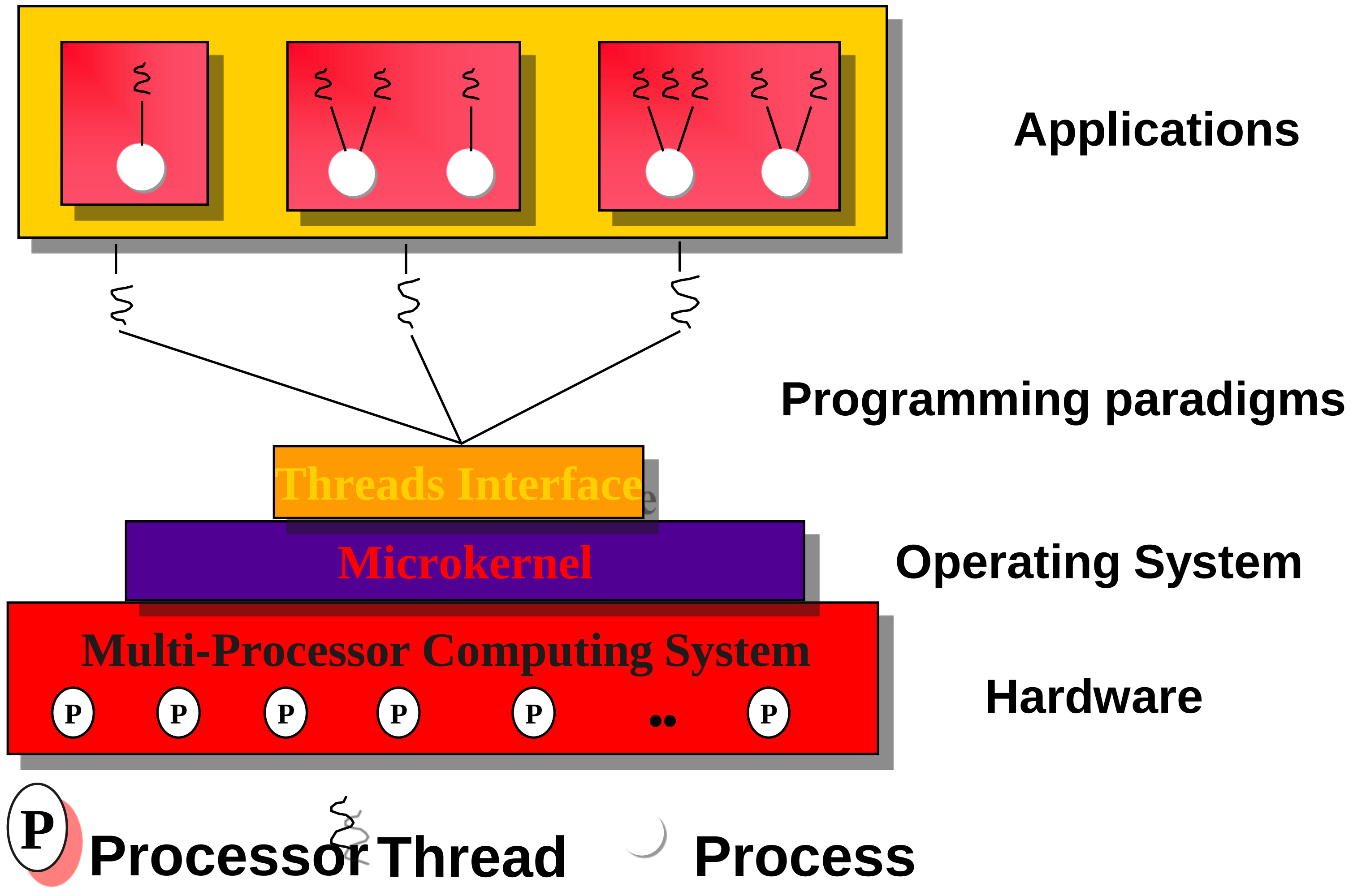
Lecture 5 Concurrent Computing

- Concurrency vs. Parallism
- Threading Model
- Thread Synchronization
- Java Thread API

Concurrent Computing Aechitecture



- **Parallel computing:** using multiple processors in parallel to solve problems more quickly than with a single processor
- Examples of parallel machines:
  - A **cluster computer** that contains multiple PCs combined together with a high speed network
  - A **shared memory multiprocessor** (SMP*) by connecting multiple processors to a single memory system
  - A **Chip Multi-Processor** (CMP) contains multiple processors (called cores) on a single chip
- Concurrent execution comes from desire for performance; unlike the inherent concurrency in a multi-user distributed system

* Technically, SMP stands for “Symmetric Multi-Processor”

Motivation for Concurrency

- Leverage hardware/software advances, e.g. multi-processors and OS thread support
- Increase performance, e.g., overlap computation and communication
- Improve response-time, e.g., GUIs and network servers
- Simplify program structure, e.g., synchronous vs. asynchronous network IPC

Definitions

- Concurrency
  - “Logically" simultaneous processing
  - Does not imply multiple processing elements
- Parallelism
  - “Physically" simultaneous processing
  - Involves multiple processing elements and/or independent device operations

Both concurrency and parallelism require controlled access to shared resources, e.g. I/O devices, files, database records, in-core data structures, consoles, etc.

Flynn's Classical Taxonomy

- Single Instruction, Single Data streams (SISD)—your single-core uni-processor PC
- Single Instruction, Multiple Data streams (SIMD)—special purpose low-granularity multi-processor m/c w/ a single control unit relaying the same instruction to all processors (w/ different data) every cc (e.g., nVIDIA graphic co-processor w/ 1000’s of simple cores)
- Multiple Instruction, Single Data streams (MISD)—pipelining is a major example
- Multiple Instruction, Multiple Data streams (MIMD)—the most prevalent model. SPMD (Single Program Multiple Data) is a very useful subset. Note that this is v. different from SIMD.

- Data Parallelism: SIMD and SPMD fall into this category
- Functional Parallelism: MISD falls into this category
- MIMD can incorporates both data and functional parallelisms (the latter at either instruction level—different instrs. being executed across the processors at any time, or at the high-level function space)

Concurrent Computing Model

Multiprocess vs. Multithread

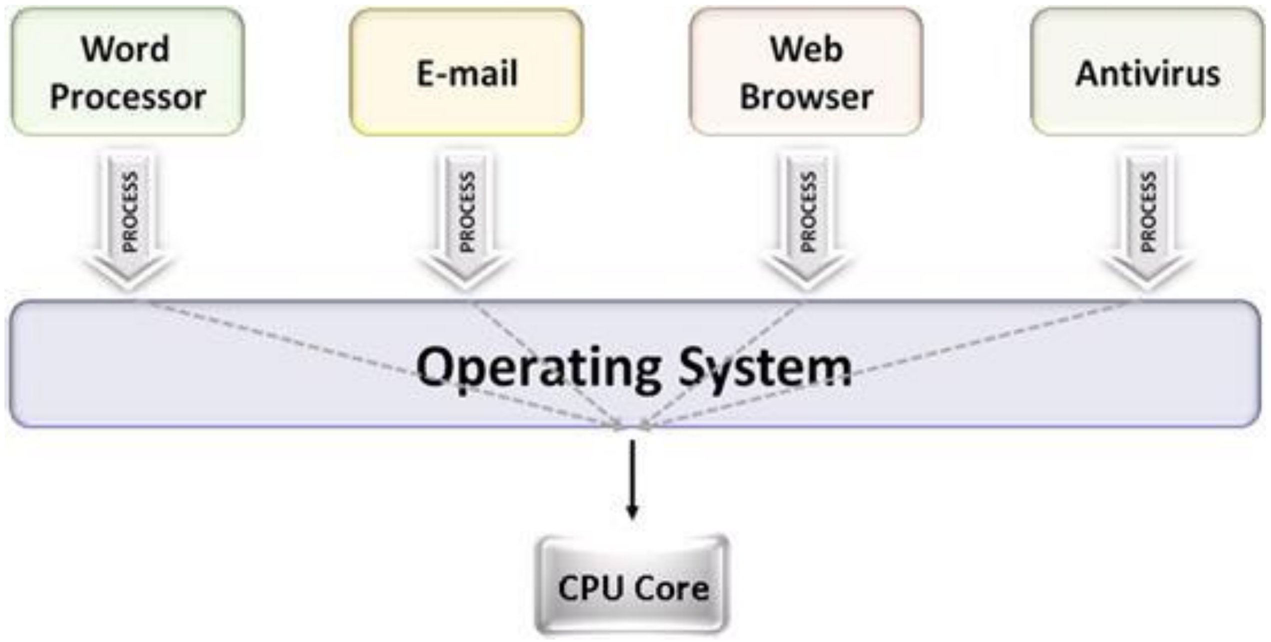
- Multiprocess

  Multiple tasks or processes share common system resources such as CPU, I/O, etc.

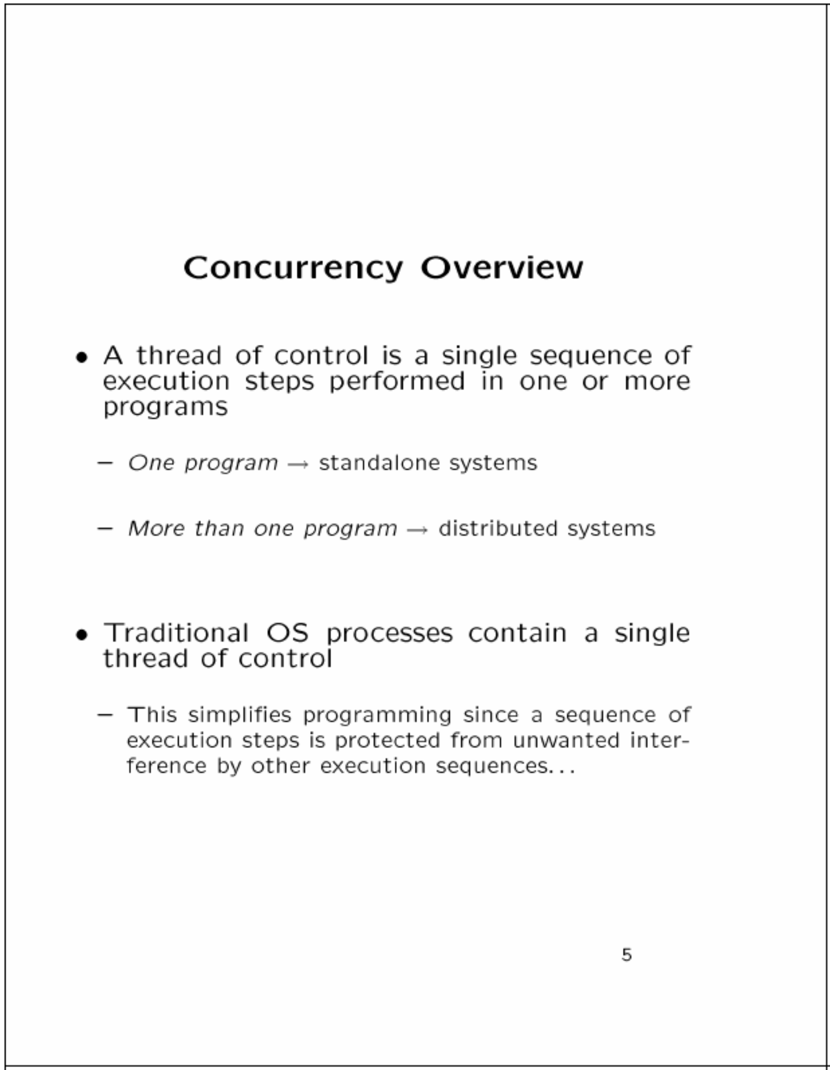
- Multithread

  Multiple execution units within a process share system resources.

Multiprocess --- single core



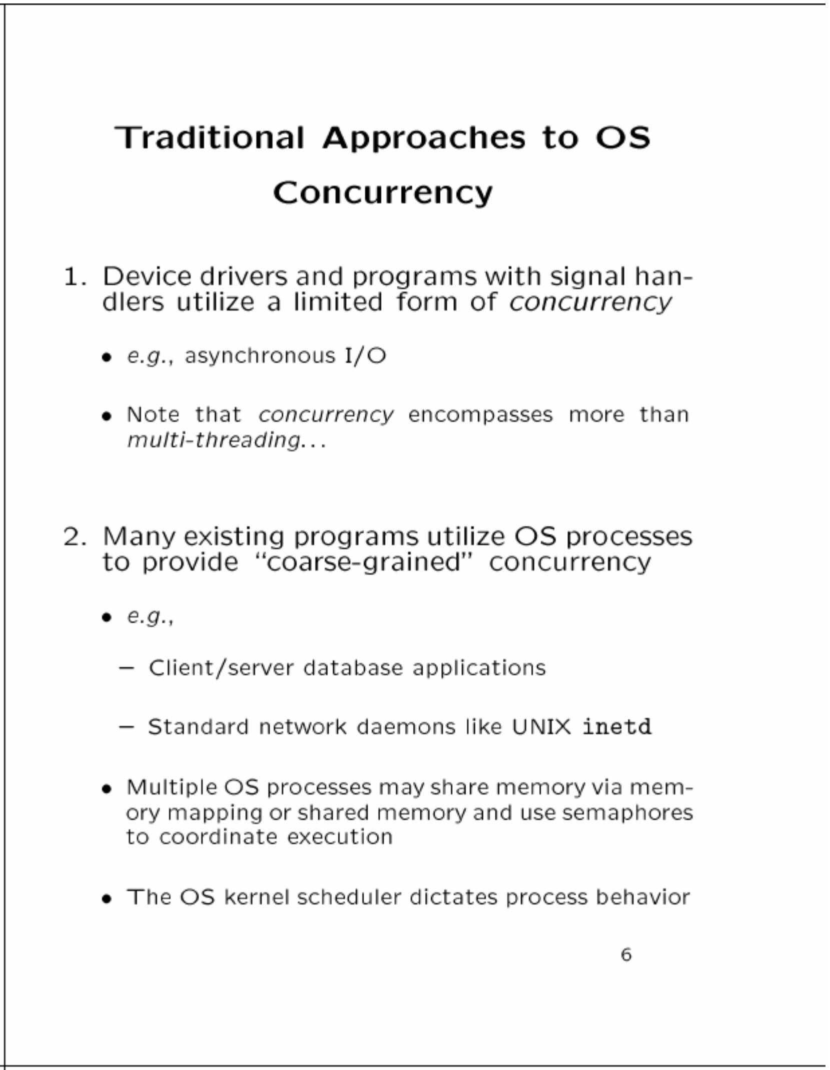
Software Architecture & Design Pattern



Concurrency Overview

- A thread of control is a single sequence of execution steps performed in one or more programs
  - *One program* → standalone systems
  - *More than one program* → distributed systems
- Traditional OS processes contain a single thread of control
  - This simplifies programming since a sequence of execution steps is protected from unwanted interference by other execution sequences...

