CÔNG NGHỆ PHẦN MỀM

Chương 6 – Thiết kế kiến trúc phần mềm

TUẦN 7, 8

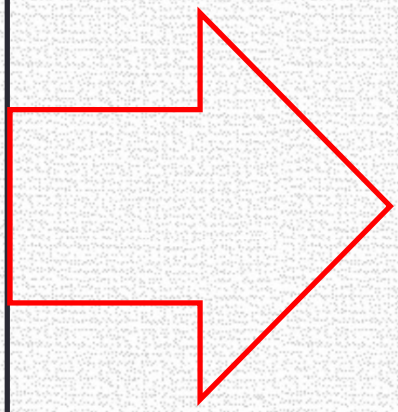
Các nội dung trình bày

- Các quyết định thiết kế kiến trúc (Architectural design decisions)
- Các góc nhìn kiến trúc (Architectural views)
- Các mẫu kiến trúc (Architectural patterns)
- Các kiến trúc phần mềm (Application architect)

Cấu trúc phần mềm

Các yêu cầu chức năng và phi chức năng

Phần mềm cần xây dựng

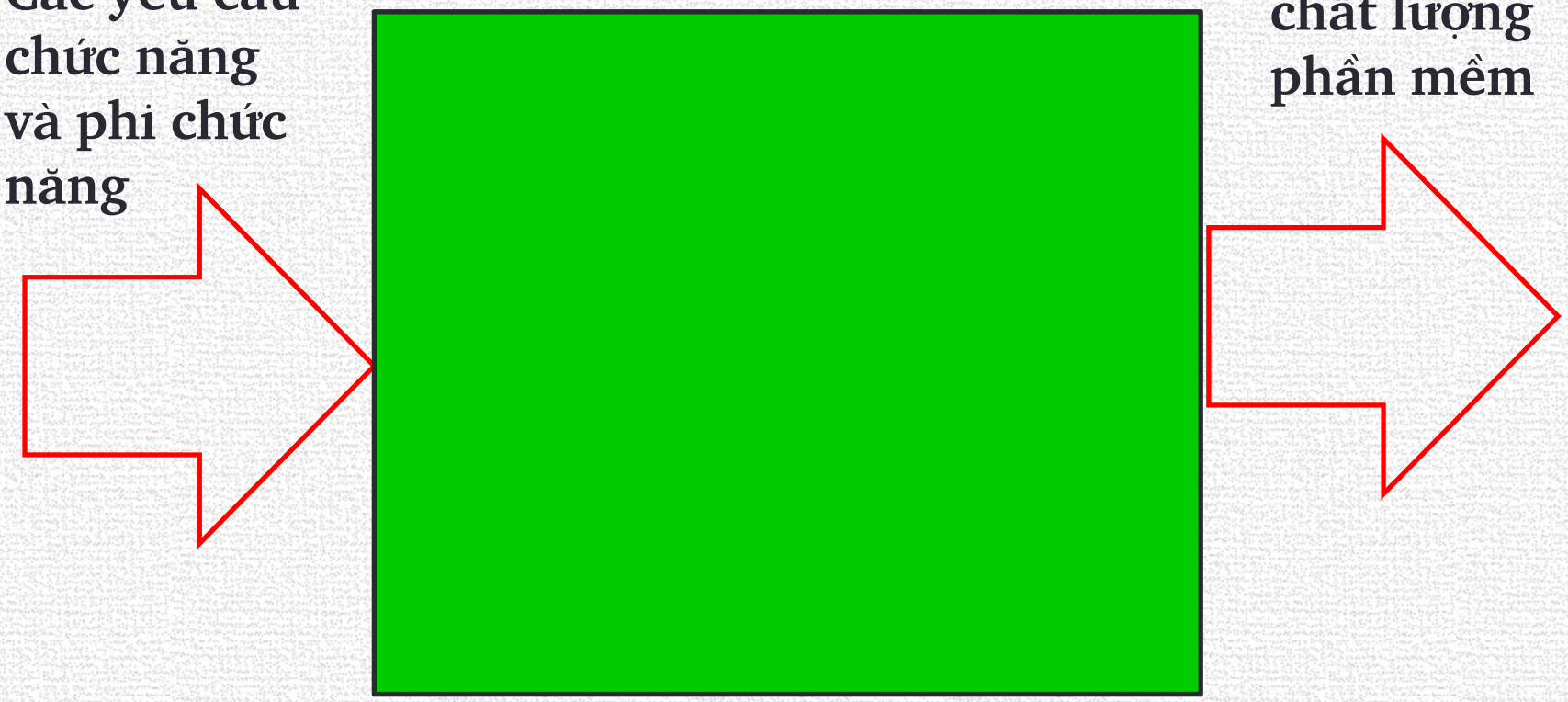


Các chức năng và chất lượng phần mềm

Phần mềm cần xây dựng có hình dạng ra sao ? Cấu trúc tổ chức của phần mềm như thế nào ?

Cấu trúc phần mềm

Các yêu cầu chức năng và phi chức năng

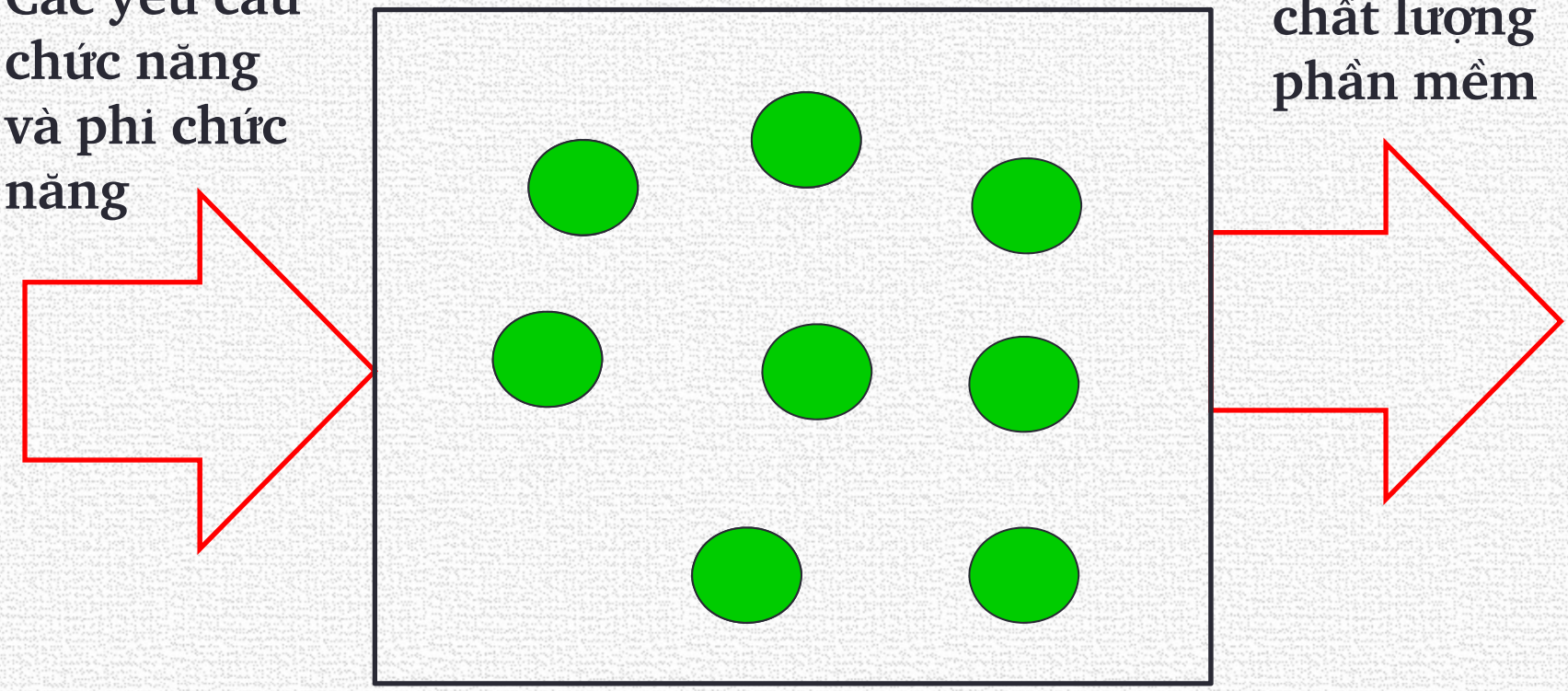


Các chức năng và chất lượng phần mềm

Cấu trúc đơn thể là dễ thấy nhất, nhưng chỉ thích hợp cho phần mềm nhỏ, còn nếu phần mềm rất lớn và phức tạp, cấu trúc đơn thể không phù hợp.

Cấu trúc phần mềm

Các yêu cầu chức năng và phi chức năng

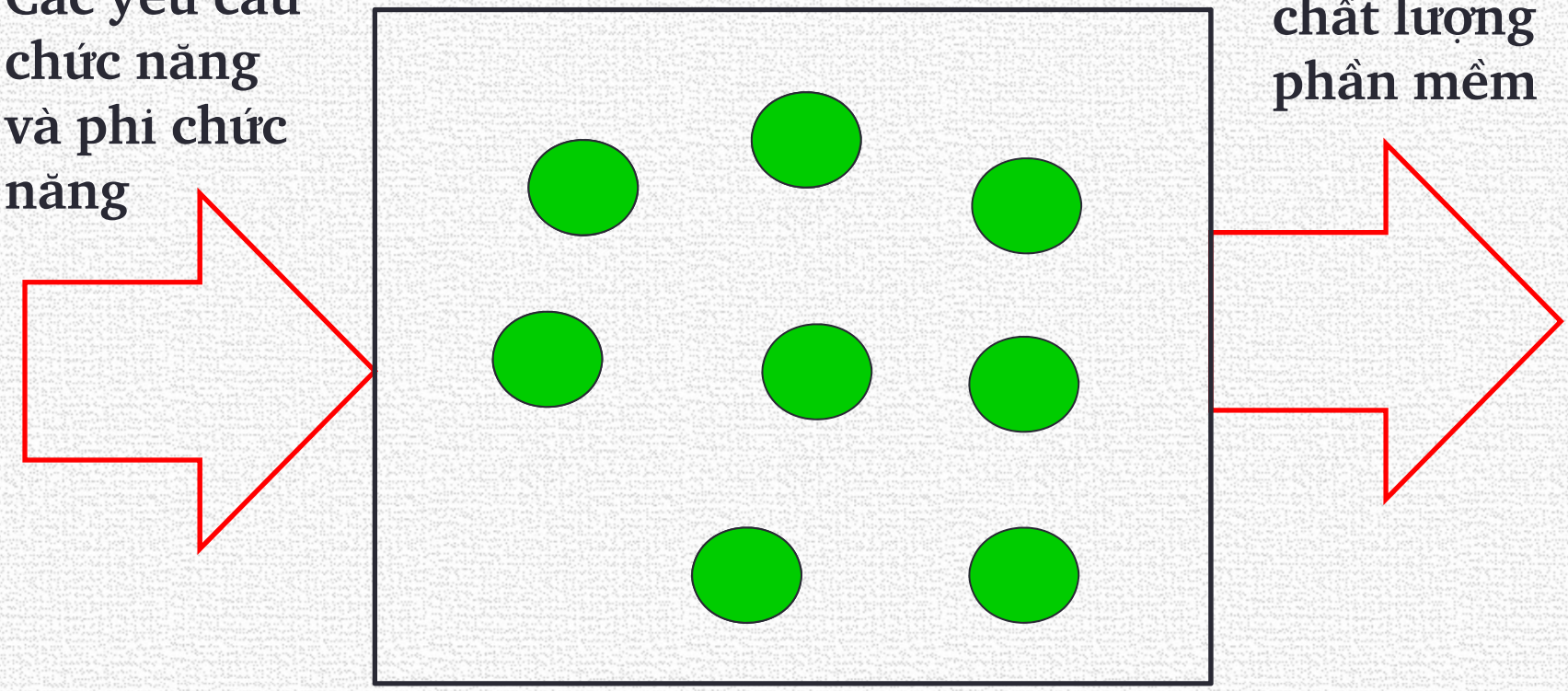


Các chức năng và chất lượng phần mềm

Nếu phần mềm rất lớn và phức tạp, ta sẽ chọn cấu trúc phức hợp : phần mềm gồm nhiều phần tử cấu thành. Chủng loại các phần tử cấu thành phần mềm như thế nào và mối quan hệ giữa chúng ra sao để ta có thể dễ dàng quản lý chúng theo thời gian?

Cấu trúc phần mềm

Các yêu cầu chức năng và phi chức năng

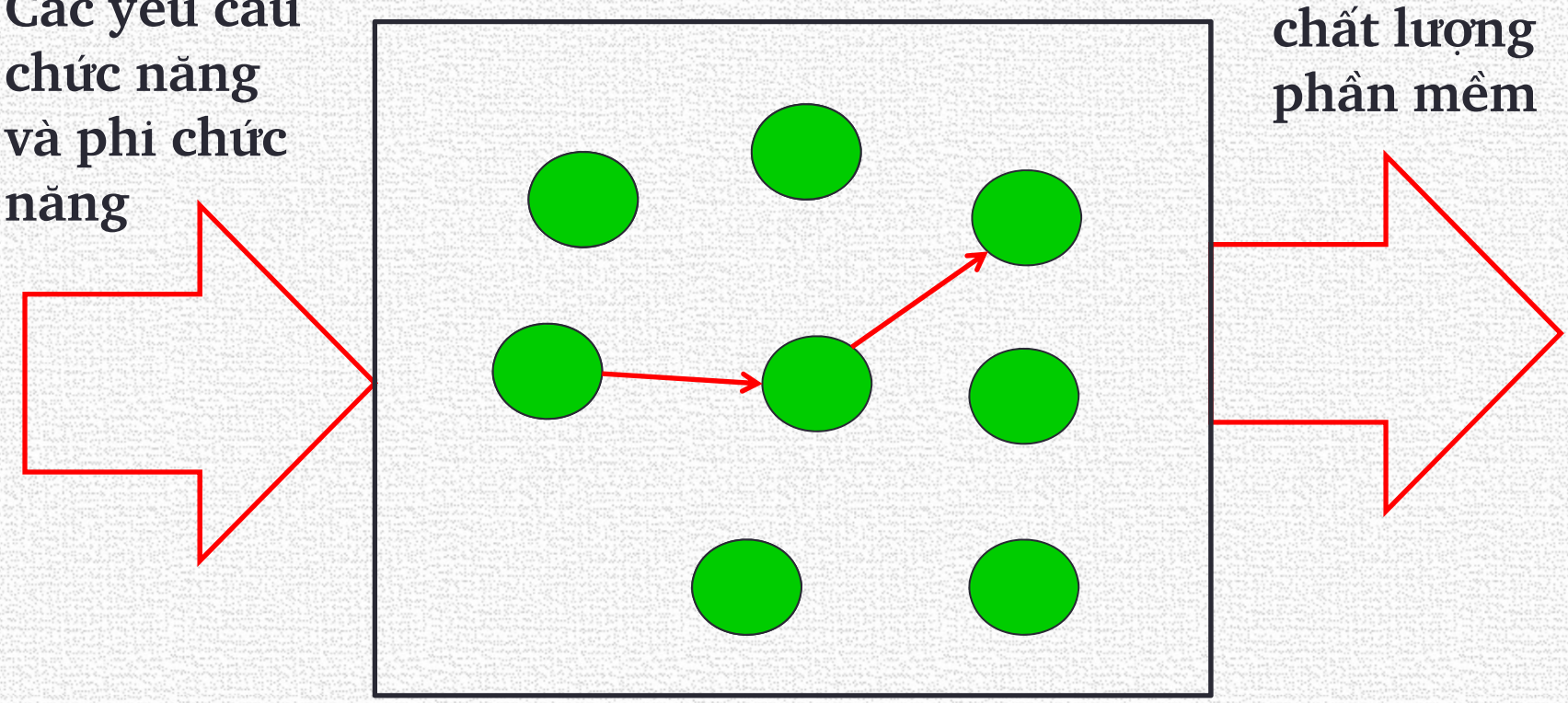


Các chức năng và chất lượng phần mềm

Các phần tử cấu thành phần mềm lớn thường có số lượng rất lớn, nhưng để dễ xây dựng và quản lý chúng, ta đòi hỏi chúng phải thuần nhất cùng một chủng loại. Mô hình hướng đối tượng gọi phần tử này là đối tượng. Như vậy phần mềm là tập các đối tượng.

