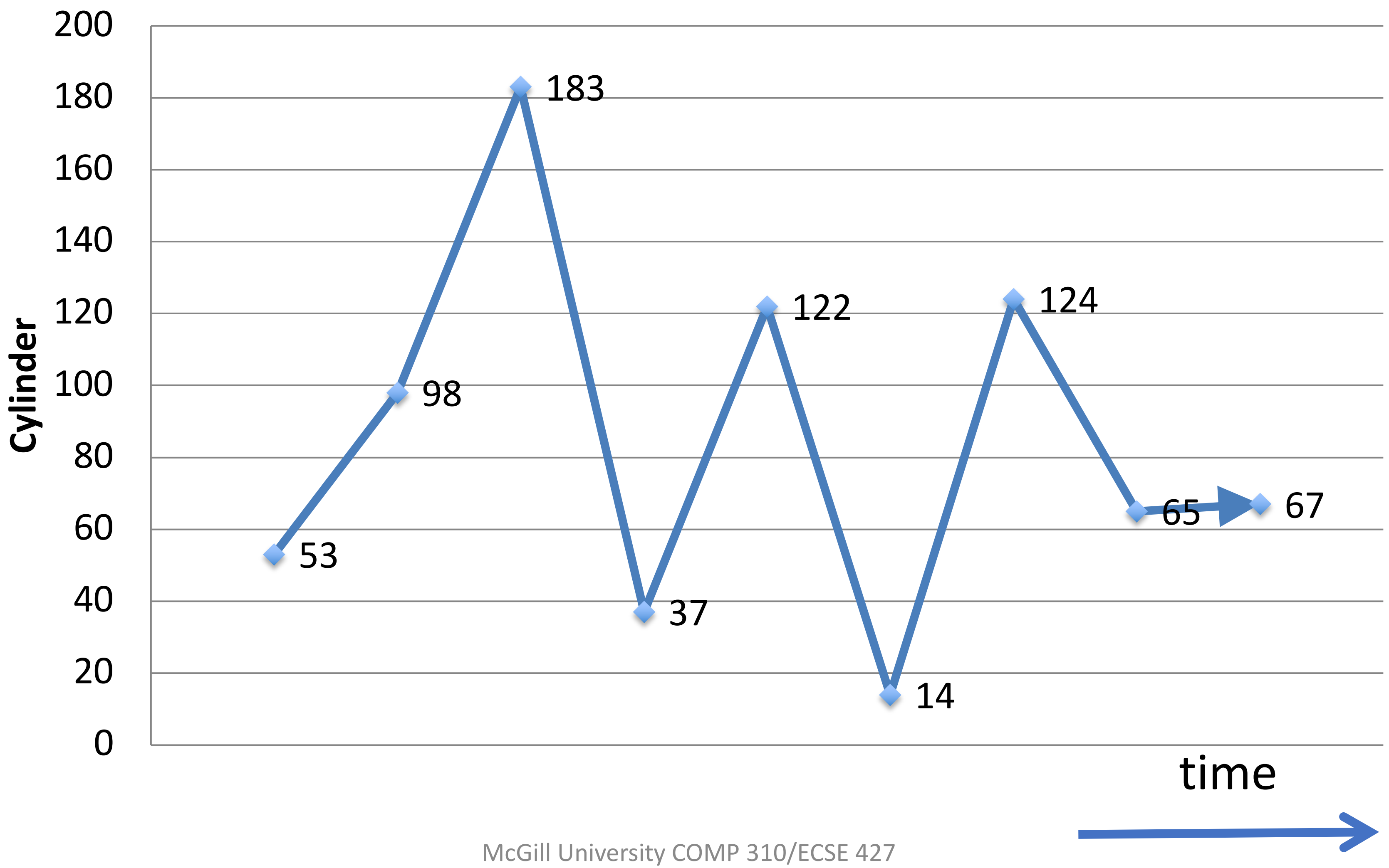


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

640

FCFS



Shortest Seek Time First (SSTF)