5

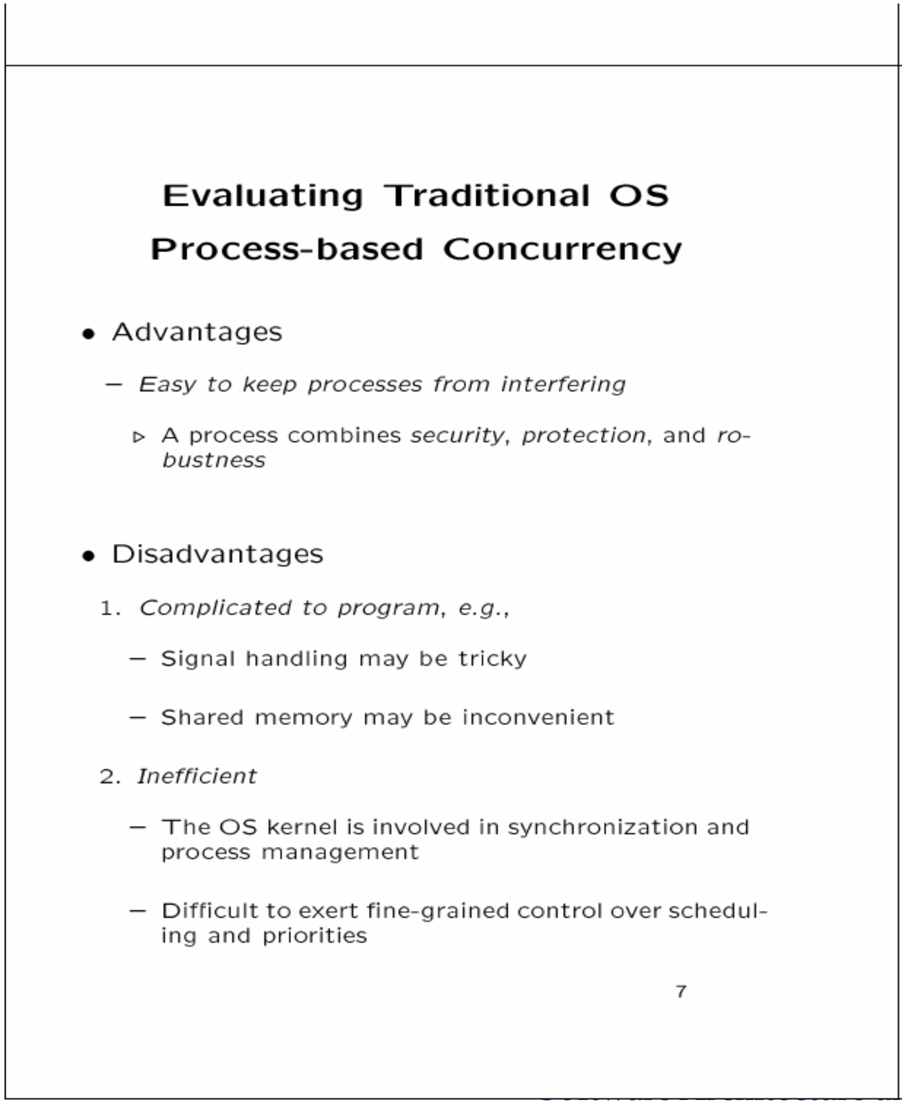


Traditional Approaches to OS Concurrency

1. Device drivers and programs with signal handlers utilize a limited form of *concurrency*
   - *e.g.*, asynchronous I/O
   - Note that *concurrency* encompasses more than *multi-threading*...
2. Many existing programs utilize OS processes to provide "coarse-grained" concurrency
   - *e.g.*,
     - Client/server database applications
     - Standard network daemons like UNIX `inetd`
   - Multiple OS processes may share memory via memory mapping or shared memory and use semaphores to coordinate execution
   - The OS kernel scheduler dictates process behavior

6

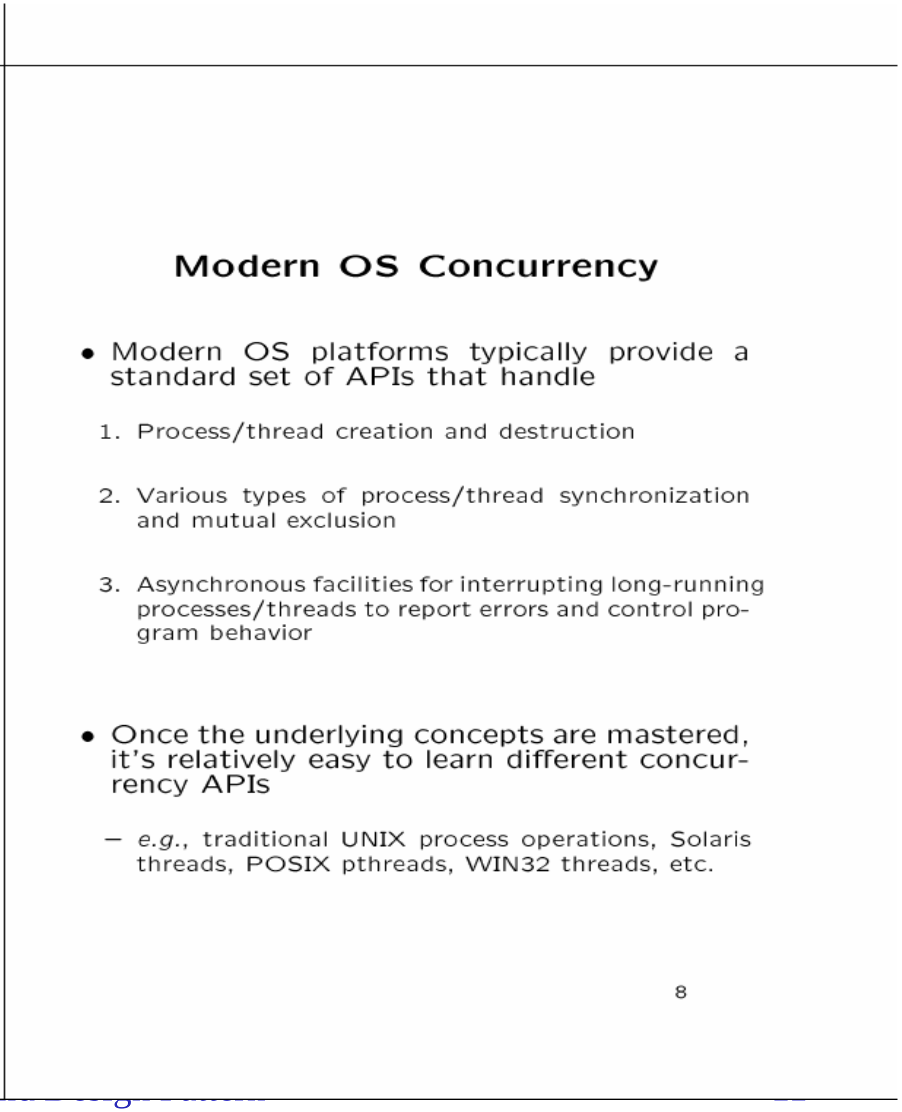
Software Architecture & Design Pattern



Evaluating Traditional OS Process-based Concurrency

- Advantages
  - *Easy to keep processes from interfering*
    - ▹ A process combines *security*, *protection*, and *robustness*
- Disadvantages
  1. *Complicated to program*, *e.g.*,
     - Signal handling may be tricky
     - Shared memory may be inconvenient
  2. *Inefficient*
     - The OS kernel is involved in synchronization and process management
     - Difficult to exert fine-grained control over scheduling and priorities

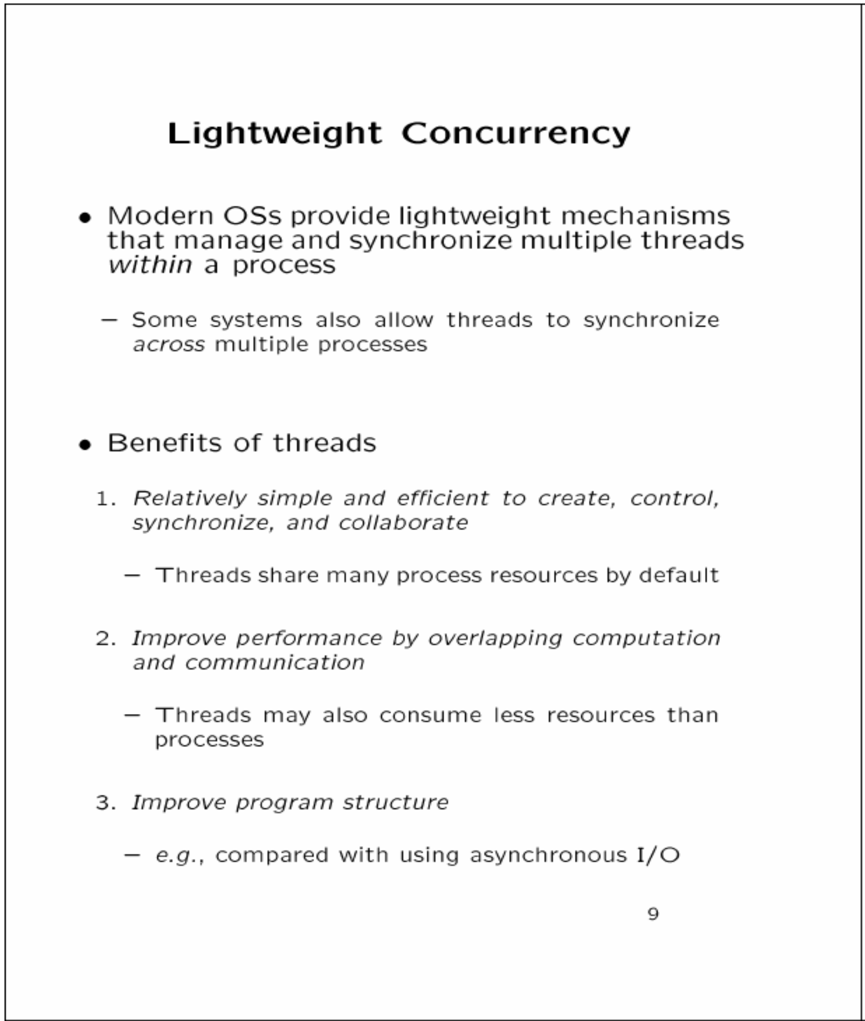
7



Modern OS Concurrency

- Modern OS platforms typically provide a standard set of APIs that handle
  1. Process/thread creation and destruction
  2. Various types of process/thread synchronization and mutual exclusion
  3. Asynchronous facilities for interrupting long-running processes/threads to report errors and control program behavior
- Once the underlying concepts are mastered, it's relatively easy to learn different concurrency APIs
  - *e.g.*, traditional UNIX process operations, Solaris threads, POSIX pthreads, WIN32 threads, etc.

8



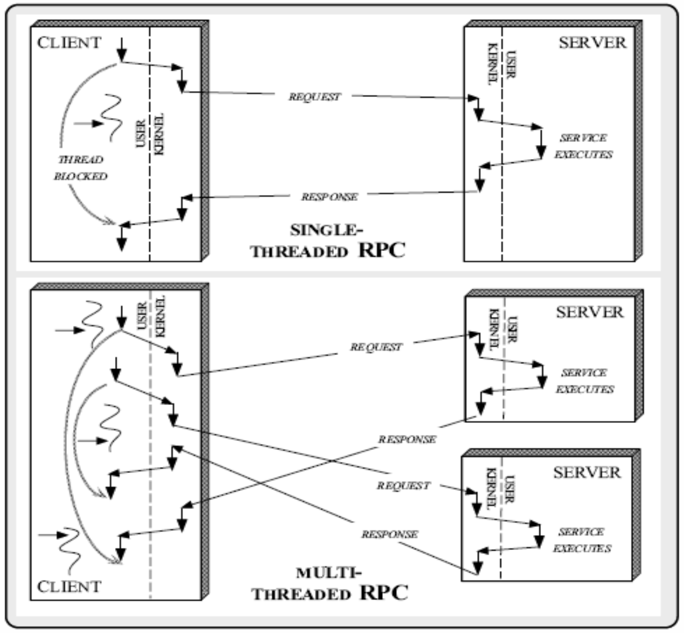
Lightweight Concurrency

- Modern OSs provide lightweight mechanisms that manage and synchronize multiple threads *within* a process
  - Some systems also allow threads to synchronize *across* multiple processes

- Benefits of threads
  1. *Relatively simple and efficient to create, control, synchronize, and collaborate*
     - Threads share many process resources by default
  2. *Improve performance by overlapping computation and communication*
     - Threads may also consume less resources than processes
  3. *Improve program structure*
     - *e.g.*, compared with using asynchronous I/O

