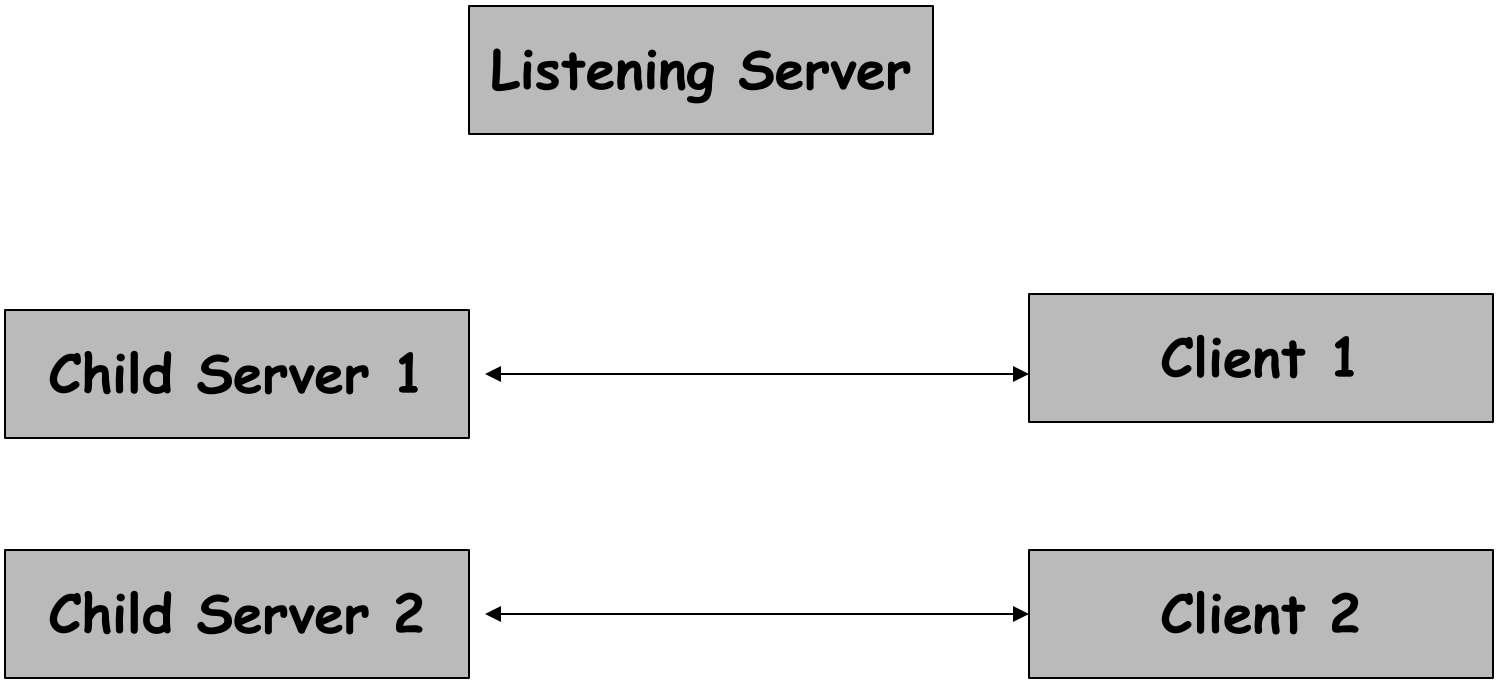
I/O Models

Kartik Gopalan

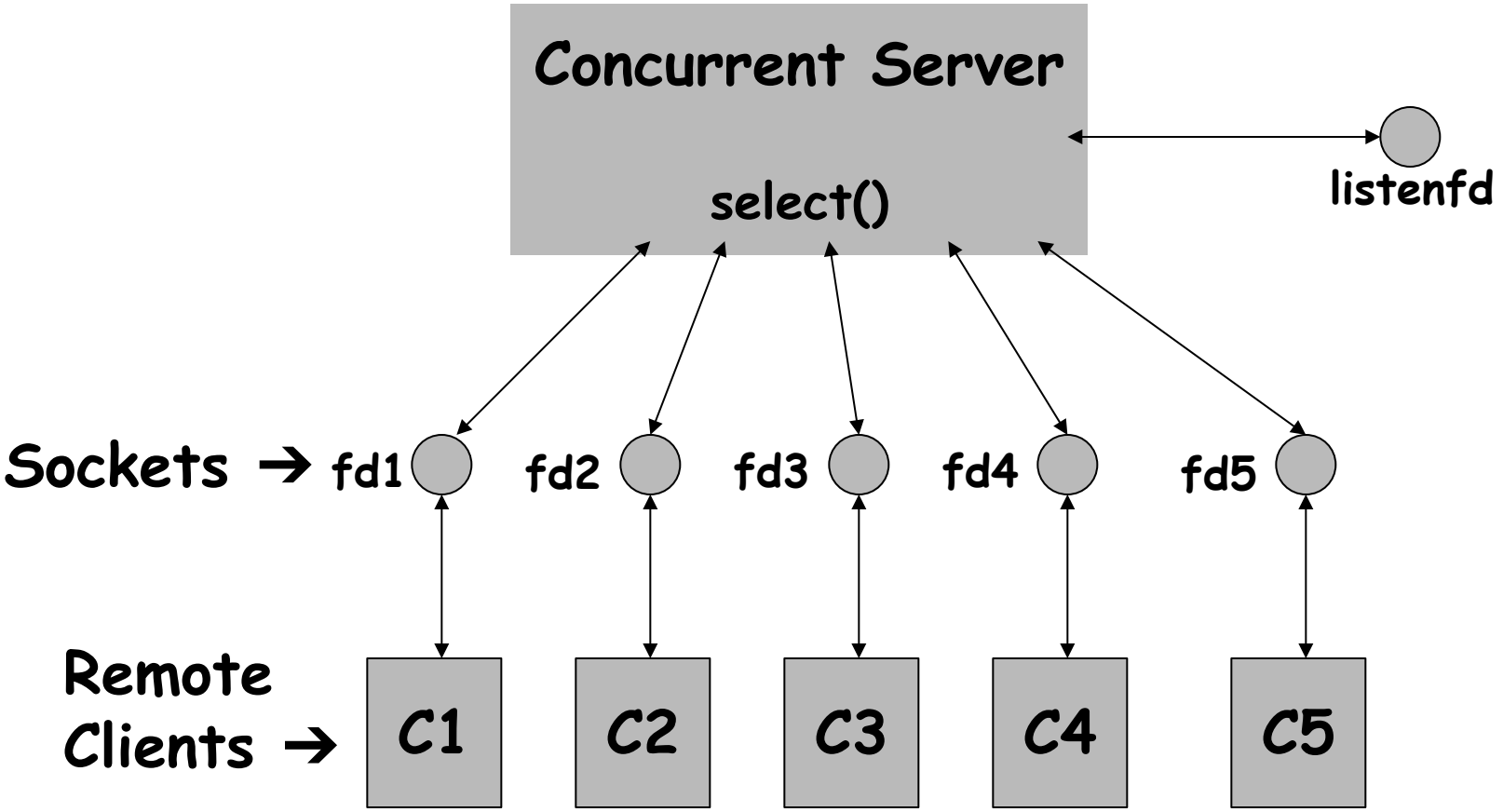
Types of Concurrency

- True Concurrency (multiple processes or threads)
  - Multi-processor machines
    - Child processes/threads execute in parallel.
  - Multi-process (forking) servers
    - If one child server blocks, another executes
  - Multi-threaded servers
    - If one thread blocks, another executes.
