ФАЙЛОВІ СИСТЕМИ ТА ОРГАНІЗАЦІЯ ВВОДУ-ВИВОДУ В СУЧАСНИХ ОПЕРАЦІЙНИХ СИСТЕМАХ

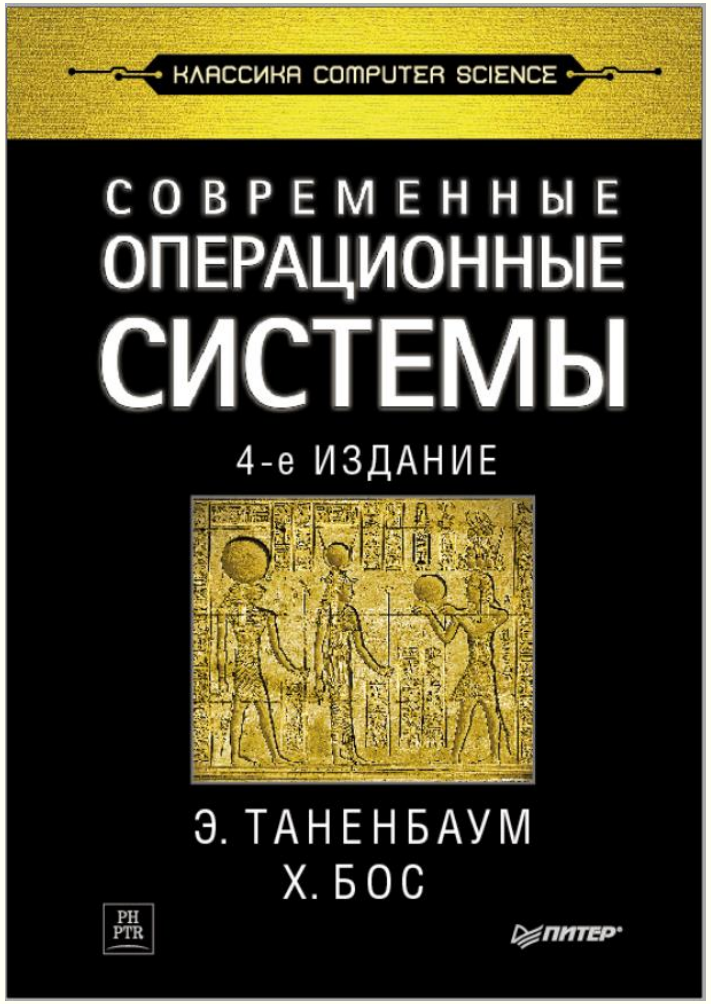
2019-2020 н. р.

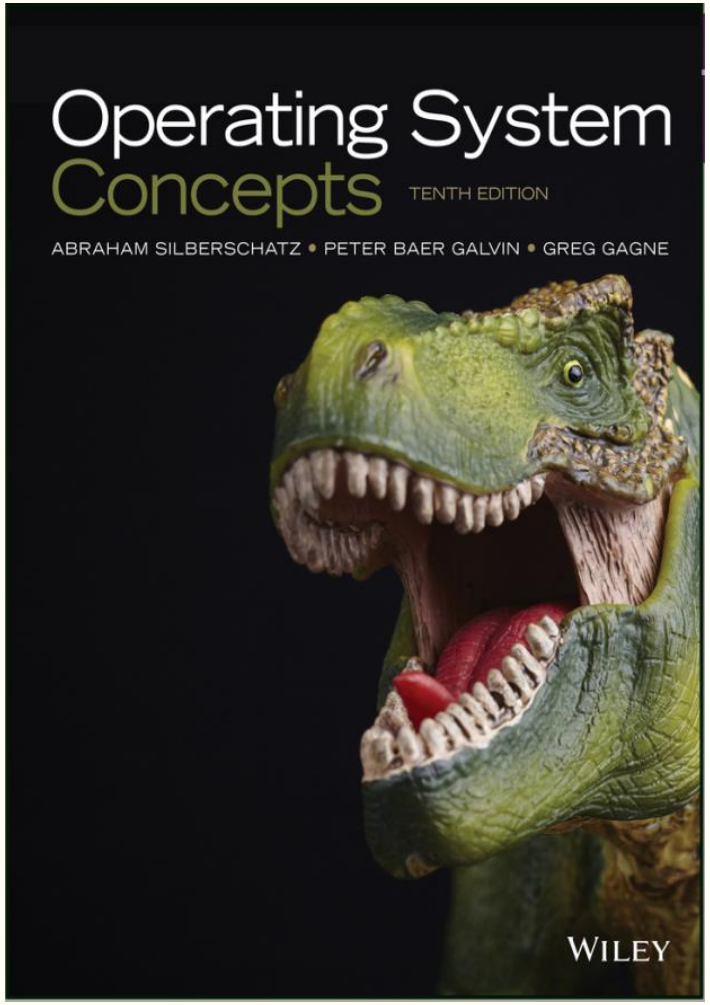
Викладач: Марченко Станіслав Віталійович

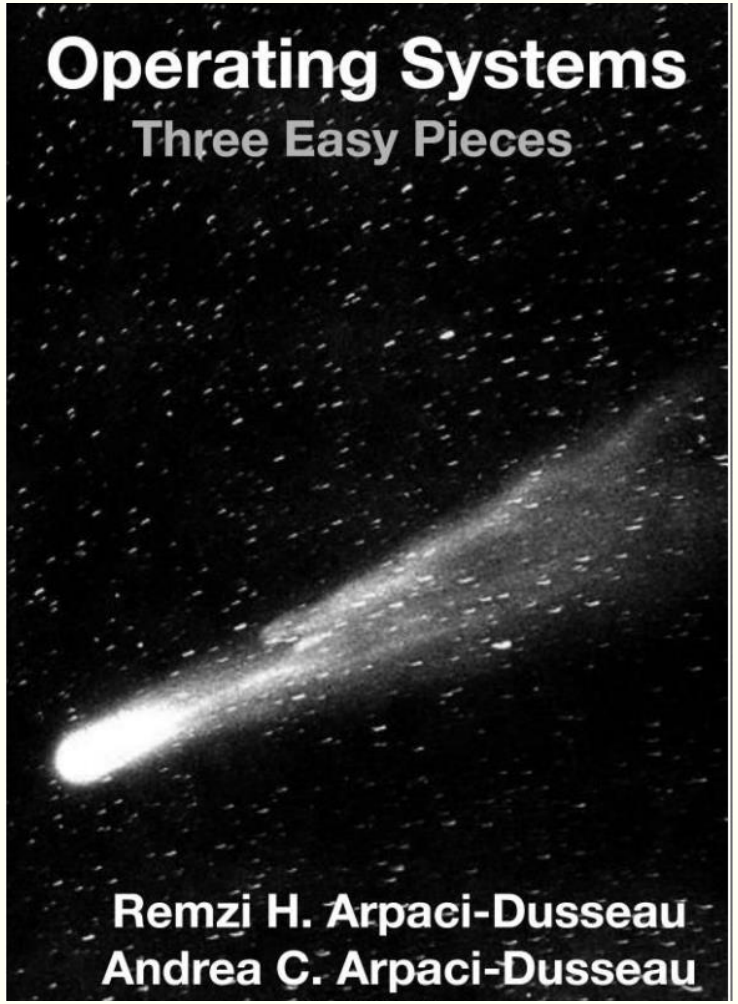
Питання лекції

- Структура носіїв інформації.
- RAID-масиви.
- Система вводу-виводу.
- Файлові системи.