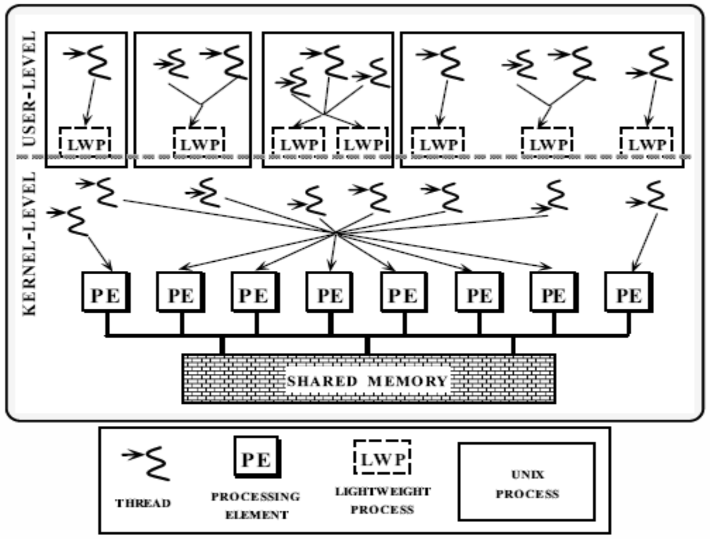
9

Single-threaded vs. Multi-threaded RPC



10

Hardware and OS Concurrency Support



- Modern OS platforms like Solaris provide kernel support for multi-threading

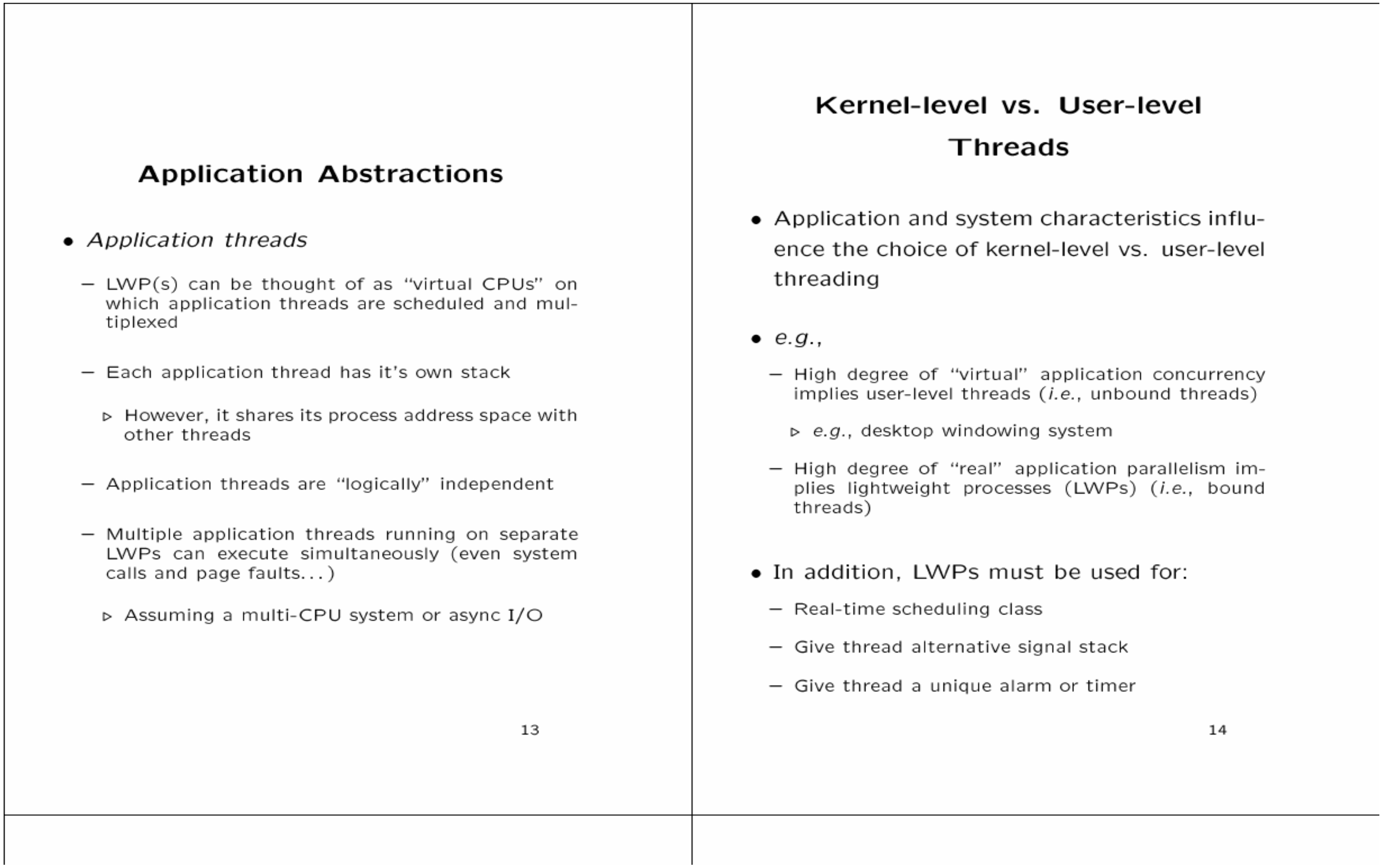
11

Kernel Abstractions

- *Kernel threads*
  - The "fundamental scheduling entities" executed by the PE(s)
  - Operate in kernel space
  - Kernel-resident subsystems use kernel threads directly
- *Lightweight processes* (LWP)
  - Every LWP is associated with one kernel thread
    - ▷ *i.e.*, 1-to–1 mapping between kernel thread and LWP per-process
  - Not every kernel thread has an LWP
    - ▷ "System threads" (*e.g.*, pagedaemon, NFS daemon, and the callout thread) have only a kernel thread

12

Software Architecture & Design Pattern



Application Abstractions

- *Application threads*
  - LWP(s) can be thought of as "virtual CPUs" on which application threads are scheduled and multiplexed
  - Each application thread has it's own stack
    - ▷ However, it shares its process address space with other threads
  - Application threads are "logically" independent
  - Multiple application threads running on separate LWPs can execute simultaneously (even system calls and page faults...)
    - ▷ Assuming a multi-CPU system or async I/O

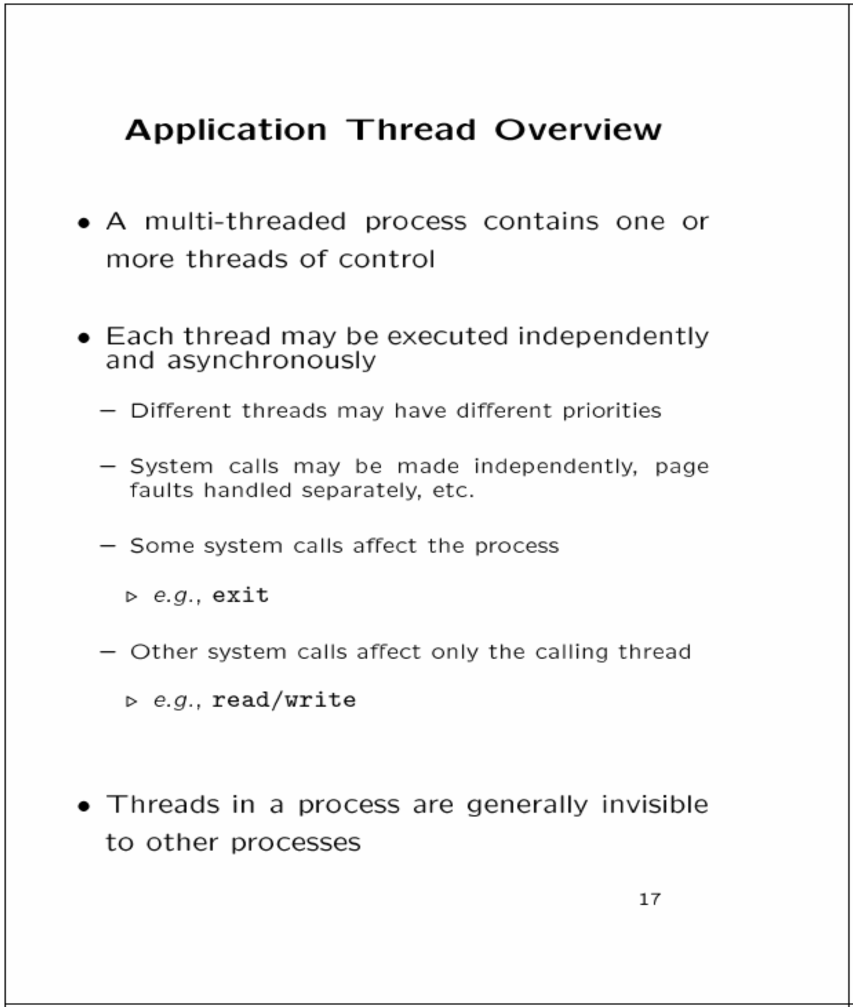
13

Kernel-level vs. User-level Threads

- Application and system characteristics influence the choice of kernel-level vs. user-level threading
- *e.g.*,
  - High degree of "virtual" application concurrency implies user-level threads (*i.e.*, unbound threads)
    - ▷ *e.g.*, desktop windowing system
  - High degree of "real" application parallelism implies lightweight processes (LWPs) (*i.e.*, bound threads)
- In addition, LWPs must be used for:
  - Real-time scheduling class
  - Give thread alternative signal stack
  - Give thread a unique alarm or timer

14

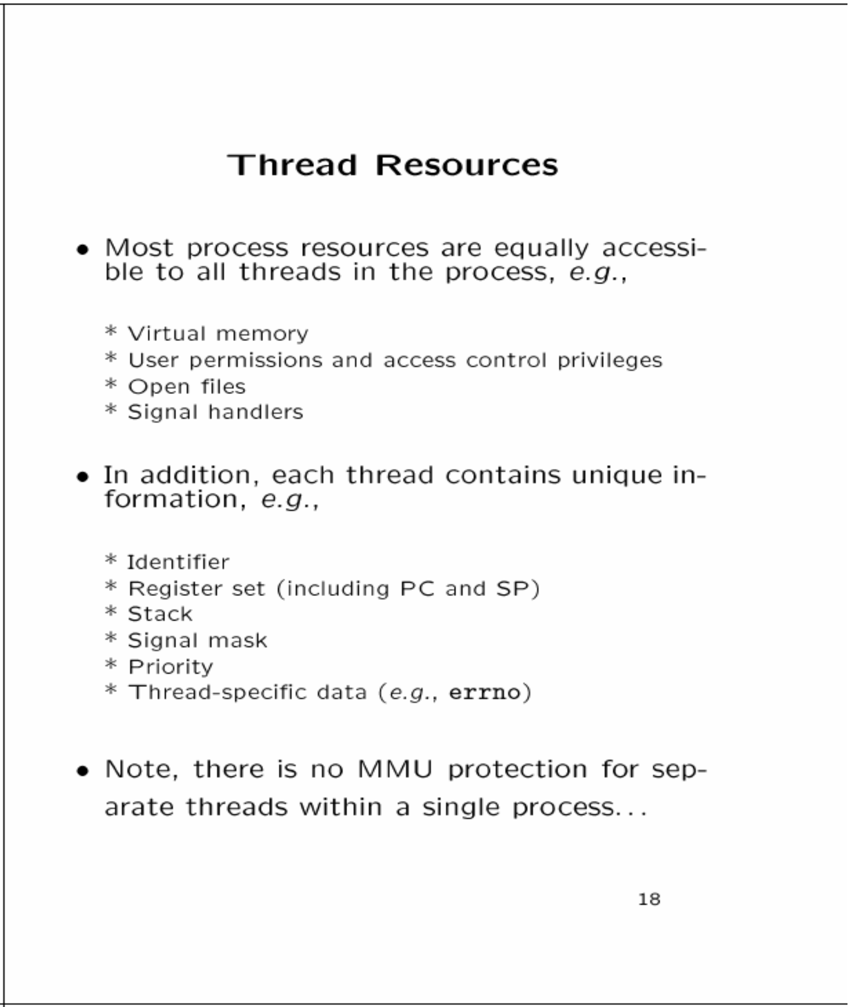
Software Architecture & Design Pattern



Application Thread Overview

- A multi-threaded process contains one or more threads of control
- Each thread may be executed independently and asynchronously
  - Different threads may have different priorities
  - System calls may be made independently, page faults handled separately, etc.
  - Some system calls affect the process
    - ▷ *e.g.*, **`exit`**
  - Other system calls affect only the calling thread
    - ▷ *e.g.*, **`read/write`**
- Threads in a process are generally invisible to other processes

17



Thread Resources

- Most process resources are equally accessible to all threads in the process, *e.g.*,

  * Virtual memory
  * User permissions and access control privileges
  * Open files
  * Signal handlers

- In addition, each thread contains unique information, *e.g.*,

  * Identifier
  * Register set (including PC and SP)
  * Stack
  * Signal mask
  * Priority
  * Thread-specific data (*e.g.*, **`errno`**)

- Note, there is no MMU protection for separate threads within a single process...

