Server in Java

Adam Lider

Contents

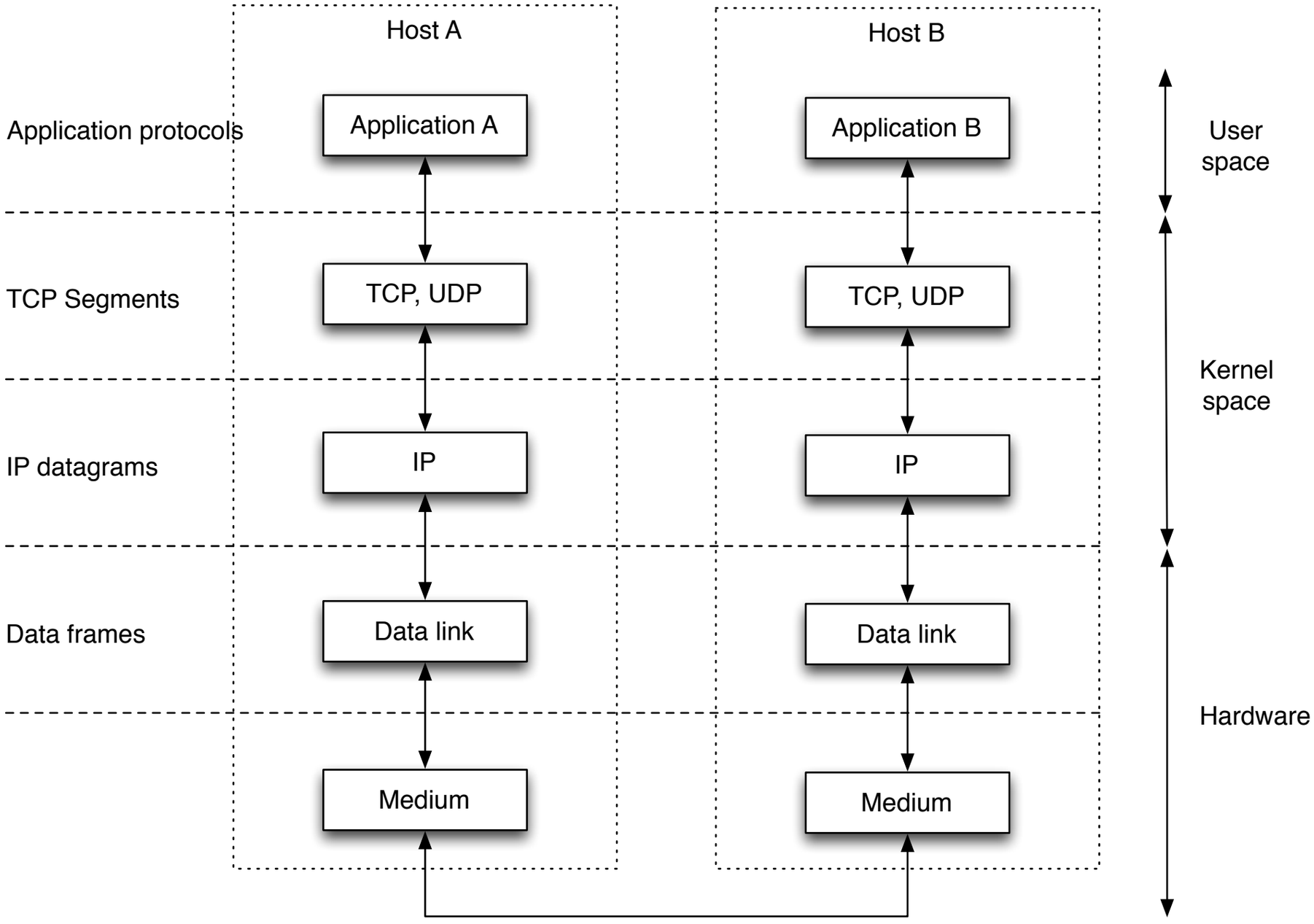
- Basics
- Custom protocol
- TCP details
- IO models
- Network architectures patterns

Basics

- Layered communication
- Establishing connections
- Exchanging data
- Closing connections

