

Công nghệ phần mềm

Thiết kế phần mềm

Nội dung chính

- **Khái niệm thiết kế phần mềm**
- Các hoạt động thiết kế
- Thiết kế hướng đối tượng

Nội dung chính

- **Khái niệm thiết kế phần mềm**
 - Khái niệm
 - Nguyên lý
 - Chất lượng
 - Nội dung thiết kế và chất lượng
- Các hoạt động thiết kế
- Thiết kế hướng đối tượng

Khái niệm thiết kế PM

- Thiết kế là chuyển đặc tả yc thành mô tả thiết kế mà người lập trình có thể chuyển thành chương trình với 1 ngôn ngữ, vận hành được, đáp ứng được các yc đặt ra.
 - Là 1 quá trình sáng tạo:
 - Tìm giải pháp công nghệ (cách thức, phương án)
 - Biểu diễn cách thức, phương án
 - Xem xét lại, chi tiết hóa.
- ➡ Đủ chi tiết để người lập trình biết phải làm ntn để chuyển thành chương trình.

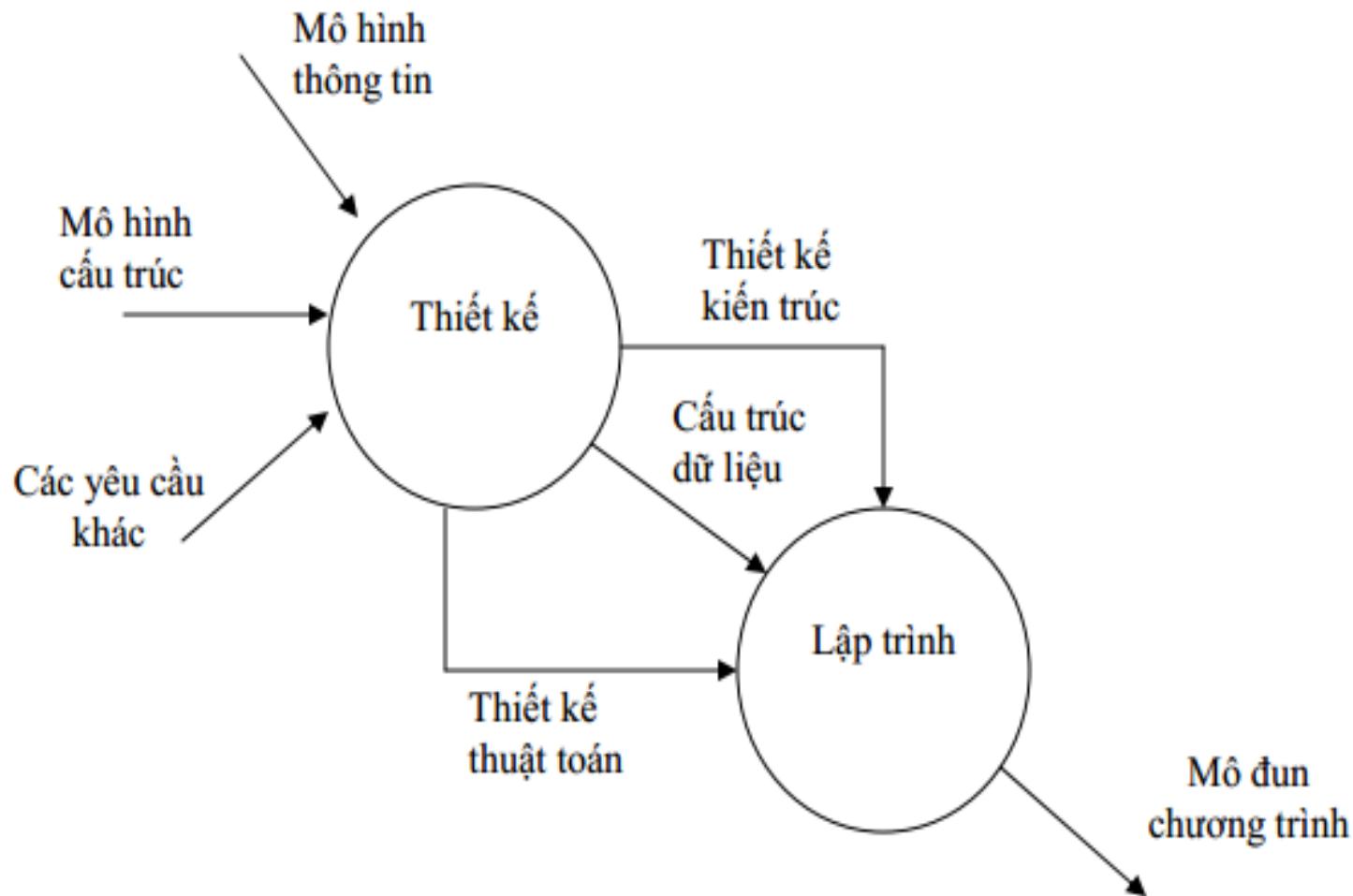
Vai trò thiết kế

- Tạo mô hình cài đặt của PM
- Là công cụ giao tiếp giữa những người tham gia phát triển, cơ sở đảm bảo chất lượng hệ thống.
 - Dễ đọc, dễ hiểu, dễ sửa đổi hơn mã chương trình
 - Có nhiều mức chi tiết; cung cấp cái nhìn tổng thể
 - Làm cơ sở để trao đổi, cải tiến.
- Cung cấp đầy đủ thông tin cho việc bảo trì sau này:
 - Giảm công sức mã hóa khi sửa đổi
 - Tiện mở rộng, bảo trì phần mềm.

Vai trò thiết kế

- Mô hình thiết kế cung cấp các thông tin về kiến trúc (architectures), giao tiếp (interfaces), thành phần (components) và dữ liệu (data).

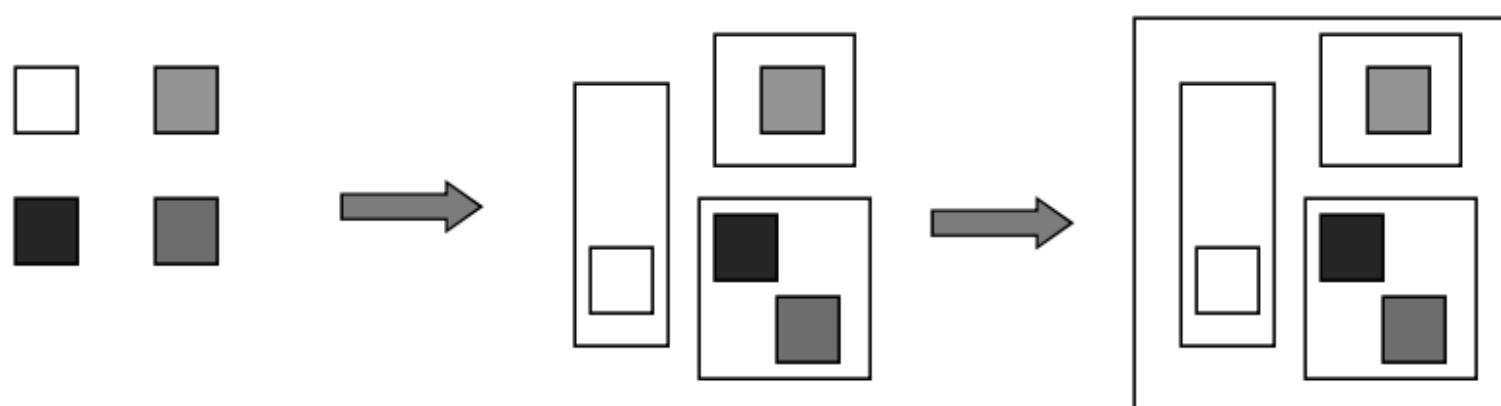
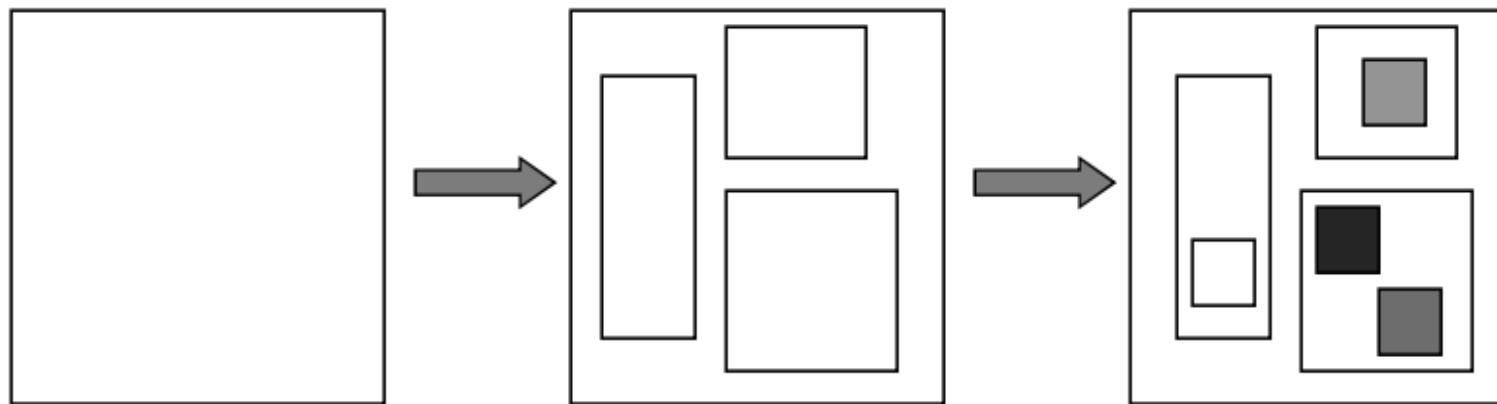
Vai trò thiết kế



Cấu trúc thiết kế

- PM là tập các module tương tác lẫn nhau
- **Module hóa** là chia khóa cho PM tốt
- Mục tiêu thiết kế là xác định
 - Các module chức năng
 - Cách thức cài đặt module
 - Tương tác giữa các module

2 hướng thiết kế



Nguyên lý thiết kế

1. Không bó buộc vào 1 cách nhìn hạn chế nào
 - Nó cần được lựa chọn từ các giải pháp có thể
2. Cho phép lặp ngược lại mô hình phân tích
 - Các module và các yc không nhất thiết phải tương ứng 1-1
 - Phải kiểm tra được sự thỏa mãn các yc
3. Không nên tạo lại các thiết kế (giải pháp) đã có, mà cần tái sử dụng tối đa chúng

Nguyên lý thiết kế

4. Mô hình thiết kế (giải pháp) nên tiến gần đến mô hình thế giới thực (bài toán)
5. Biểu diễn thiết kế phải **nhất quán** và có **tính tích hợp**
 - Thiết kế do nhiều người tiến hành song song
 - Phải thống nhất cách biểu diễn, thống nhất giao diện
6. Thiết kế cần có cấu trúc để dễ hiểu, dễ thay đổi
 - Phải được module hóa, phân cấp

Nguyên lý thiết kế

7. Thiết kế không phải là mã hóa

- Thiết kế luôn có mức trừu tượng hơn mã hóa, đảm bảo dễ hiểu, dễ thay đổi

8. Thiết kế cần được đánh giá chất lượng ngay khi được tạo ra

- Tính kết dính, tính kết nối, hiệu quả thuật toán

9. Thiết kế cần được thẩm định để tránh các lỗi mang tính hệ thống

- Thiếu chức năng, chức năng không rõ, mâu thuẫn...

Nội dung và chất lượng thiết kế

- **Nội dung thiết kế:**

- Thiết kế kiến trúc

- Phân rã hệ thống thành các hệ thống con, các module
 - Xác định giao diện tương tác giữa các module

- Thiết kế cấu trúc DL

- Xây dựng mô hình biểu diễn thông tin

- Thiết kế thuật toán (thủ tục)

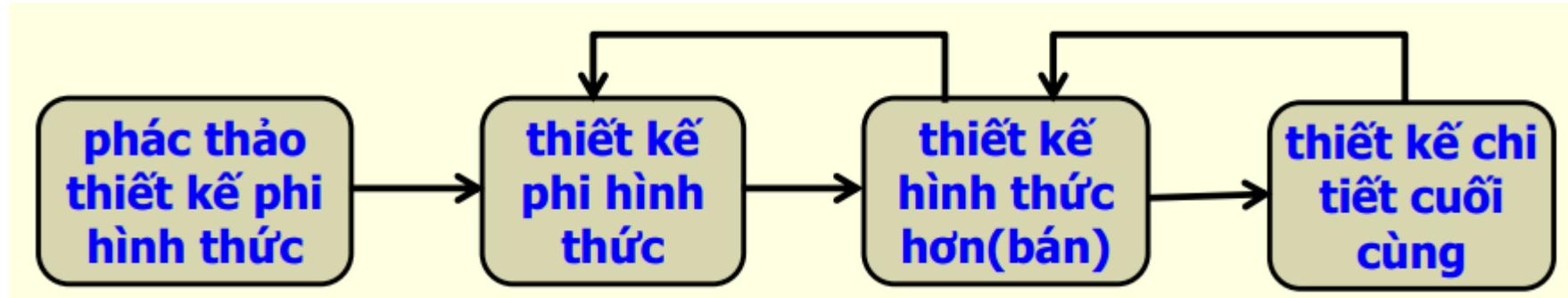
- Xác định các bước thực hiện xử lý

- Thiết kế giao diện người dùng

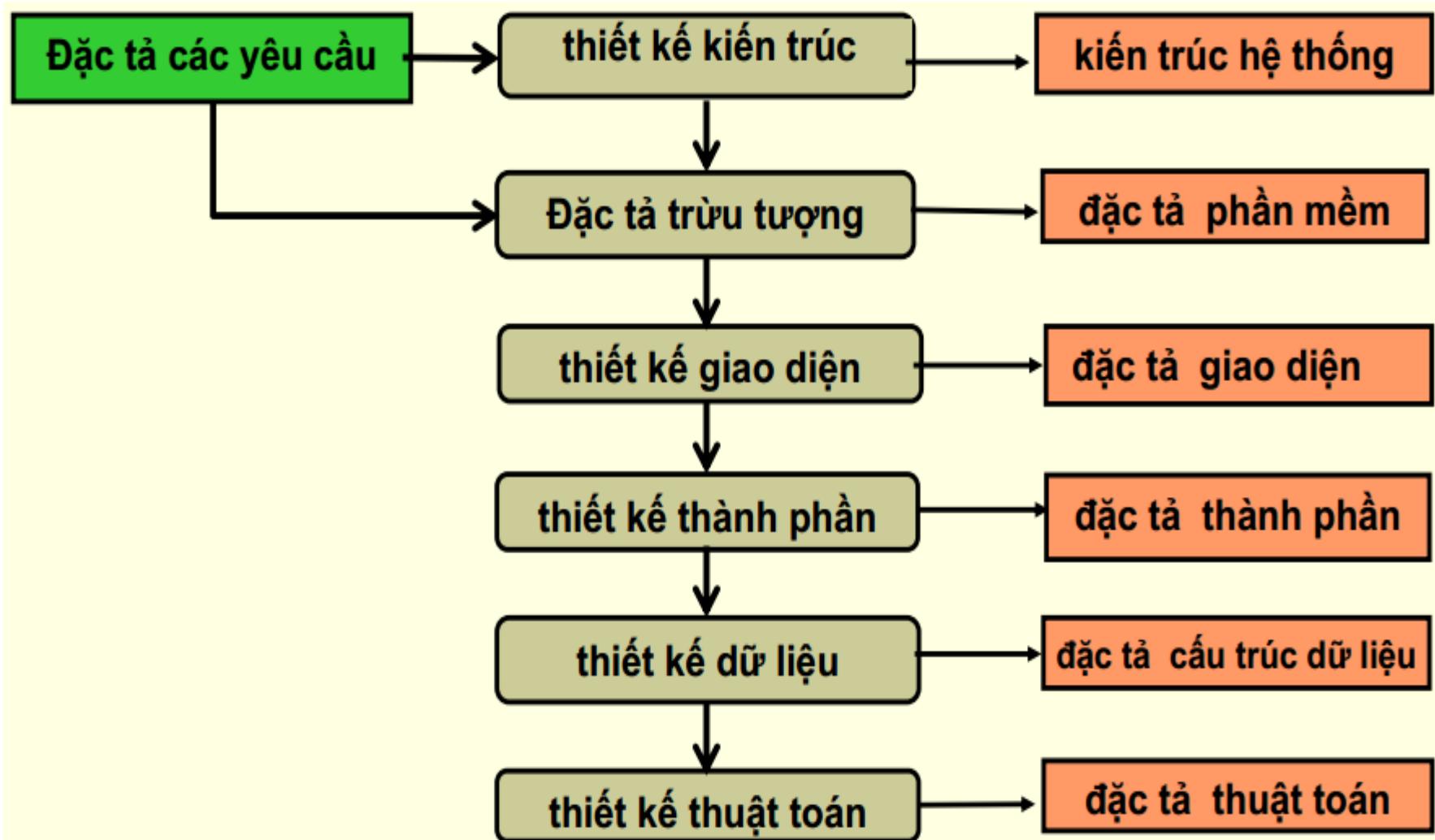
- Nên nhìn nhận giao diện là một bài toán độc lập

Mô hình tổng quát tiến trình thiết kế

- Tiến trình thiết kế là quá trình tăng cường hình thức hóa và luôn quay lại các thiết kế đúng đắn và ít hình thức hóa hơn trước đó để kiểm tra và hoàn chỉnh.



Tiến trình thiết kế



Thiết kế kiến trúc

- Sử dụng lược đồ cấu trúc (Structure chart), mô tả:
 - Cái nhìn tổng thể về hệ thống
 - Mối quan hệ giữa các module
 - Giao diện của các module
- Không cần chỉ ra:
 - Thứ tự thực hiện
 - Số lần thực hiện
 - Chi tiết thiết kế

Thiết kế CTDL

- Chọn cách biểu diễn các đối tượng thiết kế có ảnh hưởng mạnh mẽ đến chất lượng PM
- Các mức thiết kế:

Thiết kế cấu trúc logic	Thiết kế cấu trúc vật lý
Các quan hệ chuẩn Các khóa Các tham chiếu Các cấu trúc tham chiếu DL	Các file Các kiểu Kích cỡ

Thiết kế thuật toán

- Mô tả các bước hoạt động của module
- Phương pháp mô tả:
 - Giả mã (pseudo code)
 - Sơ đồ luồng (flow chart)
 - lược đồ (diagram) Nassi-Shneiderman
 - lược đồ hoạt động
 - JSP

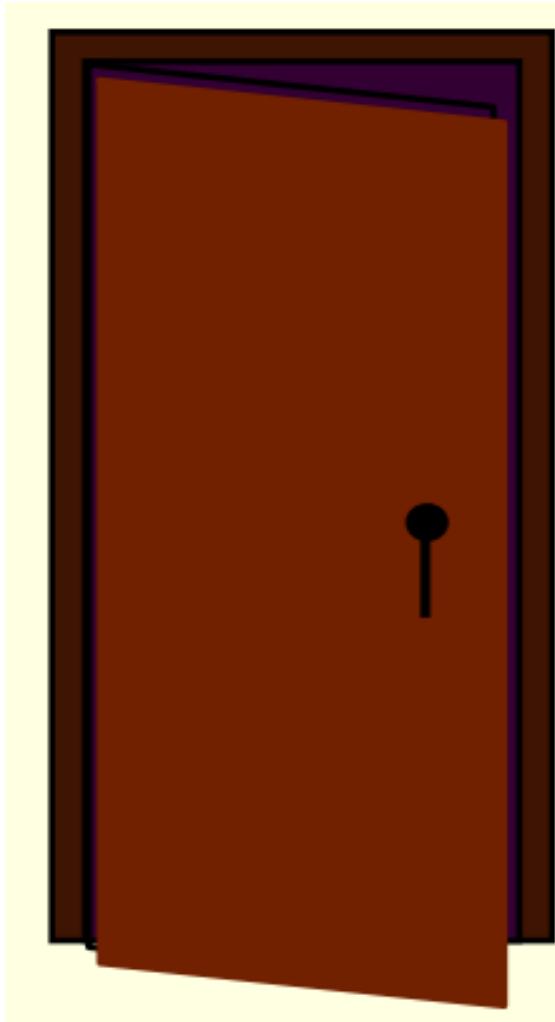
Các khái niệm thiết kế cơ sở

- Trừu tượng hóa: DL, thủ tục, điều khiển
- Làm mịn: chi tiết hóa các trừu tượng theo ý đồ
- Tính module: phân chia DL và chức năng
- Kiến trúc: cấu trúc tổng thể của PM
- Thủ tục: thuật toán để thực hiện chức năng
- Che dấu: điều khiển bằng giao diện

Trừu tượng hóa

- K/n cơ sở trong tư duy của con người
- Là quá trình ánh xạ 1 sự vật/ hiện tượng của thế giới thực thành 1 k/n logic.
- Có nhiều mức trừu tượng hóa khác nhau
 - Cho phép con người tập trung tư duy vào giải quyết vấn đề mà không cần bận tâm đến chi tiết
 - Biểu diễn vấn đề bằng 1 cấu trúc tự nhiên.

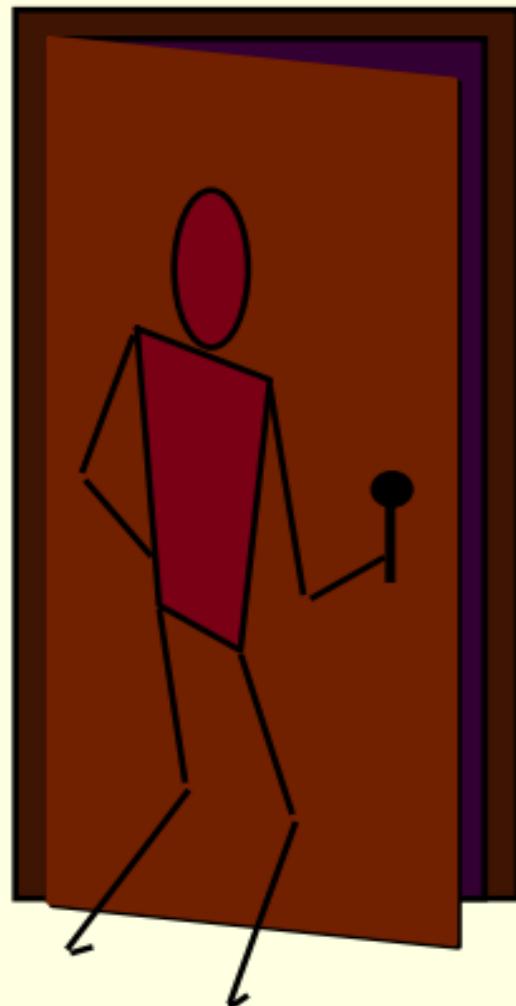
Trùu tượng dữ liệu



CỬA

Mã số: 140DR
Loại: Cửa đơn
Hướng mở: bên trái
Cao: 2.2m
Rộng: 1,0 m
Trọng lượng: 150N
Màu: nâu cánh gián

Trùu tượng thủ tục



Mở cửa

Mô tả chi tiết quá trình vào phòng qua cửa

Làm mịn từng bước

Mở cửa

Bước đến gần cửa
Đưa chìa khóa vào ổ xoay

Mở cửa



Bước qua vào phòng
Đóng cửa lại

Lắp lại cho đẽn khi chốt khóa bật ra
Nếu chốt không mở thì

Rút khóa ra, tìm chìa khác phù
hợp, cắm vào ổ khóa, tiếp tục
xoay cho đẽn khi mở được

Bước qua cửa vào phòng
Đóng cửa lại

Thiết kế module

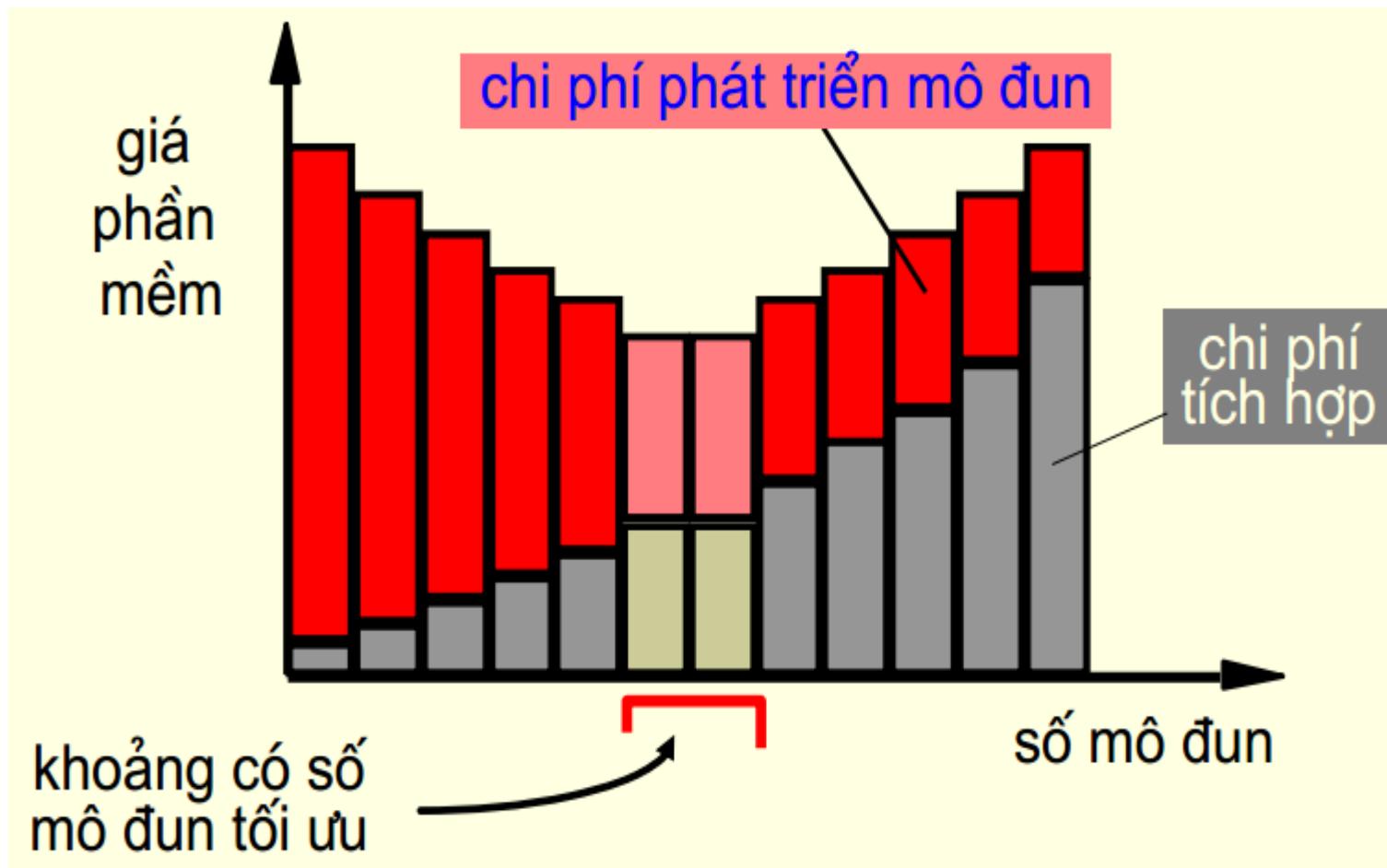
- Dựa trên quan điểm “chia để trị”.
- C: độ phức tạp $C(p_1 + p_2) > C(p_1) + C(p_2)$
- E: công sức thực hiện $E(p_1 + p_2) > E(p_1) + E(p_2)$

- Giảm độ phức tạp
- Cục bộ, dễ sửa đổi
- Có khả năng phát triển song song
- Tính tái sử dụng cao

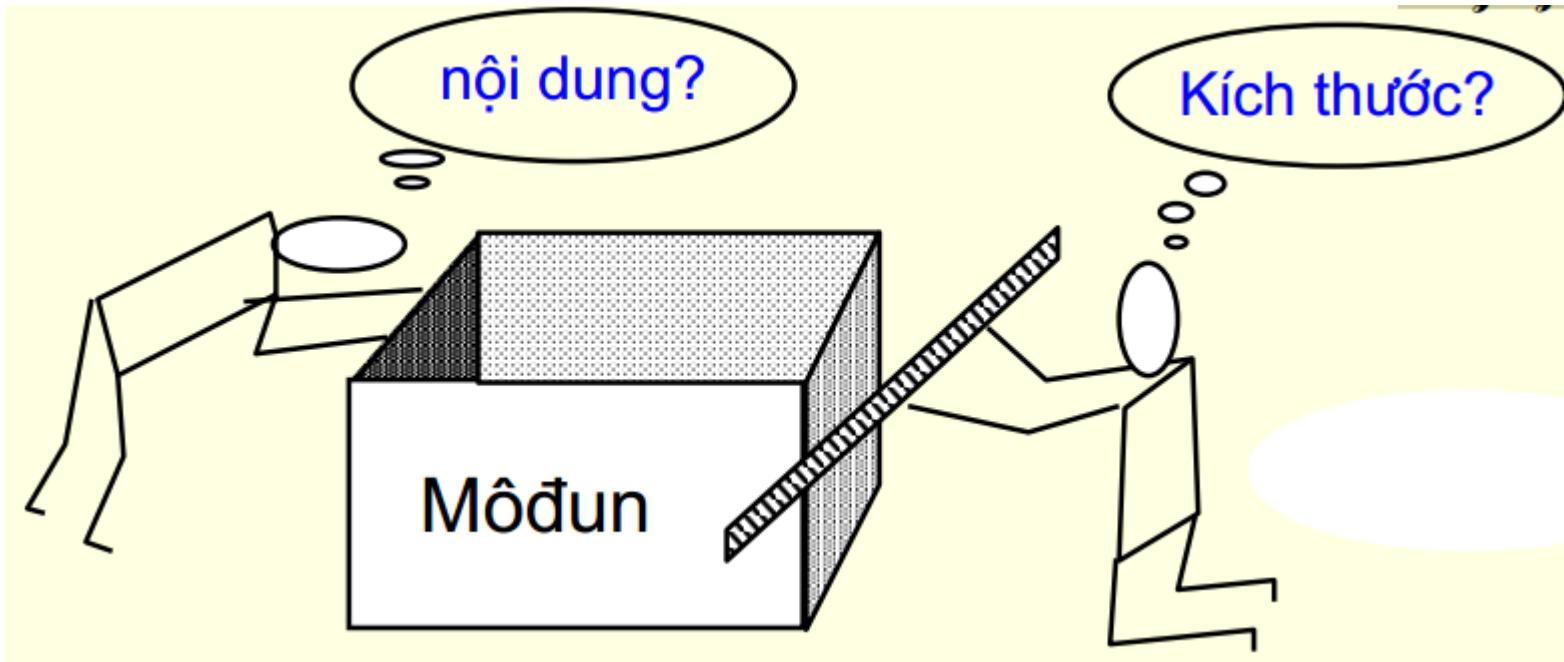


Số lượng module

- Cần xác định số module tối ưu



Kích cỡ module



- Được quyết định dựa trên k/n độc lập chức năng: mỗi module nên thực hiện 1 công việc: **dễ hiểu, dễ sửa đổi, dễ tái sử dụng**

Che giấu thông tin

- Sử dụng module thông qua các giao diện
 - Danh sách tham số và giá trị trả về
- **Không cần biết cách thức cài đặt của nó**
 - Thuật toán
 - Cấu trúc DL
 - Giao diện ngoại lai (module thứ cấp, thư viện vào/ra)
 - Tài nguyên hệ thống

Tại sao lại che giấu thông tin?