Рекомендована література



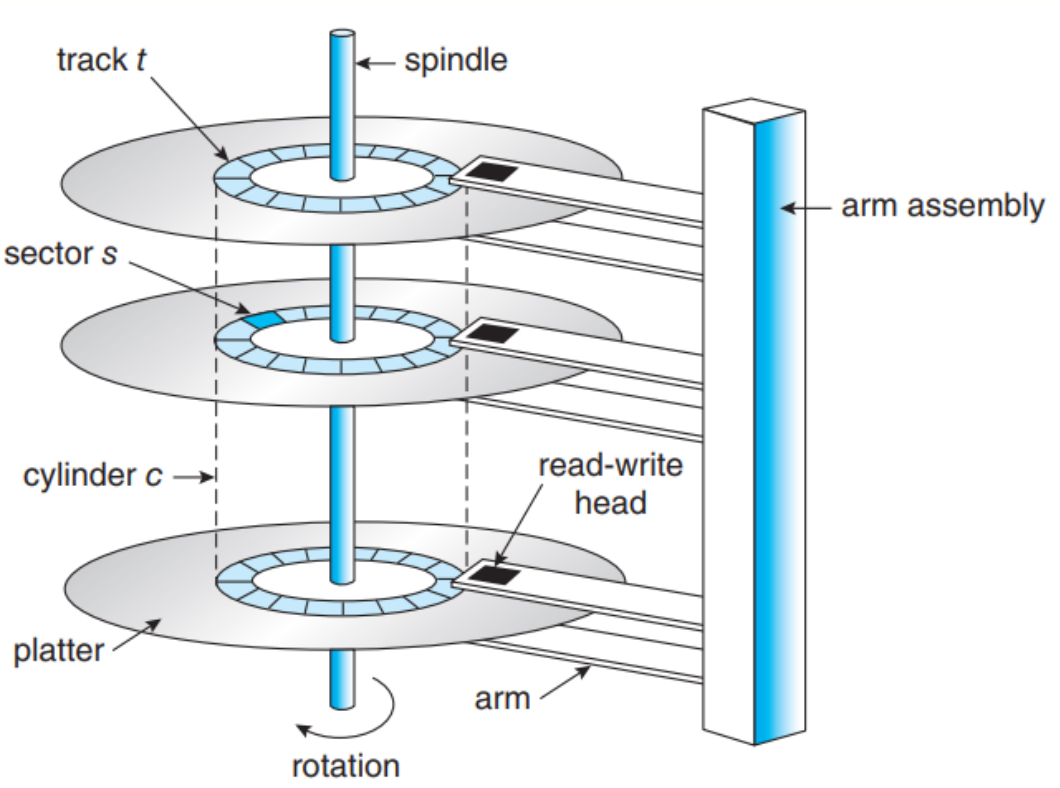




СТРУКТУРА НОСІЇВ ІНФОРМАЦІЇ

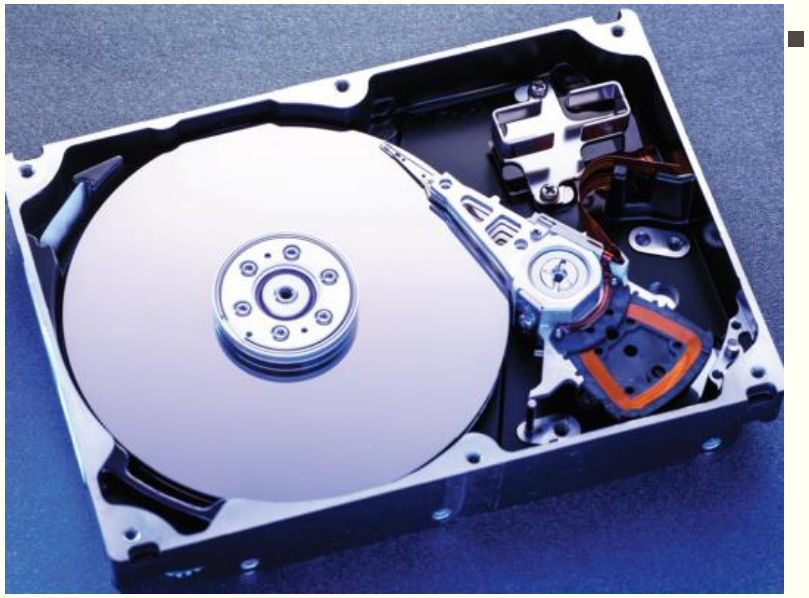
Питання 4.1

Більшість вторинних сховищ для сучасних комп'ютерів – жорсткі диски (HDD) та енергонезалежні (NVM) пристрої



- Концепція жорсткого диску досить проста.
  - Кожний диск (disk platter) має плоску кругову форму діаметром від 1.8 до 3.5 дюймів, покриту з обох сторін магнітним матеріалом.
  - Інформація зберігається шляхом магнітного запису на диски, а її зчитування – розбором магнітного «візерунку» на дисках.
- Голівка зчитування-запису "плаває" над кожною поверхнею кожного диску.
  - Голівки прикріплені до ***кронштейну (disk arm) диску***, який переміщує всі голівки як єдиний масив.
  - Поверхня диску логічно розбита на кругові ***доріжки (tracks)***, розділені на ***сектори***.
  - Набір доріжок у заданій позиції кронштейна утворює ***циліндр***.
  - Кожний сектор має фіксований розмір та є найменшою одиницею передачі інформації.
  - До 2010 року найбільш популярний розмір сектору - 512 байтів, після – 4Кб.

Двигун жорсткого диску обертає диски на високій швидкості (5,400, 7,200, 10,000, 15,000 RPM – обертів/хв)

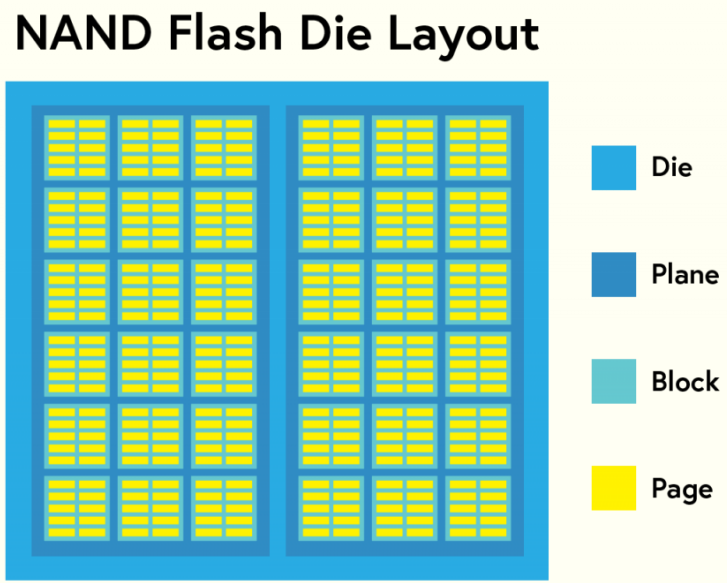


- Деякі ЖД починають обертання тільки при надходженні запиту на ввід-вивід.
  - Швидкість обертання пов'язана зі ***швидкістю передачі даних (transfer rate)*** – швидкістю обміну інформацією між ЖД та комп'ютером.
  - Інший аспект швидкодії – ***час довільного доступу (positioning time, random-access time)***, складається з 2 частин: часу переміщення кронштейну до бажаного циліндру (***час пошуку, seek time***) + час обертання потрібного сектору до плаваючої голівки ***(затримка обертання, rotational latency)***.
  - Звичайні диски можуть передавати десятки та сотні Мб/с, оскільки їх час пошуку та затримка обертання складають кілька мілісекунд.
  - Продуктивність підвищується розміщенням DRAM-буферів у контролері пристрою.
- Плаваючі голівки «літають» на мікронній відстані від поверхні дисків, прямий контакт небезпечний.
  - Хоч диски покриваються захисним прошарком, голівка іноді руйнує магнітну поверхню. Таку подію називають ***аварією голівки (head crash)***.
  - Відремонтувати таку аварію зазвичай неможливо; необхідно заміняти весь диск, а дані з нього втрачаються, якщо вони не були резервно скопійованими або захищеними RAID-масивом.

Енергонезалежні пристрої (Nonvolatile Memory Devices)

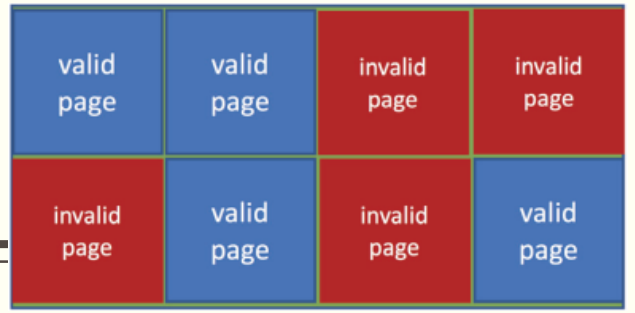
- NVM-пристрої на базі флеш-пам’яті часто використовуються в ***твердотільних накопичувачах (solid-state disk, SSD)***.
  - Інший приклад – USB-носії та плашки DRAM.
- NVM-пристрої можуть бути більш надійними, ніж ЖД, оскільки не мають рухомих частин, та швидкими, не маючи часу пошуку та затримки обертання.
  - Також вони менш енергозатратні.
- Оскільки NVM-пристрої можуть бути набагато швидшими за ЖД, стандартні інтерфейси шин можуть обмежувати пропускну здатність.
  - Деякі NVM-пристрої спроектовані для підключення напряму до системної шини, наприклад, через роз’єм PCIe.

Огляд енергонезалежних носіїв інформації



- NAND-пристрої мають власні виклики щодо надійності зберігання даних.
  - Наприклад, вони можуть зчитуватись та записуватись “посторінково” (подібно до секторів), проте дані неможливо перезаписати – NAND-комірки спочатку повинні затиратись (erase).
  - Затирання «блоку» розміром у кілька сторінок забирає набагато більше часу, ніж зчитування чи запис.
  - Виручає багаточіпова (many die) компоновка з багатьма datapath-ами на кожному чіпі, що дозволяє паралельно виконувати операції.
  - NAND-пристрої також деградують з кожним циклом затирання, а після близько 100 000 таких циклів більше не можуть тримати дані.
  - Тривалість життя NAND NVM-пристроїв вимірюється не в роках, а в ***кількості перезаписів на день (Drive Writes Per Day, DWPD)***.
  - Наприклад, NAND-пристрій обсягом 1Тб з 5 DWPD очікує перезапис 5Тб в день протягом гарантійного періоду до відмови.

