Chapter 12: Mass-Storage Systems

Chapter 12: Mass-Storage Systems

- Overview of Mass Storage Structure
- Disk Structure
- Disk Attachment
- Disk Scheduling
- Disk Management
- Swap-Space Management
- RAID Structure
- Disk Attachment
- Stable-Storage Implementation
- Tertiary Storage Devices
- Operating System Issues
- Performance Issues

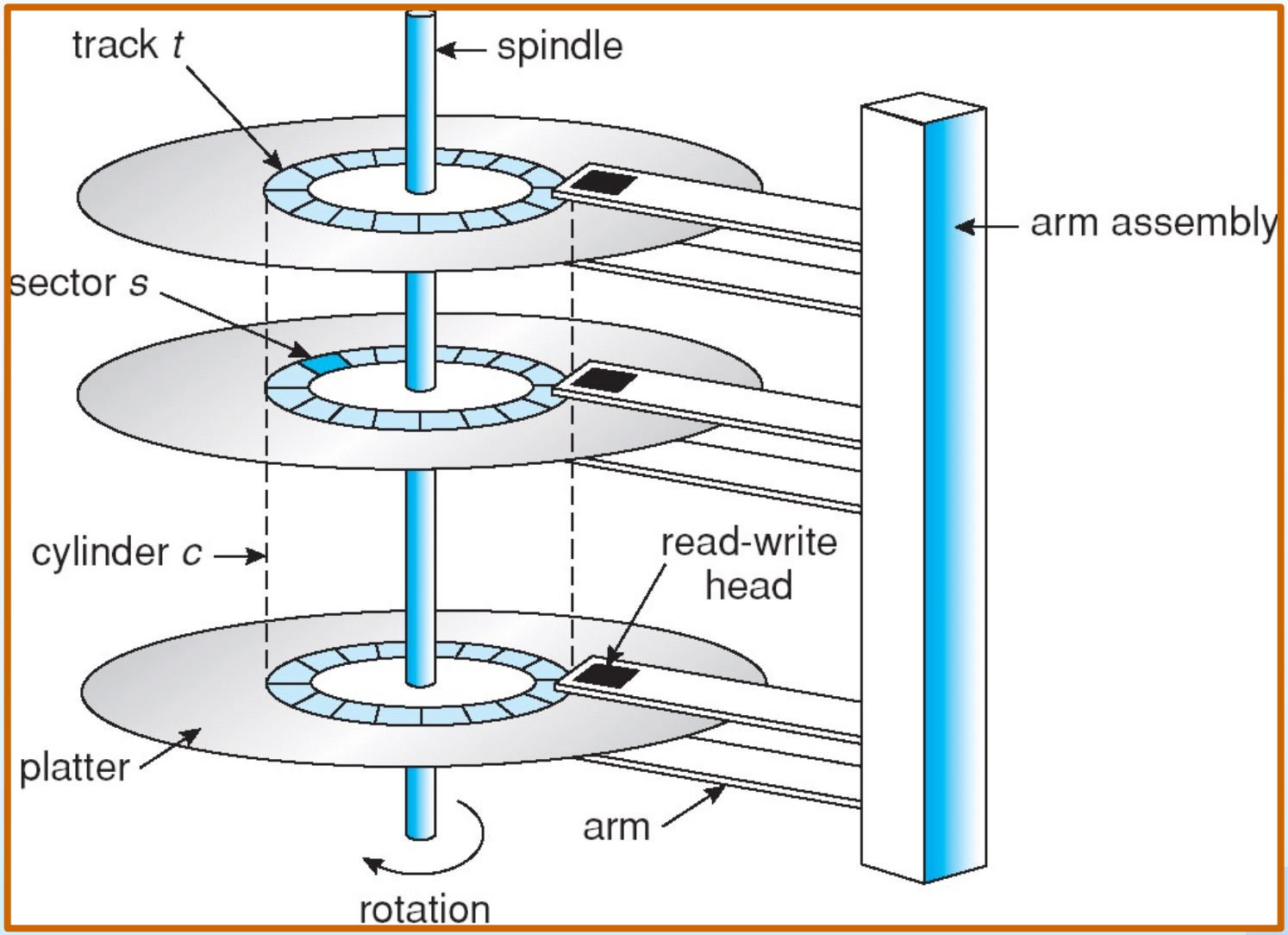
Objectives

- Describe the physical structure of secondary and tertiary storage devices and the resulting effects on the uses of the devices
- Explain the performance characteristics of mass-storage devices
- Discuss operating-system services provided for mass storage, including RAID and HSM

Overview of Mass Storage Structure

- Magnetic disks provide bulk of secondary storage of modern computers
  - Drives rotate at 60 to 200 times per second
  - **Transfer rate** is rate at which data flow between drive and computer
  - **Positioning time** (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)
  - **Head crash** results from disk head making contact with the disk surface
    - That's bad
- Disks can be removable
- Drive attached to computer via **I/O bus**
  - Busses vary, including **EIDE, ATA, SATA, USB, Fibre Channel, SCSI**
  - **Host controller** in computer uses bus to talk to **disk controller** built into drive or storage array

Moving-head Disk Machanism




This comparison of a 1974 10MB ILLIAC IV hard disk platter with a 2001 1 GB IBM microdrive illustrates how recording density has increased while overall size has decreased—in this case, by a factor of approximately 40,000. Courtesy of IBM and The Computer Museum History Center
29/04/2006

Overview of Mass Storage Structure (Cont.)

- Magnetic tape
  - Was early secondary-storage medium
  - Relatively permanent and holds large quantities of data
  - Access time slow
  - Random access ~1000 times slower than disk
  - Mainly used for backup, storage of infrequently-used data, transfer medium between systems
  - Kept in spool and wound or rewound past read-write head
  - Once data under head, transfer rates comparable to disk
  - 20-200GB typical storage
  - Common technologies are 4mm, 8mm, 19mm, LTO-2 and SDLT

Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of *logical blocks*, where the logical block is the smallest unit of transfer.
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.
  - Sector 0 is the first sector of the first track on the outermost cylinder.
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

Disk Attachment

Fibre Channel

SCSI

ATA(IDE)

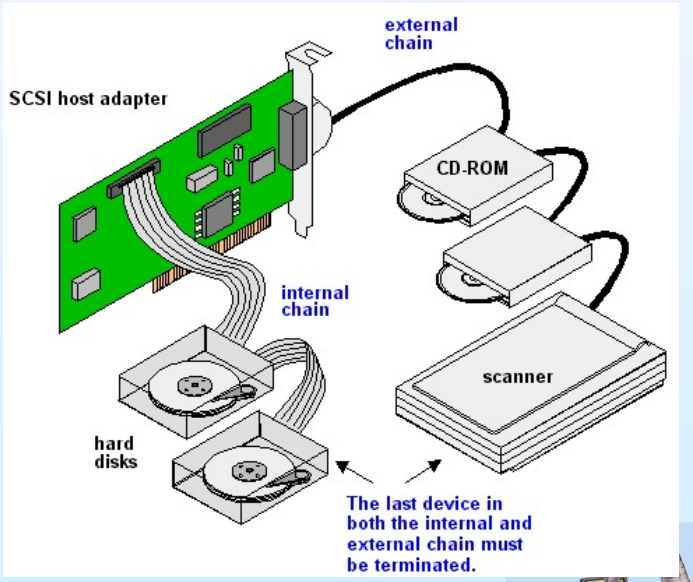
Serial ATA

Disk Attachment

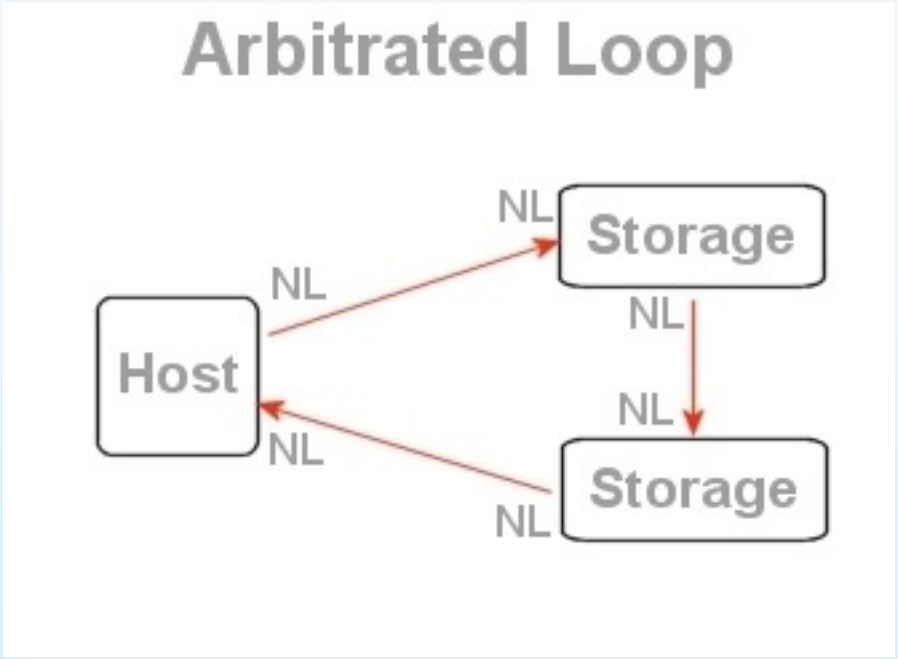
- Host-attached storage accessed through I/O ports talking to I/O buses
- SCSI(pronounced "scuzzy") itself is a bus, up to 16 devices on one cable, **SCSI initiator** requests operation and **SCSI targets** perform tasks
  - Each target can have up to 8 **logical units** (disks attached to device controller
- Fibre Channel (FC) is high-speed serial architecture
  - Commonly running at 2-, 4-, 8- and 16-gigabit per second rates
  - Can be switched fabric with 24-bit address space – the basis of **storage area networks** (**SAN**s) in which many hosts attach to many storage units
  - Can be **arbitrated loop** (**FC-AL**) of 126 devices

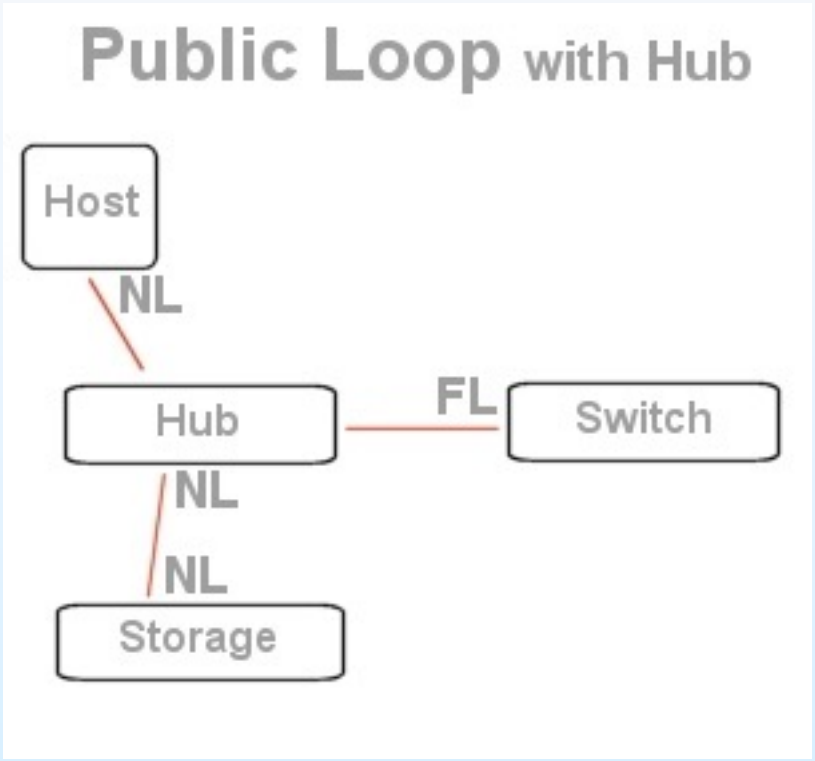
SCSI Interface

Name	Specification	# of Devices	Bus Width	Bus Speed	MBps
Asynchronous SCSI	SCSI-1	8	8 bits	5 MHz	4 MBps
Synchronous SCSI	SCSI-1	8	8 bits	5 MHz	5 MBps
Wide	SCSI-2	16	16 bits	5 MHz	10 MBps
Fast	SCSI-2	8	8 bits	10 MHz	10 MBps
Fast/Wide	SCSI-2	16	16 bits	10 MHz	20 MBps
Ultra	SCSI-3 SPI	8	8 bits	20 MHz	20 MBps
Ultra/Wide	SCSI-3 SPI	8	16 bits	20 MHz	40 MBps
Ultra2	SCSI-3 SPI-2	8	8 bits	40 MHz	40 MBps
Ultra2/Wide	SCSI-3 SPI-2	16	16 bits	40 MHz	80 MBps
Ultra3	SCSI-3 SPI-3	16	16 bits	40 MHz	160 MBps
Ultra320	SCSI-3 SPI-4	16	16 bits	80 MHz	320 MBps



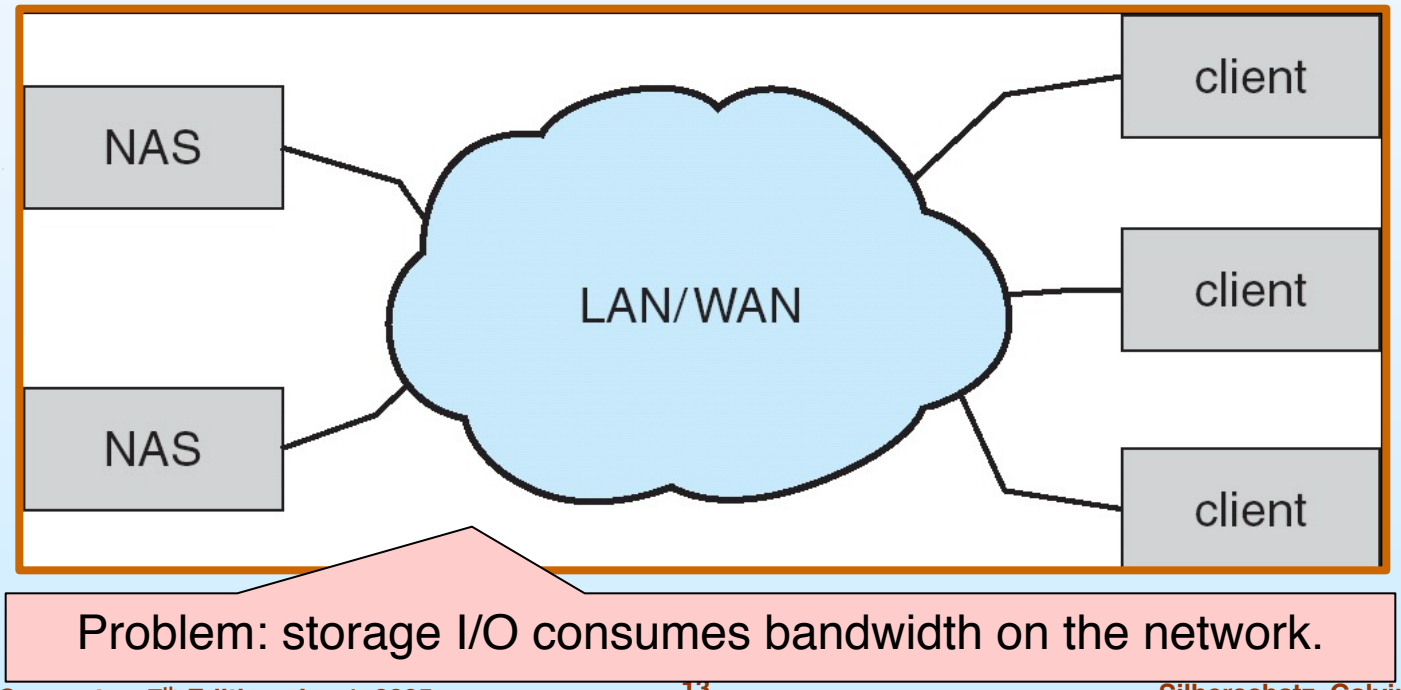
Fibre Channel





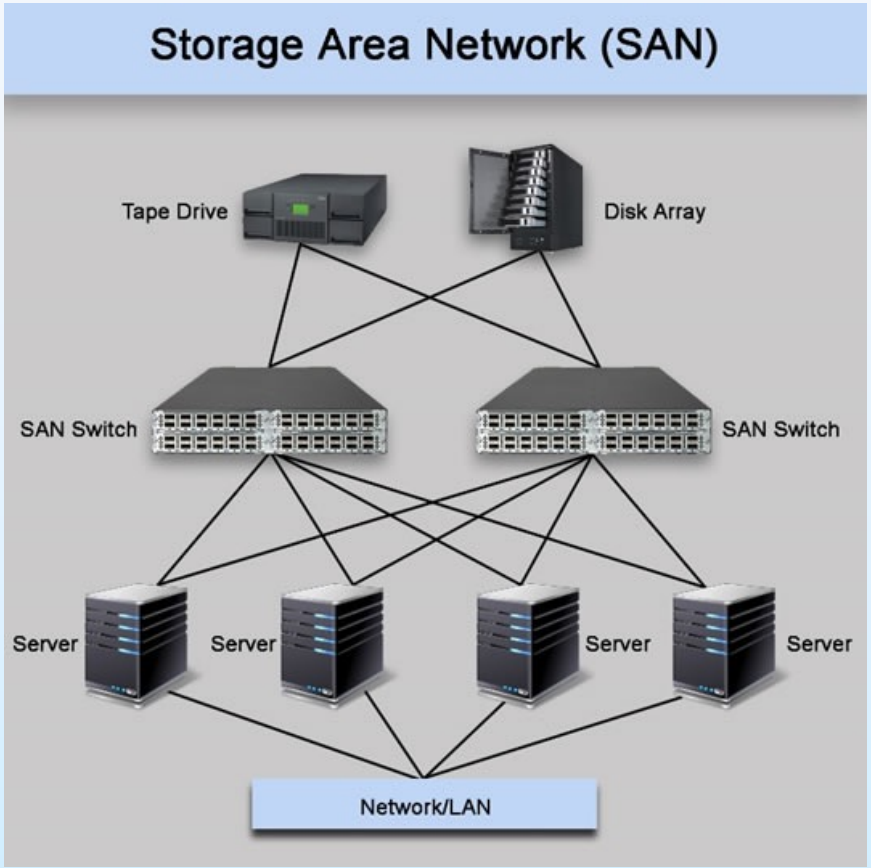
Network-Attached Storage

- Network-attached storage (**NAS**) is storage made available over a network rather than over a local connection (such as a bus)
- NFS and CIFS are common protocols
- Implemented via remote procedure calls (RPCs) between host and storage
- New iSCSI protocol uses IP network to carry the SCSI protocol

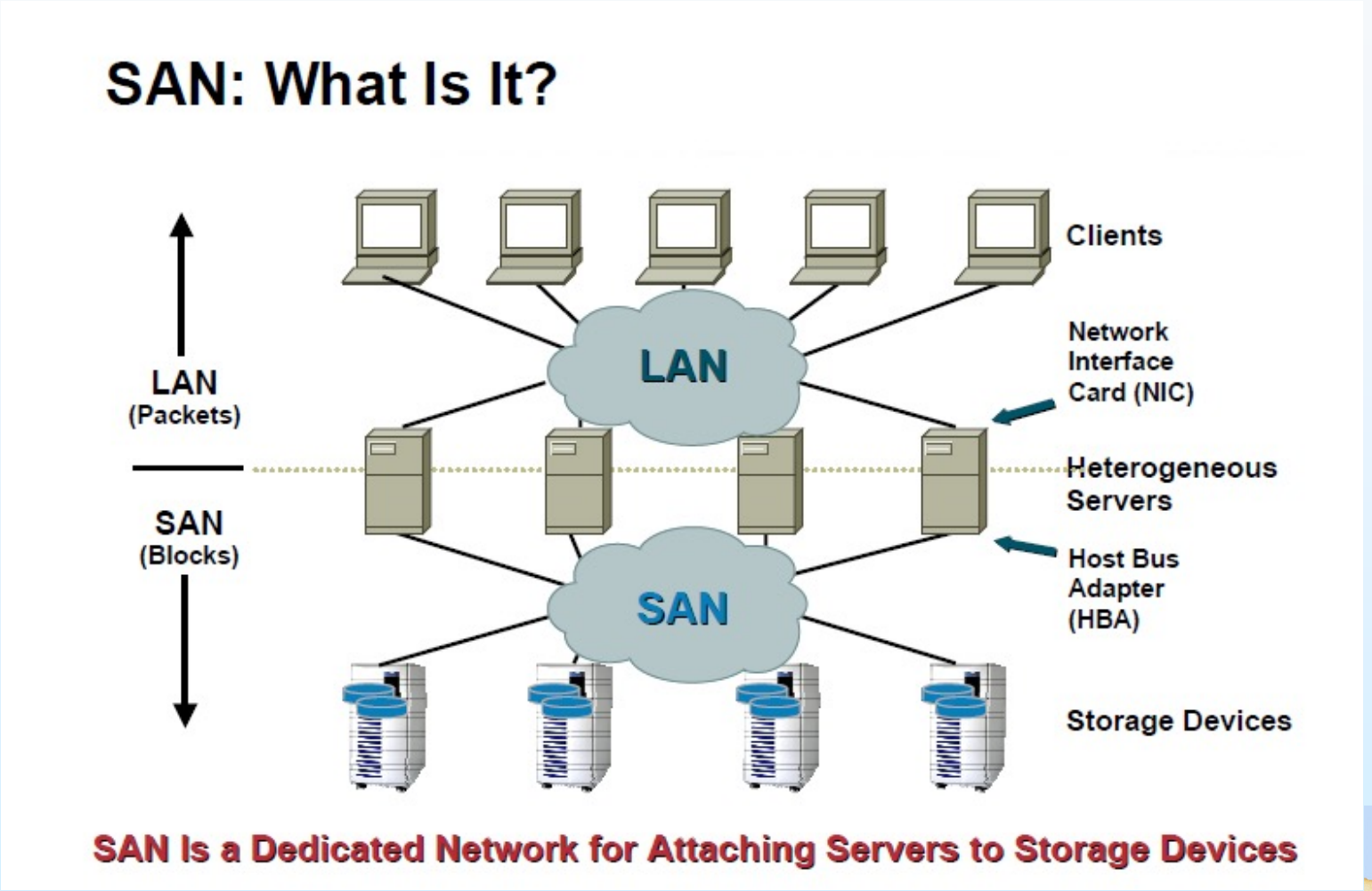


Storage Area Network

- Common in large storage environments (and becoming more common)
- Multiple hosts attached to multiple storage arrays - flexible



Storage Area Network



Accessing a Disk Page

- Time to access (read/write) a disk block:
  - *seek time* (moving arms to position disk head on track)
  - *rotational delay* (waiting for block to rotate under head)
  - *transfer time* (actually moving data to/from disk surface)
- Seek time and rotational delay dominate.
  - Seek time varies from about 1 to 20msec
  - Rotational delay varies from 0 to 10msec
  - As of 2010, a typical 7200 RPM desktop HDD has a "disk-to-buffer" data transfer rate up to 1030 Mbit/s
  - Key to lower I/O cost: reduce seek/rotation delays! Hardware vs. software solutions?

Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth.
- Access time has two major components
  - *Seek time* is the time for the disk are to move the heads to the cylinder containing the desired sector.
  - *Rotational latency* is the additional time waiting for the disk to rotate the desired sector to the disk head.
- Minimize seek time
- Metric: Seek time ≈ seek distance
- Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

Disk Scheduling (Cont.)

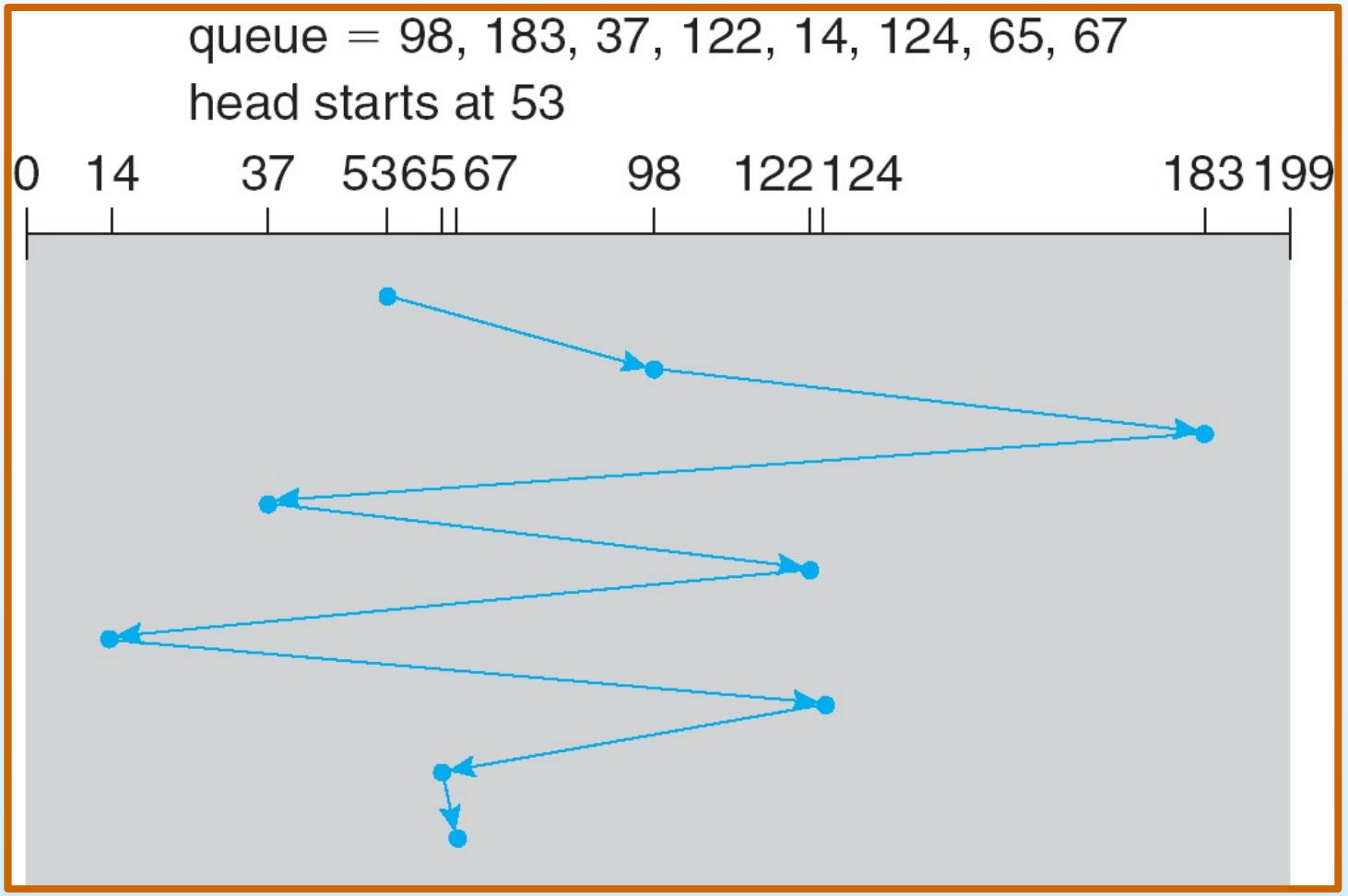
- Several algorithms exist to schedule the servicing of disk I/O requests.
- We illustrate them with a request queue (0-199).

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

Current head pointer 53

FCFS

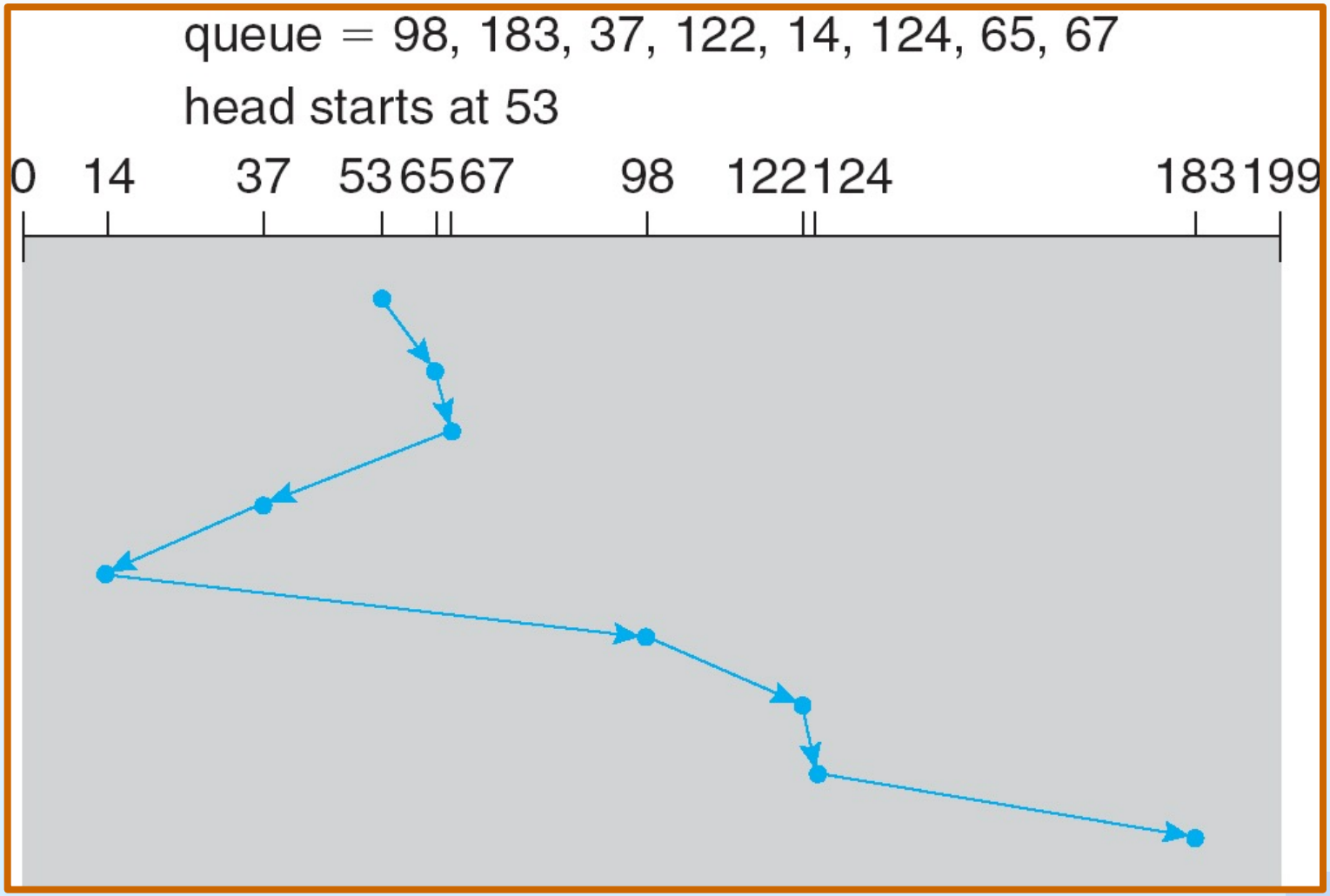
Illustration shows total head movement of 640 cylinders.



Shortest-seek-time-first (SSTF)

- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.
- Illustration shows total head movement of 236 cylinders.

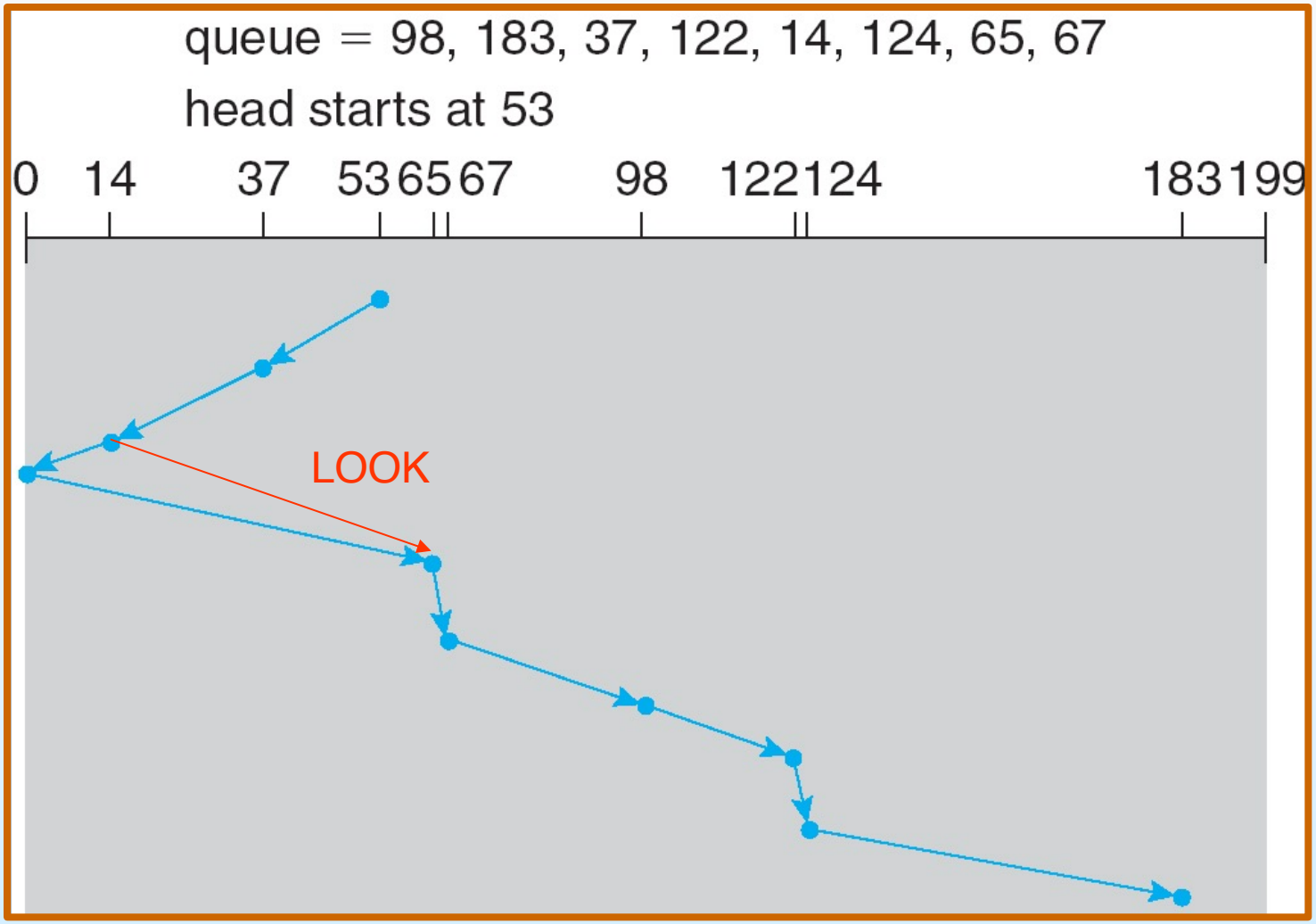
SSTF (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- Sometimes called the *elevator algorithm*.

SCAN (Cont.)

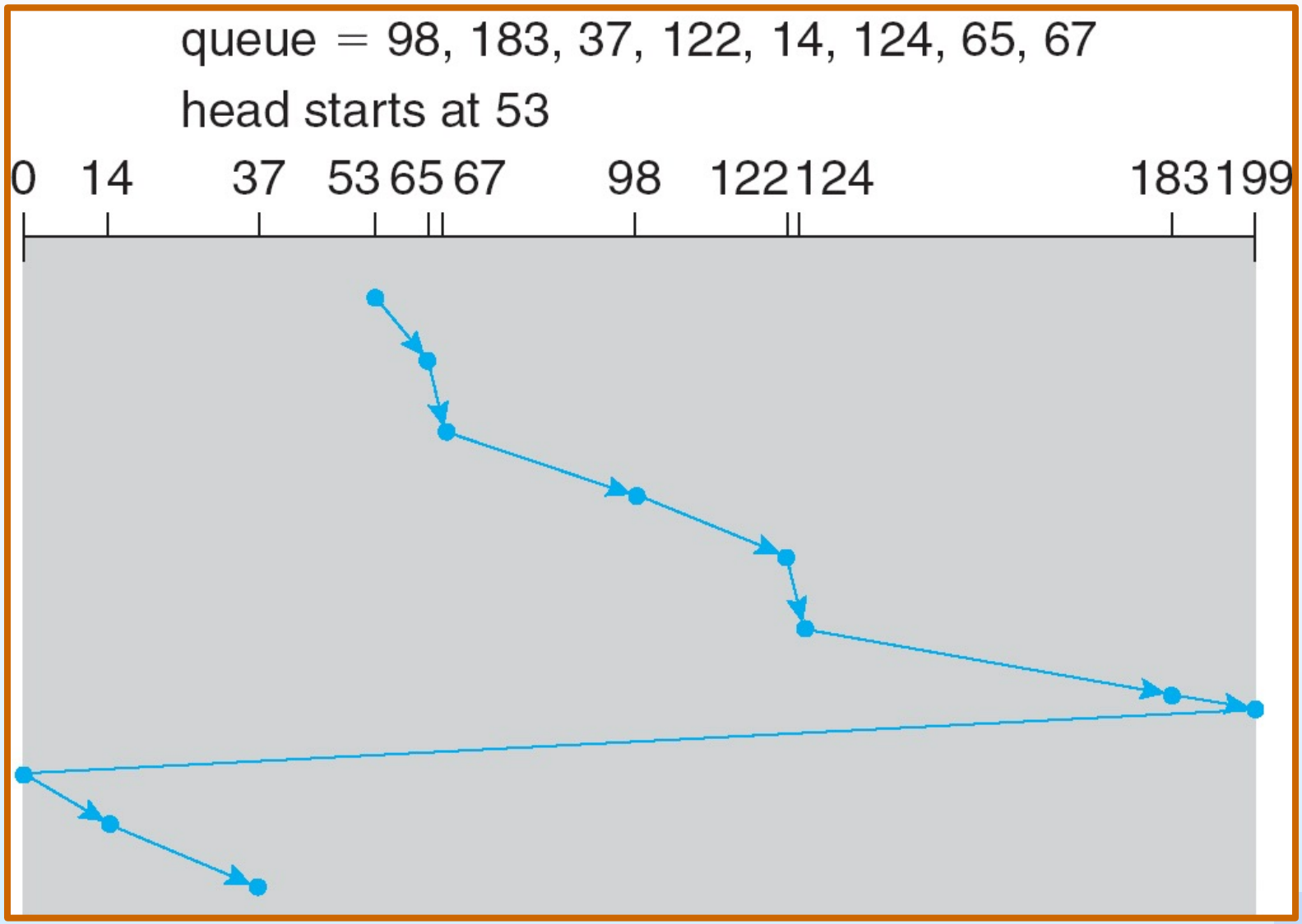


- LOOK needs total head movement of 208 cylinders.

C-SCAN

- Provides a more uniform wait time than SCAN.
- The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

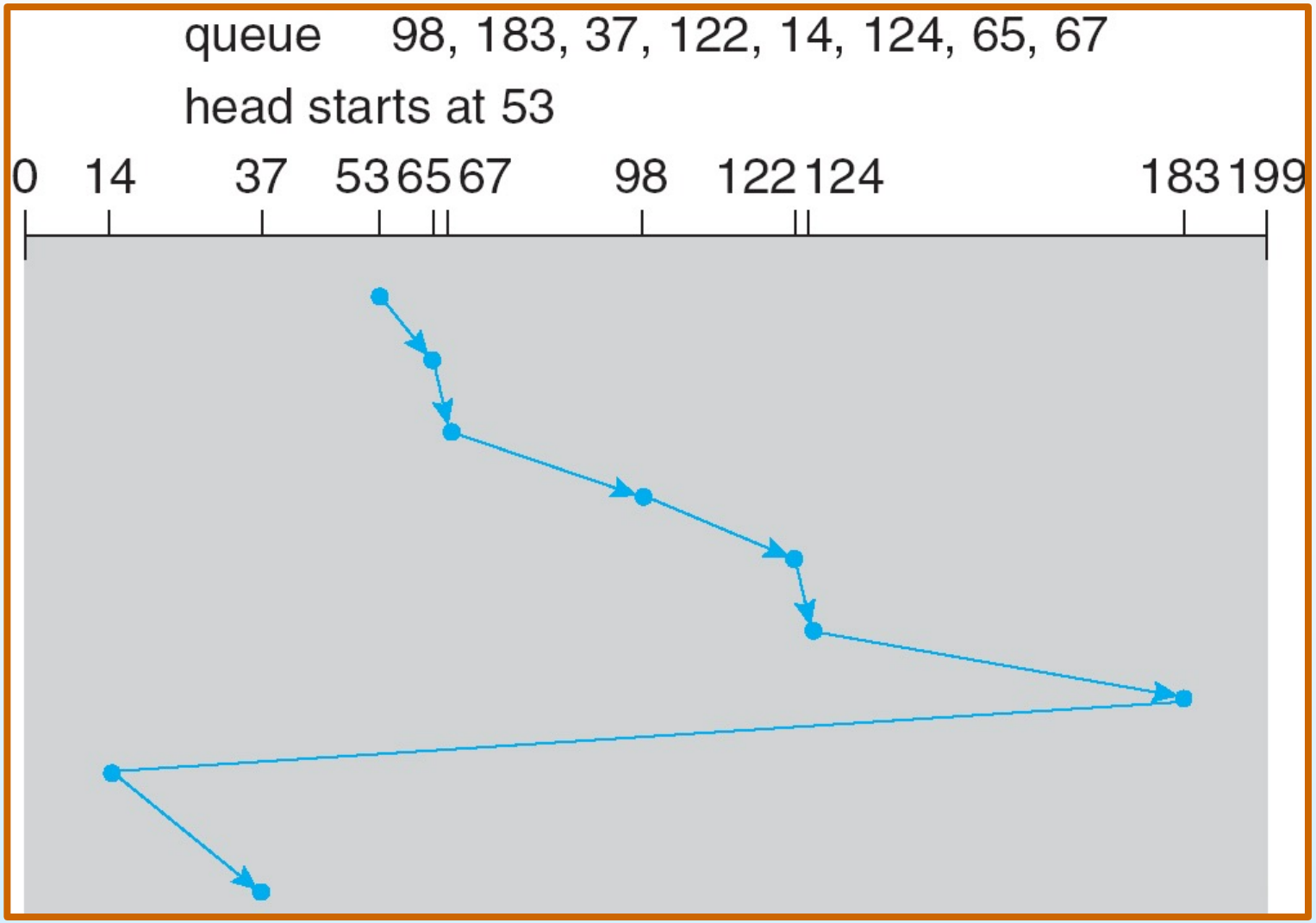
C-SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

C-LOOK

- Version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.

C-LOOK (Cont.)


queue 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

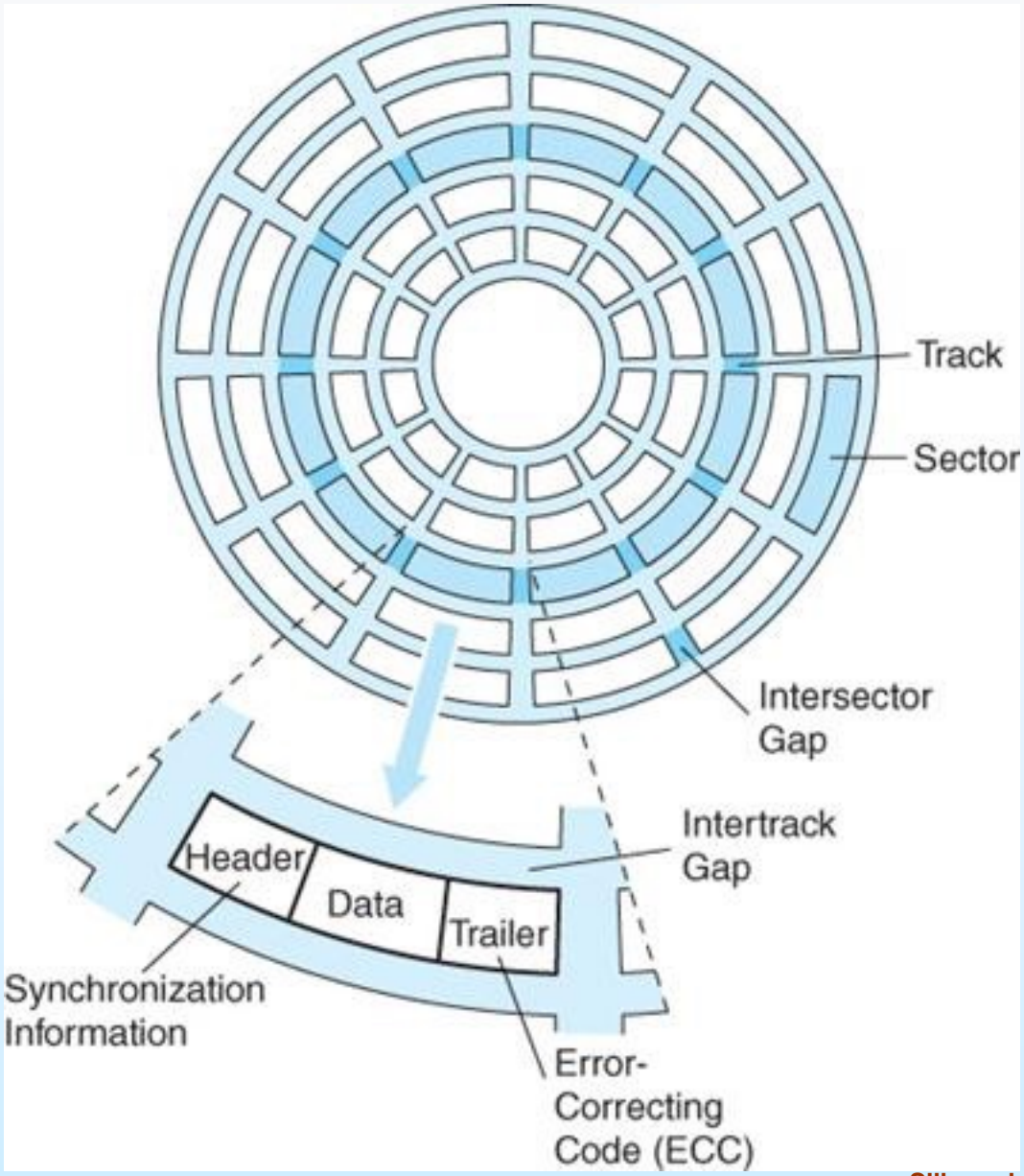
Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk.
- Performance depends on the number and types of requests.
- Requests for disk service can be influenced by the file-allocation method.
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary.
- Either SSTF or LOOK is a reasonable choice for the default algorithm.

