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);
}
```

Translation to CompletionHandler

```
try {
	C c = codeBefore(...);

	R r = operation(...);

	codeAfter(c, r);

} catch(Exception e) {
	handleException(e);
}
```

```

C c = codeBefore(...);

asyncOperation(..., c, new CompletionHandler<R,C>() {
	public void sucess(R r, C c) {
		codeAfter(c, r);
	}
	public void failure(Exception e, C c) {
		handleException(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);
```