Software Architecture and Architectural Patterns/Styles

Parinya Ekparinya

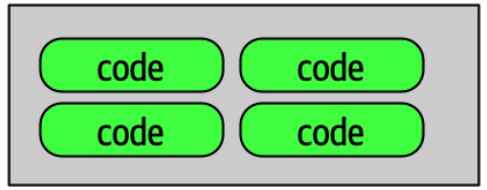
Parinya.Ek@kmitl.ac.th

หรือก็คือ unit of code

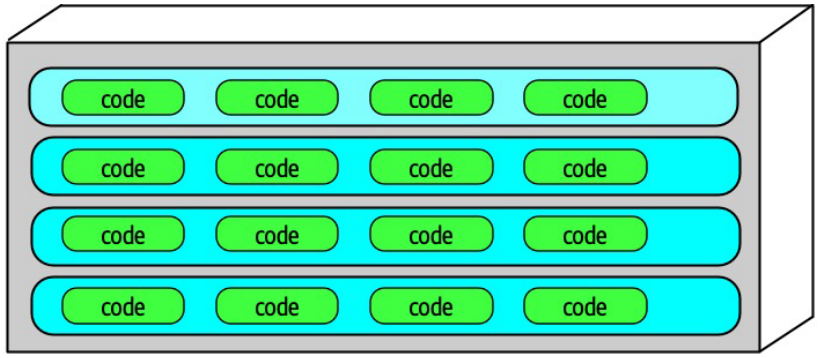
Components

- Developers physically package modules in different ways, sometimes depending on their development platform. Generally, we may call physical packaging of modules ***components***.
- Most languages support physical packaging as well: **jar** files in Java, **dll** in .NET, **gem** in Ruby, and so on.
- Components offer a language-specific mechanism to group artifacts together, often nesting them to create stratification.
- Components form the fundamental modular building block in architecture.
- One of the primary decisions an architect must make concerns the top-level partitioning of components in the architecture.

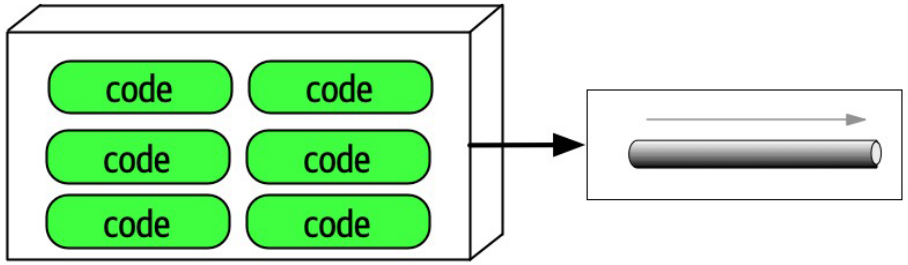
Richards, M., & Ford, N. (2020). *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media.



Wrapper of related code

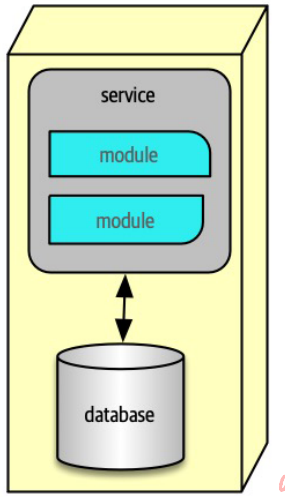


Layer or subsystem แนวคิดเหมือน OSI model



Event processor

massage on block to block



Distributed service

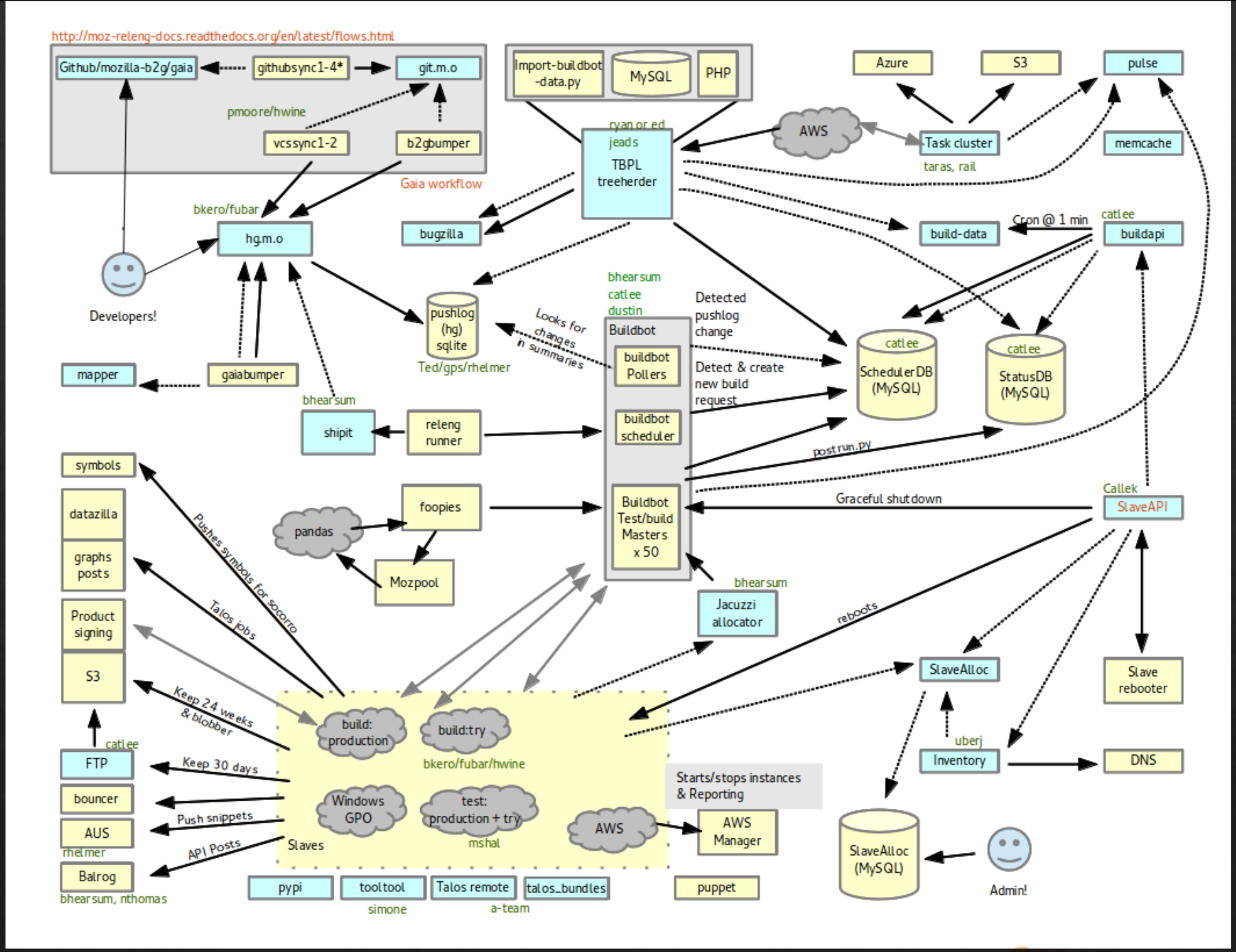
Richards, M., & Ford, N. (2020). *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media.

Architecture Partitioning

create Component

- The First Law of Software Architecture states that everything in software is a trade-off, including how architects create components in an architecture.
- Because components represent a general containership mechanism, an architect can build any type of partitioning they want.
- Several common styles exist, with different sets of trade-offs.

