Effective Software

Lecture 9: Non-blocking I/O, C10K, efficient networking, threads

David Šišlák
david.sislak@fel.cvut.cz

[1] Tanenbaum, A. S., Wetherall, D. J.: Computer Networks. Pearson, 2011.
[2] Kegel, D.: The C10K problem. http://www.kegel.com/c10k.html
[3] Hitchens, R.: Java NIO. O'Reilly, 2002.
[4] Pressler, R., Bateman, A.: JEP 436 - Virtual Threads (second preview)

Cisco Confidential

Outline

- » Network communication
  - OSI model
- » C10k problem
  - Thread-per-request vs. event-based approach
- » Non-blocking I/O
  - Select
  - Poll
  - Epoll
  - Java non-blocking I/O
    - Native memory buffer
    - NIO
- » Threads
  - Thread pools
  - Virtual threads

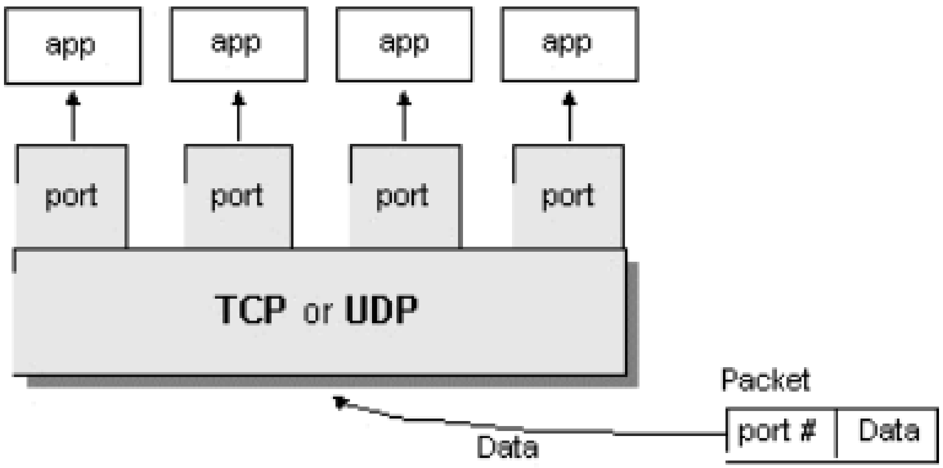
Cisco Confidential

Network Communication – OSI Model

Layer	Data unit	Function	Examples
7 – Application Interface to end user. Interaction directly with software application.		**Software App Layer** Directory services, email, network management, file transfer, web pages, database access.	FTP, HTTP, WWW, SMTP, TELNET, DNS, TFTP, NFS
6 – Presentation Formats data to be "presented" between application-layer entities.		**Syntax/Semantics Layer** Data translation, compression, encryption/decryption, formatting.	ASCII, JPEG, MPEG, GIF, MIDI
5 – Session Manages connections between local and remote application.		**Application Session Management** Session establishment/teardown, file transfer checkpoints, interactive login.	SQL, RPC, NFS
4 – Transport Ensures integrity of data transmission.	Segment	**End-to-End Transport Services** Data segmentation, reliability, multiplexing, connection-oriented, flow control, sequencing, error checking.	TCP, UDP, SPX, AppleTalk
3 – Network Determines how data gets from one host to another.	Packet	**Routing** Packets, subnetting, logical IP addressing, path determination, connectionless.	IP, IPX, ICMP, ARP, PING, Traceroute
2 – Data Link Defines format of data on the network.	Frame	**Switching** Frame traffic control, CRC error checking, encapsulates packets, MAC addresses.	Switches, Bridges, Frames, PPP/SLIP, Ethernet
1 – Physical Transmits raw bit stream over physical medium.	Bits	**Cabling/Network Interface** Manages physical connections, interpretation of bit stream into electrical signals	Binary transmission, bit rates, voltage levels, Hubs

Cisco Confidential

Network Communication – Introduction



TCP		UDP	
FTP	20,21	DNS	53
SSH	22	BooTPS/DHCP	67
Telnet	23	TFTP	69
SMTP	25	SNMP	161
DNS	53		
HTTP	80		
POP3	110		
NTP	123		
IMAP4	143		
HTTPS	443		