- Giảm hiệu ứng phụ khi sửa đổi module
- Giảm tác động của thiết kế tổng thể lên thiết kế cục bộ
- Nhấn mạnh trao đổi thông tin qua giao diện
- Loại bỏ việc sử dụng DL dùng chung
- Hướng tới đóng gói chức năng, **1 thuộc tính của thiết kế tốt**



Tạo ra sản phẩm phần mềm tốt hơn

Chất lượng thiết kế

- 3 đặc trưng cho 1 thiết kế tốt theo [MCG91]:
 - TK phải **triển khai được tất cả yc** trong mô hình phân tích & yc tiềm ẩn mà KH đòi hỏi
 - TK cần **là bản hướng dẫn dễ đọc, dễ hiểu** cho người viết chương trình, người kiểm thử, người bảo trì
 - TK cần **cung cấp 1 bức tranh đầy đủ về PM trên quan điểm triển khai** hướng đến các mặt DL, chức năng và hành vi của hệ thống.

Tiêu chí chất lượng

- Cần thiết lập các tiêu chí kỹ thuật để đánh giá 1 TK là tốt hay không?
 - TK cần **có kiến trúc tốt**: cấu thành từ các mẫu (pattern), các thành phần có đặc trưng tốt, dễ tiến hóa
 - TK **được module hóa** cho mỗi thành phần chức năng
 - TK chứa các **biểu diễn tách biệt** nhau về: DL, kiến trúc, giao diện, thành phần, môi trường.
 - **Liên kết qua giao diện** làm giảm độ phức tạp liên kết giữa các module với nhau, và giữa hệ thống với môi trường.

Độ đo chất lượng thiết kế

- Phụ thuộc vào bài toán, không có phương pháp chung.
- Một số độ đo:
 - **Coupling**: mức độ ghép nối giữa các module
 - **Cohesion**: mức độ liên kết giữa các thành phần trong cùng module
 - **Understandability**: tính hiểu được
 - **Adaptability**: tính thích nghi được

Độ đo chất lượng thiết kế

- **Coupling (ghép nối)**
 - Độ đo sự liên kết (trao đổi DL) giữa các module
 - Ghép nối chặt chẽ thì khó hiểu, khó sửa đổi do phải tính đến các liên kết có thể, dễ gây lỗi lan truyền
- **Cohesion (kết dính)**
 - Đo sự phụ thuộc lẫn nhau giữa các thành phần trong 1 module
 - Kết dính cao thì tính cục bộ cao (độc lập chức năng): dễ hiểu, dễ sửa đổi
- **Tiêu chuẩn thiết kế tốt: ghép nối lỏng, kết dính chặt**

Ghép nối - Coupling

- Mức độ quan hệ của các module:
 - Module nên ghép nối lỏng lẻo
 - Mức lỏng lẻo thể hiện qua loại hình ghép nối.



ghép nối thường
ghép nối dữ liệu
ghép nối nhãn
ghép nối điều khiển

ghép nối chung
ghép nối nội dung

loose and best
still very good
ok
ok

very bad
tight and worst

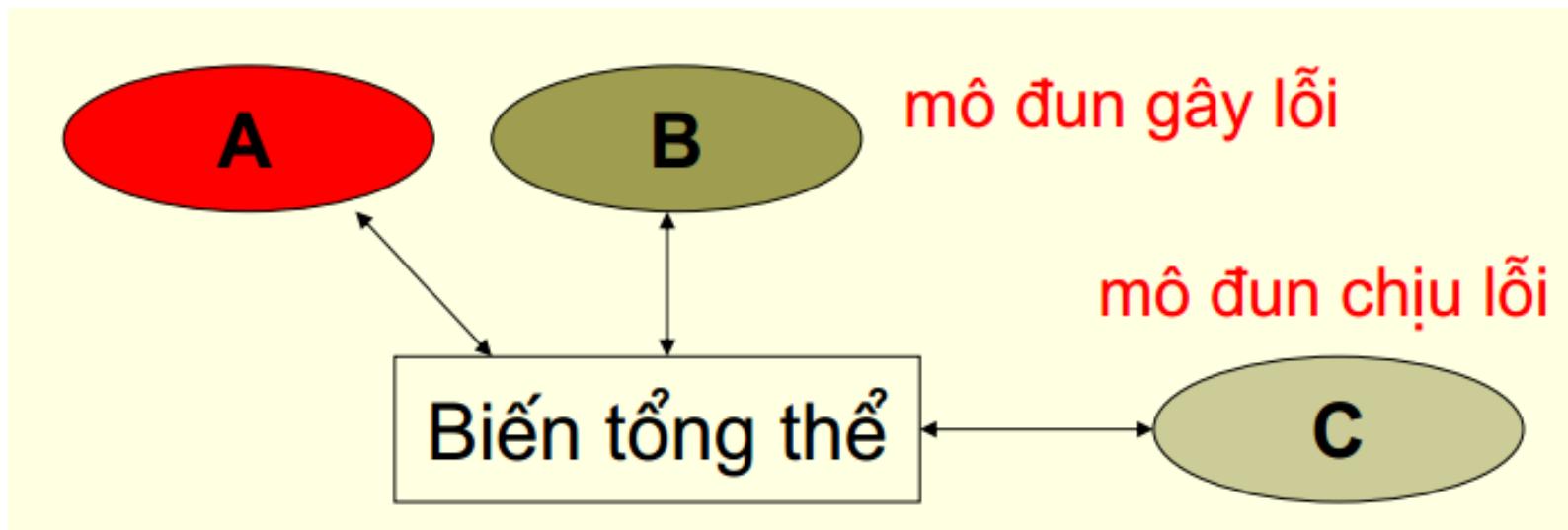
Ghép nối nội dung

- Các module dùng DL hay thông tin điều khiển được duy trì trong 1 module khác
- Là trường hợp xấu nhất. Ví dụ:
 - Các ngôn ngữ bậc thấp chỉ dùng biến chung
 - Lạm dụng lệnh goto trong 1 chu trình

10 k =1	60 print k, y
20 gosub 100	70 stop
30 if y > 120 goto 60	100 Y =3*k*k+7*k-3
40 k = k+1	110 return
50 goto 20	

Ghép nối chung

- Các module trao đổi thông qua biến tổng thể
- 1 module lỗi → ảnh hưởng đến module khác.
- Tính tái sử dụng của module thấp.



Ghép nối điều khiển

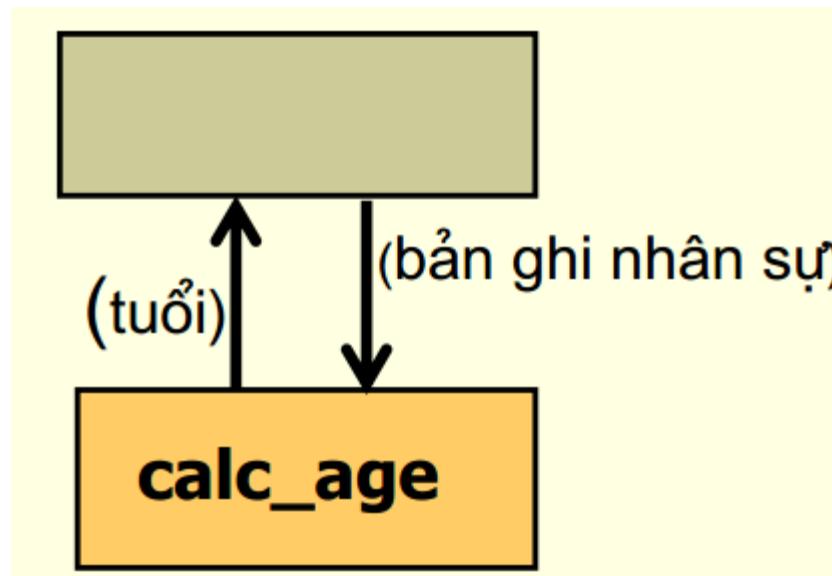
- Các module trao đổi thông tin điều khiển
- Làm cho thiết kế khó hiểu, khó sửa đổi, dễ nhầm

```
void PrintRecipe(){  
    DisplayName(name, sex);  
    ...  
}
```

```
void DisplayName(string name, char  
sex){  
    if(sex == 'm') cout<<"Mr.";  
    else cout<<"Ms.";  
    cout<<name;  
}
```

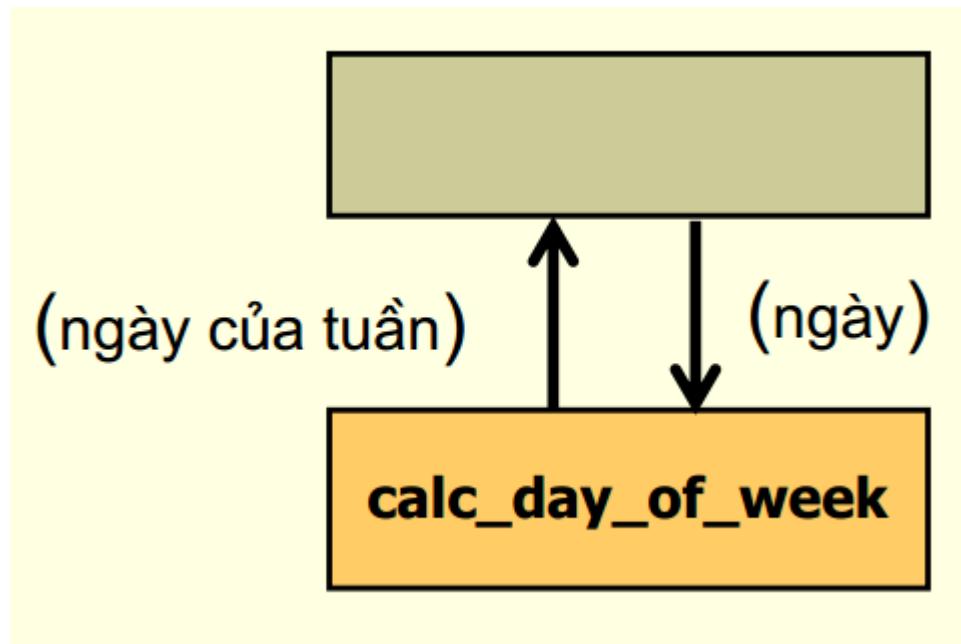
Ghép nối nhãn

- Các module trao đổi **thừa thông tin**
- Module có thể thực hiện chức năng ngoài ý muốn
- Làm giảm tính thích nghi



Ghép nối dữ liệu

- Truyền DL qua tham số
- Nhận kết quả qua tham số và giá trị trả về
- Làm giảm tính thích nghi



Ghép nối thường

- 1 module thực hiện 1 chức năng độc lập
- Hoàn toàn không nhận tham số
- Không có giá trị trả lại

Kết dính - Cohesion

- Mỗi module chỉ nên thực hiện 1 chức năng
- Mọi thành phần của module phải tham gia thực hiện chức năng đó

**chức năng
tuần tự
truyền thông
thủ tục
thời điểm
logic**



high and best
ok
still ok
not bad at all
still not bad at all
still not bad at all
lowest and worst by far

gom góp

Các loại kết dính

- Kết dính chức năng (functional cohesion)
 - Các thành phần cùng góp phần thực hiện 1 chức năng (vd: các thao tác sắp xếp)
- Kết dính tuần tự (sequential cohesion)
 - Output của 1 thành phần là input của thành phần tiếp theo: ảnh màu -> ảnh đen trắng -> ảnh nén
- Kết dính truyền thông (communicational cohesion)
 - Các thành phần truy cập đến cùng tập DL: tính toán, thống kê (max, min, average, variation...)

Các loại kết dính

- Kết dính thủ tục (procedural cohesion)
 - Các thành phần thực hiện theo 1 trình tự xác định (vd: tính lương cơ bản, tính phụ cấp, tính bảo hiểm...)
- Kết dính thời điểm (temporal cohesion)
 - Các thành phần hoạt động cùng thời điểm (vd: hàm khởi tạo thì đọc DL, cấp phát bộ nhớ...)
- Kết dính logic (logical cohesion)
 - Gồm các thành phần làm chức năng logic tương tự (hàm xử lý lỗi chung)
- Kết dính gom góp (coincidental cohesion)
 - Gom các thành phần không liên quan đến nhau

Tính hiểu được - Understandability

- Là kết quả tổng hợp từ nhiều thuộc tính:
 - Cấu trúc rõ ràng, tốt
 - Ghép nối lỏng lẻo
 - Kết dính cao
 - Được lập tài liệu đầy đủ chi tiết
 - Thuật toán, cấu trúc dễ hiểu

Tính thích nghi được - Adaptability

- Hiểu được
 - Sửa đổi được
 - Tái sử dụng được
- Tự chứa
 - Không sử dụng thư viện ngoài
 - Mâu thuẫn với xu hướng tái sử dụng

Thiết kế HĐT và chất lượng

- Thiết kế HĐT hướng tới chất lượng thiết kế tốt:
 - Đóng gói, che giấu thông tin (độc lập DL)
 - Các thực thể hoạt động độc lập (cục bộ, dùng lại)
 - Trao đổi DL qua truyền thông điệp (liên kết yếu)
 - Có tính kế thừa, đa hình, trừu tượng
 - Cục bộ, dễ hiểu, dễ tái sử dụng

Bài tập A

- 1. Các loại thiết kế và giải thích nội dung của nó?
- 2. Mô hình tổng quát trong tiến trình thiết kế?
- 3. Giải thích một số k/n cơ bản trong giai đoạn thiết kế: trừu tượng, làm mịn, module hóa, che giấu thông tin, thủ tục?
- 4. Các đặc trưng của 1 thiết kế tốt?
- 5. Có các độ đo chất lượng thiết kế nào?

Nội dung chính

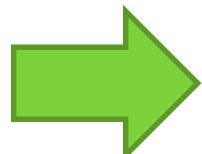
- Khái niệm thiết kế phần mềm
- **Các hoạt động thiết kế**
- Thiết kế hướng đối tượng

Nội dung chính

- Khái niệm thiết kế phần mềm
- **Các hoạt động thiết kế**
 - Thiết kế kiến trúc
 - Thiết kế giao diện
- Thiết kế hướng đối tượng

Thiết kế kiến trúc phần mềm (Software architecture design)

- Kiến trúc phần mềm là gì?
- PM không đơn nhất
 - Có các thành phần con bên trong
- Thắc mắc về các thành phần con:
 - Được tổ chức, sắp xếp ntn?
 - Mqh giữa chúng?
 - Có cấu trúc ra sao?



Kiến trúc phần mềm

Thiết kế kiến trúc phần mềm (Software architecture design)

- Khái niệm kiến trúc phần mềm
 - Kiến trúc PM chỉ **cấu trúc tổng thể** của 1 PM và cách **thức tổ chức** qua đó cho ta 1 sự tích hợp về mặt k/n của hệ thống [SHA95a].
 - Thông thường: **thể hiện bằng 1 lược đồ phân cấp** của các thành phần và quan hệ giữa chúng.
 - Đầy đủ: thể hiện **cấu trúc hệ thống theo nhiều góc nhìn khác nhau**: tĩnh, động, DL, triển khai

Vai trò kiến trúc phần mềm

Tầm quan trọng của kiến trúc:

- Ảnh hưởng đến hiệu quả hoạt động
 - Tính tốc độ (performance)
 - Tính thích ứng (scalability)
 - Tính bảo mật (security)
 - Tính chịu lỗi (fault-tolerance)
- Ảnh hưởng về chi phí
 - Khả năng triển khai
 - Khả năng vận hành
 - Khả năng bảo trì
- Ảnh hưởng về thiết kế và cài đặt

Vai trò kiến trúc phần mềm

- Không phải là mô hình hoạt động
- Là mô hình phân hoạch theo những cách nhìn khác nhau (**chức năng, DL, tiến trình, tĩnh hay động...**)
- Giúp kỹ sư hệ thống:
 - Phân tích tính hiệu quả của thiết kế đáp ứng được yc của PM
 - Tìm các giải pháp thay thế cấu trúc ở giai đoạn sớm
 - Giảm các rủi ro liên quan tới cấu trúc

Khái niệm thiết kế kiến trúc phần mềm

- **Quá trình xác định các hệ con lập thành hệ thống và khung làm việc** để điều khiển và giao tiếp giữa các hệ con với nhau.
- **Bắt đầu sớm** ngay từ giai đoạn đầu của thiết kế hệ thống, tiến hành cùng với 1 số hoạt động đặc tả
- Nó bao gồm việc **xác định các thành phần chính của hệ thống và sự truyền thông giữa chúng**

Các bước thiết kế kiến trúc phần mềm

1. Cấu trúc hóa hệ thống: phân chia hệ thống thành các hệ con (sub-system) độc lập và xác định trao đổi thông tin giữa các hệ con, xác định các giao diện của chúng

2. Mô hình hóa điều khiển: xác lập mô hình điều khiển giữa các thành phần khác nhau của hệ thống đã được xác định.

3. Phân rã thành các module: phân rã các hệ con thành các module.

- **Hệ con:** phần hệ thống hoạt động độc lập với các dịch vụ mà các hệ con khác cung cấp
- **Module:** phần hệ thống cung cấp dịch vụ và tương tác cùng phần khác để tạo ra dịch vụ hay sản phẩm

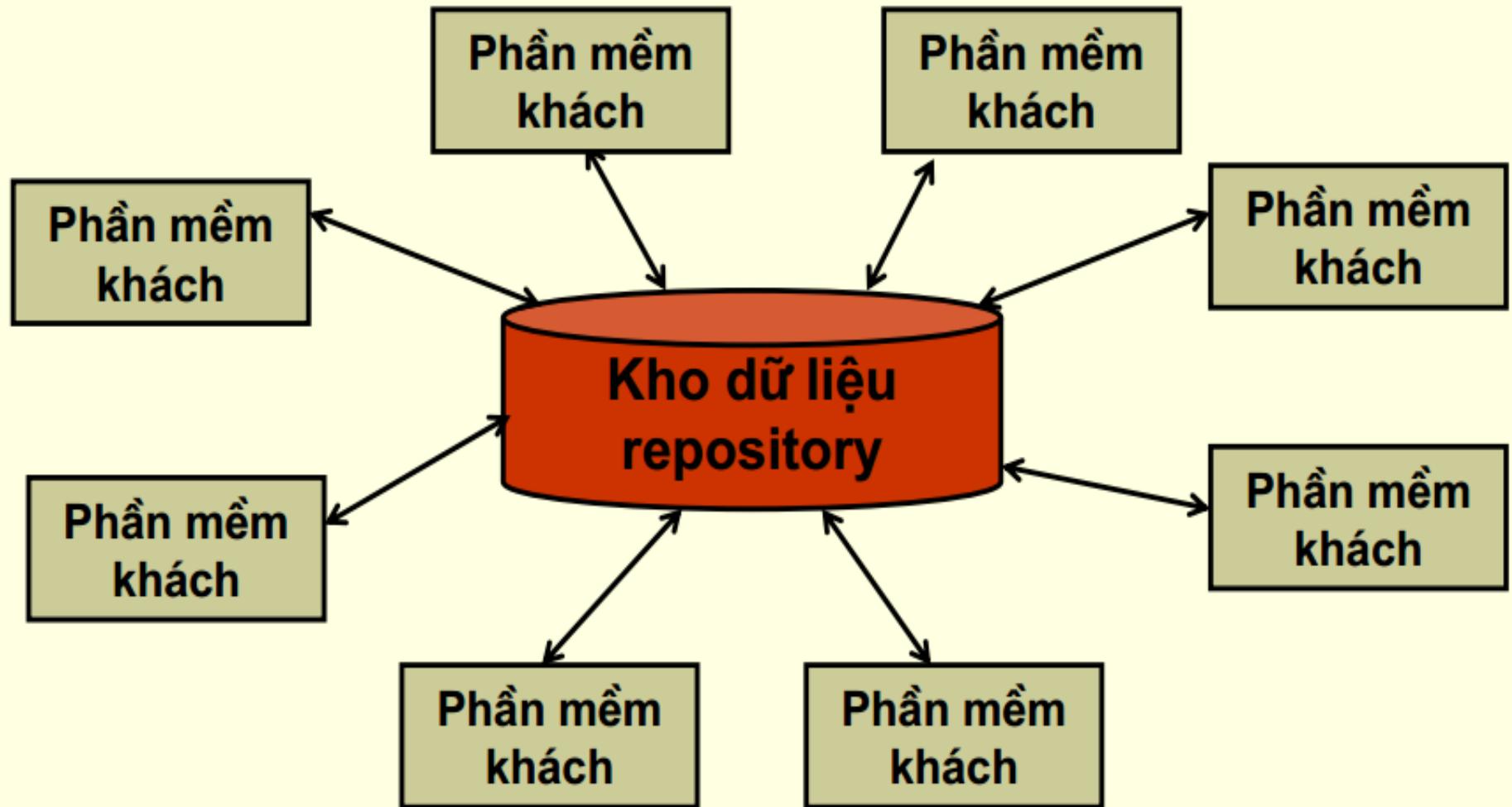
Các mô hình kiến trúc phần mềm

- Các mô hình kiến trúc khác nhau được tạo ra trong quá trình thiết kế.
- Mỗi mô hình biểu diễn 1 cách nhìn của kiến trúc:
 - Mô hình kiến trúc tĩnh: chỉ ra các thành phần chính của hệ thống (BFD)
 - Mô hình động: chỉ ra cấu trúc tiến trình của hệ thống (DFD).
 - Mô hình giao diện: xác định các giao diện của hệ thống (hệ thống giao diện tương tác)
 - Mô hình mqb: các mô hình dữ liệu như ERD

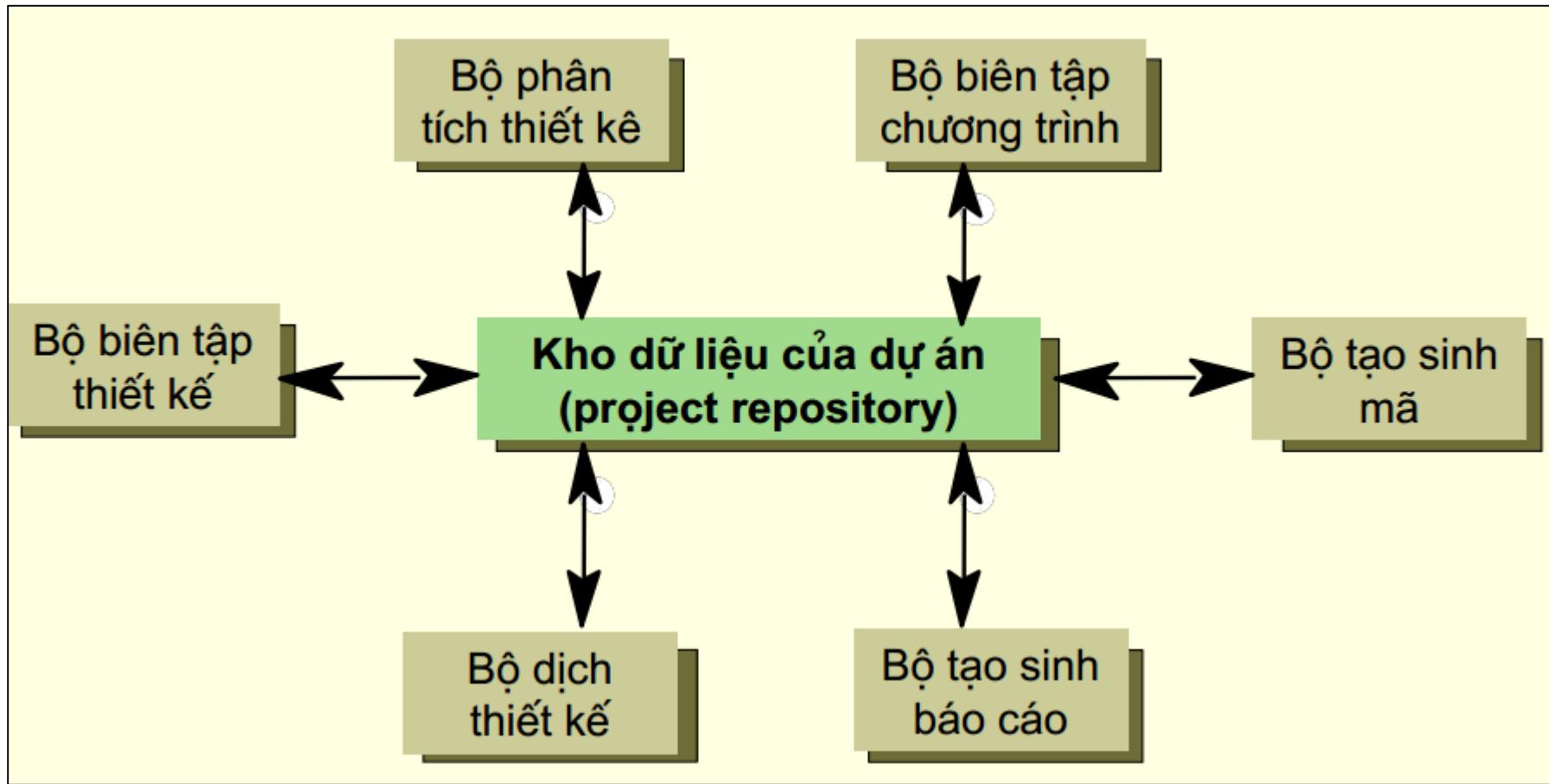
Một số mô hình kiến trúc phần mềm

1. Kiến trúc DL tập trung (Data-centered architectures)
2. Kiến trúc dịch vụ/khách (Client-Server architectures)
3. Kiến trúc phân tầng (Layered architectures)
4. Kiến trúc gọi và trả lại (Call and return architectures)
5. Kiến trúc luồng DL (Data flow architectures)
6. Kiến trúc HĐT (Object-oriented architectures)

Kiến trúc DL tập trung (Data-centered architectures)



