Chương 5

CÁC MẪU KIẾN TRÚC PHẦN MỀM PHỔ DỤNG

5.1 Đặc tả phần mềm

5.2 Đặc tả kiến trúc phần mềm

5.3 Các mẫu kiến trúc phổ dụng

5.4 Kết chương

5.1 Đặc tả phần mềm

- Máy tính số là thiết bị tổng quát hóa, nó có thể giải quyết nhiều vấn đề mà con người cần giải quyết.
- Tại từng thời điểm, để nhờ máy tính giải quyết 1 vấn đề nào đó, ta phải lập trình cho máy tính hiểu.
- Qui trình phát triển phần mềm miêu tả các công việc chức năng cần phải thực hiện cùng cách thức, trình tự thực hiện các công việc chức năng này.
- Kết quả của qui trình phát triển phần mềm là bản đặc tả đầy đủ về phần mềm.

5.1 Đặc tả phần mềm

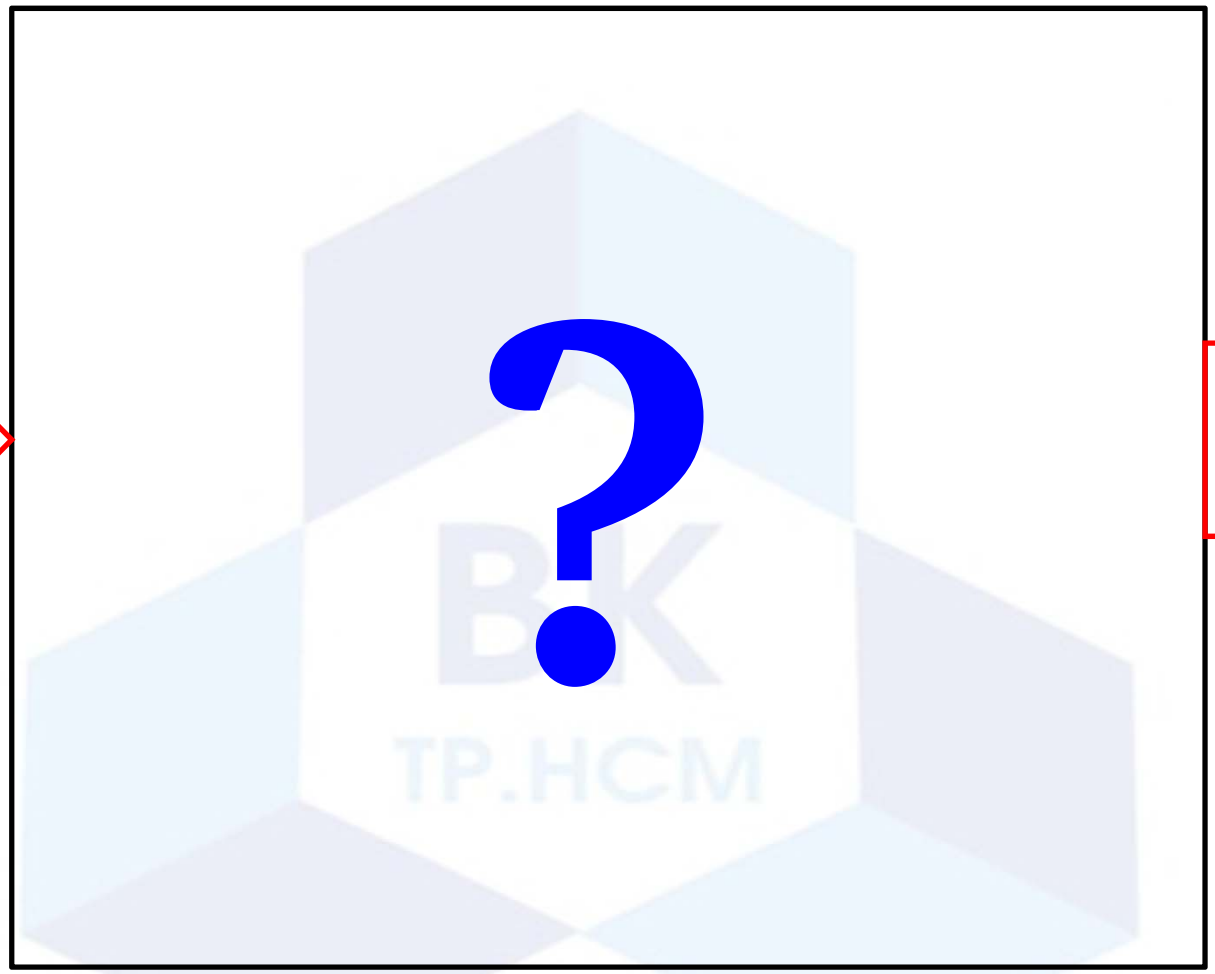
- Đặc tả đầy đủ về phần mềm là đặc tả phần mềm theo nhiều góc nhìn khác nhau :
  - Góc nhìn người dùng : tập các yêu cầu chức năng và phi chức năng của phần mềm.
  - Góc nhìn vĩ mô để người hiểu : kiến trúc phần mềm và phát sơ lược về cách giải quyết từng chức năng.
  - Góc nhìn chi tiết để người hiểu : bản thiết kế chi tiết về phần mềm.
  - Góc nhìn chi tiết để máy hiểu : các file mã nguồn và các file khả thi của chương trình.
  - …

5.2 Đặc tả kiến trúc của hệ thống phần mềm

- Kiến trúc phần mềm cho thấy cấu trúc tổng quát, vĩ mô của phần mềm.
- Kiến trúc phần mềm bao gồm các phần tử sau :
  - các thành phần : định nghĩa địa điểm tính toán, thí dụ filter, database, object, ADT.
  - các mối nối (Connector) : làm trung gian cho tương tác giữa các thành phần. gọi thủ tục, pipe, phát tán sự kiện.
  - các thuộc tính : xác định thông tin cho việc phân tích và xây dựng : chữ ký, điều kiện pre/post, đặc tả RT.

5.2 Đặc tả kiến trúc của hệ thống phần mềm

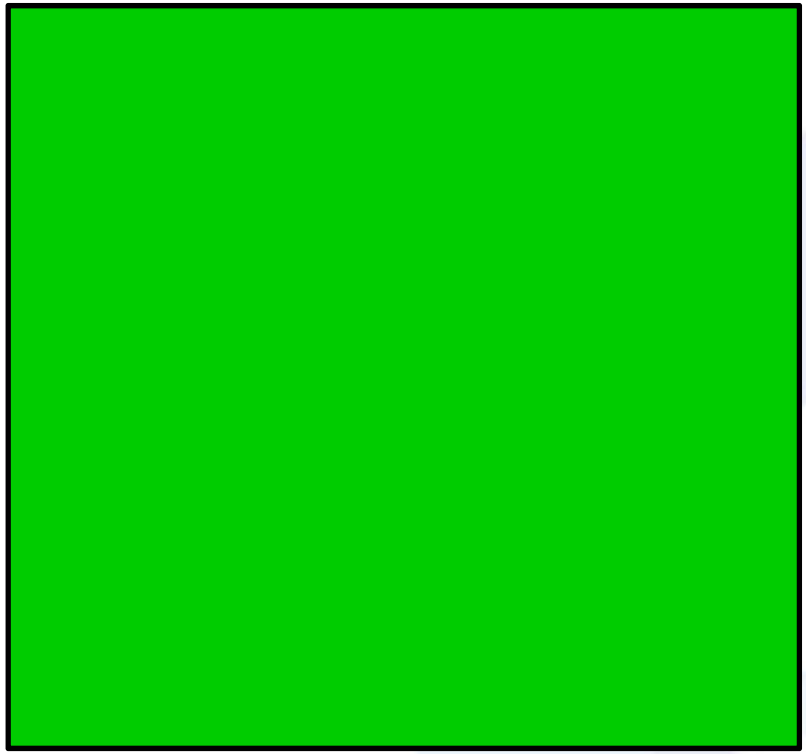
Các yêu cầu chức năng và phi chức năng

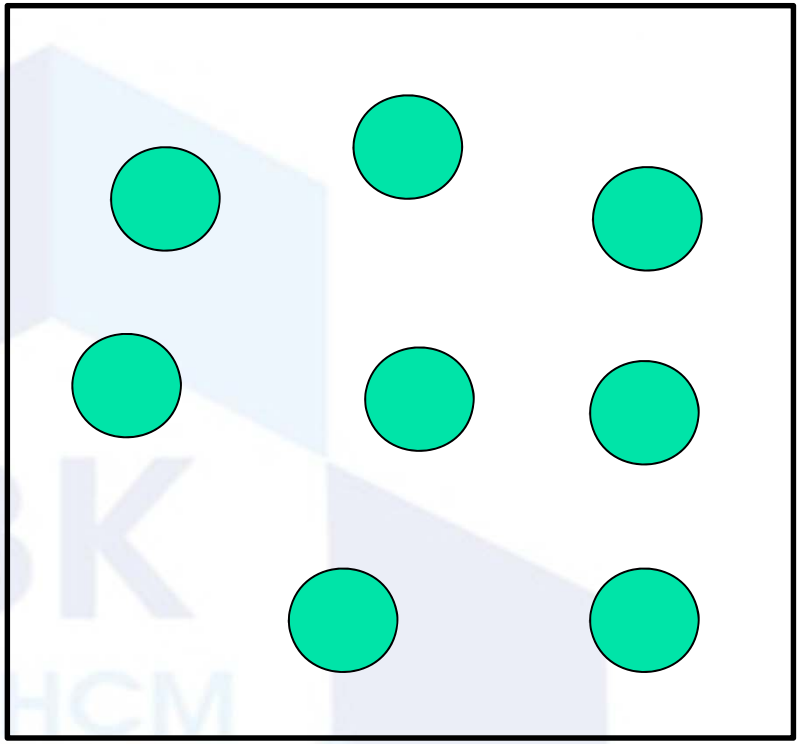


Các chức năng và chất lượng phần mềm

5.2 Đặc tả kiến trúc của hệ thống phần mềm

Chọn cấu trúc nào để xây dựng phần mềm lớn ?

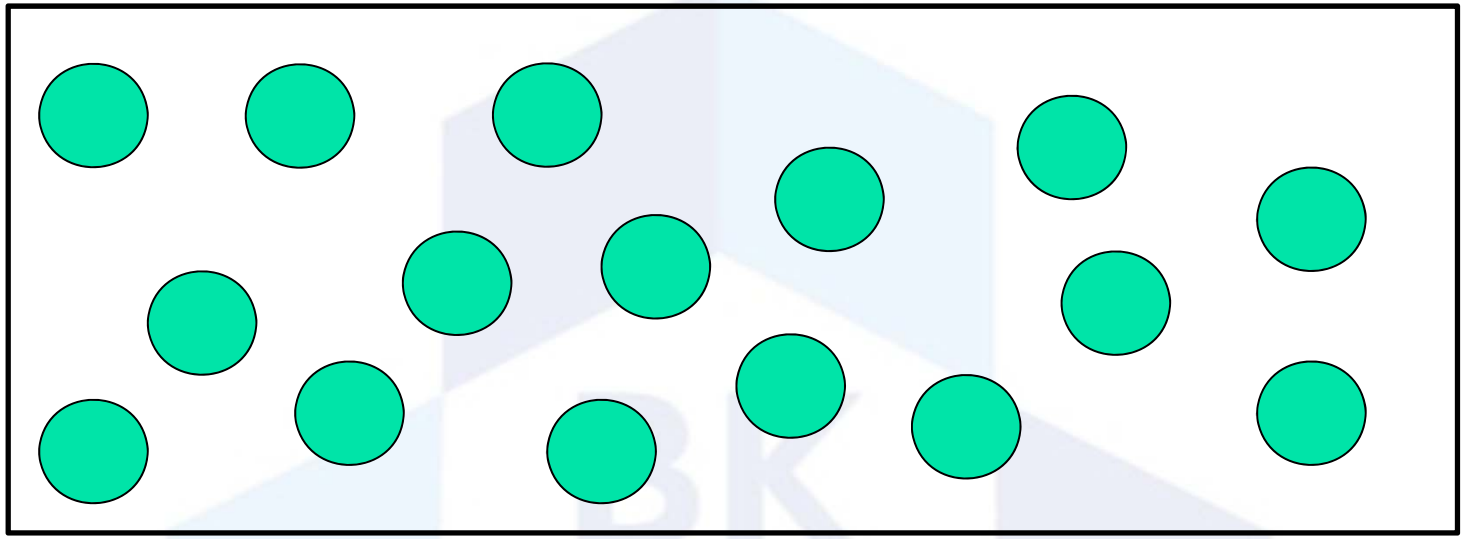




Cấu trúc đơn thể bên trái chắc chắn không phù hợp. Do đó ta sẽ chọn cấu trúc bên phải : phần mềm gồm nhiều phần tử cấu thành, các phần tử này chắc chắn phải có mối quan hệ lẫn nhau.

