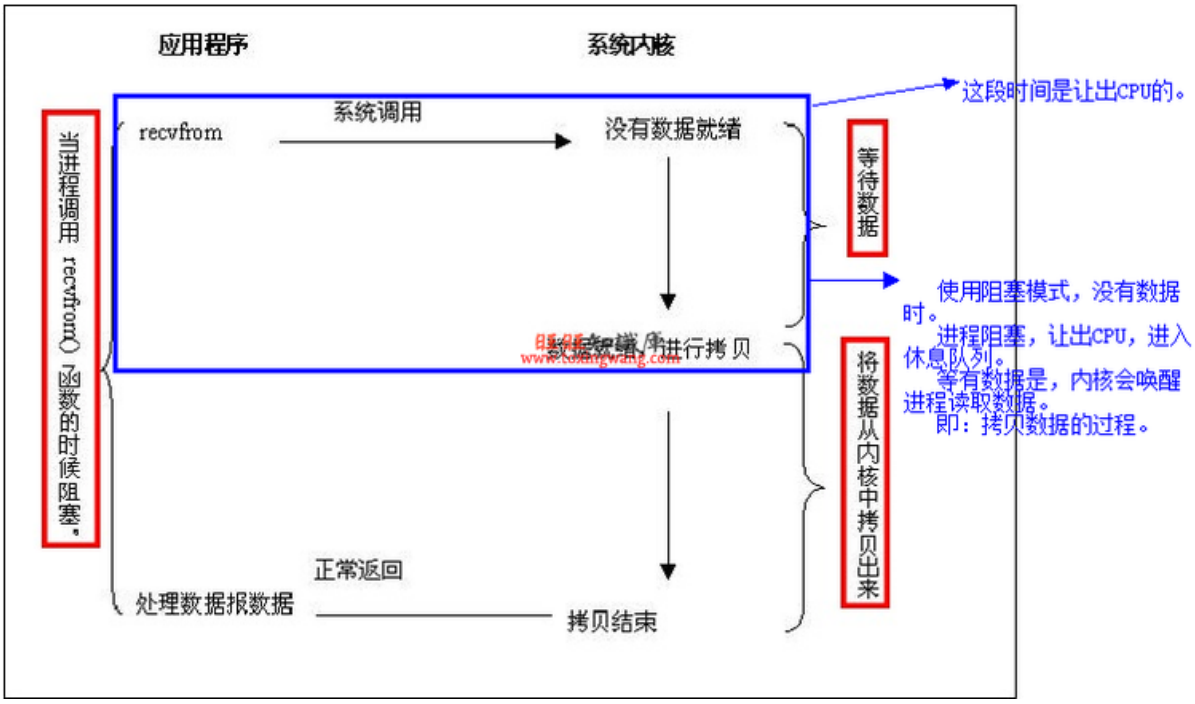
**关键词**

> 协程 异步 生成器 阻塞 yield yieldfrom async/await

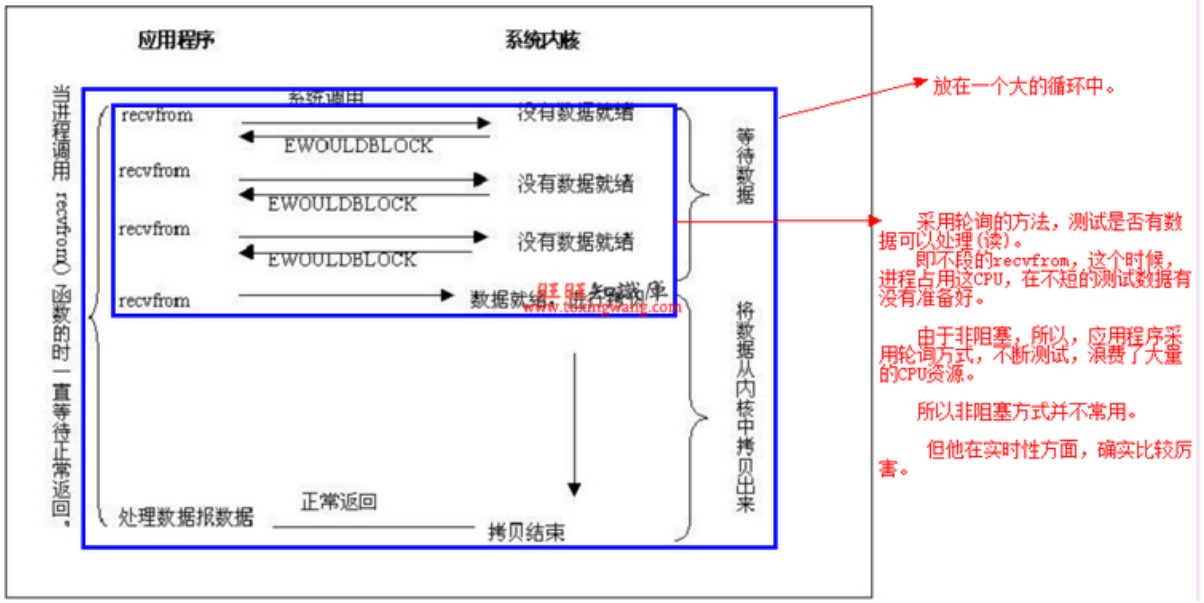
1. 并发：一个时间段内，有几个程序在同一个 CPU 上运行，但是任意时刻只有一个程序在 CPU 上运行。
2. 并行：在任意时刻点上，有多个程序同时运行在** 多个CPU **上。如果CPU有个四颗，那么并行最多只有四个
3. 第1个假设：假定异步非阻塞 同步阻塞没区别 至少不影响理解 阻塞非阻塞 就是执行一个操作是等操作结束再返回，还是马上返回。
4. 第2个假设：首先异步 同步 阻塞 非阻塞 事件驱动一定是IO操作么 其他操作会导致阻塞非阻塞么 我假定这里只有IO操作导致这些概念的产生
5. 线程之间的同步 是之间 那是用锁等东西来访问共同数据，但是这里说的同步异步阻塞非阻塞都是单线程的function call
6. IO事件

