CSL7090
Software & Data Engineering

Lecture Slides

July-Nov 2024

Google Classroom Code: r5buai5

Course Content

Unit-1: Cloud Computing and Virtualization

Unit-2: Data Management

Unit-3: Data Intensive Processing Systems

Course Content

Unit-1: Cloud Computing and Virtualization

Basics of complex software design: Concept of modular software, microservices, communication, 4+1 architectural views and patterns, Cloud Computing: Architecture of cluster computing, design of data centers, open data center platforms, fault-tolerant system design, Virtualization: Type-1 and Type-2 virtualization, virtual machine, containers, dockers

Course Content

Unit-2: Data Management

Data Management: Structured data, relational database management, unstructured data, semi-structured data, Nosql database management (mongodb), column database, graph database, XML, JSON, HDFS, Handling drift in data, sensor data reliability at software and algorithmic level, sensor data analysis techniques

Course Content

Unit-3: Data Intensive Processing Systems

Data Intensive Processing Systems: Architecture of large scale data processing systems, Hadoop, Apache Spark, Storm, parallel data processing concepts such as map-reduce, directed acyclic graph, resilient distributed datasets, dynamic resource allocation, partial & shared computation, storage architecture

Grading Policy

Mini Project: 10% (maximum group of 3)

4 Quizzes: 10% (individual)

Major Project: 20% (maximum group of 3)

Midterm Exam: 20% (individual)

Final Exam: 40% (individual)

Attendance: Attendance will be taken and submitted to ERP for records.

Mini Project

Pick any topic from the course content

Implement a system demonstrating the concepts discussed in the class

Record a presentation on the work done

Submit video link on the google classroom

Deadline for submission: 30th September 2024

Major Project

Pick any topic from the course content

Find a good quality research paper (Q1 Journals or CORE-A Conferences)

Implement a system demonstrating the concepts discussed in the research paper

Compare your results with the research paper results & **Improve** the results

Record a presentation on the work done

Submit video link on the google classroom

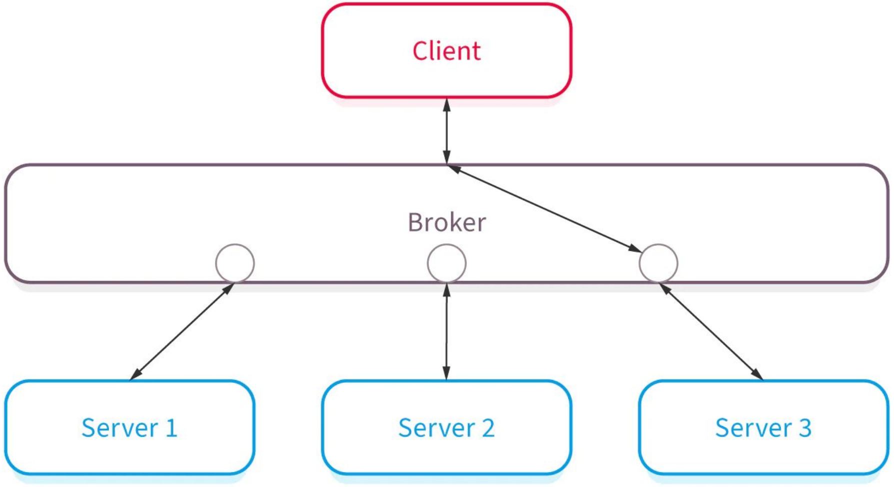
Deadline for submission: 15th November 2024