5.2 Đặc tả kiến trúc của hệ thống phần mềm

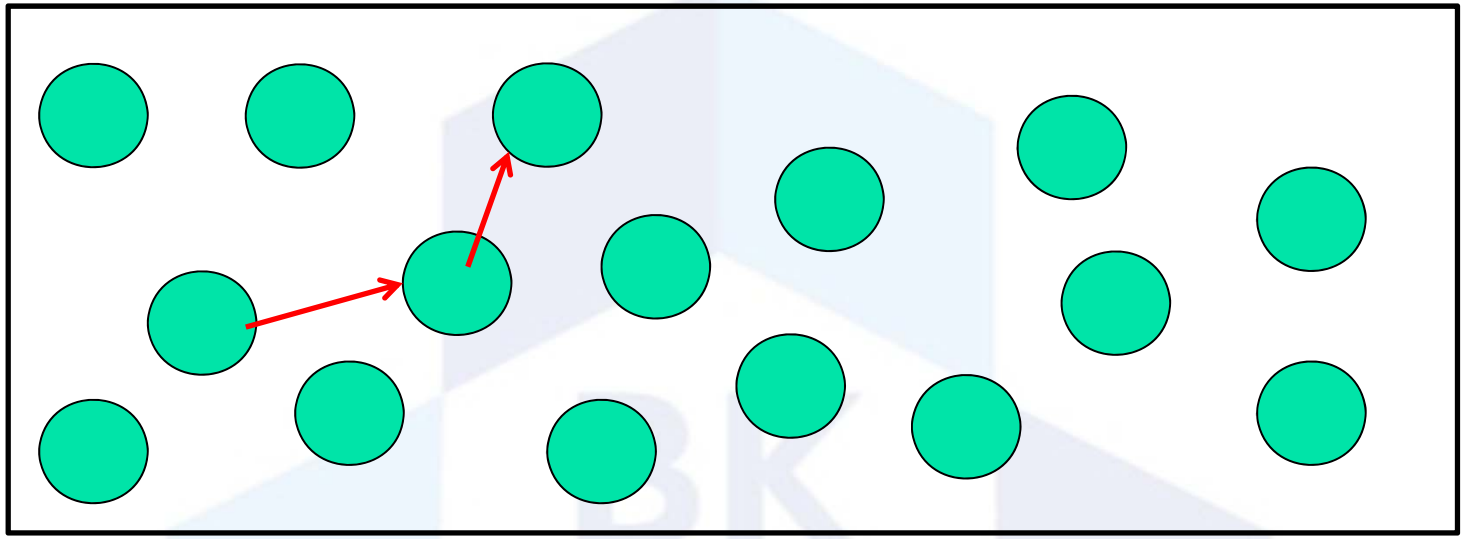
Chủng loại các phần tử cấu thành phần mềm như thế nào và mối quan hệ giữa chúng ra sao để ta có thể dễ dàng quản lý chúng theo thời gian?



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.

5.2 Đặc tả kiến trúc của hệ thống phần mềm

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.

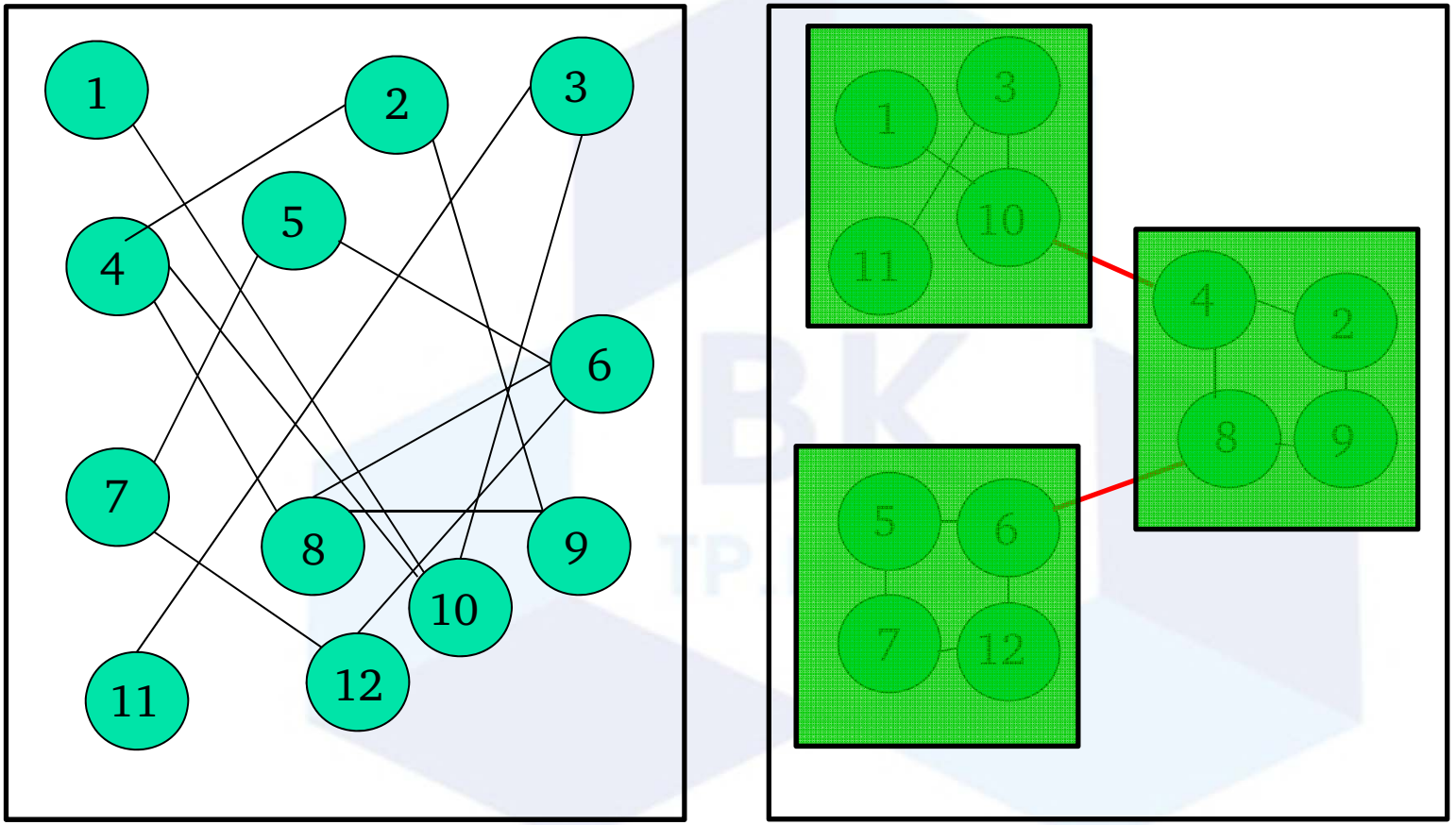


Hạn chế sự tương tác của từng đối tượng là :

- Che dấu tối đa các chi tiết hiện thực của mình, chỉ cho các phần tử khác bên ngoài thấy và dùng 1 ít các dịch vụ của mình.
- 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 cao nhất có thể có.

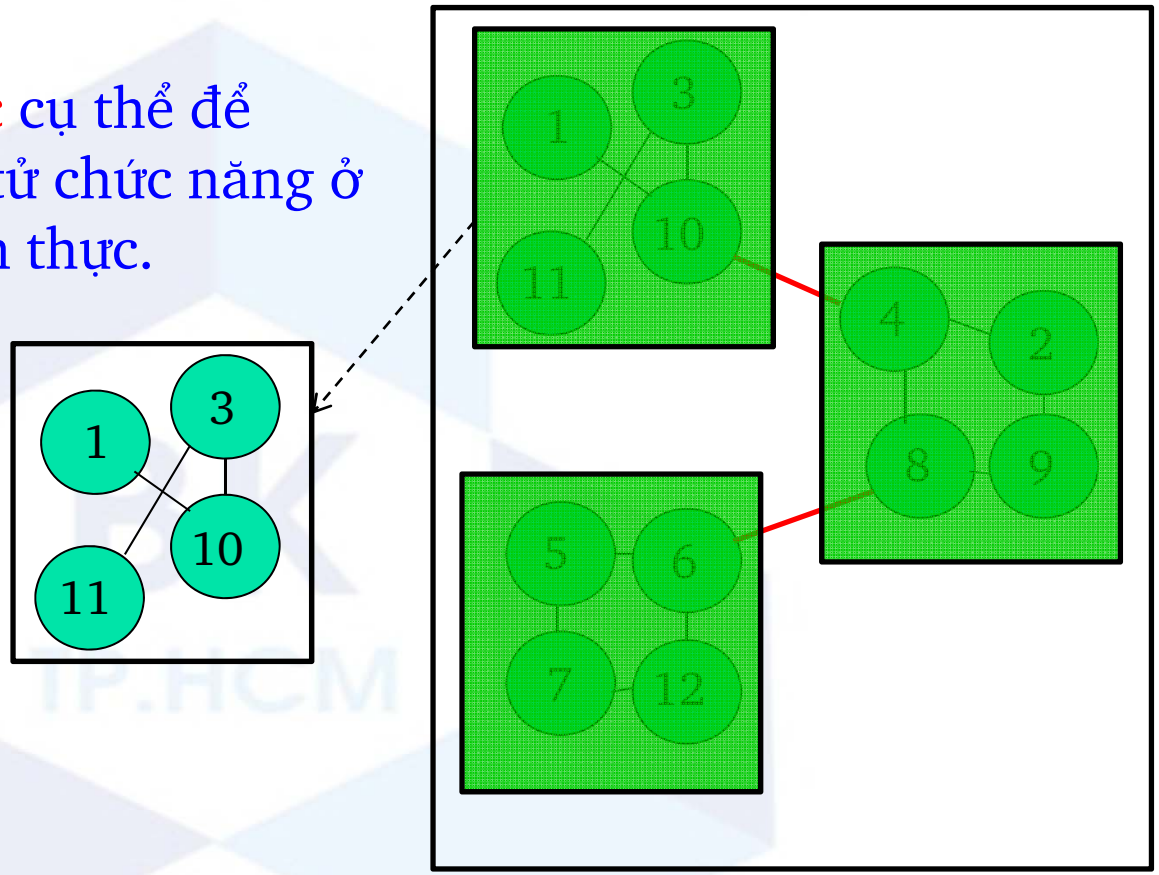
5.2 Đặc tả kiến trúc của hệ thống phần mềm

Nếu dùng cấu trúc dạng phẳng cho phần mềm lớn thì cũng rất khó quản lý vì số lượng thành phần quá lớn. Thường ta sẽ dùng cấu trúc dạng phân cấp gồm nhiều mức trừu tượng khác nhau.

