Advanced Operating Systems

Summer Semester 2024/2025

Martin Děcký

5

Storage

Address Space

- **Universal abstraction for accessing data (code is a form of data)**
  - **Physical memory**
    - Bytes, words, instructions (or similar)
  - **Virtual memory** (software / device)
    - Pages (or similar)
  - **I/O memory**
    - Bytes, words, ports (or similar)
    - Can be embedded in physical memory (memory-mapped I/O)
  - **Persistent memory**
    - Blocks, pages (or similar)
    - Can be combined with physical memory (non-volatile memory)
  - **Object space**
    - Keys, capabilities (or similar)

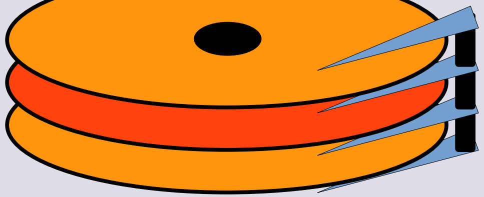
Non-Volatile Memory

- **Historically biased towards rotational media**
  - Cylinder / Head / Sector → Linear (Logical) Block Addressing
    - Originally interface abstraction not very high
      - Hard sectored → Soft sectored (with remapping)
    - 512 B blocks → 4096 B blocks (floppy/hard drives)
    - 2048 B blocks (optical drives), 2353 B blocks (raw optical drives)
  - Latency several orders of magnitude larger than volatile memory
    - Originally interface I/O efficiency not very important
      - Single tenant
      - Single request stream

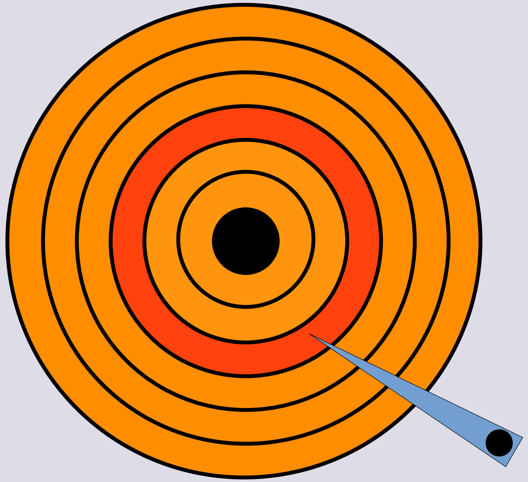
Non-Volatile Memory

- Historically biased towards rotational media

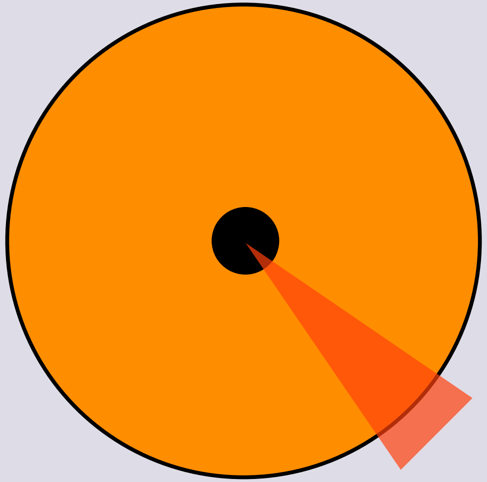
heads



cylinders
(tracks)



sectors & interleaving



Non-Volatile Memory

- **Historically biased towards rotational media**
  - Multi-tenant performance dominated by physical seek time
  - Still mostly via single request stream
    - Software I/O scheduling (shortest seek first, elevator/sweep, shortest deadline first, etc.)
      - Might not have the most accurate physical storage information (i.e. remapping)
    - I/O command batching (queuing)
      - Leaving the optimal I/O order (within the batch) to hardware
      - Incorporates interrupt coalescing