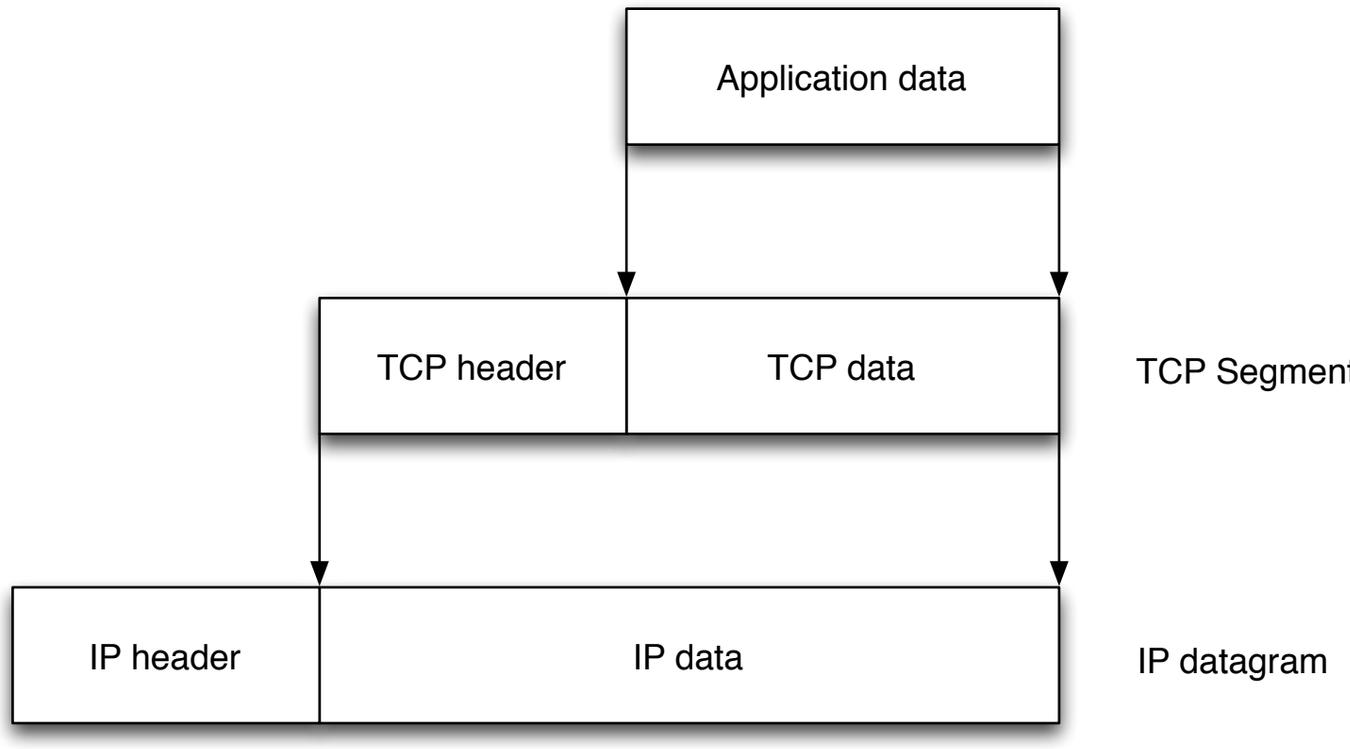
Basics

Layered communication



Basics

Layered communication - encapsulation



Basics

Establishing connections

Client

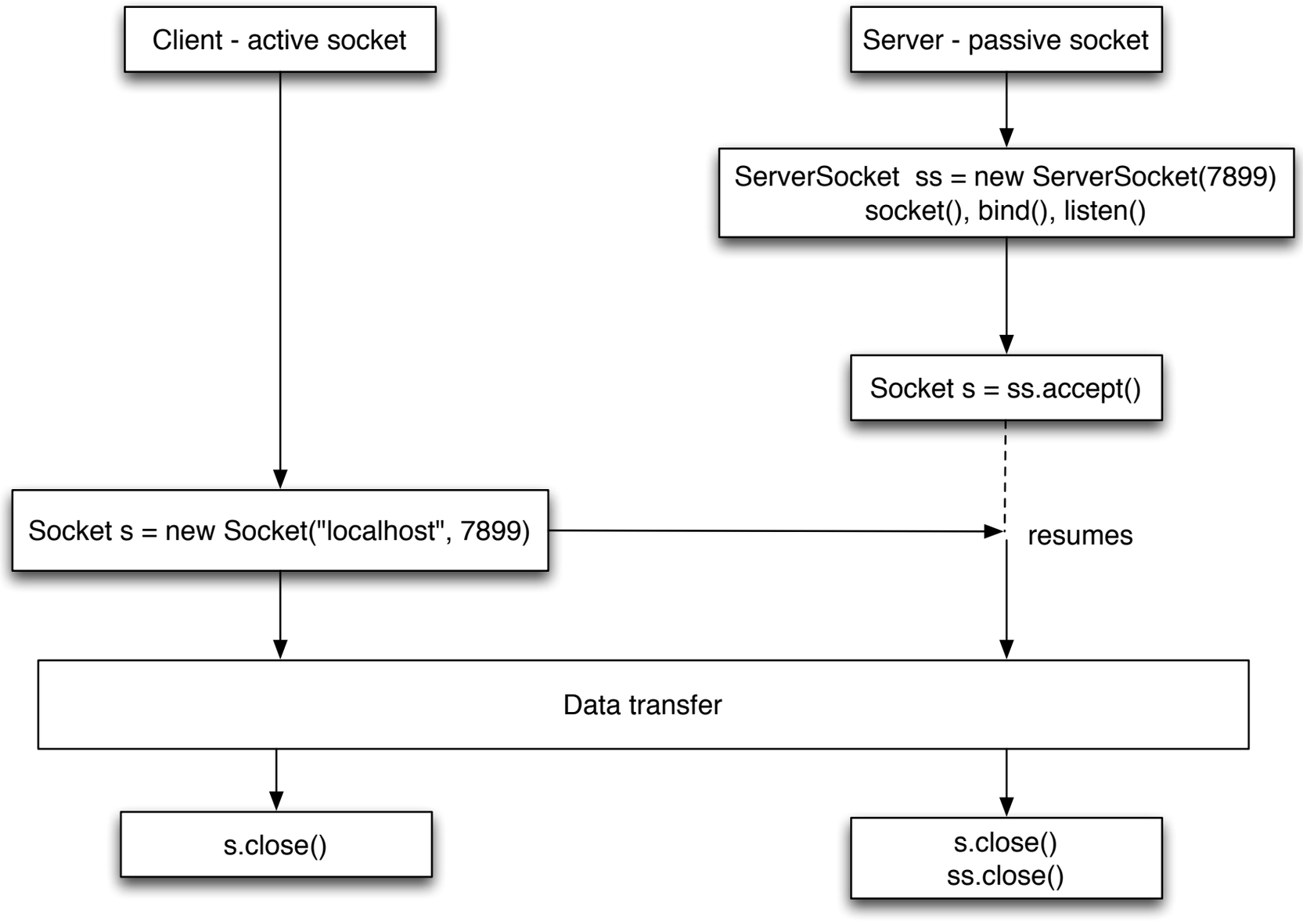
```
Socket s = new Socket("localhost", 7899);
OutputStream out = s.getOutputStream();
out.write(new byte[1024]);
s.close();
```

Server

```
ServerSocket ss = new ServerSocket(7899);
Socket s = ss.accept();
InputStream in = s.getInputStream();
byte[] buffer = new byte[1024 * 8];
int read = in.read(buffer);
s.close();
ss.close();
```

