WPF高级应用培训

Agenda

- 从战略的角度看软件开发与架构设计
- WPF高级应用之性能
- 高级应用实践
- TDD

Unit 1: 从战略的角度看软件开发与架构设计

关于学习的思考

- 学什么?
- 怎么学?
- 目的?

学习方法与信息吸收

听	***5%***
听和读	***20%***
演示	***30%***
讨论	***50%***
互动	***70%***
教给他人	***90%***

我们今天为什么在这里?

The **GOAL** is to deliver

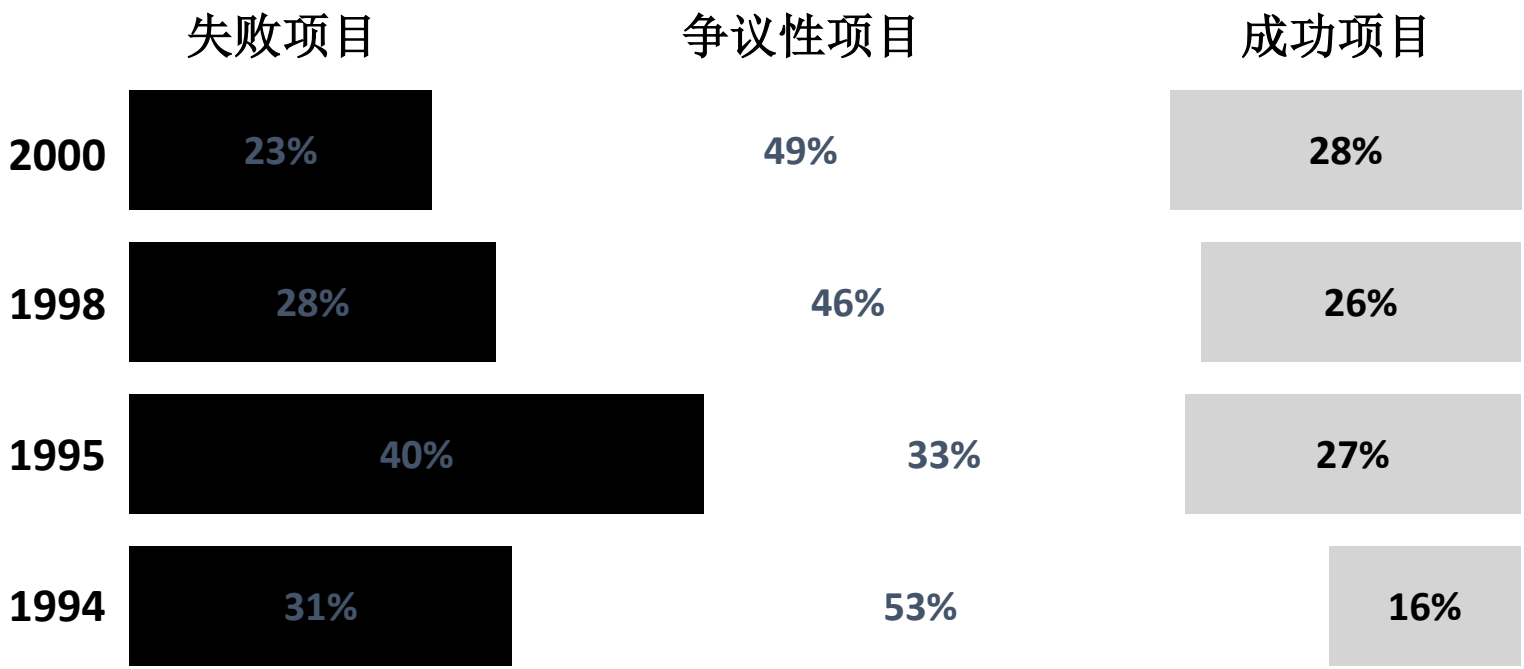
quality products

on time and ***on budget***

which meet the customer's

real needs.

成功并非易事



上图描述了斯坦迪什集团自 1994 年以来对 30,000 个应用软件项目进行调查的结果，调查对象为美国大、中、小型的跨行业公司。

来源：斯坦迪什国际集团，《Extreme Chaos》（2000）

成功的障碍

- 关注点问题：
  - 目标和职能分离
  - 业务和技术分离
- 沟通问题：
  - 缺乏共同的语言和过程
    - 目标不明确
    - 没有范围变更管理
  - 无法以一个团队的方式进行沟通和运作
- 变化问题：
  - 过程管理不够灵活，难以适应项目的变化

从战略的高度看待架构设计

商业目标决定架构目标！

使命 战略 结构 结果

使命

- **AT&T（1920s）：** 让每一个美国家庭和公司都装上电话

Microsoft 的使命

Microsoft（1980s）： 让每个办公室和每个家庭的桌上都摆上一台电脑

Google的使命

整合天下信息，让人人能获取，使人人能受益.

谢尔盖.布林和拉里.佩奇在与工程师一起讨论产品时，总是问：

这对世界有什么好处?

世界会因此而变得更美好吗?

使命决定战略

- 使命是什么：使命即定位

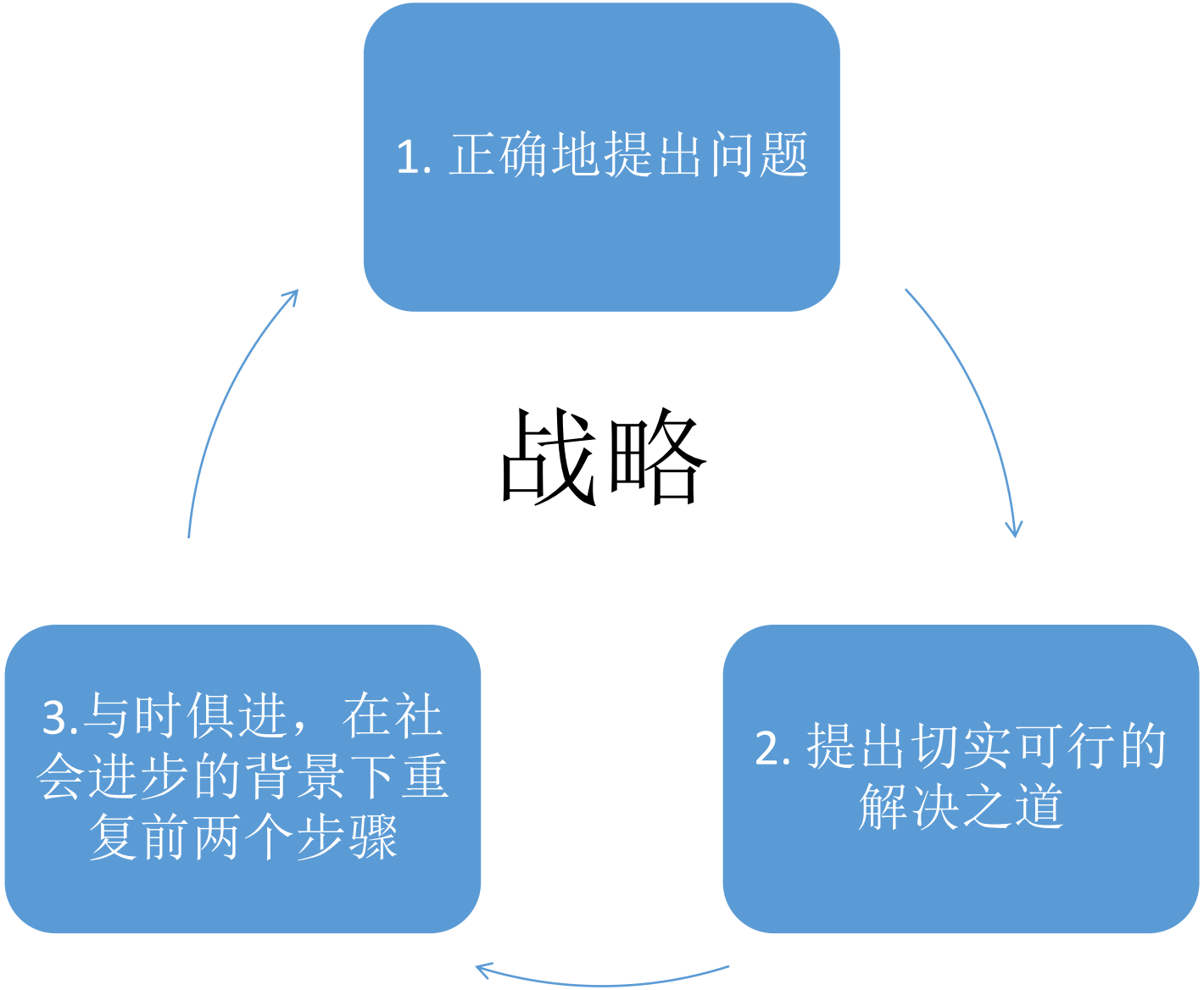
- 什么是定位：

定位三问　1）我的业务是什么？

2）我的顾客是谁？

3）我为顾客提供的独特价值是什么？

好的架构设计过程



什么是战略?

企业生存、发展战胜对手和超越对手的策略.

红海战略--------战胜对手--------生存

蓝海战略--------超越对手--------发展

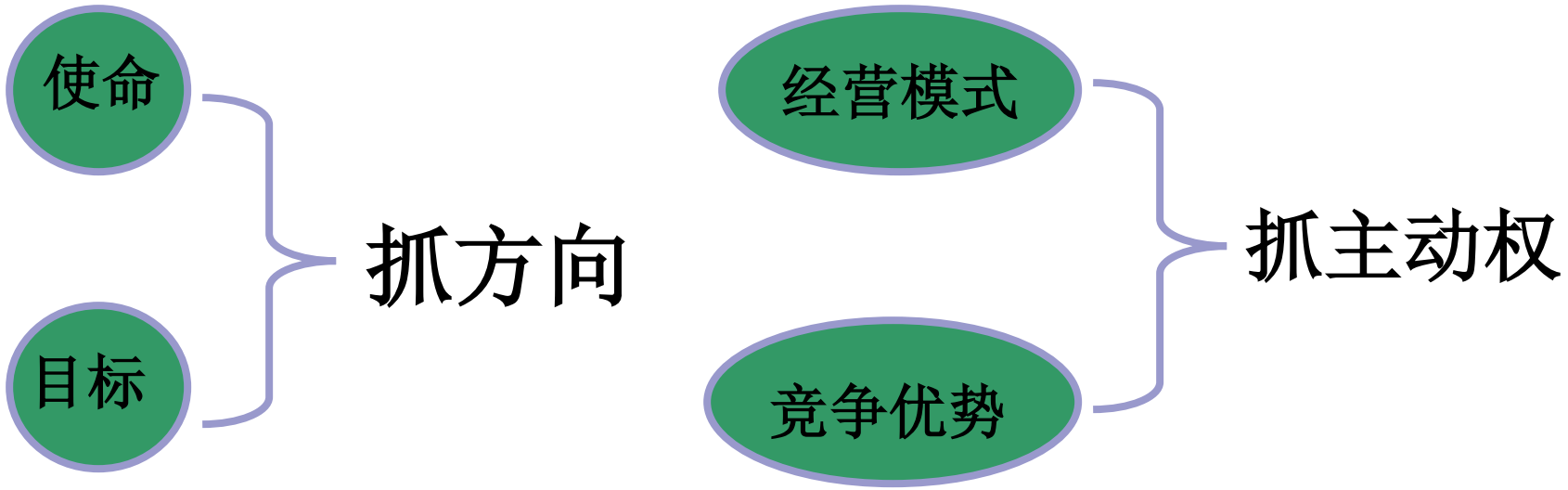
毛泽东论战略

抓战略主要抓两点：

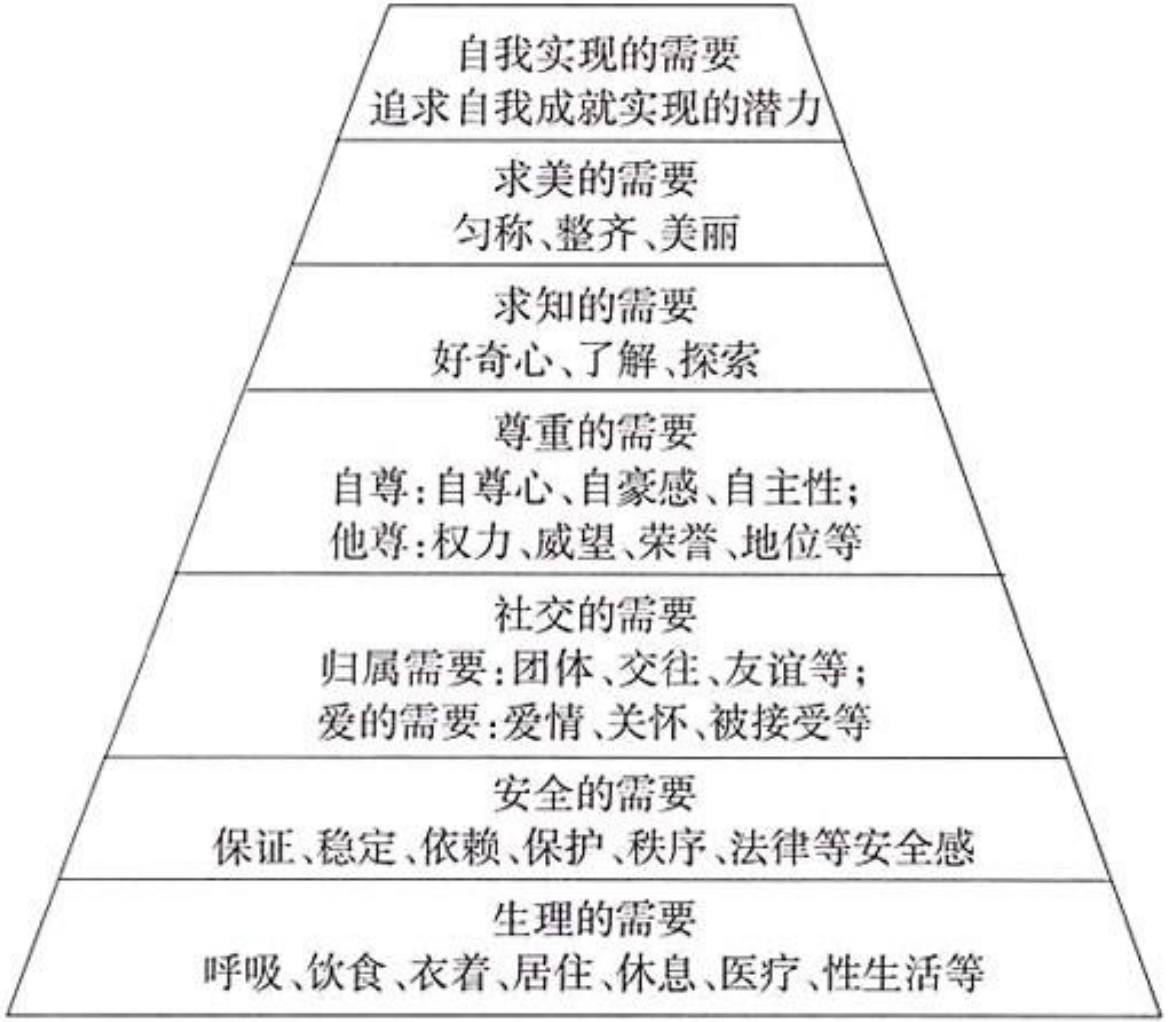
一是抓主动权，二是抓战略方向。

失去了主动权，就等于被打败。

战略管理的框架



Abraham Maslow 1954



用户满意度

用户满意度 = f(感知价值，期望价值)

软件架构的层次

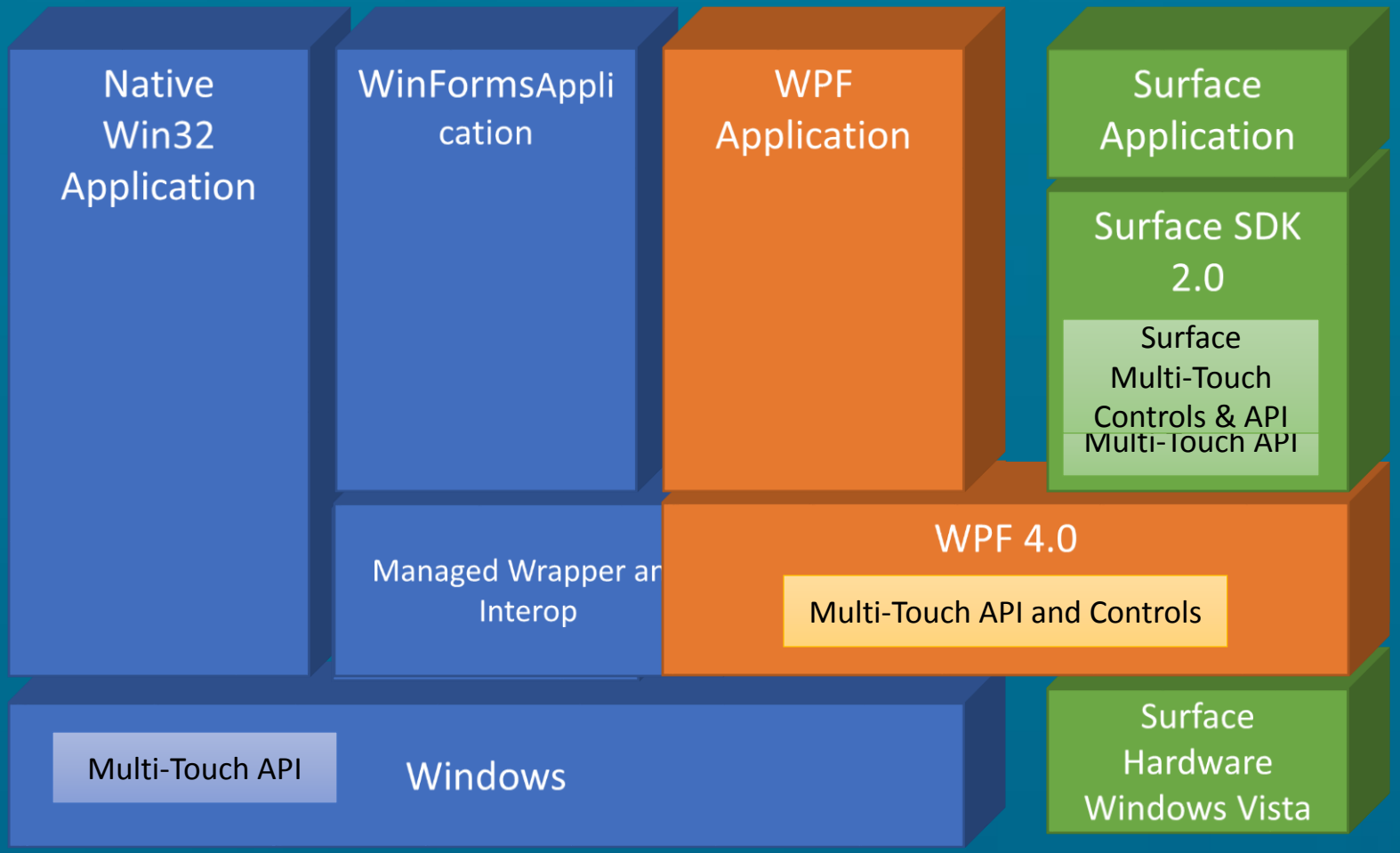
- Enterprise层
  - 说明：
    - 最高层，人数极少
  - 特征：
    - 关注整个机构、企业所有IT 系统的整体能力
    - 从整体着眼、与业务紧密相关、与IT 规划相关
- Application层
  - 说明：
    - 系统架构最高层，大型系统需要有一个架构组
  - 特征：
    - 负责应用系统的架构，奠定系统建设的基础
    - 关注系统内部的构成和子系统/模块的分划
    - 需要负责与外部相关系统的互联互通

Unit2：WPF高级应用之性能

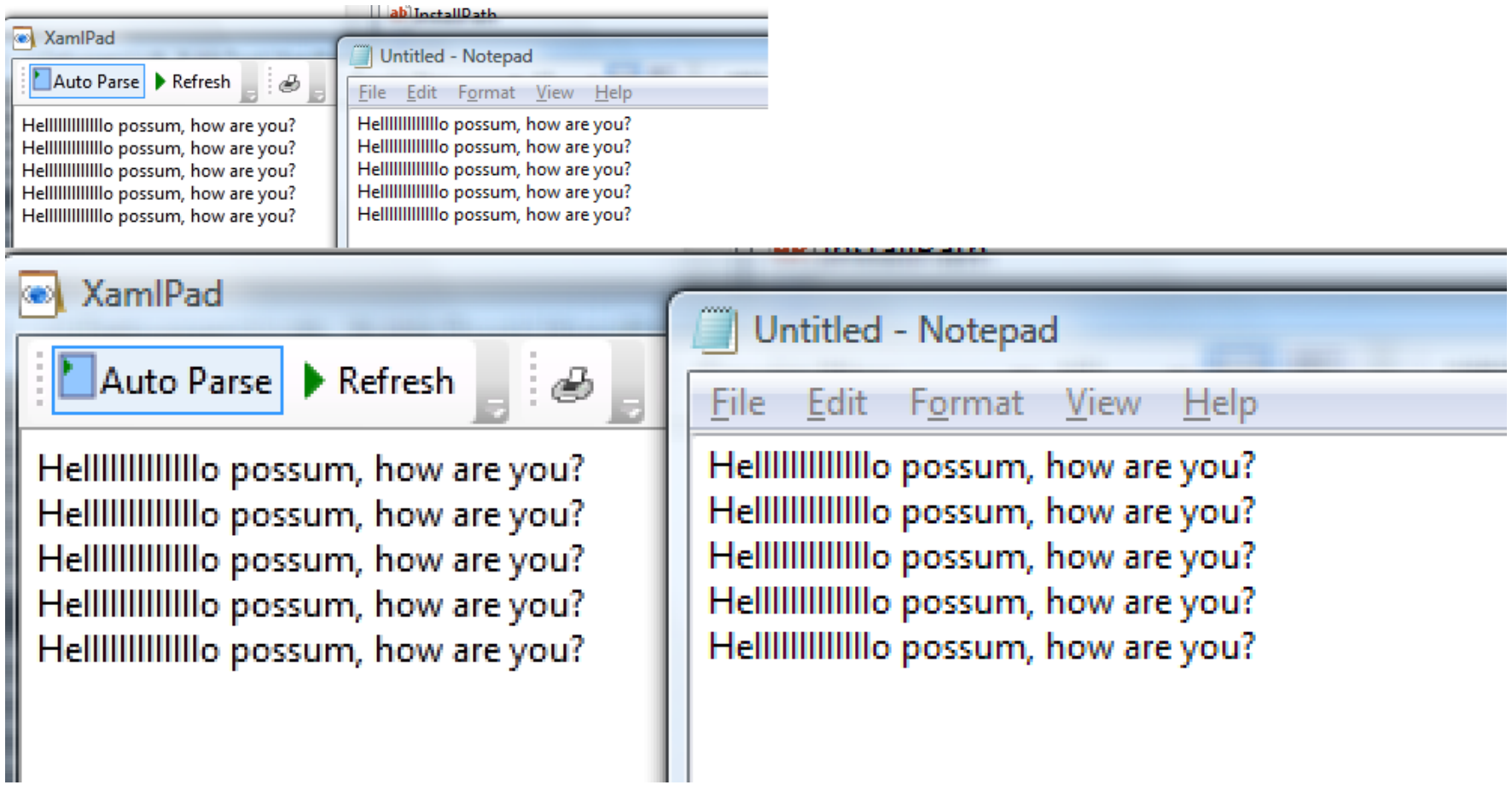
Touch Development Roadmap

NET 4.5 / Surface 2.0 Release

Windows 7/8 Release



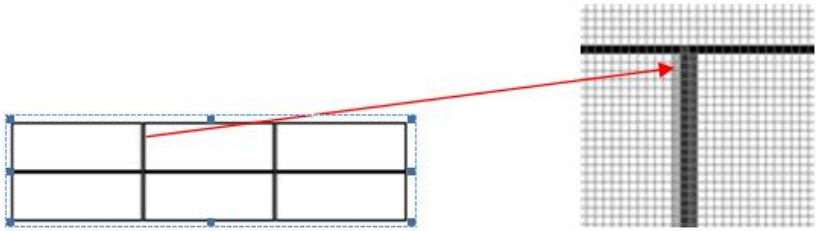
WPF 4.0 <left> versus GDI <right>

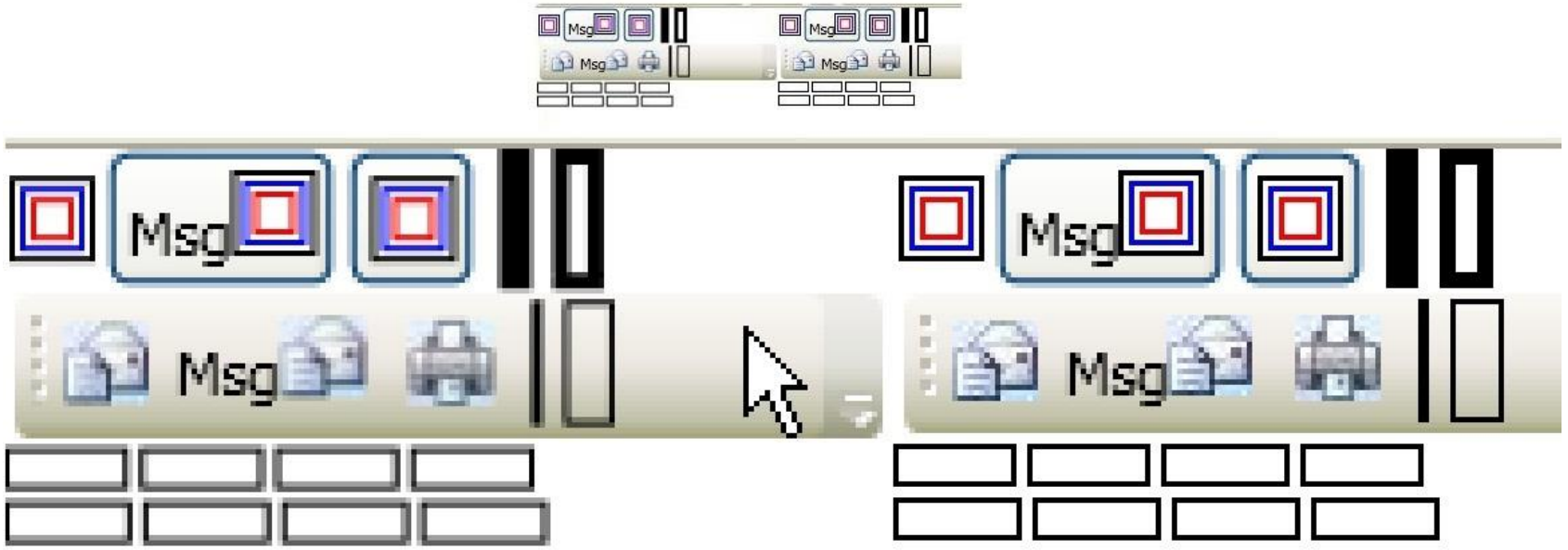


Layout Rounding

- Rounds an object's coordinates on whole pixels

UseLayoutRounding="True"





