Foundations of Distributed Systems

José Orlando Pereira

HASLab / Departamento de Informática
Universidade do Minho

2018/2019

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
- Consider:
  - Large number of clients
  - Slow connections

First threaded solution

- For each connection:
  - Handler thread
- When reading, write to all other connections
- Use buffering:
  - To minimize system calls
  - To cope with slow readers

Sockets in java.net

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

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

        InputStream is=s.getInputStream();
        OutputStream os=s.getOutputStream();

        // i/o

        s.close();
}
```

Buffers in java.net

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

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

	InputStream is=new BufferedInputStream(s.getInputStream());
	OutputStream os=new BufferedOutputStream(s.getOutputStream());

	// i/o

	os.flush();

	s.close();
}
```

Needed to actually write!

Memory

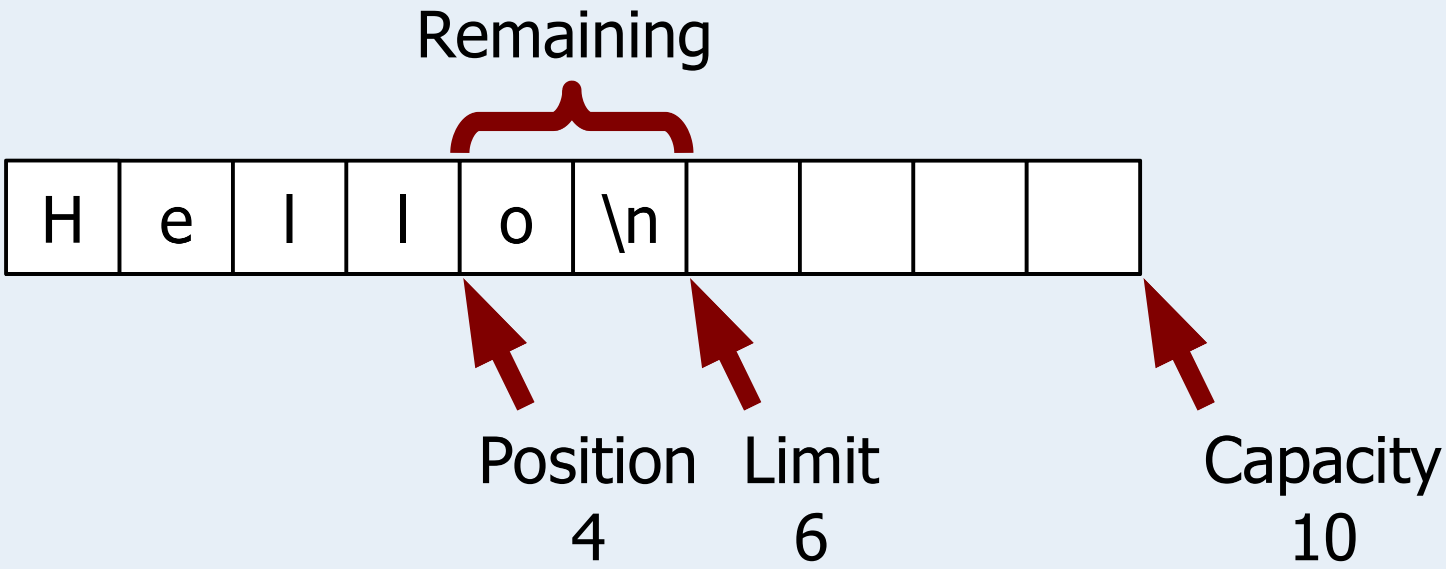
- Memory used: *n* connections x messages in transit (~ n^2)
  - Caused by data copying in stacked abstractions
    - Serialization!
  - Overhead in allocation and garbage collection
- Solution: Store transient data in reusable shared buffers
  - Pointers/indexes into statically allocated data

