INTRODUCTION TO ARCHITECTURAL PATTERNS

Kirkin Stanislav
KN – 33zh
NTU “KHPI”
stkirkin@gmail.com

DEFINITIONS

Архитектура приложения — это логическая структура, описывающая отдельные компоненты, их свойства и связи в виде единой системы.

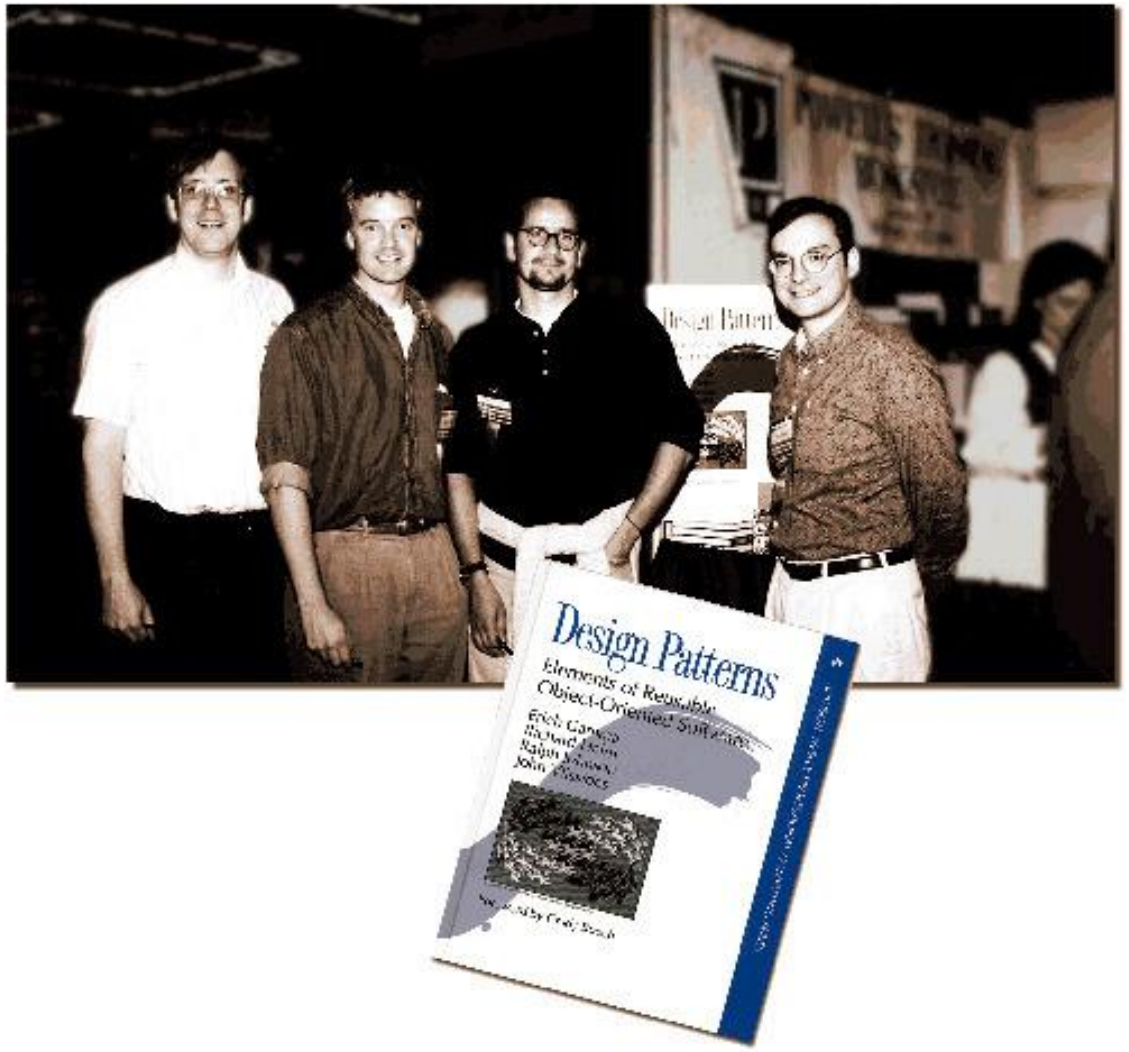
DEFINITIONS

- *Паттерны* — это описания схем детализации отдельных подсистем приложения и взаимосвязей между ними.
- *MVC* – программная парадигма *архитектурных паттернов*: модель — представление — контроллер.

BENEFITS THAT PATTERNS GIVE US

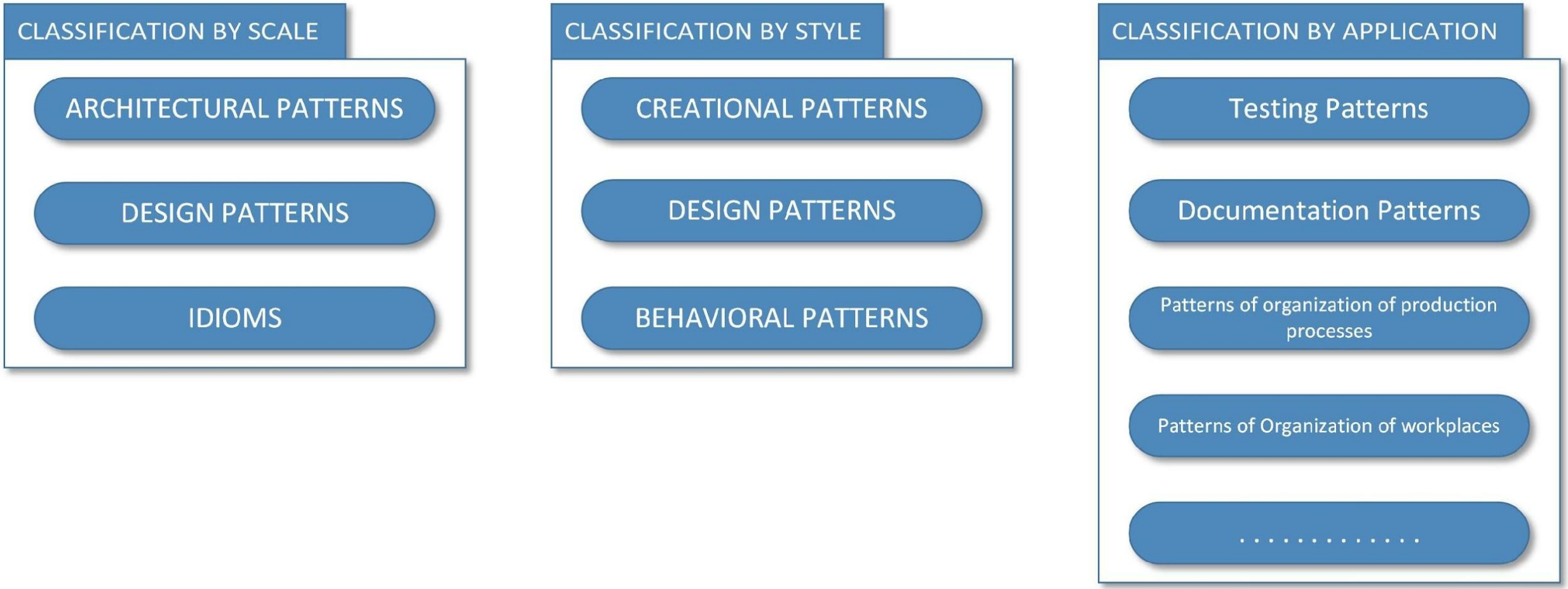
Паттерны суммируют опыт множества разработчиков и экспертов, делая его доступным рядовым разработчикам. Именование паттернов позволяют создать своего рода словарь, с помощью которого разработчики могут понять друг друга намного лучше.

Если в документации к системе указано, на основе каких паттернов она построена, это позволяет быстрее понять структуру системы.



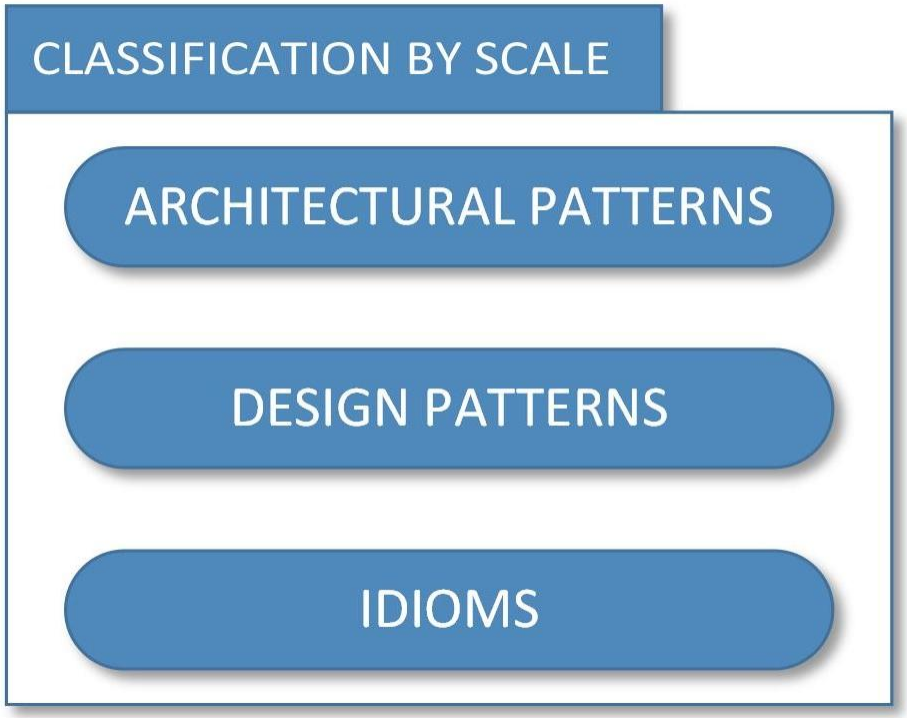
John Vlissides

PATTERNS CLASSIFICATION



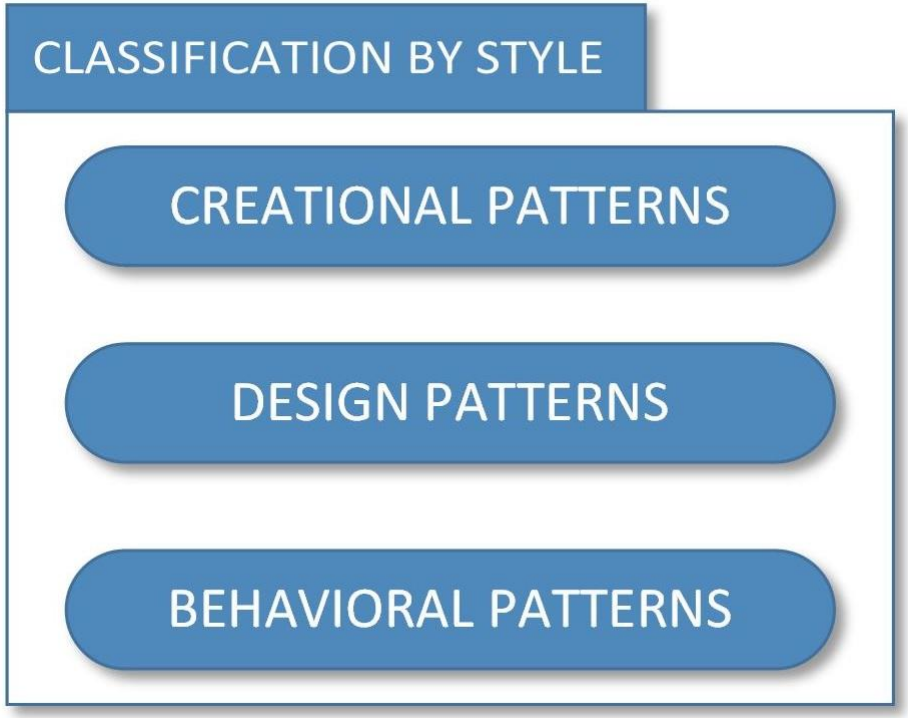
CLASSIFICATION BY SCALE

- *Архитектурные паттерны* — наивысший слой детализации, используются для описания структуры программы в целом.
- *Паттерны проектирования* — средний слой детализации, описывают компоненты отдельных архитектурных паттернов и реализацию их взаимодействия.
- *Идиомы* — низший слой детализации, описывают реализацию отдельных решений проблем применительно к конкретному языку программирования.