不同阶段的性能考虑

- 需求阶段：与客户/用户一起设定
- 架构阶段：精炼性能度量指标
- 开发阶段：经常性进行部分功能或者方法接口在单元测试的同时进行性能测试。
- 测试阶段：负载测试、压力测试等验证性能指标。
- 部署及后续阶段：对后续版本持续进行性能测试；对后续应用持续进行性能测试及热点排查。

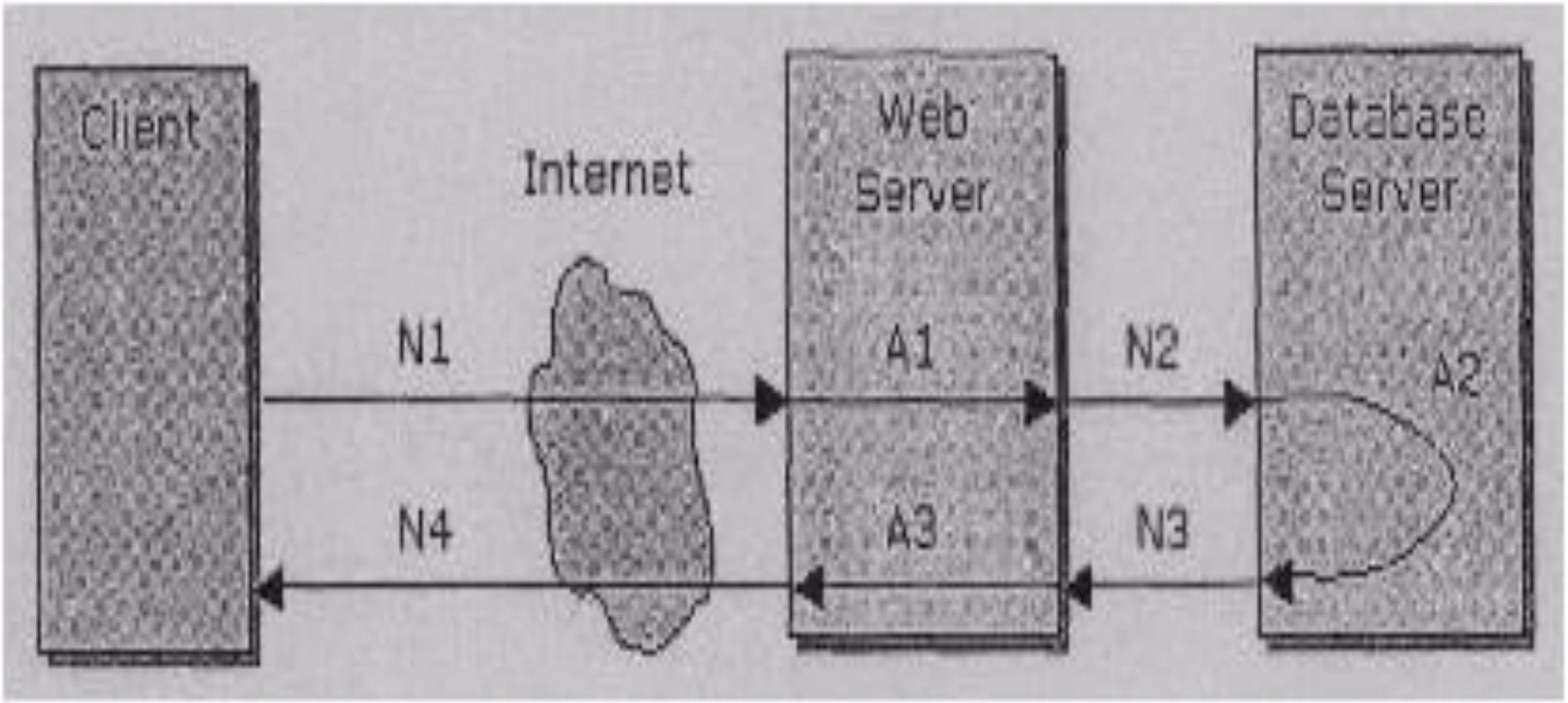
性能目标

目标	回答问题
度量最终用户的响应时间	完成一个业务流程需要多长时间
定义最优的硬件配置	哪一种硬件配置可以提供最佳性能
检查可靠性	系统无错误或无故障运行的时间长度或难度
查看硬件或软件升级	升级对性能或可靠性有何影响
评估新产品	应选择哪些服务器硬件或软件
度量系统容量	在没有显著性能下降的前提下，系统能够处理多大的负载
确定瓶颈	哪些因素会延长响应时间

Performance Metrics

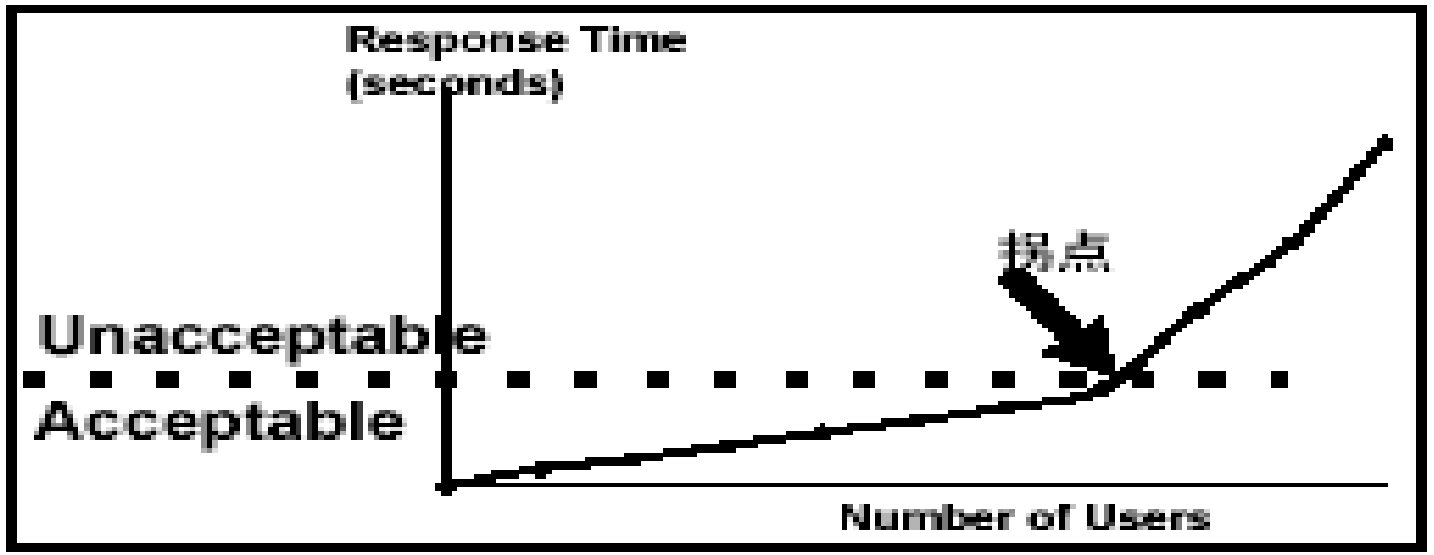
System Type	Performance Goal	Environment Constraints
External Web Server	Time from request start to full response generated should not exceed 300ms	Not more than 300 concurrently active requests
External Web Server	Virtual memory usage (including cache) should not exceed 1.3GB	Not more than 300 concurrently active requests; not more than 5,000 connected user sessions
Application Server	CPU utilization should not exceed 75%	Not more than 1,000 concurrently active API requests
Application Server	Hard page fault rate should not exceed 2 hard page faults per second	Not more than 1,000 concurrently active API requests
Smart Client Application	Time from double-click on desktop shortcut to main screen showing list of employees should not exceed 1,500ms	--
Smart Client Application	CPU utilization when the application is idle should not exceed 1%	--
Web Page	Time for filtering and sorting the grid of incoming emails should not exceed 750ms, including shuffling animation	Not more than 200 incoming emails displayed on a single screen
Web Page	Memory utilization of cached JavaScript objects for the "chat with representative" windows should not exceed 2.5MB	--
Monitoring Service	Time from failure event to alert generated and dispatched should not exceed 25ms	--
Monitoring Service	Disk I/O operation rate when alerts are not actively generated should be 0	--

响应时间细分


Client
Internet
Web Server
Database Server
N1
A1
N2
A2
N4
A3
N3

度量系统容量举例

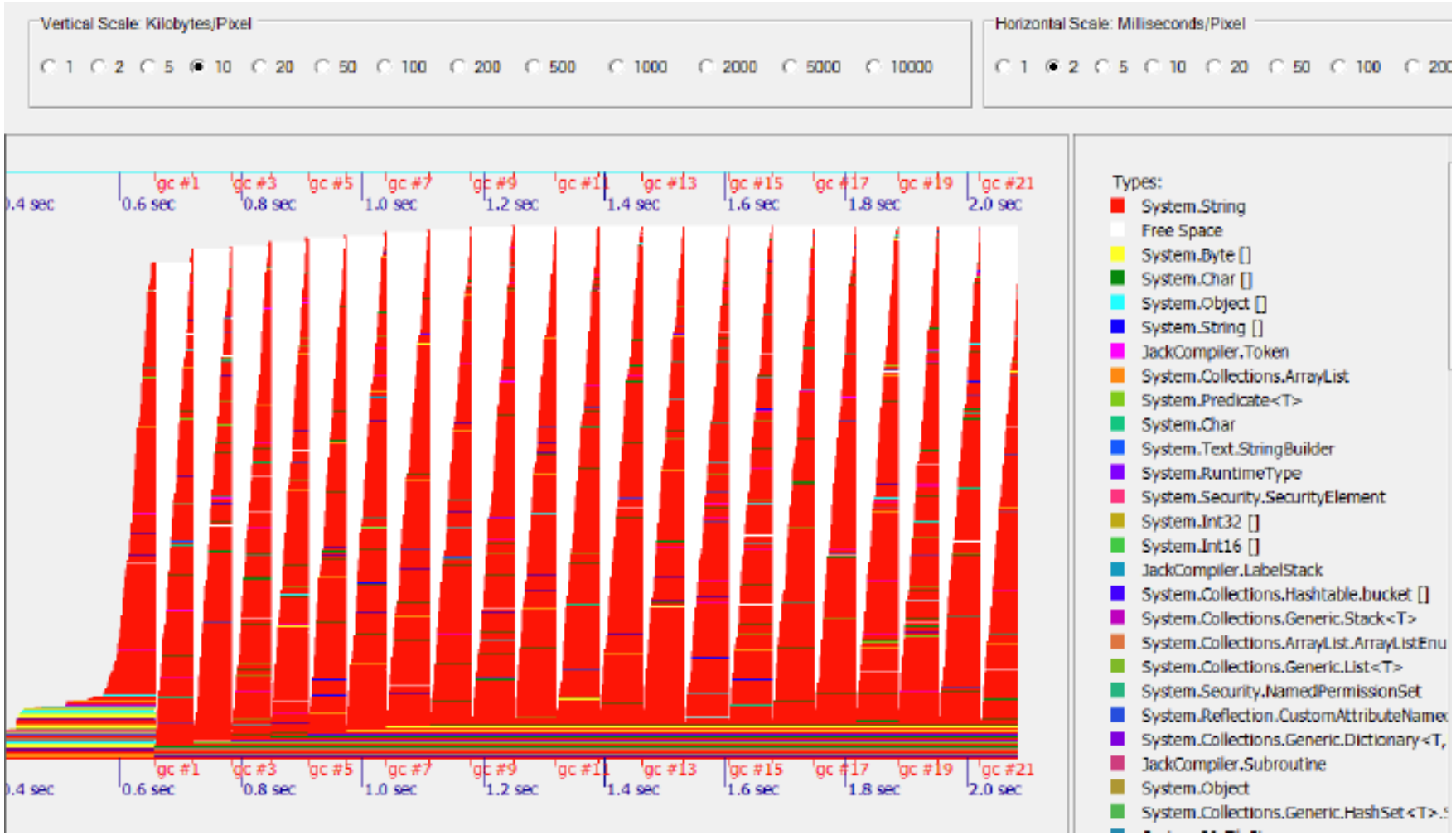
查看现有系统中性能与负载间的关系，并确定出现响应时间显著延长的位置“拐点”。可以确定是否需要增加资源以支持额外的用户。

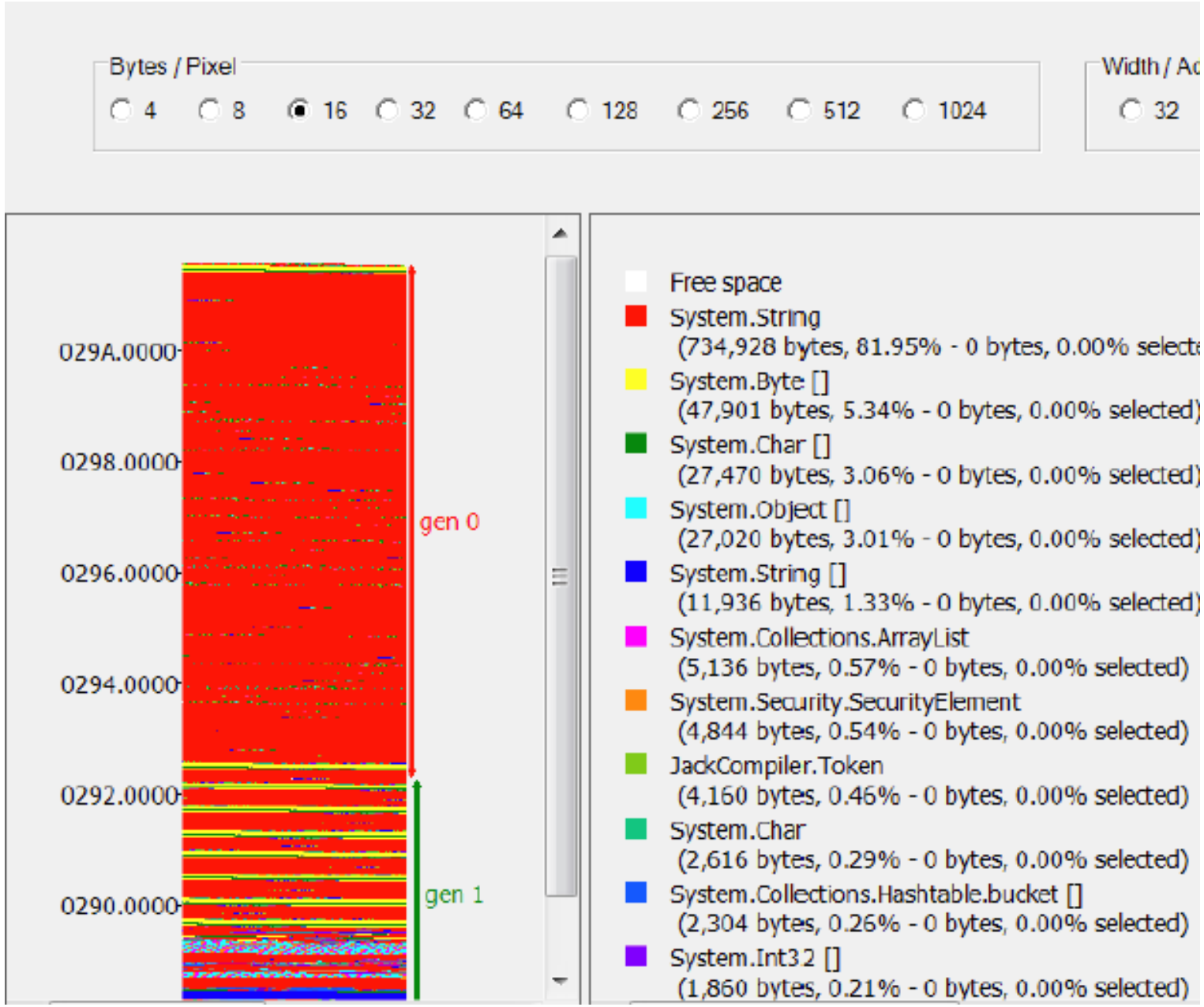


常用性能工具

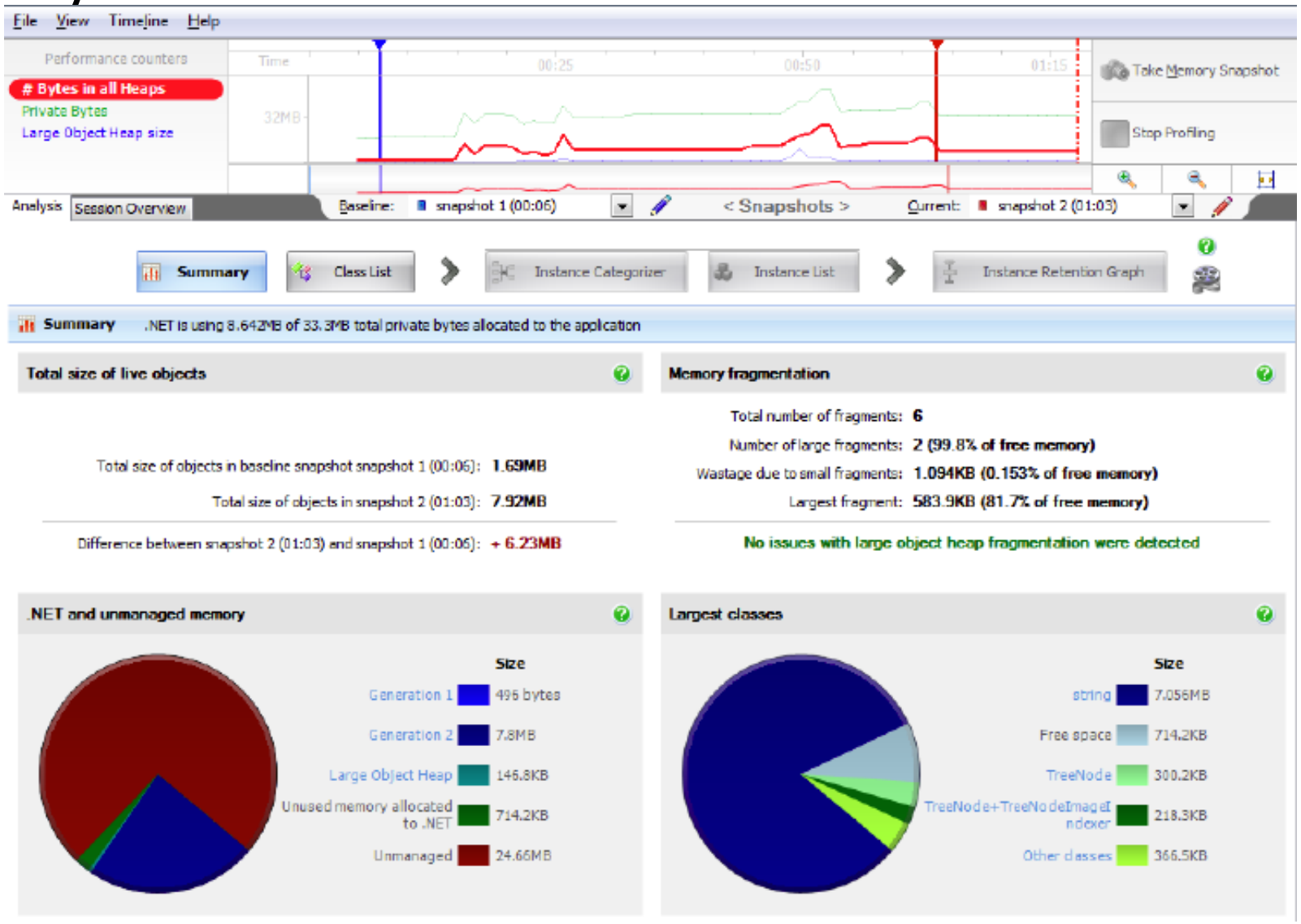
- Loadrunner
- VSTS
- Windows内置Performance Counters
- CLR Profiler
- ANTS Memory Profiler
- WinDBG
- Windows Performance Toolkit(WPT)
- PerfMonitor
- PerfView
- Process Monitor

CLR Profiler

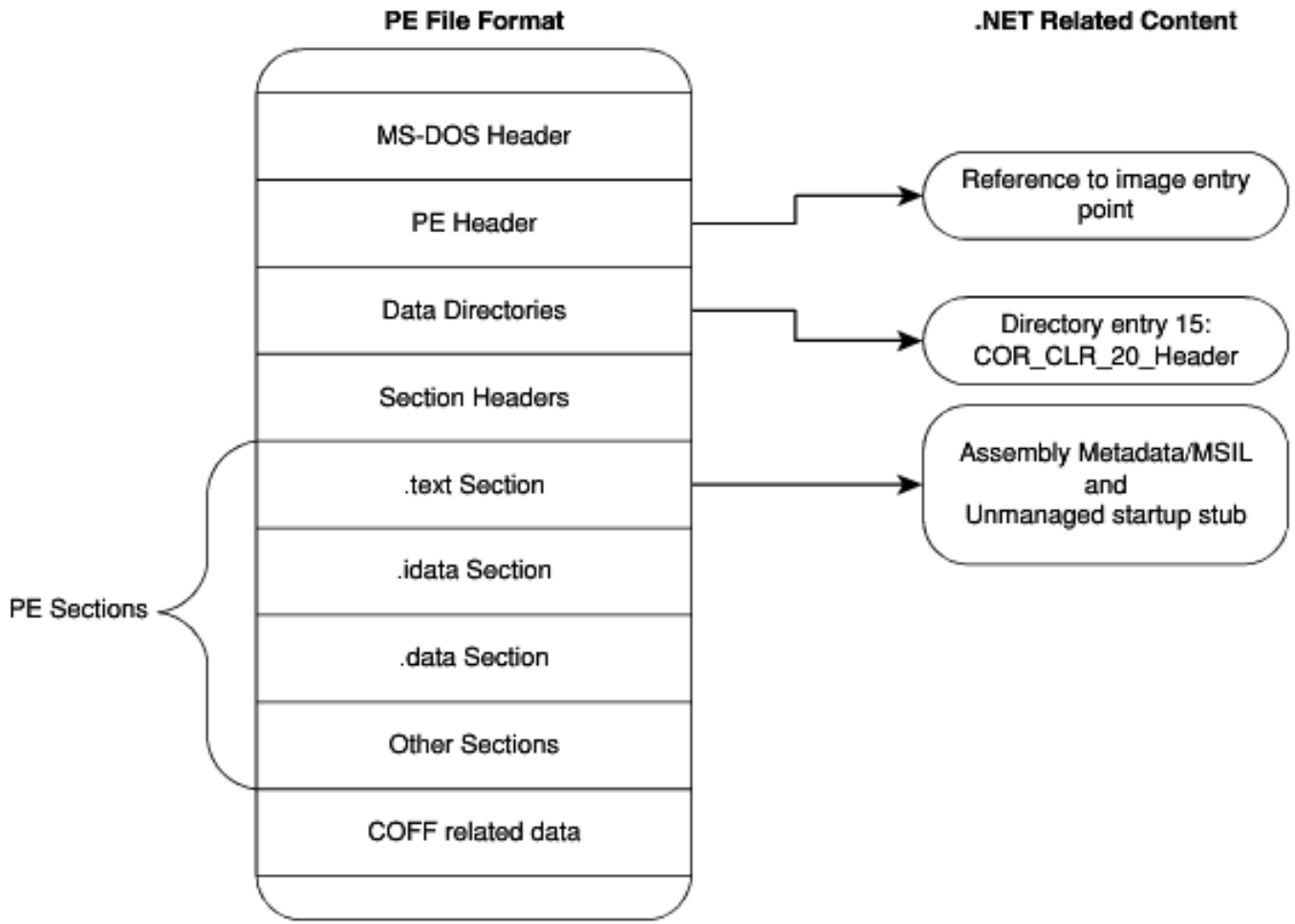



Bytes / Pixel
4
8
16
32
64
128
256
512
1024
Width / Ac
32
029A.0000
0298.0000
0296.0000
0294.0000
0292.0000
0290.0000
gen 0
gen 1
Free space
System.String
(734,928 bytes, 81.95% - 0 bytes, 0.00% selecte
System.Byte []
(47,901 bytes, 5.34% - 0 bytes, 0.00% selected)
System.Char []
(27,470 bytes, 3.06% - 0 bytes, 0.00% selected)
System.Object []
(27,020 bytes, 3.01% - 0 bytes, 0.00% selected)
System.String []
(11,936 bytes, 1.33% - 0 bytes, 0.00% selected)
System.Collections.ArrayList
(5,136 bytes, 0.57% - 0 bytes, 0.00% selected)
System.Security.SecurityElement
(4,844 bytes, 0.54% - 0 bytes, 0.00% selected)
JackCompiler.Token
(4,160 bytes, 0.46% - 0 bytes, 0.00% selected)
System.Char
(2,616 bytes, 0.29% - 0 bytes, 0.00% selected)
System.Collections.Hashtable.bucket []
(2,304 bytes, 0.26% - 0 bytes, 0.00% selected)
System.Int32 []
(1,860 bytes, 0.21% - 0 bytes, 0.00% selected)