Non-Volatile Memory

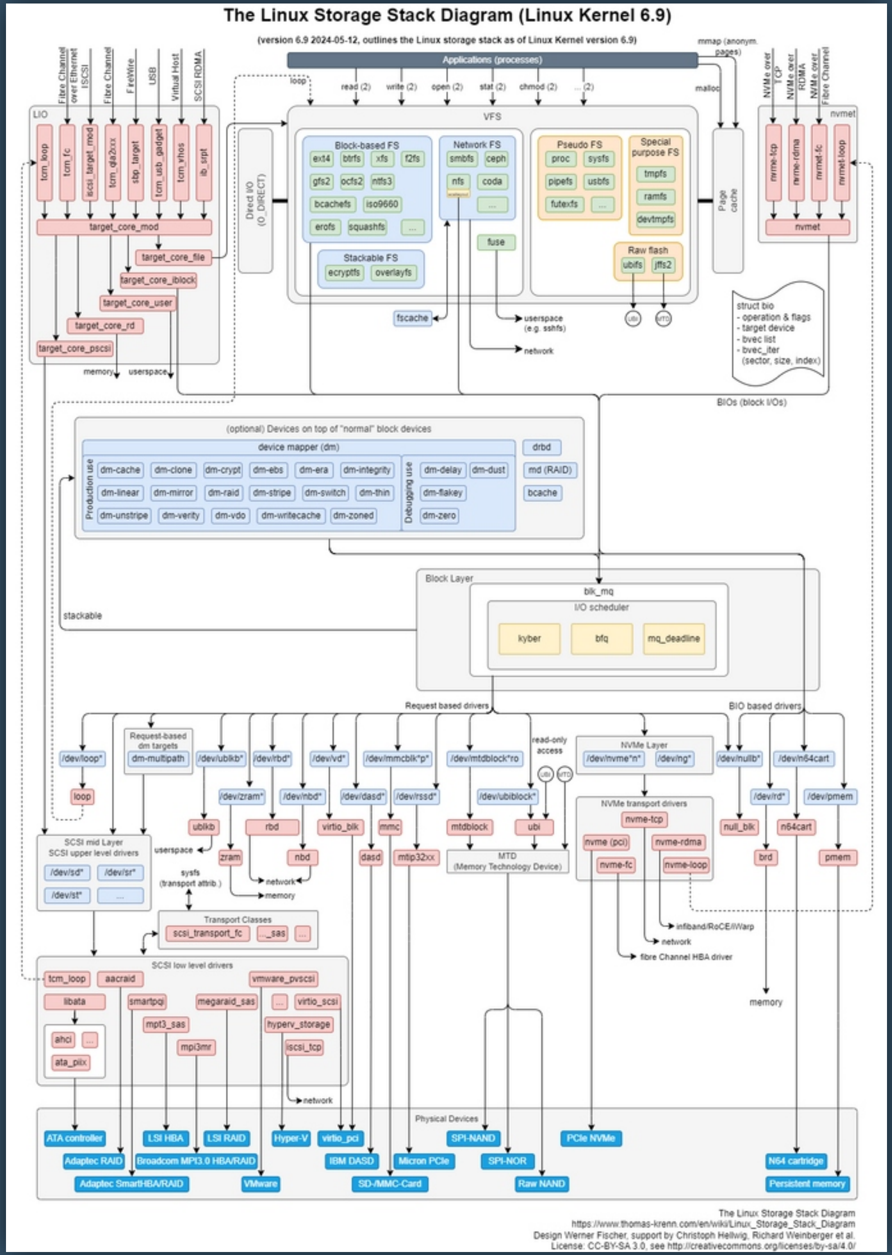
- **Solid-state drives**
  - Differing characteristics from rotational drives
    - Physical characteristics mostly unimportant
    - Addressing characteristics
      - Different native read/write and erase blocks
        - Write amplification
    - Physical addressing more like volatile memory
  - Latency much closer to volatile memory
    - Performance dominated by interface I/O efficiency
  - High degree of internal parallelism
  - Unique wear characteristics

Non-Volatile Memory

- **Solid-state drives**
  - Reflection in the I/O interface (e.g. NVMe)
    - Generally provides the common LBA abstraction
      - Wear leveling, block remapping and garbage collection in hardware Flash Transition Layer (FTL)
        - Frequently implemented as multi-level log-based storage
        - Software *trim* hint to indicate unused (erased) blocks
        - Trade-offs between write amplification, performance, idle characteristics
    - Low latency and parallel access
      - “Unlimited” request queues with lock-less access
      - “Unlimited” command queuing
      - Interrupt coalescing & multiple interrupt groups
      - Full-duplex scatter-gather DMA