CLASSIFICATION BY STYLE

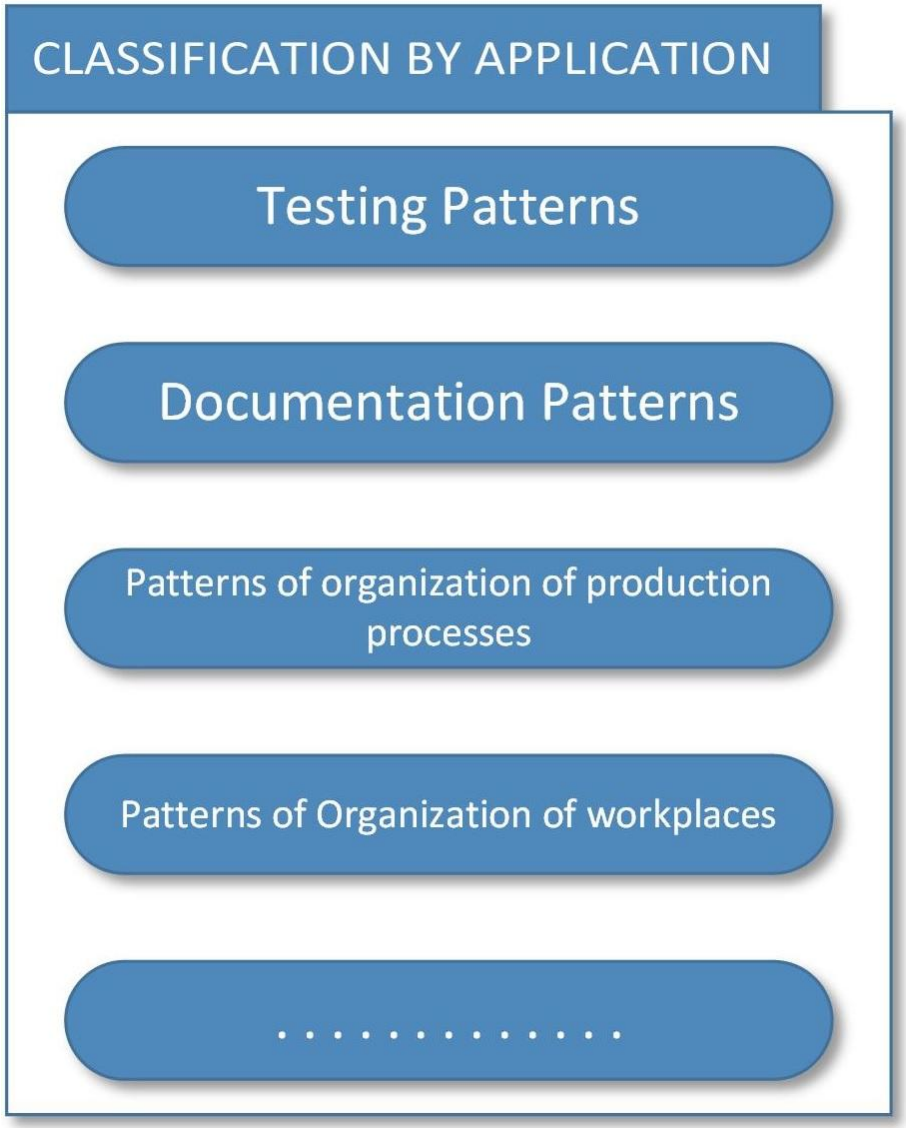
- *Порождающие паттерны* — предназначены для решения проблем создания новых объектов и связей.
- *Структурные паттерны* — предназначены для компоновки системы, при этом могут использовать различные механизмы, такие как *наследование, полиморфизм, композиция*.
- *Поведенческие паттерны*— предназначены для решения задач связи объектов и распределения задач между ними.



CLASSIFICATION BY APPLICATION

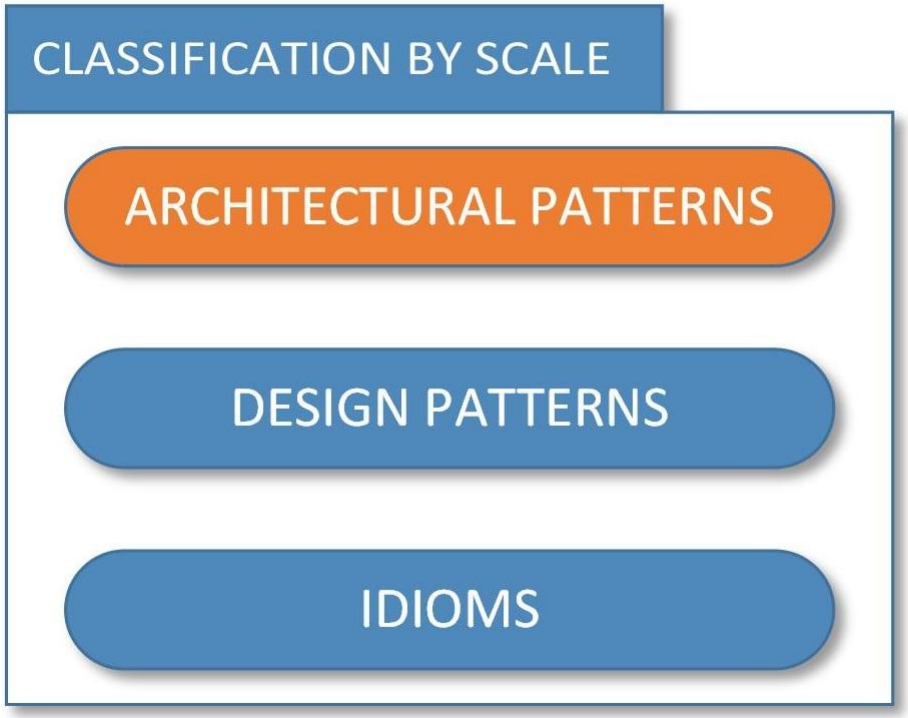
Программистам редко приходится сталкиваться с данным классом паттернов, но все же стоит о нем упомянуть, чтобы иметь хотя бы общее представление. Это самый высокоуровневый класс паттернов. В него входят целые классы паттернов. Например:

- *Паттерны тестирования*
- *Паттерны документирования*
- *Паттерны организации производственных процессов*
- *Паттерны организации рабочих мест*
- *И многие другие*

