

# МРЕЖОВО ПРОГРАМИРАНЕ С JAVA

11.12.2018

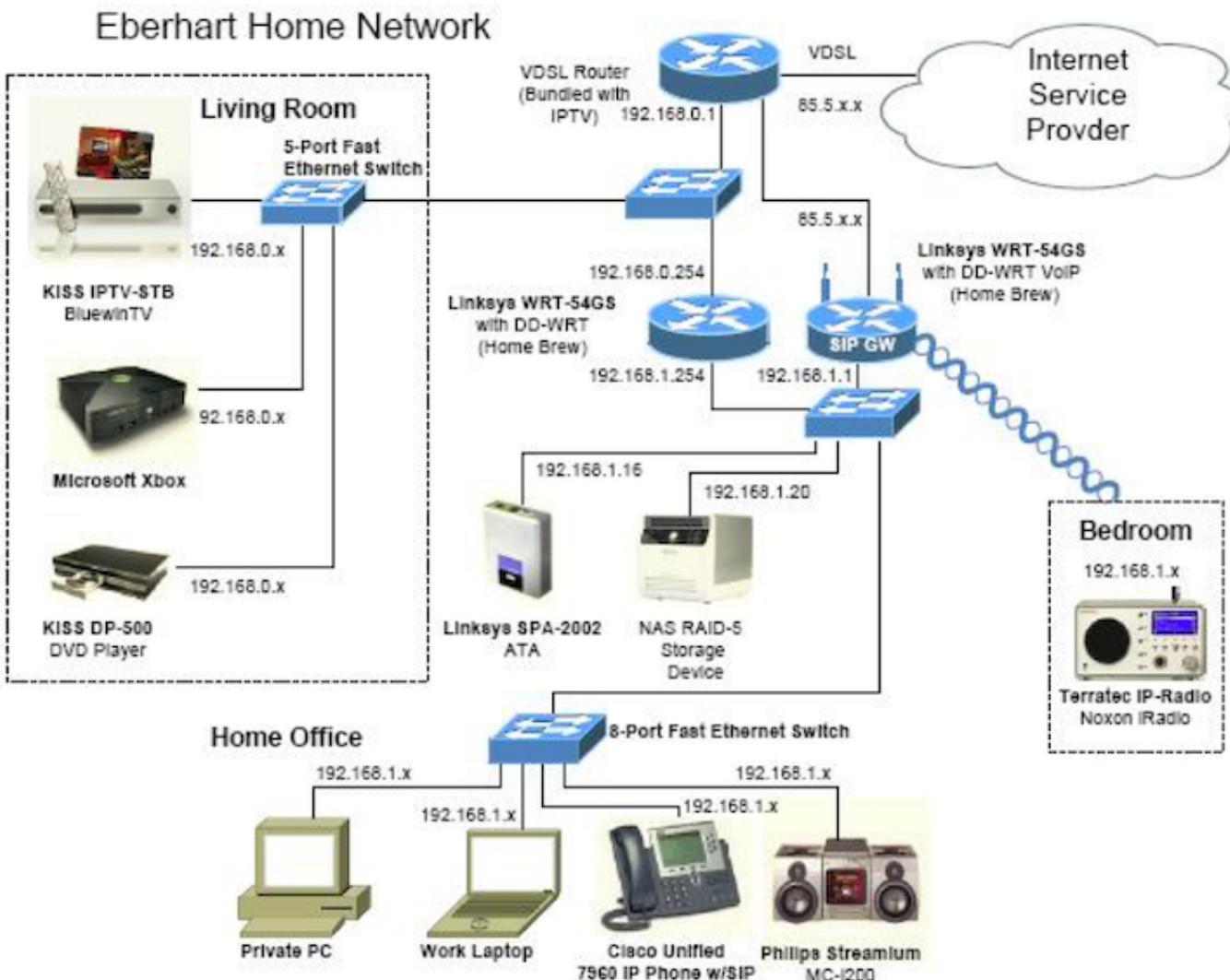


# СЪДЪРЖАНИЕ

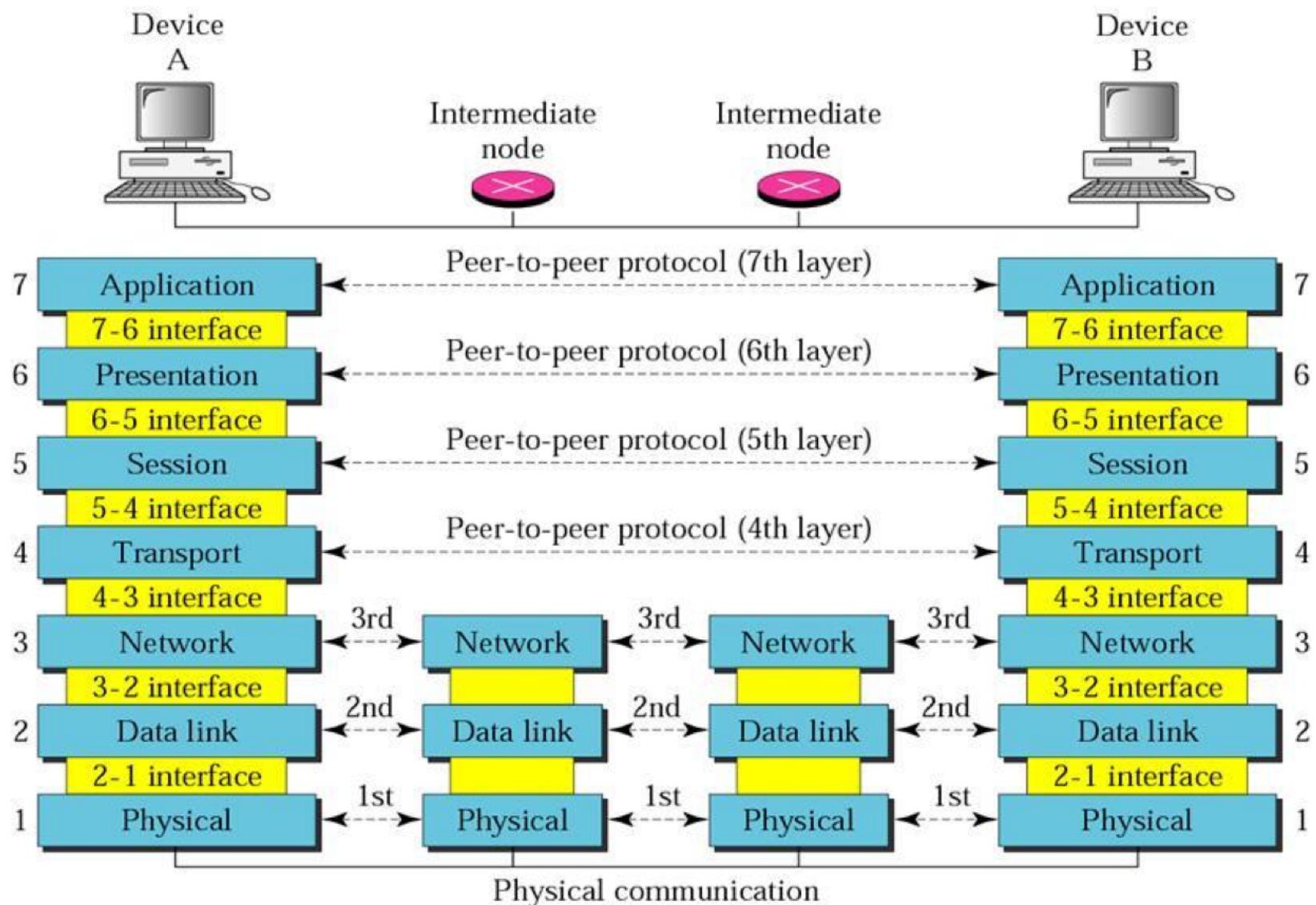
- Мрежови основи
- Моделът клиент-сървър
- Мрежова комуникация в Java