    - Only if threads supported by kernel
- Apparent Concurrency
  - Single process does multiplexing among multiple clients.
    - E.g. I/O multiplexing with select().
  - Multi-threaded Server (again!)
    - If threads implemented at user level.

True Concurrency: Forking Concurrent Servers



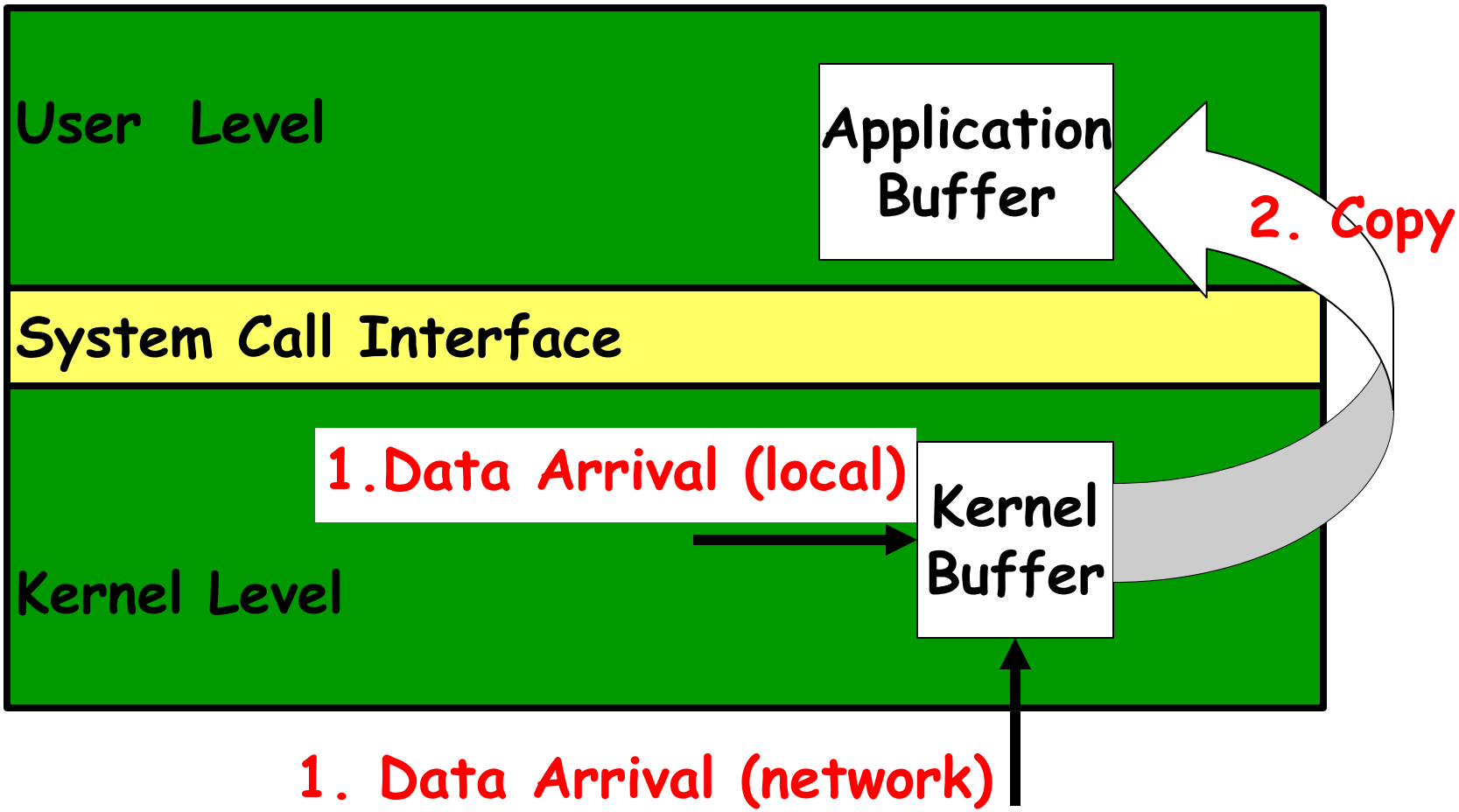
Apparent Concurrency
Non-Forking Concurrent Server



