# ARCHITECTURAL STYLE

DR. VIVEK NALLUR

VIVEK.NALLUR@SCSS.TCD.IE

https://www.scss.tcd.ie/vivek.nallur/teaching/slides/

# OUTLINE OF THIS TALK

- Architectural Styles
  - What makes up a software architectural style?
  - Some styles and their properties

## Corinthian column

## A Shinto Gate

# WHAT IS AN ARCHITECTURAL STYLE?

> *A family of systems in terms of a pattern of structural organization. More specifically, a style determines the vocabulary of components and connectors that can be combined.*

David Garlan and Mary Shaw, *An Introduction to Software Architecture, 1996*

## WHY STYLES?

- Common language
- Technology agnostic
- Inclusive of patterns and principles

## KEY PRINCIPLES OF ALL STYLES

*Separation-of-concerns*: Functionality should not be spread amongst components unnecessarily

*Single Responsibility*: Components should do only one thing, and do it well

*Principle of least knowledge*: Talk to the interface!

## CATEGORIZATION

| Category | Architectural Styles |
| --- | --- |
| Communication | Service-Oriented, Message Bus |
| Deployment | Client/Server, N-Tier, 3-Tier |
| Domain | Domain driven design |
| Structure | Component-based, Object-oriented, Layered |

# STYLE: CLIENT/SERVER

Describes a distributed system that involves a client, and a server

Simplest form has a server application accessed directly by multiple clients

E.g., Email-clients, git clients, database query tools

# CLIENT/SERVER: VARIATIONS

- *Client-Queue-Client*: Server acts as a passive queue, which clients access
- *Peer-to-Peer*: Developed from CQC, P2P allows clients and servers to swap roles to distribute and synchronize files
- *Application servers*: Server hosts and executes applications, while clients access them through a UI

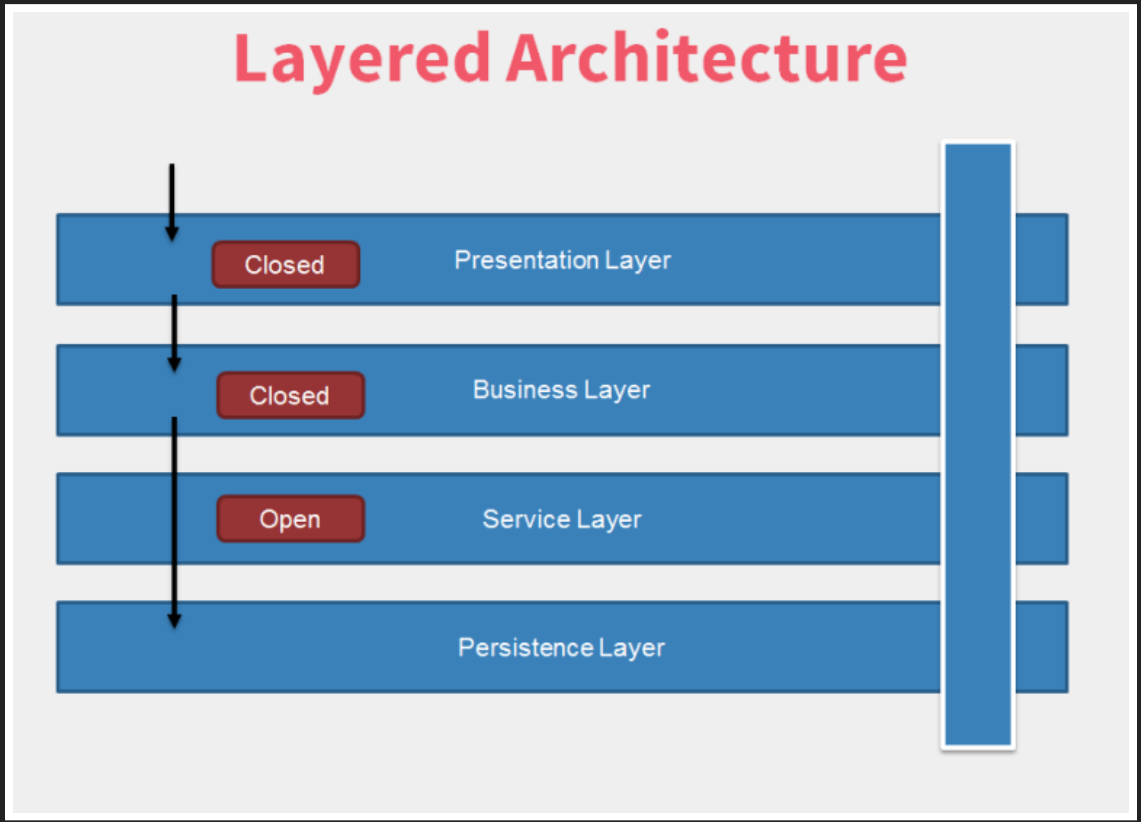
# CLIENT/SERVER: PROS AND CONS

*Security*: Since server has all the data, easier to secure
*Centralized Data store*: Access and updates to data are propagated to all clients
*Maintenance*: Client and server can generally be maintained independently, as long as access protocol remains the same