Алгоритми контролера NAND Flash



- Оскільки NAND-пристрої не допускають перезапис після запису, вони зазвичай розбиваються на сторінки з валідними та невалідними даними.
  - Розглянемо блок файлової системи, записаний одного разу та записаний знову пізніше.
  - Якщо в цей проміжок часу не було затирання, старі дані зберігаються в сторінці, записаній спочатку та невалідній, а друга сторінка матиме поточну версію блоку.
  - Для відстеження того, які логічні блоки мають валідні дані, контролер підтримує спеціальний ***прошарок трансляції (flash translation layer, FTL)*** – таблицю, яка відображає фізичні сторінки, що містять поточні валідні логічні блоки.
  - Також відстежується стан фізичного блоку: які блоки містять тільки невалідні сторінки, тобто можуть затиратись.
- Уявіть заповнений SSD з очікуючим запитом на запис.
  - Усі сторінки записані, проте можуть бути блоки, які не містять валідних даних. Тоді може трапитись затирання таких блоків і подальший запис в них.
  - А якщо вільних блоків немає? Ще може бути вільний простір, якщо окремі сторінки містять невалідні дані.
  - Тоді збирач сміття може перемістити валідні дані в інші розташування, очищуючи блоки для подальшого затирання та запису.
  - Куди збирач сміття збереже валідні дані? NVM-пристрої застосовують технологію ***Over-Provisioning (OP)***.
  - Пристрій відбирає деяку кількість сторінок (spare area – зазвичай ~7% від загального простору) як область, завжди доступну для запису.

Енергозалежна пам’ять

- DRAM також може використовуватись як носій даних, зокрема RAM-диски.
  - Такі «носії» можуть використовуватись як raw block devices, проте частіше на них створюються файлові системи для стандартних операцій з файлами.
- Яка потреба в DRAM для тимчасового сховища даних?
  - Інформація на RAM-пристрої не переживає системний збій, відключення чи відімкнення живлення.
  - Кеші та буфери виділяються програмістом або ОС, а RAM-носії дозволяють користувачу (і програмісту) розмістити дані в пам’яті для тимчасового тримання за допомогою стандартних файлових операцій.
  - В Linux існує каталог /dev/ram, в macOS – команда diskutil створює RAM-сховище, Windows має сторонні інструменти для цього. Solaris та Linux створюють /tmp під час завантаження (тип “tmpfs”), що є RAM-носієм.
- RAM-носії корисні як високошвидкісні тимчасові сховища.
  - Хоч NVM-пристрої швидкі, DRAM ще швидша, тому операції вводу-виводу в RAM-пристроях – це найшвидший спосіб створювати, читати, записувати та видаляти файли і їх вміст.
  - Програми можуть ділитись даними, записуючи та зчитуючи файли з RAM-носія.
  - Linux під час завантаження створює тимчасову кореневу файлову систему (initrd), яка дозволяє іншим частинам системи мати доступ до свого вмісту ще до завантаження інших носіїв даних.

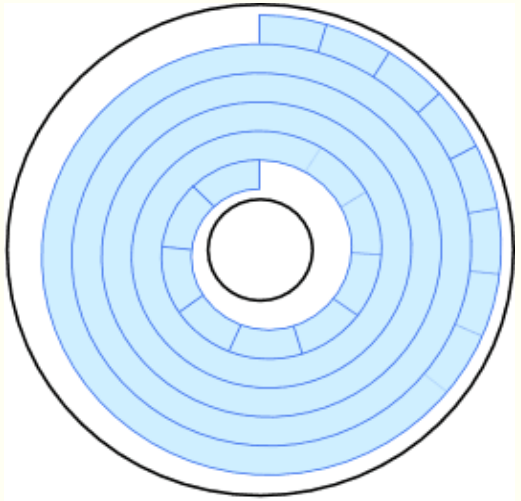
Моделі підключення вторинних сховищ

- Вторинне сховище підключається до комп'ютера по системній шині або ***шині вводу-виводу***.
  - Існує багато шин: **advanced technology attachment** (**ATA**), **serial ATA** (**SATA**), **eSATA**, **serial attached SCSI (SAS)**, **universal serial bus** (**USB**) та **fibre channel (FC)**.
  - Оскільки NVM-пристрої набагато швидші ЖД, було розроблено швидкий інтерфейс **NVM express** (**NVMe**).
  - NVMe напряму під'єднує пристрій до системної шини PCI, підвищуючи пропускну здатність та знижуючи затримку (latency) в порівнянні з іншими методами.
- Передача даних на шину відбувається за допомогою ***контролерів (controllers, host-bus adapters (HBA))***.
  - ***Контролер хосту*** знаходиться на кінці шини зі сторони комп'ютера, ***контролер пристрою*** – зі сторони носія.
  - Для виконання операції вводу-виводу з носієм даних комп'ютер розміщує команду в контролер хосту, зазвичай використовуючи memory-mapped порти вводу-виводу.
  - Контролер хосту надсилає команду з повідомленнями контролеру пристрою, який оперує носієм для її виконання.
  - Контролери пристрою зазвичай мають вбудований кеш. Передача даних на пристрої здійснюється між кешем та сховищем, а передача даних на хост – у cache host DRAM по DMA.

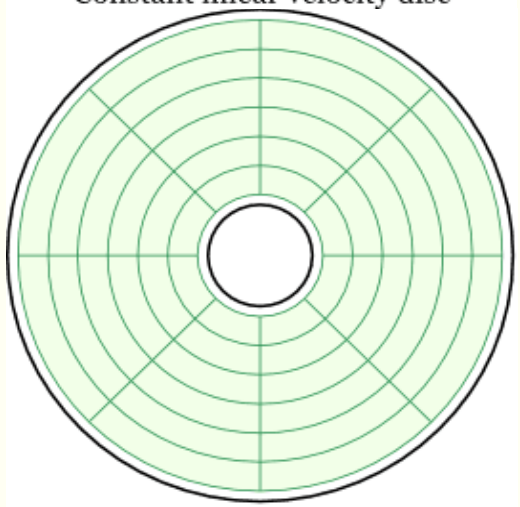
Відображення адрес

- Носії даних адресуються як великі одновимірні масиви ***логічних блоків***,- найменших одиниць передачі.
  - Кожний логічний блок відображається на фізичний сектор або напівпровідникову сторінку.
  - Наприклад, сектор 0 може бути першим сектором першої доріжки outermost циліндра на ЖД.
  - Відображення продовжується вздовж доріжки, а потім по решті доріжок циліндру, далі – по решті циліндрів, від найвіддаленішого від центру до найближчого.
  - Для NVM відображення відбувається з кортежу (впорядкованого списку) елементів (chip, block, page) на масив логічних блоків. Адресу логічного блоку (**LBA**) простіше використовувати в алгоритмах.
- Використовуючи таке відображення для ЖД, теоретично можливе перетворення номеру логічного блоку в old-style адресу – (номер циліндра, номер доріжки, номер сектору).
  - На практиці це складно, оскільки:
    - 1) більшість носіїв мають дефектні сектори, що не відображається при перетворенні. Адреси логічних блоків послідовні, а розташування фізичних секторів змінюється.
    - 2) кількість секторів на доріжку в деяких носіях неконстантна.
    - 3) виробники дисків зсередини керують відображенням LBA на фізичну адресу, тому поточні пристрої мають слабкий зв’язок між LBA та фізичними секторами.
  - Проте алгоритми, які працюють з ЖД, часто припускають відносний зв’язок між логічними адресами та відповідними фізичними адресами, тому зростаючі логічні адреси, ймовірно, відповідатимуть зростаючим фізичним адресам.

Детальніше розглянемо 2гу причину



Constant linear velocity disc



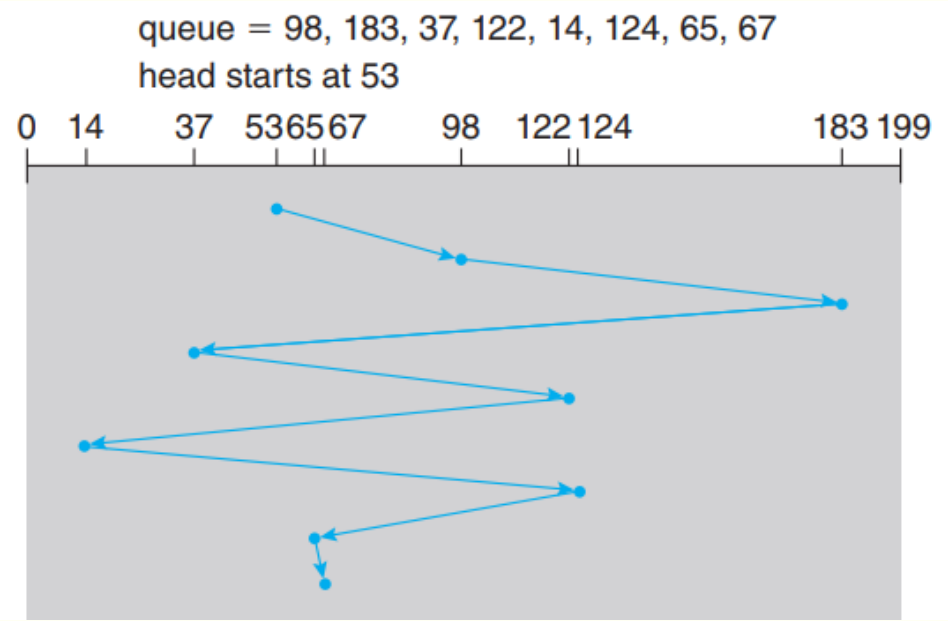
Constant angular velocity disc

- При використанні ***сталої лінійної швидкості (constant linear velocity, CLV)***, щільність бітів на доріжку однакова.
  - Чим далі доріжка від центру диску, тим вона довша і може вмістити більше секторів (до ~40%). При русі до центру кількість секторів на доріжку зменшується.
  - Накопичувач прискорює обертання диску по мірі проходження голівки від зовнішніх до внутрішніх доріжок, щоб підтримувати однакову швидкість переміщення даних під голівкою.
  - Метод використовується в CD-ROM та DVD-ROM.
- Альтернатива: швидкість обертання диску стала.
  - Тоді щільність бітів зменшується від внутрішніх доріжок до зовнішніх to keep the data rate constant (and performance relatively the same no matter where data is on the drive).
  - Цей метод використовується у жорстких дисках та називається ***сталою кутовою швидкістю (constant angular velocity, CAV)***.
- Кількість секторів на доріжку зростала з часом, нині зовнішня зона диску має кілька сотень секторів на доріжці. Аналогічно і з циліндрами – нині їх десятки тисяч на диску.
- Зауважте, що існує ще багато видів накопичувачів, які недоцільно розглядати в межах даного курсу.
  - Наприклад, існують “shingled magnetic recording” ЖД з вищою щільністю, проте гіршою швидкодією, ніж популярні жорсткі диски.
  - Існують комбіновані пристрої, які включають технології NVM та HDD, або менеджери томів, які можуть «зшити» NVM- та HDD-пристрої у накопичувач з усередненими характеристиками.

