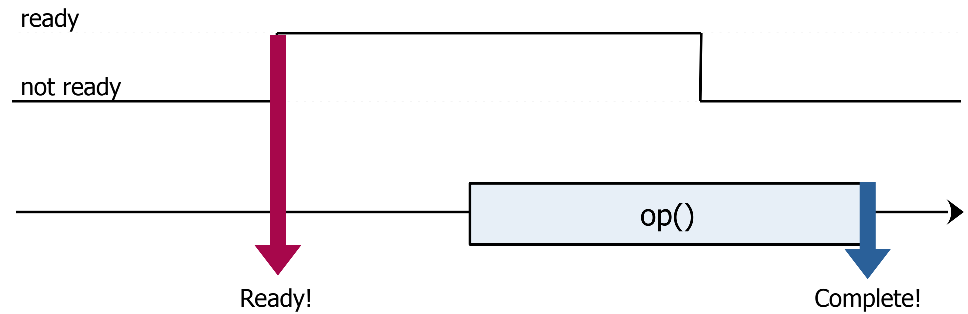
Event-driven I/O

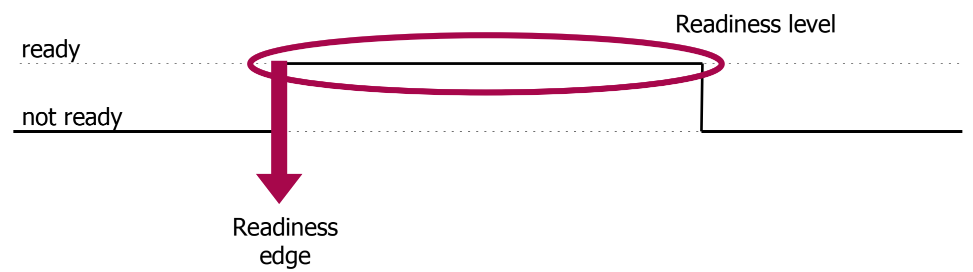
- Instead of waiting for each I/O event with a dedicated thread...
- ...let the operating system explicitly notify the application of an I/O event

Design issues

- Completion vs readiness



- Level-triggered vs edge-triggered



Implementation issues

- Managing the event set
  - User vs. kernel-level
  - Examples: select()/poll() vs epoll()
- Control transfer
  - Blocking thread vs signals vs busy polling
  - Examples: select() vs SIGIO vs DPDK/SPDK

Case studies in Java

- Asynchronous sockets (NIO2)
- Selectors (NIO)

Asynchronous I/O

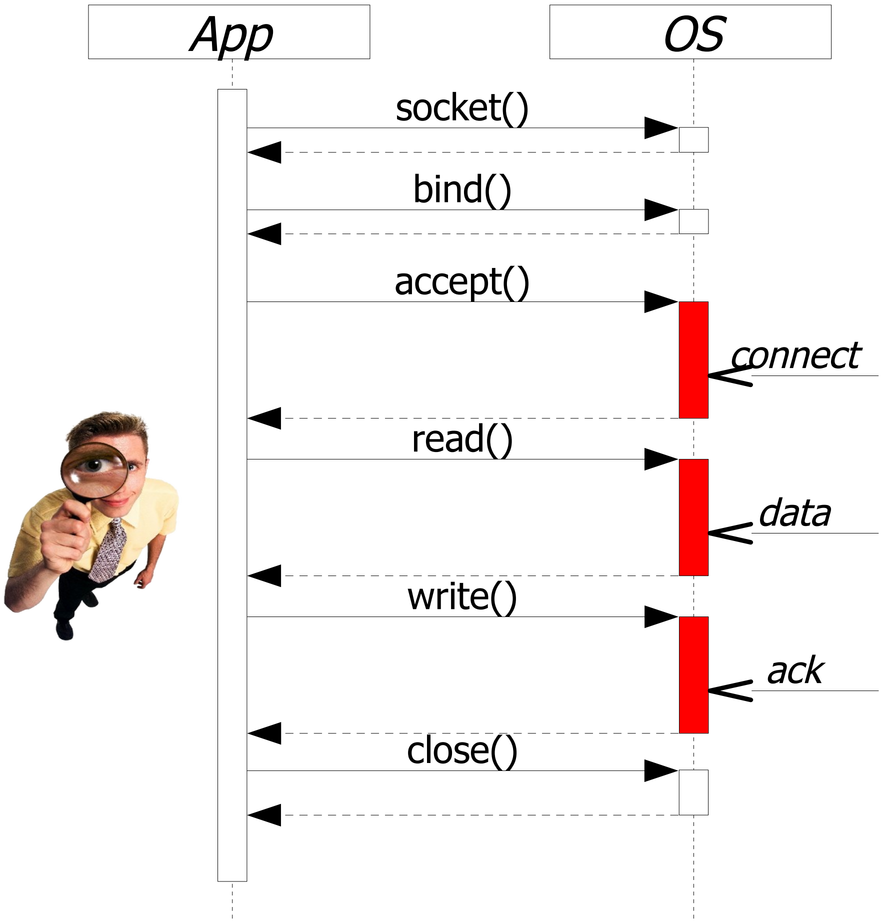
- For each blocking I/O operation, provide a callback to execute after the operation has completed
  - Completion event / edge-triggered
- General idea: Instead of:

```
read(buf); doSomething();
```