*Extensibility*: Dependence on a single server ties the application and data to a central server
*Reliability*: Central server introduces *single-point-of-failure*

# STYLE: LAYERED

Group related functionality into distinct layers, and stack layers on top of each other. Components in one layer can only interact with other components in the same layer, or in the layer below it



# LAYERED: PROPERTIES

- Abstraction
- Encapsulation
- High Cohesion and Loose Coupling
- Reusable

Typical applications are: accounting systems, web-based applications, operating systems

# LAYERED: PROS/CONS

| Pros | |
|---|---|
| *Abstraction* | Complexity is easier to handle when broken down into simpler chunks |
| *Maintainability* | Allows *separation-of-concerns* |
| *Testability* | Different components can be tested independently |

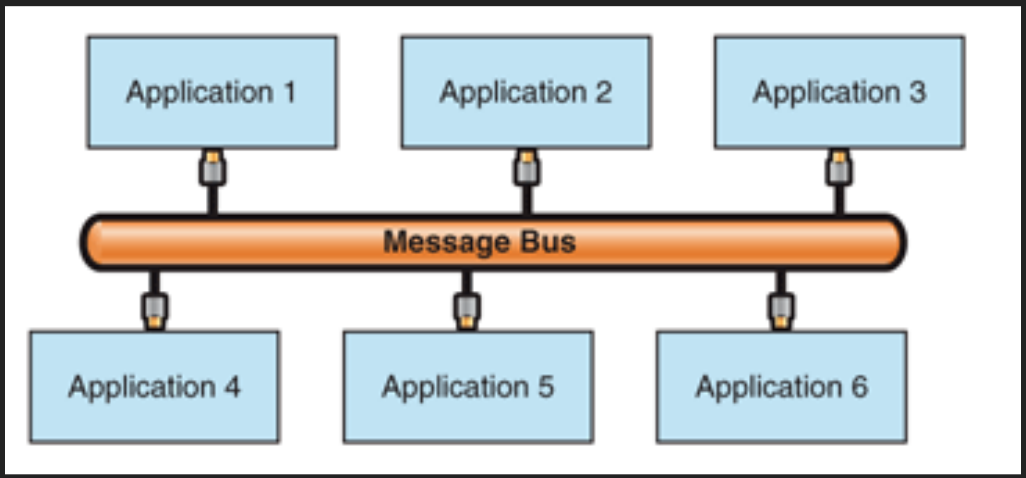
# LAYERED: PROS/CONS

| Cons | |
|---|---|
| *Performance* | Overhead of calls percolating through the layers |
| *Unnecessary Complexity* | Ease of understanding leads to frequency of mis-use |

# STYLE: MESSAGE BUS

A message based approach to creating an application using a channel to centralize communication

Allows a system to be built of interacting components that know nothing about each other, except for the interface of the bus