Планування роботи жорстких дисків (HDD Scheduling)

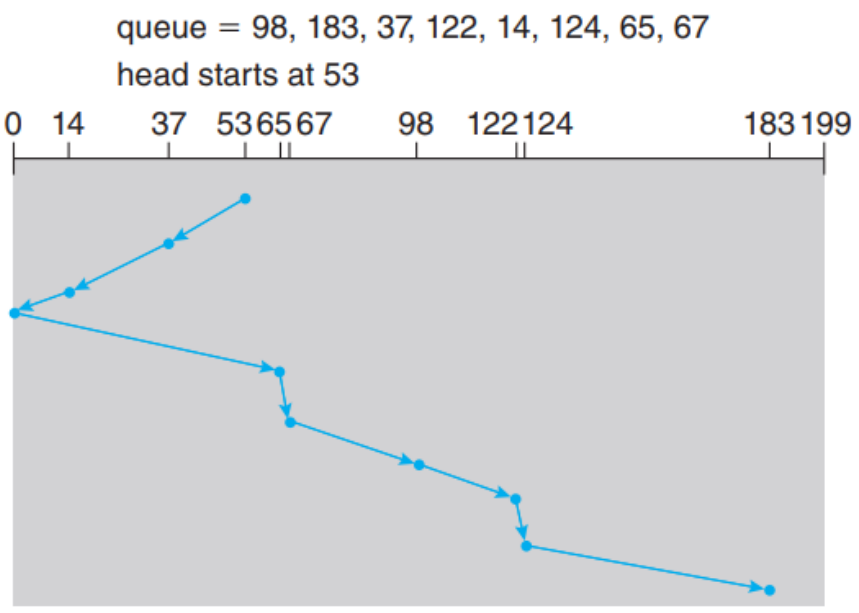
- Для ЖД та інших механічних накопичувачів на основі пластин, ***час доступу*** формується з:
  - 1) ***Часу пошуку (seek time)*** – час, потрібний на переміщення голівок на циліндр з шуканим сектором;
  - 2) ***Затримка обертання (rotational latency)*** – додатковий час на обертання пластини для досягнення шуканого сектору.
- **Теоретична пропускна здатність (Bandwidth) пристрою** – загальна кількість переданих байтів, поділена на проміжок часу від першого запиту до завершення останньої передачі.
  - Покращити час доступу та теоретичну пропускну здатність можна, керуючи порядком обробки storage I/O requests.
- При потребі вводу-виводу на/з диску процес застосовує системний виклик до ОС. Запит визначає кілька частин інформації:
  - Це операція вводу чи виводу
  - Хендлер відкритого файлу, який вказує на файл, з яким потрібно працювати
  - Яка адреса пам’яті для передачі
  - Обсяг даних для передачі
- Якщо бажані накопичувач та контролер доступні, запит може оброблятись негайно.
  - Інакше будь-який новий запит на обслуговування буде розміщено в чергу очікуючих (pending) запитів для пристрою.
  - У багатопроцесних системах черга пристрою часто може мати кілька очікуючих запитів.
  - Існування черги запитів до пристрою допомагає оптимізувати продуктивність роботи, уникаючи надмірного переміщення голівок. Драйвери пристрою можуть підвищити швидкодію, впорядковуючи чергу.

FCFS (FIFO)-планування – найпростіша версія планування роботи ЖД



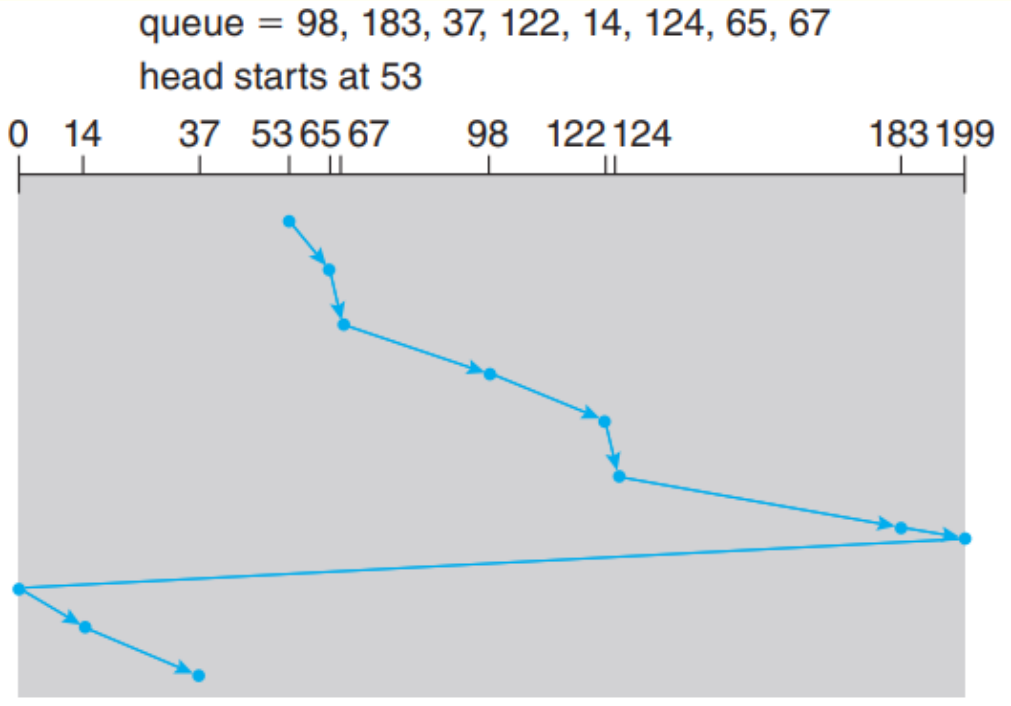
- Алгоритм справедливий, проте не найшвидший.
  - Наприклад, має по порядку запити на ввід-вивід до блоків на циліндрах: 98, 183, 37, 122, 14, 124, 65, 67.
  - Якщо голівка стартує на циліндрі 53, вона спочатку пересунеться в 98, потім у 183, 37, 122, 14, 124, 65 та 67, загальне переміщення голівки складе 640 циліндрів.
- Зображення демонструє проблему планування – біганину зі сторони в сторону.
  - Якби запити до циліндрів 37 та 14 могли оброблятись разом, до або після запитів до циліндрів 122 та 124, загальне переміщення голівки можна було б суттєво зменшити, таким чином підвищивши швидкодію.

SCAN-планування



- У даному алгоритмі кронштейн розпочинає з одного кінця диску та просувається до іншого кінця, обробляючи запити по досягненню кожного циліндру з переліку.
  - На іншому кінці напрямок руху голівки змінюється на зворотний, а обробка продовжується.
  - Голівка послідовно сканує циліндри вздовж диску.
  - SCAN-алгоритм інколи називають ***алгоритмом ліфта (elevator algorithm)***, оскільки кронштейн поводиться як ліфт у будівлі.
- До застосування SCAN-планування запитів до циліндрів необхідно знати напрямок руху голівки разом з поточною позицією голівки.
  - Припустимо, що кронштейн рухається в сторону нуля, а початкова позиція голівки - 53. Далі голівка обслуговує циліндр 37, а потім 14.
  - У циліндрі о кронштейн розвертається в іншу сторону, обслуговуючи запити на циліндрах 65, 67, 98, 122, 124 та 183.
  - Якщо запит надходить у чергу прямо перед прольотом голівки, він буде оброблений негайно, а запит позаду неї буде очікувати, поки кронштейн знову розверне голівки назад та добереться до нього.

Що буде, коли немає достатньо пам'яті для забезпечення потреб процесу, який надійшов?



- Нехай розподіл запитів щодо циліндрів буде рівномірним.
  - На зворотному ході перед голівкою буде з'являтись відносно мало запитів, оскільки ці циліндри нещодавно були обслужені.
  - Найбільша щільність запитів знаходиться на іншому кінці диску.
  - Також ці запити чекають найдовше, тому ідея наступного алгоритму – спочатку переходити до цих запитів.
- **Circular SCAN (C-SCAN) планування** – варіант SCAN, спроектований для забезпечення більш рівномірного часу очікування.
  - Як і SCAN, переміщує голівку від одного кінця диску до іншого, обробляючи запити по дорозі.
  - Коли голівка досягає іншого кінця, вона негайно повертається до початку диску, не обслуговуючи запити протягом цього шляху.