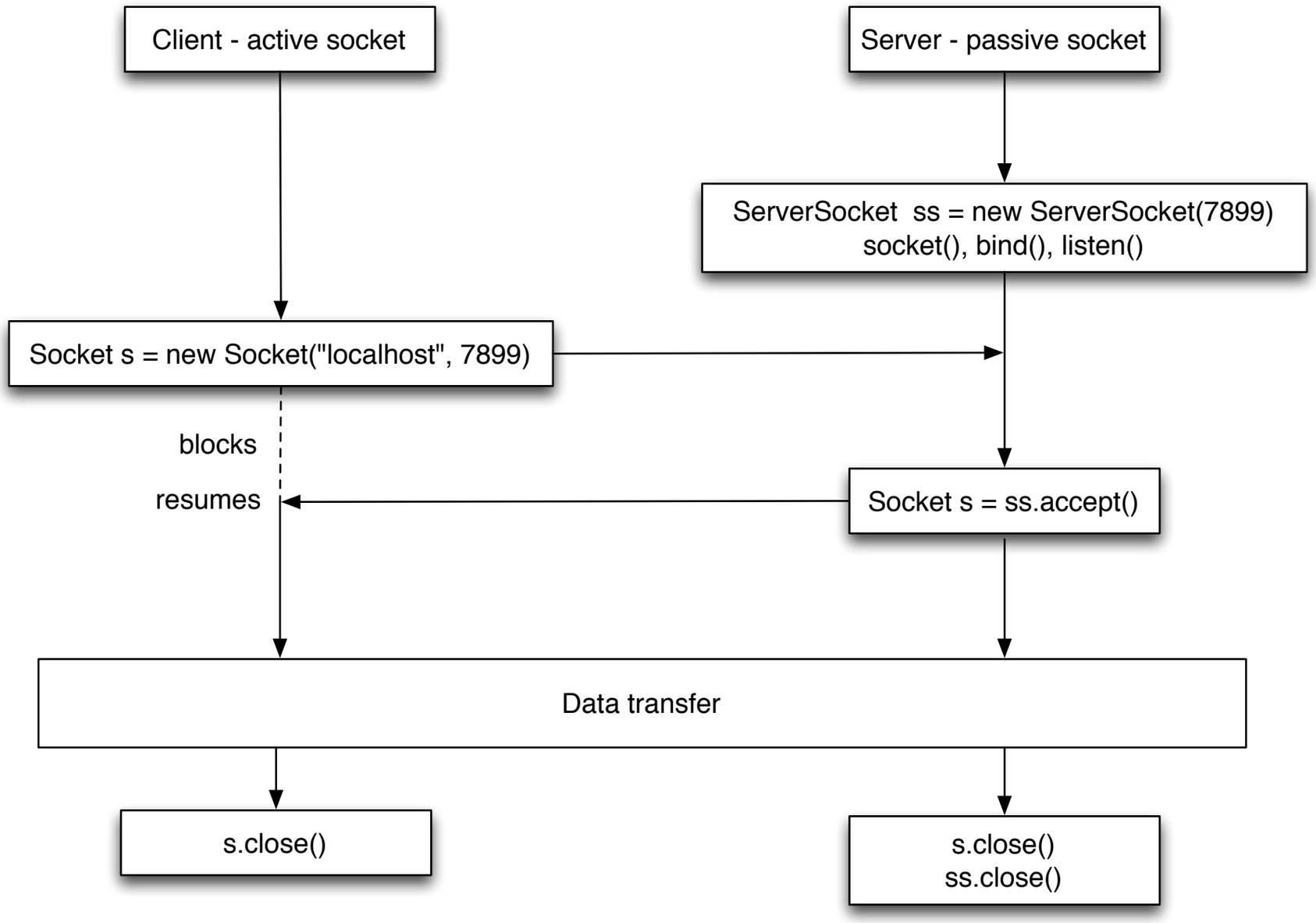
Basics

Establishing connections



Basics

Establishing connections - backlog



Java: default 50, new ServerSocket(7899, 400);
Linux: default 128, SOMAXCONN

Basics

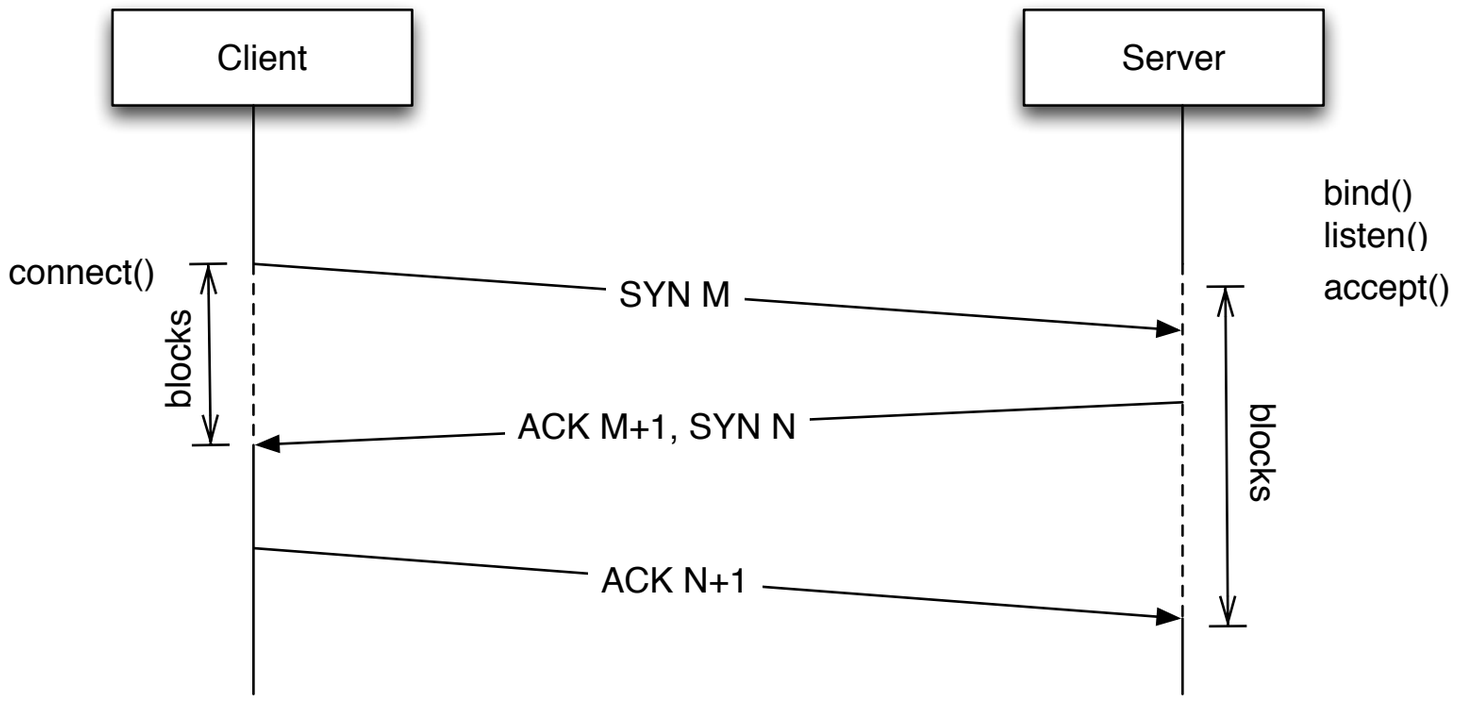
Establishing connections - three-way handshake

Client

```
Socket s = new Socket("localhost", 7899);
OutputStream out = s.getOutputStream();
out.write(new byte[1024]);
s.close();
```

Server

```
ServerSocket ss = new ServerSocket(7899);
Socket s = ss.accept();
InputStream in = s.getInputStream();
byte[] buffer = new byte[1024 * 8];
int read = in.read(buffer);
s.close();
ss.close();
```



Basics

Data transfer

Client

```
Socket s = new Socket("localhost", 7899);
OutputStream out = s.getOutputStream();
out.write(new byte[1024]);
s.close();
```

Server

```
ServerSocket ss = new ServerSocket(7899);
Socket s = ss.accept();
InputStream in = s.getInputStream();
byte[] buffer = new byte[1024 * 8];
int read = in.read(buffer);
s.close();
ss.close();
```

Basics

Data transfer

Client

```
Socket s = new Socket("localhost", 7899);
OutputStream out = s.getOutputStream();
out.write(new byte[1024]);
out.write(new byte[1024 * 2]);
out.write(new byte[1024 * 4]);
s.close();
```

