哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

软件工程

第六章　软件架构设计

6-6 软件架构概论

徐汉川
xhc@hit.edu.cn

2017年11月8日

主要内容

1. 什么是软件架构
2. 软件架构中的核心概念
3. 软件架构的四大思想

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

1. 什么是软件架构

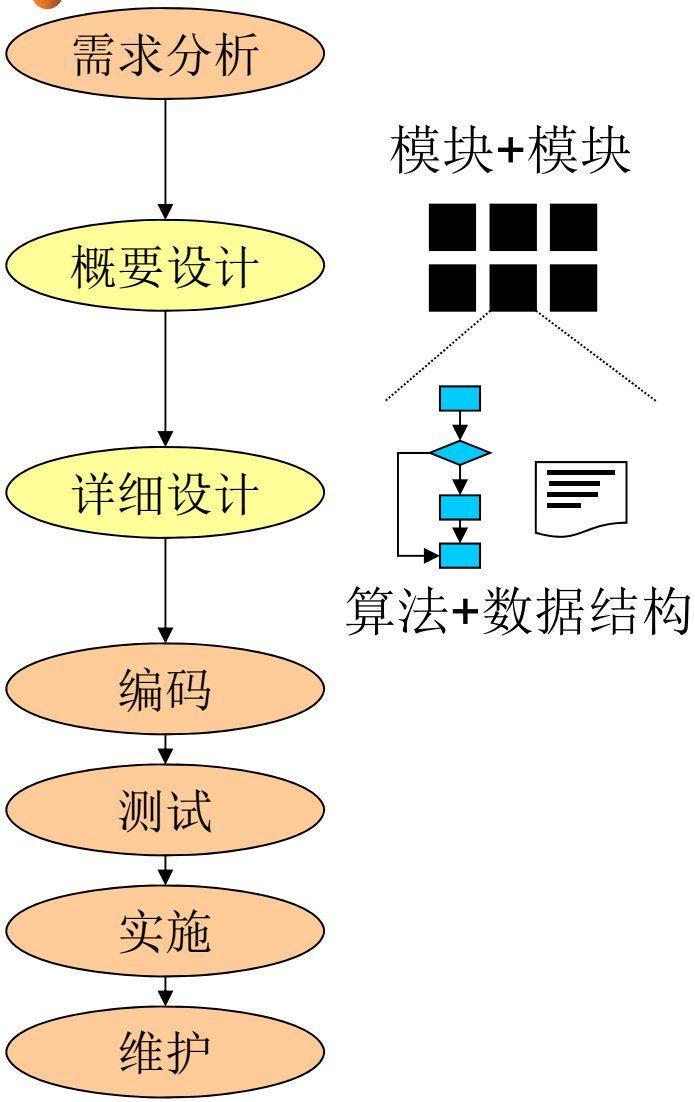
为什么需要软件架构？

- **软件越来越复杂，组成部分越来越多**
  - 多个源文件
  - 多种类型的文件：用户界面、算法、数据层程序、配置文件、etc
- **不是单纯的代码，还涉及到所依赖的硬件和网络环境**
  - 物理位置：单机、服务器、手机端、可穿戴硬件、etc
  - 网络支持：有线网络、3G/4G、WiFi等

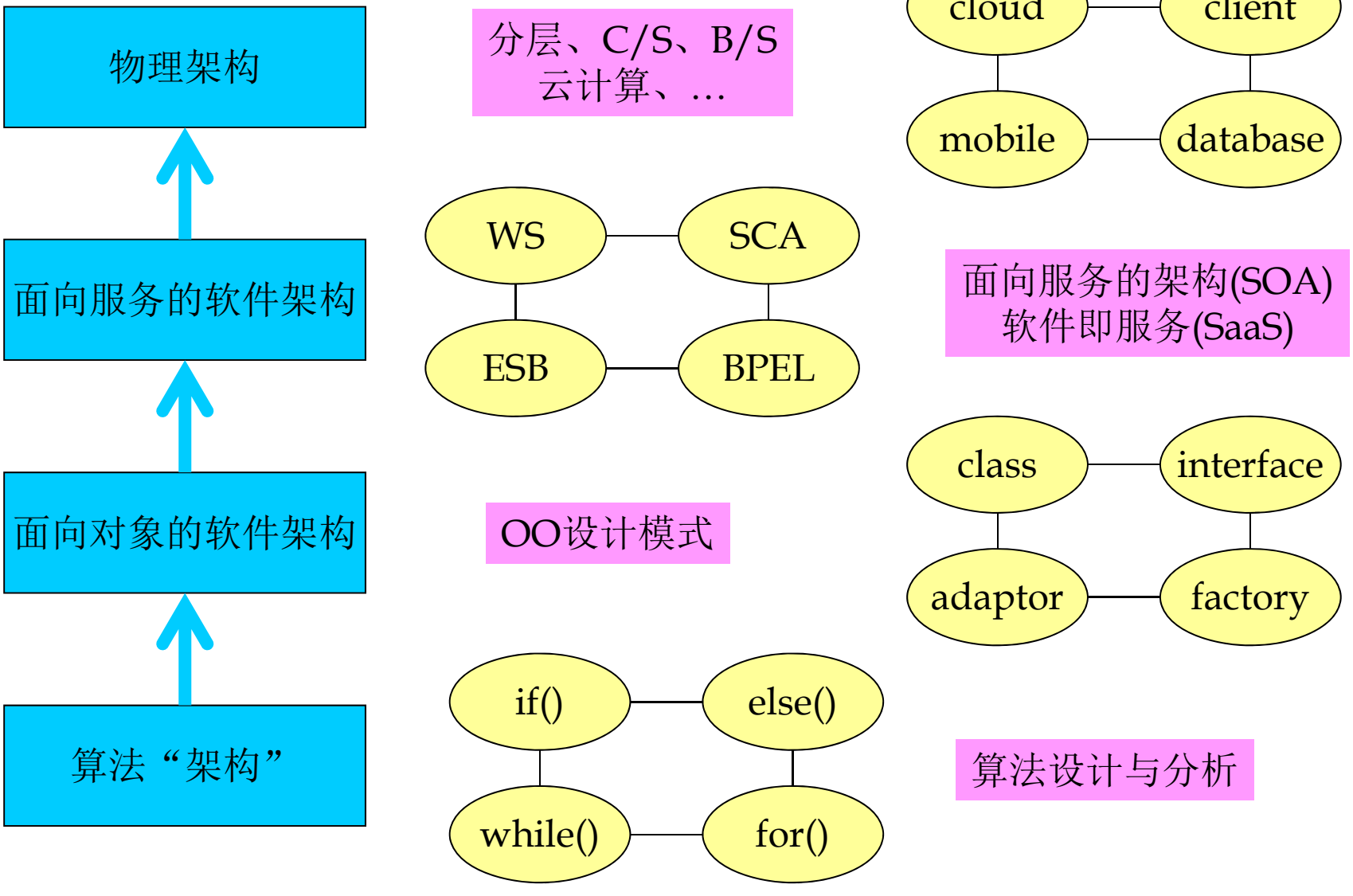
- **多个软件实体之间如何组织起来？**
- **软件和硬件之间的关系如何？**

- **此即“软件架构”(Software Architecture)所关注的内容。**

软件设计的过程



你早已接触过“架构”…


物理架构
面向服务的软件架构
面向对象的软件架构
算法“架构”
分层、C/S、B/S
云计算、…
cloud
client
mobile
database
WS
SCA
ESB
BPEL
面向服务的架构(SOA)
软件即服务(SaaS)
OO设计模式
class
interface
adaptor
factory
if()
else()
while()
for()
算法设计与分析

你早已接触过“架构”…

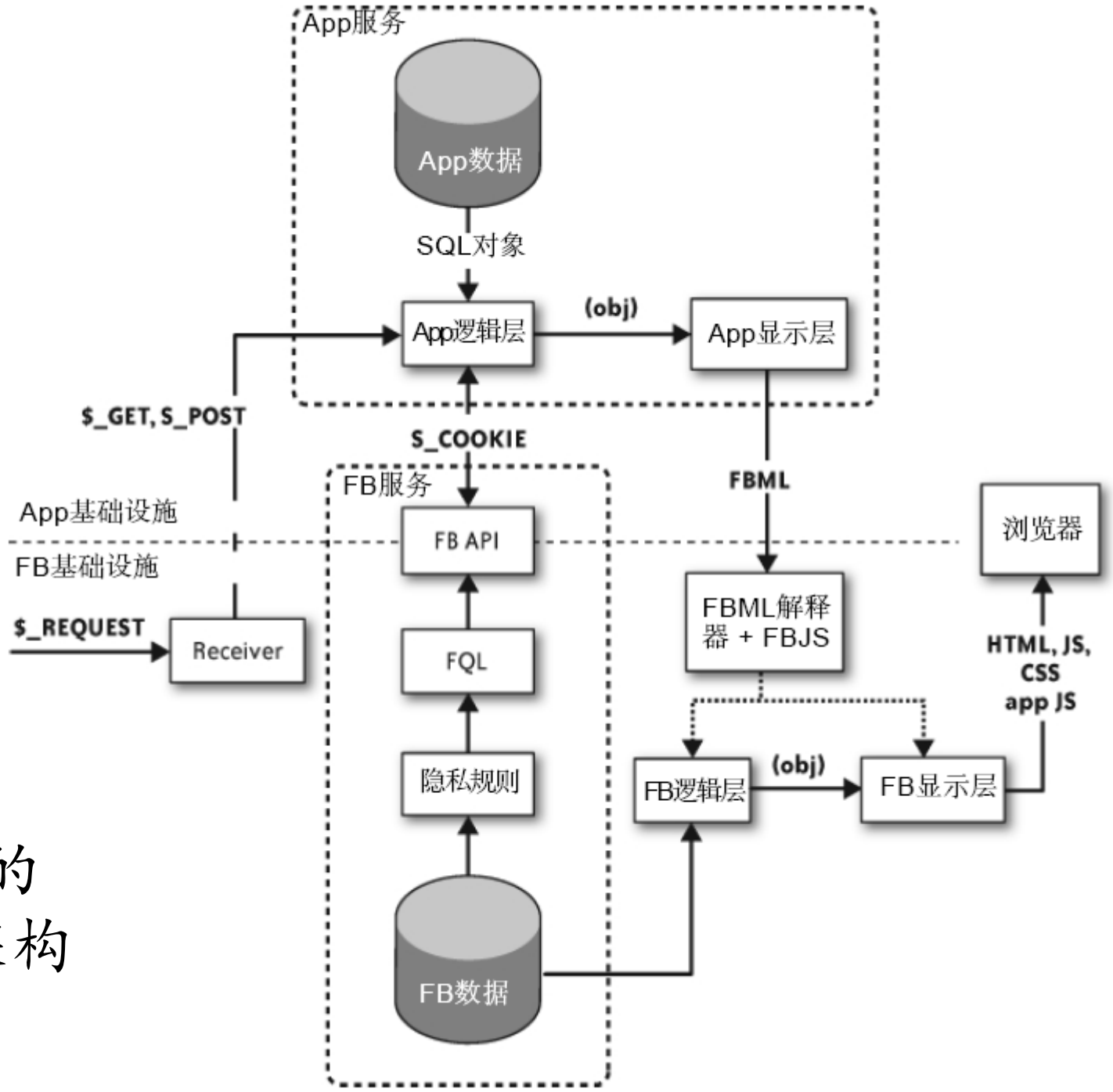
- 超市的多个收银台，顾客排长队 === 服务器并发处理的性能和容量
- 十字路口的车辆等待转弯 === 通过缓存来提高交通吞吐率

- 分层、构件化、服务化、标准化、...
- 缓存、分离/松散耦合、队列、复制、冗余、代理、...
- C/S、B/S、负载平衡、...