I/O Models

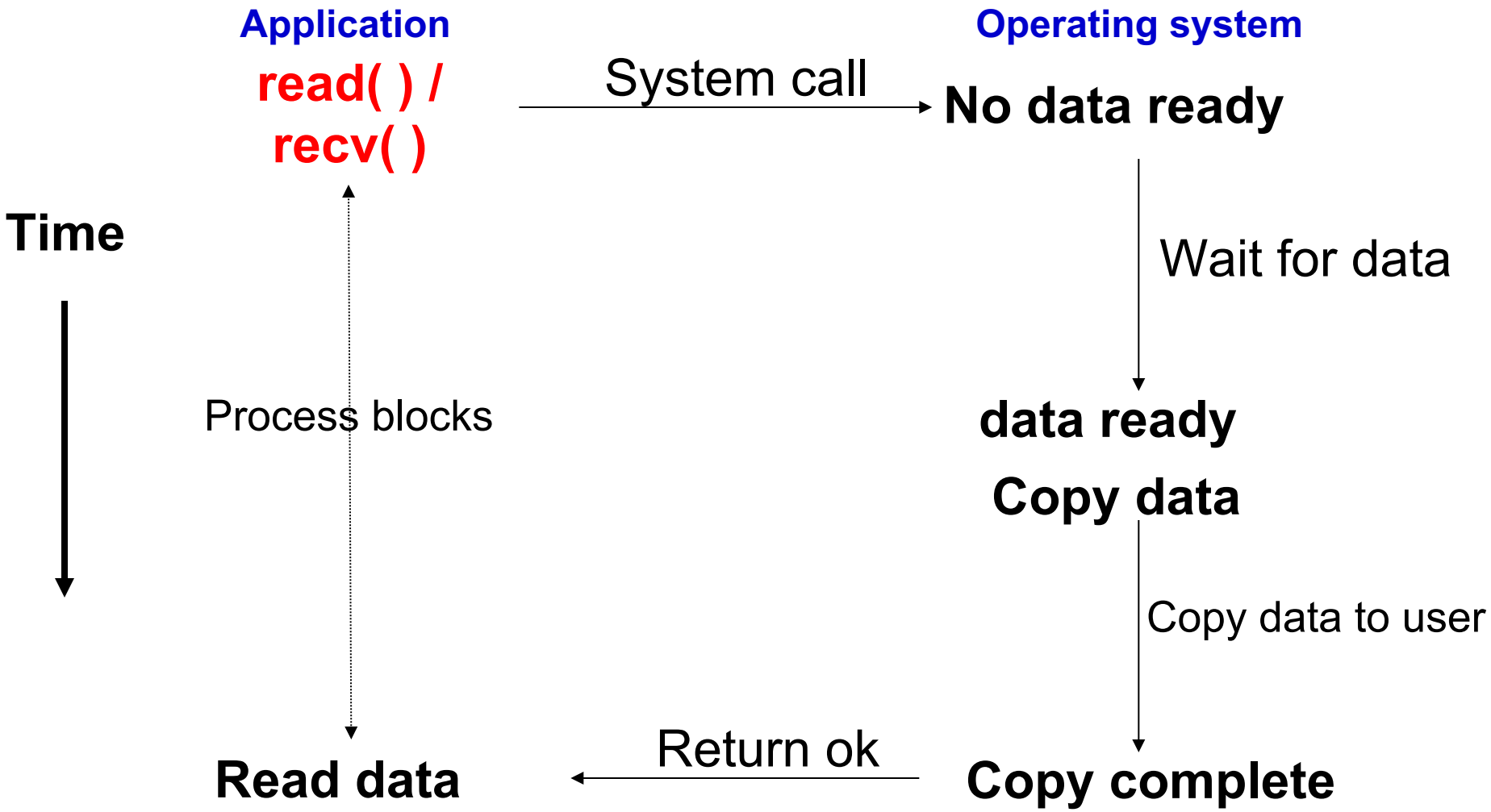
- Blocking I/O
- Non-blocking I/O
- I/O multiplexing – select()
- Signal driven I/O
- Asynchronous I/O