Richards, M., & Ford, N. (2020). *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media.



https://www.chesnok.com/daily/2014/05/02/release-engineering-a-draft-of-an-architecture-diagram/

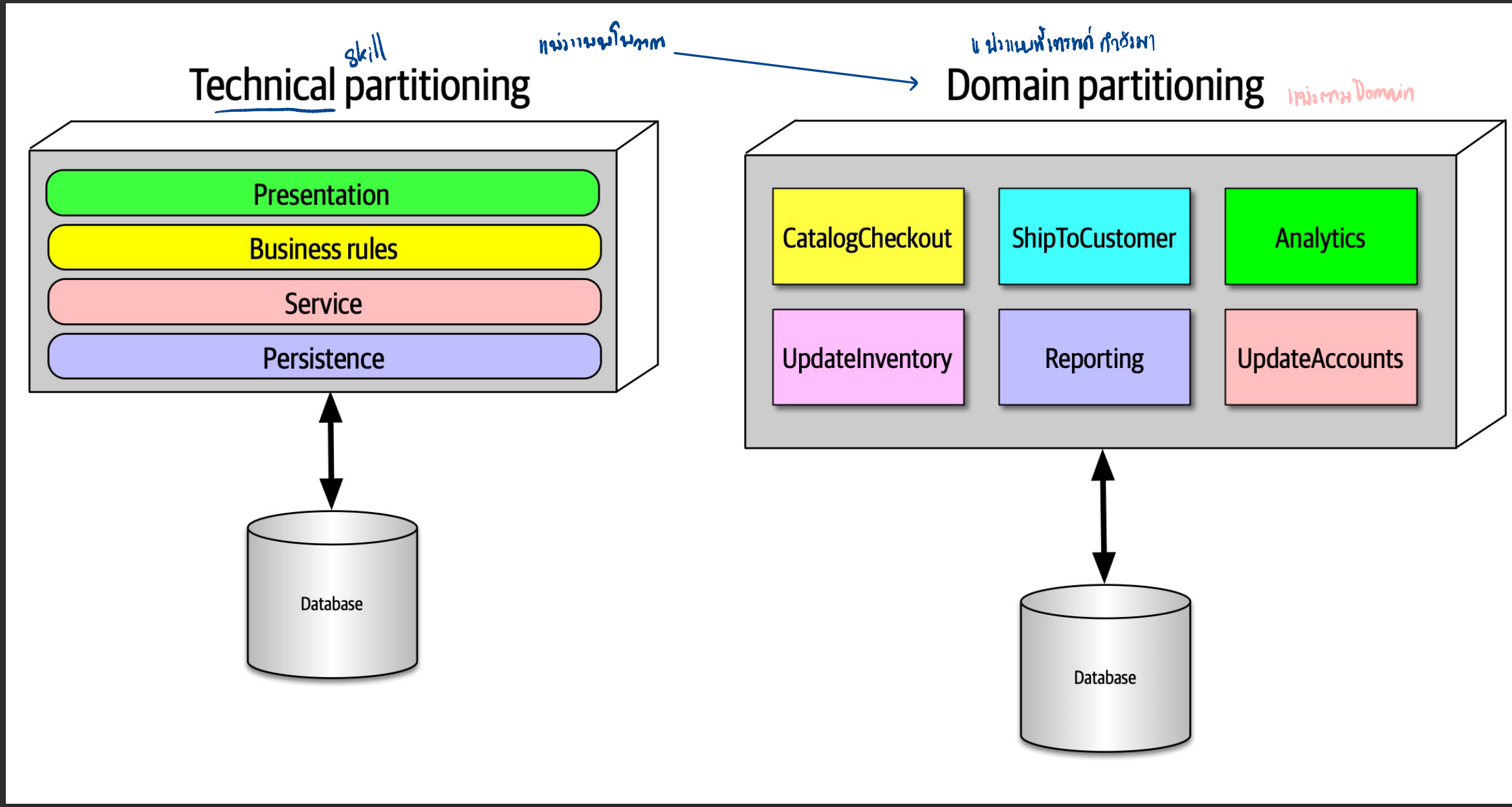
กฎของ Conway

องค์ประกอบที่สร้างออกแบบระบบ มีแนวโน้มที่จะออกแบบ copy โครงสร้าง องค์กร

“Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.”

— Melvin E. Conway

Conway, M. E. (1968). How do committees invent. *Datamation*, 14(4), 28-31.



Richards, M., & Ford, N. (2020). *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media.

Catalog of Architectural Patterns/Styles

Monolithic

- Layers
- Plug-in (Microkernel)
- Model-View-Controller (MVC)

Distributed

- Client-Server & N-Tier
- Publish-Subscribe
- Service-Oriented Architecture (SOA)
- Representational State Transfer (REST)

We list an assortment of useful and widely used patterns/styles. This catalog is not meant to be exhaustive—in fact no such catalog is possible!!

See: https://en.wikipedia.org/wiki/List_of_software_architecture_styles_and_patterns

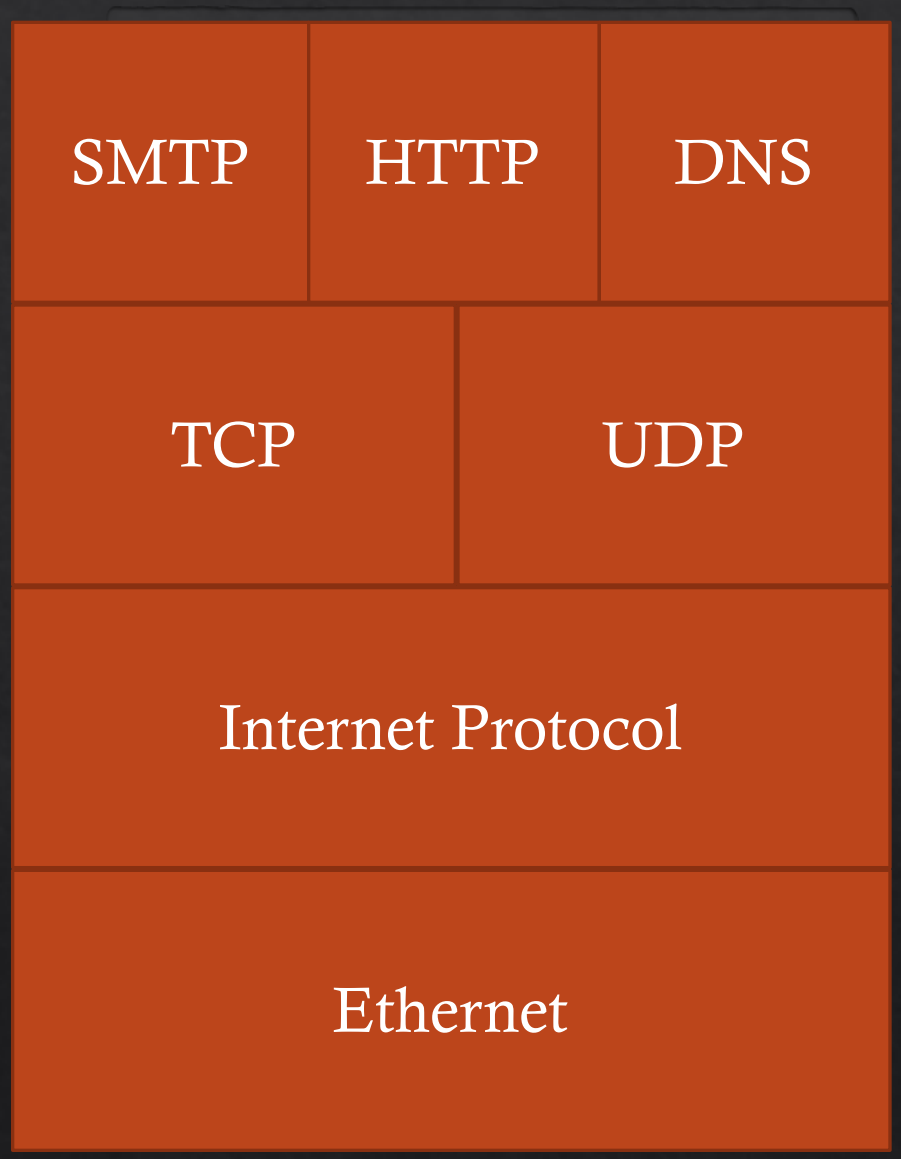
Layers

Layers

- The layers pattern divides the system in such a way that the modules can be developed and evolved separately with little interaction among the parts, which supports portability, modifiability, and reuse.
- To achieve this separation of concerns, the layers pattern divides the software into units called layers.
- Each layer is a grouping of modules that offers a cohesive set of services.
- The allowed to-use relationship among the layers is subject to a key constraint: The relations must be unidirectional.
- Upward usages are not allowed in this pattern.