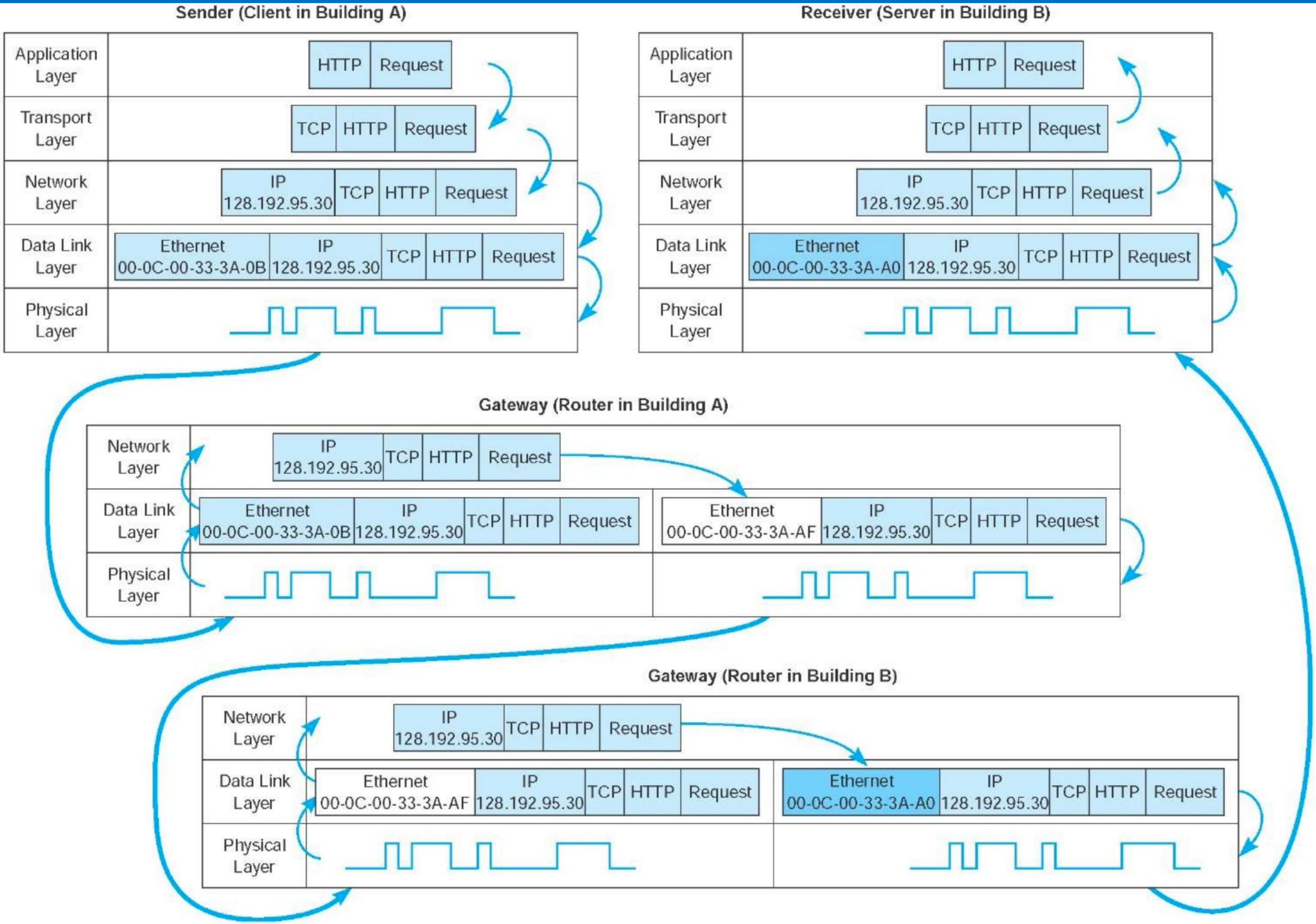
- **ports** – 16-bit number
- **IPv4** – 32-bit address
- **IPv6** – 128-bit address
  - 48-bit or more routing prefix, 16-bit or less subnet id, 64-bit interface

    http://[1fff:0:a88:85a3::ac1f]:8080/index.html

- TCP/UDP connection identification – **quad** – src IP, src port, dst IP, dst port

Cisco Confidential

Network Communication – HTTP Example



C10k Problem

- handling a huge number of clients (10 000s) at the same time (late 90s)
  - **concurrent connections by one server** requiring efficient scheduling
  - not related to requests per second
- sometime known as C1M or C10M problem (nowadays)
- approach
  - **thread-per-request servers** (*Apache*)
    - each connection handled by **own thread**/process (pooled but limited)
    - connection operations usually use **blocking** operations
    - thread scheduling doesn’t scale (+cost for thread context switching)
    - thread scheduling used as packet scheduling
  - **event-driven I/O servers** (*Nginx*)
    - do packet scheduling yourself – **single/multi-threaded event loop**
    - using **non-blocking** (asynchronous) operations with **event interceptors**
    - **multi-core scalability** with controlled number of worker threads
    - reuse thread-based data structures, avoid locks (atomics, non-blocking)

Cisco Confidential

Non-Blocking I/O Approach

» **interrupts**
  - hardware interrupts in kernel mode

» **polling**
  - looping to regularly check status (readiness for I/O)
  - wastes CPU cycles

» **signals**
  - OS generated signals on I/O readiness
  - might leave state inconsistent in the process inconsistent

» **callbacks**
  - pointer to handler function
  - stack deepening issue (callback issuing I/O)

» **event-based**
  - select
  - poll
  - epoll

Cisco Confidential

Event-Based I/O - select

» **select**

- defined in POSIX (Portable Operating System Interface)
- originally used for blocking I/O
- passed **lists of *descriptors* cannot be reused** in subsequent calls as they are modified by the system call
- **not scalable** – limited *descriptors* + iterate over to find the event

```
int
select(int nfds, fd_set *restrict readfds, fd_set *restrict writefds, fd_set *restrict errorfds,
struct timeval *restrict timeout);

void
FD_CLR(fd, fd_set *fdset);

void
FD_COPY(fd_set *fdset_orig, fd_set *fdset_copy);

int
FD_ISSET(fd, fd_set *fdset);

void
FD_SET(fd, fd_set *fdset);

void
FD_ZERO(fd_set *fdset);
```


Cisco Confidential

Event-Based I/O - poll

» **poll**

- polled descriptors not limited
- descriptors can be reused
- better but you still **need iterate over descriptors** to find events

```
int
poll(struct pollfd fds[], nfds_t nfds, int timeout);
```

```
struct pollfd {
    int    fd;       /* file descriptor */
    short  events;   /* events to look for */
    short  revents;  /* events returned */
};
```