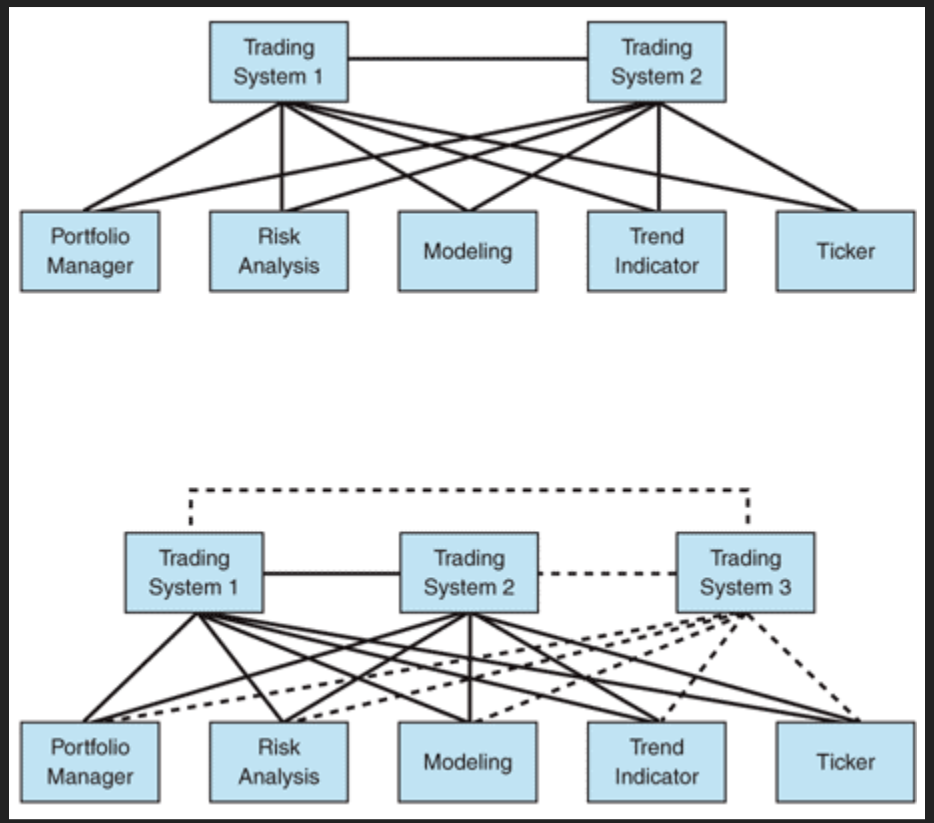


# MESSAGE BUS: PROPERTIES

- Decoupling of component-functionality
- Modifiability of components, independent of each other
- Scalability of the system can be increased by simply increasing the capacity of the bus

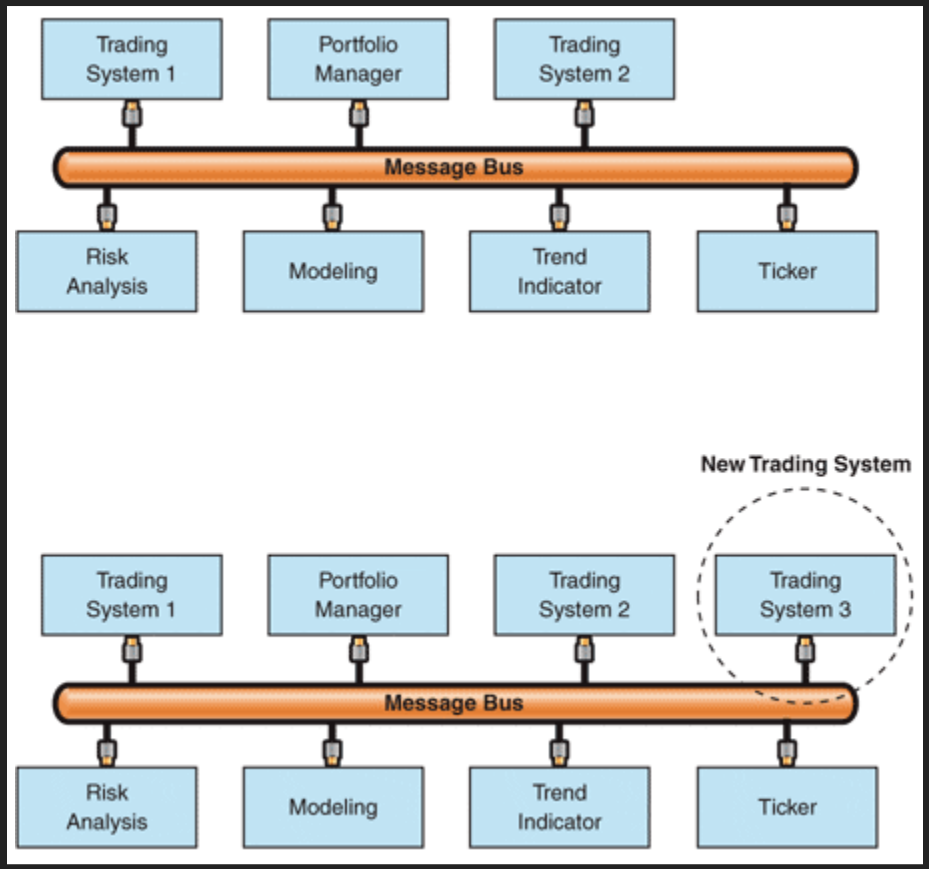
Typical applications: Financial/Trading applications, event-handling in operating systems

# MESSAGE BUS: TRADING APPLICATION




*Typical communication requirements*

# MESSAGE BUS: TRADING APPLICATION




*Solved using a message bus*

# MESSAGE BUS: PROS/CONS

| Pros | |
|---|---|
| *Simplicity* | Connecting diverse components is exceedingly simple |
| *Performance* | System can process messages as fast as the bus can deliver them |

# MESSAGE BUS: PROS/CONS

| Cons | |
|---|---|
| *Security* | A message passed on the bus is visible to everybody |
| *Single-point-of-failure* | If the bus fails, everything fails |

## THAT'S ALL, FOLKS!

---

Questions? Comments?