- 如何用它们解决具体软件问题，在博弈中寻求平衡(折中)，就是软件架构所关注的问题。

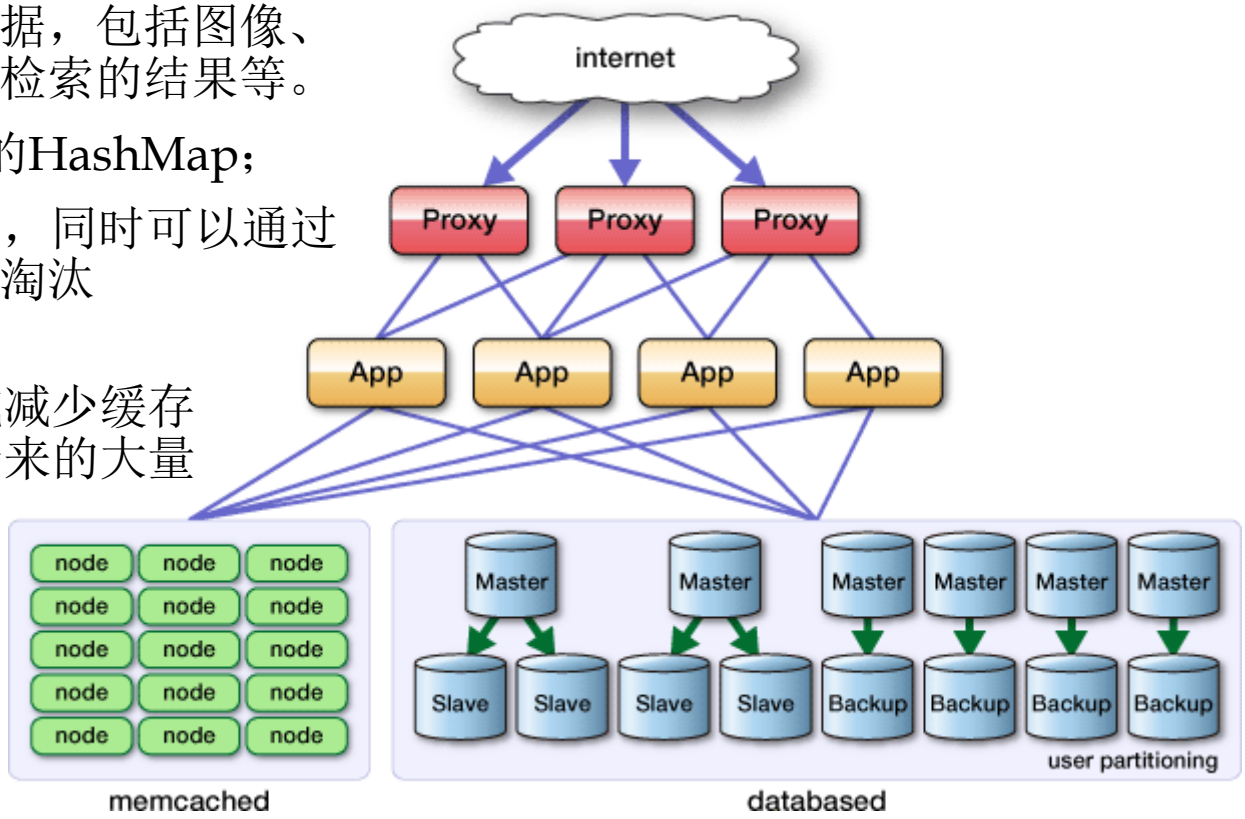
- 新的架构层出不穷，但所遵循的基本原理都是相通的。

Facebook的开放API架构

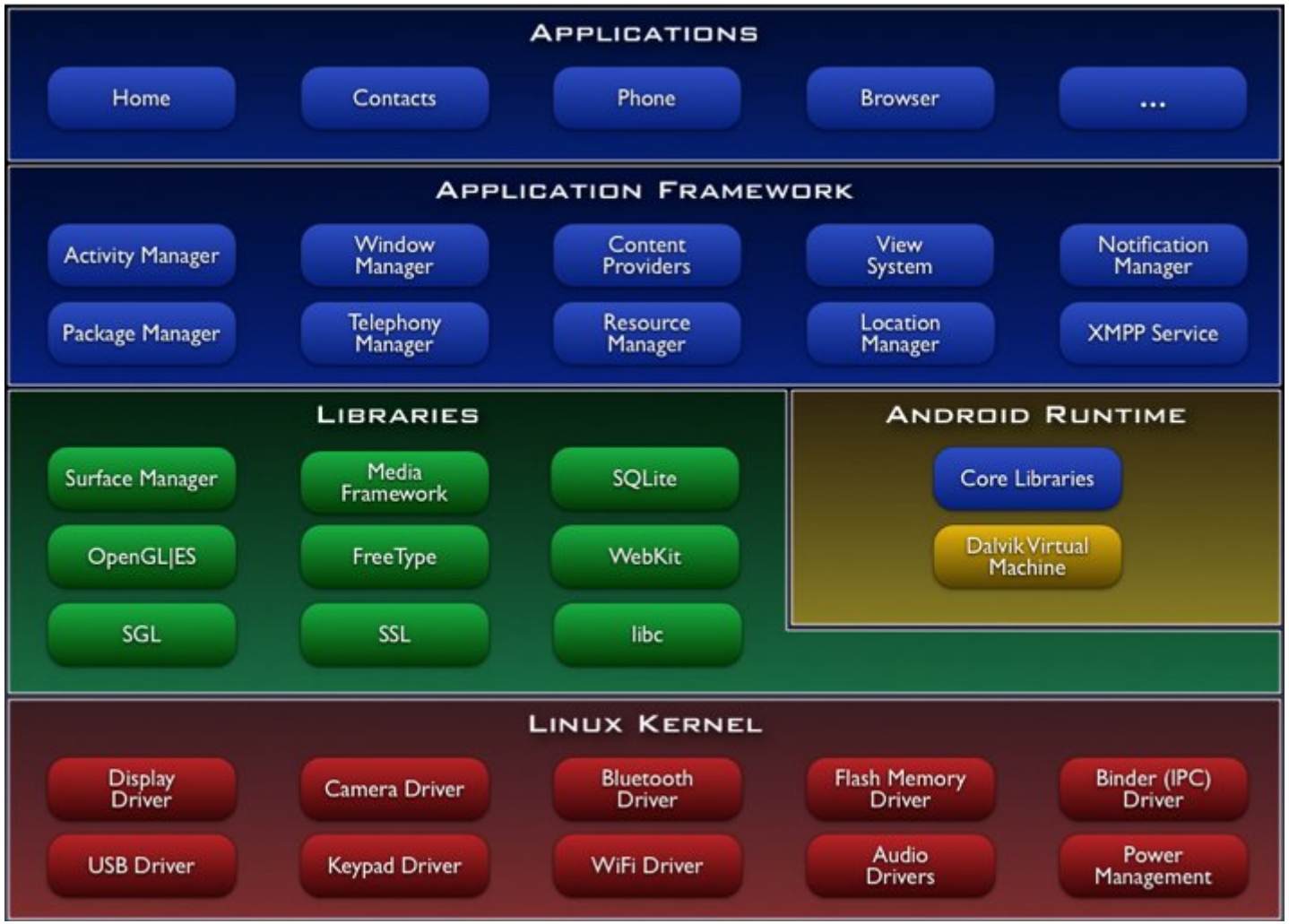


MySQL MemCached的缓存和分布式存储架构

- **MemCached：**
  - 一个高性能的**分布式**内存对象缓存系统，用于动态Web应用以减轻DB负载；
  - 它通过在内存中缓存数据和对象来减少读取DB的次数，从而提供动态、DB驱动网站的速度；
  - 用来存储各种格式的数据，包括图像、视频、文件以及数据库检索的结果等。
  - 基于一个存储键/值对的HashMap；
  - 通过LRU算法进行淘汰，同时可以通过删除和设置失效时间来淘汰存放在内存的数据
  - 一致性hash解决增加或减少缓存服务器导致重新hash带来的大量缓存失效的弊端。
- **Ruby on Rails' Cache Money (twitter)**



Google Android的分层结构



如何学习软件架构

- ——从最基本的概念入手，学习传统的软件架构设计思想
  - 数据流、数据仓库、结构化、OO、分层、事件、规则、…
- ——逐步过渡到各类新涌现的架构思想
  - P2P、web 2.0、SOA、EDA、虚拟化、SaaS、云计算、…
- ——通过模仿已有架构和案例来设计自己的系统
  - MVC、Struts、Hibernate、Spring、…
  - Facebook、Twitter、Android、…
- ——了解不同应用领域的架构套路
  - 企业级系统、移动终端软件、在线游戏、…

什么是“体系结构”

- 词典的定义：
  - The art and science of designing and erecting buildings (建筑学：设计和建造建筑物的艺术与科学);
  - A style and method of design and construction (设计及构造的方式和方法);
  - Orderly arrangement of parts; structure (部件的有序安排；结构);
  - The overall design or structure of a computer system, including the hardwareand the software required to run it, especially the internal structure of the microprocessor (计算机系统的总体设计或结构，包括其硬件和支持硬件运行的软件，尤其是微处理器内部的结构)。

起源于建筑学的“体系结构”

- **“体系结构(Architecture)”一词起源于建筑学，也翻译为“架构”**
  - 如何使用基本的建筑模块构造一座完整的建筑？
- **包含两个因素：**
  - 基本的建筑模块：砖、瓦、灰、沙、石、预制梁、柱、屋面板…
  - 建筑模块之间的粘接关系：如何把这些“砖、瓦、灰、沙、石、预制梁、柱、屋面板”有机的组合起来形成整体建筑？

起源于建筑学的“体系结构”

简单房屋

复杂需求的房屋

缺乏良好设计的房屋

计算机硬件系统的“体系结构”

- 如何将设备组装起来形成完整的计算机硬件系统？
- 包含两个因素：
  - 基本的硬件模块：控制器、运算器、内存储器、外存储器、输入设备、输出设备…
  - 硬件模块之间的连接关系：总线
- 计算机架构的风格：
  - 以存储程序原理为基础的冯·诺依曼结构
  - 存储系统的层次结构
  - 并行处理机结构
  - ……

“体系结构”的共性

- 共性：
  - 一组基本的构成要素——构件
  - 这些要素之间的连接关系——连接件
  - 这些要素连接之后形成的拓扑结构——物理分布
  - 作用于这些要素或连接关系上的限制条件——约束
  - 质量——性能

归纳：SA的定义

- **软件体系结构(SA)：**
  - 是一个关于系统形式和结构的综合框架，包括系统构件和构件的整合
  - 从一个较高的层次来考虑组成系统的构件、构件之间的连接，以及由构件与构件交互形成的拓扑结构
  - 这些要素应该满足一定的限制，遵循一定的设计规则，能够在一定的环境下进行演化
  - 反映系统开发中具有重要影响的设计决策，便于各种人员的交流，反映多种关注，据此开发的系统能完成系统既定的功能和性能需求

体系结构 = 构件 + 连接件 + 拓扑结构 + 约束 + 质量

Architecture = Components + Connectors + Topology + Constraints + Performance

为什么要重视“软件架构”

- **随着软件系统规模越来越大、越来越复杂**
  - 用户需求越来越复杂，变化越来越频繁；
  - 对软件质量(功能性/非功能性)的要求越来越高；
  - 如何将成百上千个功能组合起来，满足用户质量需求，变得越来越困难。
- **此时，整个系统的结构和规格说明显得越来越重要。**
  - 很多质量需求主要体现在体系结构中而非功能模块内部的实现中。

- 结论：对于大规模的复杂软件系统来说，对总体的系统结构设计和规格说明比起对计算的算法和数据结构的选择已经变得明显重要得多。

软件架构要回答的基本问题

- 从建筑架构看起
  - 基本的建筑单元都有哪些？
  - 有哪些实用、美观、强度、造价合理、可复用的大粒度建筑单元，使建造出来的建筑更能满足用户的需求？
  - 建筑模块怎样搭配才合理？
  - 有哪些典型的建筑风格？
  - 每种典型建筑(医院、工厂、旅馆)的典型结构是什么样子？需要什么样的构件？
  - 如何绘制建筑架构的图纸？如何根据图纸进行质量评估？
  - 如何快速节省的将图纸变为实物(即施工过程)？
  - 建筑完成之后，如何对其进行恰当程度的修改？重要模块有了更改后，如何保证整栋建筑质量不受影响？

软件架构要回答的基本问题

- 软件的基本构造单元是什么？
- 这些构造单元之间如何连接？
- 最终形成何种样式的拓扑结构？
- 每个典型应用领域的典型体系结构是什么样子？
- 如何进行软件体系结构的设计与实现？
- 如何对已经存在的软件体系结构进行修改？
- 使用何种工具来支持软件体系结构的设计？
- 如果对软件的体系结构进行描述，并据此进行分析和验证？

软件架构的目标与作用

- 软件体系结构关注的是：
  - 如何将复杂的软件系统划分为模块、如何规范模块的构成和性能、以及如何将这些模块组织为完整的系统。
- 主要目标：
  - 建立一个一致的系统及其视图集，并表达为最终用户和软件设计者需要的结构形式，支持用户和设计者之间的交流与理解。
- 作用：
  - 交流的手段：在软件设计者、最终用户之间方便的交流；
  - 可传递的、可复用的模型：对一些经过实现证明的体系结构进行复用，从而提高设计的效率和可靠性，降低设计的复杂度。
  - 早期决策的体现：全面表达和深刻理解系统的高层次关系，使设计者在复杂的、矛盾的需求面前作出正确的选择；正确的体系结构是系统成功的关键，错误选择会造成灾难性后果；