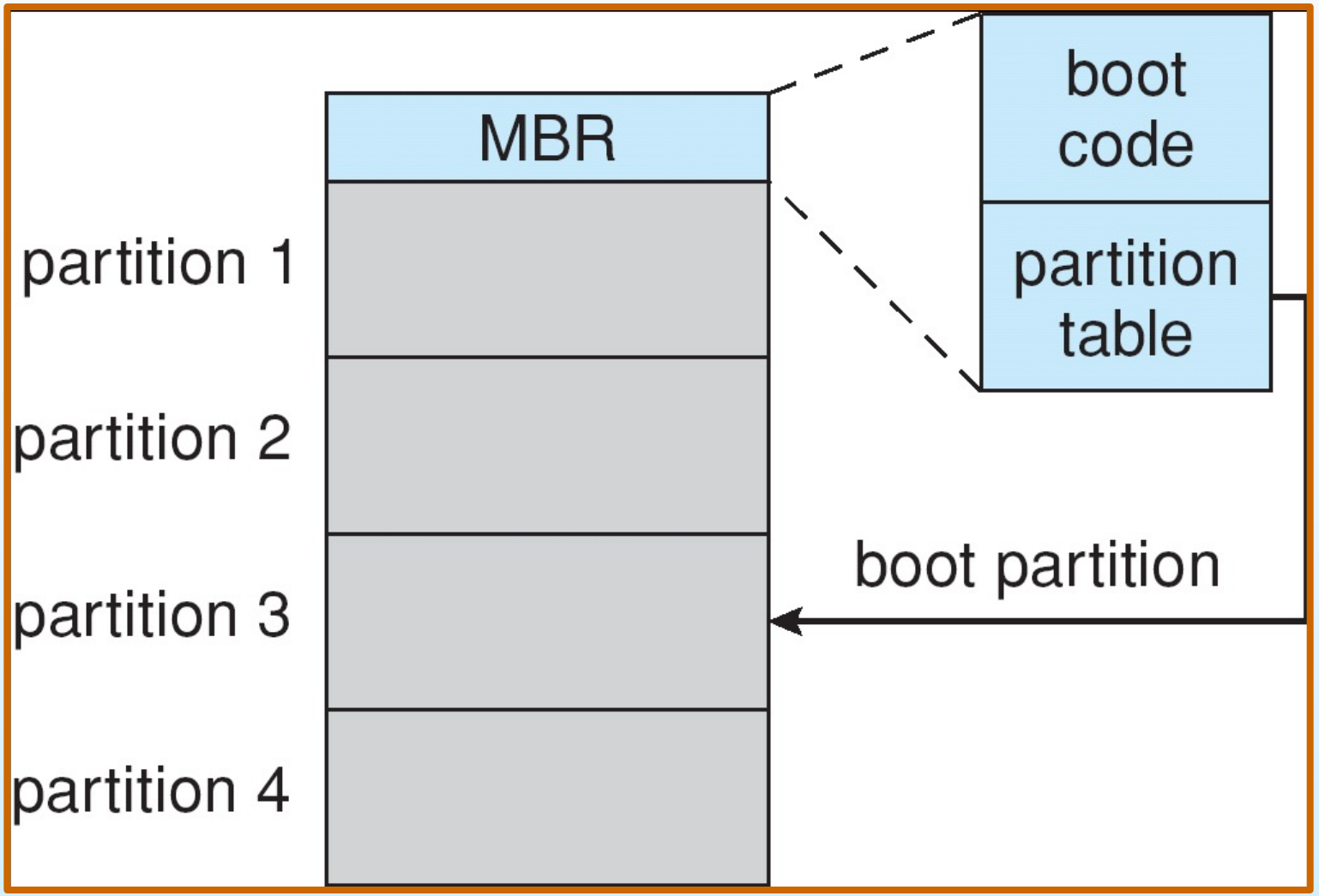
Disk Management

- *Low-level formatting*, or *physical formatting* — Dividing a disk into sectors that the disk controller can read and write.
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk.
  - *Partition* the disk into one or more groups of cylinders.
  - *Logical formatting* or “making a file system”.
- Boot block initializes system.
  - The bootstrap is stored in ROM.
  - *Bootstrap loader* program.
- Methods such as *sector sparing* used to handle bad blocks.

Disk Sector


Track
Sector
Intersector Gap
Intertrack Gap
Header
Data
Trailer
Synchronization Information
Error-Correcting Code (ECC)

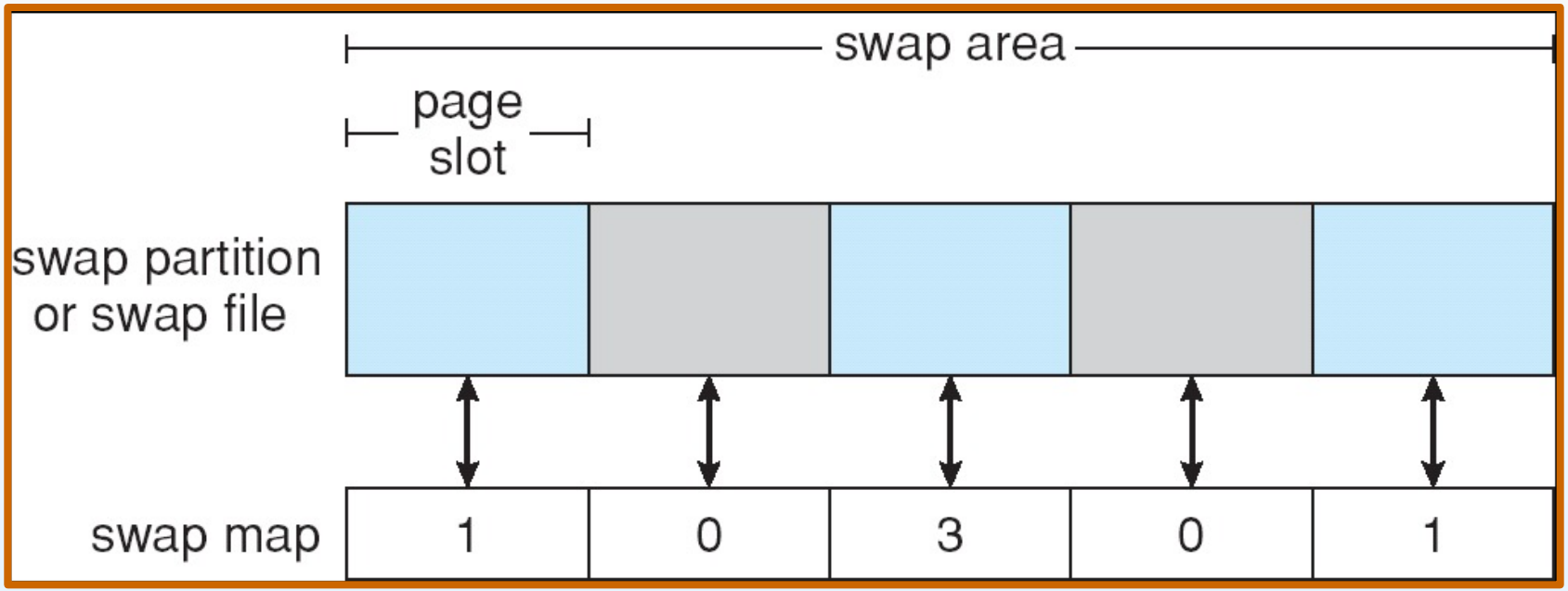
Booting from a Disk in Windows 2000



Swap-Space Management

- Swap-space — Virtual memory uses disk space as an extension of main memory.
- Swap-space can be carved out of the normal file system,or, more commonly, it can be in a separate disk partition.
- Swap-space management
  - 4.3BSD allocates swap space when process starts; holds *text segment* (the program) and *data segment.*
  - Kernel uses *swap maps* to track swap-space use.
  - Solaris 2 allocates swap space only when a page is forced out of physical memory, not when the virtual memory page is first created.

Data Structures for Swapping on Linux Systems



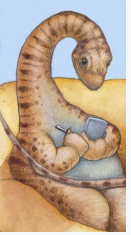
RAID Structure

- **RAID** – multiple disk drives provides **reliability** via **redundancy**.
- RAID is arranged into six different levels.
- RAID = redundant arrays of inexpensive disks

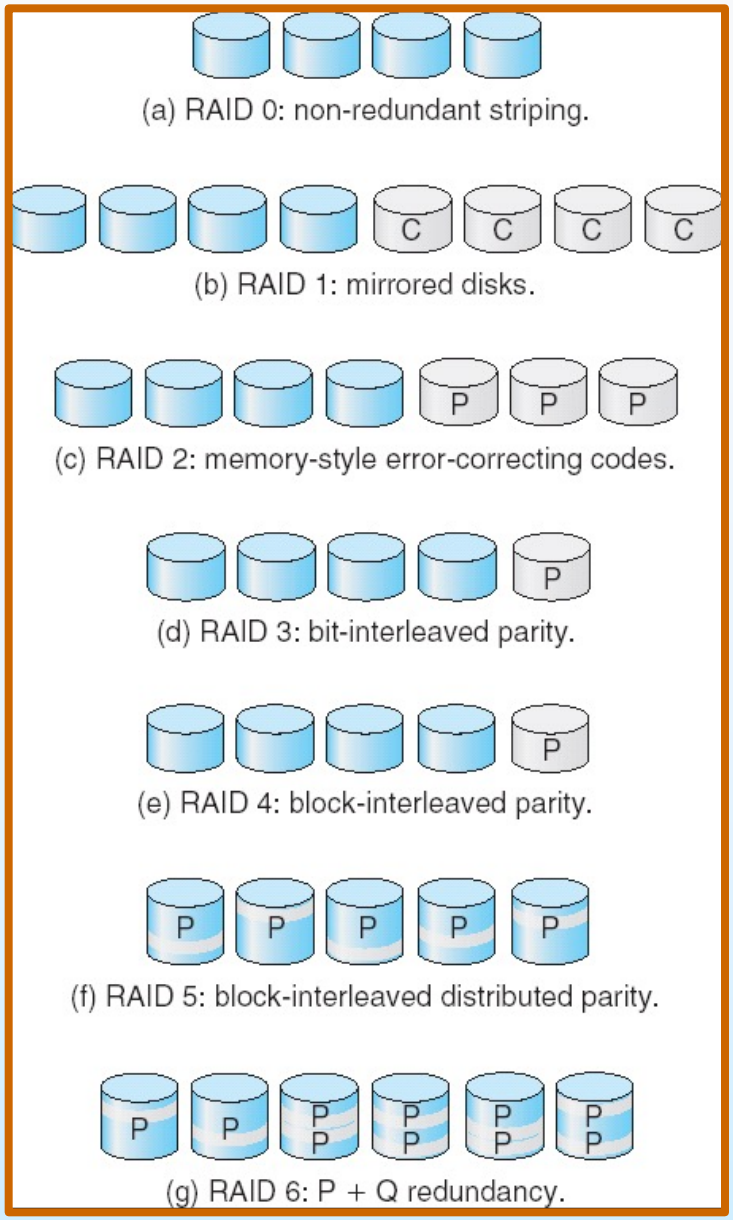
RAID (cont)

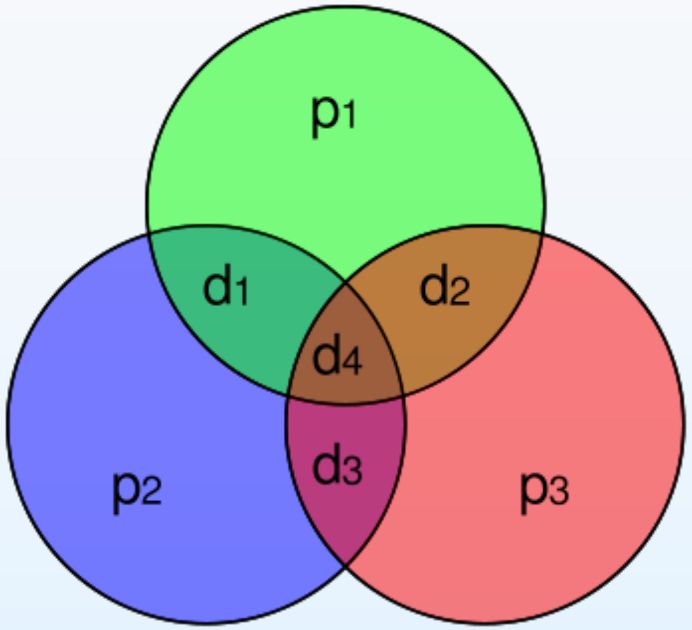
- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively.
- Disk striping uses a group of disks as one storage unit.
  - Bit-level Striping
  - Block-level Striping – different blocks of a file are striped
- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data.
  - *Mirroring* or *shadowing* keeps duplicate of each disk.
  - *Block interleaved parity* uses much less redundancy.

Disk 1	Disk 2	Disk 3	Disk 4
File 1, byte 1	File 1, byte 2	File 1, byte 3	File 1, byte 4
File 1, byte 5	File 1, byte 6	File 1, byte 7	File 2, byte 1
File 2, byte 2	File 3, byte 1	File 3, byte 2	File 4 byte 1
File 4, byte 2	File 4, byte 3	*And so on...*	



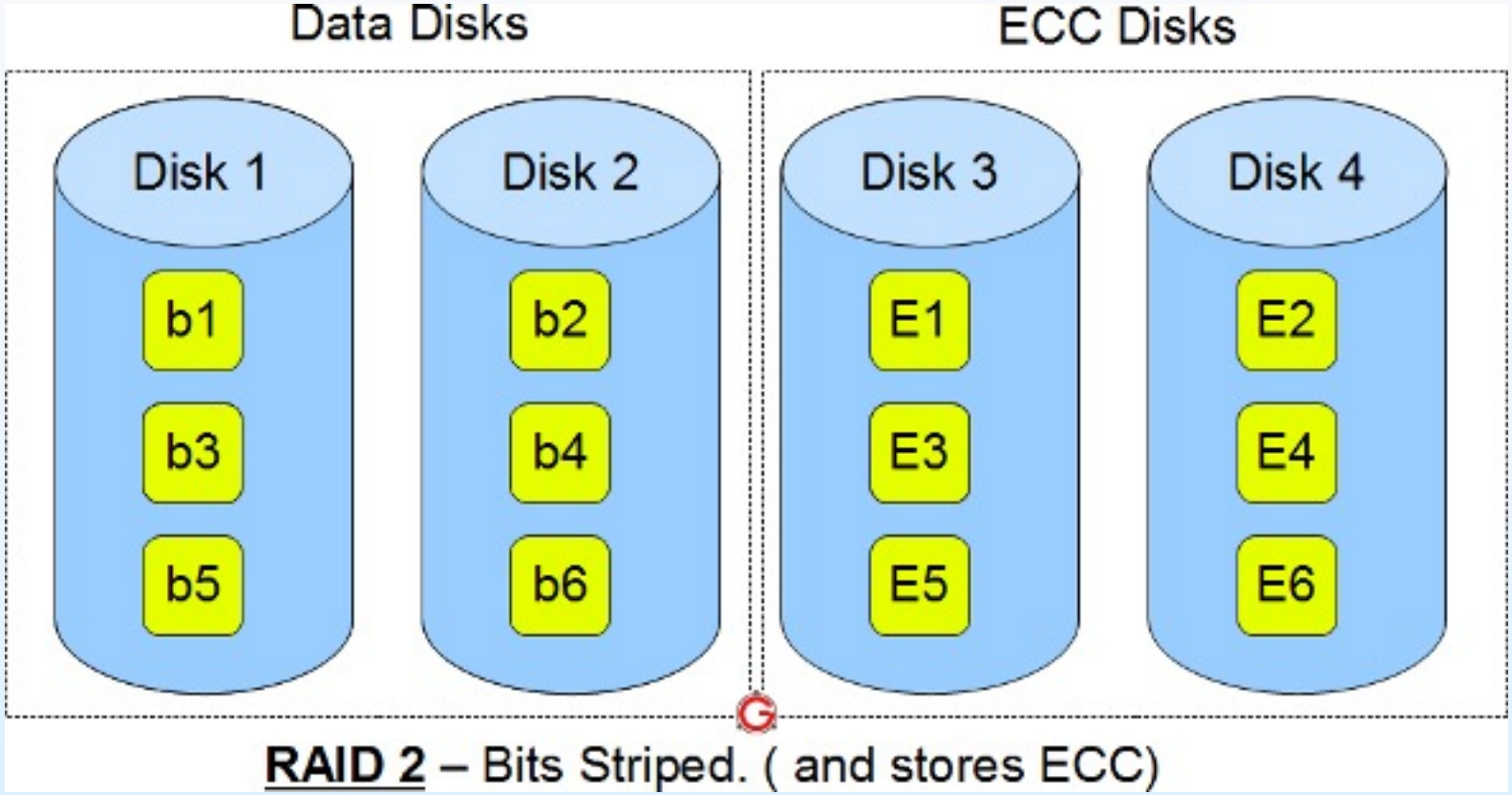
RAID Levels





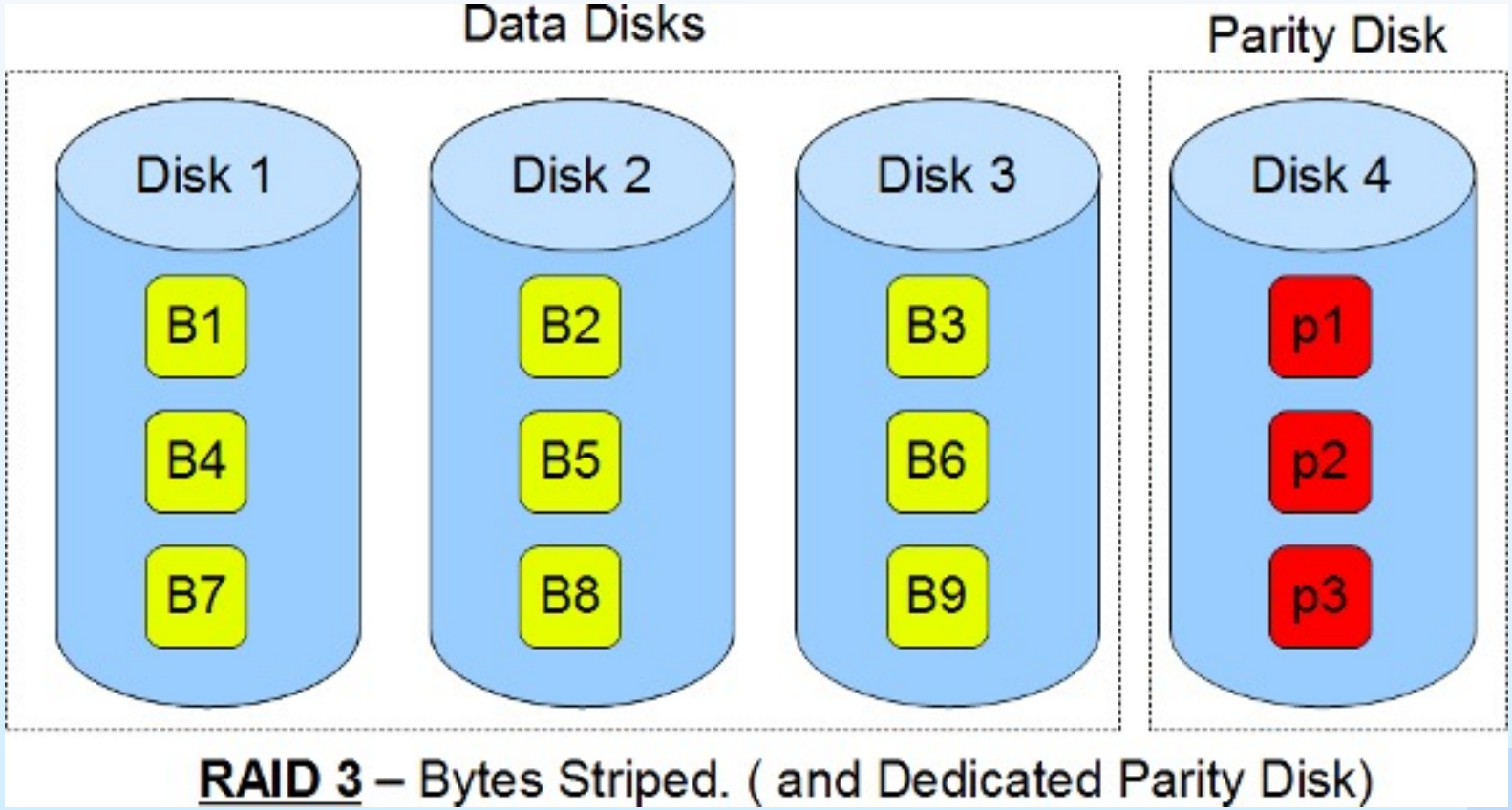
ECC, Hamming(7, 4)

Raid 2



RAID 3

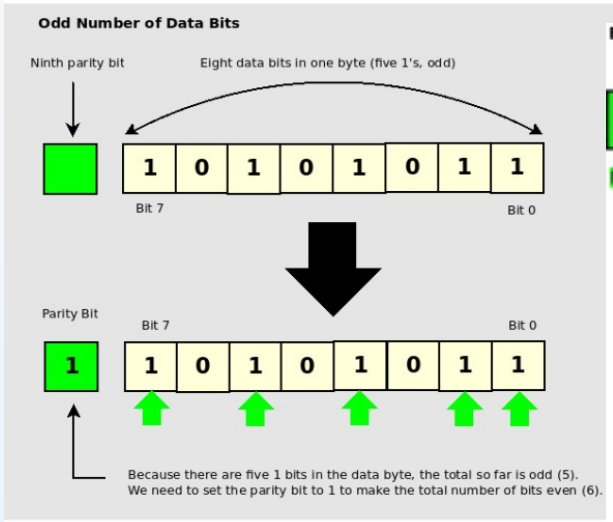
Sequential read and write will have good performance.
Random read and write will have worst performance.