# МРЕЖАТА



# ПРЕДАВАНЕ НА ДАННИ

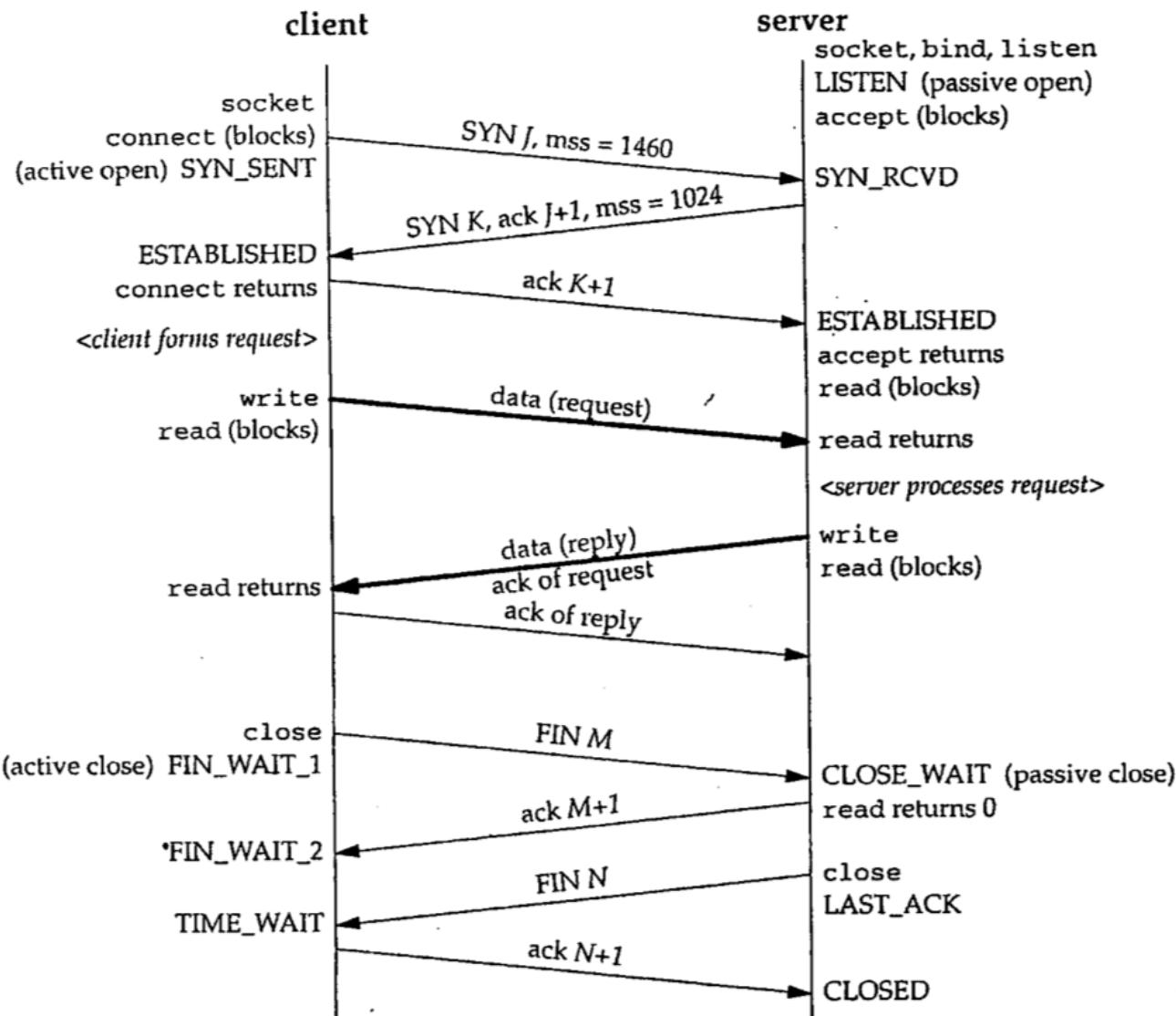


# OPEN SYSTEMS INTERCONNECTION (OSI) МОДЕЛ

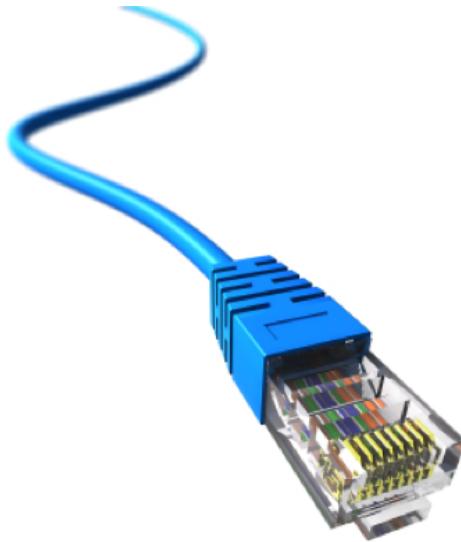
#	Слой	Описание	Протоколи
7	Application	Позволява на потребителските приложения да заявяват услуги или информация, а на сървър приложенията – да се регистрират и предоставят услуги в мрежата.	DNS, FTP, HTTP, NFS, NTP, DHCP, SMTP, Telnet
6	Presentation	Конвертиране, компресиране и криптиране на данни.	TLS/SSL
5	Session	Създаването, поддържането и терминирането на сесии. Сигурност. Логически портове.	Sockets
4	Transport	Грижи се за целостта на съобщенията, за пристигането им в точна последователност, потвърждаване за пристигане, проверка за загуби и дублиращи се съобщения.	TCP, UDP
3	Network	Управлява на пакетите в мрежата. Рутиране. Фрагментация на данните. Логически адреси.	IPv4, IPv6, IPX, ICMP
2	Data Link	Предаване на фреймове от един възел на друг. Управление на последователността на фреймовете. Потвърждения. Проверка за грешки. MAC.	ATM, X.25, DSL, IEEE 802.11
1	Physical	Отговаря за предаването и приемането на неструктурирани потоци от данни по физическият носител. Кодиране/декодиране на данните. Свързване на физическият носител.	IEEE 802.11, IEEE 1394, Bluetooth



# КАК РАБОТИ ТСР ПРОТОКОЛЪТ



# КАК ИДЕНТИФИЦИРАМЕ ЕДИН КОМПЮТЪР В



IP Адрес