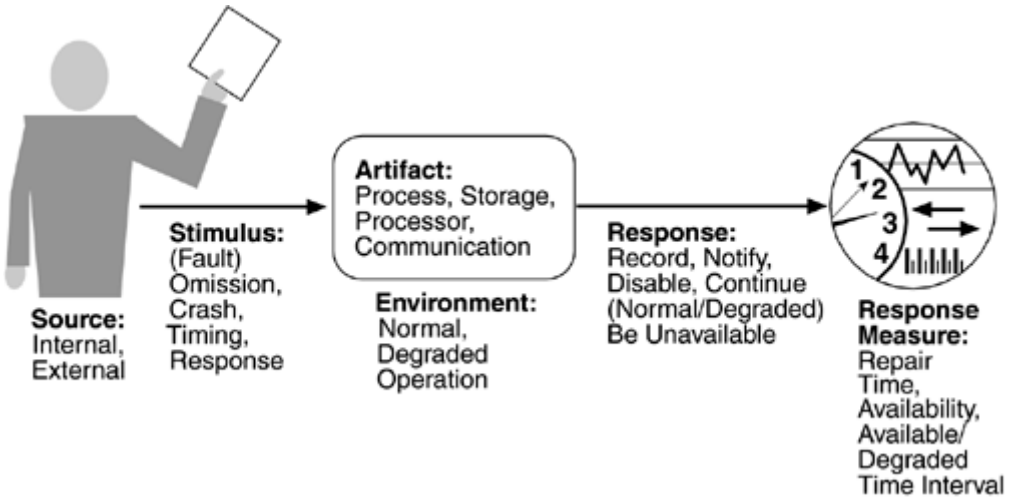
外向目标 VS 内向目标

- 外向目标
  - 关心的是：为满足最终用户的需求，系统应该做些什么。
  - 要在系统实现之前寻求需求决策并确保系统的性能。
  - 为此，需要分析当前需求、扩展或细化结构、澄清模糊性、提高一致性

- 内向目标
  - 关心的是：如何使系统满足用户需求、需要建立哪些软件模块、软件模块的结构、模块之间的关系
  - 通过对主要构件及其关系的规划，为以后的系统设计和实施活动提供了基础和依据。
  - 为了达到这个目标，需要考虑各种可选方案，重复更新和明确设计目标，必要时作出妥协(compromise)和折中(tradeoff)。

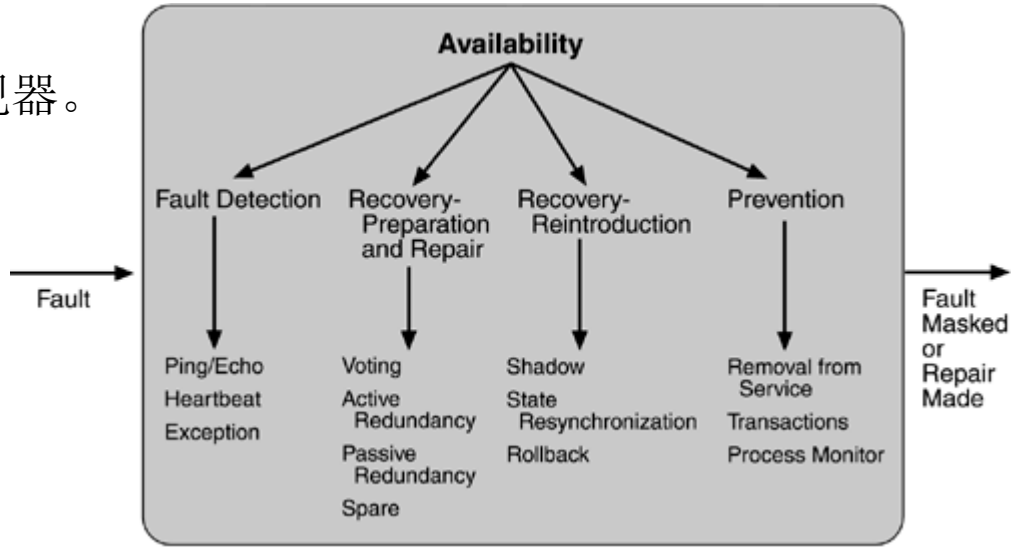
典型质量属性的架构设计：可用性(availability)

- 当系统不再提供其规格说明中所描述的服务时，就出现了系统故障，即表示系统的可用性变差。
- 关注的方面：
  - 如何检测系统故障、故障发生的频度、出现故障时的表现、允许系统有多长时间非正常运行、如何防止故障发生、发生故障后如何消除故障、等等。



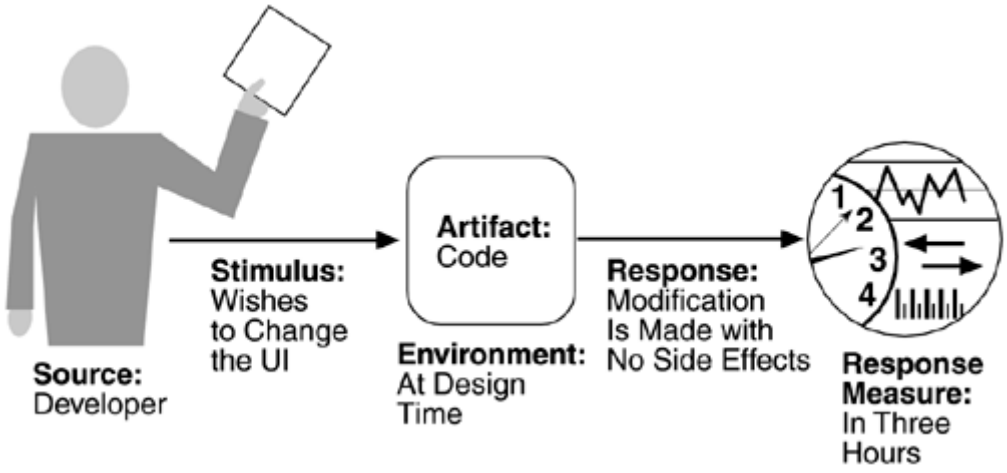
典型质量属性的架构设计：可用性(availability)

- **错误检测(Fault Detection)**
  - 命令-响应机制(ping-echo)、心跳(heartbeat)机制、异常机制(exception)；
- **错误恢复(Recovery)**
  - 表决、主动冗余(热重启)、被动冗余(暖重启/双冗余/三冗余)、备件(spare)；
  - Shadow操作、状态再同步、检查点/回滚；
- **错误预防(Prevention)**
  - 从服务中删除、事务、进程监视器。



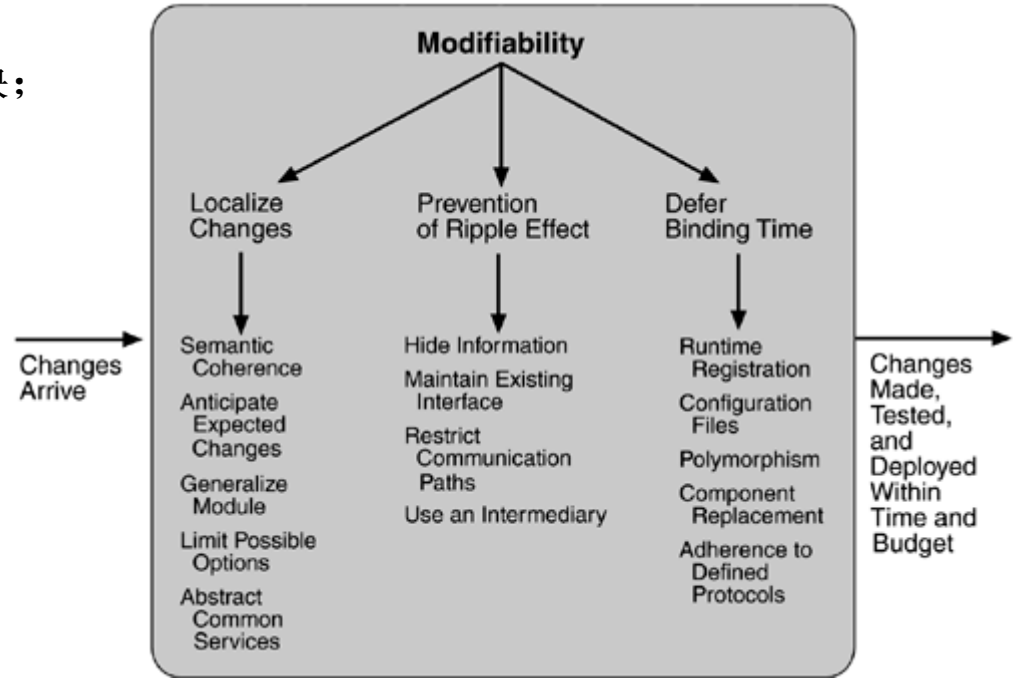
典型质量属性的架构设计：可修改性(modifiability)

- 可以修改什么——功能、平台(HW/OS/MW)、外部环境、质量属性、容量、等；
- 何时修改——编译期间、构建期间、配置期间、执行期间；
- 谁来修改——开发人员、最终用户、实施人员、管理人员；
- 修改的代价有多大？
- 修改的效率有多高？



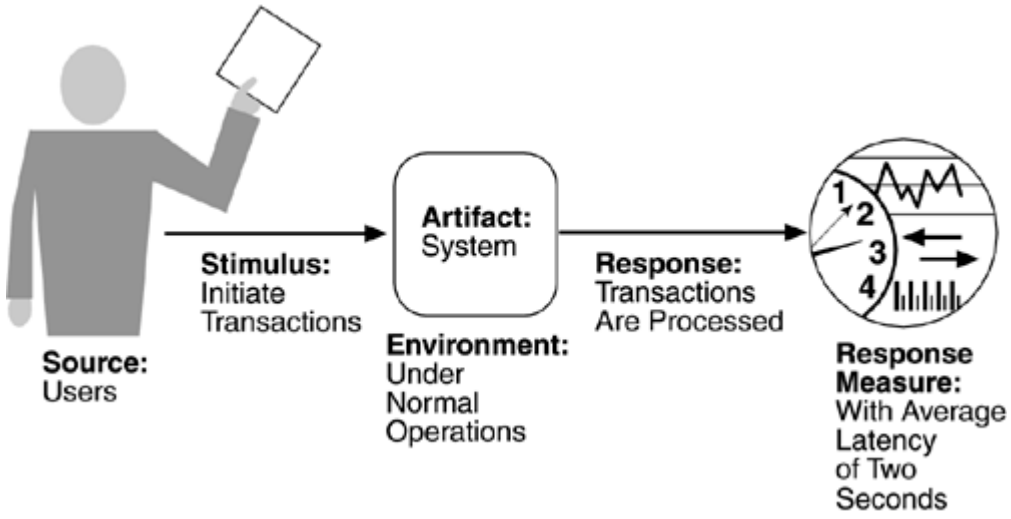
典型质量属性的架构设计：可修改性(modifiability)

- 目标：减少由某个修改所直接/间接影响的模块的数量；
- 常用决策：
  - 高内聚/低耦合、固定部分与可变部分分离、抽象为通用模块、变“编译”为“解释”；
  - 信息隐藏、保持接口抽象化和稳定化、适配器、低扇出；
  - 推迟绑定时间——运行时注册、配置文件、多态、运行时动态替换；



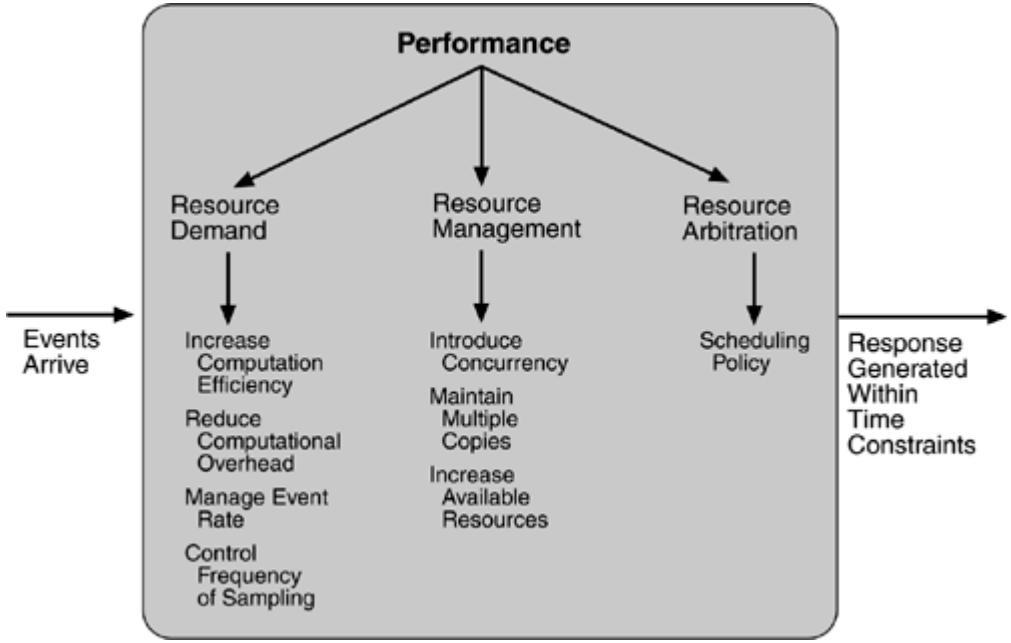
典型质量属性的架构设计：性能(performance)