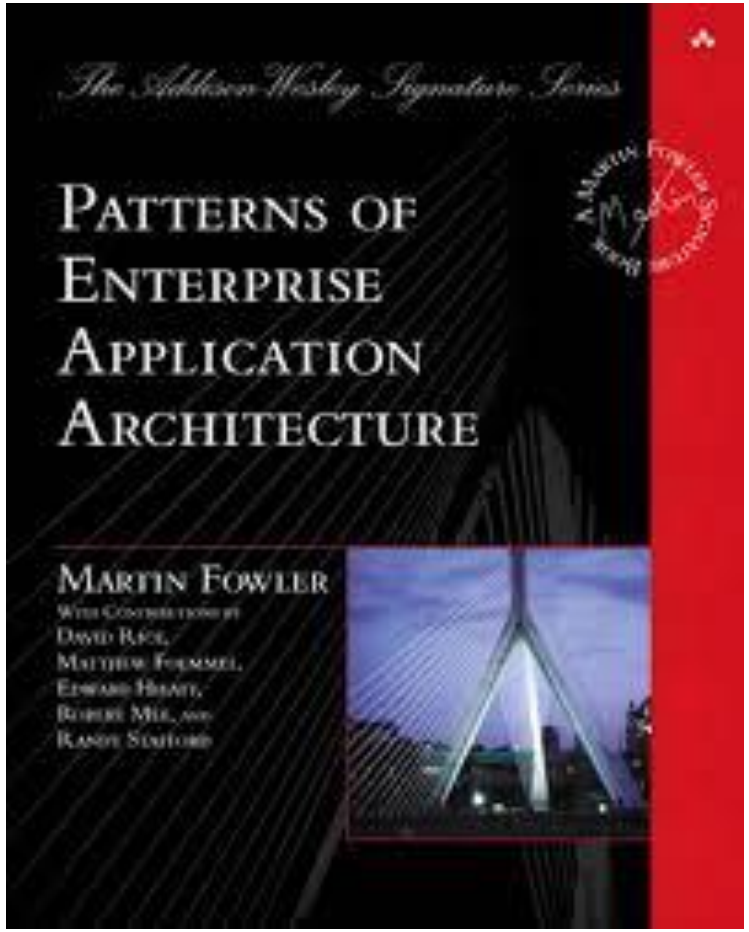


ARCHITECTURAL PATTERNS

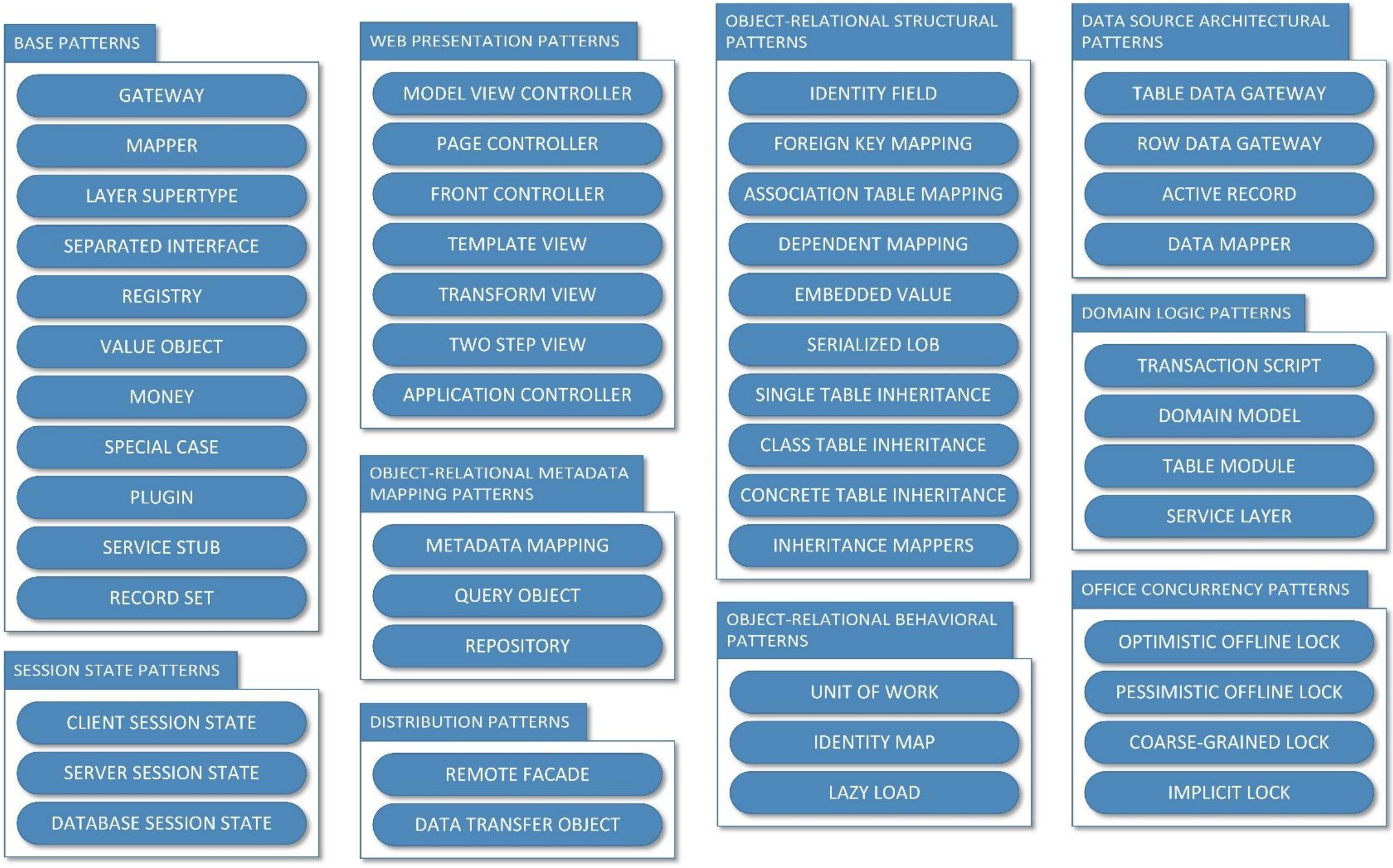
Архитектурные паттерны, являясь наиболее высокоуровневыми паттернами, описывают структурную схему программной системы в целом.



PATTERNS OF ENTERPRISE APPLICATION ARCHITECTURE

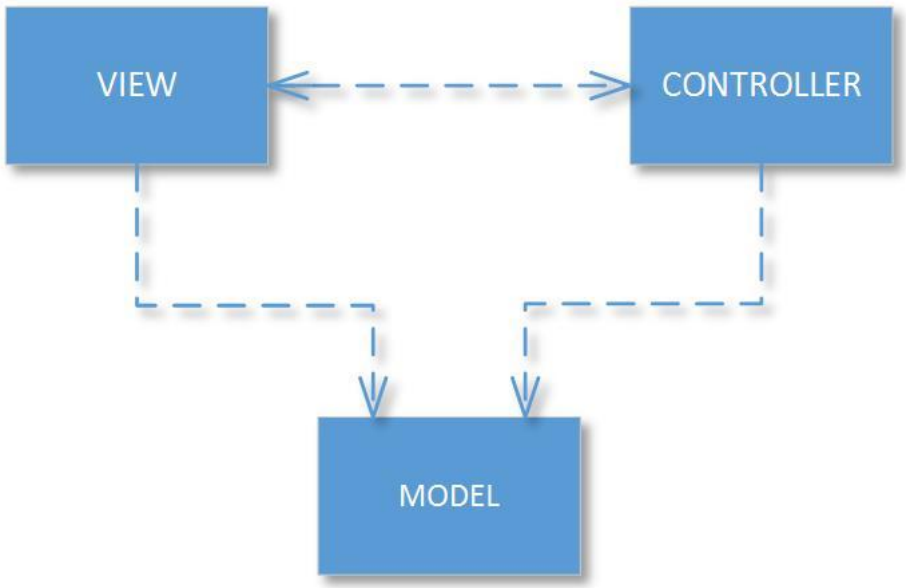


Martin Fowler

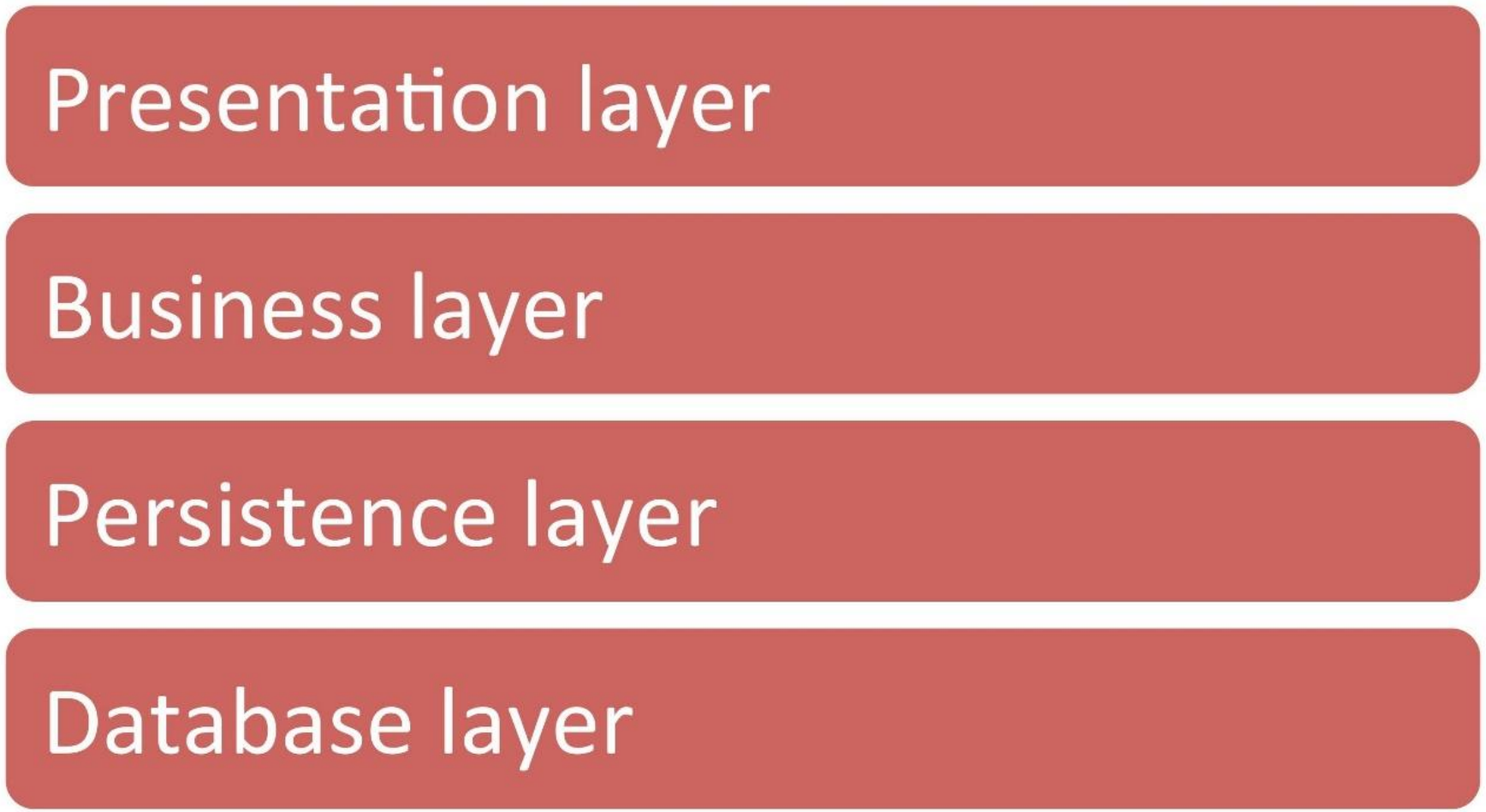

BASE PATTERNS
GATEWAY
MAPPER
LAYER SUPERTYPE
SEPARATED INTERFACE
REGISTRY
VALUE OBJECT
MONEY
SPECIAL CASE
PLUGIN
SERVICE STUB
RECORD SET
SESSION STATE PATTERNS
CLIENT SESSION STATE
SERVER SESSION STATE
DATABASE SESSION STATE
WEB PRESENTATION PATTERNS
MODEL VIEW CONTROLLER
PAGE CONTROLLER
FRONT CONTROLLER
TEMPLATE VIEW
TRANSFORM VIEW
TWO STEP VIEW
APPLICATION CONTROLLER
OBJECT-RELATIONAL METADATA MAPPING PATTERNS
METADATA MAPPING
QUERY OBJECT
REPOSITORY
DISTRIBUTION PATTERNS
REMOTE FACADE
DATA TRANSFER OBJECT
OBJECT-RELATIONAL STRUCTURAL PATTERNS
IDENTITY FIELD
FOREIGN KEY MAPPING
ASSOCIATION TABLE MAPPING
DEPENDENT MAPPING
EMBEDDED VALUE
SERIALIZED LOB
SINGLE TABLE INHERITANCE
CLASS TABLE INHERITANCE
CONCRETE TABLE INHERITANCE
INHERITANCE MAPPERS
OBJECT-RELATIONAL BEHAVIORAL PATTERNS
UNIT OF WORK
IDENTITY MAP
LAZY LOAD
DATA SOURCE ARCHITECTURAL PATTERNS
TABLE DATA GATEWAY
ROW DATA GATEWAY
ACTIVE RECORD
DATA MAPPER
DOMAIN LOGIC PATTERNS
TRANSACTION SCRIPT
DOMAIN MODEL
TABLE MODULE
SERVICE LAYER
OFFICE CONCURRENCY PATTERNS
OPTIMISTIC OFFLINE LOCK
PESSIMISTIC OFFLINE LOCK
COARSE-GRAINED LOCK
IMPLICIT LOCK

MODEL VIEW CONTROLLER (MVC)

- *Модель (Model)* представляет собой данные, с которыми оперирует приложение.
- *Вид (View)* представляет собой компонент системы для отображения состояния модели в понятном человеку представлении.
- *Контроллер (Controller)* является средством, при помощи которого пользователи взаимодействуют с системой.