Non-Volatile Memory

- **Solid-state drives**
  - Exposing more of the hardware architecture to software
    - Addressing
      - Open-channel SSD
      - NVMe Zoned Namespace
        - Note: Zones also useful for Shingled Magnetic Recording (SMR)
    - Compute off-loading
      - Basic NVMe I/O commands: *Compare*, *Write Zeroes*, *Copy*
      - NVMe Key Value command set
      - Near data computing (proposed)



Source: Werner Fischer

Storage Near Data Computing

- **Off-loading computation to storage controller**
  - Decrease latency, improve throughput, decrease energy consumption
  - Improve performance
    - Trade-off: Lower performance of embedded cores
      - Still a performance boost when compute cores are already loaded

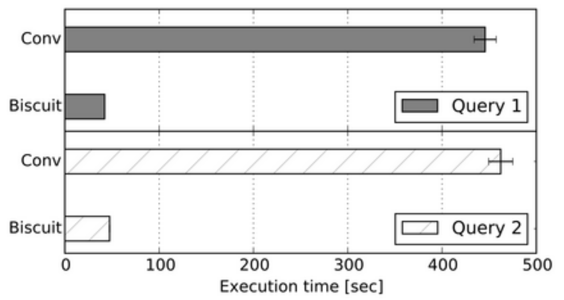
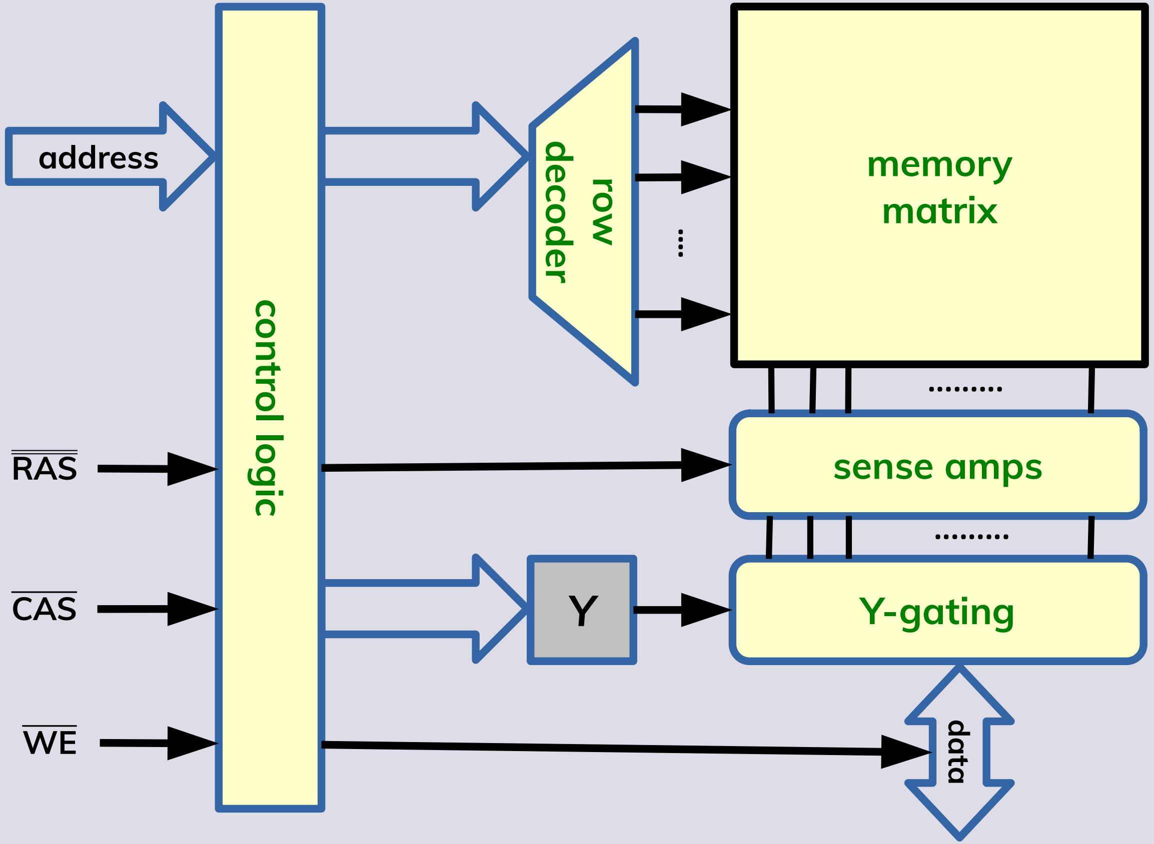