Cấu trúc phần mềm

Các yêu cầu chức năng và phi chức năng



Các chức năng và chất lượng phần mềm

Mỗi phần tử phải tương tác với 1/n phần tử khác, nếu không nó vô dụng. Nhưng để dễ dàng quản lý các phần tử, ta phải hạn chế tối đa sự tương tác giữa chúng. Tính đóng gói của mô hình hướng đối tượng giải quyết vấn đề này.

Cấu trúc phần mềm

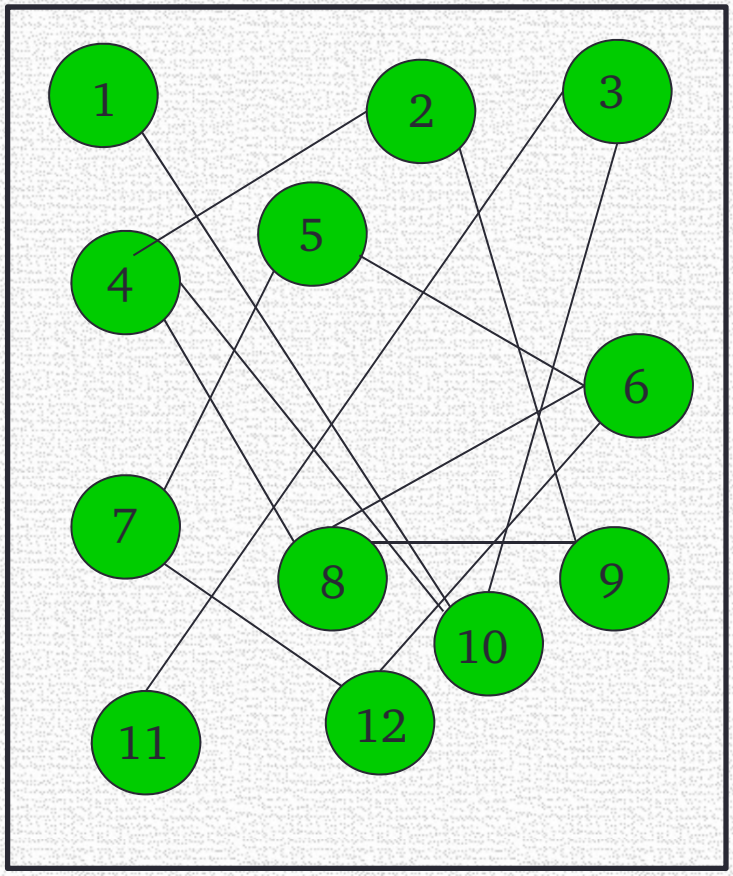
Hạn chế sự tương tác của từng phần tử là phải tối thiểu hóa các tương tác của từng chiều :

- Chiều từ ngoài tương tác vào phần tử : Che dấu tối đa các chi tiết hiện thực của mình, chỉ cho các phần tử khác bên ngoài thấy và dùng 1 số tối thiểu dịch vụ của mình.
- Chiều từ trong ra ngoài : Hạn chế tối đa sự nhờ vả các phần phần bên ngoài : ta cần làm các thành phần nội bộ của đối tượng có tính kết dính (cohesion) cao nhất có thể có.

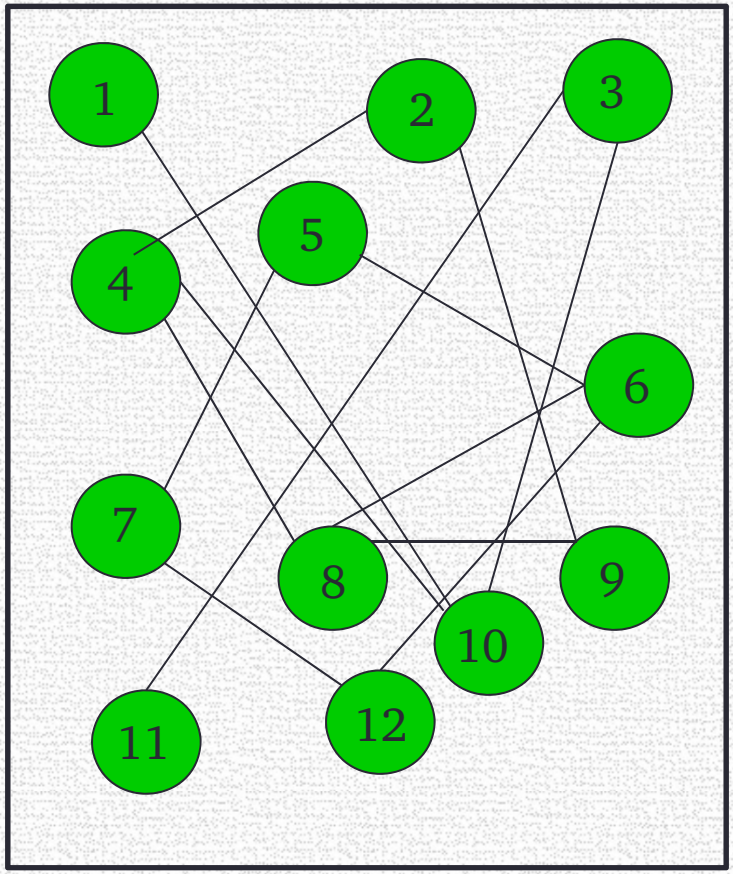
Cấu trúc phần mềm

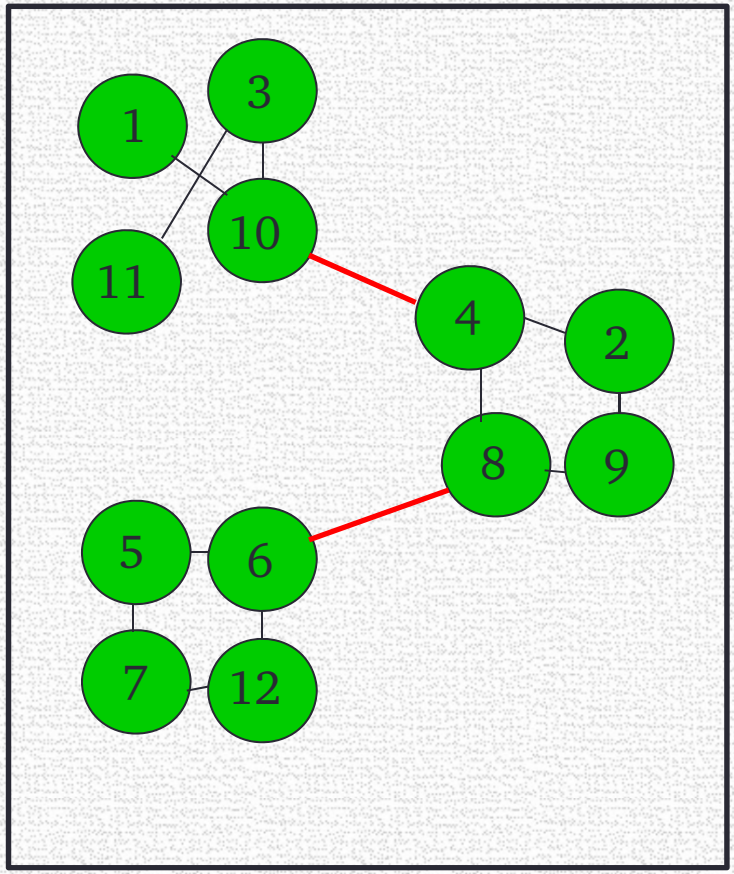
Ta dùng thuật ngữ "cấu trúc phầm mềm" để miêu tả các phần tử cụ thể cấu thành phần mềm và mối quan hệ cụ thể giữa chúng.

Thí dụ phần mềm bên phải có 12 phần tử và mối quan hệ cụ thể giữa chúng được miêu tả như hình. Nếu quan sát kỹ, ta thấy các phần tử có rất ít mối quan hệ lẫn nhau, tuy nhiên vì số lượng phần tử lớn nên ta có cảm giác phần mềm quá phức tạp, khó quản lý.



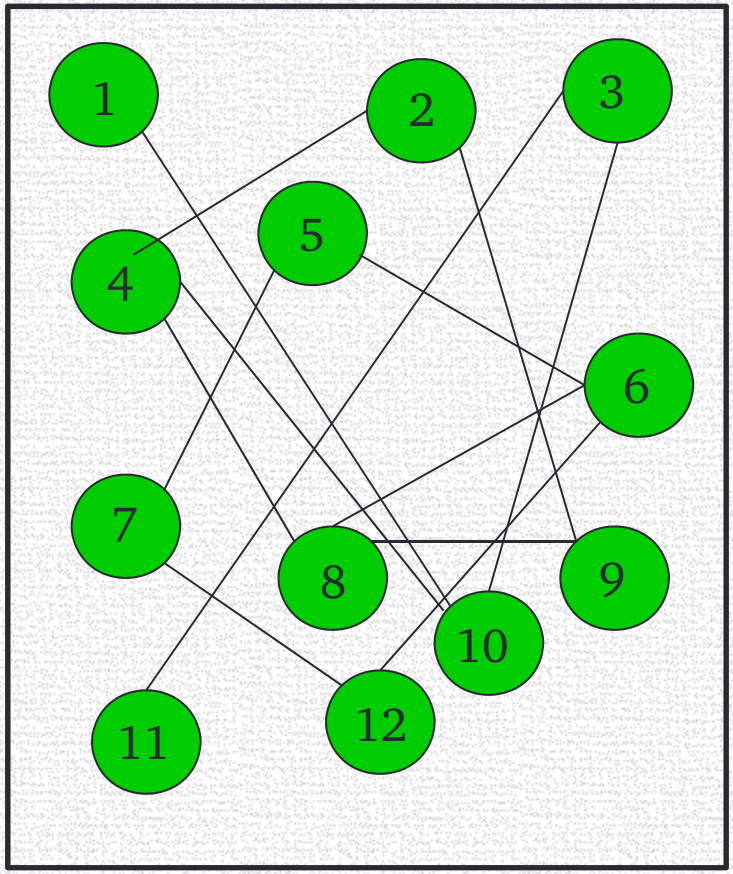
Cấu trúc phần mềm

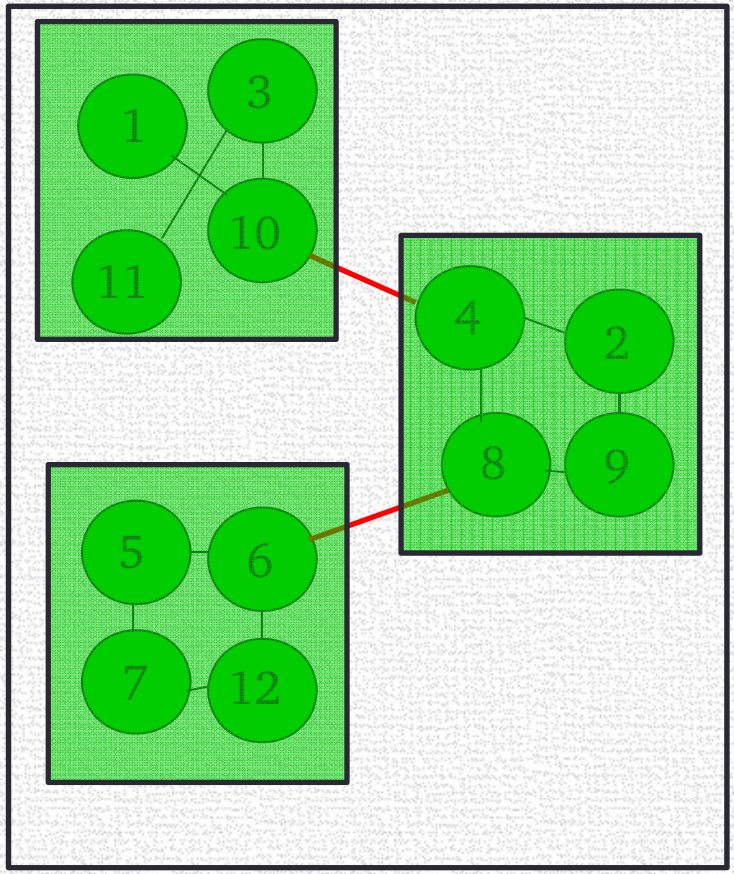




Nếu ta cố gắng sắp lại vị trí các phần tử, ta sẽ được cấu trúc bên phải trật tự hơn và trong sáng hơn.