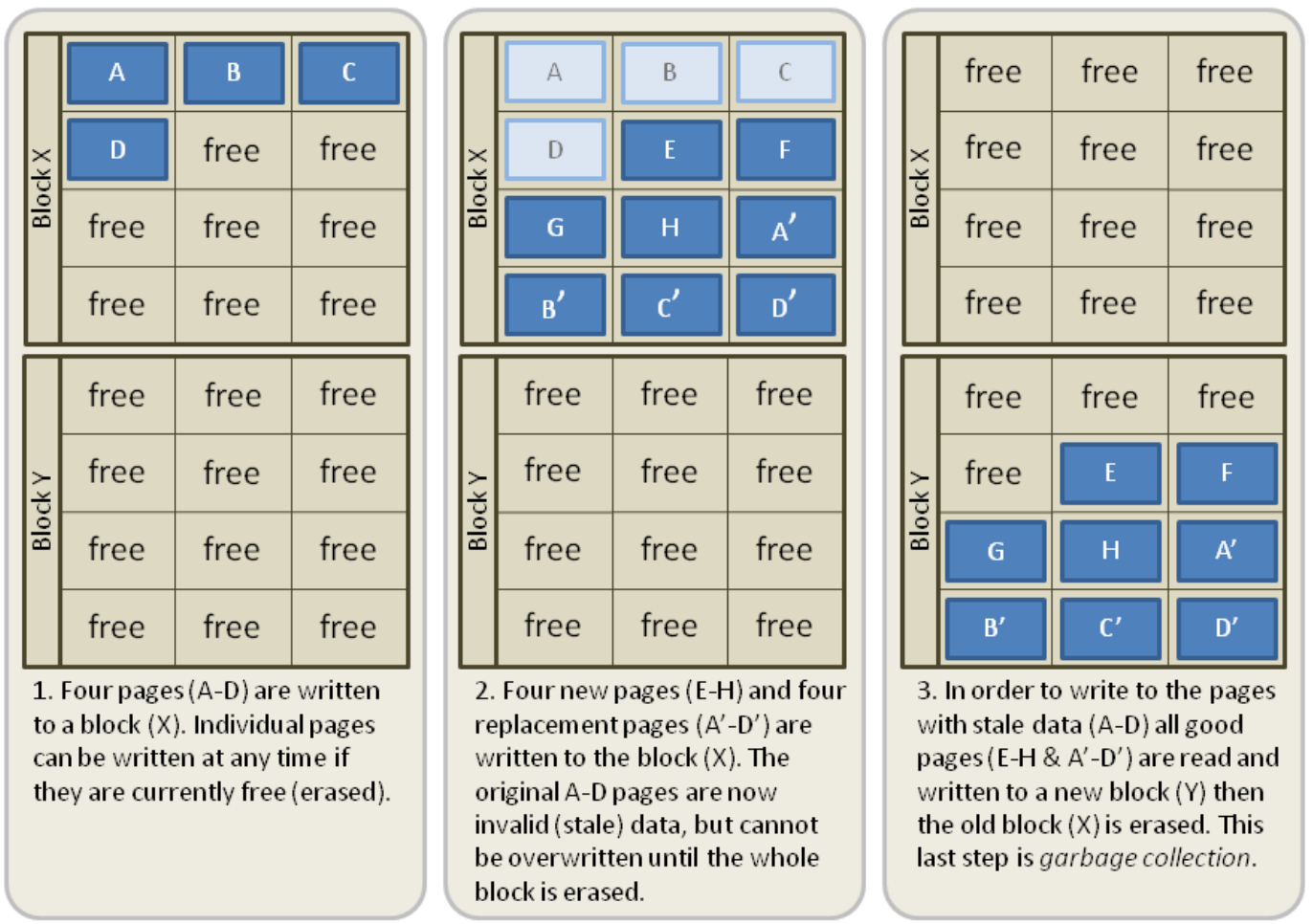
Вибір алгоритму планування дискової роботи

- Багато алгоритмів не були включені в розгляд, оскільки вони рідко використовуються.
  - Для будь-якого конкретного списку запитів можна визначити оптимальний порядок отримання даних, проте відповідні обчислення невиправдані.
  - Для алгоритмів планування продуктивність дуже залежить від кількості та типу запитів.
  - Наприклад, нехай черга зазвичай має тільки 1 outstanding запит. Тоді всі алгоритми будуть поводитись однаково – як FCFS-планування.
- SCAN та C-SCAN краще працюють в системах з високим навантаженням на диск, оскільки для них проблема голодування (starvation) problem менш імовірна.
  - Голодування все ж може бути, що призвело до створення в Linux **deadline-планувальника**. Він тримає окремі черги зчитування та запису, віддаючи пріоритет зчитуванням, оскільки блокування процесів ймовірніше при читанні, ніж запису.
  - Черги сортуються в порядку LBA, в основі лежить C-SCAN. Всі запити на ввід-вивід надсилаються в пакет (batch) в LBA-порядку.
  - Deadline має по 2 черги на зчитування та запис: одна сортується за LBA, інша – за FCFS.
  - Після кожного пакету перевіряється, чи є запити в FCFS-чергах, старші за заданий вік (за умовчанням 500мс). Якщо так, то LBA-черга (read або write), яка містить цей запит, обирається для отримання наступного пакету вводу-виводу.
- Deadline-планувальник вводу-виводу за умовчанням постачається в Linux RedHat 7, проте **RHEL** 7 включає ще 2.
  - **NOOP** краще працює в CPU-bound системах, використовуючи швидкі сховища на зразок NVM-носіїв.
  - **Completely Fair Queueing scheduler (CFQ)** – за умовчанням для SATA-пристроїв.

Планування роботи NVM-пристроїв

- NVM-пристрої не мають рухливих голівок та зазвичай застосовують просту FCFS-політику.
  - Наприклад, планувальник **NOOP** в Linux застосовує FCFS-політику, доповнюючи її злиттям (merge) суміжних запитів.
  - За результатами спостережень час обслуговування зчитувань рівномірний, проте це так через властивості flash-пам'яті, для запису такий час нерівномірний. Деякі планувальники SSD використовують цю особливість та з'єднують тільки суміжні запити на запис, обслуговуючи всі запити на зчитування в порядку FCFS.
- Ввід-вивід може відбуватись послідовно або довільно.
  - Послідовний доступ оптимальний для механічних пристроїв (HDD, магнітна стрічка тощо).
  - Ввід-вивід з довільним доступом, що виміряється в операціях вводу-виводу за секунду (**IOPS**) набагато швидший для NVM.
- NVM-пристрої мають набагато нижчу пропускну здатність при послідовному вводі-виводі, приблизно на порядок у порівнянні з ЖД при зчитуванні (запис ще повільніший).
  - Крім того, продуктивність запису NVM-носіїв залежить від заповненості його обсягу та «зношеності».
  - NVM-пристрій наприкінці життя має набагато гіршу продуктивність через велику кількість циклів затирання.
  - Один зі способів покращити тривалість життя та продуктивність NVM-накопичувачів – щоб файлова система інформувала пристрій про видалення файлів, щоб накопичувач міг затерти блоки цих файлів.

Планування роботи NVM-пристроїв



- Детальніше розглянемо вплив збирання сміття на продуктивність.
  - Нехай NVM-носій навантажується операціями довільного зчитування та запису.
  - Припустимо, що всі доступні блоки треба записати, проте ще є вільний простір.
  - Збирання сміття відбувається для повернення простору, забраного невалідними даними.
  - Загалом, один запит на запис спричинить запис сторінки (даних), одне або кілька зчитувань (збирачем сміття), один або кілька записів сторінок (хороші дані з блоків, зібраних збирачем сміття).
  - Створення запитів на ввід-вивід не додатками, а NVM-накопичувачем під час збирання сміття та керування простором, називається **посиленням запису (write amplification)** та може сильно вплинути на продуктивність операцій записування в пристрої.

Виявлення та корекція помилок

- **Error detection** визначає прояв проблеми — наприклад, біт у DRAM спонтанно інвертувався, вміст мережевого пакету змінився протягом передачі тощо.
  - Визначаючи проблему, система може зупинити операцію до того, як помилка поширюється, прозвітувати про помилку користувачеві чи адміністратору або попередити про пристрій, що почав відмовляти або вже відмовив.
- Системи пам’яті вже довго знаходять деякі помилки, використовуючи ***біти парності (parity bits)***.
  - Так, кожний байт системи пам’яті має біт парності пов’язаний з кількістю одиниць у байті (parity = 0 для парної кількості одиниць та parity = 1 для непарної).
  - Пошкодження значення одного біту (зокрема і біту парності) призводить до зміни біту парності.
  - Однобітові помилки система відстежує, двобітові можуть пройти непоміченими.
  - Парність просто обчислюється за допомогою оператора XOR, а для кожного байту пам’яті тепер потрібен додатковий біт.
  - Біт парності – форма ***контрольних сум (checksums)***, яка використовує модулярну арифметику для обчислення, зберігання та порівняння значень з фіксованою довжиною слів.
  - Інший метод виявлення помилок, поширений в мережевому програмуванні, - ***циклічні надлишкові коди (cyclic redundancy check, CRC)***, який використовує хеш-функцію для знаходження мультибітних помилок.

