Chapter 8: Software Construction for Performance
8.3 I/O and Algorithm Performance
I/O与算法性能

Wang Zhongjie
rainy@hit.edu.cn

May 12, 2019

Outline

- **I/O Performance I/O性能**
  - I/O of software systems
  - Java I/O APIs
  - Java NIO APIs
  - From io to nio, better I/O performance
- **Algorithm Performance 算法性能**

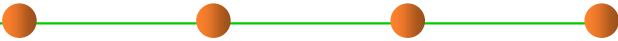8-2节了解了"空间性能"（内存管理），本节关注"时间性能"，体现在I/O和算法两方面。主要关注如何利用JDK持续演化的I/O APIs写出高效的I/O程序。

Reading

- **CMU 17-214：Oct 16**

- **Java性能权威指南：第1、2、4、5、6章**

- **Java编程思想：第13章、第18章**

- **Java Performance Tuning：第3章**
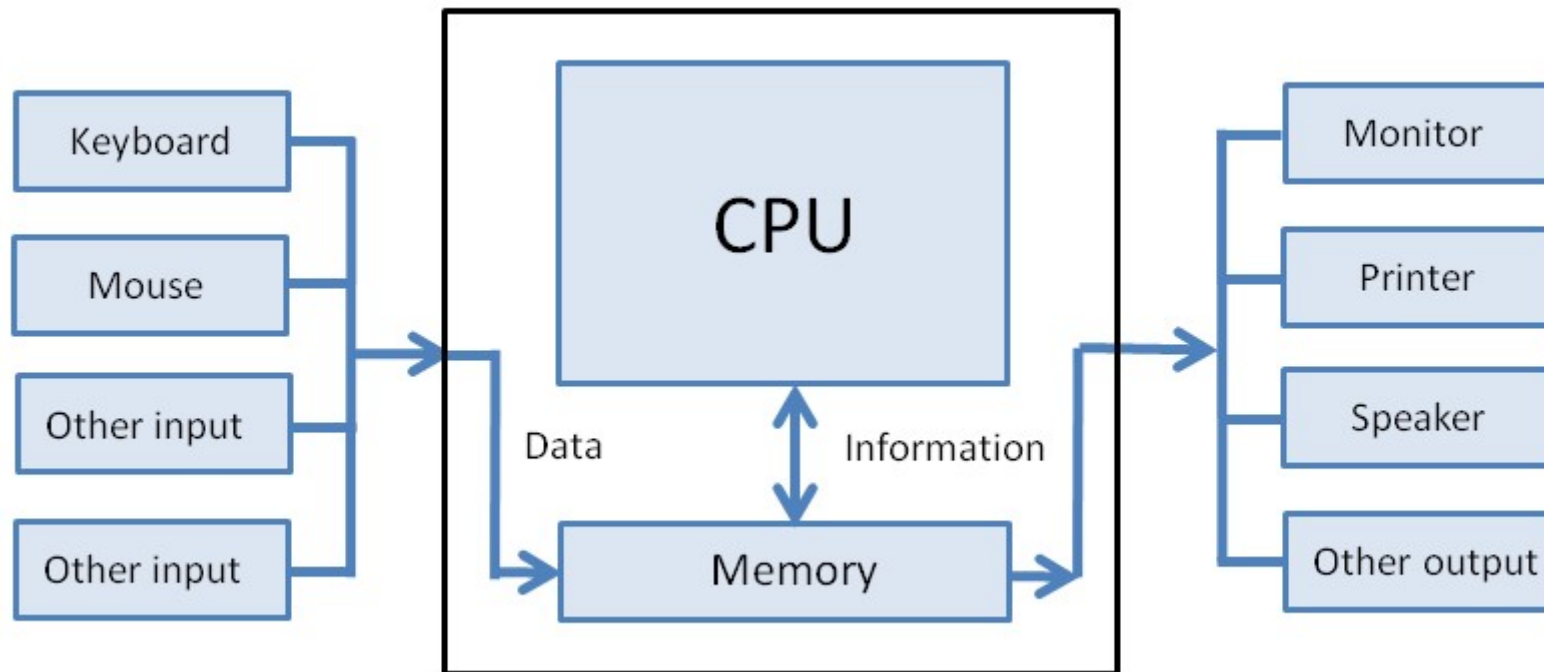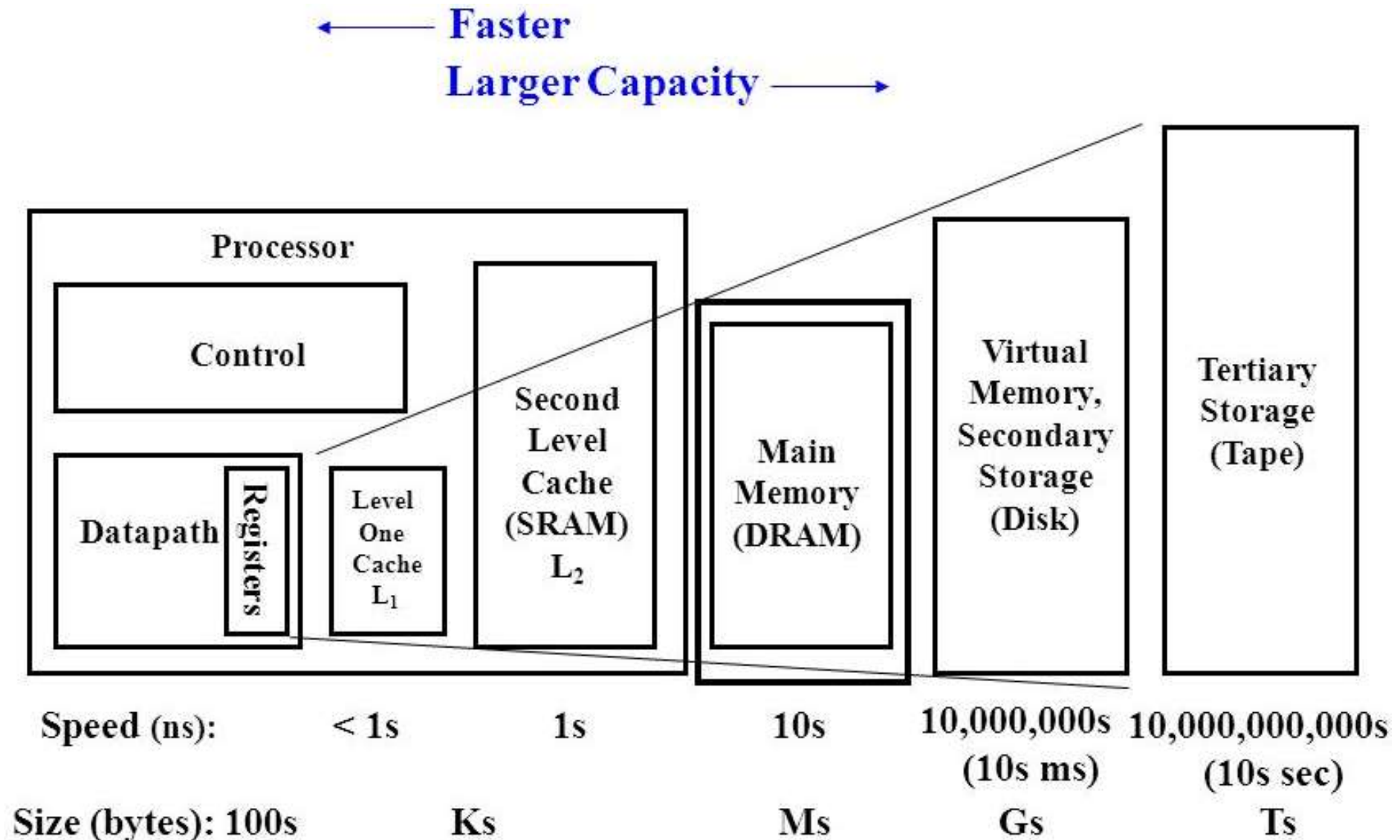
Reading is the **KEY** to learning

1 I/O of software systems

I/O of software systems

- **Inputs are the signals or data received by the system and outputs are the signals or data sent from it.**