Cấu trúc phần mềm

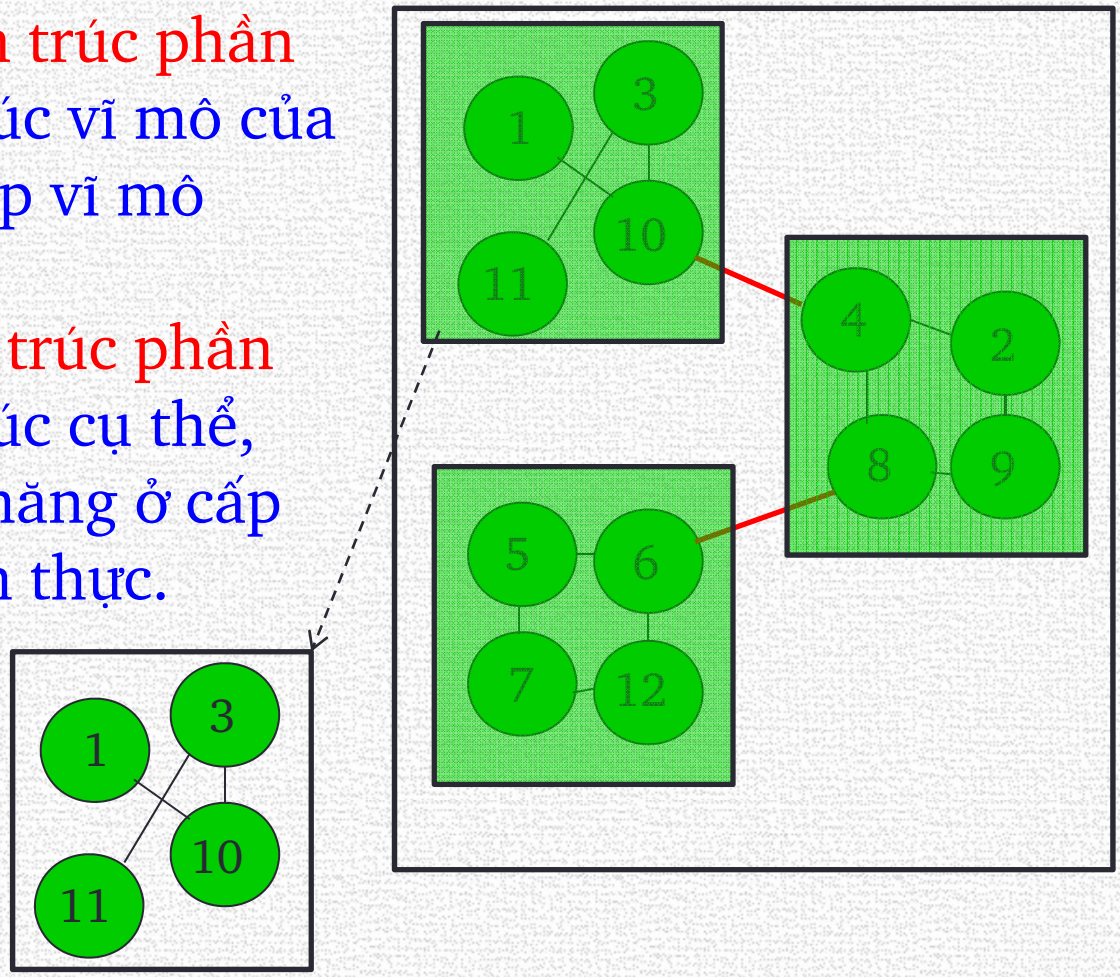




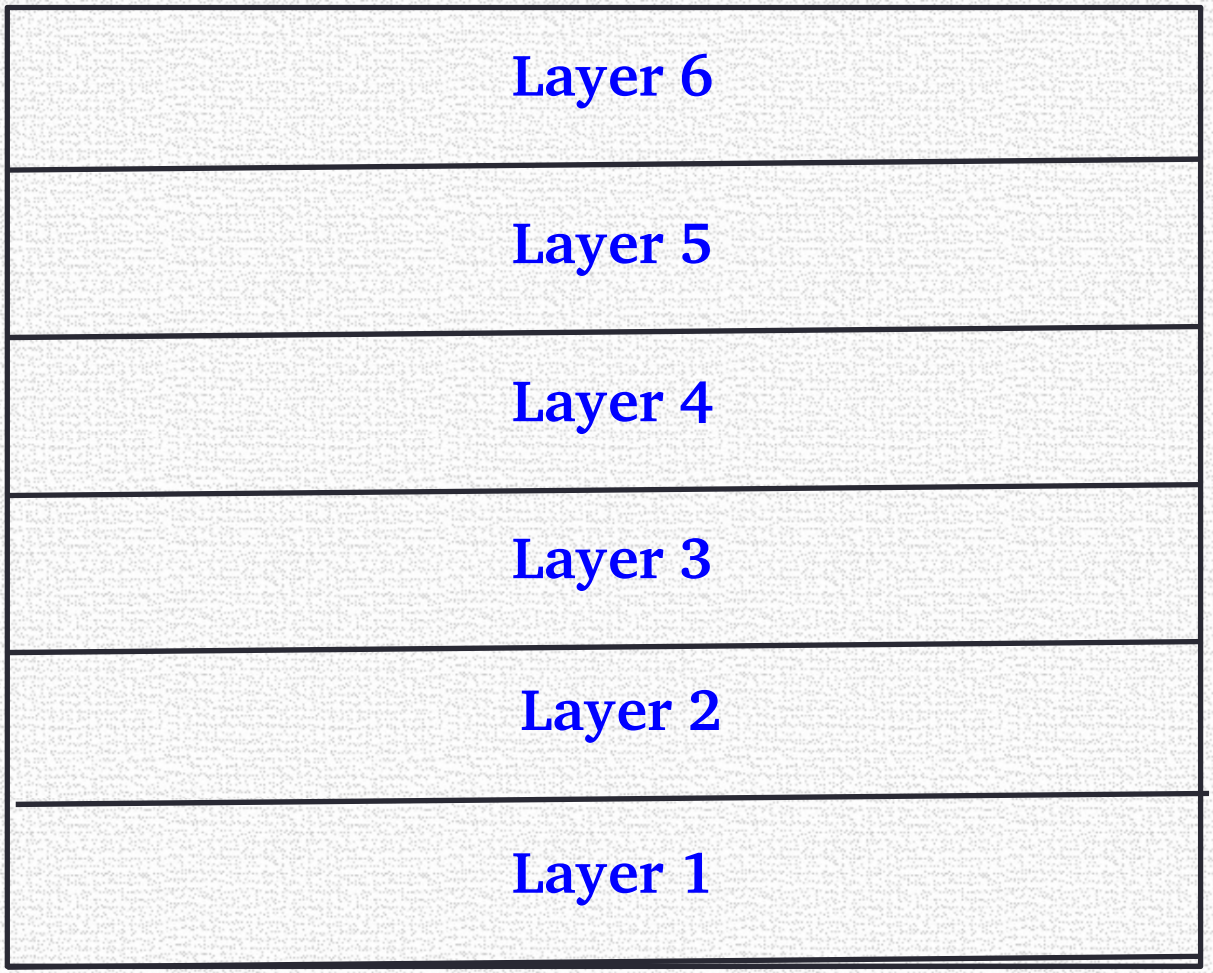
Nếu ta gom mỗi nhóm 4 phần tử thành 1 phần tử luận lý thì kết quả là cấu trúc của phần mềm đơn gản và trong sáng hơn rất nhiều : chỉ có 3 phần tử (thay vì 12) và 2 mối quan hệ (thay vì 13).

Kiến trúc phần mềm

- Ta dùng thuật ngữ kiến trúc phần mềm để miêu tả cấu trúc vĩ mô của phần mềm (ở bất kỳ cấp vĩ mô nào).
- Ta dùng thuật ngữ cấu trúc phần mềm để miêu tả cấu trúc cụ thể, gồm các đơn thể chức năng ở cấp thấp nhất mà ta sẽ hiện thực.

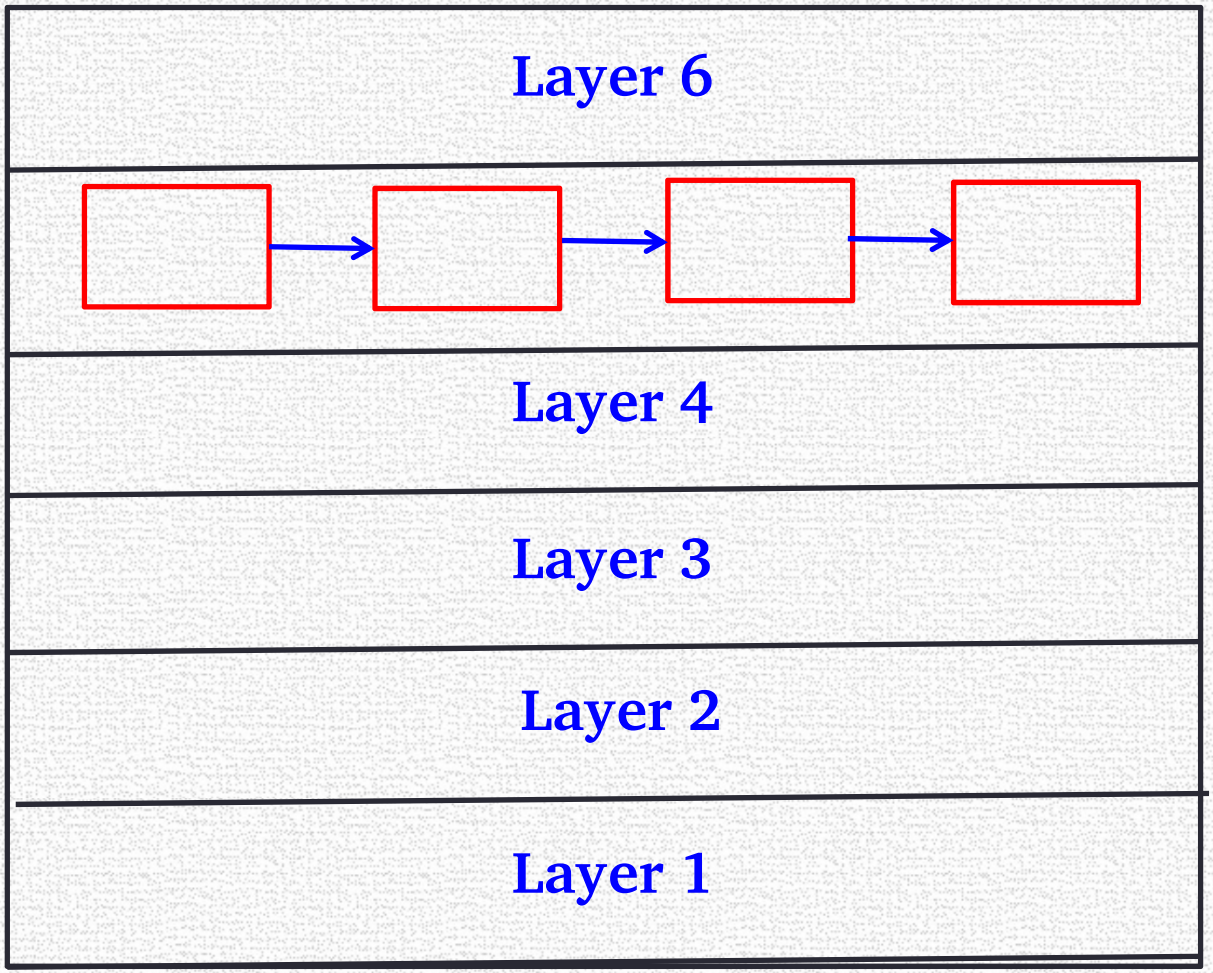


Kiến trúc phần mềm



Thí dụ phần mềm trên có kiến trúc phân cấp, một dạng kiến trúc rất thường gặp.

Kiến trúc phần mềm



Layer 5 của phần mềm có kiến trúc lô tuần tự...

Thiết kế kiến trúc

- Là qui trình nhận dạng các hệ thống con và bộ khung kiểm soát/giao tiếp giữa chúng.
- Thiết kế kiến trúc phải xây dựng được đặc tả kiến trúc phần mềm. Công việc này xảy ra ở đầu giai đoạn thiết kế phần mềm, nó miêu tả mối nối giữa qui trình đặc tả yêu cầu và qui trình thiết kế chi tiết.
- Thường được thực hiện song song với 1 số hoạt động đặc tả yêu cầu phần mềm.
- Nó liên quan đến việc nhận dạng các thành phần phần mềm chính yếu và các mối quan hệ giữa chúng.

Lợi ích của kiến trúc rõ ràng

- Phương tiện giao tiếp giữa các bên hữu quan : giúp các bên tập trung thảo luận.
- Phương tiện phân tích phần mềm : xem phần mềm có thể thỏa mãn các yêu cầu phi chức năng không ?
- Giúp dùng lại ở qui mô lớn :
  - Kiến trúc phần mềm có thể được dùng lại cho nhiều phần mềm khác nhau.
  - Các kiến trúc dòng sản phẩm (product line) có thể được xây dựng để dùng chung cho tất cả phần mềm thuộc dòng này.

Các quyết định thiết kế kiến trúc

- Thiết kế kiến trúc là quá trình sáng tạo, tùy kiểu phần mềm mà quá trình thiết kế kiến trúc sẽ khác nhau.
- Mặc dù vậy, có 1 số quyết định chung cho tất cả qui trình thiết kế, những quyết định này ảnh hưởng tới các yêu cầu phi chức năng của phần mềm.

Các quyết định thiết kế kiến trúc

- Có 1 kiến trúc phần mềm tổng quát nào đó mà có thể được dùng trong thiết kế phần mềm của ta không ?
- Phần mềm phân tán như thế nào ?
- Kiểu kiến trúc nào thích hợp cho phần mềm của mình ?
- Dùng cách tiếp cận nào để cấu trúc phần mềm ?
- Phần mềm được phân rã thành các module như thế nào ?
- Phải dùng chiến lược kiểm soát nào ?
- Thiết kế kiến trúc được đánh giá như thế nào ?
- Kiến trúc phần mềm được lập tài liệu như thế nào ?
- …

Dùng lại kiến trúc phần mềm

- Các phần mềm trong cùng 1 lĩnh vực thường có kiến trúc tương tự, kiến trúc này phản ảnh các khái niệm của lĩnh vực đó.
- Các ứng dụng thuộc 1 dòng sản phẩm được xây dựng trên 1 kiến trúc lõi chung cộng với 1 số thay đổi để thỏa mãn các yêu cầu riêng của phần mềm đó.
- Kiến trúc phần mềm có thể được thiết kế dựa trên 1 hay nhiều mẫu thiết kế có sẵn :
  - Mỗi mẫu thiết kế miêu tả sự thiết yếu của 1 kiến trúc và có thể được cụ thể hóa theo các cách khác nhau.
  - Sẽ được đề cập chi tiết trong chương này.

Các yêu cầu phi chức năng

- Có thể là các ràng buộc trong việc phát triển hay chạy phần mềm :
  - Phần mềm phải được viết trong 6 tháng.
  - Phần mềm phải chạy được trên android…
- Có thể là các thuộc tính miêu tả chất lượng phần mềm :
  - Người dùng chỉ cần tối đa 4 giờ huấn luyện là có thể dùng được tất cả chức năng của phần mềm.
  - Phải dễ nâng cấp, hiệu chỉnh…

Các yêu cầu phi chức năng

- Facility : phương tiện được tích hợp trong phần mềm. Thí dụ :
  - Trình gỏ phím Unikey phải có phương tiện chuyển chuỗi tiếng Việt từ mã này sang mã kia.

Các yêu cầu phi chức năng

- Volume : khối lượng dữ liệu lớn cần được xử lý bởi chương trình. Thí dụ :
  - Chương trình dịch phải dịch được file mã nguồn dài 10MB.
  - Trình liên kết phải liên kết được 1000 module chức năng khác nhau lại.
  - Trình xem phim phải chiếu được file film dài 100GB.

Các yêu cầu phi chức năng

- Stress : chương trình phải chạy tốt trong những tình hướng bất lợi nhất. Thí dụ :
  - Chương trình phải chạy tốt khi máy đang chạy đồng thời 10 phần mềm lớn khác.
  - Chương trình vẫn chạy tốt trong khi mạng bị đứng...

Các yêu cầu phi chức năng

- Usability : chương trình phải dễ dùng. Thí dụ :
  - Người dùng chỉ cần tối đa 4 giờ huấn luyện là có thể dùng được tất cả chức năng của phần mềm.

Các yêu cầu phi chức năng

- Usability : chương trình phải dễ dùng. Thí dụ :
  - Người dùng chỉ cần tối đa 4 giờ huấn luyện là có thể dùng được tất cả chức năng của phần mềm.
  - Các cảnh báo phải bằng tiếng Việt, ngắn gọn, trong sáng, dễ hiểu.

Các yêu cầu phi chức năng

- Seviceability : các dịch vụ cộng thêm. Thí dụ :
  - Chương trình cần xuất nội dung thô của bộ nhớ chương trình khi bị lỗi, trước khi đứng máy.

Các yêu cầu phi chức năng

- Security : tính an toàn bảo mật. Thí dụ :
  - Chương trình chỉ cung cấp dịch vụ hiệu chỉnh thông tin cho admin.

Các yêu cầu phi chức năng

- Performance : hiệu năng. Thí dụ :
  - Trình chiếu phim phải chiếu tối thiểu 60 frames độ phân giải 4k /s.
  - Trình gởi file ra mạng phải nén dữ liệu kịp thời với tốc độ gởi ra mạng….

Các yêu cầu phi chức năng

- Storage : sử dụng bộ nhớ. Thí dụ :
  - Chương trình phải chạy được trên máy chỉ có 4GB RAM.
  - Chương trình quản lý được ổ đĩa dùng lượng 16TB….

Các yêu cầu phi chức năng

- Configuration : cấu hình làm việc. Thí dụ :
  - Chương trình phải chạy được trên hệ điều hành Android.
  - Chương trình phải chạy được với IIS server….

Các yêu cầu phi chức năng

- Installability : dễ cài đặt. Thí dụ :
  - Chương trình phải được cài đặt dễ dàng.
  - Chương trình chạy được ngay mà không cần cài đặt….

Các yêu cầu phi chức năng

- Reliability : độ tin cậy, ổn định. Thí dụ :
  - Chương trình phải chạy tốt với các dữ liệu khác nhau, với các thao tác user khác nhau….

Các yêu cầu phi chức năng

- Availability : độ sẵn sàng. Thí dụ :
  - Chương trình phải luôn đáp ứng với người dùng theo thời gian….

Các yêu cầu phi chức năng

- Maintainability : độ dễ bảo trì. Thí dụ :
  - Nếu có lỗi, chương trình phải được sửa tối đa 1 giờ là có thể chạy lại bình thường….

Các yêu cầu phi chức năng

- Modifiability : độ dễ hiệu chỉnh, nâng cấp. Thí dụ :
  - Khi có yêu cầu mới, chương trình phải được hiệu chỉnh ít nhất và dễ dàng….

Các yêu cầu phi chức năng

- Buildability : tính dễ xây dựng. Thí dụ :
  - Thị trường về phần mềm tăng rất cao trong 8 tháng tới….

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc đơn thể (Monolithic)

- **Đặc tả** : Hệ thống chỉ gồm duy nhất 1 module, module này chứa mọi thứ của chương trình : danh sách các lệnh thực thi miêu tả giải thuật cần thực hiện + danh sách các biến dữ liệu bị xử lý.
  - giao tiếp giữa các thành phần là cục bộ và rất hiệu quả.
  - thích hợp cho những phần mềm nhỏ, đơn giản.
  - không thích hợp cho những phần mềm lớn và phức tạp.

