Distributed Systems Paradigms

José Orlando Pereira

Departamento de Informática
Universidade do Minho

Motivation

- Handle a large number of clients and requests with a single server
- The “c10k problem” in 1999:
  - http://www.kegel.com/c10k.html
- Examples:
  - financial, games, ...
  - notifications in mobile apps
  - machine-to-machine (M2M)

Case study

- Simple chat server:
  - Forward all messages to all clients
- Simplest form of multi-user server-based application:
  - Input from clients
  - Internal state and logic
  - Output to clients triggered by input
- Consider:
  - Large number of clients
  - Slow connections

First threaded solution

- For each connection:
  - Handler thread
- When reading, write to all other connections
- Use buffering:
  - At user level (streams): To minimize system calls
  - In the kernel (socket): To cope with slow readers

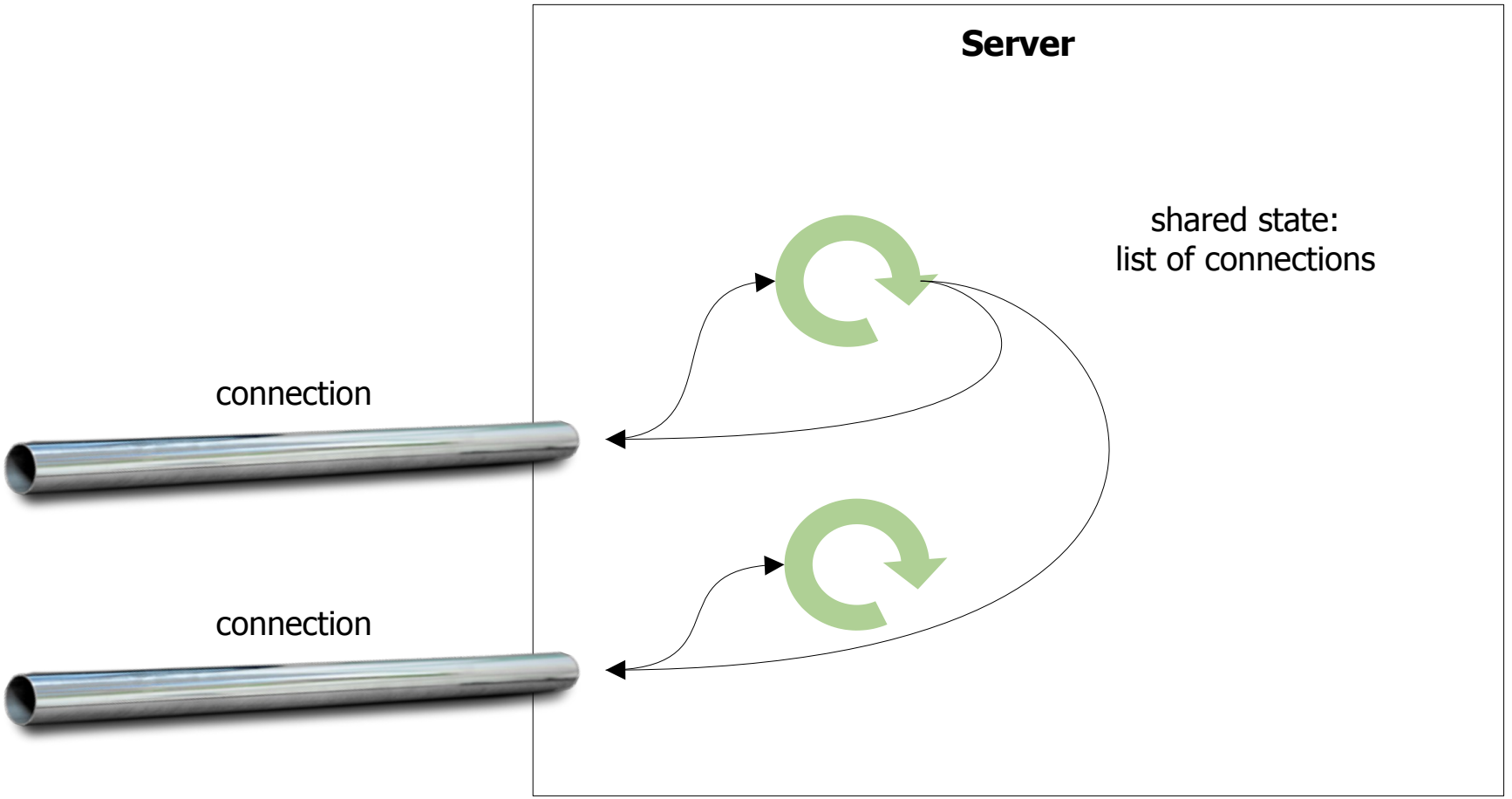
Sockets in java.net

```
ServerSocket ss=new ServerSocket(12345);

while(true) {
      Socket s=ss.accept();

      // start handler thread

      s.close();
}
```