   对于IO来说有5种IO

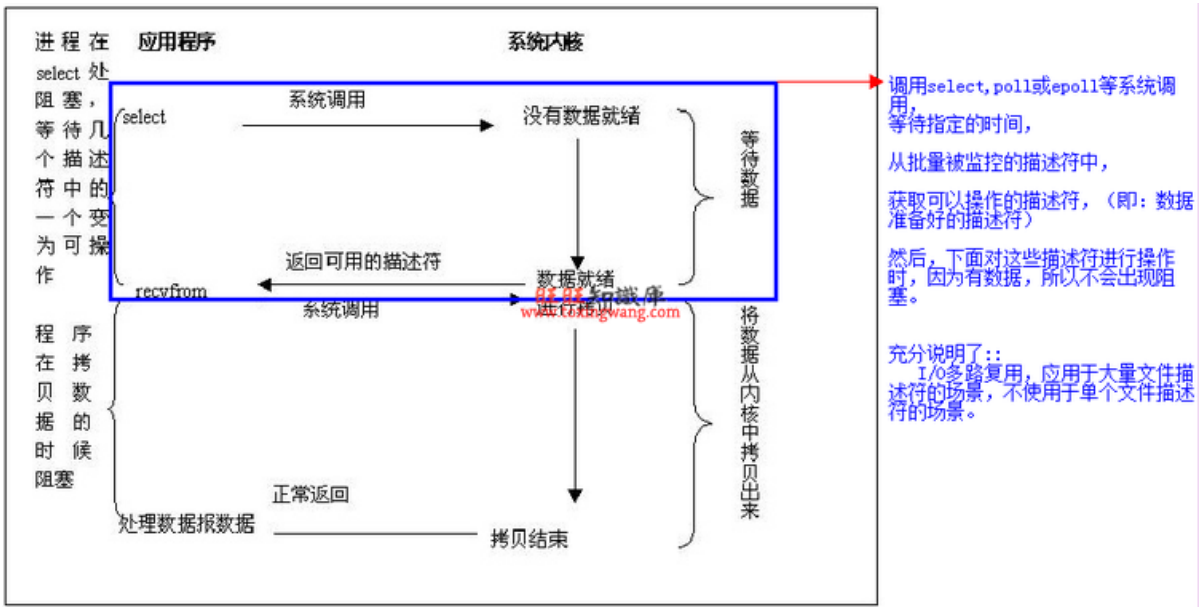
   阻塞I/O：所有过程全阻塞



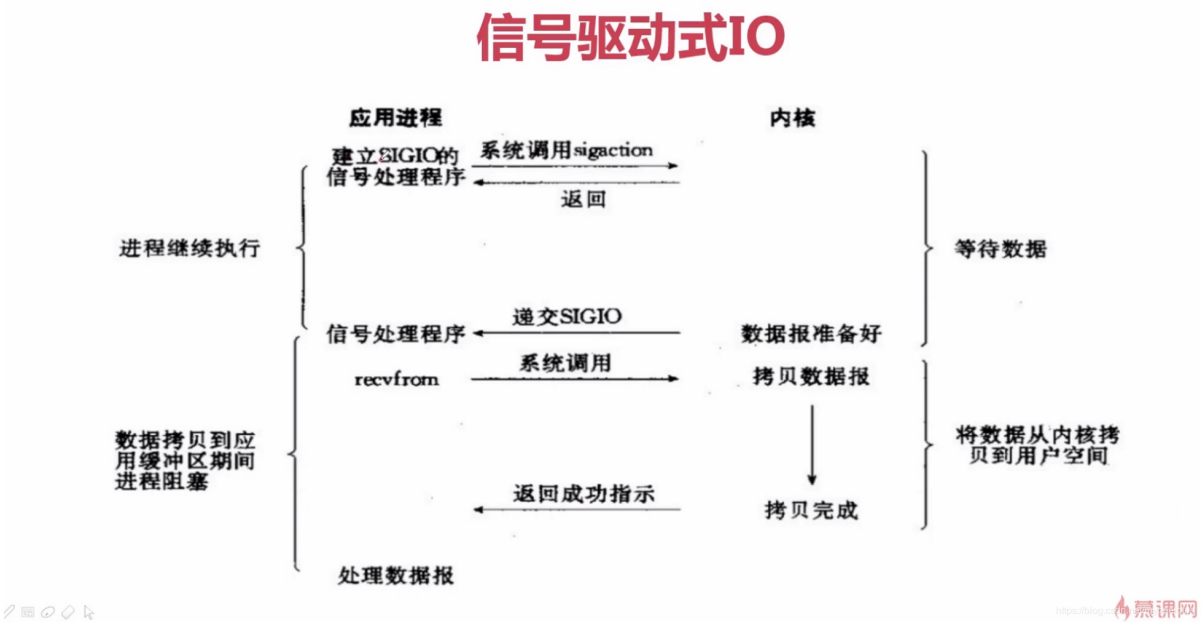

   非阻塞I/O：如果没有数据buffer，则立即返回EWOULDBLOCK



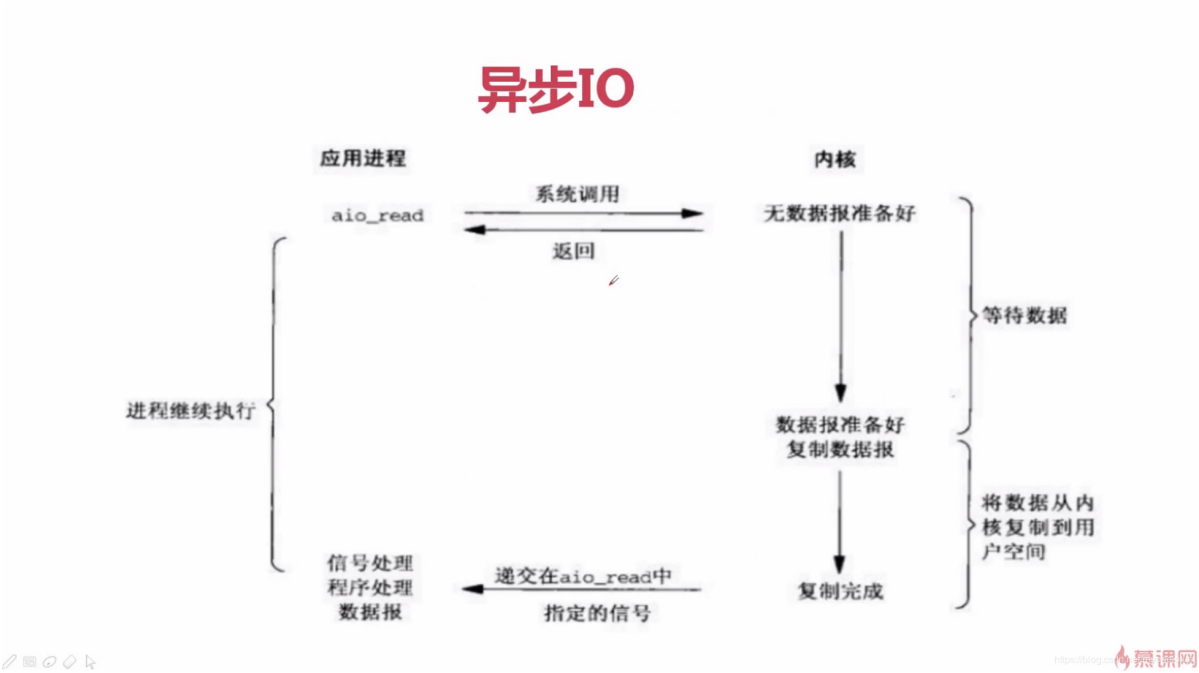

I/O复用（select和poll）：在wait和copy阶段分别阻塞




信号驱动I/O（SIGIO）：在wait阶段不阻塞，但copy阶段阻塞（信号驱动I/O，即通知）



完全异步I/O（aio）：完全五阻塞方式，当I/O完成时提供信号

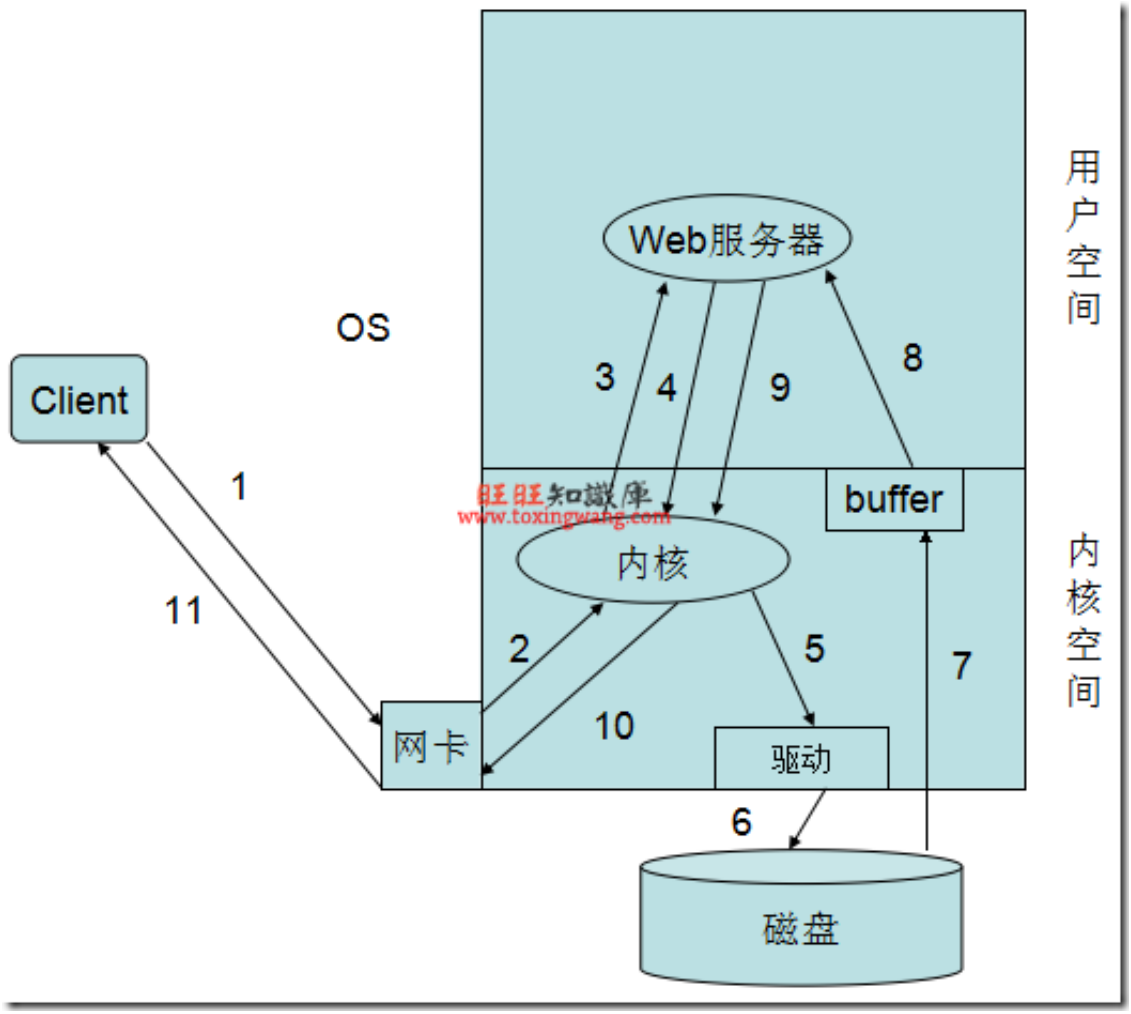