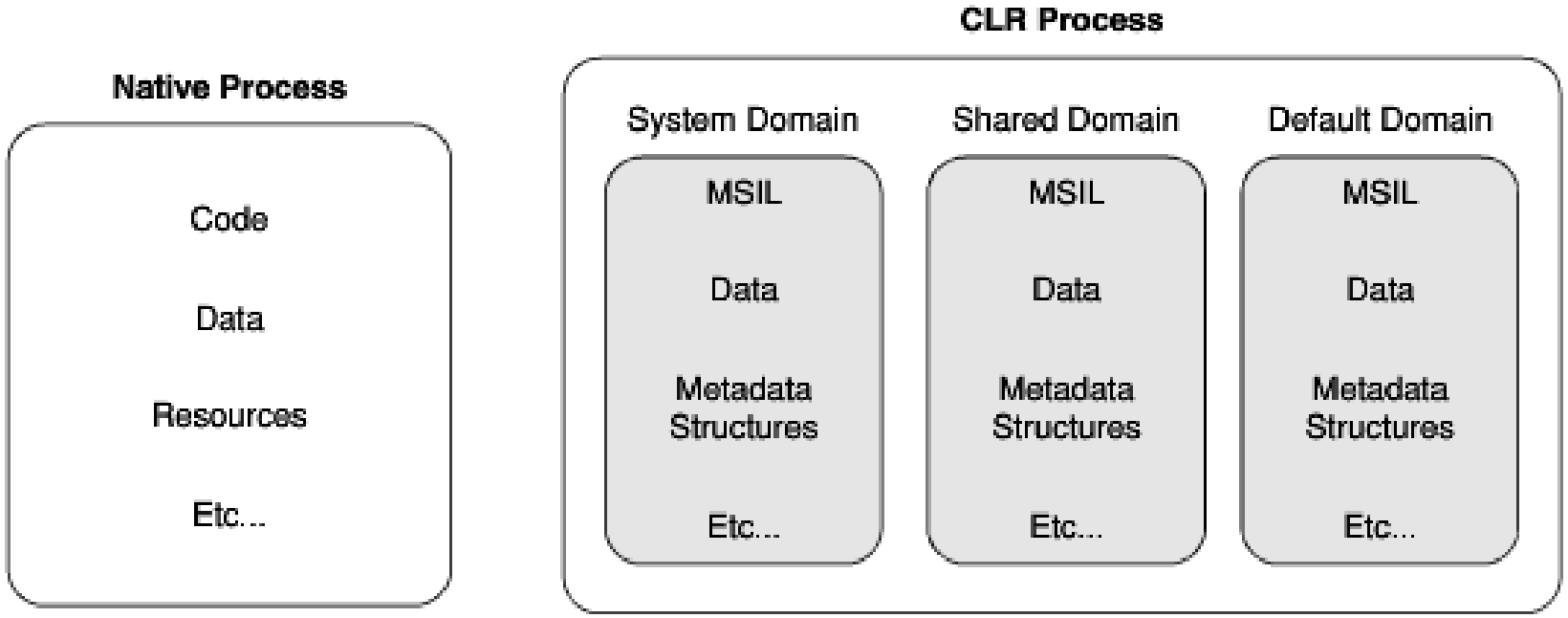
Memory Profiler



PE file结构

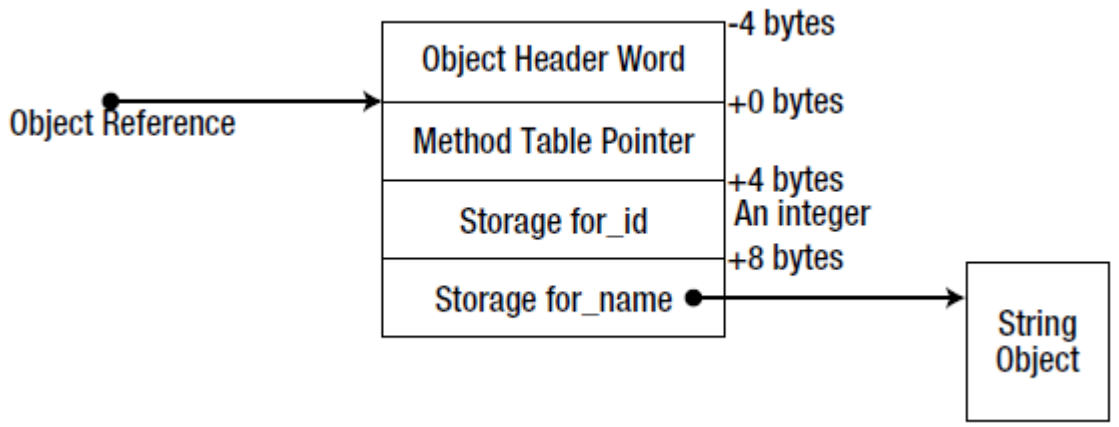

PE File Format
.NET Related Content
MS-DOS Header
PE Header
Data Directories
Section Headers
.text Section
.idata Section
.data Section
Other Sections
COFF related data
PE Sections
Reference to image entry point
Directory entry 15: COR_CLR_20_Header
Assembly Metadata/MSIL and Unmanaged startup stub

.NET Assembly process

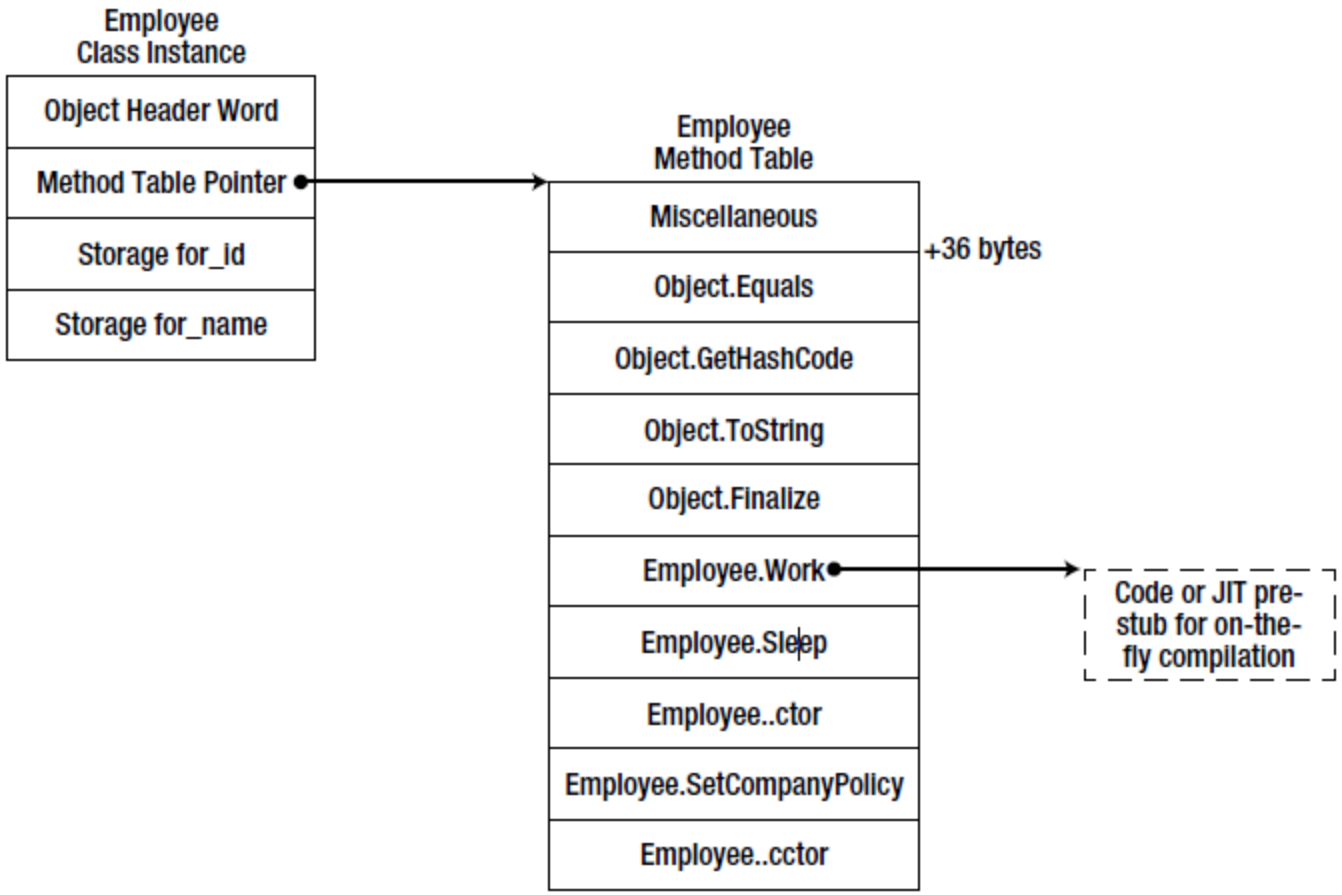


深入对象结构

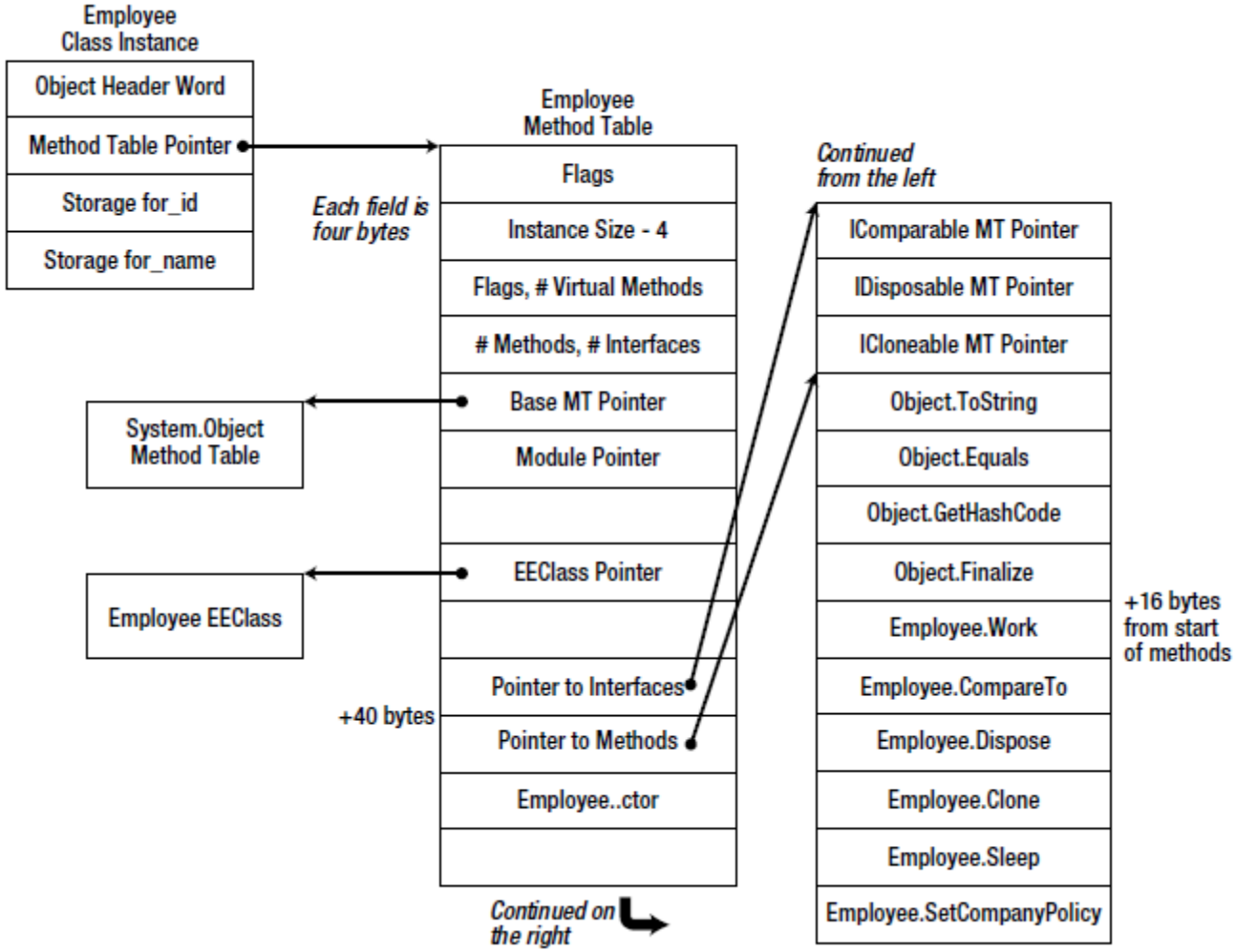
```
public class Employee
{
private int _id;
private string _name;
private static CompanyPolicy _policy;
public virtual void Work() {
Console.WriteLine("Zzzz...");
}
public void TakeVacation(int days) {
Console.WriteLine("Zzzz...");
}
public static void SetCompanyPolicy(CompanyPolicy policy) {
_policy = policy;
}
}
```



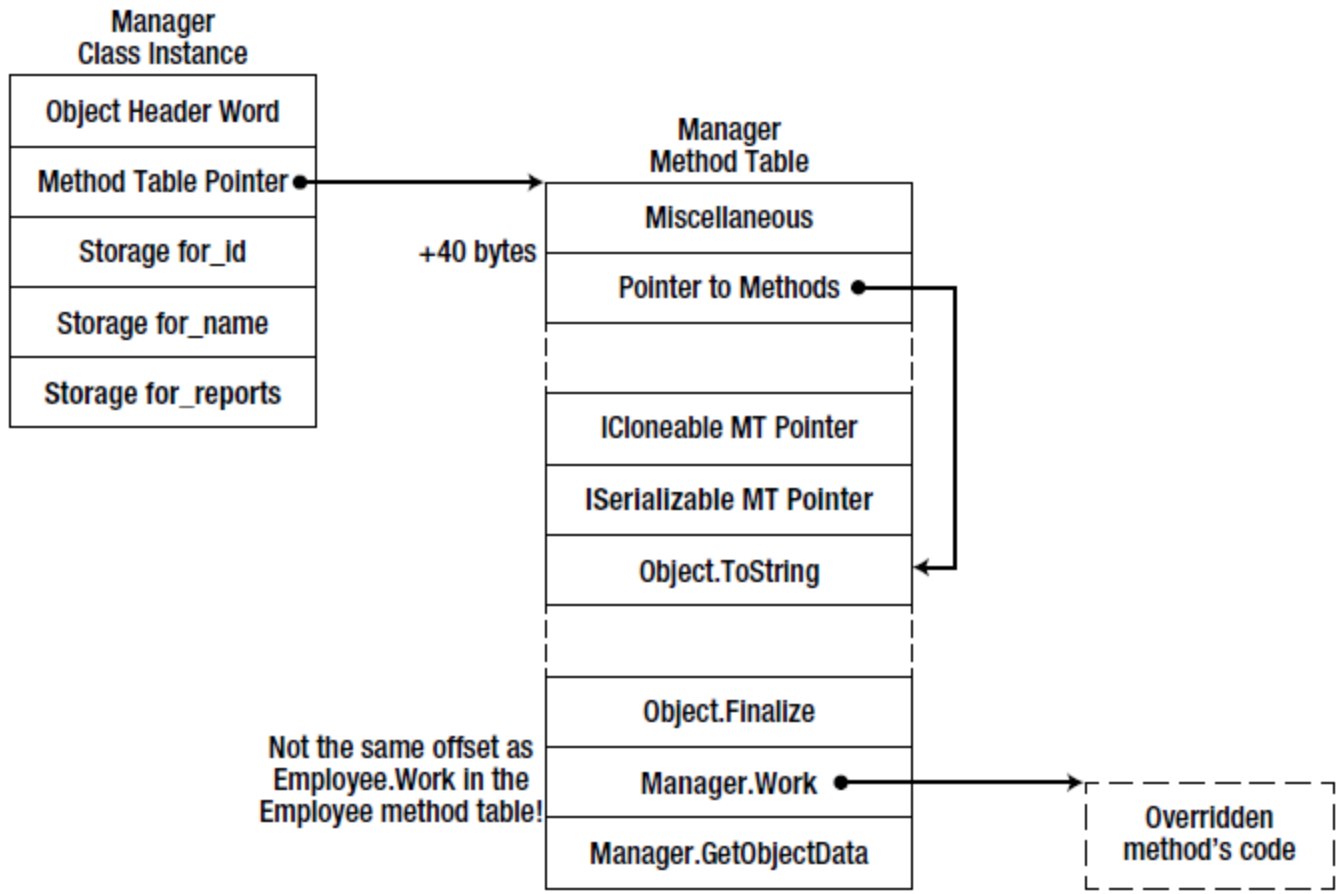
More detail

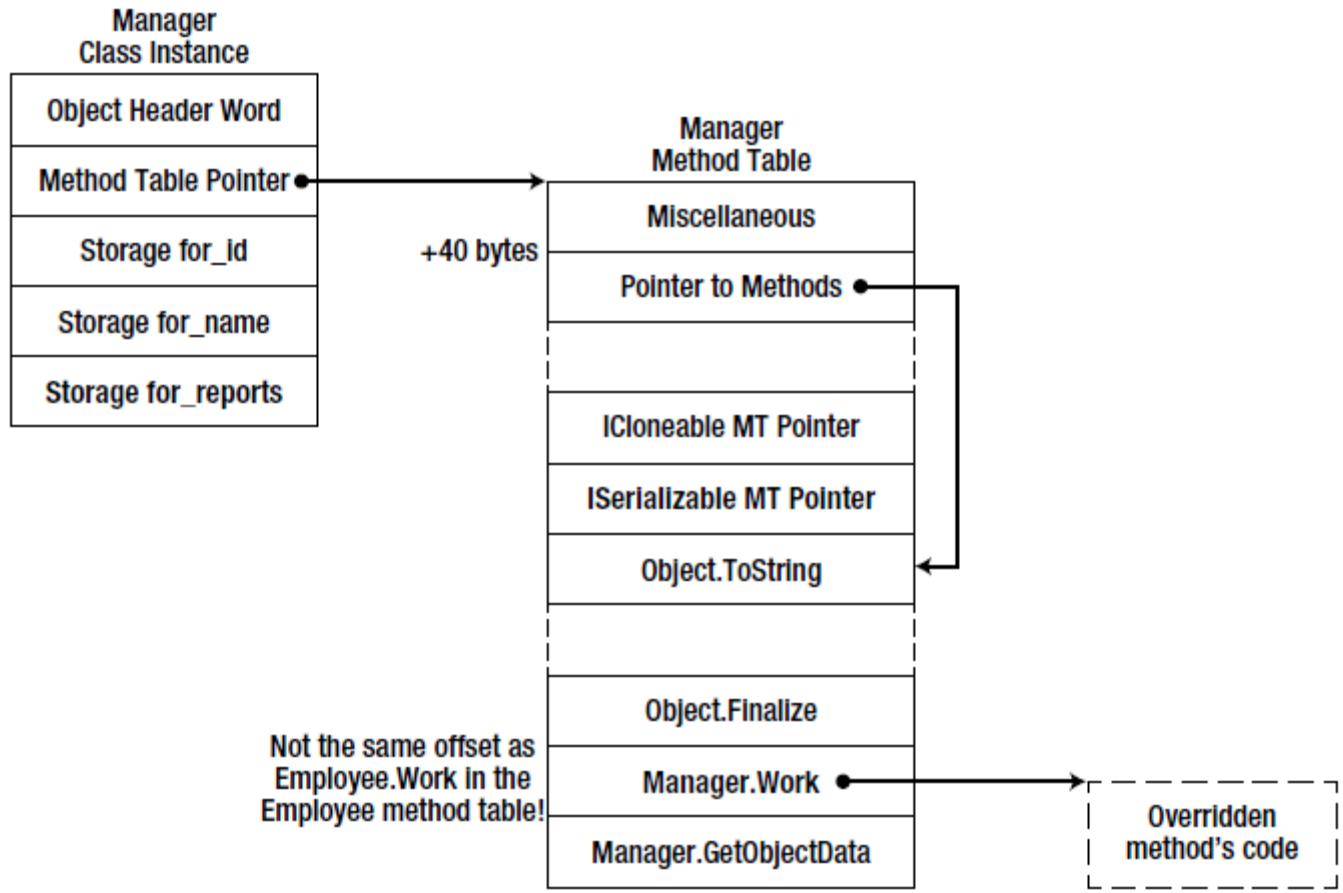

Employee
Class Instance
Object Header Word
Method Table Pointer
Storage for_id
Storage for_name
Employee
Method Table
Miscellaneous
+36 bytes
Object.Equals
Object.GetHashCode
Object.ToString
Object.Finalize
Employee.Work
Employee.Sleep
Employee..ctor
Employee.SetCompanyPolicy
Employee..cctor
Code or JIT pre-stub for on-the-fly compilation

More…

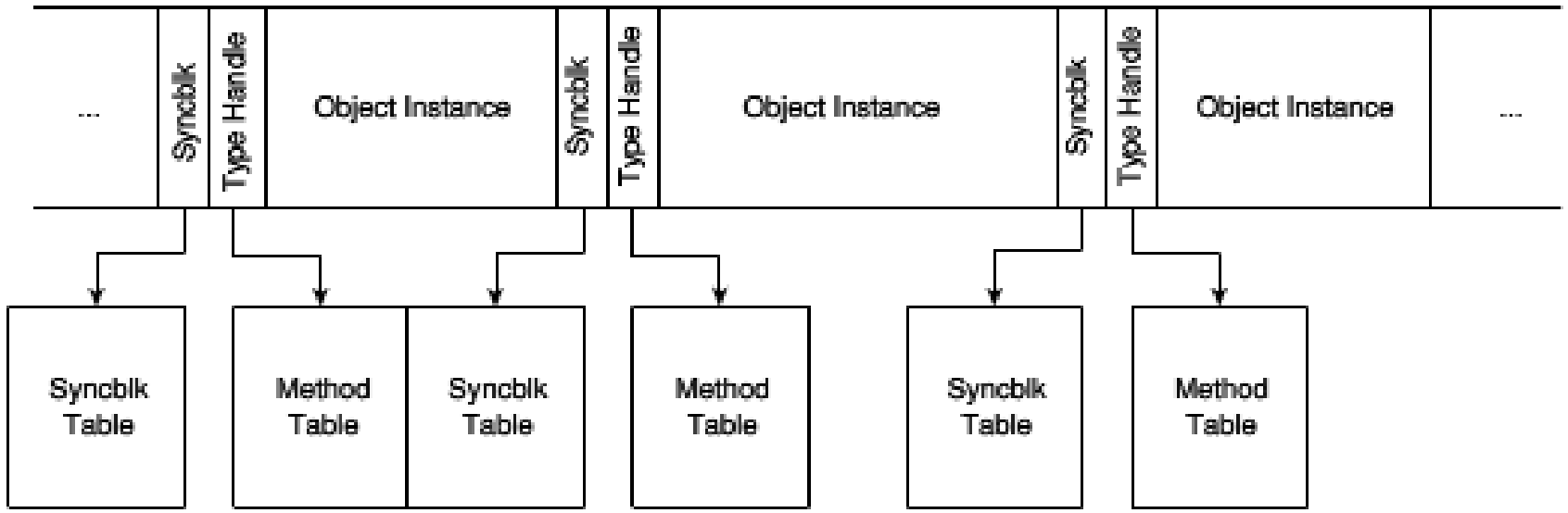

Employee
Class Instance
Object Header Word
Method Table Pointer
Storage for_id
Storage for_name
Employee
Method Table
Each field is four bytes
Flags
Instance Size - 4
Flags, # Virtual Methods
Methods, # Interfaces
Base MT Pointer
Module Pointer
EEClass Pointer
Pointer to Interfaces
Pointer to Methods
Employee..ctor
System.Object
Method Table
Employee EEClass
+40 bytes
Continued on the right
Continued from the left
IComparable MT Pointer
IDisposable MT Pointer
ICloneable MT Pointer
Object.ToString
Object.Equals
Object.GetHashCode
Object.Finalize
Employee.Work
Employee.CompareTo
Employee.Dispose
Employee.Clone
Employee.Sleep
Employee.SetCompanyPolicy
+16 bytes from start of methods

```
public class Manager : Employee, ISerializable
{
private List<Employee> _reports;
public override void Work() ...
//...implementation of ISerializable omitted for
brevity
}
```


Manager
Class Instance
Object Header Word
Method Table Pointer
Storage for_id
Storage for_name
Storage for_reports
Manager
Method Table
+40 bytes
Miscellaneous
Pointer to Methods
ICloneable MT Pointer
ISerializable MT Pointer
Object.ToString
Object.Finalize
Not the same offset as
Employee.Work in the
Employee method table!
Manager.Work
Manager.GetObjectData
Overridden
method's code

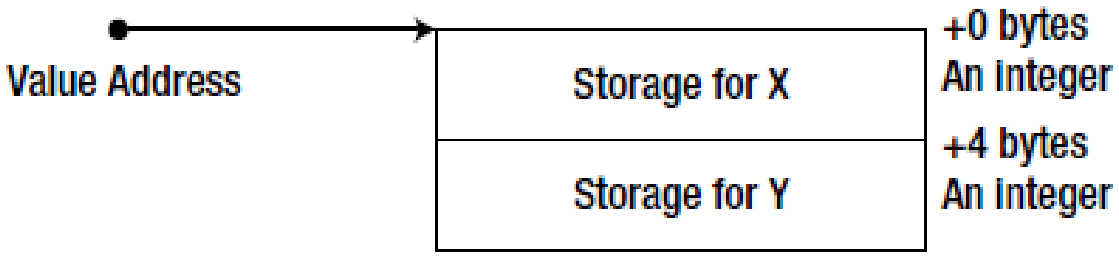

Manager
Class Instance
Object Header Word
Method Table Pointer
Storage for_id
Storage for_name
Storage for_reports
Manager
Method Table
+40 bytes
Miscellaneous
Pointer to Methods
ICloneable MT Pointer
ISerializable MT Pointer
Object.ToString
Object.Finalize
Not the same offset as
Employee.Work in the
Employee method table!
Manager.Work
Manager.GetObjectData
Overridden
method's code

Object in heap



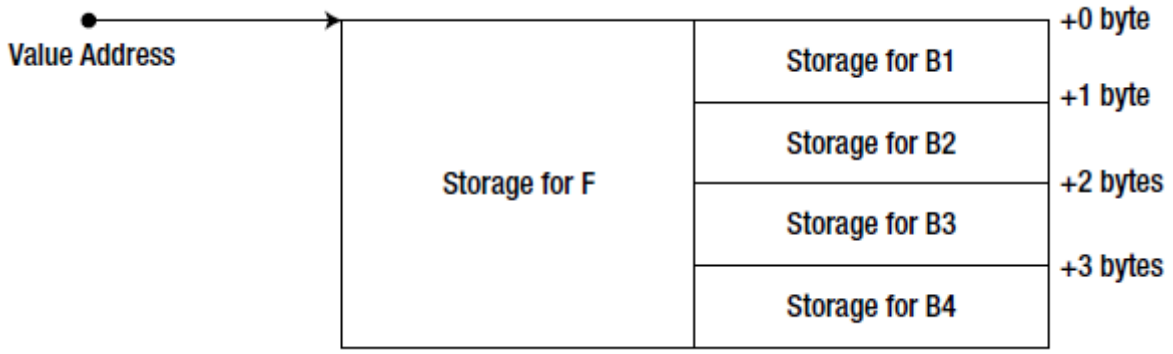
Value type

```
public struct Point2D
{
public int X;
public int Y;
}
```