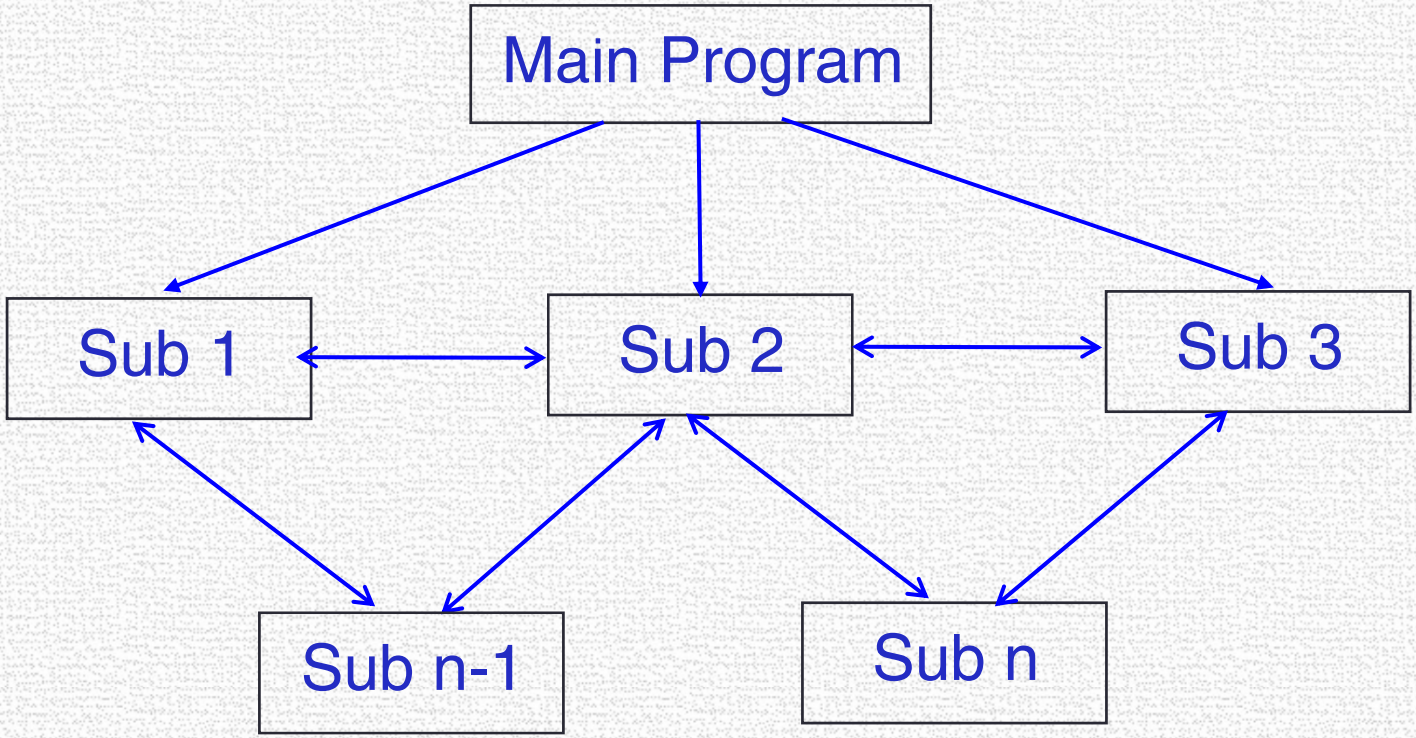
8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc Chương trình chính và thủ tục (MainProgram/Subroutine Architecture)

- **Đặc tả** : Hệ thống phần mềm gồm 1 chương trình chính và 1 tập các thủ tục chức năng cần thiết.
- dùng cách phân rã theo dạng cây phân cấp : dựa trên mối quan hệ định nghĩa-sử dụng.
- chỉ có 1 thread kiểm soát duy nhất : được hỗ trợ trực tiếp bởi các ngôn ngữ lập trình.
- ẩn chứa cấu trúc hệ thống con : các thủ tục có mối quan hệ mật thiết thường được gộp thành module.
- lý do của sự phân cấp : độ đúng đắn của 1 thủ tục phụ thuộc vào sự đúng đắn của các thủ tục mà nó gọi.

8.3 Các mẫu kiến trúc phổ dụng

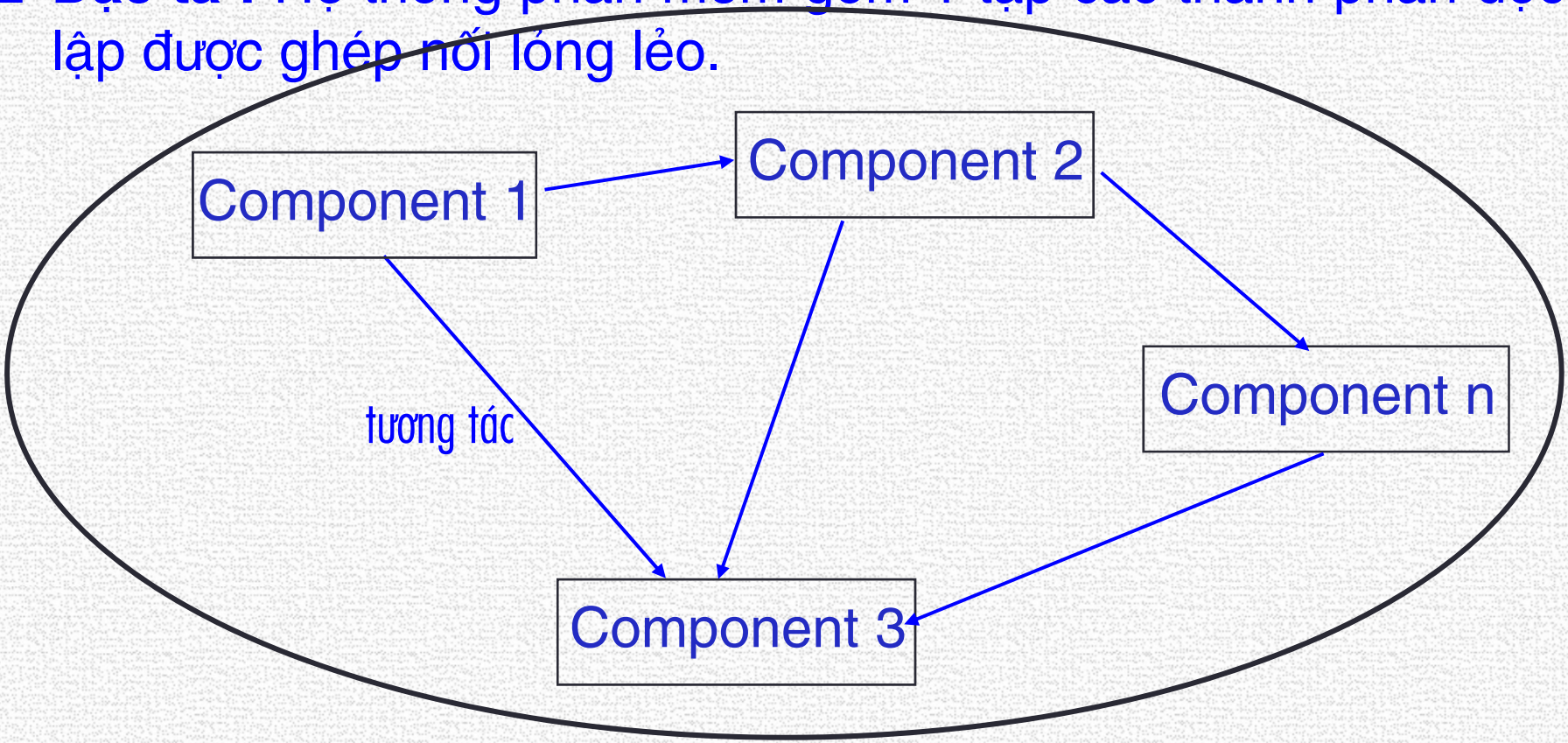
Kiến trúc Chương trình chính và thủ tục (MainProgram/Subroutine Architecture)



8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc các thành phần (Components based Architecture)

- **Đặc tả :** Hệ thống phần mềm gồm 1 tập các thành phần độc lập được ghép nối lỏng lẻo.



8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc các thành phần (Components based Architecture)

- **Thành phần** : là nguyên tử cấu thành phần mềm, nó có 1 số tính chất sau :
  - Reusable : dễ dàng dùng lại cho ứng dụng khác
  - Replaceable : dễ dàng được thay thế bởi thành phần mới
  - Not context specific : không có ngữ cảnh
  - Extensible : dễ nới rộng
  - Encapsulated : đóng gói và ẩn chi tiết hiện thực
  - Independent : độc lập cao

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc các thành phần (Components based Architecture)

- **Ưu điểm của kiến trúc các thành phần :**
  - Ease of deployment : dễ dàng triển khai
  - Reduced cost : chi phí thấp
  - Ease of development : dễ dàng phát triển
  - Reusable : dễ dàng dùng lại
  - Mitigation of technical complexity : giảm nhẹ độ phức tạp kỹ thuật

8.3 Các mẫu kiến trúc phổ dụng

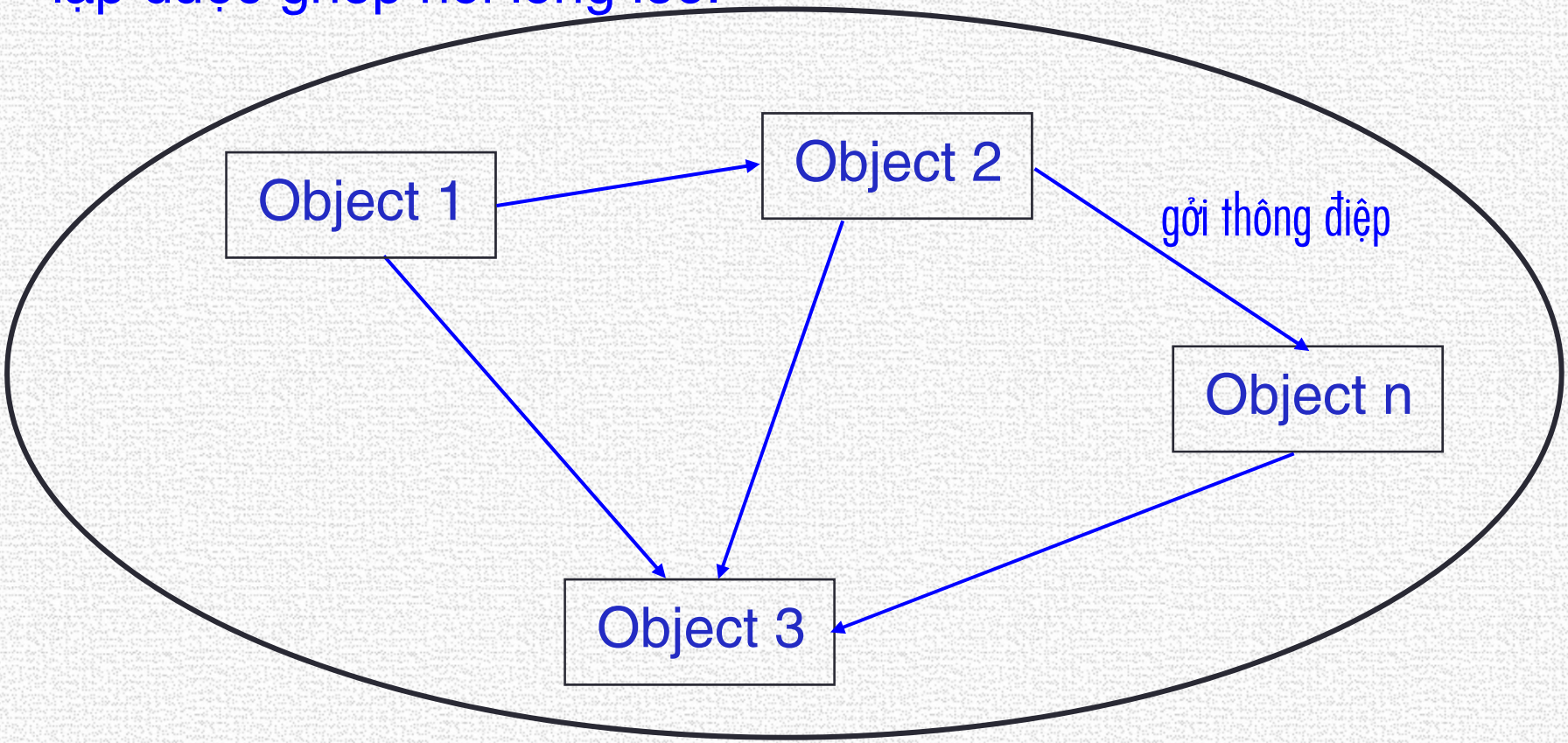
Kiến trúc các thành phần (Components based Architecture)

- **Tình huống nên dùng :** bất kỳ hệ thống phần mềm phức tạp nào.
- **Khuyết điểm :** là mẫu kiến trúc có độ tổng quát cao nên khi hiện thực ta phải tốn nhiều chi phí để vận dụng nó.

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc hướng đối tượng (Objects based Architecture)

- **Đặc tả** : Hệ thống phần mềm gồm 1 tập các đối tượng độc lập được ghép nối lỏng lẻo.



8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc hướng đối tượng (Objects based Architecture)

- **Đối tượng** : là nguyên tử cấu thành phần mềm, nó có 1 số tính chất sau :
  - Reusable : dễ dàng dùng lại cho ứng dụng khác
  - Replaceable : dễ dàng được thay thế bằng đối tượng mới hơn
  - Extensible, Heritable : thừa kế và dễ nới rộng
  - Encapsulated : đóng gói và ẩn chi tiết hiện thực
  - Independent : độ độc lập cao
  - Persistent : thường trù để sẵn sàng phục vụ
  - Aggregation... : bao gộp từ nhiều đối tượng nhỏ

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc hướng đối tượng (Objects based Architecture)

- **Các nguyên tắc chính yếu của kiến trúc hướng đối tượng :**
  - Abstraction : trừu tượng
  - Composition : tích hợp
  - Inheritance : thừa kế
  - Encapsulation : đóng gói
  - Polymorphism : đa xạ
  - Decoupling : quan hệ nhau (phụ thuộc) rất ít

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc hướng đối tượng (Objects based Architecture)

- **Ưu điểm của kiến trúc hướng đối tượng :**
  - Understandable : dễ hiểu
  - Reusable : dễ dùng lại
  - Testable : dễ kiểm thử
  - Extensible : dễ nới rộng
  - Highly Cohesive : kết dính giữa các thành phần trong từng đối tượng cao

8.3 Các mẫu kiến trúc phổ dụng

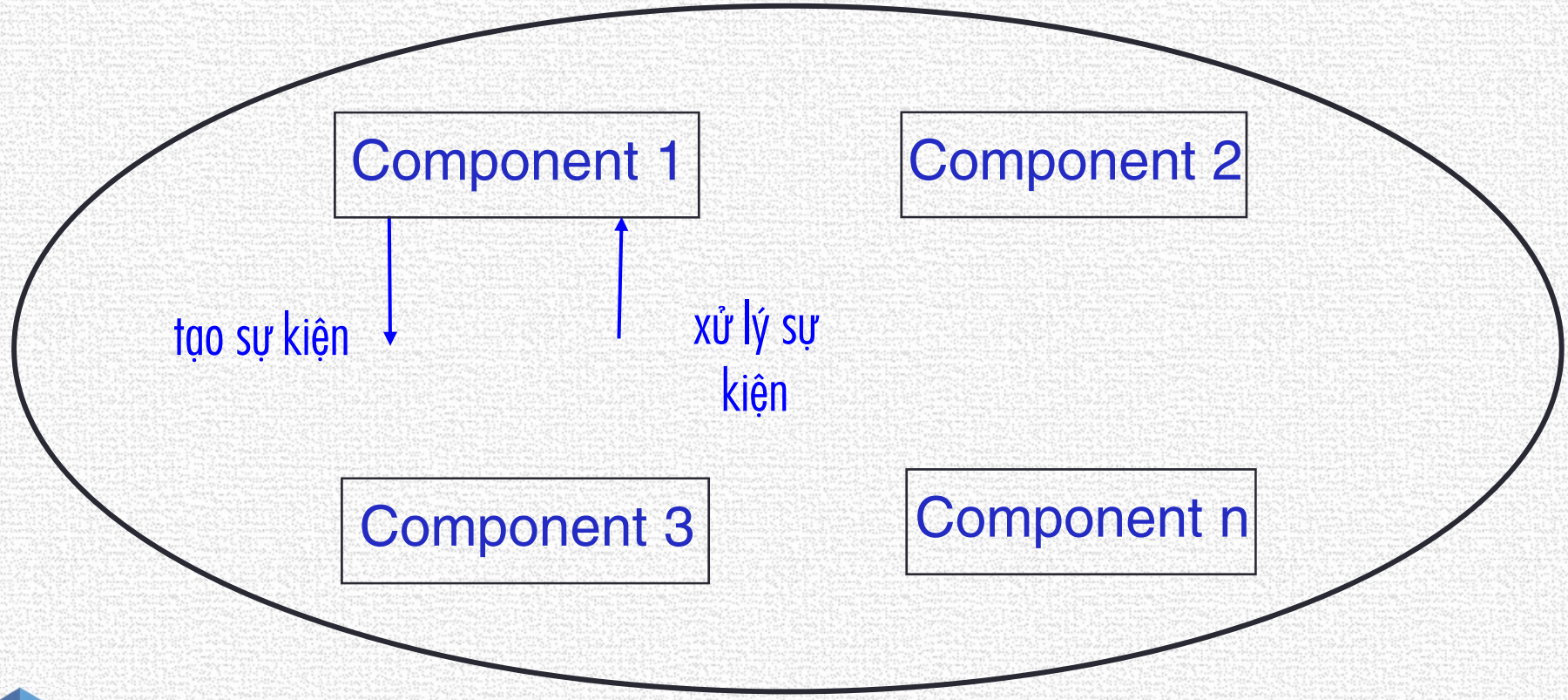
Kiến trúc hướng đối tượng (Objects based Architecture)

- **Tình huống nên dùng :** bất kỳ hệ thống phần mềm phức tạp nào.
- **Khuyết điểm :** là mẫu kiến trúc có độ tổng quát cao nên khi hiện thực ta phải tốn nhiều chi phí để vận dụng nó.