Корекція помилок

- ***Коди корекції помилок (error-correction code, ECC)*** не лише виявляють проблему, а й виправляють її.
  - Коди варіюються залежно від потрібного їм додаткового простору та кількості виправлених помилок.
  - Наприклад, дискові накопичувачі використовують per-sector ECC, а flash-накопичувачі – per-page ECC.
  - Коли контролер записує сектор/сторінку даних протягом вводу-виводу, в ECC автоматично записується обчислене за всіма байтами даних значення.
  - При зчитуванні ECC заново автоматично обчислюється та порівнюється із збереженим значенням.
  - ECC коректуючий, оскільки містить достатньо інформації для відновлення даних контролером, якщо лише кілька бітів були пошкоджені.
  - Контролер тоді звітує про відновлювану ***м'яку помилку (soft error)***. Якщо змін трапилось надто багато, сигналізується про невідновлювану ***жорстку помилка (hard error)***.

Керування накопичувачем

- До того, як чистий накопичувач зможе зберігати дані, його необхідно розбити на сектори, які контролер зможе зчитувати та записувати.
  - NVM-сторінки повинні ініціалізуватись, а також створюється прошарок трансляції (FTL).
  - Цей процес називають ***низькорівневим, або фізичним форматуванням***.
  - Пристрій заповнюється спеціальною структурою даних для кожного storage location.
  - Структура даних для сектору або сторінки зазвичай містить заголовок, область даних та trailer.
  - Заголовок та trailer містять інформацію, що використовується контролером: номер сторінки/сектору, код виявлення помилок або корекційний код тощо.
- Більшість накопичувачів піддаються низькорівневому форматуванню в процесі їх виробництва на фабриці.
  - Форматування потрібне виробникові для тестування пристрою та знаходження вільних від ефектів секторів або сторінок.
  - Часто дається можливість обирати розміри сектору, як 512 байтів або 4Кб.
  - Форматування диску більшими секторами означає меншу їх кількість на доріжці, меншу кількість заголовків та trailer-ів, що записуються на доріжку, тобто більше місця для даних користувачів.
  - Деякі ОС можуть працювати лише з одним заданим розміром сектору.

Перед використанням файлів ОС ще потрібно записати власні структури даних на пристрій

- Виконується за три кроки:
- 1) ***Розбиття (partition)*** пристрою на розділи – групи блоків або сторінок.
  - ОС може працювати з розділами, як з окремими пристроями.
  - Наприклад, розділ може містити файлову систему, всередині якої буде копія виконуваного коду ОС, інший swap-простір та інша файлова система з файлами користувачів.
  - Деякі ОС та файлові системи виконують розбиття автоматично, коли весь пристрій керуватиметься файловою системою.
  - Інформація щодо розбиття записується у фіксованому форматі в фіксованому розташуванні накопичувача.
  - У Linux команда fdisk використовується для управління розділами на накопичувачі.
  - При розпізнаванні пристрою операційною системою зчитується його інформація щодо розбиття, а потім ОС створює device entries для розділів (у Linux – в директорії /dev).
  - Далі конфігураційний файл (/etc/fstab) вказує ОС монтувати кожний розділ, який містить файлову систему в визначеному розташуванні, із заданими опціями монтування на зразок read-only.
  - ***Монтування (Mounting)*** файлової системи – це надання доступності файловій системі для використання ОС та її користувачами.

Виконується за три кроки:

- 2) Створення томів (volume) та керування ними.
  - Інколи даний крок неявний, як тоді, коли файлова система напряму розташовується в розділі. Тоді том готовий до монтування та використання.
  - В інших випадках створення та управління томом явне. Наприклад, коли кілька розділів або носіїв будуть використовуватись разом як RAID-масив з однією або багатьма файловими системами розповсюдженими на пристроях.
  - ***Linux volume manager (lvm2)*** може надати ці можливості, як і комерційні сторонні інструменти для Linux чи інших ОС.
  - ***ZFS*** забезпечує як керування томами, так і інтегровану в набір команд та можливостей файлову систему.
- 3) ***Логічне форматування*** – створення файлової системи.
  - На цьому кроці ОС зберігає початкові структури даних файлової системи на носій.
  - Ці структури даних можуть включати карти вільного та виділеного простору, а також первинну порожню директорію.

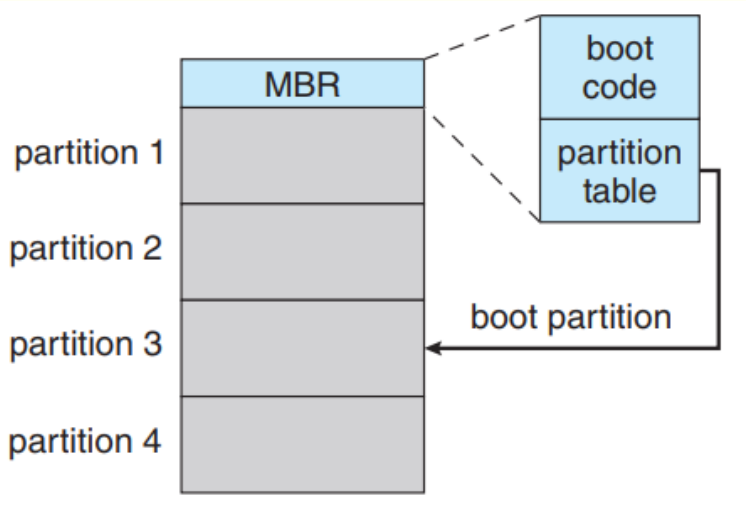
Інформація щодо розбиття також визначає, чи містить розділ завантажувальну файлову систему (з ОС всередині)

- Позначений для завантажування розділ використовується для встановлення кореню файлової системи.
  - Як тільки він монтується, носій посилається (link) на всі інші пристрої та їх розділи.
  - У Windows вони окремо іменовані літерами (C:, D:, E:), на інших системах, як Linux, під час завантаження boot-файлова система монтується, а інші файлові системи можуть монтуватись у деревовидну структуру.
- Для підвищення ефективності більшість файлових систем групують блоки у більші chunk-и, які часто називають ***кластерами***.
  - Ввід-вивід для пристрою здійснюється через блоки, проте ввід-вивід для файлової системи виконується кластерно, забезпечуючи ввід-вивід з переважаючим послідовним доступом.
  - Файлові системи намагаються групувати вміст файлів поблизу з їх метаданими, таким чином зменшуючи час пошуку для голівки ЖД при роботі з файлом.
- Деякі ОС надають спеціальним програмам можливість використовувати розділ як великий послідовний масив логічних блоків, без жодних структур даних файлової системи.
  - Такий масив інколи називають ***сирим диском (raw disk)***, а ввід вивід з цим масивом – ***сирим вводом-виводом (raw I/O).***
  - Може використовуватись, наприклад, для відкачування даних. Деякі системи БД віддають перевагу сирому вводу-виводу завдяки контролю точного розташування в пам’яті кожного запису БД.
  - Сирий ввід-вивід обходить всі служби файлової системи, такі як buffer cache, блокування файлів, prefetching, виділення простору, назви файлів та папки.

Блок завантаження (Boot Block)

- Для запуску комп’ютера потрібна початкова програма для роботи – завантажувач ***(bootstrap loader)***.
  - Більшість комп’ютерів зберігають її в прошивці в NVM-пам’яті на материнській платі та відображають у відоме розташування.
  - Програма ініціалізує всі аспекти системи, від регістрів ЦП до контролерів пристроїв та вмісту основної пам’яті.
- Крихітний завантажувач достатньо розумний, щоб отримати повноцінну програму для завантаження ОС із вторинного накопичувача.
  - Даний повноцінний завантажувач зберігається в “boot blocks” у фіксованому розташуванні накопичувача.
  - За умовчанням у Linux використовується завантажувач grub2.
  - Пристрій, який має завантажувальний розділ, називається ***завантажувальним*** або ***системним диском***.
- Код у bootstrap NVM інструктує контролер накопичувача зчитувати завантажувальні блоки в пам’ять (драйвери ще не завантажуються), а потім починає виконувати цей код.
  - Повна завантажувальна програма більш складна: вона здатна завантажити всю ОС, починаючи з нефіксованого розташування на пристрої, та запустити її.

Приклад завантажувального процесу в Windows



- Windows дозволяє носію розбиватись на розділи та визначає один розділ як ***завантажувальний розділ (boot partition)***, що містить ОС та драйвери пристроїв.
  - Windows розміщує свій код для завантаження у першому логічному блоці ЖД або першій сторінці NVM-носія, яку називають ***головним завантажувальним записом (master boot record, MBR)***.
  - Завантаження починається з запуску коду з системної прошивки, який спрямовує систему на зчитування boot-коду з MBR, знаючи лише про контролер пристрою та носій з сектором для завантаження.
  - Крім boot-коду, MBR містить таблицю розділів та прапорець, який вказує на завантажувальний розділ.
- Як тільки система визначає завантажувальний розділ, він зчитує перший сектор/сторінку (**boot sector**), що спрямовує її до ядра.
  - Далі продовжується робота з рештою процесу завантаження, яка включає завантаження різних підсистем та системних служб.