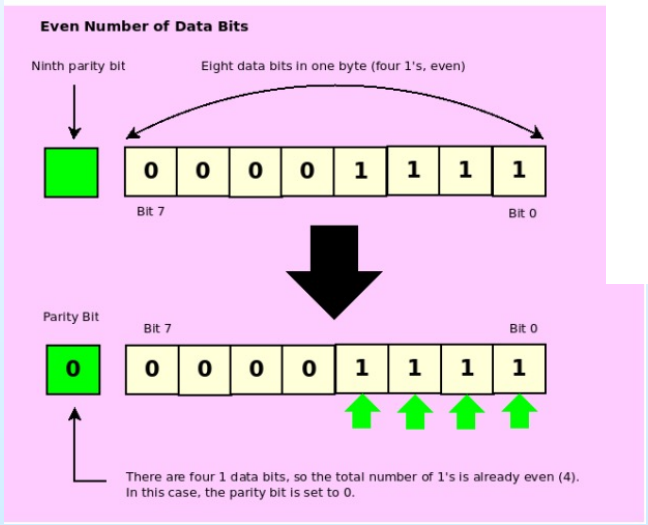


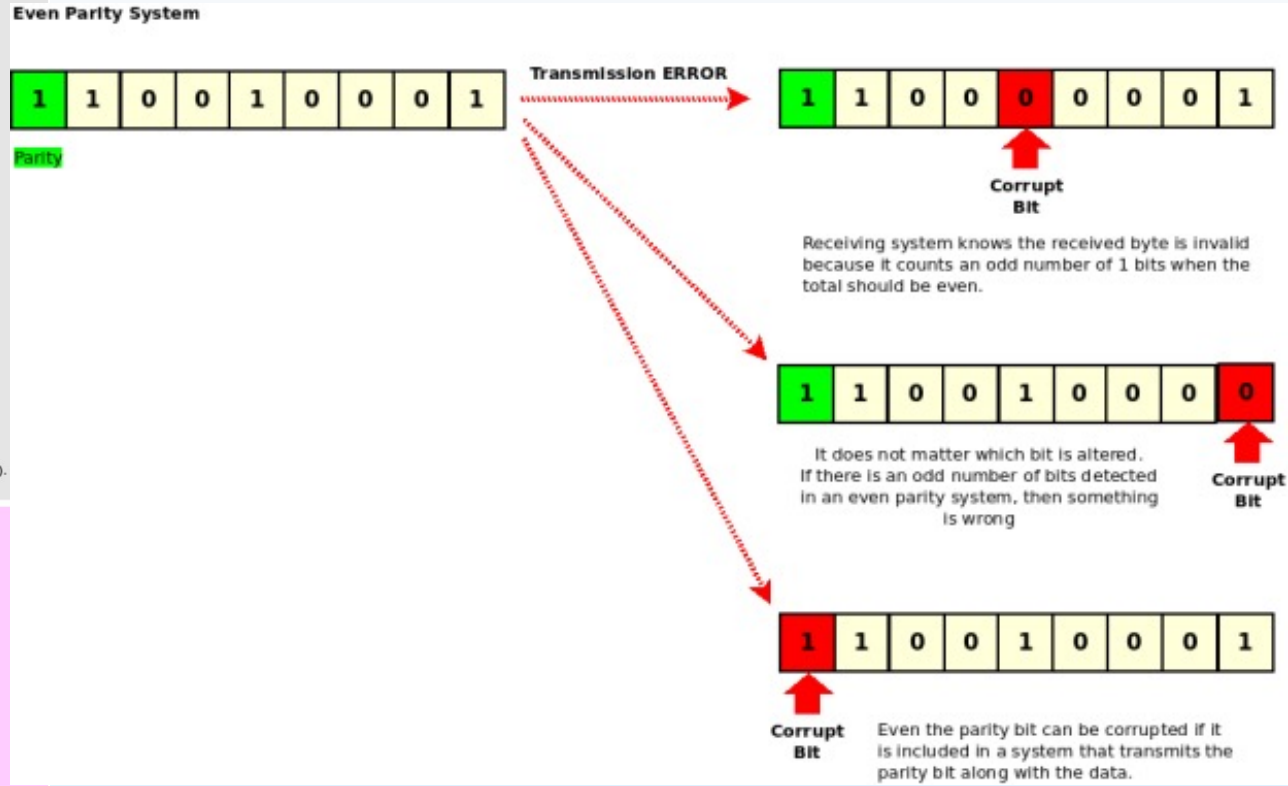
Parity (Even) Bit

Even Parity

Even parity ensures that the number of 1 bits (8 data bits + 1 parity bit) is even.

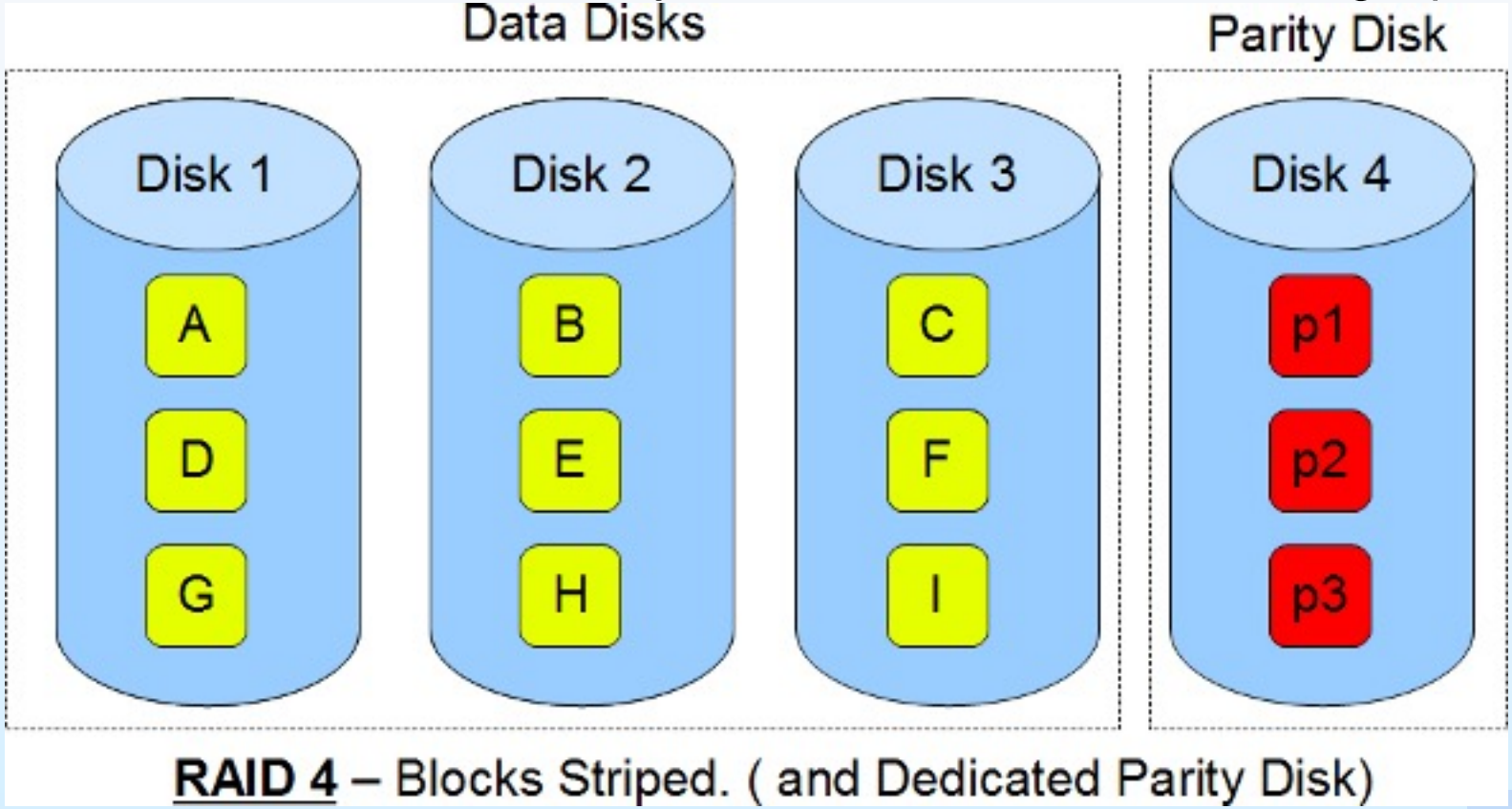






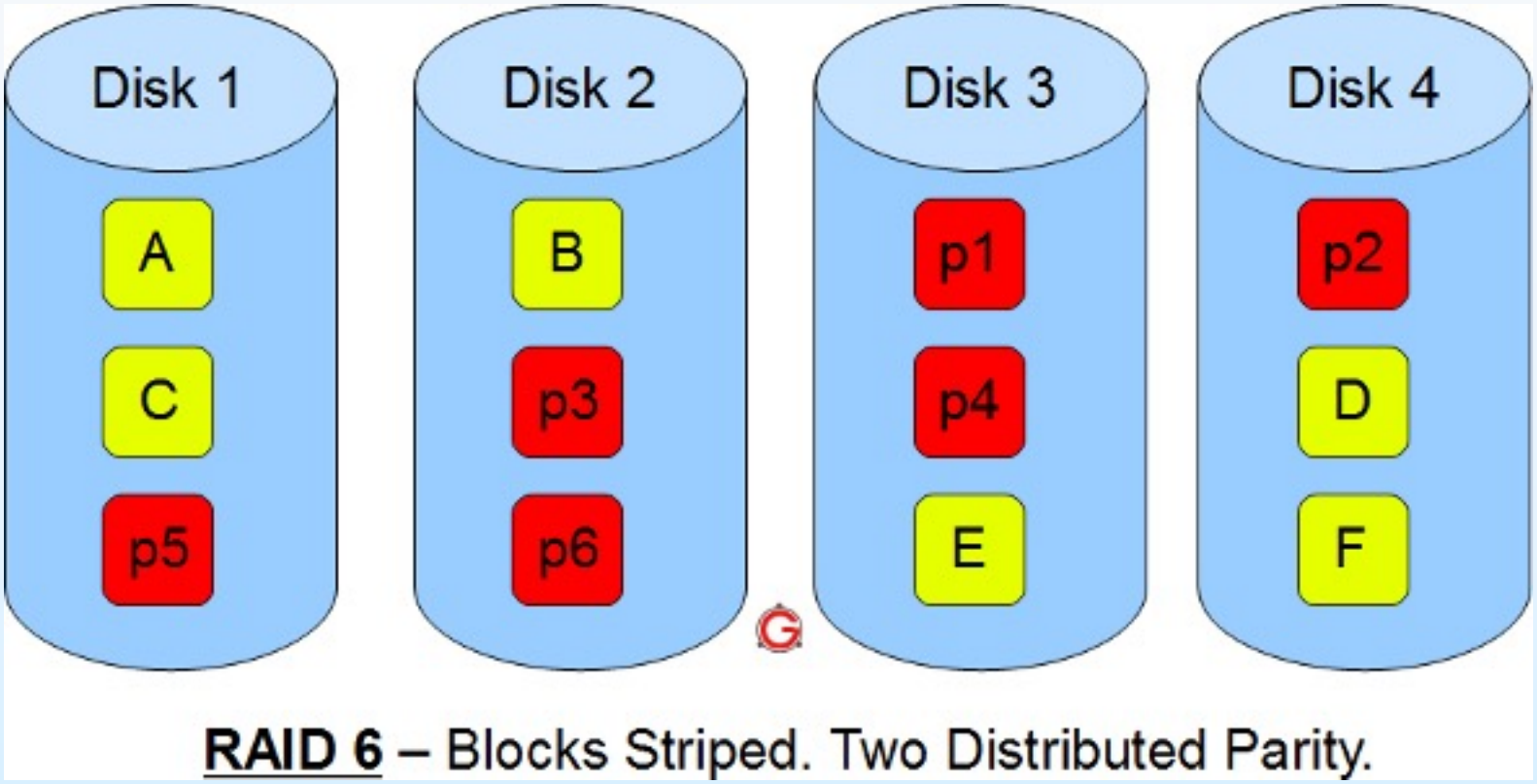
RAID 4

Minimum of 3 disks (2 disks for data and 1 for parity)
Good random reads, as the data blocks are striped.
Bad random writes, as for every write, it has to write to the single parity disk.



RAID 4 – Blocks Striped. (and Dedicated Parity Disk)

RAID 6

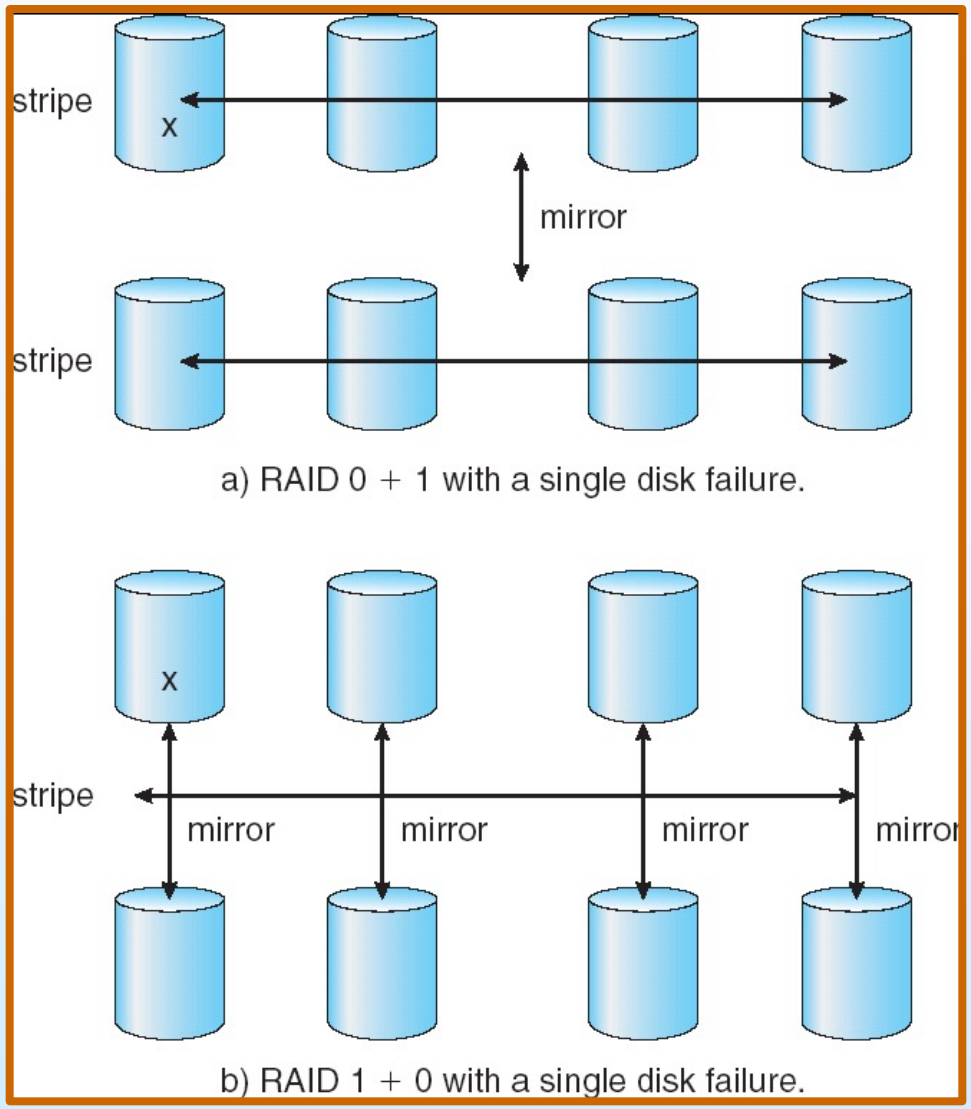


RAID 6 – Blocks Striped. Two Distributed Parity.

RAID Level

- RAID 1: Provides fault tolerance from disk errors and failure of all but one of the drives. Increased read performance occurs when using a multi-threaded operating system that supports split seeks, very small performance reduction when writing. Array continues to operate so long as at least one drive is functioning.
- RAID 2: memory-style Hamming code parity. Not really practical.
- RAID 3: The single parity disk is a bottle-neck for writing since every write requires updating the parity data.
- RAID 4: Identical with RAID 3 except using block level parity. The parity disk can become a bottleneck. Large (parallel) reads and writes are good. Small write causes 4 page accesses.
- RAID 5: Distributed parity requires all drives but one to be present to operate; drive failure requires replacement, but the array is not destroyed by a single drive failure.
- RAID 6: extra redundancy information to guard multiple disk failures.

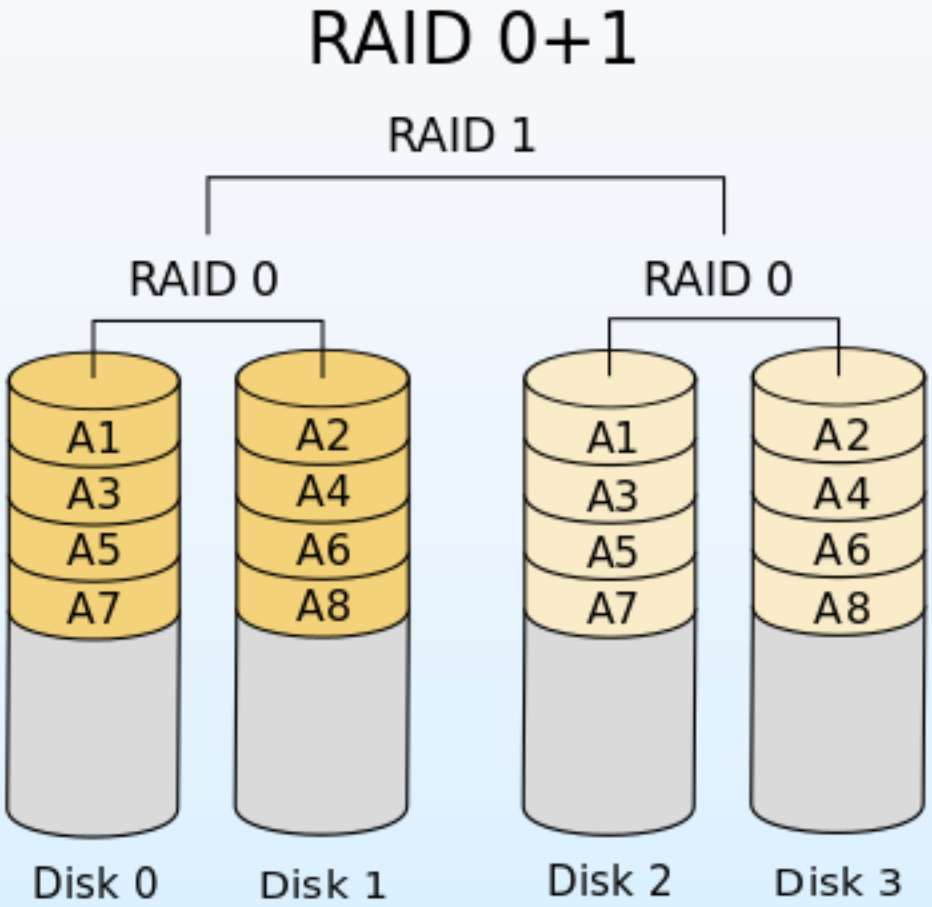
RAID (0 + 1) and (1 + 0)