Порт

---

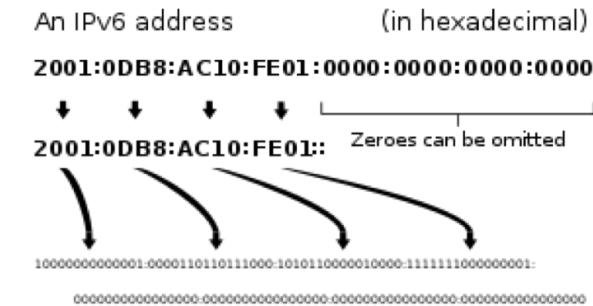
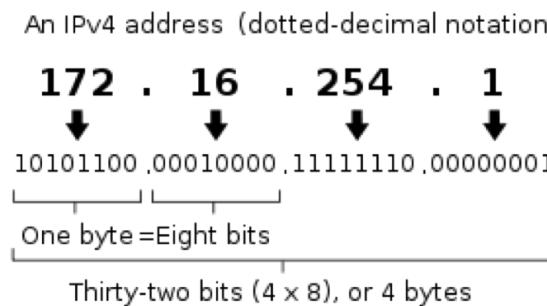
10.199.199.200    50430

IP Адрес + Порт = Socket



# IP ПРОТОКОЛ

- Всеки компютър, свързан към която и да е мрежа, се идентифицира с логически адрес.
- Най-разпространените протоколи за логически адреси в мрежата са IP (Internet Protocol) версия 4 и IP версия 6.
- Адресите в IPv4 представляват 32 битови числа, а в IPv6 - 128 битови.



# ПОРТОВЕ

- В общия случай, компютърът има една физическа връзка към мрежата. По тази връзка се изпращат и получават данни от/за всички приложения. Портовете се използват, за да се знае кои данни за кое приложение са.
- Предадените данни по мрежата винаги съдържат в себе си информация за компютъра и порта, към които са насочени.
- Портовете се идентифицират с 16-битово число, което се използва от UDP и TCP протокола, за да идентифицират за кое приложение са предназначени данните.
- Портовете могат да бъдат от номер 0 до номер 65 535.
- Портове с номера от 0 до 1023 са известни като “well-known ports”. За да се използват тези портове от вашето приложение, то трябва да се изпълнява с администраторски права.



# СОКЕТИ (SOCKET)

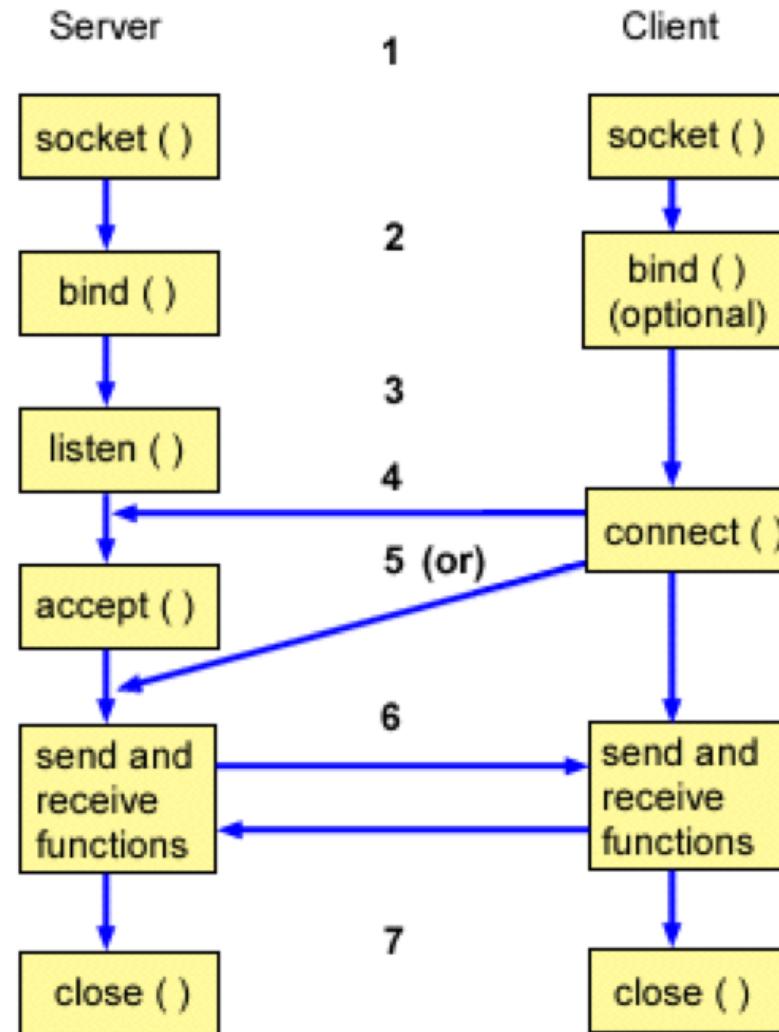
Сокетите се използват при клиент-сървър комуникация и представляват една крайна точка от двупосочна връзка.