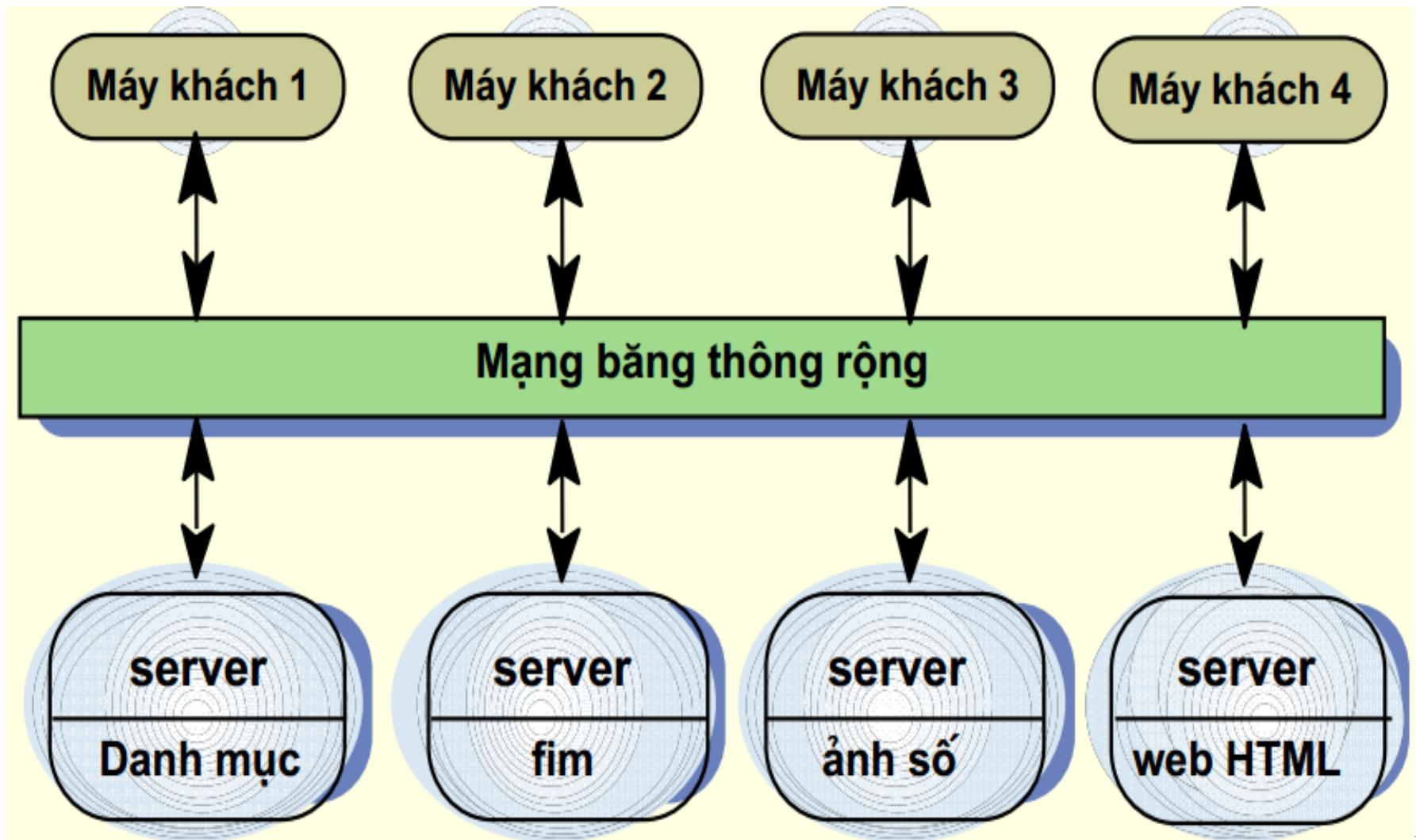
Kiến trúc của bộ công cụ CASE



Kiến trúc DL tập trung (Data-centered architectures)

- **Ưu điểm:**
 - Tiện lợi cho chia sẻ DL lớn
 - Phân hệ không cần biết DL được quản lý và tạo ra ntn?
(sao lưu, bảo trì, bảo mật...)
- **Nhược điểm:**
 - Các hệ con phải theo mô hình DL của kho
 - Việc tiến hóa DL là khó khăn và tốn kém
 - Khó có chính sách quản lý riêng cho từng hệ con
 - Khó phân bố DL một cách hiệu quả

Kiến trúc Client-Server



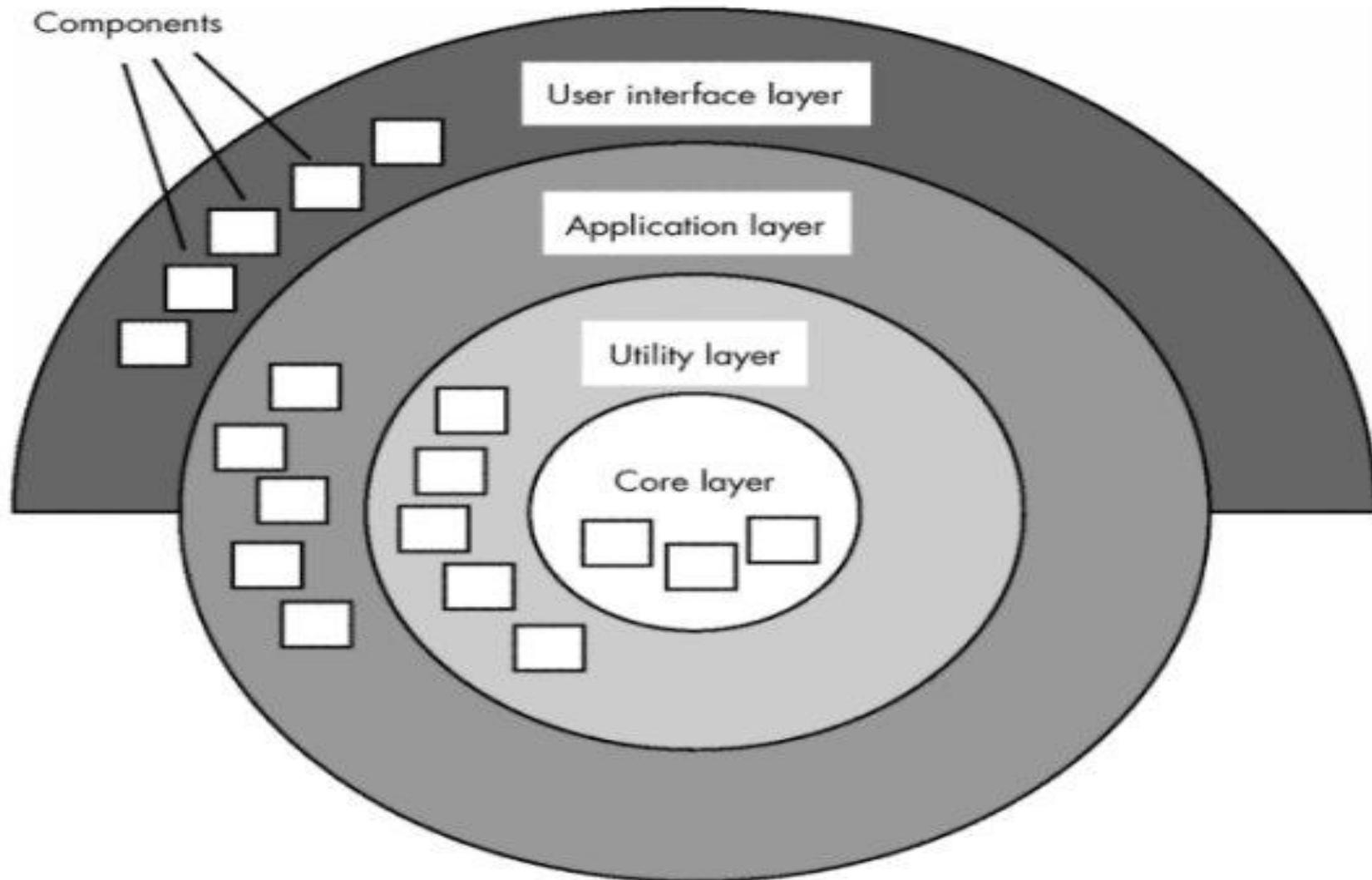
Kiến trúc Client-Server

- **Ưu điểm:**
 - Phân phối DL trực tiếp
 - Sử dụng hiệu quả mạng, dùng các thiết bị rẻ hơn
 - Dễ dàng mở rộng, tiến hóa
- **Nhược điểm:**
 - Các hệ con dùng CTDL khác nhau không chia sẻ được, trao đổi DL có thể không hiệu quả
 - Quản lý ở mỗi server là dư thừa
 - Không lưu trữ chung tên và dịch vụ → khó tìm server hay dịch vụ lỗi



Đang là mô hình phát triển ứng dụng phổ biến

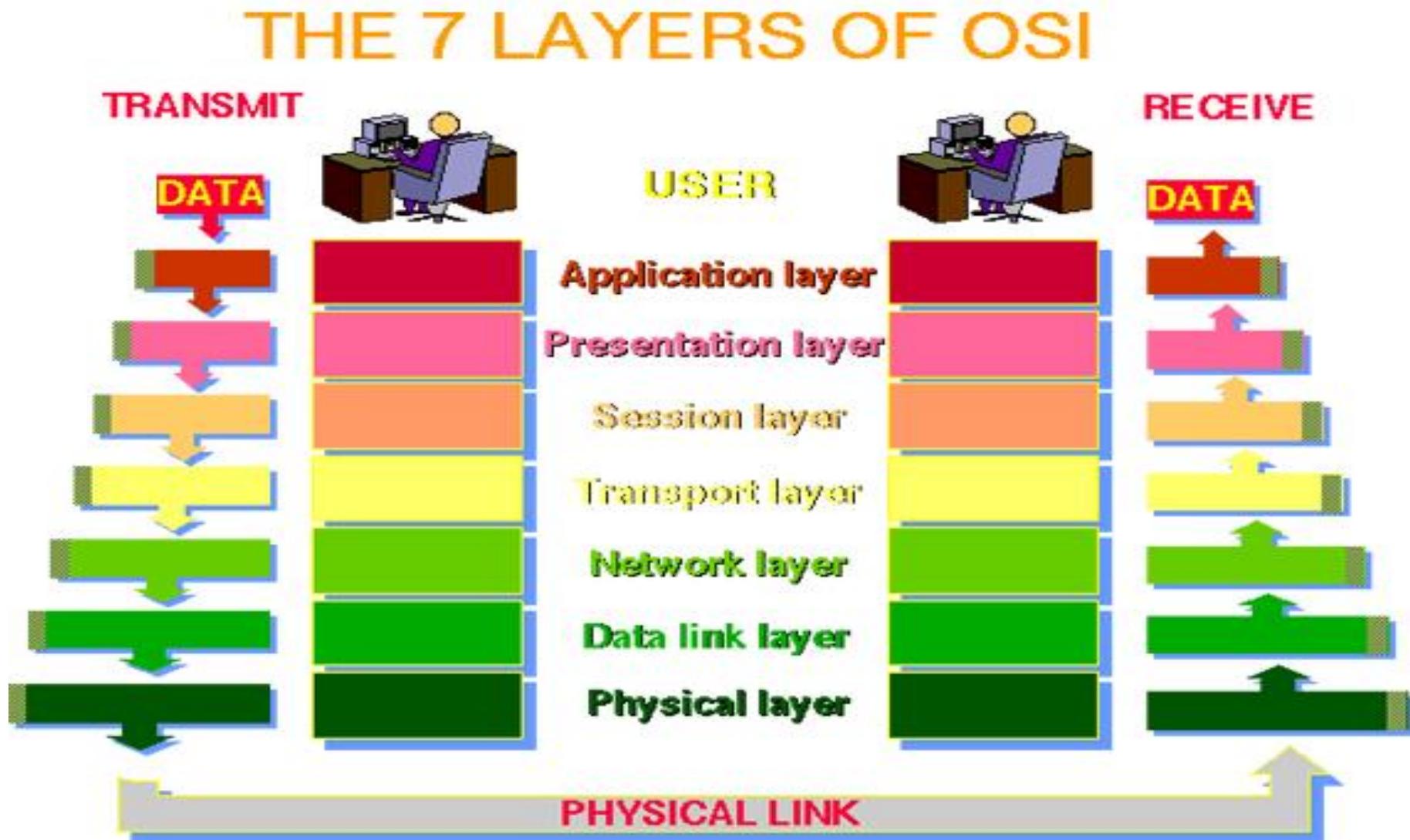
Kiến trúc phân tầng (Layered Architectures)



Kiến trúc phân tầng (Layered Architectures)

- Dùng để mô hình hóa giao diện các phân hệ
- Phân rã hệ thống thành các tầng, mỗi tầng là 1 tập các dịch vụ
- Hỗ trợ sự tăng trưởng, phát triển của các tầng, khi giao diện mỗi tầng thay đổi thì chỉ ảnh hưởng tới các tầng liền kề
- Không phải hệ thống nào cũng dễ dàng phân chia theo mô hình này

Kiến trúc phân tầng (Layered Architectures)

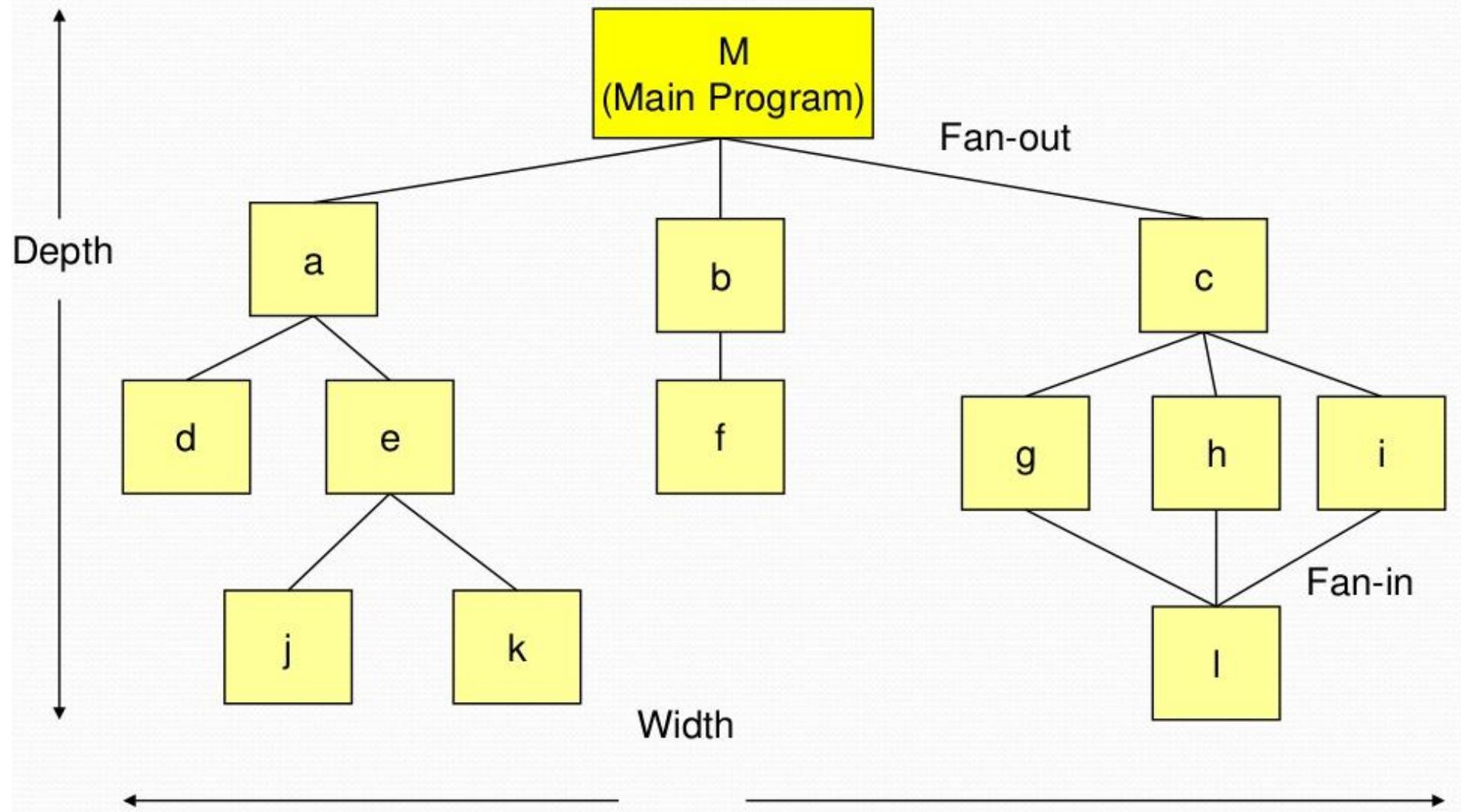


Kiến trúc gọi - trả lại

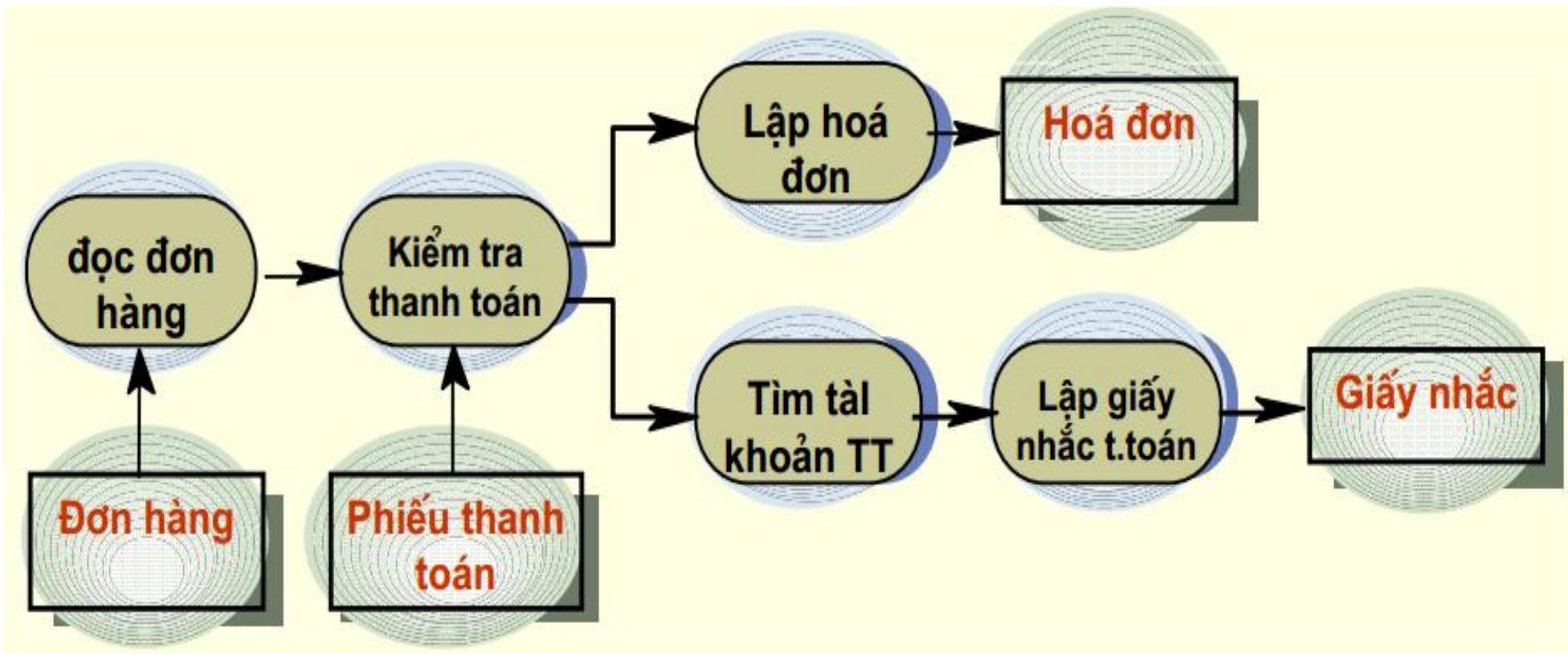
Có 2 loại:

- Kiến trúc chương trình chính/ chương trình con (main program/ sub program architecture)
- Kiến trúc gọi thủ tục từ xa (remote procedure call architecture): các thành phần được phân bổ trên nhiều máy tính