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc dựa trên sự kiện (Event-based Architecture)

- **Đặc tả :** Hệ thống phần mềm gồm 1 tập các thành phần độc lập được ghép nối lỏng lẻo dựa trên việc tạo/xử lý sự kiện.



8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc dựa trên sự kiện (Event-based Architecture)

- **Emitter** : là phần tử tạo và phát tán 1 hay nhiều sự kiện.
- **Handler** : là phần tử muốn xử lý sự kiện, nó đăng ký thủ tục xử lý sự kiện vào danh sách xử lý của sự kiện tương ứng. Khi sự kiện xảy ra, nó được kích hoạt chạy (bởi module quản lý sự kiện). Lưu ý thứ tự chạy các thủ tục xử lý sự kiện cho 1 sự kiện xác định là không xác định.
- **Event chanel** : là phương tiện truyền dẫn sự kiện từ emitter tới handler.
- Lưu ý là phần tử nào trong hệ thống đều có thể là event emitter lẫn event handler. Có thể có các dạng tương tác khác giữa các phần tử như gọi thủ tục, truy xuất dữ liệu...

8.3 Các mẫu kiến trúc phổ dụng

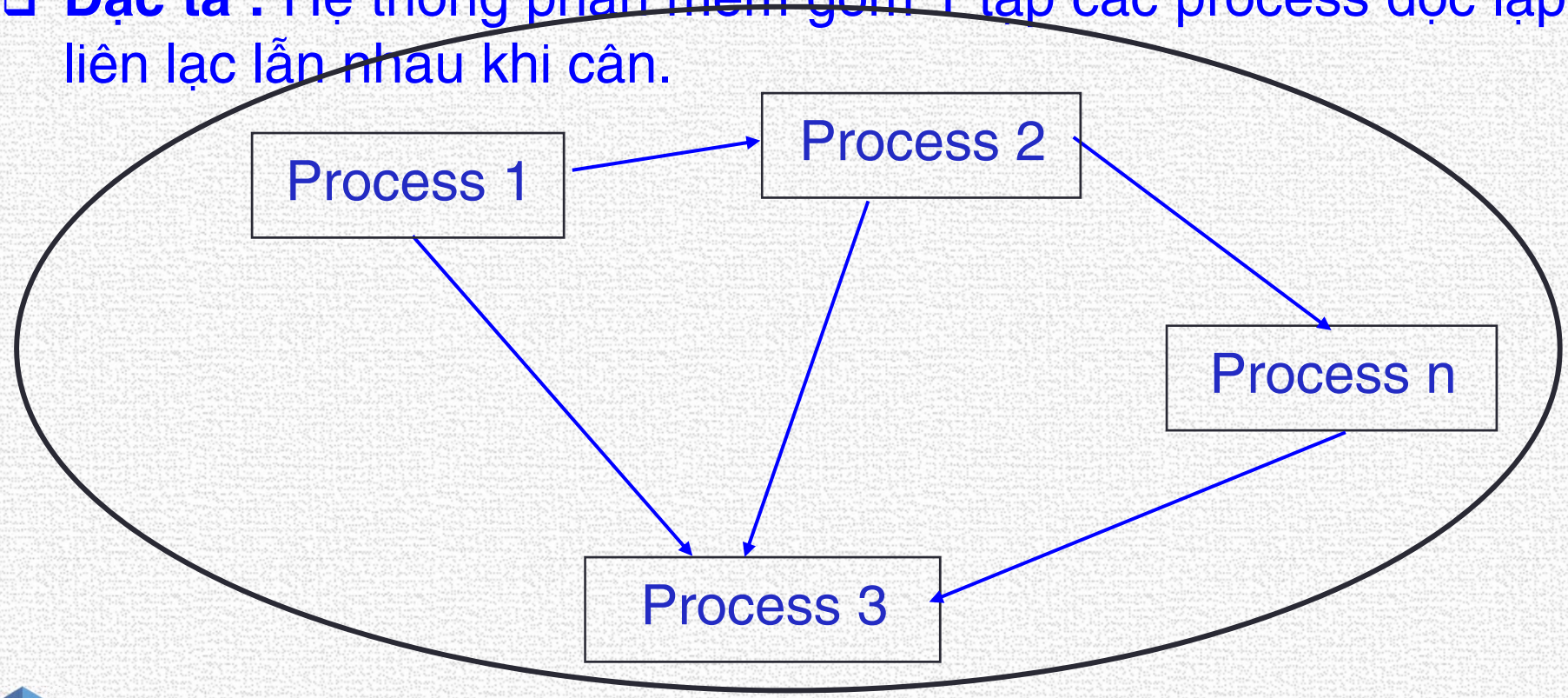
Kiến trúc dựa trên sự kiện (Event-based Architecture)

- **Tình huống nên dùng** : trong các hệ thống :
  - tương tác bẩm sinh như giao diện người dùng, mạng máy tính.
  - trả kết quả về từ việc thi hành bất đồng bộ (thread).
  - gia tăng khả năng việc dùng lại từng thành phần.
  - cải tiến hệ thống dễ dàng : thay đổi thành phần này bằng thành phần khác.
- **Khuyết điểm** : khó kiểm thử đơn vị từng thành phần, không biết sự kiện do mình gởi có được xử lý hay không và ai xử lý...

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc các process liên lạc nhau (Communication process Architecture)

- **Đặc tả :** Hệ thống phần mềm gồm 1 tập các process độc lập liên lạc lẫn nhau khi cần.



8.3 Các mẫu kiến trúc phổ dụng

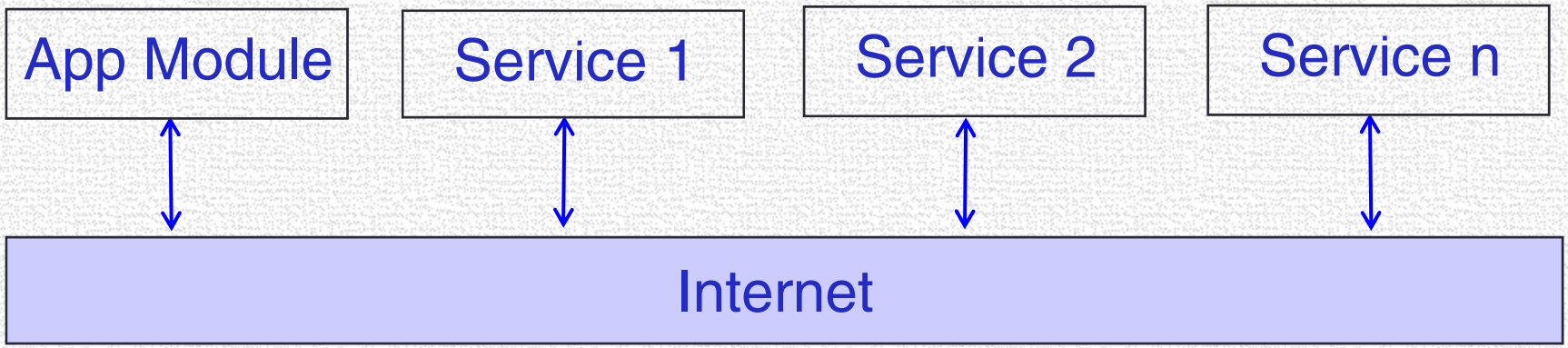
Kiến trúc các process liên lạc nhau (Communication process Architecture)

- **Process** : là nguyên tử cấu thành phần mềm, nó là 1 phần mềm chạy độc lập, mỗi process thực hiện 1 chức năng xác định.
- **Connector** : phương tiện tương tác (truyền thông báo) giữa các process :
  - điểm tới điểm
  - đồng bộ hay bất đồng bộ
  - RPC và các giao thức khác có thể được đặt trên cấp các process này.

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc hướng dịch vụ (Service-Oriented Architecture)

- **Đặc tả** : Cho phép tạo phần mềm bằng cách sử dụng các dịch vụ sẵn có.



8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc hướng dịch vụ (Service-Oriented Architecture)

- **Service** : phần tử cung cấp 1 số chức năng đa dụng nào đó và thường đã có sẵn. Các nguyên tắc chính yếu của kiến trúc hướng dịch vụ là :
  - Services are autonomous : tự trị
  - Services are distributable : dễ dàng phân phối
  - Services are loosely coupled : nối ghép rất lỏng
  - Services share schema and contract, not class : dùng chung mô hình và hợp đồng chứ không phải là class cụ thể
  - Compatibility is based on policy : độ tương thích phụ thuộc vào chính sách cụ thể.

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc hướng dịch vụ (Service-Oriented Architecture)

- **Ưu điểm của kiến trúc hướng dịch vụ :**
  - Domain alignment
  - Abstraction : độ trừu tượng cao
  - Discoverability : dễ dàng khám phá
  - Interoperability : dễ dàng làm việc chung
  - Rationalization : hợp lý hóa

8.3 Các mẫu kiến trúc phổ dụng

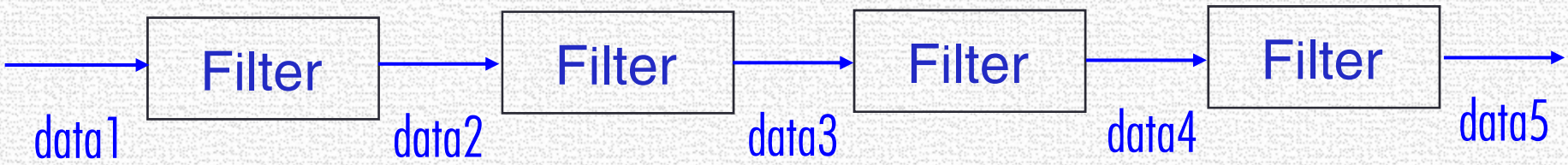
Kiến trúc hướng dịch vụ (Service-Oriented Architecture)

- **Tình huống nên dùng** : bất kỳ hệ thống phần mềm phức tạp nào mà muốn chạy trên nền Internet.
- **Khuyết điểm** : độ hiệu quả phụ thuộc vào cơ sở hạ tầng mạn và máy chạy service.

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc lô tuần tự (Batch Sequential)

- **Đặc tả** : Chương trình gồm n phần mềm độc lập và được chạy theo cơ chế tuần tự : phần mềm i chạy trước, khi xong rồi thì truyền kết quả cho phần mềm thứ i+1... Mỗi phần mềm i trong lô được gọi là filter, nó xử lý dữ liệu đầu vào theo định dạng xác định rồi tạo kết quả đầu ra theo định dạng xác định.



8.3 Các mẫu kiến trúc phổ dụng

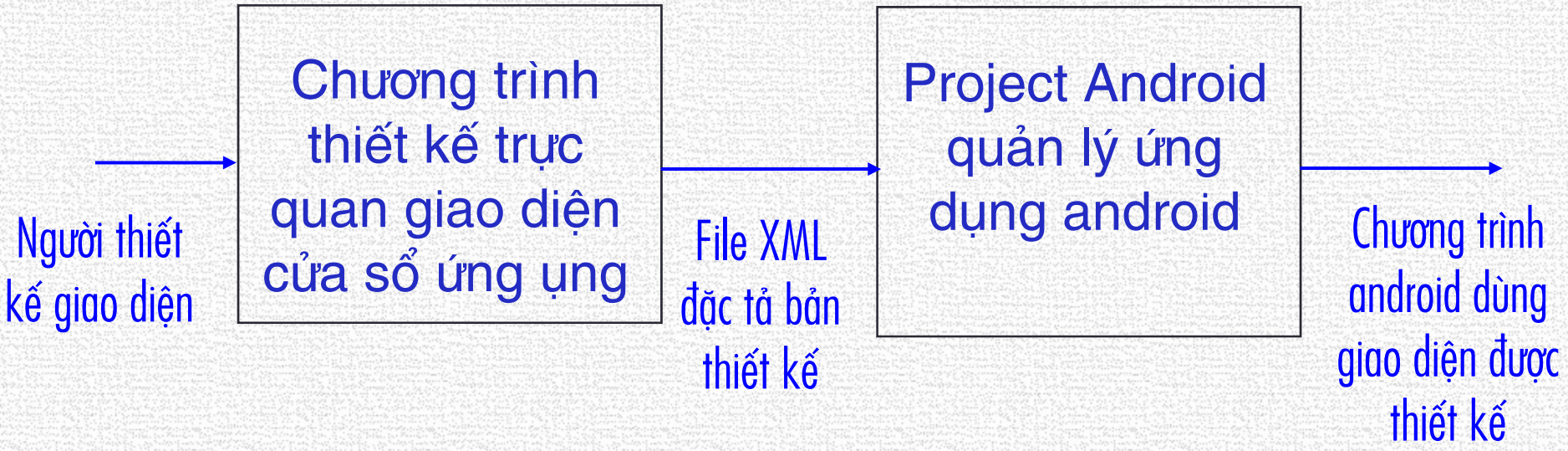
Kiến trúc lô tuần tự (Batch Sequential)

- **Tình huống nên dùng** : trong các ứng dụng xử lý dữ liệu mà dữ liệu nhập cần được xử lý bởi nhiều công đoạn khác nhau và có tính độc lập cao trước khi tạo ra kết quả cuối cùng.
- **Ưu điểm** : dễ dàng thay đổi/bảo trì/dùng lại từng filter của hệ thống, phù hợp với nhiều hoạt động nghiệp vụ, dễ dàng nâng cấp bằng cách thêm filter mới.
- **Khuyết điểm** : 2 filter kề nhau cần tuân thủ định dạng dữ liệu chung.

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc lô tuần tự (Batch Sequential)

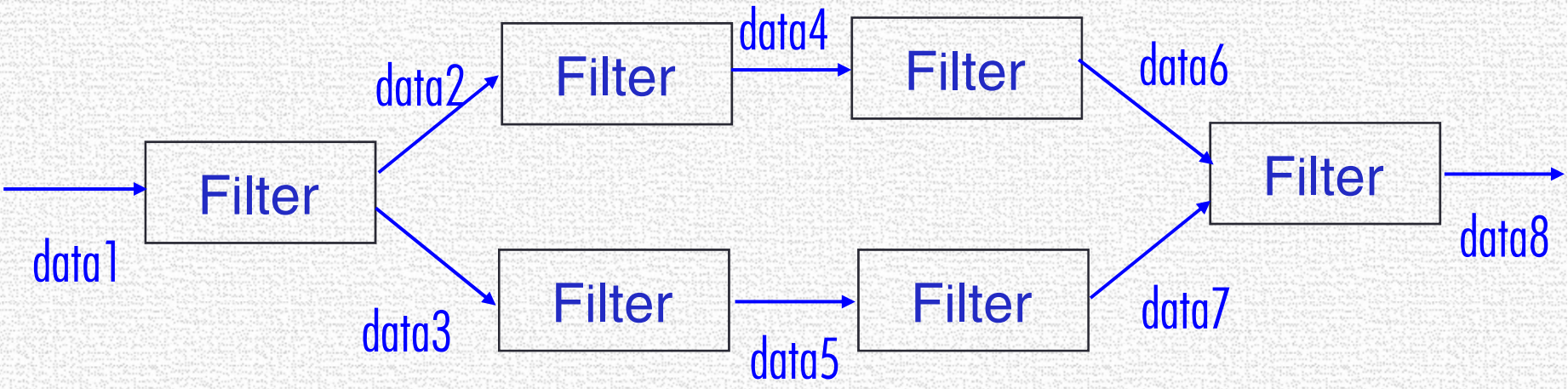
- **Thí dụ** : Thiết kế trực quan cửa sổ giao diện và dùng nó trong phần mềm android.



8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc đường ống và lọc (Pipe and filter Architecture)

- **Đặc tả** : Nới rộng kiến trúc lô tuần tự lên tầm cao mới :
  - Các filter không nhất thiết là phần mềm độc lập lẫn nhau, chúng có thể là các thread chạy trong 1 chương trình.
  - Có thể có nhiều ống con trong từng đoạn xử lý.



