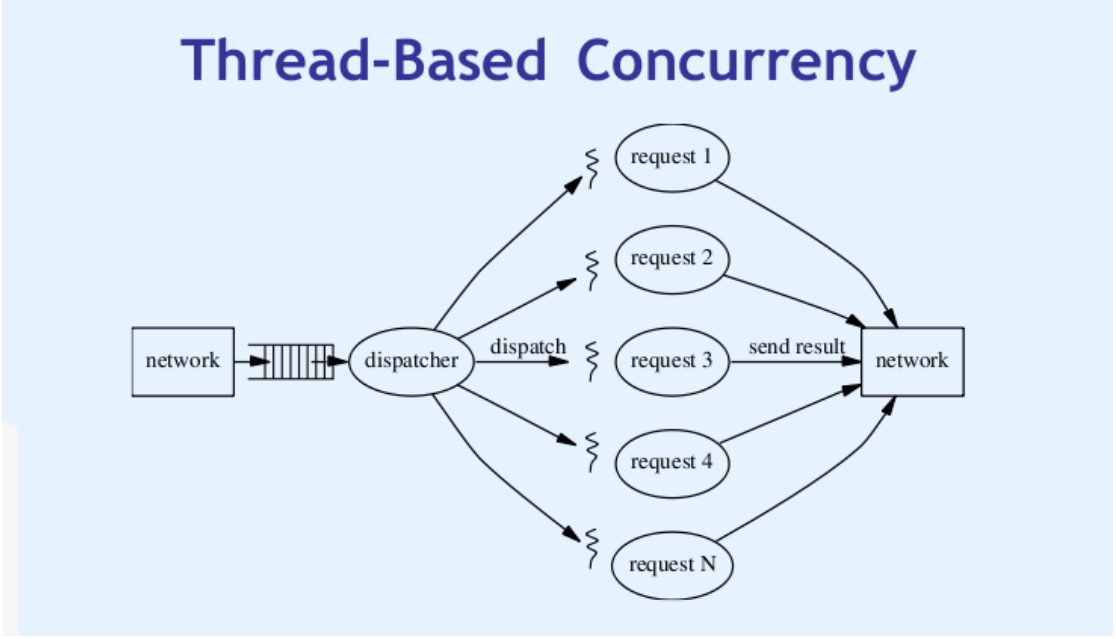
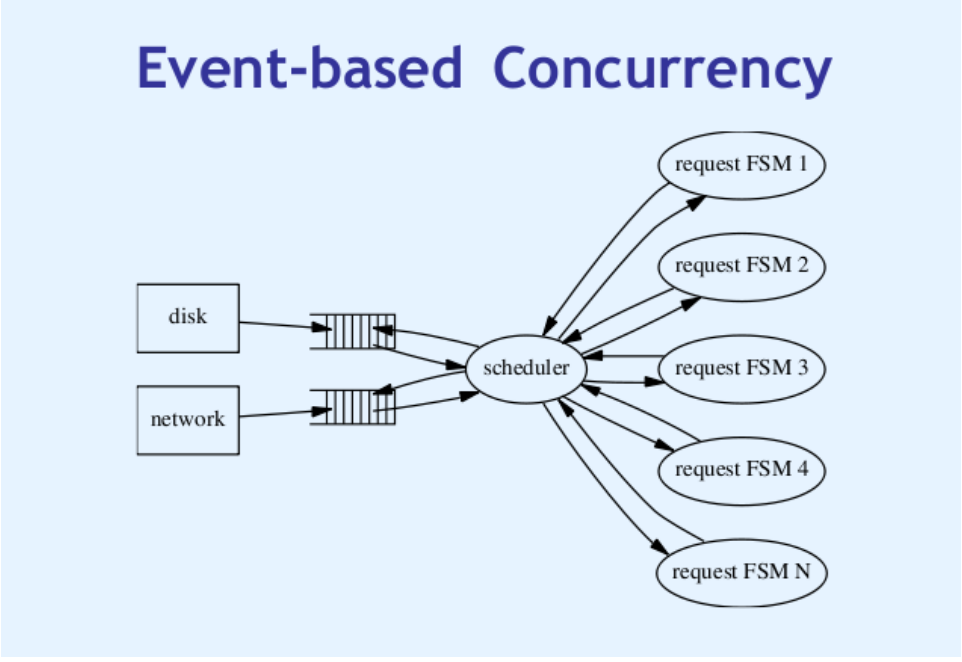
一、传统并发模型的缺点