Kiến trúc gọi - trả lại

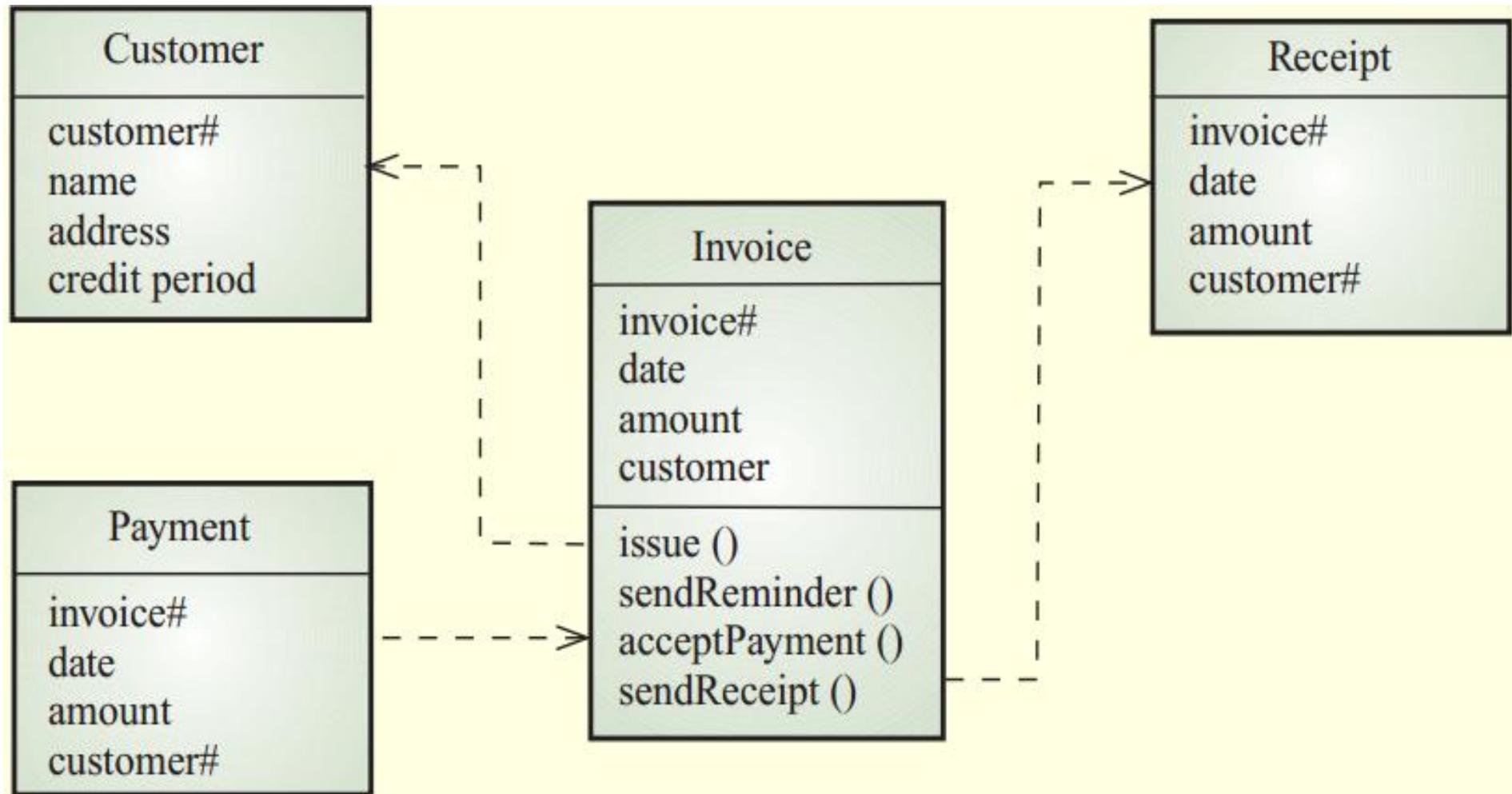


Kiến trúc luồng dữ liệu



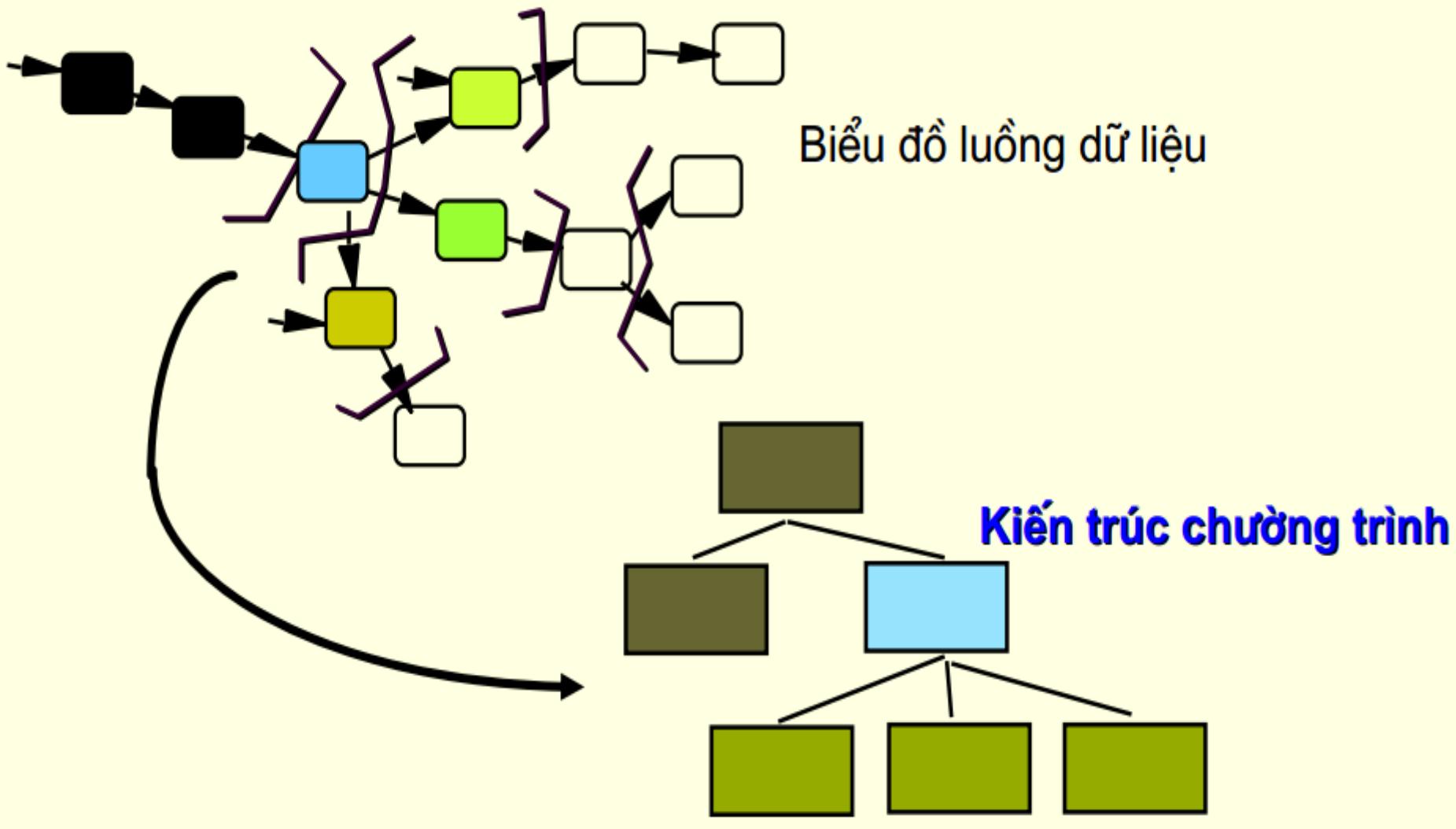
Hệ thống xử lý đơn hàng

Kiến trúc HĐT



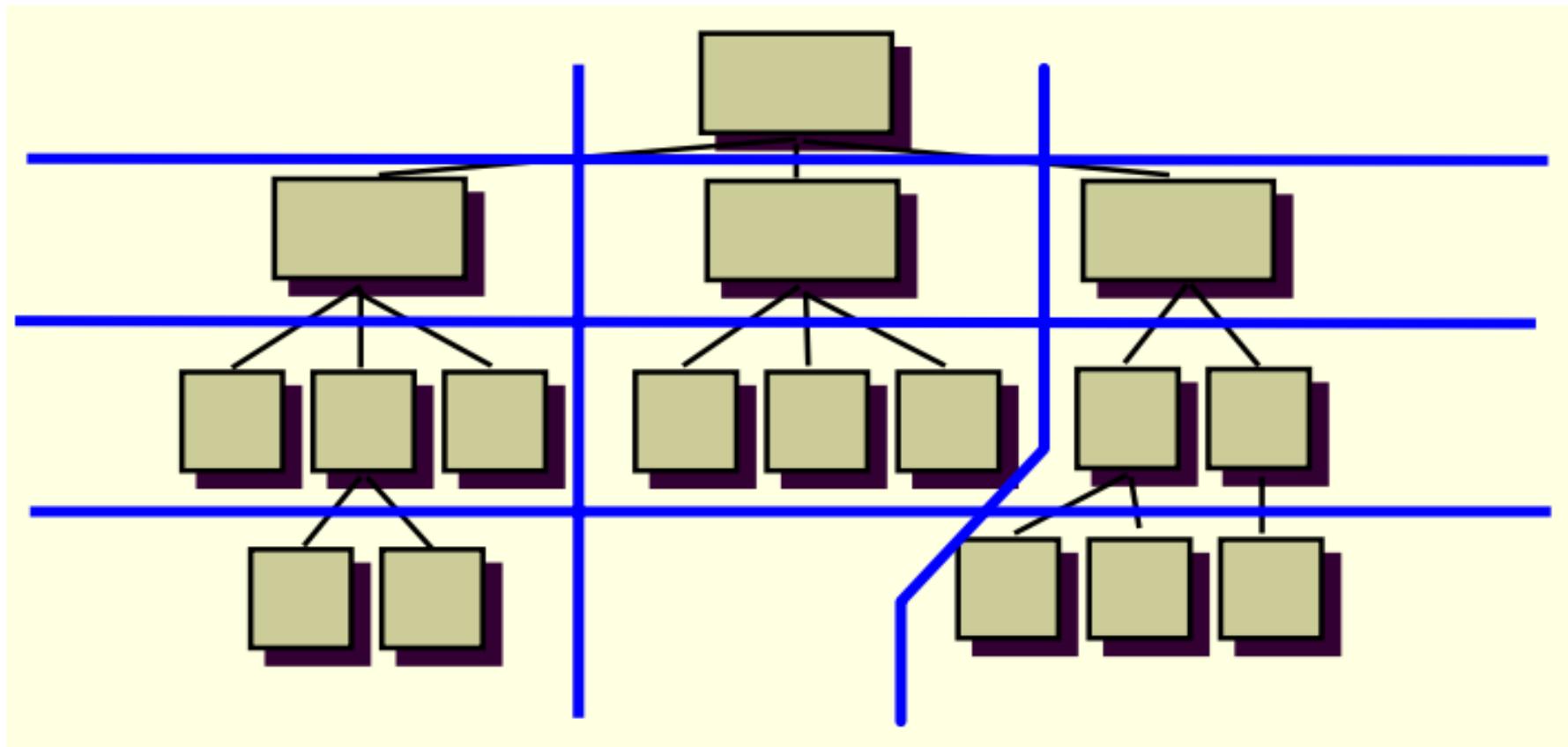
Hệ thống xử lý đơn hàng

Xây dựng kiến trúc chương trình



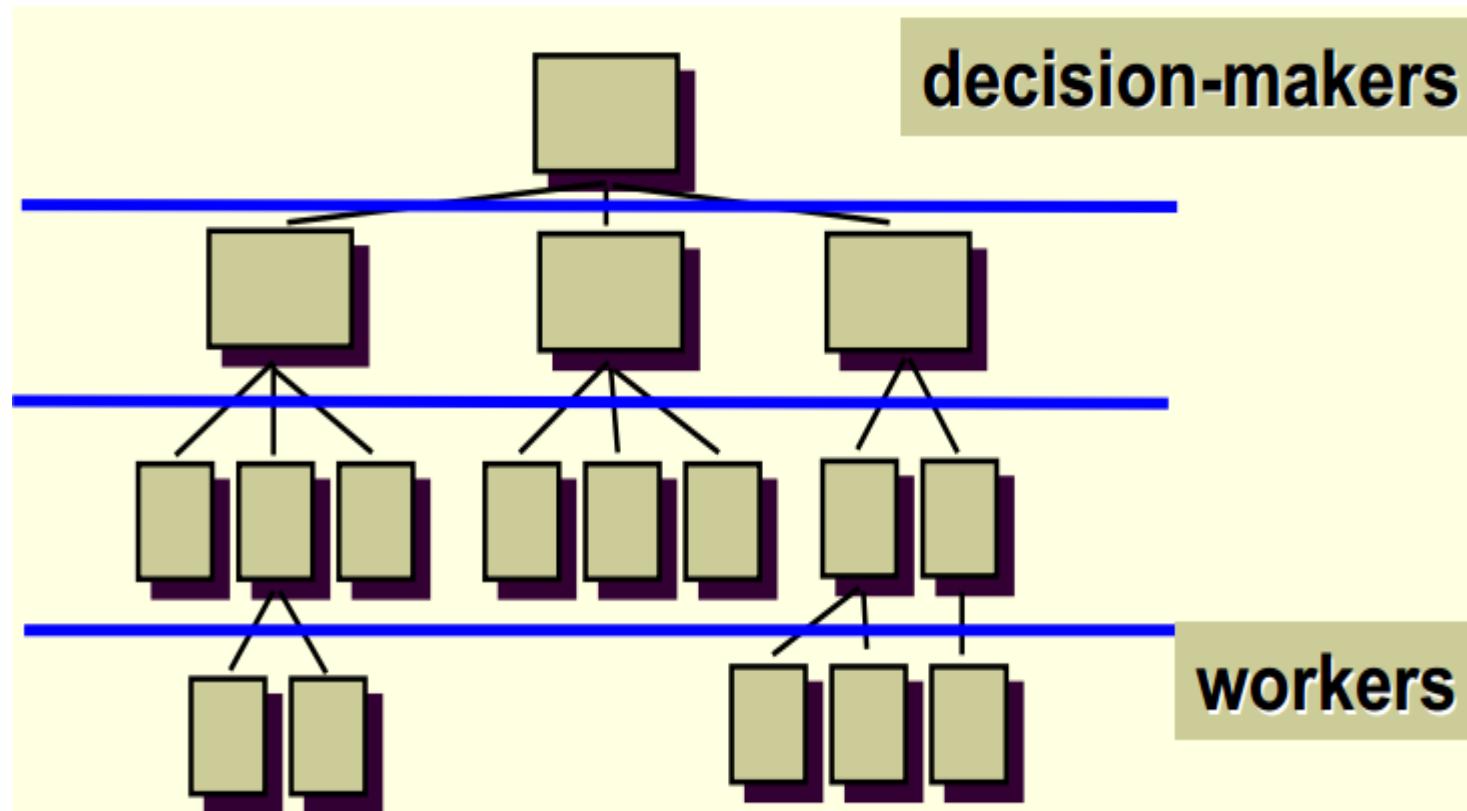
Phân hoạch kiến trúc

- Cần phân hoạch kiến trúc theo **chiều ngang** và **chiều dọc**



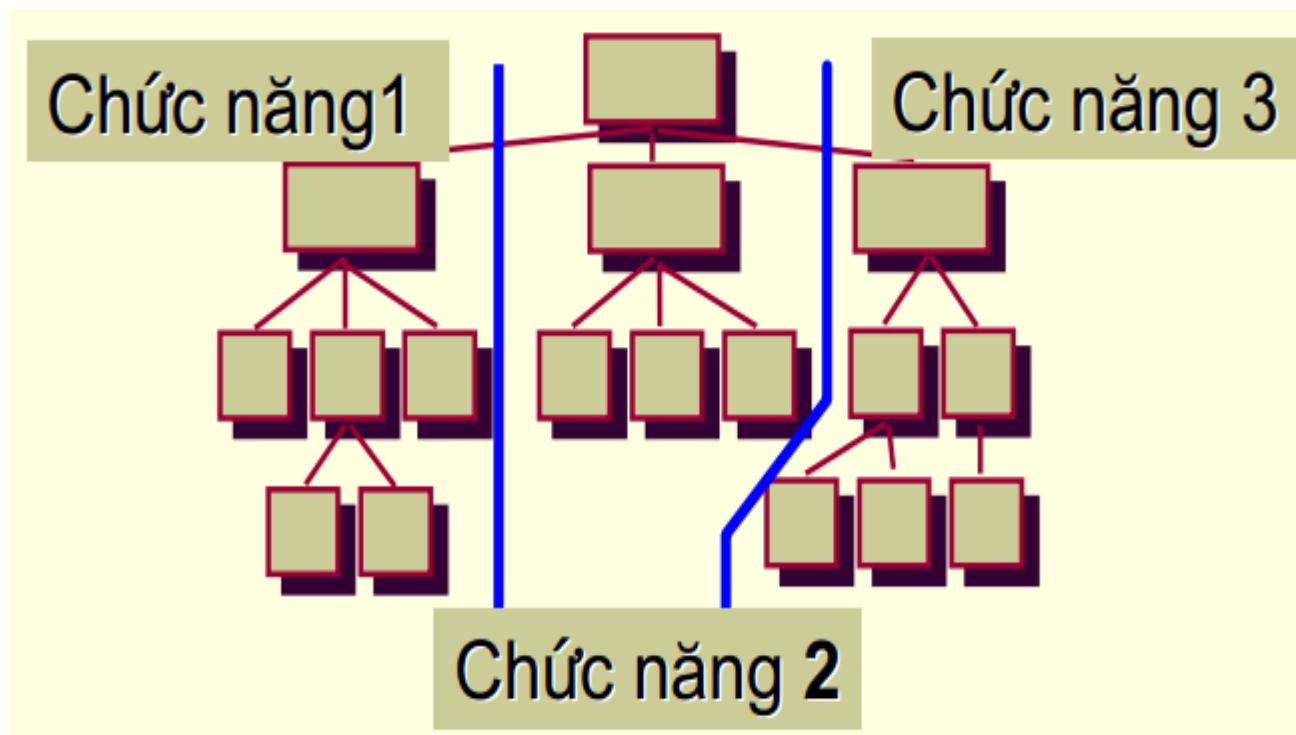
Phân hoạch kiến trúc ngang

- Phân tầng các module thành từng mức: ra quyết định (điều khiển) và module thao tác.
- Các module ra quyết định cần phải xếp ở tầng cao



Phân hoạch kiến trúc dọc

- Xác định các nhánh riêng cho các chức năng chủ chốt.
- Sử dụng các module điều khiển để điều phối thông tin giữa các chức năng



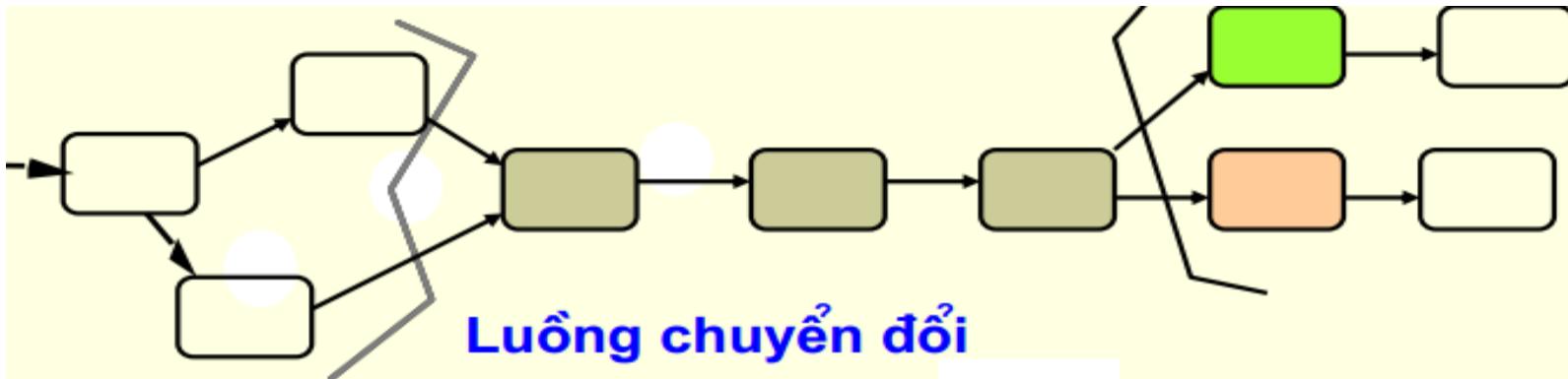
Tại sao cần phân hoạch kiến trúc?

- Để tạo ra phần mềm:
 - Dễ kiểm thử
 - Dễ bảo trì
 - Hạn chế hiệu ứng phụ khi sửa đổi
 - Dễ mở rộng

Thiết kế cấu trúc chương trình

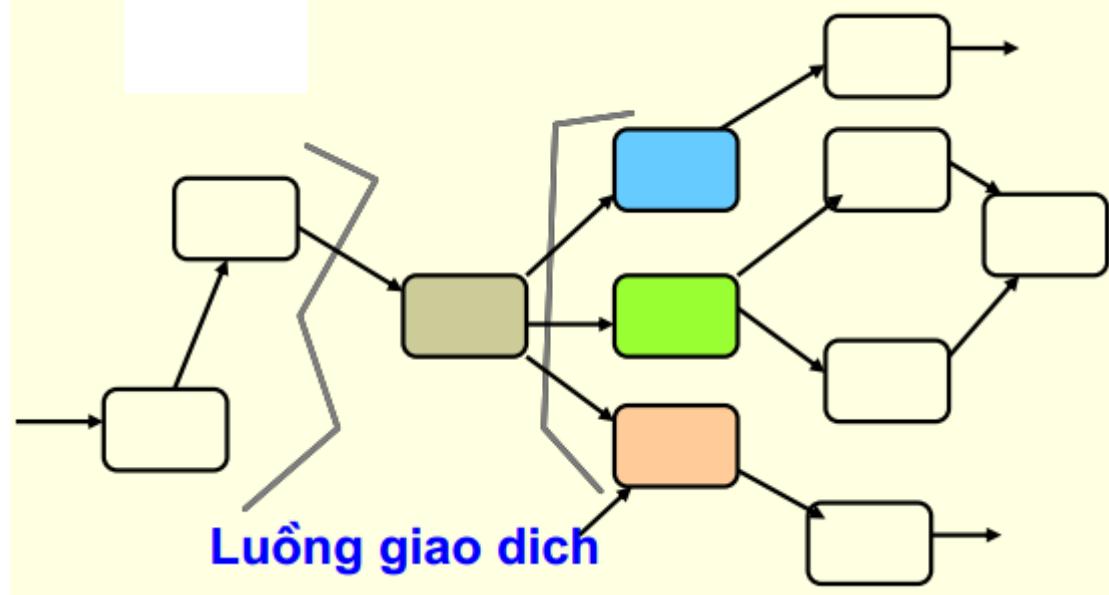
- **Mục tiêu:** Tạo ra module có kiến trúc tốt: được phân hoạch hợp lý, liên kết qua điều khiển.
- **Các tiếp cận:**
 - Chuyển đổi (mapping) DFD thành kiến trúc PM
- **Ký pháp:** lược đồ có cấu trúc (structure chart)

Đặc tính của luồng dữ liệu



2 luồng DL tiêu biểu:

- **Luồng chuyển đổi:** xử lý trung tâm
- **Luồng giao dịch:** định tuyến phân phối

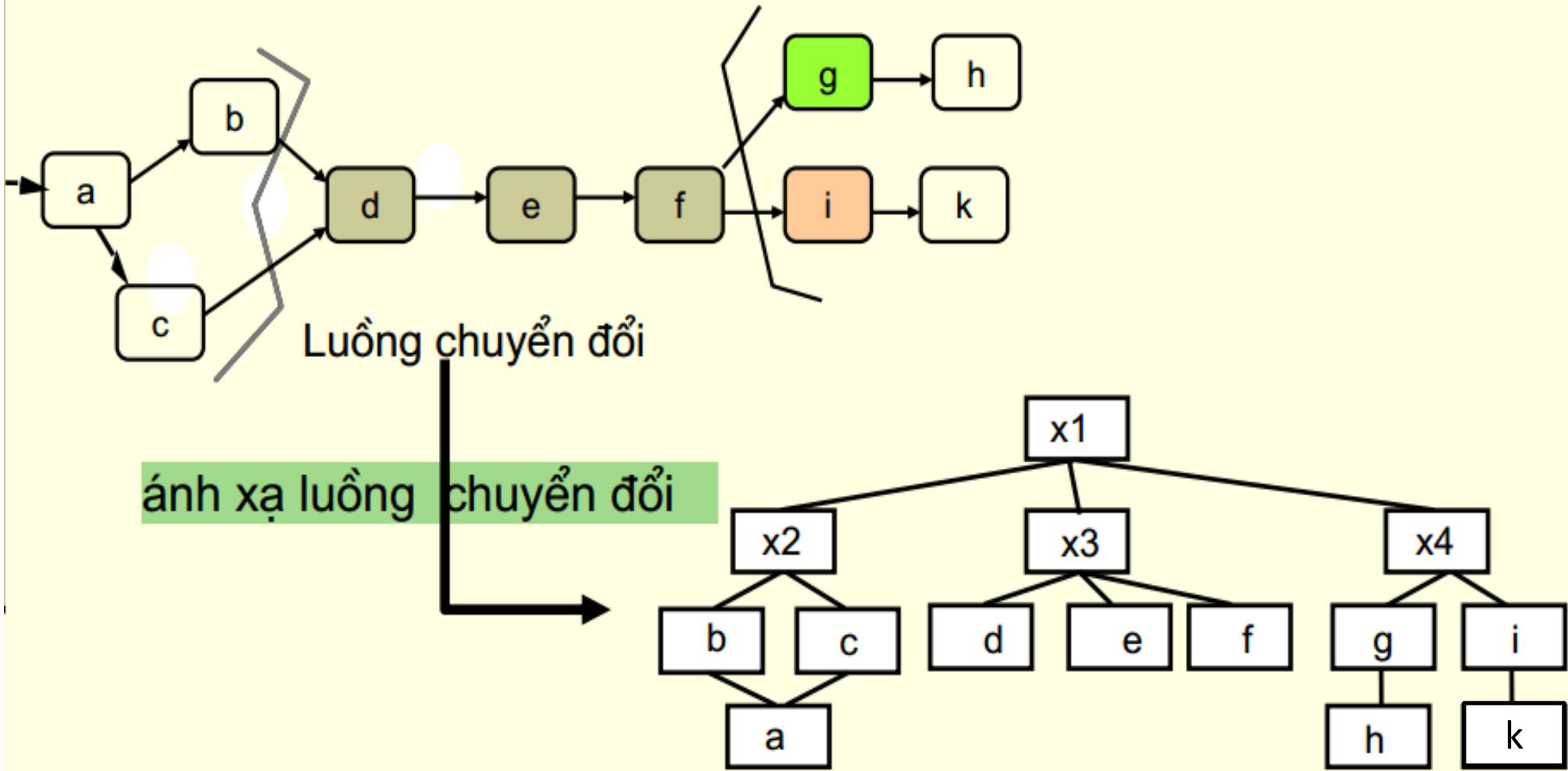


Kiến trúc luồng chuyển đổi

- Cô lập, xác định biên của các module vào/ra.
- Xác định các module **xử lý tập trung**.
- Chuyển chúng thành các module kiến trúc tương ứng
- Thêm các module điều khiển nếu cần thiết
- Tinh chỉnh (refining) kiến trúc để nâng cao tính module

Luồng chuyển đổi tập trung là luồng mà có các DL đầu vào **được tập trung xử lý ở 1 số tiến trình** rồi cho kết quả đầu ra là các tiến trình thực hiện lưu trữ, truyền đi hay biểu diễn thông tin

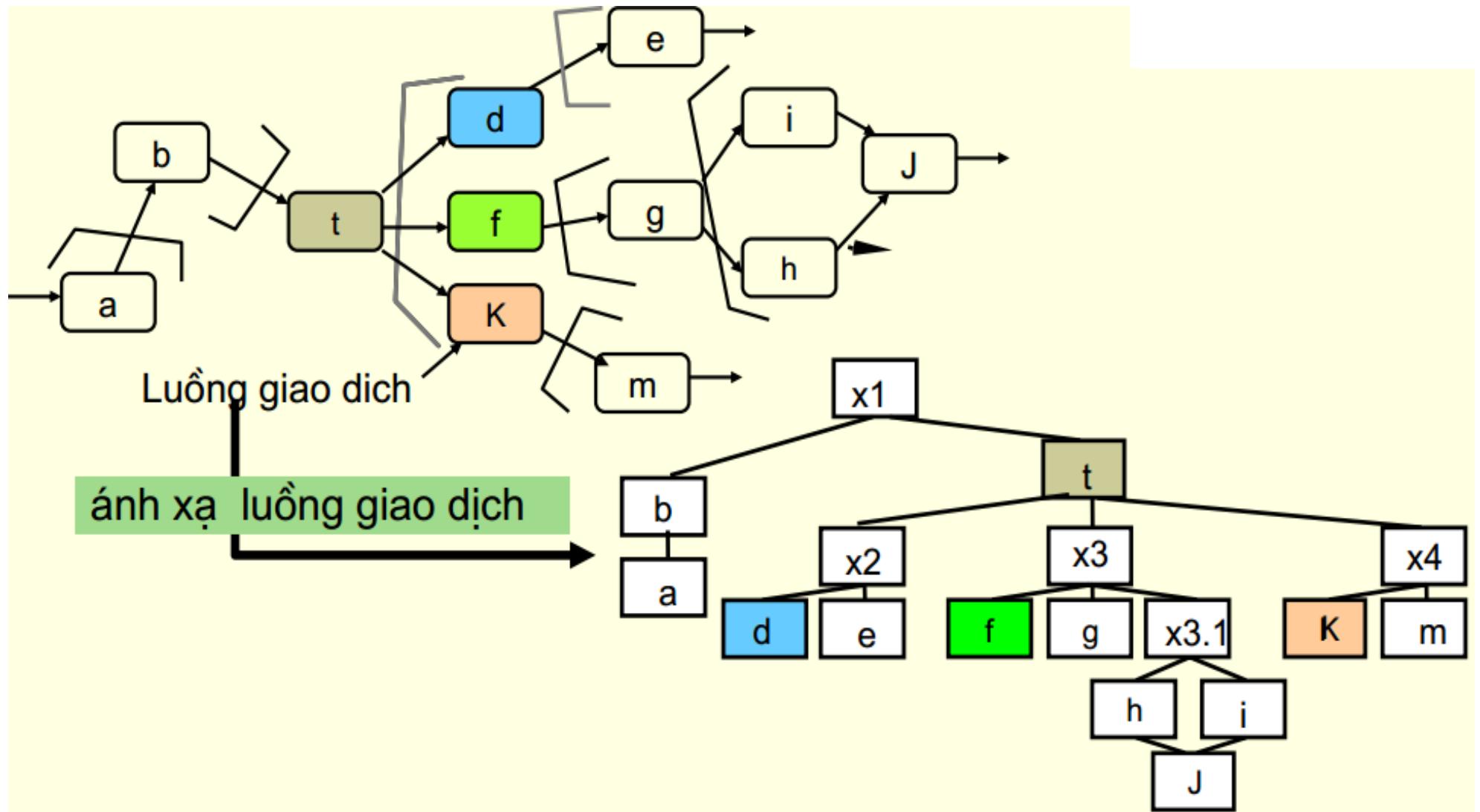
VD: Kiến trúc luồng chuyển đổi



Kiến trúc luồng giao dịch

- Xác định các luồng vào
 - Xác địnhc các luồng thực hiện
 - Xác định **trung tâm giao dịch** (module phân phối)
 - Biến đổi riêng rẽ từng luồng hành động
-
- **Luồng giao dịch** là luồng mà **mỗi đầu vào được nhận dạng và chuyển** cho các tiến trình xử lý tương ứng với nó. Tiến trình làm nhiệm vụ nhận dạng và chuyển DL đến nơi cần gọi là **trung tâm giao dịch**

VD: Kiến trúc luồng giao dịch



Nội dung chính

- Khái niệm thiết kế phần mềm
- **Các hoạt động thiết kế**
 - Thiết kế kiến trúc
 - Thiết kế giao diện
- Thiết kế hướng đối tượng

Thiết kế giao diện

- Nội dung và hình thức trình bày các màn hình giao tiếp của PM
- Hệ thống các thao tác mà người dùng thực hiện trên màn hình giao tiếp và xử lý tương ứng của PM

Kết quả của Thiết kế giao diện

- Danh sách các màn hình
- Sơ đồ liên kết các màn hình
- Đặc tả chi tiết từng màn hình
 - Đặc tả các đối tượng trên màn hình
 - Danh sách các biến cố và cách xử lý tương ứng

Thiết kế giao diện

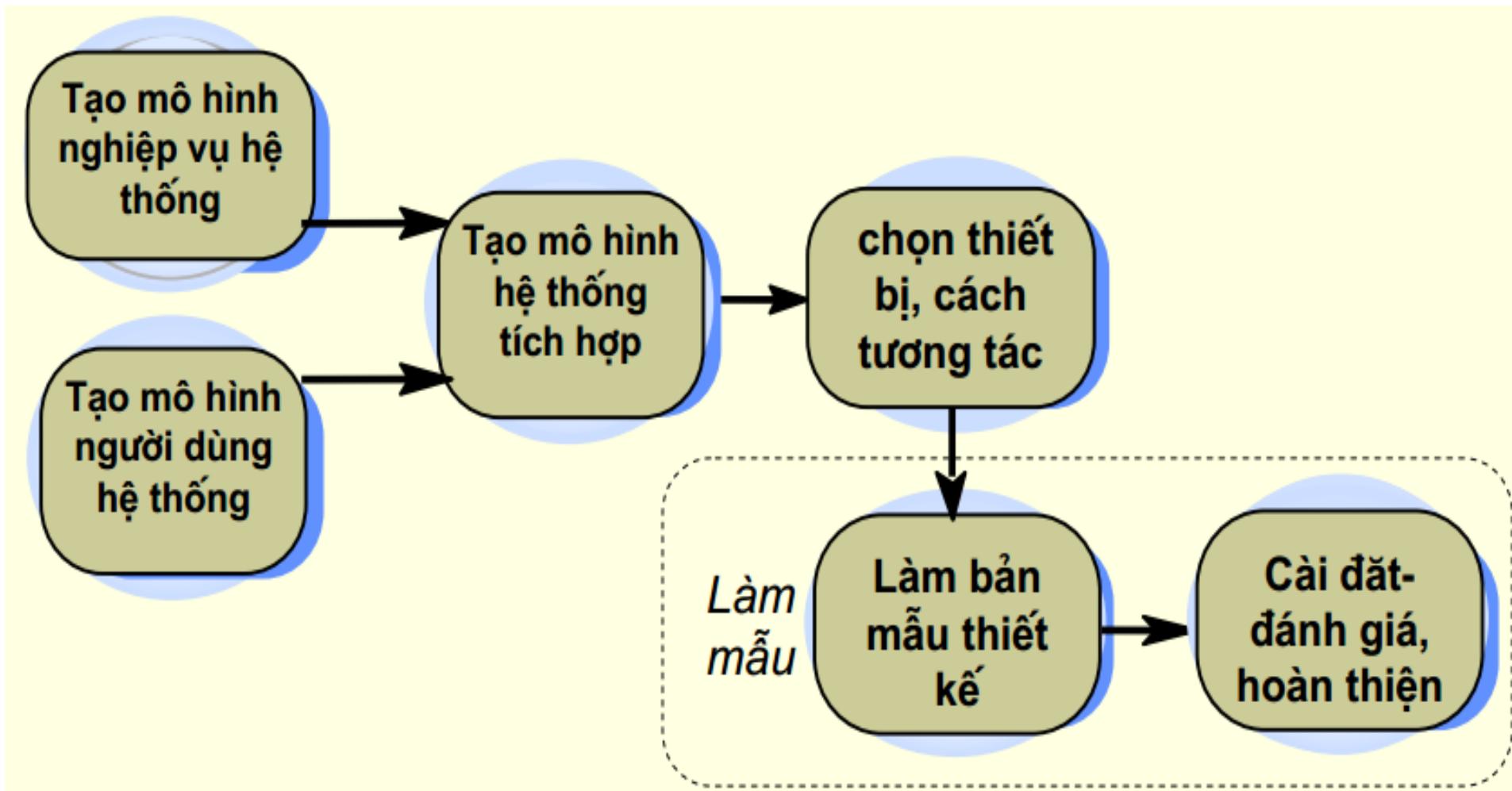
Vai trò, tầm quan trọng

- Một khâu không thể thiếu trong thiết kế PM.
Người dùng đánh giá PM qua giao diện
- Thiết kế giao diện:
 - Hướng người dùng
 - Che dấu chi tiết kỹ thuật bên trong
 - Kết hợp 3 mặt: người dùng, chức năng, công nghệ

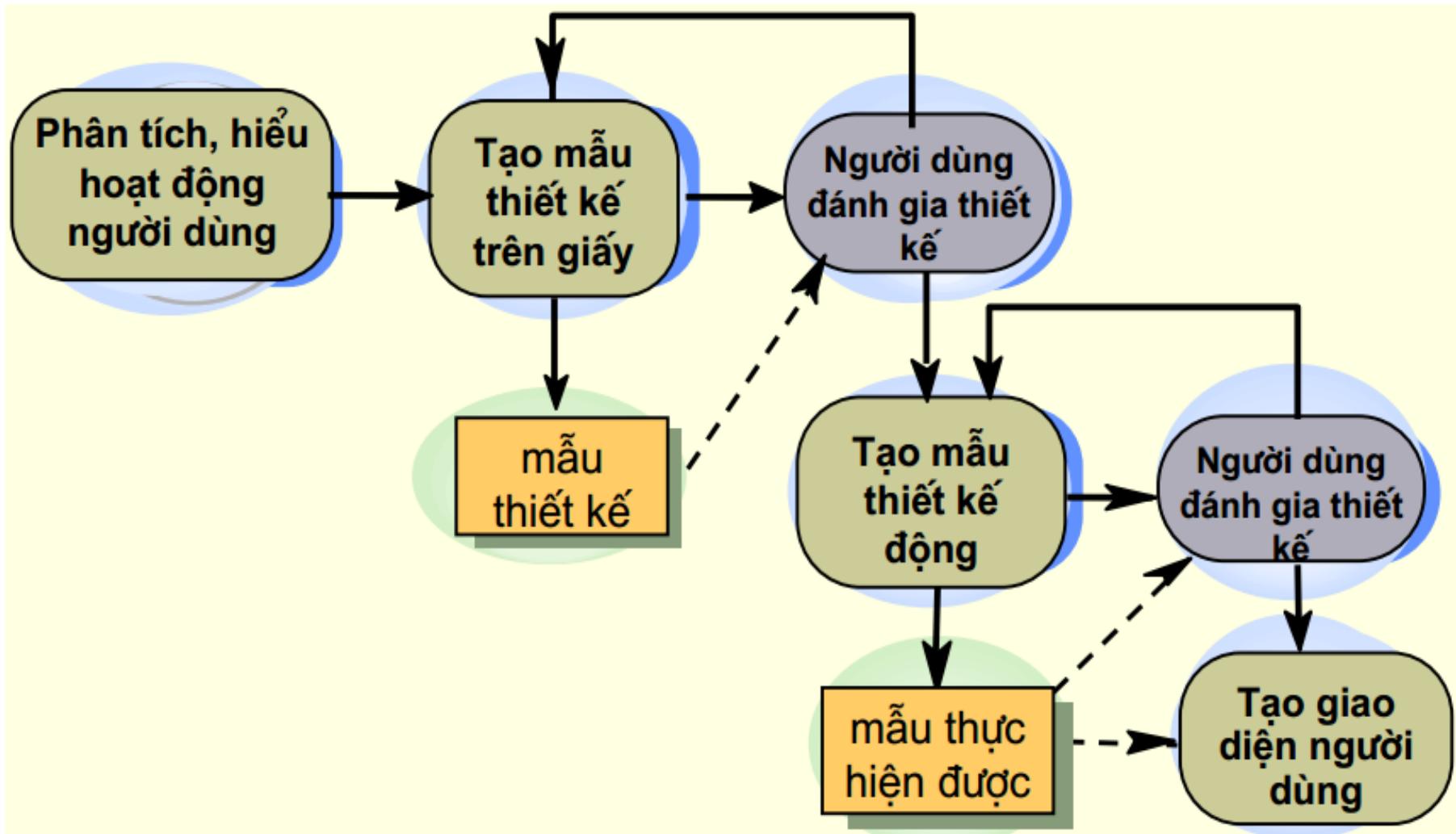
Thiết kế giao diện

- Là phương tiện để người dùng sử dụng hệ thống
 - Giao diện thiết kế nghèo nàn: người dùng dễ mắc lỗi
 - Giao diện thiết kế tồi là lý do nhiều PM không được sử dụng
- Giao diện trợ giúp người dùng làm việc với khả năng của họ
 - Giao diện trợ giúp tốt: người dùng thành công
 - Giao diện trợ giúp tồi: người dùng khó khăn, thất bại

Tiến trình thiết kế giao diện chung



Tiến trình thiết kế giao diện làm mẫu



Nguyên tắc thiết kế giao diện

Thiết kế giao diện cần phải:

DỄ HỌC?