Sockets in java.nio

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

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

	// i/o

	s.close();
}
```

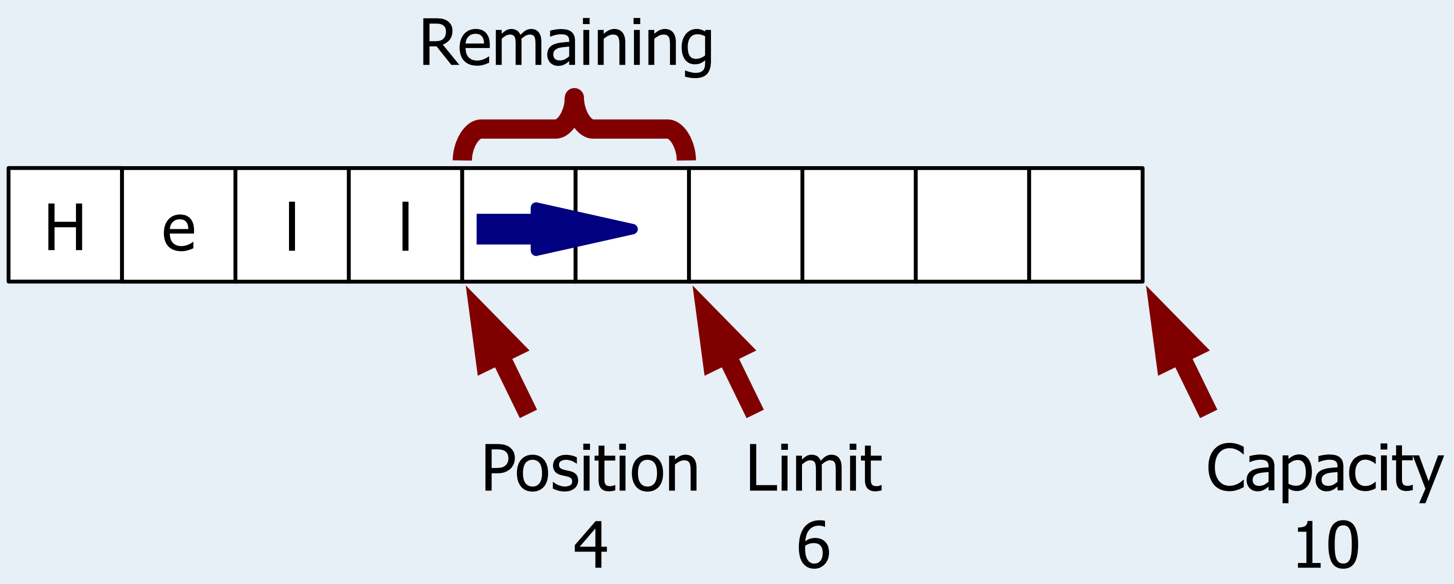
Buffers in java.nio

- Buffer = Array + Indexes:



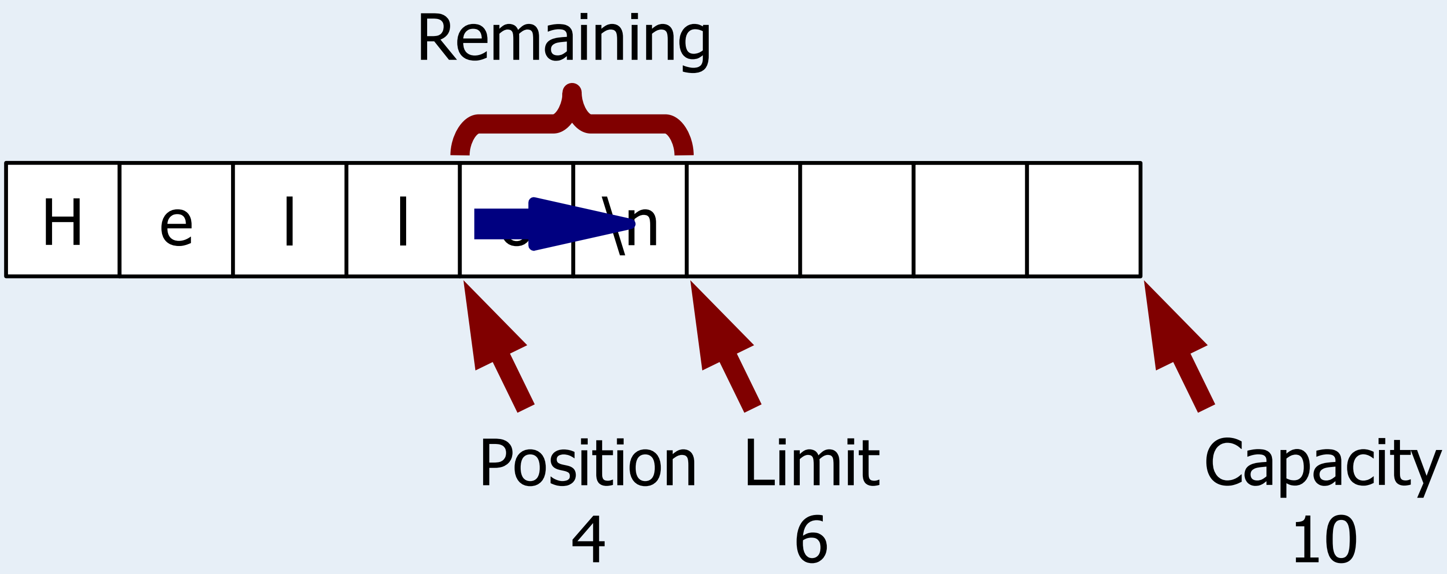
Buffers in java.nio

- Put/read: advances position, sets content



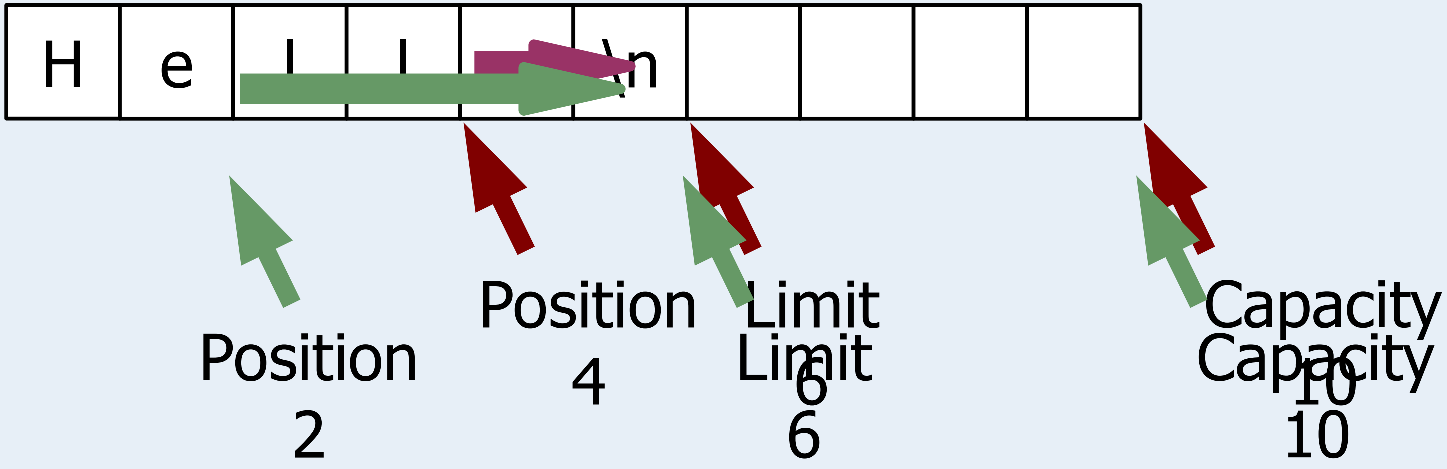
Buffers in java.nio

- Get/write: advances position, gets content

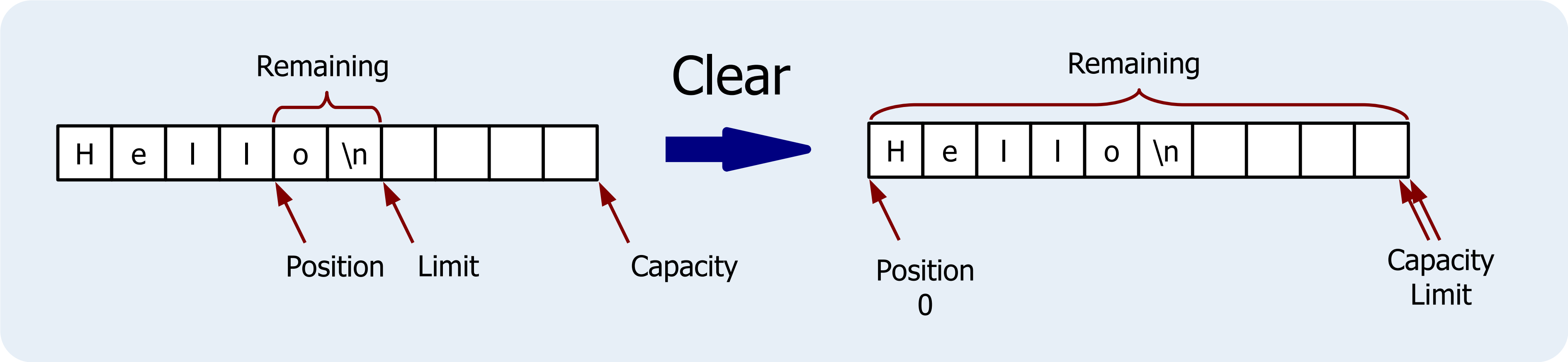


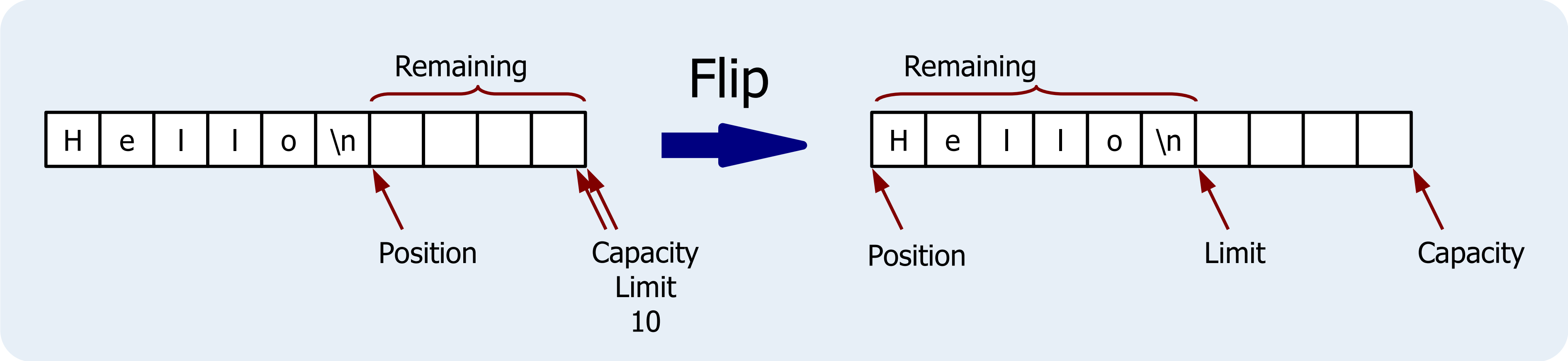
Buffers in java.nio

- Duplicate and wrap: multiple pointers into the same array