Bad Blocks

- Оскільки диски мають рухливі частини, вони схильні до відмов.
  - Часто дефективними стають один або кілька секторів.
  - Більшість дисків навіть постачаються з фабрики з ***битими блоками (bad blocks)***.
  - Залежно від диску та контролера, ці блоки обробляються по-різному.
- Старі диски биті блоки обробляли вручну.
  - Одна стратегія – відсканувати диск на биті блоки при форматуванні. Усі знайдені биті блоки позначаються як неможливі до використання, тому файлова система не виділяє їх.
  - Якщо блоки псуються протягом звичайної операції, спеціальна програма (як Linux-команда badblocks) повинна вручну запускатись для пошуку битих блоків та їх блокування.
  - Дані з битих блоків зазвичай втрачаються.
- Більш «розумні» диски пропонують способи відновлення битих блоків.
  - Контролер підтримує список битих блоків на диску.
  - Список ініціалізується низькорівневим форматуванням на етапі виробництва та оновлюється протягом життя диску. Low-level formatting also sets aside spare sectors not visible to the operating system.
  - Контролеру можна вказати логічно замінити кожний битий сектор на відповідні запасні (spare) сектори.
  - Дану схему називають ***заміною сектору (sector sparing)*** або ***форвардинг (forwarding)***.

Альтернатива: заміна битого блоку методом sector slipping

- Нехай логічний блок 17 стає дефектним, а першим доступним запасним сектором є 202.
  - Sector slipping зсуває всі наступні сектори до 202 на 1 вперед.
- Відновлювані м’які помилки можуть спричинити такі дії пристрою, за яких робиться копія даних блоку, and the block is spared or slipped.
  - Невідновлювана жорстка помилка призводить до втрати даних. Відновлення відповідного файлу вимагатиме ручного втручання.
- NVM-пристрої також мають біти, байти та навіть сторінки, які є нефункціональними вже на етапі виробництва або псуються з часом.
  - Керування такими збійними областями простіше, ніж для ЖД, оскільки не потрібно усувати втрату продуктивності, пов’язану з часом пошуку (seek time).
  - Або можна «відкласти» набір сторінок та використовувати їх як replacement-розташування, або використовувати простір з області over-provisioning (зменшуючи доступний обсяг цієї області).
  - В обох випадках контролер підтримує таблицю пошкоджених сторінок та ніколи не дає записувати в них дані.

Управління Swap-простором

- Ще одна низькорівнева задача ОС.
  - Віртуальна пам’ять використовує вторинне сховище як розширення основної пам’яті.
  - Оскільки доступ до пристрою набагато повільніший, ніж доступ до пам’яті, використання swap-простору значно знижує продуктивність системи.
  - Основна мета для проектування та реалізації swap-простору – забезпечити найкращу пропускну здатність для системи віртуальної пам’яті.
- Swap-простір використовується різними способами в різних ОС залежно від використовуваних алгоритмів управління пам’яттю.
  - Наприклад, системи, які реалізують підкачування даних, можуть використовувати swap-простір для зберігання всього образу процесу, включаючи сегменти коду та даних.
  - Системи зі сторінковою організацією пам’яті можуть просто зберігати сторінки, виштовхнуті з основної пам’яті.
  - Обсяг потрібного для системи swap-простору може варіюватись від кількох мегабайтів до гігабайтів, залежно від обсягу фізичної пам’яті, обсягу підтримуваної віртуальної пам’яті та способу її використання.

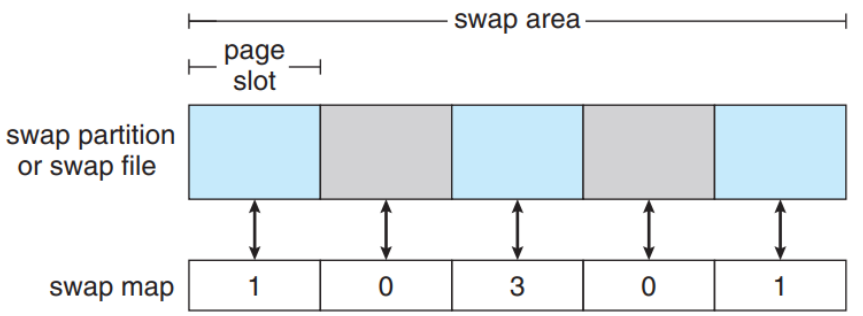
Використання Swap-простору

- Можливо, буде безпечніше переоцінити розмір потрібного swap-простору, оскільки за його нестачі система може бути змушена скасовувати роботу процесу чи повністю відмовляти.
  - Деякі системи рекомендують, щоб також виділявся простір поза swap-простором.
  - Наприклад, Solaris передбачає встановлення swap-простору, рівного за обсягом різниці розмірів віртуальної та фізичної пам’яті.
  - У минулому Linux пропонував задавати swap-простір вдвічі більший за фізичну пам’ять.
  - Нині алгоритми підкачування змінились, і більшість Linux-систем використовують набагато менше swap-простору.
- Деякі ОС, зокрема й Linux, дозволяють використання багатьох swap-просторів, включаючих як файли, так виділені swap-розділи.
  - Такі swap-простори зазвичай розташовуються на окремих накопичувачах, тому навантаження на систему вводу-виводу від paging та swapping може поширюватись до пропускної здатності системи вводу-виводу.

Розташування Swap-простору

- Swap-простір може розміщуватись в одному з 2 місць: він може «вирізатись» з нормальної файлової системи або знаходитись в окремому розділі.
  - Якщо swap-простір – просто великий файл у файловій системі, її підпрограми можна використовувати для створення, іменування та виділення цього простору.
- Як альтернатива, swap-простір може створюватись в окремому ***сирому (raw) розділі***.
  - Використовується окремий менеджер swap-простору для виділення та вивільнення блоків, який використовує оптимізовані під швидкодію алгоритми, оскільки доступ до swap-простору набагато частіший, ніж до файлових систем.
  - Внутрішня фрагментація може наростати, проте це допустимо, оскільки тривалість зберігання даних у swap-просторі загалом набагато коротша, ніж для файлів у файловій системі.
  - Swap-простір реініціалізується при завантаженні системи, тому фрагментація зазвичай короткотривала.
  - Розширення swap-простору вимагає або перерозбиття пристрою (переміщення інших розділів файловох системи або їх знищення та відновлення з резервної копії), або додавання іншого swap-простору де-небудь.
- Деякі ОС дозволяють підкачувати дані як у сирі розділи, та і файлову систему.
  - Наприклад, Linux: політика та реалізація окремі, дозволяючи адміністратору обирати тип свопінгу.
  - Це компроміс між зручністю виділення та управління простором у файловій системі та продуктивністю свопінгу в сирих розділах.

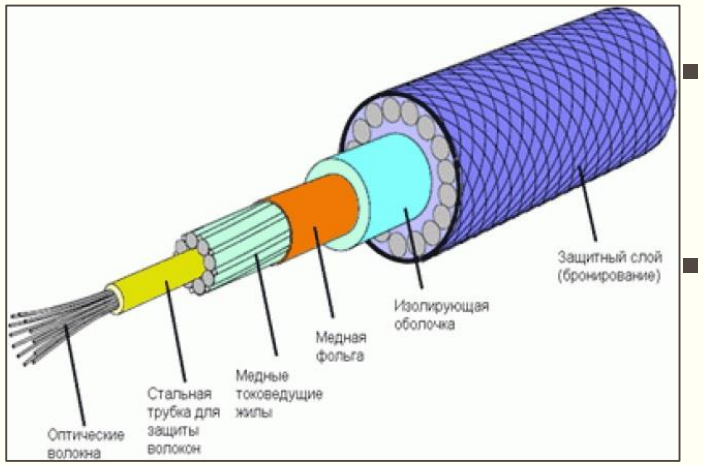
Приклад управління Swap-простором

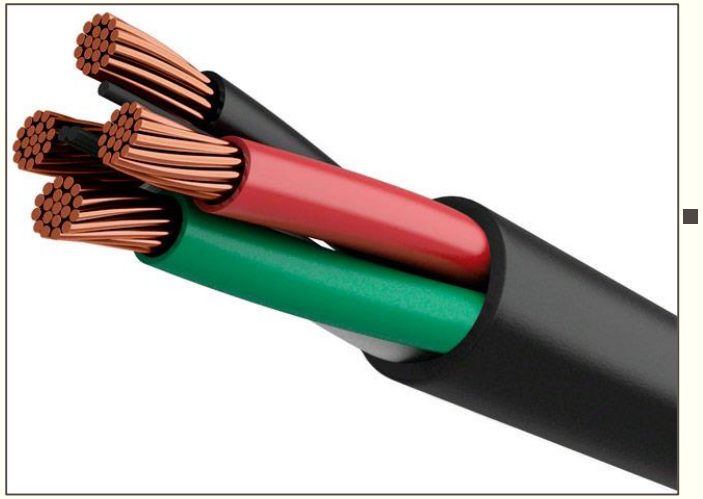


- Ядро традиційної UNIX реалізувало свопінг, копіюючи цілі процеси між послідовними областями диску та пам’яті.
  - Пізніше UNIX еволюціонувала до комбінації свопінгу та сторінкової організації, коли з’явилось відповідне апаратне забезпечення.
  - У Solaris 1 (SunOS) проектувальники змінили стандартні методи UNIX для покращення ефективності. Коли процес виконується, сторінки сегменту тексту (код) переносяться з файлової системи, доступної в основній пам’яті, та викидаються, якщо обрано pageout.
  - Ефективніше перечитати сторінку з файлової системи, ніж записати її в swap-простір, а потім перечитувати звідти.
  - Swap-простір використовується тільки як файл відкачки (backing store) для сторінок **анонімної** пам’яті (не підкріпленої файлом), що включає пам’ять, виділену під стек, кучу та неініціалізовані дані процесу.
  - Нині Solaris виділяє swap-простір тільки тоді, коли сторінка «викинута» з фізичної пам’яті, а не при першому створенні сторінки у віртуальній пам’яті. Ця схема більш продуктивна на сучасних комп’ютерах з більшою кількістю фізичної пам’яті.