Cisco Confidential

Event-Based I/O - epoll

- **epoll**
  - Linux only (e.g. Windows has IOCP – IO Completion Ports)
  - **scalable**
  - monitored events can be modified while polling (via syscall)
  - returns triggered events directly
- **API**
  - epoll_create & epoll_create1 – initialize epoll instance (kernel structure)
  - epoll_ctl – add/modify/remove descriptors to epoll instance
  - epoll_wait – wait for events up to timeout
- **modes**
  - **level triggered** – wait always returns if event is available
  - **edge triggered** (EPOLLET) – readiness returned upon incoming event only

    (you have to process all pending events before next wait !)
- **events**
  - EPOLLIN, EPOLLOUT, EPOLLPRI
  - EPOLLRDHUP, EPOLLHUP
  - EPOLLERR

Epoll Usage

epoll structure:

```
typedef union epoll_data
{
  void          *ptr;
  int            fd;
  __uint32_t    u32;
  __uint64_t    u64;
} epoll_data_t;

struct epoll_event
{
  __uint32_t   events; /* Epoll events */
  epoll_data_t data;   /* User data variable */
};
```

initialization:

```
int epfd = epoll_create1(0);
...
struct epoll_event ev;
int client_sock;
...
ev.events = EPOLLIN | EPOLLPRI | EPOLLERR | EPOLLHUP;
ev.data.fd = client_sock;
int res = epoll_ctl(epfd, EPOLL_CTL_ADD, client_sock, &ev);
```


Cisco Confidential

Epoll Event Loop

```
while (1) {
     // wait for something to do...
     int nfds = epoll_wait(epfd, events,
                                    MAX_EPOLL_EVENTS_PER_RUN,
                                    EPOLL_RUN_TIMEOUT);
     if (nfds < 0) die("Error in epoll_wait!");

     // for each ready socket
     for(int i = 0; i < nfds; i++) {
       int fd = events[i].data.fd;
       handle_io_on_socket(fd);
     }
}
```


Cisco Confidential

JAVA Blocking Networking – TCP Client

» **Socket**

- client end-point of network TCP/IP connection
- is bound to particular destination IP and port
- each TCP/IP connection is uniquely identified by its two end-points
- provides input/output streams

```java
try (
    Socket echoSocket = new Socket( host: "localhost",  port: 7);
    PrintWriter out = new PrintWriter(echoSocket.getOutputStream(),  autoFlush: true);
    BufferedReader in = new BufferedReader(new InputStreamReader(echoSocket.getInputStream()));
    BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in))
) {

    String userInput;

    while ((userInput = stdIn.readLine()) != null) {
        out.println(userInput);
        System.out.println("echo: " + in.readLine());
    }
}
```


Cisco Confidential

JAVA Blocking Networking – TCP Server

» **ServerSocket**

- server socket representing listening TCP/IP end-point
- within constructor you specify the port, and optionally IP where it should be bound
- wait for establishing connection using method Socket **accept**()

Cisco Confidential

JAVA Blocking Networking – TCP Server - Example

thread-per-request server example – each handler in own thread with blocking I/O

```
ExecutorService clientRunner = Executors.newCachedThreadPool();
try (
        ServerSocket serverSocket = new ServerSocket( port: 7)
) {
    while (true) {
        final Socket s = serverSocket.accept();
        clientRunner.execute(() -> {
            try (
                    BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
                    PrintWriter out = new PrintWriter(s.getOutputStream(),  autoFlush: true)
            ) {
                String line;
                while (s.isConnected()) {
                    if ((line = in.readLine()) != null) {
                        out.println(line);
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        });
    }
} catch (Exception e) {
    e.printStackTrace();
} finally {
    clientRunner.shutdownNow();
}
```


Cisco Confidential

JAVA Blocking Networking - UDP

» **DatagramPacket**
- independent, self-contained message sent over the network
- like network **packet**
  - InetAddress address, int port – destination
  - byte data[], int length, int offset
  - SocketAddress sa – sender

» **DatagramSocket**
- **sending or receiving point** for a packet delivery service
- can be bound to any available port (using default constructor)
- connect(InetAddress,int) – can sent or receive packets only specified host, if not set in DatagramPacket automatically fill
- send(DatagramPacket p), receive(DatagramPacket p) – blocking IO

» **MulticastSocket**
- additional capabilities for joining/leaving multicast groups, loopback
- multicast IP (IGMP – Internet Group Management Protocol)

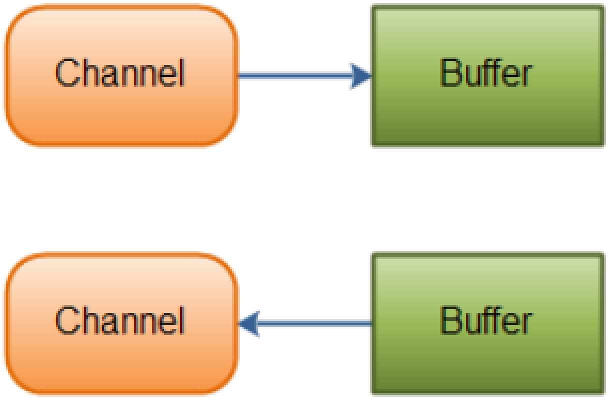
  224.0.0.0 – 239.255.255.255

Cisco Confidential

JAVA Non-blocking Networking - NIO

- **scalable I/O** – asynchronous I/O requests and polling
- high-speed **block-oriented** binary and character I/O working – including mapping files to the memory, using channels and selectors