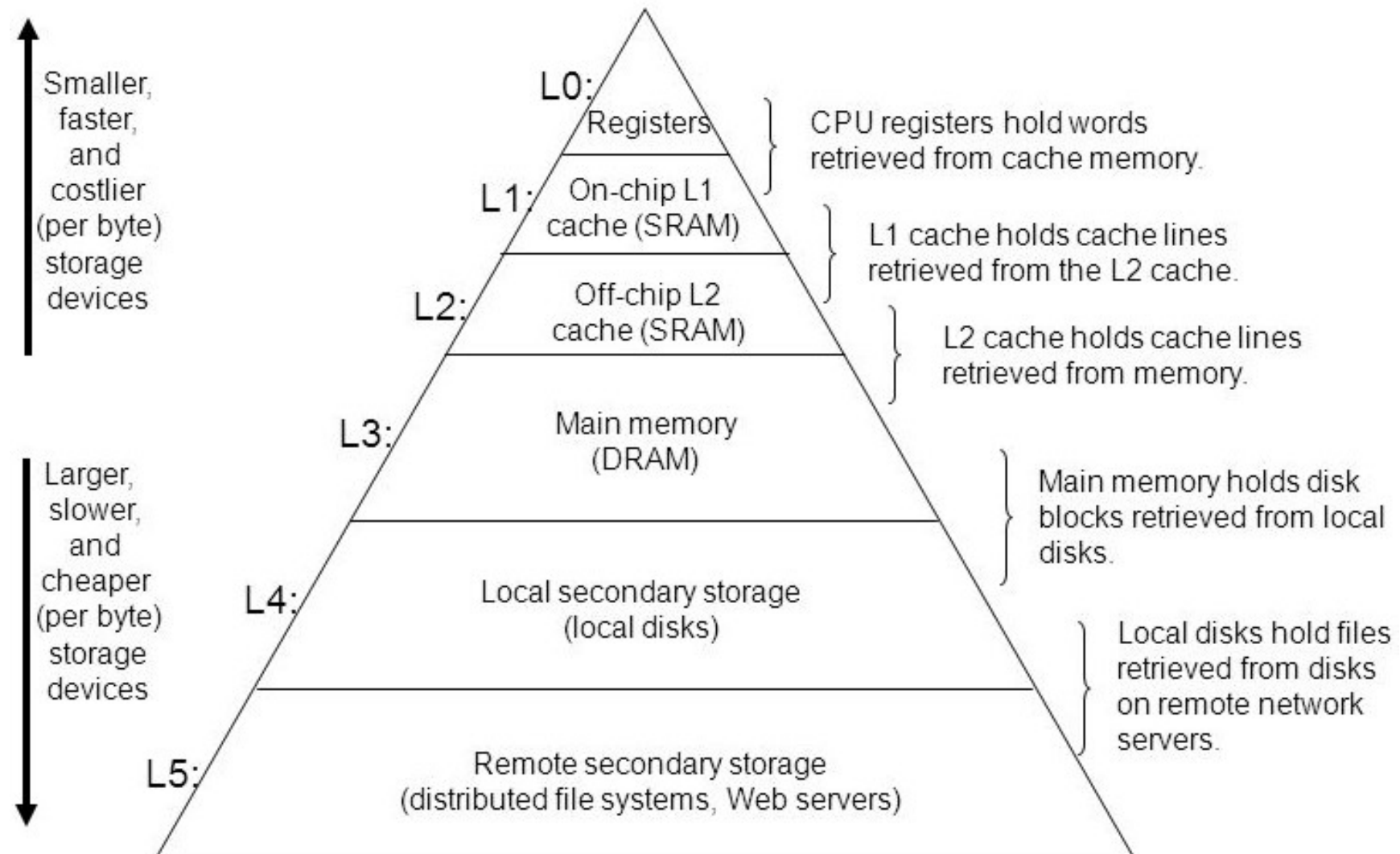| Keyboard		Monitor
Mouse	CPU	Printer
Other input		Speaker
Other input	Data Information Memory	Other output

- **I/O devices are the pieces of hardware used by a human (or other system) to communicate with a computer.**

Typical memory hierarchy

Faster

Larger Capacity →

	Processor		Second Level Cache (SRAM) L_2	Main Memory (DRAM)	Virtual Memory, Secondary Storage (Disk)	Tertiary Storage (Tape)

Processor: Control, Datapath, Registers, Level One Cache L_1

Speed (ns):	< 1s	1s	10s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks	Ms	Gs	Ts

Typical memory hierarchy

Smaller,
faster,
and
costlier
(per byte)
storage
devices

Larger,
slower,
and
cheaper
(per byte)
storage
devices

L0:
Registers

L1:
On-chip L1
cache (SRAM)

L2:
Off-chip L2
cache (SRAM)

L3:
Main memory
(DRAM)

L4:
Local secondary storage
(local disks)

L5:
Remote secondary storage
(distributed file systems, Web servers)

CPU registers hold words
retrieved from cache memory.

L1 cache holds cache lines
retrieved from the L2 cache.

L2 cache holds cache lines
retrieved from memory.

Main memory holds disk
blocks retrieved from local
disks.

Local disks hold files
retrieved from disks
on remote network
servers.

I/O of software systems

2 Java I/O APIs

`java.io` in JDK

The package `java.io` provides for system input and output through data streams, serialization and the file system.

- java.io.**RandomAccessFile** (implements java.io.Closeable, java.io.DataInput, java.io.DataOutput)
- java.io.**Reader** (implements java.io.Closeable, java.lang.Readable)
  - java.io.**BufferedReader**
    - java.io.**LineNumberReader**
  - java.io.**CharArrayReader**
  - java.io.**FilterReader**
    - java.io.**PushbackReader**
  - java.io.**InputStreamReader**
    - java.io.**FileReader**
  - java.io.**PipedReader**
  - java.io.**StringReader**
- java.io.**StreamTokenizer**
- java.lang.**Throwable** (implements java.io.Serializa
  - java.lang.**Error**
    - java.io.**IOError**
  - java.lang.**Exception**
    - java.io.**IOException**
      - java.io.**CharConversionException**
      - java.io.**EOFException**
      - java.io.**FileNotFoundException**
      - java.io.**InterruptedIOException**
      - java.io.**ObjectStreamException**
        - java.io.**InvalidClassException**
        - java.io.**InvalidObjectException**
        - java.io.**NotActiveException**
        - java.io.**NotSerializableException**
        - java.io.**OptionalDataException**
        - java.io.**StreamCorruptedException**
        - java.io.**WriteAbortedException**
      - java.io.**SyncFailedException**
      - java.io.**UnsupportedEncodingException**
      - java.io.**UTFDataFormatException**
    - java.lang.**RuntimeException**
      - java.io.**UncheckedIOException**
- java.io.**Writer** (implements java.lang.Appendable, java.io.Closeable, java.io.Flushable)
  - java.io.**BufferedWriter**
  - java.io.**CharArrayWriter**
  - java.io.**FilterWriter**
  - java.io.**OutputStreamWriter**
    - java.io.**FileWriter**
  - java.io.**PipedWriter**
  - java.io.**PrintWriter**
  - java.io.**StringWriter**

- java.lang.**Object**
  - java.io.**Console** (implements java.io.Flusha
  - java.io.**File** (implements java.lang.Compara
  - java.io.**FileDescriptor**
  - java.io.**InputStream** (implements java.io.C
    - java.io.**ByteArrayInputStream**
    - java.io.**FileInputStream**
    - java.io.**FilterInputStream**
      - java.io.**BufferedInputStream**
      - java.io.**DataInputStream** (impl
      - java.io.**LineNumberInputStrea**
      - java.io.**PushbackInputStream**
    - java.io.**ObjectInputStream** (impleme
    - java.io.**PipedInputStream**
    - java.io.**SequenceInputStream**
    - java.io.**StringBufferInputStream**
  - java.io.**ObjectInputStream.GetField**
  - java.io.**ObjectOutputStream.PutField**
  - java.io.**ObjectStreamClass** (implements ja
  - java.io.**ObjectStreamField** (implements ja
  - java.io.**OutputStream** (implements java.io
    - java.io.**ByteArrayOutputStream**
    - java.io.**FileOutputStream**
    - java.io.**FilterOutputStream**
      - java.io.**BufferedOutputStream**
      - java.io.**DataOutputStream** (im
      - java.io.**PrintStream** (implemen
    - java.io.**ObjectOutputStream** (impler
    - java.io.**PipedOutputStream**
  - java.security.**Permission** (implements java
    - java.security.**BasicPermission** (imple
      - java.io.**SerializablePermission**
    - java.io.**FilePermission** (implements j

java.io in JDK

java.lang
- Object

java.io

- ByteArrayInputStream
- FileInputStream
- FilterInputStream
  - BufferedInputStream
  - DataInputStream
  - LineNumberInputStream
  - PushbackInputStream
- ObjectInputStream
- PipedInputStream
- SequenceInputStream
- StringBufferInputStream

- InputStream
- File
- FilenameFilter
- FileFilter
- FileDescriptor
- RandomAccessFile