Buffers in java.net

```
InputStream is=new BufferedInputStream(s.getInputStream());
OutputStream os=new BufferedOutputStream(s.getOutputStream());
while(true) {
        // i/o
        is.read(...)
        for(var os: connections) {
                os.write(...);
        }
}
```

Firsth threaded solution

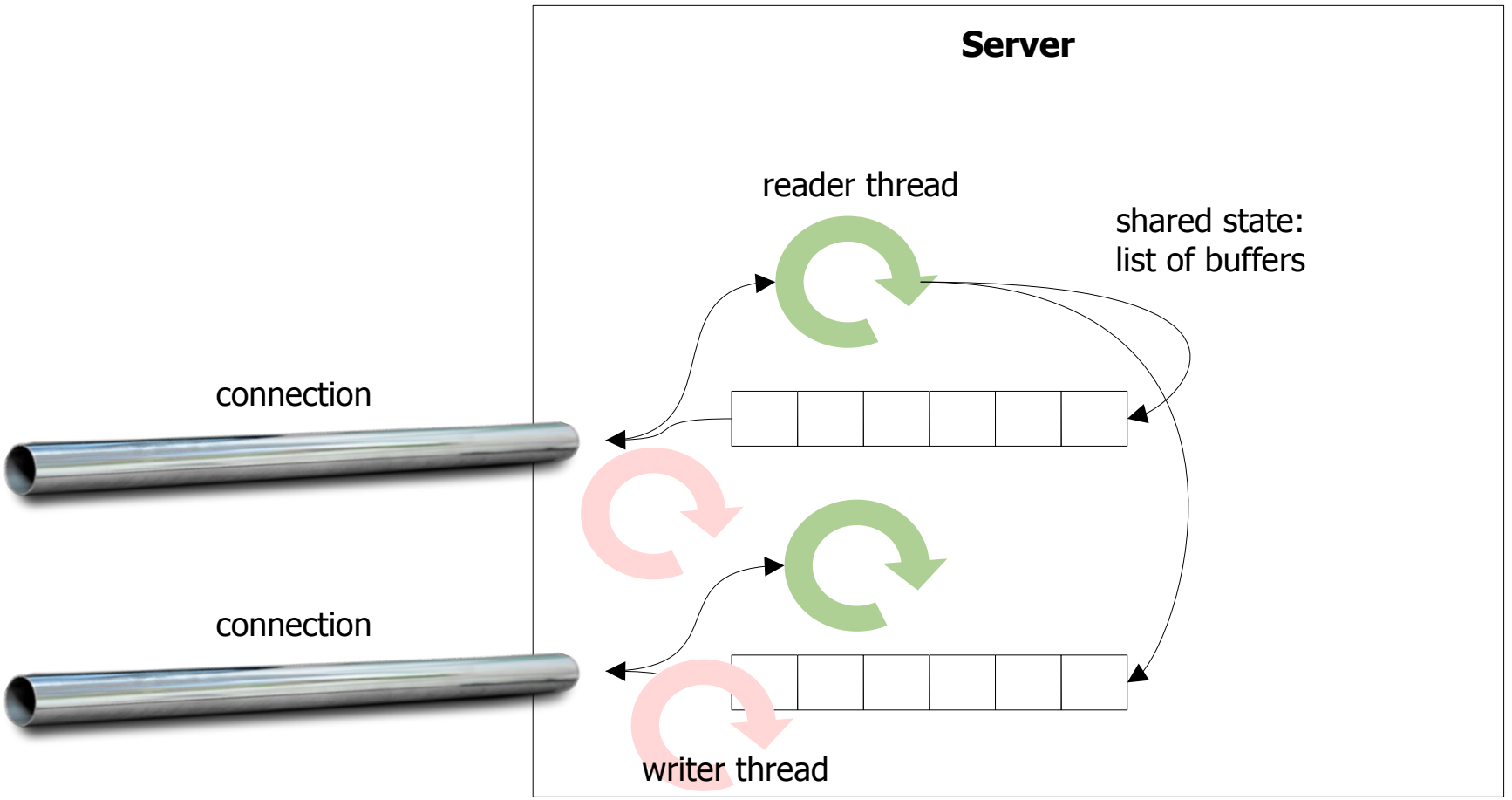


Buffers in java.net

```
InputStream is=new BufferedInputStream(s.getInputStream());
OutputStream os=new BufferedOutputStream(s.getOutputStream());
while(true) {
        // i/o
        is.read(...)
        for(var os: connections) {
                os.write(...);
                os.flush();
        }
}
```

What if flush blocks?