## Състояния на сокетите:

1. `socket()` методът създава крайна точка за комуникация и връща дескриптор на сокета.
2. `bind()` методът създава уникално име за този сокет. По този начин, сървърът е достъпен по мрежата.
3. `listen()` методът показва готовност за приемане на клиентски връзки.
4. Клиентското приложение трябва да извика метода `connect()`, за да осъществи връзка със сървъра.
5. Сървърно приложение използва `accept()` метода, за да приеме връзка от клиента.
6. След като е осъществена връзката, може да се използват методите за трансфер на потоци (streams): `send()`, `recv()`, `read()`, `write()` и други.
7. Сървърът или клиентът може да прекратят връзката с метода `close()`.

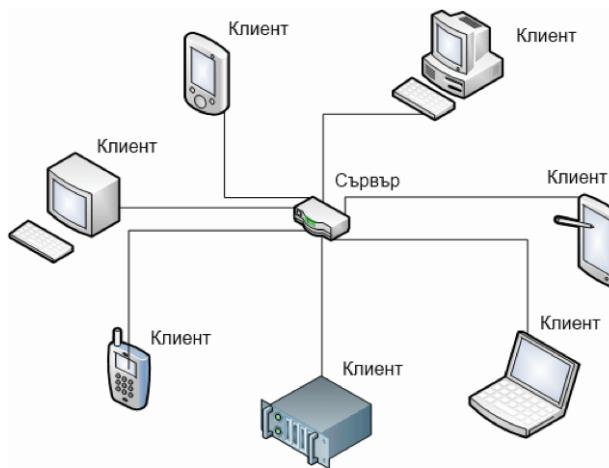


# СОКЕТИ (SOCKET)



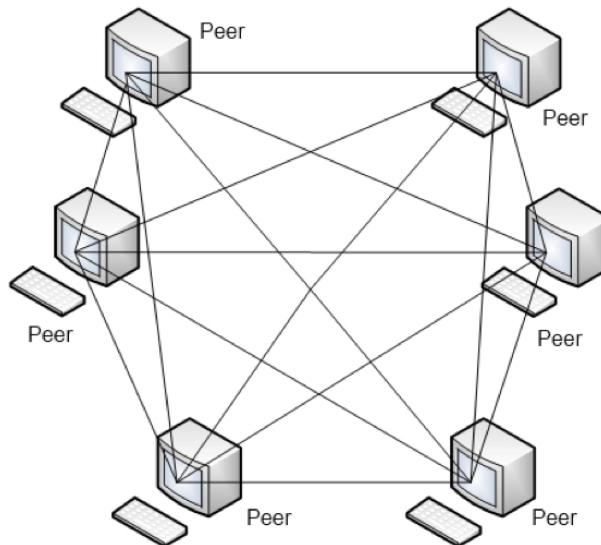
# МОДЕЛ КЛИЕНТ-СЪРВЪР

Клиент-сървър е разпределен изчислителен модел, при който част от задачите се разпределят между доставчиците на ресурси или услуги, наречени сървъри и консуматорите на услуги, наречени клиенти.



# МОДЕЛ PEER-TO-PEER

Peer-to-peer е разпределен архитектурен модел на приложение, при който задачите се разпределят по еднакъв начин между всички участници (peer, node). Всеки участник е едновременно и клиент, и сървър.



# ВИДОВЕ КЛИЕНТИ

Според наличната функционалност в клиента:

- Rich клиенти.
- Thin клиенти.

Според семантиката (протокола):

- Web клиенти – Браузери (Chrome, Firefox, IE).
- Mail клиенти – POP/SMTP клиенти (MS Outlook, Lotus notes).
- FTP клиенти – Total Commander, Filezilla, WinSCP.
- ...

# ВИДОВЕ СЪРВЪРИ

- Файл сървър (Windows, Samba, UNIX NFS, OpenAFS).
- DB сървър (MySQL, PostgreSQL, Oracle, MS SQL Server, MongoDB).
- Mail сървър (MS Exchange, GMail, Lotus Notes).
- Name сървър (DNS).
- FTP сървър (ftpd, IIS).
- Print сървър.
- Game сървър.
- Web сървър (Apache, GWS, MS IIS, nginx).
- Application сървър (Tomcat, GlassFish, JBoss, BEA, Oracle).
- ...



# ПРЕДИМСТВА И НЕДОСТАТЪЦИ НА КЛИЕНТ-СЕРВЪР

- Single Point Of Failure (SPOF).
- Увеличаването на броя на клиентите води до намаляване на производителността.
- 70-95% от времето, през което работи, сървърът е idle.

≡

# ПРЕДИМСТВА И НЕДОСТАТЪЦИ НА PEER-TO-PEER

- Няма SPOF.
- Няма намаляване на производителността при увеличаване на клиентите.
- Проблеми със сигурността.
- Риск от умишлена промяна на съдържание.
- Липса на контрол върху съдържанието и възможност за загуба на съдържание.
- Труден процес на поддръжка.