18

Java Threading Model

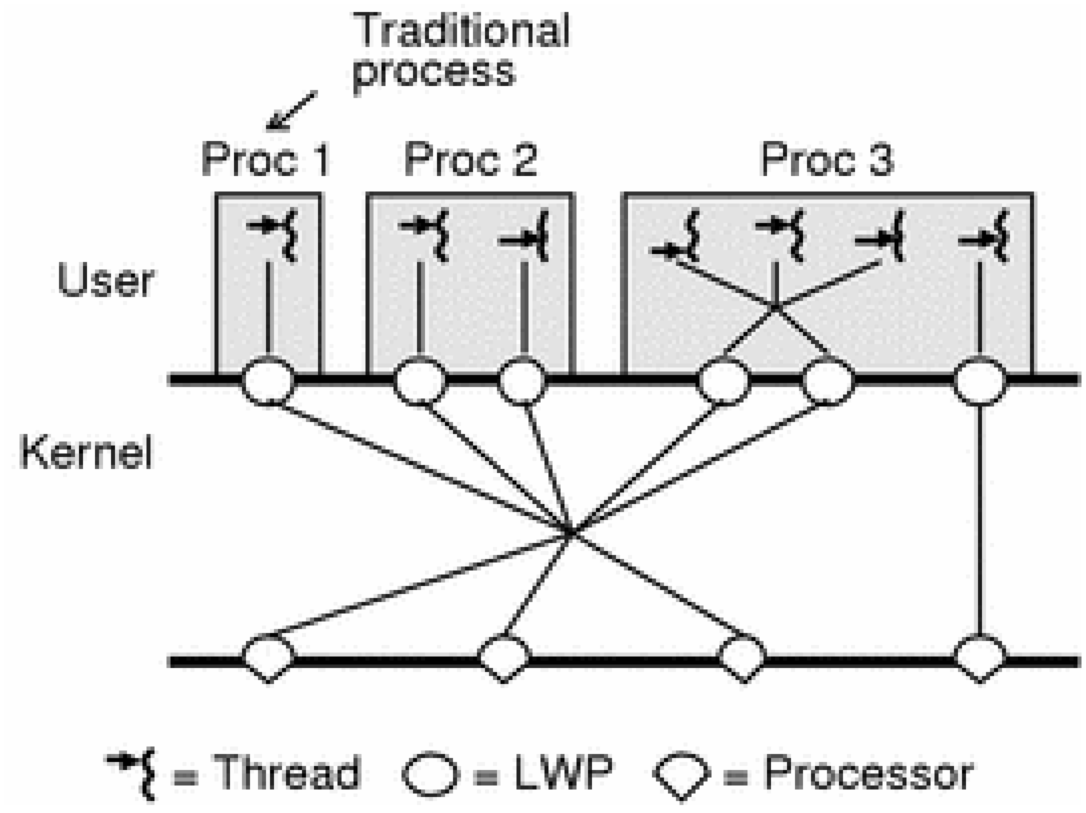
Two-level threads: user thread (java thread class)

kernel thread (O/S kernel support)

Thread model

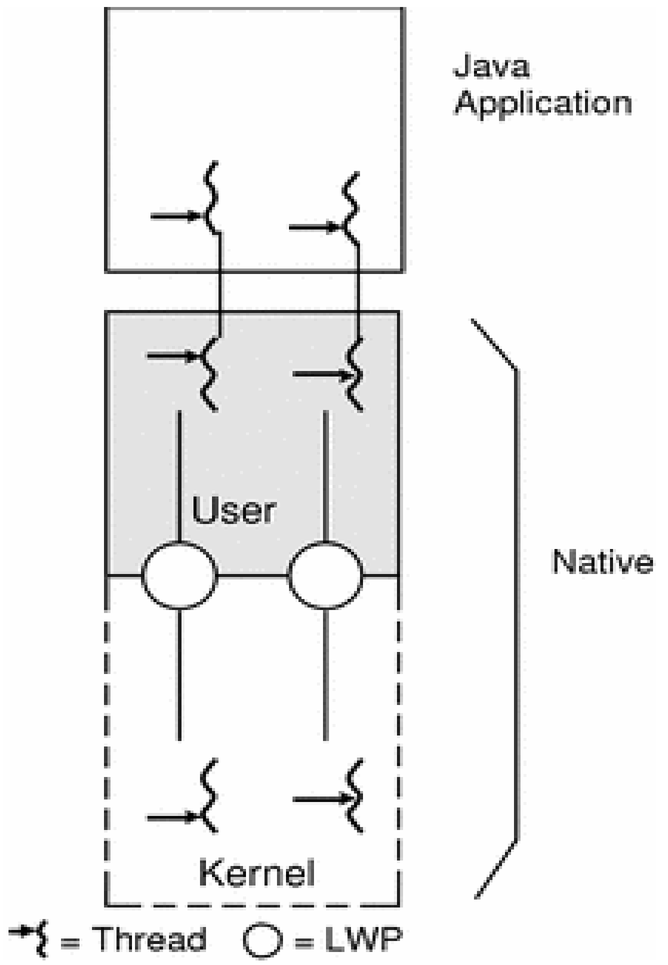
- one-to-one
- many-to-one
- many-to-many

Solaris Two-level Threading Structure



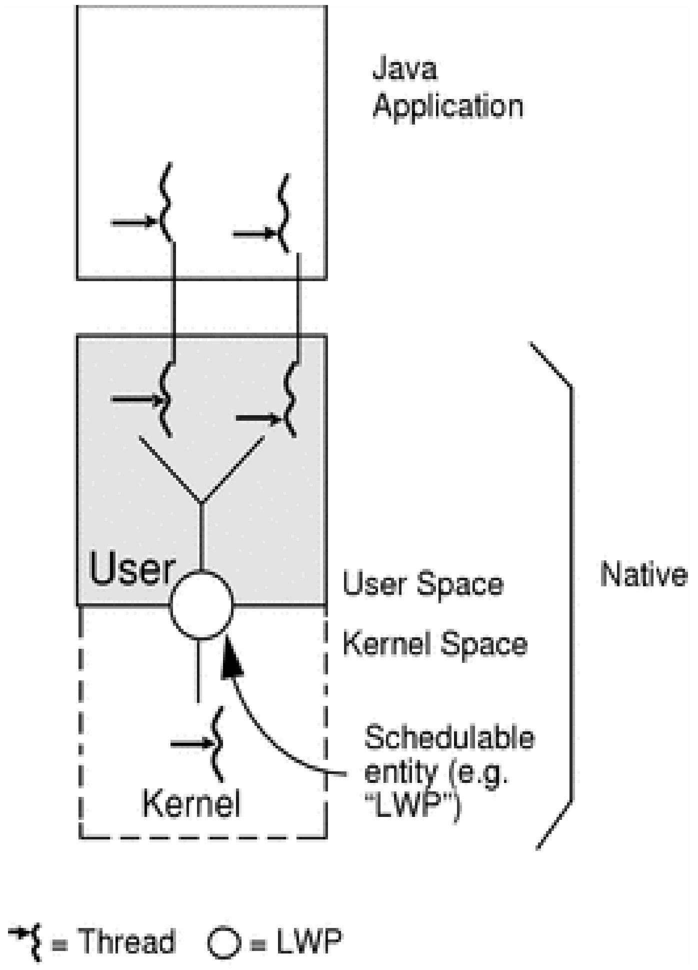
One-to-One Model

- one user thread is mapped
  to one kernel thread
- high concurrency
- may overload the kernel



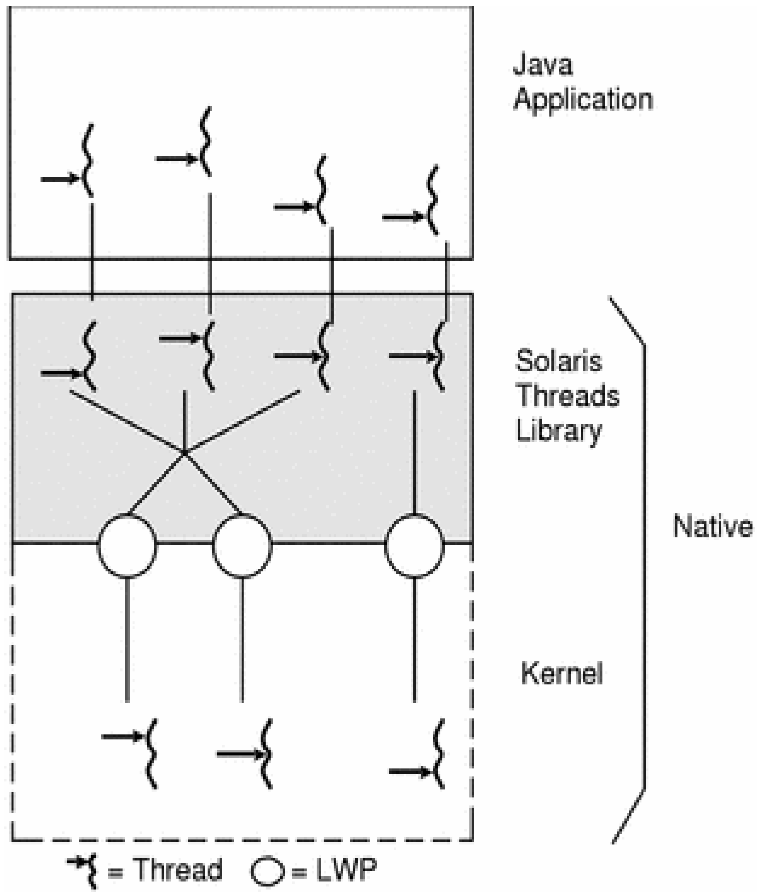
Many-to-One Model

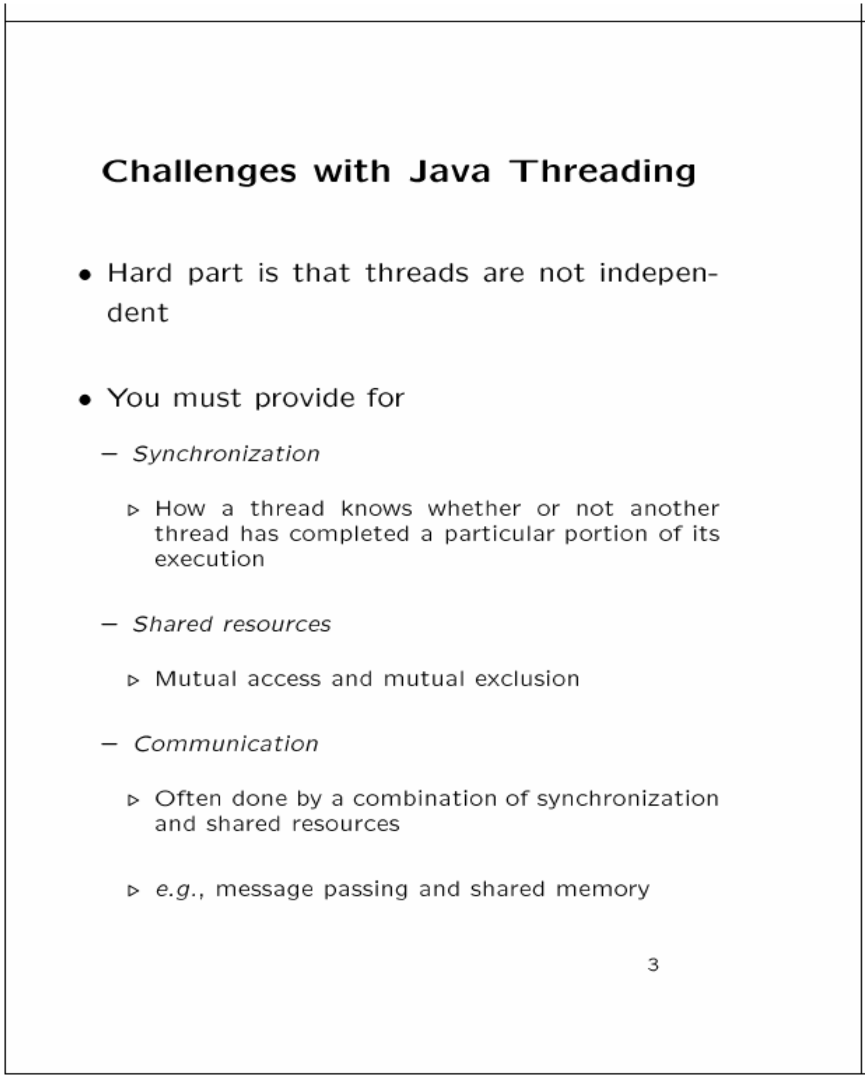
- many user threads are mapped to one kernel thread
- low throughput
- not quite used now



Many-to-Many Model

- M user threads are mapped to N kernel threads
- use resources efficiently
- scheduling issue

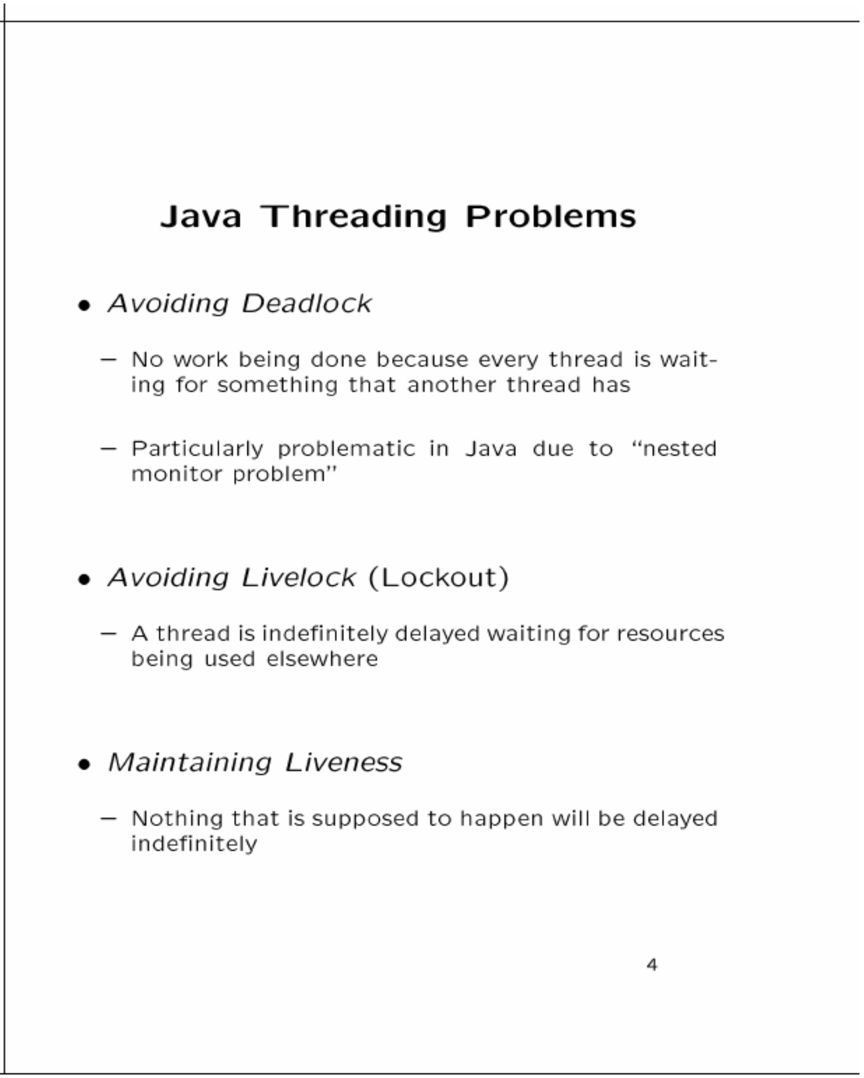




Challenges with Java Threading

- Hard part is that threads are not independent
- You must provide for
  - *Synchronization*
    - ▹ How a thread knows whether or not another thread has completed a particular portion of its execution
  - *Shared resources*
    - ▹ Mutual access and mutual exclusion
  - *Communication*
    - ▹ Often done by a combination of synchronization and shared resources
    - ▹ *e.g.*, message passing and shared memory