Writer thread



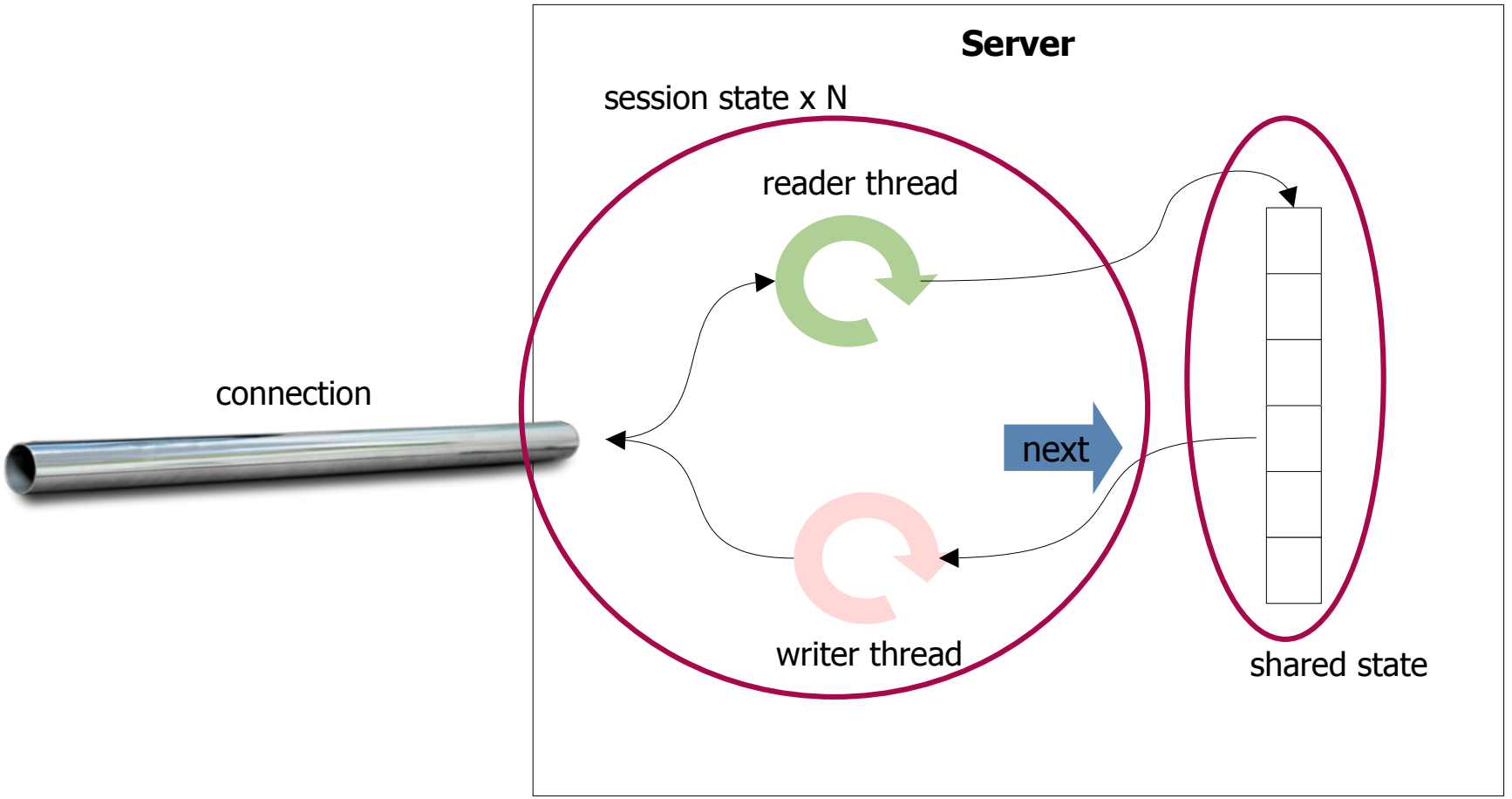
Memory

- Memory: *n* connections x messages in transit (~ n^2)
  - Caused by data copying in stacked abstractions
    - Serialization!
  - Overhead in allocation and garbage collection

Second threaded solution

- Keep a shared queue
- For each connection:
  - Reader thread + Pointer into queue + Writer thread
- When reading, insert in shared queues and notify writer threads
- When writing, advance pointer
- Lazily, remove prefix from shared queue

Server architecture



Sockets in java.nio

```
ServerSocketChannel ss=ServerSocketChannel.open();
ss.bind(new InetSocketAddress(12345));

while(true) {
      SocketChannel s=ss.accept();

      // start i/o threads
}
```

Sockets in java.nio

Explicit allocation of byte buffers

```
ByteBuffer buf=ByteBuffer.allocate(100);

s.read(buf);
buf.flip();
```

```
for(SocketChannel r: receivers) {
      r.write(buf.duplicate());
}
buf.clear();
```

