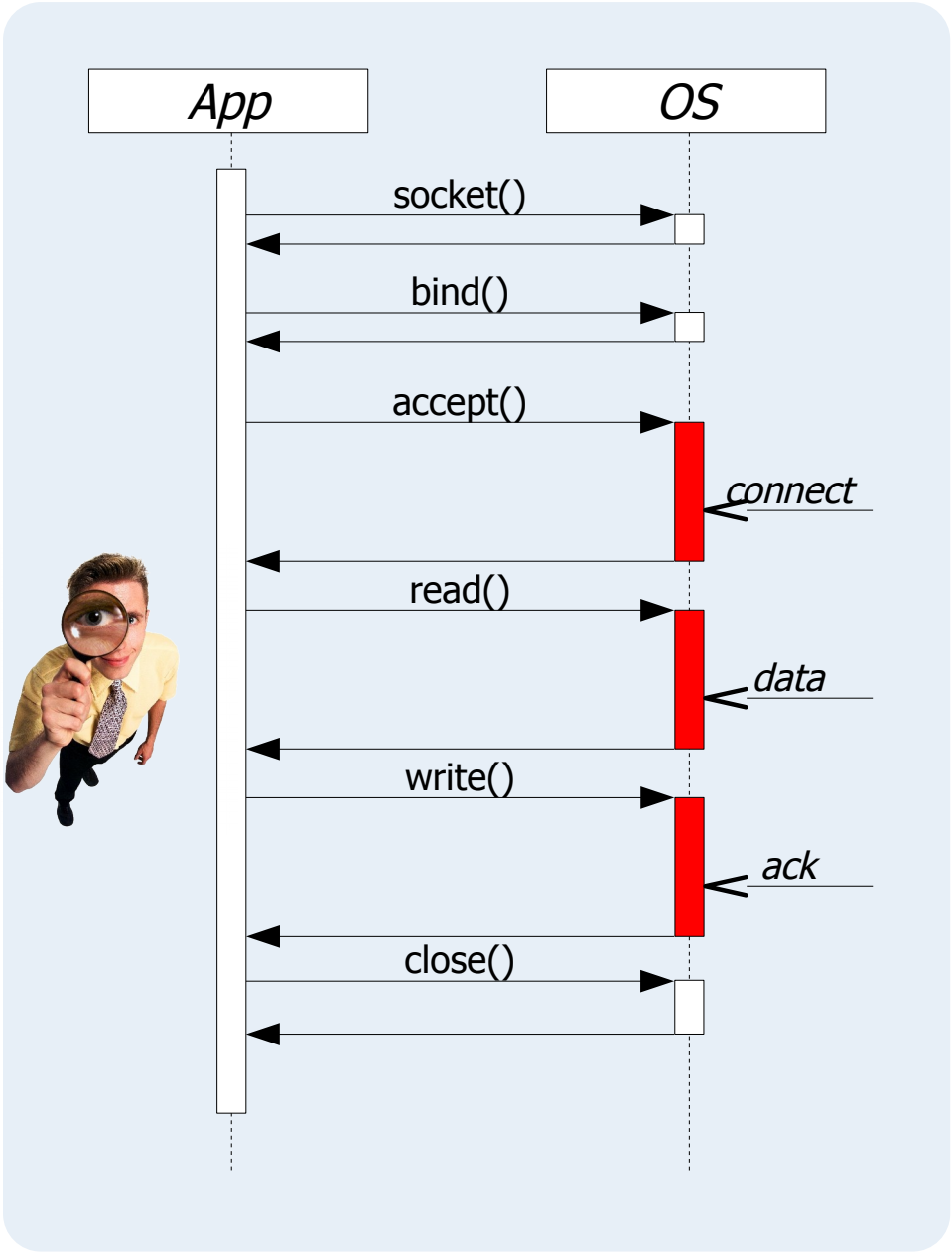
Abstract Execution

```
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
```