3

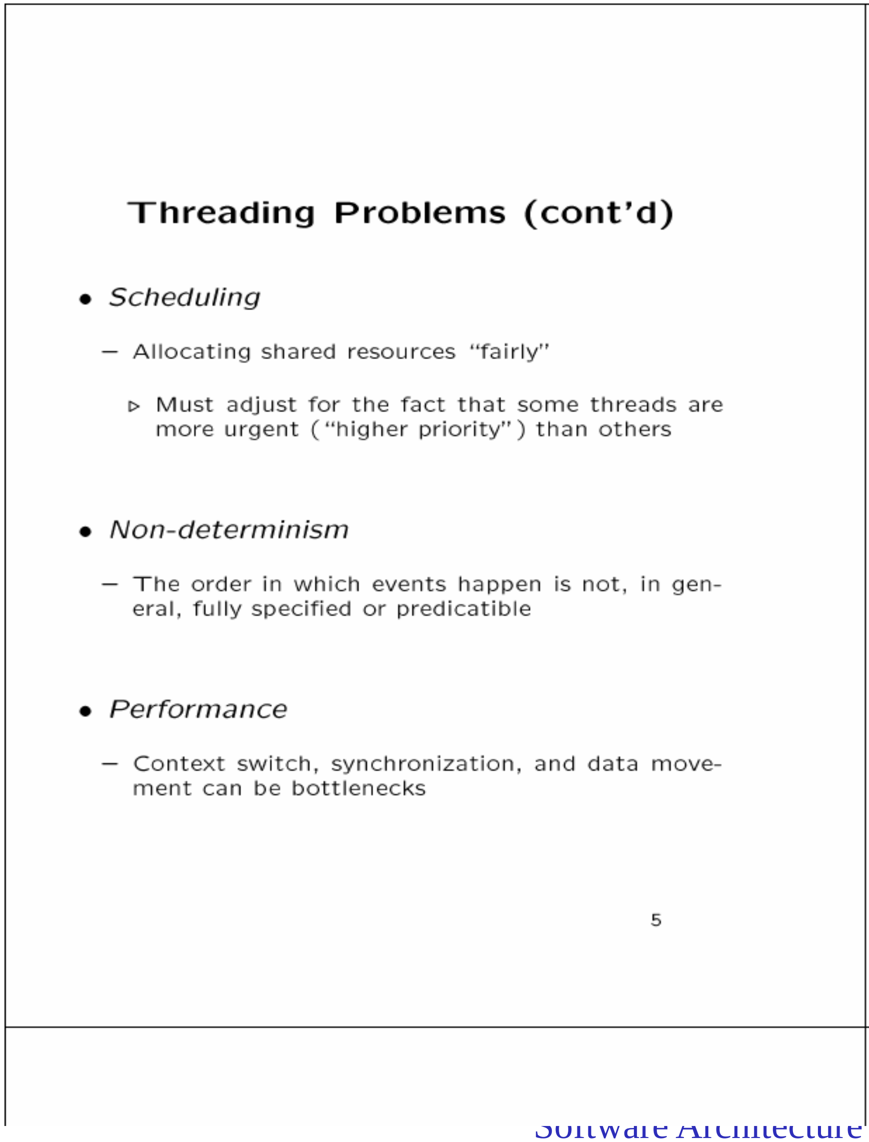


Java Threading Problems

- *Avoiding Deadlock*
  - No work being done because every thread is waiting for something that another thread has
  - Particularly problematic in Java due to “nested monitor problem”
- *Avoiding Livelock* (Lockout)
  - A thread is indefinitely delayed waiting for resources being used elsewhere
- *Maintaining Liveness*
  - Nothing that is supposed to happen will be delayed indefinitely

4

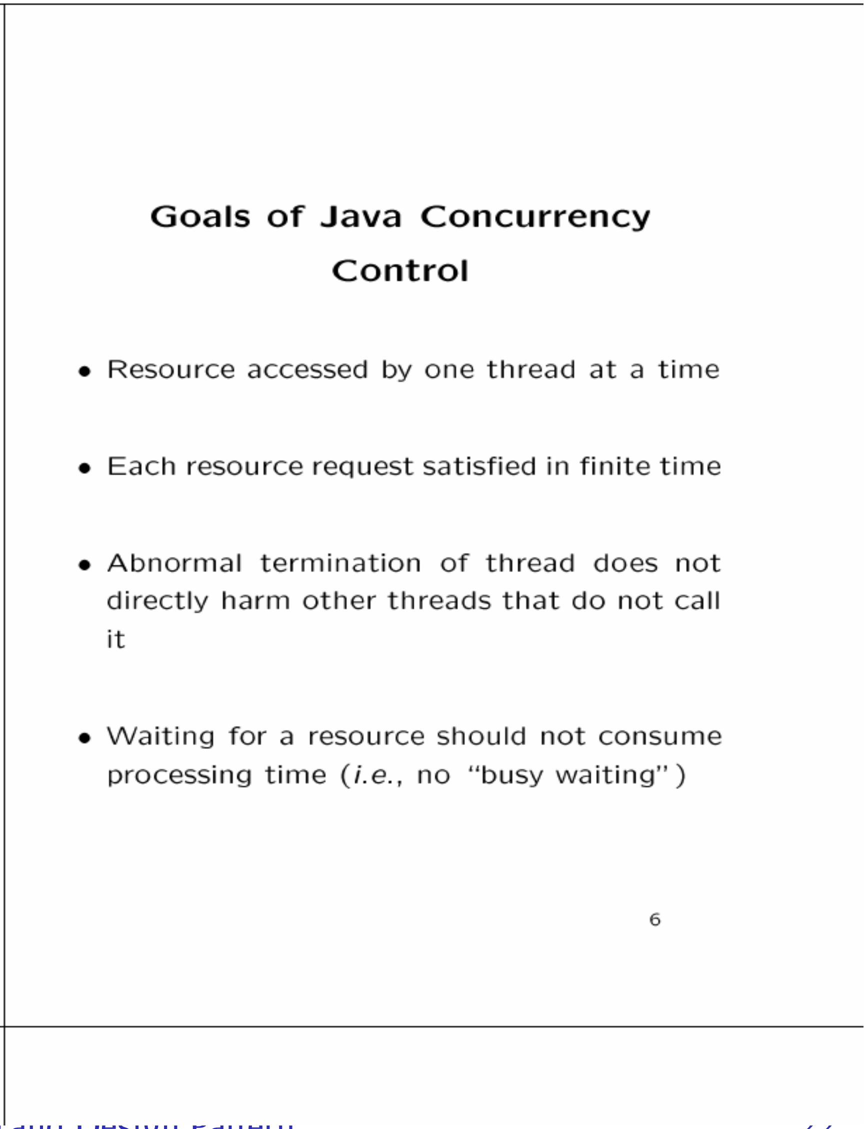
Software Architecture & Design Pattern



Threading Problems (cont'd)

- *Scheduling*
  - Allocating shared resources "fairly"
    - ▷ Must adjust for the fact that some threads are more urgent ("higher priority") than others
- *Non-determinism*
  - The order in which events happen is not, in general, fully specified or predicatible
- *Performance*
  - Context switch, synchronization, and data movement can be bottlenecks

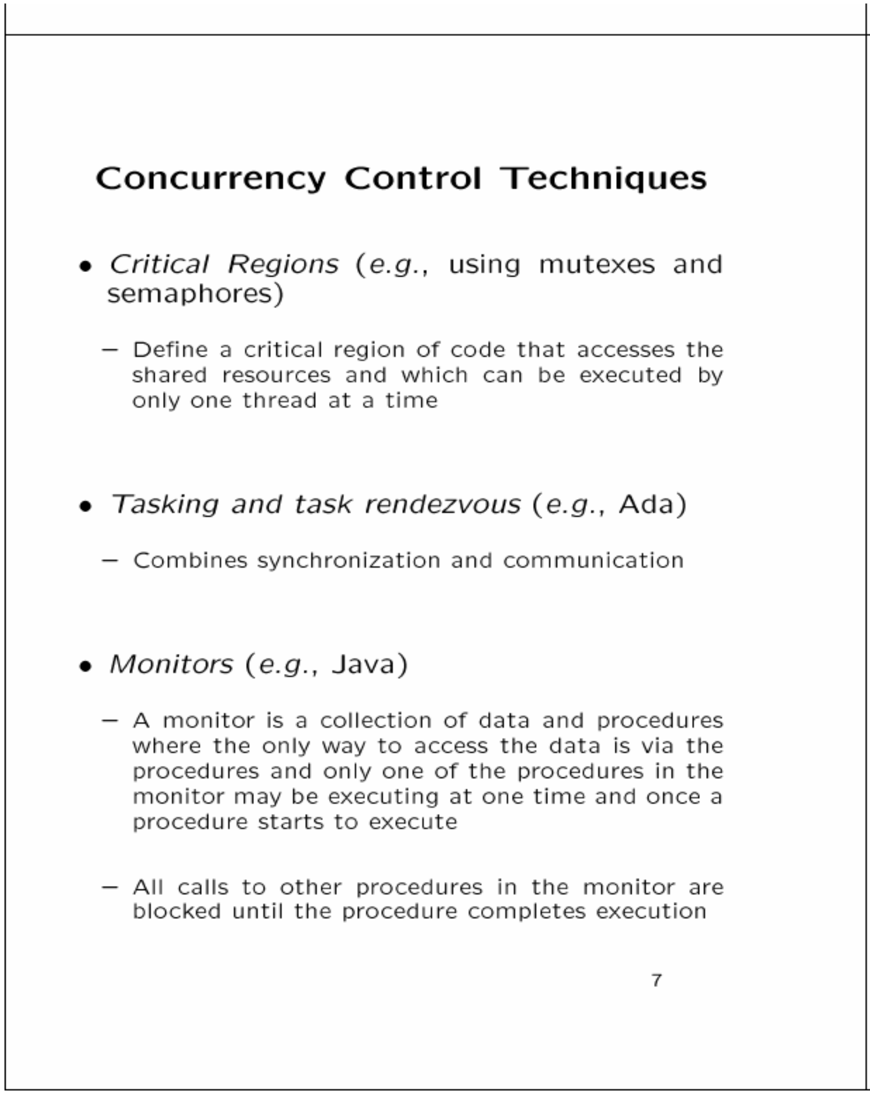
5



Goals of Java Concurrency Control

- Resource accessed by one thread at a time
- Each resource request satisfied in finite time
- Abnormal termination of thread does not directly harm other threads that do not call it
- Waiting for a resource should not consume processing time (*i.e.*, no "busy waiting")

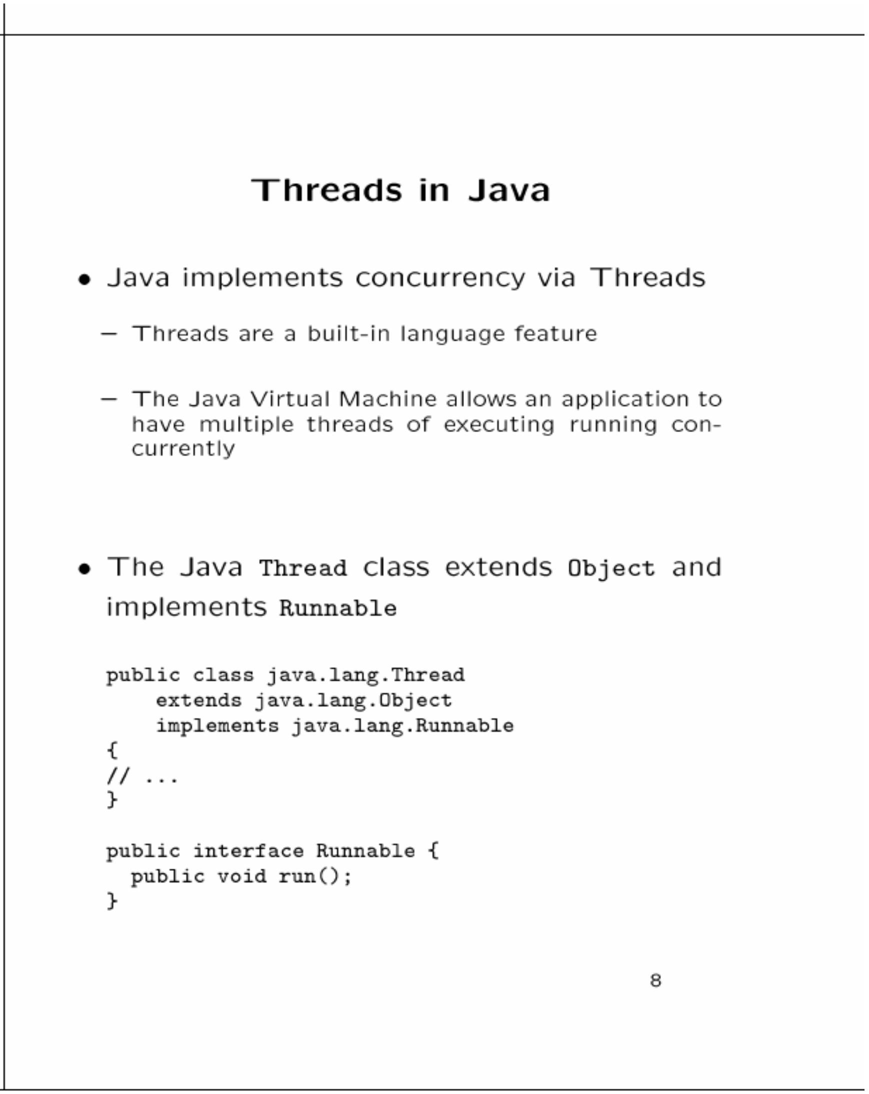
6



Concurrency Control Techniques

- *Critical Regions* (*e.g.*, using mutexes and semaphores)
  - Define a critical region of code that accesses the shared resources and which can be executed by only one thread at a time
- *Tasking and task rendezvous* (*e.g.*, Ada)
  - Combines synchronization and communication
- *Monitors* (*e.g.*, Java)
  - A monitor is a collection of data and procedures where the only way to access the data is via the procedures and only one of the procedures in the monitor may be executing at one time and once a procedure starts to execute
  - All calls to other procedures in the monitor are blocked until the procedure completes execution