- 与时间有关：当外部请求到达时，系统将要耗费多少时间做出相应；
- 因素：请求的数量和到达模式、请求的频度、耗费的资源；



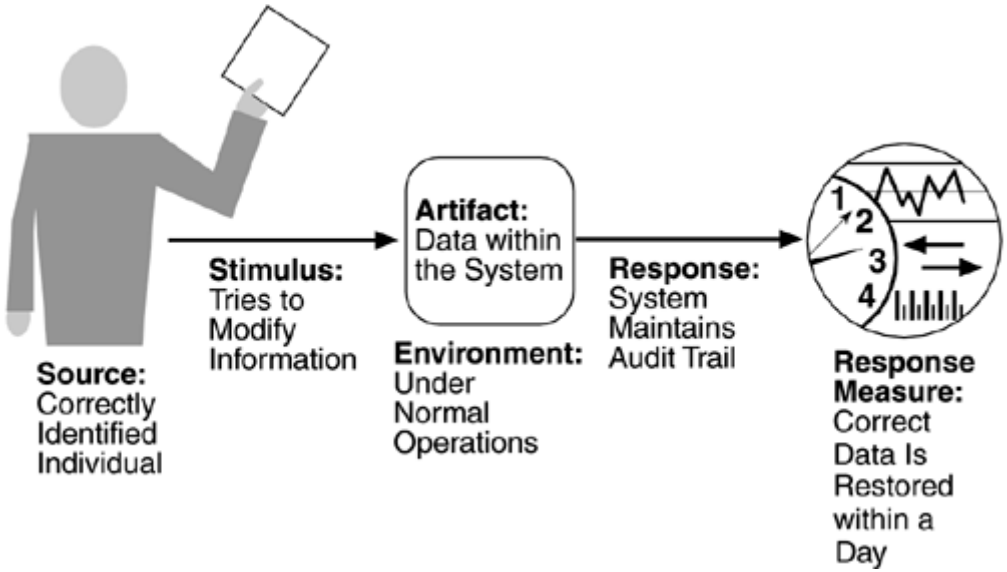
典型质量属性的架构设计：性能(performance)

- 典型战术：
  - 提高计算效率(改进算法、引入cache等)、减少计算开销(最小化通讯数据量)、控制请求频度、限制执行时间、限制队列大小；
  - 引入并发、增加可用资源、负载平衡；
  - 对请求的调度策略(FIFO、固定优先级、动态优先级)；



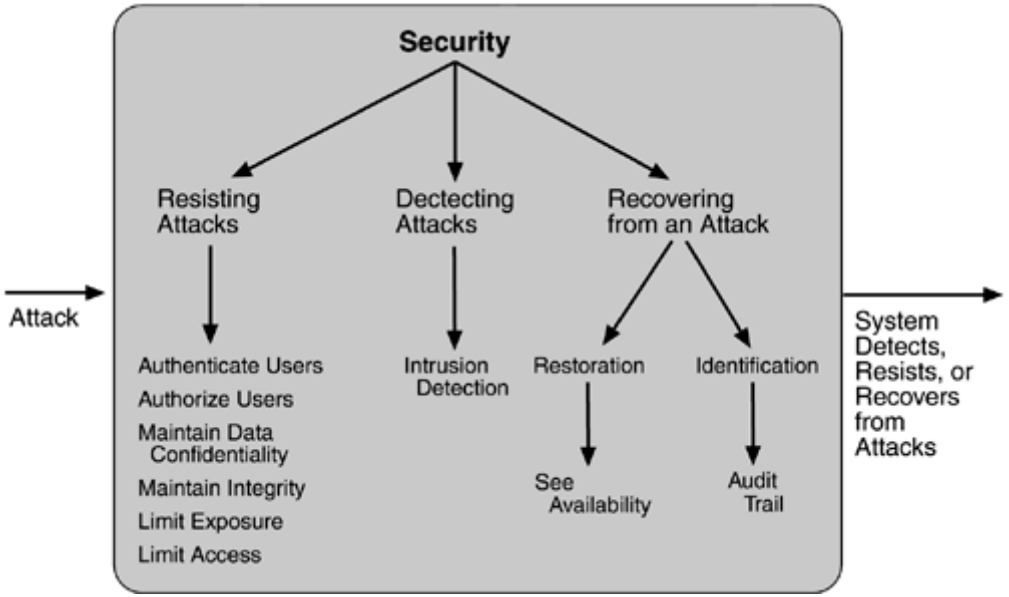
典型质量属性的架构设计：安全性(security)

- 安全性：系统在向合法用户提供服务的同时，组织非授权使用的能力
  - 未经授权试图访问服务或数据；
  - 试图修改数据或服务；
  - 试图使系统拒绝向合法用户提供服务；
- 关注点：抵抗攻击、检测攻击、从攻击中恢复。



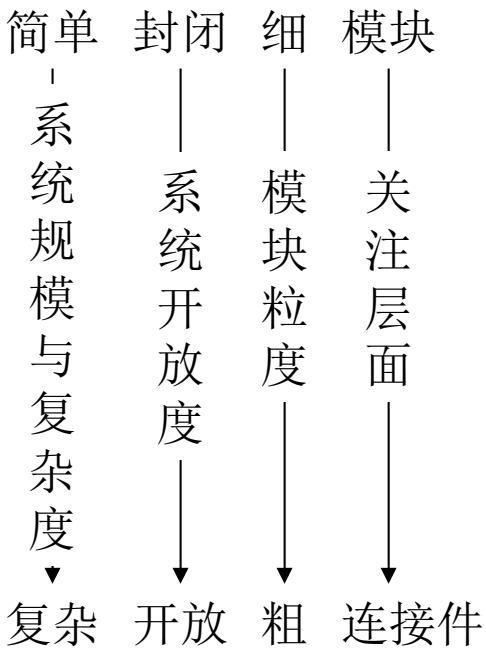
典型质量属性的架构设计：安全性(security)

- 抵抗攻击——对用户进行身份认证、对用户进行授权、维护数据的机密性、限制暴露的信息、限制访问；
- 检测攻击——模式发现、模式匹配；
- 从攻击中恢复——将服务或数据回复到正确状态、维持审计追踪。

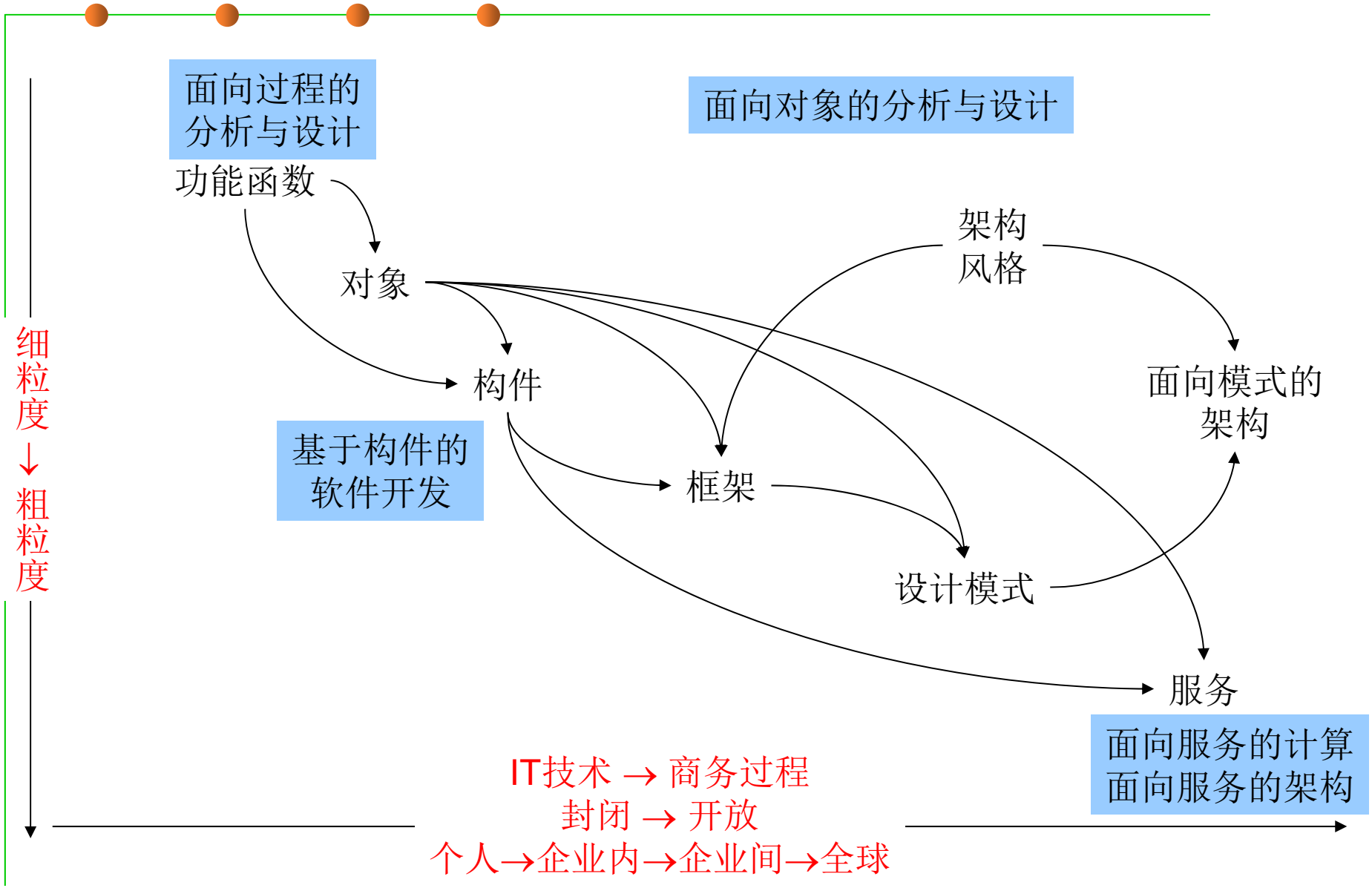


软件架构的发展与演化

- 系统 = 算法 + 数据结构 (1960's)
- 系统 = 子程序 + 函数调用 (1970's)
- 系统 = 对象 + 消息 (1980's)
- 系统 = 构件 + 连接件 (1990's)
- 系统 = 服务 + 服务总线 (2000's)



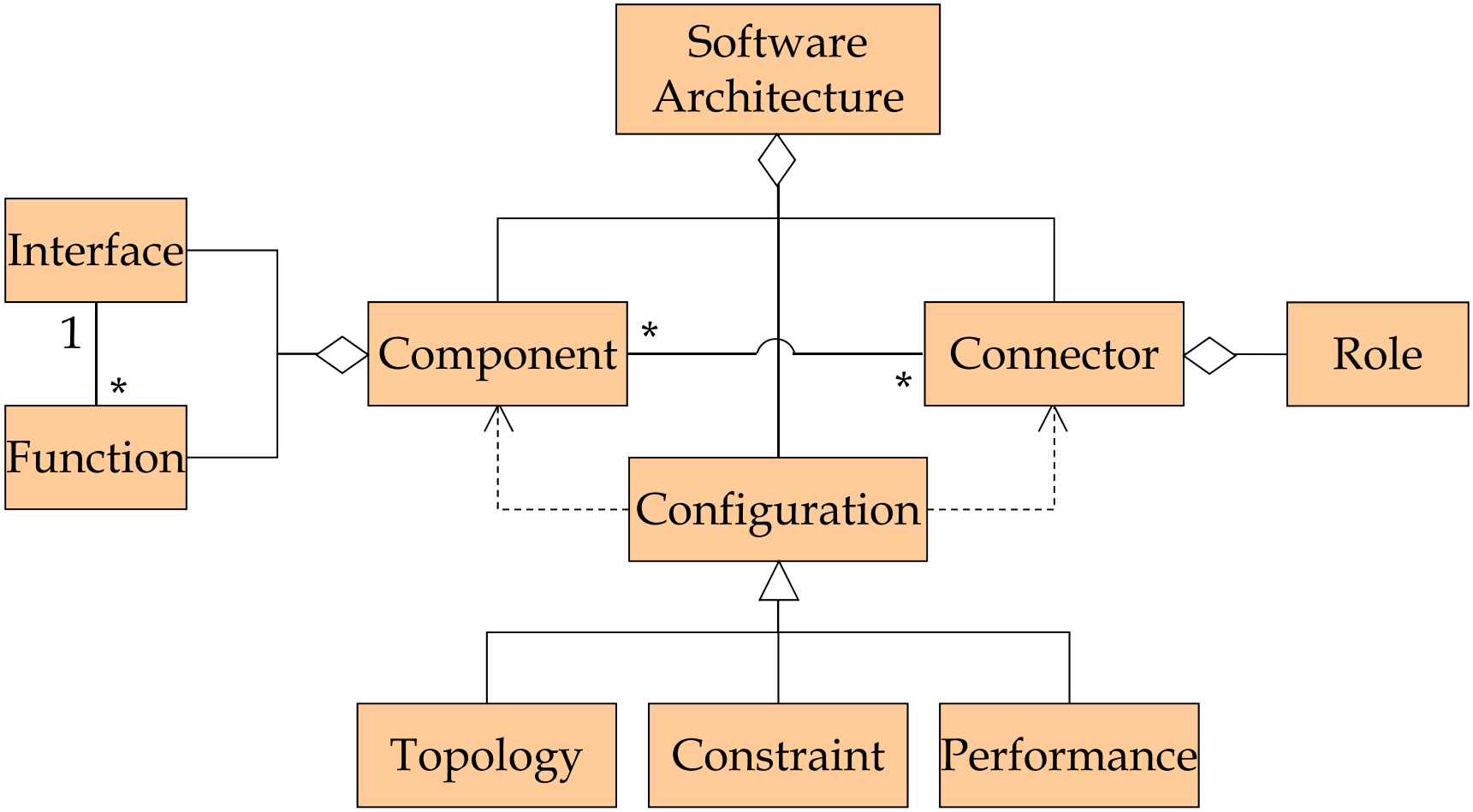
软件架构的演化史


面向过程的
分析与设计
面向对象的分析与设计
功能函数
对象
架构
风格
构件
面向模式的
架构
基于构件的
软件开发
框架
设计模式
服务
面向服务的计算
面向服务的架构
细粒度↓粗粒度
IT技术 → 商务过程
封闭 → 开放
个人→企业内→企业间→全球