Shared buffers

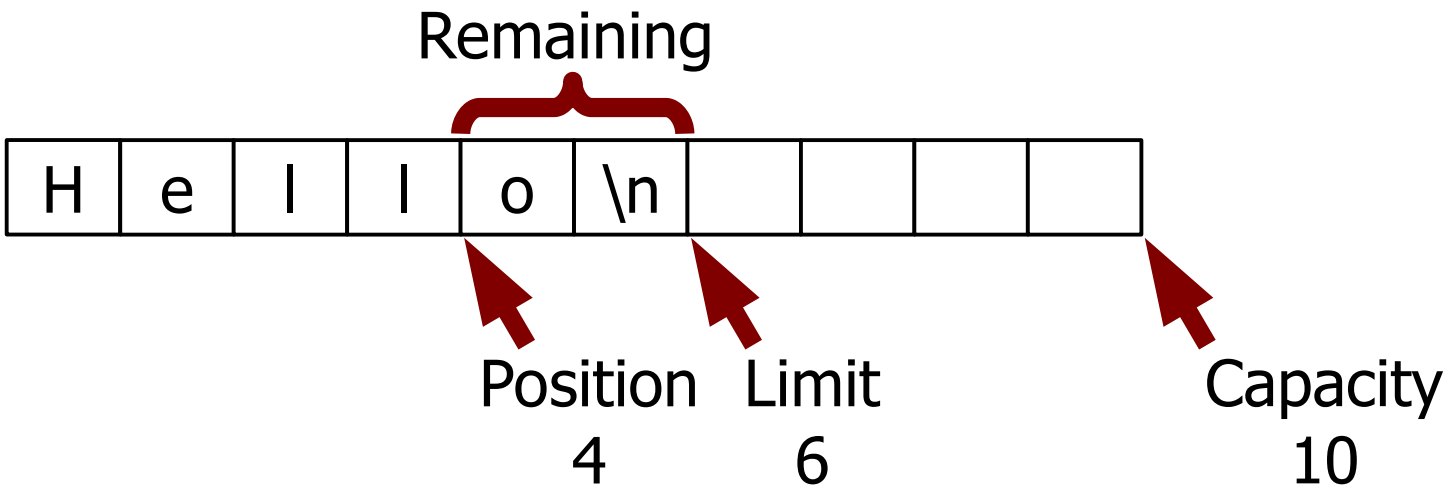
- Memory used: messages in transit (~n)
- Ideally, never allocate or dispose of memory in normal operation:
  - No overhead, but...
  - Needs reference counting to know when to reuse
- Good tradeoff:
  - Take advantage of the garbage collector