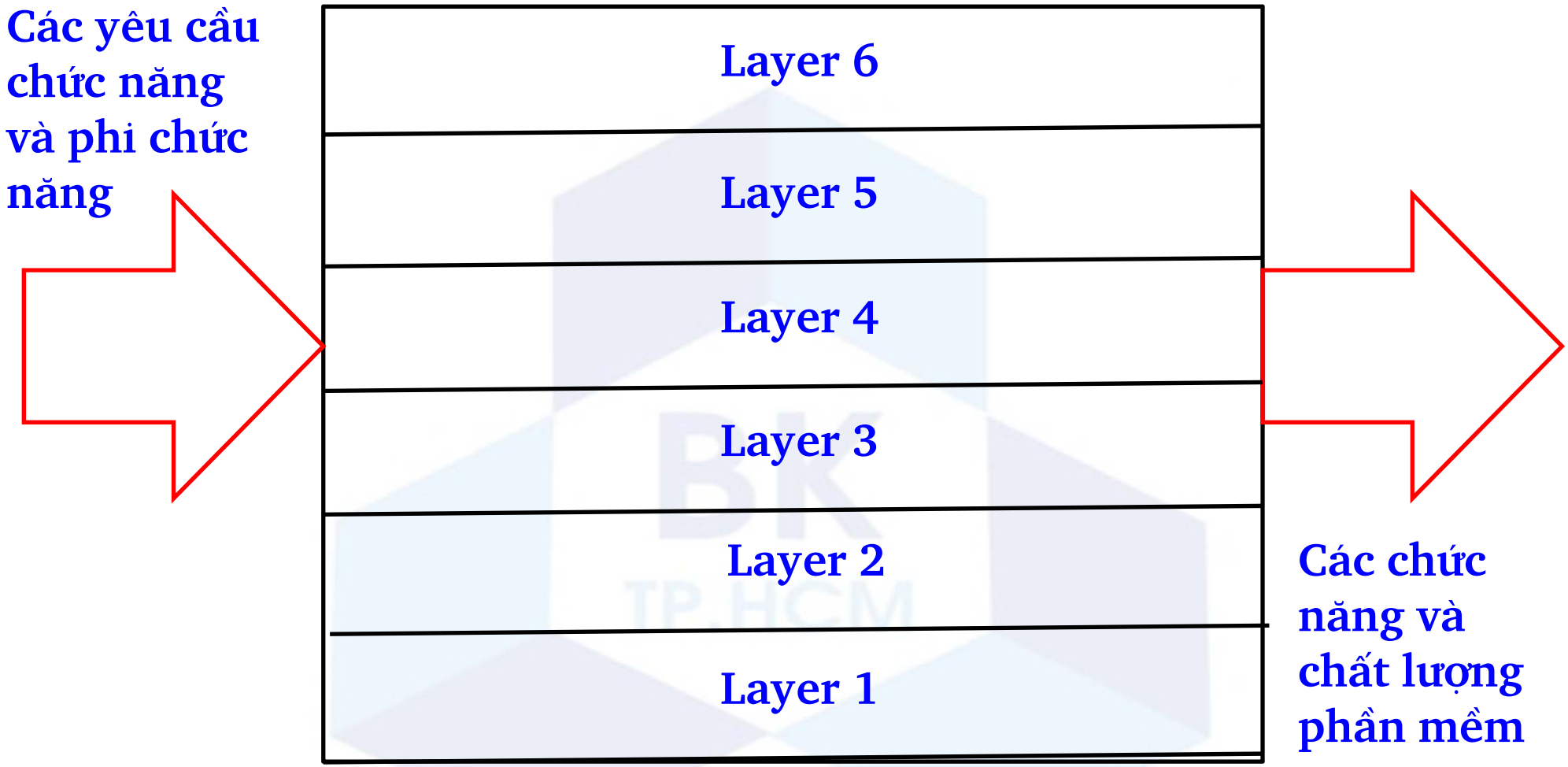


5.2 Đặc tả kiến trúc của hệ thống phần mềm

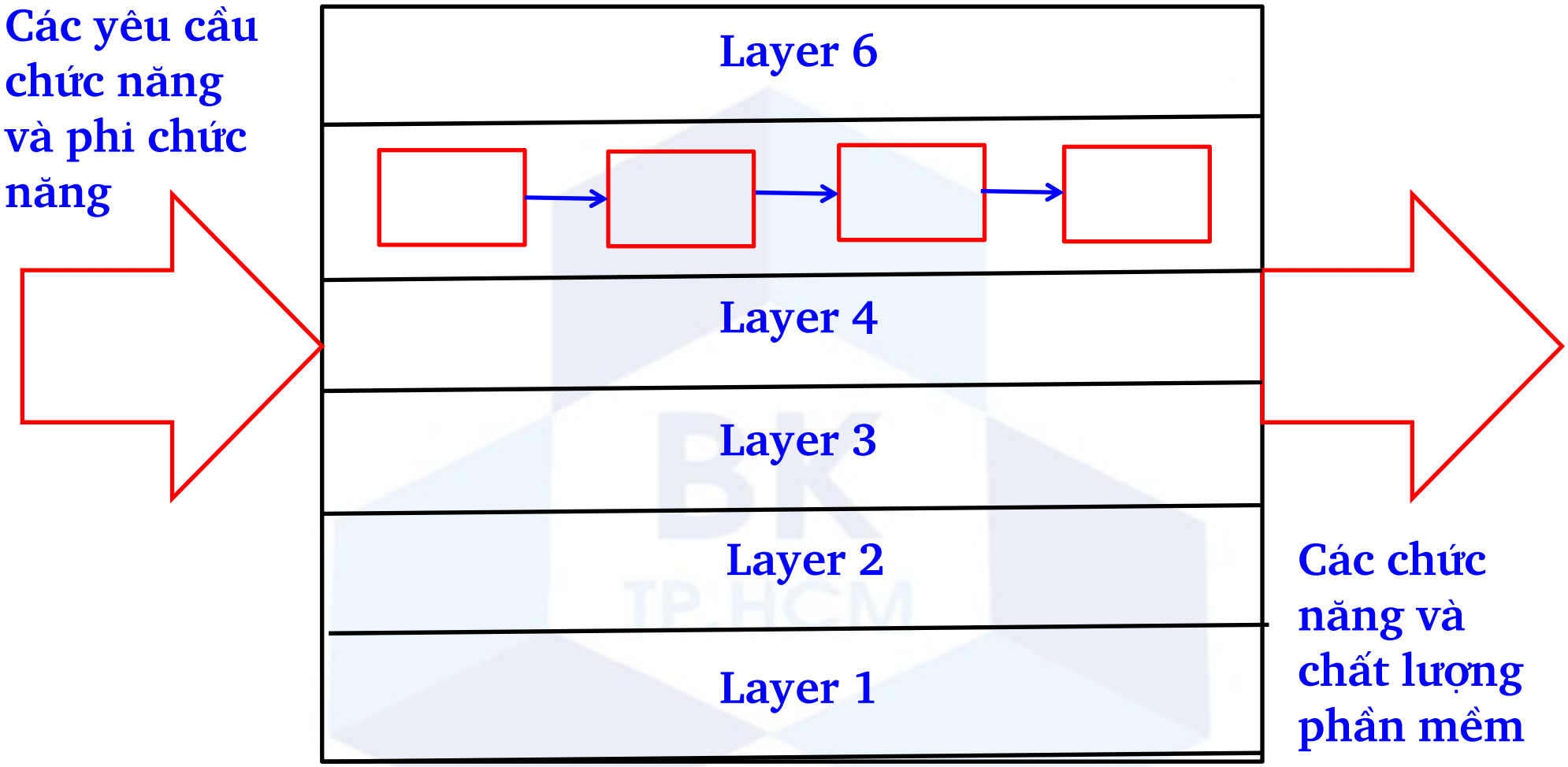
- Ta dùng thuật ngữ kiến trúc phần mềm để miêu tả cấu trúc vĩ mô của phần mềm (ở bất kỳ cấp vĩ mô nào).
- Ta dùng thuật ngữ cấu trúc cụ thể để miêu tả cấu trúc của phần tử chức năng ở cấp thấp nhất mà ta sẽ hiện thực.



5.2 Đặc tả kiến trúc của hệ thống phần mềm



5.2 Đặc tả kiến trúc của hệ thống phần mềm



5.2 Đặc tả kiến trúc của hệ thống phần mềm

Mẫu/Kiểu kiến trúc (Architecture Pattern/Style)

- Kiểu kiến trúc định nghĩa 1 họ các kiến trúc phần mềm cụ thể được giới hạn bởi :
  - từ vựng thành phần/mối nối.
  - các luật topology.
  - các ràng buộc ngữ nghĩa.

5.2 Đặc tả kiến trúc của hệ thống phần mềm

Các đặc ngữ kiến trúc phổ biến

- Các hệ thống xử lý dòng dữ liệu : lô tuần tự (Batch sequential), đường ống và lọc (Pipe and filters)
- Các hệ thống gọi-trả về : chương trình chính và thủ tục (main program & subroutines), các cấp có thứ bậc (Hierarchical layers), hệ thống hướng đối tượng (OO system).
- Các máy ảo : Trình thông dịch (Interpreters), hệ thống dựa vào luật (Rule-based system)
- Các thành phần đọc lập : các process giao tiếp nhau (Communicating processes), các hệ thống xử lý sự kiện (Event systems).
- Các hệ thống tập trung quanh dữ liệu (Repositories) : Database, Blackboard

5.3 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.

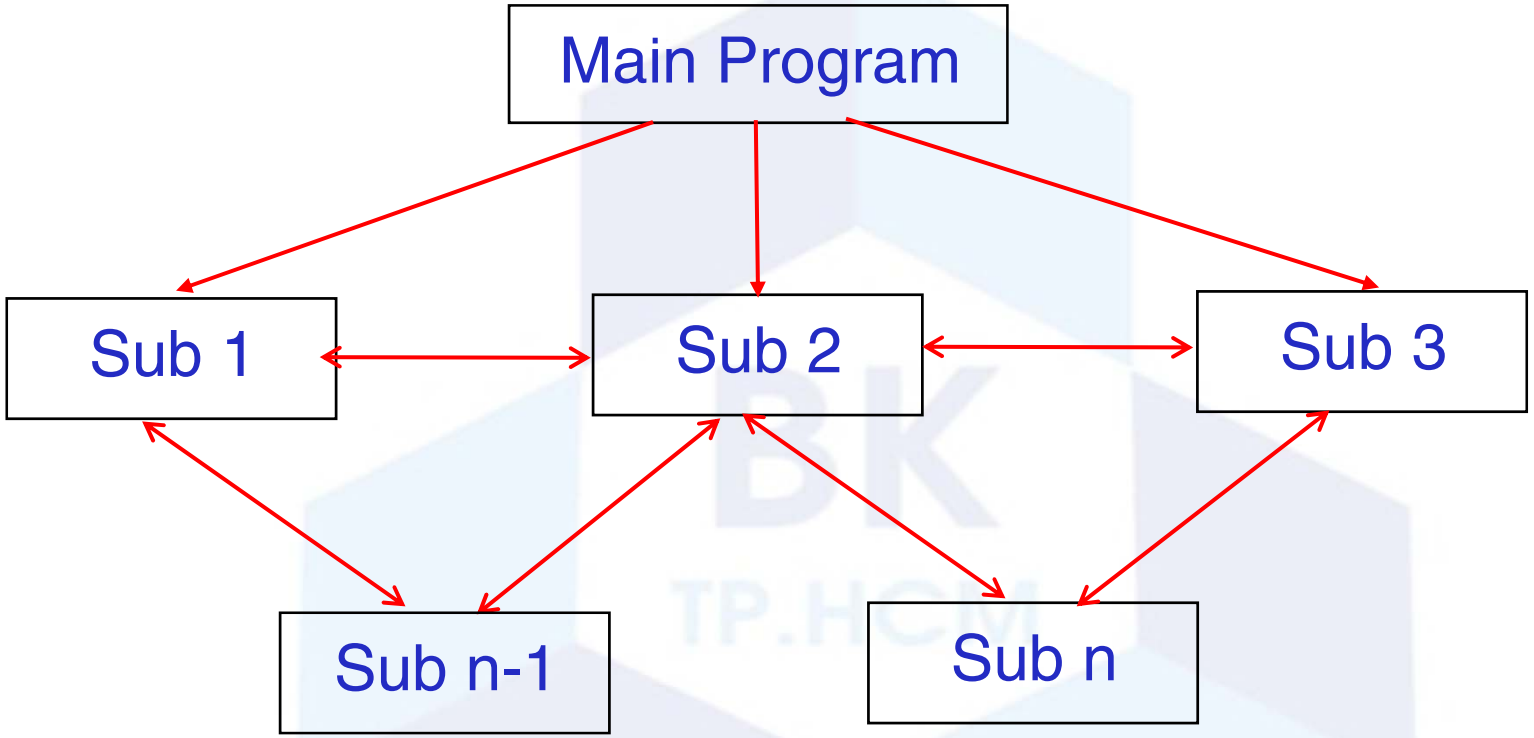
5.3 Các mẫu kiến trúc phổ dụng

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.