- DataInput
- ObjectInput
- DataOutput
- ObjectOutput

- ByteArrayOutputStream
- FileOutputStream
- FilterOutputStream
  - BufferedOutputStream
  - DataOutputStream
  - PrintStream
- ObjectOutputStream
- PipedOutputStream

- OutputStream
- ObjectStreamClass
- StreamTokenizer

- BufferedReader
  - LineNumberReader
- CharArrayReader
- FilterReader
  - PushbackReader
- InputStreamReader
  - FileReader
- PipedReader
- StringReader

- Reader

- BufferedWriter
- CharArrayWriter
- FilterWriter
- OutputStreamWriter
  - FileWriter
- PipedWriter
- PrintWriter
- StringWriter

- Writer

- Serializable
  - Externalizable
- ObjectInputValidation

KEY
- CLASS
- ABSTRACT CLASS
- DEPRECATED CLASS
- FINAL CLASS
- INTERFACE

- - - - implements
——— extends

`java.io` in JDK

	Read	**Write**

Text	Reader
	   BufferedReader
	   StringReader
	   CharArrayReader
	   FileReader
	   …
		Writer
		   BufferedWriter
		   StringWriter
		   CharArrayWriter
		   FileWriter
		   PrintWriter
		   …

Binary	InputStream
	   FileInputStream
	   ByteArrayInputStream
	   ObjectInputStream
	   PipedInputStream
	   …
		OutputStream
		   FileOutputStream
		   ByteArrayOutputStream
		   ObjectOutputStream
		   PipedOutputStream
		   …

Example 1: Reading/Writing from/to text files

```java
FileReader inputStream = null;
FileWriter outputStream = null;

try {
    inputStream = new FileReader("in.txt");
    outputStream = new FileWriter("output.txt");

    int c;
    while ((c = inputStream.read()) != -1)
        outputStream.write(c);

} finally {
    if (inputStream != null)
        inputStream.close();
    if (outputStream != null)
        outputStream.close();
}
```

Example 2: Reading/Writing from/to bin files

```java
FileInputStream in = null;
FileOutputStream out = null;

try {
    in = new FileInputStream("in.bin");
    out = new FileOutputStream("output.bin");

    int c;
    while ((c = in.read()) != -1)
        out.write(c);

} finally {
    if (in != null)
        in.close();
    if (out != null)
        out.close();
}
```


Input Stream

| I | n | | X | a | n | a | d | u | | d | i | d |

read (b)

Integer Variable

| | | d |

write (b)

| I | n | | X | a | n | a | d | u | | d |

Output Stream
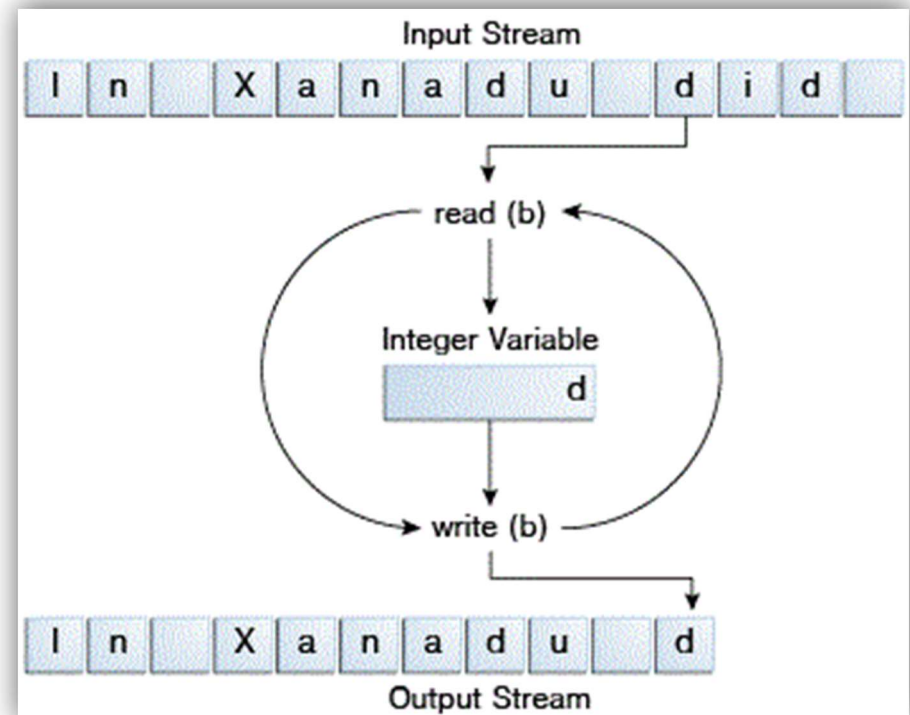
Programming for I/O is a basic skill

- **Recommend:**

  https://docs.oracle.com/javase/tutorial/essential/io/index.html

3 Java nio APIs

Speeding up I/O

- **In software systems, the reading-writing speed of I/O is slower than that of memory.**

- **I/O reading and writing on many occasions will become bottleneck of a system.**

- **Speeding up I/O has a great advantage to enhance the overall performance of the system.**

IO	NIO
Stream oriented	Buffer oriented
Blocking IO	Non blocking IO
	Selectors

Java io

An input stream

A Java IO program reads data from the input stream and/or writes data to the output stream.