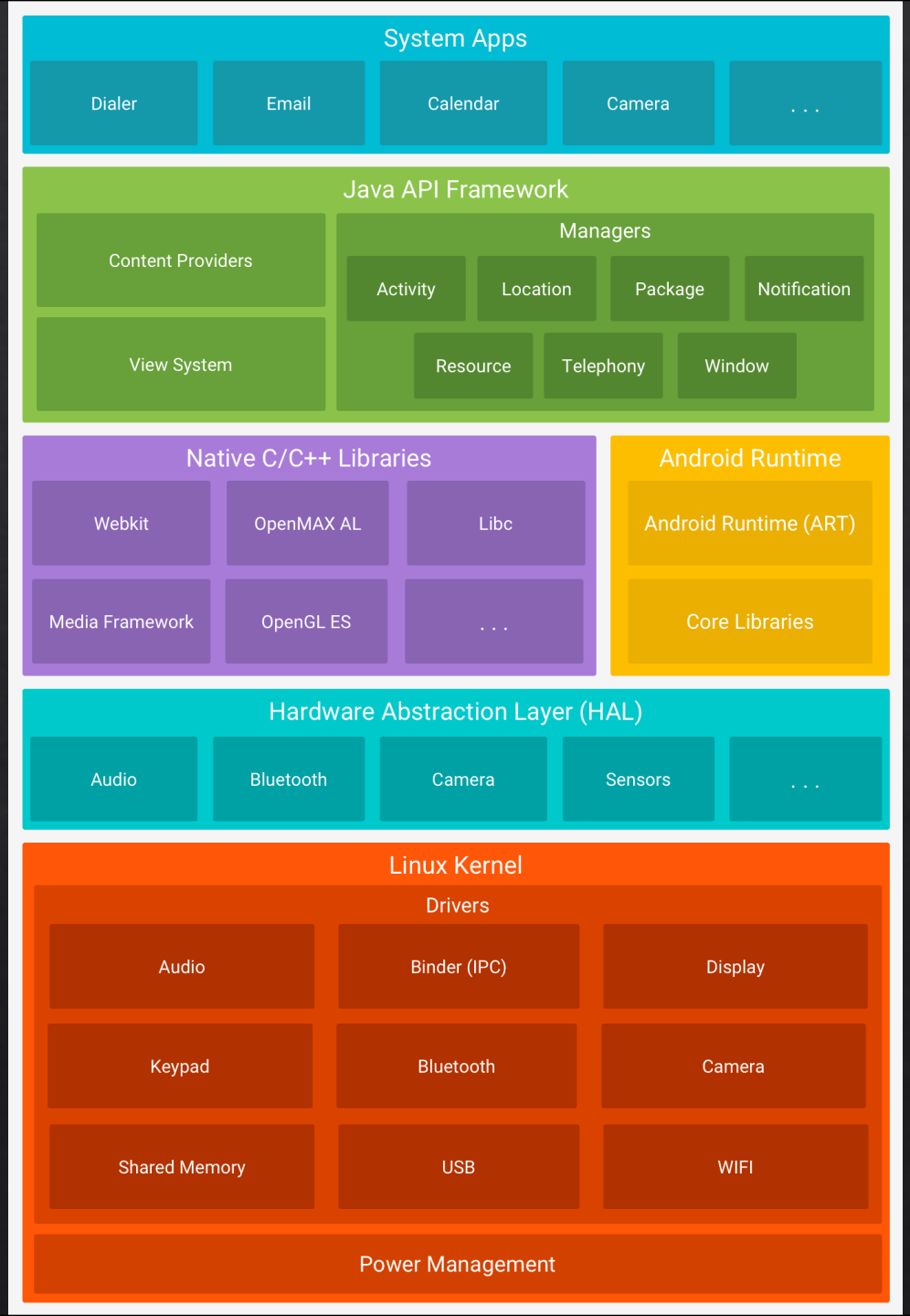
TCP/IP Protocol Suite

RFC 1122



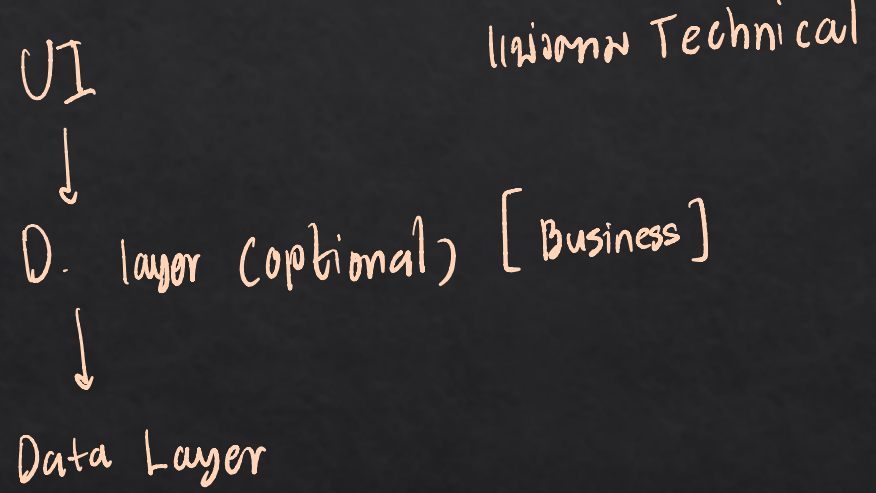
Android Platform Architecture

https://developer.android.com/guide/platform



Layers: Real-World Example

- https://developer.android.com/topic/architecture



layer บน เรียกใช้ layer ล่าง แม้ไม่ต่อกันโดยตรง

Relaxed Layers

- A variant of layers pattern with less restrictive relationship among layers.
- Some layer may be marked as *open layers*, which allow their upper layers to reach their lower layers through them.
  layer ด้านล่าง ถ้าเปิดทางจึงจะใช้ได้
- So, a layer may use non-adjacent layers below it.

Relaxed Layers: Real-World Example

- https://www.open-mpi.org/video/internals/Cisco_JeffSquyres-1up.pdf

Layers: Benefits

- Because a layer is constrained to use only lower layers, software in lower layers can be changed (as long as the interface does not change) without affecting the upper layers.
- Lower-level layers may be reused across different applications. For example, suppose a certain layer allows portability across operating systems. This layer would be useful in any system that must run on multiple, different operating systems. The lowest layers are often provided by commercial software—an operating system, for example, or network communications software.
- Because the allowed-to-use relations are constrained, the number of interfaces that any team must understand is reduced.