5.3 Các mẫu kiến trúc phổ dụng

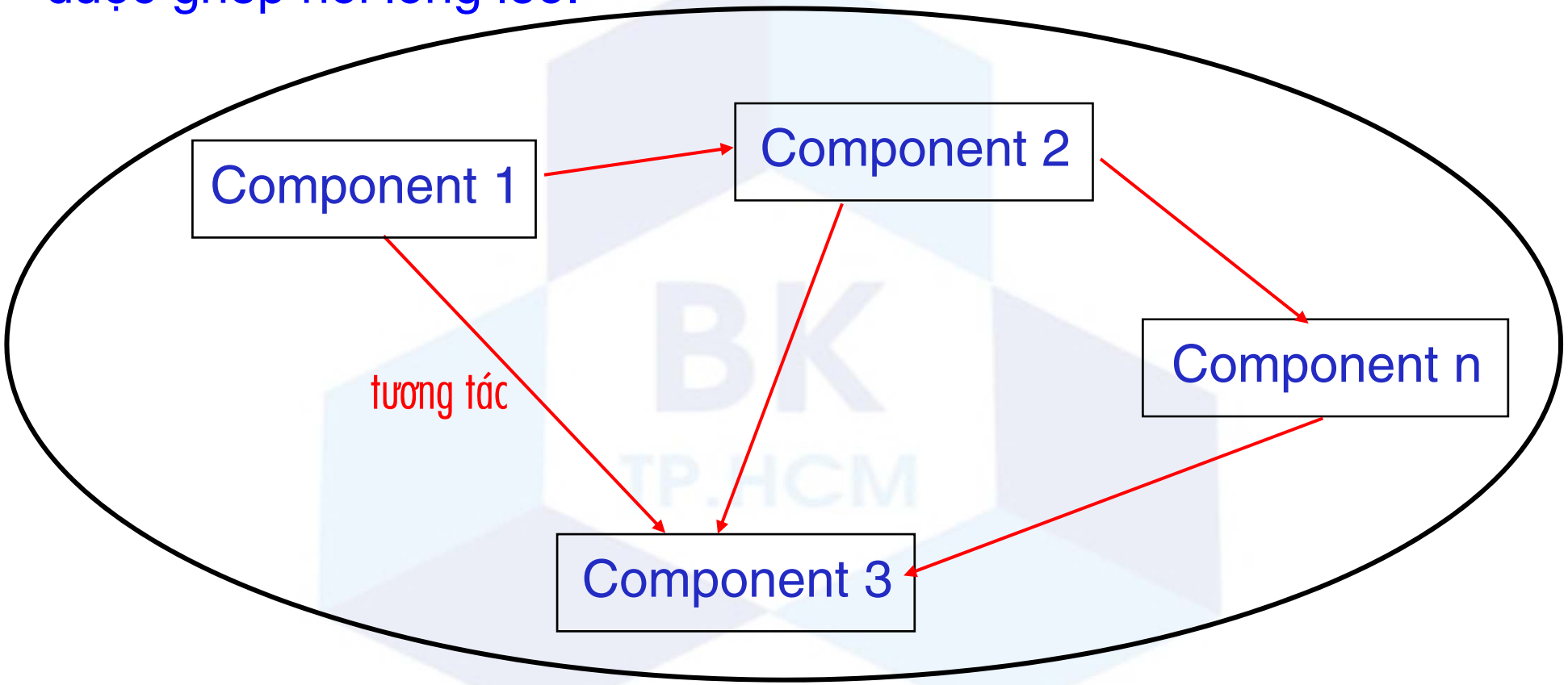
Kiến trúc Chương trình chính và thủ tục (MainProgram/Subroutine Architecture)



5.3 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.



5.3 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

5.3 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

5.3 Các mẫu kiến trúc phổ dụng

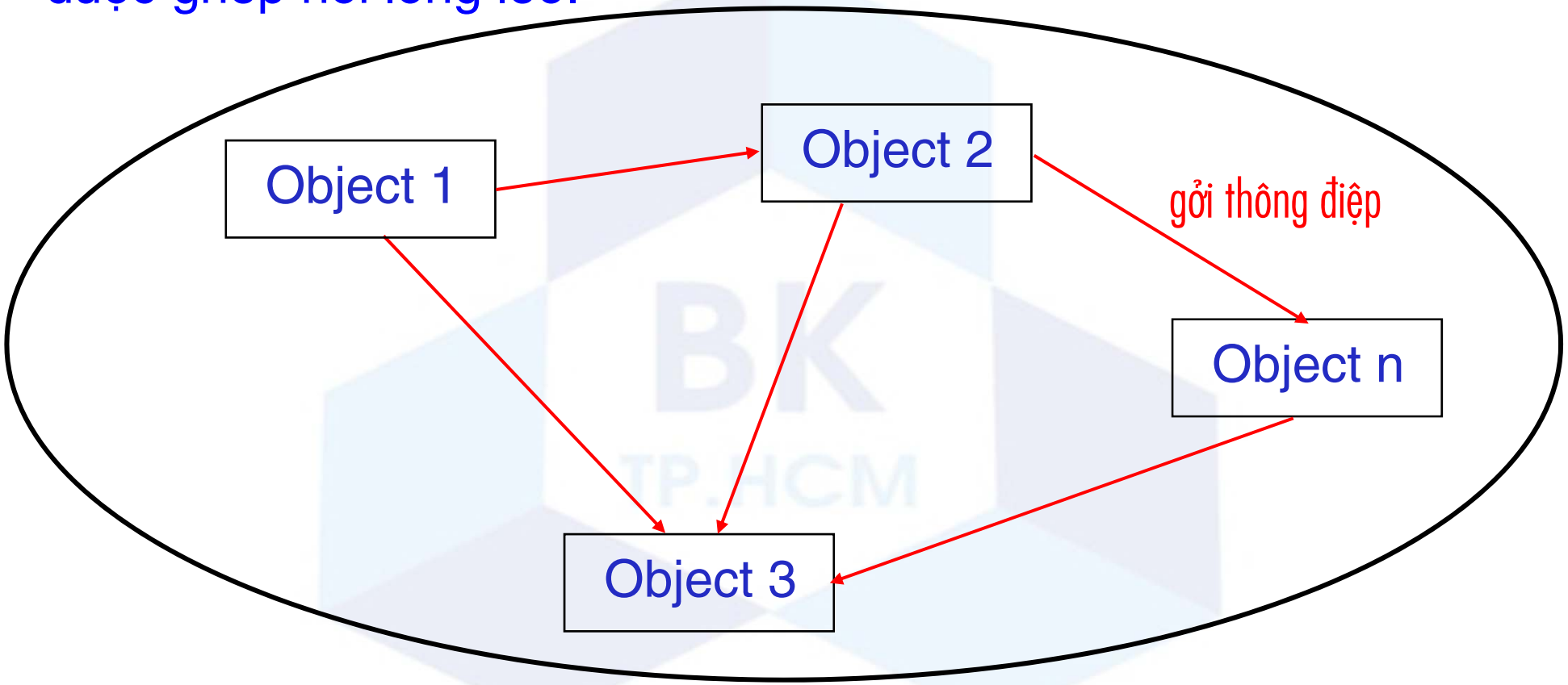
Kiến trúc các thành phần (Components based Architecture)

- **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.

5.3 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.



5.3 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

5.3 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

5.3 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

5.3 Các mẫu kiến trúc phổ dụng

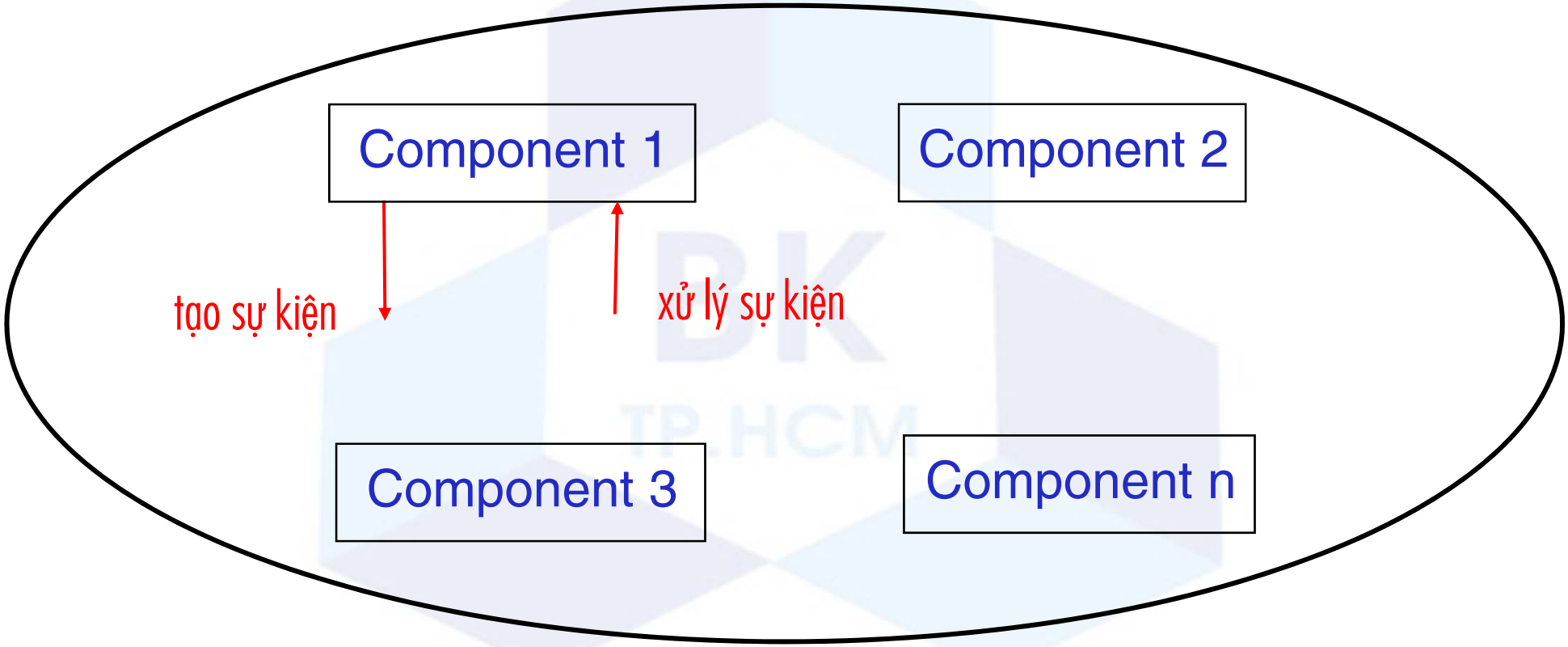
Kiến trúc hướng đối tượng (Objects based Architecture)

- **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.

5.3 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.



5.3 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...