≡

# JAVA.NET.\* | ВЪВЕДЕНИЕ

Пакетът `java.net` предоставя класове, които използват и работят на различни нива от OSI модела.

- Мрежов и data link слой – класът `NetworkInterface` предоставя достъп до мрежовите адаптери на компютъра.
- Транспортен слой – в зависимост от използваните класове, може да се използват следните транспортни протоколи:
  - TCP – класове `Socket` и `ServerSocket`.
  - UDP – класове `DatagramPacket`, `DatagramSocket`, `MulticastSocket`.
- Приложен слой – класовете `URL` и `URLConnection` се използват за достъпването на HTTP и FTP ресурси.

Класовете в пакета `java.io` се използват за обработка на потоци от данни (data streams). `InputStream`, `BufferedInputStream`, `Reader`, `InputStreamReader`, `BufferedReader`, `OutputStream`, `BufferedOutputStream`, `Writer`, `PrintWriter` са класове, които често се използват при мрежовото програмиране в Java.



# JAVA.NET.\* | МРЕЖОВИ АДАПТЕРИ | 1

- Мрежовият адаптер осъществява връзката между компютърната система и публична или частна мрежа.
- Мрежовите адаптери могат да бъдат физически или виртуални (софтуерни). Примери за виртуални са loopback интерфейсът и интерфейсите, създадени от виртуалните машини.
- Една система може да има повече от един физически и/или виртуален мрежови адаптер.
- Java предоставя достъп до всички мрежови адаптери чрез класа `java.net.NetworkInterface`.



# JAVA.NET\* | МРЕЖОВИ АДАПТЕРИ | 2

С помощта на класа NetworkInterface, може да вземете списък с всички мрежови адаптери (getNetworkInterfaces()) или да вземете точно определен (getByInetAddress() и getByName()).

```
Enumeration<NetworkInterface> nets = NetworkInterface.getNetwo  
for (NetworkInterface netIf : Collections.list(nets)) {  
    System.out.printf("Display name: %s\n", netIf.getDisplayNam  
    System.out.printf("Name: %s\n", netIf.getName());  
    System.out.printf("Addresses: %s\n", printEnum(netIf.getIne  
    System.out.printf("\n");  
}
```

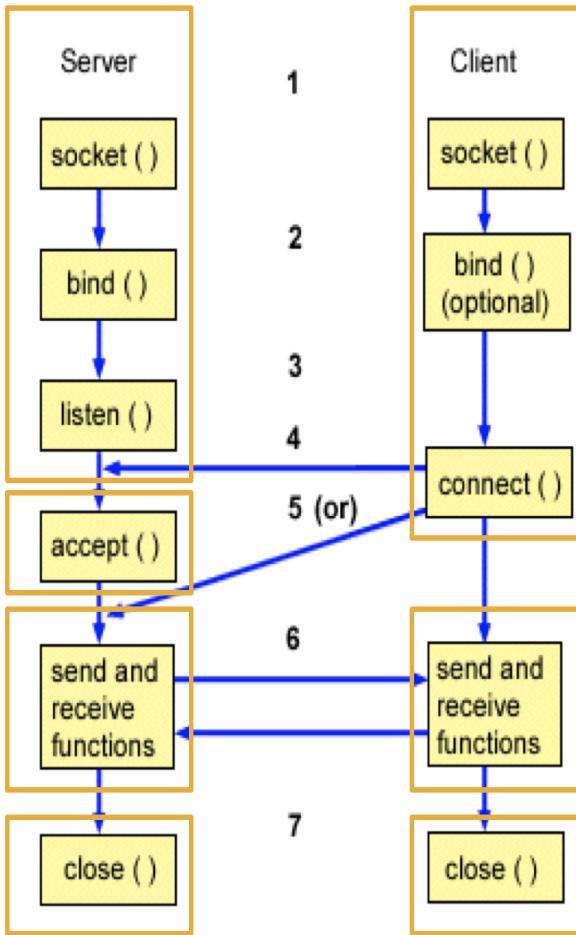
```
Display name: Software Loopback Interface 1  
Name: lo  
Addresses: /0:0:0:0:0:0:0:1, /127.0.0.1,
```

```
Display name: Intel(R) 82578DM Gigabit Network Connection  
Name: eth6  
Addresses: /10.xxx.xxx.xxx,
```

# JAVA.NET\* | TCP КОМУНИКАЦИЯ

Използват се два класа за TCP комуникацията: Socket и ServerSocket

```
ServerSocket serSock =  
new ServerSocket(4444);  
  
Socket sock =  
serverSocket.accept();  
  
OutputStream out =  
sock.getOutputStream();  
InputStream in =  
sock.getInputStream();  
  
out.close();  
in.close();  
sock.close();  
serSock.close();
```



```
Socket sock = new  
Socket("hostname", 4444);
```

```
OutputStream out =  
sock.getOutputStream();  
InputStream in =  
sock.getInputStream();
```

```
out.close();  
in.close();  
sock.close();
```

# JAVA.NET\* | КЛИЕНТ-СЪРВЪР КОМУНИКАЦИЯ |

## 1

```
// Отваряне на сокет от клиентската страна и изпращане на заявка
try {
    Socket s = new Socket("www.cia.gov", 80);
    PrintWriter pw = new PrintWriter(s.getOutputStream());
    pw.println("GET /index.html");
    pw.println();
    pw.flush();
} catch (UnknownHostException e) {
} catch (IOException e) {}
```