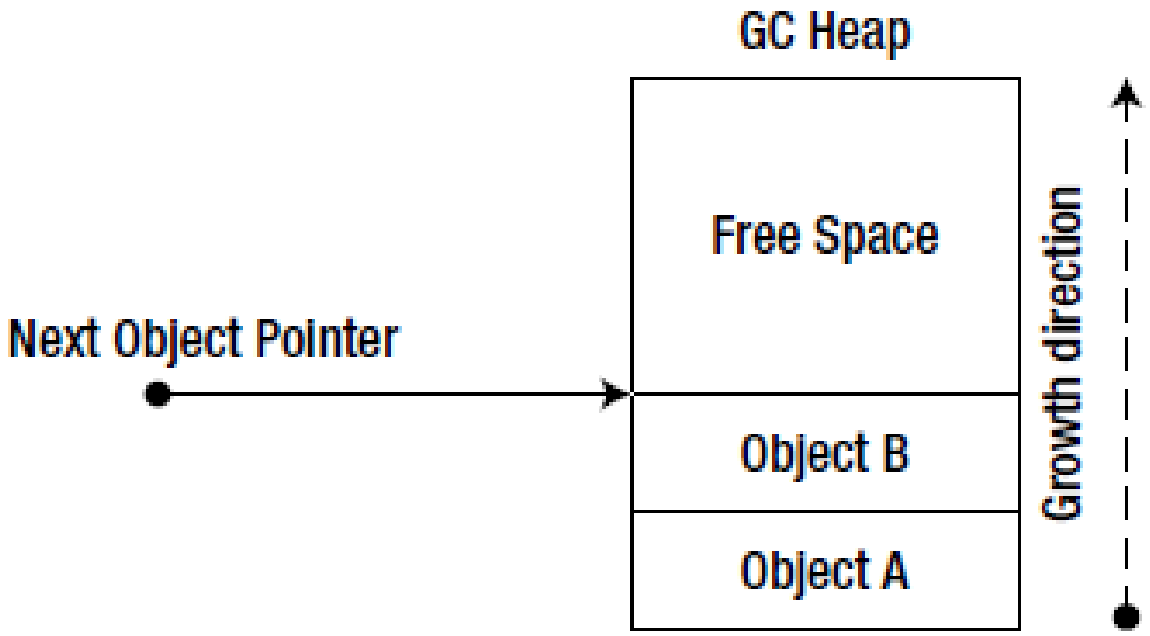


水平分配内存

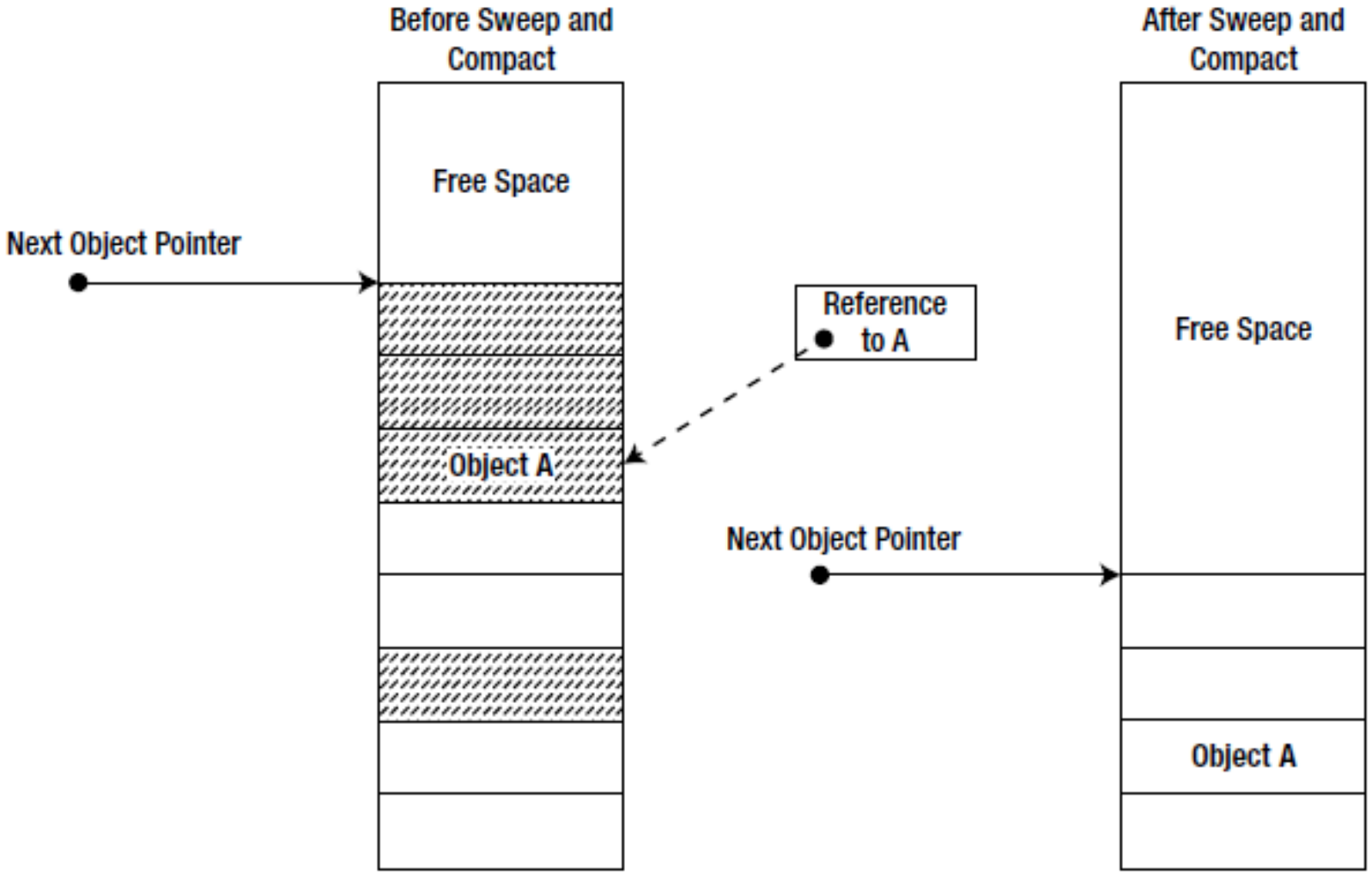
```
[StructLayout(LayoutKind.Explicit)]
public struct FloatingPointExplorer
{
[FieldOffset(0)] public float F;
[FieldOffset(0)] public byte B1;
[FieldOffset(1)] public byte B2;
[FieldOffset(2)] public byte B3;
[FieldOffset(3)] public byte B4;
}
```



GC


GC Heap
Free Space
Next Object Pointer
Object B
Object A
Growth direction

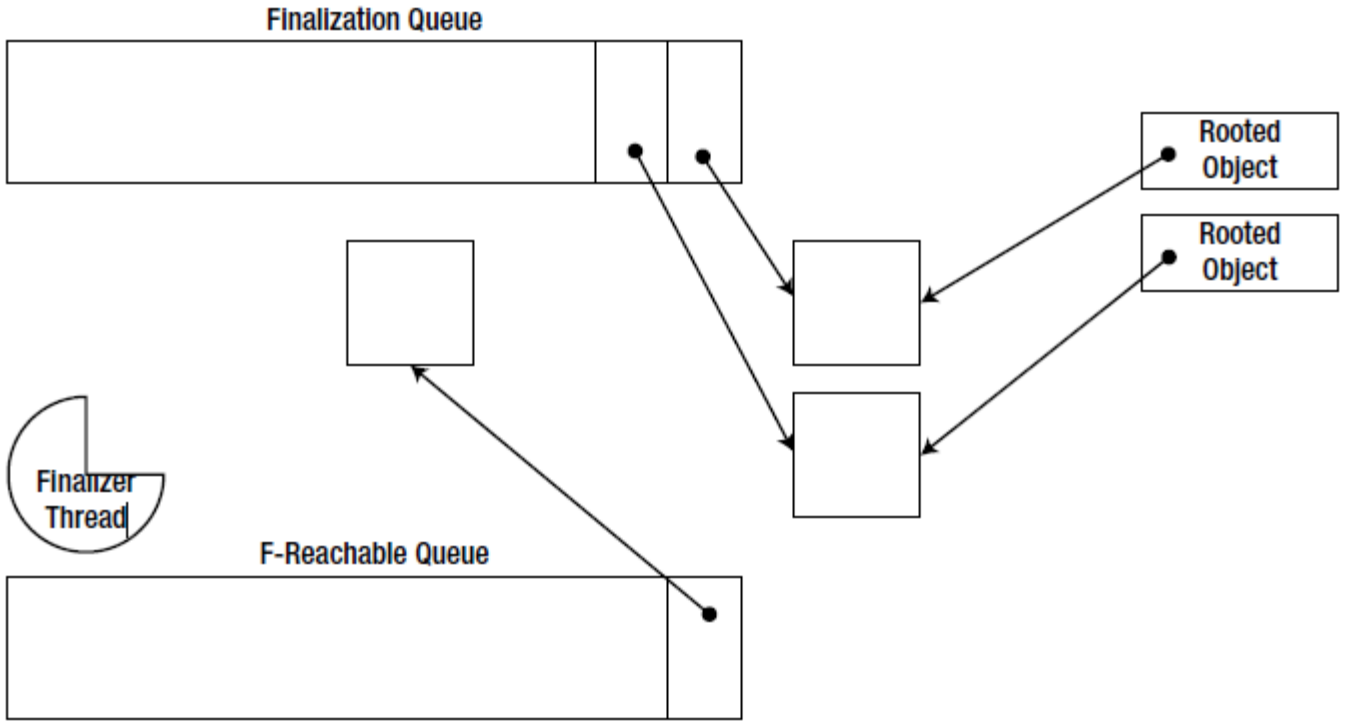
垃圾清理


Before Sweep and Compact
After Sweep and Compact
Free Space
Next Object Pointer
Reference to A
Object A
Free Space
Next Object Pointer
Object A

显示及隐式资源释放

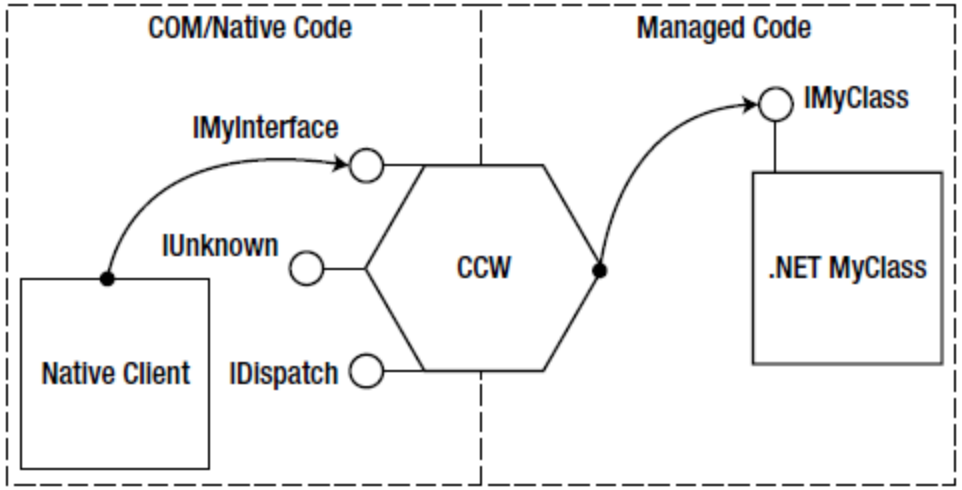
- Dispose
- Finalize

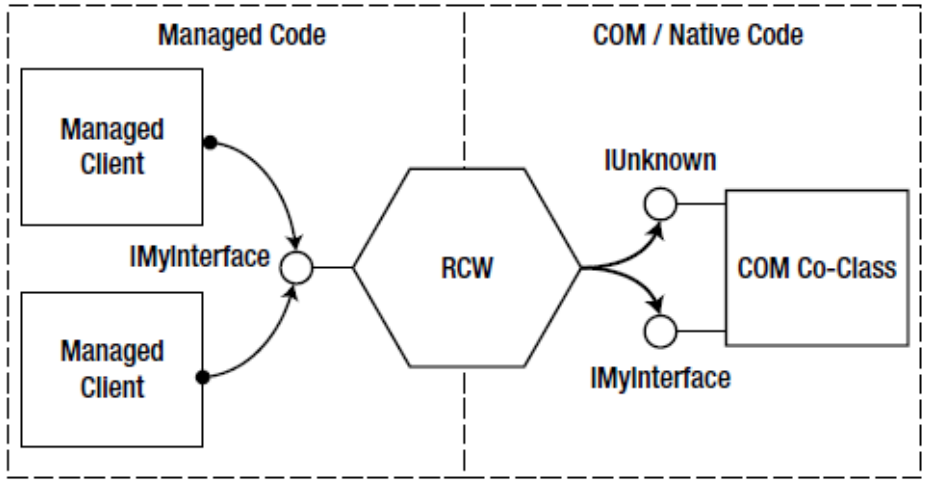
Finalize


Finalization Queue
Rooted Object
Rooted Object
Finalizer Thread
F-Reachable Queue

托管与非托管交互的问题

- COM调用
- P/Invoke





解决办法

1. Marshal类的ReleaseComObject方法：
   - 该运行库可调用包装具有引用计数，每次将 COM 接口指针映射到该运行库可调用包装时，此引用计数都将递增。ReleaseComObject 方法递减运行库可调用包装的引用计数。当引用计数达到零时，运行库将释放非托管 COM 对象上的所有引用，并在您试图进一步使用该对象时引发 System.NullReferenceException。如果从非托管代码向托管代码传递同一 COM 接口的次数超过一次，则包装上的引用计数将依次递增，而且调用 ReleaseComObject 将返回剩余引用的数目。

解决办法

2.使用句柄统计（HandleCollector）

```
class ExpensiveHandles
 {
static readonly HandleCollector myExpensiveHandleCollector =
                new HandleCollector("ExpensiveHandles", 2, 5);
   public ExpensiveHandles()
   {
     // Get an expensive handle or resource here (code omitted)
     myExpensiveHandleCollector.Add();
   }
   ~ExpensiveHandles()
   {
     // Release expensive handle here (code omitted)
     myExpensiveHandleCollector.Remove();
   }
 }
```

解决办法

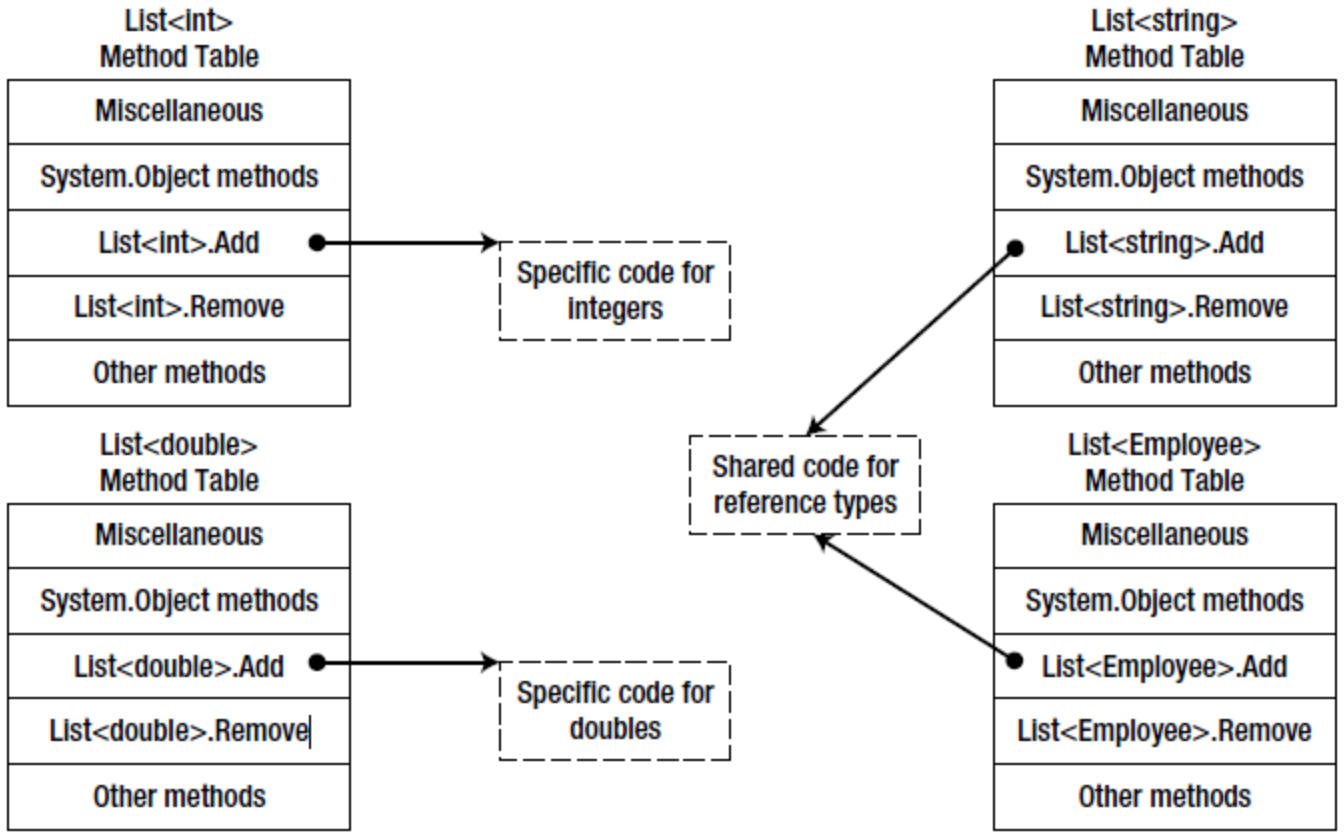
3.增加受管内存压力

```
class PressureClass
 {
   private long _pressure;

   public PressureClass(long pressureAmount)
   {
     _pressure = pressureAmount;
     GC.AddMemoryPressure(_pressure);
   }

   ~PressureClass()
   {
     GC.RemoveMemoryPressure(_pressure);
   }
 }
```

集合与泛型


List<int>
Method Table
Miscellaneous
System.Object methods
List<int>.Add
List<int>.Remove
Other methods
Specific code for integers
List<string>
Method Table
Miscellaneous
System.Object methods
List<string>.Add
List<string>.Remove
Other methods
Shared code for reference types
List<double>
Method Table
Miscellaneous
System.Object methods
List<double>.Add
List<double>.Remove
Other methods
Specific code for doubles
List<Employee>
Method Table
Miscellaneous
System.Object methods
List<Employee>.Add
List<Employee>.Remove
Other methods

各种集合实现细节

Collection	Details	Insertion Time	Deletion Time	Lookup Time	Sorted	Index Access
`List<T>`	Automatically resizable array	Amortized $O(1)$*	$O(n)$	$O(n)$	No	Yes
`LinkedList<T>`	Doubly-linked list	$O(1)$	$O(1)$	$O(n)$	No	No
`Dictionary<K,V>`	Hash table	$O(1)$	$O(1)$	$O(1)$	No	No
`HashSet<T>`	Hash table	$O(1)$	$O(1)$	$O(1)$	No	No
`Queue<T>`	Automatically resizable cyclic array	Amortized $O(1)$	$O(1)$	--	No	No
`Stack<T>`	Automatically resizable array	Amortized $O(1)$	$O(1)$	--	No	No
`SortedDictionary<K,V>`	Red-black tree	$O(\log n)$	$O(\log n)$	$O(\log n)$	Yes (keys)	No
`SortedList<K,V>`	Sorted resizable array	$O(n)$**	$O(n)$	$O(\log n)$	Yes (keys)	Yes
`SortedSet<T>`	Red-black tree	$O(\log n)$	$O(\log n)$	$O(\log n)$	Yes	No

Demo： 算法优化

Unit3：高级应用之实践

模式的起源

- 来源于建筑学和人类学的一个概念
  - 在文化人类学中，超越个人信仰和文化差异对美感的评价是否具有一致性？
  - 是否存在一个描述我们共同认知的基础？
  - 质量是客观的吗？
  - 是什么因素让我们认为一个建筑（软件）设计是好的设计？

什么是模式？

•建筑师Christopher Alexander上世纪70年代《The Timeless Way of Building》和《A Pattern Language》。

Patterns are solutions to a problem in a context.

什么是模式？

Each pattern describes a problem which occurs over and over again in our environment and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

—— Christopher Alexander

每个模式描述了一个在我们周围不断重复发生的问题，以及该问题的解决方案的核心。这样，你就能一次又一次地使用该方案而不必做重复劳动。

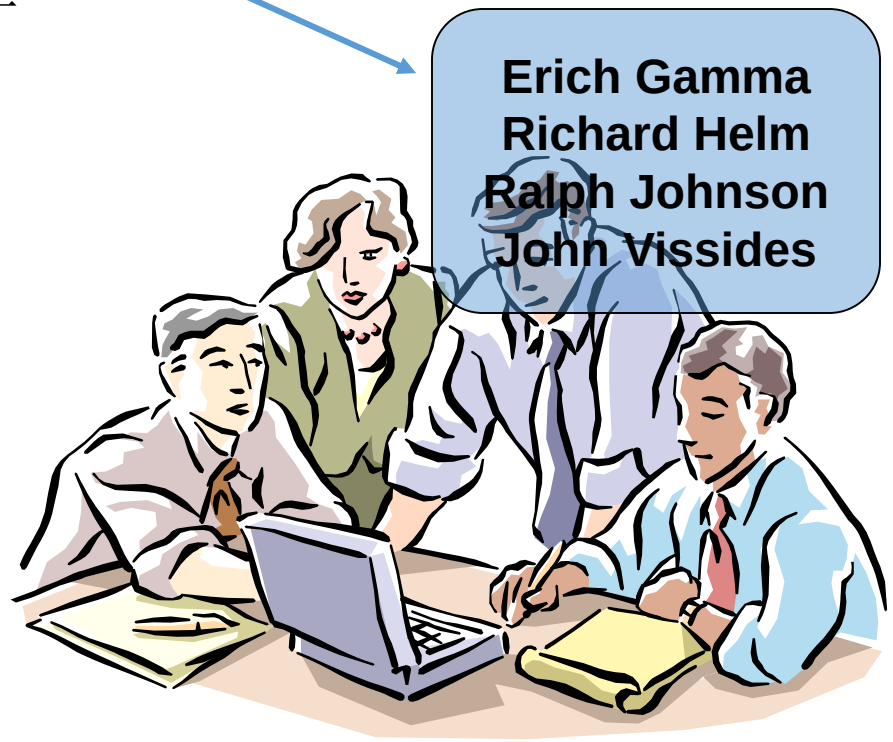
—— Christopher Alexander

模式在软件开发流程中的应用

- 需求模式
- 架构模式
- 设计模式

设计模式

- 90年代初在软件设计中开始引入并流行。
- 四人帮（Gang of Four, GoF）的《Design Patterns: Elements of Reusable Object-Oriented Software》, 1995年
  - 将设计模式的概念应用于软件设计
  - 基于设计模式的OO策略和方法。
  - 对设计模式进行编目和描述
  - 收录了23种设计模式



为什么需要设计模式

- 重用解决方案
  - 减少重复设计和编码
  - 规范设计和编码，便于修改和升级
  - 系统具有更好适应性和健壮性
- 建立通用的术语学
  - 在团队内建立对问题的通用词汇和观点
  - 便于设计阶段参考引用
  - 帮助程序员学习和团队开发
- 对软件设计的哲学思考

设计模式的缺点

- 对象过多：
  - 为了实现更好的灵活性，大部分设计模式都需要把可变内容封装到对象内，这样在增加了系统的灵活性和可维护性的同时，引入了大量的对象，比如命令模式。
- 关系复杂
  - 设计模式往往依赖于对象之间的关系，因此引入设计模式就意味着关系更为复杂。
- 测试更难
  - 引入更多的对象和更为复杂的关系后，测试用例编写更为复杂。

设计模式的核心

- 继承
  - 接口
  - 抽象类
- 多态
  - 虚拟属性/方法
  - 接口属性/方法
  - 抽象属性/方法
- .NET特异性
  - 反射

Design Pattern

- **Creational Patterns**
- **Structural Patterns**
- **Behavioral Patterns**

Creational Patterns

- **Abstract Factory（抽象工厂）** Creates an instance of several families of classes
- **Builder（生成器）** Separates object construction from its representation
- **Factory Method（工厂方法）** Creates an instance of several derived classes
- **Prototype（原型）** A fully initialized instance to be copied or cloned
- **Singleton（单件）** A class of which only a single instance can exist

Structural Patterns

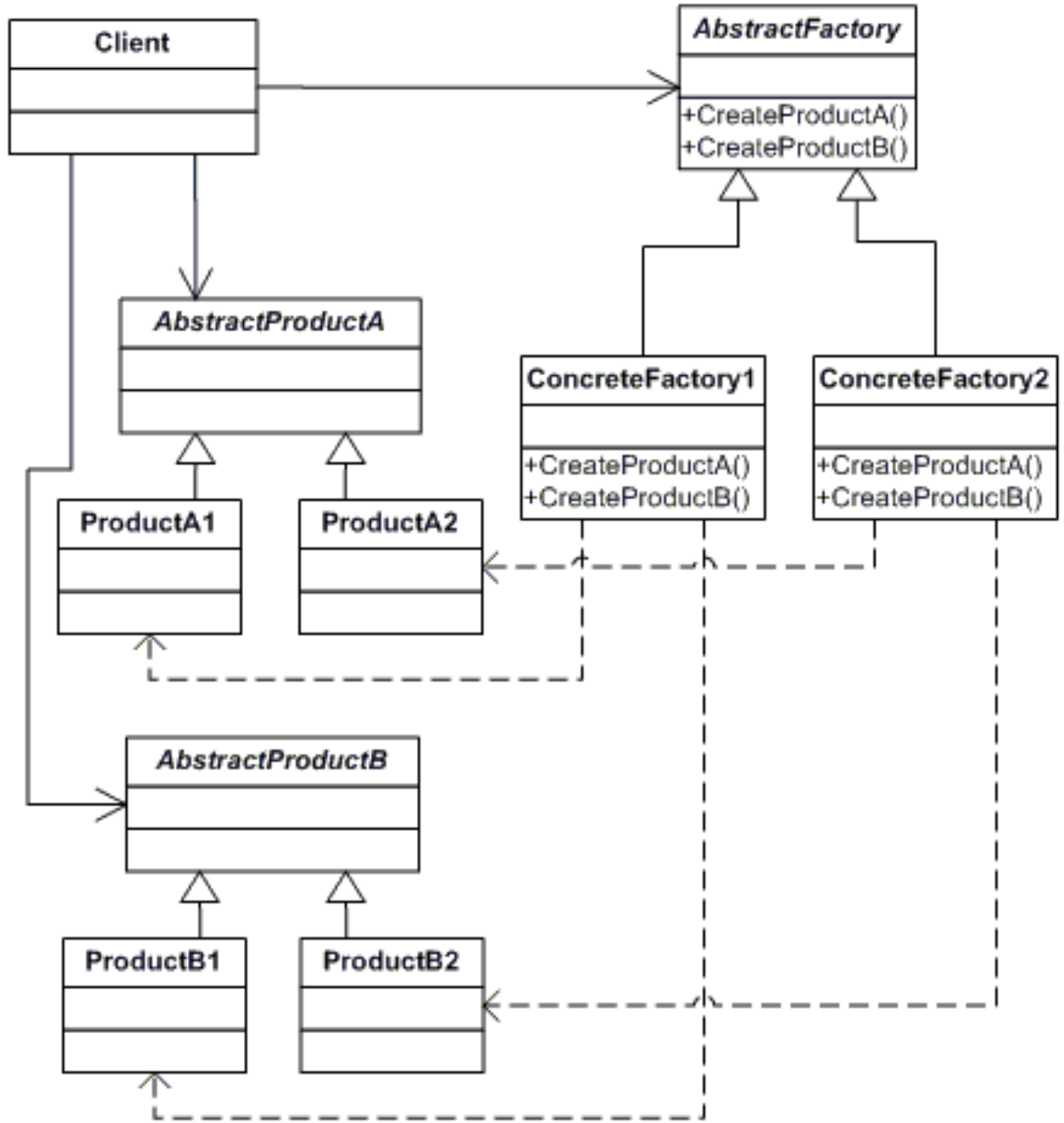
- **Adapter（适配器）** Match interfaces of different classes
- **Bridge（桥接）** Separates an object's interface from its implementation
- **Composite (组合)** A tree structure of simple and composite objects
- **Decorator （装饰）** Add responsibilities to objects dynamically
- **Façade （外观）** A single class that represents an entire subsystem
- **Flyweight （享元）** A fine-grained instance used for efficient sharing
- **Proxy （代理）** An object representing another object

Behavioral Patterns

- **Chain of Resp.**（职责链） A way of passing a request between a chain of objects
- **Command**（命令） Encapsulate a command request as an object
- **Interpreter**（解释器）A way to include language elements in a program
- **Iterator**（迭代器） Sequentially access the elements of a collection
- **Mediator**（中介者）Defines simplified communication between classes
- **Memento**（备忘录） Capture and restore an object's internal state
- **Observer**（观察者） A way of notifying change to a number of classes
- **State**（状态） Alter an object's behavior when its state changes
- **Strategy**（策略） Encapsulates an algorithm inside a class
- **Template Method**（模板方法）Defer the exact steps of an algorithm to a subclass
- **Visitor**（访问者） Defines a new operation to a class without change