Two steps in data reception

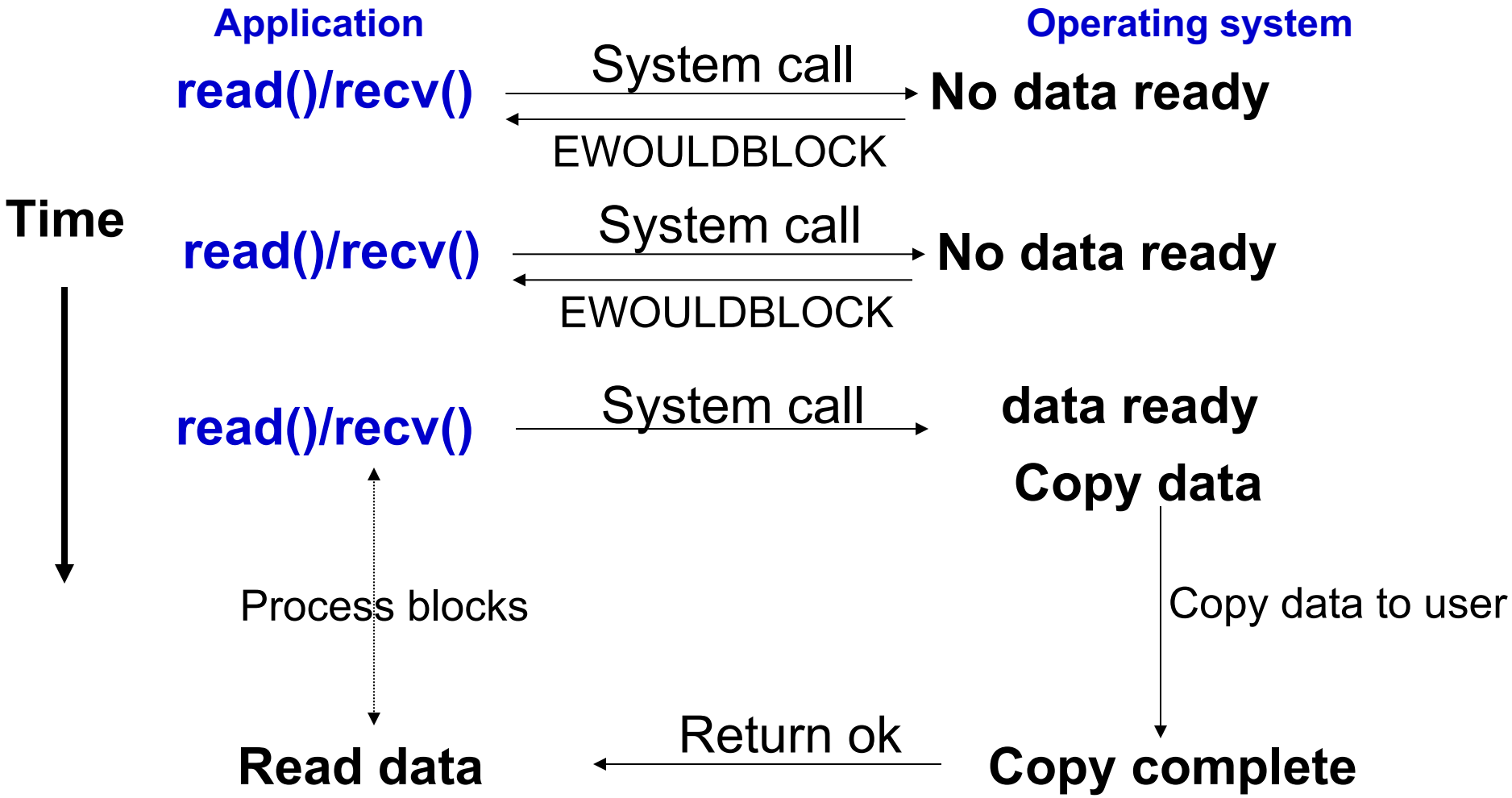


Overheads: Context switching, Data copying

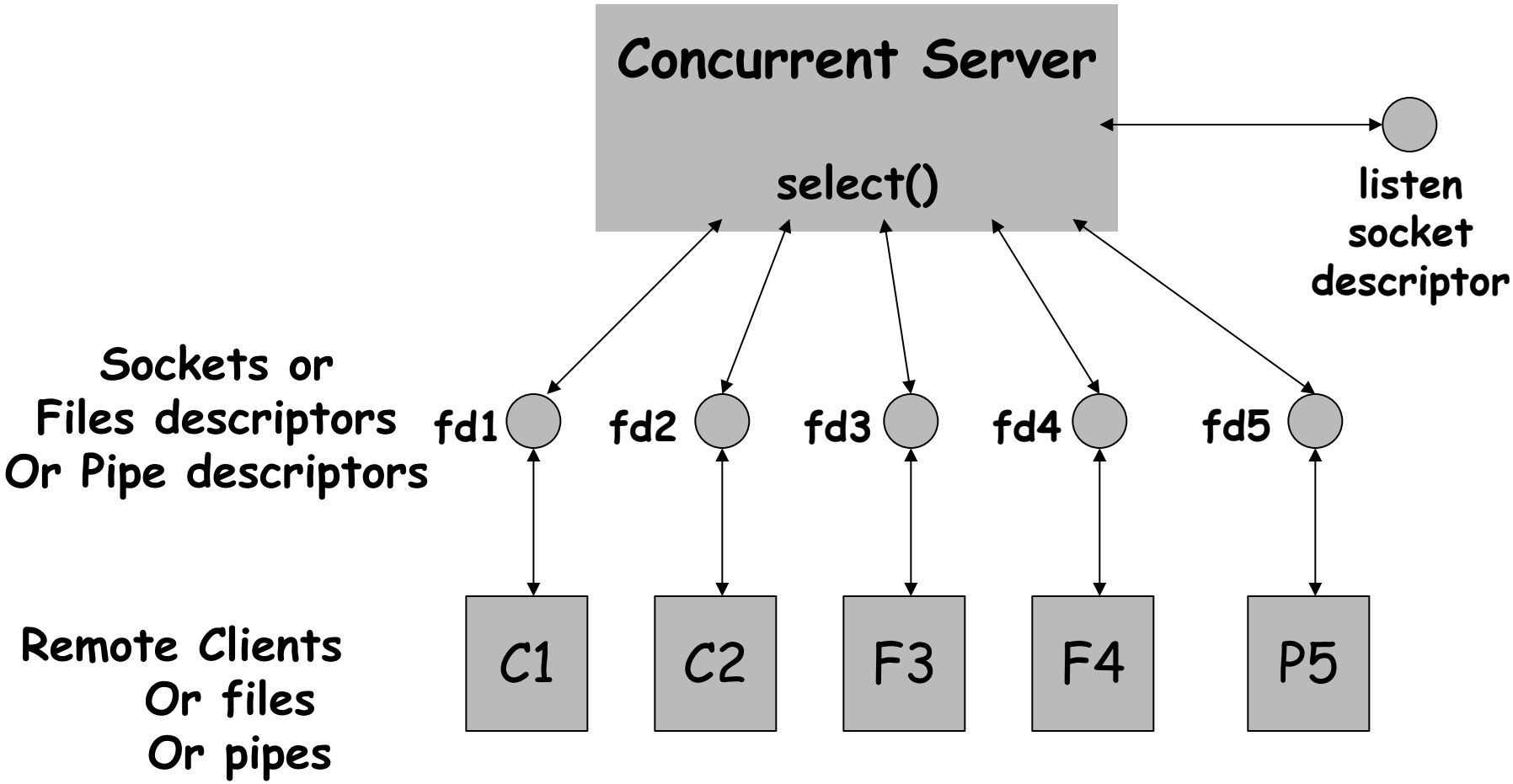
Blocking I/O


Application
read( ) /
recv( )
System call