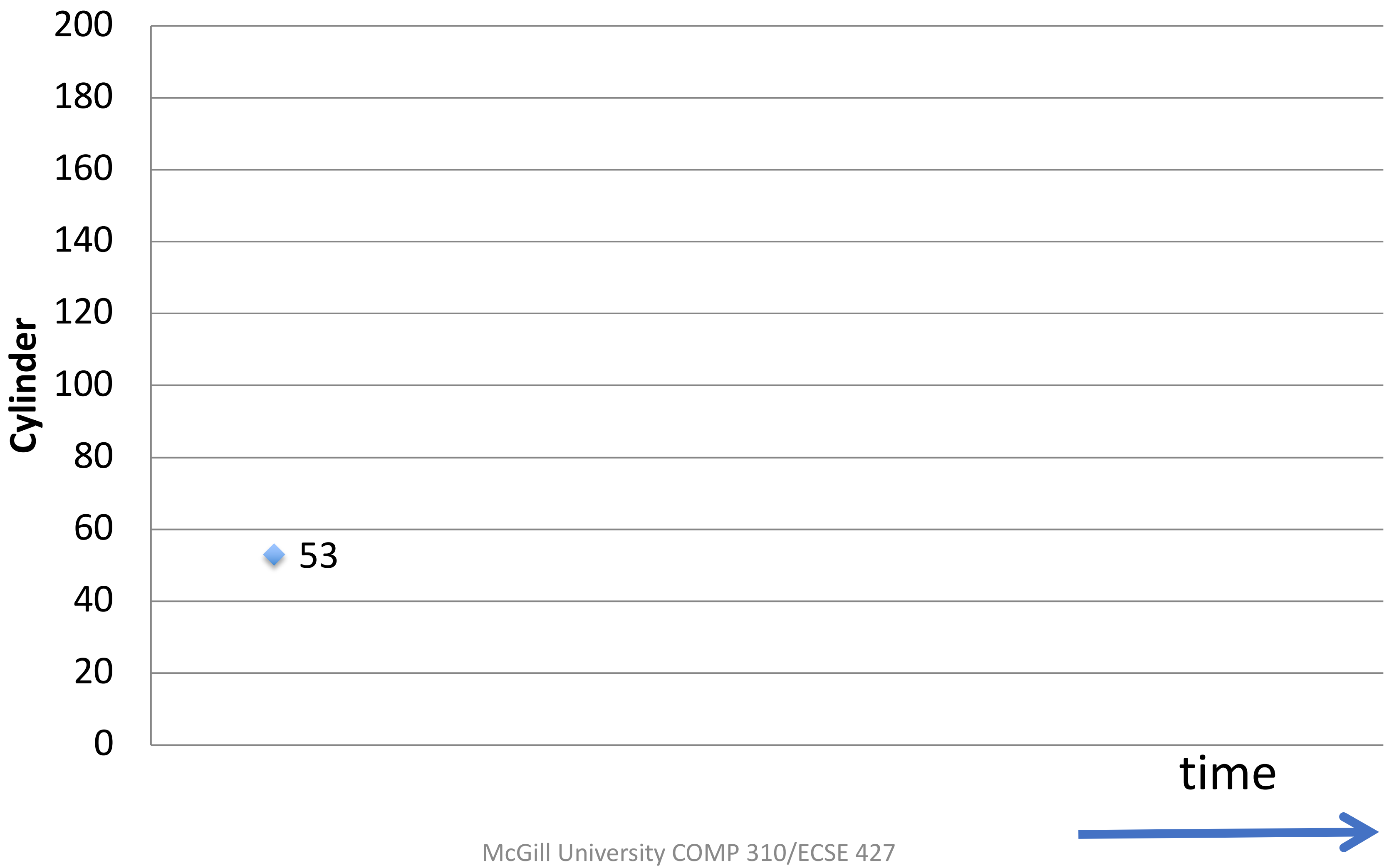
Pick **“nearest” request** in queue

- “nearest” = closest to current head position

Head = 53
Queue = 98, 183, 37, 122, 14, 124, 65, 67

0

SSTF

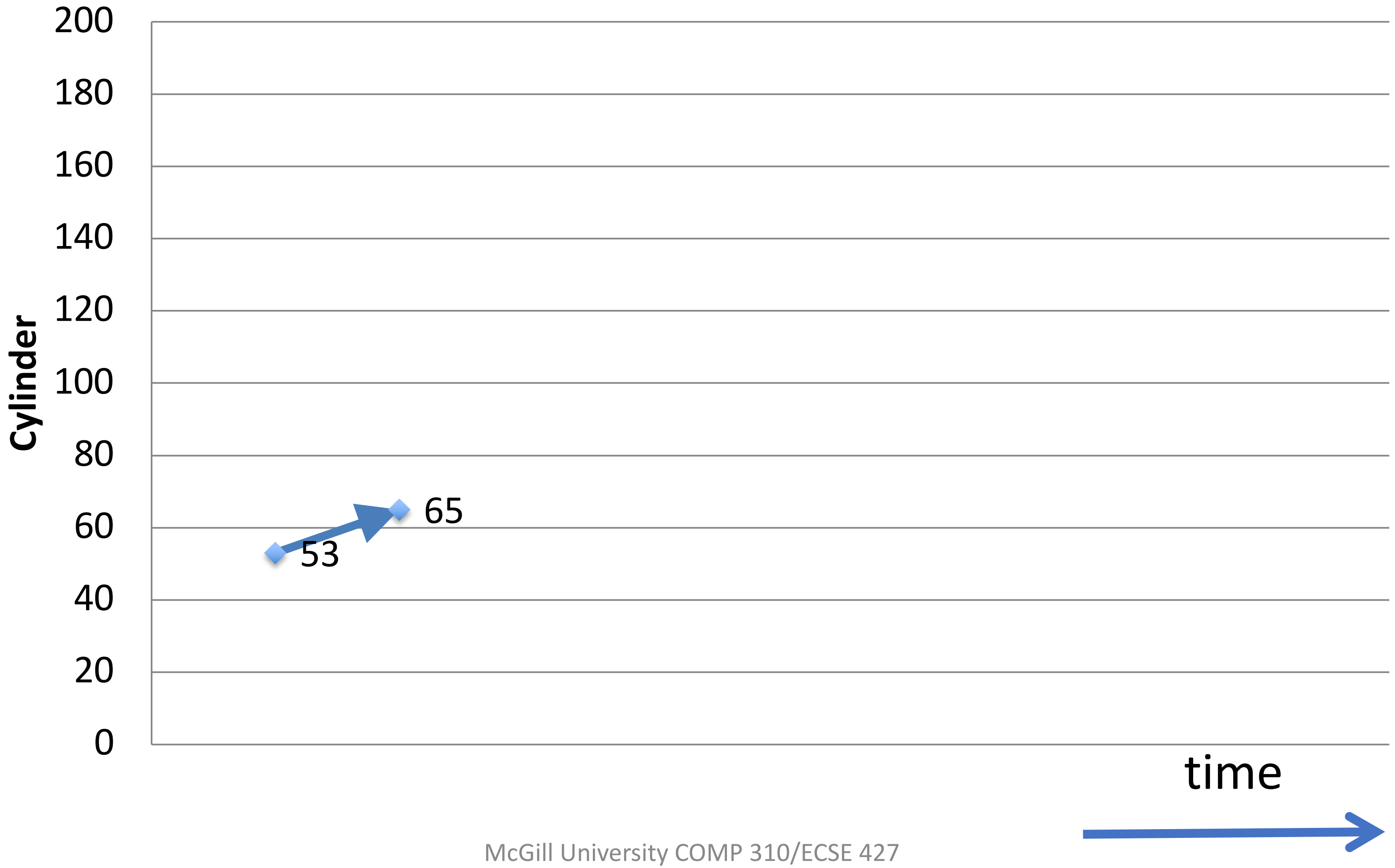


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

12

SSTF

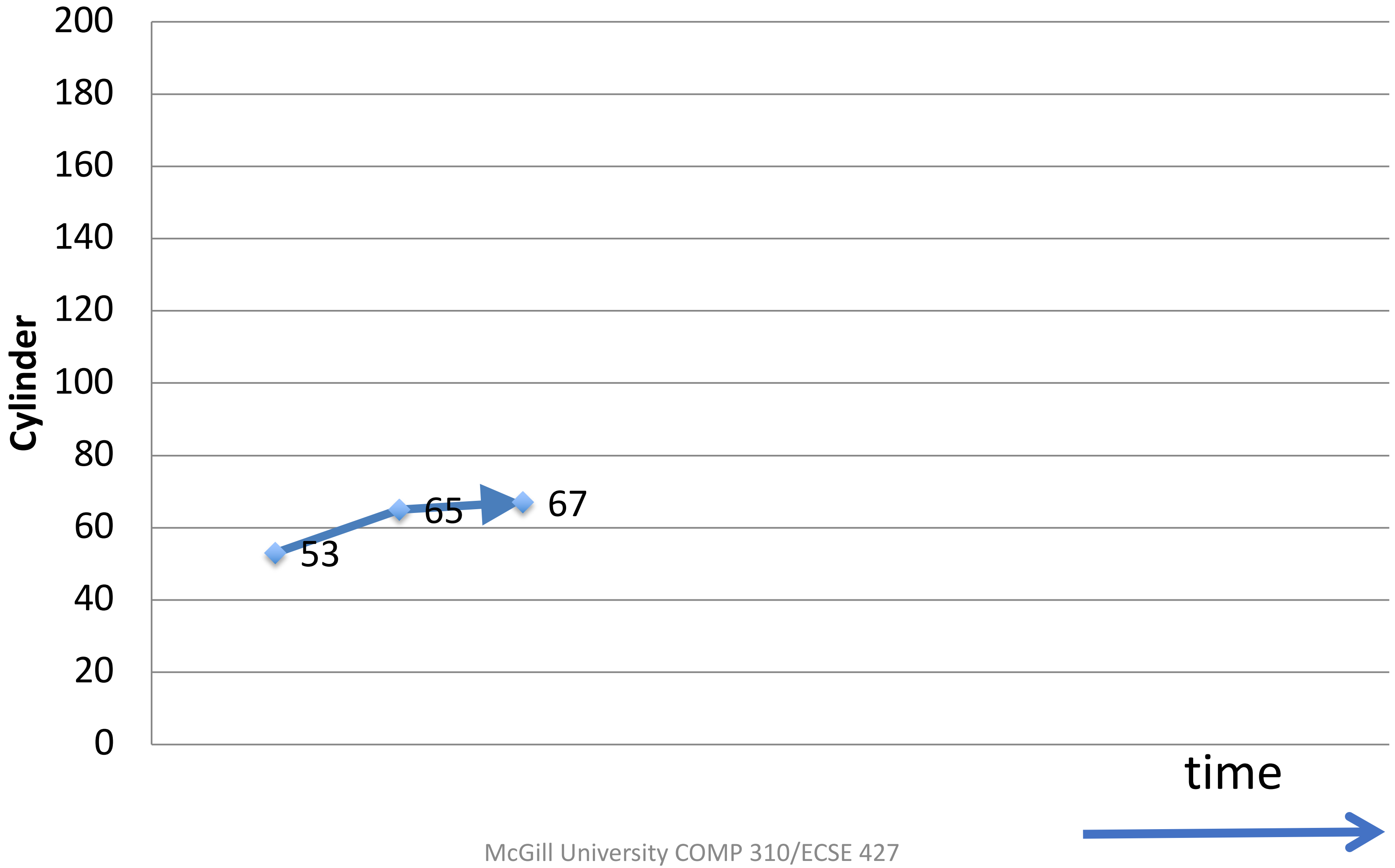


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

14

SSTF

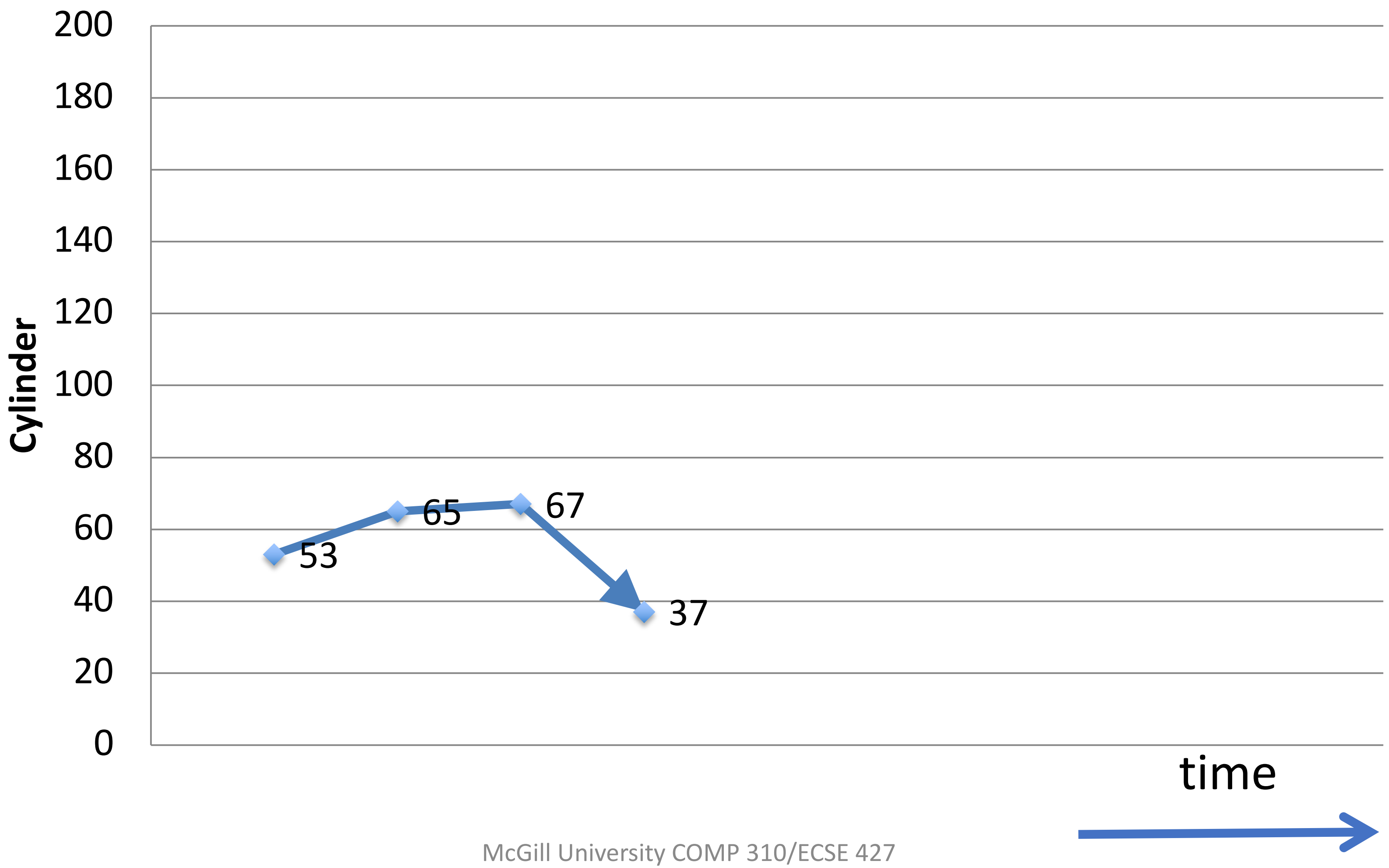


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