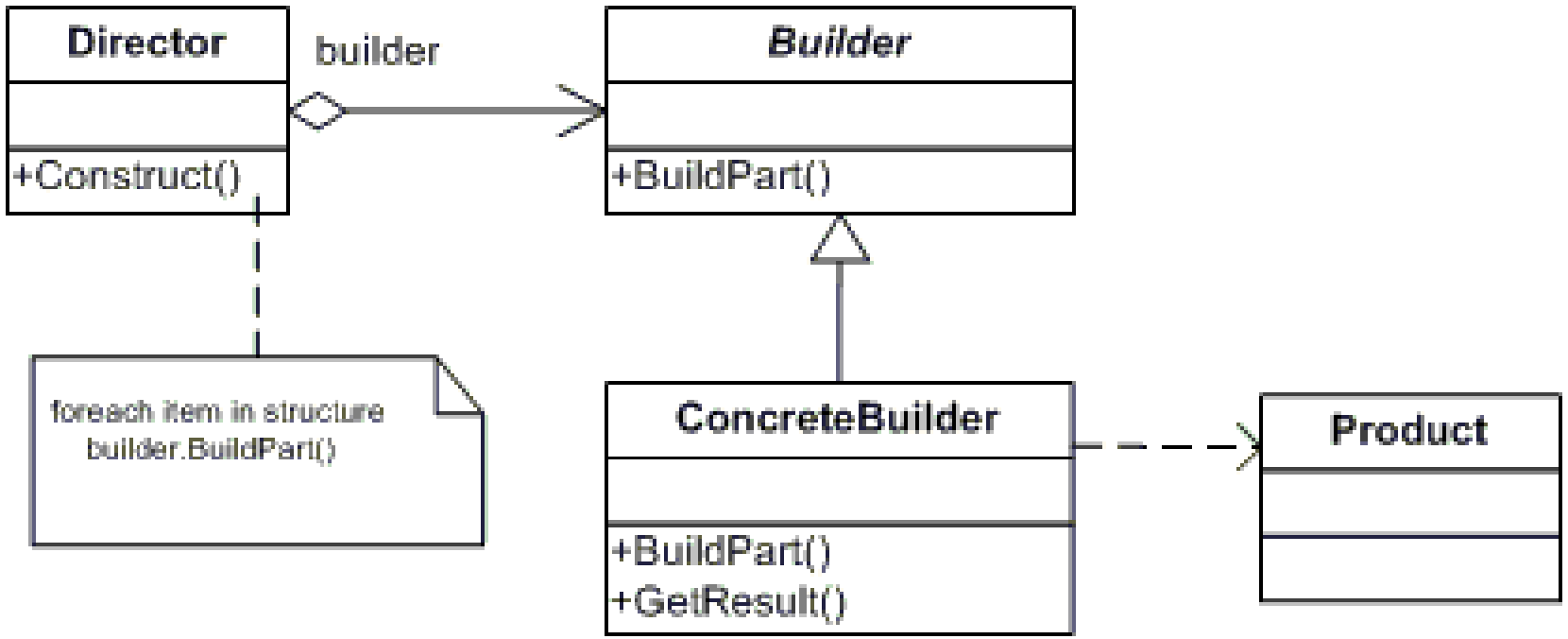
Abstract Factory

- Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
- 使用场合：
  - 系统要创建独立于其产品的创建、组合和表示
  - 系统要由多个产品系列中的一个来配对时
  - 需要提供一个产品类库，其中需要隐藏其实现，只想显示其接口。


Client
AbstractFactory
+CreateProductA()
+CreateProductB()
AbstractProductA
ConcreteFactory1
+CreateProductA()
+CreateProductB()
ConcreteFactory2
+CreateProductA()
+CreateProductB()
ProductA1
ProductA2
AbstractProductB
ProductB1
ProductB2

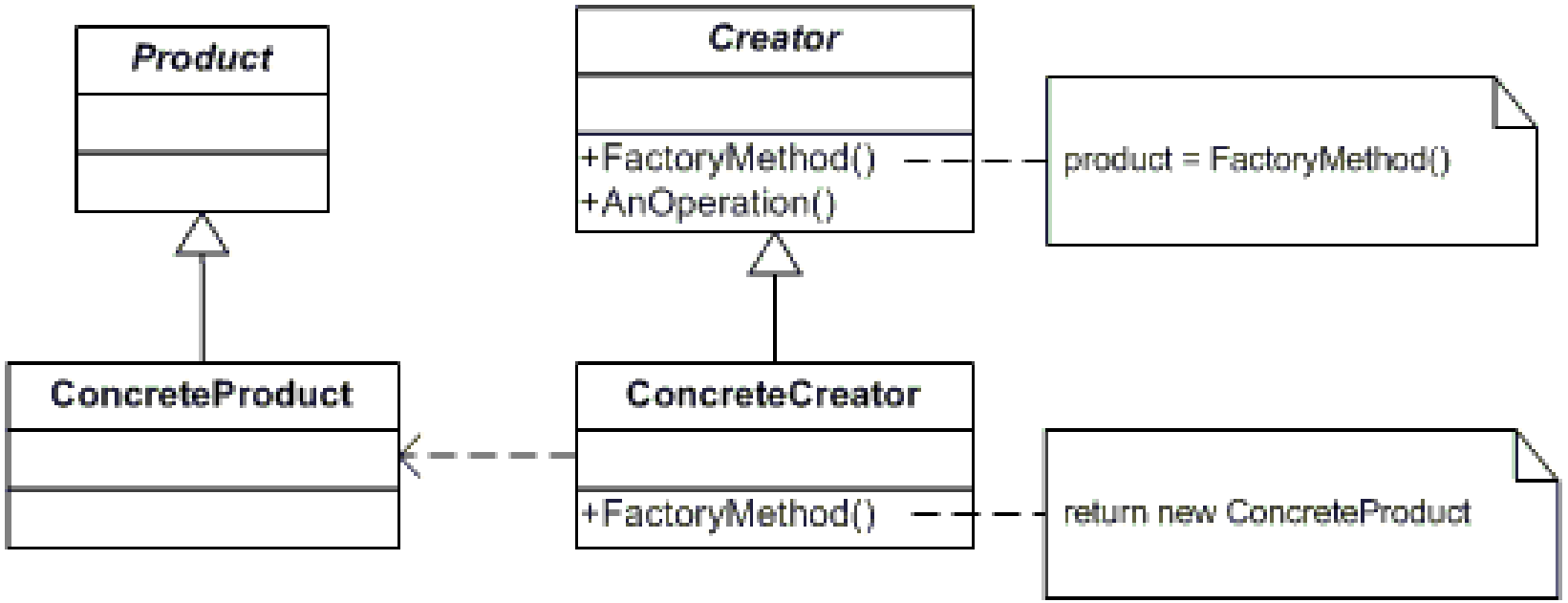
Builder

- Separate the construction of a complex object from its representation so that the same construction process can create different representations.


Director
builder
Builder
+Construct()
+BuildPart()
foreach item in structure
builder.BuildPart()
ConcreteBuilder
Product
+BuildPart()
+GetResult()

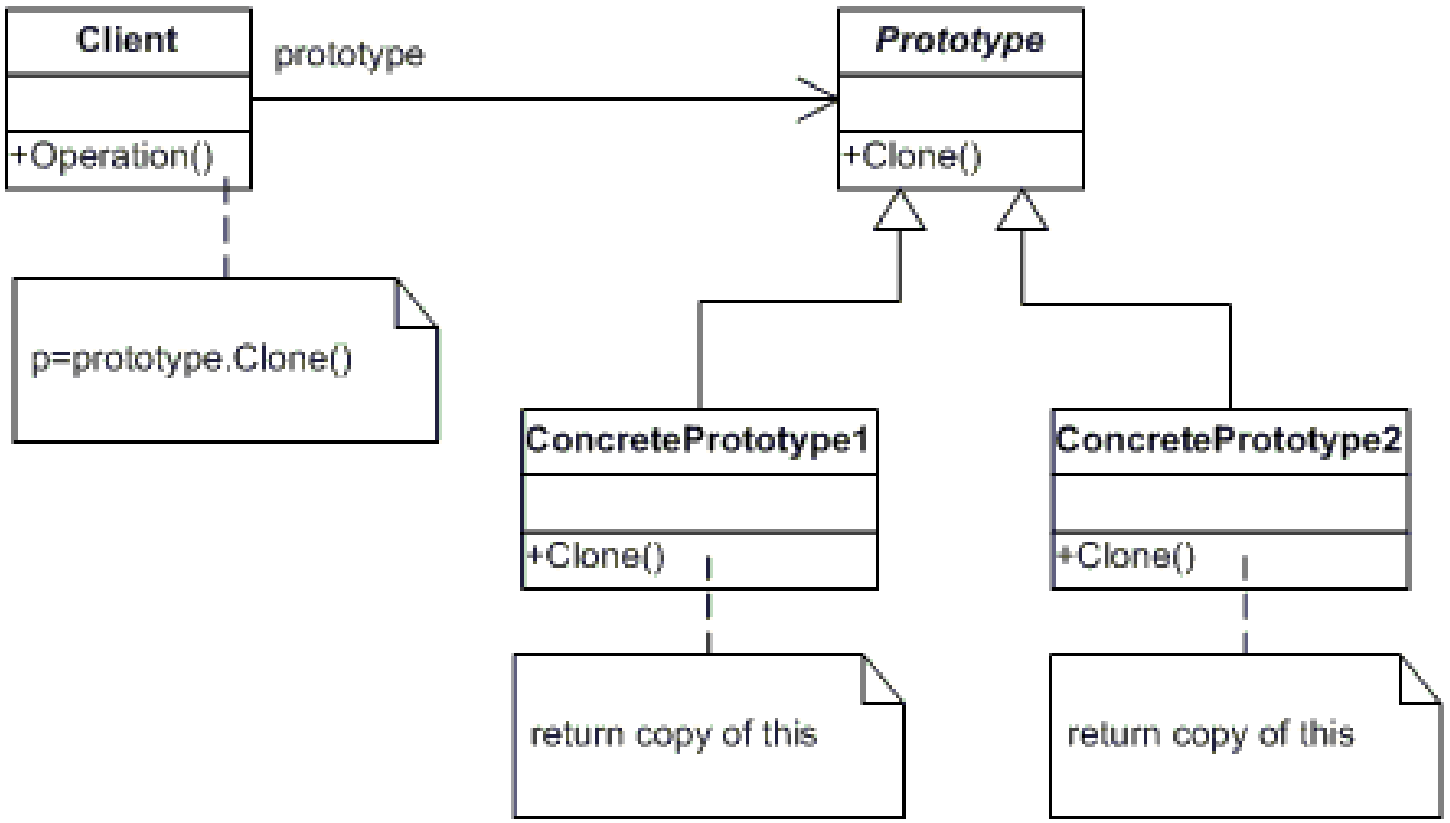
Factory Method

- Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

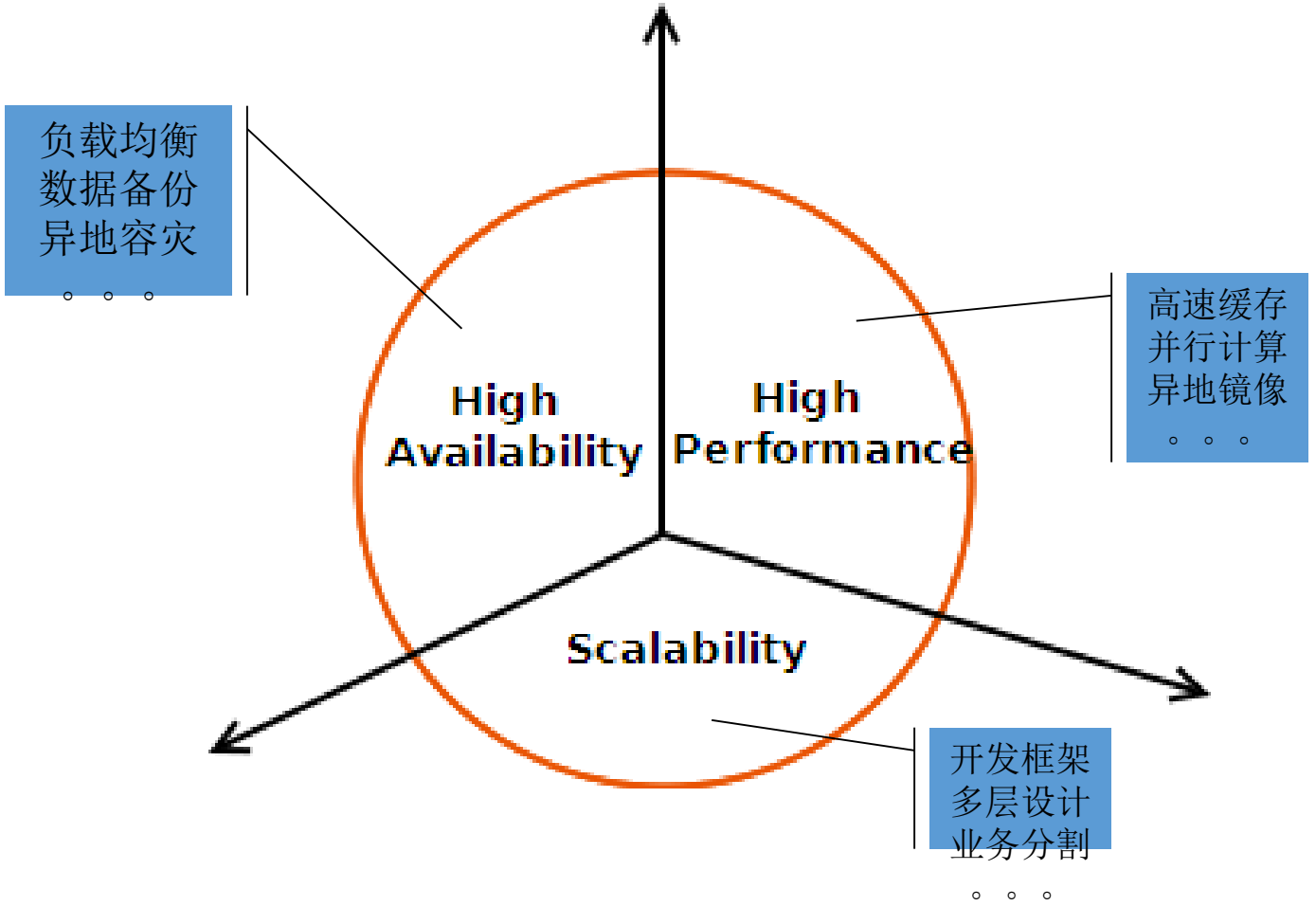

Product
Creator
+FactoryMethod()
+AnOperation()
product = FactoryMethod()
ConcreteProduct
ConcreteCreator
+FactoryMethod()
return new ConcreteProduct

Prototype

- Specify the kind of objects to create using a prototypical instance, and create new objects by copying this prototype.
- 本质上就是ICloneable接口


Client
prototype
Prototype
+Operation()
+Clone()
p=prototype.Clone()
ConcretePrototype1
ConcretePrototype2
+Clone()
+Clone()
return copy of this
return copy of this

面向海量用户设计目标



每个目标背后面临着技术、设计、维护等诸多方面的挑战。
而目标本身的期望值也会根据实际情况进行调整，这也意味着网站架构建设是个不断调整的过程。

系统整体架构的原则

- 页面静态化：全部静态化、局部静态化、伪静态化
- 存储特别是文件存储的独立
- 数据库集群和库表散列
- 缓存策略
- 镜像
- 负载均衡
- 硬件四层交换
- 软件四层交换