DỄ DÙNG?

DỄ HIỂU?



Nguyên tắc thiết kế giao diện

- Cần phản ảnh vào thiết kế:
 - Kinh nghiệm, năng lực, nhu cầu của người dùng
 - Khả năng dùng bàn phím, chuột, bút
 - Tốc độ phản ứng, khả năng nhớ thao tác.
 - Sở thích, văn hóa, lứa tuổi: màu sắc, ngôn ngữ...
 - Những hạn chế về mặt vật chất và tinh thần của người dùng (trí nhớ, vụng về... có thể mắc lỗi)
- Luôn bao gồm việc làm bản mẫu để người dùng đánh giá.

Các nguyên tắc thiết kế giao diện

- Giao diện cần có các tính chất sau đây:
 - **Tính thân thiện:** thuật ngữ, k/n, thói quen, trình tự nghiệp vụ của người dùng
 - **Tính nhất quán:** vị trí hiển thị, câu lệnh, thực đơn, biểu tượng, màu sắc
 - **Ít gây ngạc nhiên**
 - Có **cơ chế phục hồi** tình trạng trước lỗi
 - Cung cấp kịp thời **phản hồi và trợ giúp** mọi lúc, mọi nơi
 - Tiện ích **tương tác đa dạng**

Lỗi đặc trưng

Thiếu toàn vẹn

Phải nhớ quá nhiều

Không có hướng dẫn, trợ giúp

Không nhạy với ngữ cảnh

Đáp ứng nghèo nàn

Không thân thiện, khó hiểu



Thiết bị tương tác

- Các thiết bị tương tác thường gặp:
 - Màn hình
 - Bàn phím
 - Chuột, bút từ...
 - Màn hình cảm ứng
 - Mic/ Speaker
 - Smart cards

Các kiểu tương tác

- Các kiểu tương tác thông dụng:
 - Thao tác trực tiếp
 - Chọn thực đơn
 - Chọn biểu tượng
 - Điền vào mẫu biểu
 - Ngôn ngữ lệnh
 - Ngôn ngữ tự nhiên

Các loại giao diện truyền thống

- Giao diện dòng lệnh (giao diện hỏi đáp)
- Giao diện đồ họa (GUI)

Giao diện dòng lệnh

- Là phương thức tương tác ra đời sớm nhất
- Nhập lệnh/ DL từ bàn phím
- Dễ cài đặt so với GUI
 - Thực hiện thông qua hàm chuẩn của ngôn ngữ
 - Không tốn tài nguyên hệ thống
- Có khả năng tổ hợp lệnh để tạo lệnh phức tạp
 - Phối hợp các filter, tạo các lô xử lý (batch)
 - Có thể lập trình bằng Shell
 - Có thể tự động hóa.

Giao diện dòng lệnh

- Thao tác thực hiện tuần tự
 - Khó sửa lỗi thao tác trước đó
- Không phù hợp với người dùng ít kinh nghiệm
 - Khó học, khó nhớ
 - Dễ nhầm
 - Đòi hỏi kỹ năng sử dụng bàn phím

Giao diện dòng lệnh

◀ Learn the Command Line

```
$ ls
action comedy drama genres.txt
$ cd comedy
$ cd ..
$ cd comedy/slapstick
$ pwd
/home/ccuser/workspace/movies/comedy/slapstick
$ ls
waterboy.txt zoolander.txt
$ rm waterboy.txt
$ ls
zoolander.txt
$ cd ..
$ rm -r slapstick
$ ls
satire the-office.txt
$ pwd
/home/ccuser/workspace/movies/comedy
$
Display all 1515 possibilities? (y or n)
!
./
2to3
2to3-2.7
2to3-3.4
:
ControlPanel
MAKEDEV
[
[[

htmldiff          rarp
htop              raw
htpasswd          rbash
httparty          rcp
htttx2dbm         rdoc
hwclock           rdoc1.9.1
i386              rdoc2.0
iconv              read
iconvconfig       readarray
icu-config        readelf
icuinfo           readline-editor
```

Giao diện đồ họa

- Thông dụng trên PC, Apple, Unix WS
- Dễ học, dễ sử dụng, thuận tiện với người ít kinh nghiệm
- Có nhiều cửa sổ, có thể tương tác song song
- Hiển thị, tương tác DL trên nhiều vị trí trong cửa sổ
- Tương tác trực tiếp với thông tin: soạn thảo, nhập DL vào các forms

Giao diện đồ họa

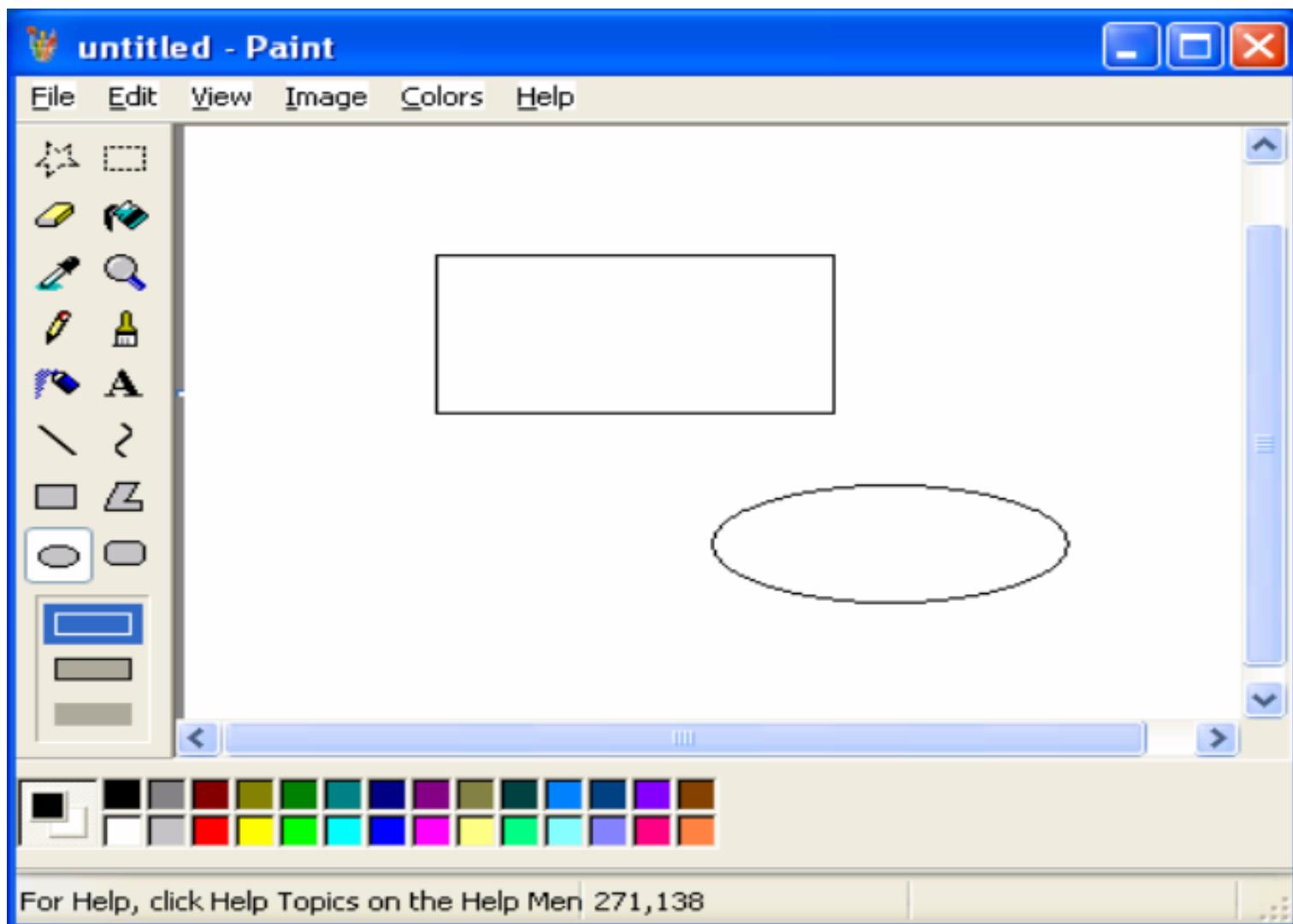
- Dễ học, dễ sử dụng
- Nhận được tức thời kết quả thao tác
- Cài đặt phức tạp, tốn tài nguyên hệ thống

Các hình thức tương tác

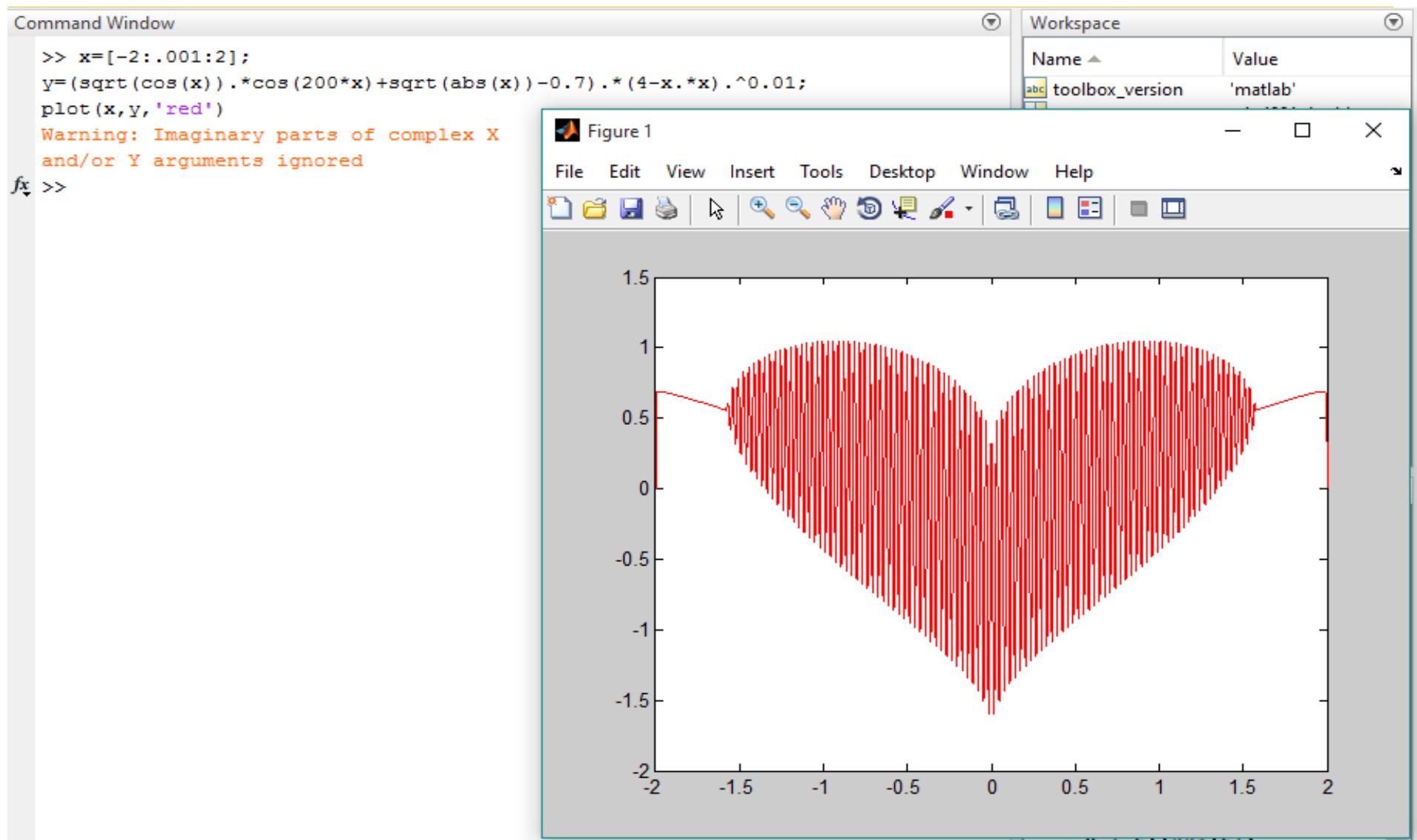
Tương tác trực tiếp và gián tiếp

- Tương tác trực tiếp:
 - Dễ nhận biết và thao tác
- Tương tác gián tiếp
 - Kém trực quan,
 - Thuận tiện khi lặp lại thao tác phức tạp
 - VD: chọn lệnh từ menu, giao diện dòng lệnh

VD: Tương tác trực tiếp



VD: Tương tác gián tiếp



Dạng thực đơn

- Không cần nhớ lệnh
- Tối thiểu hóa dùng bàn phím
- Tránh các lỗi như sai lệnh, sai tham số
- Dễ dàng tạo các trợ giúp theo ngữ cảnh

Dạng thực đơn

The screenshot shows a Microsoft Excel window titled "Microsoft Excel - danh sach lop.xls". A context menu is open over a table, with the "Format Cells..." option selected. The table has columns labeled "Số TT", "Họ Tên", "Lớp", "Số ĐT liên hệ", and "Điểm TB HK1". The data rows are numbered 1 to 10. The last row, containing the value "5.94", is highlighted with a blue selection bar.

Số TT	Họ Tên	Lớp	Số ĐT liên hệ	Điểm TB HK1
1	Nguyễn	A2	7680514	8.54
2	Nguyễn	A1	7895623	7.24
3	Hà Nam Đăng	A2	7789562	4.92
4	Nguyễn Bích Thủy	A1	8795623	5.19
5	Trần Thành Công	A2	8456932	6.21
6	Đặng Quang Hạnh	A2	8845213	4.37
7	Nguyễn Văn An	A1	9690565	8.70
8	Trần Công Thịnh	A1	9624678	5.21
9	Phạm Thị Hằng	A2	7860755	5.94
11				
12				
13				

Sheet1 / Sheet2 / Sheet3 /

Ready Sum=56.32 NUM

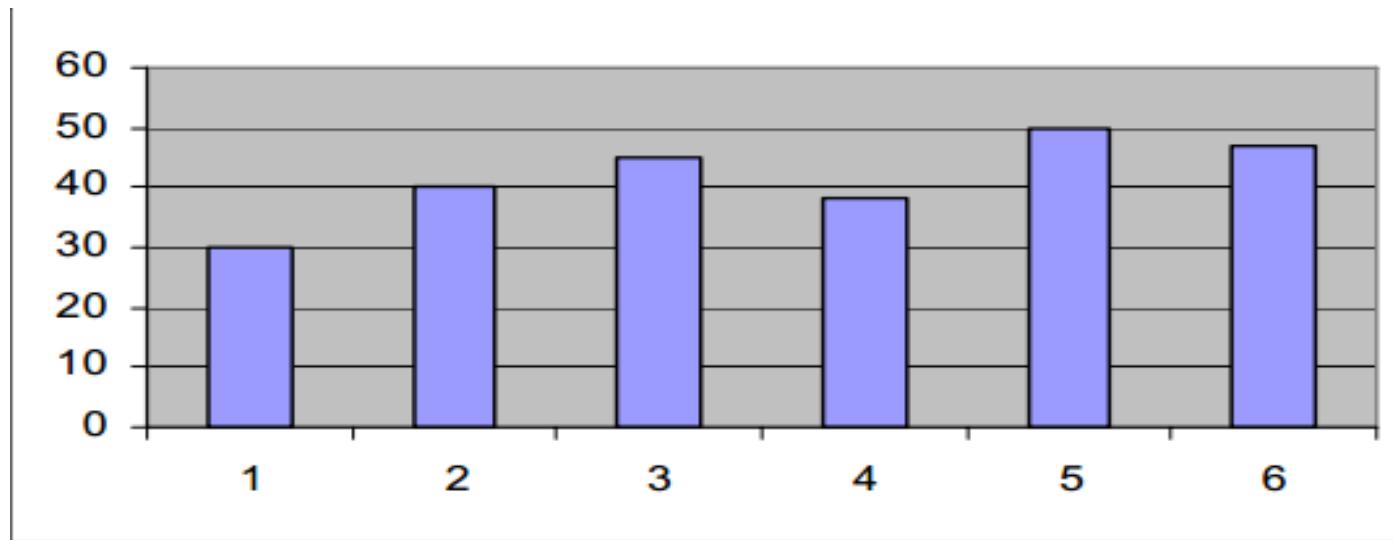
103

Một số vấn đề thiết kế giao diện

- Phương pháp hiển thị thông tin
- Thời gian phản hồi của hệ thống
- Cách thức xây dựng thông báo
- Các tiện ích trợ giúp

Hiển thị thông tin

- Hiển thị văn bản (text):
 - Chính xác, dễ cài đặt
- Hiển thị đồ họa (graphic)
 - Trực quan, dễ nhận dạng và mquh



Thời gian phản hồi của hệ thống

- Thời gian trung bình
 - Thời gian trung bình phản hồi với thao tác
 - Người dùng không thể đợi quá lâu (<3s)
 - Cần chứng tỏ hệ thống đang hoạt động
- Độ biến thiên thời gian
 - Chênh lệch không được lớn
 - Đều đặn là tốt nhất

Thông báo

- Phản hồi của hệ thống đối với thao tác
- Có nghĩa, dễ hiểu, các thông tin là hữu ích
 - Tránh đưa ra các số liệu
 - Định dạng thông báo phải nhất quán (vị trí, nội dung)
- Thông báo lỗi
 - Chính xác
 - Có tính xây dựng (nguyên nhân, cách khắc phục...)

Thông báo

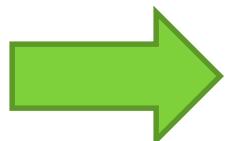
- Số lượng thông báo
 - Càng nhiều càng tốt = càng thân thiện
 - Đưa ra 1 lượng tối thiểu = im lặng là vàng
- Thời điểm và thứ tự đưa thông báo
 - Phù hợp với cách người dùng
- Yêu cầu phản hồi đối với thông báo

Tiện ích

- Cần có nhiều tiện ích khác nhau
- Tiện ích tích hợp: trợ giúp trực tuyến, theo ngữ cảnh (chú giải thao tác, giao diện)
- Các tài liệu trực tuyến: tra cứu chức năng hệ thống
- Các macro: tự động hóa thao tác

Tính kỹ nghệ

- Giao diện là thành phần dễ thay đổi
 - Thay đổi quy trình, phương thức thao tác
 - Thay đổi môi trường (phần cứng, HĐH)
 - Cải tiến (đẹp hơn, dễ dùng hơn...)
- Giao diện phải dễ sửa đổi
- Giao diện phải có tính khả chuyển



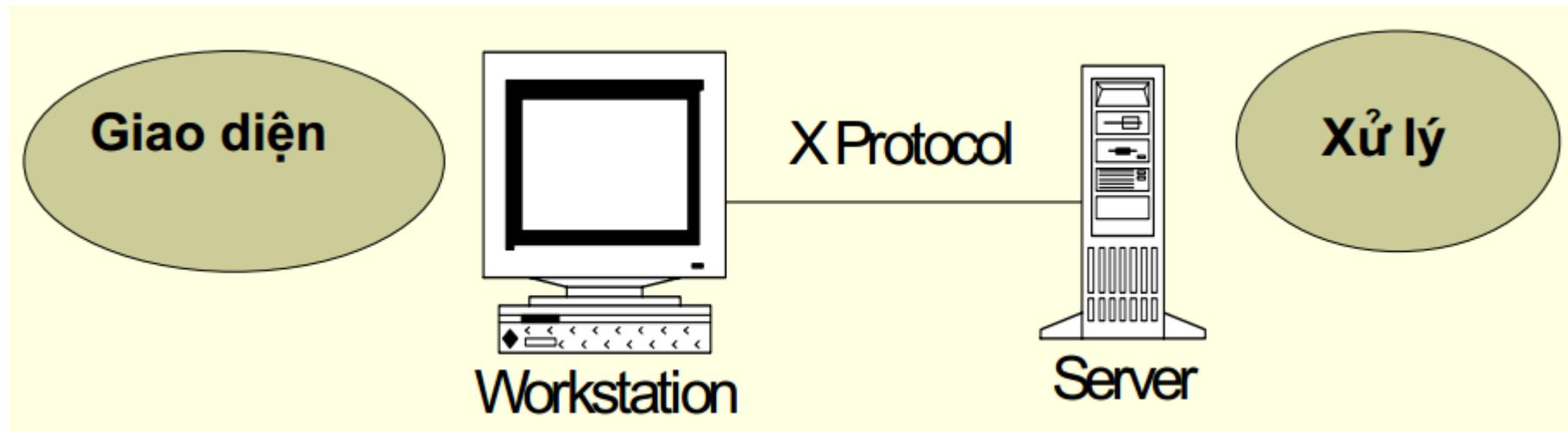
Giao diện nên độc lập với xử lý thông tin

Một số hình thức cài đặt giao diện

- Tích hợp: dòng lệnh, GUI truyền thống
 - Phát triển bằng cùng ngôn ngữ, cùng bộ công cụ
- Client/Server
 - Giao diện và xử lý là các chương trình độc lập
- X Windows (X protocol)
 - Giao diện và xử lý nằm cùng một chương trình
 - Hoạt động phân tán trên mạng

Một số hình thức cài đặt giao diện

- Web-based
 - Truy cập được từ mọi thiết bị có web browser
 - Không cần cài đặt thêm phần mềm vào client



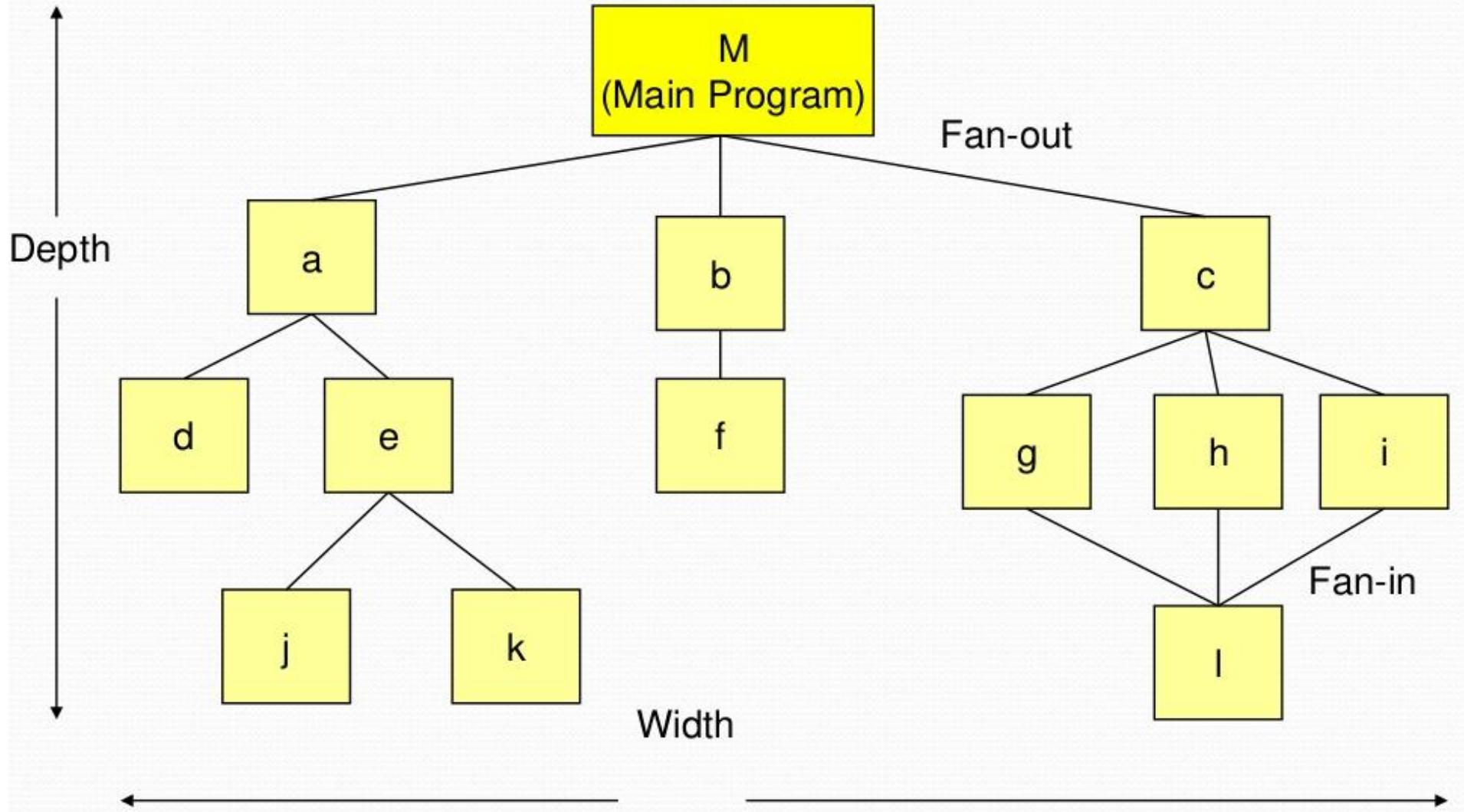
Bài tập B

- 1. Mô tả tiến trình thiết kế giao diện PM?
- 2. Các nguyên tắc thiết kế giao diện?
- 3. Có những loại giao diện nào? Cách tương tác?