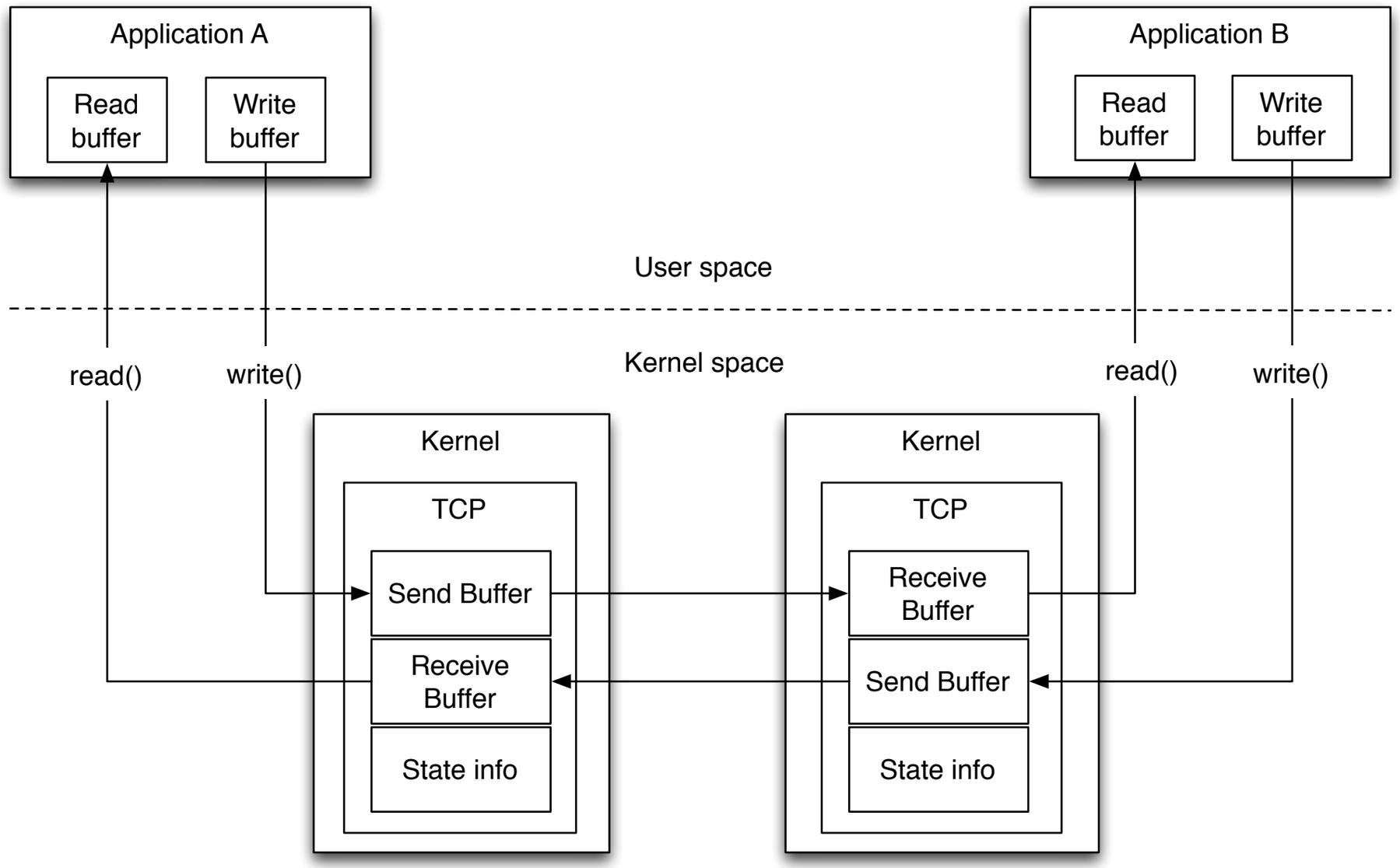
Server

```
ServerSocket ss = new ServerSocket(7899);
Socket s = ss.accept();
InputStream in = s.getInputStream();
byte[] buffer = new byte[1024 * 8];
int read = in.read(buffer);
s.close();
ss.close();
```

There is no any correspondence between writes performed at one end of connection and reads at the other end.