44

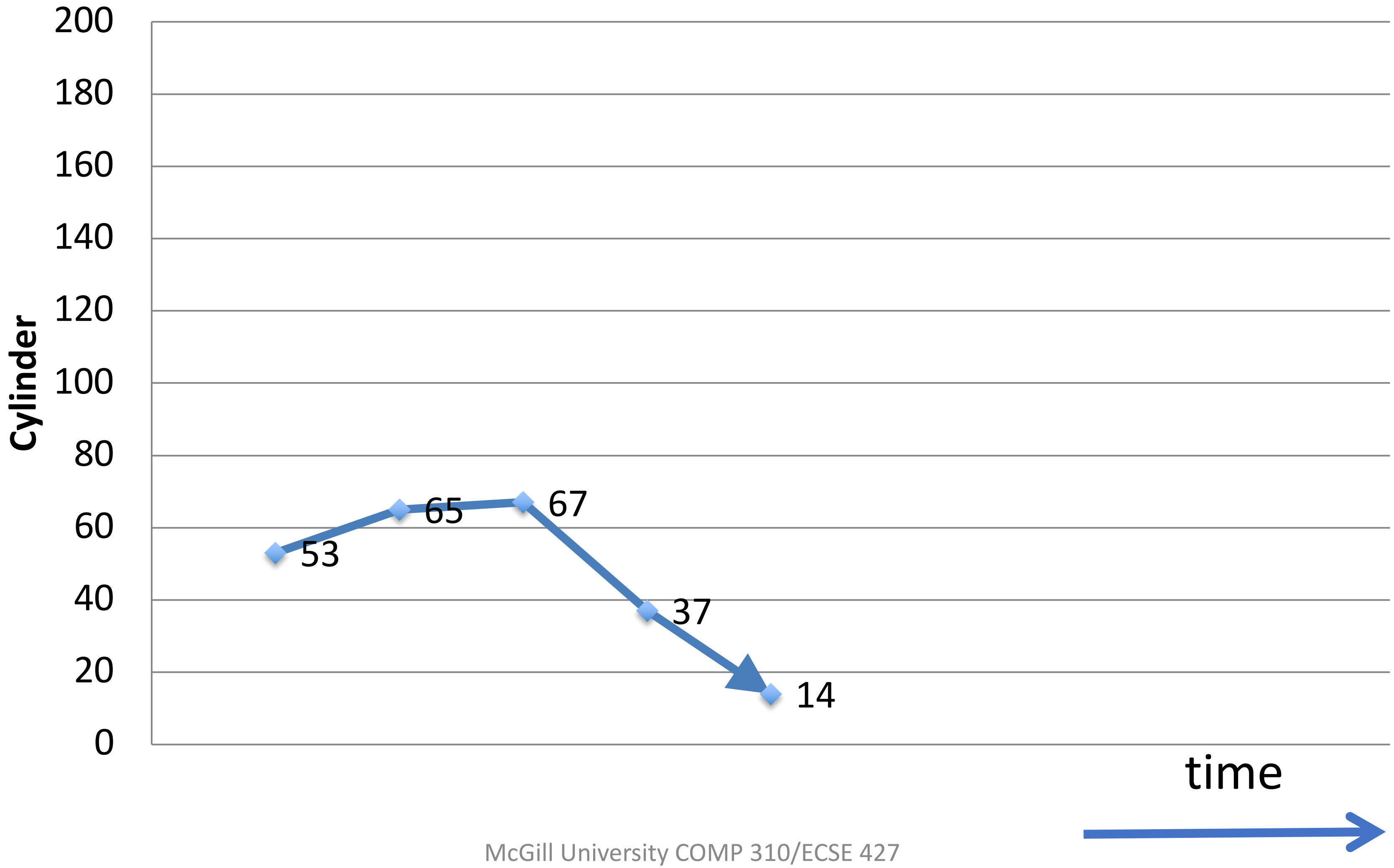
SSTF



Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

SSTF

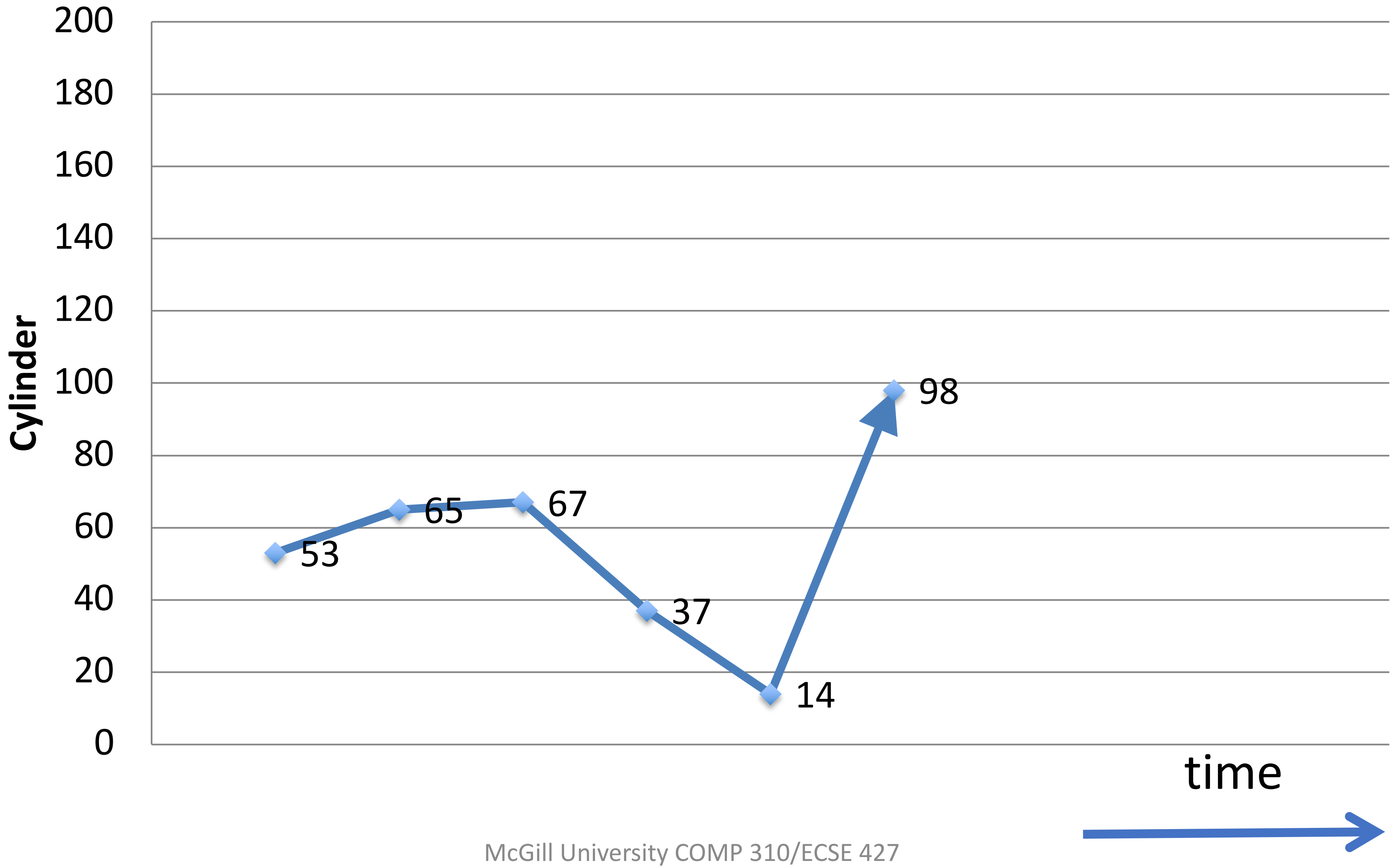


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

151

SSTF

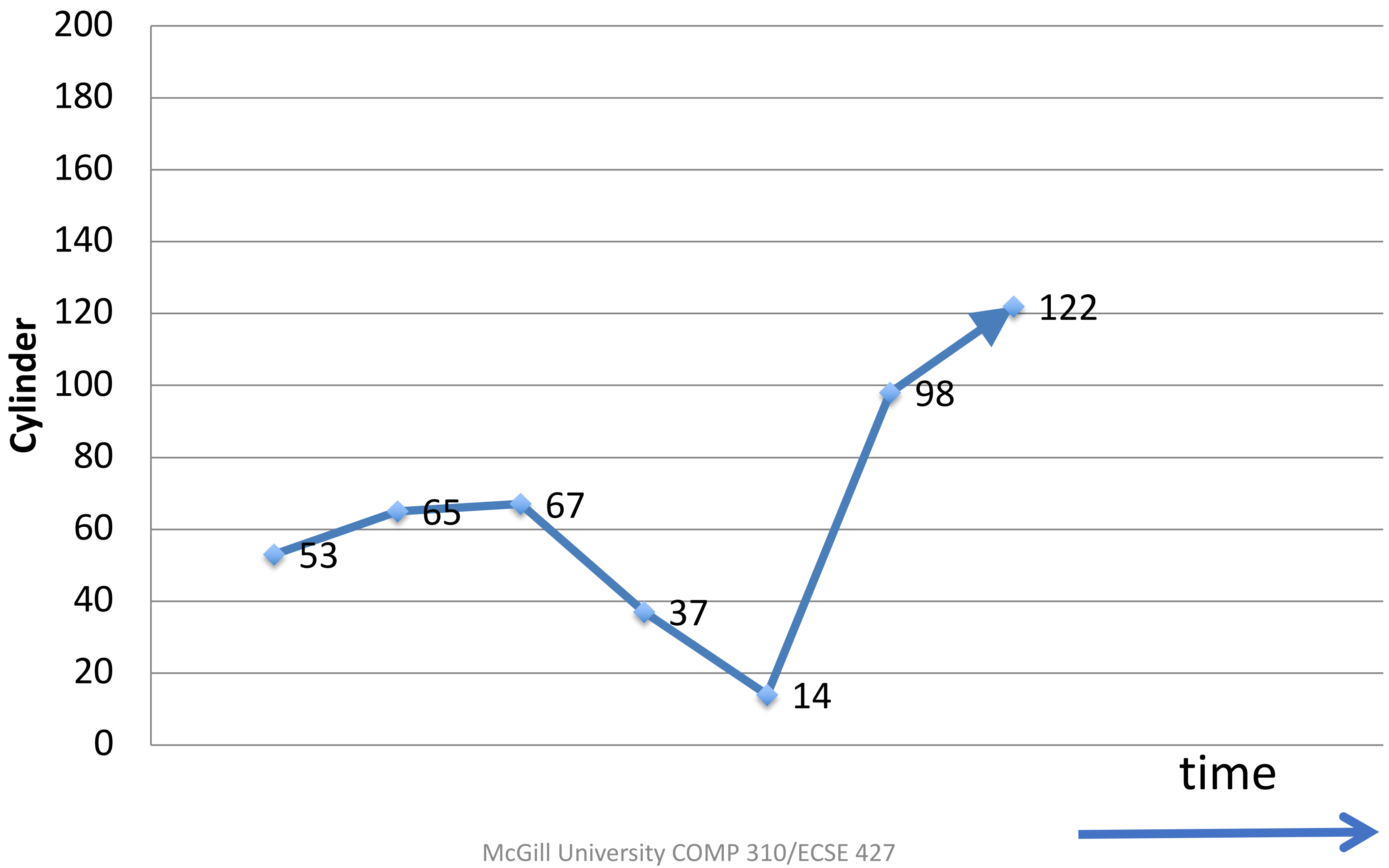


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

175

SSTF

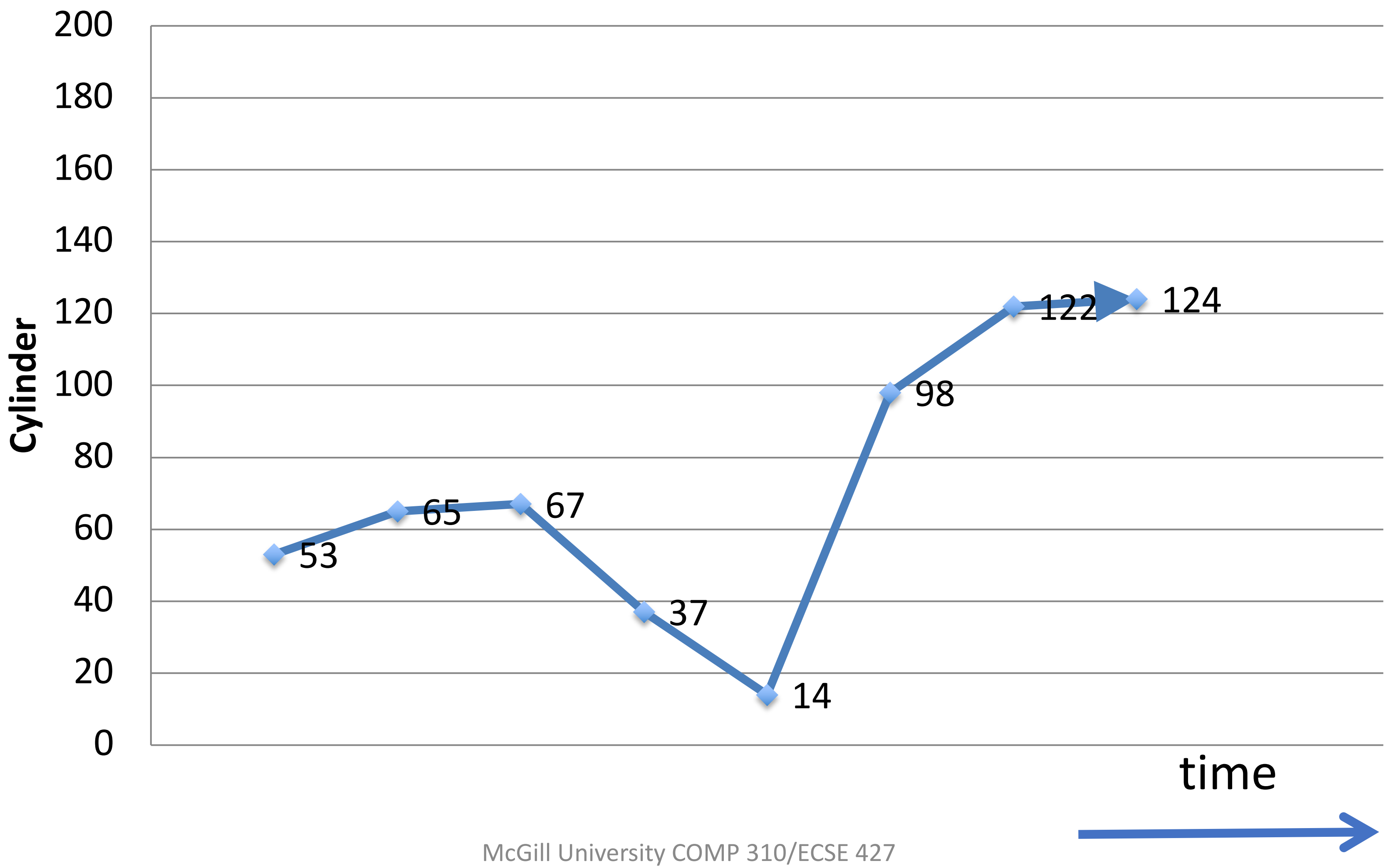


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

177

SSTF

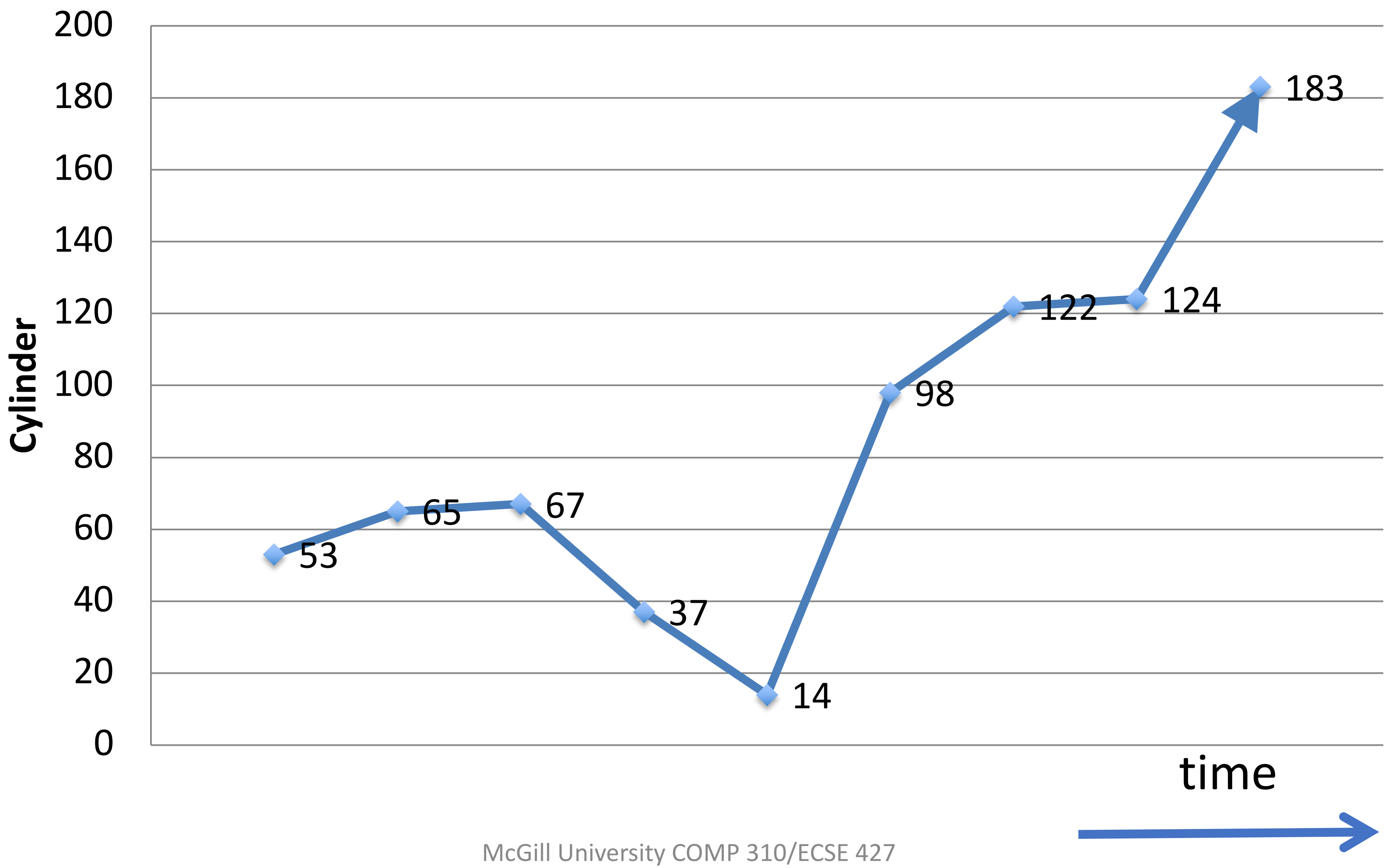