  - Reduce the copying and duplication for large objects

Threads summary

- Simple programming model
- Problems:
  - Memory overhead (stacks)
  - Context switches and lock contention
  - “Thundering herd”, hidden queues, and fairness

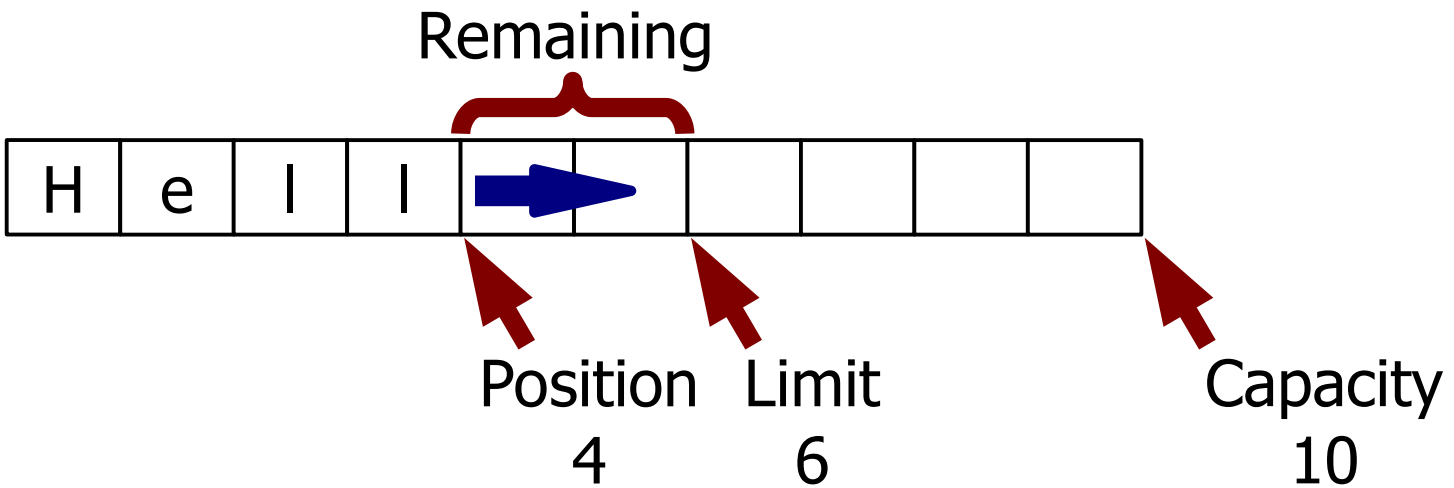
Buffers in java.nio

- Buffer = Array + Indexes:



