

Software Architecture and Architectural Patterns/Styles

Parinya Ekparinya

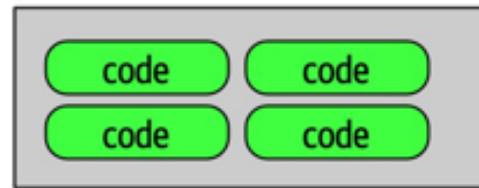
Parinya.Ek@kmitl.ac.th

Components

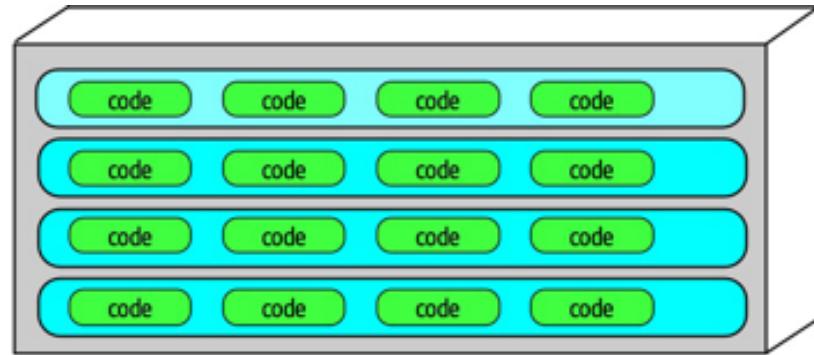
- ❖ Developers physically package modules in different ways, sometimes depending on their development platform. Generally, we may call physical packaging of modules **components**.
- ❖ Most languages support physical packaging as well: **jar** files in Java, **dll** in .NET, **gem** in Ruby, and so on.
- ❖ 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.

↳ នាយក modules របស់ខ្លួន

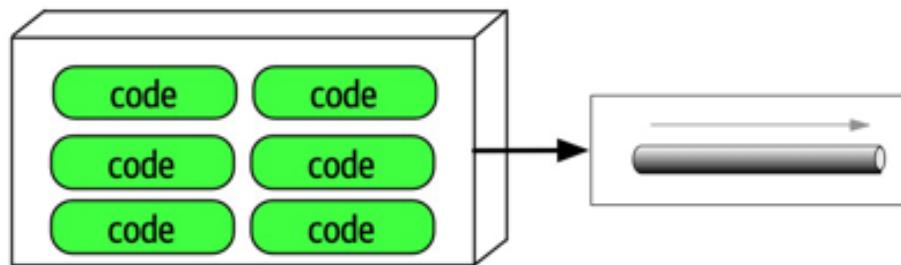
group ព័ត៌មានក្នុង



Wrapper of related code

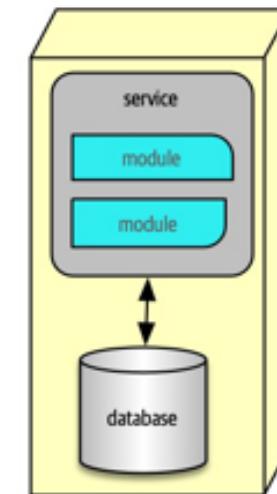


Layer or subsystem



Event processor

มีอีก ~



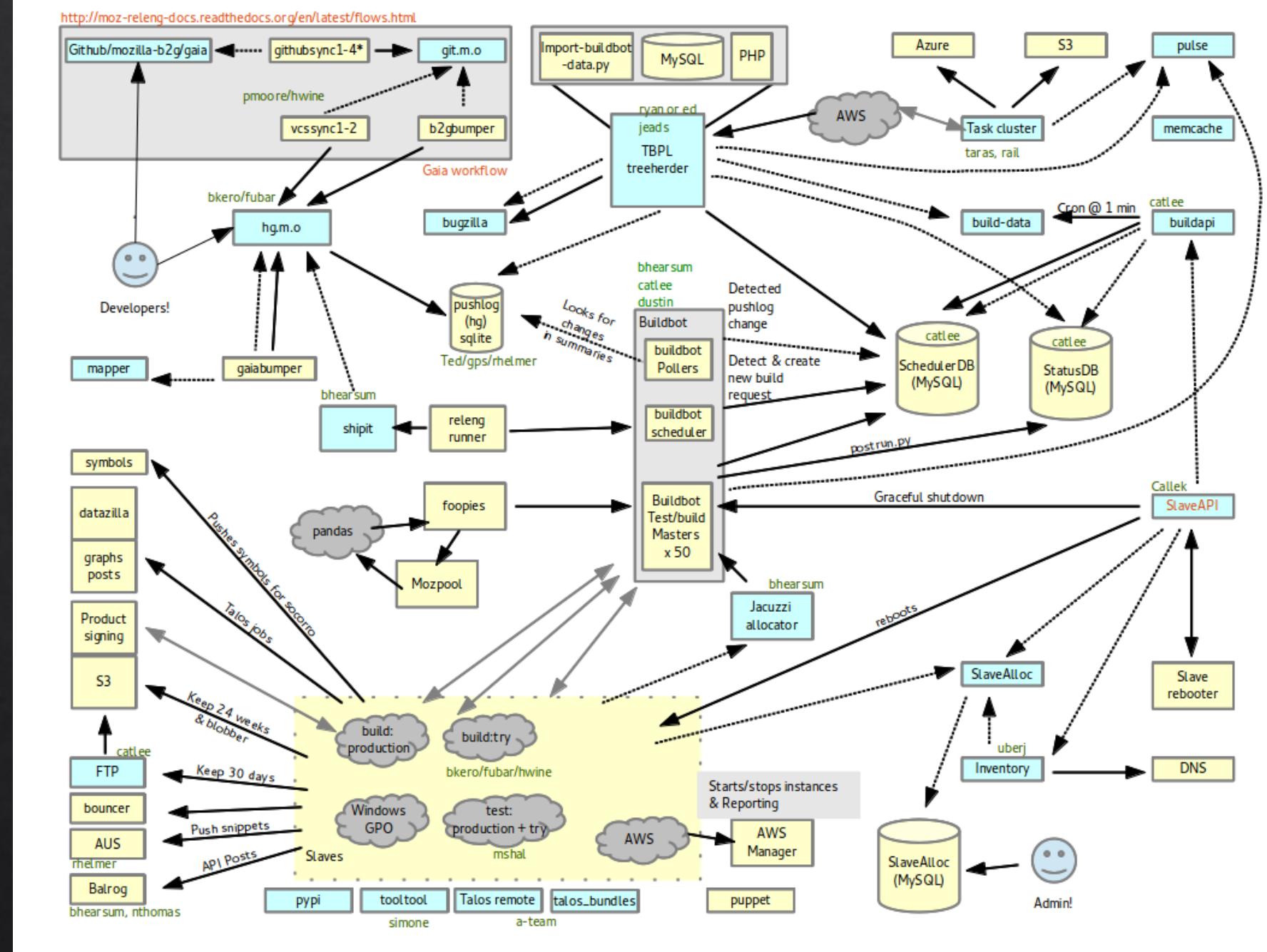
Distributed service

ເຕັມ

Architecture Partitioning

ໄດ້ຍ່າງ ເສຍອຍ່າງ

- ❖ 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.



၁၇၈

“

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

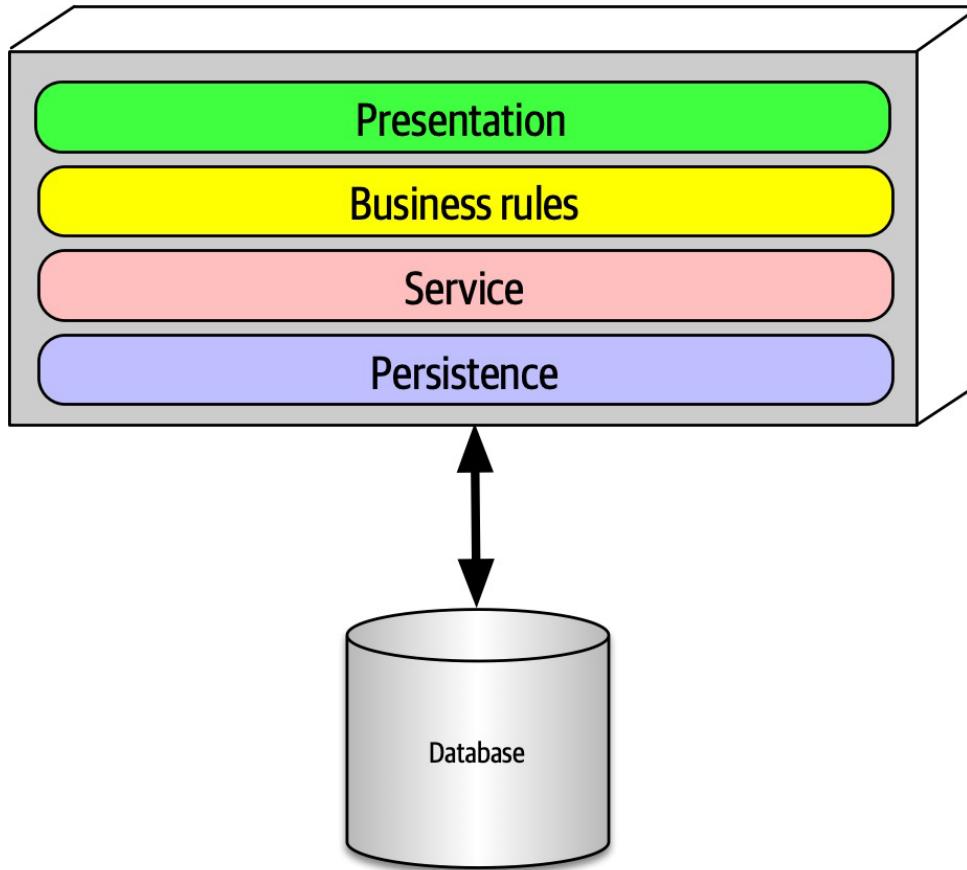
”

— Melvin E. Conway

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

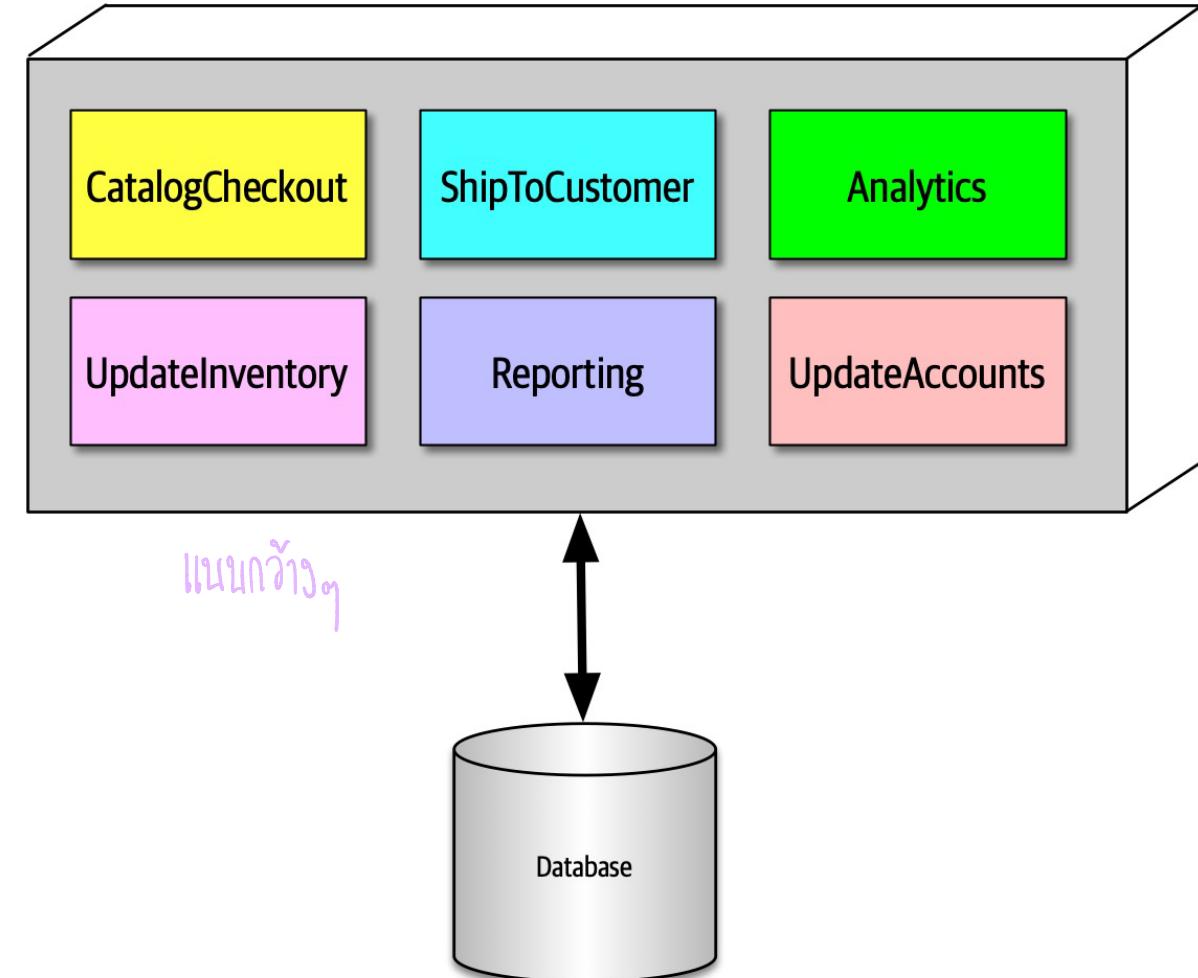
งานเพื่อนๆ กัน
OK

Technical partitioning แบ่งก้อน



ภูมิภาคตามมืออาชีวะเป็น Trend

Domain partitioning



Catalog of Architectural Patterns/Styles

- Monolithic** *ເນື້ອ component 1 ກົດນູນ*
- ❖ Layers *ໄໝມື່ກລືວອນຮາບທີ່ຈະ comp*
 - ❖ Plug-in (Microkernel) *ແກນເລັກຖານ (ຫຼັກຄົນ)*
 - ❖ Model-View-Controller (MVC)

- Distributed** *>1 comp, comp.ສ່ວນຮັກໜູນ*
- ❖ Client-Server & N-Tier
 - ❖ Publish-Subscribe
 - ❖ Service-Oriented Architecture (SOA)
 - ❖ Representational State Transfer (REST)

ສອດລົງກົນ DDD

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!! *ຢັງມີກຳກຳ*

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

Layers

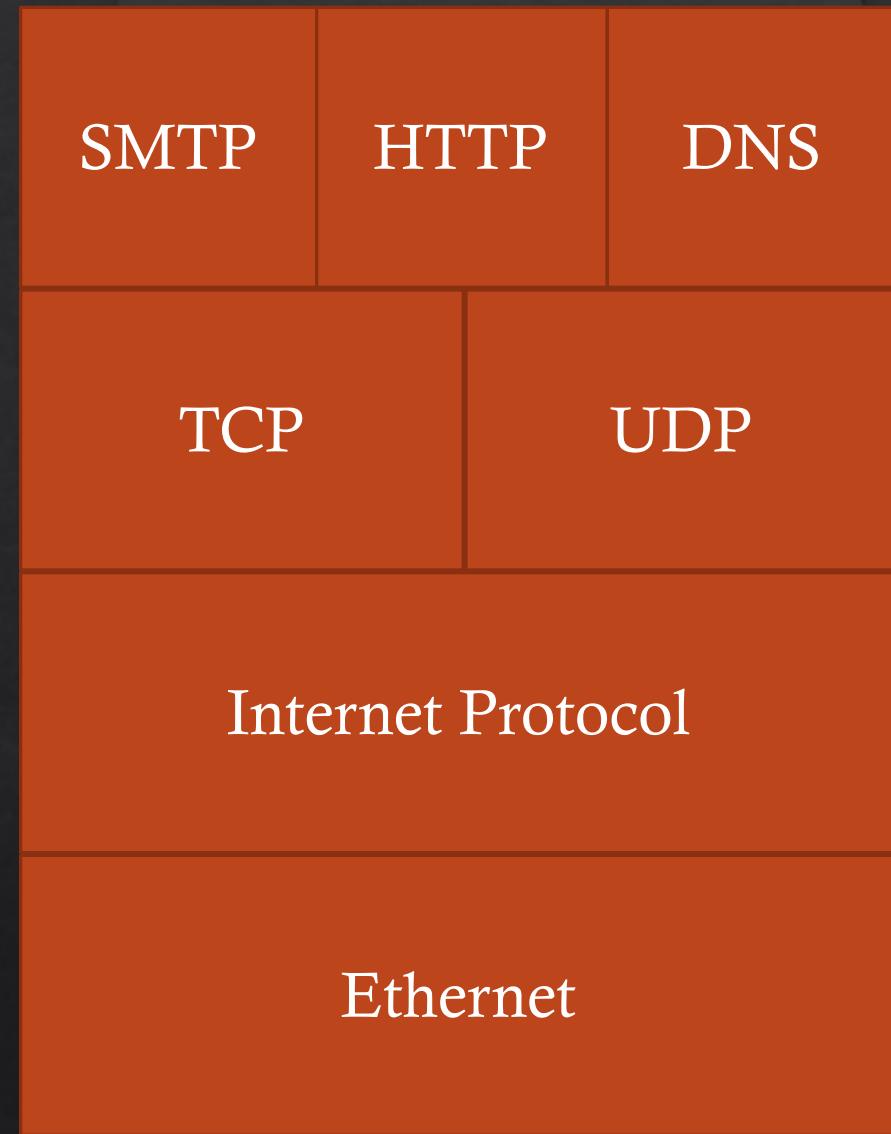
Layers → ແນ້ງໜານ ແຕ່ລະງານພື້ນາແຍກໄຟ

- ❖ 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. → ເຮັດວຽກ layer ທີ່ຕິດກັນໄດ້ ເຕັມ constraint
- ❖ Upward usages are not allowed in this pattern.

ນິ້ນໃຊ້ລ່າງໄດ້ only

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>

↳ for Android dev

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.
- ❖ 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. *One way ធន→វាំ*
- ❖ Because the allowed-to-use relations are constrained, the number of interfaces that any team must understand is reduced.

↳ deal រាយទីនេះ
B. ចូលចែងនៅឯណា ហំនេគិញស្ថាន

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. → ແບ່ງຫຼັນໄມ້ດູກ ທົ່ວລ່າງໄມ້ໃຈໜ້າຮ້າງເພື່ອທີ່ມີກຳນົດກຳນົດ ນັ້ນມີຄຳລ່າງກົມ A conflict ກຳນົດ
- ❖ 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. → ຍິ່ງເພີ່ມ performance ↓
ກະໂຄດທຳນາງ layer
- ❖ If many instances of layer bridging occur, the system may not meet its portability and modifiability goals, which strict layering helps to achieve.

B. Designໄມ້ຄື

Layered Architecture in DDD

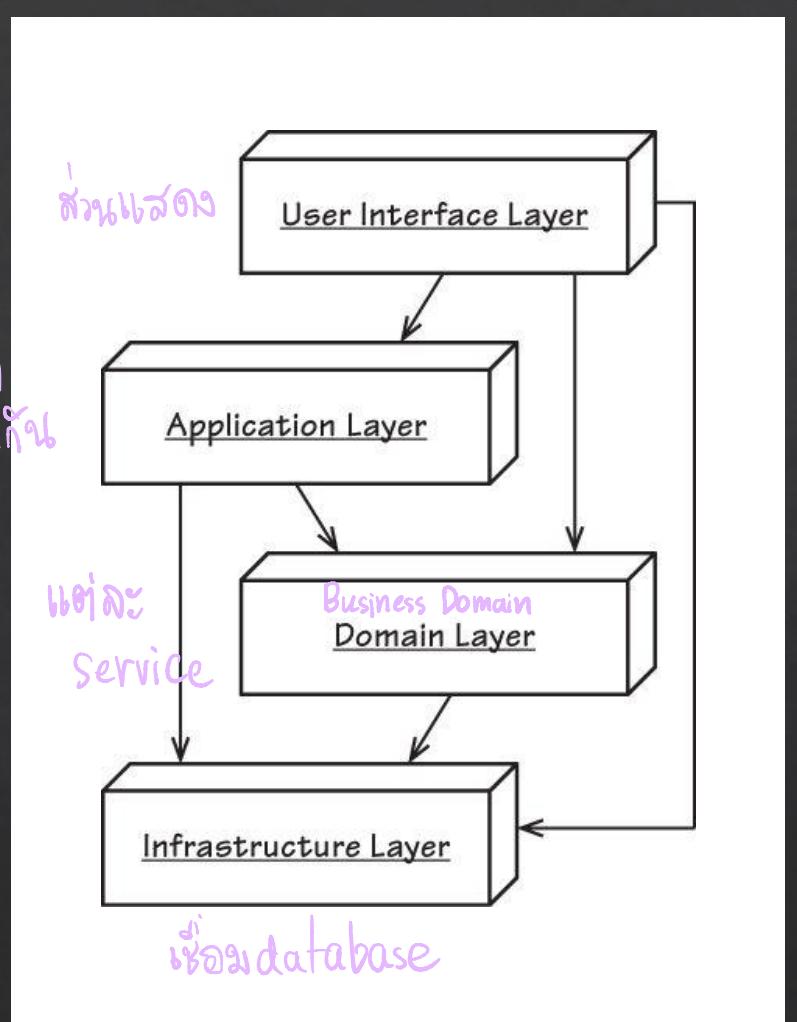
- ❖ Applying a traditional Layered Architecture, there are four layers common to a DDD application.
- ❖ The isolated Core Domain resides in one layer in the architecture. Above it are the **User Interface** and **Application Layers**. Below it is the **Infrastructure Layer**.

5 → service

ເວັບໄຊ
ການ
ກ່ຽວຂ້ອງ

ແບ່ງລະ
Service

Ex E-commerce



ឧបករណ៍ API ដីឡាបង្ហាញ

User Interface Layer → សែន display

- ❖ 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.

ឧបករណ៍
method API

Application Layer

កំឡត់ប្រាក់

- ❖ 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.
↳ នៅទេសទូទាត់បានពីលើកក្នុង UI
- ❖ 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.
↳ ក្នុង
- ❖ An Application Service may also use a Domain Service to fulfill some domain-specific task designed as a stateless operation.
↳ មិន

Infrastructure Layer database, blockchain

- ❖ In a traditional Layers Architecture the Infrastructure is at the bottom. Things like persistence and messaging mechanisms reside there. → *storage*
- ❖ 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.

Plug-in (Microkernel)

(extension)

Plug-in (Microkernel) (1/2)

កំណត់ផ្លូវការក្នុងកំណត់
ដែលជាកម្មកុង-កែ
ដើម្បីក្នុងកំណត់ផ្លូវការ

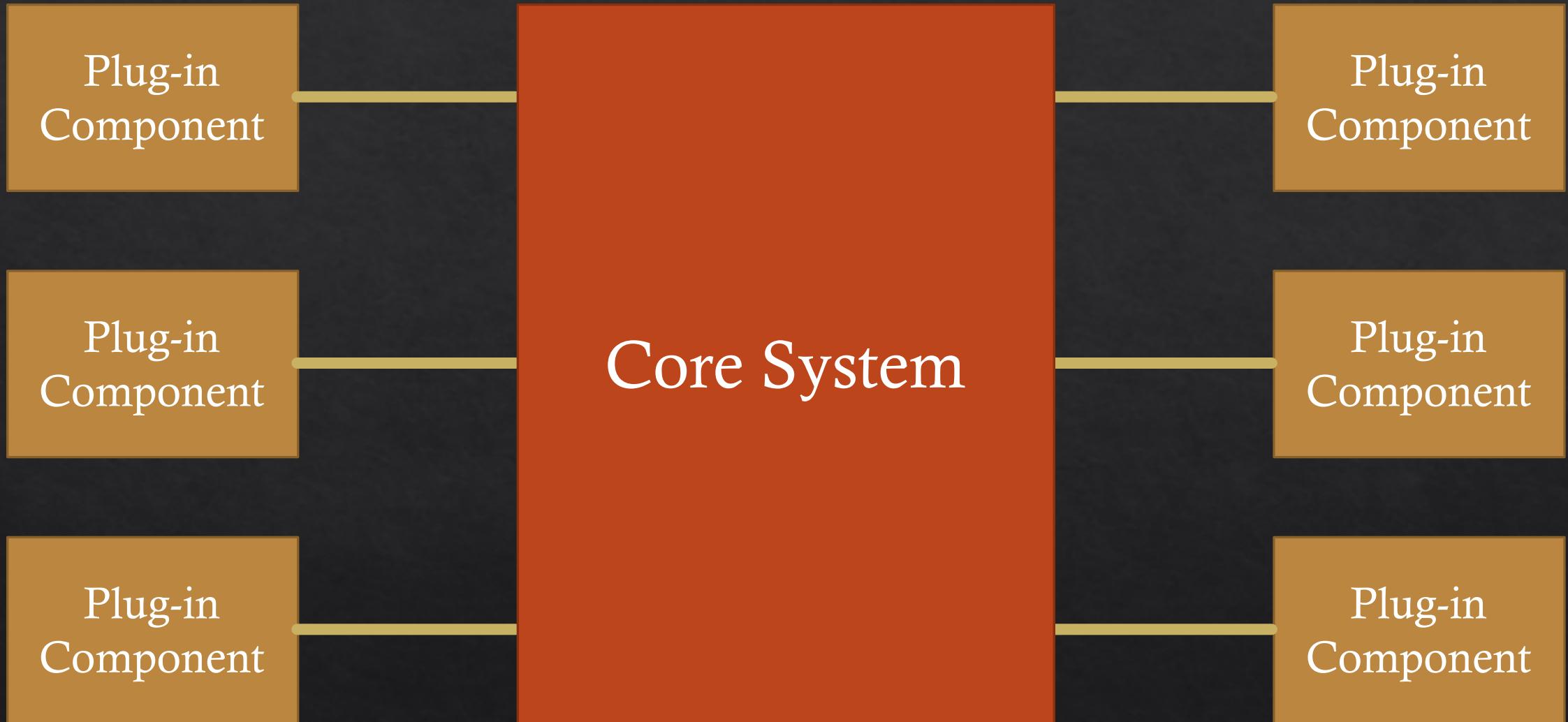
- ◆ The plug-in pattern has two types of elements—elements that provide a **core set of functionality** and **specialized variants** (called plug-ins) that add functionality to the core via a fixed set of interfaces.
- ◆ The two types are typically bound together at build time or later.

→ នៅនឹងក្នុងកំណត់ផ្លូវការ initial (add ឬលើ restart)

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.
Now នាយករោង
- ❖ 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.



Microkernel: Real-World Example

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

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. → กำหนด interface ตัว
- ❖ 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. ยกเว้น core ที่เปลี่ยน

Microkernel: Tradeoffs

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

Model-View-Controller (MVC)

ເນື້ອໃຈກົບຮະບູບເທົ່າລວມຍຸ້ນ

≈ Entity ใน database / ช่วง display (html,css)

ห้อง Web

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

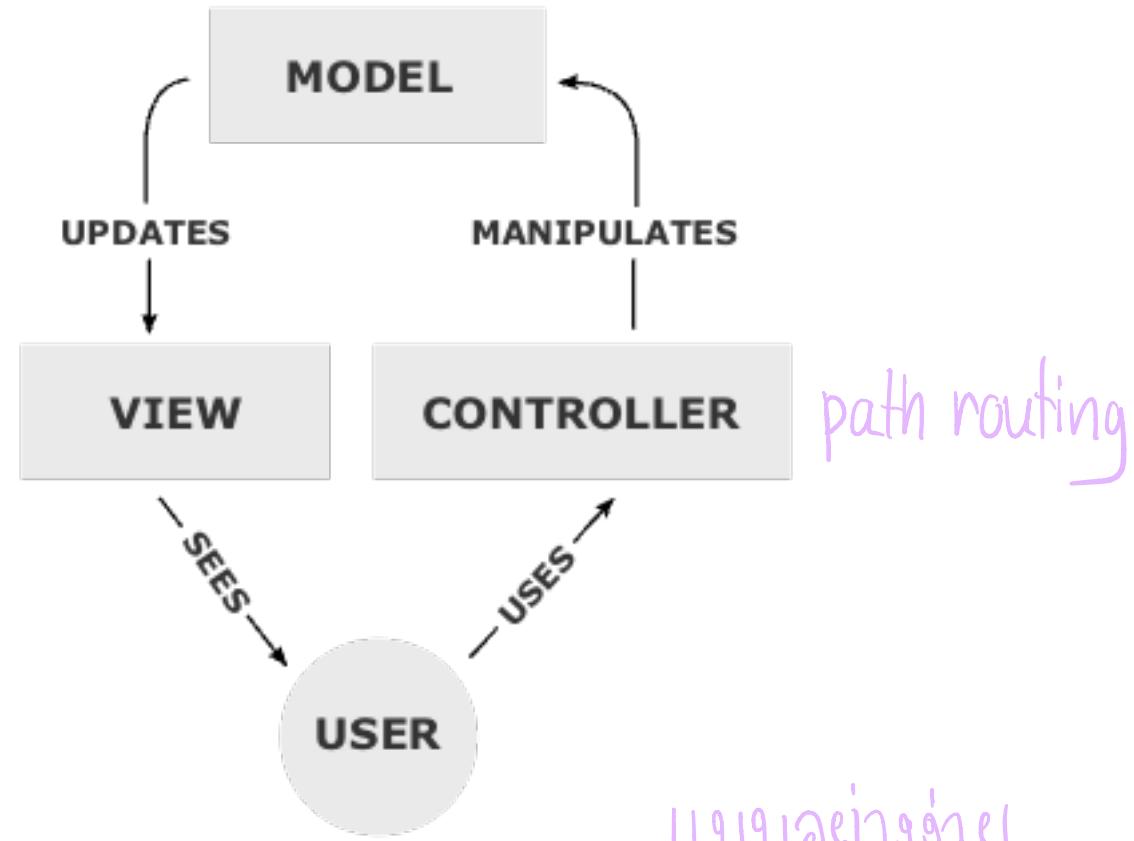
รับ input

- ❖ 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). >1 process
- ❖ If the MVC is in one process, then the updates are sent using the observer pattern (discussed in another lecture). design pattern (ມອນເທິງໃນ code)
- ❖ If the MVC is distributed across processes, then the publish subscribe pattern is often used to send updates. architural pattern



Real-World Example

- ❖ <https://developer.apple.com/library/archive/documentation/General/Conceptual/Devpedia-CocoaCore/MVC.html>

កម្រិត និង លក្ខណៈ

- ❖ 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.
ผลที่ตามมา
 - ❖ 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.
ทำงานแยกกันได้ เต็มที่คุยกัน
 - ❖ A model can be used in systems with different views, or a view might be used in systems with different models.

ວິໄລຄ່ອຍເຈວ

MVC: Tradeoffs

ເຜົ່ານກາຮະ

- ❖ 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. B. ນໍ້າຂລາຍ view
- ❖ For simple UIs, MVC adds up-front complexity that may not pay off in downstream savings.

Client-Server & N-Tier

browser - web server

any cache

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:

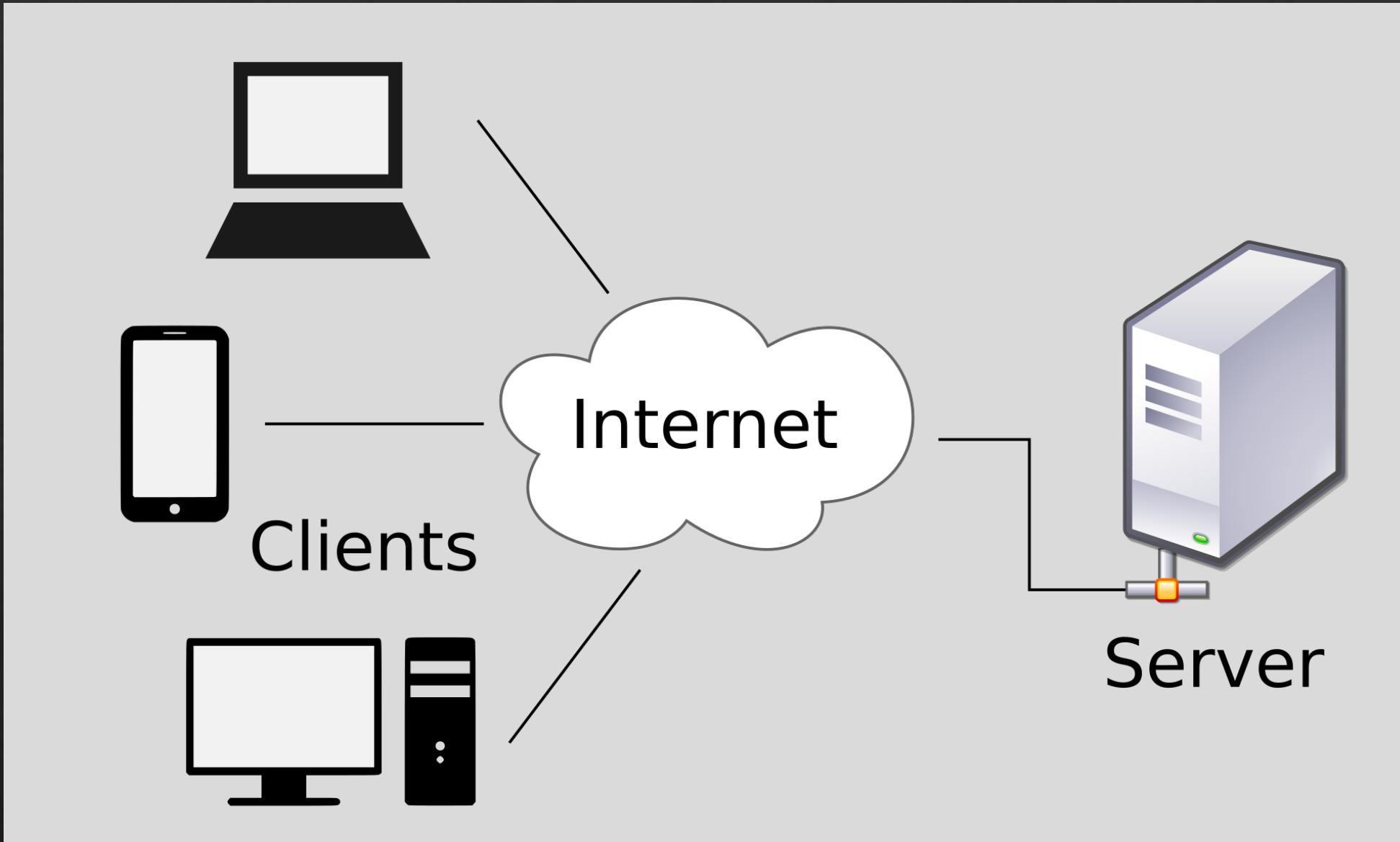
- ❖ Discovery: client หา server
 - ❖ Communication is initiated by a client, which uses a discovery service to determine the location of the server.
 - ❖ The server responds to the client using an agreed-upon protocol. → ตอนกลับ + นอกใจ ใช้ protocol อื่น?
- ❖ Interaction:
 - ❖ The client sends requests to the server.
 - ❖ The server processes the requests and responds.

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

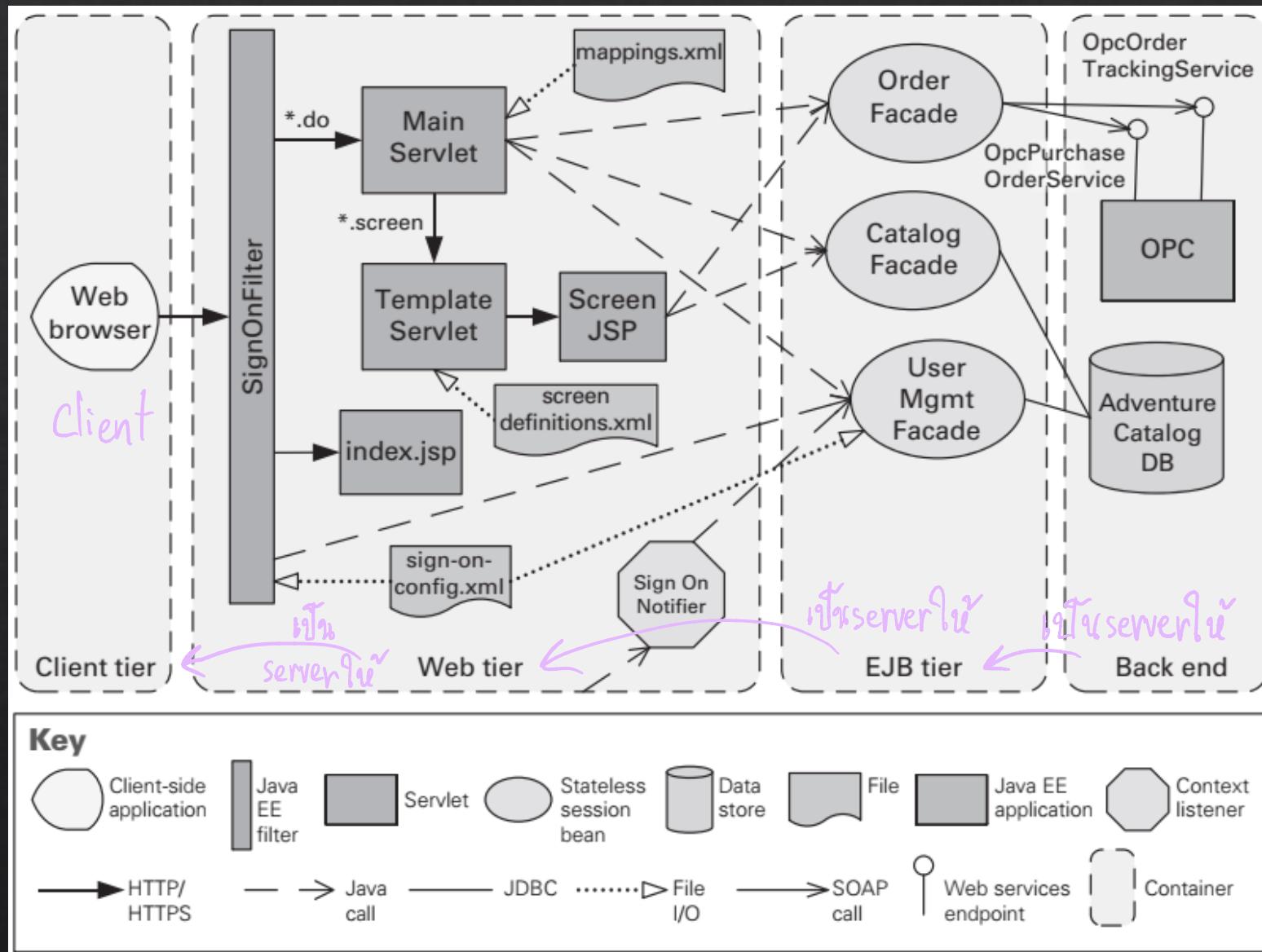
load balance

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. client timeout
 - ❖ The client should send an “end of session” message so that the server can remove resources associated with that particular client. → server state 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.



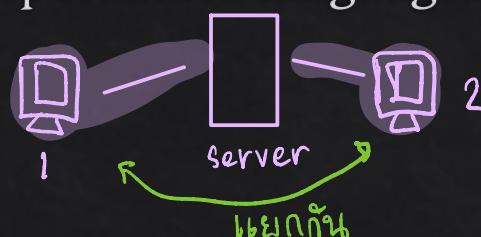
N-tier
client, sever
 klient server



Client-Server: Benefits

(server គឺមែនជោគជ័យ)
កំណត់ទៅលើការងារ

- ❖ The connection between a server and its clients is established dynamically. → កំណត់ទៅលើការងារ
- ❖ The server has no prior knowledge of its clients; that is, there is low coupling between the server and its clients.
- ❖ There is no coupling among the clients. ជីវិះជាន់ client
- ❖ 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. កំណត់
- ❖ 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.**

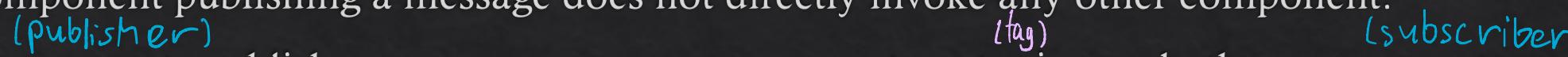


voIP network

Publish-Subscribe

Publish-Subscribe - ບາຄັ້ນຕົວກລາງ

ໄມ່ເກີດຕ່ອນ້ອງ / ພ່ອມກົນ

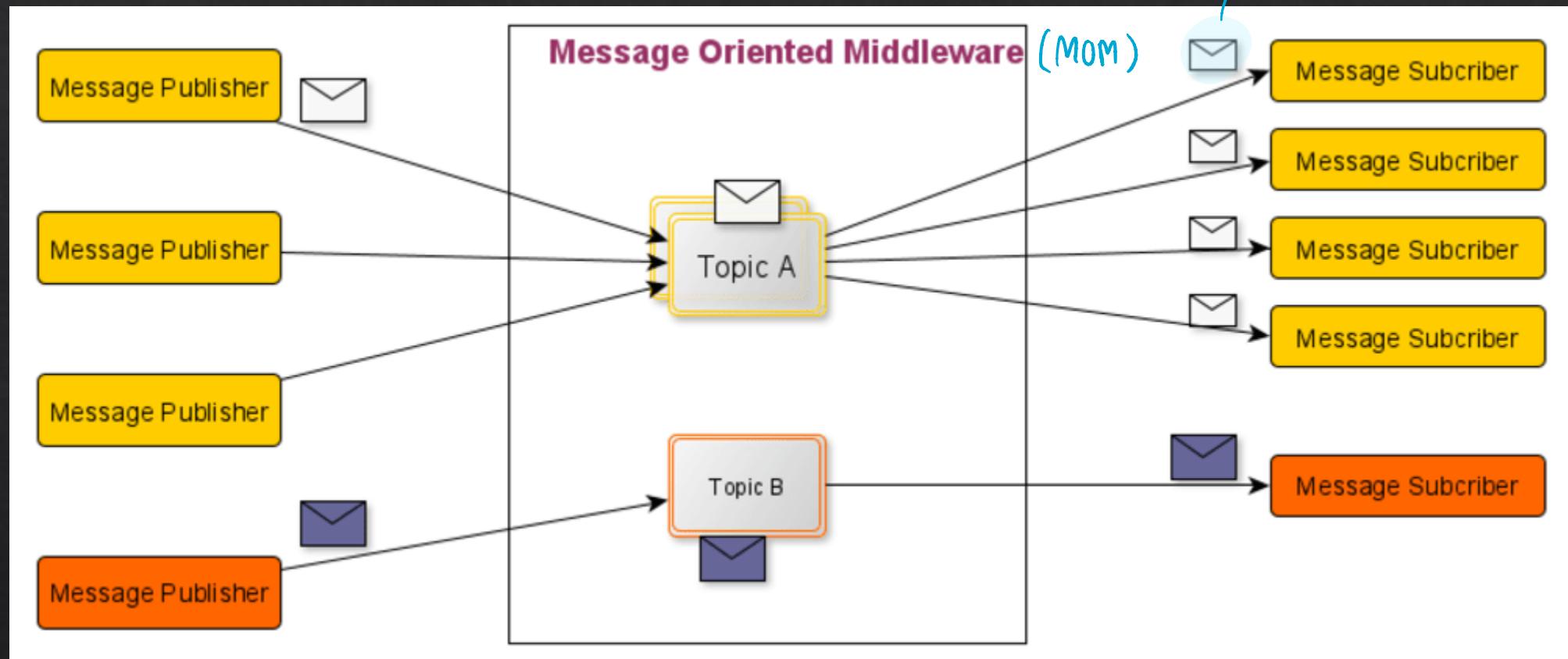
- ◆ Publish-subscribe is an architectural pattern in which components communicate primarily through asynchronous messages, sometimes referred to as “events” or “topics.”
- ◆ The publishers have no knowledge of the subscribers, and subscribers are only aware of message types. ຕະລາງ message type
- ◆ Systems using the publish-subscribe pattern rely on implicit invocation; that is, the component publishing a message does not directly invoke any other component.

(publisher) t (tag) \rightarrow (subscriber)
- ◆ 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: ≈Youtube ,Twitter(ຕົດໜາວັດຳ)

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

ໄມ່ລະວົງໄດ້ຮັບແລ້ວທໍາອັນໄວ



Real-World Example

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

Publish-Subscribe: Benefits

ព័ត៌មាន publisher, subscriber និង
topic នៃការប្រើប្រាស់

- ◆ 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.

များမျှမှတ်စွာ

Publish-Subscribe: Tradeoffs (2/2)

ເພື່ອຍນີ້ຕົວລາງ ສຳຜລແຍອະ ໄລົມທຶນໜີ້

- ❖ 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.

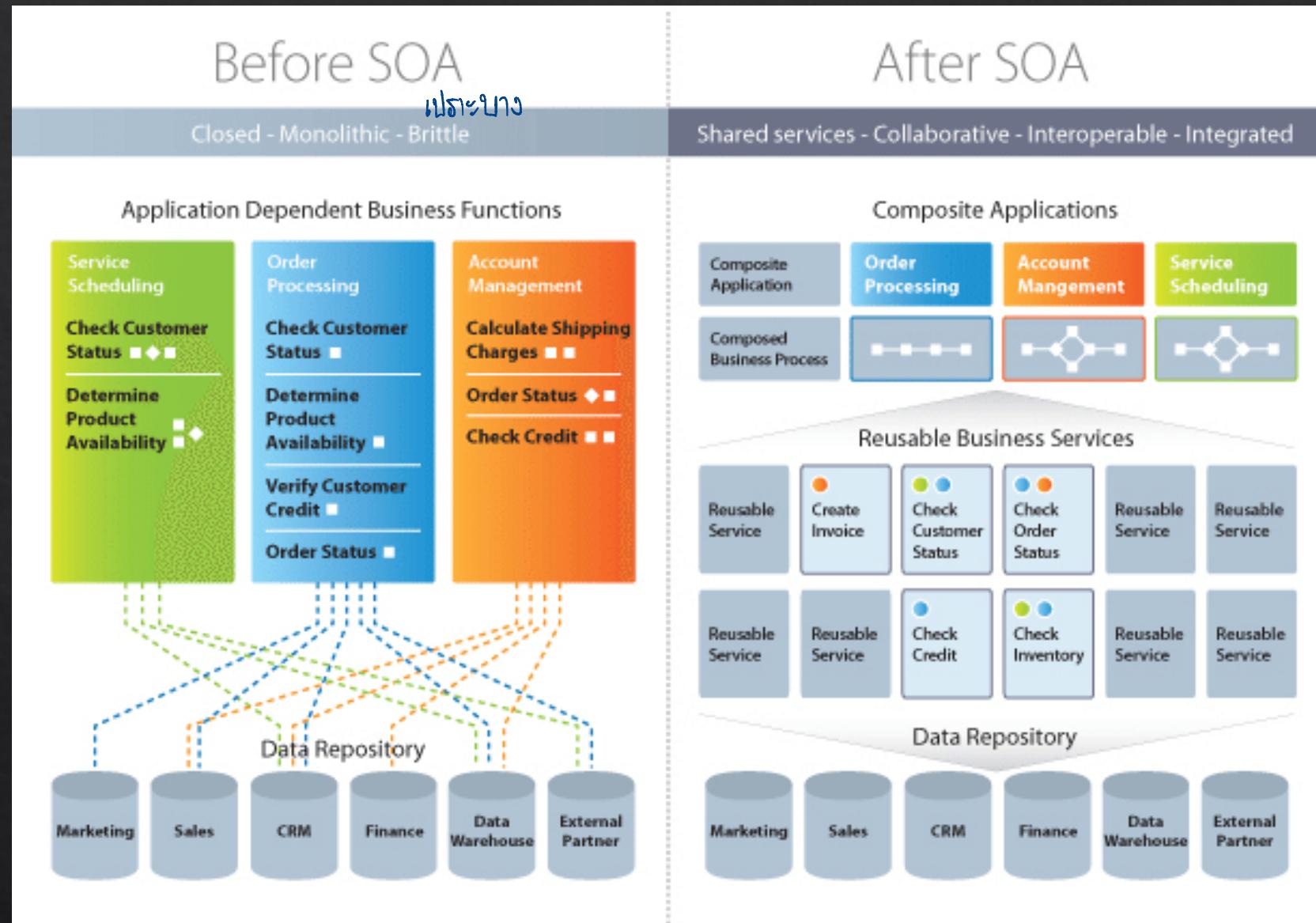
ບ.ຕົວລາງຕ້ອງເຫັນ ປູ້ອາວຄູດໄດ້

Service-Oriented Architecture (SOA)

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.

ສ່ວນລາງ
Enterprise
Service
Bus (ESB)



Layer
App Layer
Domain Layer
Infrastructure Layer

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.

ນຳມານ່ວຍກົດ

* Representational State Transfer (REST)

client side script - ស្នើសារពី client

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. វិទ្យានិពន្ធដែលបានរៀបចំឡើងដោយលោក Roy Fielding
- ❖ In web development REST allows content to be rendered when it's requested, often referred to as Dynamic Content.
render បែនាំង html ដោយផ្តល់សង្ឃគម្រោះទៅ 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.
- ❖ A web API that obeys the REST constraints is informally described as *RESTful*.

REST Architectural constraints

- ❖ Client–server architecture
- ❖ Statelessness សែប្បីពីរដោយ request
- ❖ Cacheability
- ❖ Uniform interface
- ❖ Layered system \approx N-tier
- ❖ Code on demand (optional)

Applied REST to web services

- ❖ A base URI, such as `http://api.example.com/`
- ❖ Standard HTTP methods (e.g., GET, POST, PUT, and DELETE)
- ❖ 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