7



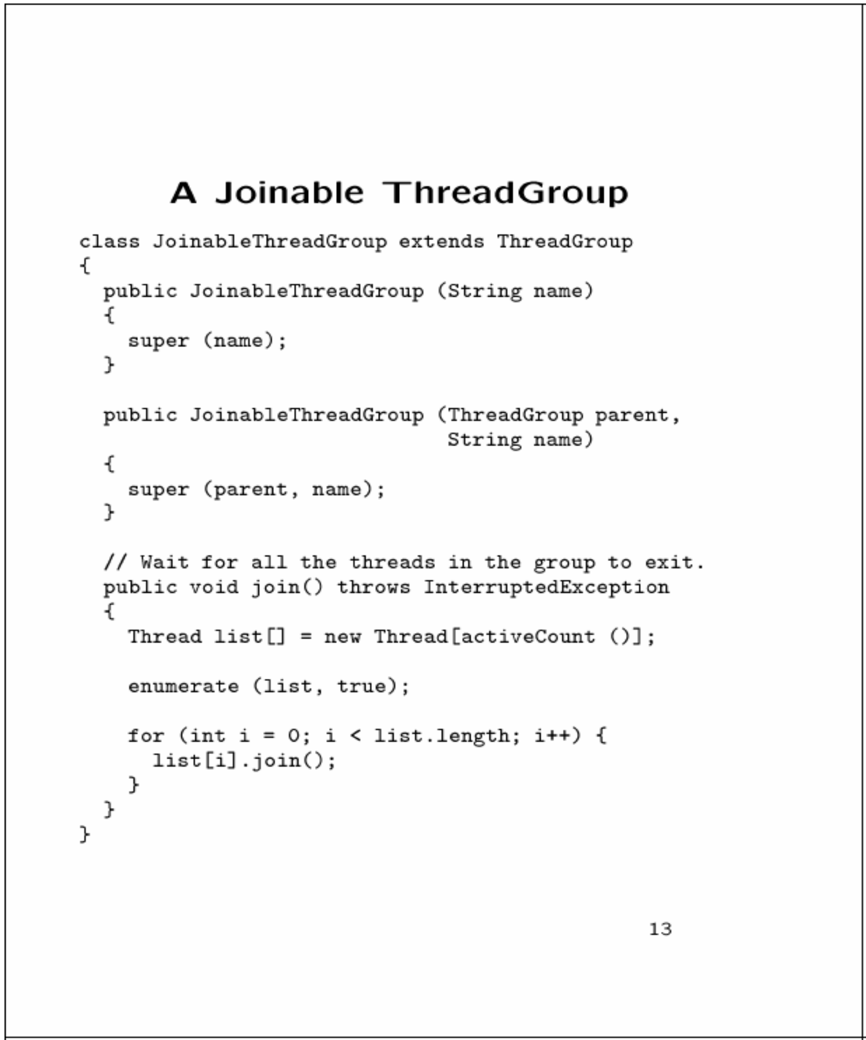
Threads in Java

- Java implements concurrency via Threads
  - Threads are a built-in language feature
  - The Java Virtual Machine allows an application to have multiple threads of executing running concurrently
- The Java `Thread` class extends `Object` and implements `Runnable`

```
public class java.lang.Thread
    extends java.lang.Object
    implements java.lang.Runnable
{
// ...
}

public interface Runnable {
  public void run();
}
```

8

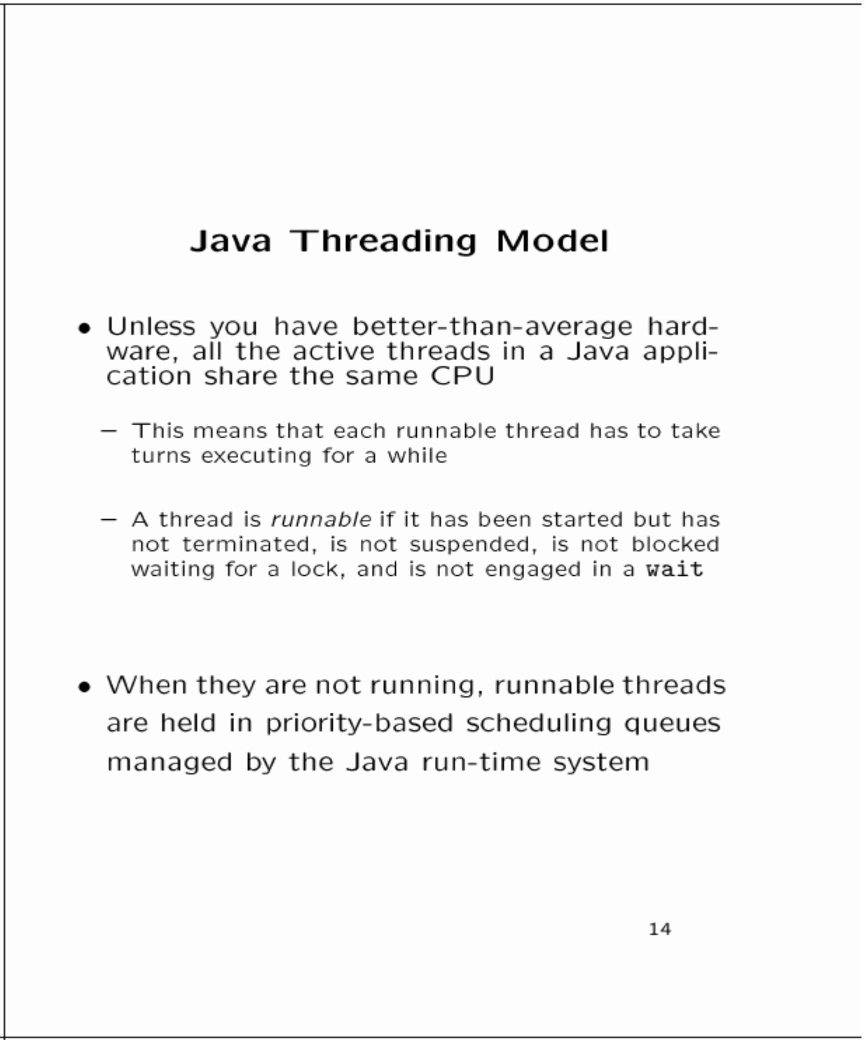
Software Architecture & Design Pattern



A Joinable ThreadGroup

```
class JoinableThreadGroup extends ThreadGroup
{
  public JoinableThreadGroup (String name)
  {
    super (name);
  }

  public JoinableThreadGroup (ThreadGroup parent,
                              String name)
  {
    super (parent, name);
  }

  // Wait for all the threads in the group to exit.
  public void join() throws InterruptedException
  {
    Thread list[] = new Thread[activeCount ()];

    enumerate (list, true);

    for (int i = 0; i < list.length; i++) {
      list[i].join();
    }
  }
}
```

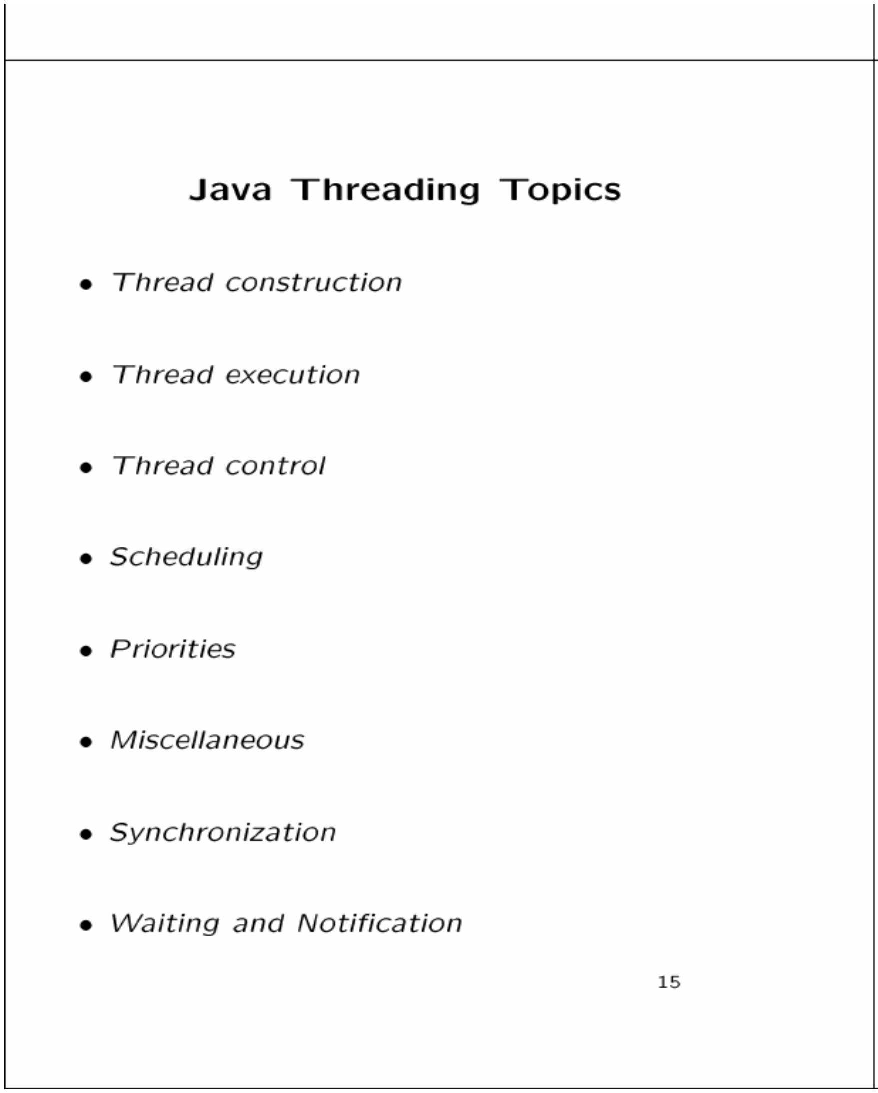
13



Java Threading Model

- Unless you have better-than-average hardware, all the active threads in a Java application share the same CPU
  - This means that each runnable thread has to take turns executing for a while
  - A thread is *runnable* if it has been started but has not terminated, is not suspended, is not blocked waiting for a lock, and is not engaged in a **wait**
- When they are not running, runnable threads are held in priority-based scheduling queues managed by the Java run-time system

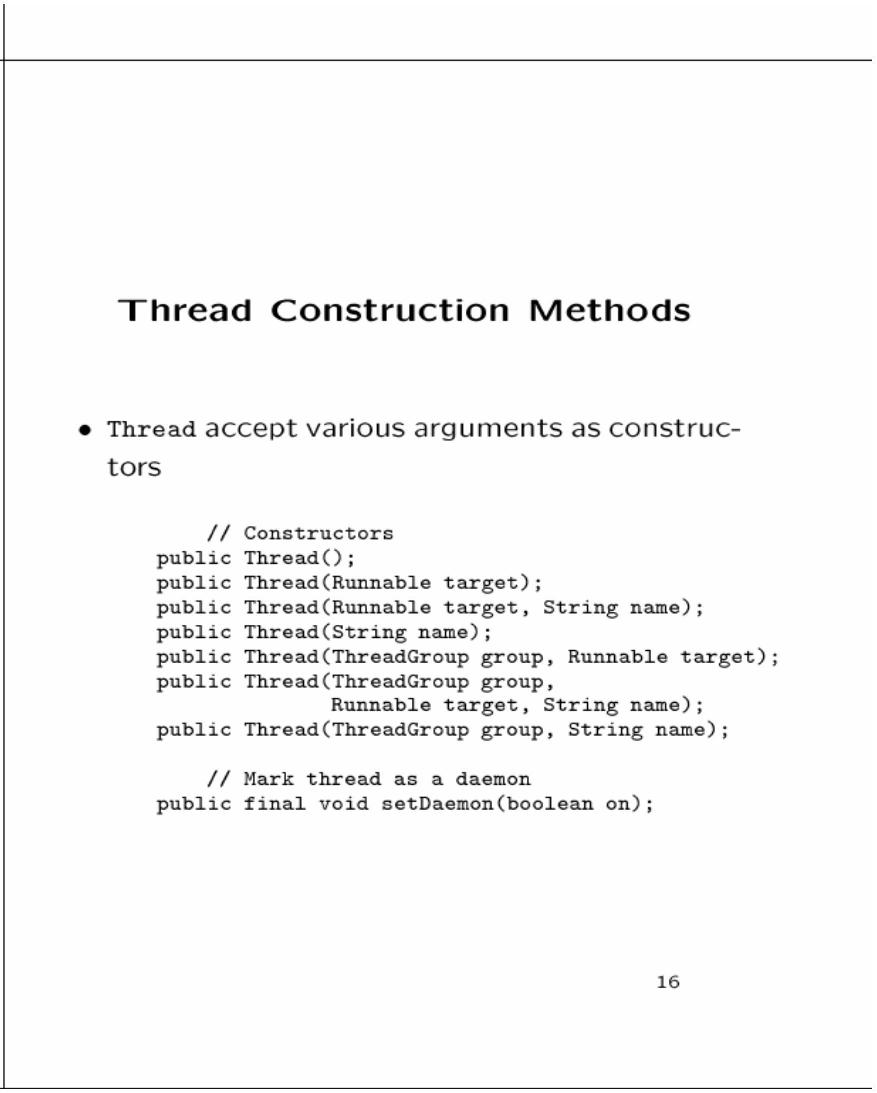
14



Java Threading Topics

- *Thread construction*
- *Thread execution*
- *Thread control*
- *Scheduling*
- *Priorities*
- *Miscellaneous*
- *Synchronization*
- *Waiting and Notification*

15



Thread Construction Methods

- `Thread` accept various arguments as constructors

```
    // Constructors
public Thread();
public Thread(Runnable target);
public Thread(Runnable target, String name);
public Thread(String name);
public Thread(ThreadGroup group, Runnable target);
public Thread(ThreadGroup group,
              Runnable target, String name);
public Thread(ThreadGroup group, String name);

    // Mark thread as a daemon
public final void setDaemon(boolean on);
```