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

236

SSTF



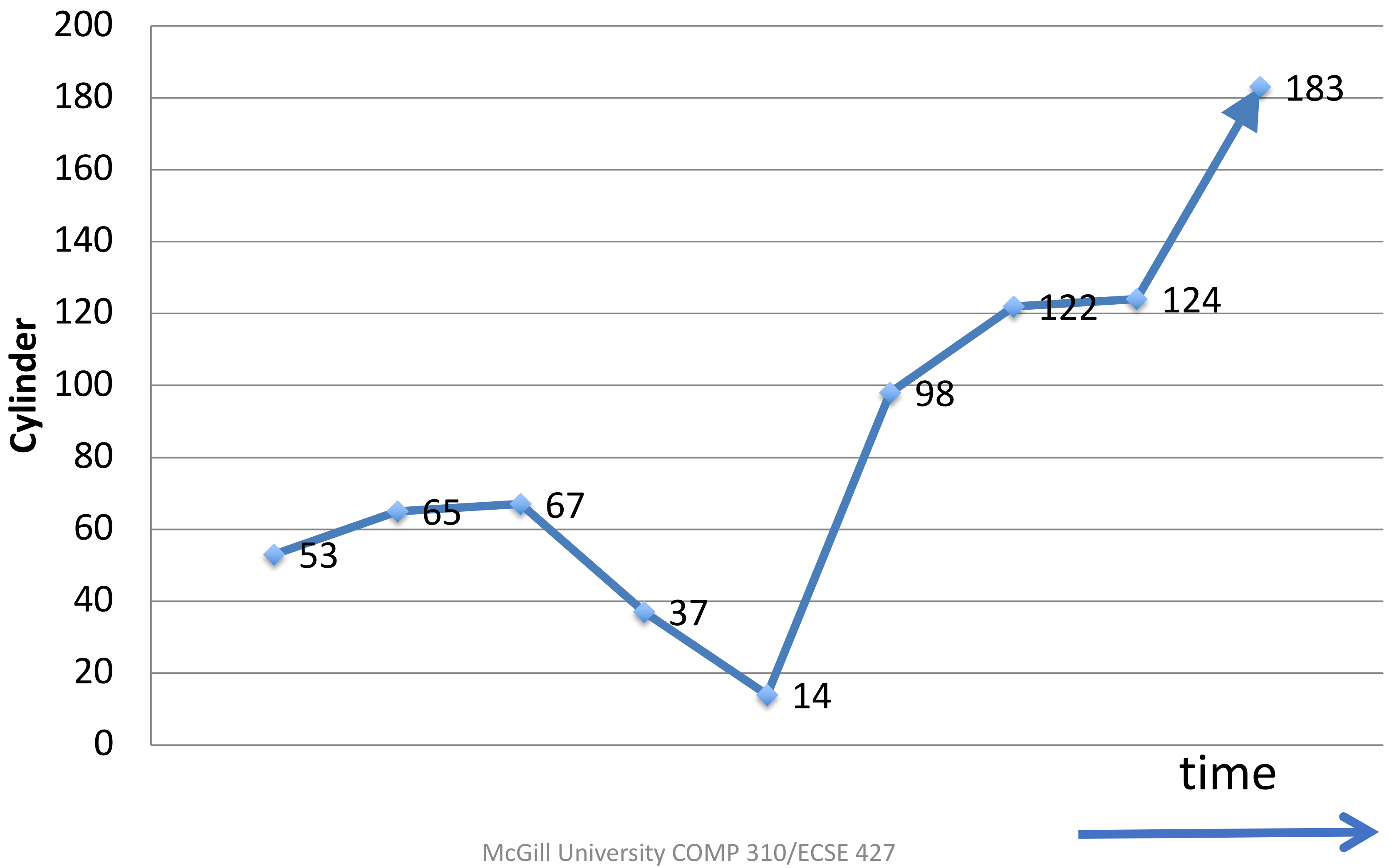
Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

236

< **FCFS (640)**

SSTF



SSTF

+ Very good seek times

☹ Subject to starvation

- Request on inside or outside can get starved

SCAN

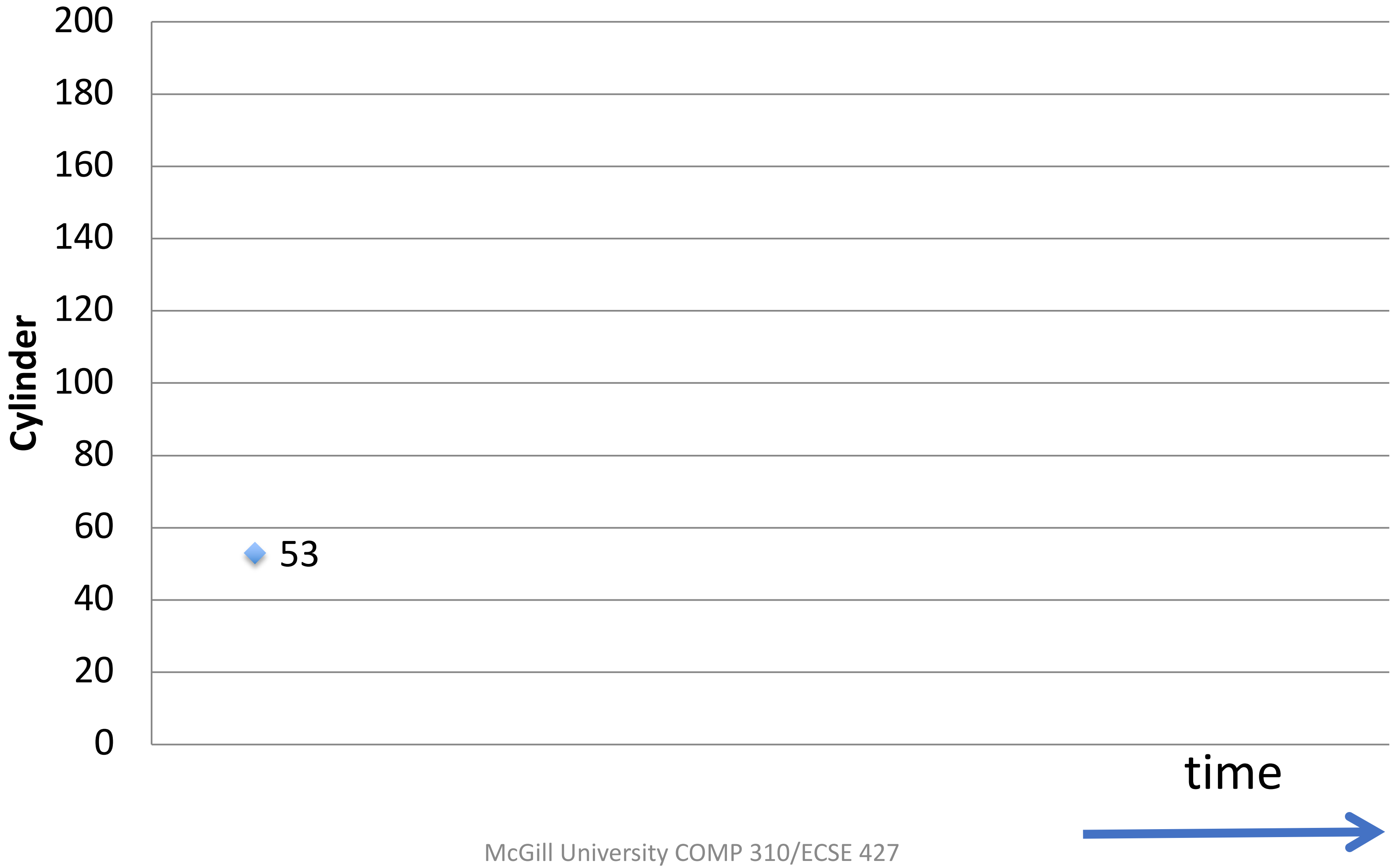
- Continue moving head in one direction
  - From 0 to MAX_CYL
  - Then, from MAX_CYL to 0
- Pick up requests as you move head