Stripe and then mirror

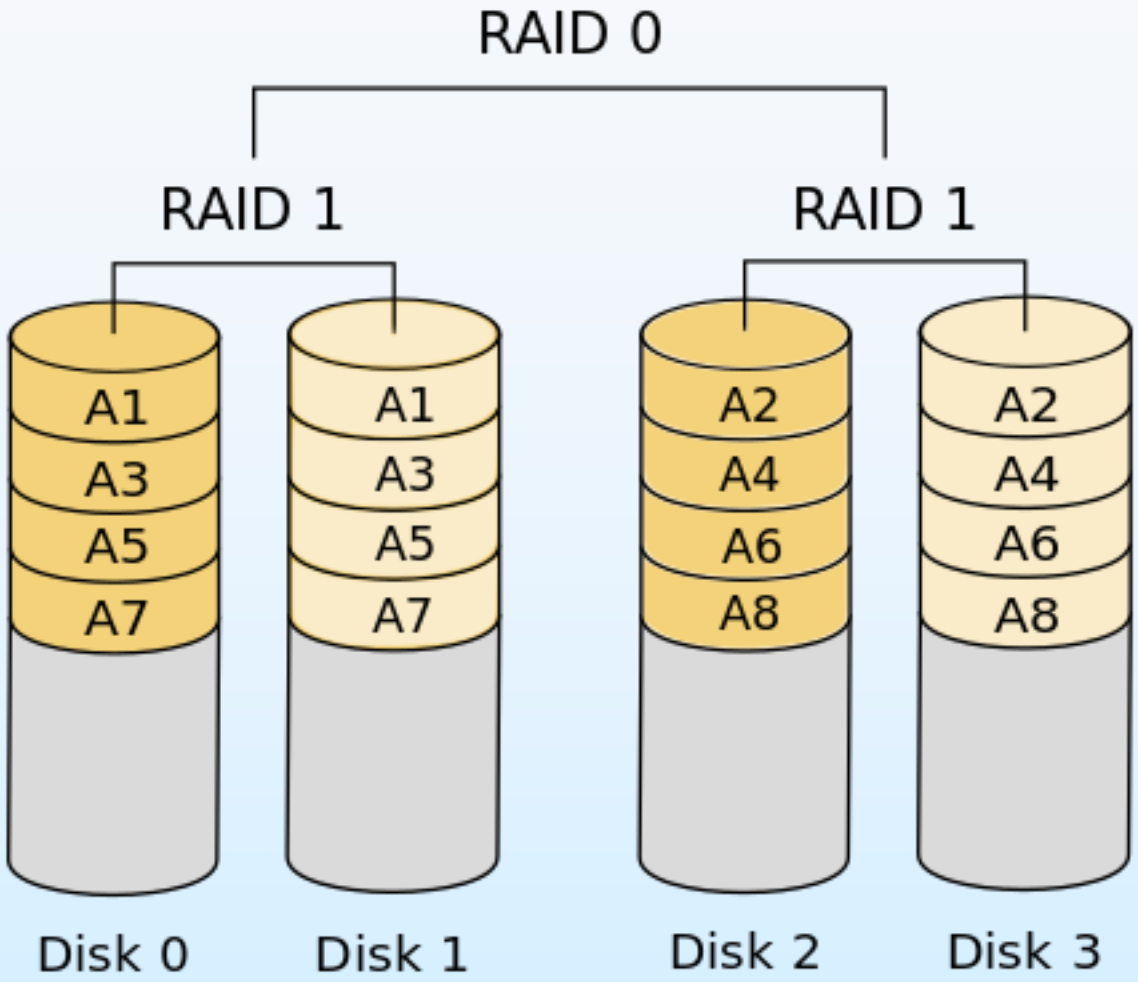
Mirror and then stripe

RAID 0+1

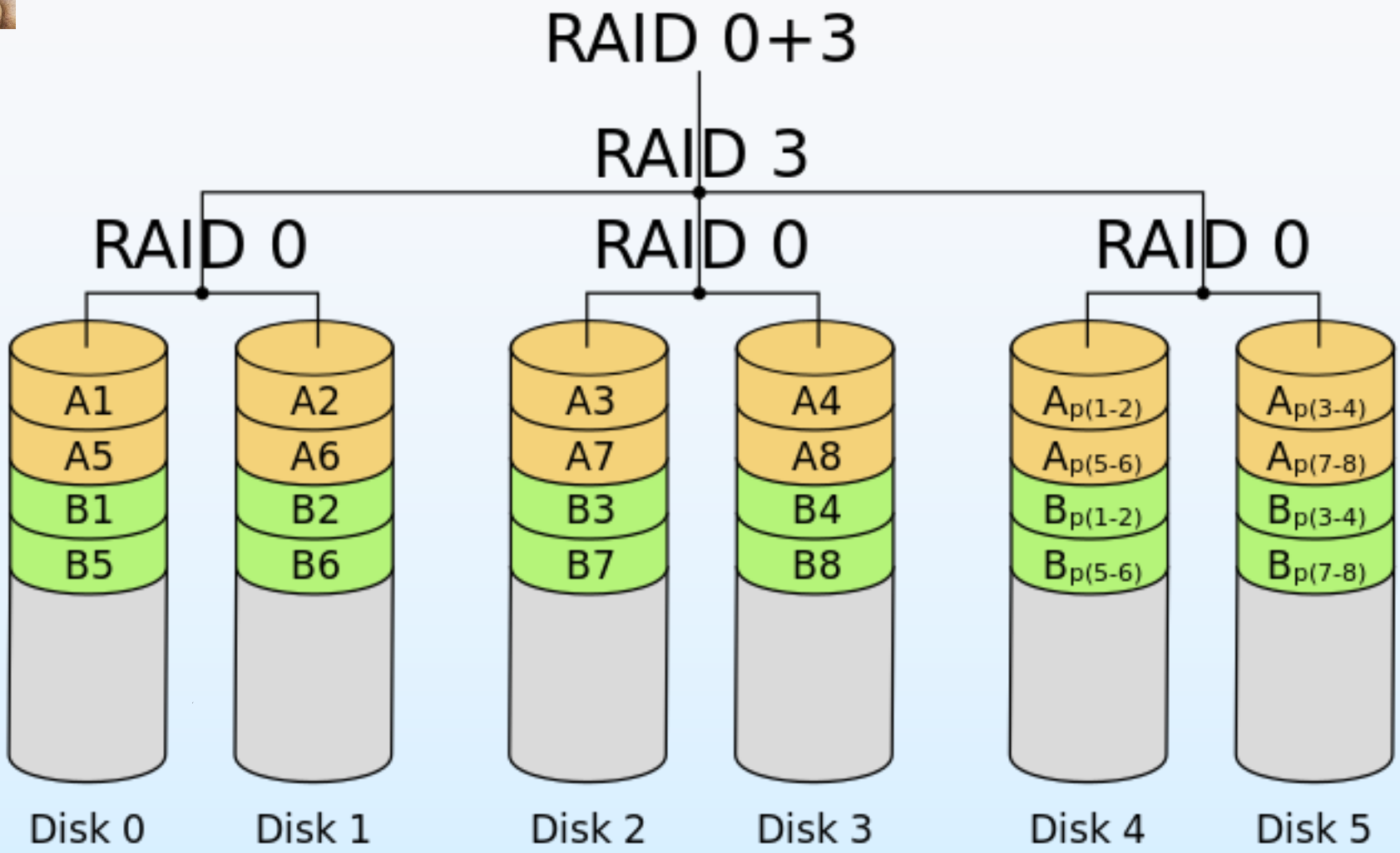


RAID 1+0

RAID 1+0

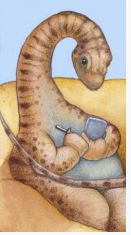


RAID 0+3



Stable-Storage Implementation

- Write-ahead log scheme requires stable storage.
- To implement stable storage:
  - Replicate information on more than one nonvolatile storage media with independent failure modes.
  - Update information in a controlled manner to ensure that we can recover the stable data after any failure during data transfer or recovery.



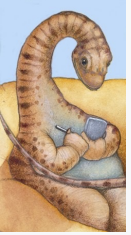
ARIES Log

LogRecord fields:

- prevLSN
- XID
- type
- update records only:
  - pageID
  - length
  - offset
  - before-image
  - after-image

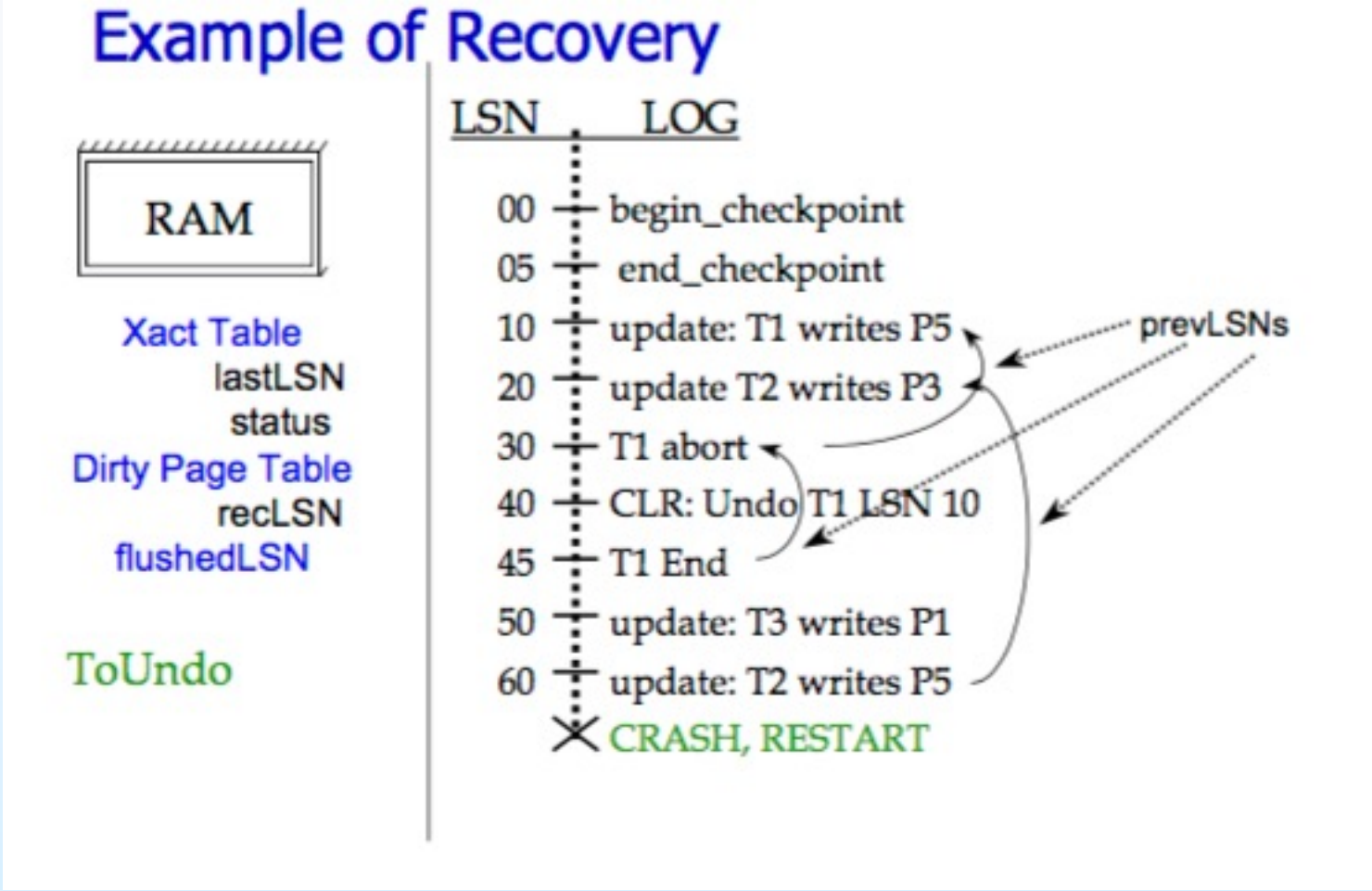
Possible log record types:

- Update
- Commit
- Abort
- End (signifies end of commit or abort)
- Compensation Log Records (CLRs)
  - for UNDO actions
  - (and some other tricks!)



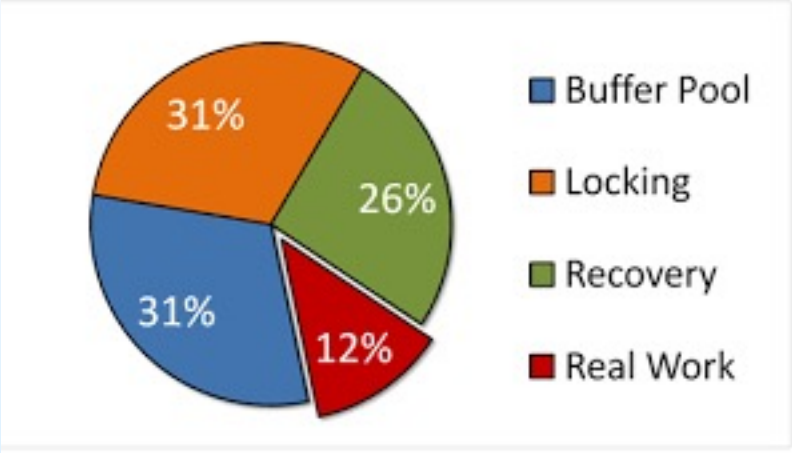
ARIES Log

Example of Recovery



What's the Problem??

- Most CPU time is not used to do the right thing.



Command Log (ICDE 2015、SIGMOD 2016)

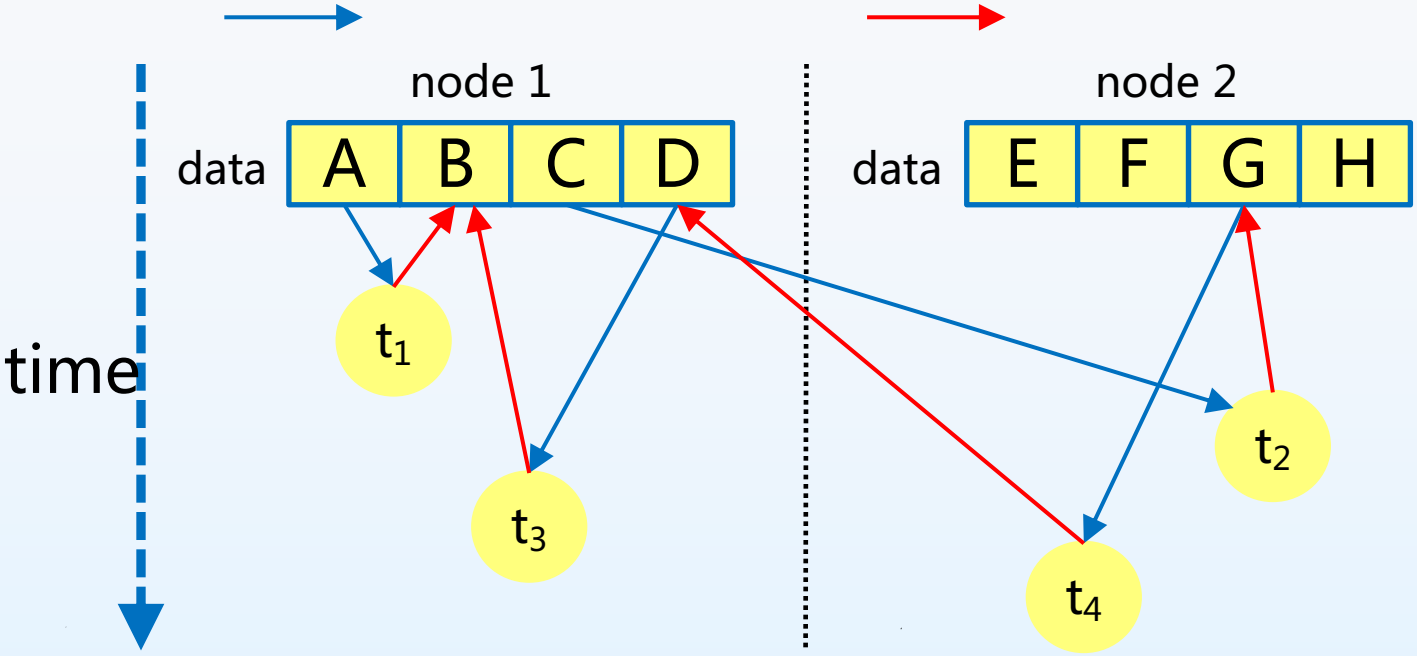


Fig 1. Example of logging techniques

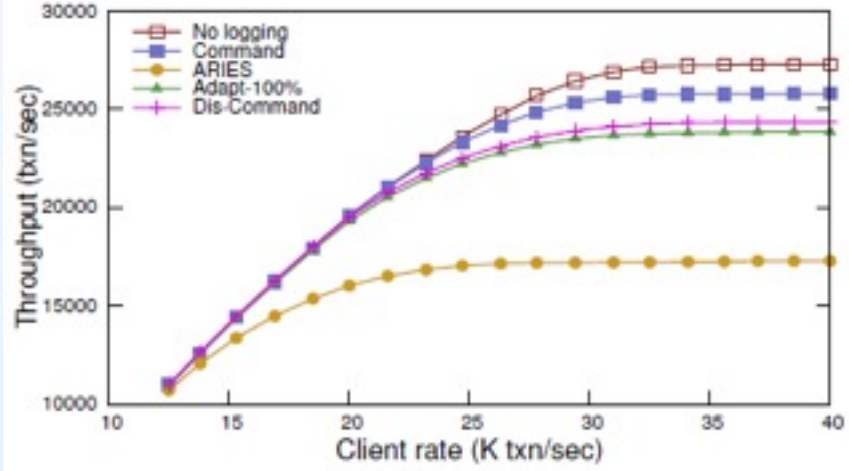
Table 1: ARIES log

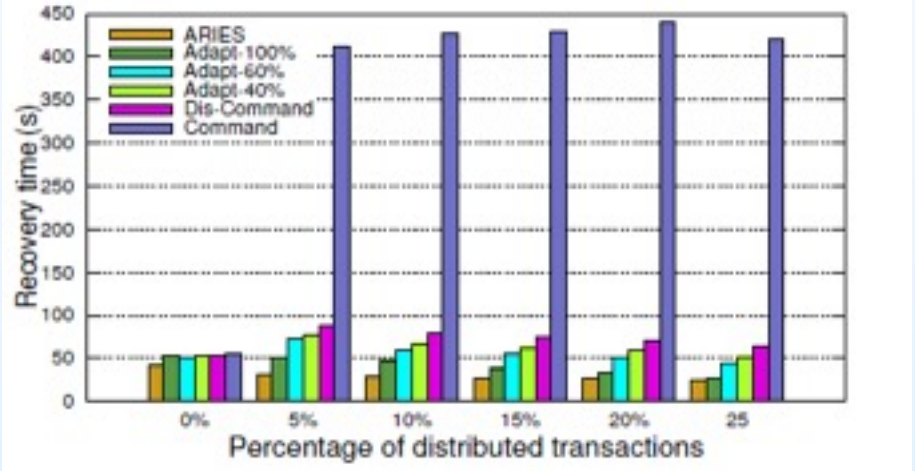
timestamp	transaction ID	parameter	old value	new value
100001	t_1	B	v(B)	2v(A)
100002	t_2	G	v(G)	2v(C)
100003	t_3	B	v(B)	2v(D)
100004	t_4	D	v(D)	2v(G)

Table 2: Command log

transaction ID	timestamp	stored procedure pointer	parameters
1	100001	p	A, B
2	100002	p	C, G
3	100003	p	D, B
4	100004	p	G, D

Command Log (ICDE 2015、SIGMOD 2016)





Command Log (ICDE 2015、SIGMOD 2016)

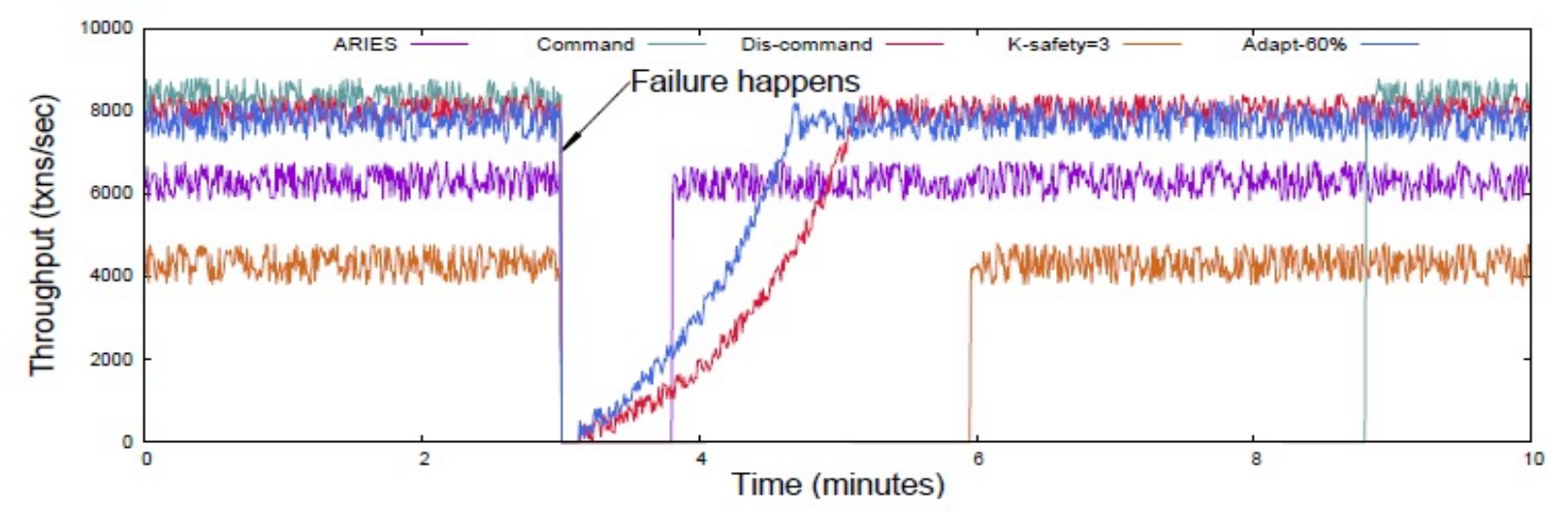
	ARIES logging	Command Logging
Number of Log Records	More	Less
Log Size	Large	Small
Complexity of Log Construction	High	Low

OLTP system with Command Logging achieves higher performance during the runtime.

	ARIES logging	Command Logging
Transaction Replay	Fast	Slow
Recovery	Only the failed nodes invoke recovery processes and work independently	The nodes in the cluster invoke recovery processes simultaneously and in a synchronous way

OLTP system with ARIES logging achieves fast and independent recovery.

Entire-cluster failure



Throughput evaluation on the TPC-C benchmark when the entire cluster fails

Tertiary Storage Devices

- Low cost is the defining characteristic of tertiary storage.
- Generally, tertiary storage is built using *removable media*
- Common examples of removable media are floppy disks and CD-ROMs; other types are available.

Removable Disks

- Floppy disk — thin flexible disk coated with magnetic material, enclosed in a protective plastic case.
  - Most floppies hold about 1 MB; similar technology is used for removable disks that hold more than 1 GB.
  - Removable magnetic disks can be nearly as fast as hard disks, but they are at a greater risk of damage from exposure.

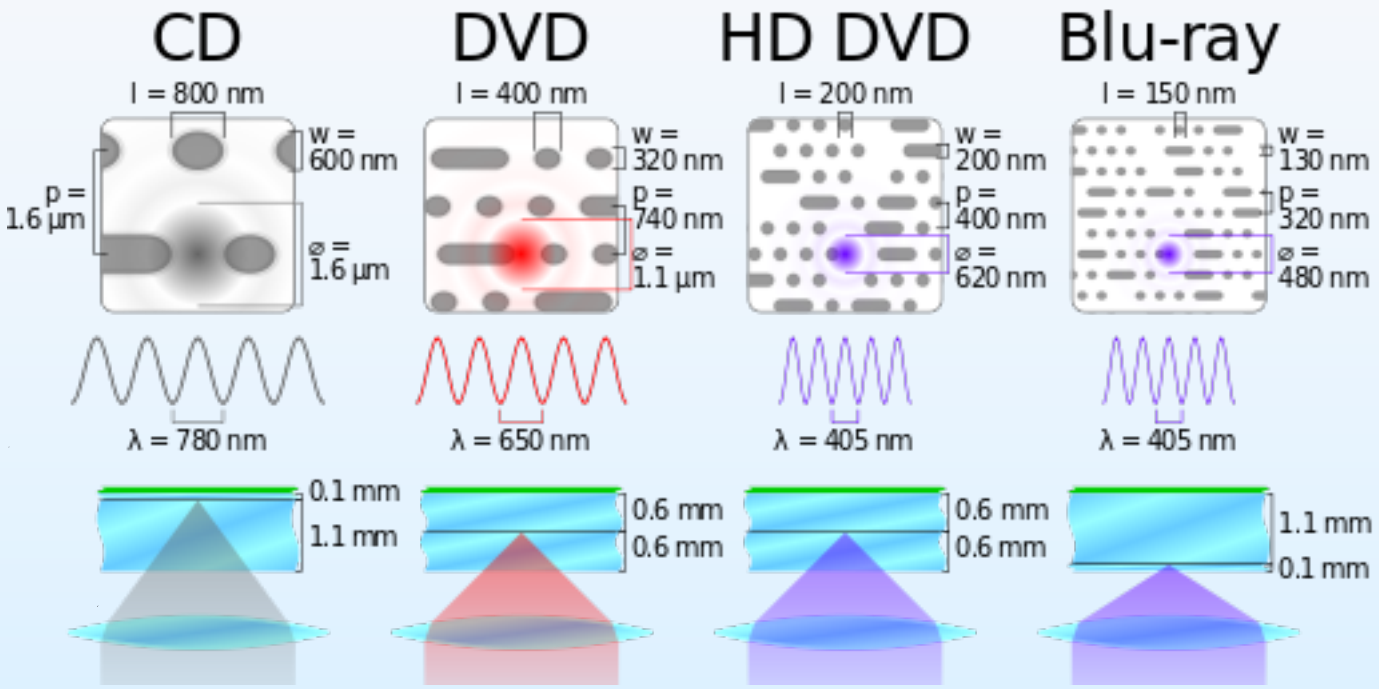
Removable Disks (Cont.)

- A magneto-optic disk records data on a rigid platter coated with magnetic material.
  - Laser heat is used to amplify a large, weak magnetic field to record a bit.
  - Laser light is also used to read data (Kerr effect).
  - The magneto-optic head flies much farther from the disk surface than a magnetic disk head, and the magnetic material is covered with a protective layer of plastic or glass; resistant to head crashes.

$$\Delta n = \lambda K E^2$$

- Optical disks do not use magnetism; they employ special materials that are altered by laser light.