  do:

```
read(buf, ()->{ doSomething(); })
```

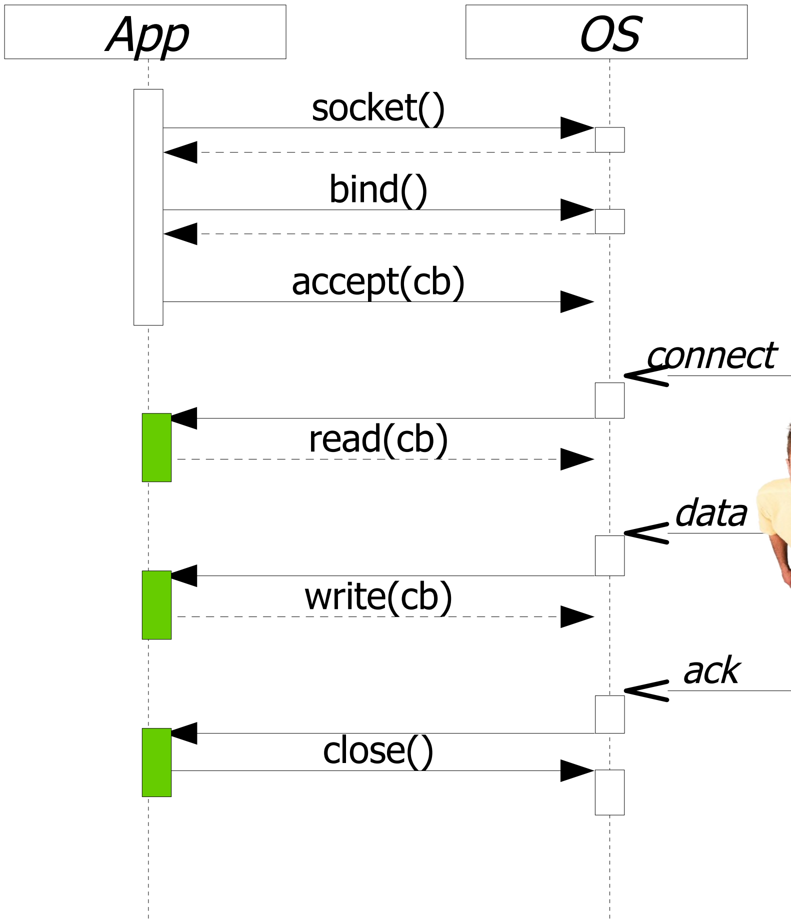
Threaded version




```
socket()

bind()
        wait for accept...

        event!
        connect socket
alloc buffer
        wait for read....
        event!
        copy data to buffer
flip
        wait for write ...
        event!
        copy data from buffer
buffer empty
close
```

code

Asynchronous version



socket()
bind()
code

wait for accept...