Head = 53, moving down

Queue = 98, 183, 37, 122, 14, 124, 65, 67

0

SCAN

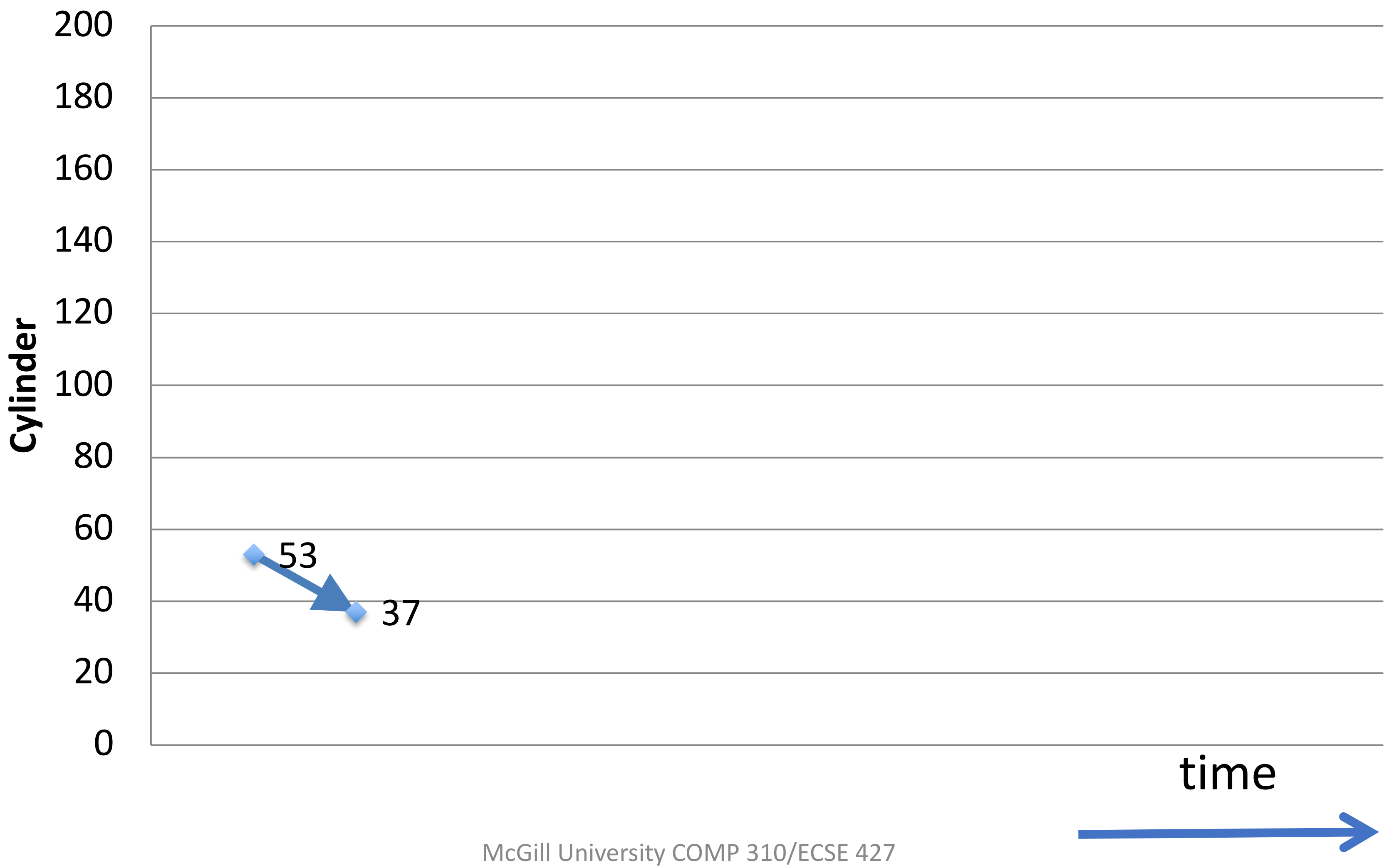


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

16

SCAN

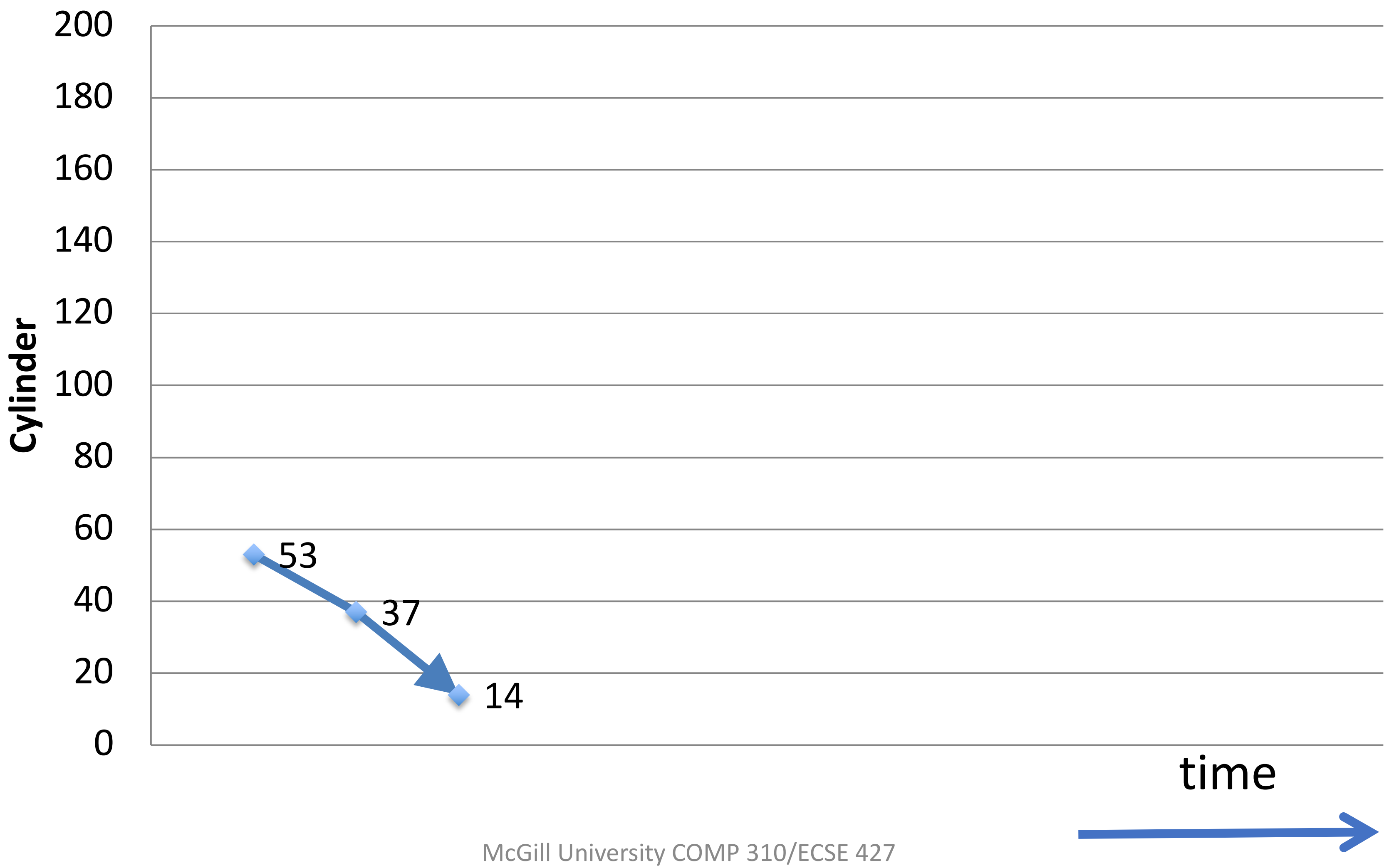


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

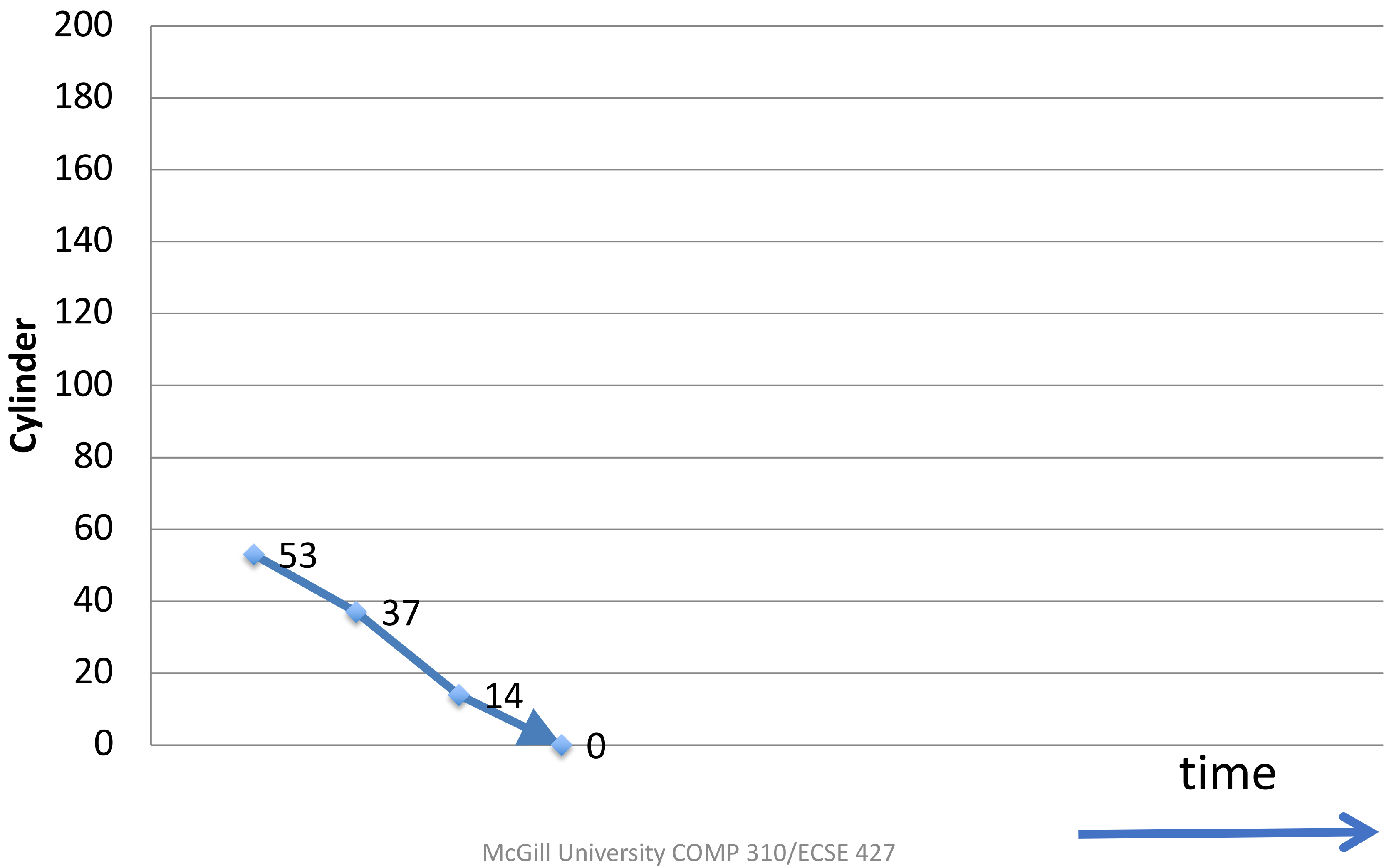
39

SCAN



Head = 53
Queue = 98, 183, 37, 122, 14, 124, 65, 67