Software Architecture

Tactics, Patterns, and Quality Attributes

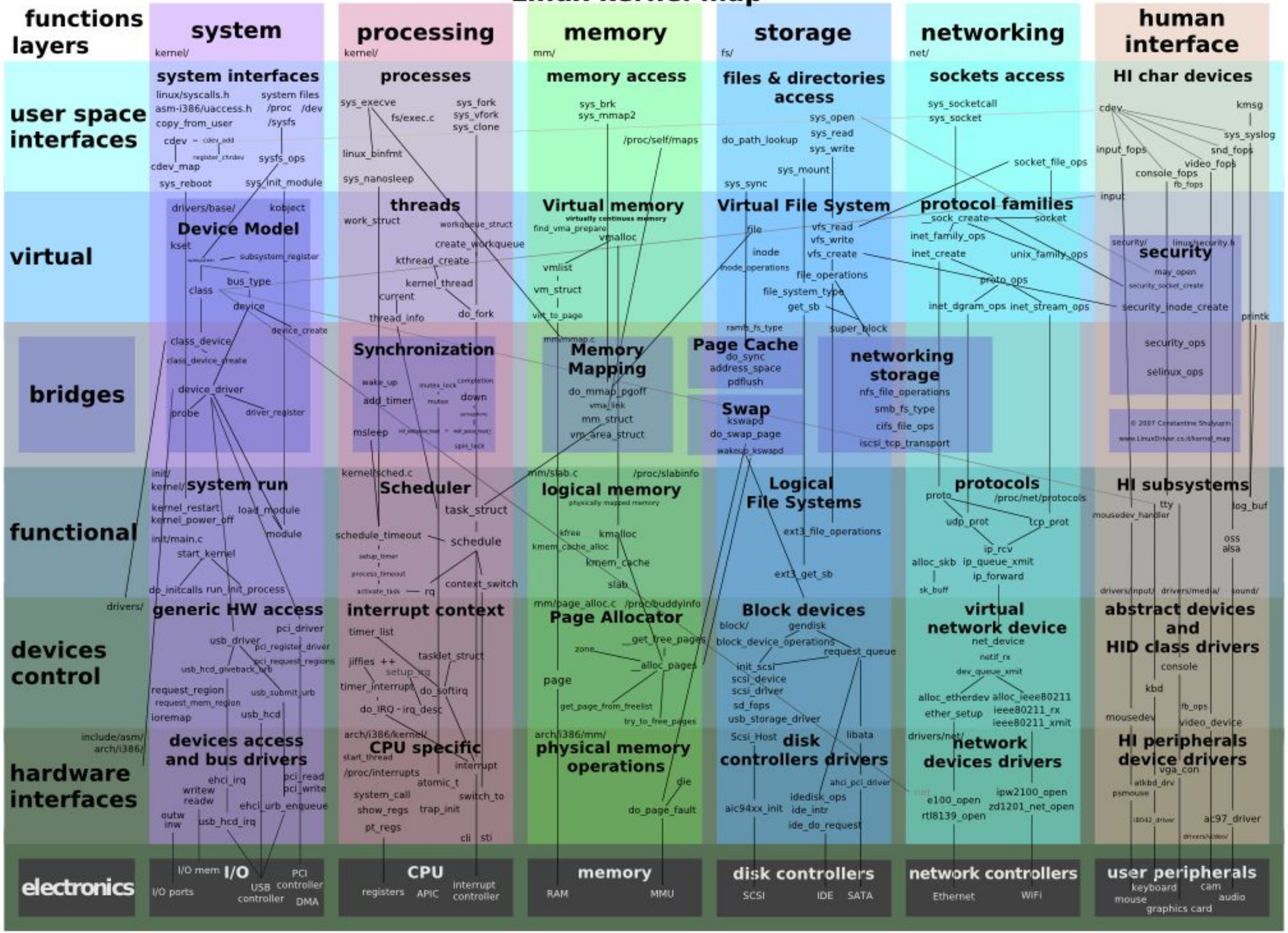
Software Architecture

The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.