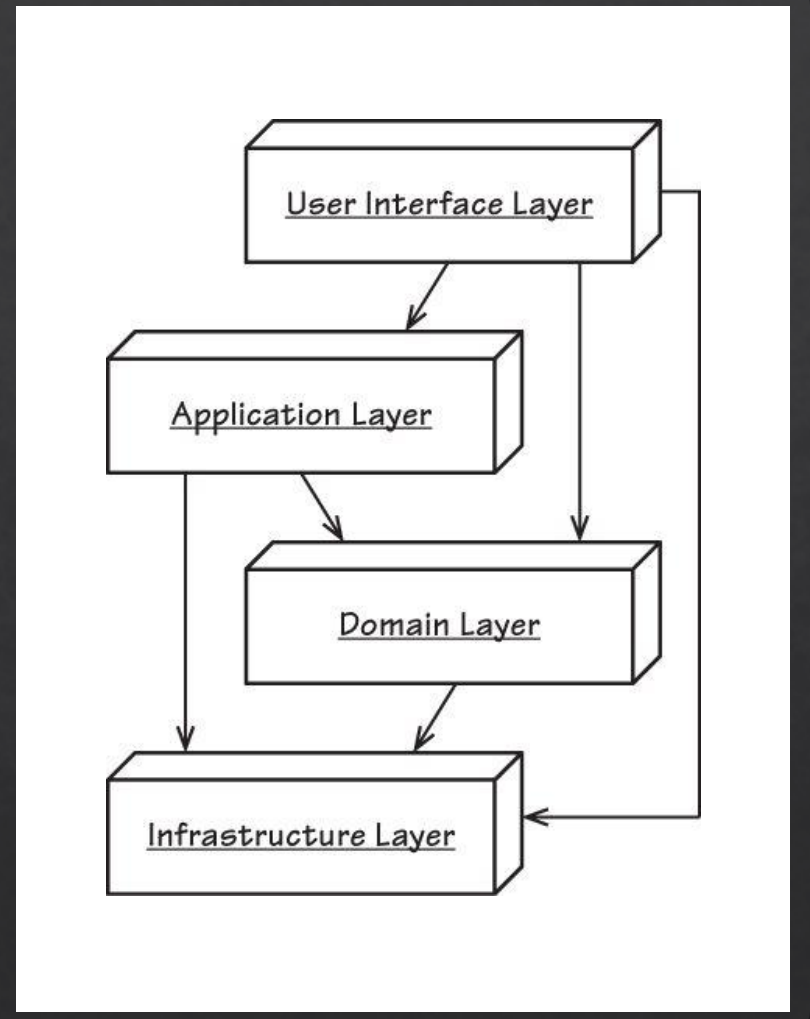
Layers: Tradeoffs

- If the layering is not designed correctly, it may actually get in the way, by not providing the lower-level abstractions that programmers at the higher levels need.
- Layering often adds a performance penalty to a system. If a call is made from a function in the top-most layer, it may have to traverse many lower layers before being executed by the hardware.
- If many instances of layer bridging occur, the system may not meet its portability and modifiability goals, which strict layering helps to achieve.

Layered Architecture in DDD

↳ 4 layer

- Applying a traditional Layered Architecture, there are four layers common to a DDD application.
- The isolated Core Domain resides in one layer in the architecture. Above it are the **User Interface** and **Application Layers**. Below it is the **Infrastructure Layer**.



User Interface Layer

- The User Interface is to contain only code that addresses user view and request concerns. It must not contain domain/business logic.
- Since a user may be either a human or other systems, sometimes this layer will provide the means to remotely invoke the services of an API in the form of an Open Host Service.
- Components in the User Interface are direct clients of the Application Layer.

Application Layer

- Application Services reside in the Application Layer. These are different from Domain Services and are thus devoid of domain logic.
- Application Services may control persistence transactions and security. They may also be in charge of sending Event-based notifications to other systems and/or for composing e-mail messages to be sent to users.
- The Application Services in this layer are the direct clients of the domain model, though themselves possessing no business logic. If our Application Services become much more complex than this, it is probably an indication that domain logic is leaking into the Application Services.
- An Application Service may also use a Domain Service to fulfill some domain-specific task designed as a stateless operation.

เกี่ยวกับการเก็บ storage

Infrastructure Layer

Physical

- In a traditional Layers Architecture the Infrastructure is at the bottom. Things like persistence and messaging mechanisms reside there.
- Messages may include those sent by enterprise messaging middleware systems or more basic e-mails (SMTP) or text messages (SMS).
- Think of all the technical components and frameworks that provide low-level services for the application. Those are usually considered to be part of Infrastructure.
- The higher-level Layers couple to the lower-level components to reuse the technical facilities provided.

เกี่ยวข้อง กับ storay DB

สามารถปรับเปลี่ยนหรือรวมได้

Plug-in (Microkernel)

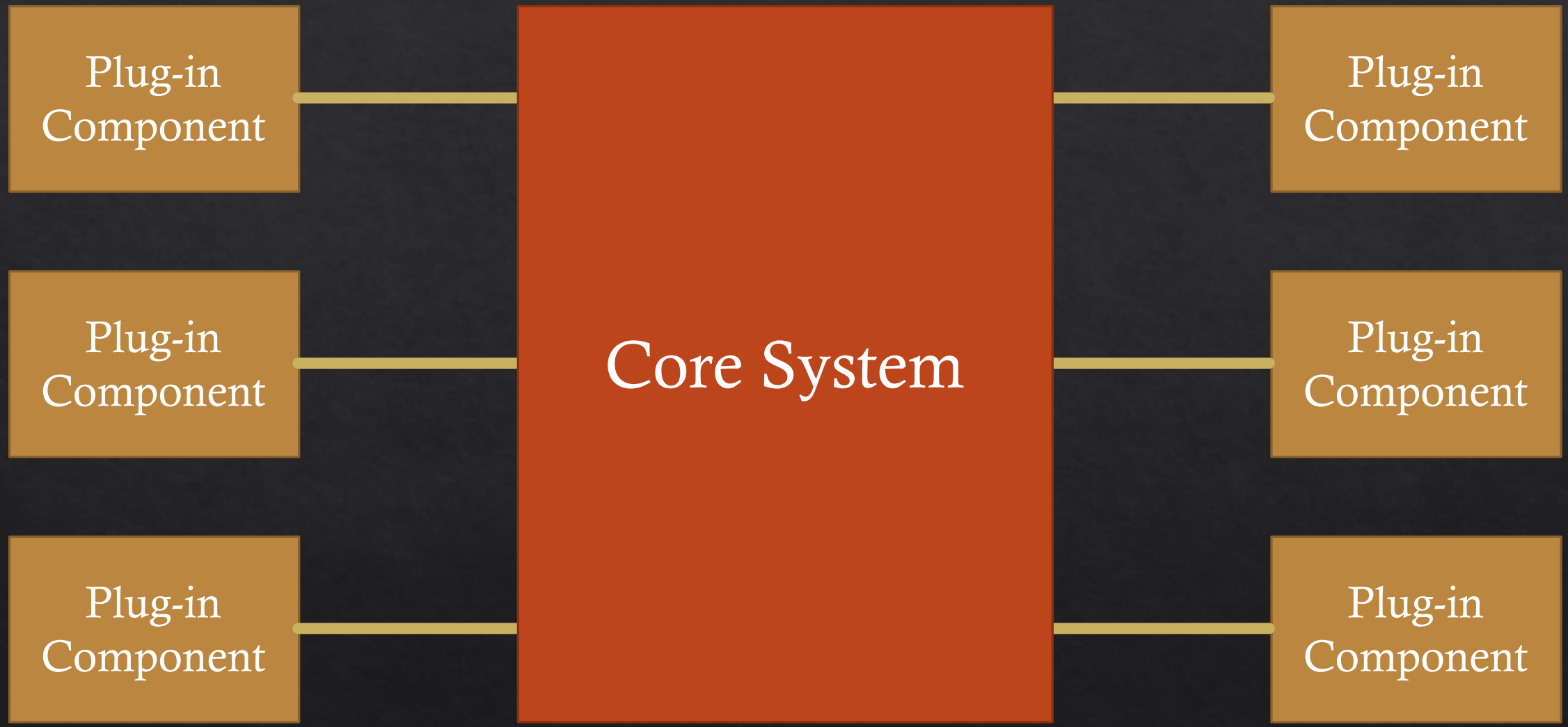
Plug-in (Microkernel) (1/2)

- The plug-in pattern has two types of elements—elements that provide a core set of functionality and specialized variants (called plug-ins) that add functionality to the core via a fixed set of interfaces.
- The two types are typically bound together at build time or later.