Operating system
No data ready
Time
Wait for data
Process blocks
data ready
Copy data
Copy data to user
Read data
Return ok
Copy complete

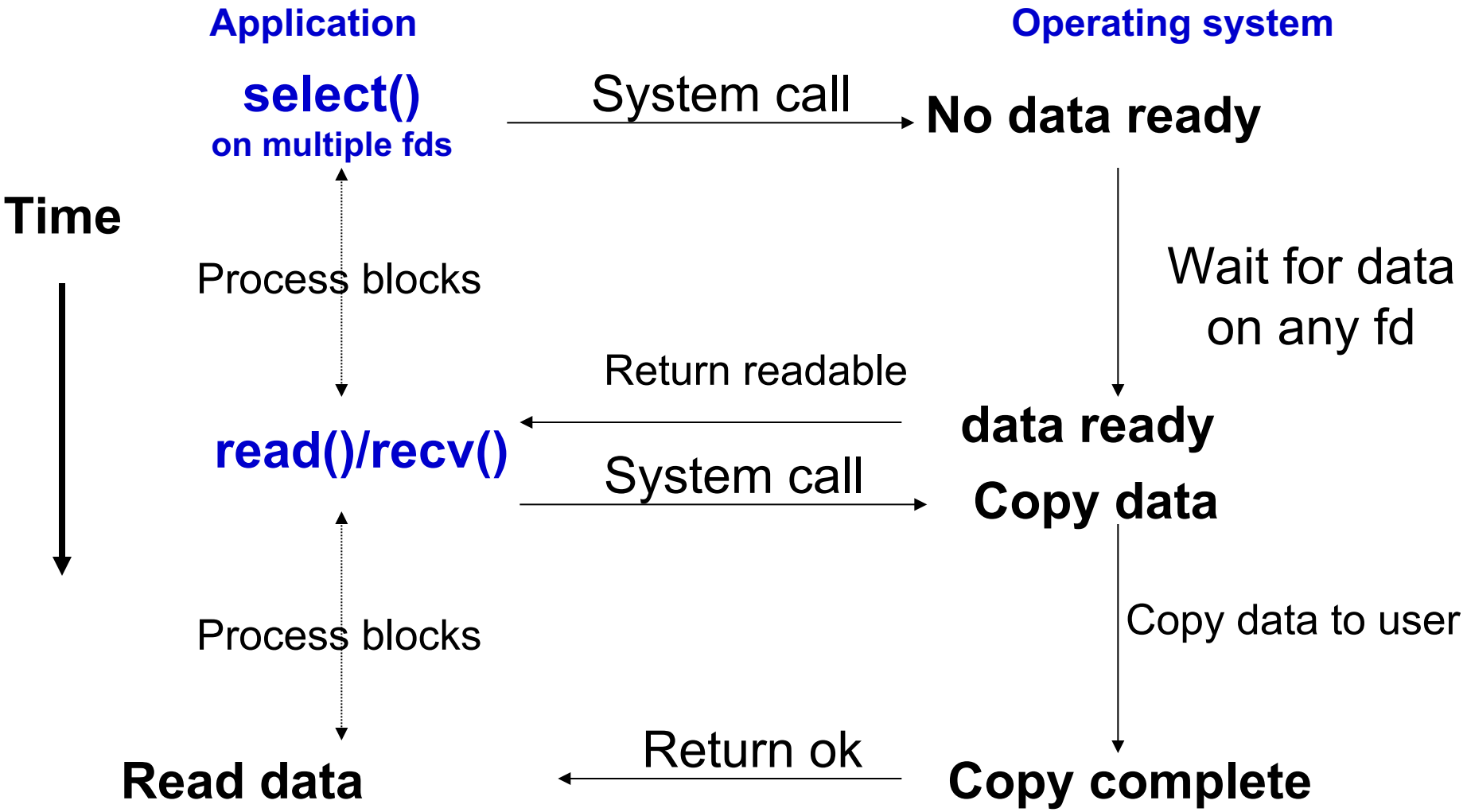
Non-Blocking I/O (polling)