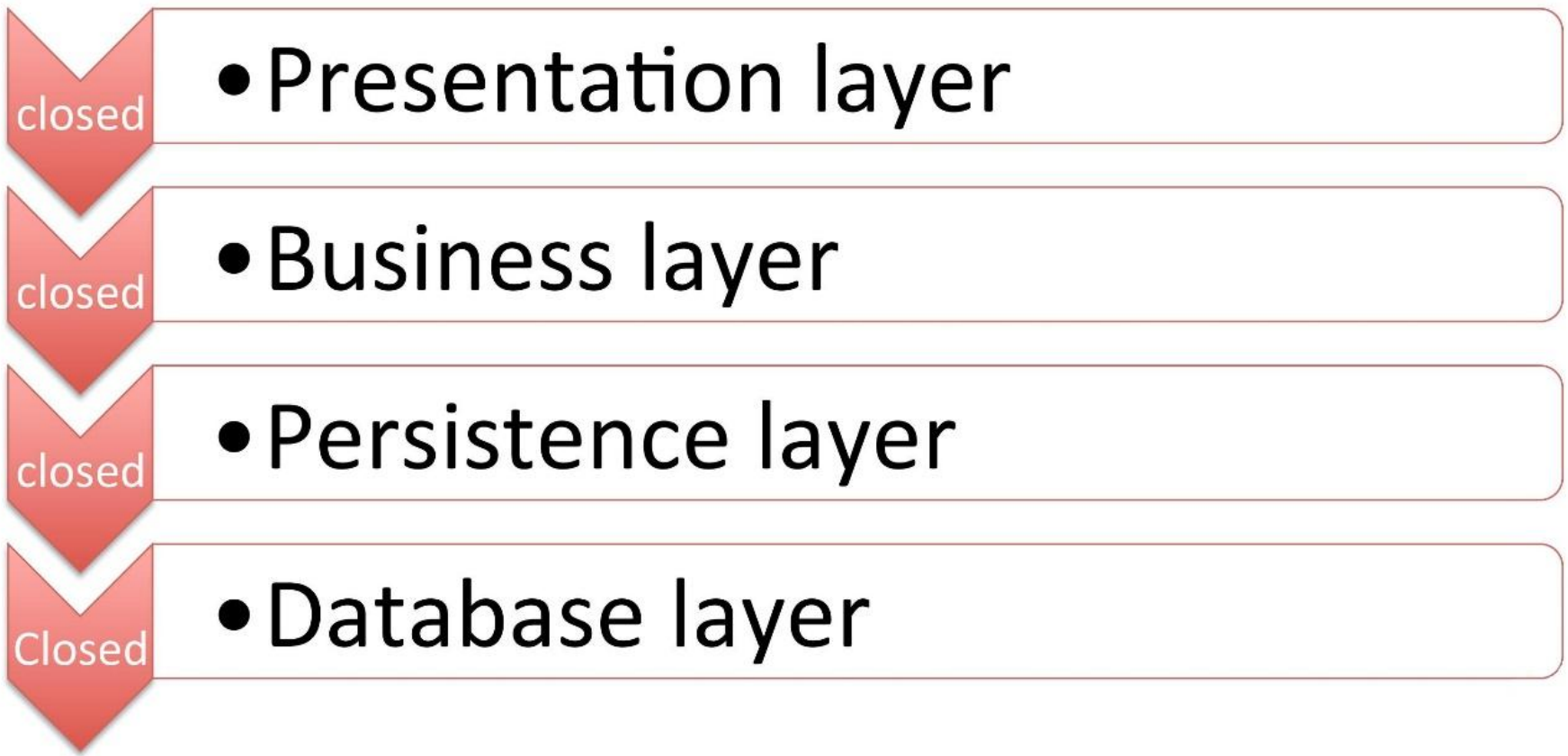


Layered architecture



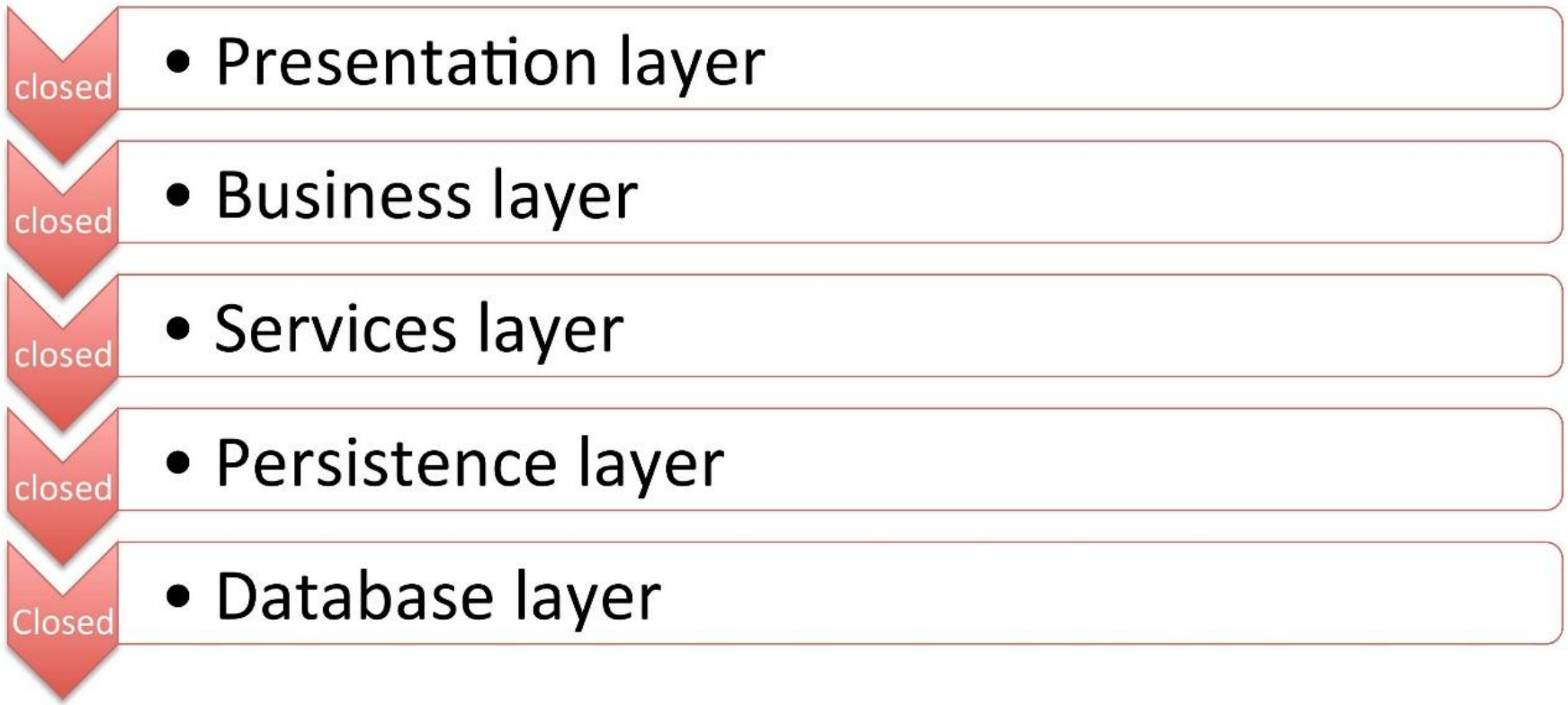
Layered architecture

- Layers communicate from top to down only
- To get layer below, you have to go through all in the middle



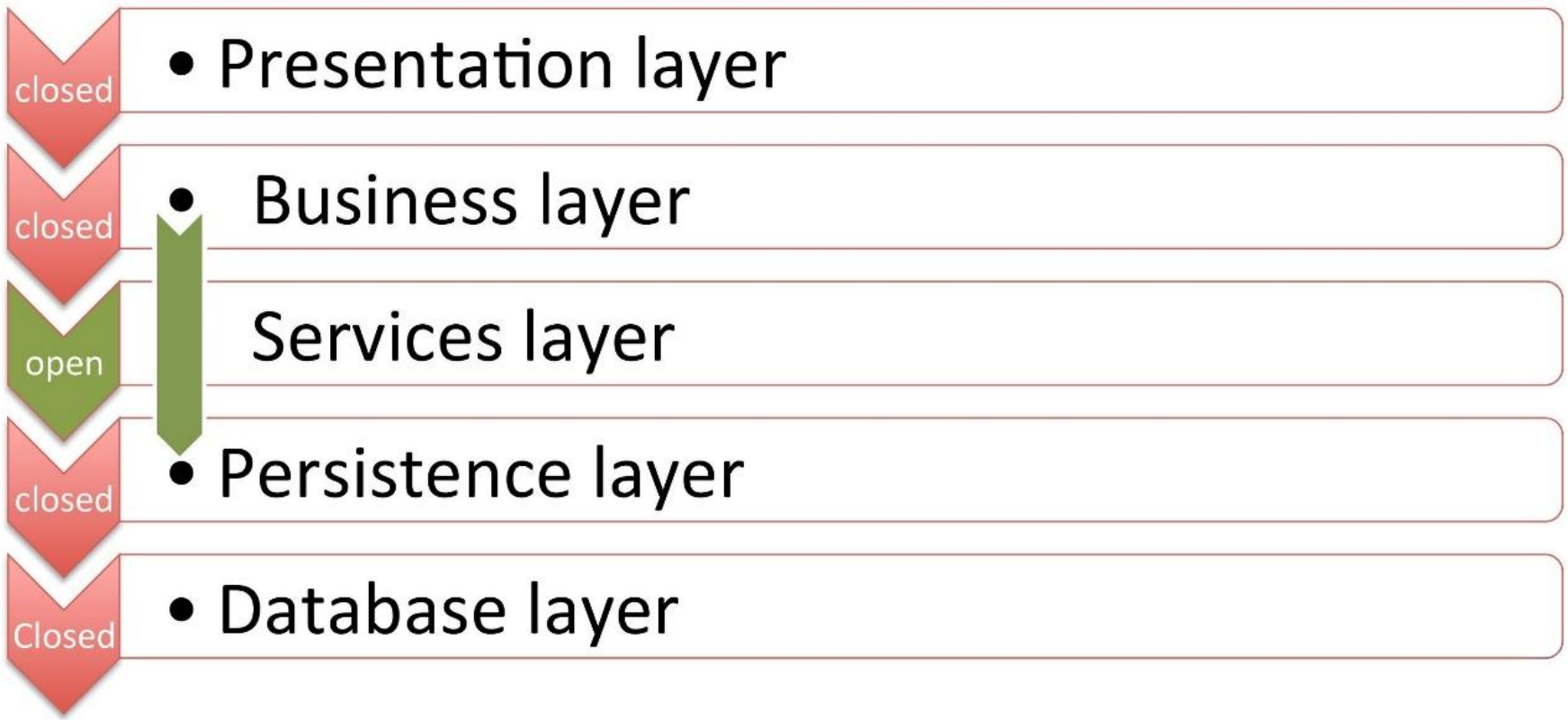
What if we have some kind of shared services?

Do we still need pass all request throw this layer?