16

并发计算架构一个主要的应用领域：

高并发高负载网站

- 用户数庞大
- 峰值高负载
- 同时读写/数据库
- 性能瓶颈
- 采用支持高并发技术

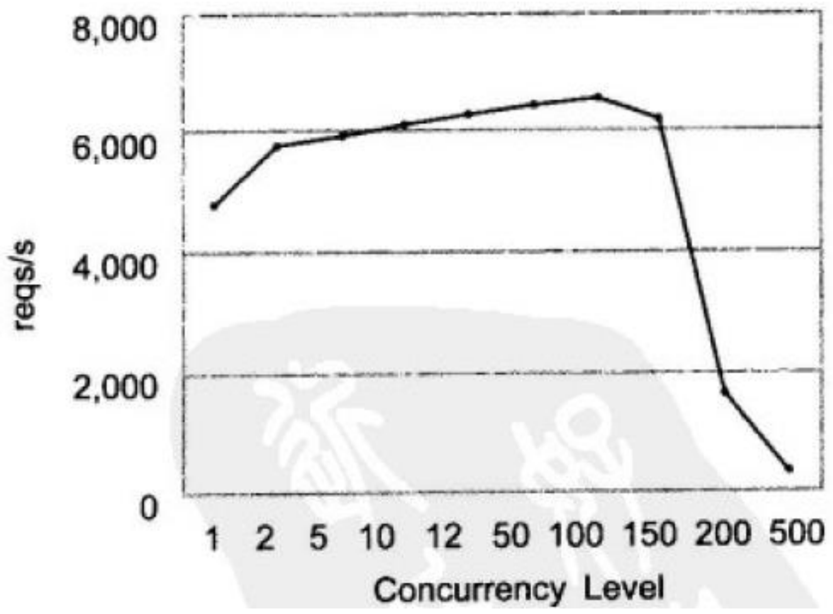
高并发网站架构设计案例 – 阿里巴巴中国站

- 中国站网站的正常流量情况
  - 并发（单台），高峰期 < 10
  - 吞吐量（TPS，单台） 高峰期 <60
  - CPU负载 Load 高峰期，<2, 大部分服务器<1
  - CPU使用率 ， 一般只占1颗核，平均60%左右
  - 服务器平均响应时间 高峰期 <150ms

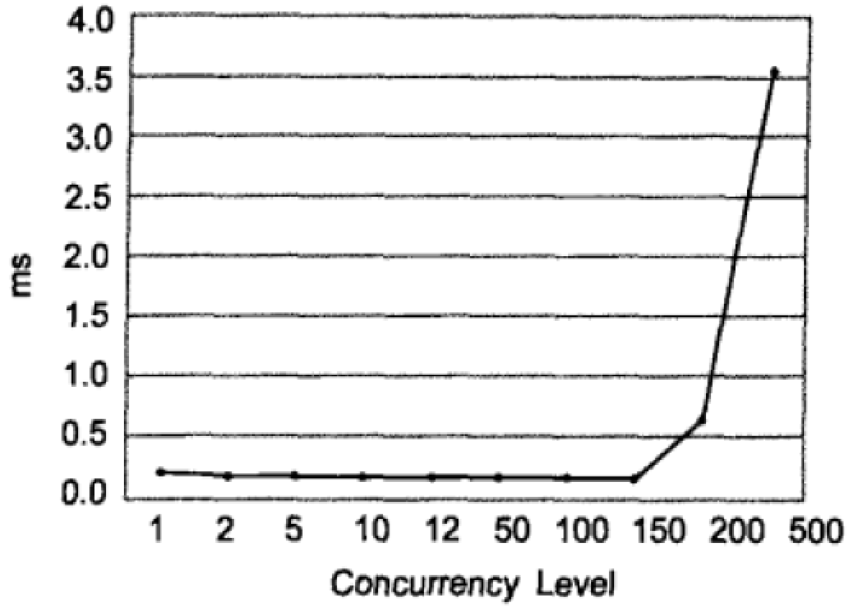
- 高并发下的风险
  - 网络带宽耗尽
  - 服务器Load飙高，停止响应
  - 数据库瘫痪

- 高并发下的事故
  - 事故：网站运营旺旺推广页面弹出，1兆大图片导致带宽耗尽
    增加审核机制：运营推广增加的图片流量不能超过现有流量的30%
  - 合作媒体推广：迅雷，暴风影音浮出广告，导致旺铺集群Crash

高并发度对网站性能的影响



并发数对吞吐量的影响



并发数对服务器平均响应时间的影响

高并发高负载实例 – 1688.com秒杀活动

- 商业需求
  - ➢ 为庆祝1688.com开业退出88小时不间断秒杀活动
  - ➢ 每小时整点推出8款商品，拖拉机，牛，马桶，沙发……
  - ➢ 每款商品供168件，每人限批3件，成交人数56人
  - ➢ CCTV黄金广告时间，各种网络,平面媒体轰炸，总广告费：1.5亿
- 技术挑战
  - – 瞬间高并发
    - ➢ 8000并发：预估秒杀在线人数可达8000人
    - ➢ 风险：带宽耗尽
    - ➢ 服务器：崩溃，可以理解成自己给自己准备的D.D.O.S攻击
  - – 秒杀器
    - ➢ 第一种：秒杀前不断刷新秒杀页面，直到秒杀开始，抢着下单
    - ➢ 第二种：跳过秒杀页面，直接进入下单页面，下单

网站并发架构

- 服务器集群
  - style服务器（Lighttpd集群）：5台
  - 图片服务器（Nginx集群）：5台
  - 静态服务器（Apache集群）：10台
  - 交易服务器（JBoss动态集群）：10台
- 网络带宽
  - 图片出口带宽上限：2.5G （出口带宽支持10G，但图片服务器集群的处理能力：图片服务集群最大并发处理能力 X 网站平均图片大小 = 2.5G）
  - CDN准备：Chinacache沟通；借用Taobao CDN