Nội dung chính

- Khái niệm thiết kế phần mềm
- Các hoạt động thiết kế
- **Thiết kế hướng đối tượng**
 - Vấn đề tồn tại trong hướng cấu trúc
 - Khái niệm liên quan đến đối tượng
 - Ngôn ngữ UML
 - Phân tích HĐT
 - Thiết kế HĐT
 - Sử dụng mẫu thiết kế

Kiến trúc PM truyền thống



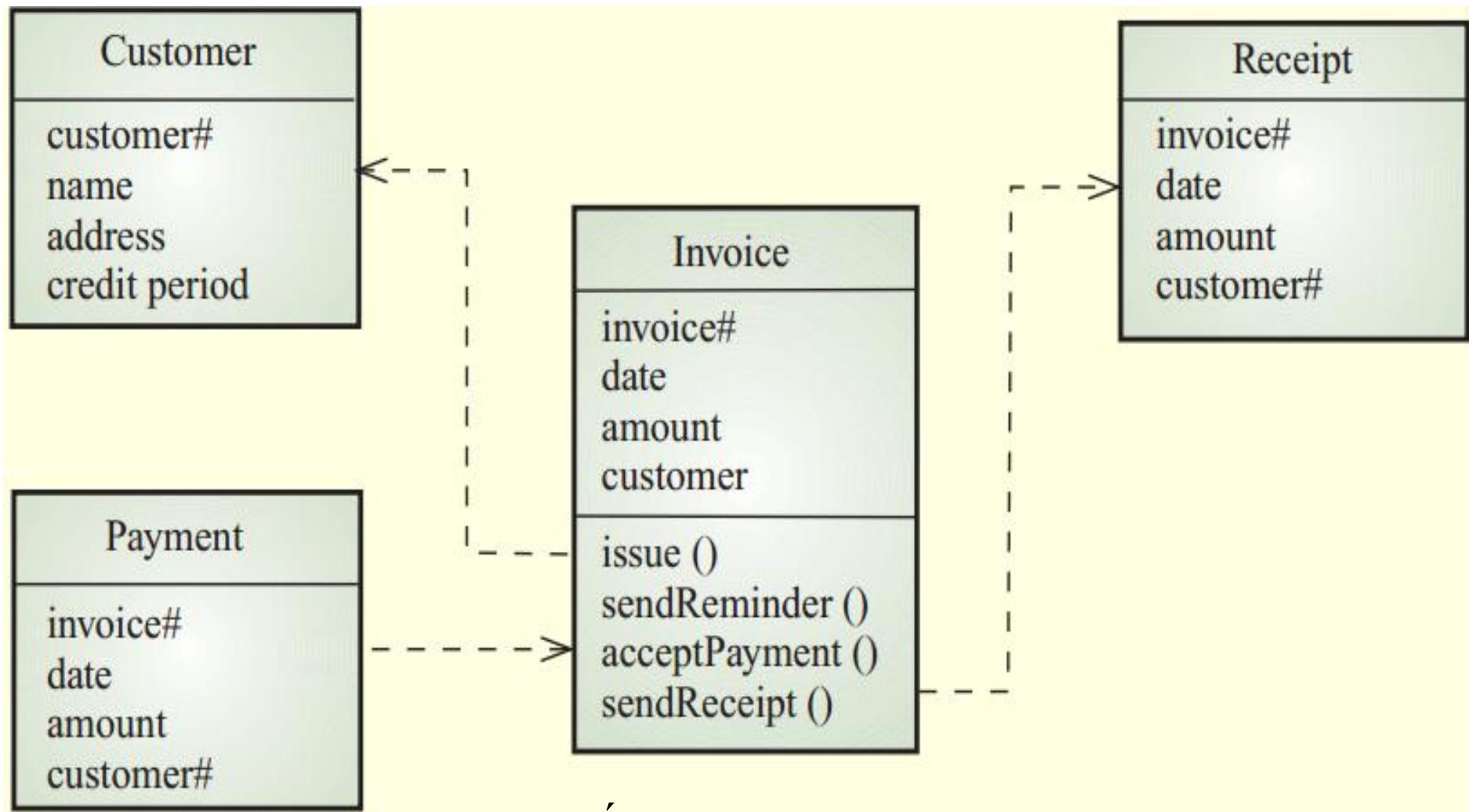
Vấn đề của thiết kế hướng thủ tục

- DL là chung cho cả hệ thống
 - Mọi thủ tục thao tác trên CSDL chung, đặc trưng cho trạng thái toàn hệ thống
 - Thao tác sai của 1 thủ tục lên DL → lan truyền lên phần khác
 - Sửa đổi 1 thủ tục → ảnh hưởng các thủ tục khác
- Thay đổi CTDL sẽ thay đổi toàn bộ hệ thống → DL cần được tổ chức tốt.
- Hệ thống lớn, phức tạp: bảo trì khó khăn

Thiết kế hướng đối tượng -OOD

- Là một cách tiếp cận khác, hiện đang trở nên phổ biến
- Nhìn nhận hệ thống theo các quan điểm:
 - Tập các đối tượng có tương tác với nhau
 - Mỗi đối tượng được bao gói: DL & hành vi
 - Cơ chế truyền thông điệp
 - Các đối tượng có thể kế thừa nhau

VD: Kiến trúc hướng đối tượng



Hệ thống xử lý đơn hàng

Ưu điểm của OOD

- Tái sử dụng
 - Có tính độc lập cao
 - Có tính kế thừa
- Tự nhiên, dễ hiểu: gần với thế giới thực
- Dễ bảo trì
 - Bao gói thông tin
 - Liên kết (tính ghép nối) lỏng lẻo

Nội dung của OOD

- Xác định các lớp đối tượng và đặc trưng của chúng
- Phân định vai trò và trách nhiệm của các lớp đối tượng trong hệ thống
- Thiết lập được sự tương tác của các lớp đối tượng để thực hiện chức năng của hệ thống PM đặt ra.

Các khái niệm của OOD

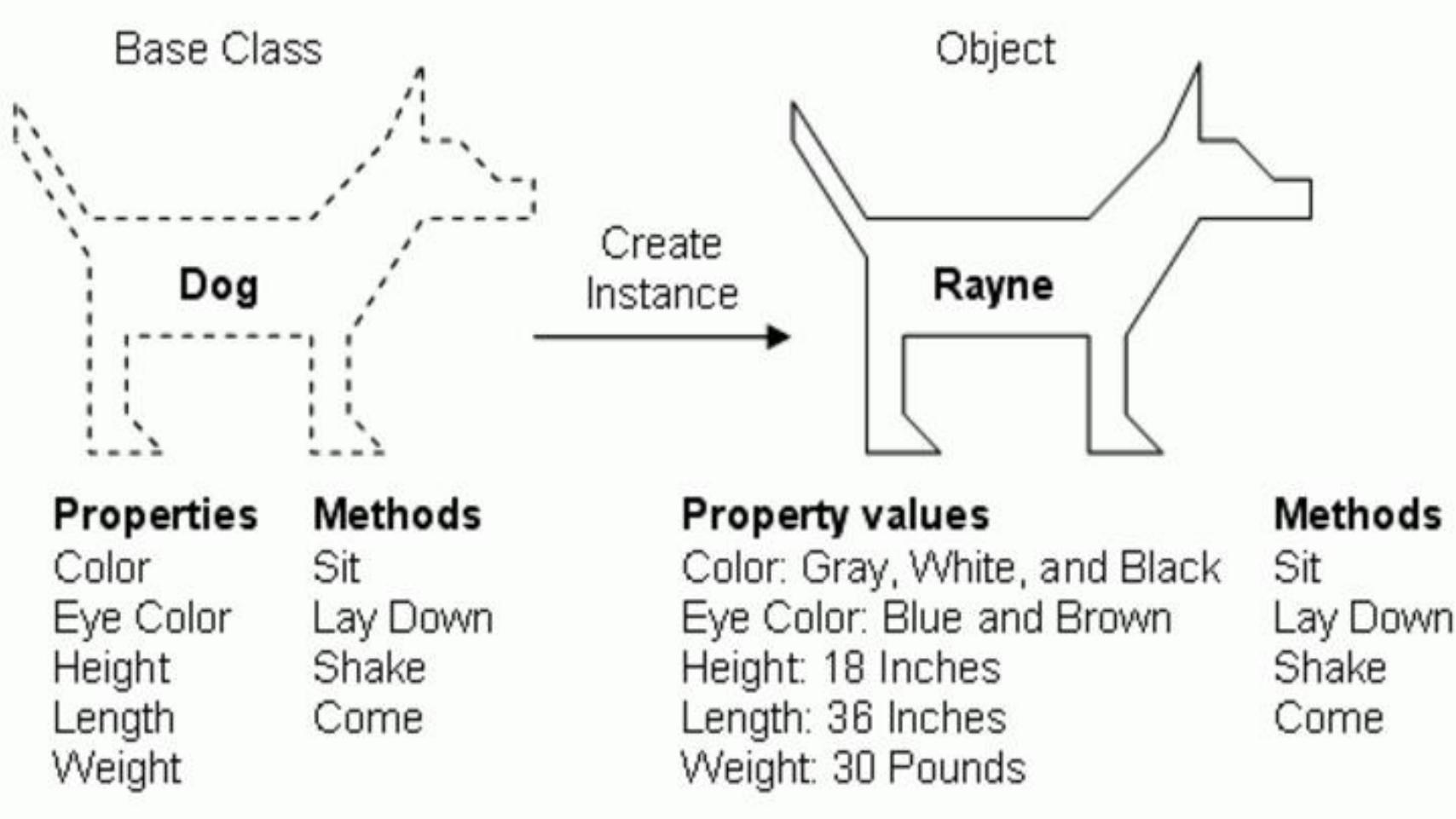
- **Đối tượng:**
 - Là các trừu tượng hóa thực thể của thế giới thực hoặc của 1 hệ thống
 - Bao gồm: **định danh, các thuộc tính và các phương thức**
 - Độc lập và đóng gói trạng thái thể hiện bằng giá trị các thuộc tính của nó ở 1 thời điểm
 - Cung cấp dịch vụ cho đối tượng khác hay yêu cầu các đối tượng khác thực hiện 1 dịch vụ

Các khái niệm của OOD

- **Lớp đối tượng:**

- Là khuôn mẫu để tạo ra tập đối tượng có các đặc trưng chung
- Có thể kế thừa thuộc tính và dịch vụ từ lớp khác.
- Được xác định bằng: Tên, các thuộc tính, các phương thức

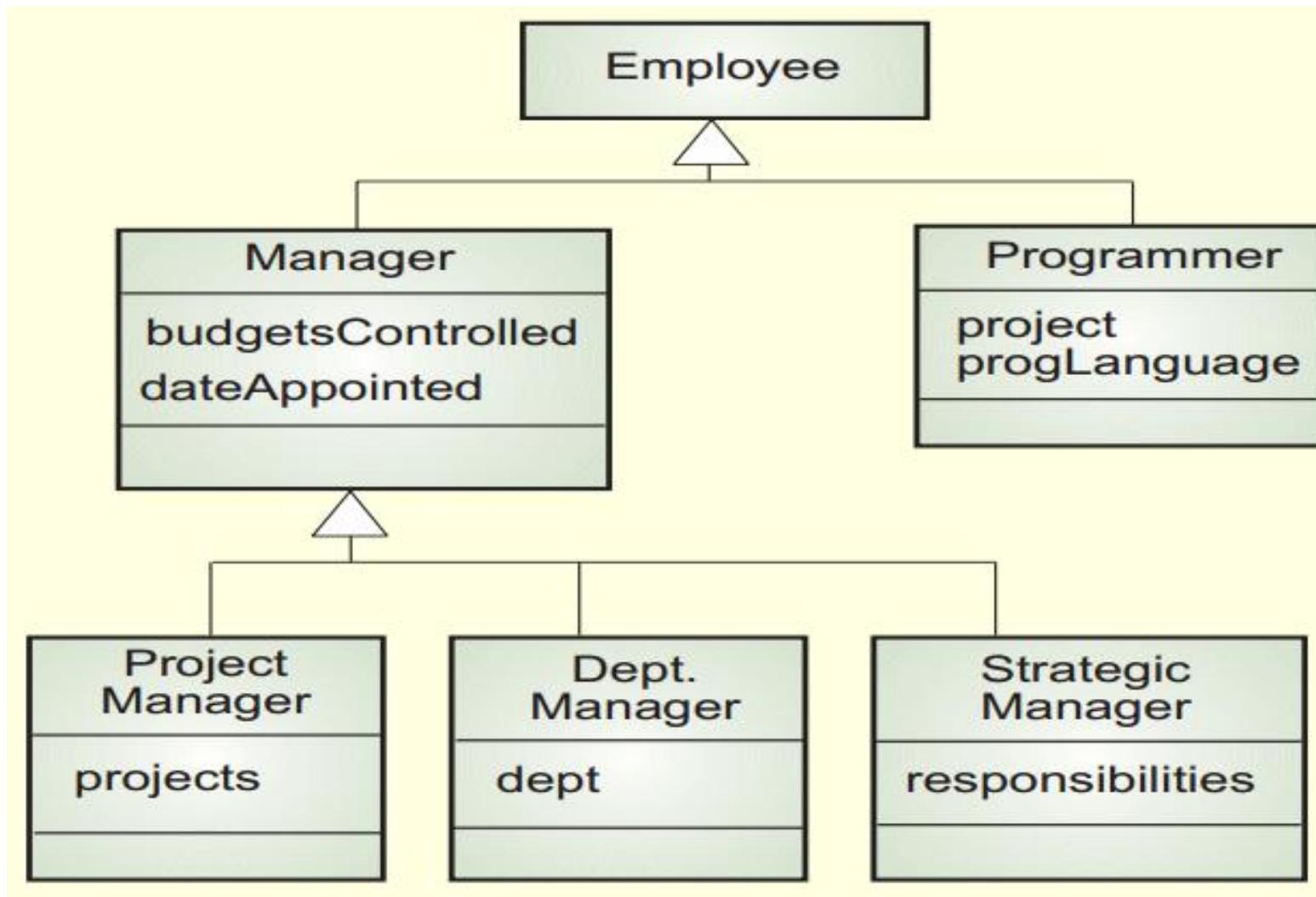
Các khái niệm của OOD



Tổng quát hóa và kế thừa

- Lớp cha có thể là tổng quát hóa của 1 số lớp con
- Lớp con kế thừa các thuộc tính và phương thức của lớp cha và có thể thêm/thay đổi phương thức, thuộc tính
- Sử dụng kế thừa giúp ta mô tả lớp con chỉ gồm các đặc trưng khác lớp cha

Ví dụ cây kế thừa



Ưu điểm của kế thừa

- Là cơ chế trừu tượng để phân loại các lớp (thực thể)
- Tái sử dụng cả ở mức thiết kế và mức lập trình
 - Tái sử dụng CSDL
 - Tái sử dụng phương thức:
 - Giao diện
 - Cài đặt (mã)
- lược đồ kế thừa là nguồn thông tin mang tính tổ chức về bài toán

Tương tác giữa các đối tượng

- Sử dụng cơ chế truyền thông điệp
- Thông điệp
 - Tên dịch vụ được yêu cầu
 - Thông tin dùng để thực hiện dịch vụ
- Các loại đối tượng:
 - Actor: chỉ gửi thông điệp
 - Agent: gửi và nhận thông điệp
 - Server: chỉ nhận thông điệp
- Thực tế, thông điệp được cài đặt bằng lời gọi hàm
 - Tên = tên hàm
 - Thông tin = danh sách tham số

Nội dung chính

- Khái niệm thiết kế phần mềm
- Các hoạt động thiết kế
- **Thiết kế hướng đối tượng**
 - Vấn đề tồn tại trong hướng cấu trúc
 - Khái niệm liên quan đến đối tượng
 - Ngôn ngữ UML
 - Phân tích HĐT
 - Thiết kế HĐT
 - Sử dụng mẫu thiết kế

Mô hình hóa

- Mô hình là kết quả của phân tích và thiết kế.
- Là sự đơn giản hóa thực tế, cho phép hiểu rõ hơn hệ thống cần phát triển.
- Mô hình cho phép:
 - Cái nhìn trực quan về hệ thống đang có hoặc hướng tới
 - Kiểm chứng hệ thống bởi khách hàng
 - Cung cấp những chỉ dẫn để xây dựng hệ thống
 - Tài liệu hóa hệ thống

Mô hình hóa

- Tồn tại nhiều cách mô hình hóa một hệ thống
 - Mô hình phù hợp → việc giải bài toán dễ hơn
- Có nhiều mức chính xác của mô hình
 - Mô hình trừu tượng -> làm mịn -> mô hình chi tiết
- Không có mô hình nào là đầy đủ
 - Cần tiếp cận (hiểu) hệ thống thông qua nhiều mô hình khác nhau
- Mô hình tốt phải là mô hình phù hợp với thế giới thực

Mô hình hóa HĐT

- **Tăng tính độc lập** của mô hình với các chức năng yêu cầu
- Dễ dàng hơn trong việc thay đổi hoặc thêm bớt các chức năng
- Gần với thế giới thực

Ví dụ về mô hình hóa



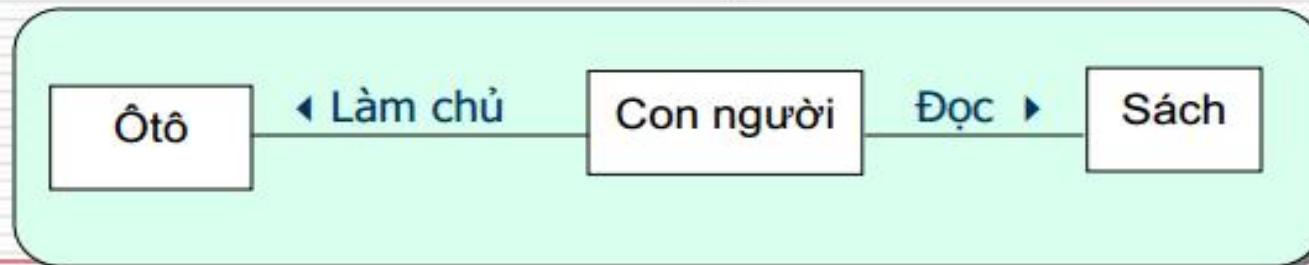
Thế giới thực



Mô hình: Quả địa cầu học sinh



Thế giới thực



Mô hình

Ngôn ngữ mô hình hóa thống nhất

Unified Modeling Language

- Là một ngôn ngữ mô hình hóa HĐT
- Được thừa nhận như một chuẩn mặc định của lĩnh vực CNTT
 - Real-time UML, Agent-UML
- UML là ngôn ngữ đồ họa cho phép đặc tả, xây dựng, lập tài liệu và hiển thị trực quan hầu hết các artifacts của hệ thống mà ta có ý định tin học hóa.

UML cho phép

- **Hiển thị**
 - Thông qua các ký pháp đồ họa có ngũ nghĩa xác định
- **Đặc tả**
 - Một cách chính xác và toàn diện
- **Xây dựng**
 - Các lớp, các quan hệ có thể được ánh xạ vào chương trình thực
- **Làm tài liệu**
Các lược đồ, các ghi chú, các ràng buộc,...

Mục đích của UML

- Biểu diễn toàn bộ hệ thống
- Tạo sự liên kết giữa các khái niệm (concepts) về hệ thống và các artifacts thực hiện được
- Vừa trực quan (hướng người dùng) vừa hỗ trợ khả năng tự động hóa (máy)
- Một ngôn ngữ chung:
 - Sử dụng được với nhiều phương pháp khác nhau
 - Đáp ứng tất cả các giai đoạn trong phát triển phần mềm

Phạm vi ứng dụng của UML

- Hệ thống thông tin doanh nghiệp
- Ngân hàng và các dịch vụ tài chính
- Viễn thông
- Giao thông
- Quân sự và hàng không (hệ nhúng)
- Khoa học
- Các ứng dụng phân tán trên web

UML Partners

Doanh nghiệp	Doanh nghiệp
Rational Software Corp	ObjecTime
Hewlett-Packard	Oracle
I-Logix	Platinum Technology
IBM	Taskon
ICON Computing	Texas Instruments
Intelicorp	Sterling Software
MCI Systemhouse	Unisys
Microsoft	

Ngôn ngữ mô hình hóa thống nhất

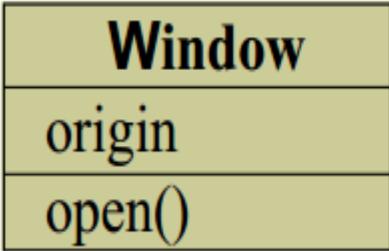
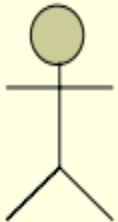
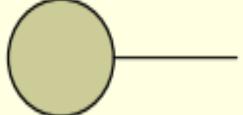
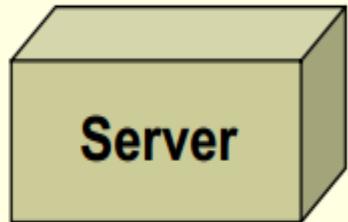
Unified Modeling Language

Gồm 3 khối cơ bản:

- **Các sự vật**
 - Các sự vật cấu trúc (Structural)
 - Các sự vật hành vi (Behavioral)
 - Các sự vật nhóm gộp (Grouping)
 - Các sự vật giải thích (Annotational)
- **Các quan hệ (Relationships)**
- **Các lược đồ (Diagrams)**

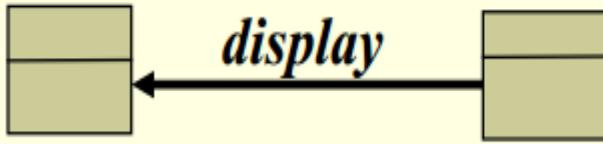
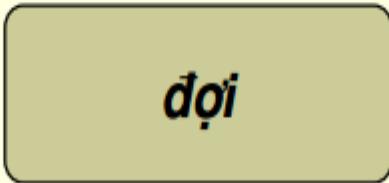
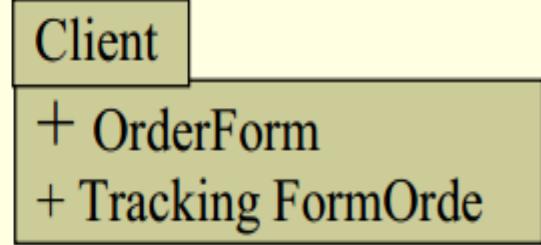
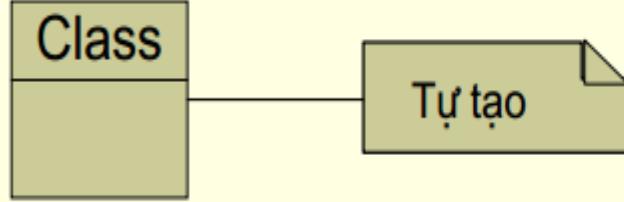
Ký pháp trong UML

- Các sự vật cấu trúc (structural)

Class – lớp	use ase – ca sử dụng	Collaboration sự cộng tác	
			
Actor	Interface	Component-thành phần	Node - nút
			

Ký pháp trong UML

- Các sự vật hành vi – nhóm gộp – giải thích

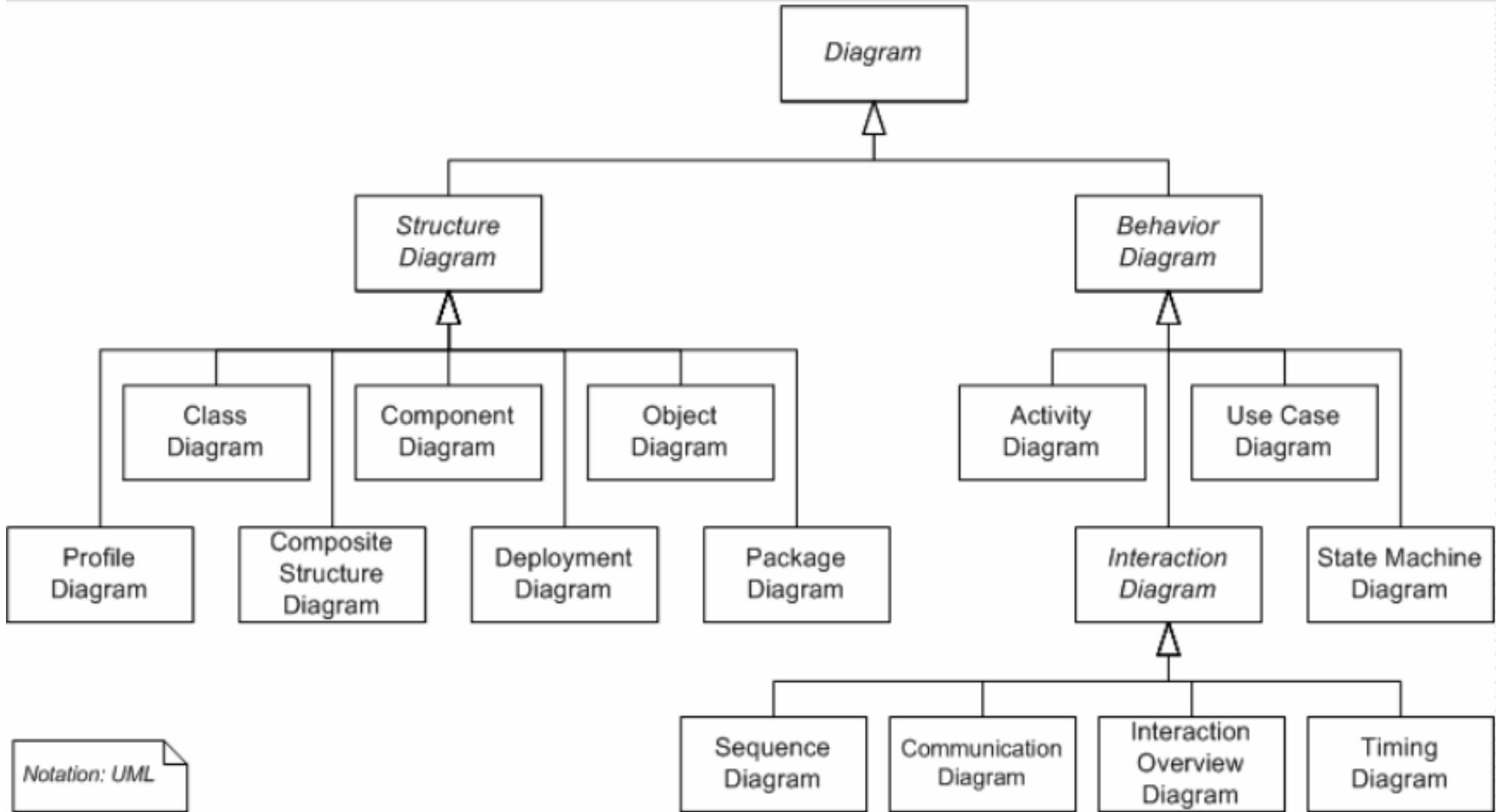
interaction	state machine	package
		
Note		
		

Ký pháp trong UML

- Các mối quan hệ (Relationships)

dependence	assosiation	generalization
	 0..1 1..*	
realization	aggregation	composite