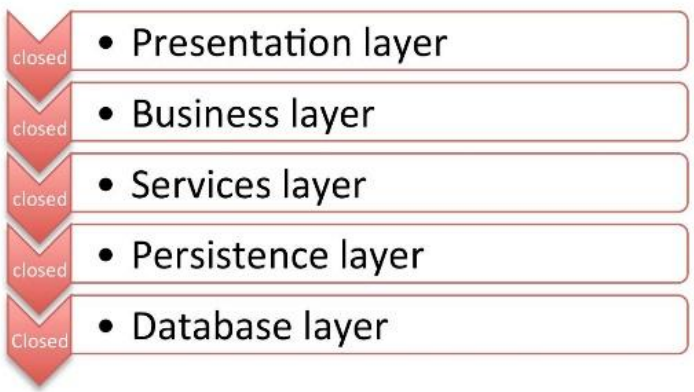
Open layered architecture

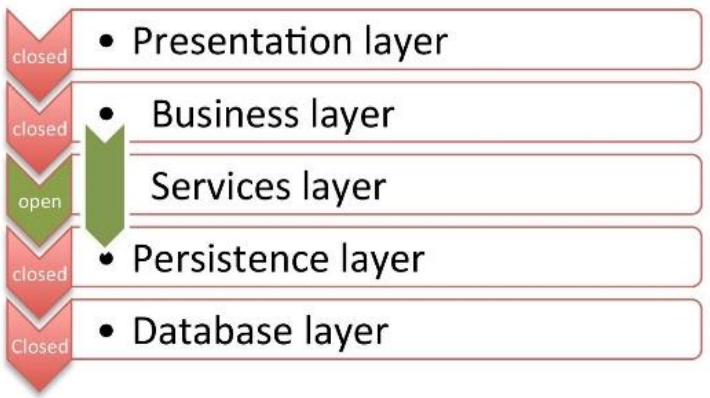
Some layers might be open.



Layered architecture

- Good general purpose architecture
- Easy to implement, test, and govern
- Good starting point for most systems
- Not always optimized for specific business drivers