5.3 Các mẫu kiến trúc phổ dụng

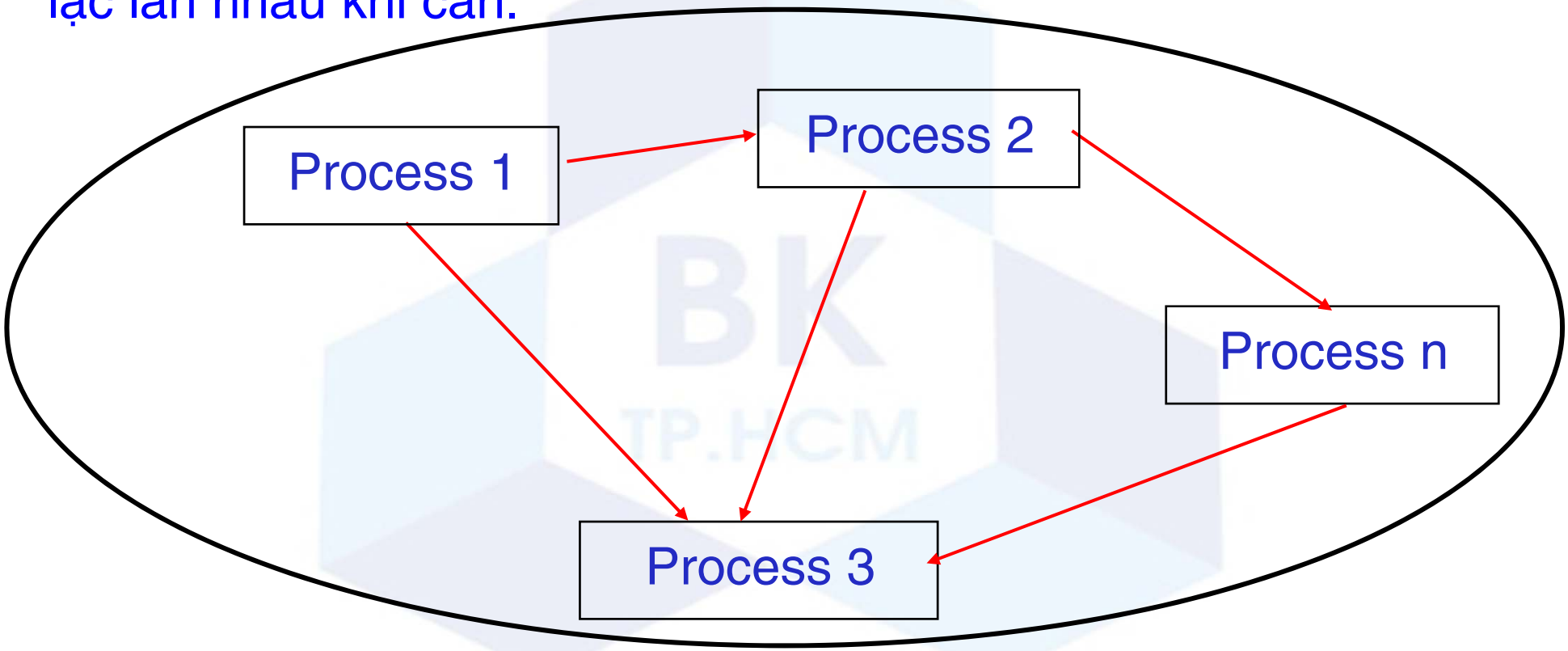
Kiến trúc dựa trên sự kiện (Event-based Architecture)

- **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...

5.3 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.



5.3 Các mẫu kiến trúc phổ dụng

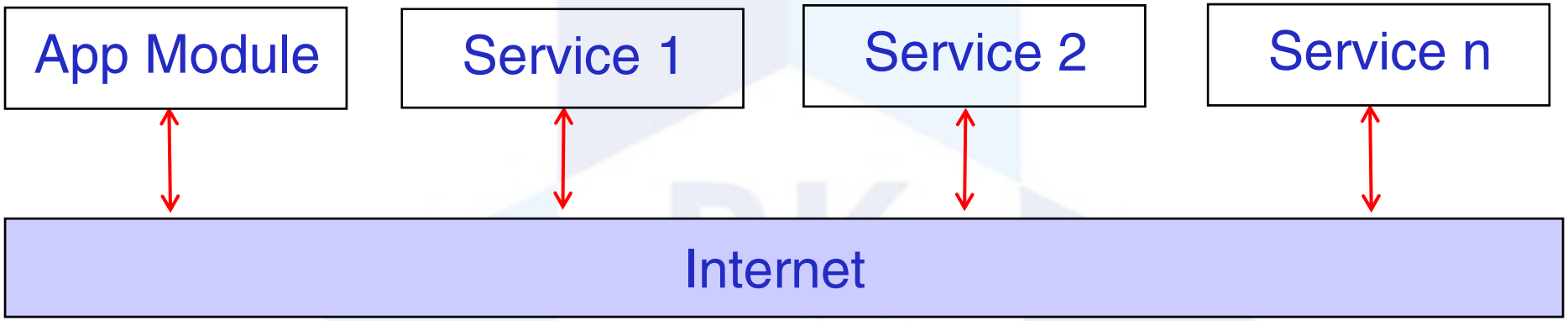
Kiến trúc các process liên lạc nhau (Communication process Architecture)

- **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.

5.3 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.



5.3 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.

5.3 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

5.3 Các mẫu kiến trúc phổ dụng

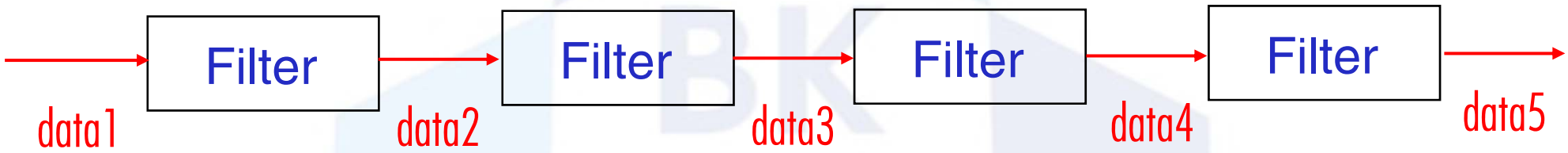
Kiến trúc hướng dịch vụ (Service-Oriented Architecture)

- **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.

5.3 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.



5.3 Các mẫu kiến trúc phổ dụng

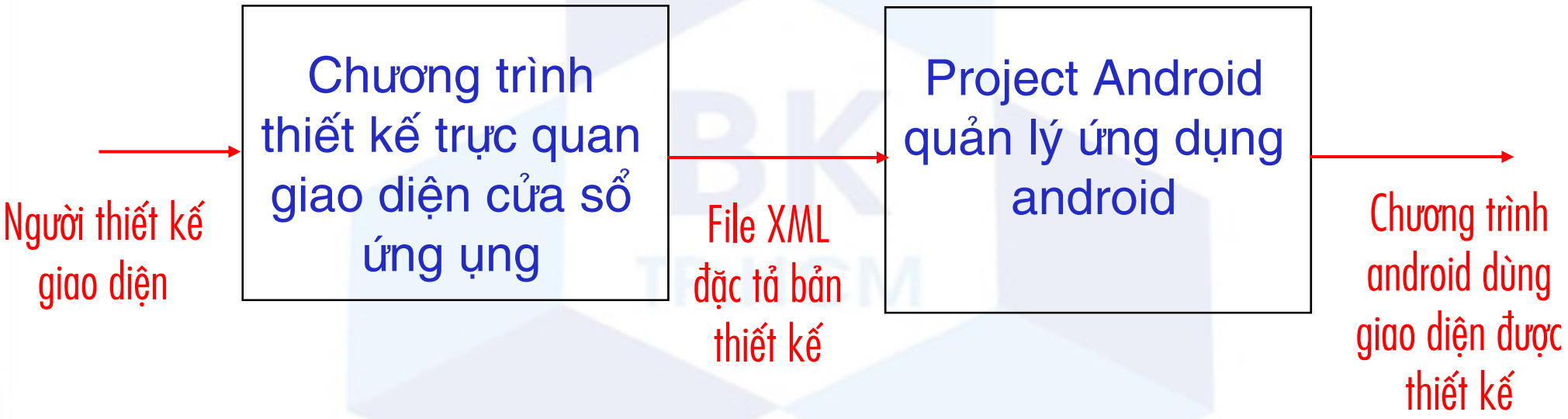
Kiến trúc lô tuần tự (Batch Sequential)

- **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.

5.3 Các mẫu kiến trúc phổ dụng

Kiến trúc lô tuần tự (Batch Sequential)

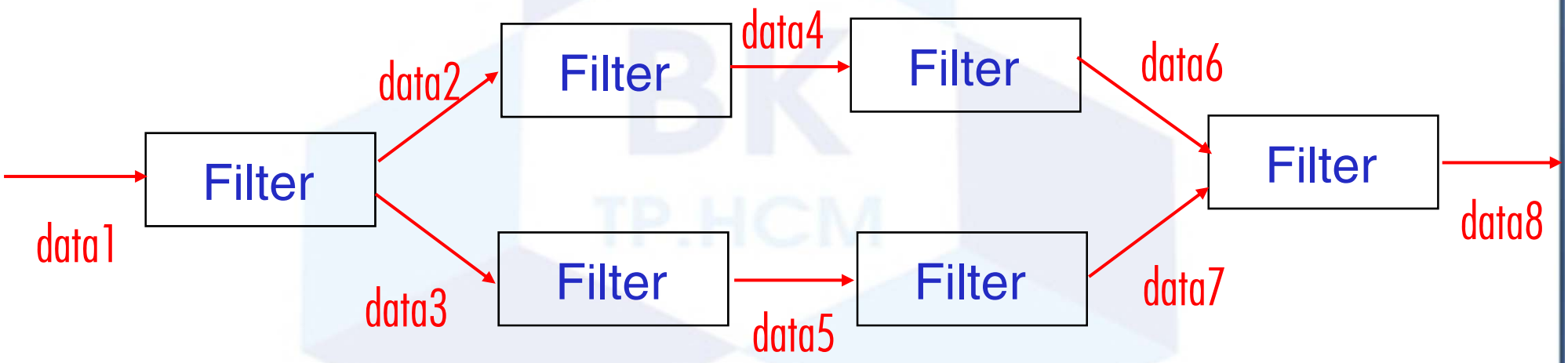
- **Thí dụ** : Thiết kế trực quan cửa sổ giao diện và dùng nó trong phần mềm android.



5.3 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.



5.3 Các mẫu kiến trúc phổ dụng

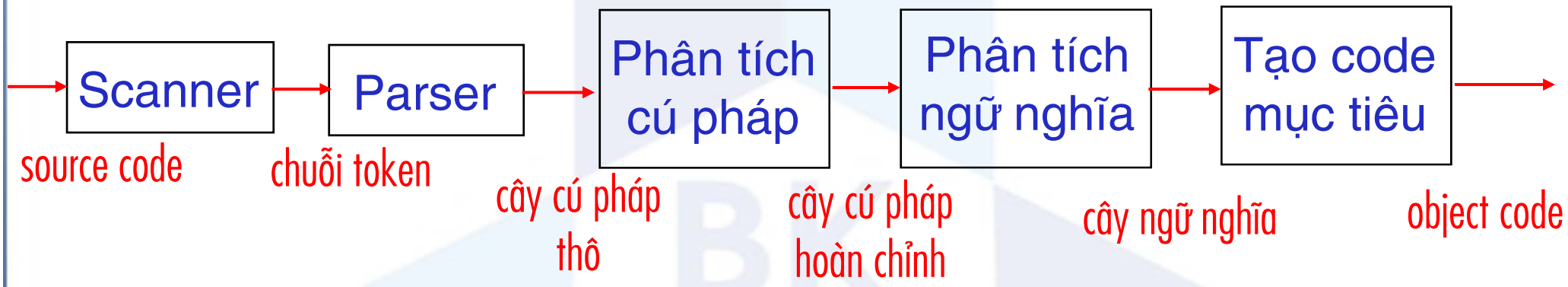
Kiến trúc đường ống và lọc (Pipe and filter Architecture)

- **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.