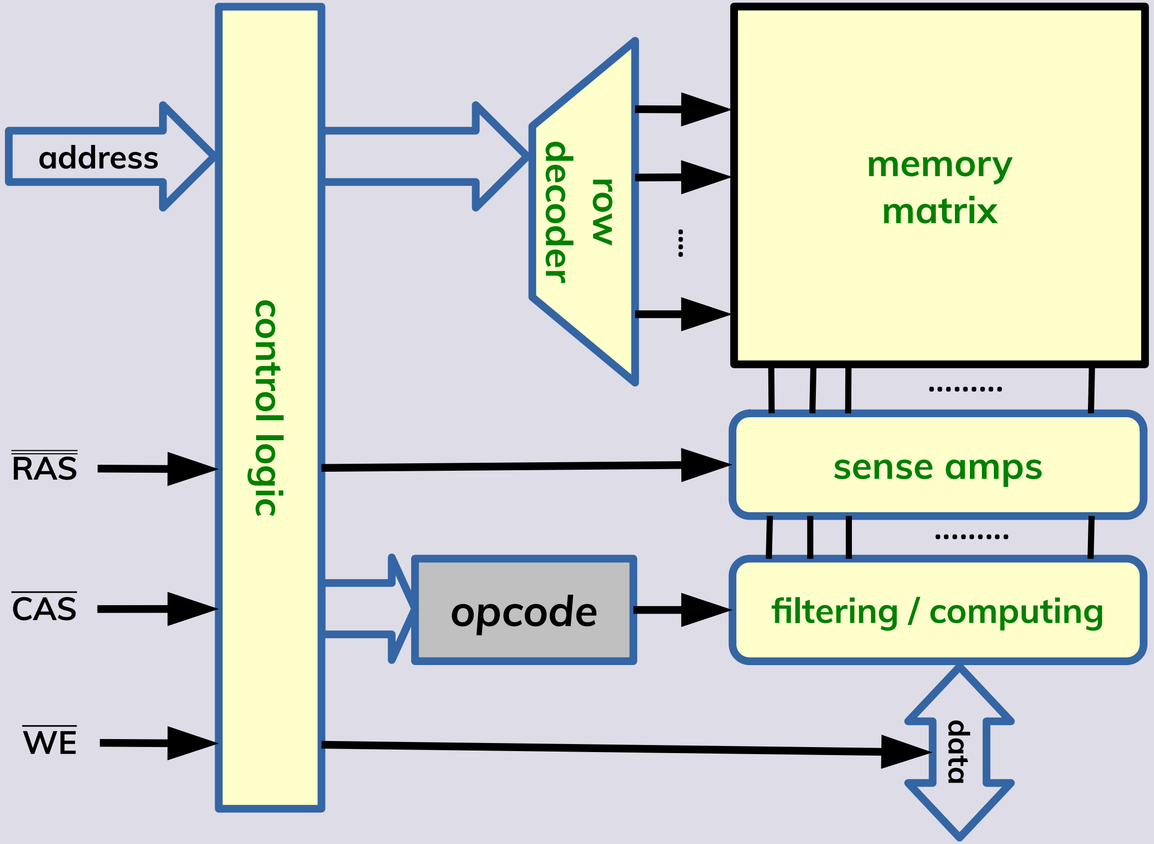
Figure 8. Performance of SQL queries on lineitem table.

Source: Gu B., Yoon A. S., Bae D.-H., Jo I., Lee J., Yoon J., Kang J.-U., Kwon M., Yoon C., Cho S., Jeong J., Chang D.: *Biscuit: A Framework for Near-Data Processing of Big Data Workloads*, in Proceedings of 43rd Annual International Symposium on Computer Architecture, ACM/IEEE, 2016