基于线程的并发



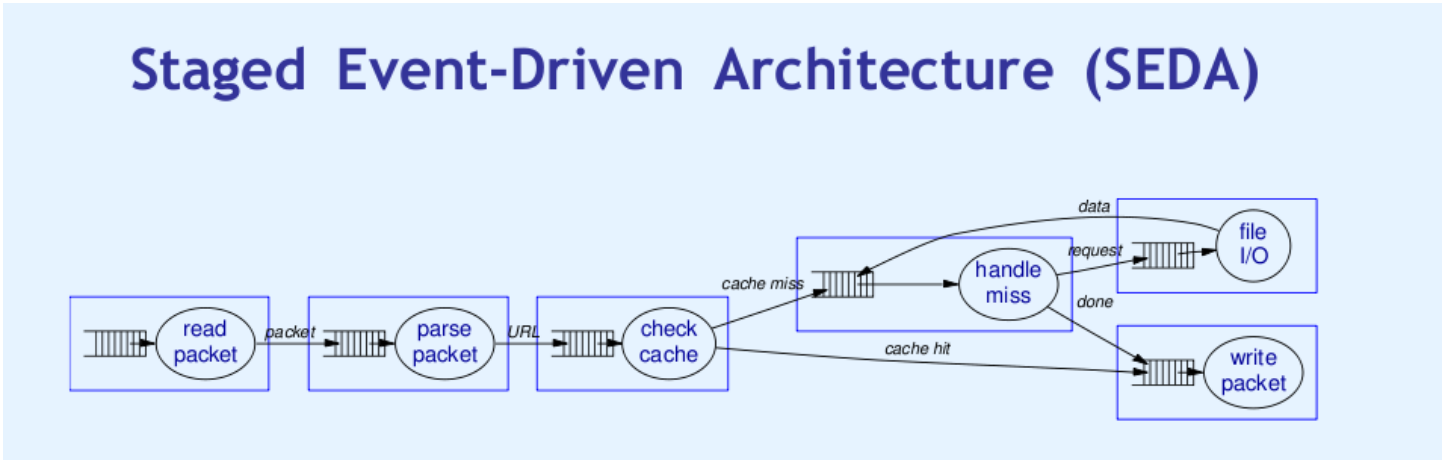
特点：
每任务一线程
直线式的编程
使用资源昂高，
context切换代价高，竞争锁昂贵
太多线程可能导致吞吐量下降，响应时间暴涨。