- Linux подібна до Solaris в тому, що swap-простір тепер використовується тільки для анонімної пам’яті.
  - Linux дозволяє встановлювати кілька ***областей підкачки (swap area)*** - або swap-файл звичайної файлової системи, або виділений swap-розділ.
  - Кожна область підкачки складається з ряду 4-Кб-них ***сторінкових слотів (page slots)***, в яких тримаються відкачані сторінки.
  - З кожною областю підкачки пов’язана ***карта підкачки (swap map)***—масив цілочисельних лічильників, кожний з яких відповідає сторінковому слоту з області підкачки.
  - Якщо значення лічильника рівне 0, відповідний сторінковий слот доступний.
  - Значення, більші за 0, вказують на зайнятість сторінкового слоту відкачаною сторінкою. Числове значення вказує на відображення відкачаної сторінки на відповідну кількість процесів (трапляється, коли відкачана сторінка зберігає область спільної пам’яті для процесів).

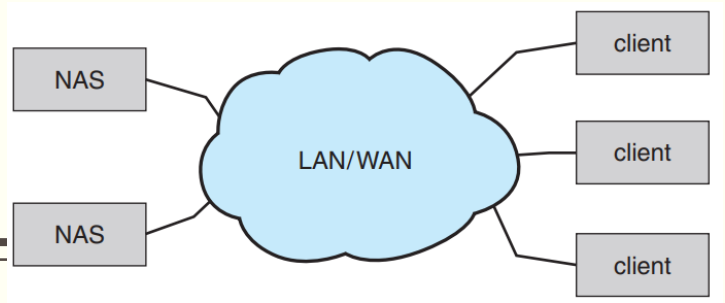
Підключення сховища (Storage Attachment)





- Отримання доступу до вторинного накопичувача відбувається трьома способами: через підключене до хосту сховище, мережеве (network-attached storage) або хмарне сховища.
- **Host-attached storage –** це сховище, доступ до якого здійснюється через локальні порти вводу-виводу.
  - Найбільш поширена технологія підключення – SATA.
  - Окремі накопичувачі підключаються за технологіями USB, FireWire або Thunderbolt.
  - Високопродуктивні робочі станції та сервери загалом потребують більше пам’яті та спільний доступ до сховищ, тому використовуються складніші архітектури вводу-виводу, на зразок **fibre channel** (**FC**), - високошвидкісної послідовної архітектури, яка може оперувати по оптоволокну або 4-жильному мідному кабелю.
- Для підключення доступні багато host-attached накопичувачів: ЖД, NVM-пристрої, CD, DVD, Blu-ray, магнітні стрічки, мережі зберігання даних (storage-area networks, SAN).
  - Командами вводу виводу для ініціації передачі даних на host-attached накопичувач є зчитування та записи логічних блоків даних, спрямованих до конкретно визначених storage units (як bus ID або target logical unit).

Network-Attached Storage (NAS)

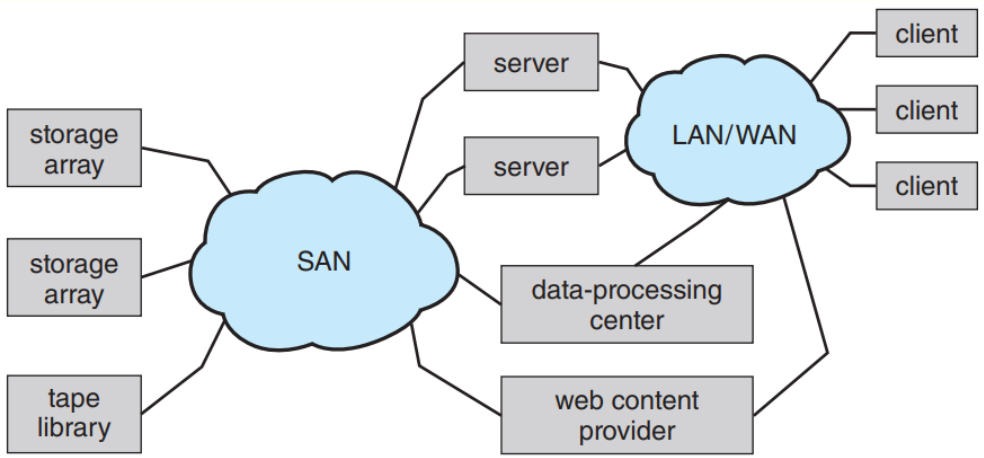


- Забезпечує доступ до сховища в мережі.
  - NAS-сервер може бути системою зберігання даних спеціального призначення або загальною комп'ютерною системою, яка постачає своє сховище інших хостам у мережі.
  - Доступ клієнтів до NAS відбувається через інтерфейс віддаленого виклику процедур (RPC), зокрема NFS для UNIX та Linux або CIFS для Windows-машин.
  - RPC-процедури передаються за протоколом TCP або UDP по IP-мережі—зазвичай тій же LAN-мережі, по якій передається весь трафік для клієнтів.
  - CIFS та NFS надають різні можливості блокування (locking), дозволяючи ділитись файлами між хостами з доступом до NAS за цими протоколами. Наприклад, користувач, авторизований у кількох клієнтах NAS, може отримати доступ до домашньої директорії з усіх цих клієнтів одночасно.
- iSCSI – найновіший протокол NAS.
  - Мережі, а не SCSI-кабелі, можуть використовуватись як інтерконнекти між хостами та сховищем.
  - У той час, як NFS та CIFS представляють файлову систему та надсилають частини файлів по мережі, iSCSI надсилає по мережі логічні блоки та залишає їх клієнту для прямого використання блоків або створення з них файлової системи.

Хмарне сховище

- Подібно до NAS, хмарні сховища надають доступ до сховища через мережу, проте сам доступ відбувається через Інтернет або іншу WAN-мережу до віддаленого датацентру.
- Ще одна відмінність від NAS – доступ та представлення сховища для користувачів.
  - Доступ до NAS здійснюється як до іншої файлової системи, якщо використовуються протоколи CIFS або NFS, або як до raw block device, якщо використовується протокол iSCSI.
  - Хмарне сховище – це набір програм, які використовують API для доступу до сховища. Прикладами є Amazon S3, Dropbox, Microsoft OneDrive, Apple iCloud та ін.
- Одна з причин використання спеціальних API – затримки та збої у WAN-мережі.
  - Протоколи NAS спроектовані для LAN-мереж, у яких затримки нижчі, ніж у WAN-мережах, а ймовірність втрати зв’язку набагато нижча.
  - Якщо LAN-підключення відмовляє, система на базі NFS або CIFS може зависнути до моменту відновлення зв’язку.
  - Для хмарного сховища подібні збої більш характерні, тому додаток просто призупинить доступ до відновлення підключення.

Storage-Area Networks та масиви зберігання даних



- Одним з недоліків NAS-систем є те, що операції вводу-виводу забирають частину пропускної здатності data network, підвищуючи затримки мережевого підключення.
- **Storage-area network** (**SAN**) – це приватна мережа (використовує storage-протоколи замість мережевих), яка з'єднує сервери та storage units.
  - Багато хостів та багато сховищ можуть підключатись до однієї SAN, а сховище може динамічно виділятись для хостів.
  - Масиви зберігання даних можуть захищатись RAID-віртуалізацією або не захищатись взагалі (**Just a Bunch of Disks** (**JBOD**)).
  - Мережевий коммутатор SAN дозволяє або забороняє доступ між хостами та сховищем.
  - Наприклад, якщо на хості закінчується дисковий простір, SAN можна сконфігурувати на виділення простору для цього хосту.
  - SANs дозволяє кластерам серверів ділитись одним сховищем, а масивам зберігання даних включати багато прямих підключень від хостів.

Storage-Area Networks та масиви зберігання даних

- Масив зберігання даних включає SAN-порти та/або мережеві порти.
  - Також він має накопичувачі для зберігання даних та контролер (або надмірний набір контролерів) для керування сховищем та дозволу доступ до сховища через мережу.
  - Контролери компонуються з ЦП-рів, пам’яті та ПЗ, яке реалізує можливості масиву: мережеві протоколи, інтерфейс користувача, RAID-захист, snapshots, реплікацію, стиснення, deduplication та шифрування.
  - Деякі масиви зберігання даних включають SSD-носії.
- FC – поширений інтерконнект SAN, хоч росте популярність iSCSI завдяки його простоті.
  - Інший інтерконнект SAN - **InfiniBand** (**IB**)—архітектура шин спеціального призначення, яка забезпечує апаратну та програмну підтримку для високошвидкісних мереж для серверів та накопичувачів (storage units).