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2. 软件架构中的核心概念

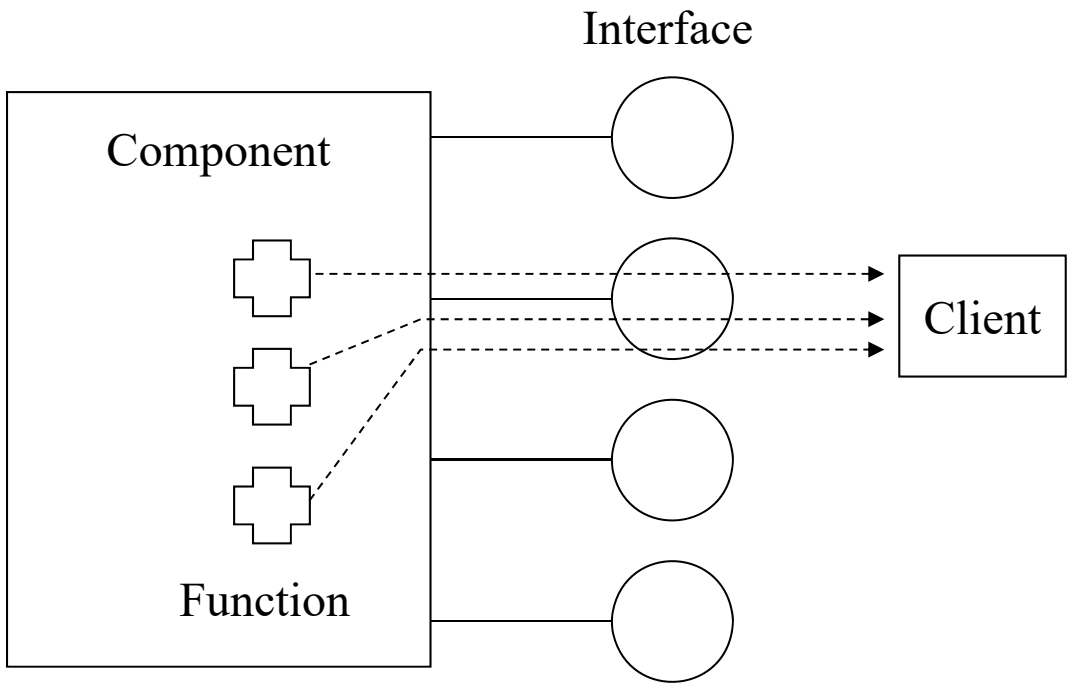
架构中的核心概念


Software Architecture
Interface
1
*
Function
Component
*
*
Connector
Role
Configuration
Topology
Constraint
Performance

构件(Component)

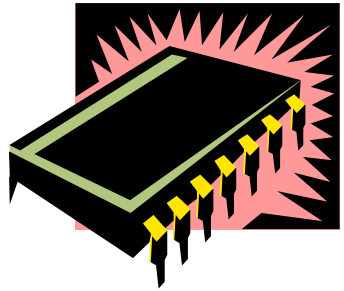
- “构件”是具有某种功能的可复用的软件结构单元，表示了系统中主要的计算元素和数据存储，具有提供的接口描述。

- 构件是一个抽象的概念，任何在系统运行中承担一定功能、发挥一定作用的软件体都可看作是构件。
  - 程序函数、模块
  - 对象、类
  - 数据库
  - 文件
  - ….



接口(Interface)与功能(Function)

- 构件=接口+功能
- 构件作为一个封装的实体，只能通过其**接口(Interface)**与外部环境交互，表示了构件和外部环境的**交互点，内部具体实现则被隐藏起来(Black-box)**；



- 构件接口与其内部实现应严格分开
- 构件内部所实现的功能以**功能(functions、behaviors)**的形式体现出来，并通过接口向外发布，进而产生与其它构件之间的关联。

构件 vs. 对象

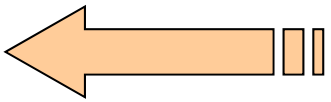
- 抽象的级别不同，构件是设计概念，而对象是实现技术
- 规模
  - 对象一般较小
  - 构件可以小(一个对象) 或大(一系列对象或一个完整的应用)
- 在对构件操作时不允许直接操作构件中的数据，数据真正被封装了。而对象的操作通过公共接口部分，这样数据是可能被访问操作的
- 对象的复用是基于技术的复用（继承），构件的复用是基于功能的复用（组装）

连接(Connection)

- 连接(Connection)：构件间建立和维护行为关联与信息传递的途径；
- 连接需要两方面的支持：
  - 连接发生和维持的机制——实现连接的物质基础(连接的机制)；
  - 连接能够正确、无二义、无冲突进行的保证——连接正确有效的进行信息交换的规则(连接的协议)。
  - 简称“机制”(mechanism)和“协议”(protocol)。

连接的机制(Mechanism)

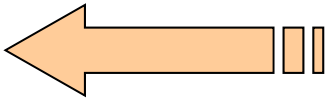
- **计算机硬件提供了一切连接的物理基础：**
  - 过程调用、中断、存储、堆栈、串行I/O、并行I/O等；
- **基础控制描述层：**
  - 过程调用、动态约束、中断/事件、流、文件、网络等；
- **资源及管理调度层：**
  - 进程、线程、共享、同步、并行、分时并发、事件、消息、异常、远程调用、注册表、剪贴板、动态连接、API等；
- **系统结构模式层：**
  - 管道、解释器、编译器、转换器、浏览器、中间件、ODBC等。



计算机组成原理
计算机系统结构
汇编语言

C高级程序语言
操作系统
计算机网络

操作系统



软件工程
软件架构

连接的协议(Protocol)

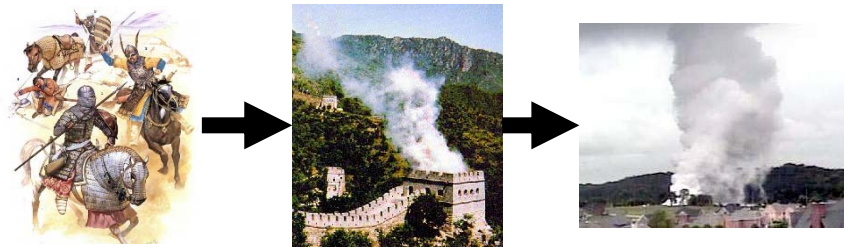
- 协议(Protocol)是连接的规约(Specification);
- 连接的规约是建立在物理层之上的有意义信息形式的表达规定
  - 对过程调用来说：参数的个数和类型、参数排列次序；

    例：double getHighestScore (int courseID, String classID) {…}
  - 对消息传送来说：消息的格式

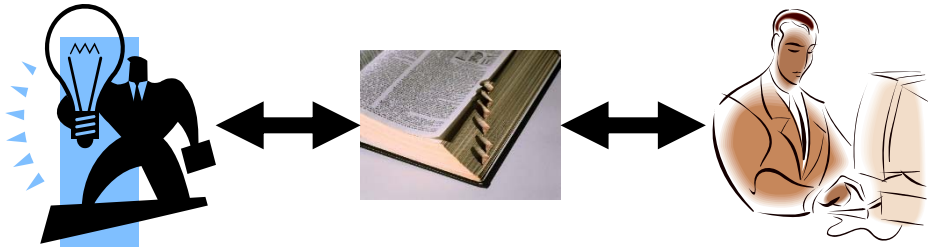
    例：class Message {int msgNo; String bookName; String status;}

- 目的：使双方能够互相理解对方所发来的信息的语义。

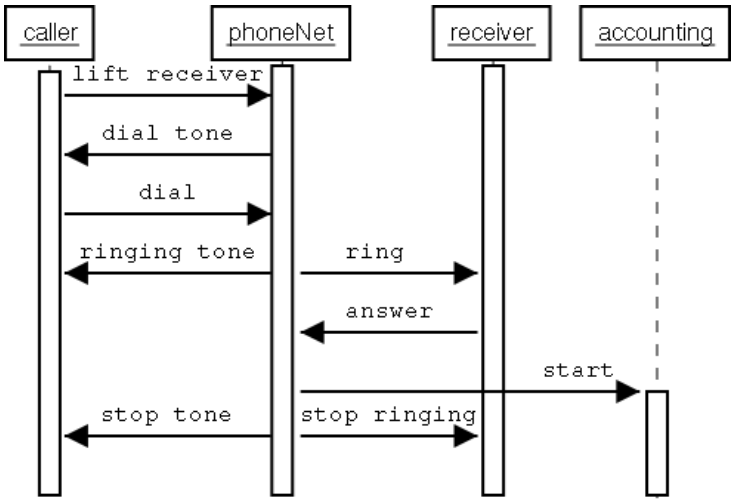
现实中的“连接”



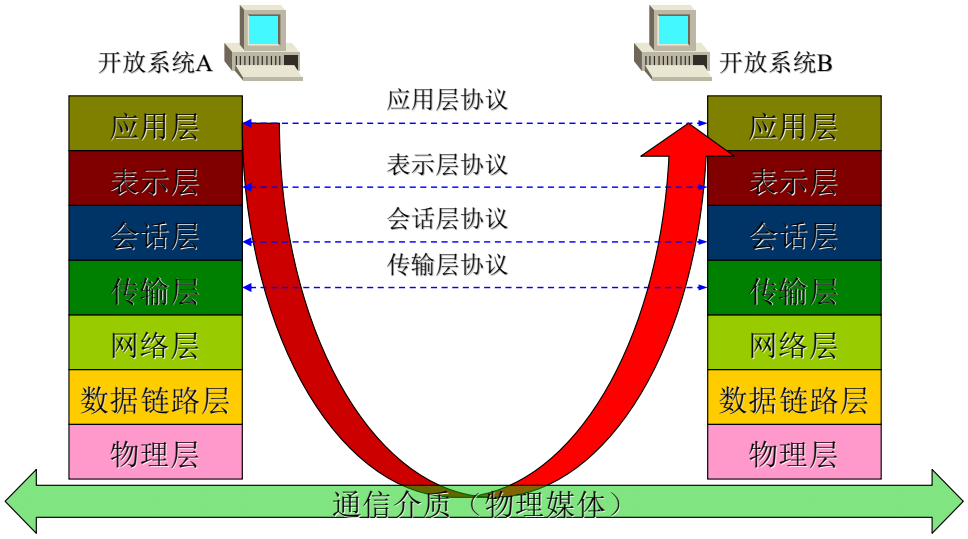
简单的机制、简单的协议



简单的机制、复杂的协议



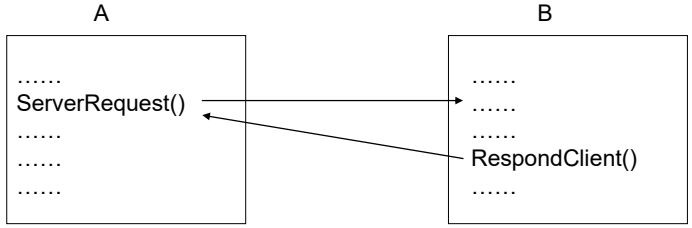
复杂的机制、简单的协议



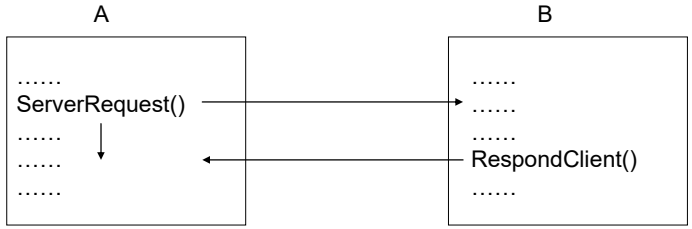
复杂的机制、复杂的协议

连接的种类

- 从连接目的与手段看
  - 函数/过程调用；　　A { call B.f();}　　B { f(); }
  - 事件/消息发送；　　A { send msg (m) to B;}　　B { receive m and do sth;}
  - 数据传输；　　A {write data to DB; }　　B {read data from DB;}
  - ..
- 除了连接机制/协议的实现难易之外，影响连接实现复杂性的因素之一是“有无连接的返回信息和返回的时间”，分为：
  - 同步 (Synchronous)
  - 异步 (Asynchronous)