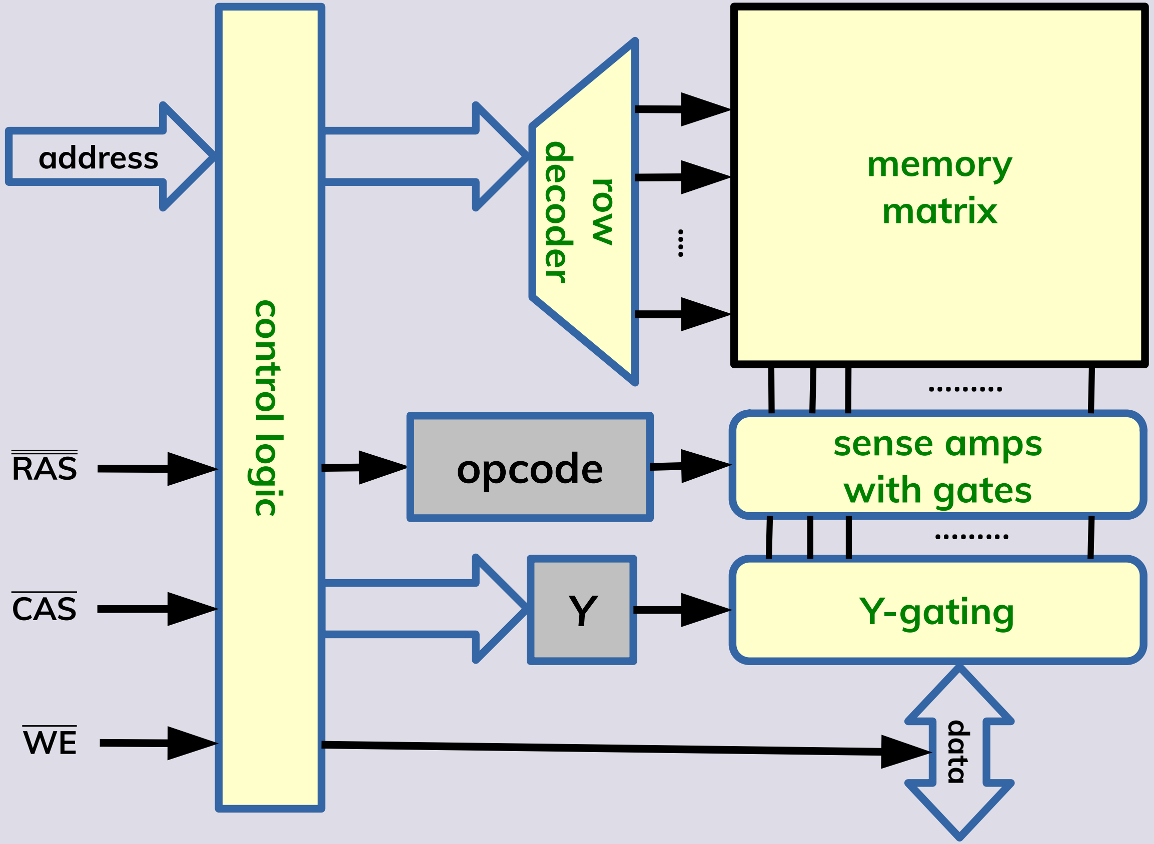
Memory Near Data Computing



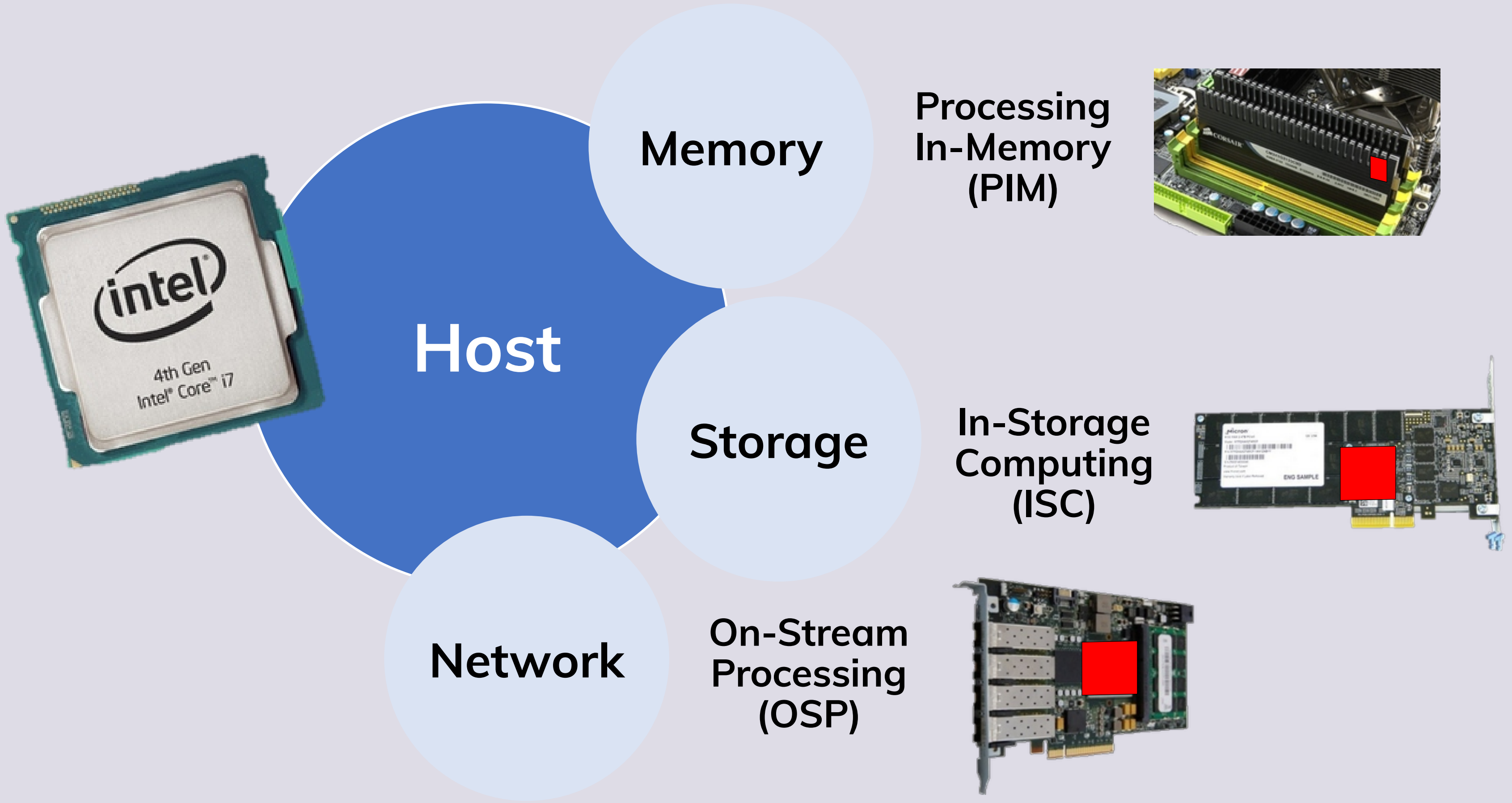
Memory Near Data Computing



Memory Near Data Computing



Generic Near Data Computing



Generic Near Data Computing

- **Current work-in-progress**
  - Universal open interface standards
    - Currently extensions of existing I/O interfaces (NVMe)
    - Compute Express Link (CXL)
- **Open questions**
  - Universal programming model
    - Stream / flow processing
    - Association of compute units with data
  - Universal compute model
    - ISA
    - Safety and security considerations
  - Off-loading vs. distributed computing

6

File Systems

Classification of File Systems

- **Traditional**
  - **Examples:** ext4, XFS, NTFS, UFS (latest variants), BFS, JFS2, etc.
  - Universal set of features
  - Distinction between directory entries and i-nodes
  - On-disk layout affected by rotational media and traditional partitioning
  - Typically use of somewhat sophisticated data structures
  - Typically larger constant overhead
    - Not usable for small media
  - Reliability via journaling of changes
    - Soft updates as an alternative

Classification of File Systems

- **Basic**
  - **Examples:** FAT, exFAT, etc.
    - Historical examples (with some advanced features): HPFS, HFS
  - Somewhat limited set of features
    - Typically missing permissions, ownership and other metadata, limited directory entry types, limited file names, limited file sizes, size of some data structures fixed, etc.
  - Frequently no distinction between directory entries and i-nodes
  - On-disk layout could be affected by slow / removable rotational media
  - Typically not so sophisticated data structures
  - Limited reliability