Broker pattern

Linux kernel map



functions layers	system kernel/	processing kernel/	memory mm/	storage fs/	networking net/	human interface
user space interfaces	**system interfaces** linux/syscalls.h asm-i386/uaccess.h copy_from_user system files /proc /dev /sysfs cdev cdev_add register_chrdev cdev_map sysfs_ops sys_reboot sys_init_module	**processes** sys_execve fs/exec.c sys_fork sys_vfork sys_clone linux_binfmt sys_nanosleep	**memory access** sys_brk sys_mmap2 /proc/self/maps	**files & directories access** sys_open sys_read sys_write do_path_lookup sys_mount sys_sync	**sockets access** sys_socketcall sys_socket socket_file_ops	**HI char devices** cdev kmsg sys_syslog input_fops snd_fops video_fops console_fops fb_fops
virtual	drivers/base/ kobject **Device Model** kset subsystem subsystem_register bus_type class device device_create	**threads** work_struct workqueue_struct create_workqueue kthread_create kernel_thread current thread_info do_fork	**Virtual memory** virtually continues memory find_vma_prepare vmalloc vmlist vm_struct virt_to_page	**Virtual File System** file vfs_read vfs_write inode vfs_create inode_operations file_operations file_system_type get_sb	**protocol families** __sock_create socket inet_family_ops inet_create unix_family_ops proto_ops inet_dgram_ops inet_stream_ops	input security/ linux/security.h **security** may_open security_socket_create security_inode_create printk
bridges	class_device class_device_create device_driver probe driver_register	**Synchronization** wake_up mutex_lock completion add_timer mutex down semaphore msleep spin_lock	mm/mmap.c **Memory Mapping** do_mmap_pgoff vma_link mm_struct vm_area_struct	ramfs_fs_type super_block **Page Cache** do_sync address_space pdflush **Swap** kswapd do_swap_page wakeup_kswapd **networking storage** nfs_file_operations smb_fs_type cifs_file_ops iscsi_tcp_transport		security_ops selinux_ops © 2007 Constantine Shulyupin www.LinuxDriver.co.il/kernel_map
functional	init/ kernel/ **system run** kernel_restart kernel_power_off load_module module init/main.c start_kernel do_initcalls run_init_process	kernel/sched.c **Scheduler** task_struct schedule_timeout schedule setup_timer process_timeout activate_task rq context_switch	mm/slab.c /proc/slabinfo **logical memory** physically mapped memory kfree kmalloc kmem_cache_alloc kmem_cache slab	**Logical File Systems** ext3_file_operations ext3_get_sb	**protocols** proto /proc/net/protocols udp_prot tcp_prot ip_rcv alloc_skb ip_queue_xmit ip_forward sk_buff	**HI subsystems** tty mousedev_handler log_buf oss alsa drivers/input/ drivers/media/ sound/
devices control drivers/	**generic HW access** pci_driver usb_driver pci_register_driver pci_request_regions usb_hcd_giveback_urb request_region usb_submit_urb request_mem_region usb_hcd ioremap	**interrupt context** timer_list jiffies ++ tasklet_struct setup_irq timer_interrupt do_softirq do_IRQ - irq_desc	mm/page_alloc.c /proc/buddyinfo **Page Allocator** __get_free_pages zone __alloc_pages page get_page_from_freelist try_to_free_pages	**Block devices** block/ gendisk block_device_operations request_queue init_scsi scsi_device scsi_driver sd_fops usb_storage_driver	**virtual network device** net_device netif_rx dev_queue_xmit alloc_etherdev alloc_ieee80211 ether_setup ieee80211_rx ieee80211_xmit	**abstract devices and HID class drivers** console kbd fb_ops mousedev video_device
hardware interfaces include/asm/ arch/i386/	**devices access and bus drivers** ehci_irq pci_read pci_write writew readw ehci_urb_enqueue outw inw usb_hcd_irq	arch/i386/kernel/ **CPU specific** start_thread interrupt /proc/interrupts atomic_t system_call switch_to show_regs trap_init pt_regs cli sti	arch/i386/mm/ **physical memory operations** die do_page_fault	Scsi_Host **disk controllers drivers** libata ahci_pci_driver idedisk_ops aic94xx_init ide_intr ide_do_request	drivers/net/ **network devices drivers** net e100_open ipw2100_open zd1201_net_open rtl8139_open	**HI peripherals device drivers** vga_con atkbd_drv psmouse i8042_driver ac97_driver drivers/video/
electronics	**I/O** I/O mem I/O ports USB controller PCI controller DMA	**CPU** registers APIC interrupt controller	**memory** RAM MMU	**disk controllers** SCSI IDE SATA	**network controllers** Ethernet WiFi	**user peripherals** keyboard mouse graphics card cam audio