基于事件的并发模型



特点：
单线程处理事件
每个并发流实现为一个有限状态机
应用直接控制并发
负载增加的时候，吞吐量饱和
响应时间线性增长

二、**SEDA**架构



特点：

(1)服务通过**queue**分解成**stage:**

每个**stage**代表**FSM**的一个状态集合

Queue引入了控制边界

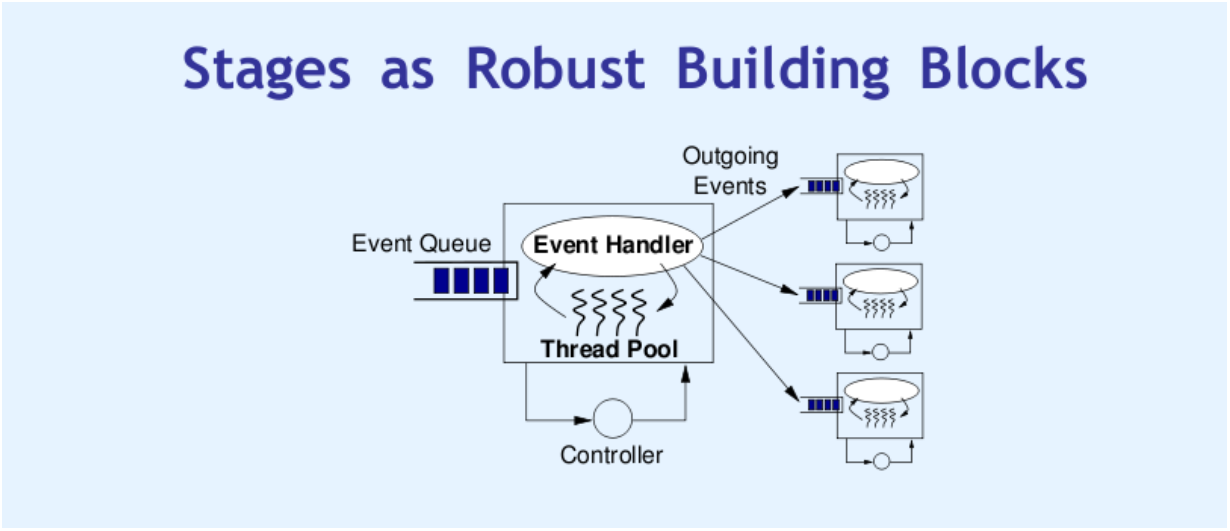
(2)使用线程池驱动**stage**的运行**:**

将事件处理同线程的创建和调度分离

Stage可以顺序或者并行执行

Stage可能在内部阻塞，给阻塞的**stage**分配较少的线程

1、**Stage-**可靠构建的基础



(1)应用逻辑封装到**Event Handler**

接收到许多事件，处理这些事件，然后派发事件加入其他**Stage**的**queue**

对**queue**和**threads**没有直接控制

Event queue吸纳过量的负载，有限的线程池维持并发

(2)Stage控制器

负责资源的分配和调度

控制派发给**Event Handler**的事件的数量和顺序

Event Handler可能在内部丢弃、过滤、重排序事件。

2、应用**=Stage**网络

(1)有限队列

入队可能失败，如果队列拒绝新项的话

阻塞在满溢的队列上来实现吸纳压力

通过丢弃事件来降低负载

(2) 队列将**Stage**的执行分解

引入了显式的控制边界