5.3 Các mẫu kiến trúc phổ dụng

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

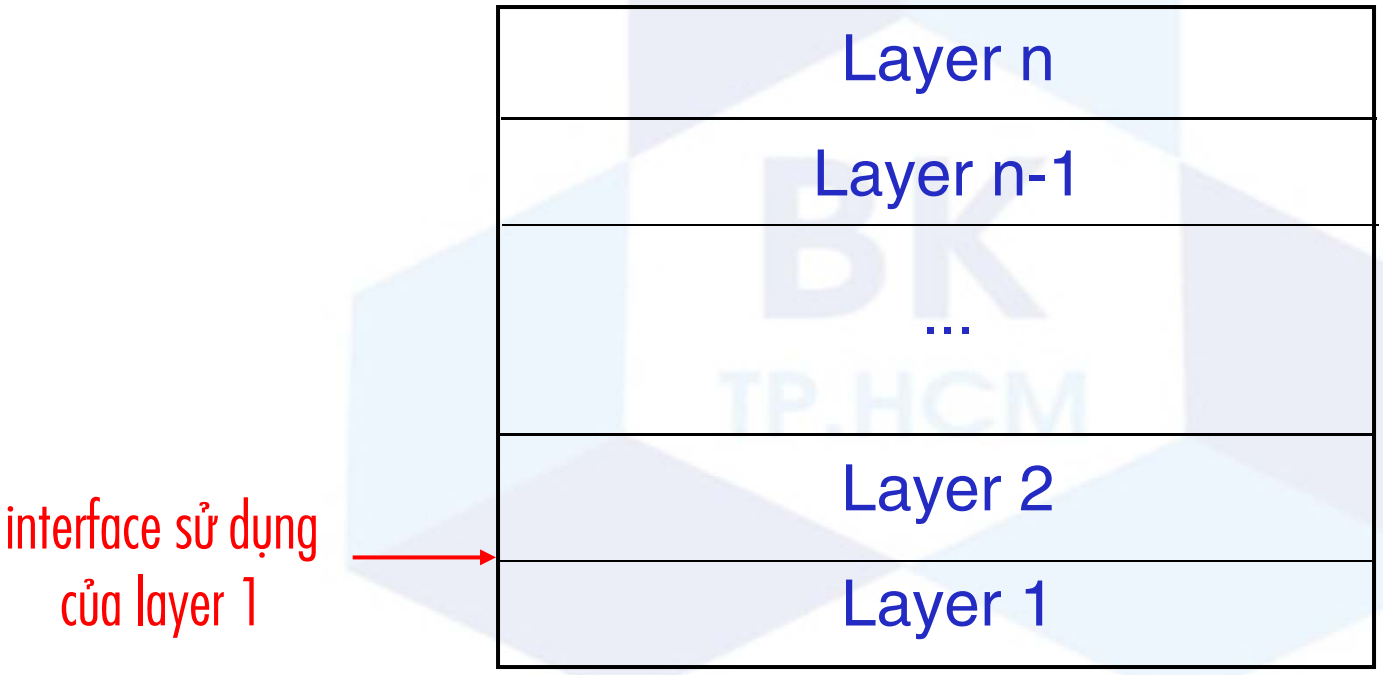
- **Thí dụ** : Chương trình dịch ngôn ngữ



5.3 Các mẫu kiến trúc phổ dụng

Kiến trúc nhiều cấp (Layered architecture)

- **Đặc tả** : Hệ thống gồm nhiều cấp chức năng dạng chồng lên nhau, mỗi layer có chức năng cụ thể, rõ ràng và cung cấp các dịch vụ cho layer ngay trên mình. Layer cấp thấp nhất chứa các dịch vụ cơ bản nhất và được dùng cho toàn hệ thống.



5.3 Các mẫu kiến trúc phổ dụng

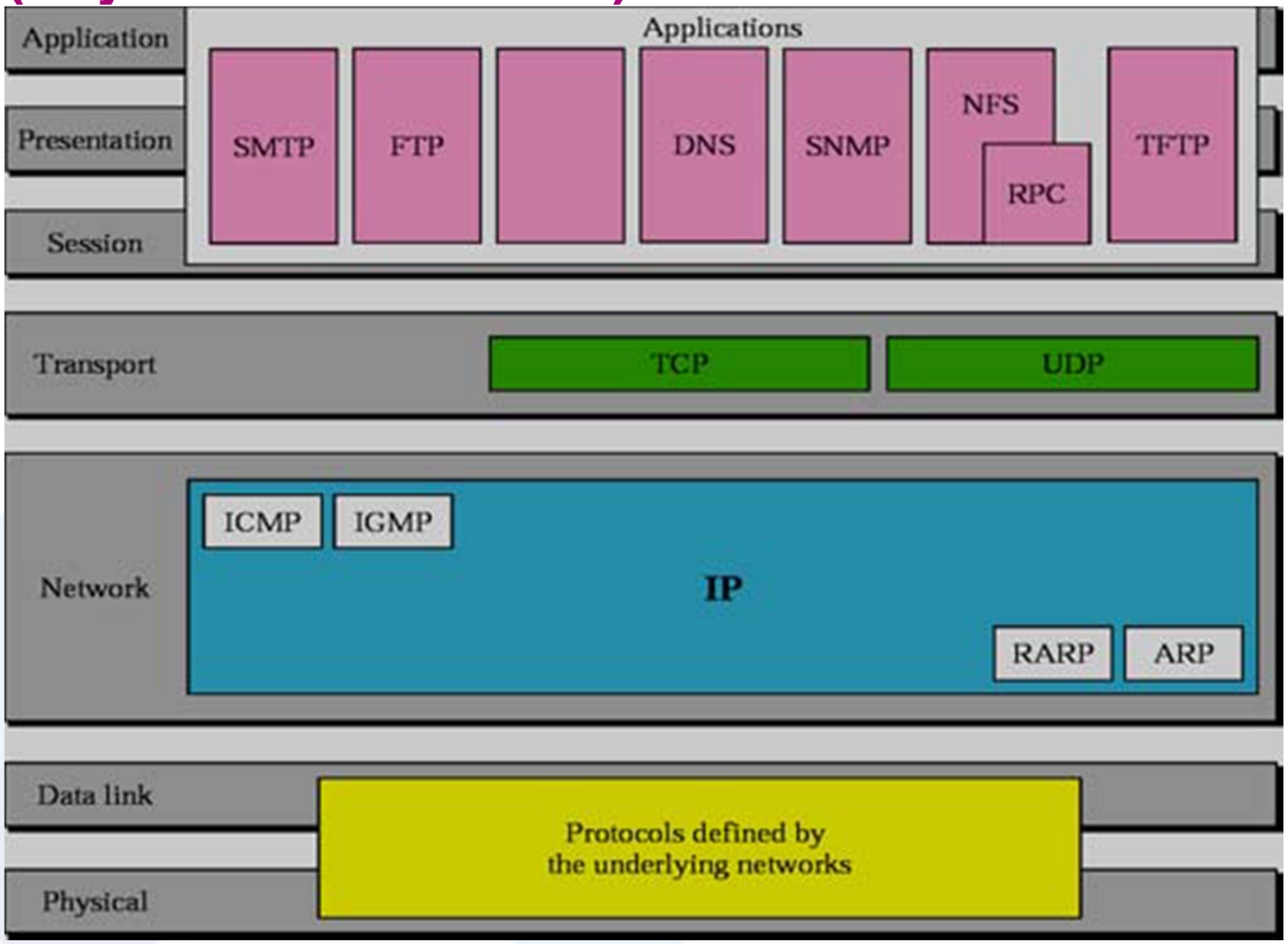
Kiến trúc nhiều cấp (Layered architecture)

- **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.

5.3 Các mẫu kiến trúc phổ dụng

Kiến trúc nhiều cấp (Layered architecture)

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



5.3 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.

5.3 Các mẫu kiến trúc phổ dụng

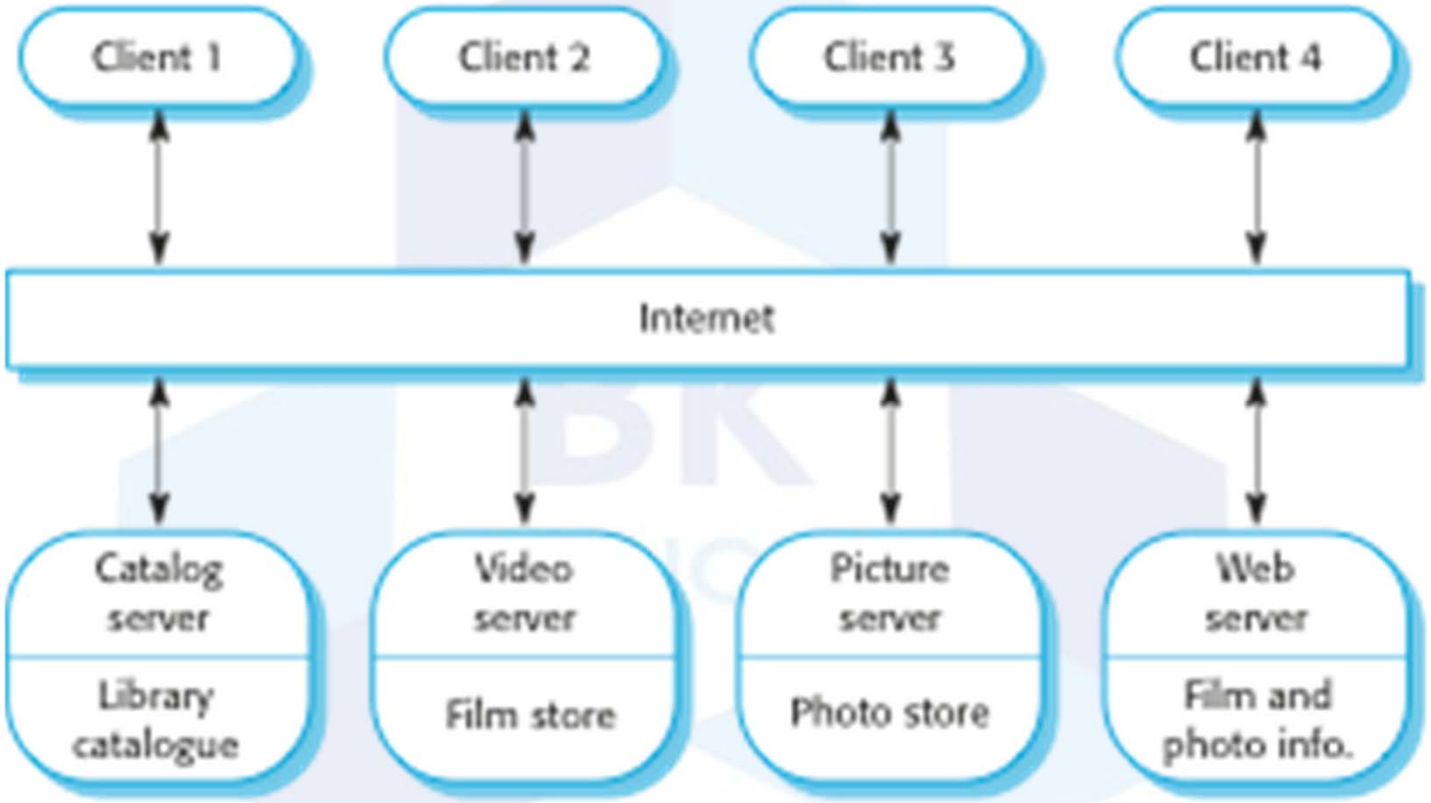
Kiến trúc client-server (client-server Architecture)

- **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.

5.3 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



5.3 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.

5.3 Các mẫu kiến trúc phổ dụng

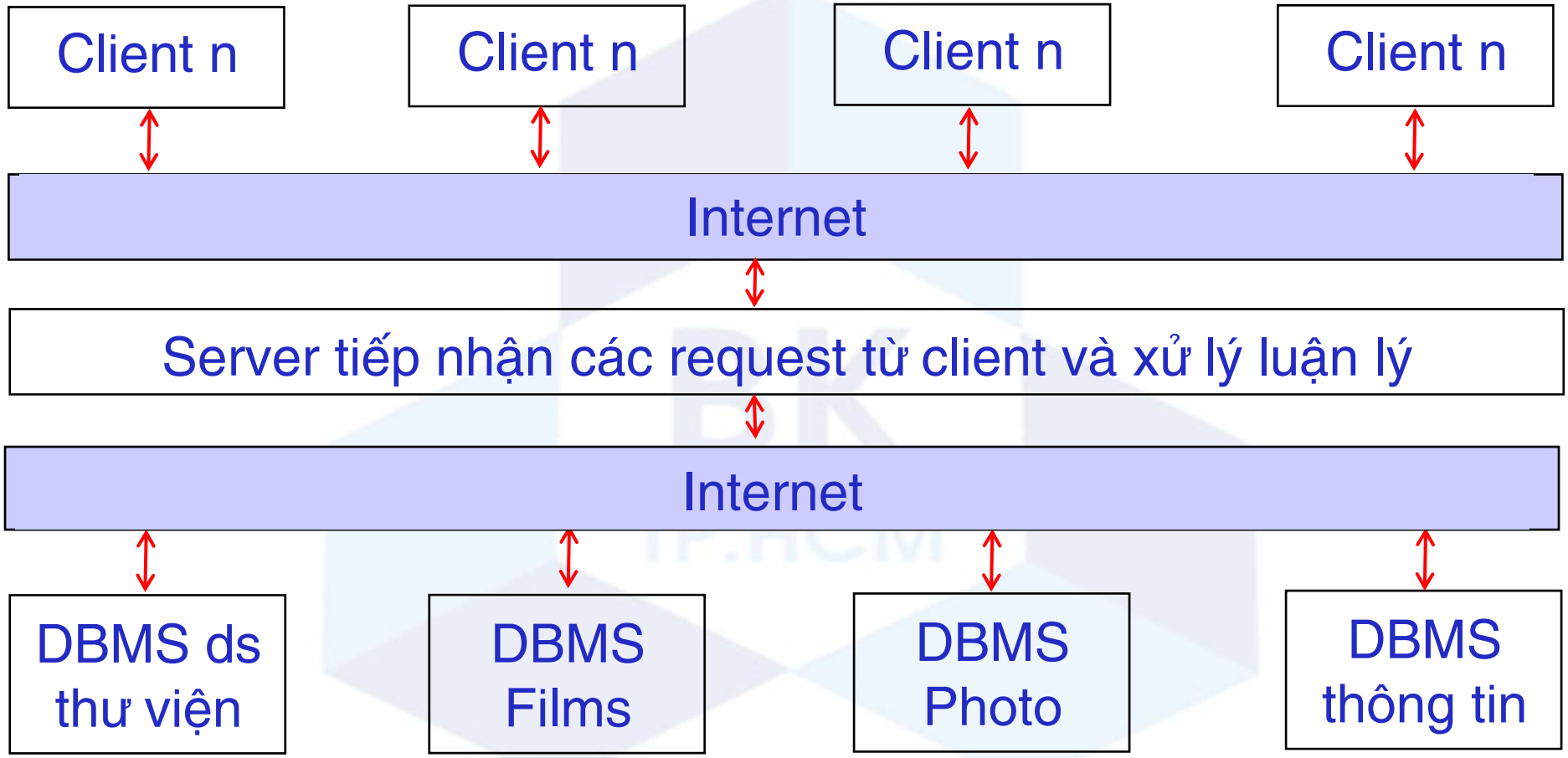
Kiến trúc 3 đối tác (3-tiers Architecture)

- **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.