event!
connect socket
alloc buffer
code
wait for read....

event!
copy data to buffer
flip
code
wait for write ...
event!
copy data from buffer
buffer empty
close
code

Inversion of Control (IoC)

- With threads:
  - The program controls flow
  - Calls into the framework for specific tasks
- With events:
  - The framework controls flow
  - Calls back the program for specific tasks

Asynchronous I/O

- Avoids having a dedicated thread for each event source
- However:
  - Requires captive memory for idle I/O channels
  - Hides threading policy within the framework
- Available in Java with NIO2 AsynchronousSockets

Polled I/O

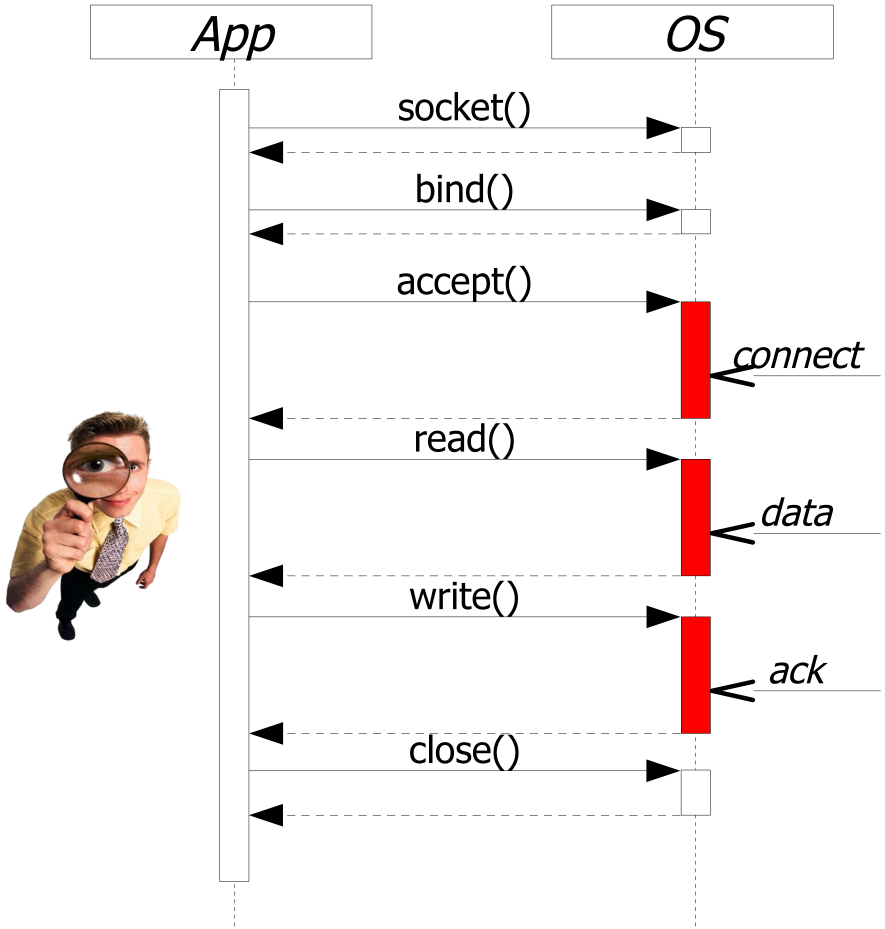
- Explicitly inform the application of which I/O channels are ready (and won't block)
  - Readiness event / level-triggered
- General idea: Instead of:

```
read(buf); doSomething();
```