Data Source

Java Program

An output stream
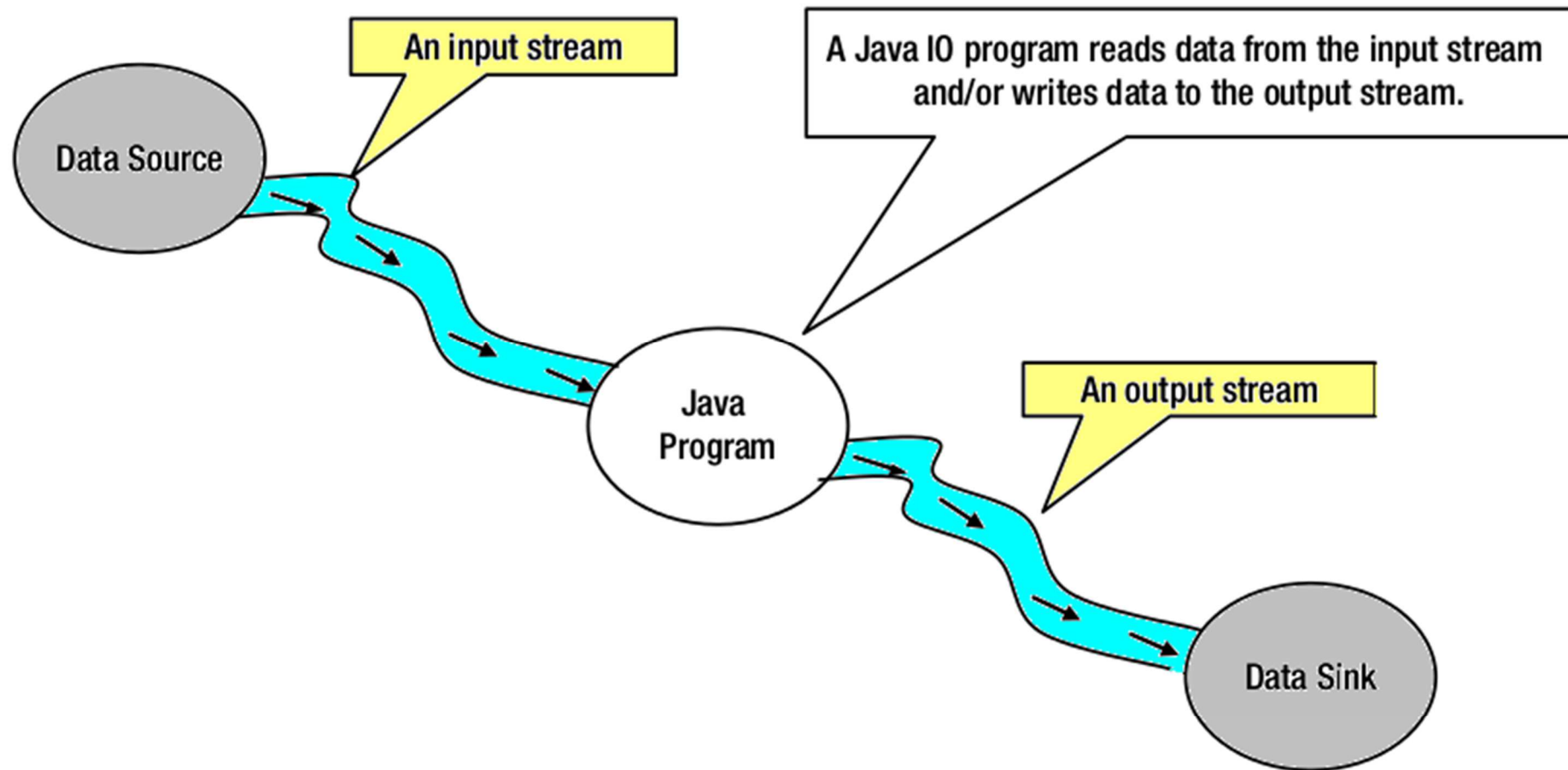
Data Sink

Flow of data using an input/output stream in a Java program

Java NIO

Channel reads data from a buffer

Program writes data into a buffer

Data Sink

Data Source

Channel

Buffer

Java Program

Buffer

Channel writes data into a buffer

Program reads data from a buffer
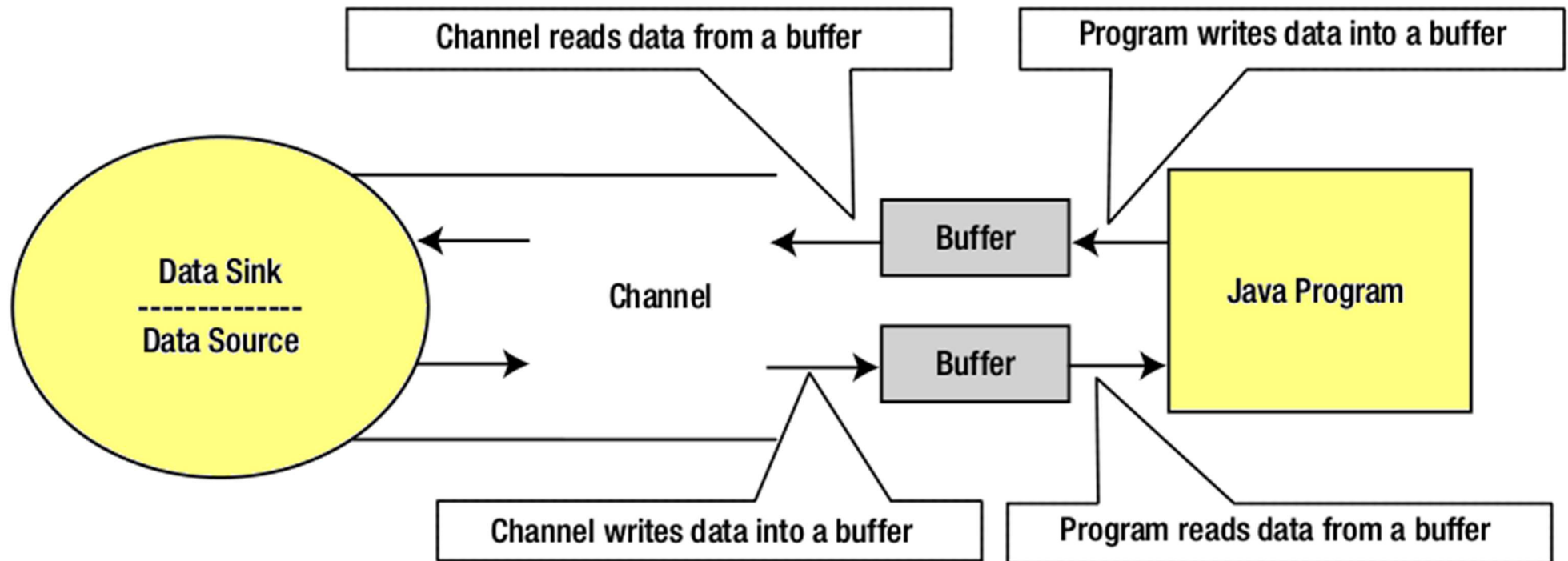
Interaction between a channel, buffers, a Java program, a data source, and a data sink

Main Differences Between Java NIO and IO

- **Stream Oriented vs. Buffer Oriented**

  - Java IO being stream oriented means that you read one or more bytes at a time, from a stream. What you do with the read bytes is up to you. They are not cached anywhere. You cannot move forth and back in the data in a stream.

  - Java NIO's buffer oriented: Data is read into a buffer from which it is later processed. You can move forth and back in the buffer as you need to. This gives you a bit more flexibility during processing.

Main Differences Between Java NIO and IO

- **Blocking vs. Non-blocking IO (sync vs. async)**
  - Java IO's various streams are blocking: when a thread invokes a `read()` or `write()`, that thread is blocked until there is some data to read, or the data is fully written. The thread can do nothing else in the meantime.
  - The same is true for non-blocking writing.
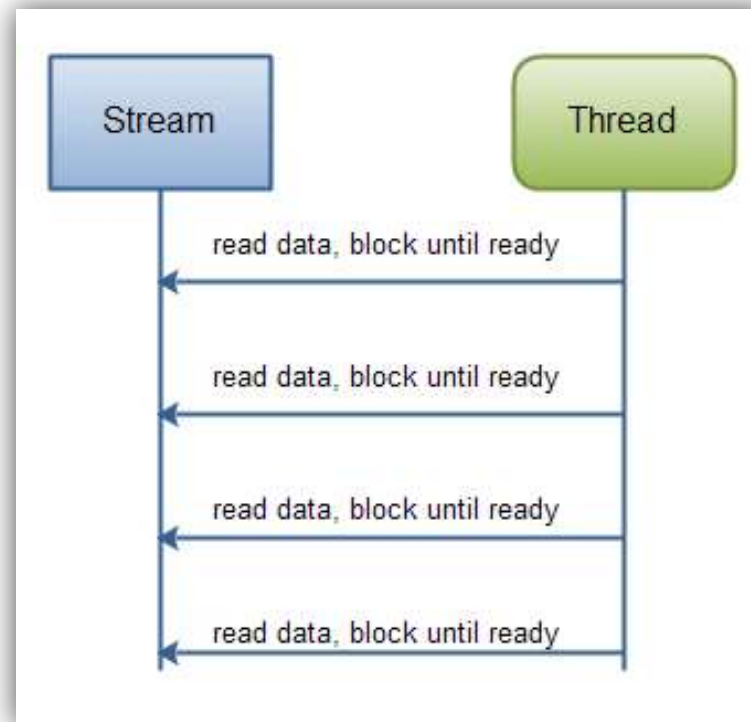
Main Differences Between Java NIO and IO

- **Blocking vs. Non-blocking IO (sync vs. async)**
  - Java NIO's non-blocking mode enables a thread to request reading data from a channel, and only get what is currently available, or nothing at all, if no data is currently available. Rather than remain blocked until data becomes available for reading, the thread can go on with something else.
  - What threads spend their idle time on when not blocked in IO calls, is usually performing IO on other channels in the meantime. That is, a single thread can now manage multiple channels of input and output.
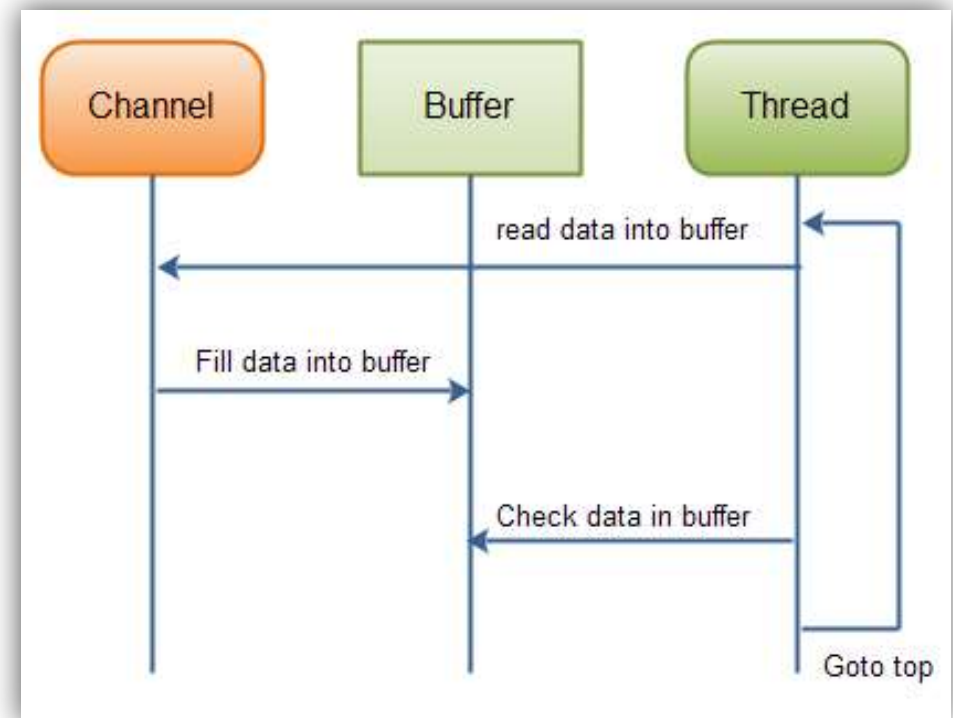
Main Differences Between Java NIO and IO