5.3 Các mẫu kiến trúc phổ dụng

Kiến trúc 3 đối tác (3-tiers Architecture)

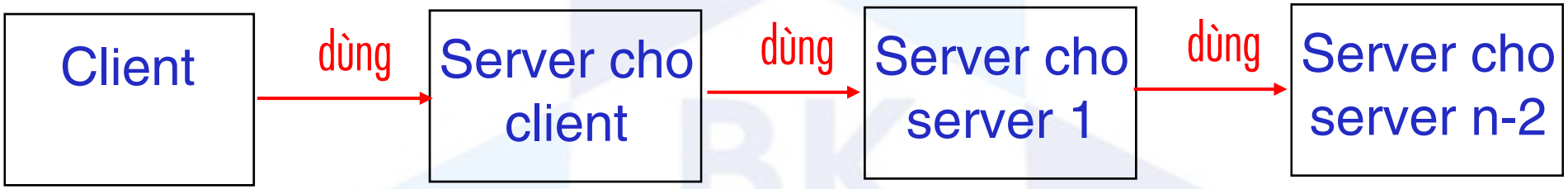
- **Thí dụ** : Hệ thống quản lý phim ảnh dùng mô hình 3-tiers



5.3 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,...



5.3 Các mẫu kiến trúc phổ dụng

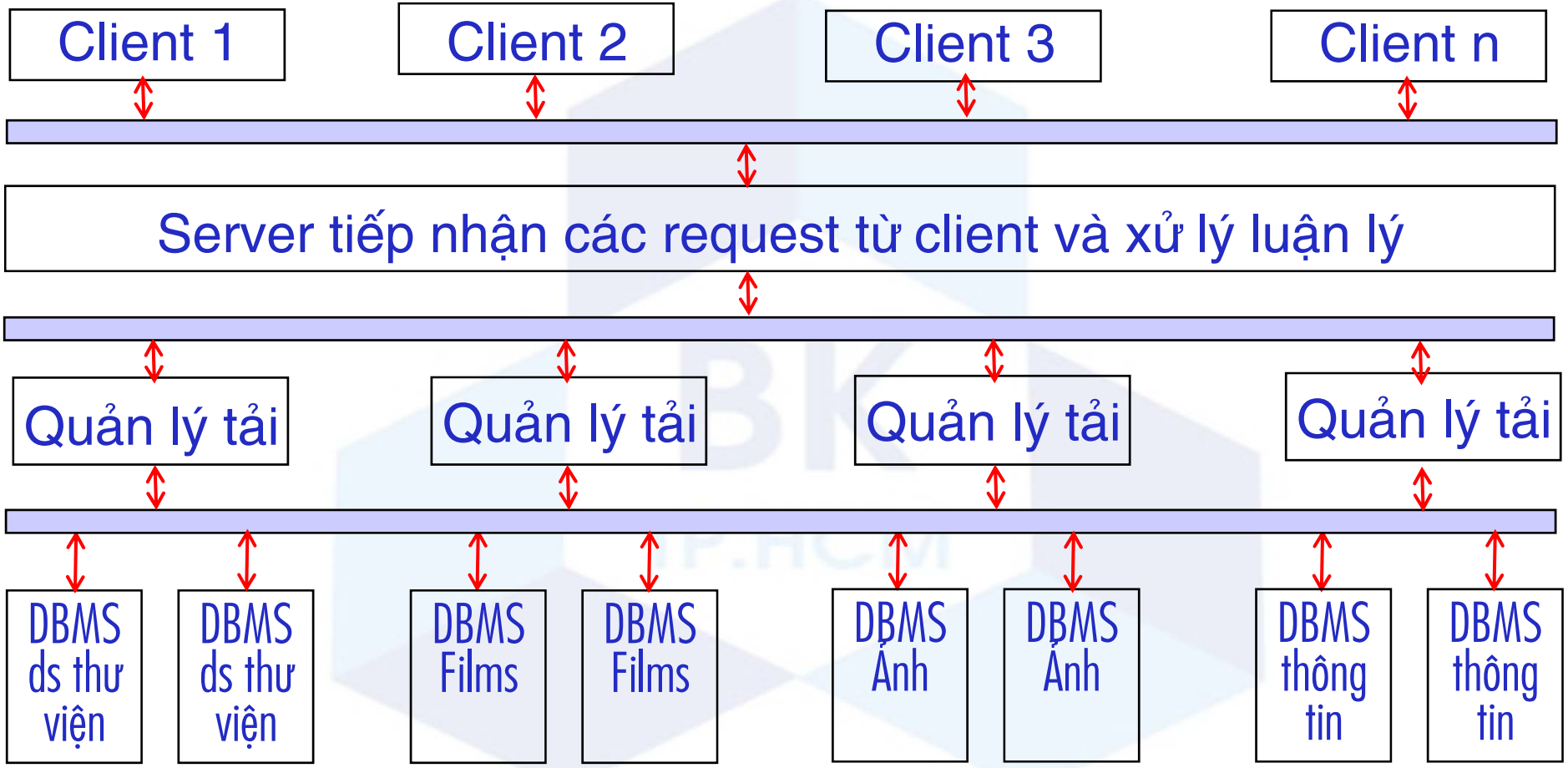
Kiến trúc n đối tác (n-tiers Architecture)

- **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.

5.3 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



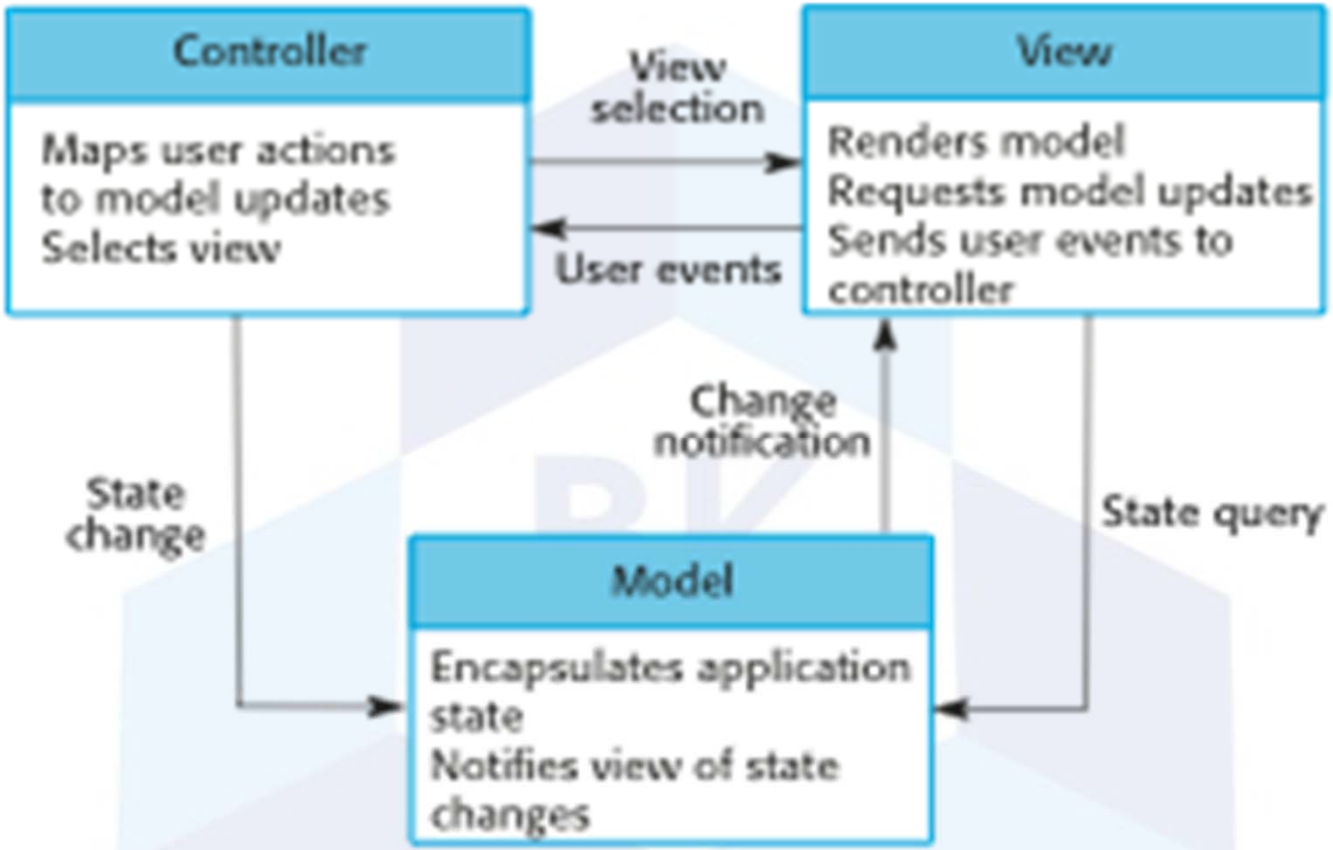
5.3 Các mẫu kiến trúc phổ dụng

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.

5.3 Các mẫu kiến trúc phổ dụng

Kiến trúc MVC (Model-View-Controller)



5.3 Các mẫu kiến trúc phổ dụng

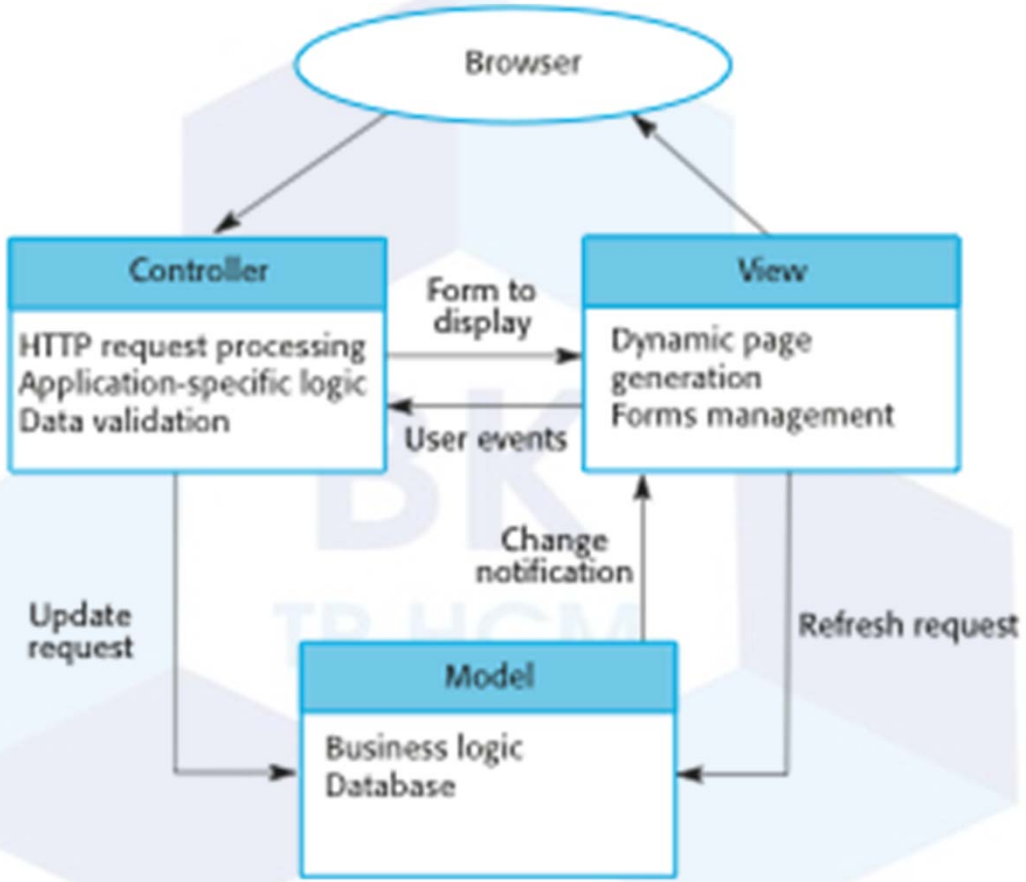
Kiến trúc MVC (Model-View-Controller)

- **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.