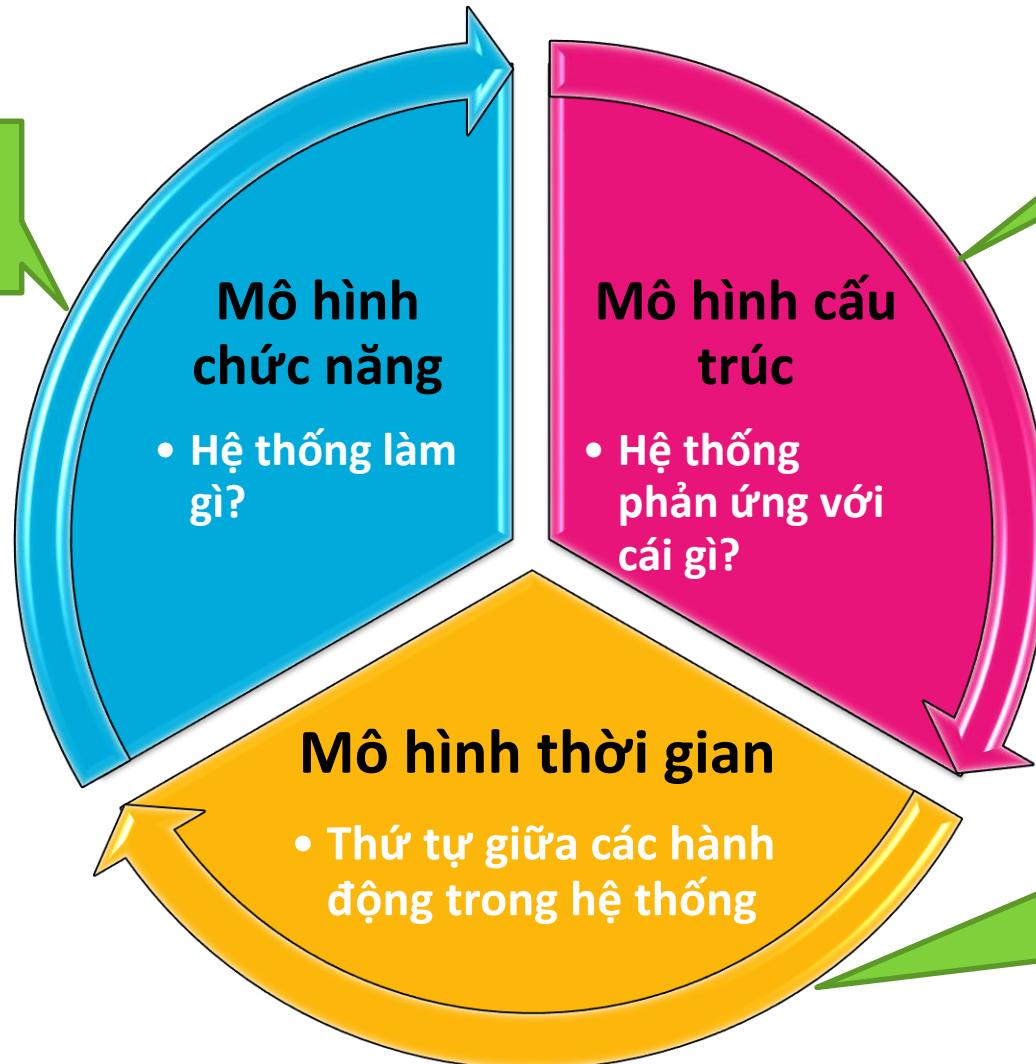
Các lược đồ trong UML



Notation: UML

Phân loại lược đồ

Lược đồ ca
sử dụng

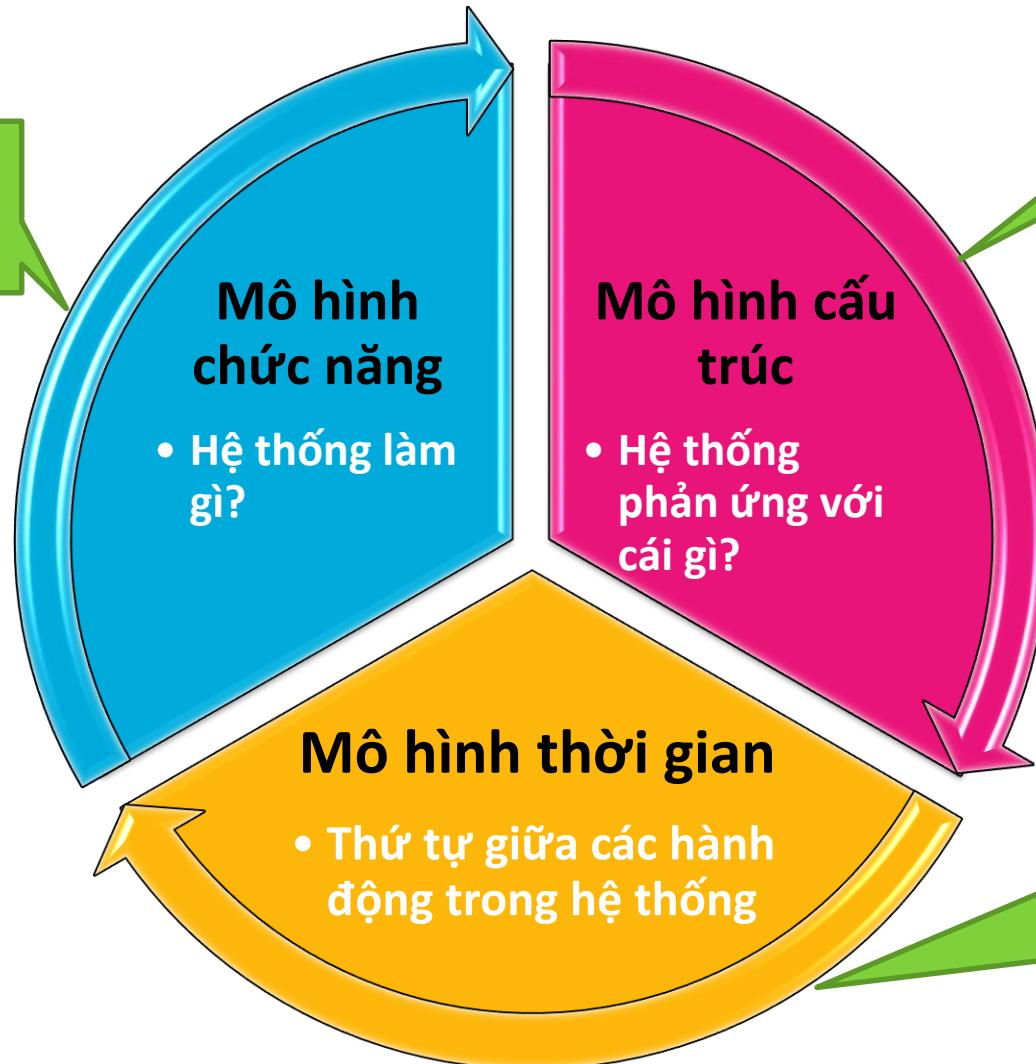


Lược đồ lớp và
đối tượng

Lược đồ tuần tự
Lược đồ tương tác
Lược đồ trạng thái
Lược đồ hoạt động

Phân loại lược đồ

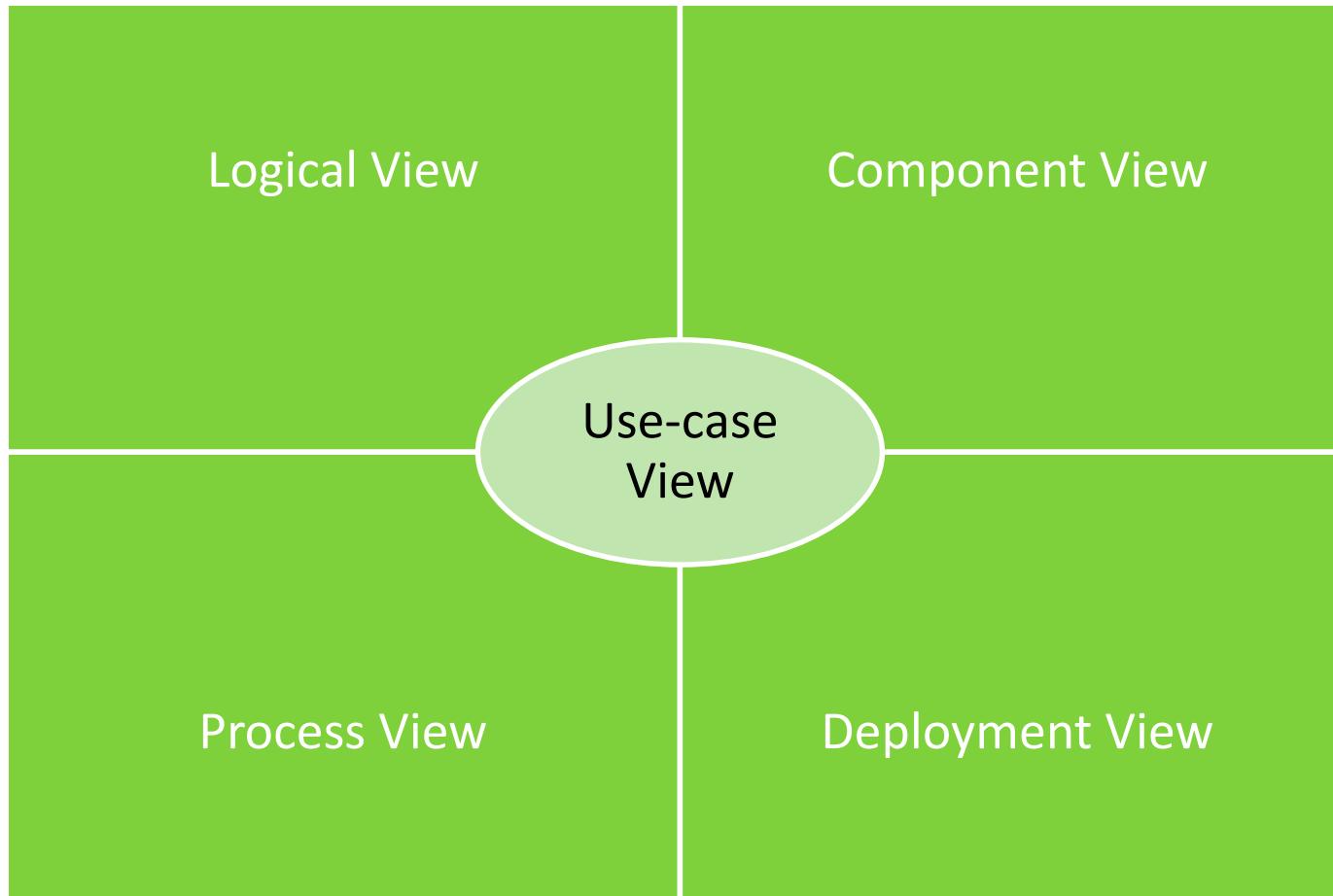
Lược đồ ca
sử dụng



Lược đồ lớp và
đối tượng

Lược đồ tuần tự
Lược đồ tương tác
Lược đồ trạng thái
Lược đồ hoạt động

4 + 1 View Model



Use-case View

- Nhìn hệ thống bởi người dùng cuối
 - Tác nhân, chức năng, kịch bản
- Phân loại được các hành vi
 - Chức năng, độ ưu tiên
- Chỉ ra các ràng buộc giữa các hành vi\
 - Sử dụng, kế thừa

Logical View

- Cung cấp cái nhìn logic về chức năng, hành vi của hệ thống
- Nhìn nhận các đối tượng
 - Các lớp và đối tượng
 - Các gói
 - Các quan hệ: Kết nối, trừu tượng, đa hình, đồng nhất
- Nhìn nhận các trạng thái/luồng công việc
- Nhìn nhận về cách thức tương tác
 - Các kịch bản của các ca sử dụng

Component View

- Phân rã theo module
- Nhóm thành các module bằng các gói
- Tổ chức thành các hệ thống con để:
 - Tăng độ chắc chắn
 - Giảm sự ghép nối và nhìn thấy
- Giúp cho:
 - Dễ dàng phát triển
 - Nhìn thấy khả năng tái sử dụng
 - Quản lý cấu hình

Deployment View

- Phân rã hệ thống thành các nút triển khai
 - Vai trò của nút
 - Liên quan giữa các nút
- Cung cấp các thông tin:
 - Hiệu năng, tính sẵn sàng của hệ thống
 - Cách thức cài đặt, bảo trì

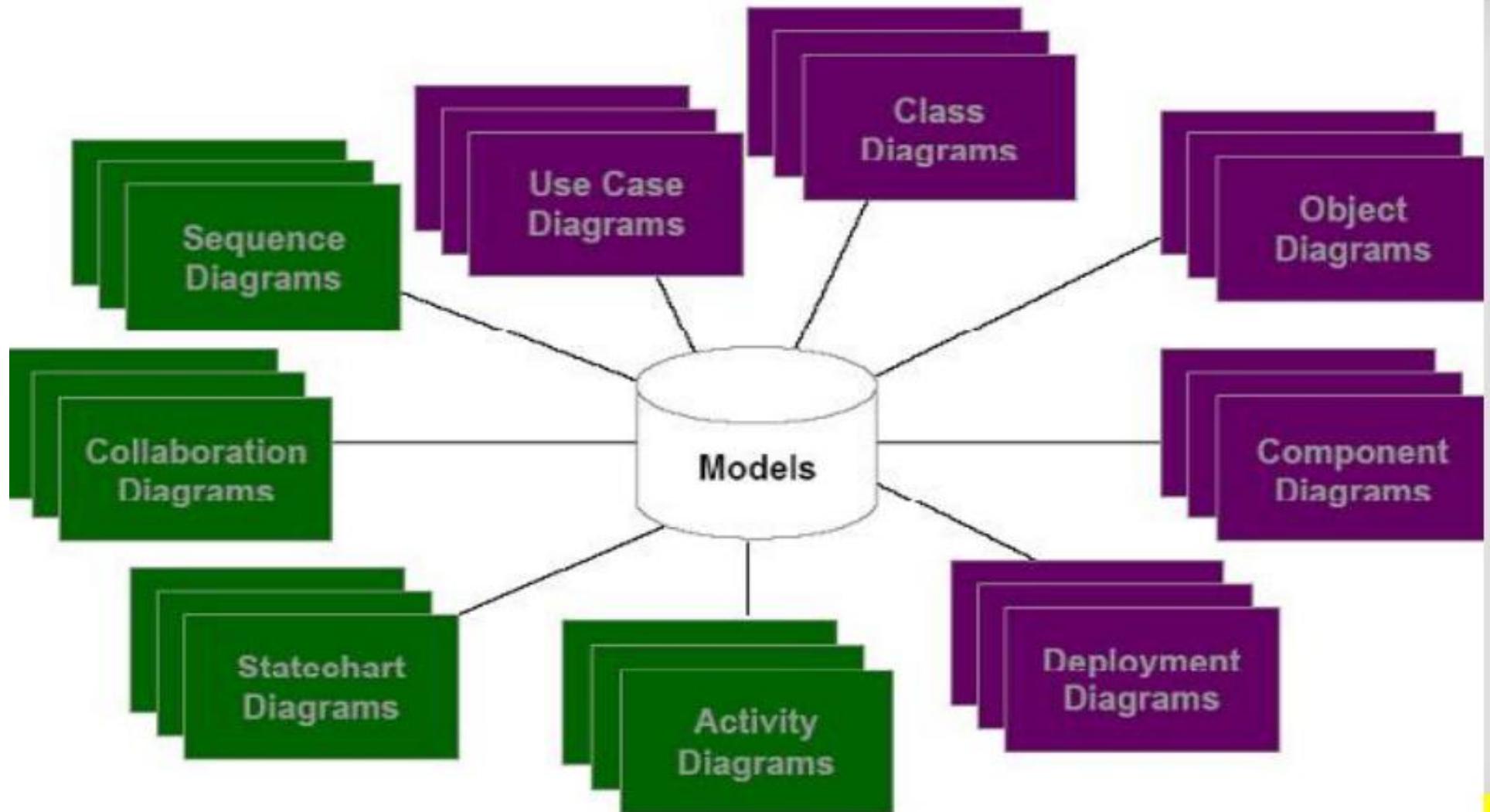
Process View

- Phân rã dựa trên nhiệm vụ và tiến trình
 - Phân loại được các nhóm tiến trình
- Cung cấp các thông tin động về hoạt động của hệ thống:
 - Thứ tự thực hiện, cách thức tương tác, ràng buộc thứ tự
 - Phân tán/song song
 - Khả năng tính tích hợp, hiệu năng,...

Đặc tả PM thông qua nhiều mô hình

- PM được đặc tả tương đối đầy đủ thông qua nhiều mô hình. Mỗi mô hình cho thấy toàn thể PM nhưng chỉ theo 1 góc nhìn xác định
- Mỗi mô hình được cấu thành từ nhiều lược đồ. Mỗi lược đồ cho thấy 1 bộ phận PM theo mô hình liên quan.

Đặc tả PM thông qua nhiều mô hình



Đặc tả PM thông qua nhiều mô hình

UML cho phép đặc tả 9 loại lược đồ phổ dụng được miêu tả theo 2 góc nhìn khác nhau về PM:

- **Góc nhìn tĩnh:** use-case diagram, class diagram, component diagram, object diagram, deployment diagram.
- **Góc nhìn động:** sequence diagram, collaboration diagram, statechart diagram, activity diagram.

UML – Lược đồ use-case

- Là artifact được xây dựng trong workflow nắm bắt yêu cầu PM.
- Được tạo phân tích viên, chuyên gia lĩnh vực liên quan và khách hàng.
- Miêu tả góc nhìn từ ngoài vào hệ thống PM, cho thấy 1 số chức năng của hệ thống PM (use-case) và ai (actor) sẽ thực hiện các use-case.
- Lược đồ use-case gồm các phần tử: actor, use-case, note, mqb giữa chúng

UML – Lược đồ lớp

- Là artifact được xây dựng trong workflow phân tích hay thiết kế.
 - Trong workflow phân tích: thường chỉ chứa thông tin sơ lược về các lớp chức năng.
 - Trong workflow thiết kế: thường chứa đầy đủ các thông tin chi tiết cũng như các mqh với những thành phần khác.
- Do phân tích viên, kỹ sư thiết kế, người hiện thực cùng nhau xây dựng và duy trì.

UML – Lược đồ lớp

- Miêu tả góc nhìn tổ chức bên trong hệ thống PM, cho thấy 1 số lớp chức năng hợp tác với nhau ntn để thực hiện use-case hay bộ phận use-case nào đó.
- Lược đồ lớp gồm các phần tử: actor, class, interface, note, mquh giữa chúng.

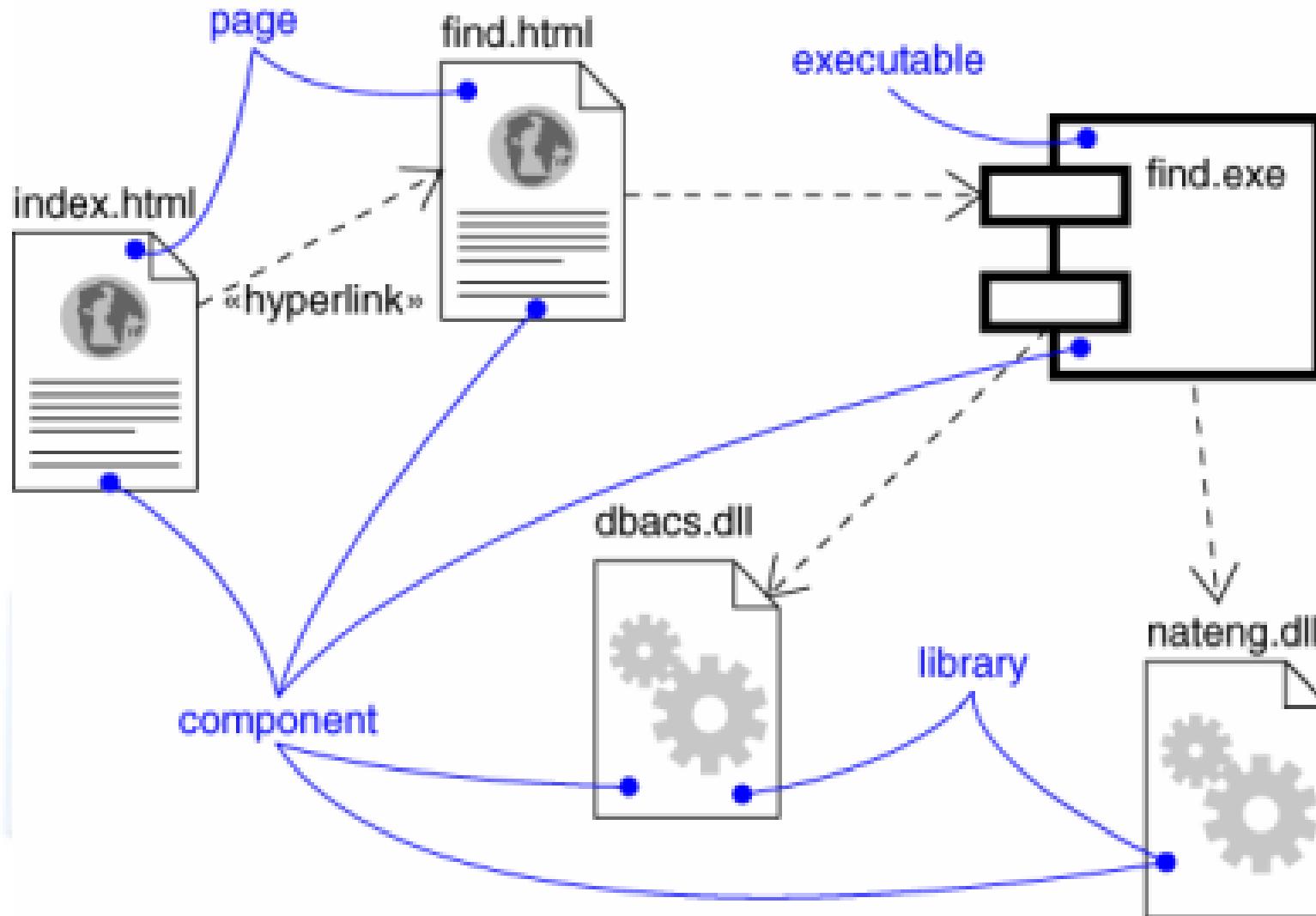
UML – Lược đồ đối tượng

- Là artifact được xây dựng trong workflow phân tích hay thiết kế.
- Do phân tích viên, kỹ sư thiết kế, người lập trình hợp tác xây dựng và duy trì.
- Miêu tả góc nhìn bên trong hệ thống PM, tại 1 thời điểm cụ thể trong lúc PM hoạt động nó cho thấy cụ thể các đối tượng, trạng thái, mquh giữa chúng.
- Lược đồ đối tượng gồm các phần tử: actor, đối tượng, note, mquh giữa chúng.

UML – Lược đồ thành phần

- Là artifact được xây dựng trong workflow hiện thực.
- Do kiến trúc sư, lập trình viên hợp tác xây dựng.
- Miêu tả góc nhìn tổ chức bên trong hệ thống PM, cho thấy các thành phần thực tế được xây dựng, công nghệ xây dựng và mqr của chúng để thực hiện 1 use-case hay 1 phần use-case nào đó.
- Lược đồ thành phần gồm các phần tử: component, note và mqr giữa chúng.

UML – Lược đồ thành phần



UML – Lược đồ triển khai

- Là artifact được xây dựng trong workflow thiết kế.
- ~~Đo~~ kiến trúc sư, kỹ sư hệ thống và kỹ sư mạng hợp tác xây dựng.
- Nó miêu tả cách tổ chức, kết nối các thành phần phần cứng để vận hành hệ thống PM tương ứng.
- Lược đồ triển khai gồm các phần tử: từng thiết bị, mạng kết nối chúng, chức năng và tính chất của chúng.

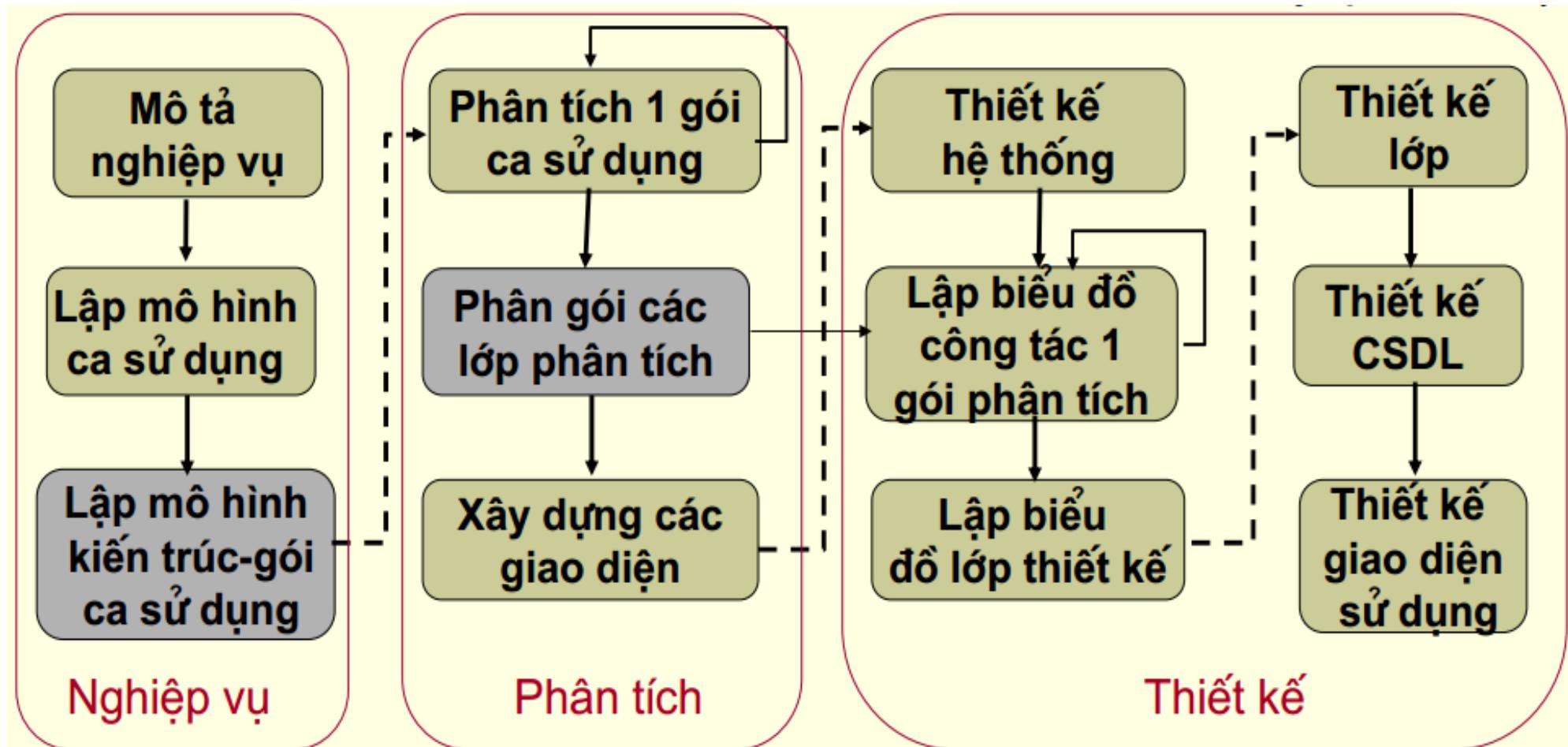
Nội dung chính

- Khái niệm thiết kế phần mềm
- Các hoạt động thiết kế
- **Thiết kế hướng đối tượng**
 - Vấn đề tồn tại trong hướng cấu trúc
 - Khái niệm liên quan đến đối tượng
 - Ngôn ngữ UML
 - Phân tích HĐT
 - Thiết kế HĐT
 - Sử dụng mẫu thiết kế

Phân tích – thiết kế HĐT

Mô hình phân tích	Mô hình thiết kế
<ul style="list-style-type: none">• Mô hình nghiệp vụ<ul style="list-style-type: none">• Mô hình miền• Lược đồ hoạt động• Mô hình ca sử dụng• Mô hình lớp phân tích• Mô hình gói lớp	<ul style="list-style-type: none">• Mô hình cấu trúc gói• Mô hình cộng tác• Mô hình lớp• Đặc tả lớp, giao diện

Tiến trình phân tích – thiết kế HĐT



Tiến trình phân tích và thiết kế hướng đối tượng

Phân tích HĐT (1)

- Mô tả nghiệp vụ
 - Bằng ngôn ngữ tự nhiên
 - Bằng các lược đồ hoạt động
- Xây dựng mô hình nghiệp vụ
 - Mô hình miền lĩnh vực
 - Mô hình ca sử dụng
- Phân tích xác định cấu trúc (**khởi thảo**)
 - Làm mịn mô hình ca sử dụng
 - Xác định các gói ca sử dụng, giao diện

Phân tích HĐT (2)

- Phân tích 1 ca sử dụng
 - Tìm các lớp phân tích
 - Xác định liên kết giữa các lớp
- Phân gói lại các lớp phân tích (**tăng cường kiến trúc**)
 - Tác các lớp dịch vụ và ứng dụng
 - Phân gói các lớp phân tích theo tầng
- Xác định và mô tả các giao diện
 - Xác định giao diện giữa các gói
 - Xác định liên kết giữa các gói

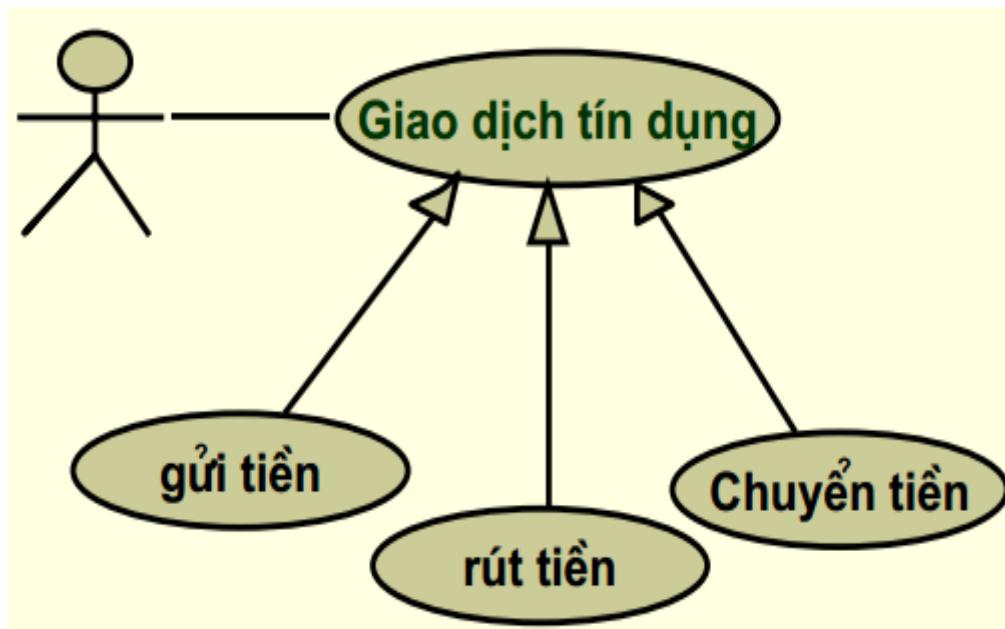
Thiết kế HĐT

- Thiết kế lược đồ tương tác mỗi gói
 - Xác định lại các lớp
 - Xây dựng lược đồ tương tác
- Phát triển lược đồ lớp thiết kế
 - Chuyển lược đồ công tác sang lược đồ lớp
 - Hoàn thiện các quan hệ công tác
- Thiết kế các lớp
 - Thiết kế các thuộc tính
 - Thiết kế các phương thức
- Thiết kế CSDL
- Thiết kế giao diện người dùng