SCAN

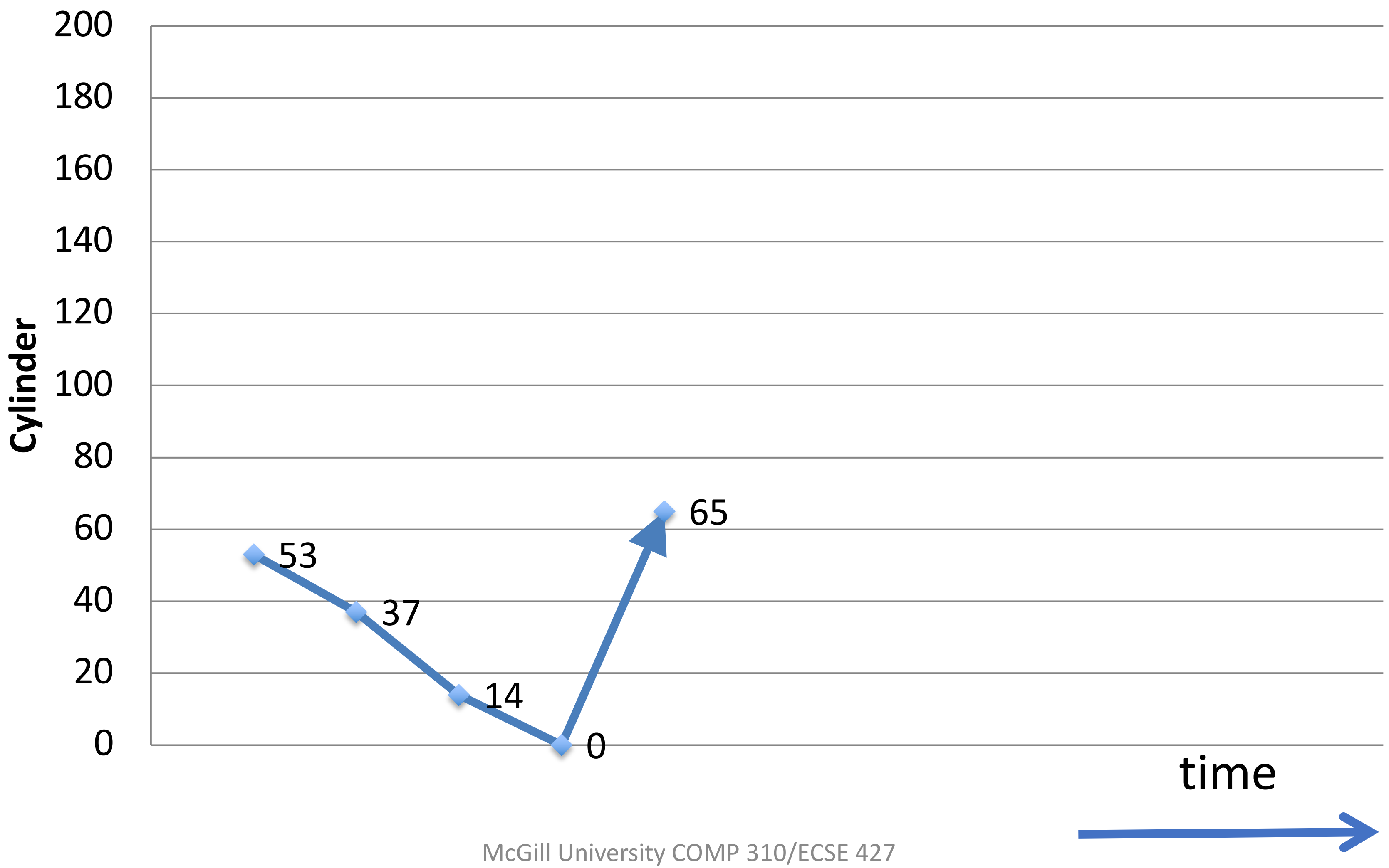


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

118

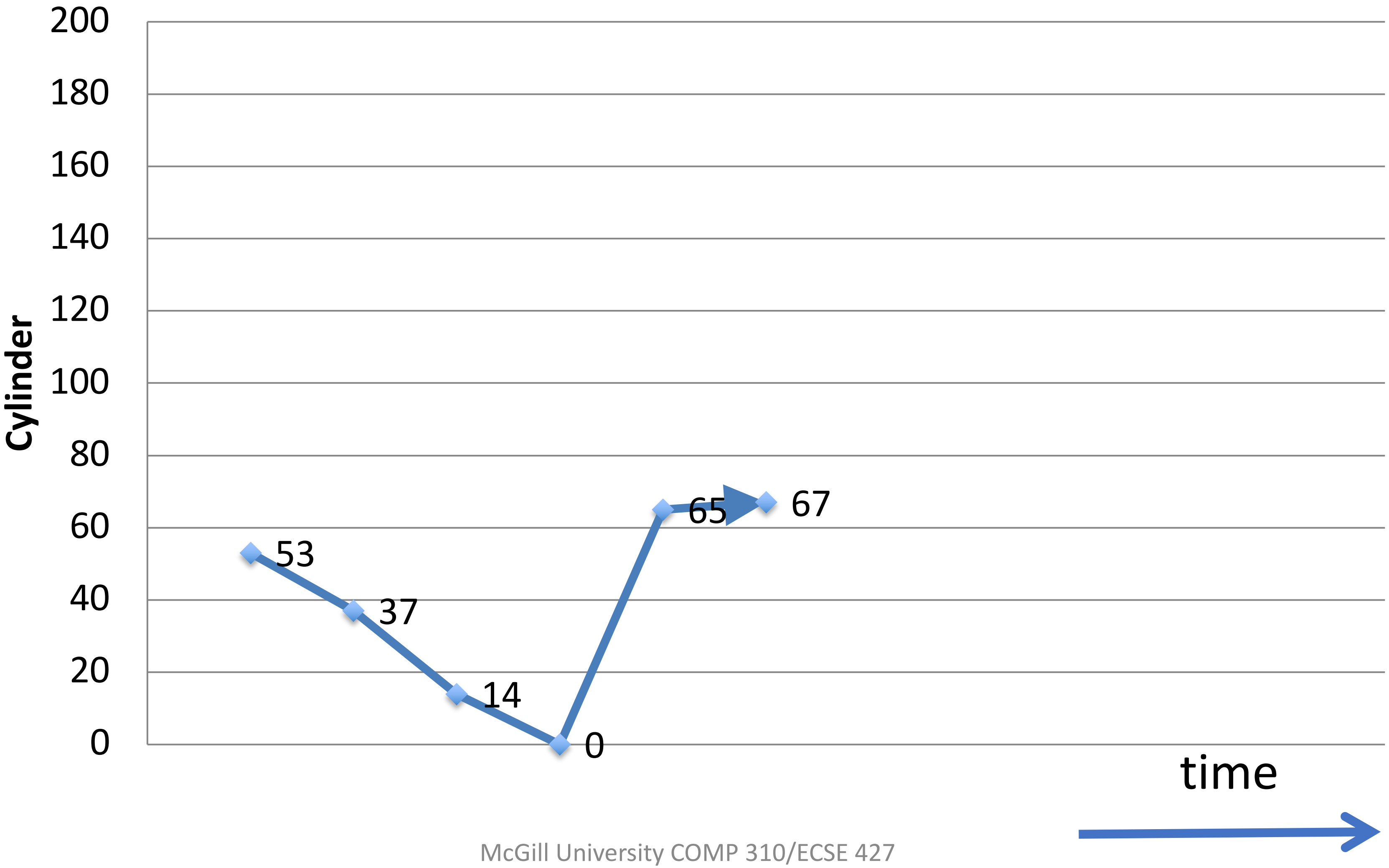
SCAN



Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

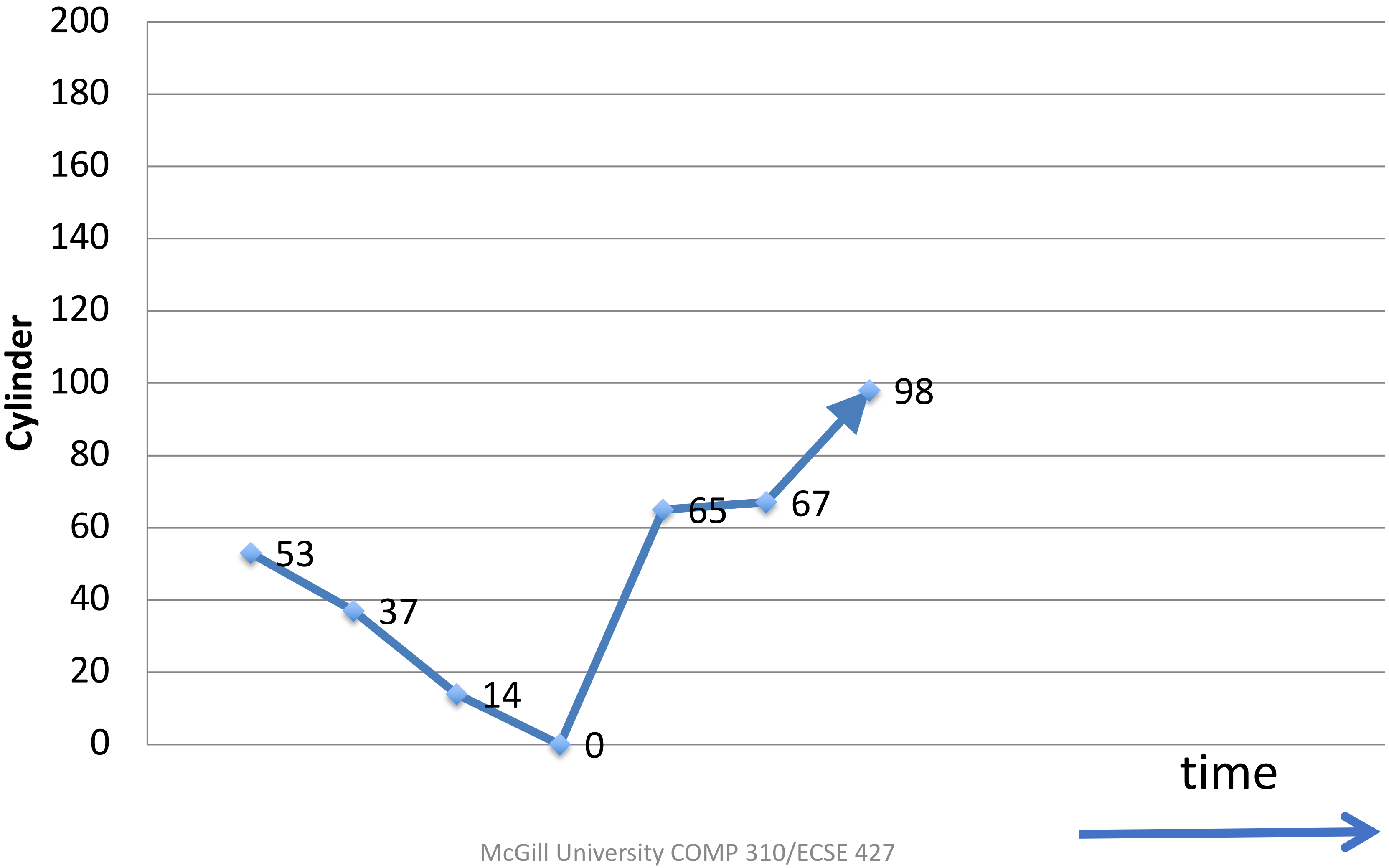
SCAN



Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

SCAN

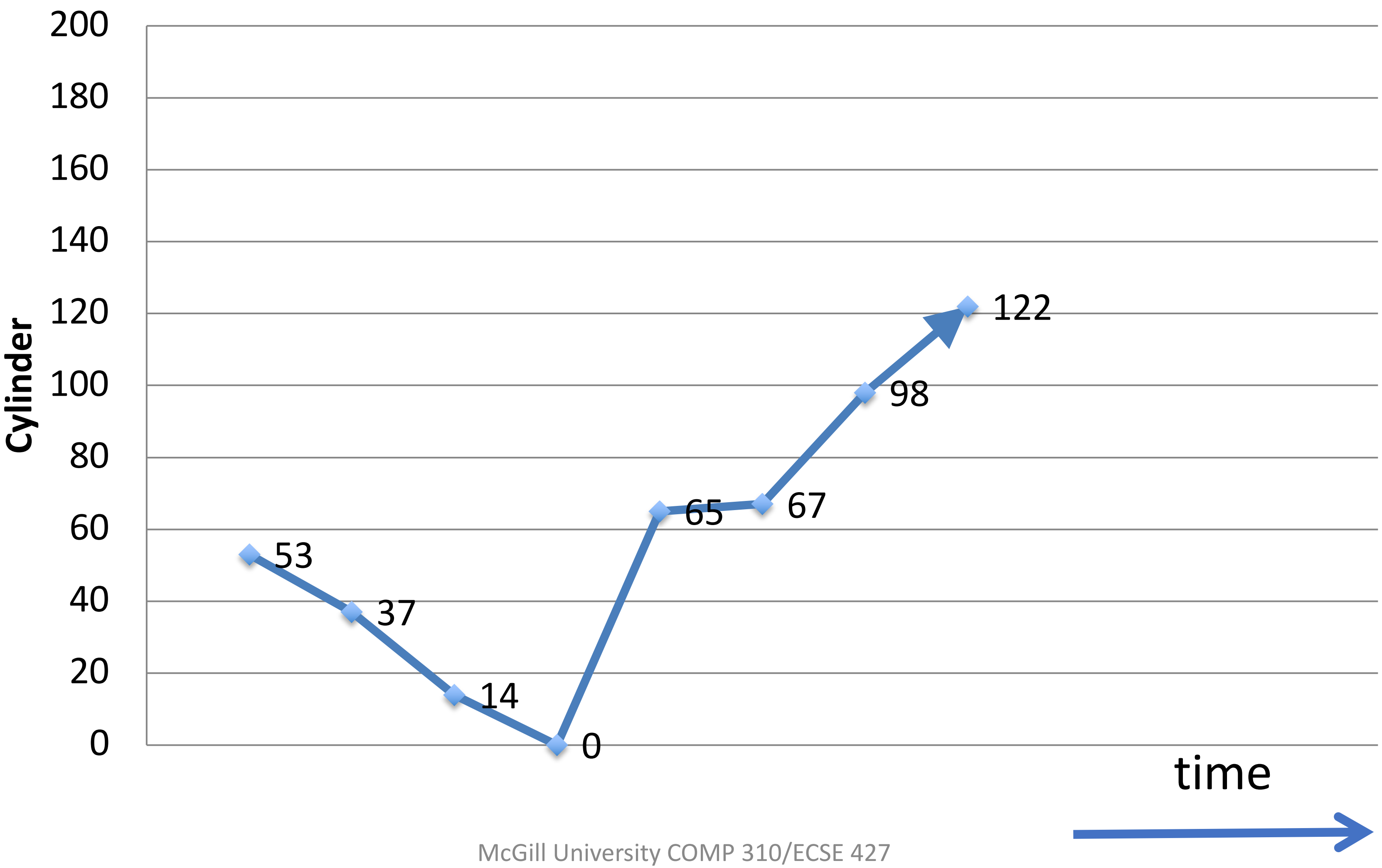