- **Selectors**
  - Java NIO's selectors allow a single thread to monitor multiple channels of input.
  - You can register multiple channels with a selector, then use a single thread to "select" the channels that have input available for processing, or select the channels that are ready for writing.
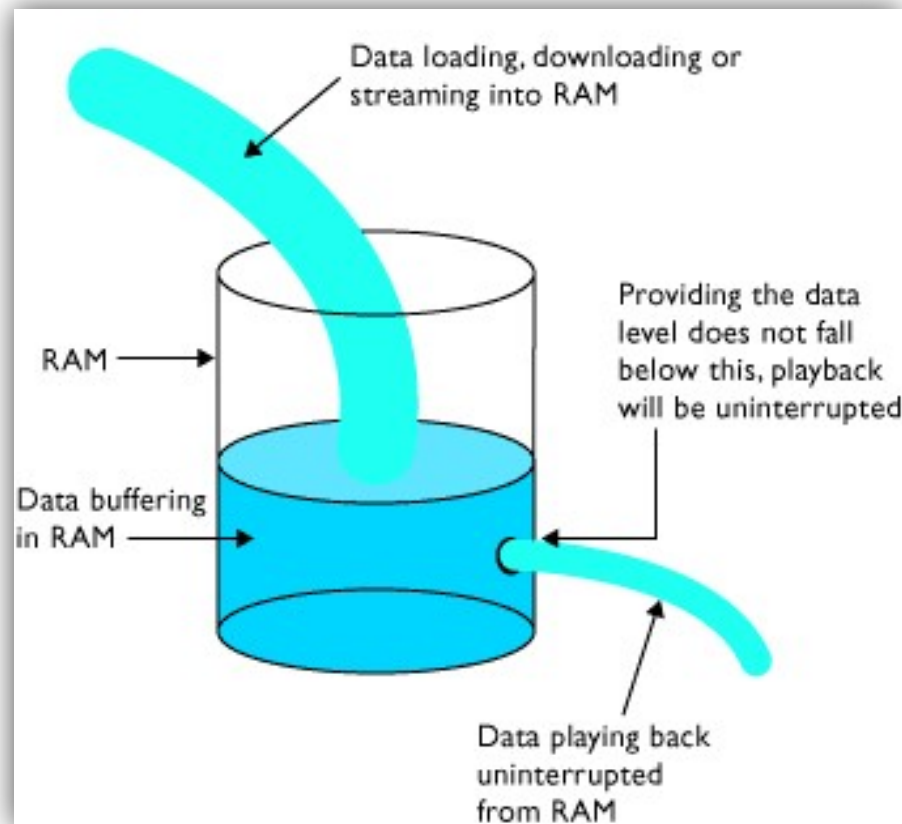 - This selector mechanism makes it easy for a single thread to manage multiple channels.
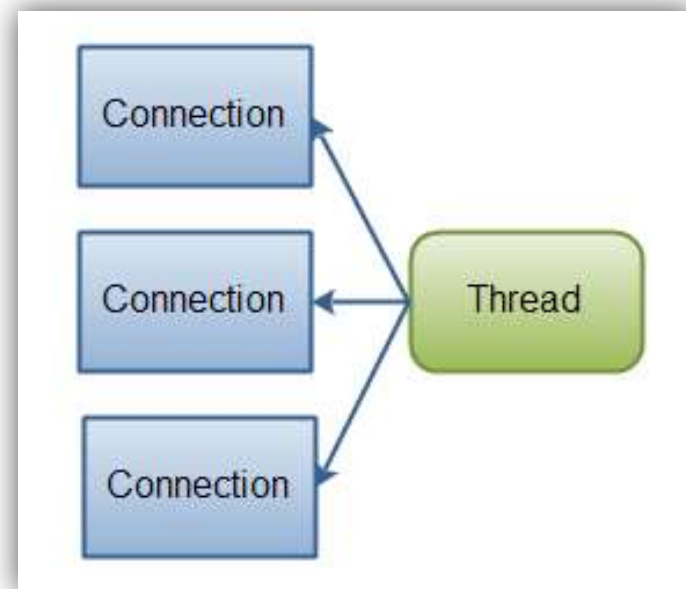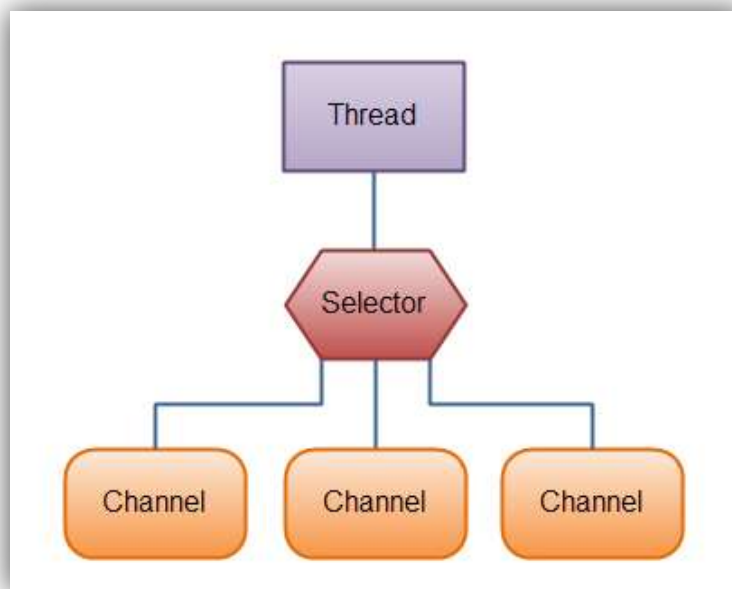
Buffer

Buffer

- **Buffer is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another.**

  – Typically, the data is stored in a buffer as it is retrieved from an input device (such as a microphone) or just before it is sent to an output device (such as speakers).

- **Buffers can be implemented:**

  – In a fixed memory location in hardware

  – By using a virtual data buffer in software, pointing at a location in the physical memory.

  – In all cases, the data stored in a data buffer are stored on a physical storage medium.

- **A majority of buffers are implemented in software, which typically use the faster RAM to store temporary data, due to the much faster access time compared with hard disk drives.**
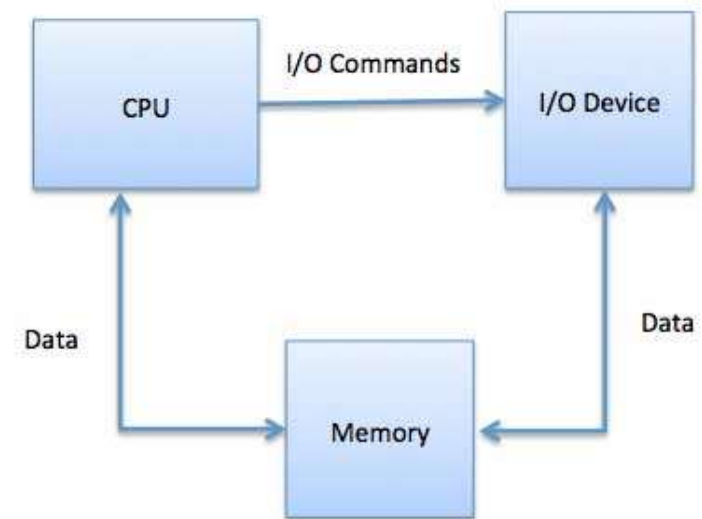
Buffer

- **Buffers are typically used when:**
  - There is a difference between the rate at which data is received and the rate at which it can be processed
  - These rates are variable, for example in a printer spooler or in online video streaming.

Data loading, downloading or streaming into RAM

RAM

Providing the data level does not fall below this, playback will be uninterrupted

Data buffering in RAM

Data playing back uninterrupted from RAM

Using Buffer to speed up I/O

```
Writer writer = new FileWriter(new File("file.txt"));
long begin =  System.currentTimeMillis();
for (int i=0; i< CIRCLE; i++){
    writer.write(i);
}

Writer.close();
System.out.println("testFileWriter spend:" +
        (System.currentTimeNillis() - begin));
```

Set CIRCLE to 100,000

Running time of the original program is 63ms;

Running time of the program with buffer is 32ms;

```
Writer writer = new BufferedWriter(
                new FileWriter(new File("file.txt")));
```

这是什么设计模式？

NIO: New I/O

- **Prior to the J2SE 1.4 release of Java, I/O had become a bottleneck.**

- **The old `java.io` stream classes had too many software layers to be fast:**

  - The specification implied much copying of small chunks of data;

  - There was no way to multiplex data from multiple sources without incurring thread context switches;

  - There was no way to exploit modern OS tricks for high performance I/O, like memory mapped files.