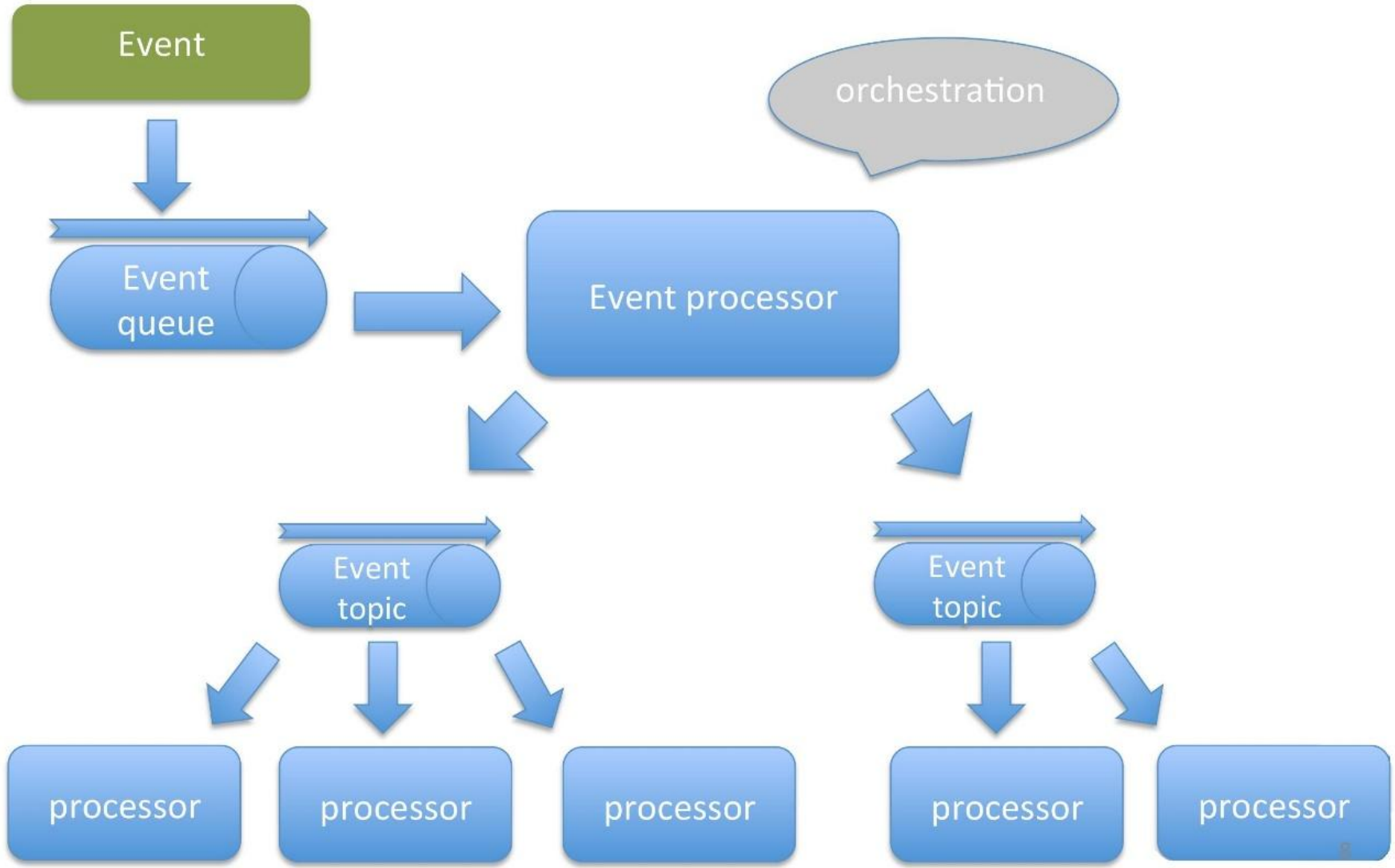




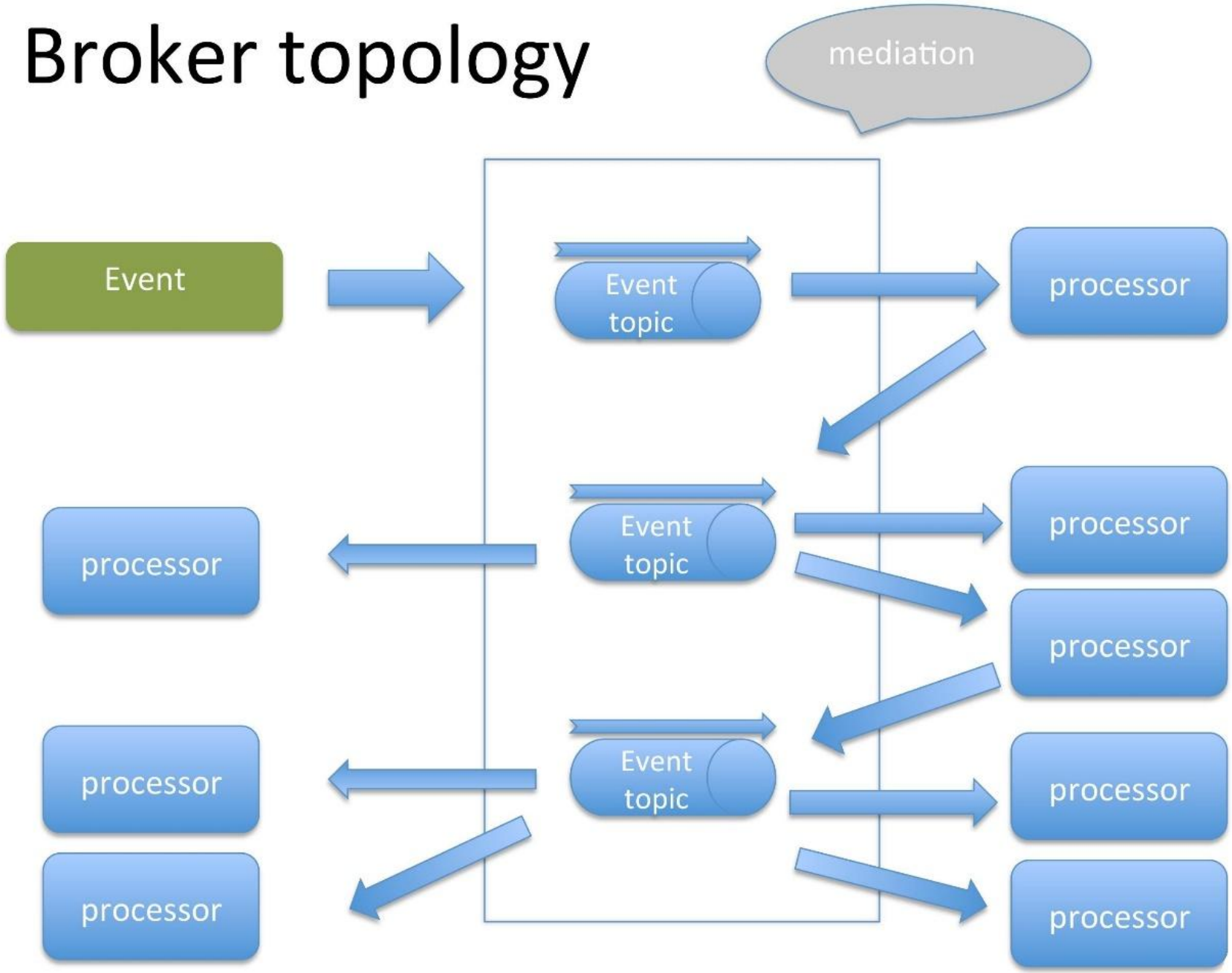
Event-driven architecture

- Event processor topology
- Broker topology
- Broker-less topology

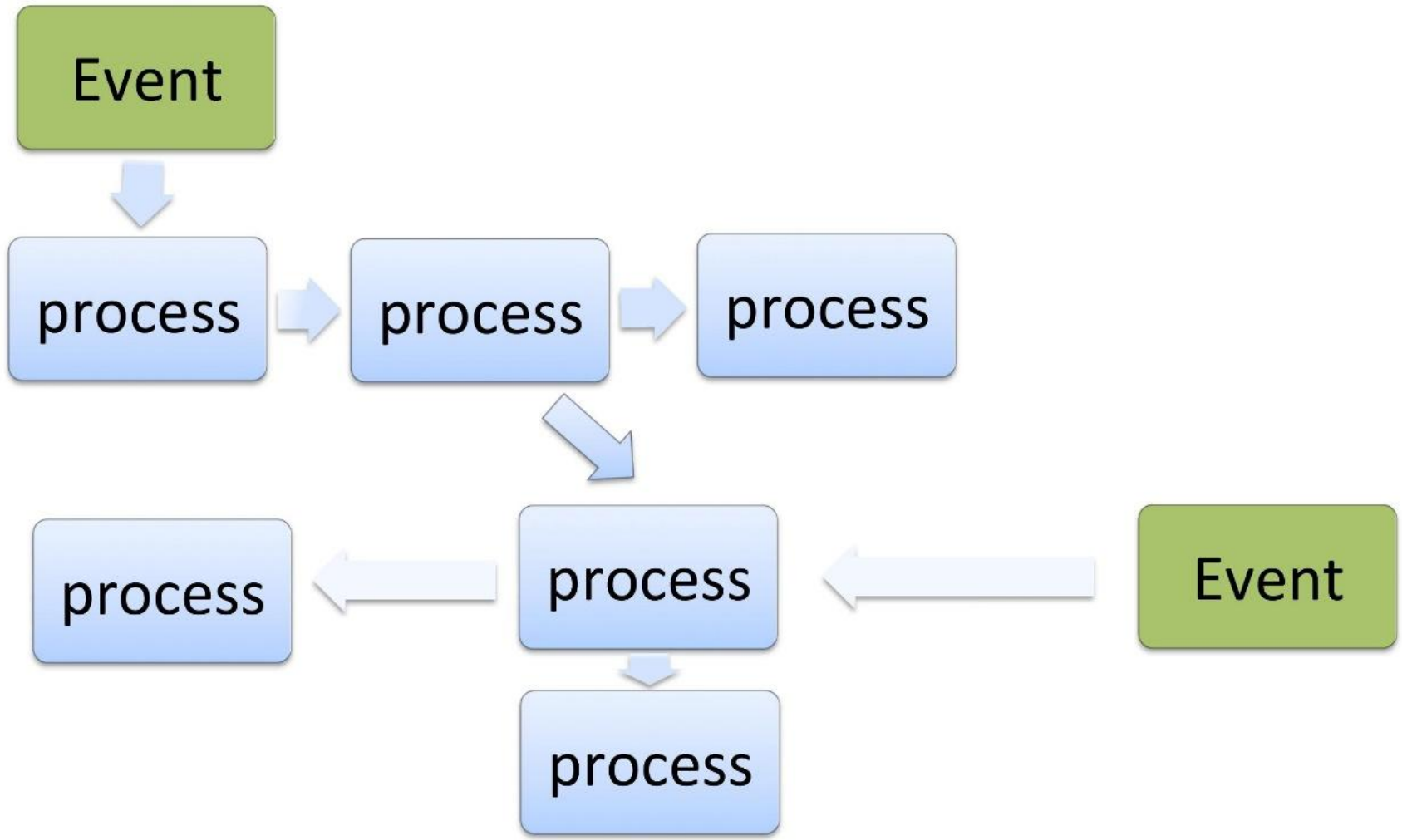
Event processor topology



Broker topology



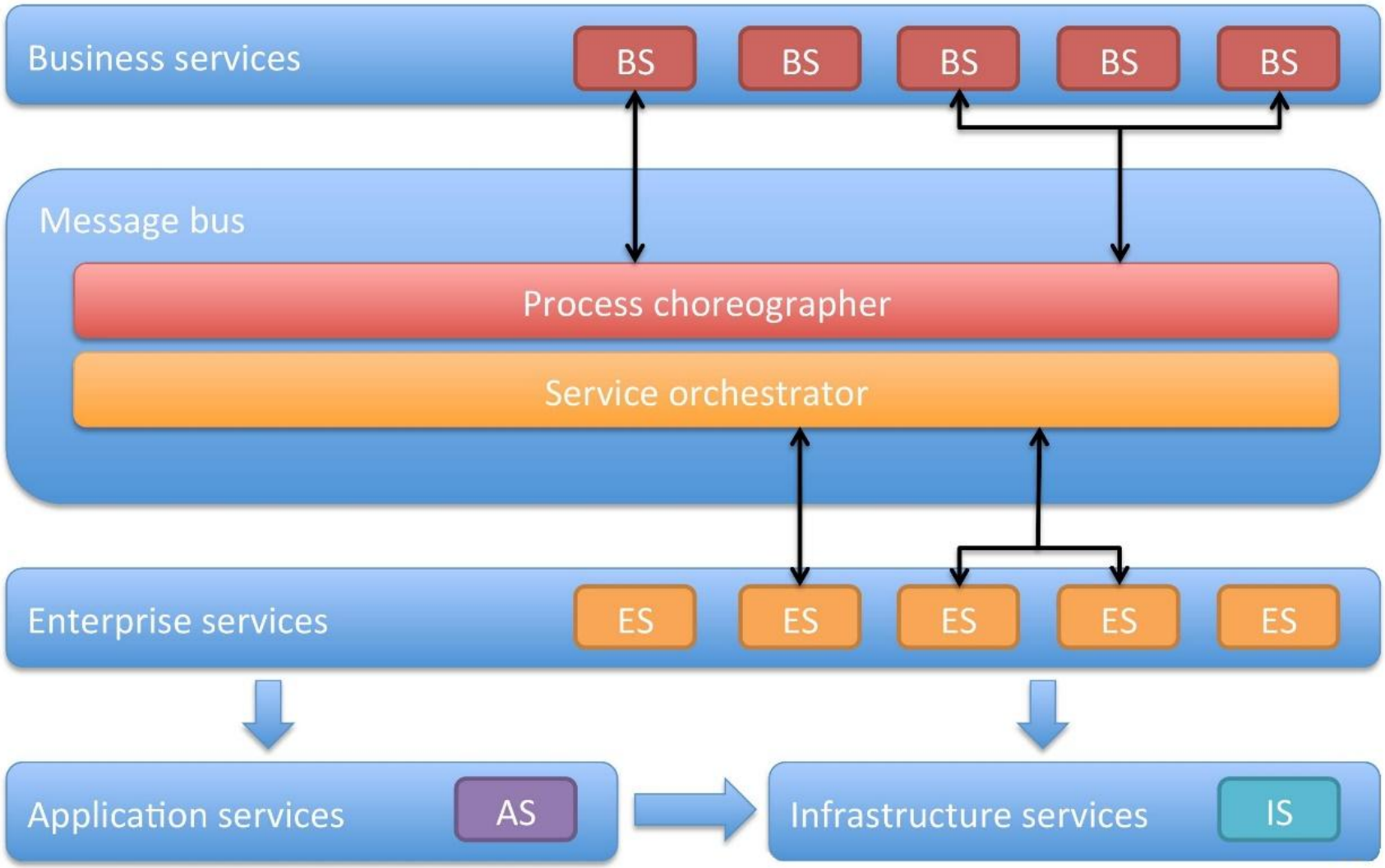
Broker-less topology



Event-driven architecture

- Highly decoupled and distributed
- Highly scalable
- High degree of complexity
- Good for event-based business models and business processes
- Not good for processes which require a high degree of data sharing, orchestration, and reuse

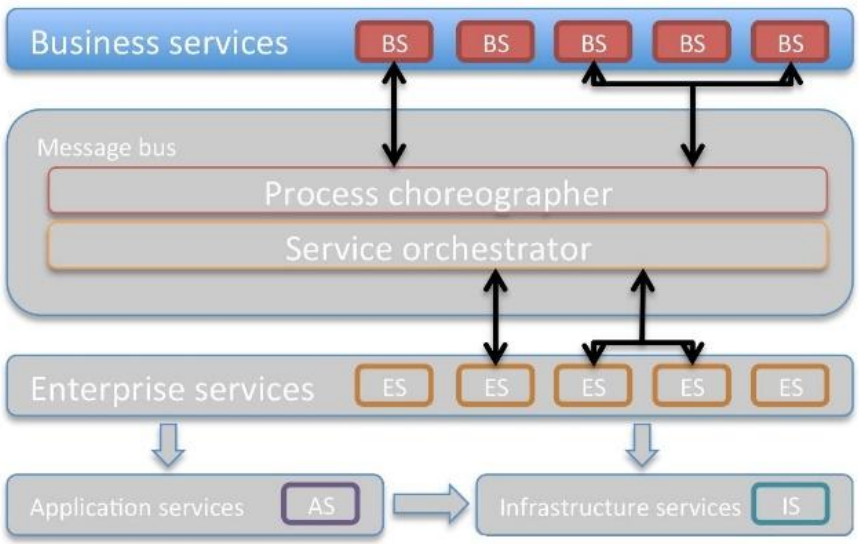
Service-oriented architecture



Business services

Abstract service used to represent a business process or function independent of the underlying technology or pattern

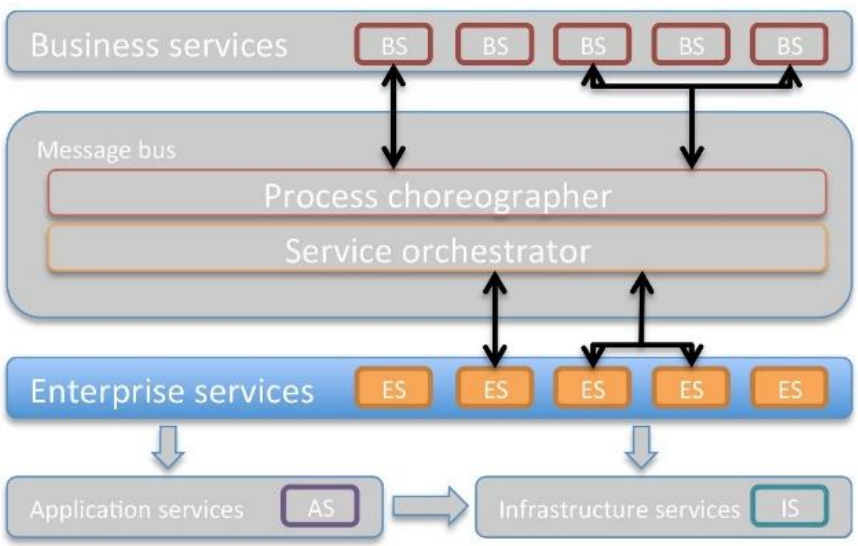
- Can be derived from use cases, user stories, user scenarios
- Contains a service name, input specification, and output specification
- Course-grained
- Shared across the enterprise



Enterprise services

Concrete services that implement Business Services

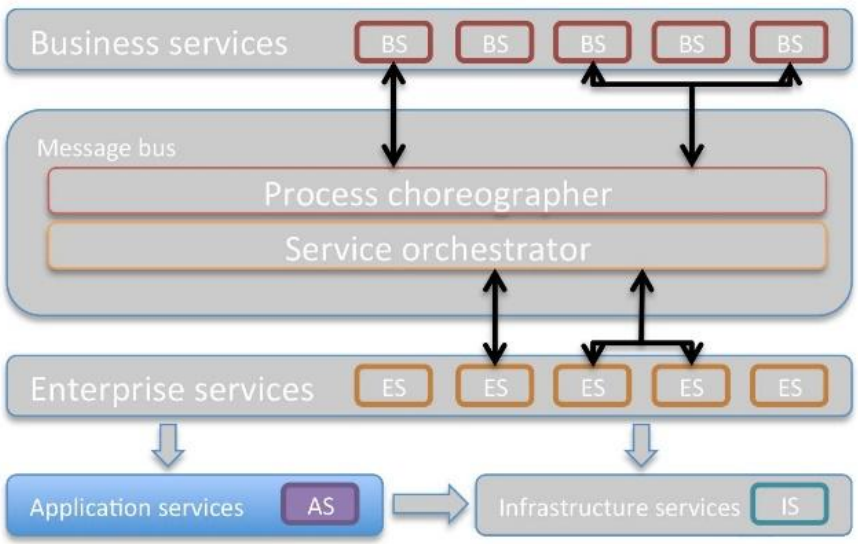
- The relationship between an Enterprise Service and a Business Service is either a one-to-one or many-to-one relationship
- Course-grained
- Represent actions against major data entities
- Usually require some sort of service orchestration
- Shared across the enterprise



Application services

Implementation of application-specific functions, such as database querying, validation, etc.