Classification of File Systems

- **Optical**
  - **Examples:** ISO 9660, UDF
  - Compact, continuous structures to minimize seeking
    - Path tables, directories, files
  - Additional sessions referencing previous sessions
    - Keeping / adding / removing files
    - Wear leveling and block remapping for rewritable media
      - As opposed to hardware abstractions (e.g. Mount Rainier)
  - Hybrid media

Classification of File Systems

- **Log-structured**
  - **Examples:** JFFS2, NILFS2, YAFFS, UBIFS, F2FS
  - **Idea:** Instead of keeping a journal for consistency, why not use the journal as the data storage?
  - Suits well zoned media (flash, SMR)
    - Block subdivision and GC more efficient than basic appending
  - Stale data can be accessed as snapshots (versions)
  - Inherently always consistent
  - Initial scan optimizations (persistent indexes)

Classification of File Systems

- **Copy-on-write**
  - **Examples:** ZFS, btrfs, bcachefs, HAMMER2, APFS, ReFS
  - **Idea:** Flexible on-disk layout, but no overwrites
  - Stale data can be accessed as snapshots (versions)
  - Multiple mountable roots
  - Other advanced features (not strictly specific to COW)
    - Data checksums (separately stored, Merkle tree), data redundancy, deduplication, integration with logical volume management, hierarchical caching, wandering intent logs, replication
  - Inherently always consistent
  - Initial scan issues avoided, but GC still needed (also serves as defragmentation)

Classification of File Systems

- **Read-only**
  - **Examples:** SquashFS, cramfs, EROFS, AXFS
  - Efficient storage of seed images (boot images, container images, thin provisioning, etc.)
    - Often coupled with union mounts for read/write support
  - Low overhead, no fragmentation, compression
  - Easy caching, execute-in-place (adaptive compression)

Classification of File Systems

- **Shared-disk**
  - **Examples:** CXFS, GPFS, GFS2, OCFS, HAMMER2
  - Support for underlying block modifications from independent sources
    - Via iSCSI, ATA over Ethernet, Fibre Channel, InfiniBand, NVMe over fabric
  - In between regular file systems and network file systems
  - Distributed lock manager vs. metadata broker

6.1

File System Curiosities

Traditional File Systems with Bonuses

- **AdvFS, NSS**
  - Fairly traditional file systems, but supporting multiple block devices
- **HFS+**
  - Hard links to directories
- **RaiserFS**
  - Tail packing (sub-allocation of blocks)

Traditional File Systems with Bonuses

- **NTFS**
  - Reparse points, file system filters
  - Caching i-node size in directory entry (non-consistent among hard links)
  - Hard links for 8.3 file names
  - Per-directory case sensitivity
    - Case insensitivity is not trivial [1][2]
  - Transactional NTFS
    - Integrated with Kernel Transaction Manager
    - Transaction-Safe FAT

Traditional File Systems with Bonuses

- **XFS**
  - Allocation groups (concurrency)
  - Multiple devices, COW, snapshots, deduplication, striping
    - Controlled by Stratis
- **ext4**
  - Journal checksums
- **StegFS**
  - Steganographic extension to ext2
    - Undetectable, hidden layer of files on a regular file system

Less Traditional File Systems

- **btrfs**
  - Integrated support for union mounting (read-only seeding)
- **Linear Tape File System (LTFS)**
- **NOVA**
  - Targeting byte-addressable persistent memory (NVRAM)
  - Log structured for metadata per i-node (concurrency)
    - Log is append-only, but non-continuous (linked list)
    - Replication and checksums
  - Data blocks managed as copy-on-write
  - Global journaling for reliability of non-atomic operations

Source: DALL·E 3 via ChatGPT 4o

Other File Systems Remarks

- **Path separator**
  - The history of slash / backslash in complicated [4][5]
- **Resource forks, extended attributes**
  - Multiple streams associated with a single file
- **Forward and backward compatibility**
  - Feature sets, feature bitmaps
  - Allowed and required features
- **File system semantics are not trivial [3]**
- **Path lengths, valid path characters**

References

[1] https://lwn.net/Articles/784041/

[2] https://www.youtube.com/watch?v=yVlEZKiMGJU

[3] https://danluu.com/deconstruct-files/

[4] https://www.os2museum.com/wp/why-does-windows-really-use-backslash-as-path-separator/

[5] https://learn.microsoft.com/en-us/archive/blogs/larryosterman/why-is-the-dos-path-character

Thank you!

Questions?