1688.com秒杀系统：架构目标

1.图片网络带宽：1.0G

新增图片带宽：必须控制在 1.0G 左右

每件商品秒杀页面的图片总大小不得超过：1000000/(1000*8) = 125K/每商品

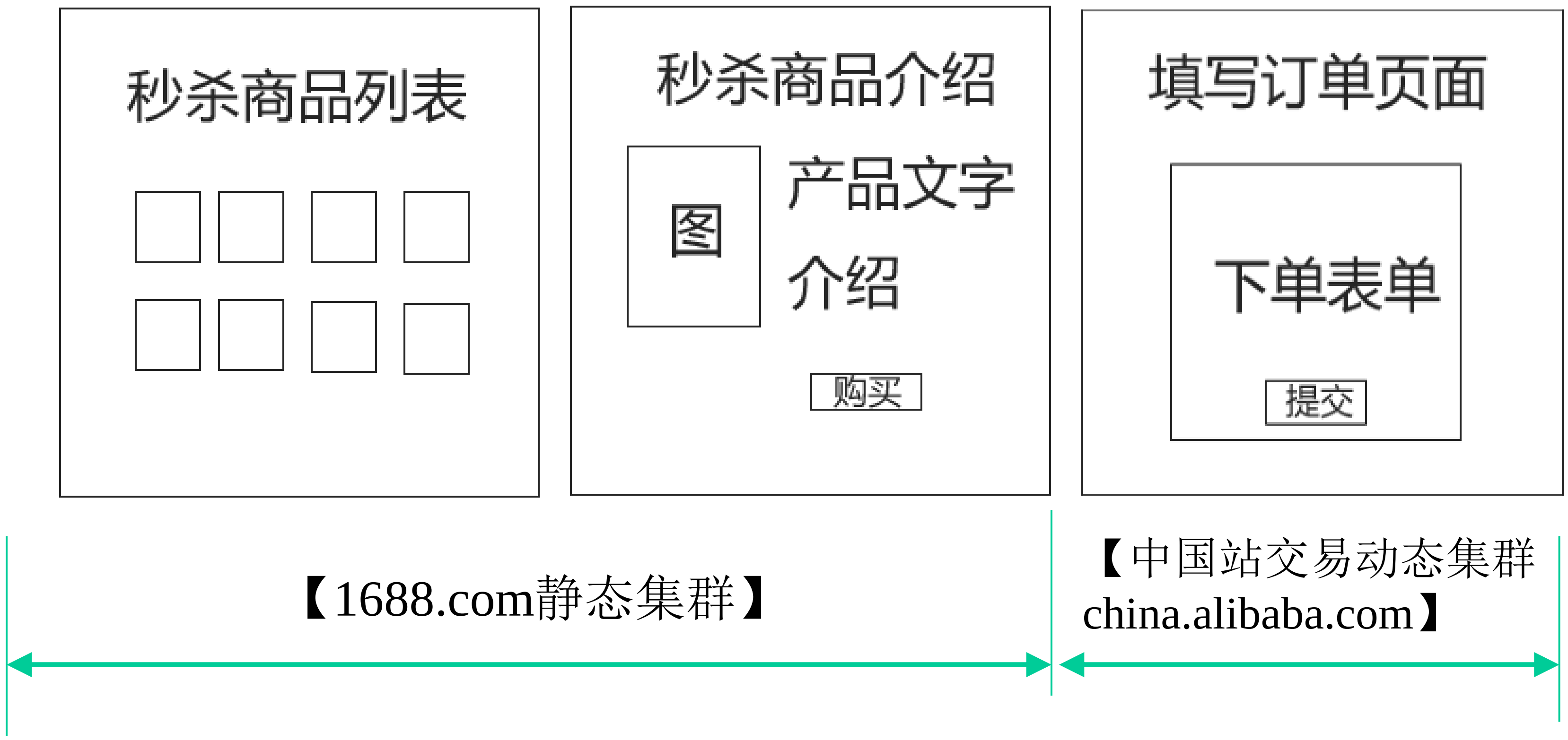
2.网站并发：

单件商品并发：1000 【来自运营的预估】

总并发: 8（件商品）X 1000（人/商品）=8000

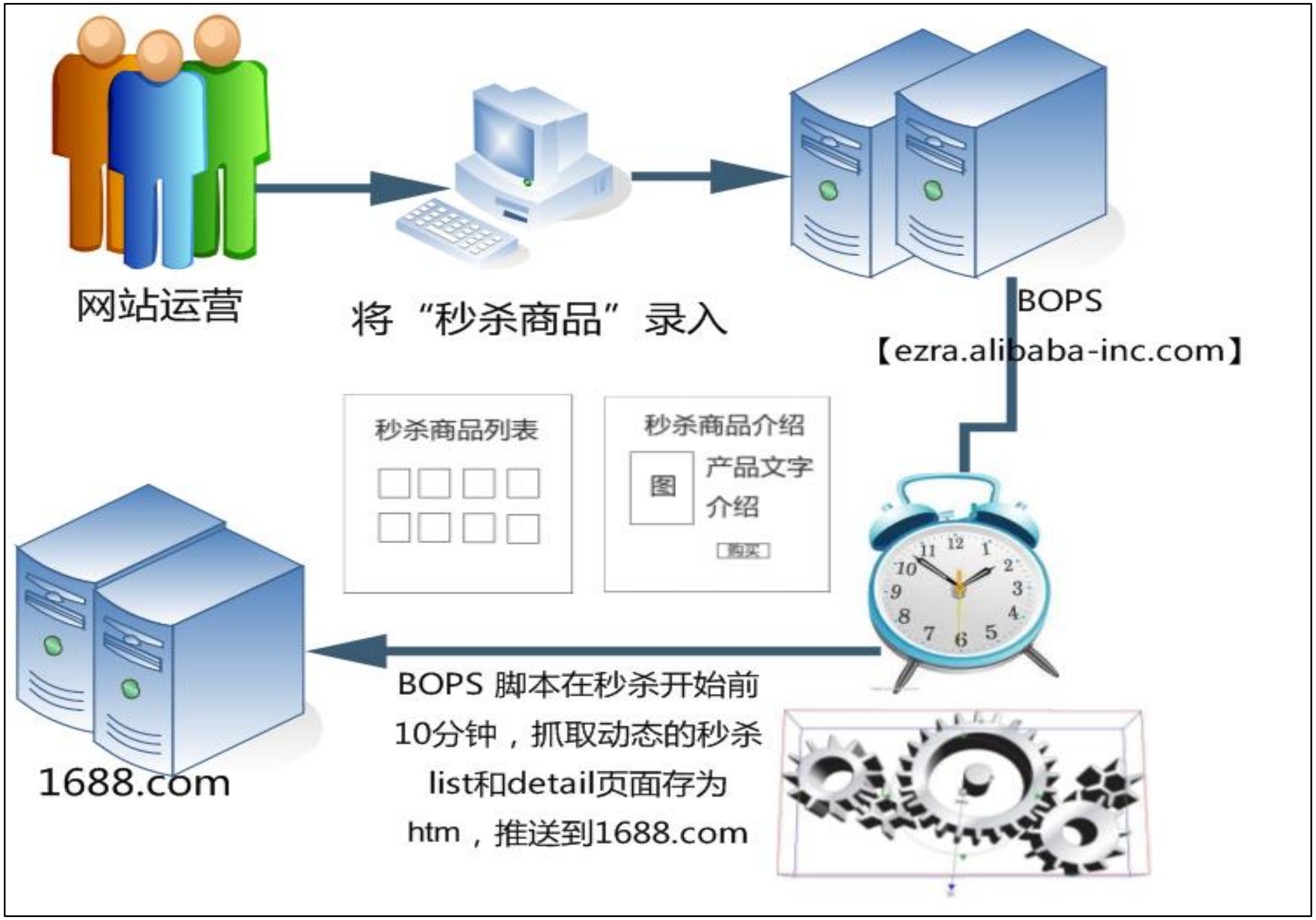
秒杀系统组成

三个页面组成：**秒杀商品列表，秒杀商品介绍，下单**



1688.com秒杀系统：设计原则

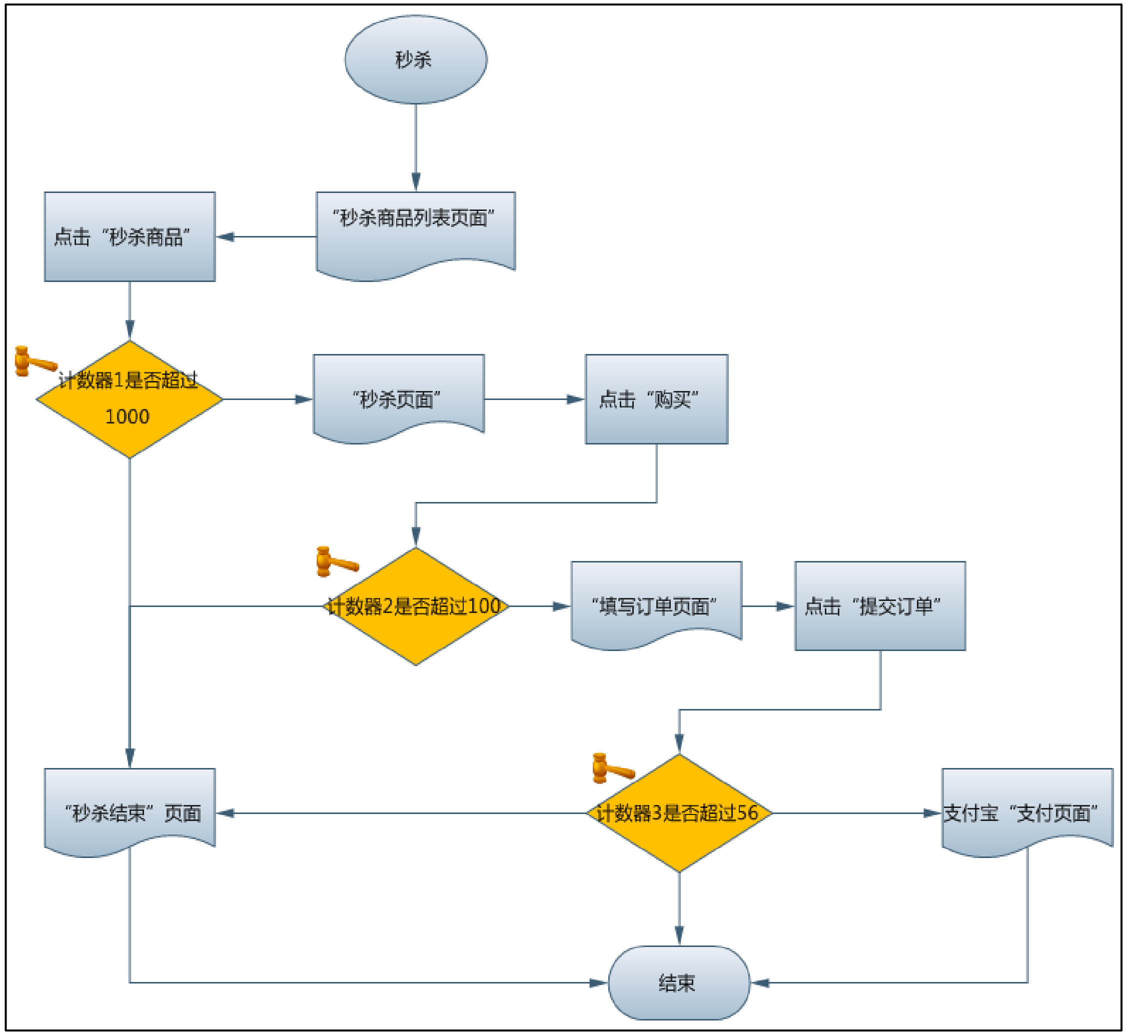
- 静态化
  - 采用JS自动更新技术将动态页面转化为静态页面
- 并发控制，防秒杀器
  - 设置阀门，只放最前面的一部分人进入秒杀系统
- 简化流程
  - 砍掉不重要的分支流程，如下单页面的所有数据库查询
  - 以下单成功作为秒杀成功标志。支付流程只要在1天内完成即可。
- 前端优化
  - 采用YSLOW原则提升页面响应速度


网站运营
将“秒杀商品”录入
BOPS
【ezra.alibaba-inc.com】
秒杀商品列表
秒杀商品介绍
图
产品文字
介绍
购买
1688.com
BOPS 脚本在秒杀开始前
10分钟，抓取动态的秒杀
list和detail页面存为
htm，推送到1688.com

三道阀门的设计

阀门：基于TT的计数器

序号	阀门上限
1	限制进入秒杀页面，1000
2	限制进入下单页面，100
3	限制进入支付宝系统，56

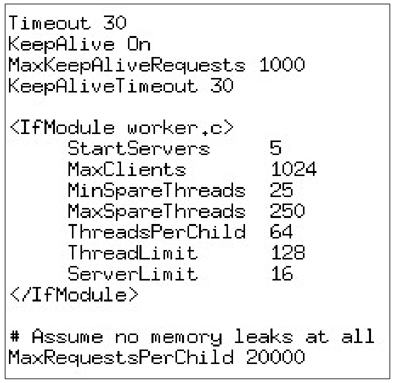


秒杀器的预防

- 秒杀Detail页面
  - URL：随机
  - 秒杀前2秒放出,脚本生成，秒杀前
  - 1000次访问上限控制【每件商品只能放入1000人浏览】

- 下单页面：
  - 订单ID，随机
  - 不能直接跳过秒杀Detail页面进入
  - 每个秒杀商品，带预先生成的随机Token作URL 参数
  - 如果秒杀过，直接跳到秒杀结束页面
  - 100次访问上限控制【每件商品只能放入1000人下单】

Web Server 调优 – Apache调优



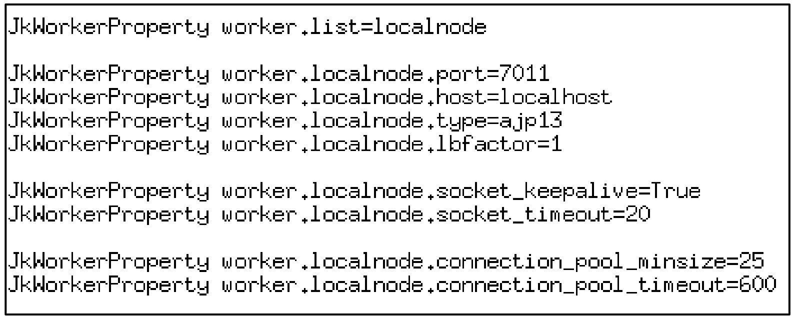
```
Timeout 30
KeepAlive On
MaxKeepAliveRequests 1000
KeepAliveTimeout 30

<IfModule worker.c>
     StartServers       5
     MaxClients         1024
     MinSpareThreads    25
     MaxSpareThreads    250
     ThreadsPerChild    64
     ThreadLimit        128
     ServerLimit        16
</IfModule>

# Assume no memory leaks at all
MaxRequestsPerChild 20000
```

- KeepAlive相关参数调优
- 其他参数调优
  - HostnameLookups 设为off，对allowfromdomain等后的域名不进行正向和反向的dns解析
- 关闭cookies-log日志
- 打开Linux sendfile()
- 关闭无用的module
  - mod_Gzip

  (秒杀页面，非图片html文本所占流量比重可忽略不计，zip意义不大),
  - mod_Beacon,
  - mod_hummock（等待反应过来，秒杀已经over了）

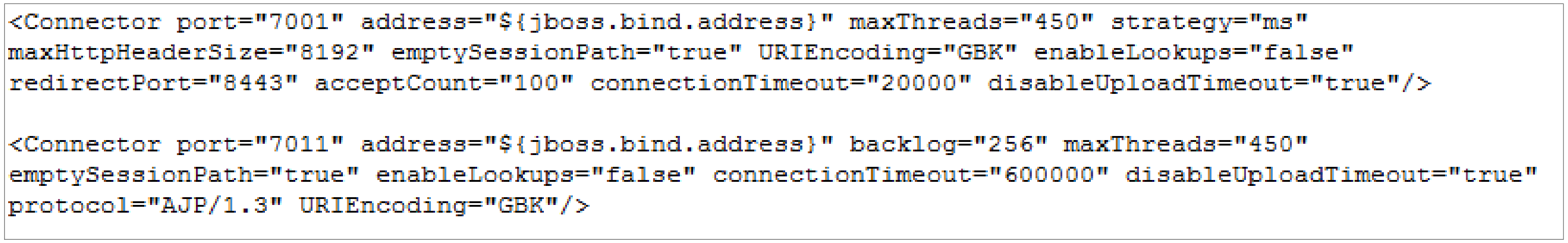
Web Server 调优 – JBoss调优

- Mod-jk worker 调优



```
JkWorkerProperty worker.list=localnode

JkWorkerProperty worker.localnode.port=7011
JkWorkerProperty worker.localnode.host=localhost
JkWorkerProperty worker.localnode.type=ajp13
JkWorkerProperty worker.localnode.lbfactor=1

JkWorkerProperty worker.localnode.socket_keepalive=True
JkWorkerProperty worker.localnode.socket_timeout=20

JkWorkerProperty worker.localnode.connection_pool_minsize=25
JkWorkerProperty worker.localnode.connection_pool_timeout=600
```

- JBoss AJP Connector



```
<Connector port="7001" address="${jboss.bind.address}" maxThreads="450" strategy="ms"
maxHttpHeaderSize="8192" emptySessionPath="true" URIEncoding="GBK" enableLookups="false"
redirectPort="8443" acceptCount="100" connectionTimeout="20000" disableUploadTimeout="true"/>

<Connector port="7011" address="${jboss.bind.address}" backlog="256" maxThreads="450"
emptySessionPath="true" enableLookups="false" connectionTimeout="600000" disableUploadTimeout="true"
protocol="AJP/1.3" URIEncoding="GBK"/>
```

- Tomcat APR 设定

秒杀静态页面优化

- 图片合并
  - 8张图片合并成1张，css偏移展示
  - 减少HTTP请求数，减少请求等待数
  - 减少发送cookies的量
- HTML内容压缩
- 图片压缩：图片Bytes < 长X宽/2250
- HTML Header Cache-Control设置
- CSS，JS 精简
  - CSS,JS 精简到极致，部分直接写在页面中，减少Http请求次数

下单页面优化

- 数据库操作：全部砍掉
  - 原下单页面要访问8次数据库，全部砍掉
- 秒杀流程精简
  - 砍掉填写或选择收货地址，放在秒杀成功后填写
  - 砍掉调用是否开通支付宝接口，秒杀首页文案提示必须开通
- 采用内存缓存
  - 秒杀Offer数据，支付宝相关信息，缓存

交易系统性能优化

- 交易系统调优目标:

	并发	TPS
下单页面（优化前）	20	100
下单页面（优化后）	40	400

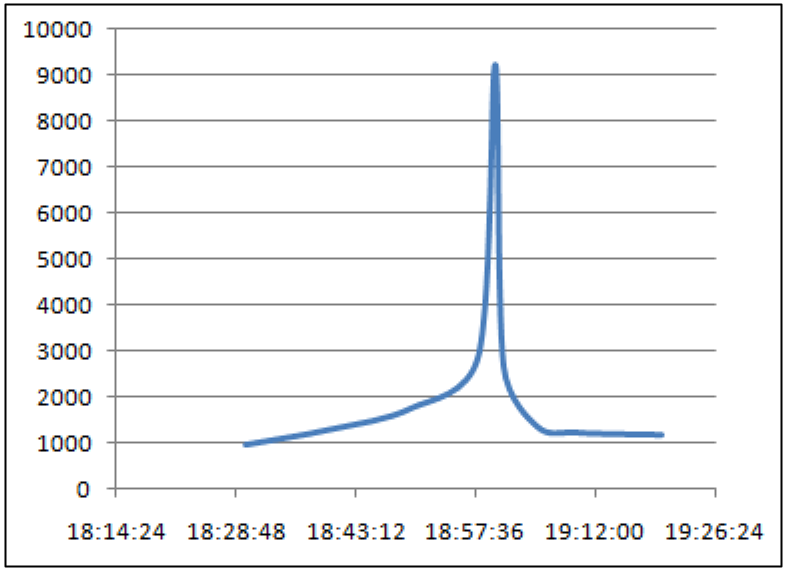
- 关闭 Keep Alive（分析交易系统accesslog，用户在短时间内连续点击概率很低）
- JVM优化
  优化CMS 垃圾回收器的参数
- 消灭 Top10 Bottlenecks
  – Velocity参数调优
  – 采用DBCP1.4 替换C3P0
  – Offer 产品参数的XML解析

应急预案

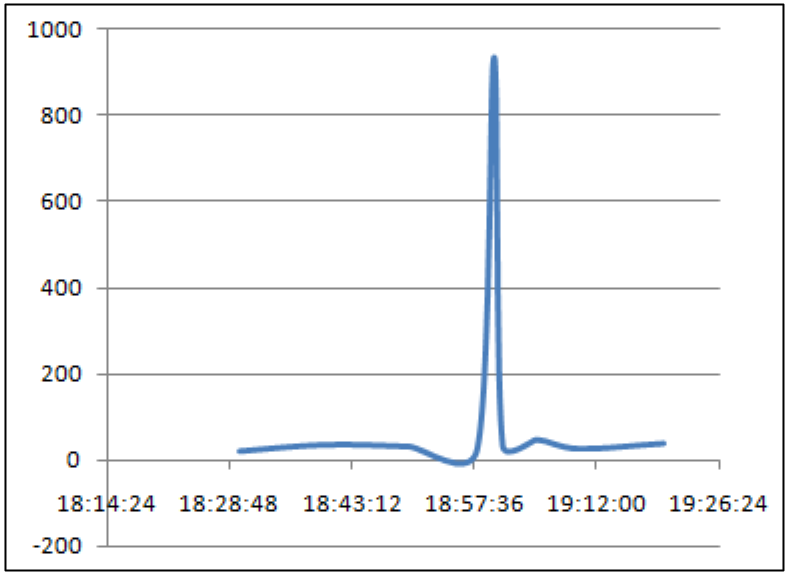
- 域名分离，独立域名，不影响中国站原有业务
  - Style集群： style.1688.china.alibaba.com
  - 图片服务器集群：img.1688.china.alibaba.com
  - 静态页面集群：page.1688.china.alibaba.com
- 机动服务器10台，备用
- 拆东墙补西墙战略
  - ✱ 5天时间来不及采购服务器，因此SA待命，随时准备将非核心应用集群的冗余服务器下线，加入到秒杀集群
- 壁虎断尾策略
  - 所有办法均失效的情况下，例如流量耗尽
    - 非核心应用集群统统停止服务，如资讯，论坛，博客等社区系统
    - 保住首页，Offer Detail,旺铺页面等核心应用的可用性
- 万能出错页面：秒杀活动已经结束
  - 任何出错都302跳转到此页面
  - 位于另外集群

秒杀活动结果

- 88小时秒杀，坚守阵地，大获成功
- 秒杀还是被秒杀？终于有了答案
- 三道阀门设计非常有效，拦住了秒杀器



1688.com 静态集群总并发情况
（首页，秒杀列表，秒杀商品页面）



交易系统集群总并发情况
（下单页面）

Questions ?   Comments?