Evolvement of Optical Disk



WORM Disks

- The data on read-write disks can be modified over and over.
- WORM ("Write Once, Read Many Times") disks can be written only once.
- Thin aluminum film sandwiched between two glass or plastic platters.
- To write a bit, the drive uses a laser light to burn a small hole through the aluminum; information can be destroyed but not altered.
- Very durable and reliable.
- *Read Only* disks, such ad CD-ROM and DVD, come from the factory with the data pre-recorded.

Tapes

- Compared to a disk, a tape is less expensive and holds more data, but random access is much slower.
- Tape is an economical medium for purposes that do not require fast random access, e.g., backup copies of disk data, holding huge volumes of data.
- Large tape installations typically use robotic tape changers that move tapes between tape drives and storage slots in a tape library.
  - stacker – library that holds a few tapes
  - silo – library that holds thousands of tapes
- A disk-resident file can be *archived* to tape for low cost storage; the computer can *stage* it back into disk storage for active use.

Tape Library

Operating System Issues

- Major OS jobs are to manage physical devices and to present a virtual machine abstraction to applications
- For hard disks, the OS provides two abstraction:
  - Raw device – an array of data blocks.
  - File system – the OS queues and schedules the interleaved requests from several applications.

Application Interface

- Most OSs handle removable disks almost exactly like fixed disks — a new cartridge is formatted and an empty file system is generated on the disk.
- Tapes are presented as a raw storage medium, i.e., and application does not open a file on the tape, it opens the whole tape drive as a raw device.
- Usually the tape drive is reserved for the exclusive use of that application.
- Since the OS does not provide file system services, the application must decide how to use the array of blocks.
- Since every application makes up its own rules for how to organize a tape, a tape full of data can generally only be used by the program that created it.

Tape Drives

- The basic operations for a tape drive differ from those of a disk drive.
- **locate** positions the tape to a specific logical block, not an entire track (corresponds to **seek**).
- The **read position** operation returns the logical block number where the tape head is.
- The **space** operation enables relative motion.
- Tape drives are “append-only” devices; updating a block in the middle of the tape also effectively erases everything beyond that block.
- An EOT mark is placed after a block that is written.

File Naming

- The issue of naming files on removable media is especially difficult when we want to write data on a removable cartridge on one computer, and then use the cartridge in another computer.
- Contemporary OSs generally leave the name space problem unsolved for removable media, and depend on applications and users to figure out how to access and interpret the data.
- Some kinds of removable media (e.g., CDs) are so well standardized that all computers use them the same way.

Hierarchical Storage Management (HSM)

- A hierarchical storage system extends the storage hierarchy beyond primary memory and secondary storage to incorporate tertiary storage — usually implemented as a jukebox of tapes or removable disks.
- Usually incorporate tertiary storage by extending the file system.
  - Small and frequently used files remain on disk.
  - Large, old, inactive files are archived to the jukebox.
- HSM is usually found in supercomputing centers and other large installations that have enormous volumes of data.

A Jukebox

Speed

- Two aspects of speed in tertiary storage are bandwidth and latency.

- Bandwidth is measured in bytes per second.
  - Sustained bandwidth – average data rate during a large transfer; # of bytes/transfer time.
    Data rate when the data stream is actually flowing.
  - Effective bandwidth – average over the entire I/O time, including **seek** or **locate**, and cartridge switching.
    Drive's overall data rate.

Speed (Cont.)

- Access latency – amount of time needed to locate data.
  - Access time for a disk – move the arm to the selected cylinder and wait for the rotational latency; < 35 milliseconds.
  - Access on tape requires winding the tape reels until the selected block reaches the tape head; tens or hundreds of seconds.
  - Generally say that random access within a tape cartridge is about a thousand times slower than random access on disk.
- The low cost of tertiary storage is a result of having many cheap cartridges share a few expensive drives.
- A removable library is best devoted to the storage of infrequently used data, because the library can only satisfy a relatively small number of I/O requests per hour.

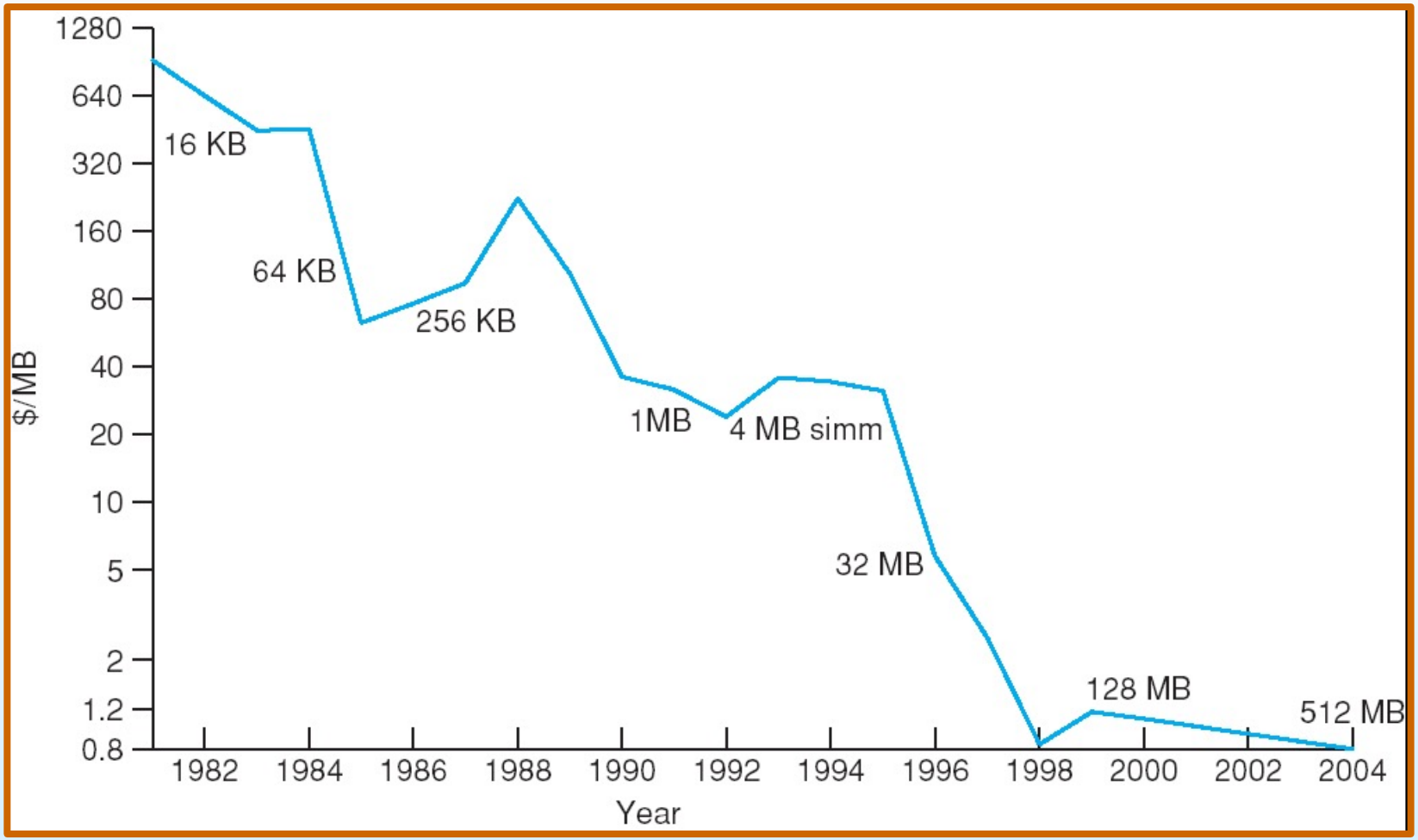
Reliability

- A fixed disk drive is likely to be more reliable than a removable disk or tape drive.
- An optical cartridge is likely to be more reliable than a magnetic disk or tape.
- A head crash in a fixed hard disk generally destroys the data, whereas the failure of a tape drive or optical disk drive often leaves the data cartridge unharmed.

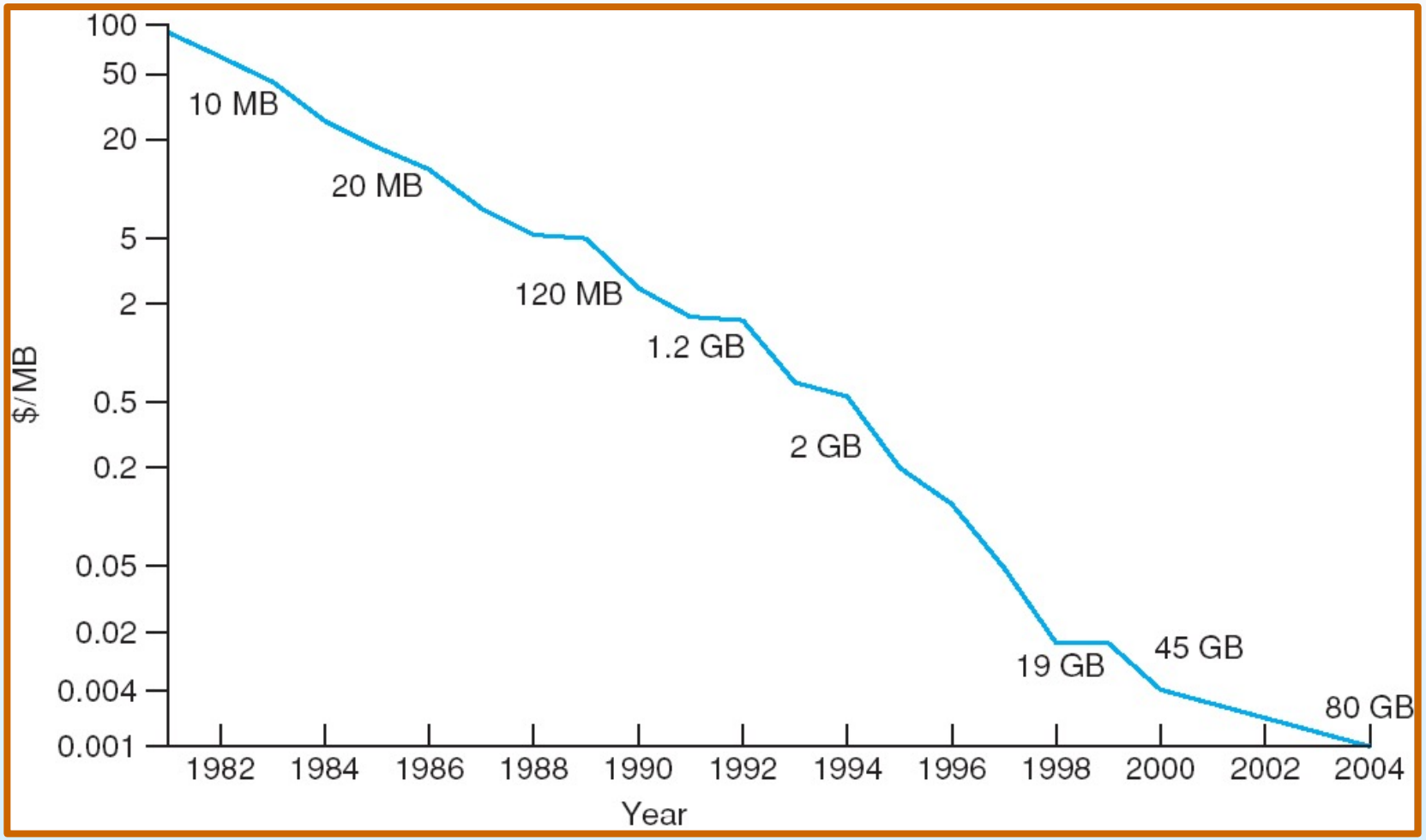
Cost

- Main memory is much more expensive than disk storage
- The cost per megabyte of hard disk storage is competitive with magnetic tape if only one tape is used per drive.
- The cheapest tape drives and the cheapest disk drives have had about the same storage capacity over the years.
- Tertiary storage gives a cost savings only when the number of cartridges is considerably larger than the number of drives.

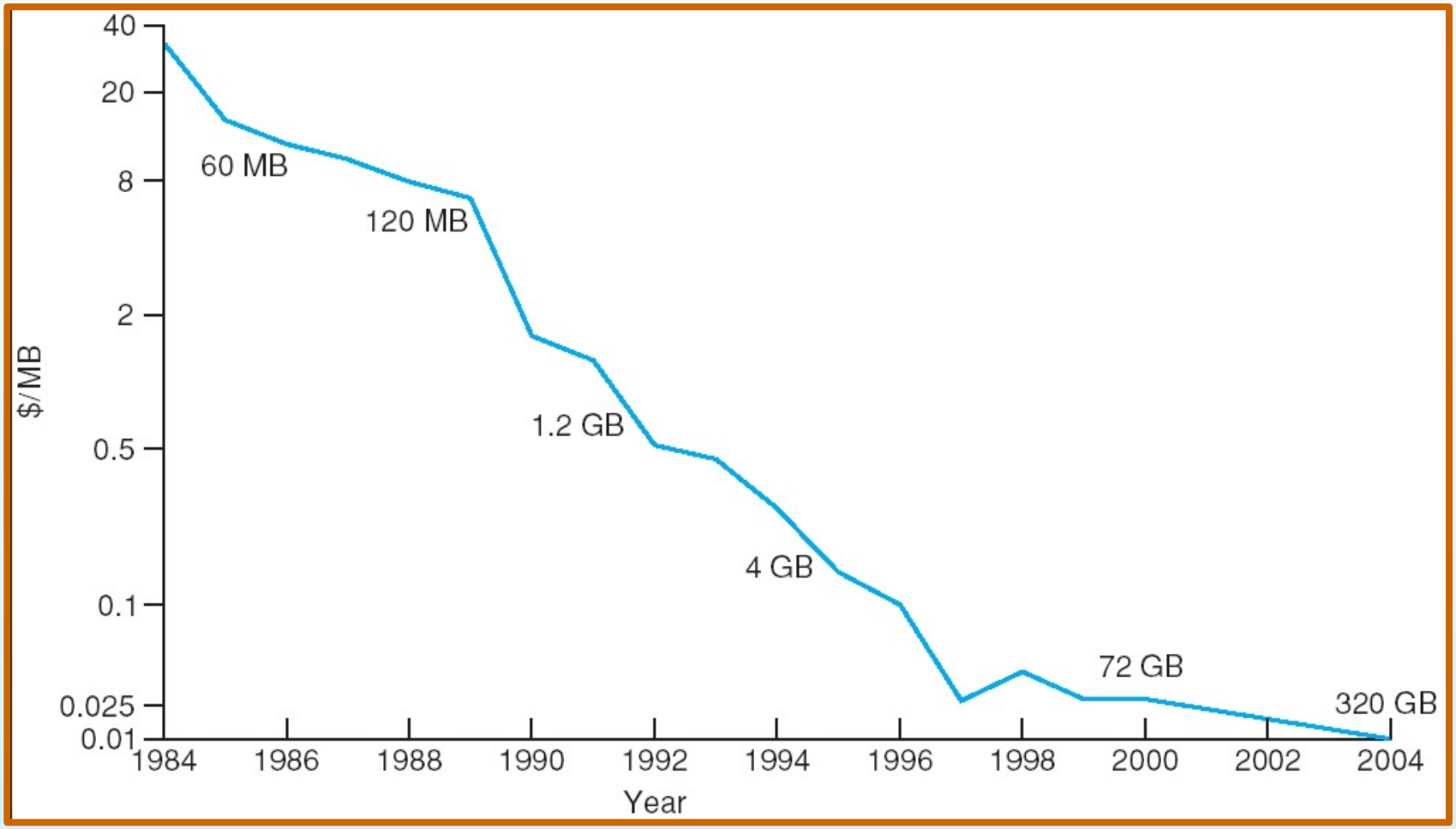
Price per Megabyte of DRAM, From 1981 to 2004



Price per Megabyte of Magnetic Hard Disk, From 1981 to 2004



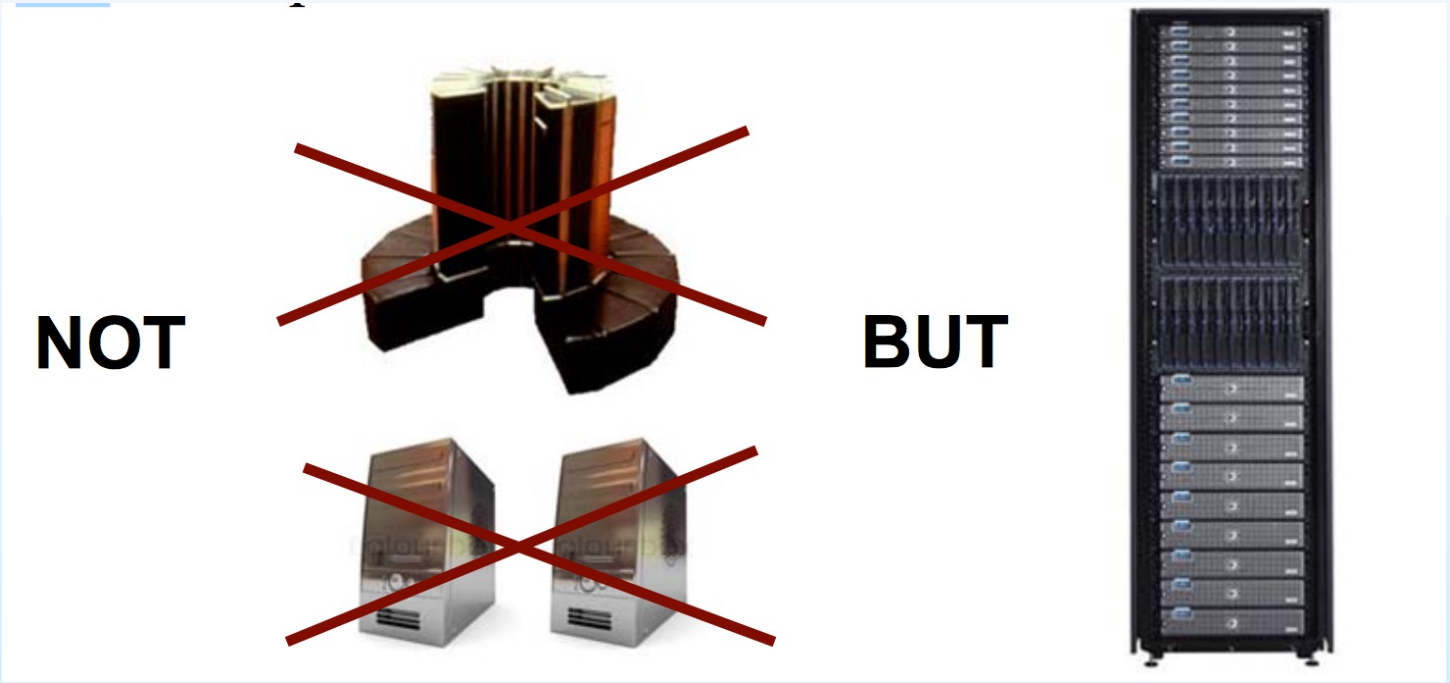
Price per Megabyte of a Tape Drive, From 1984-2000



An Introduction to MapReduce (Hadoop)

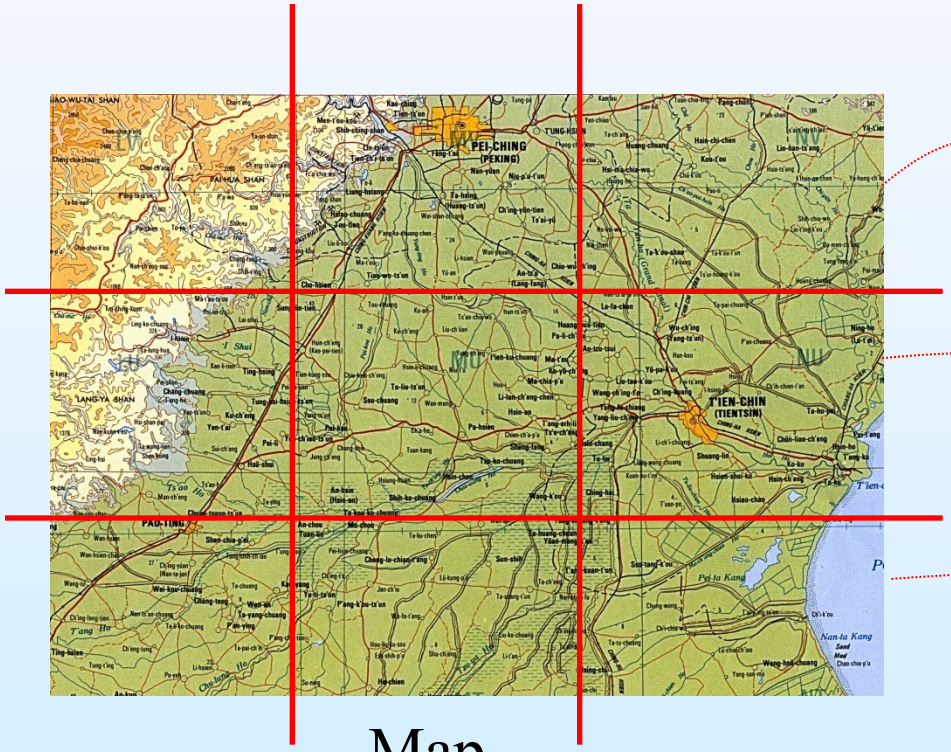
MapReduce

- To simplify the parallel processing on a large cluster, Google proposes MapReduce framework