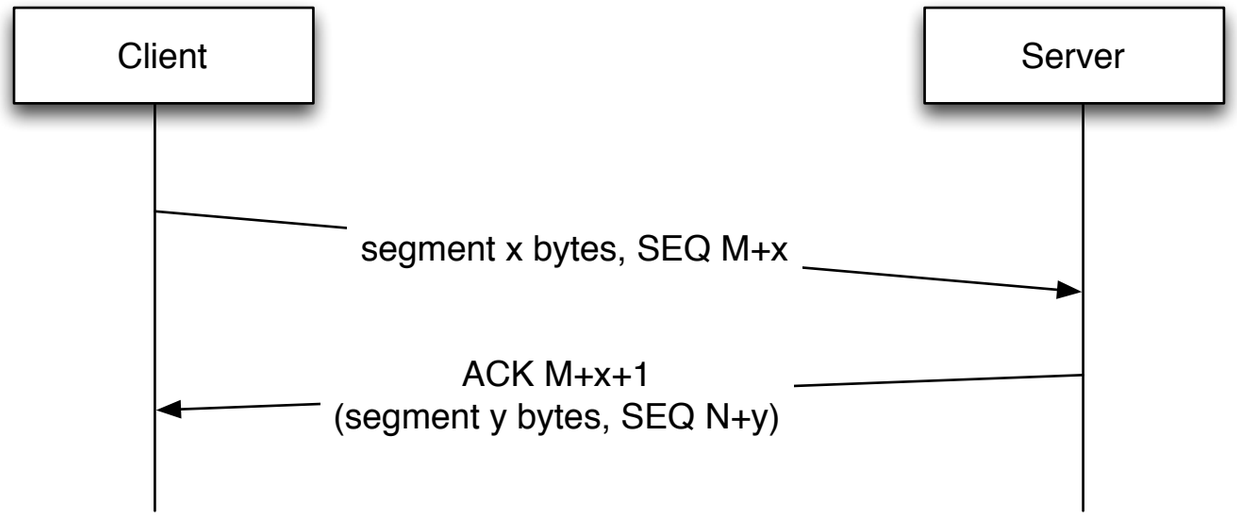
Basics

Data transfer



Basics

Data transfer



Basics

Closing connections

Client

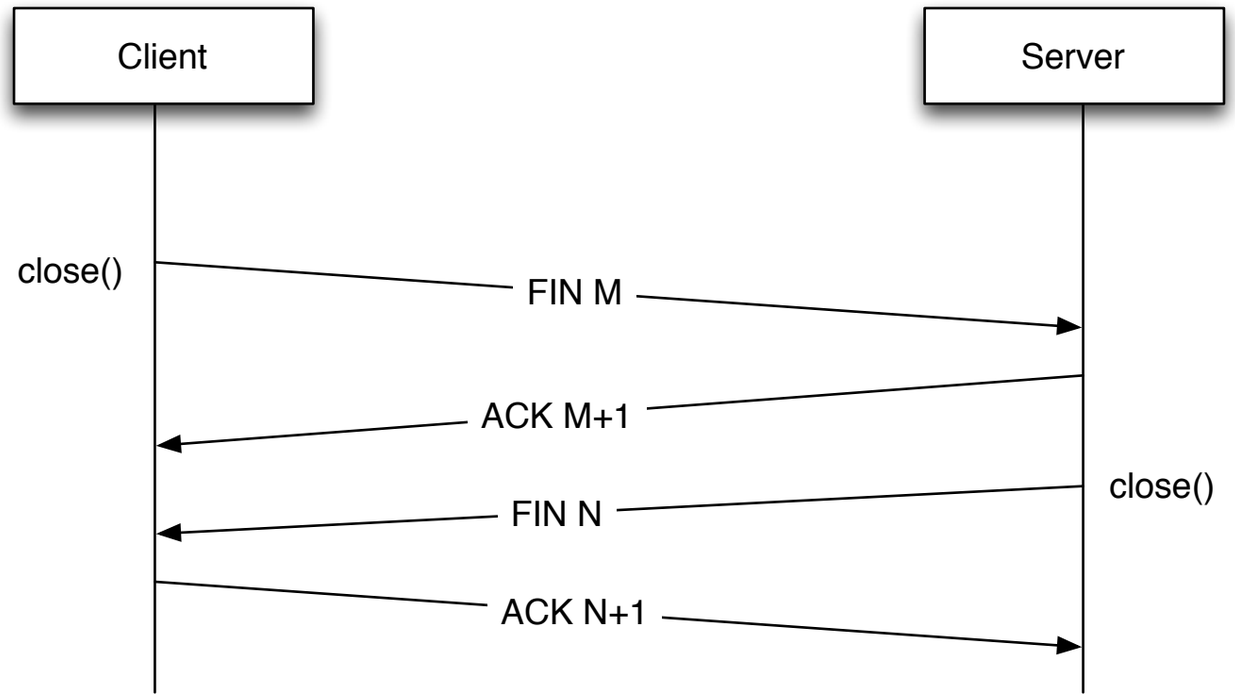
```
Socket s = new Socket("localhost", 7899);
OutputStream out = s.getOutputStream();
out.write(new byte[1024]);
s.close();
```

Server

```
ServerSocket ss = new ServerSocket(7899);
Socket s = ss.accept();
InputStream in = s.getInputStream();
byte[] buffer = new byte[1024 * 8];
int read = in.read(buffer);
s.close();
ss.close();
```