Software Tactics

Tools to achieve QAs

Impacts other tactics and QAs

Software Quality Attributes

From design to QA analysis

Predicting QAs and quantification

QA trade-off analysis

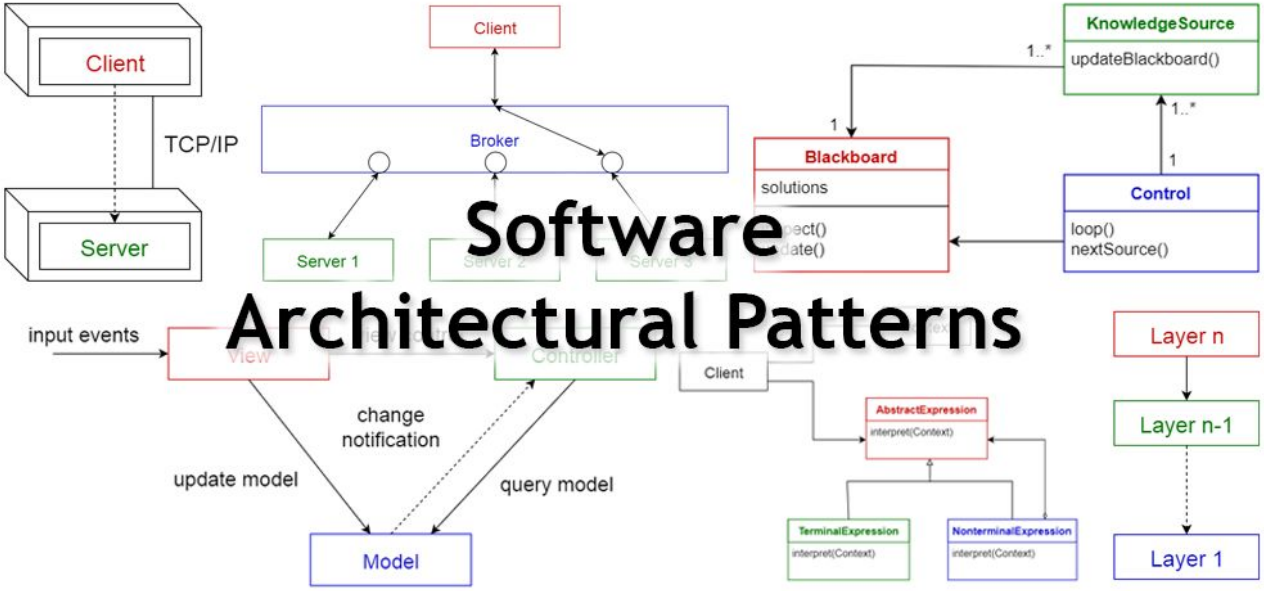
Software Patterns

A knows solution to a given problem in a particular context

Architectural Pattern

Architectural Pattern

A general, reusable solution to a commonly occurring problem in software architecture within a given context.