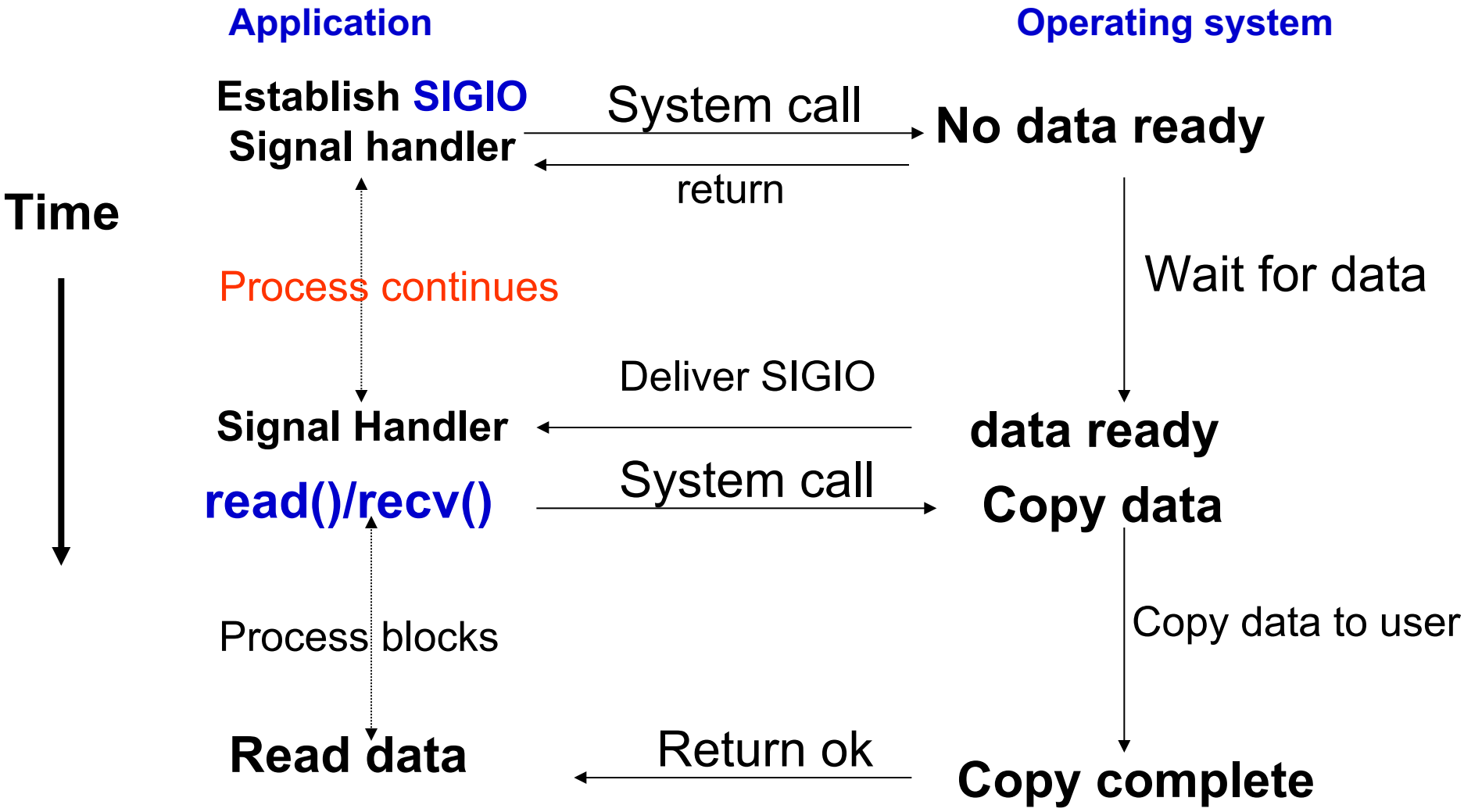
I/O Multiplexing



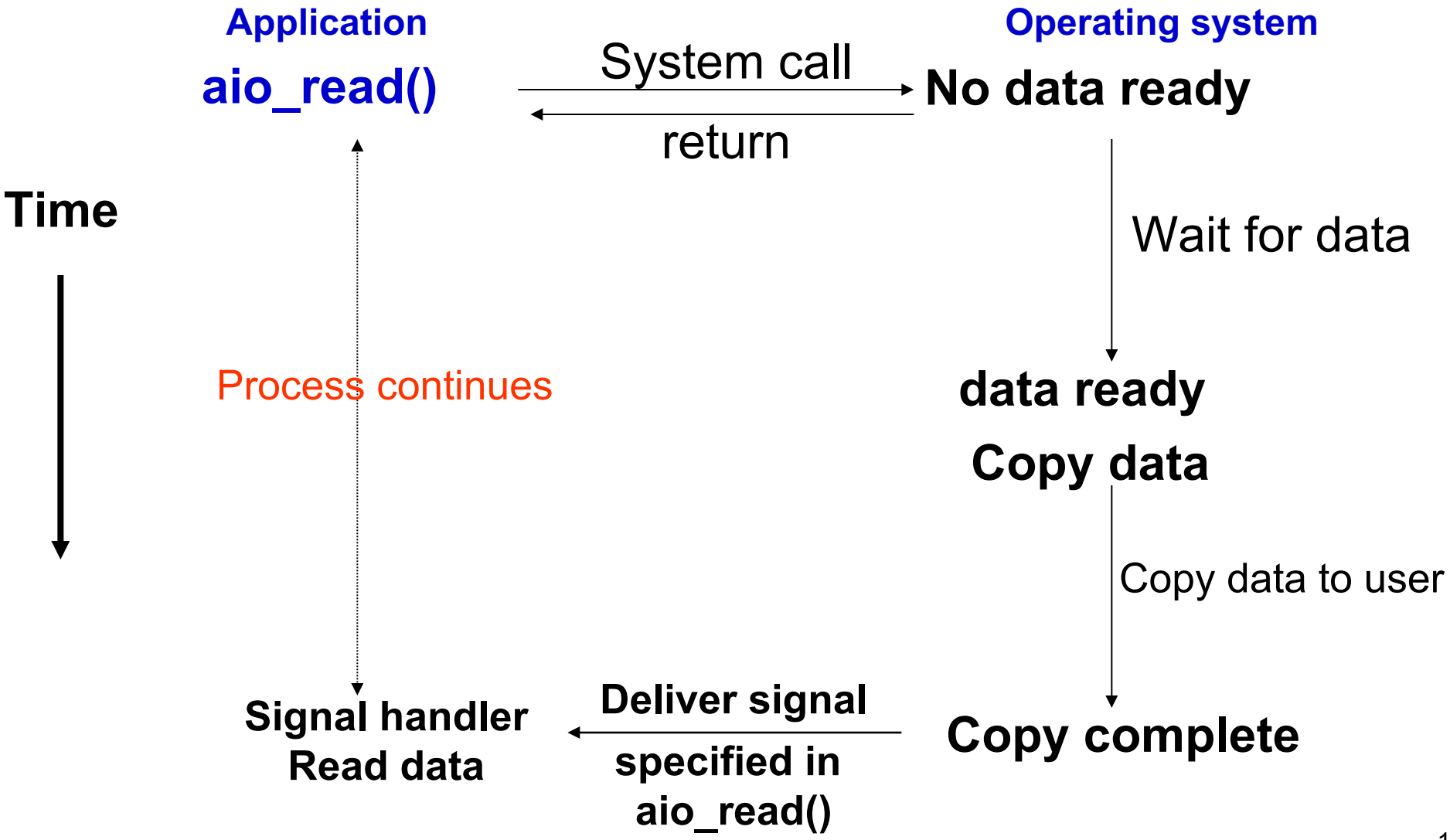
I/O Multiplexing



Signal driven I/O



Asynchronous I/O



I/O Multiplexing

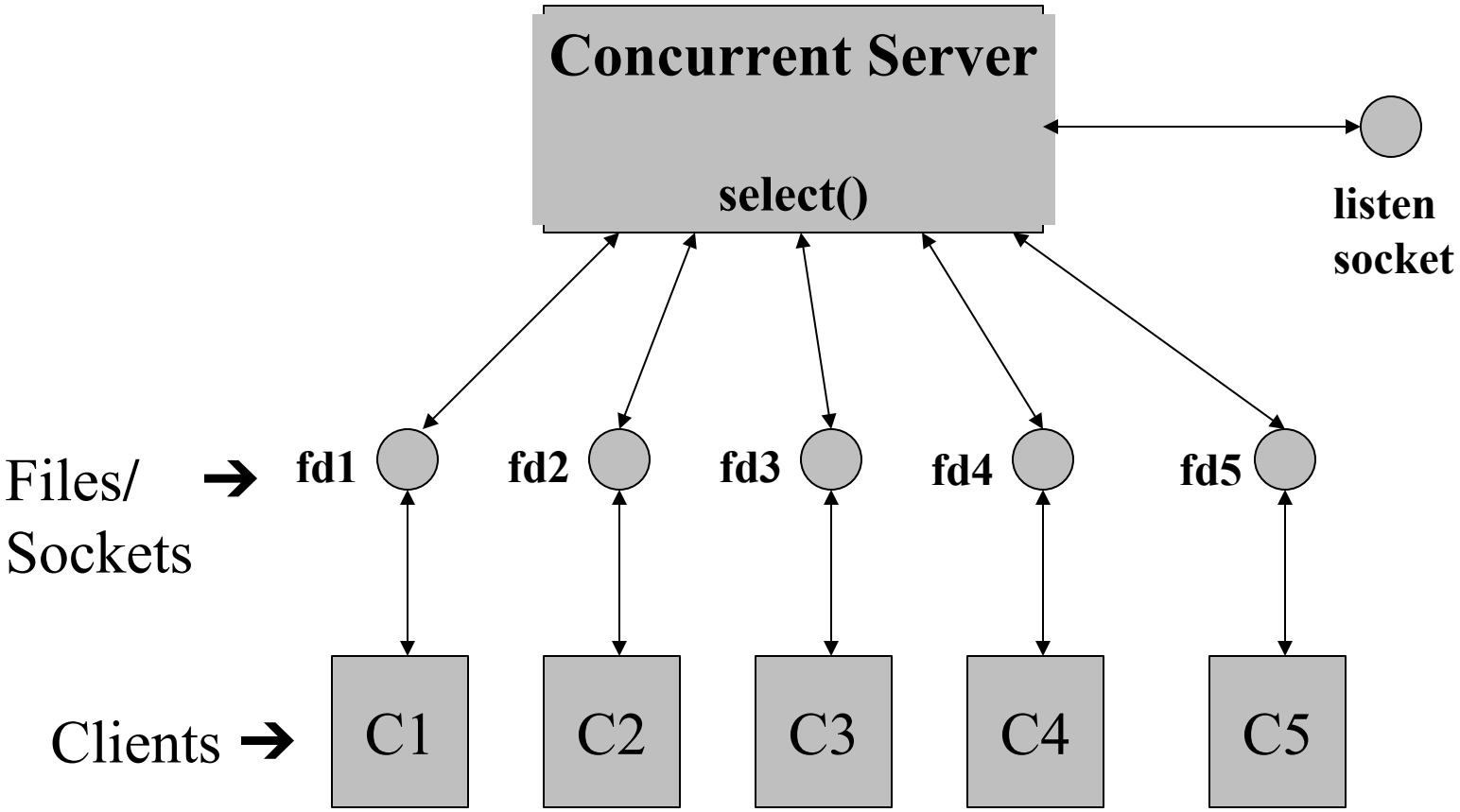
Example of Event-oriented programming

What is I/O multiplexing?

- When an application needs to handle multiple I/O descriptors at the same time
  - E.g. file and socket descriptors, multiple socket descriptors

- When I/O on any one descriptor can result in blocking

Non-forking concurrent server



select() call

- Allows a process to wait for an event to occur on any one of its descriptors.

- Types of event
  - ready for read