同步连接机制



异步连接机制

连接件(Connector)

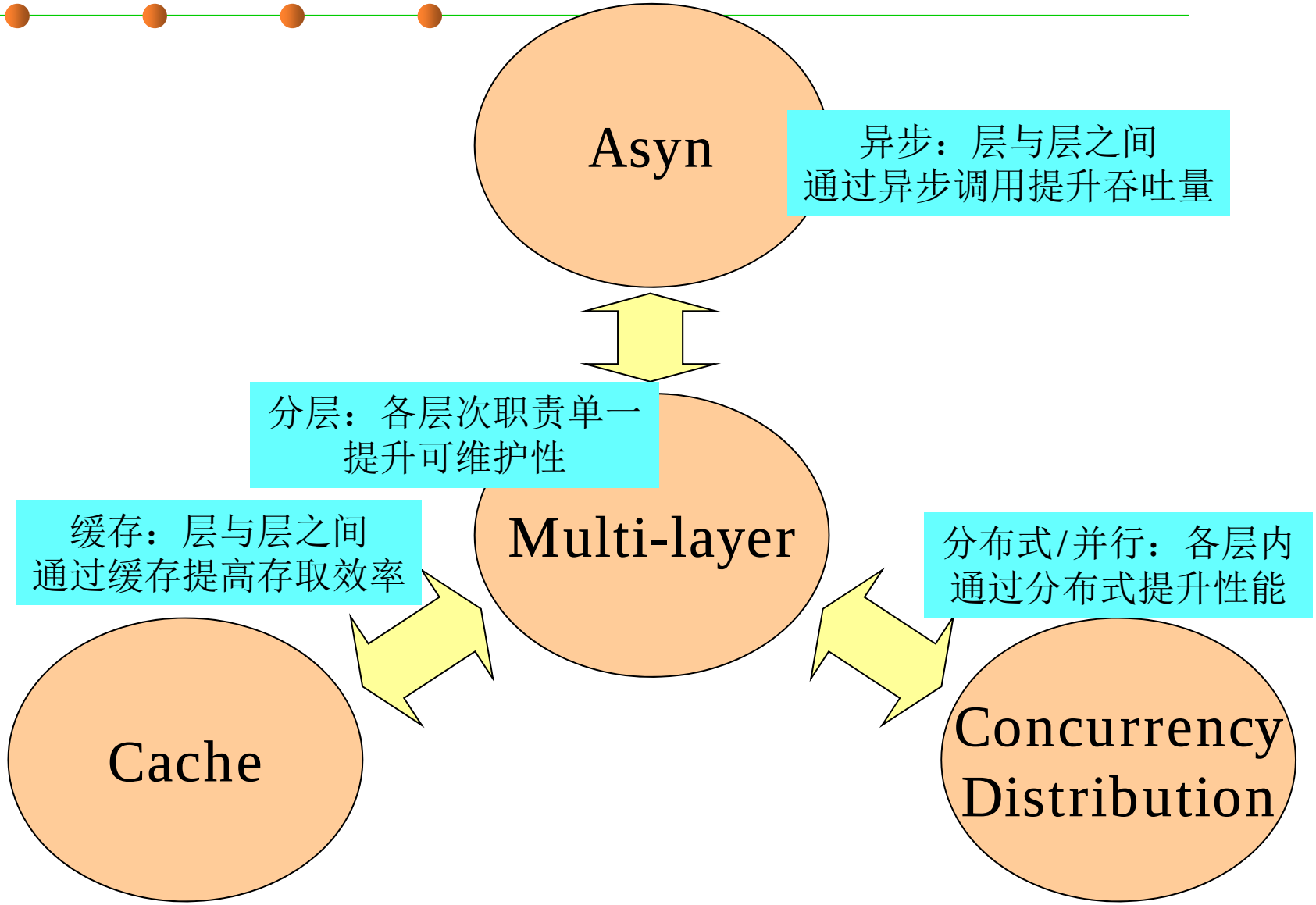
- **连接件(Connector)：表示构件之间的交互并实现构件之间的连接，如：**
  - 管道(pipe)
  - 过程调用(procedure call)
  - 事件广播(event broadcast)
  - 客户机-服务器(client-server)
  - 数据库连接(SQL)

- **典型连接件：CICS、MQ、JMS**
- **连接件也可看作一类特殊的构件，区别在于：**
  - 一般构件是软件功能设计和实现的承载体；
  - 连接件是负责完成构件之间信息交换和行为联系的专用构件

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

3. 软件架构的四大思想

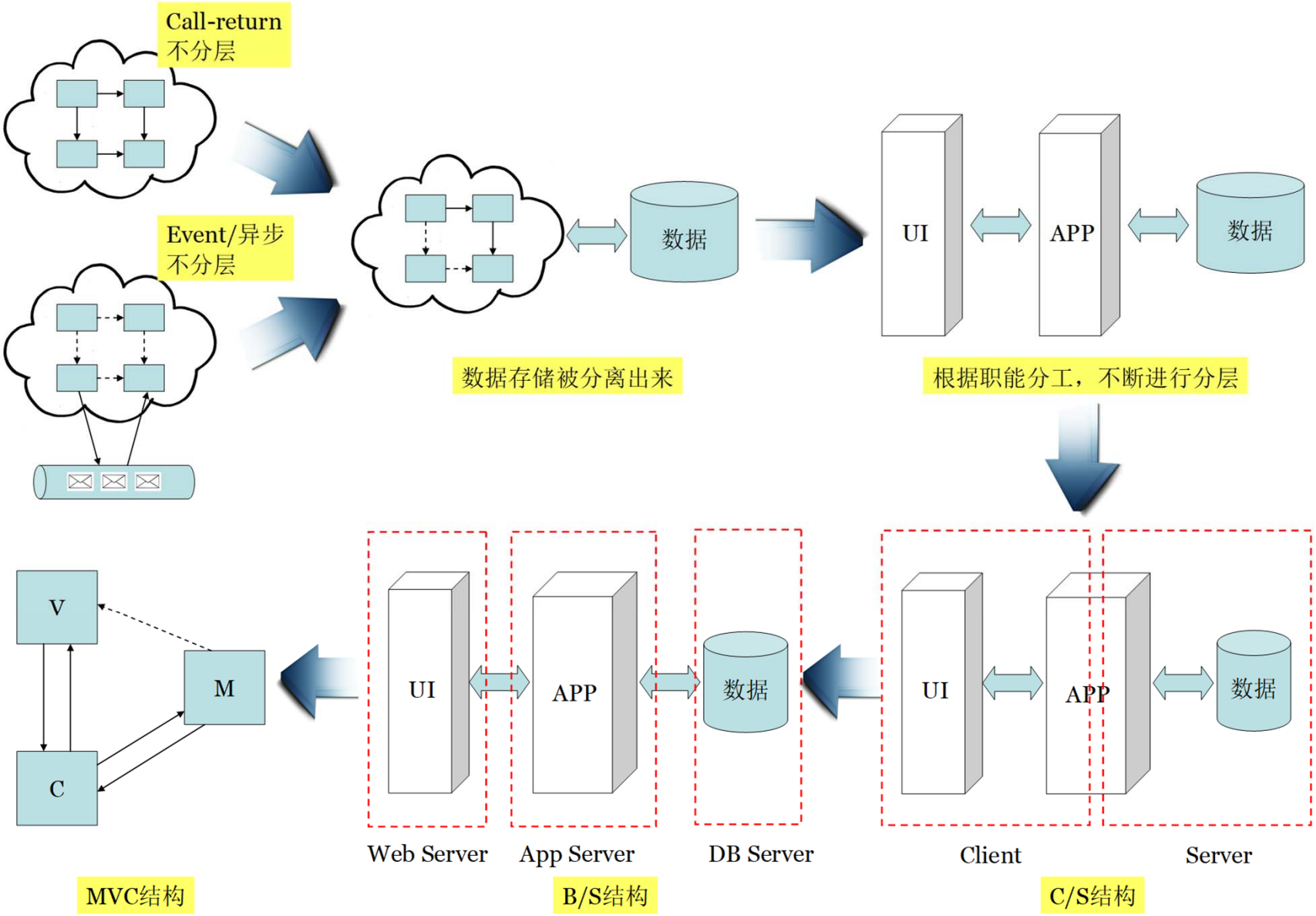
“软件架构四大器”


Asyn
异步：层与层之间
通过异步调用提升吞吐量
分层：各层次职责单一
提升可维护性
Multi-layer
缓存：层与层之间
通过缓存提高存取效率
分布式/并行：各层内
通过分布式提升性能
Cache
Concurrency
Distribution

软件架构的基本模式

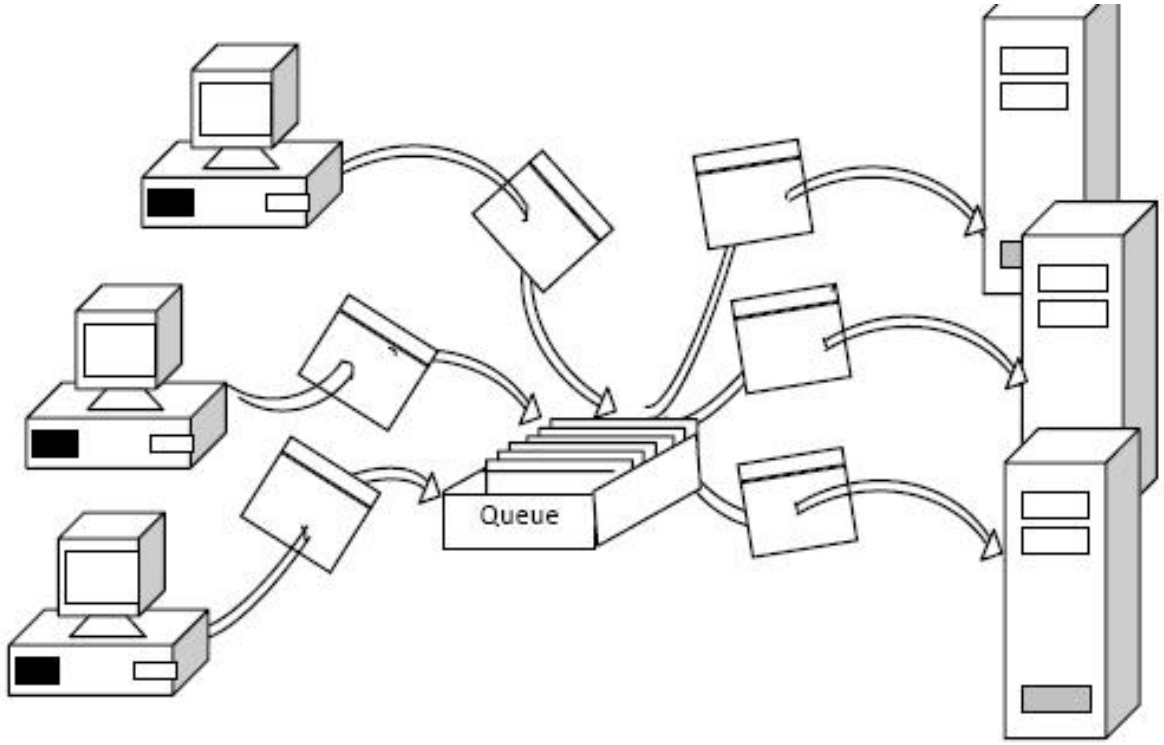
- **分层：C/S、B/S、多层，数据、计算与显示的分离(MVC)**
  - 一个模块做很多事情➔各负其责，分工明确
  - 牺牲了效率，提升了可维护性。
- **异步：事件、消息**
  - 请求之后等待结果(同步)➔请求之后继续执行，后续等待结果(异步)
  - 性能(吞吐量)提高，但实时性变差；
- **缓存：页面缓存、数据缓存、消息缓存**
  - 直接到源头去取➔预取
  - 提高了效率，但牺牲了准确性
- **并发(分布式)：集群、负载均衡、分布式数据库**
  - 原本一个模块处理很多请求➔多个模块共同处理这些请求
  - 提高了吞吐量、响应时间、可靠性，但需要考虑同步等复杂问题

分层


Call-return
不分层
Event/异步
不分层
数据
数据存储被分离出来
UI
APP
数据
根据职能分工，不断进行分层
V
M
C
MVC结构
UI
APP
数据
Web Server
App Server
DB Server
B/S结构
UI
APP
数据
Client
Server
C/S结构