Some popular examples of architectural patterns

Layered pattern

Client-server pattern

Master-slave pattern

Pipe-filter pattern

Broker pattern

Peer-to-peer pattern

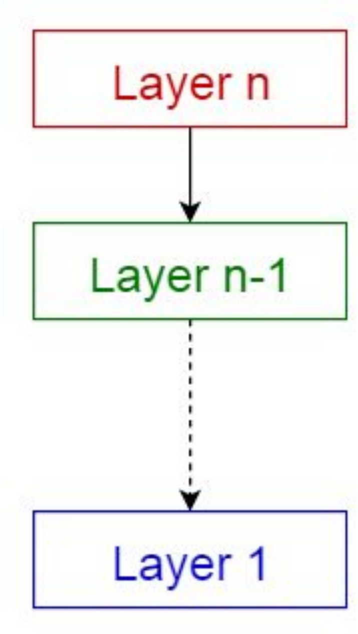
Event-bus pattern

Model-view-controller pattern

Blackboard pattern

Interpreter pattern

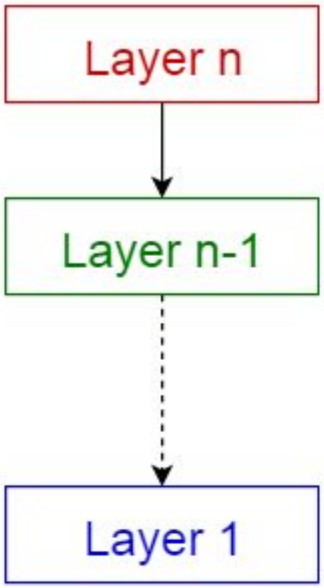
Layered Pattern



1. Organize the components of an application into horizontal logical layers and physical tiers.

2. A higher layer can use services in a lower layer, but not the other way around.

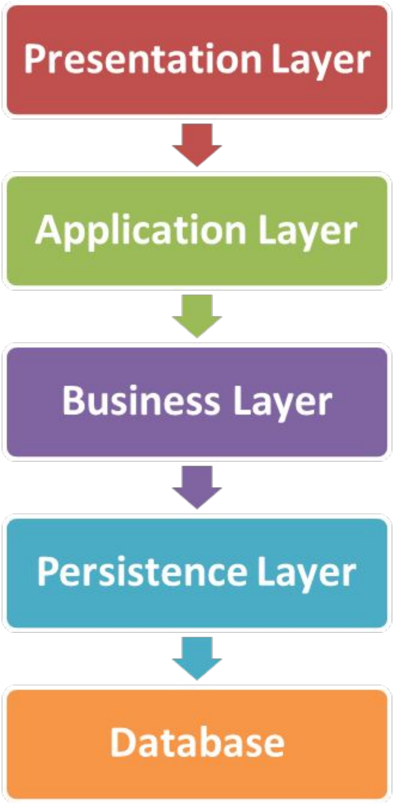
Layered Pattern



Layer n

Layer n-1

Layer 1



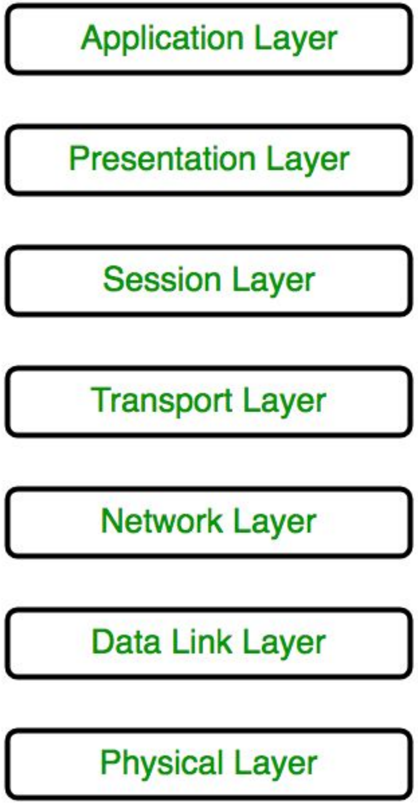
Presentation Layer

Application Layer

Business Layer

Persistence Layer

Database



Application Layer

Presentation Layer

Session Layer

Transport Layer

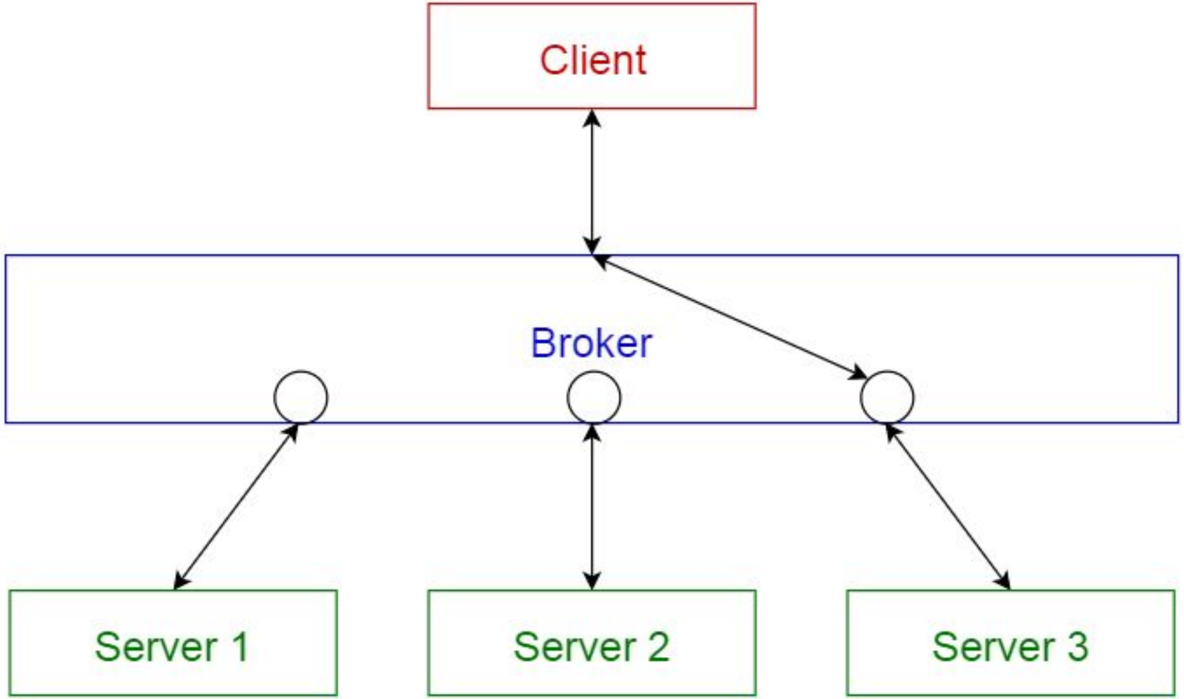
Network Layer

Data Link Layer

Physical Layer

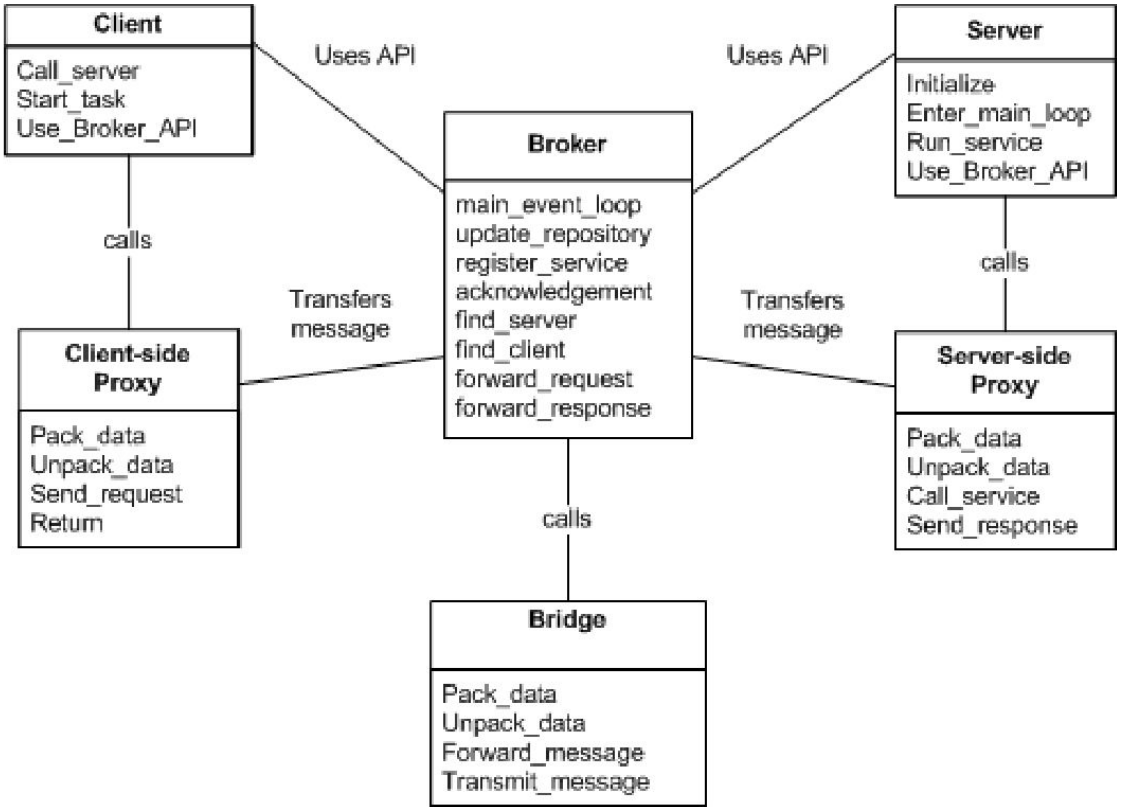
Broker Pattern

The Broker architectural pattern can be used to structure distributed software systems with decoupled components that interact by remote service invocations.



Broker Pattern

The Broker component is responsible for coordinating communication, such as forwarding requests, as well as transmitting results and exceptions



Microservice Architecture

Modular v/s Monolithic Software Design

MVC

MVC