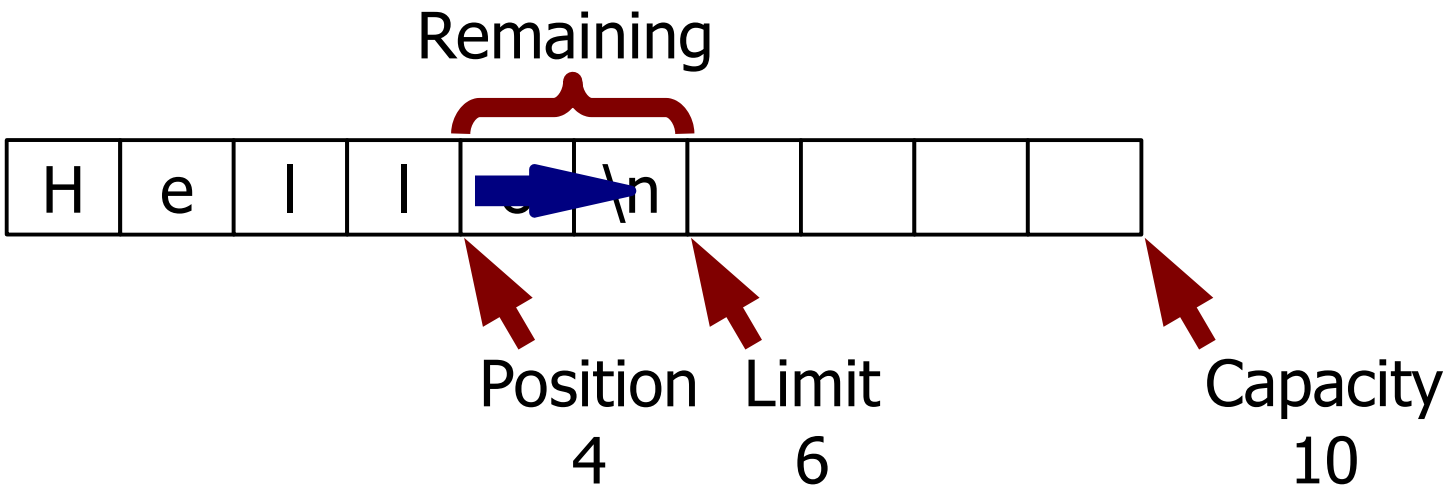
Buffers in java.nio

- Put/read: advances position, sets content



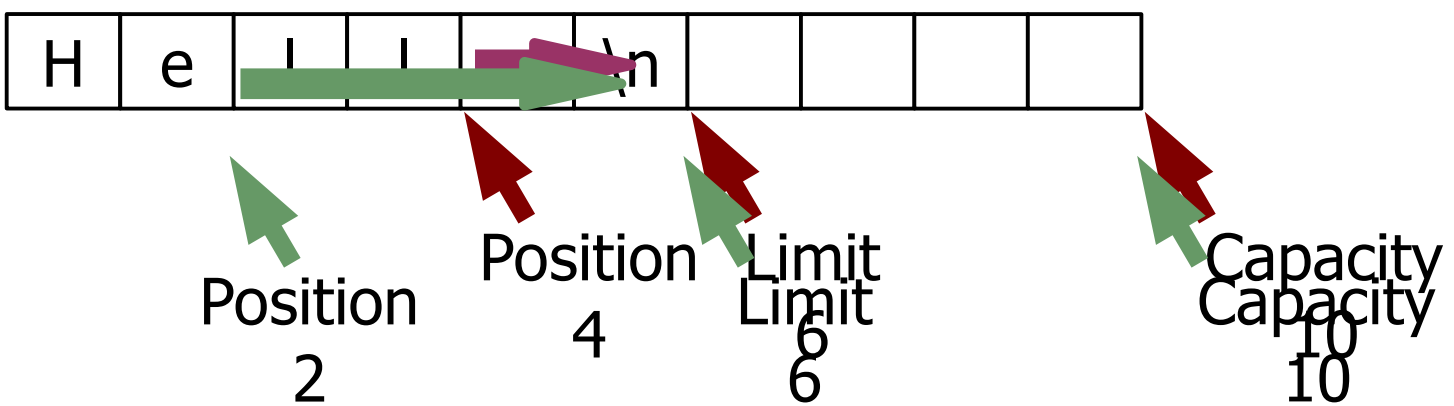
Buffers in java.nio

- Get/write: advances position, gets content

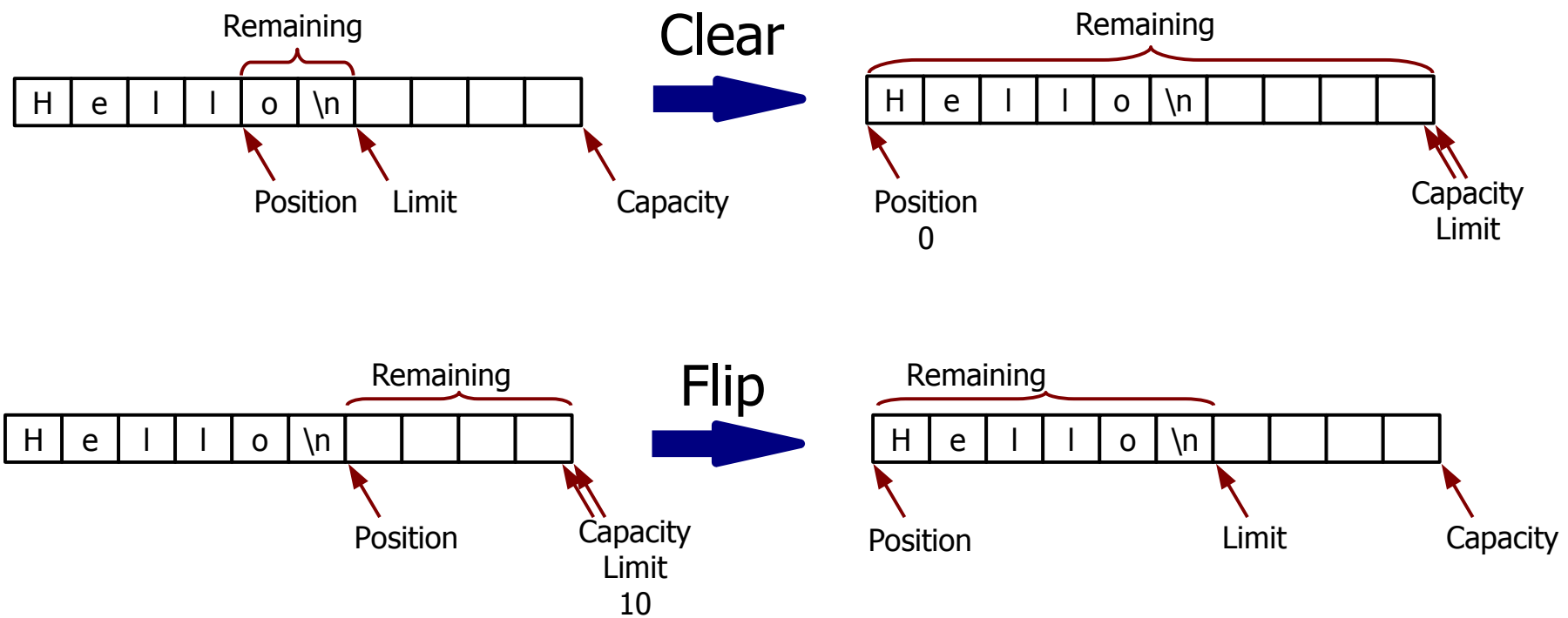


Buffers in java.nio