完全异步的操作只有windows的IOCP 信号驱动只是半异步的 两者的区别是 信号驱动 I/O 模式下，内核可以复制的时候通知给我们的应用程序发送SIGIO消息。 异步 I/O 模式下，内核在所有的操作都已经被内核操作结束之后才会通知我们的应用程序。

select poll属于IO复用 epoll也是但是有了信号驱动IO的特性比如callback

kqueue epoll相比与IOCP就是多了一层从内核copy数据到应用层的阻塞，从而不能算作asynchronous I/O类. 但是已经很优秀了

7. web服务器的操作




客户发起情况到服务器网卡； 服务器网卡接受到请求后转交给内核处理； 内核根据请求对应的套接字，将请求交给工作在用户空间的Web服务器进程 Web服务器进程根据用户请求，向内核进行系统调用，申请获取相应资源（如index.html） 内核发现web服务器进程请求的是一个存放在硬盘上的资源，因此通过驱动程序连接磁盘 内核调度磁盘，获取需要的资源 内核将资源存放在自己的缓冲区中，并通知Web服务器进程 Web服务器进程通过系统调用取得资源，并将其复制到进程自己的缓冲区中 Web服务器进程形成响应，通过系统调用再次发给内核以响应用户请求 内核将响应发送至网卡 网卡发送响应给用户 通过这样的一个复杂过程，一次请求就完成了。

8. 一些问题

> 生成器 异步生成器 协程 yield/yield from async/await之间的关系

> 生成器 异步生成器 协程 yield/yield from async/await在c中的实现

> asynchronous context manager/asynchronous generator iterator/asynchronous iterable/asynchronous iterator/

> 参考官网 https://docs.python.org/3/reference/expressions.html

## 算了python就到此为止 开始go吧 毕竟也有协程和web服务器 书也少就一本