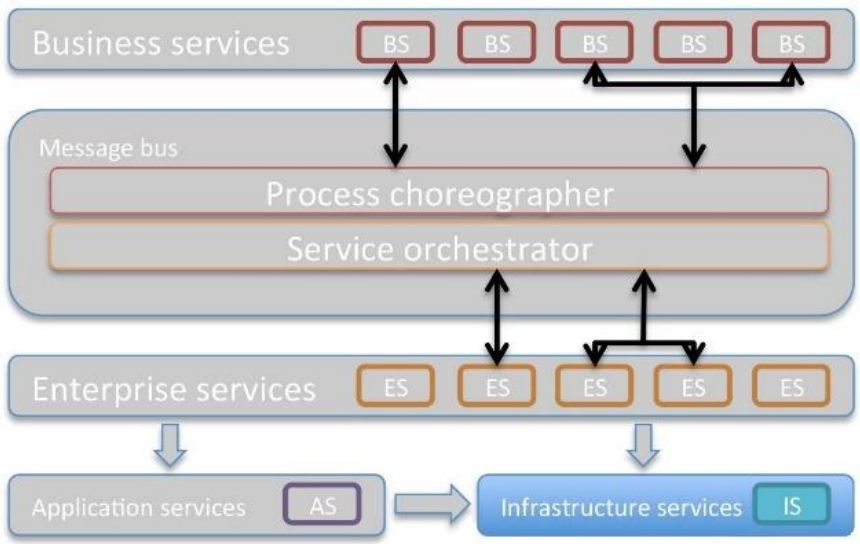
- Concrete definition
- Defined by application developers
- Fine-grained
- Tightly bound to a specific application context
- Generally not shared across the enterprise



Infrastructure services

Implementation of the non-business related functions, like logging, error handling, single sign on, etc.

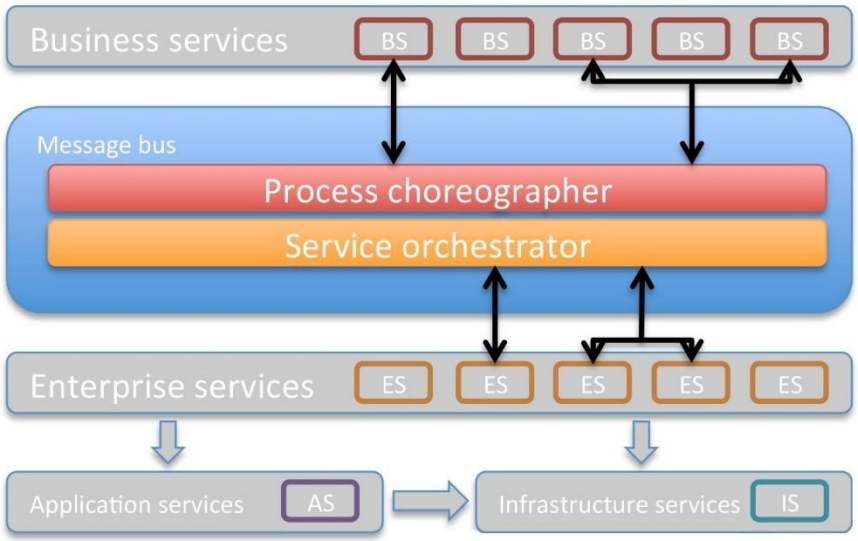
- Concrete definition
- Defined by application or system developers
- Fine-grained
- Supports the system or enterprise infrastructure
- Shared across the enterprise



Message Bus

Coordinates services and processes, it's a glue for SOA components

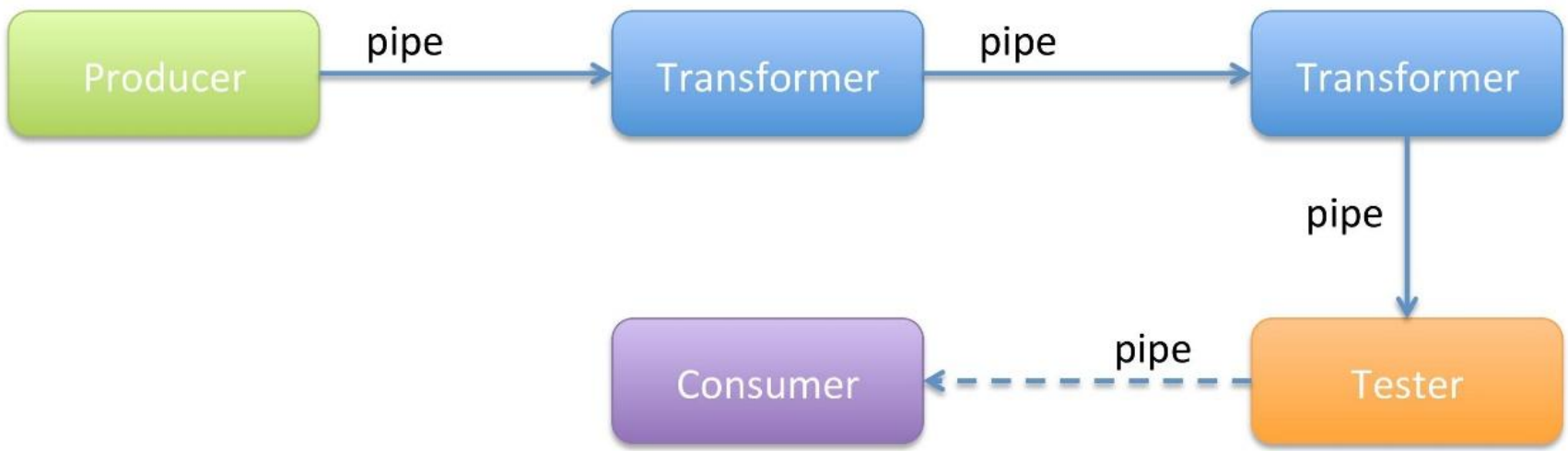
- Process choreography
- Service orchestration
- Service registry
- Protocol transformation
- Message enhancement and transformation



Service-oriented architecture

- Good pattern for understanding and implementing business processes and services
- Very high level of complexity
- Difficult to implement due to complex tools, hype, misconceptions, and heavy business user involvement
- Good pattern for large, complex, heterogeneous businesses that have a large number of common services

Pipeline architecture

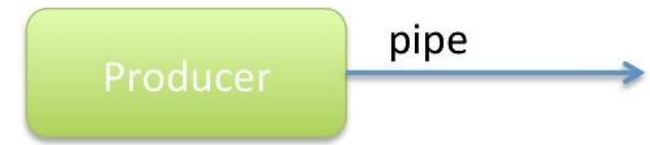


Pipes and filters

- Uni-directional only
- Usually point-to-point for high performance, but could be message-based for scalability
- Payload can be any type

- Self-contained and independent from other filters
- Usually designed to perform a single specific task

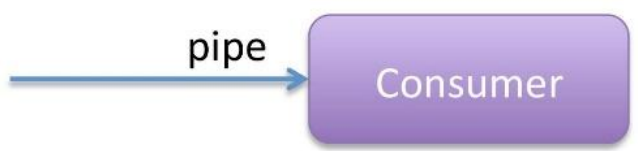
Filter types



Starting point, outbound only

Input, processing, output

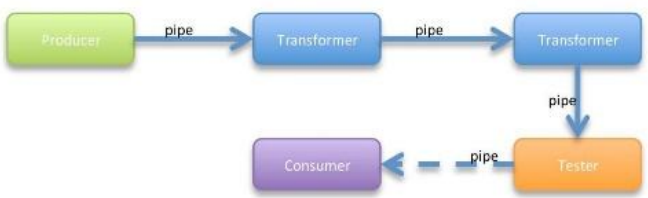
Input, discard or pass-thru



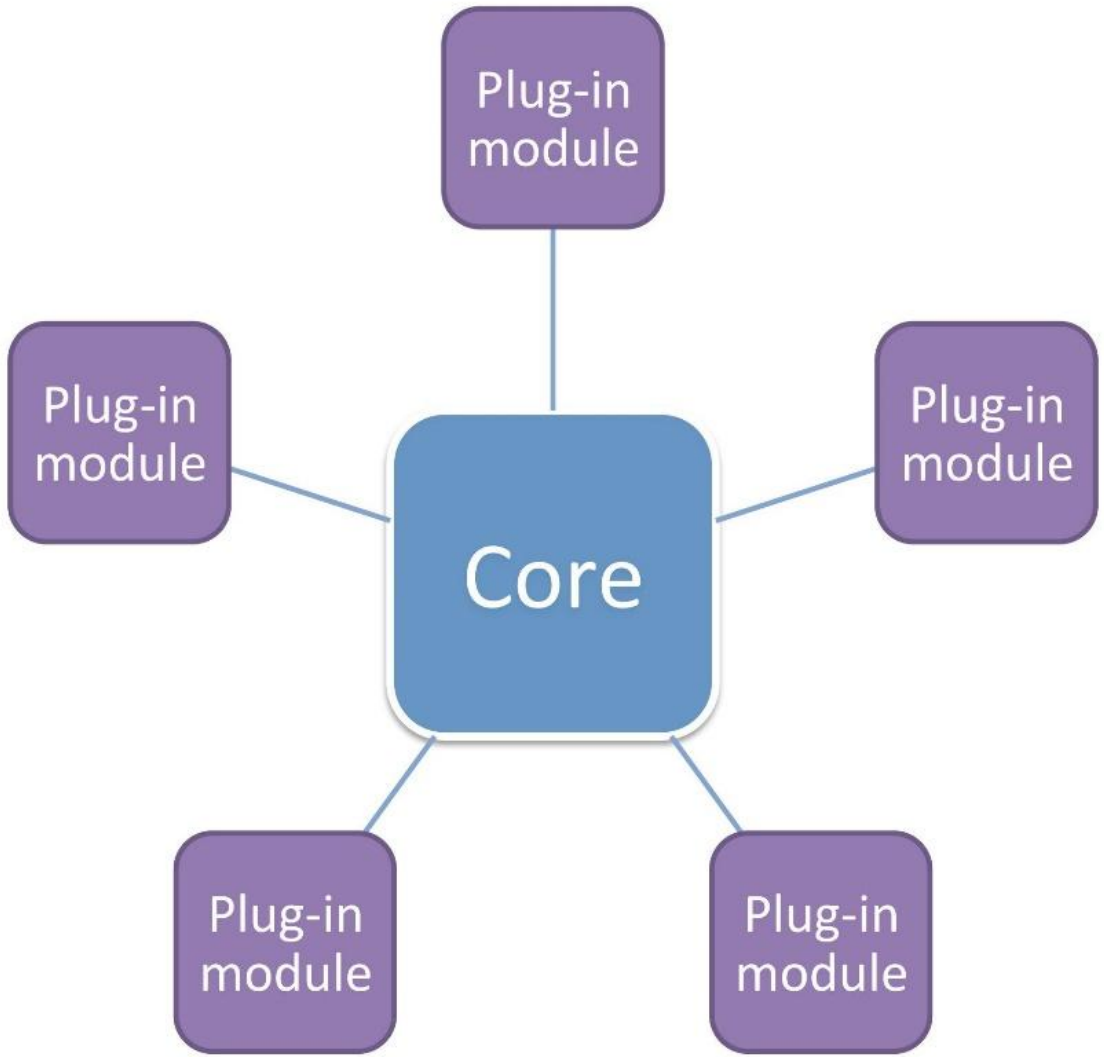
Endpoint, inbound only

Pipeline architecture

- Useful for smaller deterministic systems with a distinct processing flow
- Filters can easily be added and removed
- Provides for a high level of decoupling
- Supports evolutionary design
- Able to easily adapt to changing requirements
- Can be easily incorporated into another pattern