Basics

Closing connections



Custom protocol

Custom protocol

- Encodings
- Framing messages

Custom protocol

Encodings

- Integers
  - big vs little endian
  - signed vs unsigned
- Strings of text

All parties must agree on the representation for integer and strings of text

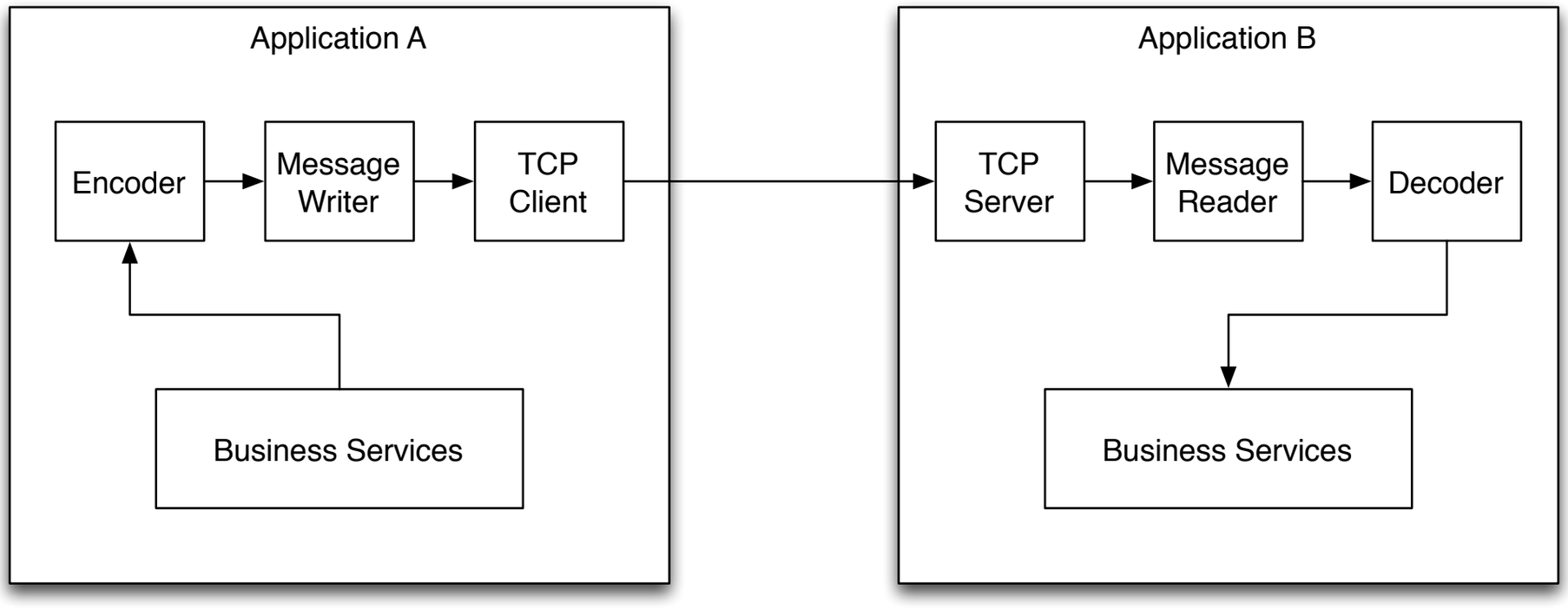
Custom protocol

Framing and parsing

- Separation of concerns
- Framing
  - Delimiter
  - Length

Custom protocol

Separation of concerns - Framing and parsing



TCP details

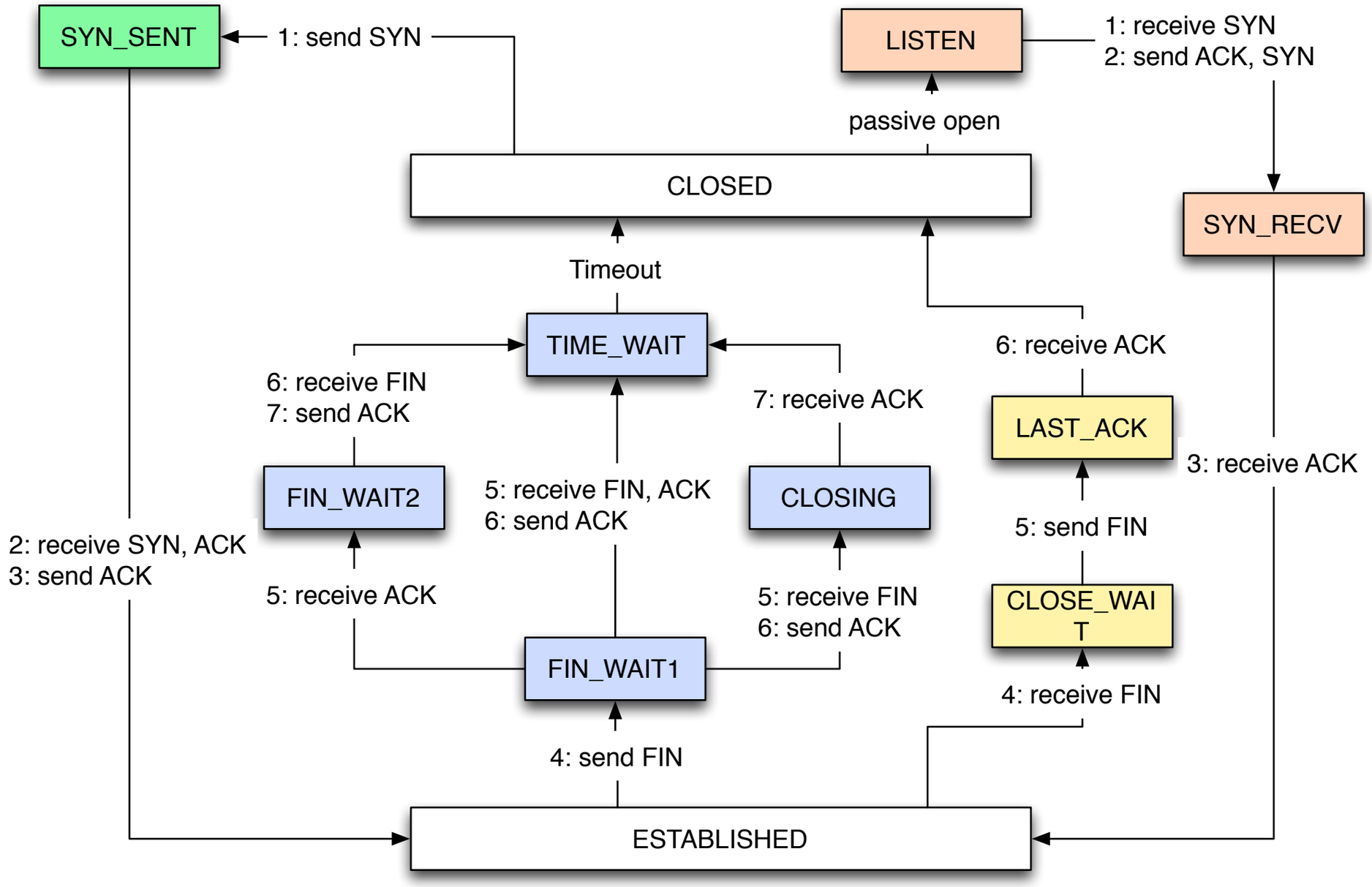
- State diagram
- Nagle's algorithm
- IP packet fragmentation
- Half-duplex connection termination
- Socket options

TCP details

- Ordered data (byte) transfer
- Retransmission of lost packets
- Error-free data transfer
- Flow control
- Congestion control

TCP details

State diagram



TCP details

Nagle's algorithm

- If there is sufficient data to compromise an entire TCP segment (data >= MSS) the send it immediately
- If there is no pending data in local buffers and no pending ACK of receipt from receiving end, then send it immediately
- If there is a pending ACK of receipt from receiving end and not enough data to compromise and entire TCP segment, then put the data into the local buffer

TCP details

IP packet fragmentation

- MSS - Maximum Segment Size
- MTU - Maximum transmission unit

TCP details

Half-duplex connection termination

Client

```
Socket s = new Socket("localhost", 7899);
OutputStream out = s.getOutputStream();
out.write(new byte[1024]);
out.write(new byte[2048]);
out.write(new byte[4096]);
out.close();
s.close();
```

Server

```
ServerSocket ss = new ServerSocket(7899);
Socket s = ss.accept();
OutputStream out = s.getOutputStream();
out.write("Hello".getBytes());

InputStream in = s.getInputStream();
byte[] buffer = new byte[1024 * 8];
int read = in.read(buffer);

s.close();
ss.close();
```

TCP details

Half-duplex connection termination

RFC 1122, section 4.2.2.13

“A host MAY implement a ‘half-duplex’ TCP close sequence, so that an application that has called CLOSE cannot continue to read data from the connection. If such a host issues a CLOSE call while received data is still pending in TCP, or if new data is received after CLOSE is called, its TCP SHOULD send a RST to show that data was lost.”

TCP details

shutdown output

Client

```
Socket s = new Socket("localhost", 7899);

OutputStream out = s.getOutputStream();
out.write(new byte[1024]);
out.write(new byte[2048]);
out.write(new byte[4096]);
s.shutdownOutput();

InputStream in = s.getInputStream();
// read data

s.close();
```

Server

```
ServerSocket ss = new ServerSocket(7899);
Socket s = ss.accept();

InputStream in = s.getInputStream();
// read data

OutputStream out = s.getOutputStream();
out.write("OK".getBytes());

s.close();
ss.close();
```

TCP details

Read and write deadlock

- Both ends read
- Both ends write

TCP details

Socket options

- TCP_NODELAY, setTcpNoDelay
- SO_REUSEADDR, setReuseAddress
- SO_RCVTIMEO, setSoTimeout ?
- SO_SNDTIMEO
- SO_SNDBUF, setSendBufferSize
- SO_RCVBUF, setReceiveBufferSize
- SO_LINGER, setSoLinger