- Channel is a block device working with Buffers



Cisco Confidential

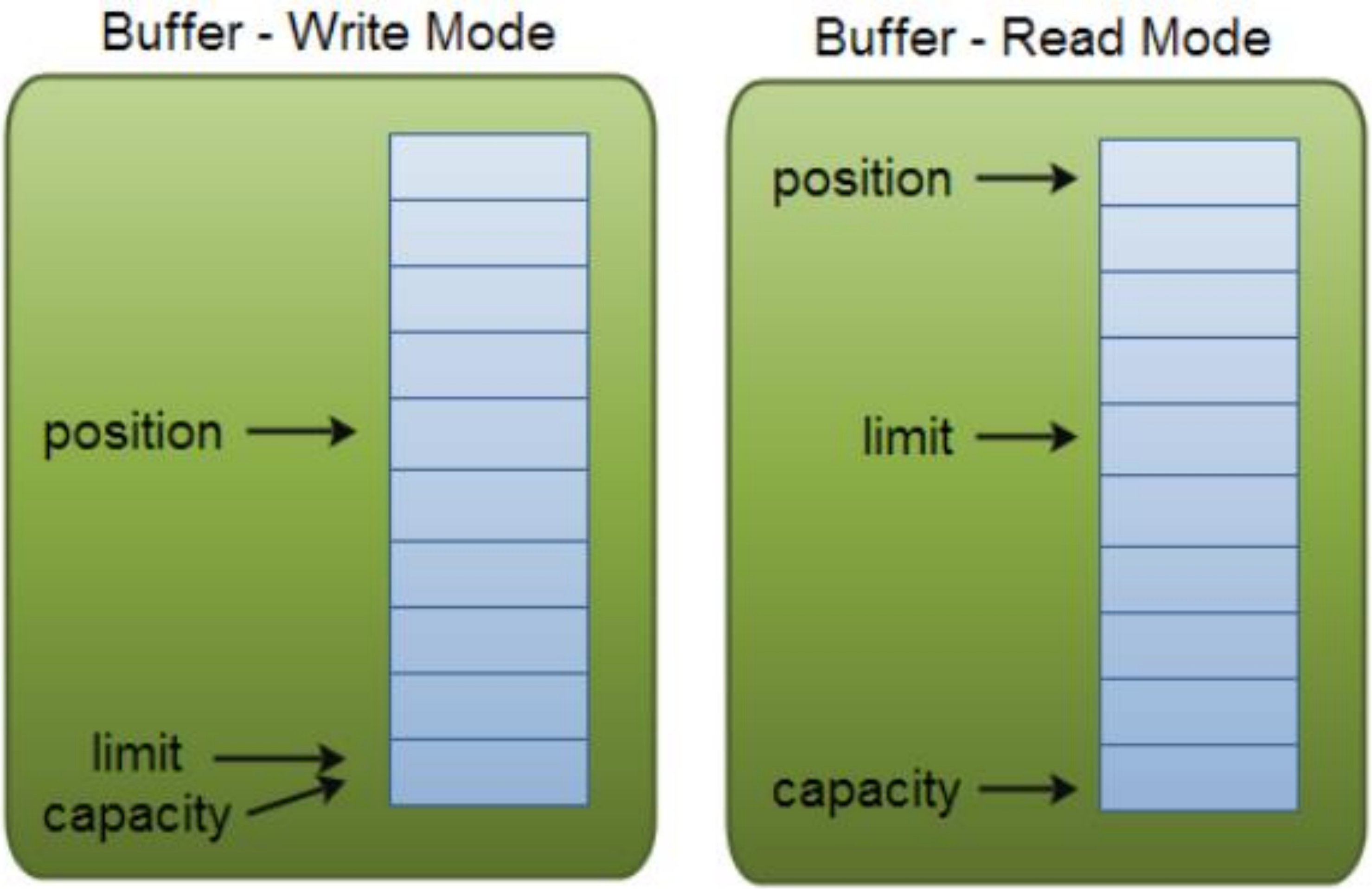
JAVA – NIO - Buffer

» **java.nio.Buffer**

- **linear, finite sequence of elements** of a specific primitive type
  - ByteBuffer, CharBuffer, DoubleBuffer, FloatBuffer, IntBuffer, LongBuffer, ShortBuffer, MappedByteBuffer {FileChannel.map(…)}
- not thread safe, **multi mode** for the same buffer (both read & write)
- **key properties** – 0 <= mark <= position <= limit <= capacity
  - capacity – numbers of elements, never changing !
  - limit – index of the first element that should not be read or written
  - position – index of the next element to be read or written
  - mark – index to which its position is set after reset()
- clear() – position=0, limit=capacity => **ready for channel read** (put)
- flip() – limit=position, position=0 => **ready for channel write** (get)
- rewind() – limit unchanged, position=0 => ready for re-reading
- mark() – mark = position
- reset() – position=mark

Cisco Confidential

JAVA – NIO - Buffer



- write mode – channel.read(buf); buf.put(…);
- read mode – channel.write(buf); … buf.get();