Name	**MVC (Model-View-Controller)**
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model.
Example	A web-based application system organized using the MVC pattern.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.

Pipe-and-Filter

Pipe-and-Filter

Name	Pipe and filter
Description	The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.
Example	A pipe and filter system used for processing invoices.
When used	Commonly used in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs.
Advantages	Easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformations is straightforward. Can be implemented as either a sequential or concurrent system.
Disadvantages	The format for data transfer has to be agreed upon between communicating transformations. Each transformation must parse its input and unparse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures.

Client-Server

Client-Server

Name	Client-server
Description	In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.
Example	An example of a film and Video/DVD library organized as a client–server system.
When used	Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organizations.

Peer-to-Peer

Peer-to-Peer

Name	Peer-to-peer
Description	
Example	
When used	
Advantages	
Disadvantages	

slido

Can we combine MVC with Pipe-n-Filter pattern?

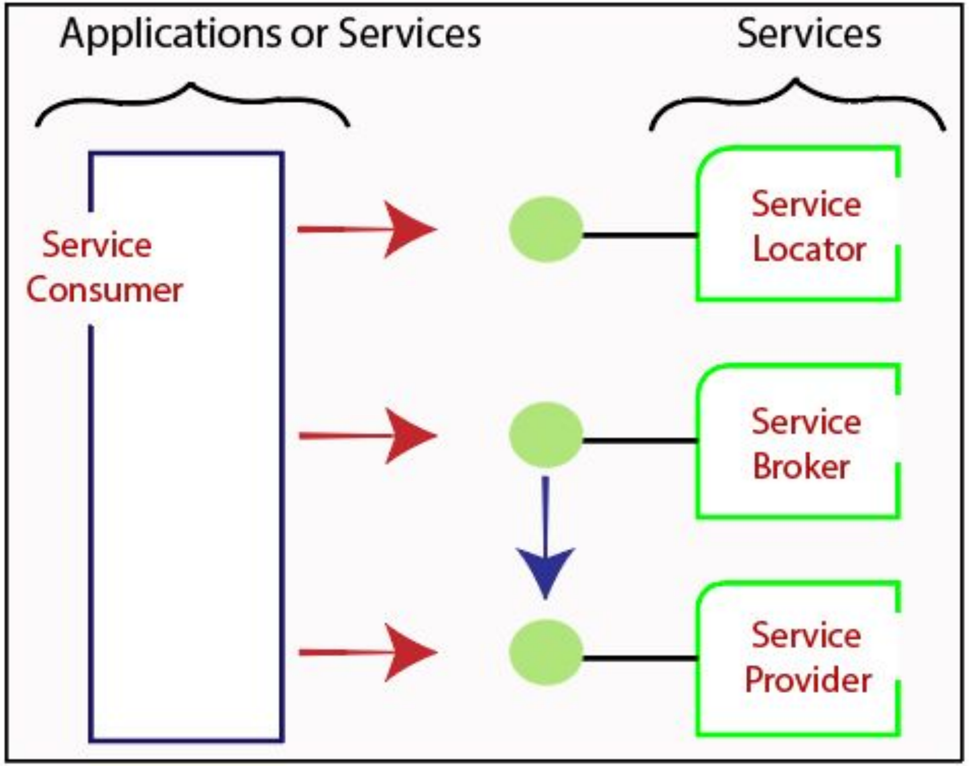
ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