`java.nio.buffer` hierarchy

Hierarchy For Package java.nio

Package Hierarchies:
All Packages

Class Hierarchy

- java.lang.**Object**
  - java.nio.**Buffer**
    - java.nio.**ByteBuffer** (implements java.lang.Comparable<T>)
      - java.nio.**MappedByteBuffer**
    - java.nio.**CharBuffer** (implements java.lang.Appendable, java.lang.CharSequence
    - java.nio.**DoubleBuffer** (implements java.lang.Comparable<T>)
    - java.nio.**FloatBuffer** (implements java.lang.Comparable<T>)
    - java.nio.**IntBuffer** (implements java.lang.Comparable<T>)
    - java.nio.**LongBuffer** (implements java.lang.Comparable<T>)
    - java.nio.**ShortBuffer** (implements java.lang.Comparable<T>)
  - java.nio.**ByteOrder**
  - java.lang.**Throwable** (implements java.io.Serializable)
    - java.lang.**Exception**
      - java.lang.**RuntimeException**
        - java.nio.**BufferOverflowException**
        - java.nio.**BufferUnderflowException**
        - java.lang.**IllegalStateException**
          - java.nio.**InvalidMarkException**
        - java.lang.**UnsupportedOperationException**
          - java.nio.**ReadOnlyBufferException**

Traditional approaches for files read/write

```
FileInputStream inputStream = new FileInputStream("foo.txt");
try {
    String everything = IOUtils.toString(inputStream);
} finally {
    inputStream.close();
}
```

```
Scanner in = new Scanner(new FileReader("filename.txt"));
StringBuilder sb = new StringBuilder();
while(in.hasNext()) {
    sb.append(in.next());
}
in.close();
outString = sb.toString();
```

Using BufferedReader/Writer for files read/write

```
BufferedReader br = new BufferedReader(new FileReader("file.txt"));
try {
    StringBuilder sb = new StringBuilder();

    St try(BufferedReader br = new BufferedReader(
    wh                           new FileReader("file.txt"))) {

            StringBuilder sb = new StringBuilder();
            String line = br.readLine();

    }       while (line != null) {
    St          sb.append(line);
} fina          sb.append(System.lineSeparator());
    br          line = br.readLine();
}           }

            String everything = sb.toString();

        }
```

NIO: New I/O

- **A hierarchy of dedicated buffer classes that allow data to be moved from the JVM to the OS with minimal memory-to-memory copying, and without expensive overheads like switching byte order; effectively buffer classes give Java a "window" on system memory.**

- **A unified family of channel classes that allow data to be fed directly from buffers to files and sockets, without going through the intermediaries of the old stream classes.**

- **A family of classes to directly implement selection (AKA readiness testing, AKA multiplexing) over a set of channels.**

- **NIO also provides file locking for the first time in Java.**

Buffers in NIO

- **A Buffer object is a container for a fixed amount of data.**

- **It behaves something like a `byte[]` array, but is encapsulated in such a way that the internal storage can be a block of system memory.**

  – Thus adding data to, or extracting it from, a buffer can be a very direct way of getting information between a Java program and the underlying operating system.

  – All modern OS's provide virtual memory systems that allow memory space to be mapped to files, so this also enables a very direct and high-performance route to the file system.

  – The data in a buffer can also be efficiently read from, or written to, a socket or pipe, enabling high performance communication.

- **The buffer APIs allow you to read or write from a specific location in the buffer directly; they also allow relative reads and writes, similar to sequential file access.**

The `ByteBuffer` Class

- **The most important buffer class in practice is the `ByteBuffer` class. This represents a fixed-size vector of primitive bytes.**

- **Six categories of operations upon byte buffers**

  - Absolute / relative `get` and `put` methods that read and write single bytes;

  - Relative bulk `get` methods that transfer contiguous sequences of bytes from this buffer into an `array`;

  - Relative bulk `put` methods that transfer contiguous sequences of bytes from a `byte` array or some other `byte` buffer into this buffer;

  - Absolute and relative `get` and `put` methods that read and write values of other primitive types, translating them to and from sequences of bytes in a particular byte order;

  - Methods for creating `view buffers`, which allow a byte buffer to be viewed as a buffer containing values of some other primitive type;

  - Methods for `compacting`, `duplicating`, and `slicing` a byte buffer.

https://docs.oracle.com/javase/7/docs/api/java/nio/ByteBuffer.html

Channel

Channels

- **A channel is a new abstraction in `java.nio`.**

  - In the package `java.nio.channels`.

- **Channels are a high-level version of the file-descriptors familiar from POSIX-compliant operating systems.**

  - So a channel is a handle for performing I/O operations and various control operations on an open file or socket.

- **For familiar with conventional Java I/O, `java.nio` associates a channel with any `RandomAccessFile`, `FileInputStream`, `FileOutputStream`, `Socket`, `ServerSocket` or `DatagramSocket` objects.**

  - The channel becomes a peer to the conventional Java handle objects; the conventional objects still exist, and in general retain their role—the channel just provides extra NIO-specific functionality.

- **NIO buffer objects can written to or read from channels directly.**

java.nio.channel hierarchy

- java.lang.**Object**
  - java.nio.channels.spi.**AbstractInterruptibleChannel** (implements java.nio.channels.Channel, java.nio.channels.InterruptibleChannel)
    - java.nio.channels.**FileChannel** (implements java.nio.channels.GatheringByteChannel, java.nio.channels.ScatteringByteChannel, java.nio.channels.SeekableByteChannel)
    - java.nio.channels.**SelectableChannel** (implements java.nio.channels.Channel)
      - java.nio.channels.spi.**AbstractSelectableChannel**
        - java.nio.channels.**DatagramChannel** (implements java.nio.channels.ByteChannel, java.nio.channels.GatheringByteChannel, java.nio.channels.MulticastChannel, java.nio.channels.ScatteringByteChannel)
        - java.nio.channels.**Pipe.SinkChannel** (implements java.nio.channels.GatheringByteChannel, java.nio.channels.WritableByteChannel)
        - java.nio.channels.**Pipe.SourceChannel** (implements java.nio.channels.ReadableByteChannel, java.nio.channels.ScatteringByteChannel)
        - java.nio.channels.**ServerSocketChannel** (implements java.nio.channels.NetworkChannel)
        - java.nio.channels.**SocketChannel** (implements java.nio.channels.ByteChannel, java.nio.channels.GatheringByteChannel, java.nio.channels
        - java.nio.channels
  - a.nio.channels.**AsynchronousChan**
  - a.nio.channels.**AsynchronousFileC**
  - a.nio.channels.AsynchronousChann
  - a.nio.channels.**AsynchronousServ**
  - a.nio.channels.AsynchronousChann
  - a.nio.channels.**AsynchronousSock**
  - java.nio.channels.AsynchronousByteCh
  - java.nio.channels.**Channels**
  - java.nio.channels.**FileChannel.MapMo**
  - java.nio.channels.**FileLock** (implement
  - java.nio.channels.**MembershipKey**
  - java.nio.channels.**Pipe**
  - java.nio.channels.**SelectionKey**
  - java.nio.channels.**Selector** (implements
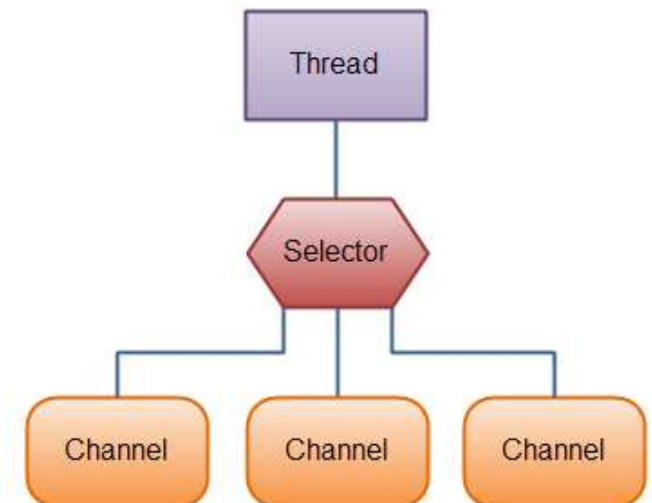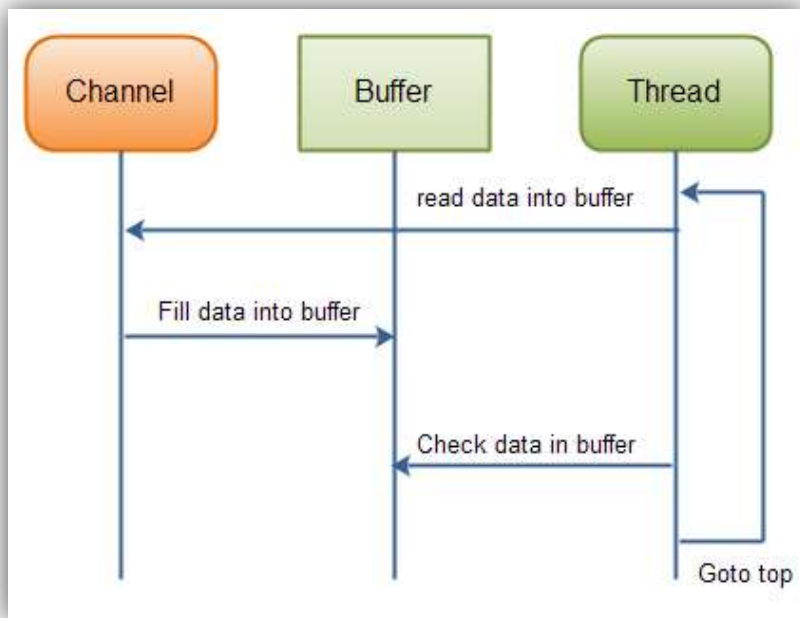
Opening Channels