Cisco Confidential

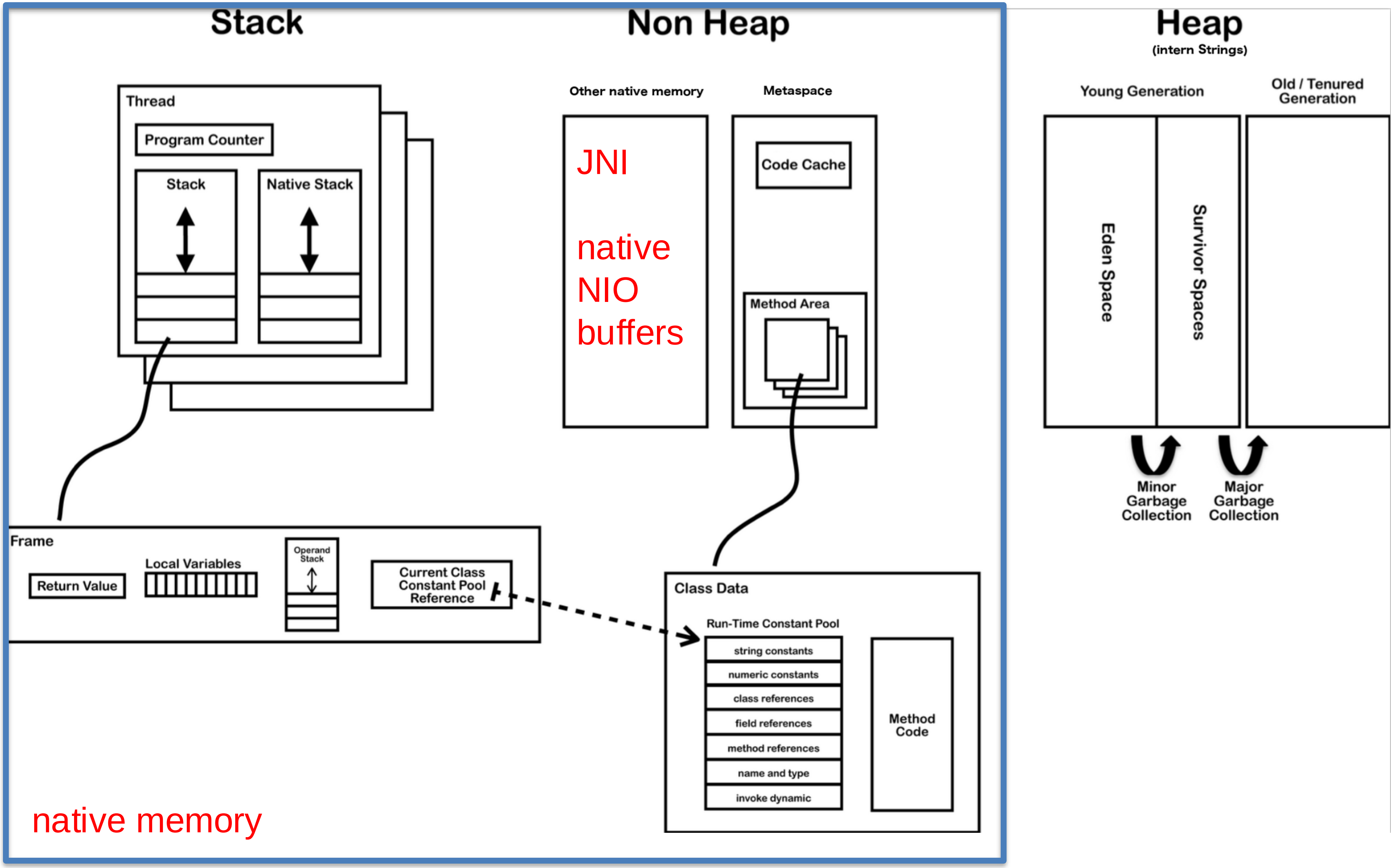
JAVA – NIO - Buffer

» java.nio.Buffer
  - isReadOnly() – can be read-only
  - hasArray() – is backed by an accessible array (array())
  - equals(), compareTo() – compare remainder sequence
  - can be **allocated to native memory** (see next slide)
  - **typical usage**
    1. Write data into the Buffer
    2. Call `buffer.flip()`
    3. Read data out of the Buffer
    4. Call `buffer.clear()` or `buffer.compact()`

Note: compact() – bytes between position and limit are copied to the beginning of the buffer and prepare for writing again