slido

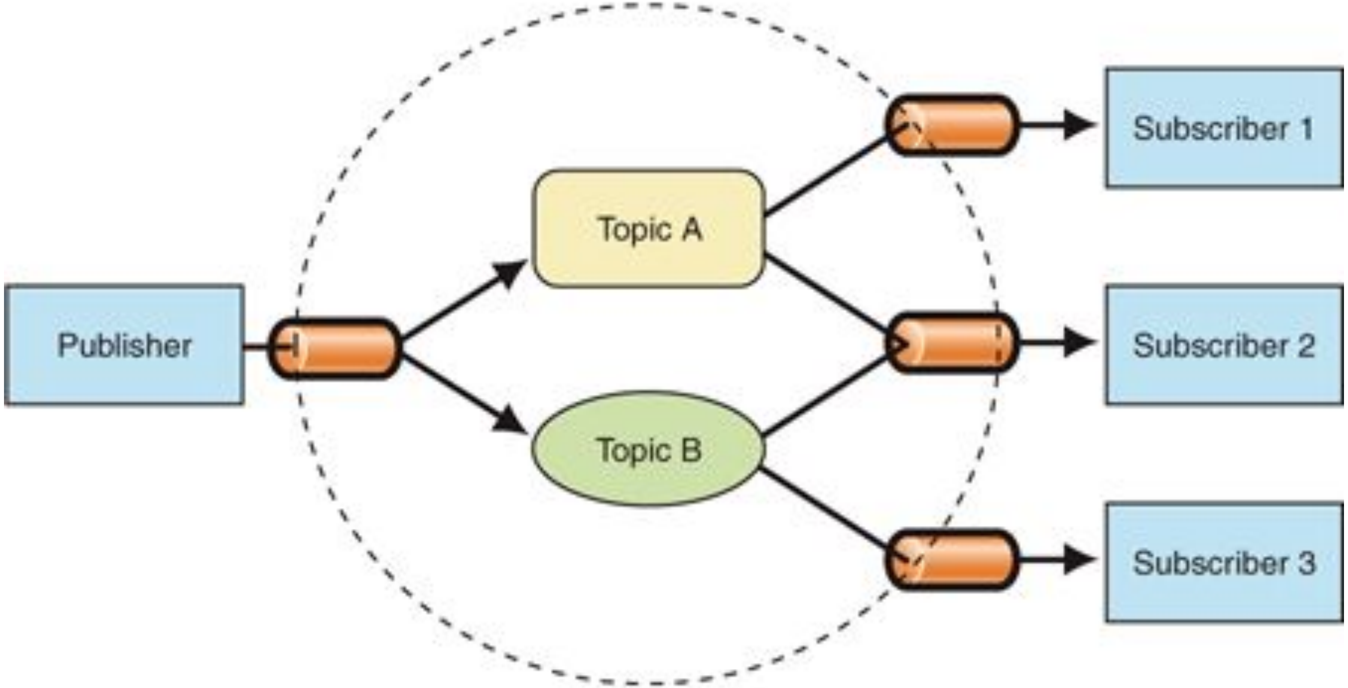
P2P is a modified form of Client-Server.