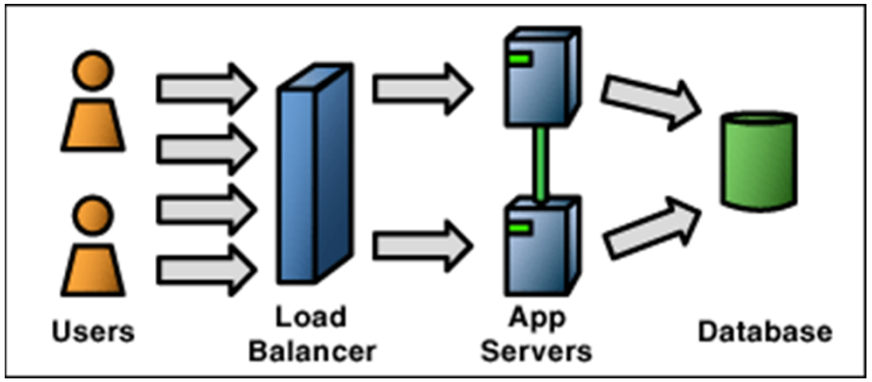
异步

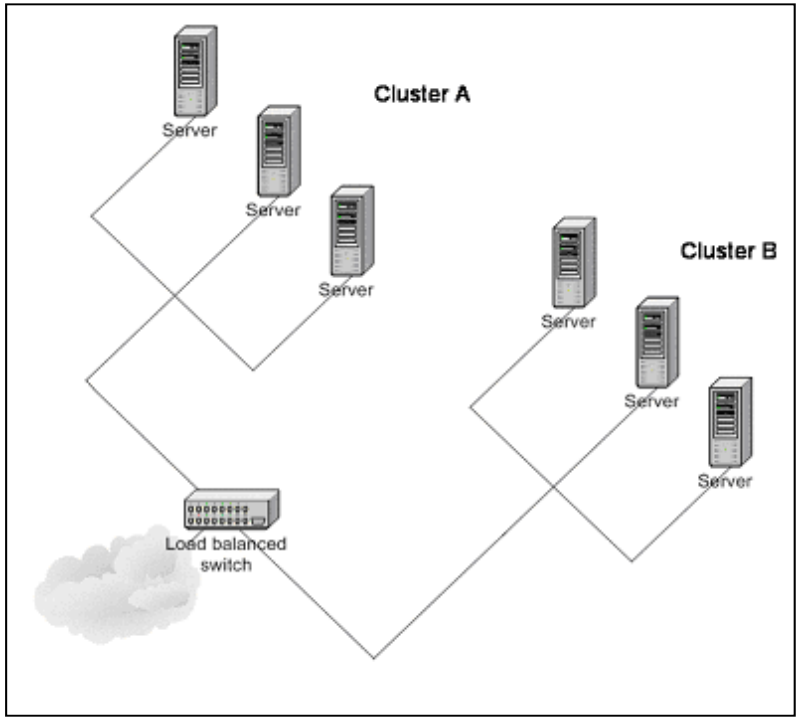
- 通过“第三者”——消息——完成模块之间的功能调用。



分布式+并发

- 事实上，功能层并不一定只驻留在同一台服务器上，数据层也是如此；
- 如果功能层(或数据层)分布在多台服务器上，那么就形成了**基于集群(Cluster)的C/S物理分布模式**。

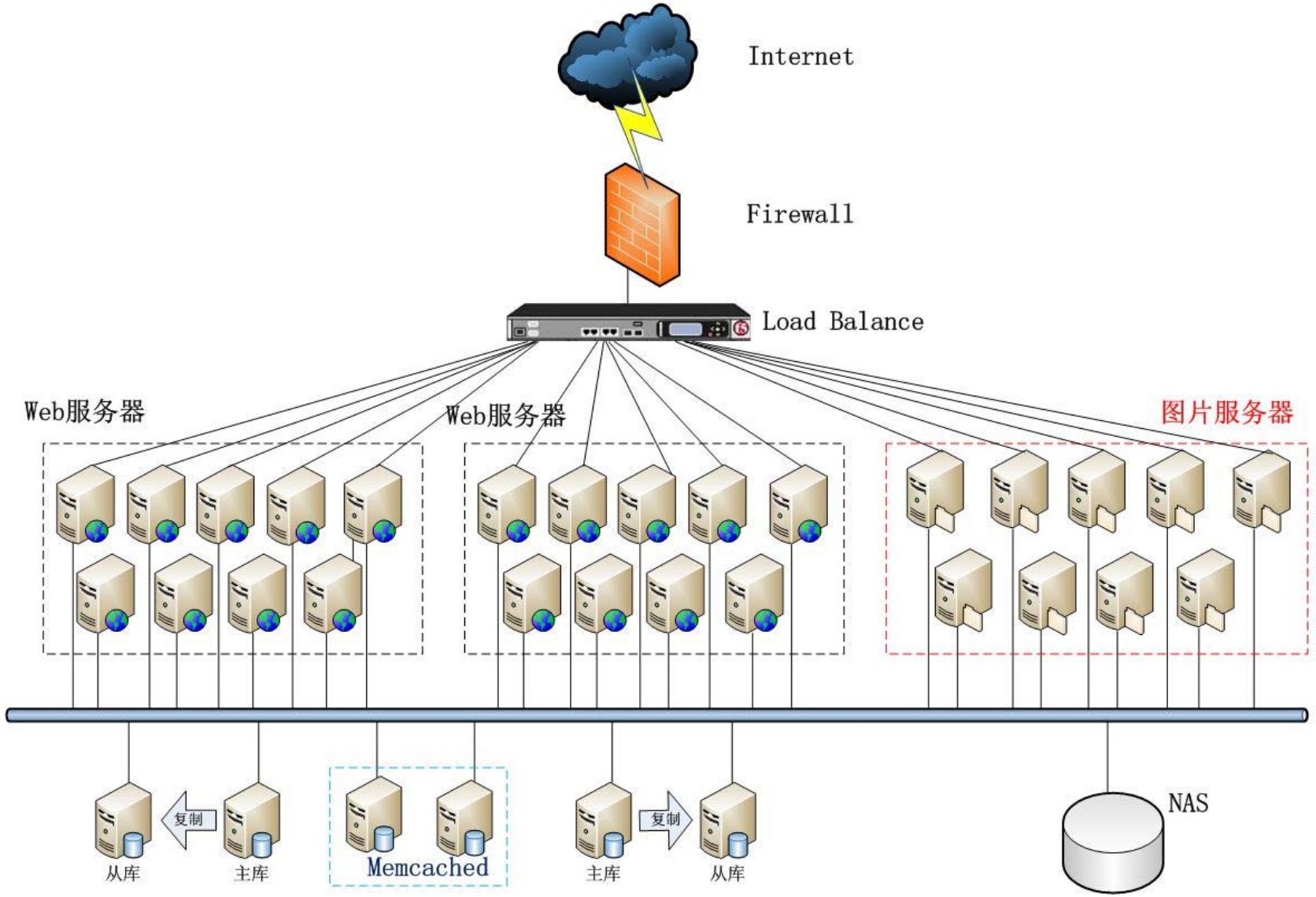




集群(Cluster)

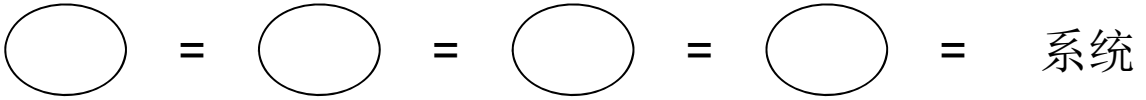
- A cluster is a group of loosely coupled servers that work together closely so that in many respects it can be viewed as though it were a single server. (一组松散耦合的服务器，共同协作，可被看作是一台服务器)
- Clusters are usually deployed to improve speed, reliability and availability over that provided by a single server, while typically being much more cost-effective than single computers of comparable speed or reliability. (用来改善速度、提高可靠性与可用性，降低成本)
- Load-balancing is a key issue in clusters. (负载平衡是集群里的一个关键要素)
- Physical cluster and logical cluster(物理集群、逻辑集群)

集群(Cluster)

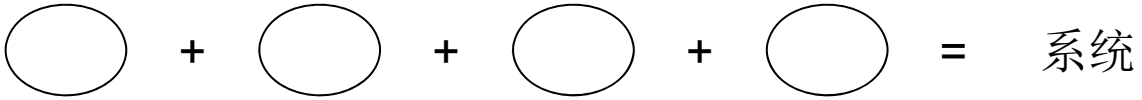

Internet
Firewall
Load Balance
Web服务器
Web服务器
图片服务器
复制
从库
主库
Memcached
主库
复制
从库
NAS

集群的方法

- 集群内各服务器上的内容保持一致(通过并发/冗余提高可靠性与可用性)

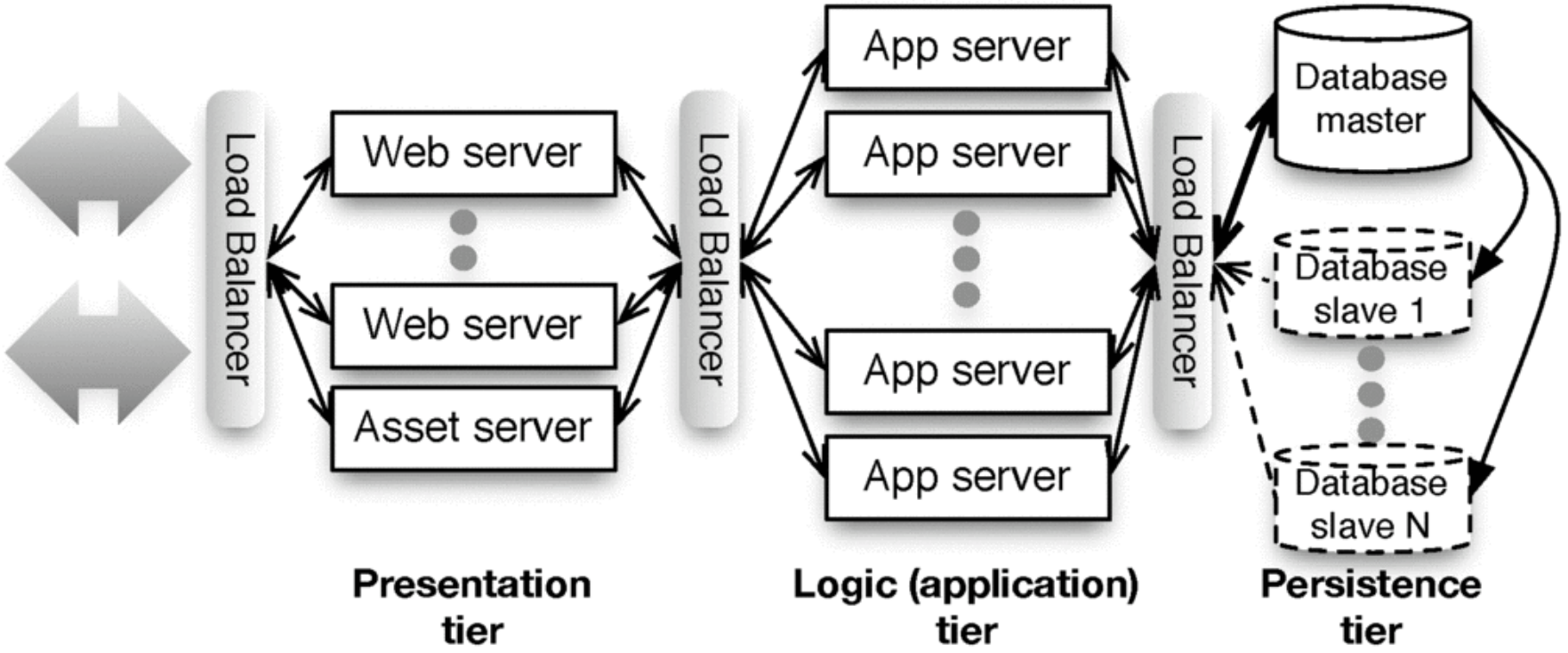


- 集群内各服务器上的内容之和构成系统完整的功能/数据，是对系统功能集合/数据集合的一个划分(通过分布式提高速度与并发性)