Cisco Confidential

JVM – Memory Layout – Native Memory



Cisco Confidential

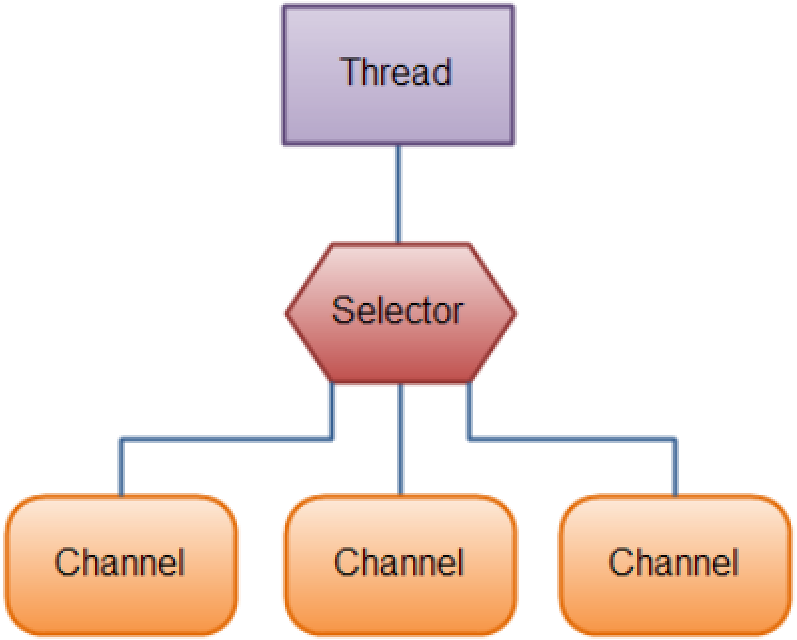
JVM – NIO - Direct Buffers

- » **ByteBuffer.allocateDirect**(…)
- » stored out of JAVA heap in **native memory**
- » allow native code and Java code to **share data without copying**
  - useful for file and socket
    - the same memory is passed to kernel during calls
- » multiple buffers can share native memory
  - slice()/duplicate() – independent position, limit, mark, shared content
  - asReadOnlyBuffer() – read only view of shared content
- » tuning/tracking
  - -XX:MaxDirectMemorySize=N (default unlimited)
  - -XX:NativeMemoryTracking=off|summary|detail
  - -XX:+PrintNMTStatistics

Note: usage of heap buffers implies content copy out/in Java heap space due to possible relocations by GC

Cisco Confidential

JAVA Networking - NIO – Channel, Selector



» **one thread** works with **multiple channels at the same time**
  - **epoll-based** if OS support epoll

» **Channel** – cover UDP+TCP network IO, file IO
  - FileChannel – from Input/OutputStream or RandomAccessFile
  - DatagramChannel
  - MulticastChannel
  - SocketChannel
  - ServerSocketChannel

Cisco Confidential

JAVA – NIO – Channel

» **Channel**
  - read/write at the same time (streams are only one-way)
  - always read/write from/to a **buffer**

» **FileChannel**
  - only **blocking**
  - support – direct buffers, mapped files, locking
  - bulk transfers between channels
    - no copy at all, direct transfer e.g. to socket
    - **transferFrom**(sourceChannel, int pos, int count)
    - **transferTo**(int pos, int count, dstChannel)

Cisco Confidential

JAVA – NIO – Channel

» **SocketChannel** – client end-point of TCP/IP

- can be configured as **non-blocking** before connecting
- SocketChannel socket.getChannel();
- SocketChannel SocketChannel.open();
- sch.connect(...)

- write(...) and read(...) may return without having written/read anything for non-blocking channel

» **ServerSocketChannel** – server end-point of TCP/IP

- can be configured as **non-blocking**
- can be created directly using open() or from ServerSocket
- accept() – returns SocketChannel in the same mode

Cisco Confidential

JAVA – NIO – Selector

» **Selector**
  - Selector Selector.open();
  - only channels in **non-blocking** mode can be registered

    channel.configureBlocking(false);

    SelectionKey channel.register(selector, SelectionKey.OP_READ);
  - FileChannel doesn't support non-blocking mode

» **SelectionKey** – events you can listen for (multiple can be combined)
  - OP_CONNECT
  - OP_ACCEPT
  - OP_READ
  - OP_WRITE

» events are filled by channel which is ready with operation

Cisco Confidential

JAVA – NIO – Selector

» **SelectionKey** – returned from register method

- interest set – your configured ops
- ready set – which ops are ready, sk.isReadable(), sk.isWritable(), …
- channel
- selector
- optional attached object – sk.attach(Object obj);
  Object sk.attachment()

SelectionKey channel.register(selector, ops, attachmentObj);

Cisco Confidential

JAVA – NIO – Selector

» Selector with registered one or more channels
  - int **select**() – blocks until at least one channel is ready
  - int select(long timeout) – with timeout milliseconds
  - int selectNow() – doesn't block at all, returns immediately