5.3 Các mẫu kiến trúc phổ dụng

Kiến trúc MVC (Model-View-Controller)

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



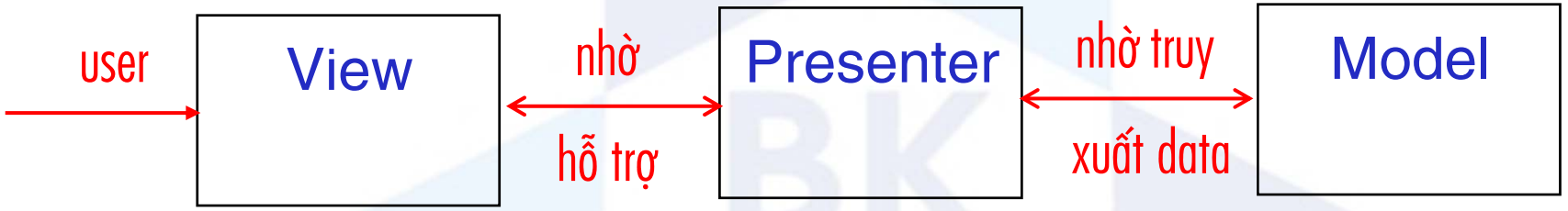
5.3 Các mẫu kiến trúc phổ dụng

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 vụ, nhờ Model truy xuất dữ liệu khi cần

5.3 Các mẫu kiến trúc phổ dụng

Kiến trúc MVP (Model-View-Presenter)



5.3 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.

5.3 Các mẫu kiến trúc phổ dụng

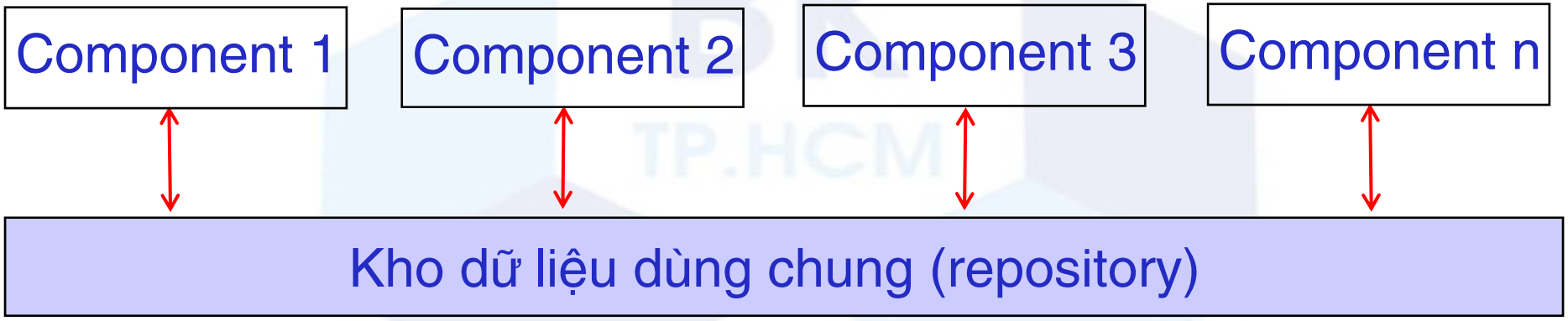
Kiến trúc MVP (Model-View-Presenter)

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

5.3 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.



5.3 Các mẫu kiến trúc phổ dụng

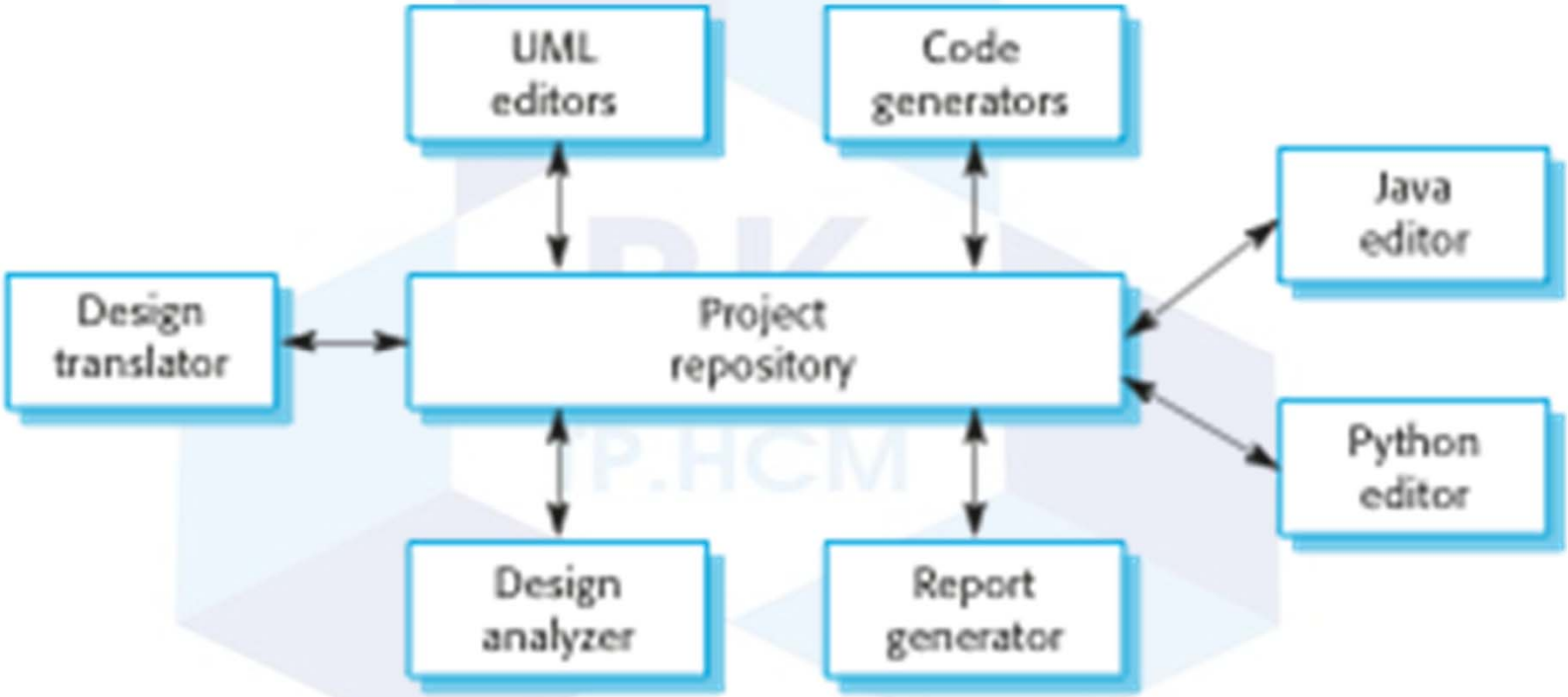
Kiến trúc kho (Repository Architecture)

- **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.

5.3 Các mẫu kiến trúc phổ dụng

Kiến trúc kho (Repository Architecture)

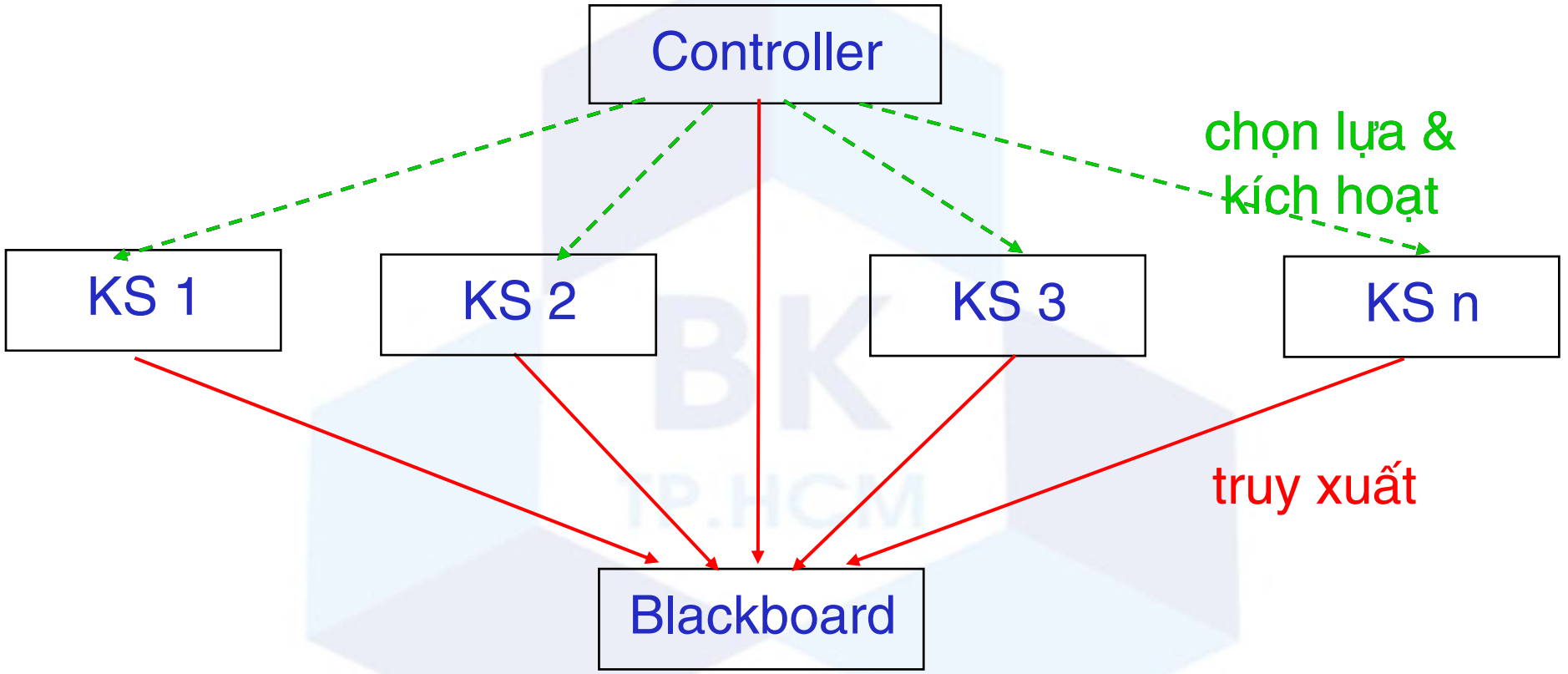
- **Thí dụ** : Môi trường IDE gồm nhiều thành phần dùng chung kho thông tin, mỗi tool tạo thông tin và để trong kho để các tool khác dùng.



5.3 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 :



5.3 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** : mỗi KS là tập các phần tử khác nhau để thực hiện 1 chức năng xác định. Mỗi phần tử 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).

5.3 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** : ?

5.4 Kết chương

- Chương này đã giới thiệu một số mẫu kiến trúc phần mềm phổ dụng, tính chất, ưu và khuyết điểm của từng mẫu.