# JAVA.NET\* | КЛИЕНТ-СЪРВЪР КОМУНИКАЦИЯ |

## 2

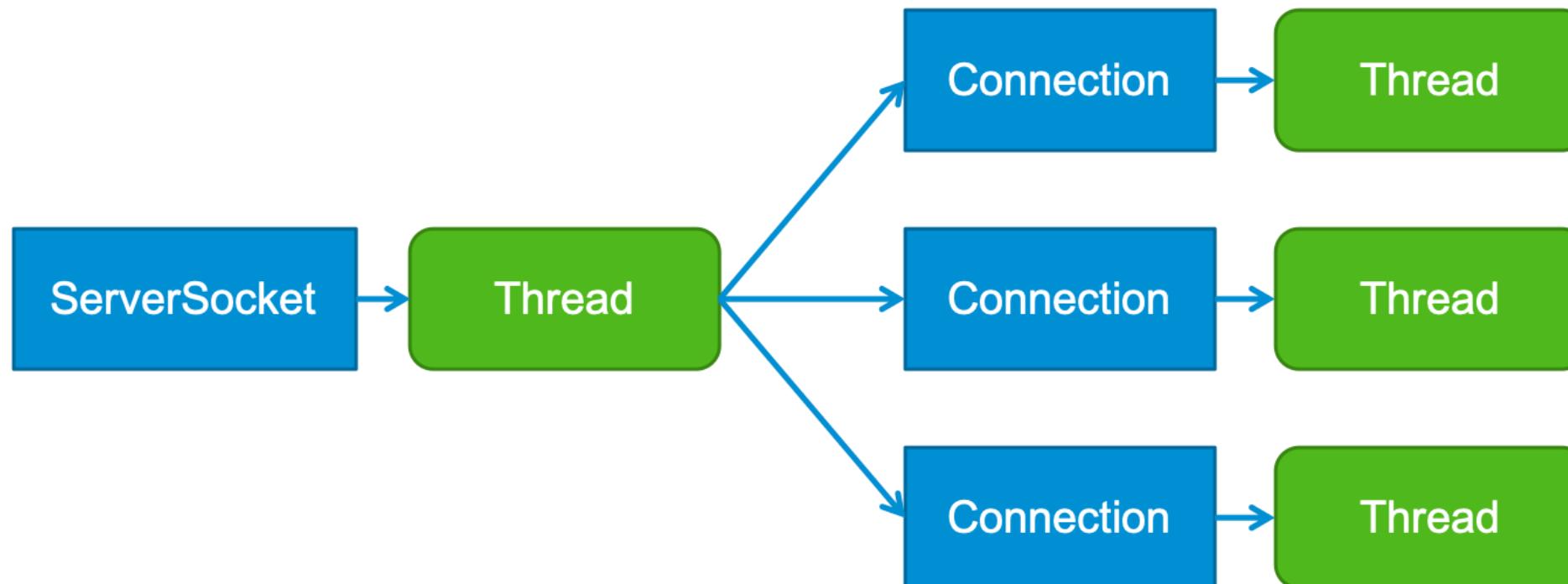
```
try {
    ServerSocket ss = new ServerSocket(80);
    Socket s = ss.accept(); // The thread is blocked.
    // New connection is established. Read the request
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    ByteArrayOutputStream bytes = new ByteArrayOutputStream();
    byte[] b = new byte[2048];
    int r = 0;
    while ((r = is.read(b)) != -1) {
        bytes.write(b, 0, r);
    }
} catch (IOException e) {}
```



# JAVA.NET\* | АРХИТЕКТУРА

За класовете от пакета `java.net` е необходимо да има една нишка за всяка връзка (connection).

Синхронни (блокиращи) операции.



# JAVA.NET\* | КЛИЕНТ-СЪРВЪР КОМУНИКАЦИЯ |

## 3

```
// Обработка на заявката и връщане на отговор:  
try {  
    // Process request  
    // Send response  
    PrintWriter pw = new PrintWriter(s.getOutputStream());  
    pw.println("Hello World");  
    pw.flush();  
    pw.close();  
    is.close();  
    s.close();  
} catch (IOException e) {}
```



# JAVA.NET\* | КЛИЕНТ-СЪРВЪР КОМУНИКАЦИЯ |

## 4

```
// Прочитане на отговора (response) от сървъра:  
try {  
    ...  
    BufferedReader br = new BufferedReader(  
        new InputStreamReader(s.getInputStream()));  
    String line;  
    while ((line = br.readLine()) != null) {  
        System.out.println(line);  
    }  
    pw.close();  
    br.close();  
    s.close();  
} catch (IOException e) {}
```



## JAVA.NIO.\* | ВЪВЕДЕНИЕ

Въведен от JDK 1.4. Позволява асинхронни входно-изходни (I/O) операции. Намалява генерирането на “боклук” чрез използването на буфери с директна памет (DMA), при които при писането и четенето в сокетите не се извършва копиране на данни.



# JAVA.NIO.\* | ОСНОВНИ ОБЕКТИ

- Буфери (Buffers) – Представляват блок от паметта, в който може да се записват данни. Използват се за четене и запис в NIO канали (channels).
- Канали (Channels) – Подобни на потоците (Stream). Представляват една връзка, като от тях може да се чете и да се записва. Основните класове: FileChannel, DatagramChannel, SocketChannel, ServerSocketChannel.
- Селектор (Selector) – Компонент, в който се регистрират канали и може да обработва повече от един канал в една нишка.



# JAVA.IO.\* | ЧТЕНИЕ И ЗАПИС С ФАЙЛОВЕ | 1

```
try (FileInputStream in = new FileInputStream("input.txt");
     FileOutputStream out = new FileOutputStream("output.txt")) {
    int c;
    while ((c = in.read()) != -1) {
        out.write(c);
    }
}
```

Отваряме  
файловете

Край на файл

Четем 1 В от  
Stream-a

Записваме

# JAVA.NIO.\* | ЧЕТЕНЕ И ЗАПИС ВЪВ/ОТ ФАЙЛ

```
FileChannel fcin = fin.getChannel();
FileChannel fcout = fout.getChannel();
```

Създаваме буфер с размер 1024 B

```
ByteBuffer buffer = ByteBuffer.allocate(1024);
```

Зачистваме буфера

```
while (true) {
```

```
    buffer.clear();
```

Прочитаме до 1024 байта от канала

```
    int r = fcin.read(buffer);
```

Край на файл

```
    if (r == -1) {break;}
```

Подготвяме буфера за запис

```
    buffer.flip();
```

```
    fcout.write(buffer);
```

Записваме



# JAVA.NIO.\* | БУФЕРИ

- Буферът е обект, който съдържа данни, които ще бъдат записани или са били прочетени.
- В NIO, за обработката на данни се използват буфери и това е една от най-големите разлики в сравнение със стандартния вход/изход, където се използват потоци (Stream).
- NIO предоставя имплементации на буфери за всички не-булеви примитивни типове: ByteBuffer, CharBuffer, ShortBuffer, IntBuffer, LongBuffer, FloatBuffer, DoubleBuffer
- Буферът като цяло е масив от данни, който пази в себе си състоянието си: докъде е запълнен, колко свободно място има, докъде са прочетени или записани данните.