What is MapReduce

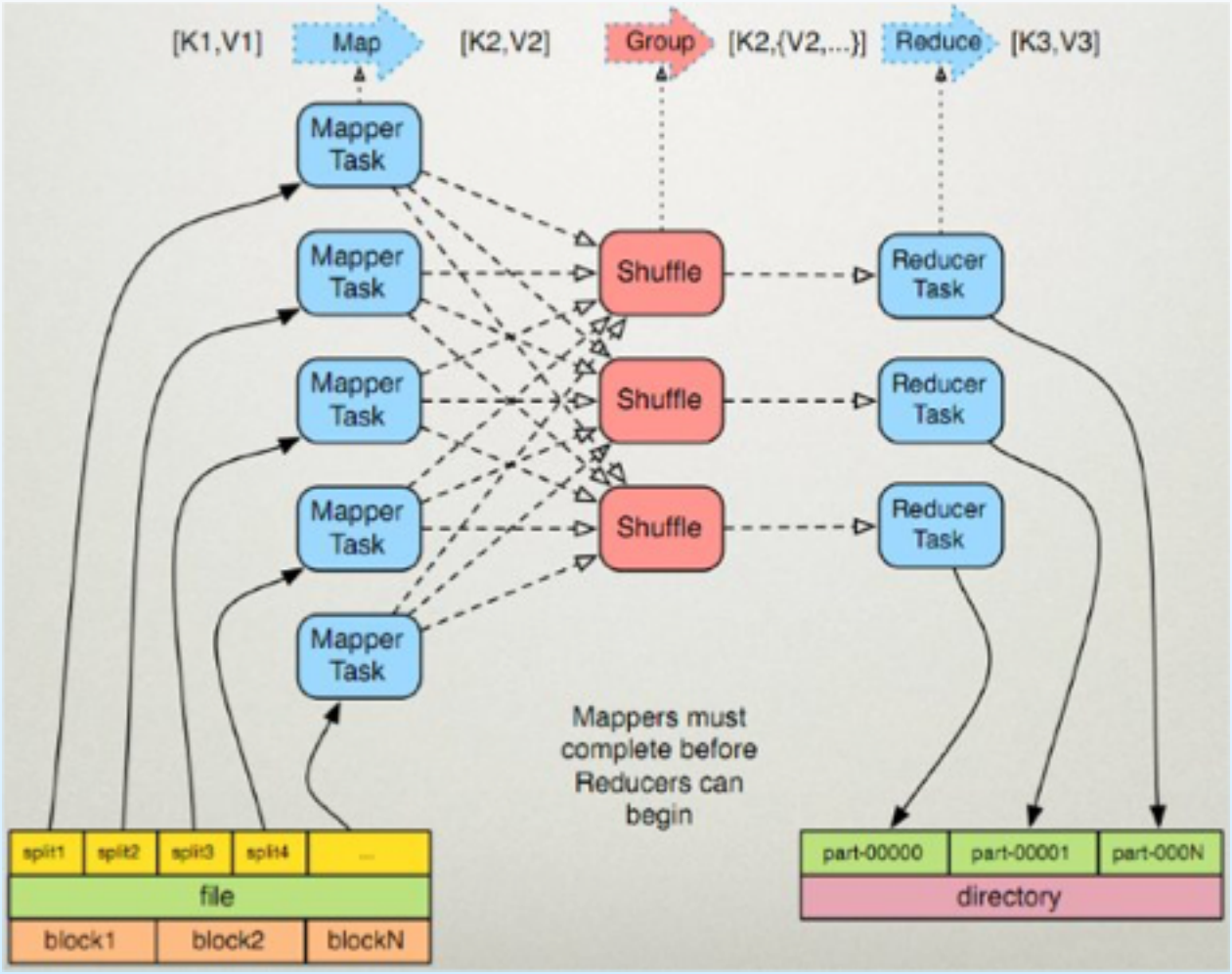
- Two customizable interfaces: map and reduce



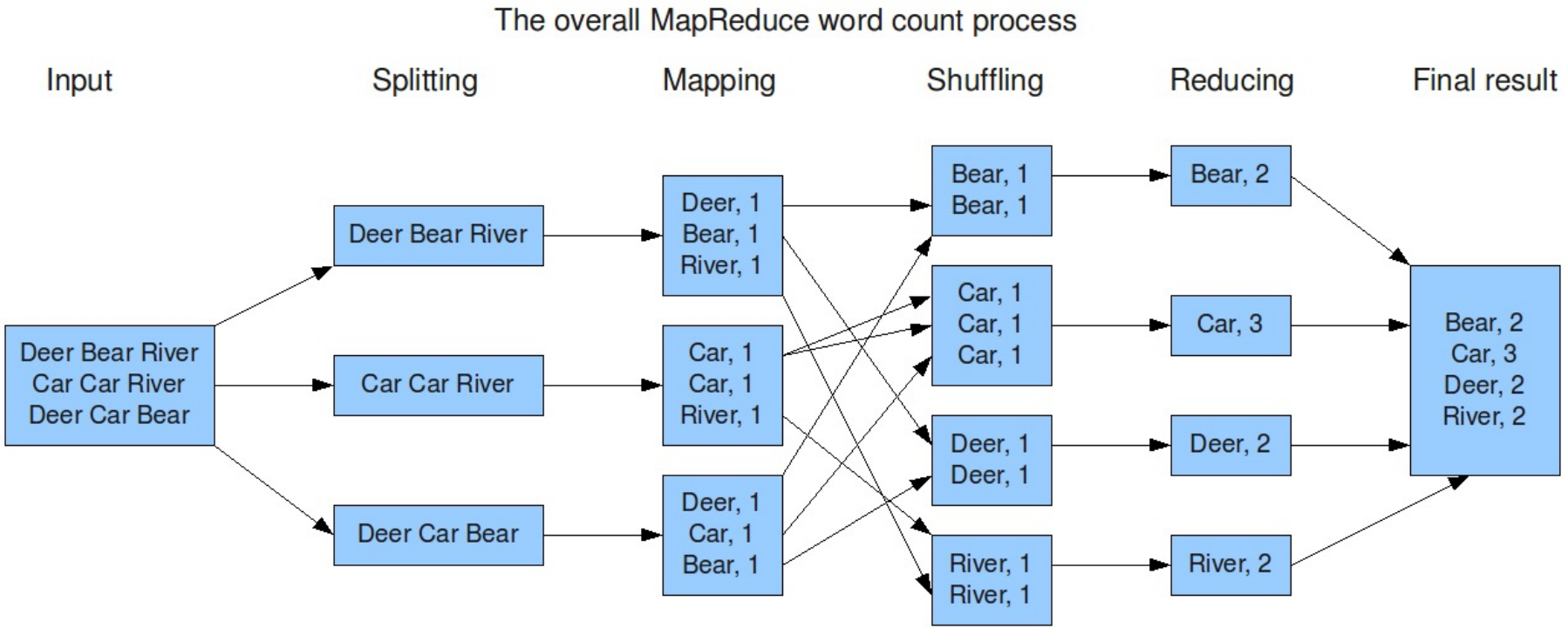
Map

Reduce

MapReduce Framework



MapReduce Job: Word Count



HDFS (Hadoop Distributed File System)

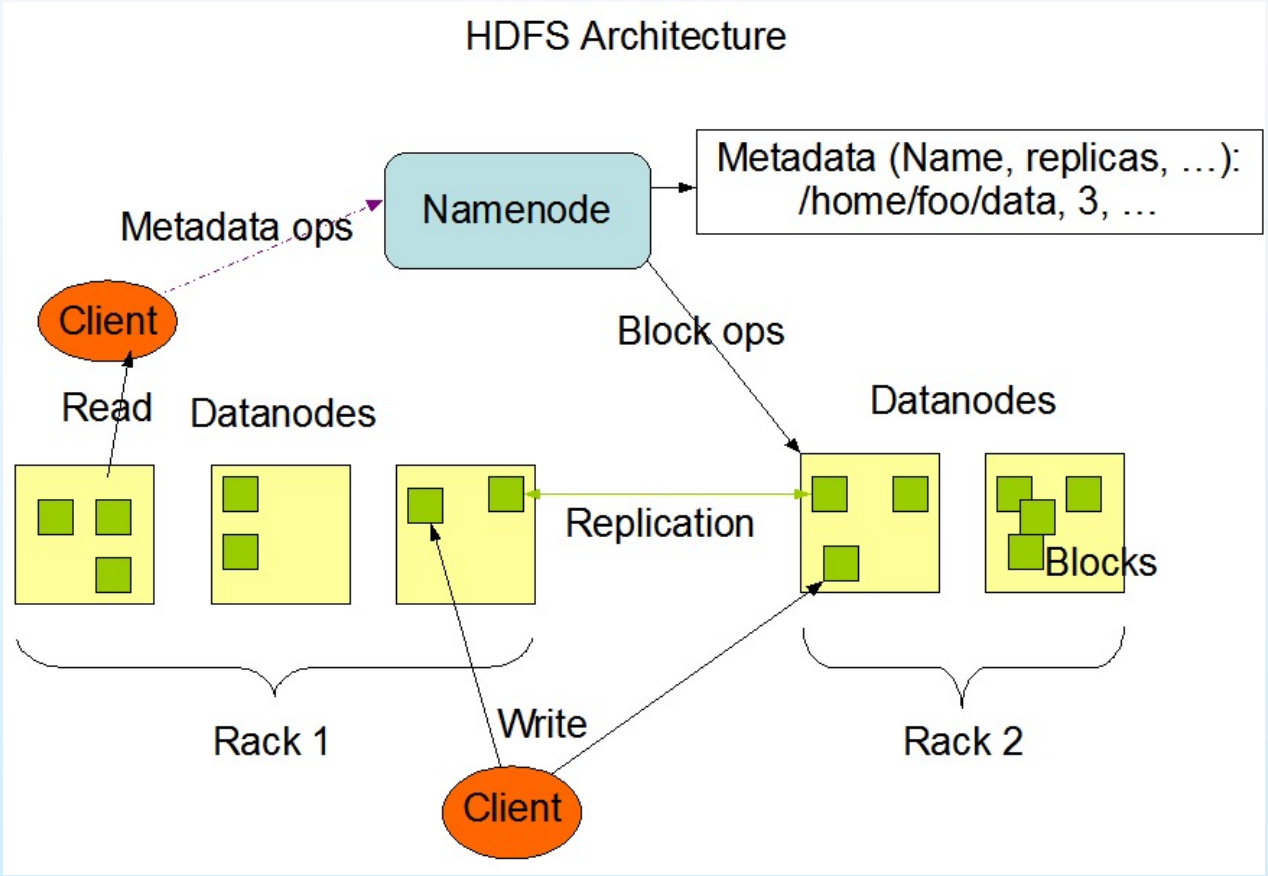
- HDFS is an open source implementation of Google File System
- HDFS is designed for
  - Storing large files
    - Terabytes, Petabytes, etc...
    - Millions rather than billions of files – 100MB or more per file
  - Streaming data
    - Write once and read-many times patterns
    - Optimized for streaming reads rather than random reads – Append operation added to Hadoop 0.21
  - “Cheap” Commodity Hardware
  - No need for super-computers, use less reliable commodity hardware

HDFS (Hadoop Distributed File System)

- HDFS is not designed for
  - Low-latency reads
    - High-throughput rather than low latency for small chunks of data
    - HBase addresses this issue
  - Large amount of small files
    - Better for millions of large files instead of billions of small files
    - For example each file can be 100MB or more • Multiple Writers
  - Single writer per file
    - Writes only at the end of file, no-support for arbitrary offset

Architecture of HDFS

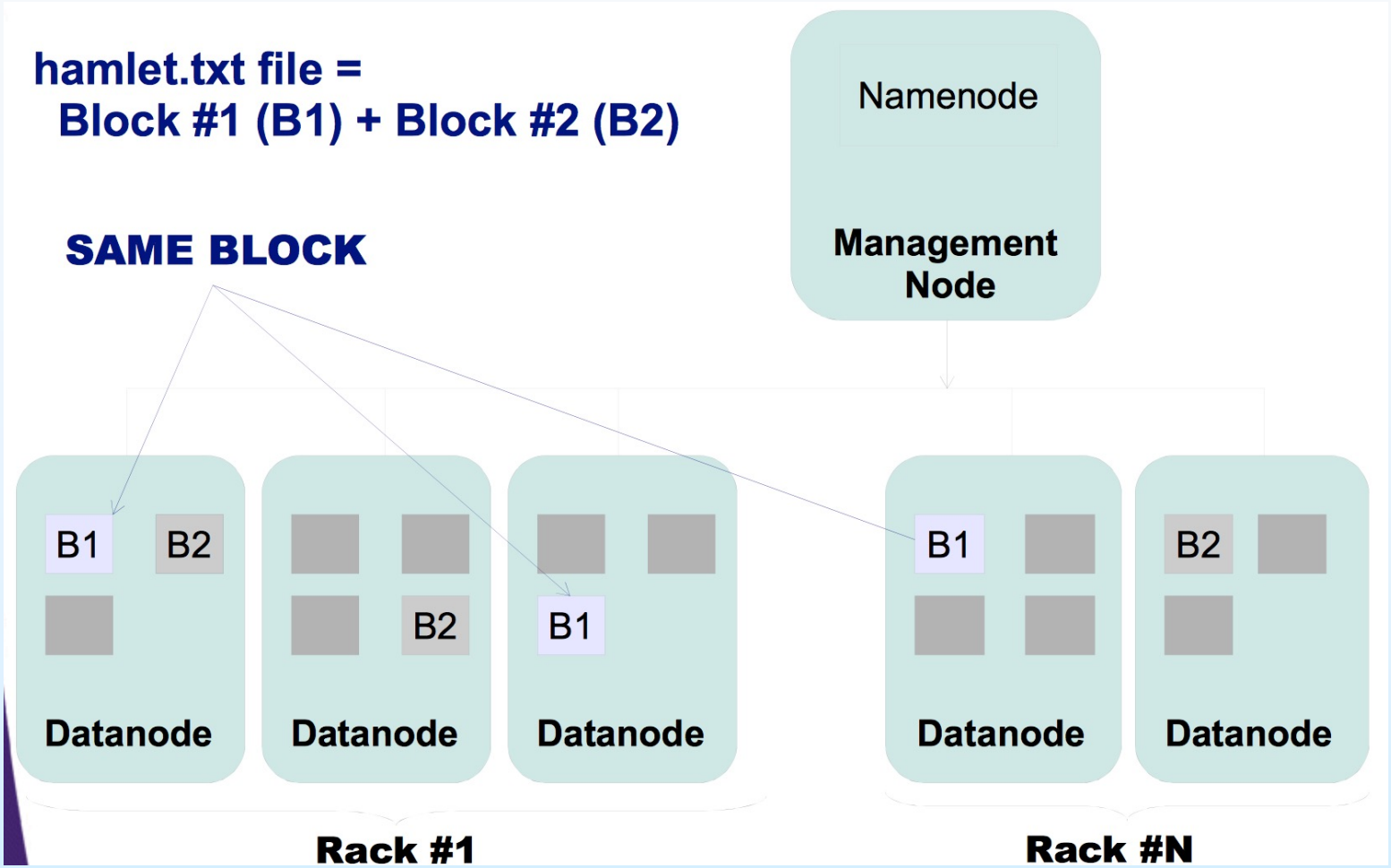
- One Namenode
- One Second Namenode
- Multiple Datanode



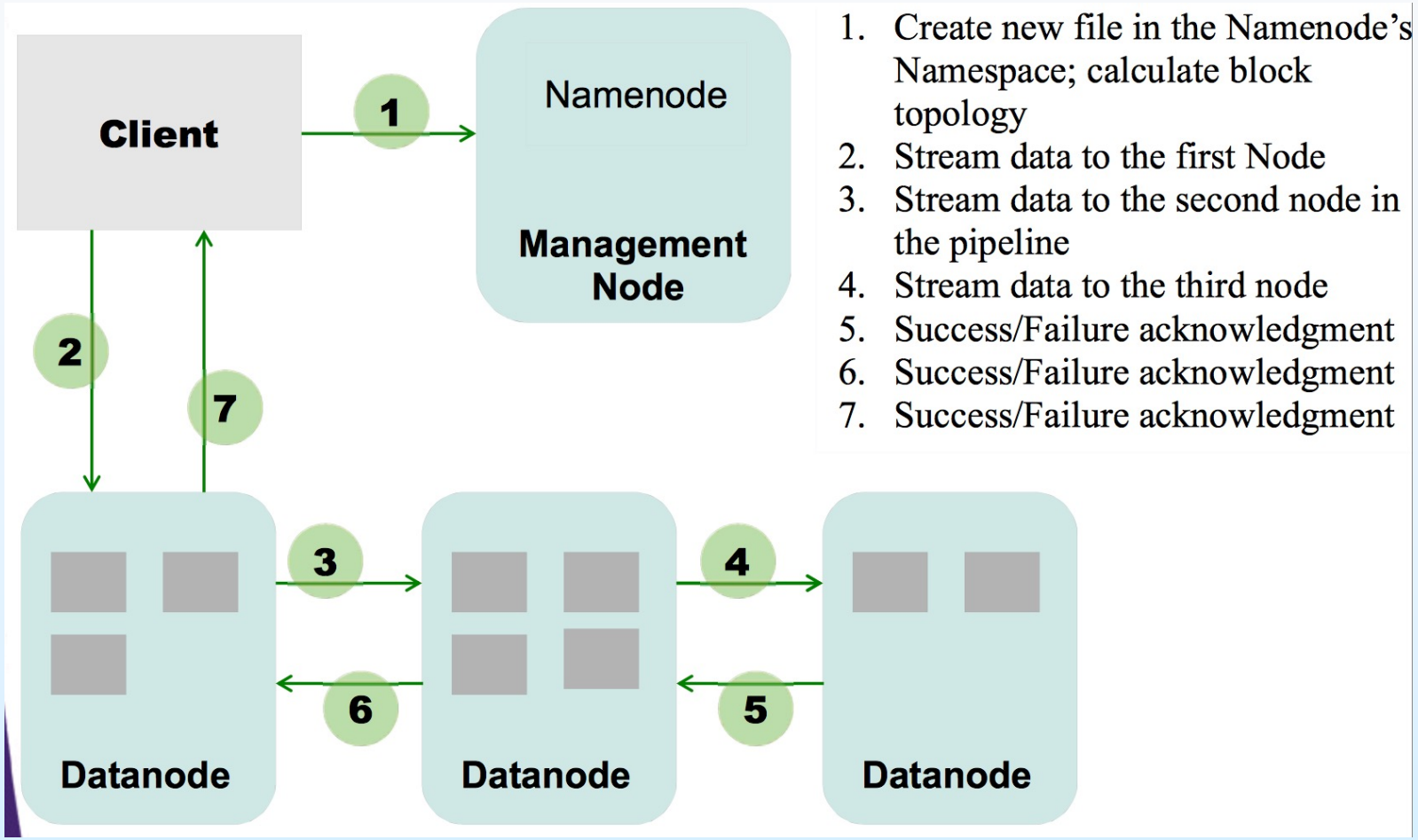
File Blocks

- Files are split into blocks (single unit of storage)
  - Managed by Namenode, stored by Datanode
  - Transparent to user
- Replicated across machines at load time
  - Same block is stored on multiple machines
  - Good for fault-tolerance and access
  - Default replication is 3
- Blocks are traditionally either 64MB or 128MB
  - Default is 64MB
- The motivation is to minimize the cost of seeks as compared to transfer rate
  - 'Time to transfer' > 'Time to seek'
- For example, lets say – seek time = 10ms
  - Transfer rate = 100 MB/s
- To achieve seek time of 1% transfer rate – Block size will need to be = 100MB

File Blocks

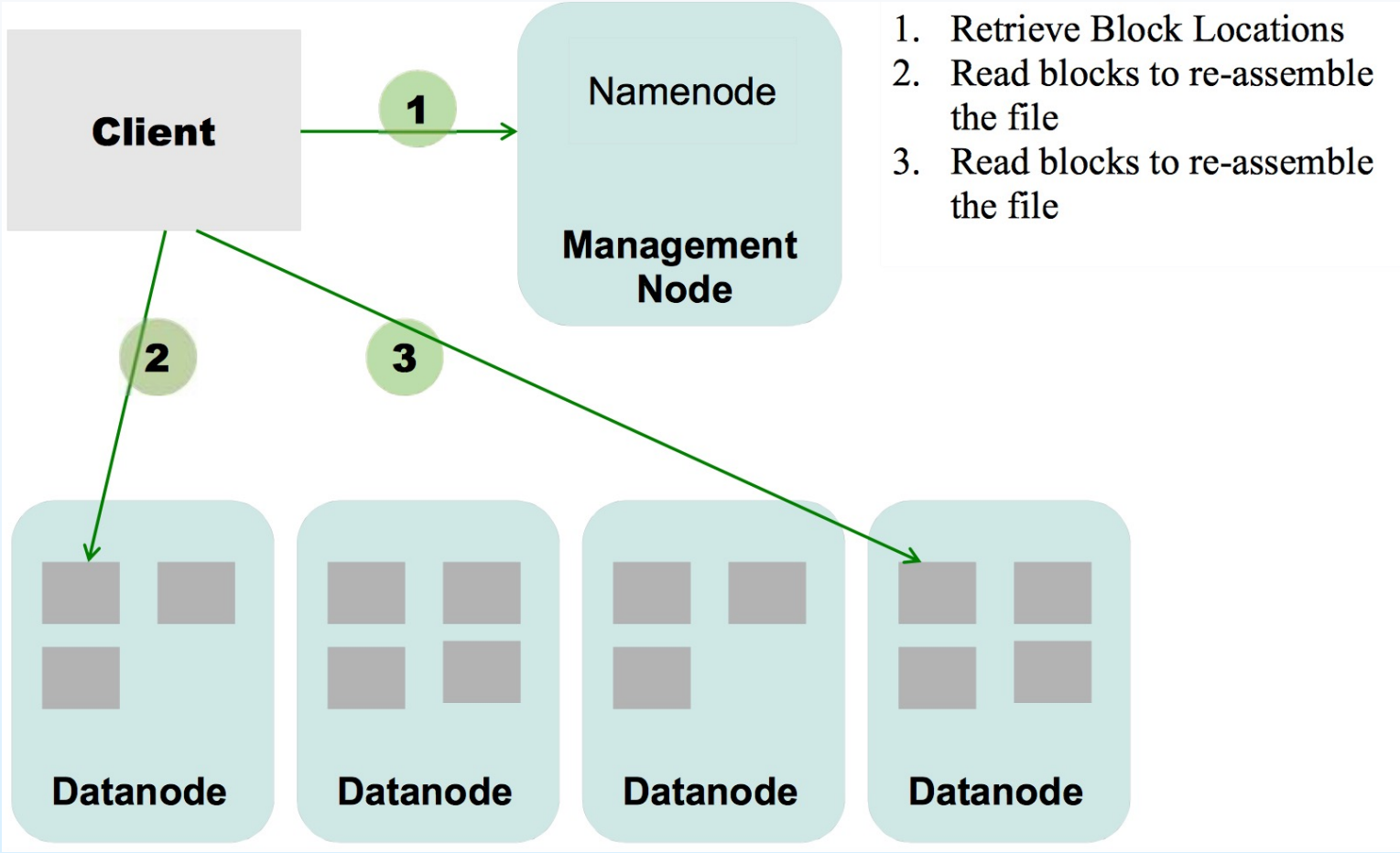

hamlet.txt file =
Block #1 (B1) + Block #2 (B2)
SAME BLOCK
Namenode
Management Node
B1
B2
B2
B1
B1
B2
Datanode
Datanode
Datanode
Datanode
Datanode
Rack #1
Rack #N

Clients Write in HDFS



1. Create new file in the Namenode's Namespace; calculate block topology
2. Stream data to the first Node
3. Stream data to the second node in the pipeline
4. Stream data to the third node
5. Success/Failure acknowledgment
6. Success/Failure acknowledgment
7. Success/Failure acknowledgment