- **Socket channel classes have static factory methods** `open()`
  - `SocketChannel sc = SocketChannel.open() ;`
  - `Sc.connect(new InetSocketAddress(hostname, portnumber)) ;`

- **File channels cannot be created directly; first use conventional Java I/O mechanisms to create a** `FileInputStream, FileOutputStream,` **or** `RandomAccessFile,` **then apply the new** `getChannel()` **method to get an associated NIO channel:**
  - `RandomAccessFile raf = new RandomAccessFile(filename, "r") ;`
  - `FileChannel fc = raf.getChannel() ;`

Using Channels

- **Any channel that implements the `ByteChannel` interface—i.e. all channels except `ServerSocketChannel`—provide a `read()` and a `write()` instance method:**

  - `int read(ByteBuffer dst)`

  - `int write(ByteBuffer src)`

  - These may look reminiscent of the read() and write() system calls in UNIX:

Memory-Mapped Files

- **In modern OS one can exploit the virtual memory system to map a physical file into a region of program memory.**

  - Once the file is mapped, accesses to the file can be extremely fast: one doesn't have to go through `read()` and `write()` system calls.

- **This low-level optimization in Java: `FileChannel`**

  - `MappedByteBuffer map(MapMode mode, long position, long size)`

  - mode should be one of `MapMode.READ_ONLY`, `MapMode.READ_WRITE`, `MapMode.PRIVATE`.

  - The returned `MappedByteBuffer` can be used wherever an ordinary `ByteBuffer` can.

An example

```
FileInputStream fis = new FileInputStream("readfile.txt");
FileChannel readChannel = fis.getChannel();

ByteBuffer buffer = ByteBuffer.allocate(500);

FileOutputStream fos = new FileOutputStream("filewrite.txt");
FileChannel writeChannel = fos.getChannel();

while (true) {
    buffer.clear();
    int mark = readChannel.read(buffer);
    if (mark==-1)
        break;
    buffer.flip();
    writeChannel.write(buffer);
}
```
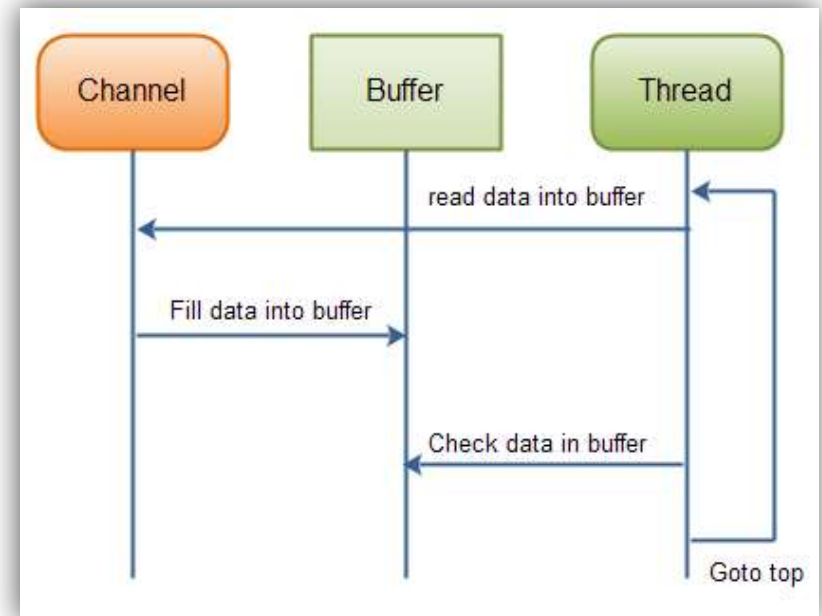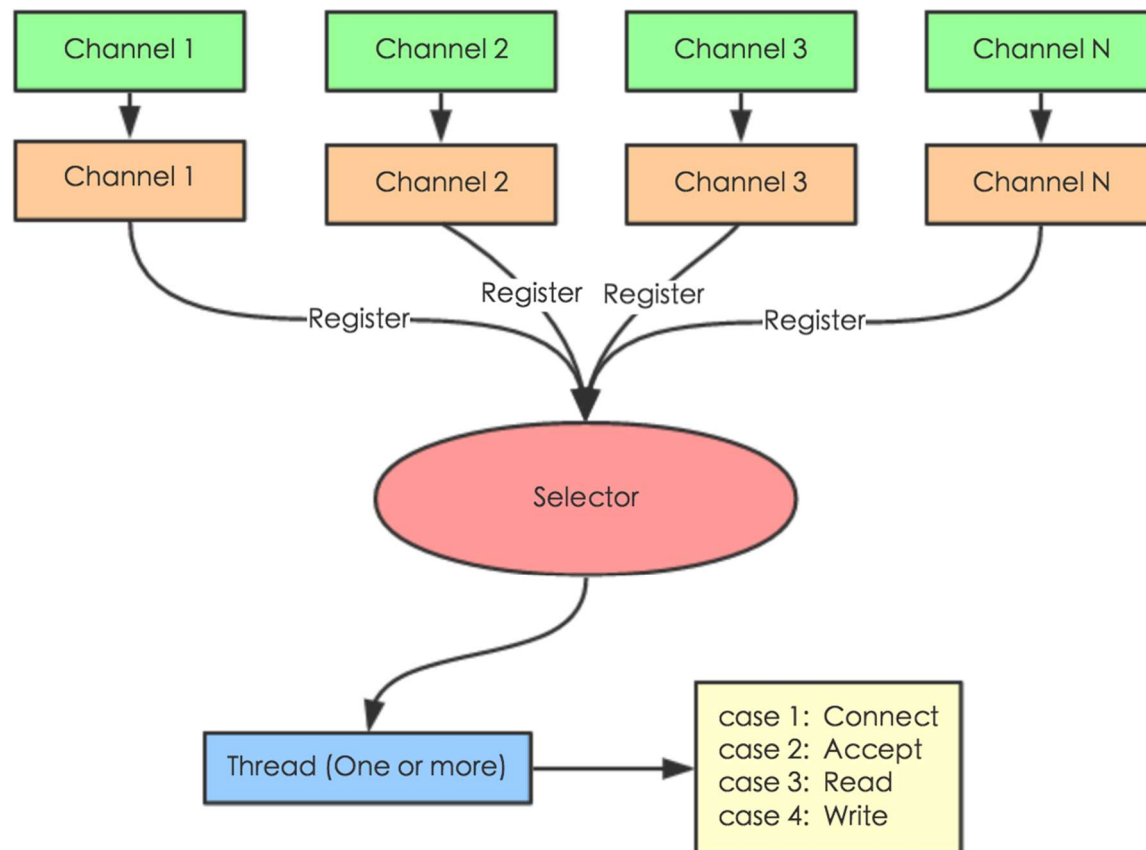
NIO selector model
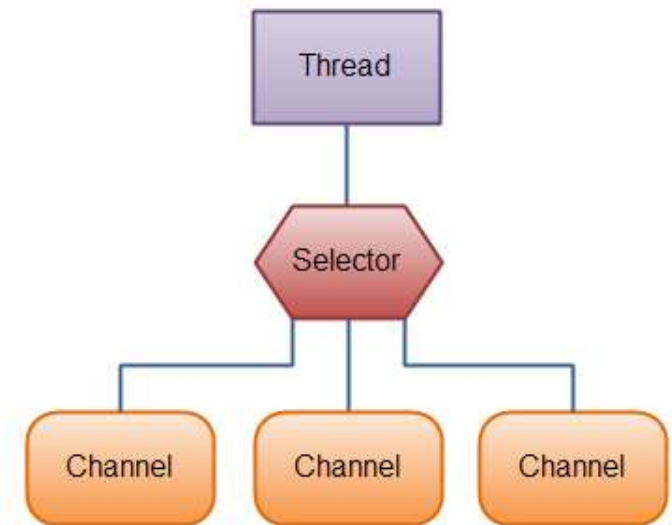
4 From io to nio, better I/O performance