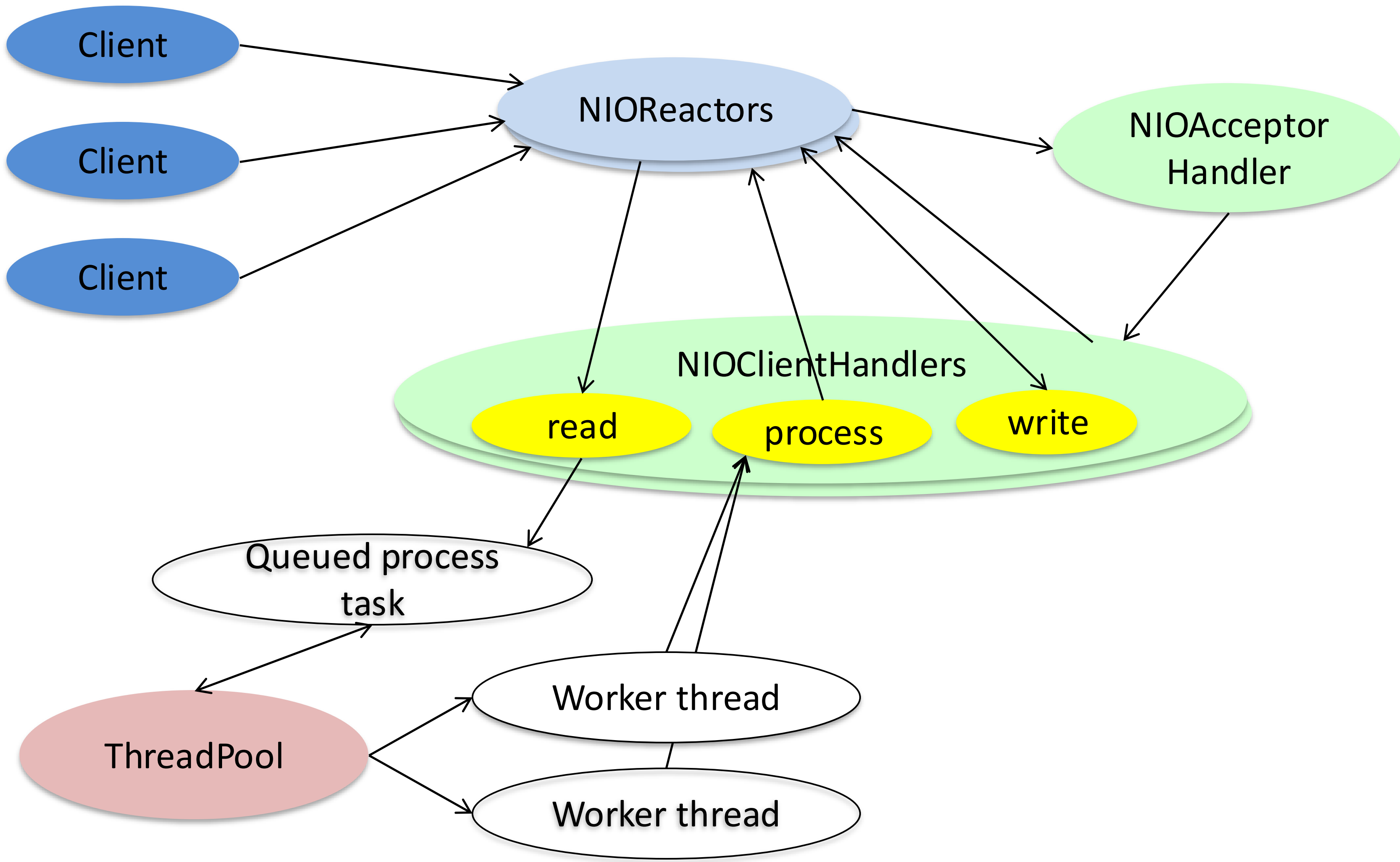
return the number of channels which are ready from the last call

Set<SelectionKey> selector.selectedKeys();

  - **wakeUp**() – different thread can "wake up" thread blocked in select()
  - **close**() – invalidates selector, channels are not closed

Cisco Confidential

JAVA – NIO Server – Using Multiple Reactors



Cisco Confidential

Threads

» **processes** vs. **threads**
  - both support concurrent execution
  - one process has one or multiple threads
  - threads share the same address space (data and code)
  - local variables, exception handling, debugging and profiling
  - context switching between threads is usually less expensive
  - thread inter-communication is relatively efficient using shared memory

» **JVM**
  - a thread executes sequence of code with own stack with frames
    t.getStackTrace()
  - own local variables
  - own method parameters

» thread creation by
  - subclass of **java.lang.Thread**
  - implementation of **java.lang.Runnable**

Cisco Confidential

JAVA Thread Pool - ExecutorService

» concept of **thread pooling**

» suitable for execution of large number of asynchronous tasks

- e.g., processing of requests in server

» **reduce overhead with Thread creation for each task, context switching**

» interface - java.util.concurrent.**ExecutorService**

- shutdown(), shutdownNow(), awaitTermination
- **execute**(Runnable r)
- Future<?> **submit**(Runnable r), Future<T> **submit**(Callable<T> c)

» java.util.concurrent.**Future**<T>

- boolean cancel(boolean mayInterruptIfRunning)
- isCancelled(), isDone()
- V get(), V get(long timeout, TimeUnit unit)

» java.util.concurrent.Executors (optionally with ThreadFactory)

- **newSingleThreadExecutor**()
- **newFixedThreadPool**(nThreads)
- **newCachedThreadPool**() – default 60 seconds keep-alive

Cisco Confidential

JAVA Virtual Threads - Introduction

» **lightweight implementation of Java thread**

» available from Java 21

- preview feature since Java 19 (attribute --enable-preview)

» **standard thread**

- thin wrapper around OS-managed platform thread
- basic unit of OS scheduling
- creation/removal is expensive and resource-heavy operation
- fixed thread stack size -> StackOverflowException
- doesn’t scale

» **alternatives**

- async/await – reactive-style programming (e.g. Kotlin)
  - asynchronous operations with callback
- issues with readable stack-traces, debugging and observability
- complex workflow for sequential composition, iteration, try-catch blocks