# JAVA.NIO.\* | ВЪТРЕШНО СЪСТОЯНИЕ НА БУФЕРА

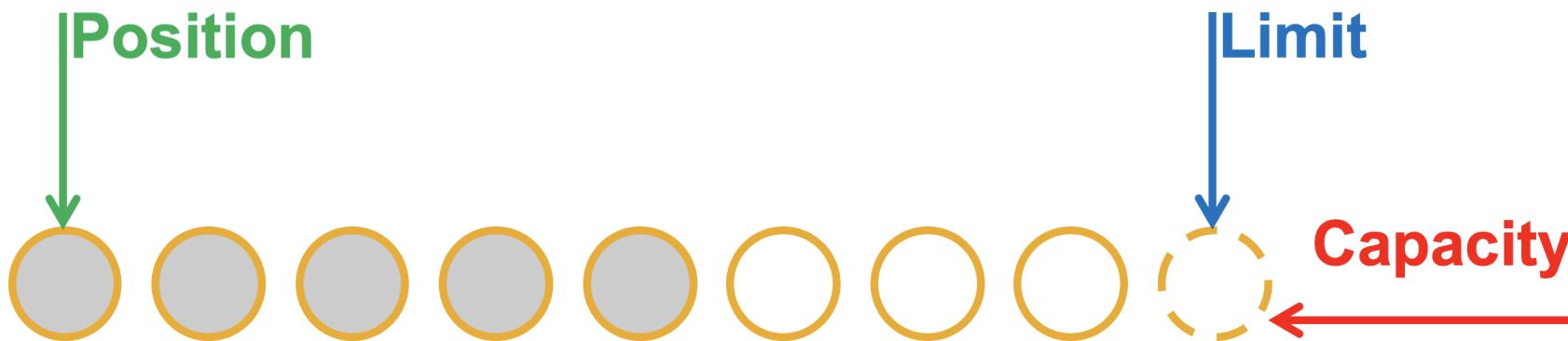
В себе си буферът пази четири състояния:

- Позиция (position) – съдържа информация до кой елемент на масива сме стигнали.
- Лимит (limit) – съдържа информация колко данни още можем да прочетем от масива или още колко може да запишем.
- Обем (capacity) – показва какво е максималното количество данни, което може да съдържа буферът.
- Маркер (mark) – програмистът може да запомни някоя позиция от масива.

Зависимостта между четирите състояния е:  $0 \leq \text{mark} \leq \text{position} \leq \text{limit} \leq \text{capacity}$



# JAVA.NIO.\* | КАК РАБОТИ БУФЕРЪТ



```
ByteBuffer buffer = ByteBuffer.allocate(8);

fcin.read(buffer);

buffer.put(new byte[2]);

buffer.flip();

fcout.write(buffer);

buffer.get(new byte[2]);

buffer.clear();
```

# JAVA.NIO.\* | ДОСТЪП ДО ПАМЕТТА | 1

Indirect Buffer (индиректен буфер).

```
ByteBuffer buffer = ByteBuffer.allocate(1024);
```

Direct Buffer (директен буфер) – ще се опита да използва native операции за четене и запис директно чрез операционната система.

```
ByteBuffer buffer = ByteBuffer.allocateDirect(1024);
```



## JAVA.NIO.\* | ДОСТЪП ДО ПАМЕТТА | 2

Memory-mapped Buffer (директен буфер) – използва функционалност на операционната система, за да обвърже даден файл с част от оперативната памет.