Client Read in HDFS



1. Retrieve Block Locations
2. Read blocks to re-assemble the file
3. Read blocks to re-assemble the file

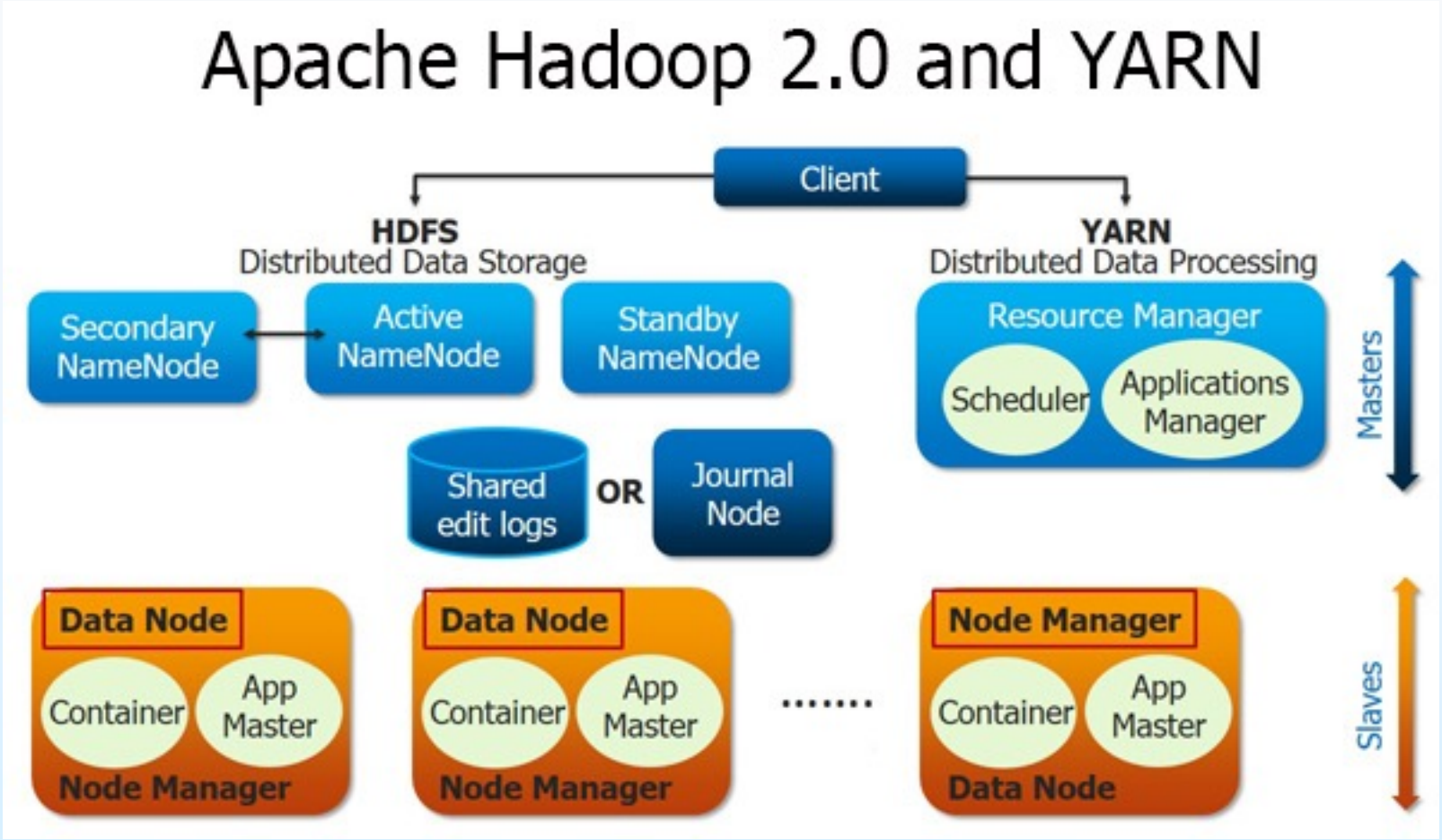
Bottleneck of the Namenode

- For fast access Namenode keeps all block metadata in-memory
  - The bigger the cluster - the more RAM required
- Best for millions of large files (100mb or more) rather than billions
  - Will work well for clusters of 100s machines
- Hadoop 2+
  - Namenode Federations
  - Each namenode will host part of the blocks • Horizontally scale the Namenode
  - Support for 1000+ machine clusters • Yahoo! runs 50,000+ machines
  - Learn more @ http://hadoop.apache.org/docs/r2.0.2-alpha/hadoop-yarn/hadoop-yarn-site/Federation.html

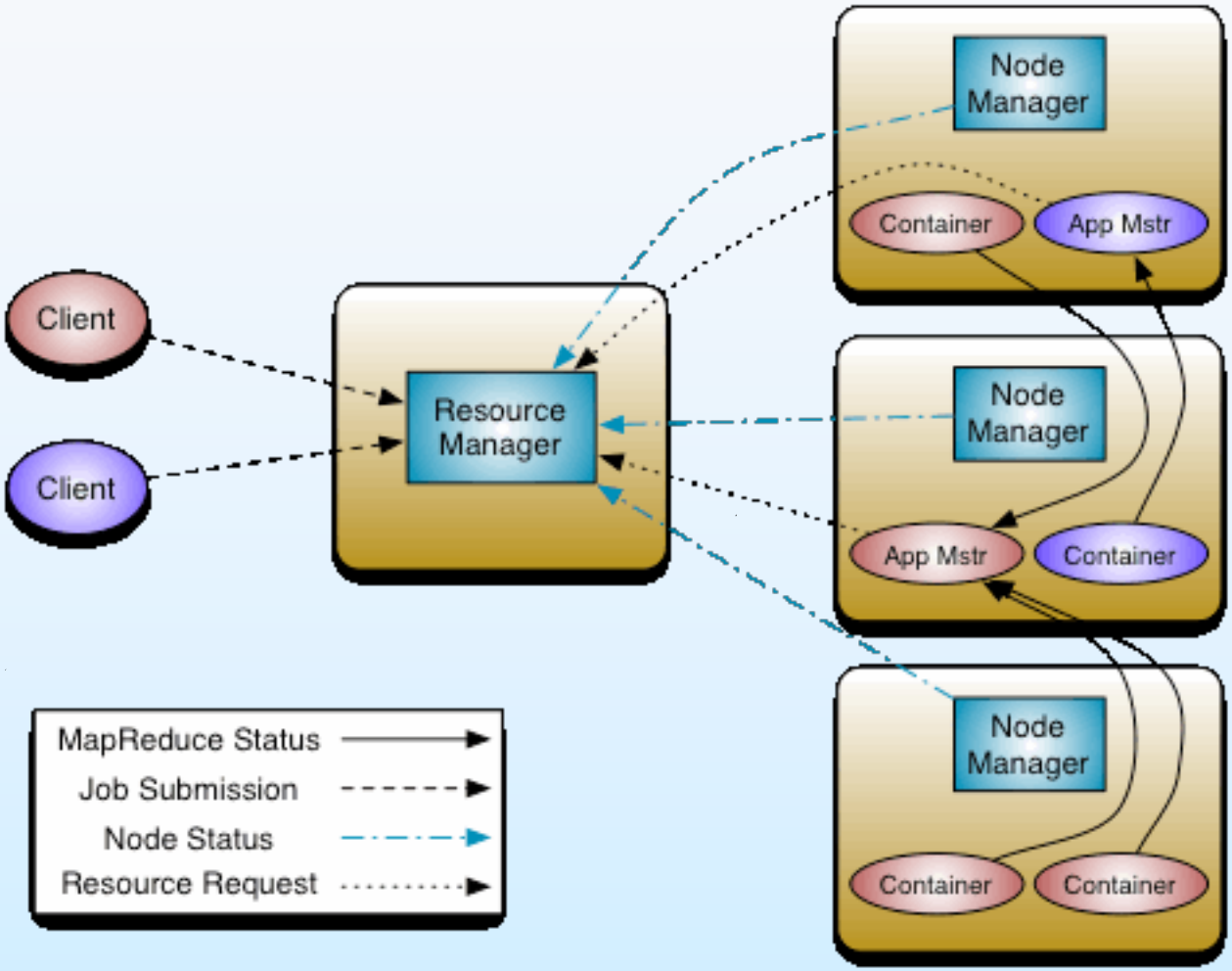
HDFS Java API

```java
public static void main(String[] args) throws IOException {
    Configuration conf = new Configuration();
    //不加的话可以读取默认的HDFS环境的配置
    conf.addResource(new Path(
            "/u/hadoop-1.1.0/conf/core-site.xml"));
    conf.addResource(new Path(
            "/u/hadoop-1.1.0/conf/hdfs-site.xml"));

    FileSystem fileSystem = FileSystem.get(conf);
    System.out.println(fileSystem.getUri());

    Path file = new Path("demo.txt");
    if (fileSystem.exists(file)) {
        System.out.println("File exists.");
    } else {
        // Writing to file
        FSDataOutputStream outStream = fileSystem.create(file);
        outStream.writeUTF("Welcome to HDFS Java API!!!");
        outStream.close();
    }

    // Reading from file
    FSDataInputStream inStream = fileSystem.open(file);
    String data = inStream.readUTF();
    System.out.println(data);
    inStream.close();

    // deleting the file. Non-recursively.
    // fileSystem.delete(file, false);

    fileSystem.close();
}
```

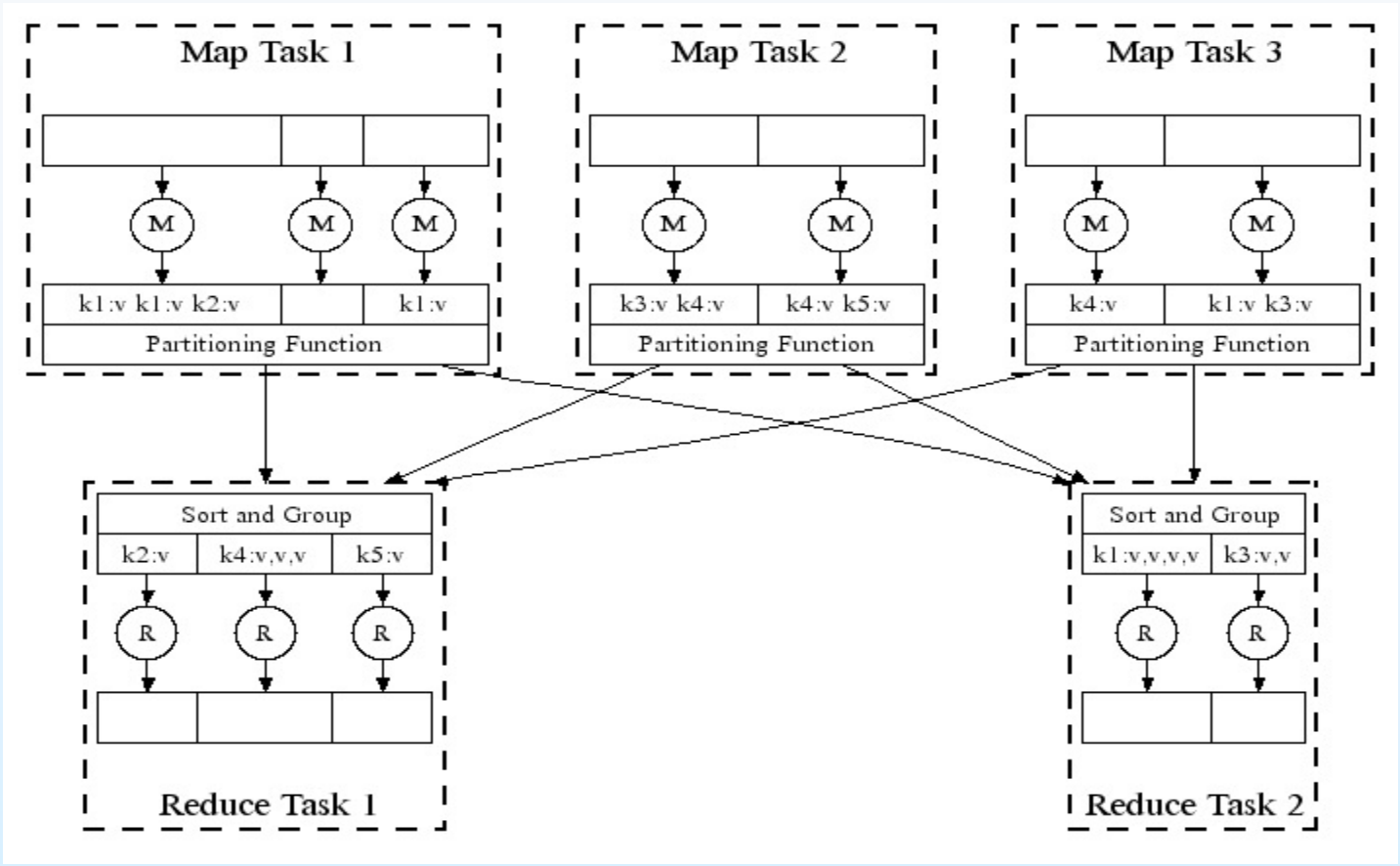
Hadoop, the Big Picture



Hadoop Yarn Achitecture

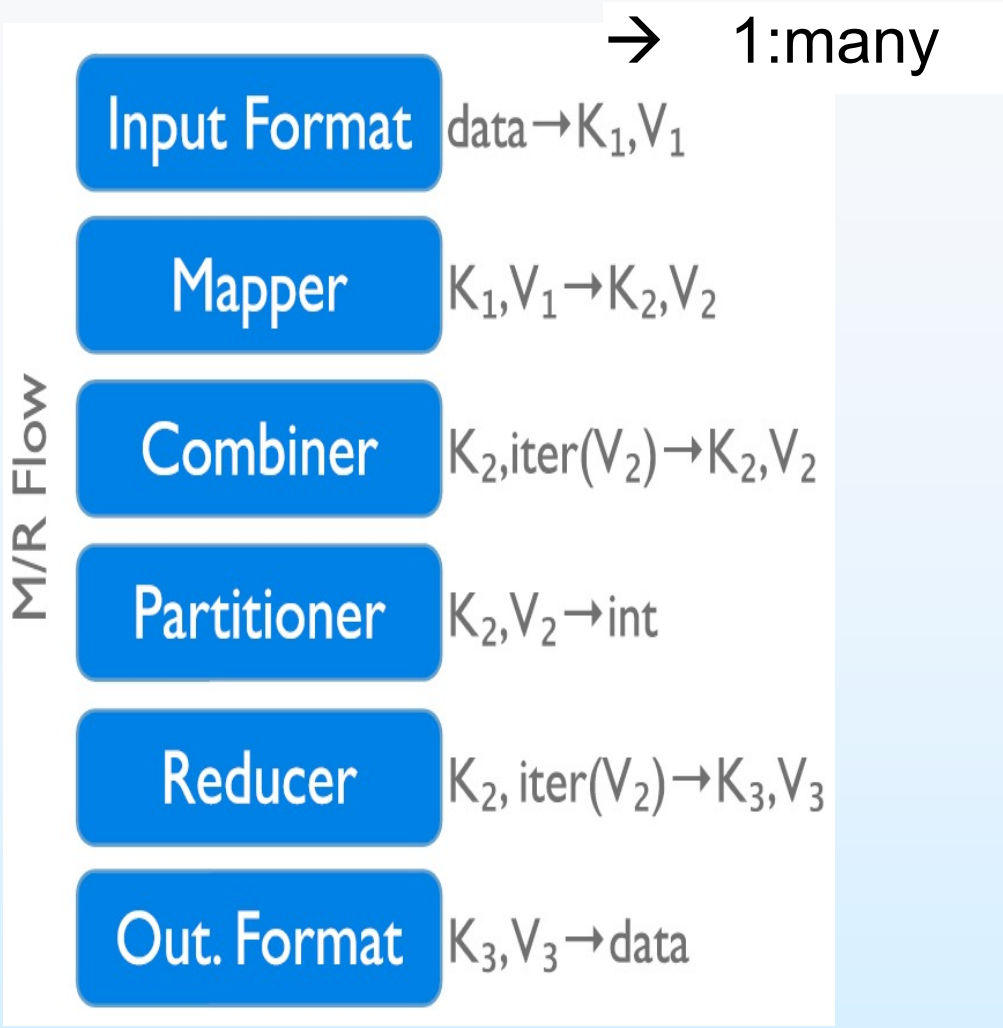


MapReduce Workflow



Functions in Hadoop

- InputFormat
- Map function
- Partitioner
- Sorting & Merging
- Combiner
- Shuffling
- Merging
- Reduce function
- OutputFormat



Lifecycle of a MapReduce Job

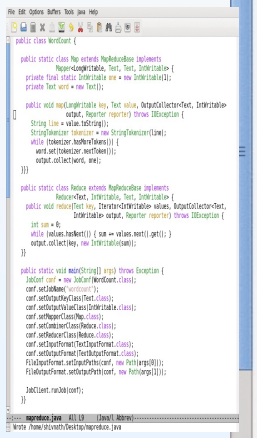
```
File  Edit  Options  Buffers  Tools  Java  Help

public class WordCount {

  public static class Map extends MapReduceBase implements
                Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable>
                    output, Reporter reporter) throws IOException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
        word.set(tokenizer.nextToken());
        output.collect(word, one);
  }}}

  public static class Reduce extends MapReduceBase implements
                Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
                       IntWritable> output, Reporter reporter) throws IOException {
      int sum = 0;
      while (values.hasNext()) { sum += values.next().get(); }
      output.collect(key, new IntWritable(sum));
  }}

  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);
    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);
    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
  }}

--:---  mapreduce.java    All L9     (Java/l Abbrev)----------
 Wrote /home/shivnath/Desktop/mapreduce.java
```

Map function

Reduce function

Run this program as a MapReduce job

Lifecycle of a MapReduce Job



```
File  Edit  Options  Buffers  Tools  Java  Help
```

```
public class WordCount {

  public static class Map extends MapReduceBase implements
                Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable>
                    output, Reporter reporter) throws IOException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
        word.set(tokenizer.nextToken());
        output.collect(word, one);
  }}}

  public static class Reduce extends MapReduceBase implements
                Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
                       IntWritable> output, Reporter reporter) throws IOException {
      int sum = 0;
      while (values.hasNext()) { sum += values.next().get(); }
      output.collect(key, new IntWritable(sum));
  }}

  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);
    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);
    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
  }}
```

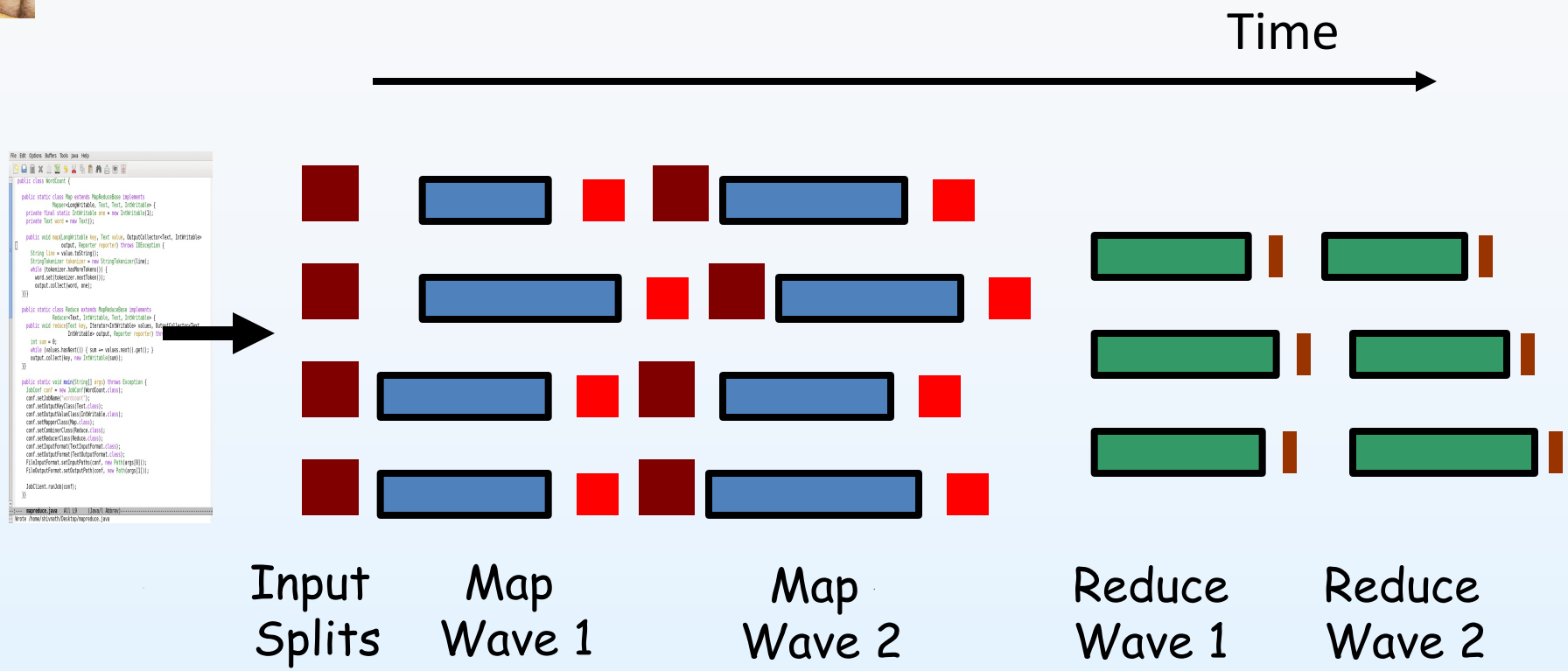
```
--:---  mapreduce.java    All L9     (Java/l Abbrev)----------
 Wrote /home/shivnath/Desktop/mapreduce.java
```

Map function

Reduce function

Run this program as a MapReduce job

Lifecycle of a MapReduce Job



How are the number of splits, number of map and reduce tasks, memory allocation to tasks, etc., determined?

Job Configuration Parameters

File Edit Options Buffers Tools SGML Help

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>

<property>
  <name>mapred.reduce.tasks</name>
  <value>1</value>
  <description>The default number of reduce tasks
               per job</description>
</property>

<property>
  <name>io.sort.factor</name>
  <value>10</value>
  <description>Number of streams to merge at once
               while sorting</description>
</property>

<property>
  <name>io.sort.record.percent</name>
  <value>0.05</value>
  <description>Percentage of io.sort.mb dedicated to
               tracking record boundaries</description>
</property>

</configuration>
```

-U:--- conf.xml All L9 (XML)----------

- 190+ parameters in Hadoop
- Set manually or defaults are used

End of Chapter 12

Problem

- You've been hired by Orange Computer to help design a new processor and Orange Pro laptop. After choosing the display, case, and other components, you are left with $460 to spend on the following components:

Item	Latency	Minimum Size	Cost
TLB	10 ns	256 entries	$0.10/entry
Main memory	180 ns	2 GB	$10/GB
Magnetic Disk	8 ms (8M ns)	300 GB	$0.10/GB

- The page size is fixed at 64 KB. Assume you want to run up to 20 applications simultaneously. Each application has an overall maximum size of 1 GB and a working set size of 256 MB. TLB entries do not have Process Identifiers. Discuss how you would divide the available funds across the various items to optimize performance.

Problem

- *We start with the disk. Since the disk is the slowest component of the system, we take the minimum size, 300 GB or $30, leaving us with $430.*

Item / Points	3 points	2 points	1 point	0 points
(a) TLB	3,800 entries	> 3,800	< 3,800	< 256
(b) Memory	5 GB	> 5 GB	< 5 GB	< 2 GB
(c) Disk	300 GB	> 300GB, if using extra money for disk instead of memory or TLB		

decrease the memory size below the requirements for the applications; a situation that will cause the system to start paging.