  - ready for write
  - Exception condition

select() call

```
int  select(
   int  maxfdp1,              /* max. fd + 1 */
   fd_set  *readfds,  /* read ready? */
   fd_set  *writefds,         /* write ready? */
   fd_set *exceptfds,         /* exceptions? */
   struct timeval *timeout);

struct  timeval {
    long tv_sec;  /* seconds */
    long tv_usec; /* microseconds */
}
```

struct fd_set

- Set of descriptors that we want to wait on for events.
- Typically holds 256 descriptor states.
- Manipulation macros
  - void FD_ZERO(fd_set *fds)
  - void FD_SET (int fd, fd_set *fds)
  - void FD_CLR (int fd, fd_set *fds)
  - int FD_ISSET(int fd, fd_set *fds)

Non-forking Concurrent Server

```
fdset rdset, wrset;
int listenfd, connfd1, connfd2;
int maxfdp1;
.....
Connection establishment etc.
.....

/* initialize */
FD_ZERO(&rdset);
FD_ZERO(&wrset);
```

```
for( ;; ) {
    FD_SET(connfd1, &rdset);
    FD_SET(connfd2, &wrset);
    FD_SET(listenfd, &rdset);

    maxfdp1 = max(connfd1, connfd2, listenfd) + 1;

    /* wait for some event */
    Select(maxfdp1, &rdset, &wrset, NULL, NULL);

    if( FD_ISSET(connfd1, &rdset) ) {
      Read data from connfd1…
    }
    if( FD_ISSET(connfd2, &wrset) ) {
      Write data to connfd2…
    }
    if( FD_ISSET(listenfd, &rdset) {
      Process a new connection…
    }
}
```