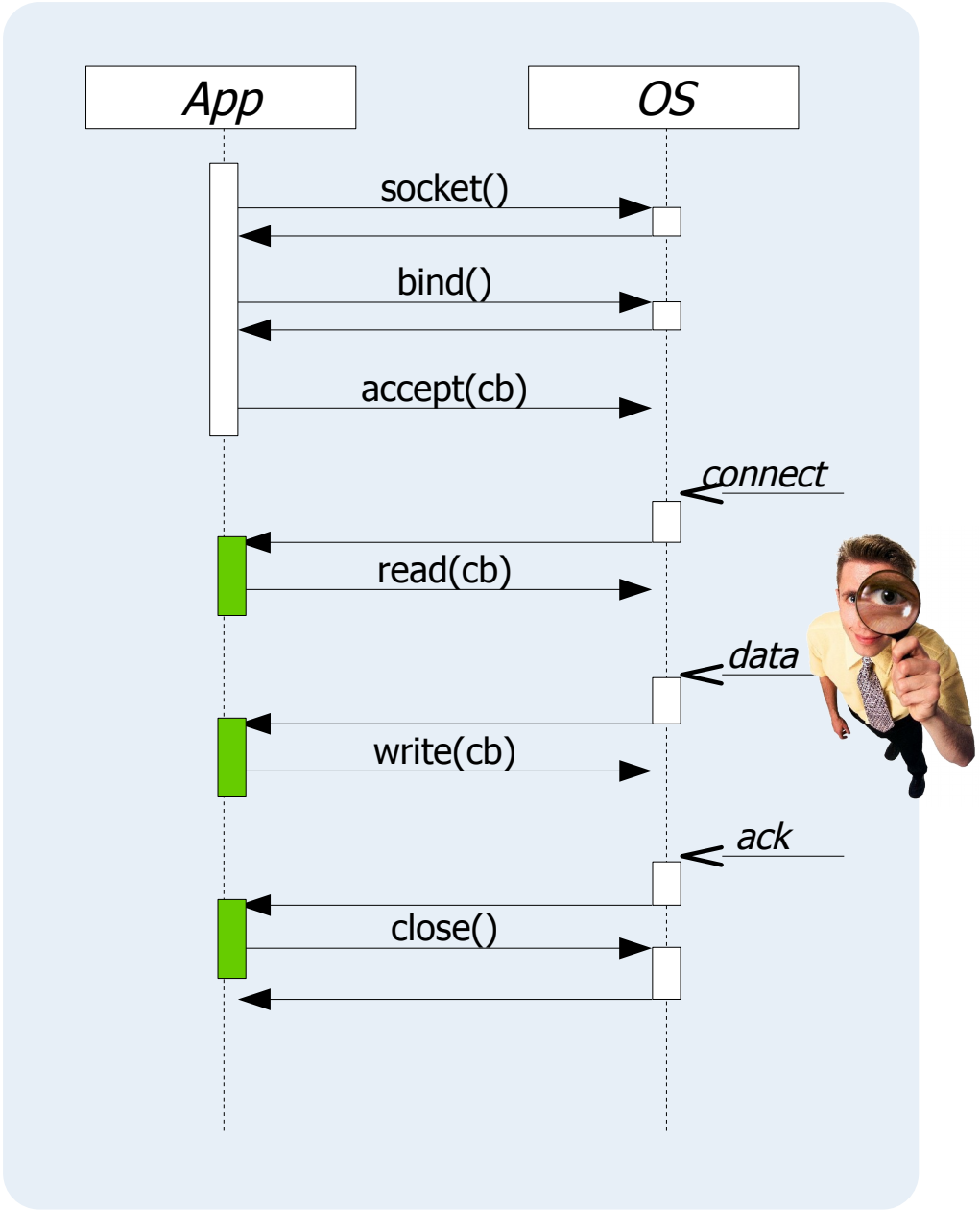
Threaded version

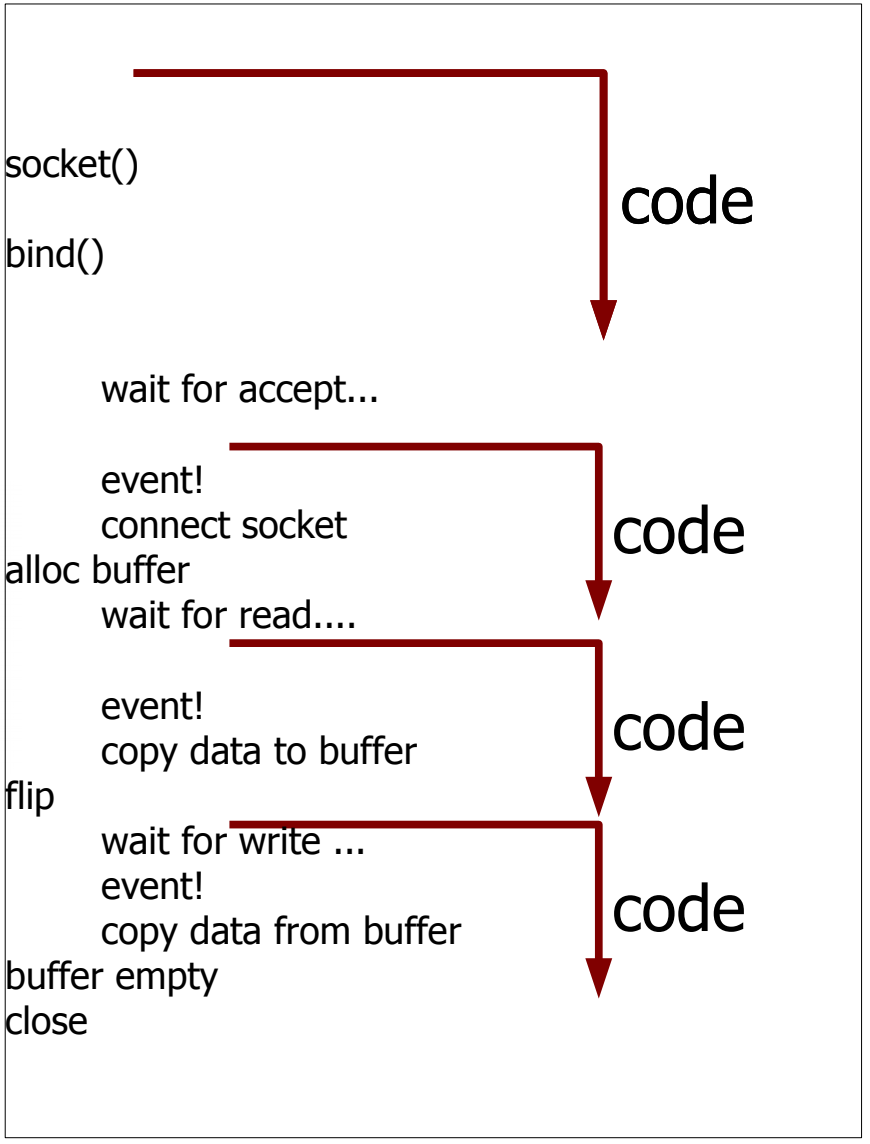



```
socket()

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

Event-driven version





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

Conclusions

- Prefer event-driven when:
  - Multiple inputs are handled concurrently and independently of the current state
- Prefer threaded when:
  - A single input is handled at each time
  - The input is handled differently depending on the current state

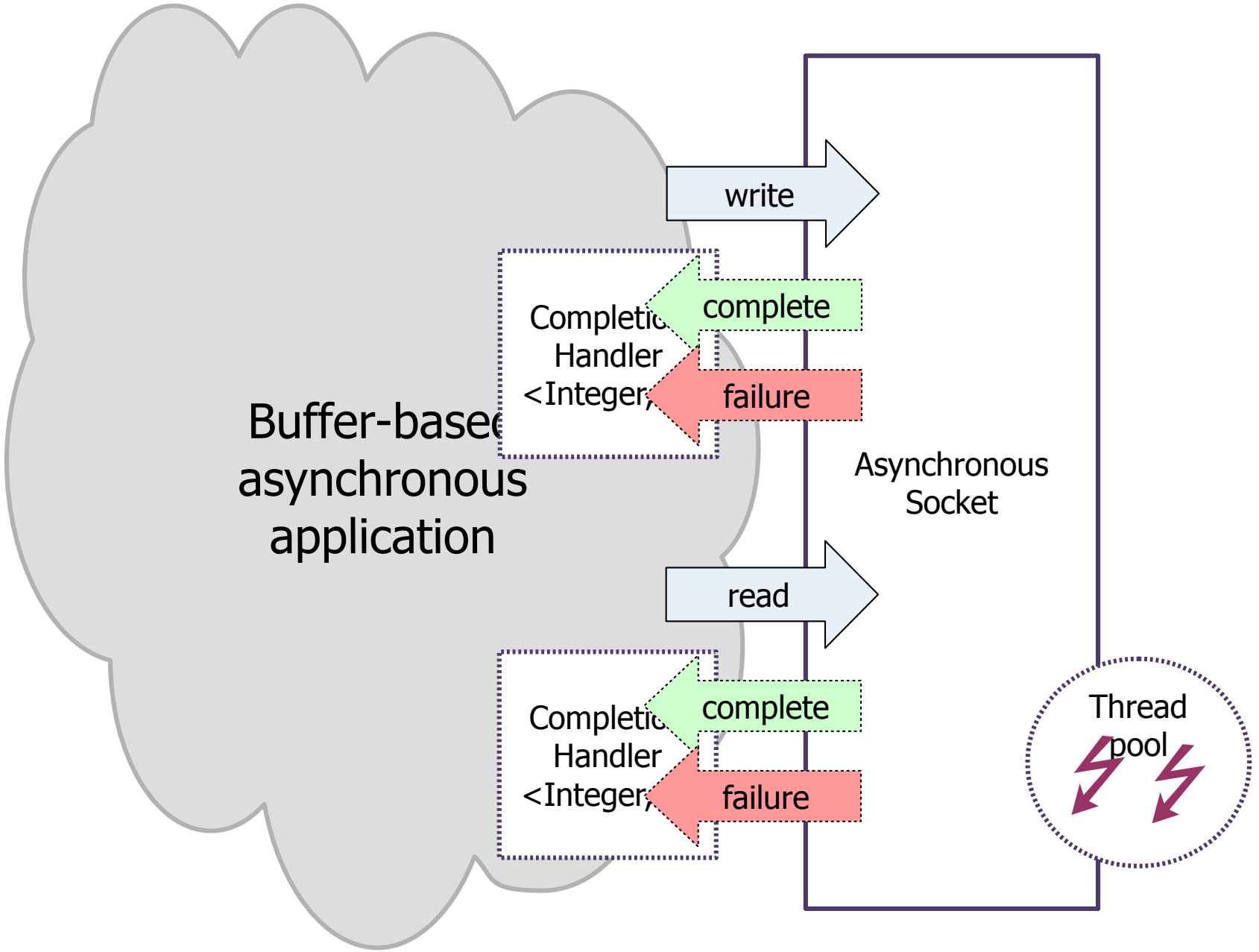
Conclusions

- Memory:
  - No data copying by pointing into the same buffers
  - Reuse and sharing reduces allocation
- Event-driven programs:
  - A single shallow stack
  - Minimal context switching
  - Explicit scheduling and queuing (can be purged)

Case study

- Improve the chat server with:
  - Work with lines, not buffers
  - Validate login and password
- Assess impact on:
  - Ease of use
  - Reuse and composition (incl. w/ threaded code)

Buffer-based application



Asynchronous line buffer

```
public class AsynchronousLineBuffer {

    private AsynchronousSocketChannel sock;

    private CompletionHandler<String, Object> rHandler;
    private Object rValue;

    public <A> void read(final A value, CompletionHandler<String, A> handler) {
        ...
    }

    private CompletionHandler<Void, Object> wHandler;
    private Object wValue;

    public <A> void write(String line, final A value,
                          CompletionHandler<Void, A> handler) {
        ...
    }
}
```

String-buffer layer

- On reading, gathers data and de-serializes
- On writing, serializes and flushes data
- Can be generalized to any object, by changing the serialization code (e.g., protobuf)

String-based application