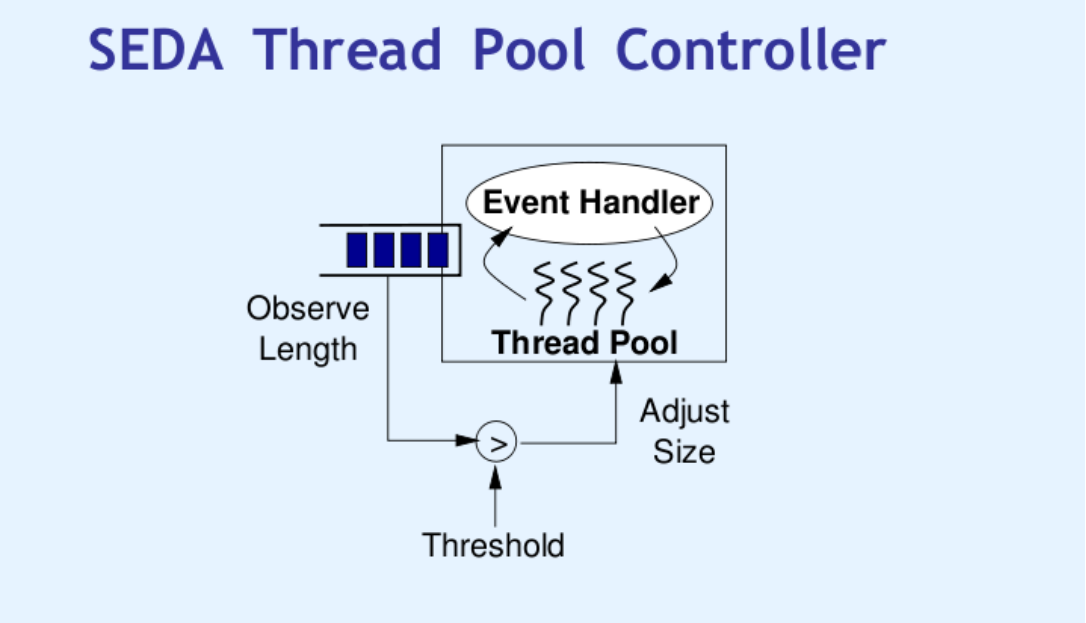
提供了隔离、模块化、独立的负载管理

(3)方便调试和**profile**

事件的投递可显
时间流可跟踪
通过监测**queue**的长度发现系统瓶颈

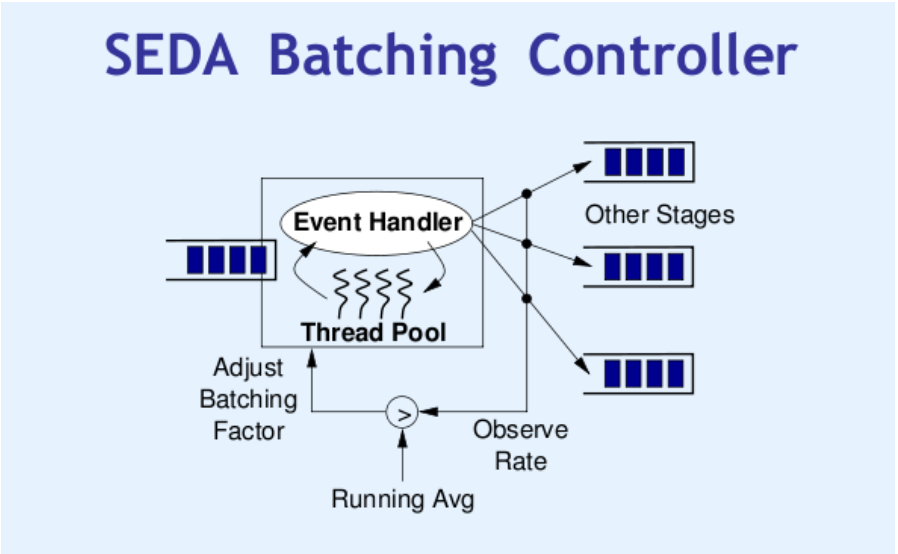
3、动态资源控制器

(1)、线程池管理器
目标： 决定**Stage**合理的并发程度
操作：
观察**queue**长度，如果超过阀值就添加线程
移除空闲线程



(2)、批量管理器
目的：低响应时间和高吞吐量的调度
操作：
Batching因子：**Stage**一次处理的消息数量
小的**batching**因子：低响应时间
大的**batching**因子：高吞吐量

尝试找到具有稳定吞吐量的最小的**batching**因子
观察**stage**的事件流出率
当吞吐量高的时候降低**batching**因子，低的时候增加



三、小结
SEDA主要还是为了解决传统并发模型的缺点，通过将服务器的处理划分各个**Stage**，利用**queue**连接起来形成一个**pipeline**

的处理链，并且在**Stage**中利用控制器进行资源的调控。资源的调度依据运行时的状态监视的数据来进行，从而形成一种反应控制的机制，而**stage**的划分也简化了编程，并且通过**queue**和每个**stage**的线程池来分担高并发请求并保持吞吐量和响应时间的平衡。简单来说，我看中的是服务器模型的清晰划分以及反应控制。