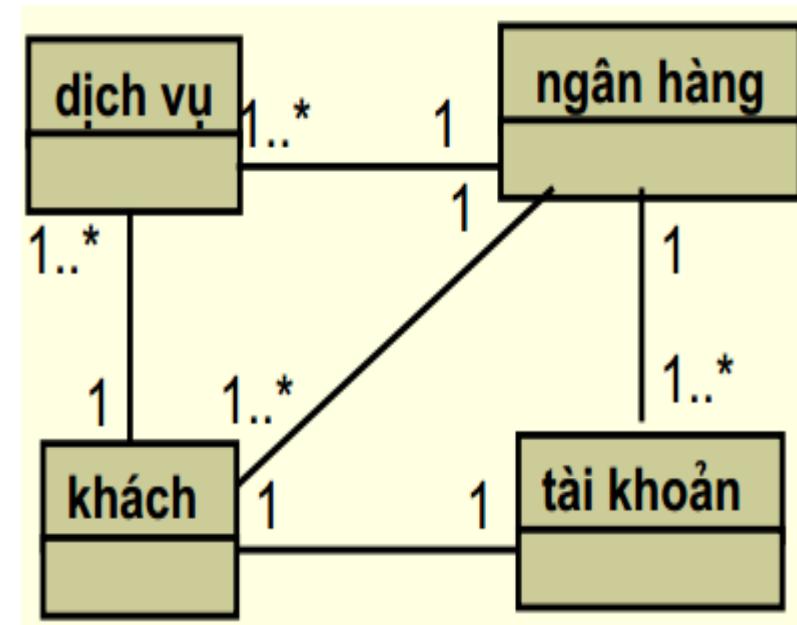
VD Phân tích HĐT

Bài toán: **Giao dịch tín dụng sử dụng ATM**

1. Mô hình nghiệp vụ



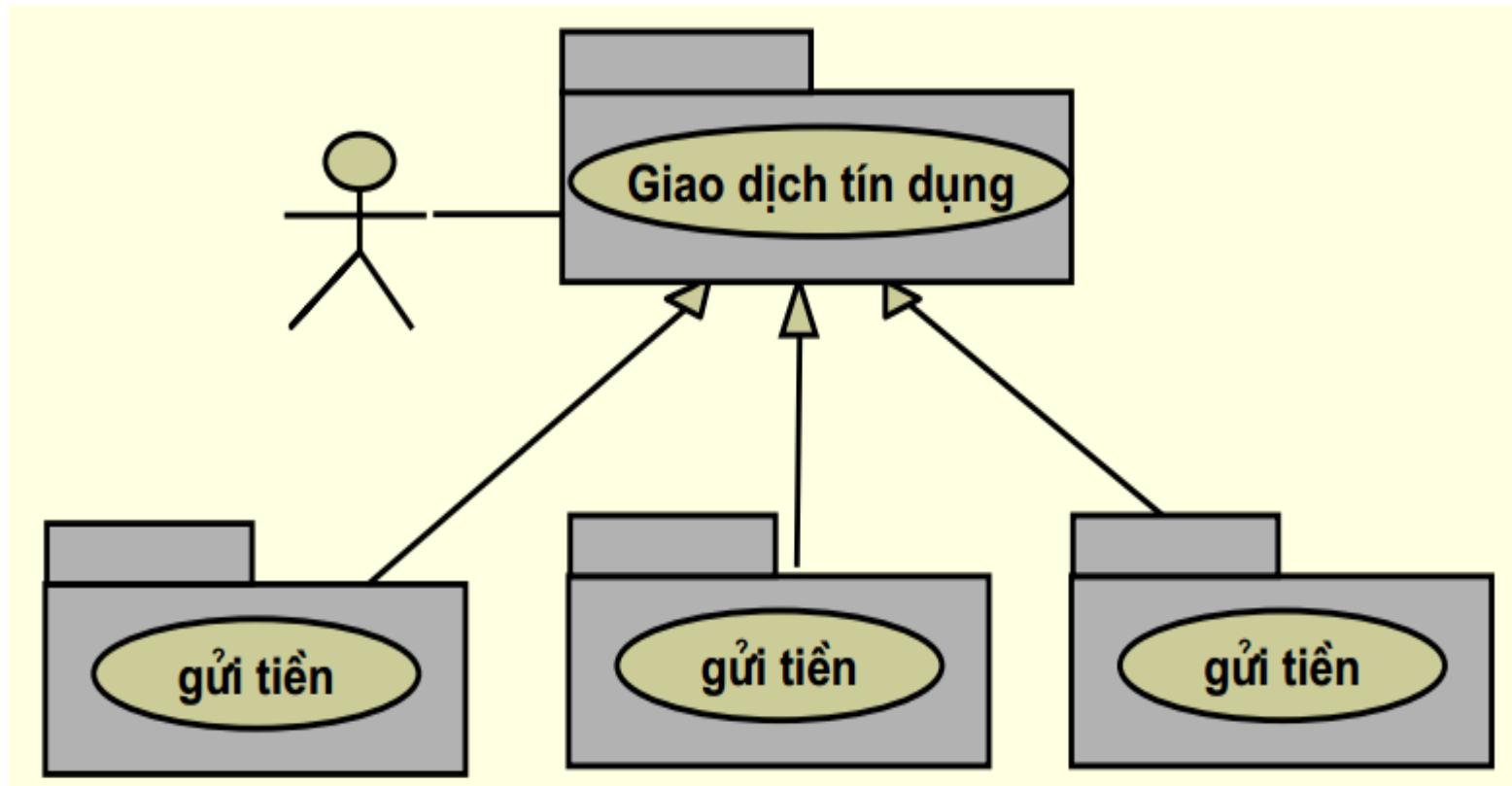
Lược đồ ca sử dụng



Mô hình miền

VD Phân tích HĐT

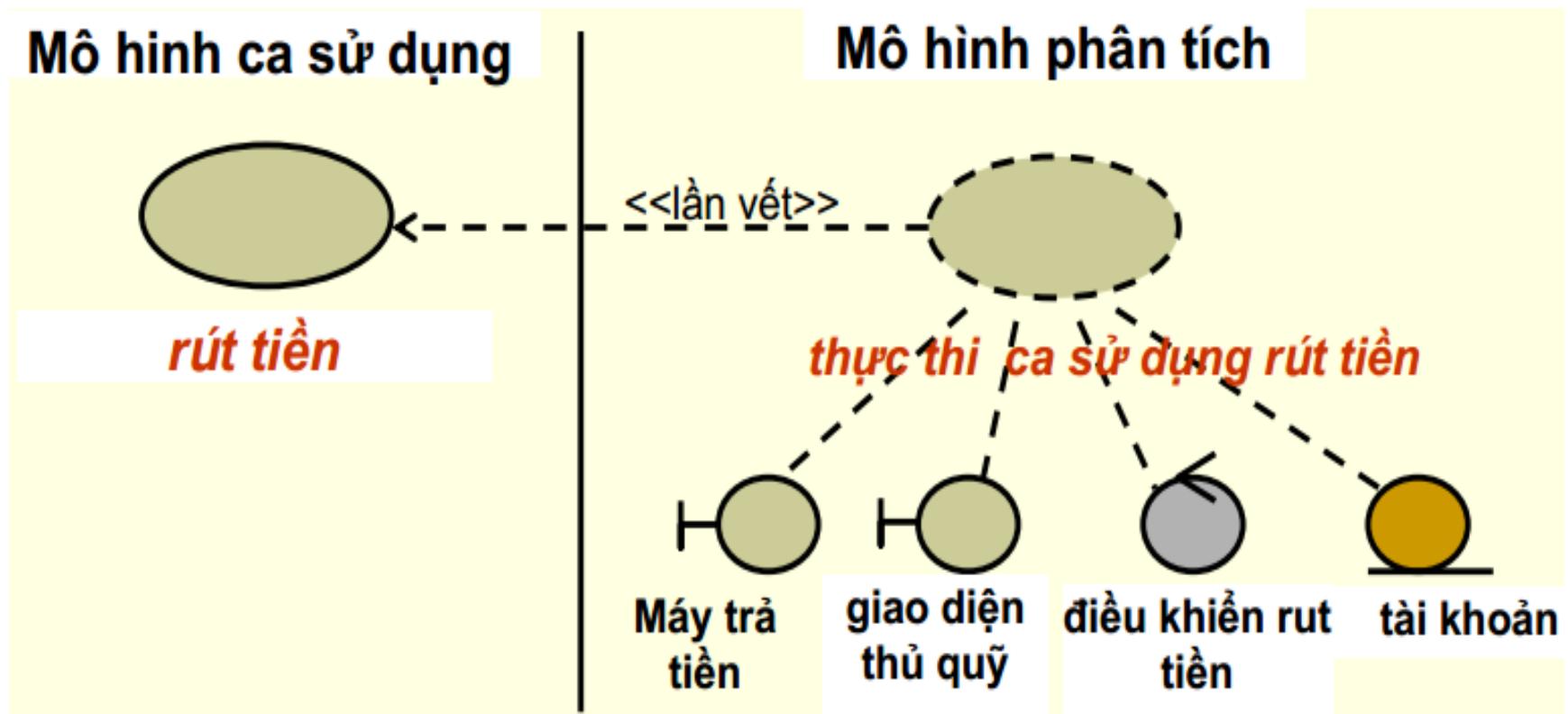
- Xác định gói các ca sử dụng



Lược đồ gói ca sử dụng

VD Phân tích HĐT

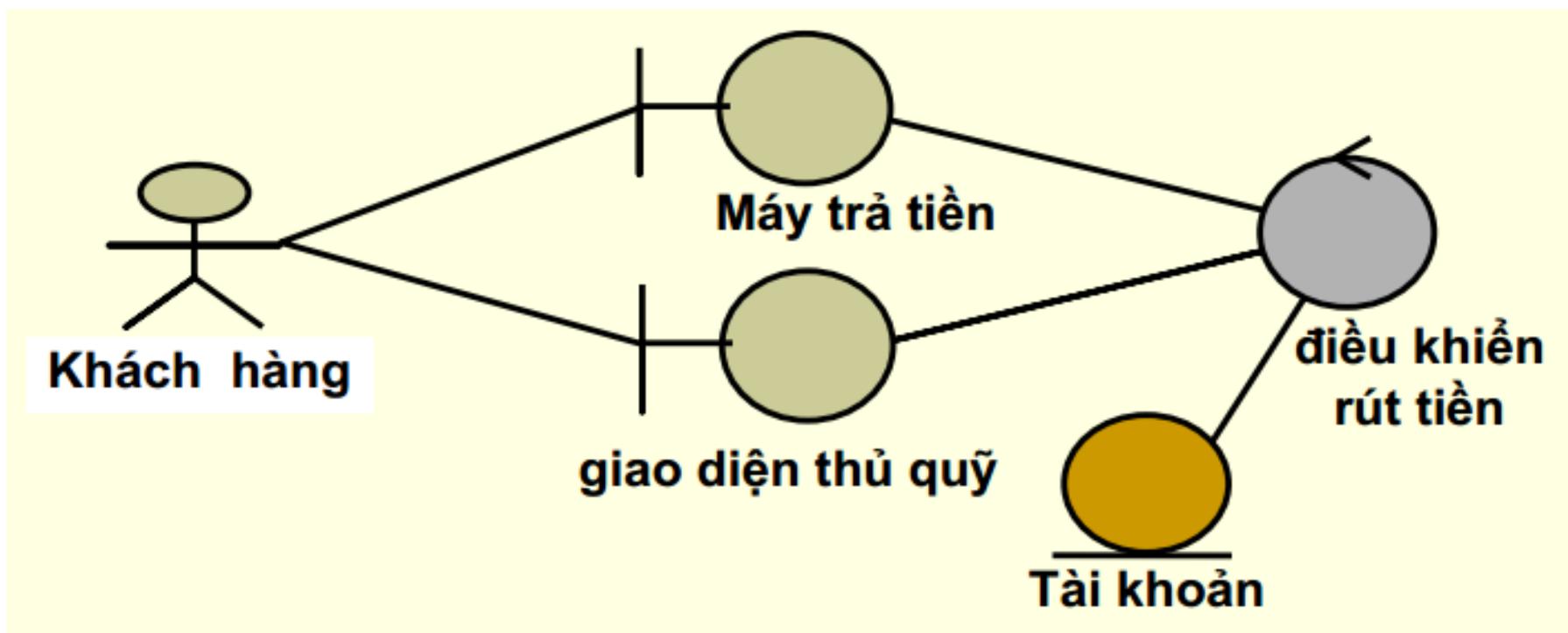
2. Phân tích 1 gói các ca sử dụng



Các lớp phân tích thực thi ca sử dụng “Rút tiền”

VD Phân tích HĐT

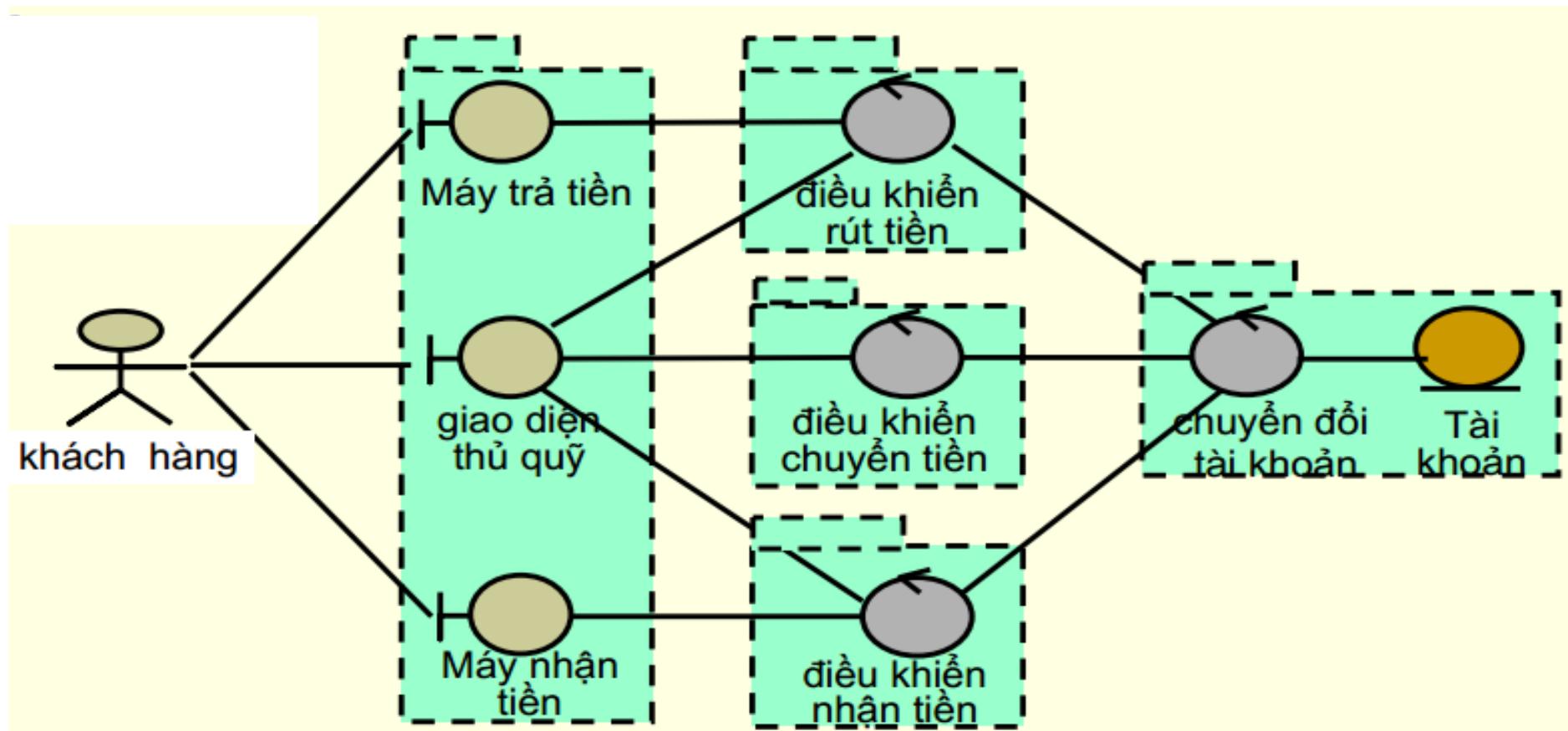
- Lược đồ phân tích 1 gói các ca sử dụng



Các lớp phân tích và quan hệ giữa chúng

VD Phân tích HĐT

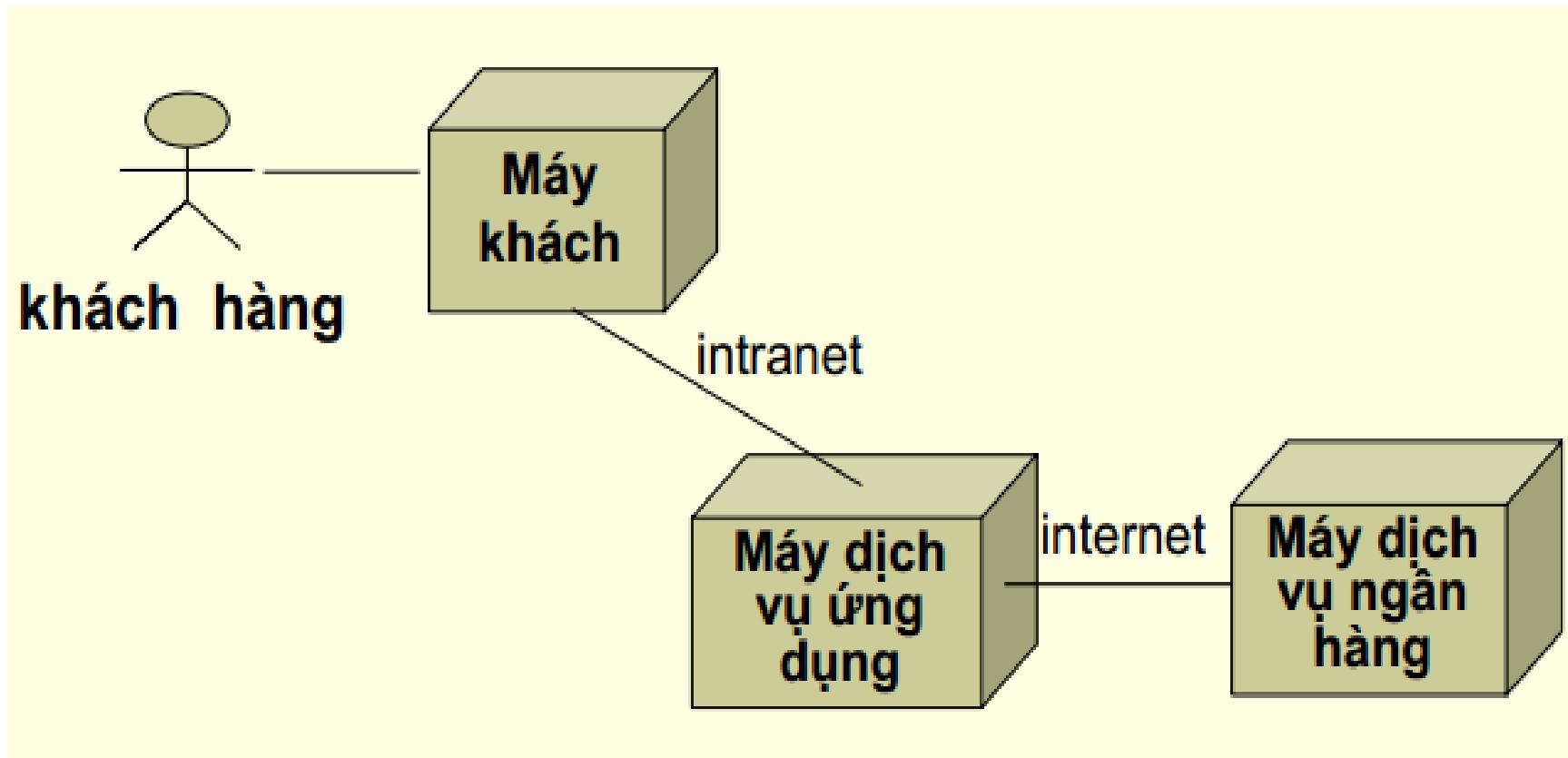
- Lược đồ gói các lớp phân tích



Các gói của các lớp phân tích

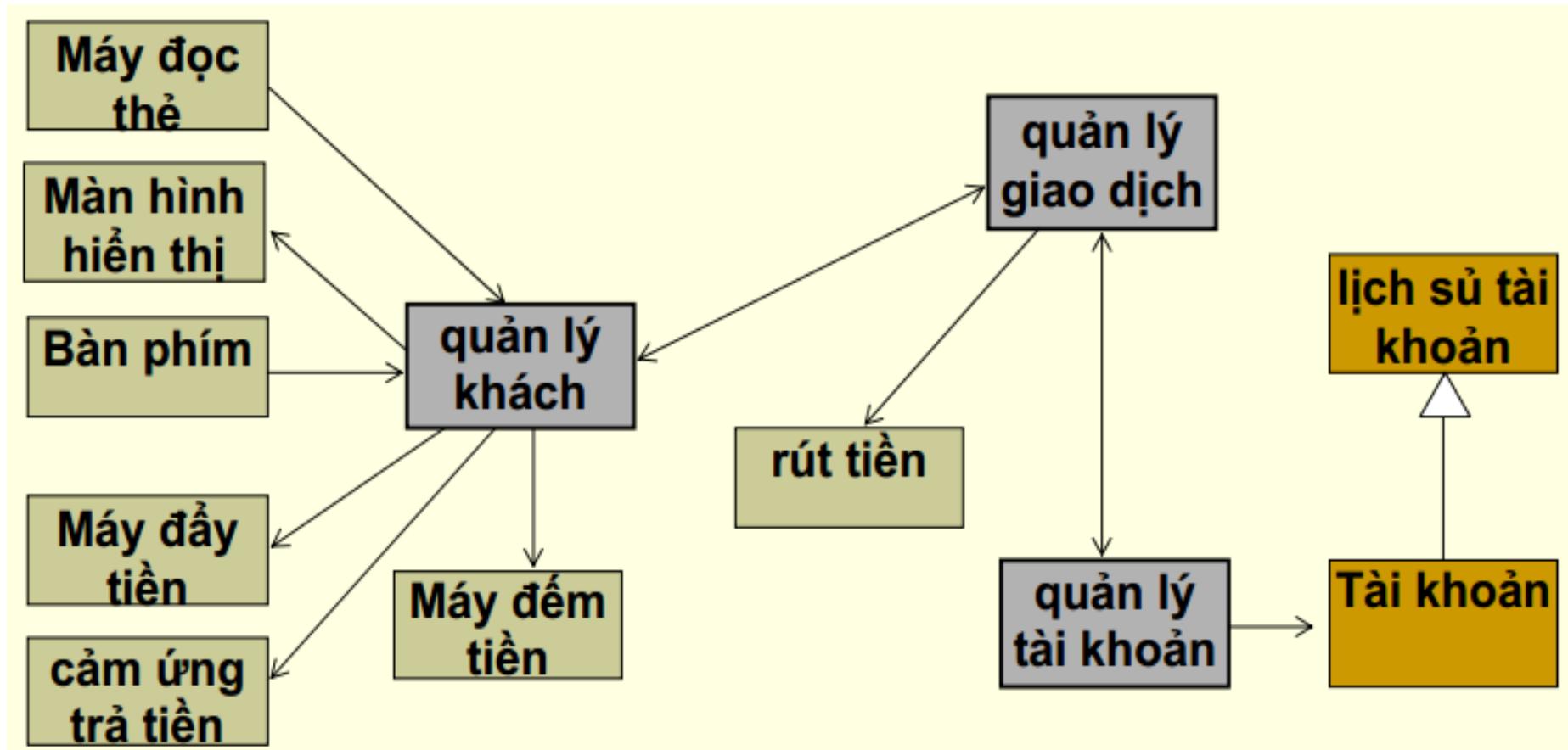
VD Thiết kế HĐT

- Thiết kế hệ thống



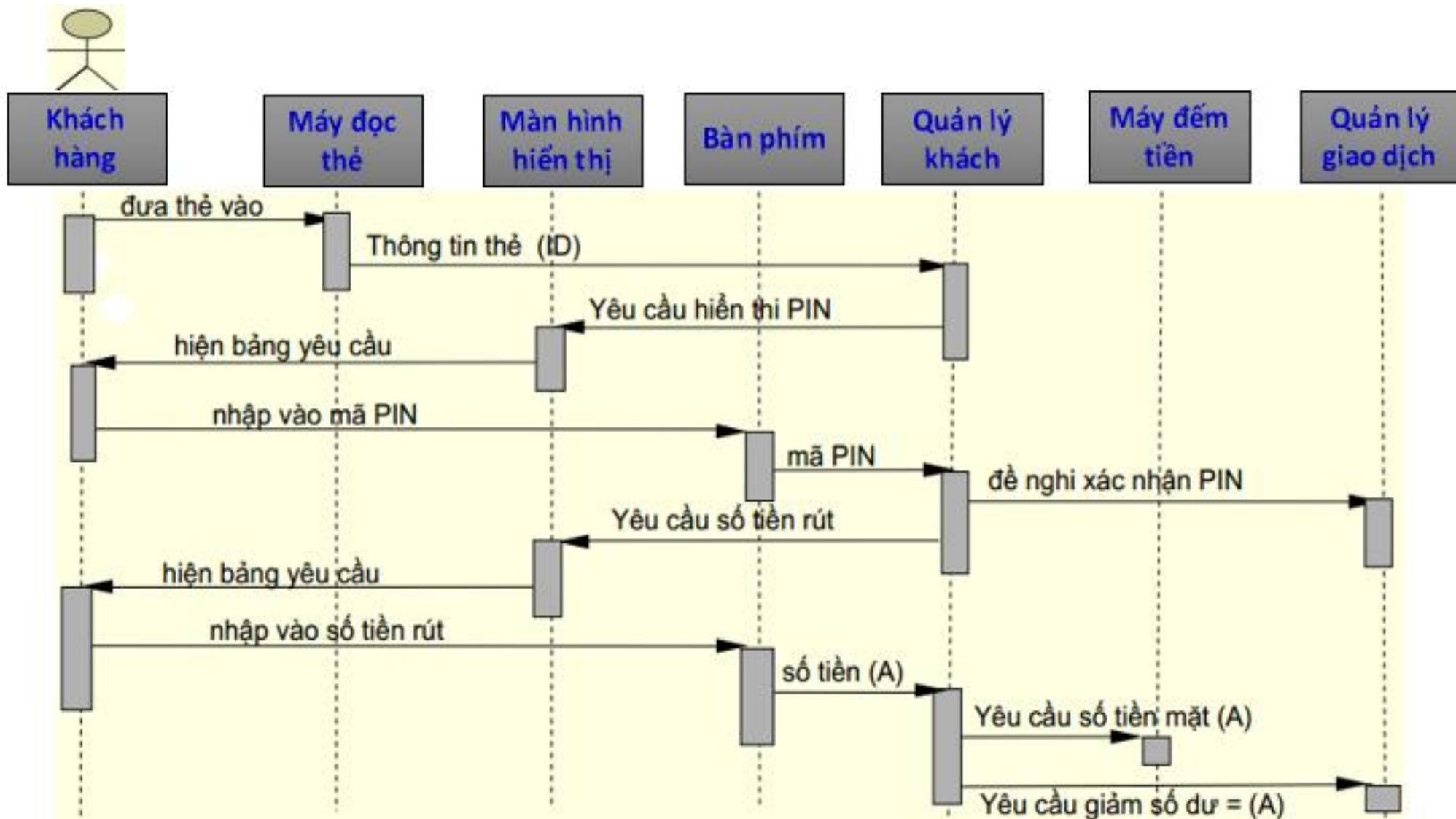
VD Thiết kế HĐT

- Biểu đồ các lớp thiết kế



Các lớp thiết kế tham gia thực hiện ca sử dụng ‘Rút tiền’

VD Thiết kế HĐT



Lược đồ tuần tự thực hiện ca sử dụng “Rút tiền”

VD Thiết kế HĐT

- **Thiết kế lớp tài khoản**
 - **Bảng các thuộc tính**

Tên thuộc tính	Kiểu	Nội dung
Idtaikh	String	Định danh tài khoản
Sotkh	String	Số tk dành cho 1 KH gồm chữ, số
Sodu	Money	Số dư có trong tk, đơn vị đo là tiền tệ

VD Thiết kế HĐT

- **Thiết kế lớp tài khoản**
 - **Bảng các tác vụ**

Tên tác vụ	Ý nghĩa
Taolap()	Tạo 1 tk cho KH mới
Gui()	Bổ sung tiền gửi vào tk
Chuyen()	Chuyển một số tiền từ 1 tk sang tk khác
Rut()	Rút một số tiền từ tk
Dong()	Đóng tk

VD Thiết kế HĐT

- Thiết kế lớp tài khoản

Taikhoan
<p>IDtaikh:<i>string</i> sotaikh: <i>string</i> sodu: <i>money</i></p> <p>+ taolap (<i>sotkh:string, sotien;money</i>) + gui(<i>sotkh:string, soien:money</i>) + chuyen(<i>sotkh:stringk, sotien:money,</i> <i>sotkh2:string</i>) + rut (<i>sotk:h:string, sotien;money</i>) dong()</p>

Lớp Taikhoan với các thuộc tính và phương thức

Nội dung chính

- Khái niệm thiết kế phần mềm
- Các hoạt động thiết kế
- **Thiết kế hướng đối tượng**
 - Vấn đề tồn tại trong hướng cấu trúc
 - Khái niệm liên quan đến đối tượng
 - Ngôn ngữ UML
 - Phân tích HĐT
 - Thiết kế HĐT
 - Sử dụng mẫu thiết kế

Mẫu thiết kế - Pattern

- Khái niệm:
 - Mô tả giải pháp của một trường hợp chung có thể áp cho các trường hợp khác tương tự gọi là mẫu thiết kế.
- Áp dụng: Khi thiết kế mà có nhiều trường hợp tương tự.
- Mô tả 1 mẫu bao gồm:
 - Vấn đề đặt ra (ngữ cảnh)
 - Giải pháp
 - Kết quả
 - Các mẫu liên quan
 - Mô hình mẫu

Mẫu thiết kế - Pattern

- Bản chất mẫu:
 - Mẫu thiết kế là một sự đúc kết từ kinh nghiệm
 - Không phải là cái gì quá mới mẻ, cao siêu.
- 5 mẫu PM gán trách nhiệm chung GRASP thường được sử dụng:
 - Expert (chuyên gia)
 - Creator (bộ tạo lập)
 - Low Coupling (ghép nối lỏng)
 - Hight Cohesion (kết dính cao)
 - Controller (bộ điều khiển)

VD một số mẫu thiết kế

- **Mô hình mẫu của mẫu chuyên gia**
 - **Vấn đề:** Nguyên tắc gán trách nhiệm cho 1 đối tượng là gì?
 - **Giải pháp:** Hãy gán trách nhiệm cho đối tượng có đủ thông tin để thực hiện trách nhiệm đó
 - **Kết quả:** Giảm sự phụ thuộc vào lớp khác
 - **Mẫu liên quan:** kết dính cao, ghép nối lỏng

Lợi ích sử dụng mẫu thiết kế

- Cho ta giải pháp của vấn đề không cần tìm kiếm
- Nâng cao tính tái sử dụng
- Cho thiết kế tốt và chất lượng hệ thống cao