- do:

```
for(key: select())
    read(buf); doSomething();
```

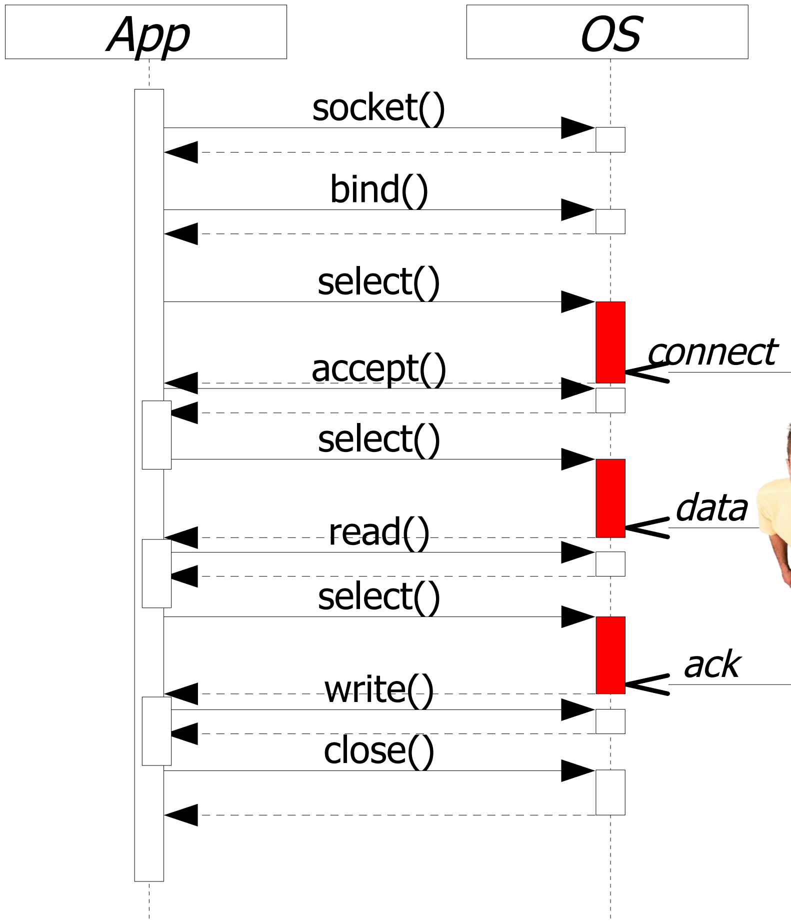
Threaded version

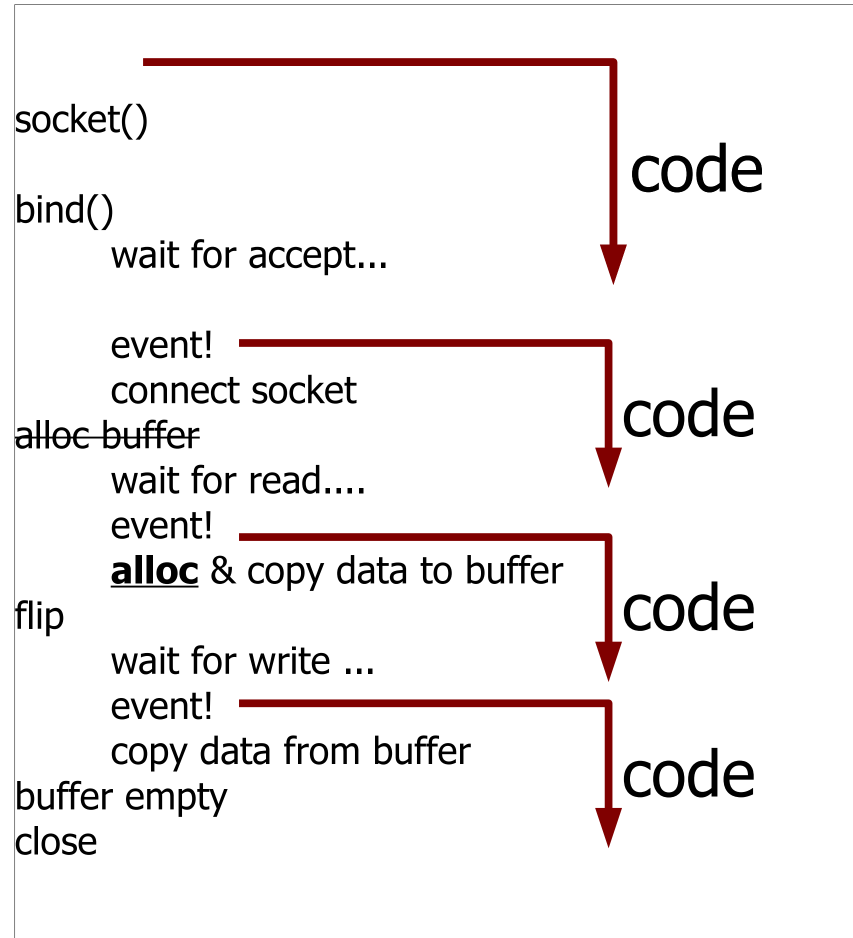



```
socket()

bind()
        wait for accept...

        event!
        connect socket
alloc buffer
        wait for read....
        event!
        copy data to buffer
flip
        wait for write ...
        event!
        copy data from buffer
buffer empty
close
```