Plug-in (Microkernel) (2/2)

Examples of usage include the following cases:

- The core functionality may be a stripped-down operating system (the microkernel) that provides the mechanisms needed to implement OS services, such as low-level address space management, and inter-process communication (IPC). The plug-ins provide the actual OS functionality, such as device drivers, task management, and I/O request management.
- The core functionality is a product providing services to its users. The plug ins provide portability, such as OS compatibility or supporting library compatibility. The plug-ins can also provide additional functionality not included in the core product. In addition, they can act as adapters to enable integration with external systems.


Plug-in Component
Plug-in Component
Plug-in Component
Core System
Plug-in Component
Plug-in Component
Plug-in Component

Microkernel: Real-World Example

- https://help.eclipse.org/latest/index.jsp?nav=%2F2_0

ระบบ ระบบ มีหลาย ปลั๊กอิน

Microkernel: Benefits

Plug-ins provide a controlled mechanism to extend a core product and make it useful in a variety of contexts.

- The plug-ins can be developed by different teams or organizations than the developers of the microkernel. This allows for the development of two different markets: for the core product and for the plug-ins.
- The plug-ins can evolve independently from the microkernel. Since they interact through fixed interfaces, as long as the interfaces do not change, the two types of elements are not otherwise coupled.

ปลั๊กอินเปลี่ยนฟังก์ชันกับ แกนหลาง

เช่น แกนหลาง อัปเกต → plug-ins อาจอัปเกต

Dev relation กับ plug-ins แบบไหน ?

Microkernel: Tradeoffs

มีช่องทางเกิดปัญหา อาจถูกโจมตีง่าย

- Because plug-ins can be developed by different organizations, it is easier to introduce security vulnerabilities and privacy threats.

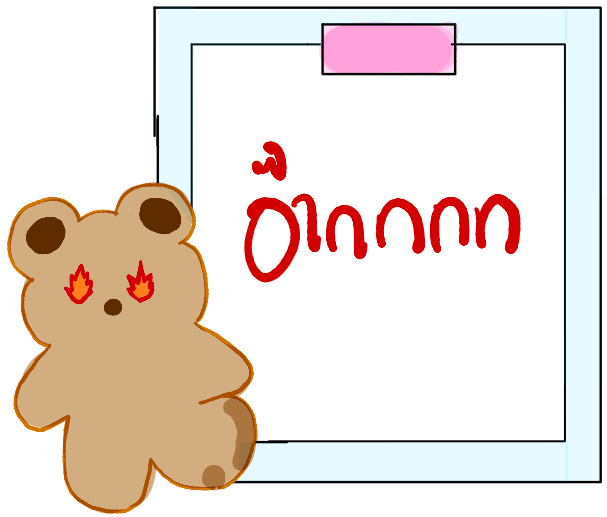
Model-View-Controller (MVC)

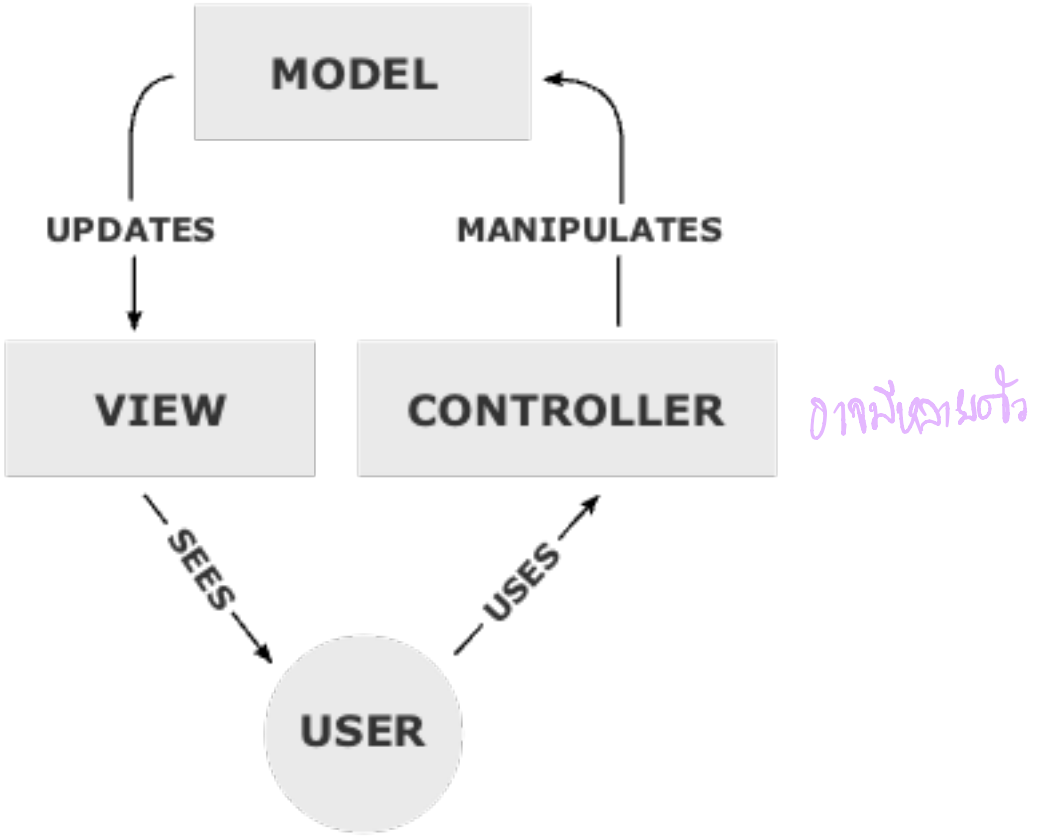
Model-View-Controller (MVC) (1/2)

- MVC is likely the most widely known pattern for usability.
- It comes in many variants, such as MVP (model-view-presenter), MVVM (model-view-view-model), MVA (model-view-adapter), and so forth.
- Essentially all of these patterns are focused on separating the model—the underlying "business" logic of the system—from its realization in one or more UI views.
- In the original MVC model, the model would send updates to a view, which a user would see and interact with.
- User interactions—key presses, button clicks, mouse motions, and so forth—are transmitted to the controller, which interprets them as operations on the model and then sends those operations to the model, which changes its state in response.
- The reverse path was also a portion of the original MVC pattern. That is, the model might be changed and the controller would send updates to the view.

Model-View-Controller (MVC) (2/2)

- The sending of updates depends on whether the MVC is in one process or is distributed across processes (and potentially across the network).
- If the MVC is in one process, then the updates are sent using the observer pattern (discussed in another lecture).
- If the MVC is distributed across processes, then the publish subscribe pattern is often used to send updates.





อาจมีหลายตัว

By RegisFrey - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=10298177

Real-World Example

- https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

MVC: Benefits

- Because MVC promotes clear separation of concerns, changes to one aspect of the system, such as the layout of the UI (the view), often have no consequences for the model or the controller.
- Additionally, because MVC promotes separation of concerns, developers can be working on all aspects of the pattern—model, view, and controller—relatively independently and in parallel. These separate aspects can also be tested in parallel.
- A model can be used in systems with different views, or a view might be used in systems with different models.