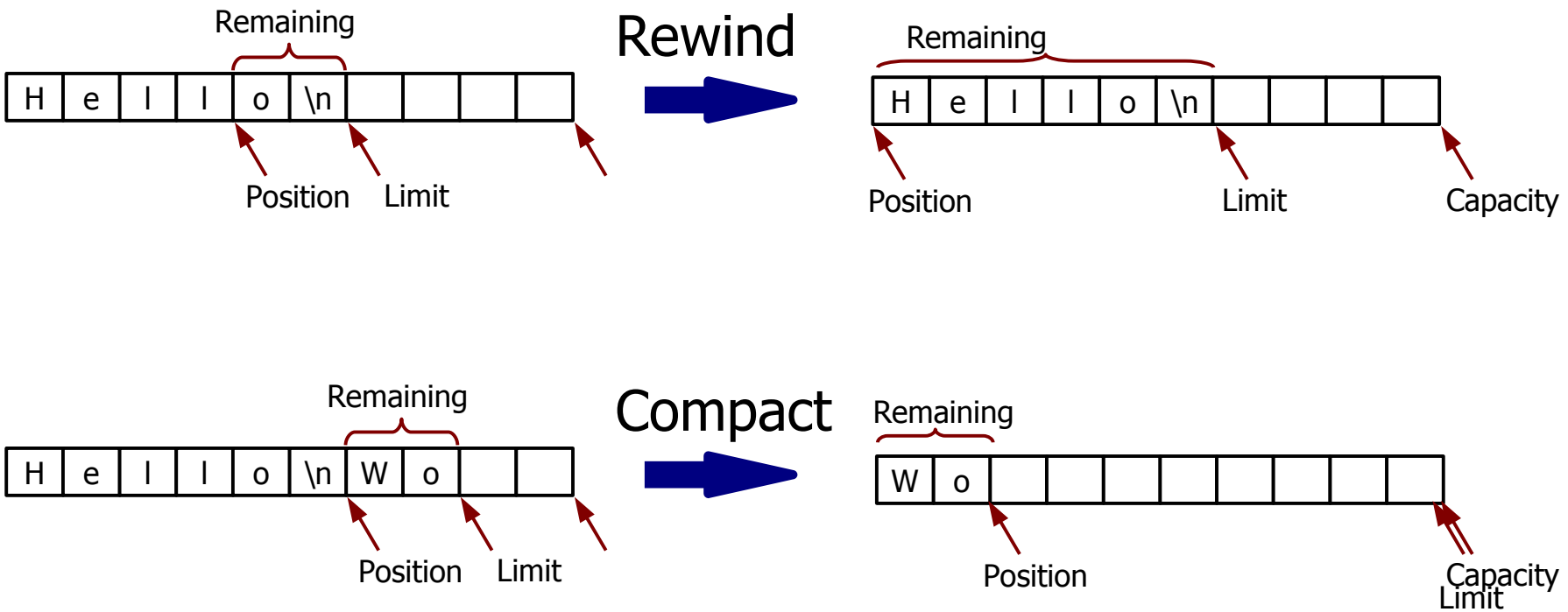
- Duplicate: multiple pointers into the same array



Buffers in java.nio



Buffers in java.nio



Serialization

- Strings need to be encoded to UTF-8, ASCII, … and decoded back when received

```
Charset utf8 = StandardCharsets.UTF8;
String s = ...
ByteBuffer b = ...

b = utf8.encode(s);

s = utf8.decode(b);
```

Serialization

- Primitive types can be translated to and from bytes with getInt/putInt, getFloat/putFloat, etc…
- They are represented according to the current byte order of the byte buffer
- The byte order is “big endian” by default but can be changed