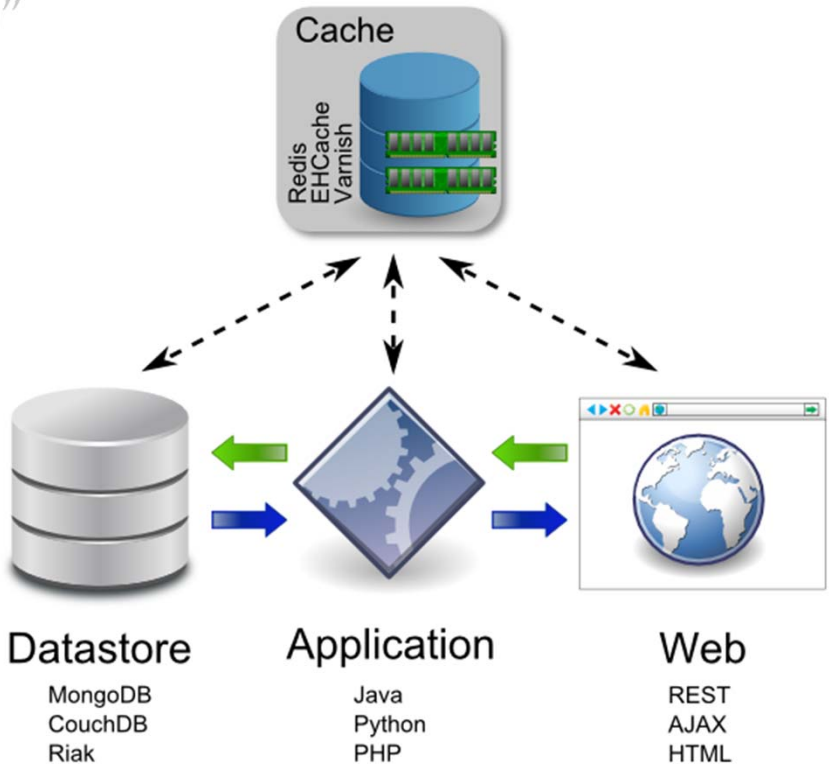
负载均衡(Load Balance)

- 三个层次的负载均衡



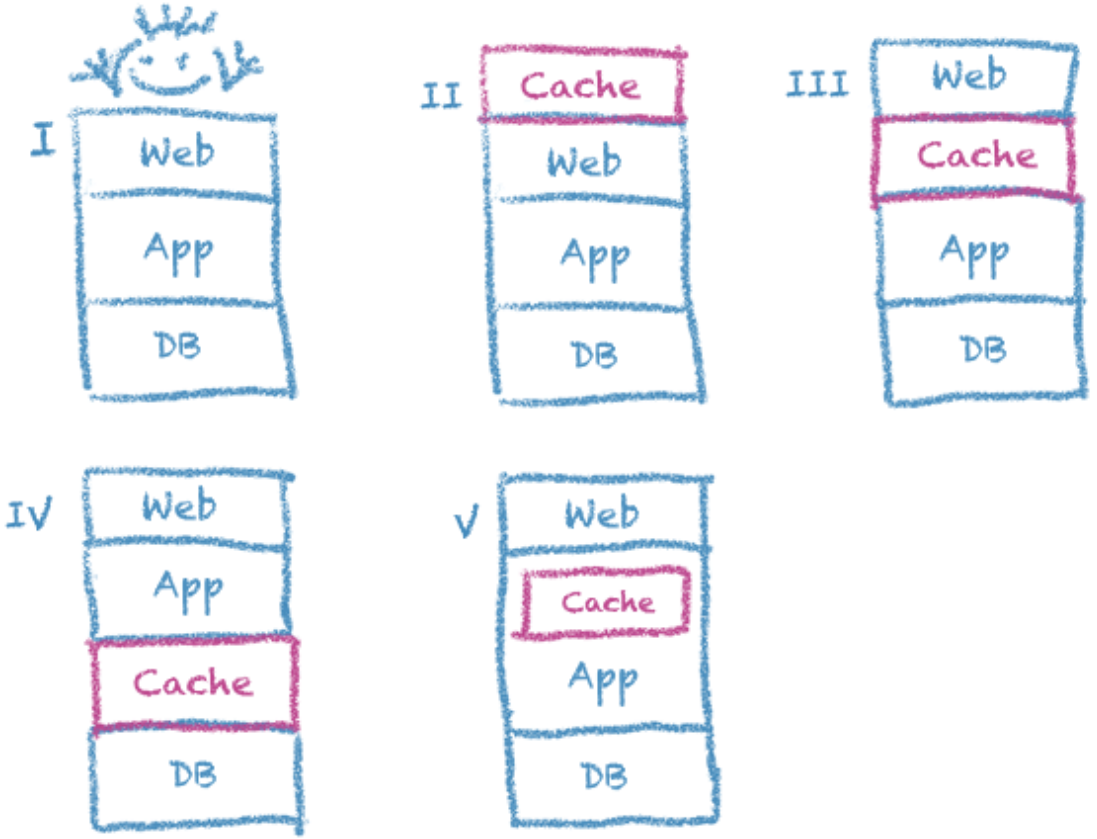
缓存(cache)

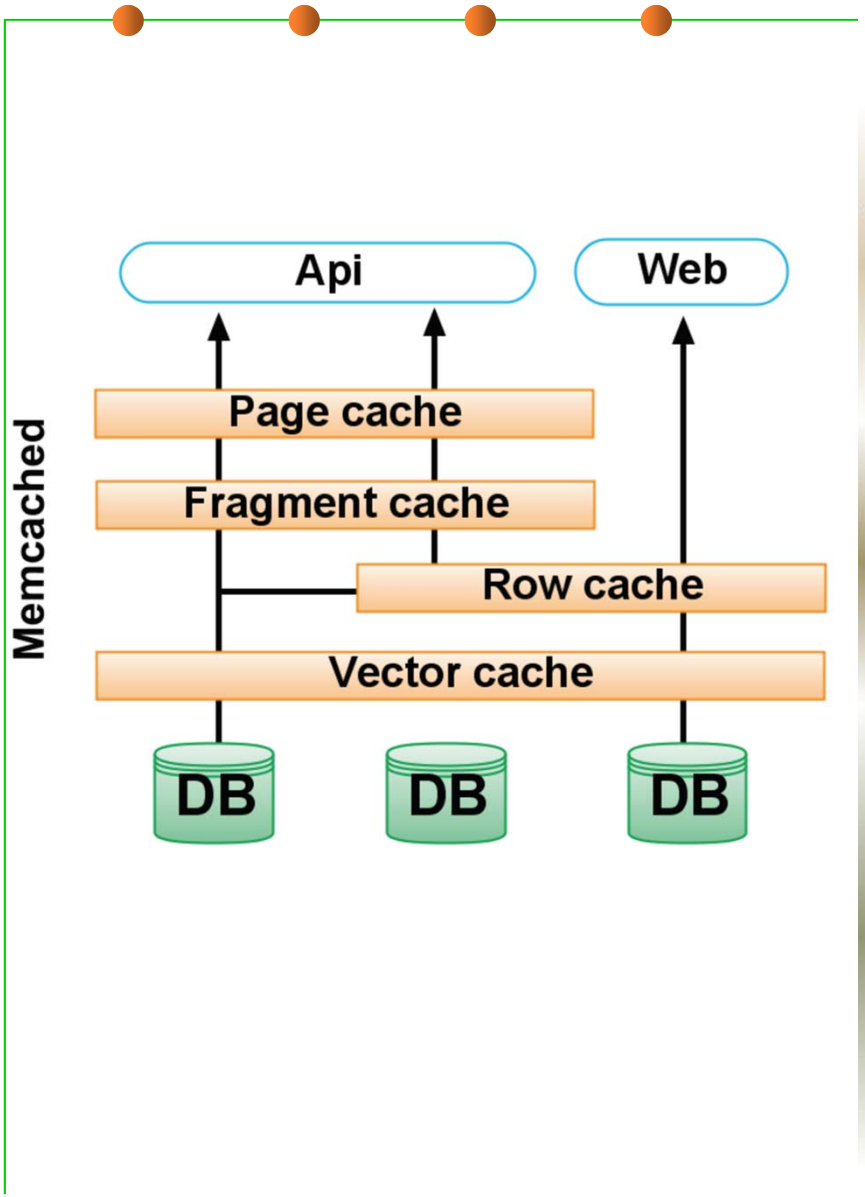
- **缓存：解决web应用性能瓶颈。**
  - “缓存就像清凉油，哪里不舒服，抹一下就好了”
- **性能瓶颈的体现：高延时、拥塞和服务器负载**
  - “下载HTML只需要总用户响应时间的10-20%，剩下的80-90%全部用于下载页面中的其它组成内容(如各种图像等)”
- **缓存：**
  - 一种临时存储，将数据复制到不同于原始数据源的位置，访问缓存数据的速度比访问原始数据的速度要快得多，从而可以减小服务器负载和带宽消耗，提高用户性能。

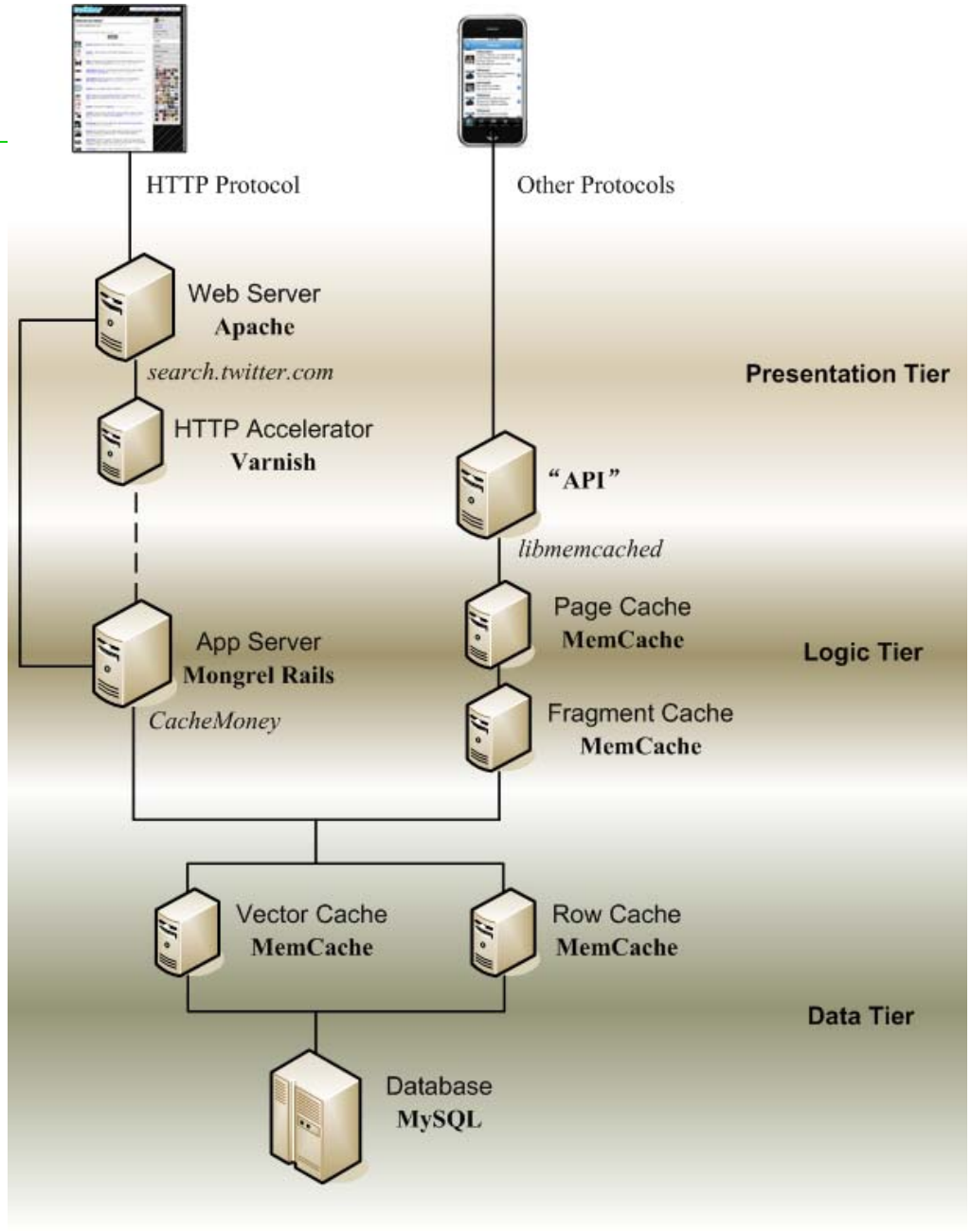


缓存(cache)

- Cache可以处在多个不同的位置




Memcached
Api
Web
Page cache
Fragment cache
Row cache
Vector cache
DB
DB
DB


HTTP Protocol
Other Protocols
Web Server
Apache
search.twitter.com
HTTP Accelerator
Varnish
"API"
libmemcached
Page Cache
MemCache
App Server
Mongrel Rails
CacheMoney
Fragment Cache
MemCache
Vector Cache
MemCache
Row Cache
MemCache
Database
MySQL
Presentation Tier
Logic Tier
Data Tier

课堂讨论

- “结合分层、cache、分布式/并发、异步四种典型架构思想，阐述各自的特征，以及对NFR所做的改善，并且以淘宝、京东、亚马逊等电子商务网站为例，给出应用举例”
- 2组发言机会，每组阐述2种典型架构思想。

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

结束

2017年11月8日