MVC: Tradeoffs

- MVC can become burdensome for complex UIs, as information is often sprinkled throughout several components. For example, if there are multiple views of the same model, a change to the model may require changes to several otherwise unrelated components.
- For simple UIs, MVC adds up-front complexity that may not pay off in downstream savings.

Client-Server & N-Tier

คู่ Client-server หมายถึง มีคุณภัย
สัมพันธ์กัน

Client-Server & N-Tier (1/2)

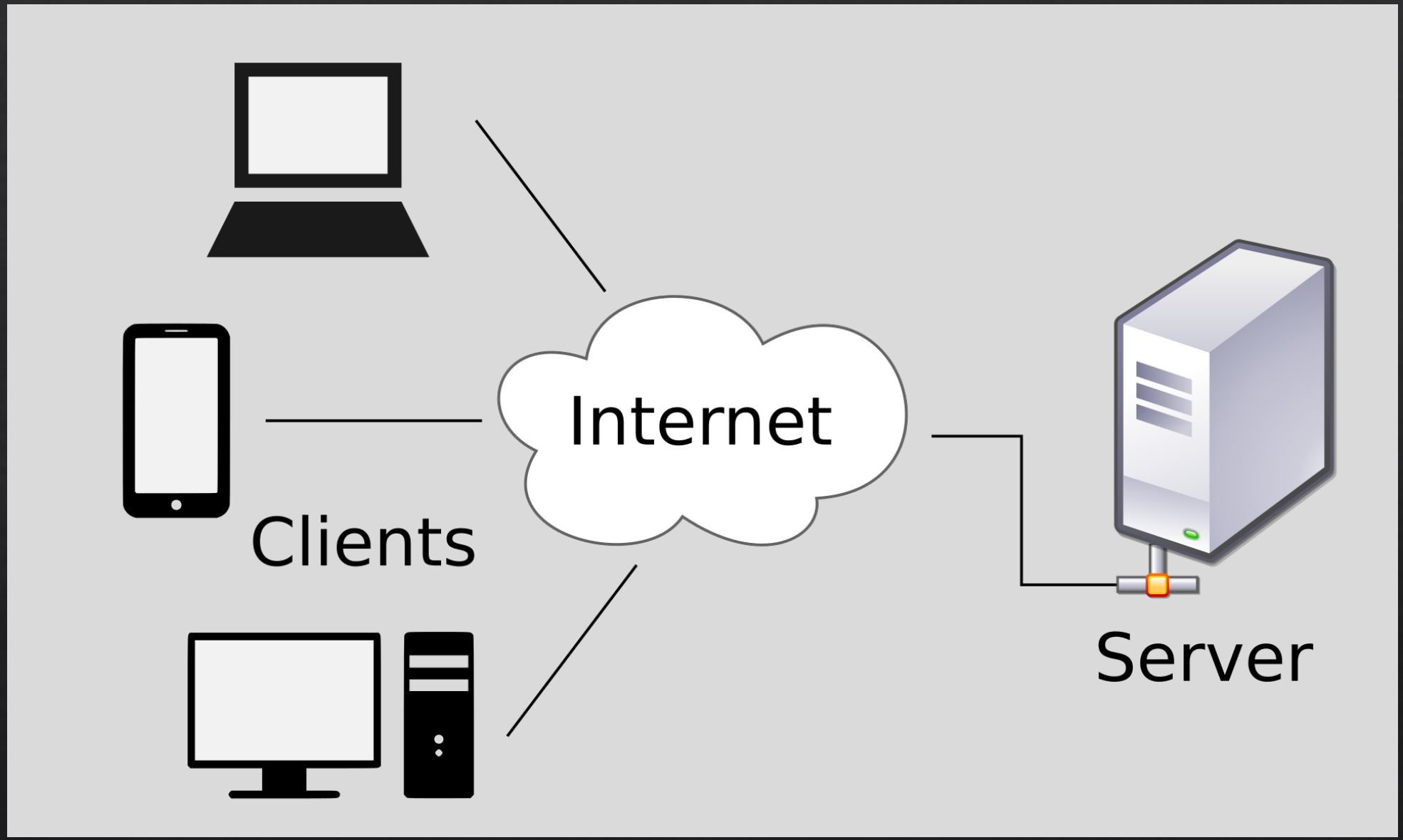
The client-server pattern consists of a server providing services simultaneously to multiple distributed clients. The most common example is a web server providing information to multiple simultaneous users of a website. The interactions between a server and its clients follow this sequence:

- Discovery:
  - Communication is initiated by a client, which uses a discovery service to determine the location of the server.
  - The server responds to the client using an agreed-upon protocol.
- Interaction:
  - The client sends requests to the server.
  - The server processes the requests and responds.

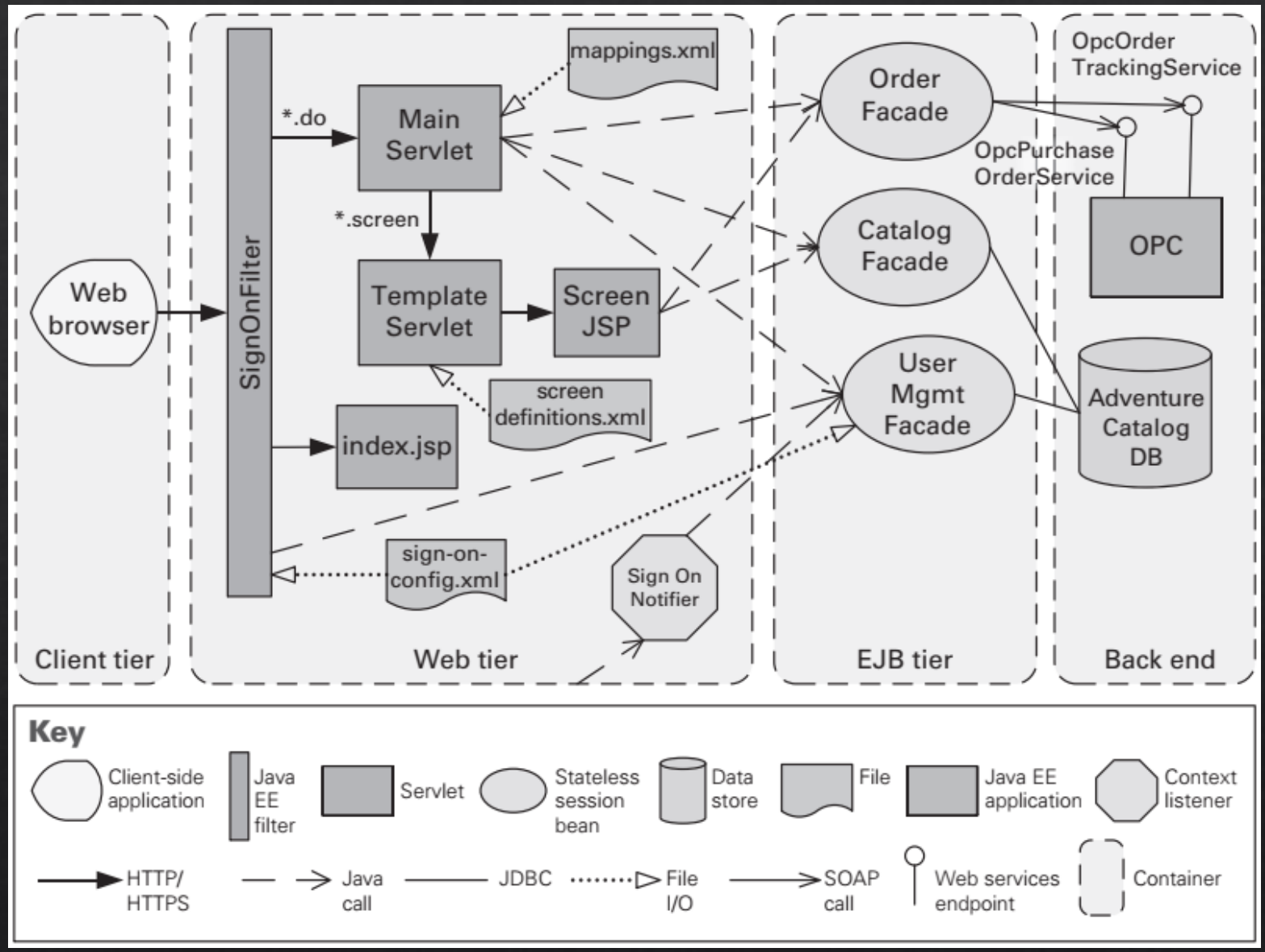
Client-Server & N-Tier (2/2)