高可用性与高性能的设计

- 从单一服务器到HA/HP架构

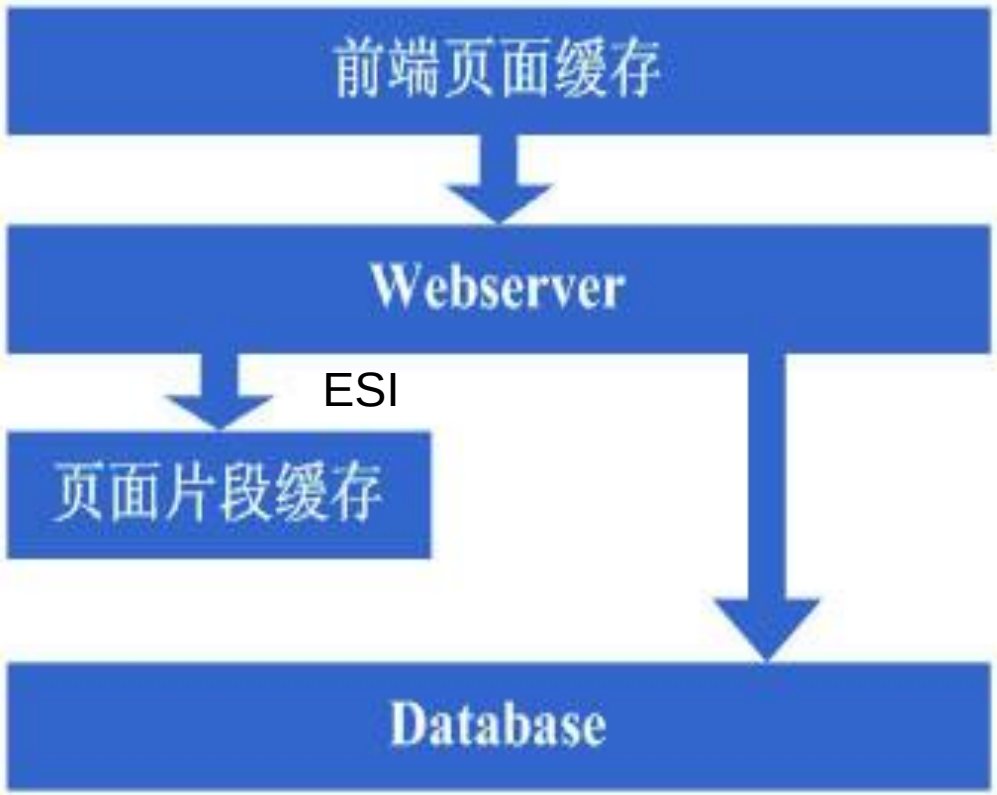
单一服务器

物理分离Web服务器和数据库

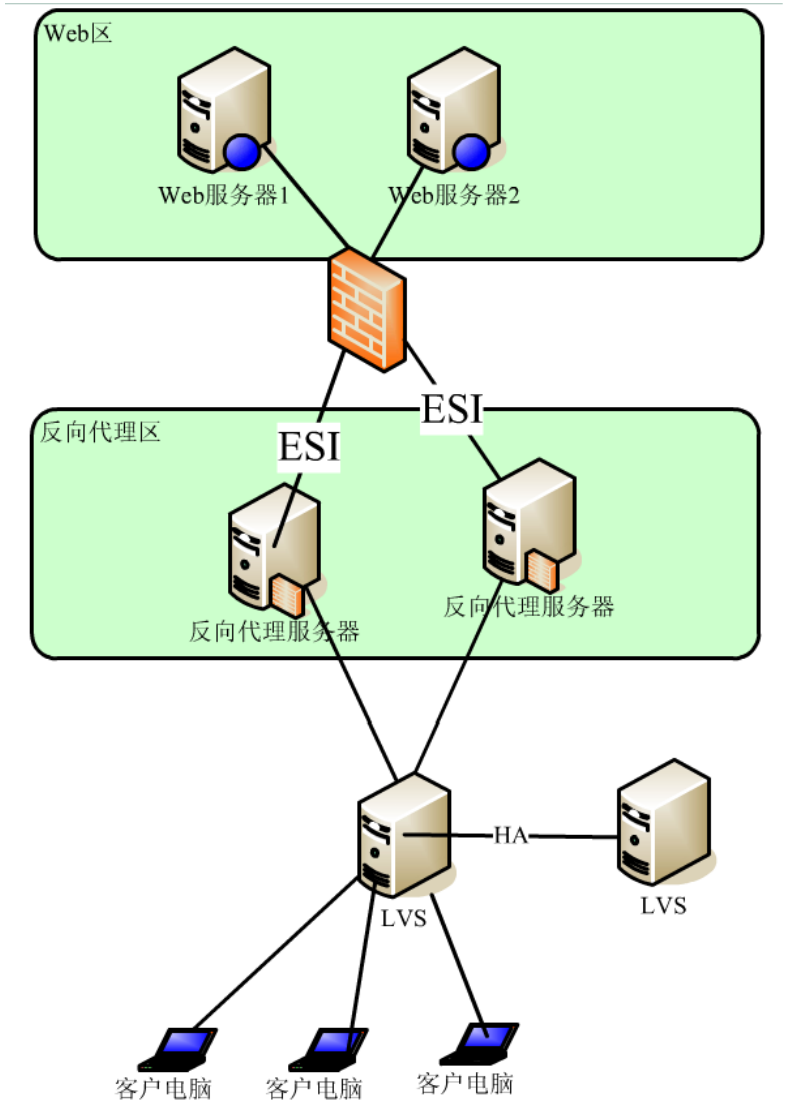
增加页面缓存

Squid/vanish

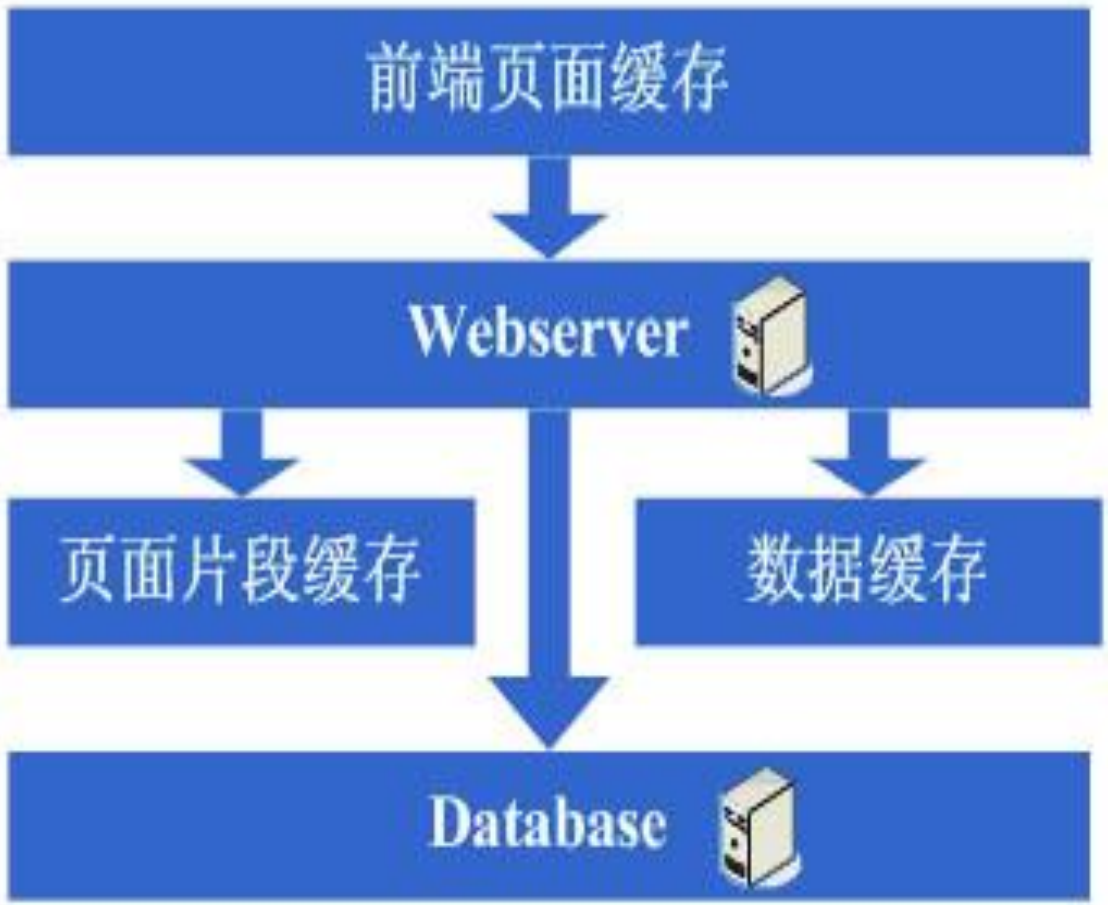
增加页面片段缓存



实际情况


Web区
Web服务器1
Web服务器2
ESI
ESI
反向代理区
反向代理服务器
反向代理服务器
HA
LVS
LVS
客户电脑
客户电脑
客户电脑

数据缓存

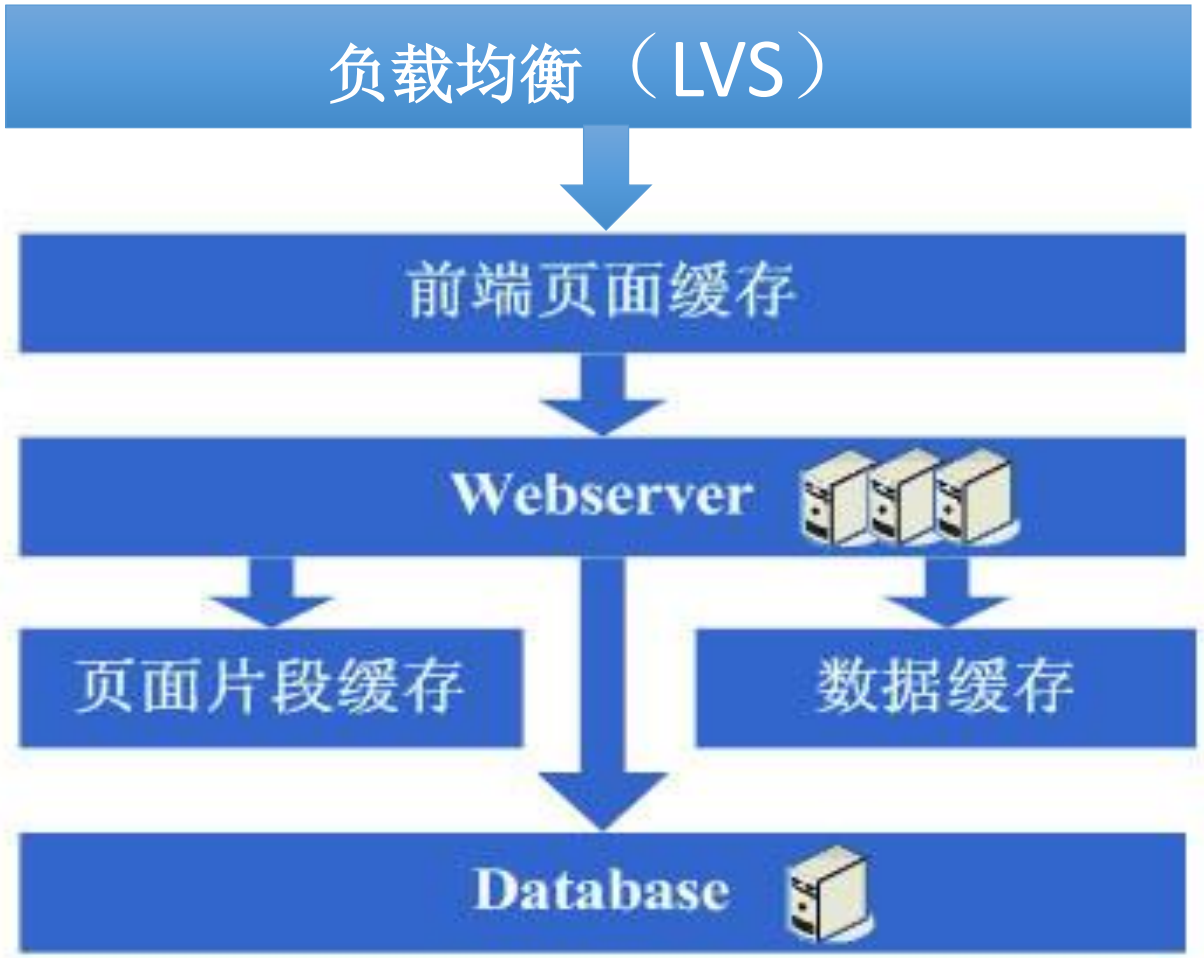


Squid/vanish

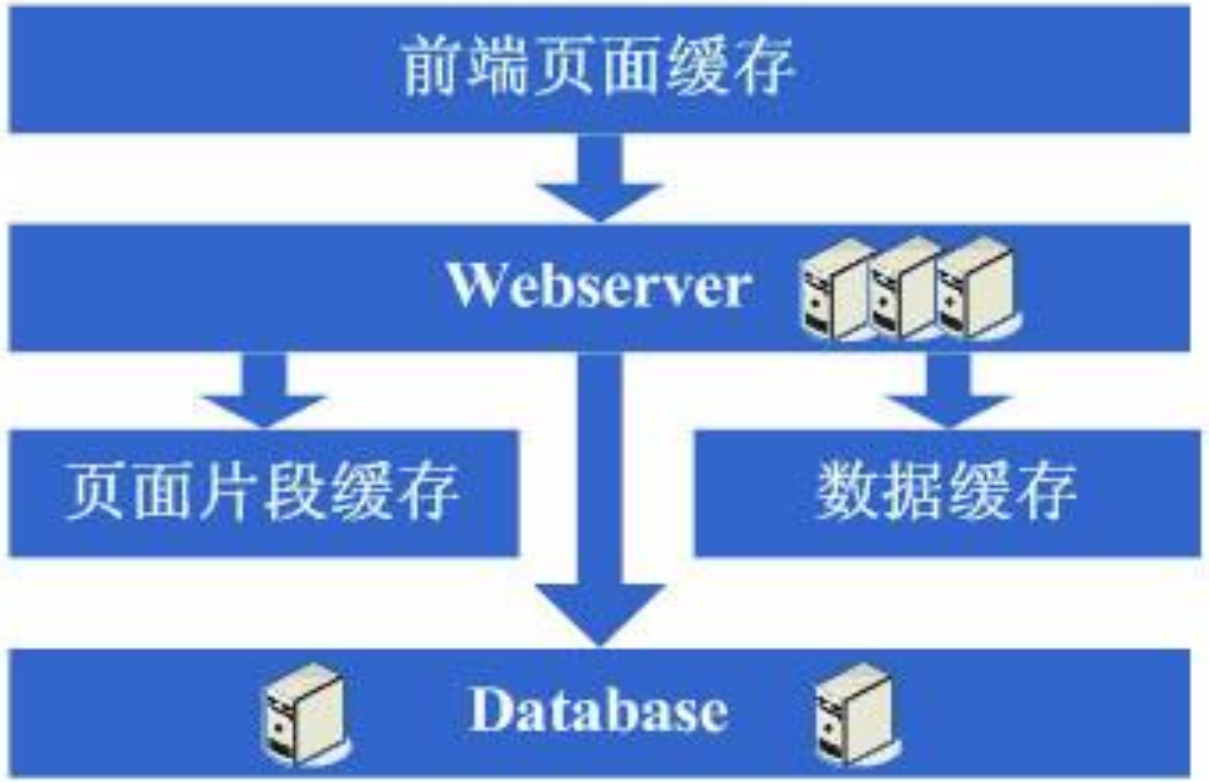
ESI

memcached

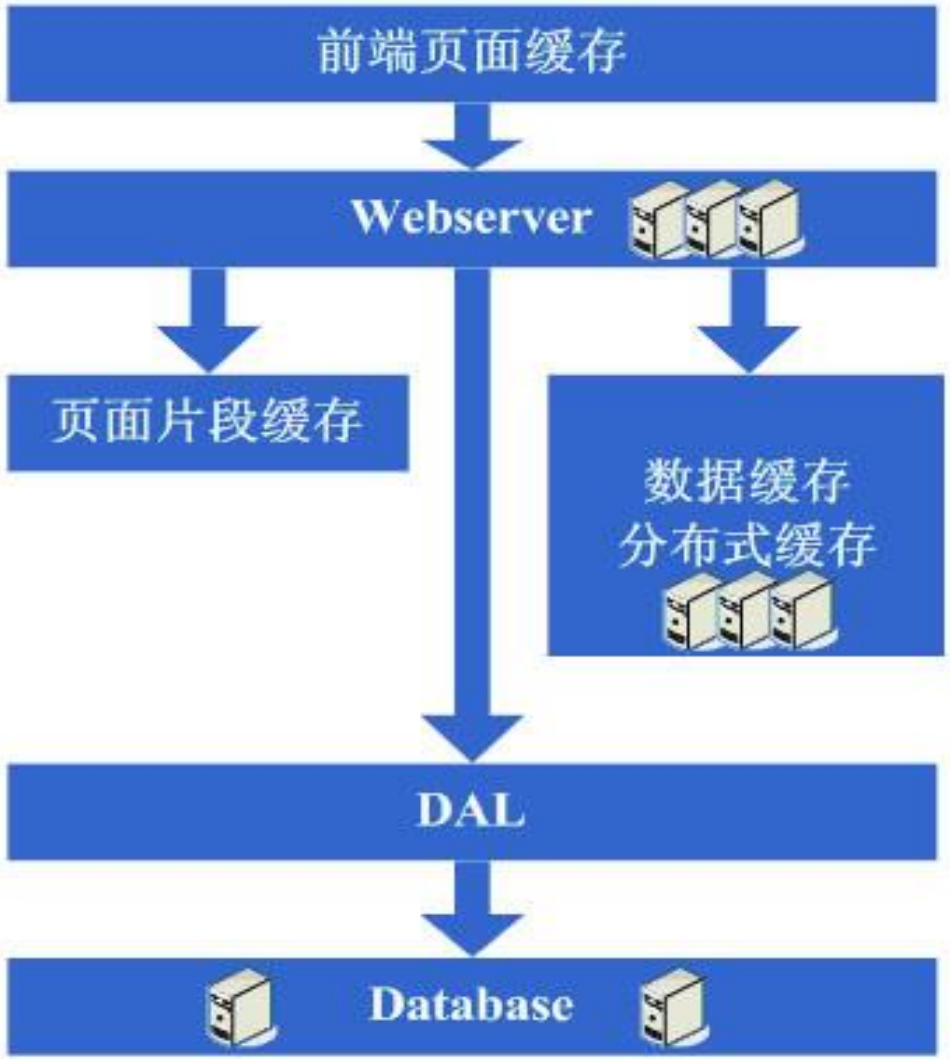
增加服务器和负载均衡



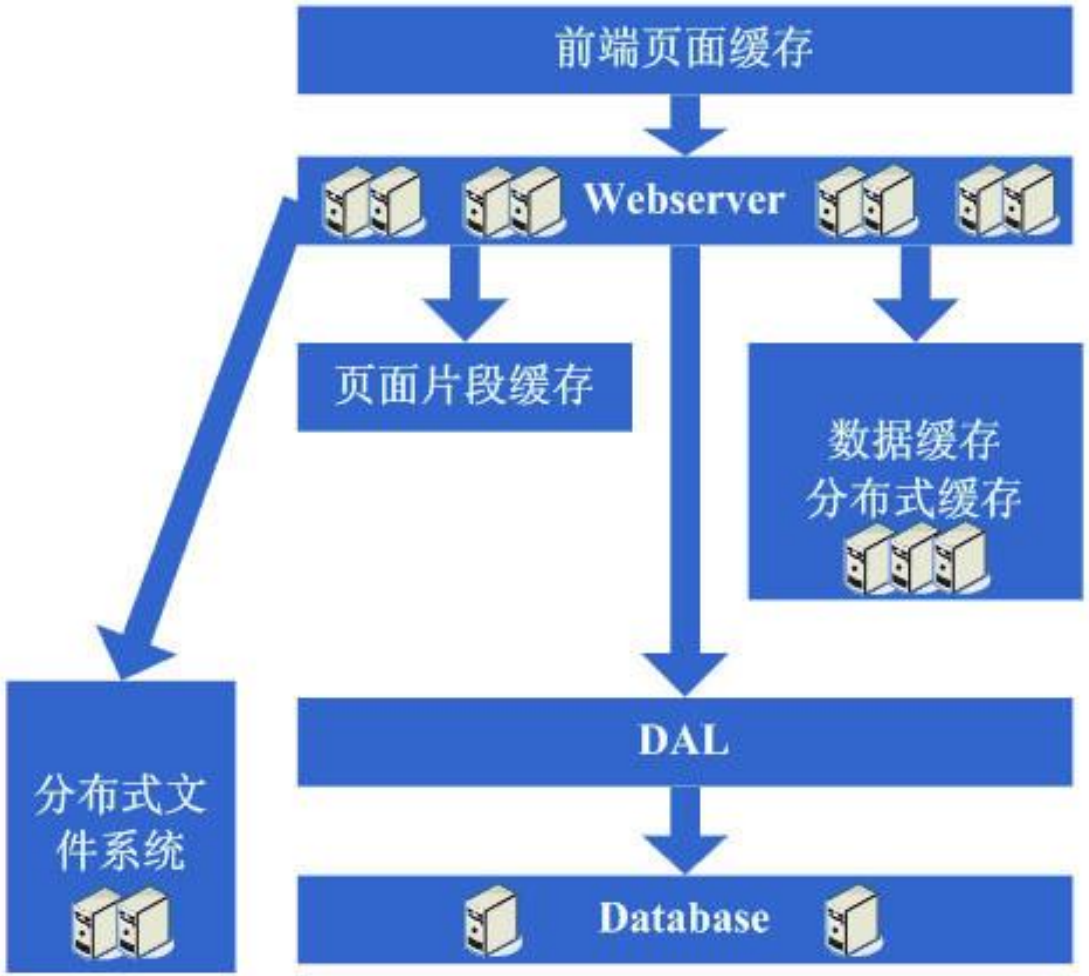
根据业务分库



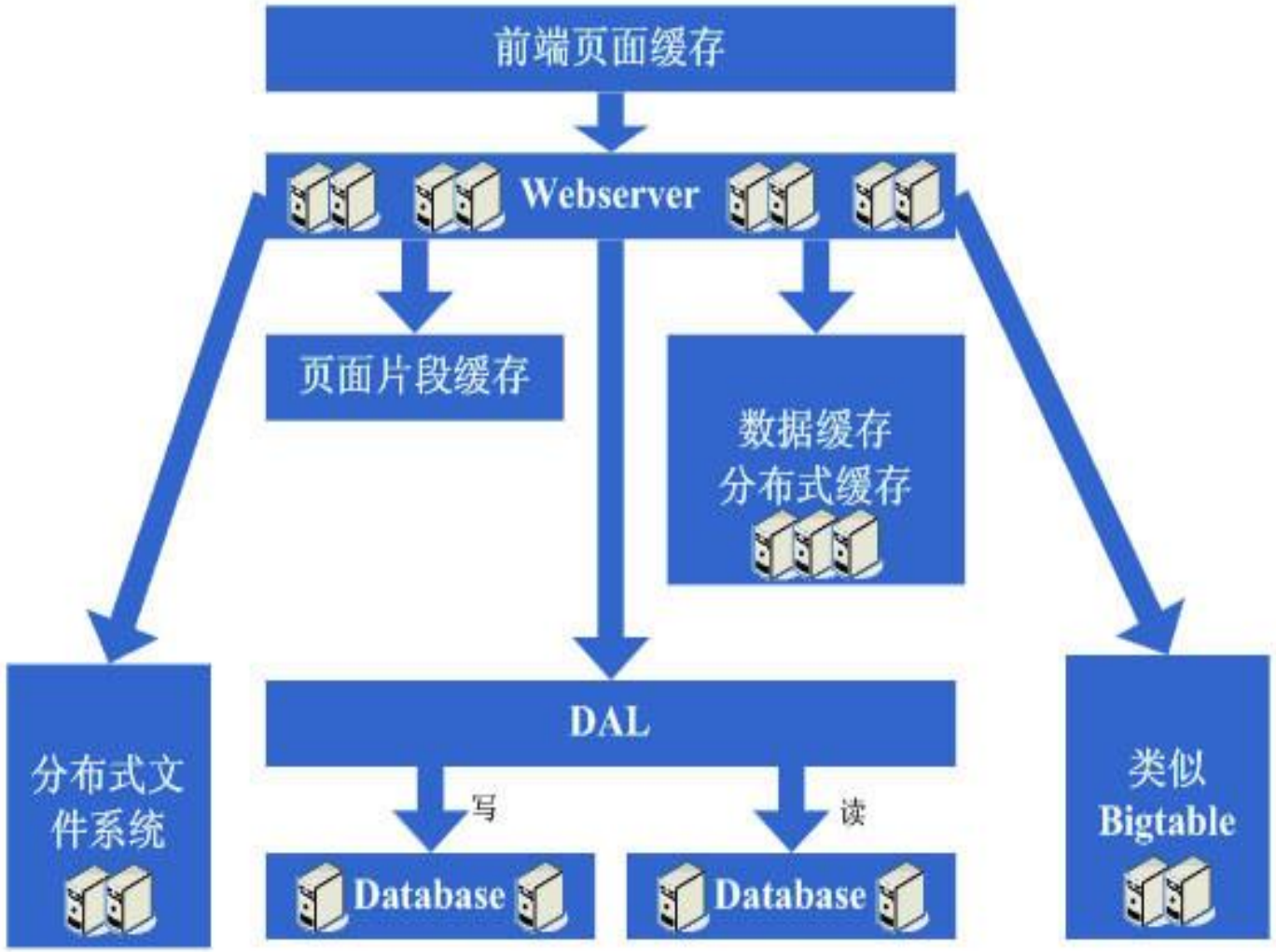
分表、DAL和分布式缓存



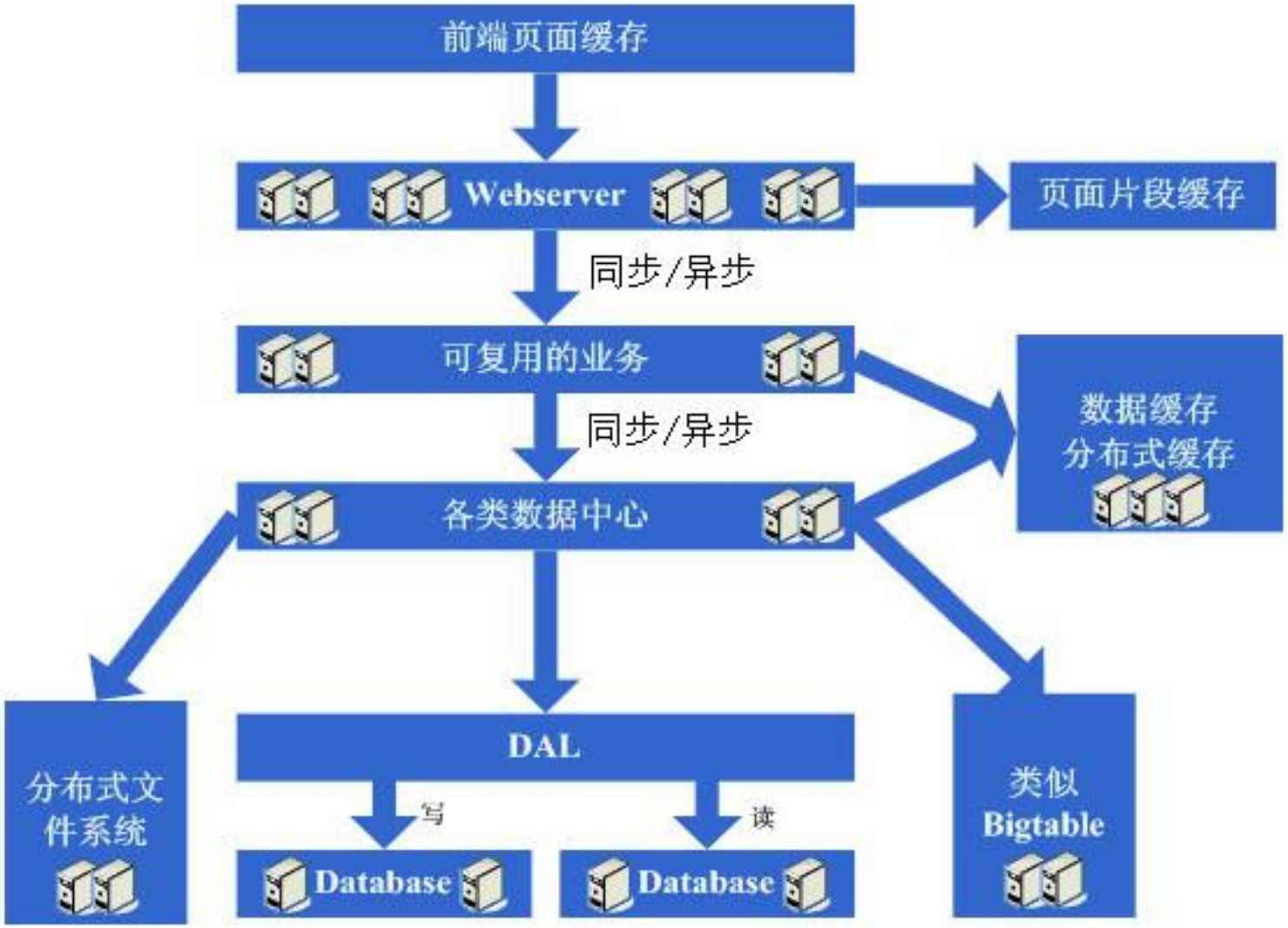
增加分布式文件系统



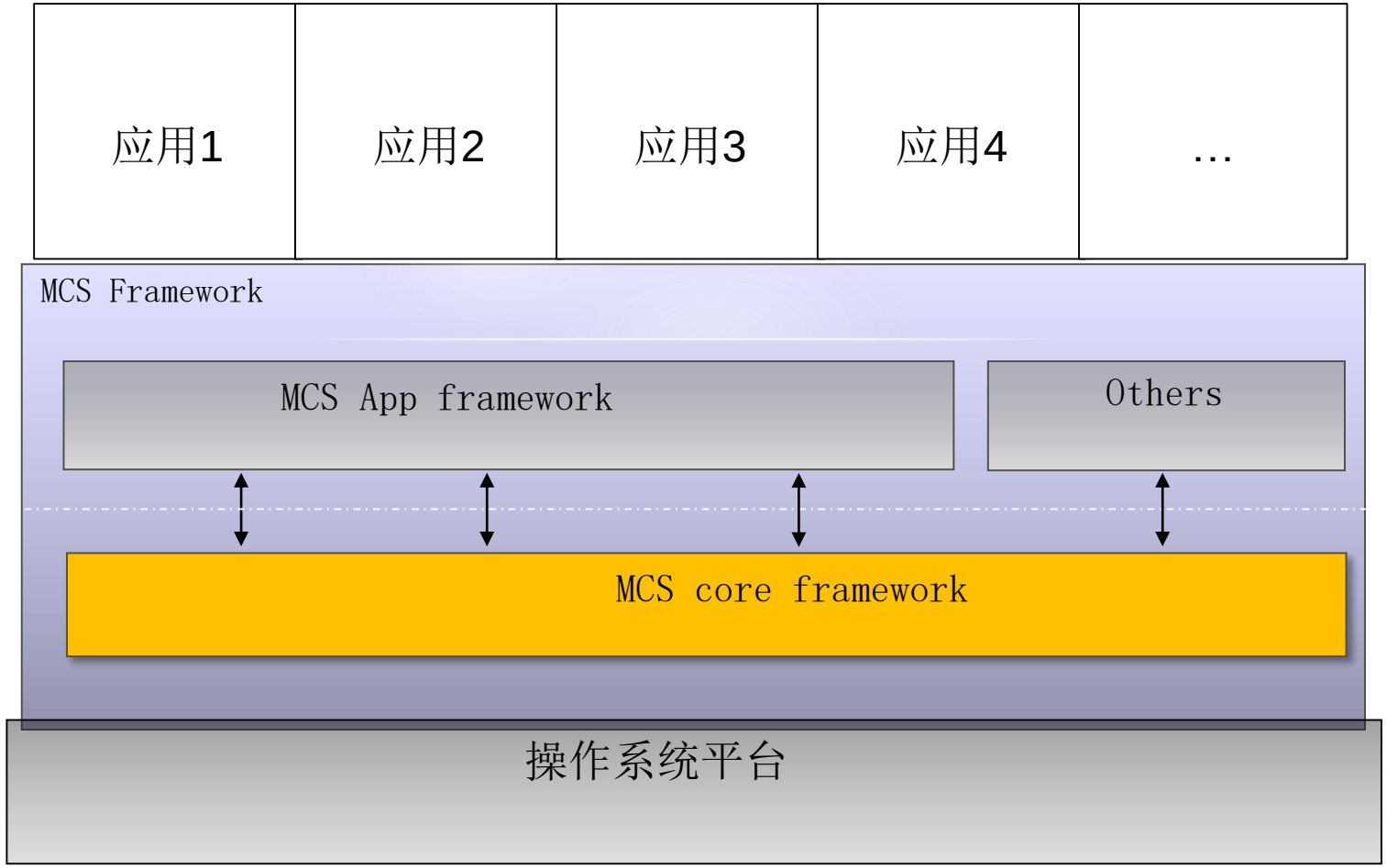
数据读写分离、廉价存储以及高性能



大型分布式应用和高性能架构

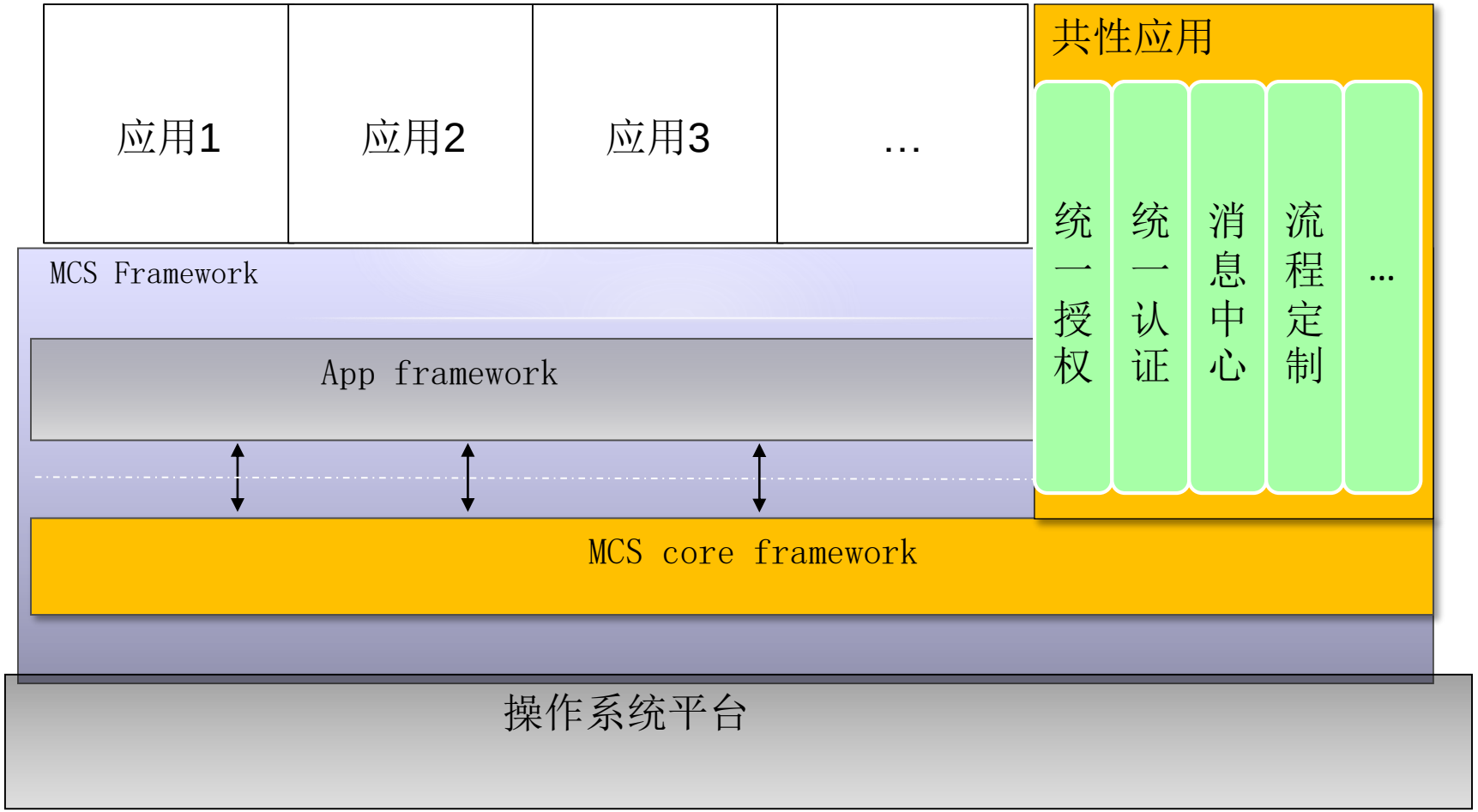


水平框架设计策略



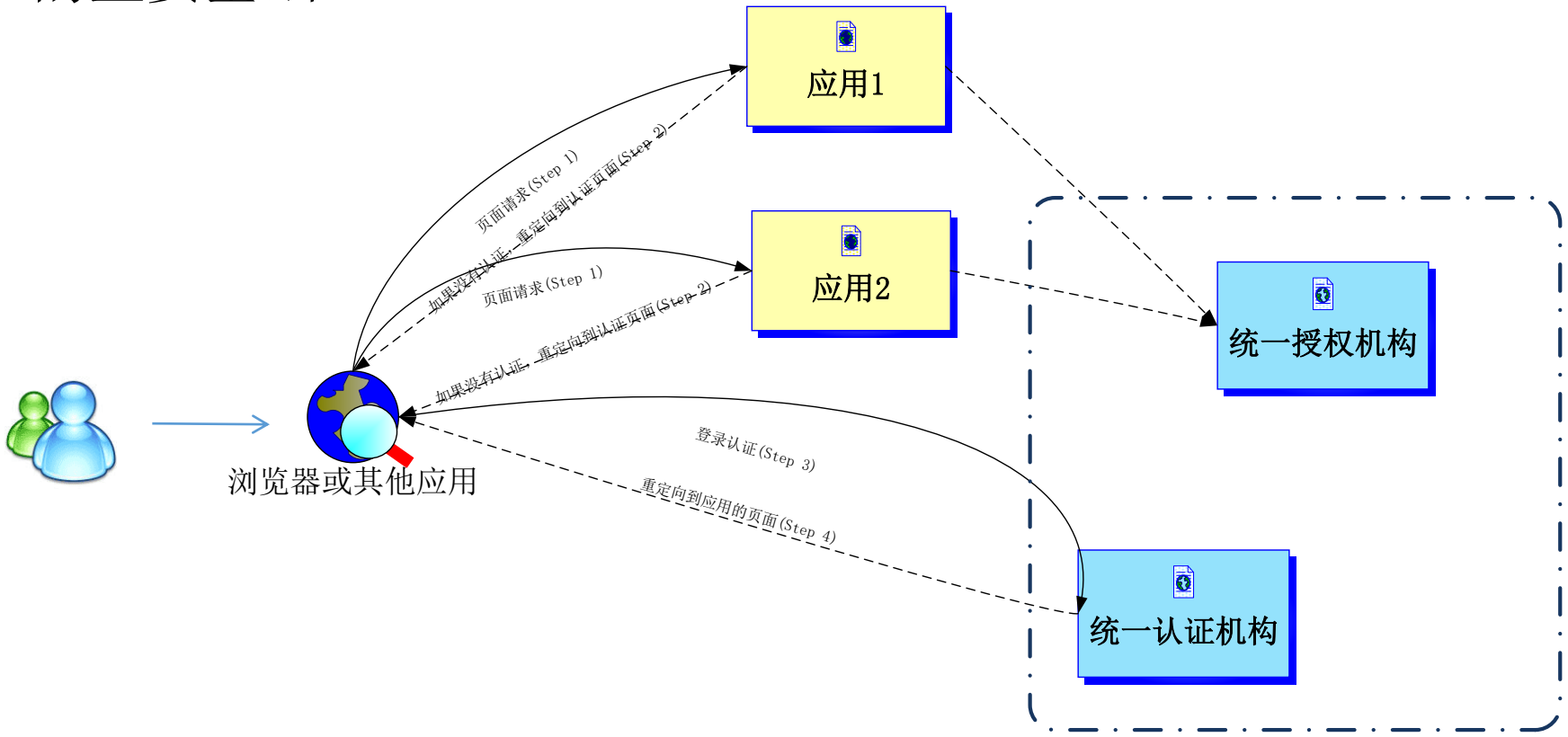
MCS App Framework：与应用相关的行业框架
MCS core Framework：与应用和行业无关的共性抽象框架