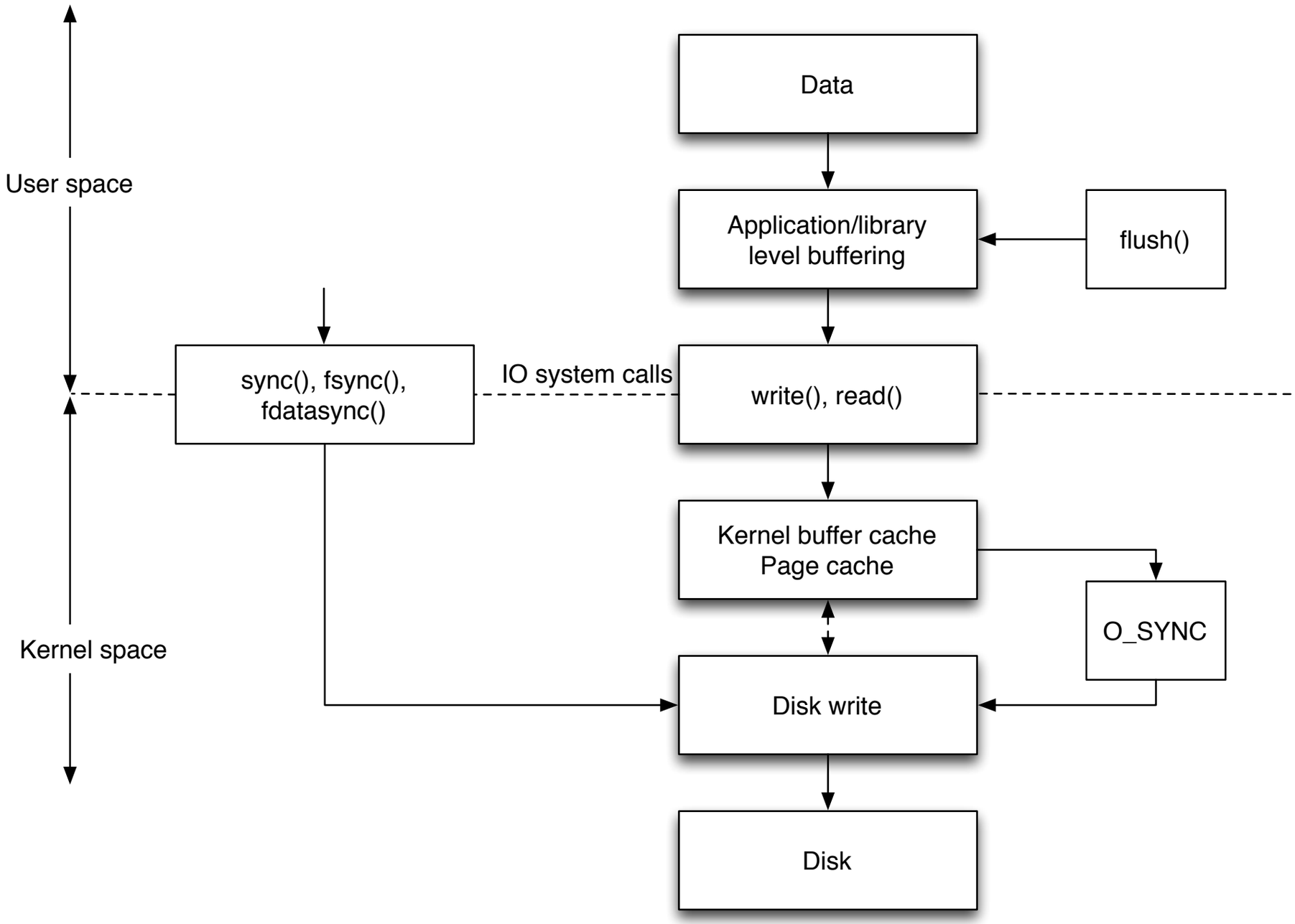
IO models

IO models

- File IO
- Blocking, non-blocking, synchronous
- Java IO, NIO, NIO.2

IO models

File IO



IO models

Non-blocking

- Sockets, pipes, terminals, FIFOs
- O_NONBLOCK
- Monitoring file descriptors:
  - select, poll
  - epoll
  - kqueue

IO models

Asynchronous

- libaio
- POSIAX AIO
- O_DIRECT

IO models

Java IO, NIO, NIO.2

- Java IO: Streams, blocking IO
- Java NIO:
  - Channels, Buffers
  - Blocking, non-blocking IO
- Java NIO.2: Asynchronous IO

IO models

Java NIO - non-blocking, sockets

```
RequestHandler handler = new RequestHandler();
long timeout = 1000;

Selector selector = Selector.open();

ServerSocketChannel listenChannel = ServerSocketChannel.open();
listenChannel.bind(new InetSocketAddress("localhost", 7789));
listenChannel.configureBlocking(false);
listenChannel.register(selector, SelectionKey.OP_ACCEPT);

for (;;) {
  if (selector.select(timeout) != 0) {
    continue;
  }
  Iterator<SelectionKey> selectedKeys = selector
                                          .selectedKeys()
                                          .iterator();
  while (selectedKeys.hasNext()) {
    SelectionKey key = (SelectionKey) selectedKeys.next();
    selectedKeys.remove();

    if (key.isAcceptable()) {
      handler.handleAccept(key);
    }
    if (key.isReadable()) {
      handler.handleRead(key);
    }
    if (key.isValid() && key.isWritable()) {
      handler.handleWrite(key);
    }
  }
}
```