Several points about this sequence are worth noting:

- The server may have multiple instances if the number of clients grows beyond the capacity of a single instance.
- If the server is stateless with respect to the clients, each request from a client is treated independently.
- If the server maintains state with respect to the clients, then:
  - Each request must identify the client in some fashion. Server ต้องรู้ว่าเป็น client ใคร
  - The client should send an "end of session" message so that the server can remove resources associated with that particular client.
  - The server may time out if the client has not sent a request in a specified time so that resources associated with the client can be removed.



https://en.wikipedia.org/wiki/Client%E2%80%93server_model#/media/File:Client-server-model.svg



Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice*. Addison-Wesley Professional.

Client-Server: Benefits

- The connection between a server and its clients is established dynamically.
- The server has no a prior knowledge of its clients; that is, there is low coupling between the server and its clients.
- There is no coupling among the clients.
- The number of clients can easily scale, and is constrained only by the capacity of the server. The server functionality can also scale if its capacity is exceeded.
- Clients and servers can evolve independently.
- Common services can be shared among multiple clients.
- The interaction with a user is isolated to the client. This factor has resulted in the development of specialized languages and tools for managing the user interface.

Client-Server: Tradeoffs

- This pattern is implemented such that communication occurs over a network, perhaps even the Internet. Thus, messages may be delayed by network congestion, leading to degradation (or at least unpredictability) of performance.
- For clients that communicate with servers over a network shared by other applications, special provisions must be made for achieving security (especially confidentiality) and maintaining integrity.

Publish-Subscribe

Publish-Subscribe

- Publish-subscribe is an architectural pattern in which components communicate primarily through asynchronous messages, sometimes referred to as “events” or “topics.”
- The publishers have no knowledge of the subscribers, and subscribers are only aware of message types.
- Systems using the publish-subscribe pattern rely on implicit invocation; that is, the component publishing a message does not directly invoke any other component.
- Components publish messages on one or more events or topics, and other components register an interest in the publication.
- At runtime, when a message is published, the publish–subscribe (or event) bus notifies all of the elements that registered an interest in the event or topic. In this way, the message publication causes an implicit invocation of (methods in) other components.

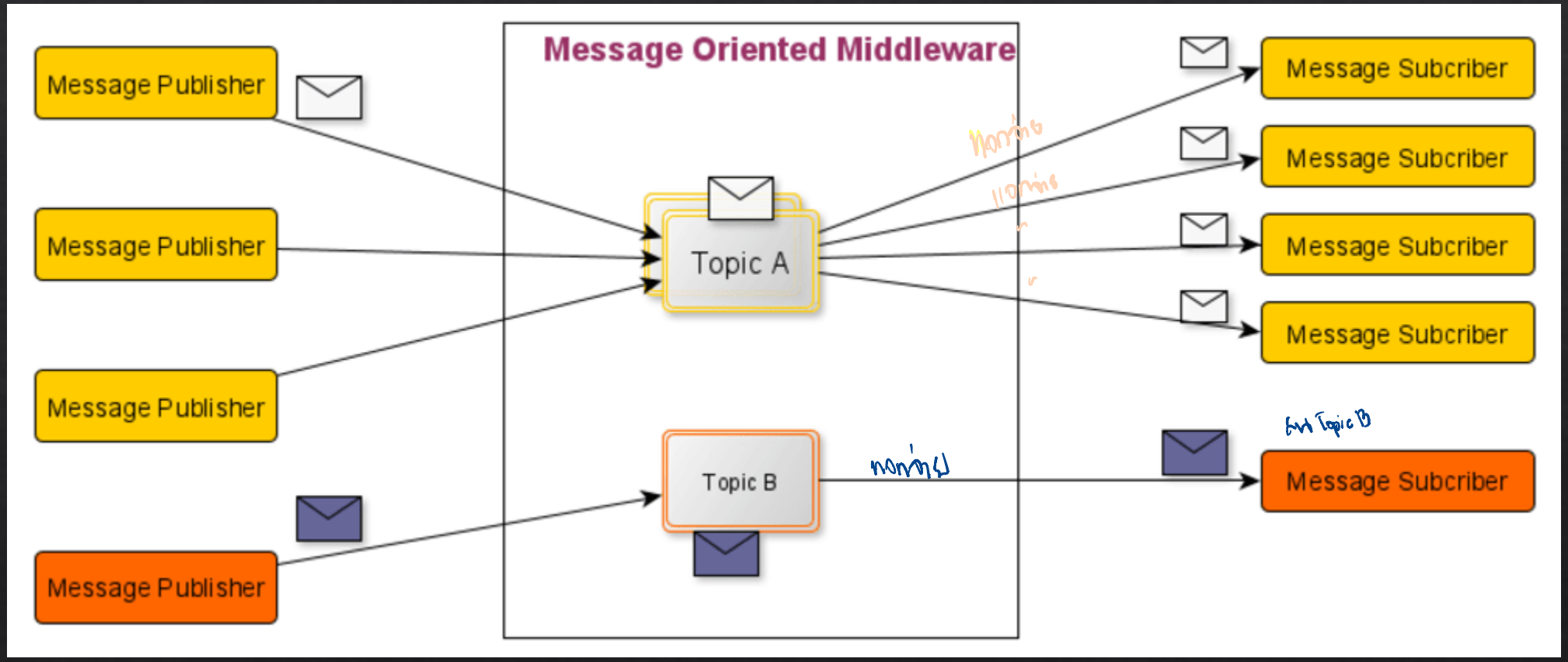
Publish-Subscribe: Elements

The publish-subscribe pattern has three types of elements:

- Publisher component. Sends (publishes) messages.
- Subscriber component. Subscribes to and then receives messages.
- Event bus. Manages subscriptions and message dispatch as part of the runtime infrastructure.

Publisher

Suberiber



https://dev.to/tranthanhdeveloper/point-to-point-and-publish-subscribe-messaging-model-41j0

Real-World Example

- https://developer.adobe.com/commerce/php/development/components/events-and-observers/

Publish-Subscribe: Benefits

- Publishers and subscribers are independent and hence loosely coupled. Adding or changing subscribers requires only registering for an event and causes no changes to the publisher.
- System behavior can be easily changed by changing the event or topic of a message being published, and consequently which subscribers might receive and act on this message. This seemingly small change can have large consequences, as features may be turned on or off by adding or suppressing messages.
- Events can be logged easily to allow for record and playback and thereby reproduce error conditions that can be challenging to recreate manually.

Publish-Subscribe: Tradeoffs (1/2)

- Some implementations of the publish-subscribe pattern can negatively impact performance (latency). Use of a distributed coordination mechanism will ameliorate the performance degradation.
- In some cases, a component cannot be sure how long it will take to receive a published message. In general, system performance and resource management are more difficult to reason about in publish-subscribe systems.
- Use of this pattern can negatively impact the determinism produced by synchronous systems. The order in which methods are invoked, as a result of an event, can vary in some implementations.