Cisco Confidential

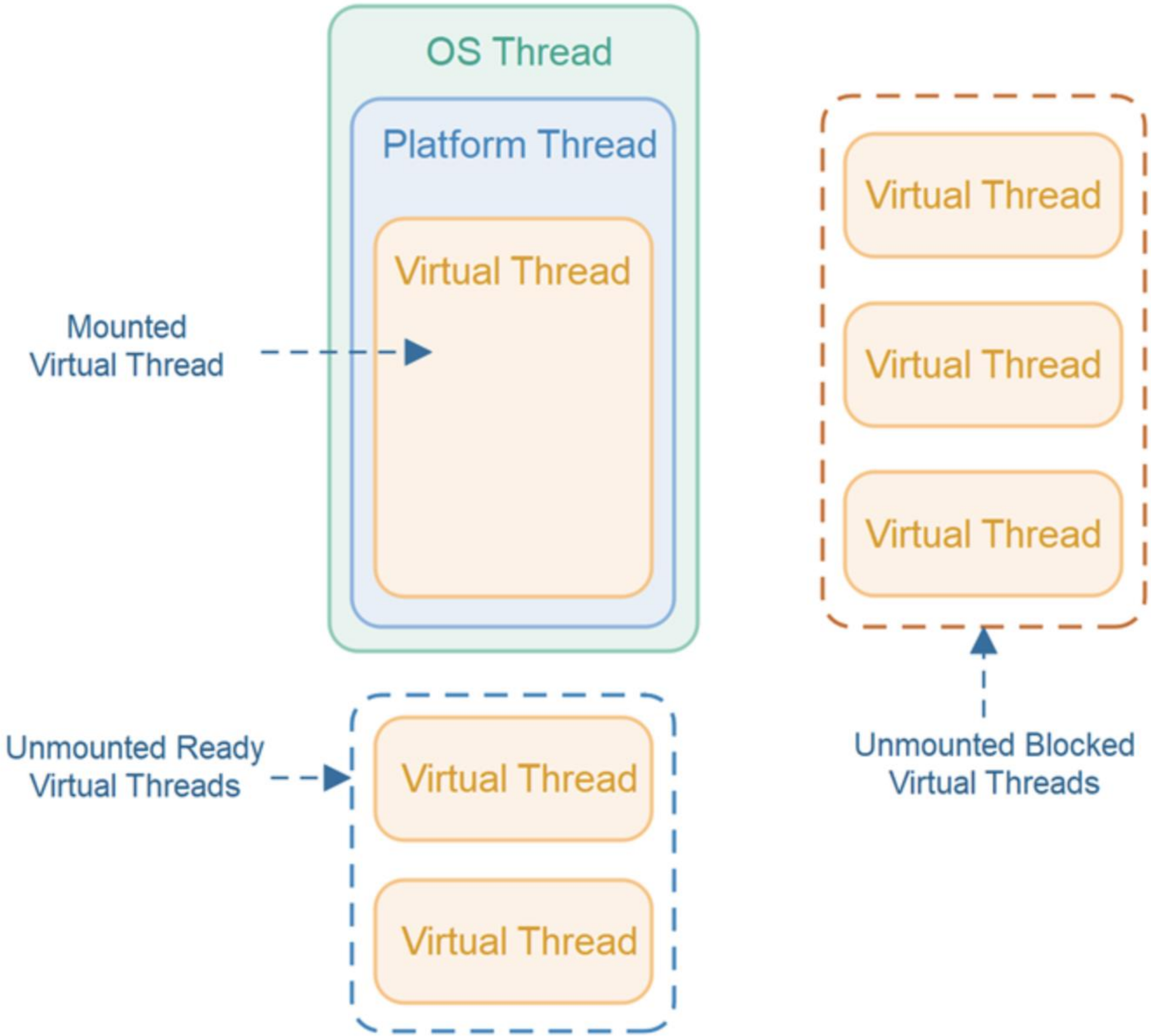
JAVA Virtual Threads - Details

» **virtual thread**

- reduce effort of writing high-throughput concurrent applications
- **thread-per-request** approach with almost optimal hardware utilization
- compatible with Thread API
- support debugging and profiling with existing tools
- stack frames in heap
  - stack size dynamically resizes as needed – expand and shrink
- OS still manages only platform threads
- **virtual thread** is **mounted** to **carrier thread** for execution
  - copy stack frames from heap to stack of carrier thread
  - **unmounted** when blocked for IO, lock or other resource
  - mounting/unmounting is invisible from Java code
  - thread dump, stack trace do not include carrier thread frames
  - carrier threads are from ForkJoinPool operating in FIFO mode
    - using number of available logical CPU cores

Cisco Confidential

JAVA Virtual Threads - Details



Cisco Confidential

JAVA Virtual Threads - Details

- **virtual thread API**
  - Thread::ofVirtual()
  - implementation of the ordinary Thread
    - Thread::currentThread() returns virtual thread, not carrier thread
    - ThreadLocal, interruption, stack walking works the same way
    - always daemon thread, Thread::setDaemon has no effect
    - priority cannot be changed
  - Executors.newVirtualThreadPerTaskExecutor()
    - each task run in own VirtualThread
- **scalability**
  - fast creation, small memory footprint
  - execution efficiency is the same as for platform threads
  - **scale for IO-bound workloads** (even for short-lived tasks)
    - simplified design with thread-per-request
    - suitable for server applications
  - no additional value for CPU-bound workloads

Cisco Confidential

JAVA Virtual Threads - Details

- example with 100k virtual threads

```
try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {
    IntStream.range(0, 100_000).forEach(i -> {
        executor.submit(() -> {
            Thread.sleep(Duration.ofSeconds(1));
            return i;
        });
    });
}
```

- after warm-up takes about 1.1 seconds
- with Executors.*newFixedThreadPool(1000)* it takes about 1000 seconds
- **drawbacks**
  - synchronized pins virtual thread to its carrier -> **use RentrantLock**
  - execution of JNI pins as well
  - release carrier only on blocking operation, **no preemption !**

Cisco Confidential