code

Polled version





Polled I/O

- Avoids having a dedicated thread for each event source
- Avoids captive memory for idle I/O channels
- Makes threading policy explicit
- However:
  - Requires additional system calls (and copies)

- Polled I/O in Java with NIO Selectors

Polled I/O in Java

- Main loop:

```
Selector sel=SelectorProvider.provider().openSelector();

while(true) {
	sel.select();
	for(Iterator<SelectionKey> i=sel.selectedKeys().iterator(); i.hasNext(); ) {
		SelectionKey key = i.next();

		// i/o

		i.remove();
	}
}
```

Polled I/O in Java

- Register interest in server socket:

```
ServerSocketChannel ss=ServerSocketChannel.open();
ss.bind(new InetSocketAddress(12345));
ss.configureBlocking(false);
ss.register(sel, SelectionKey.OP_ACCEPT);
```

- Handle connection event:

```
if (key.isAcceptable()) {
    SocketChannel s=ss.accept();

    s.configureBlocking(false);
    s.register(sel, SelectionKey.OP_READ);
}
```

Polled I/O in Java

```
if (key.isReadable()) {
    ByteBuffer buf=ByteBuffer.allocate(100);
    SocketChannel s=(SocketChannel)key.channel();

    int r=s.read(buf);
    if (r<0) {
        key.cancel();
        s.close();
    } else {
        buf.flip();
        for(Socket r: ..) {
            r.write(buf);
            buf.rewind();
        }
    }
}
```

What if write blocks?

Polled I/O in Java

- Need to poll before writing
- Bytes read must be saved until writing is possible
- Signal interest on writing

```
if (key.isReadable()) {
        ...
        } else {
                buf.flip();
                for(SelectionKey k: ...) {
                        key.attach(buf.duplicate());
                        key.interestOps( ... | SelectionKey.OP_WRITE);
                }
        }
}
```