Microkernel architecture



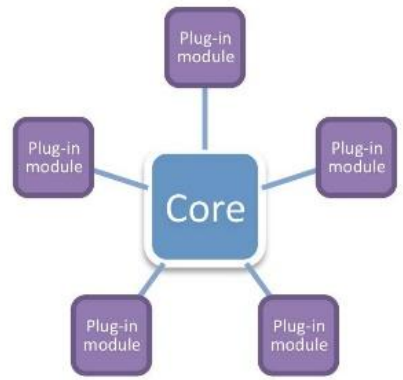
Microkernel architecture

Core

- Minimal functionality to run system
- General business rules and logic
- Doesn’t contain custom processing

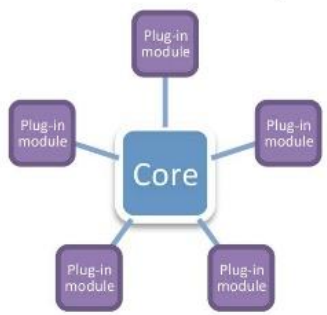
Plug-in module

- Standalone independent module
- Specific additional rules or logic

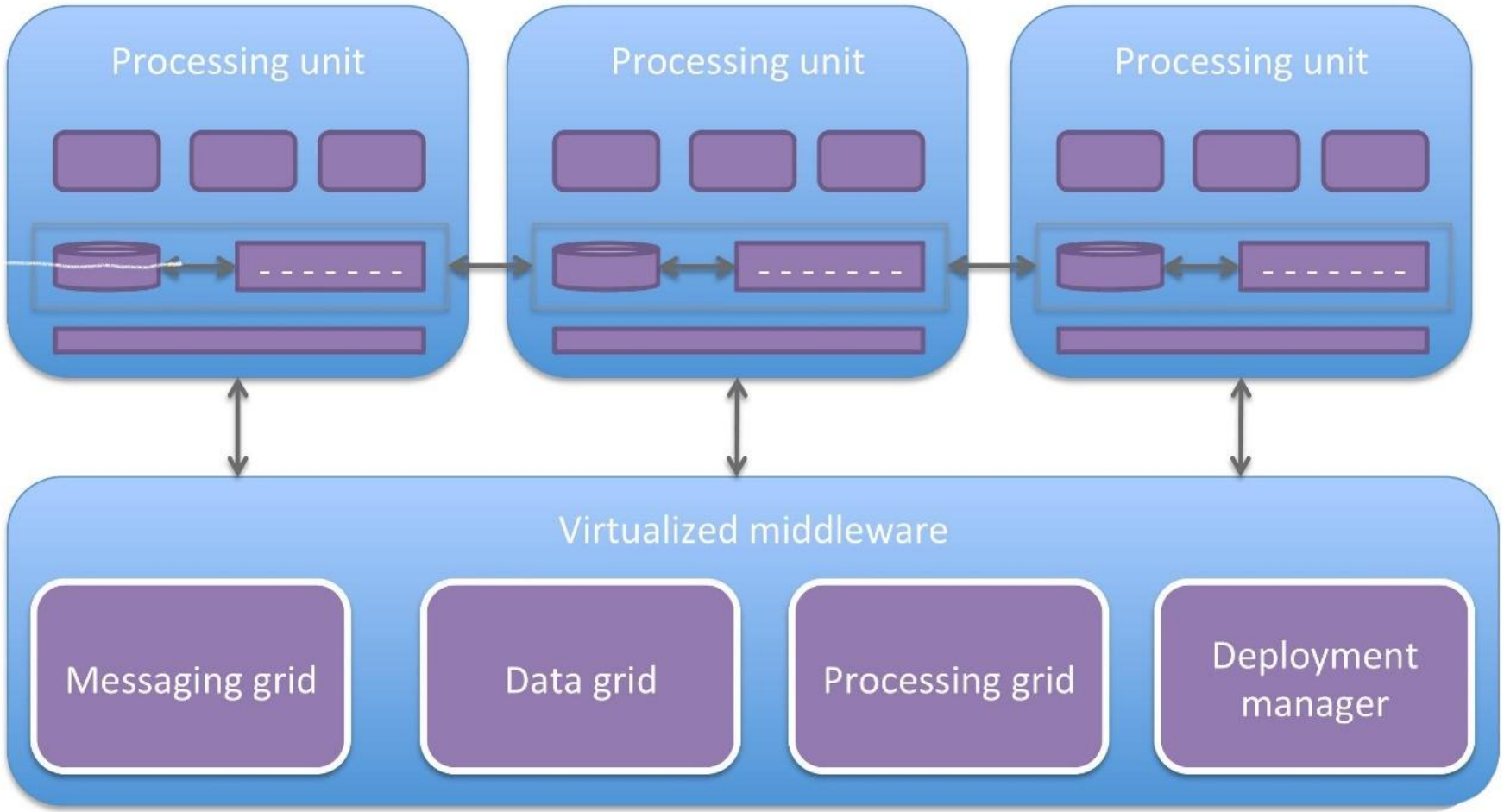


Microkernel architecture

- Useful for systems that have custom processing or processing is susceptible to change
- Plug-in modules can be easily added and removed
- Supports evolutionary design
- Able to easily adapt to changing requirements
- Can easily be incorporated into another pattern

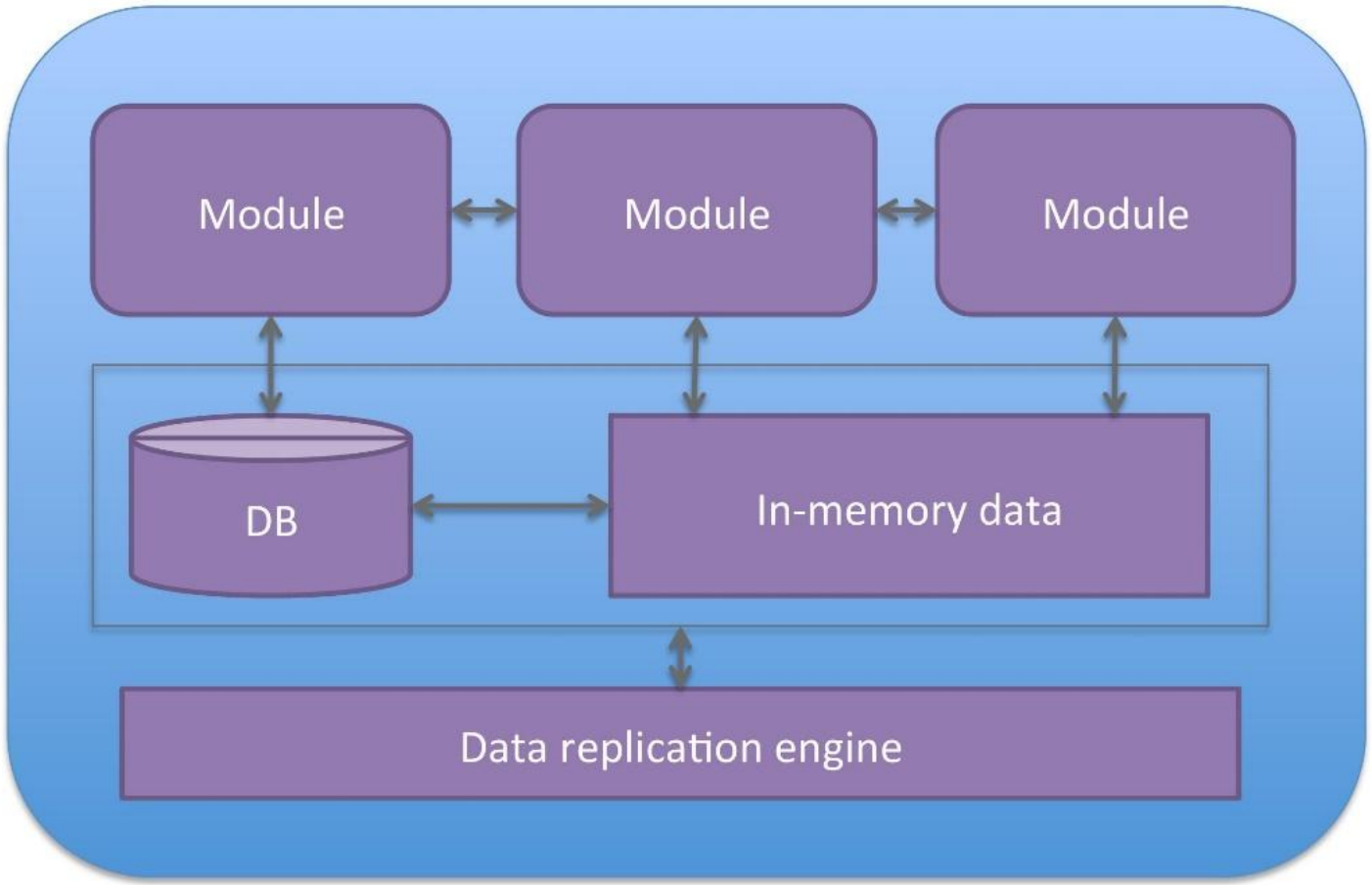


Space-based architecture

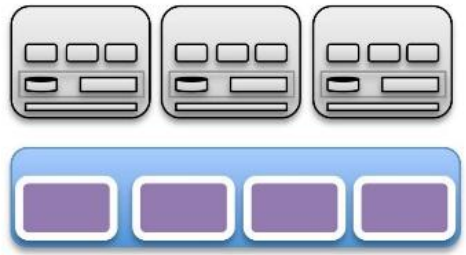


Processing unit

In fact it is standalone version of yours application



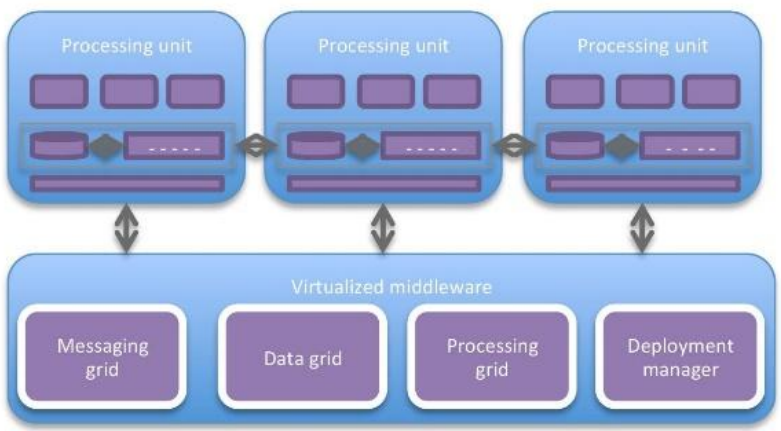
Virtualized middleware



Messaging grid	Manages input request and session
Data grid	Manages data replication between processing units
Processing grid	Manages parallel request processing
Deployment manager	Manages dynamic processing unit deployment

Space-based architecture

- Good for applications that have variable load or inconsistent peak times
- Not good fit for traditional large-scale relational database systems
- Relatively complex and expensive pattern to implement



LIST OF SOURCES

- https://laravel.ru/posts/3#uw3-%D0%B2%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-4
- http://citforum.ck.ua/SE/project/pattern/
- **Patterns of Enterprise Application Architecture**, Martin Fowler
- MVC // http://design-pattern.ru/patterns/mvc.html
- MVC // http://www.berdaflex.com/ru/eclipse/books/rcp_filemanager/ch04s06.html


THANK YOU FOR ATTENTION