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

175

SCAN

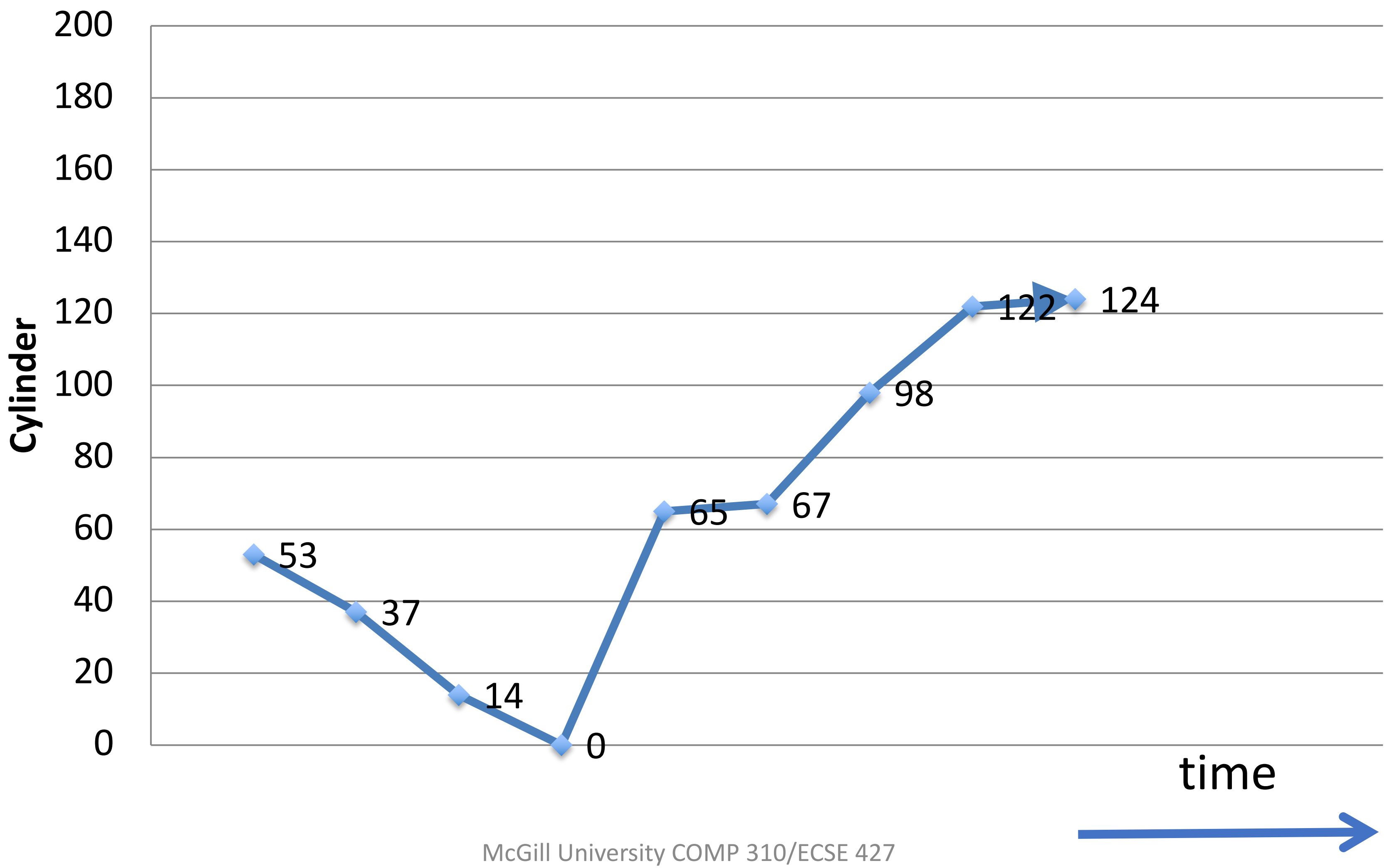


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

177

SCAN

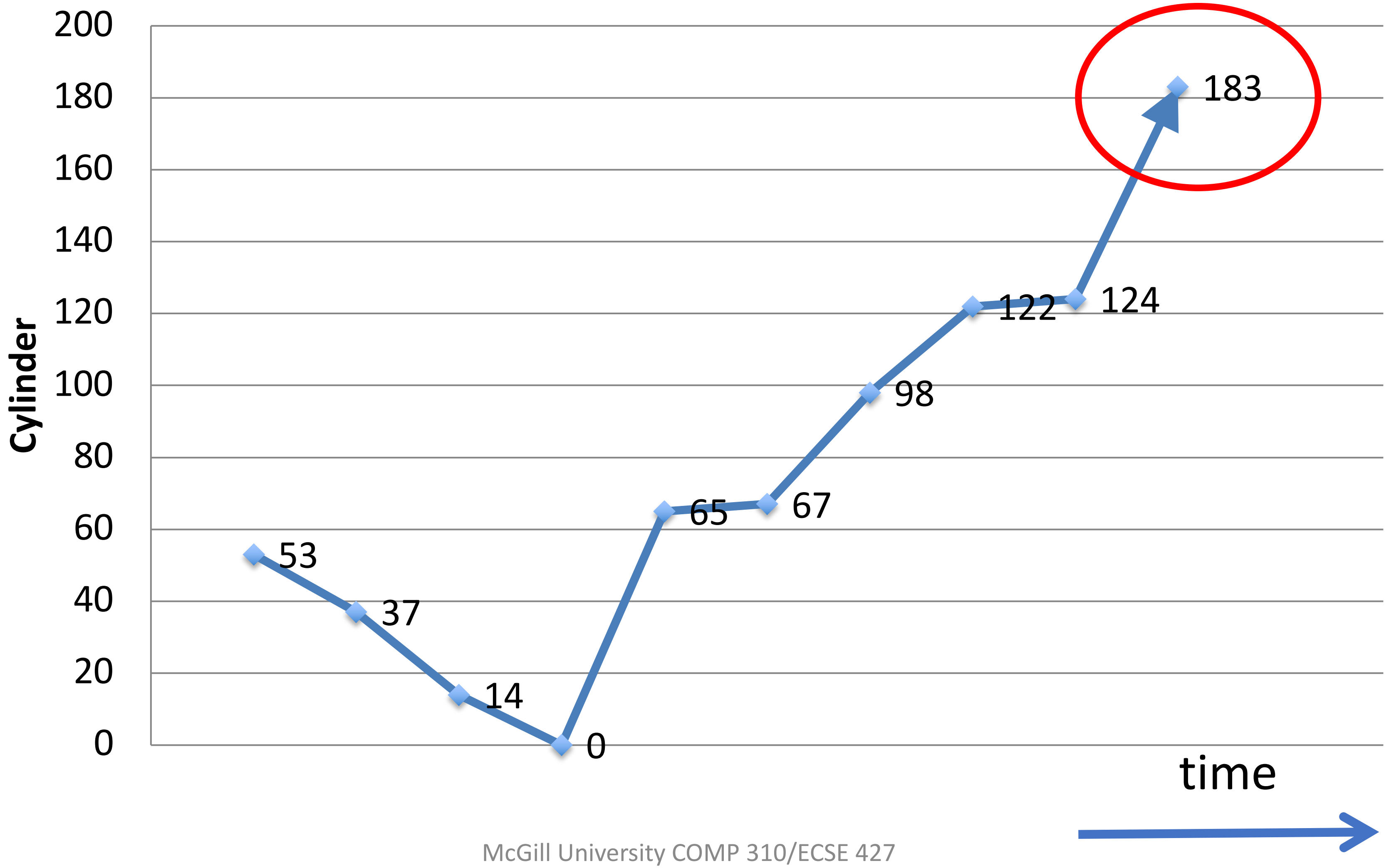


Head = 53

Queue = 98, 183, 37, 122, 14, 124, 65, 67

236

SCAN



Here we do not go all the way to MAX_CYL, because this is the last request in the queue.