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

Service Oriented Architecture

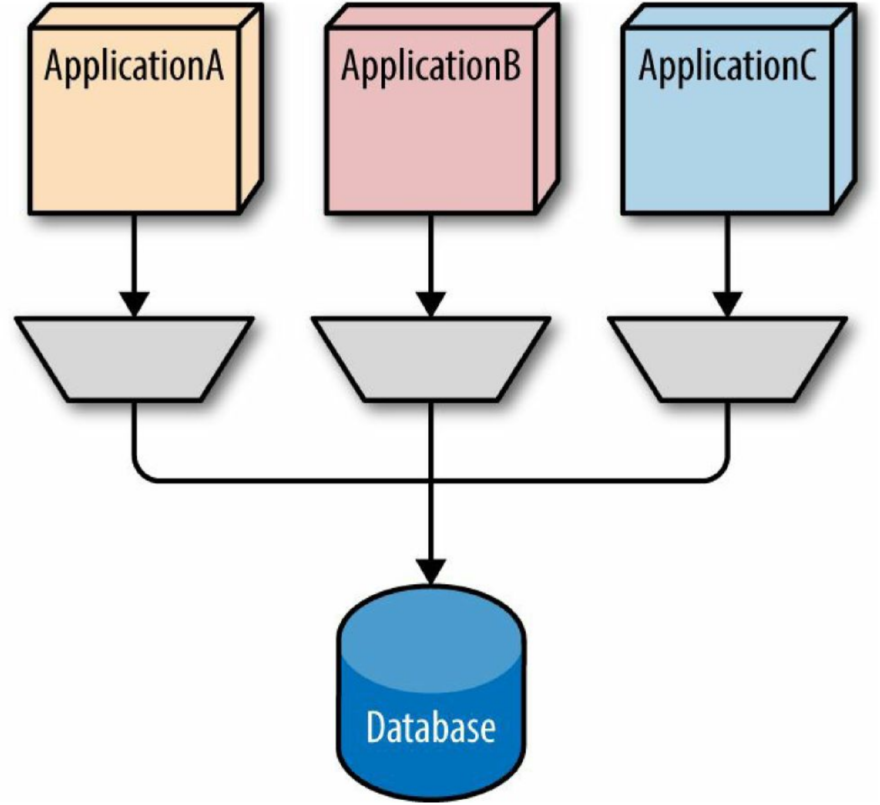


Publisher-Subscriber Pattern



Aren’t Publishers providing services to Subscribers? If Yes, how is it different from SOA?

Shared Data Pattern

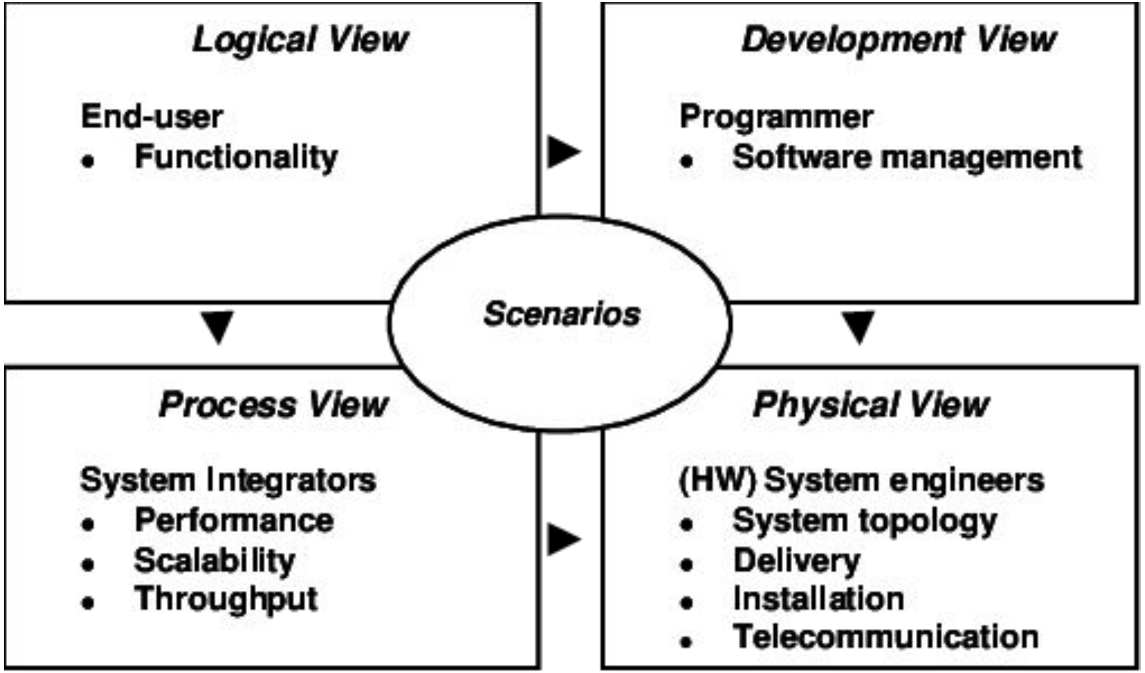


Multiple Subscribers are using common set of Publishers. How is it different from Shared Data Pattern?

4+1 Architectural Views

- A logical view, which shows the key abstractions in the system as objects or object classes.
- A process view, which shows how, at run-time, the system is composed of interacting processes.
- A development view, which shows how the software is decomposed for development.
- A physical view, which shows the system hardware and how software components are distributed across the processors in the system.
- Related using use cases or scenarios (+1)

Original 4+1 View Model Proposed by Kruchten (1995)



slido

Blackboard pattern and Pub-Sub patterns are same

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

slido

Is it possible to use Shared Data Pattern with Pub-Sub pattern for providing data sharing services?

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

slido

Which view is prepared first?

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.