Publish-Subscribe: Tradeoffs (2/2)

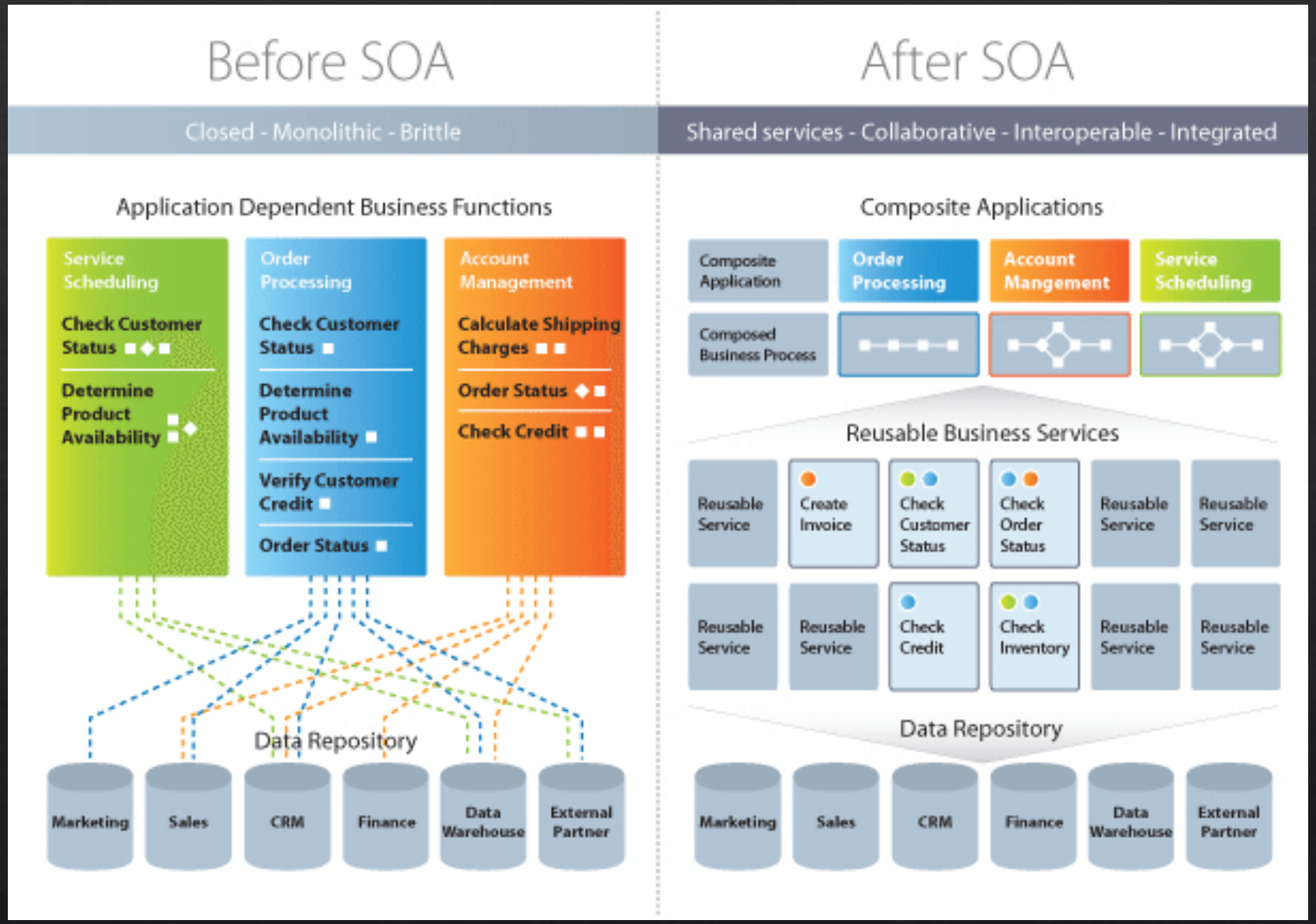
ตัวกลางสำเปลี่ยนแปลงเล็กน้อยอาจผลกระทบได้

- Use of the publish-subscribe pattern can negatively impact testability. Seemingly small changes in the event bus—such as a change in which components are associated with which events—can have a wide impact on system behavior and quality of service.
- Some publish-subscribe implementations limit the mechanisms available to flexibly implement security (integrity). Since publishers do not know the identity of their subscribers, and vice versa, end-to-end encryption is limited. Messages from a publisher to the event bus can be uniquely encrypted, and messages from the event bus to a subscriber can be uniquely encrypted; however, any end-to-end encrypted communication requires all publishers and subscribers involved to share the same key.

Service-Oriented Architecture

Service-Oriented Architecture

- The service-oriented architecture (SOA) pattern describes a collection of distributed components that provide and/or consume services.
- In an SOA, ***service provider*** components and ***service consumer*** components can use different implementation languages and platforms.
- Services are largely standalone entities: Service providers and service consumers are usually deployed independently, and often belong to different systems or even different organizations.
- Components have interfaces that describe the services they request from other components and the services they provide.
- SOAs provide reusable components that are assumed to be heterogeneous and managed by distinct organizations.



https://medium.com/sirisoft/eai-soa-esb-microservices-35ca49503d12

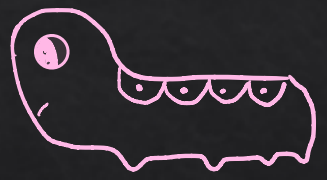
SOA: Benefits

- Services are designed to be used by a variety of clients, leading them to be more generic. Many commercial organizations will provide and market their service with the goal of broad adoption.
- Services are independent. The only method for accessing a service is through its interface and through messages over a network. Consequently, a service and the rest of the system do not interact except through their interfaces.
- Services can be implemented heterogeneously, using whatever languages and technologies are most appropriate.

SOA: Tradeoffs

- SOAs, because of their heterogeneity and distinct ownership, come with a great many interoperability features such as WSDL and SOAP. This adds complexity and overhead.

Representational State Transfer (REST)



request ไปที่ตรงนี้

Host ⇄ subsystem (re / content)

Representational State Transfer (REST)

- A software architectural style that describes a uniform interface between physically separate components, often across the Internet in a Client-Server architecture.
- The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation.
- In web development REST allows content to be rendered when it's requested, often referred to as Dynamic Content.

Client side script เช่น js
↳ เอาไว้ใช้ที่ฝั่ง client

- RESTful Dynamic content uses server-side rendering to generate a web site and send the content to the requesting web browser, which interprets the server's code and renders the page in the user's web browser.

เป็น html มาแล้ว เอาต่อมาจาก request → render เว็บ

- A web API that obeys the REST constraints is informally described as *RESTful*.

REST Architectural constraints

- Client–server architecture
- Statelessness vs State full
- Cacheability ควรเก็บ content ที่ควร cache / ไม่ควร
- Uniform interface
- Layered system N-tier client-server
- Code on demand (optional)

Applied REST to web services

หรือ URL

- A base URI, such as http://api.example.com/
- Standard HTTP methods (e.g., GET, POST, PUT, and DELETE) ตัวสั่งว่าจะทำอะไรกับ object statefull
- A media type that defines state transition data elements (e.g., Atom, microformats, application/vnd.collection+json, etc.). The current representation tells the client how to compose requests for transitions to all the next available application states. This could be as simple as a URI or as complex as a Java applet.

REST vs CRUD

CRUD	HTTP Method	Description
Create	POST	Let the target resource process the representation enclosed in the request.
Read / Retrieve	GET	Get a representation of the target resource's state.
Update	PUT	Create or replace the state of the target resource with the state defined by the representation enclosed in the request.
Delete / Destroy	DELETE	Delete the target resource's state.

Recommended Reading List

- https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf