Software Design

Week 08

ดัดแปลงจาก slides ของหนังสือ Software Engineering [1]

หัวข้อที่จะศึกษา

- Architectural design decisions
- Architectural views
- Architectural patterns
- Application architectures

Architectural design

- Architectural design (การออกแบบทางสถาปัตยกรรม) เกี่ยวข้องกับ
  - การจัดองค์ประกอบของระบบซอฟต์แวร์
  - การออกแบบโครงสร้างโดยรวมของระบบดังกล่าว
- การออกแบบทางสถาปัตยกรรมเป็นจุดเชื่อมโยงที่สำคัญระหว่างวิศวกรรมการออกแบบ (design engineering) และวิศวกรรมความต้องการ (requirement engineering)
  - เป็นการกำหนดองค์ประกอบโครงสร้างหลักในระบบ
  - เป็นการกำหนดหรืออธิบายความสัมพันธ์ระหว่างกัน
- ผลลัพธ์ของกระบวนการออกแบบสถาปัตยกรรมคือ
  - การอธิบายว่าระบบถูกจัดองค์ประกอบโครงสร้างอย่างไร
  - มีการสื่อสารระหว่างองค์ประกอบเหล่านั้นอย่างไร

ความเกี่ยวข้องระหว่าง agile และ architecture

- มีการยอมรับโดยทั่วไปว่า ขั้นแรกๆ ของกระบวนการ agile คือการออกแบบสถาปัตยกรรมของระบบ
  - Agile ต้องการความรวดเร็ว ซึ่งการออกแบบสถาปัตยกรรมที่ดี จะช่วยให้ระบบสอดคล้องกับวัตถุประสงค์ของ agile
- การปรับเปลี่ยนสถาปัตยกรรมของระบบจะทำให้เกิดต้นทุนที่สูง เนื่องจากกระทบต่อองค์ประกอบย่อย ๆ ของระบบ
  - Agile เน้นความกระชับ การตัดสิ่งที่ไม่จำเป็นออกไปให้มากที่สุดจะทำได้เมื่อทีมพัฒนาได้เห็นสถาปัตยกรรมโดยรวมของระบบ

Architectural abstraction

- สถาปัตยกรรมในระบบขนาดเล็ก
  - เกี่ยวข้องกับสถาปัตยกรรมของแต่ละโปรแกรม
  - ในระดับนี้เราให้ความสำคัญกับวิธีการนำองค์ประกอบต่าง ๆ มารวมนเป็นโปรแกรม
- สถาปัตยกรรมในระบบขนาดใหญ่
  - เกี่ยวข้องกับสถาปัตยกรรมของระบบที่ซับซ้อนมาก
  - อาจรวมเอาโปรแกรมต่าง ๆ , ระบบอื่น ๆ หรือแม้กระทั่งระบบจากต่างหน่วยงาน เข้าไว้ในระบบเดียวกัน

Advantages of explicit architecture

- การสื่อสารของผู้มีส่วนได้ส่วนเสีย
  - สถาปัตยกรรมสามารถใช้เพื่อเป็นสิ่งควบคุมประเด็นในการอภิปรายโดยผู้มีส่วนได้เสียของระบบ (focus)
- การวิเคราะห์ระบบ
  - สามารถช่วยในการวิเคราะห์ว่าระบบสามารถตอบสนอง non-functional requirements ได้หรือไม่
- สามารถนำกลับมาใช้ใหม่ (reuse) ได้ในปริมาณมาก
  - สถาปัตยกรรมเผยให้เห็นสิ่งที่สามารถนำมา reuse ในส่วนต่าง ๆ ของระบบ
  - อาจมีการพัฒนาสถาปัตยกรรมที่เอื้อต่อการผลิตแบบ line การผลิตของโรงงาน

การวาดแผนผังสถาปัตยกรรมแบบต่าง ๆ

- Block diagram แบบง่าย ๆ และไม่เป็นทางการ
  - ใช้แสดงถึงเอนทิตีและความสัมพันธ์
  - ใช้บ่อยที่สุดสำหรับการจัดทำเอกสารสถาปัตยกรรมซอฟต์แวร์
- แต่อาจจะถูกปฏิเสธกาใช้งาน เนื่องจา
  - ไม่มีความหมายที่ชัดแจ้ง
  - ไม่แสดงประเภทของความสัมพันธ์ระหว่างเอนทิตี
  - ไม่แสดงคุณสมบัติที่มองเห็นได้ชัดของเอนทิตี

Box and line diagrams

- มีความเป็นนามธรรมสูง
  - ไม่แสดงลักษณะของความสัมพันธ์ขององค์ประกอบ
  - ไม่แสดงรายละเอียดของระบบย่อย
- มีประโยชน์เมื่อ
  - สื่อสารกับผู้มีส่วนได้ส่วนเสีย
  - วางแผนโครงการ

Use of architectural models

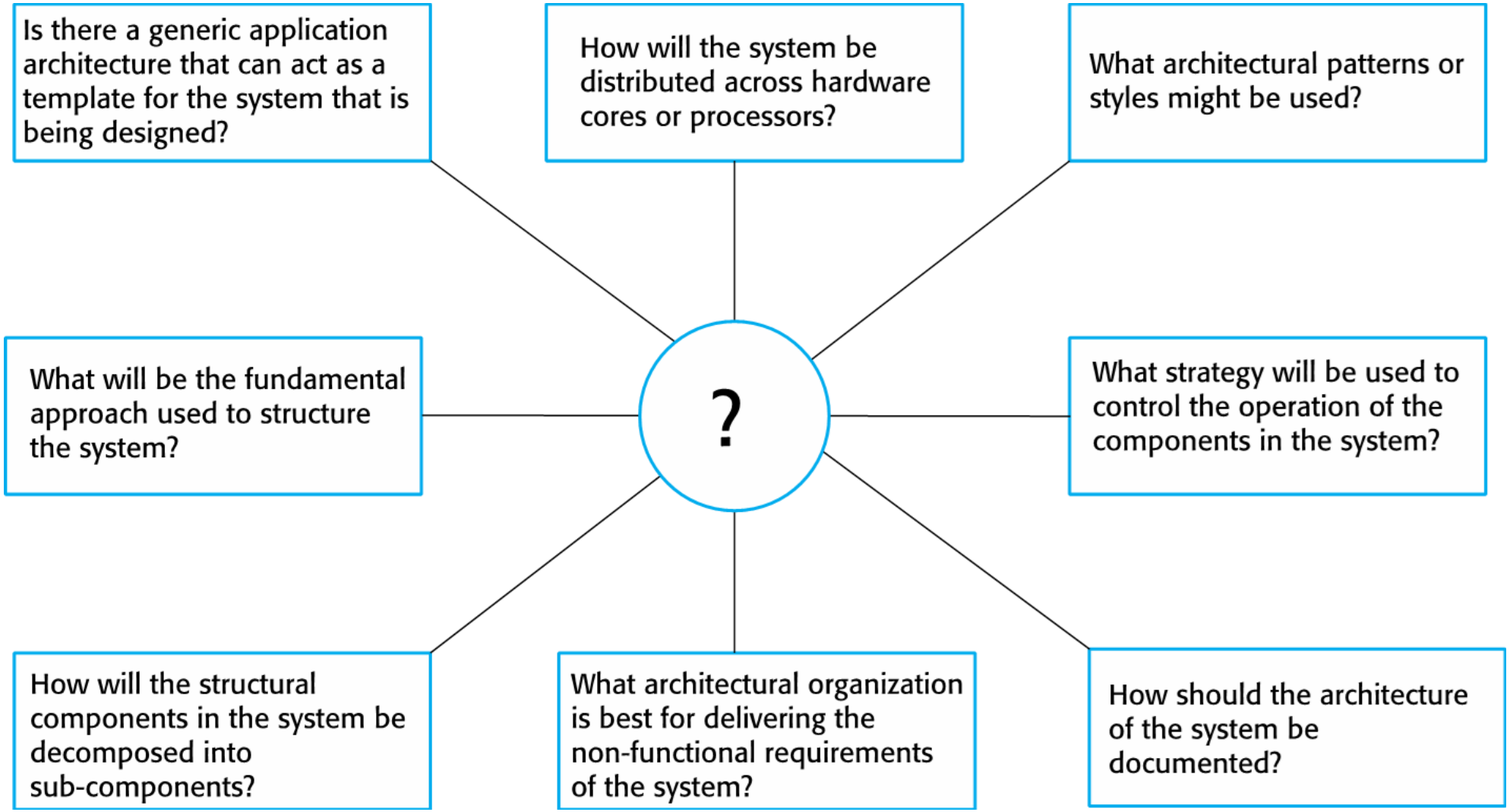
- เพื่ออำนวยความสะดวกในการอภิปรายเกี่ยวกับการออกแบบระบบ
  - มุมมองในระดับบนของระบบจะเป็นประโยชน์สำหรับการสื่อสารกับ stakeholders และการวางแผนโครงการ เนื่องจากไม่มีรายละเอียดที่รกรุงรัง
  - Stakeholders สามารถทำความเข้าใจเกี่ยวกับมุมมองที่เป็นนามธรรมของระบบ และสามารถพูดคุยเกี่ยวกับระบบโดยรวม ไม่ต้องสับสนกับรายละเอียด
- ช่วยอำนวยความสะดวกในการจัดทำเอกสารสถาปัตยกรรม
  - จุดมุ่งหมายหลักของเอกสารคือการสร้างแบบจำลองระบบที่มีองค์ประกอบต่าง ๆ อย่างสมบูรณ์ พร้อมทั้งแสดงการเชื่อมต่อระหว่างองค์ประกอบเหล่านั้น

Architectural design decisions

Architectural design decisions

- การออกแบบทางสถาปัตยกรรมเป็นกระบวนการที่สร้างสรรค์สิ่งที่ยังไม่มี
  - ดังนั้นกระบวนการจึงแตกต่างกัน ขึ้นอยู่กับชนิดของระบบที่กำลังพัฒนา
- อย่างไรก็ตามการตัดสินใจร่วมกันจากทุกฝ่ายที่เกี่ยวข้อง จะครอบคลุมกระบวนการออกแบบทั้งหมด
  - การตัดสินใจเหล่านี้ส่งผลต่อคุณลักษณะที่เป็น non-functional ของระบบ

Architectural design decisions



Architecture reuse

- ระบบในโดเมนเดียวกัน มักมีสถาปัตยกรรมที่คล้ายคลึงกัน ซึ่งสะท้อนแนวคิดของโดเมน
- โดยทั่วไป application มักจะถูกพัฒนาขึ้นจากสถาปัตยกรรมที่เป็น core หลัก
  - แล้วทำการปรับเปลี่ยนให้มีรูปแบบเฉพาะที่ตอบสนองความต้องการของลูกค้า
- สถาปัตยกรรมของระบบอาจได้รับการออกแบบจากสถาปัตยกรรมอย่างใดอย่างหนึ่งหรือแบบผสมผสาน
  - วิธีการออกแบบสถาปัตยกรรม เรียกว่า pattern ซึ่งเป็นรูปแบบมาตรฐาน

Architecture and system characteristics

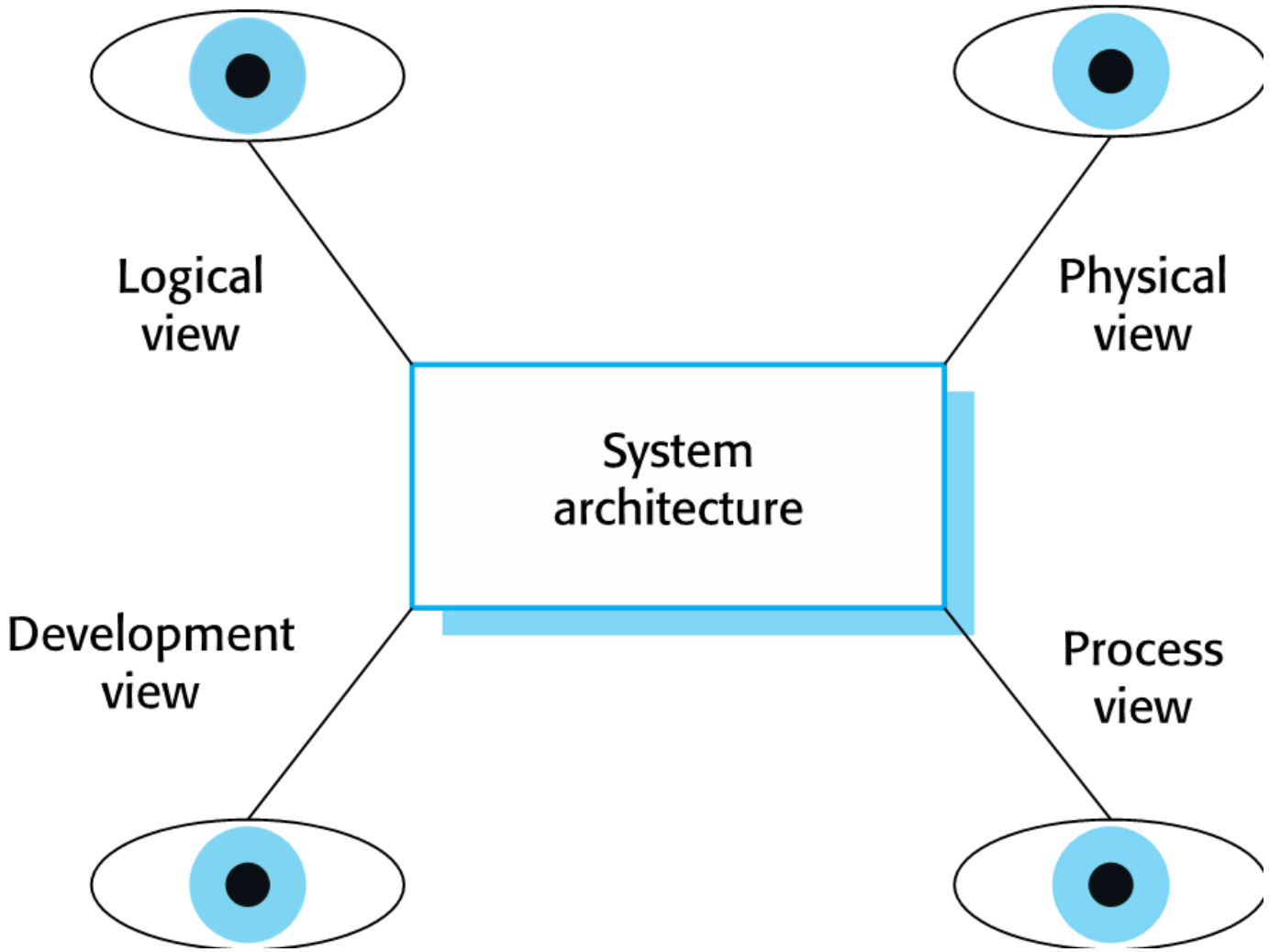
- ประสิทธิภาพ (Performance)
  - ใช้ component ที่มีขนาดใหญ่จะดีกว่า component ขนาดเล็ก
- ความมั่นคง (Security)
  - ใช้สถาปัตยกรรมแบบลำดับชั้น (layered) โดยนำส่วนสำคัญกว่าไปไว้ด้านใน
- ความปลอดภัย (Safety)
  - วางตำแหน่งส่วนที่เกี่ยวกับความปลอดภัยไว้ในตำแหน่งที่เหมาะสม และให้อยู่ในระบบย่อยที่มีจำนวนจำกัด
- ความพร้อมใช้งาน (Availability)
  - อาจจะรวมถึงระบบสำรอง (redundant) และวิธีการรับมือกับความผิดพลาด (fault tolerance)
- การบำรุงรักษา (Maintainability)
  - ใช้ components ขนาดเล็กที่สลับสับเปลี่ยนได้

Architectural views

Architectural views

- มุมมองใดมีประโยชน์เมื่อออกแบบและจัดทำเอกสารสถาปัตยกรรมของระบบ?
- ควรใช้ notation อะไรในการอธิบายรูปแบบสถาปัตยกรรม?
- แต่ละรูปแบบสถาปัตยกรรม จะแสดงมุมมองหรือมุมมองหนึ่งของระบบเท่านั้น
  - อาจแสดงให้เห็นว่าระบบสามารถแยกเป็นโมดูลย่อยได้อย่างไร
  - ในขณะที่ระบบทำงาน มีวิธีการทำงานร่วมกันของระบบต่าง ๆ อย่างไร
- โดยปกติ ในการออกแบบและทำเอกสารประกอบ เราสามารถนำเสนอมุมมองของสถาปัตยกรรมซอฟต์แวร์ได้อย่างหลากหลาย

Architectural views



4 + 1 view model of software architecture

- มุมมองตรรกะ (logical view)
  - แสดง abstractions สำคัญในระบบเป็นวัตถุหรือ class ของวัตถุ
- มุมมองกระบวนการ (process view)
  - แสดงให้เห็นว่าในขณะดำเนินการระบบประกอบด้วยกระบวนการโต้ตอบอย่างไร
- มุมมองการพัฒนา (development view)
  - แสดงให้เห็นว่าซอฟต์แวร์ถูกแบ่งแยกเพื่อการพัฒนาได้อย่างไร
- มุมมองทางกายภาพ (physical view)
  - แสดงฮาร์ดแวร์ระบบและส่วนประกอบของซอฟต์แวร์ที่กระจายอยู่บนโปรเซสเซอร์ต่าง ๆ ของระบบ
- ความเกี่ยวข้องโดยใช้ use cases หรือ scenarios (+1)

Representing architectural views

- นักพัฒนาเห็นพ้องกันว่า Unified Modeling Language (UML) เป็นสัญกรณ์ (notation) ที่เหมาะสมสำหรับการอธิบายและการจัดทำเอกสารเกี่ยวกับสถาปัตยกรรมระบบ
  - อย่างไรก็ตามมันอาจจะไม่เป็นเช่นนั้น เนื่องจาก UML ไม่เหมาะกับการเขียน abstractions ที่เหมาะสมสำหรับคำอธิบายระบบในระดับสูง
- มีการพัฒนาภาษาคำอธิบายเกี่ยวกับสถาปัตยกรรม (Architectural description languages : ADLs) แต่ยังไม่ได้ใช้กันอย่างแพร่หลาย

Architectural patterns

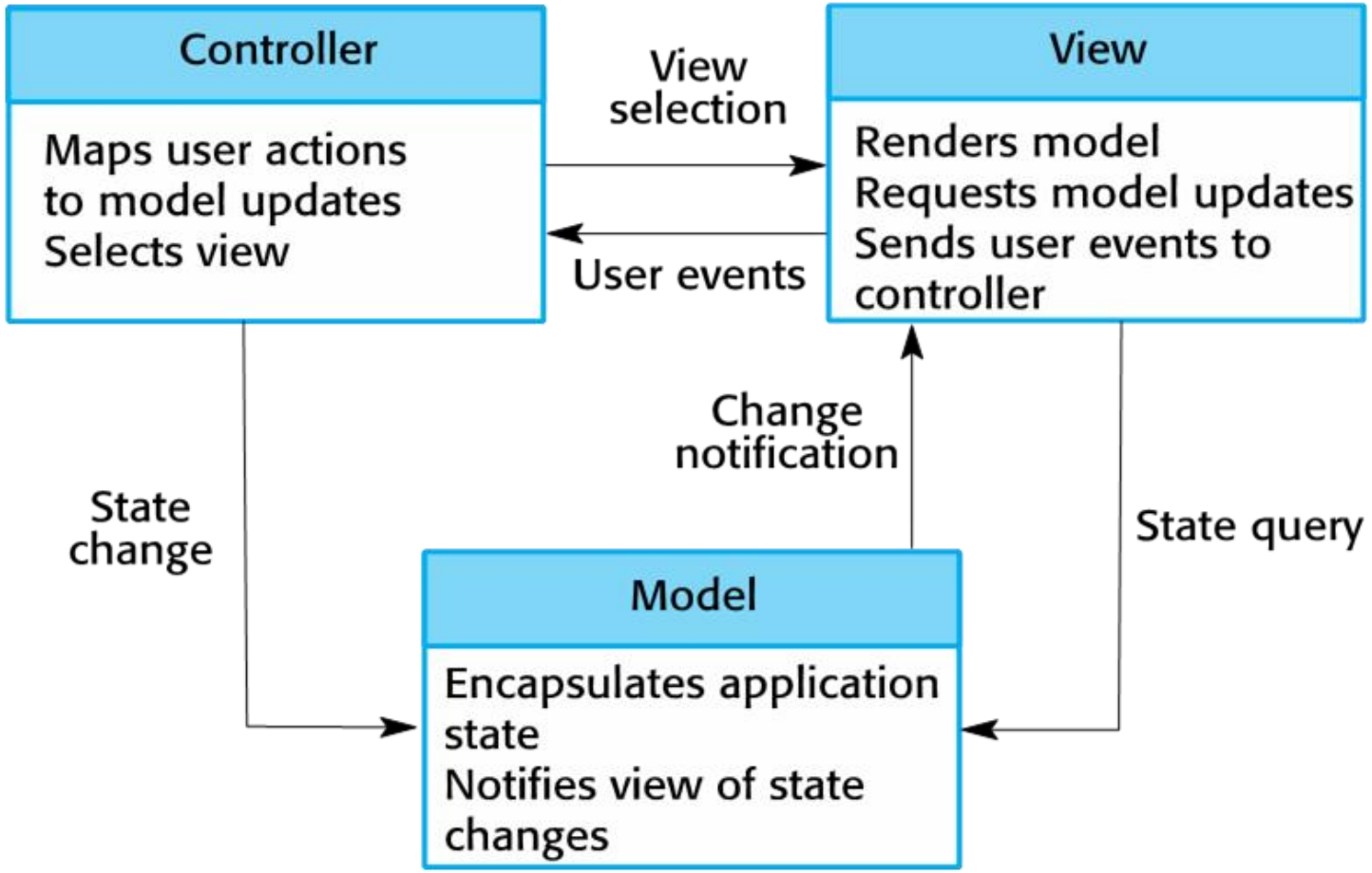
Architectural patterns

- Pattern หมายถึงการนำเสนอ (representing) การแบ่งปันและการนำความรู้กลับมาใช้ใหม่
- Architectural pattern เป็นคำอธิบายที่มีรูปแบบที่ดีในการออกแบบ ซึ่งได้รับการทดลองใช้และทดสอบในสภาพแวดล้อมที่แตกต่างกัน
- Patterns ควรมีข้อมูลที่บอกให้รู้ว่าเมื่อใดที่ควรใช้และเมื่อใดที่ไม่มีประโยชน์
- Patterns อาจแสดงโดยใช้คำอธิบายแบบตารางหรือแบบกราฟิก

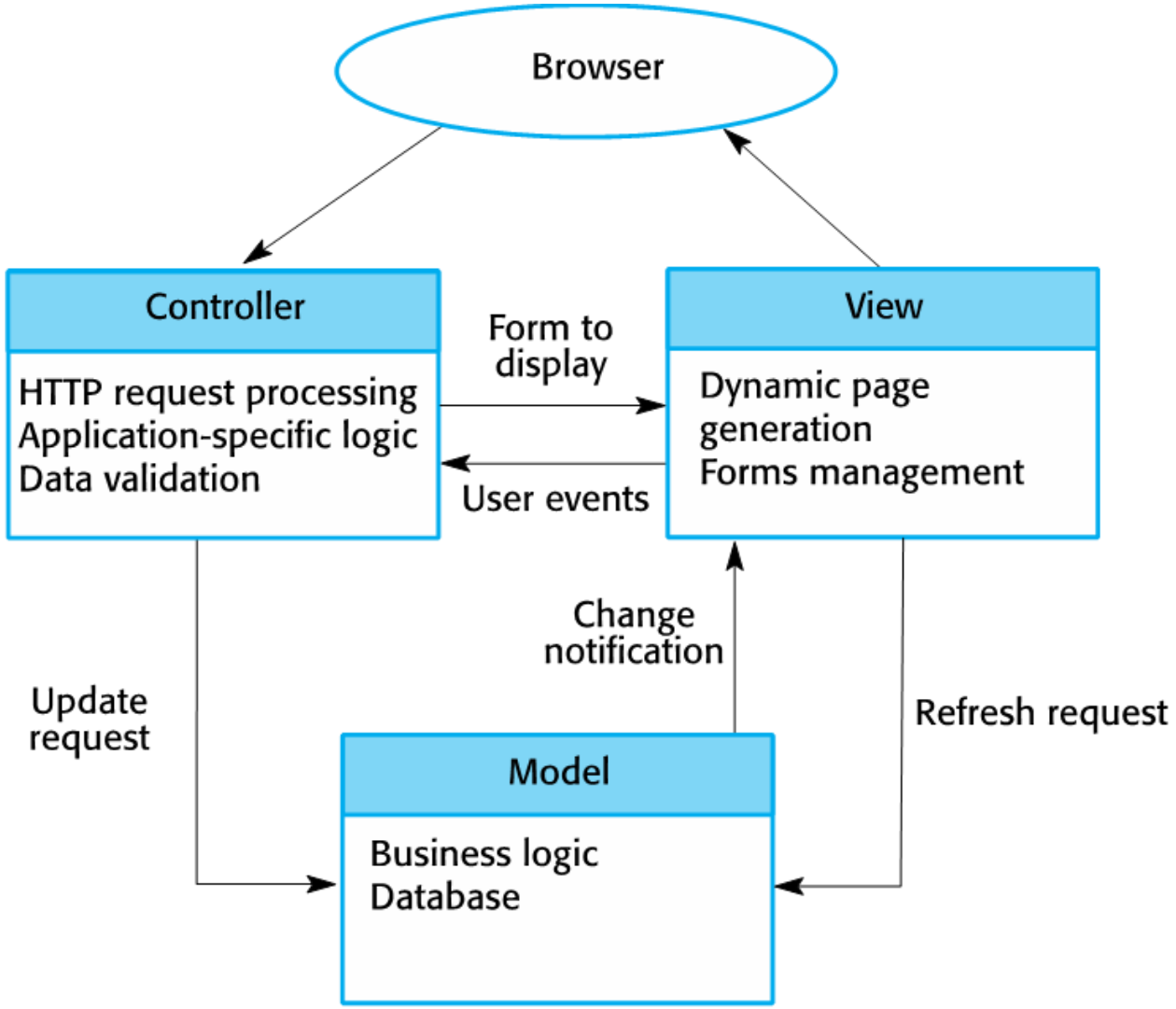
The Model-View-Controller (MVC) pattern

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. See Figure 6.3.
Example	Figure 6.4 shows the architecture of 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.

The organization of the Model-View-Controller



Web application architecture using the MVC pattern



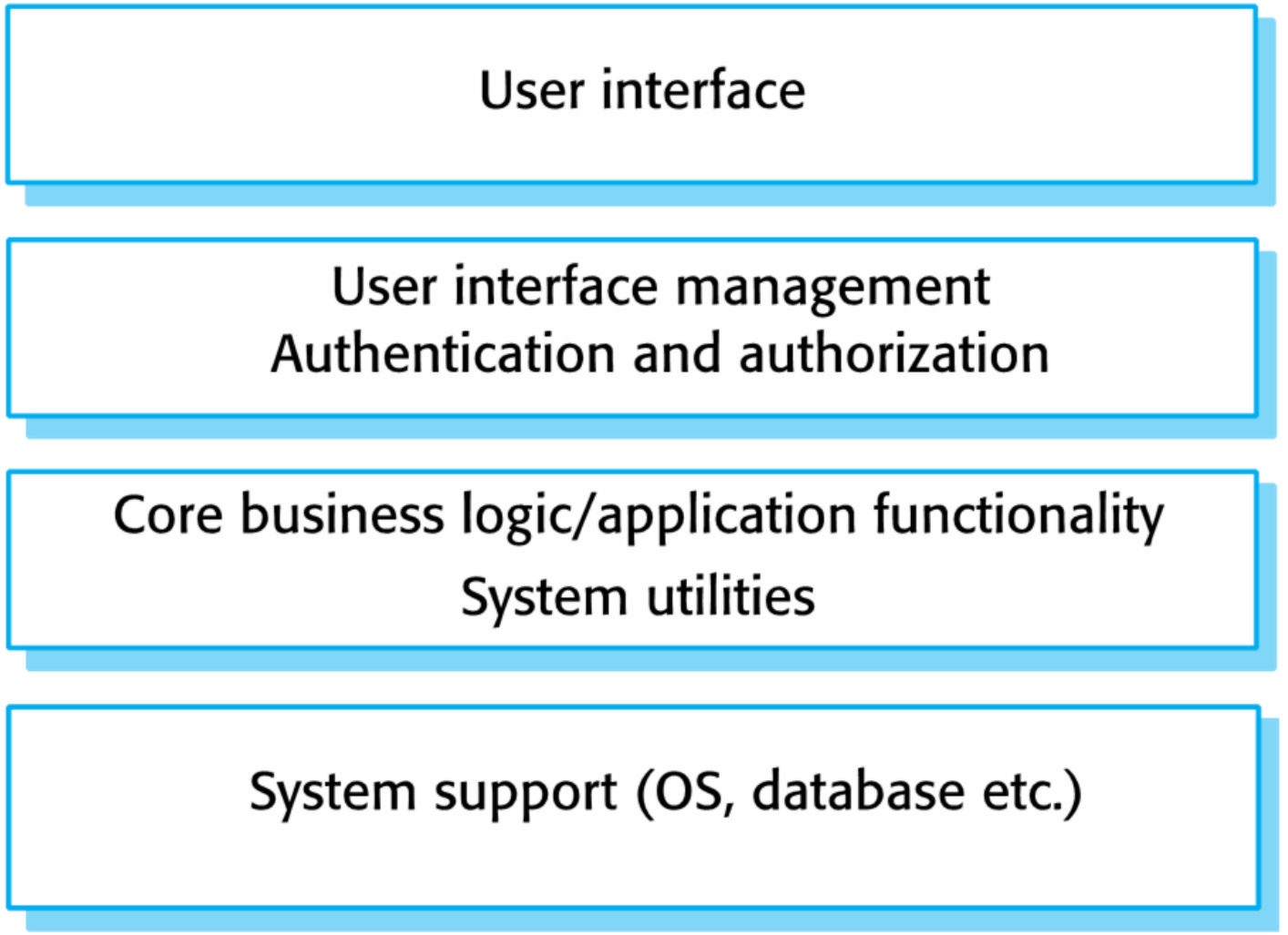
Layered architecture

- เหมาะที่จะใช้ในการจำลองการเชื่อมต่อของระบบย่อย
- จัดระบบให้เป็นชุดของ layers หรือ abstract machine
  - แต่ละ lauer จะมีชุดของ services อยู่ภายใน
- สนับสนุนการพัฒนาแบบ incremental ของระบบย่อยใน layers ต่างๆ
  - เมื่อเลเยอร์อินเทอร์เฟซเปลี่ยนไปจะมีผลต่อเลเยอร์ที่อยู่ติดกันเท่านั้น
- ส่วนใหญ่ มักจะใช้วิธีการนี้ในการวางระบบ

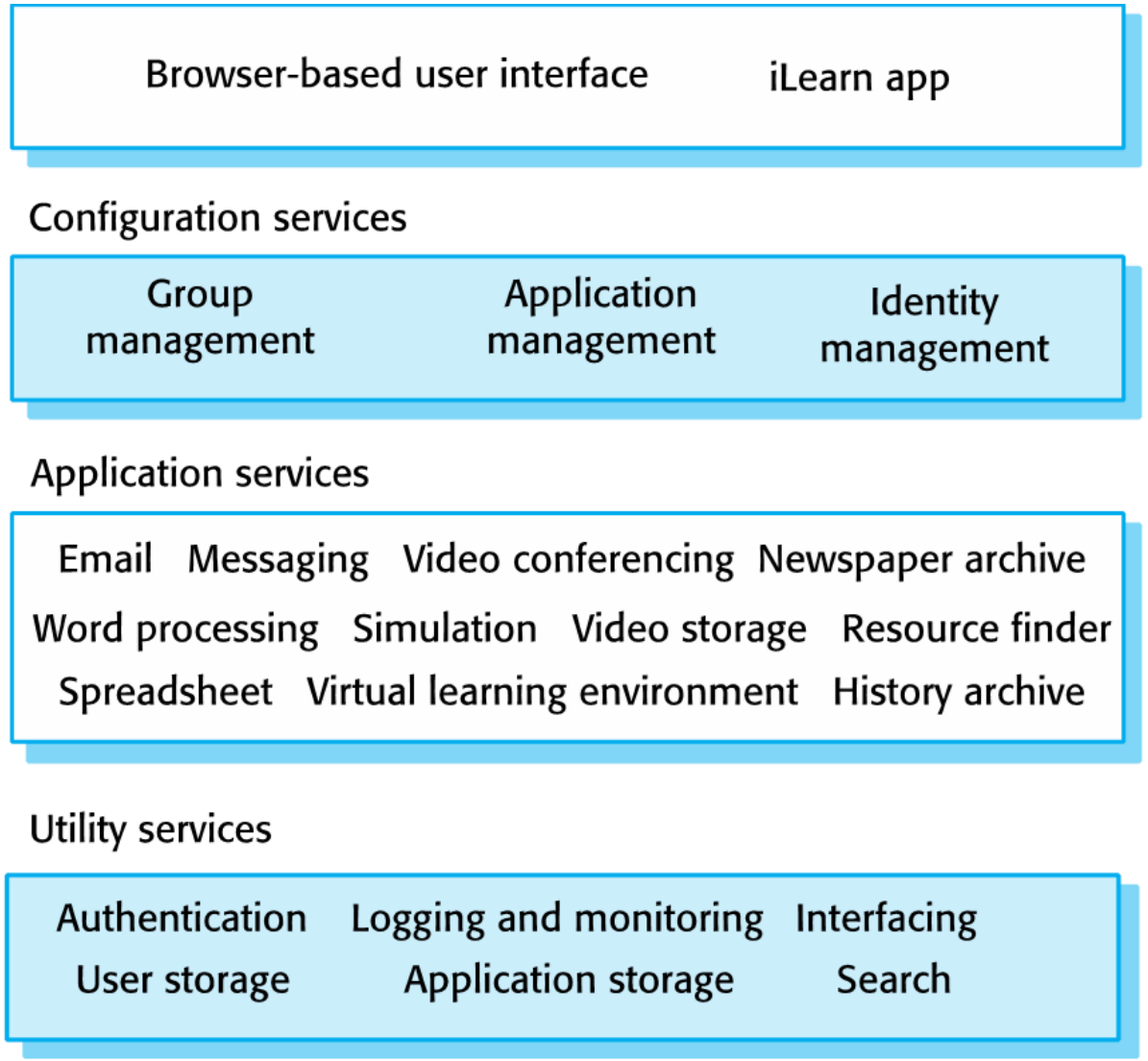
The Layered architecture pattern

Name	Layered architecture
Description	Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system. See Figure 6.6.
Example	A layered model of a system for sharing copyright documents held in different libraries, as shown in Figure 6.7.
When used	Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsibility for a layer of functionality; when there is a requirement for multi-level security.
Advantages	Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

A generic layered architecture



The architecture of the iLearn system



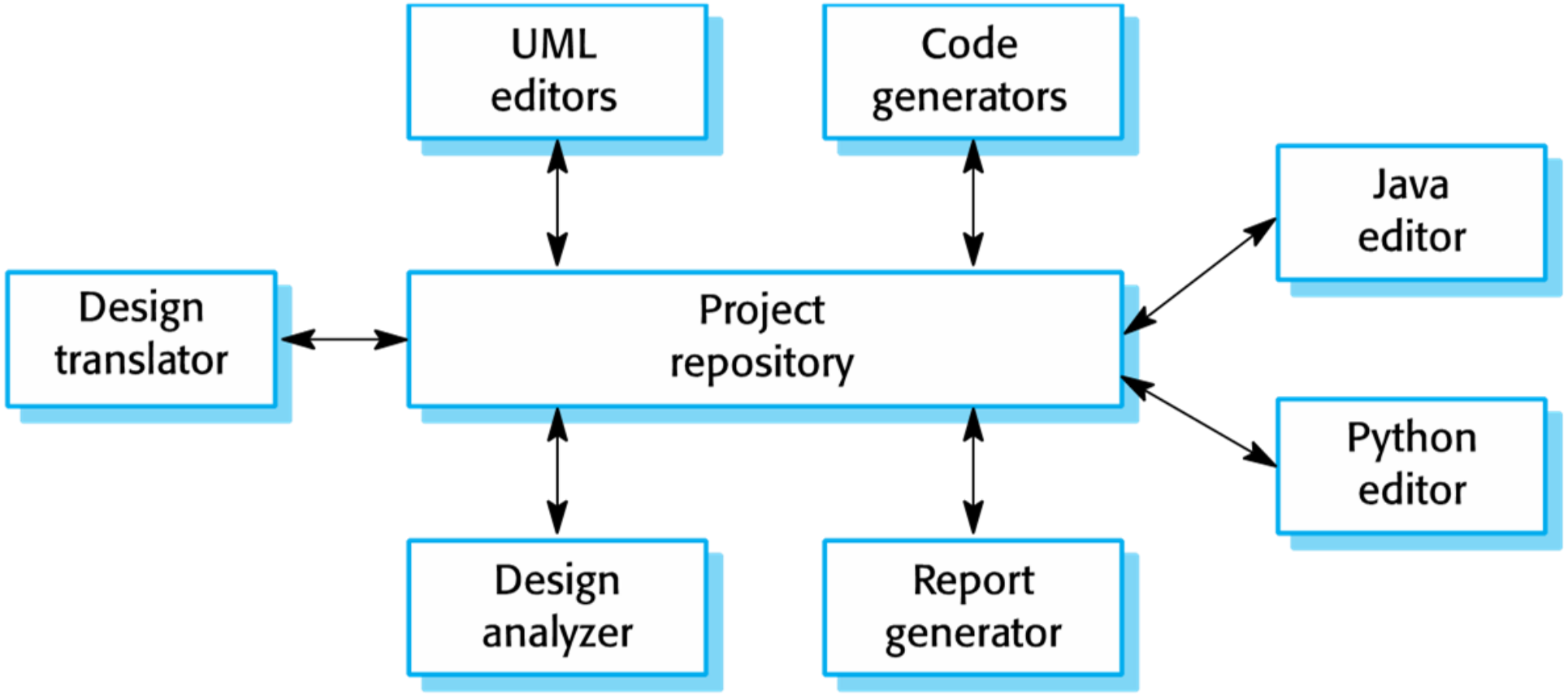
Repository architecture

- ระบบย่อยต้องแลกเปลี่ยนข้อมูล ซึ่งอาจทำได้ในสองวิธี:
  - ข้อมูลที่ใช้ร่วมกันถูกเก็บไว้ในฐานข้อมูลส่วนกลางหรือพื้นที่เก็บข้อมูลและอาจเข้าถึงโดยระบบย่อยทั้งหมด
  - แต่ละระบบย่อยจะเก็บรักษาฐานข้อมูลของตนเองและส่งผ่านข้อมูลไปยังระบบย่อยอื่น ๆ อย่างชัดเจน
- การออกแบบระบบที่มีการแชร์ข้อมูลจำนวนมากนั้น repository model จะถูกใช้มากที่สุด
  - เป็นกลไกการแบ่งปันข้อมูลที่มีประสิทธิภาพ

The Repository pattern

Name	Repository
Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
Example	Figure 6.9 is an example of an IDE where the components use a repository of system design information. Each software tool generates information which is then available for use by other tools.
When used	You should use this pattern when you have a system in which large volumes of information are generated that has to be stored for a long time. You may also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	Components can be independent—they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.
Disadvantages	The repository is a single point of failure so problems in the repository affect the whole system. May be inefficiencies in organizing all communication through the repository. Distributing the repository across several computers may be difficult.

A repository architecture for an IDE



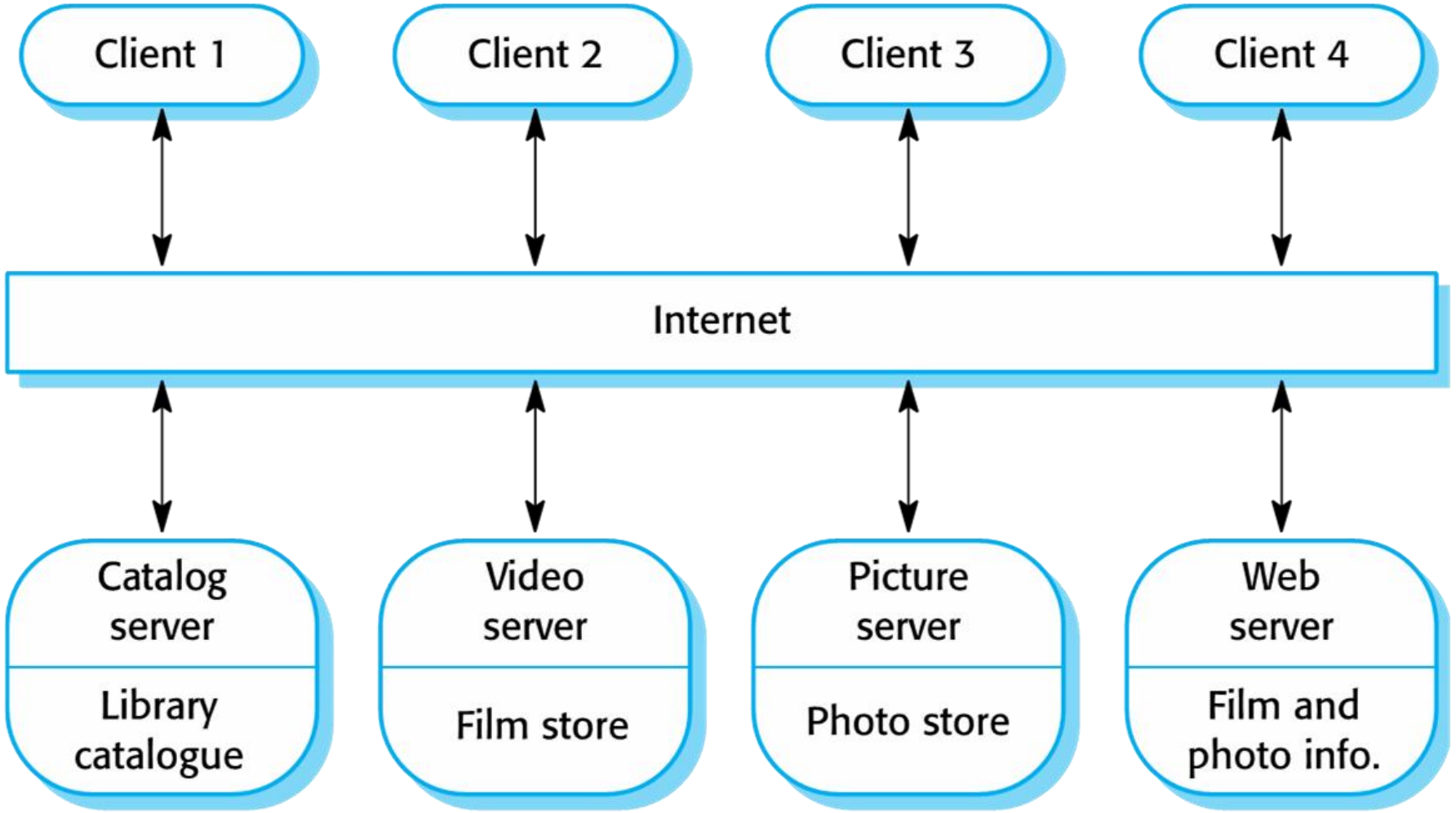
Client-server architecture

- แบบจำลองระบบแบบกระจาย
  - จะแสดงวิธีการกระจายข้อมูลและการประมวลผลในส่วนประกอบต่าง ๆ
  - จริง ๆ แล้วสามารถใช้งานได้บนคอมพิวเตอร์เครื่องเดียวก็ได
- ตัวอย่าง stand-alone servers เช่น การพิมพ์ (printing) การจัดการข้อมูล (data management) ฯลฯ
- Client ทั้งหมดในระบบ จะต้องสามารถเข้าถึง services
  - อาจจะประกอบด้วย network ที่ช่วยให้ Client เข้าถึงได้จากระยะไกล
-

The Client–server pattern

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	Figure 6.11 is 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.

A client–server architecture for a film library



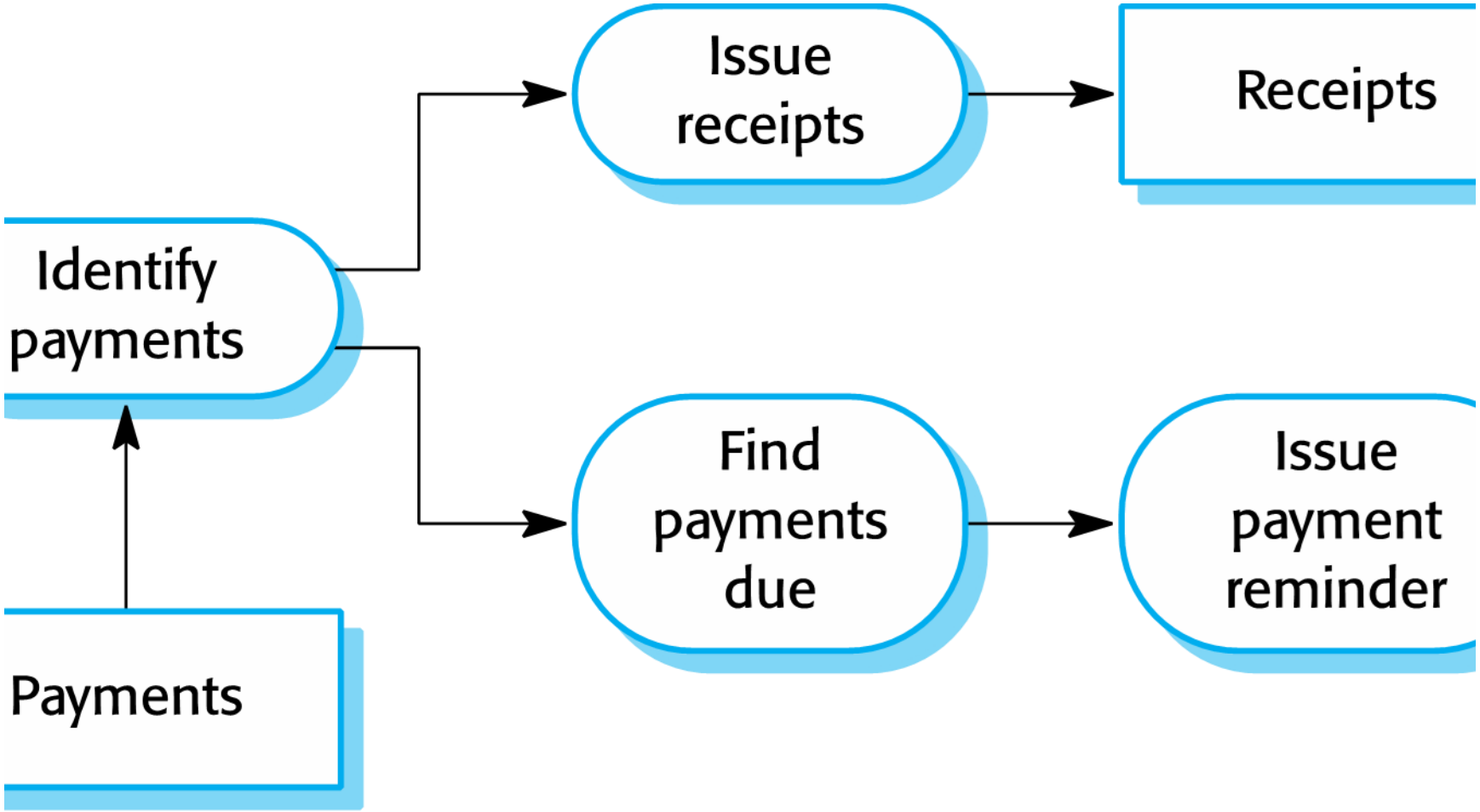
Pipe and filter architecture

- ส่วนที่ทำงาน (functional) ของระบบจะประมวลผล input เพื่อสร้าง output
- อาจเรียกว่า pipe และ filter (เช่นเดียวกับใน UNIX shell)
- รูปแบบนี้นิยมใช้กันเยอะมาก
  - การป้อนคำสั่งเป็นลำดับ จะถูกมองเป็นคำสั่งแบบ batch และถูกใช้อย่างมากในระบบประมวลผลข้อมูล
- ไม่เหมาะสำหรับระบบโต้ตอบ
  - สั่งงานแล้วรอผลอย่างเดียว อาจจะไม่มีการรายงานใด ๆ จนกว่าจะเสร็จ

The pipe and filter pattern

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	Figure 6.13 is an example of 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.

An example of the pipe and filter architecture used in a payments system



Application architectures

Application architectures

- ระบบ Application ได้รับการออกแบบเพื่อตอบสนองความต้องการขององค์กร
- โดยส่วนใหญ่ระบบธุรกิจจะมีกิจกรรมที่เหมือน ๆ กัน
  - ระบบ application มักมีสถาปัตยกรรมที่สะท้อนถึงความต้องการของ
- สถาปัตยกรรมแอ็พพลิเคชันทั่วไป (generic application architecture) เป็นสถาปัตยกรรมที่สามารถ config ระบบให้เข้ากับ requirement ได้โดยง่าย

Use of application architectures

- เป็นจุดเริ่มต้นของการออกแบบสถาปัตยกรรม
- เป็นรายการตรวจสอบการออกแบบ
- เป็นวิธีการจัดงานของทีมพัฒนา
- เป็นวิธีการประเมินส่วนประกอบเพื่อนำกลับมาใช้ใหม่
- เป็นคำศัพท์สำหรับการพูดถึงประเภทของแอพพลิเคชั่น

Examples of application types

- แอปพลิเคชันประมวลผลข้อมูล (Data processing applications)
  - ขับเคลื่อนด้วยข้อมูลซึ่งประมวลผลข้อมูลในแบทช์
  - ไม่มีการแทรกแซงหรือโต้ตอบใดๆ จากผู้ใช้อย่างชัดเจนในระหว่างการประมวลผล
- แอปพลิเคชันประมวลผลธุรกรรม (Transaction processing applications)
  - แอ็พพลิเคชันที่เน้นข้อมูล
  - เป็นข้อมูลซึ่งประมวลผลคำขอของผู้ใช้และอัพเดตข้อมูลในฐานข้อมูลระบบ
- ระบบประมวลผลเหตุการณ์ (Event processing systems)
  - การทำงานของระบบจะขึ้นอยู่กับการตีความเหตุการณ์จากสภาพแวดล้อมของระบบ
- ระบบประมวลผลภาษา (Language processing systems)
  - อินพุตจากผู้ใช้ระบุไว้ในภาษาทางการ ซึ่งประมวลผลและตีความโดยระบบ

Application type examples

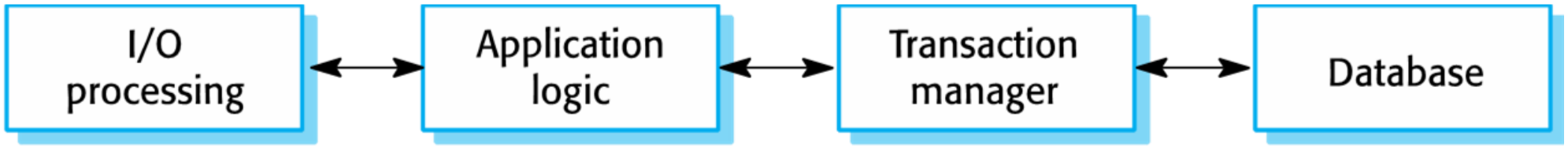
สถาปัตยกรรมแอ็พพลิเคชันทั่วไปที่ใช้กันอย่างแพร่หลายสองอย่างคือ

- ระบบประมวลผลธุรกรรม (Transaction processing systems)
  - ระบบอีคอมเมิร์ซ
  - ระบบการจอง
- ระบบประมวลผลภาษา (Language processing systems)
  - คอมไพเลอร์
  - ระบบประมวลผลคำสั่ง (Command interpreters)

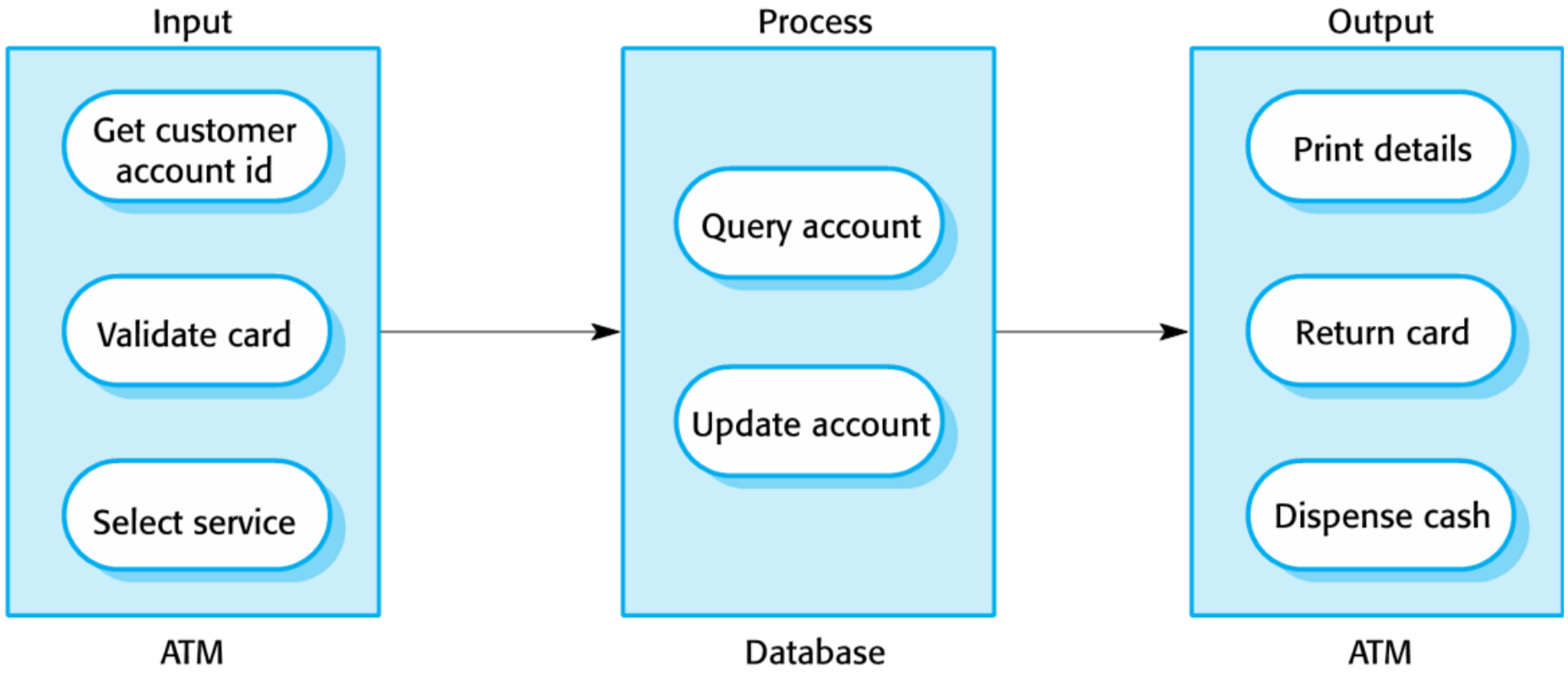
Transaction processing systems

- ประมวลผลคำขอของผู้ใช้ฐานข้อมูล เพื่อขอข้อมูลหรือขออัพเดตฐานข้อมูล
- จากมุมมองของผู้ใช้ธุรกรรมคือ:
  - ทำงานตามลำดับการทำงานที่กำหนดไว้ เพื่อให้บรรลุเป้าหมาย
  - ตัวอย่างเช่น - หาเวลาเที่ยวบินจากลอนดอนไปปารีส
- ผู้ใช้ร้องขอแบบ asynchronous สำหรับบริการที่ประมวลผลโดย transaction manager

The structure of transaction processing applications



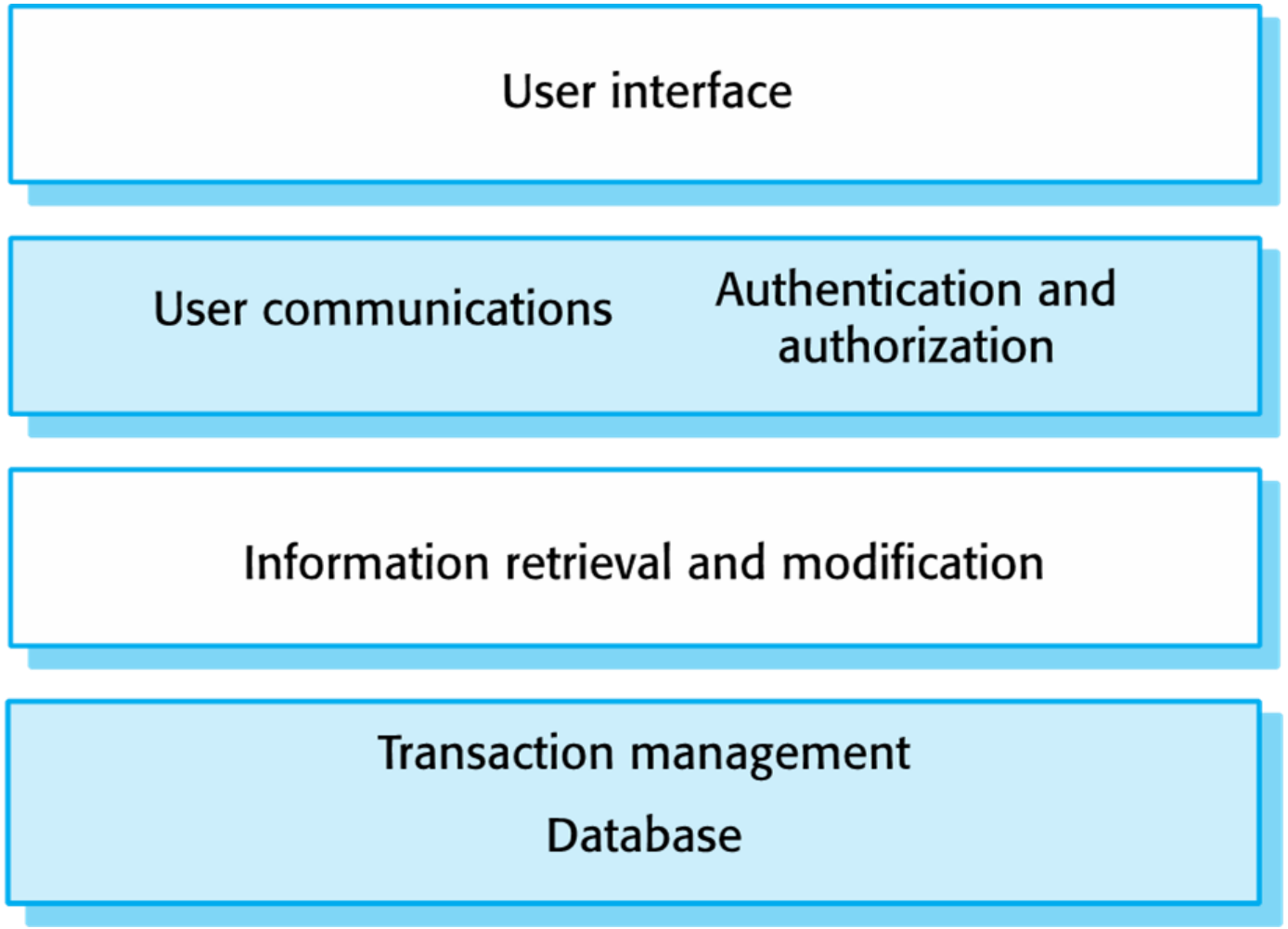
The software architecture of an ATM system



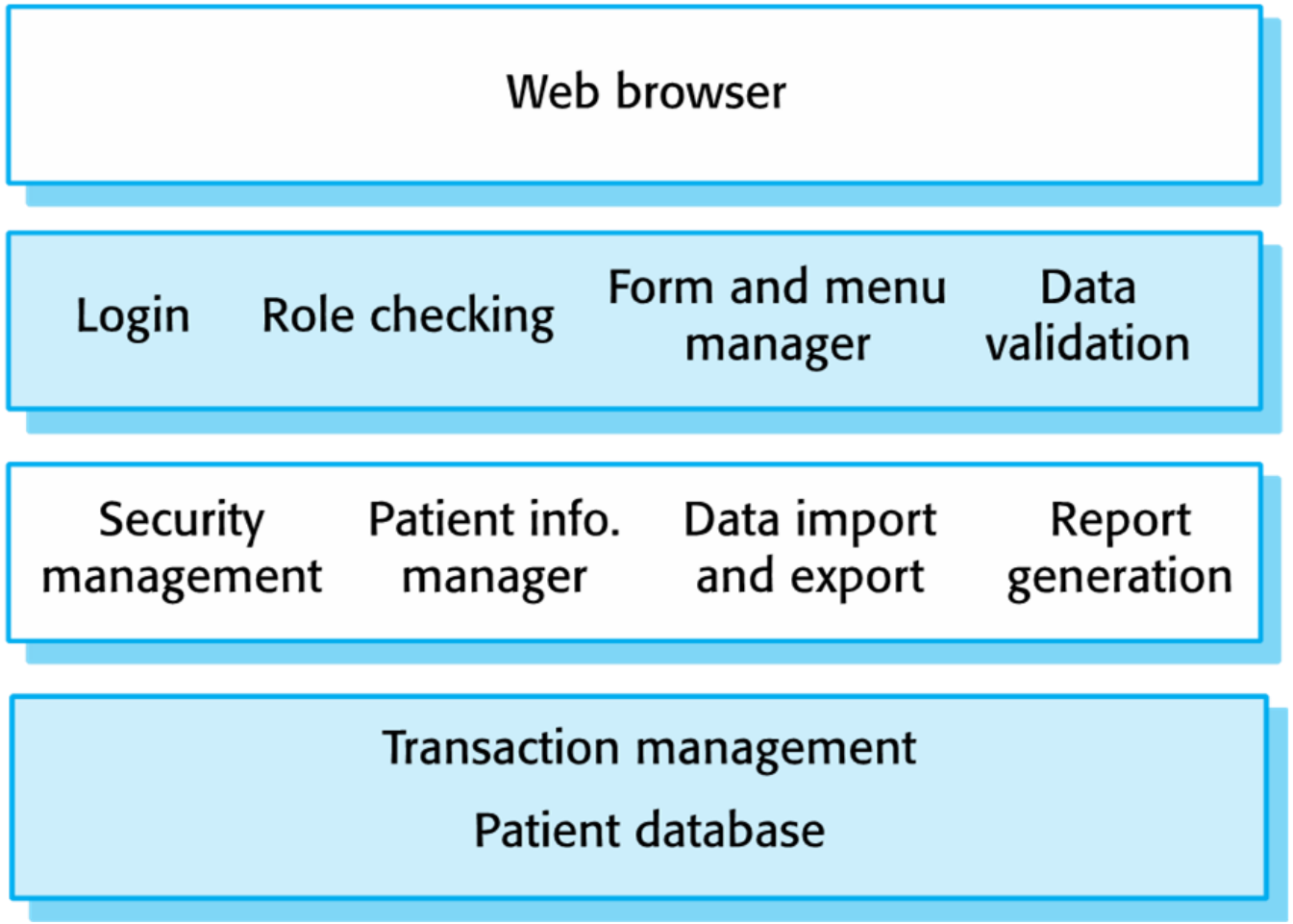
Information systems architecture

- ระบบสารสนเทศมีสถาปัตยกรรมทั่วไป (generic architecture) ที่สามารถจัดเป็นสถาปัตยกรรมแบบชั้น ๆ ได้
- การทำธุรกรรมตามการปฏิสัมพันธ์กับระบบเหล่านี้มักเกี่ยวข้องกับการทำธุรกรรมฐานข้อมูล
- เลเยอร์ต่าง ๆ รวมถึง
  - ส่วนติดต่อผู้ใช้
  - การสื่อสารกับผู้ใช้
  - การดึงข้อมูล
  - ฐานข้อมูลระบบ

Layered information system architecture



The architecture of the Mentcare system



Server implementation

- เซิร์ฟเวอร์มักถูกสร้างเป็นสถาปัตยกรรมของไคลเอ็นต์หลายชั้น (multi-tier)
  - เว็บเซิร์ฟเวอร์มีหน้าที่รับผิดชอบในการสื่อสารกับผู้ใช้ทุกคน โดยมี user interface สำหรับผู้ใช้ที่ทำงานบนเว็บเบราเซอร์
  - application server มีหน้าที่รับผิดชอบในการดำเนินลอจิกเฉพาะแอ็พพลิเคชัน รวมถึงการจัดเก็บและเรียกข้อมูล
  - database server จะย้ายข้อมูลเข้า-ออกจากฐานข้อมูลและจัดการธุรกรรม

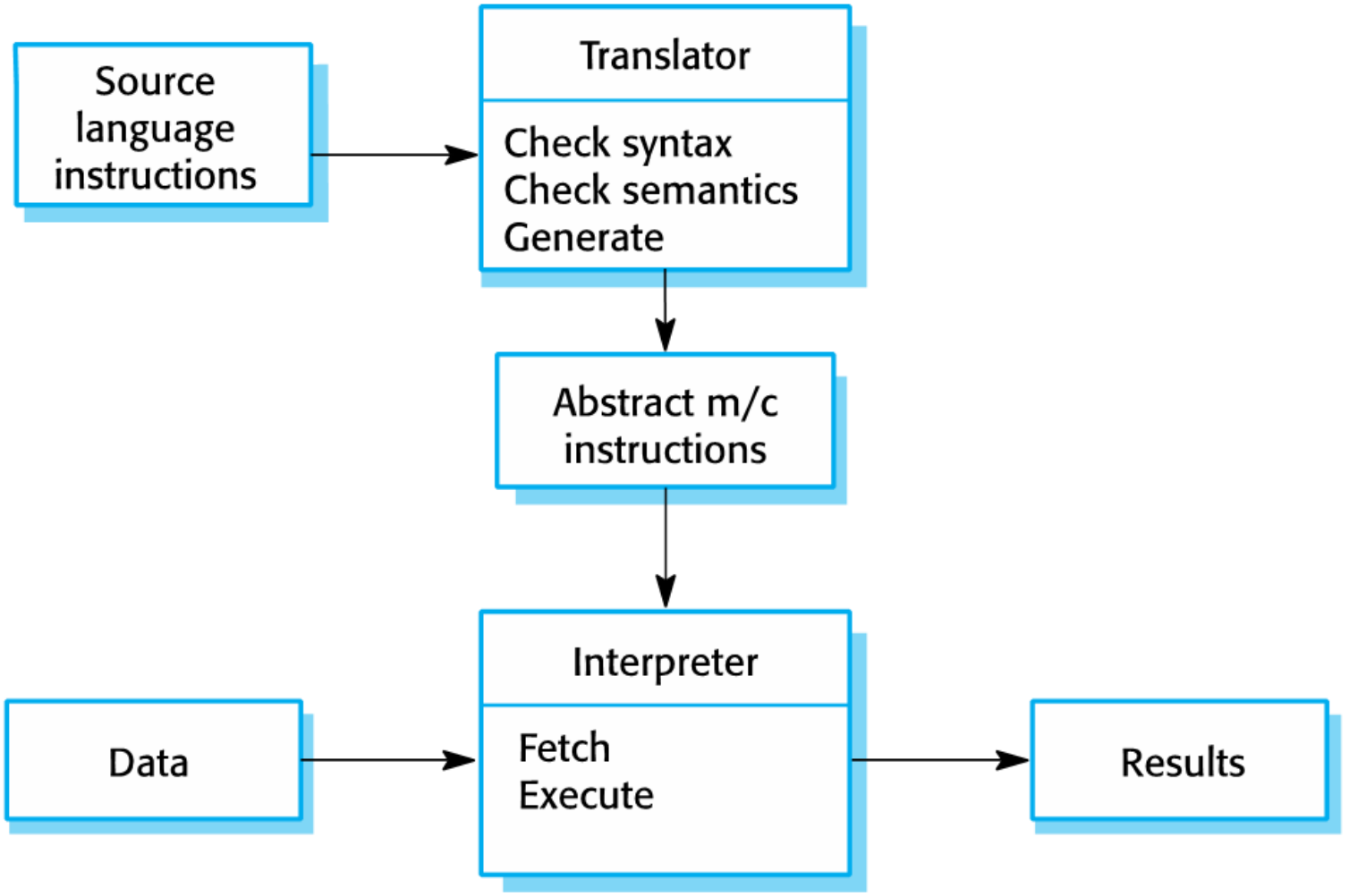
Language processing systems

- ระบบนี้จะยอมรับภาษาธรรมชาติหรือภาษาเทียมเป็นข้อมูลเข้าและสร้างภาษาอื่น ๆ ขึ้นมาแทนที่
- อาจมี interpreter เพื่อดำเนินการตามคำแนะนำในภาษาที่กำลังดำเนินการอยู่
- นิยมใช้เมื่อต้องการวิธีที่ง่ายที่สุด สำหรับแก้ปัญหาในการอธิบายอัลกอริทึมหรืออธิบายข้อมูลระบบ
  - Meta-case tools process tool descriptions, method rules, เป็นต้น รวมทั้ง generate tools

Web-based information systems

- ระบบสารสนเทศและการจัดการทรัพยากรมักเป็นระบบบนเว็บ
  - ส่วนอินเทอร์เฟซผู้ใช้ จะ implement เป็นระบบที่ใช้เว็บเบราเซอร์
- ตัวอย่างเช่นระบบอีคอมเมิร์ซ
  - เป็นระบบการจัดการทรัพยากรทางอินเทอร์เน็ตที่รับการสั่งซื้อสินค้าหรือบริการอิเล็กทรอนิกส์แล้วจัดเตรียมการส่งมอบสินค้าหรือบริการเหล่านี้ให้กับลูกค้า
- ในระบบอีคอมเมิร์ซ
  - application-specific layer มีฟังก์ชันเพิ่มเติมที่สนับสนุน 'รถเข็นช็อปปิ้ง'
  - ผู้ใช้สามารถวางรายการสินค้าจำนวนหนึ่งไว้ในธุรกรรมแยกต่างหาก
  - จากนั้นจ่ายเงินทั้งหมดในการทำรายการเดียว

The architecture of a language processing system



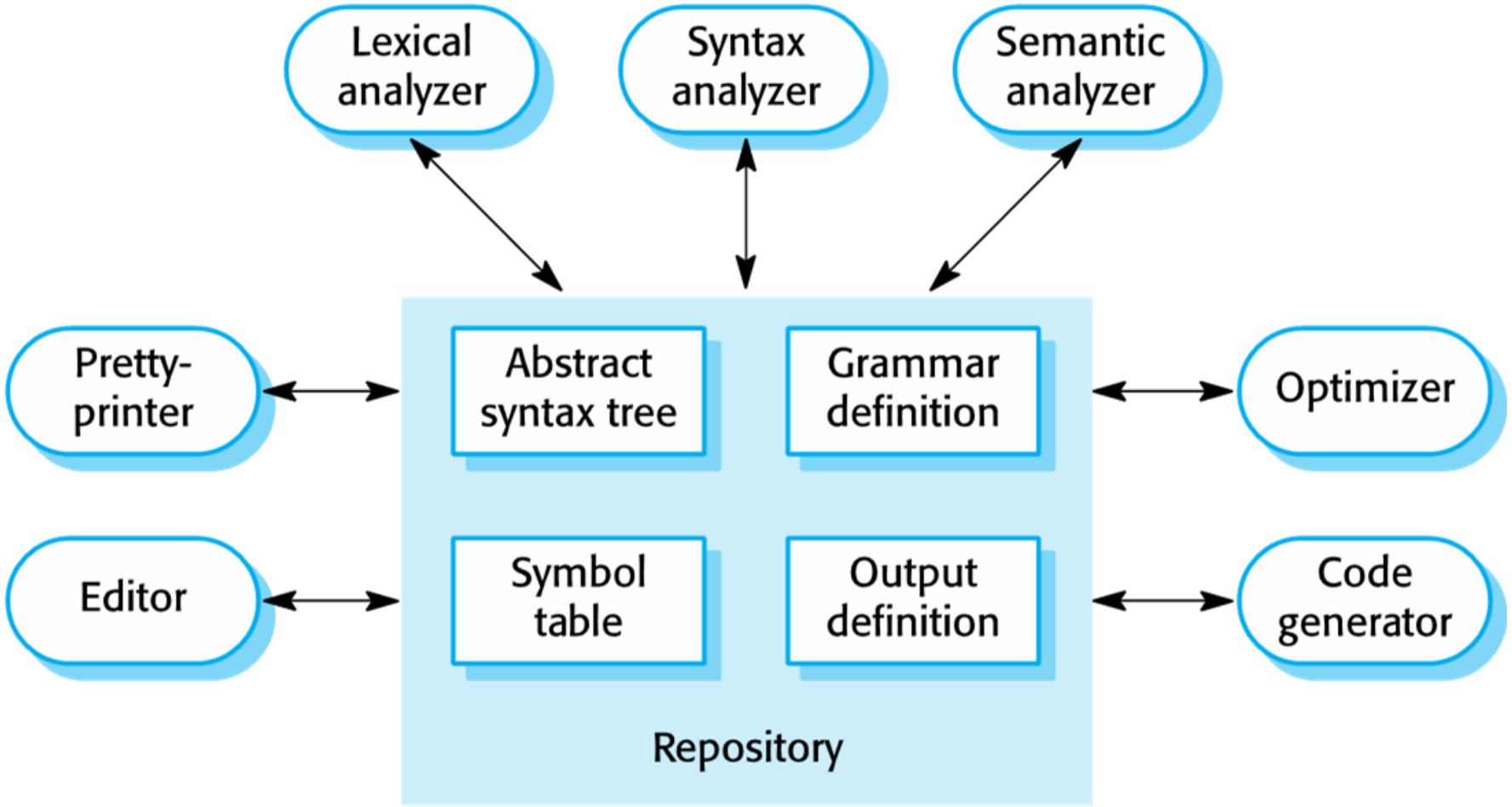
Compiler components

- A lexical analyzer
  - รับสัญลักษณ์ของภาษาในการป้อนข้อมูลและแปลงเป็นรูปแบบภายใน
- A symbol table
  - เก็บข้อมูลเกี่ยวกับชื่อของเอนทิตี (ตัวแปร ชื่อคลาส ชื่อวัตถุ ฯลฯ) ที่ใช้ในข้อความที่กำลังแปล
- A syntax analyzer
  - วิเคราะห์ไวยากรณ์ ซึ่งจะตรวจสอบไวยากรณ์ของภาษาที่แปล
- A syntax tree
  - โครงสร้างไวยากรณ์ ซึ่งเป็นโครงสร้างภายในที่แสดงถึงโปรแกรมที่คอมไพล์

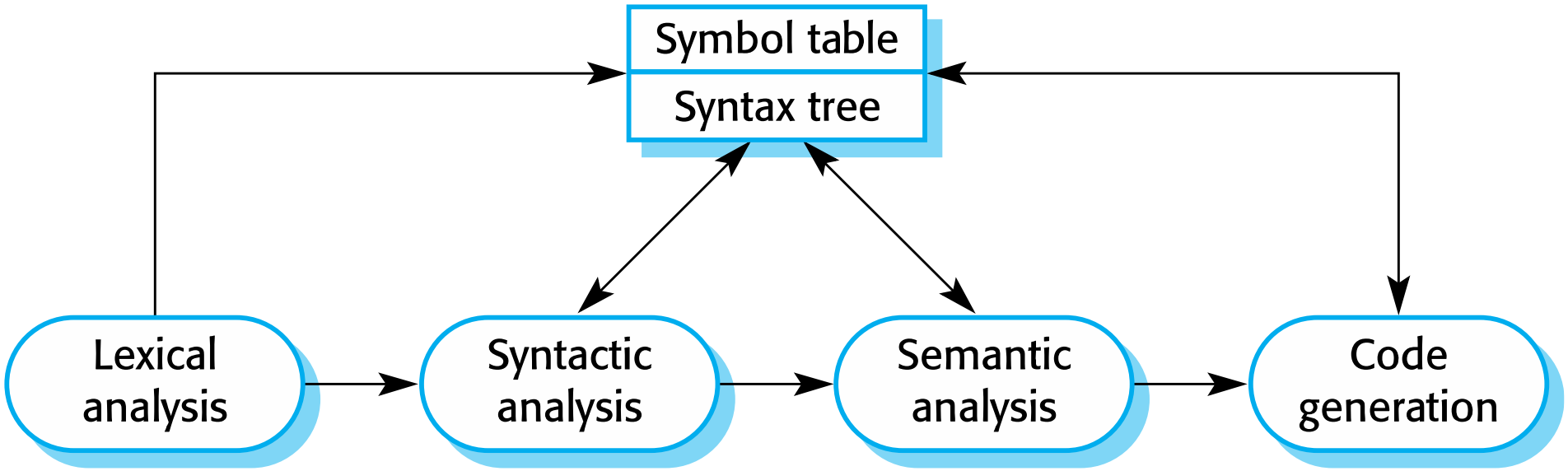
Compiler components

- A semantic analyzer
  - ใช้ข้อมูลจากโครงสร้างไวยากรณ์และตารางสัญลักษณ์เพื่อตรวจสอบความถูกต้องของความหมายของข้อความภาษาอินพุต
- A code generator
  - ดำเนินการตาม syntax tree และสร้างรหัสภาษาเครื่อง

A repository architecture for a language processing system



A pipe and filter compiler architecture



Key points

- สถาปัตยกรรมซอฟต์แวร์เป็นคำอธิบายเกี่ยวกับการจัดองค์ประกอบระบบซอฟต์แวร์
- การตัดสินใจในการออกแบบสถาปัตยกรรมประกอบด้วย
  - การตัดสินใจเกี่ยวกับประเภทของโปรแกรม
  - การกระจายขององค์ประกอบต่างๆ ในระบบ
  - รูปแบบสถาปัตยกรรมที่จะนำมาใช้
- สถาปัตยกรรมอาจได้รับการจัดทำเป็นเอกสารจากมุมมองต่าง ๆ
  - มุมมองแนวความคิด
  - มุมมองเชิงเหตุผล
  - มุมมองกระบวนการ
  - มุมมองการพัฒนา

Key points

- Architectural patterns เป็นวิธีการนำความรู้เกี่ยวกับสถาปัตยกรรมระบบทั่วไปมาใช้งาน
  - มีคำอธิบายสถาปัตยกรรม อธิบายว่าเมื่อใดควรใช้และอธิบายข้อดีและข้อเสียของมัน
- โมเดลของสถาปัตยกรรมระบบแอ็พพลิเคชันช่วยให้เรา
  - สามารถทำความเข้าใจและเปรียบเทียบแอพพลิเคชัน
  - ตรวจสอบการออกแบบระบบแอ็พพลิเคชัน
  - ประเมินส่วนประกอบที่มีขนาดใหญ่เพื่อนำกลับมาใช้ใหม่
- ระบบประมวลผลธุรกรรมเป็นระบบโต้ตอบที่อนุญาตให้ผู้ใช้จำนวนมากสามารถเข้าถึงและแก้ไขข้อมูลในฐานข้อมูลจากระยะไกลได้

Key points

- ระบบประมวลผลภาษาใช้ในการแปลข้อความจากภาษาหนึ่งไปยังอีกภาษาหนึ่ง
  - ดำเนินการตามคำแนะนำที่ระบุในภาษาสำหรับการป้อนข้อมูล
  - ประกอบด้วยเครื่องแปลภาษาต้นทางไปยังภาษปลายทางที่ต้องการสร้าง

คำถาม???