8.3 Các mẫu kiến trúc phổ dụng

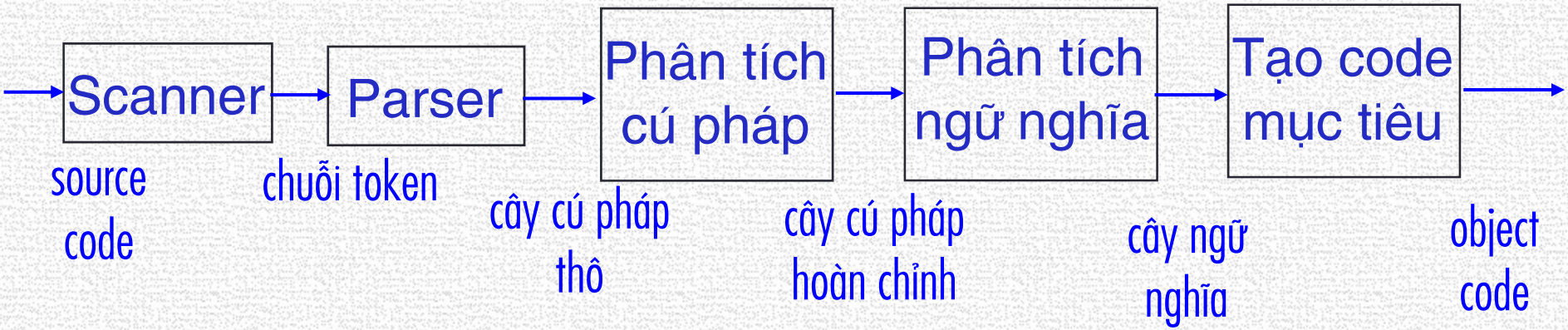
Kiến trúc đường ống và lọc (Pipe and filter Architecture)

- **Tình huống nên dùng** : trong các ứng dụng xử lý dữ liệu mà dữ liệu nhập cần được xử lý bởi nhiều công đoạn khác nhau và có tính độc lập cao trước khi tạo ra kết quả cuối cùng.
- **Ưu điểm** : dễ dàng thay đổi/bảo trì/dùng lại từng filter của hệ thống, phù hợp với nhiều hoạt động nghiệp vụ, dễ dàng nâng cấp bằng cách thêm filter mới, hiệu quả cao hơn kiến trúc lô tuần tự.
- **Khuyết điểm** : 2 filter kề nhau cần tuân thủ định dạng dữ liệu chung.

8.3 Các mẫu kiến trúc phổ dụng

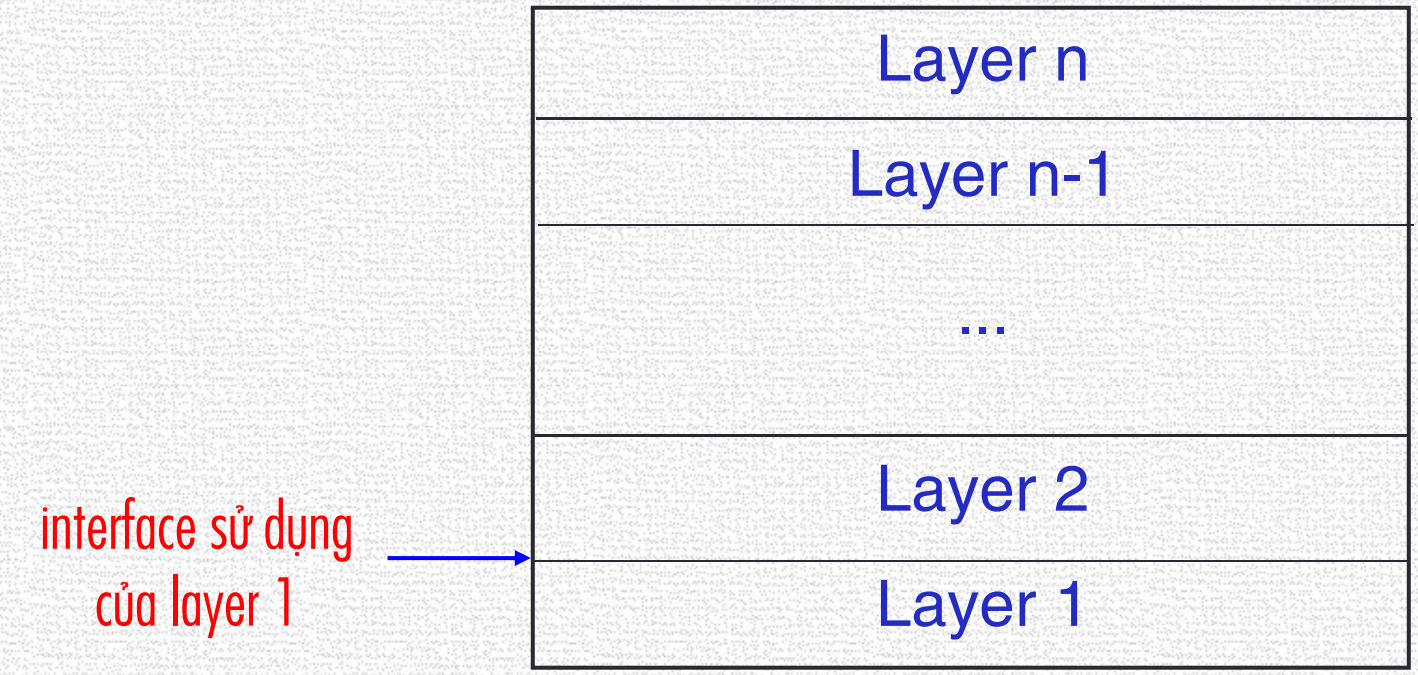
Kiến trúc đường ống và lọc (Pipe and filter Architecture)

- **Thí dụ** : Chương trình dịch ngôn ngữ



Các mẫu kiến trúc phần mềm phổ dụng

- **Kiến trúc nhiều cấp** (Layered architecture)
- **Đặc tả** : Phần mềm gồm nhiều cấp (layer) chức năng dạng chồng lên nhau, mỗi cấp có chức năng cụ thể, rõ ràng và cung cấp các dịch vụ cho cấp ngay trên mình. Cấp thấp nhất chứa các dịch vụ cơ bản nhất.



8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc nhiều cấp (Layered architecture)

- **Tình huống nên dùng** : xây dựng thêm khả năng mới trên hệ thống có sẵn, hay khi có nhiều nhóm phát triển khác nhau, mỗi nhóm chịu trách nhiệm về 1 layer chức năng cụ thể, hay khi có yêu cầu bảo mật nhiều cấp.
- **Ưu điểm** : cho phép hiệu chỉnh bên trong layer bất kỳ sao cho interface không đổi. Có thể giải quyết 1 chức năng nào đó (xác nhận user) ở nhiều cấp theo cách thức tăng dần.
- **Khuyết điểm** : khó tách bạch chức năng của từng cấp, layer trên khó tương tác với layer phía dưới nó nhưng không liền kề. Hiệu quả giảm sút khi nhiều layer phải tương tác nhau để giải quyết 1 chức năng nào đó.

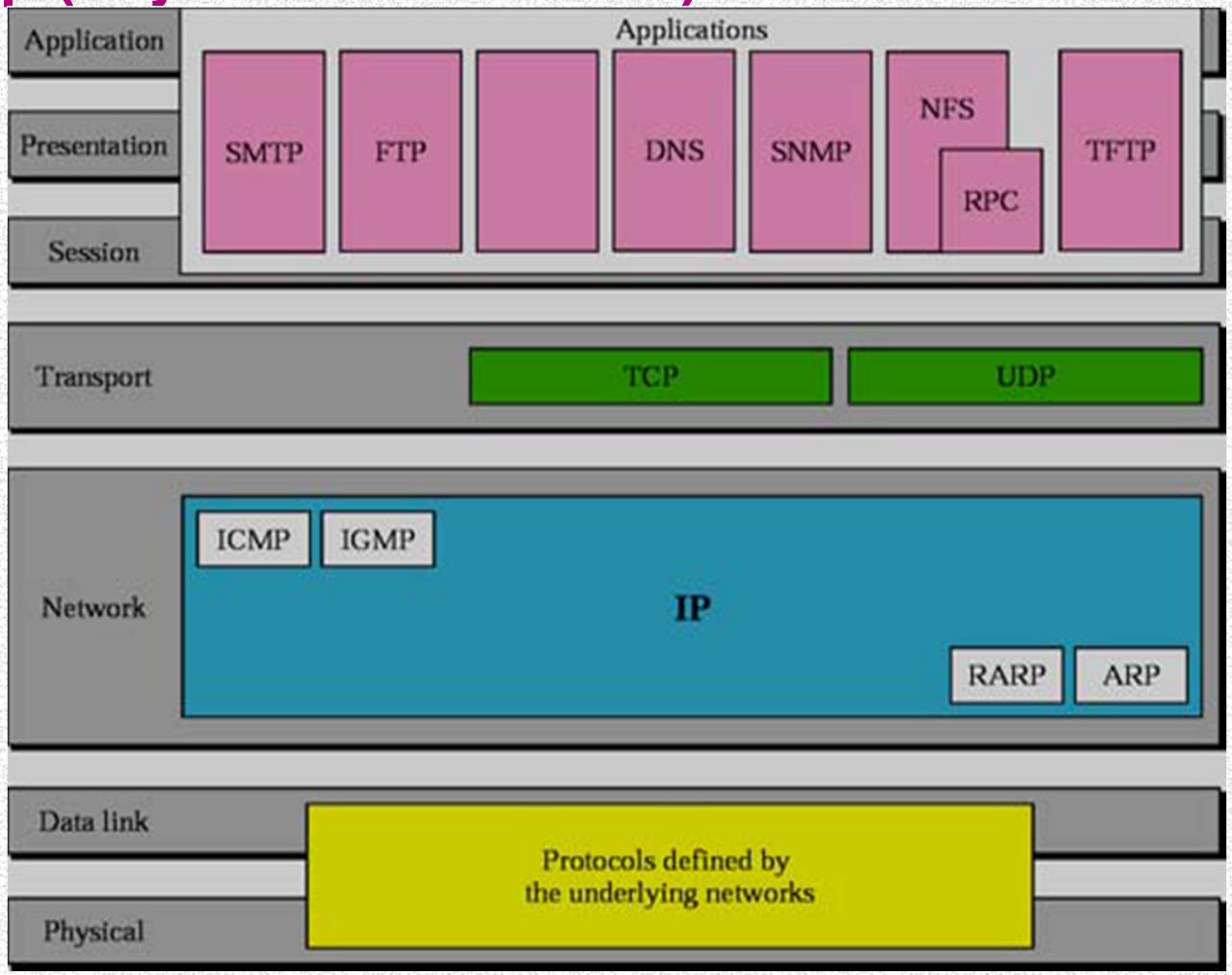
Các mẫu kiến trúc phần mềm phổ dụng

Yêu cầu phi chức năng	Đánh giá
Khả năng thích ứng (agility)	Thấp
Khả năng triển khai (Deployment)	Thấp
Khả năng kiểm thử (Testability)	Cao
Hiệu năng (Performance)	Thấp
Khả năng mở rộng (Scalability)	Thấp
Khả năng phát triển (development)	Cao

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc nhiều cấp (Layered architecture)

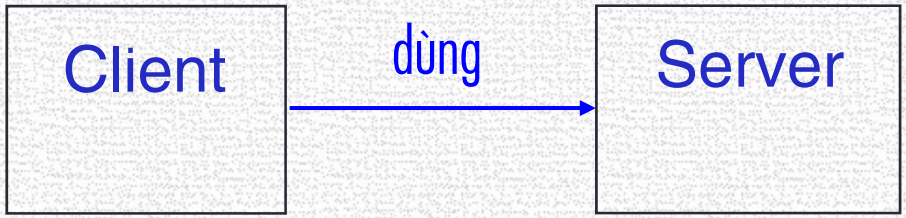
- Thí dụ : Kiến trúc mạng OSI và kiến trúc mạng internet.



8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc client-server (client-server Architecture)

- **Đặc tả** : Hệ thống gồm 2 loại phần tử chức năng : server cung cấp 1 số dịch vụ, client là phần tử sử dụng dịch vụ bằng cách truy xuất đến server tương ứng.



8.3 Các mẫu kiến trúc phổ dụng

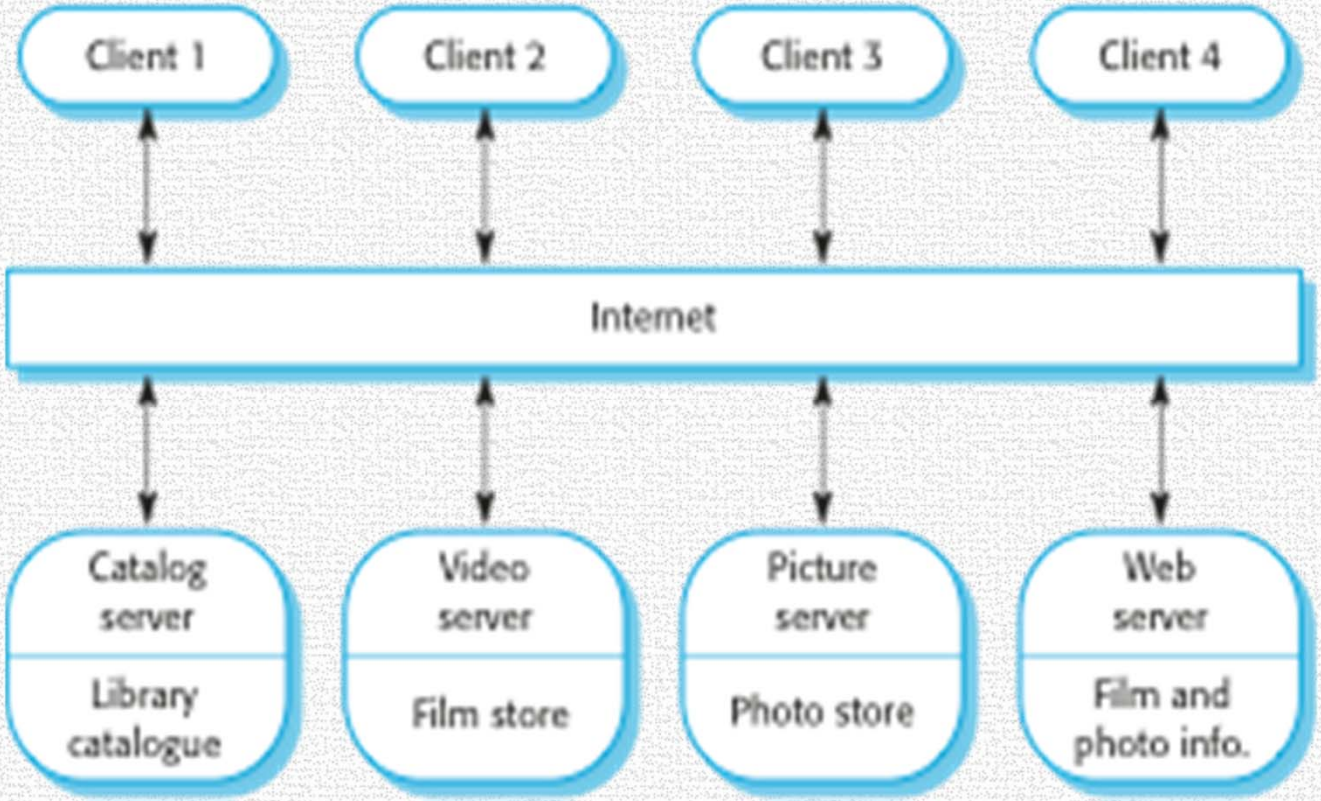
Kiến trúc client-server (client-server Architecture)

- **Tình huống nên dùng** : khi database dùng chung từ nhiều vị trí khác nhau hay khi tải hệ thống thay đổi động (nhân bản server thành nhiều phần tử).
- **Ưu điểm** : server có thể phân tán tự do trên mạng.
- **Khuyết điểm** : độ hiệu quả phụ thuộc vào mạng và hệ thống nên khó lường trước. Nếu các server được quản lý bởi các tổ chức khác nhau thì có vấn đề về quản lý chúng.

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc client-server (client-server Architecture)

- **Thí dụ** : Hệ thống quản lý phim ảnh dùng mô hình client-server



8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc 3 đối tác (3-tiers Architecture)

- **Đặc tả** : Sự cải tiến của kiến trúc client-server. Hệ thống gồm 3 loại phần tử chức năng : client, server, và server của server.

8.3 Các mẫu kiến trúc phổ dụng

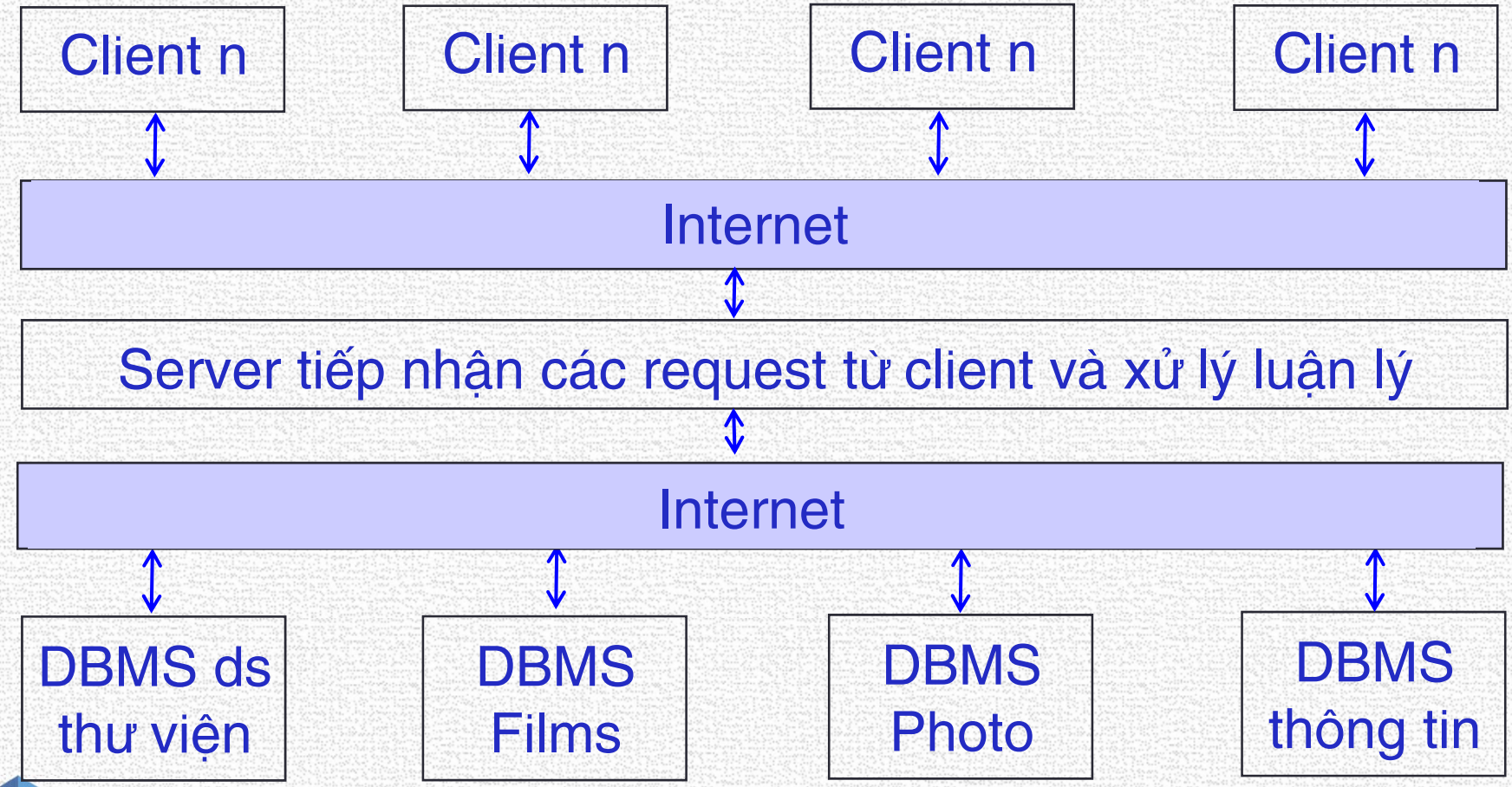
Kiến trúc 3 đối tác (3-tiers Architecture)

- **Tình huống nên dùng** : khi database dùng chung từ nhiều vị trí khác nhau hay khi tải hệ thống thay đổi động (nhân bản server thành nhiều phần tử).
- **Ưu điểm** : server có thể phân tán tự do trên mạng.
- **Khuyết điểm** : độ hiệu quả phụ thuộc vào mạng và hệ thống nên khó lường trước. Nếu các server được quản lý bởi các tổ chức khác nhau thì có vấn đề về quản lý chúng.

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc 3 đối tác (3-tiers Architecture)

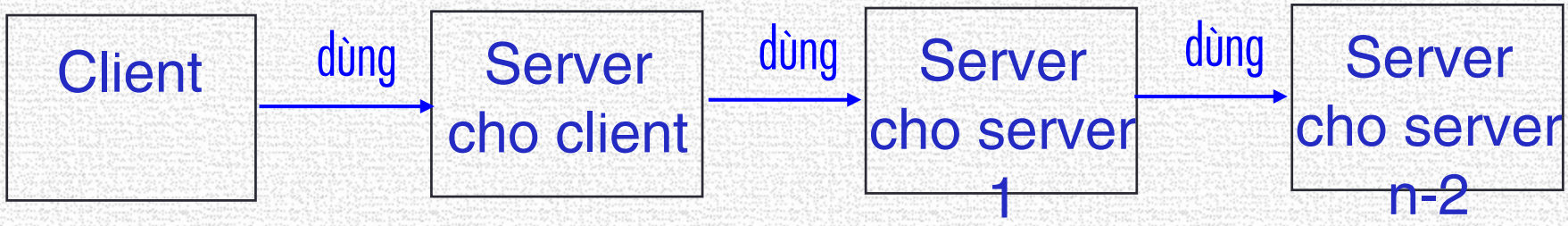
- **Thí dụ** : Hệ thống quản lý phim ảnh dùng mô hình 3-tiers



8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc n đối tác (n-tiers Architecture)

- **Đặc tả** : Sự tổng quát của kiến trúc 3-tiers. Hệ thống gồm n loại phần tử chức năng : client, server, và server của server,...



8.3 Các mẫu kiến trúc phổ dụng

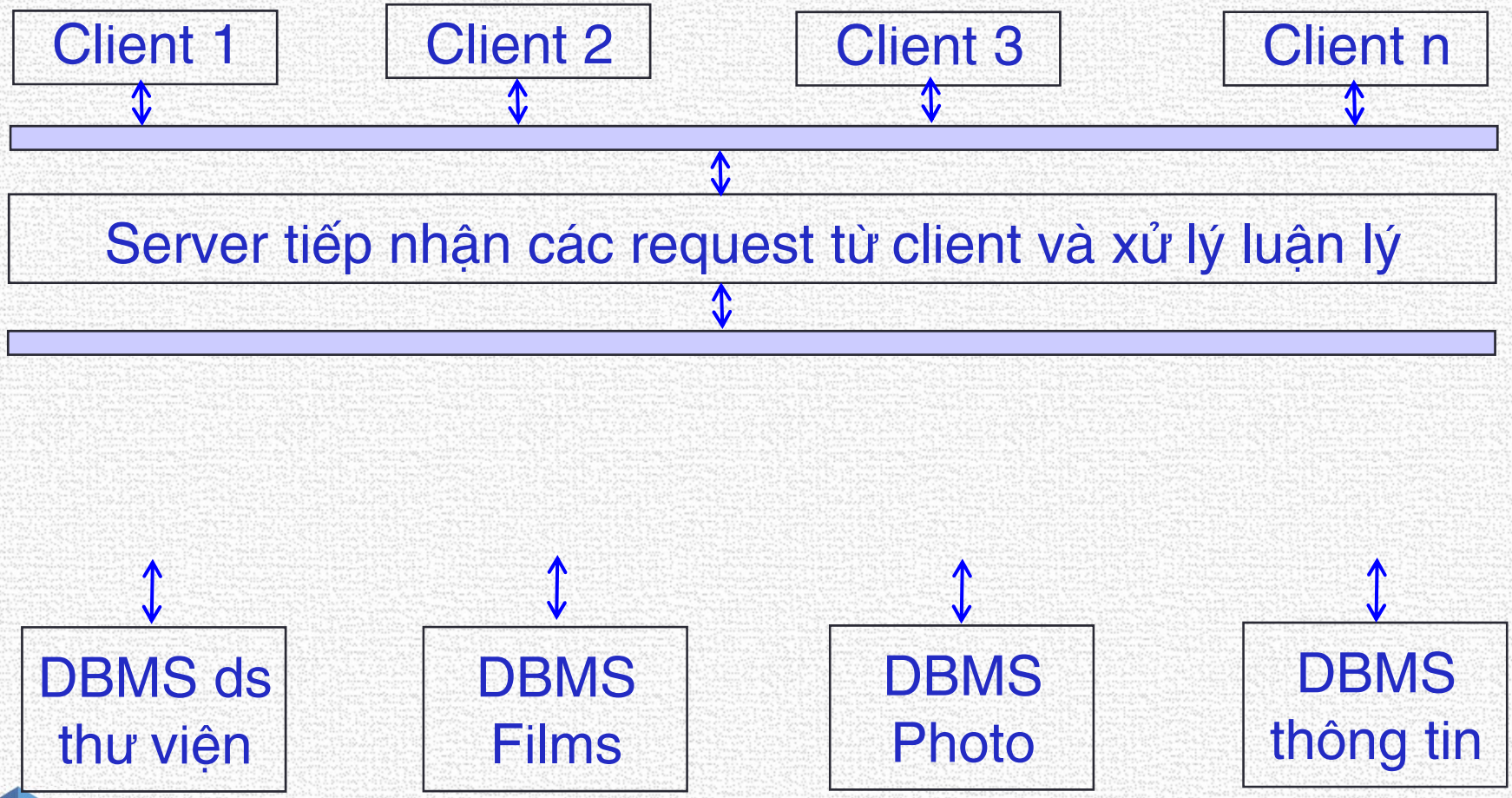
Kiến trúc n đối tác (n-tiers Architecture)

- **Tình huống nên dùng** : khi database dùng chung từ nhiều vị trí khác nhau hay khi tải hệ thống thay đổi động (nhân bản server thành nhiều phần tử).
- **Ưu điểm** : server có thể phân tán tự do trên mạng.
- **Khuyết điểm** : độ hiệu quả phụ thuộc vào mạng và hệ thống nên khó lường trước. Nếu các server được quản lý bởi các tổ chức khác nhau thì có vấn đề về quản lý chúng.

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc n đối tác (n-tiers Architecture)

- **Thí dụ** : Hệ thống quản lý phim ảnh dùng mô hình n-tiers



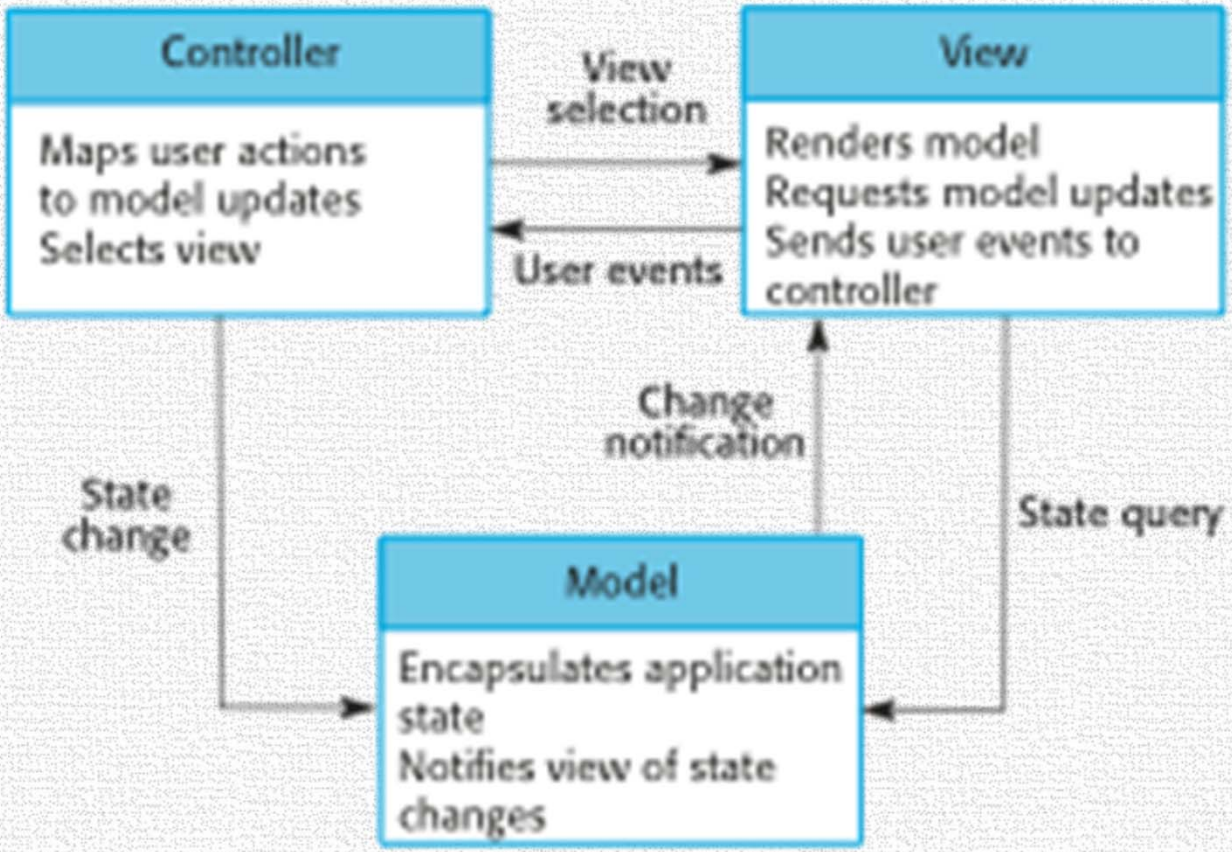
8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc MVC (Model-View-Controller)

- **Đặc tả** : Hệ thống gồm 3 thành phần luận lý tương tác lẫn nhau :
  - Model quản lý dữ liệu và các tác vụ liên quan đến dữ liệu này.
  - View định nghĩa và quản lý cách thức dữ liệu được trình bày cho user.
  - Controller quản lý các tương tác với user như ấn phím, click chuột… và gởi thông tin tương tác này tới View và/hoặc Model.

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc MVC (Model-View-Controller)



8.3 Các mẫu kiến trúc phổ dụng

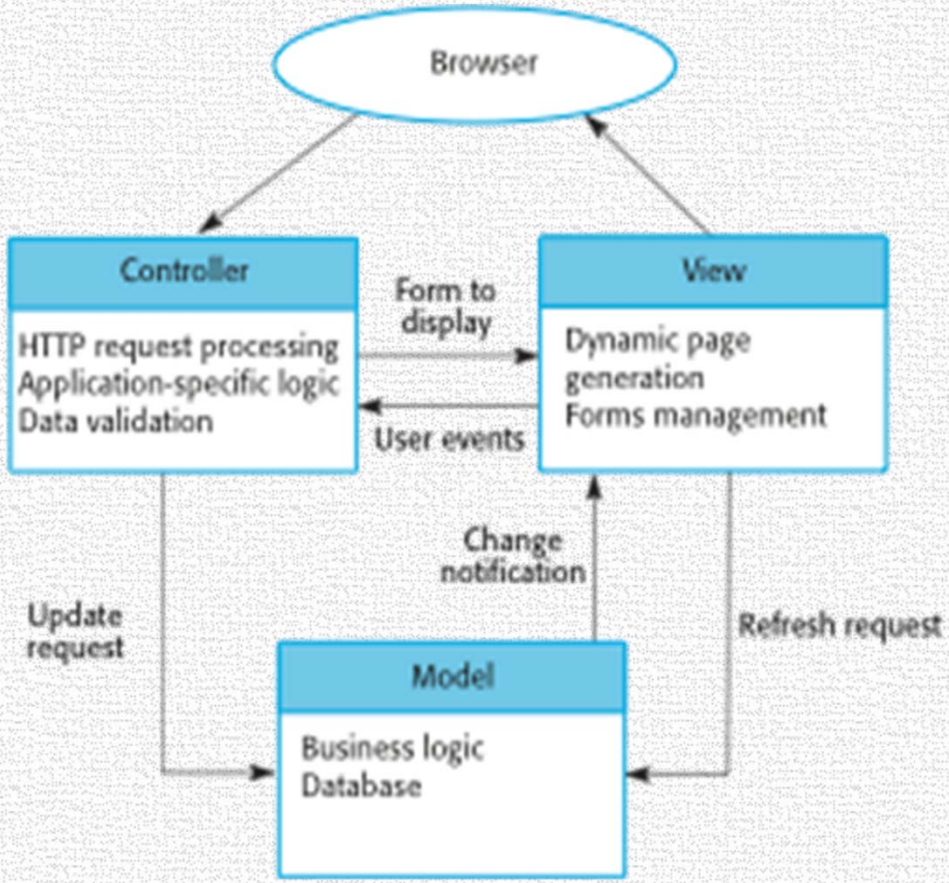
Kiến trúc MVC (Model-View-Controller)

- **Tình huống nên dùng** : Hệ thống có nhiều cách để view và tương tác với dữ liệu, hoặc ta chưa biết trước các yêu cầu tương lai về sự tương tác và biểu diễn dữ liệu của chương trình.
- **Ưu điểm** : cho phép dữ liệu thay đổi độc lập với cách thức thể hiện nó và ngược lại.
- **Khuyết điểm** : có thể cần nhiều code hơn và code có thể phức tạp hơn khi mô hình dữ liệu và sự tương tác chỉ ở mức độ đơn giản.

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc MVC (Model-View-Controller)

- **Thí dụ** : Hệ thống web dùng kiến trúc MVC :



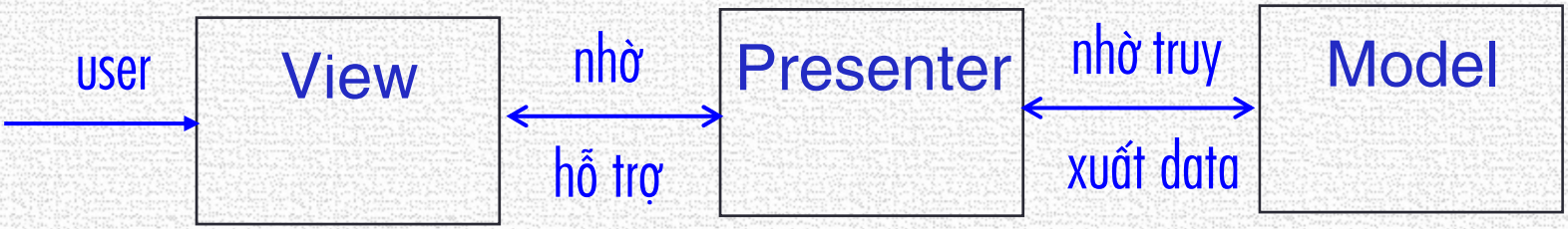
8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc MVP (Model-View-Presenter)

- **Đặc tả** : Hệ thống gồm 3 thành phần luận lý tương tác lẫn nhau :
  - Model quản lý dữ liệu và các tác vụ liên quan đến dữ liệu này.
  - View định nghĩa và quản lý cách thức dữ liệu được trình bày cho user, quản lý các tương tác với user như ấn phím, click chuột… và gởi thông tin tương tác này tới Presenter để nhờ xử lý chi li hơn.
  - Presenter xử lý chi li về luận lý nghiệp vu, nhờ Model truy xuất dữ liệu khi cần

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc MVP (Model-View-Presenter)



8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc MVP (Model-View-Presenter)

- **Tình huống nên dùng** : Hệ thống có nhiều cách để view và tương tác với dữ liệu, hoặc ta chưa biết trước các yêu cầu tương lai về sự tương tác và biểu diễn dữ liệu của chương trình.
- **Ưu điểm** : cho phép dữ liệu thay đổi độc lập với cách thức thể hiện nó và ngược lại.
- **Khuyết điểm** : có thể cần nhiều code hơn và code có thể phức tạp hơn khi mô hình dữ liệu và sự tương tác chỉ ở mức độ đơn giản.

8.3 Các mẫu kiến trúc phổ dụng

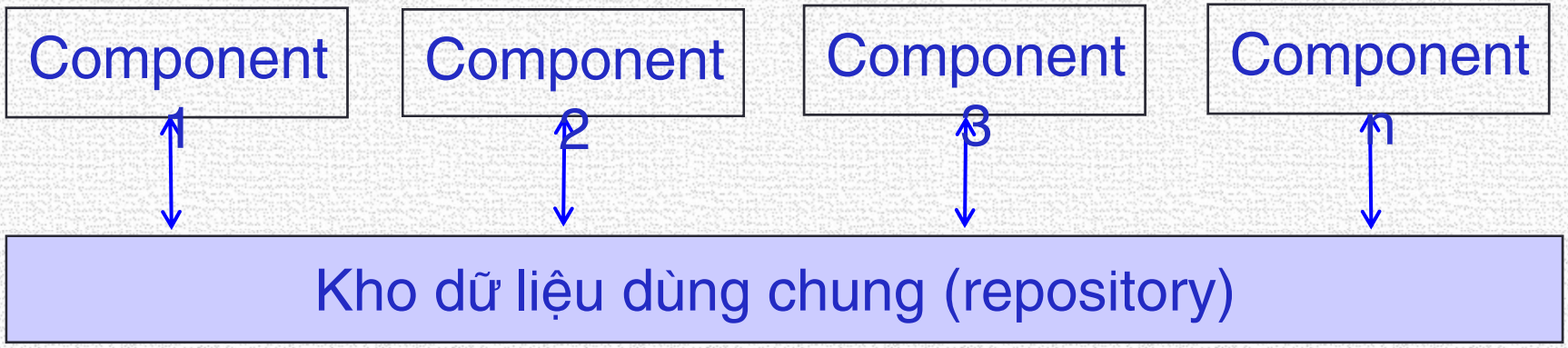
Kiến trúc MVP (Model-View-Presenter)

- **Thí dụ** : Hệ thống web dùng kiến trúc MVP :

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc kho (Repository Architecture)

- **Đặc tả** : Tất cả dữ liệu của hệ thống được quản lý trong 1 kho chứa tập trung, mọi thành phần chức năng của hệ thống đều có thể truy xuất kho chứa này. Các thành phần không tương tác trực tiếp với nhau, chỉ thông qua kho chứa tập trung.



8.3 Các mẫu kiến trúc phổ dụng

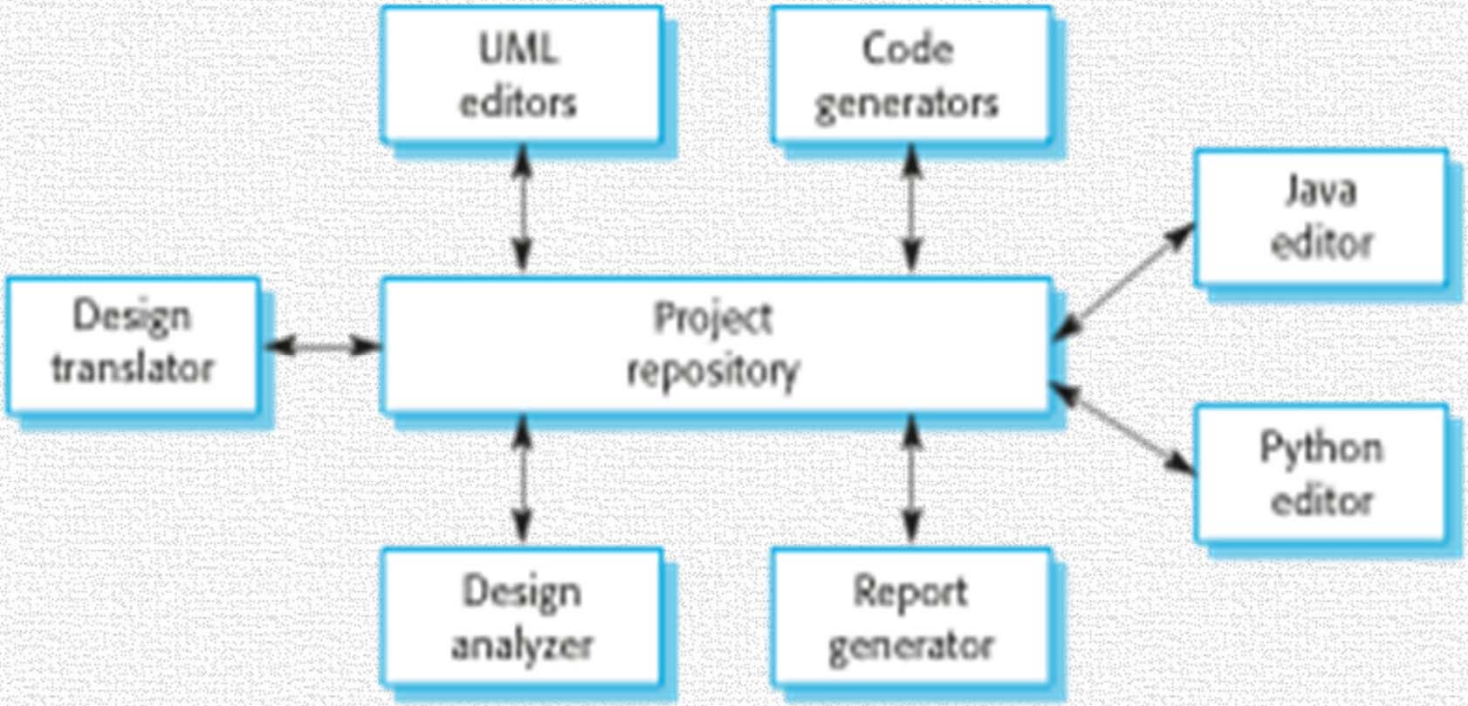
Kiến trúc kho (Repository Architecture)

- **Tình huống nên dùng** : khi hệ thống tạo và chứa 1 lượng rất lớn thông tin trong thời gian dài, hay trong các hệ thống dựa vào dữ liệu, ở đó việc chứa thông tin vào kho sẽ kích hoạt 1 tool hay 1 chức năng hoạt động.
- **Ưu điểm** : các thành phần độc lập nhau, không ai biết gì về ai khác.
- **Khuyết điểm** : kho là điểm yếu nhất, nếu có lỗi sẽ ảnh hưởng toàn bộ các thành phần chức năng. Có vấn đề về truy xuất đồng thời kho, phân tán kho trên nhiều máy cũng khó khăn.

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc kho (Repository Architecture)

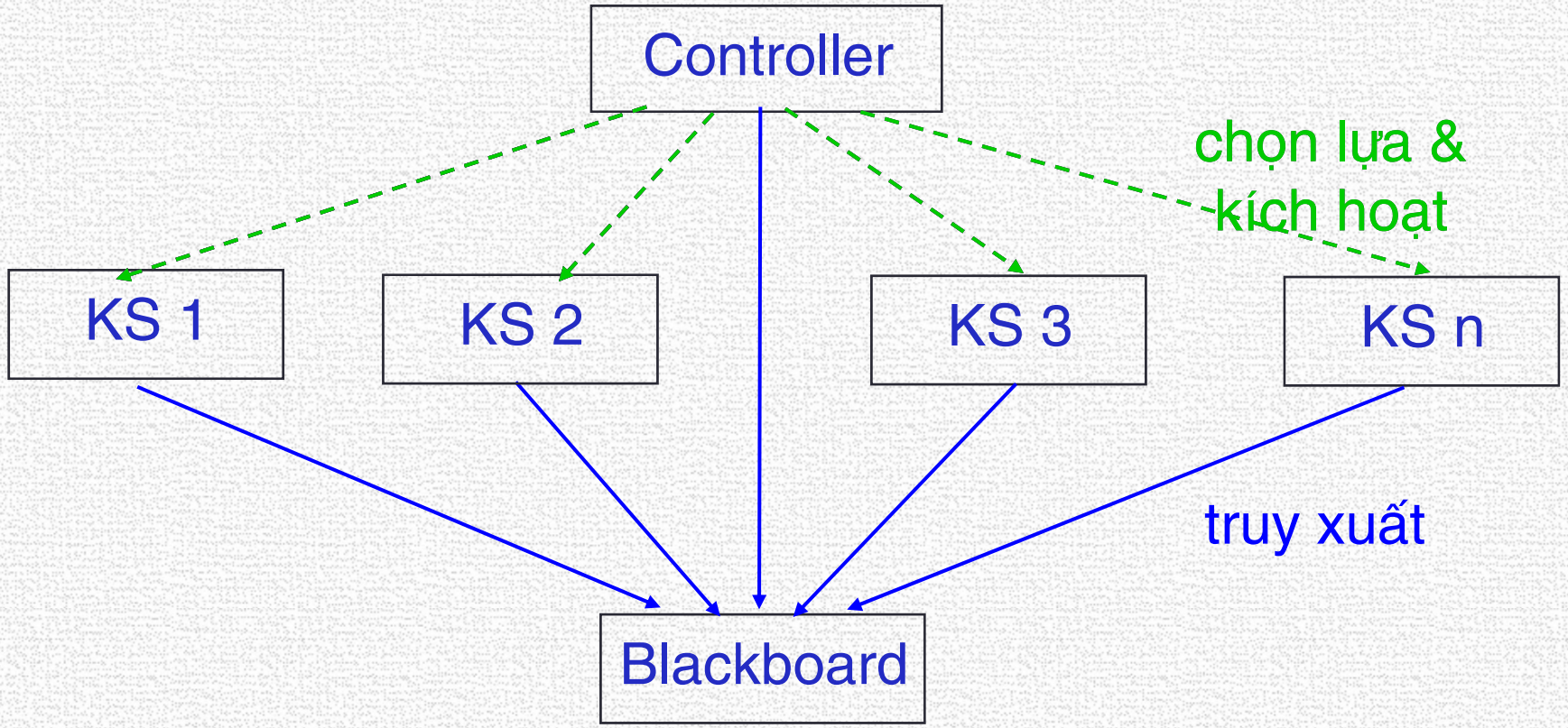
- **Thí dụ** : Môi trường IDE gồm nhiều thành phần dùng chung kho thông tin, mỗi tool tạo thông tin và để trong kho để các tool khác dùng.



8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc bảng đen (Blackboard Architecture)

- **Đặc tả** : Hệ thống phần mềm gồm 3 loại thành phần tương tác nhau như sau :



8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc bảng đen (Blackboard Architecture)

- **Blackboard** : là vùng nhớ toàn cục có cấu trúc của phần mềm, nó chứa các đối tượng của bài toán cần giải quyết, còn được gọi là các nút, chúng được tổ chức dạng phân cấp.
- **Knowledge sources** : là những module chức năng chuyên dụng có cách biểu diễn riêng biệt. Mỗi KS được đặc trưng bởi 1 tập các điều kiện kích hoạt xác định và đoạn code xử lý dữ liệu từ blackboard rồi đóng góp kết quả vào quá trình giải quyết bài toán.
- **Control** : là phần tử điều khiển chung, nó cấu hình, chọn lựa và thi hành các KS. Việc xác định KS nào là dựa vào trạng thái của quá trình giải quyết bài toán (được miêu tả trong blackboard).

8.3 Các mẫu kiến trúc phổ dụng

Kiến trúc bảng đen (Blackboard Architecture)

- **Tình huống nên dùng** : trong các hệ chuyên gia giải quyết vấn đề mà không có cách giải quyết tất định và có thể tin tưởng được.
- **Khuyết điểm** : ?

Kết chương

- **Mô hình** là 1 góc nhìn trừu tượng về phần mềm, nó bỏ qua các chi tiết. Ta thường xây dựng nhiều mô hình khác nhau để miêu tả phần mềm theo nhiều góc nhìn khác nhau.
- **Mô hình ngữ cảnh (hay mô hình use-case)** miêu tả vị trí của phần mềm trong ngữ cảnh xung quanh nó, những actor nào có liên quan đến phần mềm, chúng tương tác với phần mềm thông qua các chức năng gì ? Lược đồ use-case và lược đồ trình tự được dùng chủ yếu để miêu tả các tương tác giữa actor với phần mềm.
- **Mô hình cấu trúc** trình bày tổ chức và kiến trúc phần mềm. Lược đồ class ở cấp phân tích, lược đồ class ở cấp thiết kế, lược đồ thành phần, lược đồ đối tượng, lược đồ triển khai cho chúng ta mô hình cấu trúc phần mềm theo nhiều mức độ trừu tượng khác nhau.

Kết chương

- **Mô hình hành vi** trình bày góc nhìn động của phần mềm khi chạy. Các lược đồ trình tự, lược đồ trạng thái, lược đồ hoạt động, lược đồ cộng tác giúp ta đặc tả được mô hình hành vi theo nhiều góc nhìn khác nhau.
- **Kỹ thuật xây dựng phần mềm được dẫn xuất từ mô hình (Model-driven engineering)** là cách tiếp cận mới, ở đó con người chỉ cần đặc tả các mô hình phần mềm, tiện ích máy tính sẽ dịch tự động các mô hình ra phần mềm chạy được.