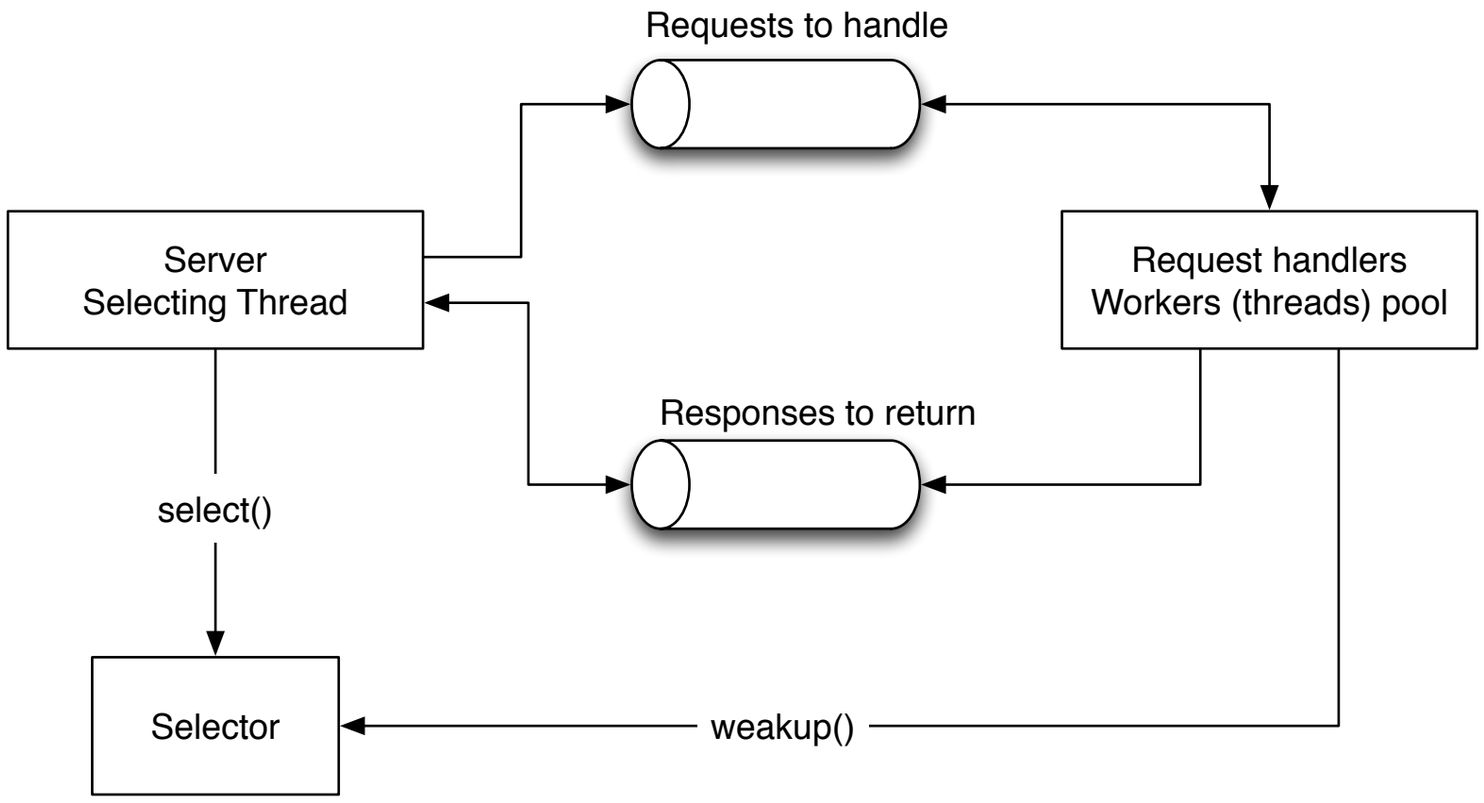
IO models

Java NIO - non-blocking, sockets

```
public static class RequestHandler {

  public void handleAccept(SelectionKey key) throws IOException {
    ServerSocketChannel serverChannel = (ServerSocketChannel) key.channel();
    SocketChannel clientChannel = serverChannel.accept();
    clientChannel.configureBlocking(false);
    clientChannel.register(
                    key.selector(),
                    SelectionKey.OP_READ,
                    ByteBuffer.allocate(1024));
  }

  public void handleRead(SelectionKey key) throws IOException {
    SocketChannel channel = (SocketChannel) key.channel();
    ByteBuffer buffer = (ByteBuffer) key.attachment();
    int read = channel.read(buffer);
    if (read == -1) {
      channel.close();
    } else {
      key.interestOps(SelectionKey.OP_READ | SelectionKey.OP_WRITE);
    }
  }

  public void handleWrite(SelectionKey key) throws IOException {
    SocketChannel channel = (SocketChannel) key.channel();
    ByteBuffer buffer = (ByteBuffer) key.attachment();
    buffer.flip();
    channel.write(buffer);
    if (!buffer.hasRemaining()) {
      key.interestOps(SelectionKey.OP_READ);
    }
    buffer.compact();
  }
}
```

IO models

Java NIO - non-blocking, sockets



IO models

Java NIO - non-blocking, sockets

- Not thread-safe selection keys
- OP_WRITE
- Old Sun Windows implementation problems:
  - OP_WRITE | OP_READ
  - Sockets not closed

IO models

Java NIO.2

- Start non-blocking I/O operations
- Provide notifications when I/O completes:
  - java.util.concurrent.Future<V> - pending result
  - CompletionHandler - complete result (callback)

IO models

Java NIO.2 - Threading

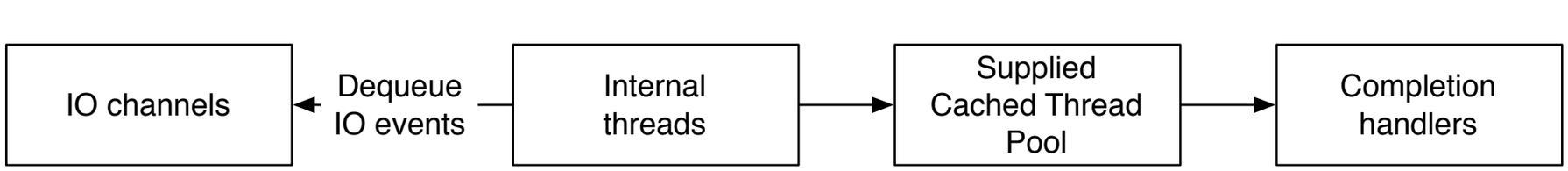
- Default Group
- Custom Groups
  - Fixed Thread Pool
  - Cached Thread Pool

IO models

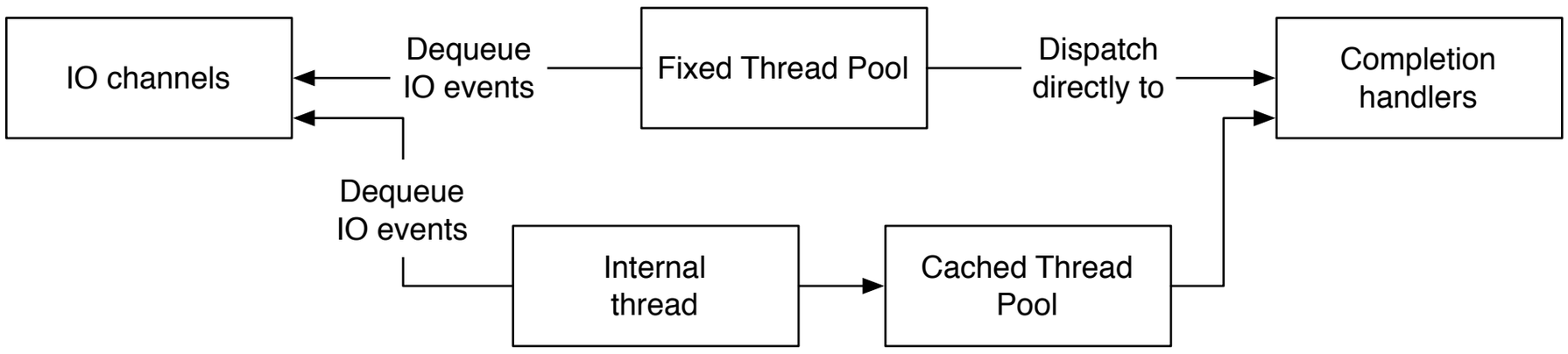
Java NIO.2 - Threading

Fixed thread pool

Cached thread pool



Default Group



IO models

Java NIO.2 - Issues

- Fixed Thread Pool
  - blocking completion handlers
- Cached Thread Pool
  - context switches
  - unbounded queuing needed, OOM
- ByteBuffer pool
- write() -> complete() -> write()

Network architecture patterns

Network architecture patterns

- Serial
- Process per connection
- Thread per connection
- Preforking
- Thread pool
- Evented (Reactor)
- Hybrids