C-SCAN

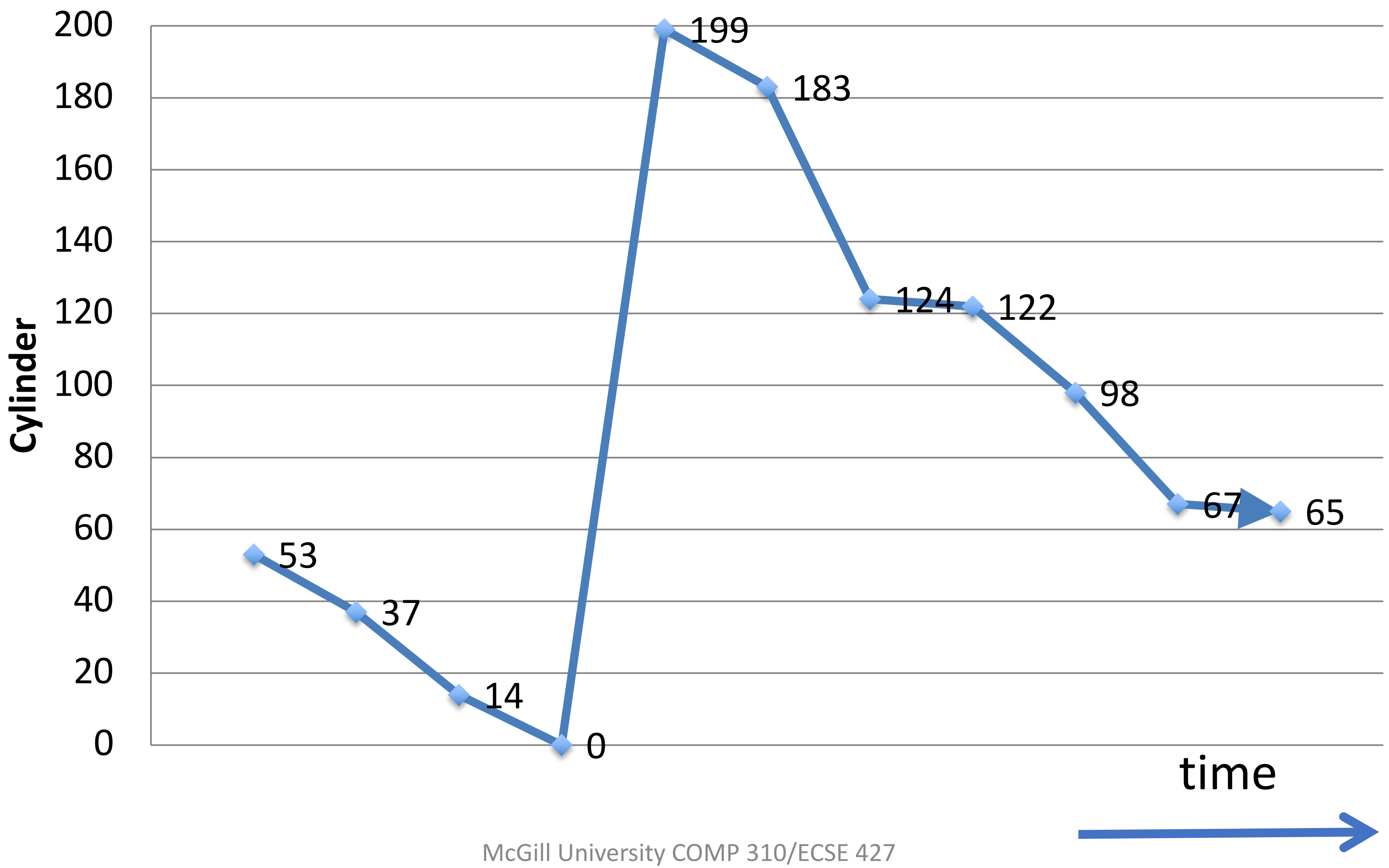
- Similar to SCAN
- Always move head
  - From MAX_CYL to 0; pick up requests as head moves
  - From 0 to MAX_CYL; no requests served

- C-SCAN can also be implemented in reverse
  - From MAX_CYL to 0; no requests served
  - From 0 to MAX_CYL; pick up requests as head moves

Head = 53, moving down
Queue = 98, 183, 37, 122, 14, 124, 65, 67

388

C-SCAN



C-SCAN

☹ Number of cylinders slightly higher

+ More uniform wait time

C-LOOK

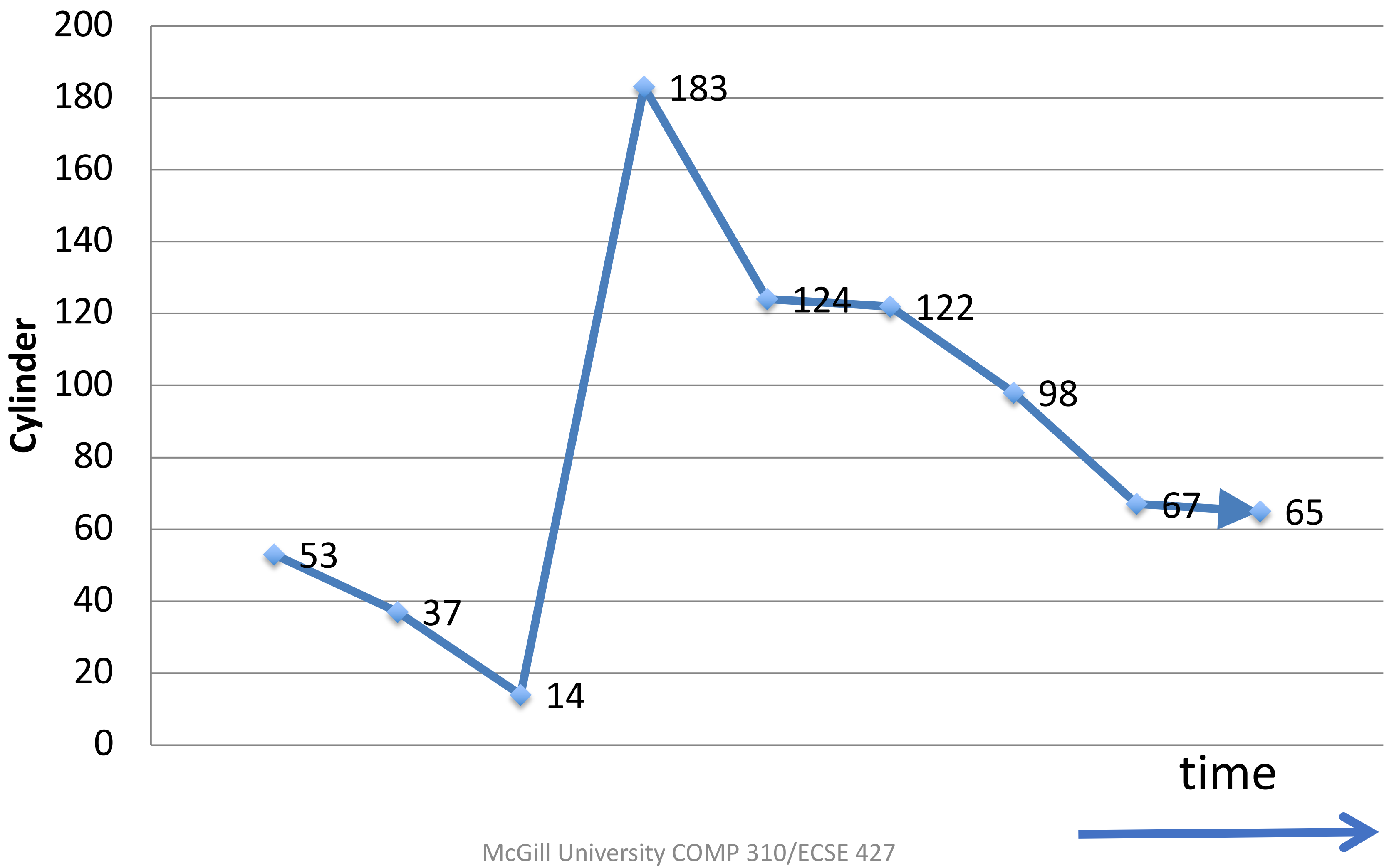
- Similar to C-SCAN
- Always move head
  - From MAX_CYL_IN_QUEUE to MIN_CYL; serve requests as head moves
  - From MIN_CYL to MAX_CYL_IN_QUEUE; no requests served

- C-LOOK can also be implemented in reverse

Head = 53, moving down
Queue = 98, 183, 37, 122, 14, 124, 65, 67

326

C-LOOK



In Practice

- Some variation of C-LOOK

Optimize Disk Access

Rule 4:

Avoid rotational latency

- Clever disk allocation
- Locate consecutive blocks of file on consecutive sectors in a cylinder

When does what work well?

- Low load: clever allocation
- High load: disk scheduling

Why? – Under High Load

- Many scheduling opportunities
  - Many requests in the queue

- Allocation gets defeated
  - By interleaved requests for different files

Why? – Under Low Load

- Not much scheduling opportunity
  - Not many requests in the queue
- Sequential user access -> sequential disk access
- Cache tends to reduce load

Summary – Disk Management

- Disk characteristics
  - Access disk >> access memory
  - Seek > Rotational Latency > Transfer
- Optimizations
  - Cache
  - Read-ahead
  - Disk allocation
  - Disk scheduling

Summary – Key Concepts

- I/O devices
  - OS role for integrating I/O devices in systems
  - Polling, Interrupts
- Notion of "permanent" storage
- File system interface
- Disk Management for HDDs

Further Reading

Operating Systems: Three Easy Pieces by R. & A. Arpaci-Dusseau

Chapters 36, 37, 39.

https://pages.cs.wisc.edu/~remzi/OSTEP/

Credits:

Some slides adapted from the OS courses of Profs. Remzi and Andrea Arpaci-Dusseau (University of Wisconsin-Madison), Prof. Willy Zwaenepoel (University of Sydney), and Prof. Youjip Won (Hanyang University).