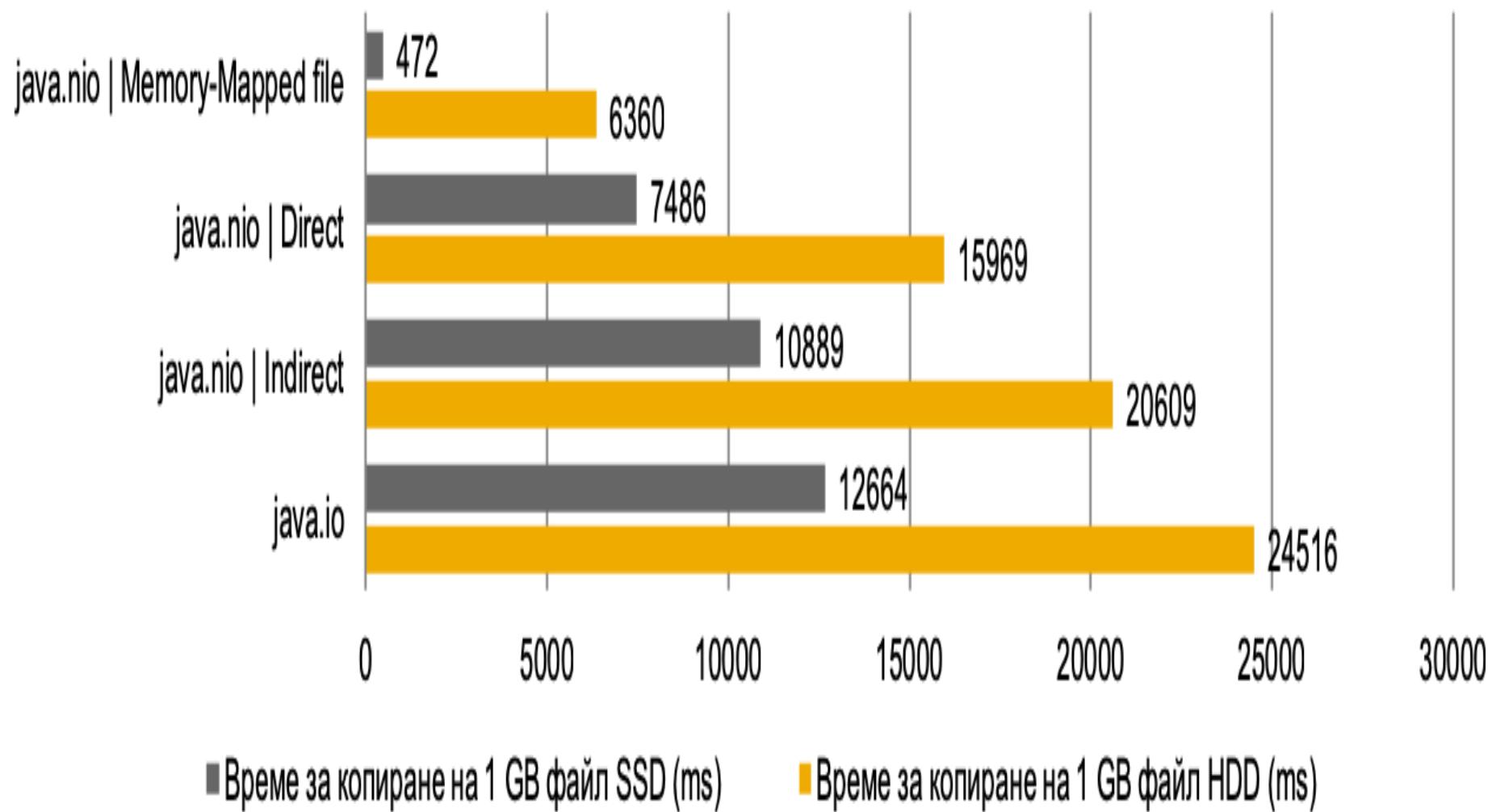
```
MappedByteBuffer mbb = fc.map(FileChannel.MapMode.READ_WRITE,
```

Read-only Buffer (буфер за четене).

```
ByteBuffer readBuffer = buffer.asReadOnlyBuffer();
```



# JAVA.NIO.\* | ДОСТЪП ДО ПАМЕТТА | 3



# JAVA.NIO.\* | SERVERSOCKETCHANNEL

```
// ServerSocketChannel е канал (java.nio.Channel), който може  
// да слуша за входящи TCP повиквания, точно както ServerSocke  
  
ServerSocketChannel serverSocketChannel = ServerSocketChannel.  
serverSocketChannel.socket().bind(new InetSocketAddress(9999))  
serverSocketChannel.configureBlocking(false);  
  
while (true) {  
    SocketChannel socketChannel = serverSocketChannel.accept()  
    ...  
}
```



# JAVA.NIO.\* | SOCKETCHANNEL | РЕГИСТРАЦИЯ | 1

SocketChannel представлява една TCP връзка. За да се използва асинхронно, трябва да се регистрира в селектор. Каналът трябва да се постави в nonblocking режим (чрез извикване на `configureBlocking(false)`) преди да може да бъде регистриран със селектор.

```
Selector selector = Selector.open();
socketChannel.configureBlocking(false);
socketChannel.register(selector, SelectionKey.OP_READ);
```



# JAVA.NIO.\* | SOCKETCHANNEL | РЕГИСТРАЦИЯ | 2

Когато регистрираме SocketChannel, трябва да укажем за какви операции да бъдем известени:

- OP\_READ – ще се получи известие (notification), когато се получи нещо за четене от канала.
- OP\_WRITE – ще се получи известие (notification), когато каналът е готов за запис.
- OP\_CONNECT – ще се получи известие (notification), когато каналът е готов да завърши последователността си за свързване към отдалечената система.
- OP\_ACCEPT – ще се получи известие (notification), когато пристигне нова конекция.



# JAVA.NIO.\* | SOCKETCHANNEL | NOTIFICATION

```
// Получаване на известие за няколко канала.  
while (true) {  
    int readyChannels = selector.select();  
    if (readyChannels == 0) { continue; }  
  
    Set<SelectionKey> selectedKeys = selector.selectedKeys();  
    Iterator<SelectionKey> keyIterator = selectedKeys.iterator()  
    while (keyIterator.hasNext()) {  
        SelectionKey key = keyIterator.next();  
        if (key.isReadable()) {  
            // A channel is ready for reading  
        }  
        keyIterator.remove();  
    }  
}
```



# JAVA.NIO.\* | SOCKETCHANNEL | NOTIFICATION | ЧТЕНЕ

```
// Четене на данните от канал.  
if (key.isReadable()) {  
    // Read the data  
    SocketChannel sc = (SocketChannel) key.channel();  
    while (true) {  
        buffer.clear();  
        int r = sc.read( buffer );  
        if (r<=0) { break; }  
        buffer.flip();  
        sc.write( buffer );  
    }  
} else if (key.isWritable()) {  
    ...  
}
```



# JAVA.NIO.\* | SOCKETCHANNEL | ЗАПИС

```
// Използваме SocketChannel канал (channel) и за изпращане на
// по TCP връзката (connection).

SocketChannel socketChannel = SocketChannel.open();
socketChannel.connect(new InetSocketAddress("127.0.0.1", 9999))

String newData = "Current time: " + System.currentTimeMillis()
ByteBuffer buf = ByteBuffer.allocate(48);

buf.put(newData.getBytes());
buf.flip();
while (buf.hasRemaining()) {
    socketChannel.write(buf);
}
```



# JAVA.NIO.\* | АРХИТЕКТУРА

Нишка селектор (Selector), която позволява обработването на няколко канала от една нишка.

Намалява броя на нишките, като премахва нуждата от отделна нишка за всяка връзка (connection).

Асинхронни (неблокиращи) операции.