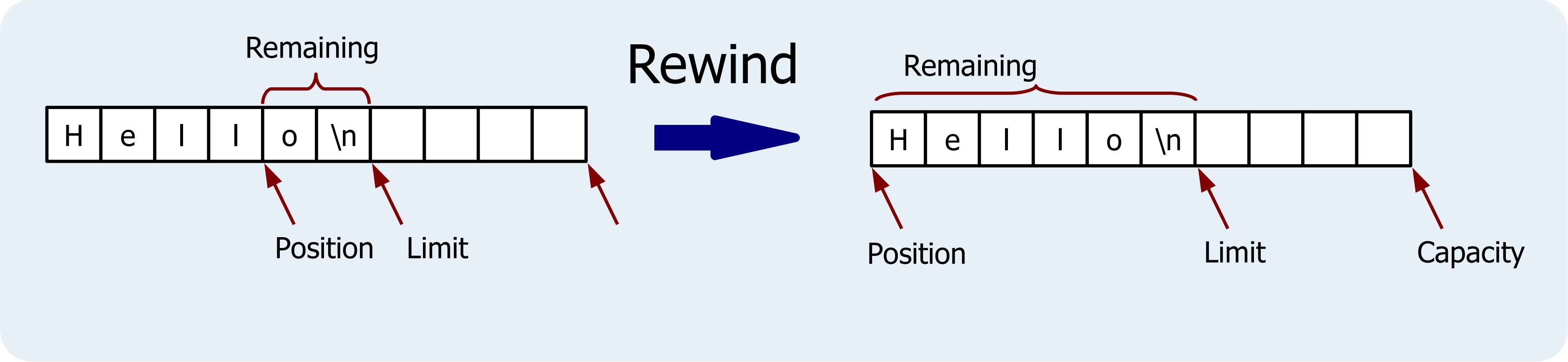


Buffers in java.nio





Buffers in java.nio



Rewind

Remaining

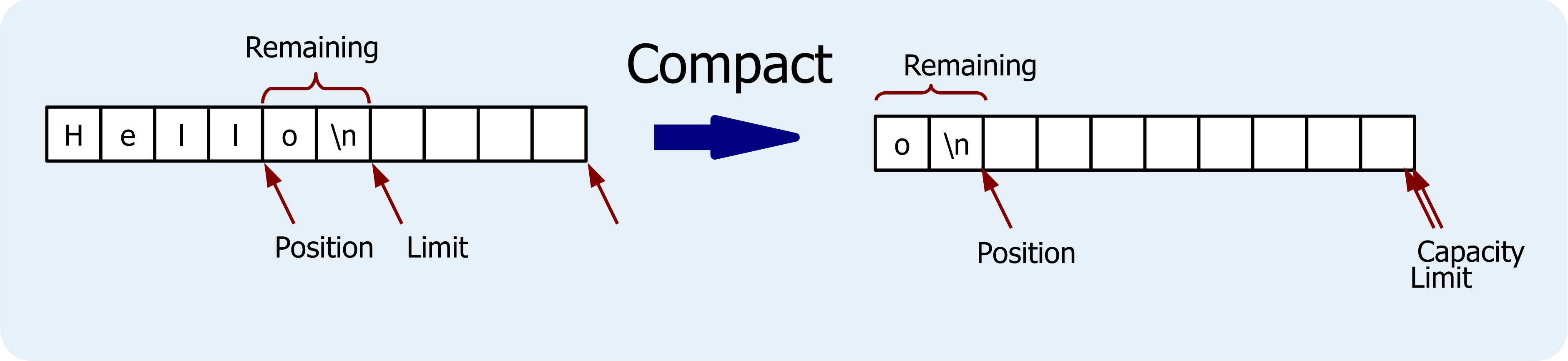
H	e	l	l	o	\n				

Position Limit

Remaining

H	e	l	l	o	\n				

Position Limit Capacity



Compact

Remaining

H	e	l	l	o	\n				

Position Limit

Remaining

o	\n								

Position

Capacity
Limit

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

Flushing buffers

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

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

	InputStream is=new BufferedInputStream(s.getInputStream());
	OutputStream os=new BufferedOutputStream(s.getOutputStream());

	// i/o

	os.flush();

	s.close();
}
```

What if
write blocks?

Second threaded solution

- For each connection:
  - Reader thread
  - Writer thread
  - Pending queue
- When reading, insert in queues and notify writer threads
- When writing, remove from queue and notify reader threads

Threads

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

Blocking sockets

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

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

        r.write(buf);

} catch(IOException e) {
        report(e);
}
```

Asynchronous sockets

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

s.read(buf, null, new CompletionHandler() {
        public void completed(Integer result, Object a) {
                buf.flip();

                r.write(buf, ...);
        }
        public void failure(Throwable t, Object a) {
                report(t);
        }
);
```

Thread pools

- For non-blocking, short-lived events:
  - One pool thread for hardware thread
- While all threads are blocked, the application stops handling events

```
AsynchronousChannelGroup g =
     AsynchronousChannelGroup.withFixedThreadPool(...);

AsynchronousSocketChannel s =
     AsynchronousSocketChannel.open(g);

...  /* callbacks use g.shutdown() to exit */

g.awaitTermination(Long.MAX_VALUE, TimeUnit.SECONDS);
```