垂直设计策略


应用1
应用2
应用3
...
共性应用
统一授权
统一认证
消息中心
流程定制
...
MCS Framework
App framework
MCS core framework
操作系统平台

独立的身份验证及授权设计

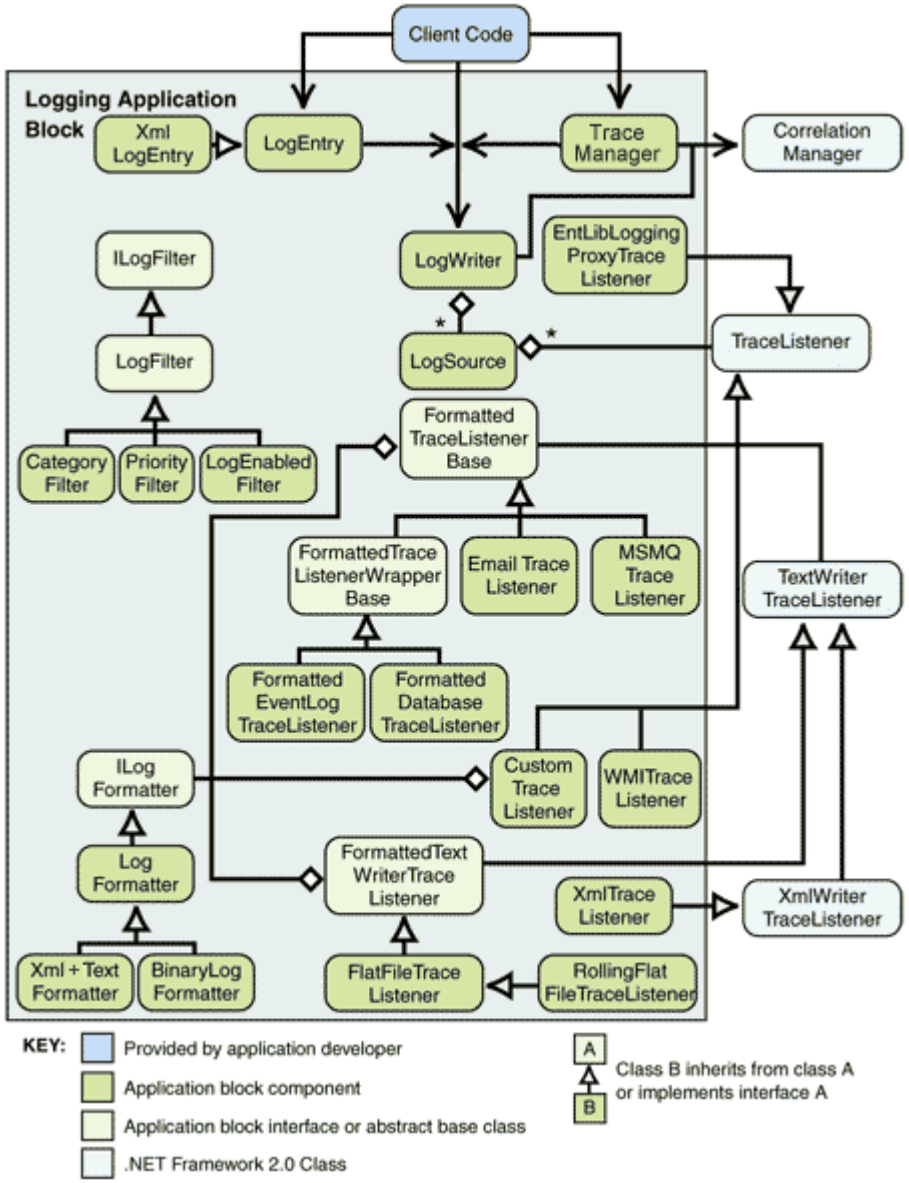
- 独立的统一认证机构和统一的授权机构： 把应用和授权/身份认证独立处理，使得应用开发从复杂的认证和权限中独立出来，这是实现HA/HP应用扩展的基础，也是实现SaaS在软件应用层面的必要环节，也是实现集团化应用程序开发的重要基础。



日志记录设计

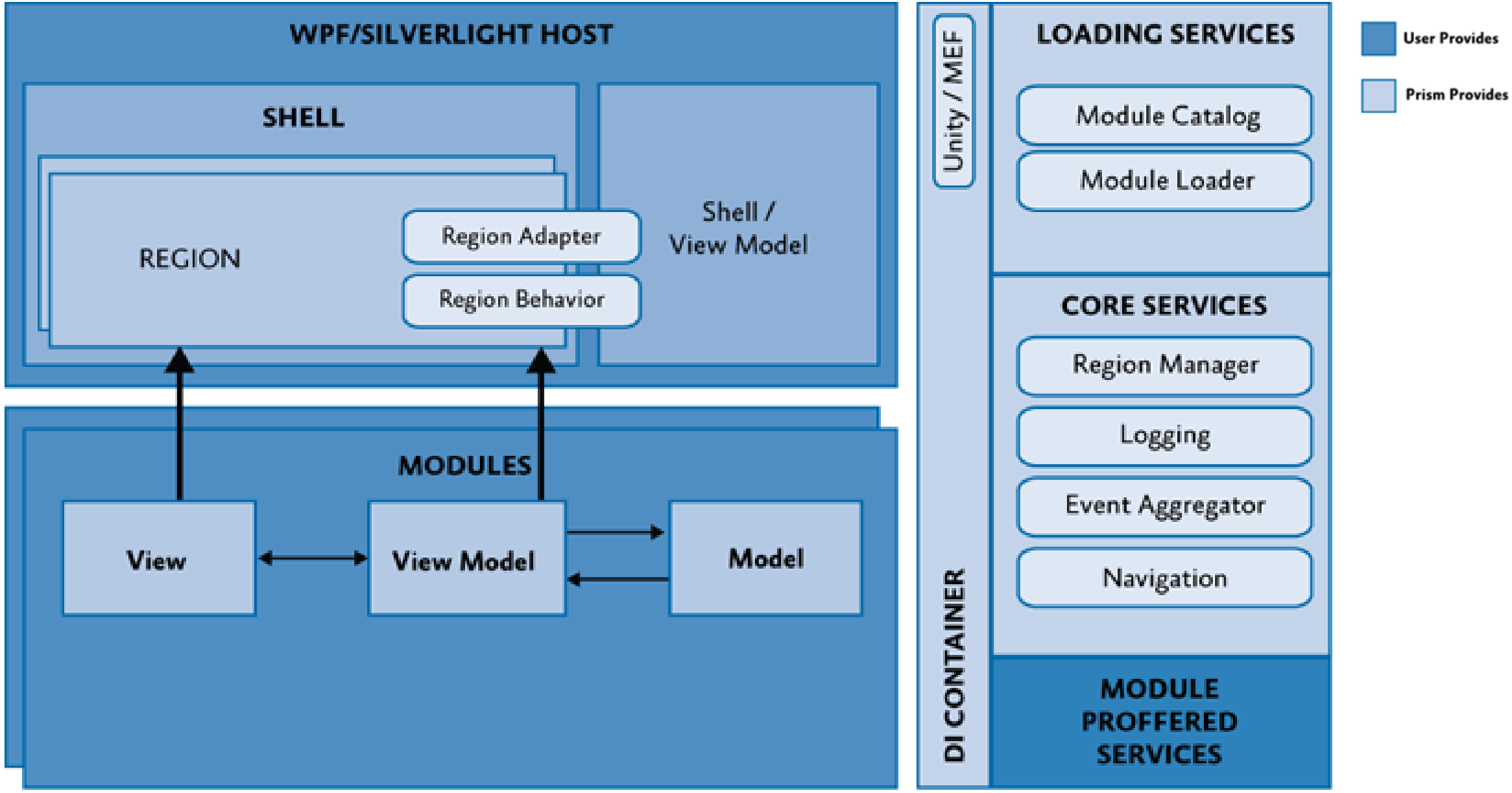
- 单向
- 只写
- 异步

Log

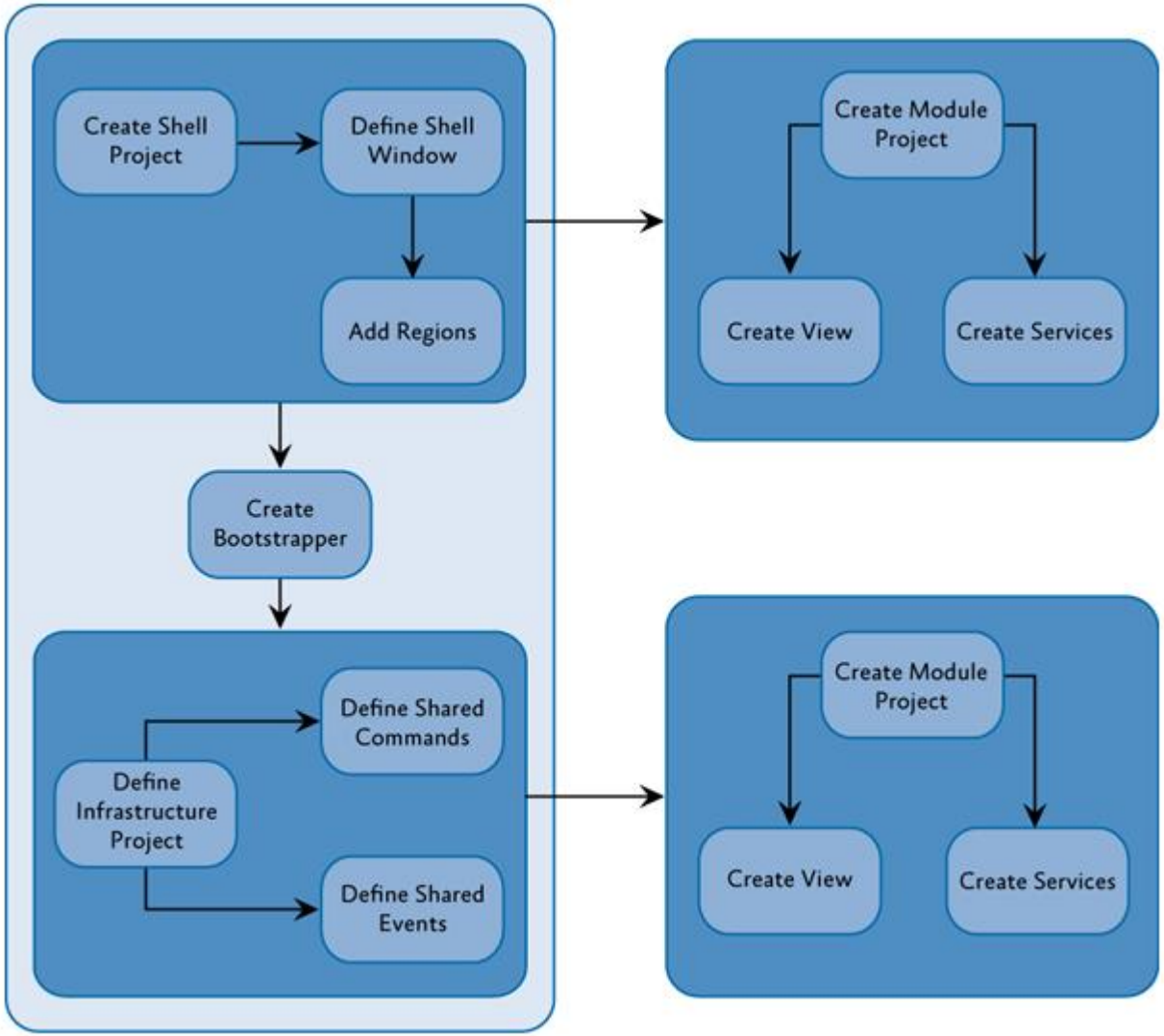

Client Code
Logging Application Block
Xml LogEntry
LogEntry
Trace Manager
Correlation Manager
ILogFilter
LogWriter
EntLibLogging ProxyTrace Listener
LogFilter
LogSource
TraceListener
Formatted TraceListener Base
Category Filter
Priority Filter
LogEnabled Filter
FormattedTrace ListenerWrapper Base
Email Trace Listener
MSMQ Trace Listener
TextWriter TraceListener
Formatted EventLog TraceListener
Formatted Database TraceListener
ILog Formatter
Custom Trace Listener
WMITrace Listener
Log Formatter
FormattedText WriterTrace Listener
XmlTrace Listener
XmlWriter TraceListener
Xml + Text Formatter
BinaryLog Formatter
FlatFileTrace Listener
RollingFlat FileTraceListener
KEY:
Provided by application developer
Application block component
Application block interface or abstract base class
.NET Framework 2.0 Class
A
B
Class B inherits from class A or implements interface A

Prism

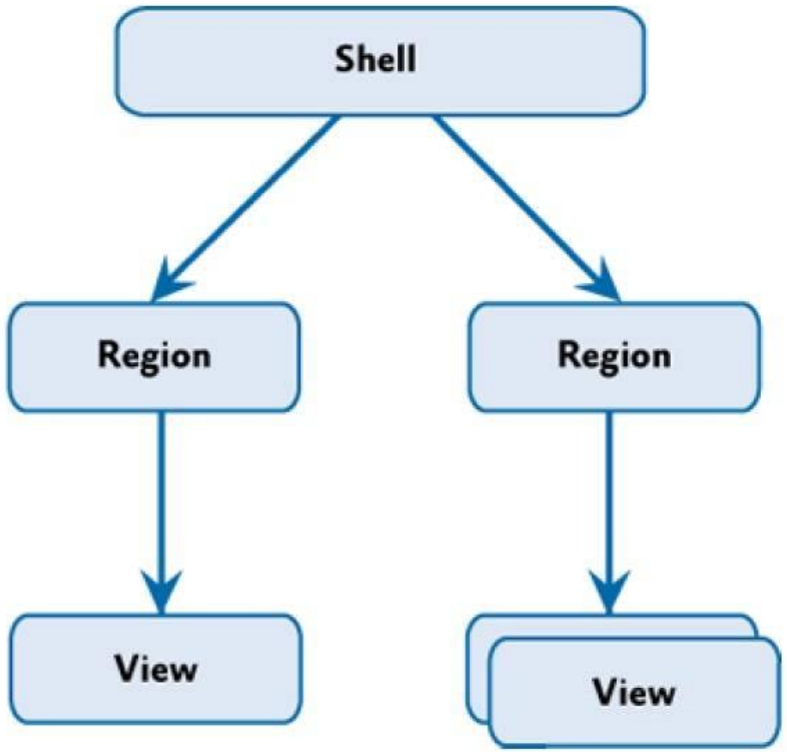
Prism Architecture



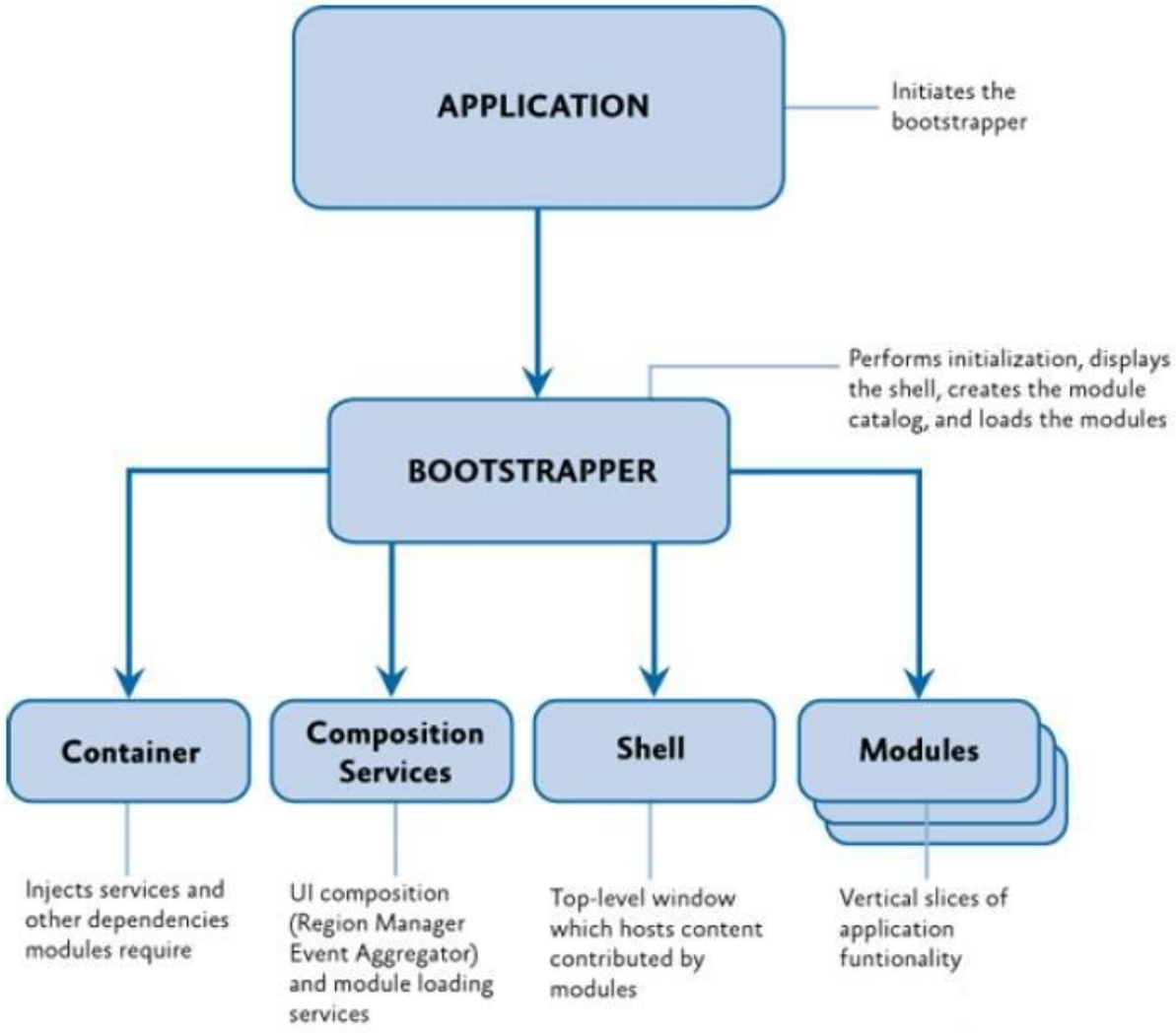
Activities for creating a composite application