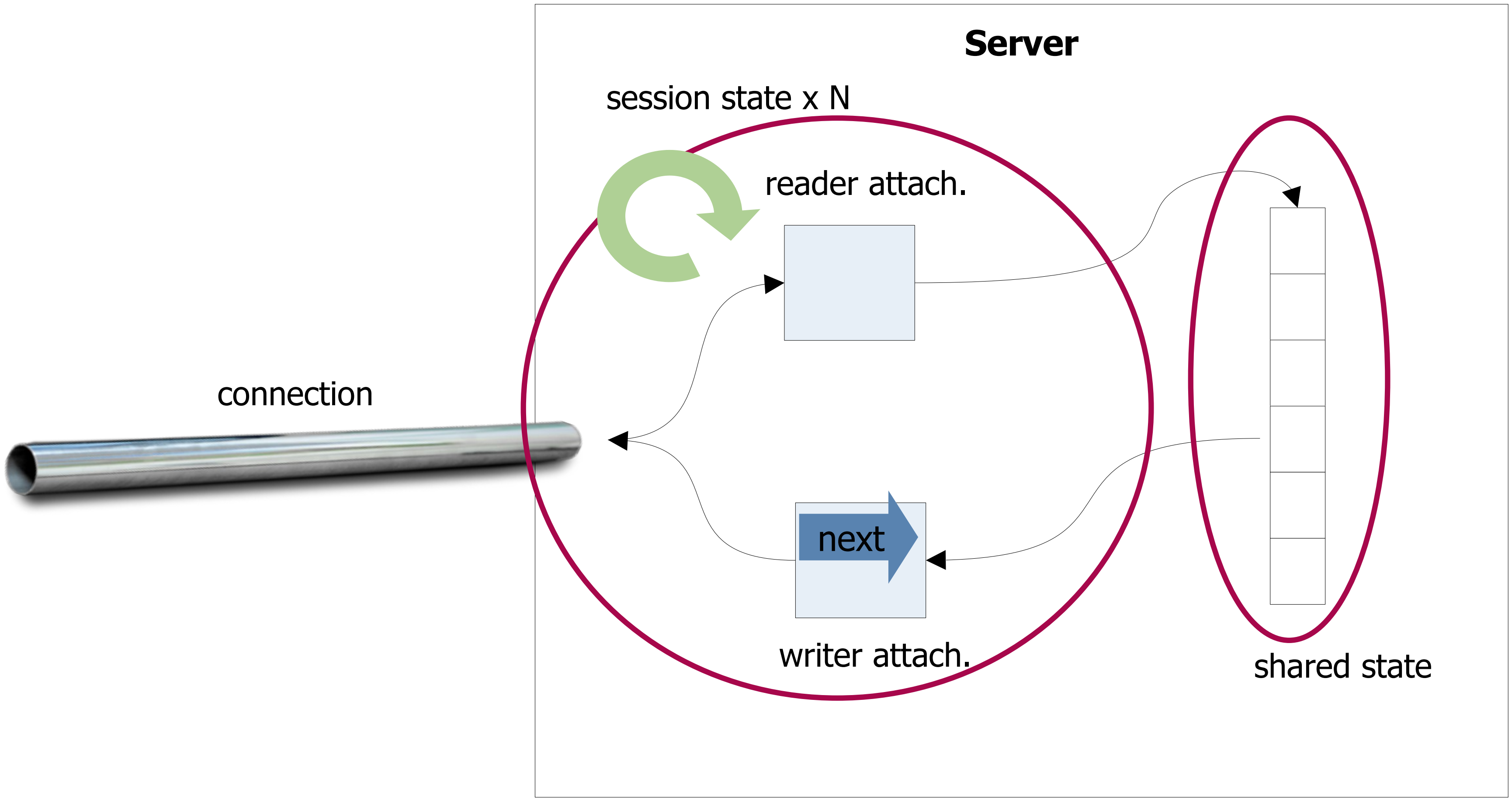
What if multiple writes pending?

Polled I/O in Java

- Get bytes attached to key
- Reset interest to reading

```
if (key.isWritable()) {
        SocketChannel s=(SocketChannel)key.channel();
        ByteBuffer buf=(ByteBuffer)key.attachment();

        s.write(buf);
        key.interestOps(SelectionKey.OP_READ);
}
```

Server architecture



Summary

- Memory:
  - No data copying by reference aliasing
  - Reduced allocation by avoiding captive buffers
- Event-driven programs:
  - A single shallow stack
  - Minimal context switching
  - Explicit scheduling and queuing (can be purged)

References

- Michael Kerrisk. *The Linux Programming Interface - A Linux and UNIX® System Programming Handbook.* No Starch Press, 2010.
  - Chap. 63.
- Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces.* Arpaci-Dusseau Books, 2018.
  - Chap. 33: https://pages.cs.wisc.edu/~remzi/OSTEP/threads-events.pdf