A brief, sad history of I/O in Java

Release, Year	Changes
JDK 1.0, 1996	`java.io.InputStream/OutputStream` – byte-based
JDK 1.1, 1997	`java.io.Reader/Writer` – char-based wrappers
J2SE 1.4, 2002	`java.nio.Channel/Buffer` – "Flexible" + select/poll, mmap
J2SE 5.0, 2004	`java.util.Scanner`, `String.printf/format` – Formatted
Java 7, 2011	`java.nio.file Path/Files` – file systems `java.nio.AsynchronousFileChannel` - *Real* async I/O
Java 8, 2014	`Files.lines` – lambda/stream integration
3d party, 2014	`com.squareup.okio.Buffer` – "Modern"

1: Stream

```java
/**
 * Reads all lines from a text file and prints them.
 * Uses Java 1.0-era (circa 1996) Streams to read the file.
 */
public class StreamCat {
    public static void main(String[] args) throws IOException {
        DataInputStream dis = new DataInputStream(
            new FileInputStream(args[0]));
         // Don't do this! DataInputStream.readLine is DEPRECATED!
        String line;
        while ((line = dis.readLine()) != null)
            System.out.println(line);
    }
}
```

2: Reader

```java
/**
 * Reads all lines from a text file and prints them.
 * Uses Java 1.1-era (circa 1997) Streams to read the file.
 */
public class ReaderCat {
    public static void main(String[] args) throws IOException {
        try (BufferedReader rd = new BufferedReader(
                new FileReader(args[0]))) {
            String line;
            while ((line = rd.readLine()) != null) {
                System.out.println(line);
                // you could also wrap System.out in a PrintWriter
            }
        }
    }
}
```

3: Nio

```java
/**
 * Reads all lines from a text file and prints them.
 * Uses nio FileChannel and ByteBuffer.
 */
public class NioCat {
    public static void main(String[] args) throws IOException {
        ByteBuffer buf = ByteBuffer.allocate(512);
        try (FileChannel ch = FileChannel.open(Paths.get(args[0]),
                    StandardOpenOption.READ)) {
            int n;
            while ((n = ch.read(buf)) > -1) {
                System.out.print(new String(buf.array(), 0, n));
                buf.clear();
            }
        }
    }
}
```

4: Scanner

```java
/**
 * Reads all lines from a text file and prints them
 * Uses Java 5 scanner.
 */
public class ScannerCat {
    public static void main(String[] args) throws IOException {
        try (Scanner s = new Scanner(new File(args[0]))) {
            while (s.hasNextLine())
                System.out.println(s.nextLine());
        }
    }
}
```

5: Lines

```
/**
* Reads all lines from a text file and prints them. Uses Files,
* Java 8-era Stream API (not IO Streams!) and method references.
*/
public class LinesCat {
    public static void main(String[] args) throws IOException {
        Files.lines(Paths.get(args[0])).forEach(System.out::println);
    }
}
```

Another recommendation

- **An interesting class (`java.nio.file.Files`)** which consists exclusively of static methods that operate on files, directories, or other types of files.

- In most cases, the methods defined here will delegate to the associated file system provider to perform the file operations.

  - https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html

```java
Path sourceFile = Paths.get("file1.txt");
Path targetFile = Paths.get("file2.txt");

try {
    Files.copy(sourceFile, targetFile,
        StandardCopyOption.REPLACE_EXISTING);
} catch (IOException ex) {
    System.err.format("I/O Error when copying file");
}
```
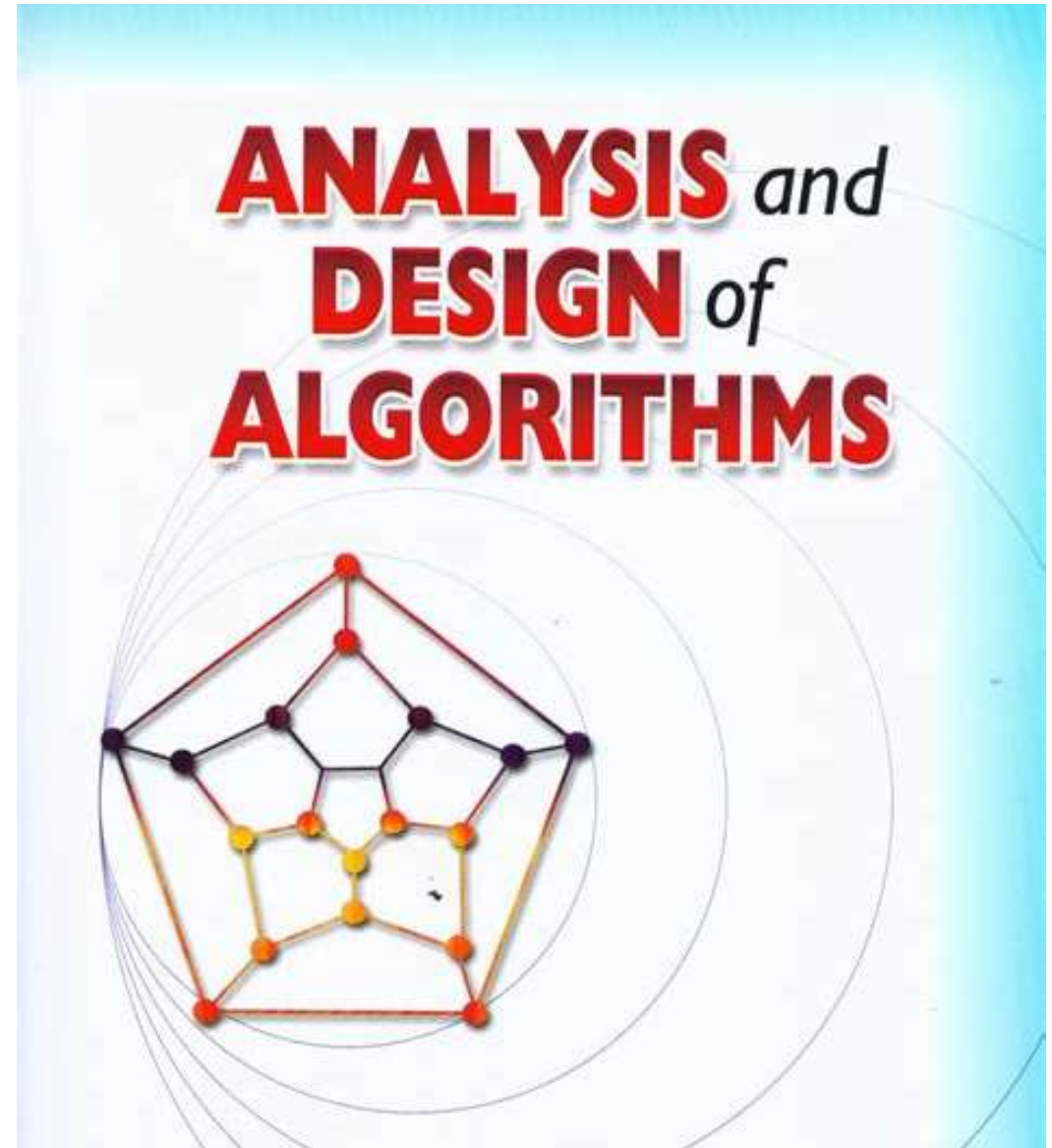
5 Algorithm Performance

Algorithm Performance

- **Algorithm Performance is out of scope of this course.**

- **Train your algorithm design and optimization skills in "Algorithm" course.**

- **And then apply your skills in software construction.**

The end

May 12, 2019