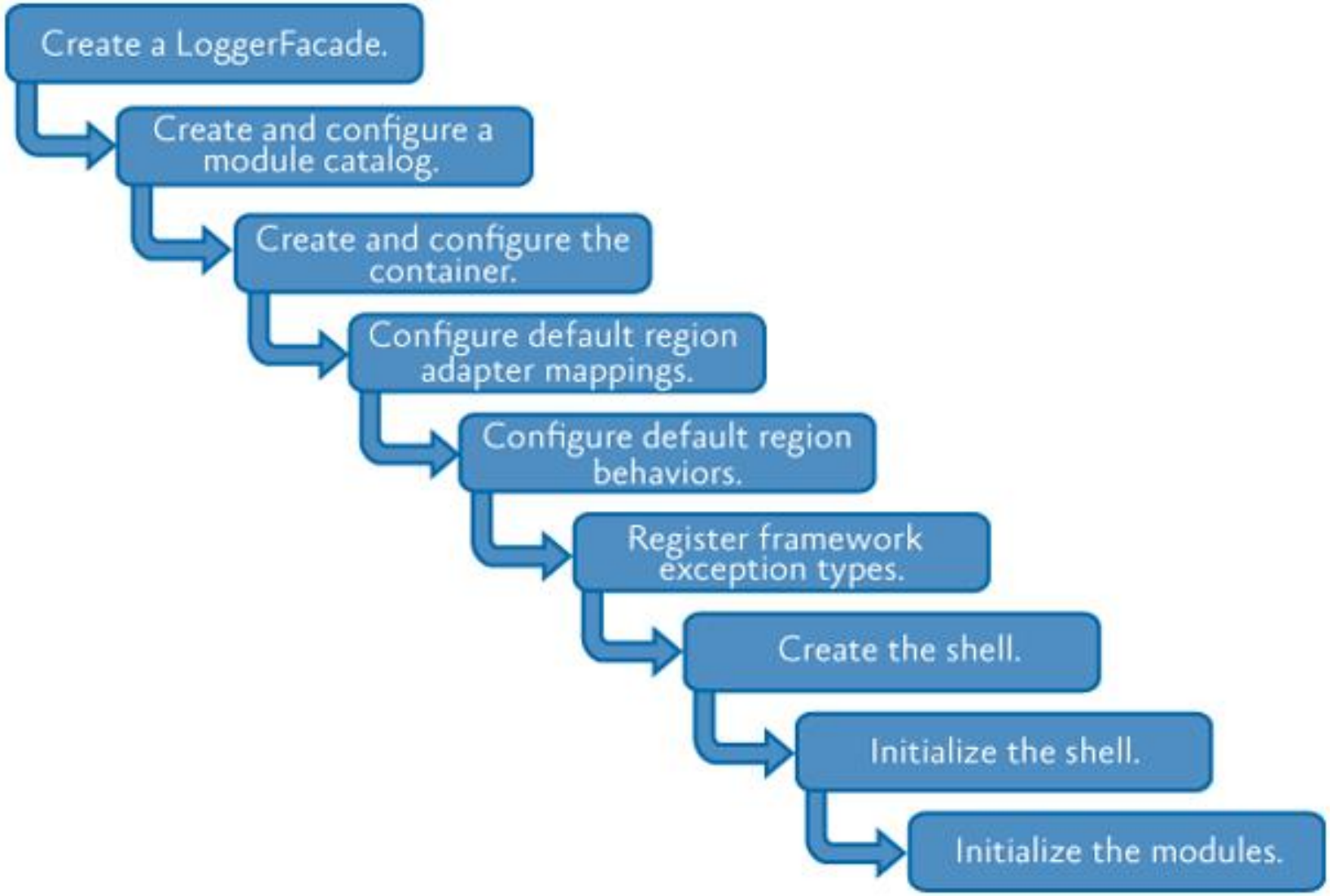
Shell、Region和View的关系



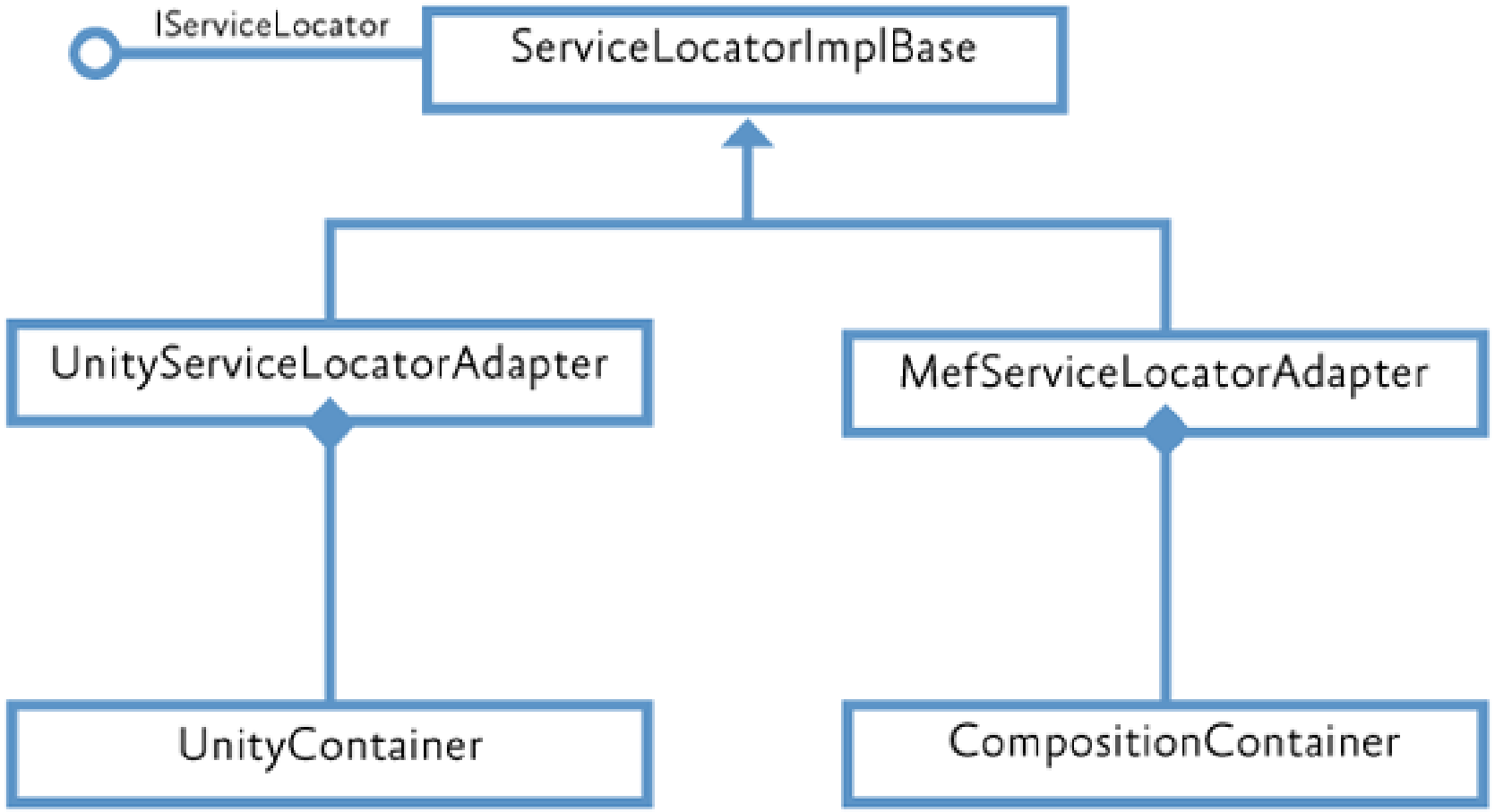
Bootstrapper



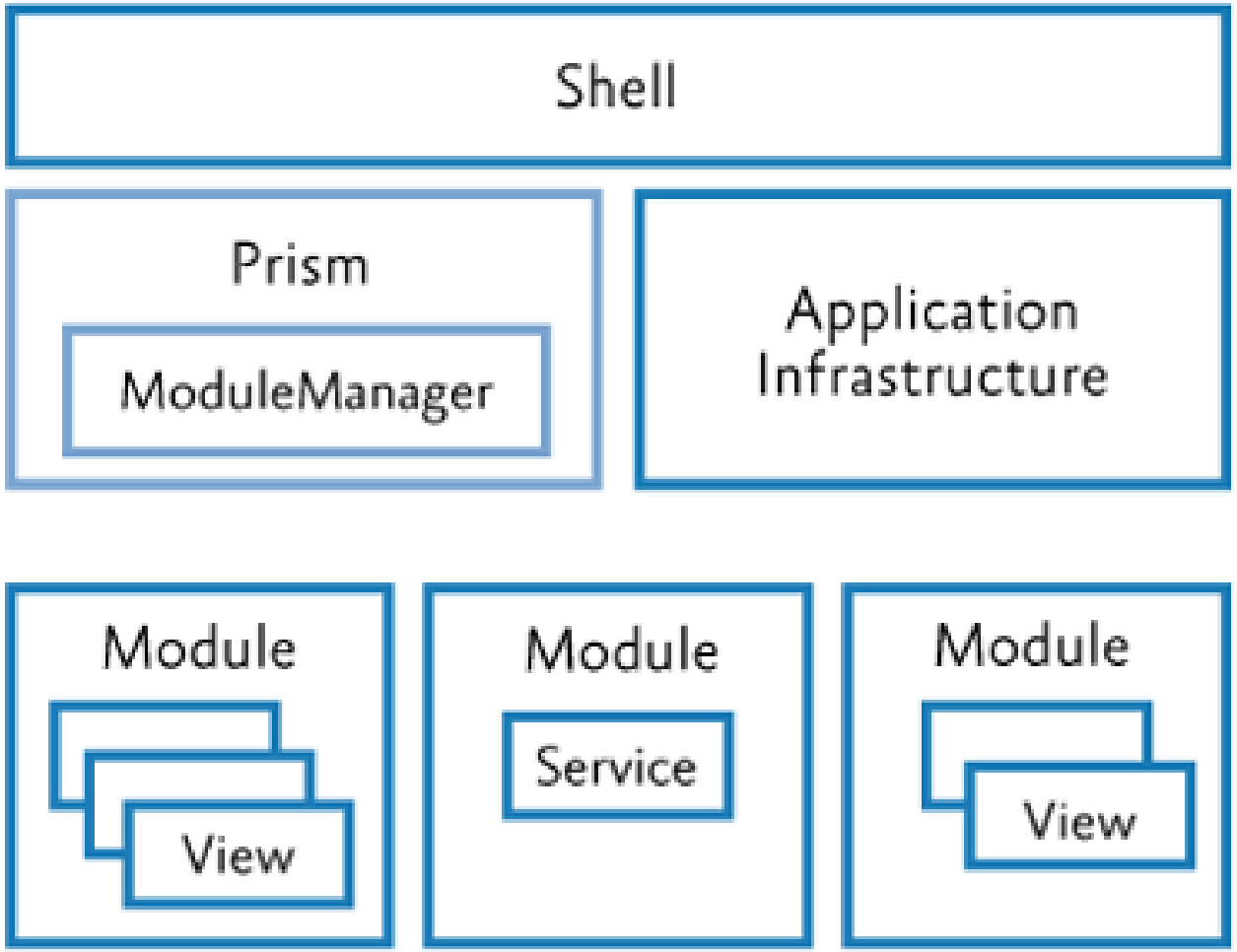
Bootstrapping process



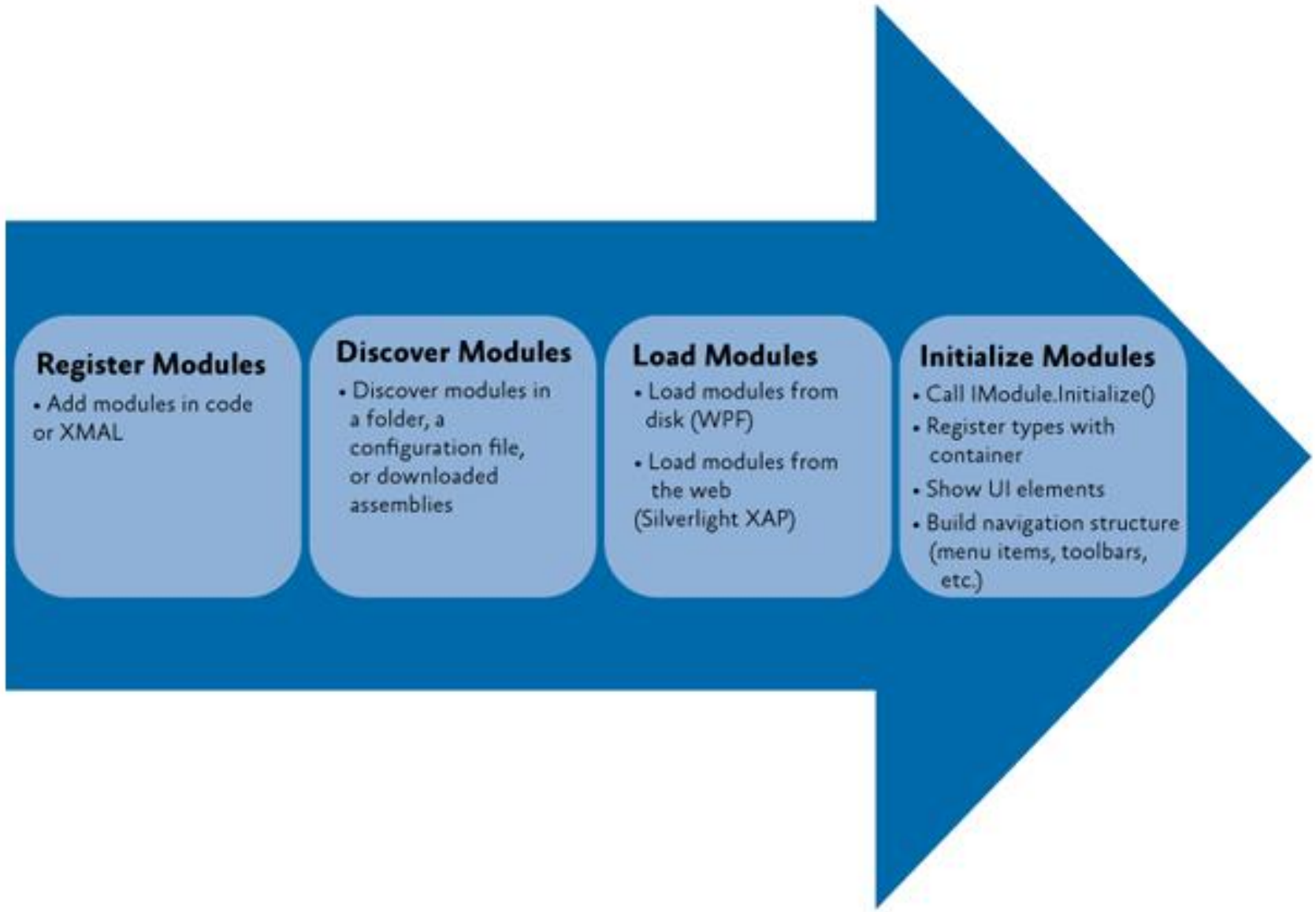
Common service locator



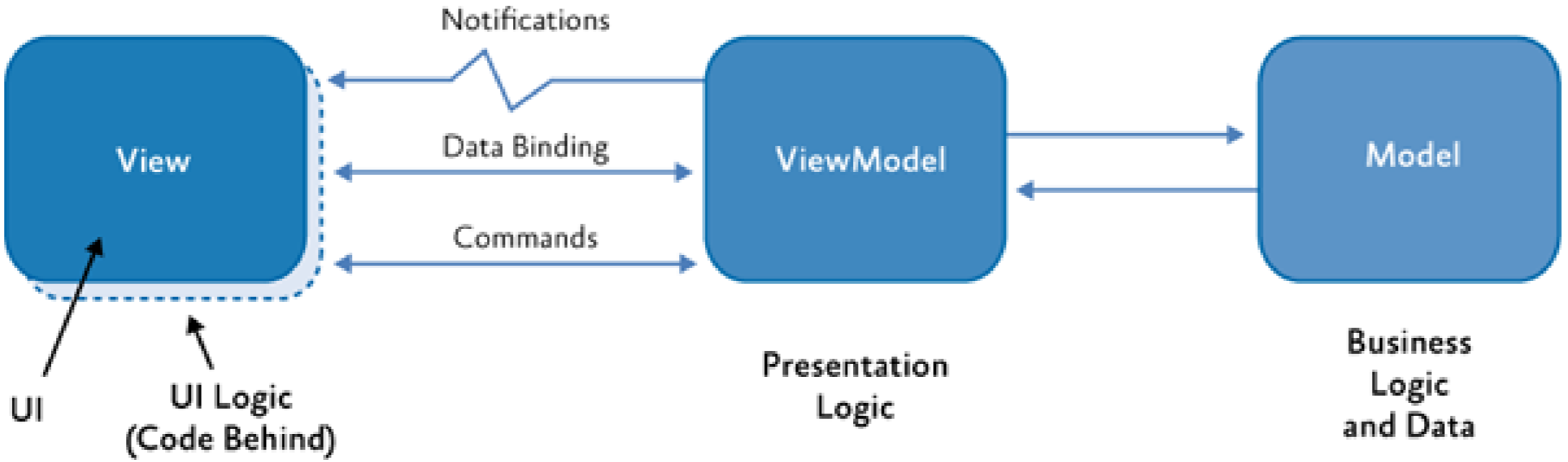
Modular application with multiple modules



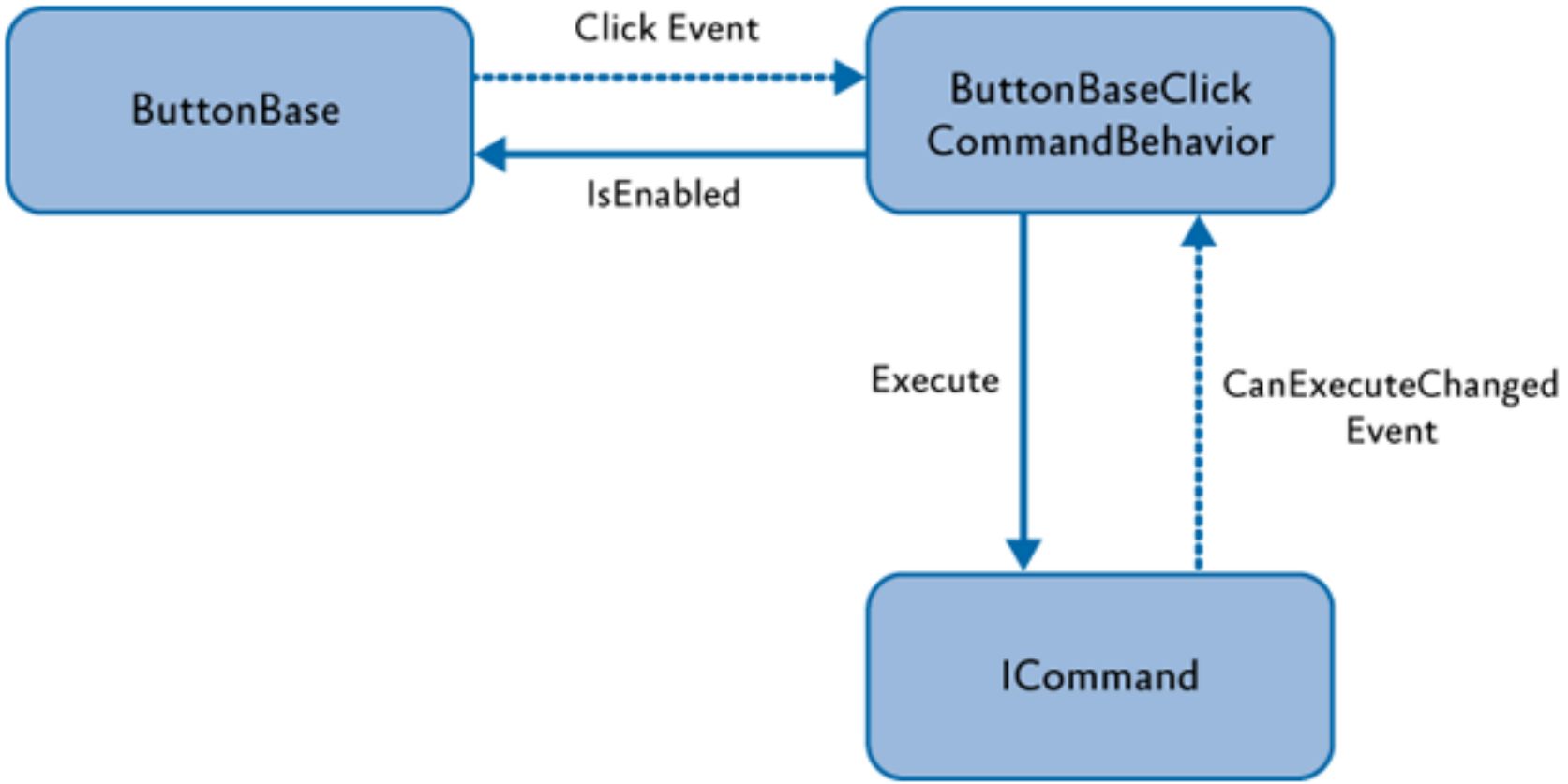
Module loading process



MVVM classes and interaction



Event forwarding



Unit4：TDD

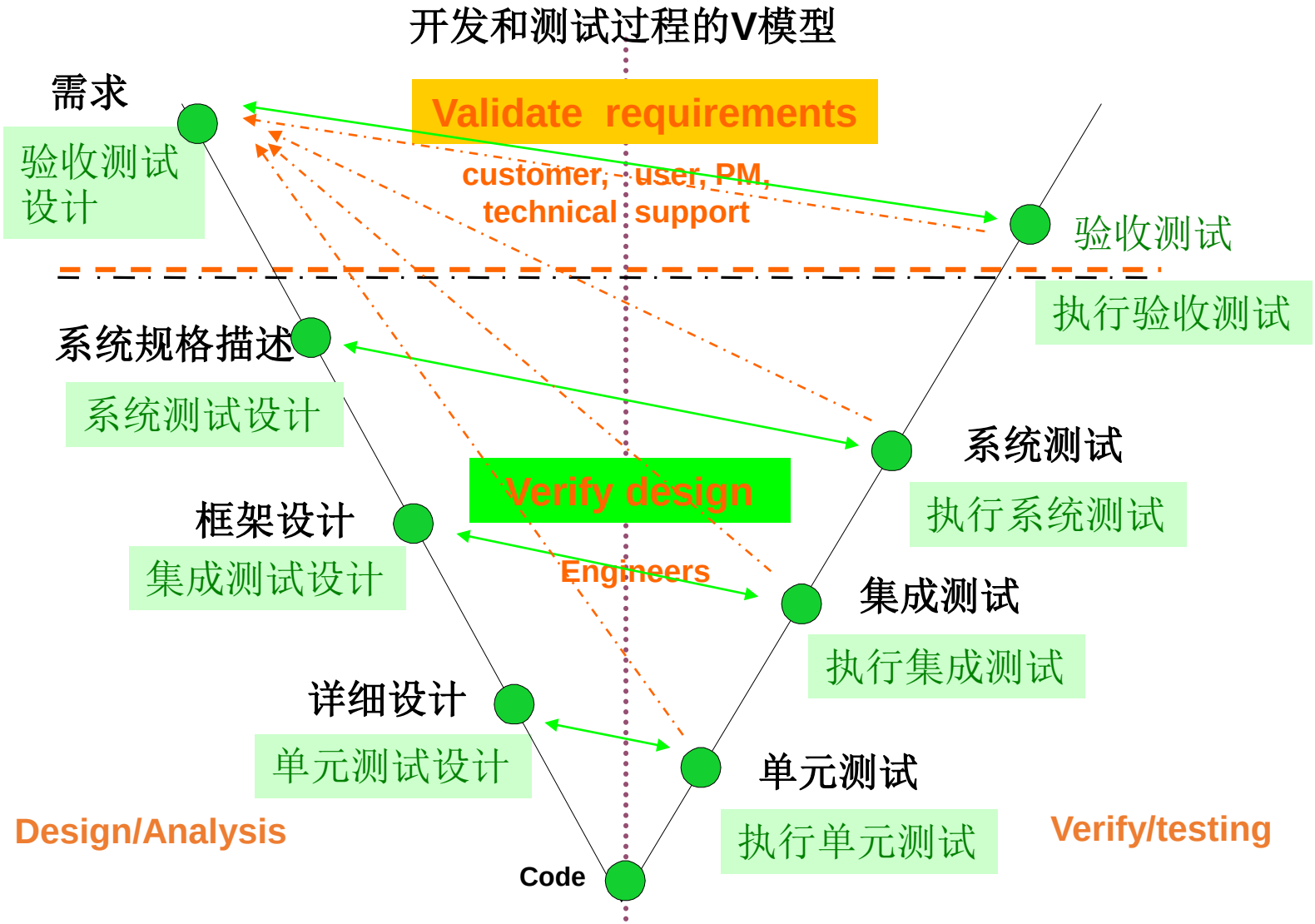
TDD（Test Drive Development）

TDD要求在编写某个功能的代码之前先编写测试代码，然后只编写使测试通过的功能代码，通过测试来推动整个开发的进行。这有助于编写简洁可用和高质量的代码，并加速开发过程。

TDD流程

1. 明确当前要完成的功能。可以记录成一个 TODO 列表。
2. 快速完成针对此功能的测试用例编写。
3. 测试代码编译不通过。
4. 编写对应的功能代码。
5. 测试通过。
6. 对代码进行重构，并保证测试通过。
7. 循环完成所有功能的开发。

V 模型 (Improved)

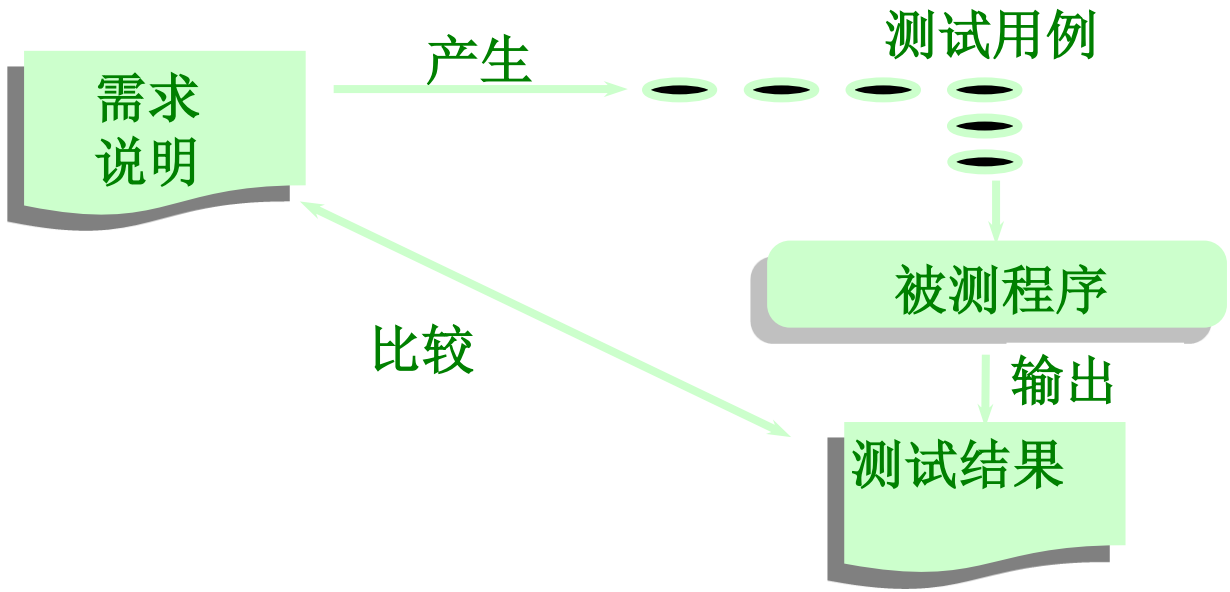


软件测试技术

- 黑盒测试/白盒测试

黑盒测试和白盒测试

- 什么是黑盒测试
  - 又称功能测试或数据驱动测试，是针对软件的功能需求/实现进行测试；基于规格说明书或用户手册的测试
  - 它所依据的是程序的外部特性。通过测试来检测每个功能是否符合需求，不考虑程序内部的逻辑结构
  - 穷举输入测试



黑盒测试的准则

- 基于需求规格说明书的测试（需求矩阵）
  - 需求列表
  - 设计列表:概要设计和详细设计
  - 测试用例与需求的对照表

需求	设计	编码	测试
rq1	de1	f1	tc1
rq1	de2	f2	tc2
rq1	de3	f3	tc3
rq1	de3	f4	tc4
rq2	de4	…	tc5
rq3	de5	…	tc6
rq4	de6		tc7

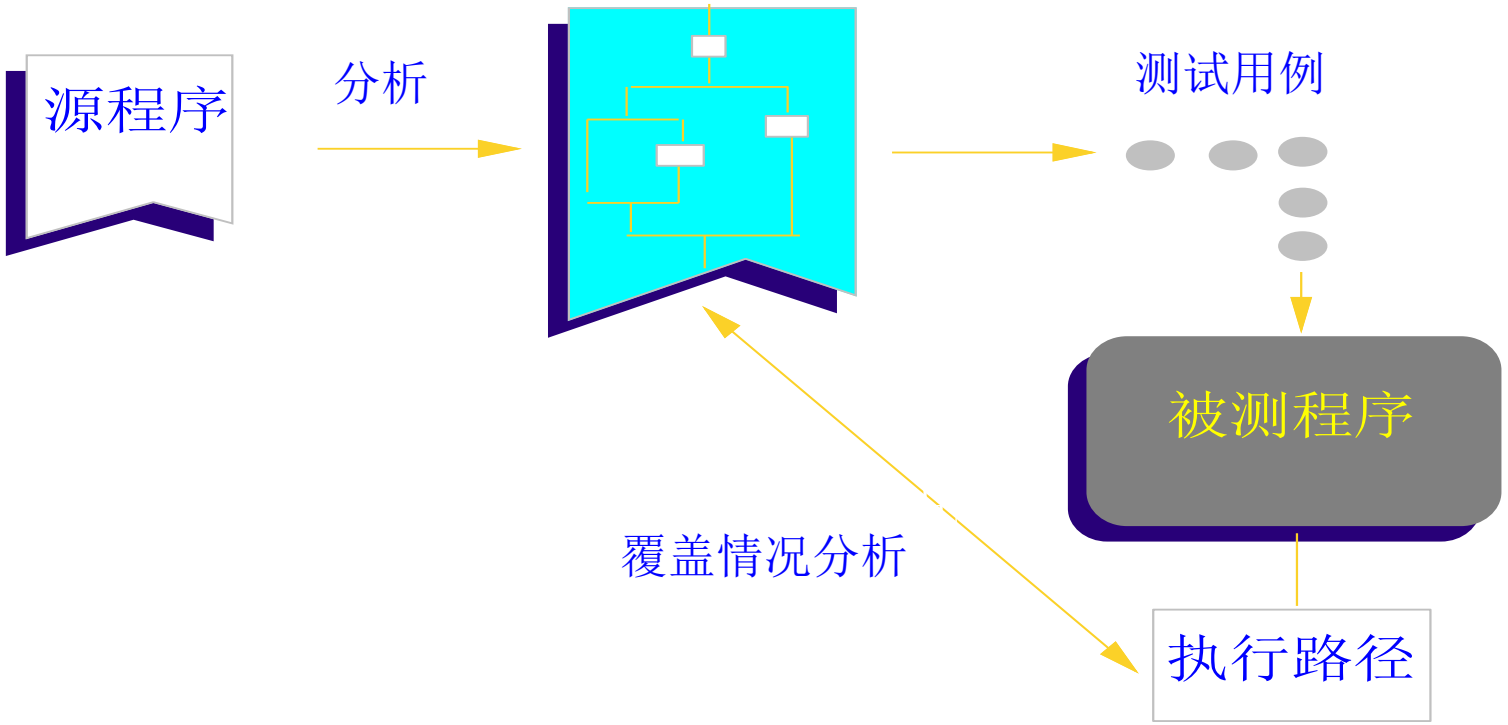
黑盒测试和白盒测试

- 黑盒测试方法
  - 功能划分
  - 等价类划分
  - 边界值分析
  - 因果图
  - 错误推测等

黑盒测试和白盒测试

什么是白盒测试

- 白盒测试也称结构测试或逻辑驱动测试
- 必须知道软件内部工作过程，通过测试来检测软件内部是否按照需求、设计正常运行
- 通过逻辑覆盖、路径覆盖等方式选择测试用例，可以用测试覆盖率评价测试用例



黑盒测试和白盒测试

- 白盒测试的主要方法
  - 对应于程序的一些主要结构：语句、分支、逻辑路径、变量；白盒测试的主要方法是：
    - 语句覆盖方法
    - 分支覆盖方法
    - 逻辑覆盖方法

白盒测试覆盖准则

- 语句覆盖
  - -在测试时，设计若干测试用例，运行被测程序，使程序中的每个可执行语句至少执行一次。分支覆盖-在测试时，设计若干测试用例，运行被测程序，使程序中的每个判断真假的分支至少遍历一次。
- 条件覆盖
  - -在测试时，设计若干测试用例，运行被测程序，使程序中的每个条件的可能取值至少满足一次。
- 条件分支覆盖
  - -在测试时，设计足够的测试用例，使得判断中每个条件的所有可能取值至少出现一次，并且每个判断本身的判定结果也至少出现一次。
- 路径覆盖
  - - 设计足够多的测试用例，要求覆盖程序中所有可能的路径。

Moq

- Moq(发音”Mock-you”或”Mock”), 是一个基于.NET的Mocking框架。支持模拟接口和类等。
- 为何要用Moq:
  - 真实对象具有不可确定的行为(产生不可预测的结果，如股票的行情)
  - 真实对象很难被创建(比如具体的web容器)
  - 真实对象的某些行为很难触发(比如网络错误)
  - 真实情况令程序的运行速度很慢
  - 真实对象实际上并不存在
  - 等等

其他模拟对象的框架

- NSubstitute
- Rhino Mocks
- TypeMock
- EasyMock.NET
- NMock
- FakeItEasy

Thank you!